



You don't have to be an engineer to be a racing driver, but you do have to have **mechanical sympathy**

Sir Jackie Stewart - three-time F1 world champion



Concept applied to software by Martin Thompson



- Concept applied to software by Martin Thompson
- As developers, we don't need to be hardware engineers



- Concept applied to software by Martin Thompson
- As developers, we don't need to be hardware engineers
- Yet, having an understanding of how does a machine work can make us a better developer (algorithms, data structures)



- Concept applied to software by Martin Thompson
- As developers, we **don't need** to be hardware engineers
- Yet, having an understanding of how does a machine work can make us a **better developer** (algorithms, data structures)

Today: How to be a better Go developer by understanding how CPUs are working









Teiva Harsanyi **y**teivah

Software Engineer - Beat

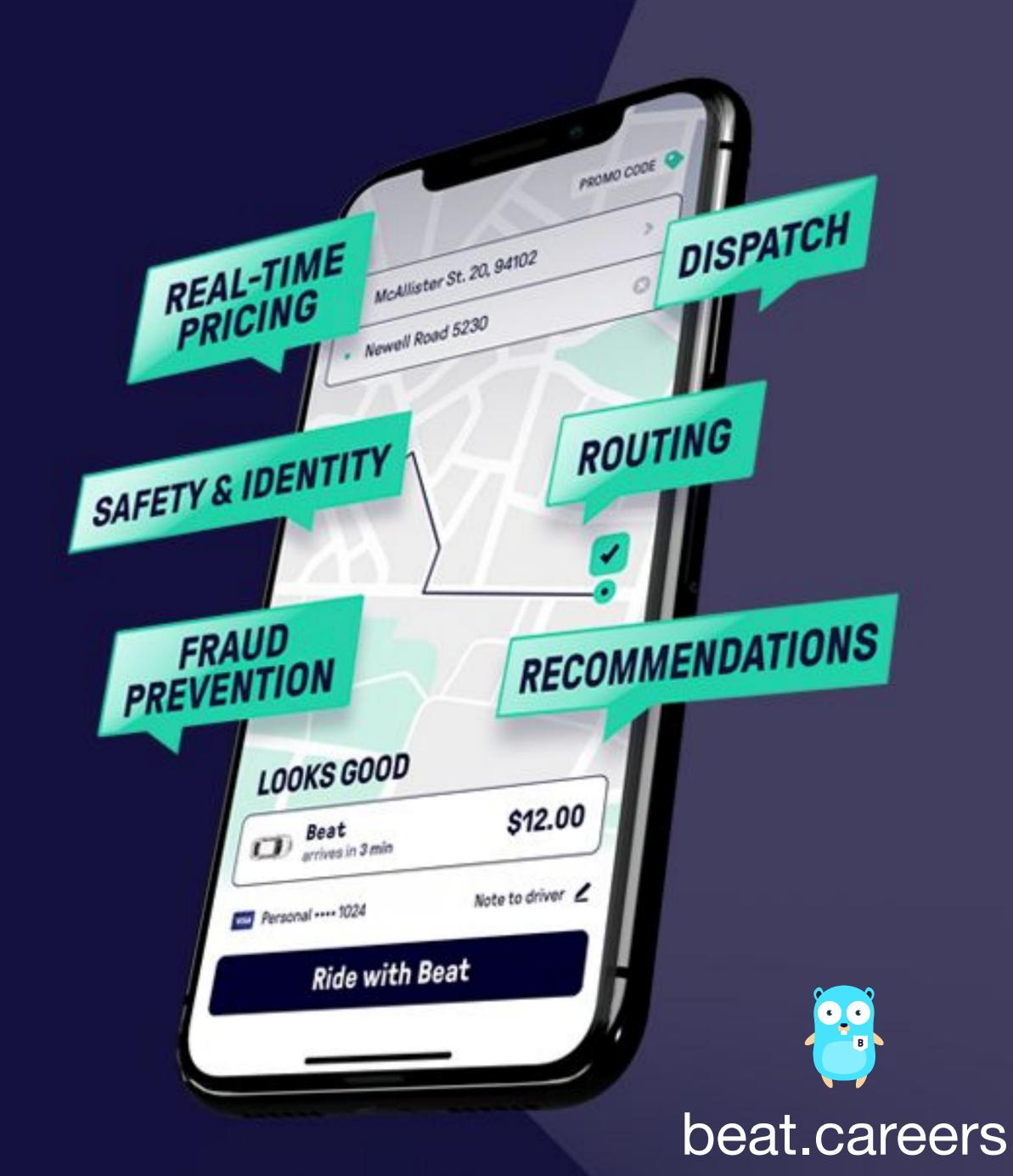


HIRING SOFWARE

ON CUTTING-EDGE TECHNOLOGIES

DISCOVER YOUR NEXT CHALLENGE IN AMSTERDAM, ATHENS OR REMOTE

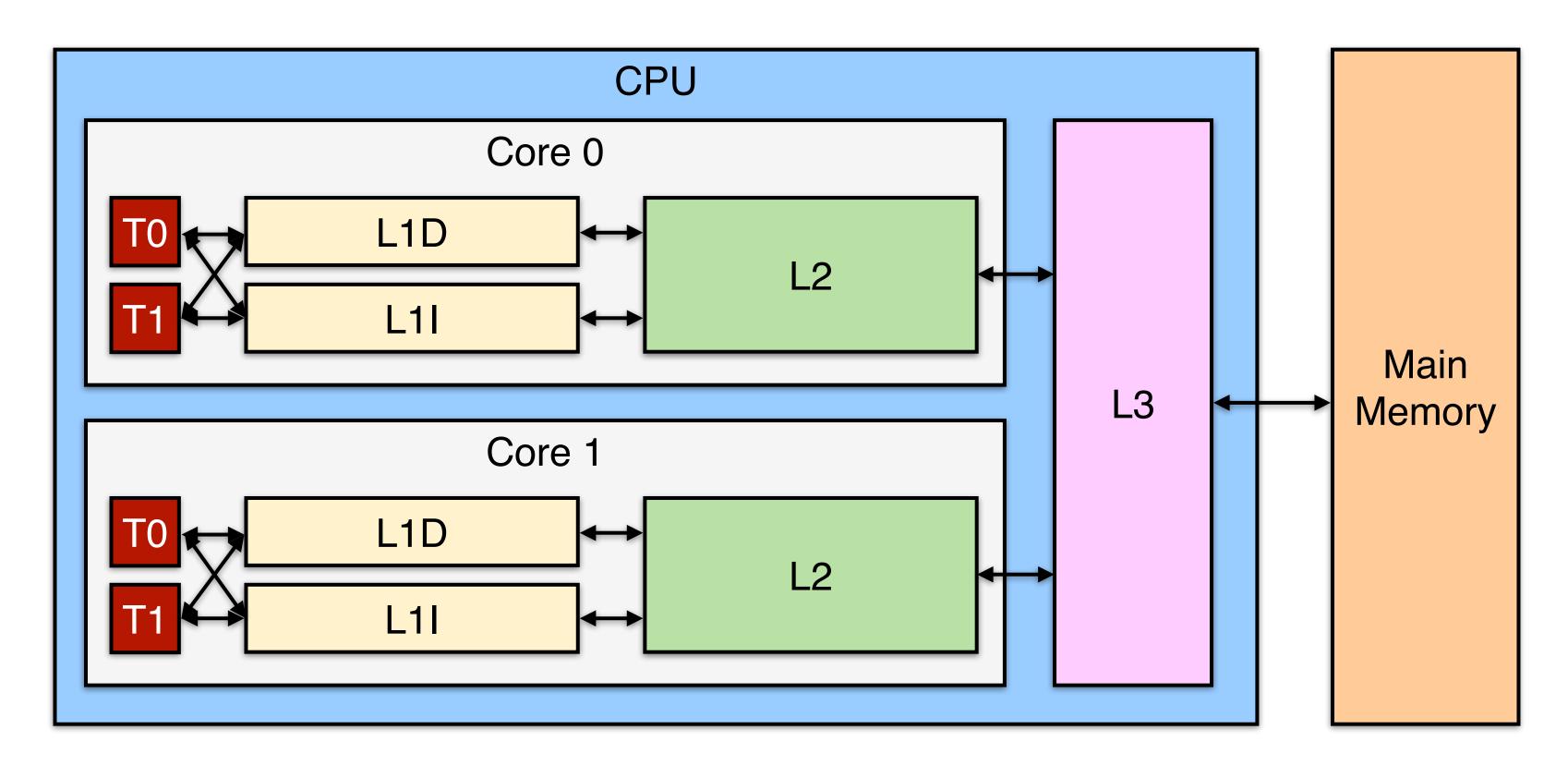




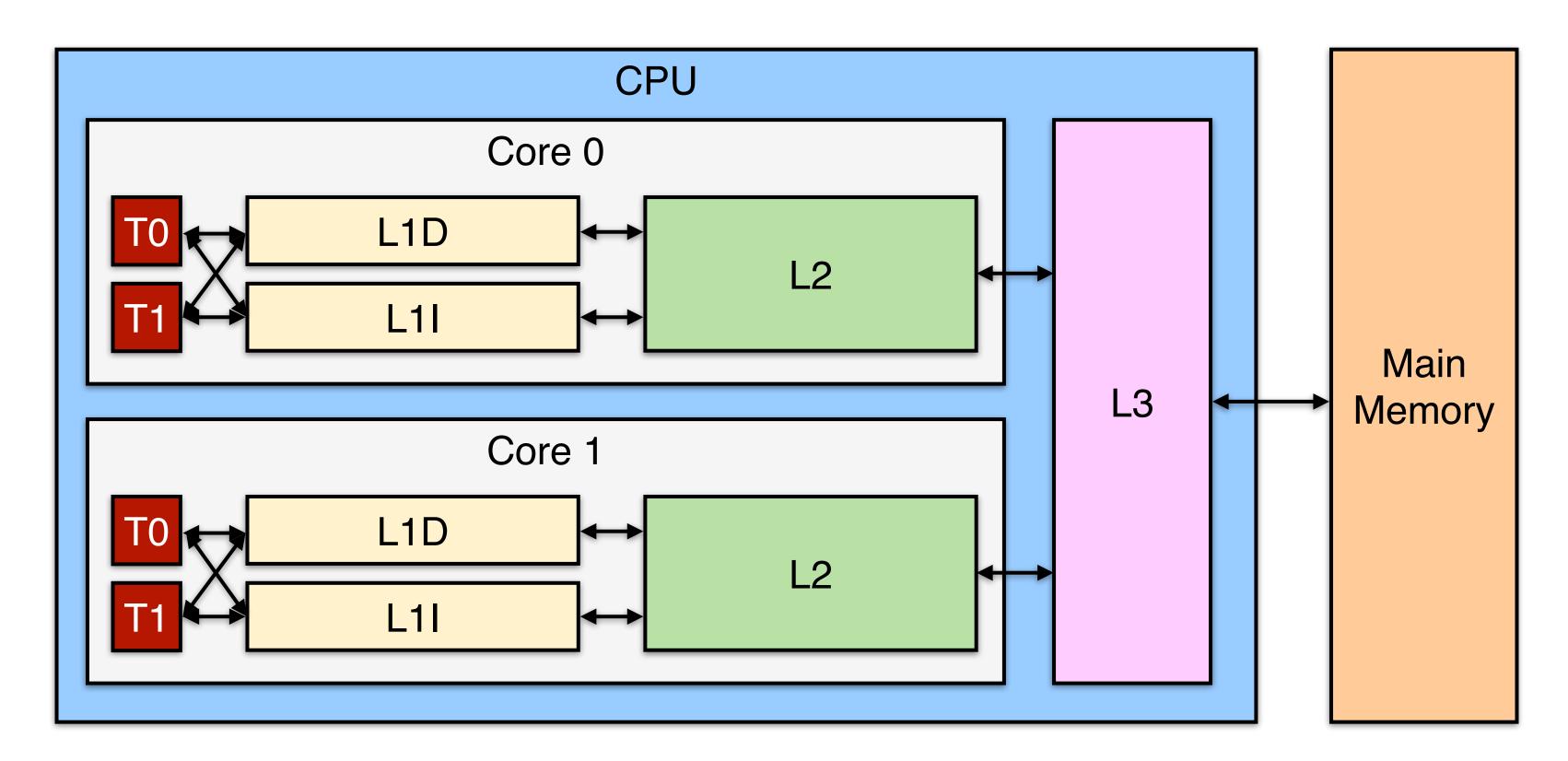


CPU Architecture Locality of Reference **Data-Oriented Design** Caching Pitfall Concurrency





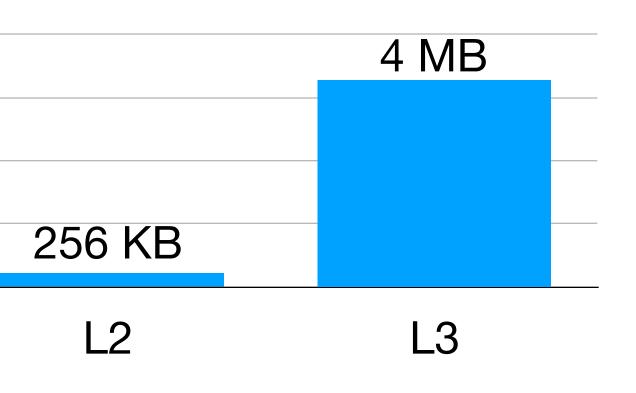




Cache size

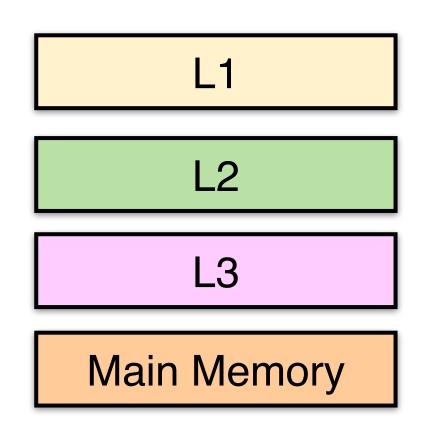
32 KB

L1D/L1I

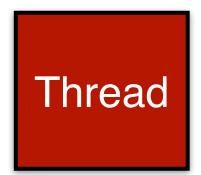


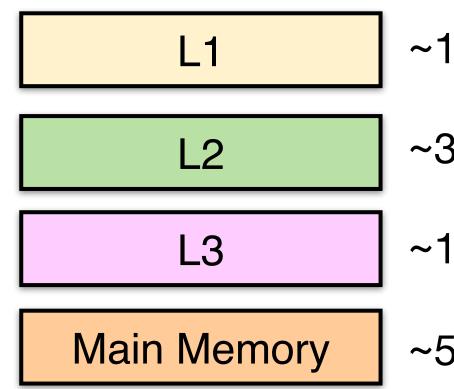




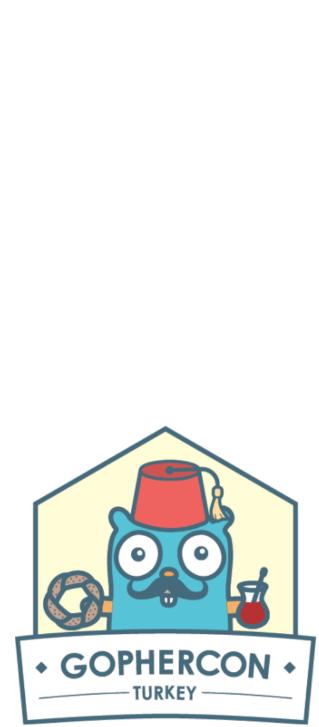








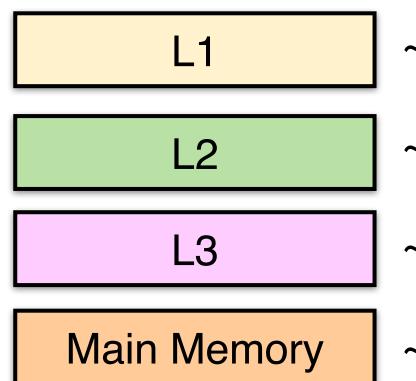
- ~1 ns
- ~3 times slower than L1
- ~10 times slower than L1
- ~50/100 times slower than L1



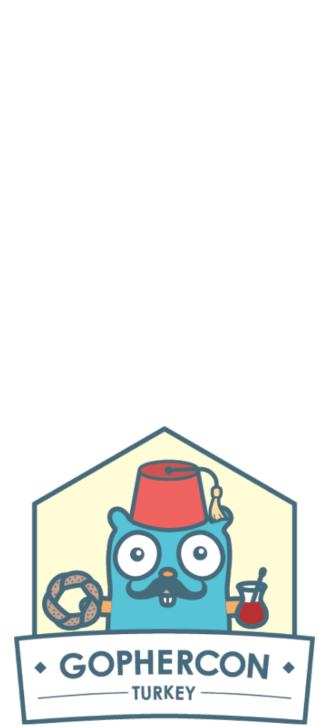




As a developer, I would like my application to leverage CPU caches



- ~1 ns
- ~3 times slower than L1
- ~10 times slower than L1
- ~50/100 times slower than L1



CPU Architecture Locality of Reference **Data-Oriented Design** Caching Pitfall Concurrency



If a particular memory location is referenced, it is likely that...



If a particular memory location is referenced, it is likely that...

The same location will be referenced again in a near future



If a particular memory location is referenced, it is likely that...



The same location will be referenced again in a near future



If a **particular memory location** is referenced, it is **likely** that...



The same location will be referenced again in a near future

sum := 0 s := initSliceOfInts() length := len(s)sum += s[i]



If a **particular memory location** is referenced, it is **likely** that...



The same location will be referenced again in a near future

sum := 0 s := initSliceOfInts() length := len(s)sum += s[i]



If a **particular memory location** is referenced, it is **likely** that...



The same location will be referenced again in a near future

- sum := 0
- s := initSliceOfInts() sum += s[i]
- length := len(s)for i := 0; i < length; i++ {</pre>



If a **particular memory location** is referenced, it is **likely** that...



The same location will be referenced again in a near future

- sum := 0
- s := initSliceOfInts()
- length := len(s)
- - sum += s[i]



If a **particular memory location** is referenced, it is **likely** that...

Temporal Locality

The same location will be referenced again in a near future

Nearby memory locations will be referenced in a near future

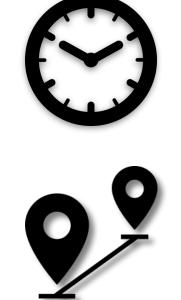
- sum := 0
- s := initSliceOfInts()
- length := len(s)
- - sum += s[i]



If a **particular memory location** is referenced, it is **likely** that...

Temporal Locality

> Spatial Locality



The same location will be referenced again in a near future

Nearby memory locations will be referenced in a near future

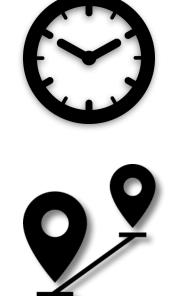
- sum := 0
- s := initSliceOfInts()
- length := len(s)
- - sum += s[i]



If a **particular memory location** is referenced, it is **likely** that...

Temporal Locality

> Spatial Locality



The same location will be referenced again in a near future

Nearby memory locations will be referenced in a near future

sum := 0 s := initSliceOfInts() length := len(s)SUM += S[i]





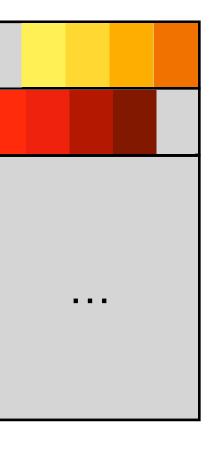
Slice (in our heads)





Slice (in our heads)

Matrix (main memory)

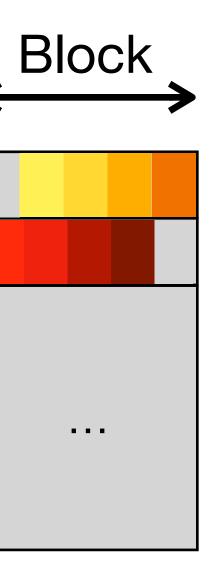






Slice (in our heads)

Matrix (main memory)

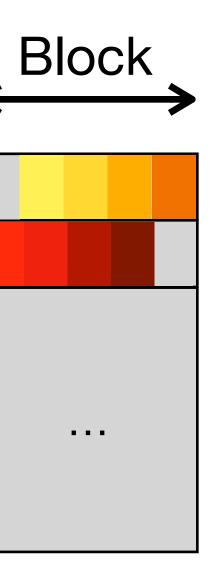






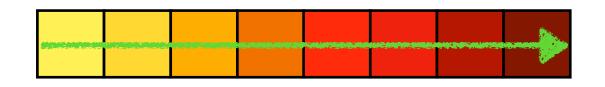
Slice (in our heads)

Matrix (main memory)



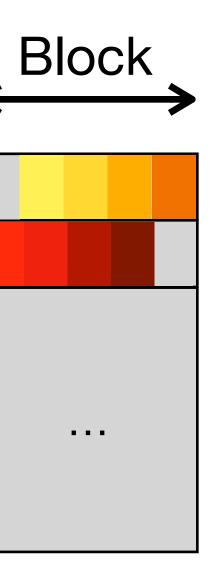






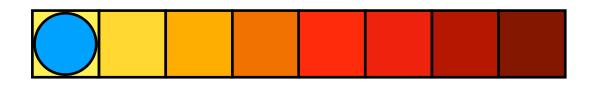
Slice (in our heads)

Matrix (main memory)



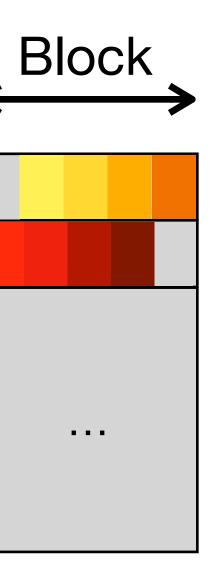






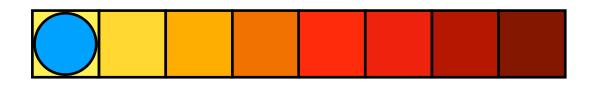
Slice (in our heads)

Matrix (main memory)



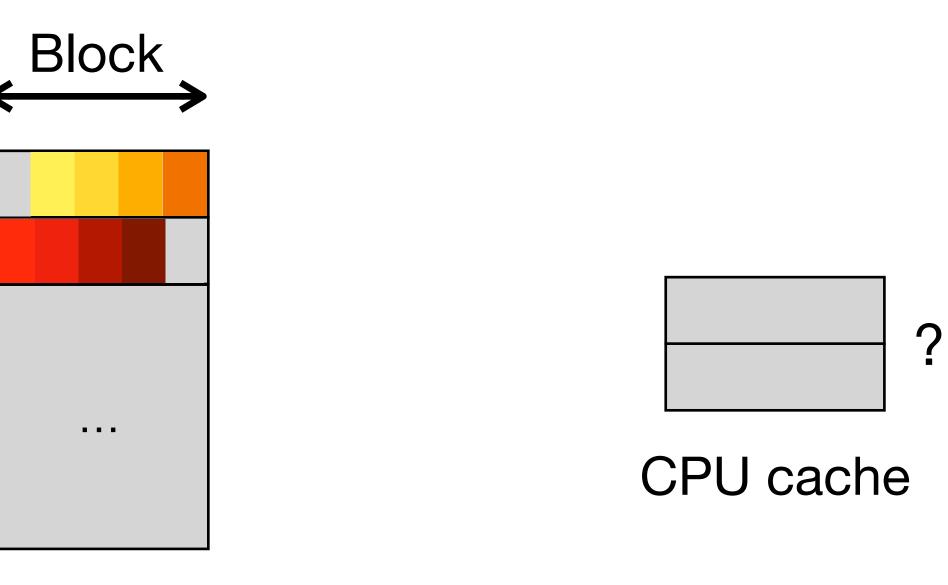




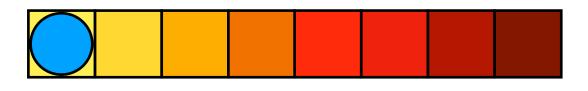


Slice (in our heads)

Matrix (main memory)

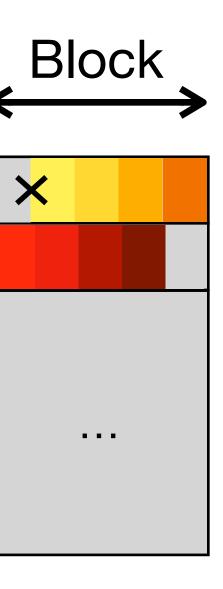






Slice (in our heads)

Matrix (main memory)





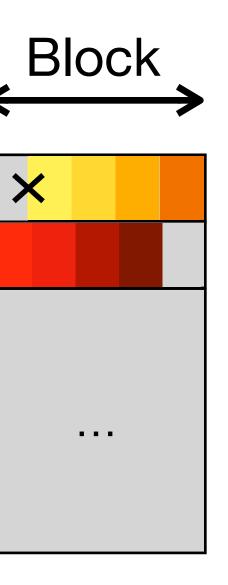




Slice (in our heads)

Matrix (main memory)

• Instead of copying a single variable, the processor will copy a cache line

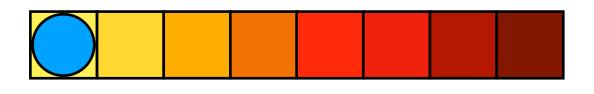








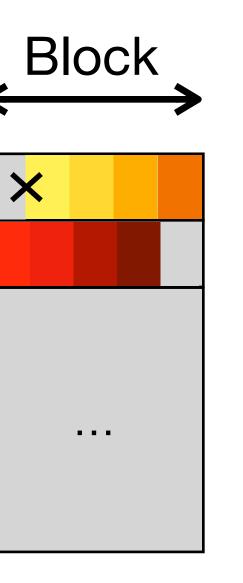




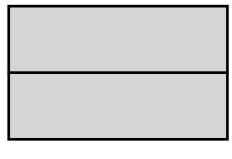
Slice (in our heads)

Matrix (main memory)

- Instead of copying a single variable, the processor will copy a cache line
- Cache line: contiguous segment of memory of a fixed size (usually 64 bytes)





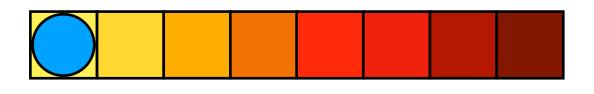


CPU cache

rocessor will copy a <mark>cache line</mark> ory of a **fixed size** (usually 64 bytes)



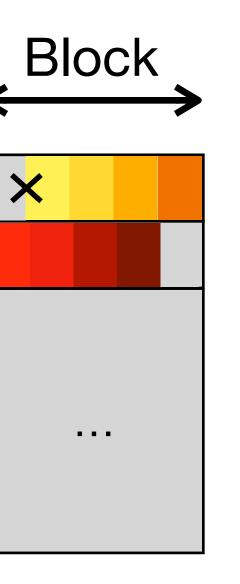


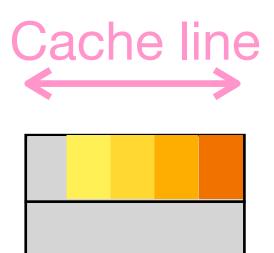


Slice (in our heads)

Matrix (main memory)

- Instead of copying a single variable, the processor will copy a cache line
- Cache line: contiguous segment of memory of a fixed size (usually 64 bytes)





CPU cache

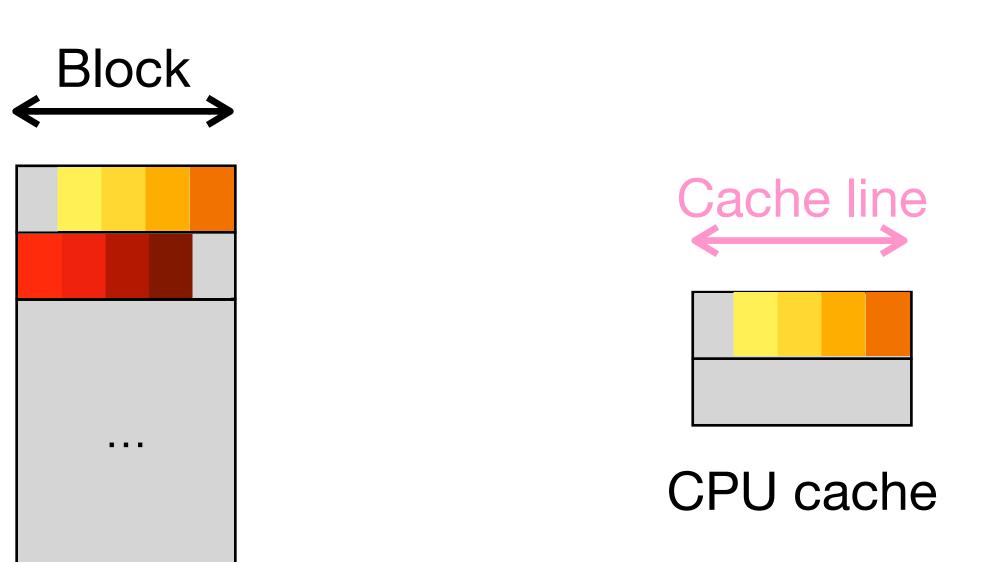




Slice (in our heads)

Matrix (main memory)

- Instead of copying a single variable, the processor will copy a cache line
- Cache line: contiguous segment of memory of a fixed size (usually 64 bytes)



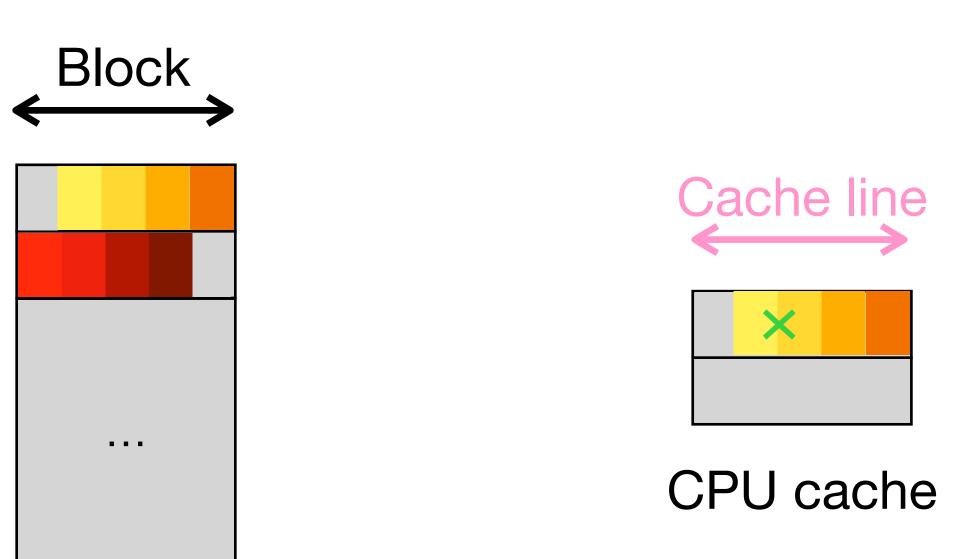




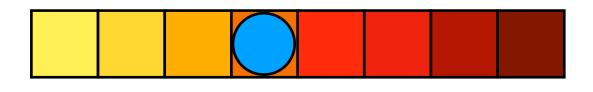
Slice (in our heads)

Matrix (main memory)

- Instead of copying a single variable, the processor will copy a cache line
- Cache line: contiguous segment of memory of a fixed size (usually 64 bytes)



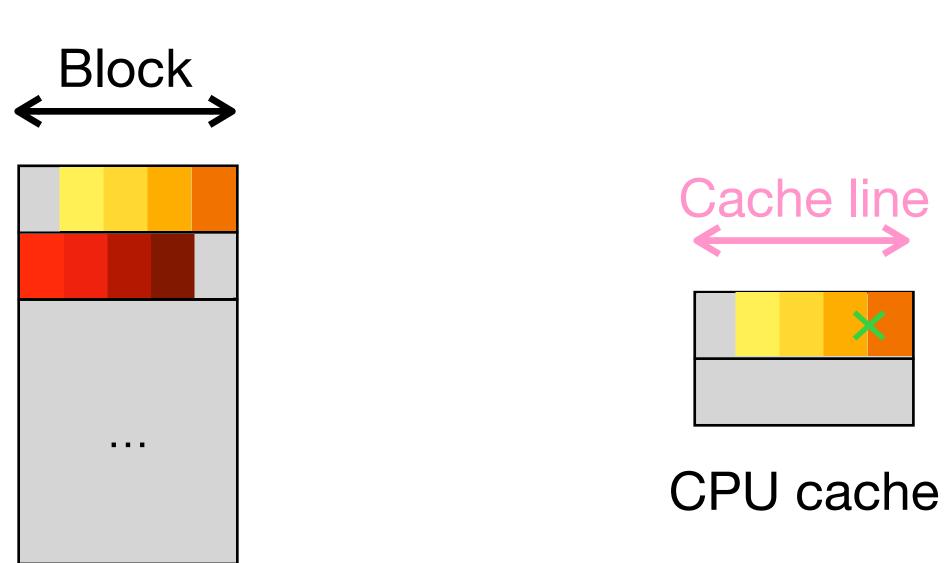




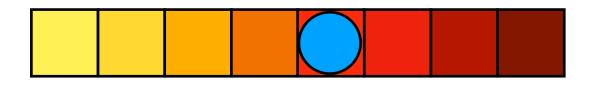
Slice (in our heads)

Matrix (main memory)

- Instead of copying a single variable, the processor will copy a cache line
- Cache line: contiguous segment of memory of a fixed size (usually 64 bytes)



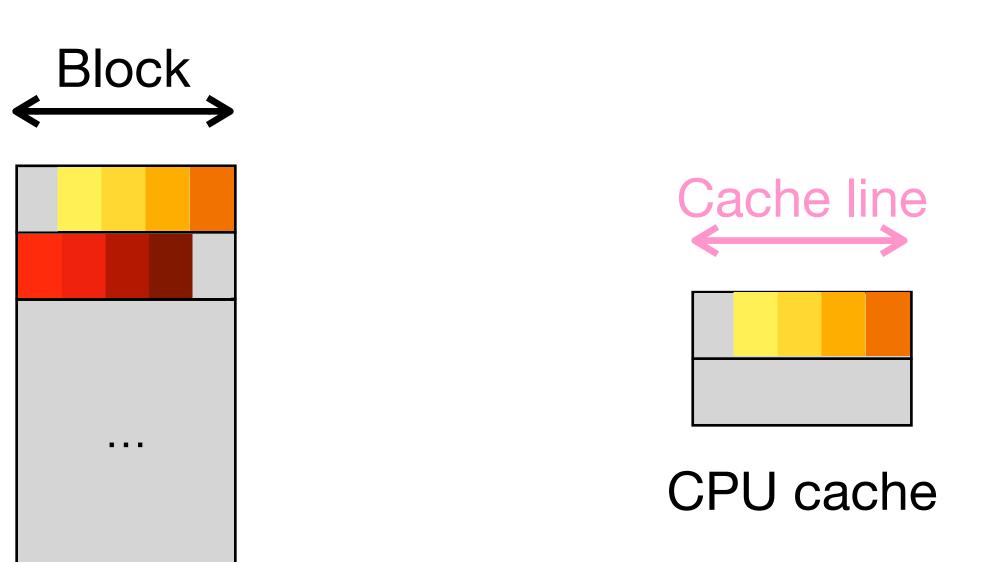




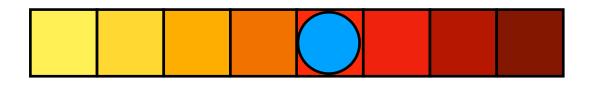
Slice (in our heads)

Matrix (main memory)

- Instead of copying a single variable, the processor will copy a cache line
- Cache line: contiguous segment of memory of a fixed size (usually 64 bytes)



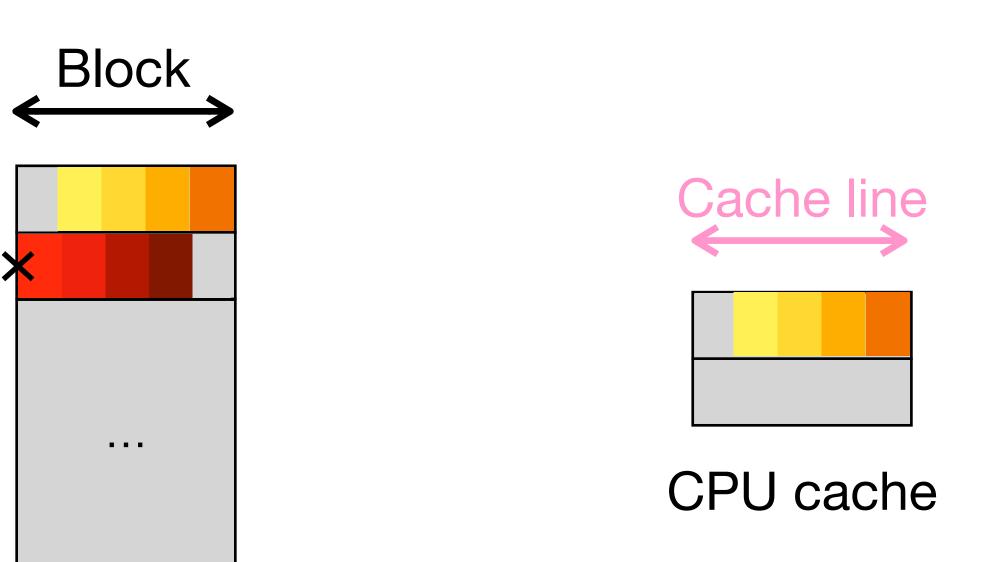




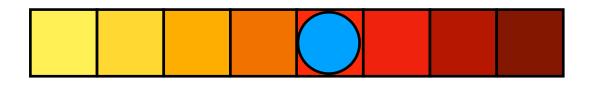
Slice (in our heads)

Matrix (main memory)

- Instead of copying a single variable, the processor will copy a cache line
- Cache line: contiguous segment of memory of a fixed size (usually 64 bytes)



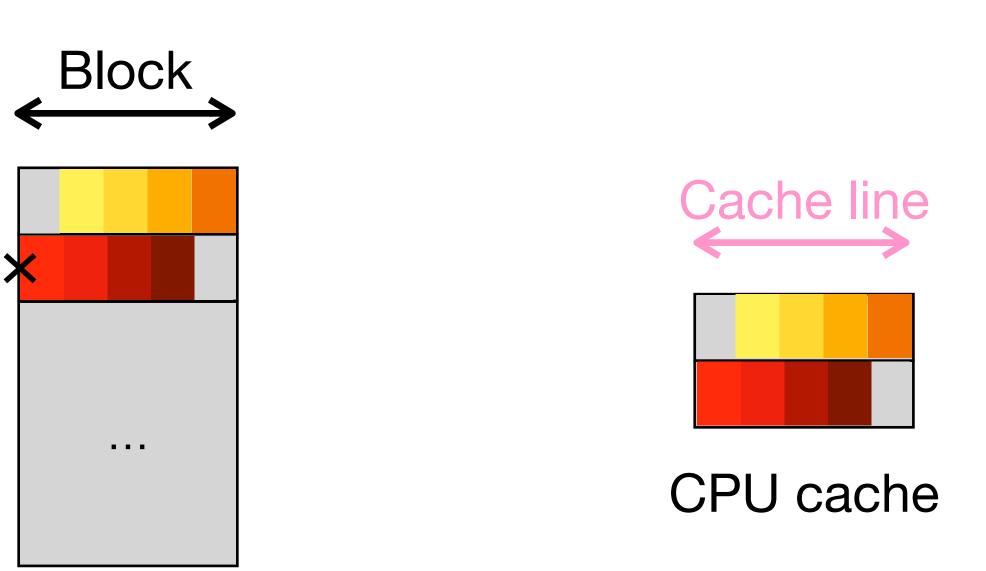




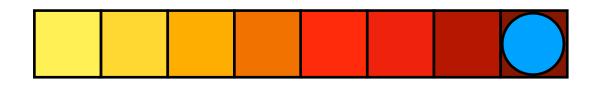
Slice (in our heads)

Matrix (main memory)

- Instead of copying a single variable, the processor will copy a cache line
- Cache line: contiguous segment of memory of a fixed size (usually 64 bytes)



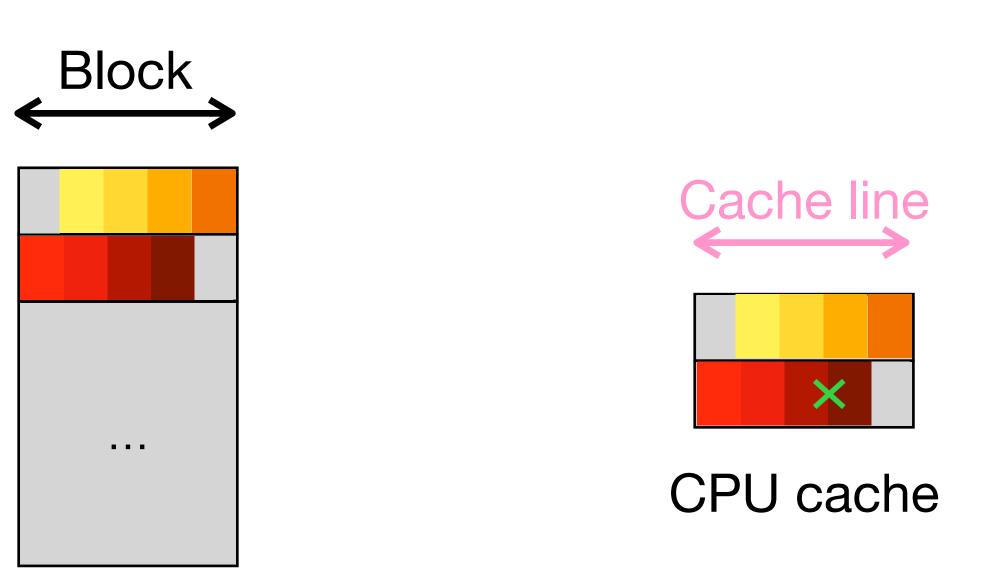




Slice (in our heads)

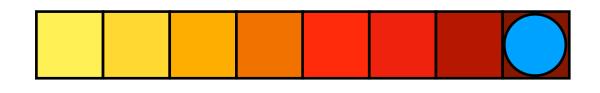
Matrix (main memory)

- Instead of copying a single variable, the processor will copy a cache line
- Cache line: contiguous segment of memory of a fixed size (usually 64 bytes)





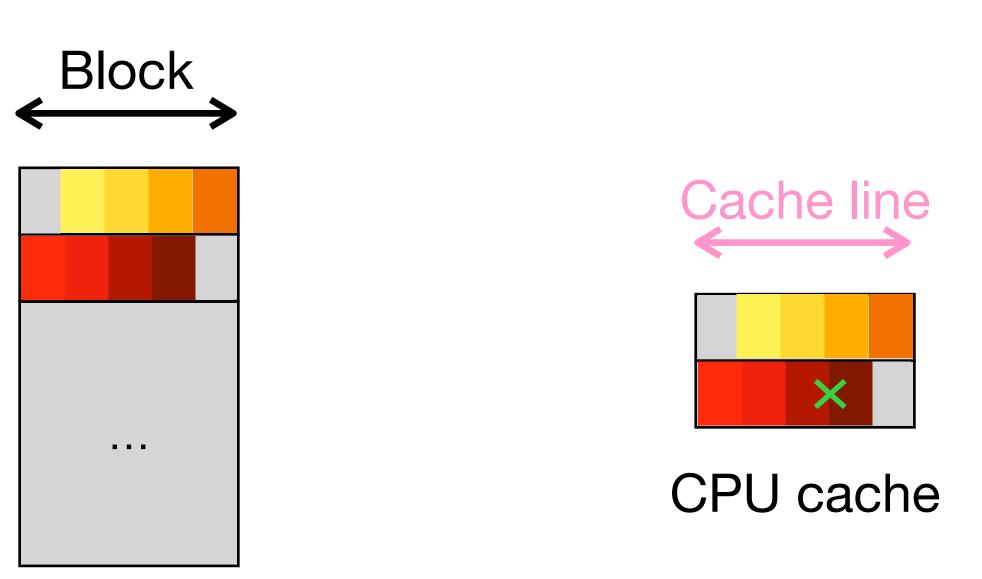




Slice (in our heads)

Matrix (main memory)

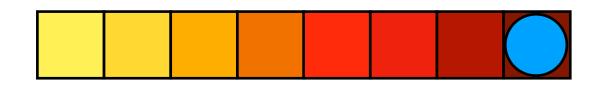
- Instead of copying a single variable, the processor will copy a cache line
- Cache line: contiguous segment of memory of a fixed size (usually 64 bytes)
- Limited number of cache miss (compulsory miss)



rocessor will copy a **cache line** ory of a **fixed size** (usually 64 bytes) ry miss)



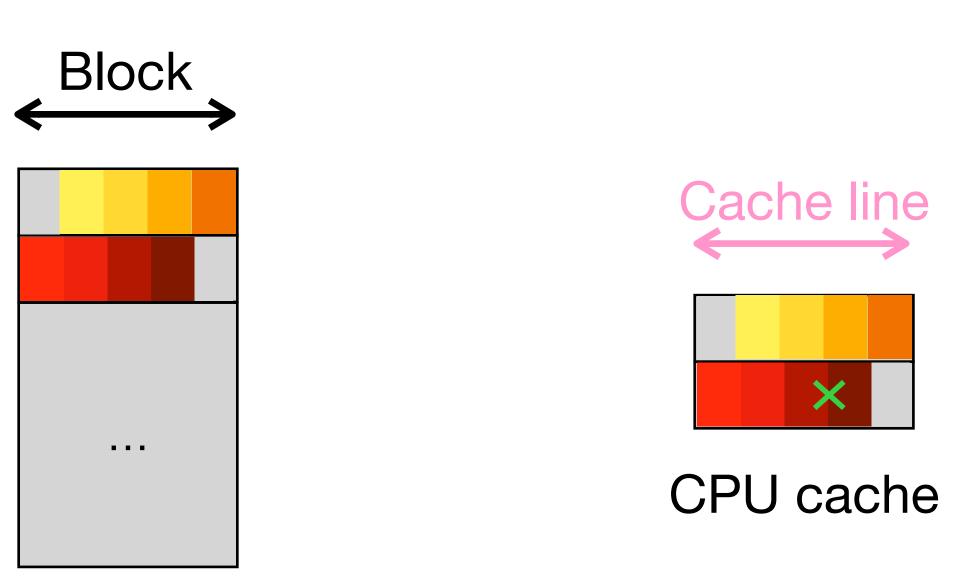




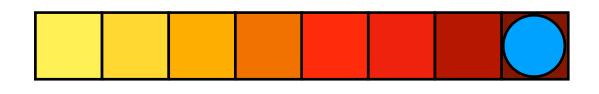
Slice (in our heads)

Matrix (main memory)

- Instead of copying a single variable, the processor will copy a cache line
- Cache line: contiguous segment of memory of a fixed size (usually 64 bytes)
- Limited number of cache miss (compulsory miss)
- Theory (other applications can run at the same time on the same core)



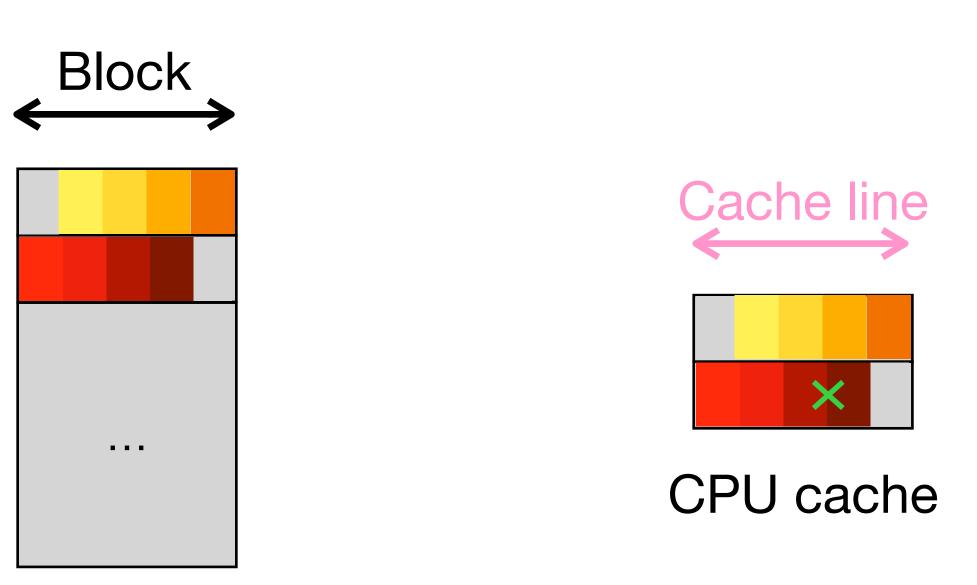




Slice (in our heads)

Matrix (main memory)

- Instead of copying a single variable, the processor will copy a cache line
- Cache line: contiguous segment of memory of a fixed size (usually 64 bytes)
- Limited number of cache miss (compulsory miss)
- Theory (other applications can run at the same time on the same core)
- Cache placement policy (L1, L2 or L3?)





Helping the CPU



Helping the CPU

• To help the CPU, an application needs to leverage locality of reference



Helping the CPU

• To help the CPU, an application needs to leverage locality of reference

• ... and predictability



•

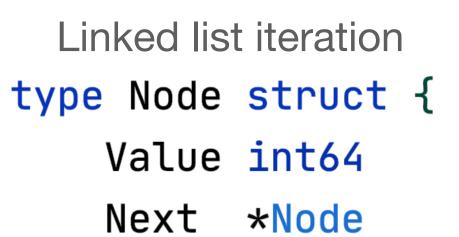


•

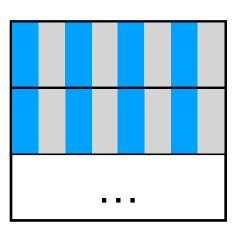
Linked list iteration type Node struct { Value int64 Next *Node }



•



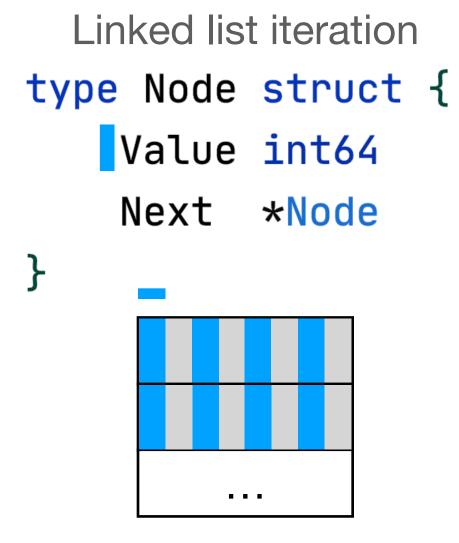




Main memory



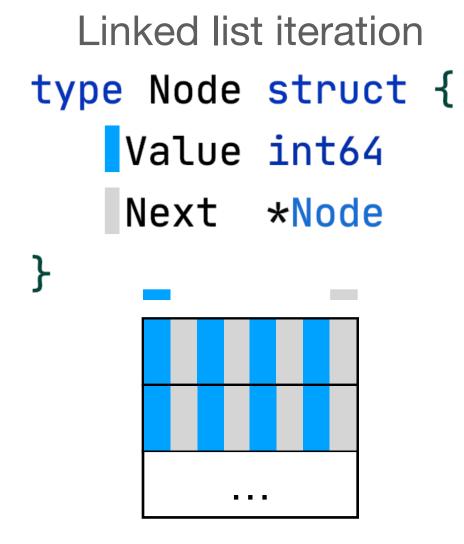
•



Main memory



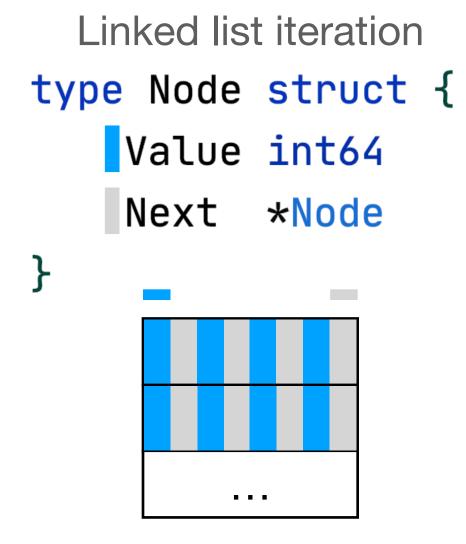
•



Main memory



Iterating on a linked list that should be allocated contiguously should be decent ullet

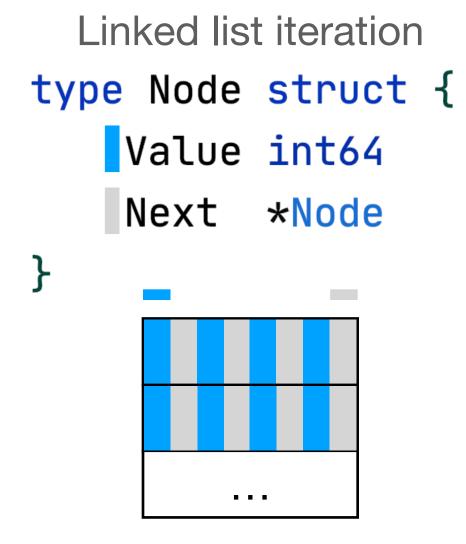


Main memory

```
Slice iteration: one element out of two
 for i := 0; i < len(s); i+=2 {</pre>
     sum += s[i]
 }
```

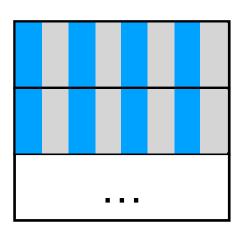


lacksquare



Main memory

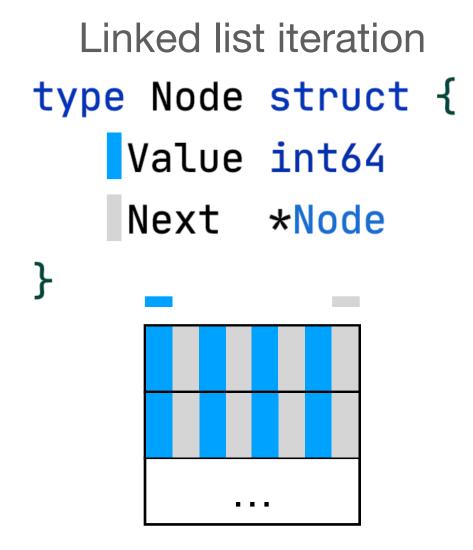
```
Slice iteration: one element out of two
 for i := 0; i < len(s); i+=2 {</pre>
     sum += s[i]
 }
```



```
Main memory
```



 \bullet



Main memory

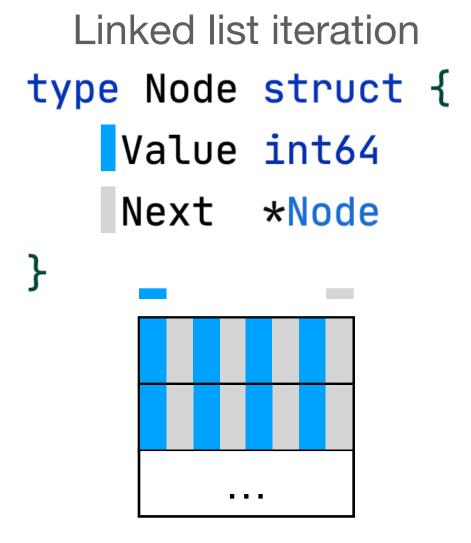
~230% slower



```
Slice iteration: one element out of two
 for i := 0; i < len(s); i+=2 {</pre>
     sum += s[i]
 }
                 . . .
         Main memory
   Slice iteration
```



 \bullet



Main memory

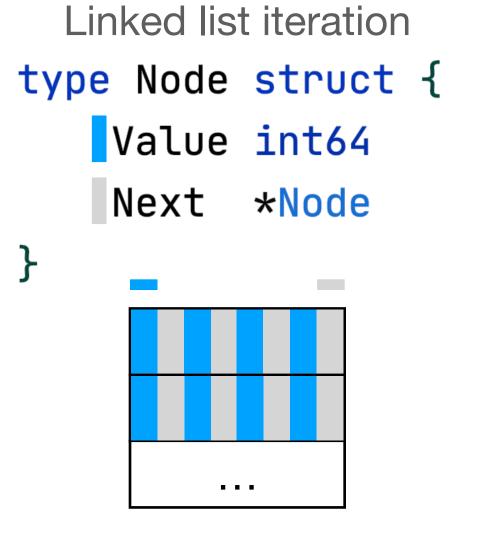
~230% slower



• **Possible** spatial locality

```
Slice iteration: one element out of two
 for i := 0; i < len(s); i+=2 {</pre>
     sum += s[i]
 }
                 . . .
         Main memory
   Slice iteration
```





Main memory

~230% slower



- **Possible** spatial locality
- But **not predictable** for the CPU (no line fetching)

```
Slice iteration: one element out of two
 for i := 0; i < len(s); i+=2 {</pre>
     sum += s[i]
 }
                 . . .
         Main memory
   Slice iteration
```



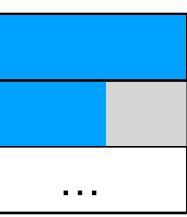
• What if we iterate **backwards** on a slice?



• What if we iterate **backwards** on a slice?



Main memory





• What if we iterate **backwards** on a slice?

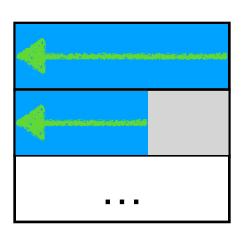


Main memory





• What if we iterate **backwards** on a slice?



Main memory

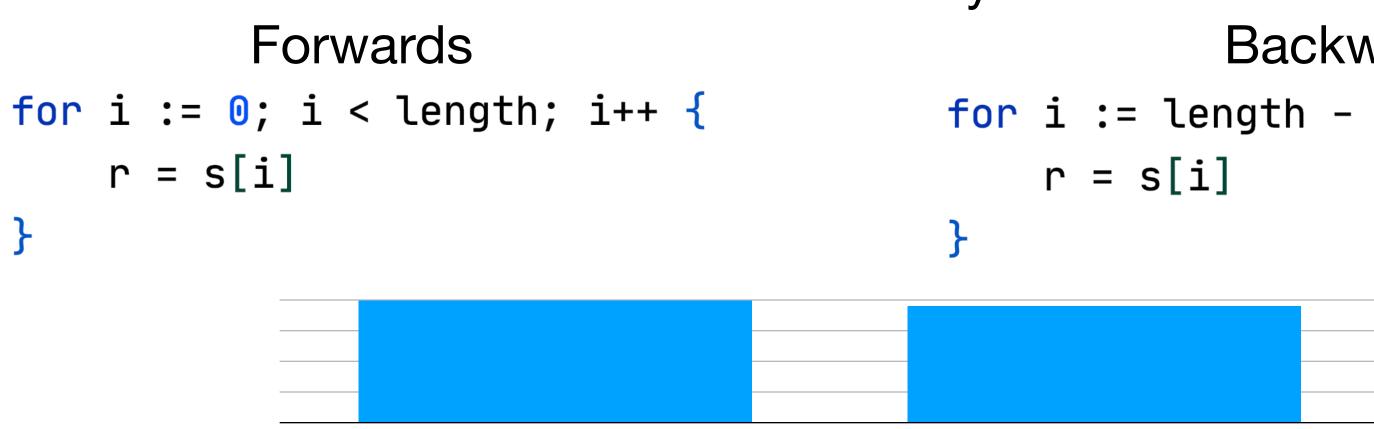
Forwards for i := 0; i < length; i++ {</pre> r = s[i]}

Backwards for i := length - 1; i >= 0; i-- { r = s[i]}



• What if we iterate **backwards** on a slice?





Forwards



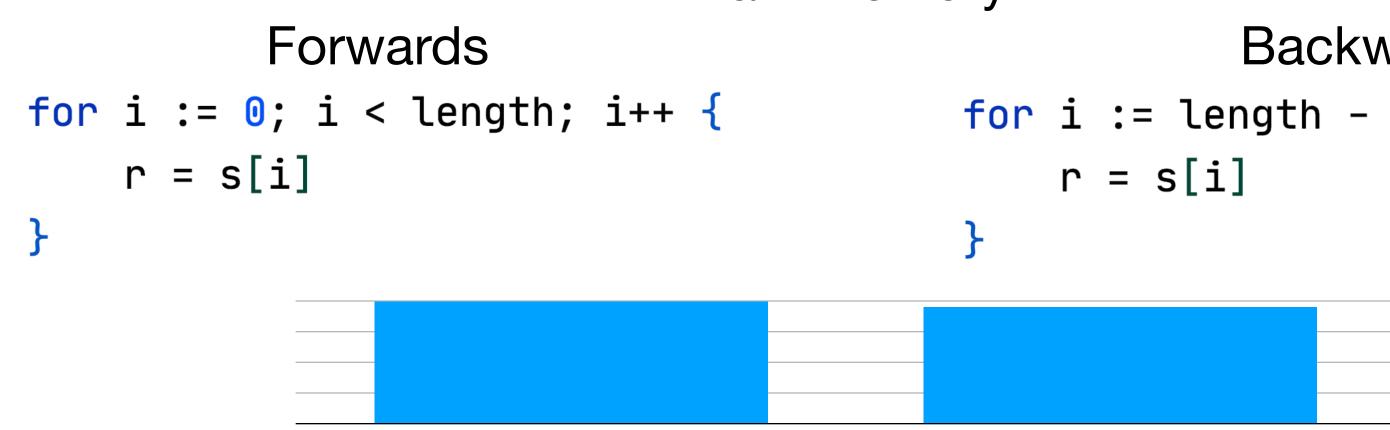
Main memory Backwards for i := length - 1; i >= 0; i-- {

Backwards



• What if we iterate **backwards** on a slice?





Forwards

• Spatial locality



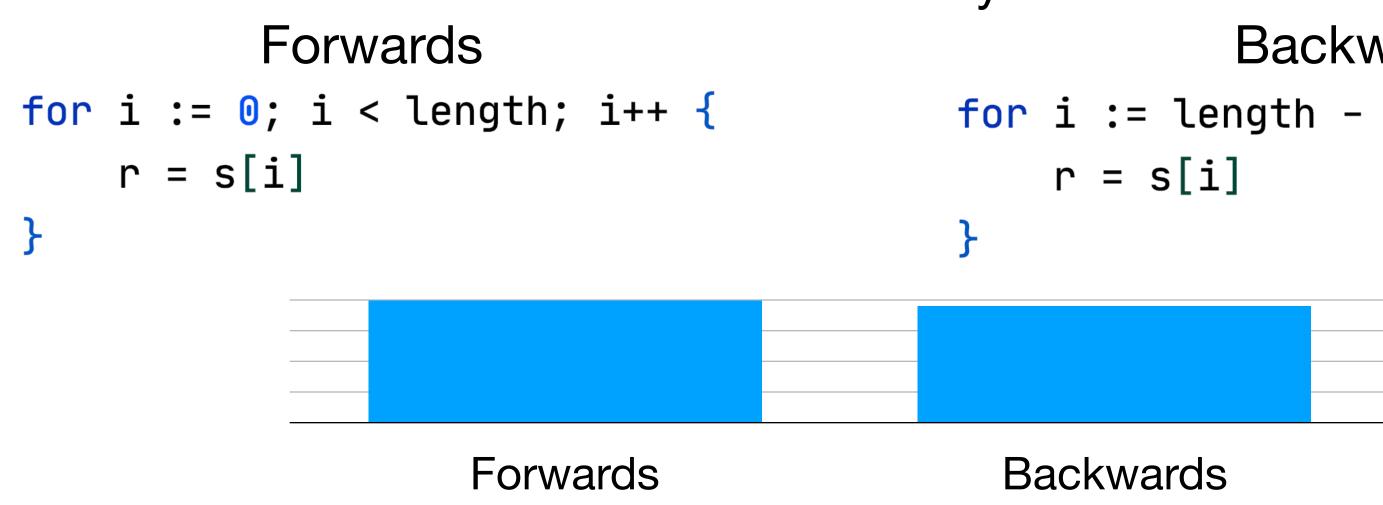
Main memory Backwards for i := length - 1; i >= 0; i-- {

Backwards



• What if we iterate **backwards** on a slice?





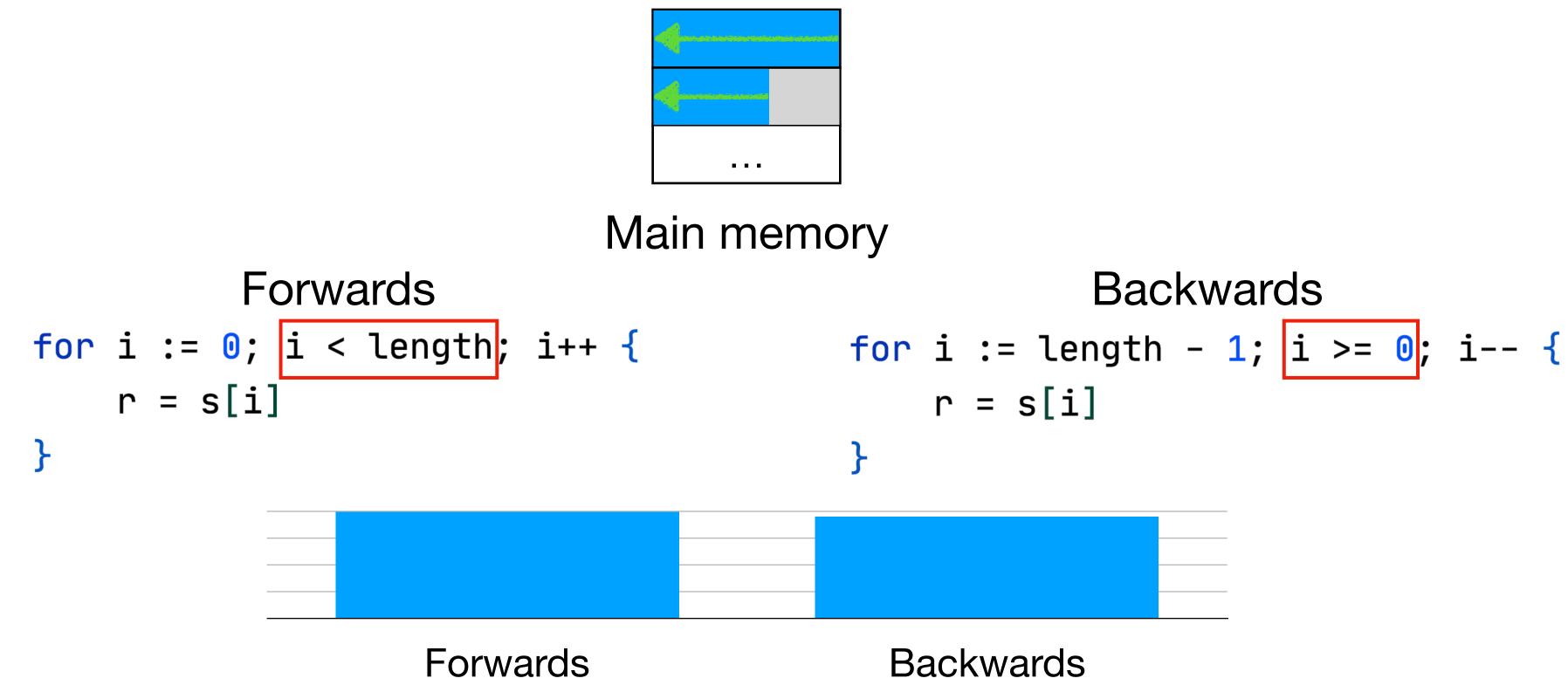
- Spatial locality
- The CPU was able to **predict** that we iterate backwards



Main memory Backwards for i := length - 1; i >= 0; i-- {



• What if we iterate **backwards** on a slice?



- Spatial locality \bullet
- The CPU was able to **predict** that we iterate backwards
- Slightly faster because the bound check is faster







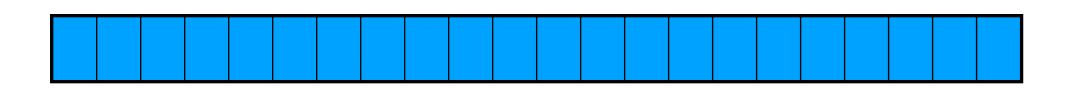
• Striding: how does a CPU work through our data?



- Striding: how does a CPU work through our data?
 - Unit stride: each element

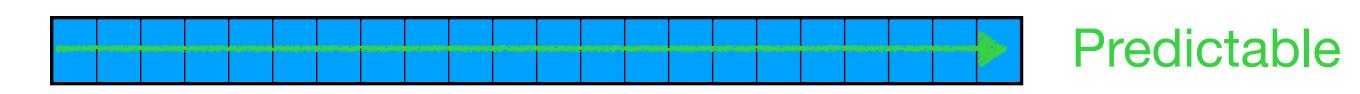


- Striding: how does a CPU work through our data?
 - Unit stride: each element lacksquare





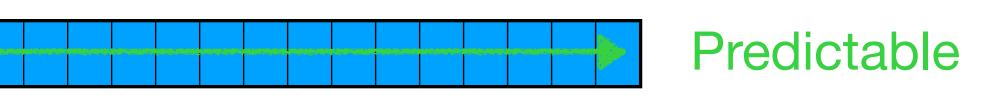
- Striding: how does a CPU work through our data?
 - Unit stride: each element lacksquare





- Striding: how does a CPU work through our data?
 - Unit stride: each element lacksquare

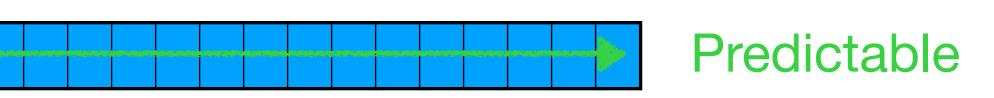


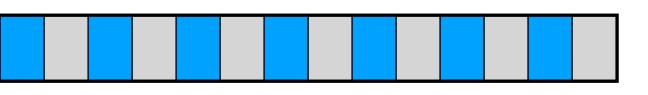




- Striding: how does a CPU work through our data?
 - Unit stride: each element lacksquare

• Constant stride: each x element (e.g. one out of two)

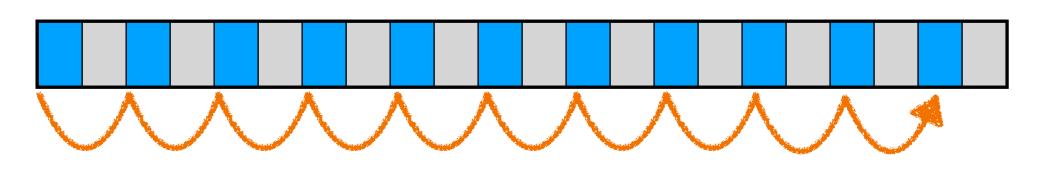


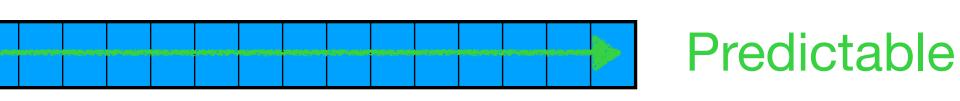




- Striding: how does a CPU work through our data?
 - Unit stride: each element lacksquare

• Constant stride: each x element (e.g. one out of two)





Predictable but less efficient

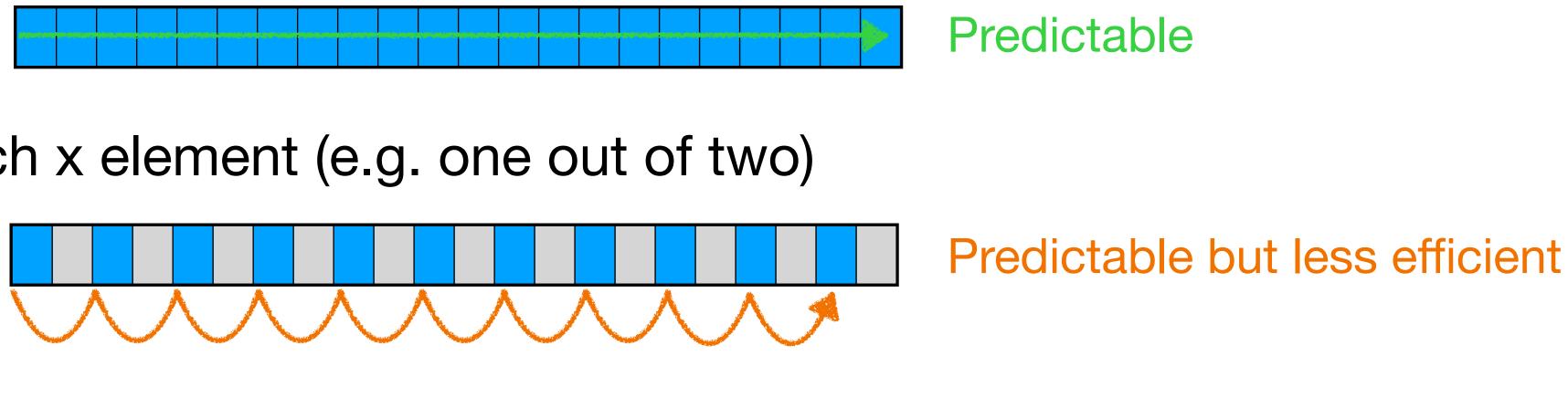






- Striding: how does a CPU work through our data?
 - Unit stride: each element

• Constant stride: each x element (e.g. one out of two)



• Non-unit stride: *might* be spread across memory (linked list)

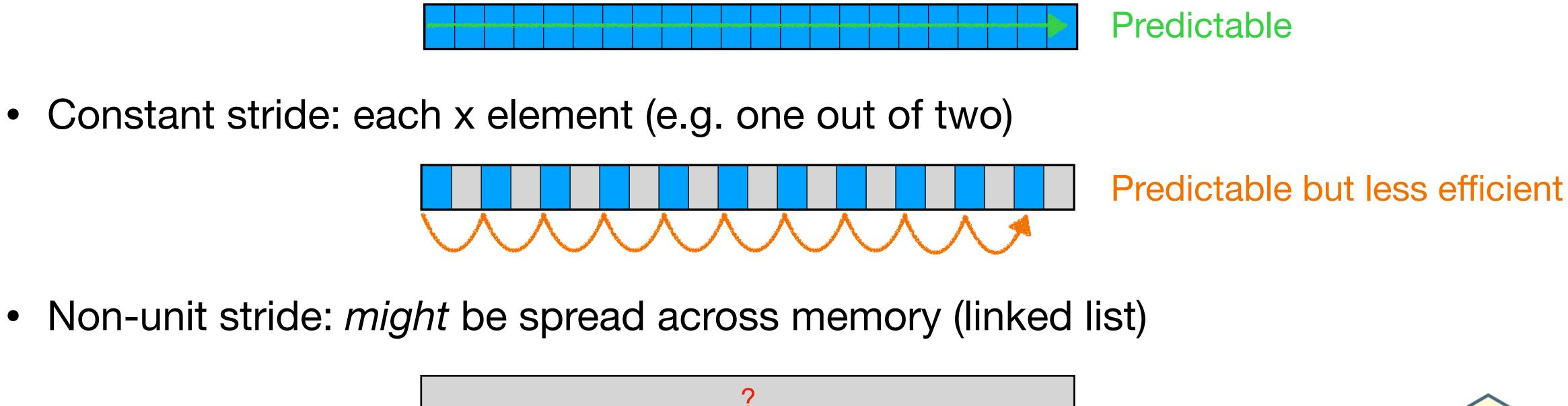






- Striding: how does a CPU work through our data?
 - Unit stride: each element

• Constant stride: each x element (e.g. one out of two)



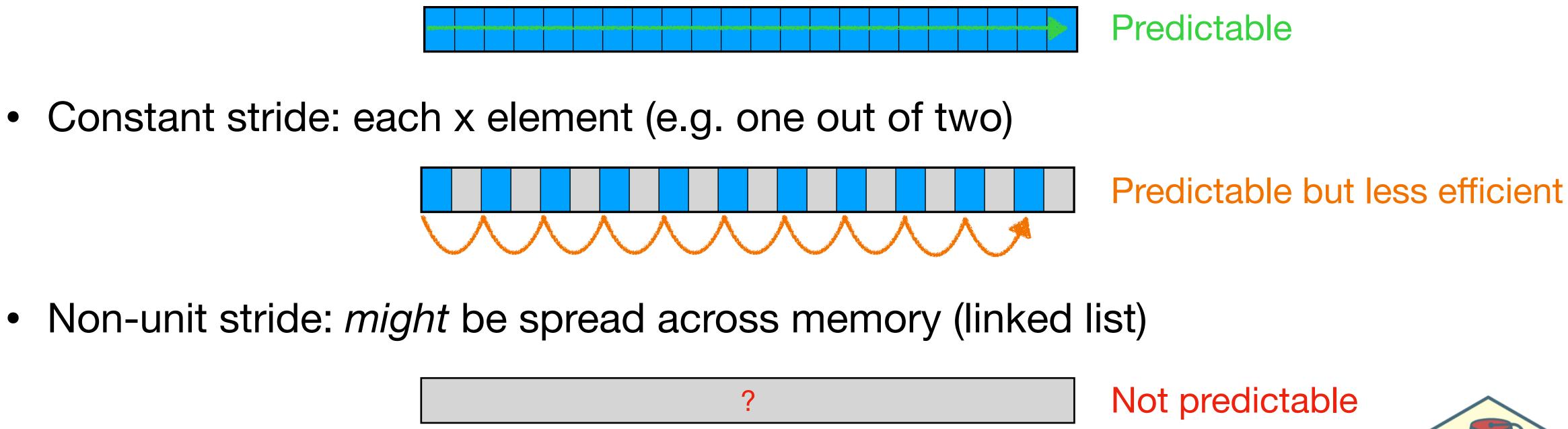






- Striding: how does a CPU work through our data?
 - Unit stride: each element

• Constant stride: each x element (e.g. one out of two)















CPU caches are extremely fast





- CPU caches are extremely fast
- A CPU doesn't cache a single variable but a cache line





- CPU caches are extremely fast
- A CPU doesn't cache a single variable but a cache line
- I can help the CPU if my application leverages: •





- CPU caches are extremely fast
- A CPU doesn't cache a single variable but a cache line
- I can help the CPU if my application leverages: lacksquare
 - Locality of reference \bullet





- CPU caches are extremely fast
- A CPU doesn't cache a single variable but a cache line
- I can help the CPU if my application leverages: lacksquare
 - Locality of reference
 - Predictability



CPU Architecture Locality of Reference **Data-Oriented Design** Caching Pitfall Concurrency





from one form to another" - Mike Acton

• "The purpose of all programs and all parts of those programs is to transform data





from one form to another" - Mike Acton

"The purpose of all programs and all parts of those programs is to transform data

Object-Oriented design is a way to mirror how we interact with the real world





from one form to another" - Mike Acton

- Yet, hardware does not like objects

"The purpose of all programs and all parts of those programs is to transform data

Object-Oriented design is a way to mirror how we interact with the real world





from one form to another" - Mike Acton

- Yet, hardware does not like objects

of each cache line

"The purpose of all programs and all parts of those programs is to transform data

Object-Oriented design is a way to mirror how we interact with the real world

Data-Oriented design is about organising data in a way to get the most value out





- 2 concrete examples:
 - Structure alignment
 - Slice of structures vs structure of slices





```
type I1 struct {
    b1 bool
    i int64
    b2 bool
}
```

```
func BenchmarkI1(b *testing.B) {
    s := make([]I1, it)
    var r int64
    b.ResetTimer()
    for j := 0; j < it; j++ {
        r += s[j].i
    }
    result = r
}</pre>
```



type I1 struct {
 b1 bool
 i int64
 b2 bool
}

func BenchmarkI1(b *testing.B) {
 s := make([]I1, it)
 var r int64
 b.ResetTimer()
 for j := 0; j < it; j++ {
 r += s[j].i
 }
 result = r
}</pre>



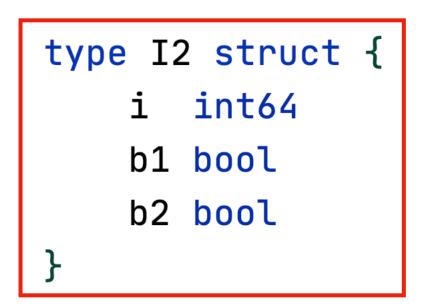
```
type I1 struct {
    b1 bool
    i int64
    b2 bool
}
```

```
func BenchmarkI1(b *testing.B) {
    s := make([]I1, it)
    var r int64
    b.ResetTimer()
    for j := 0; j < it; j++ {
        r += s[j].i
    }
    result = r
}</pre>
```



```
type I1 struct {
    b1 bool
    i int64
    b2 bool
}
```

```
func BenchmarkI1(b *testing.B) {
    s := make([]I1, it)
    var r int64
    b.ResetTimer()
    for j := 0; j < it; j++ {
        r += s[j].i
    }
    result = r
}</pre>
```



```
func BenchmarkI2(b *testing.B) {
    s := make([]I2, it)
    var r int64
    b.ResetTimer()
    for j := 0; j < it; j++ {
        r += s[j].i
    }
    result = r
}</pre>
```



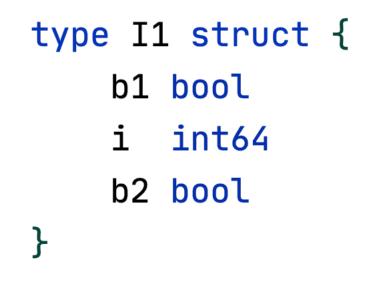
• The size of a structure is a multiple of the word size (64 bits on a 64-bit, etc.)

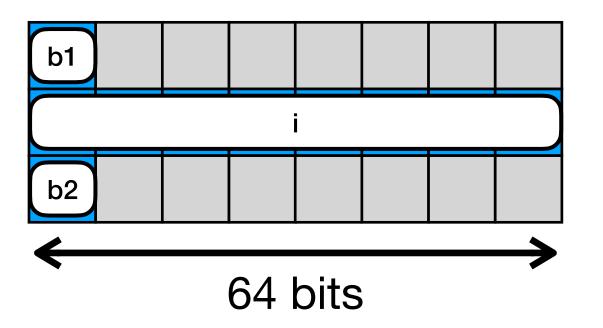


type I1 struct { b1 bool i int64 b2 bool }

• The size of a structure is a multiple of the word size (64 bits on a 64-bit, etc.)





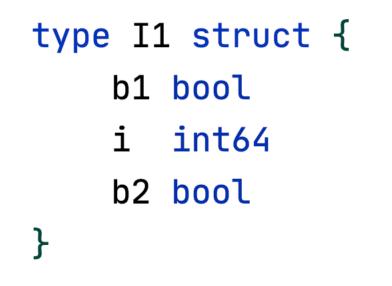


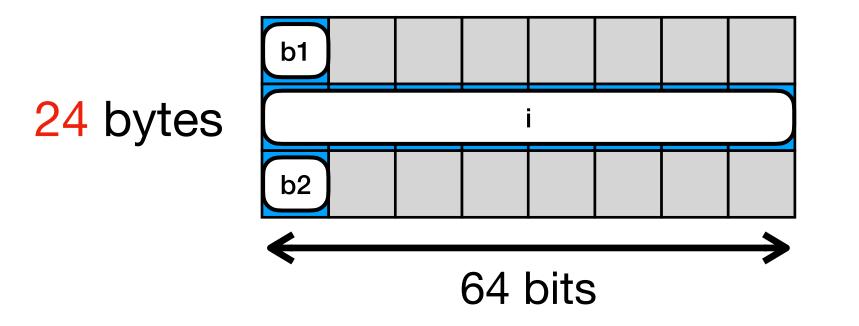


• The size of a structure is a multiple of the word size (64 bits on a 64-bit, etc.)

Structure alignment





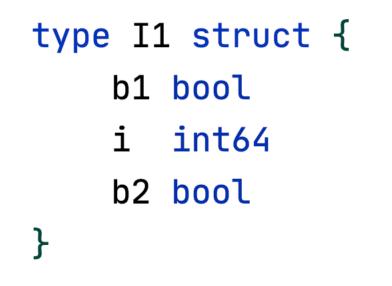


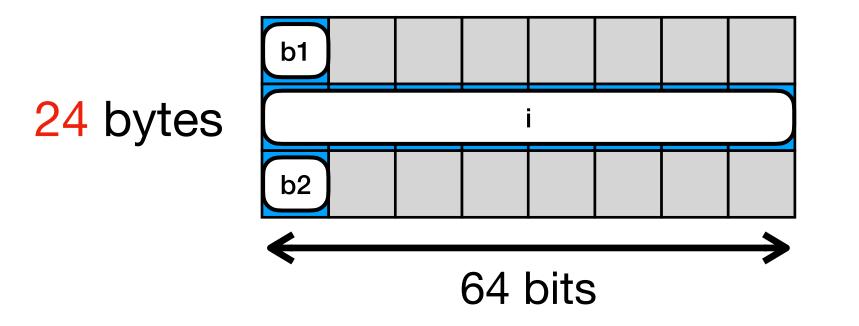


• The size of a structure is a multiple of the word size (64 bits on a 64-bit, etc.)

Structure alignment







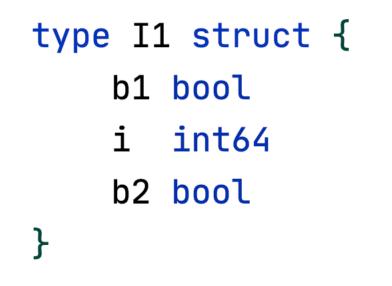


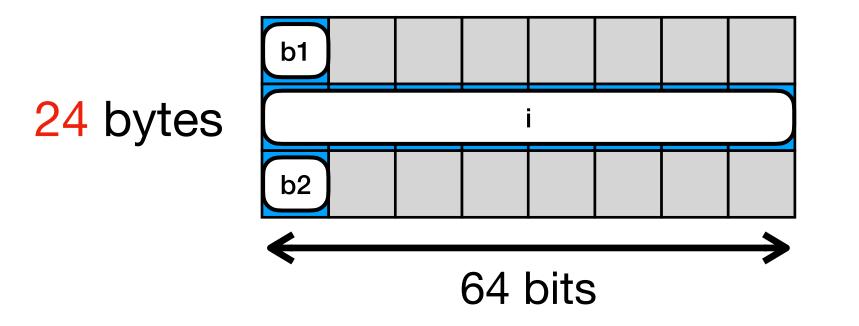
• The size of a structure is a multiple of the word size (64 bits on a 64-bit, etc.)

type I2 struct { i int64 b1 bool b2 bool }

Structure alignment

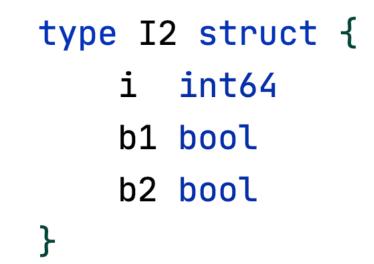


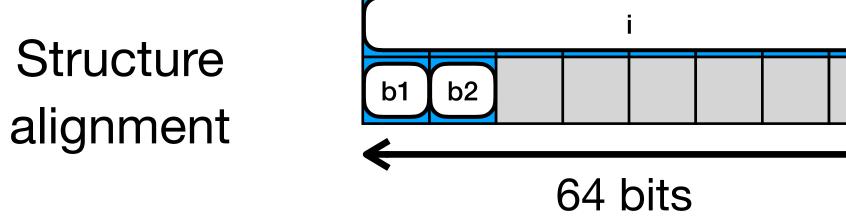




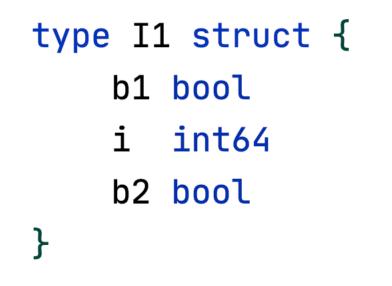


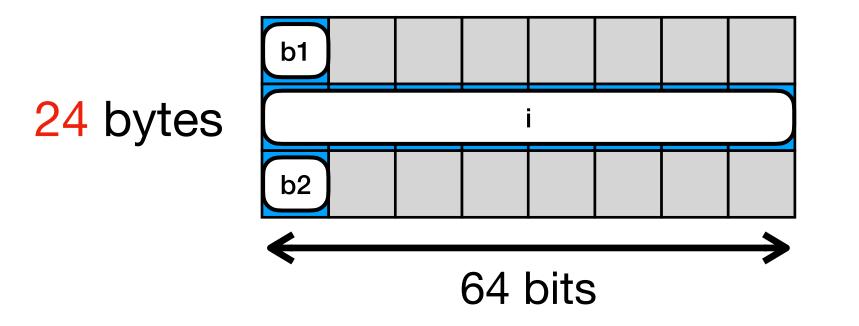
• The size of a structure is a multiple of the word size (64 bits on a 64-bit, etc.)





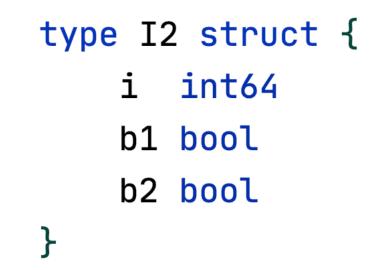


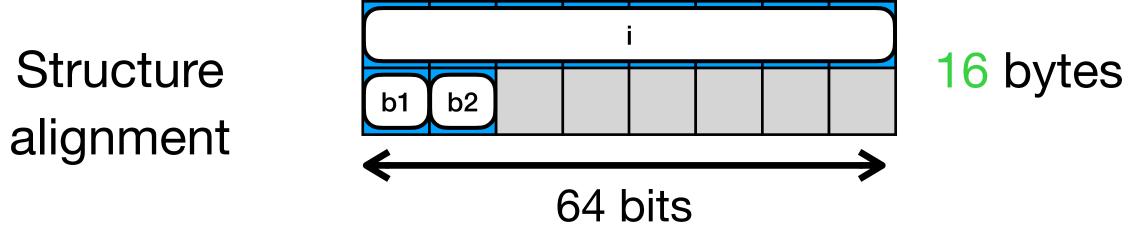




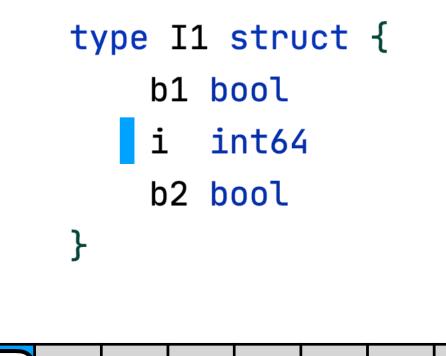


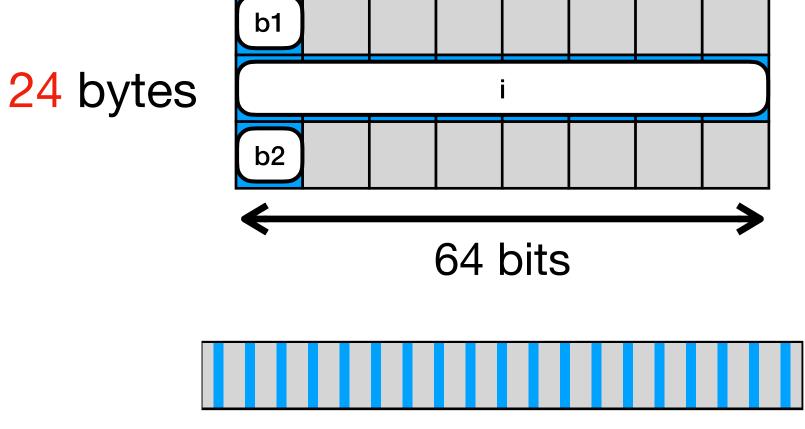
• The size of a structure is a multiple of the word size (64 bits on a 64-bit, etc.)





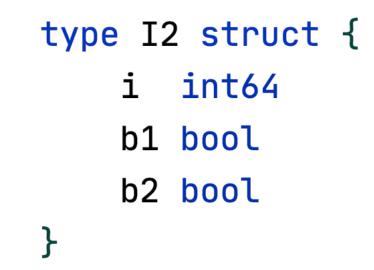


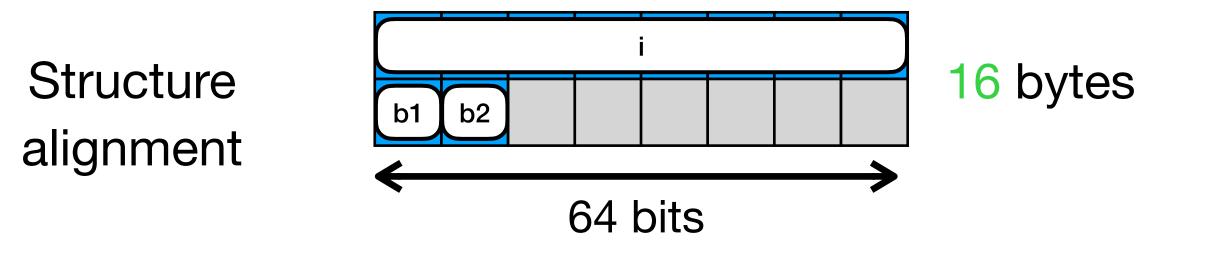




10k slice: 3750 cache lines (cache line: 64 bytes)

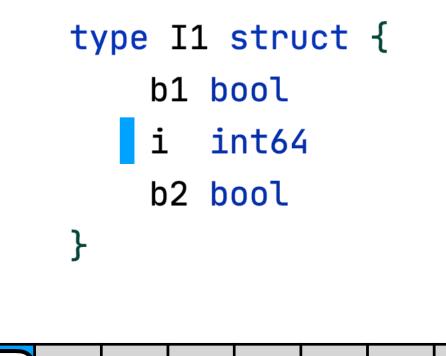
• The size of a structure is a multiple of the word size (64 bits on a 64-bit, etc.)

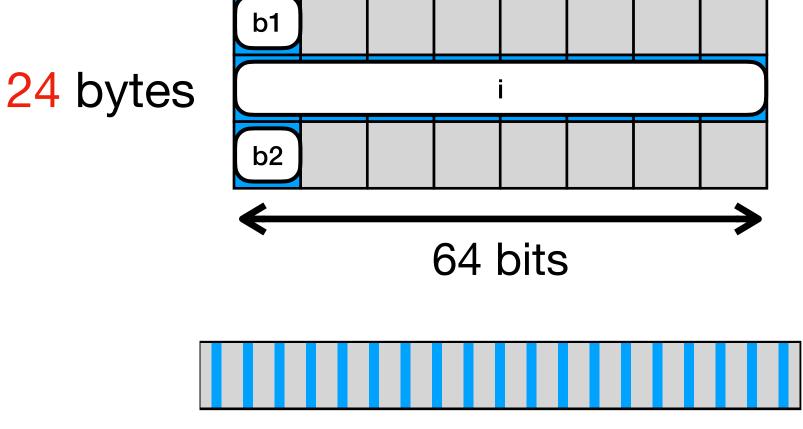




In memory

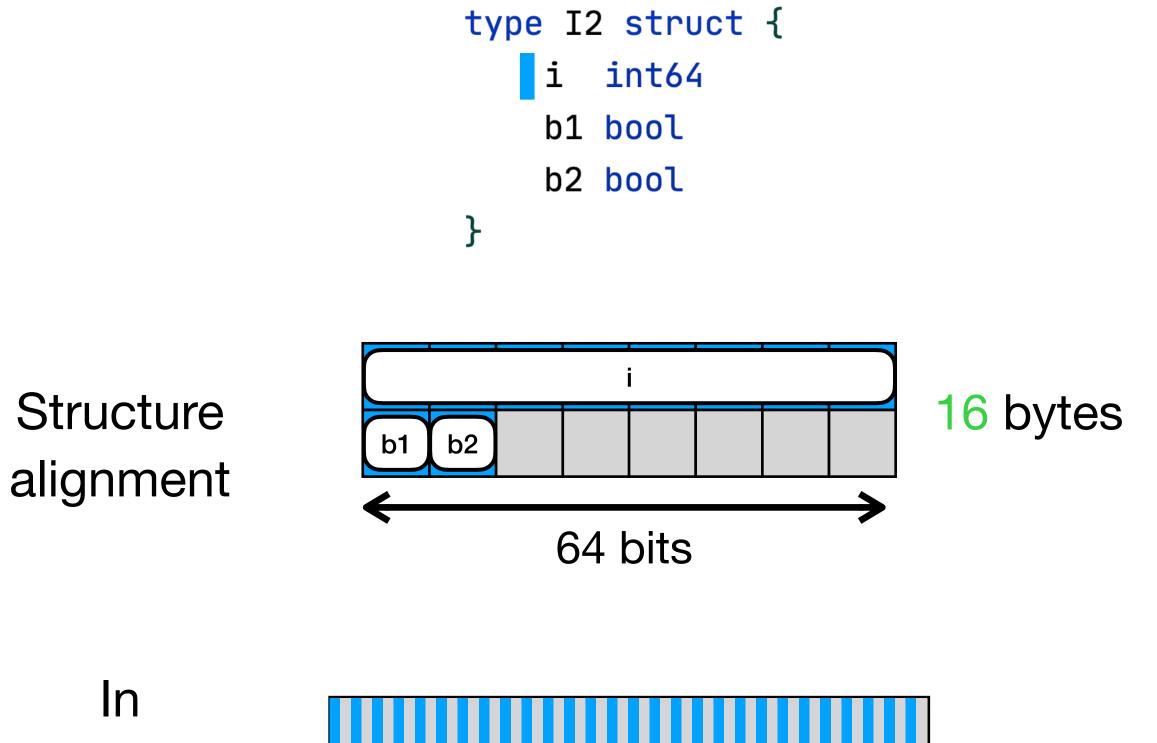






10k slice: 3750 cache lines (cache line: 64 bytes)

• The size of a structure is a multiple of the word size (64 bits on a 64-bit, etc.)



memory

10k slice: 2500 cache lines

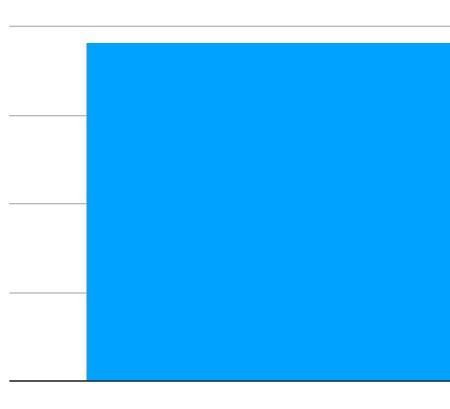


type I1 struct {
 b1 bool
 i int64
 b2 bool
}

```
type I2 struct {
    i int64
    b1 bool
    b2 bool
}
```

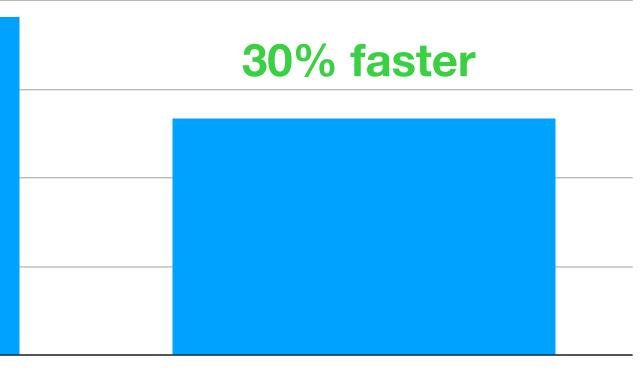


type I1 struct {
 b1 bool
 i int64
 b2 bool
}



Not compact

```
type I2 struct {
    i int64
    b1 bool
    b2 bool
}
```

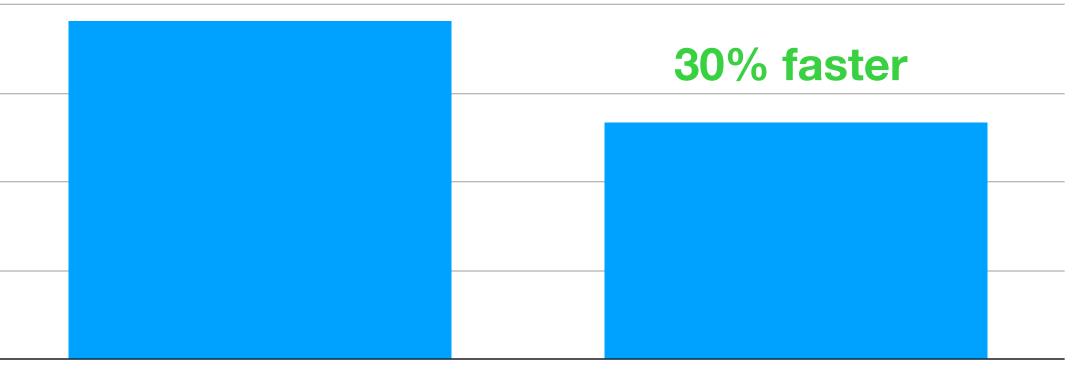


Compact



Structure Alignment

type I1 struct {
 b1 bool
 i int64
 b2 bool
}



Not compact

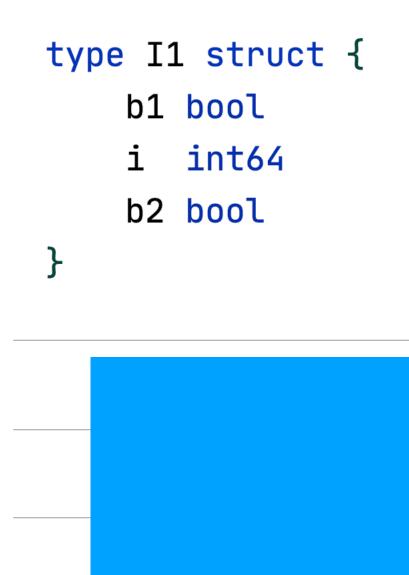
• Memory footprint (GC pressure)

```
type I2 struct {
    i int64
    b1 bool
    b2 bool
}
```

Compact



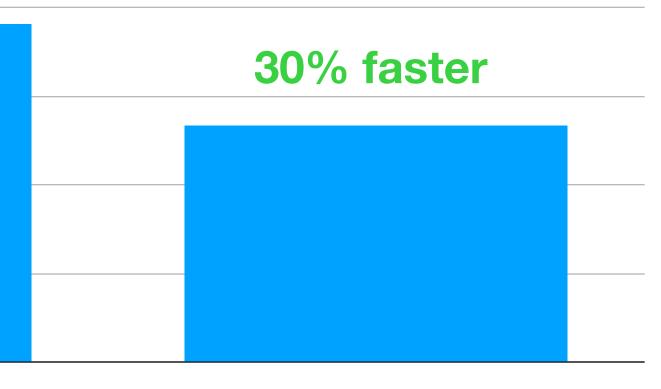
Structure Alignment



Not compact

- Memory footprint (GC pressure)
- Iterating over a **compact** data structure is more efficient as it requires less caches lines

```
type I2 struct {
    i int64
    b1 bool
    b2 bool
}
```



Compact





```
type Struct1 struct {
    a int32
    b int64
}
```

```
func BenchmarkSliceOfStructures(b *testing.B) {
    s := make([]Struct1, it)
    var r int32
    b.ResetTimer()
    for i := 0; i < b.N; i++ {
        for i := 0; i < it; i++ {
            r = s[i].a
        }
    }
    result = r
}</pre>
```



type Struct1 struct {
 a int32

b int64

}

func BenchmarkSliceOfStructures(b *testing.B) {

```
s := make([]Struct1, it)
var r int32
b.ResetTimer()
for i := 0; i < b.N; i++ {
    for i := 0; i < it; i++ {
        r = s[i].a
     }
}
result = r</pre>
```



type Struct1 struct {
 a int32

b int64

}

func BenchmarkSliceOfStructures(b *testing.B) {

```
s := make([]Struct1, it)
var r int32
b.ResetTimer()
for i := 0; i < b.N; i++ {
    for i := 0; i < it; i++ {
        r = s[i].a
     }
}
result = r</pre>
```

```
type Struct2 struct {
    a []int32
    b []int64
}
```

```
a: make([]int32, it),
```

```
b: make([]int64, it),
```

```
-
```

```
var r int32
b.ResetTimer()
for i := 0; i < b.N; i++ {
    for i := 0; i < it; i++ {
        r = s.a[i]
    }
</pre>
```

```
י
result = r
```



```
type Struct1 struct {
    a int32
    b int64
}
```

```
func BenchmarkSliceOfStructures(b *testing.B) {
    s := make([]Struct1, it)
    var r int32
    b.ResetTimer()
    for i := 0; i < b.N; i++ {
        for i := 0; i < it; i++ {
            r = s[i].a
        }
    }
    result = r
}</pre>
```

```
type Struct2 struct {
    a []int32
    b []int64
}
```

[func BenchmarkStructureOfSlices(b *testing.B) {

```
s := Struct2{
    a: make([]int32, it),
    b: make([]int64, it),
var r int32
b.ResetTimer()
for i := 0; i < b.N; i++ {</pre>
    for i := 0; i < it; i++ {</pre>
        r = s.a[i]
result = r
```



```
type Struct1 struct {
    a int32
    b int64
}
```

```
func BenchmarkStructureOfSlices(b *testing.B) {
func BenchmarkSliceOfStructures(b *testing.B) {
                                                          s := Struct2{
    s := make([]Struct1, it)
                                                              a: make([]int32, it),
    var r int32
                                                              b: make([]int64, it),
    b.ResetTimer()
    for i := 0; i < b.N; i++ {</pre>
                                                          var r int32
        for i := 0; i < it; i++ {</pre>
                                                          b.ResetTimer()
            r = s[i].a
                                                          for i := 0; i < b.N; i++ {</pre>
        }
                                                              for i := 0; i < it; i++ {</pre>
                                                                   r = s.a[i]
    result = r
```

```
type Struct2 struct {
    a []int32
    b []int64
}
```

```
י
result = r
```



```
type Struct1 struct {
    a int32
    b int64
}
```

```
func BenchmarkStructureOfSlices(b *testing.B) {
func BenchmarkSliceOfStructures(b *testing.B) {
                                                          s := Struct2{
    s := make([]Struct1, it)
                                                              a: make([]int32, it),
    var r int32
                                                              b: make([]int64, it),
    b.ResetTimer()
    for i := 0; i < b.N; i++ {</pre>
                                                          var r int32
        for i := 0; i < it; i++ {</pre>
                                                          b.ResetTimer()
            r = s[i].a
                                                          for i := 0; i < b.N; i++ {</pre>
        }
                                                              for i := 0; i < it; i++ {</pre>
                                                                  r = s.a[i]
    result = r
                                                          result = r
```

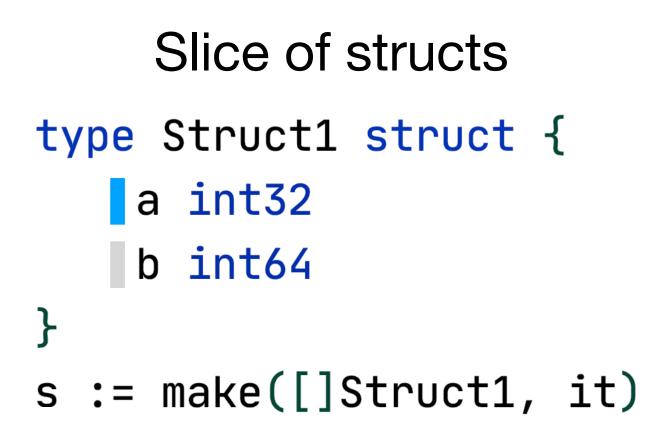
```
type Struct2 struct {
    a []int32
    b []int64
}
```

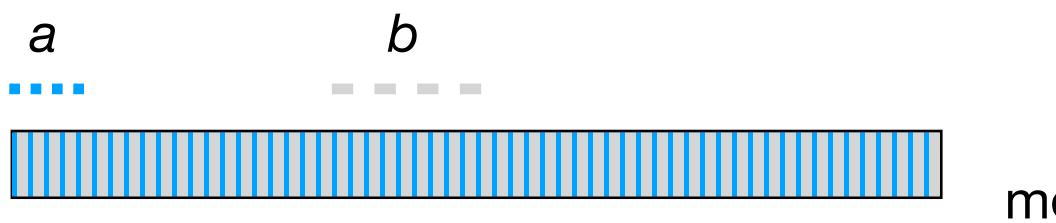


```
Slice of structs
type Struct1 struct {
    a int32
    b int64
}
s := make([]Struct1, it)
```

```
Struct of slices
type Struct2 struct {
    a []int32
    b []int64
}
s := Struct2{
    a: make([]int32, it),
    b: make([]int64, it),
}
```





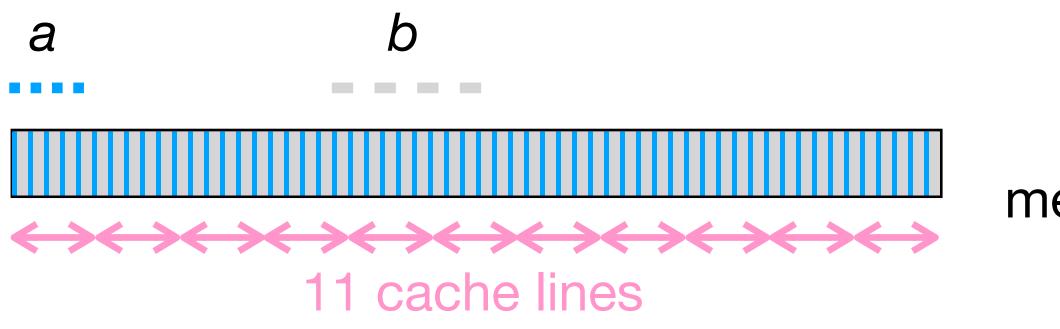


```
Struct of slices
type Struct2 struct {
    a []int32
    b []int64
}
s := Struct2{
    a: make([]int32, it),
    b: make([]int64, it),
}
```

In memory



Slice of structs
type Struct1 struct {
 a int32
 b int64
}
s := make([]Struct1, it)

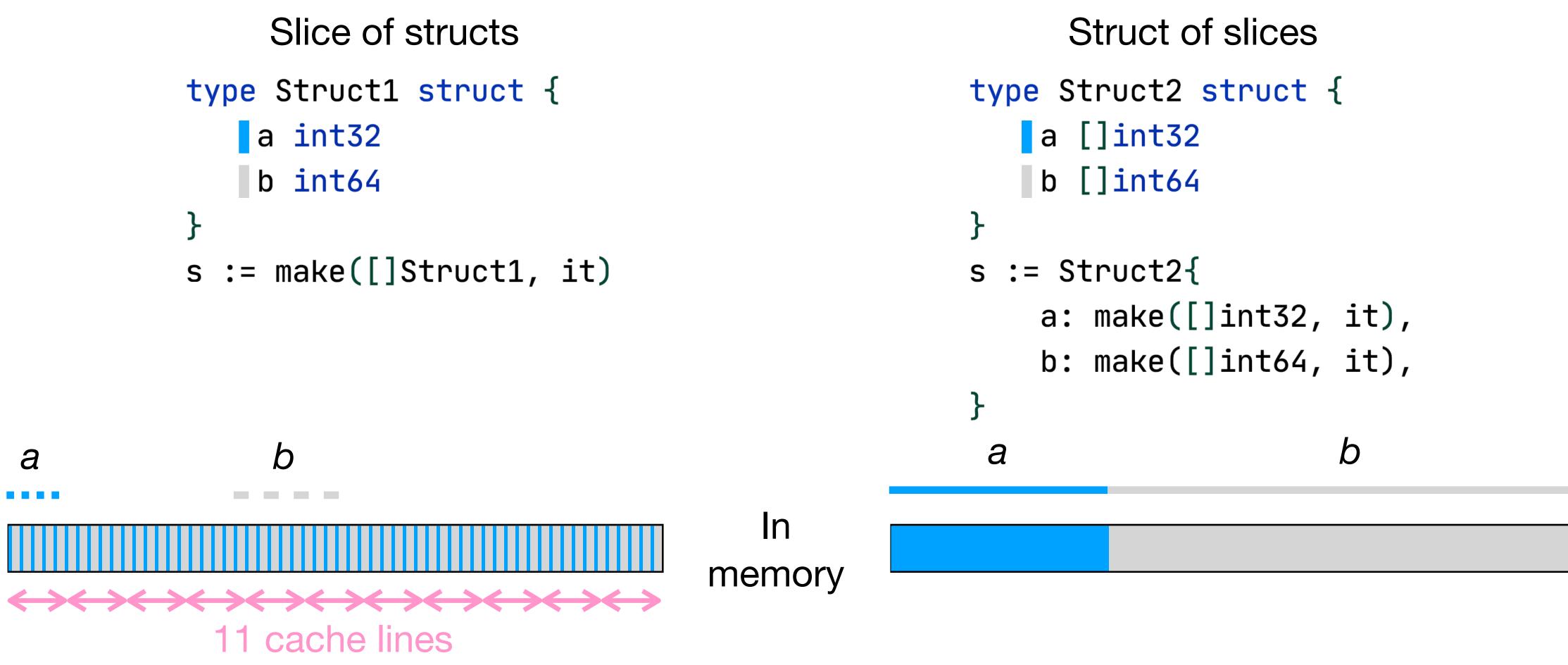


```
Struct of slices
type Struct2 struct {
    a []int32
    b []int64
}
s := Struct2{
    a: make([]int32, it),
    b: make([]int64, it),
}
```

In memory

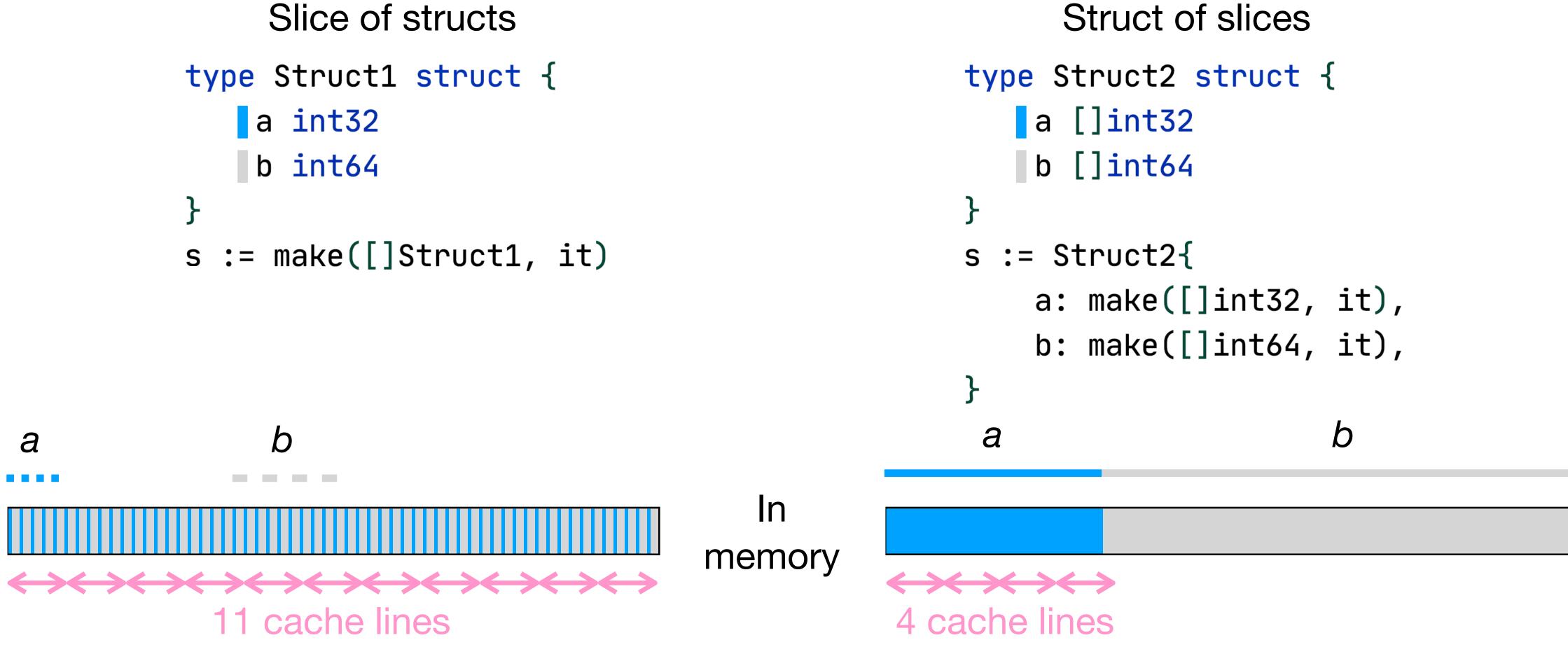


Slice of structs a int32 b int64 }



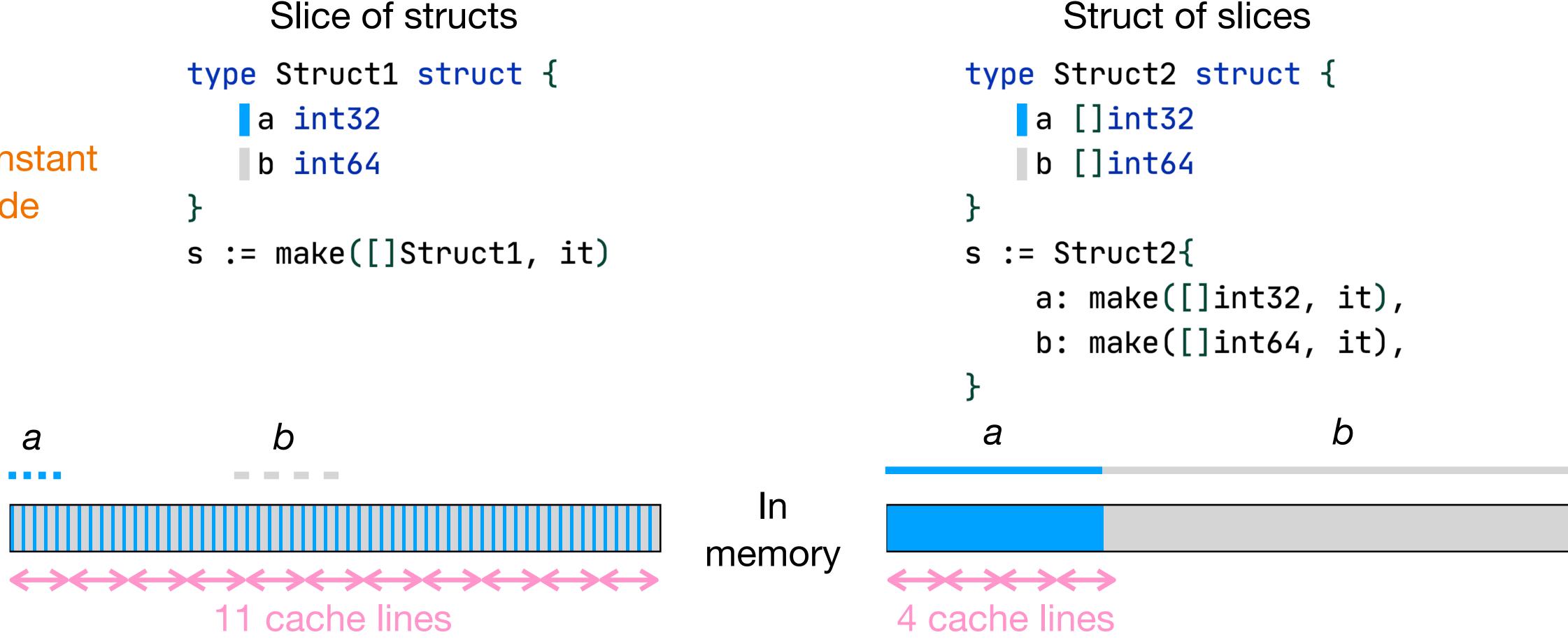


Slice of structs a int32 b int64 }



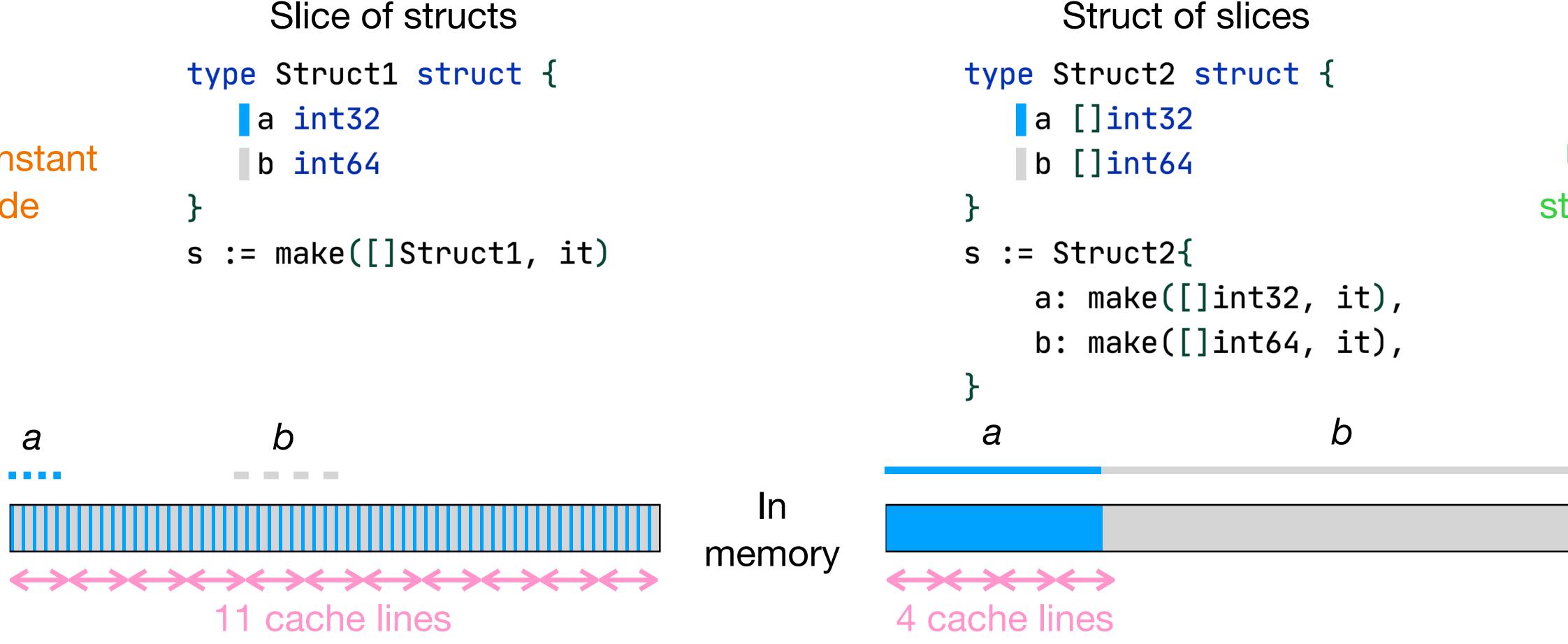


Slice of structs type Struct1 struct { a int32 Constant b int64 stride } s := make([]Struct1, it)



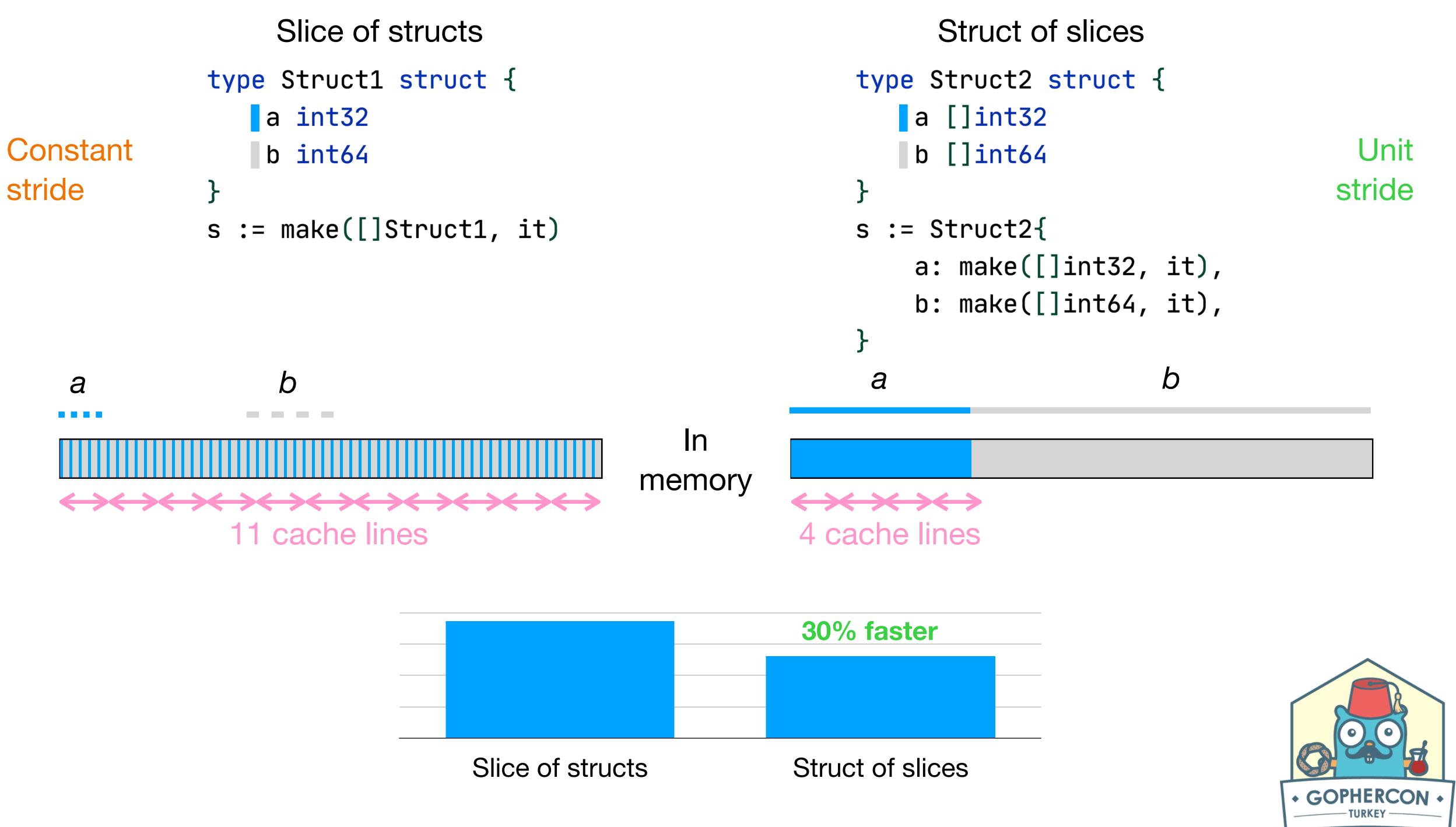


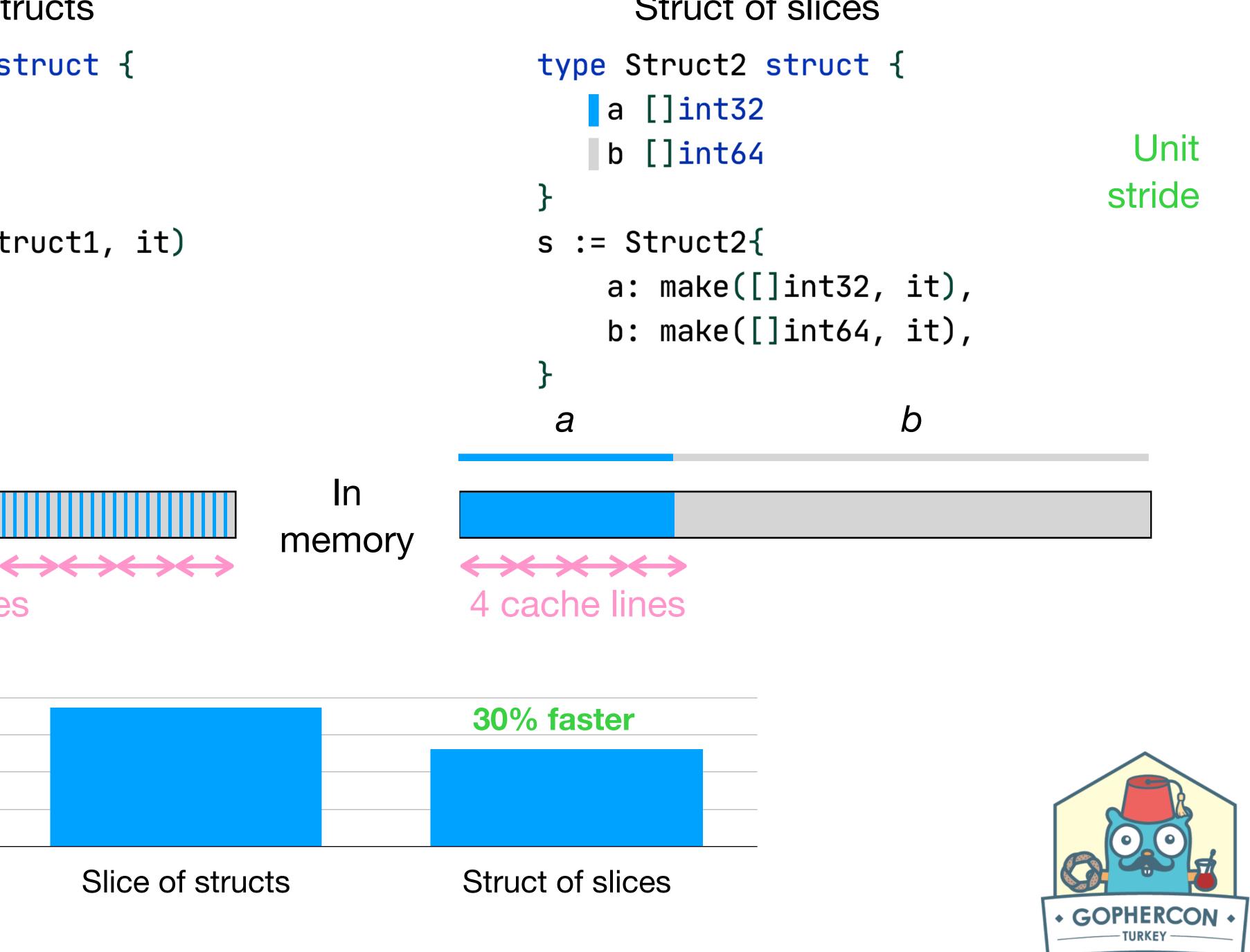
Slice of structs type Struct1 struct { a int32 Constant b int64 stride } s := make([]Struct1, it)





Unit stride







• A concrete example: Go standard flate package



• A concrete example: Go standard flate package

 Flate is a compression algorithm based on two other algorithms: huffman encoding an LZ77 compression



Go flate package

	type hcode struct { code, len uint16												
}	}												
<pre>type huffmanEncoder struct {</pre>													
codes	odes []hcode												
freqcache	[]literalN	lod	е										
bitCount	[17]int32												
lns	byLiteral	//	stored	to	avoid	repeated	allocation	in	gener				
lfs	byFreq	//	stored	to	avoid	repeated	allocation	in	gener				
}													

https://github.com/golang/go: src/compress/flate/huffman_code.go

erate erate

Go flate package

type hcode struct {
 code, len uint16
}

}

type huffmanEncoder struct {

codes	[]hcode								
freqcache	[]literal	Node	е						
bitCount	[17]int32								
lns	byLiteral	//	stored	to	avoid	repeated	allocation	in	gene
lfs	byFreq	//	stored	to	avoid	repeated	allocation	in	gene

https://github.com/golang/go: src/compress/flate/huffman_code.go

erate erate

Go flate package

type hcode struct { code, len uint16

type huffmanEncoder struct {

codes	[]hcode								
freqcache	[]literal	Node	е						
bitCount	[17]int32								
lns	byLiteral	//	stored	to	avoid	repeated	allocation	in	gene
lfs	byFreq	//	stored	to	avoid	repeated	allocation	in	gene

https://github.com/golang/go: src/compress/flate/huffman_code.go

```
type hcodes struct {
           code []uint16
           len []uint16
       }
       type huffmanEncoder struct {
           codes
                     hcodes
           freqcache []literalNode
           bitCount [17]int32
erate
                    byLiteral // stored to avoid repeated allocation in generate
           lns
erate
                               // stored to avoid repeated allocation in generate
           lfs
                    byFreq
```

Go flate package

type hcode struct { code, len uint16

}

type huffmanEncoder struct {

codes	[]hcode	typ	e huffmanEı	ncoder struct {
fregcache	[]literalNode		codes	hcodes
	[17]int32		freqcache	[]literalNode
lns	byLiteral // stored to avoid repeated allocation in generate		bitCount	[17]int32
lfs	byFreq // stored to avoid repeated allocation in generate		lns	byLiteral // stored to avoid repeated allocation in ge
	Syrreq // Stored to avoid repeated attooution in generate		lfs	byFreq // stored to avoid repeated allocation in ge
		}		

https://github.com/golang/go: src/compress/flate/huffman_code.go

Go flate package modified

type <u>hco</u>	des struct	{
code	[]uint16	
len	[]uint16	
}		

type huffmanEncoder struct {	
------------------------------	--

generate generate

Go flate package

	<pre>ype hcode struct { code, len uint16</pre>											
}												
<mark>type</mark> huffmanEr	type huffmanEncoder struct {											
codes	[]hcode											
freqcache	[]literal	lod	е									
bitCount	[17]int32											
lns	byLiteral	//	stored	to	avoid	repeated	allocation	in	genei			
lfs	byFreq	//	stored	to	avoid	repeated	allocation	in	genei			
}												

https://github.com/golang/go: src/compress/flate/huffman_code.go

• 5 iteration loops on either hcode.code or hcode.len

```
type hcodes struct {
           code []uint16
           len []uint16
      }
       type huffmanEncoder struct {
                    hcodes
           codes
          freqcache []literalNode
          bitCount [17]int32
erate
                    byLiteral // stored to avoid repeated allocation in generate
          lns
erate
                              // stored to avoid repeated allocation in generate
           lfs
                    byFreq
```

Go flate package

```
type hcodes struct {
type hcode struct {
                                                                                 code []uint16
    code, len uint16
                                                                                 len []uint16
                                                                             }
type huffmanEncoder struct {
                                                                             type huffmanEncoder struct {
              []hcode
    codes
                                                                                 codes
                                                                                           hcodes
   freqcache []literalNode
                                                                                 freqcache []literalNode
   bitCount [17]int32
                                                                                 bitCount [17]int32
             byLiteral // stored to avoid repeated allocation in generate
   lns
                                                                                           byLiteral // stored to avoid repeated allocation in generate
                                                                                 lns
                       // stored to avoid repeated allocation in generate
              byFreq
   lfs
                                                                                 lfs
                                                                                           byFreq
                                                                                                     // stored to avoid repeated allocation in generate
```

https://github.com/golang/go: src/compress/flate/huffman_code.go

• 5 iteration loops on either hcode.code or hcode.len

• Example:

```
for i := 0; i < numCodegens; i++ {</pre>
    value := uint(w.codegenEncoding.codes[codegenOrder[i]].len)
    w.writeBits(int32(value), nb: 3)
```



Go flate package

```
type hcodes struct {
type hcode struct {
                                                                                 code []uint16
    code, len uint16
                                                                                 len []uint16
type huffmanEncoder struct {
                                                                             type huffmanEncoder struct {
              []hcode
   codes
                                                                                 codes
                                                                                           hcodes
   freqcache []literalNode
                                                                                 freqcache []literalNode
   bitCount [17]int32
                                                                                 bitCount [17]int32
             byLiteral // stored to avoid repeated allocation in generate
   lns
                                                                                           byLiteral // stored to avoid repeated allocation in generate
                                                                                 lns
                       // stored to avoid repeated allocation in generate
   lfs
             byFreq
                                                                                                     // stored to avoid repeated allocation in generate
                                                                                 lfs
                                                                                           byFreq
```

https://github.com/golang/go: src/compress/flate/huffman_code.go

• 5 iteration loops on either hcode.code or hcode.len

• Example:

```
for i := 0; i < numCodegens; i++ {</pre>
for i := 0; i < numCodegens; i++ {</pre>
                                                                          value := uint(w.codegenEncoding.codes.len[codegenOrder[i]])
    value := uint(w.codegenEncoding.codes[codegenOrder[i]].len)
                                                                          w.writeBits(int32(value), nb: 3)
    w.writeBits(int32(value), nb: 3)
                                                                      }
```



Go flate package

```
type hcodes struct {
type hcode struct {
                                                                                 code []uint16
    code, len uint16
                                                                                 len []uint16
type huffmanEncoder struct {
                                                                             type huffmanEncoder struct {
              []hcode
   codes
                                                                                 codes
                                                                                           hcodes
   freqcache []literalNode
                                                                                 freqcache []literalNode
   bitCount [17]int32
                                                                                 bitCount [17]int32
             byLiteral // stored to avoid repeated allocation in generate
   lns
                                                                                           byLiteral // stored to avoid repeated allocation in generate
                                                                                 lns
                       // stored to avoid repeated allocation in generate
   lfs
             byFreq
                                                                                                     // stored to avoid repeated allocation in generate
                                                                                 lfs
                                                                                           byFreq
```

https://github.com/golang/go: src/compress/flate/huffman_code.go

• 5 iteration loops on either hcode.code or hcode.len

• Example:

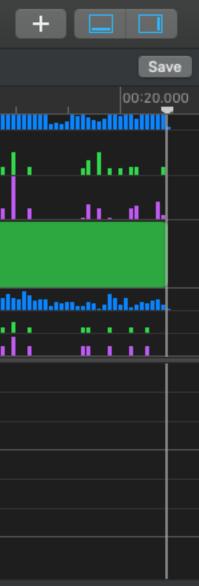
```
for i := 0; i < numCodegens; i++ {</pre>
for i := 0; i < numCodegens; i++ {</pre>
                                                                          value := uint(w.codegenEncoding.codes.len[codegenOrder[i]])
    value := uint(w.codegenEncoding.codes[codegenOrder[i]].len)
                                                                          w.writeBits(int32(value), nb: 3)
    w.writeBits(int32(value), nb: 3)
                                                                      }
```

Metrics?



		🔍 🔍 Ma	cBook P	Pro de teival	n) All	Processes			•
🐨 Tra	ack Filte						All Tracks		
	_					00:00.000		00:05.000	
	Instru	nent j			PEND <u>.</u> REPLA		. 		
1	Therm Instrum	al State		LID.		ent Fair	uttease lassale concettor e	n laataa Etaa a	da arm
	Core 0	idex SMT		L1D_		- بىللىپ بىيە -	hliumhanalhliuaddhaathlaachd 	e de la constante de la constan	
2	Thread flate1 (925) L1D			L1D_	Samp PEND ₋ REPLA				
Main Thread 0x342c Thread flate2 (926)			CPU L1D_	Samp PEND	-···	Liah. Lillii I	<u> </u>	1. 1 1 1. 1	
🙉 Cou	inters)	⊇ Call Tre	e) Cal	ll Tree					
• • • •	Total			Self (ms)		L1D_PEND_MISS.PENDING	L1D.REPLACEMENT	Symbol Name	
			21.9%	0,0			22 634 154 978 442 250 000 000		
	4 2 1	2.0ms 1.0ms	0.0% 0.0% 0.0%	0,0 0,0		36 892 925 199 884 853 000 281 474 072 705 310 10 253	0	 ▶_dyld_start dyld ▶dyld_get_min_os_version ▶runtime.bgscavenge flate 	
		5934.0ms 5932.0ms			_		22 578 814 746 220 986 000 000 22 578 814 746 220 986 000 000	▼runtime.main flate1 ▼main.main flate1	

Т	rack Filte				All Tracks								
				00:00.000		00:05.000		00:10.000		00:15.000			
	Instrum	ment					\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\			llininhlin d.linhl	h	111IIIIIIIII	
			L1D_PEND	2	بالمتحدية المتلاح المالة	I have been and	ala ka ana ara k	I an I an a she sade					ılı .
			L1D.REPL						_				
	Thorm	al State			<u> ditaa kaali sartin i</u>	<u>i laatian litan adr</u>	in a second statistical second se	<u>nantale te card fi carda fa lite</u>	<u>. 11 . 116. I 6</u>	te fat fan de ing	<u>. m. h. l</u>	<u> </u>	<u> </u>
J	Instrum		Curr	rent Fair									
			CPI I Sam	n a sa militat sala	allana sedaha dutta sua sedah	and the second block	111101011.101	halut dati itali dur lati.	alde alline hat t	Unite and there as	l la stra a		I antiballar
	Core 0		L1D_PEND	D		dh.1000.a.1000.dh10001dd000.h.1	an a a a la ca la ca l				le e	http://doc.org/10.004	
	CPU In	idex SMT	L1D.REPL	A				<u></u>		h.			
2	Main T	hread 0x341a	CPU Sam	-									
		d flate1 (925)	L1D_PEND	D	1	մ եվ, մետ սե	1.11.						
			L1D.REPL										
▶_]	Main T	hread 0x342c	CPU Sam	_		<u> u . I il lu Il Il III-il</u>							
		flate2 (926)	L1D_PEND										
		natez (820)	L1D_PEN				hl d h u	սիլի է սուլսոսիի հրե	int in the re-				
				A			bl d I I.	<u>a MML La der Le</u>					
🛞 Co	unters	⊂ Call Tree) Cal	l Tree									ß	
-	Total	Running Time	Self (ms)	L1D_PEND_MISS.PENDING	L1D.REPLACEMENT	Symbol Name			^ Heaviest Sta	ck Trace		—	
	103	- 103.0ms 0.3%	0,0	332 042 519 206 384 200 000	332 041 393 326 658 750 000	►firefox (332)				.0 flate2 (926)			
	5944	5944.0ms 21.9%	0,0	22 634 155 259 928 865 000 0	22 634 154 978 442 250 000 000	▼flate1 (925) 📀				.0 runtime.main			
	4	4.0ms 0.0%		36 892 925 199 884 853 000					4934	.0 main.main			
	2	2.0ms 0.0% 1.0ms 0.0%	0,0 💿 0,0 👤	281 474 072 705 310 10 253		bdyld_get_min_os_version libo >runtime.bgscavenge flate1	iyid.dylib		4905	.0 github.com/teivah/me	chanical-sympathy-i	n-go/cmd/flate2.(*cor	mpressor).clos
	5934	5934.0ms 21.9%			22 578 814 746 220 986 000 000	▼runtime.main flate1			4895	.0 github.com/teivah/me	chanical-sympathy-i	n-go/cmd/flate2.(*cor	mpressor).stor
		5932.0ms 21.9%			22 578 814 746 220 986 000 000	▼main.main flate1				.0 github.com/teivah/me			
	1	1.0ms 0.0%		1 491	0		nical-sympathy-in-go/cmd/flate.(*Writer).			.0 github.com/teivah/me	chanical-sympathy-i	n-go/cmd/flate2.(*huf	fmanEncoder)
		5906.0ms 21.8%			22 523 474 513 998 427 000 000		nical-sympathy-in-go/cmd/flate.(*compre) sort.Sort			
		5900.0ms 21.8% 3083.0ms 11.4%			22 505 027 769 925 007 000 000 12 248 638 064 937 155 000 000		anical-sympathy-in-go/cmd/flate.(*comp chanical-sympathy-in-go/cmd/flate.(*huff) sort.quickSort			
	45			221 360 928 872 791 280 000			echanical-sympathy-in-go/cmd/flate.(*hu		204.) sort.quickSort sort.quickSort			
	26	26.0ms 0.0%	26,0 💶	92 233 157 434 736 660 000	92 233 720 369 331 080 000	github.com/teivah/m	echanical-sympathy-in-go/cmd/flate.(*hu	ffmanBitWriter).writeDynamicHeader	flate1	sort.insertionSort			
				6 124 313 965 952 697 000 000			echanical-sympathy-in-go/cmd/flate.(*hu			3011.1136110113011			
	6 20			18 446 744 068 843 934 000 36 893 206 696 542 073 000			anical-sympathy-in-go/cmd/flate.(*huffm nical-sympathy-in-go/cmd/flate.(*compre		1				
	20	2.0ms 0.0%		281 476 883 776 767	308 653 940		incar-sympathy-in-go/cmu/nate.(*compre						
	1	1.0ms 0.0%	0,0 💿	18 446 462 599 628 040 000		▶thread_start libsystem_pthree	ad.dylib						
	4944	4944.0ms 18.2%			13 982 632 007 873 746 000 000								
	2	2.0ms 0.0% 1.0ms 0.0%	0,0 💿 0,0 💿	18 446 462 598 960 579 000 1 239 265	18 446 744 073 584 247 000 40 778	▶_dyld_start dyld ▶dyld_get_min_os_version libe	tyld dylib						
	1	1.0ms 0.0%		281 474 758 259 731	125 616 097	runtime.asmcgocall flate2	lyid.dyilo						
	4938	4938.0ms 18.2%	4,0 💶	13 964 185 263 827 763 000 0	13 964 185 263 800 037 000 000	▼runtime.main flate2							
	4934	4934.0ms 18.2%			13 945 738 519 726 745 000 000	▼main.main flate2							
	2	2.0ms 0.0% 4905.0ms 18.1%		18 446 462 598 382 694 000	18 446 744 073 570 361 000 13 835 058 055 284 682 000 000		nical-sympathy-in-go/cmd/flate2.(*Writer nical-sympathy-in-go/cmd/flate2.(*compr						
		4895.0ms 18.1%			13 835 058 055 284 682 000 000		anical-sympathy-in-go/cmd/flate2.(*comp						
		2372.0ms 8.7%			6 714 614 842 831 794 000 000		chanical-sympathy-in-go/cmd/flate2.(*hu	· · · ·	•				
	33	33.0ms 0.1%		92 233 720 357 337 330 000	92 233 720 367 897 770 000		echanical-sympathy-in-go/cmd/flate2.(*h		()				
	34 1518	34.0ms 0.1% 1518.0ms 5.6%	34,0 👤 911,0 👤	166 020 133 688 638 500 000 4 039 836 670 803 177 400 000	166 020 696 661 746 650 000 4 039 836 952 149 304 500 000		echanical-sympathy-in-go/cmd/flate2.(*h echanical-sympathy-in-go/cmd/flate2.(*h		flate2				
	4	4.0ms 0.0%		937 650			anical-sympathy-in-go/cmd/flate2.(*huffi		e2				
	20		20,0 💶	73 786 413 347 266 200 000	73 786 976 294 798 580 000		nical-sympathy-in-go/cmd/flate2.(*compr				Xcod	e Instri	IMA
	1	1.0ms 0.0%		450 604	7 391	▶runtime.schedinit flate2							
	1	1.0ms 0.0%	0,0 💩	1 905 519	32 467	▶thread_start libsystem_pthree	ad.dylib						
Input	t Filter	Involves Syml	bol	Call Tree Call Tree Constraints	Data Mining								



close

storeHuff

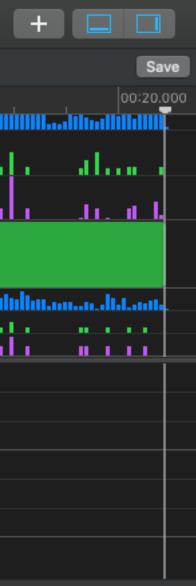
riter).writeBlockHuff der).generate



	MacBook Pro de	teivah 👌 All Pro	ocesses	•
🐨 Tr	ack Filter		All Tracks	
	Instrument	L1D_PEND		
		L1D.REPLA	<u>a sa kasa dhasa tatu wa kasala waxa na kasa kasa kuwa</u>	
1	Thermal State	Current		
	Core 0 CPU Index SMT	L1D_PEND		
2	Main Thread 0x341a	CPU Sampl		
	Thread flate1 (925)	L1D_PEND	http://www.com/action/act	L Lill
2	Main Thread 0x342c	CPU Sampl	ultratertende	-1- 1 1 1- 1
	Thread flate2 (926)	L1D_PEND		
		L1D.REPLA		

٦ 🕲	Track Filter			All Tracks					
			00:00.000		00:05.000		:10.000	00:15.000	
	Instrument					,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,			ta nadullua nadulluan 1000aana0000a nadulluandulluuuuuuuu
		L1D_PEND		alar a sa sa sa sa		the second s			المرابقة المتحدينية والمرابع
		L1D.REPLA	···	والمتعالية والمتعادية	المصالية والمسالية	<u>مية الطبيات في من من مستحد ا</u>	an an an tao tabu dh	and a state of a state of a	mana tana ang kandha kala
	Thermal State								
	Instrument	Curre	ent Fair						
		0011.0				Tala seconda ter a su a ta atantia int	n dala at la lana dia dia	attle la la clas de la composición de	
	Core 0	CPU Samp	» I	dillididə ərə Həlmədillidə ədilə	kanadahan. Malamahahah Musa.	.101	handillaadiihaa.iiihaadiihaadiihaa	WWw.WW.WWWWW.with	հեռ մեննուեւհ,Ոհաւհոննինիստ
	CPU Index SMT	LID_PEND		a called a charle		and the second	an an an barbarbarbarbarbarbarbarbarbarbarbarbarb		and a state of the second s
≥-	Main Thread 0x341a	CPU Samp	ol						
	Thread flate1 (925)	L1D_PEND	[daaraa baalaa ah	المصادية وأرارات	. Inden in			
		L1D.REPLA			<u> </u>				
▶-	Main Thread 0x342c	CPU Samp	ol						
	Thread flate2 (926)	L1D_PEND							
		L1D.REPLA							
			<u> </u>			ht_d_t	When a she had been been been been been been been bee		
🙆 Co	ounters) 🔁 Call Tree) Ca	all Tree							()
	Total Running Time		L1D_PEND_MISS.PENDING	L1D.REPLACEMENT			^	Heaviest Stack Trace	
	103 103.0ms 0.3%		332 042 519 206 384 200 000					4944.0 flate2 (926)	
	5944 5944.0ms 21.99 4 4.0ms 0.09		22 634 155 259 928 865 000 0 36 892 925 199 884 853 000	22 634 154 978 442 250 000 000 36 893 488 147 244 220 000				4938.0 runtime.main	
	2 2.0ms 0.09		281 474 072 705 310		▶_dyld_start dyld ▶dyld_get_min_os_version lib	dvld dvlib		1934.0 main.main	
	1 1.0ms 0.09		10 253		▶runtime.bgscavenge flate1	ayatayib		4905.0 github.com/teivah/med	chanical-sympathy-in-go/cmd/flate2.(*compressor).clos
	5934 5934.0ms 21.9%			22 578 814 746 220 986 000 000				4895.0 github.com/teivah/med	chanical-sympathy-in-go/cmd/flate2.(*compressor).stor
	5932 5932.0ms 21.9%	% 5,0 💶	22 578 815 309 178 397 000 0	22 578 814 746 220 986 000 000	▼main.main flate1				chanical-sympathy-in-go/cmd/flate2.(*huffmanBitWriter
	1 1.0ms 0.0%		1 491			nical-sympathy-in-go/cmd/flate.(*Writer).Reset			chanical-sympathy-in-go/cmd/flate2.(*huffmanEncoder)
	5906 5906.0ms 21.8%			22 523 474 513 998 427 000 000	<u> </u>	nical-sympathy-in-go/cmd/flate.(*compressor).		1 445.0 sort.Sort	
	5900 5900.0ms 21.89 3083 3083.0ms 11.49			22 505 027 769 925 007 000 000 12 248 638 064 937 155 000 000		hanical-sympathy-in-go/cmd/flate.(*compressor		436.0 sort.quickSort	
	45 45.0ms 0.19		221 360 928 872 791 280 000			chanical-sympathy-in-go/cmd/flate.(*huffmanBi nechanical-sympathy-in-go/cmd/flate.(*huffman		204.0 sort.quickSort	
	26 26.0ms 0.09		92 233 157 434 736 660 000		· ·	nechanical-sympathy-in-go/cmd/flate.(*huffman		63.0 sort.quickSort	
	1614 1614.0ms 5.9%	· _	6 124 313 965 952 697 000 000		· · ·	nechanical-sympathy-in-go/cmd/flate.(*huffman		31.0 sort.insertionSort	
	6 6.0ms 0.0%	% 4,0 💶	18 446 744 068 843 934 000	18 446 744 073 418 738 000	▶github.com/teivah/mecl	hanical-sympathy-in-go/cmd/flate.(*huffmanBit\	Writer).writeStoredHeader flate1		
	20 20.0ms 0.0%		36 893 206 696 542 073 000		- · ·	nical-sympathy-in-go/cmd/flate.(*compressor).	write flate1		
	2 2.0ms 0.0%		281 476 883 776 767						
	1 1.0ms 0.0%		18 446 462 599 628 040 000 13 982 632 007 901 488 000 0	18 446 744 073 615 712 000 13 982 632 007 873 746 000 000		ead.dyllb			
	2 2.0ms 0.09		18 446 462 598 960 579 000						
	1 1.0ms 0.0%		1 239 265			dyld.dylib			
	1 1.0ms 0.0%	% 1,0 💶	281 474 758 259 731	125 616 097	runtime.asmcgocall flate2				
	4938 4938.0ms 18.29	· _		13 964 185 263 800 037 000 000	▼runtime.main flate2				
	4934 4934.0ms 18.29			13 945 738 519 726 745 000 000	▼main.main flate2		-1 4-1-0		
	2 2.0ms 0.0% 4905 4905.0ms 18.1%		18 446 462 598 382 694 000	18 446 744 073 570 361 000 13 835 058 055 284 682 000 000		nical-sympathy-in-go/cmd/flate2.(*Writer).Rese			
	4895 4895.0ms 18.19			13 835 058 055 284 682 000 000		nical-sympathy-in-go/cmd/flate2.(*compressor) hanical-sympathy-in-go/cmd/flate2.(*compressor)			
	2372 2372.0ms 8.79			6 714 614 842 831 794 000 000		chanical-sympathy-in-go/cmd/flate2.(*huffman			
	33 33.0ms 0.1%		92 233 720 357 337 330 000			nechanical-sympathy-in-go/cmd/flate2.(*huffma			
	34 34.0ms 0.19		166 020 133 688 638 500 000		· · · ·	nechanical-sympathy-in-go/cmd/flate2.(*huffma			
	1518 1518.0ms 5.6%		4 039 836 670 803 177 400 000			nechanical-sympathy-in-go/cmd/flate2.(*huffma			
	4 4.0ms 0.0%		937 650			hanical-sympathy-in-go/cmd/flate2.(*huffmanBi			
	20 20.0ms 0.0%		73 786 413 347 266 200 000		· · ·	nical-sympathy-in-go/cmd/flate2.(*compressor)).write flate2		Xcode Instrume
	1 1.0ms 0.0% 1 1.0ms 0.0%	_	450 604		Fruntime.schedinit flate2 thread_start libsystem_pthr	ead dylib			
	1 1.0ms 0.09	% 0,0 💿	1 905 519	32 467	Funeau_start ibsystem_pthr	Cauluyiib			

Call Tree Call Tree Constraints Data Mining



close

storeHuff

riter).writeBlockHuff der).generate

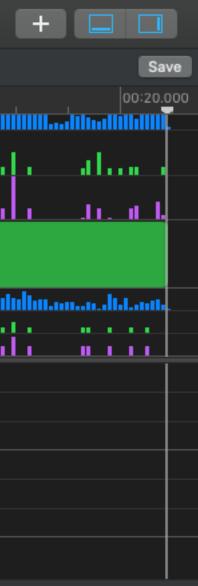


	MacBook Pro de	e teivah $ angle$ All Pro	cesses						•
🗑 Tra	ack Filter				All Tracks				
			00:00.000		l		00:05.000		
	Instrument	L1D_PEND				. In 1	I. I	1	n in
		L1D.REPLA	<u></u>			1.1	1	المتعالم	
(1)	Thermal State	Current							
	Core 0 CPU Index SMT	CPU Sampl L1D_PEND L1D.REPLA	0	a constraint for	and the second second	dia an		and a star	
2-	Main Thread 0x341a	CPU Sampl							
	Thread flate1 (925)	L1D_PEND		http://www.ice			at 1.	al a lla sur	l. 1 i.l.l
		L1D.REPLA		ult cate.					
>_	Main Thread 0x342c	CPU Sampl							
	Thread flate2 (926)	L1D_PEND							
		L1D.REPLA							

🐨 Trac	k Filter				All Tracks					
				00:00.000		00:05.000		00:10.000	00:15.000	
(Instrument]						\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\		la nadulla nadullana dullanadulla nadullanadullululululan	
			L1D_PEN	₽	den ser en	1	and the second			المرابقة المتحدينية والمرابع
			L1D.REPL	A	والمتعالية والمتعارية والمتعادية	يلي منا المراجعات	والمتعادية والمتعادية والمتعادية والمتعادية والمتعادية والمتعاد والمتعاد والمتعاد والمتعاد والمتعاد والمتعاد و	<u>ana mantina tahun b</u>	and a set at the large set	mana tana ang karal kalan ing ka
() ·	Thermal State									
· ·	Instrument		Cur	rent Fair						
	Core O		CPU Sam	¹⁰¹ II	illilililillinullillilill	analidhaa IIIItan dobbhlilhana	111111111111	dillaadillaadillaa.ilib.aadillaa.ilibtaa	1111m.1111.111mlhmth11.m	հեռ հետ հայտարեր հայտարեն հայտարեն
(CPU Index SMT		LID_PEN		a calle a challe a		and the last the second se			in a star and a star
								terne en erel elle elle		
Main Thread 0x341a		CPU Sam	ווקו							
[Thread flate1 (925)	ad flate1 (925)		D	tuara ta duna a a	lan an a' de la ann a bhliain				
				-A		I have a second to the second se				
2	Main Thread 0x34	read 0x342c		ıpl						
۱	Thread flate2 (926	;)	L1D_PEN	D						
			L1D.REPL							
							ht d.t.t.s.			
🐼 Coun	ters) 三 Call Tre	e) Call T	Tree							()
•										
Т		g Time So		L1D_PEND_MISS.PENDING	L1D.REPLACEMENT			^	Heaviest Stack Trace	
	103 103.0ms		0,0		332 041 393 326 658 750 000				4944.0 flate2 (926)	
	5944 5944.0ms 4 4.0ms	21.9% 0.0%	0,0 0,0 💿		22 634 154 978 442 250 000 000 36 893 488 147 244 220 000				1 4938.0 runtime.main	
		0.0%	0,0 🔹	281 474 072 705 310	93 445 947		vld.dvlib		1 4934.0 main.main	
		0.0%	0,0 💶	10 253	0	▶runtime.bgscavenge flate1	,,			hanical-sympathy-in-go/cmd/flate2.(*compressor).clos
	5934 5934.0ms	21.9%		22 578 815 309 178 397 000 0	22 578 814 746 220 986 000 000	▼runtime.main flate1				hanical-sympathy-in-go/cmd/flate2.(*compressor).sto
	5932 5932.0ms			22 578 815 309 178 397 000 0	22 578 814 746 220 986 000 000	▼main.main flate1				hanical-sympathy-in-go/cmd/flate2.(*huffmanBitWriter
		0.0%	1,0 👤	1 491	0		ical-sympathy-in-go/cmd/flate.(*Writer).Re			hanical-sympathy-in-go/cmd/flate2.(*huffmanEncoder
	5906 5906.0ms 5900 5900.0ms			22 523 475 358 404 820 000 0 22 505 028 614 335 974 000 0		· · ·	lical-sympathy-in-go/cmd/flate.(*compress		1445.0 sort.Sort	
	3083 3083.0ms			12 248 628 494 795 629 000 0			anical-sympathy-in-go/cmd/flate.(*compres hanical-sympathy-in-go/cmd/flate.(*huffma		436.0 sort.quickSort	
	45 45.0ms		45,0 1		221 360 928 883 581 350 000		echanical-sympathy-in-go/cmd/flate.(*huffr		204.0 sort.quickSort	
	26 26.0ms	0.0%	26,0 💶	92 233 157 434 736 660 000	92 233 720 369 331 080 000	· · ·	echanical-sympathy-in-go/cmd/flate.(*huffr		63.0 sort.quickSort	
	1614 1614.0ms		933,0 👤	6 124 313 965 952 697 000 000	6 124 319 032 469 608 000 000		echanical-sympathy-in-go/cmd/flate.(*huffr	· · ·	31.0 sort.insertionSort	
		0.0%	4,0 💶	18 446 744 068 843 934 000	18 446 744 073 418 738 000		anical-sympathy-in-go/cmd/flate.(*huffman			
	20 20.0ms	0.0% 0.0%	20,0 💶	36 893 206 696 542 073 000 281 476 883 776 767	36 893 488 148 606 350 000 308 653 940		ical-sympathy-in-go/cmd/flate.(*compress	or).write flate1		
		0.0%	0,0 👤 0,0 🐼	18 446 462 599 628 040 000	18 446 744 073 615 712 000		ad dylib			
	4944 4944.0ms		0,0		13 982 632 007 873 746 000 000	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	a a a a a a a a a a a a a a a a a a a			
	2 2.0ms	: 0.0%	0,0 💿	18 446 462 598 960 579 000						
		0.0%	0,0 💿	1 239 265	40 778	►dyld_get_min_os_version libe	yld.dylib			
		0.0%	1,0 💶	281 474 758 259 731	125 616 097	runtime.asmcgocall flate2				
	4938 4938.0ms 4934 4934.0ms		4,0 👤 7,0 👤		13 964 185 263 800 037 000 000 13 945 738 519 726 745 000 000	▼runtime.main flate2 ▼main.main flate2				
		0.0%	1,0 💻	18 446 462 598 382 694 000	18 446 744 073 570 361 000		nical-sympathy-in-go/cmd/flate2.(*Writer).R	teset flate2		
	4905 4905.0ms		6,0 💶	13 835 058 899 743 300 000 0			ical-sympathy-in-go/cmd/flate2.(*compres			
	4895 4895.0ms	18.1%	2523,0 👤	13 835 058 899 743 300 000 0	13 835 058 055 284 682 000 000	▼ github.com/teivah/mech	anical-sympathy-in-go/cmd/flate2.(*compre	essor).storeHuff flate2		
	2372 2372.0ms			6 714 611 183 731 171 000 000			hanical-sympathy-in-go/cmd/flate2.(*huffn			
	33 33.0ms		33,0 💶	92 233 720 357 337 330 000	92 233 720 367 897 770 000		echanical-sympathy-in-go/cmd/flate2.(*huf			
	34 34.0ms 1518 1518.0ms		34,0 👤 911,0 👤	166 020 133 688 638 500 000 4 039 836 670 803 177 400 000	166 020 696 661 746 650 000 4 039 836 952 149 304 500 000		echanical-sympathy-in-go/cmd/flate2.(*huf echanical-sympathy-in-go/cmd/flate2.(*huf	fmanBitWriter).writeDynamicHeader flate2		
		0.0%	4,0 💶	937 650	4 039 836 952 149 304 500 000 47 947		anical-sympathy-in-go/cmd/flate2.(*huffma			
	20 20.0ms		20,0 💶	73 786 413 347 266 200 000	73 786 976 294 798 580 000		nical-sympathy-in-go/cmd/flate2.(*compres			Xcode Instrume
		0.0%	0,0 💶	450 604	7 391	▶runtime.schedinit flate2				Acouc instrume
	1 1.0ms	0.0%	0,0 💿	1 905 519	32 467	▶thread_start libsystem_pthree	ad.dylib			
Input Ei	lter 🕞 Involv			Call Tree Call Tree Constraints	Data Mining					

Call Tree Call Tree Constraints Data Mining

Input Filter 💿 Involves Symbo



close

storeHuff

riter).writeBlockHuff der).generate



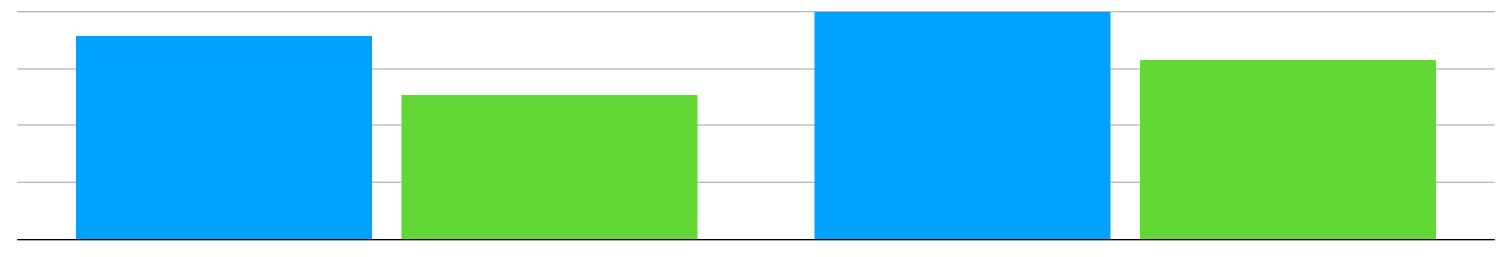
Go flate package

```
type hcodes struct {
type hcode struct {
                                                                                code []uint16
    code, len uint16
                                                                                len []uint16
                                                                            }
type huffmanEncoder struct {
                                                                            type huffmanEncoder struct {
              []hcode
    codes
                                                                                          hcodes
                                                                                codes
   freqcache []literalNode
                                                                                freqcache []literalNode
   bitCount [17]int32
                                                                                bitCount [17]int32
             byLiteral // stored to avoid repeated allocation in generate
   lns
                                                                                          byLiteral // stored to avoid repeated allocation in generate
                                                                                lns
                       // stored to avoid repeated allocation in generate
   lfs
             byFreq
                                                                                                    // stored to avoid repeated allocation in generate
                                                                                lfs
                                                                                          byFreq
```



Go flate package

<pre>type hcode struct { code, len uint16</pre>												
}												
<pre>type huffmanEncoder struct {</pre>												
	codes	des []hcode										
	freqcache	[]literalNode										
	bitCount	itCount [17]int32										
	lns	byLiteral	//	stored	to	avoid	repeated	allocation	in	gener		
	lfs	byFreq	//	stored	to	avoid	repeated	allocation	in	gener		
}												



Encode/Digits/Huffman/1e6

Slice of structs

Go flate package modified

```
type hcodes struct {
           code []uint16
          len []uint16
      }
      type huffmanEncoder struct {
                    hcodes
           codes
          freqcache []literalNode
          bitCount [17]int32
erate
                    byLiteral // stored to avoid repeated allocation in generate
          lns
erate
                               // stored to avoid repeated allocation in generate
          lfs
                    byFreq
```

Encode/Newton/Huffman/1e6



Struct of slices











• I can design algorithms to leverage CPU caches





• I can design algorithms to leverage CPU caches

• I can also organise my data to get the most value out of cache lines





• I can design algorithms to leverage CPU caches

I can also organise my data to get the most value out of cache lines

• **Unit stride** > Constant stride > Non-unit stride



CPU Architecture Locality of Reference **Data-Oriented Design** Caching Pitfall Concurrency





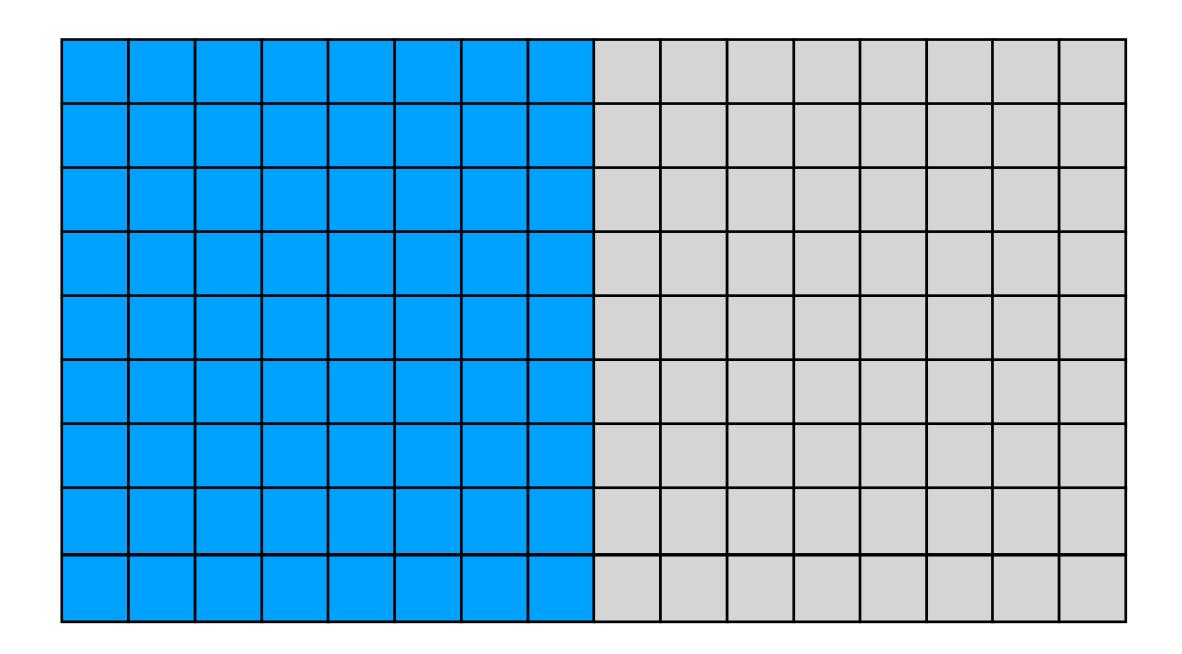
Two-dimensional array of int64s
 64 bytes cache line (8 elements)

-		-					



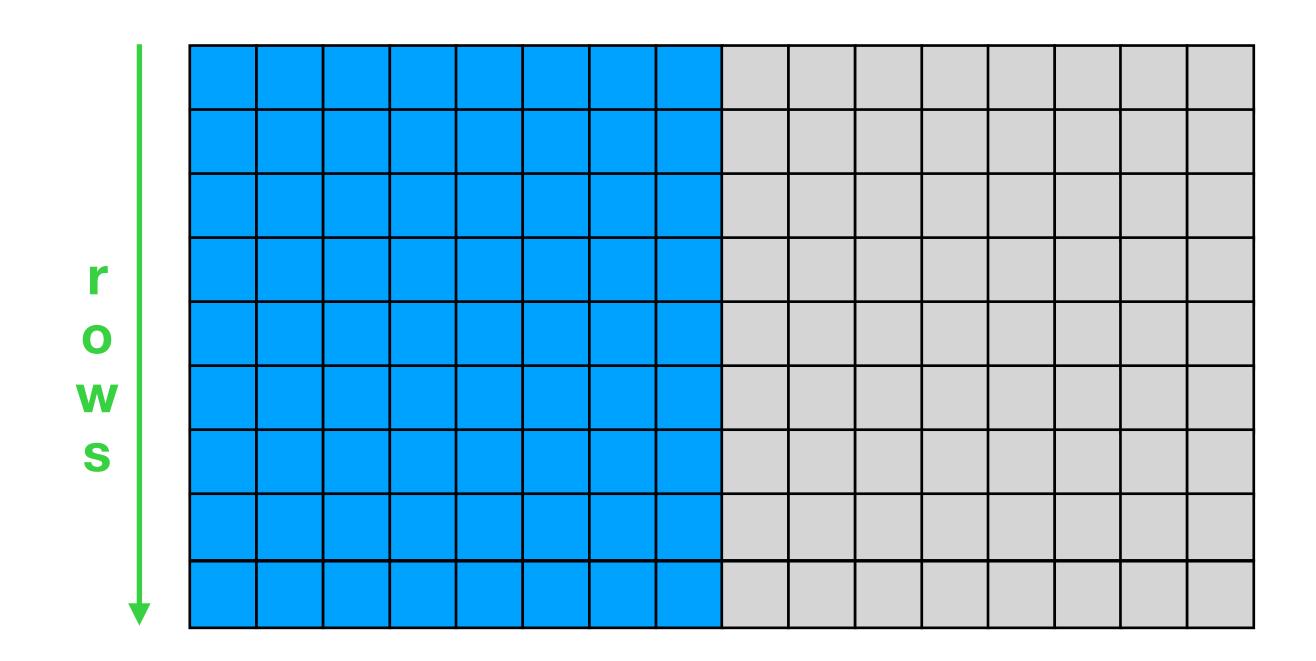


Two-dimensional array of int64s
 64 bytes cache line (8 elements)



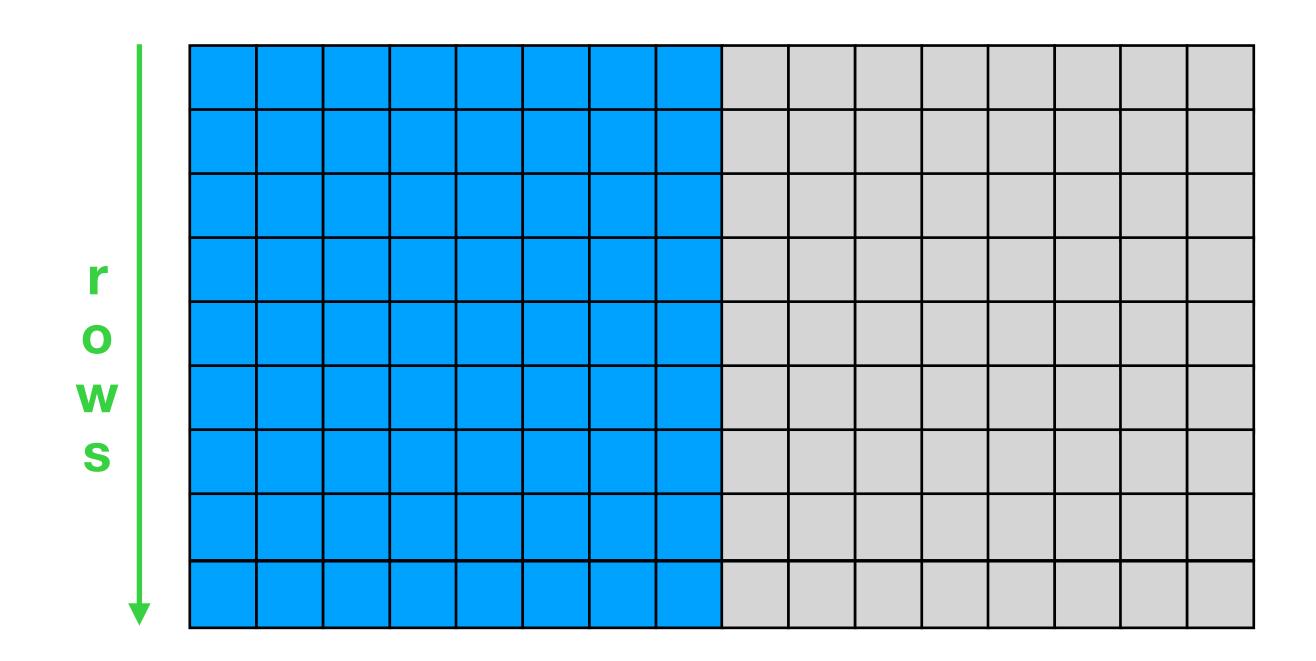


Two-dimensional array of int64s
 64 bytes cache line (8 elements)



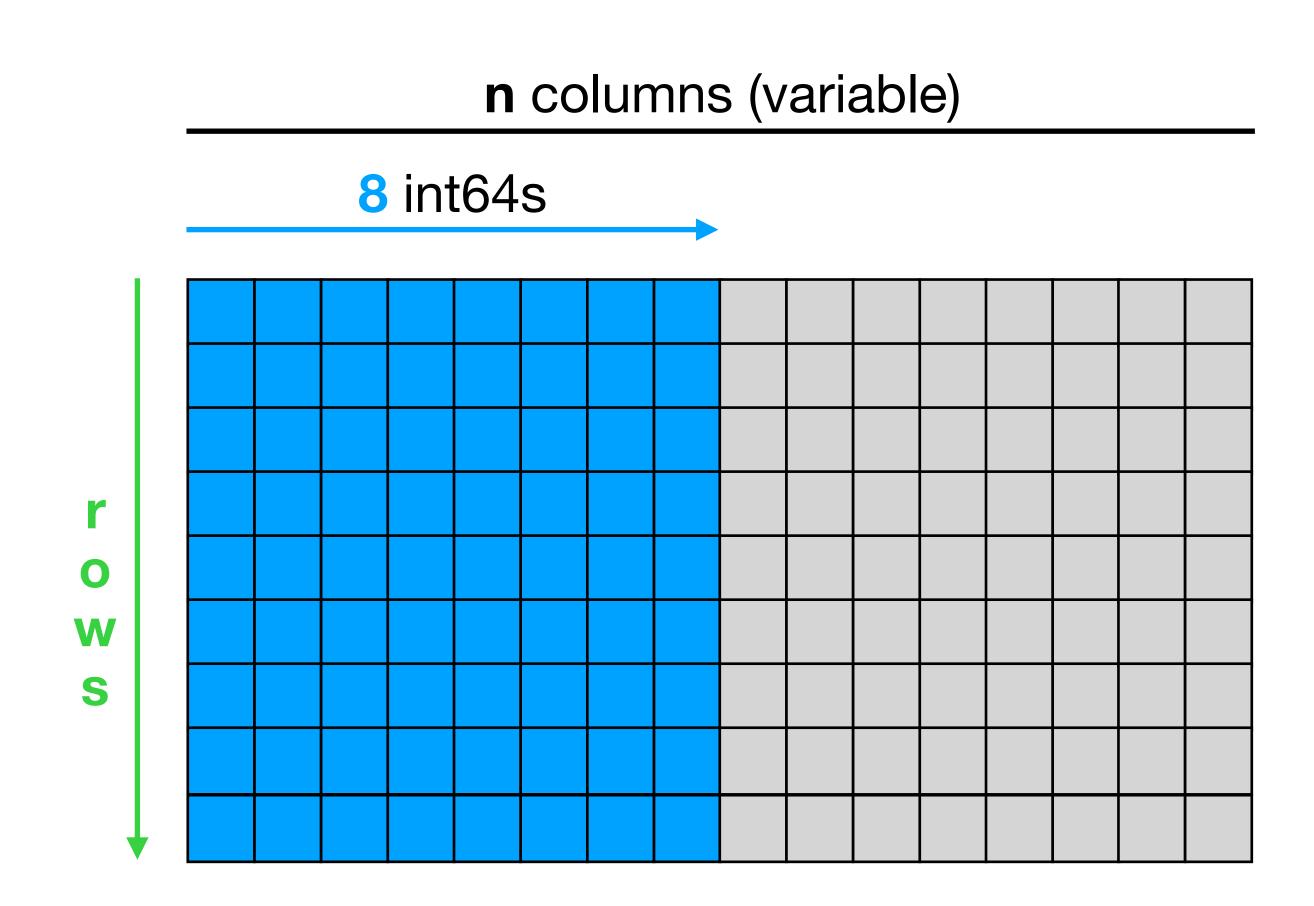


- Two-dimensional array of int64s
 64 bytes cache line (8 elements)
- Traverse each row multiple times the first
 8 columns only



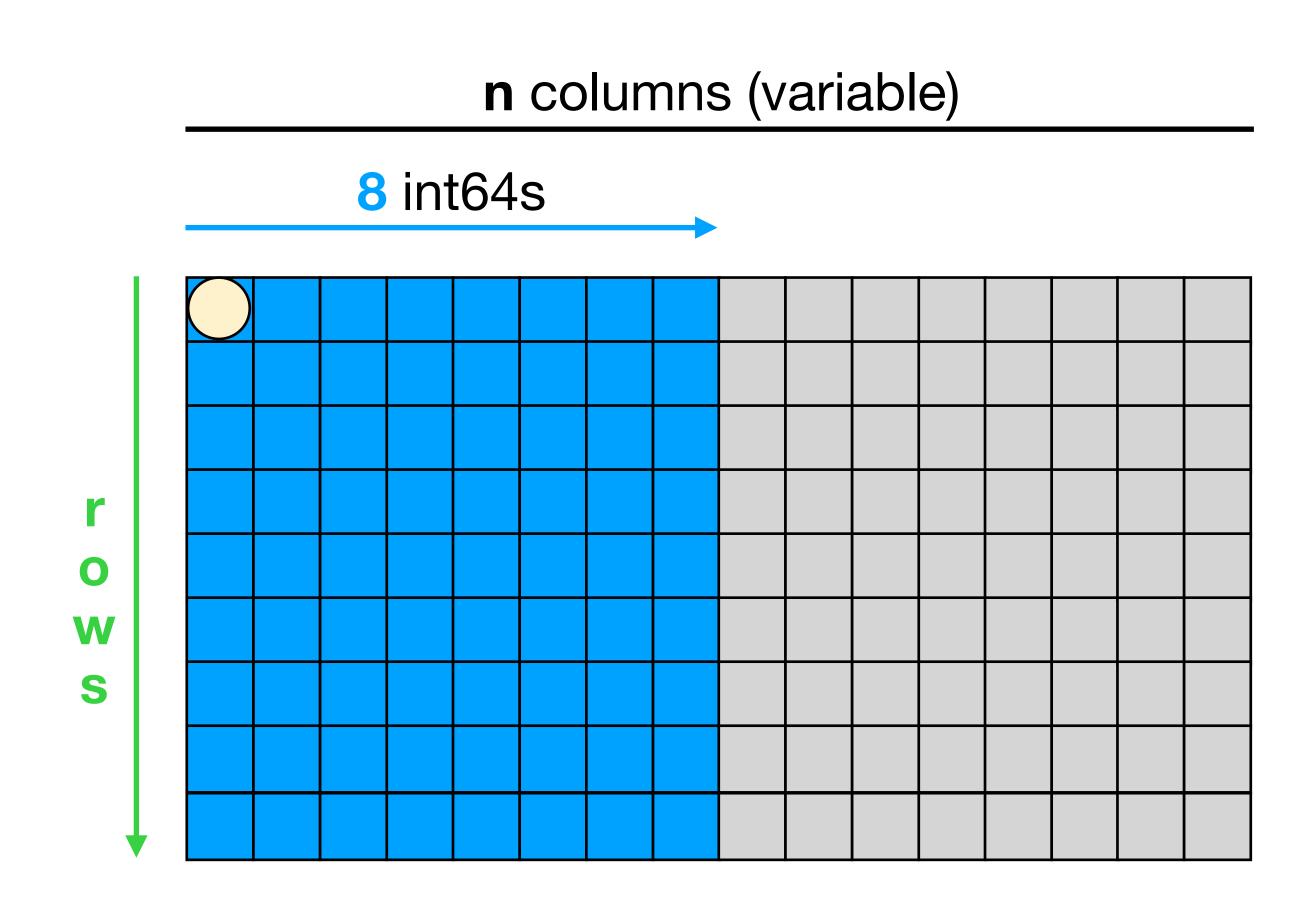


- Two-dimensional array of int64s
 64 bytes cache line (8 elements)
- Traverse each row multiple times the first
 8 columns only



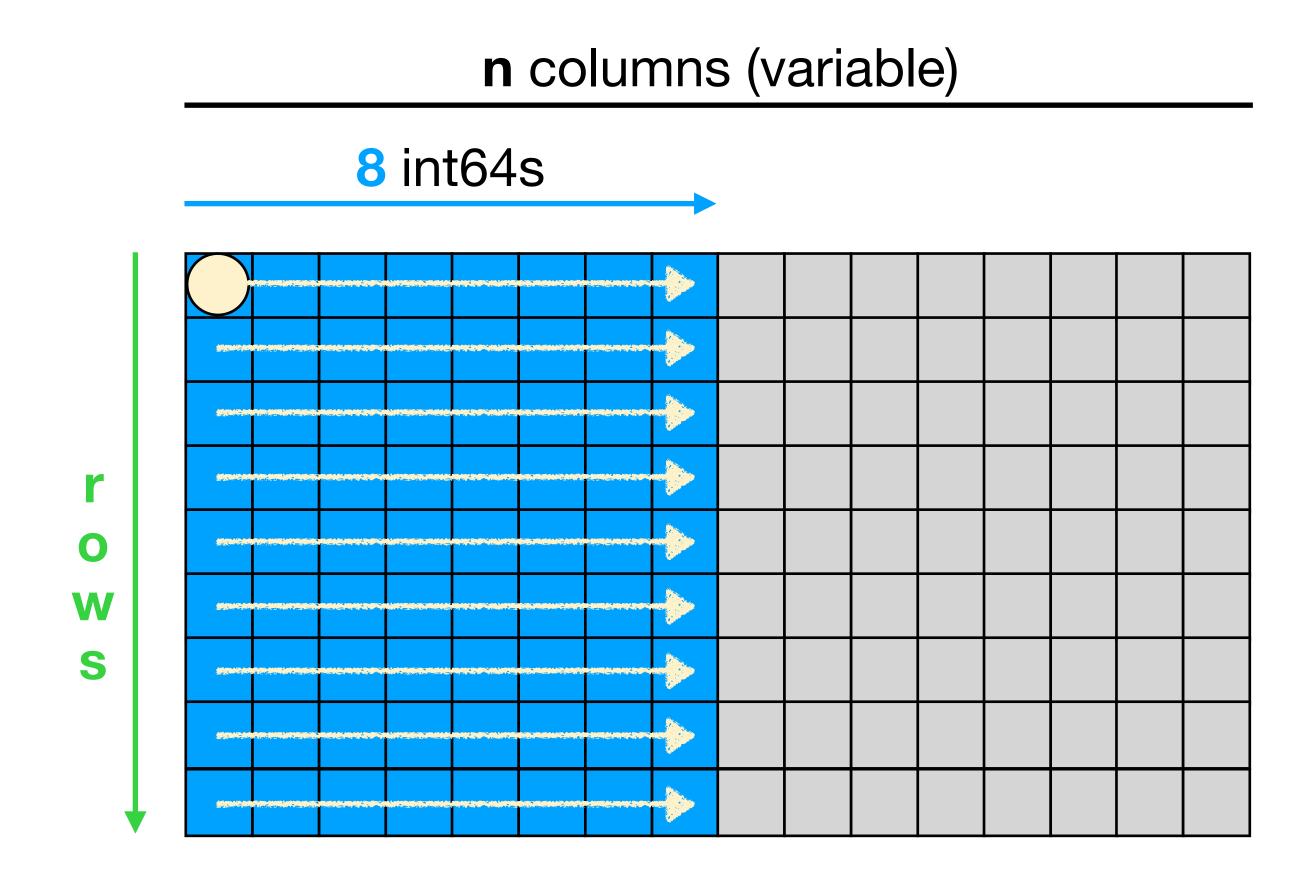


- Two-dimensional array of int64s
 64 bytes cache line (8 elements)
- Traverse each row multiple times the first
 8 columns only



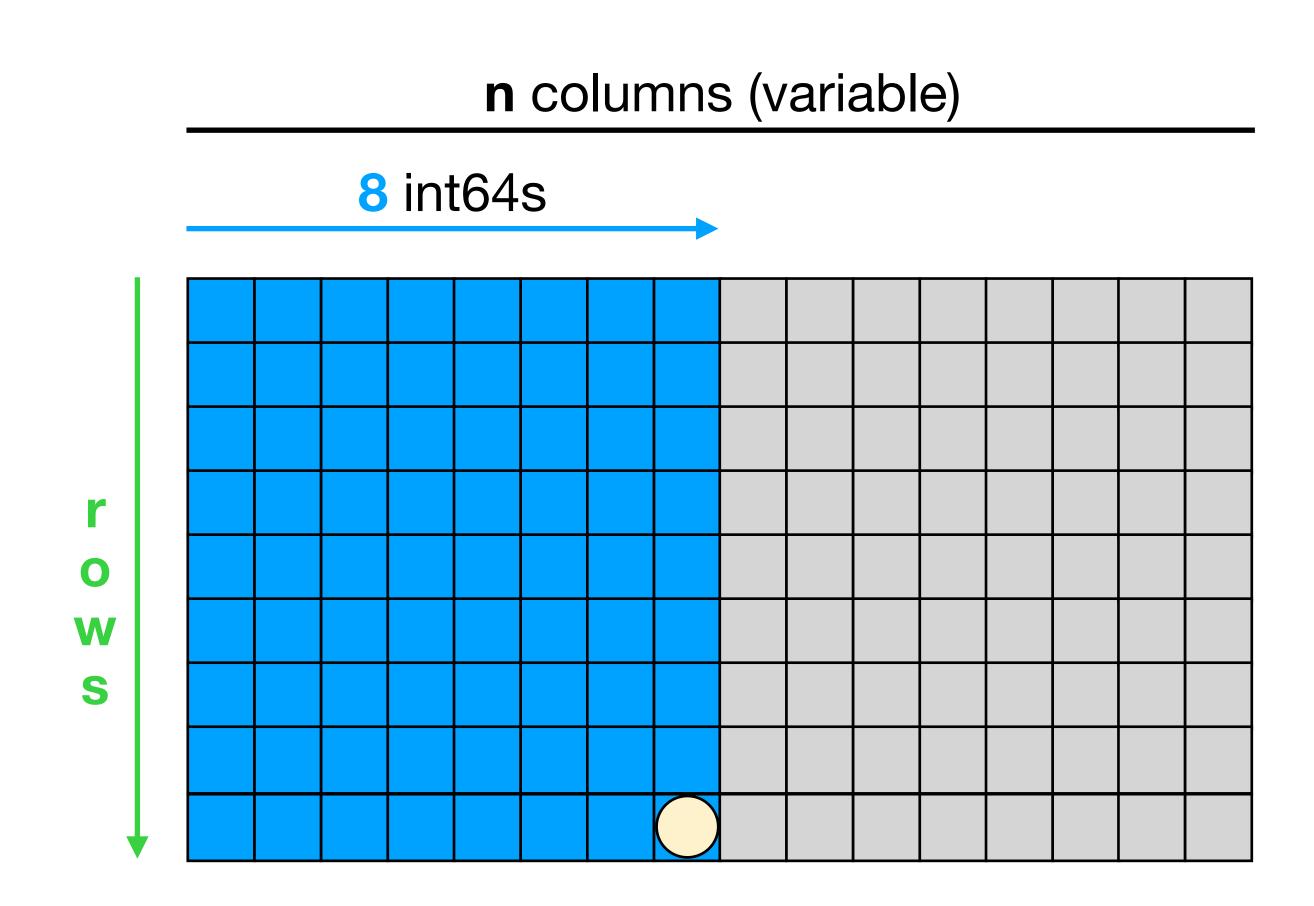


- Two-dimensional array of int64s
 64 bytes cache line (8 elements)
- Traverse each row multiple times the first
 8 columns only



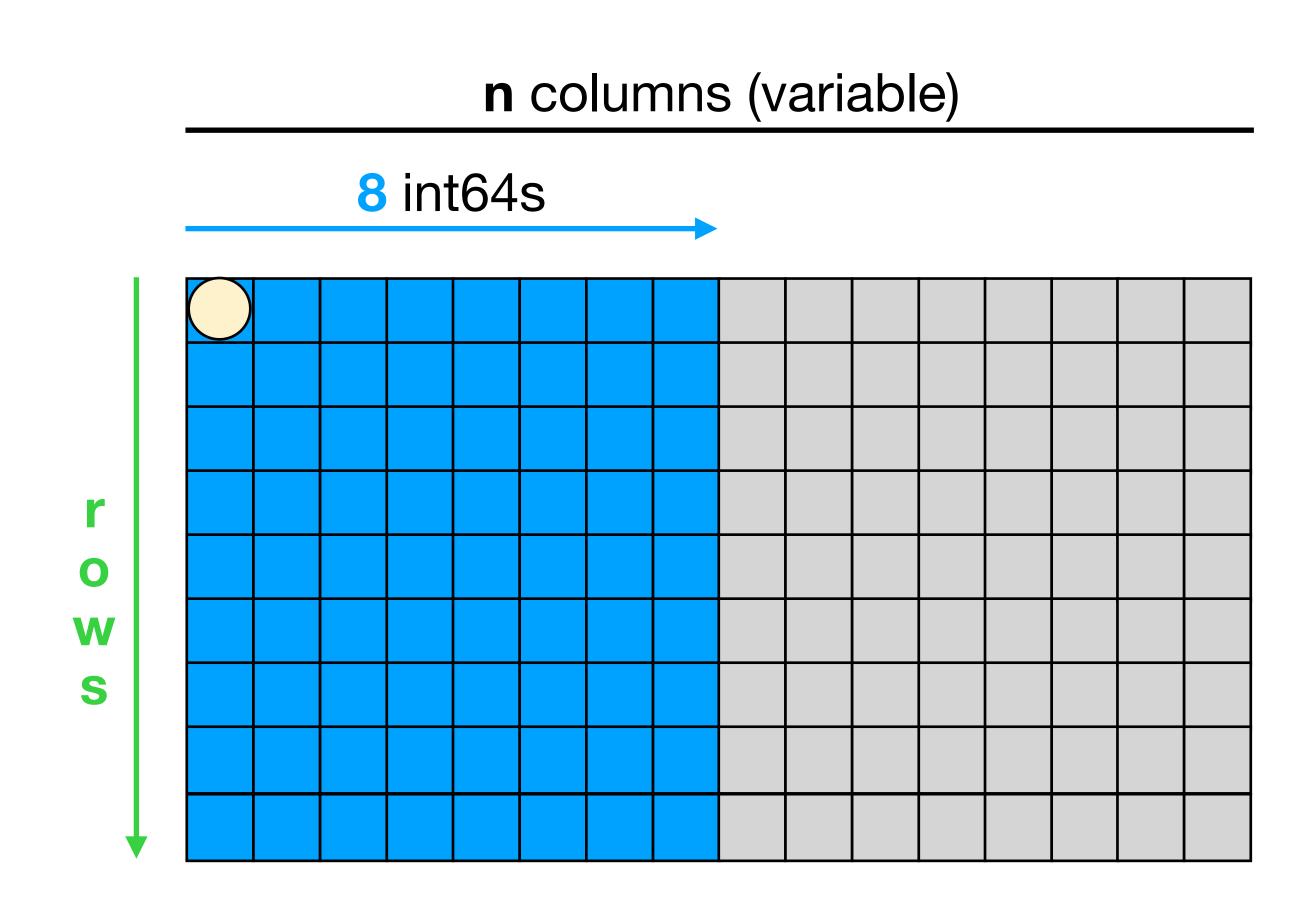


- Two-dimensional array of int64s
 64 bytes cache line (8 elements)
- Traverse each row multiple times the first
 8 columns only





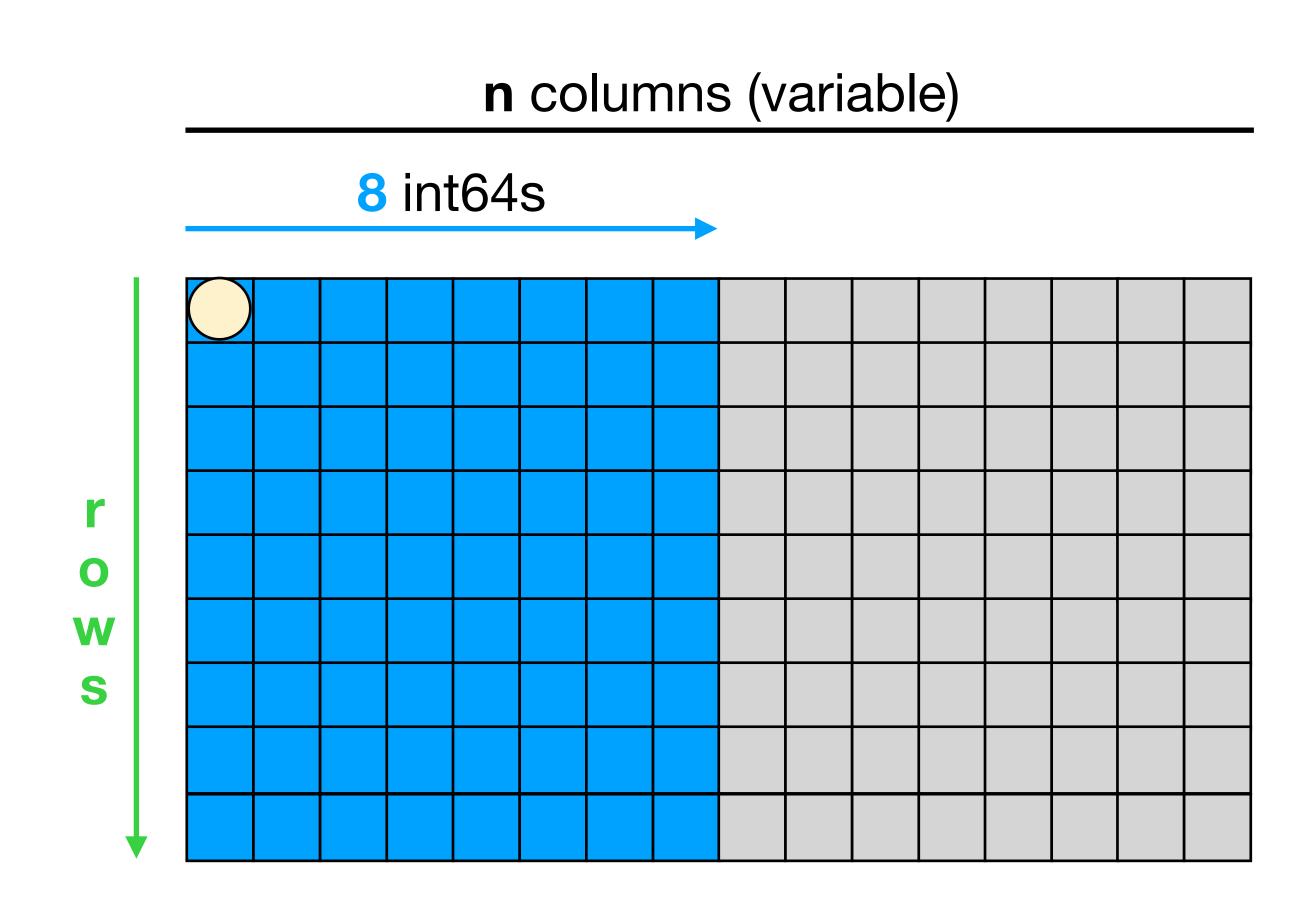
- Two-dimensional array of int64s
 64 bytes cache line (8 elements)
- Traverse each row multiple times the first
 8 columns only





- Two-dimensional array of int64s 64 bytes cache line (8 elements)
- Traverse each row multiple times the first
 8 columns only

```
for 0..k {
    for i in 0..rows {
        for j in 0..8 {
            sum += a[i][j]
        }
    }
}
```

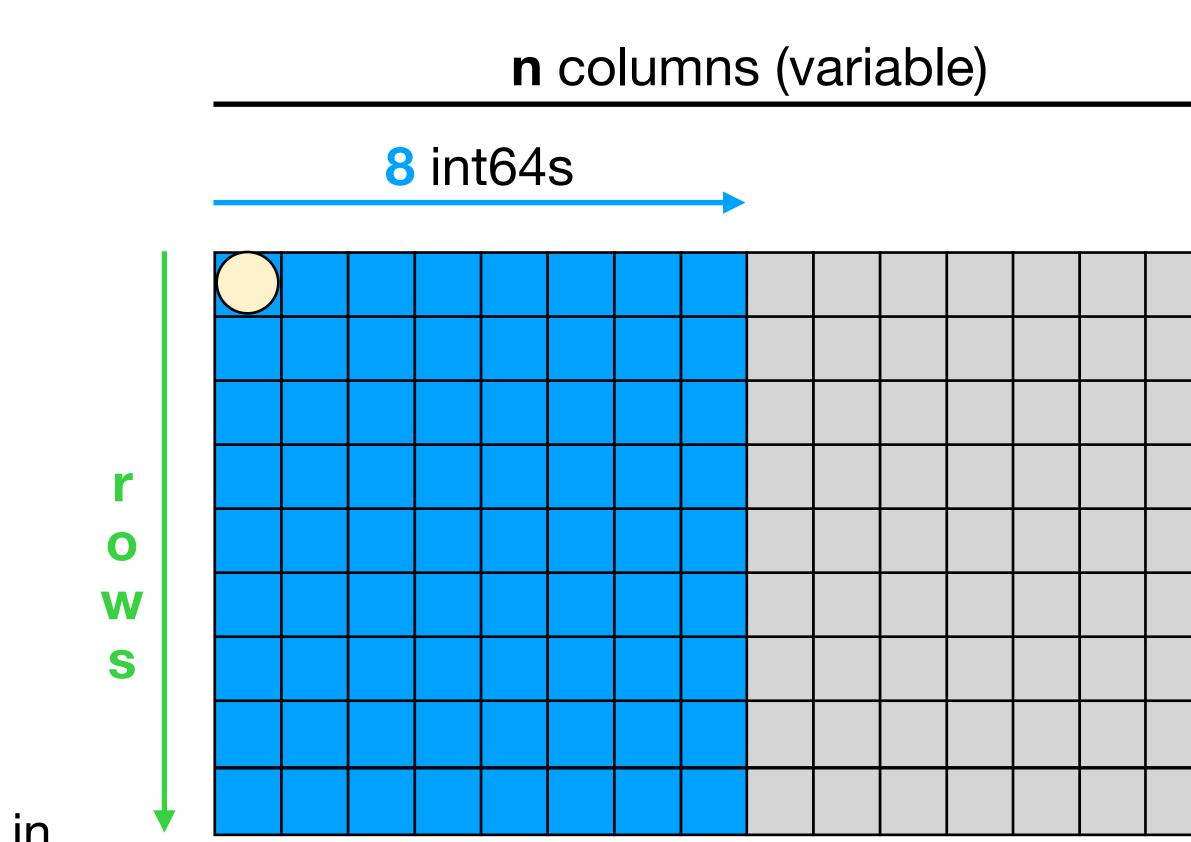




- Two-dimensional array of int64s 64 bytes cache line (8 elements)
- Traverse each row multiple times the first
 8 columns only

```
for 0..k {
    for i in 0..rows {
        for j in 0..8 {
            sum += a[i][j]
        }
    }
}
```

 rows is small enough so that each line should fit in the cache





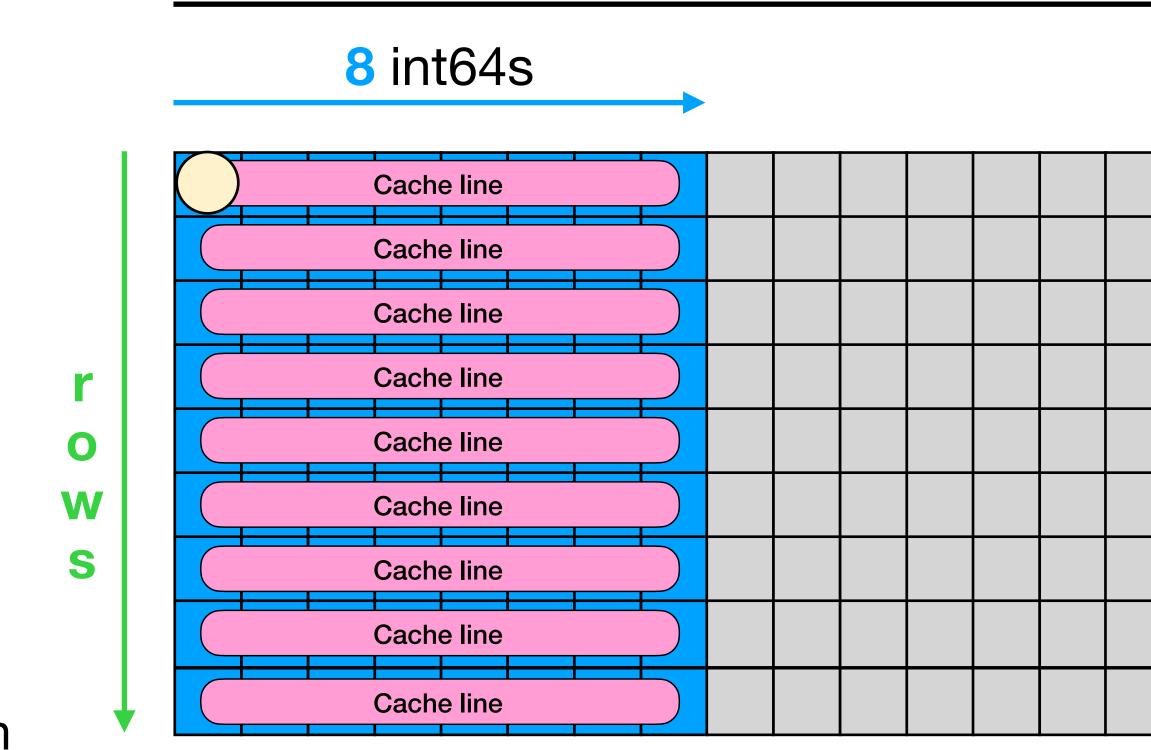


- Two-dimensional array of int64s 64 bytes cache line (8 elements)
- Traverse each row multiple times the first
 8 columns only

```
for 0..k {
    for i in 0..rows {
        for j in 0..8 {
            sum += a[i][j]
        }
    }
}
```

 rows is small enough so that each line should fit in the cache

n columns (variable)



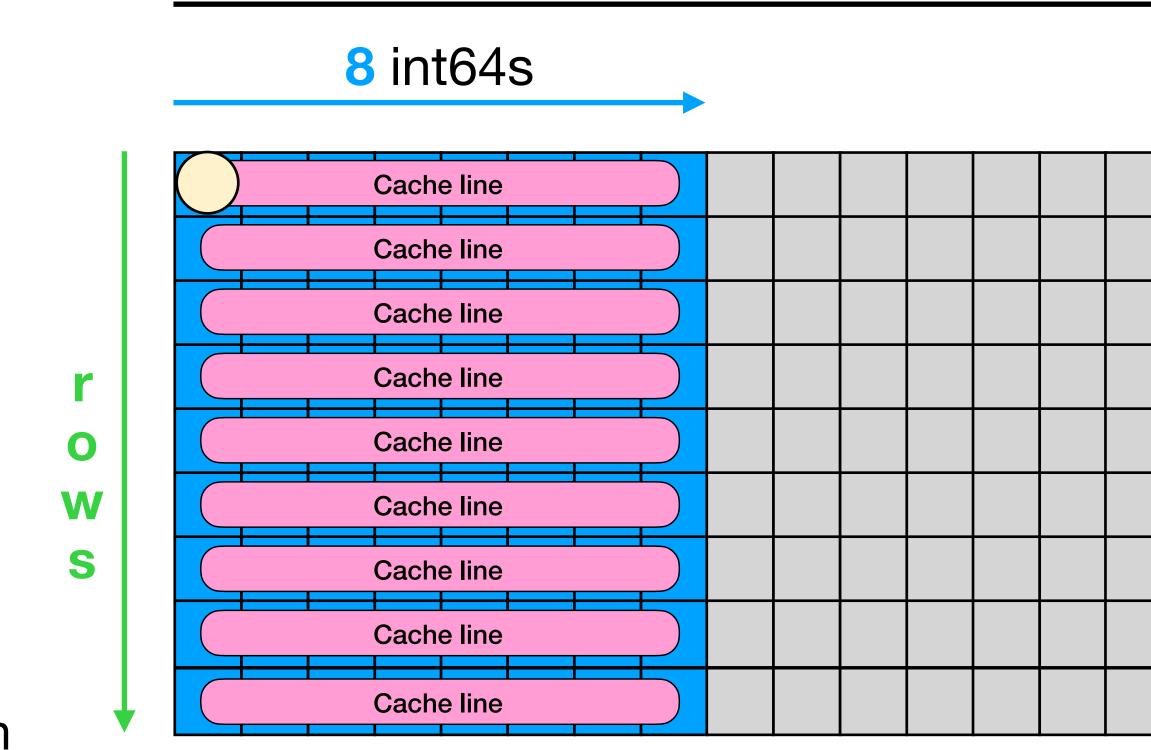




- Two-dimensional array of int64s
 64 bytes cache line (8 elements)
- Traverse each row multiple times the first
 8 columns only

- rows is small enough so that each line should fit in the cache
- The execution time depends on **n** (?)

n columns (variable)





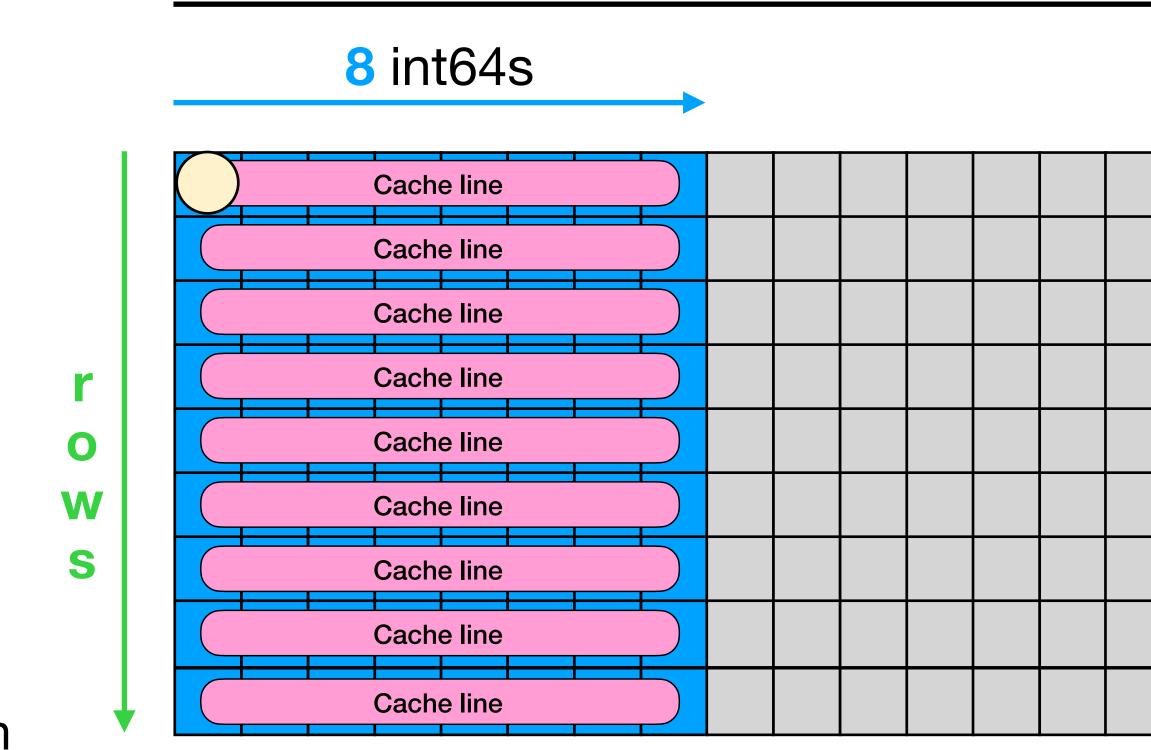


- Two-dimensional array of int64s 64 bytes cache line (8 elements)
- Traverse each row multiple times the first
 8 columns only

```
for 0..k {
    for i in 0..rows {
        for j in 0..8 {
            sum += a[i][j]
        }
    }
}
```

- rows is small enough so that each line should fit in the cache
- The execution time depends on **n** (?)
- Depending on n, the execution can be up to 100% slower

n columns (variable)







0000000
0000100
0001000
0001100
0010000
0010100
0011000
0011100
0100000
0110100
0111000
0111100
1000000

0010000



A block is referenced by an address

0010000



A block is referenced by an address

We want to iterate on each blue block

0000000
0000100
0001000
0001100
0010000
0010100
0011000
0011100
0100000
0110100
0111000
0111100
1000000 >>

0010000



A block is referenced by an address

We want to iterate on each blue block

0000000
0000100
0001000
0001100
0010000
0010100
0011000
0011100
0100000
0110100
0111000
0111100
1000000

Cache

0010000

Program:

. . .

load address 0000000

. . .



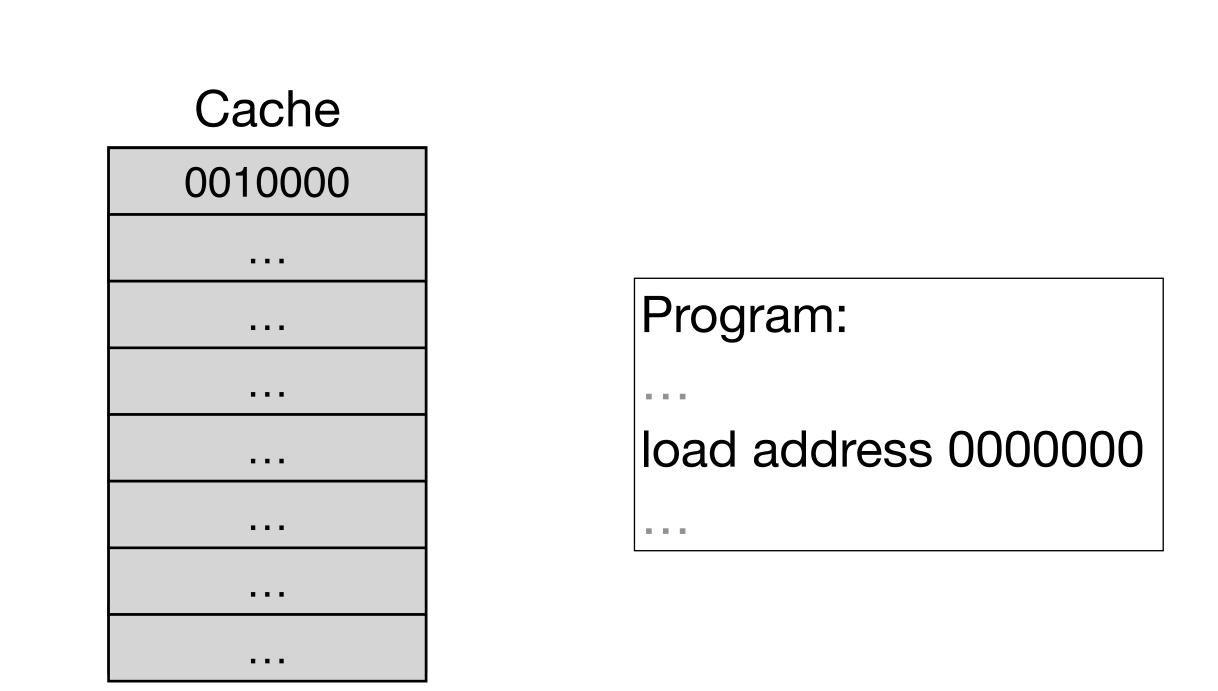


A block is referenced by an address

We want to iterate on each **blue block**

0000000
0000100
0001000
0001100
0010000
0010100
0011000
0011100
0100000
0110100
0111000
0111100
1000000

 In a fully associative cache, we may h address is present



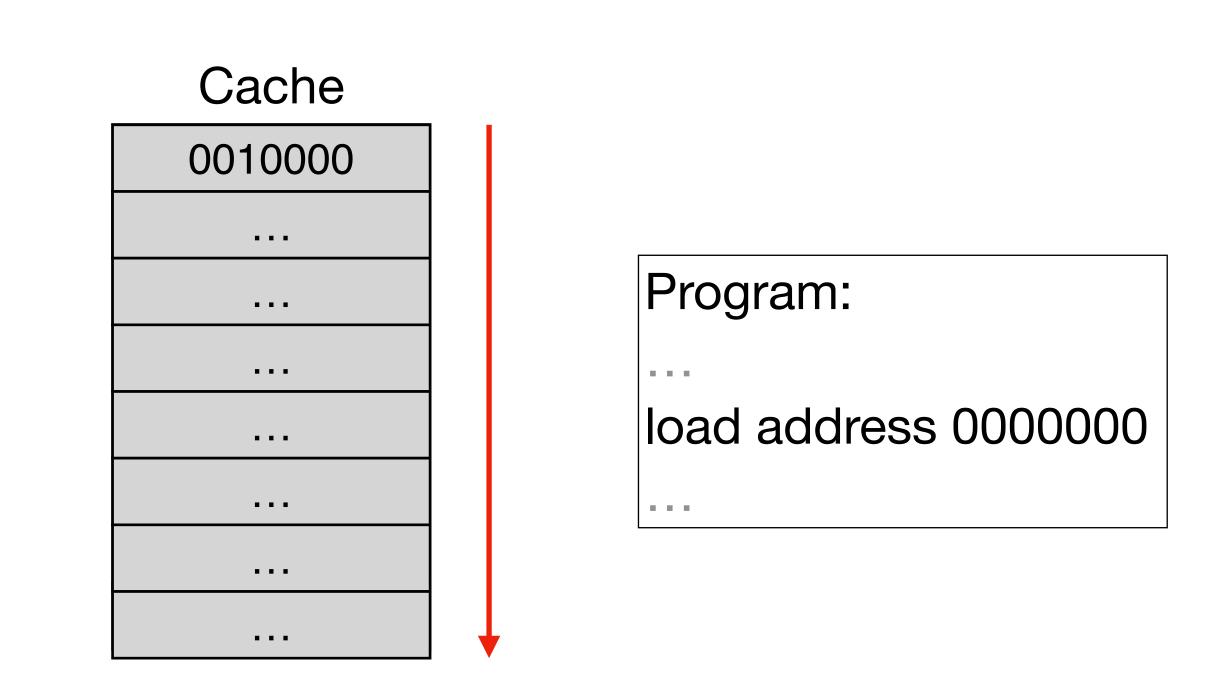


A block is referenced by an address

We want to iterate on each **blue block**

0000000
0000100
0001000
0001100
0010000
0010100
0011000
0011100
0100000
0110100
0111000
0111100
1000000

 In a fully associative cache, we may h address is present



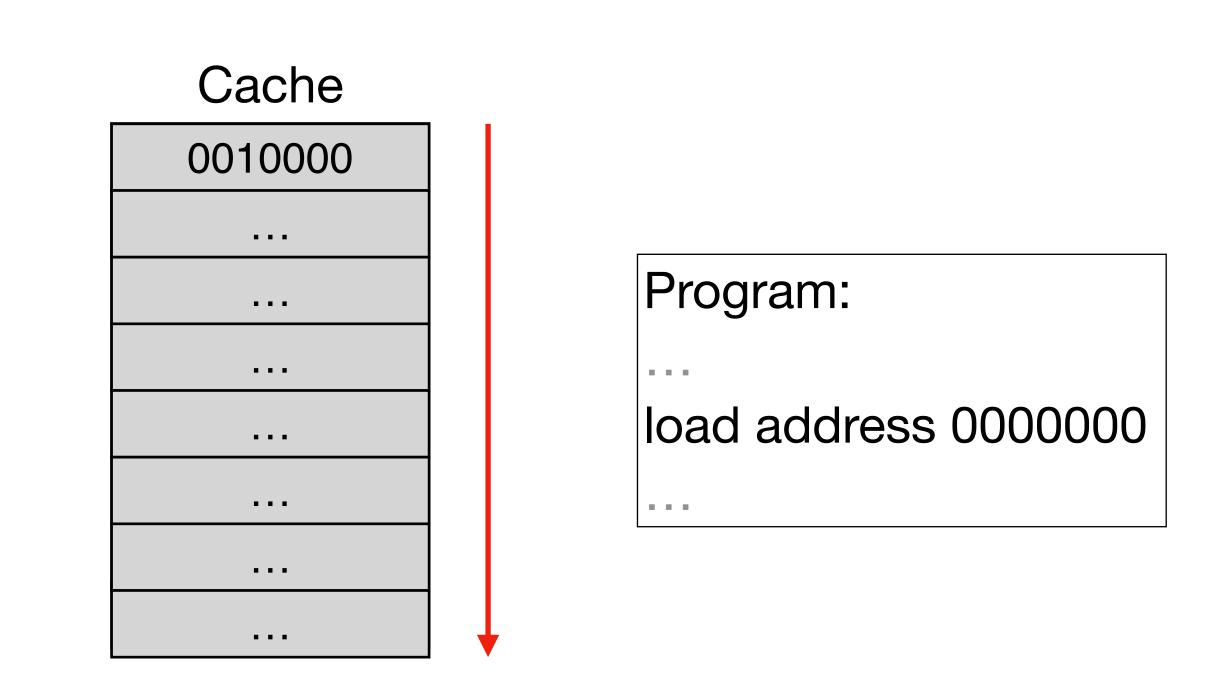


A block is referenced by an address

We want to iterate on each **blue block**

0000000
0000100
0001000
0001100
0010000
0010100
0011000
0011100
0100000
0110100
0111000
0111100
1000000

- address is present
- Example on an Intel Core i5-7300 L1D: we need to iterate on 512 lines lacksquare



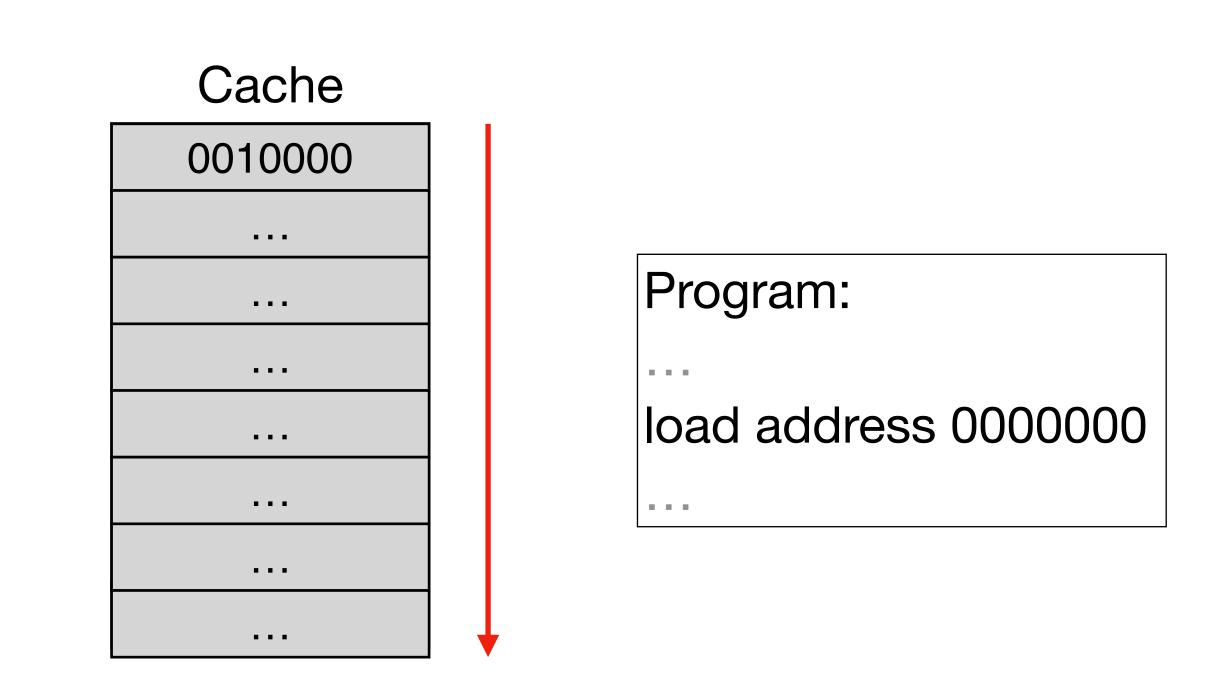


A block is referenced by an address

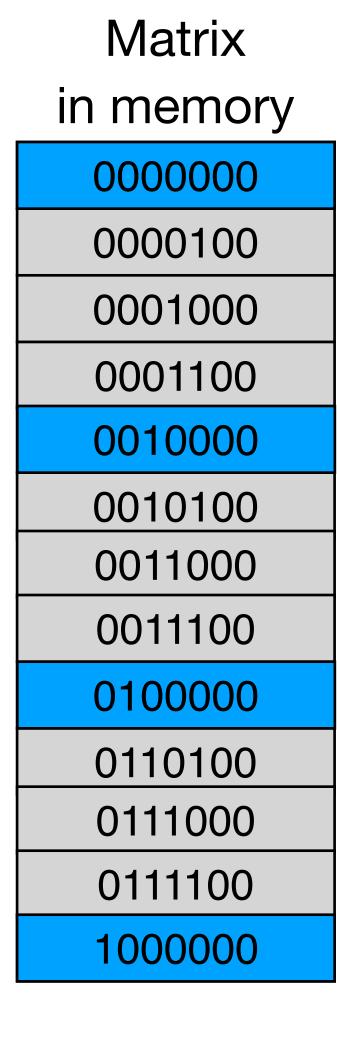
We want to iterate on each **blue block**

0000000
0000100
0001000
0001100
0010000
0010100
0011000
0011100
0100000
0110100
0111000
0111100
1000000

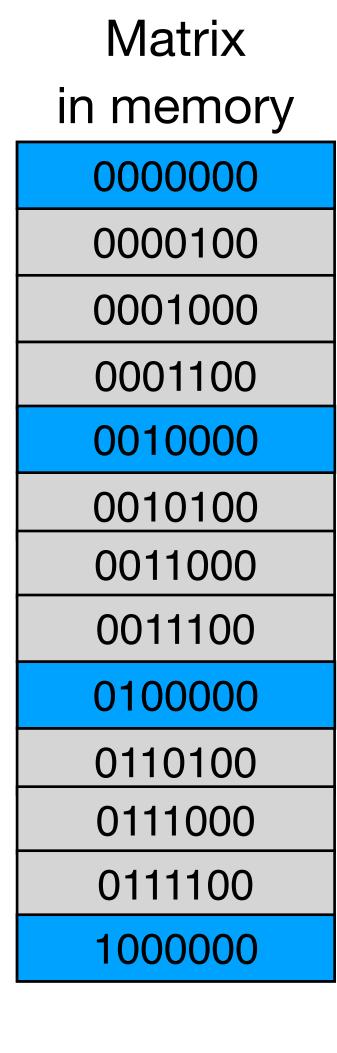
- address is present
- Example on an Intel Core i5-7300 L1D: we need to iterate on 512 lines
- Solution: partitioning





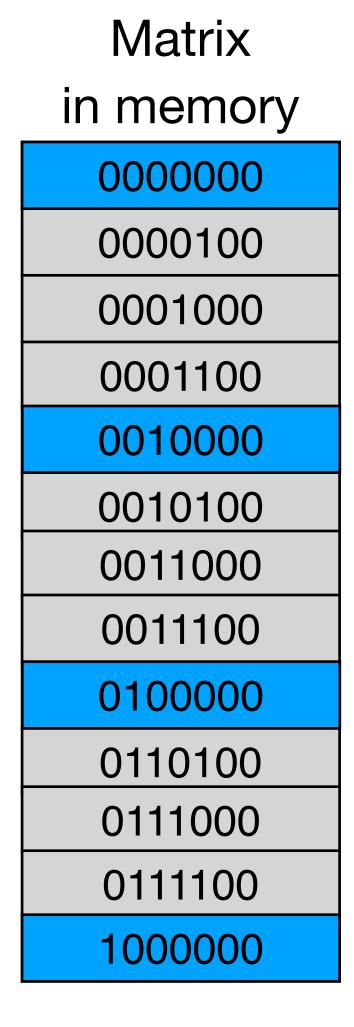






0 0 0 0 0 0

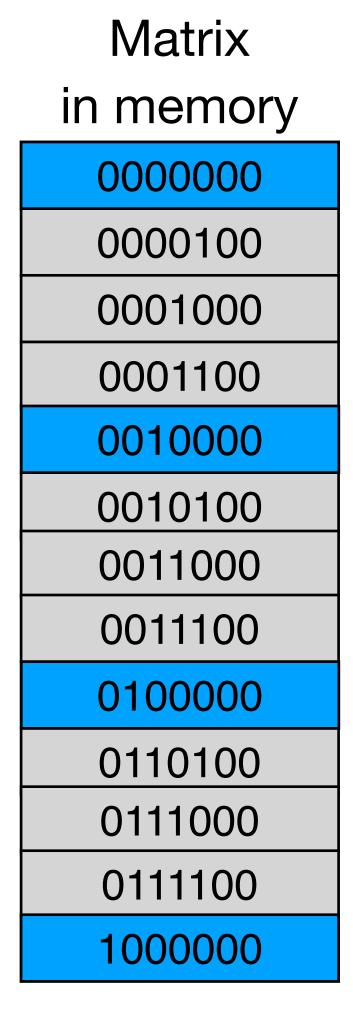




0 0 0 0 0 0

E.g Block size: 4 bits

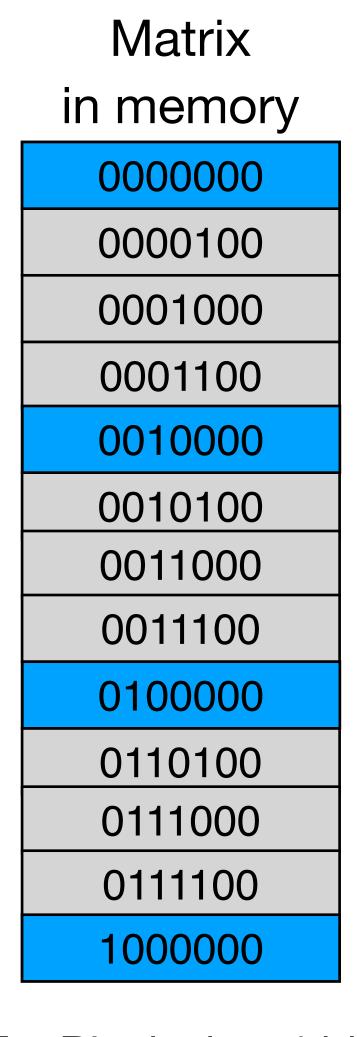




0 0 0 0 0 0

E.g Block size: 4 bits
4 = 2²
2 represents the
block offset (bo)



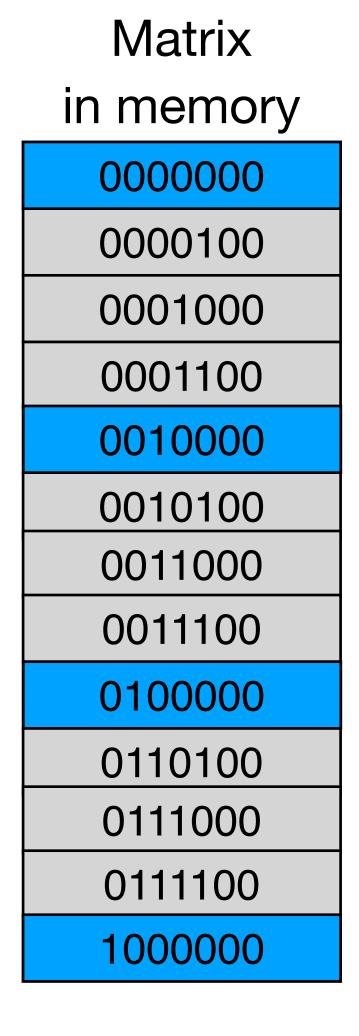


0 0 0 0 0 0 0

bo

E.g Block size: 4 bits
4 = 2²
2 represents the
block offset (bo)





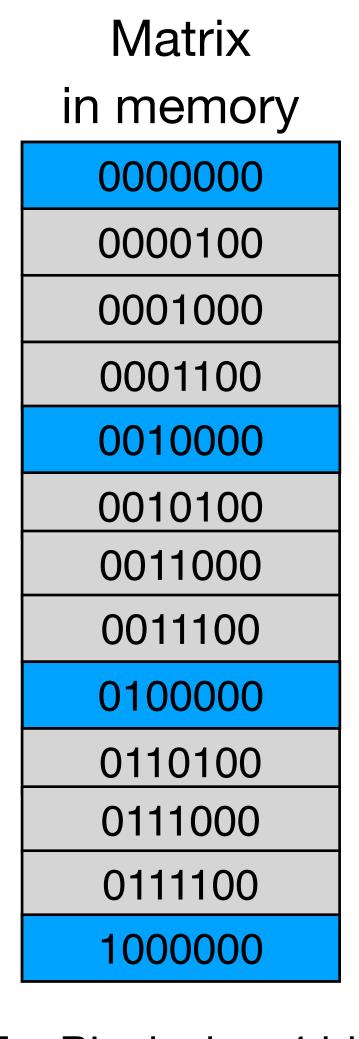
0 0 0 0 0 0 0

bo

E.g Block size: 4 bits $4 = 2^{2}$ 2 represents the block offset (bo)

A cache is **partitioned** into **sets**





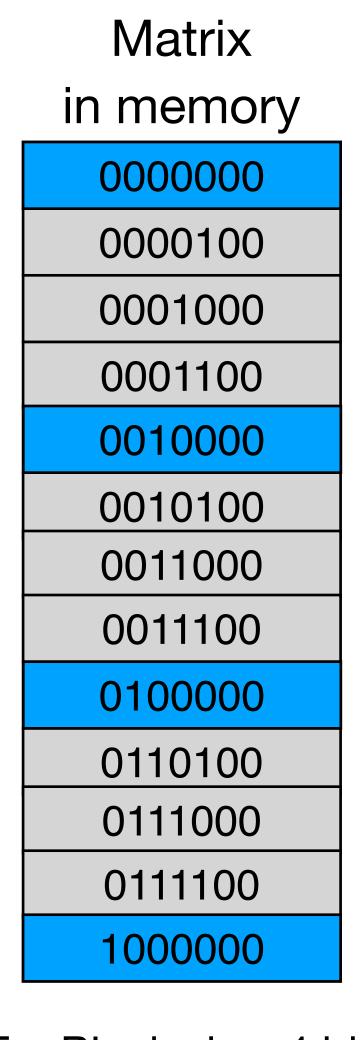
0 0 0 0 0 0 0

bo

E.g Block size: 4 bits $4 = 2^{2}$ 2 represents the block offset (bo)

A cache is **partitioned** into **sets** A block can belong to **only one** set





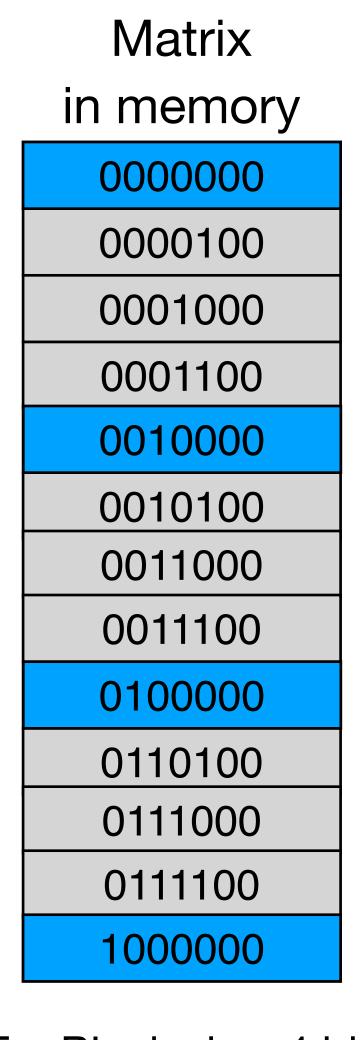
0 0 0 0 0 0 0

bo

E.g Block size: 4 bits $4 = 2^{2}$ 2 represents the **block offset** (bo)

A cache is **partitioned** into **sets** A block can belong to **only one** set k-way associative cache: k lines per set





0 0 0 0 0 0 0

bo

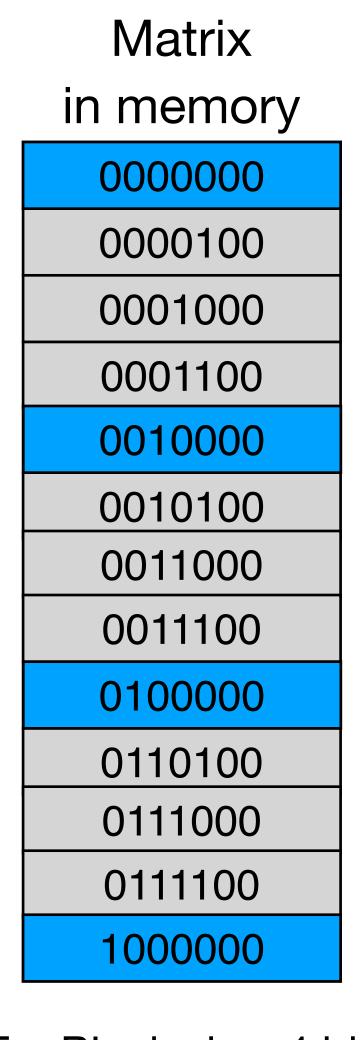
A cache is **partitioned** into **sets** A block can belong to **only one** set k-way associative cache: k lines per set

E.g Block size: 4 bits $4 = 2^{2}$ 2 represents the block offset (bo)

E.g. 8 lines, 2-way associative







0 0 0 0 0 0 0

bo

A cache is **partitioned** into **sets** A block can belong to **only one** set k-way associative cache: k lines per set

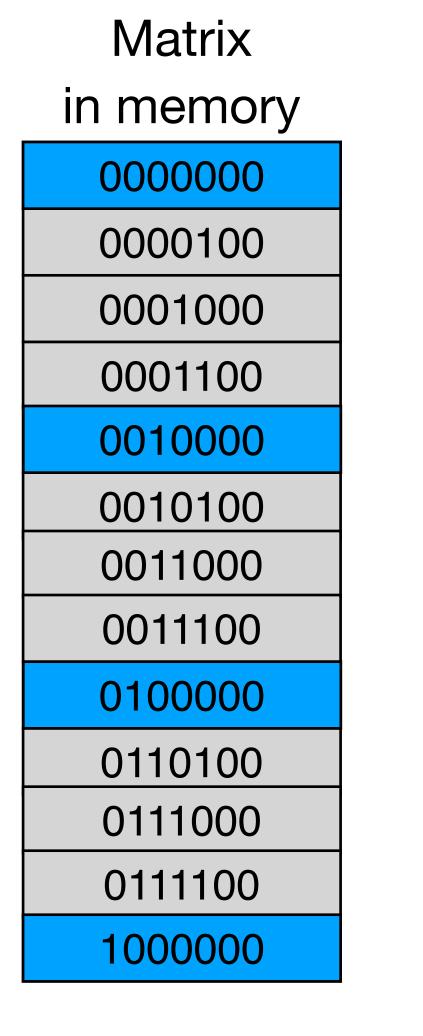
E.g Block size: 4 bits $4 = 2^{2}$ 2 represents the block offset (bo)

Cache

E.g. 8 lines, 2-way associative nb of sets = 8 / 2 = 4







0	0	0	0	0	0 bc		A ca A blo k-way a
							set 0
							set 1
							set 2
							set 3

E.g Block size: 4 bits $4 = 2^{2}$ 2 represents the block offset (bo)

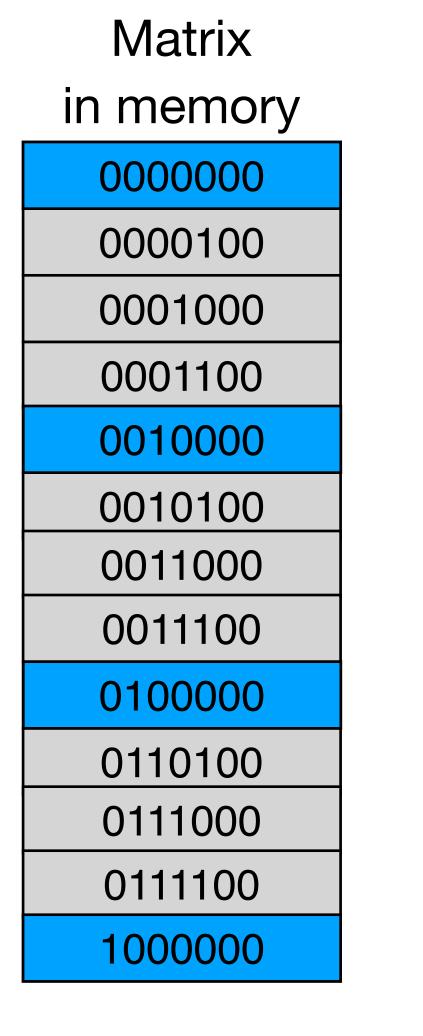
ache is **partitioned** into sets ock can belong to only one set associative cache: k lines per set

Cache

E.g. 8 lines, 2-way associative nb of sets = 8/2 = 4







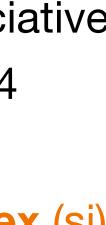
Load

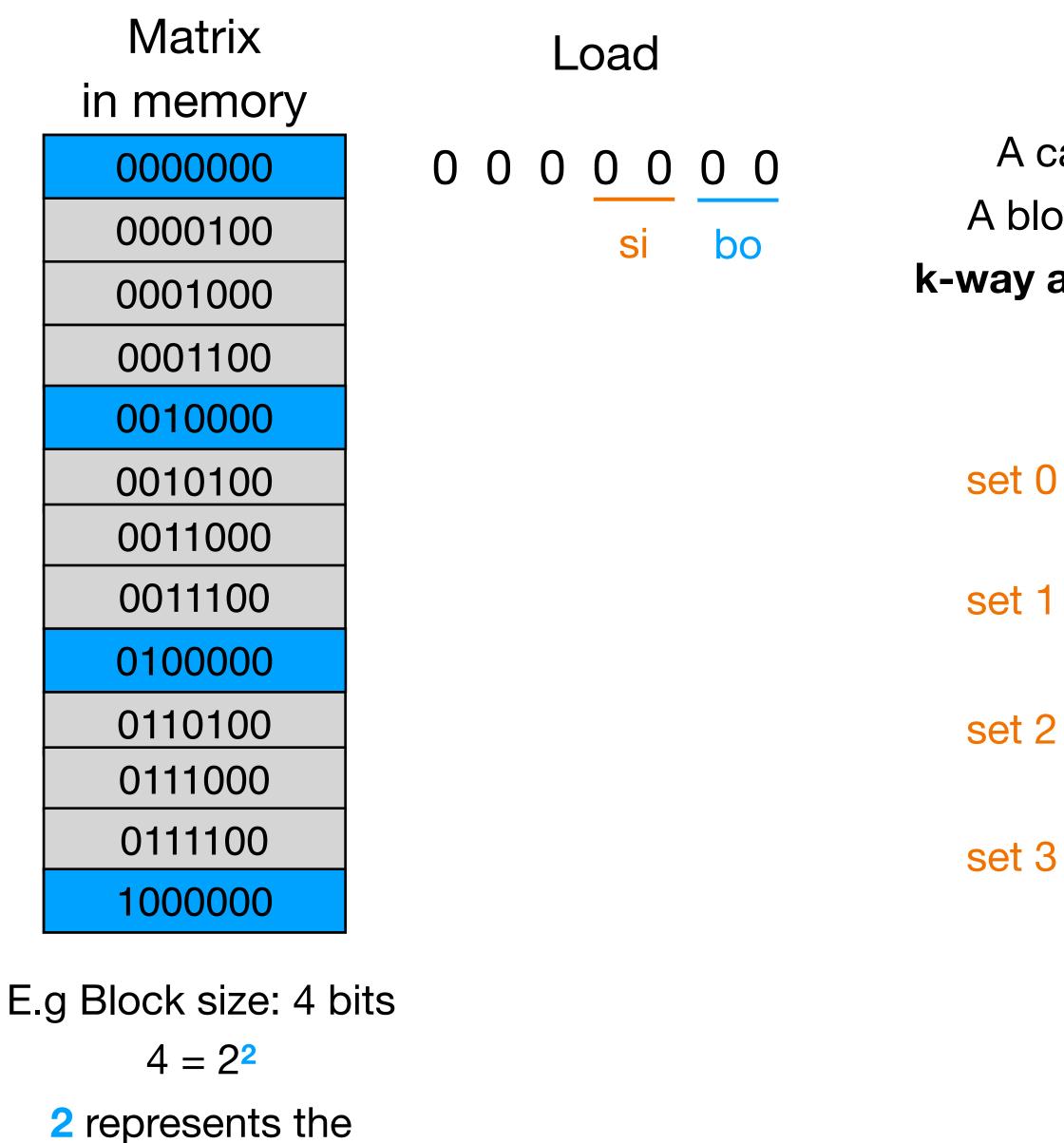
A cache is partitioned in A block can belong to only k-way associative cache: k		
	Cache	
set 0		
set 1		
set 2		
set 3		

E.g Block size: 4 bits $4 = 2^{2}$ 2 represents the **block offset** (bo)

nto sets y one set lines per set E.g. 8 lines, 2-way associative nb of sets = 8/2 = 4 $4 = 2^{2}$







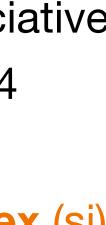
A cache is **partitioned** into **sets** A block can belong to **only one** set k-way associative cache: k lines per set

	Cache							
)								
2								
3								

Cacho

E.g. 8 lines, 2-way associative nb of sets = 8 / 2 = 4 $4 = 2^{2}$



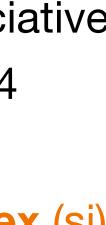


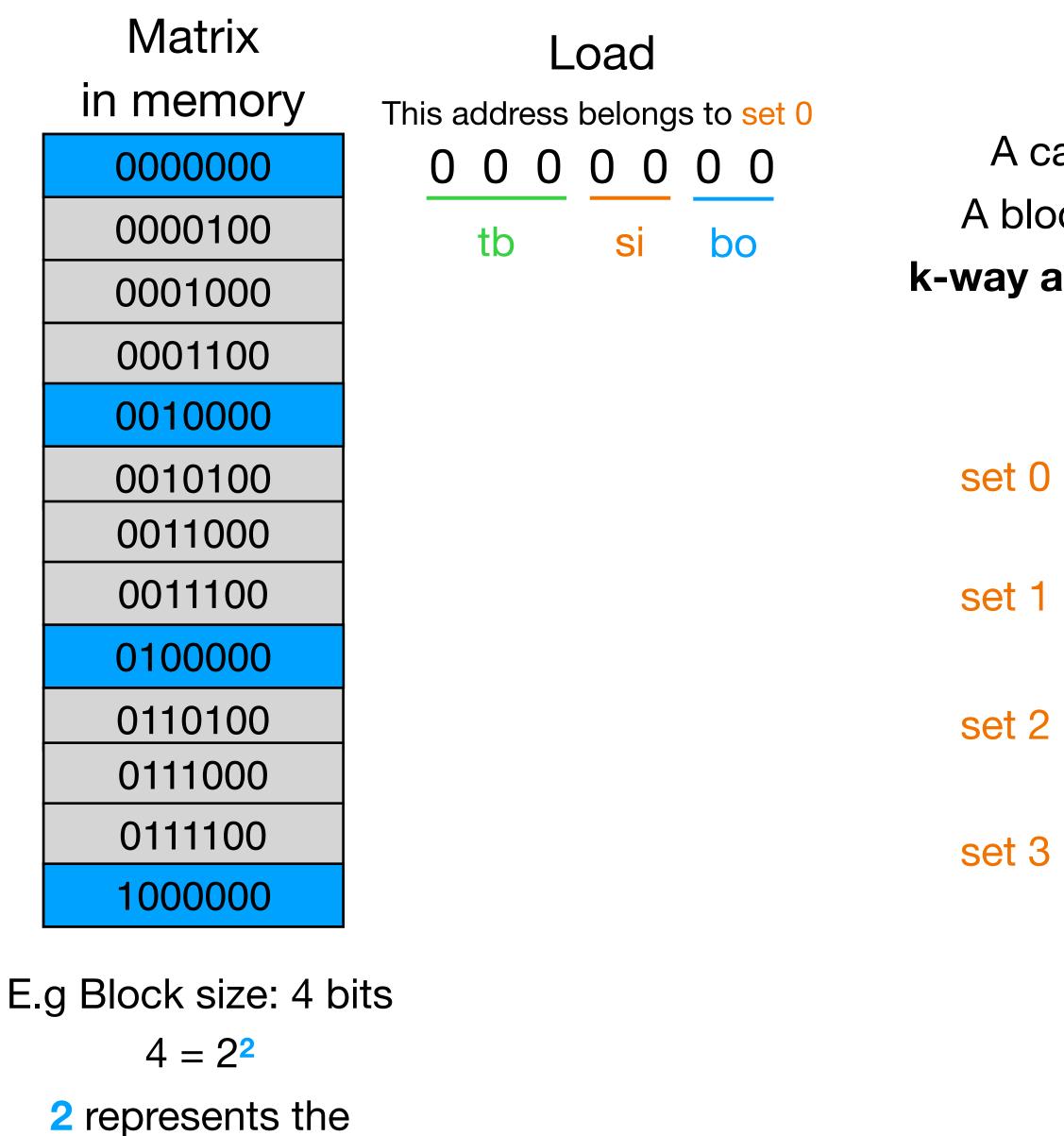
	Matrix in memory	Load This address belongs to set 0	-	
	0000000	000000		he is partitioned int
	0000100	si bo		can belong to only
	0001000		k-way ass	sociative cache: k li
	0001100			Cache
	0010000		1	
	0010100		set 0	
	0011000			
	0011100		set 1	
	0100000			
	0110100		set 2	
	0111000			
	0111100		set 3	
	1000000			
E.	g Block size: 4 b $4 = 2^2$	bits		

2 represents the block offset (bo)

nto sets ly one set lines per set E.g. 8 lines, 2-way associative nb of sets = 8 / 2 = 44 = 2<mark>2</mark>







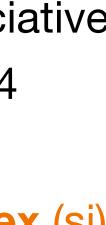
A cache is **partitioned** into **sets** A block can belong to **only one** set k-way associative cache: k lines per set

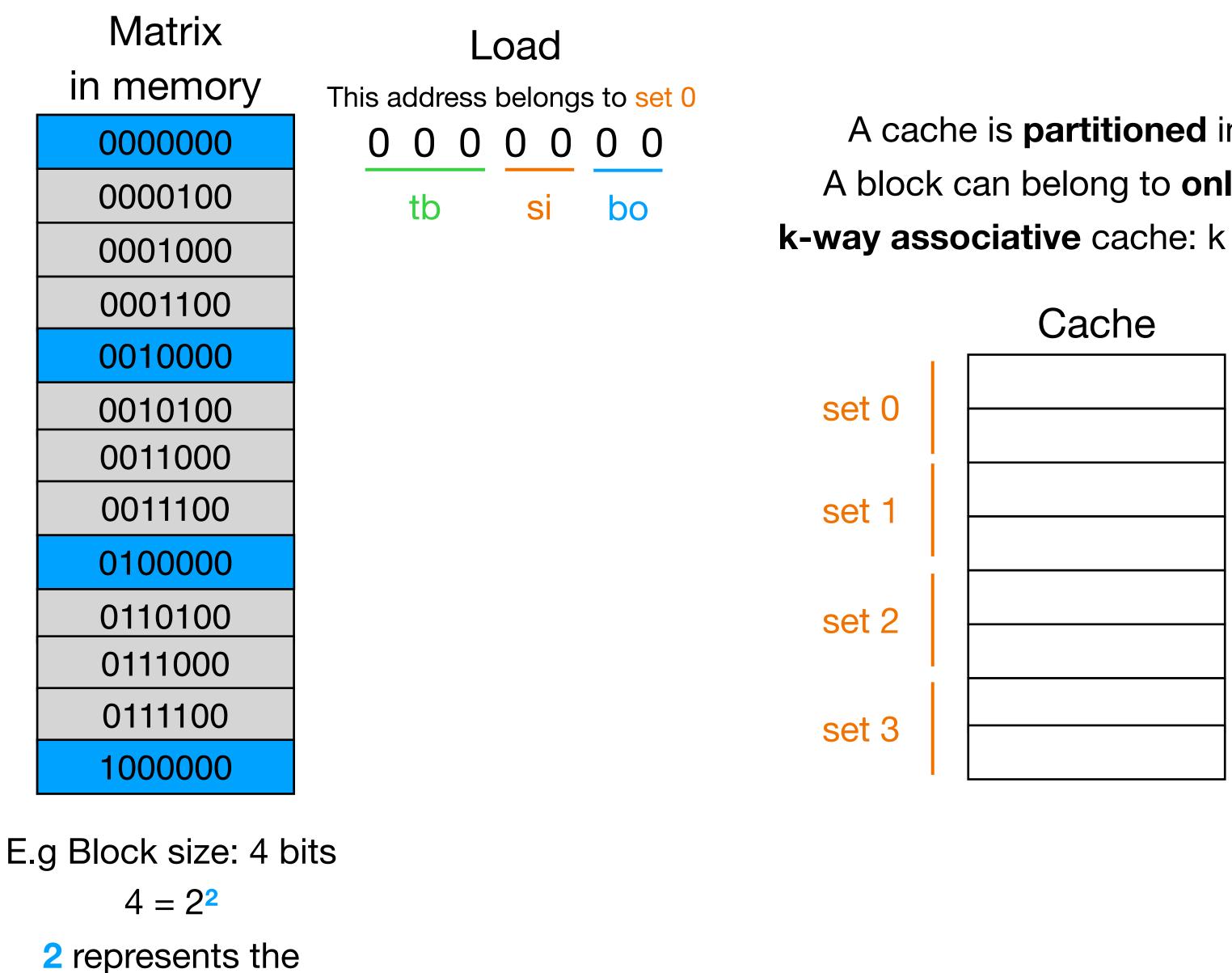
	Cache							
)								
2								
3								

Cacho

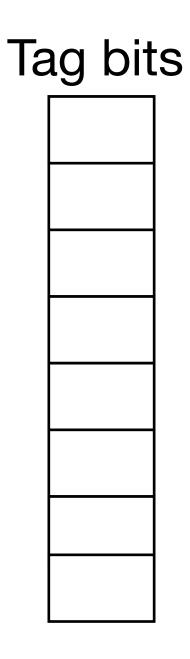
E.g. 8 lines, 2-way associative nb of sets = 8 / 2 = 4 $4 = 2^{2}$



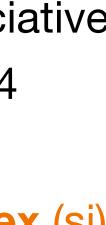


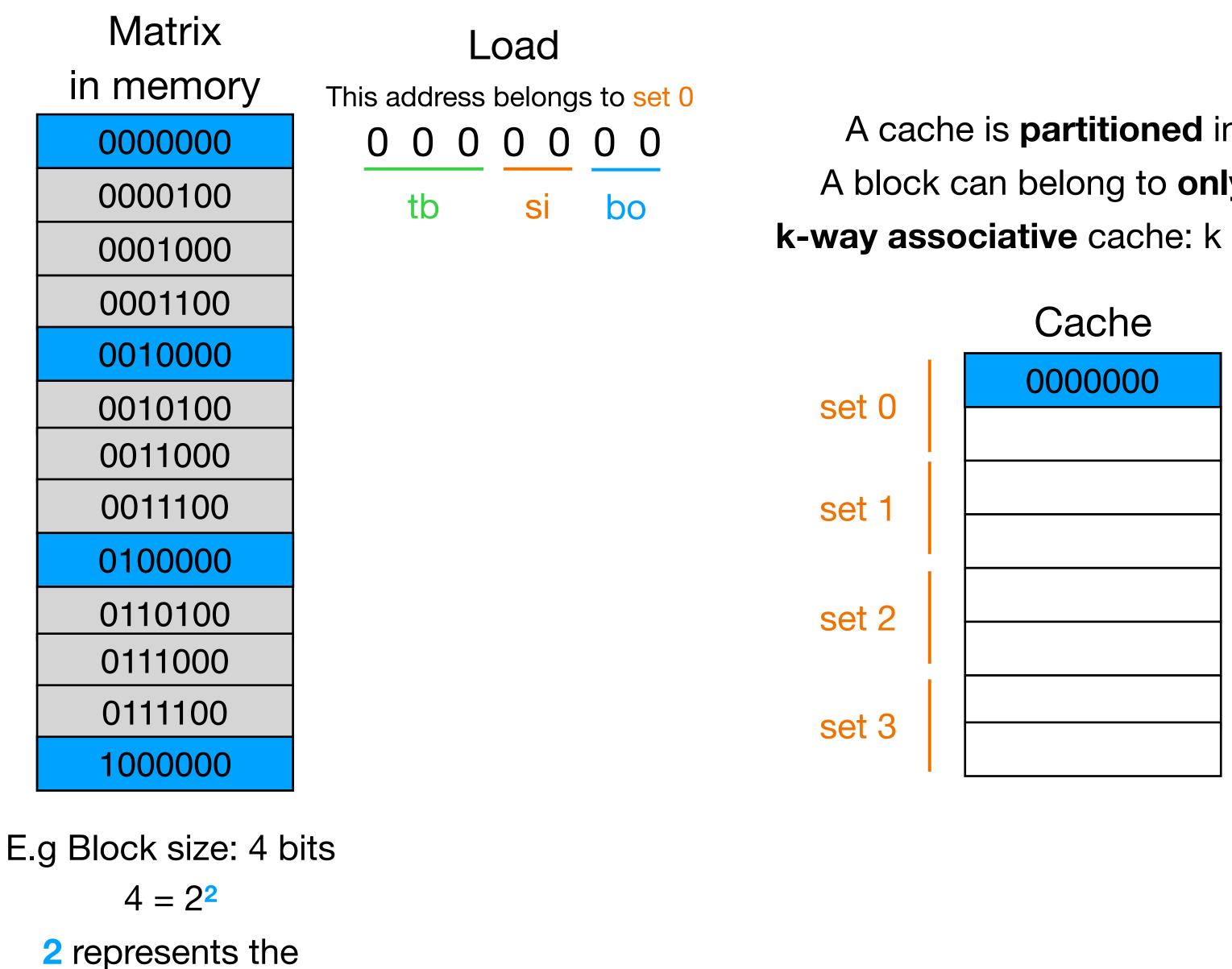


A cache is **partitioned** into **sets** A block can belong to **only one** set k-way associative cache: k lines per set E.g. 8 lines, 2-way associative nb of sets = 8/2 = 4 $4 = 2^{2}$

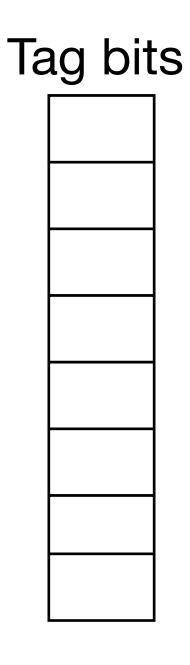




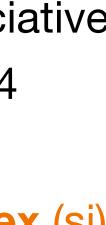


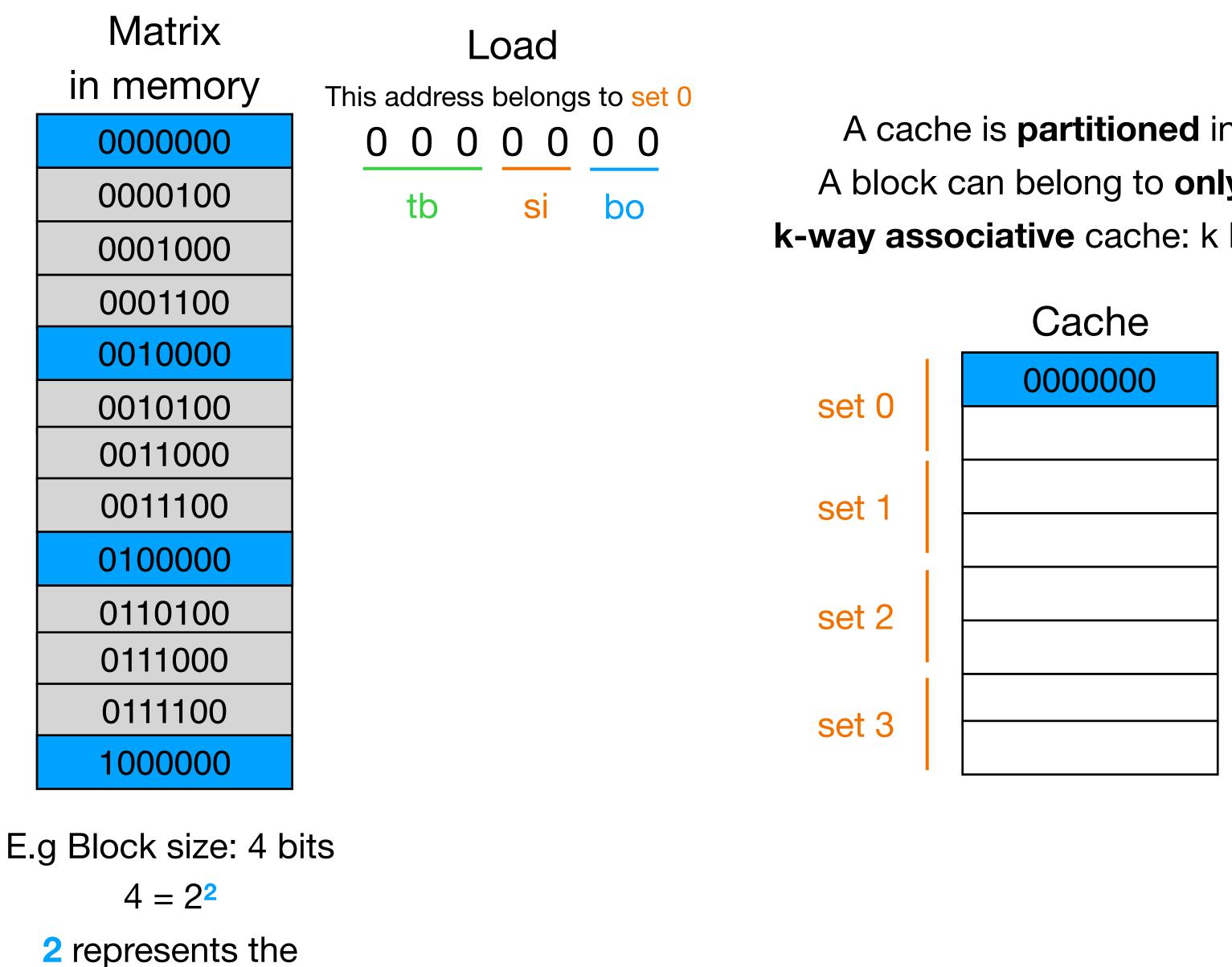


A cache is **partitioned** into **sets** A block can belong to **only one** set k-way associative cache: k lines per set E.g. 8 lines, 2-way associative nb of sets = 8/2 = 4 $4 = 2^{2}$

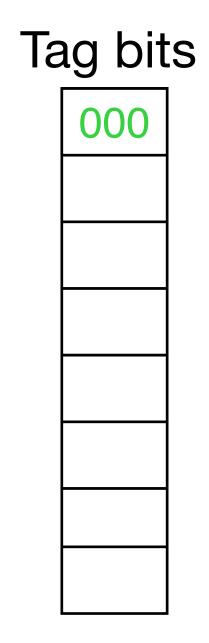




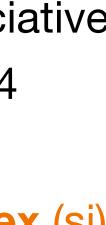




A cache is **partitioned** into **sets** A block can belong to **only one** set k-way associative cache: k lines per set E.g. 8 lines, 2-way associative nb of sets = 8/2 = 4 $4 = 2^{2}$

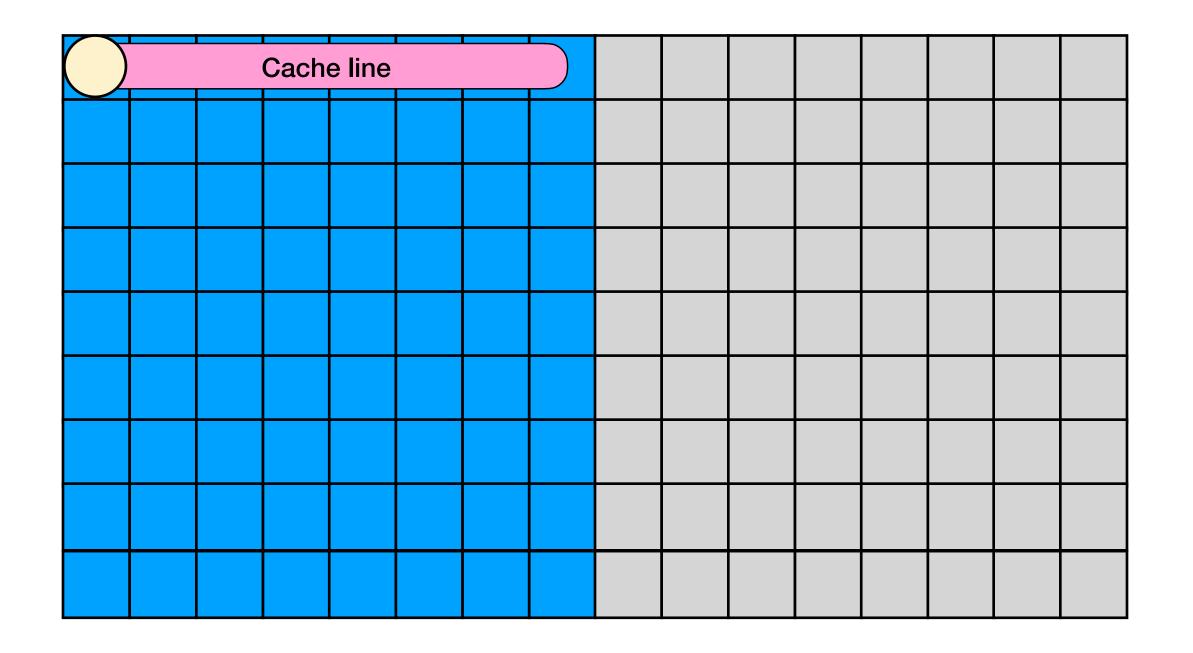






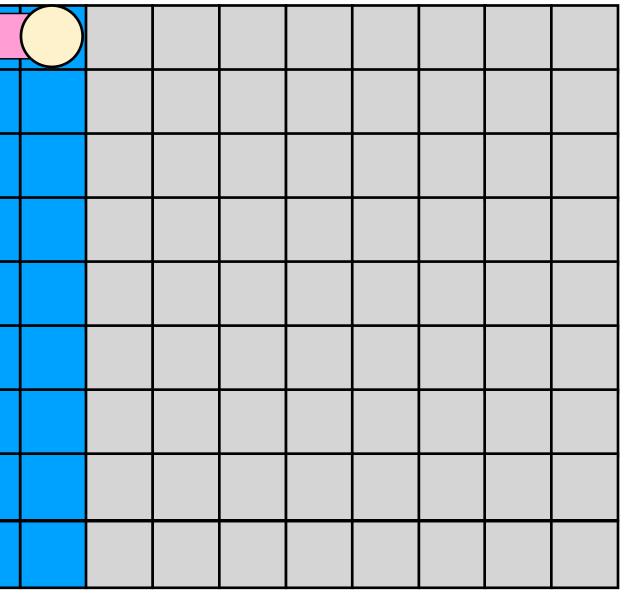
\bigcirc								



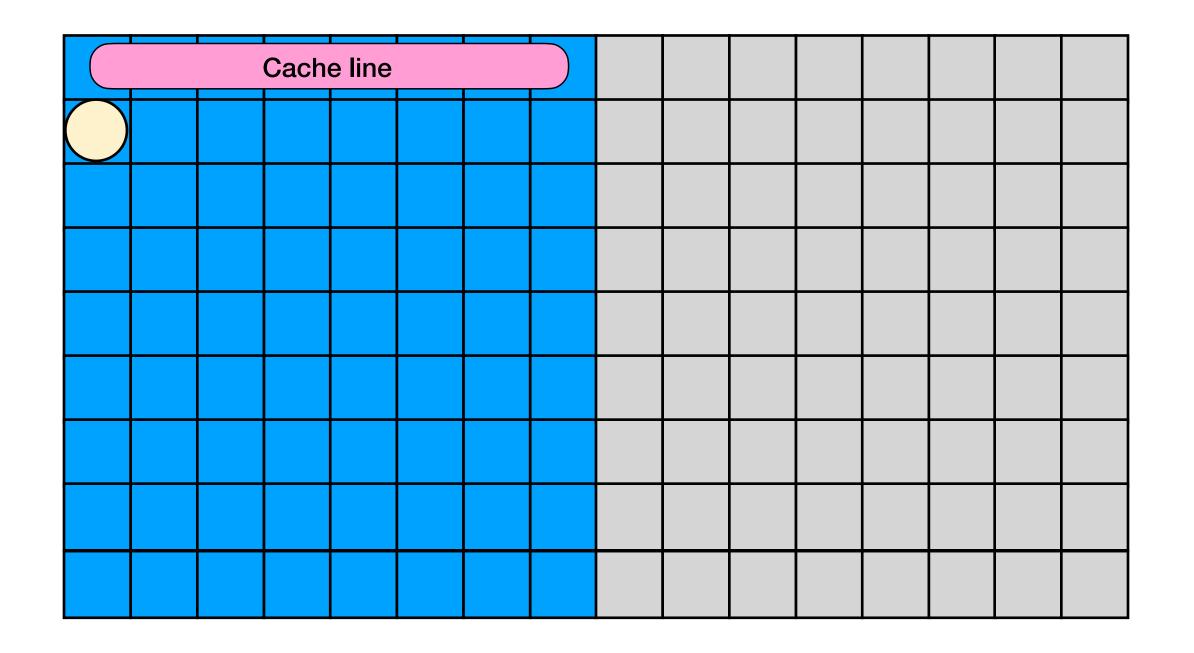




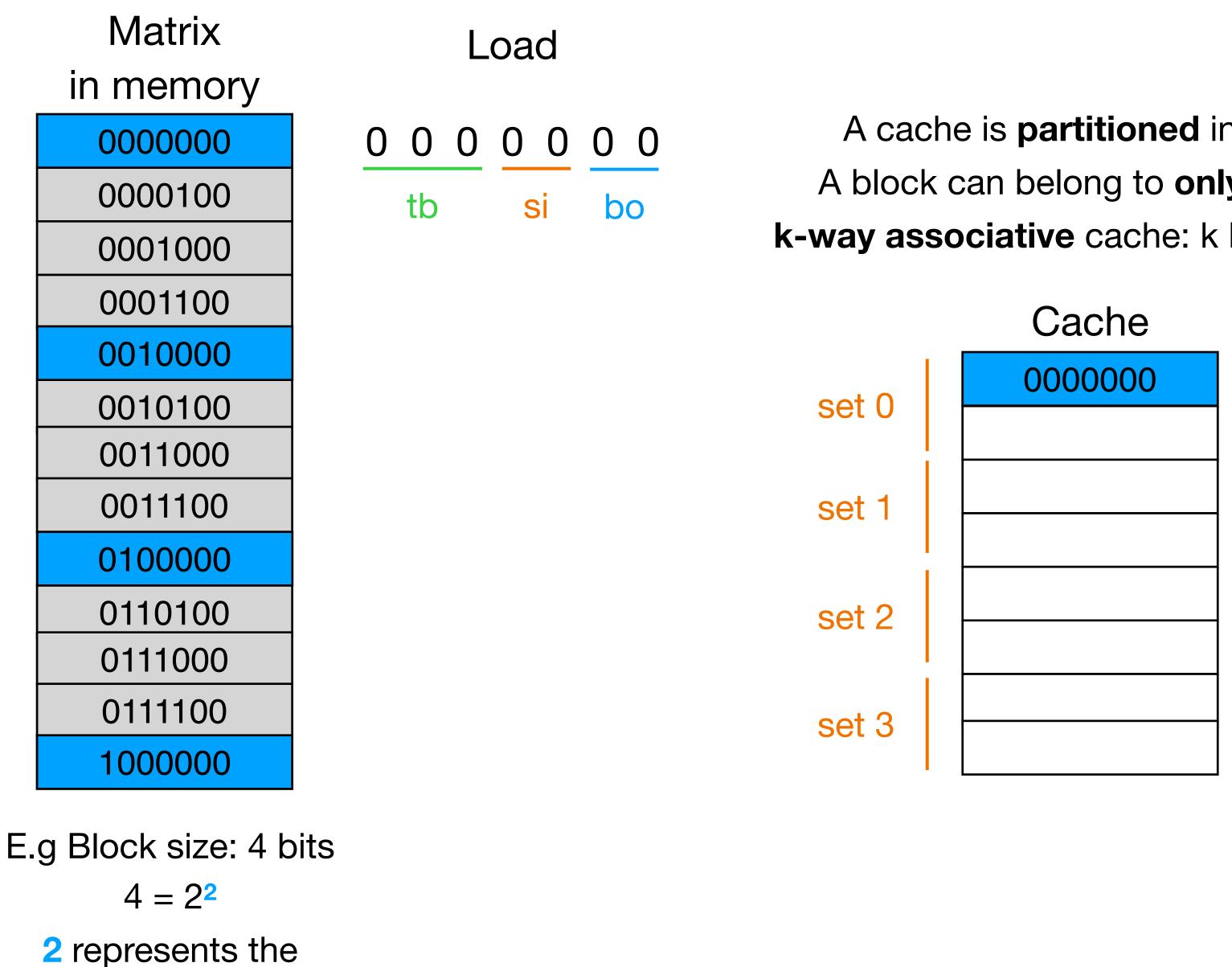
Cache line						



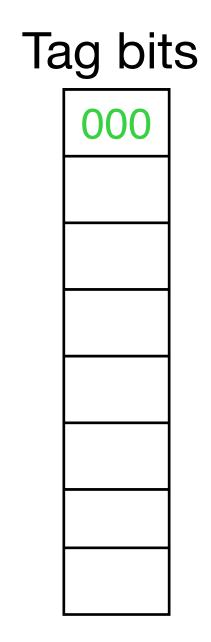




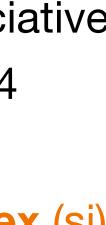


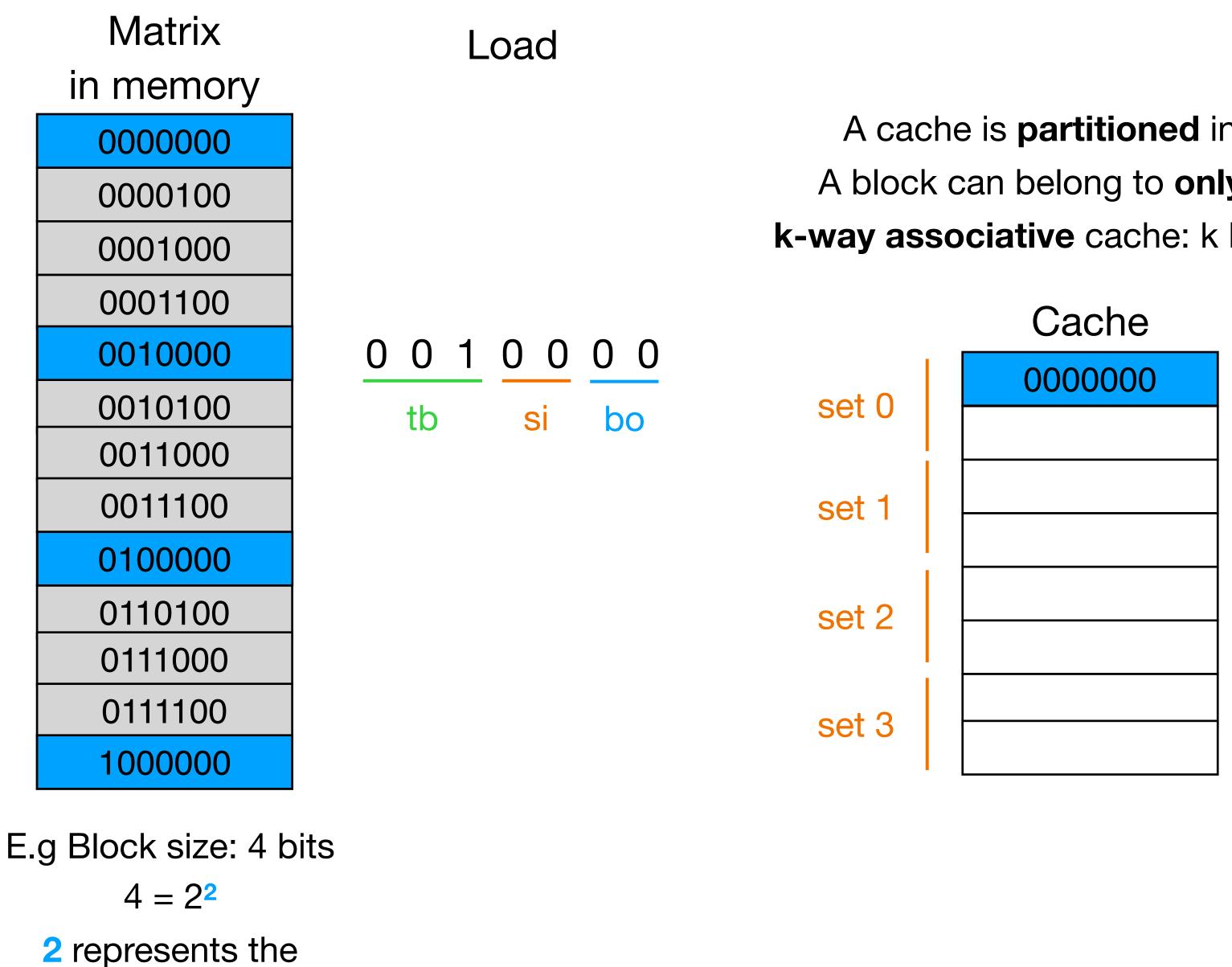


A cache is **partitioned** into **sets** A block can belong to **only one** set k-way associative cache: k lines per set E.g. 8 lines, 2-way associative nb of sets = 8/2 = 4 $4 = 2^{2}$

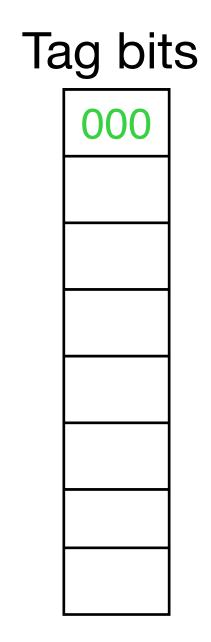




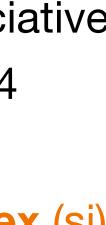


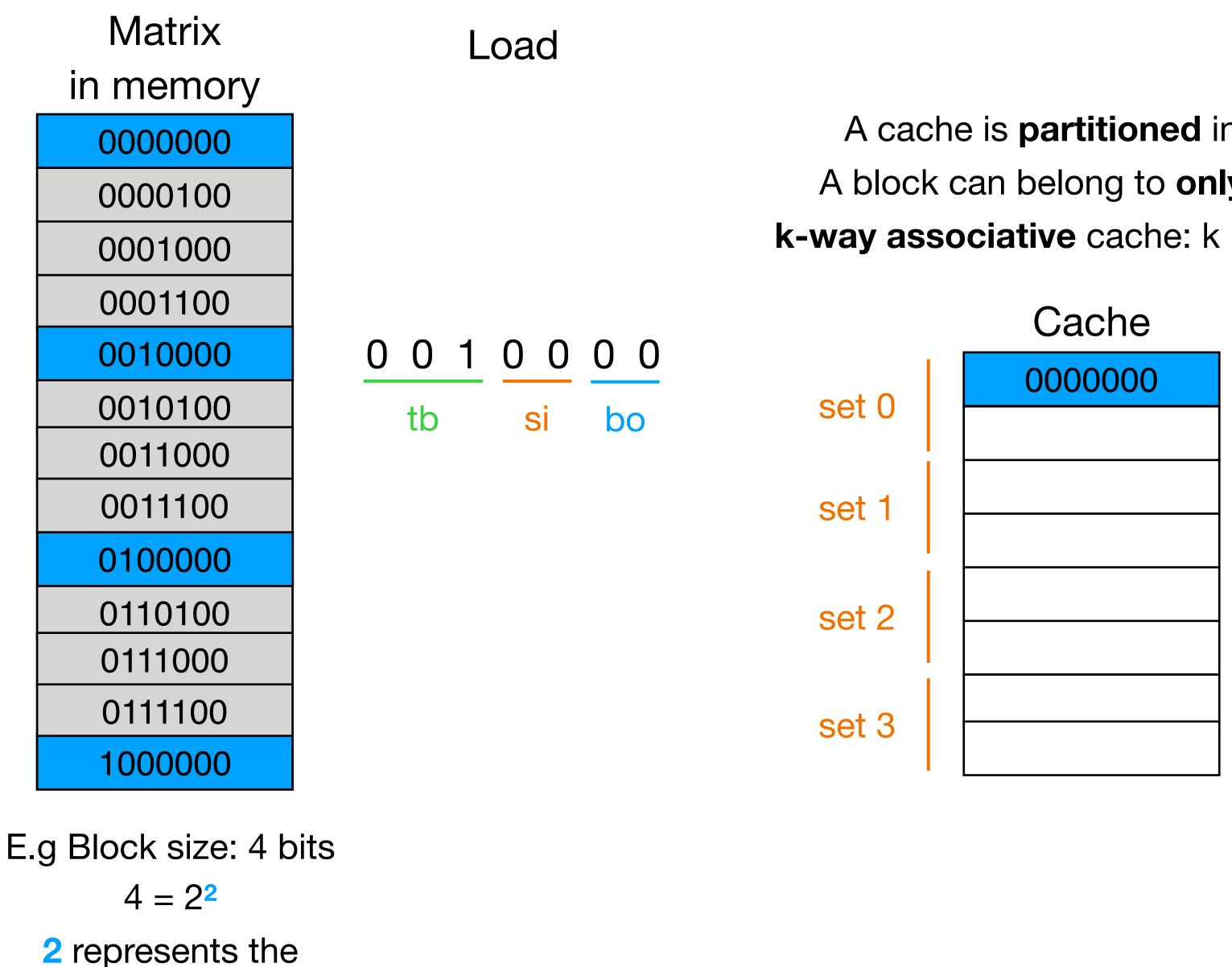


A cache is **partitioned** into **sets** A block can belong to **only one** set k-way associative cache: k lines per set E.g. 8 lines, 2-way associative nb of sets = 8/2 = 4 $4 = 2^{2}$

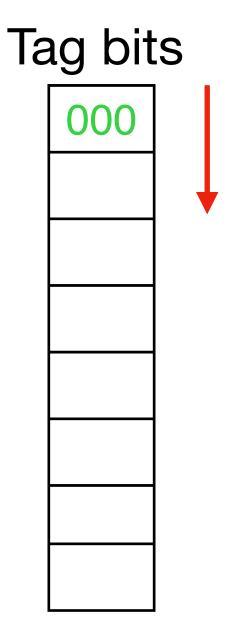




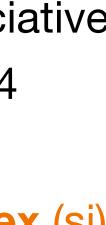


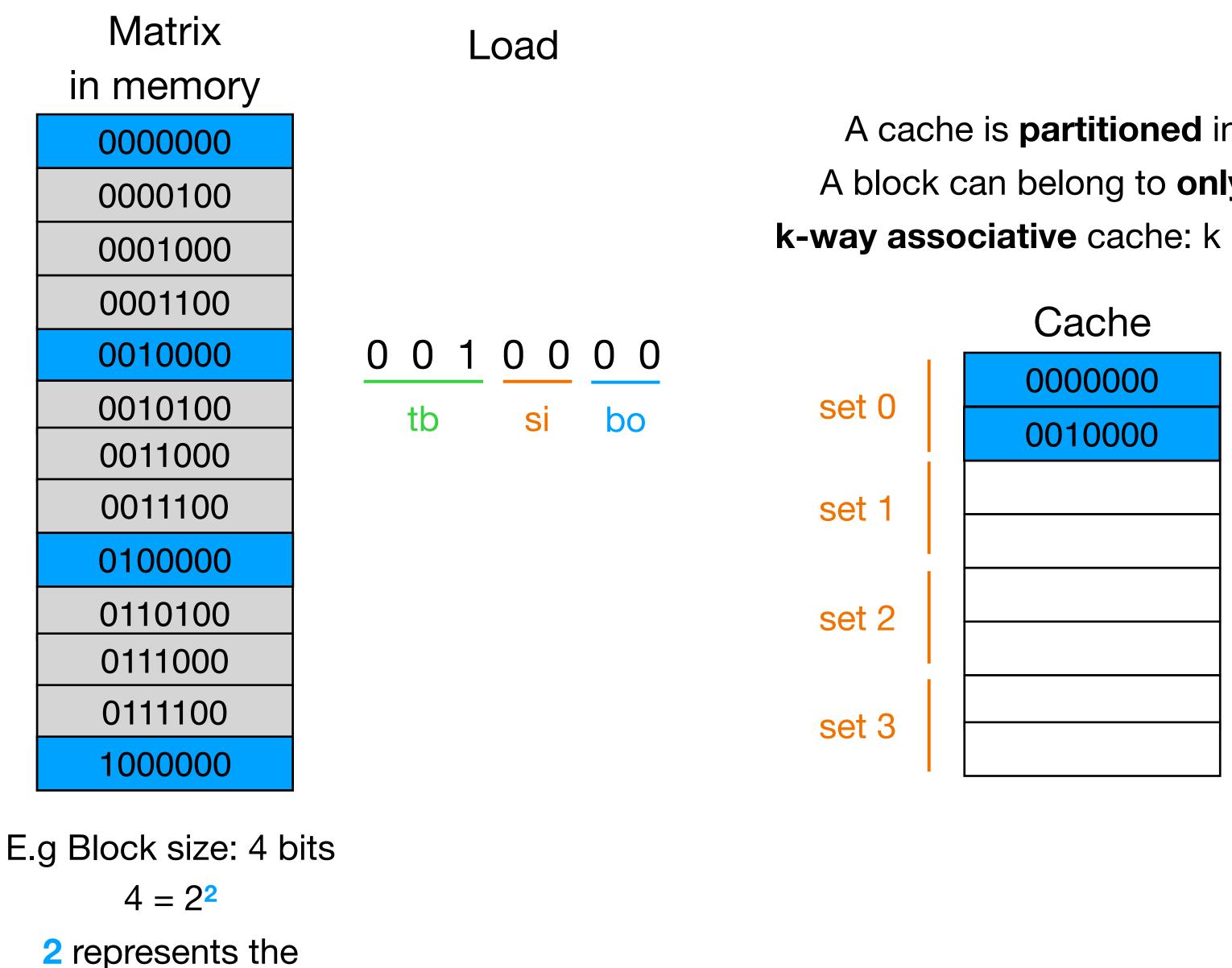


A cache is **partitioned** into **sets** A block can belong to **only one** set k-way associative cache: k lines per set E.g. 8 lines, 2-way associative nb of sets = 8/2 = 4 $4 = 2^{2}$

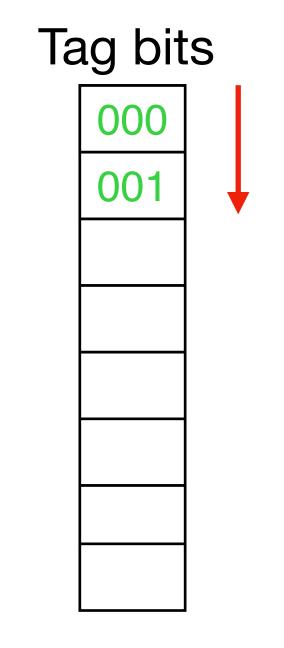




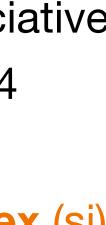


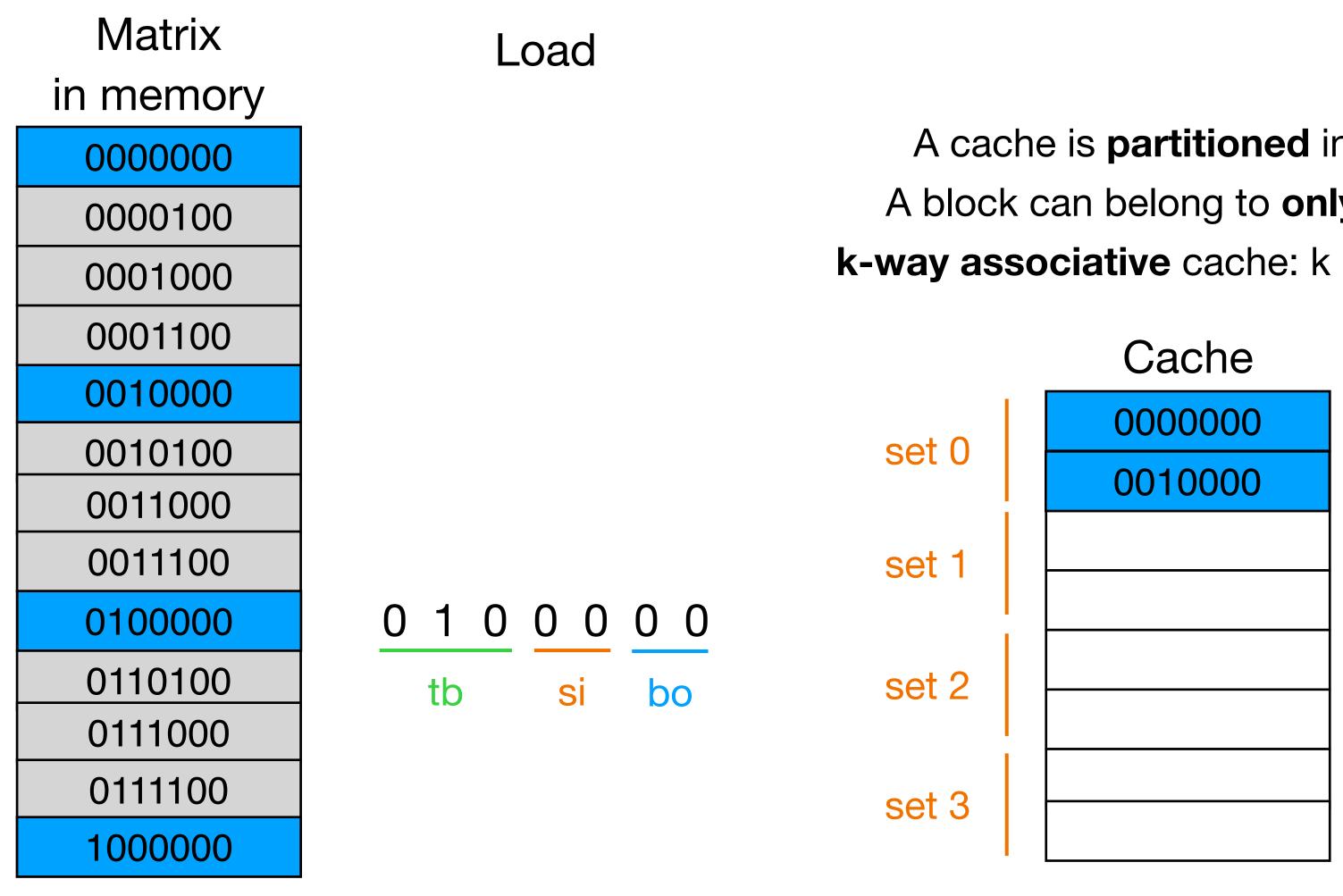


A cache is **partitioned** into **sets** A block can belong to **only one** set k-way associative cache: k lines per set E.g. 8 lines, 2-way associative nb of sets = 8/2 = 4 $4 = 2^{2}$



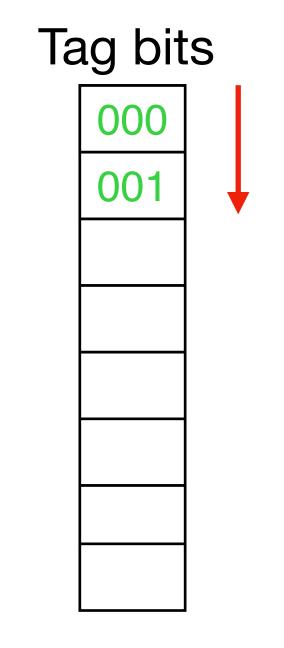




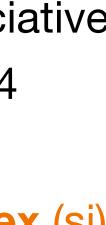


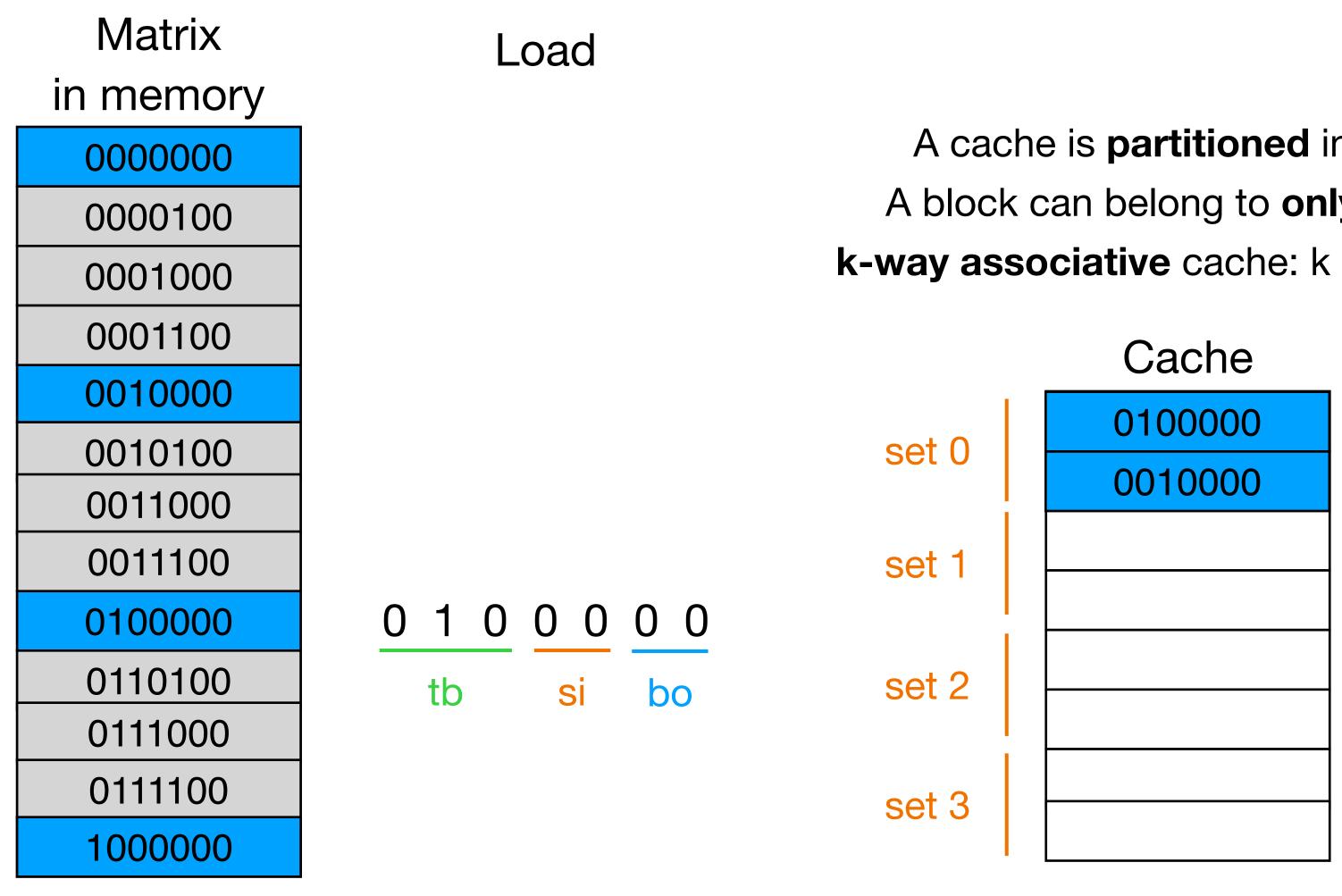
E.g Block size: 4 bits $4 = 2^{2}$ **2** represents the block offset (bo)

A cache is **partitioned** into **sets** A block can belong to **only one** set k-way associative cache: k lines per set E.g. 8 lines, 2-way associative nb of sets = 8/2 = 4 $4 = 2^{2}$



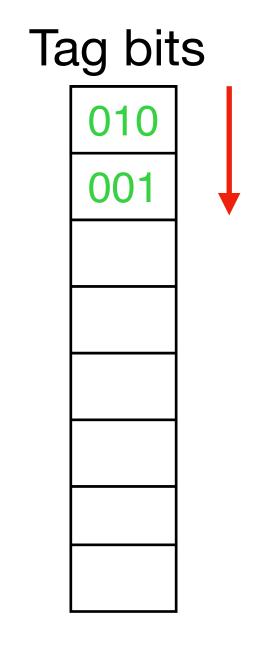




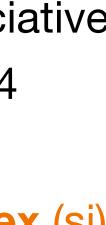


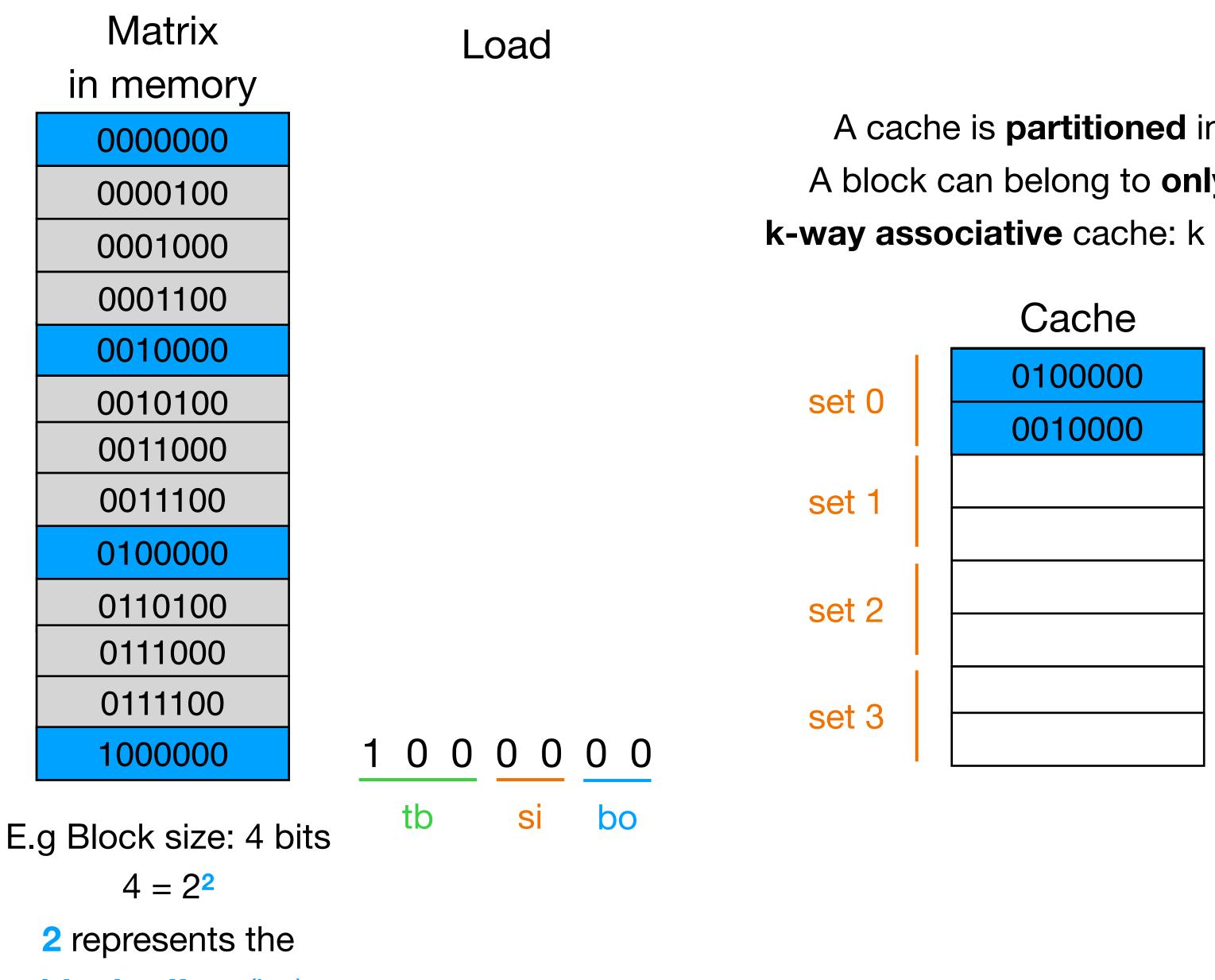
E.g Block size: 4 bits $4 = 2^{2}$ **2** represents the block offset (bo)

A cache is **partitioned** into **sets** A block can belong to **only one** set k-way associative cache: k lines per set E.g. 8 lines, 2-way associative nb of sets = 8/2 = 4 $4 = 2^{2}$

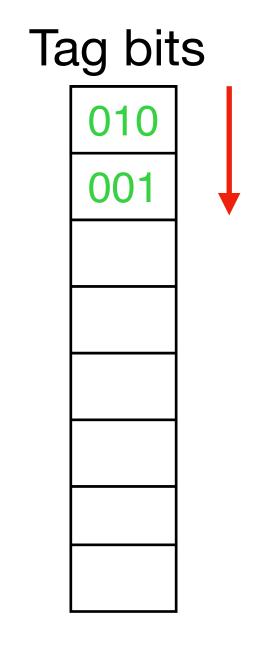




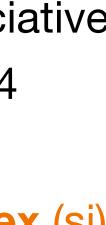


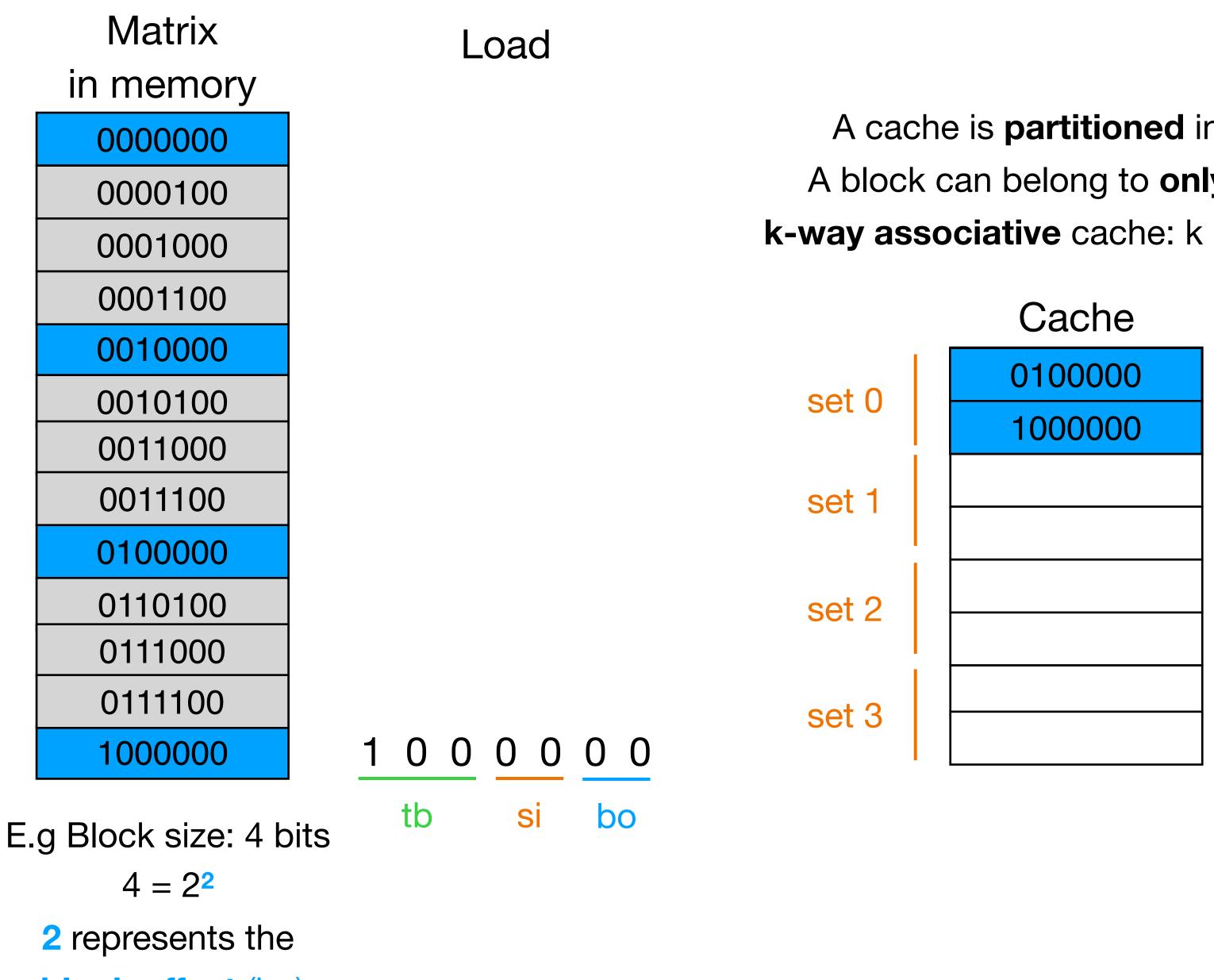


A cache is **partitioned** into **sets** A block can belong to **only one** set k-way associative cache: k lines per set E.g. 8 lines, 2-way associative nb of sets = 8 / 2 = 4 $4 = 2^{2}$

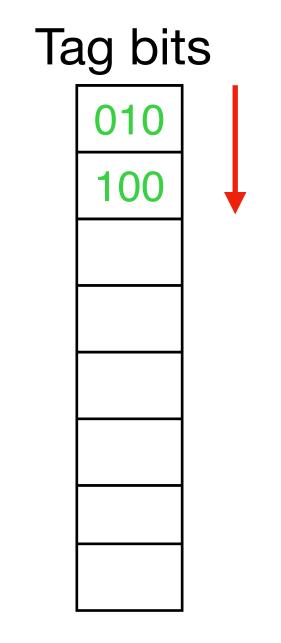




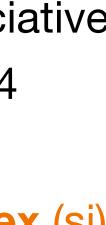


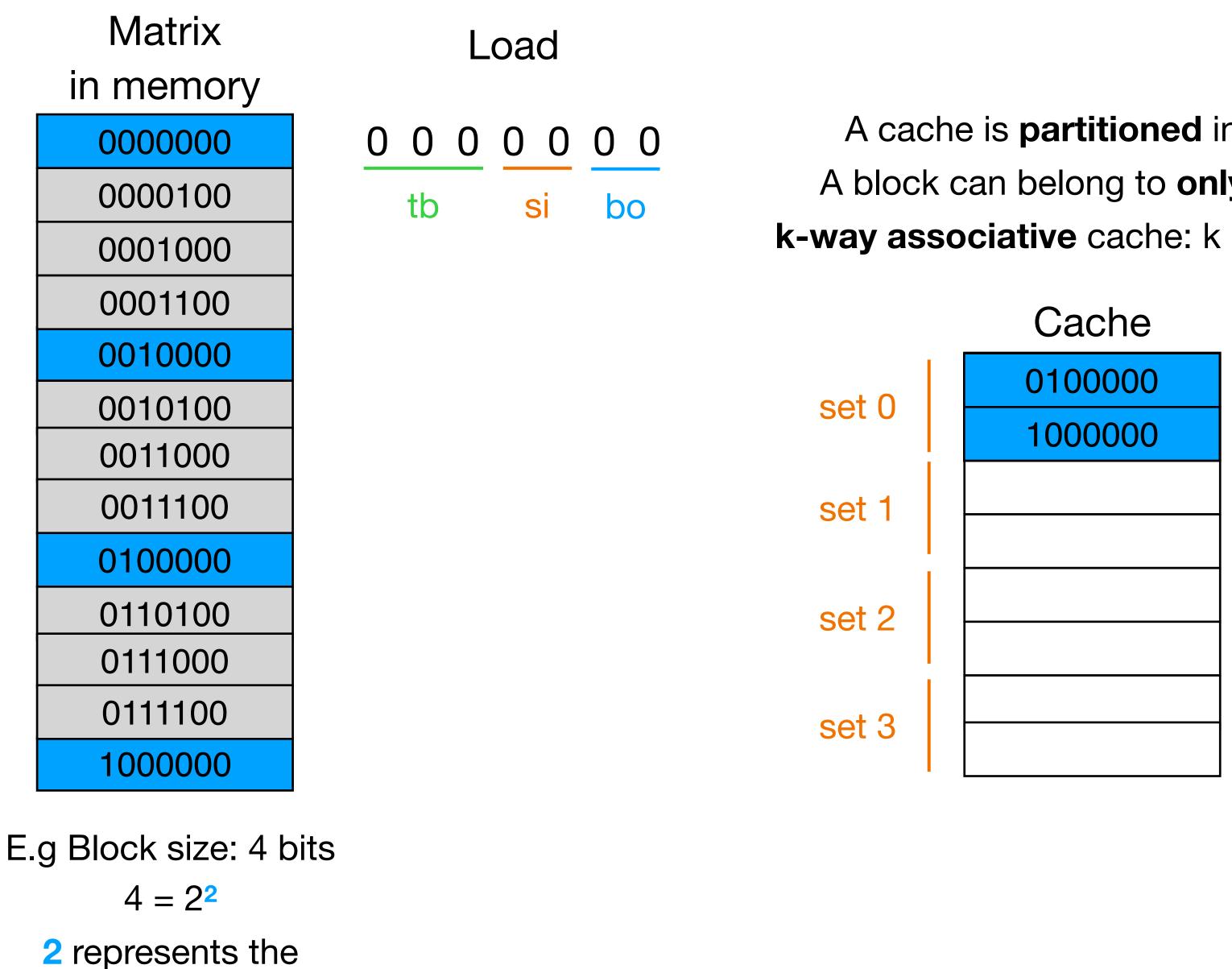


A cache is **partitioned** into **sets** A block can belong to **only one** set k-way associative cache: k lines per set E.g. 8 lines, 2-way associative nb of sets = 8 / 2 = 4 $4 = 2^{2}$

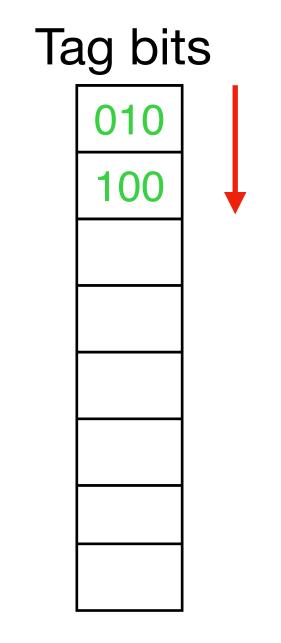




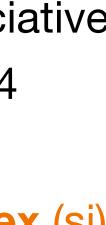




A cache is **partitioned** into **sets** A block can belong to **only one** set k-way associative cache: k lines per set E.g. 8 lines, 2-way associative nb of sets = 8/2 = 4 $4 = 2^{2}$



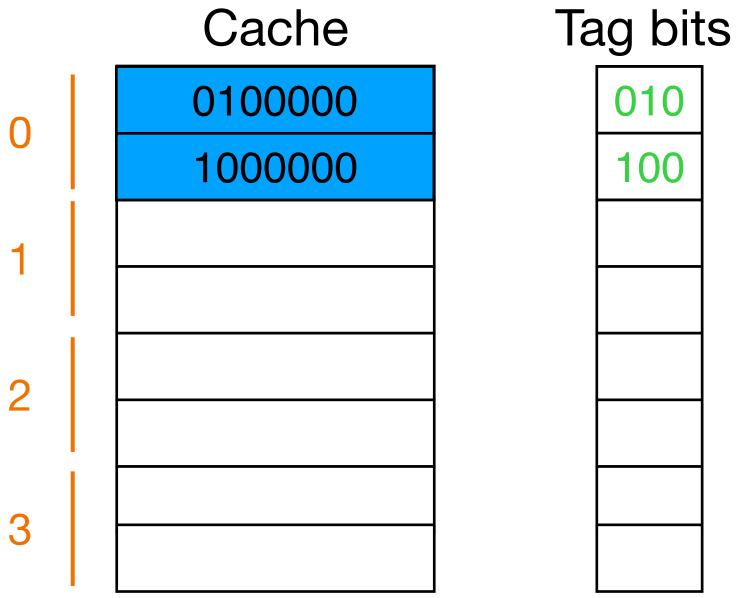




	Matrix	Load	
	in memory		_
	0000000	0000000	A ca
	0000100	tb si bo	A bloc
	0001000		k-way as
	0001100		
	0010000		
	0010100		set 0
	0011000		
	0011100		set 1
	0100000		
	0110100		set 2
	0111000		
	0111100		set 3
	1000000		
F	g Block size: 4 bi	ts	
	$4 = 2^2$		The distributio
	2 represents the		

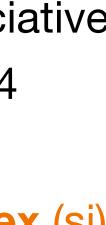
ache is **partitioned** into sets ck can belong to **only one** set associative cache: k lines per set E.g. 8 lines, 2-way associative nb of sets = 8/2 = 4 $4 = 2^{2}$

2 represents the set index (si)



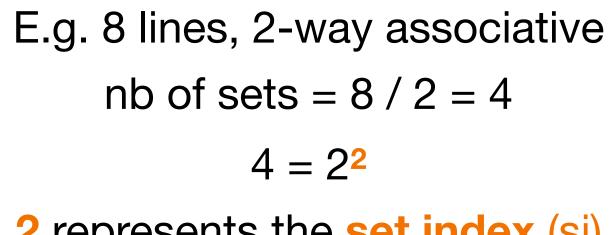
on is not even, we used **only one set**



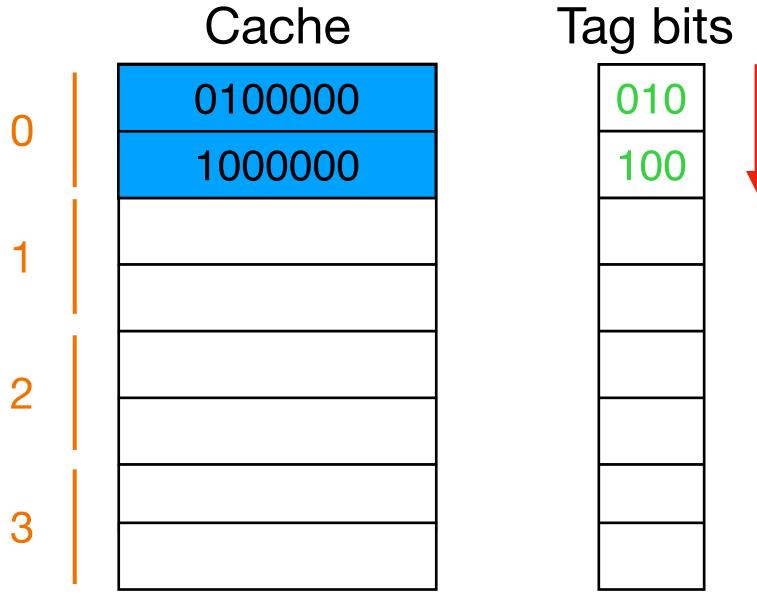


. Matrix	Load	
in memory		٨
0000000	0000000	A ca
0000100	tb si bo	A bloc
0001000		k-way as
0001100		
0010000		
0010100		set 0
0011000		
0011100		set 1
0100000		
0110100		set 2
0111000		
0111100		set 3
1000000		
E.g Block size: 4 b	oits	
$4 = 2^{2}$		The distributio
2 represents the		It will generate
block offset (bo)	it tim genorate

ache is **partitioned** into sets ck can belong to **only one** set associative cache: k lines per set



2 represents the set index (si)



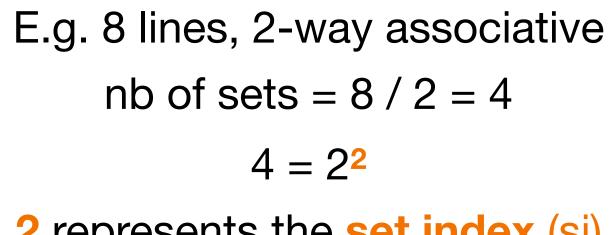
on is not even, we used **only one set**

e a lot of cache misses (conflict miss)

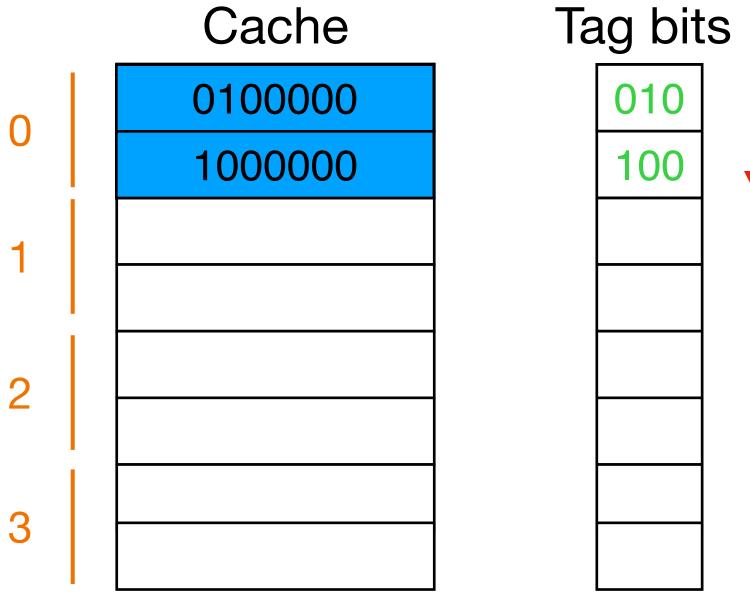


Matrix	Load	
in memory 0000000	0000000	A ca
0000100	tb si bo	A bloc
0001000		k-way as
0001100		
0010000		
0010100		set 0
0011000		
0011100		set 1
0100000		
0110100		set 2
0111000		
0111100		set 3
1000000		
E.g Block size: 4 b	its	
4 = 2 <mark>2</mark>		The distribution
2 represents the		It will generate
block offset (bo		This constar

ache is **partitioned** into **sets** ck can belong to only one set associative cache: k lines per set



2 represents the set index (si)



on is not even, we used **only one set**

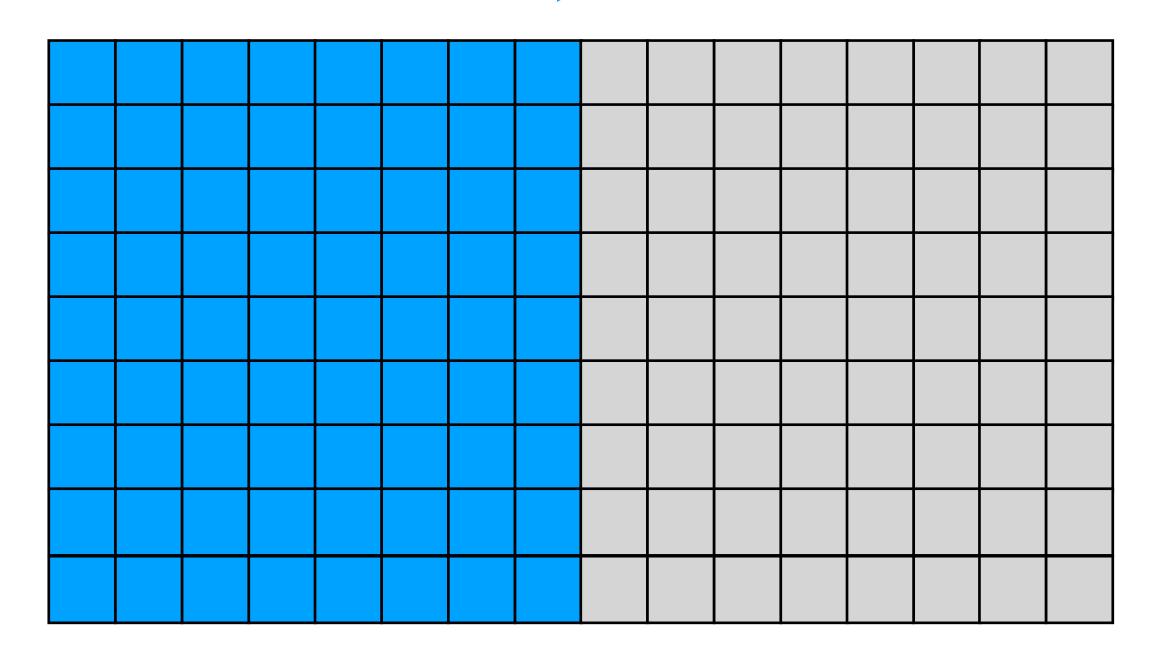
e a lot of cache misses (conflict miss)

ant stride is called the critical stride



n elements (variable)

8 int64s - 64 bytes

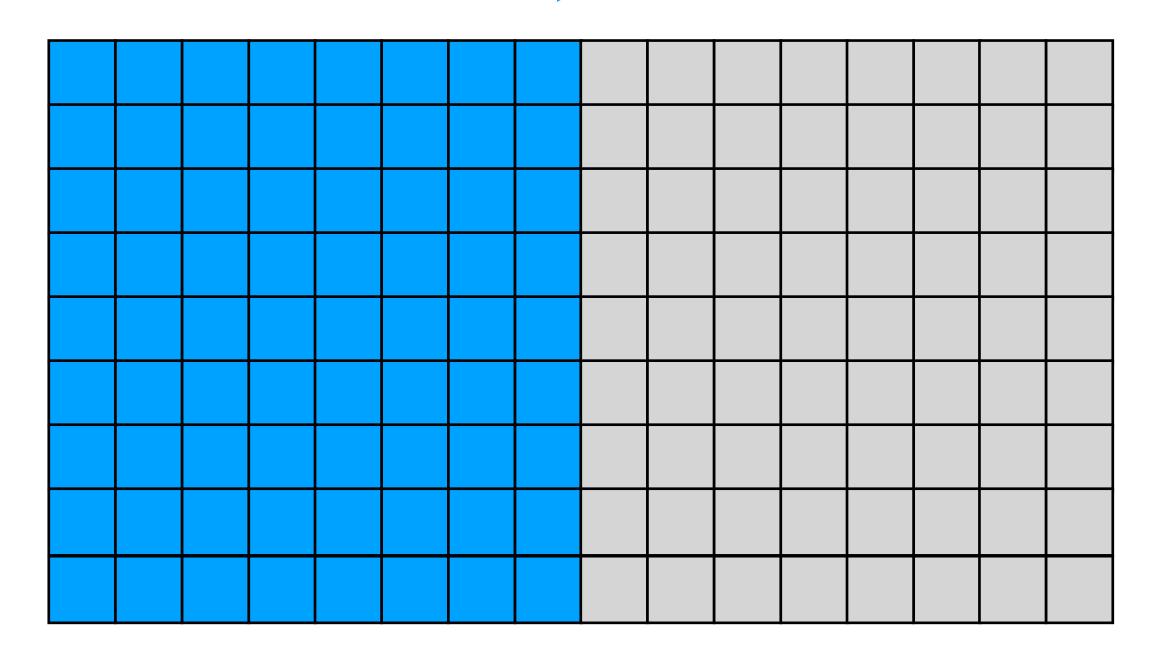




• Critical stride = nb sets x cache line size

n elements (variable)

8 int64s - 64 bytes

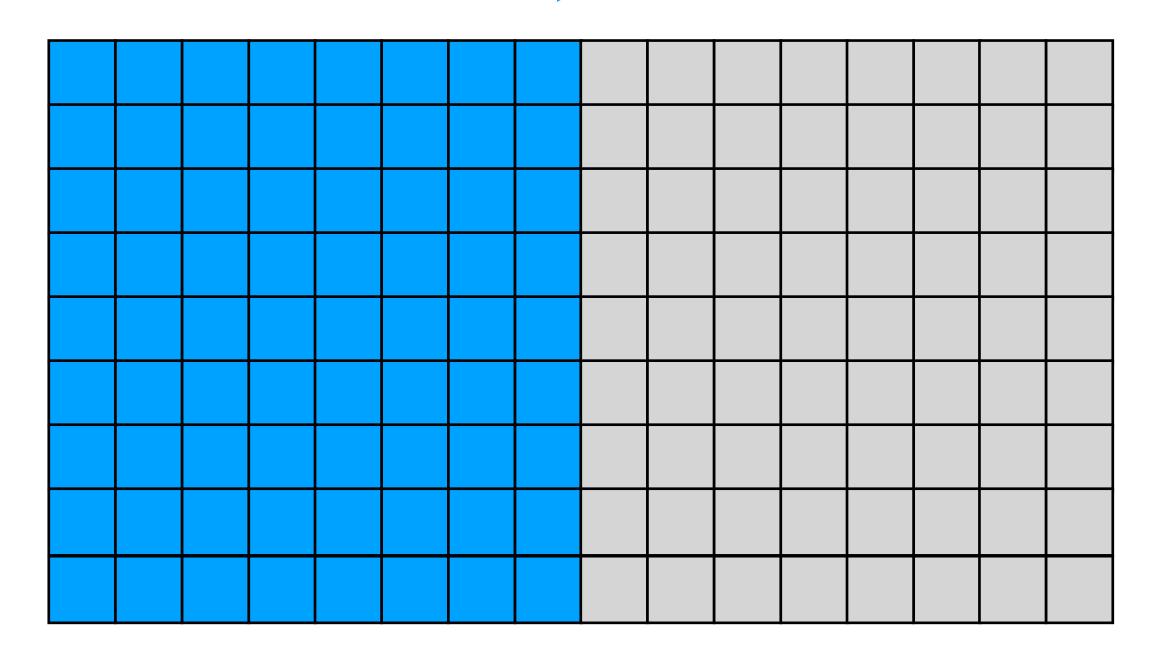




- Critical stride = nb sets x cache line size
- Example with an Intel Core i5-7300:
 - Cache line = 64 bytes
 - 32 KB, 8-way set associative, 64 sets
 - Critical stride = $64 \times 64 = 4 \text{ KB}$

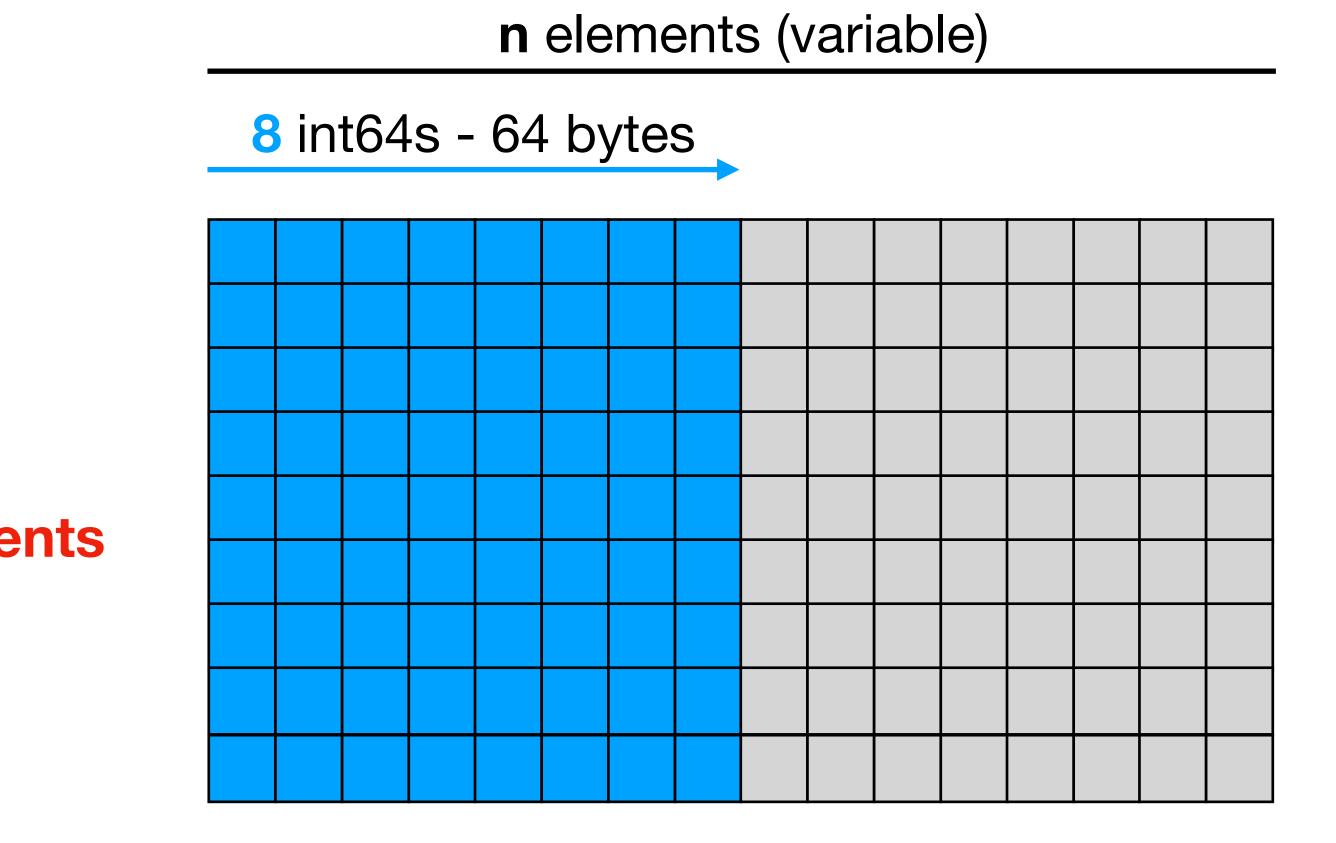
n elements (variable)

8 int64s - 64 bytes



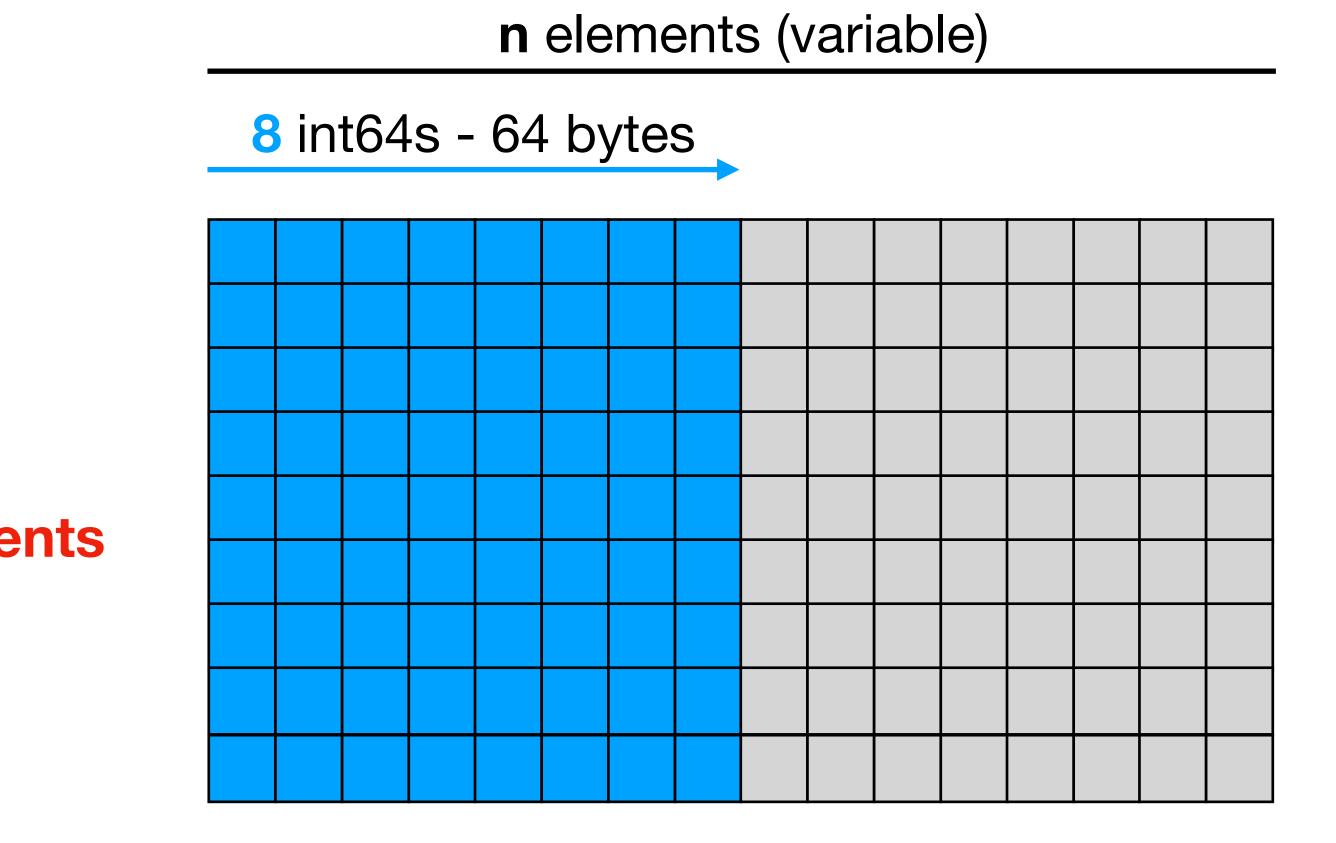


- Critical stride = nb sets x cache line size
- Example with an Intel Core i5-7300:
 - Cache line = 64 bytes
 - 32 KB, 8-way set associative, 64 sets
 - Critical stride = $64 \times 64 = 4 \text{ KB}$
- We reach a critical stride with n = **512 elements**



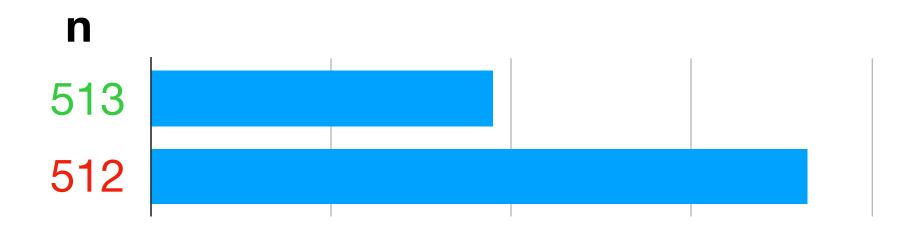


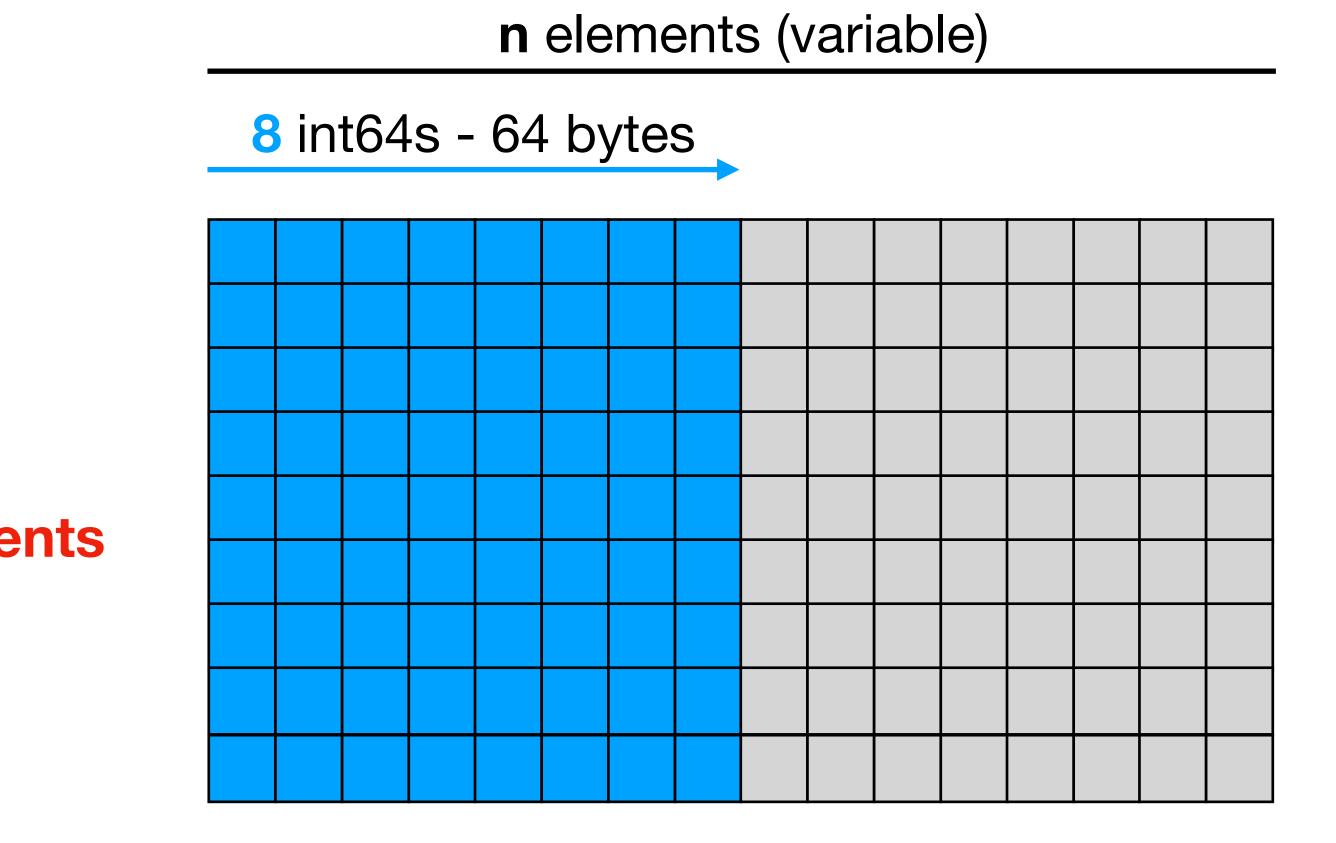
- Critical stride = nb sets x cache line size
- Example with an Intel Core i5-7300:
 - Cache line = 64 bytes
 - 32 KB, 8-way set associative, 64 sets
 - Critical stride = $64 \times 64 = 4 \text{ KB}$
- We reach a critical stride with n = **512 elements**
- If n = 512, we are going to use **1** set only





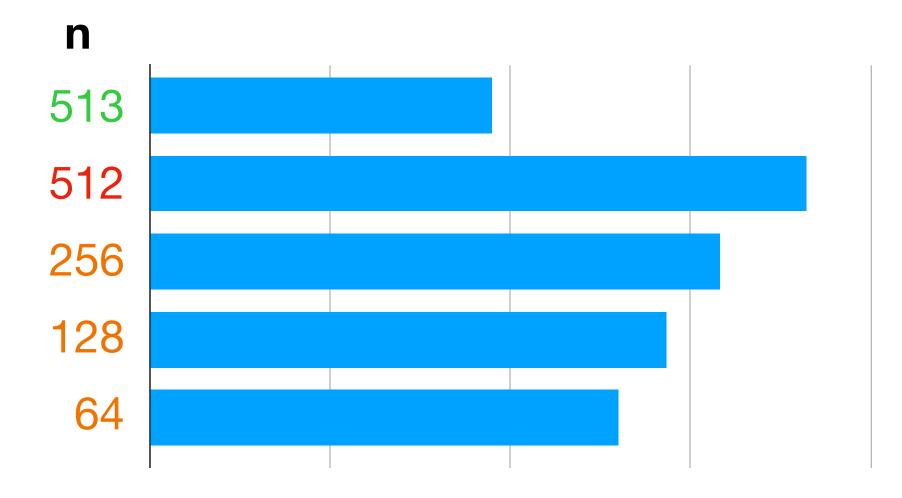
- Critical stride = nb sets x cache line size
- Example with an Intel Core i5-7300:
 - Cache line = 64 bytes
 - 32 KB, 8-way set associative, 64 sets
 - Critical stride = $64 \times 64 = 4 \text{ KB}$
- We reach a critical stride with n = **512** elements
- If n = 512, we are going to use 1 set only

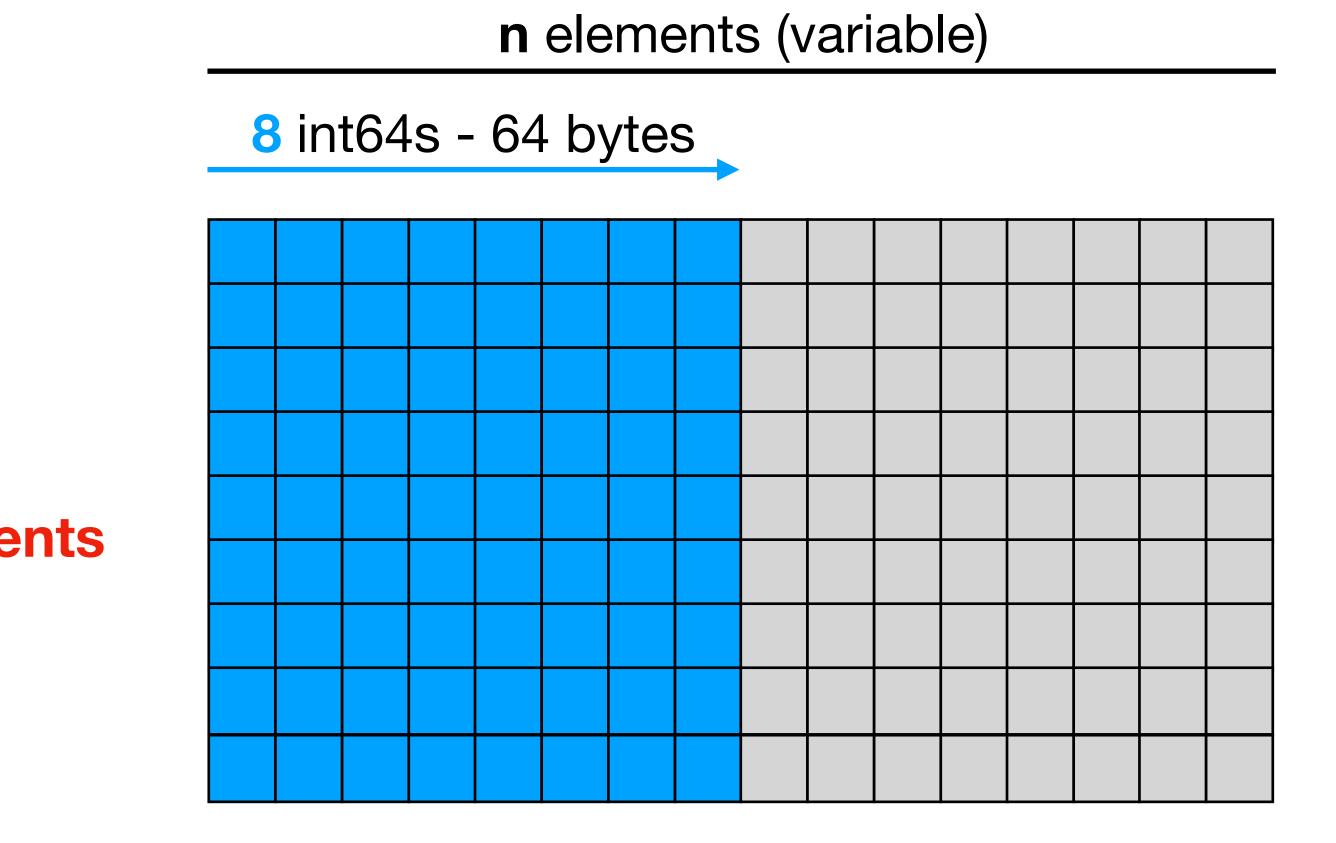






- Critical stride = nb sets x cache line size
- Example with an Intel Core i5-7300:
 - Cache line = 64 bytes
 - 32 KB, 8-way set associative, 64 sets
 - Critical stride = $64 \times 64 = 4 \text{ KB}$
- We reach a critical stride with n = **512 elements**
- If n = 512, we are going to use 1 set only













CPU caches are partitioned





CPU caches are partitioned

• Depending on my data, my application can occupy a fraction of the cache only





CPU caches are partitioned

Critical stride

• Depending on my data, my application can occupy a fraction of the cache only

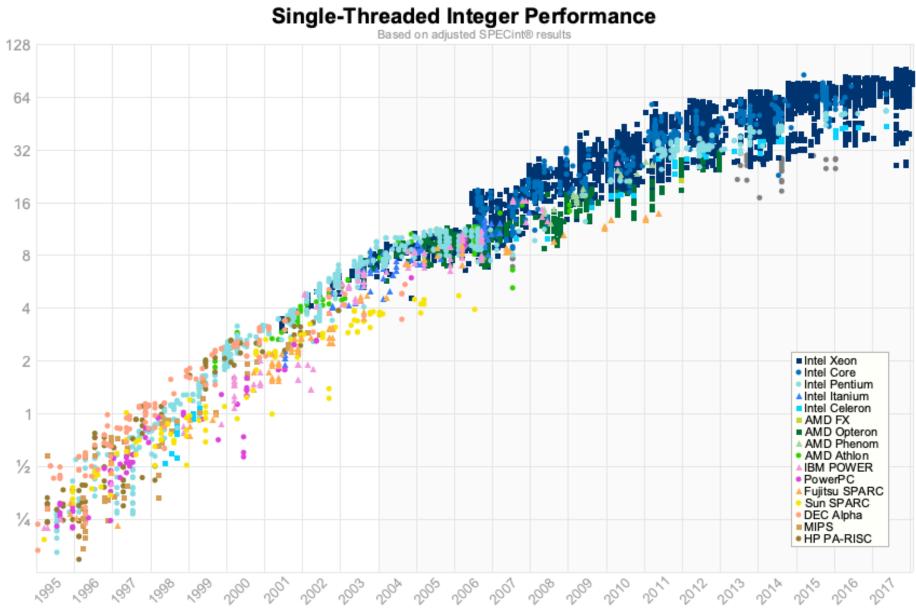


CPU Architecture Locality of Reference **Data-Oriented Design** Caching Pitfall Concurrency

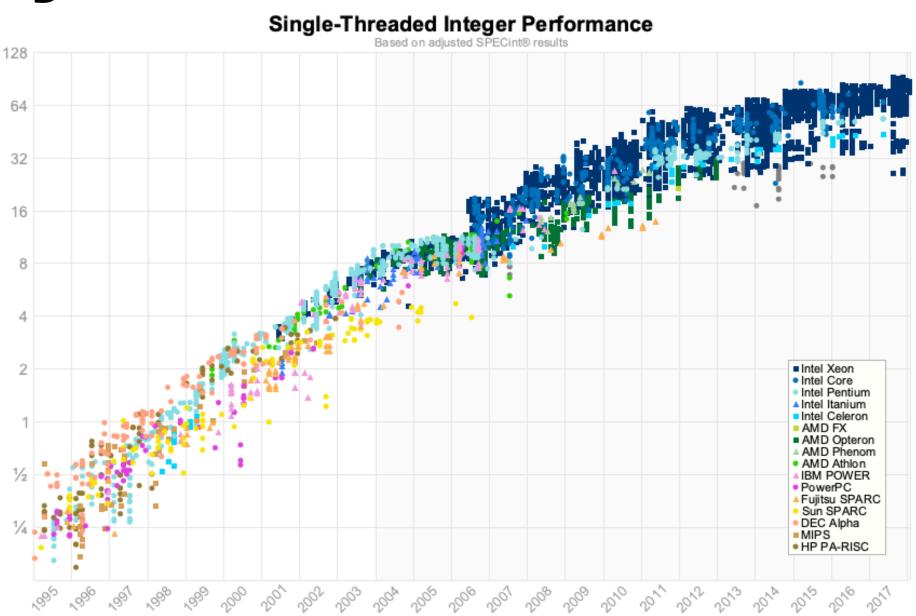


Why Concurrency?



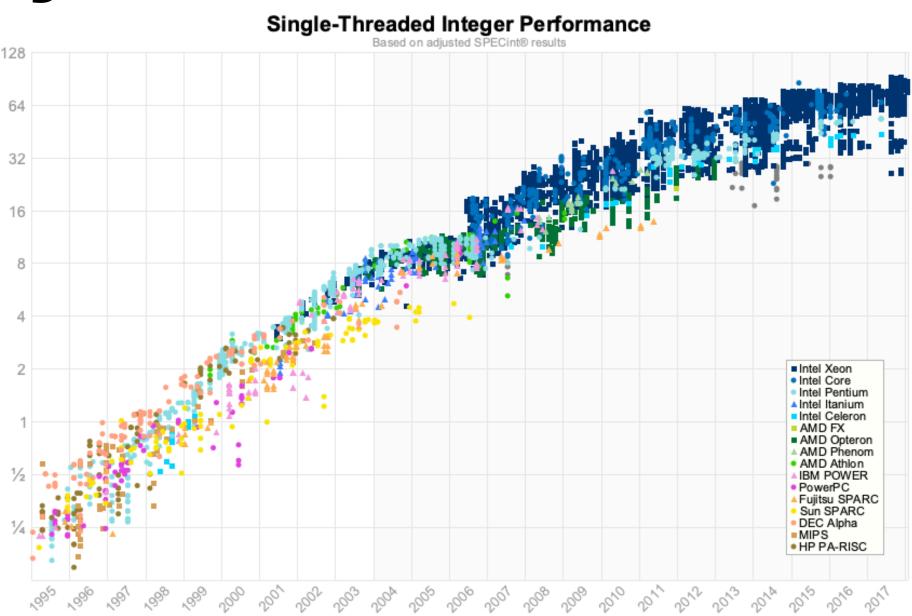






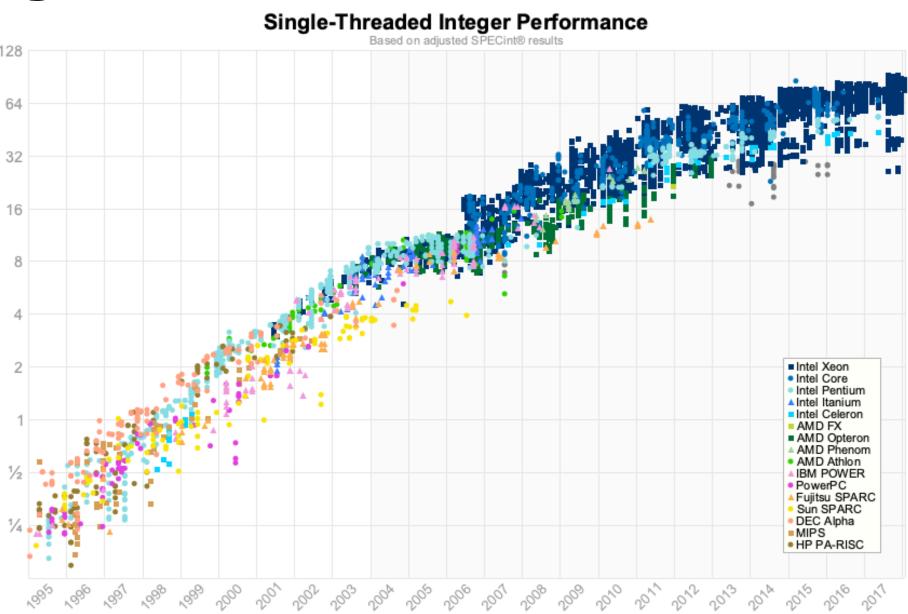
 Instead of focusing on clock speed, vendors focus on multicores and hyperthreading architectures





- Instead of focusing on clock speed, vendors focus on multicores and hyperthreading architectures
- The free lunch is over Herb Sutter, 2005





- Instead of focusing on clock speed, vendors focus on multicores and hyperthreading architectures
- The free lunch is over Herb Sutter, 2005
- We cannot rely solely on the hardware to make our programs faster **Concurrency** is the next major revolution in how we write software



```
type Struct struct {
    n int
}
var <mark>result</mark> int
func BenchmarkIteration(b *testing.B) {
    structA := Struct{} // Initialization
    structB := Struct{} // Initialization
    wg := sync.WaitGroup{}
    b.ResetTimer()
    for i := 0; i < b.N; i++ {</pre>
        wg.Add( delta: 2)
        go func() { // Spin up first goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structA.n += j
             }
            wg.Done()
        }()
        go func() { // Spin up second goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structB.n += j
             }
            wg.Done()
        }()
        wg.Wait() // Wait
        result = structA.n + structB.n // Aggregate
}
```



```
type Struct struct {
    n int
var <mark>result</mark> int
func BenchmarkIteration(b *testing.B) {
    structA := Struct{} // Initialization
    structB := Struct{} // Initialization
    wg := sync.WaitGroup{}
    b.ResetTimer()
    for i := 0; i < b.N; i++ {</pre>
        wg.Add( delta: 2)
        go func() { // Spin up first goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structA.n += j
             }
            wg.Done()
        }()
        go func() { // Spin up second goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structB.n += j
             }
            wg.Done()
        }()
        wg.Wait() // Wait
        result = structA.n + structB.n // Aggregate
}
```



```
type Struct struct {
    n int
}
var <mark>result</mark> int
func BenchmarkIteration(b *testing.B) {
    structA := Struct{} // Initialization
    structB := Struct{} // Initialization
    wg := sync.WaitGroup{}
    b.ResetTimer()
    for i := 0; i < b.N; i++ {</pre>
        wg.Add( delta: 2)
        go func() { // Spin up first goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structA.n += j
             }
            wg.Done()
        }()
        go func() { // Spin up second goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structB.n += j
             }
            wg.Done()
        }()
        wg.Wait() // Wait
        result = structA.n + structB.n // Aggregate
}
```



```
type Struct struct {
    n int
}
var <mark>result</mark> int
func BenchmarkIteration(b *testing.B) {
    structA := Struct{} // Initialization
    structB := Struct{} // Initialization
    wg := sync.WaitGroup{}
    b.ResetTimer()
    for i := 0; i < b.N; i++ {</pre>
        wg.Add( delta: 2)
        go func() { // Spin up first goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structA.n += j
             }
            wg.Done()
        }()
        go func() { // Spin up second goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structB.n += j
             }
            wg.Done()
        }()
        wg.Wait() // Wait
        result = structA.n + structB.n // Aggregate
```

}



```
type Struct struct {
    n int
var <mark>result</mark> int
func BenchmarkIteration(b *testing.B) {
    structA := Struct{} // Initialization
    structB := Struct{} // Initialization
    wg := sync.WaitGroup{}
    b.ResetTimer()
    for i := 0; i < b.N; i++ {</pre>
        wg.Add( delta: 2)
        go func() { // Spin up first goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structA.n += j
             }
            wg.Done()
        }()
        go func() { // Spin up second goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structB.n += j
             }
            wg.Done()
        wg.Wait() // Wait
        result = structA.n + structB.n // Aggregate
```

}

}



```
type Struct struct {
    n int
var <mark>result</mark> int
func BenchmarkIteration(b *testing.B) {
    structA := Struct{} // Initialization
    structB := Struct{} // Initialization
    wg := sync.WaitGroup{}
    b.ResetTimer()
    for i := 0; i < b.N; i++ {</pre>
        wg.Add( delta: 2)
        go func() { // Spin up first goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structA.n += j
             }
            wg.Done()
        }()
        go func() { // Spin up second goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structB.n += j
             }
            wg.Done()
        }()
        wg.Wait() // Wait
        result = structA.n + structB.n // Aggregate
```

}

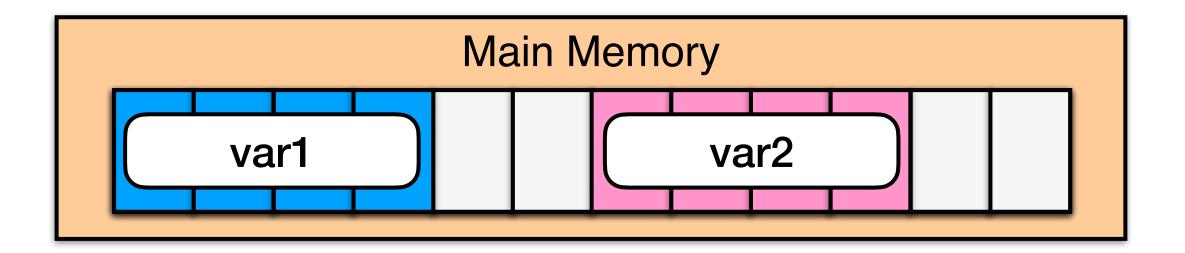
}

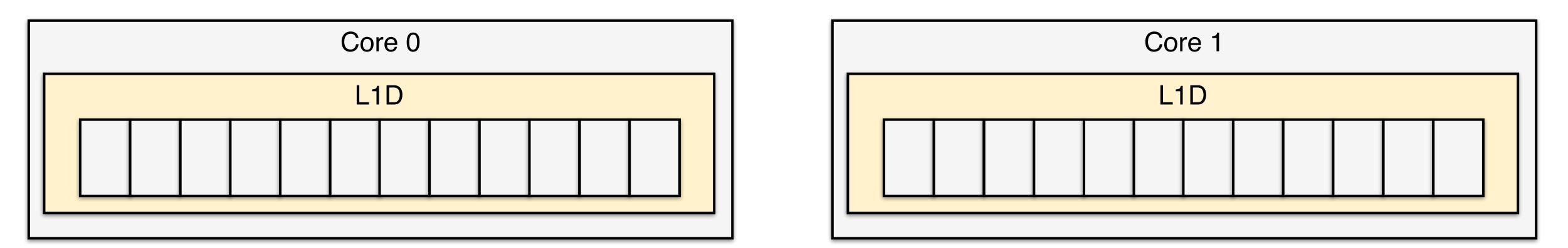


```
type Struct struct {
    n int
}
var <mark>result</mark> int
func BenchmarkIteration(b *testing.B) {
    structA := Struct{} // Initialization
    structB := Struct{} // Initialization
    wg := sync.WaitGroup{}
    b.ResetTimer()
    for i := 0; i < b.N; i++ {</pre>
        wg.Add( delta: 2)
        go func() { // Spin up first goroutine
             for j := 0; j < iteration; j++ {</pre>
                 structA.n += j
             }
            wg.Done()
        }()
        go func() { // Spin up second goroutine
             for j := 0; j < iteration; j++ {</pre>
                 structB.n += j
             }
            wg.Done()
        }()
        wg.Wait() // Wait
        result = structA.n + structB.n // Aggregate
}
```

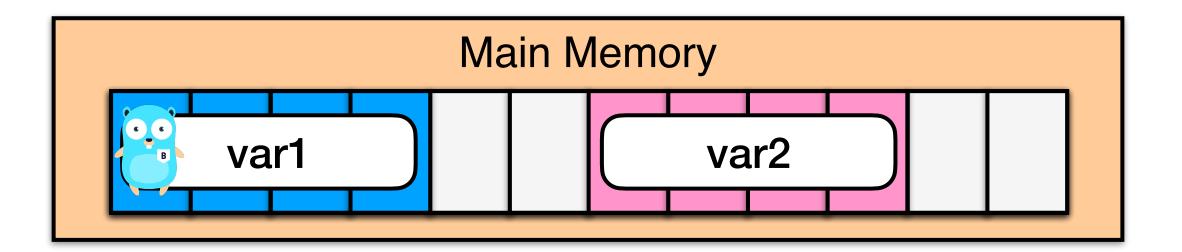
Race-free implementation!

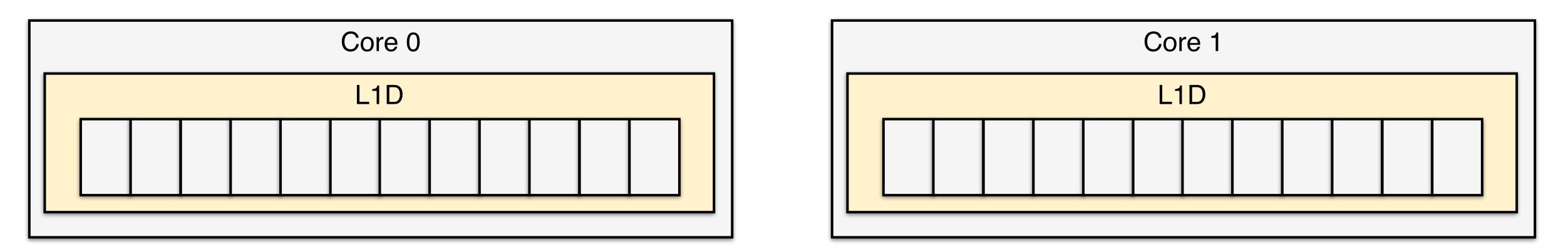




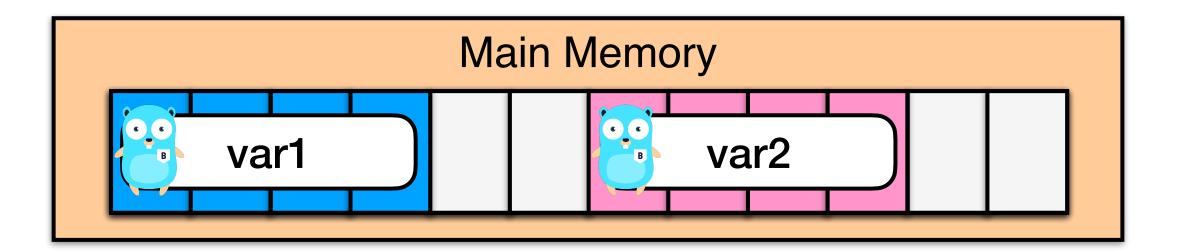


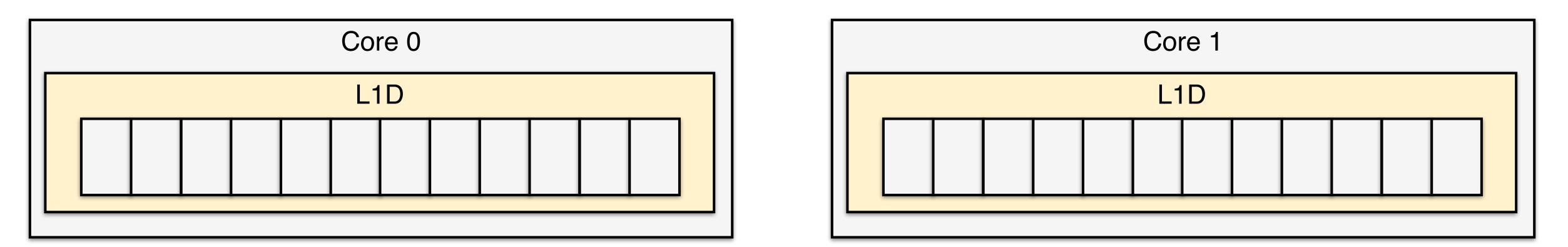




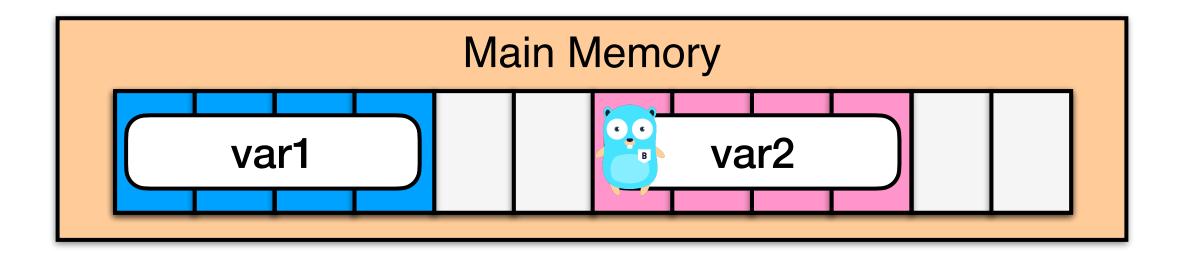


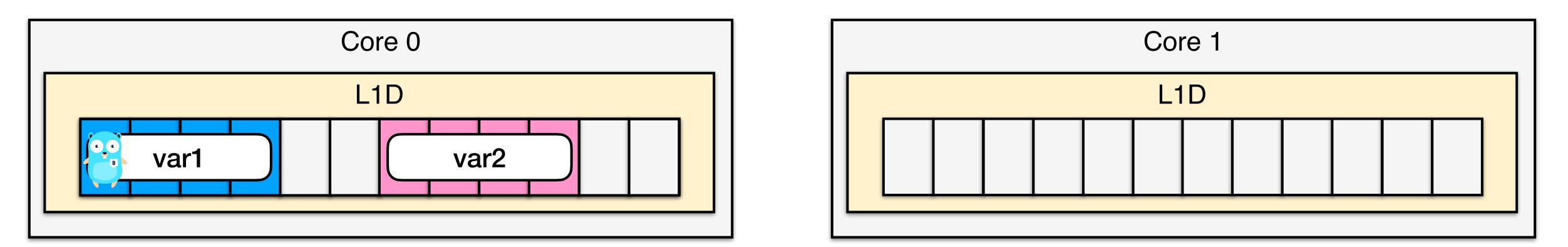




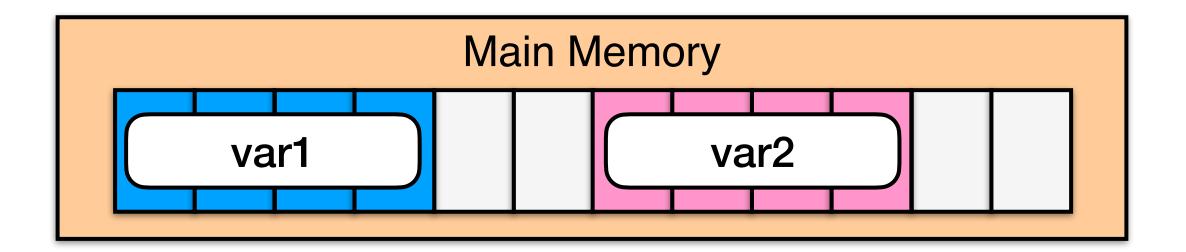


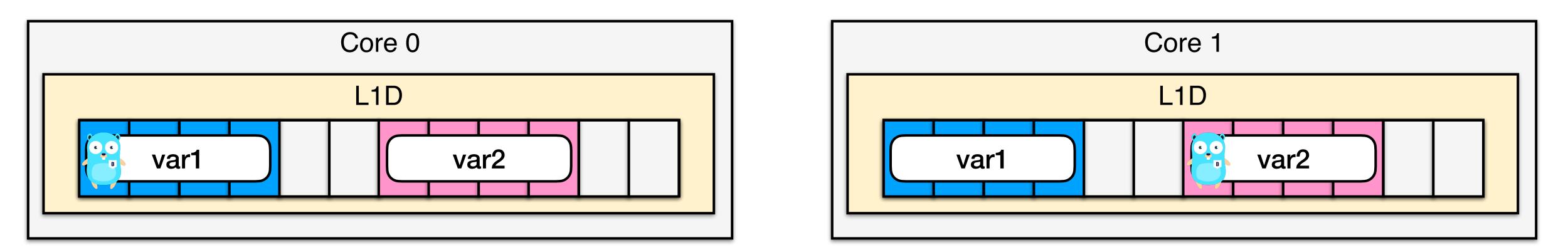




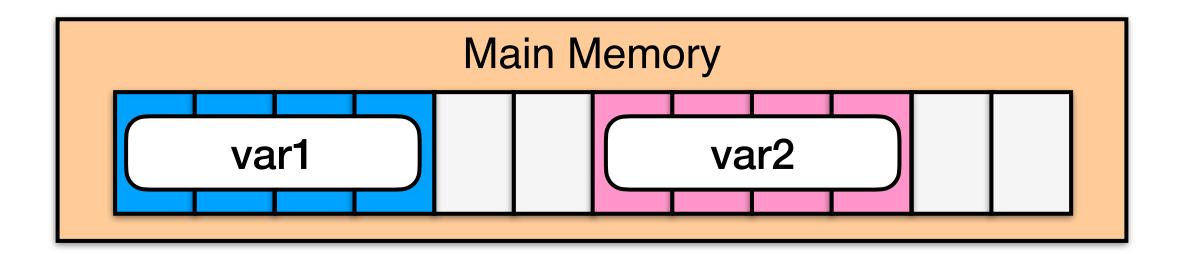


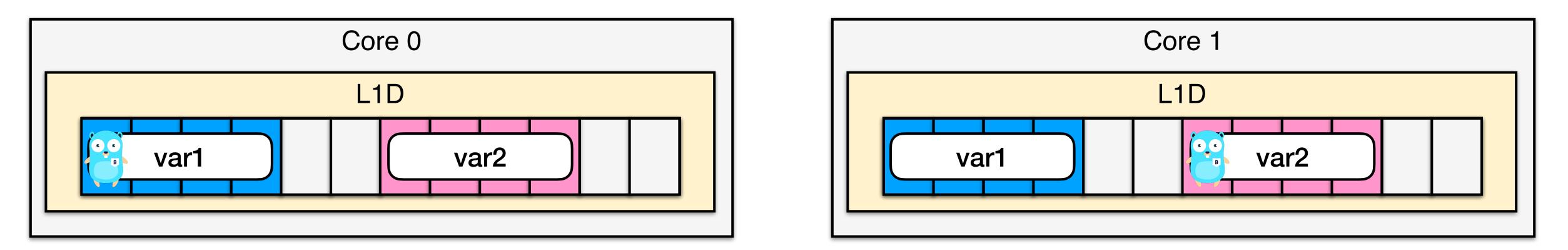






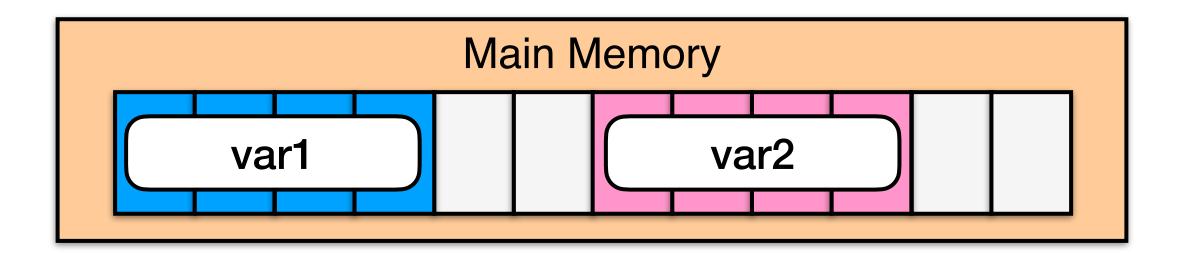


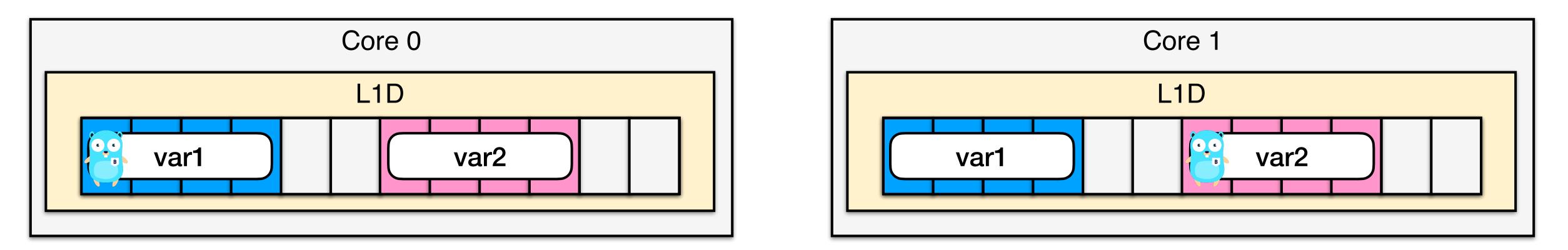




• What if both goroutines want to update their own lines

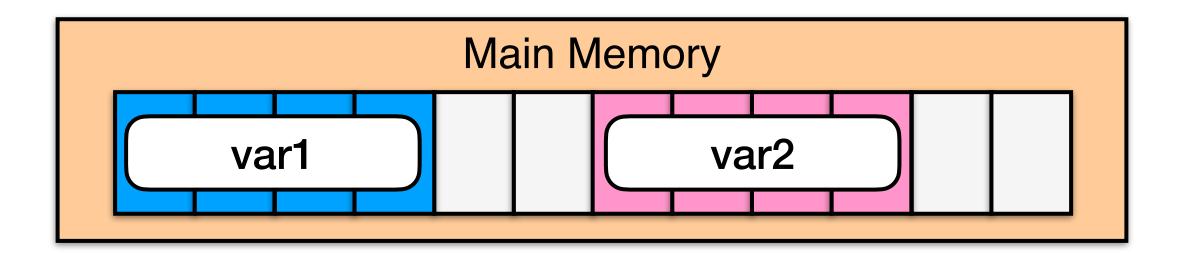


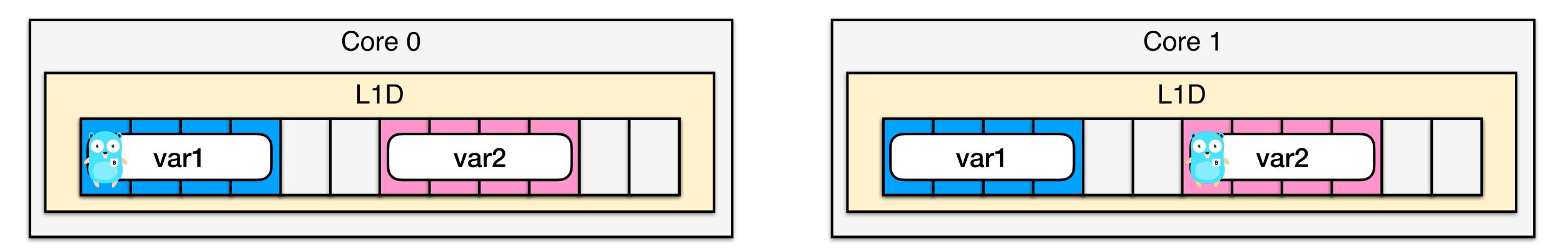




- What if both goroutines want to update their own lines
- The CPU must guarantee cache coherency

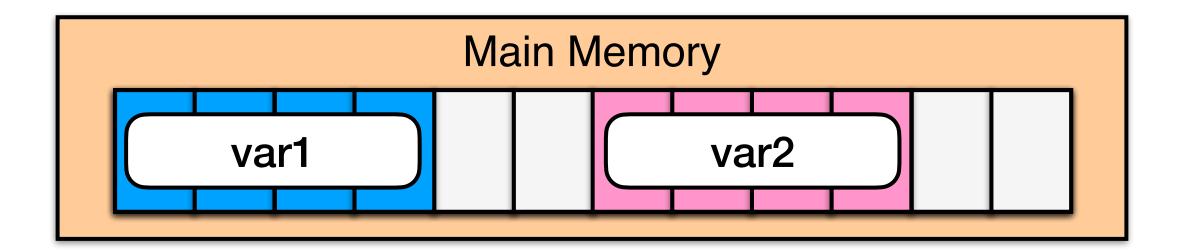


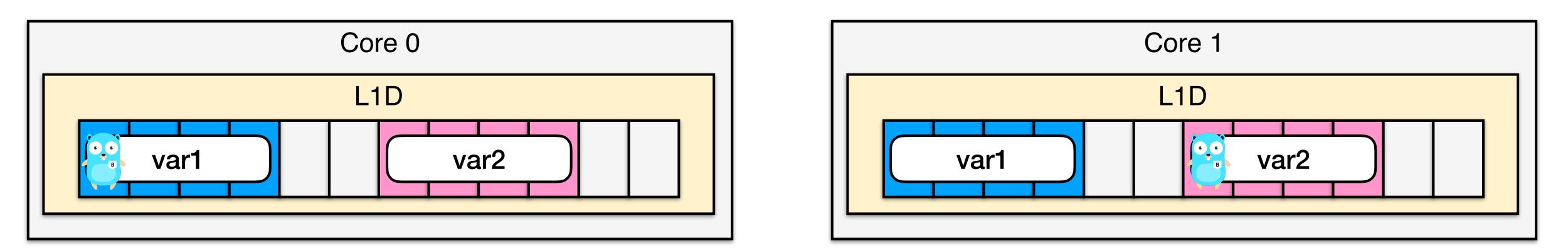




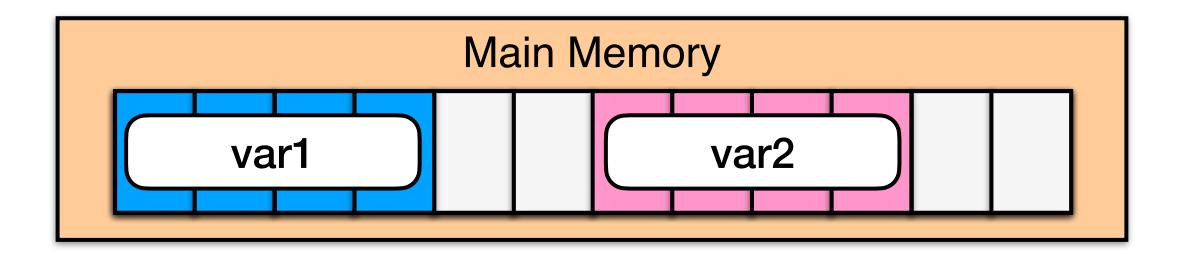
- What if both goroutines want to update their own lines
- The CPU must guarantee cache coherency
- MESI protocol (Modified, Exclusive, Shared, Invalid) ullet

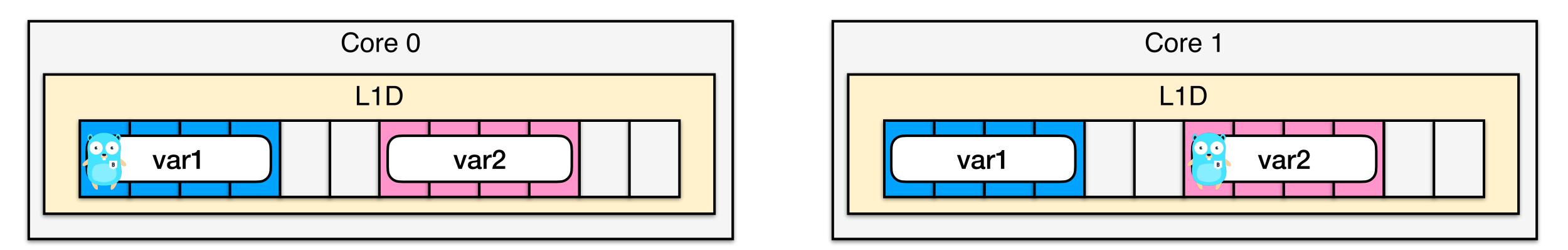




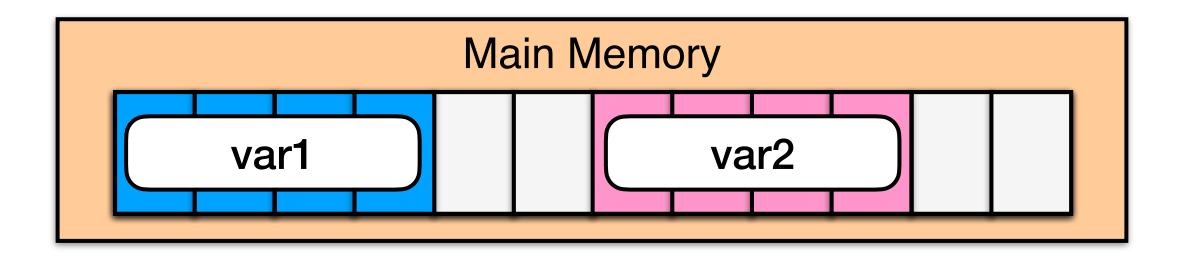


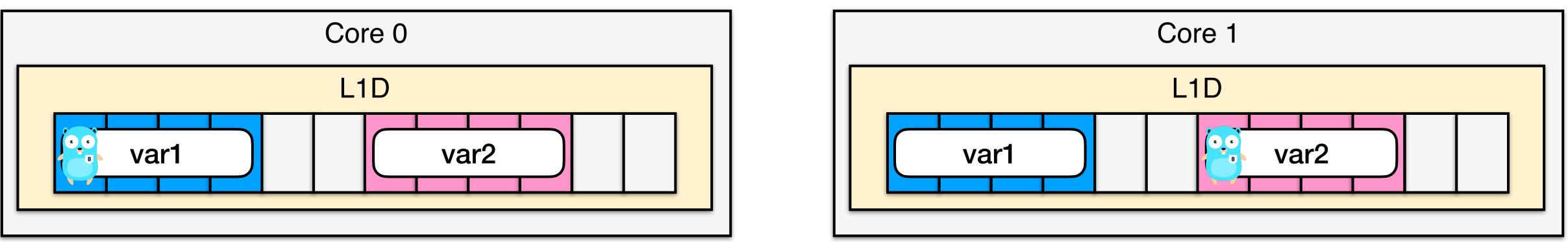




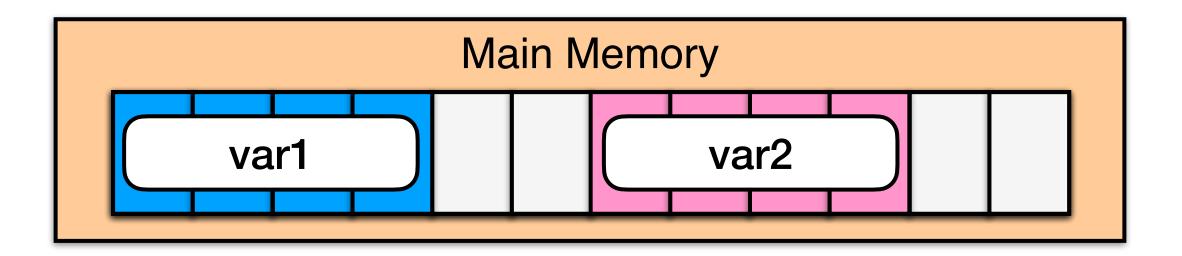


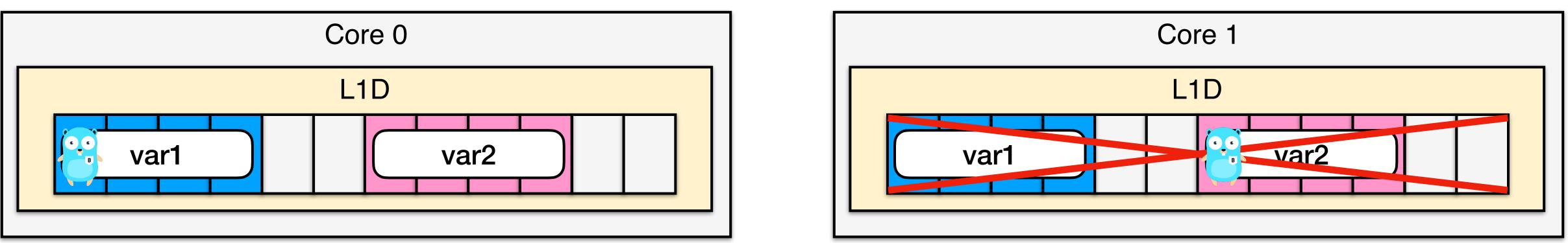




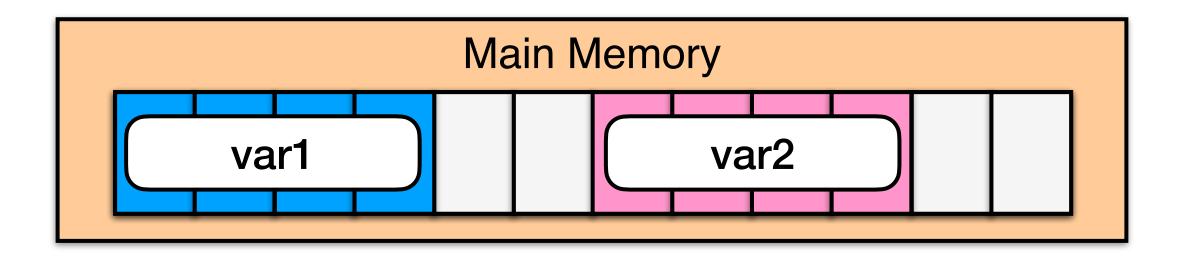


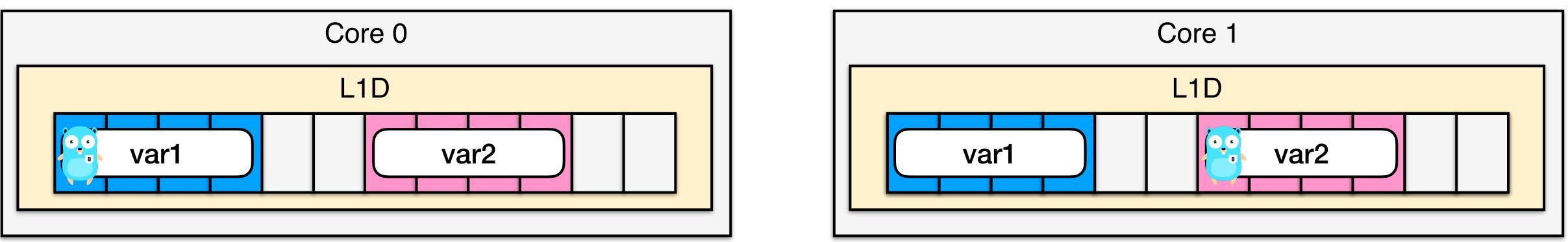




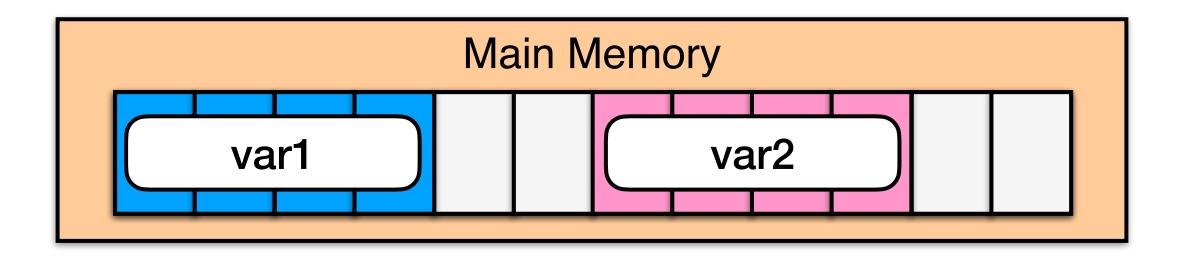


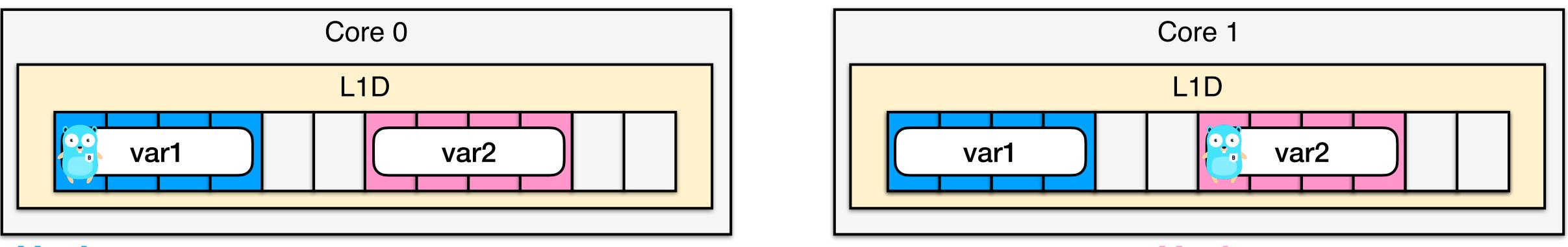






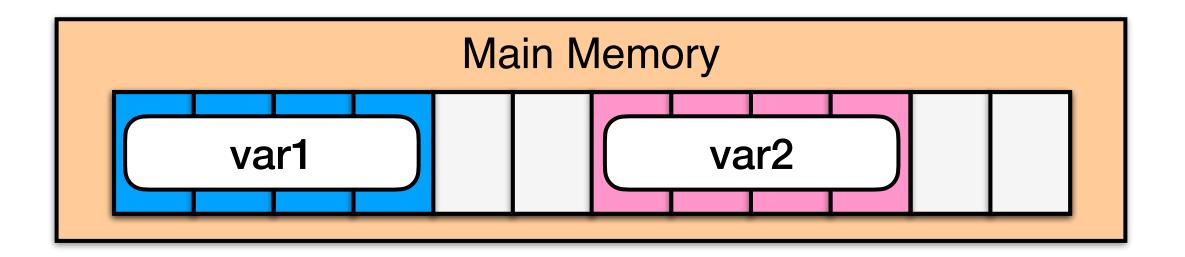


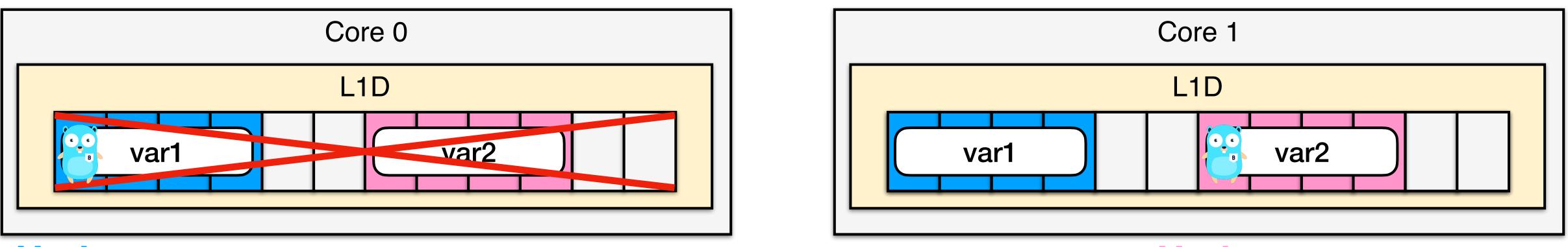




• Why does it matter?

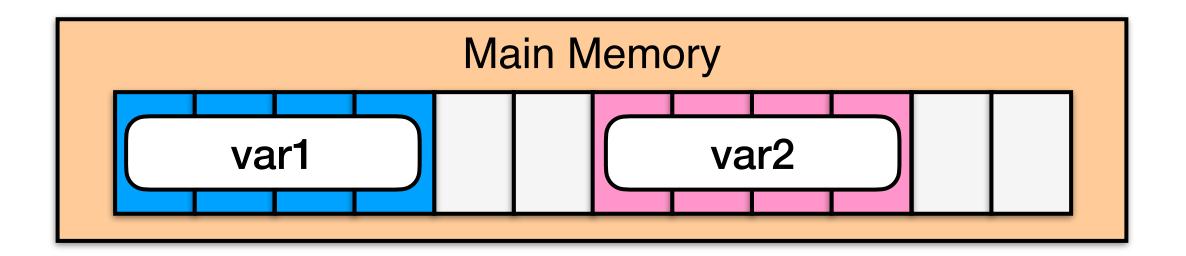


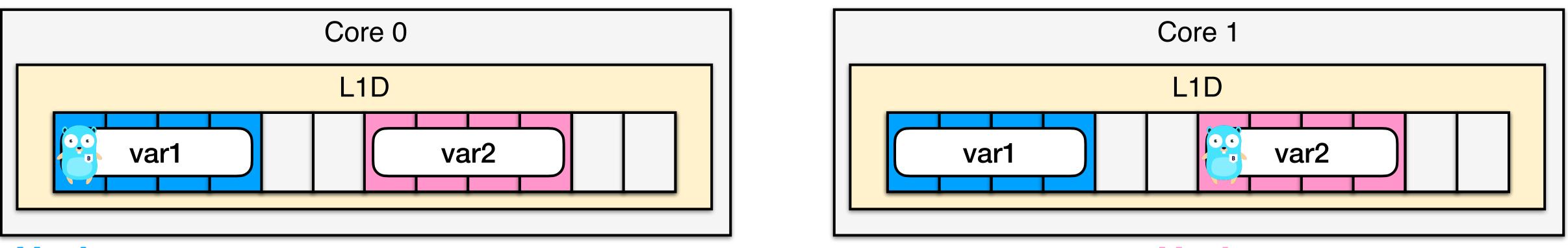




• Why does it matter?

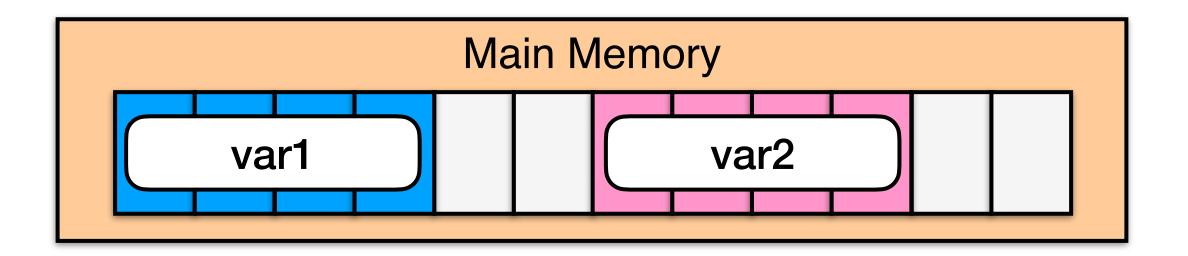


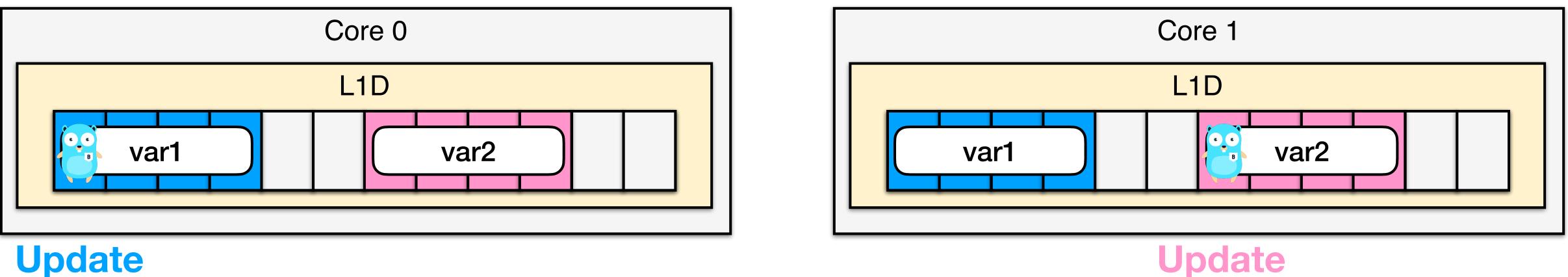




• Why does it matter?

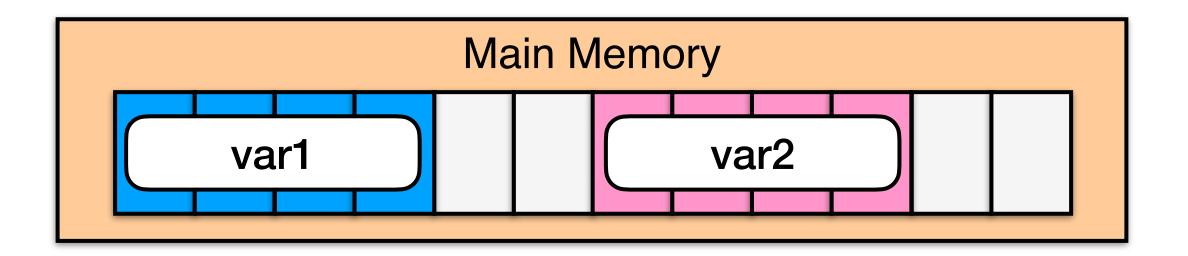


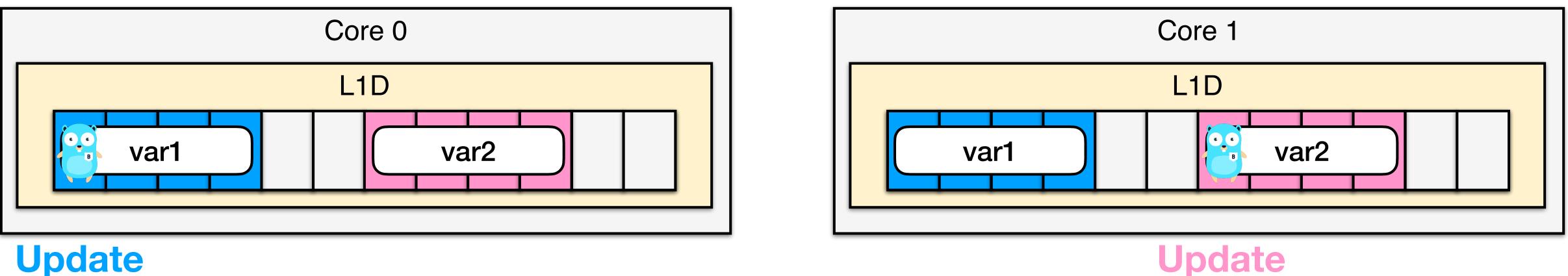




- Why does it matter?
- False sharing a cache line is shared across two cores with at least one goroutine being a writer







- Why does it matter?
- False sharing a cache line is shared across two cores with at least one goroutine being a writer
- Sharing memory is an **illusion**



```
type Struct struct {
    n int
}
var <mark>result</mark> int
func BenchmarkIteration(b *testing.B) {
    structA := Struct{} // Initialization
    structB := Struct{} // Initialization
    wg := sync.WaitGroup{}
    b.ResetTimer()
    for i := 0; i < b.N; i++ {</pre>
        wg.Add( delta: 2)
        go func() { // Spin up first goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structA.n += j
             }
            wg.Done()
        }()
        go func() { // Spin up second goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structB.n += j
             }
            wg.Done()
        }()
        wg.Wait() // Wait
        result = structA.n + structB.n // Aggregate
}
```



```
type Struct struct {
    n int
}
var <mark>result</mark> int
func BenchmarkIteration(b *testing.B) {
    structA := Struct{} // Initialization
    structB := Struct{} // Initialization
    wg := sync.WaitGroup{}
    b.ResetTimer()
    for i := 0; i < b.N; i++ {</pre>
        wg.Add( delta: 2)
        go func() { // Spin up first goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structA.n += j
             }
            wg.Done()
        }()
        go func() { // Spin up second goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structB.n += j
             }
            wg.Done()
        }()
        wg.Wait() // Wait
        result = structA.n + structB.n // Aggregate
}
```

structA.n and structB.n belongs to the same cache line



```
type Struct struct {
    n int
}
var <mark>result</mark> int
func BenchmarkIteration(b *testing.B) {
    structA := Struct{} // Initialization
    structB := Struct{} // Initialization
    wg := sync.WaitGroup{}
    b.ResetTimer()
    for i := 0; i < b.N; i++ {</pre>
        wg.Add( delta: 2)
        go func() { // Spin up first goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structA.n += j
             }
            wg.Done()
        }()
        go func() { // Spin up second goroutine
             for j := 0; j < iteration; j++ {</pre>
                 structB.n += j
             }
            wg.Done()
        }()
        wg.Wait() // Wait
        result = structA.n + structB.n // Aggregate
}
```

structA.n and structB.n belongs to the same cache line





• How to prevent false sharing?



- How to prevent false sharing?
- Solution 1: Do not communicate by sharing memory; instead, share memory by **communicating**







- How to prevent false sharing?
- Solution 1: Do not communicate by sharing memory; instead, share memory by communicating



```
func BenchmarkIterationCommunication(b *testing.B) {
    ch := make(chan int, 2)
    for i := 0; i < b.N; i++ {</pre>
        go func() { // Spin up first goroutine
            i := 0 // Local state
            for j := 0; j < iteration; j++ {</pre>
                i += j
            ch <- i
        }()
        go func() { // Spin up second goroutine
            i := 0 // Local state
            for j := 0; j < iteration; j++ {</pre>
                i += j
            }
            ch <- i
        }()
        result = <-ch + <-ch // Wait and aggregate
```



- How to prevent false sharing?
- Solution 1: Do not communicate by sharing memory; instead, share memory by communicating



```
func BenchmarkIterationCommunication(b *testing.B) {
    ch := make(chan int, 2)
    for i := 0; i < b.N; i++ {</pre>
        go func() { // Spin up first goroutine
            i := 0 // Local state
            for j := 0; j < iteration; j++ {</pre>
                i += j
            ch <- i
        }()
        go func() { // Spin up second goroutine
            i := 0 // Local state
            for j := 0; j < iteration; j++ {</pre>
                i += j
            ch <- i
        }()
        result = <-ch + <-ch // Wait and aggregate
```



- How to prevent false sharing?
- Solution 1: Do not communicate by sharing memory; instead, share memory by communicating



```
func BenchmarkIterationCommunication(b *testing.B) {
    ch := make(chan int, 2)
    for i := 0; i < b.N; i++ {</pre>
        go func() { // Spin up first goroutine
            i := 0 // Local state
            for j := 0; j < iteration; j++ {</pre>
                i += j
            ch <- i
        }()
        go func() { // Spin up second goroutine
            i := 0 // Local state
            for j := 0; j < iteration; j++ {</pre>
                 i += j
            ch <- i
        }()
        result = <-ch + <-ch // Wait and aggregate
```



- How to prevent false sharing?
- Solution 1: Do not communicate by sharing memory; instead, share memory by communicating



```
func BenchmarkIterationCommunication(b *testing.B) {
    ch := make(chan int, 2)
    for i := 0; i < b.N; i++ {</pre>
        go func() { // Spin up first goroutine
            i := 0 // Local state
            for j := 0; j < iteration; j++ {</pre>
                 i += j
            ch <- i
        }()
        go func() { // Spin up second goroutine
            i := 0 // Local state
            for j := 0; j < iteration; j++ {</pre>
                 i += j
            ch <- i
        }()
        result = <-ch + <-ch // Wait and aggregate
```



- How to prevent false sharing?
- Solution 2: padding



- How to prevent false sharing?
- Solution 2: padding

```
type PaddedStruct struct {
    _ cpu.CacheLinePad
    n int
    _ cpu.CacheLinePad
}
func BenchmarkIterationWithPadding(b *testing.B) {
    structA := PaddedStruct{} // Initialization
    structB := PaddedStruct{} // Initialization
    wg := sync.WaitGroup{}
    b.ResetTimer()
    for i := 0; i < b.N; i++ {</pre>
        wg.Add( delta: 2)
        go func() { // Spin up first goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structA.n += j
            }
            wg.Done()
        }()
        go func() { // Spin up second goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structB.n += j
            wg.Done()
        }()
        wg.Wait() // Wait
}
```



- How to prevent false sharing?
- Solution 2: padding

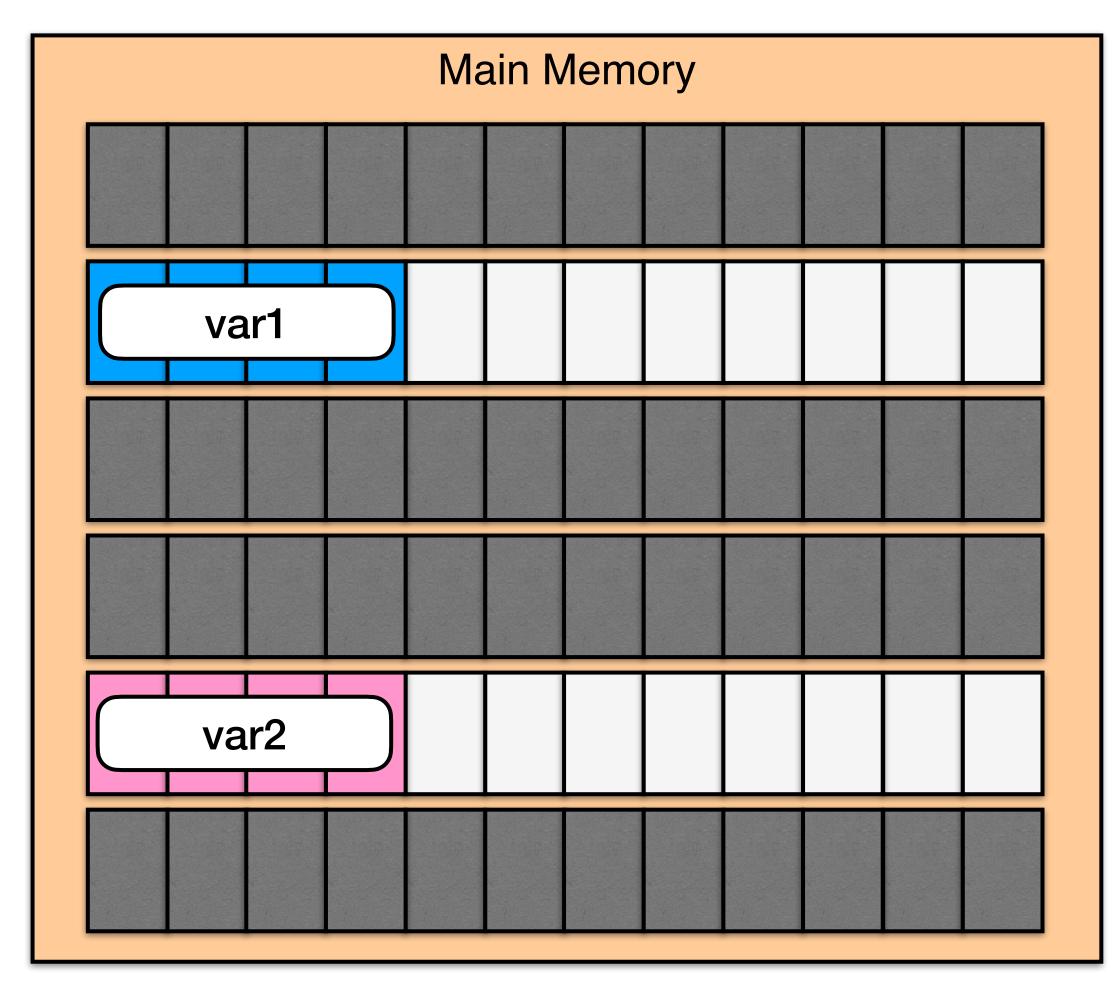
```
type PaddedStruct struct {
   _ cpu.CacheLinePad // 64 bytes
   n int
   _ cpu.CacheLinePad // 64 bytes
}
```

}

```
func BenchmarkIterationWithPadding(b *testing.B) {
    structA := PaddedStruct{} // Initialization
    structB := PaddedStruct{} // Initialization
    wg := sync.WaitGroup{}
    b.ResetTimer()
    for i := 0; i < b.N; i++ {</pre>
        wg.Add( delta: 2)
        go func() { // Spin up first goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structA.n += j
            }
            wg.Done()
        }()
        go func() { // Spin up second goroutine
            for j := 0; j < iteration; j++ {</pre>
                 structB.n += j
            wg.Done()
        }()
        wg.Wait() // Wait
```



- How to prevent false sharing?
- Solution 2: padding



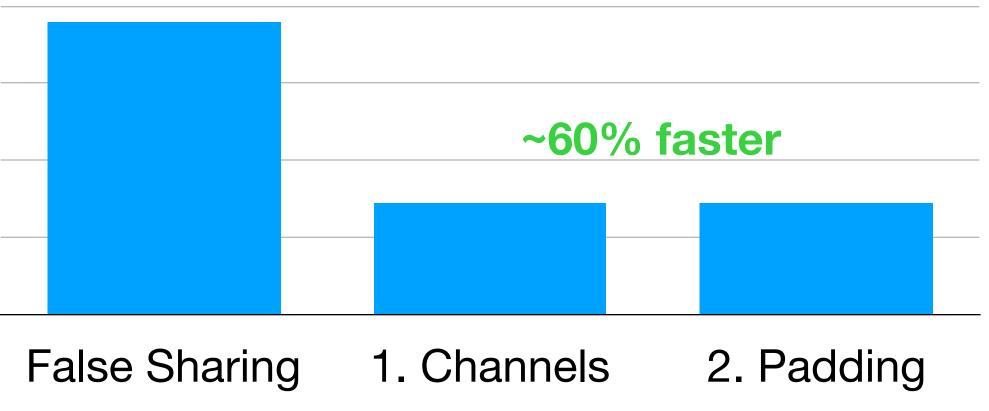
```
type PaddedStruct struct {
    _ cpu.CacheLinePad // 64 bytes
    n int
    _ cpu.CacheLinePad // 64 bytes
}
func BenchmarkIterationWithPadding(b *testing.B) {
    structA := PaddedStruct{} // Initialization
    structB := PaddedStruct{} // Initialization
    wg := sync.WaitGroup{}
    b.ResetTimer()
    for i := 0; i < b.N; i++ {</pre>
        wg.Add( delta: 2)
        go func() { // Spin up first goroutine
            for j := 0; j < iteration; j++ {</pre>
                structA.n += j
            wg.Done()
        }()
        go func() { // Spin up second goroutine
            for j := 0; j < iteration; j++ {</pre>
                structB.n += j
            wg.Done()
        }()
        wg.Wait() // Wait
```



Let's compare the results:

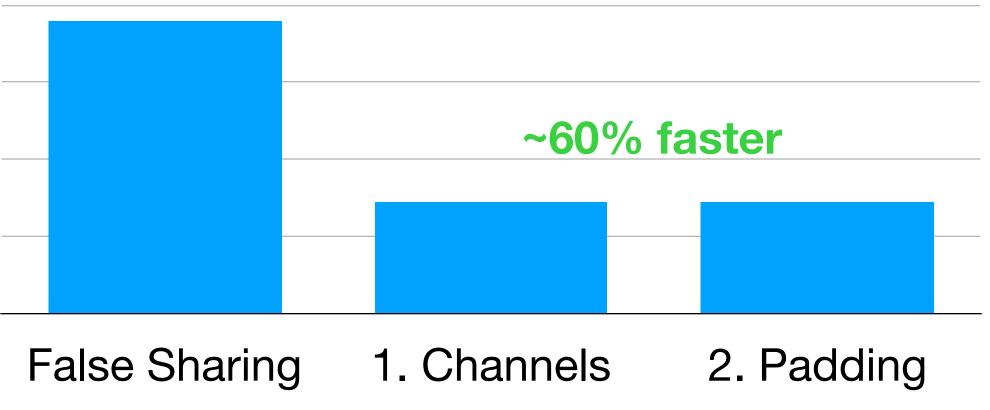


• Let's compare the results:





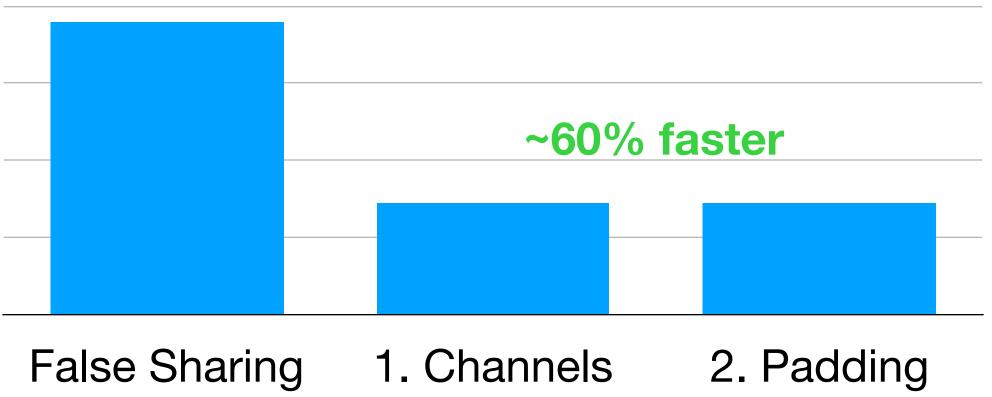
• Let's compare the results:



- Padding is hard Dave Cheney



• Let's compare the results:



- Padding is hard Dave Cheney
- Sometimes, padding is necessary. E.g. we are obliged to share memory and we want to prevent false sharing (library, etc.).



Conclusion





• Sharing memory is an illusion



- Sharing memory is an illusion
- A code that looks perfectly valid might still be **problematic** at CPU level:



- Sharing memory is an **illusion**
- A code that looks perfectly valid might still be **problematic** at CPU level:
 - Caching distribution



- Sharing memory is an **illusion**
- A code that looks perfectly valid might still be **problematic** at CPU level:
 - Caching distribution
 - False sharing



- Sharing memory is an **illusion**
- A code that looks perfectly valid might still be **problematic** at CPU level:
 - Caching distribution
 - False sharing
- We can help the CPU with locality of reference and predictability (algorithms & data structures)



- Sharing memory is an illusion
- A code that looks perfectly valid might still be **problematic** at CPU level:
 - Caching distribution lacksquare
 - False sharing
- We can help the CPU with locality of reference and predictability (algorithms & data structures)

What else?

- Watch out for premature:
 - Optimisations



- Sharing memory is an illusion
- A code that looks perfectly valid might still be **problematic** at CPU level:
 - Caching distribution
 - False sharing
- We can help the CPU with locality of reference and predictability (algorithms & data structures)

What else?

- Watch out for premature:
 - Optimisations
 - Concurrency



- Sharing memory is an illusion
- A code that looks perfectly valid might still be **problematic** at CPU level:
 - Caching distribution
 - False sharing
- We can help the CPU with locality of reference and predictability (algorithms & data structures)

What else?

- Watch out for premature:
 - Optimisations
 - Concurrency
- Mechanical sympathy goes beyond the very scope of CPUs







