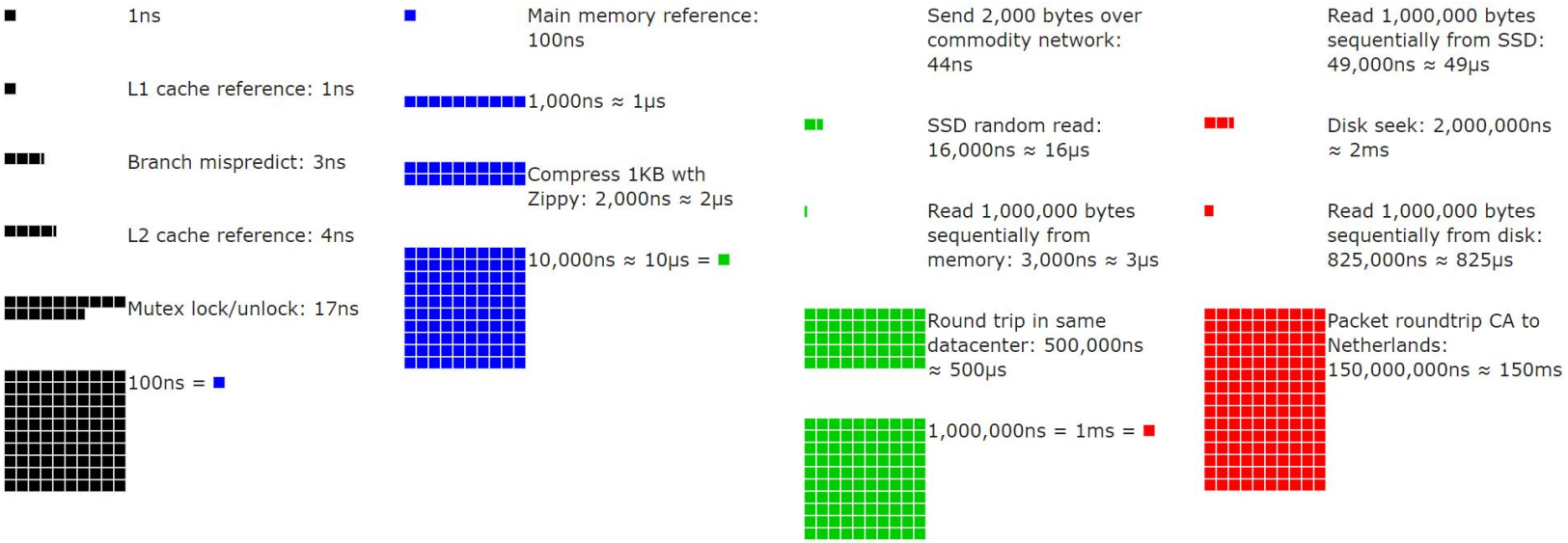
Recitation 10

Computer Architecture (section 1)

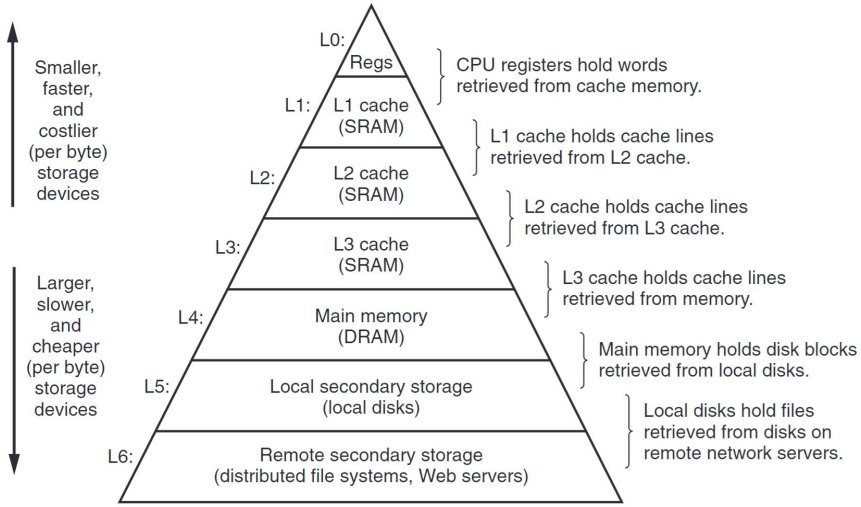
Numbers every programmer should know



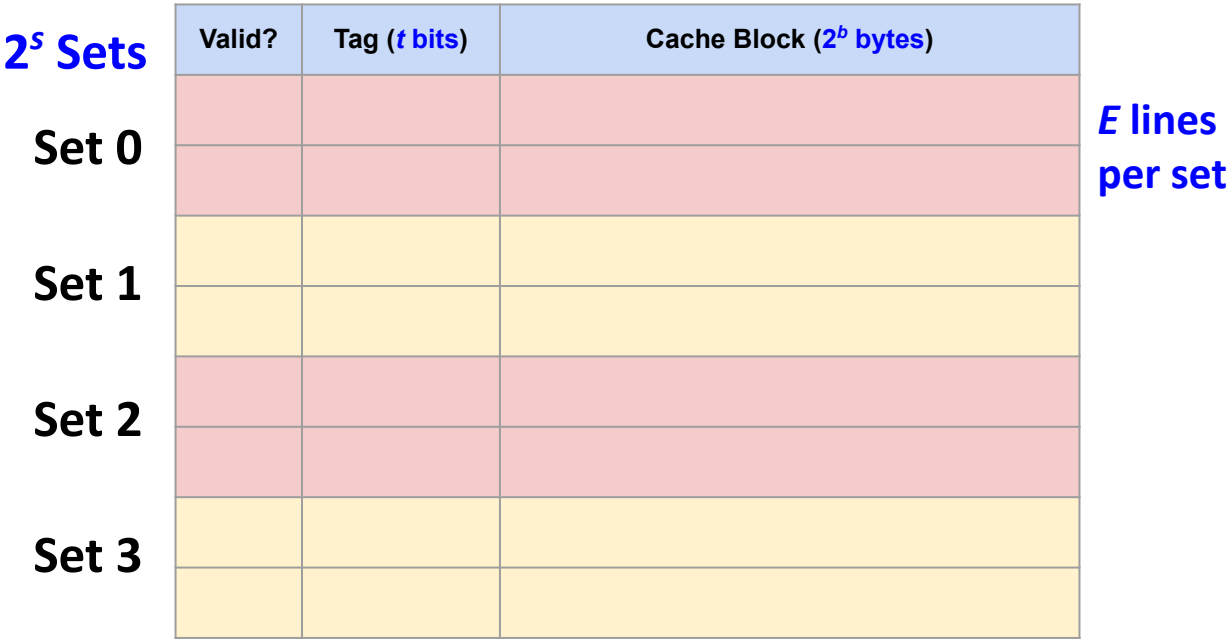
https://colin-scott.github.io/personal_website/research/interactive_latency.html

Caches

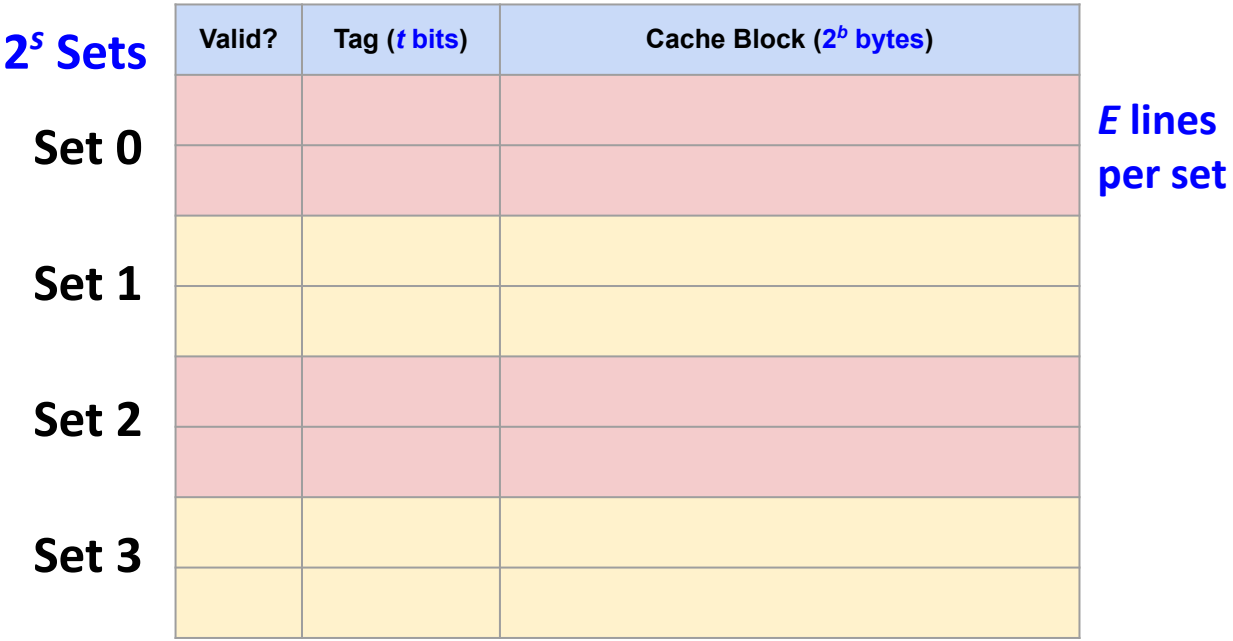
- How can we hide memory access latency?
  - Exploit locality
    - Temporal
    - Spatial
- A cache is a fast memory store used for frequently accessed data.
  - Basically operates as a hash table.



Cache Organization



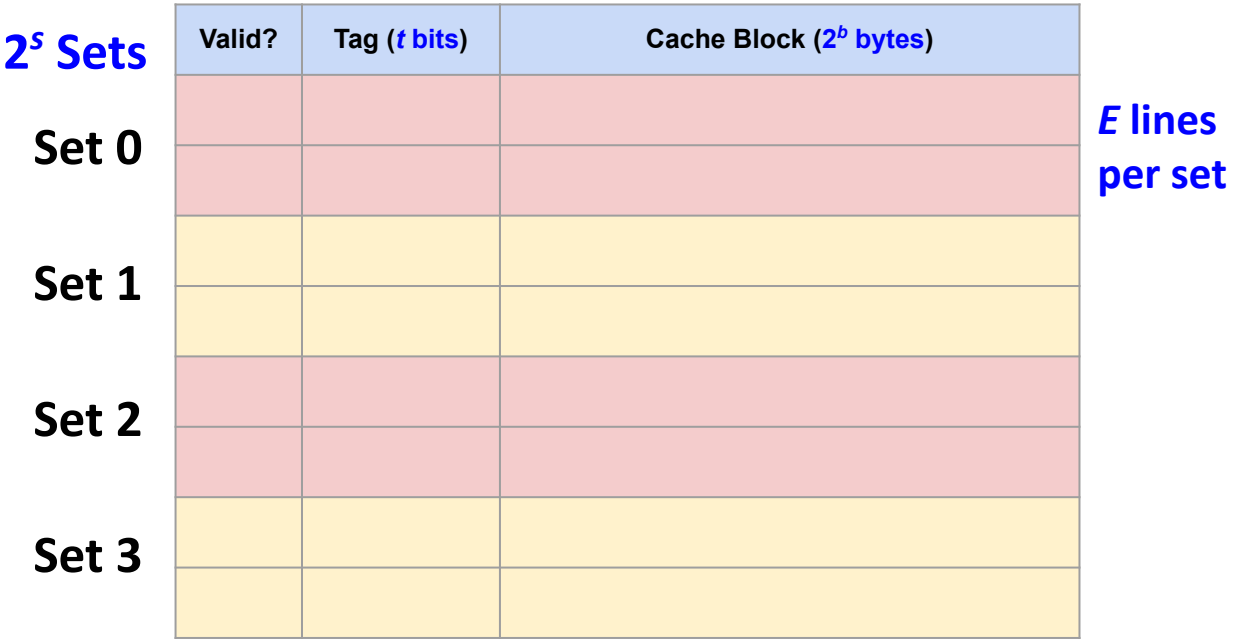
Cache Organization



How do memory addresses map into the cache?

Address

Cache Organization



How do memory addresses map into the cache?

Address

Tag	Set Index	Block Offset
t bits	s bits	b bits

Cache size = $2^b \times E \times 2^s$

Practice Problem

You are given a system with the following configuration:

- 32-bit address space
- Cache size of 2048 bytes
- Block size of 64 bytes
- 4 lines per set

Practice Problem

You are given a system with the following configuration:

- 32-bit address space
- Cache size of 2048 bytes
- Block size of 64 bytes
- 4 lines per set

How many sets are in this cache?

Practice Problem

You are given a system with the following configuration:

- 32-bit address space
- Cache size of 2048 bytes
- Block size of 64 bytes
- 4 lines per set

How many sets are in this cache?

$$\text{Cache Size} = 2^b \times E \times 2^s$$

Practice Problem

You are given a system with the following configuration:

- 32-bit address space
- Cache size of 2048 bytes
- Block size of 64 bytes
- 4 lines per set

How many sets are in this cache?

$$\text{Cache Size} = 2^b \times E \times 2^s$$

$$2048 = 64 \times 4 \times 2^s$$

Practice Problem

You are given a system with the following configuration:

- 32-bit address space
- Cache size of 2048 bytes
- Block size of 64 bytes
- 4 lines per set

How many sets are in this cache?

$$\text{Cache Size} = 2^b \times E \times 2^s$$

$$2048 = 64 \times 4 \times 2^s$$

$$2^s = \textbf{8 sets}$$

Practice Problem

You are given a system with the following configuration:

- 32-bit address space
- Cache size of 2048 bytes
- Block size of 64 bytes
- 4 lines per set

How does a 32-bit memory address map into this cache?

Practice Problem

You are given a system with the following configuration:

- 32-bit address space
- Cache size of 2048 bytes
- Block size of 64 bytes
- 4 lines per set

How does a 32-bit memory address map into this cache?

$2^b=64$, $2^s=8$

Practice Problem

You are given a system with the following configuration:

- 32-bit address space
- Cache size of 2048 bytes
- Block size of 64 bytes
- 4 lines per set

How does a 32-bit memory address map into this cache?

$2^b=64$, $2^s=8$

$b=6$, $s=3$, t = 32-(6+3)

Practice Problem

You are given a system with the following configuration:

- 32-bit address space
- Cache size of 2048 bytes
- Block size of 64 bytes
- 4 lines per set

How does a 32-bit memory address map into this cache?

$2^b=64$, $2^s=8$

$b=6$, $s=3$, t = 32-(6+3)

Cache Mapping Techniques

By varying the number of lines per cache set (*E*), the associativity of the cache changes.

Cache Mapping Techniques

By varying the number of lines per cache set (*E*), the associativity of the cache changes.

- 1 line per set
  - A direct mapped cache

Cache Mapping Techniques

By varying the number of lines per cache set (*E*), the associativity of the cache changes.

- 1 line per set
  - A direct mapped cache
- 2,3,4… (n) lines per set
  - n-way set-associative cache

Cache Mapping Techniques

By varying the number of lines per cache set (*E*), the associativity of the cache changes.

- 1 line per set
  - A direct mapped cache
- 2,3,4… (n) lines per set
  - n-way set-associative cache
- All lines in a single set
  - Fully-associative cache

Direct Mapped Cache

	Valid?	Tag	Cache Block (8 bytes)
Set 0			
Set 1			
Set 2			
Set 3			
Set 4			
Set 5			
Set 6			
Set 7			

***E* = 1**
line
per set

Direct Mapped Cache

	Valid?	Tag	Cache Block (8 bytes)
Set 0			
Set 1			
Set 2			
Set 3			
Set 4			
Set 5			
Set 6			
Set 7			

E = 1 line per set

How do memory addresses map into the cache?

Direct Mapped Cache

	Valid?	Tag	Cache Block (8 bytes)
Set 0			
Set 1			
Set 2			
Set 3			
Set 4			
Set 5			
Set 6			
Set 7			

E = 1 line per set

How do memory addresses map into the cache?

8-bit Address

Tag	Set Index	Block Offset
2 bits	*3* bits	*3* bits

Direct Mapped Cache in Action

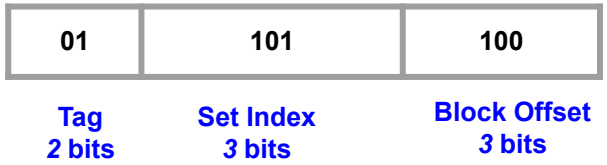
	Valid?	Tag	Cache Block (8 bytes)
Set 0	0		
Set 1	0		
Set 2	0		
Set 3	0		
Set 4	0		
Set 5	0		
Set 6	0		
Set 7	0		

Read Request

Direct Mapped Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	0		
Set 1	0		
Set 2	0		
Set 3	0		
Set 4	0		
Set 5	0		
Set 6	0		
Set 7	0		

Read Request

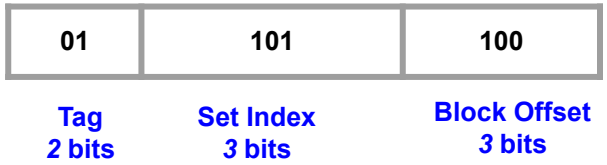


Cache miss

Direct Mapped Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	0		
Set 1	0		
Set 2	0		
Set 3	0		
Set 4	0		
Set 5	1	01	▫▫▫▫▩▫▫▫
Set 6	0		
Set 7	0		

Read Request



Direct Mapped Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	0		
Set 1	0		
Set 2	0		
Set 3	0		
Set 4	0		
Set 5	1	01	□□□□□□□□
Set 6	0		
Set 7	0		

Read Request

01	001	110
Tag *2* bits	Set Index *3* bits	Block Offset *3* bits

Direct Mapped Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	0		
Set 1	0		
Set 2	0		
Set 3	0		
Set 4	0		
Set 5	1	01	□□□□□□□□
Set 6	0		
Set 7	0		

Read Request

Tag
2 bits

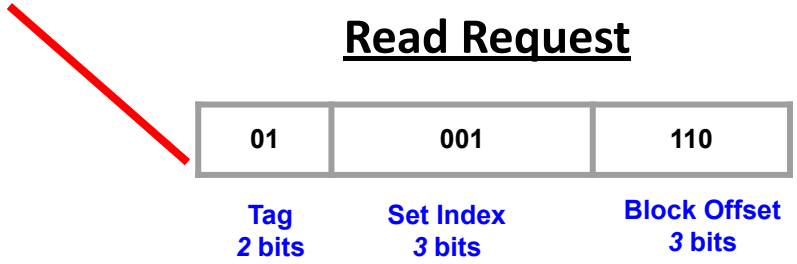
Set Index
3 bits

Block Offset
3 bits

Cache miss

Direct Mapped Cache in Action

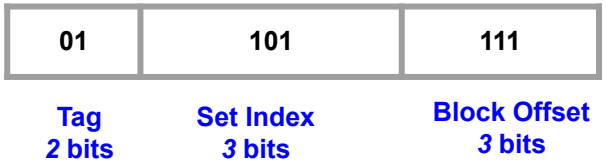
	Valid?	Tag	Cache Block (8 bytes)
Set 0	0		
Set 1	1	01	▫▩▫▫▫▫▫▫
Set 2	0		
Set 3	0		
Set 4	0		
Set 5	1	01	▫▫▫▫▫▫▫▫
Set 6	0		
Set 7	0		



Direct Mapped Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	0		
Set 1	1	01	□□□□□□□□
Set 2	0		
Set 3	0		
Set 4	0		
Set 5	1	01	□□□□□□□□
Set 6	0		
Set 7	0		

Read Request



Direct Mapped Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	0		
Set 1	1	01	□□□□□□□□
Set 2	0		
Set 3	0		
Set 4	0		
Set 5	1	01	▩□□□□□□□
Set 6	0		
Set 7	0		

Read Request

Cache hit

Direct Mapped Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	0		
Set 1	1	01	▫▫▫▫▫▫▫▫
Set 2	0		
Set 3	0		
Set 4	0		
Set 5	1	01	▫▫▫▫▫▫▫▫
Set 6	0		
Set 7	0		

Read Request

11	001	101
Tag *2 bits*	Set Index *3 bits*	Block Offset *3 bits*

Direct Mapped Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	0		
Set 1	1	01	▫▫▫▫▫▫▫▫
Set 2	0		
Set 3	0		
Set 4	0		
Set 5	1	01	▫▫▫▫▫▫▫▫
Set 6	0		
Set 7	0		

Read Request

Tag
2 bits

Set Index
3 bits

Block Offset
3 bits

Cache miss

Tags do not match!

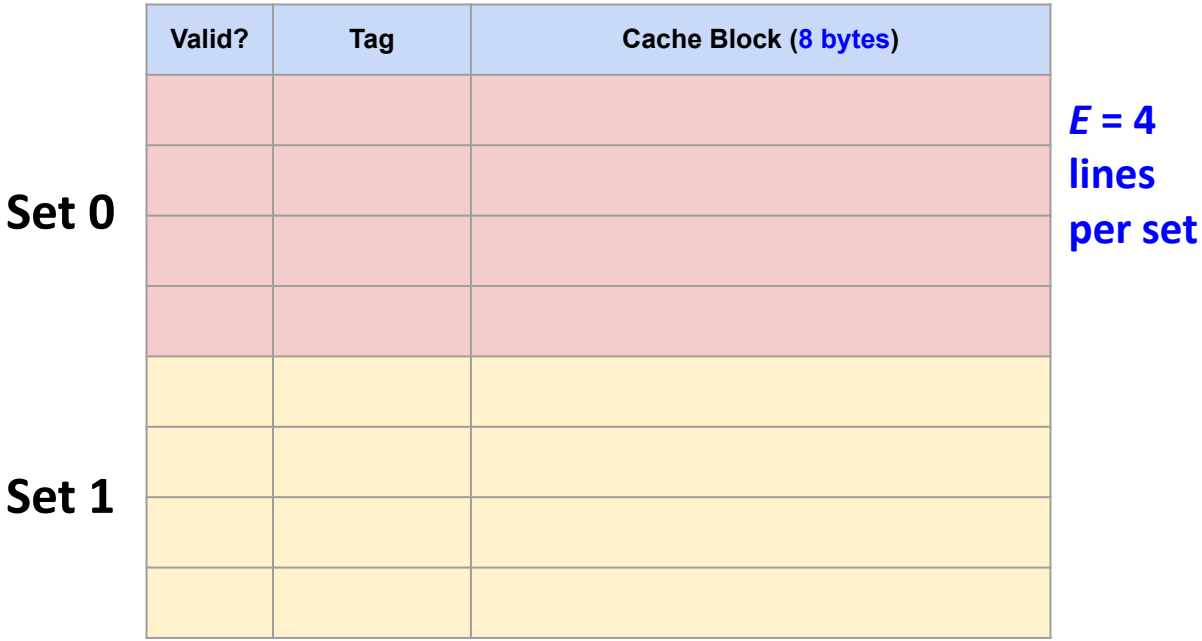
Direct Mapped Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	0		
Set 1	1	11	□□□▩□□□□
Set 2	0		
Set 3	0		
Set 4	0		
Set 5	1	01	□□□□□□□□
Set 6	0		
Set 7	0		

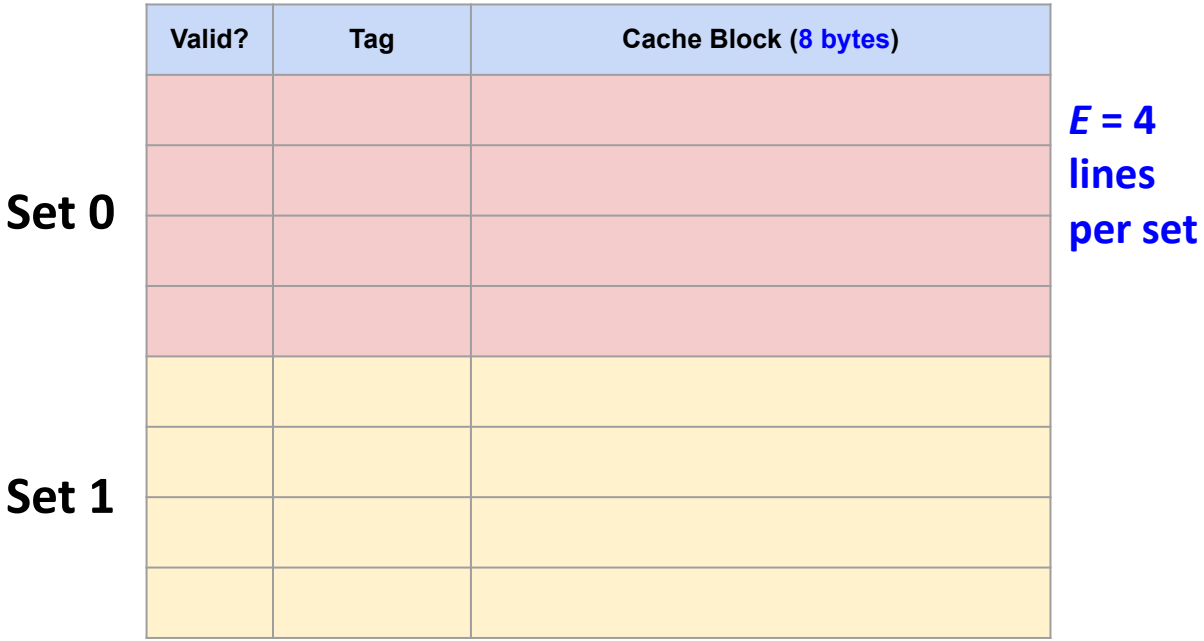
Read Request

11	001	101
Tag *2 bits*	Set Index *3 bits*	Block Offset *3 bits*

Set-associative Cache (4-way)

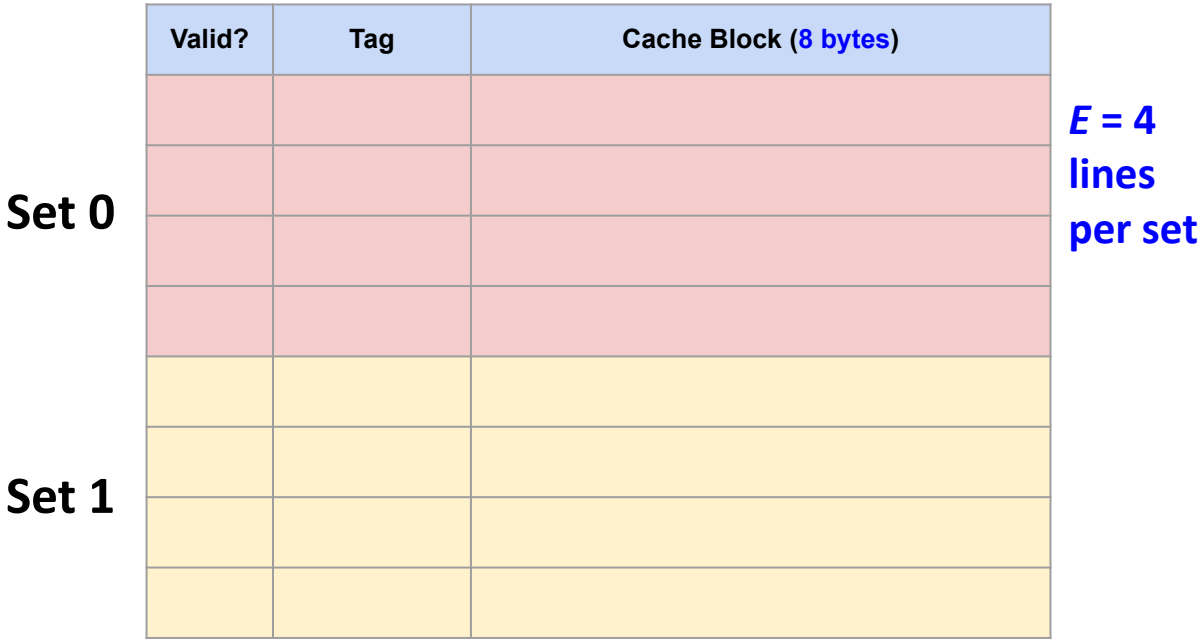


Set-associative Cache (4-way)



How do memory addresses map into the cache?

Set-associative Cache (4-way)



E = 4 lines per set

How do memory addresses map into the cache?

8-bit Address

4-way Set Associative Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	0		
	0		
	0		
	0		
Set 1	0		
	0		
	0		
	0		

Read Request

1101	0	111
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

4-way Set Associative Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	0		
	0		
	0		
	0		
Set 1	0		
	0		
	0		
	0		

Read Request

1101	0	111
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

Cache miss

4-way Set Associative Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	▩□□□□□□□
	0		
	0		
	0		
Set 1	0		
	0		
	0		
	0		

Read Request

1101	0	111
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

4-way Set Associative Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	▫▫▫▫▫▫▫▫
	0		
	0		
	0		
Set 1	0		
	0		
	0		
	0		

Read Request

0001	0	000
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

4-way Set Associative Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	▫▫▫▫▫▫▫▫
	0		
	0		
	0		
Set 1	0		
	0		
	0		
	0		

Read Request

0001	0	000
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

Cache miss

4-way Set Associative Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	□□□□□□□□
	1	0001	□□□□□□□▩
	0		
	0		
Set 1	0		
	0		
	0		
	0		

Read Request

0001	0	000
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

4-way Set Associative Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	□□□□□□□□
	1	0001	□□□□□□□□
	0		
	0		
Set 1	0		
	0		
	0		
	0		

Read Request

1001	0	111
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

4-way Set Associative Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	▫▫▫▫▫▫▫▫
	1	0001	▫▫▫▫▫▫▫▫
	0		
	0		
Set 1	0		
	0		
	0		
	0		

Read Request

1001	0	111
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

Cache miss

4-way Set Associative Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	▫▫▫▫▫▫▫▫
	1	0001	▫▫▫▫▫▫▫▫
	1	1001	▩▫▫▫▫▫▫▫
	0		
Set 1	0		
	0		
	0		
	0		

Read Request

1001	0	111
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

4-way Set Associative Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	▫▫▫▫▫▫▫▫
	1	0001	▫▫▫▫▫▫▫▫
	1	1001	▫▫▫▫▫▫▫▫
	0		
Set 1	0		
	0		
	0		
	0		

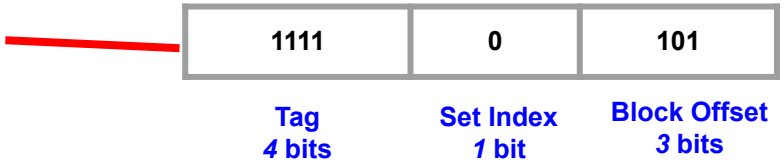
Read Request

1111	0	101
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

4-way Set Associative Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	▫▫▫▫▫▫▫▫
	1	0001	▫▫▫▫▫▫▫▫
	1	1001	▫▫▫▫▫▫▫▫
	0		
Set 1	0		
	0		
	0		
	0		

Read Request



Cache miss

4-way Set Associative Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	□□□□□□□□
	1	0001	□□□□□□□□
	1	1001	□□□□□□□□
	1	1111	□□□▩□□□□
Set 1	0		
	0		
	0		
	0		

Read Request

1111	0	101
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

4-way Set Associative Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	▫▫▫▫▫▫▫▫
	1	0001	▫▫▫▫▫▫▫▫
	1	1001	▫▫▫▫▫▫▫▫
	1	1111	▫▫▫▫▫▫▫▫
Set 1	0		
	0		
	0		
	0		

Read Request

1011	0	111
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

4-way Set Associative Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	▫▫▫▫▫▫▫▫
	1	0001	▫▫▫▫▫▫▫▫
	1	1001	▫▫▫▫▫▫▫▫
	1	1111	▫▫▫▫▫▫▫▫
Set 1	0		
	0		
	0		
	0		

Read Request

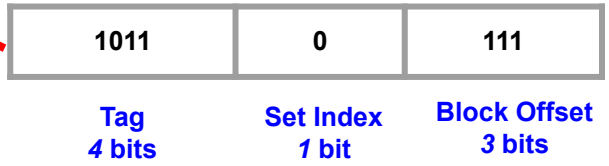
1011	0	111
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

Cache miss

4-way Set Associative Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	▫▫▫▫▫▫▫▫
	1	0001	▫▫▫▫▫▫▫▫
	1	1001	▫▫▫▫▫▫▫▫
	1	1111	▫▫▫▫▫▫▫▫
Set 1	0		
	0		
	0		
	0		

Read Request



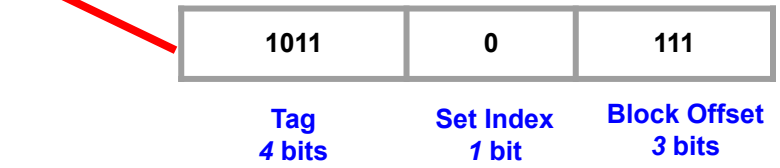
Cache miss

We need to evict a line from the cache.

4-way Set Associative Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	▫▫▫▫▫▫▫▫
	1	0001	▫▫▫▫▫▫▫▫
	1	1001	▫▫▫▫▫▫▫▫
	1	1111	▫▫▫▫▫▫▫▫
Set 1	0		
	0		
	0		
	0		

Read Request



To evict, we use a cache eviction policy. Let's evict at random for now.

4-way Set Associative Cache in Action

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	□□□□□□□□
	1	0001	□□□□□□□□
	1	1011	▩□□□□□□□
	1	1111	□□□□□□□□
Set 1	0		
	0		
	0		
	0		

Read Request

1011	0	111
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

Fully Associative Cache

Set 0

Valid?	Tag	Cache Block (8 bytes)

E = 8 lines per set

Fully Associative Cache

Set 0

Valid?	Tag	Cache Block (8 bytes)

E = 8 lines per set

How do memory addresses map into the cache?

Fully Associative Cache

Set 0

Valid?	Tag	Cache Block (8 bytes)

***E* = 8 lines per set**

How do memory addresses map into the cache?

8-bit Address

Fully Associative Cache

Set 0

Valid?	Tag	Cache Block (8 bytes)

E = 8 lines per set

How do memory addresses map into the cache?

8-bit Address

With 1 set we don't need a set index

Writing to the Cache

So far we have seen cache behaviour for memory reads.

How should the cache behave when we write to memory?

Writing to the Cache

So far we have seen cache behaviour for memory reads.

How should the cache behave when we write to memory?

Should we write directly to memory and ignore the cache?

Should we write to both the cache and memory?

Should we only write to the cache?

Write-through Cache

If memory block is in cache:

1. Write changes to cache.
2. Then write changes to memory.

Write-through Cache

If memory block is in cache:

1. Write changes to cache.
2. Then write changes to memory.

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	□□□□□□□□
Set 1	1	1001	□□□□□□□□

Write-through Cache

If memory block is in cache:

1. Write changes to cache.
2. Then write changes to memory.

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	□□□□□□□□
Set 1	1	1001	□□□□□□□□

Write Request

1001	1	000
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

Write-through Cache

If memory block is in cache:

1. Write changes to cache.
2. Then write changes to memory.

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	▫▫▫▫▫▫▫▫
Set 1	1	1001	▫▫▫▫▫▫▫▪

Write Request

1001	1	000
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

Write-through Cache

If memory block is in cache:

1. Write changes to cache.
2. Then write changes to memory.

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	▫▫▫▫▫▫▫▫
Set 1	1	1001	▫▫▫▫▫▫▫▪

Write Request

1001	1	000
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

Now we write this line to memory.

Write-back Cache

If memory block is in cache:

1. Write changes to cache.
2. On eviction of the cache line, write changes to memory.

Write-back Cache

If memory block is in cache:

1. Write changes to cache.
2. On eviction of the cache line, write changes to memory.

The caveat is we must now store a *dirty bit* for each line.

Write-back Cache

If memory block is in cache:

1. Write changes to cache.
2. On eviction of the cache line, write changes to memory.

The caveat is we must now store a *dirty bit* for each line.

	Valid?	Tag	Dirty	Cache Block (8 bytes)
Set 0	1	1101	0	□□□□□□□□
Set 1	1	1001	0	□□□□□□□□

Write-back Cache

If memory block is in cache:

1. Write changes to cache.
2. On eviction of the cache line, write changes to memory.

The caveat is we must now store a *dirty bit* for each line.

	Valid?	Tag	Dirty	Cache Block (8 bytes)
Set 0	1	1101	0	▫▫▫▫▫▫▫▫
Set 1	1	1001	0	▫▫▫▫▫▫▫▫

Write Request

1001	1	000
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

Write-back Cache

If memory block is in cache:

1. Write changes to cache.
2. On eviction of the cache line, write changes to memory.

The caveat is we must now store a *dirty bit* for each line.

	Valid?	Tag	Dirty	Cache Block (8 bytes)
Set 0	1	1101	0	□□□□□□□□
Set 1	1	1001	1	□□□□□□□▩

Write Request

1001	1	000
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

Write-back Cache

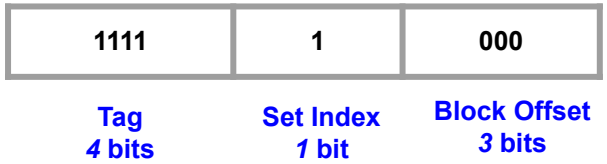
If memory block is in cache:

1. Write changes to cache.
2. On eviction of the cache line, write changes to memory.

The caveat is we must now store a *dirty bit* for each line.

	Valid?	Tag	Dirty	Cache Block (8 bytes)
Set 0	1	1101	0	□□□□□□□□
Set 1	1	1001	1	□□□□□□□□

Read Request



Write-back Cache

If memory block is in cache:

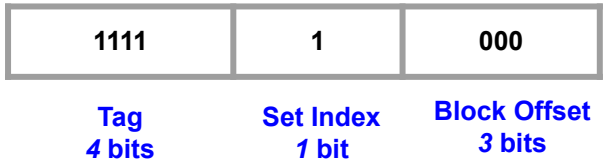
1. Write changes to cache.
2. On eviction of the cache line, write changes to memory.

The caveat is we must now store a *dirty bit* for each line.

	Valid?	Tag	Dirty	Cache Block (8 bytes)
Set 0	1	1101	0	□□□□□□□□
Set 1	1	1001	1	□□□□□□□□

Write to memory on eviction

Read Request



Cache miss

Write-allocate Cache

If memory block is not in cache:

1. Read block from memory into cache.
2. Use a write-back policy.

Write-allocate Cache

If memory block is not in cache:

1. Read block from memory into cache.
2. Use a write-back policy.

	Valid?	Tag	Dirty	Cache Block (8 bytes)
Set 0	0	1101	0	□□□□□□□□
Set 1	0	1111	0	□□□□□□□□

Write-allocate Cache

If memory block is not in cache:

1. Read block from memory into cache.
2. Use a write-back policy.

	Valid?	Tag	Dirty	Cache Block (8 bytes)
Set 0	0	1101	0	▫▫▫▫▫▫▫▫
Set 1	0	1111	0	▫▫▫▫▫▫▫▫

Write Request

1001	1	000
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

Write-allocate Cache

If memory block is not in cache:

1. Read block from memory into cache.
2. Use a write-back policy.

	Valid?	Tag	Dirty	Cache Block (8 bytes)
Set 0	0	1101	0	▫▫▫▫▫▫▫▫
Set 1	1	1001	0	▫▫▫▫▫▫▫▫

Write Request

1001	1	000
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

Write-allocate Cache

If memory block is not in cache:

1. Read block from memory into cache.
2. Use a write-back policy.

	Valid?	Tag	Dirty	Cache Block (8 bytes)
Set 0	0	1101	0	▫▫▫▫▫▫▫▫
Set 1	1	1001	1	▫▫▫▫▫▫▫▪

Write Request

1001	1	000
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

Write-allocate Cache

If memory block is not in cache:

1. Read block from memory into cache.
2. Use a write-back policy.

	Valid?	Tag	Dirty	Cache Block (8 bytes)
Set 0	0	1101	0	▫▫▫▫▫▫▫▫
Set 1	1	1001	1	▫▫▫▫▫▫▫▫

Read Request

1111	1	000
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

Write-allocate Cache

If memory block is not in cache:

1. Read block from memory into cache.
2. Use a write-back policy.

	Valid?	Tag	Dirty	Cache Block (8 bytes)
Set 0	0	1101	0	▫▫▫▫▫▫▫▫
Set 1	1	1001	1	▫▫▫▫▫▫▫▫

Write to memory on eviction

Read Request

1111	1	000
Tag *4* bits	Set Index *1* bit	Block Offset *3* bits

Cache miss

No-write-allocate Cache

If memory block is not in cache:

1. Directly write to memory.

With this policy, the cache is only populated on a read miss

No-write-allocate Cache

If memory block is not in cache:

1. Directly write to memory.

With this policy, the cache is only populated on a read miss

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	□□□□□□□□
Set 1	1	1001	□□□□□□□□

No-write-allocate Cache

If memory block is not in cache:

1. Directly write to memory.

With this policy, the cache is only populated on a read miss

	Valid?	Tag	Cache Block (8 bytes)
Set 0	1	1101	□□□□□□□□
Set 1	1	1001	□□□□□□□□

Write Request