Introduction to Cache

Caiwen Ding

Department of Computer Science and Engineering

University of Connecticut

Adapted from *Computer Organization and Design*
by Patterson & Hennessy

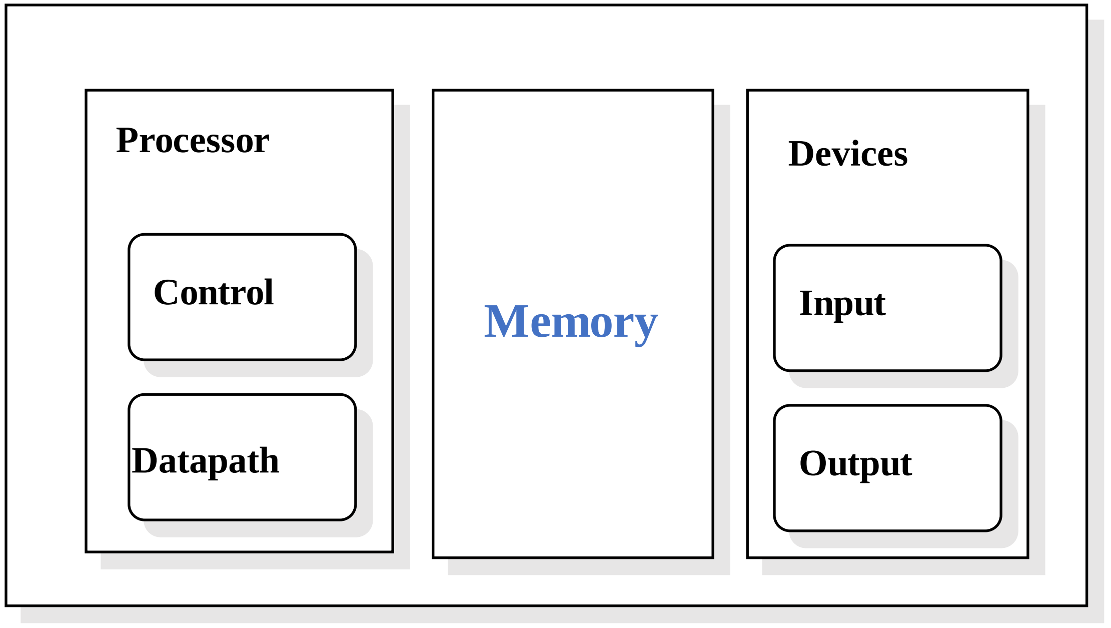
Cache

- Memory hierarchy
- Cache
  - What? Why? How?
- Direct mapped cache
  - Cache access (read)
  - How bits in addresses are used

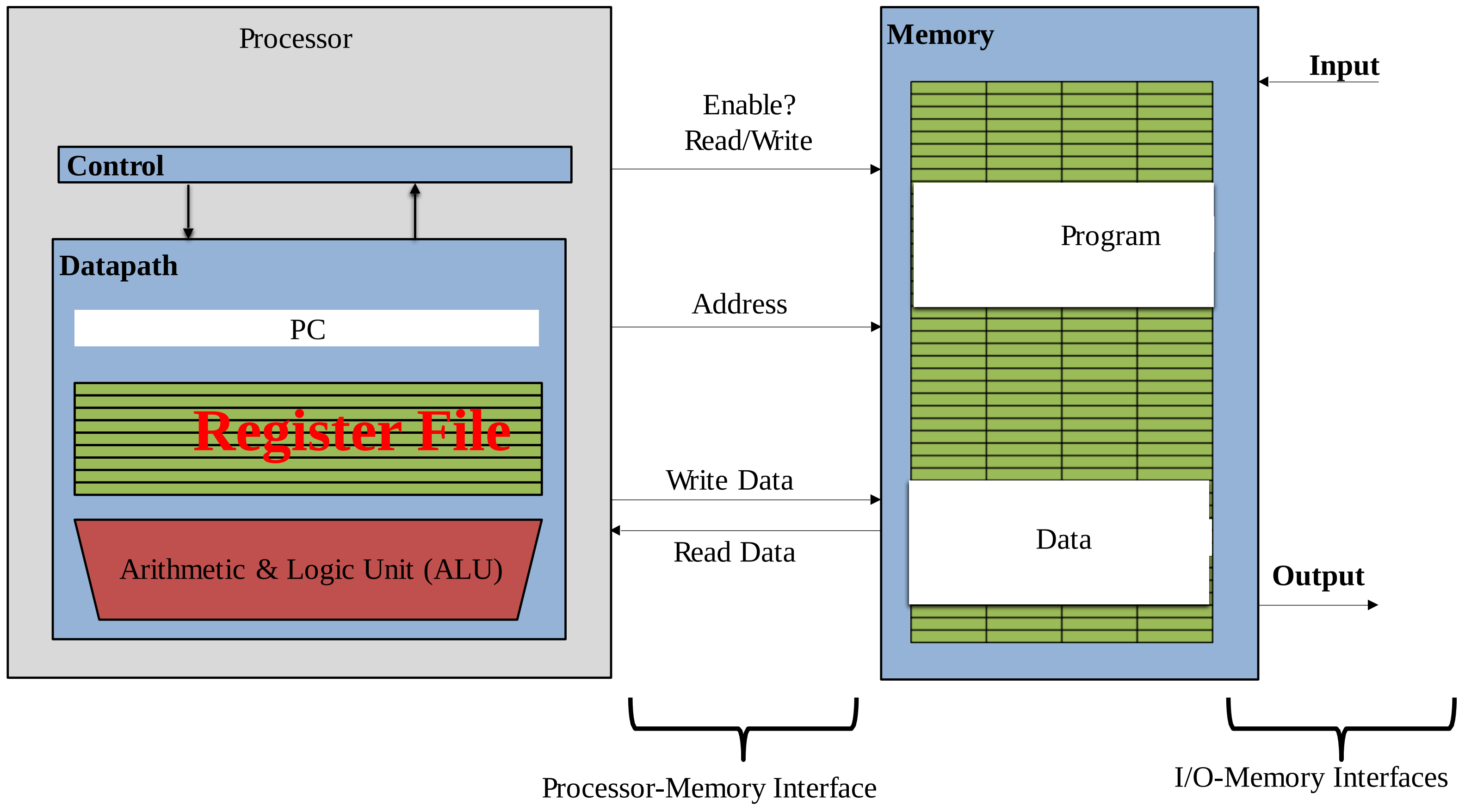
Reading: Sections 5.1 and 5.3

Section 5.2 is very helpful.

Review: Major Components of a Computer



Review: Computer



5-stage?

Memory Hierarchy

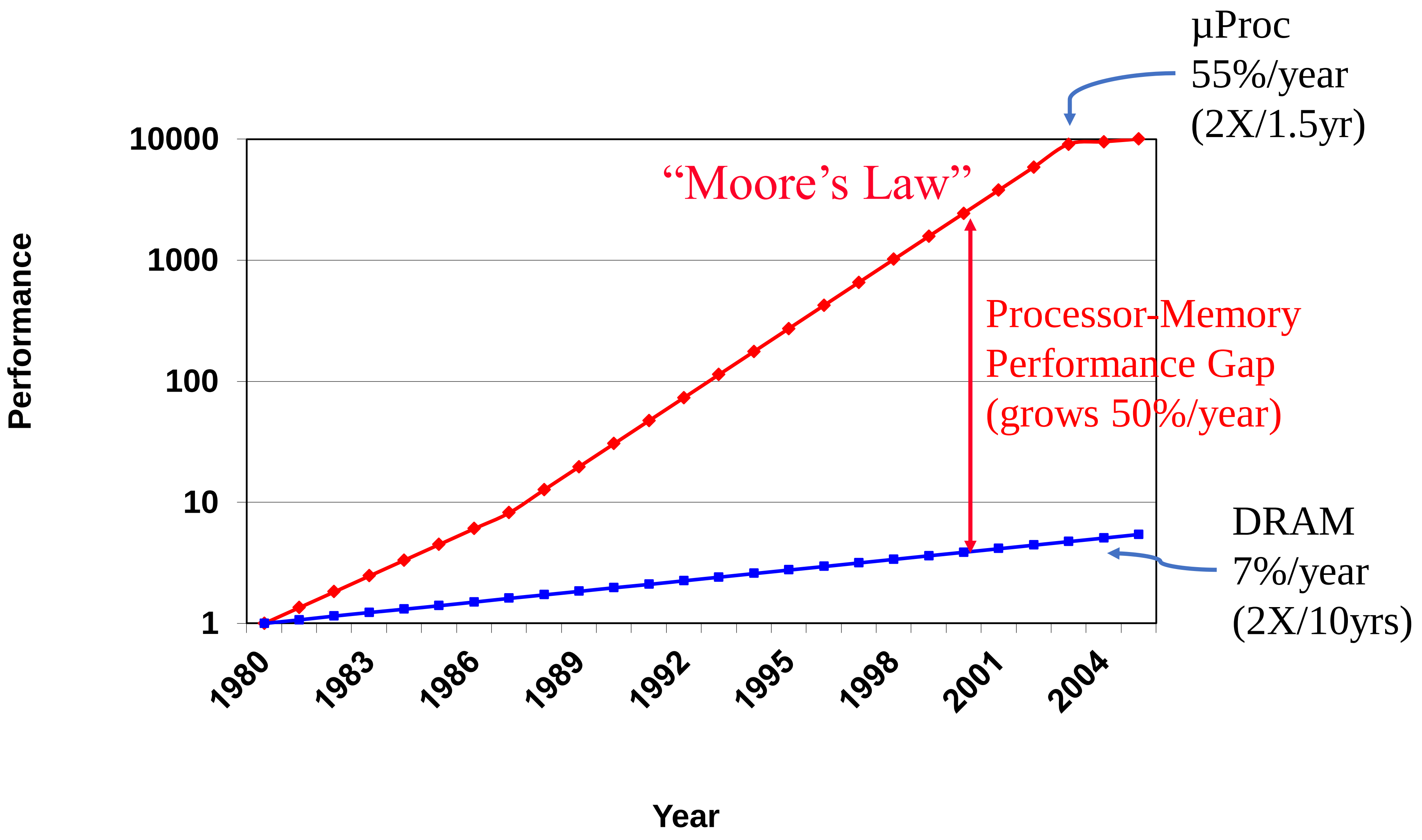
- Processor
- Memory (“main memory”)
- Disk

Memory Technologies

- SRAM is fast, but more expensive
  - Fast (typical access times of 0.5 to 2.5 ns)
  - Low density (6 transistor cells), higher power, expensive
    - $2000 to $5000 per GB in 2008
  - Static: content will last “forever” (as long as power is left on)

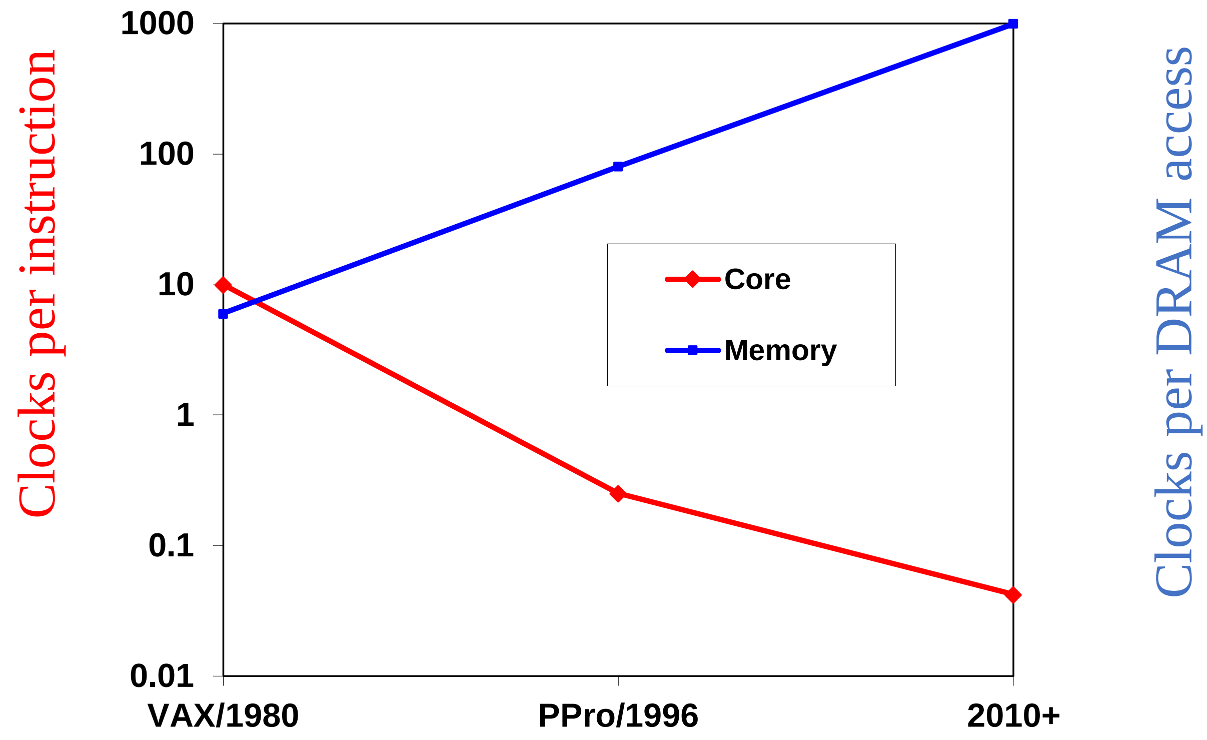
- DRAM is larger, and cheaper (because of higher density)
  - For main memory
  - Slower (typical access times of 50 to 70 ns)
  - High density (1 transistor cells), lower power, cheaper
    - $20 to $75 per GB in 2008
  - Dynamic: needs to be “refreshed” regularly (~ every 8 ms)
    - consumes1% to 2% of the active cycles of the DRAM

Processor-Memory Performance Gap



The “Memory Wall”

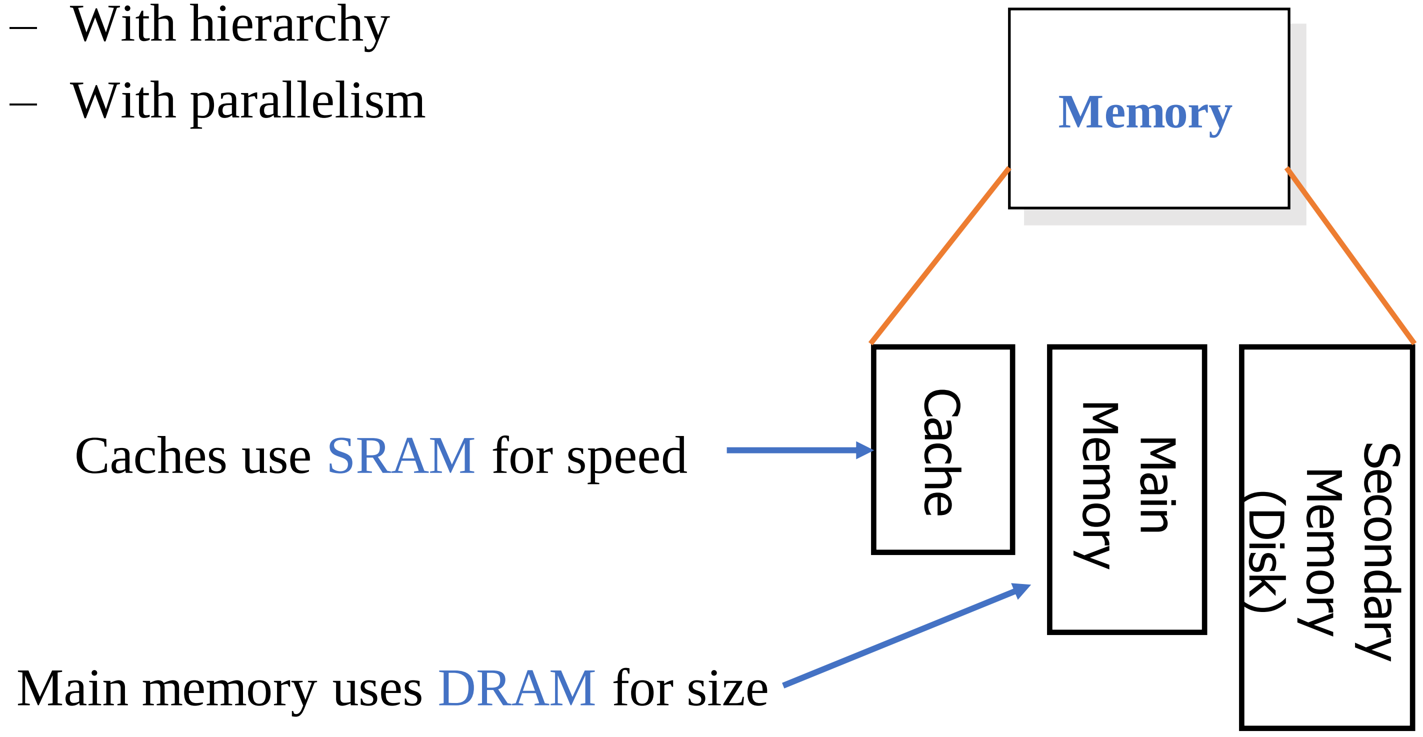
- Processor vs DRAM speed disparity continues to grow



Good memory hierarchy (cache) design is increasingly important to overall performance

The Memory Hierarchy Goal

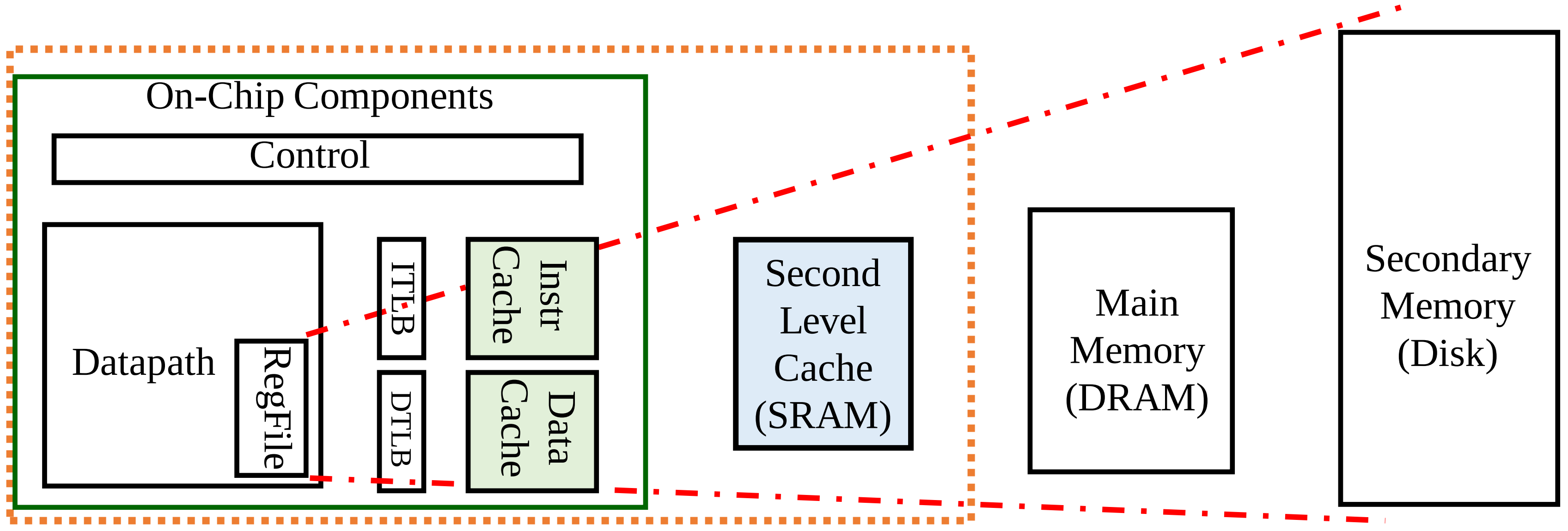
- Fact: Large memories are slow and fast memories are small
- How do we create a memory that gives the illusion of being large, cheap and fast (most of the time)?
  - With hierarchy
  - With parallelism



Cache is much smaller than the main memory, but much faster!

A Typical Memory Hierarchy

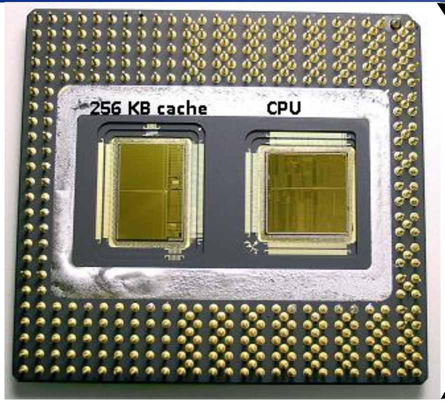
- Present the user with as much memory as is available in the *cheapest* technology at the speed offered by the *fastest* technology

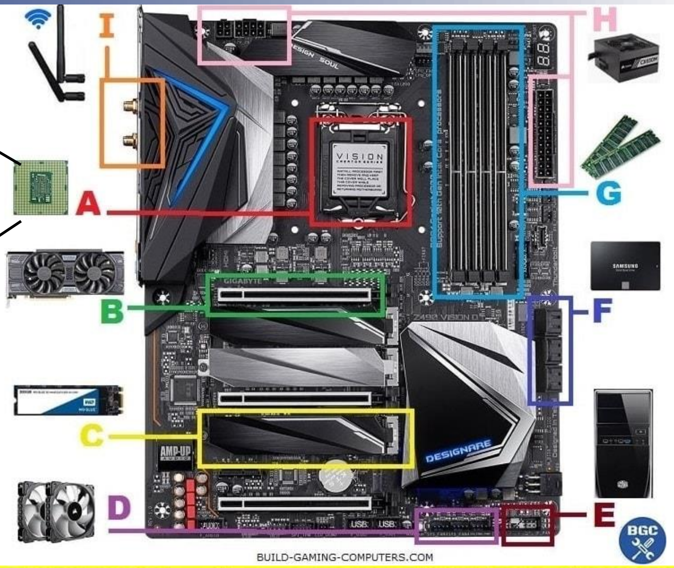


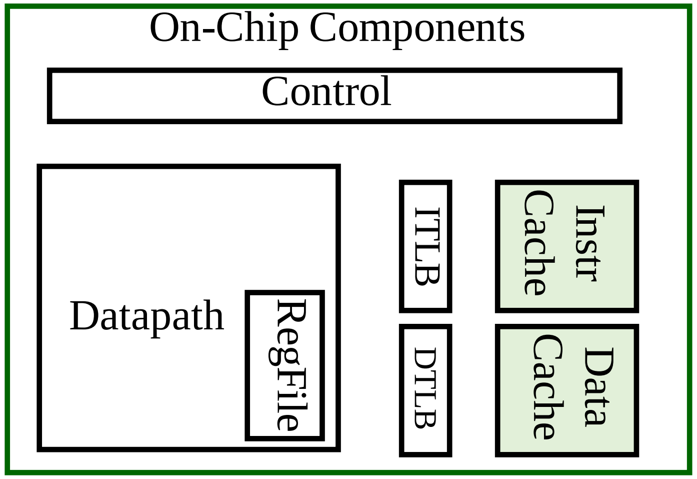
Speed (%cycles):	½'s	1's	10's	100's	10,000's
Size (bytes):	100's	10K's	M's	G's	T's
Cost:	highest				lowest

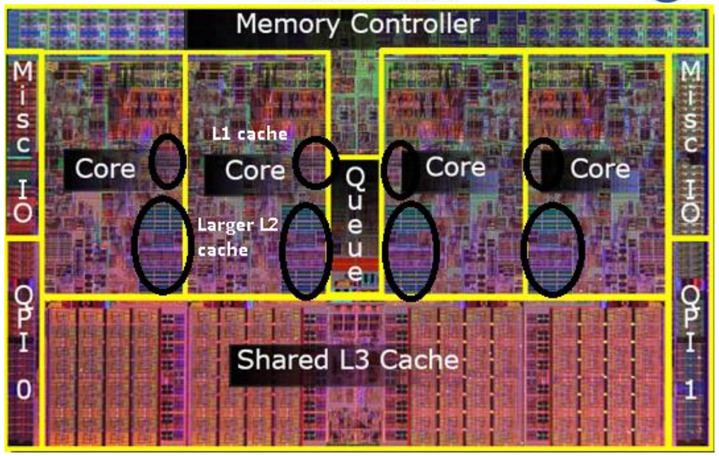
A Typical Memory Hierarchy

What we see



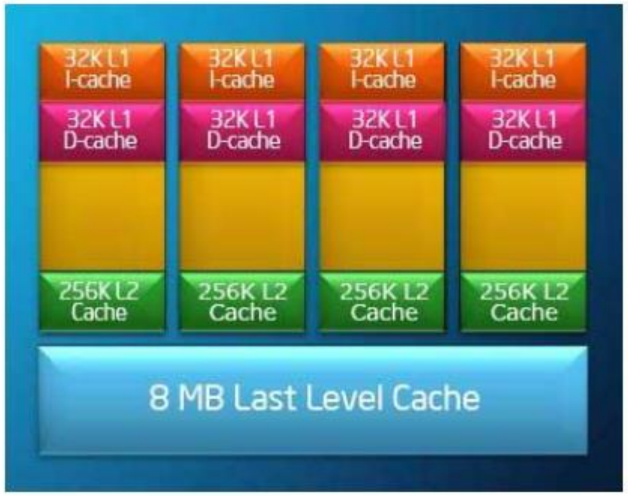


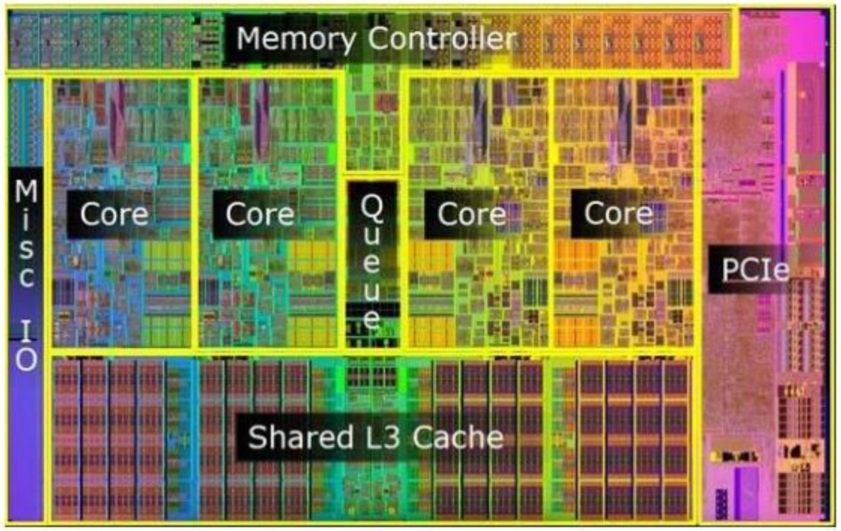




https://superuser.com/questions/196143/where-exactly-l1-l2-and-l3-caches-located-in-computer

A Typical Memory Hierarchy





Intel Core i7 cache hierarchy

https://www.slideserve.com/waldemar-buckminster/cs-201-the-memory-hierarchy-i

Memory Hierarchy: Why Does it Work?

The access patterns of instructions and data are not random!

- Temporal Locality (locality in time)
  – If a memory location is referenced then it will tend to be referenced again soon

  ⇒ Keep most recently accessed data items closer to the processor

- Spatial Locality (locality in space)
  – If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

  ⇒ Move blocks consisting of contiguous words closer to the processor

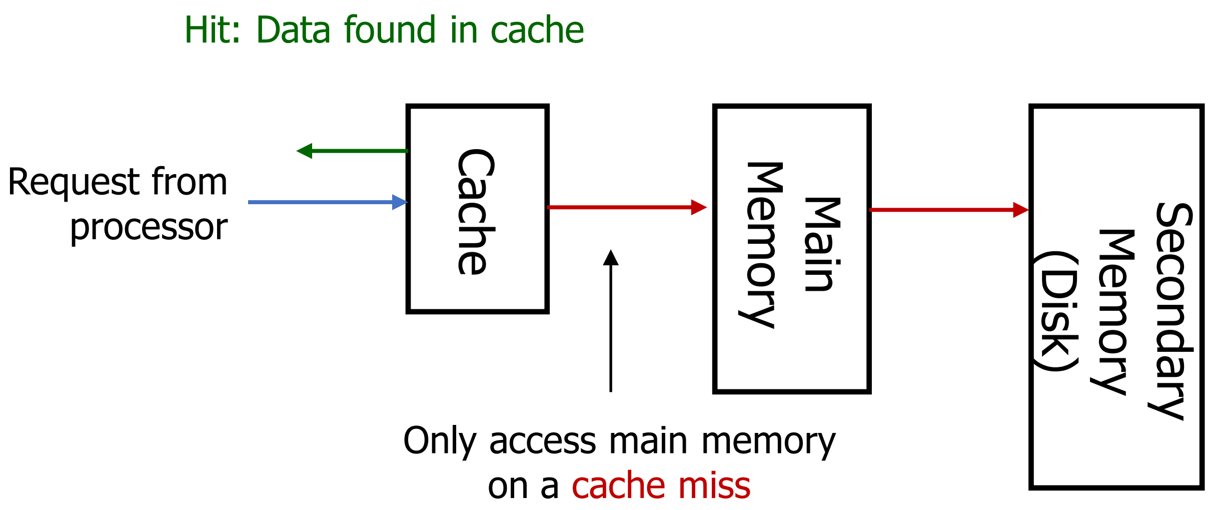
Analogy

- You are taking several courses
- Each course requires several books
- You carry to school only the books needed on that day

Table (a couple of books) Only the books you read at the moment	Register
Backpack (< 10 books) Only the books you need for a day	Cache
Bookshelf (< 100 books) Only the books you need for the semester	DRAM
Library (Many books)	Hard Disk

Cache

- A hardware component that stores data so that future requests for the data can be served faster
- Processor sends requests to cache, not to main memory
  - If data/instruction is found in cache, it is called a cache hit
  - If data/instruction is not in cache, it is called a cache miss



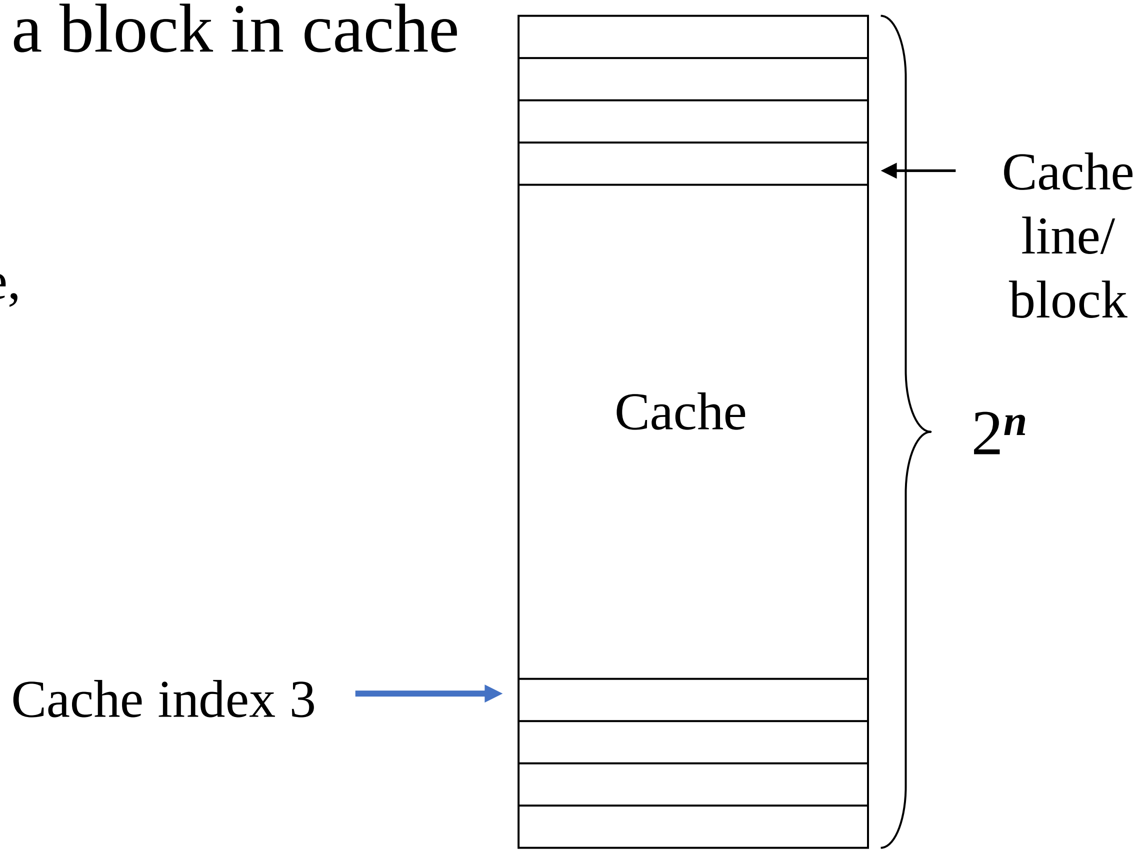
Cache blocks (cache lines)

- Cache consists of cache blocks (or lines)
  - A block is the smallest unit of data that is present in a cache
  - Block size is always a power of 2
- Cache index selects a block in cache

When loading data into cache,

the entire block is loaded.

Why?



Blocks in memory

- Since cache deals with blocks, we divide the entire main memory space into blocks

Example:

Assume block size = 16 bytes

Data in memory
As blocks

Block number	16 bytes in each block	
268435456		
268435455		
…		
2		Bytes 32 to 47 are in block 2
1		Bytes 16 to 31 are in block 1
0		Bytes 0 to 15 are in block 0

A closer look at bytes in a block

Assume block size = 16 bytes

Bytes in Block 1

Bytes in Block 0

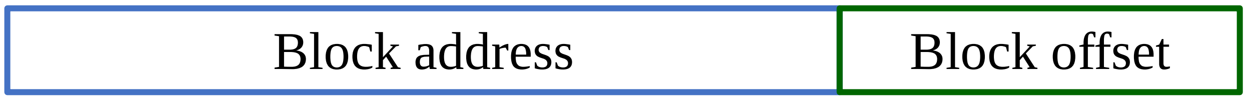
Do you see any patterns?

Block number	Address	Value
1	0000…00010011	🟩
1	0000…00010010	🟩
1	0000…00010001	🟩
1	0000…00010000	🟩
0	0000…00001111	🟨
0	0000…00001110	🟨
0	…	🟨
0	0000…00000100	🟨
0	0000…00000011	🟨
0	0000…00000010	🟨
0	0000…00000001	🟨
0	0000…00000000	🟨

Load block 0 into cache if the processor reads any bytes in the block: byte 0, byte 1, … byte 15

Bits in address

- The bits in an address can be divided two fields



- All bytes in a block have the same block address
- The offset of the bytes is different
  - Using offset, we can select every byte in a block

- The block size determines the number of bits in block offset
  - The rest of the bits is block address

How many bits are in block offset for block size of 64 bytes?

The block offset in textbook does not include byte offset. It is in words.

Question

Given an address, how do you find out its block address?

And block offset?

Example:

Assume block size = 16 bytes

What is the block address of address 0x3666?

What is the block offset?

Div and mod

Block number = 0x3666 div 16 = ?

Byte offset in the block = 0x3666 mod 16 = ?

0b 0000 0000 0000 0000 0011 0110 0110 0110

Question

Block size = 64 bytes

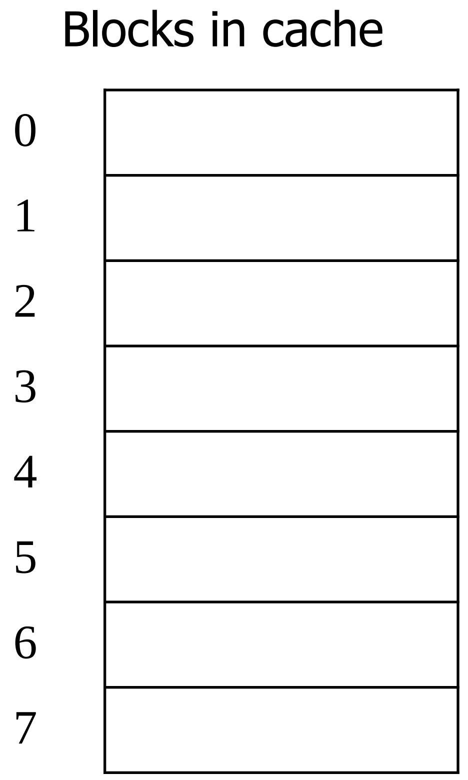
What is the block address for memory address 0x4C0?

Enter two hexadecimal digits representing the lowest 8 bits.

Example: c0

Question

- Suppose a cache has 8 blocks. Where, in the cache, should we put blocks 0, 1, 2, 3, 4, … from the main memory?
  - What is a simple way to map block address to cache index?



Block 0 goes to cache block ___.

Block 1 goes to cache block ___.

Block 2 goes to cache block ___.

…

Block 7 goes to cache block ___.

Block 8 goes to cache block ___.

Block 9 goes to cache block ___.

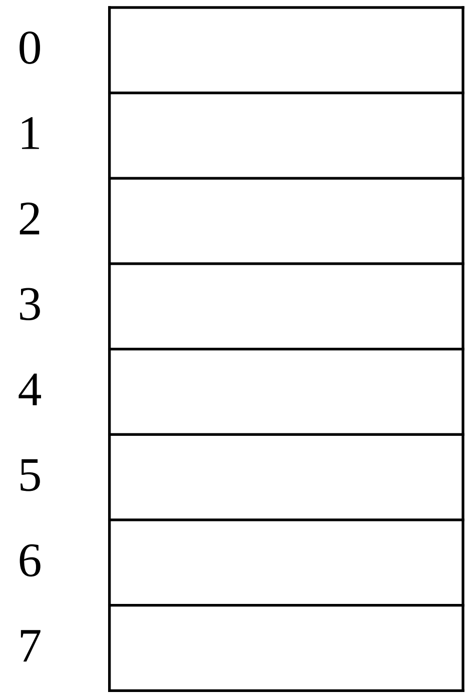
Placing a block in (direct-mapped) cache

The common way to place a block in cache:

Cache index = Block address mod Number of blocks in cache

Suppose the cache has 8 blocks.
Given a block address, can you quickly find the cache index?

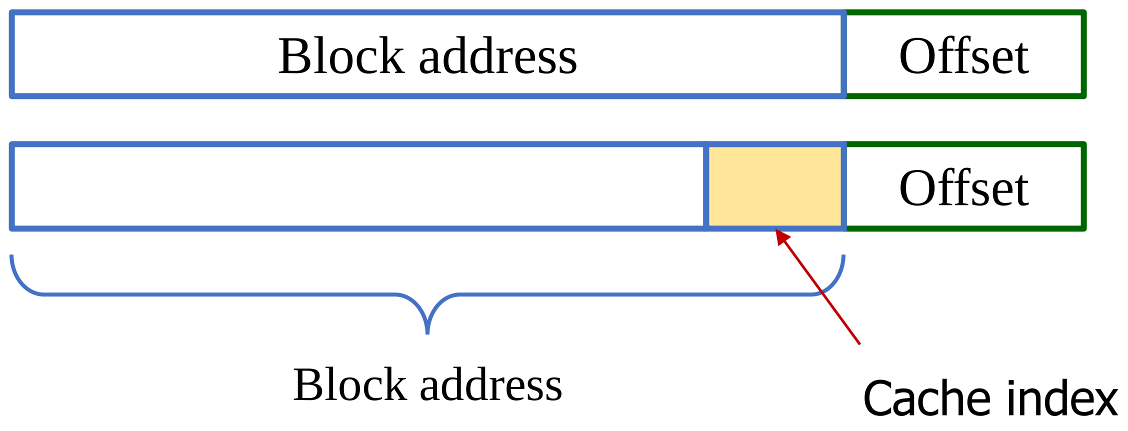
```
0b0000 0000 0000 0000 0011 0111 0101 0110
```



Cache index

- The cache index is from the lower end of block address
  - How do you find out the number of bits in the cache index?

    Hint: How do you calculate cache index?

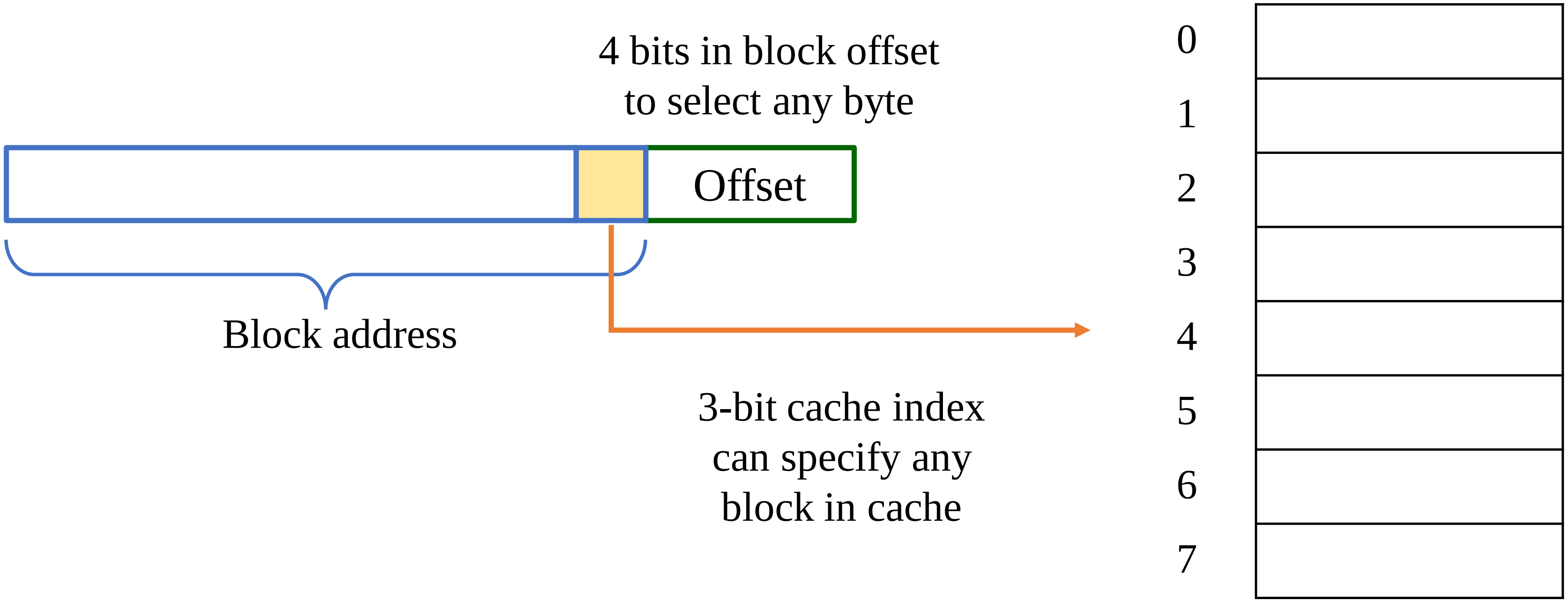


Example: Mapping from block address to cache index

Assume a cache of 8 blocks and block size is 16 bytes.

Cache index = Block address mod 8

The lowest 3 bits in block address are the cache index



Question

Assume a cache has 4 blocks and block size is 16 bytes

What is the cache index when accessing cache with the address 0x3666?

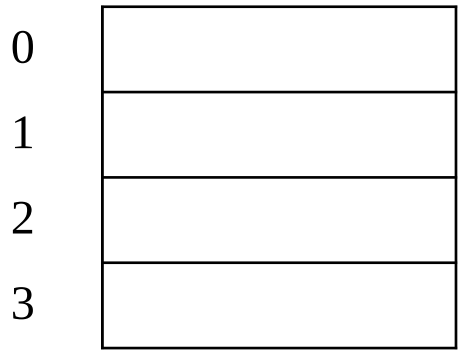
A: 0

B: 1

C: 2

D: 3

```
0b0000 0000 0000 0000 0011 0110 0110 0110
```

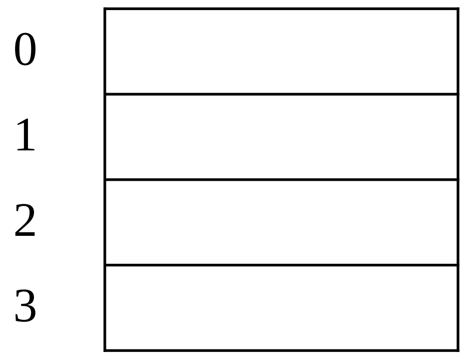


Question

Assume a cache has 4 blocks and block size is 16 bytes

What is the smallest address that has the same cache index?

```
0b0000 0000 0000 0000 0011 0110 0110 0110
```



Access cache

Assume a cache has 4 blocks and block size is 16 bytes

When the process needs to load a byte from 0x3666, the address is sent to cache. Is the byte in the cache?

```
0b0000 0000 0000 0000 0011 0110 0110 0110
```

	Data
0	
1	
2	
3	

Valid bit

Each cache block has a valid bit, indicating if the cache block contains valid data.

Is byte 0x3666 in cache?

```
0b0000 0000 0000 0000 0011 0110 0110 0110
```

	V	Data
0	0	
1	0	
2	0	
3	0	

At beginning, all cache blocks are invalid.

Placing a block in cache

Byte 0x3666 is not in cache because cache block 2 is invalid.

It is a cache miss.

The block containing 0x3666 is loaded in cache.

```
0b0000 0000 0000 0000 0011 0110 0110 0110
```

	V	Data
0	0	
1	0	
2	0	
3	0	

At beginning, all cache blocks are invalid.

	V	Data
0	0	
1	0	
2	1	🟩
3	0	

After block 0x366 is loaded into cache.

What is the lowest address in the block?
What is the highest address in the block

More references

What if the processor reads the following addresses after 0x3666 is accessed?

For each address, find the block address and the cache index, and decide if the read is a hit or a miss.

```
0x3666 0b0000 0000 0000 0000 0011 0110 0110 0110
0x3664 0b0000 0000 0000 0000 0011 0110 0110 0100
0x3660 0b0000 0000 0000 0000 0011 0110 0110 0000
0x4666 0b0000 0000 0000 0000 0100 0110 0110 0110
```

	V	Data
0	0	
1	0	
2	1	🟩
3	0	

After block 0x366 is loaded into cache.

Conflict

We have two different block addresses that have the same cache index `10`.

```
0x3666        0011 0110 0110 0110
0x4666        0100 0110 0110 0110
```

How can we tell which is in cache block 2?

Block 0x366 or block 0x466?

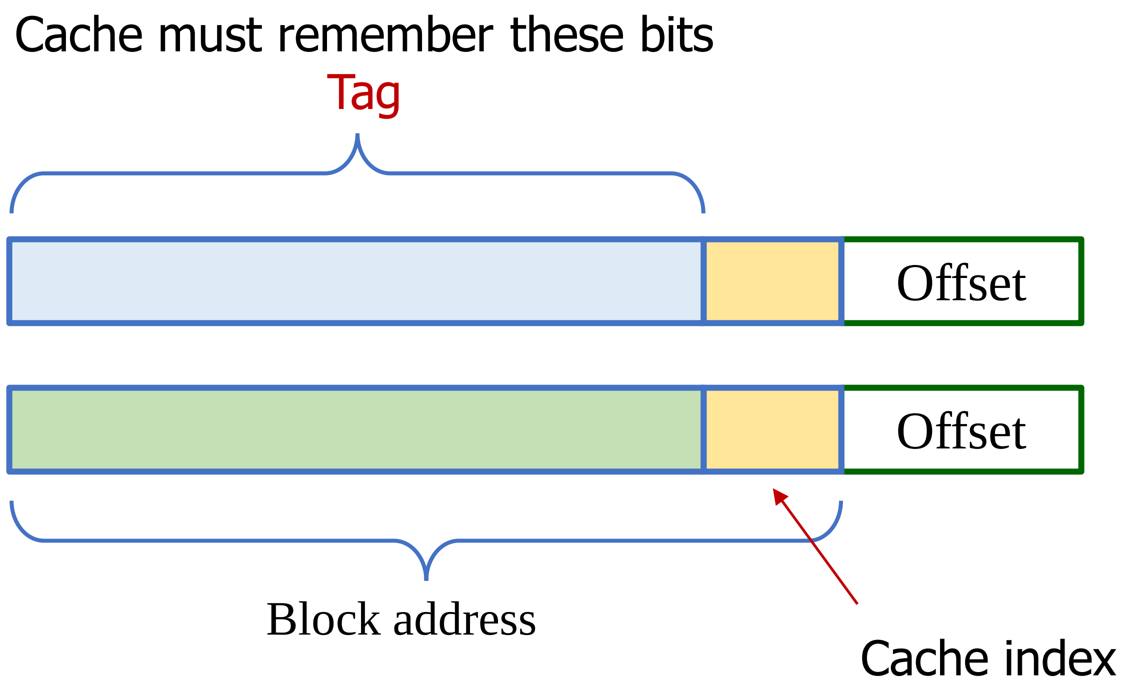
	V	Data
0	0	
1	0	
2	1	🟩
3	0	

After block 0x366 is loaded into cache.

Tag

Two different block addresses may have the same cache index

We must compare every bit in the block address



Cache with tag added

- Cache stores tags. Each cache block is associated with a tag
- Is reading 0x4666 after 0x3666 a hit or a miss?

```
0x3666 0b0000 0000 0000 0000 0011 0110 0110 0110
0x4666 0b0000 0000 0000 0000 0100 0110 0110 0110
```

	V	Tag	Data
0	0		
1	0		
2	1	000..00 0011 0110 01	
3	0		

When block 0x366 is loaded into cache, the tag is updated, too.

Cache miss

- It is a miss because the tag in the cache does not match the tag from the address

```
0x3666 0b0000 0000 0000 0000 0011 0110 0110 0110
0x4666 0b0000 0000 0000 0000 0100 0110 0110 0110
```

	V	Tag	Data
0	0		
1	0		
2	1	000..00 0011 0110 01	
3	0		

Replacement

- Block 0x466, which contains 0x4666, is loaded into cache block 2
  - We say block 0x366 is evicted from the cache
  - If we access 0x3666 again, it will be a miss

```
0x3666 0b0000 0000 0000 0000 0011 0110 0110 0110
0x4666 0b0000 0000 0000 0000 0100 0110 0110 0110
```

	V	Tag	Data
0	0		
1	0		
2	1	000..00 0100 0110 01	
3	0		

After block 0x466 is loaded into cache

Direct-Mapped Cache

- A block address is mapped to exactly one location in the cache

  Cache index = Block address mod Number of blocks in the cache

- Many blocks are mapped to the same cache block
  - A tag is associated with each cache block for identifying a block
  - Tag contains all the upper portion of the block address, which are not in the cache index

- The access is a hit if and only if the cache block is valid and the tags match
  - Otherwise, it is a miss
  - On a miss, a new block is loaded into cache, replacing any old block

Using bits in address to access a direct-mapped cache

An address has three fields:

Offset: Identify a byte/word in a block

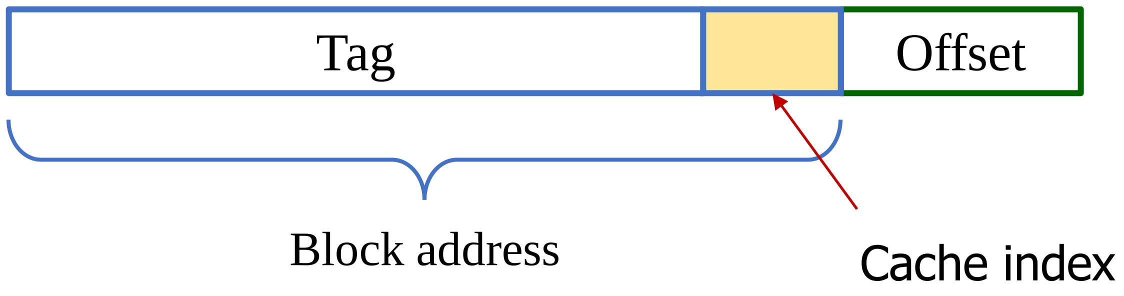
Number of bits in the offset is determined by the block size

Cache index: Locate a block in cache

Number of bits in index is determined by the number of blocks in cache

Tag: Make sure the block found in cache is the correct block

Bits in an address excluding cache index and offset bits



Cache Access Summary

Memory address is divided into 3 fields. All bits are used!

1. Use cache index to locate a block in cache
2. Check if it is the block to be accessed.

If the cache block is valid AND the tag in cache matches the tag from the address

    The access is a hit

    Use the offset to select the byte/word

Else

    It is a miss

    Load a block from memory into cache

    Update tag

    Set the valid bit

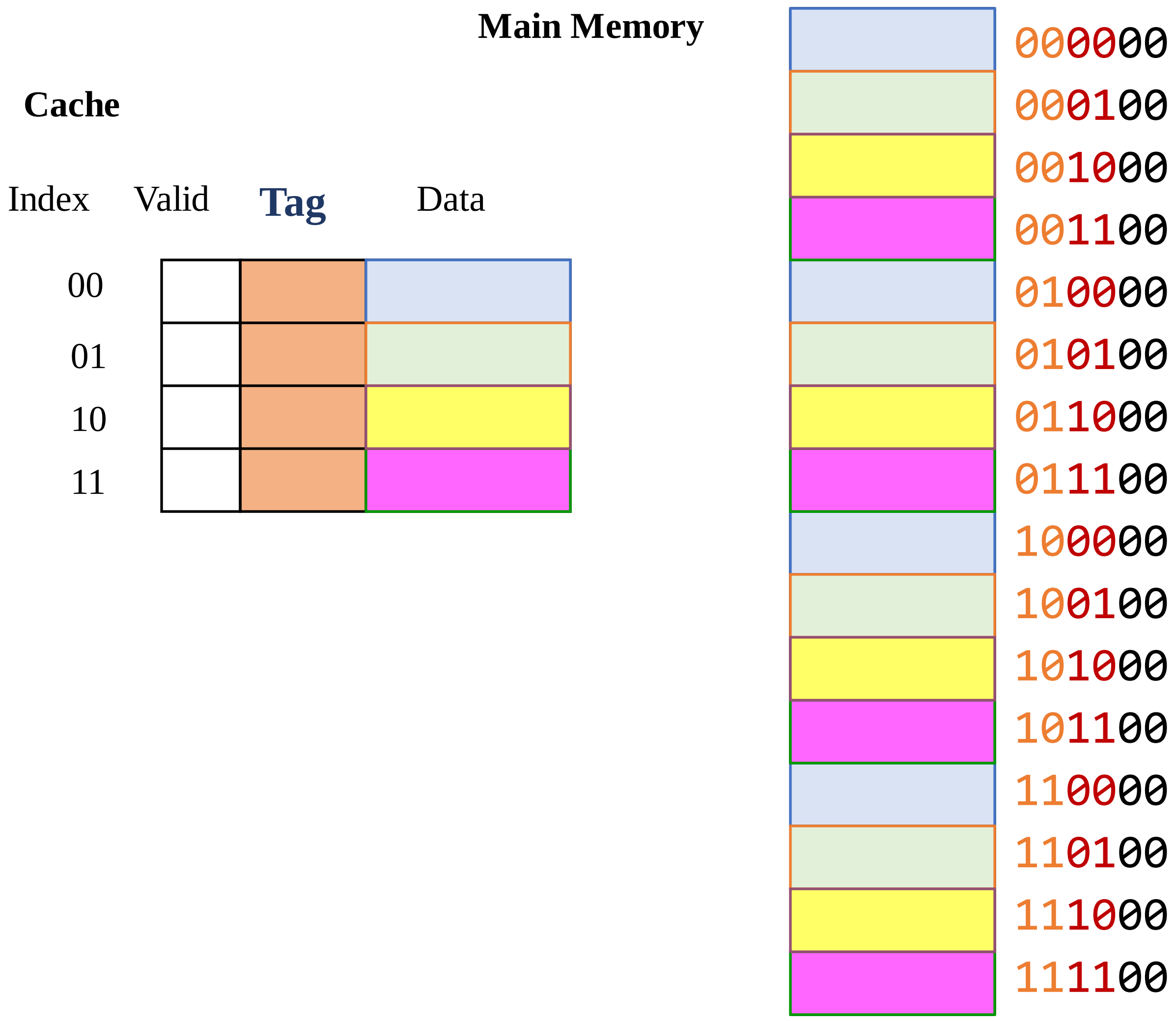
    Access cache again

Exercises with a simple cache

- A cache has only 4 blocks, each block is one word (4 bytes)
- An address has only 6 bits
  - Or the higher 26 bits are all 0

- Assume all requested data are words
- Mem[0] means the word at address 0, Mem[4] means the word at address 4, and so on
- A blank cache block means the block is invalid

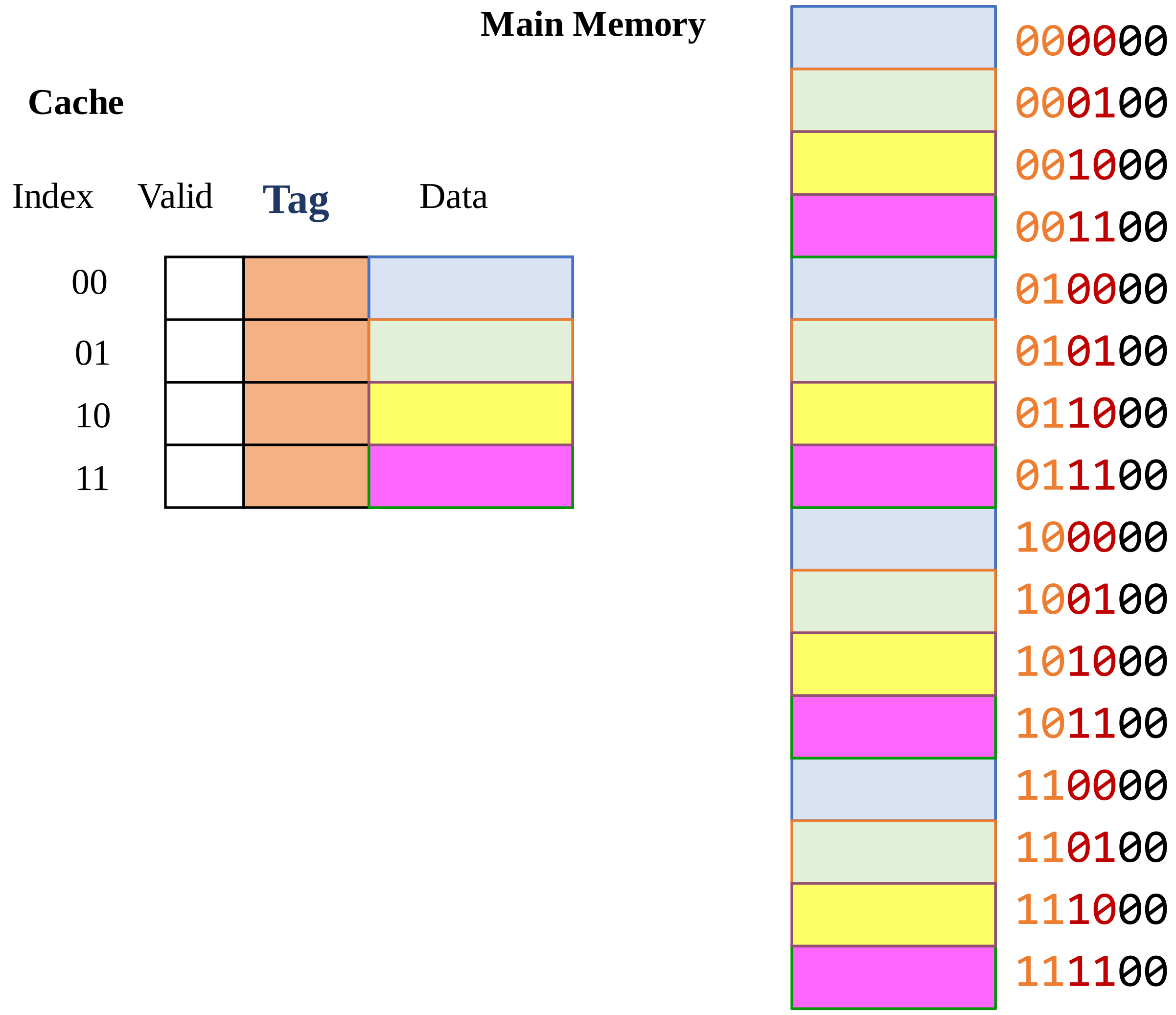
A Simple Cache



One word (4 bytes) blocks
Block offset (2 bits) specifies a byte in the word

(block address) modulo (# of blocks in the cache)

A Simple Cache



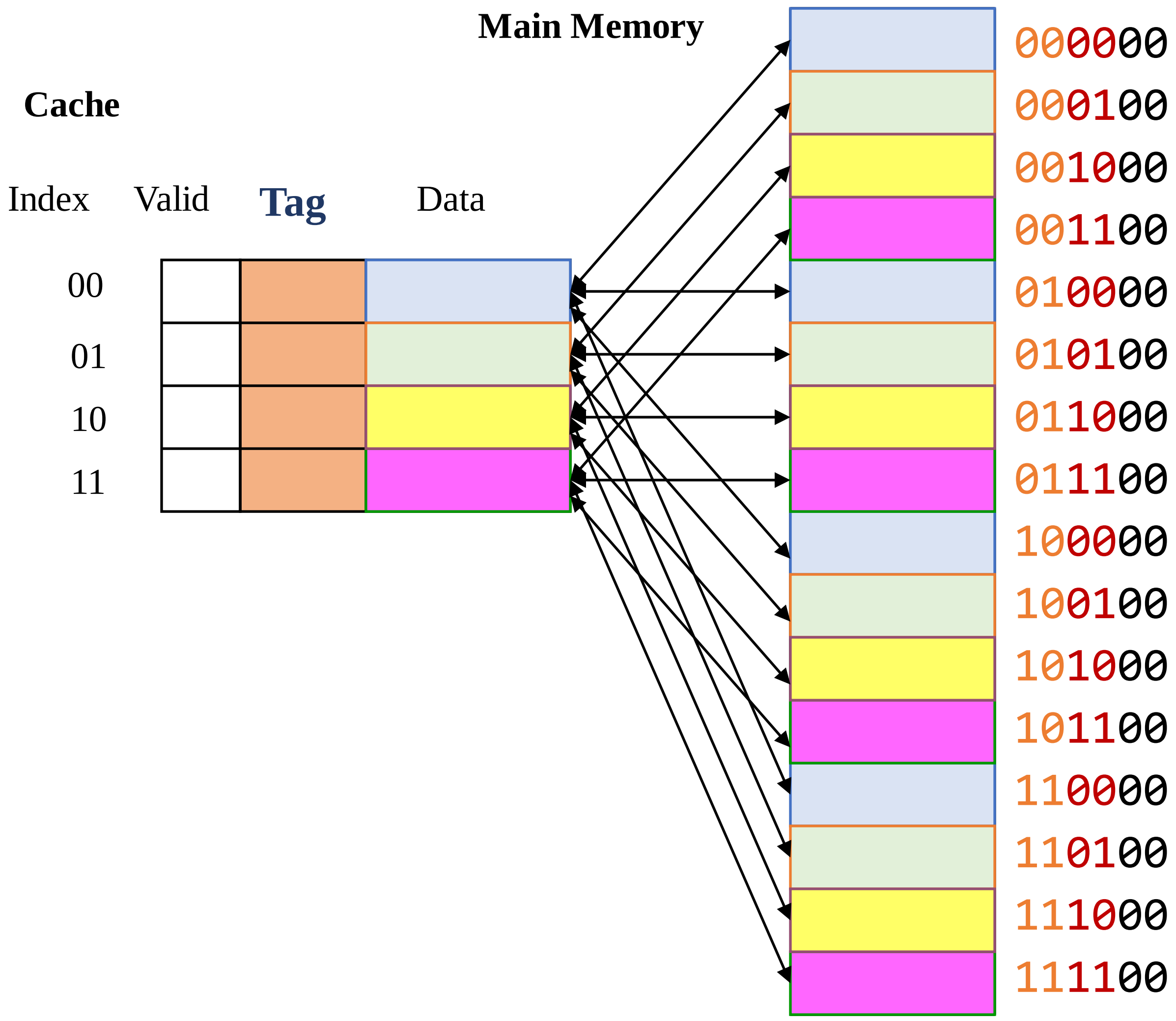
One word (4 bytes) blocks
Block offset (2 bits) specifies a byte in the word

Q1:
Where can we place it?
Where can we find it?

Use 2 lowest bits in block address as cache index to select a cache block (i.e., modulo the number of blocks in the cache)

(block address) modulo (# of blocks in the cache)

A Simple Cache



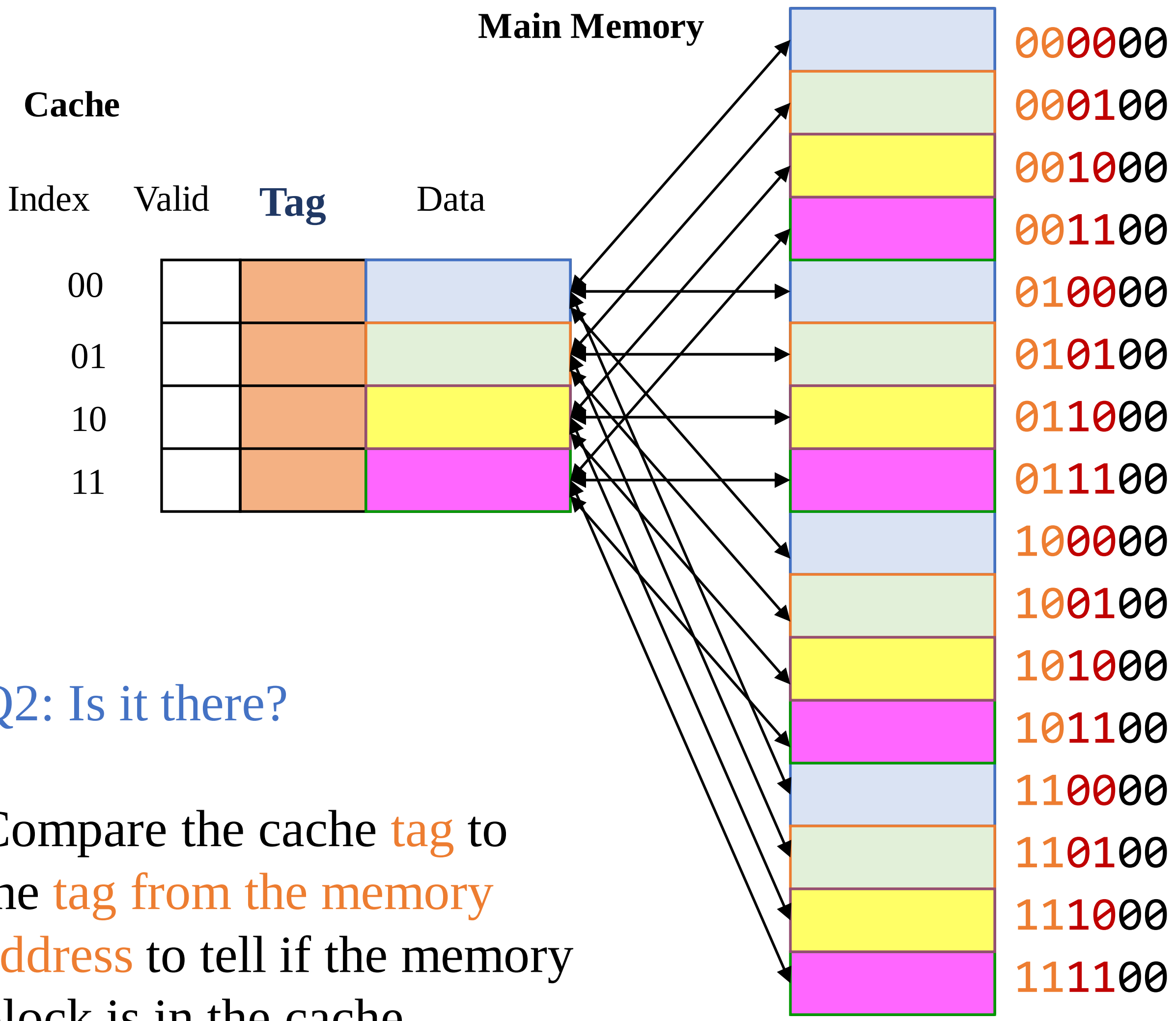
One word (4 bytes) blocks
Block offset (2 bits) specifies a byte in the word

Q1:
Where can we place it?
Where can we find it?

Use 2 lowest bits in block address as cache index to select a cache block (i.e., modulo the number of blocks in the cache)

(block address) modulo (# of blocks in the cache)

A Simple Cache



One word (4 bytes) blocks
Block offset (2 bits) specifies a byte in the word

Q1:
Where can we place it?
Where can we find it?

Use 2 lowest bits in block address as cache index to select a cache block (i.e., modulo the number of blocks in the cache)

Q2: Is it there?

Compare the cache tag to the tag from the memory address to tell if the memory block is in the cache

(block address) modulo (# of blocks in the cache)

Exercises

Start with an empty cache (all blocks are not valid/blank).

Access the words at the following addresses in sequence.

0	4	8	12	16	12	16	60
000000	000100	001000	001100	010000	001100	010000	111100

0

Tag	

4

8

12

16

12

16

60

Exercises

Start with an empty cache (all blocks are not valid/blank).

Access the words at the following addresses in sequence.

0	4	8	12	16	12	16	60
000000	000100	001000	001100	010000	001100	010000	111100

Tag **0** miss

4

8

12

16

12

16

60

Exercises

Start with an empty cache (all blocks are not valid/blank).

Access the words at the following addresses in sequence.

0	4	8	12	16	12	16	60
000000	000100	001000	001100	010000	001100	010000	111100

Tag **0** miss

00	Mem[0]

4

8

12

16

12

16

60

Exercises

Start with an empty cache (all blocks are not valid/blank).

Access the words at the following addresses in sequence.

0	4	8	12	16	12	16	60
000000	000100	001000	001100	010000	001100	010000	111100

Tag **0** miss

00	Mem[0]

4

00	Mem[0]

8

12

16

12

16

60

Exercises

Start with an empty cache (all blocks are not valid/blank).

Access the words at the following addresses in sequence.

0	4	8	12	16	12	16	60
000000	000100	001000	001100	010000	001100	010000	111100

Tag **0** miss

00	Mem[0]

4 miss

00	Mem[0]

8

12

16

12

16

60

Exercises

Start with an empty cache (all blocks are not valid/blank).

Access the words at the following addresses in sequence.

0	4	8	12	16	12	16	60
000000	000100	001000	001100	010000	001100	010000	111100

Tag **0** miss

00	Mem[0]

4 miss

00	Mem[0]
00	Mem[4]

8

12

16

12

16

60

Exercises

Start with an empty cache (all blocks are not valid/blank).

Access the words at the following addresses in sequence.

0	4	8	12	16	12	16	60
000000	000100	001000	001100	010000	001100	010000	111100

Tag **0** miss

00	Mem[0]

4 miss

00	Mem[0]
00	Mem[4]

8

00	Mem[0]
00	Mem[4]

12

16

12

16

60

Exercises

Start with an empty cache (all blocks are not valid/blank).

Access the words at the following addresses in sequence.

0	4	8	12	16	12	16	60
000000	000100	001000	001100	010000	001100	010000	111100

Tag **0** miss

00	Mem[0]

4 miss

00	Mem[0]
00	Mem[4]

8 miss

00	Mem[0]
00	Mem[4]

12

16

12

16

60

Exercises

Start with an empty cache (all blocks are not valid/blank).

Access the words at the following addresses in sequence.

0	4	8	12	16	12	16	60
000000	000100	001000	001100	010000	001100	010000	111100

Tag **0** miss

00	Mem[0]

4 miss

00	Mem[0]
00	Mem[4]

8 miss

00	Mem[0]
00	Mem[4]
00	Mem[8]

12

16

12

16

60

Exercises

Start with an empty cache (all blocks are not valid/blank).

Access the words at the following addresses in sequence.

0	4	8	12	16	12	16	60
000000	000100	001000	001100	010000	001100	010000	111100

Tag

0 miss

00	Mem[0]

4 miss

00	Mem[0]
00	Mem[4]

8 miss

00	Mem[0]
00	Mem[4]
00	Mem[8]

12 miss

16

12

16

60

Exercises

Start with an empty cache (all blocks are not valid/blank).

Access the words at the following addresses in sequence.

0	4	8	12	16	12	16	60
000000	000100	001000	001100	010000	001100	010000	111100

Tag

0 miss

00	Mem[0]

4 miss

00	Mem[0]
00	Mem[4]

8 miss

00	Mem[0]
00	Mem[4]
00	Mem[8]

12 miss

16

12

16

60

Find out the data and tags in cache after each access. How many misses in total?

Memory Cost

Technology	Access Time (ns)	$/GB (in 2012)
SRAM	0.2– 2.5	$500 – 1000
DRAM	50– 70	$10 – 20
Flash	5,000 – 50,000	$0.75 – 1
Magnetic Disk	$5 \times 10^6 – 20 \times 10^6$	$0.05 – 0.10