CSCI 210: Computer Architecture
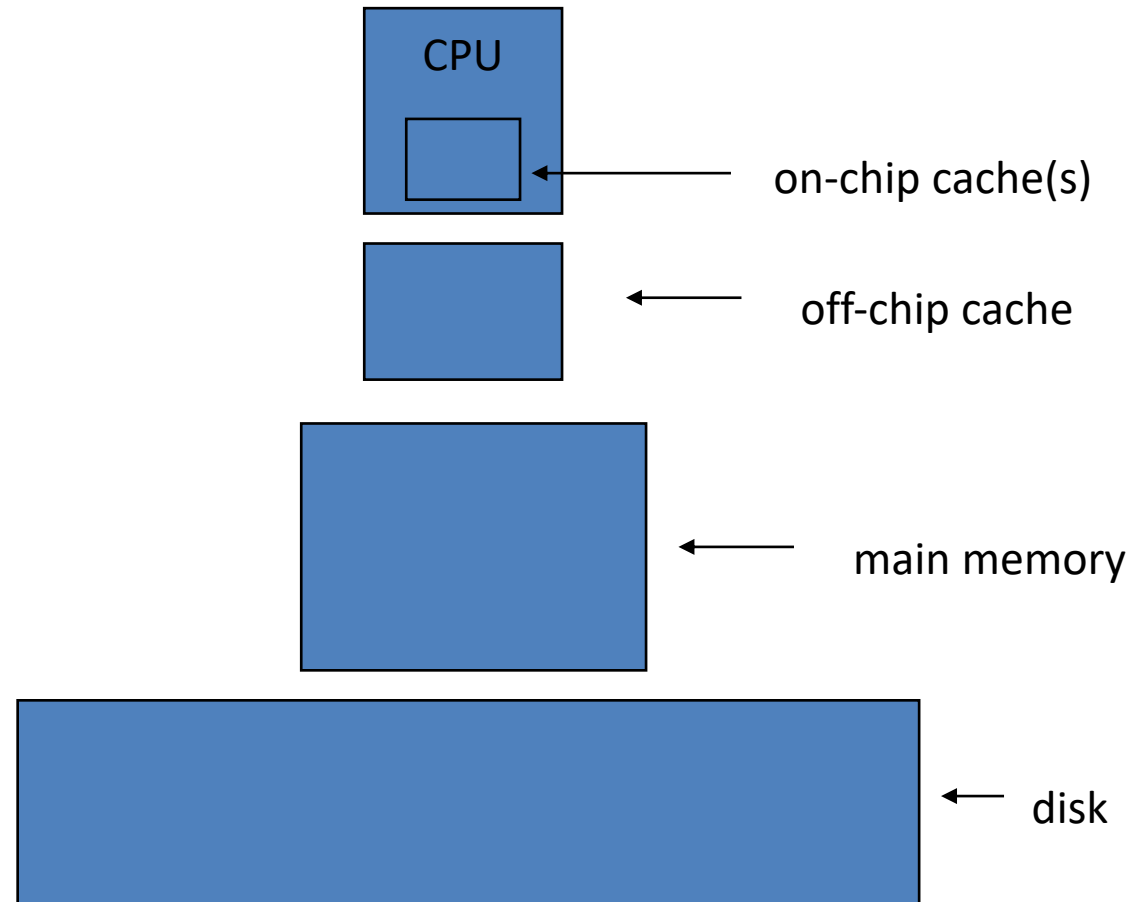# Lecture 34: Caches II

Stephen Checkoway

Oberlin College

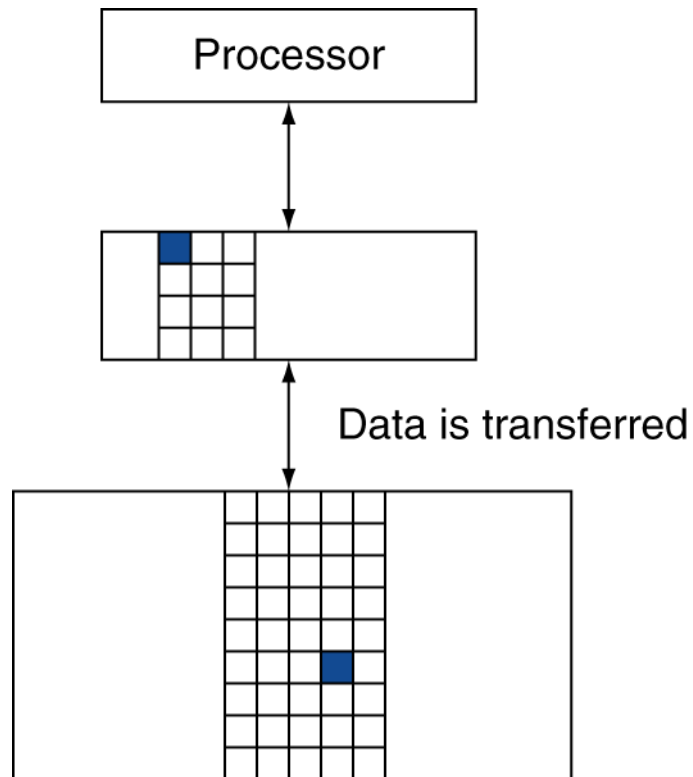Jan. 5, 2022

Slides from Cynthia Taylor

Memory Access

- Use main memory addresses

- When looking for data, check
  - 1. cache
  - 2. main memory
  - 3. disk

CPU

on-chip cache(s)

off-chip cache

main memory

disk

Memory Hierarchy Terms

Processor

Data is transferred
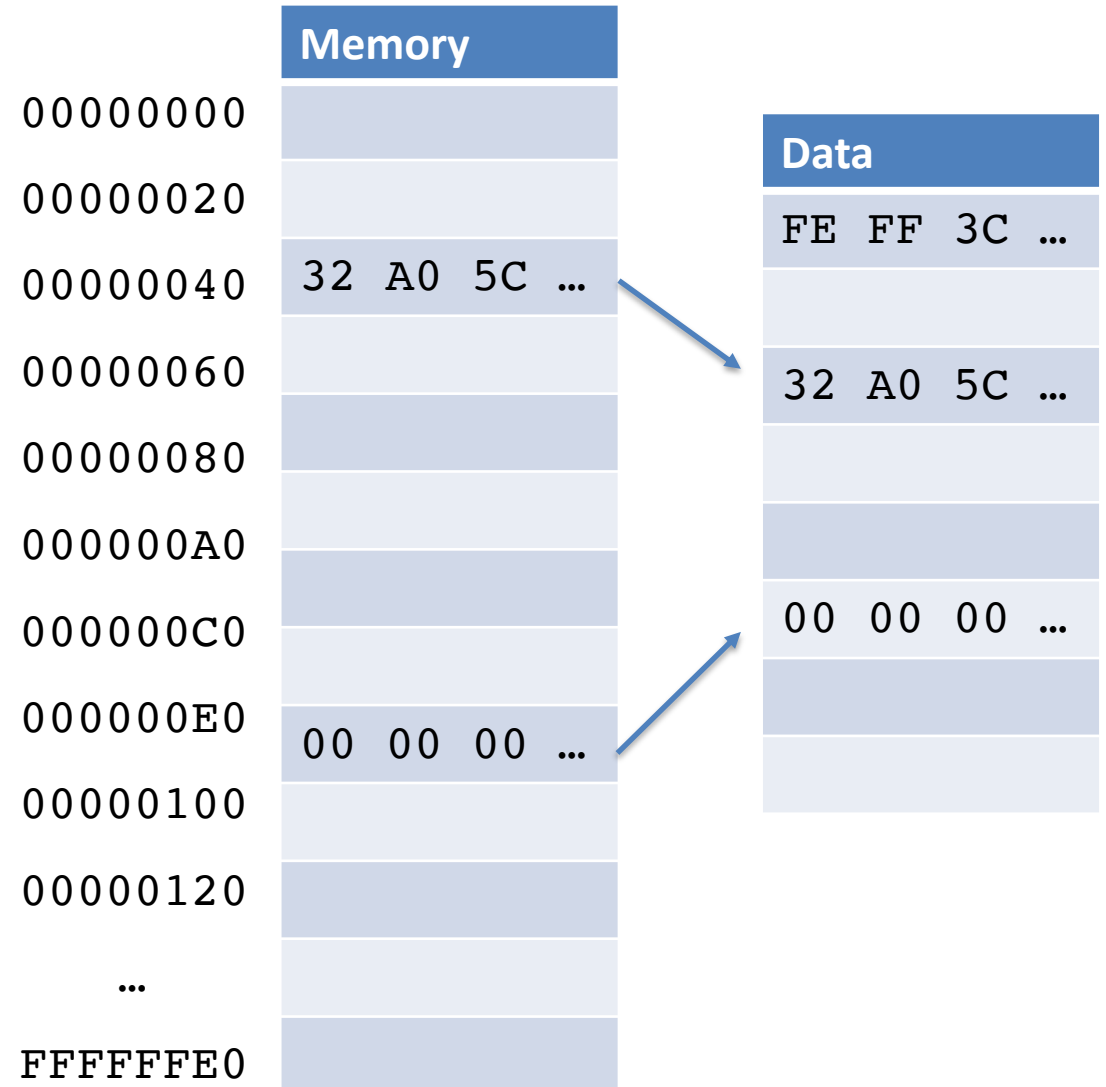
- Block: unit of copying
  - May be multiple words
  - On x86-64, a block is 64 bytes
- Hit: data in the cache
  - Hit ratio: hits/accesses
- Miss: data not in the cache
  - Time taken: miss penalty
  - Miss ratio: misses/accesses
    = 1 – hit ratio

High-level cache strategy

- Divide all of memory into consecutive blocks

- Copy data (memory ↔ cache) one block at a time

- To access data, check if it exists in the cache before checking memory

Memory	
00000000	
00000020	
00000040	32 A0 5C …
00000060	
00000080	
000000A0	
000000C0	
000000E0	00 00 00 …
00000100	
00000120	
…	
FFFFFFE0	

Data
FE FF 3C …
32 A0 5C …
00 00 00 …

Memory addresses, block addresses, offsets

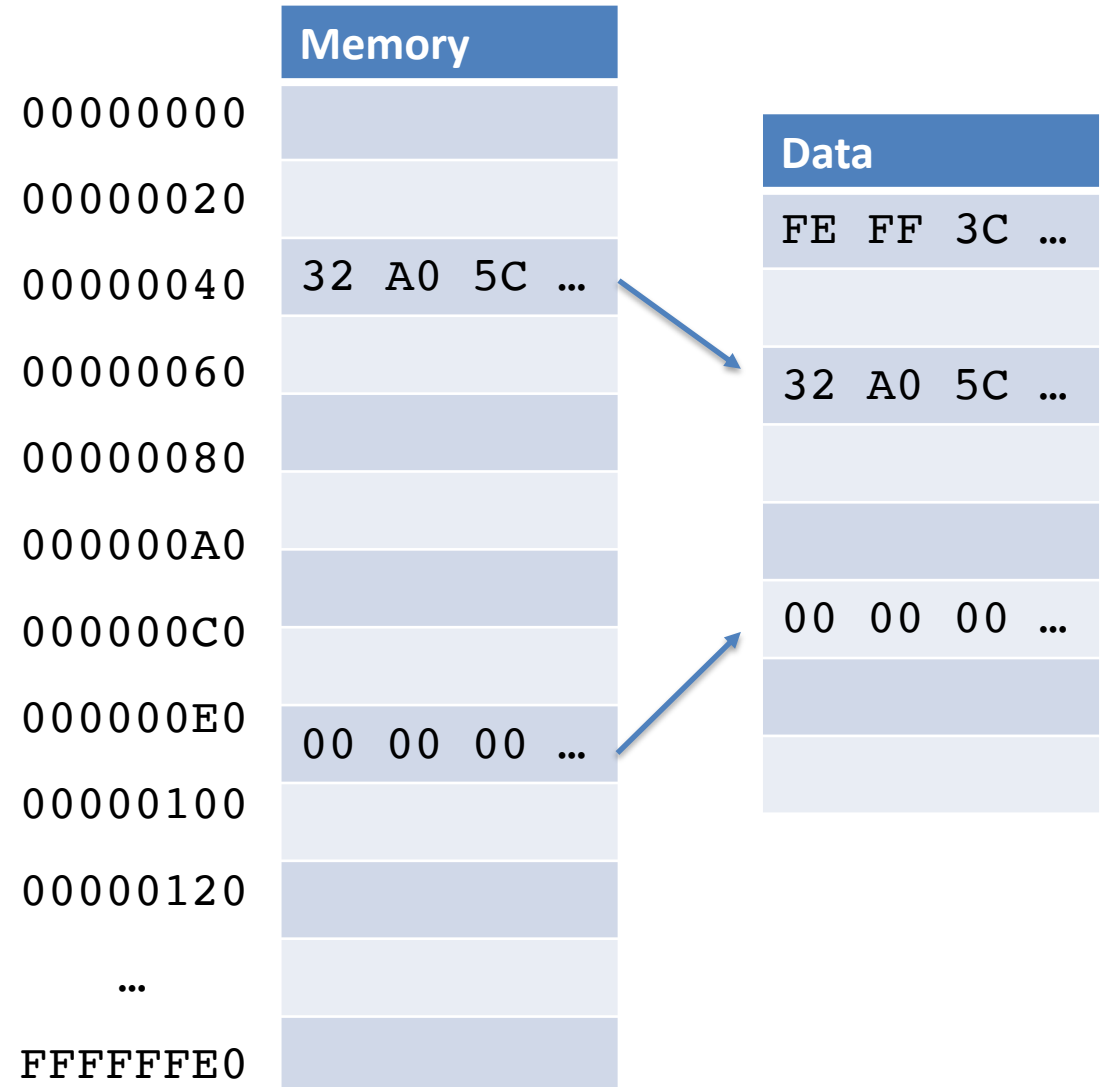| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

- Imagine we have blocks of size 32 bytes (not bits!)
- Every byte of memory can be specified by giving
  - A (32 − 5)-bit block address (in purple)
  - A 5-bit offset into the block (in green)
- To read a byte of memory
  - find the appropriate 32-byte block in either cache or memory using the block address
  - Use the offset to select the appropriate byte from the block

With a block size of 64 bytes, how many bits is the block address? How many bits is the offset? (Assume 32-bit addresses.)

A. Block address size is 32 – 4 = 28 bits; offset size is 4 bits

B. Block address size is 32 – 5 = 27 bits; offset size is 5 bits

C. Block address size is 32 – 6 = 26 bits; offset size is 6 bits

D. Block address size is 32 – 5 = 27 bits; offset size is 4 bits

E. Block address size is 32 – 5 = 27 bits; offset size is 6 bits

Where is a block of memory stored in cache?

- Given a memory address, we can divide it into a block address and an offset

- Where in cache is the block stored?

- Basic problem: Cache is smaller than main memory

Memory	
00000000	
00000020	
00000040	32 A0 5C …
00000060	
00000080	
000000A0	
000000C0	
000000E0	00 00 00 …
00000100	
00000120	
…	
FFFFFFE0	

Data
FE FF 3C …
32 A0 5C …
00 00 00 …

Direct-mapped cache

- Block location in cache determined by block address
- Direct mapped: only one possible location
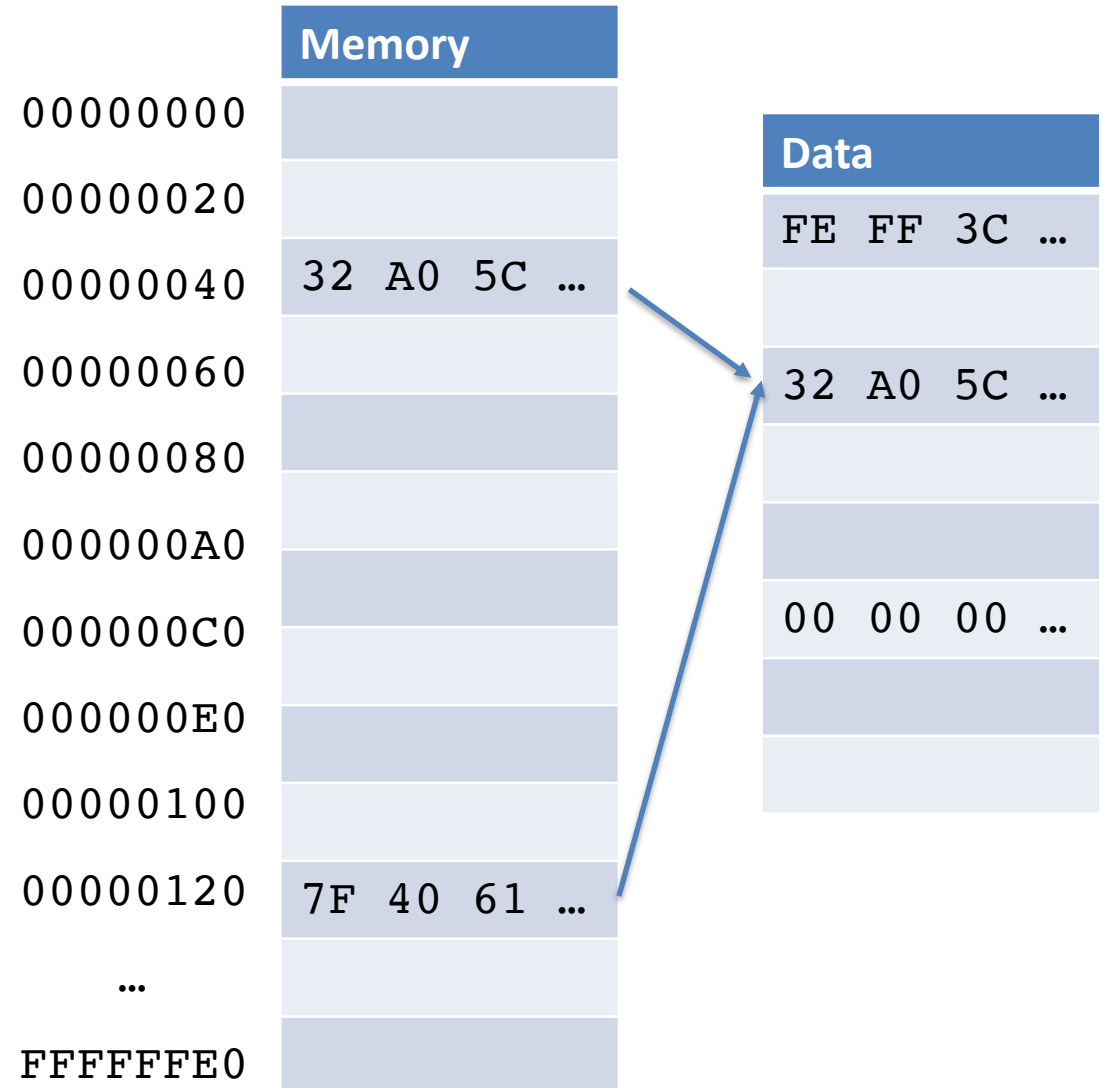 - (Block address) modulo (#Blocks in cache)
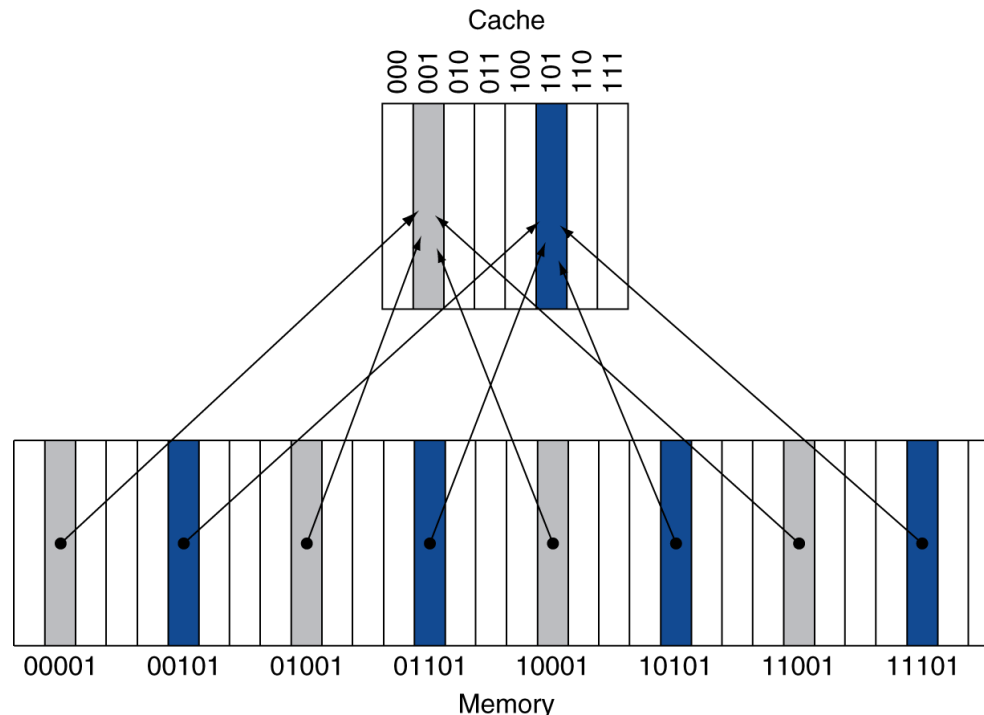
- #Blocks is a power of 2
- Use low-order address bits

Problem: Collisions

- Many block addresses map to the same cache location
- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the **tag**

Memory	
00000000	
00000020	
00000040	32 A0 5C …
00000060	
00000080	
000000A0	
000000C0	
000000E0	
00000100	
00000120	7F 40 61 …
…	
FFFFFFE0	

Data
FE FF 3C …
32 A0 5C …
00 00 00 …

Memory addresses, block addresses, offsets

| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

- Block size of 32 bytes (not bits!)
- 8-block cache (this is purely an example!)
- Each address
  - A (32 – 5)-bit block address (in purple and blue)
  - A 5-bit offset into the block (in green)
- Block address can be divided into
  - A (32 – 3 – 5)-bit **tag** (purple)
  - A 3-bit cache **index** (blue)

If we have a block size of 64-bytes and our cache holds 256 entries how large are the tag, index, and offset?

tag	index	offset

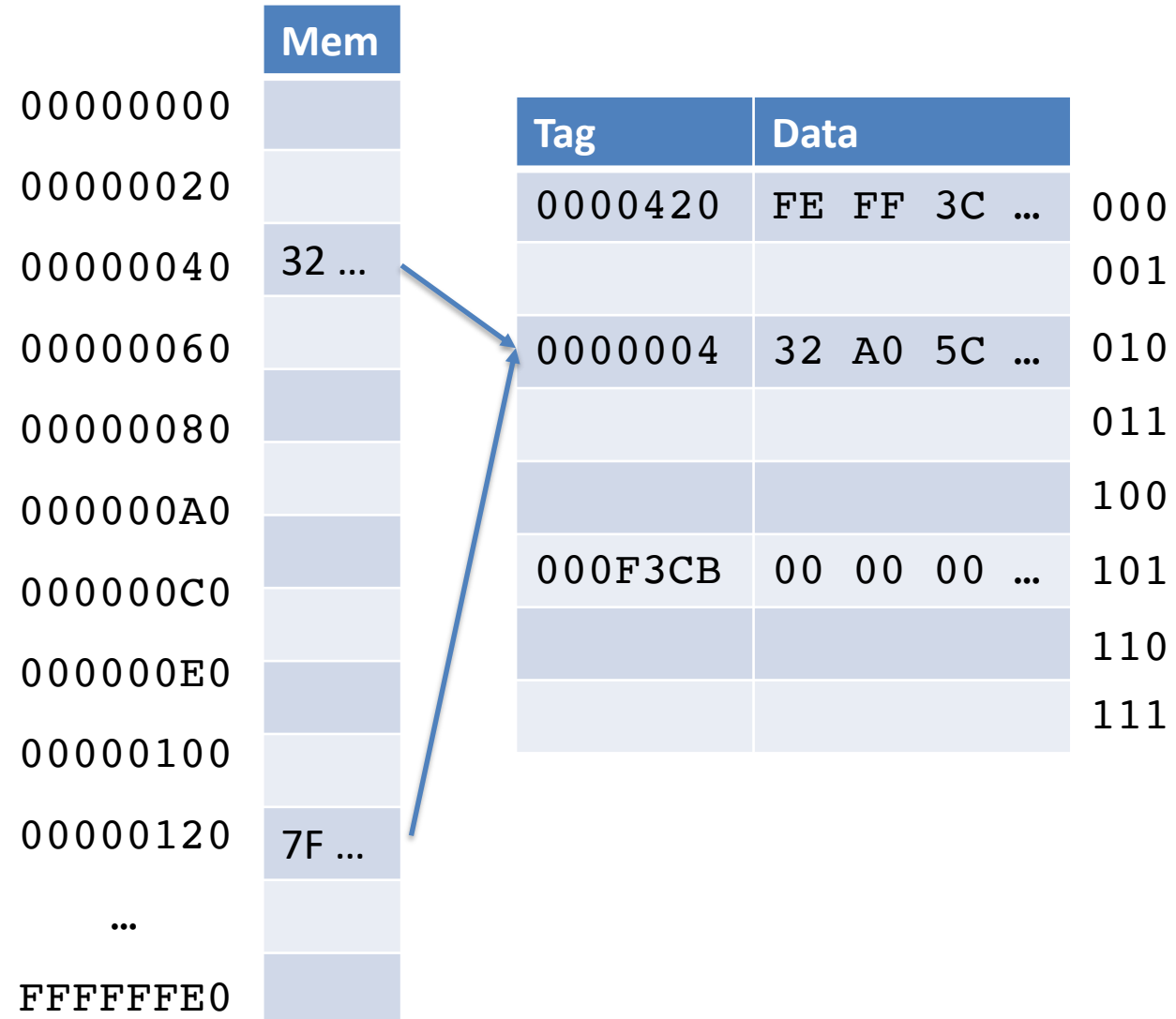|	Tag size (bits)	Index size (bits)	Offset size (bits)
A	$32 - 3 - 8$	3	8
B	$32 - 3 - 6$	3	6
C	$32 - 6 - 8$	6	8
D	$32 - 8 - 6$	8	6
E	$32 - 8 - 8$	8	8

Cache layout (so far)

- Tag stores high-order bits of address

- Data stores all of the data for the block (e.g., 32 bytes)

Tag	Data
0000420	FE FF 3C 7F …
0012345	32 A0 5C 21 …
000F3CB	00 00 00 00 …

High-level cache strategy

- Divide all of memory into consecutive blocks

- Copy data (memory ↔ cache) one block at a time

- Cache lookup:
  – Get the index of the block in the cache from the address
  – Compare the tag from the address with the tag in the cache

Mem

00000000	
00000020	
00000040	32 …
00000060	
00000080	
000000A0	
000000C0	
000000E0	
00000100	
00000120	7F …
…	
FFFFFFE0	

Tag	Data	
0000420	FE FF 3C …	000
		001
0000004	32 A0 5C …	010
		011
		100
000F3CB	00 00 00 …	101
		110
		111

How do we know if it's in the cache?

- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
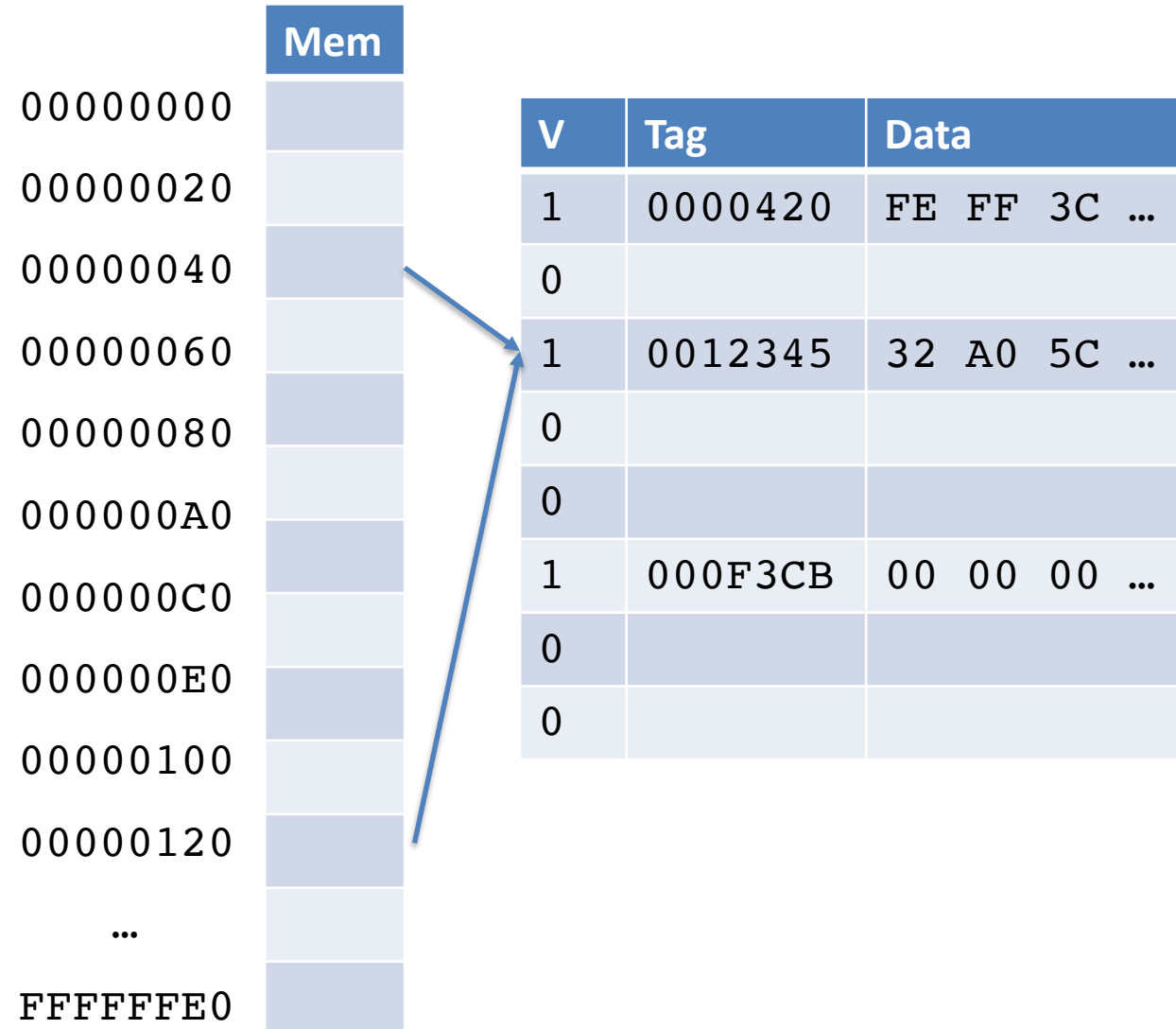 - Initially 0

Direct-mapped cache layout

- Valid stores 1 if data is present in cache

- Tag stores high-order bits of address

- Data stores all of the data for the block (e.g., 32 bytes)

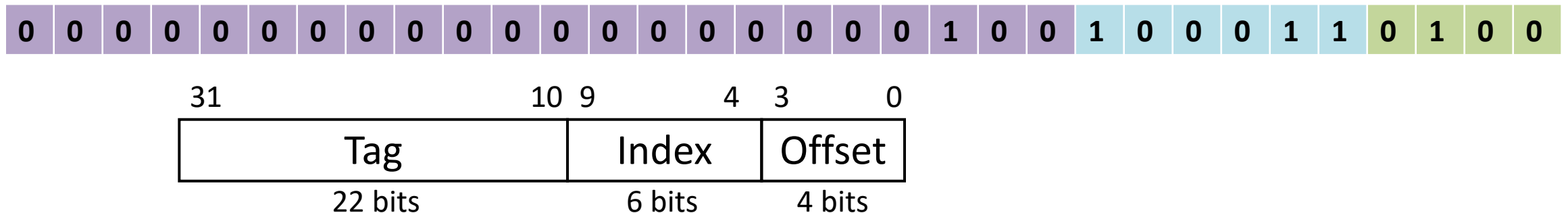| Valid	Tag	Data
1	0000420	FE FF 3C 7F …
0		
1	0012345	32 A0 5C 21 …
0		
0		
1	000F3CB	00 00 00 00 …
0		
0		

High-level cache strategy

- Divide all of memory into consecutive blocks

- Copy data (memory ↔ cache) one block at a time

- Cache lookup:
  - Get the index of the block in the cache from the address
  - Check the valid bit; compare the tag to the address

Mem
00000000
00000020
00000040
00000060
00000080
000000A0
000000C0
000000E0
00000100
00000120
…
FFFFFFE0

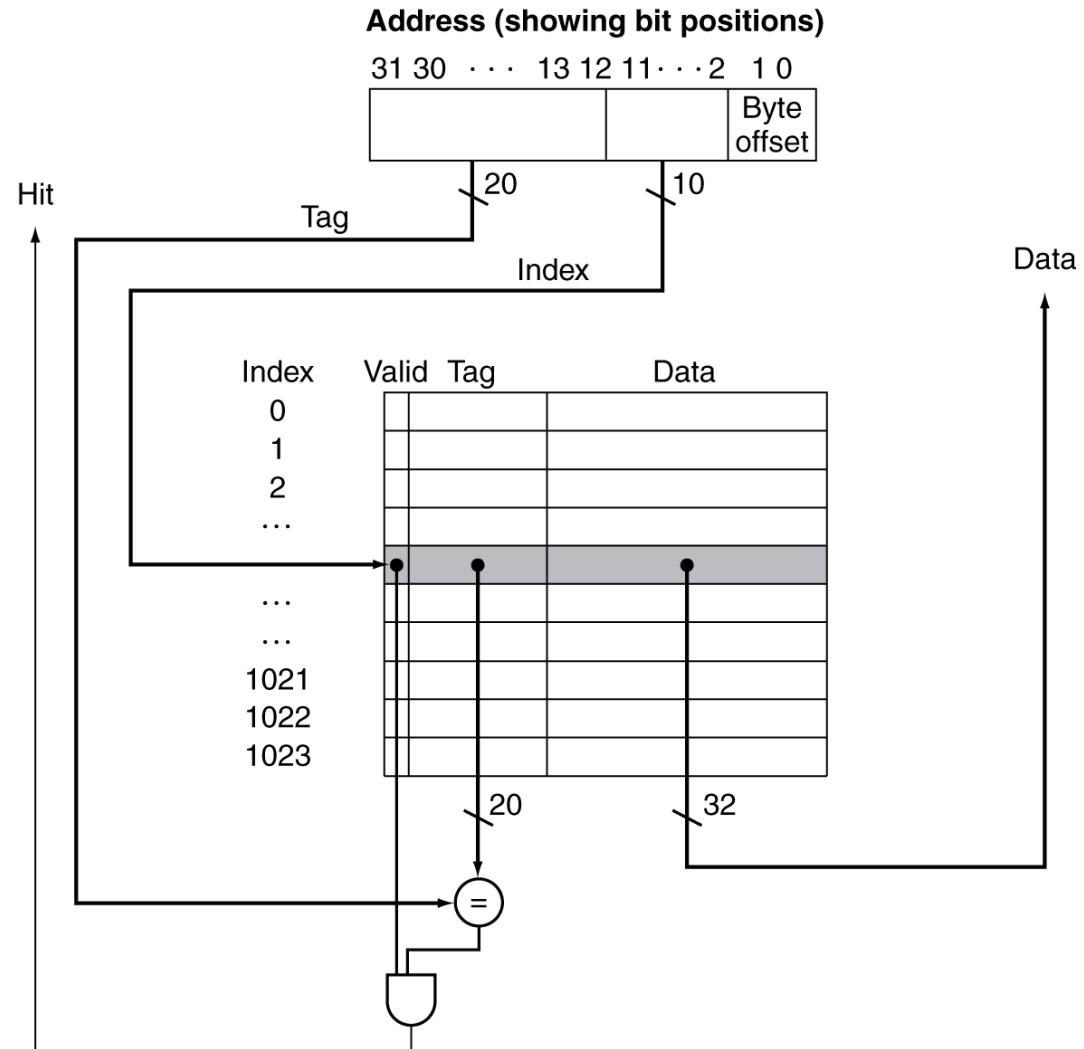| V	Tag	Data
1	0000420	FE FF 3C …
0		
1	0012345	32 A0 5C …
0		
0		
1	000F3CB	00 00 00 …
0		
0		

Example

- 64 blocks, 16 bytes/block
    - To what cache index does address 0x1234 map?

- Block address = $\lfloor 0x1234/16 \rfloor$ = 0x123

- Index = 0x123 modulo 64 = 0x23

- No actual math required: just select appropriate bits from address!

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |

31		10	9		4	3	0
	Tag			Index		Offset	
	22 bits			6 bits		4 bits	

Memory access

Direct Mapped Cache

data	byte addresses	A	B	C	D
x	00 00 01 00	M	M	M	M
y	00 00 10 00	M	M	M	H
z	00 00 11 00	M	M	M	M
x	00 00 01 00	H	H	H	H
y	00 00 10 00	H	H	H	H
w	00 01 01 00	M	M	M	M
x	00 00 01 00	M	M	H	H
y	00 00 10 00	H	H	H	H
w	00 01 01 00	H	M	H	H
u	00 01 10 00	M	M	M	M
z	00 00 11 00	H	H	M	H
y	00 00 10 00	H	M	H	H
x	00 00 01 00	H	M	M	M

E None are correct

	tag	data
00		
01		
10		
11		

Four blocks, each block holds four bytes

Reading

- Next lecture: More Caches!
  - Section 6.4

- Problem Set 11 due Friday