COMP-273 – Machine Structures

Caches, Part I

Kaleem Siddiqi

Slides from Patterson's 61C

Outline

° **Memory Hierarchy**

° **Direct-Mapped Cache**

° **Types of Cache Misses**

° **A (long) detailed example**

Slides from Patterson's 61C

Memory Hierarchy (1/4)

° **Processor**

- **executes programs**

- **runs on order of nanoseconds to picoseconds**

- **needs to access code and data for programs: where are these?**

° **Disk**

- **HUGE capacity (virtually limitless)**

- **VERY slow: runs on order of milliseconds**

- **so how do we account for this gap?**

Slides from Patterson's 61C

Memory Hierarchy (2/4)

○ **Memory (DRAM)**

  • **smaller than disk (not limitless capacity)**

  • **contains subset of data on disk: basically portions of programs that are currently being run**

  • **much faster than disk: memory accesses don't slow down processor quite as much**

  • **Problem: memory is still too slow (hundreds of nanoseconds)**

  • **Solution: add more layers (caches)**

Slides from Patterson's 61C

Memory Hierarchy (3/4)

Processor

Higher

Levels in memory hierarchy

Lower

Level 1

Level 2

Level 3

. . .

Level n

Increasing Distance from Proc., Decreasing speed

Size of memory at each level

As we move to deeper levels the latency goes up and price per bit goes down.

Slides from Patterson's 61C
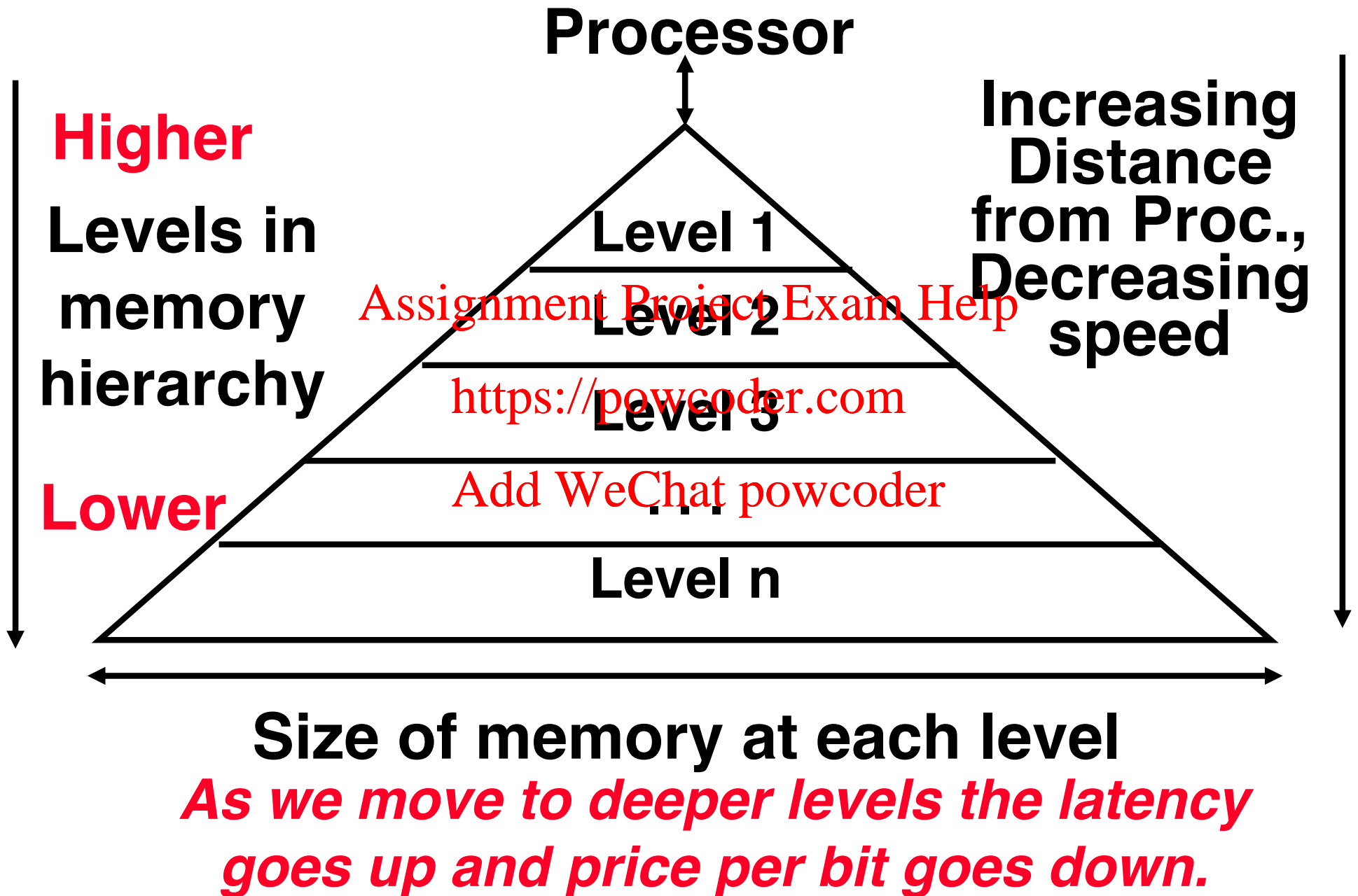
Memory Hierarchy (4/4)

° **If level is closer to Processor, it must…**

- • **Be smaller**

- • **Be faster**

- • **Contain a subset (most recently used data) of lower levels beneath it**

- • **Contain all the data in higher levels above it**

° **Lowest Level (usually disk) contains all available data**

° **Is there another level lower than disk?**

Slides from Patterson's 61C

Memory Hierarchy

Computer

Processor
(active)

Control
("brain")

Datapath
("brawn")

Memory
(passive)
(where programs, data live when running)

Devices

Input

Output

Keyboard, Mouse

Disk, Network

Display, Printer
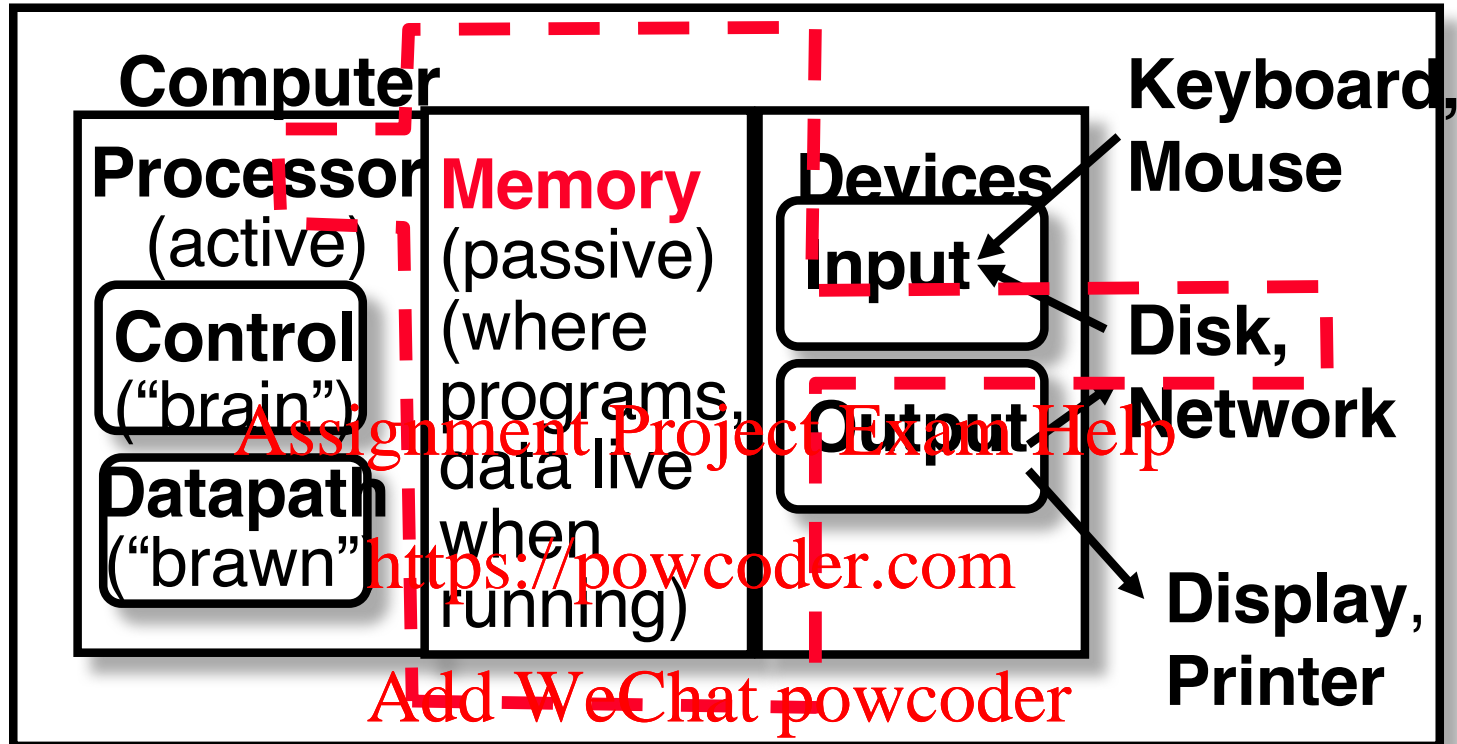
° **Purpose:**

• **Faster access to large memory from processor**

Slides from Patterson's 61C

Memory Hierarchy Analogy: Library (1/2)

° **You're writing a term paper (processor) at a table in Schulich**

° **Schulich Library is equivalent to disk**

- **essentially limitless capacity**

- **very slow to retrieve a book**

° **Table is memory**

- **smaller capacity: means you must return book when table fills up**

- **easier and faster to find a book there once you've already retrieved it**

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Slides from Patterson's 61C

Memory Hierarchy Analogy: Library (2/2)

° **Open books on table are cache**

- **smaller capacity: can have very few open books fit on table; again, when table fills up, you must close a book**

Assignment Project Exam Help

- **much, much faster to retrieve data**

https://powcoder.com

° **Illusion created: whole library open on the tabletop**

Add WeChat powcoder

- **Keep as many recently used books open on table as possible since likely to use again**

- **Also keep as many books on table as possible, since faster than going to library**

Slides from Patterson's 61C

Memory Hierarchy Basis

° **Disk contains everything.**

° **When Processor needs something, bring it into all lower levels of memory.**

° **Cache contains copies of data in memory that are being used.**

° **Memory contains copies of data on disk that are being used.**

° **Entire idea is based on <u>Temporal Locality</u>: if we use it now, we'll want to use it again soon**

Slides from Patterson's 61C

Cache Design

° **How do we organize cache?**

° **Where does each memory address map to? (Remember that cache is subset of memory, so multiple memory addresses map to the same cache location.)**

° **How do we know which elements are in cache?**

° **How do we quickly locate them?**

Slides from Patterson's 61C

Direct-Mapped Cache (1/2)

° **In a <u>direct-mapped cache</u>, each memory address is associated with one possible <u>block</u> within the cache**

  • **Therefore, we only need to look in a single location in the cache for the data if it exists in the cache**

  • **Block is the unit of transfer between cache and memory**

Direct-Mapped Cache (2/2)

Memory Address

Memory

Cache Index

4 Byte Direct Mapped Cache

0
1
2
3
4
5
6
7
8
9
A
B
C
D
E
F

0
1
2
3

° **Cache Location 0 can be occupied by data from:**

- **Memory location 0, 4, 8, ...**

- **In general: any memory location that is multiple of 4**

308-273 (13)

Slides from Patterson's 61C
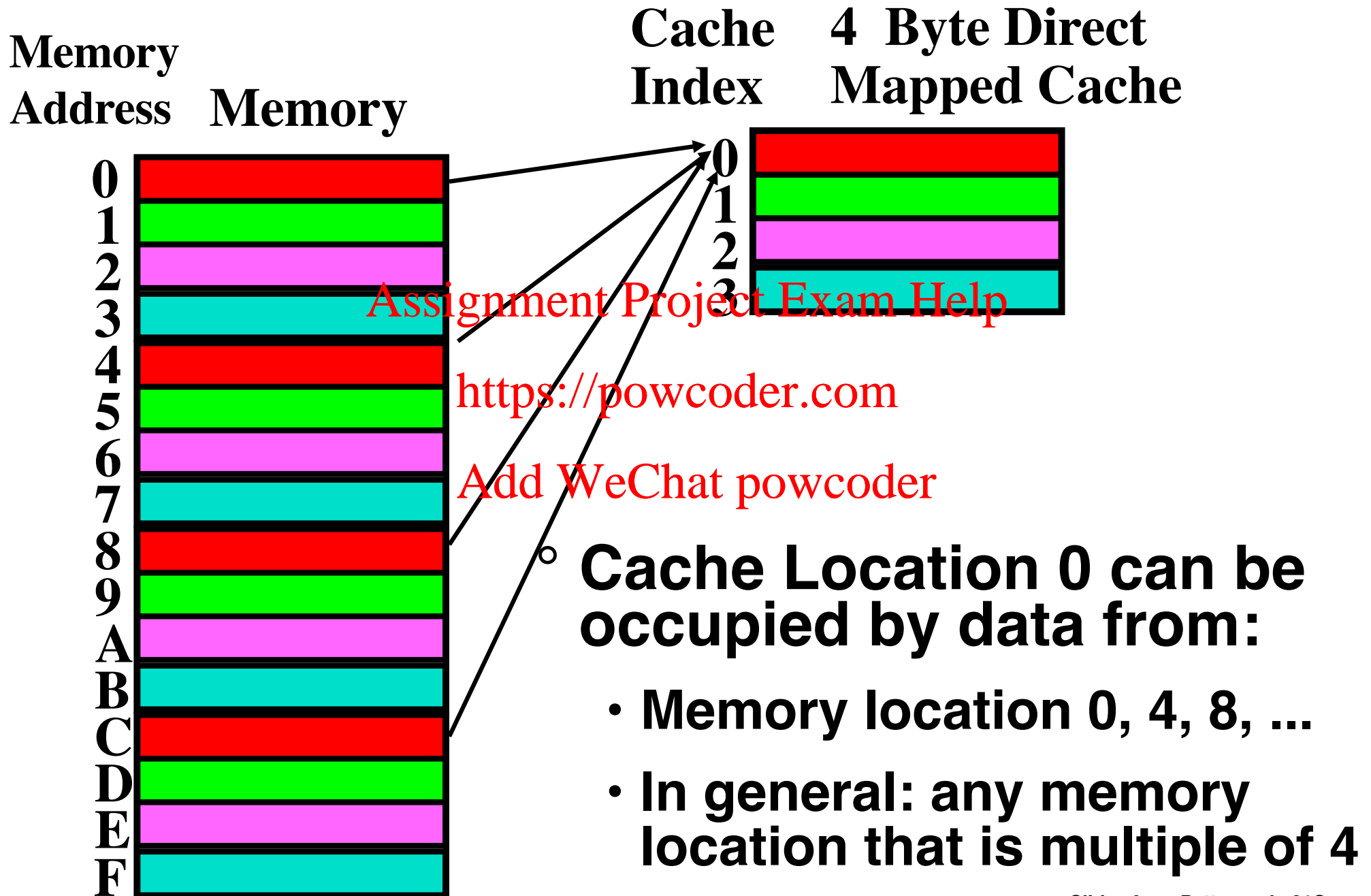
Issues with Direct-Mapped

1 Since multiple memory addresses map to same cache index, how do we tell which one is in there?

2 What if we have a block size > 1 byte?

° **Solution: divide memory address into three fields**

tttttttttttttttttttttt	iiiiiiiiii	oooo
tag **to check** **if have** **correct block**	**index** **to** **select** **block**	**offset** **byte** **within** **block**

Slides from Patterson's 61C
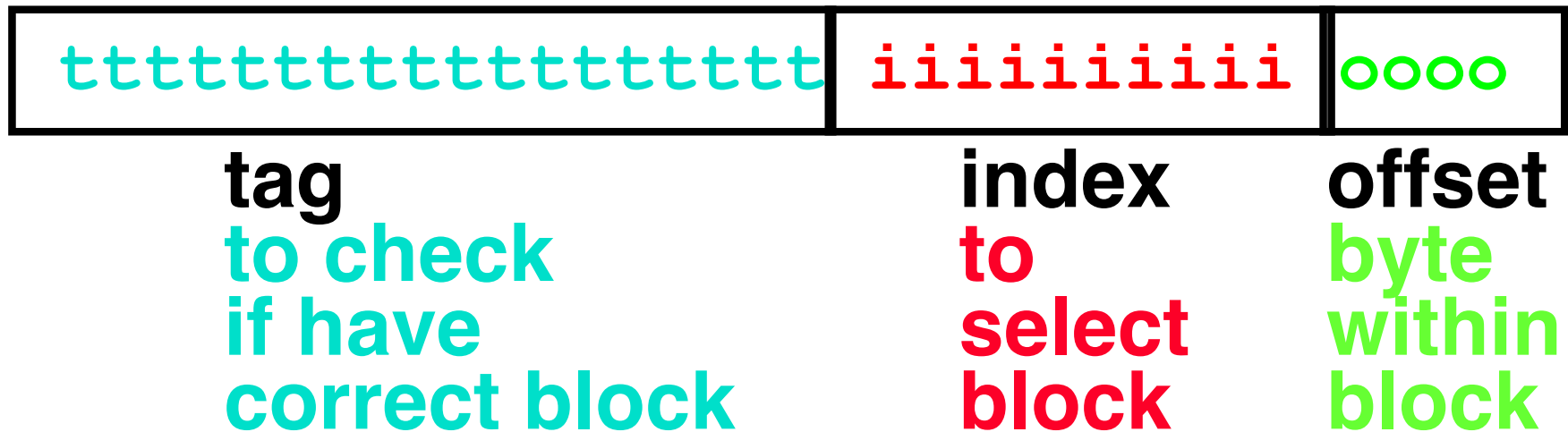
Direct-Mapped Cache Terminology

° **All fields are read as unsigned integers.**

° **Index: specifies the cache index (which "row" of the cache we should look in)**

° **Offset: once we've found correct block, specifies which byte within the block we want**

° **Tag: the remaining bits after offset and index are determined; these are used to distinguish between all the memory addresses that map to the same location**

Slides from Patterson's 61C

Direct-Mapped Cache Example (1/3)

° **Suppose we have a 16KB direct-mapped cache with 4 word blocks.**

° **Determine the size of the tag, index and offset fields if we're using a 32-bit architecture.**

° **Offset**

- **need to specify correct byte within a block**

- **block contains   4 words = 16 bytes = 2^4 bytes**

- **need 4 bits to specify correct byte**

Slides from Patterson's 61C

Direct-Mapped Cache Example (2/3)

° **Index**

- **need to specify correct row in cache**

- **cache contains 16 KB = 2^4 2^{10} = 2^{14} bytes block contains 2^4 bytes (4 words)**

- **# rows/cache = # blocks/cache (since there's one block/row)**

$$= \frac{\text{bytes/cache}}{\text{bytes/row}}$$

$$= \frac{2^{14}\text{ bytes/cache}}{2^4\text{ bytes/row}}$$

$$= 2^{10}\text{ rows/cache}$$

- **need <u>10 bits</u> to specify this many rows**

Slides from Patterson's 61C

° **Tag**

- **used remaining bits as tag**

- **tag length = mem addr length**
  - offset
  - index
  = 32 - 4 - 10 bits
  = 18 bits

- **so tag is leftmost 18 bits of memory address**

Slides from Patterson's 61C

Accessing data in a direct mapped cache

° **Example: 16KB, direct-mapped, 4 word blocks**

° **Read 4 addresses**

  • **0x00000014, 0x0000001C, 0x00000034, 0x00008014**

° **Memory values on right:**

  • **only cache/memory level of hierarchy**

Memory

Address (hex) Value of Word

Address	Value
...	...
00000010	a
00000014	b
00000018	c
0000001C	d
...	...
00000030	e
00000034	f
00000038	g
0000003C	h
...	...
00008010	i
00008014	j
00008018	k
0000801C	l
...	...

Slides from Patterson's 61C

Accessing data in a direct mapped cache

○ **4 Addresses:**

  • **0x00000014, 0x0000001C, 0x00000034, 0x00008014**

○ **4 Addresses divided (for convenience) into Tag, Index, Byte Offset fields**

```
0000000000000000  0000000001  0100

0000000000000000  0000000001  1100

0000000000000000  0000000011  0100

0000000000000010  0000000001  0100
```

Tag   **Index**   **Offset**

Slides from Patterson's 61C

Accessing data in a direct mapped cache

° **So lets go through accessing some data in this cache**

  - **16KB, direct-mapped, 4 word blocks**

° **Will see 3 types of events:**

° **cache miss: nothing in cache in appropriate block, so fetch from memory**

° **cache hit: cache block is valid and contains proper address, so read desired word**

° **cache miss, block replacement: wrong data is in cache at appropriate block, so discard it and fetch desired data from memory**

Slides from Patterson's 61C

16 KB Direct Mapped Cache, 16B blocks

° **Valid bit:** determines whether anything is stored in that row (when computer initially turned on, all entries are invalid)

Valid

Example Block

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0		Assignment Project Exam Help			
1	0					
2	0		https://powcoder.com			
3	0					
4	0		Add WeChat powcoder			
5	0					
6	0					
7	0					

... ...

| 1022 | 0 | | | | | |
| 1023 | 0 | | | | | |

Read 0x00000014 = 0…00 0..001 0100

° 00000000000000000000 0000000001 0100

               Tag field   **Index field** **Offset**

Valid

Index	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0				
1	0				
2	0				
3	0				
4	0				
5	0				
6	0				
7	0				
…		…			
1022	0				
1023	0				

Slides from Patterson's 61C

So we read block 1 (0000000001)

° 0000000000000000000 **0000000001** 0100

Tag field   **Index field**   **Offset**

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	0		Assignment Project Exam Help			
2	0					
3	0		https://powcoder.com			
4	0					
5	0		Add WeChat powcoder			
6	0					
7	0					
...			...			
1022	0					
1023	0					

Slides from Patterson's 61C

No valid data

° 00000000000000000000 **0000000001** 0100

Tag field   **Index field**   **Offset**

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	0		Assignment Project Exam Help			
2	0					
3	0		https://powcoder.com			
4	0					
5	0		Add WeChat powcoder			
6	0					
7	0					
...			...			
1022	0					
1023	0					

So load that data into cache, setting tag, valid

° 00000000000000000 0000000001 0100

Tag field   **Index field**   **Offset**

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	**1**	**0**	**a**	**b**	**c**	**d**
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...			...			
1022	0					
1023	0					

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Slides from Patterson's 61C

Read from cache at offset, return word

° 00000000000000000 0000000001 **0100**

Tag field   **Index field** **Offset**

Index	Valid Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0				
1	1 0	a	b	c	d
2	0				
3	0				
4	0				
5	0				
6	0				
7	0				
...		...			
1022	0				
1023	0				

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Read 0x0000001C = 0...00 0..001 1100

° 00000000000000000000 0000000001 <u>1100</u>

| | **Tag field** | **Index field** | **Offset** |

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					

... ...

| 1022 | 0 | | | | | |
| 1023 | 0 | | | | | |

Slides from Patterson's 61C

Data valid, tag OK, so read offset return word d

° 00000000000000000000   0000000001   1100

	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
Index						
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...			...			
1022	0					
1023	0					

Read 0x00000034 = 0…00 0..011 0100

° 00000000000000000000 **0000000011** 0100

Tag field   **Index field**   **Offset**

Valid

Index	Tag		0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					

…   …

| 1022 | 0 | | | | | |
| 1023 | 0 | | | | | |

Slides from Patterson's 61C

So read block 3

° 00000000000000000000 0000000011 0100

Tag field   **Index field**   **Offset**

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...			...			
1022	0					
1023	0					

No valid data

° 00000000000000000000 0000000011 0100

		Tag field		**Index field**	**Offset**
	Valid				
Index		**Tag**	**0x0-3**	**0x4-7**	**0x8-b**	**0xc-f**

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...			...			
1022	0					
1023	0					

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Slides from Patterson's 61C

Load that cache block, return word

° 00000000000000000000 0000000011 0100

Tag field   **Index field**   **Offset**

Valid

Index		Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
...			...			
1022	0					
1023	0					

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Read 0x00008014 = 0…10 0..001 0100

° 00000000000000000010   **0000000001**   0100

| | | **Tag field** | | **Index field** | **Offset** |

Valid

Index		Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	**0**	a	b	c	d
2	0					
3	1	**0**	e	f	g	h
4	0					
5	0					
6	0					
7	0					

… …

| 1022 | 0 | | | | | |
| 1023 | 0 | | | | | |

So read Cache Block 1, Data is Valid

° 00000000000000010 0000000001 0100

Tag field   **Index field**   **Offset**

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
...			...			
1022	0					
1023	0					

Slides from Patterson's 61C

Cache Block 1 Tag does not match (0 != 2)

° 00000000000000010 0000000001 0100

Tag field   **Index field**   **Offset**

Index	Valid	Tag	`0x0-3`	`0x4-7`	`0x8-b`	`0xc-f`
0	0					
1	1	0	a	b	c	d
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
...			...			
1022	0					
1023	0					

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Slides from Patterson's 61C

Miss, so replace block 1 with new data & tag

° 00000000000000010 0000000001 0100

	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
Index			**Tag field**		**Index field**	**Offset**
0	0					
1	1	**2**	i	j	**k**	**l**
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					

...   ...

| 1022 | 0 | | | | | |
| 1023 | 0 | | | | | |

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Slides from Patterson's 61C

And return word

° 0000000000000000010 0000000001 0100

Tag field **index field** **Offset**

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	2	i	j	k	l
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
...			...			
1022	0					
1023	0					

Do an example yourself. What happens?

° **Chose from: Cache:Hit, Miss, Miss w. replace**
Values returned: a ,b, c, d, e, ..., k, l

° **Read address 0x00000030 ?**
00000000000000000 0000000011 0000

° **Read address 0x0000001c ?**
00000000000000000 0000000001 1100

Cache

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	2	i	j	k	l
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					

...   ...

Slides from Patterson's 61C

Answers

° **0x00000030 a <u>hit</u>**

   Index = 3, Tag matches, Offset = 0, value = e

° **0x0000001c a miss**

   Index = 1, Tag mismatch, so replace from memory, Offset = 0xc, value = d

° **Therefore, returned values are:**

   • **0x00000030 = e**

   • **0x0000001c = d**

Memory

Address	Value of Word
...	...
00000010	a
00000014	b
00000018	c
0000001c	d
...	...
00000030	e
00000034	f
00000038	g
0000003c	h
...	...
00008010	i
00008014	j
00008018	k
0000801c	l
...	

Slides from Patterson's 61C

"And in Conclusion…"

° **We would like to have the capacity of disk at the speed of the processor: unfortunately this is not feasible.**

° **So we create a memory hierarchy:**

- **each successively higher level contains "most used" data from next lower level**

- **exploits temporal locality and spatial locality**

- **do the common case fast, worry less about the exceptions (design principle of MIPS)**

° **Locality of reference is a Big Idea**

Slides from Patterson's 61C