EEL 5764 Computer Architecture

Sandip Ray

Department of Electrical and Computer Engineering

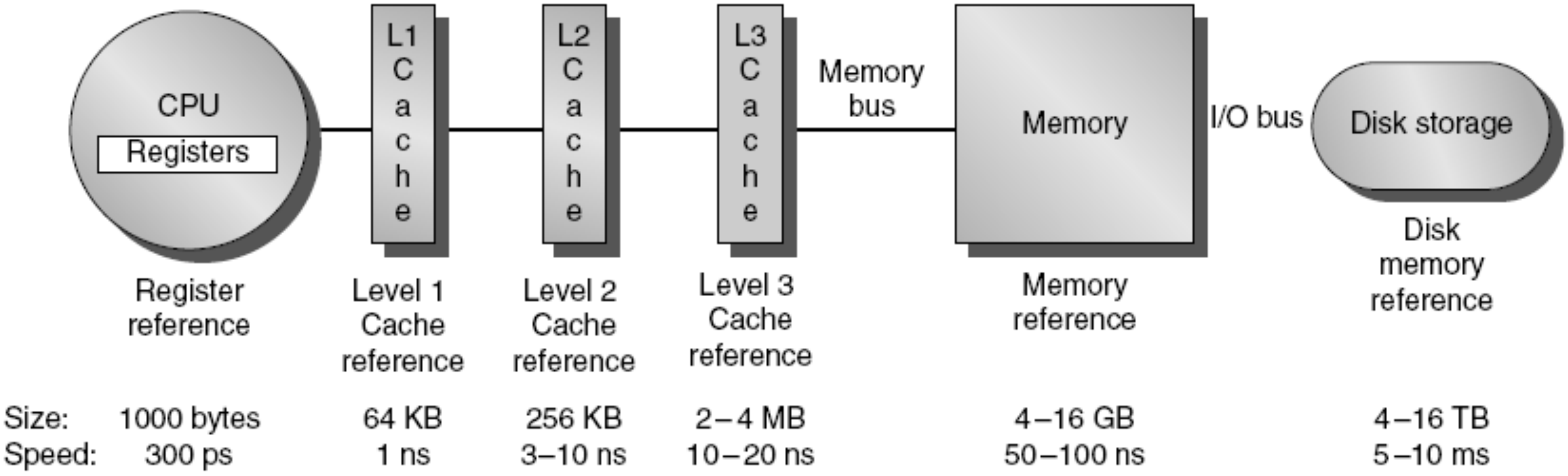
University of Florida

Lecture 6: Memory Hierarchy and Cache

Announcements

- **The time of the two in-class exams will be changed (sorry)**
  - →Same day, time in the evening (likely 8:30-10:00pm)
  - →Please keep yourselves available for the entire evening those days to allow for the exam
  - →Please let me know (in writing, with documentation) by **midnight Friday September 6** if you can't take the exams on the requisite day.

Memory Hierarchy



CPU

Registers

L1 Cache

L2 Cache

L3 Cache

Memory bus

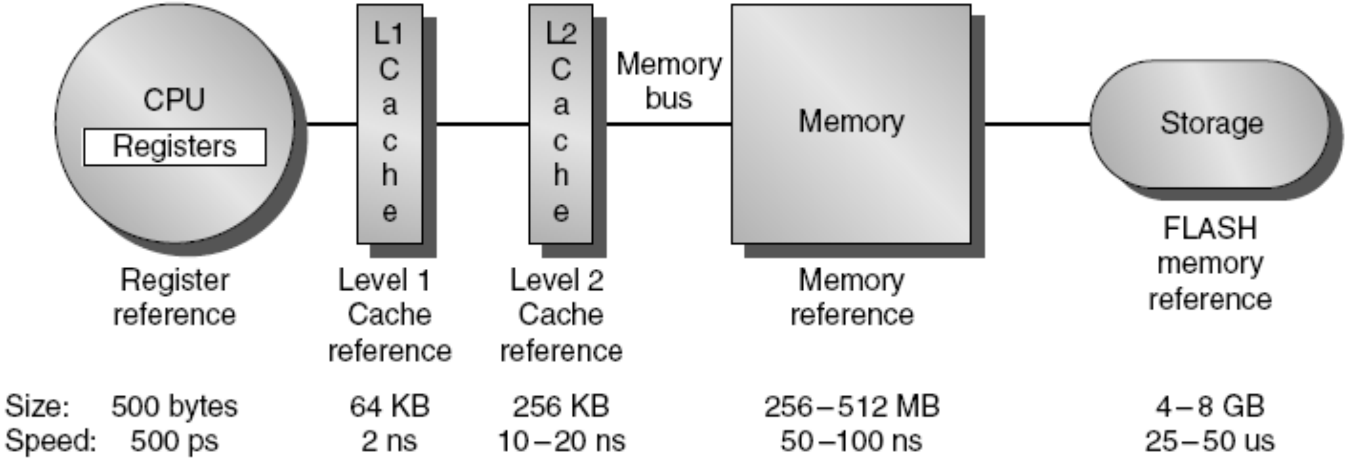
Memory

I/O bus

Disk storage

	Register reference	Level 1 Cache reference	Level 2 Cache reference	Level 3 Cache reference	Memory reference	Disk memory reference
Size:	1000 bytes	64 KB	256 KB	2–4 MB	4–16 GB	4–16 TB
Speed:	300 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	5–10 ms

(a) Memory hierarchy for server



CPU

Registers

L1 Cache

L2 Cache

Memory bus

Memory

Storage

	Register reference	Level 1 Cache reference	Level 2 Cache reference	Memory reference	FLASH memory reference
Size:	500 bytes	64 KB	256 KB	256–512 MB	4–8 GB
Speed:	500 ps	2 ns	10–20 ns	50–100 ns	25–50 us

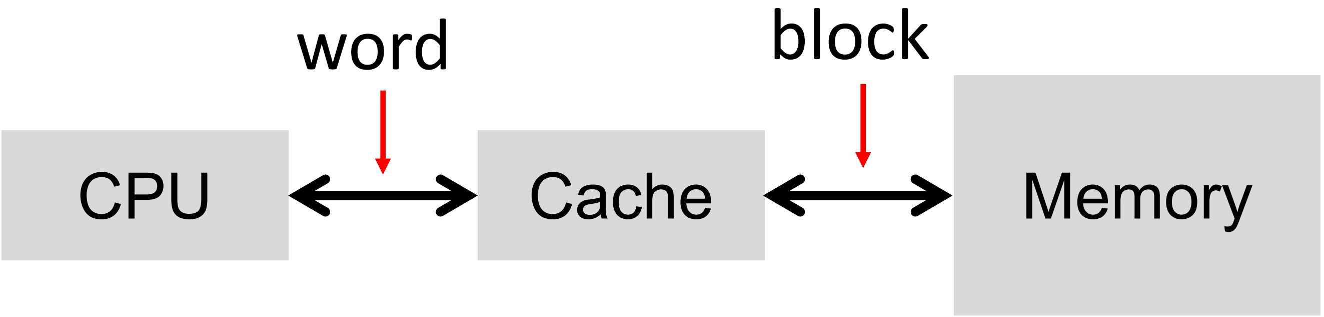
(b) Memory hierarchy for a personal mobile device

Principle of Locality – Review

- Programs often access a small proportion of their address space at any time
- Temporal locality
  - →Items accessed recently are likely to be accessed again soon
  - →e.g., instructions in a loop, induction variables
- Spatial locality
  - →Items near those accessed recently are likely to be accessed soon
  - →E.g., sequential instruction access, array data

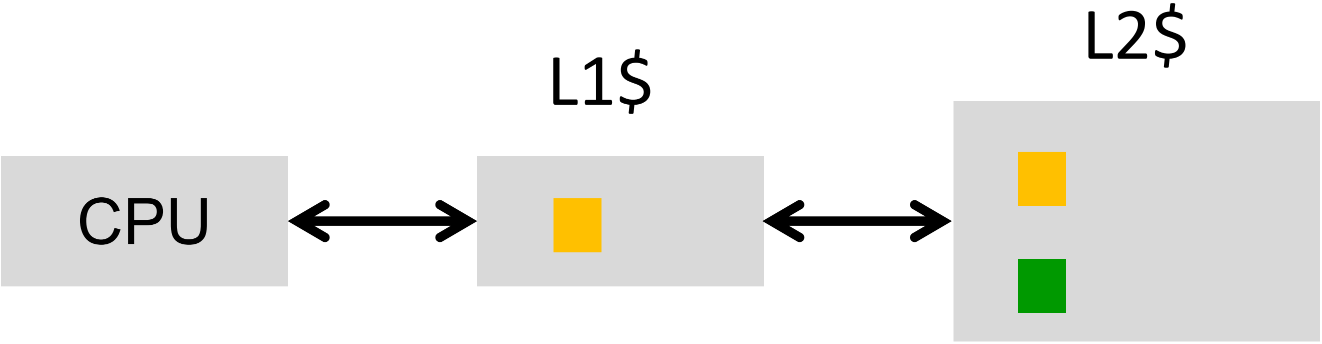
Cache Basics

- When a word is found in cache -> cache hit.
- When a word is not found in the cache, a *miss* occurs:
  - → Fetch word from lower level in hierarchy, requiring a higher latency reference
  - → Lower level may be another cache or the main memory
  - → Also fetch the other words contained within the **block**
    - ➤ Takes advantage of spatial locality
  - → Place block into cache in any location within its ***set***, determined by address

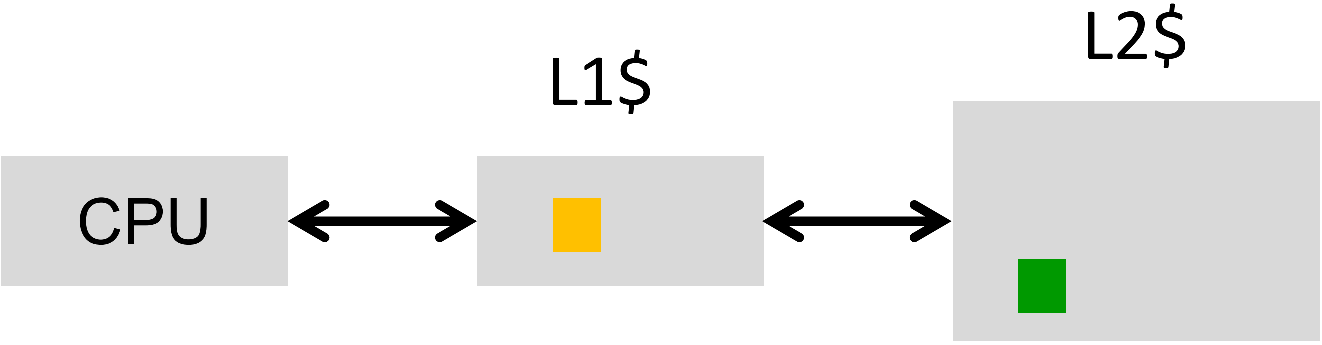


Cache Basics

- Inclusive cache



- Exclusive cache

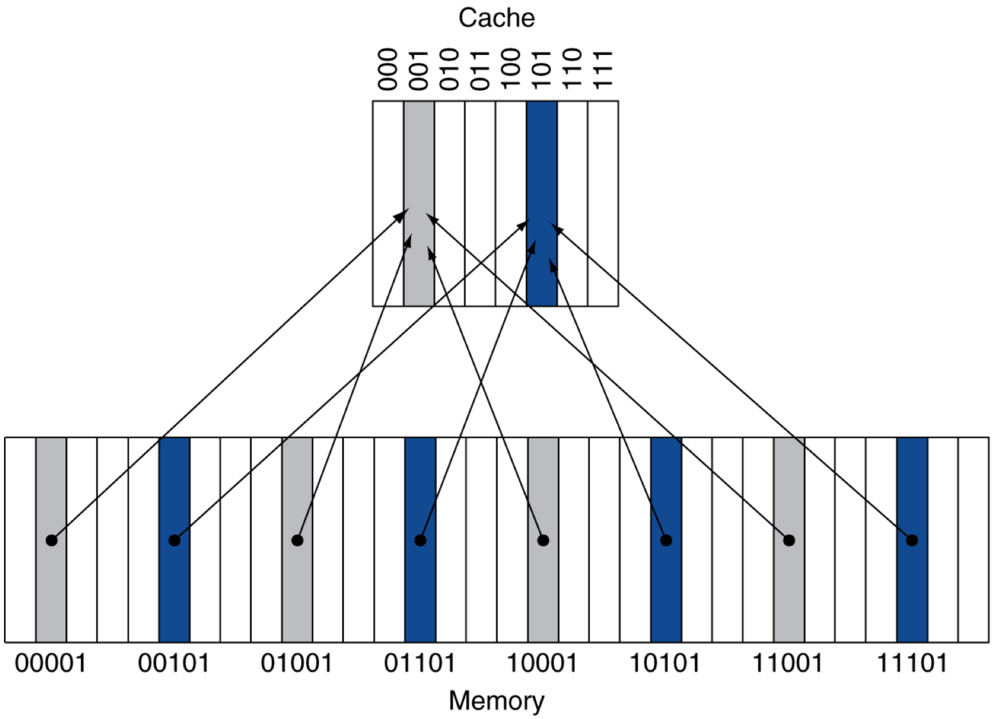


Memory Hierarchy Questions

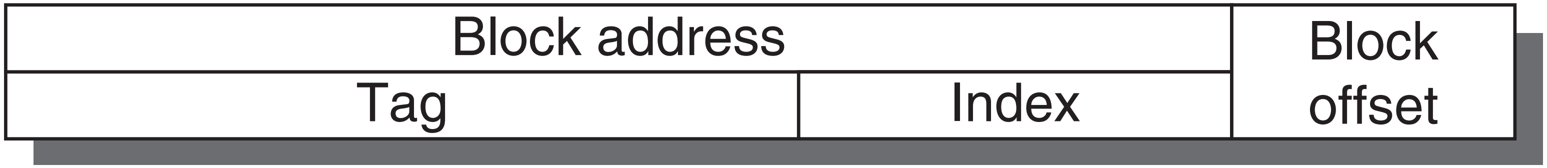
- **Block placement** - where Can a block be placed in the upper level?
- **Block identification** – how to find a block in the upper level?
- **Block replacement** - which block should be replaced on a miss?
- **Write strategy** – how to handle writes?

Direct Mapped Cache

- Only one choice **(Block address) MOD (#Blocks in cache)**



- *#Blocks is a power of 2*
- *Use low-order address bits*



Block address

Tag

Index

Block offset

Tags and Valid Bits

- Multiple memory blocks mapped to a single cache location.
- How do we know which particular memory block is stored in a cache location?
  - →Store **tag** as well as the data
  - →Tag is the high-order bits of a block address
- What if there is no data in a location?
  - →**Valid** bit: 1 = present, 0 = not present
  - →Initially 0

Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

Index	V	Tag	Data
000	0		
001	0		
010	0		
011	0		
100	0		
101	0		
110	0		
111	0		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	0		
001	0		
010	0		
011	0		
100	0		
101	0		
110	**1**	**10**	**Mem[10110]**
111	0		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	0		
001	0		
010	**1**	**11**	**Mem[11010]**
011	0		
100	0		
101	0		
110	1	10	Mem[10110]
111	0		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Index	V	Tag	Data
000	**1**	**10**	**Mem[10000]**
001	0		
010	1	11	Mem[11010]
011	**1**	**00**	**Mem[00011]**
100	0		
101	0		
110	1	10	Mem[10110]
111	0		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	1	10	Mem[10000]
001	0		
010	**1**	**10**	**Mem[10010]**
011	1	00	Mem[00011]
100	0		
101	0		
110	1	10	Mem[10110]
111	0		

Cache Misses

- On cache *hit*, CPU proceeds normally
- On cache *miss*
  - → Stall the CPU
  - → Fetch a block from next level of mem. hierarchy
  - → Instruction cache miss
    - ➤ Restart instruction fetch
  - → Data cache miss
    - ➤ Complete data access

- Note that speculative and multithreaded processors may execute other instructions during a miss
  - → Reduces performance impact of misses