

Twists on caching

2/5/25

Administrivia

- HW 5 (linked lists in C) due tomorrow night
- Midterm out Friday morning
 - Multi-day takehome due sometime next week
 - No class on Monday (2/10)
 - Covers everything up to (and including) caching

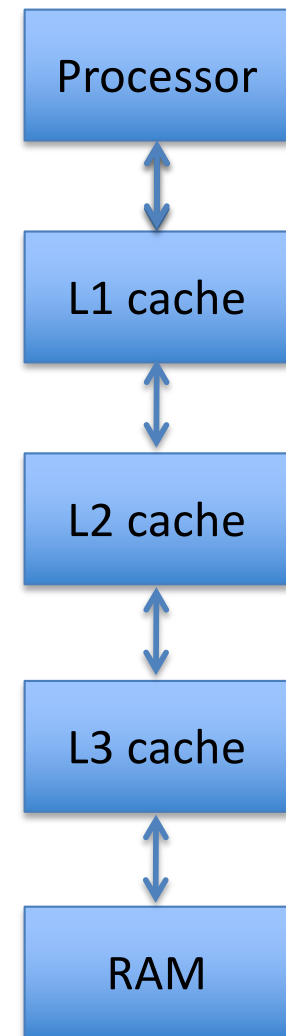
Recall: Idea of caching

- Have small, fast memory near the processor and bigger, slower memory far way
- Locality:
 - temporal: Likely to reuse memory addresses soon
 - spatial: Likely to use memory addresses near those we use now



line in which this
address can be stored

where addr falls
in cache line



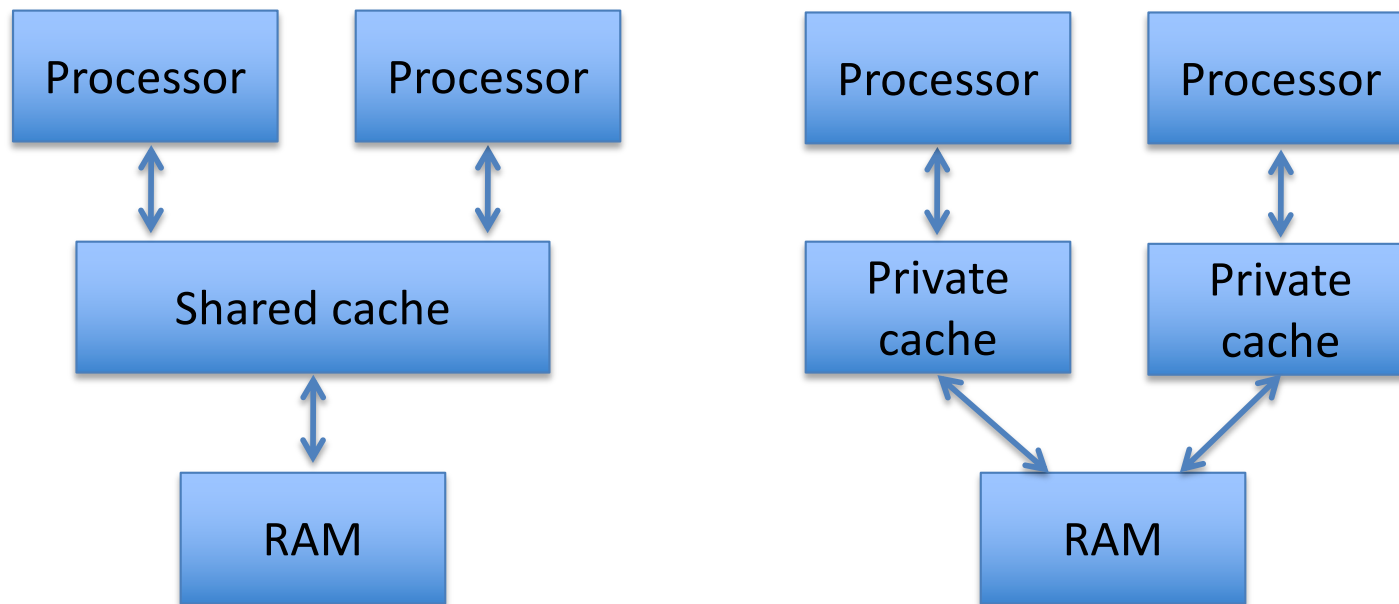
Caching: So cool it's not just for the processor

- Web browser caches pages that you've viewed
- Name servers cache translations they've recently done between names and IP addresses (e.g. cs.knox.edu becomes 72.26.72.37)

Multiprocessor caching

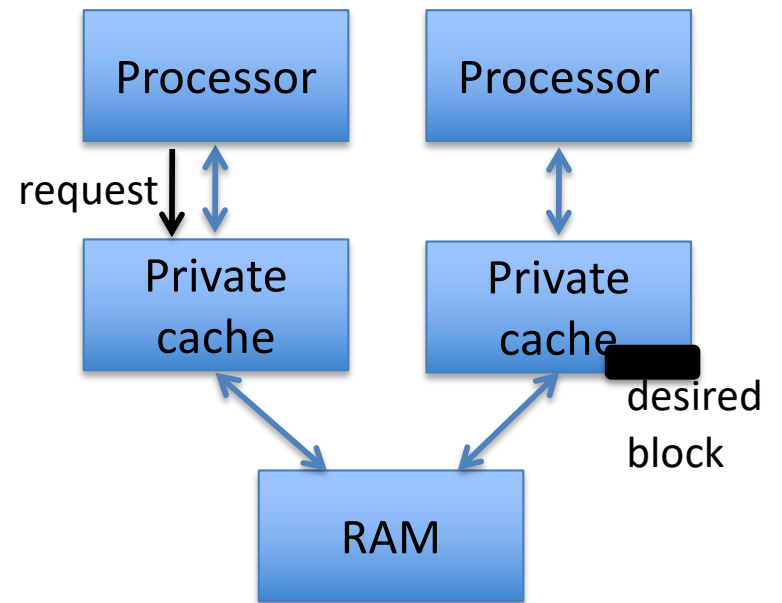
Multiprocessor caching

- How do you arrange caching for multiple processors/cores in the same address space?



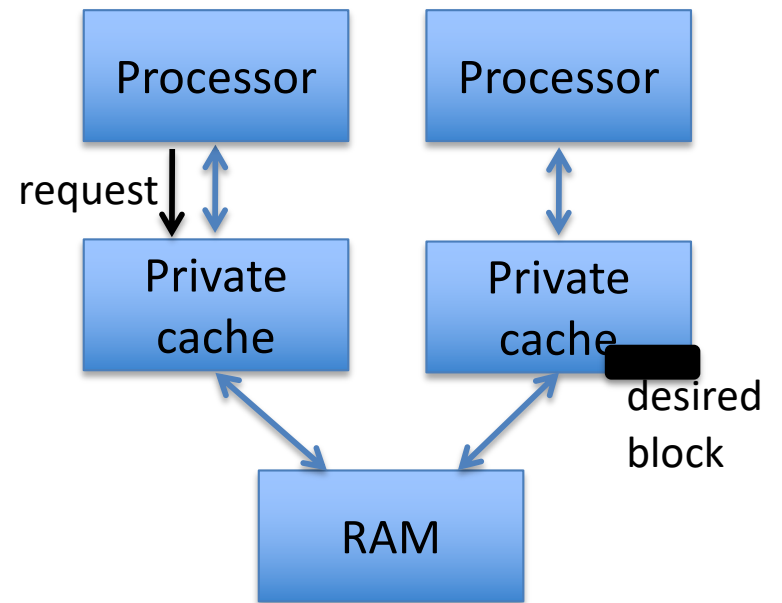
Cache coherence

- Caches should present unified view of memory
- Potentially an issue when a data block is in one cache and other PE requests it



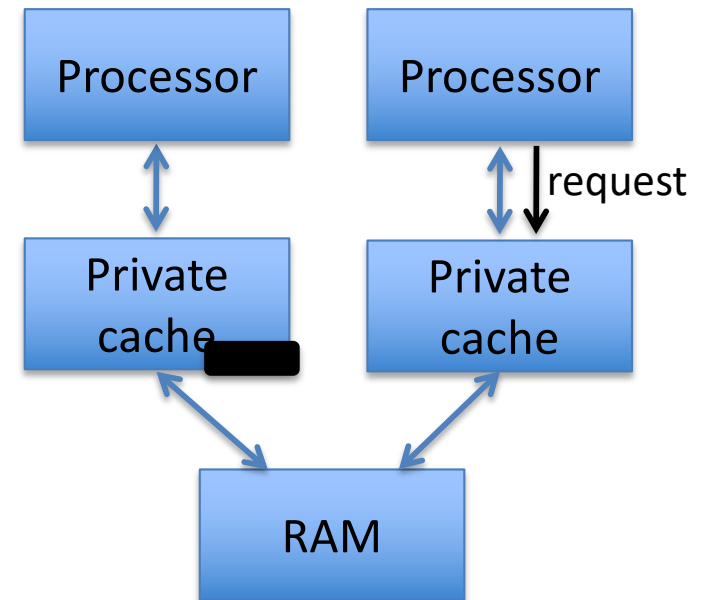
Cache coherence

- Caches should present unified view of memory
- Potentially an issue when a data block is in one cache and other PE requests it
- Left cache must “steal” the requested block from the right cache



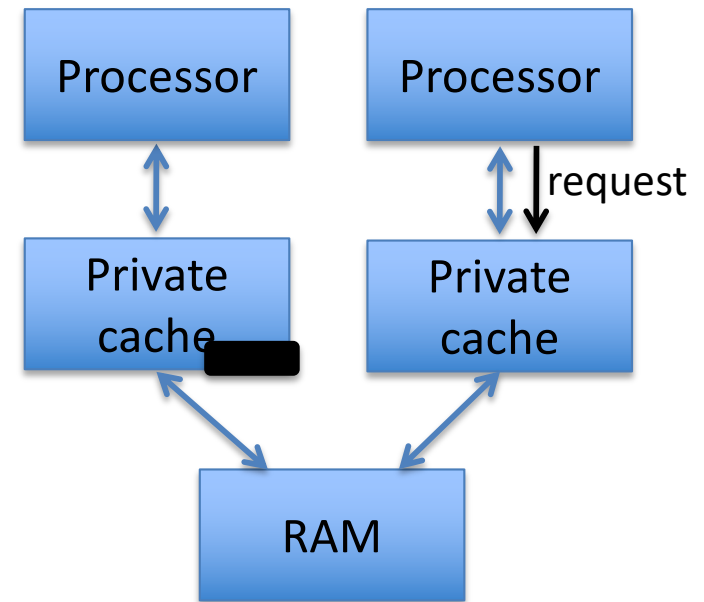
Worst case

- Data block keeps getting stolen back and forth between the private caches



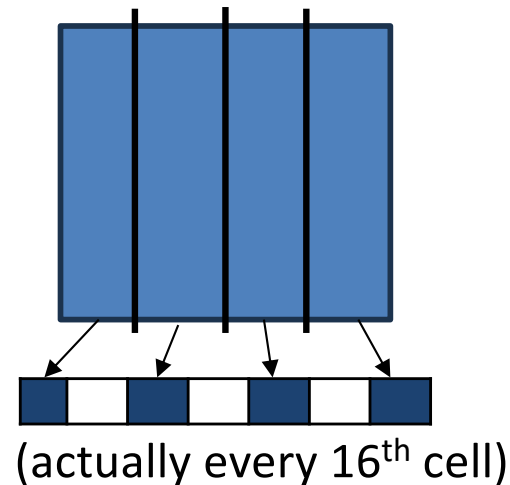
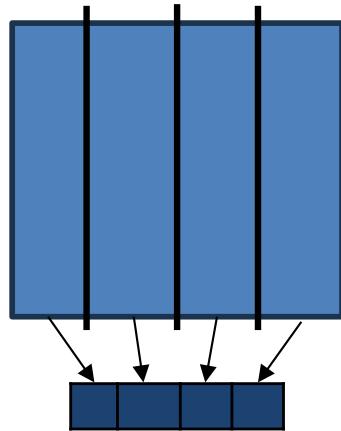
Worst case

- Data block keeps getting stolen back and forth between the private caches
- Even worse: There isn't any data being shared (called *false sharing*)



Example

- Take an image, split into regions, and count black pixels in each region



- With 16 threads, version on right took about half the time

Are you awake?

- A. Yes
- B. Yes, but I didn't come to class today
- C. Now I am
- D. Somewhat
- E. No

Virtual memory

Awkward facts

- Modern processors use 64-bit addresses
⇒ can access 2^{64} addresses

Awkward facts

- Modern processors use 64-bit addresses

⇒ can access 2^{64} addresses

$$2^{10} = 1024 \approx 1000$$

$$2^{64} = 2^4(2^{10})^6 \approx 16(1000)^6$$

$$\approx 16 \cdot 1,000,000,000,000,000,000$$

aka 16 exabytes

Awkward facts

- Modern processors use 64-bit addresses

⇒ can access 2^{64} addresses

$$2^{10} = 1024 \approx 1000$$

$$2^{64} = 2^4(2^{10})^6 \approx 16(1000)^6$$

$$\approx 16 \cdot 1,000,000,000,000,000,000$$

aka 16 exabytes

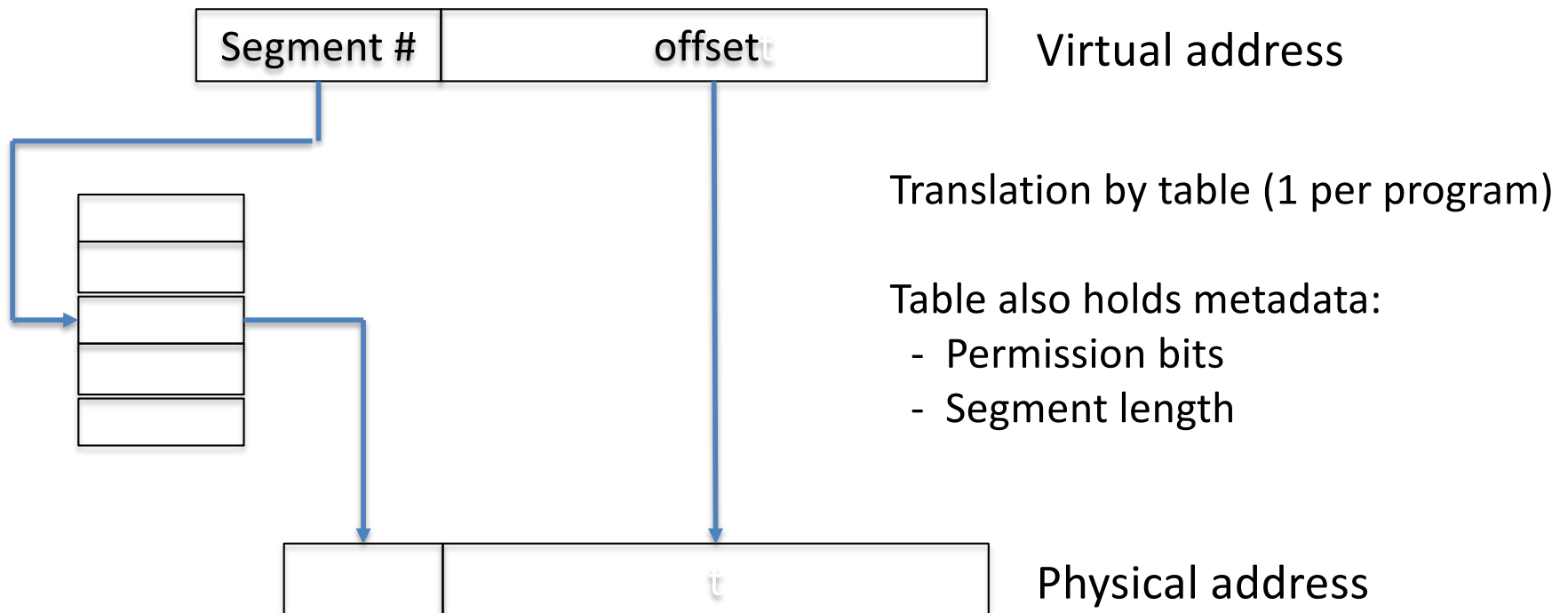
- Every program gets its own address space

Virtual memory

- Programs use virtual addresses that map to physical addresses
- Give each program its own address space
 - Simplifies programming:
 - Programs don't have to manage memory
 - Simplifies multitasking
 - Programs use any addresses they want
 - Isolates programs from each other and the OS

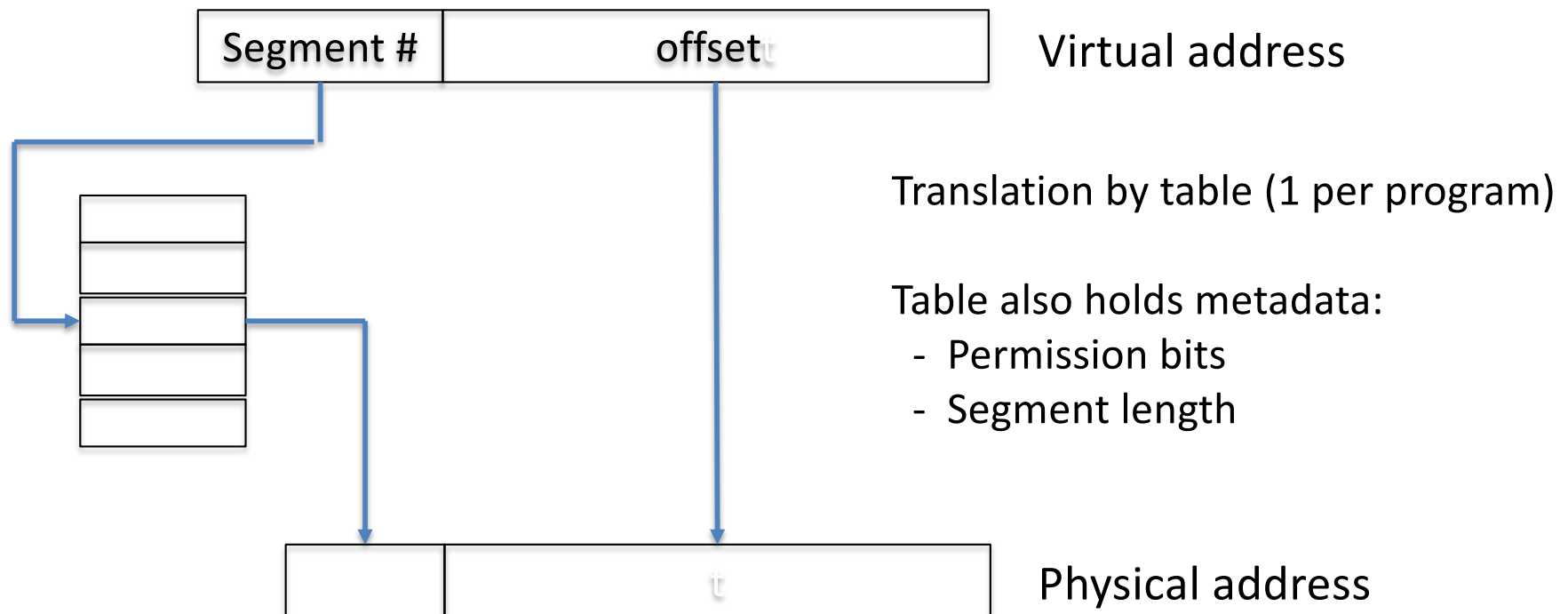
Version 1: Segmentation

- Memory divided by logical function (OS, program code, library, data, heap, ...) into segments



Version 1: Segmentation

- Memory divided by logical function (OS, program code, library, data, heap, ...) into segments



What if segment length exceeds maximum possible?

External fragmentation

- Occurs when there is enough free space for a desired segment, but it's not all together

Memory:

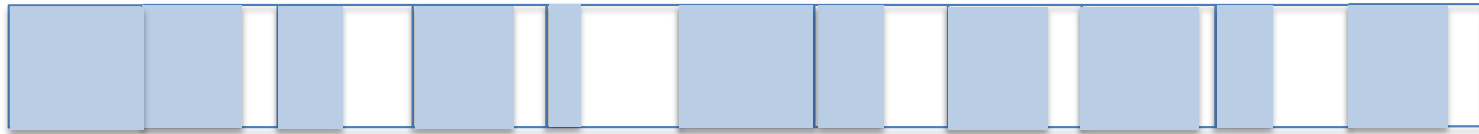


Desired segment:



Internal fragmentation

- Occurs when space inside the segments is wasted



Version 2: Paging

- Memory organized into small (4-16KB), fixed-size “pages” which are allocated as needed
- Translation by “page table”
- Accessing unmapped memory is “page fault”

Real systems use a combination

- Memory organized into small pages, each containing one type of data
- Terminology from both: “segmentation fault”, “page fault”, “page table”