Memory & Caches IV

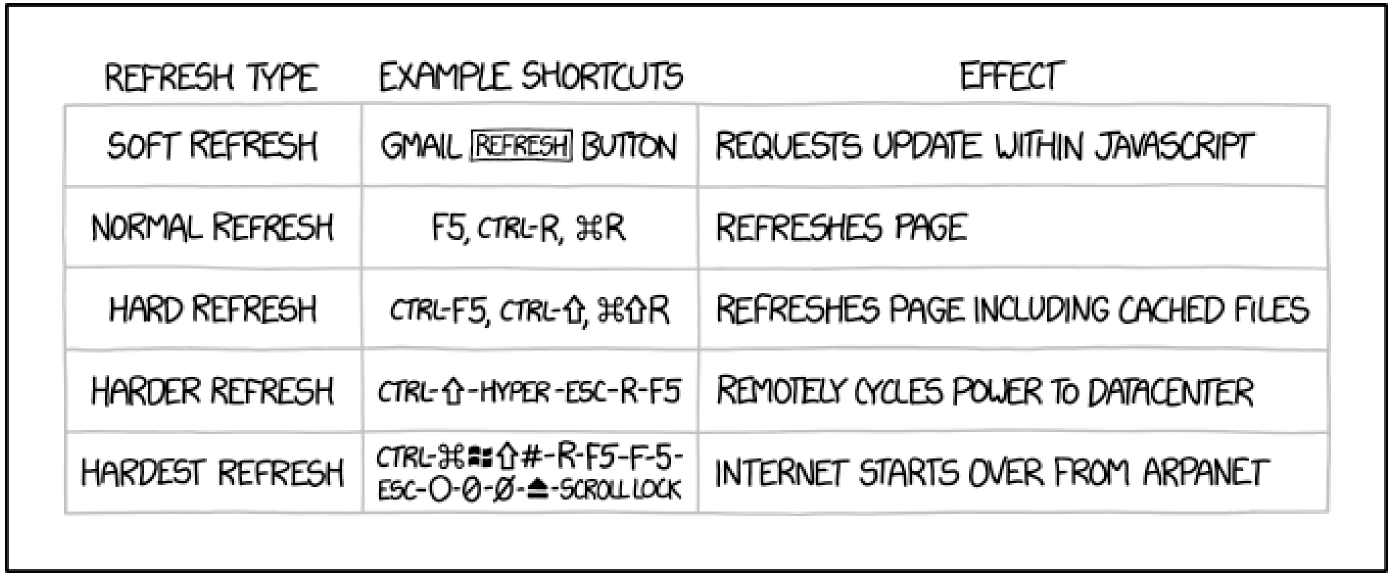
CSE 351 Winter 2024

Instructor:

Justin Hsia

Teaching Assistants:

Adithi Raghavan
Aman Mohammed
Connie Chen
Eyoel Gebre
Jiawei Huang
Malak Zaki
Naama Amiel
Nathan Khuat
Nikolas McNamee
Pedro Amarante
Will Robertson

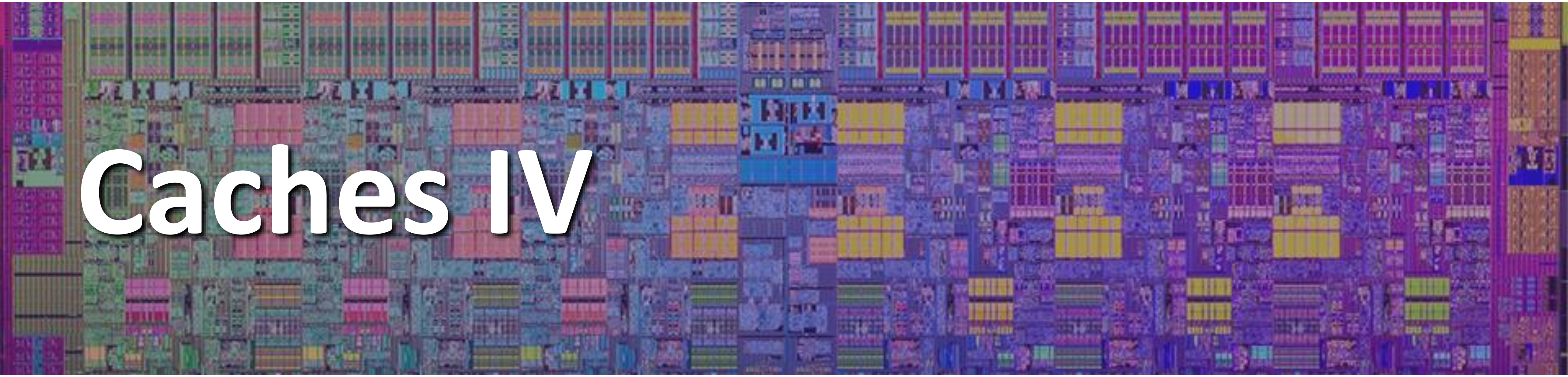


REFRESH TYPE	EXAMPLE SHORTCUTS	EFFECT
SOFT REFRESH	GMAIL [REFRESH] BUTTON	REQUESTS UPDATE WITHIN JAVASCRIPT
NORMAL REFRESH	F5, CTRL-R, ⌘R	REFRESHES PAGE
HARD REFRESH	CTRL-F5, CTRL-⇧, ⌘⇧R	REFRESHES PAGE INCLUDING CACHED FILES
HARDER REFRESH	CTRL-⇧-HYPER-ESC-R-F5	REMOTELY CYCLES POWER TO DATACENTER
HARDEST REFRESH	CTRL-⌘⊞⇧#-R-F5-F-5-ESC-O-0-Ø-⏏-SCROLL LOCK	INTERNET STARTS OVER FROM ARPANET

http://xkcd.com/1854/

Relevant Course Information

- HW16 due tonight, HW17 due Wed (2/21), HW18 due Fri (11/17)
  - HW18 is specifically Lab 4 preparation
- Lab 3 due tonight, late deadline is Monday (2/19)
- Lab 4 released today, due in two weeks (Fri, 3/1)
  - Cache parameter puzzles and code optimizations

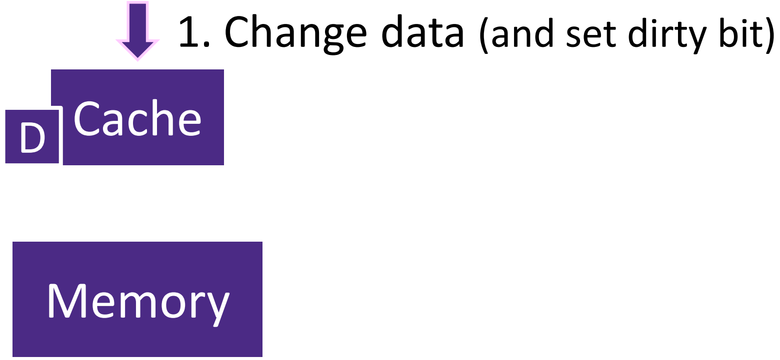


Caches IV

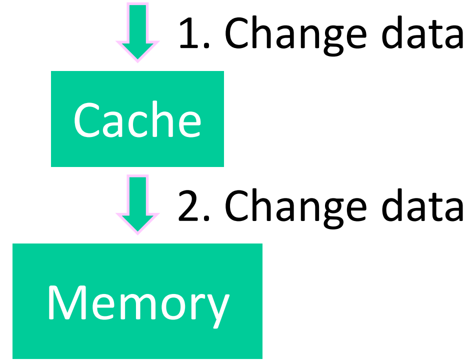
Lesson Summary (1/3)

- The 3 C's of cache misses: **compulsory**, **conflict**, and **capacity**
  - There are both parameter and code changes that can help with each kind

- Write-**hit** policies:
  - Write back + write allocate
    - Each line of cache has a *dirty bit*



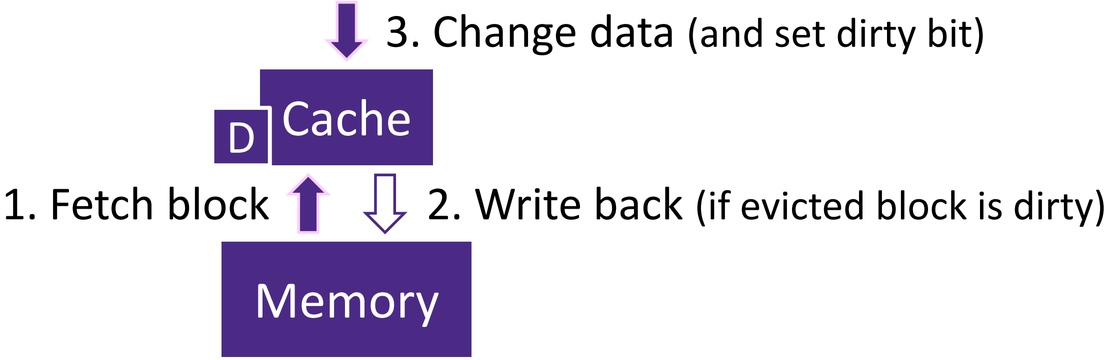
  - Write through + no write allocate



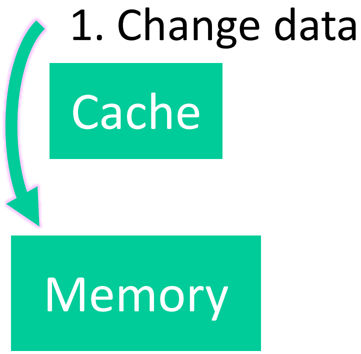
Lesson Summary (2/3)

- The 3 C's of cache misses: **compulsory**, **conflict**, and **capacity**
  - There are both parameter and code changes that can help with each kind

- Write-**miss** policies:
  - Write back + write allocate
    - Each line of cache has a *dirty bit*

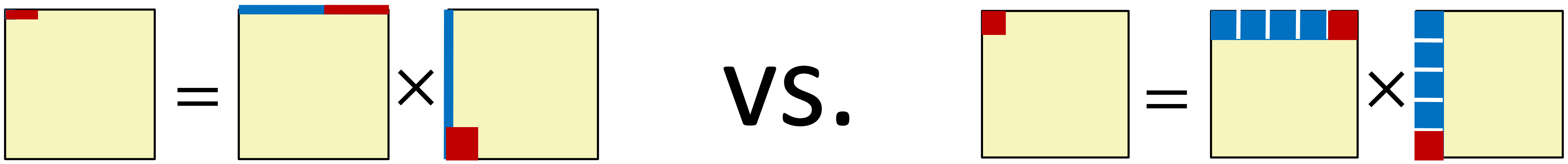


  - Write through + no write allocate



Lesson Summary (3/3)

- ***Cache blocking*** is a cache optimization technique that reorders memory accesses to maximize the use of cache blocks while they are in the cache
  - Use data in cache block as much as possible before evicting that block
  - Subdivide larger problem (*e.g.*, matrix multiplication) into smaller ones where *working set* can fit in the cache



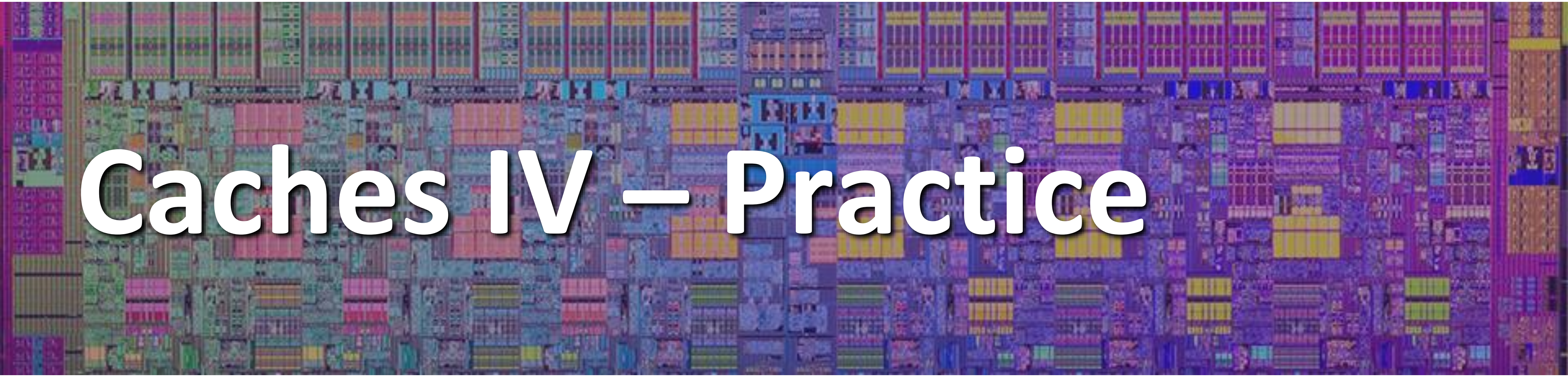
- **Cache-friendly code:**
  - Work with a reasonably small amount of data at any given time
  - Use small strides whenever possible in terms of loop and index ordering
  - Focus your time and energy on optimizing the inner loop code

Lesson Q&A

- Learning Objectives:
  - Apply techniques, such as cache blocking, to optimize cache performance.
  - Analyze how changes to cache parameters and policies affect performance metrics such as AMAT.

- What lingering questions do you have from the lesson?
  - Chat with your neighbors about the lesson for a few minutes to come up with questions

Caches IV – Practice



Polling Question

- Which of the following cache statements is FALSE?

A. We can reduce compulsory misses by decreasing our block size

B. We can reduce conflict misses by increasing associativity

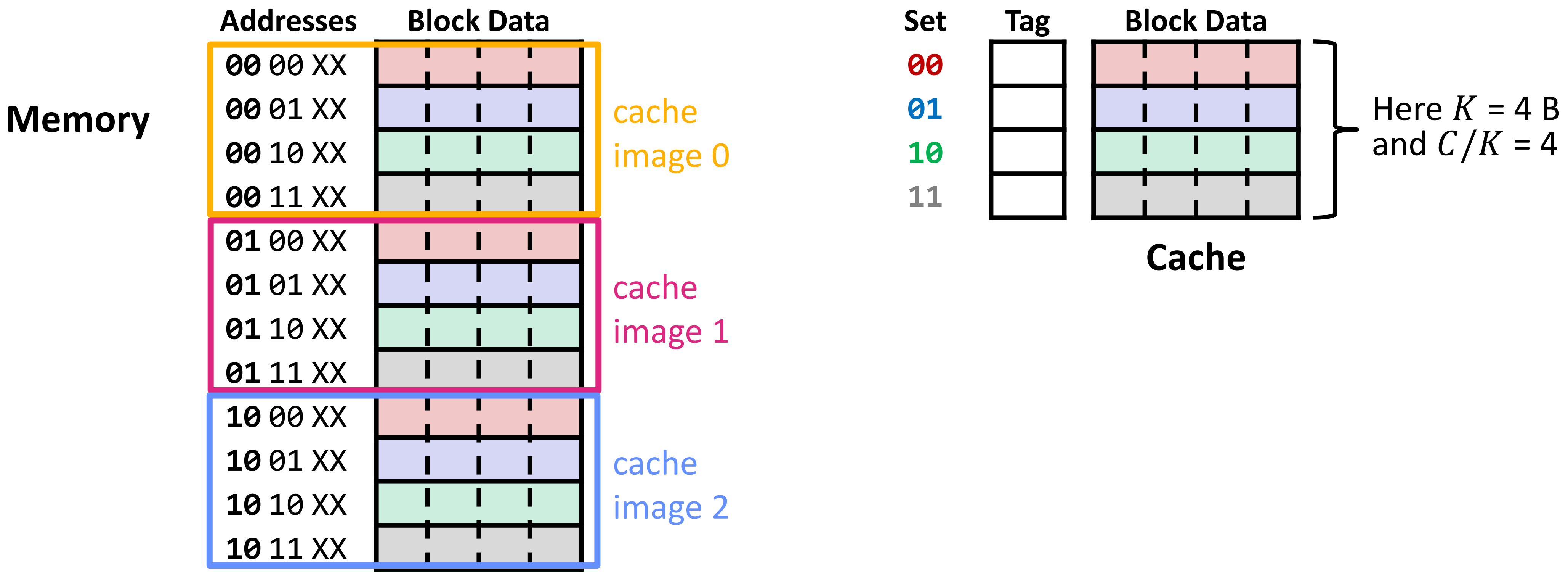
C. A write-back cache will save time for code with good temporal locality on writes

D. A write-through cache will always match data with the memory hierarchy level below it

E. We're lost…

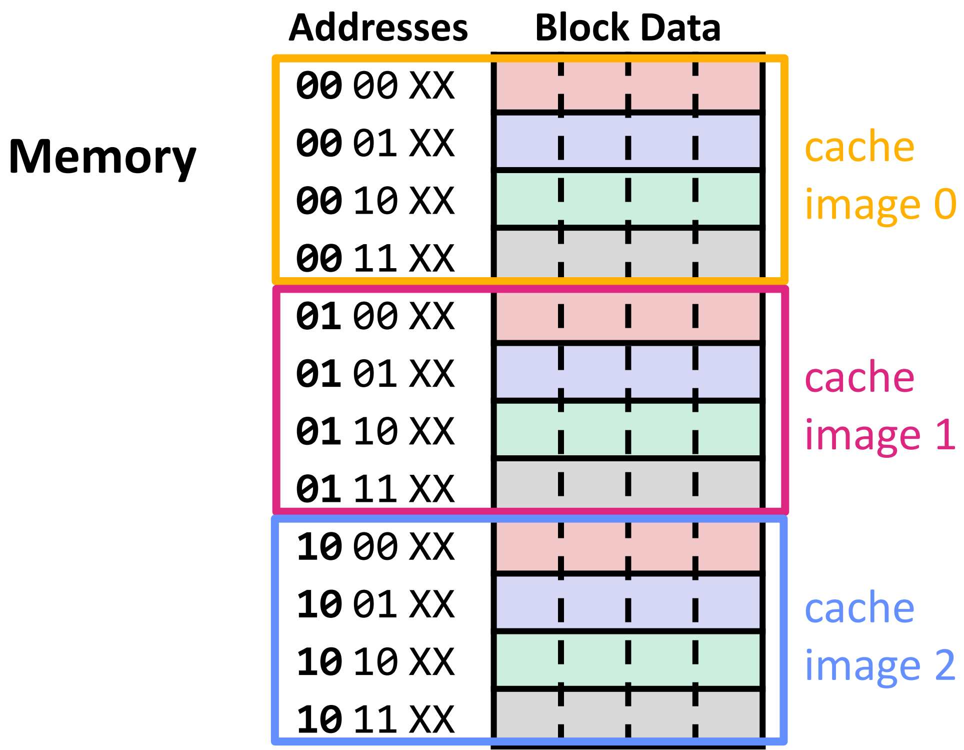
Homework Setup (1/2)

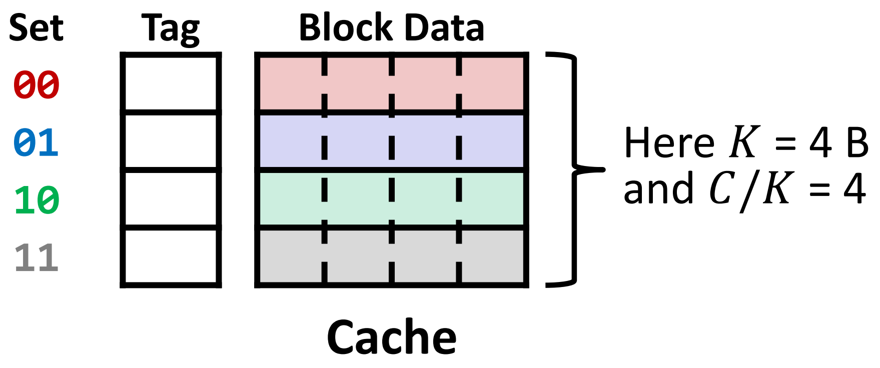
- Homework 18 explores the idea of a ***cache image*** – a view of memory chunking by cache size instead of block size
  - Each cache image maps entirely onto (*i.e.*, exactly fills) the cache
  - Each cache image has a unique *tag* (instead of block number)



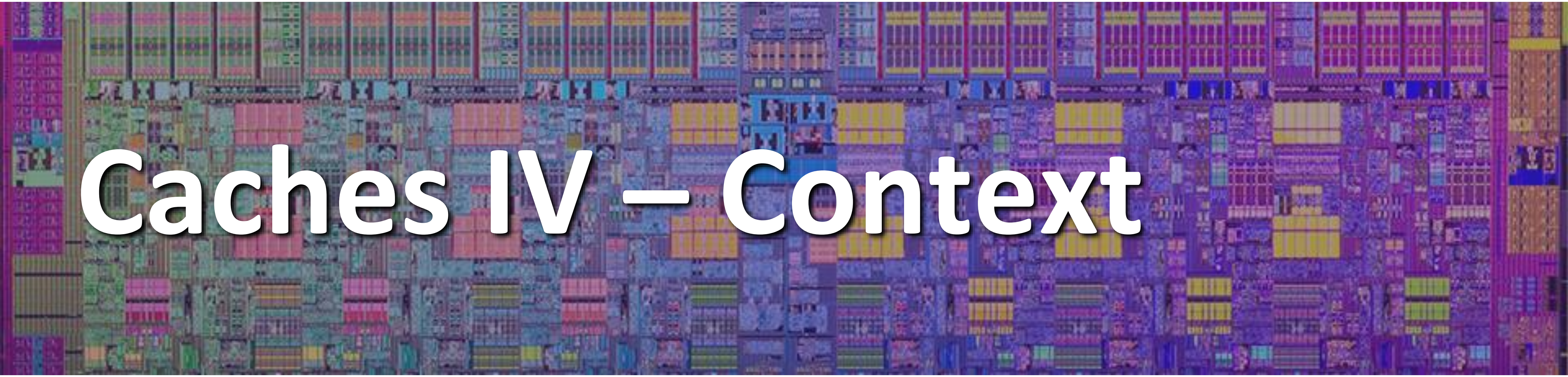
Homework Setup (2/2)

- Assume our code currently does: R 0x00, W 0x20, R 0x01, W 0x21
  - What is the current miss rate?
  - How could we rearrange these accesses to improve our miss rate?

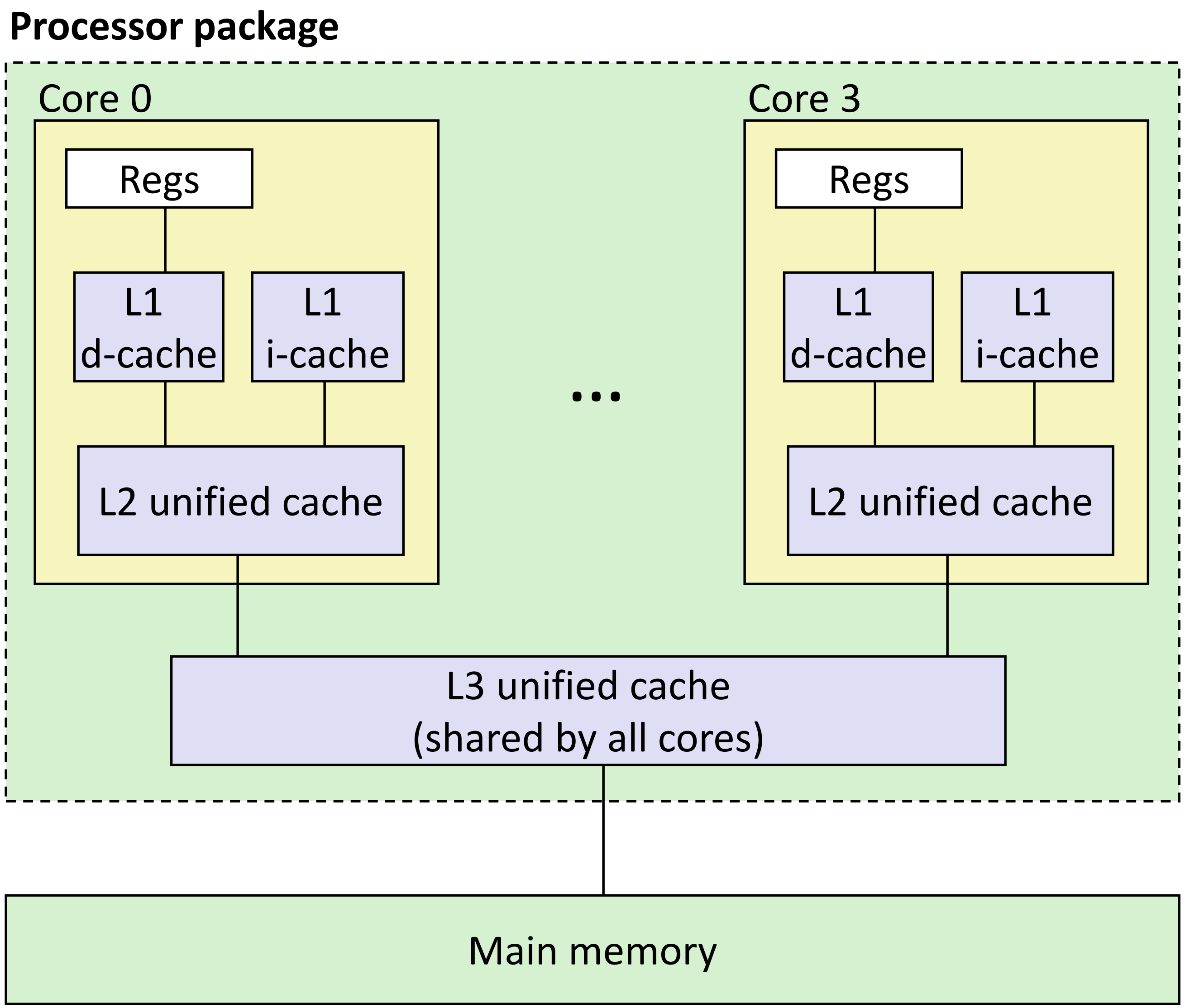




Caches IV – Context



Intel Core i7 Cache Hierarchy



Block size:
64 bytes for all caches

L1 i-cache and d-cache:
32 KiB, 8-way,
Access: 4 cycles

L2 unified cache:
256 KiB, 8-way,
Access: 11 cycles

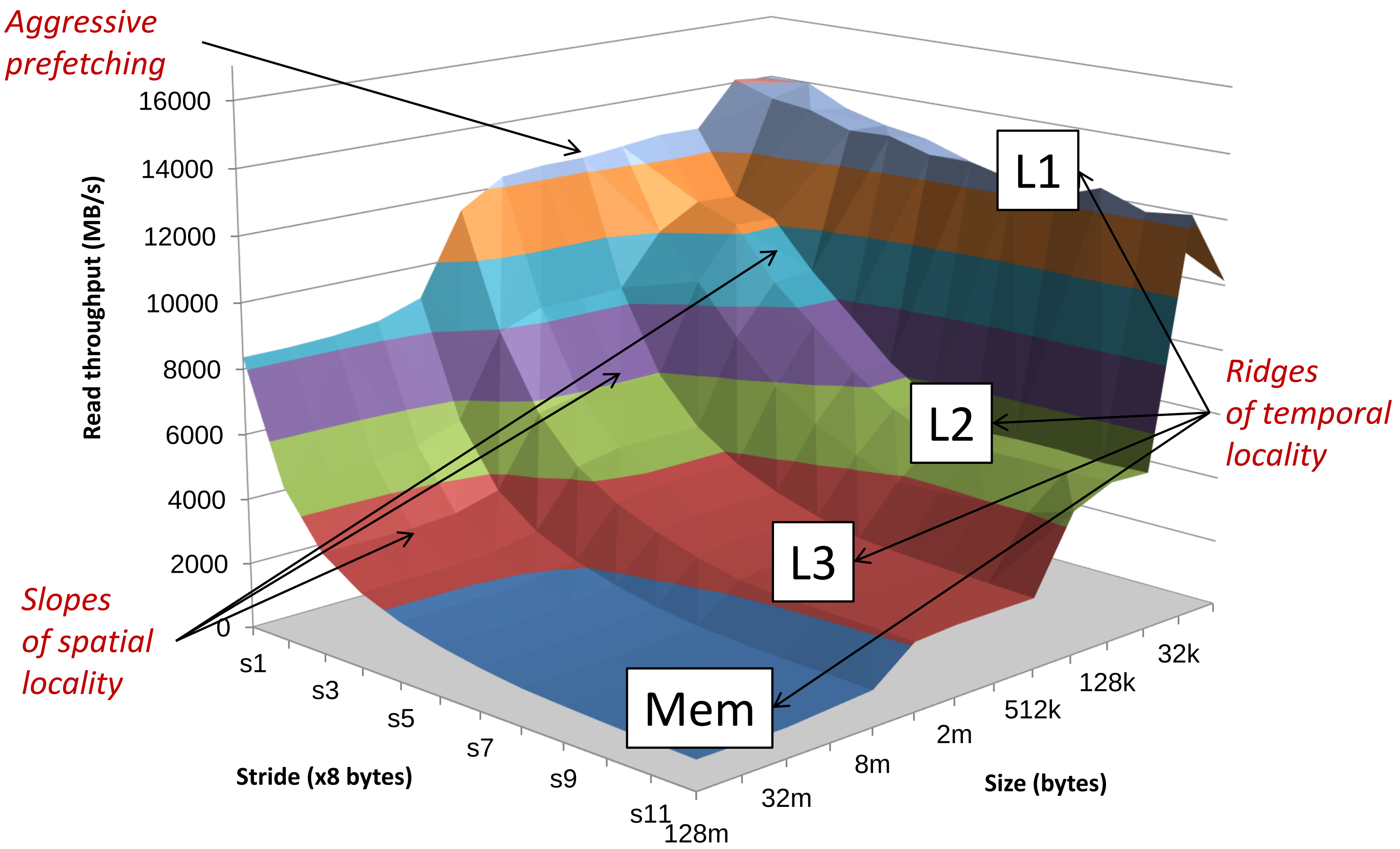
L3 unified cache:
8 MiB, 16-way,
Access: 30-40 cycles

Learning About Your Machine

- **Linux:**
  - `lscpu`
  - ls /sys/devices/system/cpu/cpu0/cache/index0/
    - Example: cat /sys/devices/system/cpu/cpu0/cache/index*/size
- **Windows:**
  - `wmic memcache get <query>` (all values in KB)
  - Example: `wmic memcache get MaxCacheSize`
- Modern processor specs: http://www.7-cpu.com/

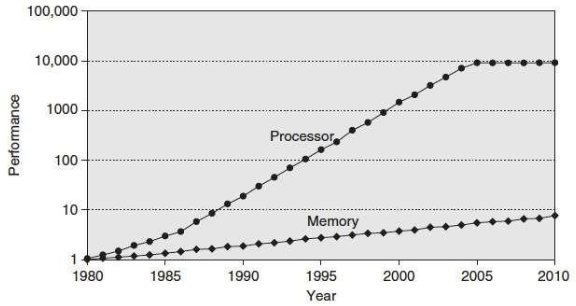
The Memory Mountain

Core i7 Haswell
2.1 GHz
32 KB L1 d-cache
256 KB L2 cache
8 MB L3 cache
64 B block size



Cache Motivation, Revisited

- Memory accesses are expensive!
  - Massive speedups to processors without similar speedups in memory only made the problem worse
  - “Processor-Memory Bottleneck”:



- We defined “locality”, based on observations about existing programs, written by an extremely small subset of the population
  - We built hardware that utilizes locality to improve performance (*e.g.*, AMAT)

Cache "Conclusions"

- All systems favor "cache-friendly code"
  - Can get most of the advantage with generic coding rules

- ⚠ We implicitly made value judgments about "good" and "bad" code
  - "Good" code exhibits "good" locality
  - "Good" code might be considered the (desired) *common case*

Common Case Optimizations

- Optimizing for the common case is a classic (arguably foundational) CS technique!
  - *e.g.*, algorithms analysis often uses worse case or average case performance
  - *e.g.*, caches optimize for an *average program* ("most programs") that exhibits locality

- Natural conclusion is to make the common case as performant as possible at the expense of edge-cases
  - Generally, bigger performance impact with common case than edge case optimizations
  - What's the danger here?

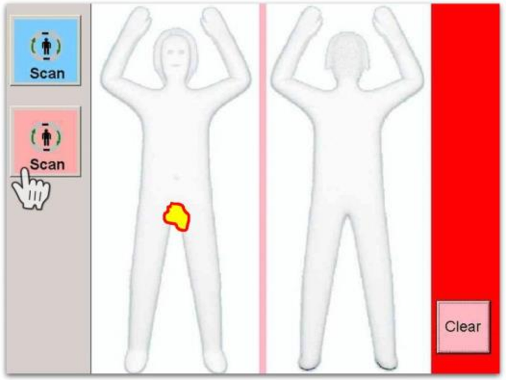
The Common Case and Normativity

- "**Normativity** is the phenomenon in human societies of designating some actions or outcomes as good or desirable or permissible and others as bad or undesirable or impermissible."
  - https://en.wikipedia.org/wiki/Normativity

- **Norms** are what are considered "usual" or "expected"
  - These often get conflated with the common case: *norm* gets "common case" treatment, *abnormal* gets "edge case" treatment
  - Who determines the norms?

Example: TSA Body Scanners

- TSA used machine learning to determine predictable variation among “average” bodies
  - Built two models: one for “men” and one for “women”
- TSA agent chooses model to use *based on how the traveler is presenting:*



- Who are the “edge cases?”
- What is the “edge case performance?”

Design Considerations

- Make sure you account for non-normative cases
  - Is this (change to) edge-case behavior okay/acceptable?

- Be careful of implicit normative assumptions
  - Can erase people's experiences and diversity, even labeling/categorizing them as threats
  - Caches aren't neutral, either – they assume that the underlying data doesn't change
    - Changes can come from above (the CPU), but not from below
    - *e.g.*, changing your name in Google Drive "breaks" the browser cache

Discussion Questions

- Discuss the following question(s) in groups of 3-4 students
  - I will call on a few groups afterwards so please be prepared to share out
  - Be respectful of others' opinions and experiences

- Where else do you see normative assumptions made in tech or CS? What are the consequences of the "edge case" behaviors in these situations?

Group Work Time

- During this time, you are encouraged to work on the following:
  1) If desired, continue your discussion
  2) Work on the homework problems
  3) Work on the lab (if applicable)

- Resources:
  - You can revisit the lesson material
  - Work together in groups and help each other out
  - Course staff will circle around to provide support