Lecture 4

Basic Caches

Fall 2016

Pejman Lotfi-Kamran

Adapted from slides originally developed by Profs. Falsafi, Hill, Hoe, Lipasti, Shen, Smith, Sohi, and Vijaykumar of Carnegie Mellon University, EPFL, Purdue University, and University of Wisconsin.

Where Are We?

Fr	Sa	Su	Mo	Tu
	27-Shahrivar		29-Shahrivar	
	3-Mehr	→	5-Mehr	
	10-Mehr		12-Mehr	
	17-Mehr		19-Mehr	
	24-Mehr		26-Mehr	
	1-Aban		3-Aban	
	8-Aban		10-Aban	
	15-Aban		17-Aban	
	22-Aban		24-Aban	
	29-Aban		1-Azar	
	6-Azar		8-Azar	
	13-Azar		15-Azar	
	20-Azar		22-Azar	
	27-Azar		29-Azar	
	4-Dey		6-Dey	

This Lecture

- Basic caches

Next Lecture:

- Low miss-ratio caches

Memory Systems

Basic caches

- introduction
- fundamental questions
- cache size, block size, associativity

today

Advanced caches

Main memory

Virtual memory

Motivation

CPU can only go as fast as memory!

- memory reference/inst x bytes-per-reference x IPC/cycle time
- In 1990: (1+0.2) x 4 x 1 / 2ns = 2.4 GB/s
- In 2000: (1x4+0.2x8) x 3 / 0.3ns = 56 GB/s

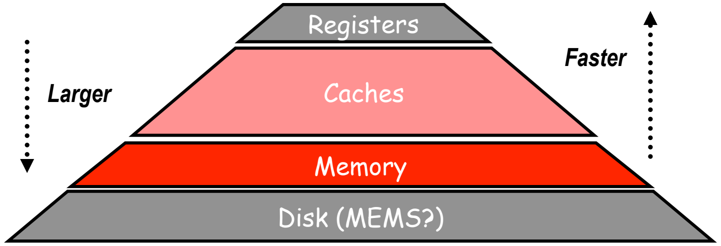
Want storage memory:

- as fast as CPU
- as large as required by all of the running applications

Memory Hierarchy

Make common case fast:

- common: temporal & spatial locality
- fast: smaller more expensive memory



Storage Hierarchies

Storages are layered by hierarchies away from the CPU in the order of

- increasing latency (t_i) $t_i < t_{i+1}$
- increasing size (s_i)
  - ⇒ decrease unit cost (c_i) $s_i < s_{i+1}$, $c_i < c_{i+1}$
- decreasing bandwidth (b_i) $b_i > b_{i+1}$
- increasing xfer unit (x_i) $x_i < x_{i+1}$

Level 0 Registers — ISA feature

Memory Abstractions

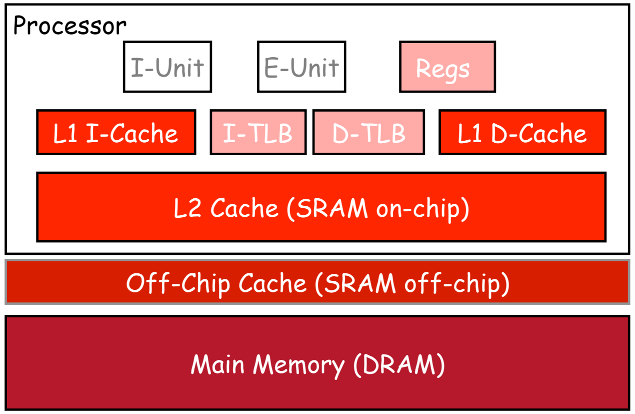
Level 1 (n levels of) Caches

Level 2 Main Memory (Primary Storage)

Level 3 Disks (Secondary Storage)

Level 4 Tape Backup (Tertiary Storage)

Processor/Memory Boundaries



Caches

An automatically managed hierarchy

"A hiding place, esp. of goods, treasure, etc." -- OED

Keep recently accessed block

- temporal locality

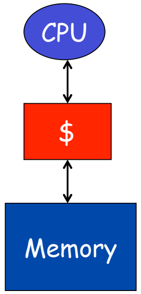
Break memory into blocks (several bytes)

and transfer data to/from cache in blocks

- spatial locality

A lot of architectures opt for software managed scratch-pad memory instead

e.g. Cray-1, embedded processors, Why??



Cache Performance

Assume

- Cache access time is equal to 1 cycle
- Cache miss ratio is 0.01
- Cache miss penalty is 20 cycles

Mean access time

= Cache access time + miss ratio * miss penalty

= 1 + 0.01 * 20 = 1.2

Typically

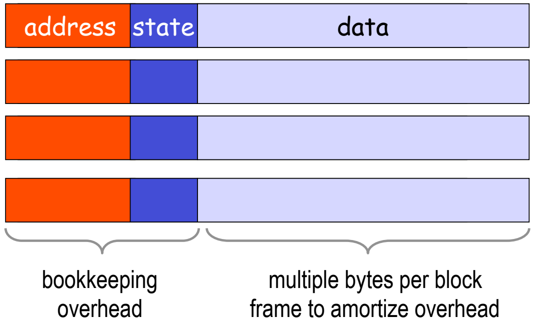
- level-1 is 16K-64K, level-2 is 512K-4M,memory is 8G-2TB
- level-1 as fast as the processor *(increasingly 3-cycles)*
- level-1 is 1/10000 capacity but contains 98% of references

Memoization & amortization

Cache (Abstractly)

Keep recently accessed block in “block frame”

- state (e.g., valid)
- address tag
- data



address | state | data

bookkeeping overhead

multiple bytes per block frame to amortize overhead

Cache (Abstractly)

On memory read

- if incoming address corresponds to one of the stored address tag then
  - HIT
  - return data
- else
  - MISS
  - choose & displace a current block in use
  - fetch new (referenced) block from memory into frame
  - return data

- Where and how to look for a block? (Block placement)

- Which block is replaced on a miss? (Block replacement)

- What happens on a write? Write strategy (Later)

Terminology

block (cache line) — minimum unit that may be present

hit — block is found in the cache

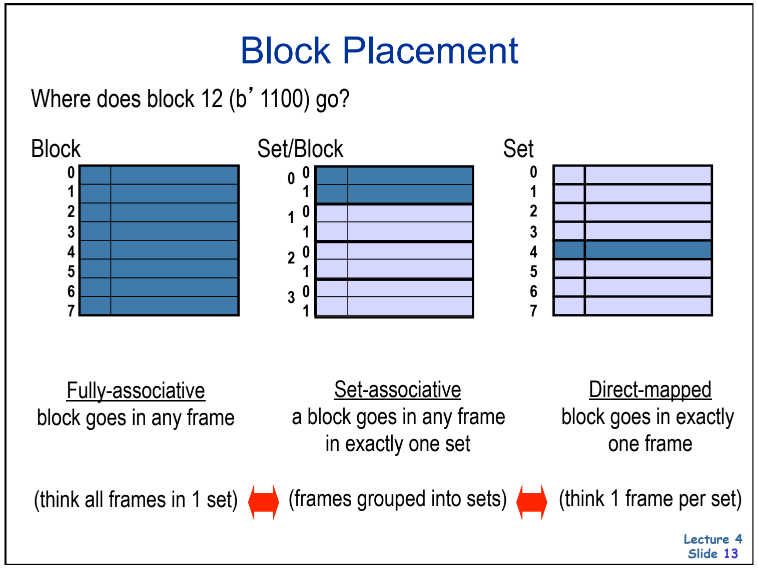
miss — block is not found in the cache

miss ratio — fraction of references that miss

hit time — time to access the cache

miss penalty

- time to replace block in the cache + deliver to upper level
- access time — time to get first word
- transfer time — time for remaining words



Block Placement

Where does block 12 (b' 1100) go?

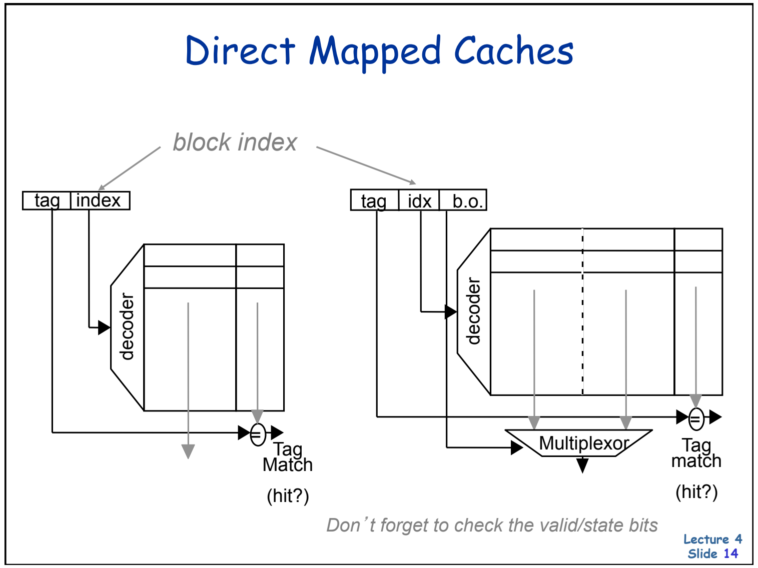
Fully-associative
block goes in any frame

Set-associative
a block goes in any frame
in exactly one set

Direct-mapped
block goes in exactly
one frame

(think all frames in 1 set) ⟷ (frames grouped into sets) ⟷ (think 1 frame per set)

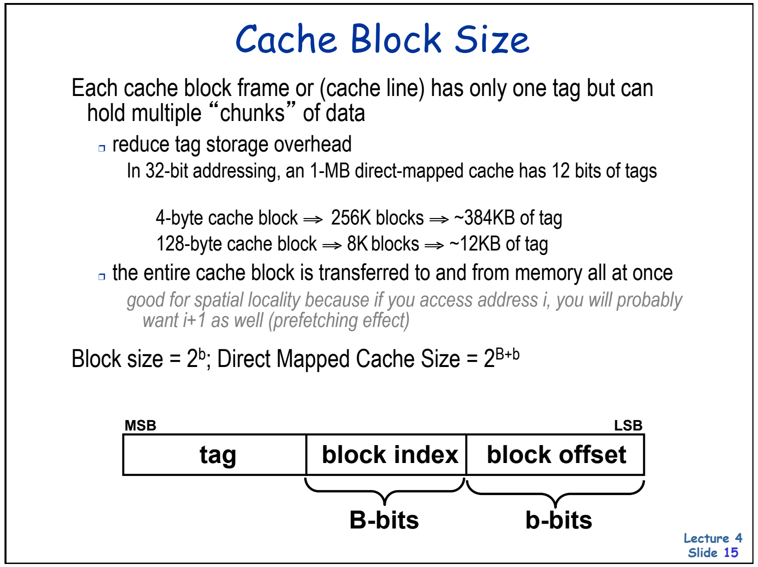
Lecture 4
Slide 13



Direct Mapped Caches

Don't forget to check the valid/state bits

Lecture 4
Slide 14



Cache Block Size

Each cache block frame or (cache line) has only one tag but can hold multiple "chunks" of data

- reduce tag storage overhead

  In 32-bit addressing, an 1-MB direct-mapped cache has 12 bits of tags

  4-byte cache block ⇒ 256K blocks ⇒ ~384KB of tag

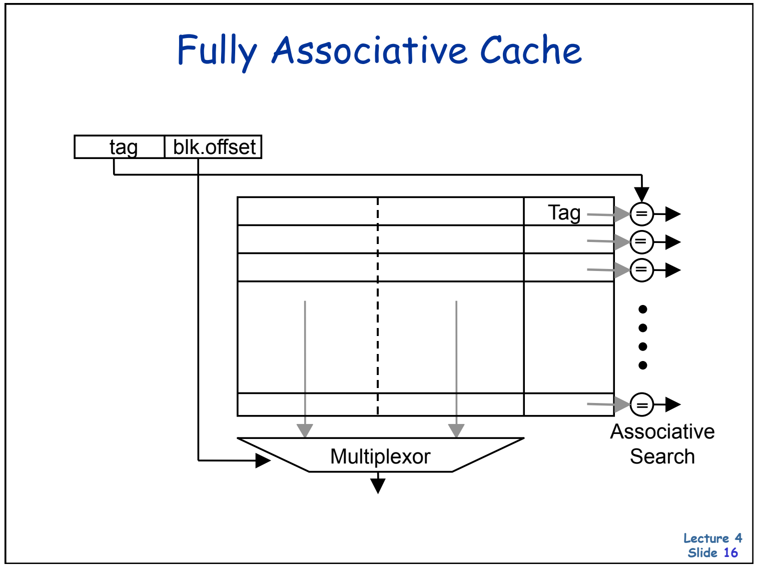
  128-byte cache block ⇒ 8K blocks ⇒ ~12KB of tag

- the entire cache block is transferred to and from memory all at once

  good for spatial locality because if you access address i, you will probably want i+1 as well (prefetching effect)

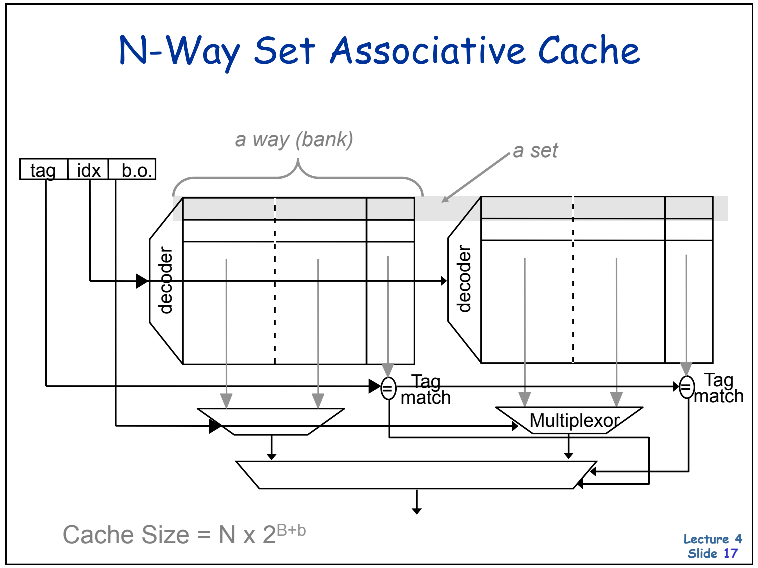
Block size = 2^b; Direct Mapped Cache Size = 2^{B+b}

Lecture 4
Slide 15



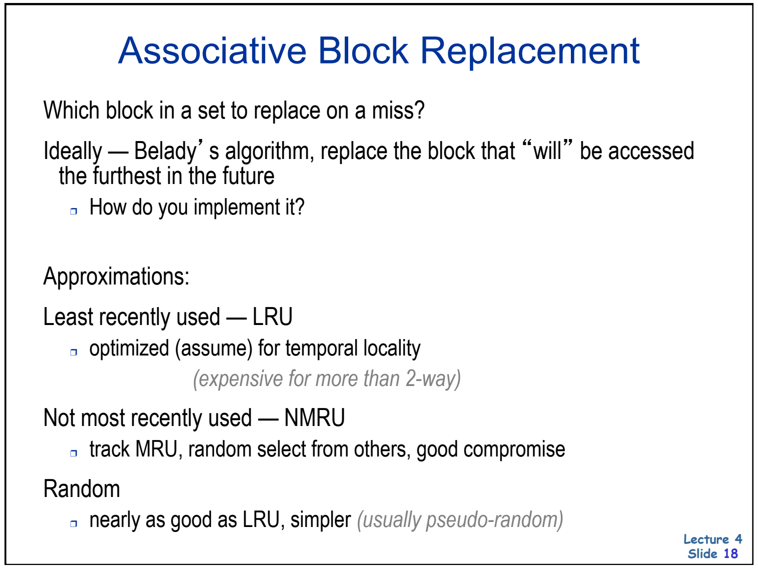
Fully Associative Cache

Lecture 4
Slide 16



N-Way Set Associative Cache

Cache Size = N x 2^{B+b}



Associative Block Replacement

Which block in a set to replace on a miss?

Ideally — Belady's algorithm, replace the block that "will" be accessed the furthest in the future

- How do you implement it?

Approximations:

Least recently used — LRU

- optimized (assume) for temporal locality

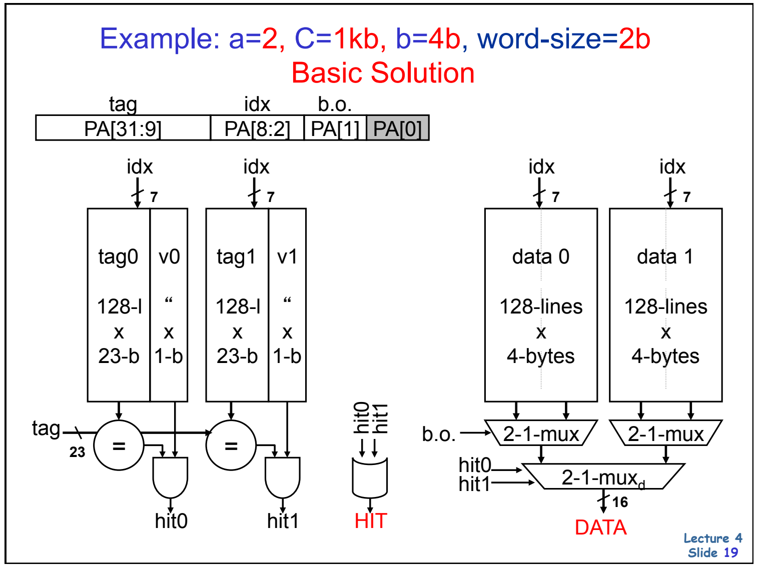
  (expensive for more than 2-way)

Not most recently used — NMRU

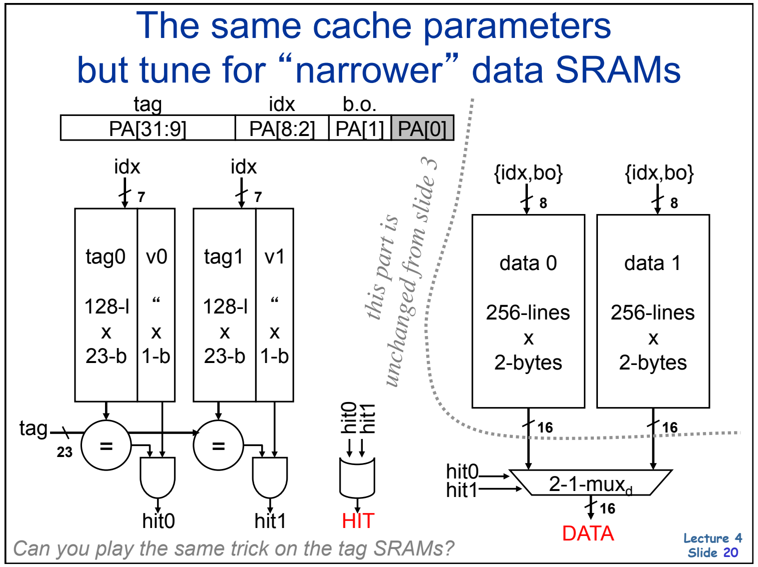
- track MRU, random select from others, good compromise

Random

- nearly as good as LRU, simpler *(usually pseudo-random)*

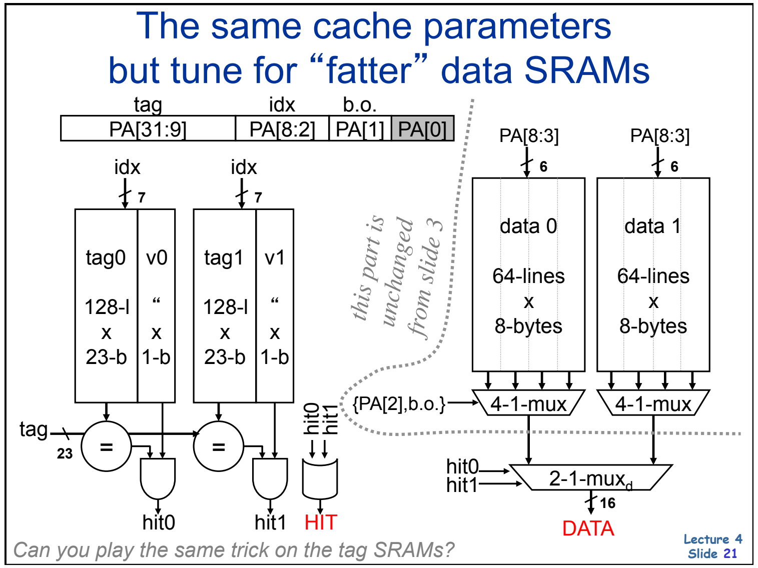


Example: a=2, C=1kb, b=4b, word-size=2b
Basic Solution



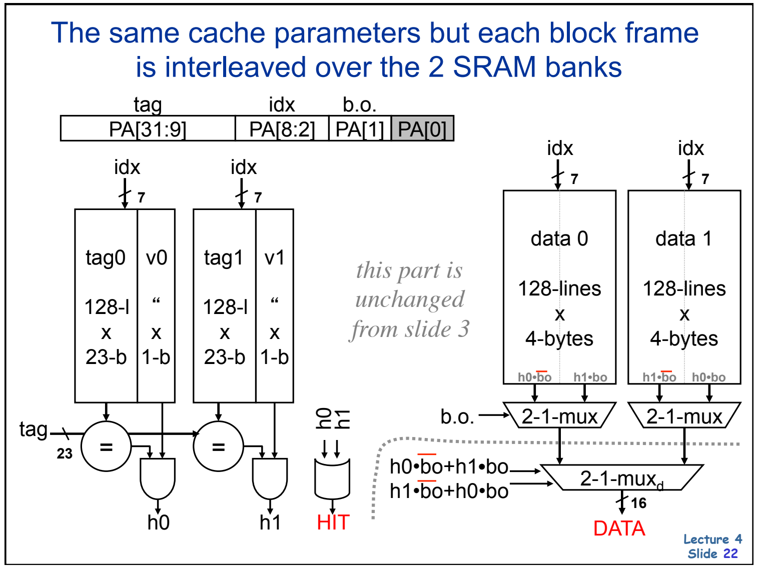
The same cache parameters
but tune for "narrower" data SRAMs

Can you play the same trick on the tag SRAMs?

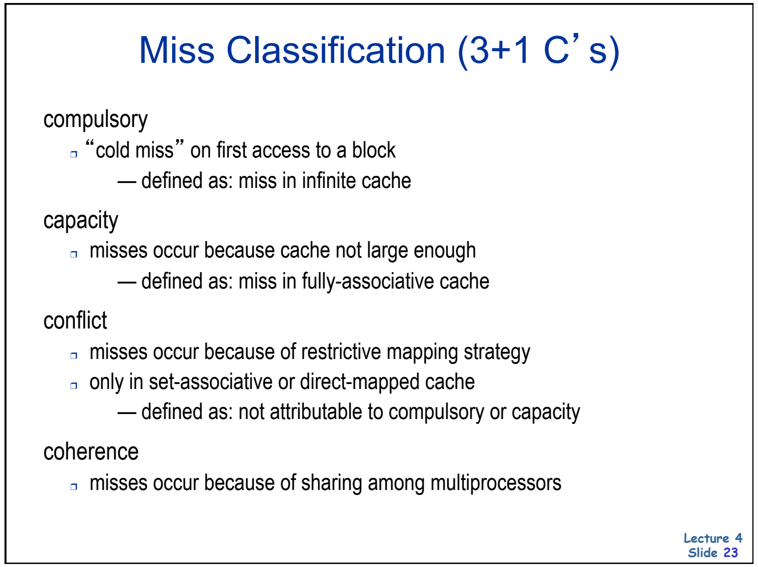


The same cache parameters but tune for “fatter” data SRAMs

Can you play the same trick on the tag SRAMs?



The same cache parameters but each block frame is interleaved over the 2 SRAM banks



Miss Classification (3+1 C’ s)

compulsory

- “cold miss” on first access to a block
  - — defined as: miss in infinite cache

capacity

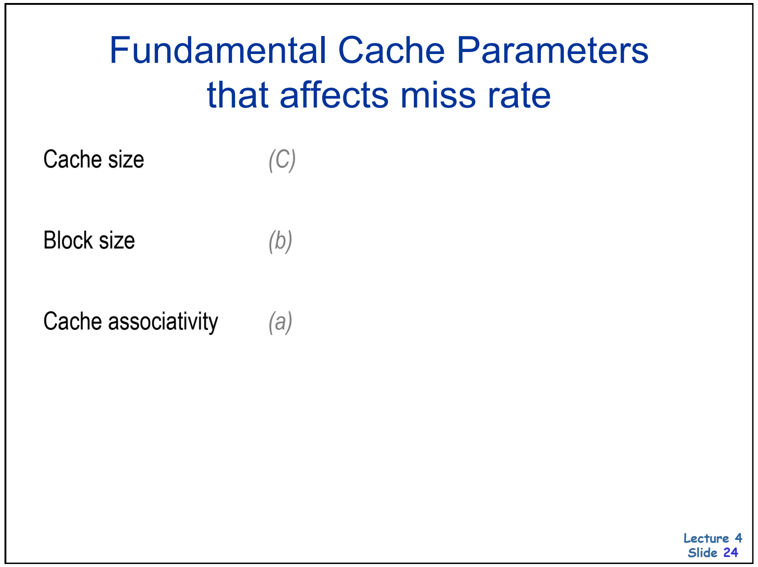
- misses occur because cache not large enough
  - — defined as: miss in fully-associative cache

conflict

- misses occur because of restrictive mapping strategy
- only in set-associative or direct-mapped cache
  - — defined as: not attributable to compulsory or capacity

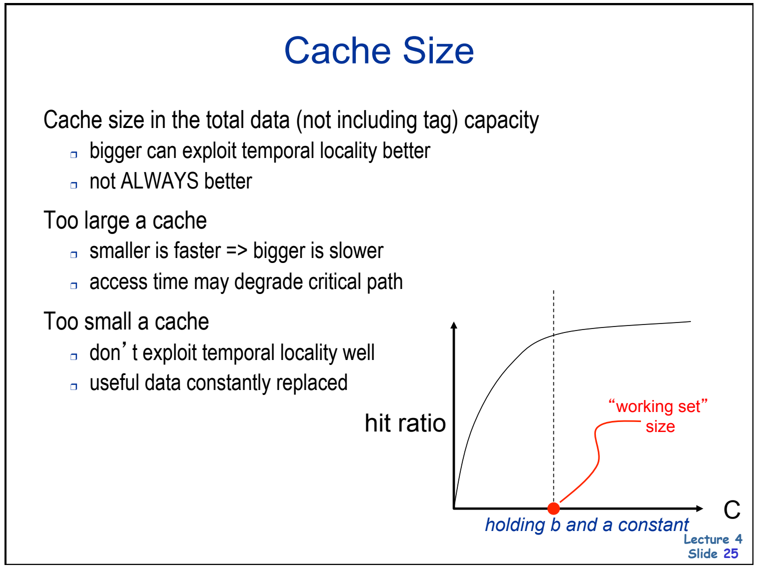
coherence

- misses occur because of sharing among multiprocessors



Fundamental Cache Parameters that affects miss rate

Cache size	*(C)*
Block size	*(b)*
Cache associativity	*(a)*



Cache Size

Cache size in the total data (not including tag) capacity

- bigger can exploit temporal locality better
- not ALWAYS better

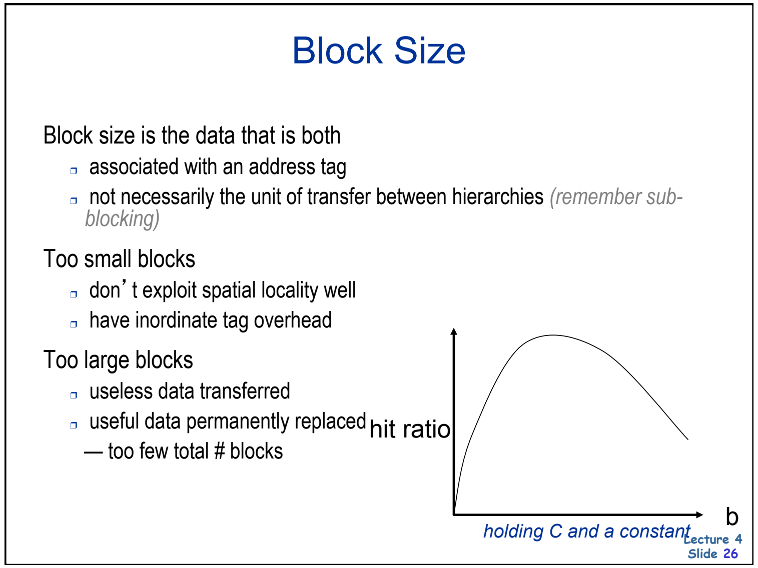
Too large a cache

- smaller is faster => bigger is slower
- access time may degrade critical path

Too small a cache

- don’t exploit temporal locality well
- useful data constantly replaced

Lecture 4
Slide 25



Block Size

Block size is the data that is both

- associated with an address tag
- not necessarily the unit of transfer between hierarchies *(remember sub-blocking)*

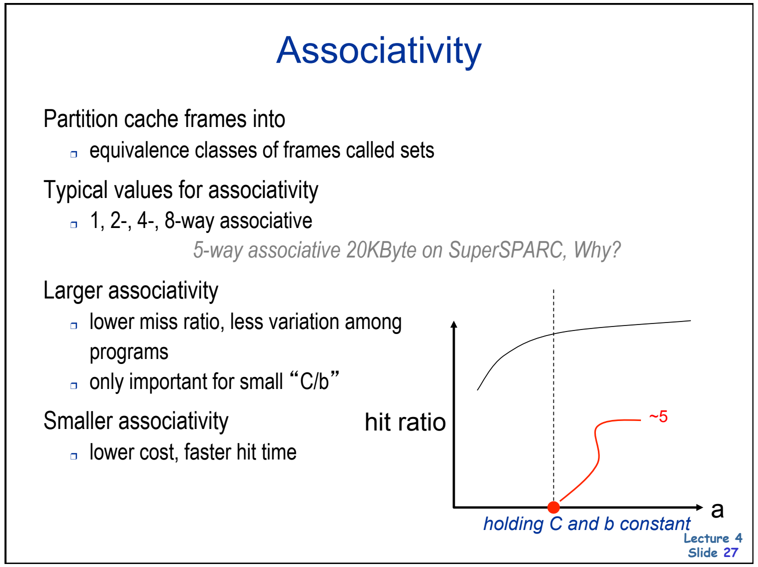
Too small blocks

- don’t exploit spatial locality well
- have inordinate tag overhead

Too large blocks

- useless data transferred
- useful data permanently replaced
  — too few total # blocks

Lecture 4
Slide 26



Associativity

Partition cache frames into

- equivalence classes of frames called sets

Typical values for associativity

- 1, 2-, 4-, 8-way associative

5-way associative 20KByte on SuperSPARC, Why?

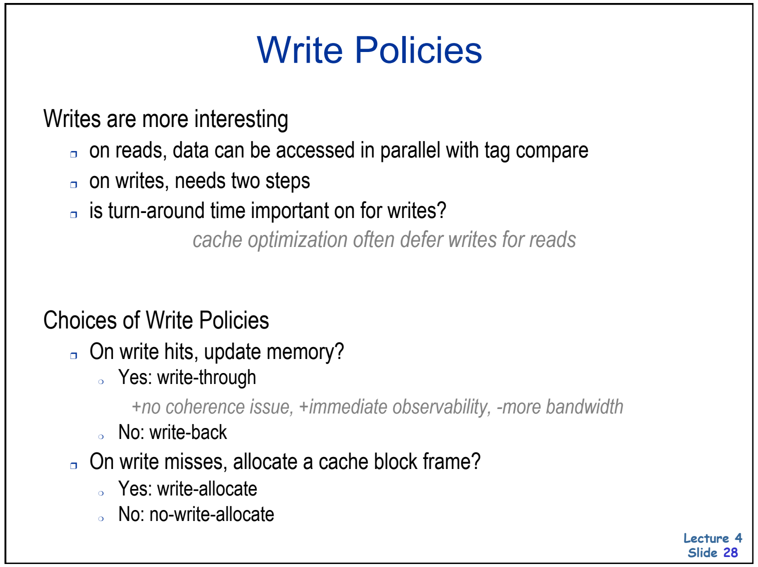
Larger associativity

- lower miss ratio, less variation among programs
- only important for small “C/b”

Smaller associativity

- lower cost, faster hit time

Lecture 4
Slide 27



Write Policies

Writes are more interesting

- on reads, data can be accessed in parallel with tag compare
- on writes, needs two steps
- is turn-around time important on for writes?

cache optimization often defer writes for reads

Choices of Write Policies

- On write hits, update memory?
  - Yes: write-through
    +no coherence issue, +immediate observability, -more bandwidth
  - No: write-back
- On write misses, allocate a cache block frame?
  - Yes: write-allocate
  - No: no-write-allocate

Lecture 4
Slide 28

Write Policies (Cont.)

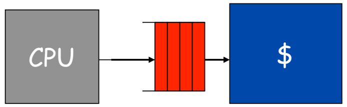
Write-through

- update memory on each write
- keeps memory up-to-date
- traffic/reference = f_{writes}, e.g. 0.20
  independent of cache performance (miss ratio)

Write-back

- update memory only on block replacement
- many cache lines are only read and never written to
- add “dirty” bit to status word
  - originally cleared after replacement
  - set when a block frame is written to
  - only write back a dirty block, and “drop” clean blocks w/o memory update
- traffic/reference = f_{dirty} x miss x B
  - e.g., traffic/reference = 1/2 x 0.05 x 4 = 0.1

Write Buffers



Buffer CPU writes

- allows reads to proceed
- stall only when full
- data dependence?
  - What happens on dependent loads/stores?

Write Buffers (Cont.)

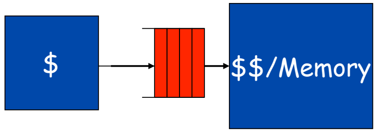
Write Policy	Write Alloc	Hit/Miss	Write Buffer writes to
Back	Yes	Both	Cache
Back	No	Hit	Cache
Back	No	Miss	Memory
Through	Yes	Both	Both
Through	No	Hit	Both
Through	No	Miss	Memory

Write Buffers (Cont.)

Design issues:

- Design for bursts
- Coalesce adjacent writes?
- Sixteen entries is typical

Writeback Buffers



Between write-back cache and next level

1. Move replaced, dirty blocks to buffer
2. Read new line
3. Move replaced data to memory

Usually need 8 write-back buffer entries

Mark Hill’s DM vs. SA: “Bigger & Dumber is Better”

$t_{avg} = t_{hit}$ + miss ratio x t_{miss}

- comparable DM and SA caches with same t_{miss}
- but, associativity that minimizes t_{avg} is often smaller than associativity that minimizes miss ratio

remember:

$$diff(t_{cache}) = t_{cache}(SA) - t_{cache}(DM) \geq 0$$

$$diff(\text{miss}) = \text{miss}(SA) - \text{miss }(DM) \leq 0$$

e.g.,

assuming $diff(t_{cache})$= 0 => SA better

assuming $diff$(miss) = -1%, t_{miss} = 20

$\Rightarrow$ if $diff(t_{cache})$ > 0.2 cycle then SA loses

“Harvard” vs. “Princeton”

Unified *(sometimes known as Princeton)*

- less costly, dynamic response, handles writes to instructions

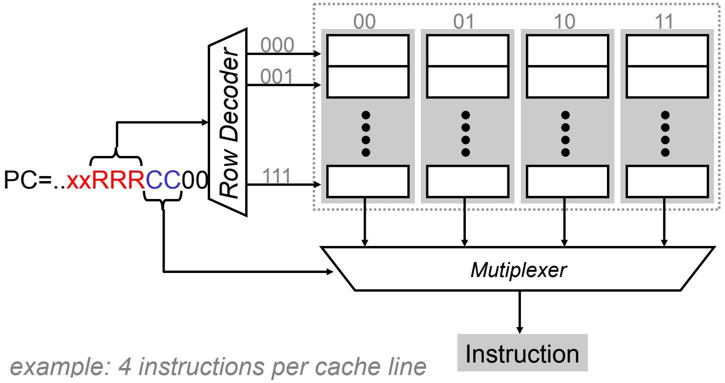
Split I and D *(sometimes known as Harvard)*

- most of the time code and data don’t mix
- 2x bandwidth, place close to I/D ports
- can customize size (I-footprint generally smaller than d-footprint), no interference between I/D
- self-modifying code can cause “coherence” problems

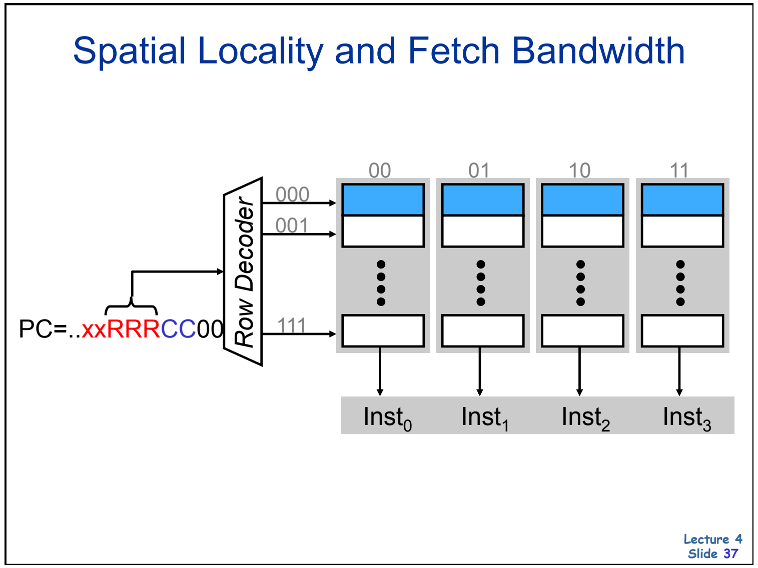
Caches should be split for frequent simultaneous I & D access

- no longer a question in “high-performance” on-chip L-1 caches

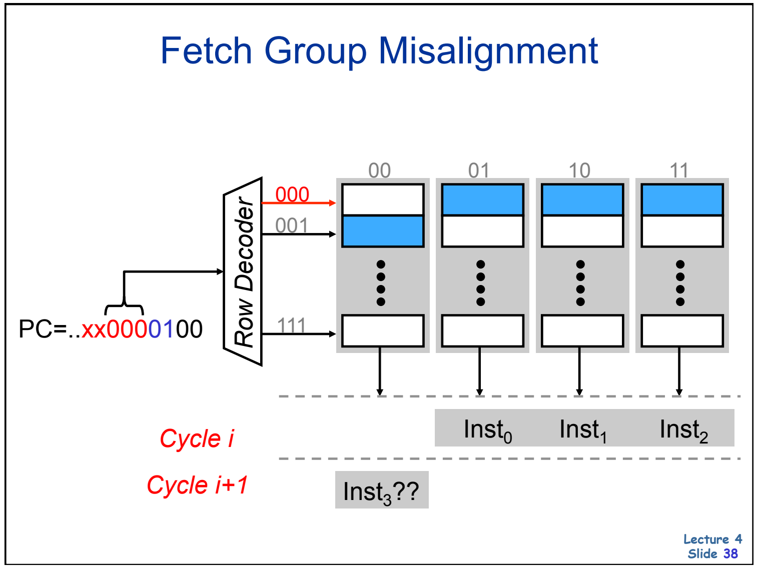
L1 Instruction Cache Issues



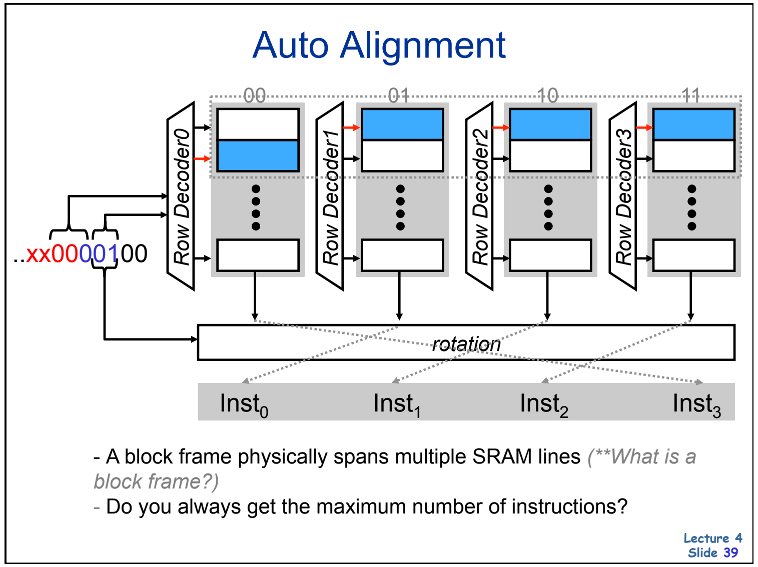
example: 4 instructions per cache line



Spatial Locality and Fetch Bandwidth



Fetch Group Misalignment



Auto Alignment

- A block frame physically spans multiple SRAM lines *(**What is a block frame?)*
- Do you always get the maximum number of instructions?