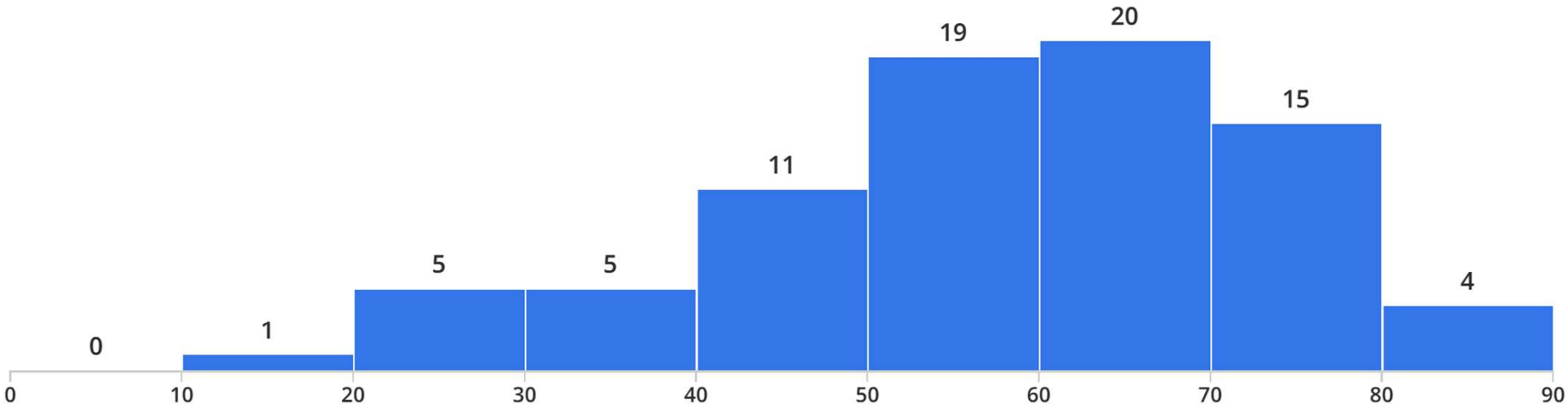
18-447 Lecture 15: Caching in Concept

James C. Hoe
Department of ECE
Carnegie Mellon University

Midterm Class Distribution

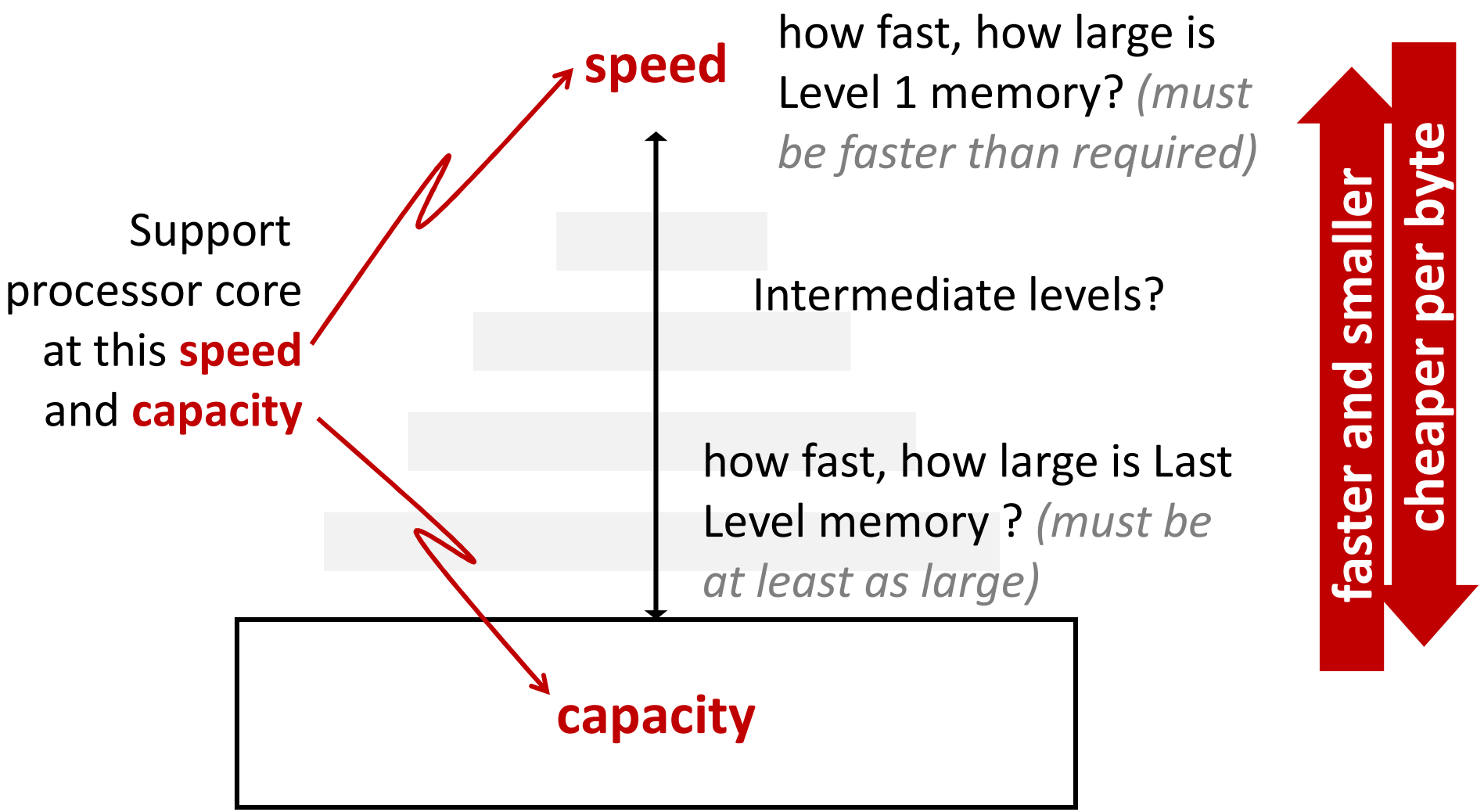


Minimum	Median	Maximum	Mean	Std Dev
19.0	59.0	85.0	57.38	15.7

Housekeeping

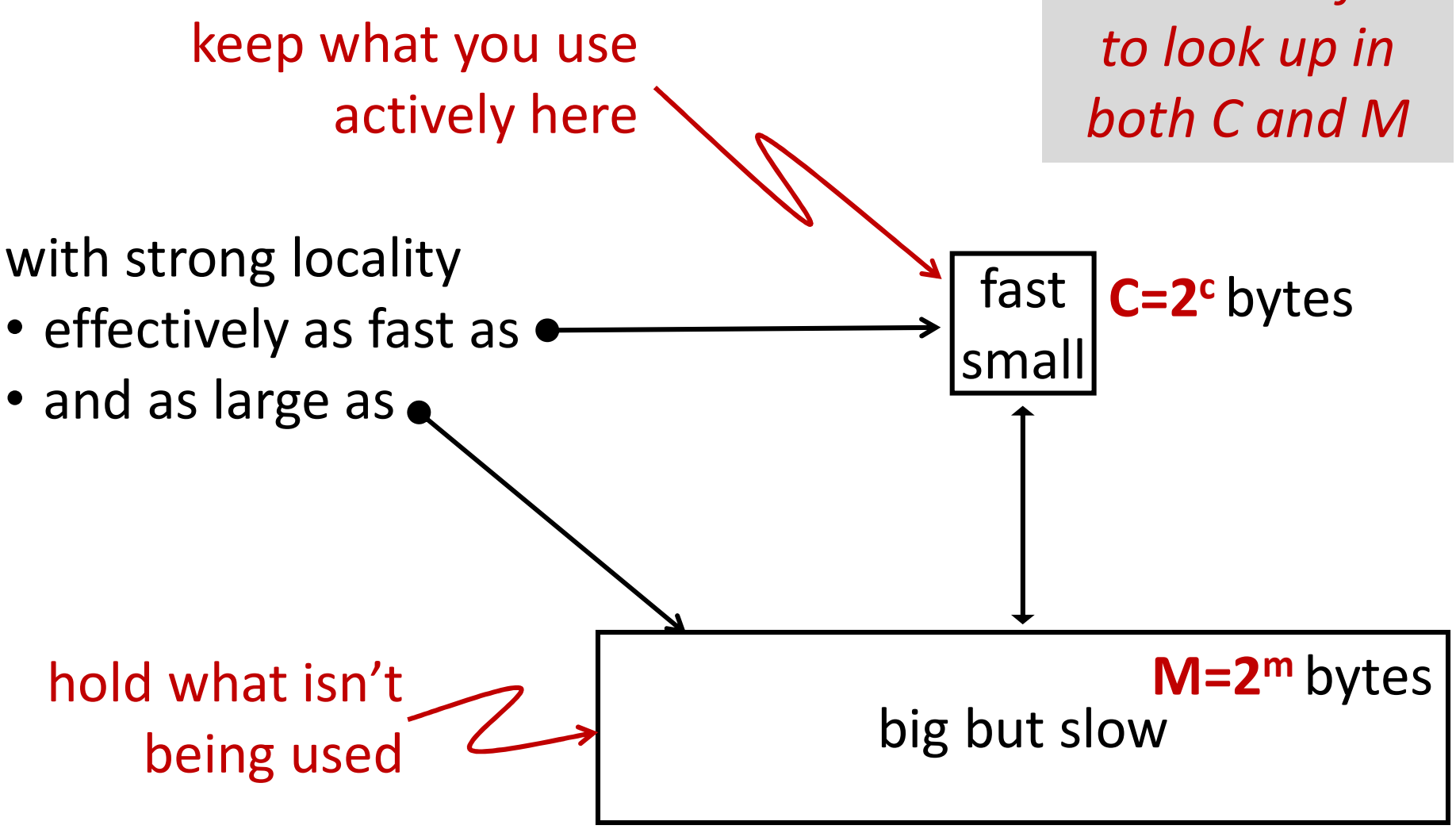
- Your goal today
  - understand caching ("aBC" and "3 C's") in the abstract and in isolation—*where performance means hit vs miss, one cache by itself*
- Notices
  - HW 4, out next week
  - Lab 3, due next week
  - Final Exam, Fri, May 3rd, 1pm
- Readings
  - P&H Ch 5

Last Lecture: Memory Hierarchy Design



Today: A Cache

use index of M to look up in both C and M



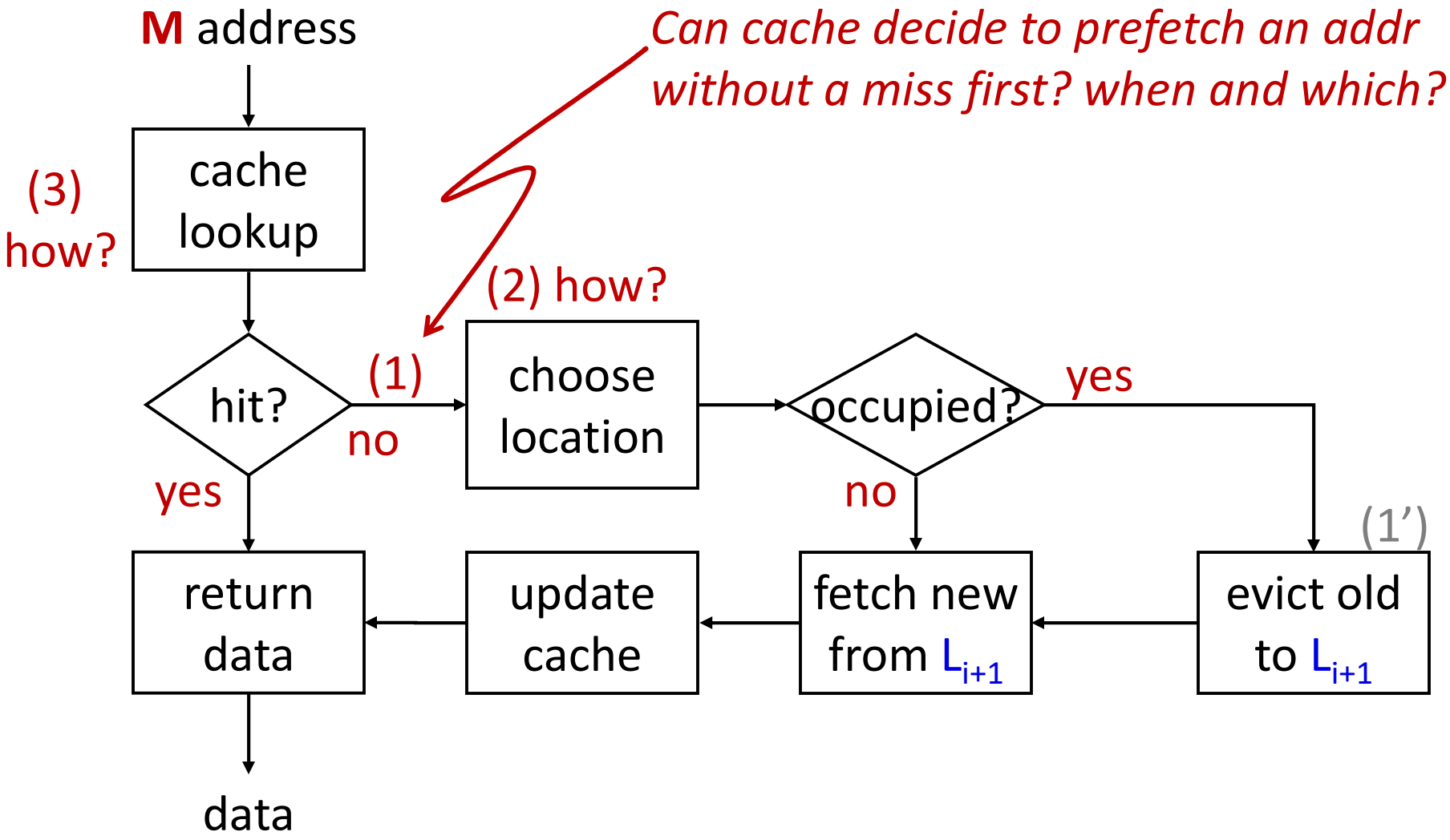
Bottomline Issues

- Potentially **$M=2^m$** bytes of memory, how to keep "copies" of most frequently used locations in **C** bytes of fast storage where **C << M**

- Basic issues (intertwined)
  (1) when to cache a "copy" of a memory location
  (2) where in fast storage to keep the "copy"
  (3) how to find the "copy" later on *(LW and SW only give indices into* **M***)*

- Viable solutions must be fast and efficient

Basic Operation
Ans (1): demand-driven



Basic Cache Parameters

ISA

- **M = 2^m** : size of address space in bytes
  example values: 2^{32}, 2^{64}
- **G=2^g** : cache access granularity in bytes
  example values: 4, 8

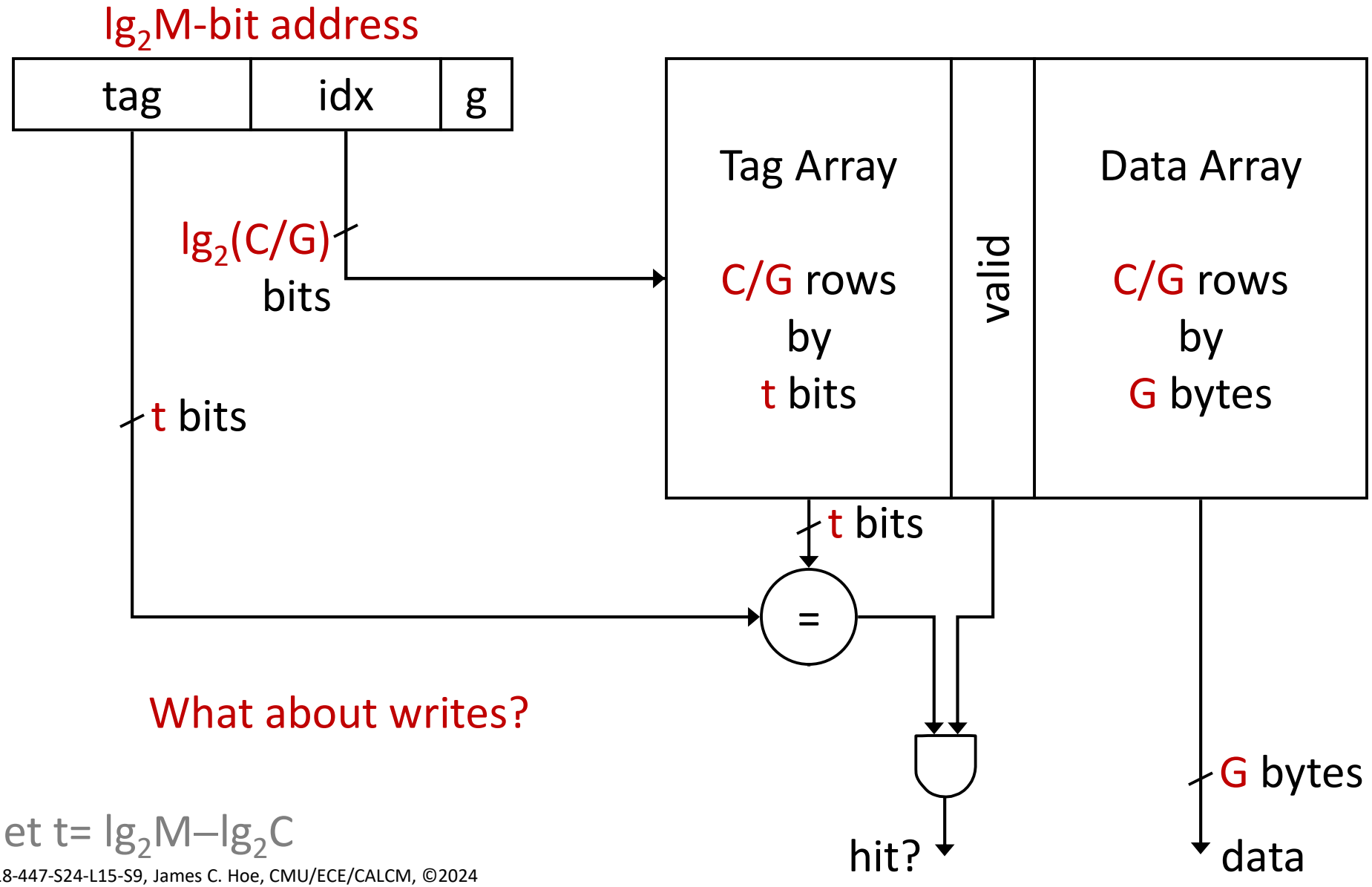
Implementation

- **C** : "capacity" of cache in bytes
  example values: 16 KByte (L1), 1 MByte (L2)
- **B = 2^b**: "block size" in bytes
  example values: 16 (L1), >64 (L2)
- **a**: "associativity" of the cache
  example values: 1, 2, 4, 5(?),... "C/B"

introduce today

C/a should be a 2-power

Direct-Mapped Placement (first try)



let t= $lg_2M - lg_2C$

Storage Overhead and Block Size

- For each cache block of **G** bytes, also storing "**t+1**" bits of tag (where $t=\lg_2 M - \lg_2 C$)
  - if $M=2^{32}$, $G=4$, $C=16K=2^{14}$

    $\Rightarrow$ **t**=18 bits for each 4-byte block

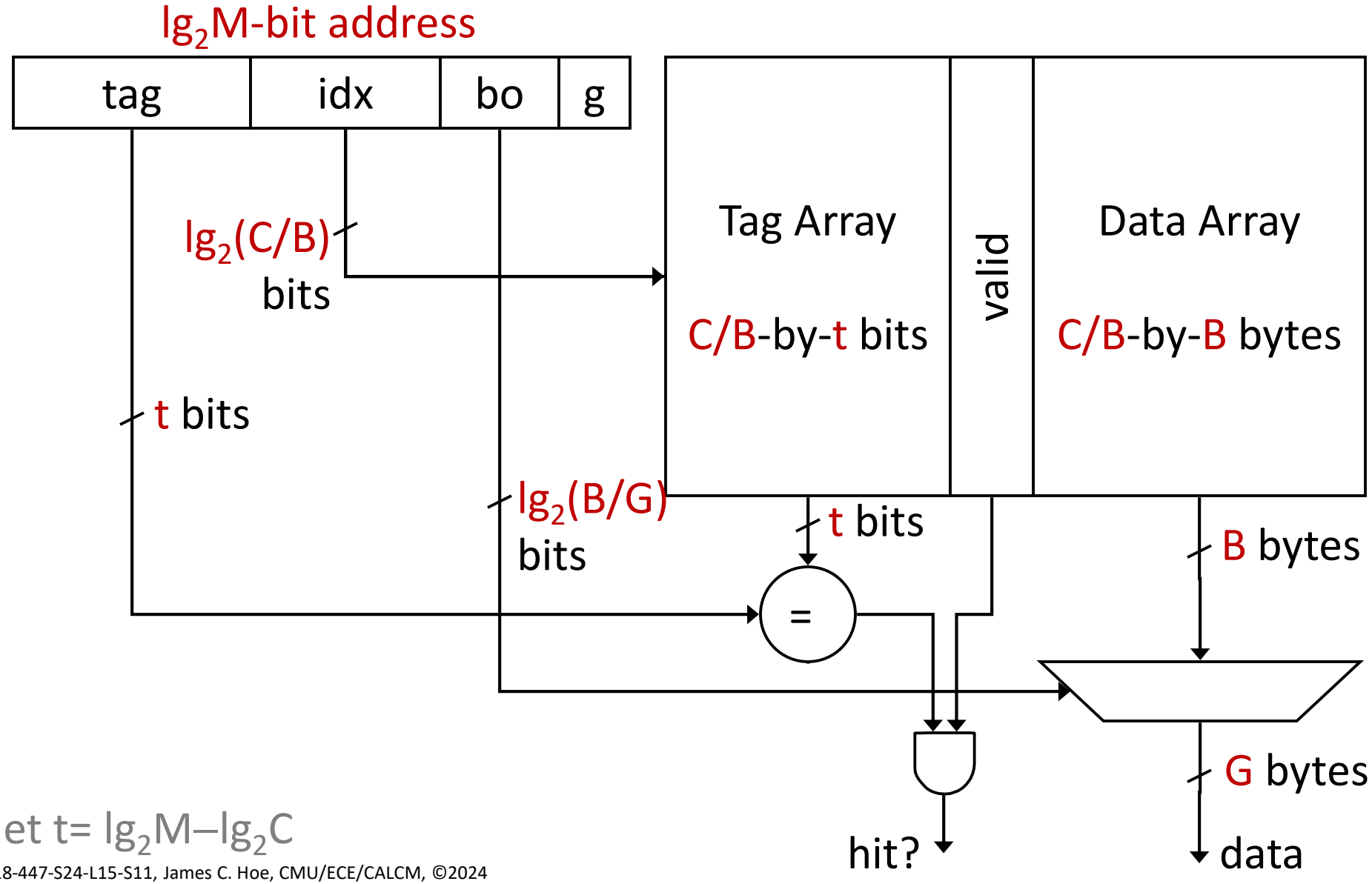
    60% overhead; 16KB cache actually 25.5KB SRAM
- Solution: "amortize" tag over larger **B**-byte block
  - manage **B/G** consecutive words as indivisible unit
  - if $M=2^{32}$, **B**=16, **G**=4, **C**=16K

    $\Rightarrow$ **t**=18 bits for each 16-byte block

    15% overhead; 16KB cache actually 18.4KB SRAM
  - spatial locality also says this is good *(Q1: when)*
- Larger caches wants even bigger blocks

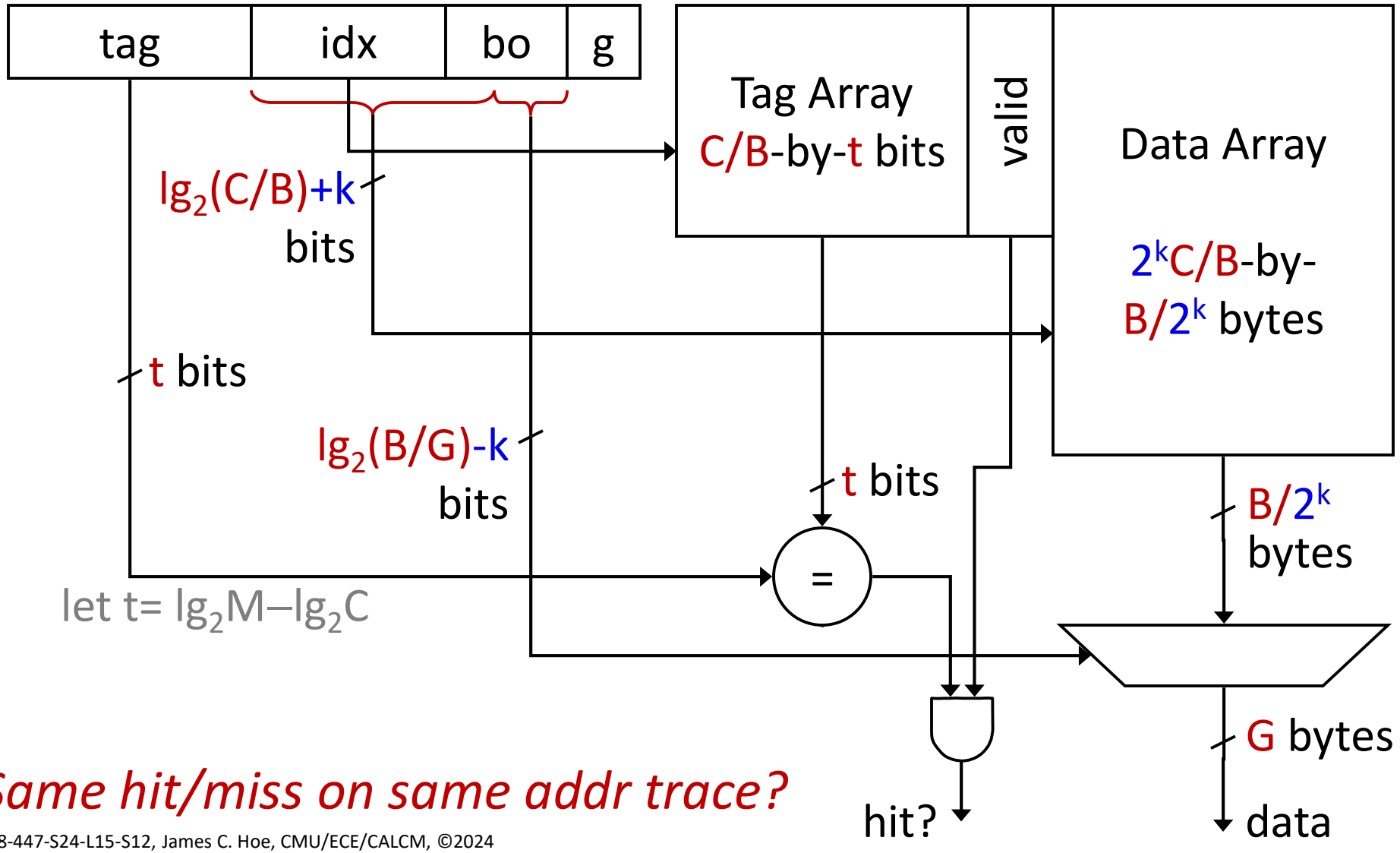
B

Direct-Mapped$_{M,G,C,B}$ "Flow Chart" (2)&(3)



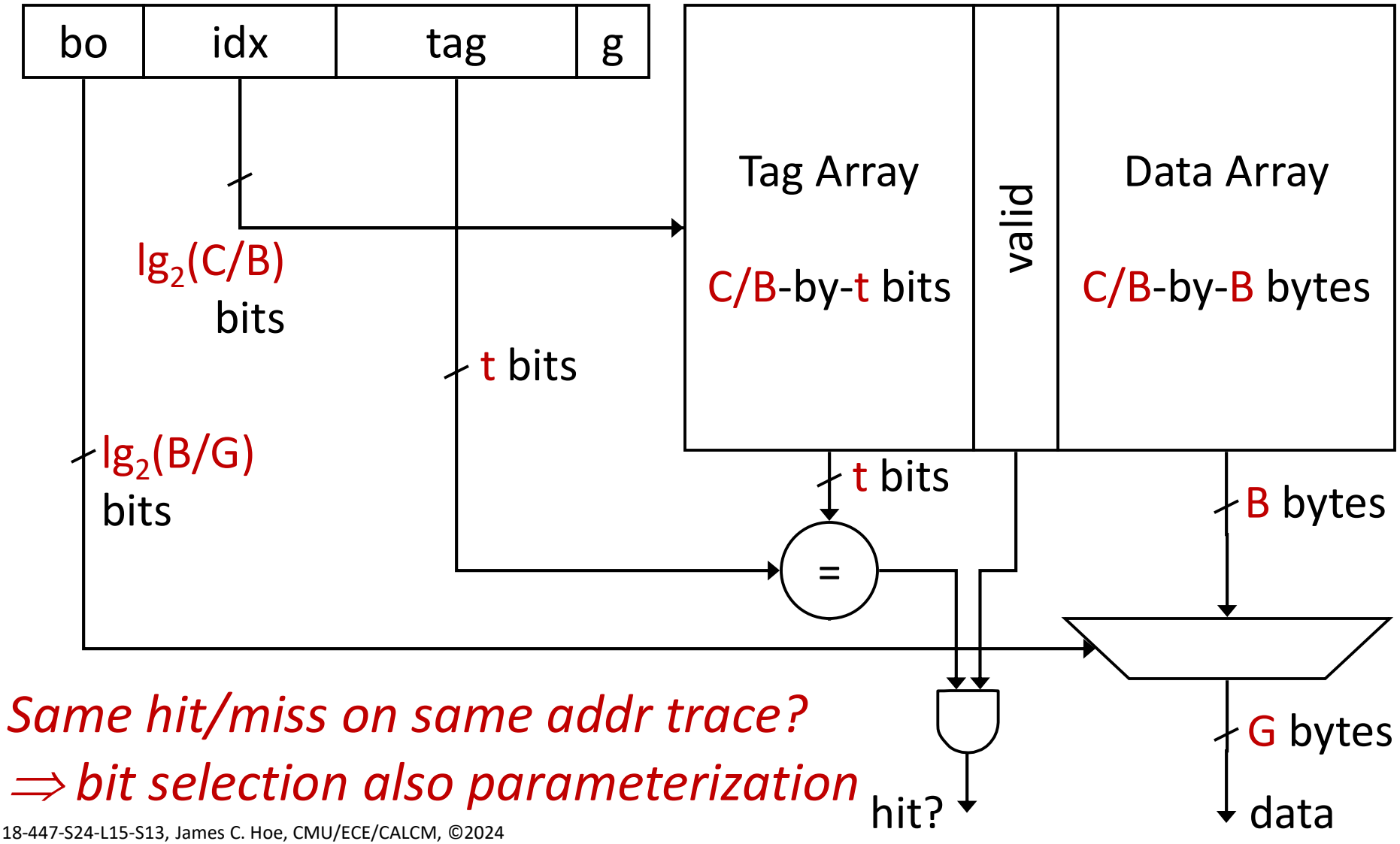
let t= $lg_2M - lg_2C$

Is this the same placement policy?



Same hit/miss on same addr trace?

Is this Direct-Mapped$_{M,G,C,B}$? Is it Valid?



Same hit/miss on same addr trace?

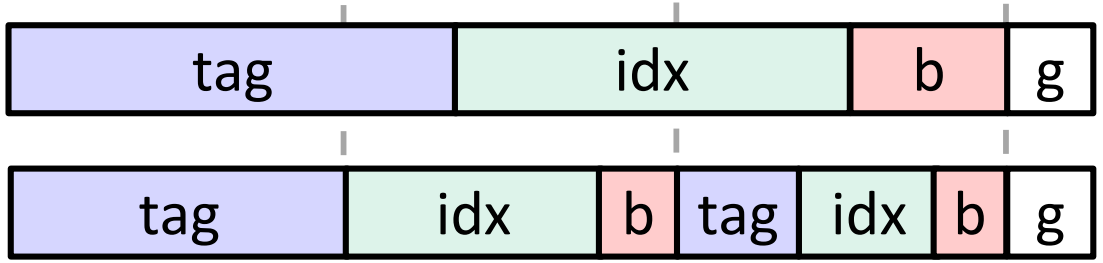
$\Rightarrow$ bit selection also parameterization

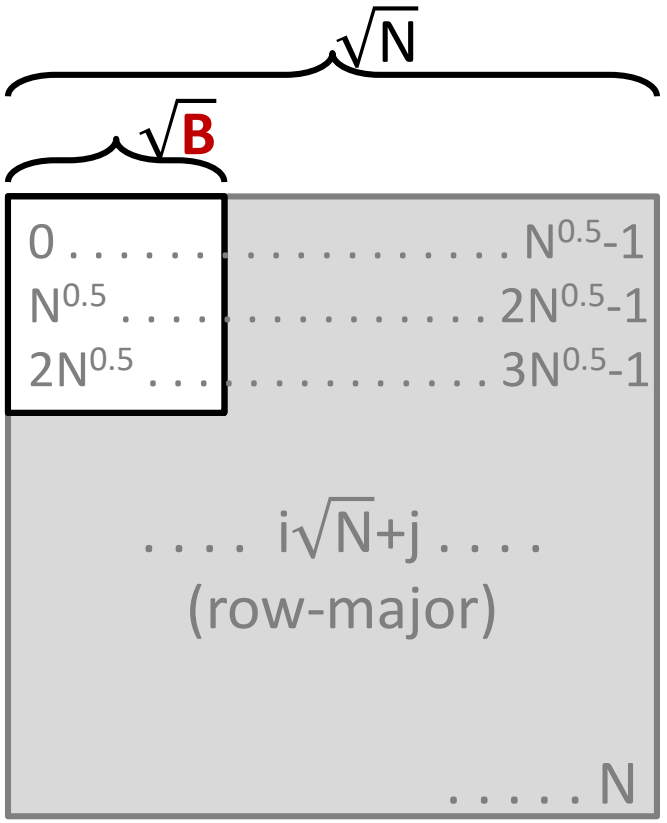
Matching Placement to Atypical Locality

- Suppose hypothetical GPU
  - $\sqrt{N}$-by-$\sqrt{N}$ **G**-byte pixels per frame
  - row-major layout

row-major 2D array

- N>>**C**/**G**
  - spatial locality in $\sqrt{B}$-by-$\sqrt{B}$ tiles
  - working set is consecutive columns
- Cache indexing given same **B**, **C**

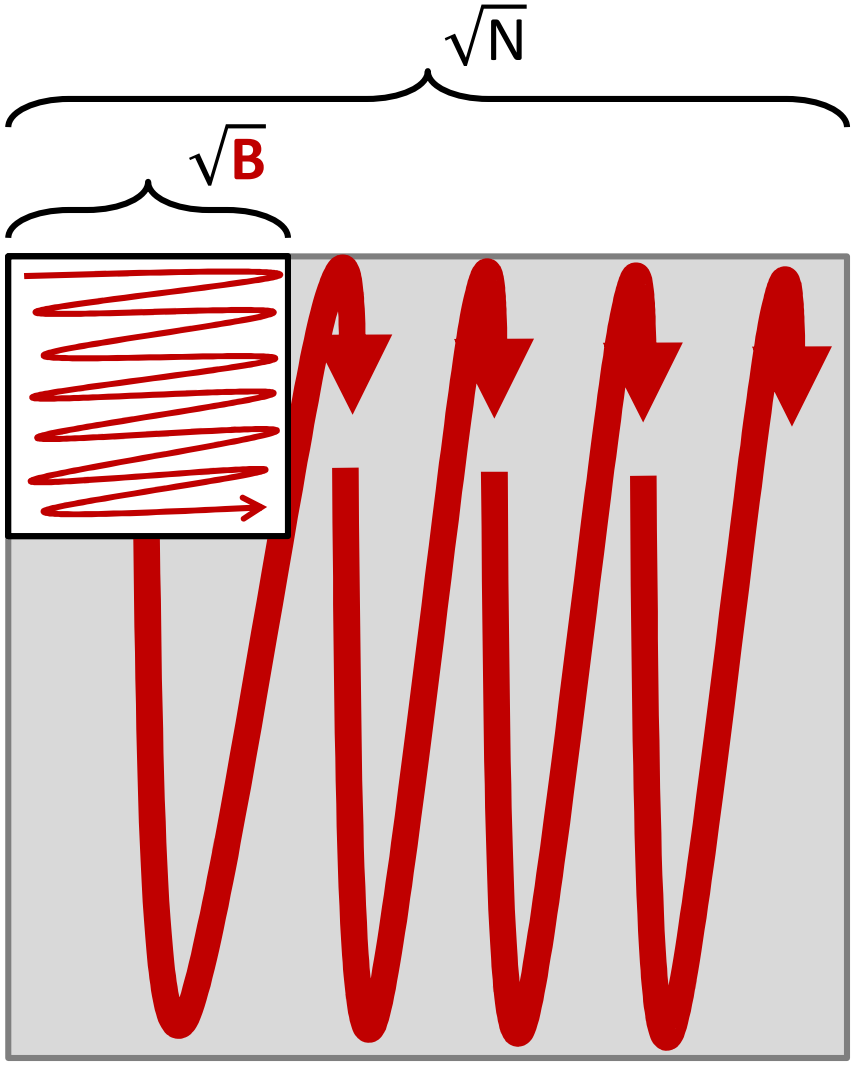




textbook

domain-specialized

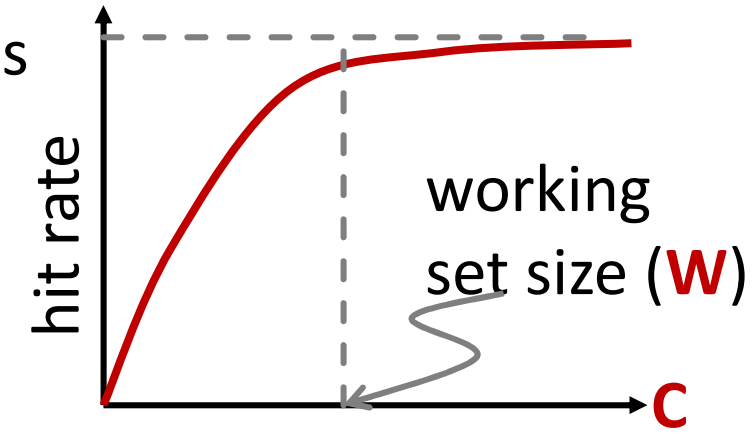
Also the other way: match data layout to placement



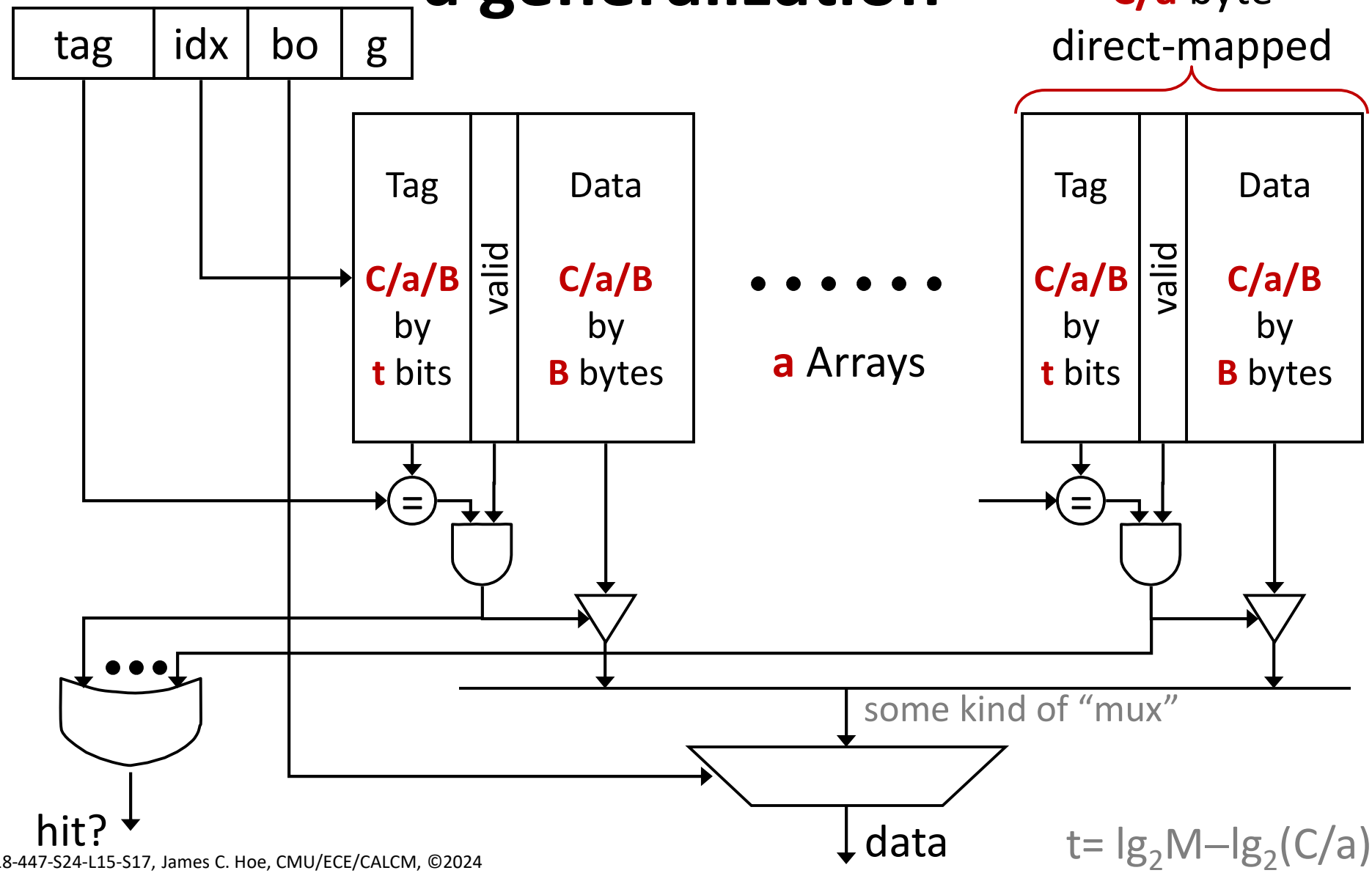
Direct-Mapped Policy in Essence

- **C**-byte storage array managed as **C**/**B** cache blocks
- A given block address directly maps to exactly one choice of cache block (by block index field)
- Block addresses with same block index field map to same cache block
  - of 2^{t} such addresses, hold only one at a time
  - even if **C** > working set size, conflict is possible

    ("working set" is not one continuous region)
  - probability 2 random addresses conflict is 1/(**C**/**B**); likelihood for conflict increases with decreasing number of blocks



Set Associative Placement "Flow Chart": a generalization



$t = \lg_2 M - \lg_2 (C/a)$

a-way Set-Associative Placement Policy

- **C** bytes of storage divided into **a** direct-mapped arrays (aka "ways" and sometimes "banks")
  - each "way" has (**C**/**a**)/**B** cache blocks
  - a given block address maps to exactly one choice per "way"; **a** choices constitute the "set"

    direct-mapped is special case **a**=1
  - overhead: **a** comparators and **a**-to-1 multiplexer
- Block addresses with same index map to same set
  - 2^t such addresses; hold **a** different ones at a time
  - if **C** > working set size

    higher-degree of associativity ⇒ fewer conflicts

    What if **C** < working set size?

associativity

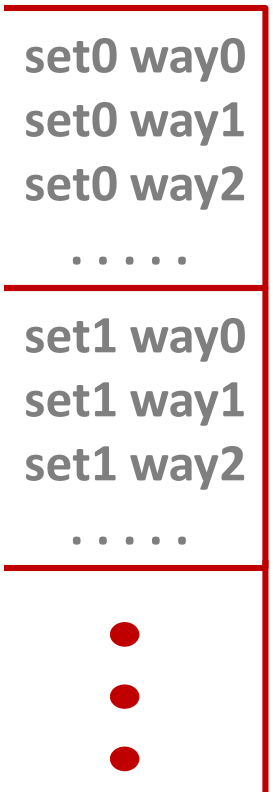
Replacement Policy to Choose from a

- New block displaces an existing block from "set"
  - pick the one that is least recently used (LRU)

    exactly LRU expensive for **a**>2
  - pick any one except the most recently used
  - ~~pick the most recently used one~~
  - ~~pick one based on some part of the address bits~~
  - pick the one used again furthest in the future Belady
  - pick a (pseudo) random one
- No real best choice; second-order impact only
  - if actively using less than **a** blocks in a set, any sensible replacement policy will quickly converge
  - if actively using more than **a** blocks in a set, no replacement policy can help you

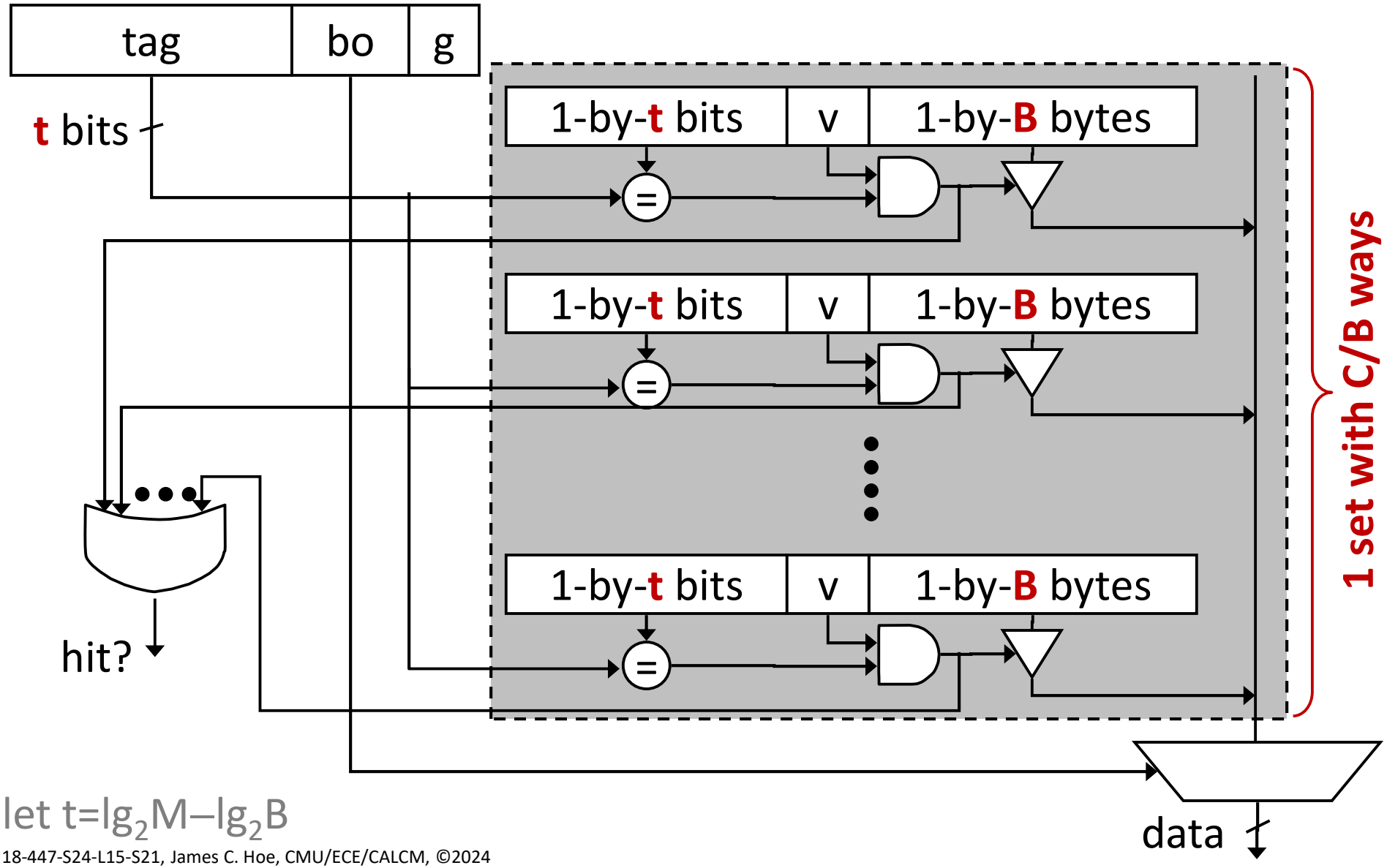
Policy vs Realization

- Associativity is a placement policy
  - it says a block address could be placed in one of **a** different blocks
  - it doesn't say "ways" are parallel look-up banks
- "Pseudo" **a**-way associative cache
  - given a direct-mapped array with **C**/**B** blocks
  - logically partition into **C**/**B**/**a** sets
  - given an address **A**, index into set and sequentially search its ways
- Optimization: record the most recently used way (MRU) to check first



e.g., used by MIPS R10K off-CPU L2

Fully Associative Cache: a≡C/B

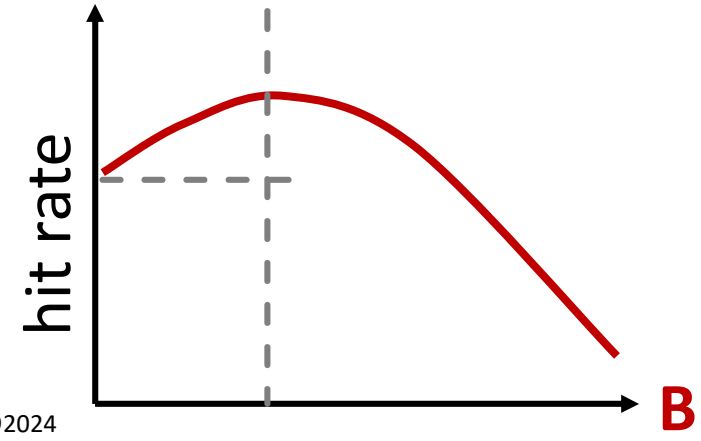


let $t=\lg_2 M - \lg_2 B$

3C's of Cache Misses

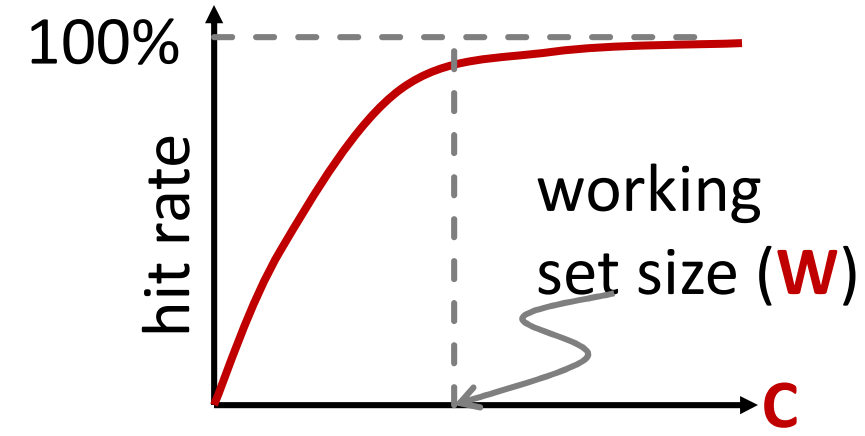
Compulsory Miss

- First reference to a block address always misses (if no prefetching)
- Dominates when locality is poor
  - for example, in a "streaming" data access pattern where many addresses are visited, but each is used only once
- Main design factor: **B** and "prefetching"



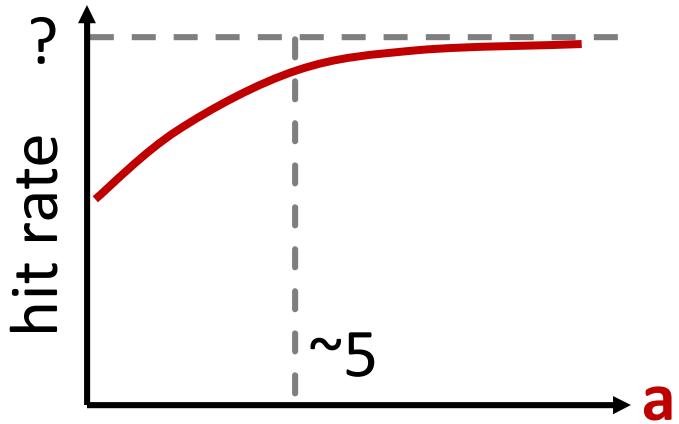
Capacity Miss

- Cache is too small to hold everything until reuse
- Defined as non-compulsory misses that would occur in a fully-associative cache of the same **C** and **B** using optimum (Belady) replacement
- Dominates when **C** < **W**
  - for example, the L1 cache usually not big enough due to cycle-time tradeoff
- Main design factor: **C**



Conflict Miss

- Miss to a previously visited block address displaced due to conflict under direct-mapped or set-associative allocation
- Defined as "a miss that is neither compulsory nor capacity"
- Dominates when **C≈W** or when **C/B** is small
- Main design factor: **a**



3'C worksheet: a=1, B=1, C=2, G=1

addr	set#	which C?	set[2]	F.A. + Belady
0x0	0	compulsory	[-,-] → [0,-]	{ } → {0}
0x2	0			
0x0	0			
0x2	0			
0x1	1			
0x0	0			
0x2	0			
0x0	0			

3'C worksheet: a=1, B=1, C=2, G=1

addr	set#	which C?	set[2]	F.A. + Belady
0x0	0	compulsory	[-,-] → [0,-]	{ } → {0}
0x2	0	compulsory	[0,-] → [2,-]	{0} → {0,2}
0x0	0	conflict	[2,-] → [0,-]	$\{0,2\}_{hit}$
0x2	0	conflict	[0,-] → [2,-]	$\{0,2\}_{hit}$
0x1	1	compulsory	[2,-] → [2,1]	{0,2} → {0,1}
0x0	0	conflict	[2,1] → [0,1]	$\{0,1\}_{hit}$
0x2	0	capacity	[0,1] → [2,1]	{0,1} → {0,2}
0x0	0	conflict	[2,1] → [0,1]	$\{0,2\}_{hit}$

Recap: Basic Cache Parameters

ISA

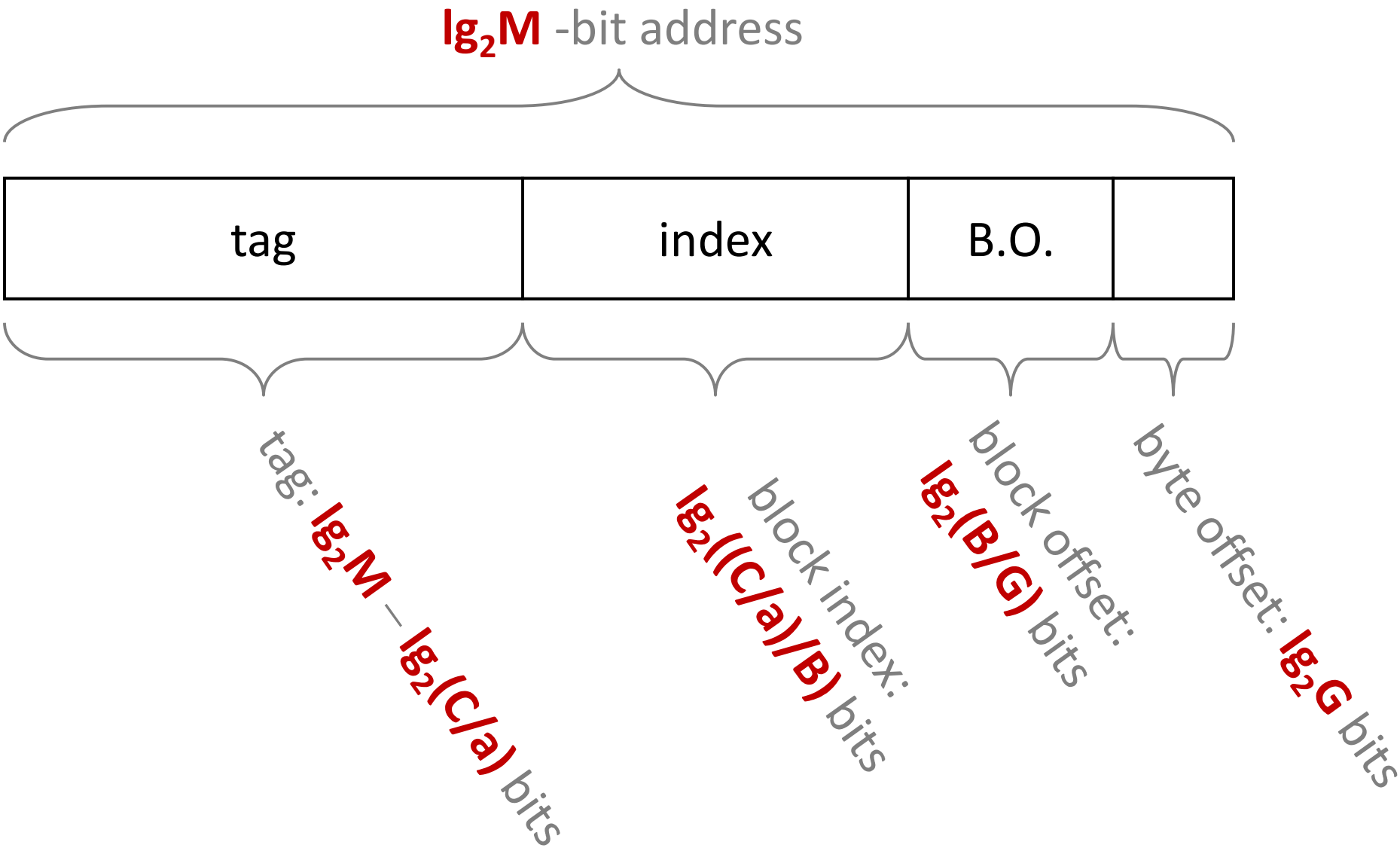
- **M = 2^m** : size of address space in bytes
  example values: 2^{32}, 2^{64}
- **G=2^g** : cache access granularity in bytes
  example values: 4, 8

Implementation

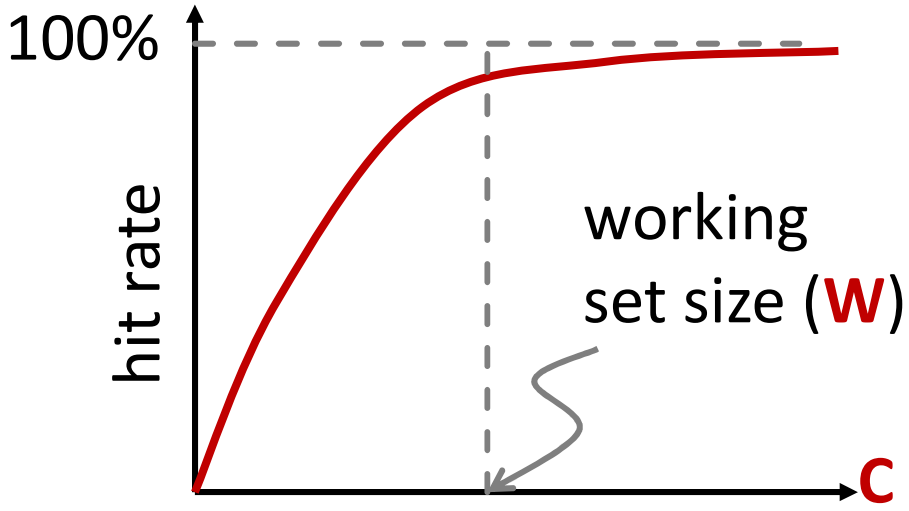
- **C** : "capacity" of cache in bytes
  example values: 16 KByte (L1), 1 MByte (L2)
- **B = 2^b**: "block size" in bytes
  example values: 16 (L1), >64 (L2)
- **a**: "associativity" of the cache
  example values: 1, 2, 4, 5(?),... "C/B"
- "map": addr to idx

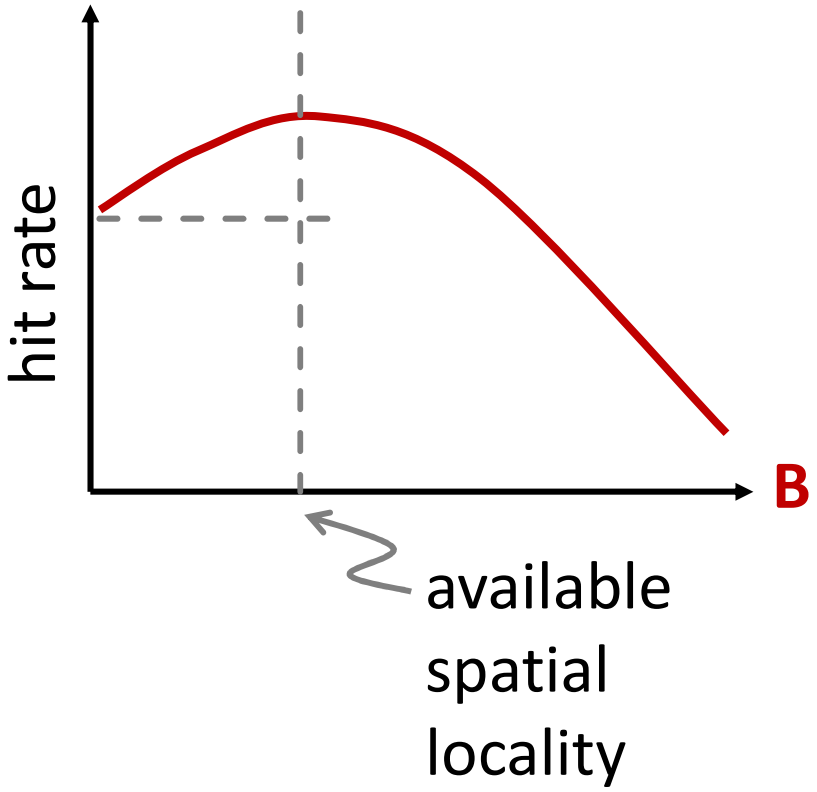
C/a should be a 2-power

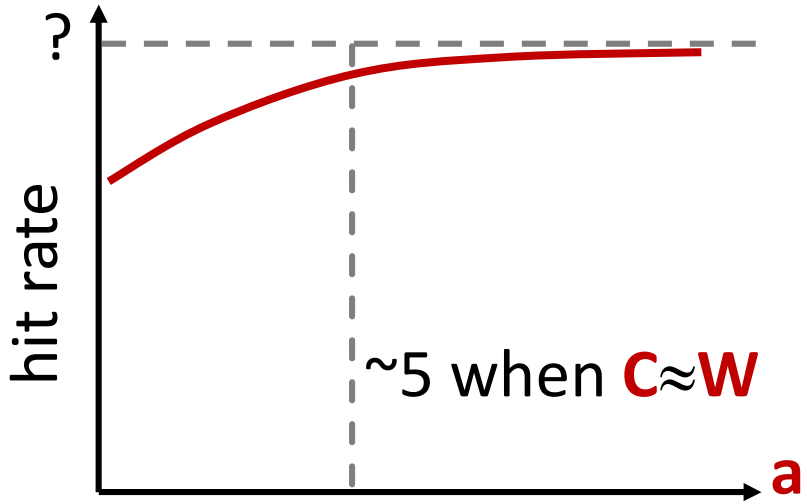
Recap: Address Map for Typical Locality



Recap: aBC Rule of Thumb Cribsheet



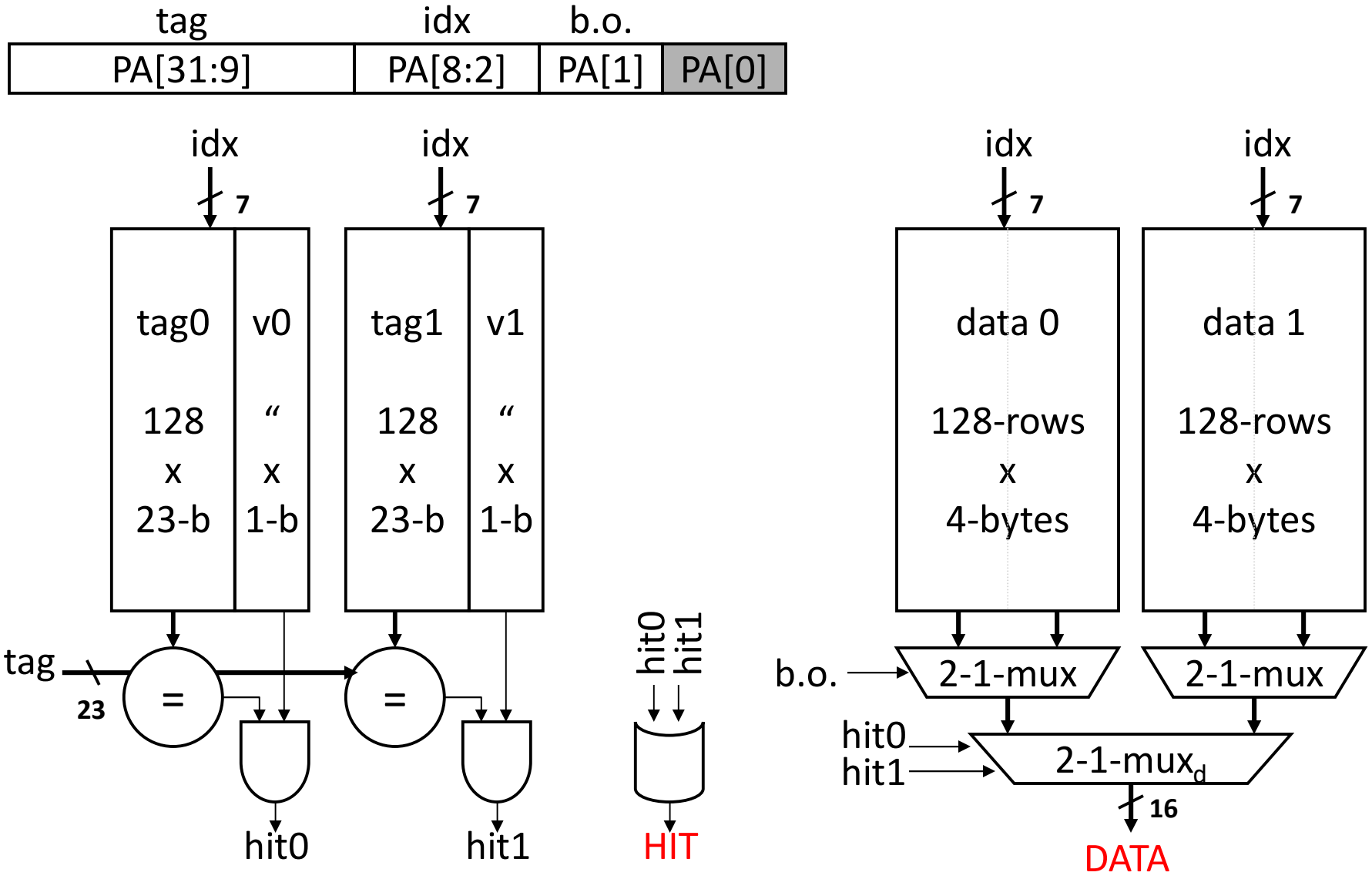




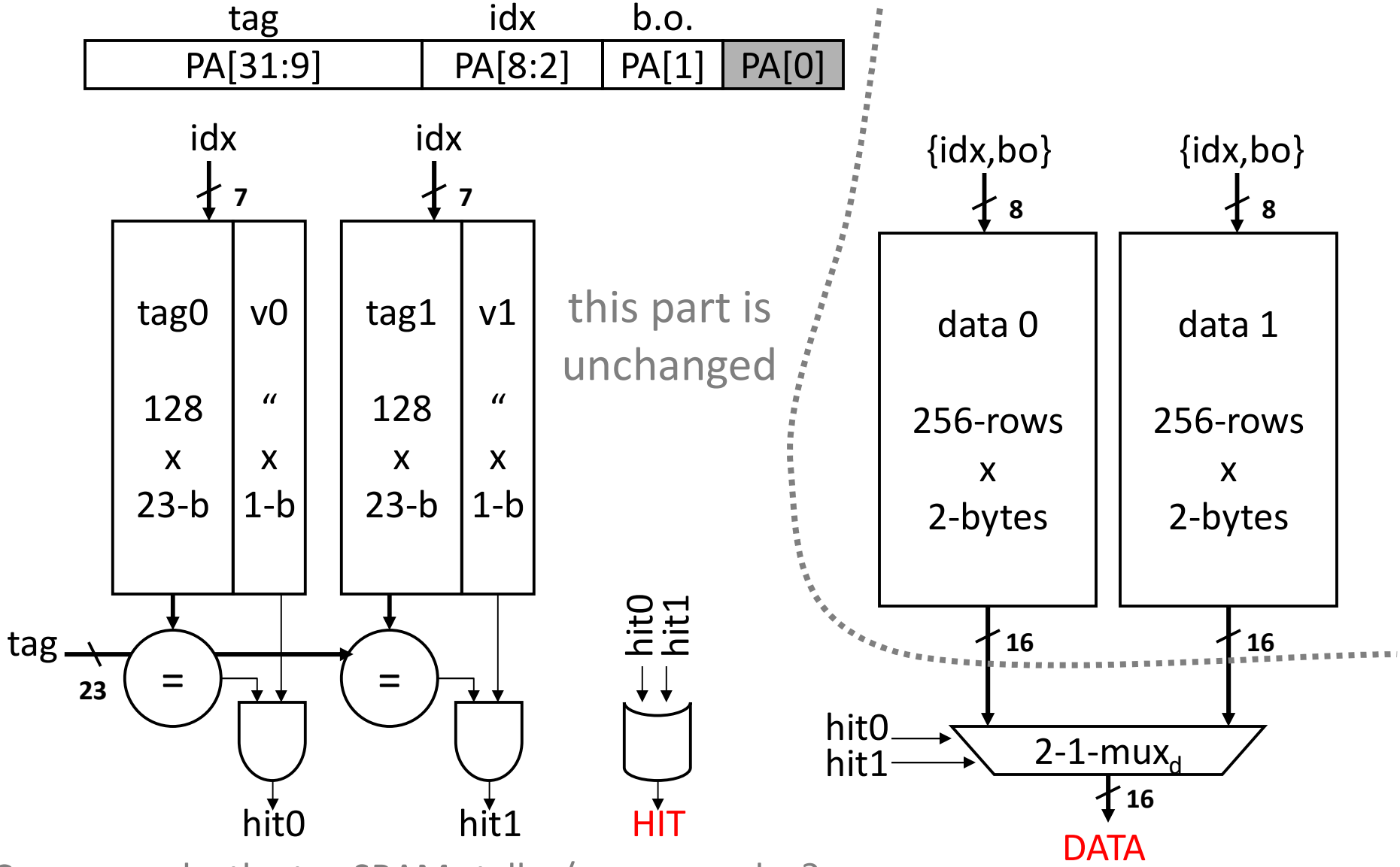
For "typical" programs

$M=2^{32}$, $a=2$, $C=1K$, $B=4$, $G=2$

M=2^{32}, a=2, C=1K, B=4, G=2: "textbook" solution

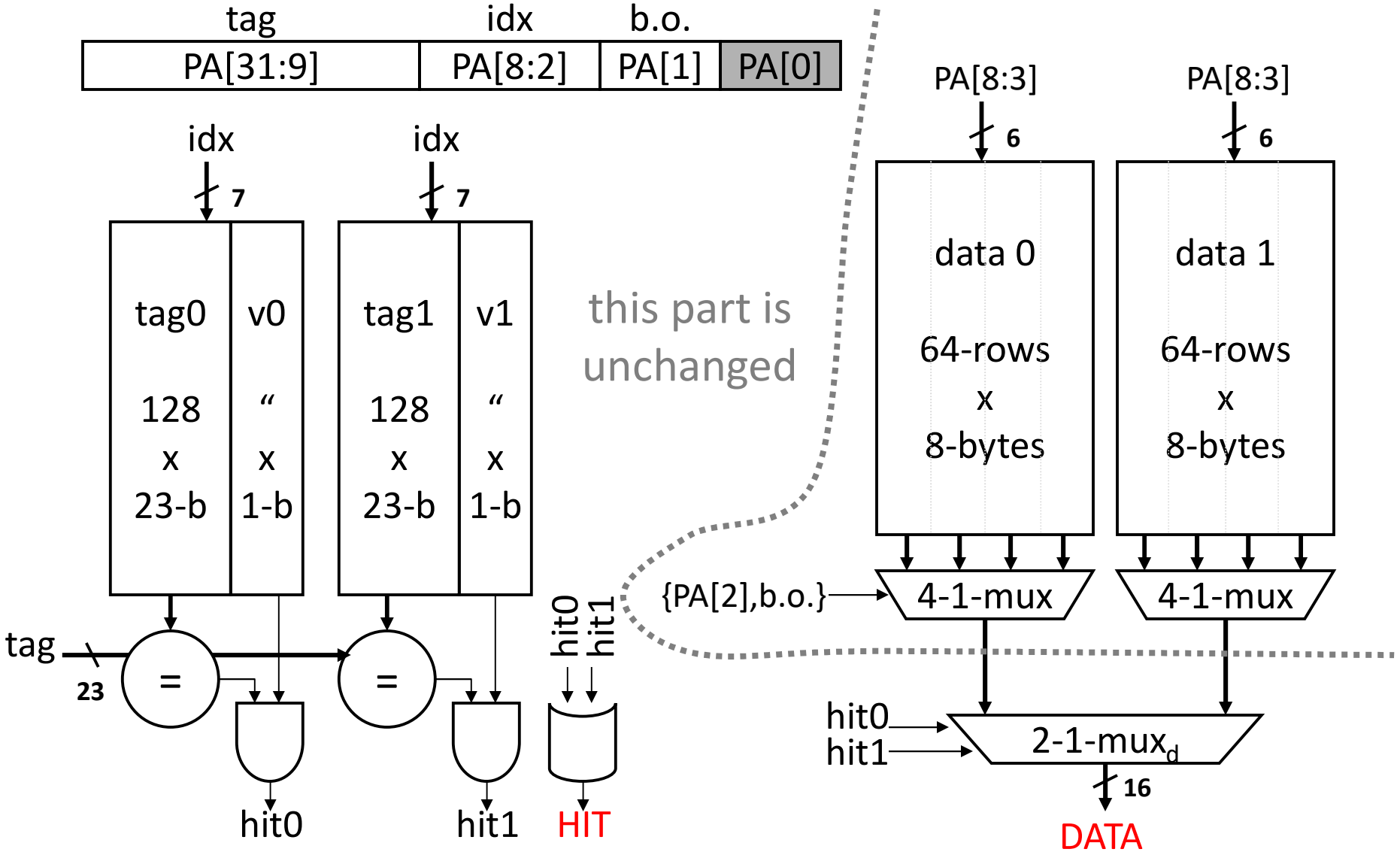


Same cache parameters
but tune for "narrower" data SRAM banks



Can you make the tag SRAMs taller/narrower also?

Same cache parameters but tune for "fatter" data SRAM banks



Can you make the tag SRAMs shorter/wider also?

Same cache parameters but each block frame is interleaved over 2 SRAM banks