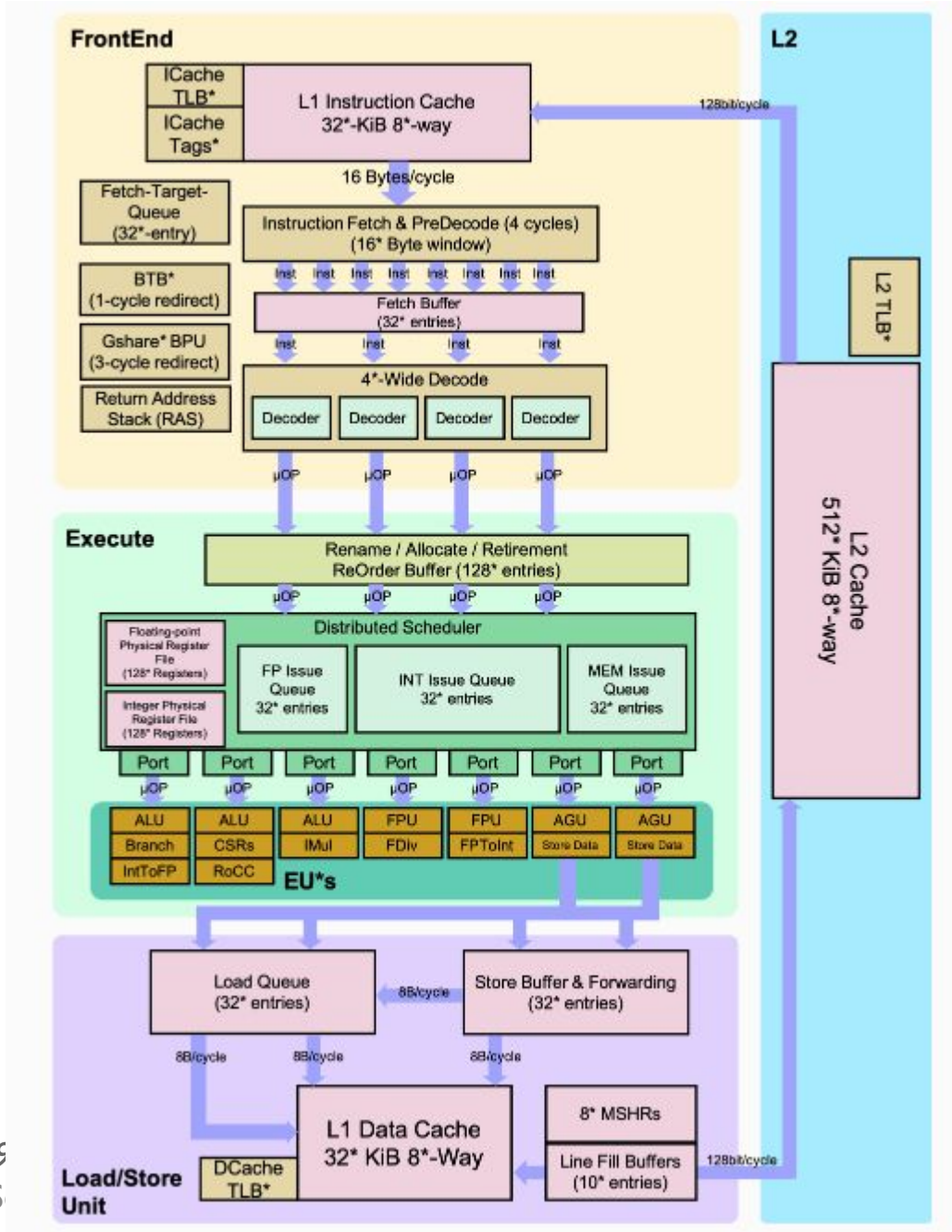
Module 6 – Memory Sybsystem

(ECE 209) Secure and Advanced Computer Architecture

Nader Sehatbakhsh
Department of Electrical and Computer Engineering
University of California, Los Angeles

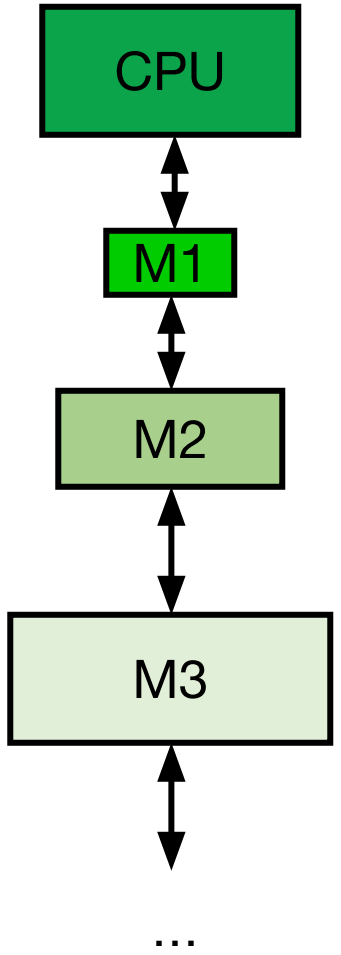
BOOM Core

- Out-of-order execution



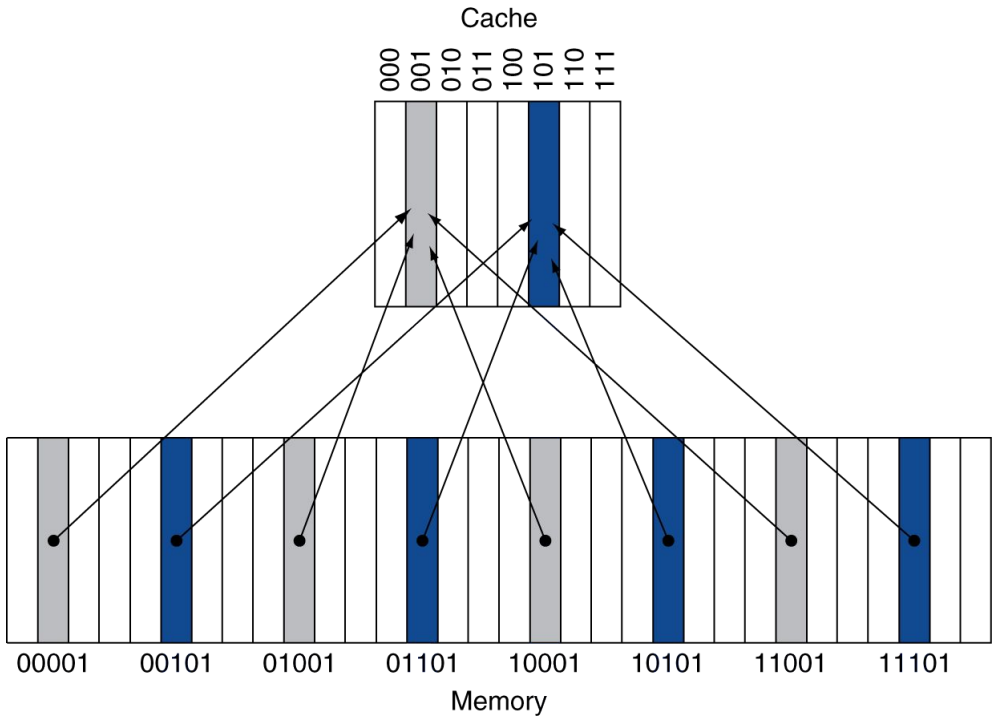
Memory Hierarchy

★ ***Idea:*** *have multiple layers of memory to balance between size and speed!*

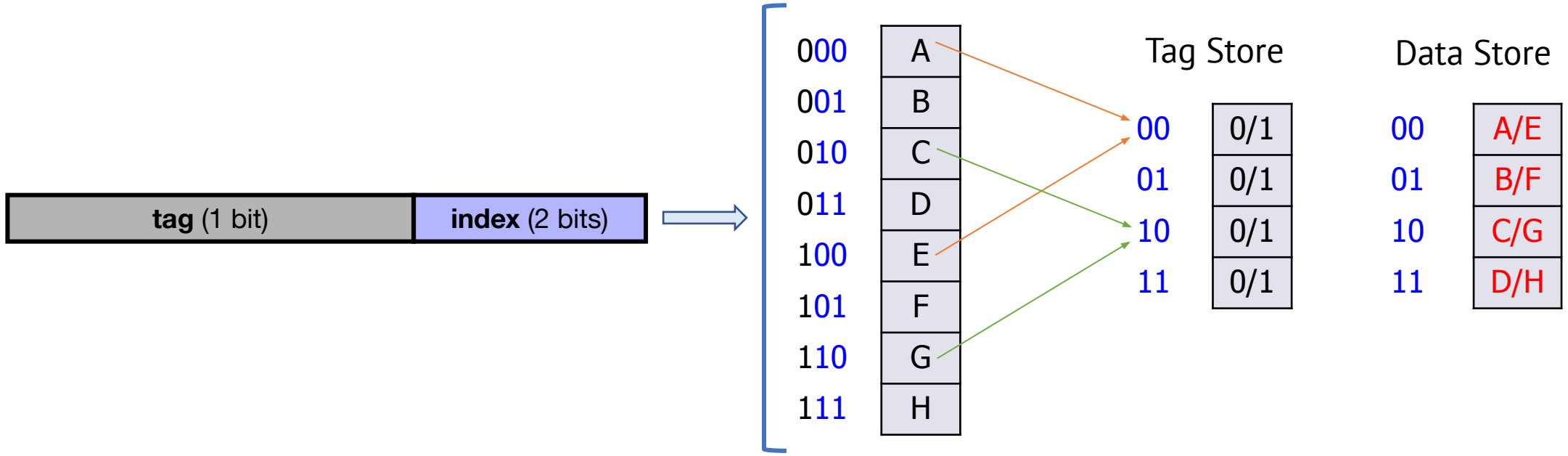


Reality: Cache << Memory

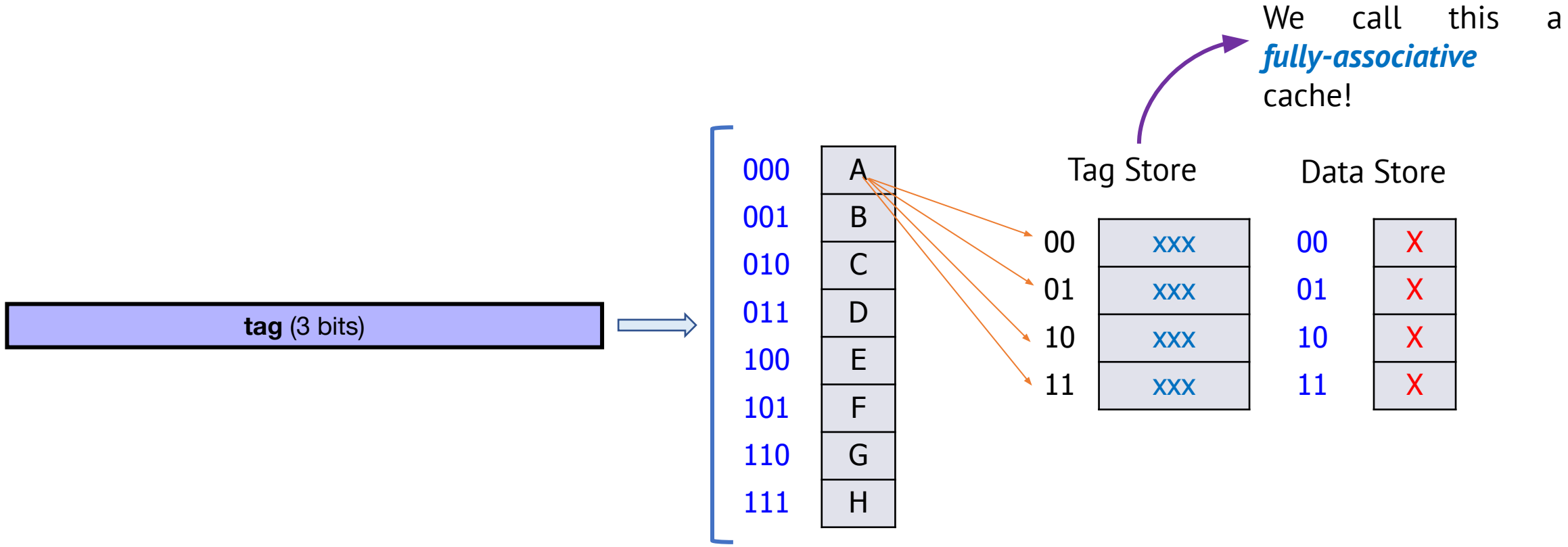
- ***One*** cache row is assigned to ***many*** memory lines.



Direct-Mapped



Alternative to Direct-Mapped?



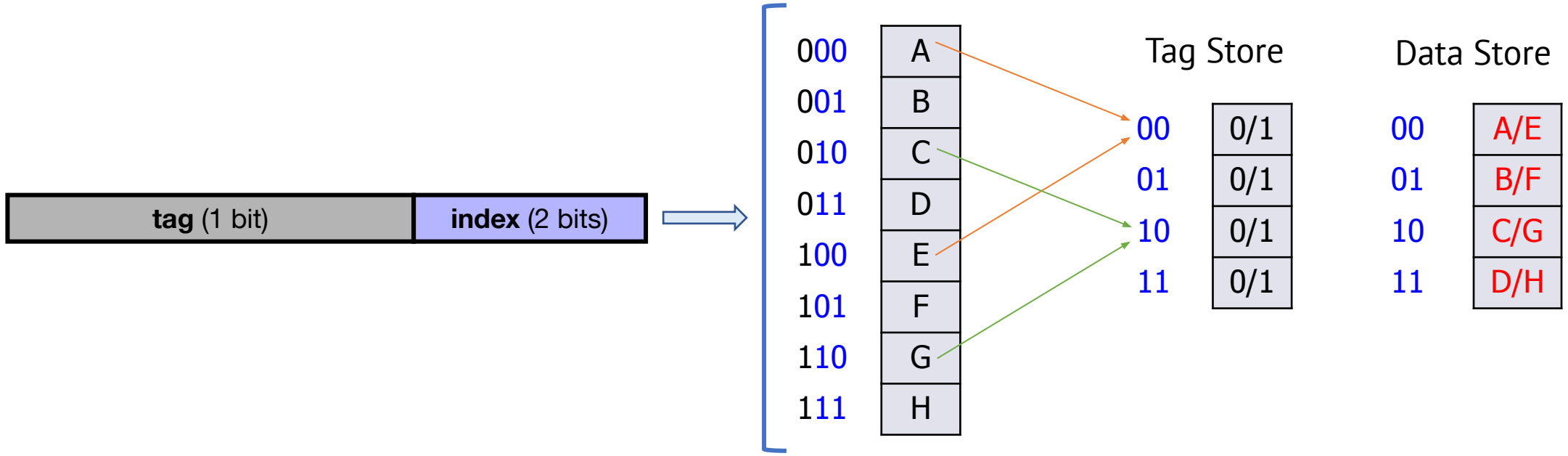
Can we have ***both***?

Can we have ***both***?

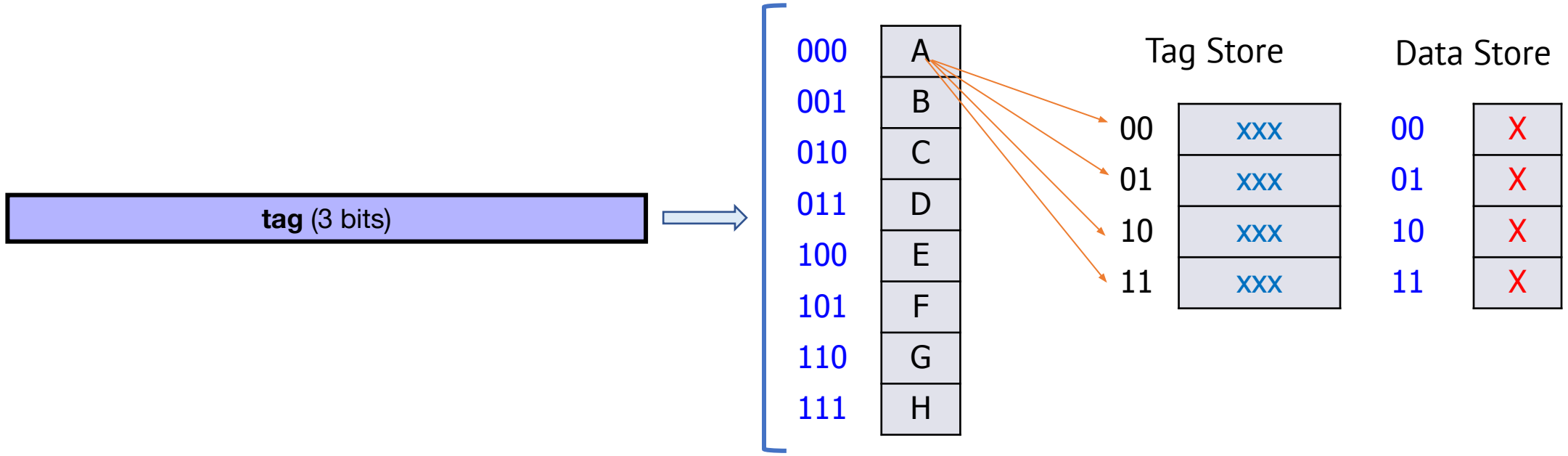
★ *Set-associative cache*

- *Add associativity within each set!*

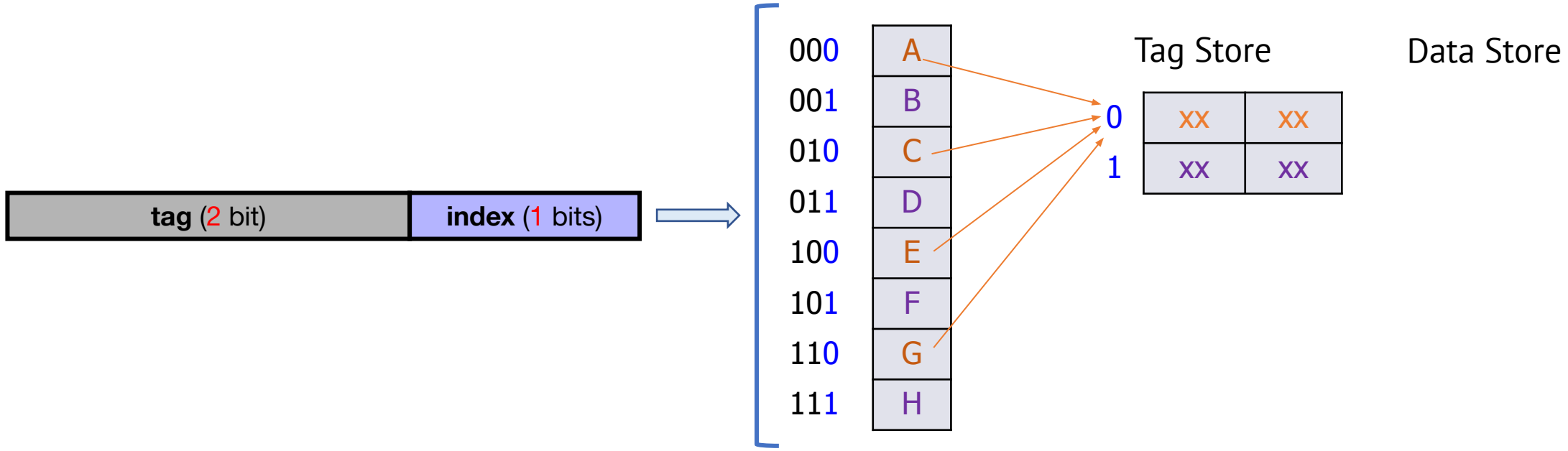
Recall: DM


tag (1 bit)
index (2 bits)
000
001
010
011
100
101
110
111
A
B
C
D
E
F
G
H
Tag Store
00
01
10
11
0/1
0/1
0/1
0/1
Data Store
00
01
10
11
A/E
B/F
C/G
D/H

Recall: FA

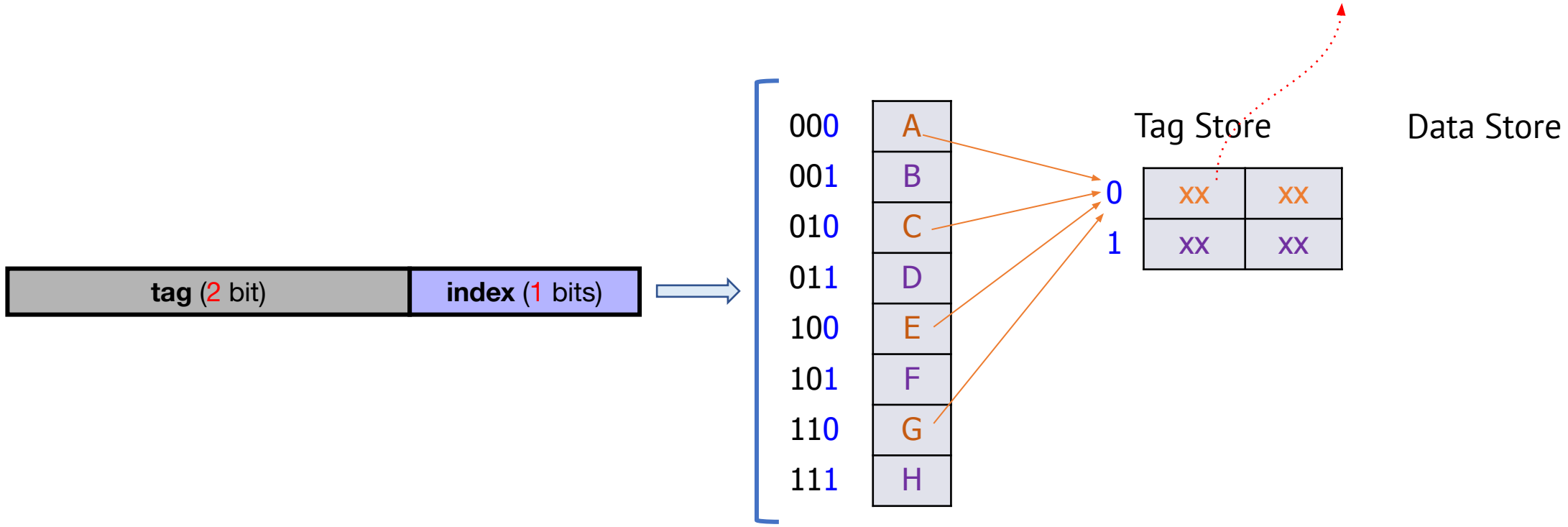

tag (3 bits)
000
001
010
011
100
101
110
111
A
B
C
D
E
F
G
H
Tag Store
00
01
10
11
xxx
xxx
xxx
xxx
Data Store
00
01
10
11
X
X
X
X

Set-Associative Cache



Set-Associative Cache

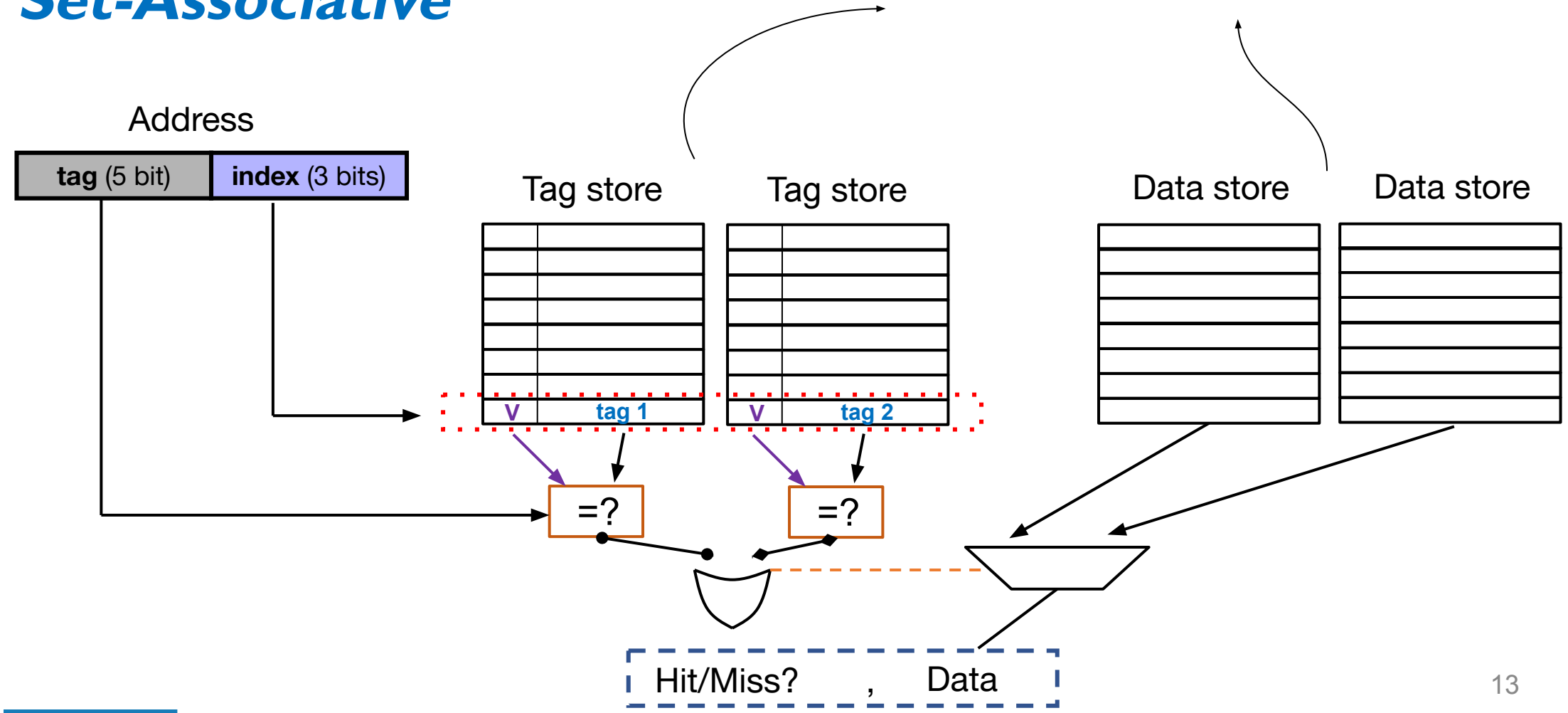
We call each column ***way***. This cache is 2-way set-assoc.



Cache Lookup

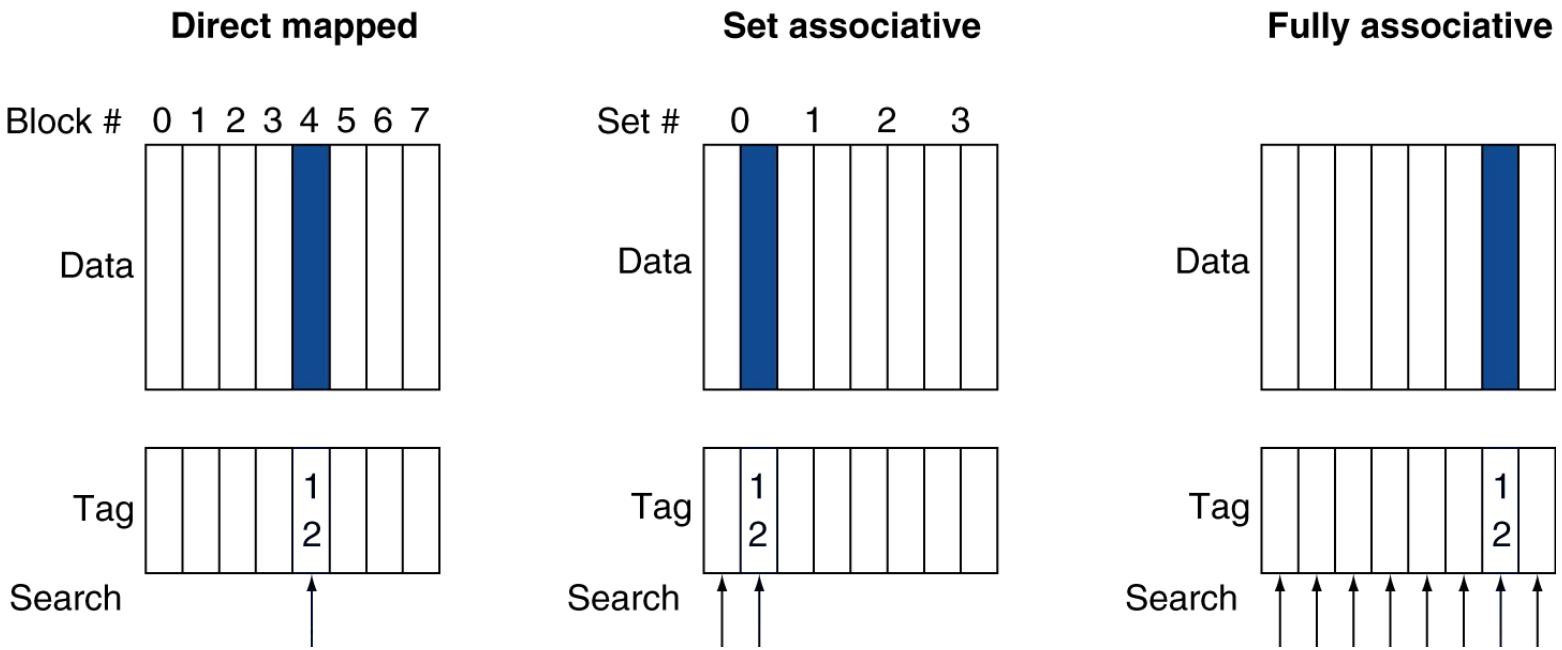
Set-Associative

To improve latency, ***all*** tag searches and ***all*** data accesses happen ***parallel***



Nader Sehatbakhsh <nsehat@ee.ucla.edu>

Associative Cache



Nader Sehatbakhsh <nsehat@ee.ucla.edu>

Direct-Mapped vs. Fully Associative vs. Set Associative

- Average Access Time = HitTime + MissRate × MissPenalty
- HitTime
- MissRate
- Miss Penalty

Direct-Mapped vs. Fully Associative vs. Set Associative

- Average Access Time = HitTime + MissRate × MissPenalty
- HitTime
  DM < SA << FA
- MissRate
  DM >> SA > FA
- Miss Penalty
  DM == FA == SA

What to do when cache is full?

Replacement Policy

- We will talk about this next week!

Cache Performante

- Average Access Time = HitTime + MissRate × MissPenalty

★ *How to optimize this?*

Reducing Miss Rate

Reducing Miss Rate

increase block size

- Increase Block Size
  - Bring more than one byte per access → Exploit more *spatial locality*
  - Think about the library example!
  - Arrays, integers, etc.

Cache Block

- Instead of storing 1 byte per row, we can store a block with multiple bytes.
  - Every time we need to load something to the cache, we load it at block level.

Cache Block

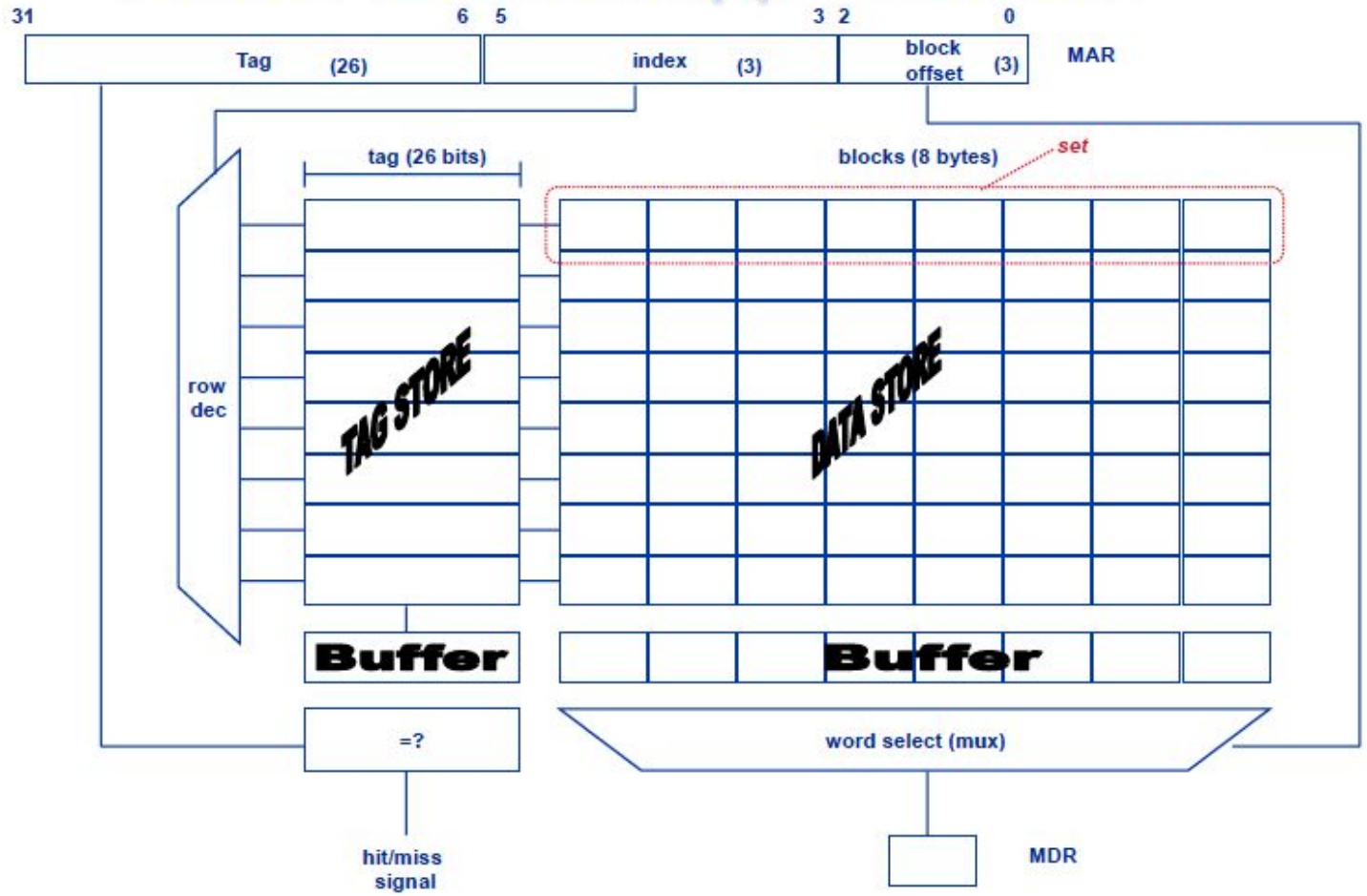
- Instead of storing 1 byte per row, we can store a block with multiple bytes.
  - Every time we need to load something to the cache, we load it at block level.
  - We still send things to the CPU at byte level.
    - How to do that? → We need *block offset* to decide which byte within the block should be selected!
  - How big a block should be?
    - It depends! Typically, somewhere between 8-64B.

Cache Block

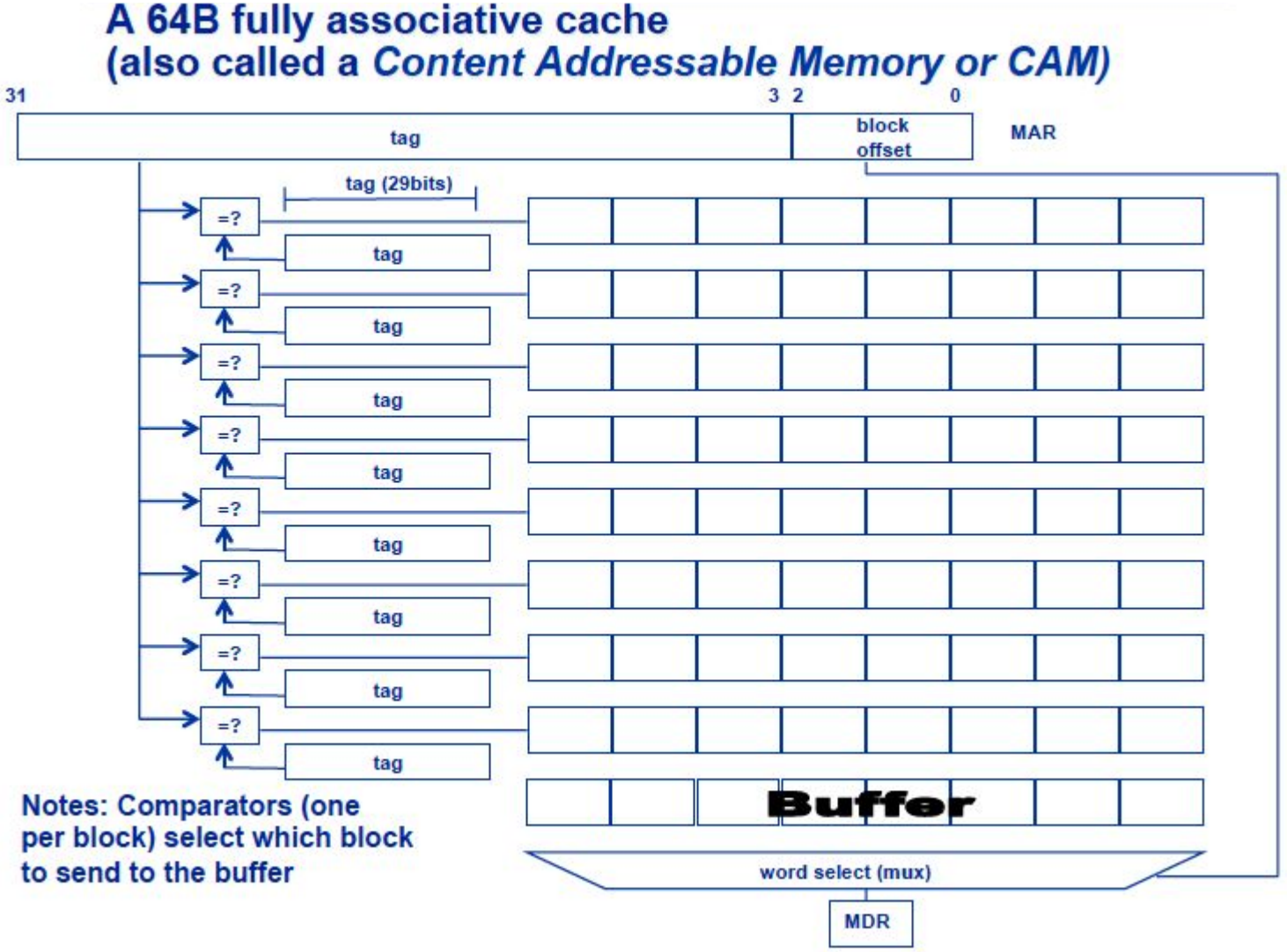
- Instead of storing 1 byte per row, we can store a block with multiple bytes.
  - Every time we need to load something to the cache, we load it at block level.
  - We still send things to the CPU at byte level.
    - How to do that? → We need *block offset* to decide which byte within the block should be selected!
  - How big a block should be?
    - It depends! Typically, somewhere between 8-64B.

- Cache Address = {tag, index, block offset}

A 64B direct mapped cache

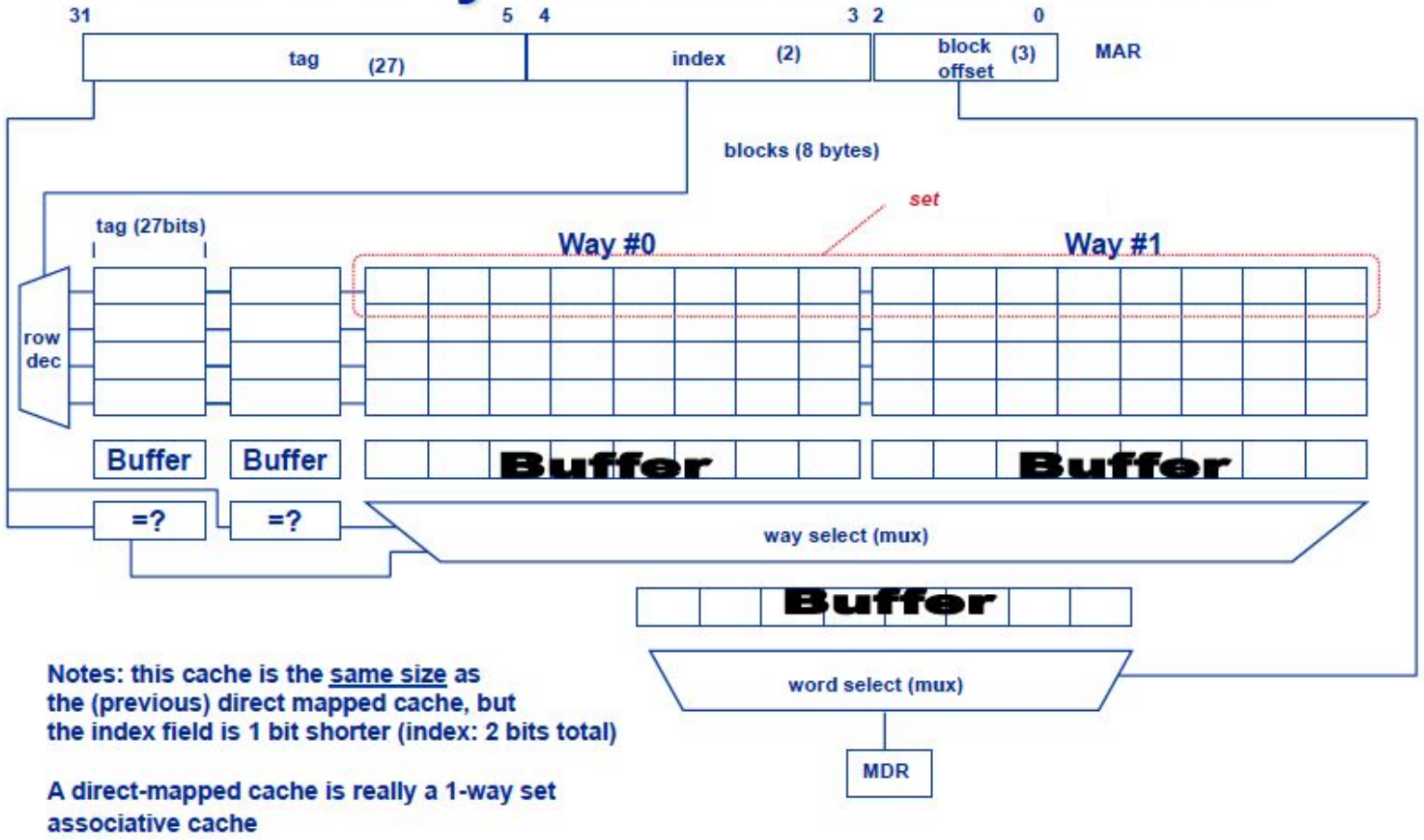


Picture is borrowed from CS4290, Tom Conte (Georgia Tech).



Picture is borrowed from CS4290, Tom Conte (Georgia Tech).

A 64B 2-way set associative cache



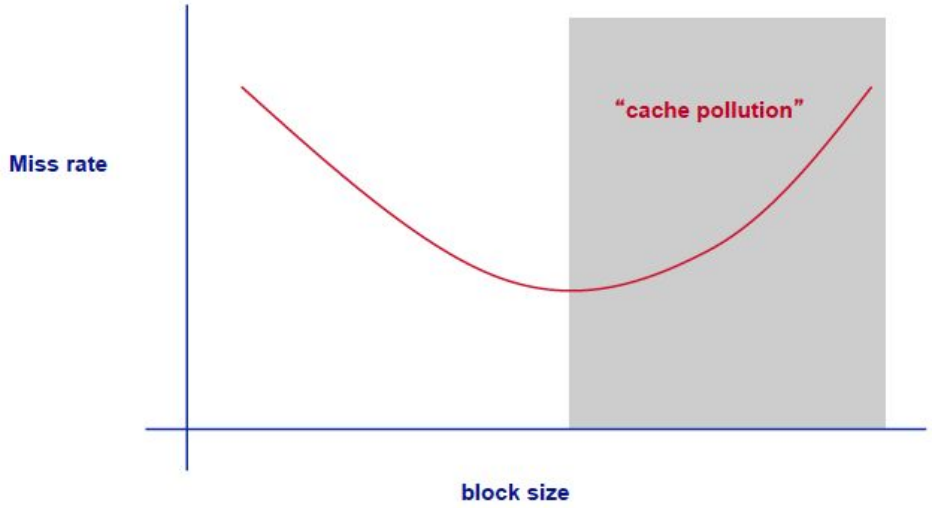
Picture is borrowed from CS4290, Tom Conte (Georgia Tech).

How big a block should be?

- If bringing more is helpful, why don't we have very large blocks?

Reducing Miss Rate
increase block size

- Increase Block Size
  - Exploit more *spatial locality*
  - Large block size brings too many useless data *(called cache pollution).*
  - Increase miss penalty
    (have to bring more in on a miss)



Reducing Miss Rate

increase associativity

- FA > SA > DM

Reducing Miss Rate

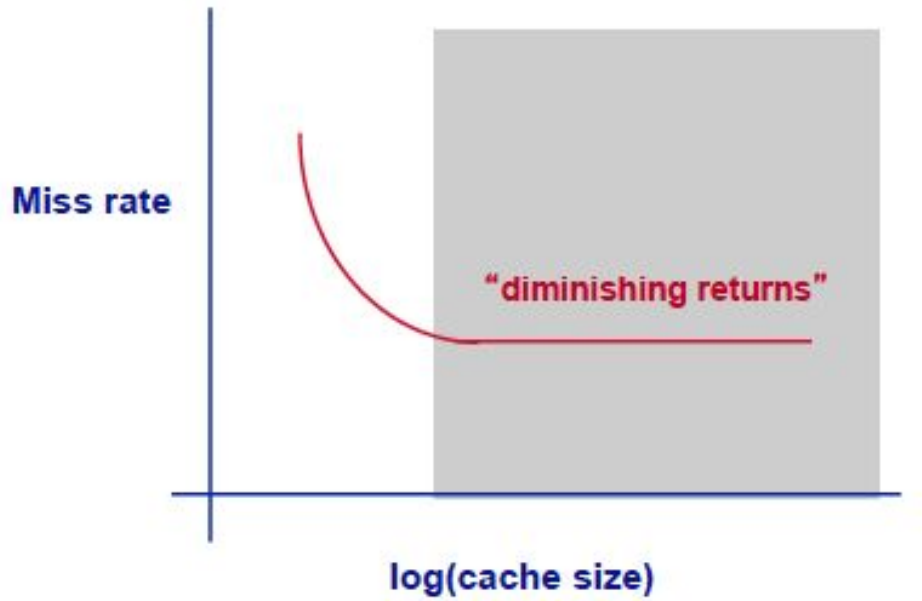
increase associativity

- FA > SA > DM
- More ways → higher hit time (diminishing returns)

-- How many ways is good enough?

- 8-way is as good as FA

(after that, the limit is capacity miss).



Reducing Miss Rate

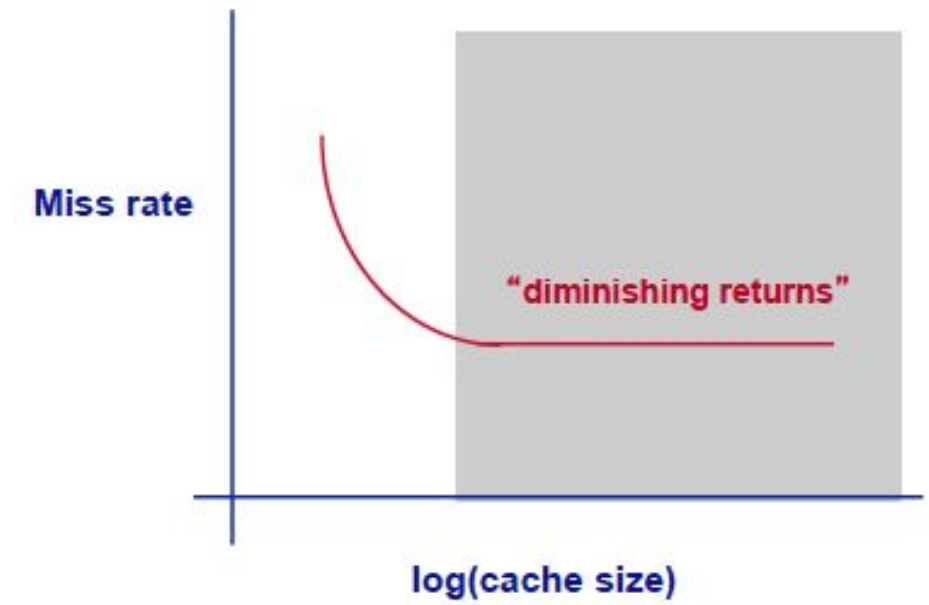
increase cache size

- Larger cache → more data!

Reducing Miss Rate

increase cache size

- Larger cache → more data!
- Larger cache → slower hit time
- Diminishing returns:
  - For a large cache
    double size != double performance
    (the limit becomes compulsory misses!)



Reducing Miss Rate

prefetching

★ Idea: *if we can guess the access pattern, we can bring data before it's needed!*

i, i+1, i+2, i+3, …

Reducing Miss Rate
prefetching

★ Idea: *if we can guess the access pattern, we can bring data before it's needed!*

- Reduce compulsory misses
- Cache pollution
  - Need to monitor prefetching accuracy to change its *aggressiveness.*
  - Other than this, no other negative impacts! No correctness issues!

i, i+1, i+2, i+3, …

Prefetching - Four Questions!

- What
  - What addresses to prefetch (i.e., address prediction algorithm)
- When
  - When to initiate a prefetch request (early, late, on time)
- Where
  - Where to place the prefetched data (caches, separate buffer)
- How
  - How does the prefetcher operate and who operates it (software, hardware, hybrid)

Software vs. Hardware Prefetch

- Software prefetching
  - ISA provides prefetch instructions
  - Programmer or compiler inserts prefetch instructions (effort)
  - Usually works well only for "regular access patterns"

- Hardware prefetching
  - Hardware monitors processor accesses
  - Memorizes or finds patterns/strides
  - Generates prefetch addresses automatically

Examples: Software

```
for (i=0; i<N; i++) {
__prefetch(a[i+8]);
__prefetch(b[i+8]);
sum += a[i]*b[i];
}

while (p) {
__prefetch(pnext);
work(pdata);
p = pnext;
}
```

Example: Hardware

- Next line prefetcher:
  - Always prefetch next N cache lines after a demand access
  - Tradeoffs:
    - Simple to implement.
    - No need for sophisticated pattern detection
    - Works well for sequential/streaming access patterns (instructions?)
    - Can waste bandwidth with irregular patterns
    - Low prefetch accuracy if access stride = 2 or when the program is traversing memory from higher to lower addresses

Example: Hardware

- Next line prefetcher:
  - Always prefetch next N cache lines after a demand access
  - Tradeoffs:
    - Simple to implement.
    - No need for sophisticated pattern detection
    - Works well for sequential/streaming access patterns (instructions?)
    - Can waste bandwidth with irregular patterns
    - Low prefetch accuracy if access stride = 2 or when the program is traversing memory from higher to lower addresses

★ *Better options?* stride prefetcher, stream buffers, etc.

Reducing Miss Rate
prefetching

★ Idea: *if we can guess the access pattern, we can bring data before it's needed!*

- Reduce compulsory misses
- Cache pollution
  - Need to monitor prefetching accuracy to change its *aggressiveness.*
  - Other than this, no other negative impacts! No correctness issues!

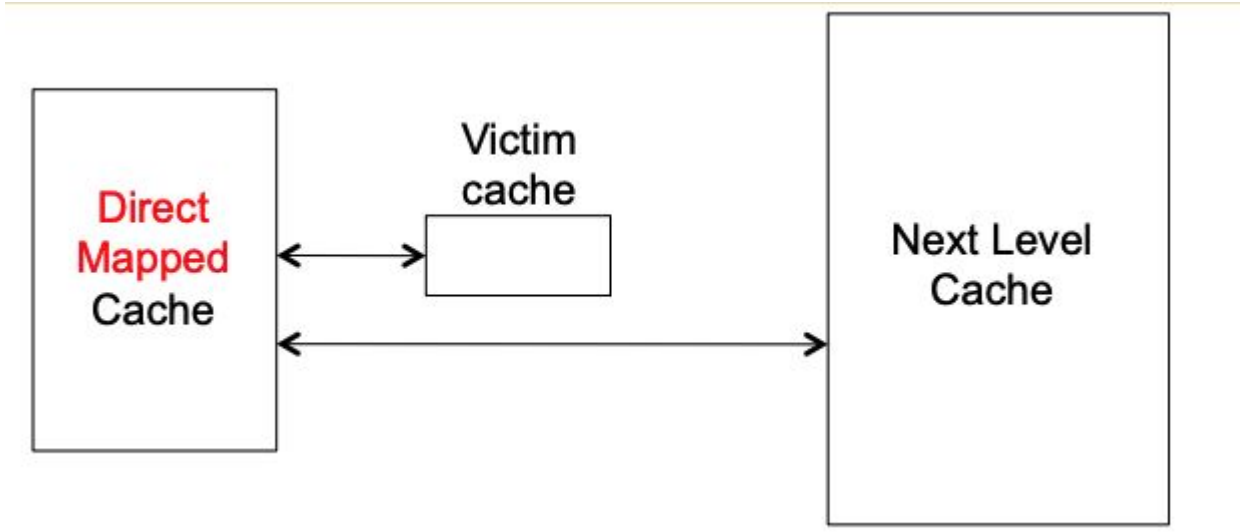
i, i+1, i+2, i+3, …

Reducing Miss Rate

victim cache

★ Idea: *for heavily conflicting addresses, a few "extra" temporary sets could remove conflicts!*

- *Use a very small buffer (called victim cache) to save the recently discarded blocks. Search through them as well.*



Reducing Miss Rate

victim cache

★ Idea: *for heavily conflicting addresses, a few "extra" temporary sets could remove conflicts!*

- *Use a very small buffer (called victim cache) to save the recently discarded blocks. Search through them as well.*

- Reduce conflict misses
  - Research shows a 4-entry victim cache can remove up to 90% of conflicts.
- Extra overhead.
- More complex design.

Reducing Miss Rate

compiler and software

- Re-order accesses/arrays to increase locality.
- Combine loops with similar behavior.
- Use "tiling" to access arrays region by region instead of whole.
- Use compiler profiling to improve prefetching.
- …

Tiling

- If column-major
  - `x[i+1,j]` follows `x[i,j]` in memory
  - `x[i,j+1]` is far away from `x[i,j]`

Poor code

```
for     i     =     1,     rows
  for    j     =    1,    columns
    sum = sum + x[i,j]
```

Better code

```
for     j     =     1,     columns
  for    i     =    1,    rows
    sum = sum + x[i,j]
```

Reducing Miss Rate

replacement policy

- LRU vs. PLRU vs. Random

- Storage vs. accuracy tradeoff!

How to reduce miss penalty?

Reducing Miss Penalty
write buffer

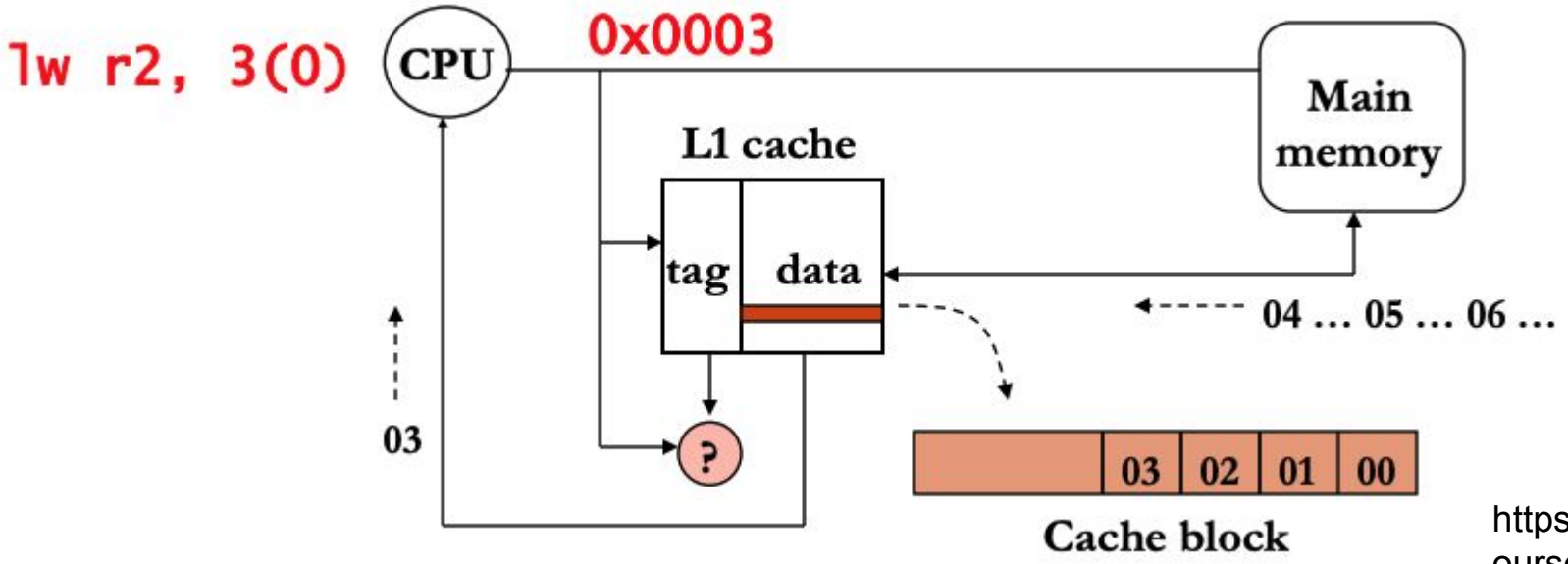
- *Use load-store queue.*

- No wait for stores needed.
- Lower miss penalty for loads.
- More overhead.

Reducing Miss Penalty

early restart

- *Instead of waiting for all bytes (in a block) to arrive, forward data to CPU as soon as the requested byte(s) arrives.*



https://www.inf.ed.ac.uk/teaching/courses/car/Notes/2013-14/lecture07a-cache_performance.pdf

Reducing Miss Penalty

early restart with critical word first

- *Instead of waiting for all bytes (in a block) to arrive, forward data to CPU as soon as the requested byte(s) arrives.*

- *To further optimize this, first read the requested byte!*

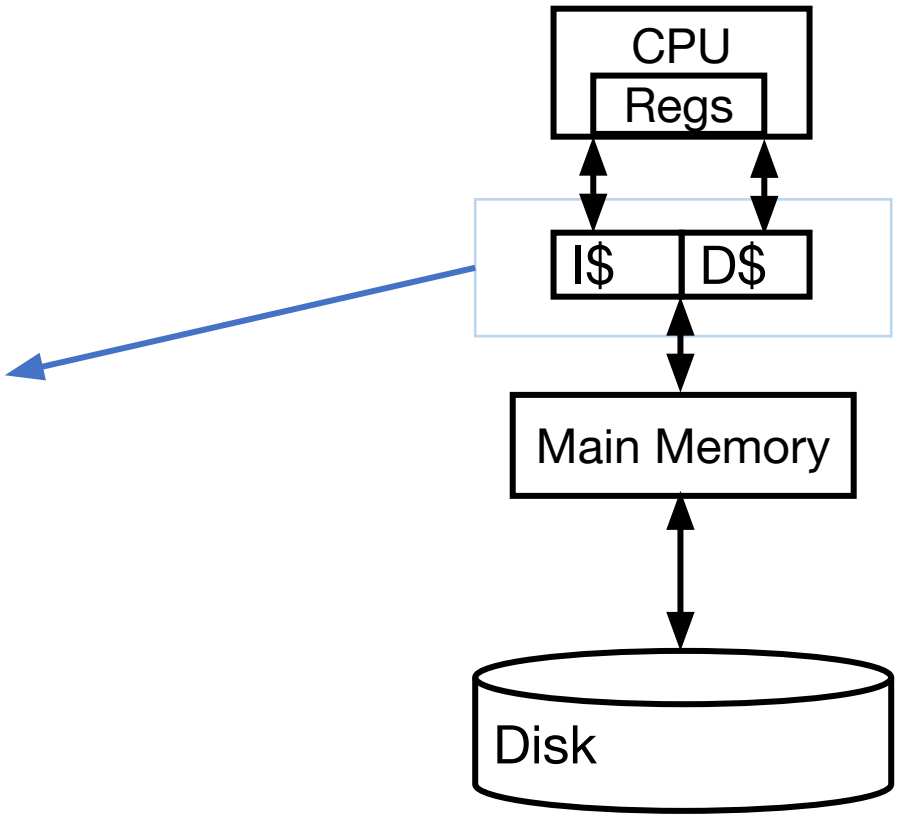
Reducing Miss Penalty

add more levels

- *Adding L2, L3, etc. to further reduce miss penalty!*

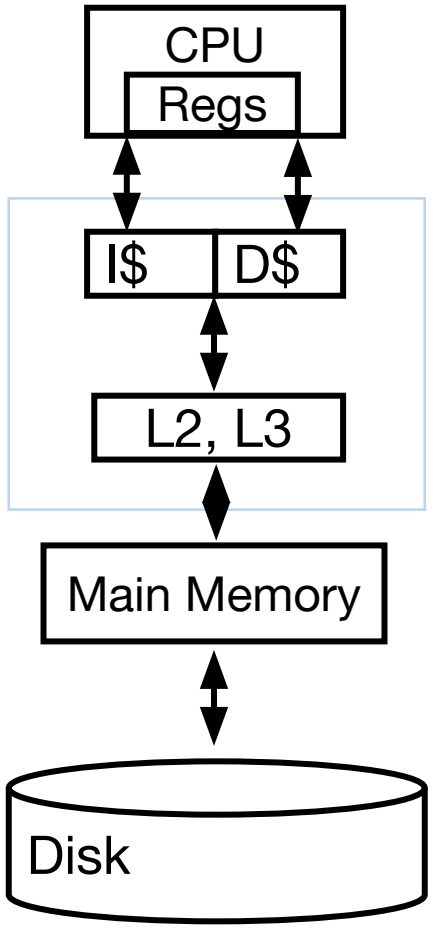
Cache

Storing temporary data close to the CPU.



Multi-Level Cache

Higher level → bigger but slower
(still both smaller and faster than the main memory.)



Cache Performance

- Average time to access the cache:

  AAT = HitTime + MissRate × MissPenalty

  - HitTime: Time it takes to access the (L1) cache.
  - MissRate: The average frequency of misses (in L1).
  - MissPenalty: The time required to access the main memory.

Cache Performance

- Average time to access the cache:

  AAT = HitTime + MissRate × MissPenalty

  - HitTime: Time it takes to access the (L1) cache.
  - MissRate: The average frequency of misses (in L1).
  - MissPenalty: The time required to access the main memory.

- *What if there are multiple levels?*

Cache Performance

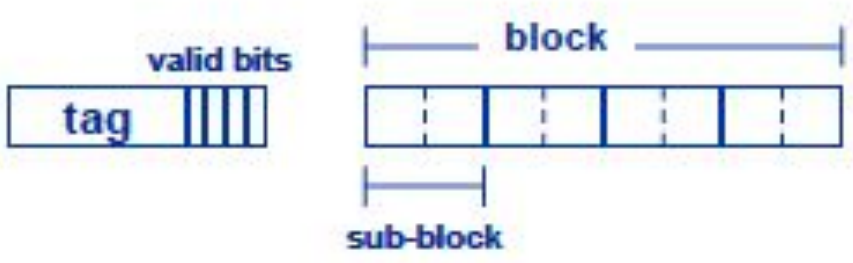
multi-level

AAT = HitTime + MissRate × MissPenalty

Reducing Miss Penalty
sub-blocking

- Higher block size improves miss rate but increase miss penalty!

★ ***Idea:*** keep a large block size, but divide it into smaller *"subblocks"*. Bring only a subset of subblocks on a miss.



Reducing Miss Penalty
sub-blocking

- Subblocks:
  - Lower miss rate.
  - Lower miss penalty.
  - Need separate storage for valid bits for each subblock.
  - More complex circuitry.

How to reduce hit time?

Reducing Hit Time

reduce associativity and size

- DM < FA
  - ➔ Use SA to balance between the two

- Use smaller cache in lower levels (L1, L2, …)

Reducing Hit Time

parallel lookup

- Access tag and data in parallel.

- Access each way in parallel.

Reducing Hit Time

speculative load

- *Instead of waiting for a store (potentially conflicting), issue the load speculatively.*
  - Once store is resolved, check whether there was a conflict or not. Recover if there was.

Summary

- Miss Rate
  - Increase block size
  - Increase associativity
  - Increase cache size
  - Prefetching
  - Victim cache
  - Compiler
  - Replacement Policy
- Miss Penalty
  - Write buffer
  - Early restart with critical block first
  - Adding more levels
  - Sub-blocking
- Hit Time
  - Set associative cache
  - Add more levels
  - Parallel lookup
  - Speculative loads

Background

- Watch Lectures 12 and 13 for M116

End of Presentation

UCLA Samu
School of E

Acknowledgement

- This course is partly inspired by the following courses created by my colleagues:
  - 6.888 MIT (Yan)
  - CS 598 UIUC (Fletcher)
  - CS 7290 Georgia Tech (Kim)
  - ECE752 Wisconsin (Lipasti)