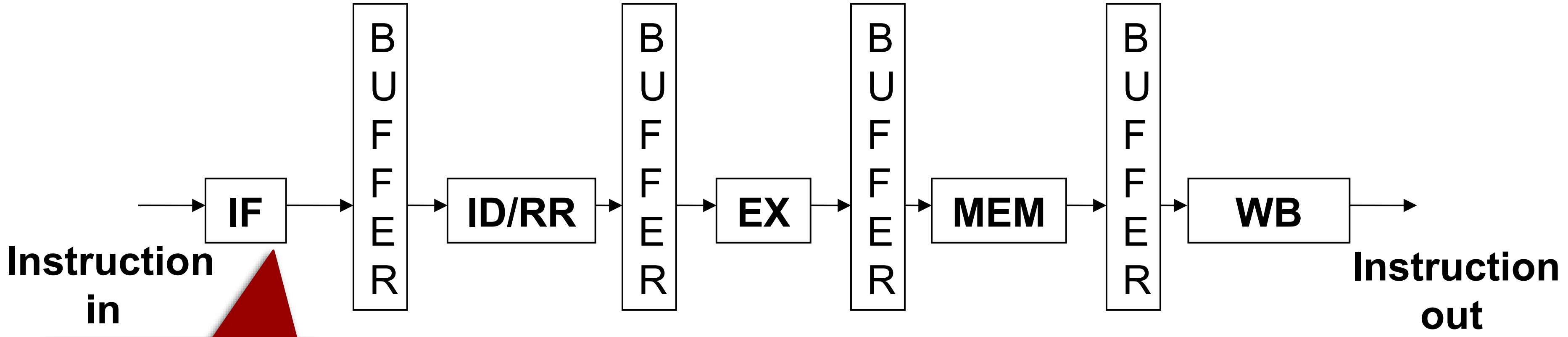
Memory Hierarchy

Chapter 9, Ramachandran and Leahy

Recall: In a paged memory system

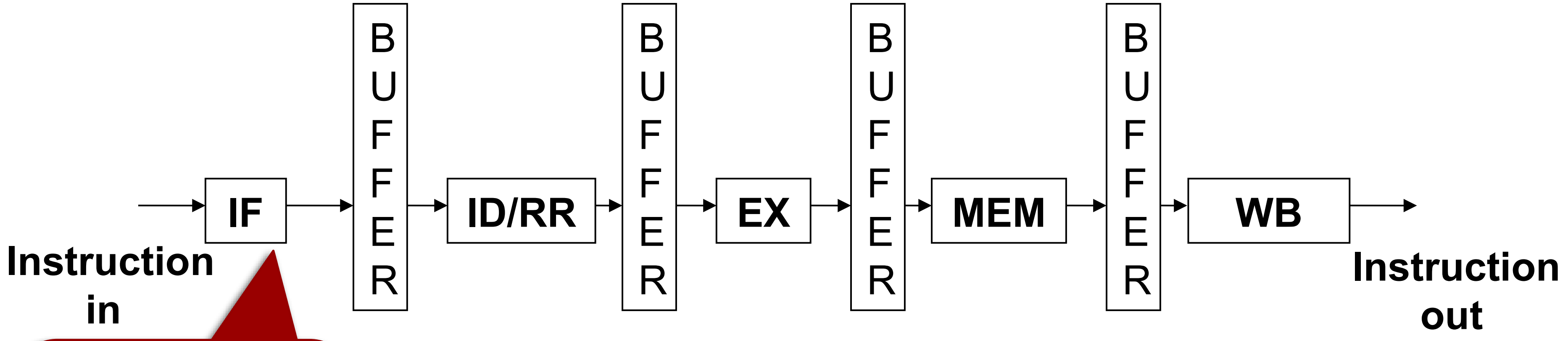


Two memory accesses for every instruction:
- access PTE
- access instr

- What can we do?
  - Increase cycle time
  - Take 2 cycles in IF

$\left.\begin{array}{l}\text{Increase cycle time}\\\text{Take 2 cycles in IF}\end{array}\right\}$ Both bad!

In a paged memory system



Two memory accesses for every instruction:
- access PTE
- access instr

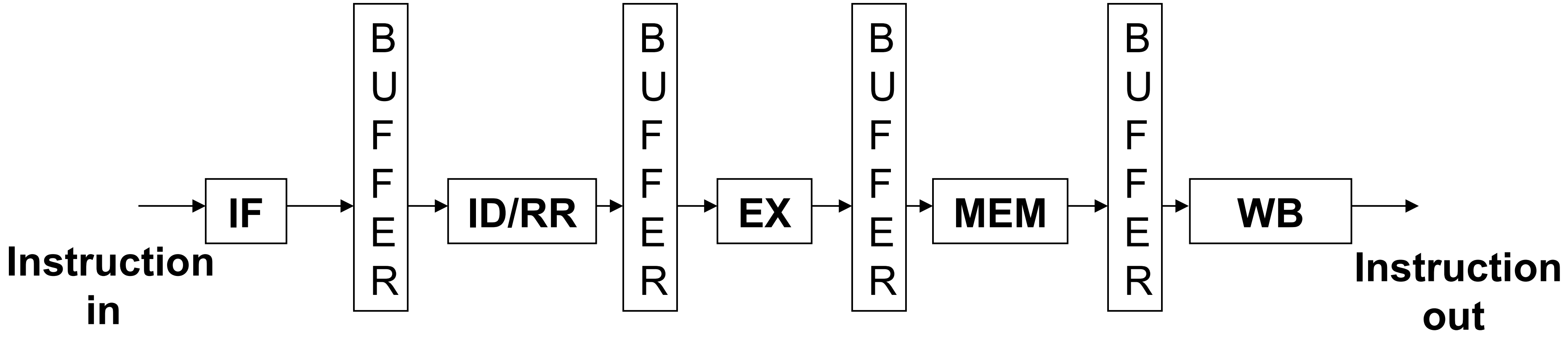
- TLB saves the day…
  - PC Ⓟ TLB Ⓟ memory is only one memory access
  - So we’re back in business….

Are we really?

- What's the CPU clock cycle speed? ~1ns
- What's the memory access time? ~100ns

- It's roughly a 100:1 ratio!

What happens to the pipeline?



- With a 100:1 ratio, what happens in the IF stage on a memory fetch?
- 99 bubbles... on every instruction!

Not sustainable.
FAIL!

The register file and TLB…

- They're "kinda" memories, right?
- Why are they not a problem?
- TLB + Reg file Ⓟ Static RAM (SRAM) technology
- Memory Ⓟ Dynamic RAM (DRAM) technology
- SRAM Ⓟ 6 transistors per cell Ⓟ faster Ⓟ bulkier
- DRAM Ⓟ 1 transistor per cell Ⓟ slower Ⓟ denser

- Upshot: You can have size or speed but not both!
- This also means "memory access" cannot always be part of the CPU clock cycle time

What principle makes TLB work?

- Locality
  - Sequentiality of instructions in the program
  - Sequentiality of data structures like arrays and structs

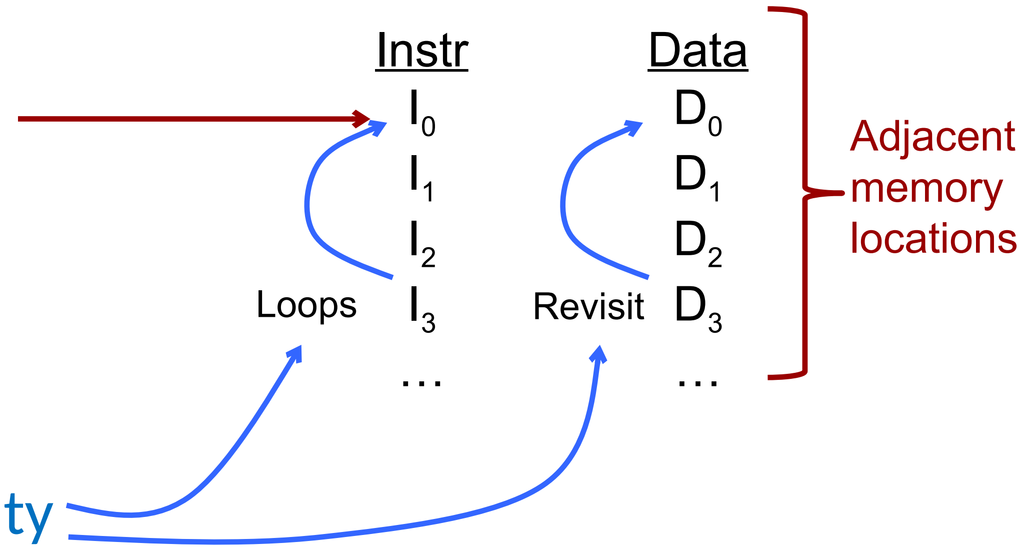
  - “Recent” translations are stashed away from in-memory page tables into high-speed hardware
  - Subsequent translations become faster

    Ⓟ no need to go to memory for most PTEs

  - What if we apply the same principle to the data and instructions we need?

The principle is locality

- Spatial locality
- Temporal locality

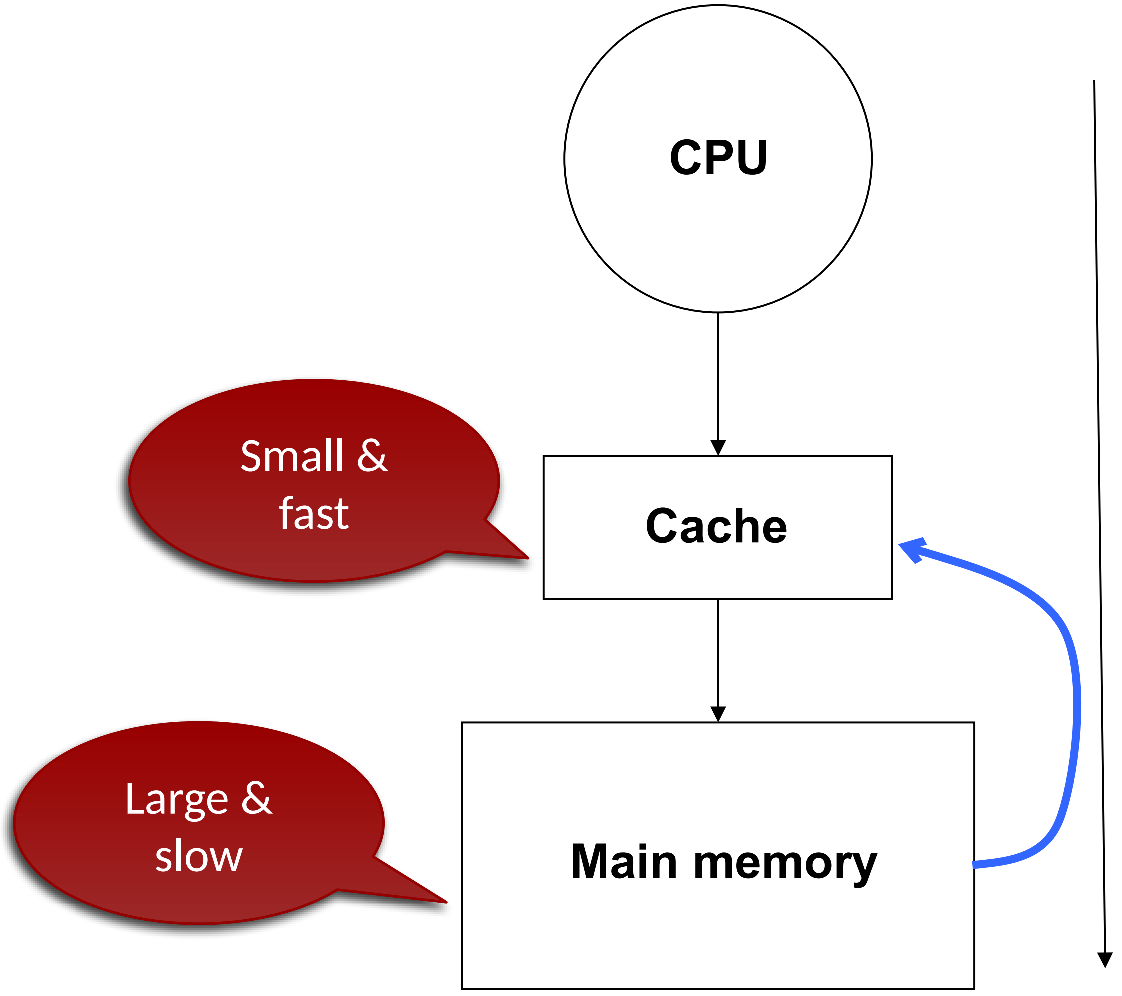


How do we put these two ideas together?

- Locality
- Small & fast or large & slow storage

- Ⓟ The concept of “cache”
- TLB is a special instance of caching “addresses”

The model



- How do you use
  - Spatial locality?
  - Temporal locality?

Terminologies

Hit	Hit ratio h	$h + m = 1$
Miss	Miss ratio m	
Cycle time	T_c cache access time	
	T_m memory access time	
Memory access time	Same for SRAM Different for DRAM	
Memory cycle time		
Miss penalty	T_m	
Effective memory access time (EMAT)	EMAT = $T_c + T_m * m$	

Current-day memory hierarchy

$EMAT_i = T_i + M_i * EMAT_{i+1}$

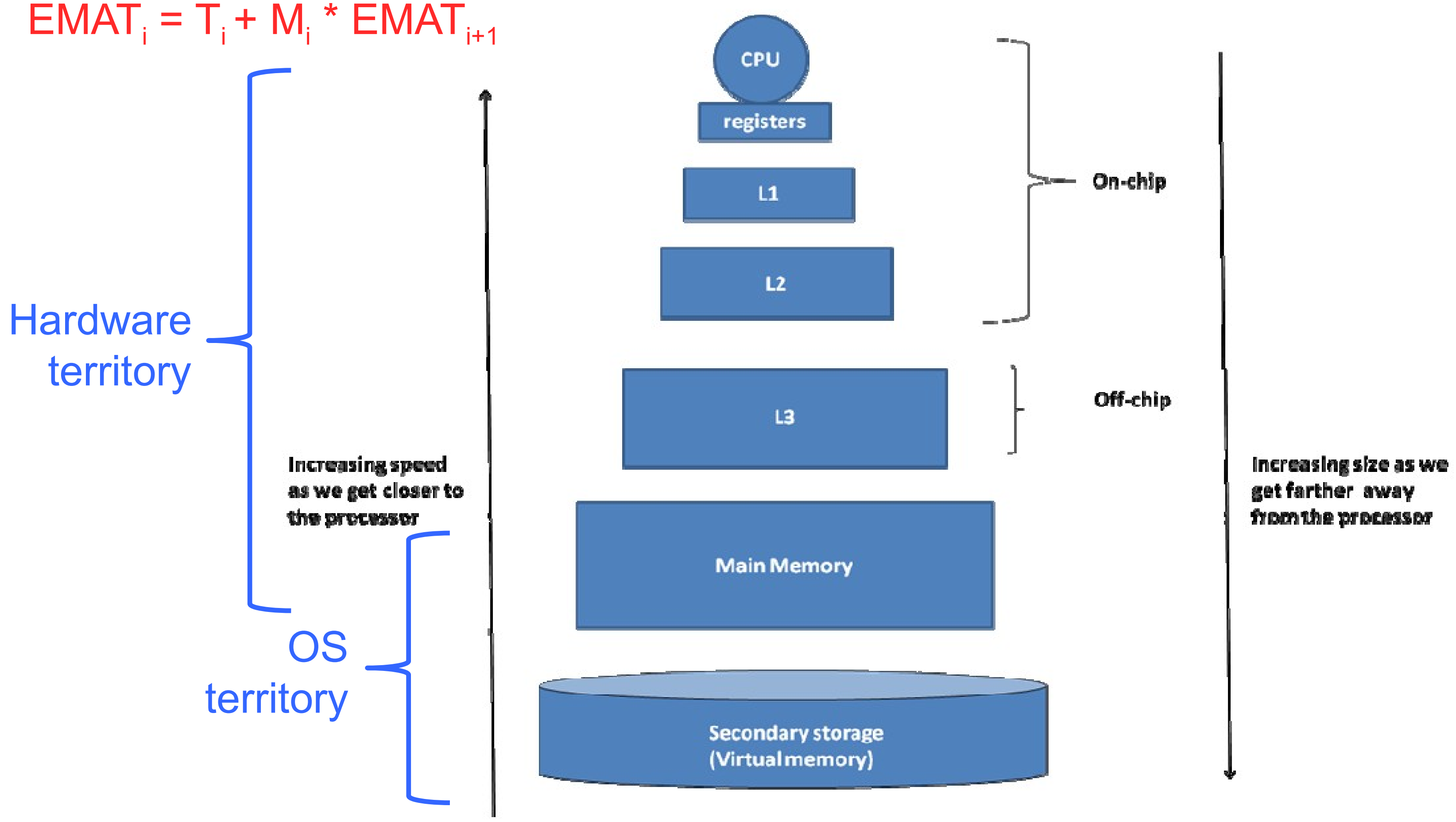

Hardware territory
OS territory
Increasing speed as we get closer to the processor
CPU
registers
L1
L2
L3
Main Memory
Secondary storage (Virtual memory)
On-chip
Off-chip
Increasing size as we get farther away from the processor

Figure 9.2: The entire memory hierarchy stretching from processor registers to the virtual memory.

Calculating EMAT

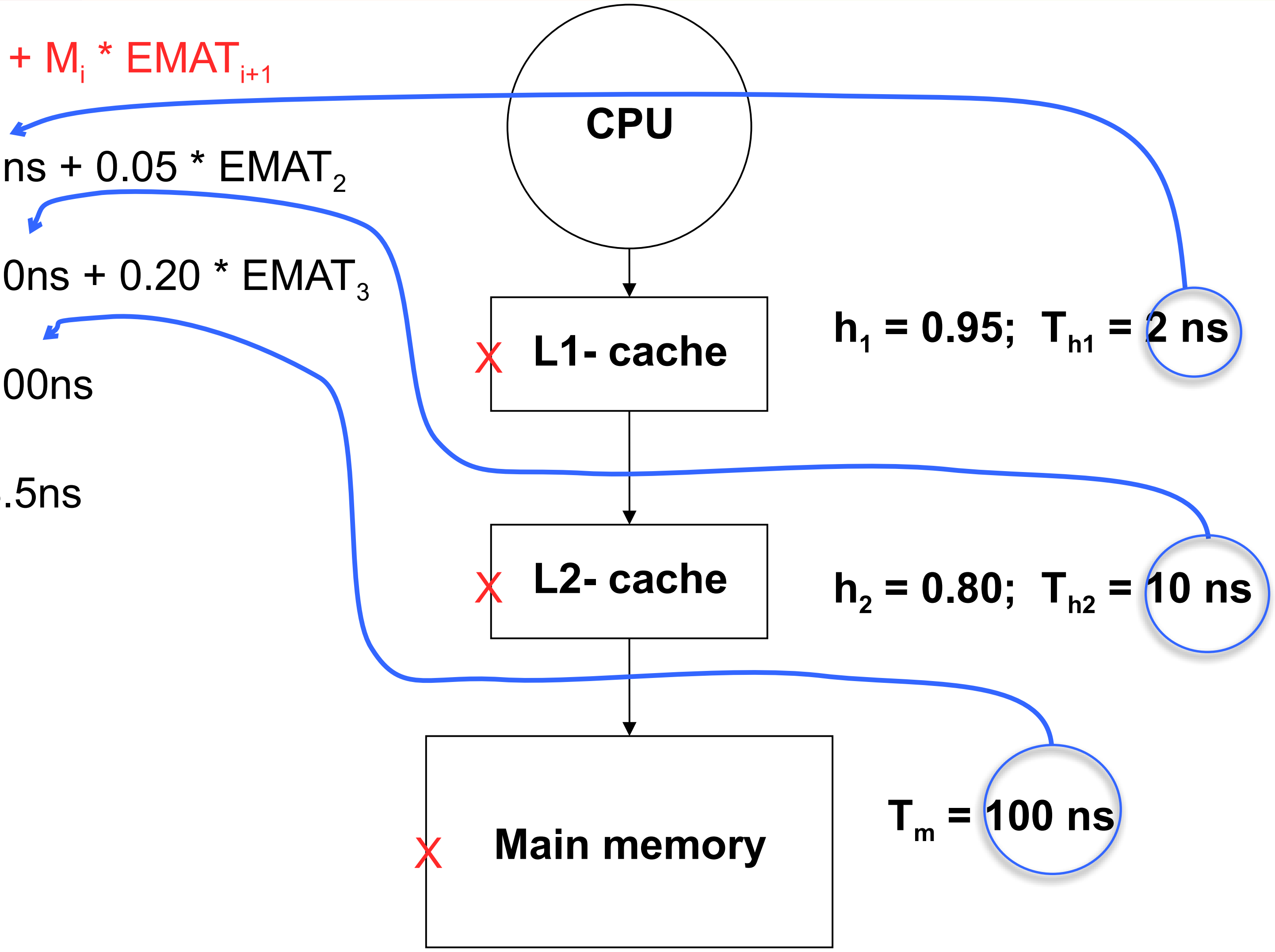
$EMAT_i = T_i + M_i * EMAT_{i+1}$

$EMAT_1 = 2ns + 0.05 * EMAT_2$

$EMAT_2 = 10ns + 0.20 * EMAT_3$

$EMAT_3 = 100ns$

$EMAT_1 = 3.5ns$



Calculate the EMAT?

You have a one-level cache that has a hit rate of 90% and an access time of 10ns; your main memory has an access time of 100 ns.

A. 11ns
B. 110ns
C. 12ns
[✓ D. 20ns]

$\mathrm{EMAT}_i = T_i + M_i * \mathrm{EMAT}_{i+1}$

$\mathrm{EMAT}_1 = 10\mathrm{ns} + 0.10 * \mathrm{EMAT}_2$

$\mathrm{EMAT}_2 = 100\mathrm{ns}$

$\mathrm{EMAT}_1 = 10\mathrm{ns} + 0.10 * 100\mathrm{ns}$

$\mathrm{EMAT}_1 = 20\mathrm{ns}$

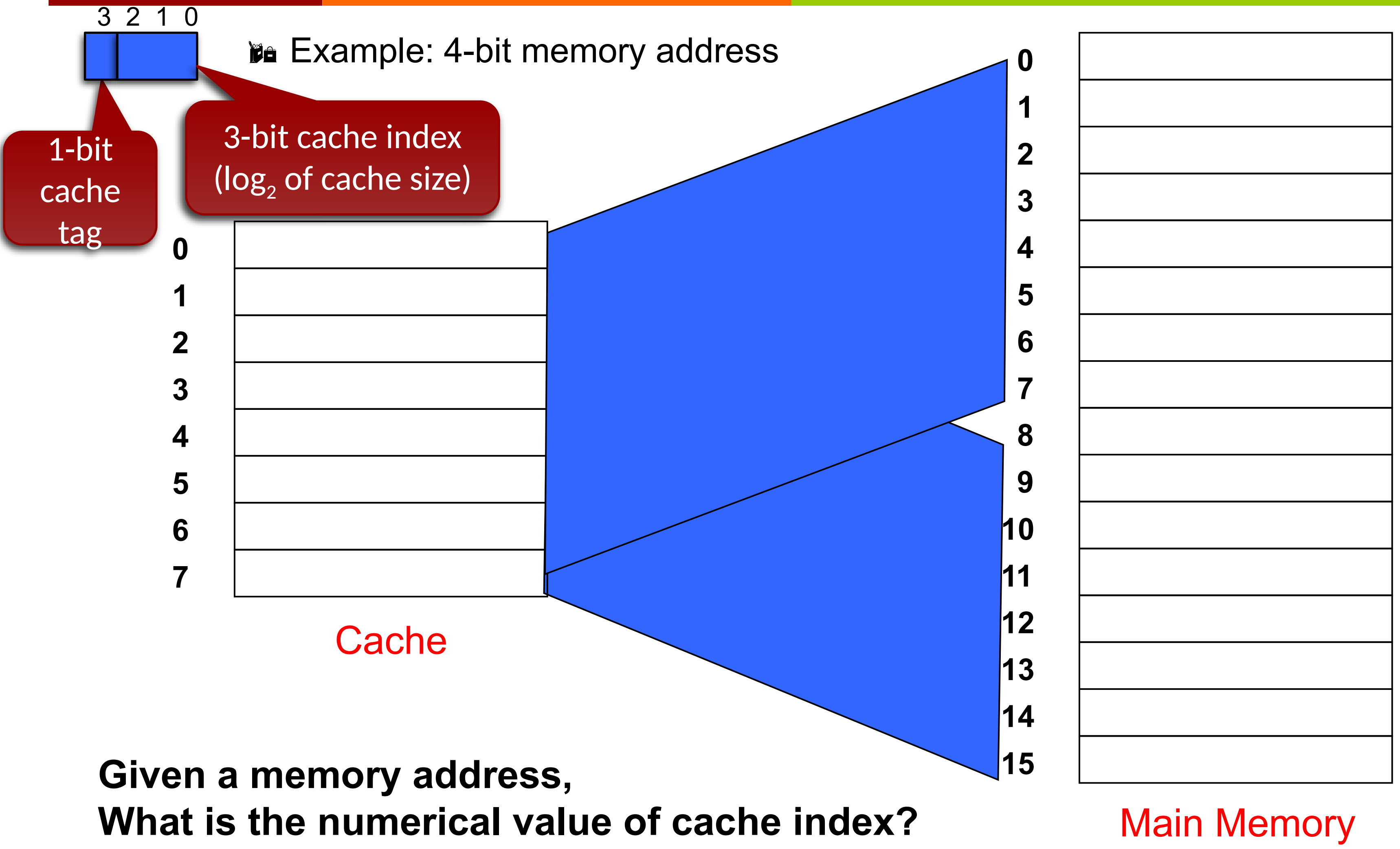
Today’s number is 30,333

Cache Organizations

- Direct mapped
- Fully associative
- Set associative

Direct mapped cache

- Example: 4-bit memory address



Given a memory address,
What is the numerical value of cache index?

Types of misses

- Compulsory (first time a block is brought in)
- Capacity (cache is full)
- Conflict (vying for space in a full set in the cache)

- Compulsory > Capacity > Conflict

- Another way to think about Conflict misses: Conflict misses are misses that would not occur if the cache were fully associative with LRU replacement. Fully associate caches can’t have conflict misses by definition.

Example

Memory references: $\underbrace{0, 1, 2, 3}_{\text{misses}}, \underbrace{1, 3, 0}_{\text{hits}}, \underset{\underset{\text{miss}}{\uparrow}}{8}, 0, 9, 10$

These were compulsory misses.

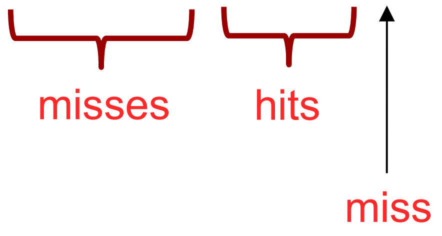
This is the first time the address is begin brought into the cache.

0	mem loc ~~0~~
1	mem loc 1
2	mem loc 2
3	mem loc 3
4	empty
5	empty
6	empty
7	empty

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

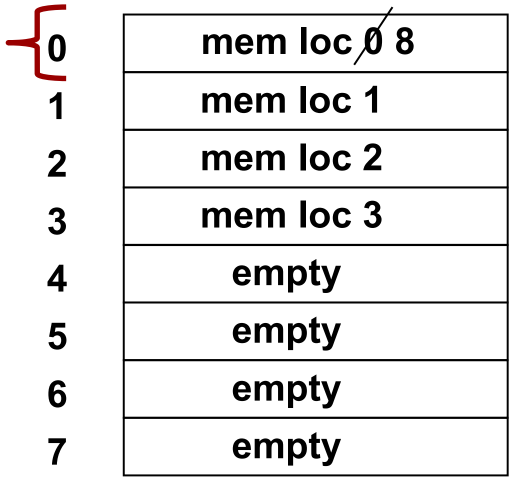
Example

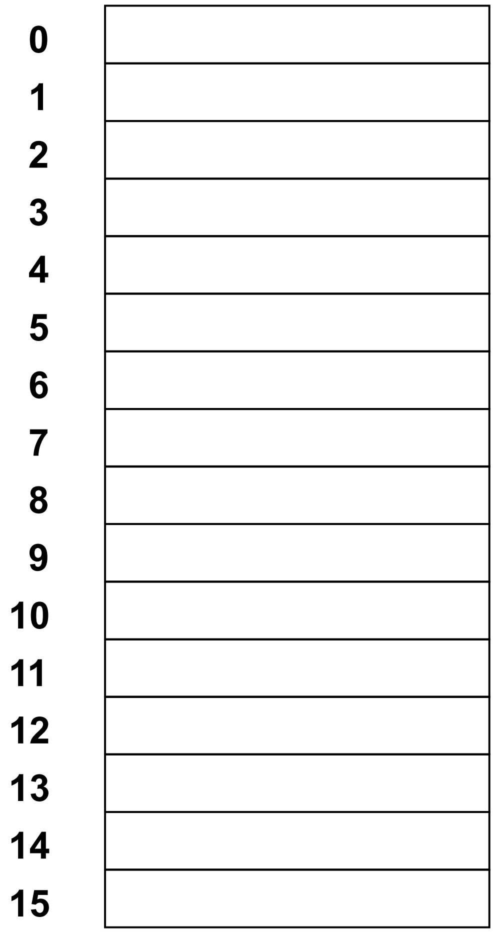
Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10



This is also a compulsory miss.

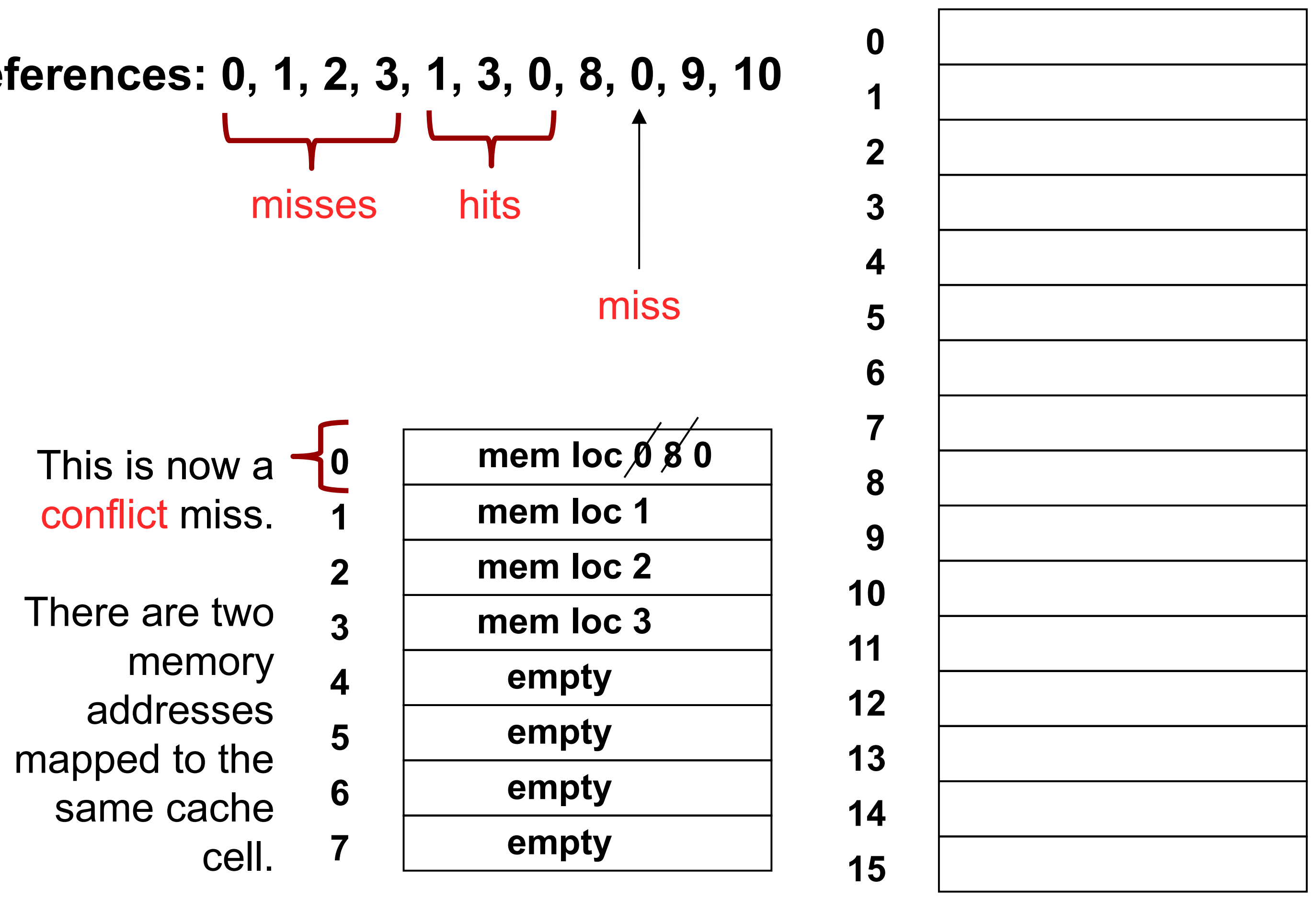
This is the first time address 8 has been brought into the cache.





Example

Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10

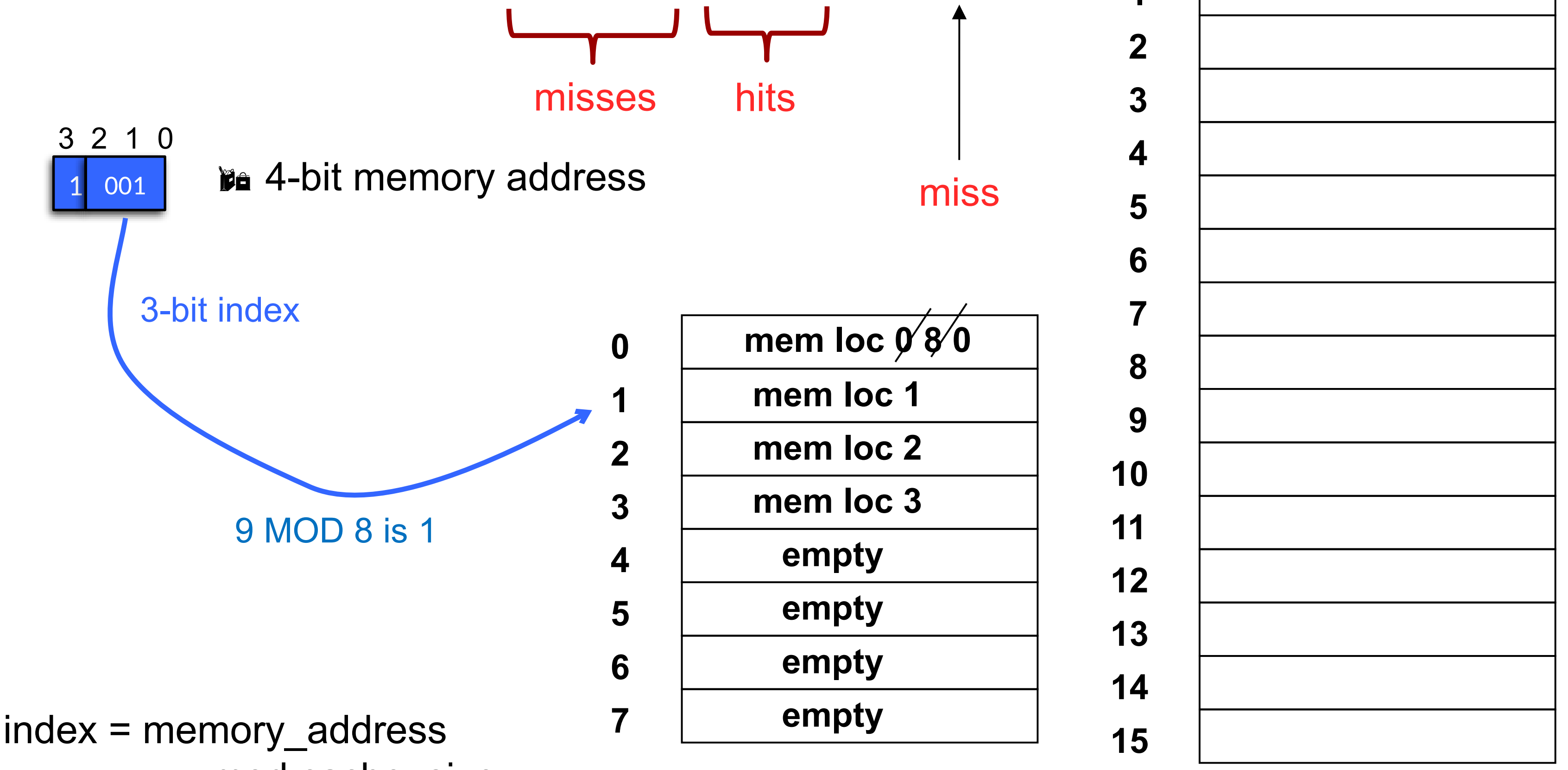


This is now a conflict miss.

There are two memory addresses mapped to the same cache cell.

Example

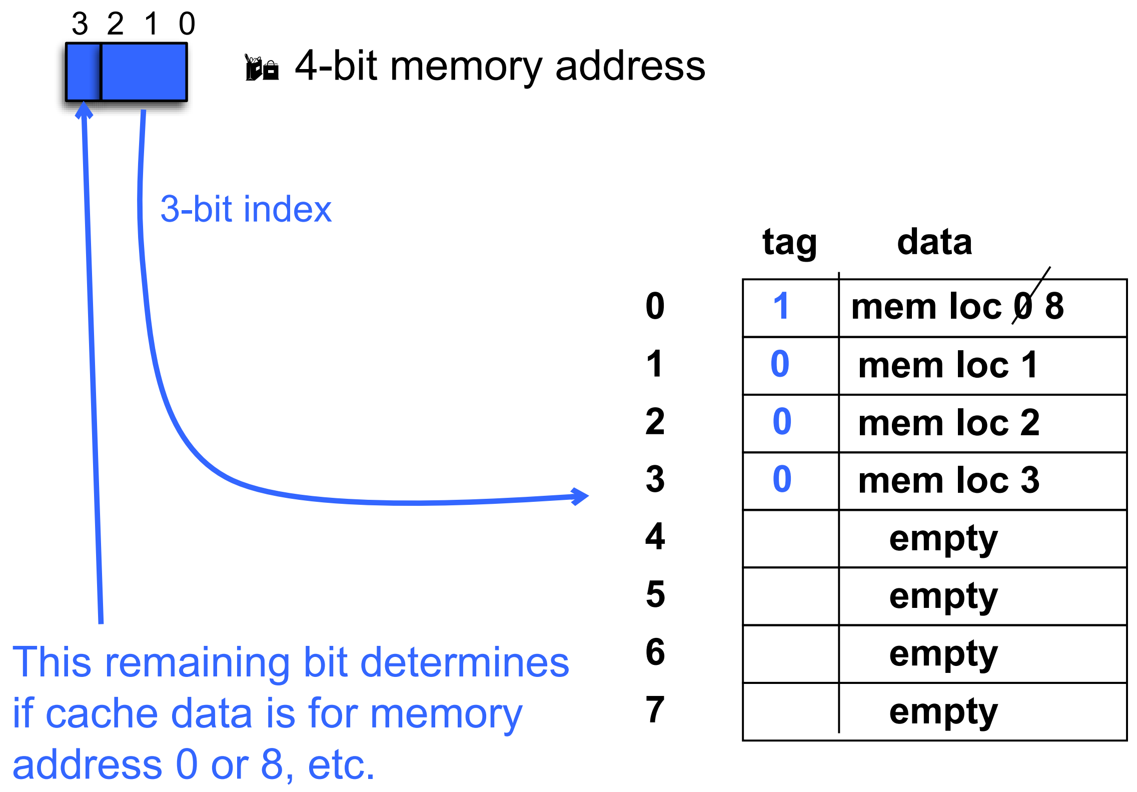
Memory references: 0, 1, 2, 3, 1, 3, 0, 8, 0, 9, 10



index = memory_address
mod cache_size

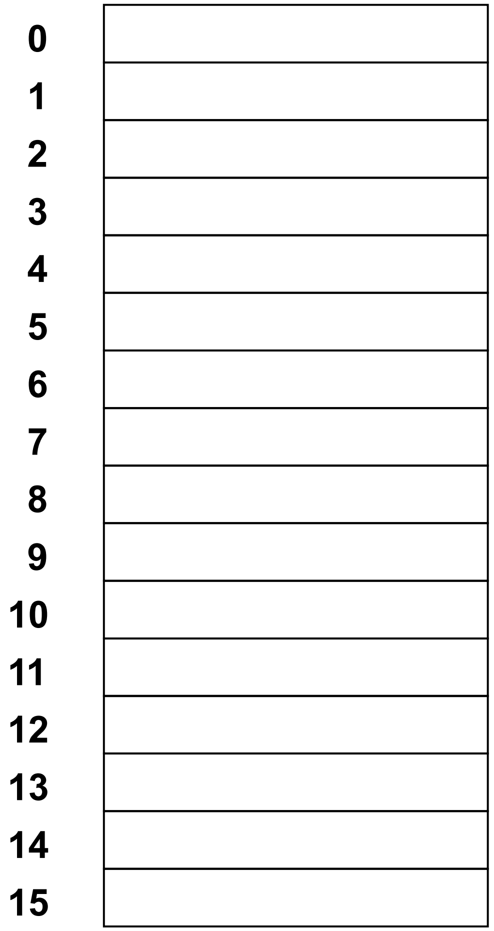
How do we disambiguate data?

We use the rest of the memory address as the tag to label the data in the cache



3 2 1 0
4-bit memory address
3-bit index
This remaining bit determines if cache data is for memory address 0 or 8, etc.

	tag	data
0	1	mem loc ~~0~~ 8
1	0	mem loc 1
2	0	mem loc 2
3	0	mem loc 3
4		empty
5		empty
6		empty
7		empty



How do we know data is valid?

- At power-up, a cache may contain garbage!
- Tags help disambiguate, not validate

	valid	tag	data
0	1	1	loc 8
1	1	0	loc 1
2	1	0	loc 2
3	1	0	loc 3
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Fields in a Direct Mapped Cache

$$\underbrace{\boxed{\textbf{Valid}}\boxed{\textbf{Tag}}}_{\text{Meta data}}\underbrace{\boxed{\qquad\qquad\textbf{Data}\qquad\qquad}}_{\text{Real data}}$$

What type of locality affects direct mapped cache performance?

A. Spatial locality
B. Temporal locality
C. Virtual locality
[✓ D. Spatial and temporal locality]

What does the cache entry contain?

You have a 64 entry direct-mapped cache for a memory with 16-bit addresses and 16-bit words (like LC-3).

A. 1 bit valid flag, 6 bit tag, 10 bit data

[✓ B. 1 bit valid flag, 10 bit tag, 16 bit data]

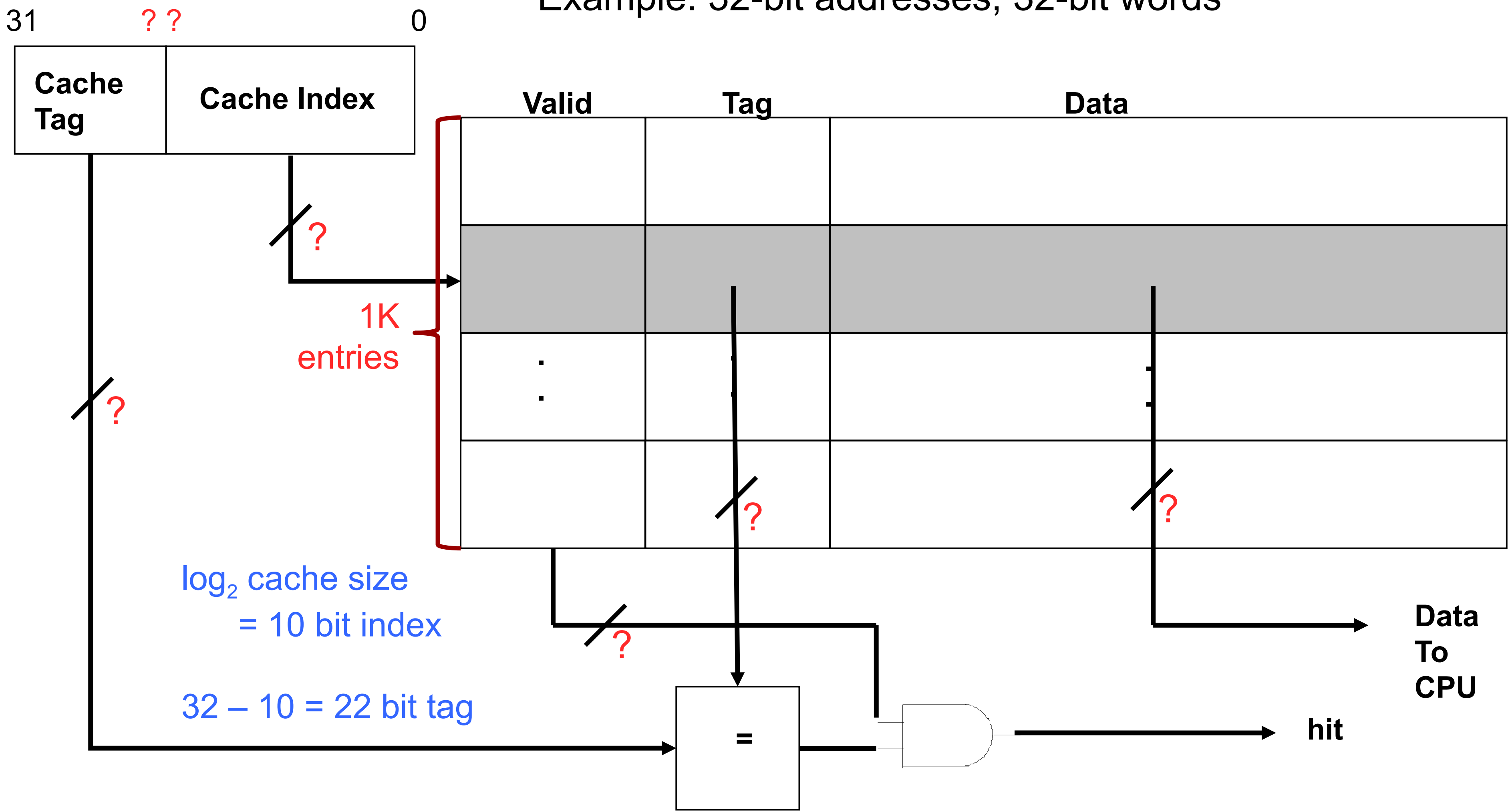
C. 1 bit valid flag, 10 bit tag, 6 bit data

D. 2 bit valid flag, 12 bit tag, 16 bit data

One bit is required for the valid flag
The cache index is going to be 6 because $\log_2(64)=6$, so tag will be 16-6=10 bits
Memory word is 16 bits, so data has to be a 16 bits (or a multiple)

Hardware

Example: 32-bit addresses, 32-bit words



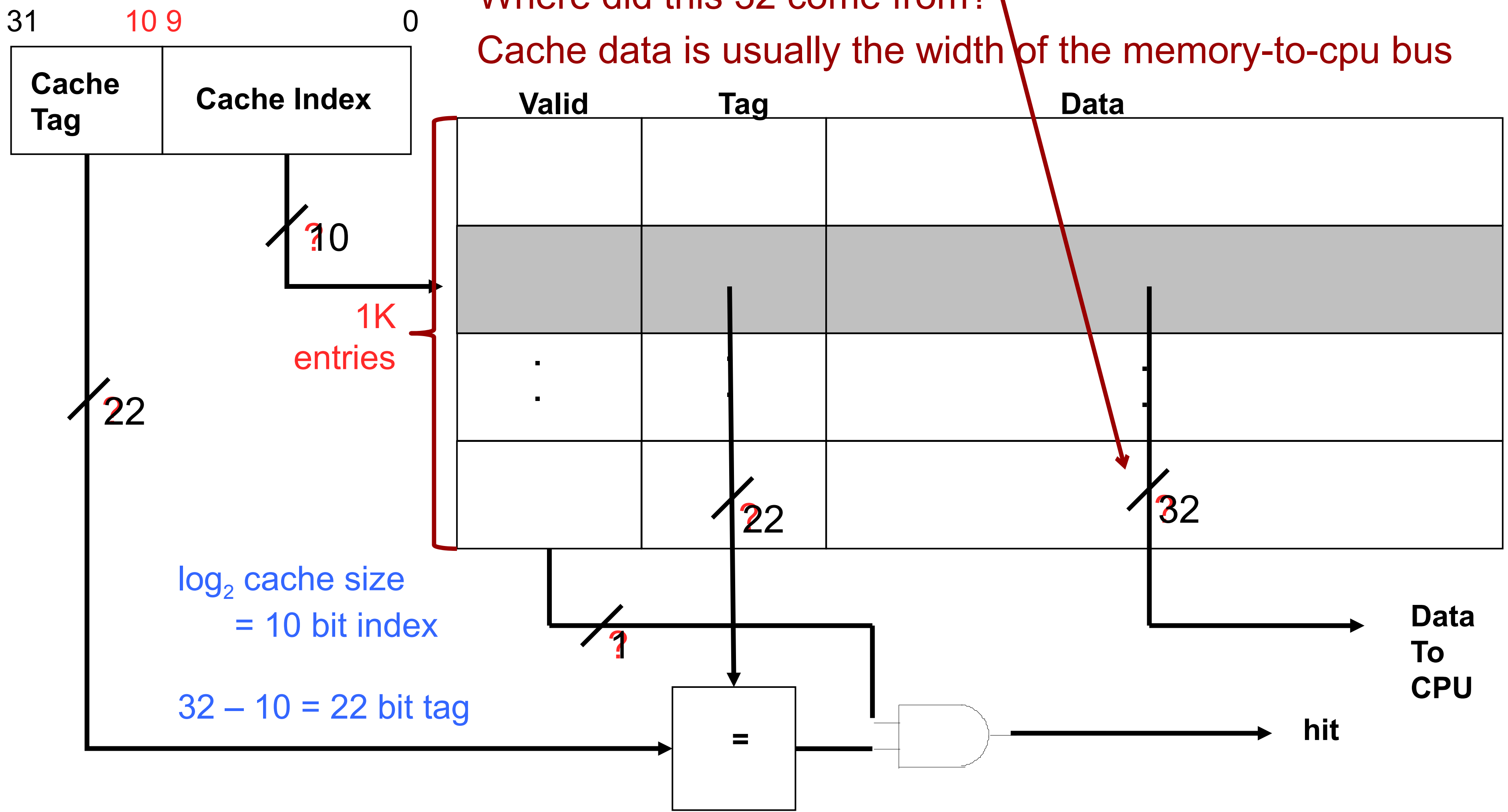
log$_2$ cache size
= 10 bit index

32 – 10 = 22 bit tag

Hardware

Where did this 32 come from?

Cache data is usually the width of the memory-to-cpu bus



$\log_2$ cache size = 10 bit index

32 – 10 = 22 bit tag

Interpreting memory addresses

Memory address

Cache Tag	Cache Index

- index = memory addr mod cache size
  - MemAddr = 8 Ⓟ index = 0
  - MemAddr = 0 Ⓟ index = 0

- number of tag bits = memory addr size
  – cache index size

Why not the other way around?
Ⓟ use the low bits for cache tag?

Index first, tag last?

	valid	tag	data
0	1	0	loc 0
1	0	X	empty
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access location 0

	valid	tag	data
0	1	1	loc ~~0~~ 1
1	0	X	empty
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access location 1

	valid	tag	data
0	1	1	loc ~~0~~ 1
1	1	0	loc 2
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access location 2

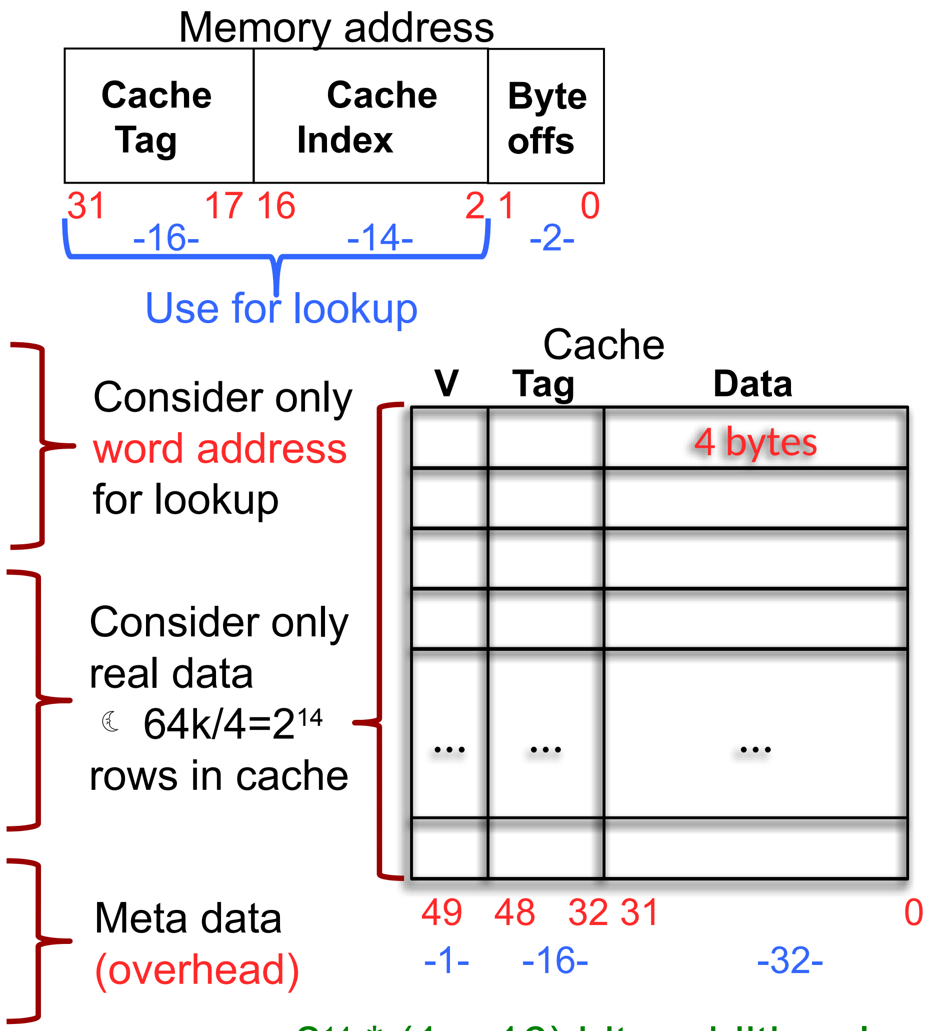
	valid	tag	data
0	1	1	loc ~~0~~ 1
1	0	1	loc ~~2~~ 3
2	0	X	empty
3	0	X	empty
4	0	X	empty
5	0	X	empty
6	0	X	empty
7	0	X	empty

Access location 3

Cache occupancy if we switch index and tag is BAD!! Ⓟ loss of spatial locality

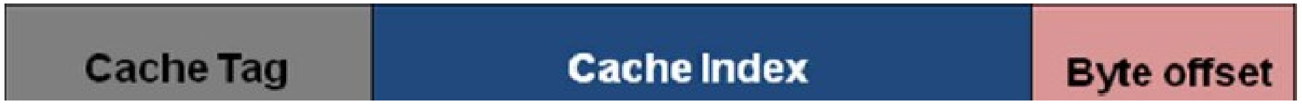
Example

- Let us consider the design of a direct-mapped cache for a realistic memory system.
  - Assume that the CPU generates a 32-bit byte-addressable memory address.
  - Each memory word contains 4 bytes.
  - A memory access brings in a full word into the cache.
  - The direct-mapped cache is 64K bytes in size (this is the amount of data that can be stored in the cache), with each cache entry containing one word of data.
  - Compute the additional storage space needed for the valid bits and the tag fields of the cache.

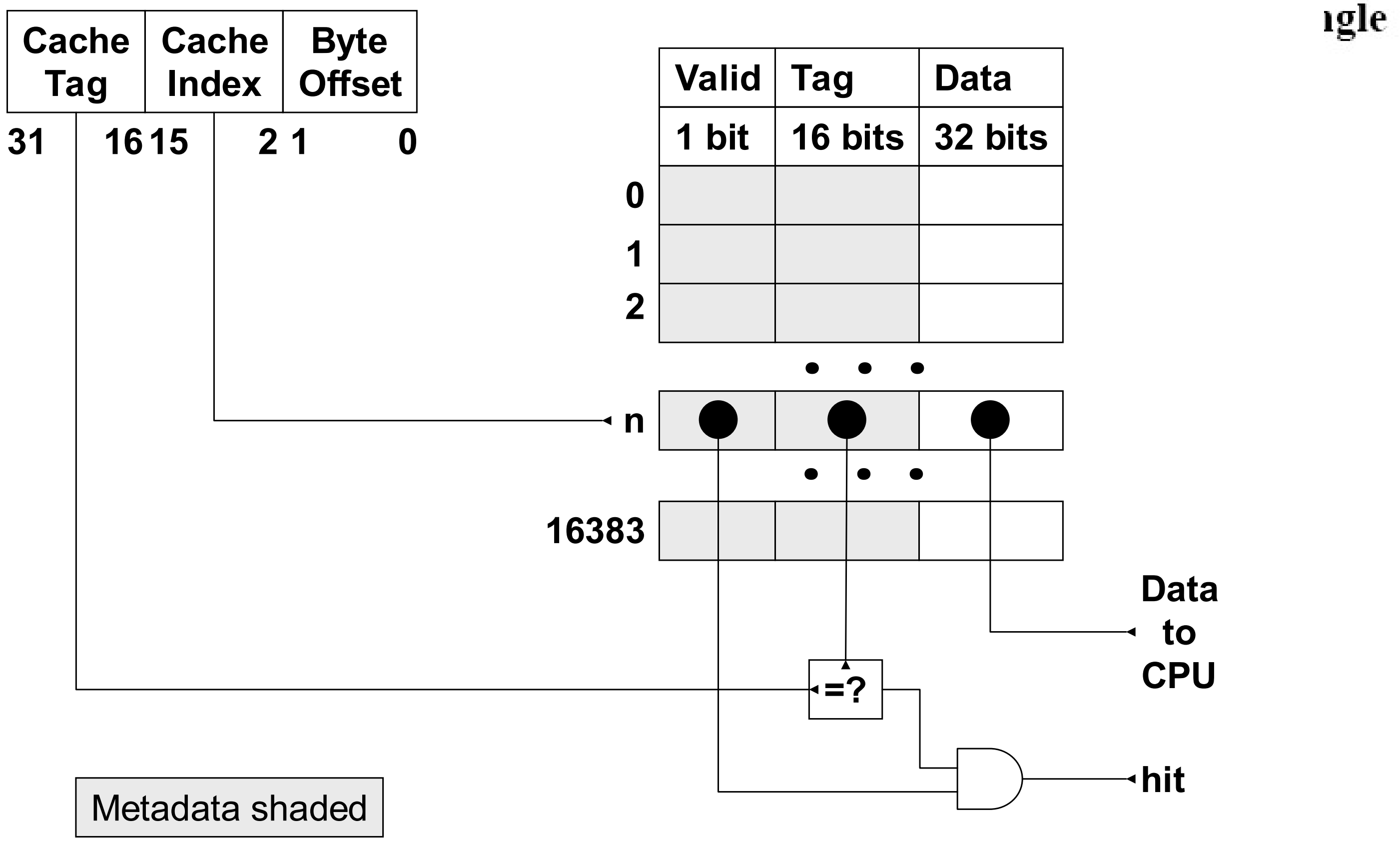


$2^{14} * (1 + 16)$ bits additional

Memory address interpretation when single cache block contains multiple bytes



F ıgle



A direct-mapped cache

[✓ A.] Has a many-to-one mapping between a memory location and a cache location

B. Allows a memory location to be en-cached wherever there is space in the cache

C. Is so-called because there is a directory associated with the contents of the cache

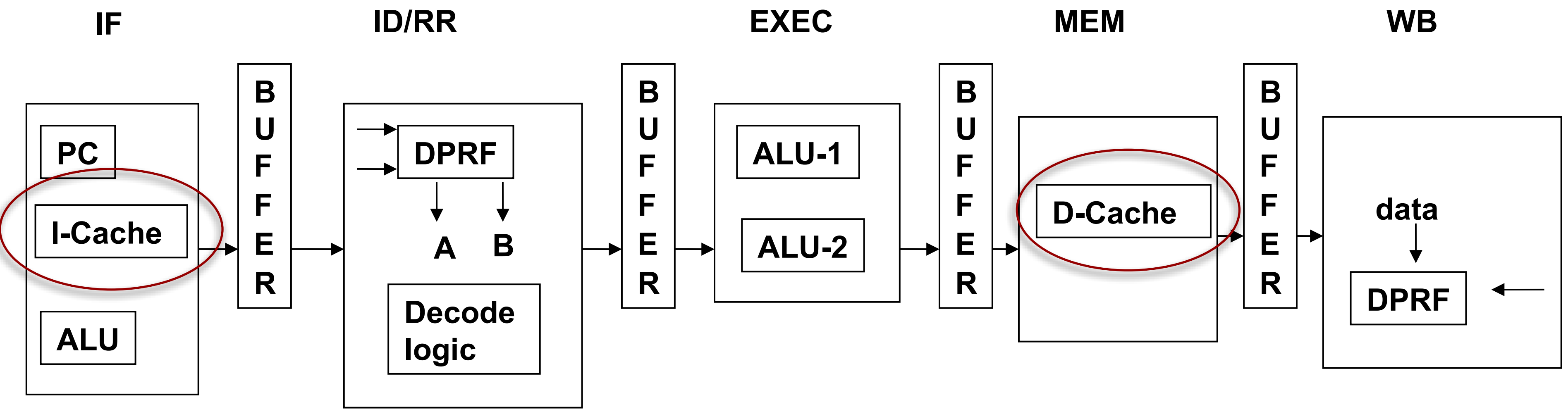
D. Is usually much smaller than any other type of cache organization

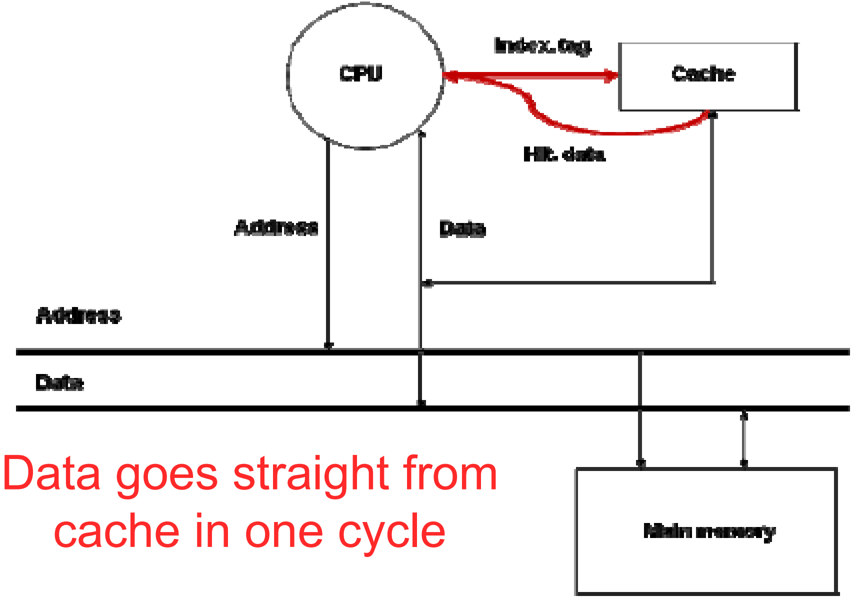
Today's number is 69,727

In a direct-mapped cache with a t-bit tag

A. There is a 1-bit tag comparator for each cache line
B. There is a t-bit tag comparator for each cache line
C. There is a 1-bit tag comparator for the entire cache
[✓ D. There is a t-bit tag comparator for the entire cache]
E. What is a comparator?

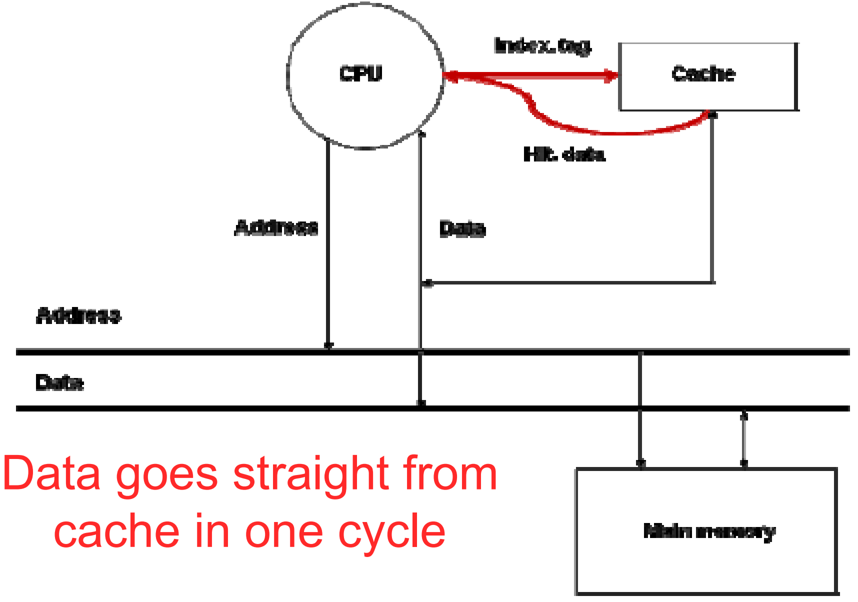
Pipelined processor with caches





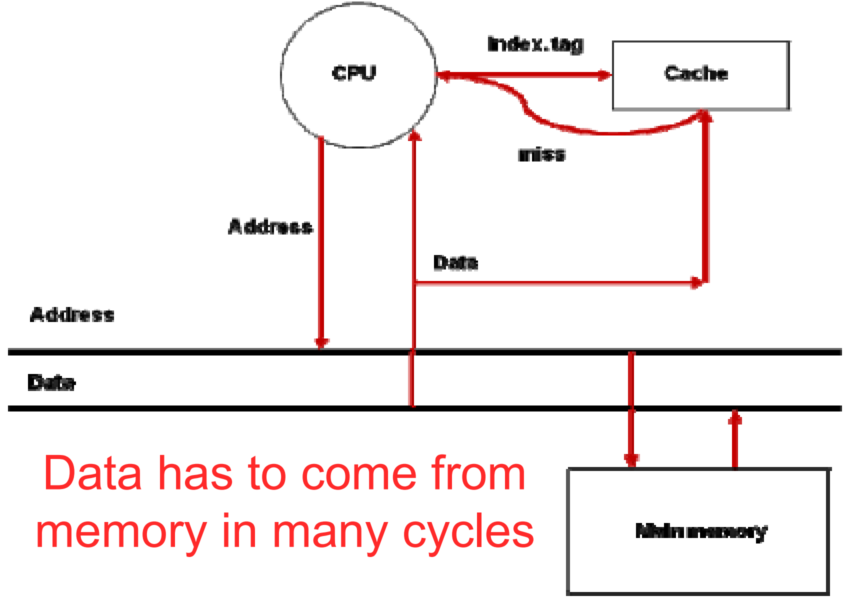
Data goes straight from cache in one cycle

(a) Read Hit



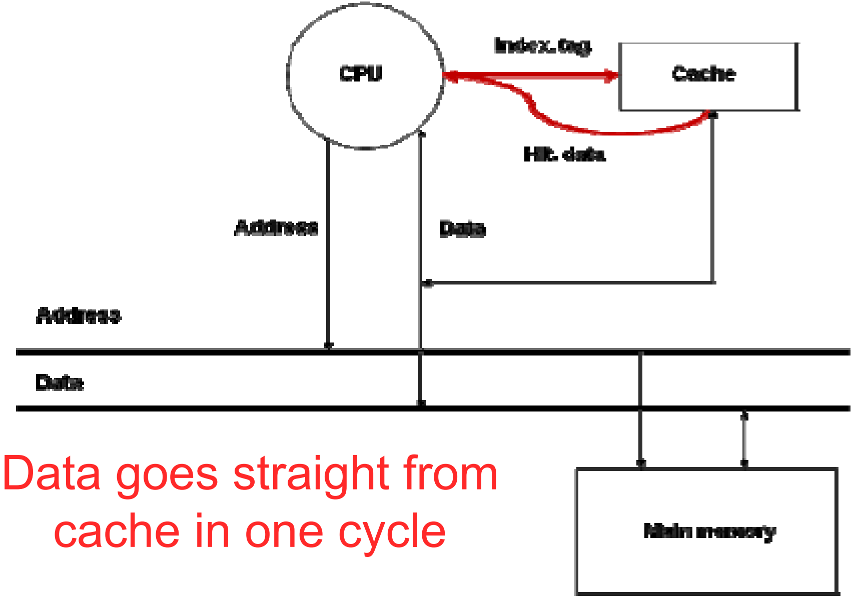
Data goes straight from cache in one cycle

(a) Read Hit



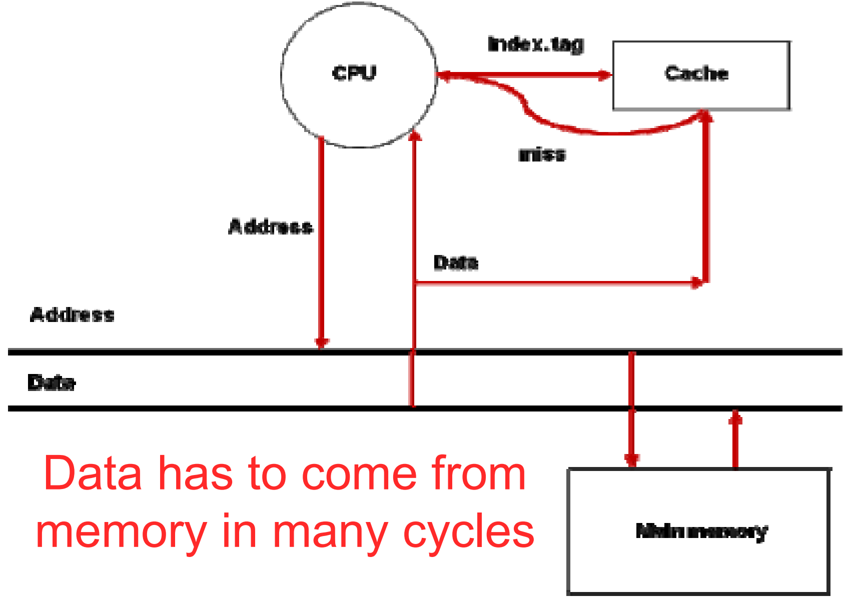
Data has to come from memory in many cycles

(b) Read miss



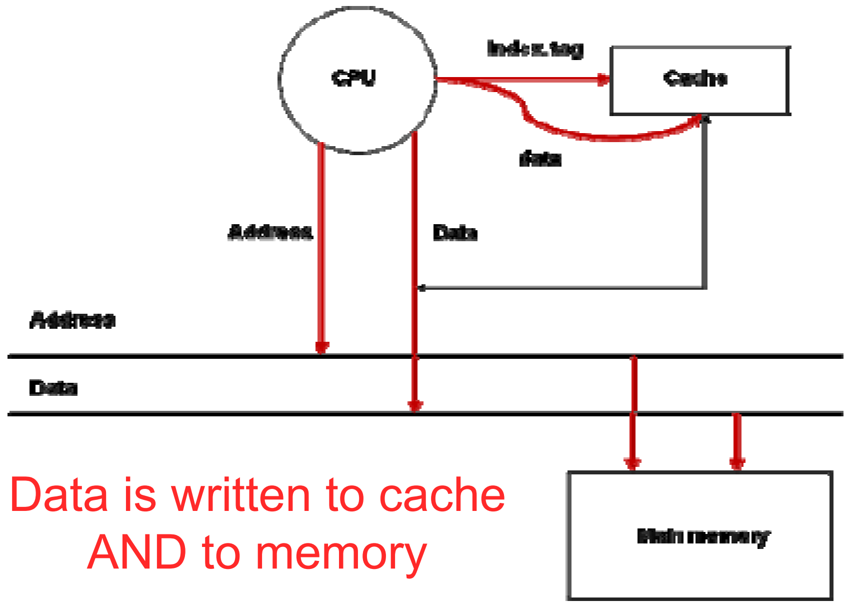
Data goes straight from cache in one cycle

(a) Read Hit



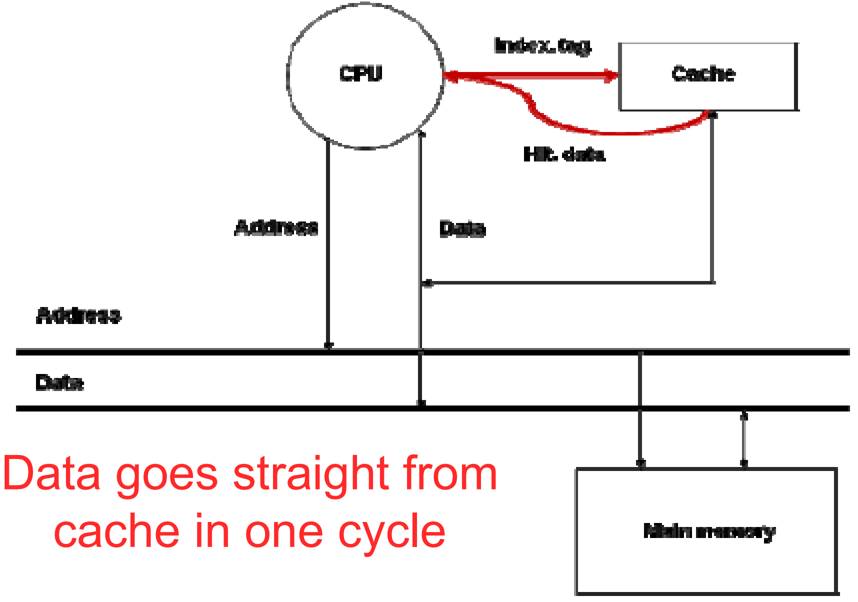
Data has to come from memory in many cycles

(b) Read miss



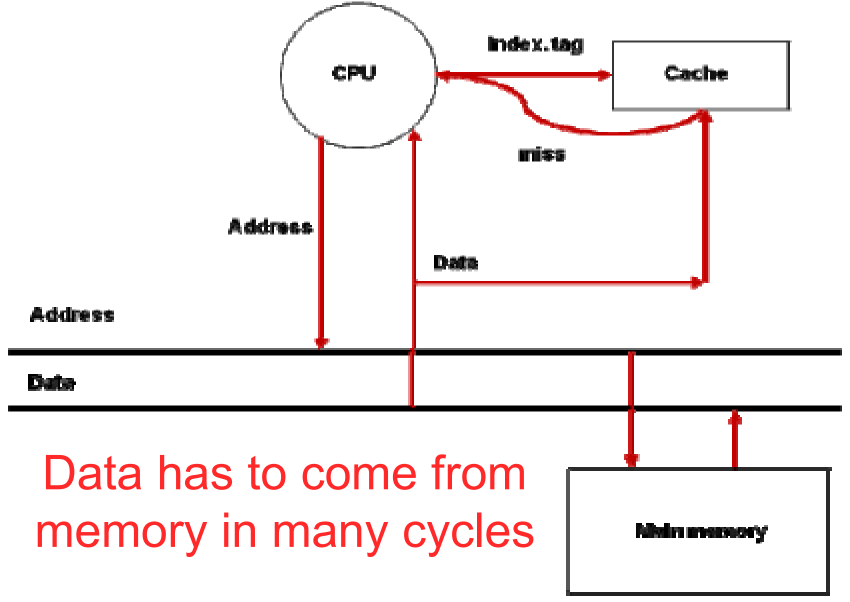
Data is written to cache AND to memory

(c) Write-through



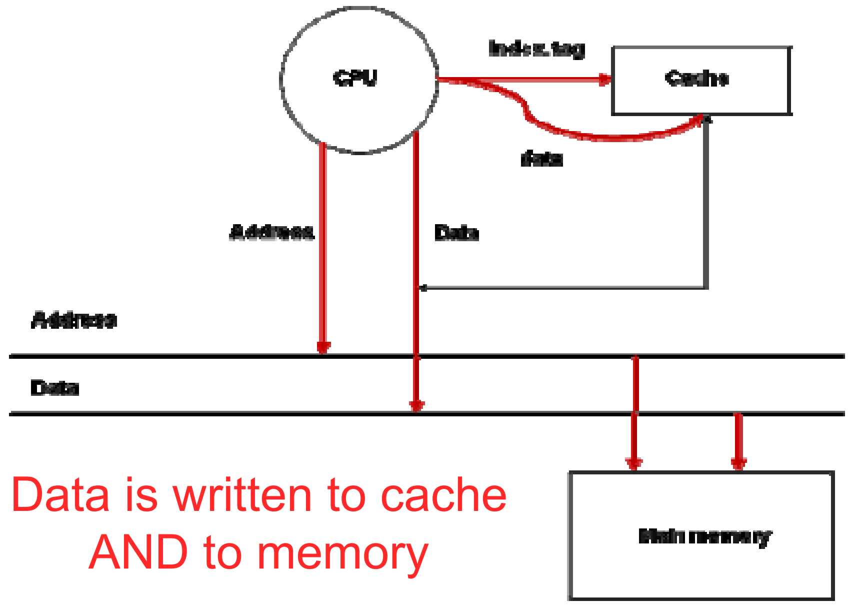
Data goes straight from cache in one cycle

(a) Read Hit



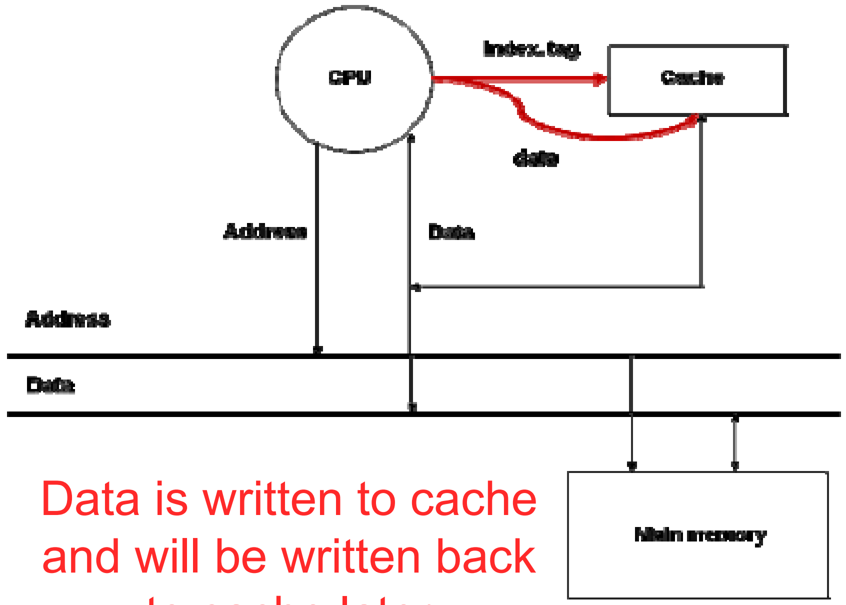
Data has to come from memory in many cycles

(b) Read miss



Data is written to cache AND to memory

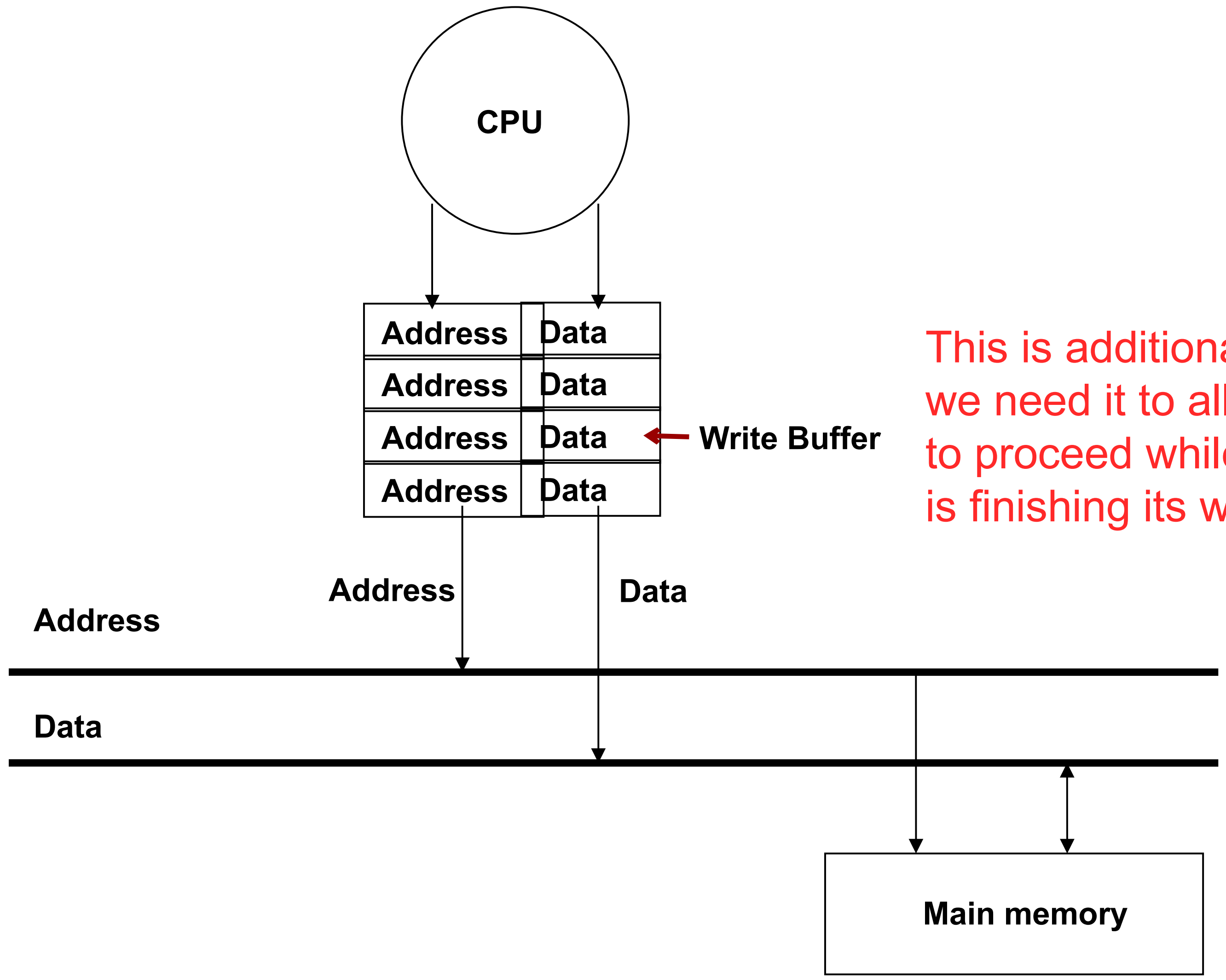
(c) Write-through



Data is written to cache and will be written back to cache later

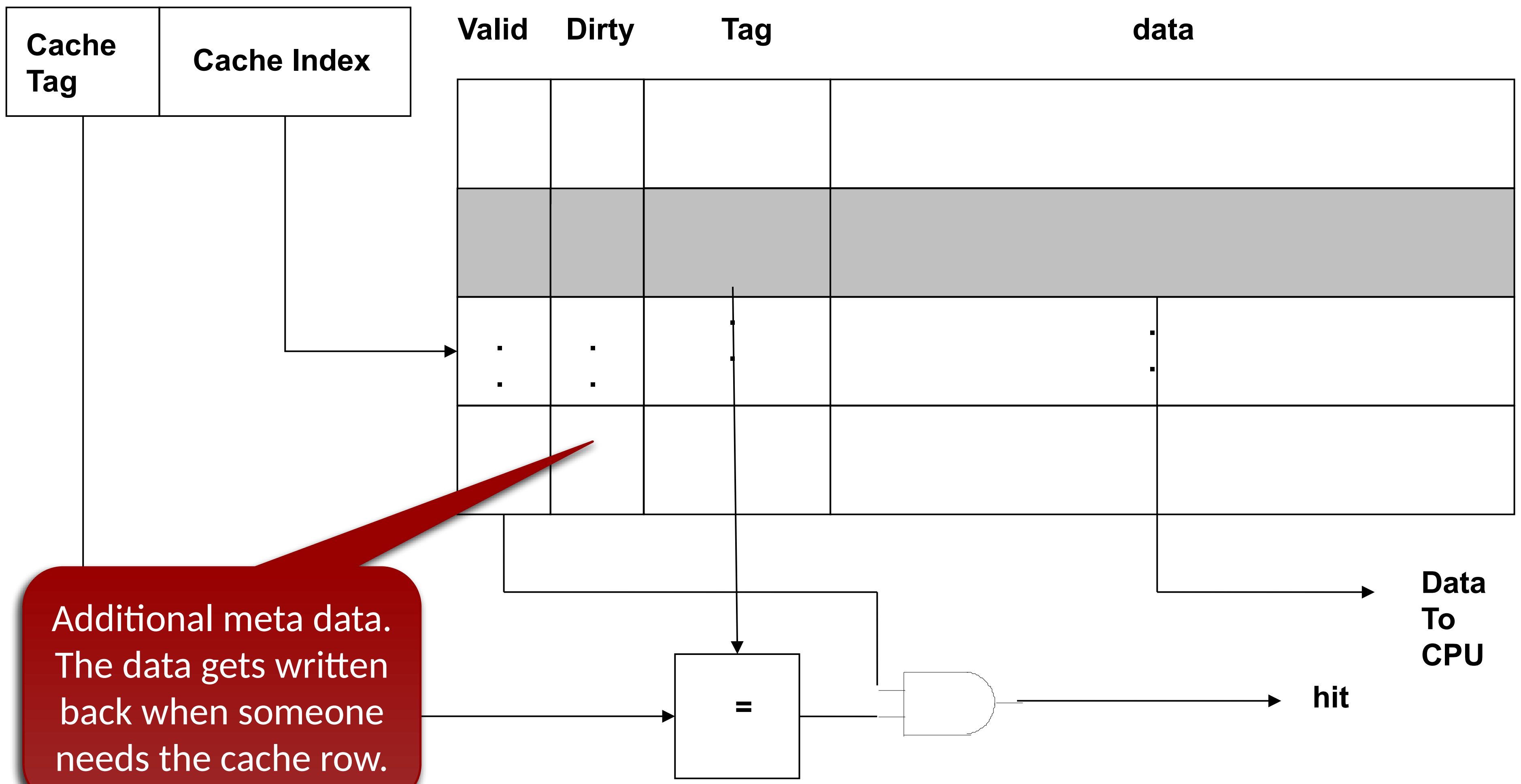
(d) Write-back

Write-through operation



This is additional hardware – we need it to allow the CPU to proceed while the memory is finishing its write cycle

Write back cache



Additional meta data. The data gets written back when someone needs the cache row.

Read miss stalls

```
I1: ld  r1,a;      r1 <- memory at a
I2: add r3,r4,r5; r3 <- r4 + r5
I3: and r6,r7,r8; r6 <- r7 & r8
I4: add r2,r4,r5; r2 <- r4 + r5
I5: add r2,r1,r2; r2 <- r1 + r2
```

- We can treat a read-miss in MEM in a similar fashion as we did previously with registers and the busy bits
- MEM can reset the busy bit for r1 when it sees the read complete for I1 (instead of waiting for WB as we did before)

Execution time with caches

~~Execution time = N * CPI_{Avg} * cycle time~~

CPI_{eff} = CPI_{Avg} + $\text{Memory-stalls}_{\text{Avg}}$

Execution time = N * CPI_{eff} * cycle time

Execution time = N * (CPI_{Avg} + $\text{M-stalls}_{\text{Avg}}$) * cycle time

$\text{Memory-stalls}_{\text{Avg}}$ = misses per $\text{instruction}_{\text{Avg}}$ * $\text{miss-penalty}_{\text{Avg}}$

Total memory stalls = N * $\text{Memory-stalls}_{\text{Avg}}$

The effective CPI is...

Average CPI = 1.5

Average cache miss per instruction = 3%

Miss penalty = 20

A. 1.8
B. 2.1
C. 21.5
D. 7.5
E. No clue

CPI_{eff} = 1.5 + (3% of 20) = 1.5 + 0.6 = 2.1 CPI

Example

- Consider a pipelined processor that has an average CPI of 1.8 without accounting for memory stalls.
  - I-Cache has a hit rate of 95%
  - D-Cache has a hit rate of 98%.
- Assume that memory reference instructions account for 30% of all the instructions executed.
  - 80% are loads
  - 20% are stores
- On average
  - read-miss penalty is 20 cycles and the
  - write-miss penalty is 5 cycles.

Compute the effective CPI of the processor accounting for the memory stalls.

Solution

Cost of instruction misses:

  = I-cache miss rate * read miss penalty

  = (1 – 0.95) * 20

  = 1 cycle per instruction

Cost of data read misses:

  = % memory reference instructions

    * fraction that are loads * D-cache miss rate

    * read miss penalty

  = 0.3 * 0.8 * (1 – 0.98) * 20

  = 0.096 cycles per instruction

Cost of data write misses:

  = % memory reference instructions

    * fraction that are stores * D-cache miss rate

    * write miss penalty

  = 0.3 * 0.2 * (1 – 0.98) * 5

  = 0.006 cycles per instruction

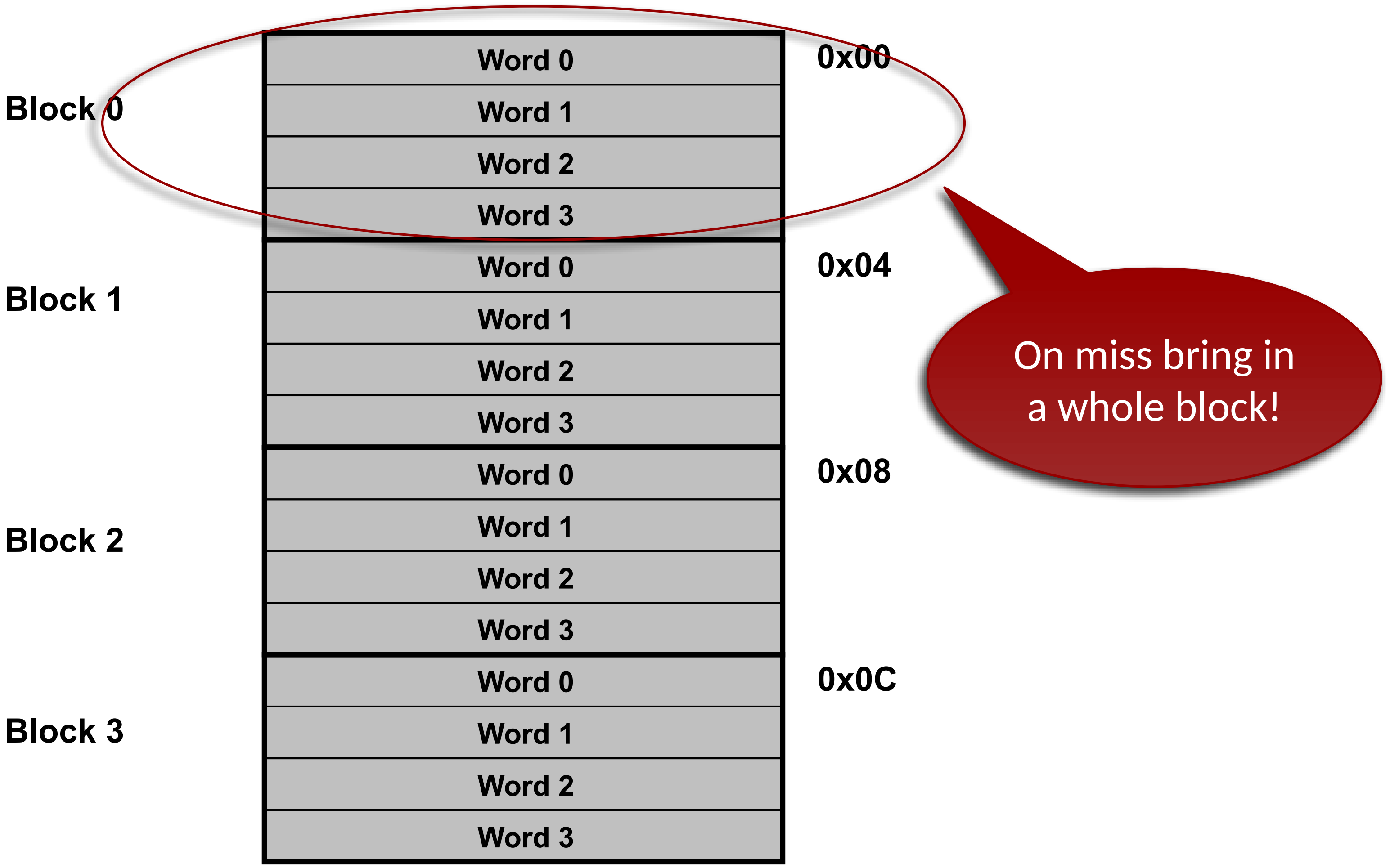
$CPI_{eff} = CPI_{avg}$ + cost of I-cache misses + cost of D-cache misses

  = 1.8 + 1 + (0.096 + 0.006) = 2.902

How to improve cache efficiency

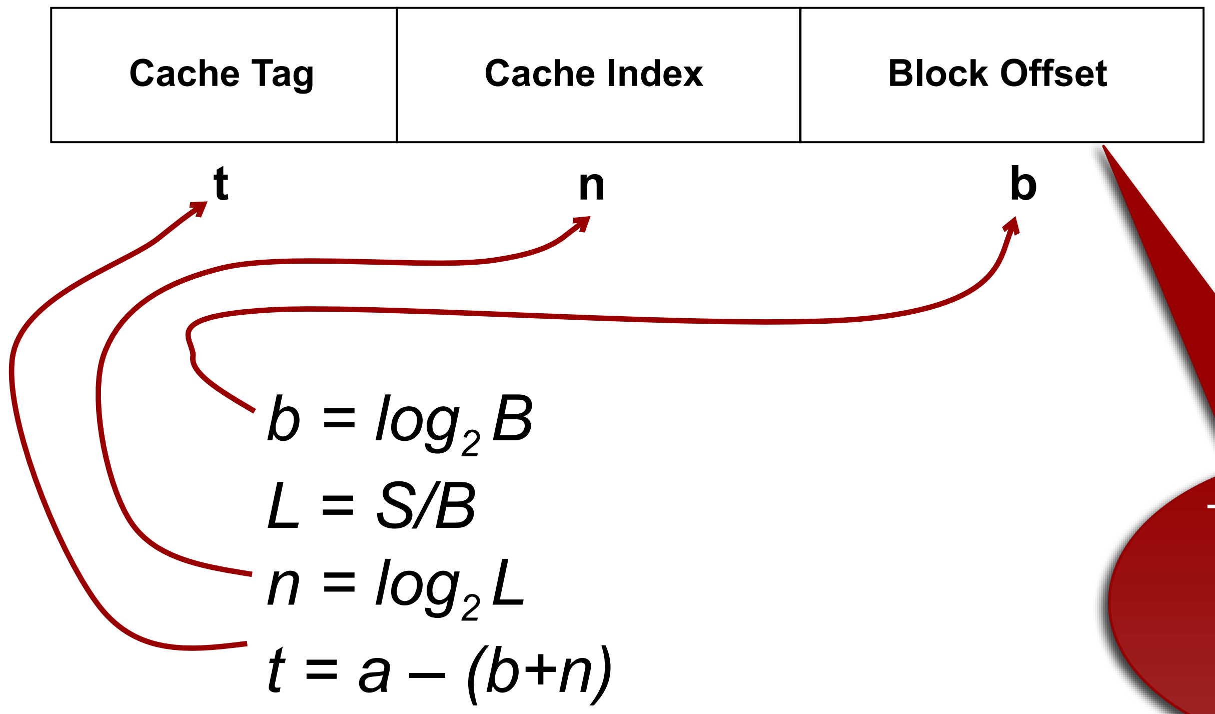
- Further exploit spatial locality
  - Bring more from memory into cache at a time
- Better organization
  - Exploit working set concept

Exploting Spatial Locality



Interpreting memory address

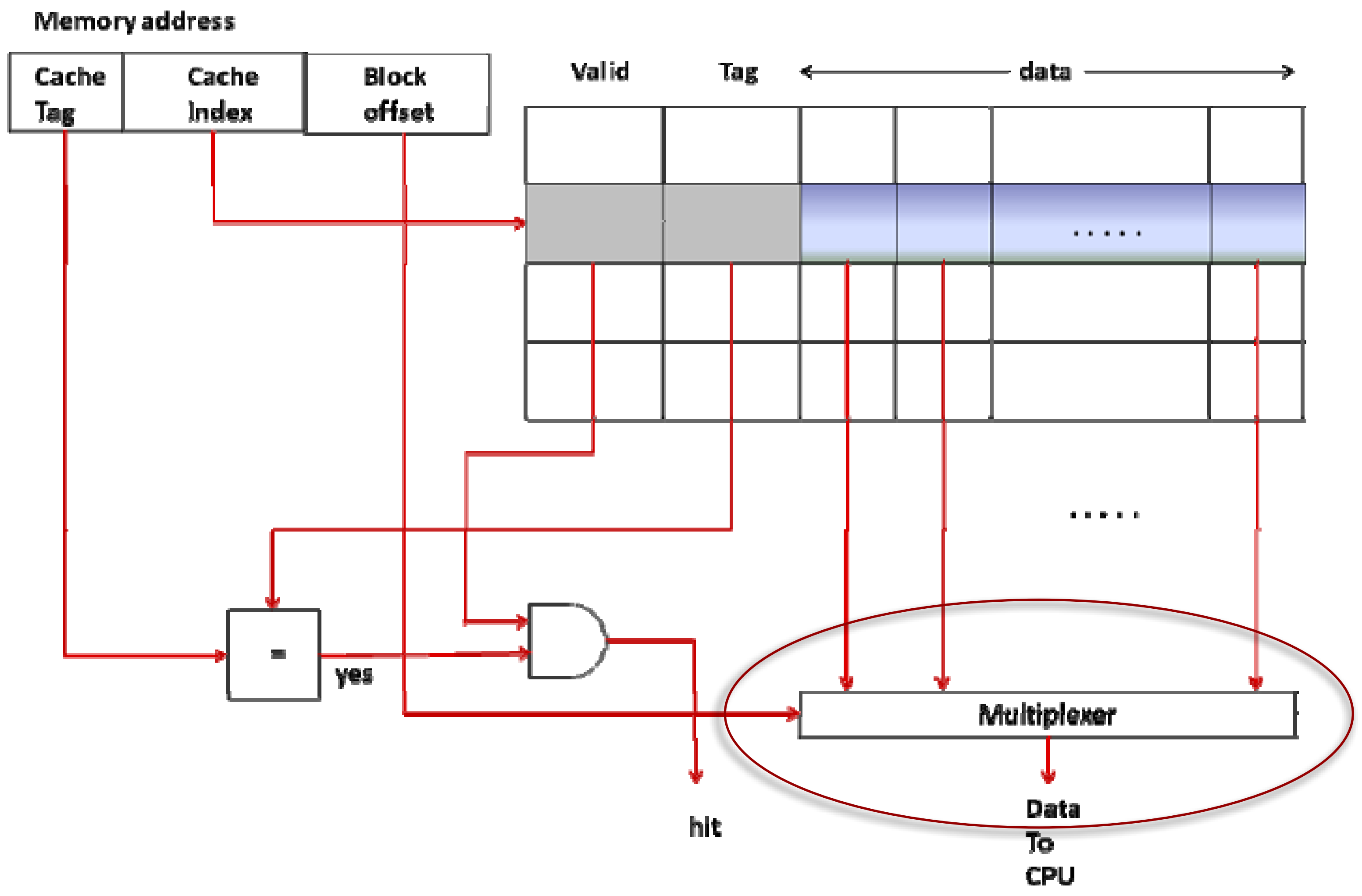
S = Size of cache; B = Block size; L = lines in cache



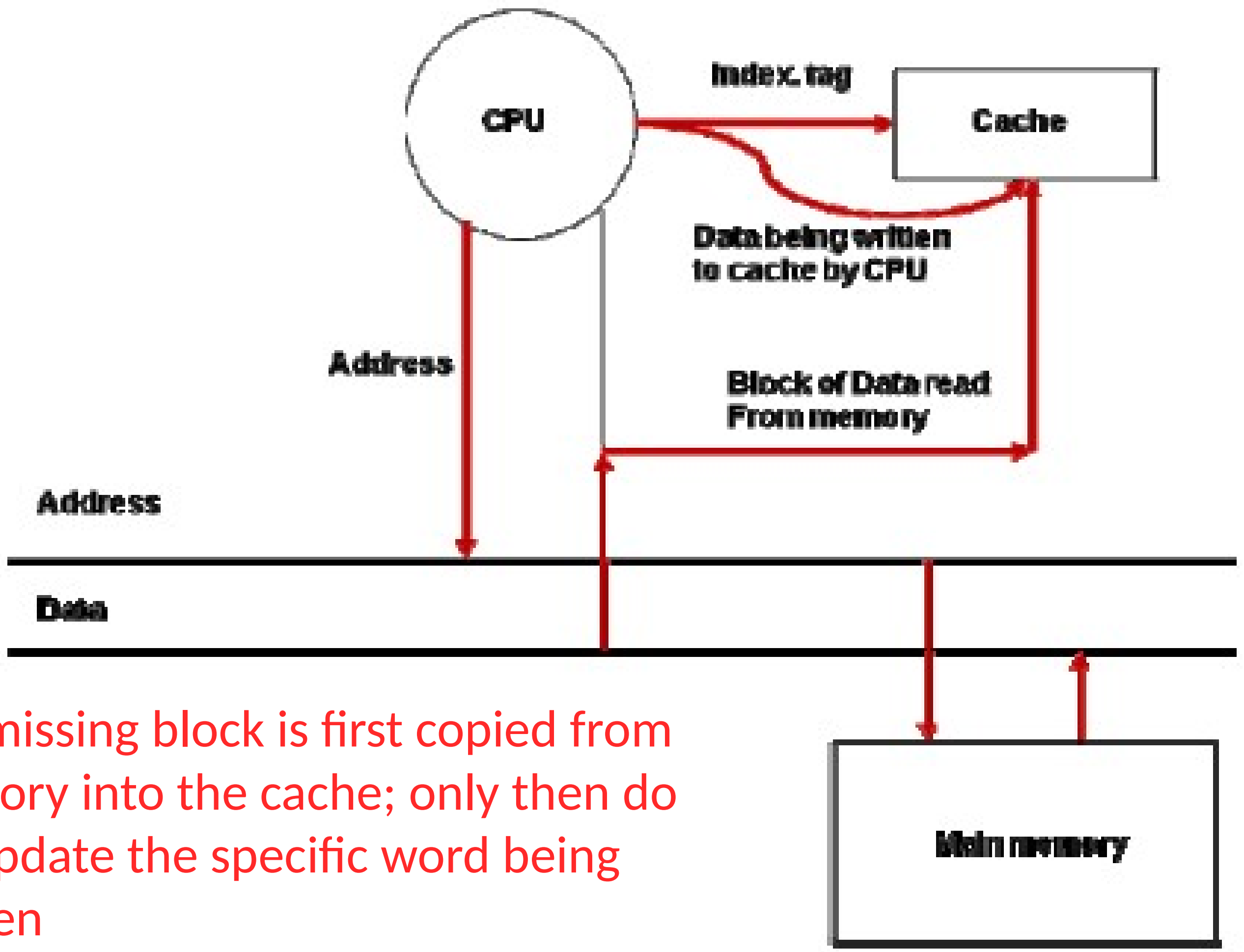
$$b = \log_2 B$$
$$L = S/B$$
$$n = \log_2 L$$
$$t = a - (b+n)$$

This is really just the word number within the block

Multi-word cache organization



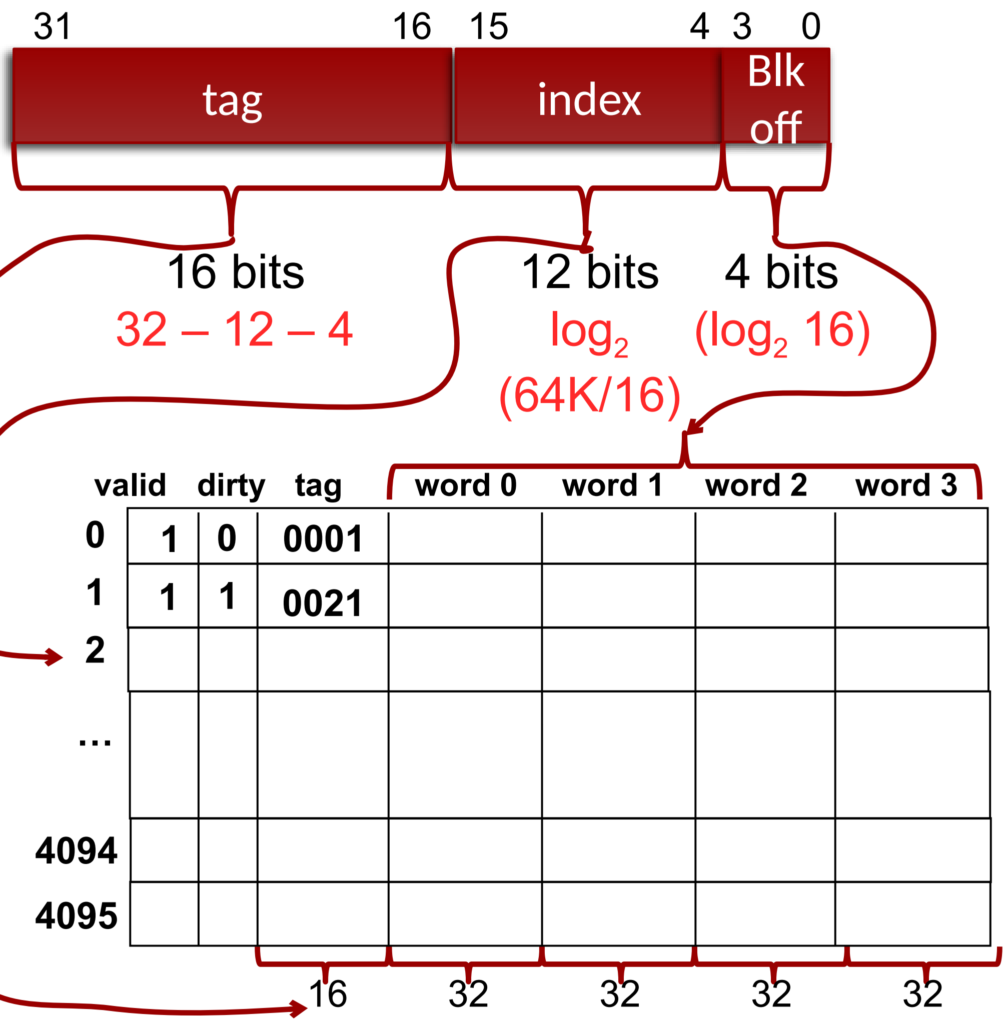
Write miss with multiple word cache



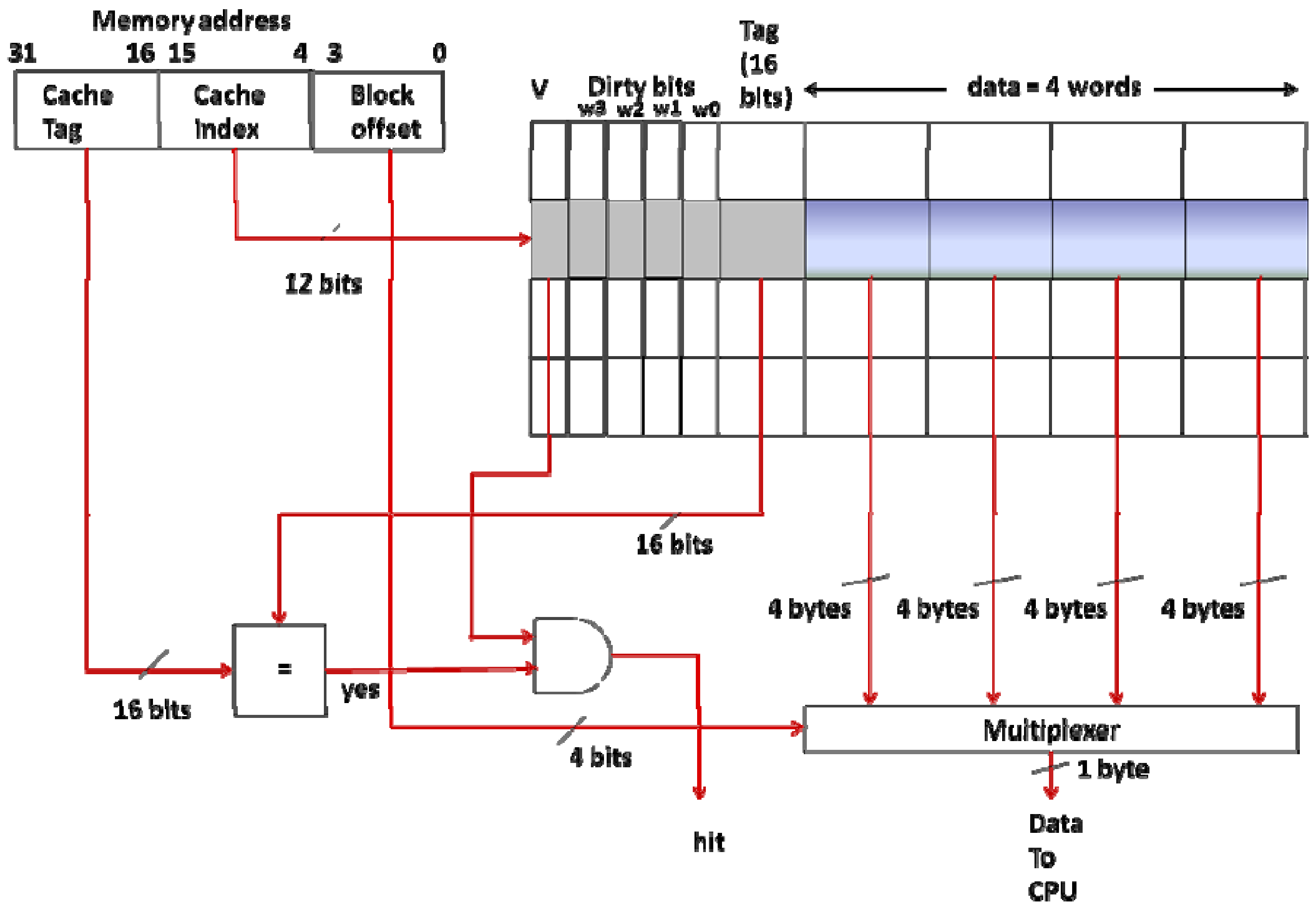
The missing block is first copied from memory into the cache; only then do we update the specific word being written

Multiple word cache example

- Direct-mapped cache
  - 32-bit byte-addressable memory address
  - Each memory word contains 4 bytes
  - Block size = 4 words (16 bytes)
  - A memory access brings in a block
  - 64K byte cache.
  - Write back cache with a dirty bit per word

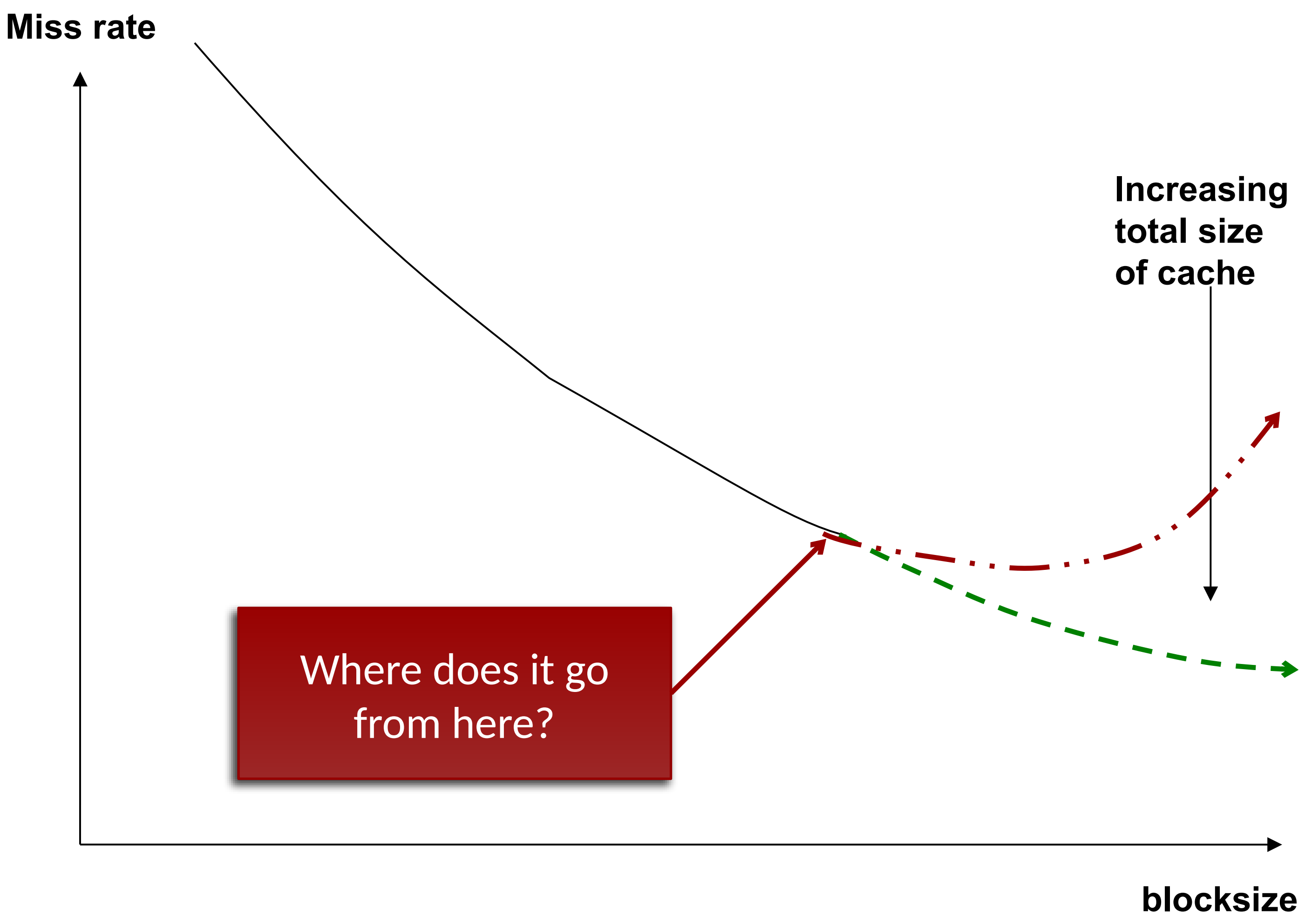


Implementing a multi word cache



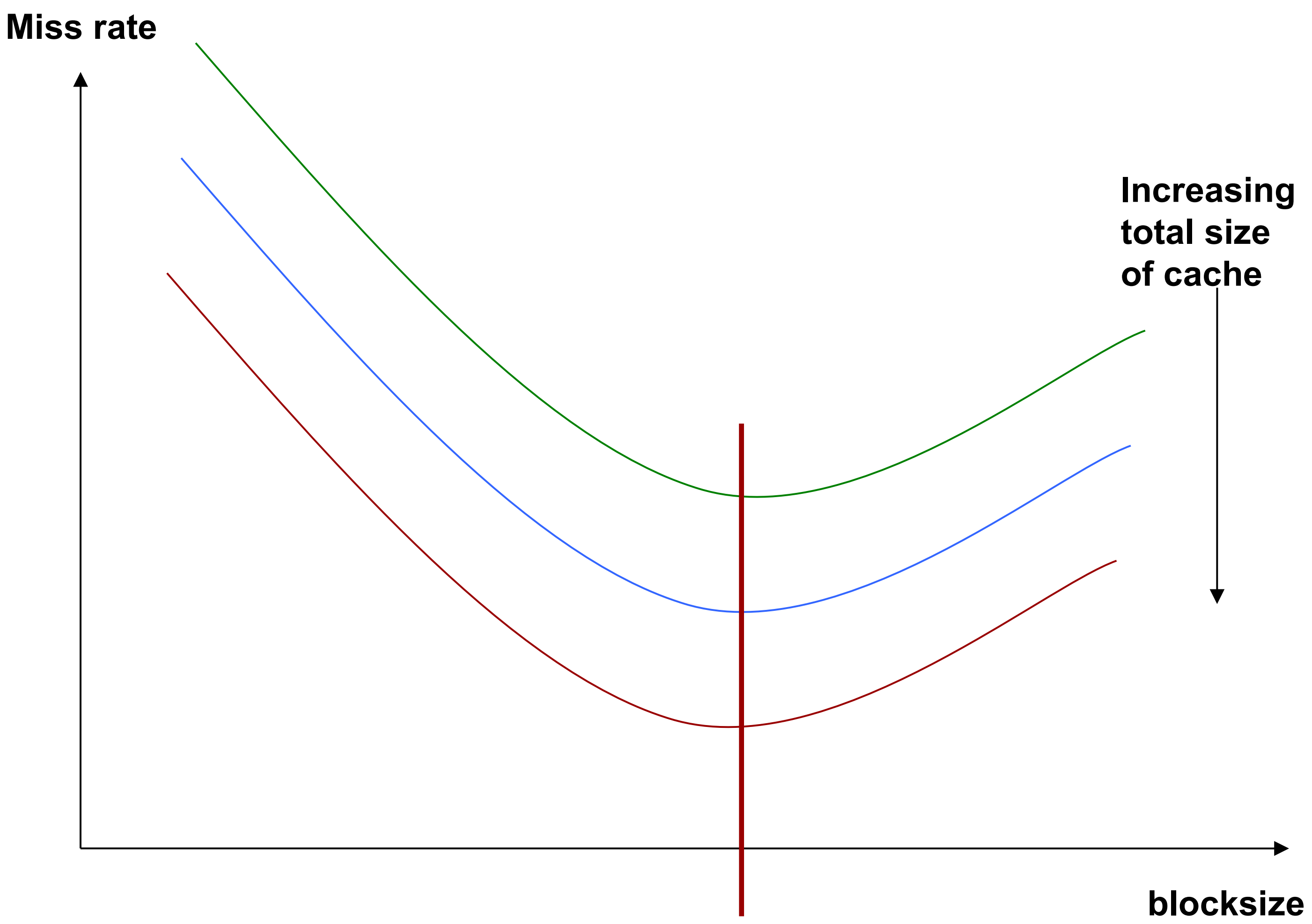
Increased block size?

- Exploits more spatial locality
- Reduces miss rate

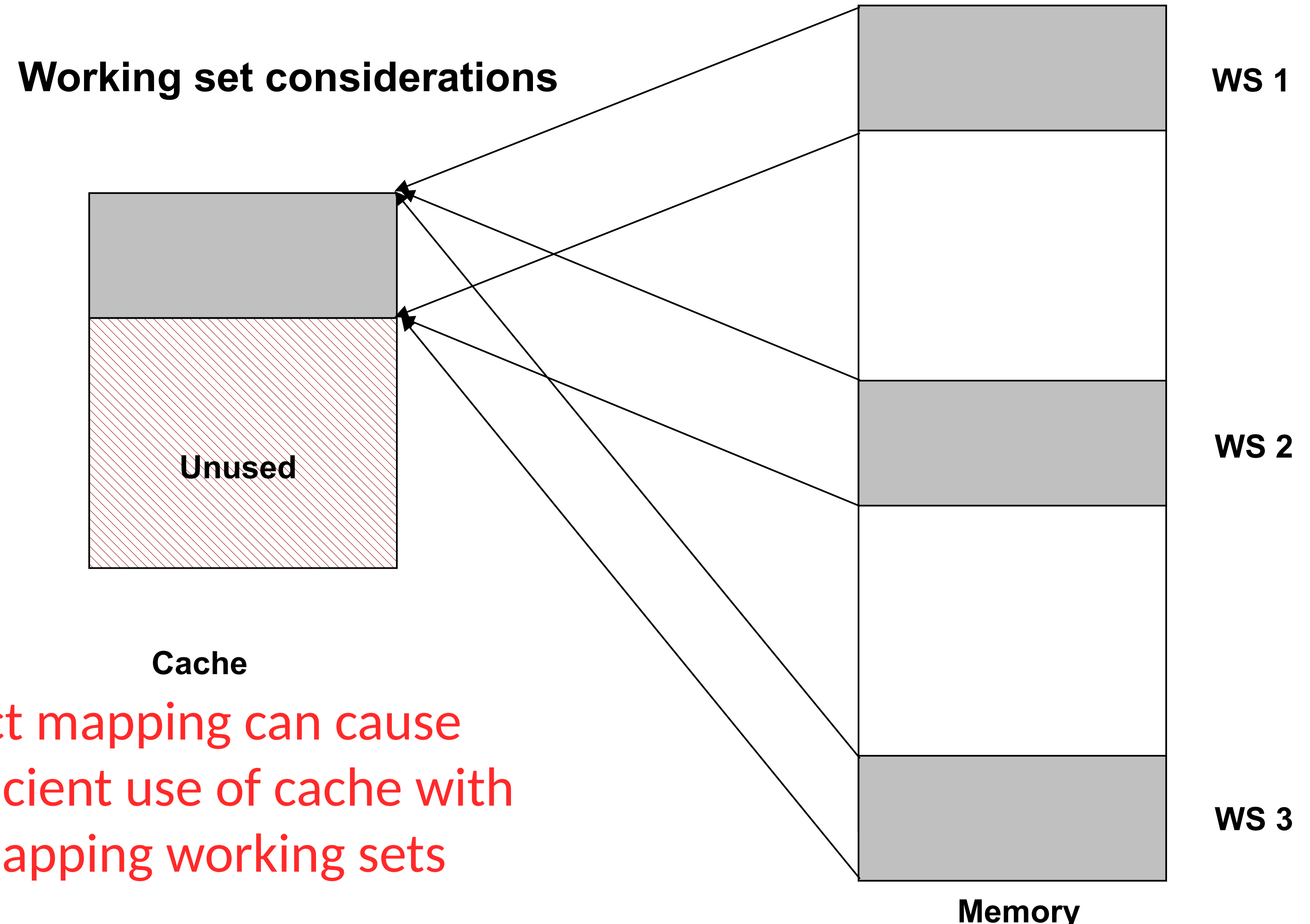


Increased block size?

- There is a point where things get worse
- When the working set changes, larger blocks have to be fetched
- Memory can only transfer so fast and it can become the bottleneck



Working set considerations



- Direct mapping can cause inefficient use of cache with overlapping working sets

What would be best?

- Allow any memory block to be brought into any cache block
- This is similar to be able to bring in a virtual page into any available physical page frame

Ⓟ Fully associative mapping

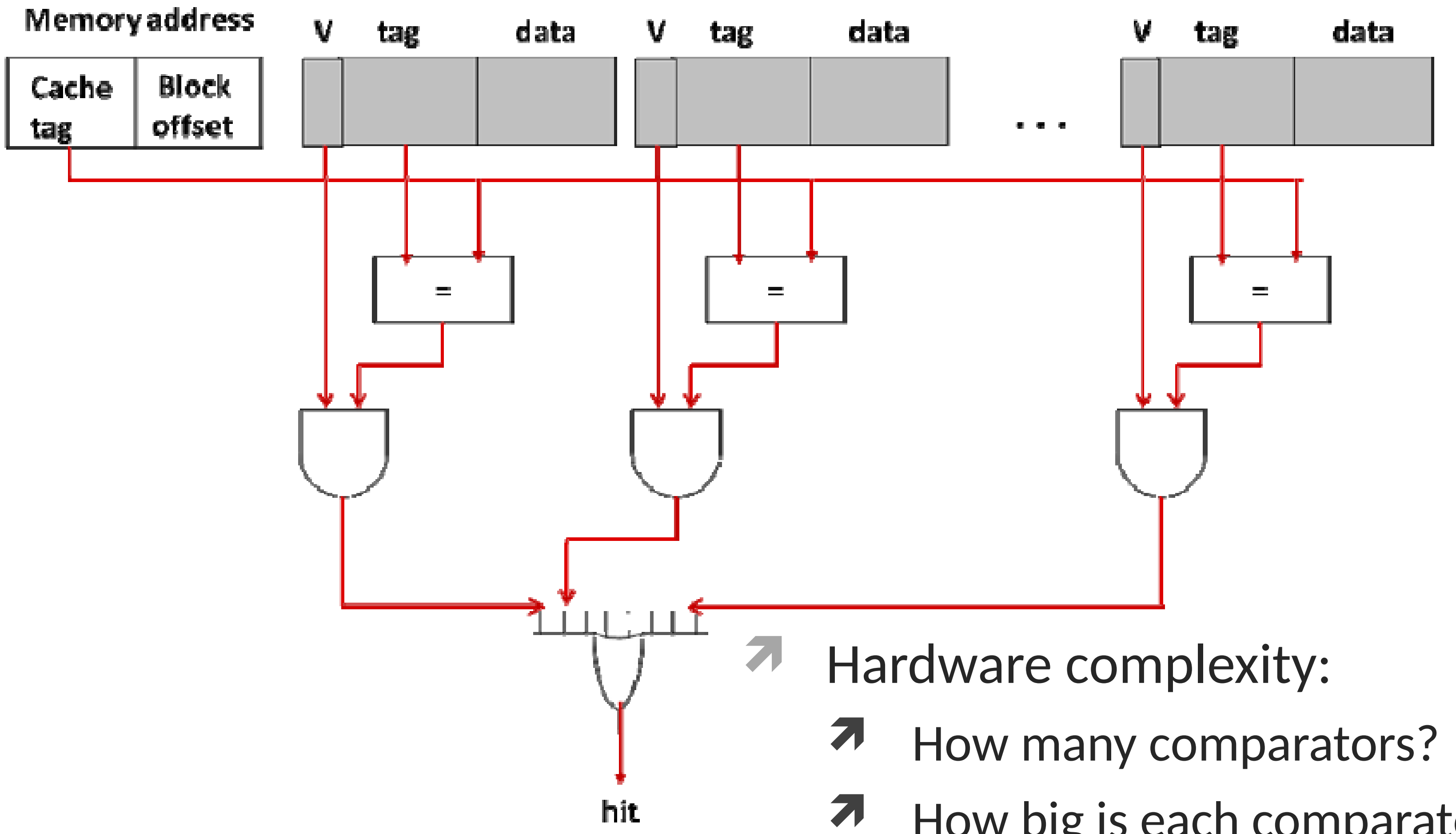
Address interpretation in FA cache

Cache Tag	**Index**

- No splitting memory addresses into "index" and "tag"
- It all becomes tag!

Cache Tag

Fully associative cache circuitry

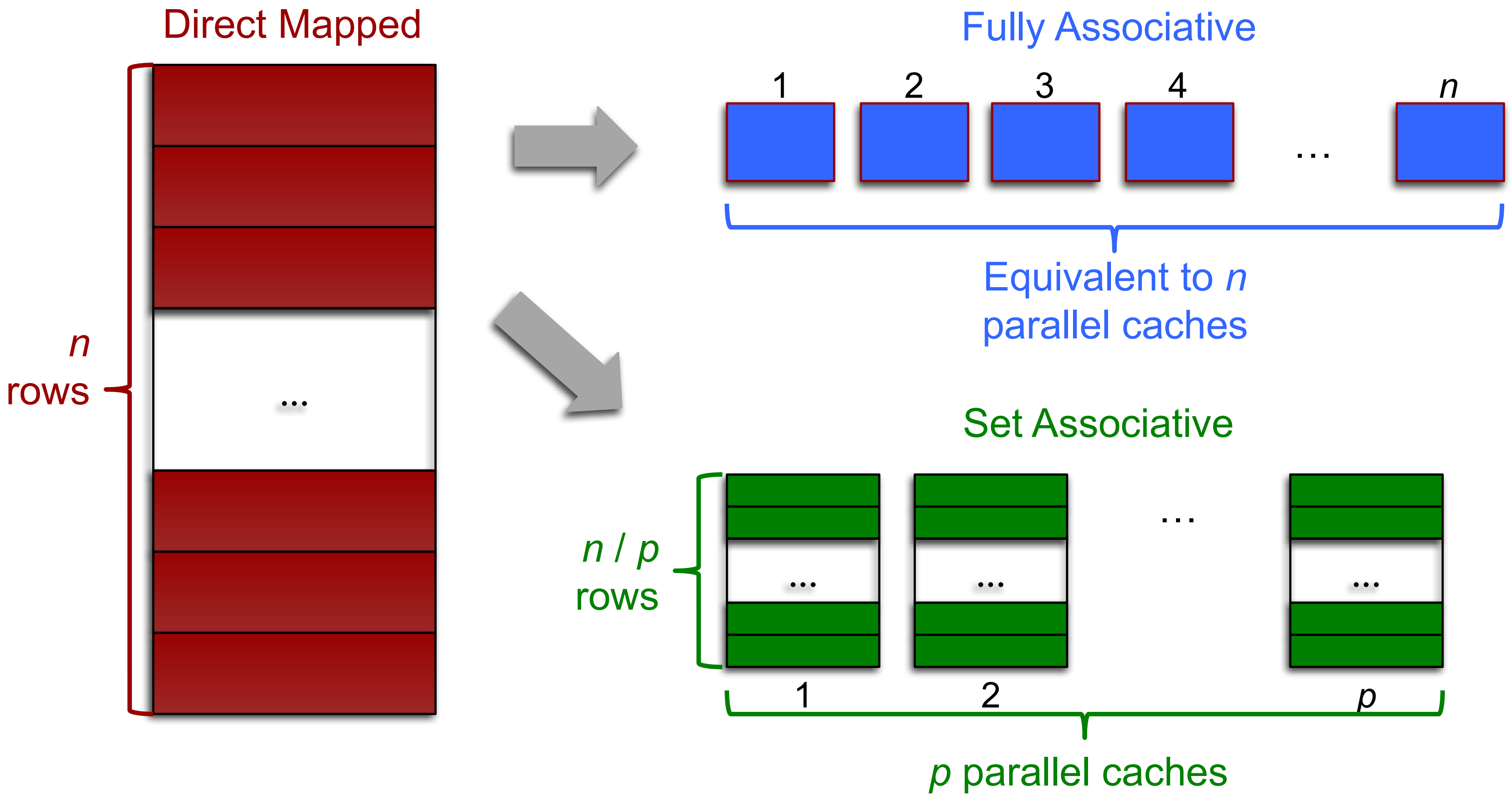


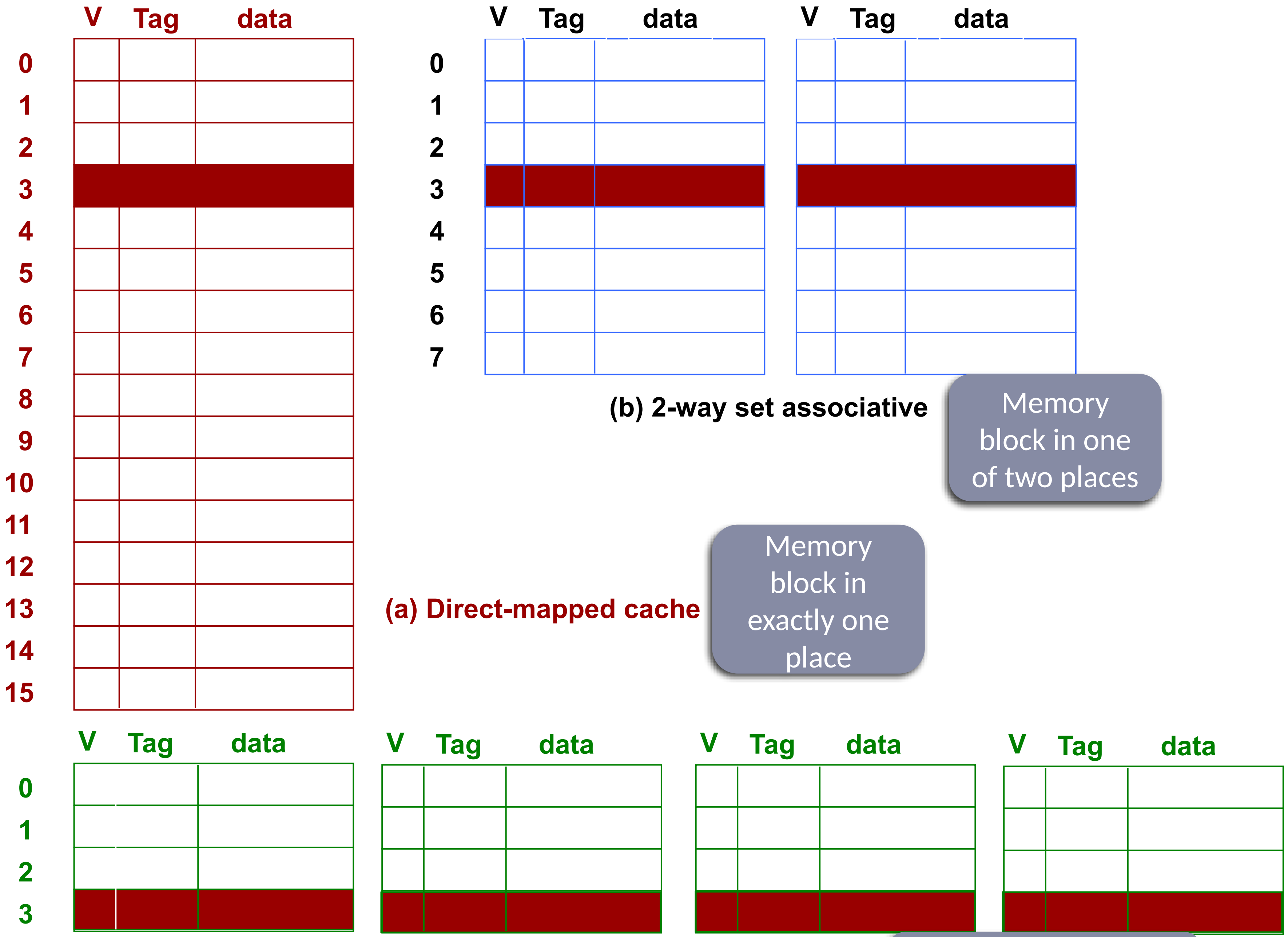
- Hardware complexity:
  - How many comparators?
  - How big is each comparator?

Cache organizations

- Fully associative cache Ⓟ
  - Too much hardware complexity
  - Most flexible
- Direct mapped cache Ⓟ
  - Least hardware complexity
  - Least flexible
- Can we do better? Is there a compromise?
- Yes! It's called a set-associative cache
- Turns out direct-mapped and fully-associative caches are degenerate cases of a set-associative cache!

Generalization?





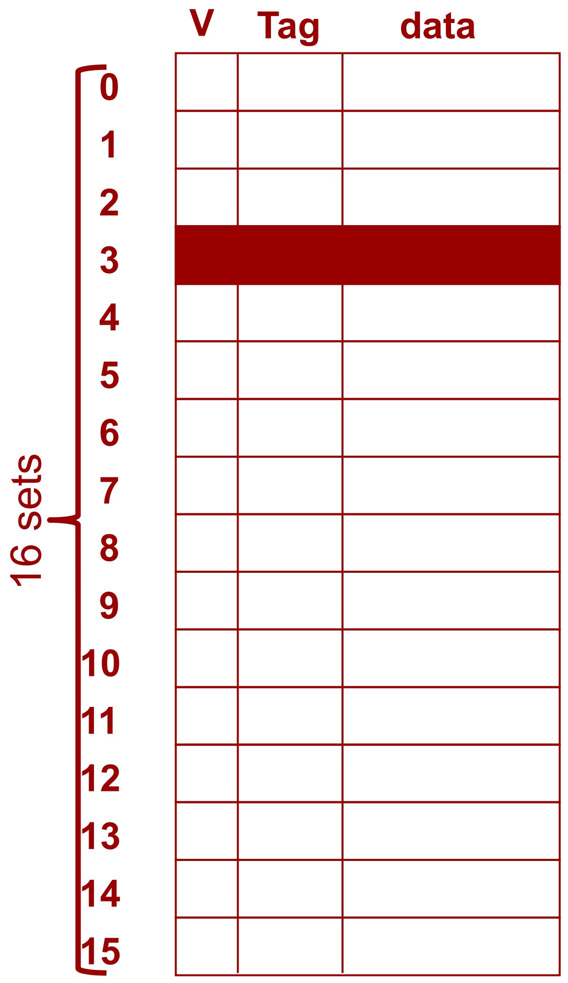
Memory block in one of two places

Memory block in exactly one place

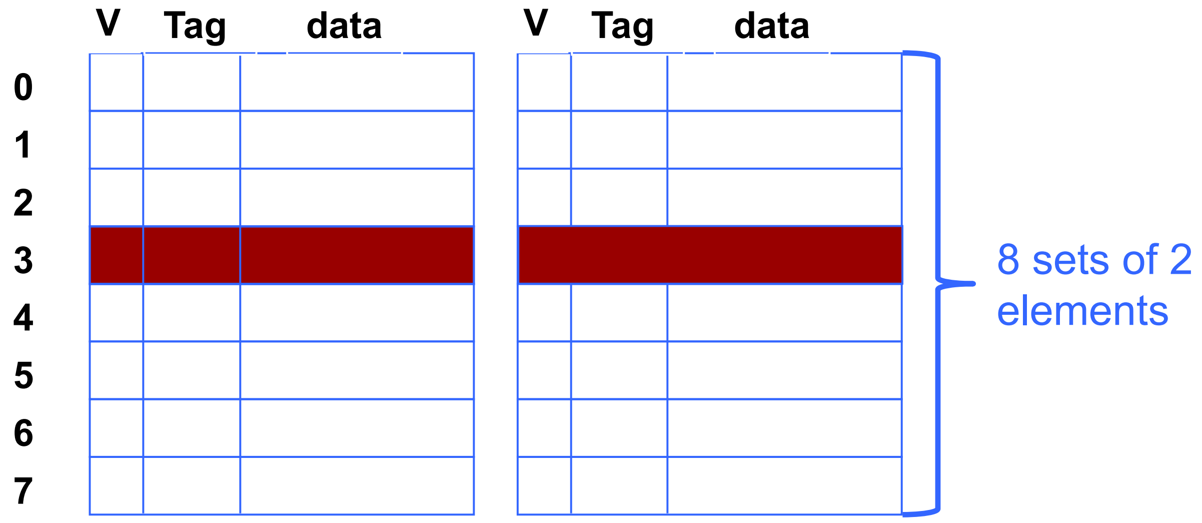
Memory block in one of four places

Synonyms

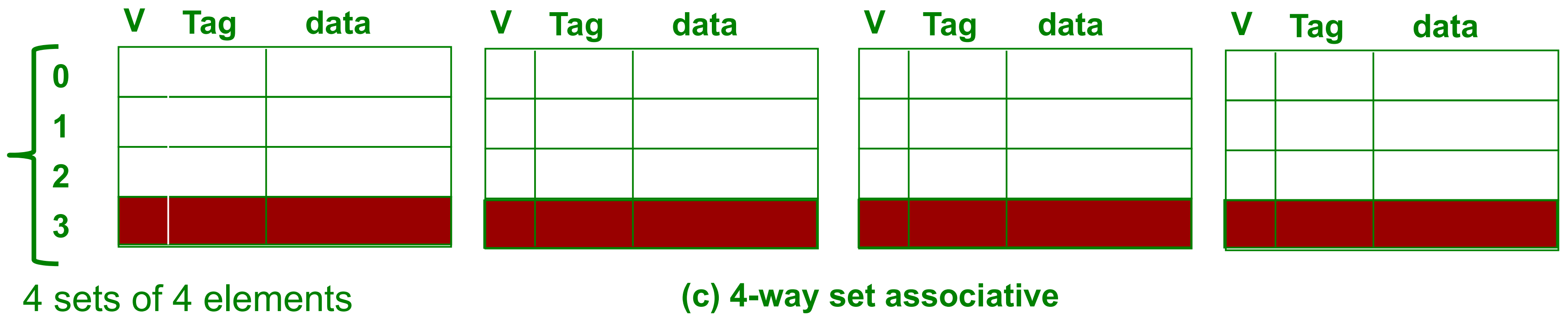
- Unfortunate but…
  - Four different ways of saying the same thing
    - Cache line
    - Cache block
    - Cache entry
    - Cache element
  - All mean the same thing…the basic unit of data transferred into/out of the cache at a time
  - The textbook has a couple of typos in which it implies *cache set* is also a synonym; that's wrong based on current usage as well as the text on 9-14 and 9-34!
- A cache set is a "row" in the cache. The number of blocks per set is determined by the type of the cache
  - Direct mapped: *n* sets, 1 element
  - P-way set associative: *n*/*p* sets, *p* elements
  - Fully associative: 1 set, *n* elements



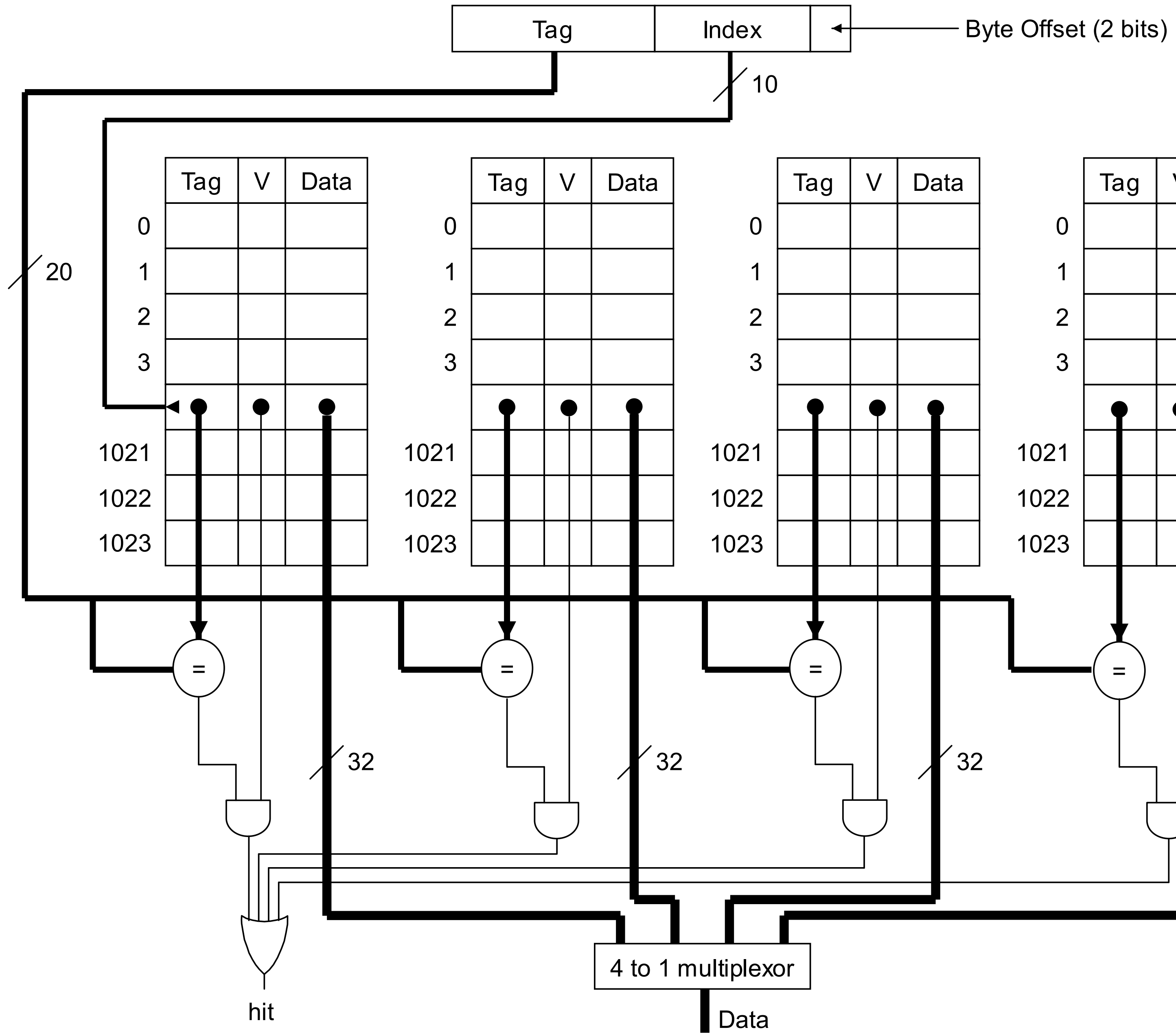
(a) Direct-mapped cache



(b) 2-way set associative

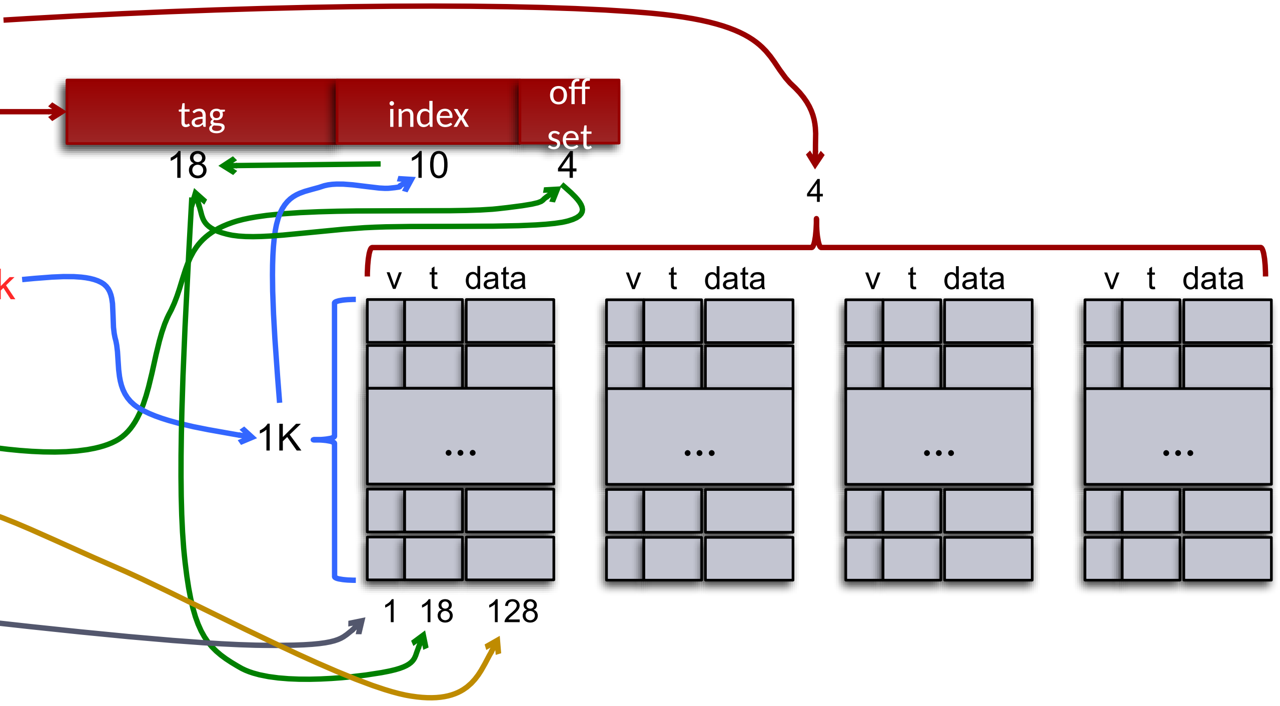


(c) 4-way set associative


Tag
Index
Byte Offset (2 bits)
10
20
Tag
V
Data
0
1
2
3
1021
1022
1023
=
32
hit
4 to 1 multiplexor
Data

Example: 4-way set associative cache

- 4-way set associative cache
- 32-bit memory, byte-addressable
- Cache size of 64 Kbytes. 64K/4/16 = 1k
- Each memory word contains 4 bytes
- Cache block size is 16 bytes. 16*8 = 128
- Write-through policy
- One valid bit per block.



Compute the total amount of storage for implementing the cache (i.e. actual data plus the meta-data)

(1+18+128) * 1K * 4 / 8

= 75,264 bytes

Hardware complexity?

4 18-bit comparators

In a fully associative cache, …

…with 64K bytes of data, 64 bytes / block and a *t*-bit tag

A. There are four *t*-bit tag comparators
B. There are 64 *t*-bit tag comparators
[✓ C. There are 1k *t*-bit tag comparators]
D. There is one *t*-bit tag comparator for the entire cache

Today's number is 11,111

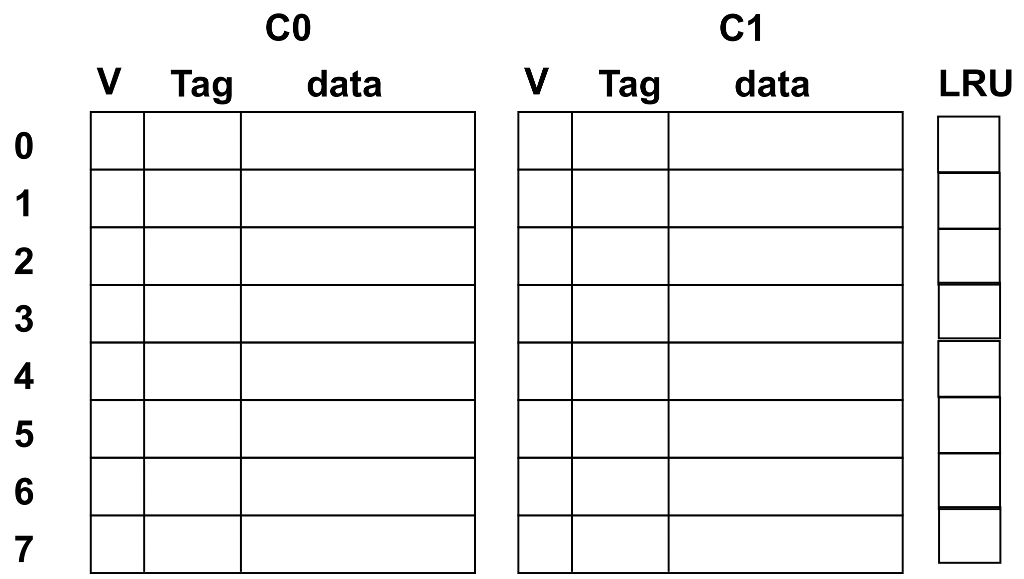
FA caches have one comparator for each block/line/entry in the cache. That means 64K/64 = 1K is the number of comparators.

In a 4-way set associative cache, ...

...with 64K bytes of data, 64 bytes / block, with a *t*-bit tag

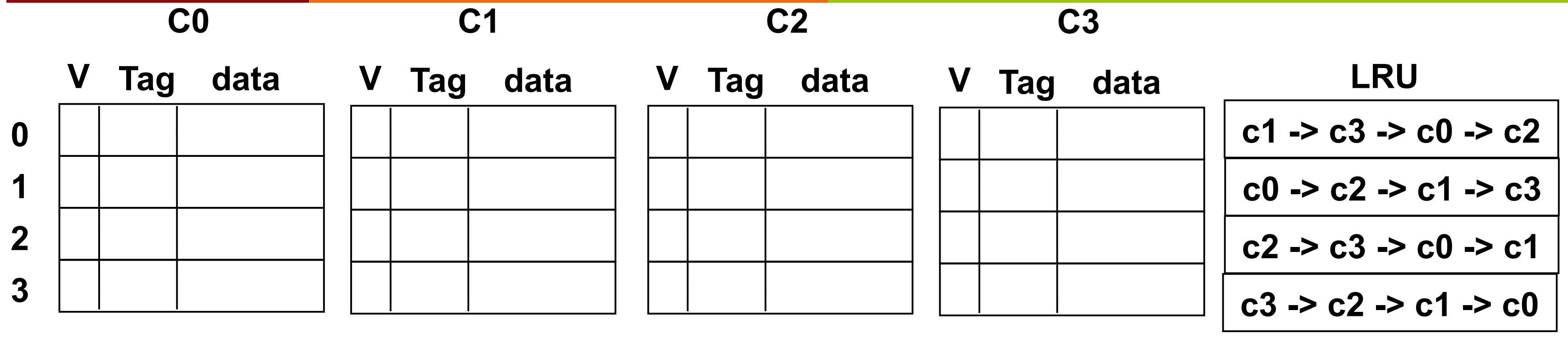
A. [✓ There are four *t*-bit tag comparators]
B. There are 64 *t*-bit tag comparators
C. There are 1k *t*-bit tag comparators
D. There is one *t*-bit tag comparator for the entire cache

What about cache replacement policy?



- What kind of cache is this? 2-way set associative
- How many LRU bits do we need? Just one.
- What happens on every memory access
  Set LRU to 0/1 if we read from/store in C0/C1
- So what do we do with a 4-way set associative cache?

LRU replacement in a 4-way cache



- Do we need a state machine for each cache line?
- Using as many state machines as the number of rows in the cache is a lot of hardware
- Each state machine Ⓟ 4! States Ⓟ 5 bit state register

What happens on a context switch?

- TLB? Flush user portion
- Cache? Nothing! We're using physical addresses

One caveat

Note: If the OS brings in a page from disk directly to a physical page frame and bypasses the cache, it must flush any cache locations for the previous contents. This isn't a context switch issue, it's an issue that if I/O bypasses the cache, then any cache entries referencing the I/O buffer are definitely invalid.

On a process context switch

A. The cache entries for the process being suspended are flushed.
B. The cache and TLB entries for the process being suspended are flushed
C. [✓ The TLB entries for the process being suspended are flushed]
D. None of the above
E. What does bathroom plumbing have to do with computer science?

Today’s number is 11,111

When a page is evicted from a page frame in memory

A. The cache entries for the evicted page are flushed.

B. The cache and TLB entries for the evicted page are flushed

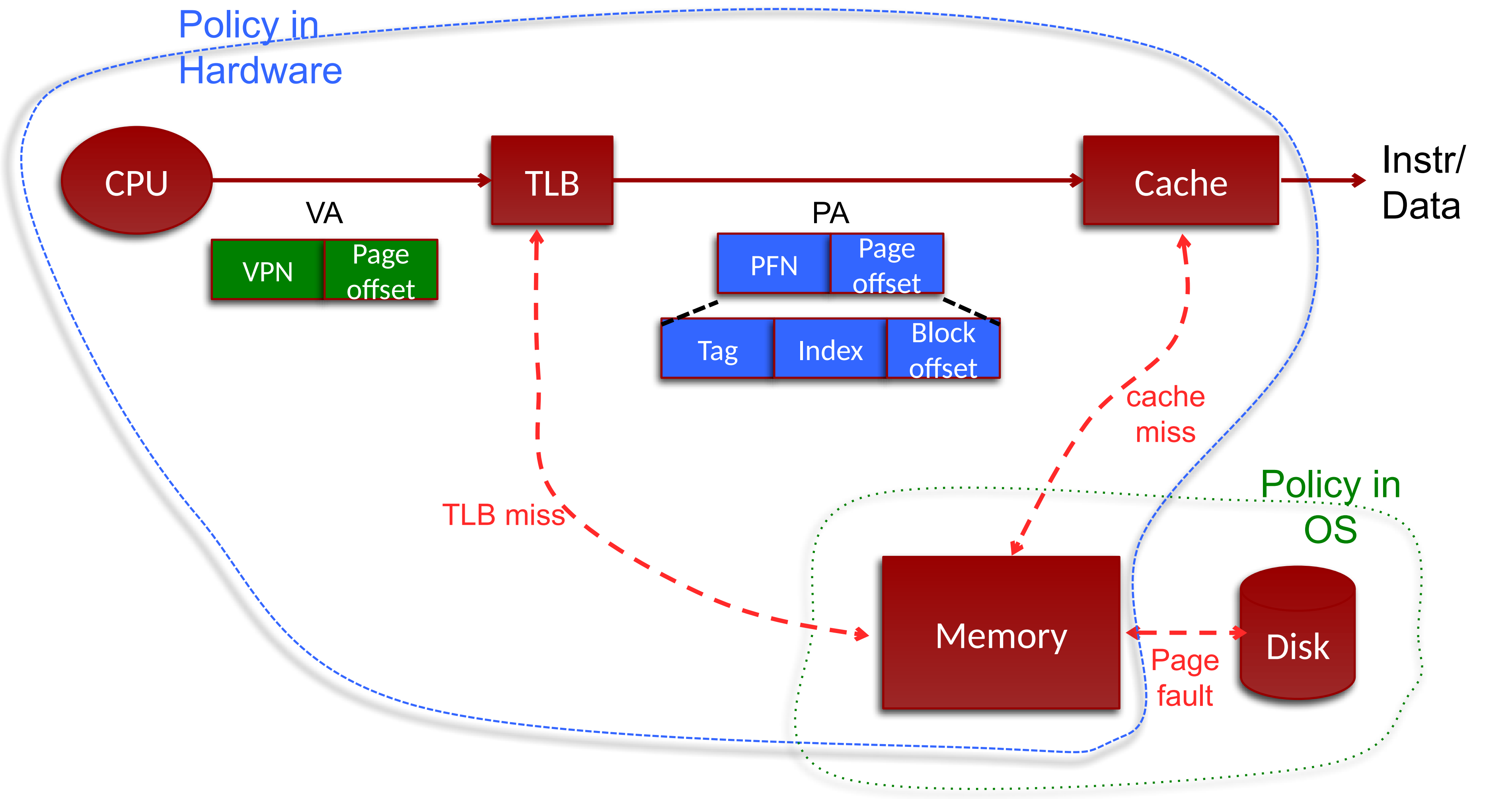
[✓ C. The TLB entries for the evicted page are flushed]

D. None of the above

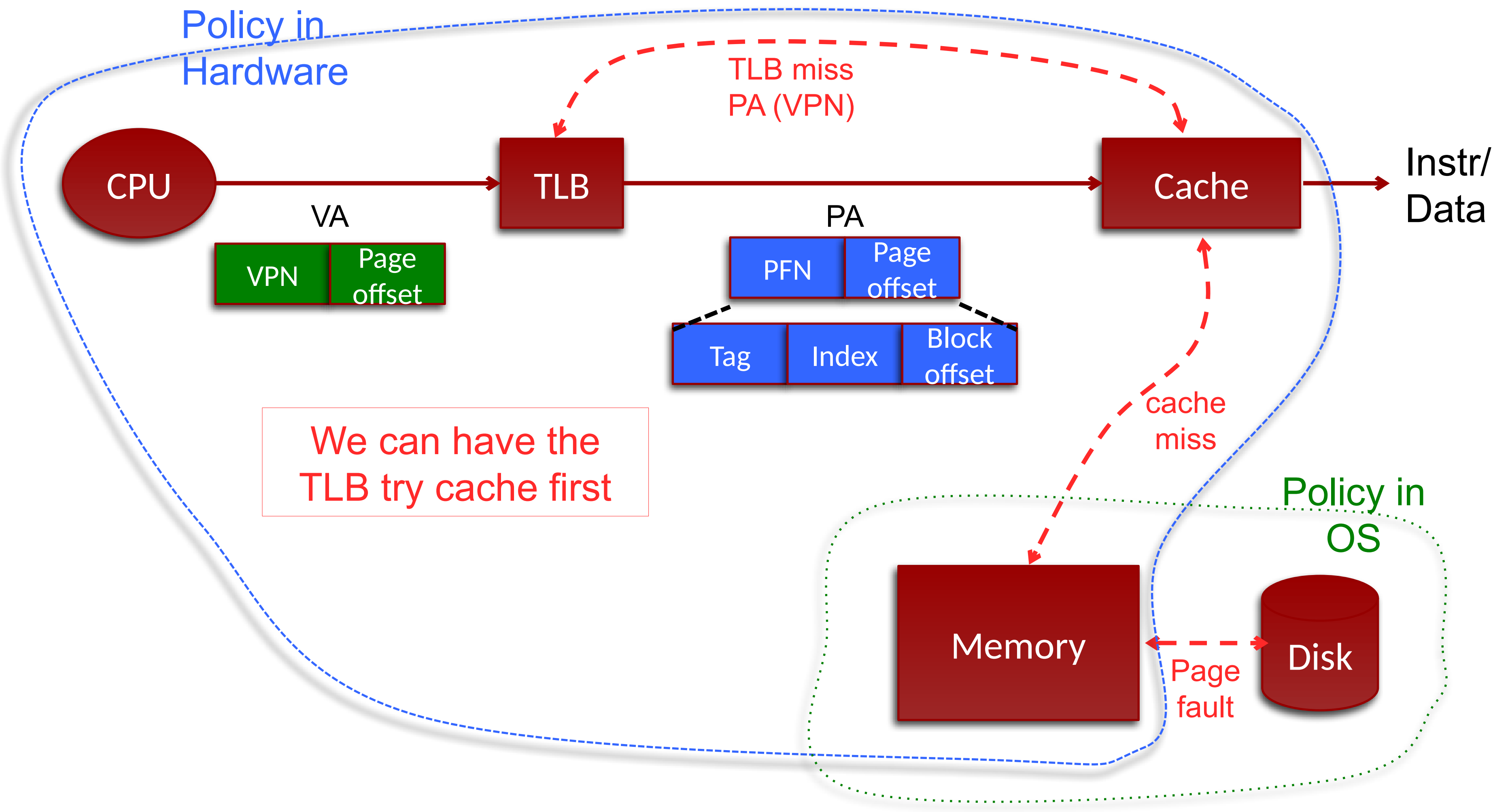
E. Still fixated on bathroom plumbing, huh?

The caches we've studied so far hold physical addresses (address translation has already occurred before the cache gets involved), then no flush is needed on page replacement. If the cache holds virtual addresses, then flushing cache entries from the page is required.

Putting cache and VM together

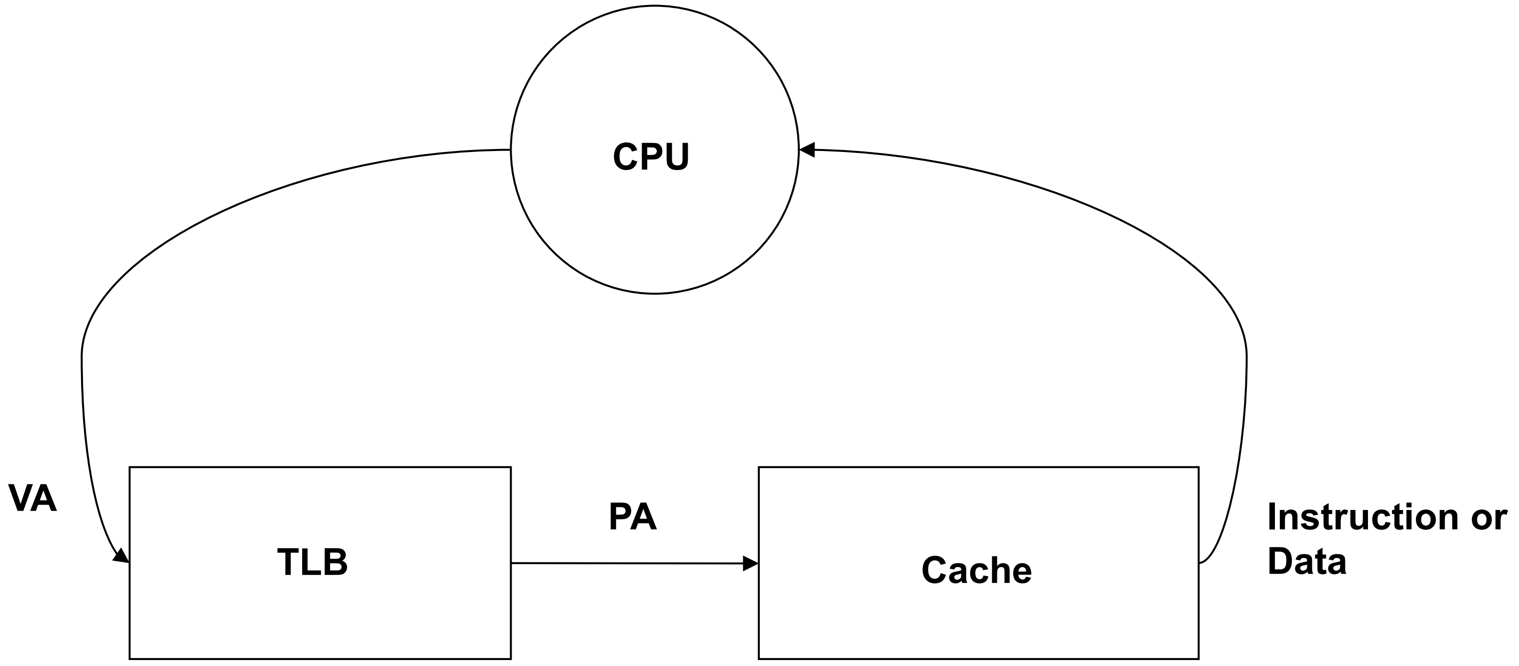


Adding some speed



We can have the TLB try cache first

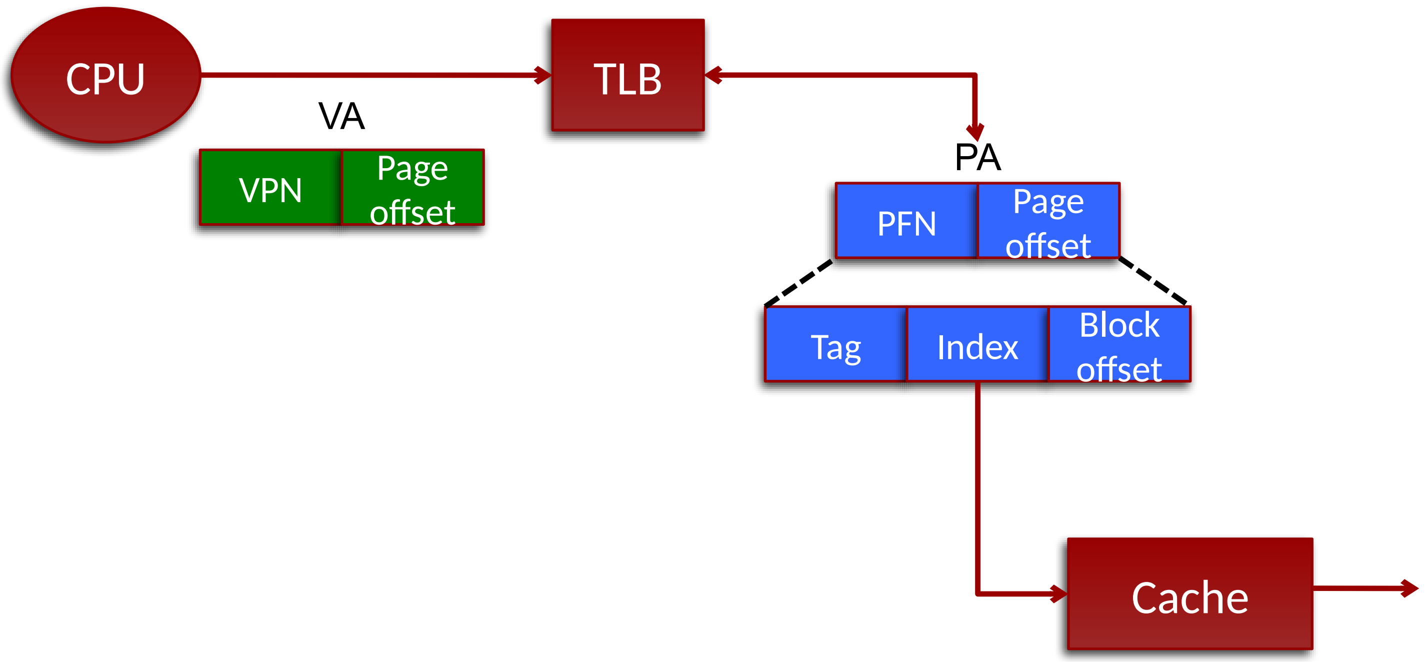
Putting cache and TLB together



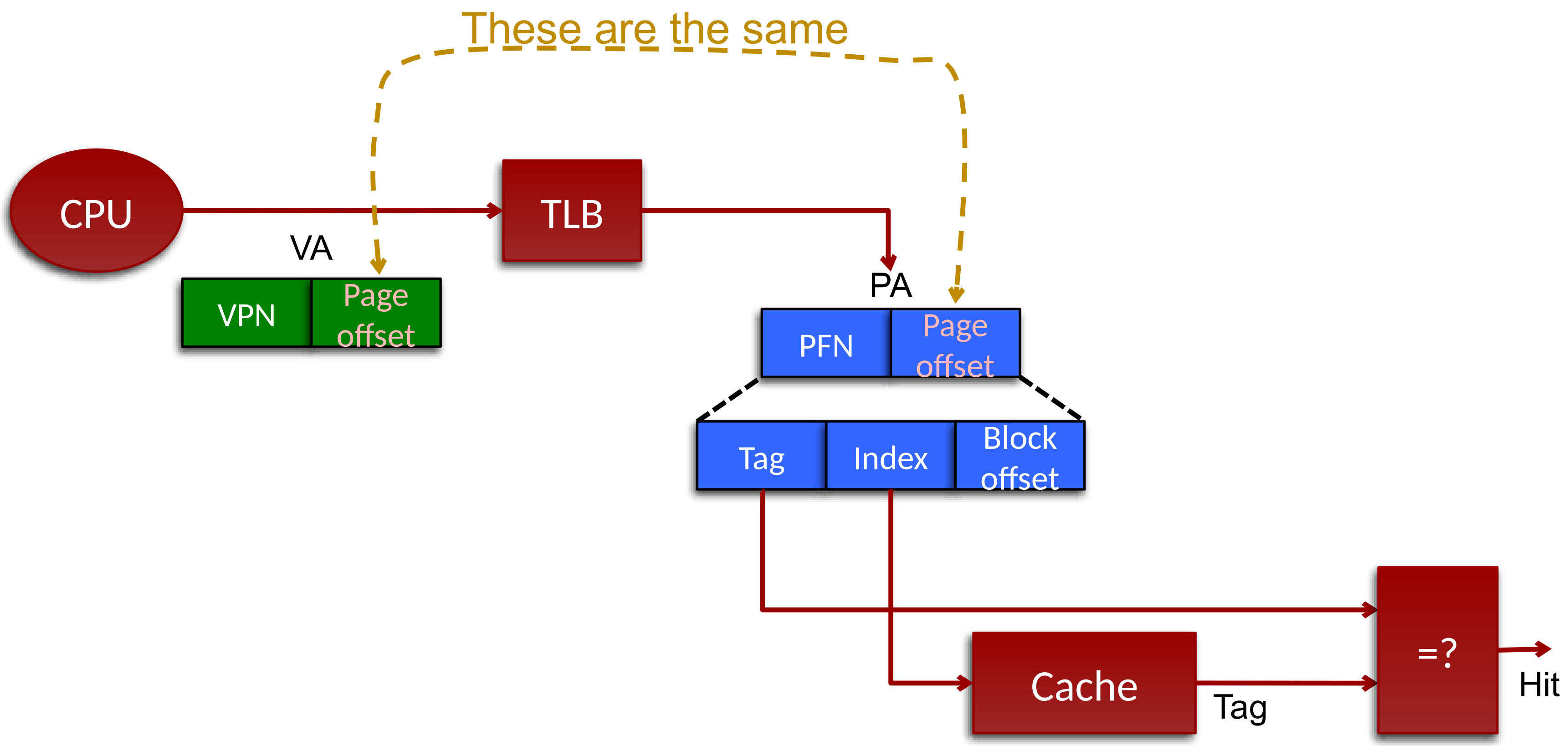
- TLB access is in the critical path of cache access Ⓟ increased clock cycle time

- Can we remove TLB from the critical path?

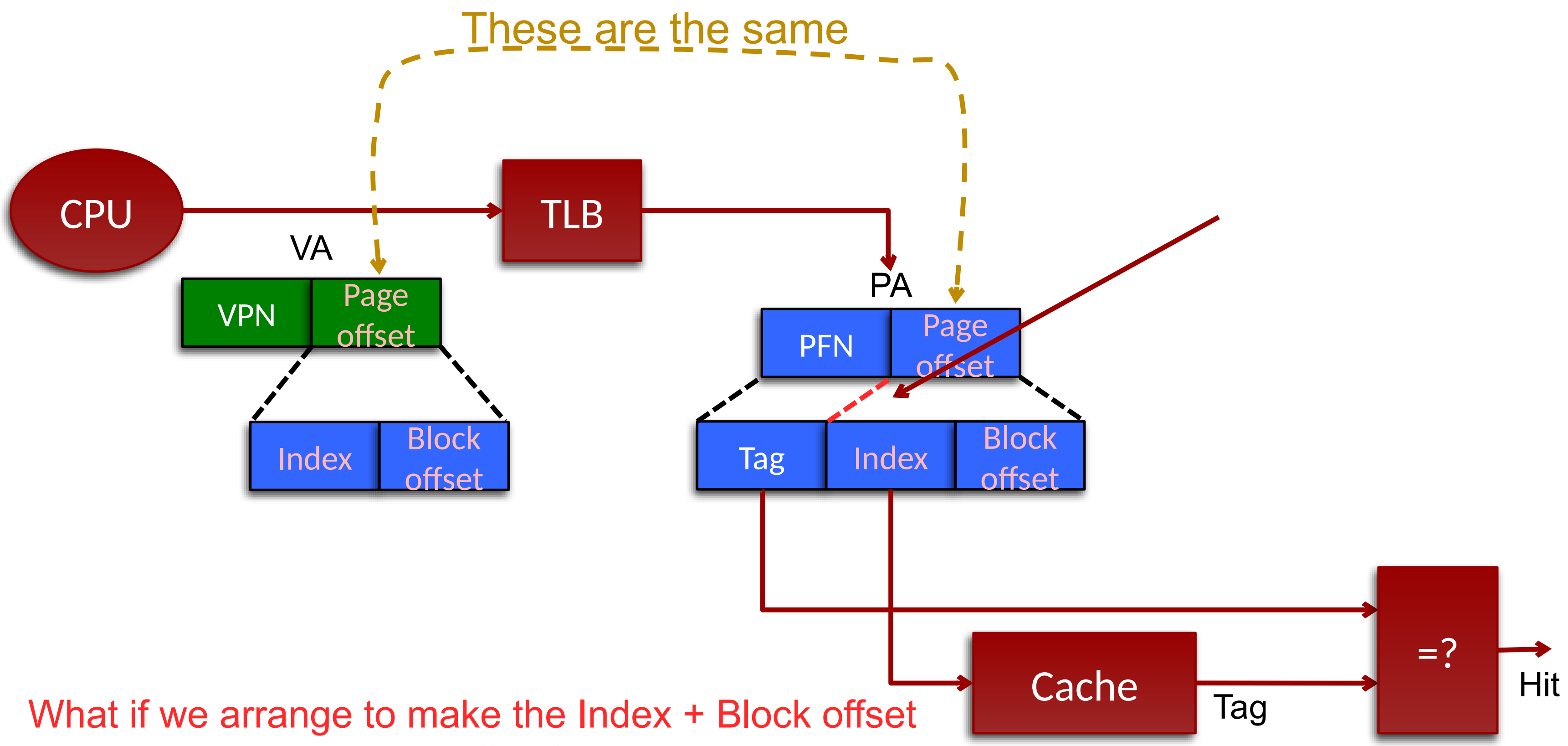
Recall how TLB and Cache work together



Recall

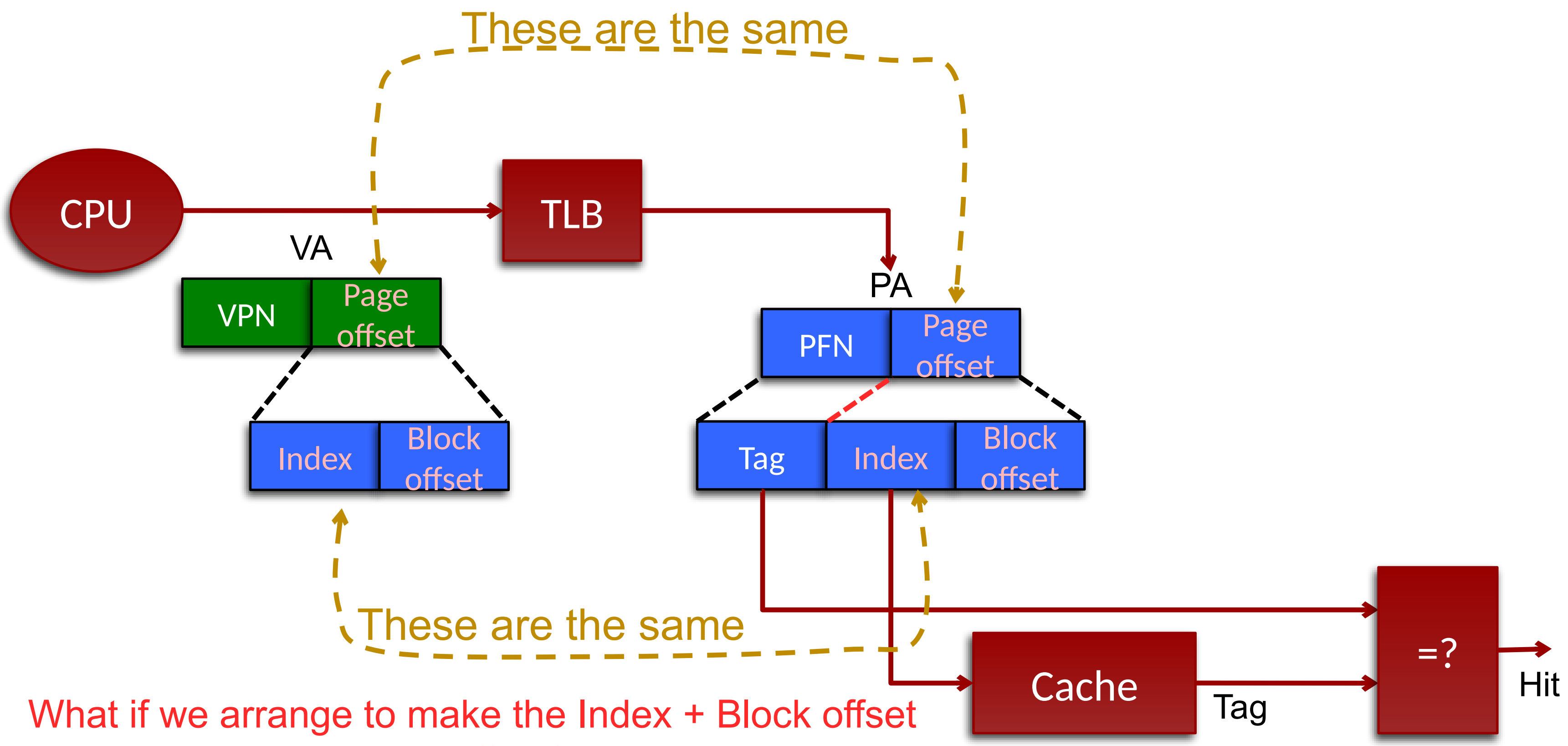

These are the same
CPU
TLB
VA
VPN
Page offset
PA
PFN
Page offset
Tag
Index
Block offset
Cache
Tag
=?
Hit

Make the cache index the same size as the page offset?



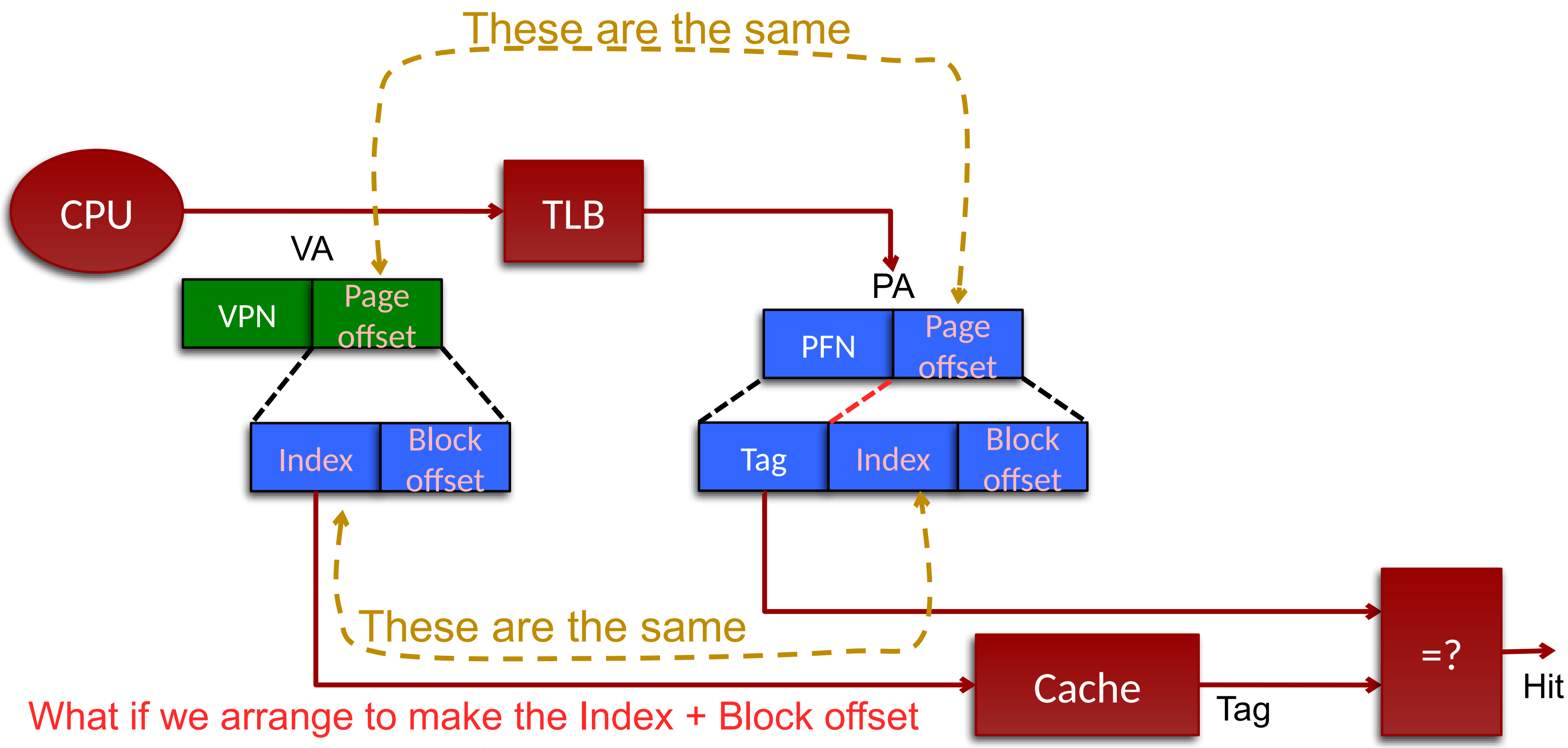
What if we arrange to make the Index + Block offset the same width as Page offset?

Now Index is the same for VA and PA!



What if we arrange to make the Index + Block offset the same width as Page offset?

Cache and TLB can start in parallel!



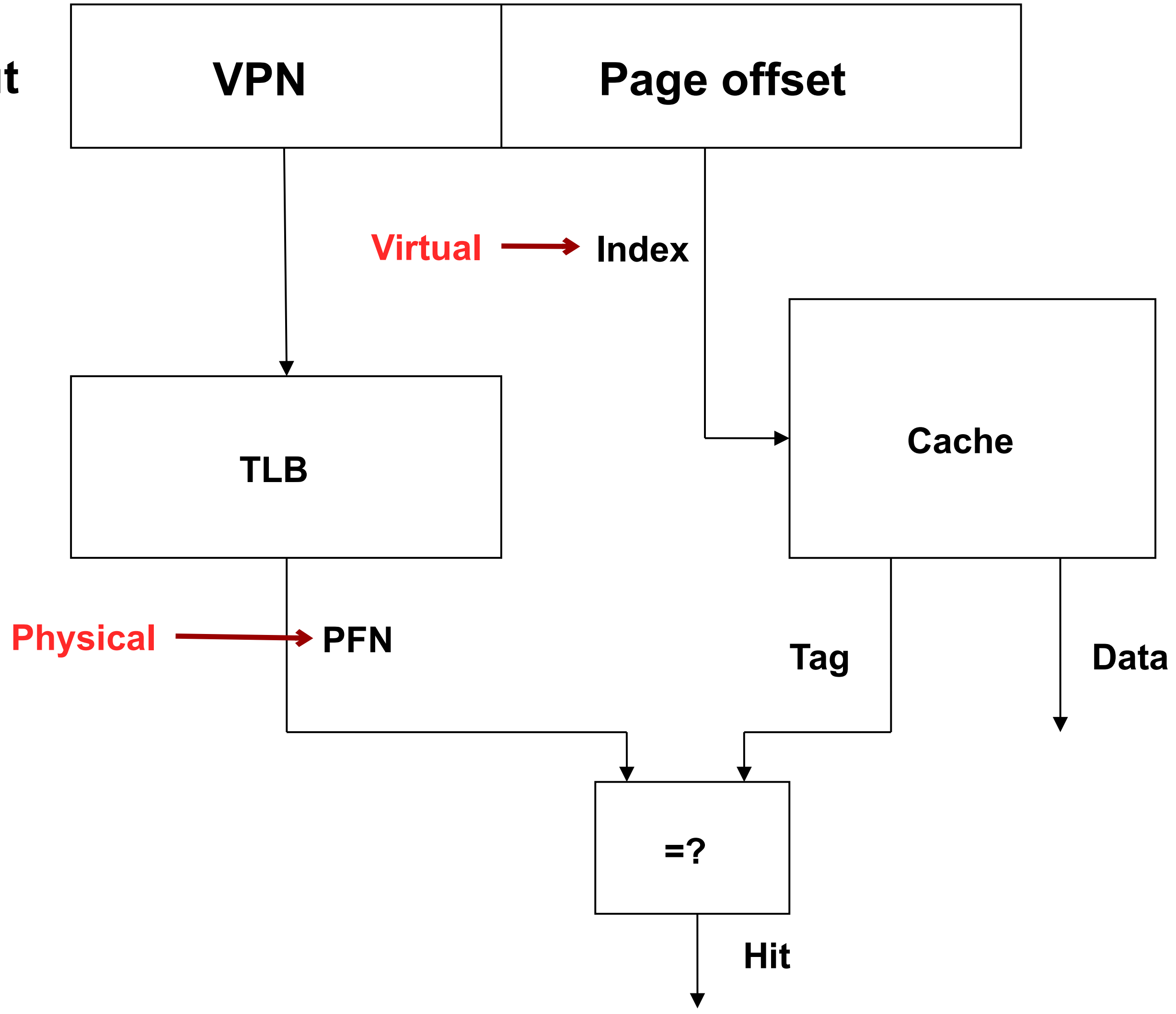
What if we arrange to make the Index + Block offset the same width as Page offset?

Then we can get the cache index from the VA to start the cache read without waiting for the TLB!

Virtually indexed physically tagged cache

How to get TLB out of the critical path

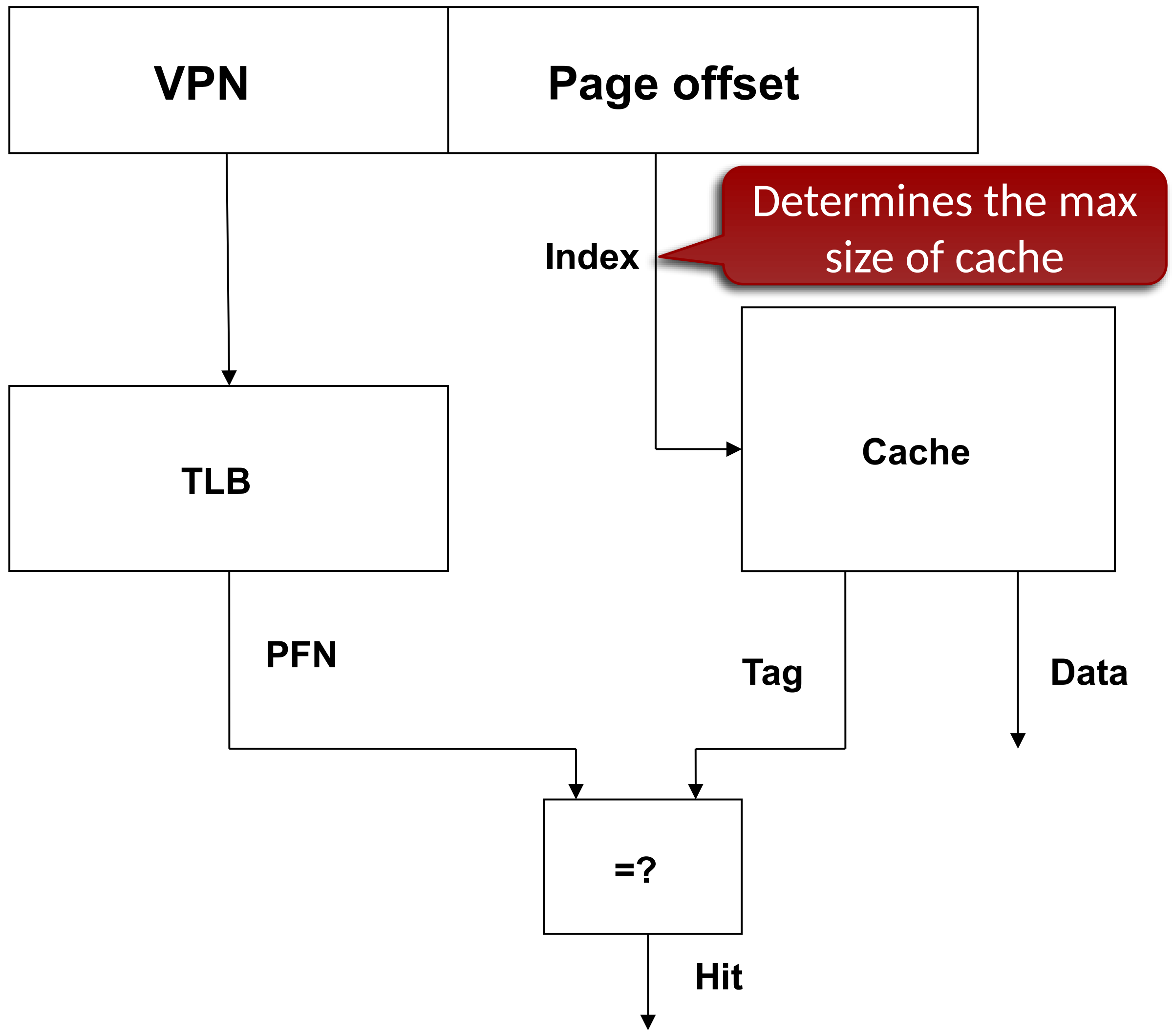
TLB & cache access in parallel



In a virtually indexed physically tagged cache

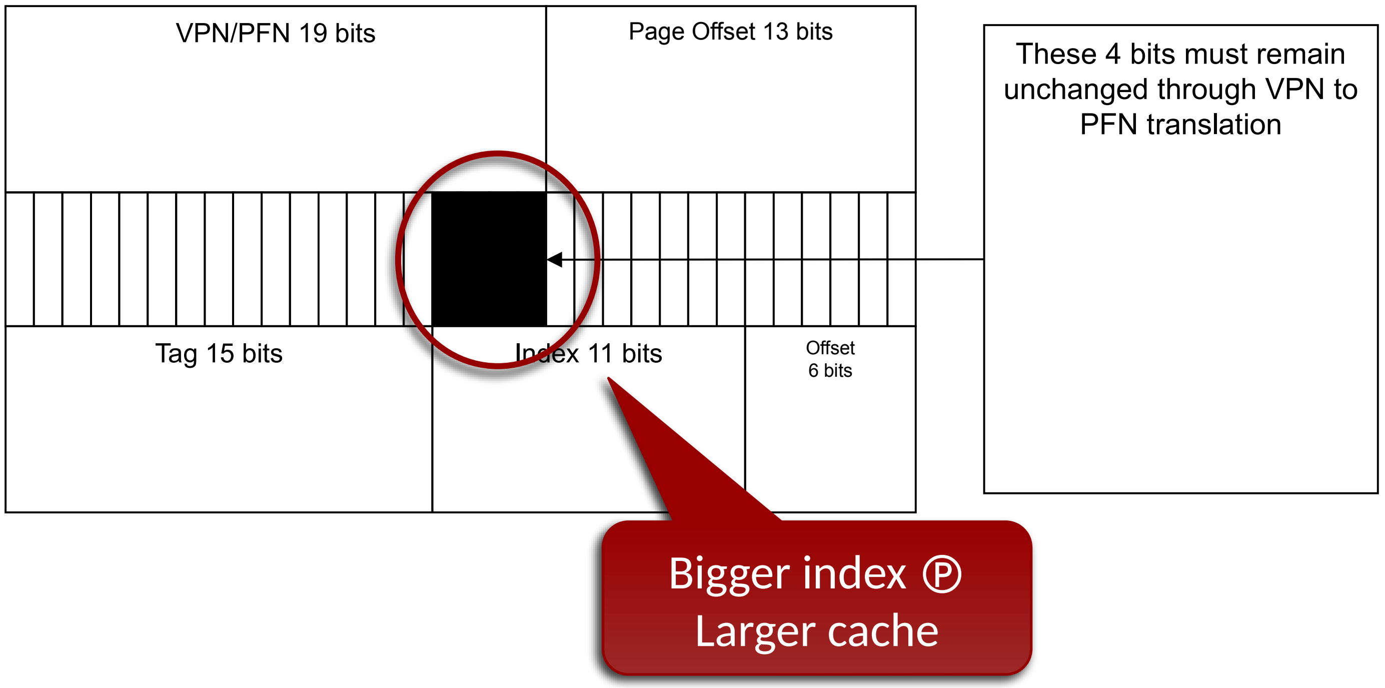
[✓ A. The TLB and cache are accessed in parallel]
B. The TLB and cache are accessed sequentially
C. The TLB and cache are accessed at random
D. None of the above

Is there a limitation?



Page coloring example

OS guarantees some bits of VPN will remain unchanged

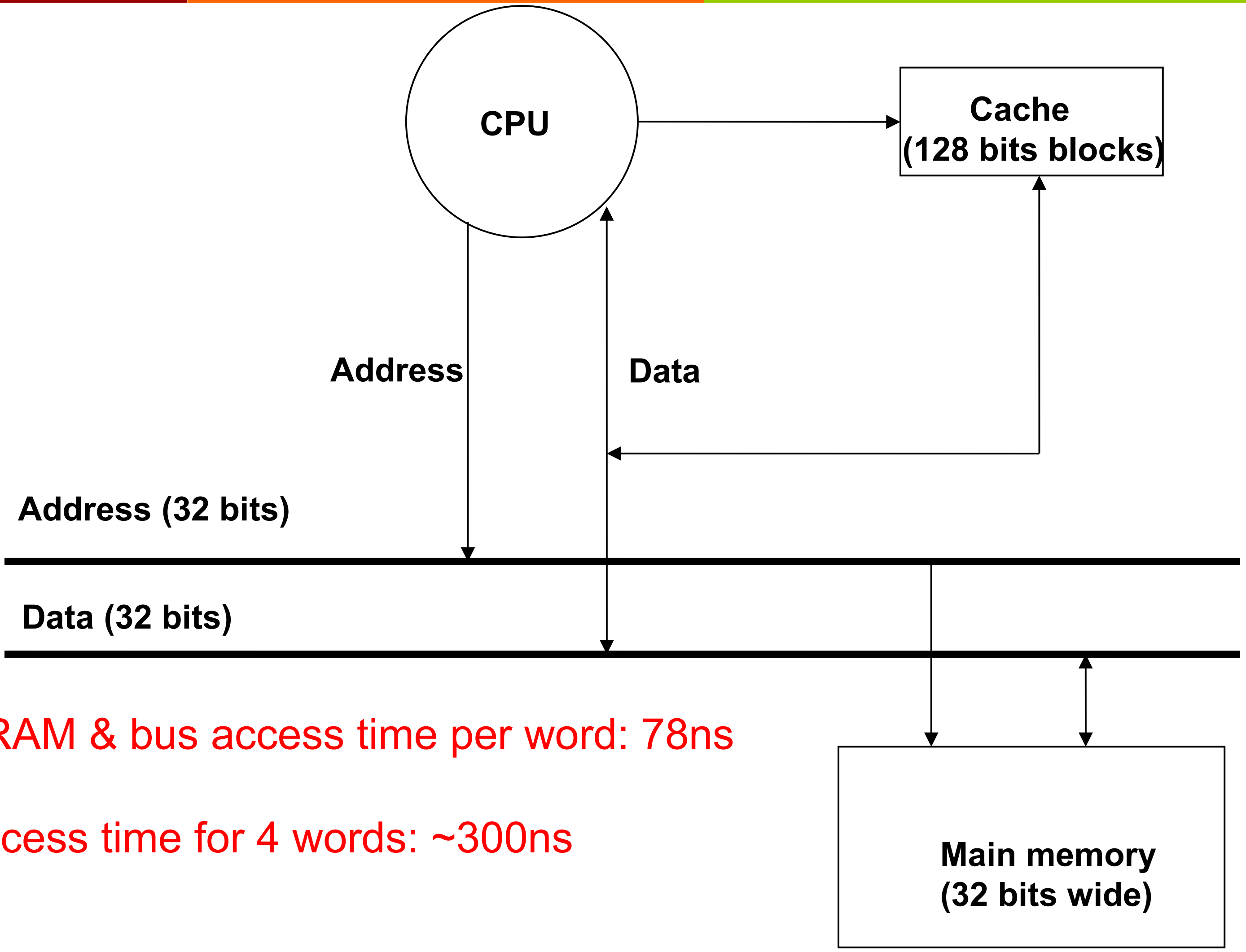

VPN/PFN 19 bits
Page Offset 13 bits
These 4 bits must remain unchanged through VPN to PFN translation
Tag 15 bits
Index 11 bits
Offset 6 bits
Bigger index ⓟ Larger cache

Work out Example 10 on your own and check the solution

Page coloring impacts

- Require the low bits of the VPN and PFN to be identical (4 low order bits in this example)
  - This allows the use of these bits as part of either the virtual or physical address, just like we use the offset
- What does that impact?
  - A virtual page can only occupy a subset of page frames in which the low bits of the VPN match the low bits of the PFN.
  - We refer to this as **page coloring** which means the page replacement algorithm must keep track of the “color” of the VPNs and PFNs (namely those low bits) to ensure virtual pages are only loaded into like-colored page frames.
  - It could make it harder to fit a working set into physical memory, but it’s often workable because processes tend to use contiguous pages so the VPNs of the pages are spread evenly among the colors
  - In this example using the low 4 bits of the VPN/PFN, we get 16 different “colors” of page/page frame. So a page with a VPN ending in 0010_2 can only be placed in a page frame with a number that ends in 0010_2

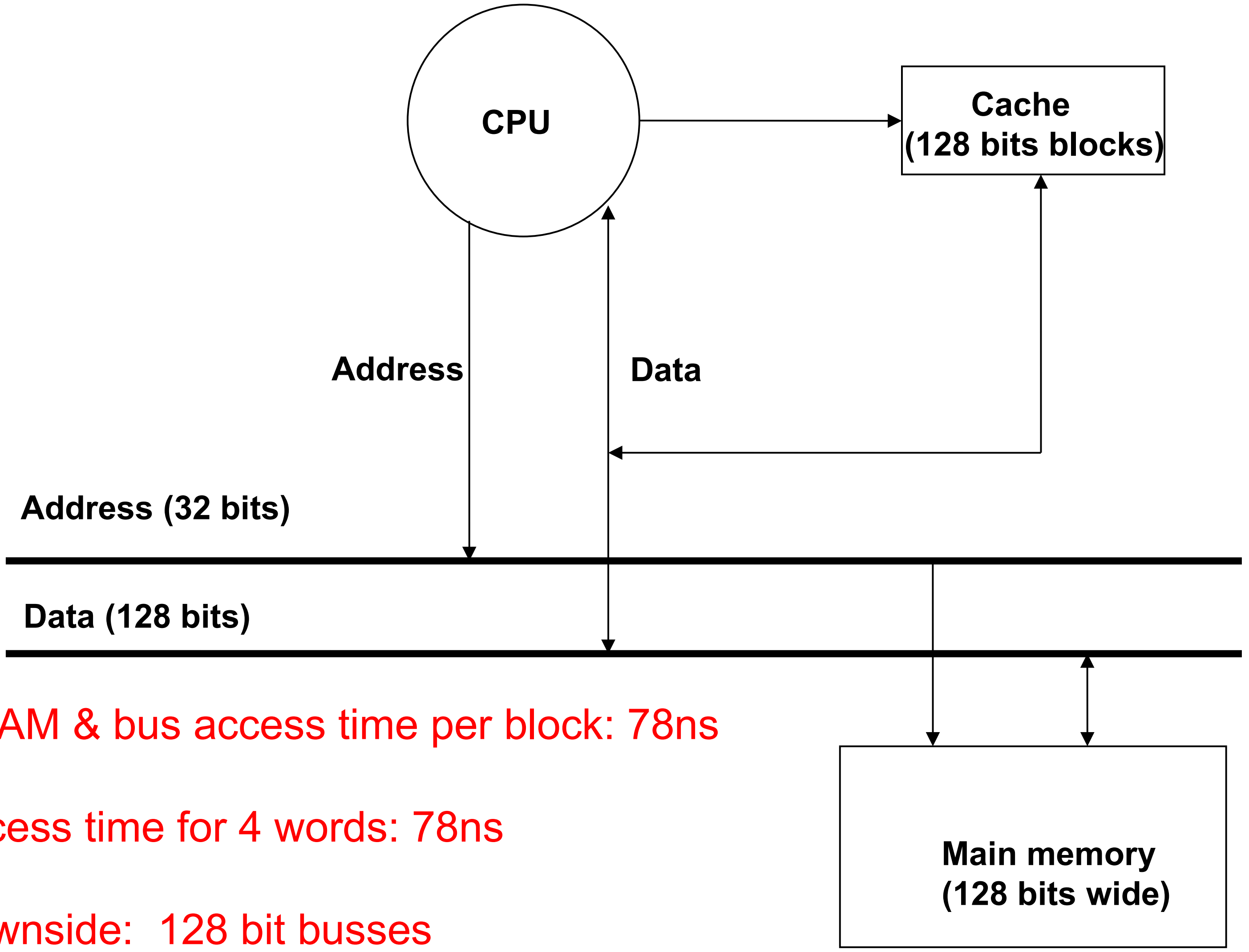
Simple memory system



DRAM & bus access time per word: 78ns

Access time for 4 words: ~300ns

Memory system matching cache blocksize

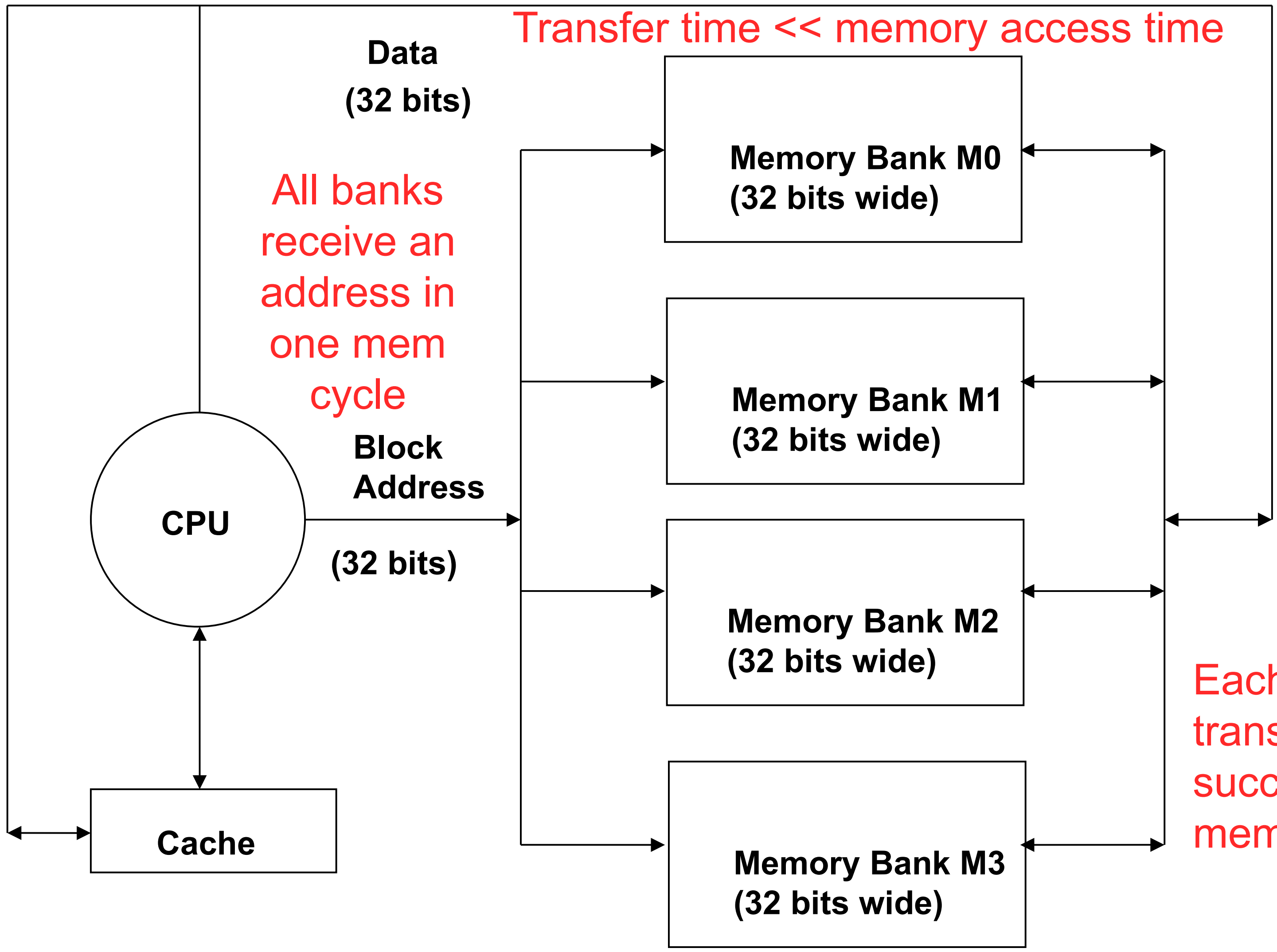


DRAM & bus access time per block: 78ns

Access time for 4 words: 78ns

Downside: 128 bit busses

Interleaved memory



Transfer time << memory access time

All banks receive an address in one mem cycle

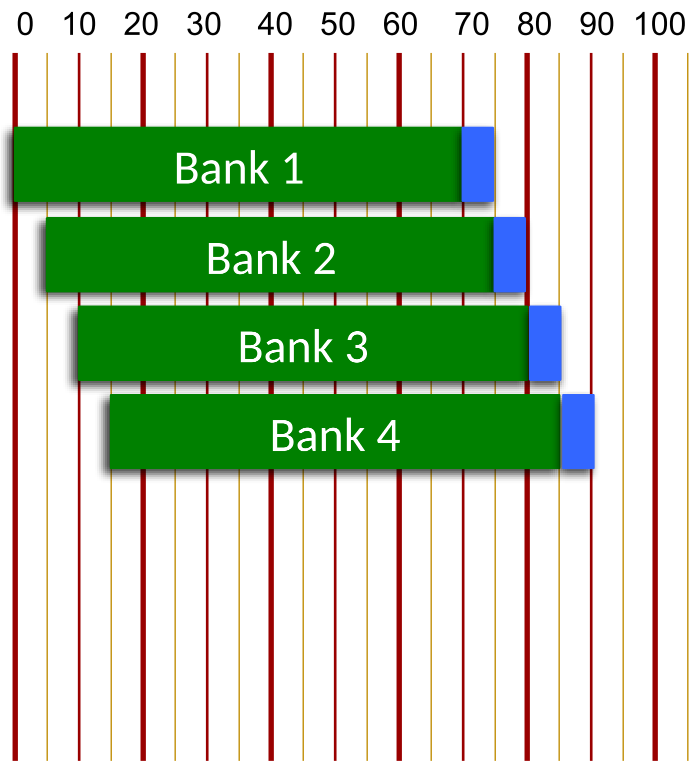
Each bank transfers in successive memory cycles

Example

- 4-way interleaved memory
- DRAM access time: 70 cycles.
- Memory bus cycle time: 5 cycles
- Compute the block transfer time for a block size of 4 words. Assume all 4 words are first retrieved from the DRAM before the data transfer to the CPU.

Example solution sketch

- Start memory fetches at 0, 5, 10, 15ns
- Bank 4 memory fetch finishes at 85ns
- Data transfers to CPU/cache occur at 70, 75, 80, 85ns



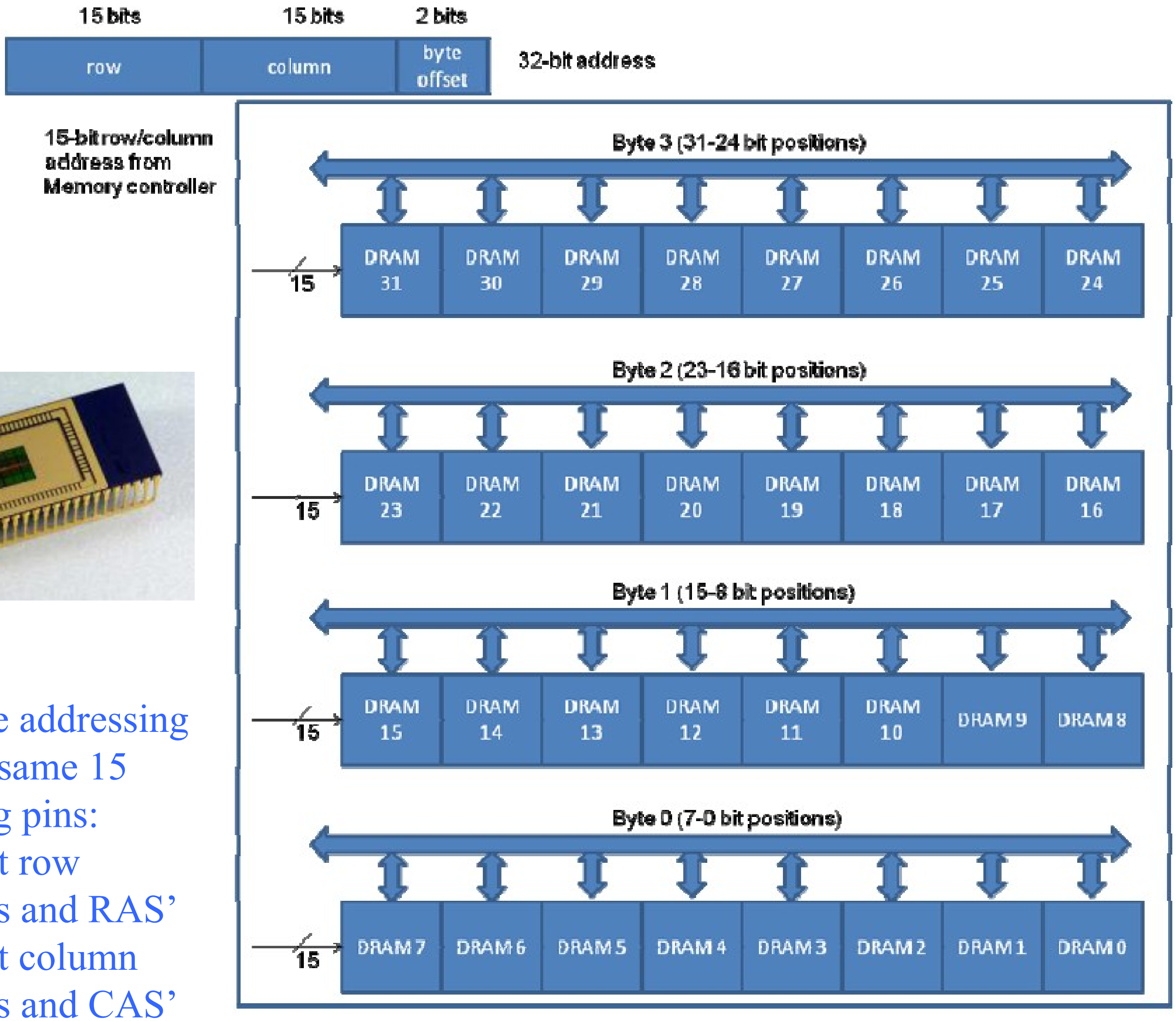
Interleaved memory is useful because

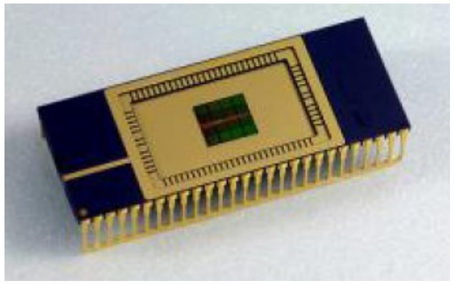
A. It increases the cache block size

B. It reduces the cache block size

[✓ C. It allows concurrent fetches of data blocks from memory without requiring wider busses]

D. It decreases the size of the cache index





Two-cycle addressing using the same 15 addressing pins:

- Present row address and RAS'
- Present column address and CAS'

Figure 9.42: A 4 GByte memory system using a 1 Gbit DRAM chip

Relative sizes & latencies, ca. 2009

Type of Memory	Typical Size	Approximate latency in CPU clock cycles to read one word of 4 bytes
CPU registers	8 to 32	Usually immediate access (0-1 clock cycles)
L1 Cache	32 (Kilobyte) KB to 128 KB	3 clock cycles
L2 Cache	128 KB to 4 Megabyte (MB)	10 clock cycles
Main (Physical) Memory	256 MB to 4 Gigabyte (GB)	100 clock cycles
Virtual Memory (on disk)	1 GB to 1 Terabyte (TB)	10^6 to 10^7 clock cycles (not accounting for the software overhead of handling page faults)

Relative sizes & latencies, ca. 2022

Type of Memory	Typical Size	Approximate latency in CPU clock cycles to read one word of 4 bytes
CPU registers	8 to 32	Usually immediate access (0-1 clock cycles)
L1 Cache	32 KB to 4MB	1-3 clock cycles
L2 Cache	32 KB to 32 MB	~10 clock cycles
L3 Cache	1 MB to 768 MB	~30 clock cycles
Main (Physical) Memory	256 MB to 8 TB	~100 clock cycles
Virtual Memory (on disk)	1 GB to 8 TB?	10^6 to 10^7 clock cycles (not including the software overhead of handling page faults)

Summary of Chapter 9 terminology

Category	Vocabulary	Details
Principle of locality (Section 9.2)	Spatial	Access to contiguous memory locations
	Temporal	Reuse of memory locations already accessed
Cache organization	Direct-mapped	One-to-one mapping (Section 9.6)
	Fully associative	One-to-any mapping (Section 9.12.1)
	Set associative	One-to-many mapping (Section 9.12.2)
Cache reading/writing (Section 9.8)	Read hit/Write hit	Memory location being accessed by the CPU is present in the cache
	Read miss/Write miss	Memory location being accessed by the CPU is not present in the cache
Cache write policy (Section 9.8)	Write through	CPU writes to cache and memory
	Write back	CPU only writes to cache; memory updated on replacement
Cache parameters	Total cache size (S)	Total data size of cache in bytes
	Block Size (B)	Size of contiguous data in one data block
	Degree of associativity (p)	Number of homes a given memory block can reside in a cache
	Number of cache lines (L)	S/pB
	Cache access time	Time in CPU clock cycles to check hit/miss in cache
	Unit of CPU access	Size of data exchange between CPU and cache
	Unit of memory transfer	Size of data exchange between cache and memory
	Miss penalty	Time in CPU clock cycles to handle a cache miss
Memory address interpretation	Index (n)	$\log_2 L$ bits, used to look up a particular cache line
	Block offset (b)	$\log_2 B$ bits, used to select a specific byte within a block
	Tag (t)	a – (n+b) bits, where a is number of bits in memory address; used for matching with tag stored in the cache

Summary of Chapter 9 terminology

Category	Vocabulary	Details
Cache entry/cache block/cache line	Valid bit	Signifies data block is valid
	Dirty bits	For write-back, signifies if the data block is more up to date than memory
	Tag	Used for tag matching with memory address for hit/miss
	Data	Actual data block
Performance metrics	Hit rate (h)	Percentage of CPU accesses served from the cache
	Miss rate (m)	1 – h
	Avg. Memory stall	Misses-per-instruction$_{Avg}$ * miss-penalty$_{Avg}$
	Effective memory access time (EMAT$_i$) at level i	EMAT$_i$ = T$_i$ + m$_i$ * EMAT$_{i+1}$
	Effective CPI	CPI$_{Avg}$ + Memory-stalls$_{Avg}$
Types of misses	Compulsory miss	Memory location accessed for the first time by CPU
	Conflict miss	Miss incurred due to limited associativity even though the cache is not full
	Capacity miss	Miss incurred when the cache is full
Replacement policy	FIFO	First in first out
	LRU	Least recently used
Memory technologies	SRAM	Static RAM with each bit realized using a flip flop
	DRAM	Dynamic RAM with each bit realized using a capacitive charge
Main memory	DRAM access time	DRAM read access time
	DRAM cycle time	DRAM read and refresh time
	Bus cycle time	Data transfer time between CPU and memory
	Simulated interleaving using DRAM	Using page mode bits of DRAM