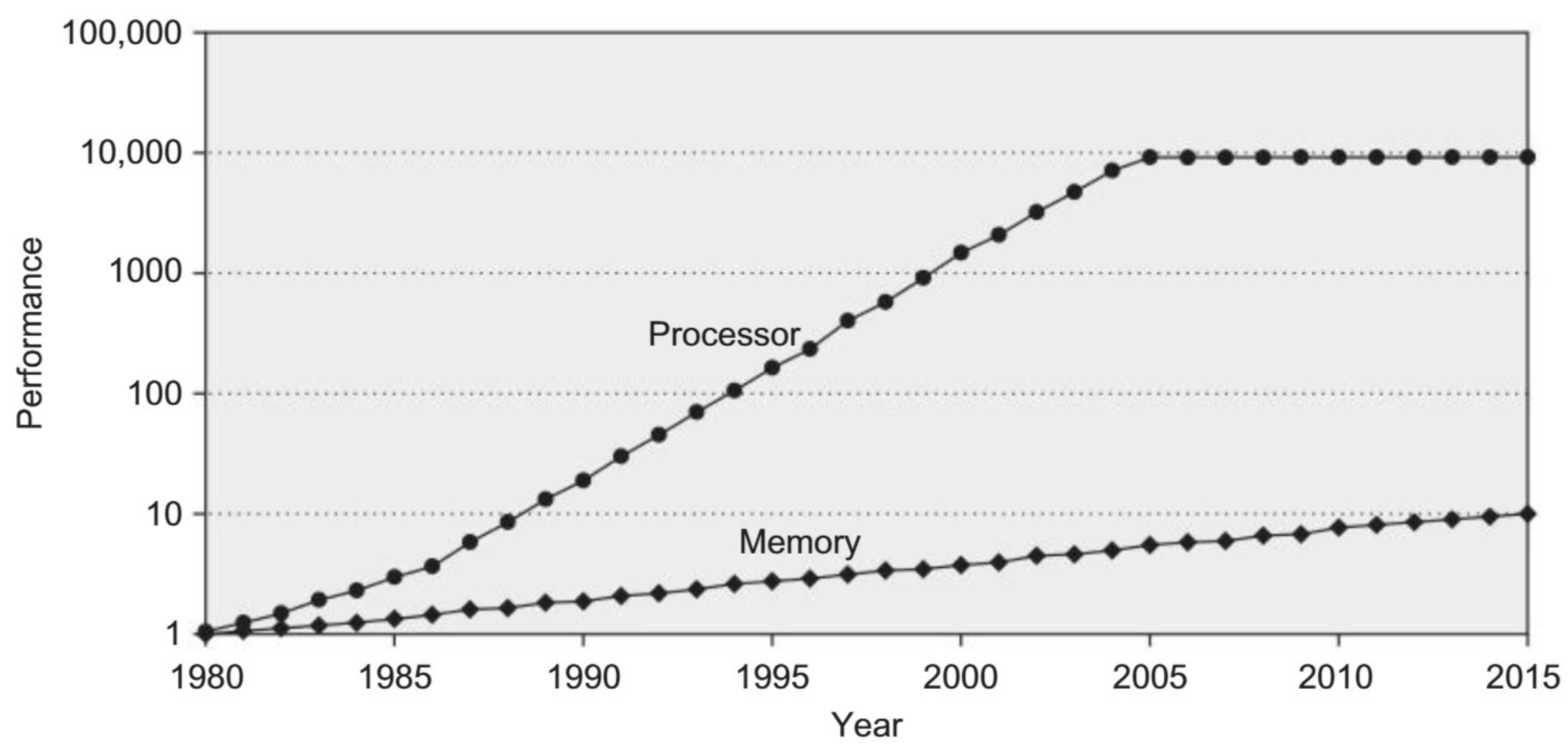
The cache hierarchy

For single-core processors

(working our way up to ~2005)

The memory wall



John L. Hennessy and David A. Patterson. "Computer Architecture: A Quantitative Approach, 6th ed." Morgan Kaufmann, 2017

- DRAM performance improvement
  - 7% per year
- Processor performance improvement
  - 30 to 50% per year until 2005

Comparing caches

Hardware implementations

Examples of locality

Trade-offs

Cache organization

Reinforce and build on concepts from the prep

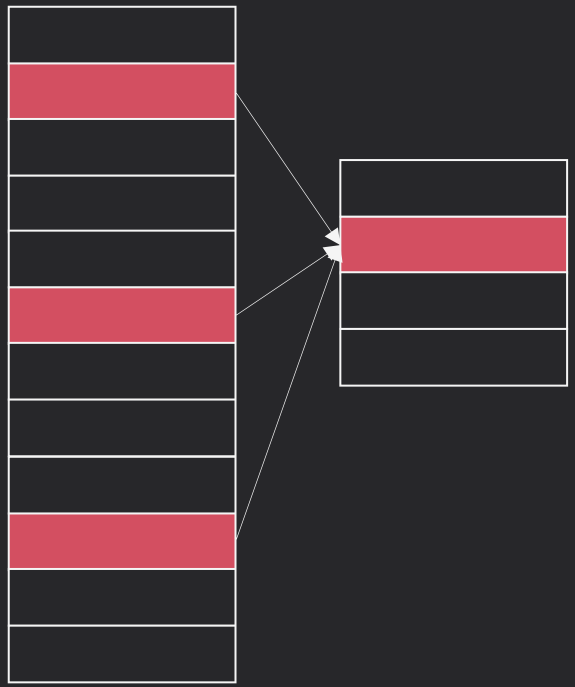
Recapping cache terminology

Symbol	Brief description	Units	Relationship
C	Cache capacity	e.g., number of words, bytes	$C = B \times b$
b	Block size	e.g., number of words, bytes	$b = \frac{C}{B}$
S	Number of sets	Count	$S = \frac{B}{N}$
B	Number of blocks (number of “lines”)	Count	$B = \frac{C}{b}$
N	Degree of associativity (number of “ways”)	Count	$N = \frac{B}{S}$

Cache organization

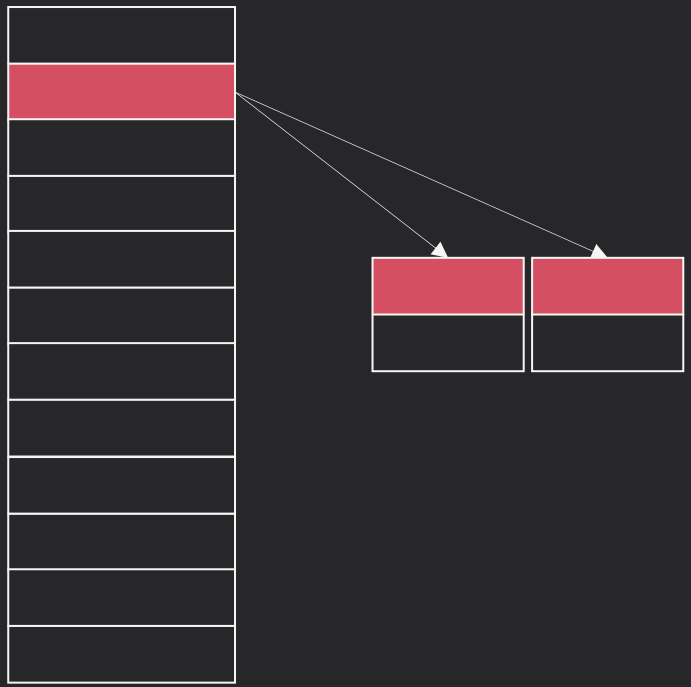
- Caches are organized into (S) *sets*. A set consists of one or more blocks
- Each memory address maps to one set in the cache

direct mapped cache



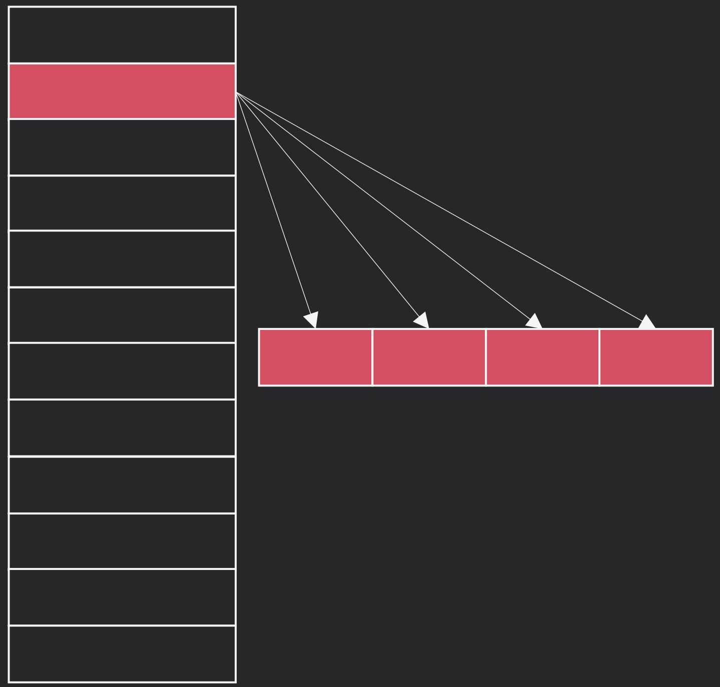
Each set contains one block ($S = B$).

N-way set-associative cache



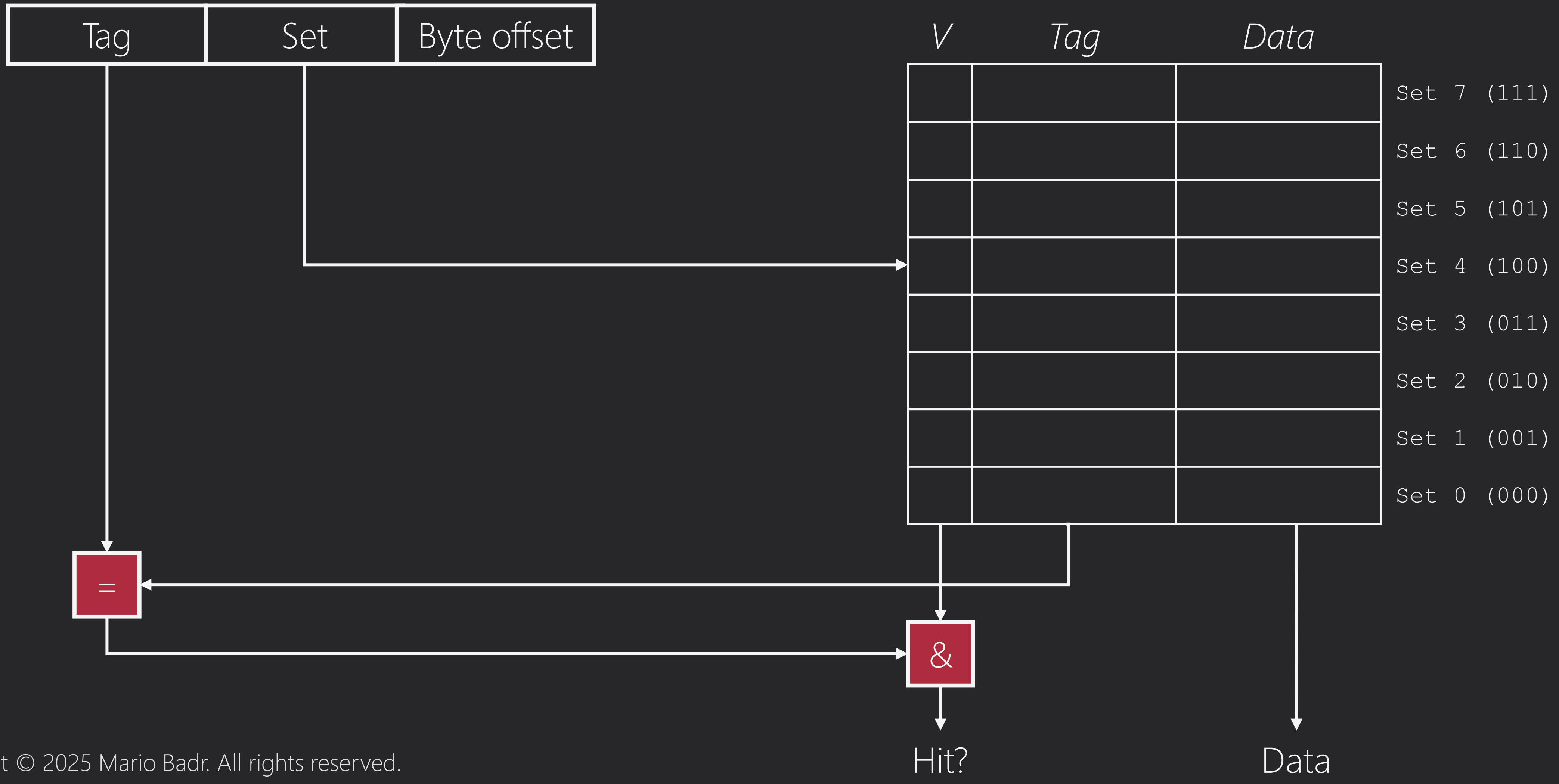
Each set contains N blocks ($S = {}^{B}/_{N}$).
Data from memory address goes into *any* block in its corresponding set.

Fully associative cache



There is only 1 set.
Data can go in any of the B blocks in the set.

8-word direct mapped cache (b=1)



Example: 8-word direct mapped cache (b=1)

```
addi $t0, $zero, 5
loop:
    lw $t1, 0x4($zero)
    lw $t2, 0x8($zero)
    add $s0, $t1, $t2
    lw $t3, 0xC($zero)
    add $s0, $s0, $t3
    addi $t0, $t0, -1
    bne $t0, $zero, loop
```

V	Tag	Data	
			Set 7 (111)
			Set 6 (110)
			Set 5 (101)
			Set 4 (100)
			Set 3 (011)
			Set 2 (010)
			Set 1 (001)
			Set 0 (000)

Hits	
Misses	
References	

Example: 8-word direct mapped cache (b=1)

```
addi $t0, $zero, 5
loop:
    lw $t1, 0x4($zero)
    lw $t2, 0x8($zero)
    add $s0, $t1, $t2
    lw $t3, 0xC($zero)
    add $s0, $s0, $t3
    addi $t0, $t0, -1
    bne $t0, $zero, loop
```

V	*Tag*	*Data*	
0			Set 7 (111)
0			Set 6 (110)
0			Set 5 (101)
0			Set 4 (100)
1	0 ... 0	Mem[0xC]	Set 3 (011)
1	0 ... 0	Mem[0x8]	Set 2 (010)
1	0 ... 0	Mem[0x4]	Set 1 (001)
0			Set 0 (000)

Hits	12
Misses	3
References	15

Causing a cache block conflict (conflict miss)

```
addi $t0, $zero, 5
loop:
    lw $t1, 0x4($zero)
    lw $t2, ____($zero)
    add $s0, $t1, $t2
    addi $t0, $t0, -1
    bne $t0, $zero, loop
```

- Assume our direct mapped cache
- What should the offset in the ___ be to cause a conflict?

- Assuming the second lw causes a conflict, what is the hit and miss rate?

Hits	
Misses	
References	

Causing a cache block conflict (conflict miss)

```
addi $t0, $zero, 5
loop:
    lw $t1, 0x4($zero)
    lw $t2, 0x24($zero)
    add $s0, $t1, $t2
    addi $t0, $t0, -1
    bne $t0, $zero, loop
```

- Assume our direct mapped cache
- What should the offset in the ___ be to cause a conflict?
  - 0x24
- Assuming the second lw causes a conflict, what is the hit and miss rate?

Hits	0
Misses	10
References	10

Checkpoint 1

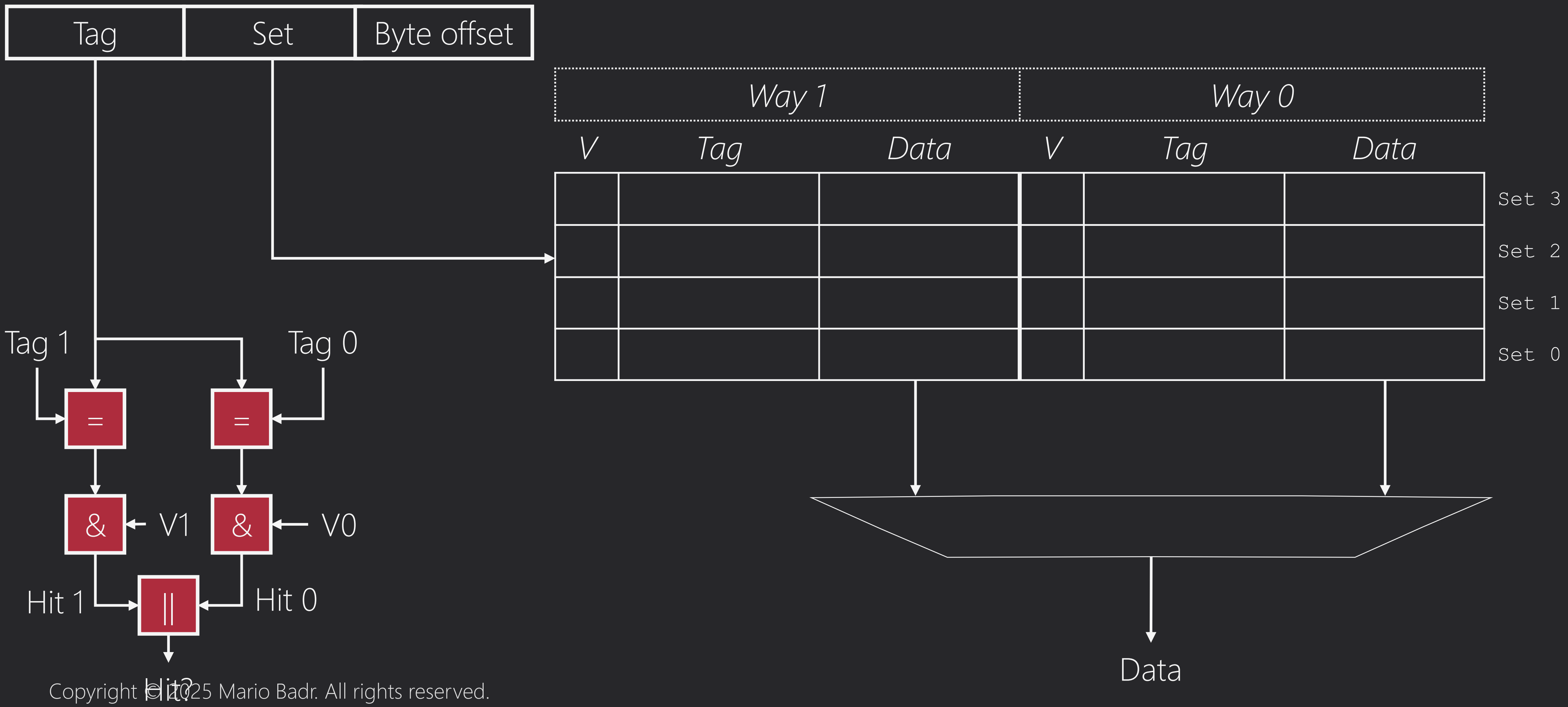
The impact of conflicts

- Direct mapped caches will have larger miss ratios than caches with some degree of associativity
- Direct mapped caches have terrible worst-case behaviour
  - All accesses map to the same set

Classifying cache misses (the 3 C's)

- Compulsory (or cold) miss
  - The first request to a cache block
- Conflict miss
  - An address maps to the same set, evicting a cache block that is still needed
- Capacity miss
  - The cache is too small to hold all a workload's *working set*

8-word 2-way set-associative cache



Avoiding a cache block conflict (conflict miss)

```
addi $t0, $zero, 5
loop:
    lw $t1, 0xC($zero)
    lw $t2, 0x2C($zero)
    add $s0, $t1, $t2
    addi $t0, $t0, -1
    bne $t0, $zero, loop
```

Way 1			Way 0			
V	*Tag*	*Data*	*V*	*Tag*	*Data*	
						Set 3
						Set 2
						Set 1
						Set 0

Hits	
Misses	
References	

Avoiding a cache block conflict (conflict miss)

```
addi $t0, $zero, 5
loop:
    lw $t1, 0xC($zero)
    lw $t2, 0x2C($zero)
    add $s0, $t1, $t2
    addi $t0, $t0, -1
    bne $t0, $zero, loop
```

Way 1			*Way 0*			
V	*Tag*	*Data*	*V*	*Tag*	*Data*	
1	0 ... 10	Mem[0x2C]	1	0 ... 00	Mem[0xC]	Set 3
						Set 2
						Set 1
						Set 0

Hits	8
Misses	2
References	10

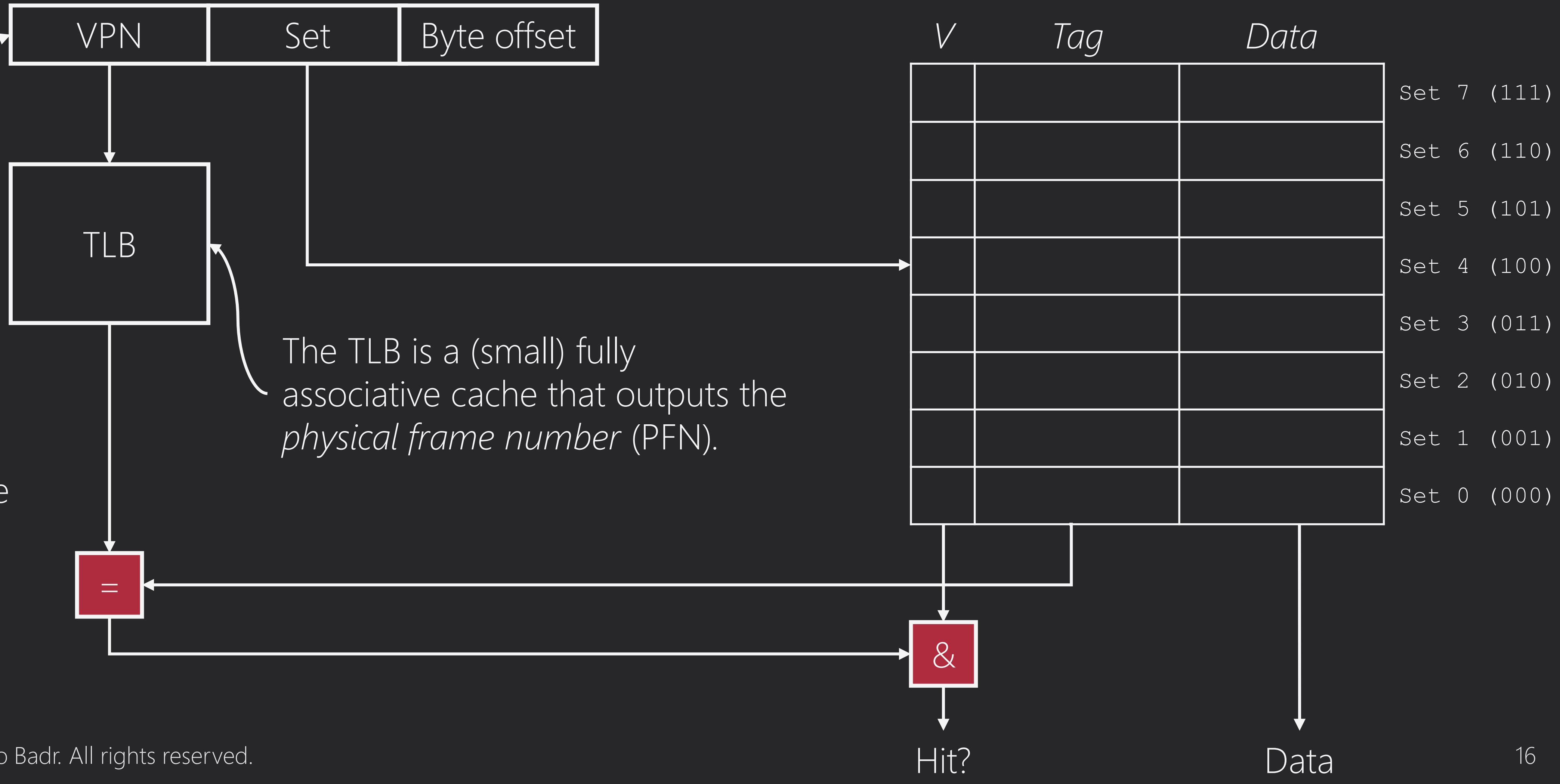
8-word fully associative cache

Way 7			*Way ...*			*Way 0*		
V	*Tag*	*Data*	*V*	*Tag*	*Data*	*V*	*Tag*	*Data*

- How many comparators (for tag comparisons) are needed?
- What kind of multiplexer is needed?
- Are fully associative caches ever used?

The translation lookaside buffer (TLB)

The operating system uses this part of the address to contain the *virtual page number* (VPN).



The TLB is a (small) fully associative cache that outputs the *physical frame number* (PFN).

The *tag* of our cache contains the PFN.

A replacement policy

- Direct mapped caches don't need a replacement policy
  - Why?

- Caches with associativity need a replacement policy
  - When a cache set is full, choose a block (the *victim*) to *evict*

- Based on temporal locality, one replacement policy is *least recently used*
  - Two-way set associative caches can add a *use bit* for this policy.
    - U = 0 means Way 0 is least recently used.
    - U = 1 means Way 1 is least recently used.

Least recently used (LRU) replacement

Consider this sequence:

```
1. lw $s0, 0x04($zero)
2. lw $s1, 0x14($zero)
3. lw $s2, 0x34($zero)
```

Way 1				Way 0			
V	U	Tag	Data	V	Tag	Data	
							Set 3
							Set 2
							Set 1
							Set 0

These all map to Set 1. How to update the cache?

1.

2.

3.

Least recently used (LRU) replacement

Consider this sequence:

```
1. lw $s0, 0x04($zero)
2. lw $s1, 0x14($zero)
3. lw $s2, 0x34($zero)
```

Way 1				Way 0			
V	U	Tag	Data	V	Tag	Data	
							Set 3
							Set 2
							Set 1
1	1	0 ... 01	Mem[0x14]	1	0 ... 11	Mem[0x34]	Set 0

These all map to Set 1. How to update the cache?

1. Put data in Way 0, set U to 1
2. Put data in Way 1, set U to 0. (Set 1 is now full)
3. Put data in Way 0, set U to 1. (evicted 0x04)

Other replacement policies

- First-in, first-out (FIFO)

- Least frequently used (LFU)
  - The block referenced the fewest number of times

- Pseudo-LRU (or “not most recently used”)
  - Problem: One use bit (U) is not enough if there are more than 2 ways
  - Solution: Split ways into two groups. U now indicates which group was least recently used. Replace a block at random within the group.

Checkpoint 2

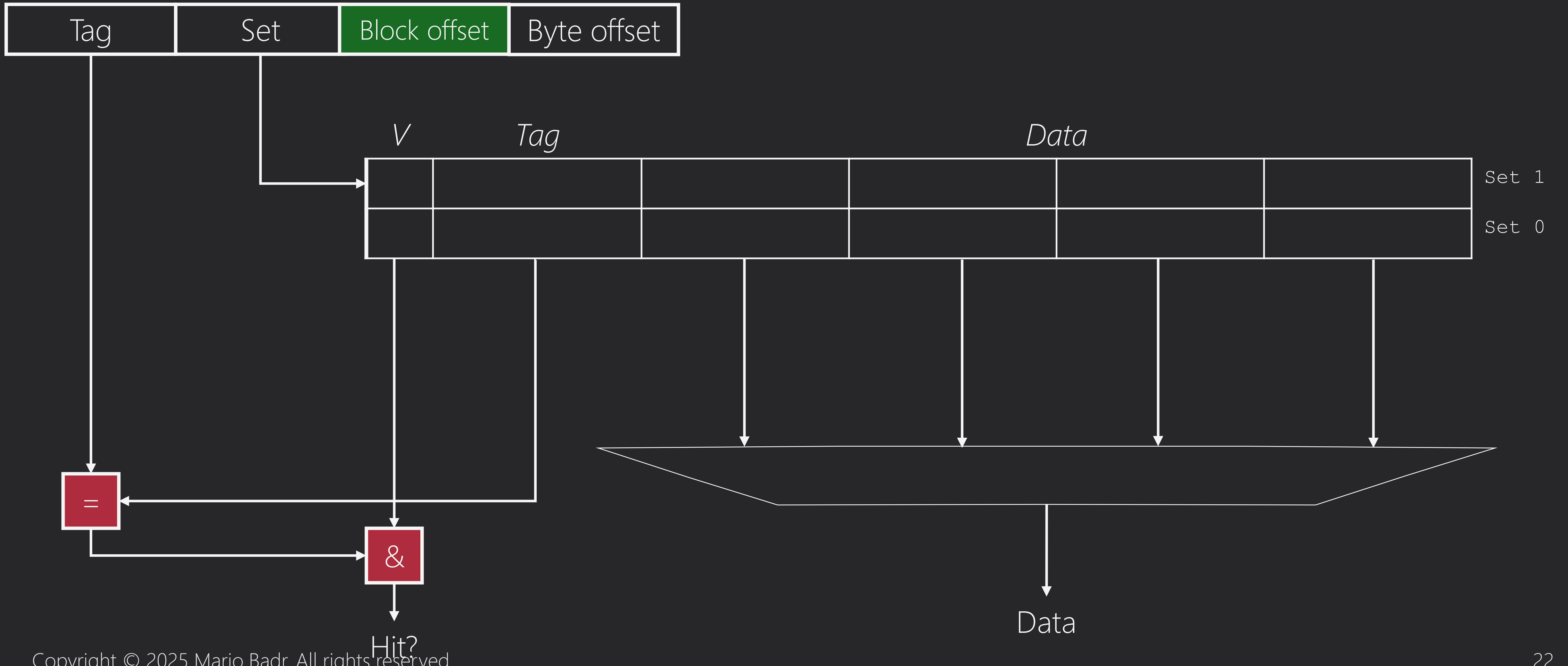
Advantages of N > 1

- Lower miss ratio
  - e.g., due to less conflict misses
- Less likely worst-case behaviour
  - e.g., due to repeated conflicts

Disadvantages of N > 1

- More hardware
  - e.g., comparators, gates, larger multiplexers
- Slower hit time
  - Longer critical path to determine a hit
- More metadata
  - e.g., The use bit (U)
- More logic
  - e.g., to select blocks according to replacement policy

8-word direct mapped cache (b=4)



Example: 8-word direct mapped cache (b=4)

```
addi $t0, $zero, 5
loop:
    lw $t1, 0x4($zero)
    lw $t2, 0x8($zero)
    add $s0, $t1, $t2
    lw $t3, 0xC($zero)
    add $s0, $s0, $t3
    addi $t0, $t0, -1
    bne $t0, $zero, loop
```

V	Tag	Data				
						Set 1
						Set 0

Hits	
Misses	
References	

Example: 8-word direct mapped cache (b=4)

```
addi $t0, $zero, 5
loop:
    lw $t1, 0x4($zero)
    lw $t2, 0x8($zero)
    add $s0, $t1, $t2
    lw $t3, 0xC($zero)
    add $s0, $s0, $t3
    addi $t0, $t0, -1
    bne $t0, $zero, loop
```

V	*Tag*	*Data*				
0						Set 1
1	0 ... 0	Mem[0xC]	Mem[0x8]	Mem[0x4]	Mem[0x0]	Set 0

Hits	14
Misses	1
References	15

A recap of cache organization concepts

- Caches exploit *temporal locality*
- Increasing block size (b) improves *spatial locality*
  - What are the trade-offs?
- Increasing associativity (N) reduces *conflict misses*
  - What are the trade-offs?
- Increasing capacity (C) improves the *hit rate*
  - What are the trade-offs?

Separating instructions and data

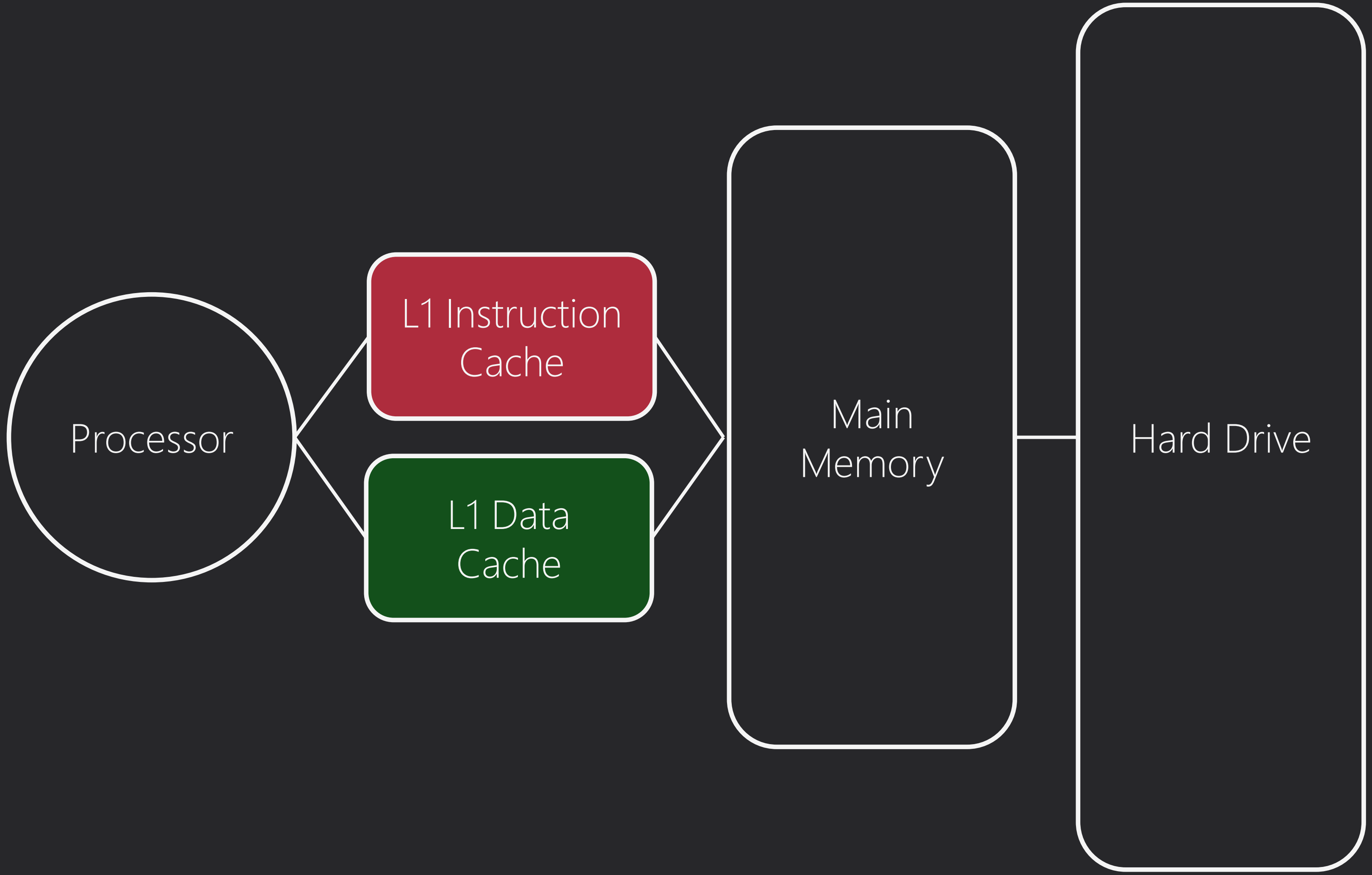
Multiple levels

Dealing with store instructions

Other cache concepts

Working toward a modern cache hierarchy

A Harvard architecture



- Harvard Mark 1
  - Instructions: tape
  - Data: electro-mechanical counters
- Instruction fetch has different access patterns than data
  - Addresses localized to the "code section" of a program
- When instruction and data are mixed, the cache is called *unified*

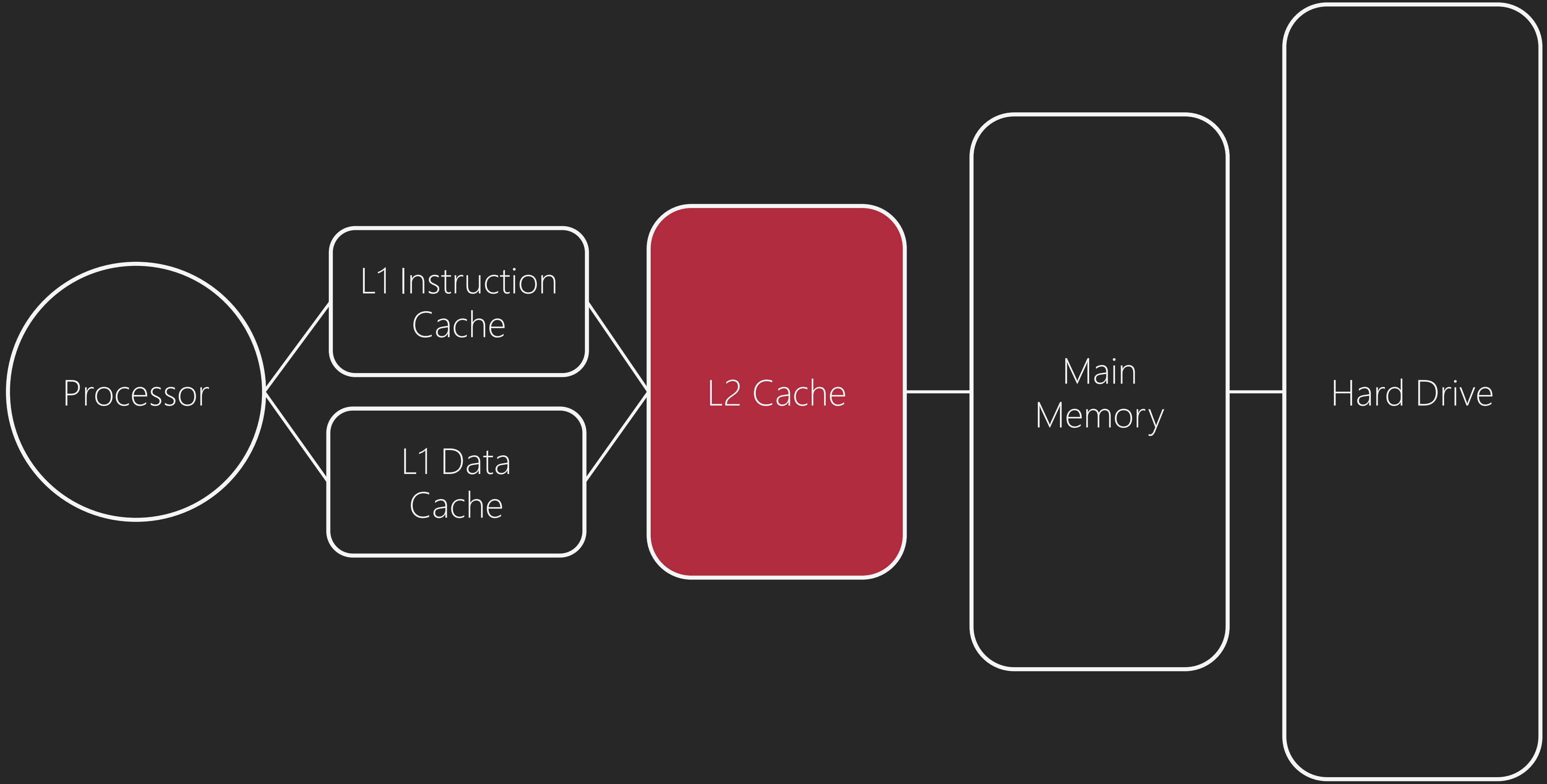
Example: Impact of “real” memory on CPI

- A pipelined processor has a CPI of 1.3 when using “magical memory” (everything takes 1 cycle)
- It is now connected to a real memory:
  - L1 instruction cache (access = 1 cycle, miss rate = 5%)
  - L1 data cache (access = 1 cycle, miss rate = 15%)
  - Main memory (access = 100 cycles)
- What is the CPI after adding a “real” instruction cache if 30% of all instructions access the data cache?

Example: Impact of “real” memory on CPI

- A pipelined processor has a CPI of 1.3 when using “magical memory” (everything takes 1 cycle)
- It is now connected to a real memory:
  - L1 instruction cache (access = 1 cycle, miss rate = 5%)
  - L1 data cache (access = 1 cycle, miss rate = 15%)
  - Main memory (access = 100 cycles)
- What is the CPI after adding a “real” instruction cache if 30% of all instructions access the data cache?
  - $CPI = CPI_{base} + CPI_{penalty} = 1.3 + (0.05)(100) + (0.3)(0.15)(100) = 10.8$

Multiple levels of cache



The cache size trade-off

- As cache size increases
  - Miss rates decrease (more data can be held in the cache)
  - Access latency increases

- Distribute data across multiple levels of cache
  - L1: Smallest and fastest (1- to 2- cycles on a hit)
  - L2: Larger and slower than L1
  - L3: Larger and slower than L2
  - L4: ...

Example: AMAT with L1 and L2 caches

- Memory system includes an L1, L2, and main memory with access latencies of 1, 10, and 100 cycles.
- A workload is run and hits in the L1 95% of the time, but misses in the L2 20% of the time.

- What is the AMAT?

- If there were 1,000 accesses made, how many missed in the L2?

Example: AMAT with L1 and L2 caches

- Memory system includes an L1, L2, and main memory with access latencies of 1, 10, and 100 cycles.
- A workload is run and hits in the L1 95% of the time, but misses in the L2 20% of the time.

- What is the AMAT?
  - $AMAT = 1 + (1 - 0.95)\big(10 + 0.2(100)\big) = 2.5$ cycles
- If there were 1,000 accesses made, how many missed in the L2?
  - $1000 \times (1 - 0.95) \times 0.2 = 10$

Memory coherence (for single-core)

A load instruction returns the value of the previous store (program order) to the same address

Memory inclusion

- Consider two caches: L2 and L1

- L1 and L2 are *inclusive* if every cached item in L1 is also in L2
  - Helps maintain coherence
  - Also helps in multi-core cache hierarchies (covered in later weeks)

- Disadvantages
  - New lines in L1 must also be allocated in L2
  - Evictions in L2 must also evict lines in L1
  - Duplicated data across the cache hierarchy

Memory exclusion

- Consider two caches: L2 and L1
- L1 and L2 are *exclusive* if none of the cached items in L1 are in L2 (and vice versa)
  - Now, the total cache capacity is the sum of L1 and L2's size
- Disadvantages
  - New L1 cache blocks (from a miss) must invalidate L2's
  - Replacement in L1 may possibly allocate a new block at L2

Store instructions

- Hit
  - Instruction updates word in cache block
- Miss
  - Cache block is fetched from main memory, then the word is updated

- But when do stores get written to the next level of the hierarchy?
  - Option 1: “immediately” (write-through)
  - Option 2: “later” (write-back)

Example: memory traffic from stores

```
sw $t0, 0x0
sw $t0, 0x8
sw $t0, 0xC
sw $t0, 0x4
```

- Our hierarchy is:
  - L1 cache with b = 4
  - Main memory

- How many access to main memory are needed when using:
  a) A write-through policy
  b) A write-back policy

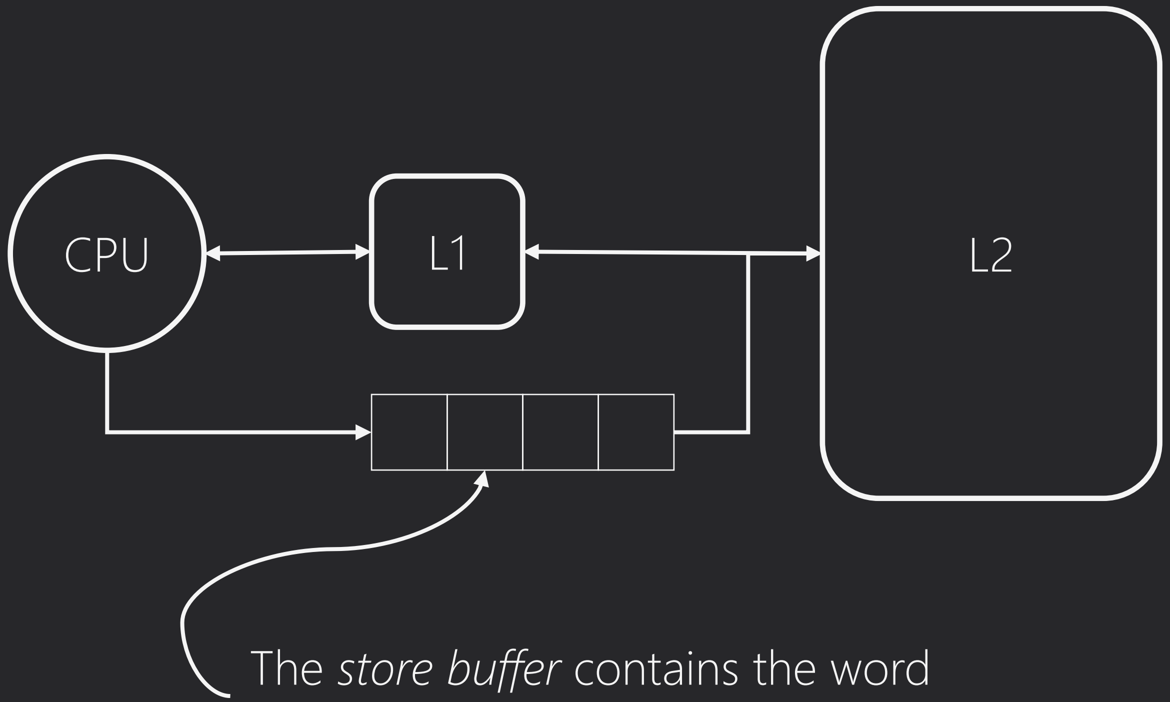
Example: memory traffic from stores

```
sw $t0, 0x0
sw $t0, 0x8
sw $t0, 0xC
sw $t0, 0x4
```

- Our hierarchy is:
  - L1 cache with b = 4
  - Main memory

- How many access to main memory are needed when using:
  a) A write-through policy: 4
  b) A write-back policy: 1 (on eviction)

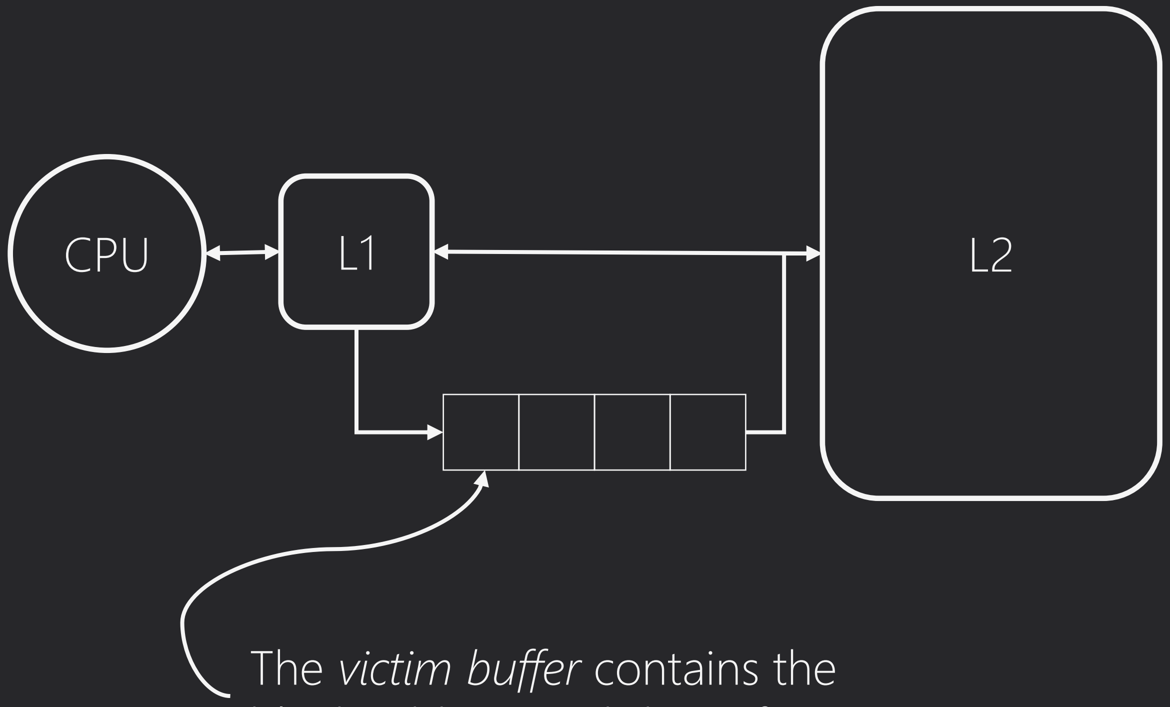
Write-through and the store buffer



The *store buffer* contains the word address and data of stores.

- Writing immediately to L2 would increase the latency of stores
  - A store buffer removes the processor from the critical path
  - But stalls if store buffer is full
- Typically, a store miss does not update the L1 cache
  - Only inserted into store buffer
  - Load misses need to check both L1 and store buffer now

Write-back and the victim buffer



The *victim buffer* contains the block address and data of stores.

- Each cache block now needs a *dirty bit (D)*
  - D = 1: cache block has been written
  - D = 0: otherwise (“clean”)
- Written to main memory only on eviction if the block is dirty
- Load misses to a dirty cache block?
  - Move the *victim* to a buffer while handling the load (on critical path)
  - Write back the victim later to the next level (off critical path)

Which write policy to use?

Write-through

- Store traffic does not improve with cache size
  - Limits improvements to the cache
- Preferred for small L1 caches

Write-back

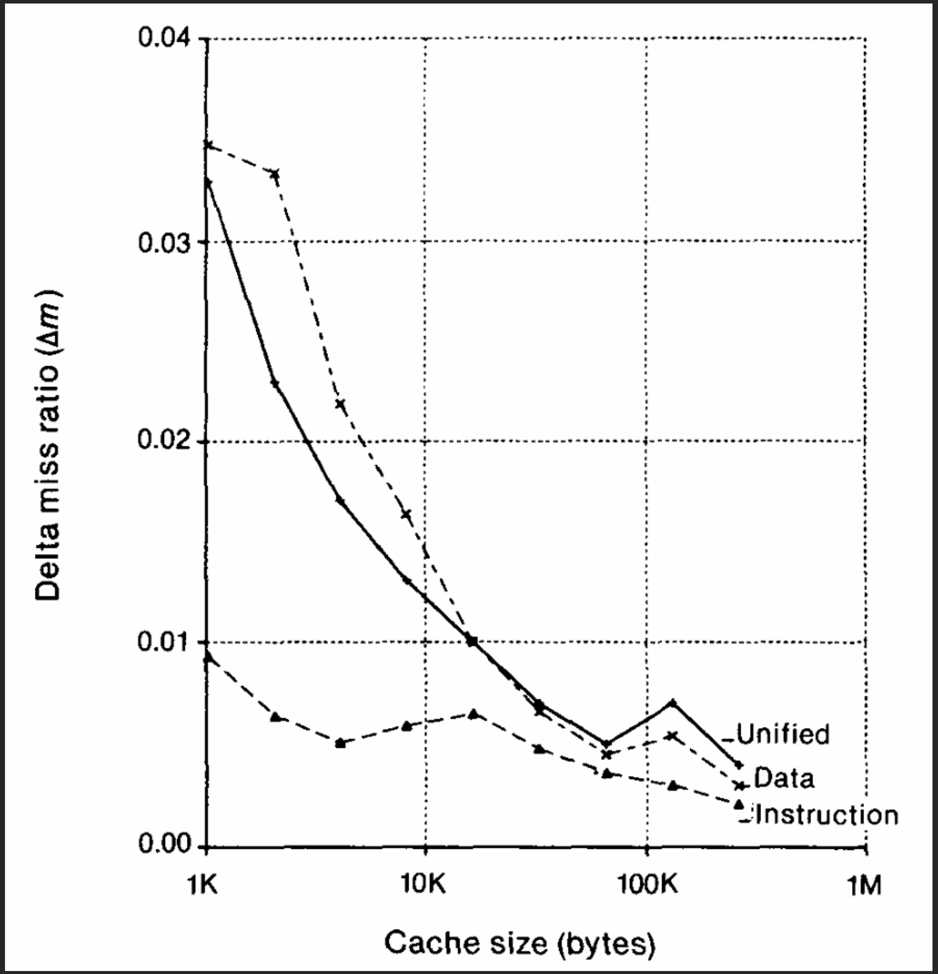
- Easier to maintain coherence between two levels
  - Don’t need to check the buffer
- Less traffic back to higher levels
  - Access latency to main memory is very high
- Preferred for larger caches

Conclusion

Connecting back to industry and academia

Miss ratio differences

Hill, Mark D. "A case for direct-mapped caches." Computer 21.12 (1988): 25-40.



- Let C1 and C2 be two caches with the same capacity and b = 32 bytes
  - C1: direct mapped
  - C2: 2-way set-associative
- Let m(C) be the miss ratio, then
  - $\Delta m = m(C_2) - m(C_1)$
- What does the graph show?
  - As cache size increases, the difference in miss ratio decreases

MIPS caches over time

Year	L1 Cache	L2 Cache
1985	None	None
1990	32 KB direct mapped	None
1991	8 KB direct mapped	1 MB direct mapped
1995	32 KB two-way	4 MB two-way
2001	32 KB two-way	16 MB two-way
2004	64 KB two-way	16 MB two-way

From Sweetman, Dominic. *See MIPS run*. Elsevier, 2010.

The IBM Power 4 and 5 caches

The IBM Power 4 (2002)					
Cache	Capacity	Associativity	Block size	Write policy	Replacement
L1 I-cache	64 KB	1	128 bytes	N/A	N/A
L1 D-cache	32 KB	2	128 bytes	Write-through	LRU
L2	1.5 MB	8	128 bytes	Write-back	Pseudo-LRU
L3	32 MB	8-way	512 bytes	Write-back	?
The IBM Power 5 (2005)					
Cache	Capacity	Associativity	Block size	Write policy	Replacement
L1 I-cache	64 KB	2	128 bytes	N/A	LRU
L1 D-cache	32 KB	4	128 bytes	Write-through	LRU
L2	2 MB	10	128 bytes	Write-back	Pseudo-LRU
L3	36 MB	12-way	512 bytes	Write-back	?