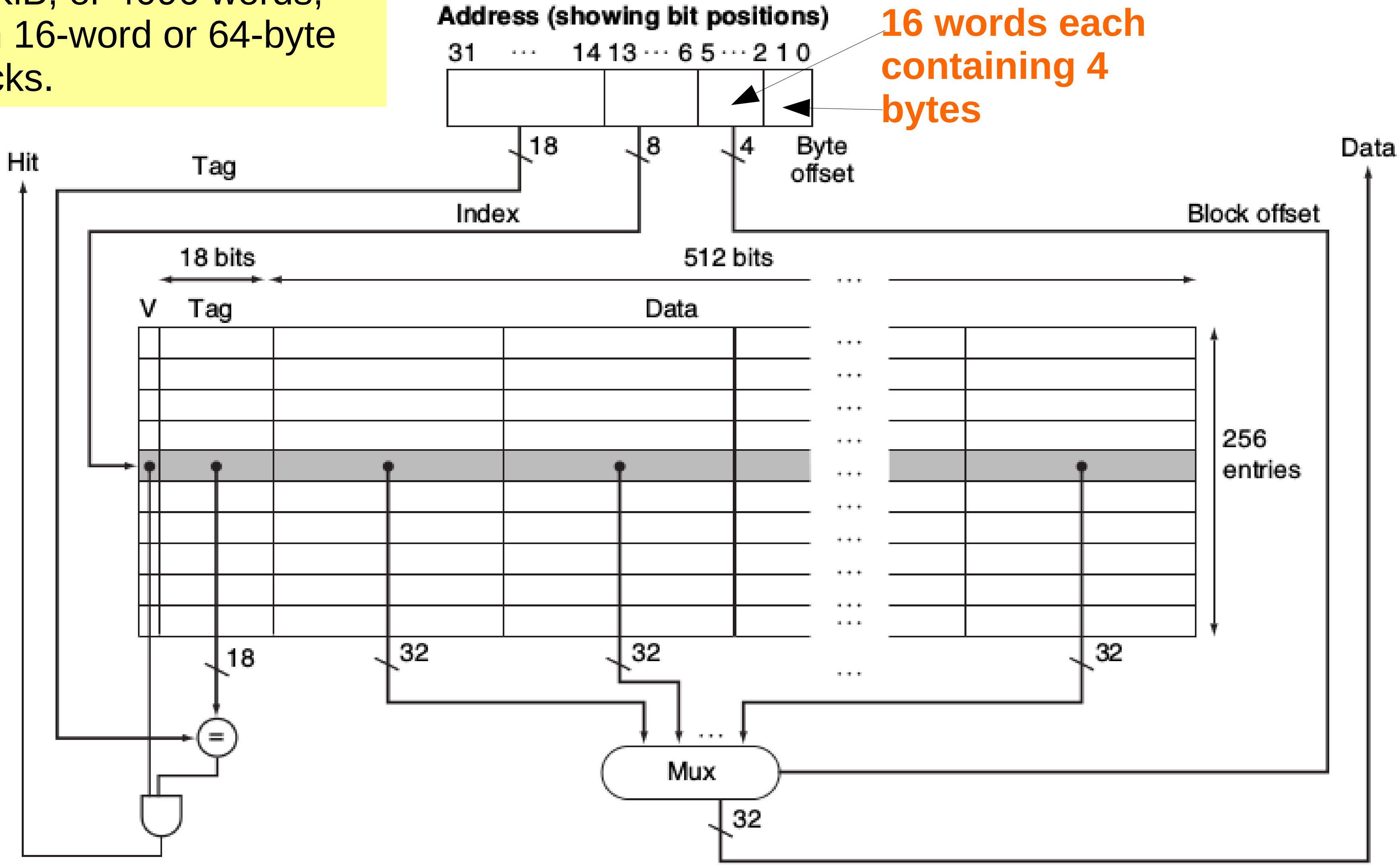
Caches - 3

Multi-level caches

- L1 – D cache for Data
  - Loads/Stores- Rd/Wr
- L1- I cache for Instructions
  - Only Loads/Reads
- L2, L3- onwards are Unified (D+I)
- In general, loads are more critical than stores

FastMATH processor

16 KiB, or 4096 words, with 16-word or 64-byte blocks.



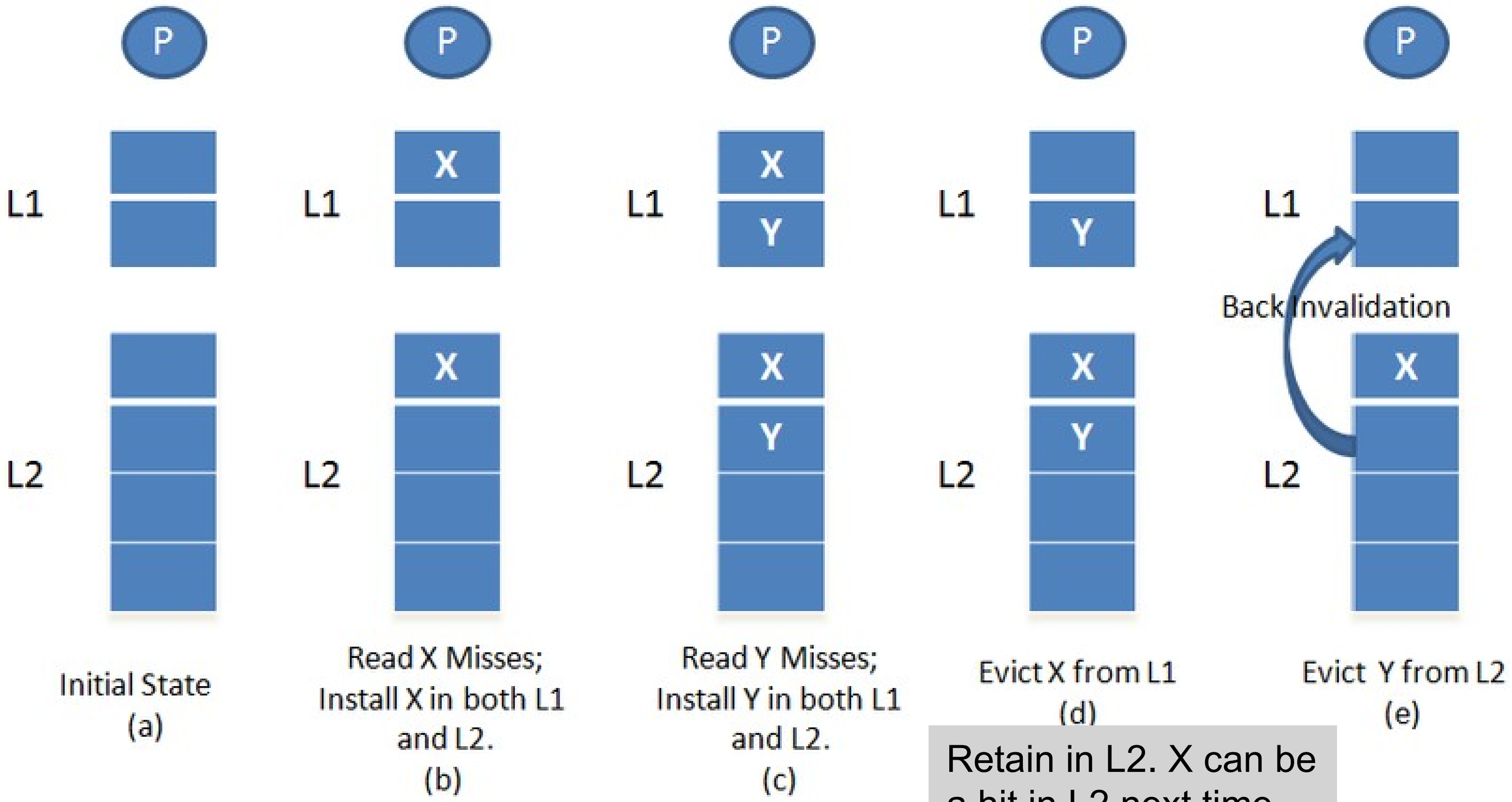
16 words each containing 4 bytes

Inclusion and exclusion policy in multi level caches

Inclusive policy

These are set associative L1 and L2 caches, not Direct mapped caches

L2 ways >> L1 ways



Initial State
(a)

Read X Misses;
Install X in both L1
and L2.
(b)

Read Y Misses;
Install Y in both L1
and L2.
(c)

Evict X from L1
(d)

Evict Y from L2
(e)

if there is an eviction of Y from L2, it sends a back invalidation to the L1 cache, so that inclusion is not violated.

Eliminate from L1 as well

Retain in L2. X can be a hit in L2 next time

Every block existing in the first level also exists in the next level. L1 is a subset of L2

L2 is said to be inclusive of L1
Intel quad- core processor with inclusive L2 caches and L3 cache

X	Y

X, Y, Z, A, B

Z	Y

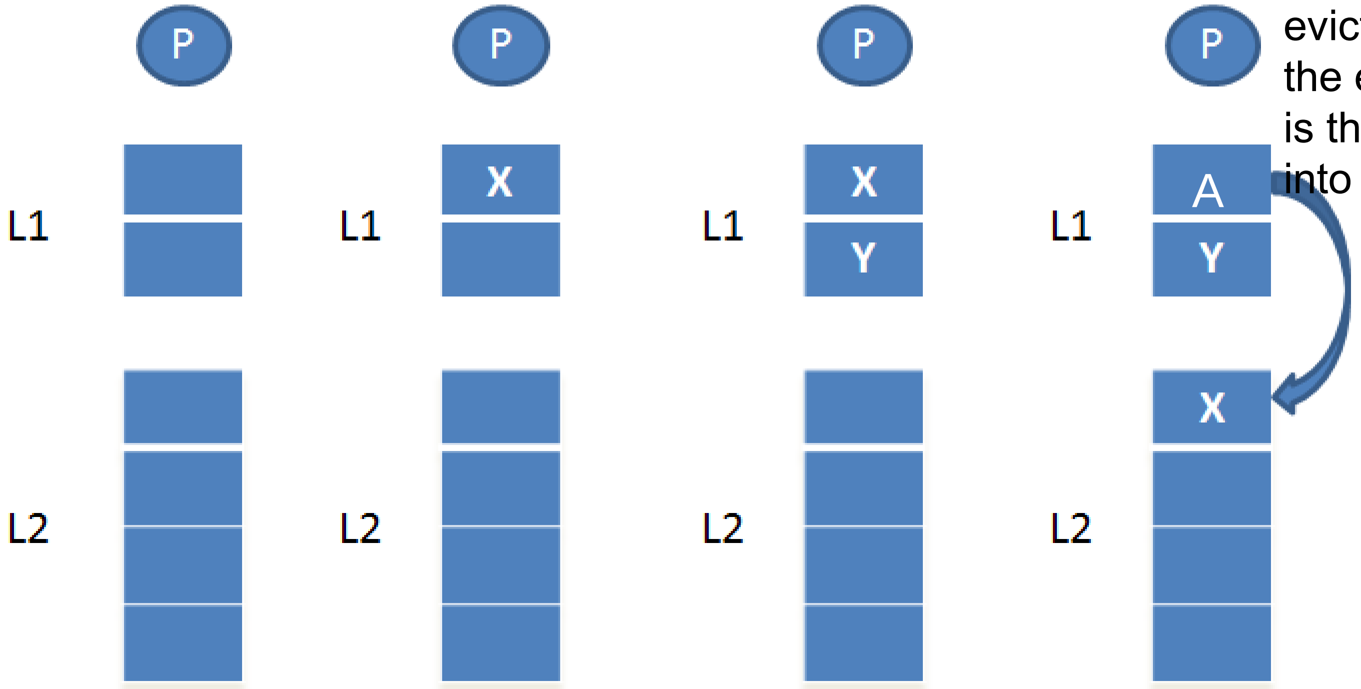
Z	A

X	Y		

X	Y	Z	

X	Y	Z	A

Exclusive policy



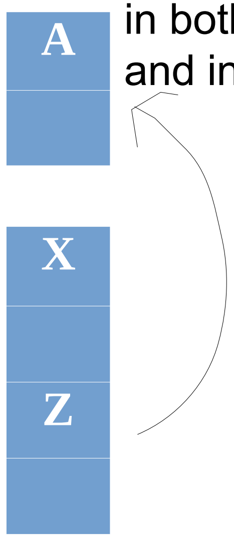
Initial State
(a)

Read X Misses;
Install X in L1 only.
(b)

Read Y Misses;
Install Y in L1 only.
(c)

Evict X from L1 and
place in L2
(d)

block to be evicted from L1, the evicted block is then placed into L2.



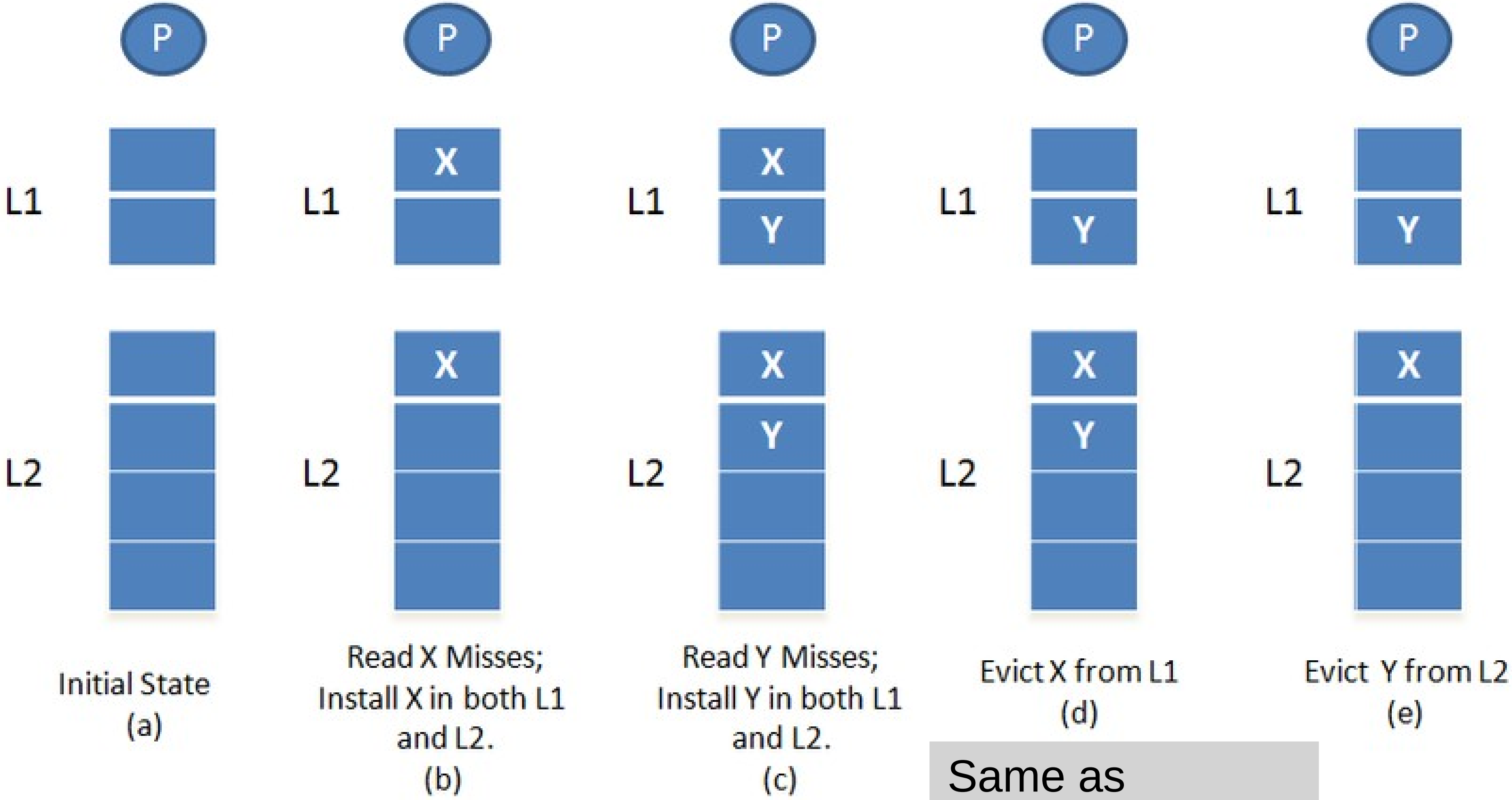
Z: L1 miss, L2 hit. Move from L to L1-

in both exclusive and inclusive

If the lower level cache contains only blocks that are not present in the higher level cache, then the lower level cache--> exclusive of the higher level cache

AMD Opteron with 512kB L2 cache – exclusive of L1

Non-inclusive, non-exclusive policy (NINE)



Same as inclusive till here

No back-invalidation (like in inclusive)
Retain in L1 even if not present in L2

AMD Opteron with NINE L3 cache

Comparison

- Inclusive:

  + ??

  - ??

- Exclusive

  + ??

  - ??

Comparison

- Inclusive:

  + L1 evictions do not fill up L2--> silent eviction of clean lines, which is faster, less power

  + advantageous if L1 is small. Duplication is less.

  - Wasted cache capacity due to duplication

  - Maintaining inclusion takes effort (forced evictions from L1) and back invalidation

  - More latency – to fill up data in the 2 cache levels

  - L1 miss --> Higher chances of L2 or L3 miss as compared to exclusive since half the data will be same

Comparison

- Exclusive – advantageous if L1 is large

  + More unique memory capacity, better utilisation of cache space

  + Fill up L1, without filling up L2- faster

  + L1 miss --> Higher chances of L2 or L3 hit as compared to inclusive since the data will be different

  + L2 eviction, need not evict from L1

  - L1 eviction fills up L2 (Fills up L2 even if there is no miss in L2). More work is needed here

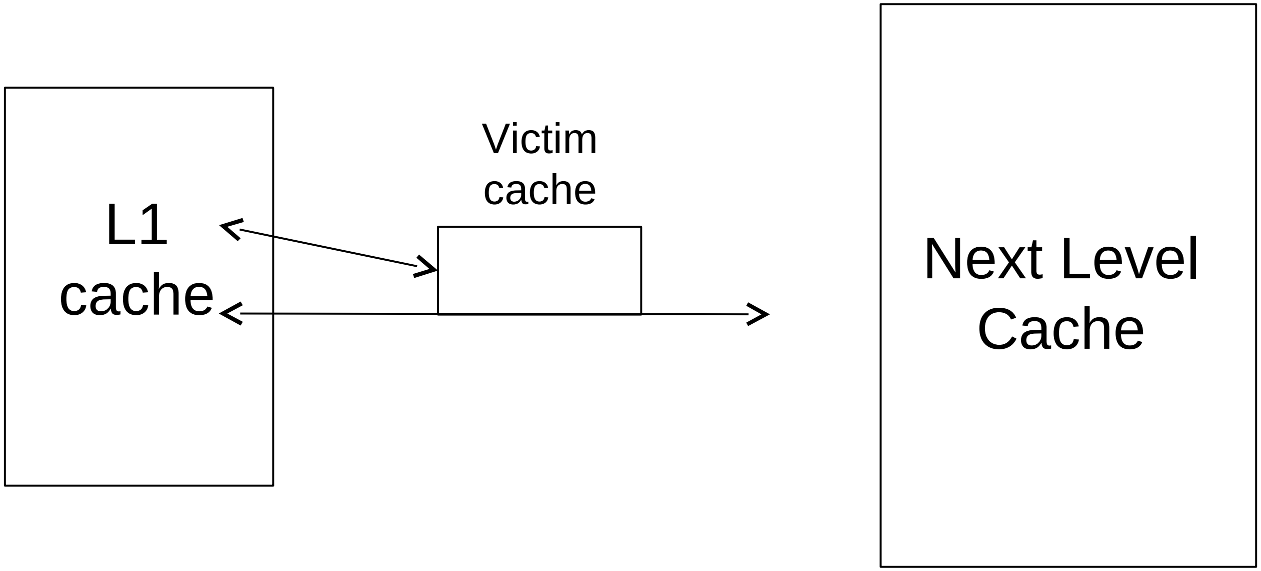
  - L2 needs to be large enough to take into account L1's evictions

- NINE

  + L2 Fills up only on a miss

  + L1 evicts only on a local miss (not on an L2 eviction)

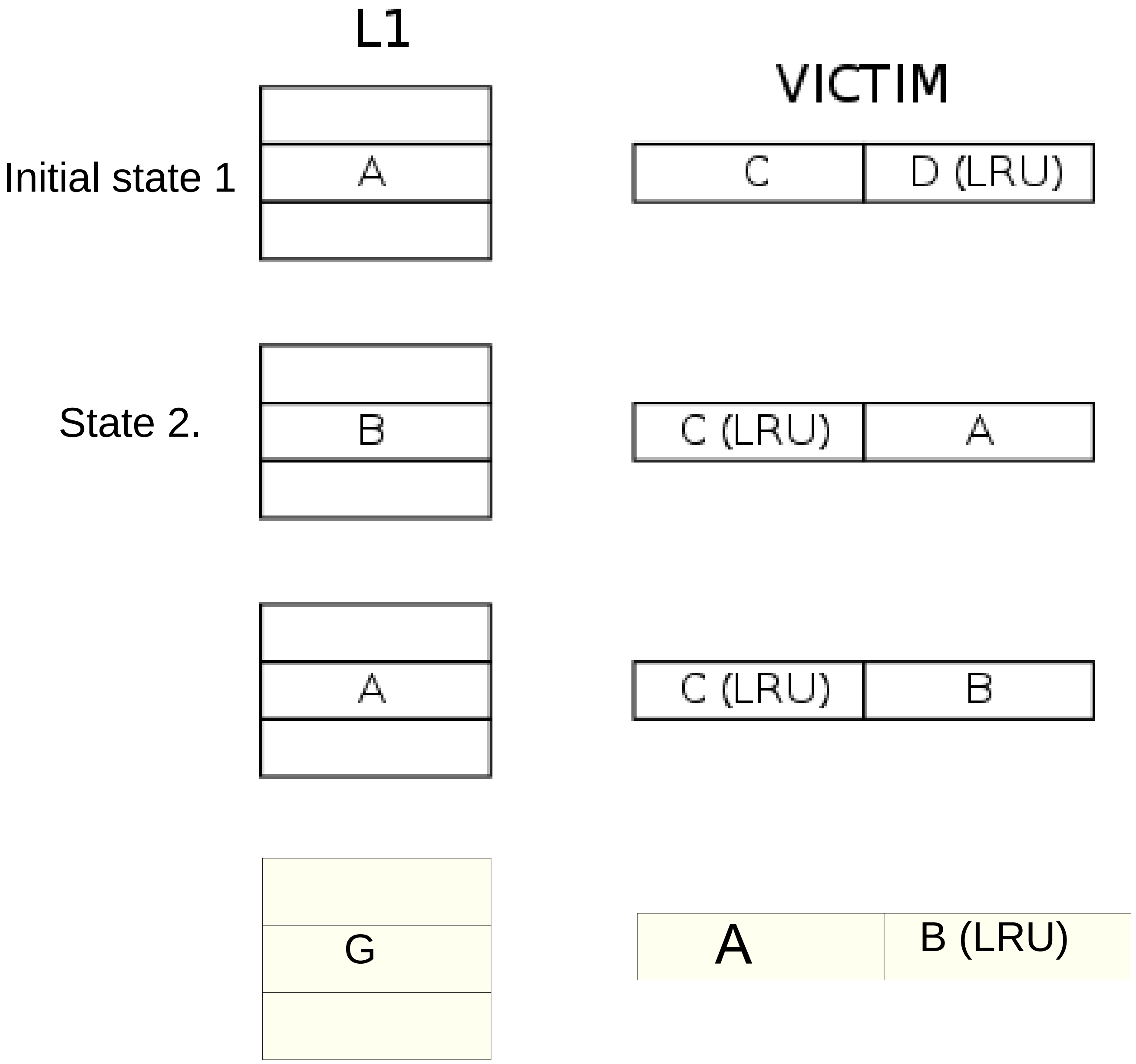
Victim Cache: Reducing Conflict Misses



- Jouppi, “Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers,” ISCA 1990.
- Victim: Use a small (fully-associative) buffer of 4 to 8 entries, to store evicted blocks before going to L2
- Exclusive L2 serves as a victim cache for L1

Victim Cache: Reducing Conflict Misses

Assume sequence:
A, B, A, G ....



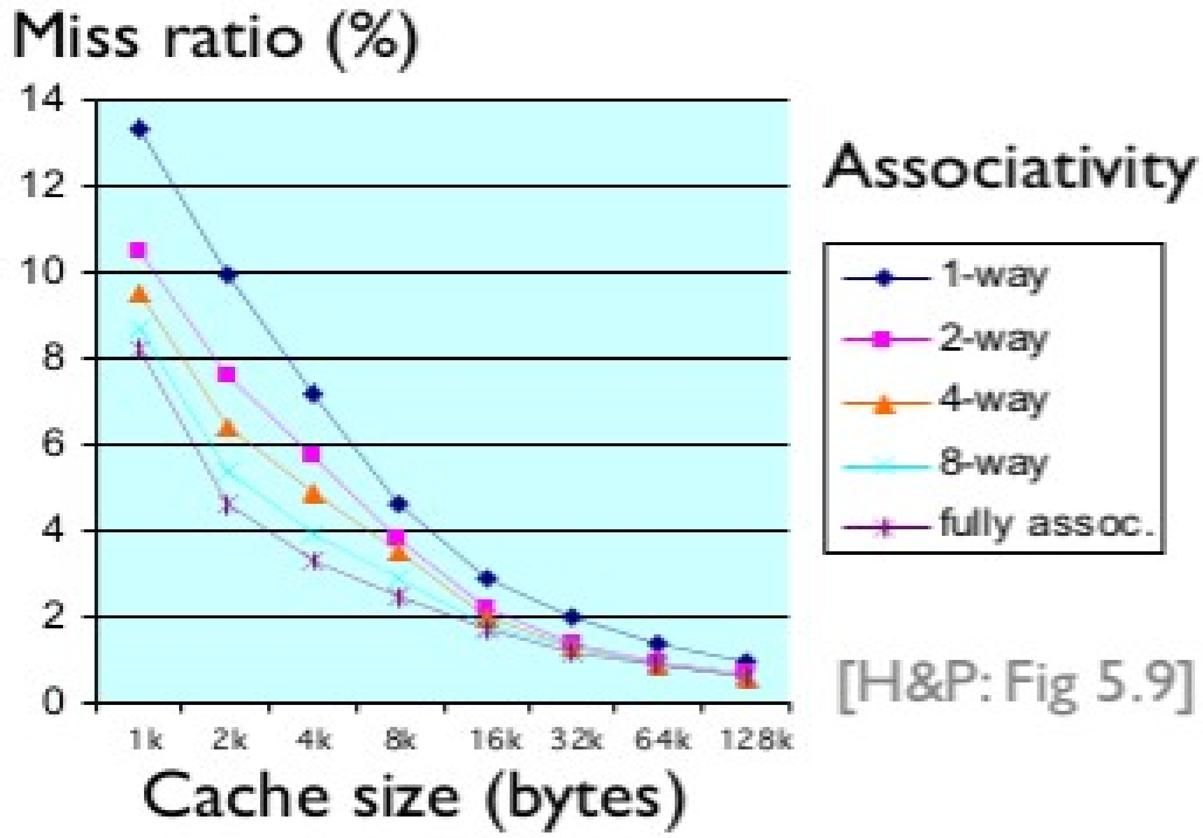
B is Cache Miss, Evivt A from L1, move to victim

A is Cache Miss, Victim Hit: Swap contents of victim and L1

Cache Miss, Victim Miss, Evict A and put it in the Victim. Evict the LRU C from the Victim cache

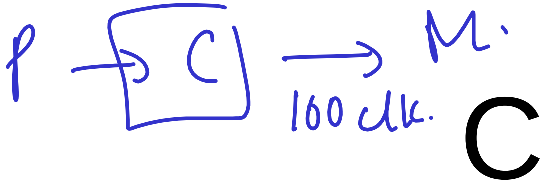
14

Associativity Tradeoffs

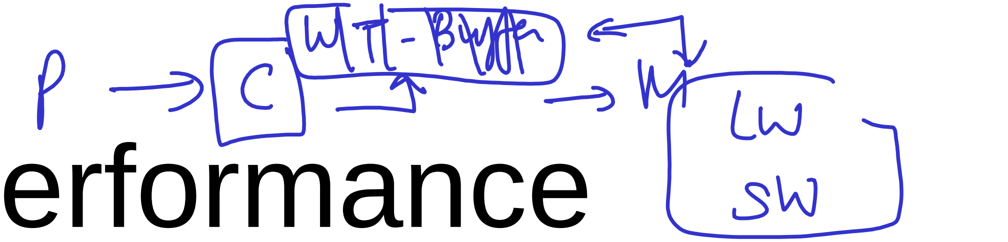


[H&P: Fig 5.9]

Increasing associativity requires more comparators and more tag bits per cache block.



CPU performance



ALU ← cpu.

Program

1

- CPU time = (CPU execution clock cycles + Memory-stall clock cycles) x Clock cycle time

2

- Memory-stall clock cycles = (Read-stall cycles + Write-stall cycles)

loads. LW/LB/LH. 10 x 2%

- Read-stall cycles = Reads per Program * Read miss rate * Read miss penalty
- Write-stall cycles = (Writes per Program * Write miss rate * Write miss penalty) + Write buffer stalls

CPU performance

Ignoring Write buffer stalls

- Memory-stall clock cycles = (Read-stall cycles + Write-stall cycles)
- Memory-stall clock cycles = Memory accesses per Program * Miss rate * Miss penalty
  - Miss rate combines read and write miss rates

Loads + Store

Read / write

R / W

4 Block L

D-cache 100 → 30

LW/SW.

Example 10

MUL ←
LW ←
SW ←
Add c

I-cache

I

Calculate the number of memory stall cycles

MR

Im

- Assume the miss rate of an instruction cache is 2% and the miss rate of the data cache is 4%. Assume miss penalty is 100 cycles for all misses, and the number of instructions is 1000.

LW/SW → 30%

- Assume the frequency of all loads and stores (Data accesses) is 30%.

$$\Rightarrow \left(\frac{30}{100} I\right) \times \frac{4}{100} \left(4/100 \, I\right)$$

Instructions per Program * Miss rate * Miss penalty

I-MISS

(LW-SW)

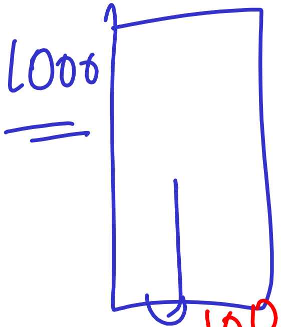
PC →

IF | ID EX M W.

OOO →

D-MISS

Solution



200 cycles

100 clk cycles

- Instruction miss cycles = I x 2% x 100 =2 ~~I~~ 100
- Data miss cycles = I x 30% x 4% x 100 = 1.2 I
- Memory-stall cycles = (2 + 1.2) I

1.2x 100

120

320 clk

CPU performance

Avg (Clk - Per instr) = 1.

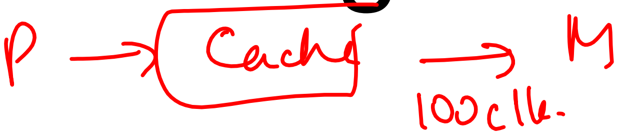
- Total CPI = Base CPI + Memory-stall cycles per instruction

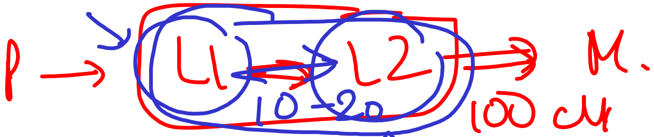
1. + 1.2

1 + 2.3

= 2 > 1

Local and global miss rate

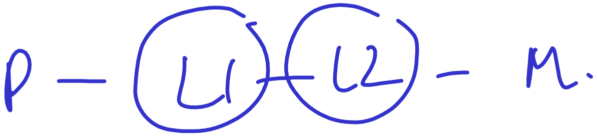




- Local miss rate: Fraction of references to one level of a cache that miss
  - e.g. L2 local MR = L2 misses/L1 misses

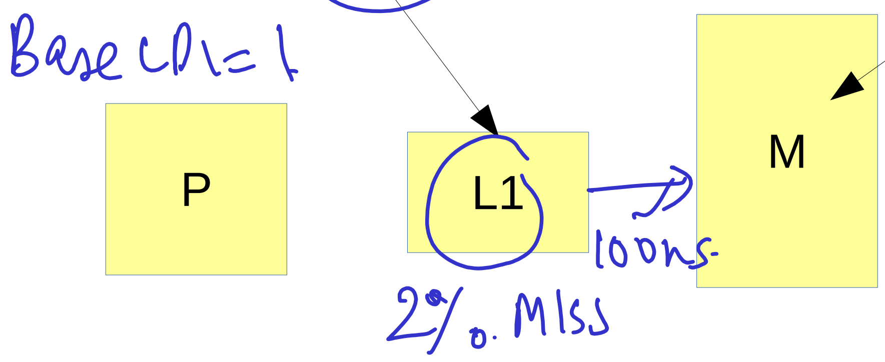
L1 – L2 – L3

- Global miss rate: Fraction of all references that miss in all levels of a multilevel cache
  - Property of the overall memory hierarchy
  - Global MR is the product of all local MRs
- Global MR L2
  - Fraction of total accesses that miss at L1 and L2



Example 11

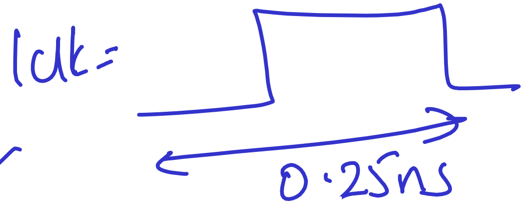
- Suppose we have a processor with a base CPI of 1.0, assuming all references hit in the primary cache, and a clock rate of 4 GHz. Assume a main memory access time of 100 ns. Suppose the miss rate per instruction at the primary cache is 2%, what is the total CPI?



Solution

For the processor with one level of caching,

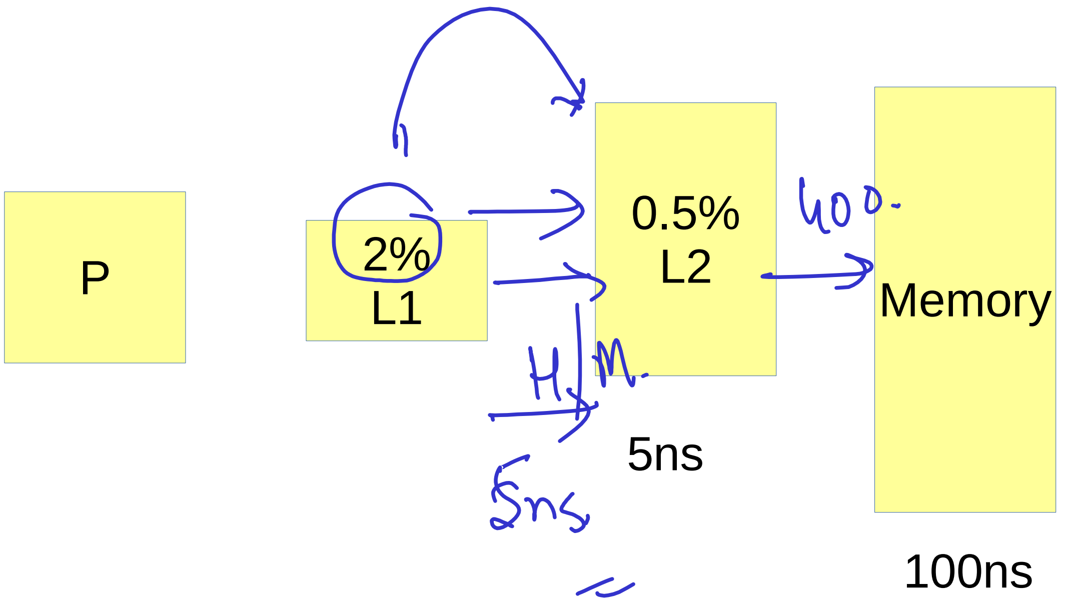
- Total CPI = Base CPI + Memory-stall cycles per instruction
- Total CPI = Base CPI + (Miss rate * Miss penalty in terms of clock cycles)
- Clock = 4G --> Period = 0.25ns
- Miss penalty = 100ns / 0.25 ns = 400 clock cycles
- Total CPI= 1.0 + $\underset{\text{MISS}}{2\%}$ x $\underbrace{400}_{\text{clk cycle}}$ = 9



Example 12

- Add a secondary cache that has a 5 ns access time for either a hit or a miss and has a “global” miss rate to main memory of 0.5% (reduced). What is the CPI?

L2



G-MR - L2 → $\frac{0.5}{100}$

L-MR L2

$\frac{2}{100} \times \frac{0.5}{100}$

Solution

2% x 20-

MR x MPenalty

=

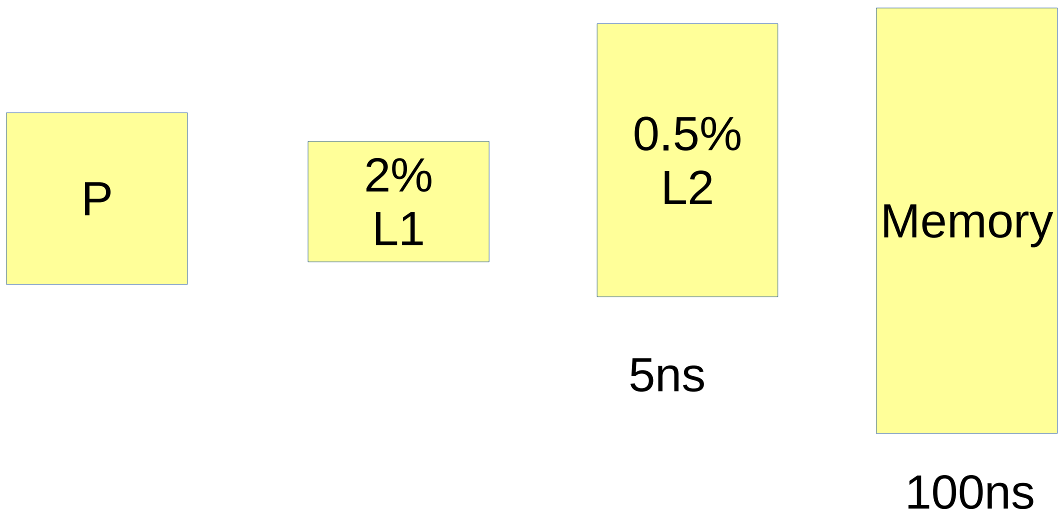
$\frac{0.5}{100} \times 400$

5ns --> 20 clock cycles

For a 2 level cache:

- Total CPI = Base CPI + L1 stalls + L2 stalls
- = 1 + (2% * 20 clk to go to L2) + (0.5% * 400 clk cycles to go to main memory)
- Total CPI= 1.0 + (2% x 20 clock cycles) + (0.5% x 400) = 3.4
- Speed up with L2 = 9/3.4 = 2.6

+L



→ Multi-levels
L1-L3

→ Caching scheme
↓MR

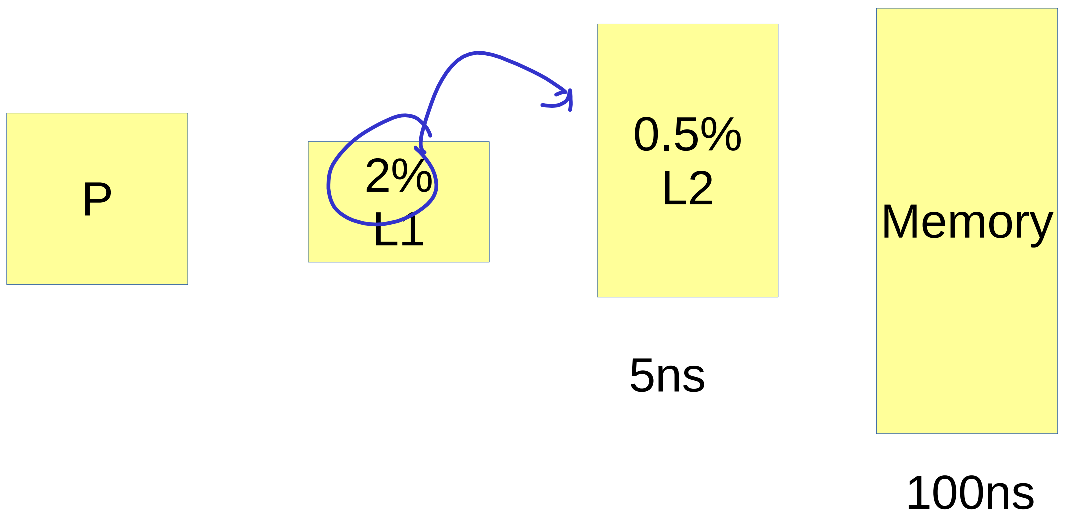
Solution

Alternately, if the L2 has a "local" miss rate of 0.5%, then it does not include the L1 misses. In this case, the equation will be:

4 L2 stalls

= 1 + (2% * 20 clk to go to L2) + (0.5% *2% * 400 clk cycles to go to main memory)

L1



0

$\frac{2}{100} \times \frac{0.5}{100}$

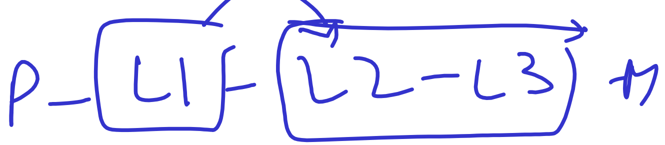
How to reduce miss rates?

- ✓ Higher block size?
- ✓ Higher cache size?
- ✓ More hierarchies? Multi-level cache L1- L4
- ✓ Higher associativity?

Summary

AMAT = Access Time for 1st level + Miss Rate × MissPenalty

- Larger cache size: Lower miss rate, higher access time, more power
- Larger block size: Take advantage of spatial locality, Higher miss penalty
- More associativity (ways): Lower miss rate
- More intelligent replacement: Lower miss rate, higher cost
- Write policy: More complexity
- How to choose? Simulate different cache organizations on real programs



Multilevel Cache AMAT

Avg mem access time hit time Miss penalt

- (1) AMAT = L1 HT + L1 MR × L1 Miss penalty
- L1 MP = L2 HT + L2 MR × L2 MP

hi

L3 HT + L3MR × L3 MP

- For two levels:
  - AMAT = L1 HT + L1 MR × (L2 HT + L2 MR × L2 MP)

P
L1
5ns
2ns
L2
10ns

Acknowledgements

- https://www.cc.gatech.edu/~hyesoon/fall10/hw3_sol.html