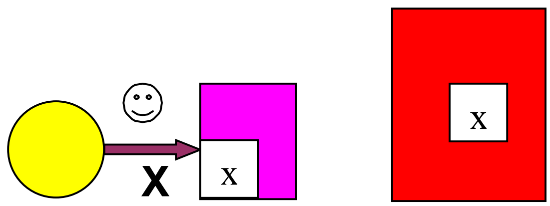
Writing into Cache

Case 1. Write hit



(store X: X is in C)

Write through	Write back
Write into C & M	Write into C only. Update M only when discarding the block containing x

Q1. Isn't write-through inefficient?

Not all cache accesses are for write.

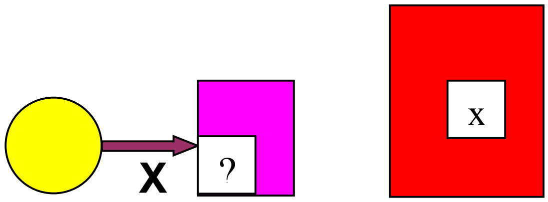
Q2. What about data consistency in write-back cache?

If M is not shared, then who cares?

Most implementations of **Write through** use a **Write Buffer**.

How does it work?

Case 2. Write miss



(Store X, X is NOT in C)

Write allocate

Allocate a C-block to X.

Load the block containing X from M to C.

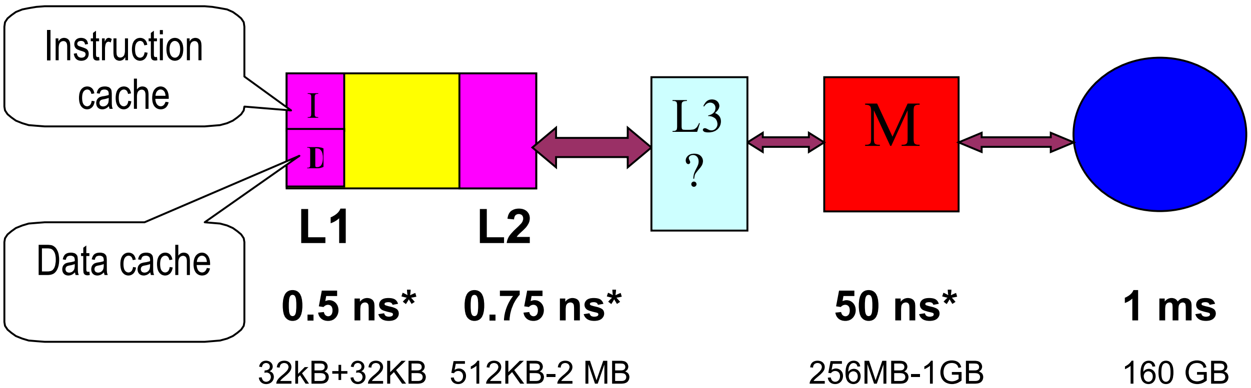
Then write into X in C.

.

Write around

Write directly into X bypassing C

A state-of-the-art memory hierarchy



Reading Operation

- Hit in L1.
- Miss in L1, hit in L2, copy from L2.
- Miss in L1, miss in L2, copy from M.

Write Hit

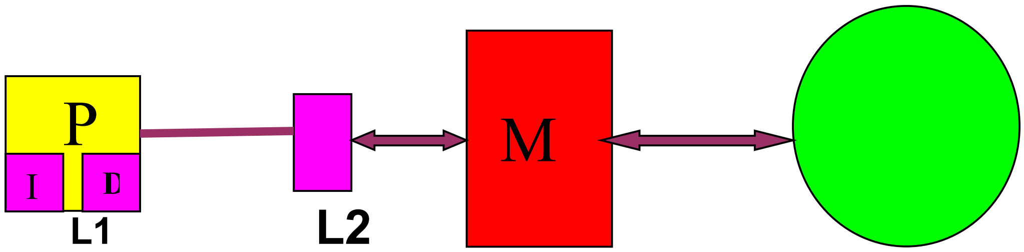
- Write through: Write in L1, L2, M.
- Write back

  Write in L1 only. Update L2 when discarding an L1 block. Update M when discarding a L2 block.

Write Miss

Write-allocate or write-around

Inclusion Property



In a consistent state,

- Every valid L1 block can also be found in L2.
- Every valid L2 block can also be found in M.

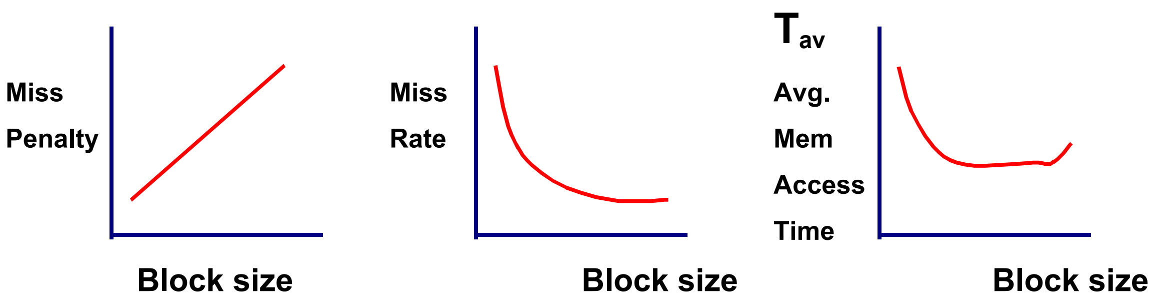
Average memory access time =

$(\text{Hit time})_{L1} + (\text{Miss rate})_{L1} \times (\text{Miss penalty})_{L1}$

$(\text{Miss penalty})_{L1} = (\text{Hit time})_{L2} + (\text{Miss rate})_{L2} \times (\text{Miss penalty})_{L2}$

Performance improves with additional level(s) of cache if we can afford the cost.

Optimal Size of Cache Blocks



Large block size supports program locality and reduces the miss rate.

But the miss penalty grows linearly, since more bytes are copied from M to C after a miss.

T_{av} = Hit time + Miss rate x Miss penalty.

The optimal block size is 8-64 bytes. Usually, I-cache has a higher hit ratio than D-cache. Why?