

**cache hit**

If the cache reports a hit, the computer continues using the data as if nothing happened.

**cache miss**

For a cache miss, we can stall the entire processor, essentially freezing the contents of the temporary and programmer-visible registers, while we wait for memory.



We can now define the steps to be taken on an instruction cache miss:

1. Send the original PC value to the memory. (PC-4)
2. Instruct main memory to perform a read and wait for the memory to complete its access.
3. Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address (from the ALU) into the tag field, and turning the valid bit on.
4. Restart the instruction execution at the first step, which will refetch the instruction, this time finding it in the cache.



Time Sequence	1	2	3	4	5	6	7	8	thrashing
	1	2	3	4	1	2	3	4	
FIFO	1	1	1*	4	4	4*	3	3	No hit
n=3		2	2	2*	1	1	1*	4	
			3	3	3*	2	2	2*	
LRU	1	1	1*	4	4	4*	3	3	No hit
n=4		2	2	2*	2	2	2*	2	
			3	3	3*	3	3	3*	
OPT	1	1	1	4	4	4	4	4	Hit 3 times
		2	2	2	2	2*	3*	3	
			3*	4*	4	4	4	4*	

➤ Hit rate is related to access sequence.



# Stack replacement algorithm

- $B_t(n)$  represents the set of access sequences contained in a cache block of size  $n$  at time  $t$ .
- $B_t(n)$  is the subset of  $B_t(n + 1)$ .



# LRU replacement algorithm is stack replacement algorithm

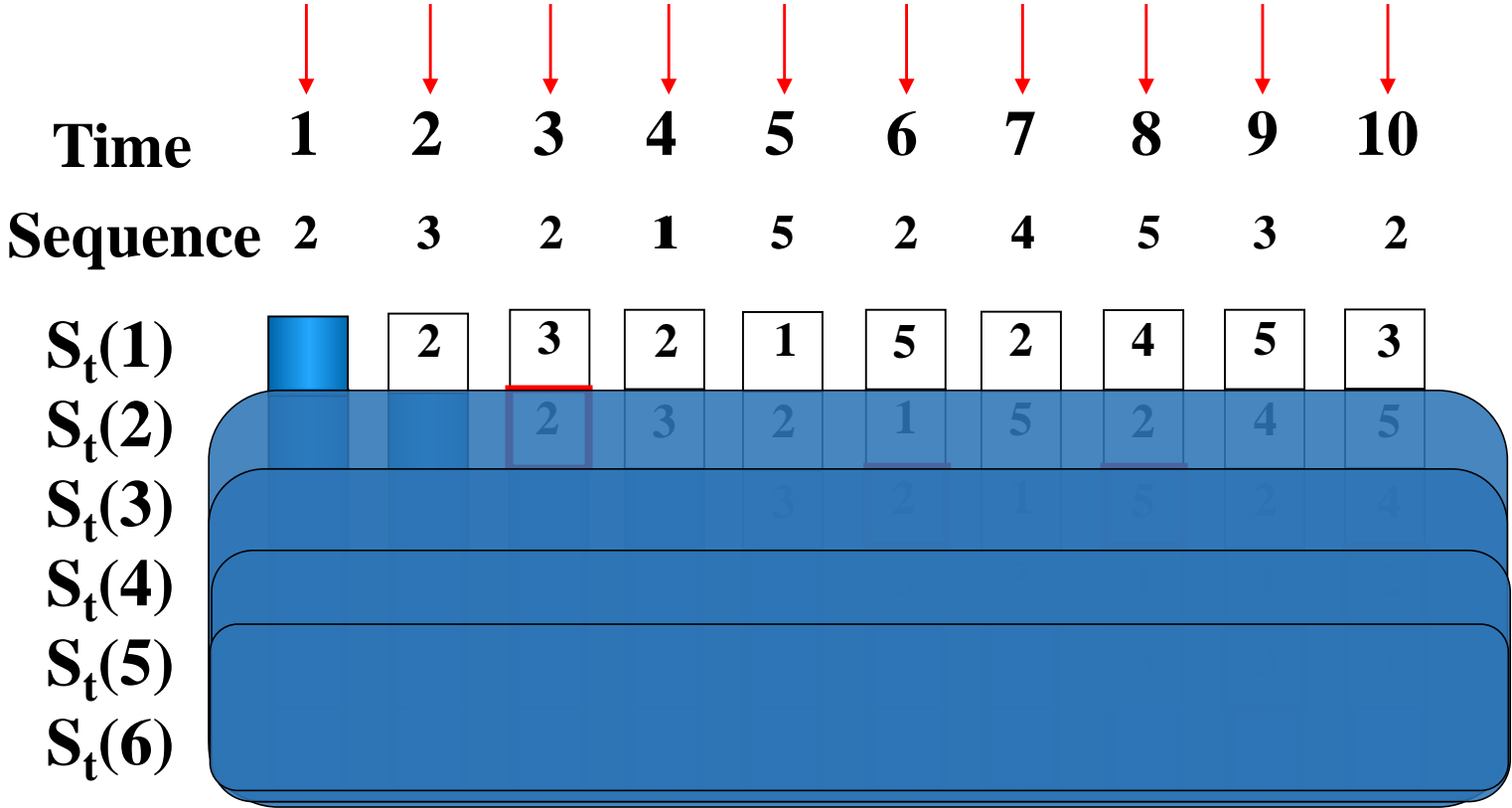
Time	1	2	3	4	5	6	7	8	9	10	11	12
Sequence	1	2	3	4	1	2	5	1	2	3	4	5
LRU $n=3$ $Bt(n)$	<div>1</div>	<div>1 2</div>	<div>1* 2 3</div>	<div>4 2* 3</div>	<div>4 1 3*</div>	<div>4* 1 2</div>	<div>5 1* 2</div>	<div>5 1 2*</div>	<div>5* 1 2</div>	<div>3 1* 2</div>	<div>3 4 2*</div>	<div>3* 4 5</div>
LRU $n+1=4$ $Bt(n+1)$	<div>1</div>	<div>1 2</div>	<div>1 2 3</div>	<div>1* 2 3 4</div>	<div>1 2* 3 4</div>	<div>1 2 3* 4</div>	<div>1 2 5 4*</div>	<div>1 2 5 4*</div>	<div>1 2 5 4*</div>	<div>1 2 5* 3</div>	<div>1* 2 4 3</div>	<div>5 2* 4 3</div>



Time	1	2	3	4	5	6	7	8	9	10	11	12
Sequence	1	2	3	4	1	2	5	1	2	3	4	5
FIFO n=3	<div><div>1</div><div></div><div></div></div>	<div><div>1</div><div>2</div><div></div></div>	<div><div>1*</div><div>2</div><div>3</div></div>	<div><div>4</div><div>2*</div><div>3</div></div>	<div><div>4</div><div>1</div><div>3*</div></div>	<div><div>4*</div><div>1</div><div>2</div></div>	<div><div>5</div><div>1*</div><div>2</div></div>	<div><div>5</div><div>1*</div><div>2</div></div>	<div><div>5</div><div>1*</div><div>2</div></div>	<div><div>5</div><div>3</div><div>2*</div></div>	<div><div>5*</div><div>3</div><div>4</div></div>	<div><div>5*</div><div>3</div><div>4</div></div>
FIFO n+1=4	<div><div>1</div><div></div><div></div><div></div></div>	<div><div>1</div><div>2</div><div></div><div></div></div>	<div><div>1</div><div>2</div><div>3</div><div></div></div>	<div><div>1*</div><div>2</div><div>3</div><div>4</div></div>	<div><div>1*</div><div>2</div><div>3</div><div>4</div></div>	<div><div>1*</div><div>2</div><div>3</div><div>4</div></div>	<div><div>5</div><div>2*</div><div>3</div><div>4</div></div>	<div><div>5</div><div>1</div><div>3*</div><div>4</div></div>	<div><div>5</div><div>1</div><div>2</div><div>4*</div></div>	<div><div>5*</div><div>1</div><div>2</div><div>3</div></div>	<div><div>4</div><div>1*</div><div>2</div><div>3</div></div>	<div><div>4</div><div>5</div><div>2*</div><div>3</div></div>



# Using LRU



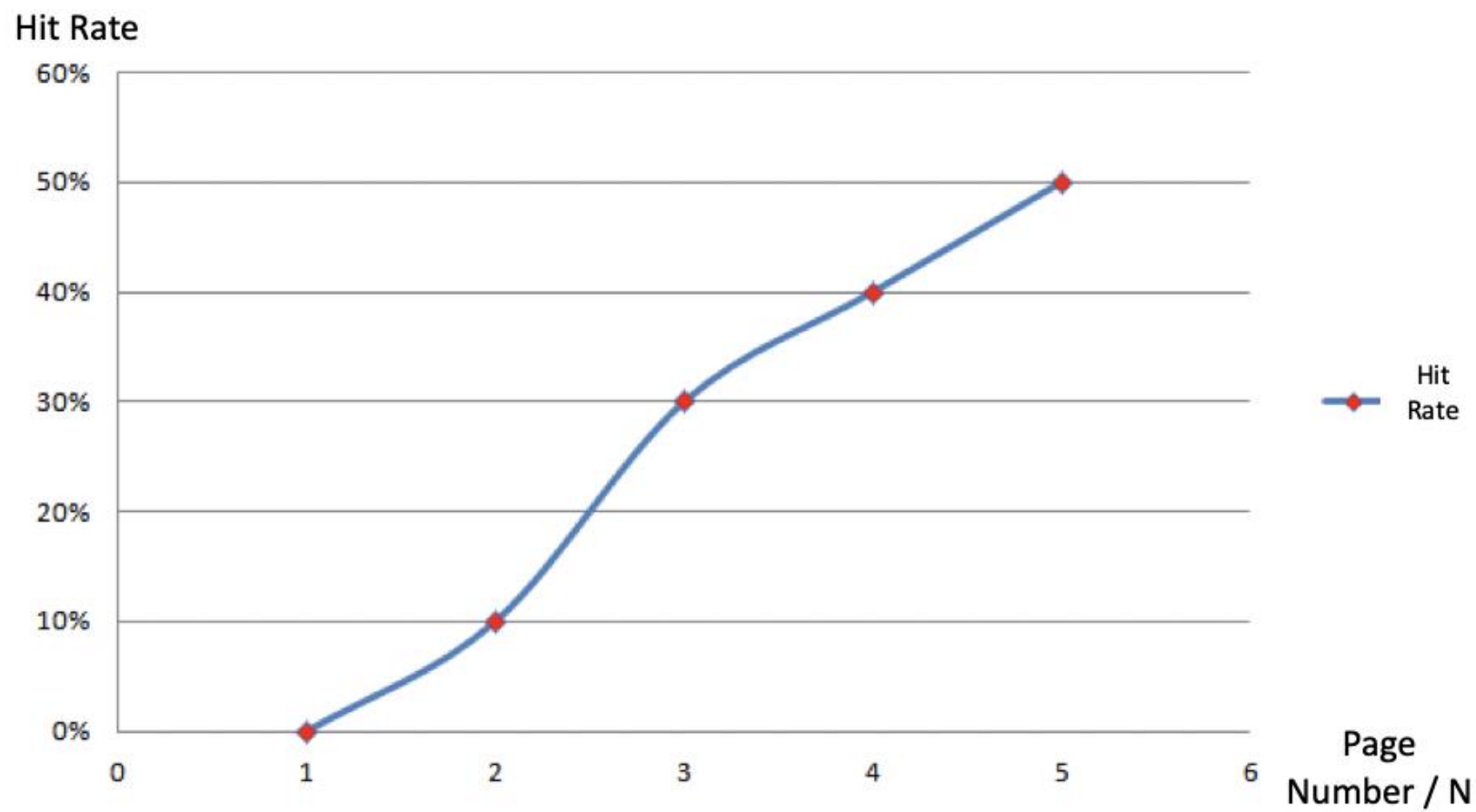
**N=3**

~~Hit~~

Hit

Hit Hit Hit







For LRU algorithm, the hit ratio always increases with the increase of cache block.

Time	1	2	3	4	5	6	7	8	9	10	11	12
Sequence	1	2	3	4	1	2	5	1	2	3	4	5
LRU n=3 Hit 2 times	<div><div>1</div><div></div><div></div></div>	<div><div>1</div><div>2</div><div></div></div>	<div><div>1*</div><div>2</div><div>3</div></div>	<div><div>4</div><div>2*</div><div>3</div></div>	<div><div>4</div><div>1</div><div>3*</div></div>	<div><div>4*</div><div>1</div><div>2</div></div>	<div><div>5</div><div>1*</div><div>2</div></div>	<div><div>5</div><div>1</div><div>2*</div></div>	<div><div>5*</div><div>1</div><div>2</div></div>	<div><div>3</div><div>1*</div><div>2</div></div>	<div><div>3</div><div>4</div><div>2*</div></div>	<div><div>3*</div><div>4</div><div>5</div></div>
LRU n=4 Hit 4 times	<div><div>1</div><div></div><div></div><div></div></div>	<div><div>1</div><div>2</div><div></div><div></div></div>	<div><div>1</div><div>2</div><div>3</div><div></div></div>	<div><div>1*</div><div>2</div><div>3</div><div>4</div></div>	<div><div>1</div><div>2*</div><div>3</div><div>4</div></div>	<div><div>1</div><div>2</div><div>3*</div><div>4</div></div>	<div><div>1</div><div>2</div><div>5</div><div>4*</div></div>	<div><div>1</div><div>2</div><div>5</div><div>4*</div></div>	<div><div>1</div><div>2</div><div>5</div><div>4*</div></div>	<div><div>1</div><div>2</div><div>5*</div><div>3</div></div>	<div><div>1*</div><div>2</div><div>4</div><div>3</div></div>	<div><div>5</div><div>2*</div><div>4</div><div>3</div></div>
					Hit	Hit		Hit	Hit			



Belady

Time	1	2	3	4	5	6	7	8	9	10	11	12
Sequence	1	2	3	4	1	2	5	1	2	3	4	5
FIFO n=3 Hit 3 times	1	1	1*	4	4	4*	5	5	5	5	5*	5*
		2	2	2*	1	1	1*	1*	1*	3	3	3
			3	3	3*	2	2	2	2	2*	4	4
								Hit	Hit			Hit
FIFO n+1=4 Hit 2 times	1	1	1	1*	1*	1*	5	5	5	5*	4	4
		2	2	2	2	2	2*	1	1	1	1*	5
			3	3	3	3	3	3*	2	2	2	2*
				4	4	4	4	4	4*	3	3	3
					Hit	Hit						



# LRU

- How can I implement the LRU replacement algorithm with only ordinary gates and triggers?
- Comparison Pair Method
- Basic idea: Let each cache block be combined in pairs, use a **comparison pair flip-flop** to record the order in which the two cache blocks have been accessed in the comparison pair, and then use a gate circuit to combine the state of each comparison pair flip-flop, you can find the block to be replaced according to the LRU algorithm.



# Example

- There are three cache blocks (A, B, and C), which can be combined into 6 pairs (AB, BA, AC, CA, BC, and CB). Among them, AB and BA, AC and CA, BC and CB are repeated, so only take AB, AC, BC.
- The access sequence of each pair is represented by “comparison pair flip-flops”  $T_{AB}$ ,  $T_{AC}$ , and  $T_{BC}$  respectively.  $T_{AB}$  is "1", which means that A has been accessed more recently than B;  $T_{AB}$  is "0", which means that B has been accessed more recently than A.  $T_{AC}$  and  $T_{BC}$  are similarly defined.



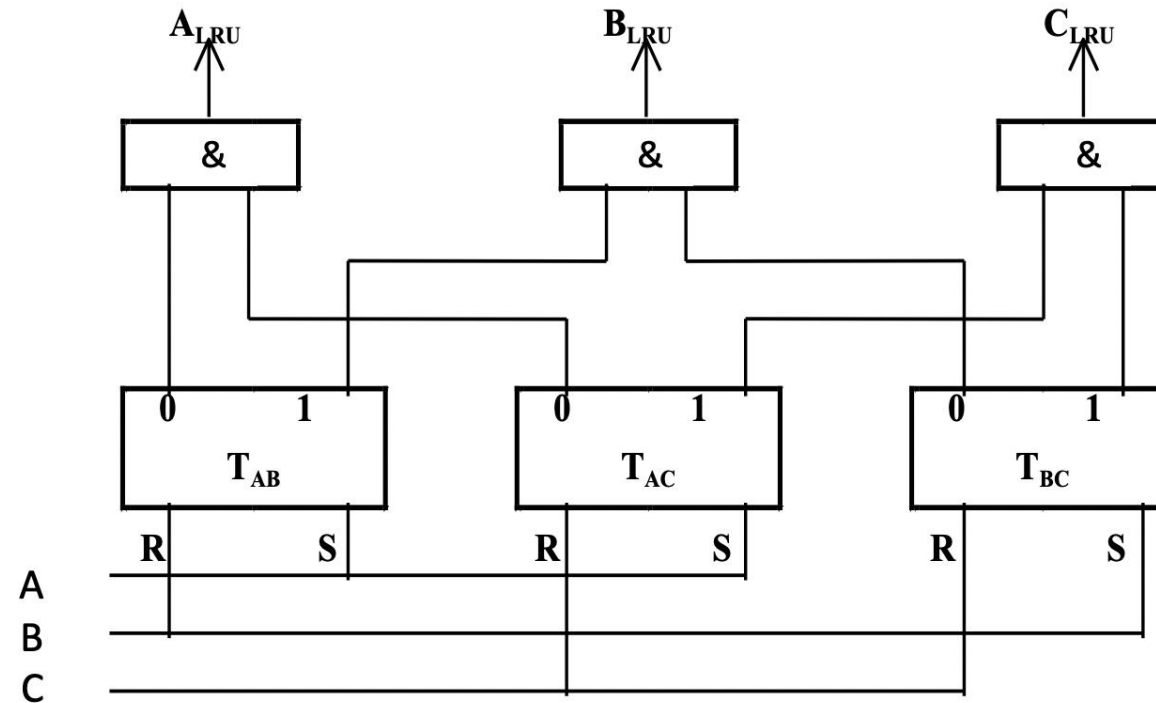
- If the most recently accessed block is A and C is the block that has not been accessed for the longest time, the three flip-flops' states must be respectively:  $T_{AB}=1$ ,  $T_{AC}=1$ , and  $T_{BC}=1$ .
- If the most recently accessed block is B and C is the block that has not been accessed for the longest time, the three flip-flops' states must be respectively:  $T_{AB}=0$ ,  $T_{AC}=1$ , and  $T_{BC}=1$ .
- Therefore, the block C that has not been accessed for the longest will be replaced. In that, the Boolean algebra expression must be:

$$C_{LRU} = T_{AB} \bullet T_{AC} \bullet T_{BC} + \overline{T_{AB}} \bullet T_{AC} \bullet T_{BC} = T_{AC} \bullet T_{BC}$$

$$B_{LRU} = T_{AB} \bullet \overline{T_{BC}}$$

$$A_{LRU} = \overline{T_{AB}} \bullet \overline{T_{AC}}$$





- Change the state of the flip-flop after each access.
  - After accessing block A:  $T_{AB}=1$ ,  $T_{AC}=1$
  - After accessing block B:  $T_{AB}=0$ ,  $T_{BC}=1$
  - After accessing block C:  $T_{AC}=0$ ,  $T_{BC}=0$



# Hardware usage analysis (if $p$ is the number of cache blocks)

- Since each block may be replaced, its signal needs to be generated with an AND gate, so the number of AND gates will be equal to  $p$ .
- Each AND gate receives inputs from its related flip-flops, for example,  $A_{LRU}$  AND gates must have inputs from  $T_{AB}$  and  $T_{AC}$ ,  $B_{LRU}$  must have inputs from  $T_{AB}$  and  $T_{BC}$ , and the number of comparison pair flip-flops is the block number minus 1, so the input number of the AND gate is  $p - 1$ .
- If  $p$  is the block number, for pairwise combination, the number of comparison pair flip-flops should be  $C_p^2$ , which is  $p \cdot (p-1)/2$ .



# The Relationship between the Block Number and Required Hardware for Comparison Pair

Block Number	3	4	6	8	16	64	256
Number of Flip-flop	3	6	15	28	120	2016	32640
Number of And-Gate	3	4	6	8	16	64	256
Input Number of And-Gate	2	3	5	7	15	63	255





## Q4: Write Strategy

- When data is written into the cache (on a store), is the data also written to main memory?
  - If the data is written to memory, the cache is called a **write-through cache**
    - Can always discard cached data - most up-to-date data is in memory
    - Cache control bit: only a valid bit
    - memory (or other processors) always have latest data
  - If the data is NOT written to memory, the cache is called a **write-back cache**
    - Can't just discard cached data - may have to write it back to memory
    - Cache control bits: both valid and dirty bits
    - much lower bandwidth, since data often overwritten multiple times
- Write-through adv: Read misses don't result in writes, memory hierarchy is consistent and it is simple to implement.
- Write back adv: Writes occur at speed of cache and main memory bandwidth is smaller when multiple writes occur to the same block.

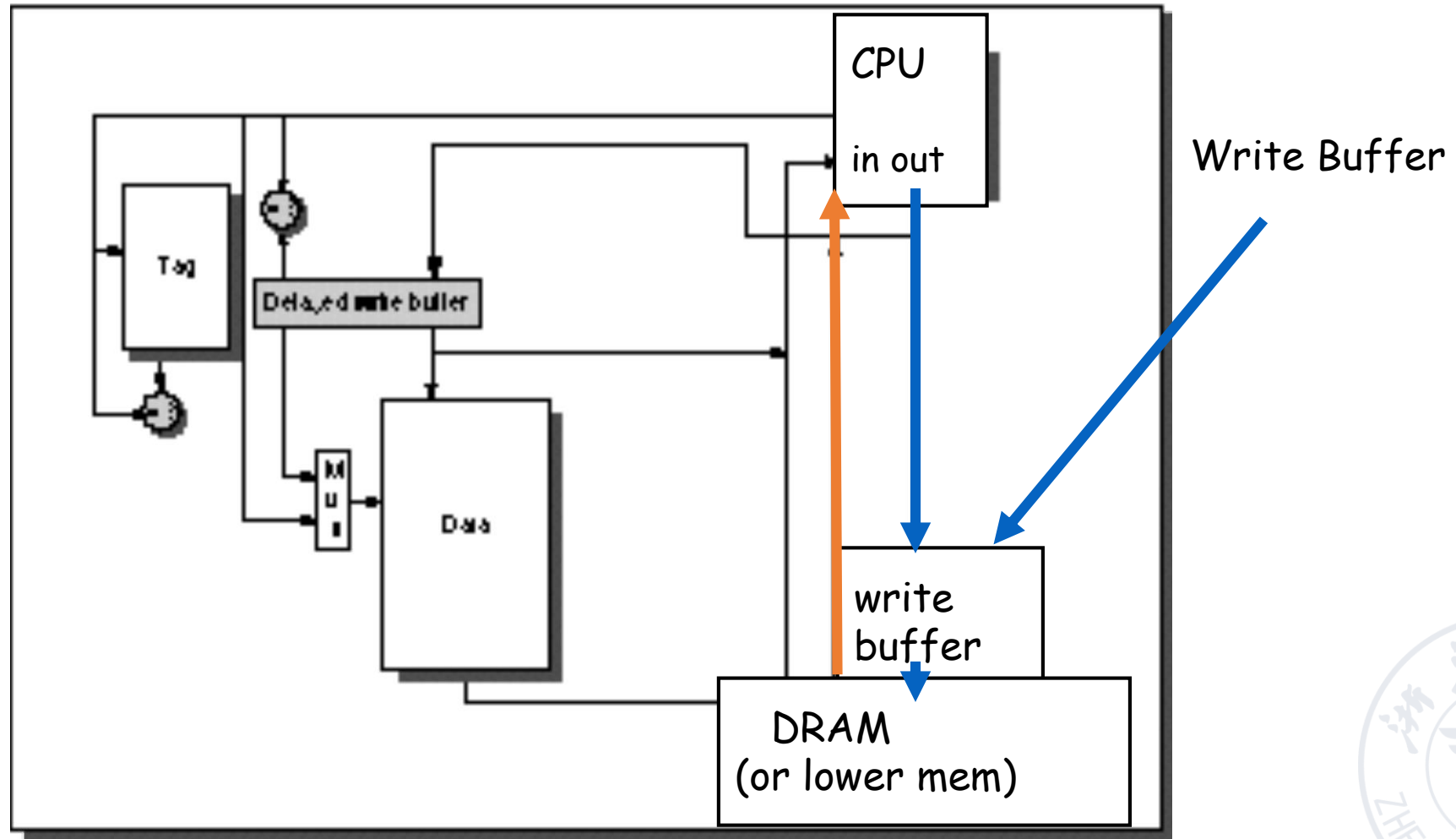


# Write stall

- **Write stall** ---When the CPU must wait for writes to complete during write through
- Write buffers
  - A small cache that can hold a few values waiting to go to main memory.
  - To avoid stalling on writes, many CPUs use a write buffer.
  - This buffer helps when writes are clustered.
  - It does not entirely eliminate stalls since it is possible for the buffer to fill if the burst is larger than the buffer.



# Write buffers



# Write misses

- Write misses
  - If a miss occurs on a write (the block is not present), there are two options.
  - Write allocate
    - The block is loaded into the cache on a miss before anything else occurs.
  - Write around (no write allocate)
    - The block is only written to main memory
    - It is not stored in the cache.
- In general, write-back caches use write-allocate , and write-through caches use write-around .



# Example

- Assume a fully associative write-back cache with many cache entries that starts empty below is a sequence of five memory operations(the address is in square brackets):

```
1  write Mem[100];  
2  write Mem[100];  
3  read  Mem[200];  
4  write Mem[200];  
5  write Mem[100];
```

What are the number of hits and misses when using no-write allocate versus write allocate?

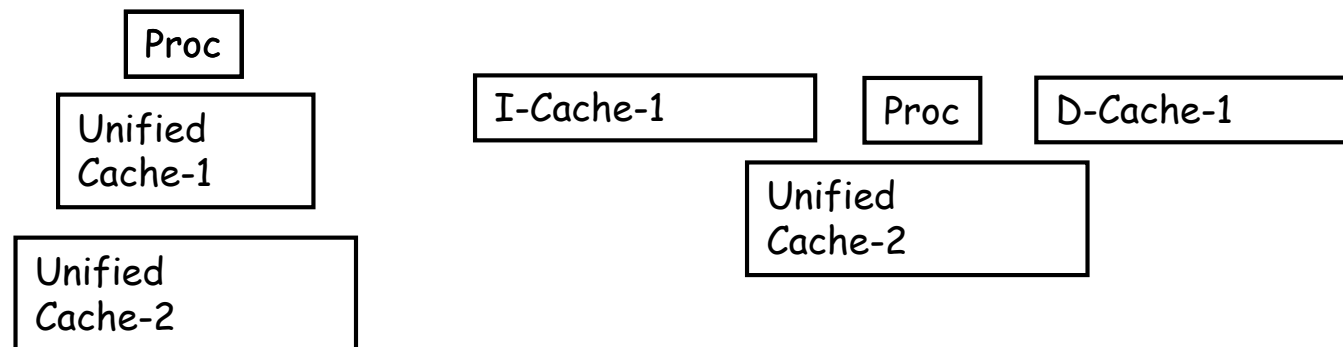
Answer :

for no-write allocate	misses: 1,2,3,5	hit: 4
for write allocate	misses: 1,3	hit: 2,4,5

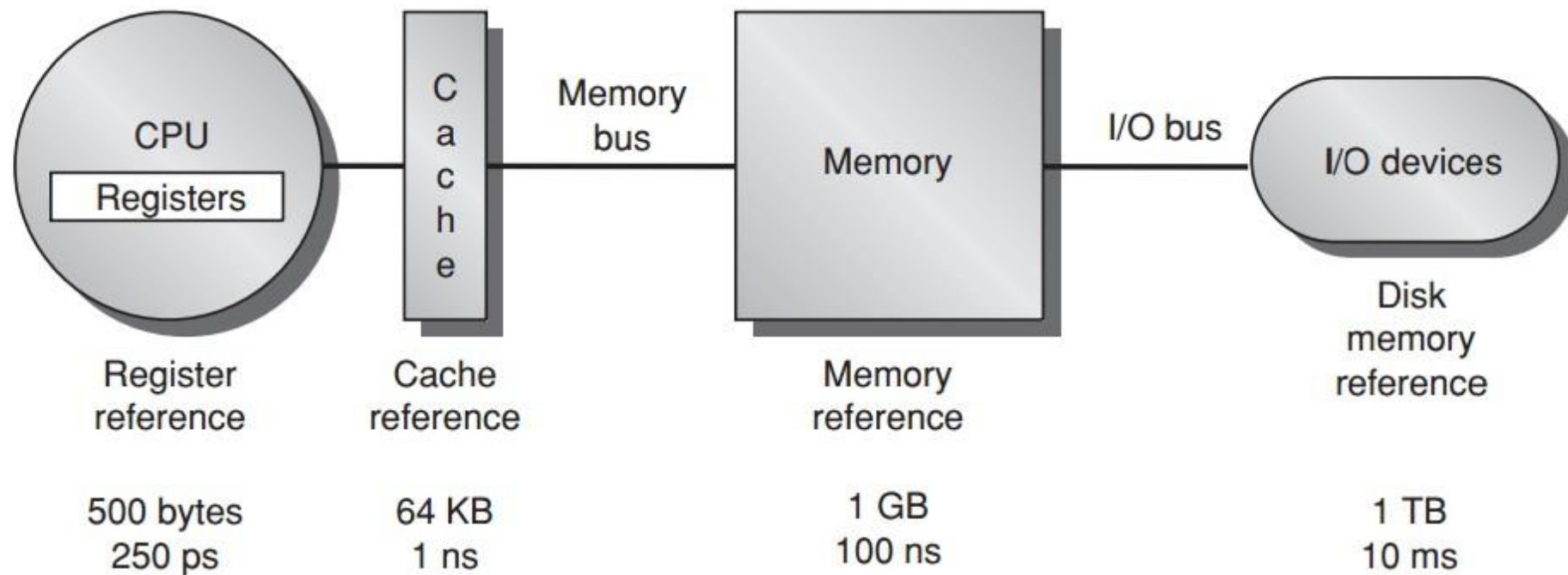


# Split vs. unified caches

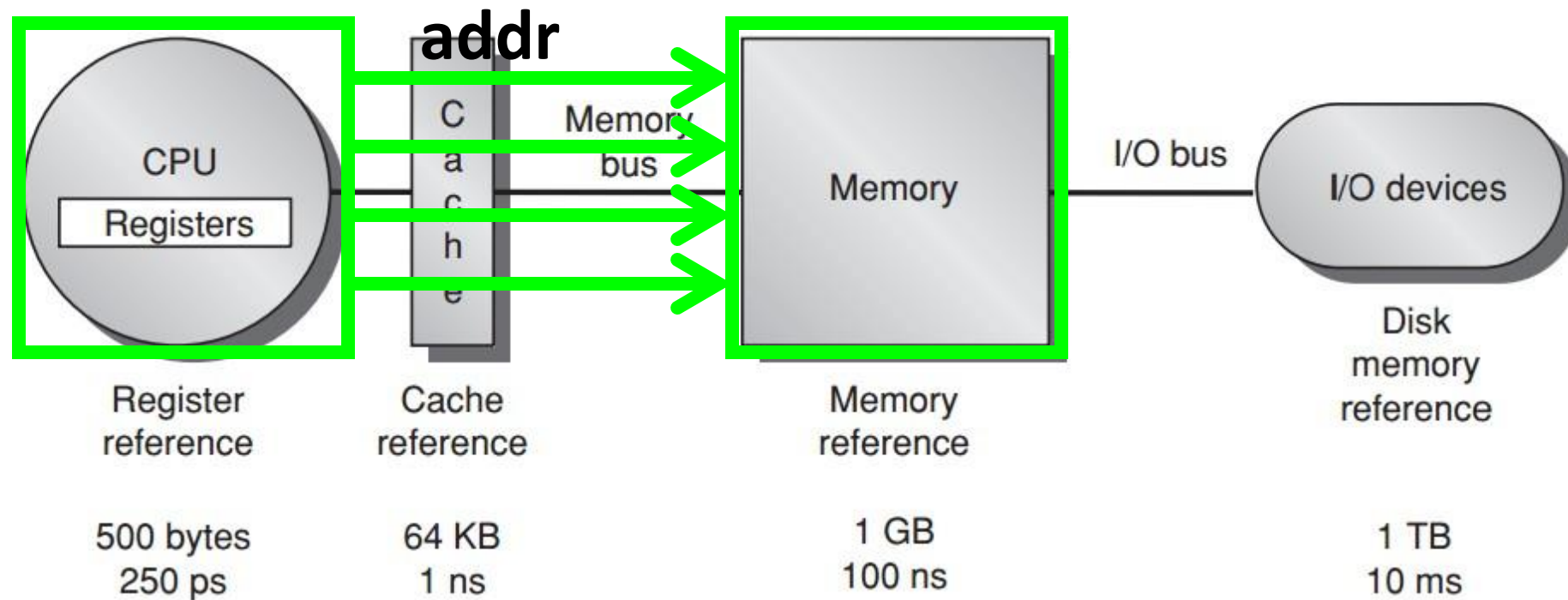
- Unified cache
  - All memory requests go through a single cache.
  - This requires less hardware, but also has lower performance
- Split I & D cache
  - A separate cache is used for instructions and data.
  - This uses additional hardware, though there are some simplifications (the I cache is read-only).



# Memory Hierarchy

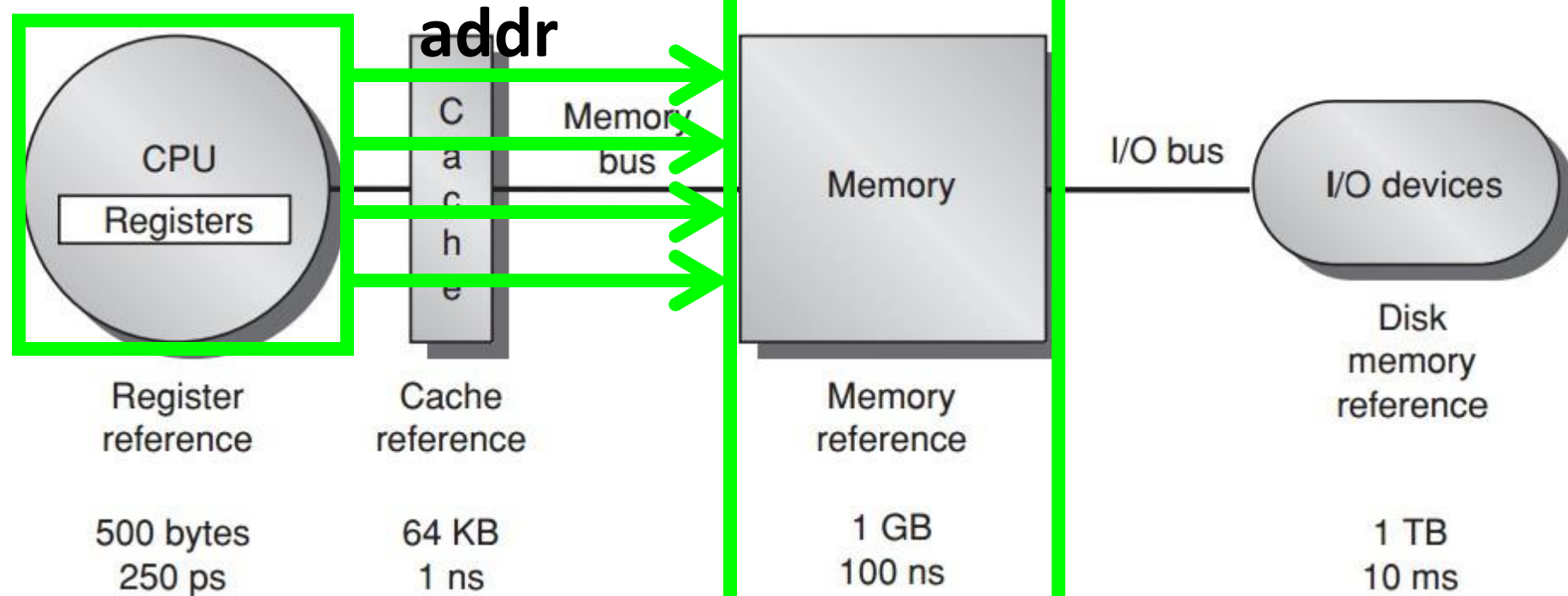


# Memory Hierarchy



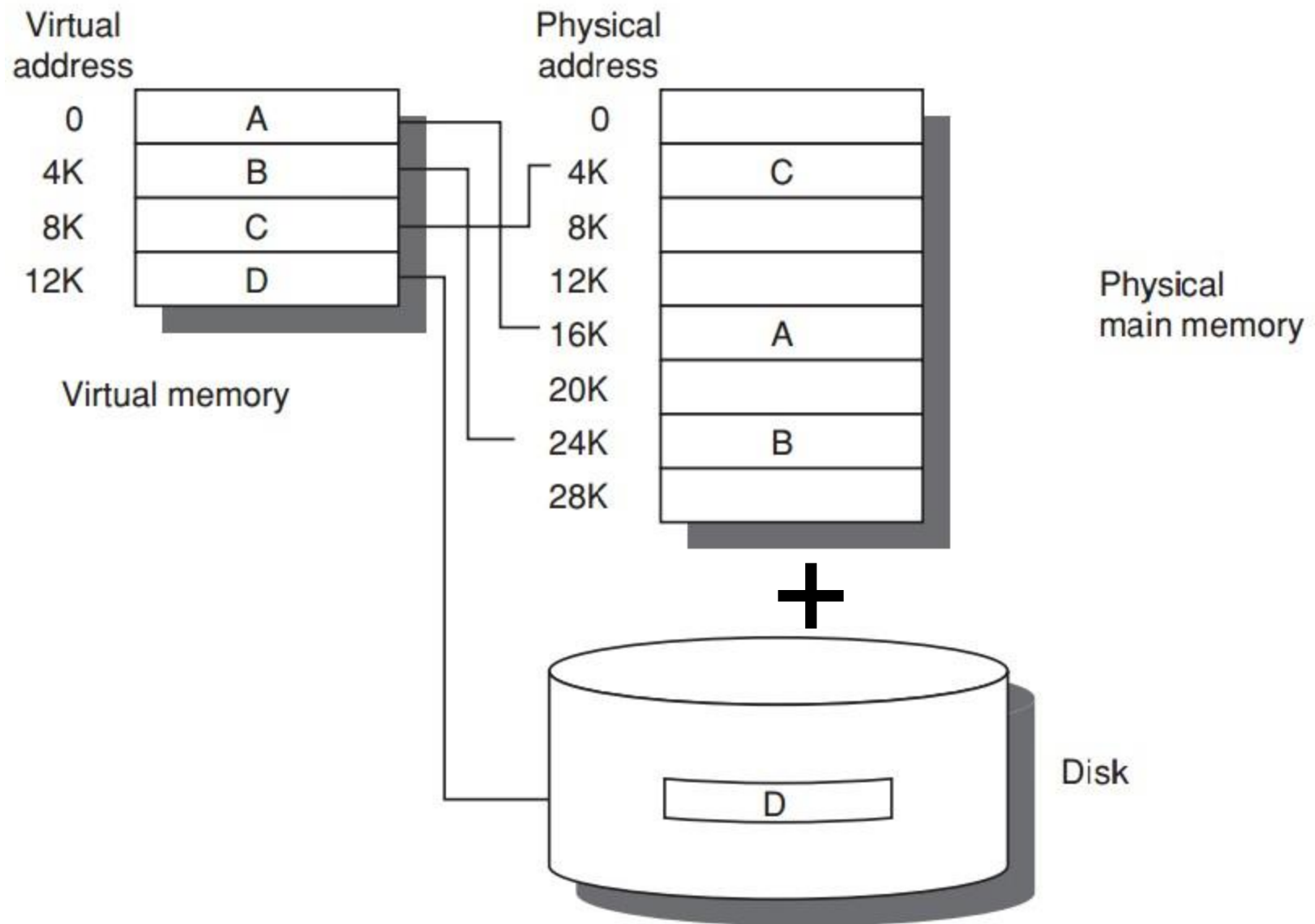


# Memory Hierarchy

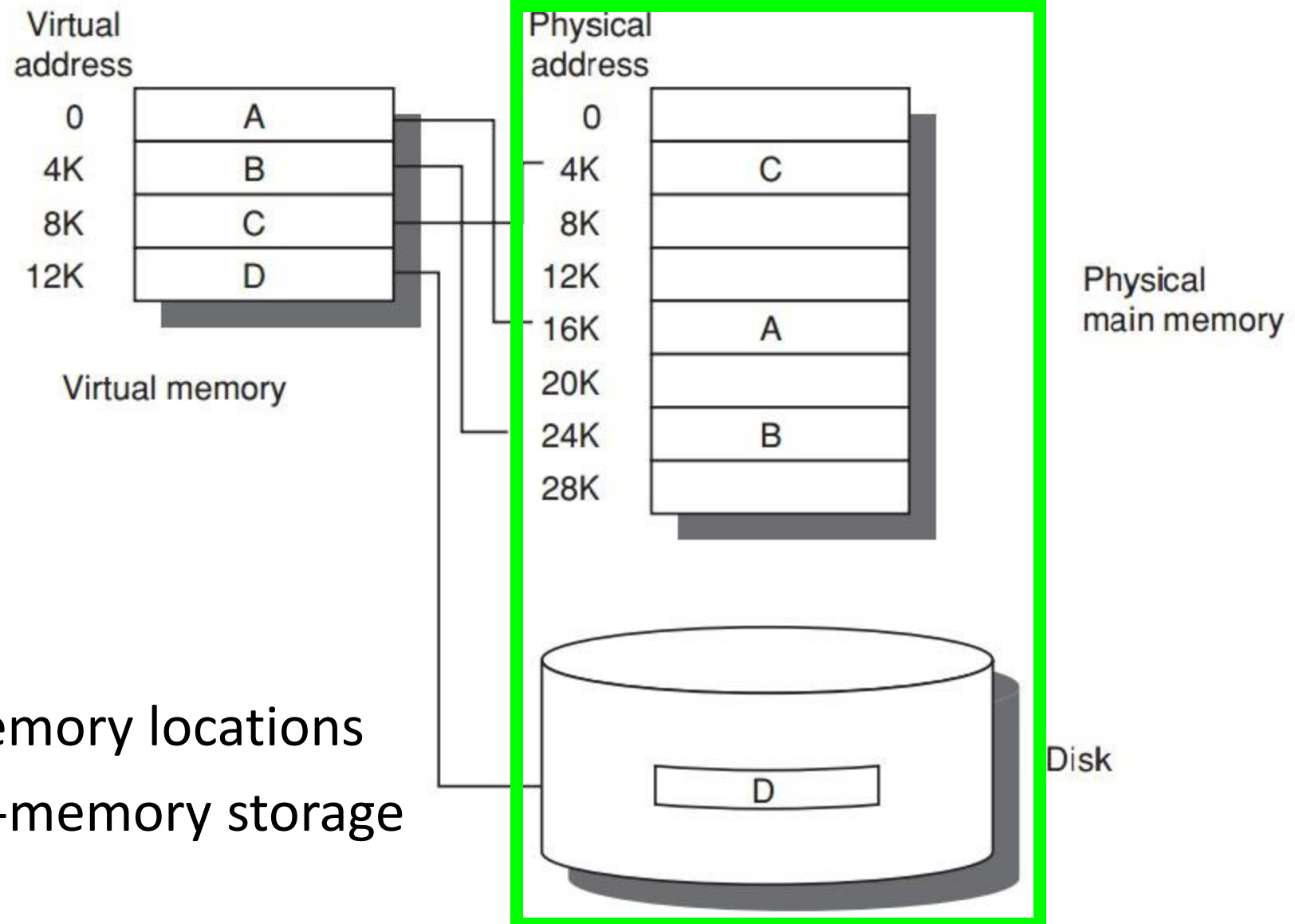


***Larger memory for more processes?***

# Virtual Memory



# Virtual Memory



Program **uses**

- discontinuous memory locations
- + secondary/non-memory storage

# How memory hierarchy works?

Questions:

- Q1. *Where can a block be placed in main memory?*
- Q2. *How is a block found if it is in main memory?*
- Q3. *Which block should be replaced on a virtual memory miss?*
- Q4. *What happens on a write?*

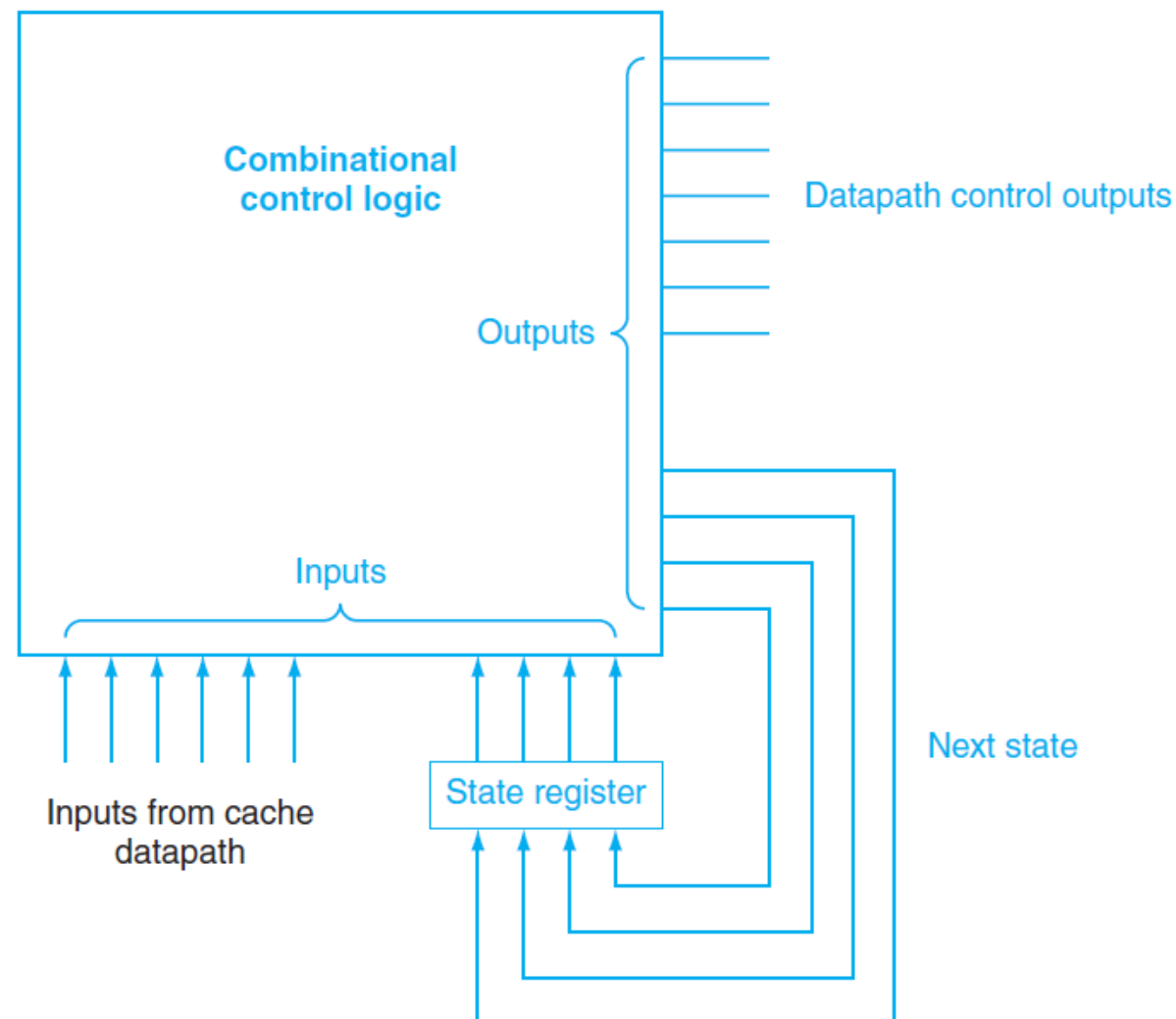
## Using a Finite-State Machine to Control a Simple Cache

### finite-state machine

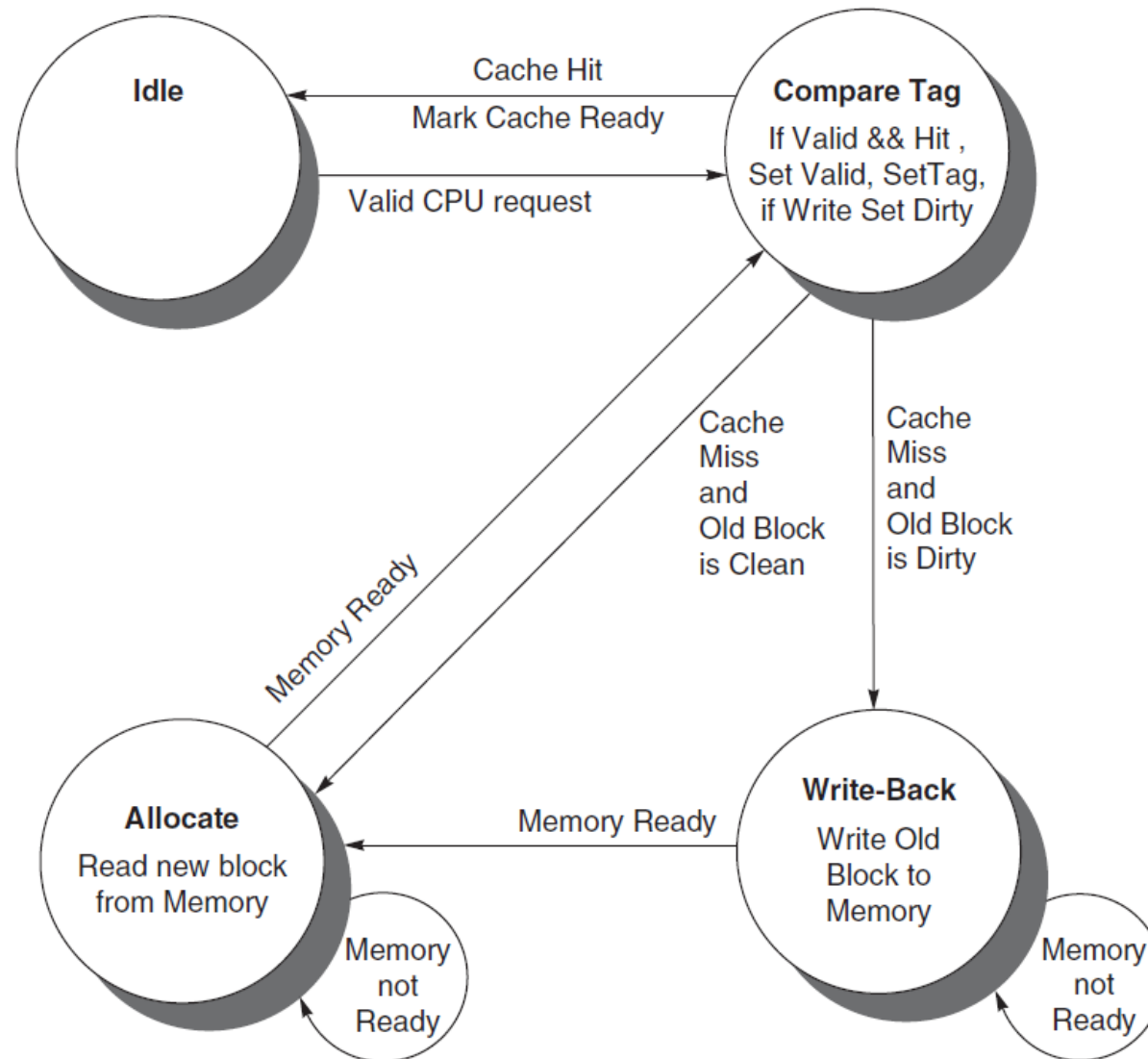
A sequential logic function consisting of a set of inputs and outputs, next-state function that maps the current state and the inputs to a new state, and an output function that maps the current state and possibly the inputs to a set of asserted outputs.

### next-state function

A combinational function that, given the inputs and the current state, determines the next state of a finite-state machine.



## Using a Finite-State Machine to Control a Simple Cache



# Memory System Performance

CPU Execution time

- CPU Execution time = (CPU clock cycles + Memory stall cycles) × Clock cycle time

Memory stall cycles = IC × MemAccess refs per instructions × Miss rate × Miss penalty

$$CPUtime = IC \times \left( CPI_{Execution} + \frac{MemAccess}{Inst} \times MissRate \times MissPenalty \right) \times CycleTime$$

$$CPUtime = IC \times \left( CPI_{Execution} + \frac{MemMisses}{Inst} \times MissPenalty \right) \times CycleTime$$

- CPI Execution includes ALU and Memory instructions



# Average Memory Access Time

- Average Memory Access Time

$$\begin{aligned}
 \text{Average Memory Access Time} &= \frac{\text{Whole accesses time}}{\text{All memory accesses in program}} \\
 &= \frac{\text{Accesses time on hitting} + \text{Accesses time on miss}}{\text{All memory accesses in program}} \\
 &= \text{Hit time} + (\text{Miss Rate} \times \text{Miss Penalty}) \\
 &= (\text{HitTime}_{Inst} + \text{MissRate}_{Inst} \times \text{MissPenalty}_{Inst}) \times \text{Inst}\% \\
 &\quad + (\text{HitTime}_{Data} + \text{MissRate}_{Data} \times \text{MissPenalty}_{Data}) \times \text{Data}\%
 \end{aligned}$$

$$\text{CPUtime} = IC \times \left( \frac{\text{AluOps}}{\text{Inst}} \times \text{CPI}_{\text{AluOps}} + \frac{\text{MemAccess}}{\text{Inst}} \times \text{AMAT} \right) \times \text{CycleTime}$$





# Example1: Impact on Performance

- Suppose a processor executes at
  - Clock Rate = 200 MHz (5 ns per cycle), Ideal (no misses) CPI = 1.1
  - 50% arith/logic, 30% ld/st, 20% control
- Suppose that 10% of memory operations get 50 cycle miss penalty
- Suppose that 1% of instructions get same miss penalty
- What is the CPUtime and the AMAT ?

• Answer: CPI = ideal CPI + average stalls per instruction =  $1.1(\text{cycles/ins}) + [0.30(\text{DataMops/ins}) \times 0.10(\text{miss/DataMop}) \times 50(\text{cycle/miss})] + [1(\text{InstMop/ins}) \times 0.01(\text{miss/InstMop}) \times 50(\text{cycle/miss})] = (1.1 + 1.5 + .5) \text{ cycle/ins} = 3.1$

• AMAT =  $(1/1.3) \times [1 + 0.01 \times 50] + (0.3/1.3) \times [1 + 0.1 \times 50] = 2.54$



## Example2: Impact on Performance

**Assume :** Ideal CPI=1 (no misses)

- L/S's structure . 50% of instructions are data accesses
- Miss penalty is 25 clock cycles
- Miss rate is 2%
- How faster would the computer be if all instructions were cache hits?

**Answer:** first compute the performance for always hits:

$$\begin{aligned}
 \text{CPU}_{\text{execution time}} &= (\text{CPU clock cycles} + \text{memory stall cycles}) \times \text{clock cycle} \\
 &= (IC \times \text{CPI} + 0) \times \text{Clock cycle} \\
 &= IC \times 1.0 \times \text{clock cycle}
 \end{aligned}$$

- Now for the computer with the real cache, first compute memory stall cycles:

$$\begin{aligned}
 \text{Memory stall cycles} &= IC \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Missrate} \times \text{Miss penalty} \\
 &= IC \times (1 + 0.5) \times 0.02 \times 25 = IC \times 0.75
 \end{aligned}$$



## Example2: Impact on Performance

**Assume:** Ideal CPI=1 (no misses)

- L/S's structure . 50% of instructions are data accesses
- Miss penalty is 25 clock cycles
- Miss rate is 2%
- How faster would the computer be if all instructions were cache hits?

The total performance is thus:

$$\begin{aligned}\text{CPU execution time cache} &= (\text{IC} \times 1.0 + \text{IC} \times 0.75) \times \text{Clock cycle} \\ &= 1.75 \times \text{IC} \times \text{Clock cycle}\end{aligned}$$



## Example2: Impact on Performance

Assume: Ideal CPI=1 (no misses)

- L/S's structure . 50% of instructions are data accesses
- Miss penalty is 25 clock cycles
- Miss rate is 2%
- How faster would the computer be if all instructions were cache hits?

The performance ratio is the inverse of the execution times

$$\frac{\text{CPU execution time}_{\text{cache}}}{\text{CPU execution time}} = \frac{1.75 \times \text{IC} \times \text{Clock cycle}}{1.0 \times \text{IC} \times \text{clock cycle}} = 1.75$$

The computer with no cache misses is 1.75 time faster.



## Example3: Impact on Performance

**Assume:** CPI=2(perfect cache)      clock cycle time = 1.0 ns

- MPI(memory reference per instruction) = 1.5
- Size of both caches is 64K and size of both block is 64 bytes
- One cache is direct mapped and other is two-way set associative. the former has miss rate of 1.4%, the latter has miss rate 1.0%
- The selection multiplexor forces CPU clock cycle time to be stretched 1.25 times
- Miss penalty is 75ns, and hit time is 1 clock cycle
- What is the impact of two cache organizations on performance of CPU (first, calculate the average memory access time and then CPU performance)?

**Answer :**

Average memory access time is

- Average memory access time = Hit time + Miss rate  $\times$  miss penalty

Thus, the time for each organization is

- Average memory access time 1-way =  $1.0 + (0.014 \times 75) = 2.05$  ns
- Average memory access time 2-way =  $1.0 \times 1.25 + (0.01 \times 75) = 2.00$  ns



## Example3: Impact on Performance

The average memory access time is better for the 2-way set-associative cache.

CPU performance is

$$\begin{aligned}
 CPUtime &= IC \times \left( CPI_{execution} + \frac{Misses}{Instruction} \times Miss\ penalty \right) \times Clock\ cycle\ time \\
 &= IC \times \left[ \left( CPI_{execution} \times Clock\ cycle\ time \right) \right. \\
 &\quad \left. + \left( Miss\ rate \times \frac{Memory\ accesses}{Instruction} \times Miss\ penalty \times Clock\ cycle\ time \right) \right]
 \end{aligned}$$

Substituting 75 ns for (miss penalty  $\times$  Clock cycle time), the performance of each cache organization is

$$CPU\ time_{1-way} = IC \times (2 \times 1.0 + (1.5 \times 0.014 \times 75)) = 3.58 \times IC$$

$$CPU\ time_{2-way} = IC \times (2 \times 1.0 \times 1.25 + (1.5 \times 0.010 \times 75)) = 3.63 \times IC$$



## Example3: Impact on Performance

Relative performance is

$$\frac{CPU\ time_{2-way}}{CPU\ time_{1-way}} = \frac{3.63 \times Instruction\ count}{3.58 \times Instruction\ count} = \frac{3.63}{3.58} = 1.01$$

In contrast to the results of average memory access time, the direct-mapped leads to slightly better average performance. Since CPU time is our bottom-line evaluation.



# How to Improve

Hence, there are more than 20 cache optimizations into these categories:

$$AMAT = HitTime + MissRate \times MissPenalty$$

1. Reduce the miss penalty
  - multilevel caches, critical word first, read miss before write miss, merging write buffers, and victim caches
2. Reduce the miss rate
  - larger block size, large cache size, higher associativity, way prediction and pseudo-associativity, and compiler optimizations
3. Reduce the time to hit in the cache
  - small and simple caches, avoiding address translation, pipelined cache access, and trace caches<sup>3</sup>.
4. Reduce the miss penalty and miss rate via parallelism
  - non-blocking caches, hardware prefetching, and compiler prefetching





# Summary of Cache Optimization Technology

Technique	Hit time	Miss penalty	Miss rate	Hardware complexity	Comment
Larger block size		–	+	0	Trivial; Pentium 4L2 uses 128 bytes
Larger cache size	–		+	1	Widely used, especially for L2 caches
Higher associativity	–		+	1	Widely used
Multilevel caches		+		2	Costly hardware; harder if L1 block size $\neq$ L2 block size; widely used
Read priority over writes		+		1	Widely used
Avoiding address translation during cache indexing	+			1	Widely used

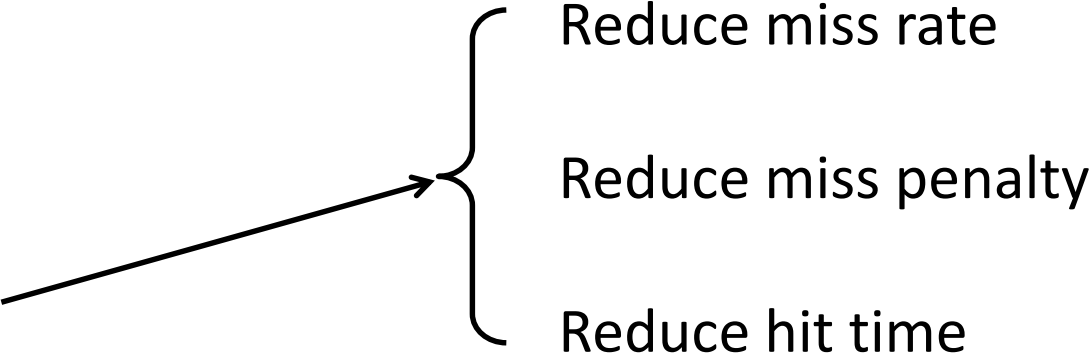


# Summary of Cache Optimization Technology

Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined & banked caches	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Critical word first and early restart			+			2	Widely used
Merging write buffer			+			1	Widely used with write through
Compiler techniques to reduce cache misses				+		0	Software is a challenge, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	–	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs
HBM as additional level of cache		+/-	–	+	+	3	Depends on new packaging technology. Effects depend heavily on hit rate improvements



# Summary

- Memory hierarchy
    - From single level to multi level
    - Evaluate the performance parameters of the storage system (average price per bit  $C$ ; hit rate  $H$ ; average memory access time  $T$ )
  - Cache basic knowledge
    - Mapping rules
    - Access method
    - Replacement algorithm
    - Write strategy
    - Cache performance analysis
- 
- Virtual Memory (the influence of memory organization structure on Cache failure rate)

