Computer Architecture

Spring 2016

Lecture 02: Introduction II

Shuai Wang

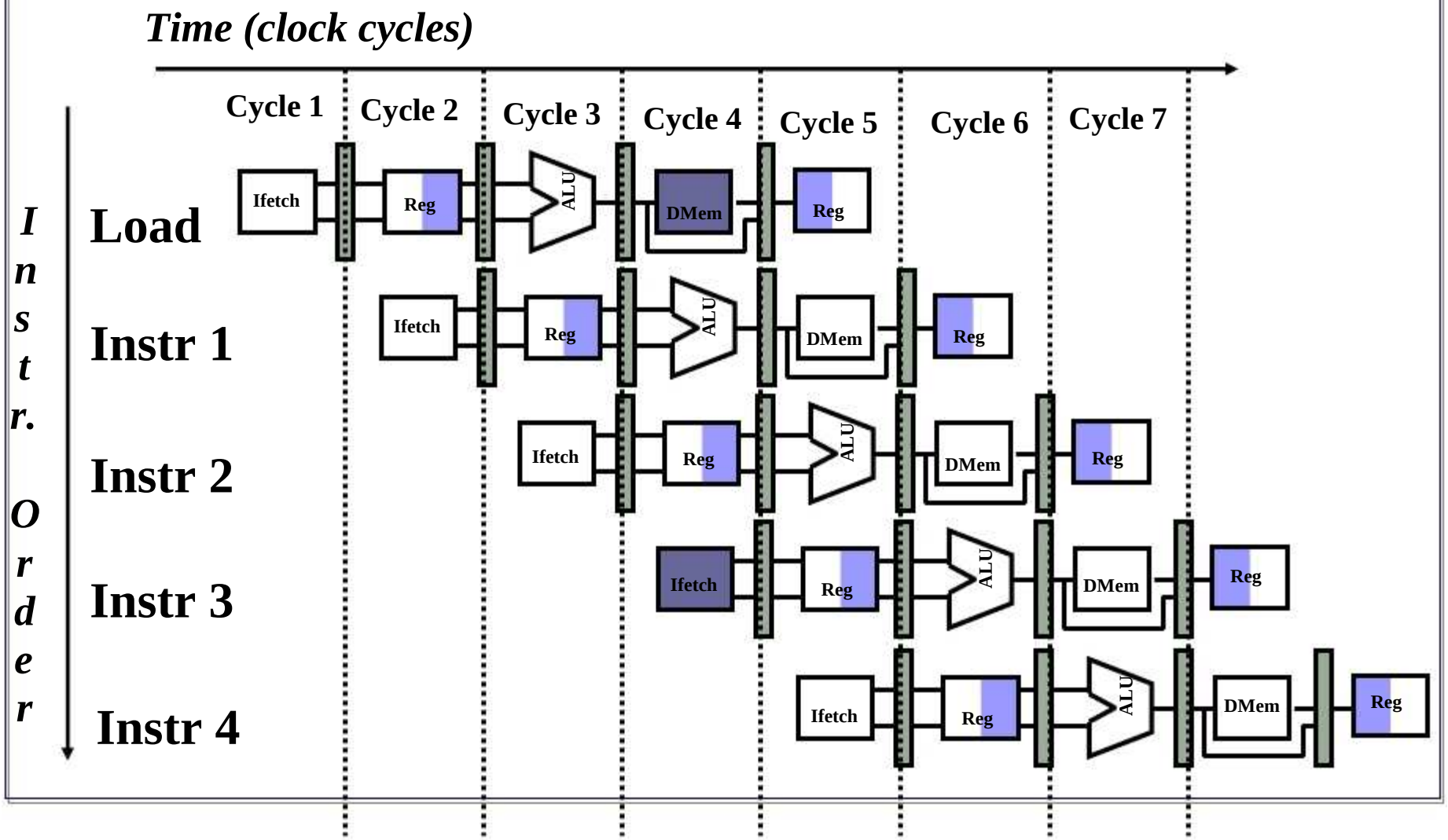
Department of Computer Science and Technology

Nanjing University

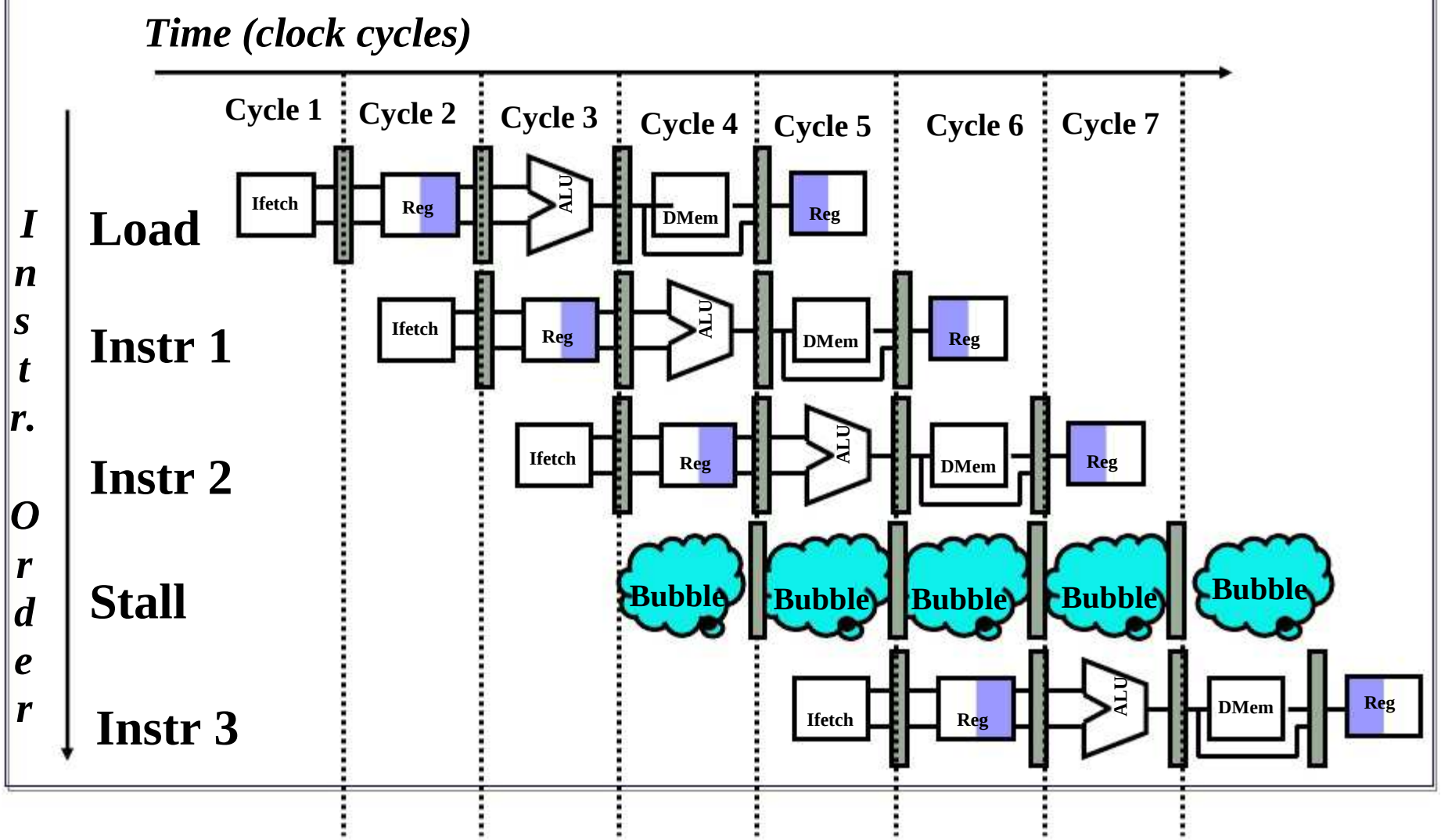
Pipeline Hazards

- Major hurdle to pipelining: **hazards** prevent the next instruction from executing during its designated clock cycle
  - **Structural hazards:** hardware cannot support all possible combinations of instructions simultaneously
  - **Data hazards:** instruction depends on result of a previous instruction still in the pipeline
  - **Control hazards:** caused by delay between the fetching of instructions and decisions about changes in control flow

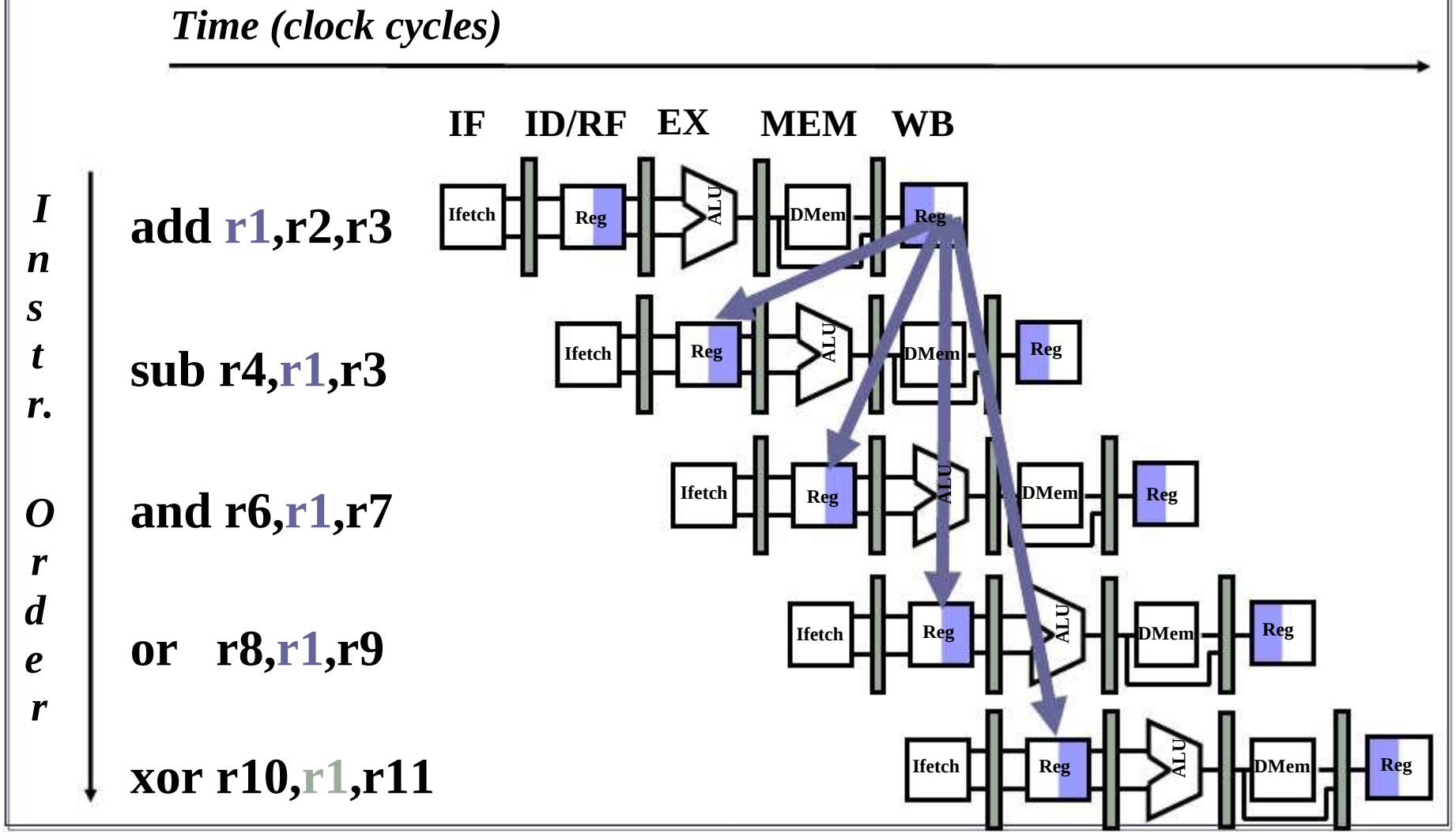
One Memory Port/Structural Hazards



One Memory Port/Structural Hazards



Data Hazard on R1



Three Generic Data Hazards

- Read After Write (RAW)

  $Instr_J$ tries to read operand before $Instr_I$ writes it

```
I:   add  r1,r2,r3
J:   sub  r4,r1,r3
```

- Caused by a "Dependence".
  This hazard results from an actual need for communication.

Three Generic Data Hazards

- Write After Read (WAR)

  Instr$_J$ tries to write operand *before* Instr$_I$ reads it

I: sub r4,r1,r3
J: add r1,r2,r3
K: mul r6,r1,r7

- Called an "anti-dependence" by compiler writers. This results from reuse of the name "r1".
- Can't happen in MIPS 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Reads are always in stage 2, and
  - Writes are always in stage 5

Three Generic Data Hazards

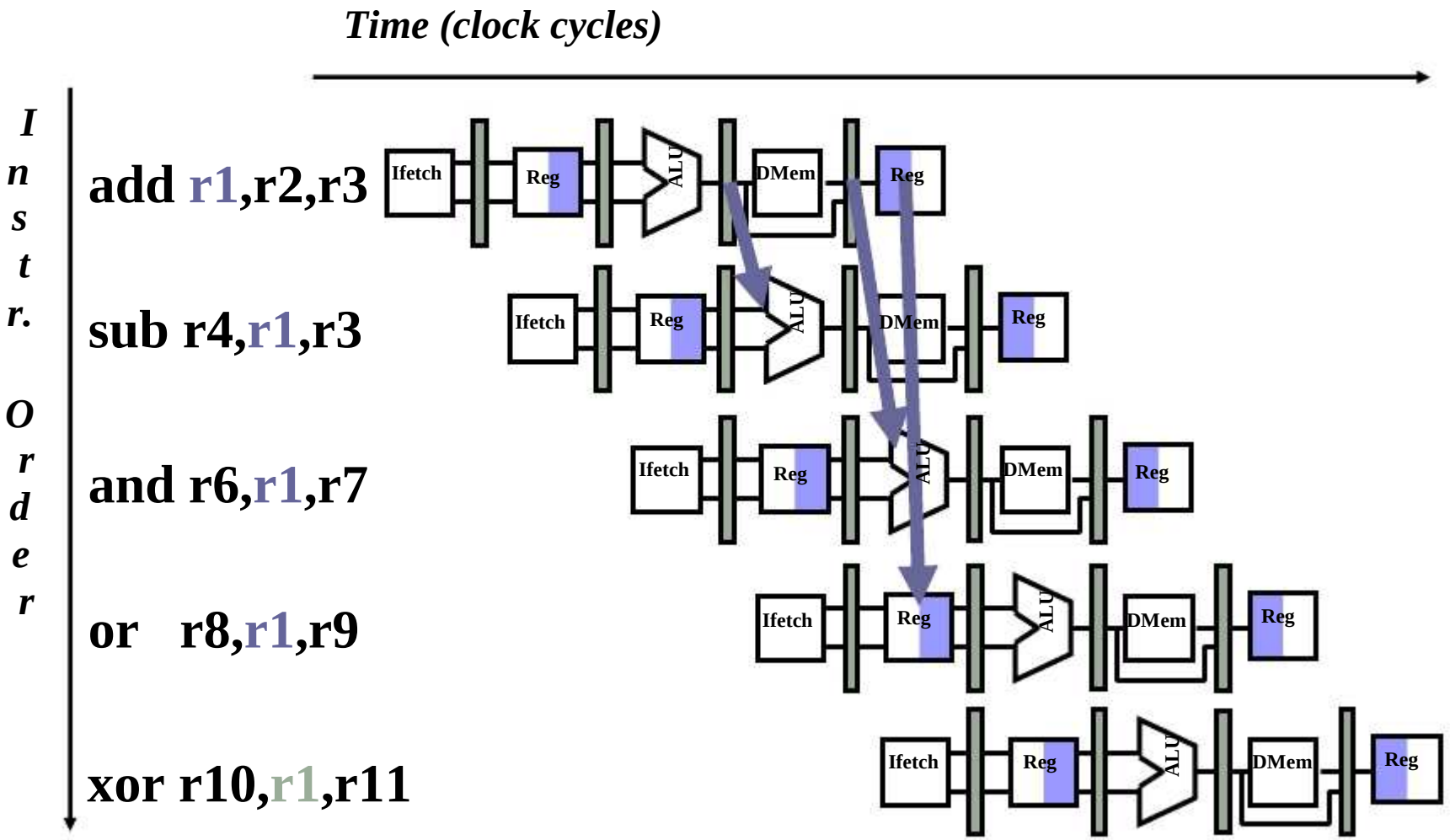
- Write After Write (WAW)

  Instr$_J$ writes operand *before* Instr$_I$ writes it.

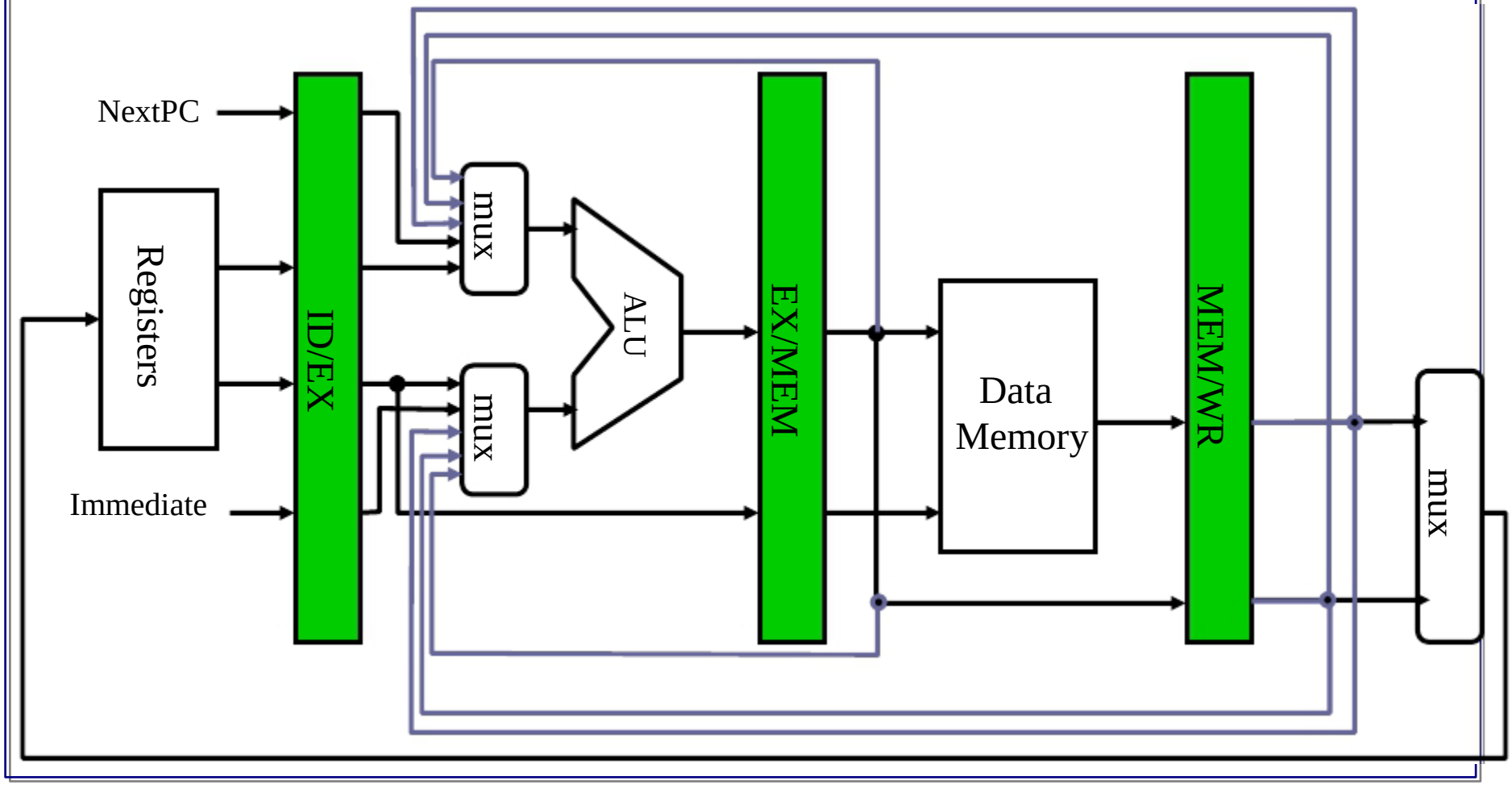
I: sub r1,r4,r3
J: add r1,r2,r3
K: mul r6,r1,r7

- Called an "output dependence" by compiler writers
  This also results from the reuse of name "r1".
- Can't happen in MIPS 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Writes are always in stage 5
- Will see WAR and WAW in later more complicated pipes

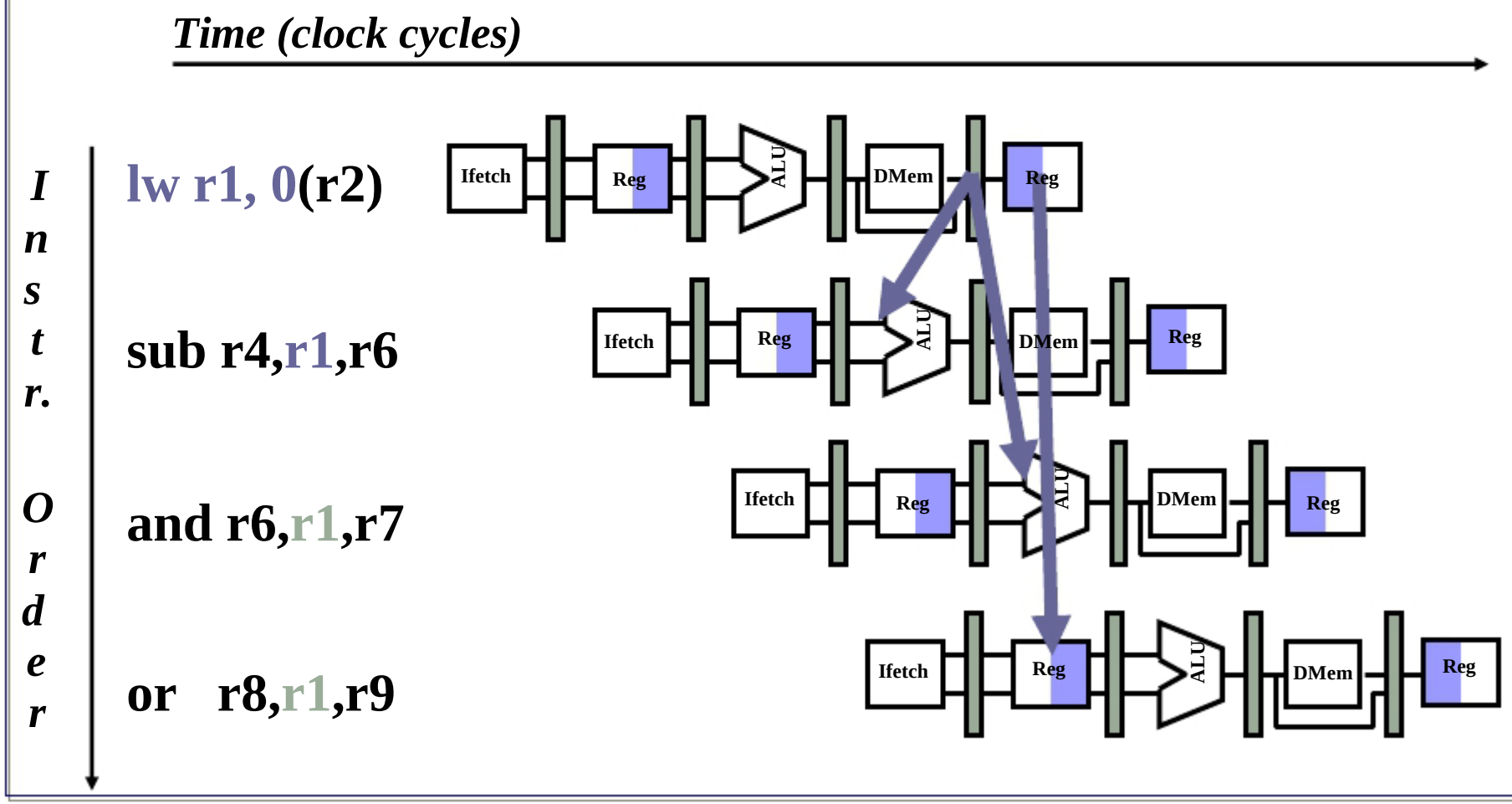
Forwarding to Avoid Data Hazard



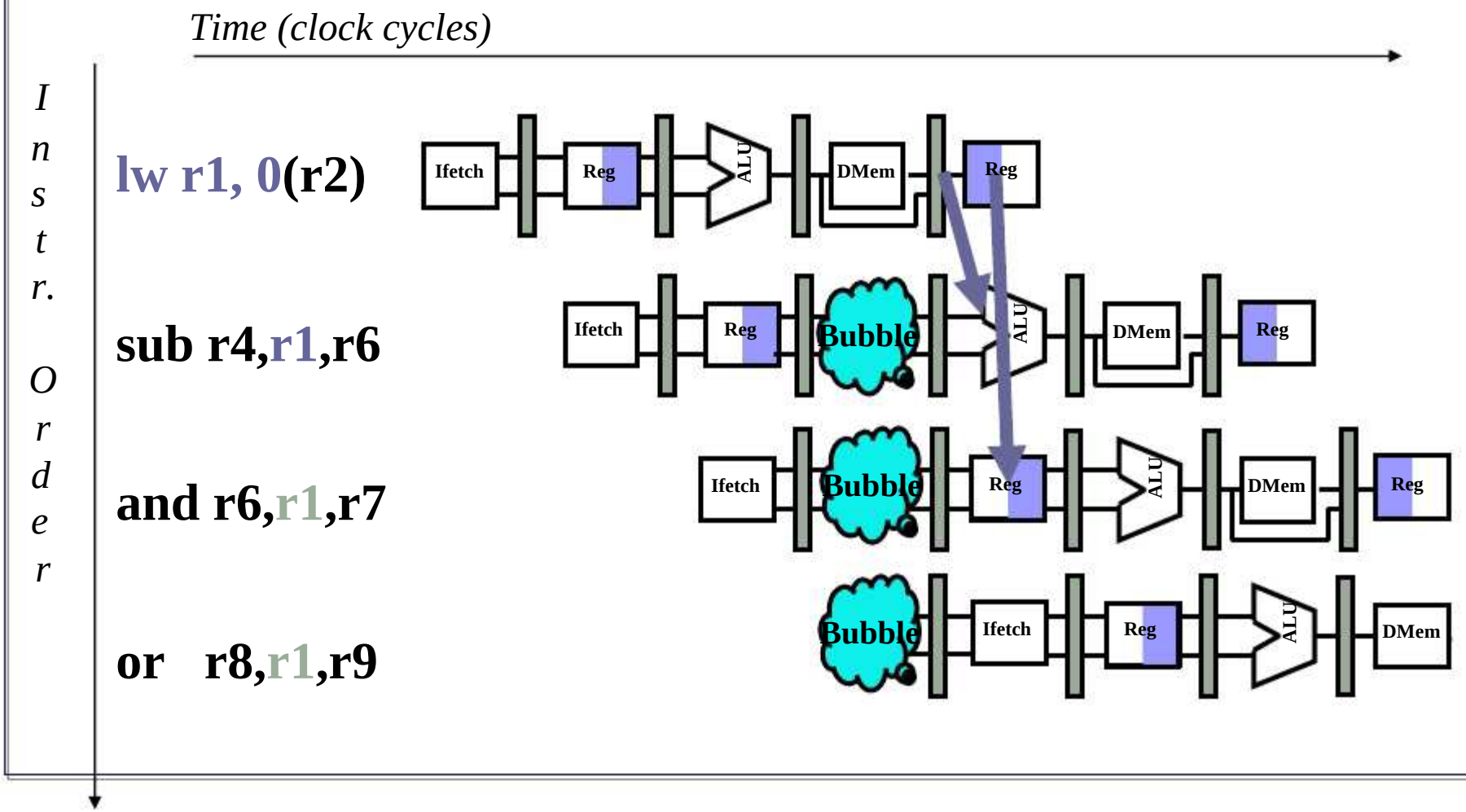
HW Change for Forwarding



Data Hazard Even with Forwarding



Data Hazard Even with Forwarding



Control Hazard on Branches: 3 Stage Stall

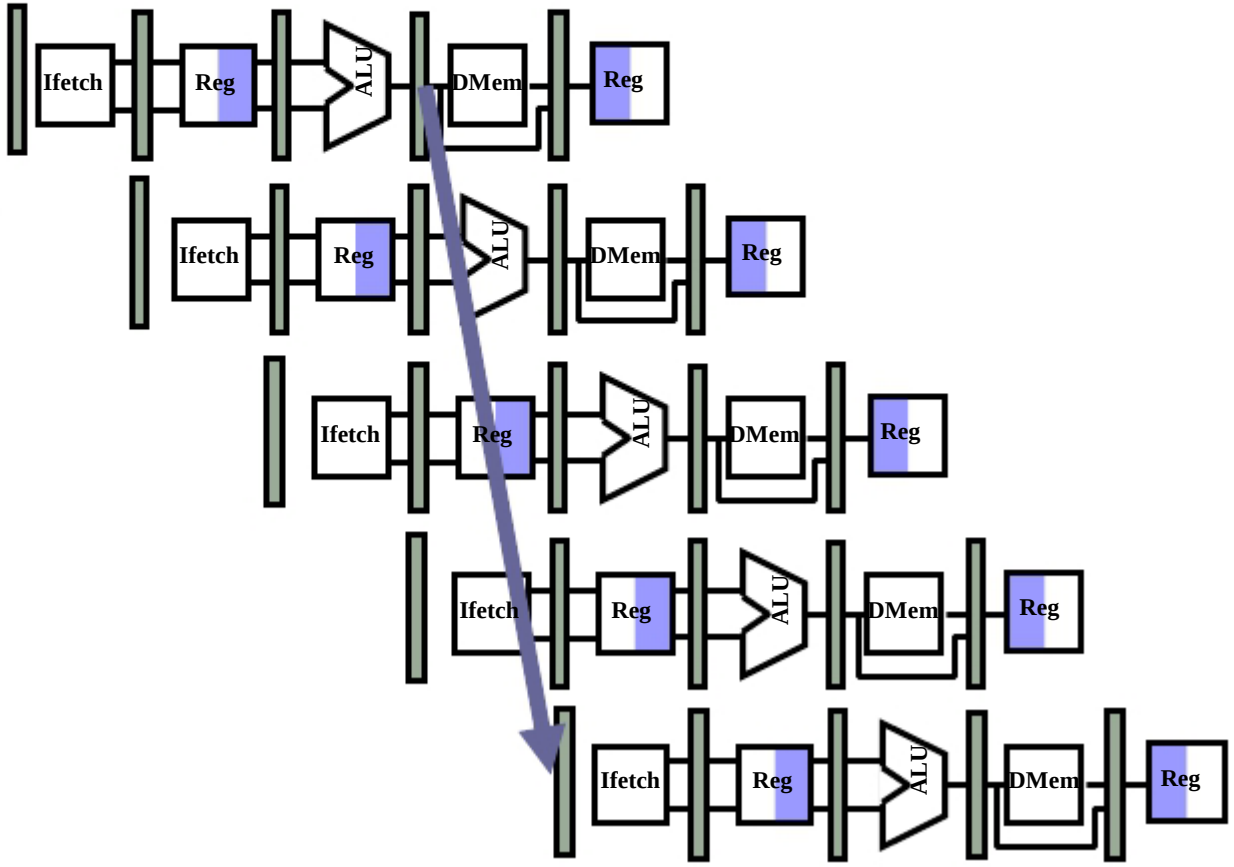
10: beq r1,r3,36

14: and r2,r3,r5

18: or r6,r1,r7

22: add r8,r1,r9

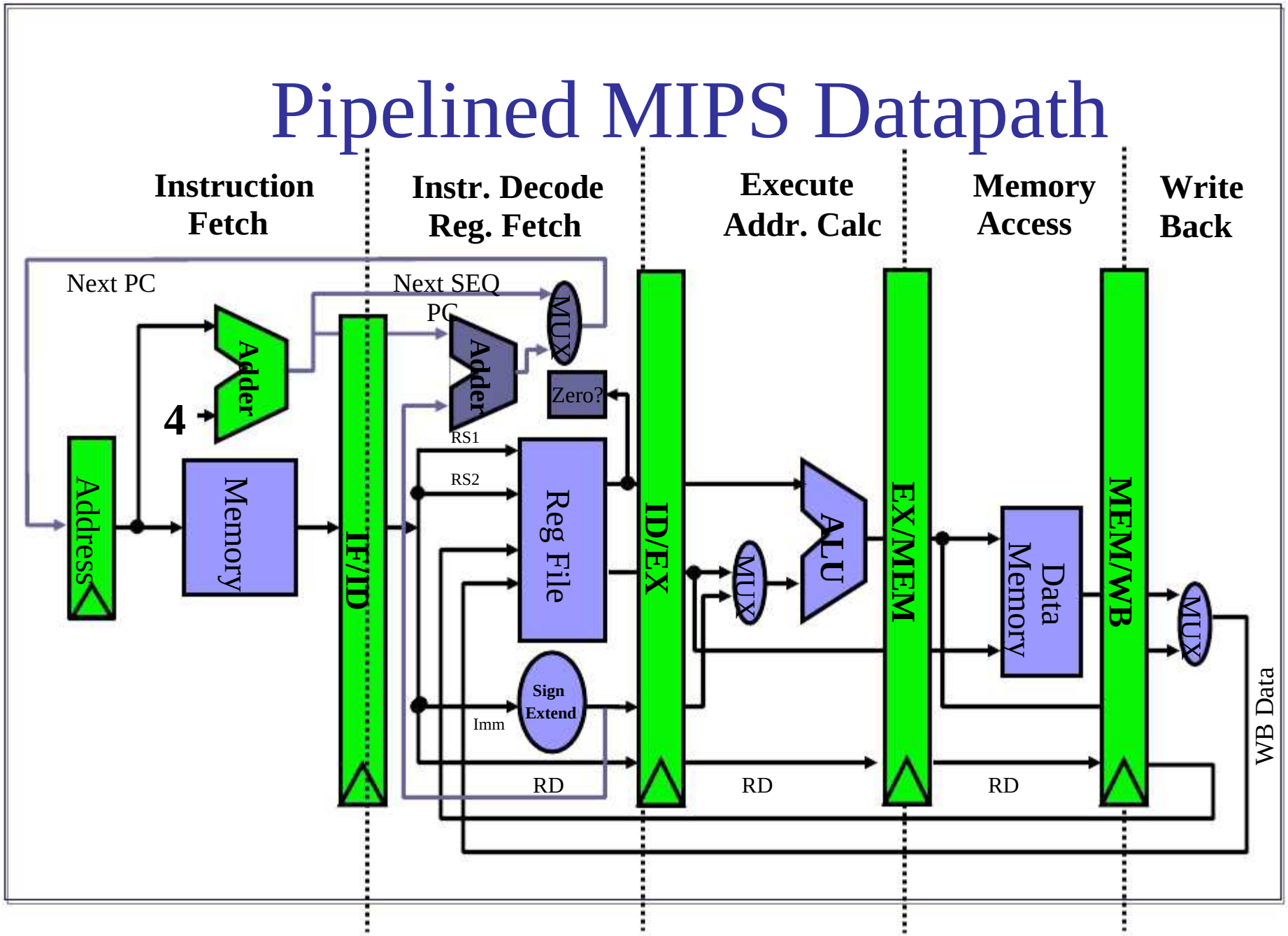
36: xor r10,r1,r11



Example: Branch Stall Impact

- If 30% branch, Stall 3 cycles significant
- Two part solution:
  - Determine branch taken or not sooner, AND
  - Compute taken branch address earlier
- MIPS branch tests if register = 0 or $\neq$ 0
- MIPS Solution:
  - Move Zero test to ID/RF stage
  - Adder to calculate new PC in ID/RF stage
  - 1 clock cycle penalty for branch versus 3

Pipelined MIPS Datapath



Four Branch Hazard Alternatives

#1: Stall until branch direction is clear

#2: Predict Branch Not Taken

- Execute successor instructions in sequence
- “Squash” instructions in pipeline if branch actually taken
- Advantage of late pipeline state update
- 47% MIPS branches not taken on average
- PC+4 already calculated, so use it to get next instruction

#3: Predict Branch Taken

- 53% MIPS branches taken on average
- But haven’t calculated branch target address in MIPS
  - MIPS still incurs 1 cycle branch penalty
  - Other machines: branch target known before outcome

Four Branch Hazard Alternatives

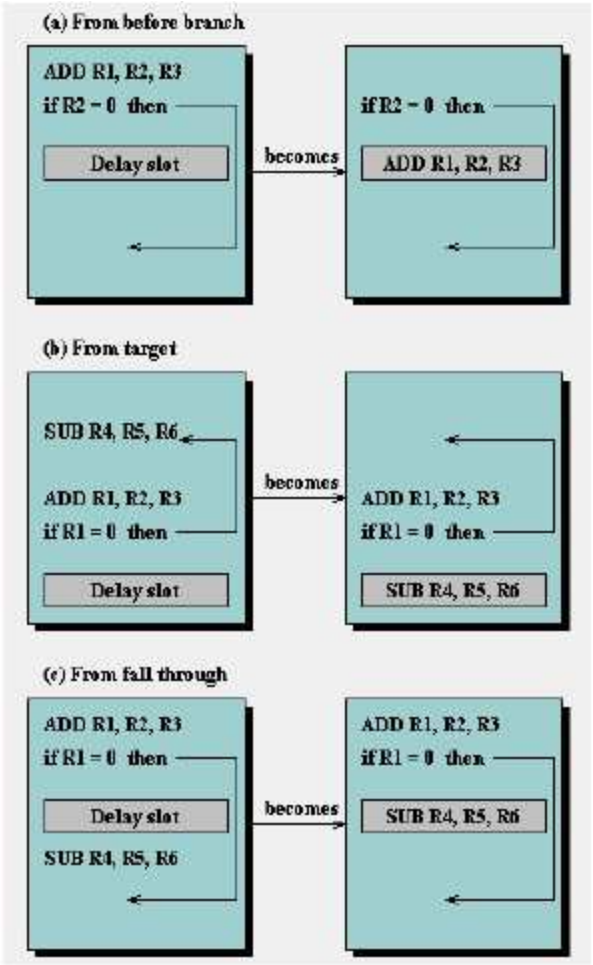
#4: Delayed Branch

- Define branch to take place AFTER a following instruction

```
branch  instruction
sequential  successor1
sequential  successor2
........
sequential  successorn
branch  target  if  taken
```

- 1 slot delay allows proper decision and branch target address in 5 stage pipeline
- MIPS uses this

Scheduling Branch-delay Slot



- Three branch-scheduling schemes:
  - From before branch
  - From target
  - From fall through
- In (a) the delay slot is scheduled with an independent instruction from before the branch. This is the best choice.
- Strategies (b) and (c) are used when (a) is not possible.
- To make this optimization legal for (b) and (c), it must be OK to execute the SUB instruction when the branch goes in the unexpected direction. OK means that the work might be wasted but the program will still execute correctly.

Delayed Branch

- Where to get instructions to fill branch delay slot?
  - Before branch instruction
  - From the target address: only valuable when branch taken
  - From fall through: only valuable when branch not taken
  - Canceling branches allow more slots to be filled
- Compiler effectiveness for single branch delay slot:
  - Fills about 60% of branch delay slots
  - About 80% of instructions executed in branch delay slots useful in computation
  - About 50% (60% x 80%) of slots usefully filled
- Delayed Branch downside: 7-8 stage pipelines, multiple instructions issued per clock (superscalar)

Today's Lecture

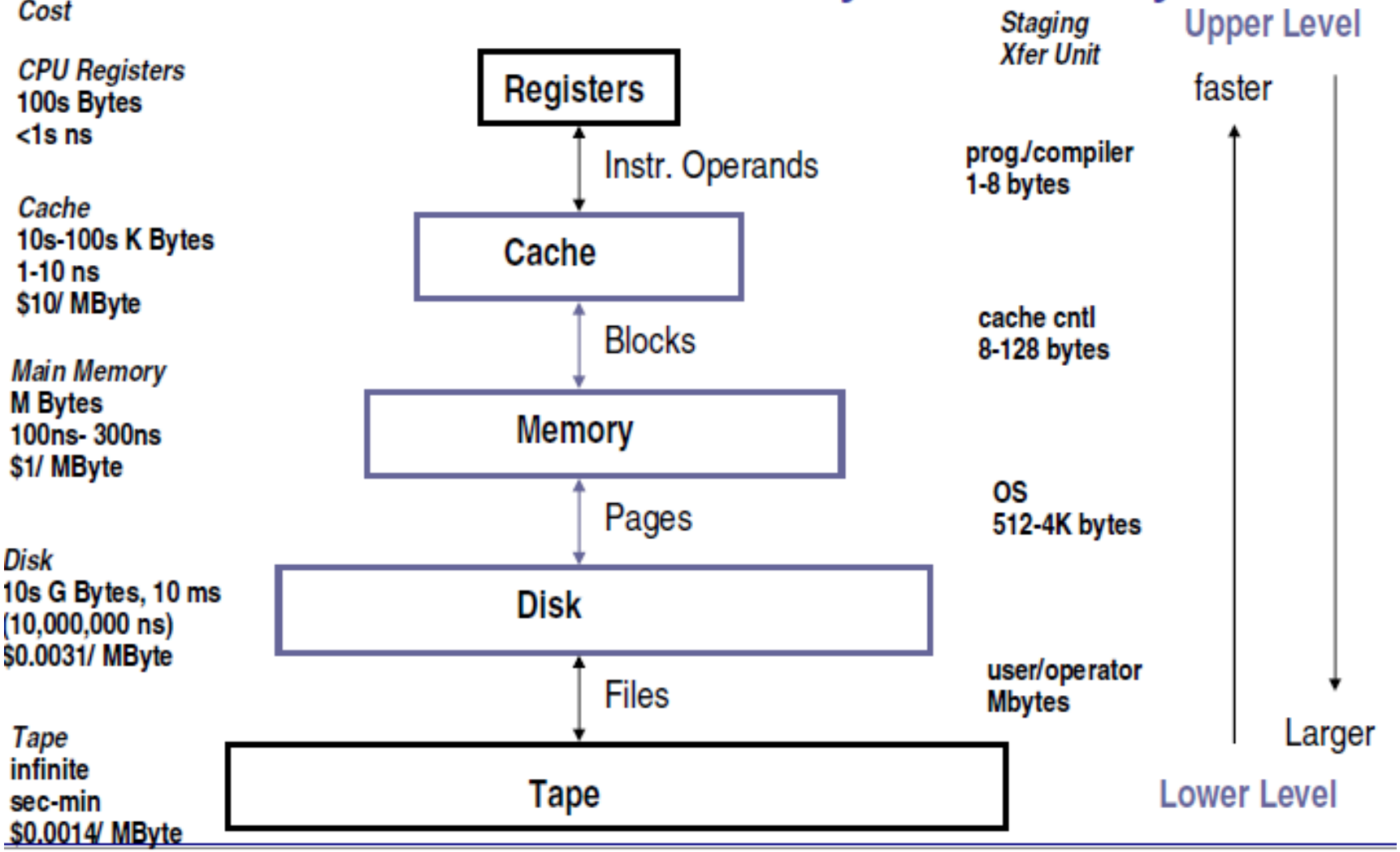
Review of the following topics

- Pipelining
- Caches
- Virtual memory

Fundamentals of computer design

- Performance
- Cost
- Technology

Levels of the Memory Hierarchy



The Principle of Locality

- Principle of locality:
  - Programs tend to reuse data and instructions they have used recently.
  - Program access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)
- Last 15 years, HW (hardware) relied on locality for speed

Memory Hierarchy: Terminology

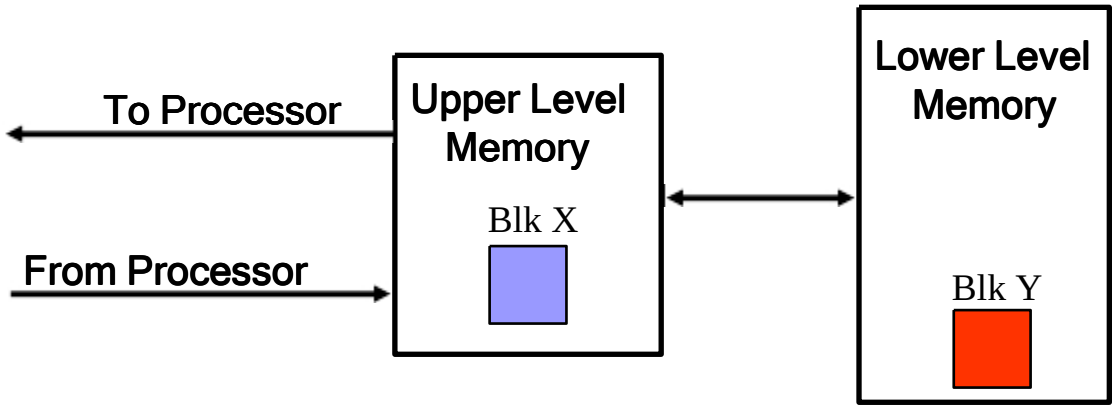
Hit: data appears in some block in the upper level (example: Block X)

- Hit Rate: the fraction of memory access found in the upper level
- Hit Time: Time to access the upper level which consists of
  RAM access time + Time to determine hit/miss

Miss: data needs to be retrieve from a block in the lower level (Block Y)

- Miss Rate = 1 - (Hit Rate)
- Miss Penalty: Time to replace a block in the upper level +
  Time to deliver the block to the processor

Hit Time << Miss Penalty (500 instructions on 21264!)



Measuring Cache Performance

Hit rate: fraction found in that level

- So high that usually talk about *Miss rate*
- Miss rate fallacy: as MIPS to CPU performance,
- Miss rate to average memory access time in memory

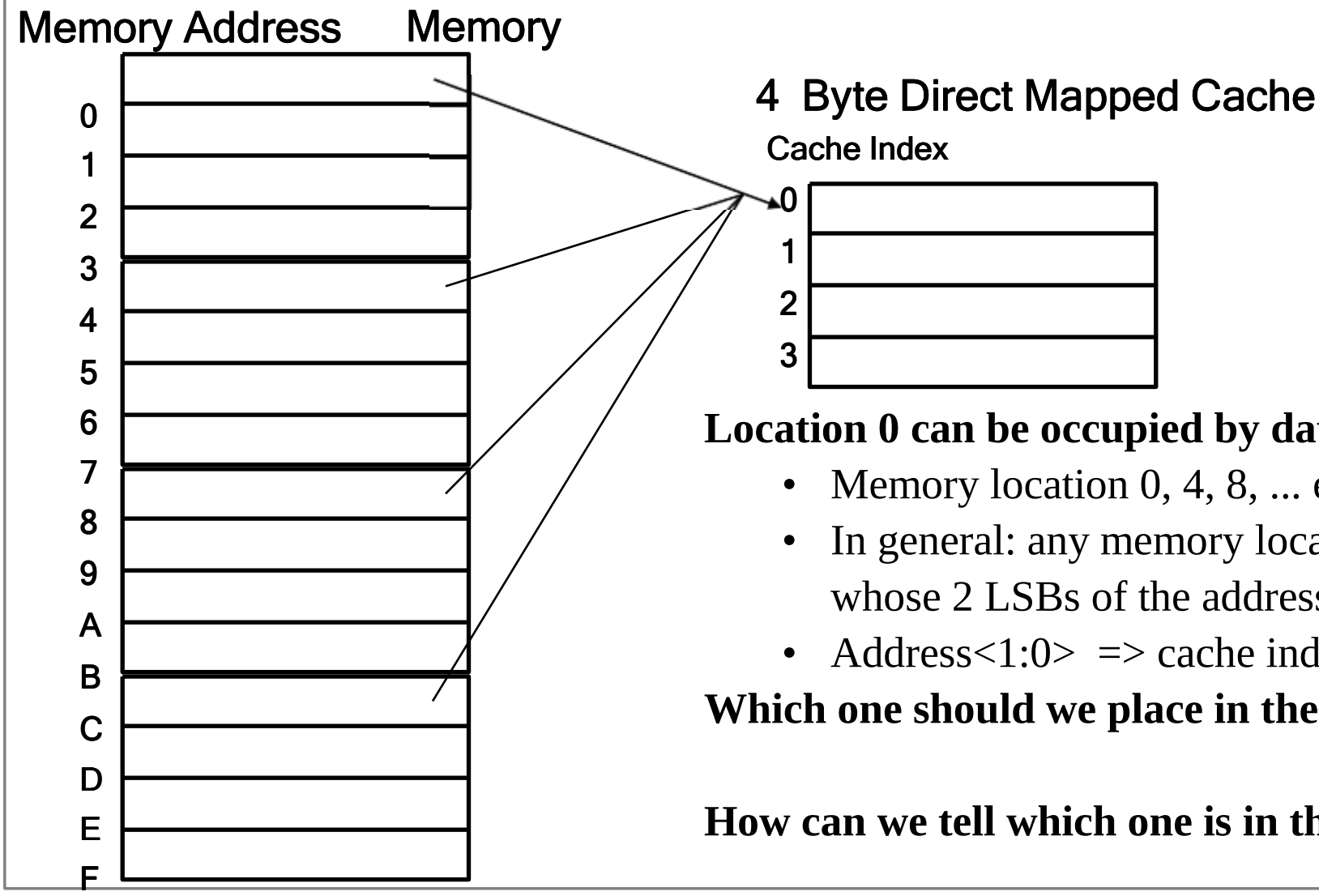
Average memory-access time
= Hit time + Miss rate x Miss penalty (ns or clocks)

Miss penalty: time to replace a block from lower level, including time to replace in CPU

access time: time to lower level = f (latency to lower level)

transfer time: time to transfer block = f (BW between upper & lower levels)

Cache Organization: Direct Mapped



Location 0 can be occupied by data from:

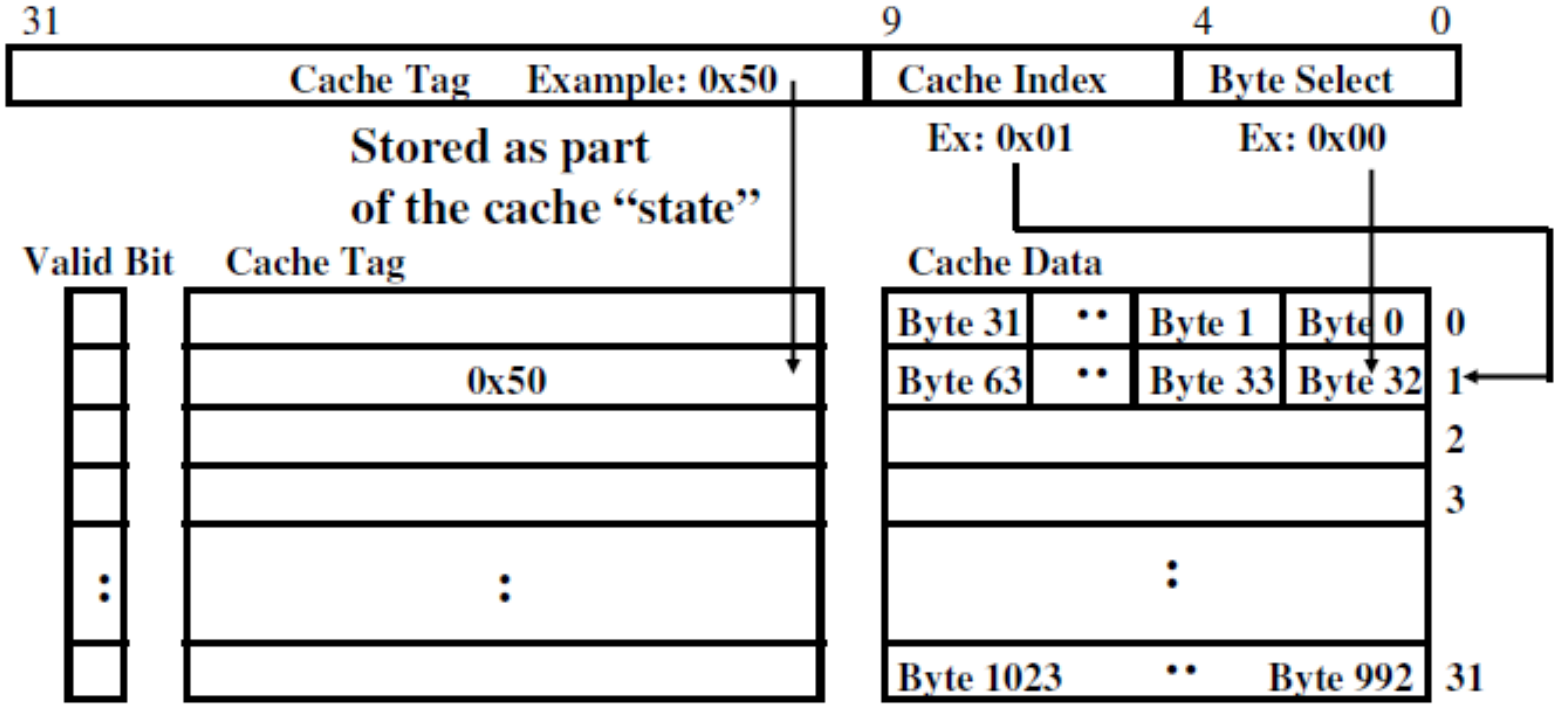
- Memory location 0, 4, 8, ... etc.
- In general: any memory location whose 2 LSBs of the address are 0s
- Address<1:0> => cache index

Which one should we place in the cache?

How can we tell which one is in the cache?

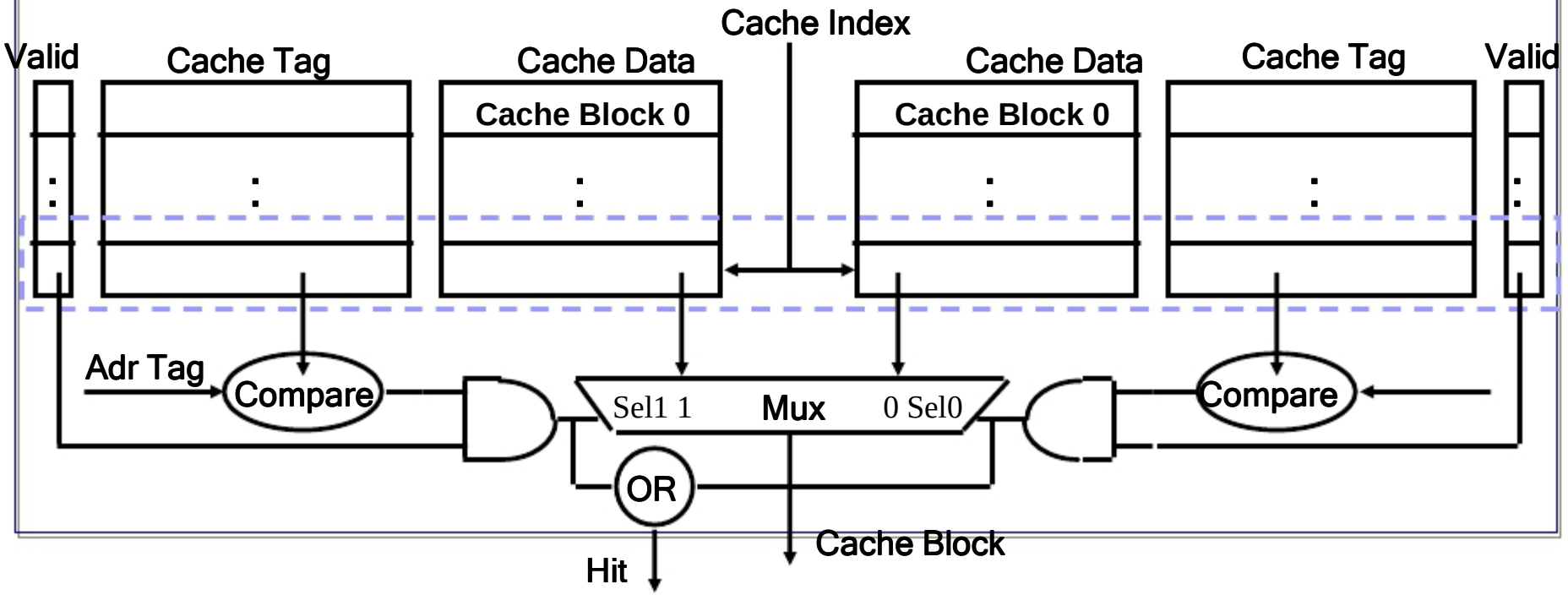
1 KB Direct Mapped Cache, 32B blocks

- For a 2 ** N byte cache:
  - The uppermost (32 - N) bits are always the Cache Tag
  - The lowest M bits are the Byte Select (Block Size = 2 ** M)



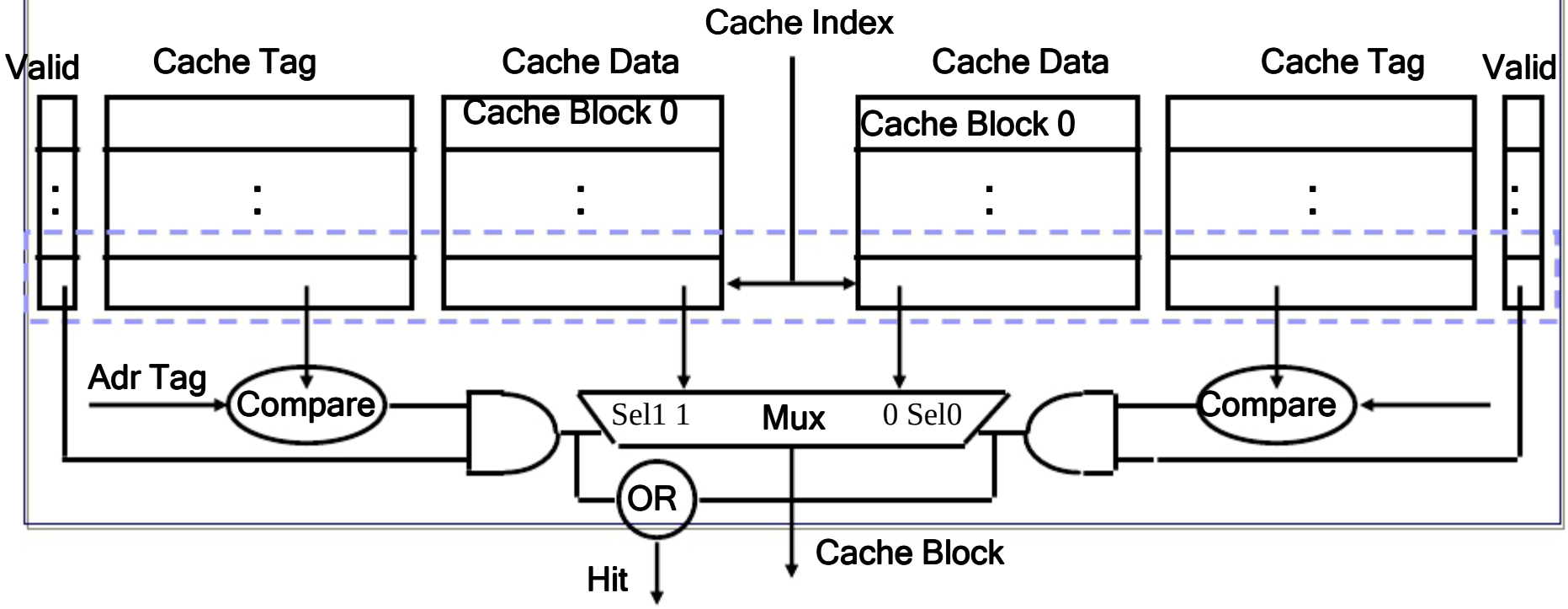
Two-way Set Associative Cache

- N-way set associative: N entries for each Cache Index
  - N direct mapped caches operates in parallel (N typically 2 to 4)
- Example: Two-way set associative cache
  - Cache Index selects a “set” from the cache
  - The two tags in the set are compared in parallel
  - Data is selected based on the tag result



Disadvantage of Set Associative Cache

- N-way Set Associative Cache v. Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.



Four Questions for Memory Hierarchy

Q1: Where can a block be placed in the upper level? *(Block placement)*

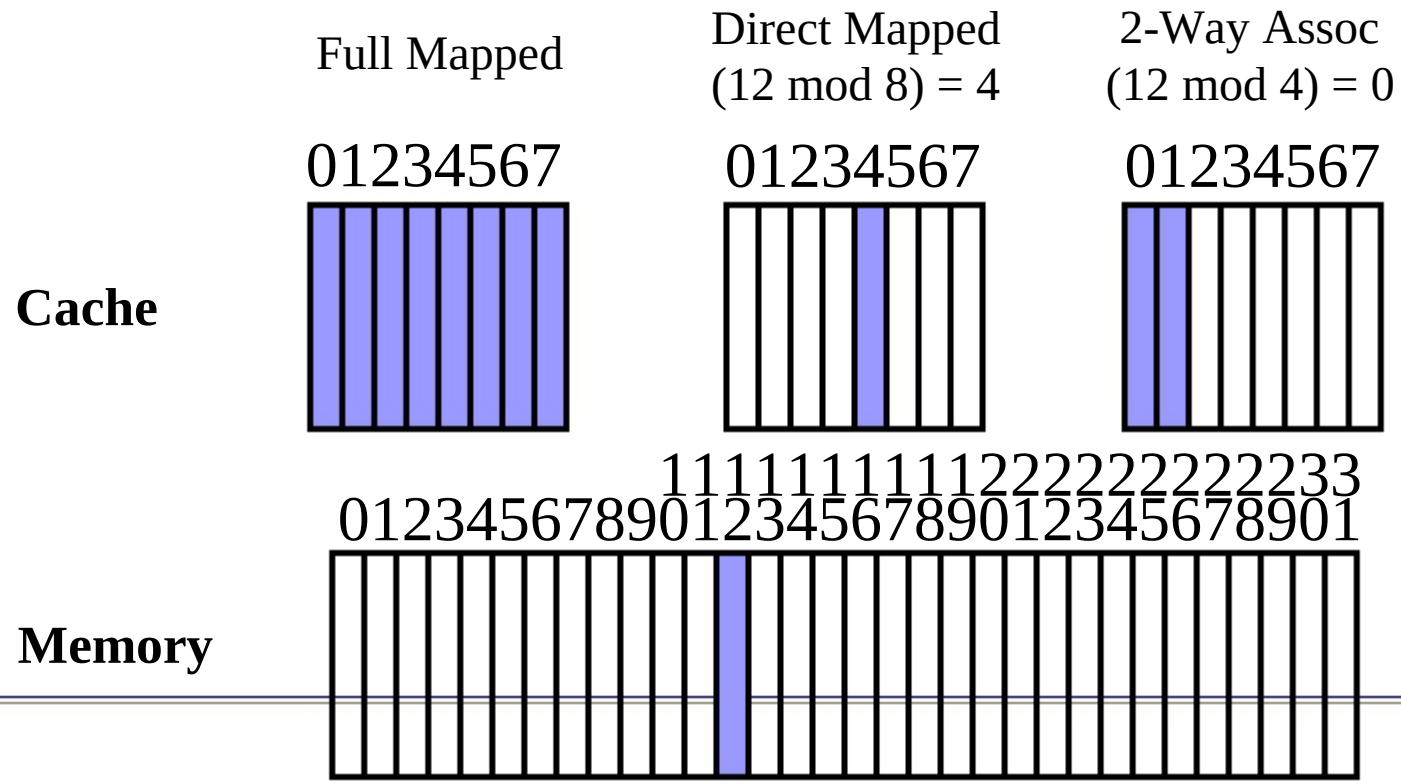
Q2: How is a block found if it is in the upper level? *(Block identification)*

Q3: Which block should be replaced on a miss? *(Block replacement)*

Q4: What happens on a write? *(Write strategy)*

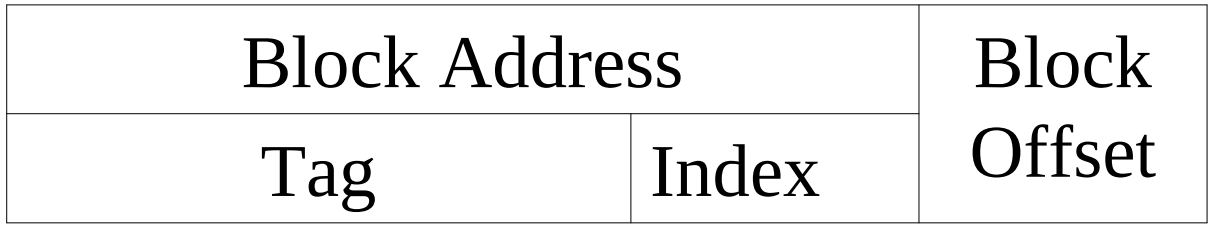
Q1: Where can a block be placed in the upper level?

- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets



Q2: How is a block found if it is in the upper level?

- Tag on each block
  - No need to check index or block offset
- Increasing associativity shrinks index, ⟶ expands tag ⟶



Q3: Which block should be replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

Assoc:	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Q4: What happens on a write?

• *Write through*—The information is written to both the block in the cache and to the block in the lower-level memory.

• *Write back*—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.

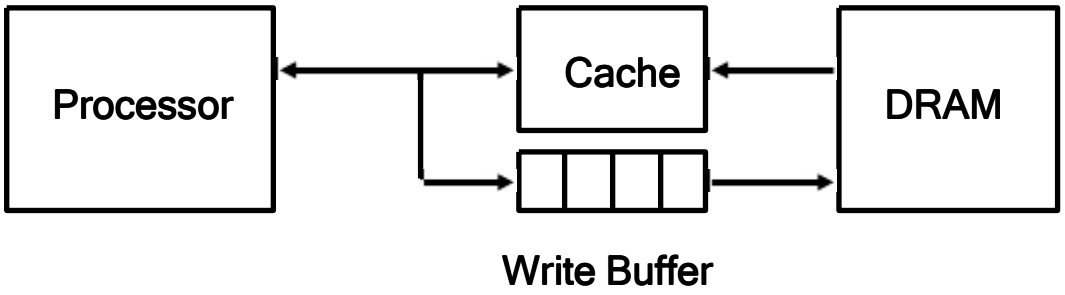
  - Is block clean or dirty?

• Pros and Cons of each?

  - WT: read misses cannot result in writes
  - WB: no repeated writes to same location

• WT always combined with write buffers so that don't wait for lower level memory

Write Buffer for Write Through



- **A Write Buffer is needed between the Cache and Memory**
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- **Write buffer is just a FIFO:**
  - Typical number of entries: 4
  - Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle
- **Memory system designer's nightmare:**
  - Store frequency (w.r.t. time) -> 1 / DRAM write cycle
  - Write buffer saturation

The Cache Design Space

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
- The optimal choice is a compromise
  - depends on access characteristics
    - workload
    - use (I-cache, D-cache, TLB)
  - depends on technology / cost
- Simplicity often wins