inst.eecs.berkeley.edu/~cs61c

CS61C : Machine Structures

Lecture #22 CPU Design: Pipelining to Improve Performance II

2007-8-1

Scott Beamer, Instructor

Review: Processor Pipelining (1/2)

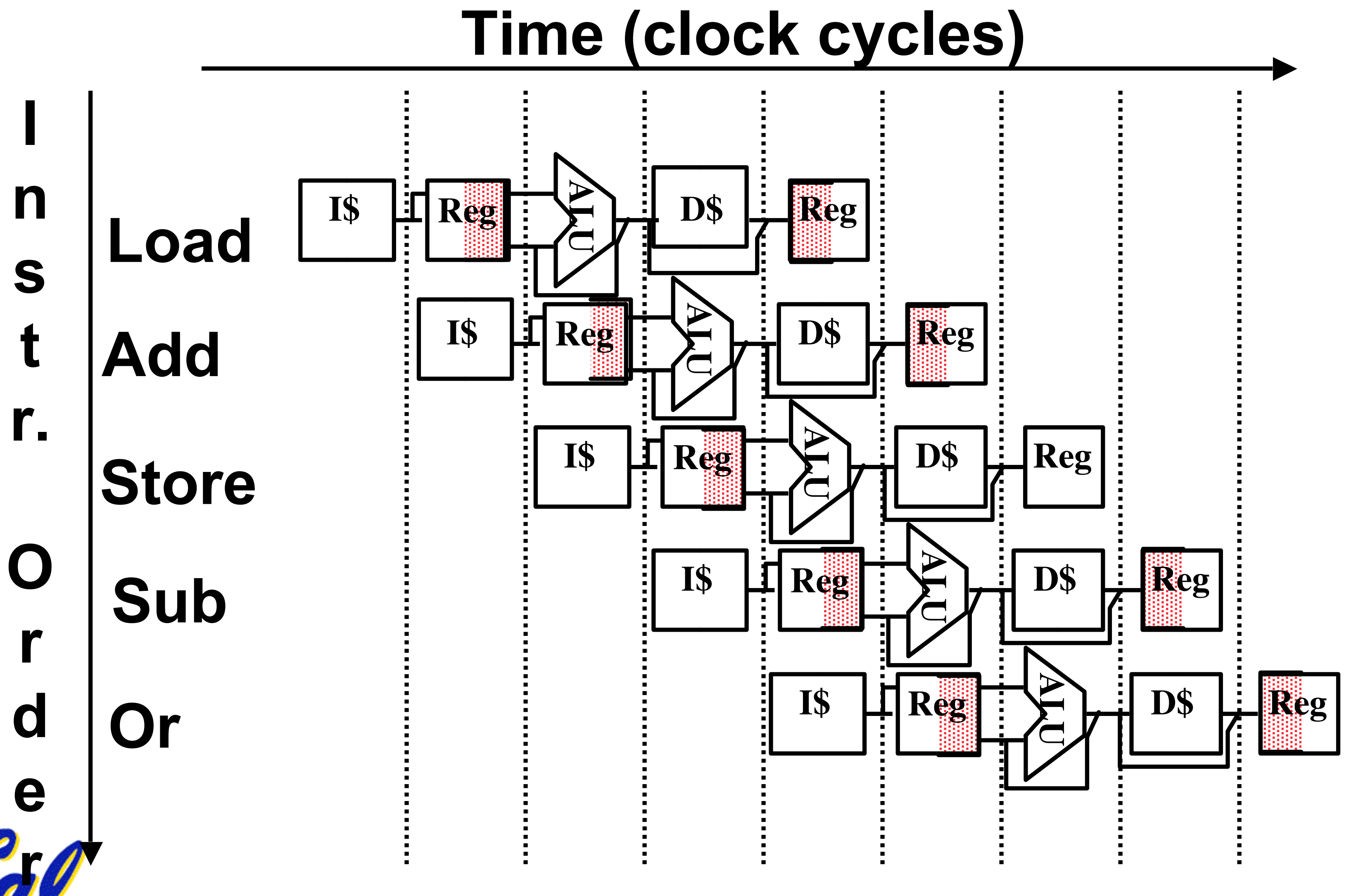
- “Pipeline registers” are added to the datapath/controller to neatly divide the single cycle processor into “pipeline stages”.
- Optimal Pipeline
  - Each stage is executing part of an instruction each clock cycle.
  - One inst. finishes during <u>each</u> clock cycle.
  - On average, execute far more quickly.
- What makes this work well?
  - Similarities between instructions allow us to use same stages for all instructions (generally).
  - Each stage takes about the same amount of time as all others: little wasted time.

Review: Pipeline (2/2)

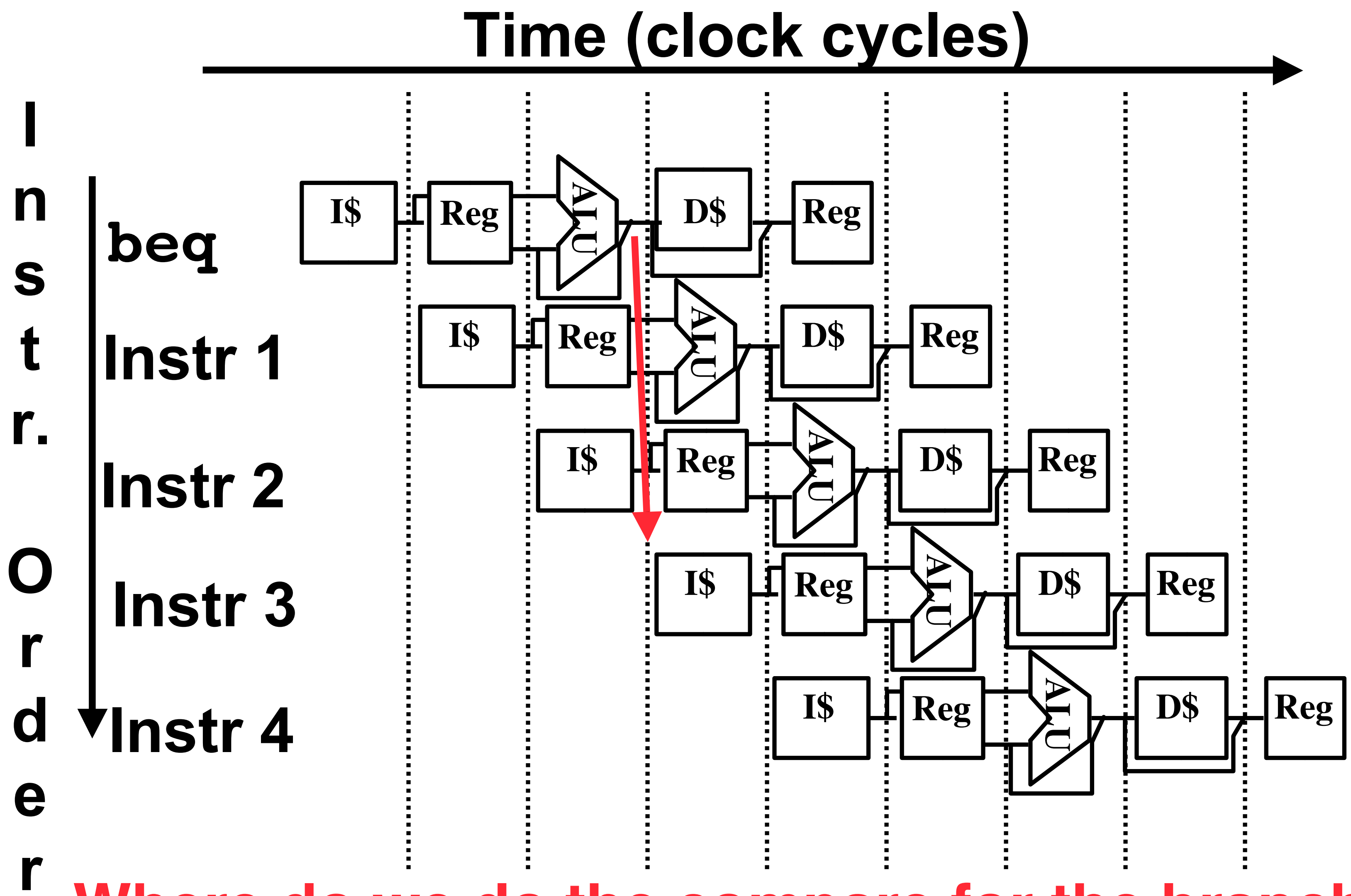
- Pipelining is a BIG IDEA
  - widely used concept
- What makes it less than perfect?
  - Structural hazards: Conflicts for resources. Suppose we had only one cache?
    $\Rightarrow$ Need more HW resources
  - Control hazards: Branch instructions effect which instructions come next.
    $\Rightarrow$ Delayed branch
  - Data hazards: Data flow between instructions.
    $\Rightarrow$ Forwarding

Graphical Pipeline Representation

(In Reg, right half highlight read, left half write)



Control Hazard: Branching (1/8)



Where do we do the compare for the branch?

Control Hazard: Branching (2/8)

- **We had put branch decision-making hardware in ALU stage**
  - therefore two more instructions after the branch will *always* be fetched, whether or not the branch is taken
- **Desired functionality of a branch**
  - if we do not take the branch, don't waste any time and continue executing normally
  - if we take the branch, don't execute any instructions after the branch, just go to the desired label

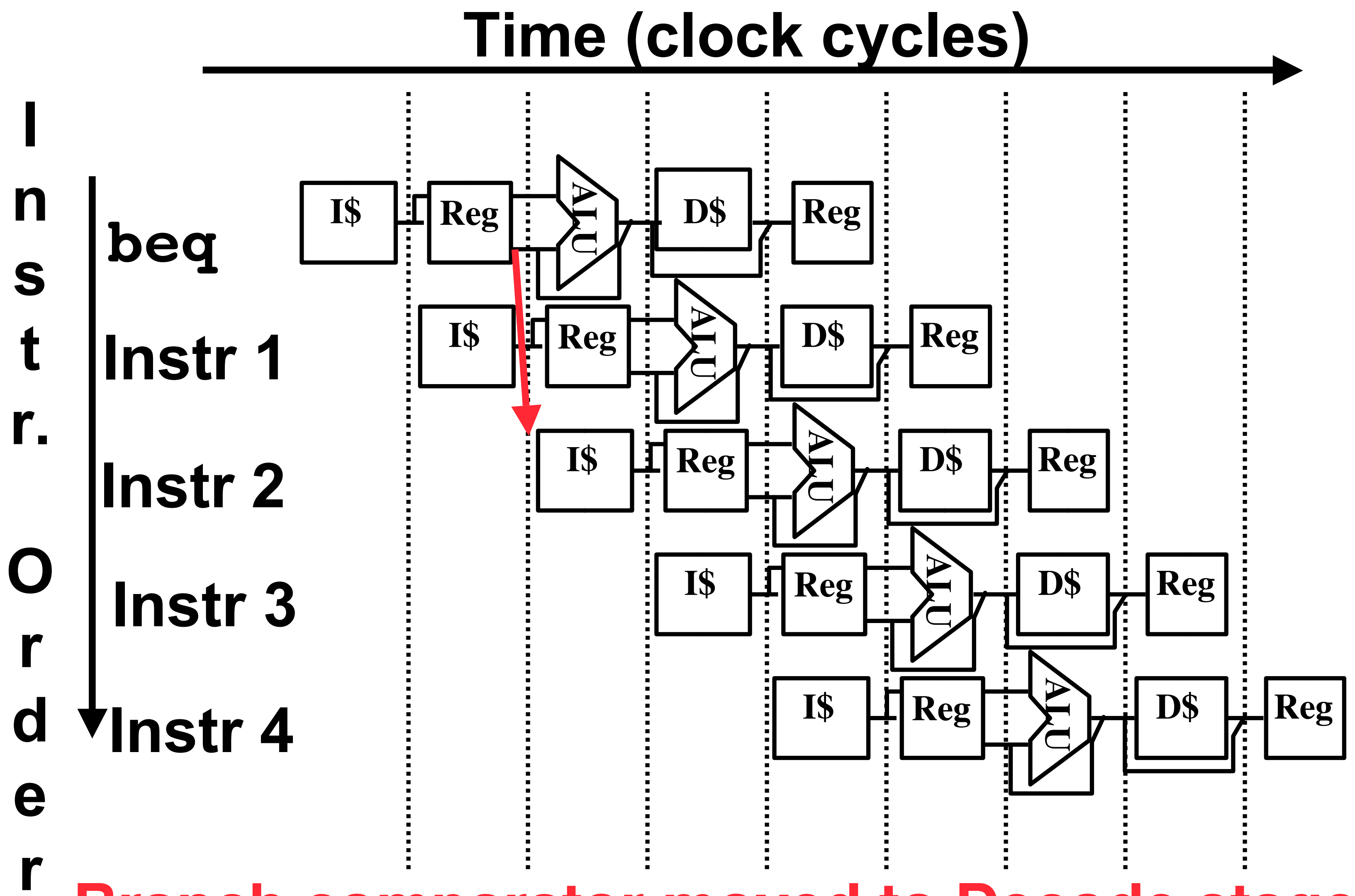
Control Hazard: Branching (3/8)

- Initial Solution: Stall until decision is made
  - insert “no-op” instructions (those that accomplish nothing, just take time) or hold up the fetch of the next instruction (for 2 cycles).
  - Drawback: branches take 3 clock cycles each (assuming comparator is put in ALU stage)

Control Hazard: Branching (4/8)

- **Optimization #1:**
  - insert special branch comparator in Stage 2
  - as soon as instruction is decoded (Opcode identifies it as a branch), immediately make a decision and set the new value of the PC
  - Benefit: since branch is complete in Stage 2, only one unnecessary instruction is fetched, so only one no-op is needed
  - Side Note: This means that branches are idle in Stages 3, 4 and 5.

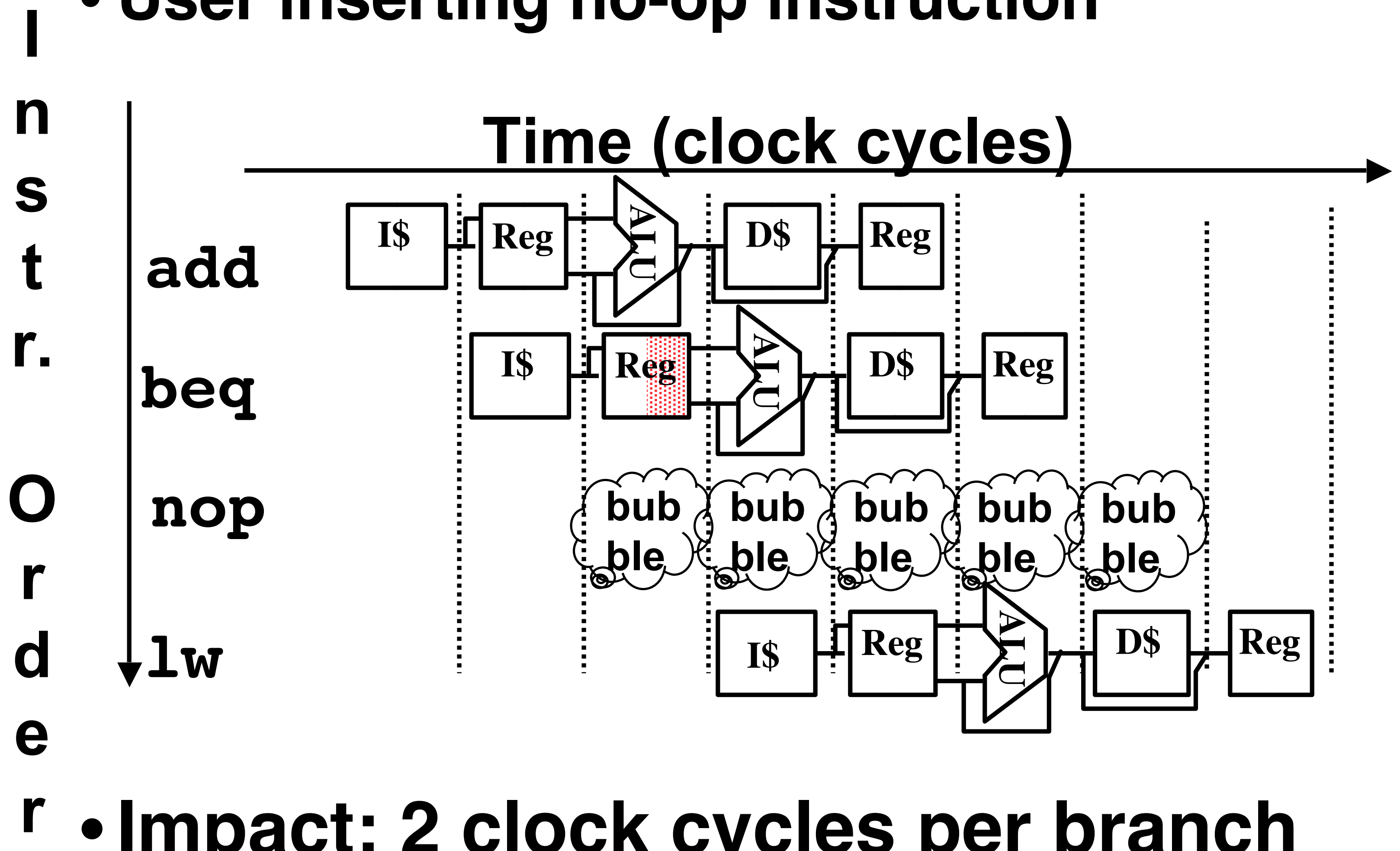
Control Hazard: Branching (5/8)



Branch comparator moved to Decode stage.

Control Hazard: Branching (6a/8)

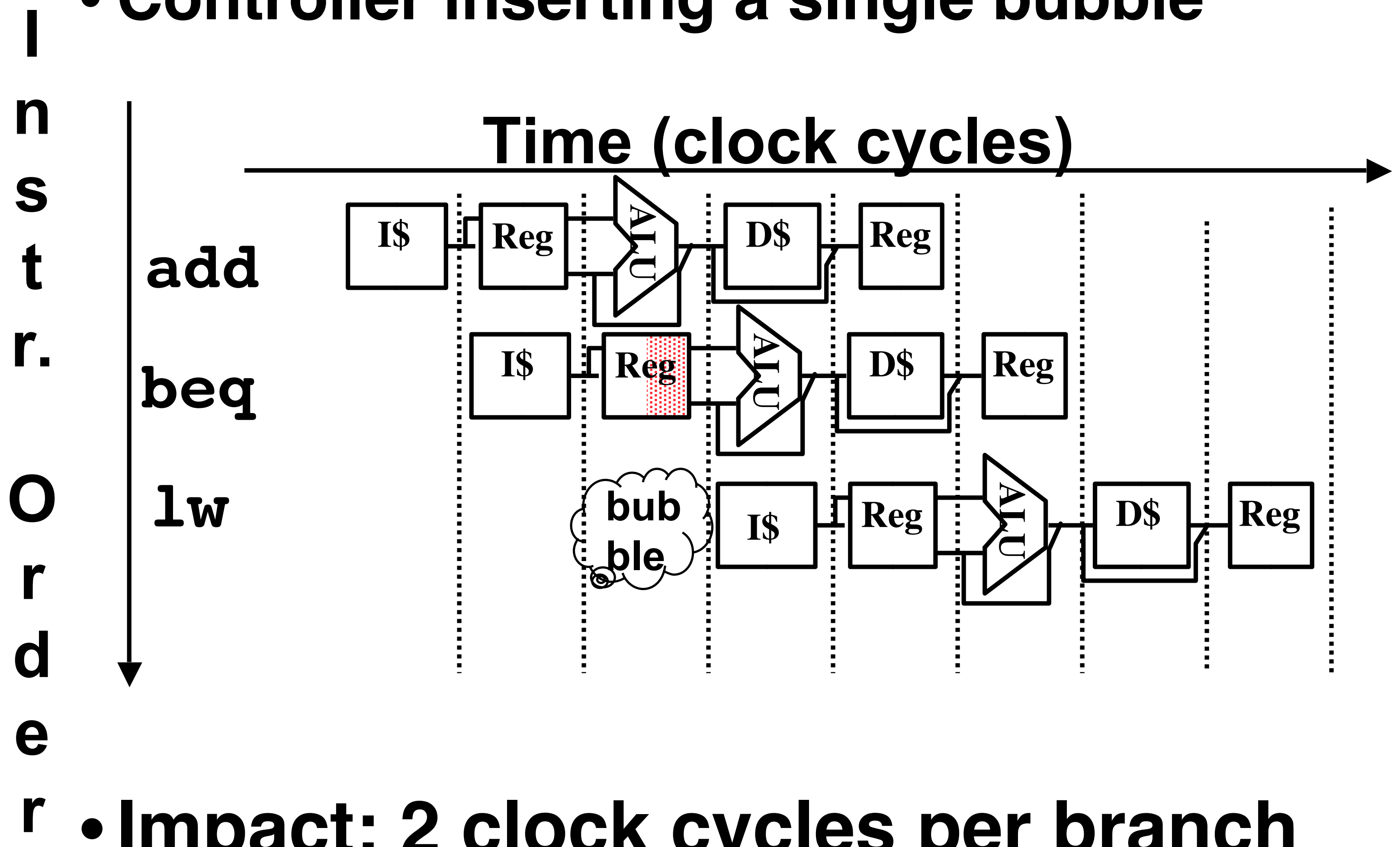
- User inserting no-op instruction



- Impact: 2 clock cycles per branch instruction ⇒ slow

Control Hazard: Branching (6b/8)

- Controller inserting a single bubble



- Impact: 2 clock cycles per branch instruction ⇒ slow

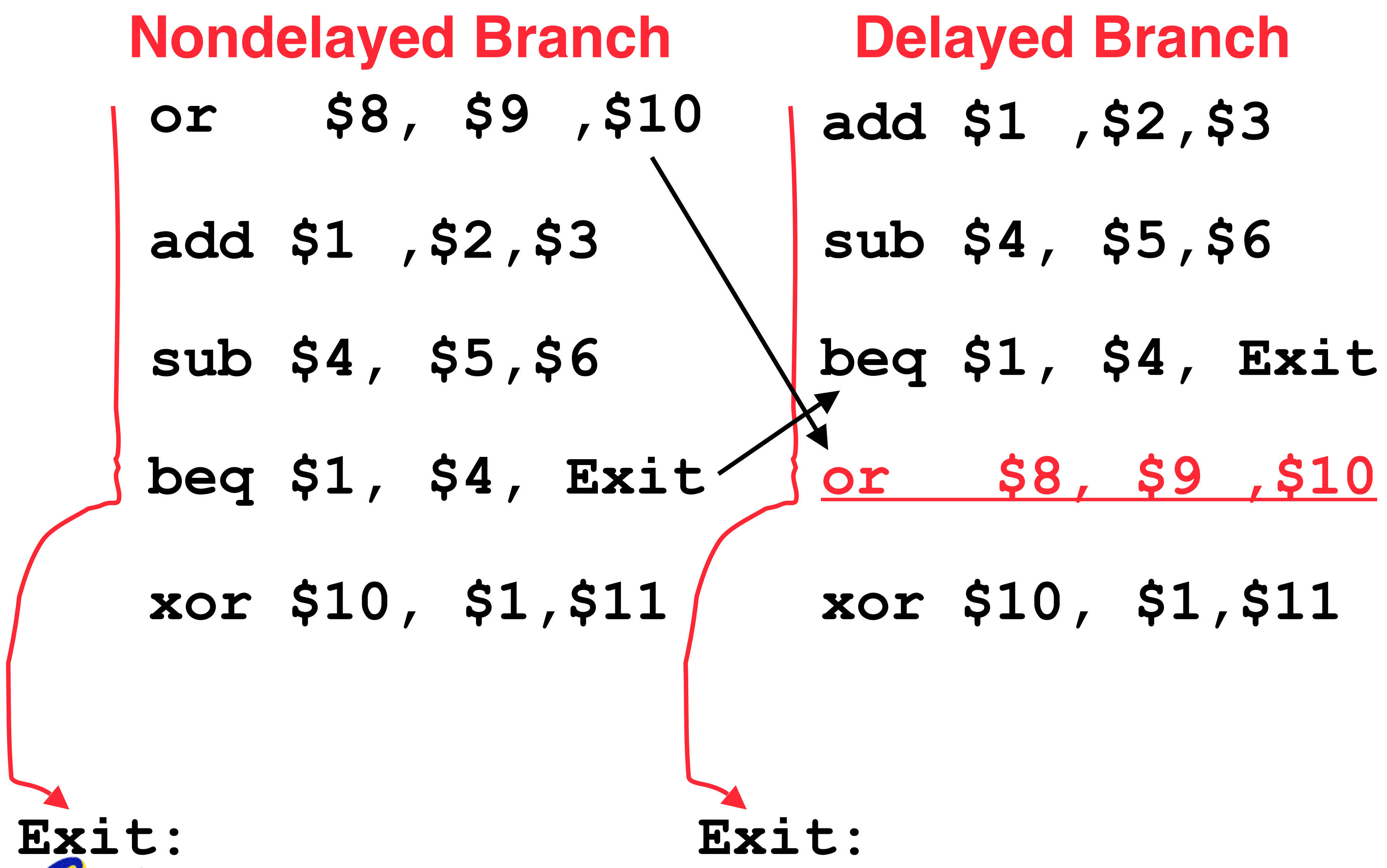
Control Hazard: Branching (7/8)

- **Optimization #2: Redefine branches**
  - **Old definition: if we take the branch, none of the instructions after the branch get executed by accident**
  - **New definition: whether or not we take the branch, the single instruction immediately following the branch gets executed (called the branch-delay slot)**
- **The term "Delayed Branch" means we always execute inst after branch**
- **This optimization is used on the MIPS**

Control Hazard: Branching (8/8)

- Notes on Branch-Delay Slot
  - Worst-Case Scenario: can always put a no-op in the branch-delay slot
  - Better Case: can find an instruction preceding the branch which can be placed in the branch-delay slot without affecting flow of the program
    - re-ordering instructions is a common method of speeding up programs
    - compiler must be very smart in order to find instructions to do this
    - usually can find such an instruction at least 50% of the time
    - Jumps also have a delay slot…

Example: Nondelayed vs. Delayed Branch



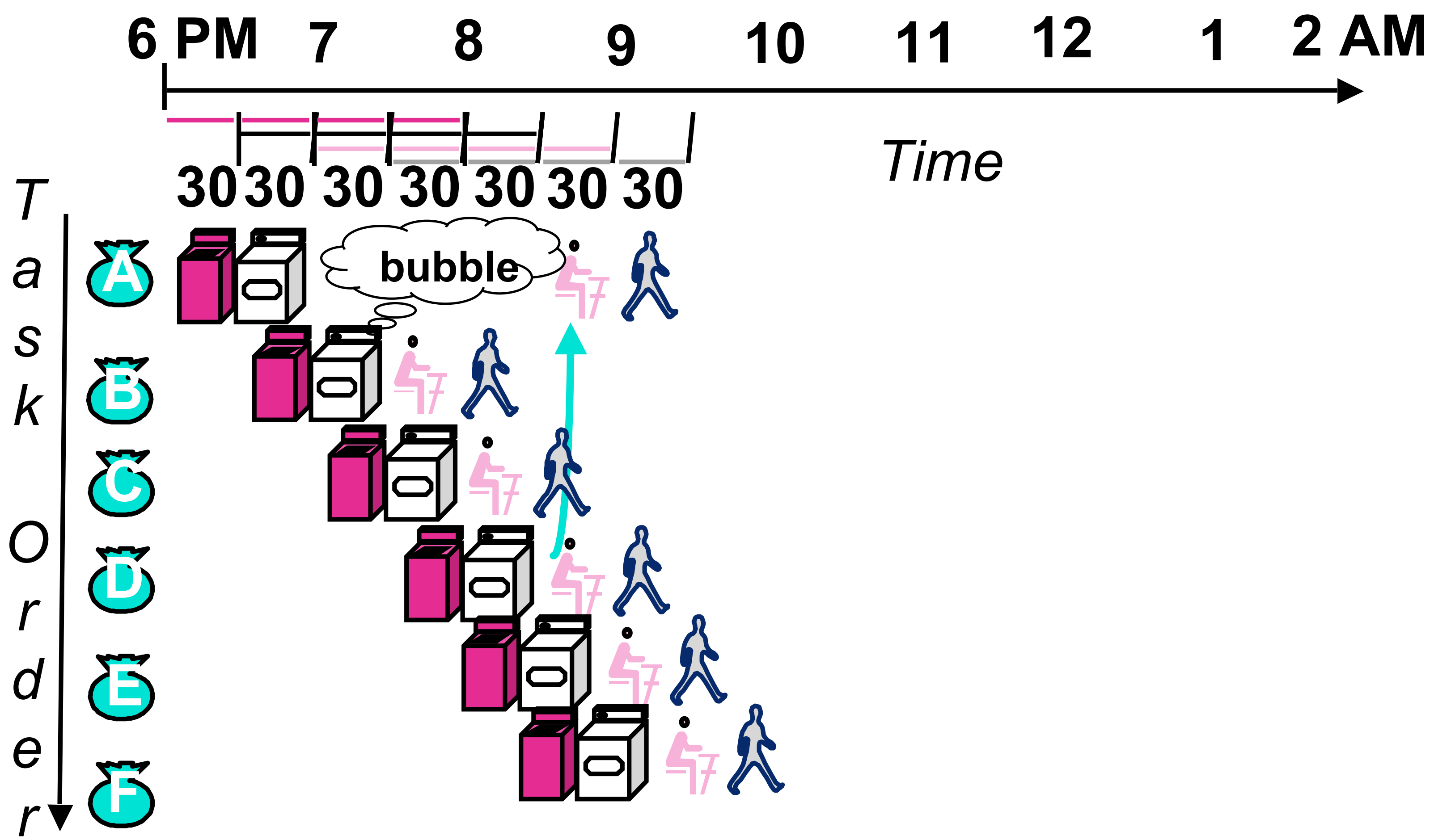
Nondelayed Branch

```
or    $8, $9 ,$10

add $1 ,$2,$3

sub $4, $5,$6

beq $1, $4, Exit

xor $10, $1,$11

Exit:
```

Delayed Branch

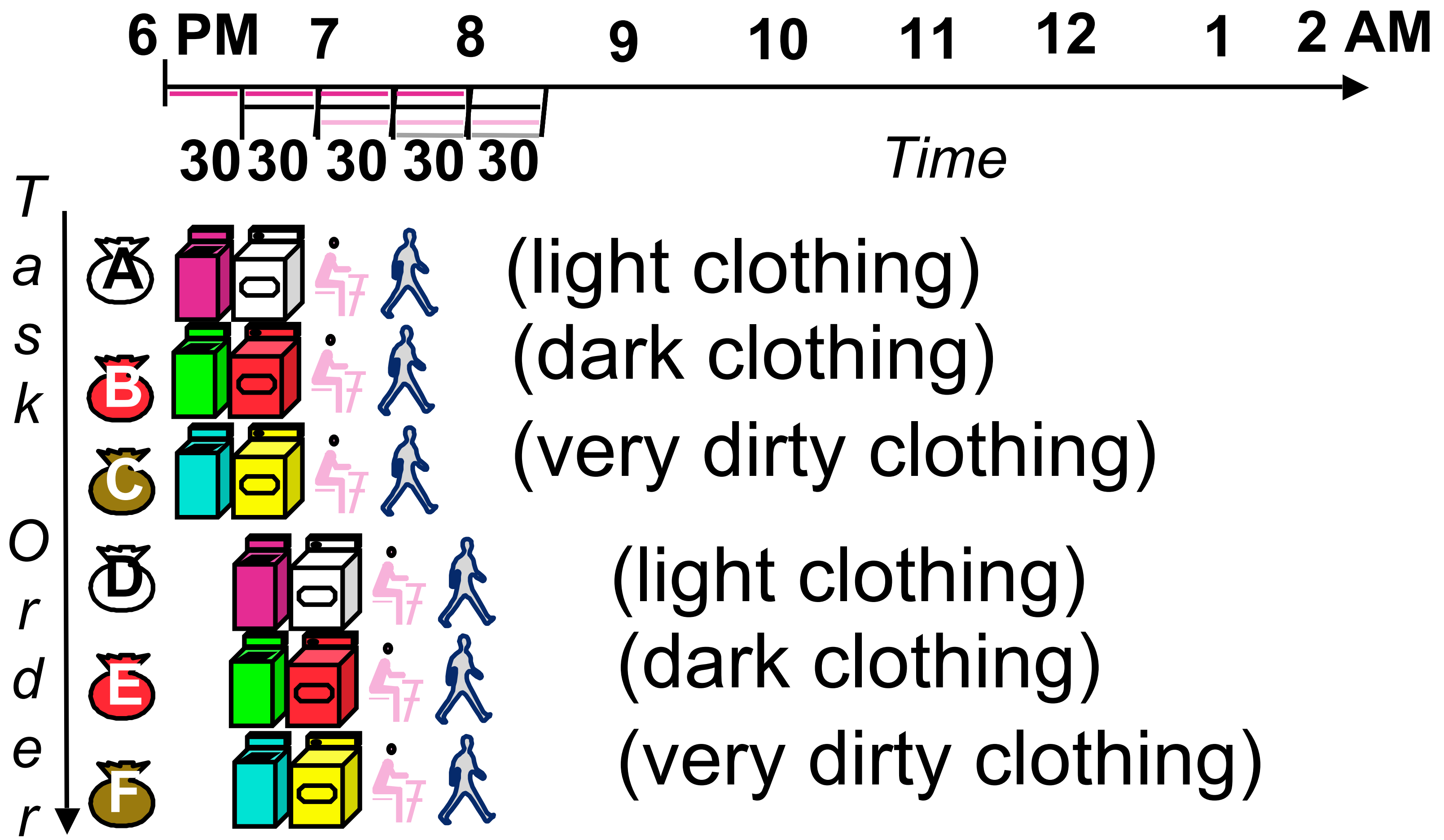
```
add $1 ,$2,$3

sub $4, $5,$6

beq $1, $4, Exit

or    $8, $9 ,$10

xor $10, $1,$11

Exit:
```

Out-of-Order Laundry: Don't Wait



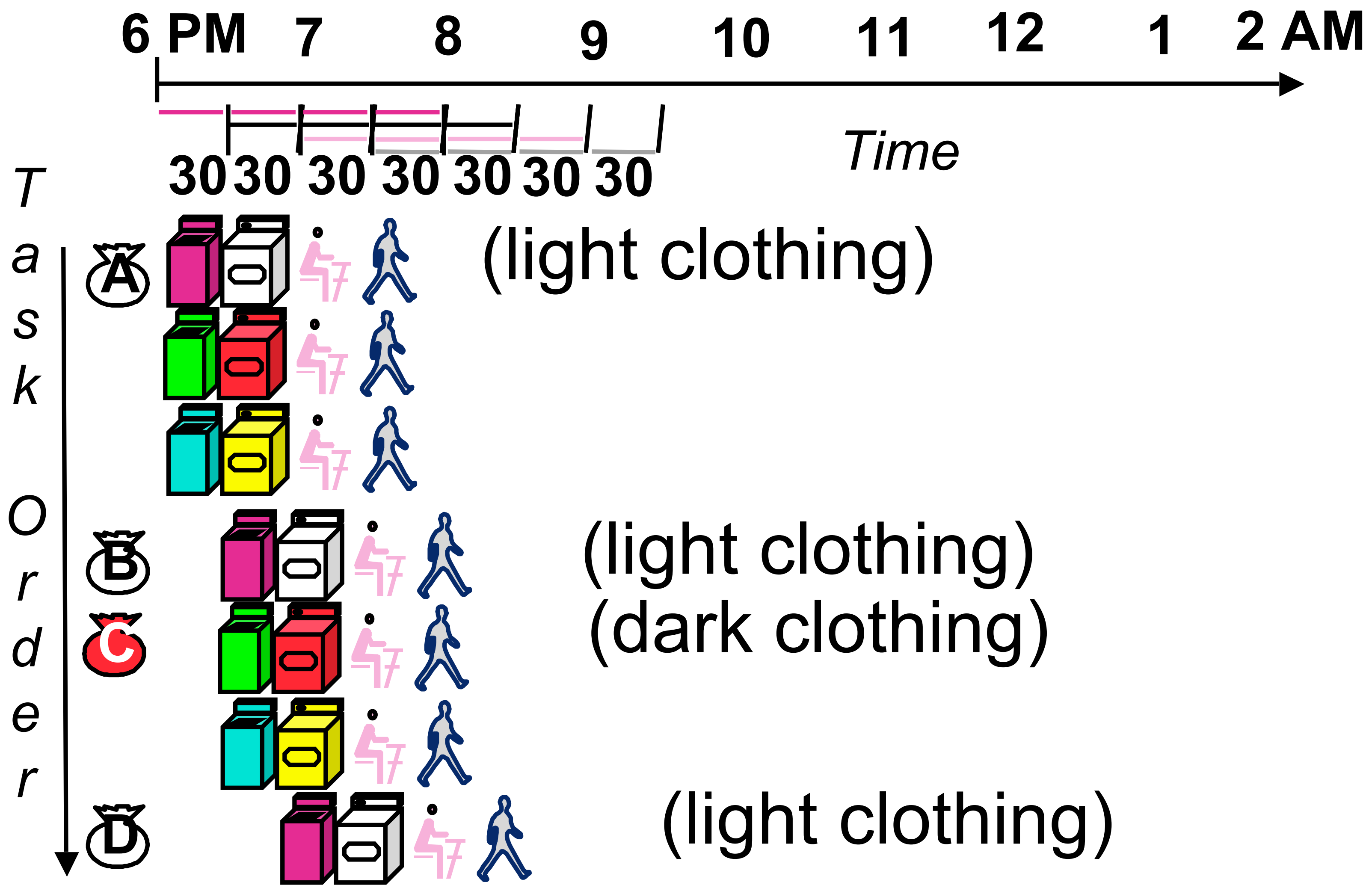
A depends on D; rest continue; need more resources to allow out-of-order

Superscalar Laundry: Parallel per stage



More resources, HW to match mix of parallel tasks?

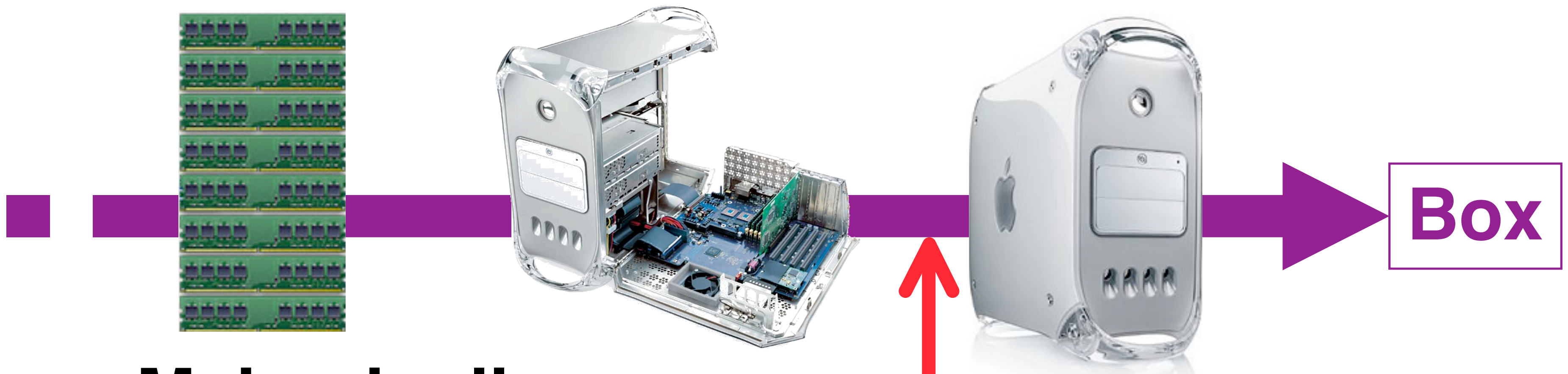
Superscalar Laundry: Mismatch Mix



Task mix underutilizes extra resources

Real-world pipelining problem

- You're the manager of a HUGE assembly plant to build computers.

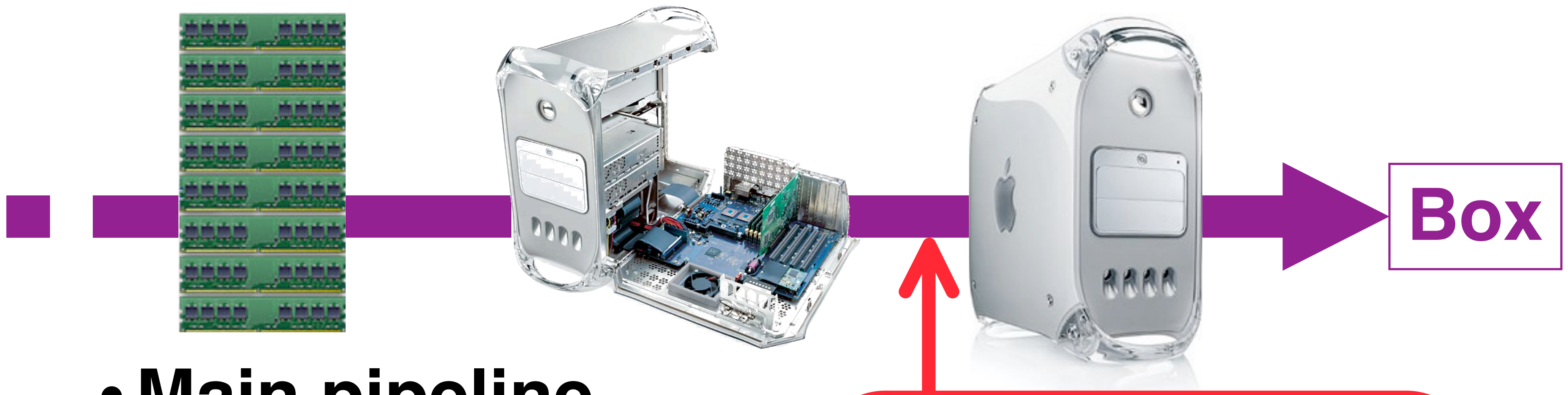


- Main pipeline
  - 10 minutes/ pipeline stage
  - 60 stages
  - Latency: 10hr

Problem: need to run 2 hr test before done..help!

Real-world pipelining problem solution 1

- You remember: "a pipeline frequency is limited by its slowest stage", so…

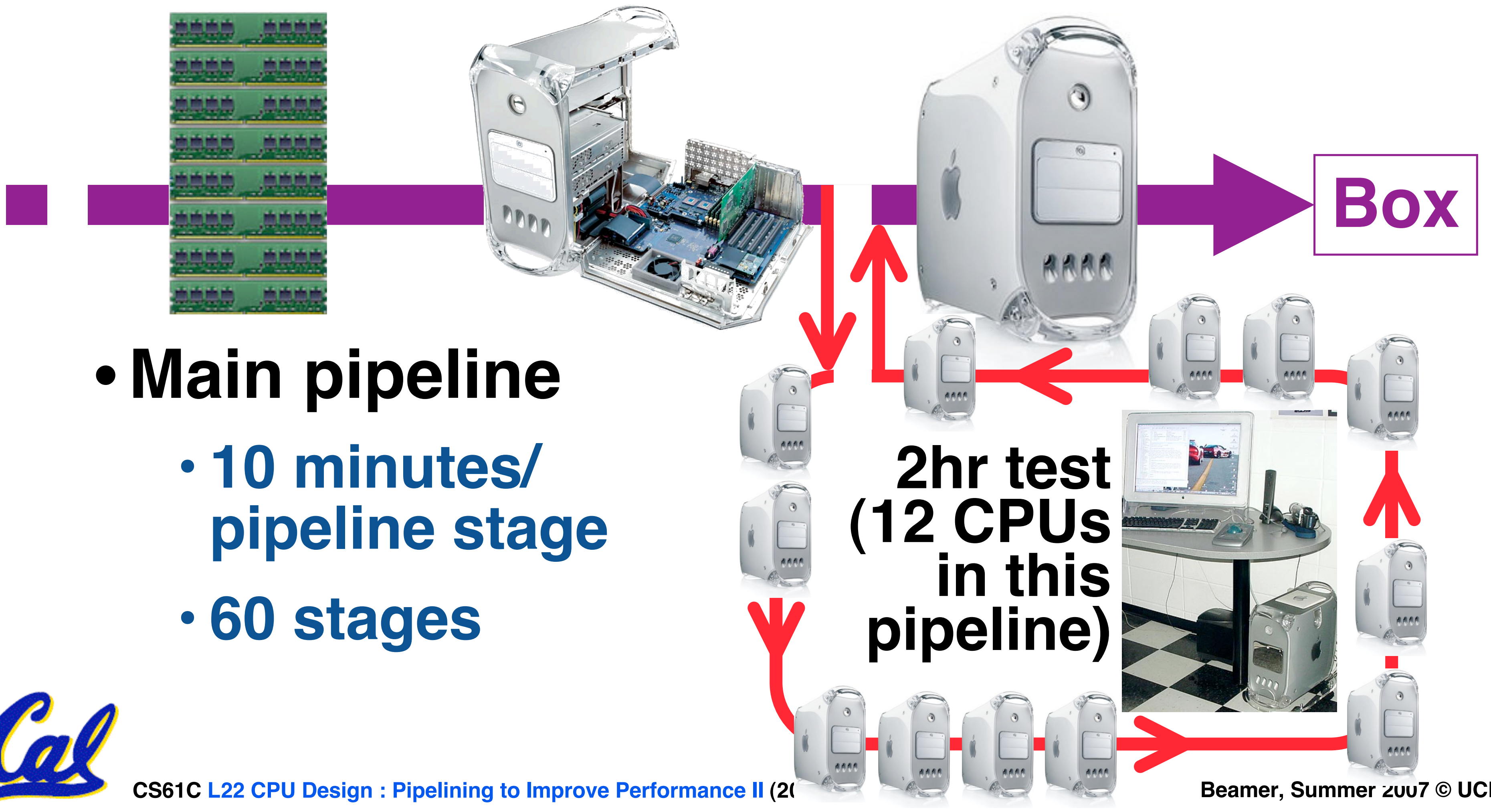


- Main pipeline
  - 2hours/ pipeline stage
  - 60 stages
  - Latency: 120hr

Problem: need to run 2 hr test before done..help!

Real-world pipelining problem solution 2

- Create a sub-pipeline!



- Main pipeline
  - 10 minutes/ pipeline stage
  - 60 stages

Peer Instruction (1/2)

Assume 1 instr/clock, delayed branch, 5 stage pipeline, forwarding, interlock on unresolved load hazards (after 10^3 loops, so pipeline full)

```
Loop:     lw    $t0, 0($s1)
          addu  $t0, $t0, $s2
          sw    $t0, 0($s1)
          addiu $s1, $s1, -4
          bne   $s1, $zero, Loop
          nop
```

- **How many pipeline stages (clock cycles) per loop iteration to execute this code?**

1
2
3
4
5
6
7
8
9
10

Peer Instruction Answer (1/2)

- **Assume 1 instr/clock, delayed branch, 5 stage pipeline, forwarding, interlock on unresolved load hazards. 10^3 iterations, so pipeline full.**

2. (data hazard so stall)

```
Loop: 1. lw     $t0, 0($s1)
      3. addu   $t0, $t0, $s2
      4. sw     $t0, 0($s1)
      5. addiu  $s1, $s1, -4
      7. bne    $s1, $zero, Loop
      8. nop
```

6. (!= in DCD)

(delayed branch so exec. nop)

- **How many pipeline stages (clock cycles) per loop iteration to execute this code?**

1 2 3 4 5 6 7 [✓ 8] 9 10

Peer Instruction (2/2)

Assume 1 instr/clock, delayed branch, 5 stage pipeline, forwarding, interlock on unresolved load hazards (after 10^3 loops, so pipeline full). **Rewrite this code to reduce pipeline stages (clock cycles) per loop to as few as possible.**

```
Loop:     lw    $t0, 0($s1)
          addu  $t0, $t0, $s2
          sw    $t0, 0($s1)
          addiu $s1, $s1, -4
          bne   $s1, $zero, Loop
          nop
```

- **How many pipeline stages (clock cycles) per loop iteration to execute this code?**

1
2
3
4
5
6
7
8
9
10

Peer Instruction (2/2) How long to execute?

- Rewrite this code to reduce clock cycles per loop to as few as possible:

(no hazard since extra cycle)

```
Loop: 1. lw    $t0, 0($s1)
      2. addiu $s1, $s1, -4
      3. addu  $t0, $t0, $s2
      4. bne   $s1, $zero, Loop
      5. sw    $t0, +4($s1)
```

(modified sw to put past addiu)

- How many pipeline stages (clock cycles) per loop iteration to execute your revised code? (assume pipeline is full)

1 2 3 4 [✓ 5] 6 7 8 9 10

“And in Early Conclusion..”

- Pipeline challenge is hazards
  - Forwarding helps w/many data hazards
  - Delayed branch helps with control hazard in 5 stage pipeline
  - Load delay slot / interlock necessary
- More aggressive performance:
  - Superscalar
  - Out-of-order execution

Administrivia

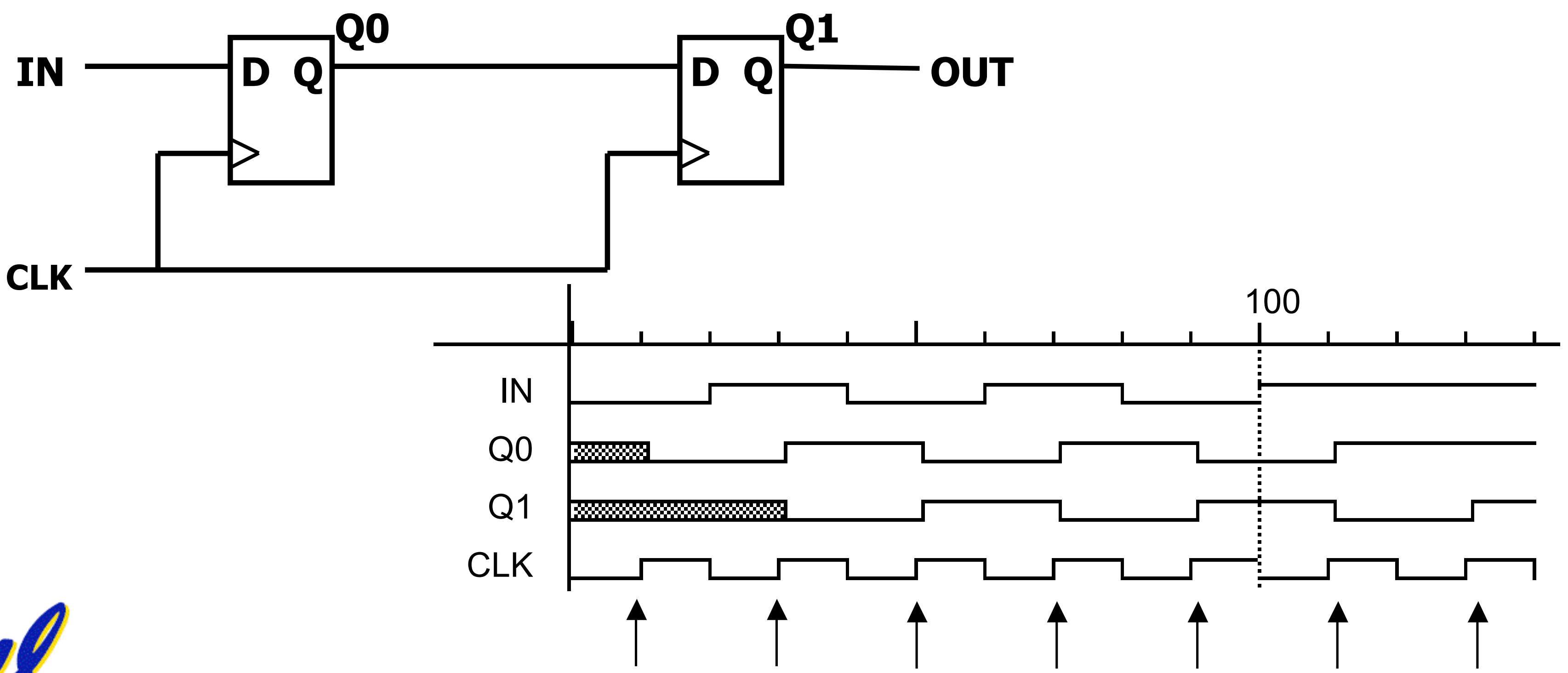
- Assignments
  - HW7 due 8/2
  - Proj3 due 8/5
- Midterm Regrades due Today
- Logisim in lab is now 2.1.6
  - `java -jar ~cs61c/bin/logisim`
- Valerie's OH on Thursday moved to 10-11 for this week

Why Doesn't It Work?

- DO NOT MESS WITH THE CLOCK
  - Crafty veterans may do it very rarely and carefully
- Doing so will cause unpredictable and hard to track errors
- Following slides are from CS 150 Lecture by Prof. Katz

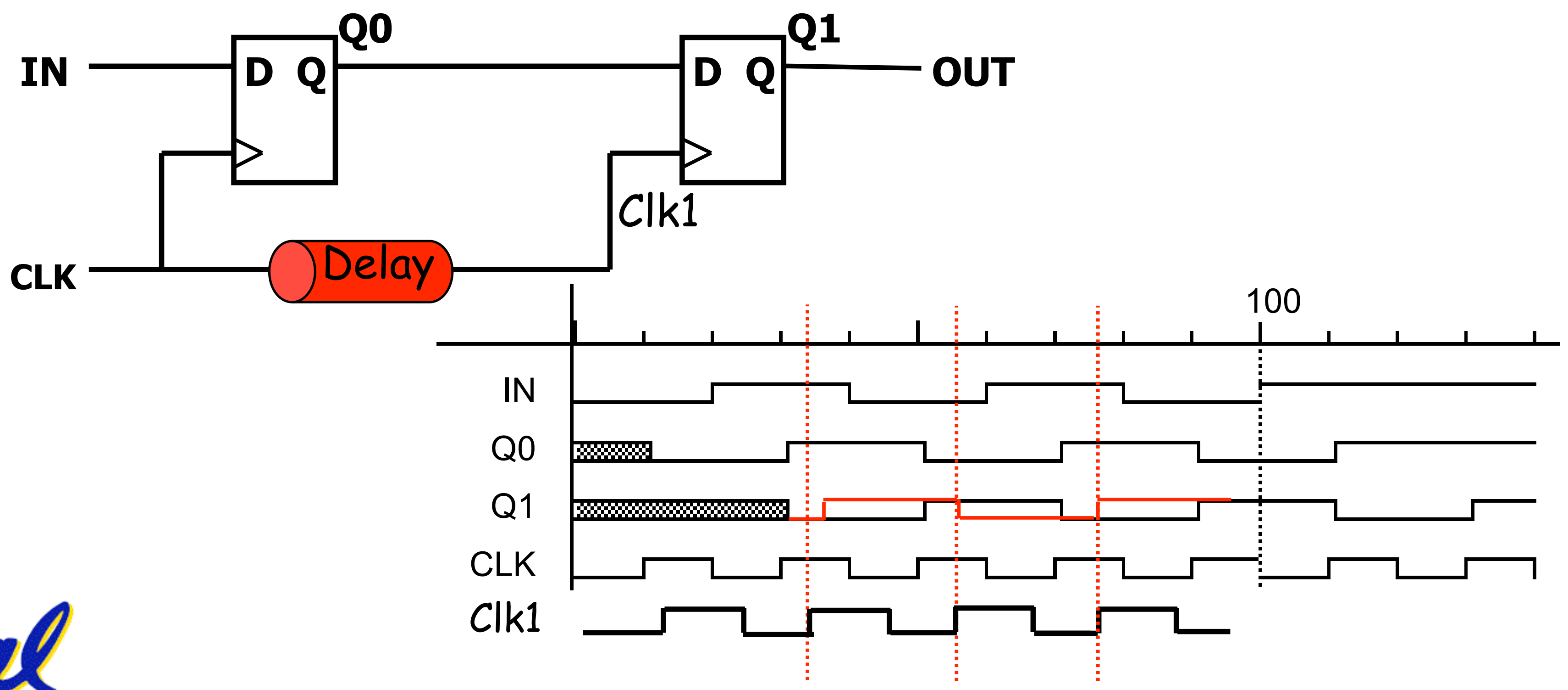
Cascading Edge-triggered Flip-Flops

- **Shift register**
  - New value goes into first stage
  - While previous value of first stage goes into second stage
  - Consider setup/hold/propagation delays (prop must be > hold)



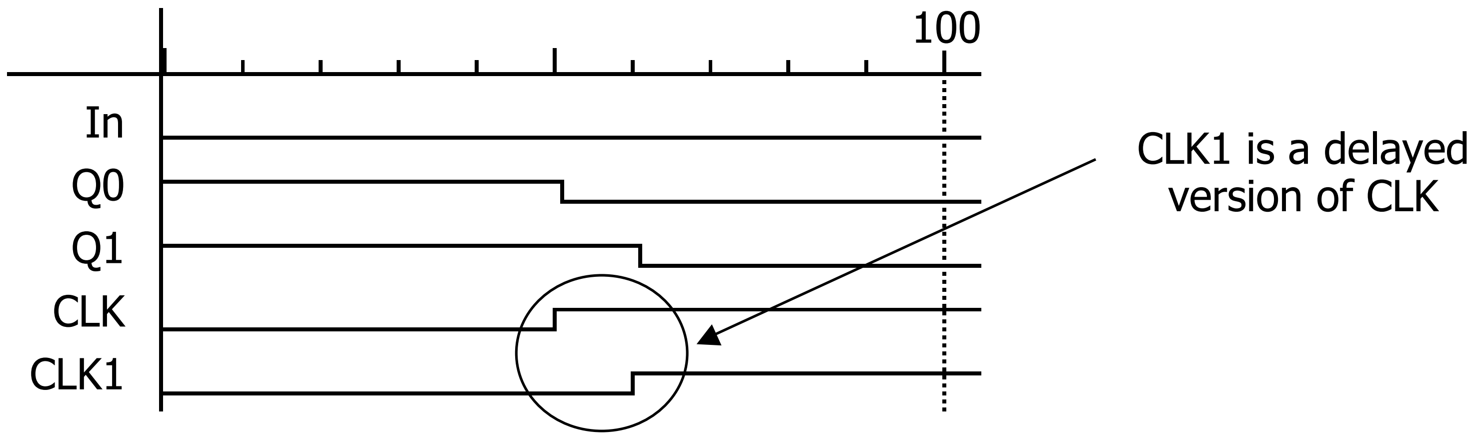
Cascading Edge-triggered Flip-Flops

- **Shift register**
  - New value goes into first stage
  - While previous value of first stage goes into second stage
  - Consider setup/hold/propagation delays (prop must be > hold)



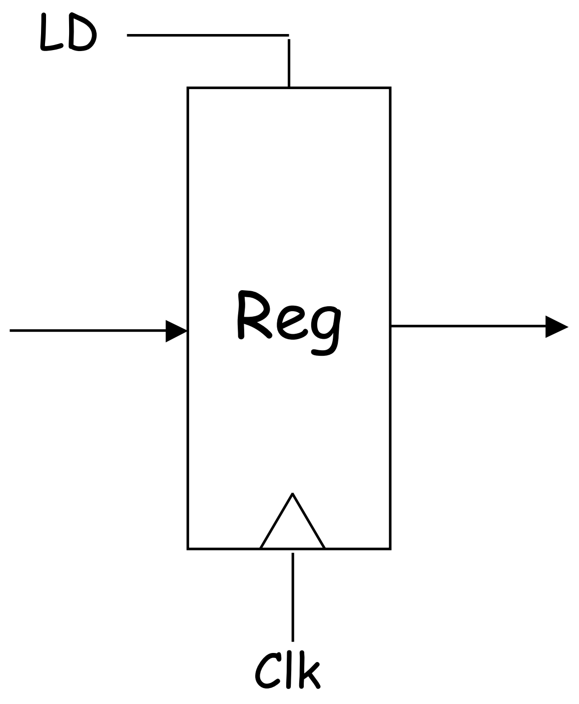
Clock Skew

- **The problem**
  - **Correct behavior assumes next state of all storage elements determined by all storage elements at the same time**
  - **Difficult in high-performance systems because time for clock to arrive at flip-flop is comparable to delays through logic (and will soon become greater than logic delay)**
  - **Effect of skew on cascaded flip-flops:**

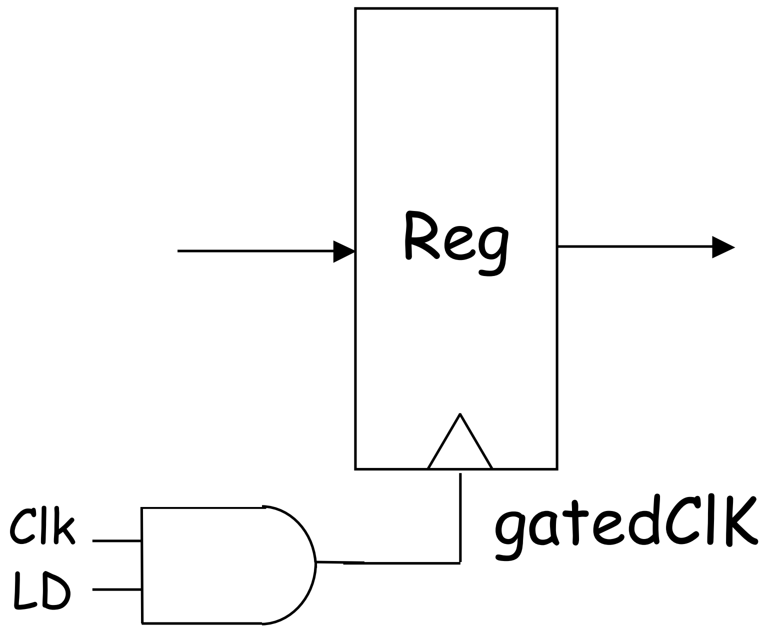


original state: IN = 0, Q0 = 1, Q1 = 1
due to skew, next state becomes: Q0 = 0, Q1 = 0, and not Q0 = 0, Q1 = 1

Why Gating of Clocks is Bad!



GOOD

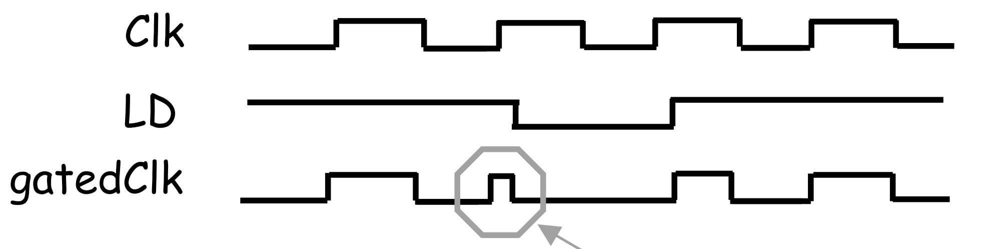


BAD

Do NOT Mess With Clock Signals!

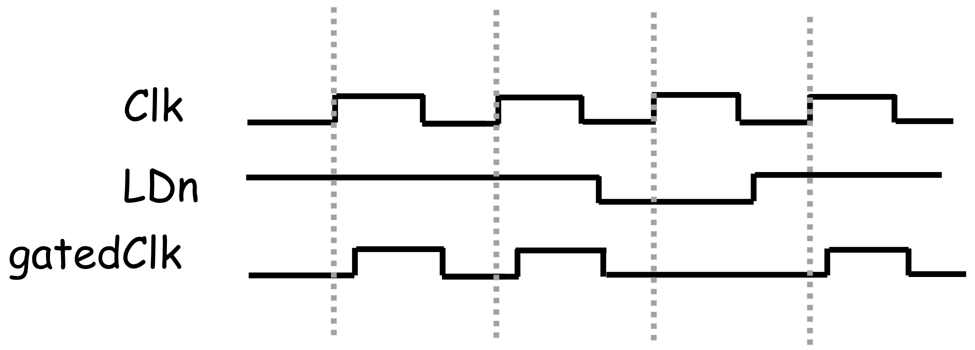
Why Gating of Clocks is Bad!

LD generated by FSM
shortly after rising edge of CLK



Runt pulse plays HAVOC with register internals!

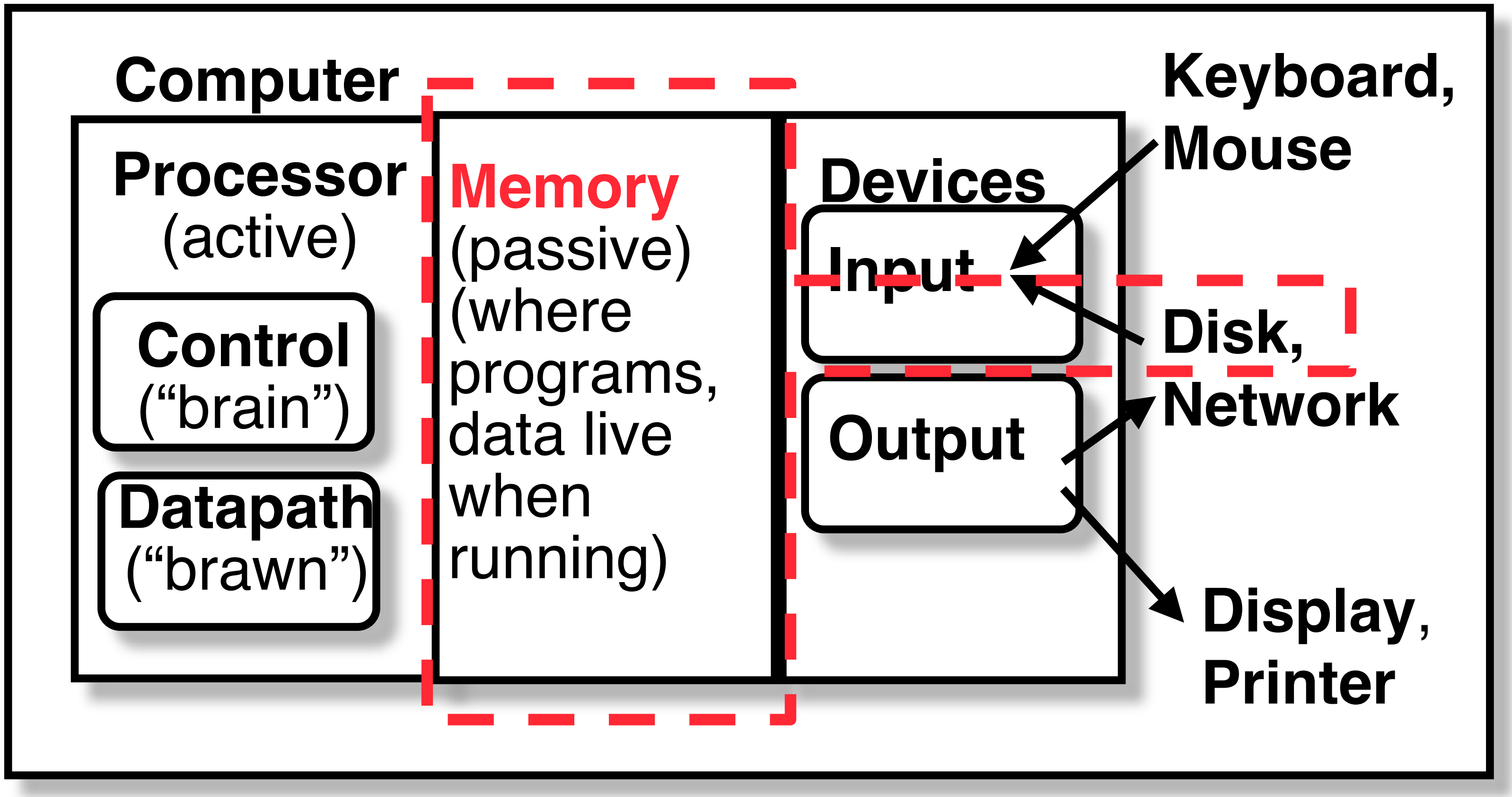
NASTY HACK: delay LD through negative edge triggered FF to ensure that it won't change during next positive edge event



Clk skew PLUS LD delayed by half clock cycle …
What is the effect on your register transfers?

Do NOT Mess With Clock Signals!

The Big Picture

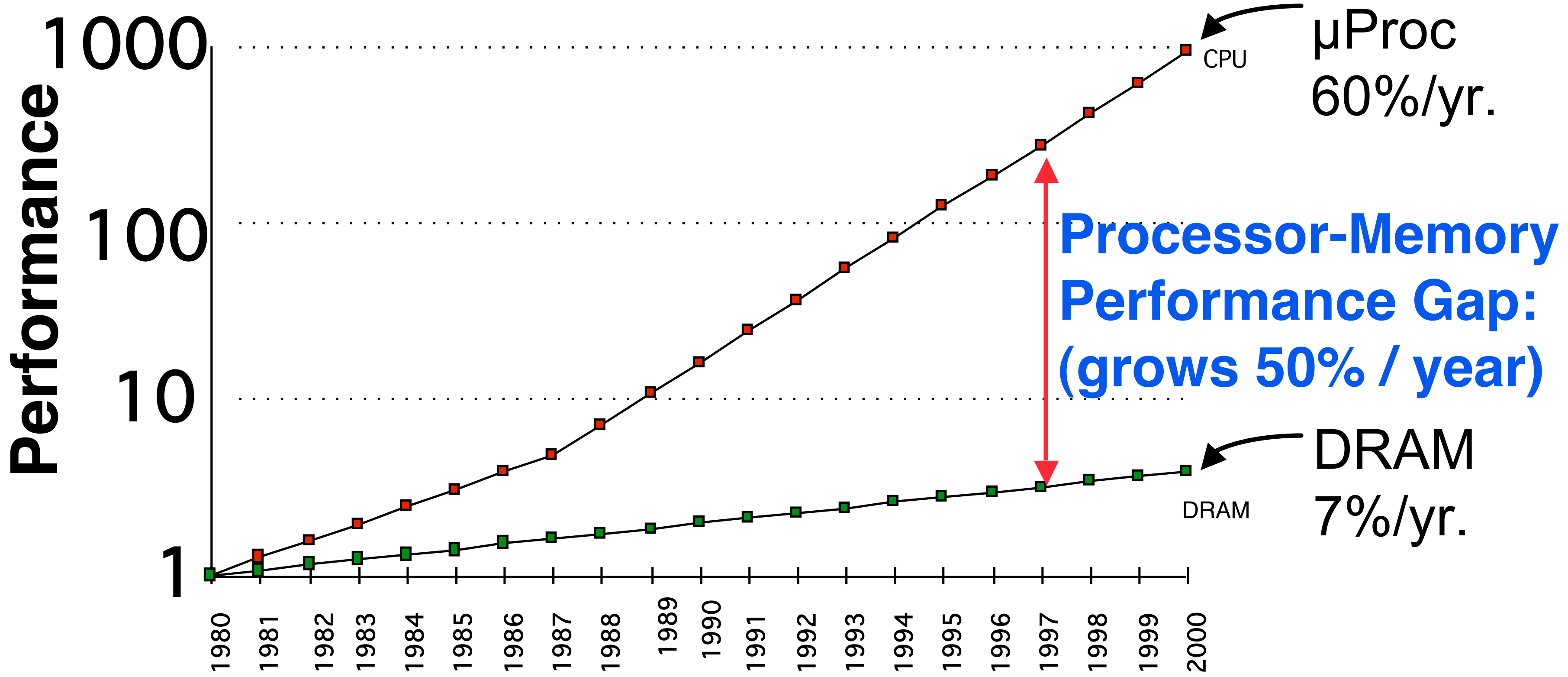


Memory Hierarchy

Storage in computer systems:

- Processor
  - holds data in register file (~100 Bytes)
  - Registers accessed on nanosecond timescale
- Memory (we'll call "main memory")
  - More capacity than registers (~Gbytes)
  - Access time ~50-100 ns
  - Hundreds of clock cycles per memory access?!
- Disk
  - HUGE capacity (virtually limitless)
  - VERY slow: runs ~milliseconds

Motivation: Why We Use Caches (written $)

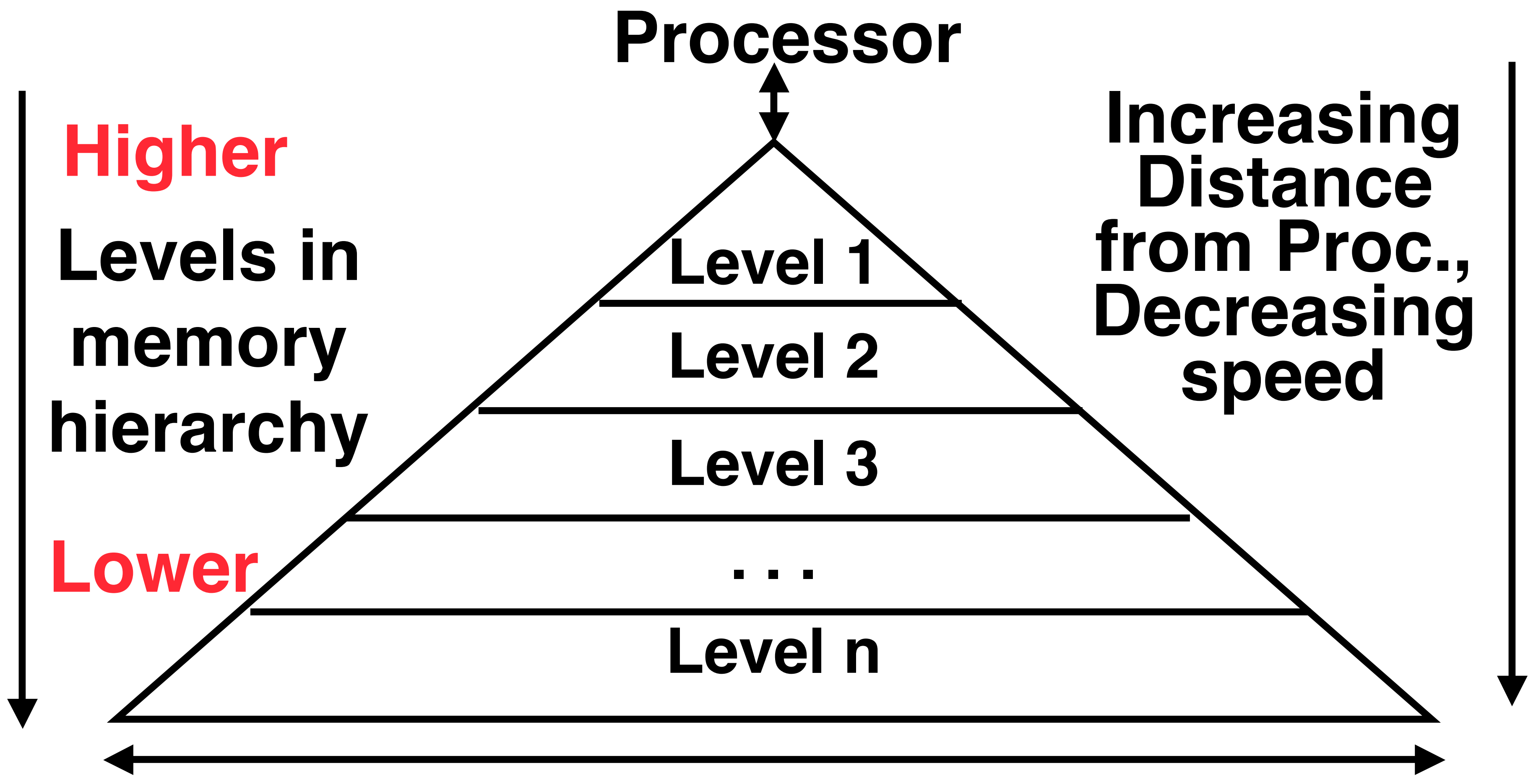


- 1989 first Intel CPU with cache on chip
- 1998 Pentium III has two levels of cache on chip

Memory Caching

- Mismatch between processor and memory speeds leads us to add a new level: a memory cache
- Implemented with same IC processing technology as the CPU (usually integrated on same chip): faster but more expensive than DRAM memory.
- Cache is a copy of a subset of main memory.
- Most processors have separate caches for instructions and data.

Memory Hierarchy


Processor
Higher
Levels in memory hierarchy
Lower
Level 1
Level 2
Level 3
. . .
Level n
Increasing Distance from Proc., Decreasing speed

Size of memory at each level

As we move to deeper levels the latency goes up and price per bit goes down.

Memory Hierarchy

- If level closer to Processor, it is:
  - smaller
  - faster
  - subset of lower levels (contains most recently used data)
- Lowest Level (usually disk) contains all available data (or does it go beyond the disk?)
- Memory Hierarchy presents the processor with the illusion of a very large very fast memory.