18-447 Lecture 8:
Data Hazard and Resolution

James C. Hoe
Department of ECE
Carnegie Mellon University

Housekeeping

- Your goal today
  - detect and resolve data hazards in in-order instruction pipelines
  - control dependence next time
- Notices
  - HW 2, due Mon 2/19
  - Lab 2, status check wk6, due wk7 (Handout #7)
- Readings
  - P&H Ch 4

Instruction Pipeline Reality

- Not identical tasks
  - coalescing instruction types into one "multi-function" pipe
  - external fragmentation (some idle stages)
- Not uniform suboperations
  - group or sub-divide steps into stages to minimize variance
  - internal fragmentation (some too-fast stages)
- Not independent tasks
  - dependency detection and resolution
  - next lecture(s)

Recall

Even more messy if not RISC

Data Dependence

Data dependence

$x3 \leftarrow x1 \ op \ x2$ **Read-after-Write (RAW)**

. . . .

$x5 \leftarrow x3 \ op \ x4$

false dependence

Anti-dependence

$x3 \leftarrow x1 \ op \ x2$ **Write-after-Read (WAR)**

. . . .

$x1 \leftarrow x4 \ op \ x5$

Output-dependence

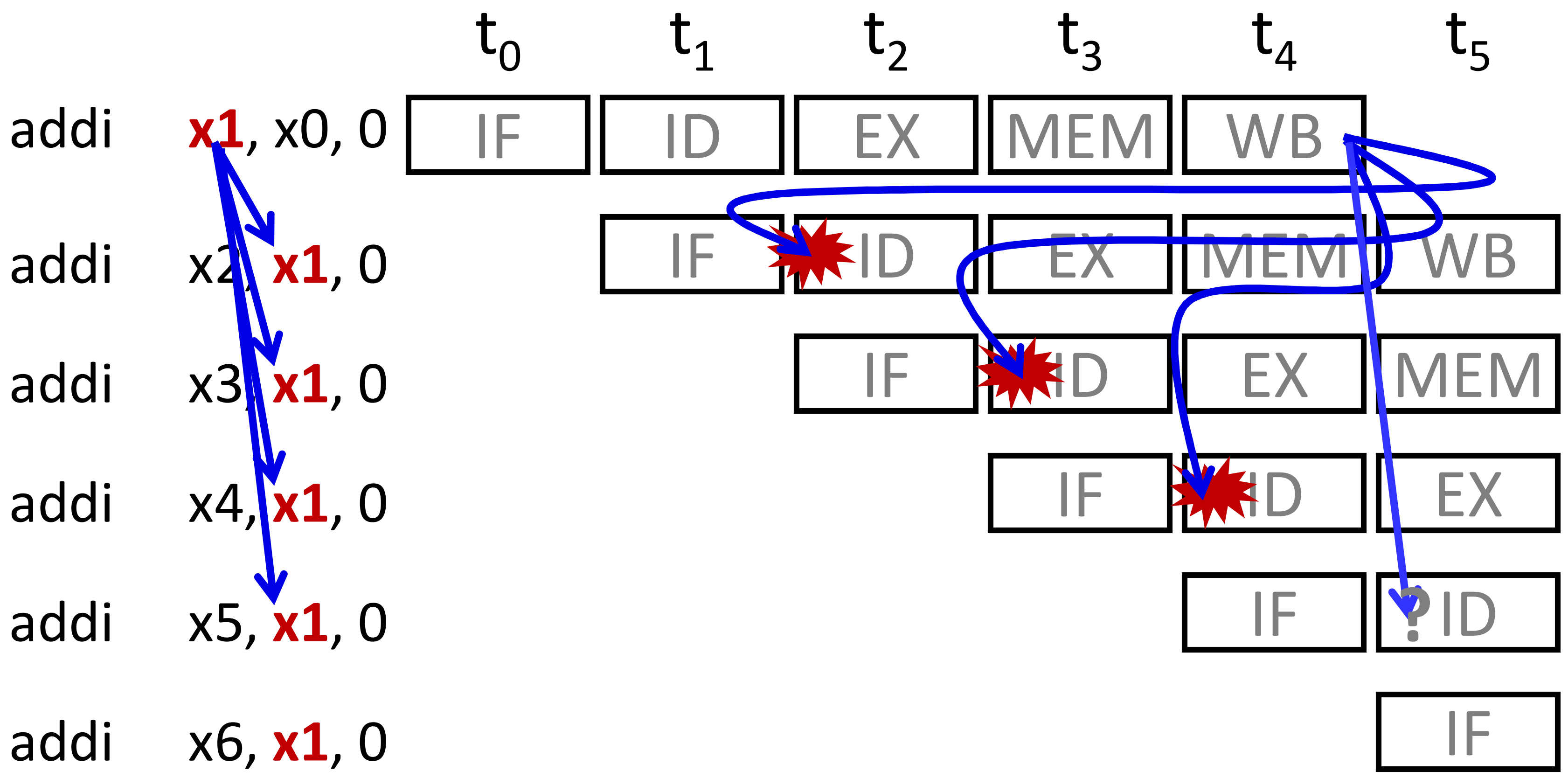
$x3 \leftarrow x1 \ op \ x2$ **Write-after-Write (WAW)**

. . . .

$x3 \leftarrow x6 \ op \ x7$

Don't forget memory instructions

Dependence vs Hazard: e.g. RAW



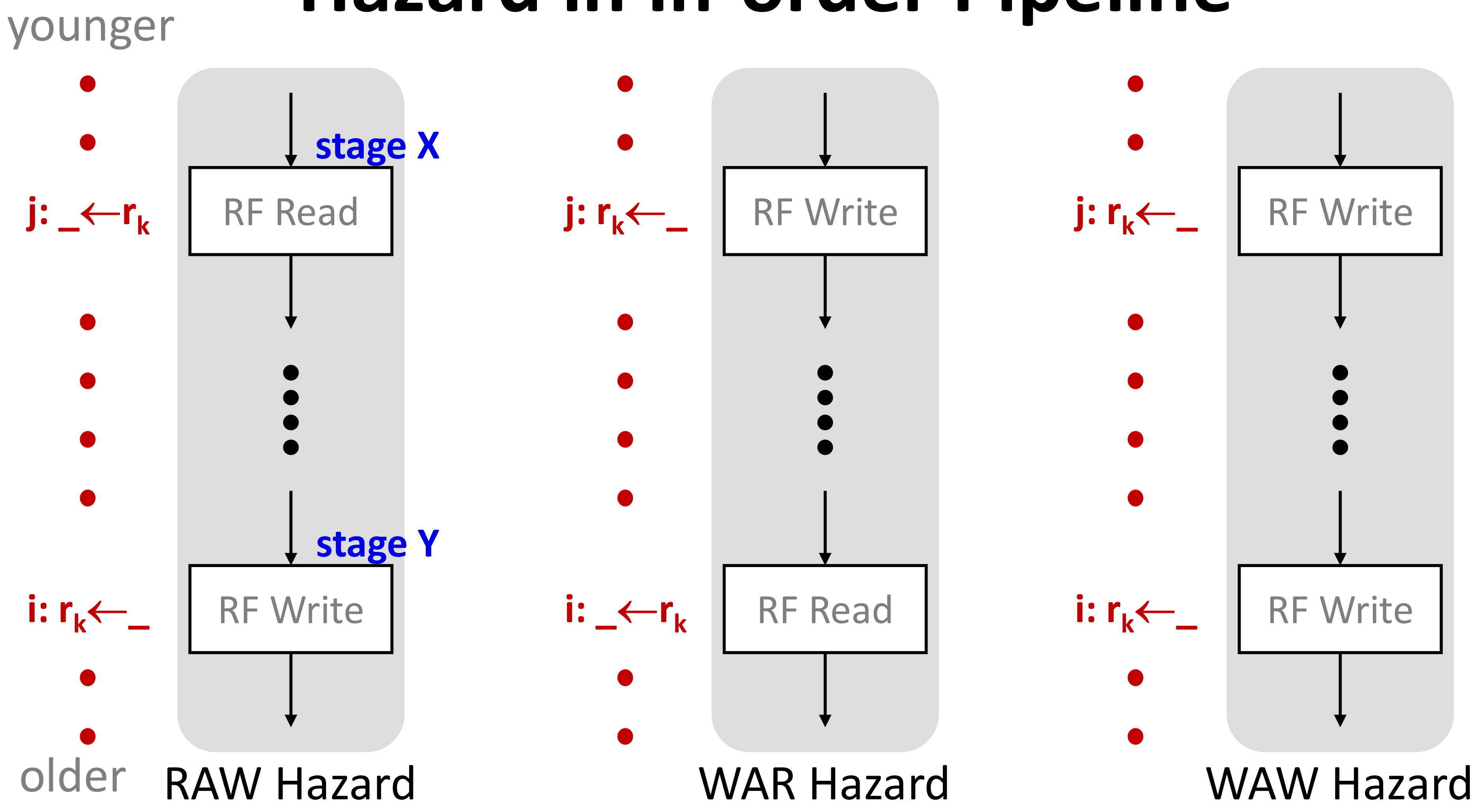
Dependence is property of program;
hazards specific to microarchitecture

Register Data Hazard Analysis

	R/I-Type	LW	SW	Bxx	Jal	Jalr
IF						
ID	**read RF**	**read RF**	**read RF**	**read RF**		**read RF**
EX						
MEM						
WB	**write RF**	**write RF**			**write RF**	**write RF**

- For a given pipeline, when is there a register data hazard between 2 dependent instructions?
  - dependence type: RAW, WAR, WAW?
  - instruction types involved?
  - distance between the two instructions?

Hazard in In-order Pipeline



$\text{dist}_{\text{dependence}}(\mathbf{i},\mathbf{j}) \le \text{dist}_{\text{hazard}}(\mathbf{X},\mathbf{Y}) \Rightarrow$ **Hazard!!**

$\text{dist}_{\text{dependence}}(\mathbf{i},\mathbf{j}) > \text{dist}_{\text{hazard}}(\mathbf{X},\mathbf{Y}) \Rightarrow$ **Safe**

RAW Hazard Analysis Example

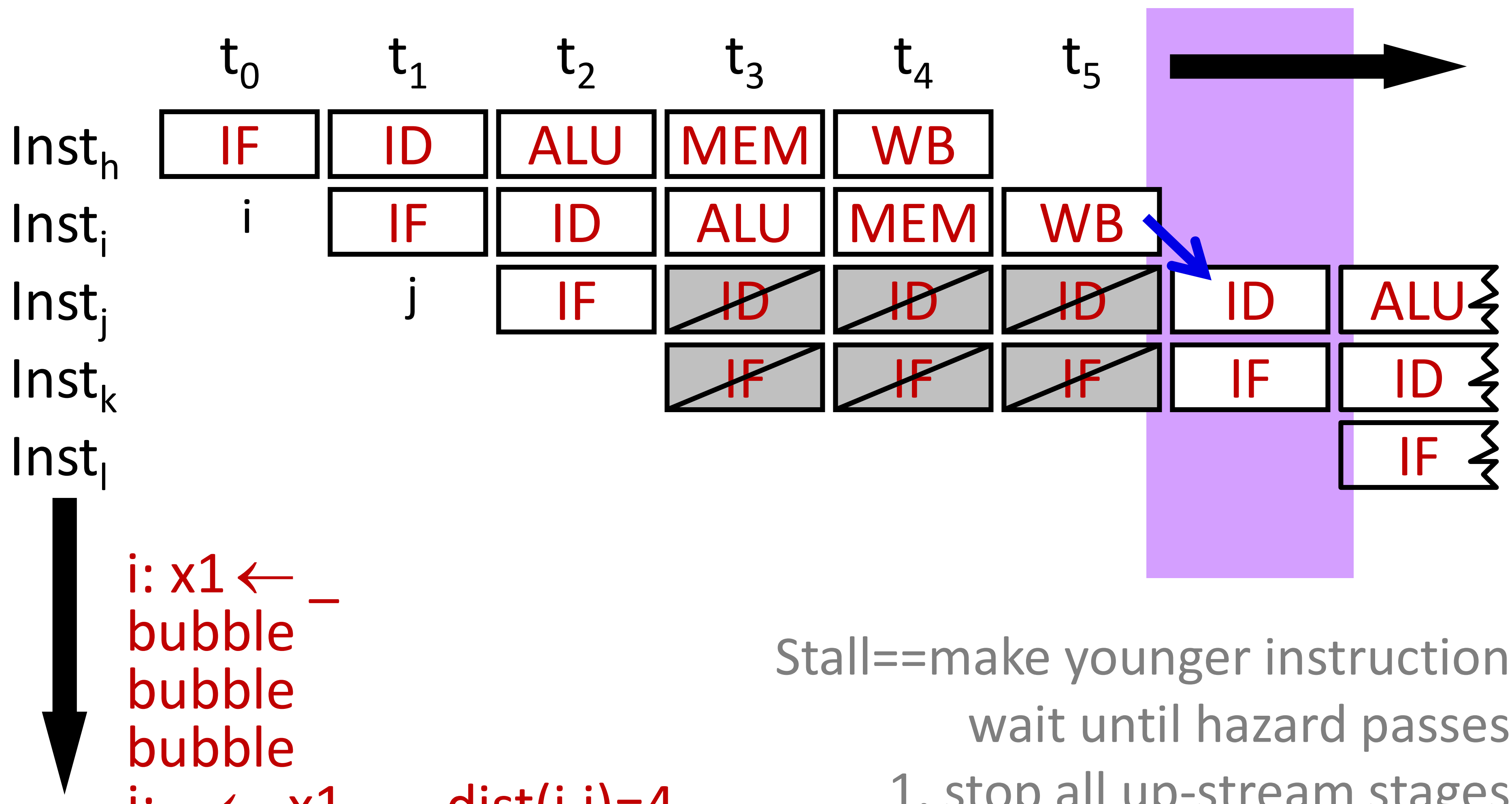
	R/I-Type	LW	SW	Bxx	Jal	Jalr
IF						
ID	**read RF**	**read RF**	**read RF**	**read RF**		**read RF**
EX						
MEM						
WB	**write RF**	**write RF**			**write RF**	**write RF**

- Older **I_A** and younger **I_B** have RAW hazard iff
  - **I_B** (R/I, LW, SW, Bxx or JALR) reads a register written by **I_A** (R/I, LW, or JAL/R)
  - dist(**I_A**, **I_B**) $\leq$ dist(ID, WB) = 3

What about WAW and WAR hazard?

What about memory data hazard?

Pipeline Stall: universal hazard resolution



i: x1 ← _
bubble
bubble
bubble
j: _ ← x1

dist(i,j)=4

Stall==make younger instruction wait until hazard passes

1. stop all up-stream stages
2. drain all down-stream stages

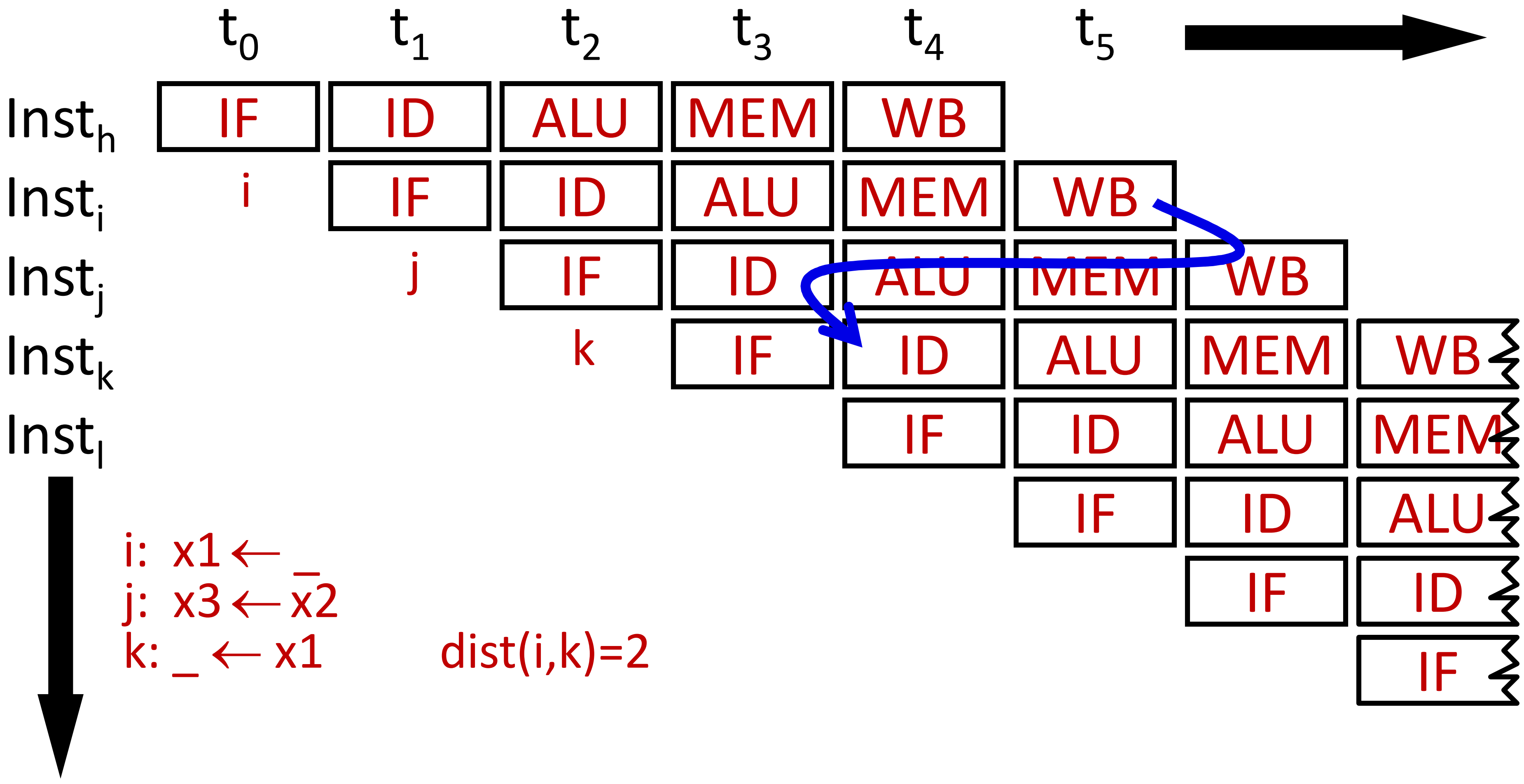
Pipeline Stall

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
IF	i	j	k	k	k	k	l				
ID	h	i	j	j	j	j	k	l			
EX		h	i	bub	bub	bub	j	k	l		
MEM			h	i	bub	bub	bub	j	k	l	
WB				h	i	bub	bub	bub	j	k	l

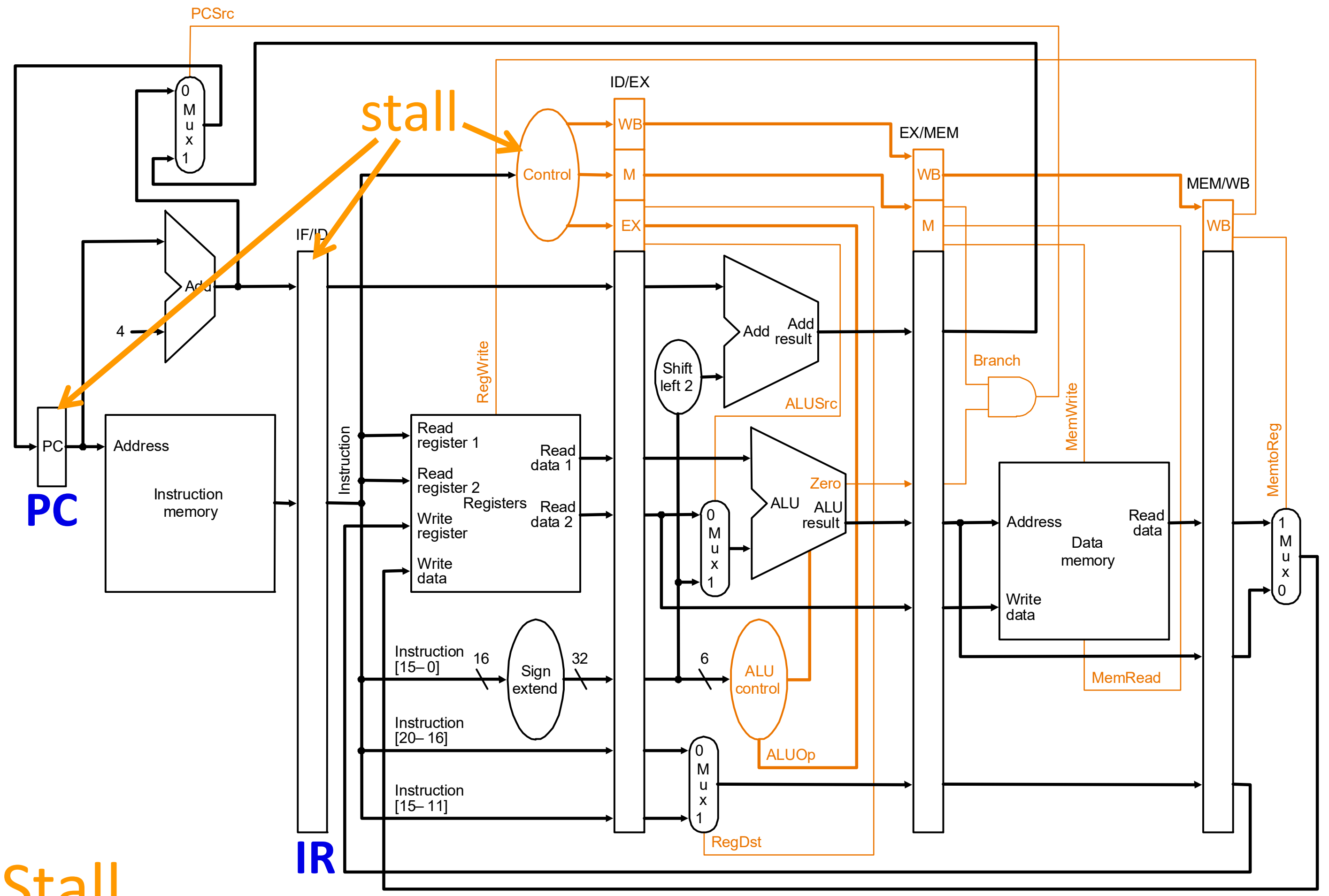
i: x1 ← _

j: _ ← x1

Pop Quiz: What happens in this case?



Stall



- Stall
  - disable **PC** and **IR** latching
  - set $\text{RegWrite}_{\text{ID}}$=0 and $\text{MemWrite}_{\text{ID}}$=0

When to Stall

- Older I_A and younger I_B have RAW hazard iff
  - I_B (R/I, LW, SW, Bxx or JALR) reads a register written by I_A (R/I, LW, or JAL/R)
  - dist(I_A, I_B) $\leq$ dist(ID, WB) = 3

Above is about existence of hazard

- Operationally, to detect hazard in time to prevent:
  - before I_B in ID reads a register, I_B needs to check if any I_A in EX, MEM or WB is going to update it

(if so, value in RF is "stale")

Watch out for x0!!

Stall Condition

- Helper function
  - *useRs1*(I) returns true if I uses rs1
- Stall IF and ID when

$(rs1_{ID}==rd_{EX})$ && $RegWrite_{EX}$ && $useRs1(\mathbf{IR_{ID}})$ && $rs1_{ID}!=x0$ or

$(rs1_{ID}==rd_{MEM})$ && $RegWrite_{MEM}$ && $useRs1(\mathbf{IR_{ID}})$ && $rs1_{ID}!=x0$ or

$(rs1_{ID}==rd_{WB})$ && $RegWrite_{WB}$ && $useRs1(\mathbf{IR_{ID}})$ && $rs1_{ID}!=x0$ or

$(rs2_{ID}==rd_{EX})$ && $RegWrite_{EX}$ && $useRs2(\mathbf{IR_{ID}})$ && $rs2_{ID}!=x0$ or

$(rs2_{ID}==rd_{MEM})$ && $RegWrite_{MEM}$ && $useRs2(\mathbf{IR_{ID}})$ && $rs2_{ID}!=x0$ or

$(rs2_{ID}==rd_{WB})$ && $RegWrite_{WB}$ && $useRs2(\mathbf{IR_{ID}})$ && $rs2_{ID}!=x0$

It is crucial that EX, MEM and WB continue to advance during stall

Impact of Stall on Performance

- Each stall cycle corresponds to 1 lost ALU cycle
- A program with N instructions and S stall cycles:

  average IPC=N/(N+S)

- S depends on
  - frequency of hazard-causing dependencies
  - distance between hazard-causing instruction pairs
  - distance between hazard-causing dependencies

    (suppose i_1,i_2 and i_3 all depend on i_0, once i_1's hazard is resolved by stalling, i_2 and i_3 do not stall)

Sample Assembly [P&H]

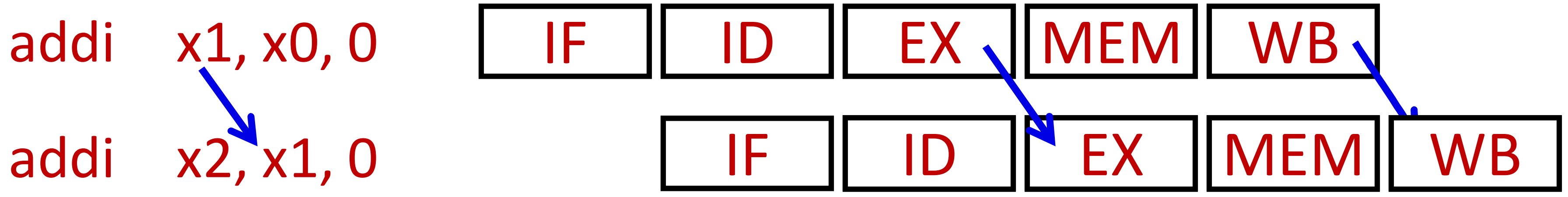
for (j=i-1; j>=0 && v[j] > v[j+1]; j-=1) { ...... }

```
            addi   $s1, $s0, -1                3 stalls
for2tst:    slti   $t0, $s1, 0
                                               3 stalls
            bne    $t0, $zero, exit2
            sll    $t1, $s1, 2                 3 stalls
            add    $t2, $a0, $t1
                                               3 stalls
            lw     $t3, 0($t2)
            lw     $t4, 4($t2)                 3 stalls
            slt    $t0, $t4, $t3
                                               3 stalls
            beq    $t0, $zero, exit2
            .........
            addi   $s1, $s1, -1
            j      for2tst
exit2:
```

Data Forwarding (or Register Bypassing)

- What does "ADD r_x r_y r_z" mean? Get inputs from RF[r_y] and RF[r_z] and put result in RF[r_x]?
- But, RF is just a part of an abstraction
  - a way to connect dataflow between instructions

    "operands to ADD are resulting **values** of the last instructions to assign to RF[r_y] and RF[r_z]"
  - RF doesn't have to exist/behave as a literal object!!!
- If only dataflow matters, don't wait for WB . . .



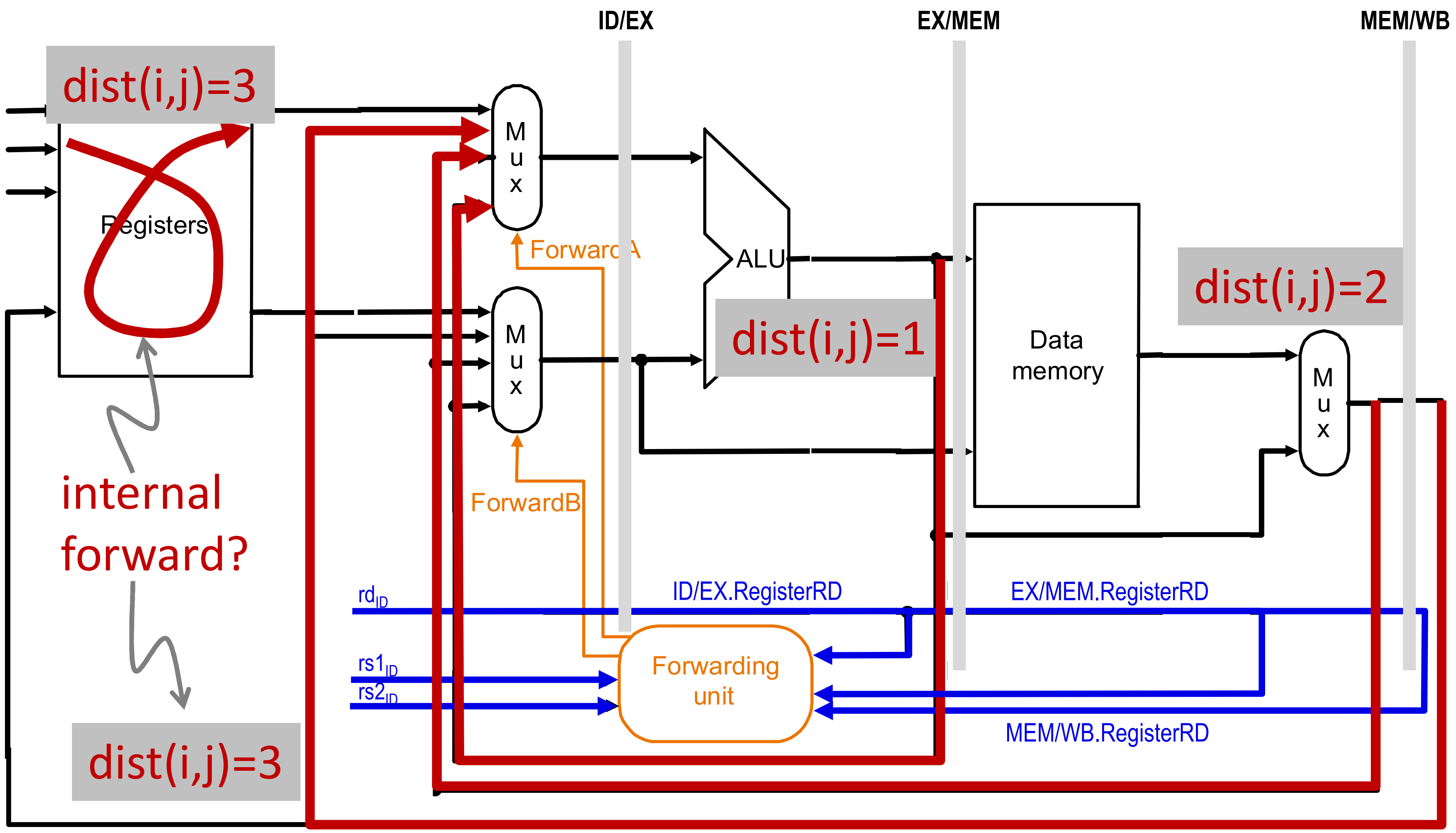
```
addi    x1, x0, 0
addi    x2, x1, 0
```

Resolving RAW Hazard by Forwarding

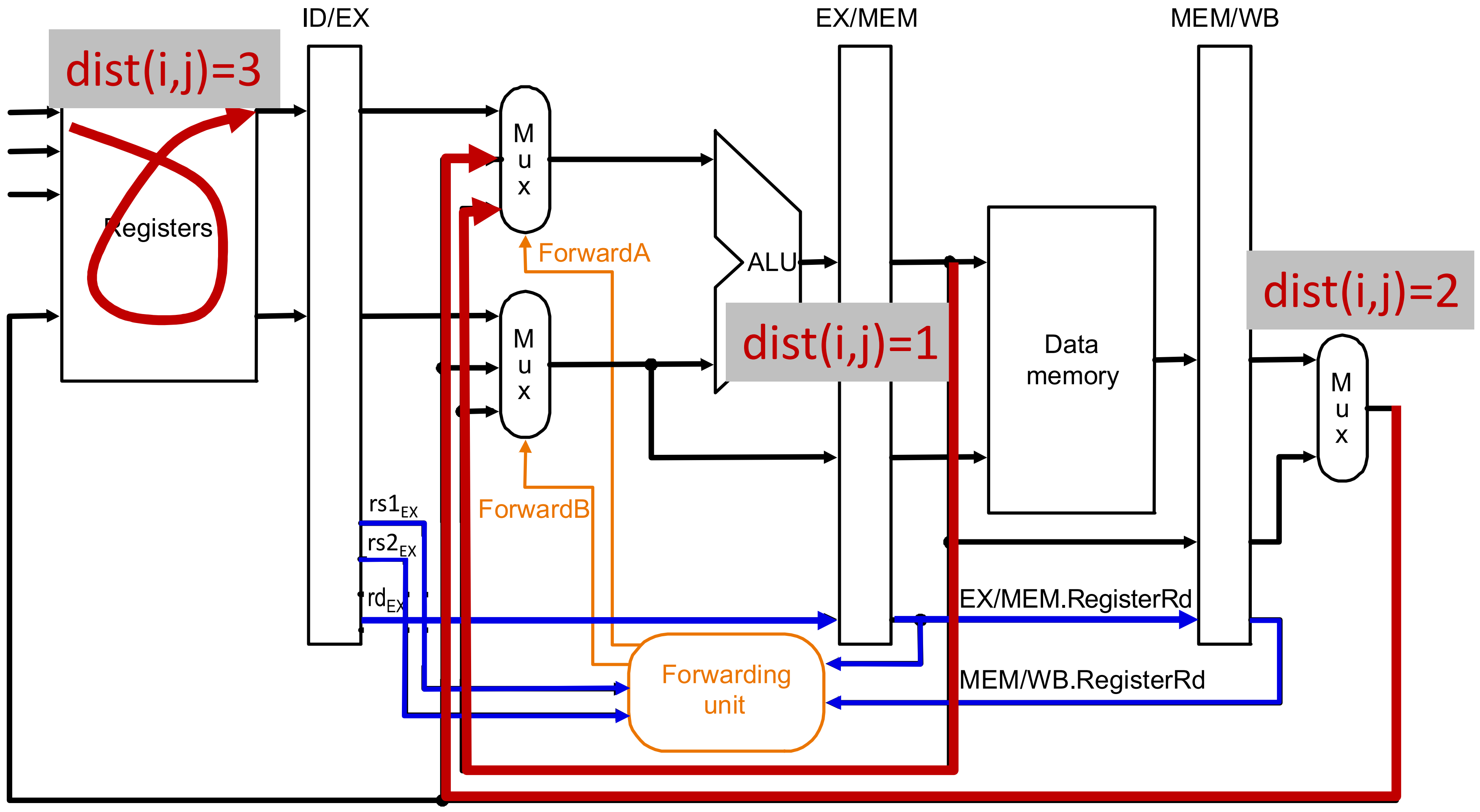
A hazard exits

- Older I_A and younger I_B have RAW hazard iff
  - I_B (R/I, LW, SW, Bxx or JALR) reads a register written by I_A (R/I, LW, or JAL/R)
  - dist(I_A, I_B) $\leq$ dist(ID, WB) = 3
- To detect hazard in time to prevent, before I_B in ID reads a register, I_B needs to check if any I_A in EX, MEM or WB is going to update it
- Before: I_B need to stall for I_A to update RF
- Now: I_B need to stall for I_A to produce result
  - retrieve I_A result from datapath when ready
  - retrieve **youngest** if multiple "apparent" hazards

Forwarding Paths (v1)



Forwarding Paths (v2)



better if EX is the fastest stage

Forwarding Logic (for v1)

```
if ($rs1_{ID}$!=0) && ($rs1_{ID}$==$rd_{EX}$) && $RegWrite_{EX}$ then
    forward writeback value from EX                // dist=1
else if ($rs1_{ID}$!=0) && ($rs1_{ID}$==$rd_{MEM}$) && $RegWrite_{MEM}$ then
    forward writeback value from MEM               // dist=2
else if ($rs1_{ID}$!=0) && ($rs1_{ID}$==$rd_{WB}$) && $RegWrite_{WB}$ then
    forward writeback value from WB                // dist=3
else
    use $A_{ID}$                                   // dist > 3
```

Must prioritize young-to-old

Why doesn't *useRs1*( ) appear?

Isn't it bad to forward from LW in EX?

Data Hazard Analysis (with Forwarding)

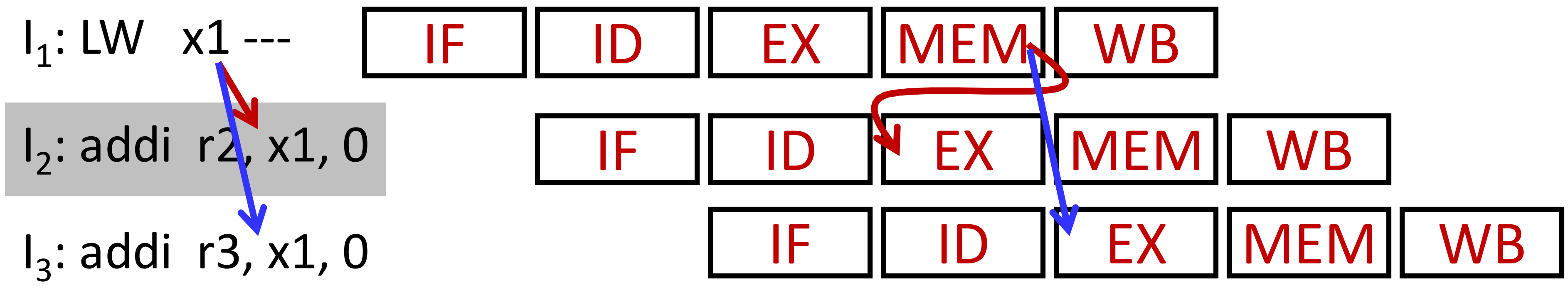
	R/I-Type	LW	SW	Bxx	Jal	Jalr
IF						
ID					(produce)	(produce)
EX	use produce	use	use	use	produce	use produce
MEM		produce	(use)			
WB						

- Even with forwarding, RAW dependence on immediate preceding LW results in hazard

$$\text{Stall}=\{[rs1_{ID}==rd_{EX}\ \&\&\ \mathit{useRs1}(\mathbf{IR}_{ID})\ \&\&\ rs1_{ID}!=0]\ ||$$

$$[rs2_{ID}==rd_{EX}\ \&\&\ \mathit{useRs2}(\mathbf{IR}_{ID})]\ \&\&\ rs2_{ID}!=0]\ \}\ \&\&\ \overbrace{\text{MemRead}_{EX}}^{\text{i.e., } op_{EX}=Lx}$$

Historical: MIPS Load "Delay Slot"



- R2000 defined LW with arch. latency of 1 inst
  - invalid for I_2 (in LW's delay slot) to ask for LW's result
  - any dependence on LW at least distance 2
- Delay slot vs dynamic stalling
  - fill with an independent instruction (no difference)
  - if not, fill with a NOP (no difference)
- Can't lose on 5-stage . . . good idea?

Hint: 1. non-atomic instruction; 2. μarch influence

Sample Assembly [P&H]

for (j=i-1; j>=0 && v[j] > v[j+1]; j-=1) { …… }

```
          addi   $s1, $s0, -1
for2tst:  slti   $t0, $s1, 0
          bne    $t0, $zero, exit2
          sll    $t1, $s1, 2
          add    $t2, $a0, $t1
          lw     $t3, 0($t2)
          lw     $t4, 4($t2)
          slt    $t0, $t4, $t3
          beq    $t0, $zero, exit2
          .........
          addi   $s1, $s1, -1
          j      for2tst
exit2:
```

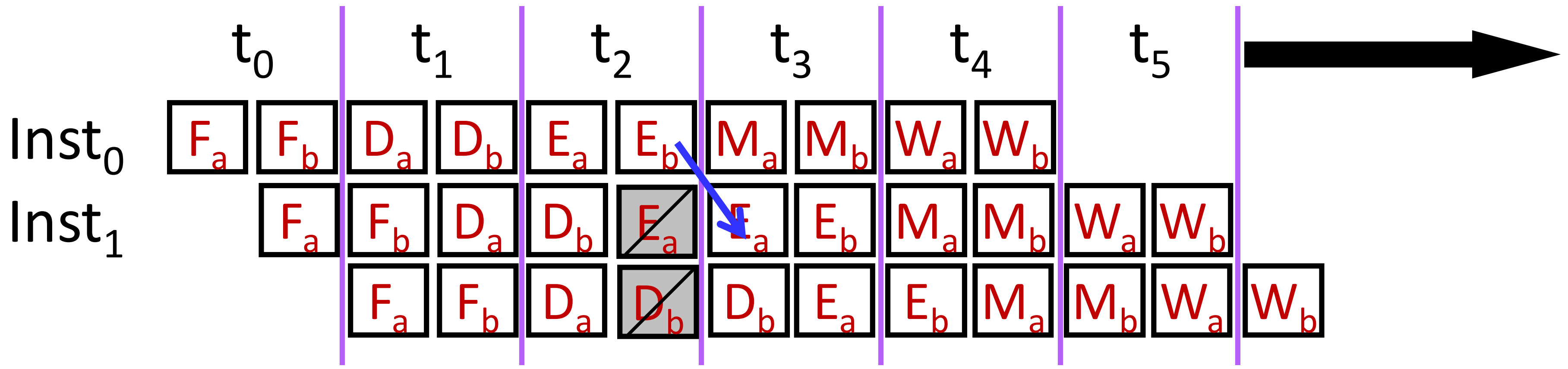
1 stall or
1 nop (MIPS)

Why not very deep pipelines?

- With only 5 stages, still plenty of combinational logic between registers
- "Superpipelining" $\Rightarrow$ increase pipelining such that even intrinsic operations (e.g. ALU, RF access, memory access) require multiple stages
- What's the problem?

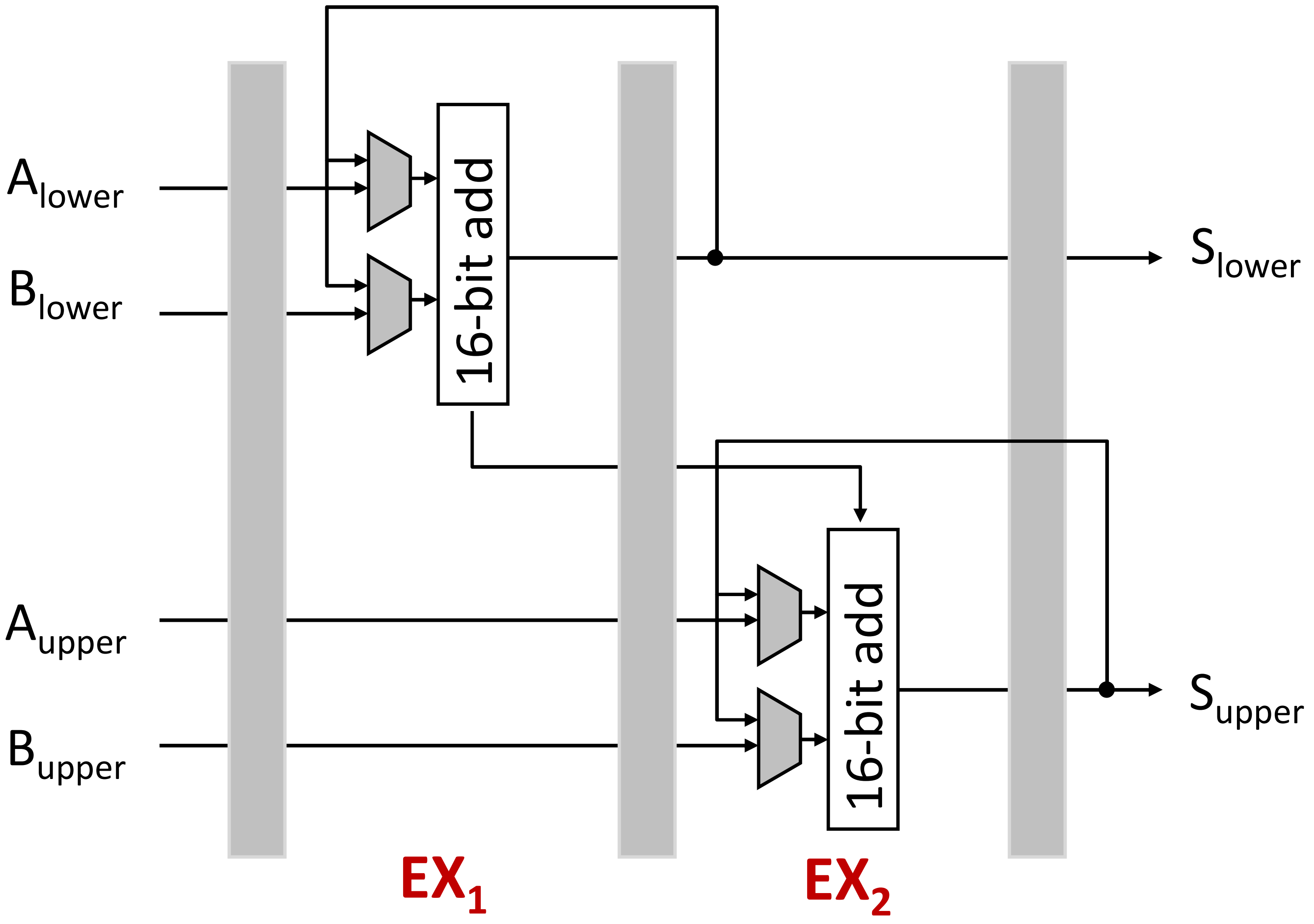
Inst$_0$: addi x1, x0, 0

Inst$_1$: addi x2, x1, 0



t_0
t_1
t_2
t_3
t_4
t_5
Inst$_0$
Inst$_1$
F_a F_b D_a D_b E_a E_b M_a M_b W_a W_b
F_a F_b D_a D_b E_a E_a E_b M_a M_b W_a W_b
F_a F_b D_a D_b D_b E_a E_b M_a M_b W_a W_b

Aside: Intel P4's Superpipelined Adder



32-bit addition pipelined over 2 stages, BW=1/latency$_{16\text{-bit-add}}$

No stall between back-to-back dependencies

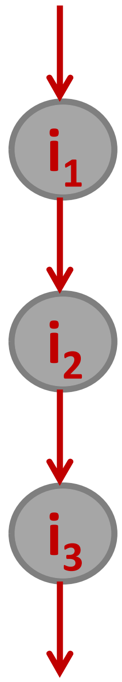
Terminology

- Dependency
  - property of program
  - ordering requirement between instructions
- Pipeline Hazard:
  - property of uarch when interacting with program
  - (potential) violation of dependencies in program
- Hazard Resolution:
  - static $\Rightarrow$ schedule instructions at compile time to avoid hazards
  - dynamic $\Rightarrow$ detect hazard and adjust pipeline operation

Stall, Flush or Forward

Dependencies and Pipelining (architecture vs. microarchitecture)

Sequential and atomic instruction semantics



Defines what is correct; doesn't say do it this way

True dependence between two instructions may only require ordering of certain sub-operations