Lecture 8:
Pipelined CPU Implementation: Hazards and Resolutions

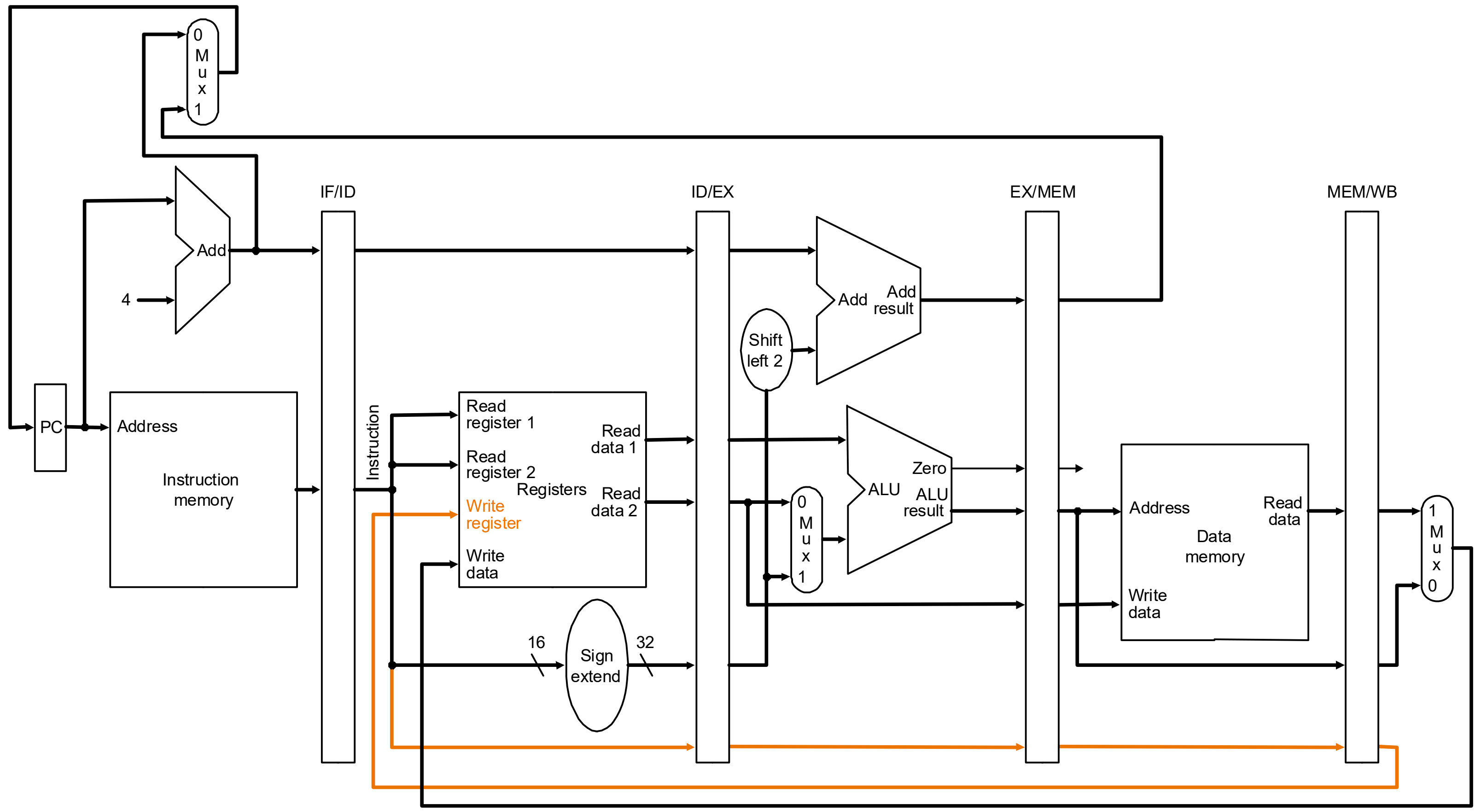
Jangwoo Kim (Seoul National University)
jangwoo@snu.ac.kr

Announcement

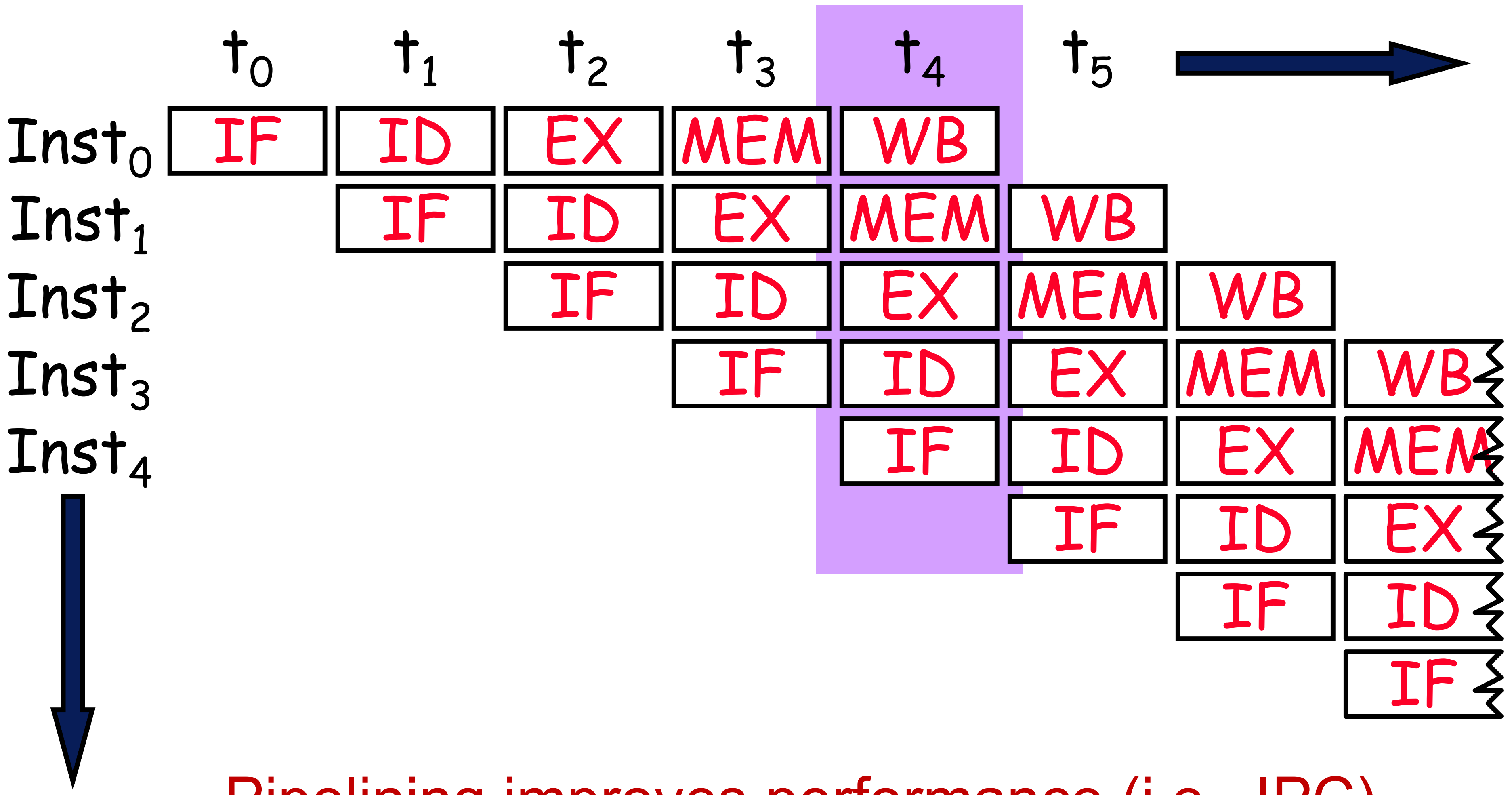
- Homework #2 posted
  - **Due: 4/12**

- Mid-term Exam
  - **Mid-term 4/20 (Friday, 6:30pm)**

Review: Pipelined Operation (1/2)



Review: Pipelined Operation (2/2)



Pipelining improves performance (i.e., IPC)

Reality of Instruction Pipelining

- No identical operations

  ⇒ Unify instruction types *e.g., R-type, I-type & J-type*

  - Combine instruction types to flow through "multi-function" pipe

- No uniform sub-operations

  ⇒ Balance pipeline stages *e.g., Memory read VS. addition?*

  - Stage-latency calculation to make balanced stages

Major problem!

e.g., Reg. read & write in 1 clock?
Waiting data to be produced?

- No independent operations

  ⇒ Remove dependency and/or busy resources

  - Duplicate contended resources
  - Inter-instruction dependency detection and resolution

So, MIPS ISA was made for improved pipelineability.

Data Dependence

Today's topic!

Data dependence

$$\begin{tikzcd} r_3 & \leftarrow & r_1 & \text{op} & r_2 \\ r_5 & \leftarrow & r_3 \arrow[ull] & \text{op} & r_4 \end{tikzcd}$$

Read-after-Write
(RAW)

Anti-dependence

$$\begin{tikzcd} r_3 & \leftarrow & r_1 & \text{op} & r_2 \\ r_1 \arrow[urr] & \leftarrow & r_4 & \text{op} & r_5 \end{tikzcd}$$

Write-after-Read
(WAR)

Later…

Output-dependence

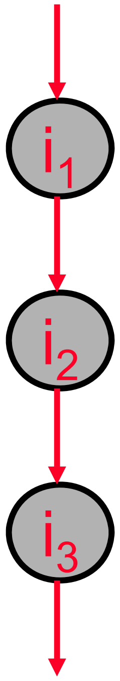
$$\begin{tikzcd} r_3 & \leftarrow & r_1 & \text{op} & r_2 \\ r_5 & \leftarrow & r_3 & \text{op} & r_4 \\ r_3 \arrow[uu, bend left] & \leftarrow & r_6 & \text{op} & r_7 \end{tikzcd}$$

Write-after-Write
(WAW)

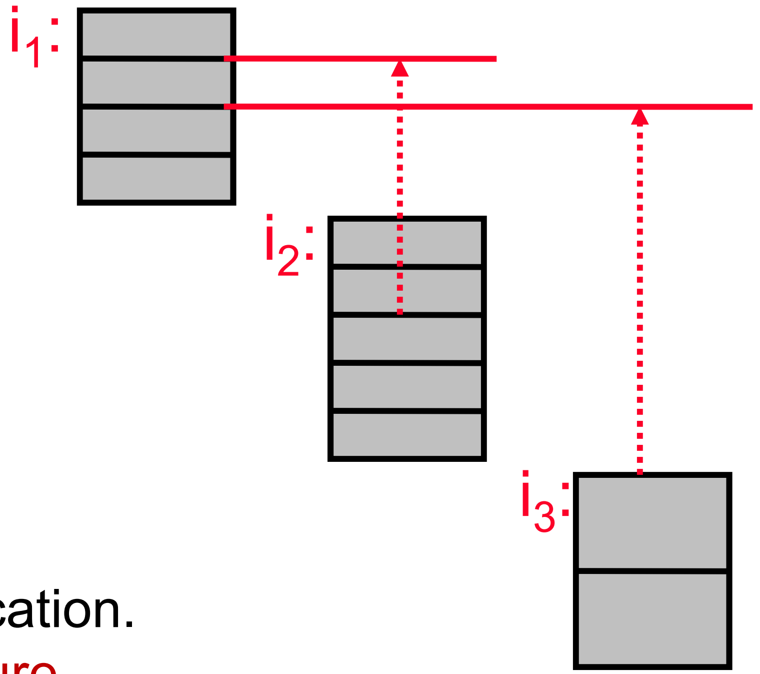
We discuss control-flow dependence in a later lecture

Dependencies and Pipelined Execution

Sequential and **Atomic** instruction semantics



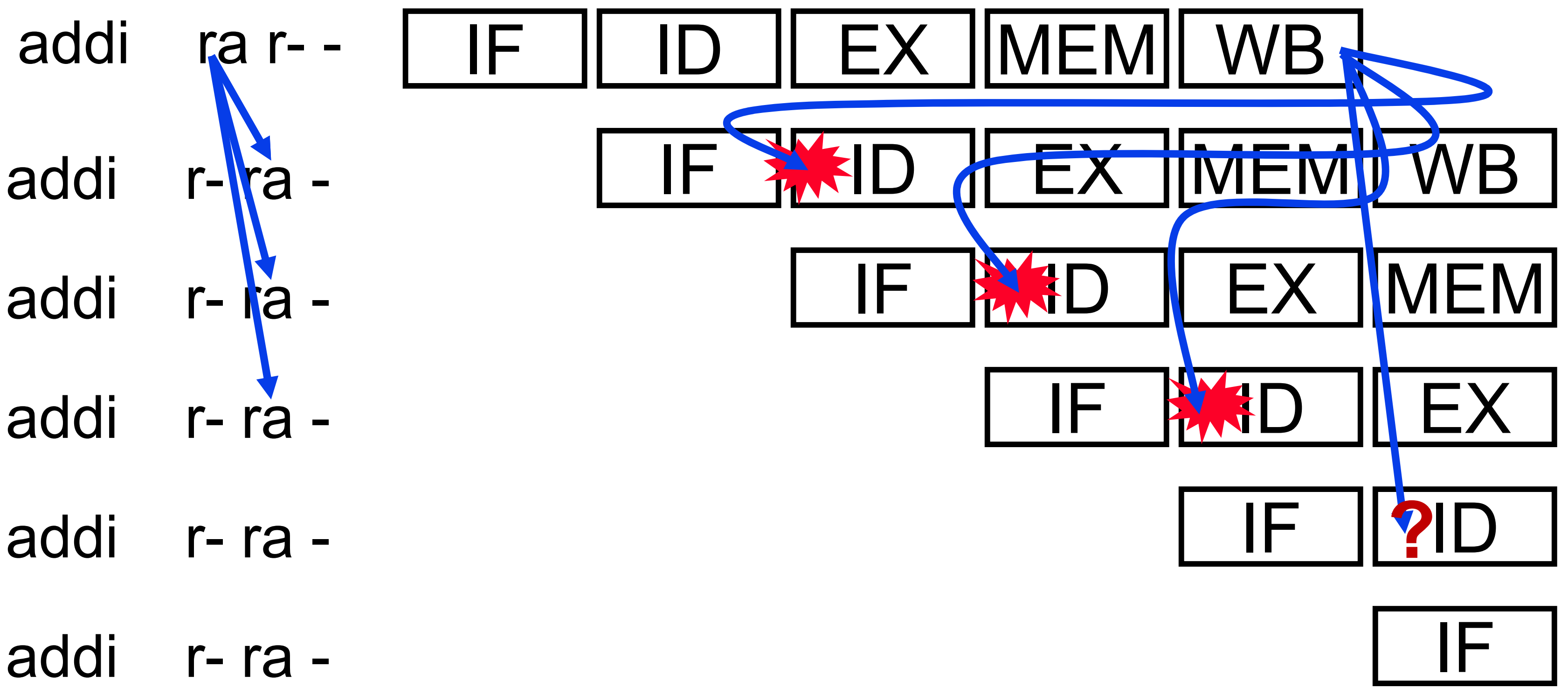
The true dependency between two instructions may only require ordering of certain sub-operations



This semantics is a "too simplified" specification.
It defines what is correct. ← Architecture
NOT how it is implemented. ← Microarchitecture

RAW Dependency and Hazard

- RAW dependencies lead to hazards in the 5-stage pipeline

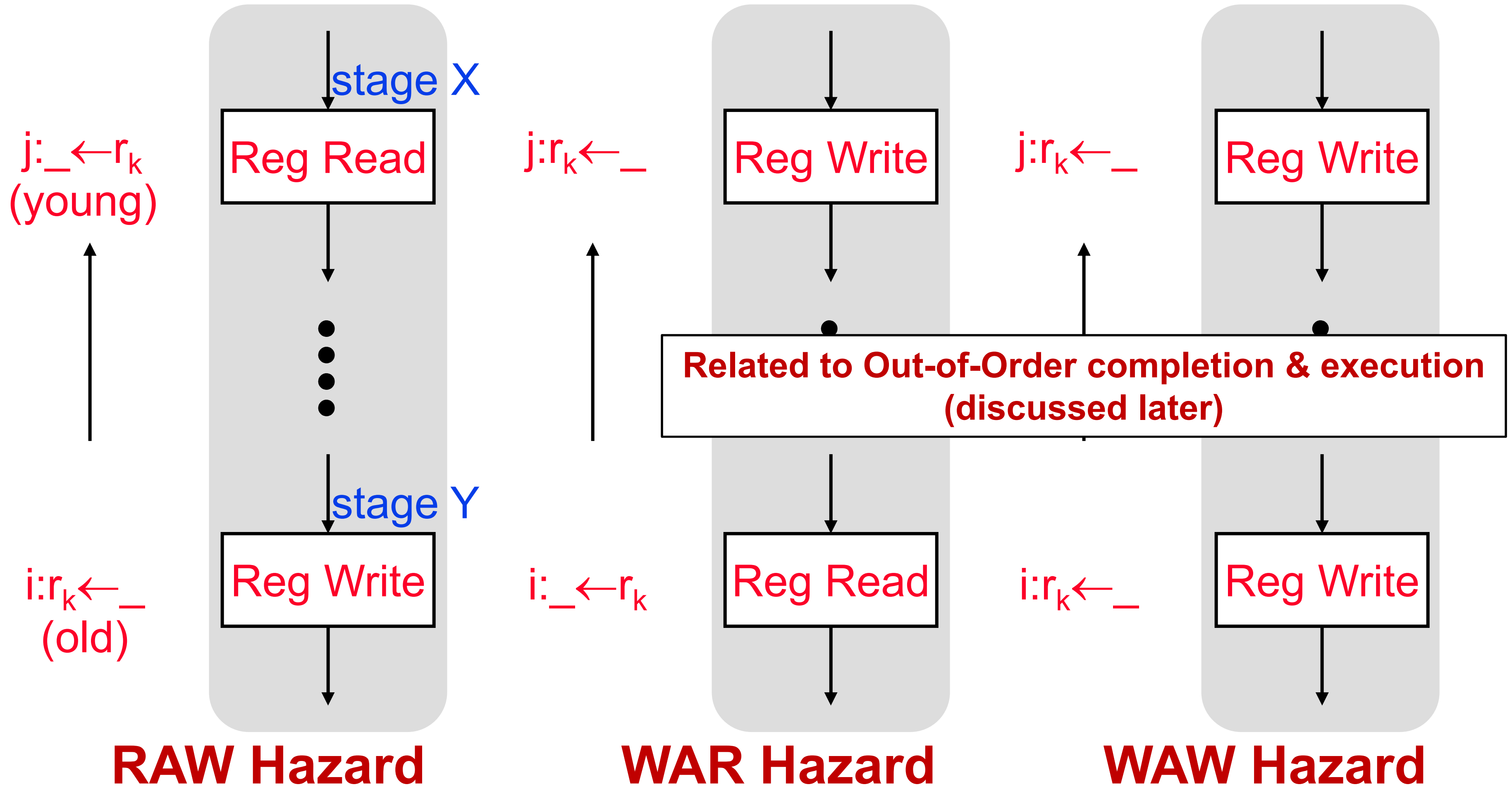


Register Data Hazard Analysis

	R/I-Type	LW	SW	Br	J	Jr
IF						
ID	read RF	read RF	read RF	read RF		read RF
EX						
MEM						
WB	write RF	write RF				

- When does a register data hazard occur between two data dependent instructions?
  - Dependence type?
  - Instruction types involved?
  - Distance between the two instructions?

Necessary Condition for Hazards



Related to Out-of-Order completion & execution (discussed later)

dist(i,j) $\le$ dist(X,Y) $\Rightarrow$ Hazard!!
dist(i,j) $>$ dist(X,Y) $\Rightarrow$ Safe

RAW Hazard Analysis Example

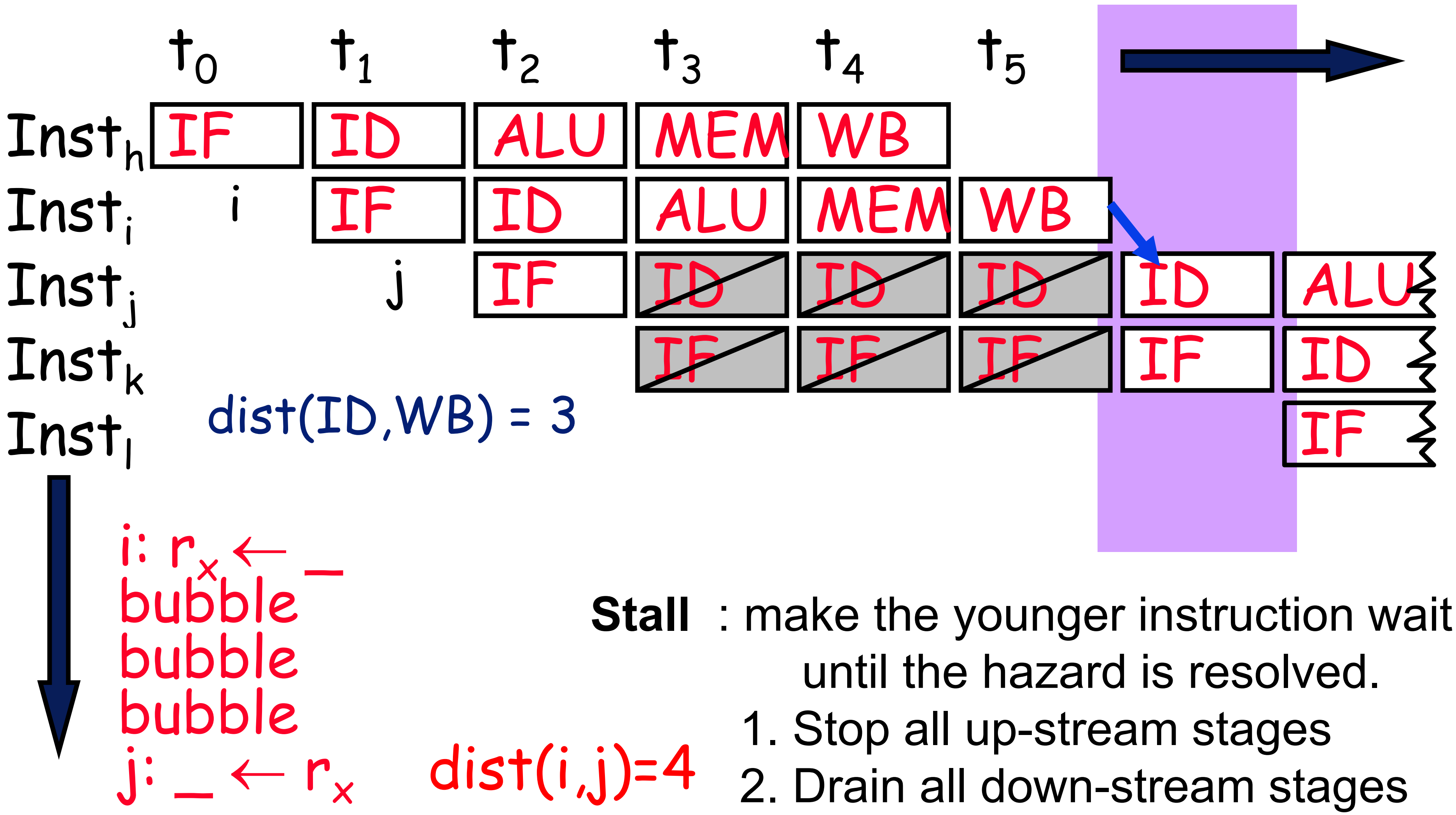
	R/I-Type	LW	SW	Br	J	Jr
IF						
ID	read RF	read RF	read RF	read RF		read RF
EX						
MEM						
WB	write RF	write RF				

- Instructions I_A and I_B (where I_A comes before I_B) have RAW hazard iff
  - (1) I_B (R/I, LW, SW, Br or JR) reads a register written by I_A (R/I or LW)
  - (2) dist(I_A, I_B) $\leq$ dist(ID, WB) = 3

What about WAW and WAR hazard?

What about memory data hazard?

A universal hazard resolution: "Pipeline Stall" (= let's wait)



Stall : make the younger instruction wait until the hazard is resolved.

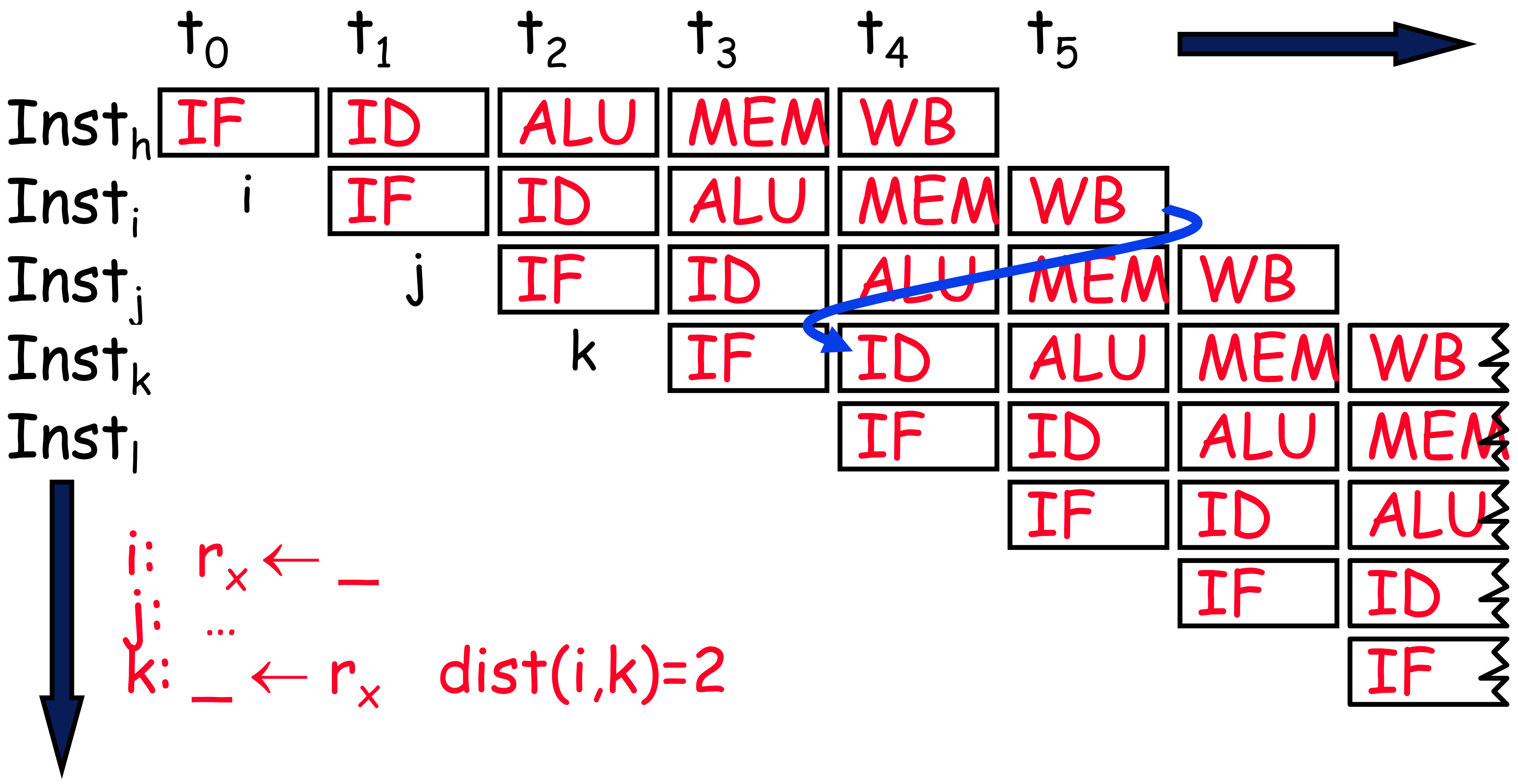
1. Stop all up-stream stages
2. Drain all down-stream stages

Pipeline Stall

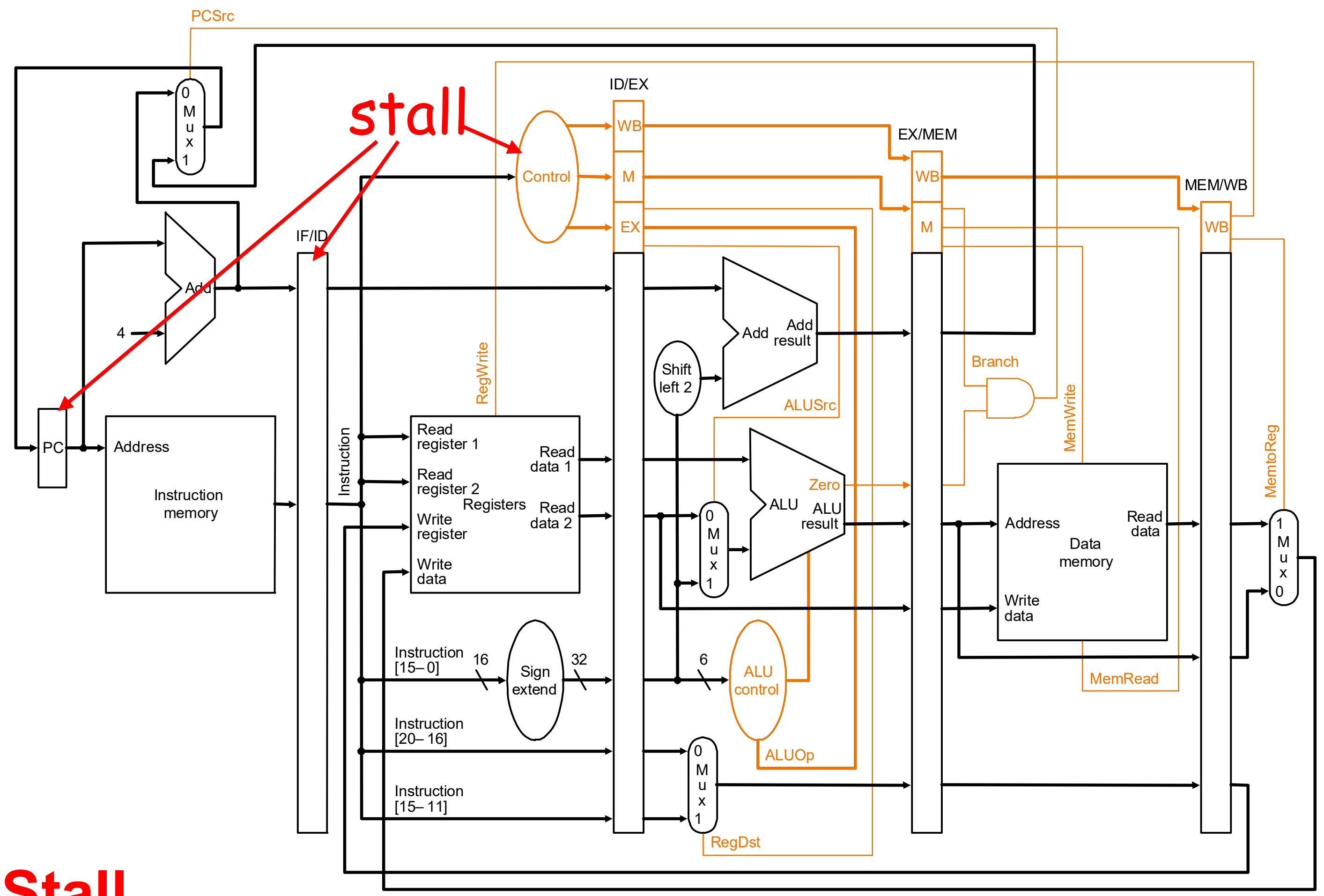
	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
IF	i	j	k	k	k	k	l				
ID	h	i	j	j	j	j	k	l			
EX		h	i	bub	bub	bub	j	k	l		
MEM			h	i	bub	bub	bub	j	k	l	
WB				h	i	bub	bub	bub	j	k	l

i: rx ← _
j: _ ← rx

How to Stall?


t_0
t_1
t_2
t_3
t_4
t_5
$Inst_h$
$Inst_i$
$Inst_j$
$Inst_k$
$Inst_l$
IF ID ALU MEM WB
i
j
k
$i: \; r_x \leftarrow \_$
$j: \; \ldots$
$k: \; \_ \leftarrow r_x$
dist(i,k)=2

How to Stall?



- **Stall**
  - Disable **PC** and **IR** latching
  - Control should set RegWrite=0 and MemWrite=0

When to Stall?

- Instructions I_A and I_B (where I_A comes before I_B) have RAW hazard iff
  - I_B (R/I, LW, SW, Br or JR) reads a register written by I_A (R/I or LW)
  - Dist(I_A, I_B) $\leq$ dist(ID, WB) = 3

- In other words, we must stall I_B in ID stage if it wants to read a register **to be written** by **ALL** I_A that might exist in EX, MEM or WB stage

Stall Condition

- Helper functions
  - rs(I) returns the rs field of I (instruction register)
  - use_rs(I) returns true if I requires RF[rs] and rs!=r0
- Stall when
  - (rs(IR_{ID})==$dest_{EX}$) && use_rs(IR_{ID}) && $RegWrite_{EX}$ or
  - (rs(IR_{ID})==$dest_{MEM}$) && use_rs(IR_{ID}) && $RegWrite_{MEM}$ or
  - (rs(IR_{ID})==$dest_{WB}$) && use_rs(IR_{ID}) && $RegWrite_{WB}$ or

  - (rt(IR_{ID})==$dest_{EX}$) && use_rt(IR_{ID}) && $RegWrite_{EX}$ or
  - (rt(IR_{ID})==$dest_{MEM}$) && use_rt(IR_{ID}) && $RegWrite_{MEM}$ or
  - (rt(IR_{ID})==$dest_{WB}$) && use_rt(IR_{ID}) && $RegWrite_{WB}$

It is crucial that the EX, MEM and WB stages continue to advance as normal during these stall cycles

Impact of Stall on Performance

- Without stall

  Ideal IPC = 1

- With stall
  - Each stall cycle corresponds to 1 lost cycle
  - For a program with N instructions and S stall cycles,

    Average IPC (with stall) =N/(N+S)

- S depends on
  - Frequency of RAW hazards
  - Exact distance between the hazard-causing instructions
  - Distance between hazards

    Suppose i_1,i_2 and i_3 all depend on i_0, once i_1's hazard is resolved, i_2 and i_3 must be okay too!

Sample Assembly

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:
```

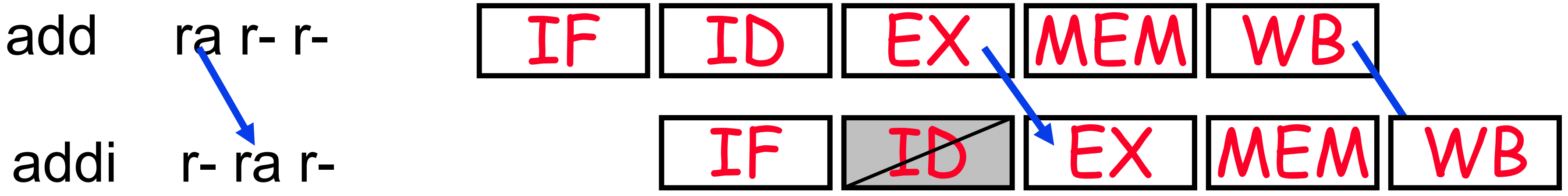
Data Forwarding (=Register Bypassing)

- It is intuitive to think of RF as 'flow of right data'
  - "add rx ry rz" literally means get values from RF[ry] and RF[rz] respectively and put result in RF[rx]
  - So, "add rx ry rz" really means

  (1) Get "the computation results" of the last instruction to define the values of RF[ry] and RF[rz], respectively, and

  (2) Until another instruction redefines RF[rx], younger instructions that refers to RF[rx] can use this instruction's result

- What really matters is to maintain the correct "dataflow" between operations, thus



Resolving RAW Hazard by Forwarding

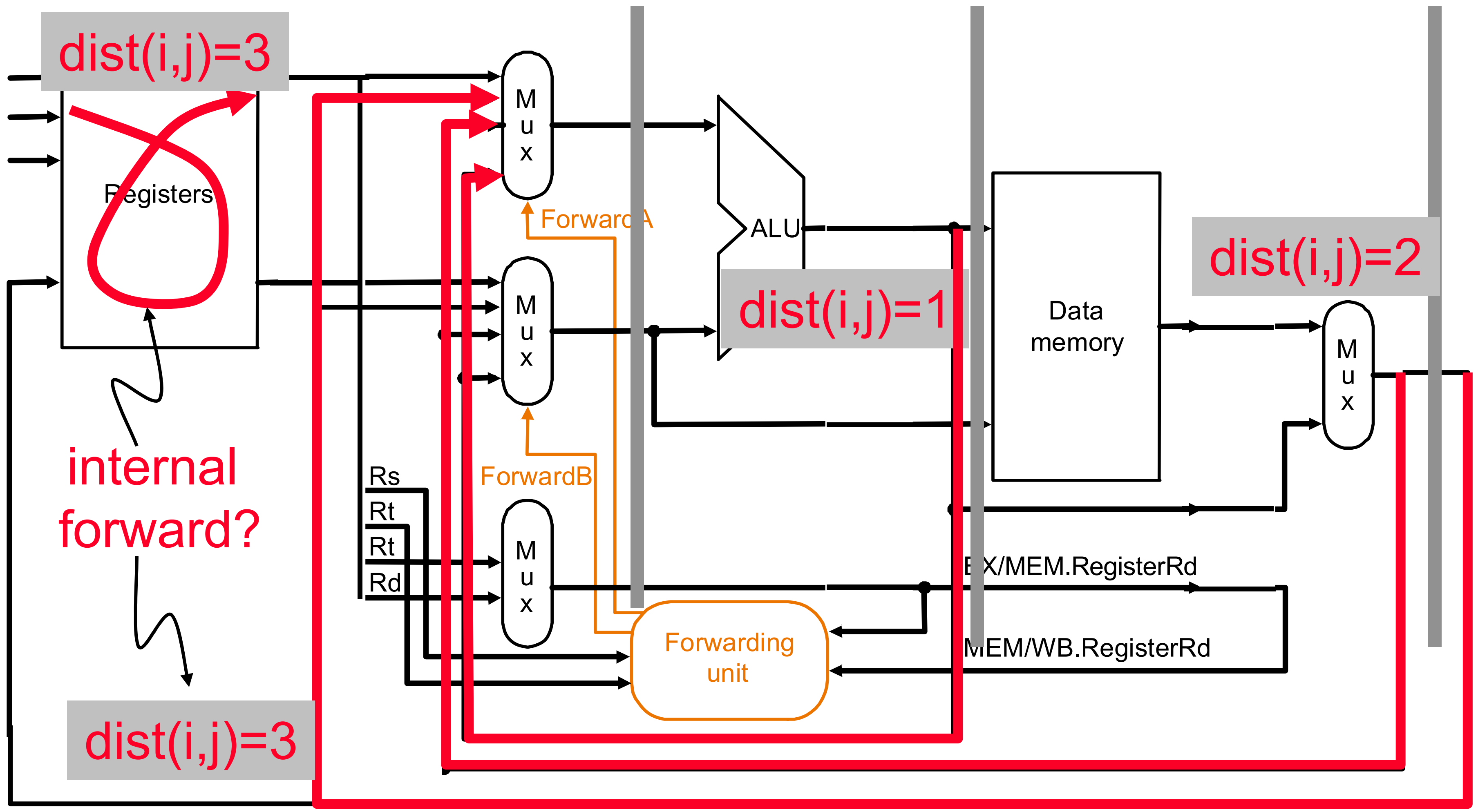
- Instructions I_A and I_B (where I_A comes before I_B) have RAW hazard iff
  - I_B (R/I, LW, SW, Br or JR) reads a register written by I_A (R/I or LW)
  - $dist(I_A, I_B) \leq dist(ID, WB) = 3$

- In other words, if I_B in ID stage reads a register written by I_A in EX, MEM or WB stage, then the operand required by I_B is not yet in RF

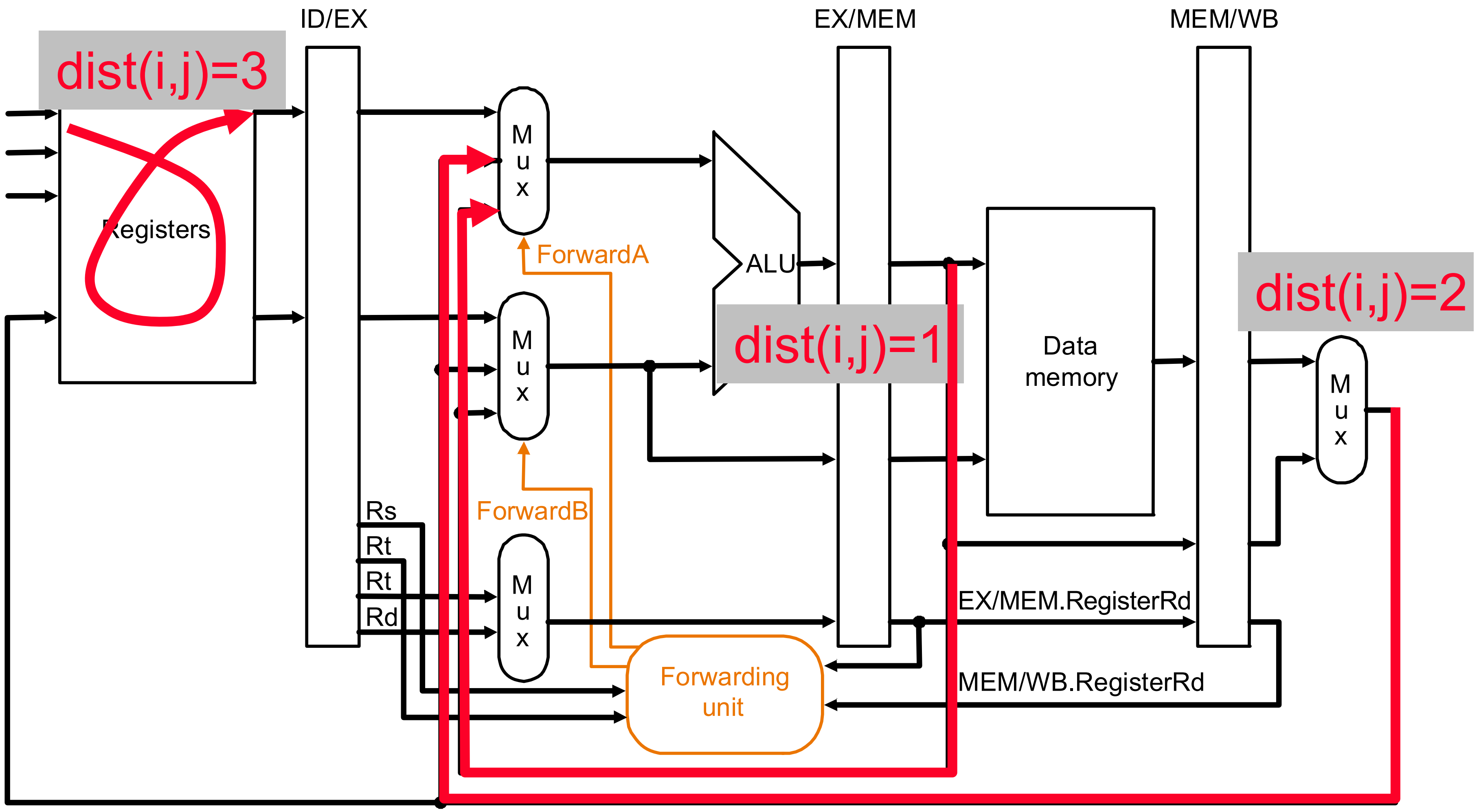
  → Retrieve operand **"from datapath"** instead of waiting until the RF is updated!

  → Retrieve operand **"from the youngest definition"** if multiple definitions are outstanding!

Forwarding Paths (v1)



Forwarding Paths (v2)



b. With forwarding

Assumes RF forwards internally

Forwarding Logic (for v2)

```
if ($rs_{EX}$!=0) && ($rs_{EX}$==$dest_{MEM}$) && $RegWrite_{MEM}$ then
    forward operand from MEM stage      // dist=1
else if ($rs_{EX}$!=0) && ($rs_{EX}$==$dest_{WB}$) && $RegWrite_{WB}$ then
    forward operand from WB stage       // dist=2
else
    use the operand from register file  // dist >= 3
```

Ordering matters!! Must check the youngest match first!

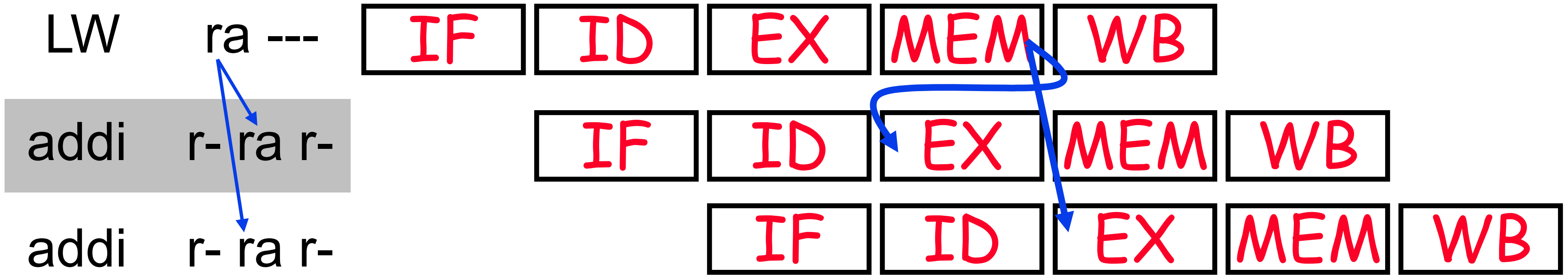
Why doesn't use_rs( ) appear in the forwarding logic?

Data Hazard Analysis (with Forwarding)

	R/I-Type	LW	SW	Br	J	Jr
IF						
ID						use
EX	use produce	use	use	use		
MEM		produce	(use)			
WB						

- Even with data-forwarding, RAW dependence on an immediate preceding LW instruction produces a hazard

Load Delay Slot



- R2000 defined load with arch. latency of 1 inst
  - The instruction immediately following a load (in the "delay slot") still sees the old register value (this is the behavior if we don't do anything special beyond forwarding)
  - ISA feature tailored to the 5-stage pipelined microarchitecture WARNING. microarchitecure exposed to outside.
- If loads are defined normally, i.e., atomic
  - A dependent immediate successor to LW must stall 1 cycle in ID
  - Stall = (rs($\mathbf{IR}_{\mathbf{ID}}$)==$\text{dest}_{\text{EX}}$) && use_rs($\mathbf{IR}_{\mathbf{ID}}$) && $\text{MemRead}_{\text{EX}}$

Sample Assembly

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)
          nop
          slt    $t0, $t4, $t3
          beq    $t0, $zero, exit2
          .........
          addi   $s1, $s1, -1
          j      for2tst
exit2:
```

Terminology

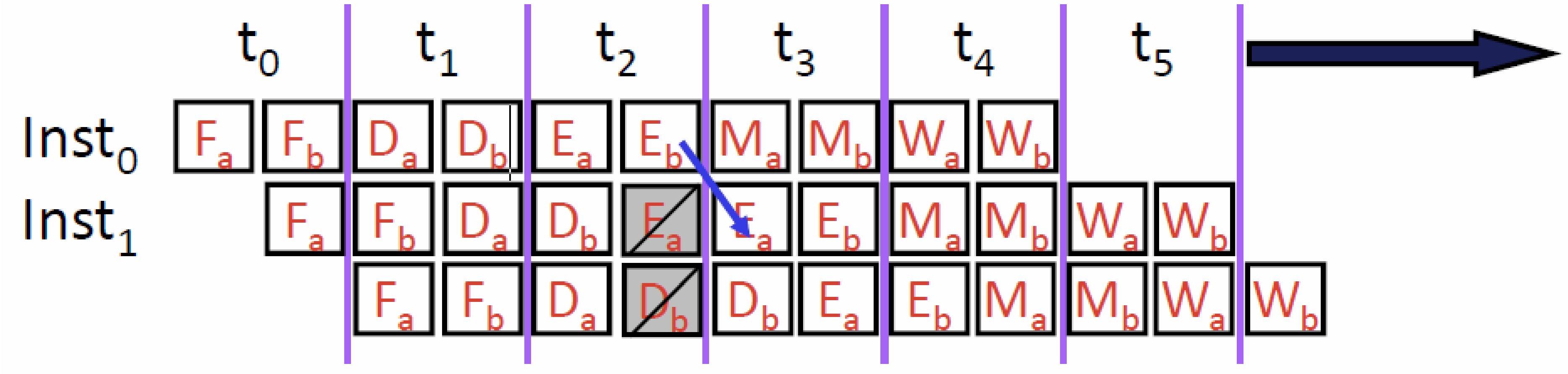
- Dependencies
  - Ordering requirement between instructions
- Pipeline Hazards:
  - (Potential) violations of dependencies
- Hazard Resolution:
  - **Static** → Schedule instructions at compile time to avoid hazards
  - **Dynamic** → Detect hazard and adjust pipeline operation

    Stall, Flush or Forward
- Pipeline Interlock:
  - Hardware mechanisms for dynamic hazard resolution
  - Detect and enforce dependences at run time

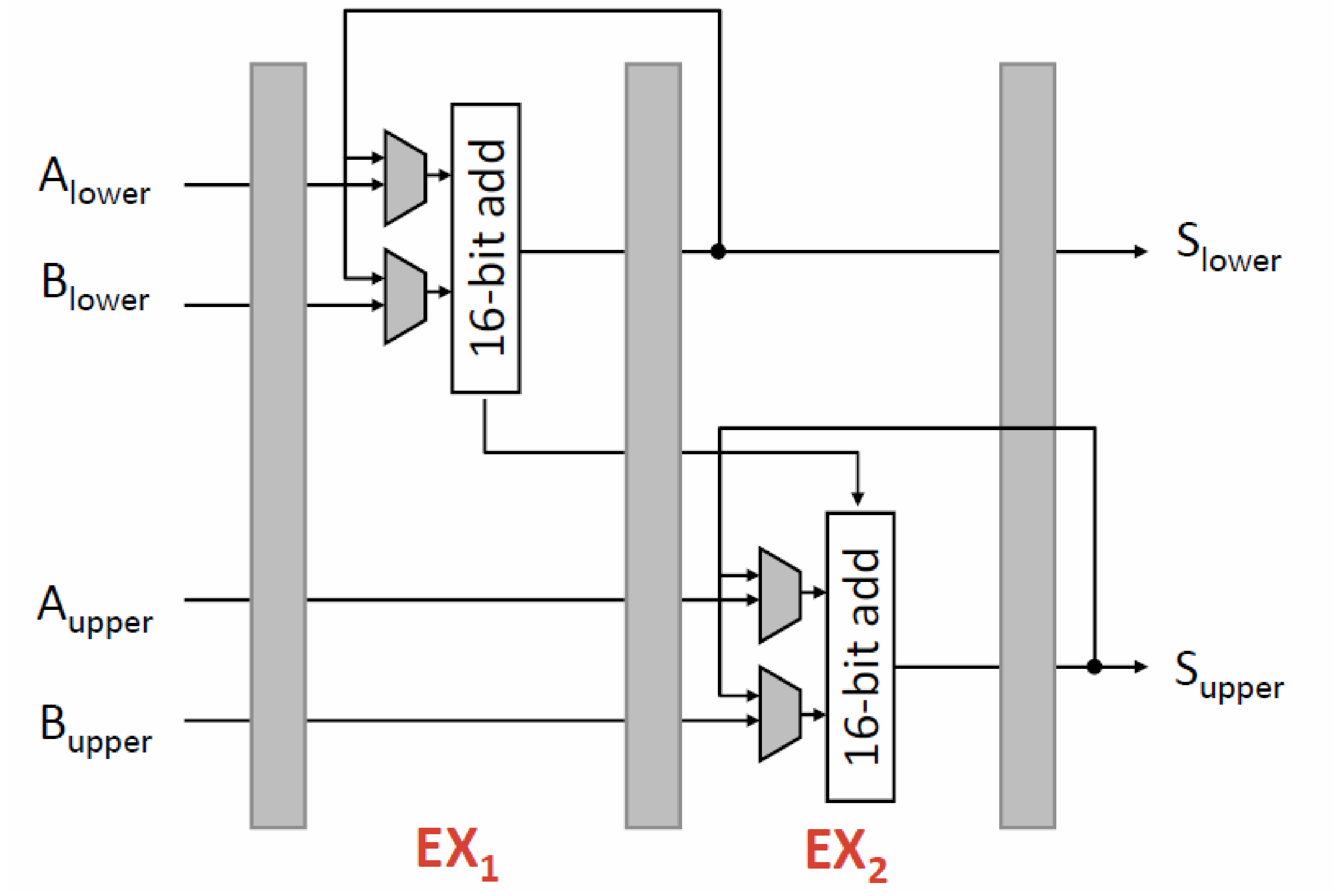
Why not very deep pipelines?

- 5-stage pipeline still has plenty of combinational delay between registers
- "**Superpipelining**" → increase pipelining degree such that even intrinsic operations (e.g., ALU, RF read/write, memory access) require multiple stages
- What's the problem?

Inst_0 : r1 ← r2 + r3
Inst_1 : r4 ← r1 + 2


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

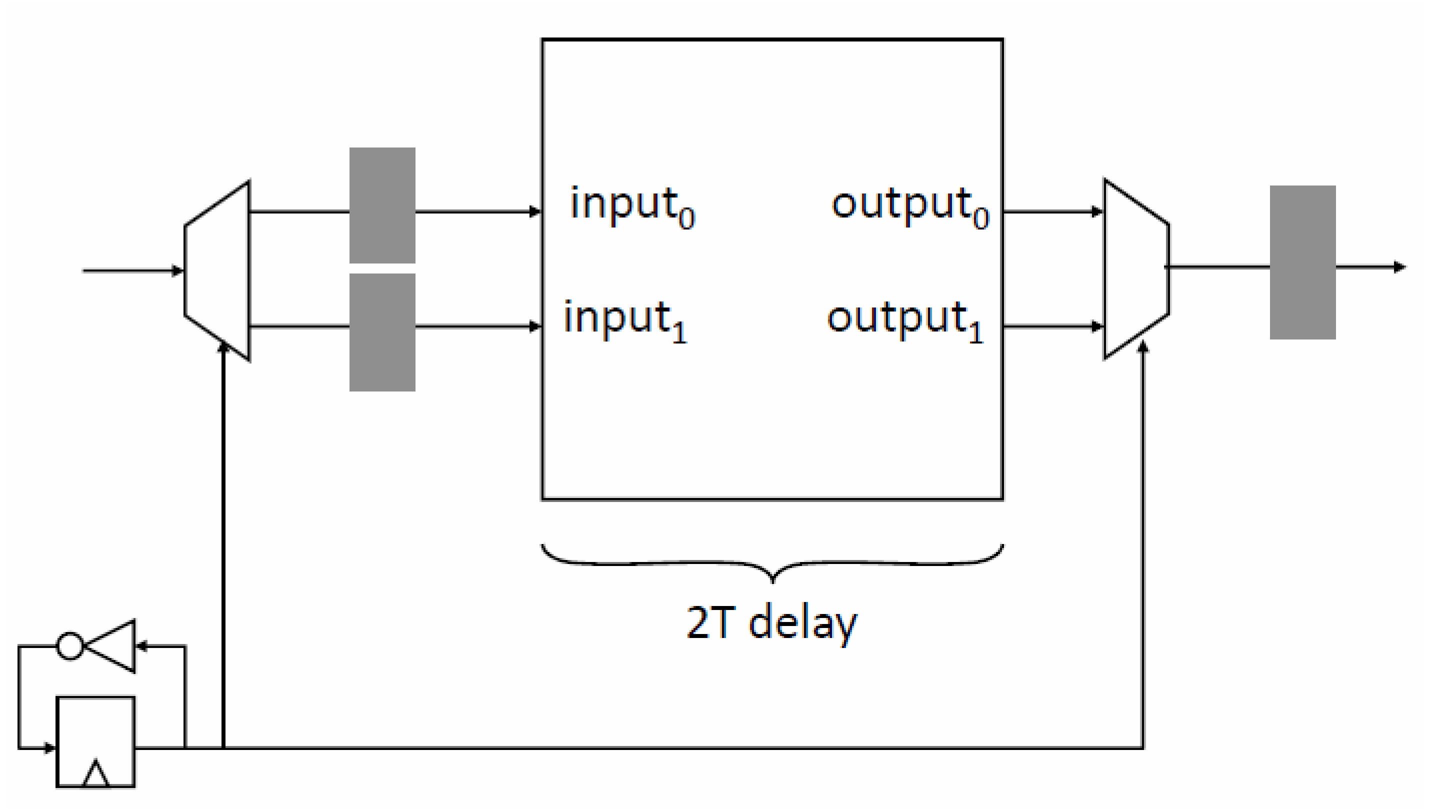
Intel P4's Superpipelined Integer ALU



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

No stall between back-to-back dependencies

What if we can't really superpipeline?



If you can't double the bandwidth by deeper pipelining, you can put more resources to increase the bandwidth.

However, this does not mean that RAW is 100% solved.

Question?

Announcements: homework #2 will be collected (due: 4/12)

Reading: finish reading P&H Ch.4

Handouts: none