

# Lecture 10: Processor design – pipelining

---

- Overlapping the execution of instructions
- Pipeline hazards
  - Different types
  - How to remove them



# Pipelining

---

- Let's assume that all instructions take 5 steps, e.g.: `lw r1, n(r2) # r1=memory[n+r2]`

<u>Step</u>	<u>Name</u>	<u>Datapath operation</u>
0	IF	Fetch instruction; PC+4 → PC
1	REG	Get value from r2
2	ALU	ALU n+r2
3	MEM	Get data from memory
4	WB	Write memory data into r1

IF = instruction fetch (includes PC increment)

REG = fetching values from general purpose registers

ALU = arithmetic/logic operations

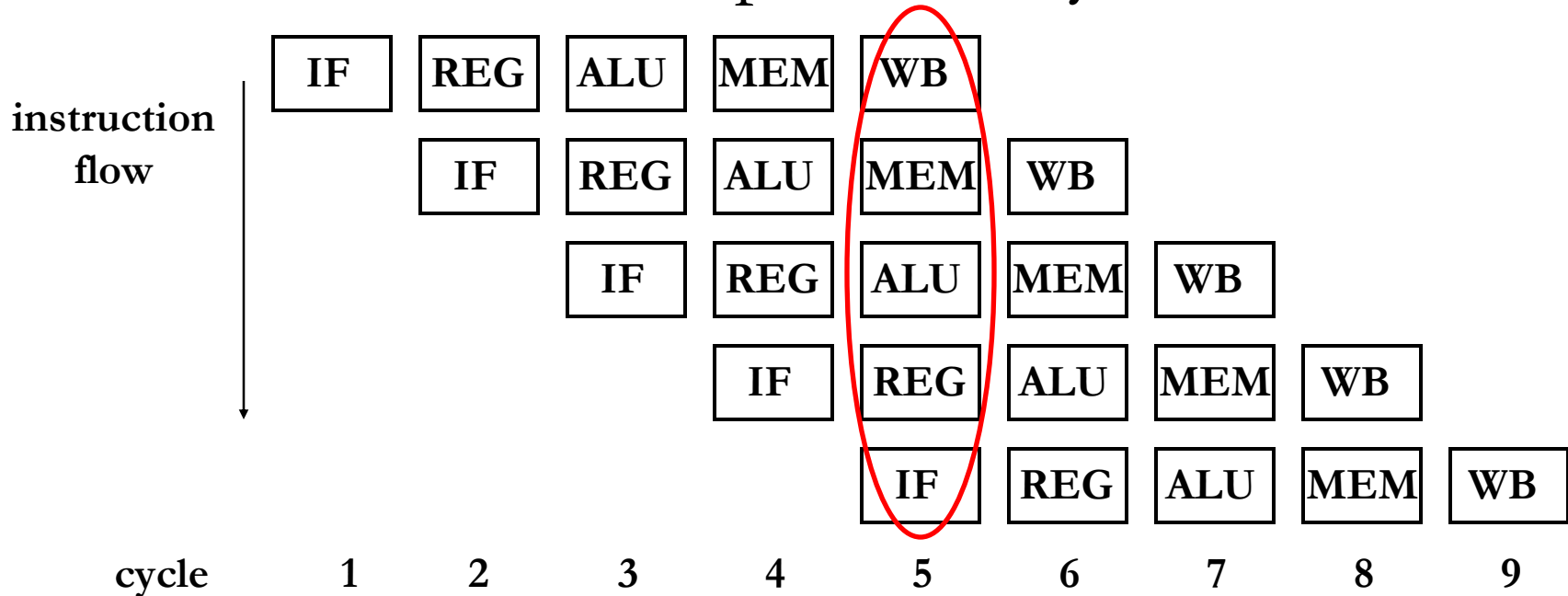
MEM = memory access

WB = write back results to general purpose registers



# Pipelining

- Start one instruction per clock cycle



- Five instructions are being executed (in different stages) during the same cycle
- Each instruction still takes 5 cycles, but instructions now complete every cycle:  $CPI \rightarrow 1$



# Preparing instructions for pipelining

---

- Stretch the execution to the max number of cycles, e.g.

**sw r1, n(r2) # memory[n+r2]=r1**

IF	Fetch instruction; PC+4 → PC
REG	Get values of r1 and r2 from registers
ALU	ALU n+r2
MEM	Store value of r1 to memory
WB	Do nothing

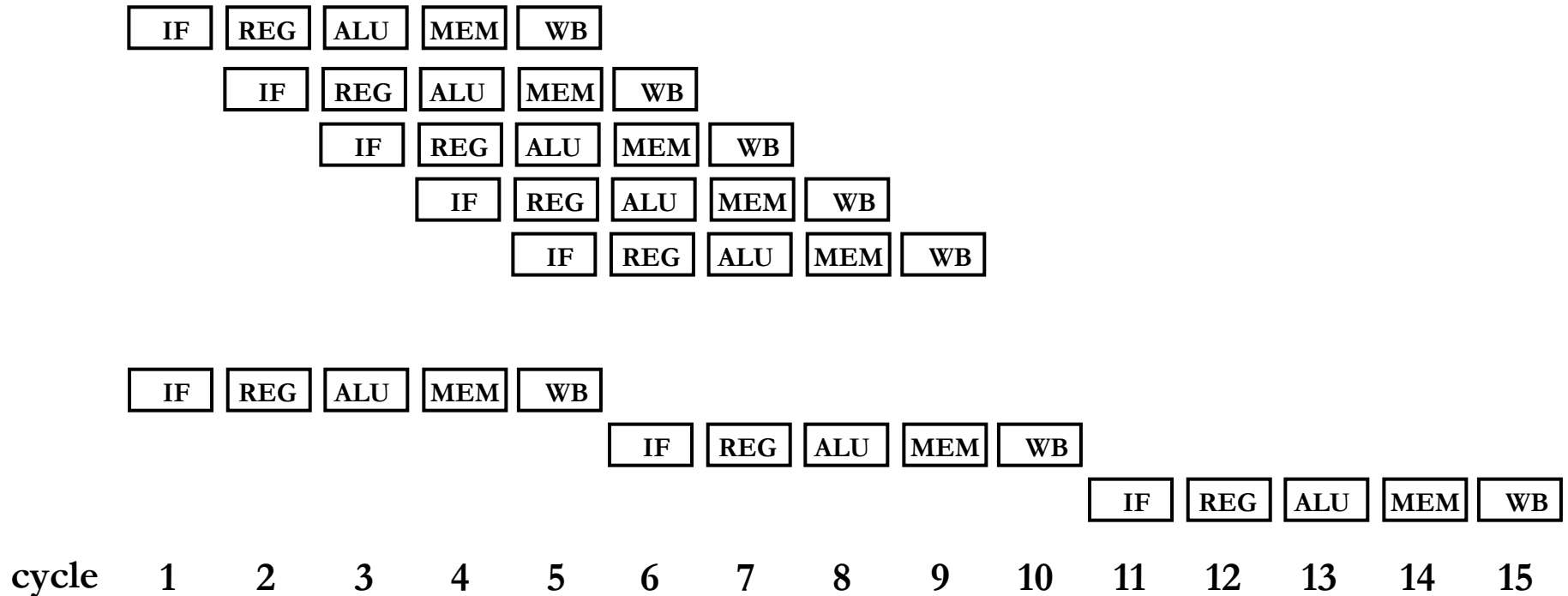
**add r1, r2, r3 # r1=r2+r3**

IF	Fetch instruction; PC+4 → PC
REG	Get values of r2 and r3 from registers
ALU	ALU r2+r3
MEM	Do nothing
WB	Write result to r1



# Execution speedup

---



- Speed-up roughly equal to the number of stages



# Pipeline hazards

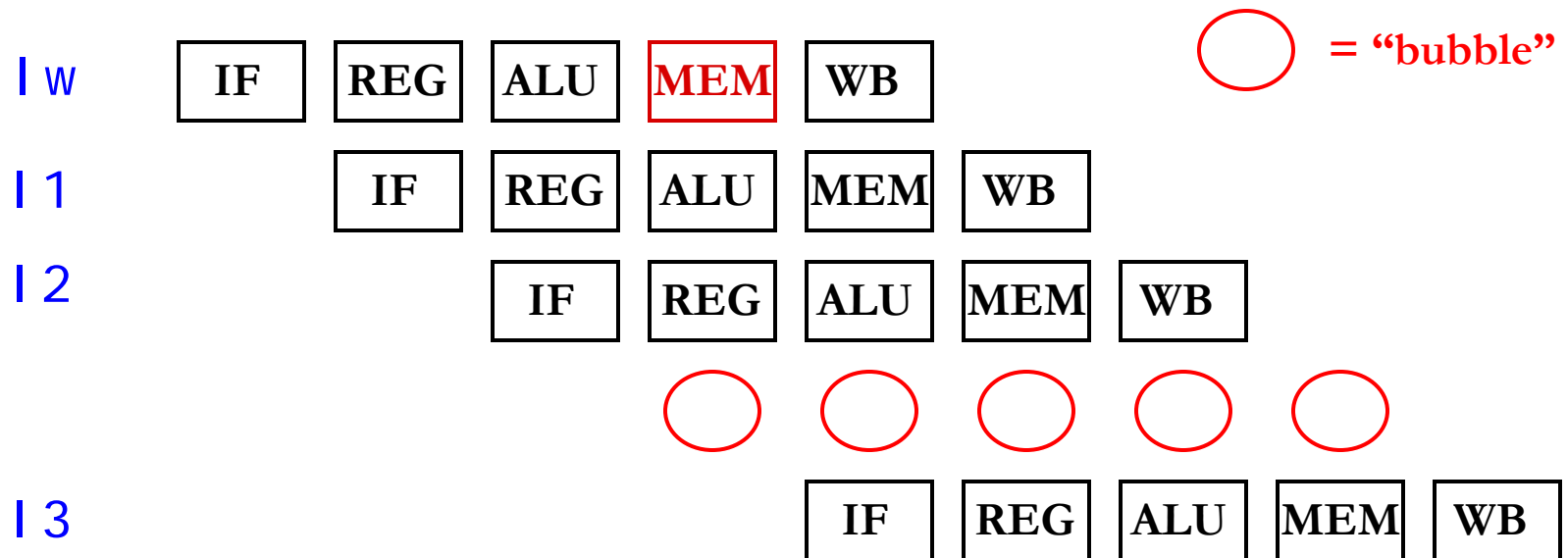
---

- Complications in pipelining, called **hazards**
  - Structural
  - Data
  - Control
- Speedup achieved is limited, CPI over 1



# Structural hazards

- Example: instructions in IF and MEM stages may conflict for access to memory (cache)



# Structural hazards

---

- Not enough hardware resources to execute a combination of instructions in the same clock cycle
- Straightforward solution: use more resources
  - E.g. split cache into instruction cache (used in IF) and data cache (used in MEM)
- Good design – provide enough resources to avoid hazards for common/frequent cases



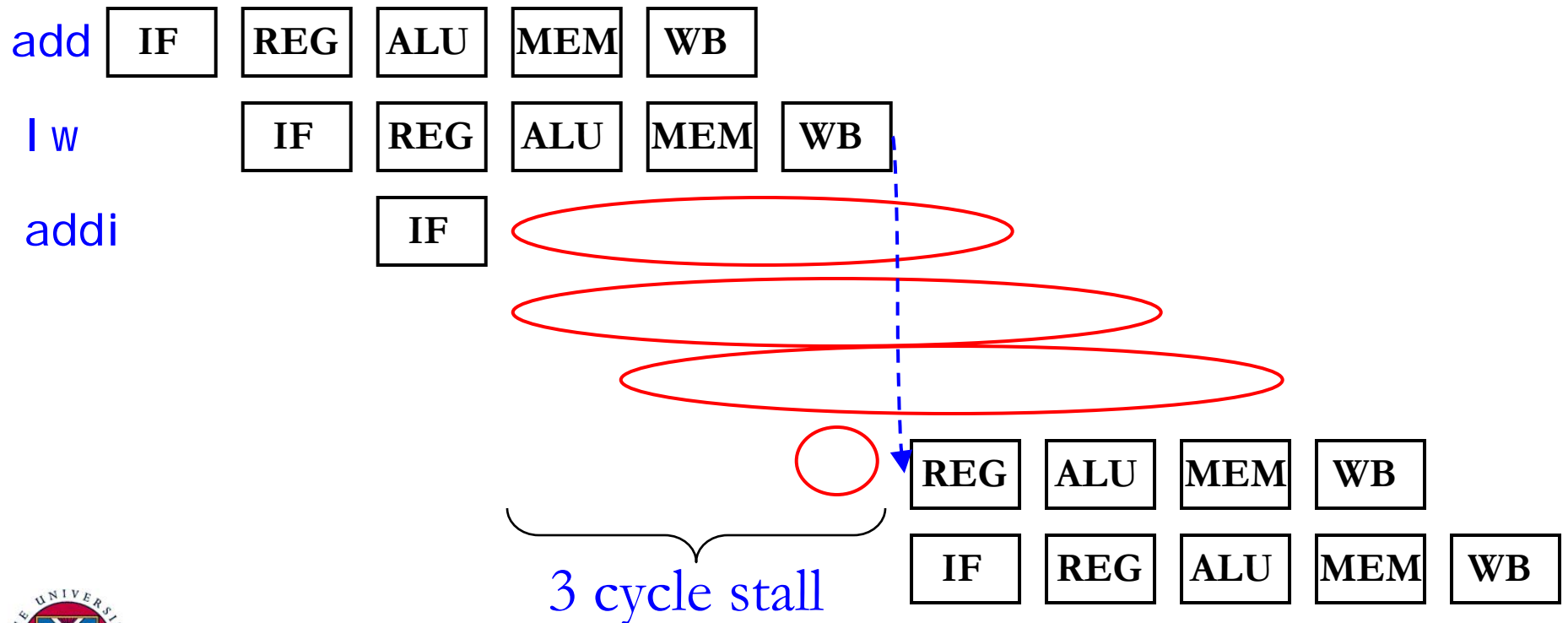


# Data hazards

- One instruction must use value produced by a previous instruction
- Example:
 

```

add  r2, r1, r5
lw   r3, 4(r1)
addi r4, r3, n
      
```

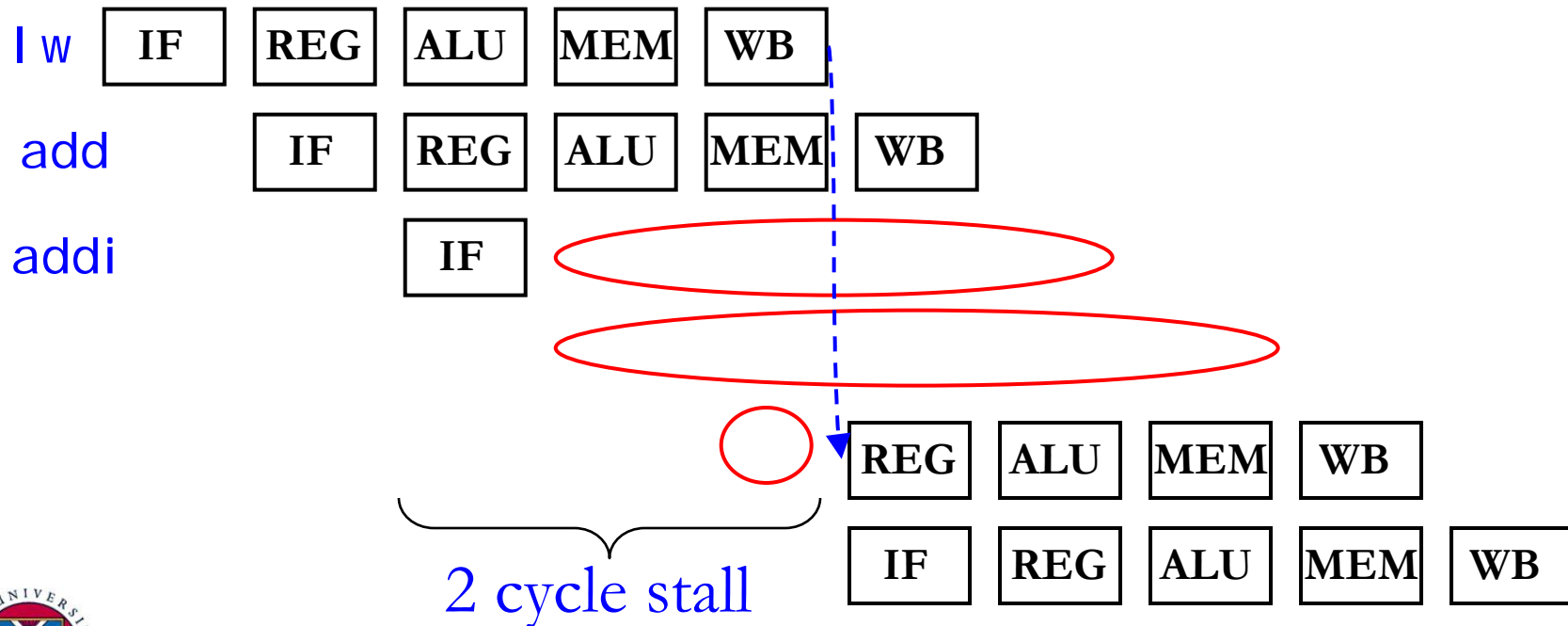


# Data hazards

- Processor must detect hazards and insert bubbles
- Solution: compiler can separate dependent instructions

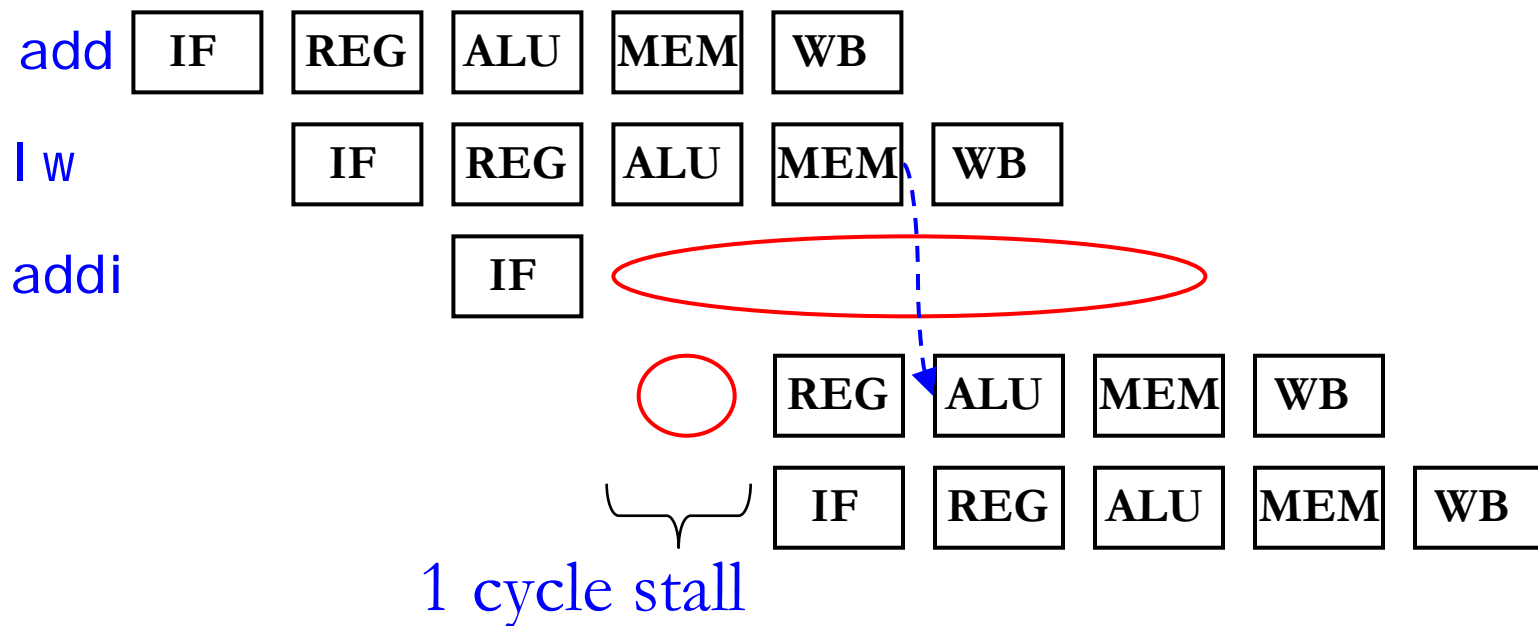
```

lw r3, 4(r1)
add r2, r1, r5
addi r4, r3, n
    
```



# Data forwarding

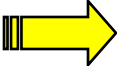
- The data is actually available before the end of WB
- Why not forward it directly to the unit/stage where they are needed?



# Control hazards

---

- Before a conditional branch instruction is **resolved**, the processor does not know where to fetch the next instruction from
- Example: **beq r1, r2, n**

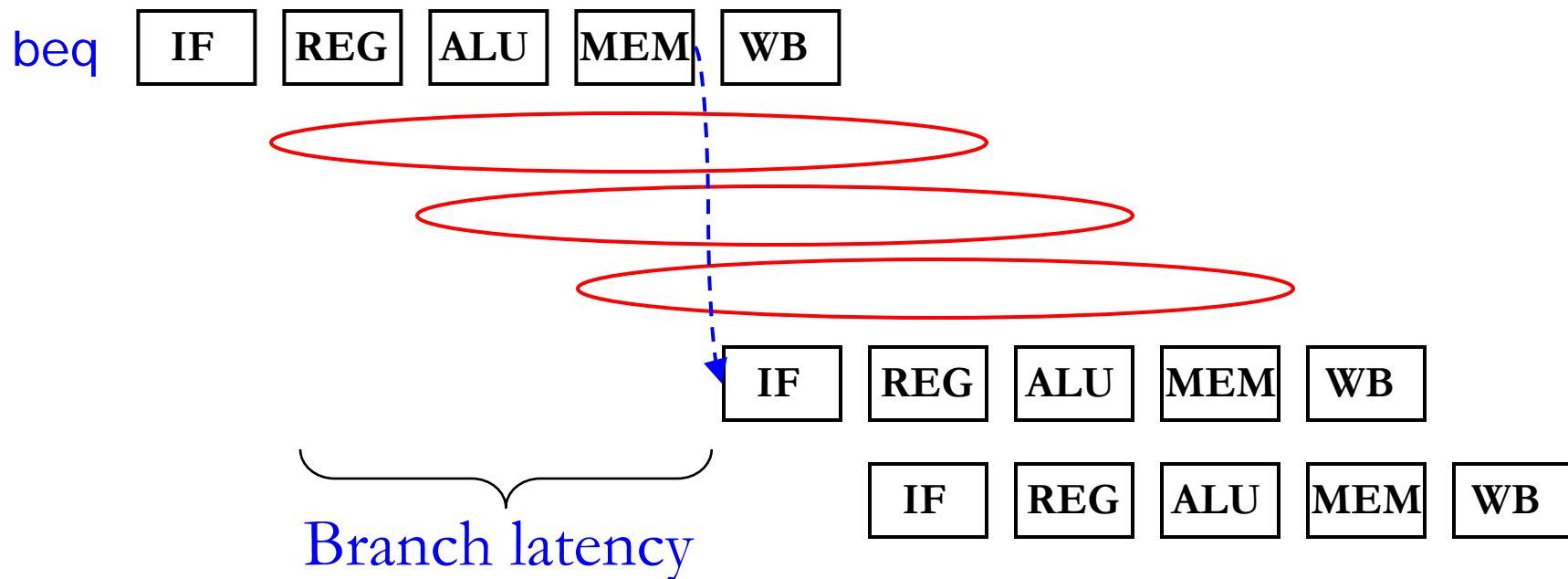
	IF	Fetch instruction; $PC+4 \rightarrow PC$
	REG	Get values of r1 and r2 from registers
	ALU	ALU $r1-r2$ and $PC+n$
	MEM	If $r1-r2==0$ update PC
	WB	<b>Do nothing</b>

- Branch is identified in IF but only resolved in MEM



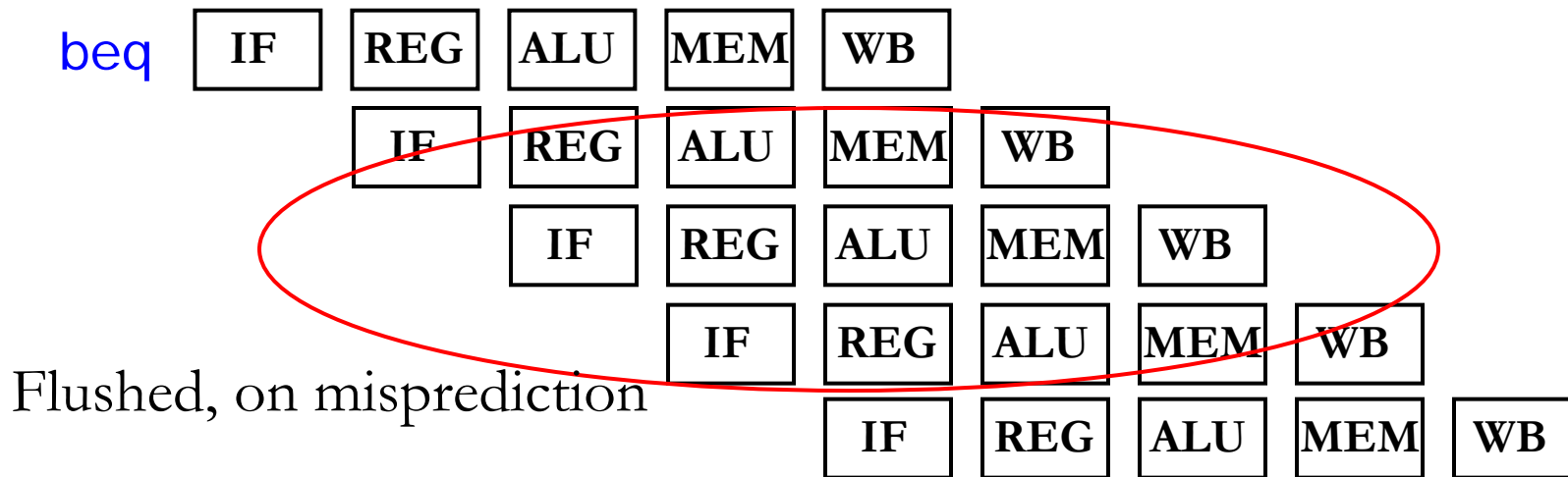
# Control hazards

---



# Branch prediction

- Solution: predict outcome of branch
  - If prediction is correct then no bubble
  - If prediction incorrect, processor must discard (“flush” or “squash”) incorrectly loaded instructions



# Is this the end? in performance improvement

---

- Superscalar processors:
  - Can fetch more than 1 instruction per cycle
  - Have multiple pipelines and ALUs to execute multiple instructions simultaneously
- Predicated execution:
  - Execute simultaneously instructions from both targets of the branch and discard the incorrect one (e.g. IA-64) (against control hazards)
- Value prediction:
  - Predict result of instructions (against data hazards)
- Multiprocessors

