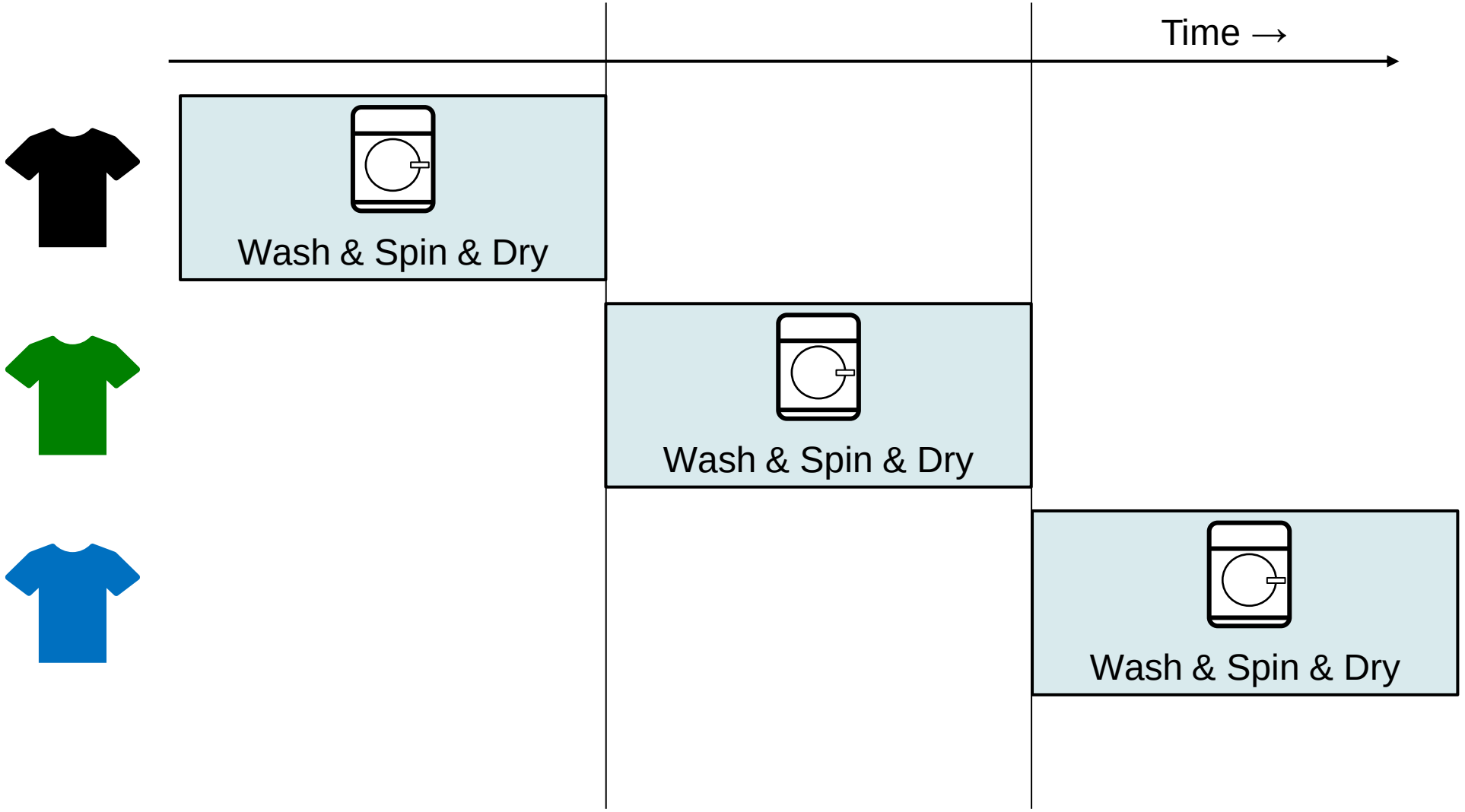
Introduction to Computer Architecture
Chapter 4 - 3

Pipelining

Hyungmin Cho

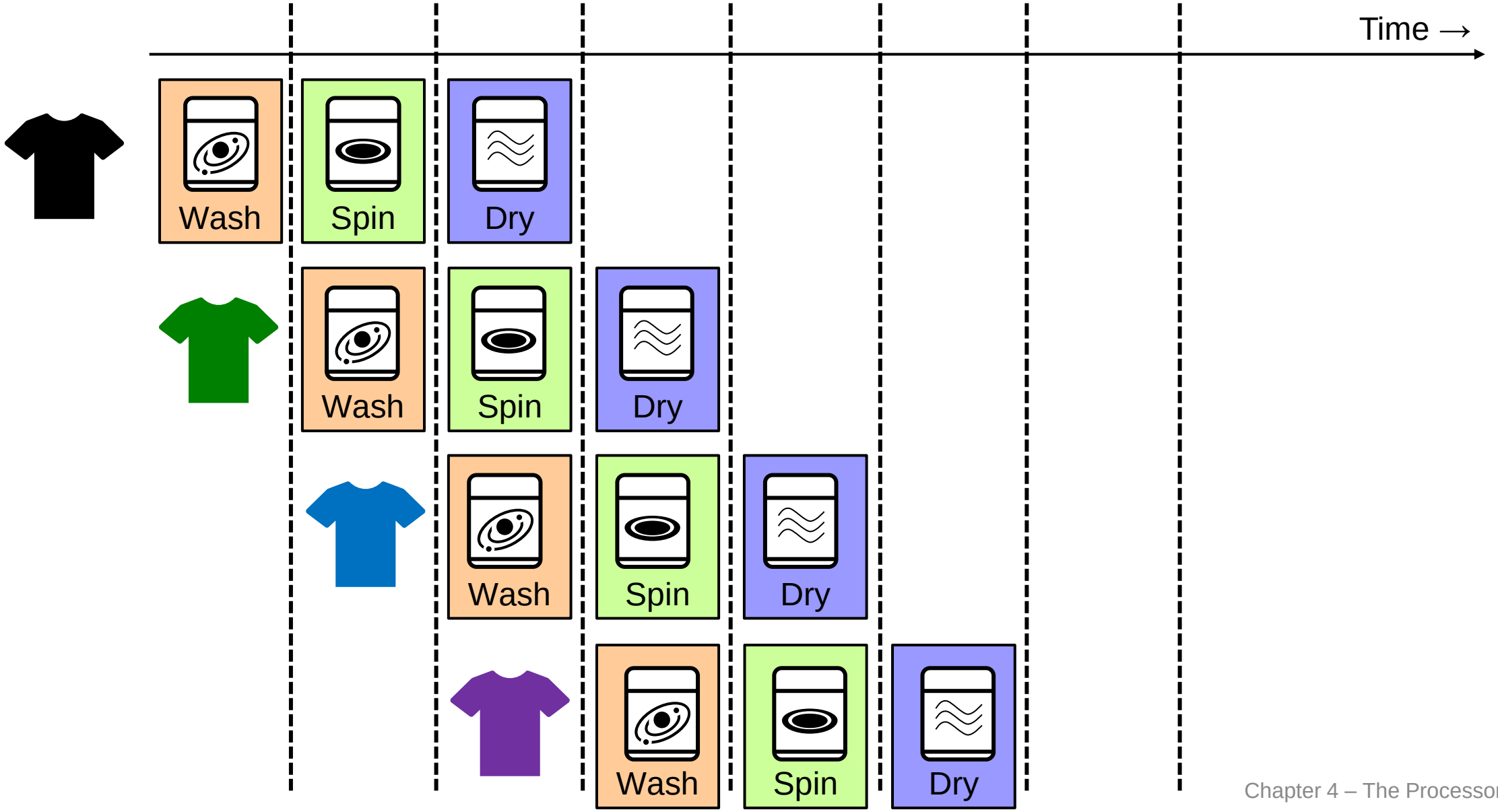
Department of Computer Science and Engineering
Sungkyunkwan University

Pipelining Analogy

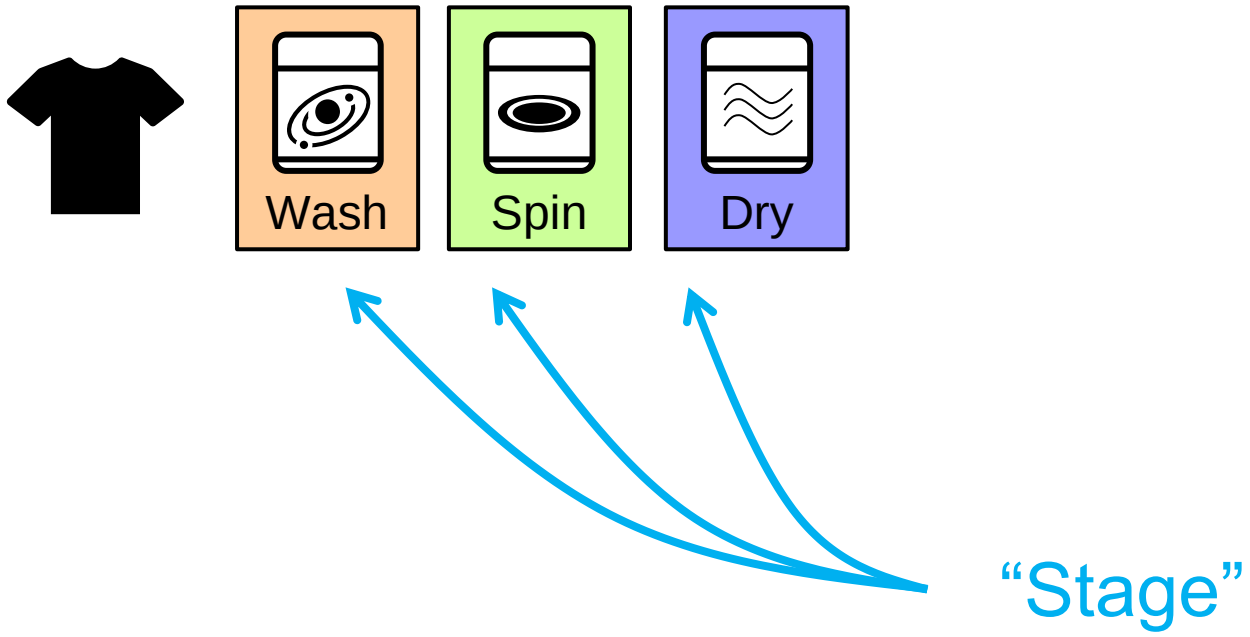


Pipelining Analogy

Pipelining Analogy

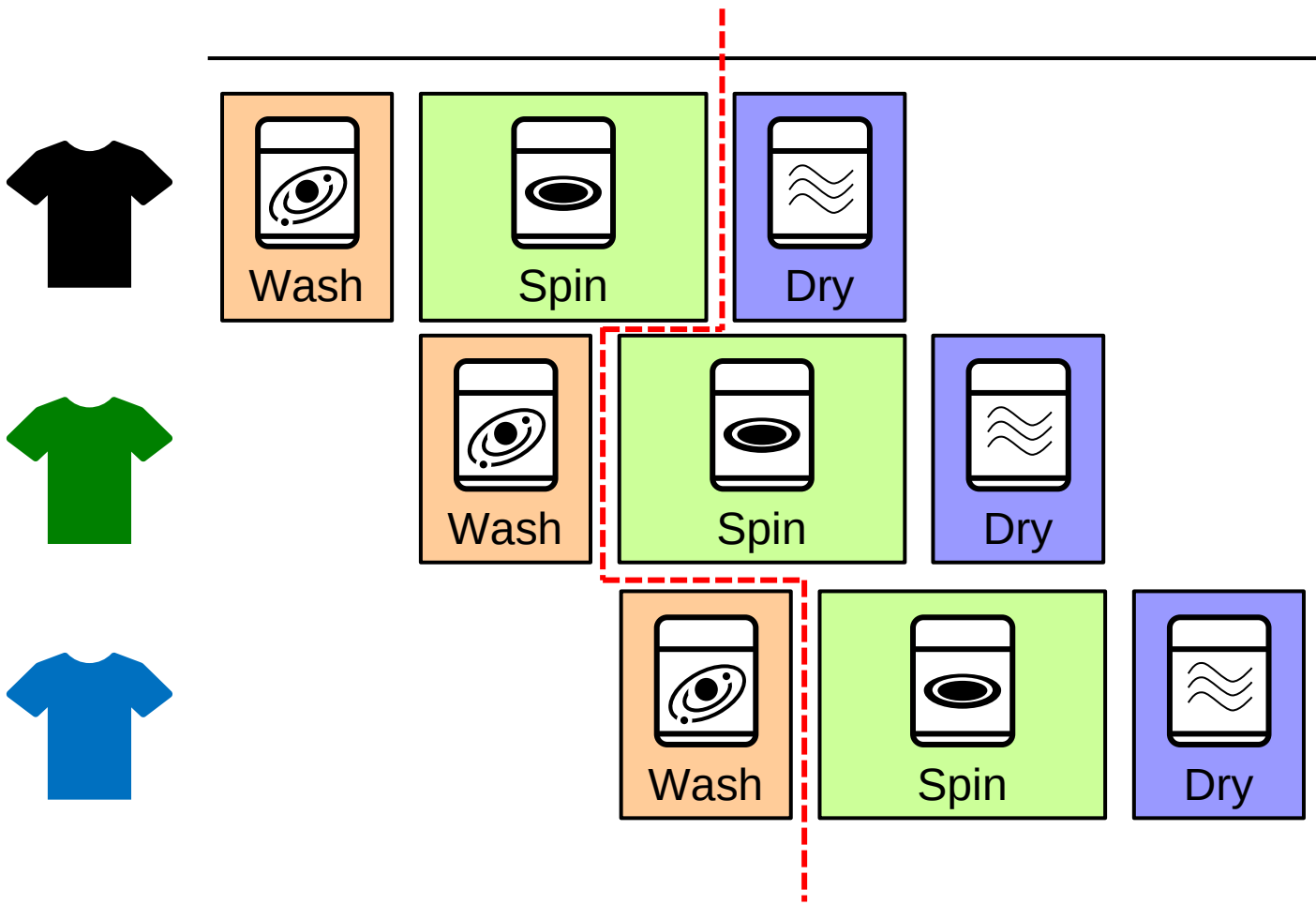


Pipelining Analogy



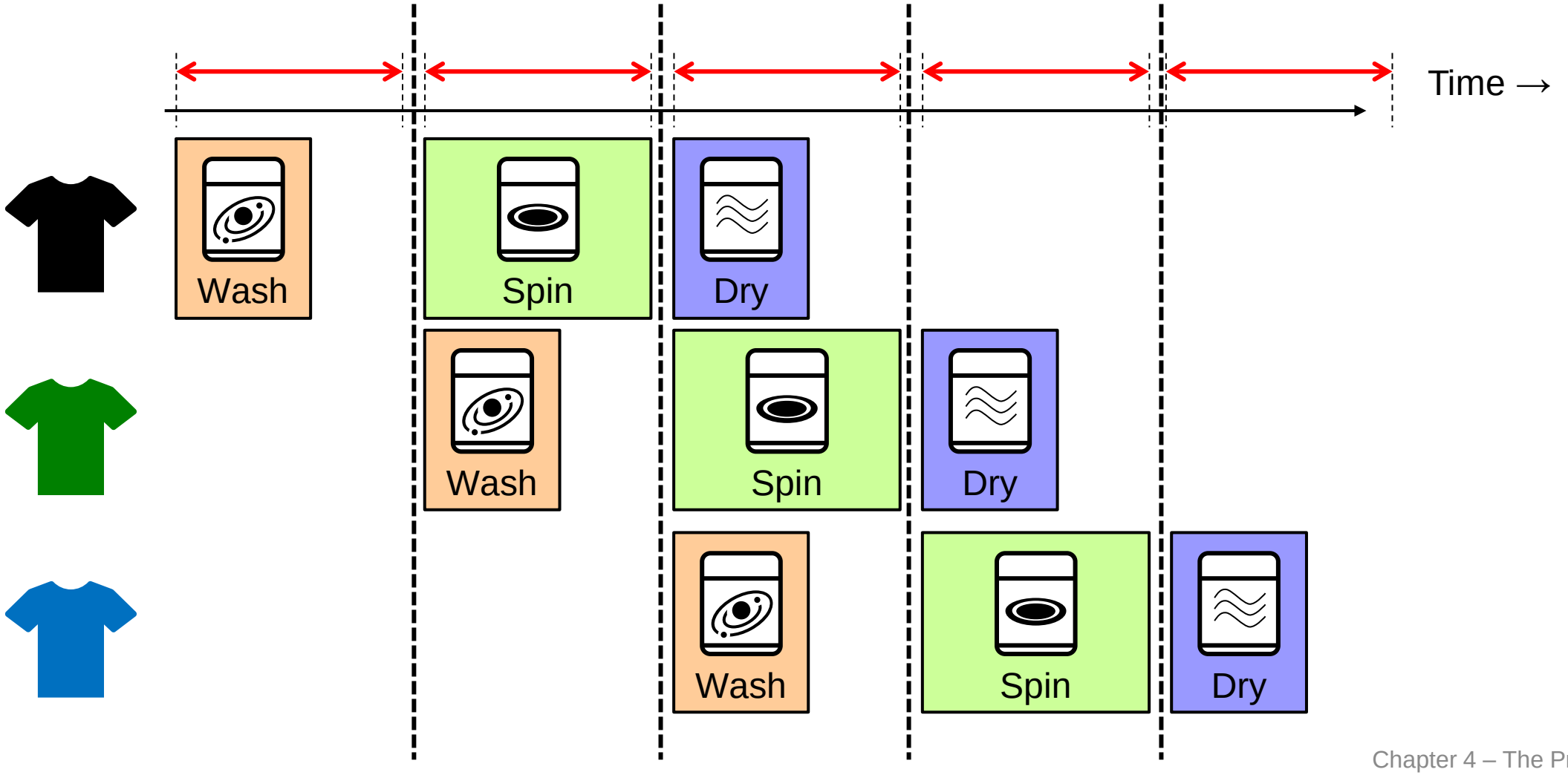
Clock Period in Pipeline

- All stages use the same clock input
- Determined by the longest stage



Clock Period in Pipeline

- Determined by the longest stage

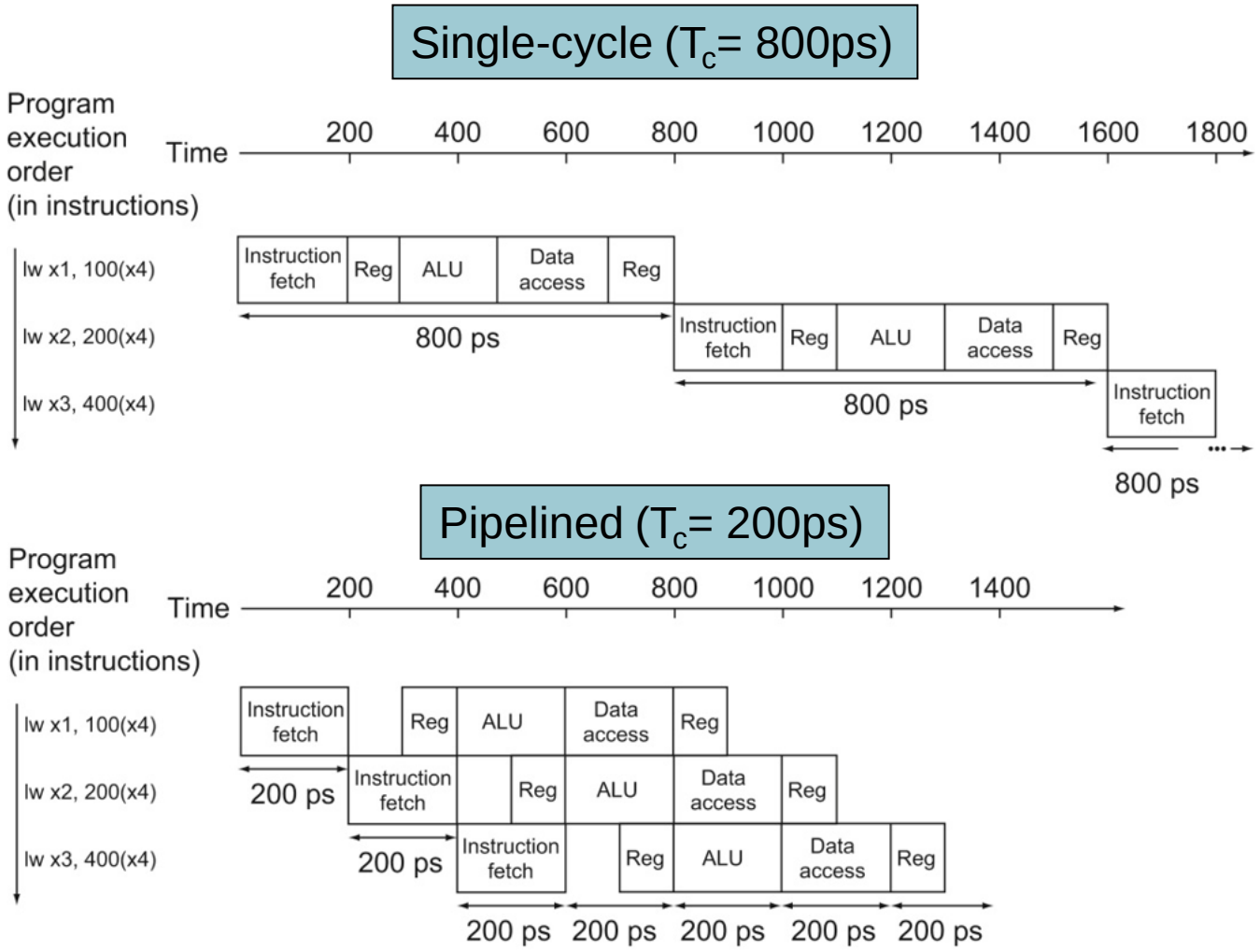


Pipeline Performance

- Assume the time for each stages is
  - 100ps for register read or write
  - 200ps for other stages

Instruction type	Instruction fetch	Register read	ALU op	Memory access	Register write	Total time
`lw`	200ps	100 ps	200ps	200ps	100 ps	800ps
`sw`	200ps	100 ps	200ps	200ps		700ps
`R-format`	200ps	100 ps	200ps		100 ps	600ps
`beq`	200ps	100 ps	200ps			500ps

Pipeline Performance



Pipeline Speedup

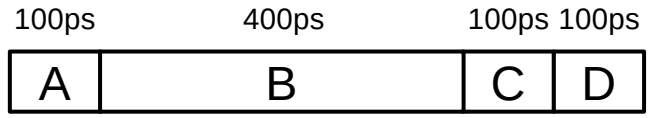
- If all stages are balanced
  - i.e., all take the same time

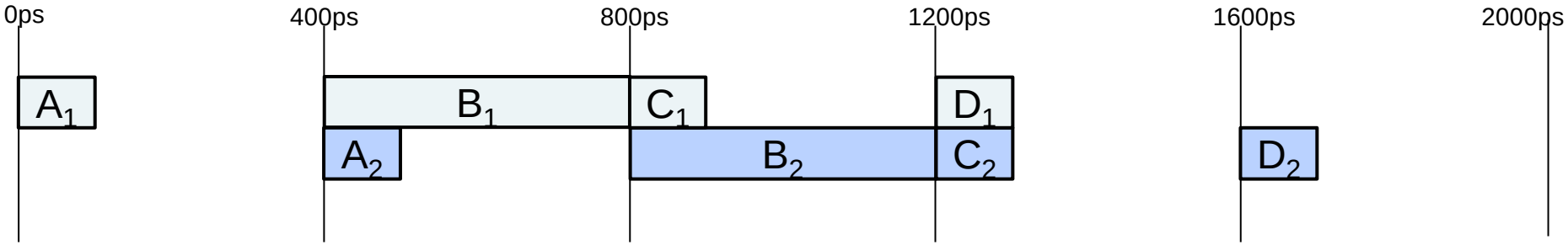
$$T_{pipelined} = \frac{T_{\boldsymbol{non-pipelined}}}{Number\ of\ stages}$$

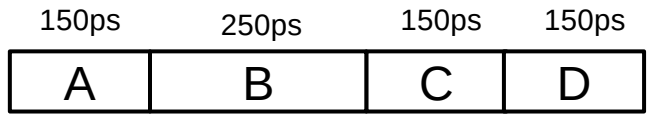
- If not balanced, speedup is limited
  - The longest stage determines the clock period

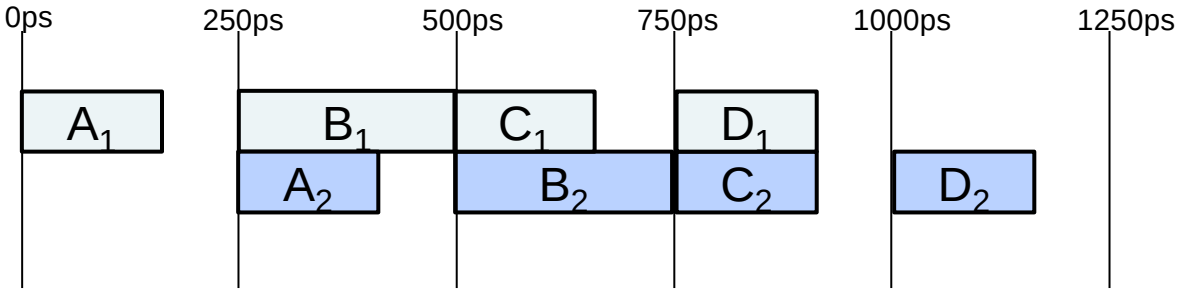
- Speedup due to increased throughput
  - Latency (time for each instruction) does not decrease

Pipeline Speedup Examples

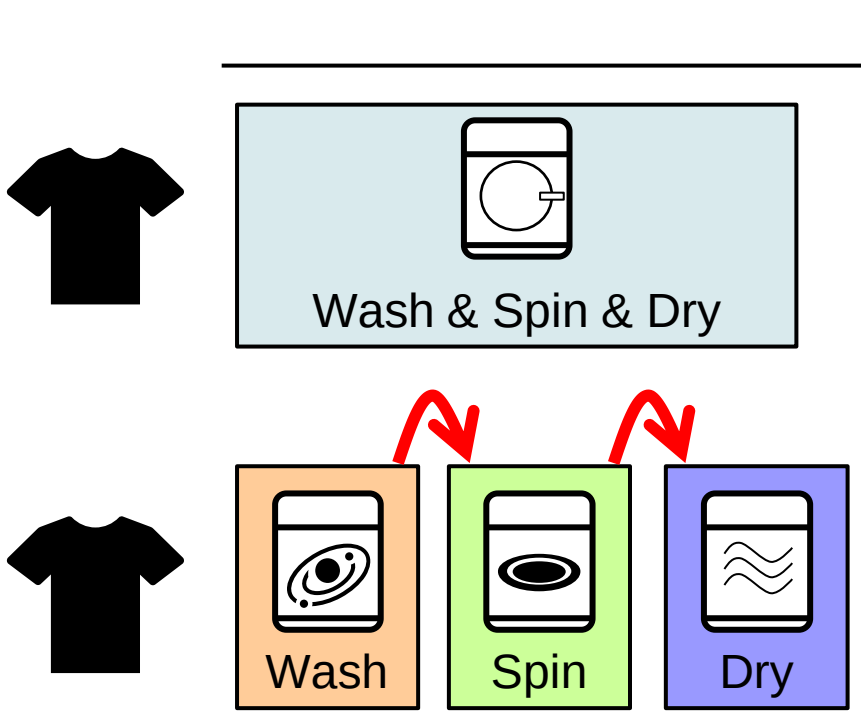








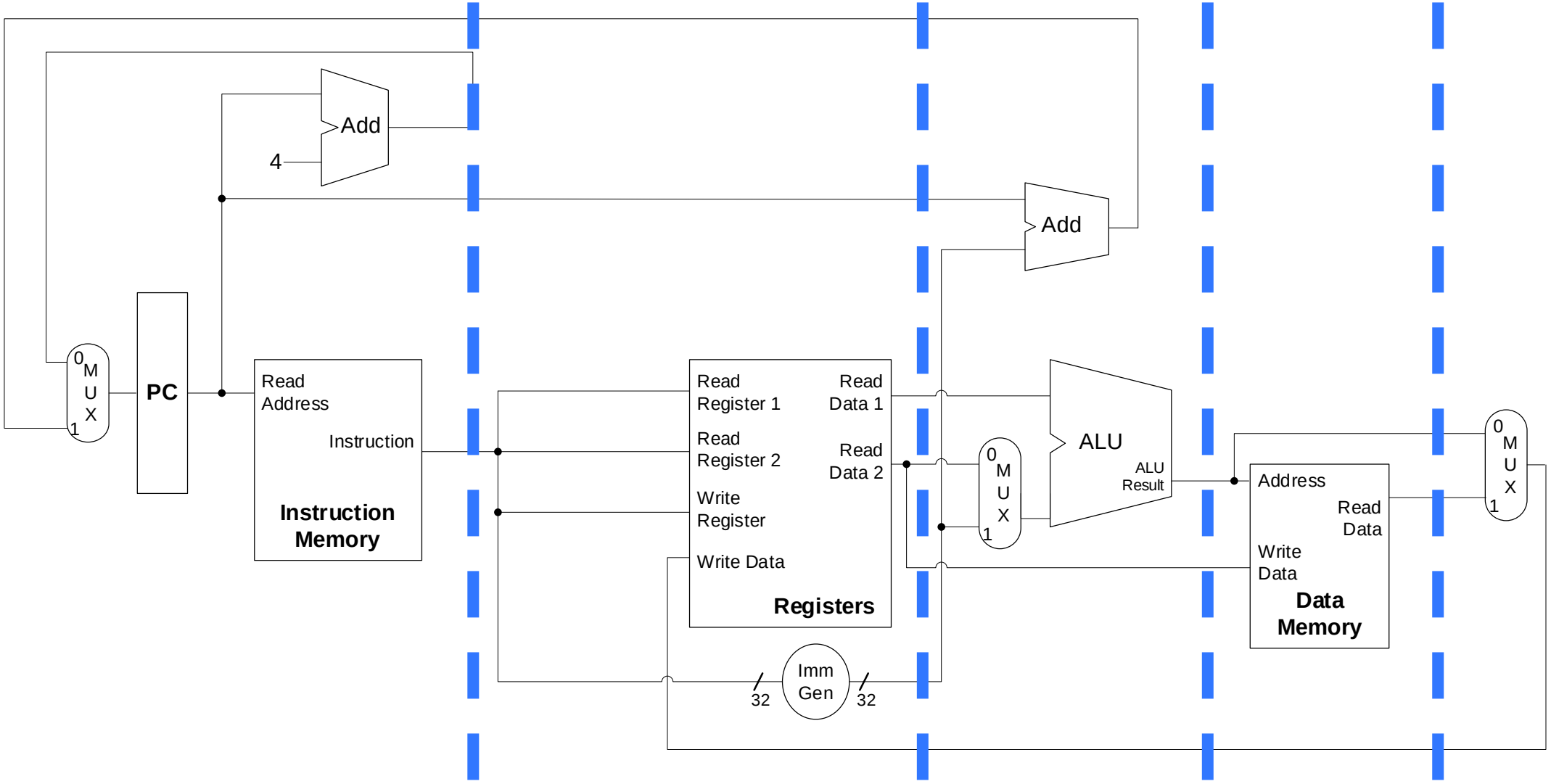
Pipelining Analogy



Pipelining and ISA Design

- RISC-V ISA designed for pipelining
  - All instructions are 32-bits
    - Easier to fetch and decode in one cycle
    - c.f. x86: 1- to 17-byte instructions
  - Few and regular instruction formats
    - Can decode and read registers in one step
  - Load/store addressing
    - Can calculate address in 3rd stage, access memory in 4th stage

RISC-V Datapath

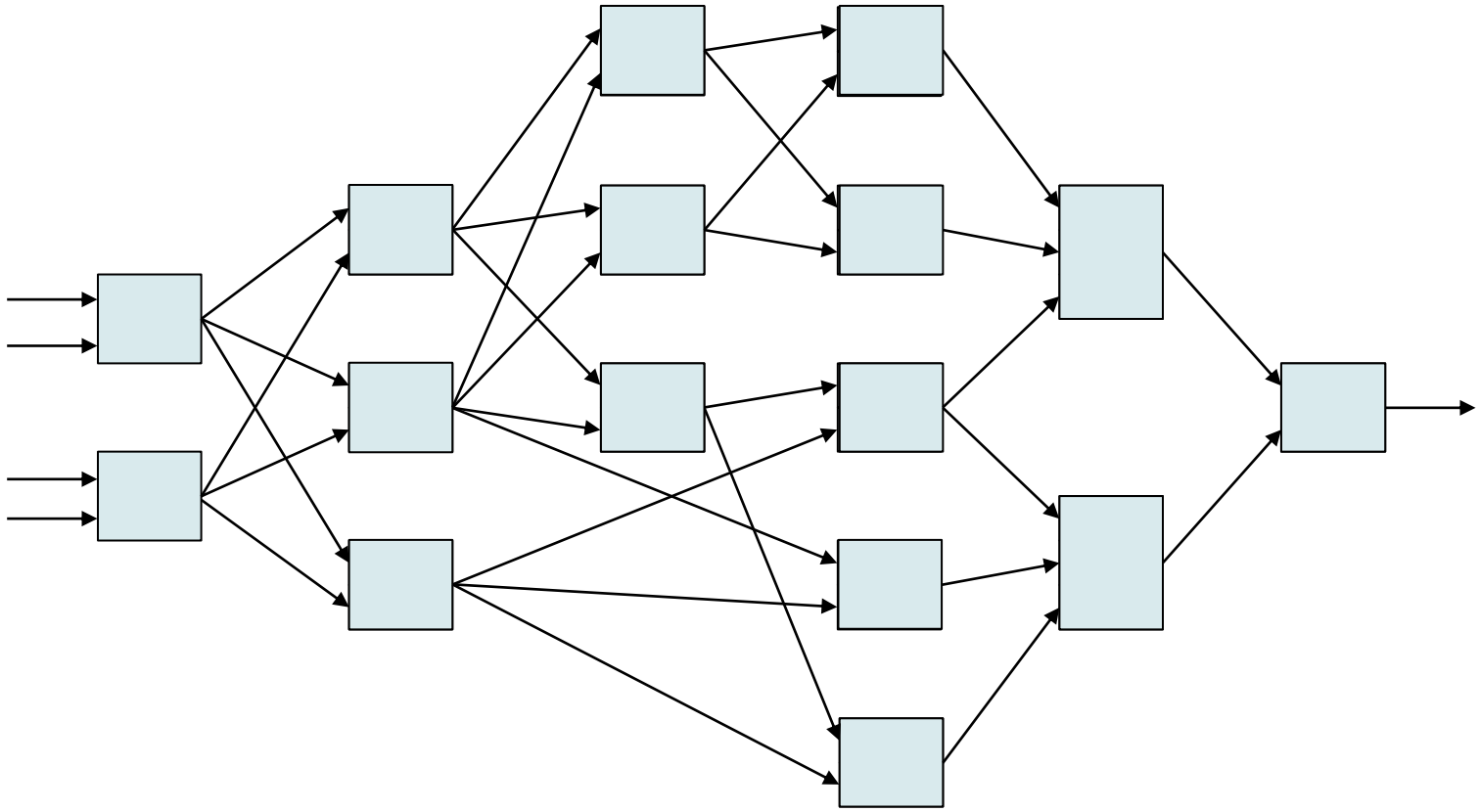


RISC-V Pipeline Stages

- Five stages, one step per stage
  1. **IF**: Instruction fetch from memory
  2. **ID**: Instruction decode & register read
  3. **EX**: Execute operation or calculate address
  4. **MEM**: Access memory operand
  5. **WB**: Write result back to register

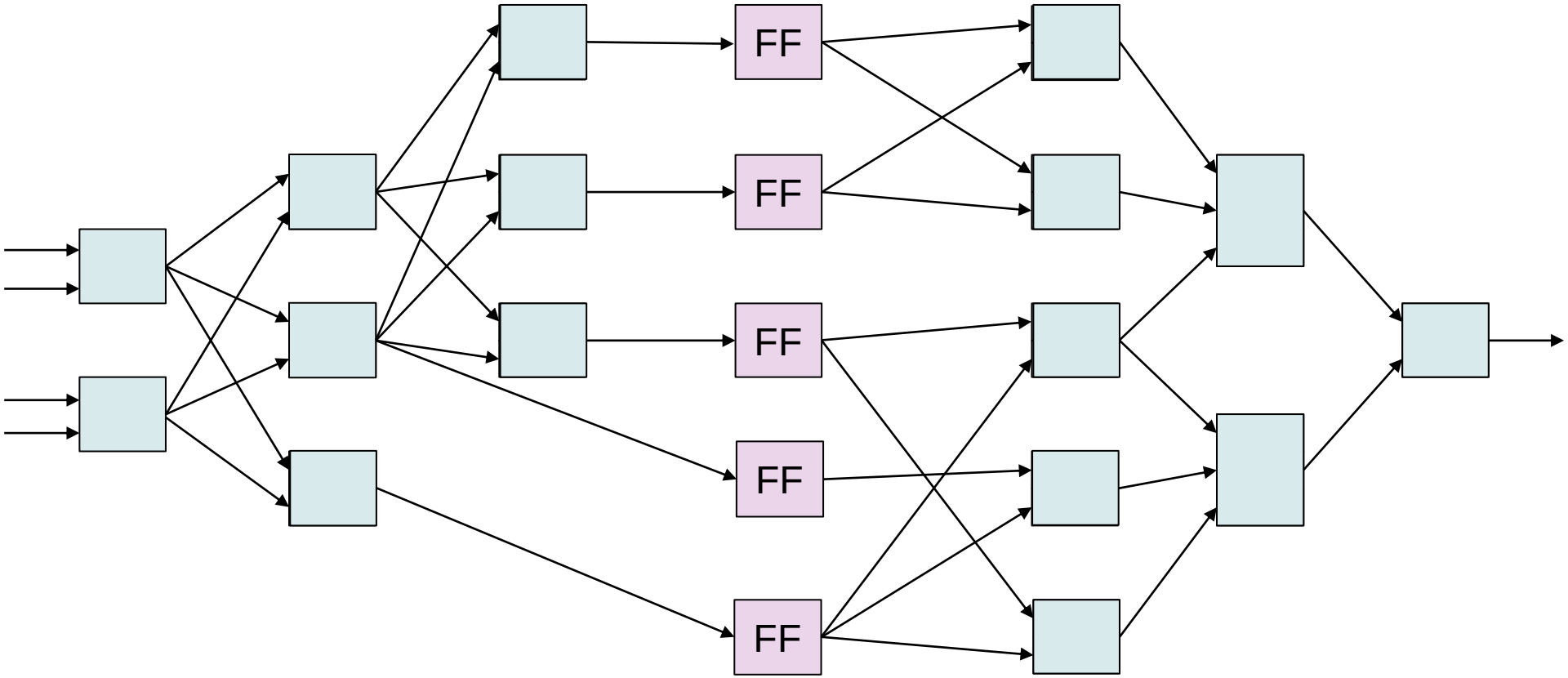
Pipeline Stage

- Can we split a combinational logic at arbitrary locations to obtain a perfectly balanced pipeline?



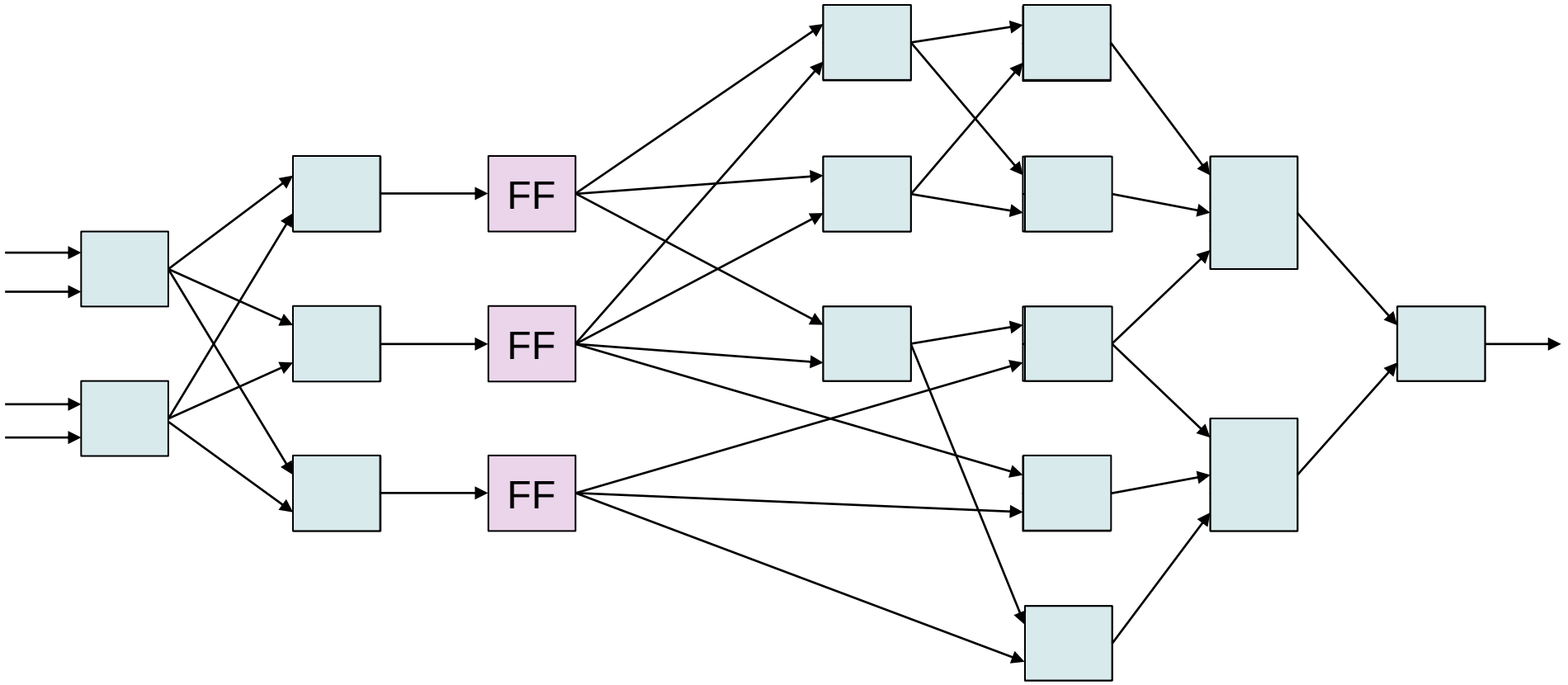
Pipeline Stage

- Can we split a combinational logic at arbitrary locations to obtain a perfectly balanced pipeline?

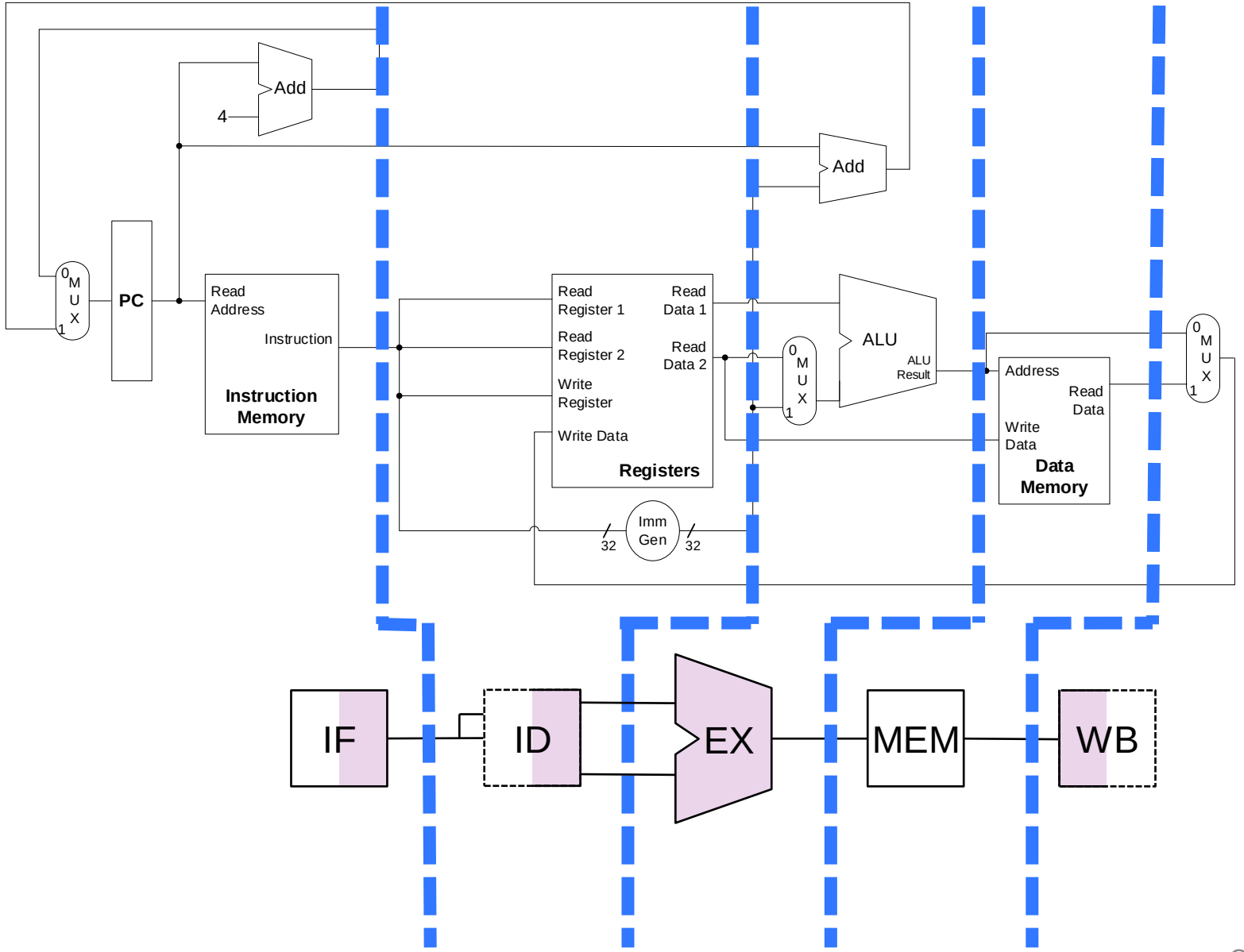


Pipeline Stage

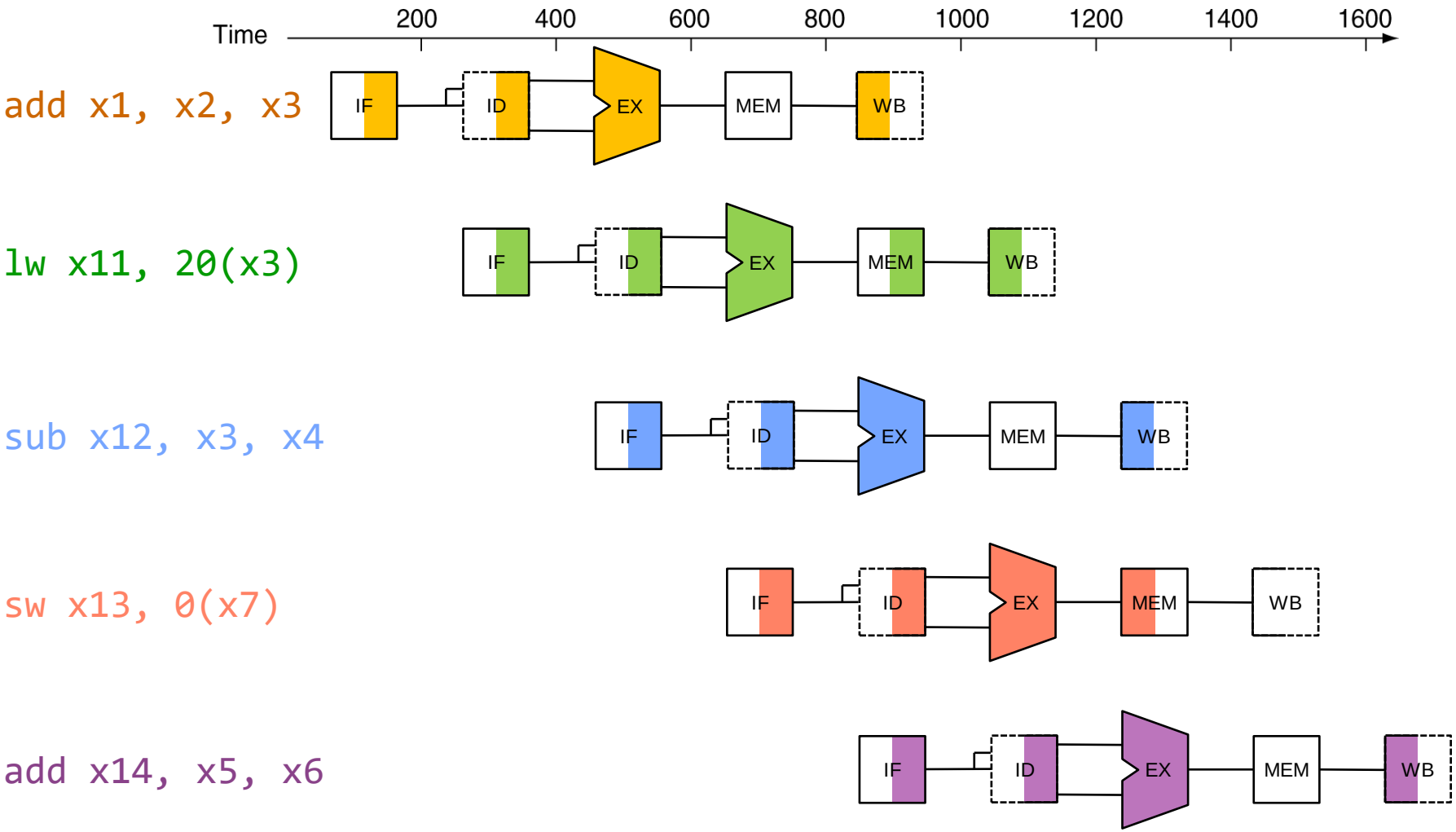
- Can we split a combinational logic at arbitrary locations to obtain a perfectly balanced pipeline?



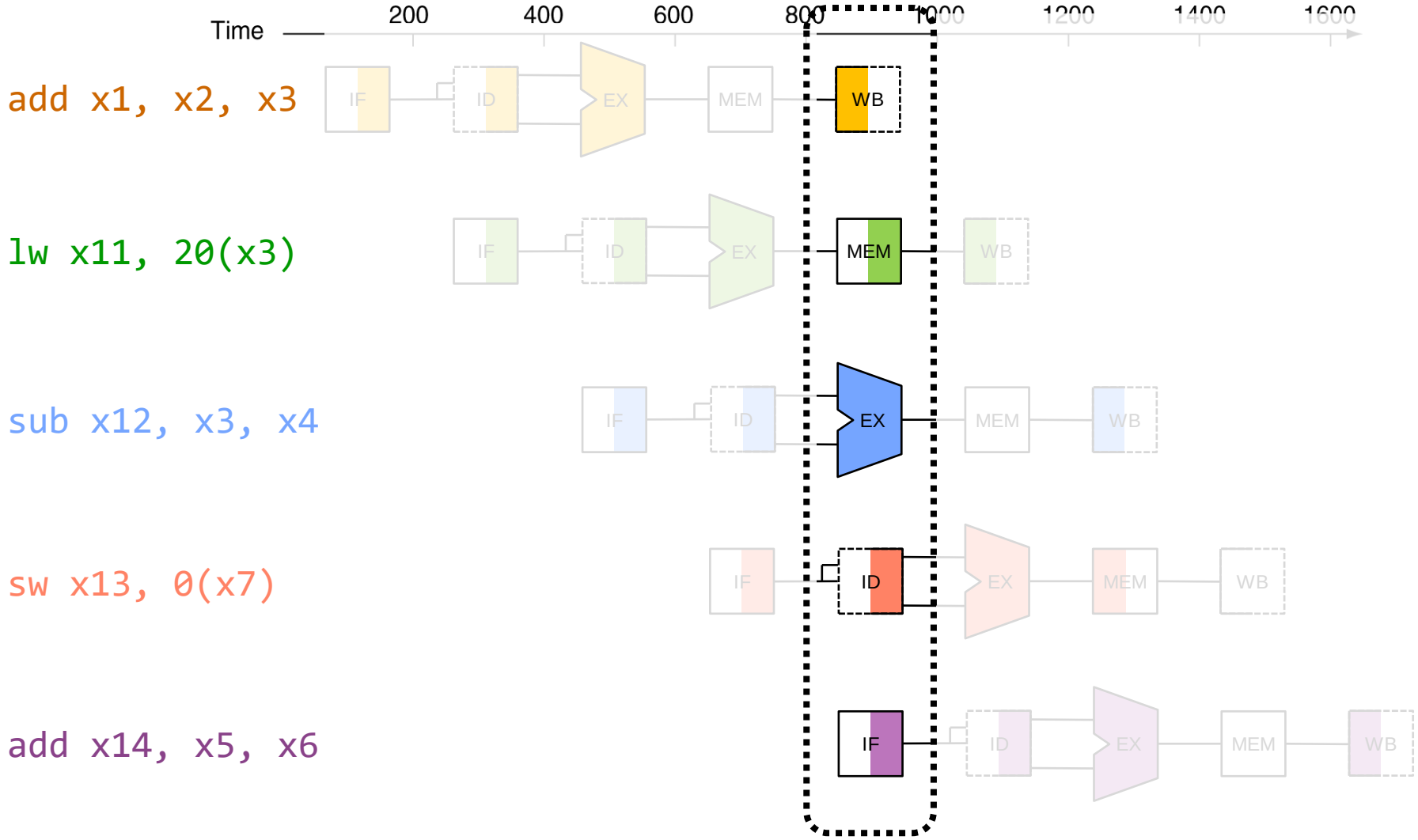
Simplified Representation of RISC-V Pipeline



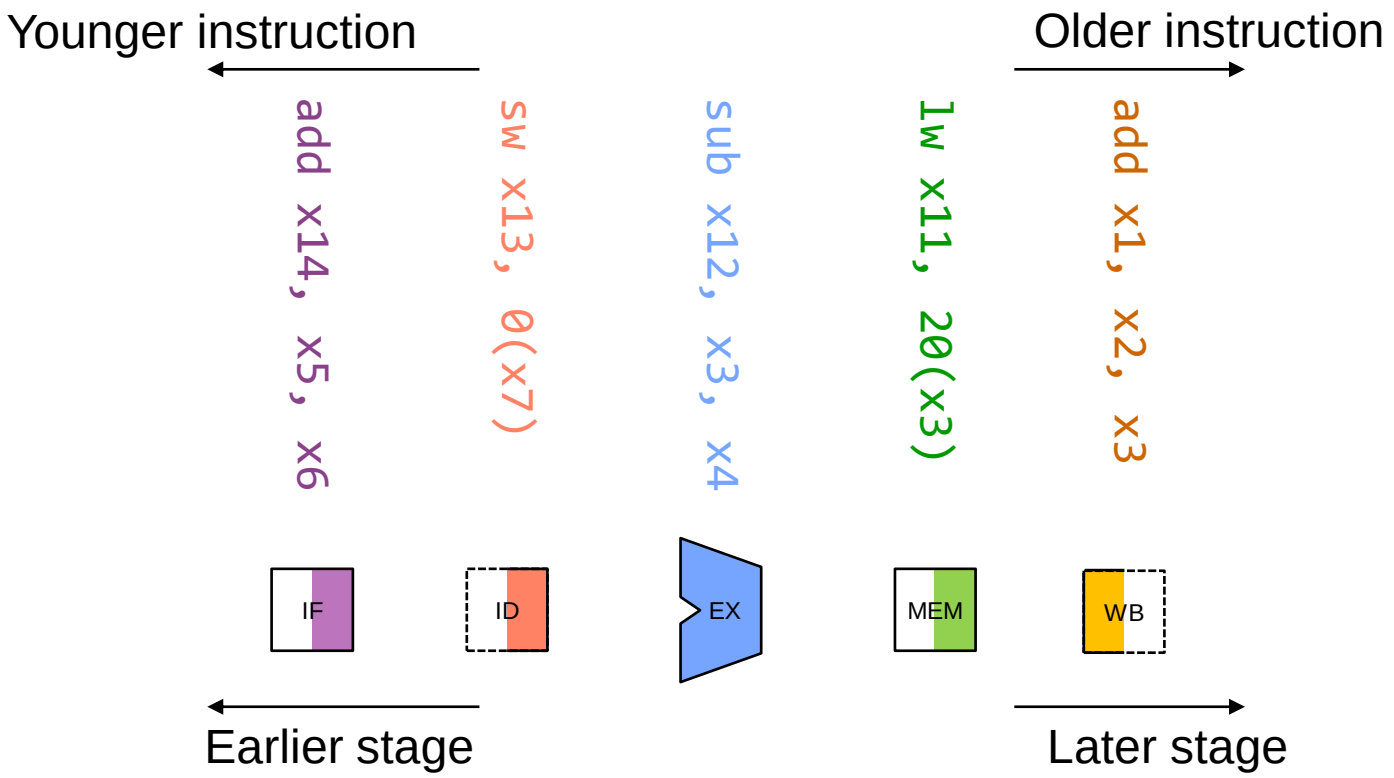
Simplified Representation of RISC-V Pipeline



Simplified Representation of RISC-V Pipeline



Instructions in a Pipeline



Hazards

- Hazard: Danger, risk, …

- Pipeline hazard:
  - A situation where the processor cannot process the next instruction in the following cycle.

Types of Hazards

1. Structural hazards
   - A required resource is busy
   - Multiple instructions are trying to use the same component in the same cycle

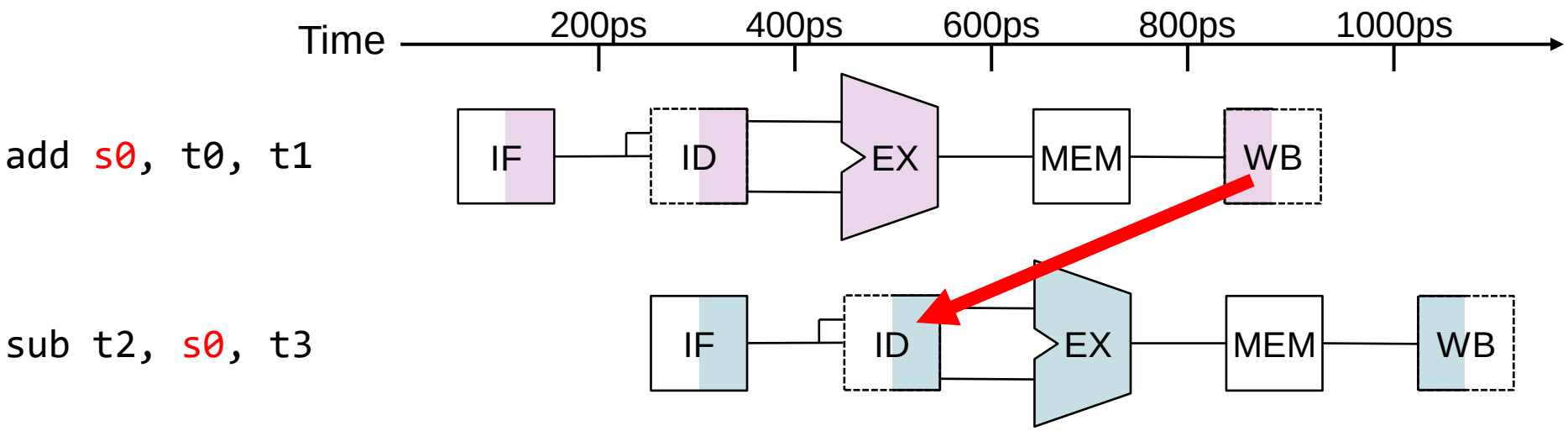
2. Data hazards
   - Need to wait for previous instruction to complete its data read/write
   - Data value is not ready for the next instruction.

3. Control hazards
   - Deciding on control action depends on previous instruction
   - Similar to data hazard, but related to instructions that changes PC

Data Hazards

- Using a register value updated by the preceding instruction

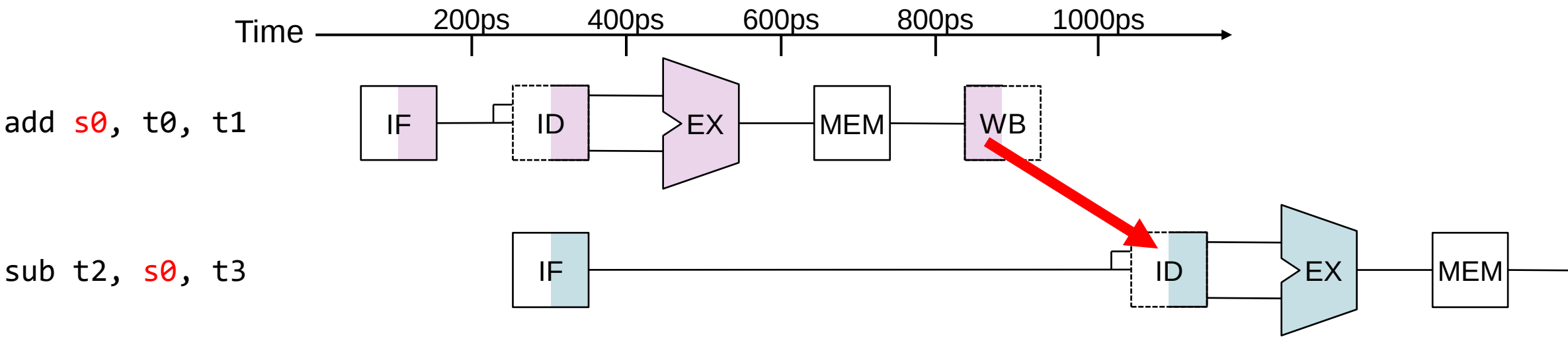
```
add     s0, t0, t1
sub     t2, s0, t3
```



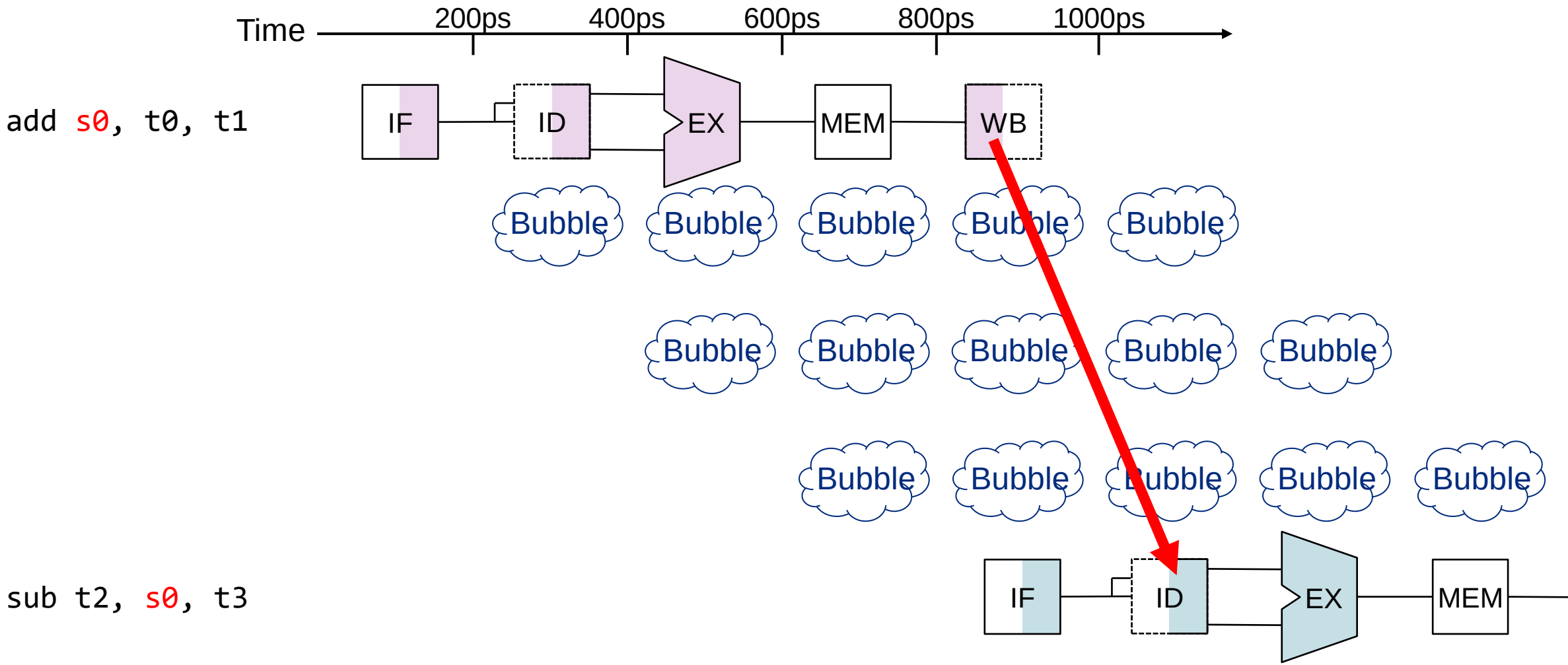
Data Hazards

- Using a register value updated by the preceding instruction

```
add     s0, t0, t1
sub     t2, s0, t3
```

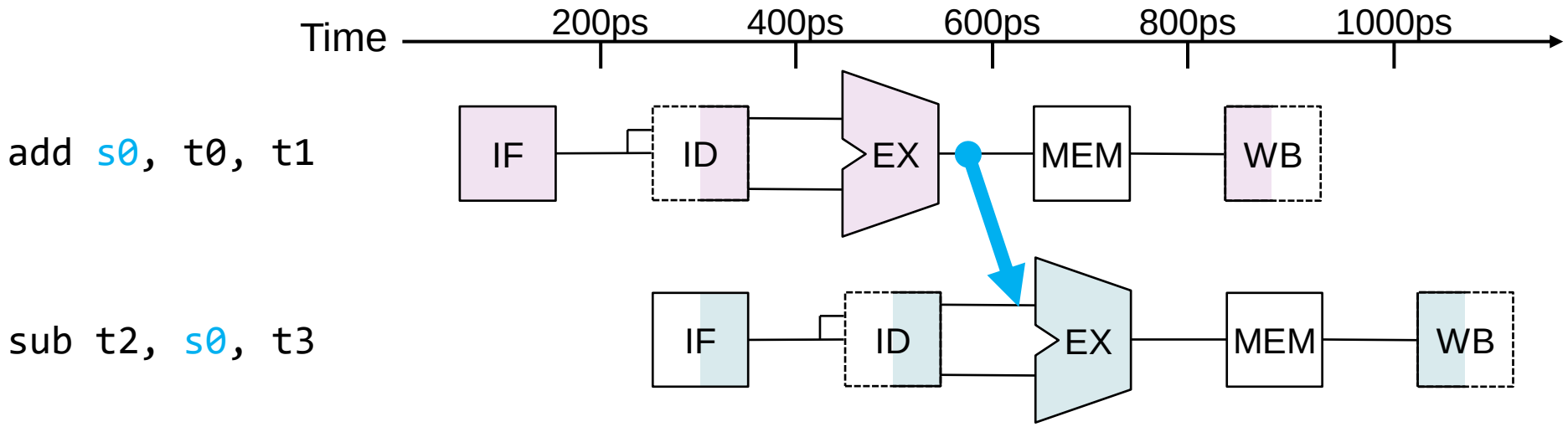


Pipeline Stall



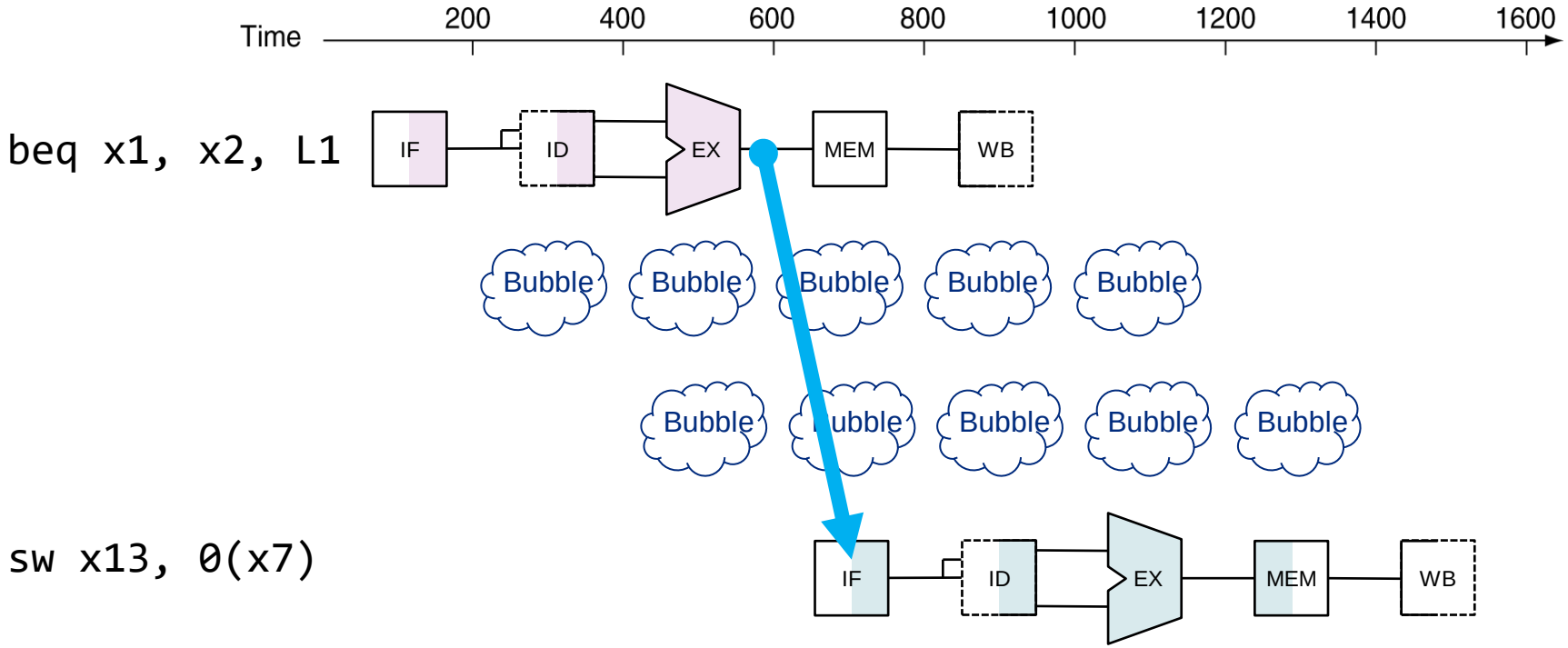
Forwarding

- Use result when it is computed
  - Don’t wait for it to be stored in a register
  - Requires extra connections in the datapath



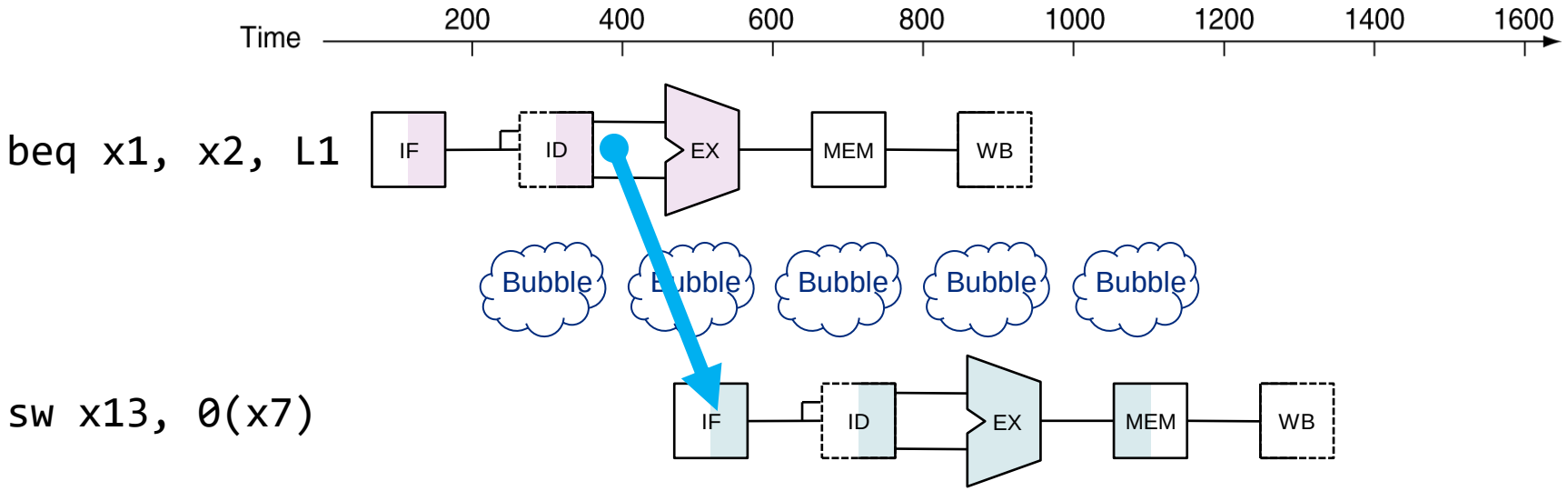
Control Hazards

- Branch determines flow of control
  - PC address should be detemined BEFORE the IF stage
  - Branch decision: EX stage (ALU)
  - Branch address: EX stage (Additional adder ALU)



Control Hazards

- The best we can do…
  - Add extra hardware resource in the ID stage
    - Compare the register values in the ID stage
    - Branch address calculation



  - Still need 1 cycle stall (Stall on branch)

Branch Prediction

- Longer pipelines can’t determine branch outcome early
  - Intel i7: 14 pipeline stages
  - Stall penalty becomes unacceptable

- Predict outcome of branch
  - Only stall if prediction is wrong

Pipeline Summary

- Pipelining improves performance by increasing instruction throughput
  - Executes multiple instructions in parallel
  - Each instruction has the same latency

- Subject to hazards
  - Structure, data, control