PIPELINE

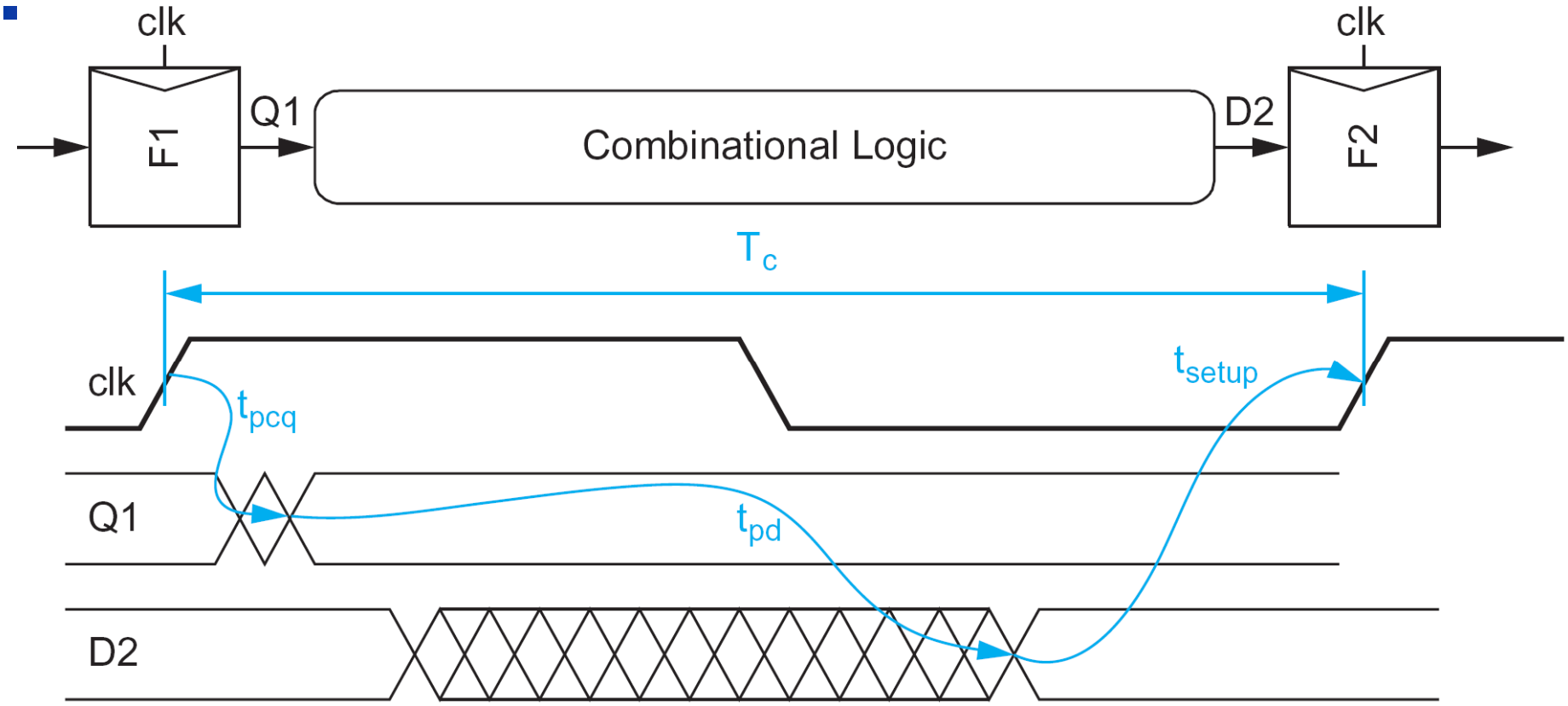
The following material has been adapted from:

Patterson, D. A. and Hennesy, J. L., ***Computer Organization and Design MIPS Edition: The Hardware/Software Interface***, 5th Edition, Morgan Kaufmann

Performance Background

- Clock frequency in ICs is determined by the longest propagation delay.
- Propagation delay is the time it takes for a signal to propagate from
  - An input to a flip-flop
  - A flip-flop to an output
  - A flip-flop to another flip-flop

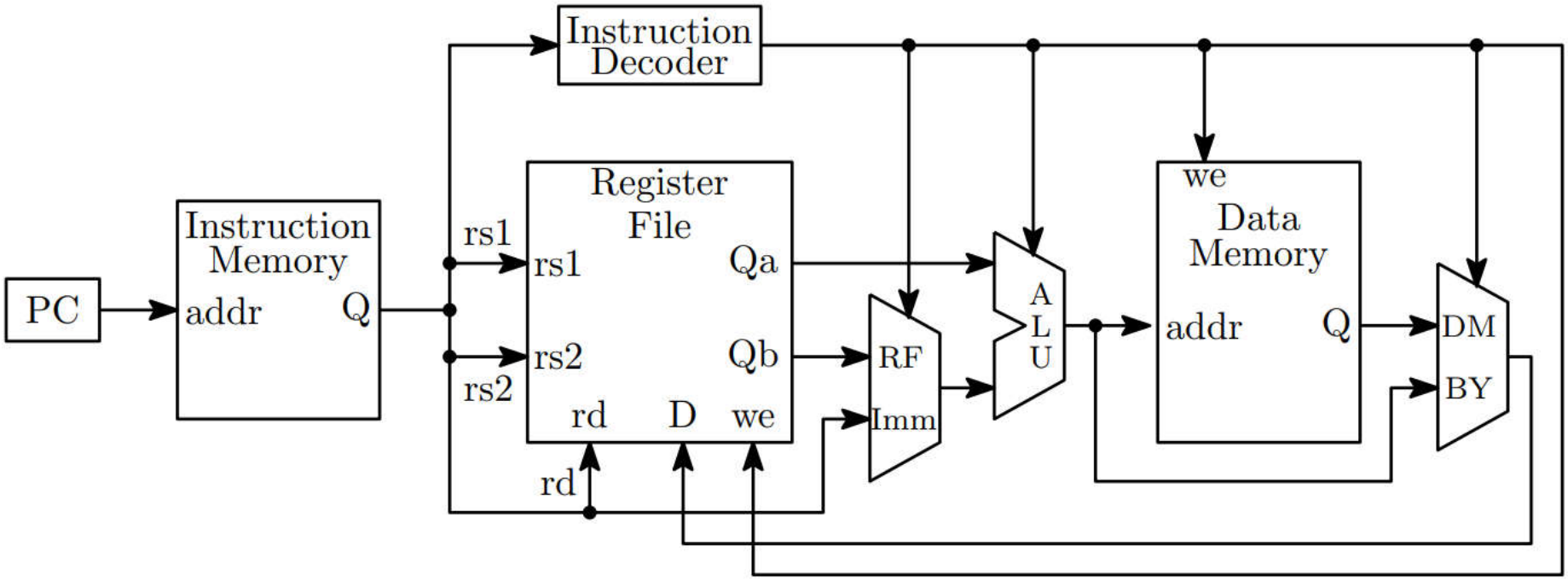
Performance Background



- Clock period should be larger than largest propagation delay
  - Tc > tpd

Performance Background

- Which is the critical path in this simplified MIPS block diagram?



Performance Issues

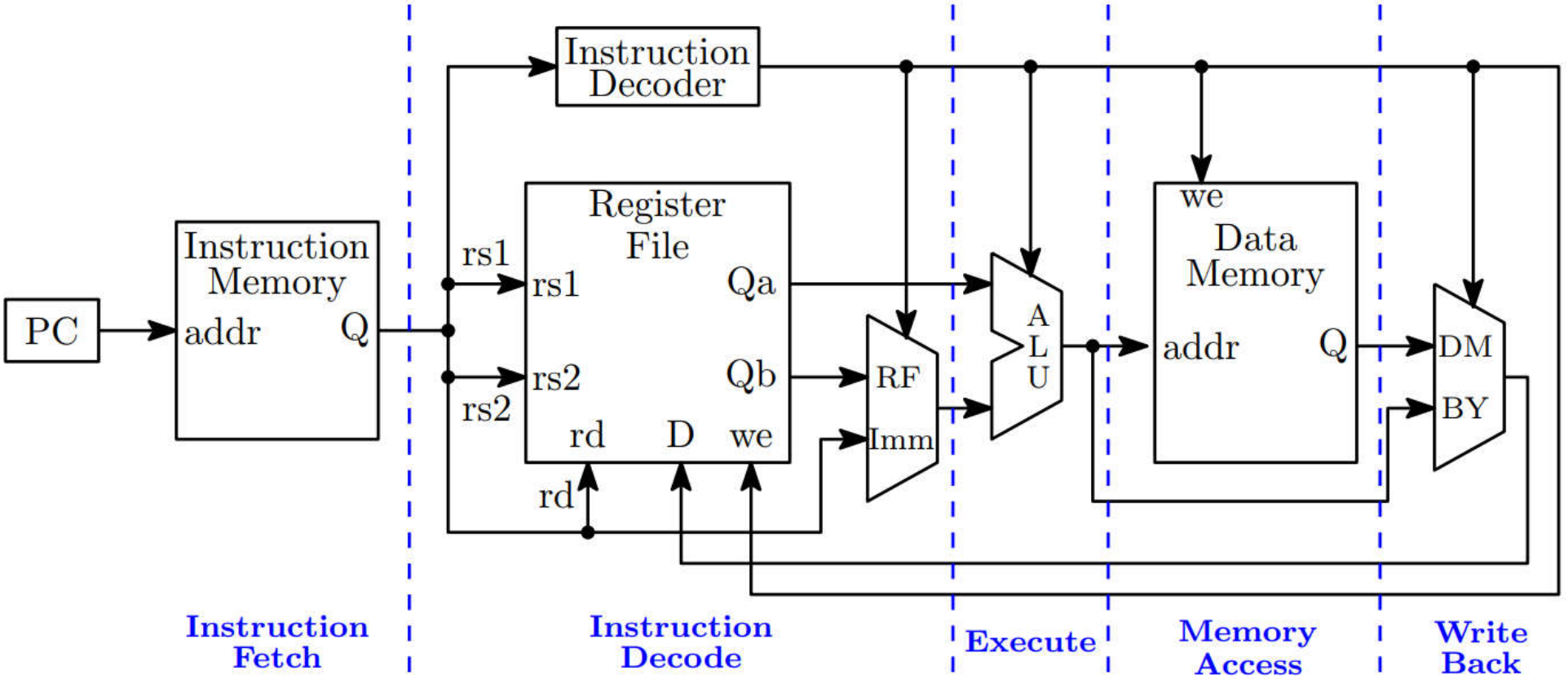
- Longest delay determines clock period
  - Critical path: load instruction
  - Instruction memory → register file → ALU → data memory → register file
- This critical path limits the clock frequency and the number of instructions per second that we could theoretically perform

Performance Issues

- How could we improve our MIPS performance?
  - We could start by splitting the critical path into smaller paths.

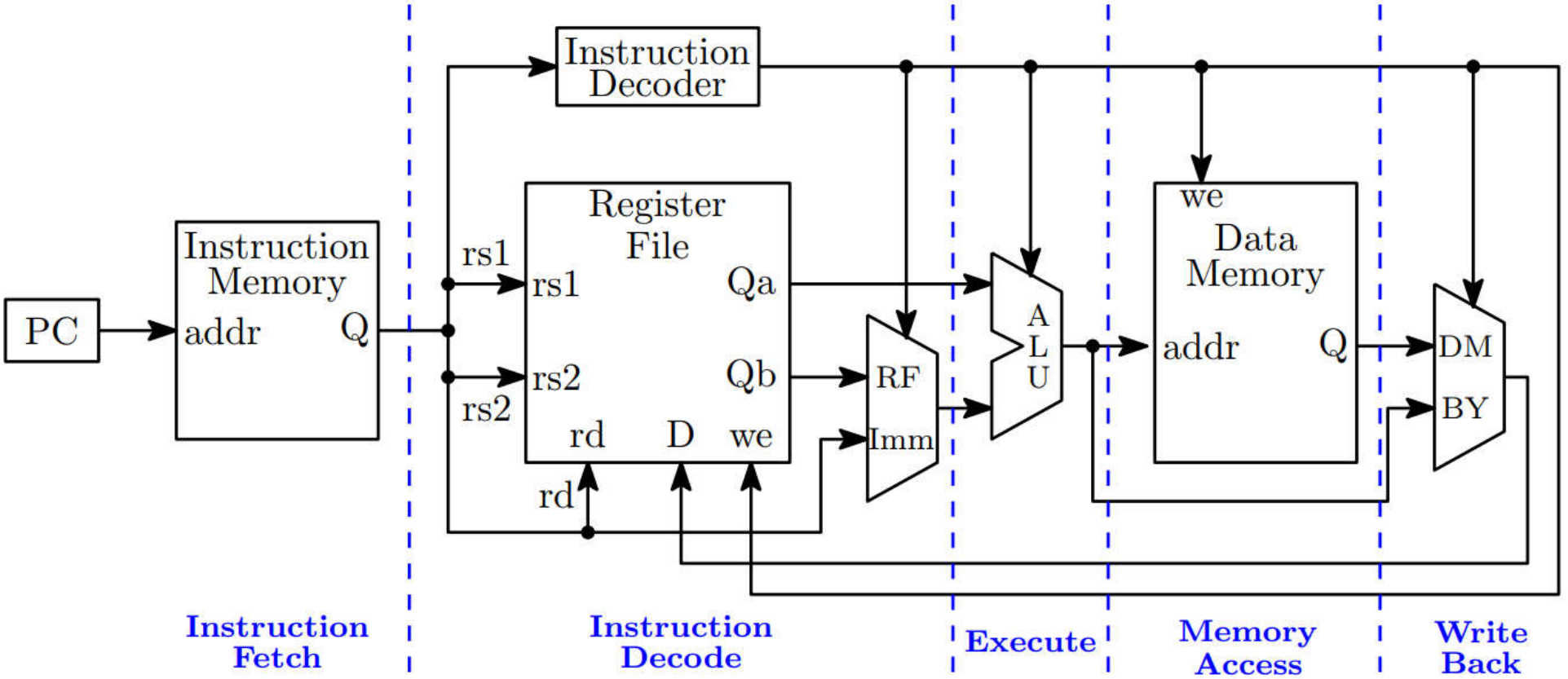
Performance Issues

- We could split our single-cycle execution into different stages.



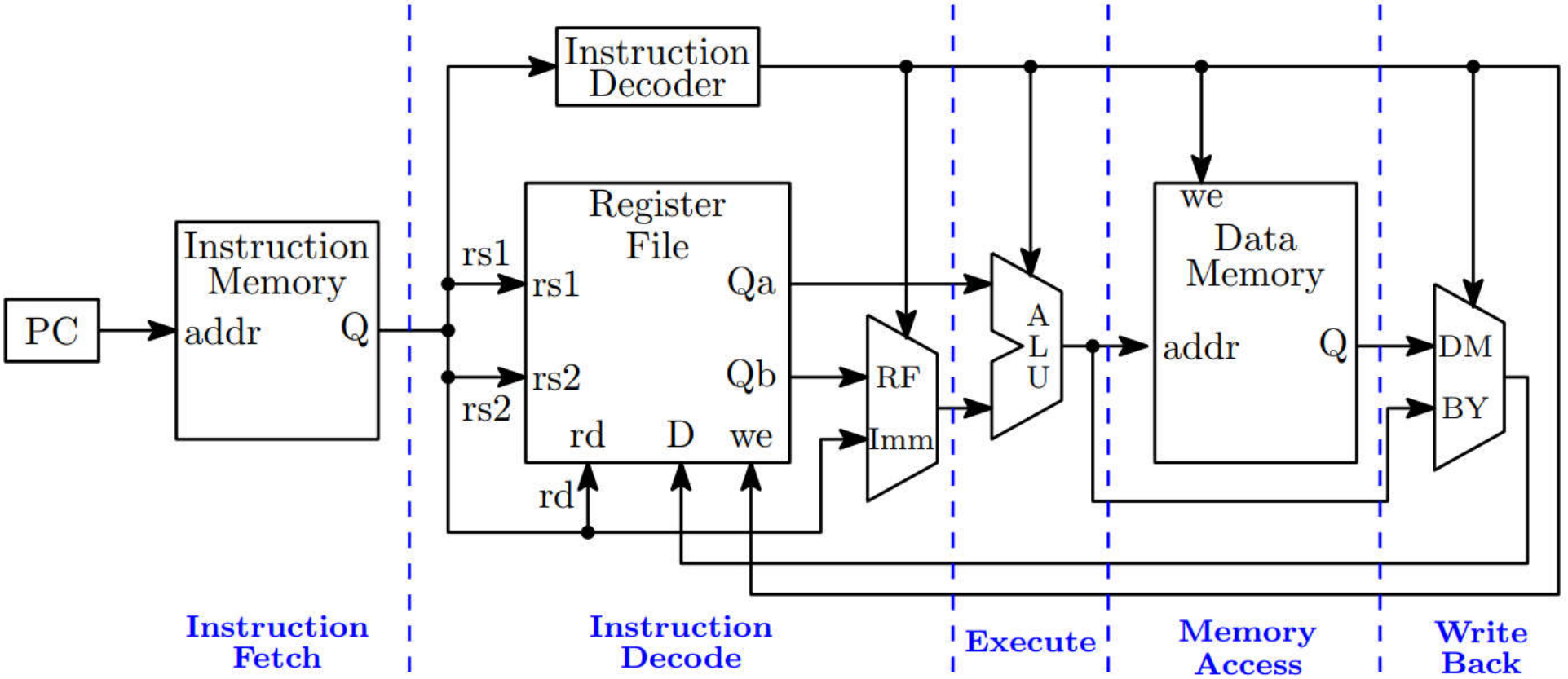
Performance Issues

- This technique is called Pipelining



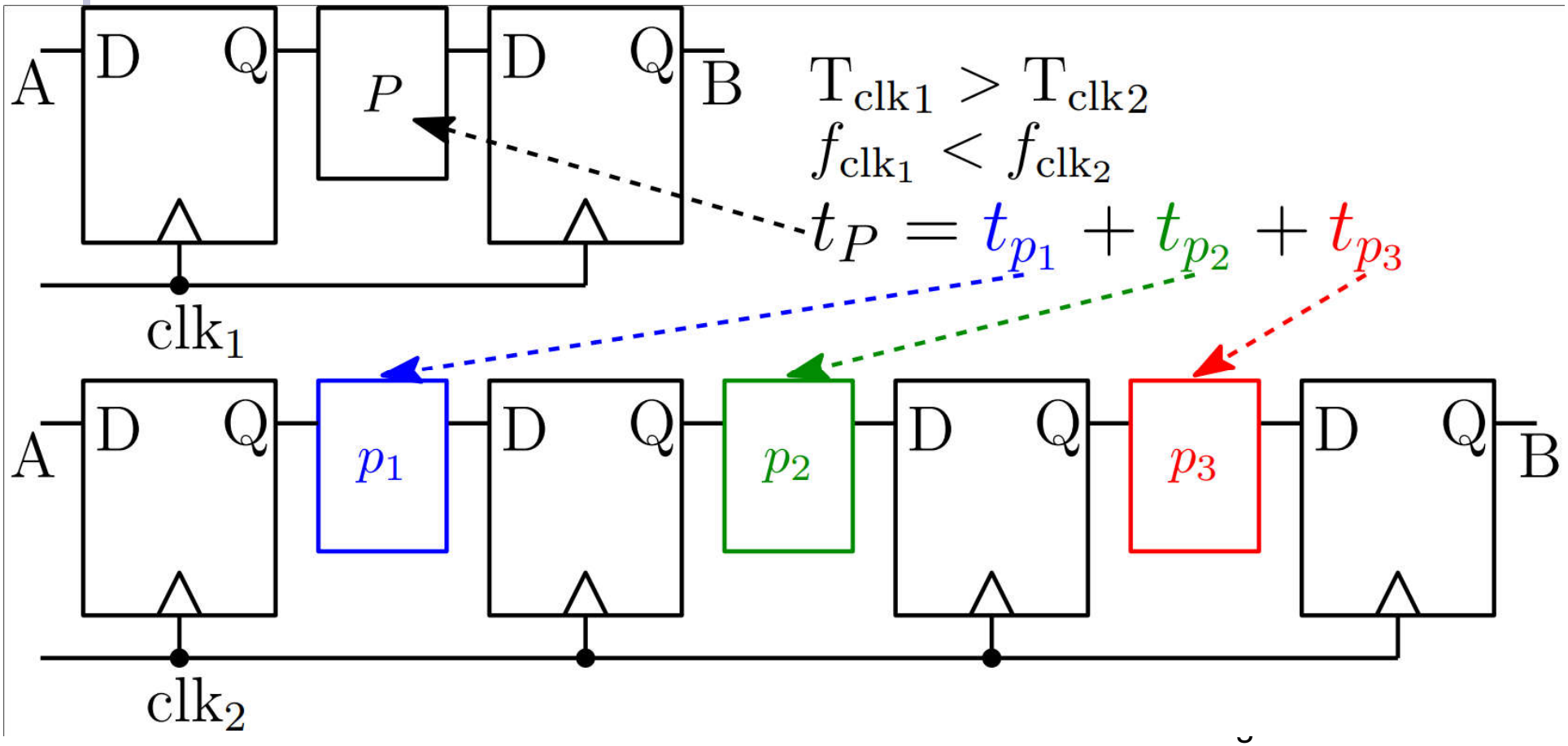
Performance Issues

- This technique is called Pipelining



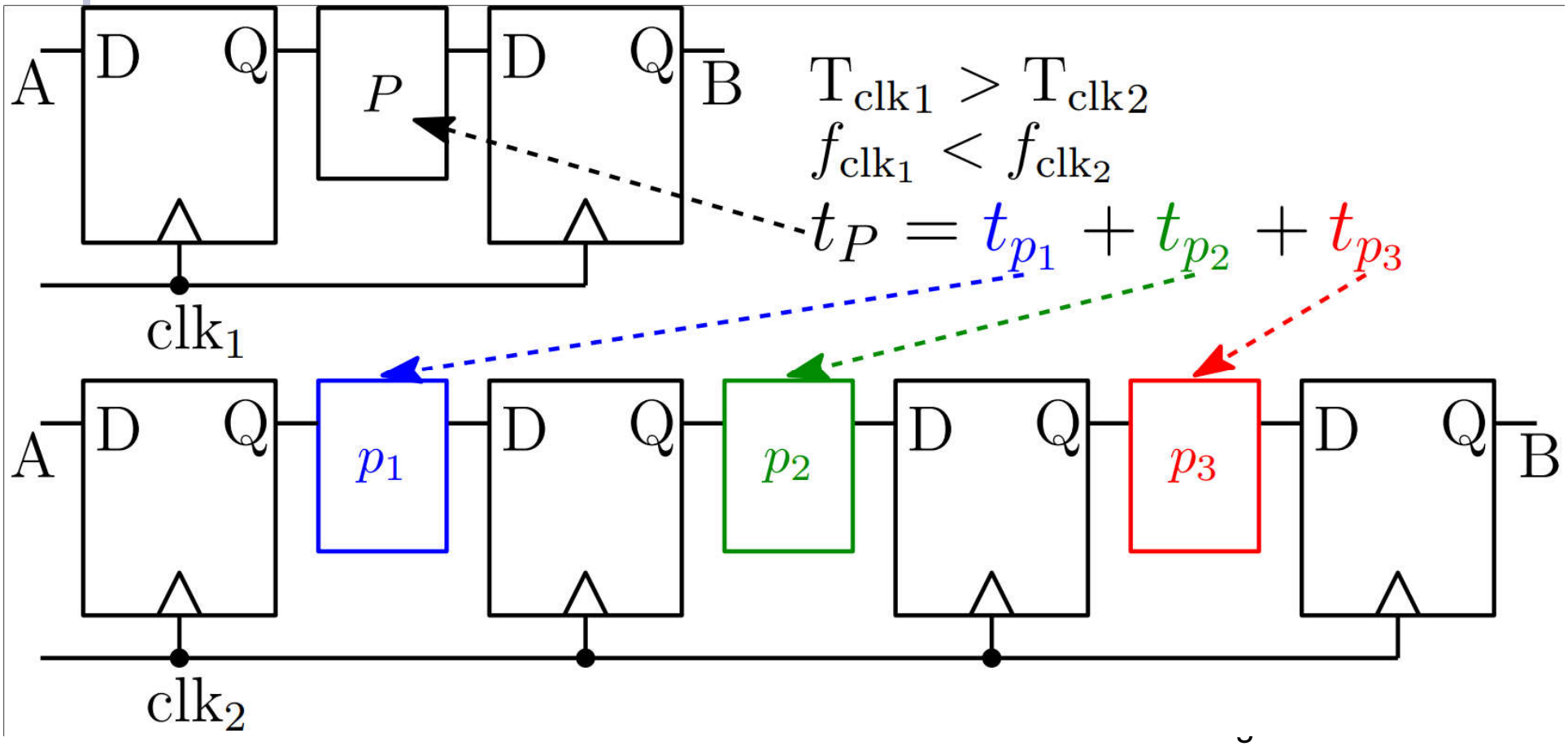
Pipeline

- Pipelining consist in breaking down a single task into several stages.



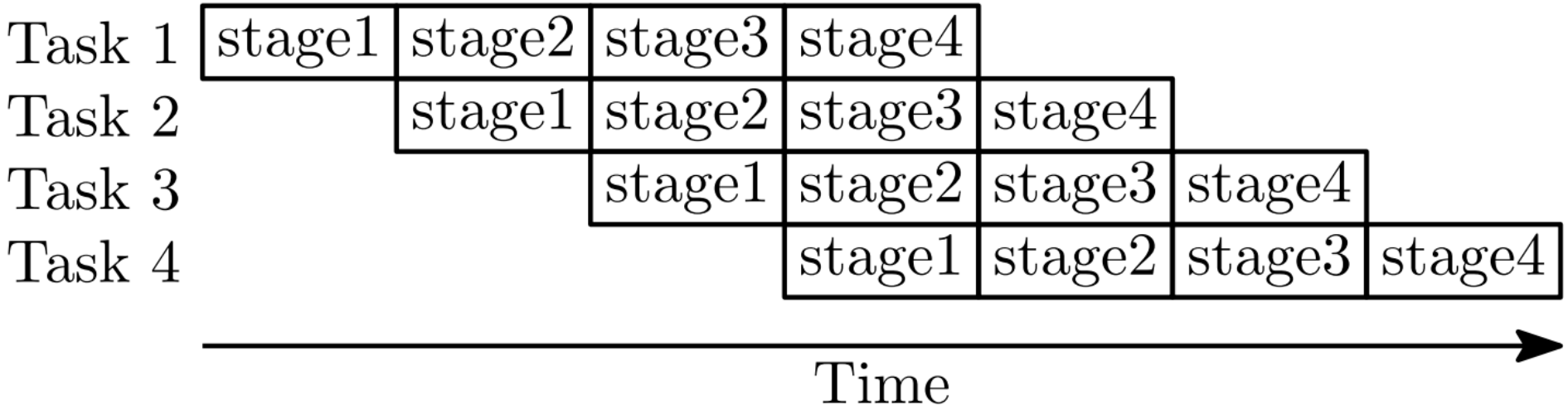
Pipeline

- Task from A to B takes the same time in both cases. What's the advantage?



Pipeline

- We can perform tasks in parallel

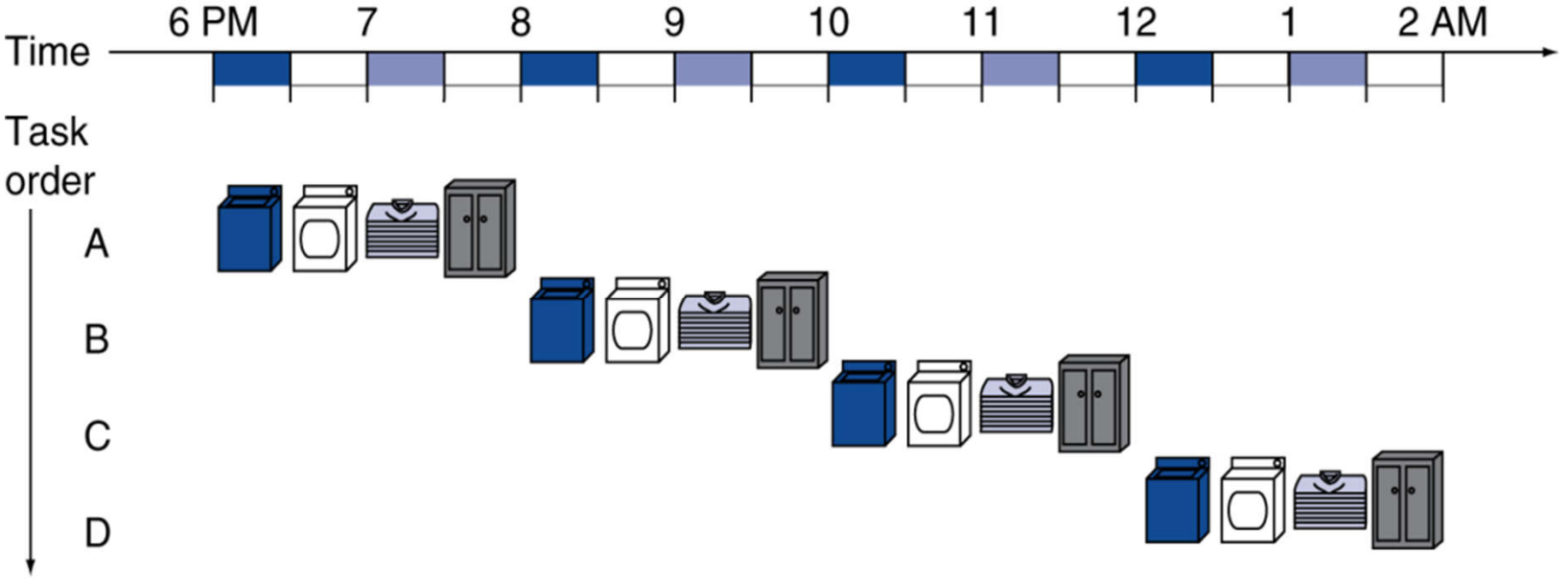


Pipelining Analogy

- Assume you work in a laundry shop.
- You divide your job into the following stages.
  - Wash
  - Dry
  - Fold
  - Store
- Assume each stage takes 30 minutes to complete

Pipelining Analogy

- How long would it take you to complete a single laundry request?
- 30min x 4 stages = 2 hours



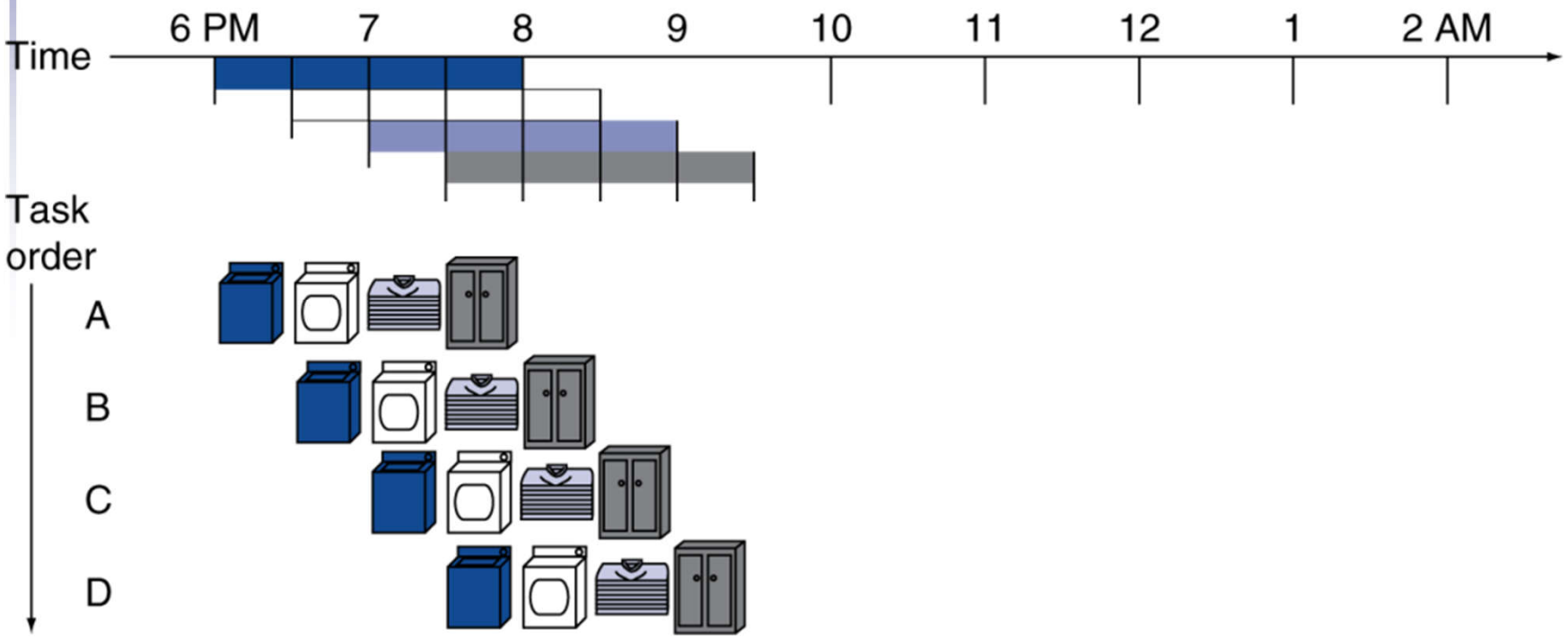
- What about 4 laundry request? – 8 hours

Pipelining Analogy

- Start a new load right after the washing machine becomes available.
- Dry first load and wash second load at the same time.
- Fold first load, dry second load and wash third load at the same time
- Continue with this principle until you complete all four loads

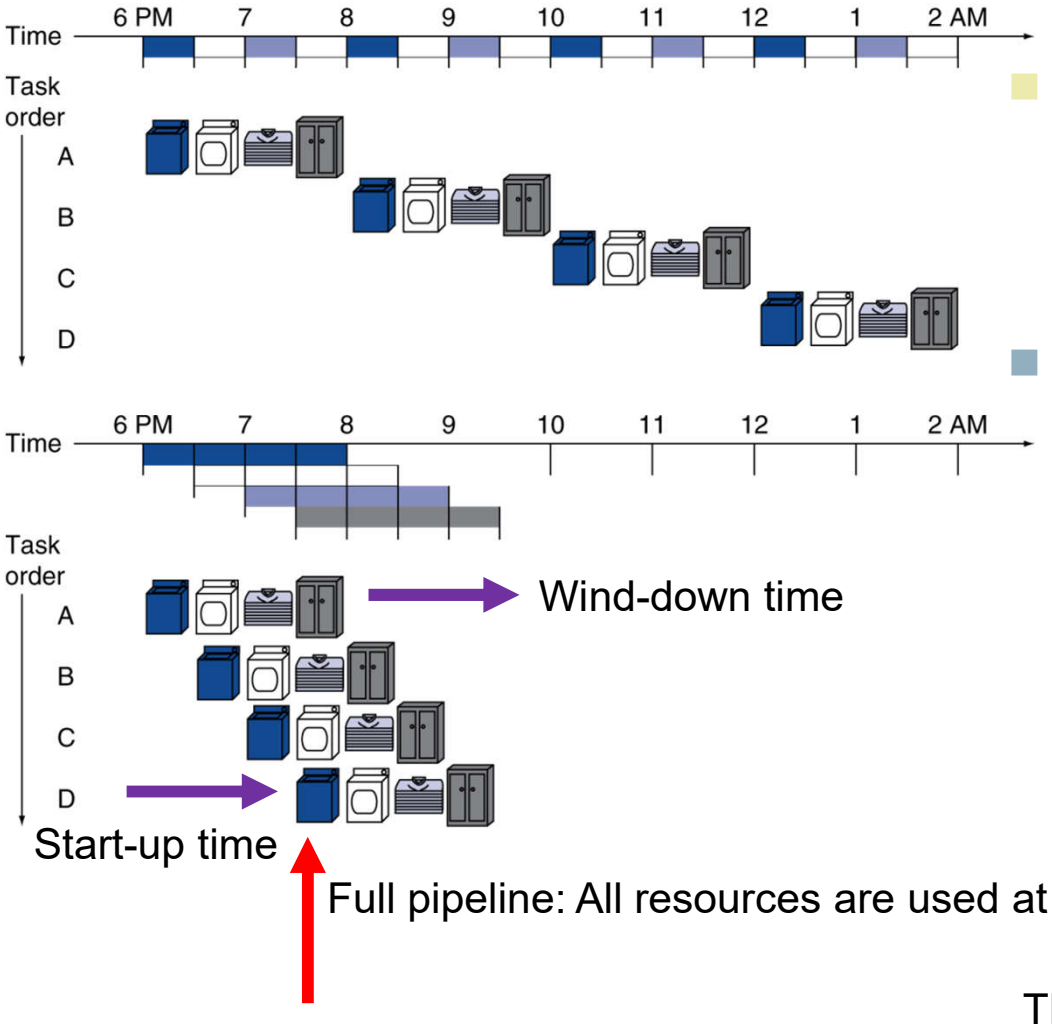
Pipelining Analogy

- 4 loads now takes 3.5 hours



Pipelining Analogy

- Improved performance



- Four loads:
  - Speedup = 8/3.5 = 2.3
- What about infinite number of loads?

Pipelining Speedup Equation

m is the number of tasks
n is the number of stages per task
t is the time required to complete a stage

$$T_{seq} = m \cdot \sum_{i=1}^{n} t_i$$

If all n stages take the same time t, then

$$T_{seq} = m \cdot n \cdot t$$

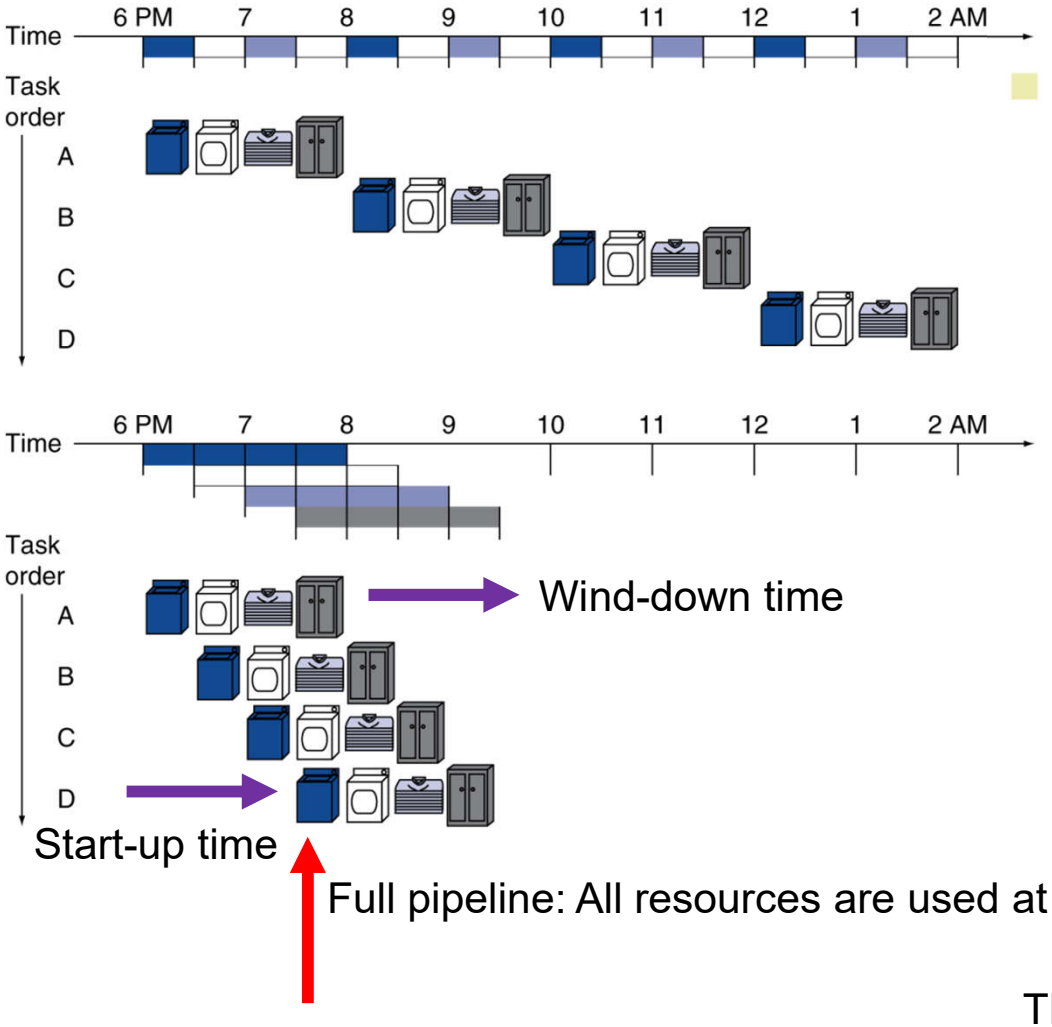
$$T_{pipe} = (m-1) \cdot t + n \cdot t$$

$$S = \frac{T_{seq}}{T_{pipe}} = \frac{m \cdot n \cdot t}{(m-1) \cdot t + n \cdot t} = \frac{m \cdot n}{(m-1)+n}$$

$$\lim_{m \to \infty} \frac{m \cdot n}{(m-1)+n} = n$$

Pipelining Speedup

- For the laundry analogy



- Infinite loads:
  - Speedup ≈ 4

MIPS Pipeline

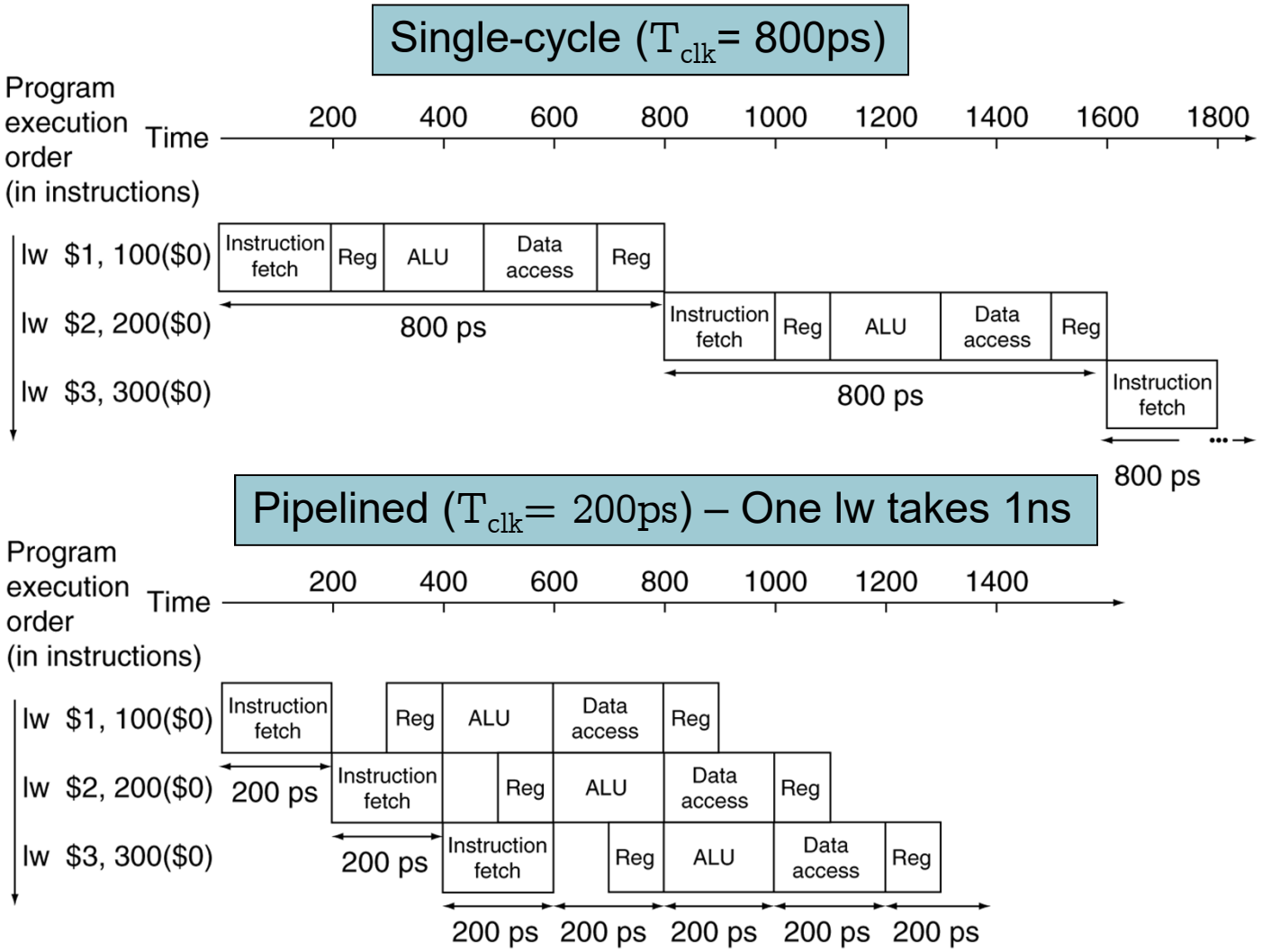
- Five stages, one step per stage
  1. IF: Instruction fetch from memory
  2. ID: Instruction decode & register read
  3. EX: Execute operation or calculate address
  4. MEM: Access memory operand
  5. WB: Write result back to register

Pipeline Performance

- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- Compare pipelined datapath with single-cycle datapath

Instr	Instr 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 stages take the same time
  - Time between instructions$_{pipelined}$
    = $\frac{\text{Time between instructions}_{nonpipelined}}{\text{Number of stages}}$
- If not balanced, speedup is less
- Speedup due to increased throughput
  - Latency per instruction does not decrease

Pipelining and ISA Design

- MIPS ISA designed for pipelining
  - All instructions are 32-bits
    - Easier to fetch and decode in one cycle
  - 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
  - Alignment of memory operands
    - Memory access takes only one cycle