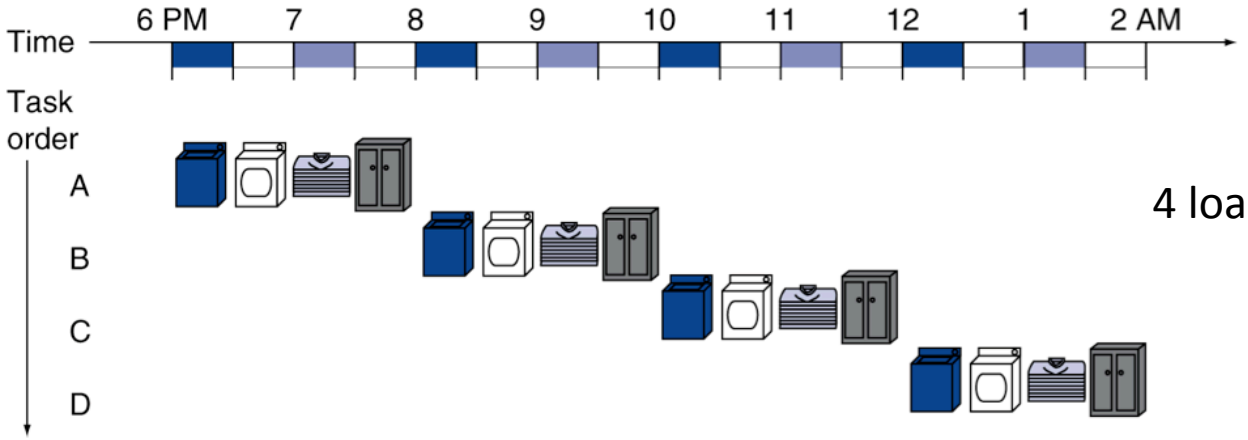
Pipelining Analogy

Doing multiple loads of laundry

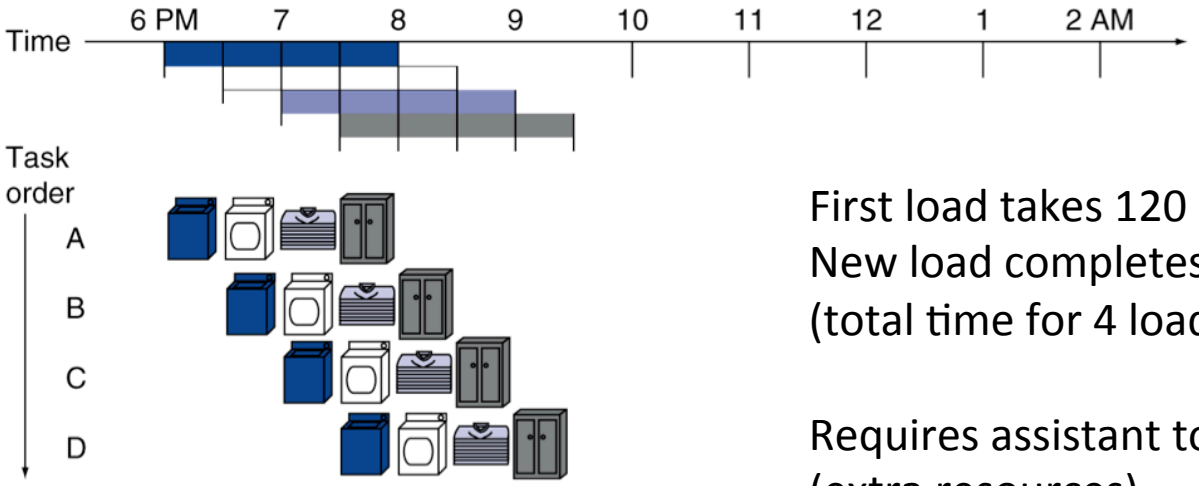


4 loads take 4 *120 = 480 minutes

1. Use washer for next dirty load of clothes (30 min.)
2. When done, use dryer to dry clothes (30 min.)
3. When done, fold the clean dry clothes (30 min)
4. When done, put the clothes away (30 min)

Pipelined Laundry Analogy

Parallelism improves performance by overlapping

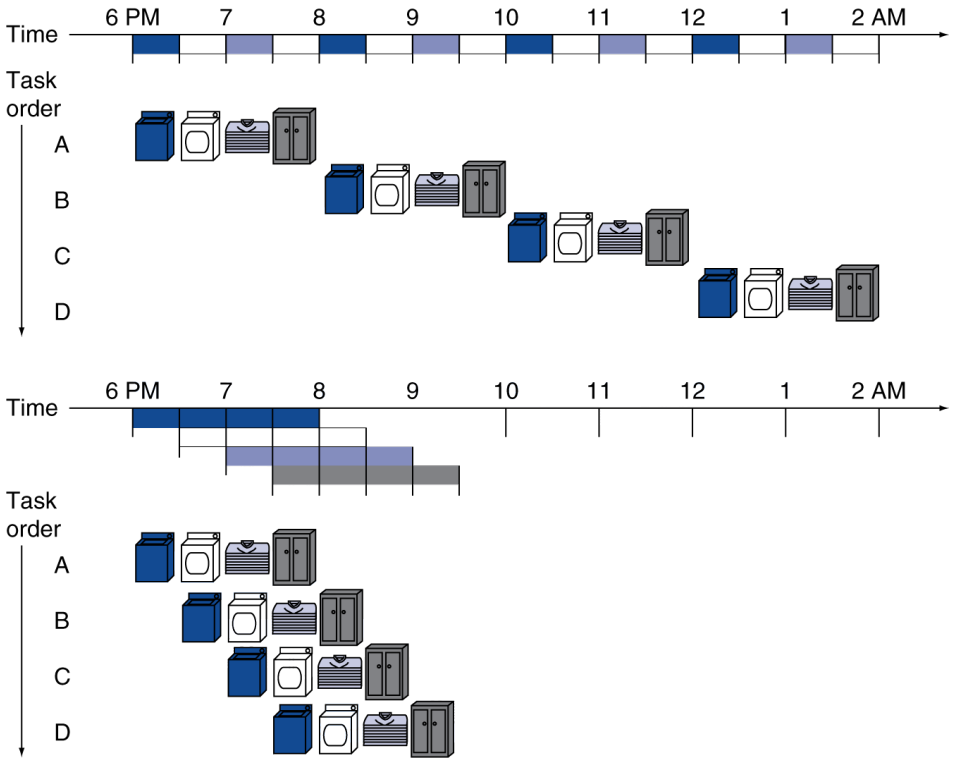


First load takes 120 minutes
New load completes every 30 min thereafter
(total time for 4 loads = 120 + 90 = 210 min)

Requires assistant to work in parallel
(extra resources)

1. Wash 1st load
2. When done place 1st into dryer and wash 2nd
3. When done, fold 1st load, dry 2nd and wash 3rd
4. When done, put away 1st, fold 2rd, dry 3rd and wash 4th

Pipelining improves throughput



- Four loads:
  - Speedup
    = 480/210 = 2.3
- Non-stop:
  - Speedup
    = 120n/(120+30n) ≈ 4
    = number of stages

Each load still takes 120 minutes (2 hours)

But more loads are completed per hour

MIPS Pipelining

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

MIPS ISA & Pipelining

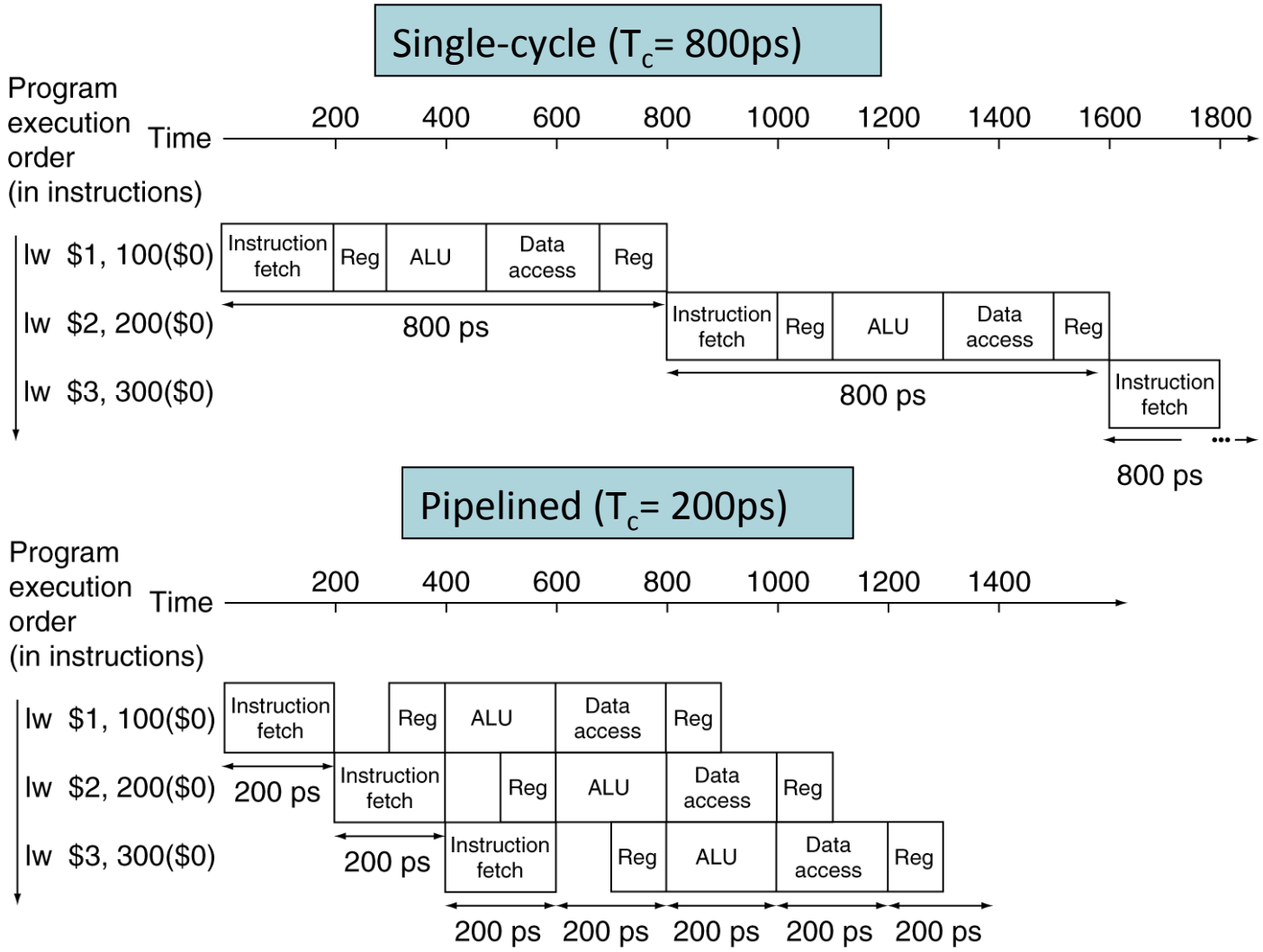
- All instructions are 32-bits
  - Easier to fetch and decode in one cycle
  - c.f. x86: 1- to 17-byte instructions
- Few and regular instruction formats
  - Can decode and read registers in one step
- Load/store addressing
  - Can calculate address in 3rd stage, access memory in 4th stage
- Alignment of memory operands
  - Memory access takes only one cycle

MIPS Pipelining

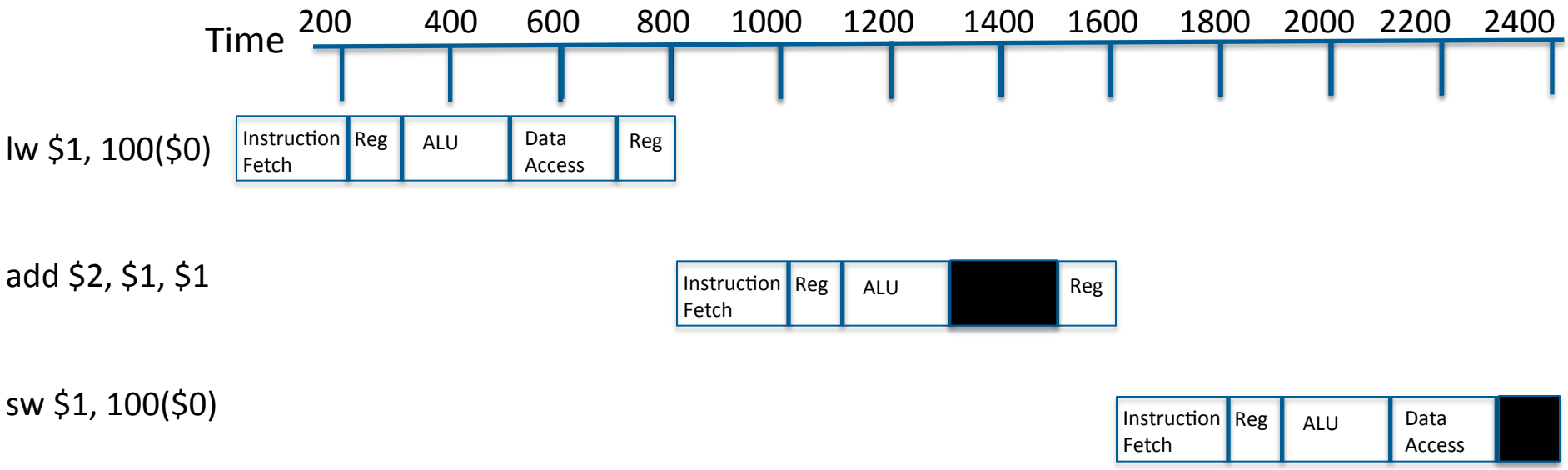
- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- Compare pipelined datapath with single-cycle datapath

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

MIPS Pipelining



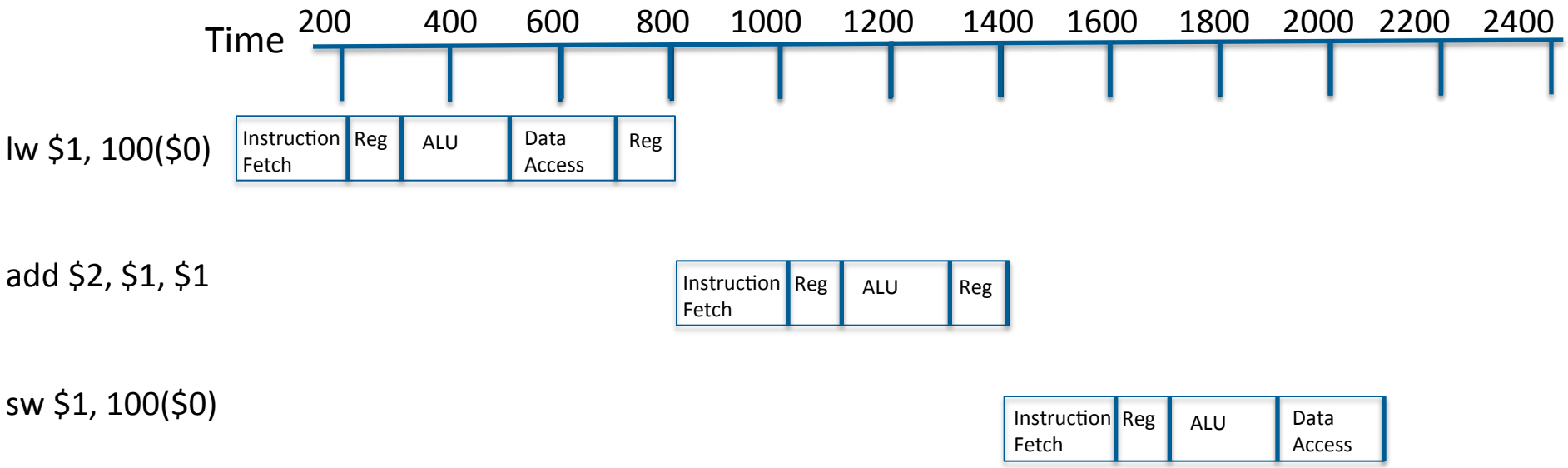
Single-Cycle Performance



Single-cycle (T_c= 800ps)
T_{total} = 2400 ps

Cycle time matches longest instruction

Multi-Cycle Performance



Multi-cycle with variable cycle time

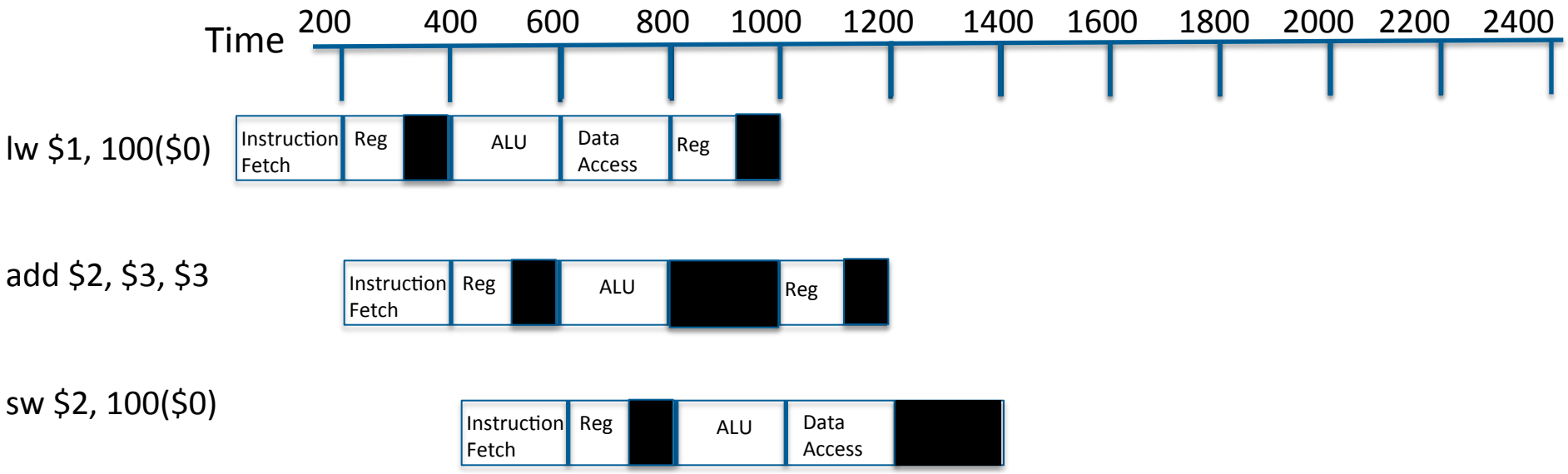
T_{total} = 800 + 600 + 700 =2100 ps

Multi-cycle with fixed cycle time (200ps)

T_{total} = 1000 + 800 + 800 =2600 ps

Cycle time matches longest step

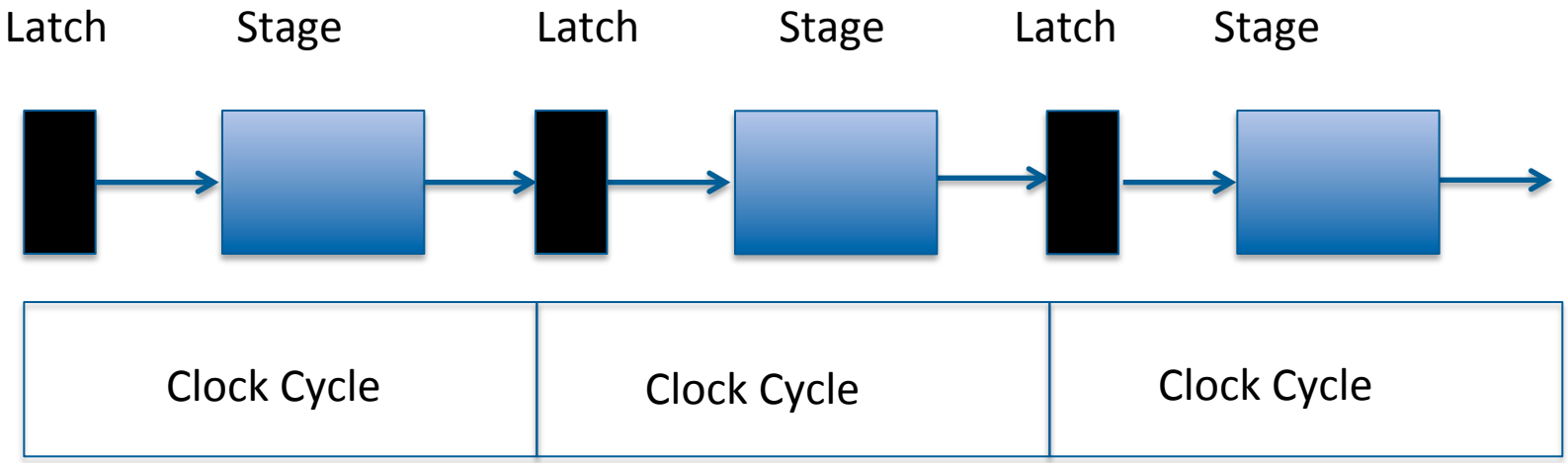
Pipelined Performance



Pipelined-cycle (T_c= 200ps)
T_{total} = 1000 + 200 + 200 = 1400 ps

Cycle time matches longest step

Pipelining



- Although it is possible to divide the work in to approximately equal stages, they are unlikely to be exactly equal
- Latches are added to synchronize the stages
- The latches themselves may add a small latency to each stage (folded into cycle time)

Pipeline Speedup

$T_{pipelined} = T_{nonpipelined} \div$ Number of stages

Speedup ≈ N assuming:

- independent instructions
- balanced pipeline stages

Dependencies cause stalls and reduce speedup

It is throughput that is increased

Latency or fill time is 5 cycles

- A new instruction completes for each cycle thereafter
- Each instruction still takes 5 cycles