EE-421: Digital Systems Design

Pipelining

Dr. Rehan Ahmed [rehan.ahmed@seecs.edu.pk]

NUST

SCHOOL OF ELECTRICAL ENGINEERING &
COMPUTER SCIENCE (SEECS)

What is Pipelining?

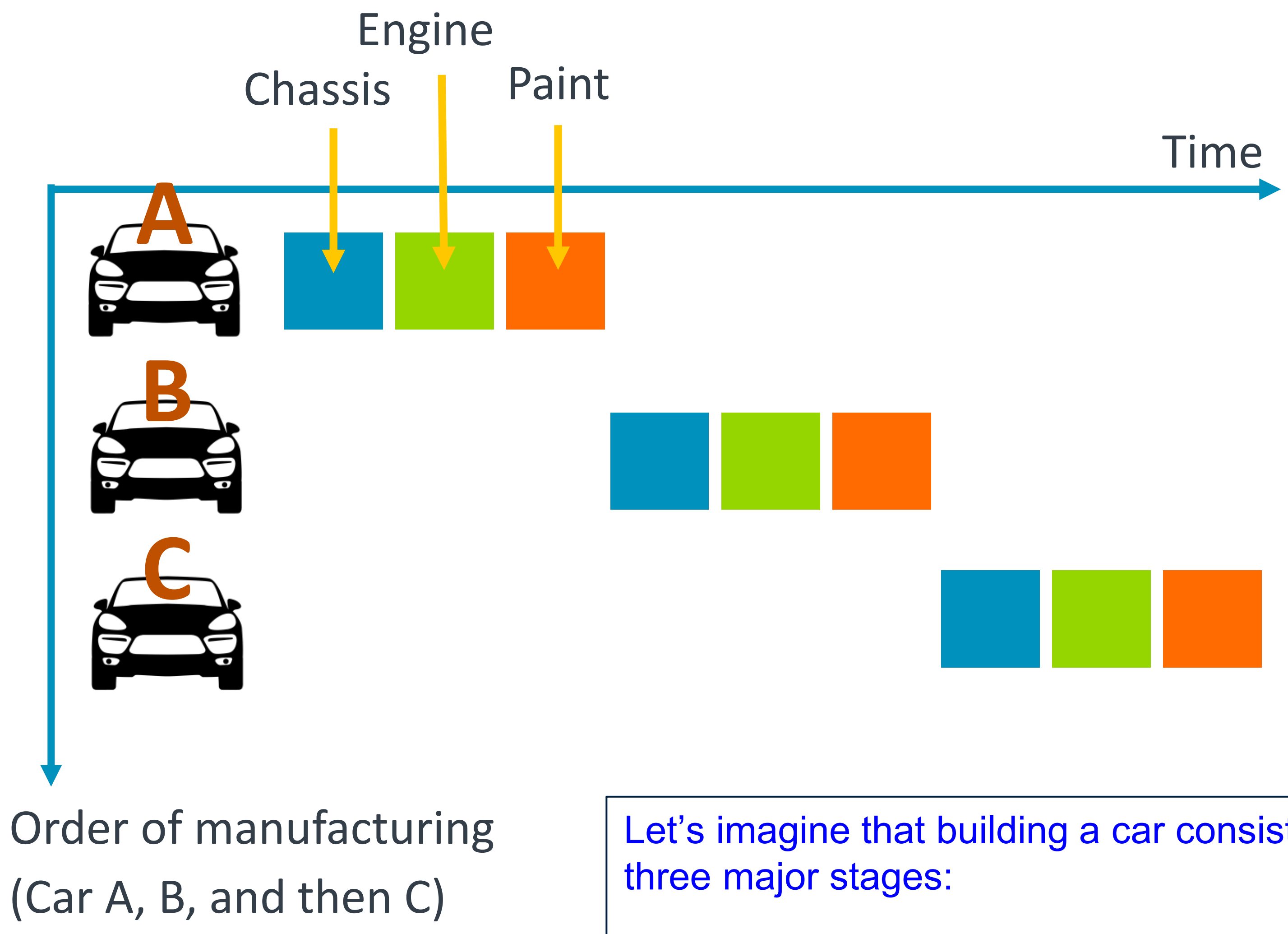
- Basic Idea:
  - We arrange for the different phases of execution to be overlapped.
  - We aim to exploit "temporal" parallelism.

- How are latency and throughput affected?
  - Let's understand through examples (next)

Pipelining: Assembly Line Example

Volkswagen Beetle Assembly Line
(By Alden Jewell, license: CC BY 2.0)

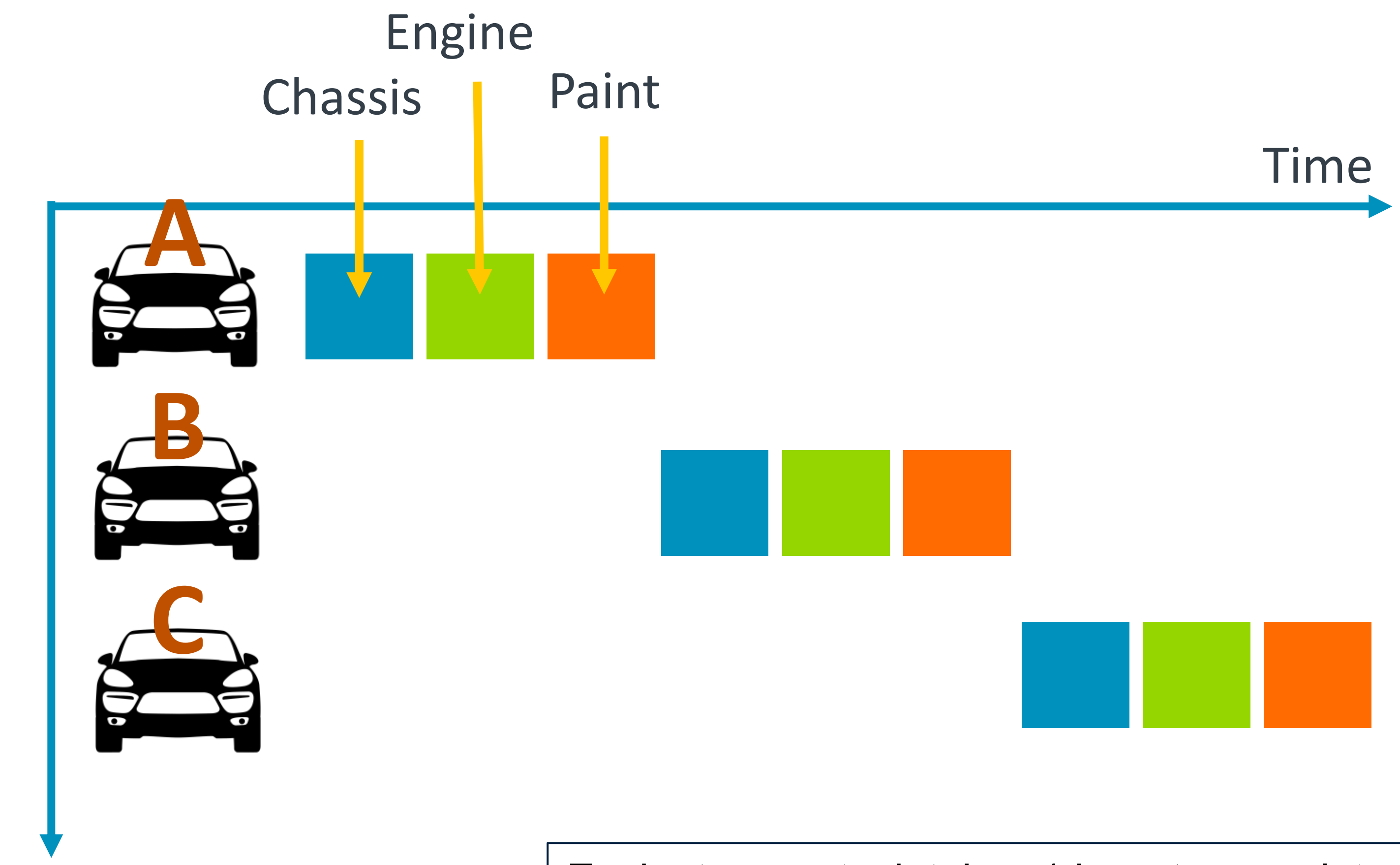
Sequential Car Manufacturing



Let's imagine that building a car consists of three major stages:

(1)constructing the chassis
(2)installing the engine
(3)painting.

Sequential Car Manufacturing



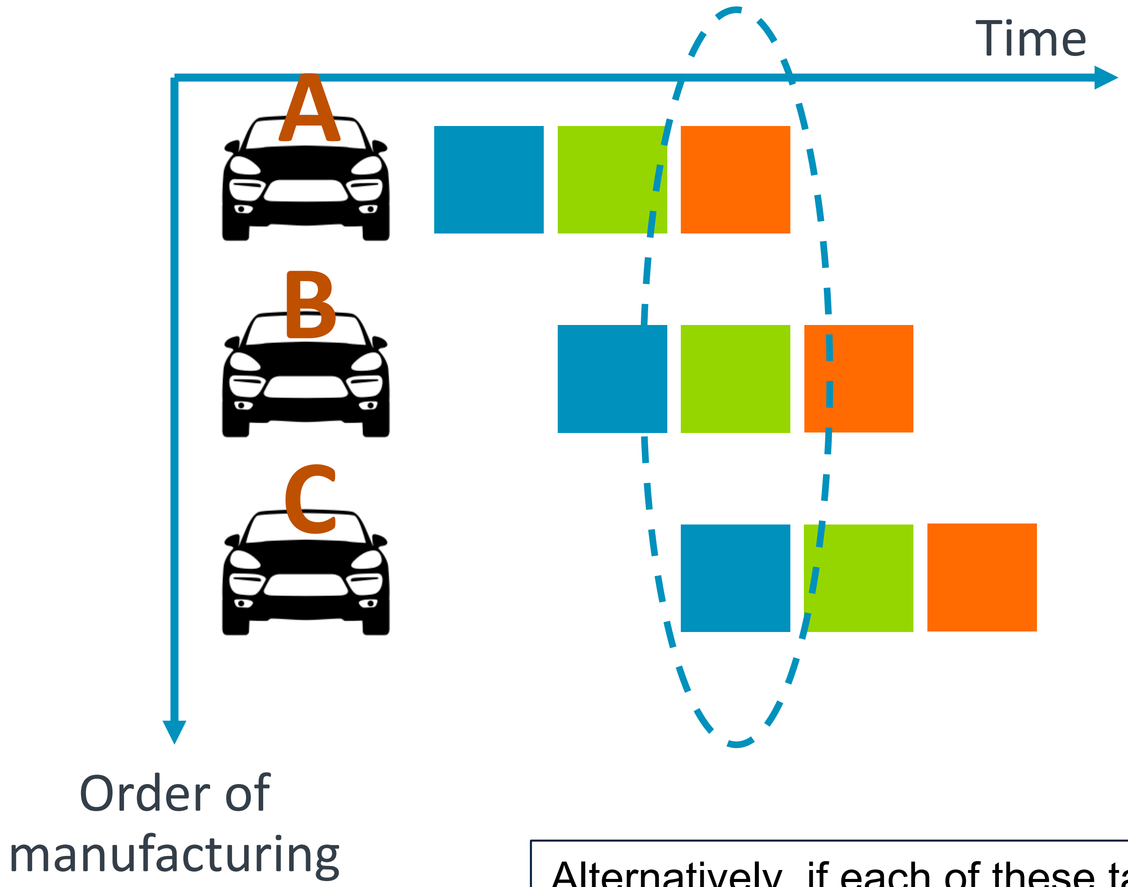
Order of manufacturing

(Car A, B, and then C)

Each stage or task takes 1 hour to complete.

We could complete each of these steps before starting to build a new car. A new car will be produced every 3 hours. Our throughput will be 8 cars per day, assuming our factory runs 24 hours.

Pipelined Car Manufacturing

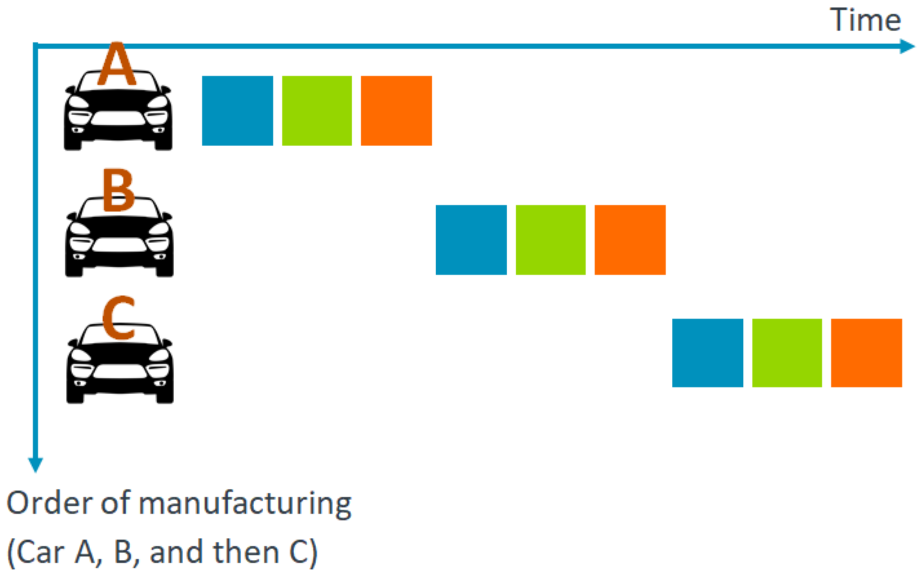


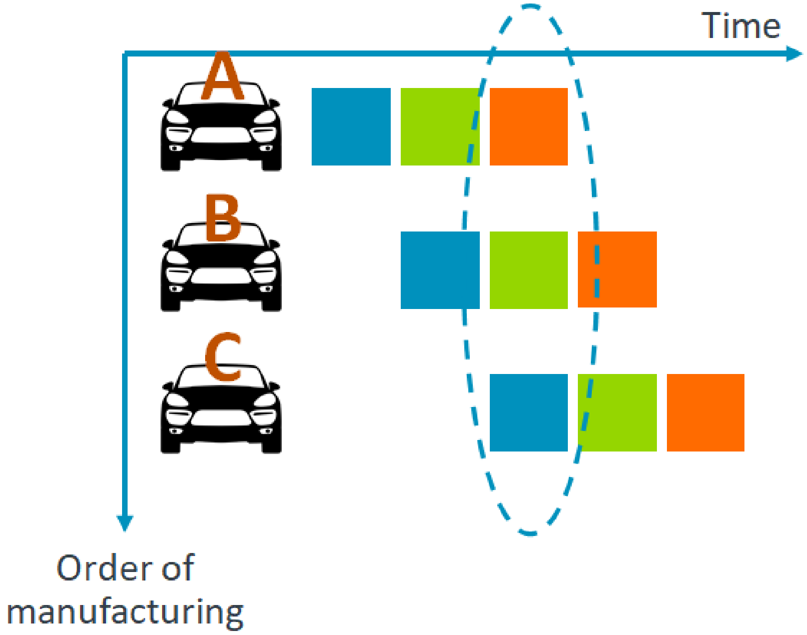
Alternatively, if each of these tasks is completed at a different station, we could pipeline the cars construction. Now, the construction of a new car is started as soon as the first task is complete. Once our pipeline is full, a car will be completed every hour. Our throughput is now 24 cars per day.

Pipelining Lessons

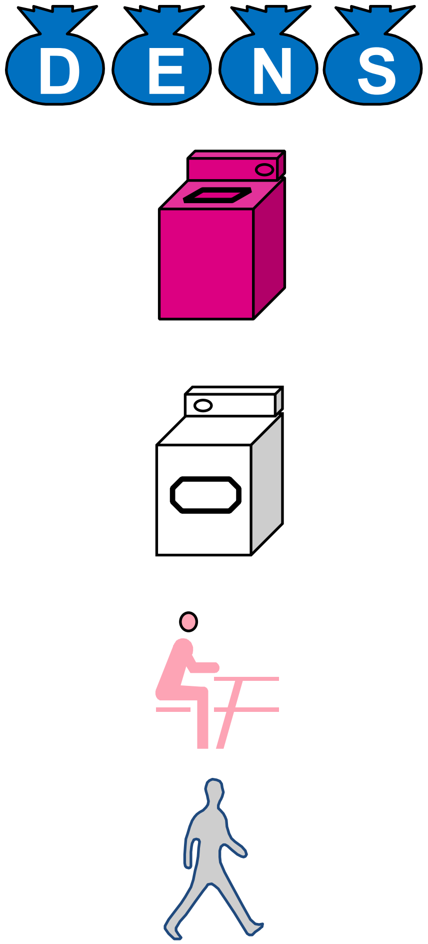
- Of course, a single car is not produced more quickly, **latency** may actually **increase** (due to unbalanced stages).

- Nevertheless, the rate at which cars are produced, our **throughput,** is much **higher** than if we constructed a single car at a time.



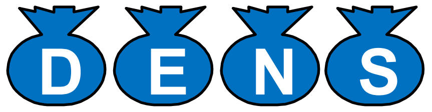


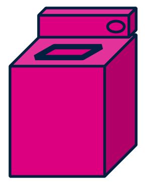
Pipelining: Laundry Example

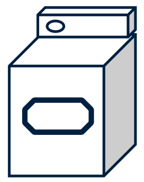


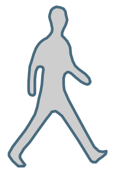
Pipeline Analogy: Doing Laundry

- Damon, Emaan, Nick, and Steven each have one load of clothes to wash, dry, fold, and put away
  - Washer takes 30 minutes
  - Dryer takes 30 minutes
  - "Folder" takes 30 minutes
  - "Stasher" takes 30 minutes to put clothes into drawers

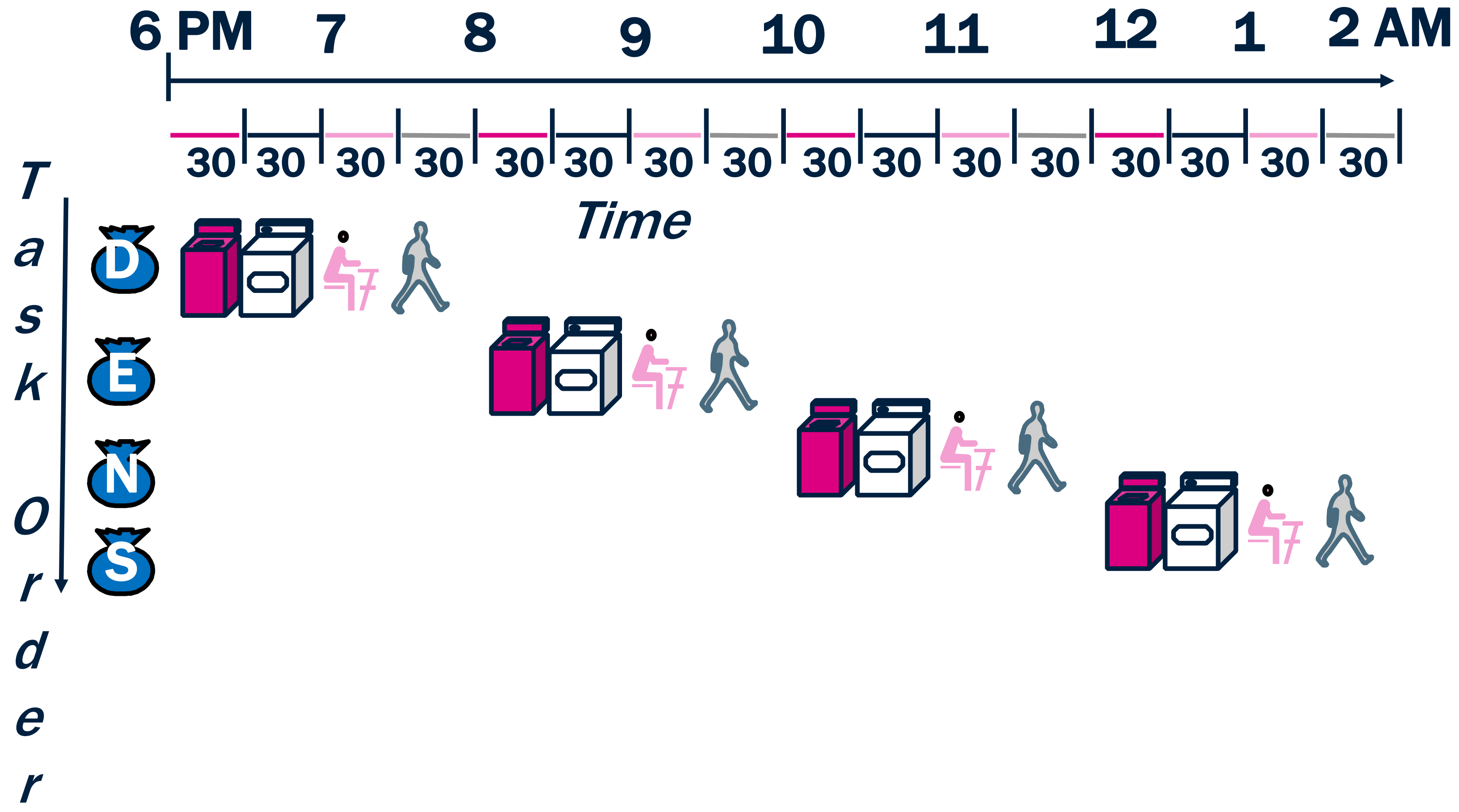






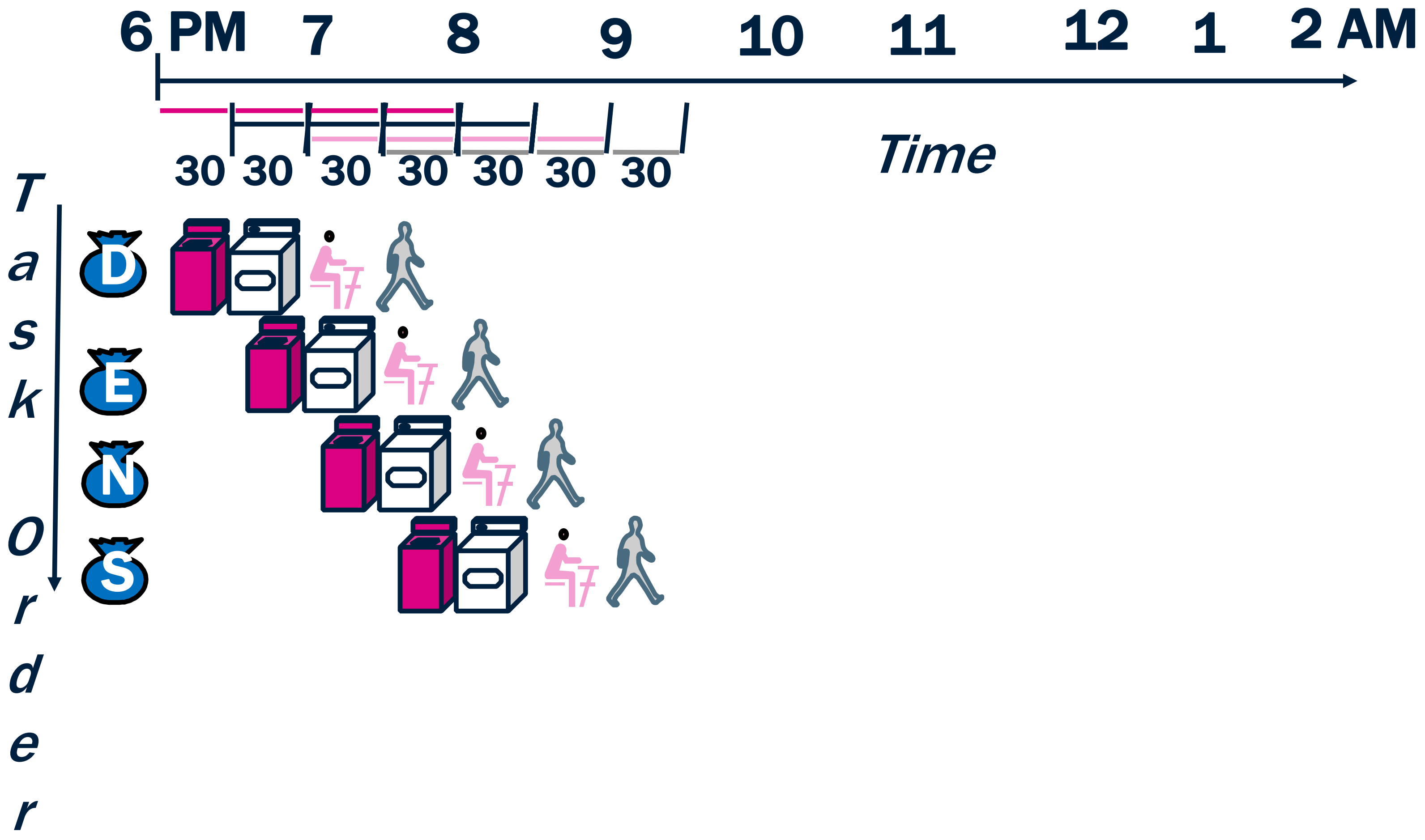


Sequential Laundry



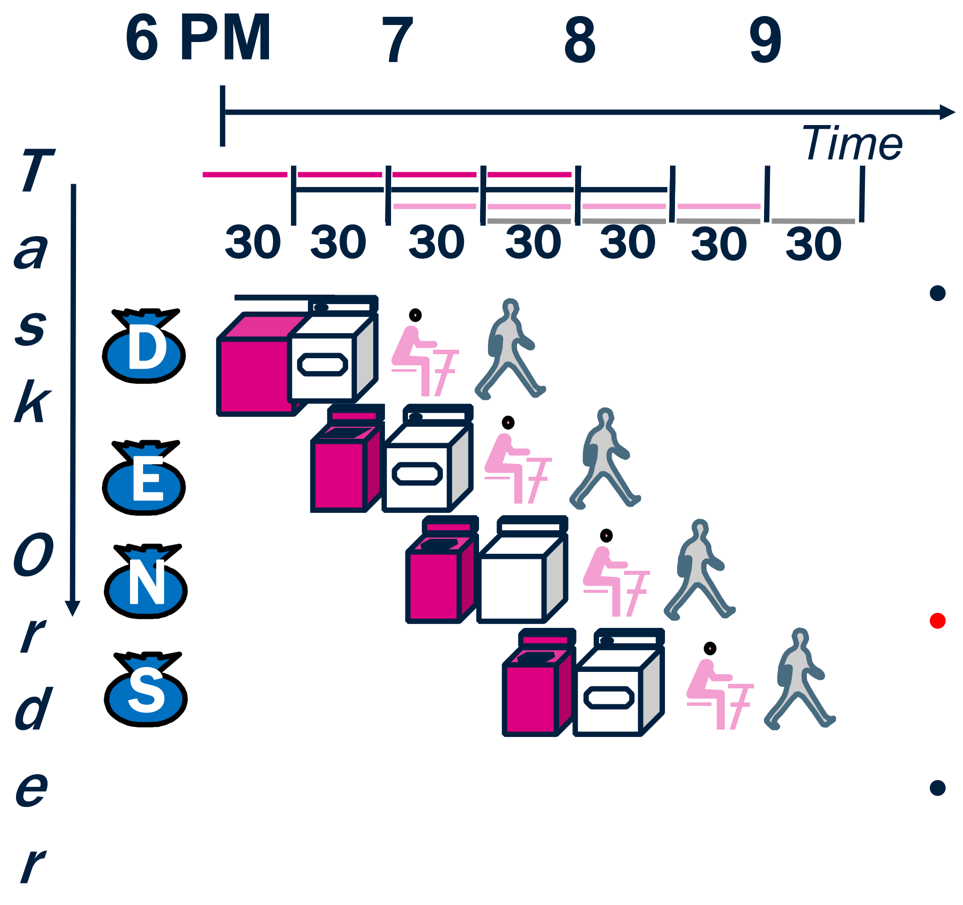
Sequential laundry takes 8 hours for 4 loads

Pipelined Laundry



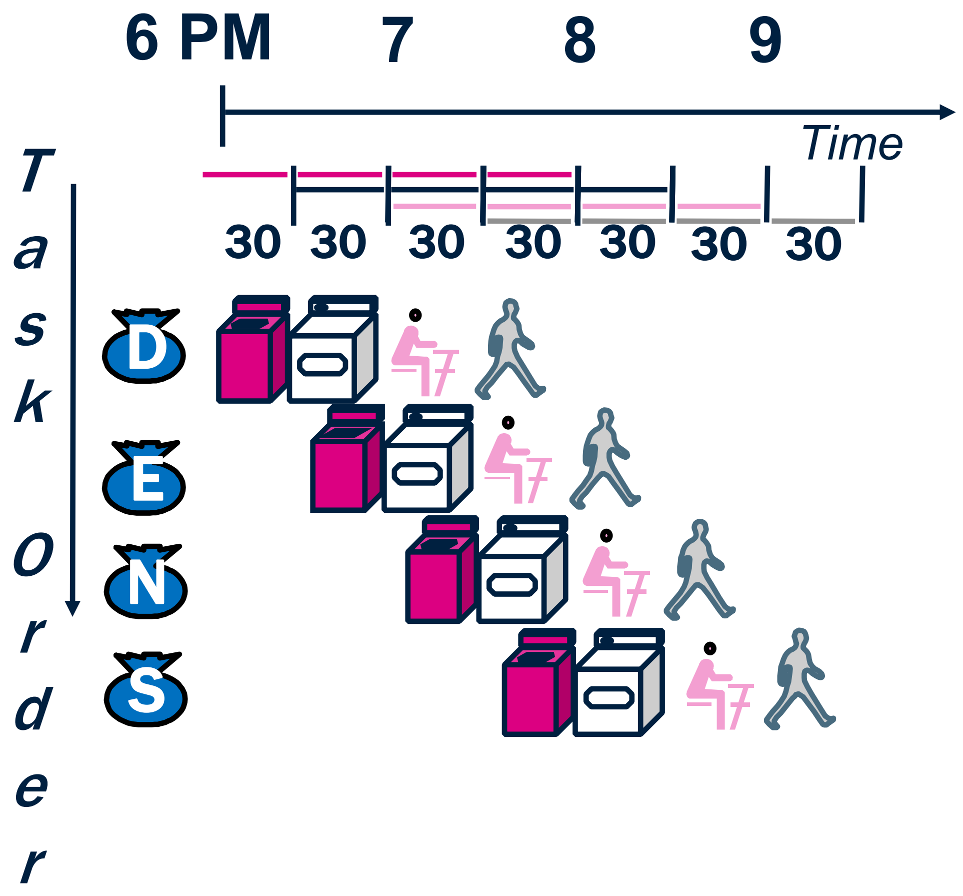
Pipelined laundry takes 3.5 hours for 4 loads!

Pipelining Lessons (1/2)



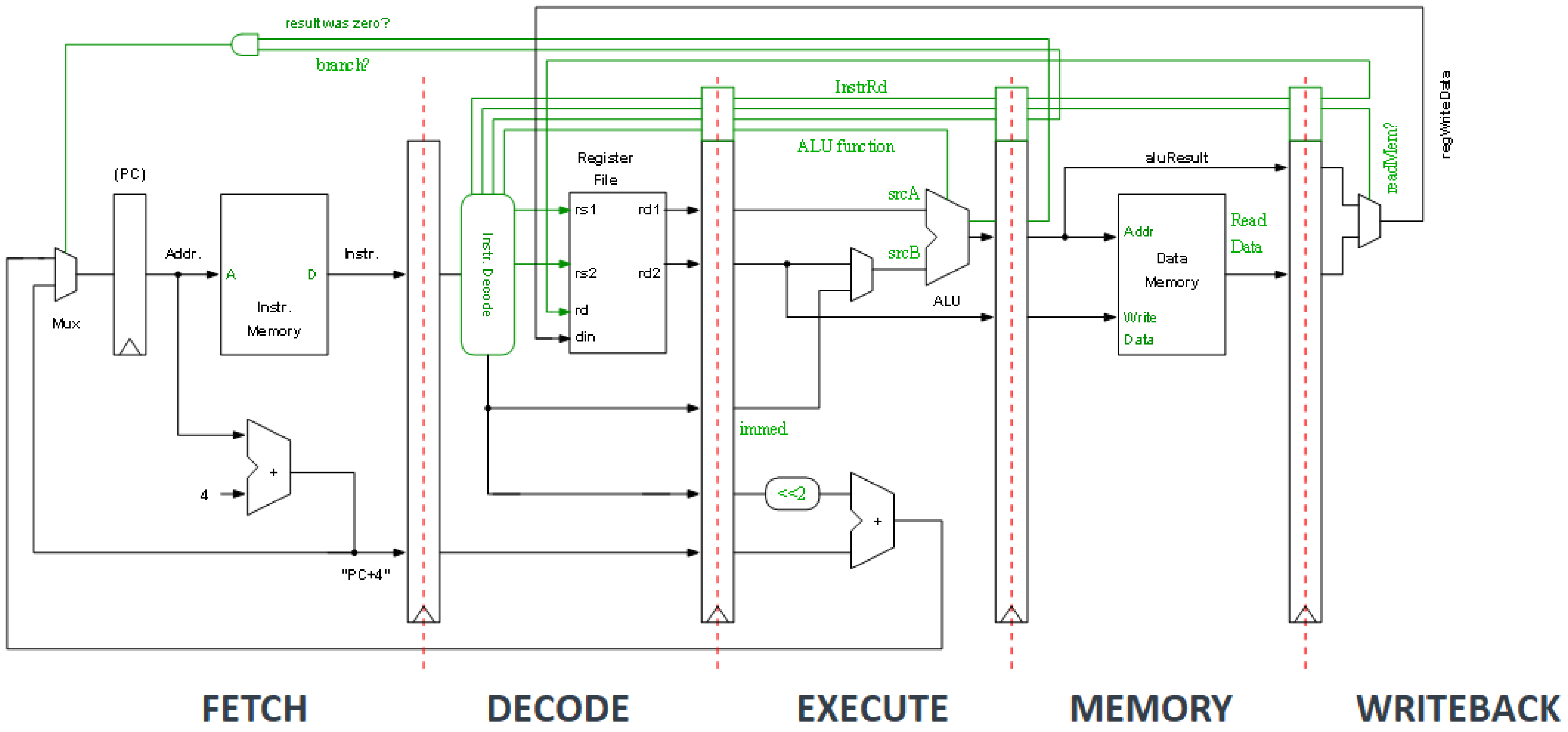
- Pipelining doesn't help *latency* of single task, just *throughput* of entire workload is increased!
- *Multiple* tasks operating simultaneously using different resources
- Potential speedup = number of pipeline stages
- Speedup reduced by time to *fill* and *drain* the pipeline: 8 hours/3.5 hours or 2.3X v. potential 4X in this example

Pipelining Lessons (2/2)

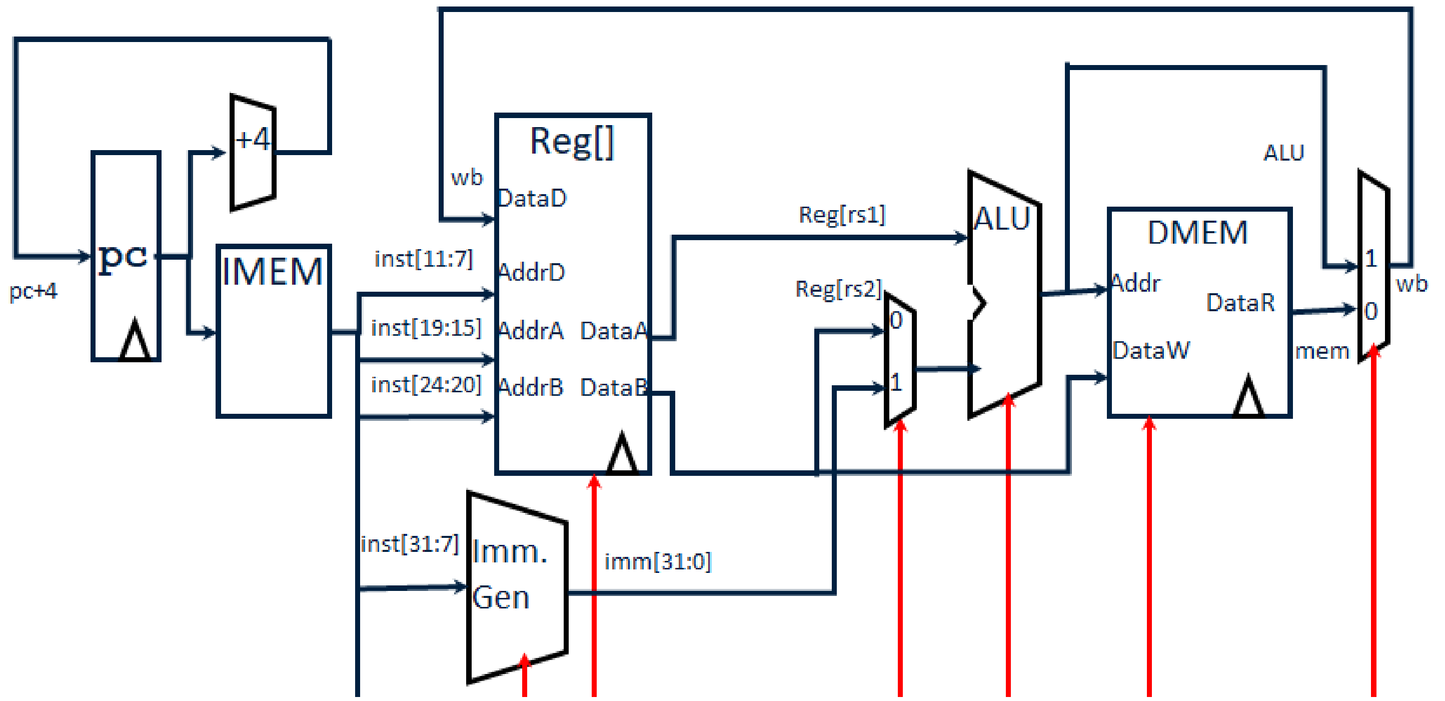


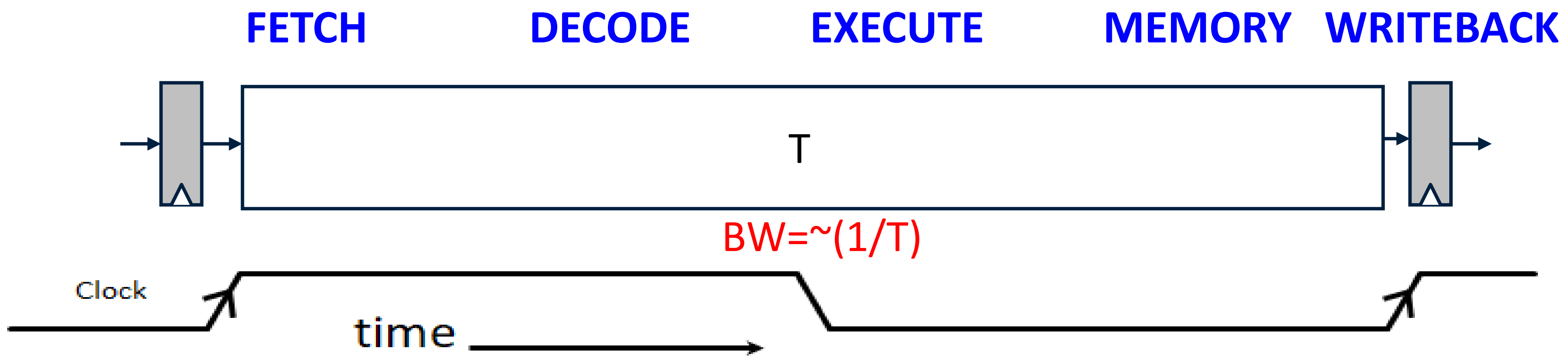
- Suppose new Washer takes 20 minutes, new Stasher takes 20 minutes. How much faster is pipeline?
  - Pipeline rate limited by *slowest* pipeline stage
  - Unbalanced lengths of pipeline stages reduces speedup

Pipelining: Processor Datapath Example



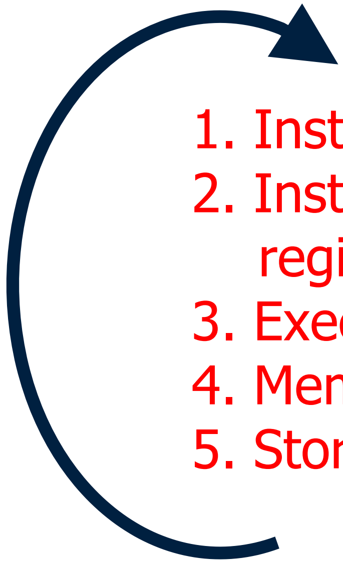
Single-Cycle Datapath





Single-Cycle Datapath

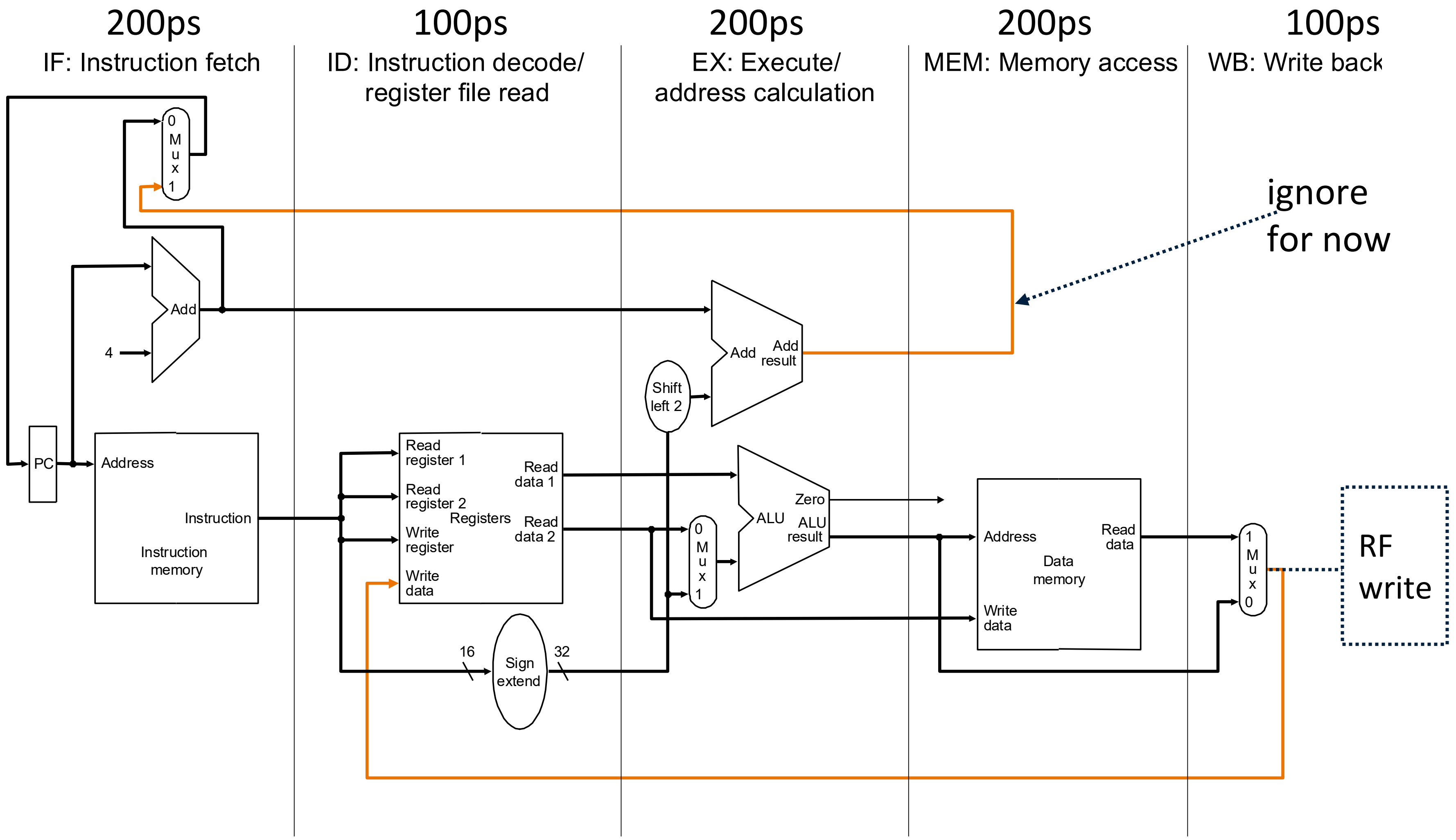
- Our single-cycle processor executes each instruction in one clock cycle, i.e., it has a Clocks Per Instruction (CPI) of 1.



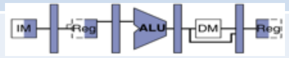
1. Instruction fetch (IF)
2. Instruction decode and register operand fetch (ID/RF)
3. Execute/Evaluate memory address (EX/AG)
4. Memory operand fetch (MEM)
5. Store/writeback result (WB)

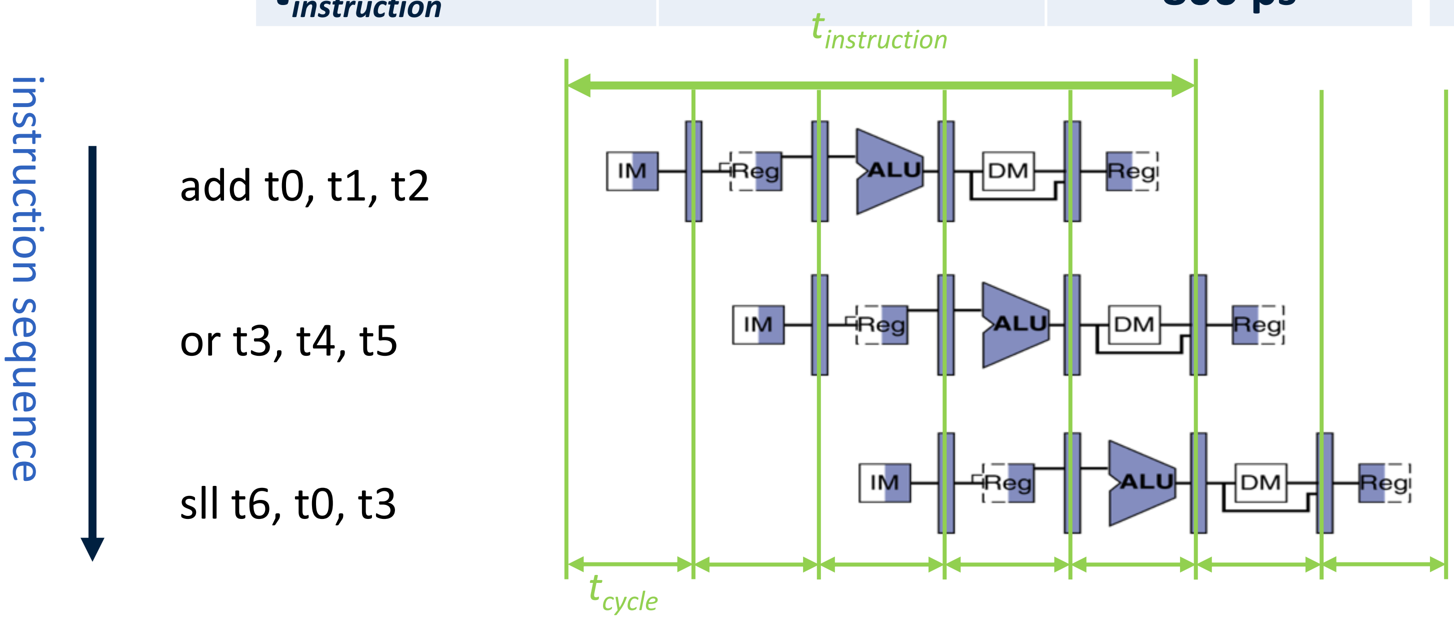
- How might we improve our clock frequency without significantly increasing CPI?
  - We can break the execution of instructions into stages and overlap the execution of different instructions.
  - We need to ensure that the results produced by our new pipelined processor are no different to the unpipelined one.

Pipelined Datapath: Dividing Into Stages



Pipelined Datapath

Phase	Pictogram	t_{step} Serial	t_{cycle} Pipelined
Instruction Fetch		200 ps	200 ps
Reg Read		100 ps	200 ps
ALU		200 ps	200 ps
Memory		200 ps	200 ps
Register Write		100 ps	200 ps
$t_{instruction}$		**800 ps**	**1000 ps**



18

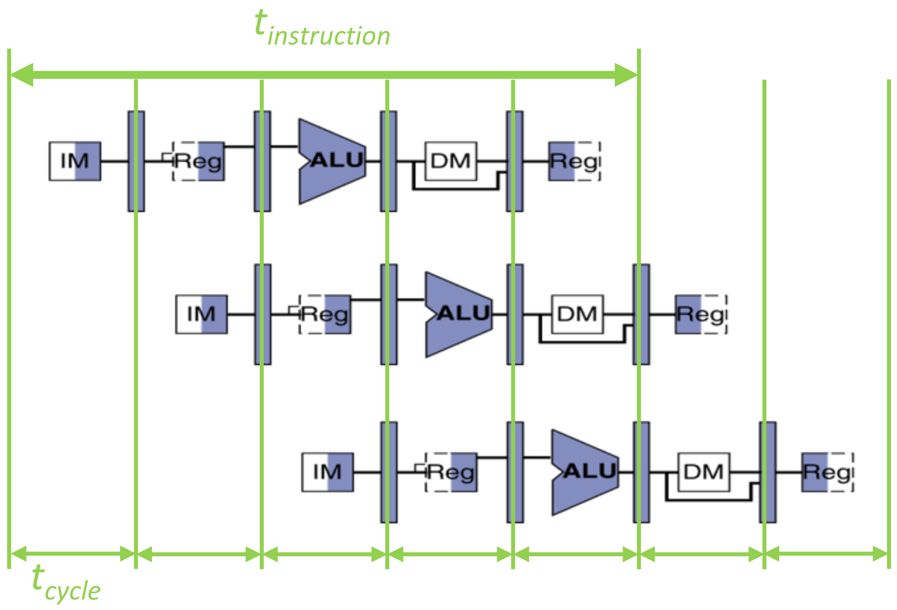
Pipelined Datapath

instruction sequence

add t0, t1, t2

or t3, t4, t5

sll t6, t0, t3



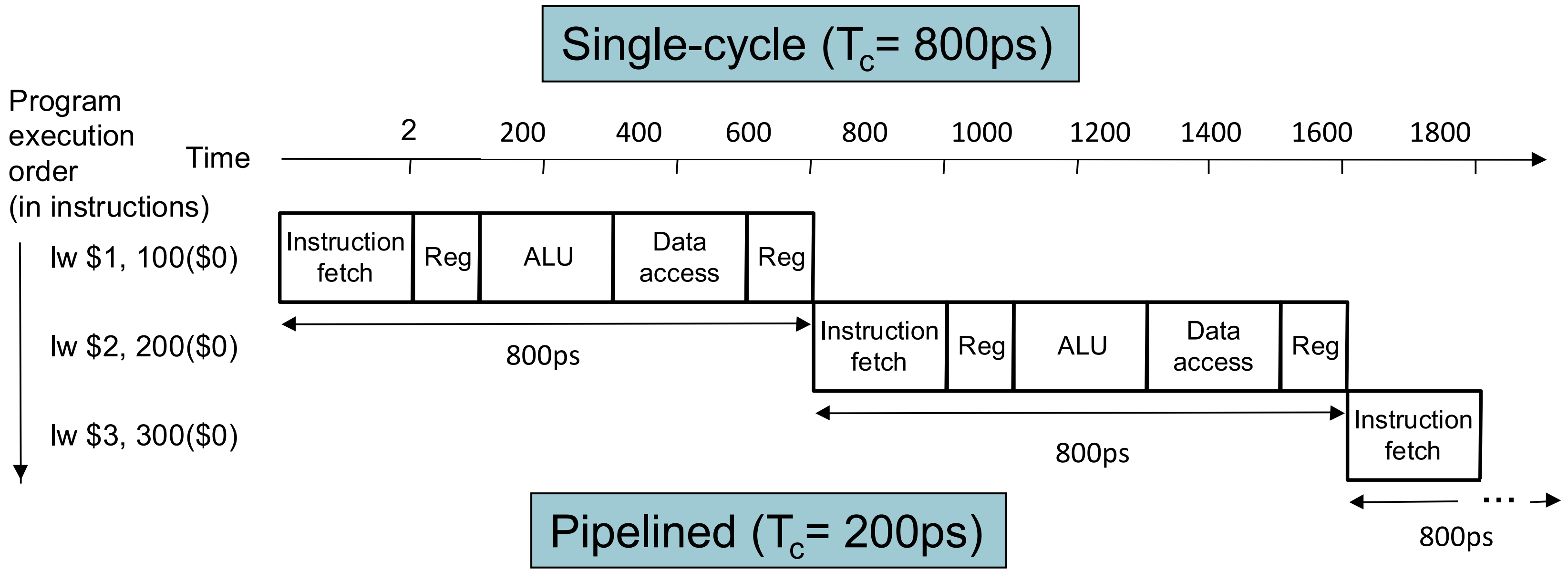
	Single Cycle	Pipelining
Timing	t_{step} = 100 … 200 ps	t_{cycle} = 200 ps
	Register access only 100 ps	All cycles same length
Instruction time, $t_{instruction}$	= t_{cycle} = 800 ps	1000 ps
Clock rate, f_s	1/800 ps = 1.25 GHz	1/200 ps = 5 GHz
Relative speed	1 x	4 x

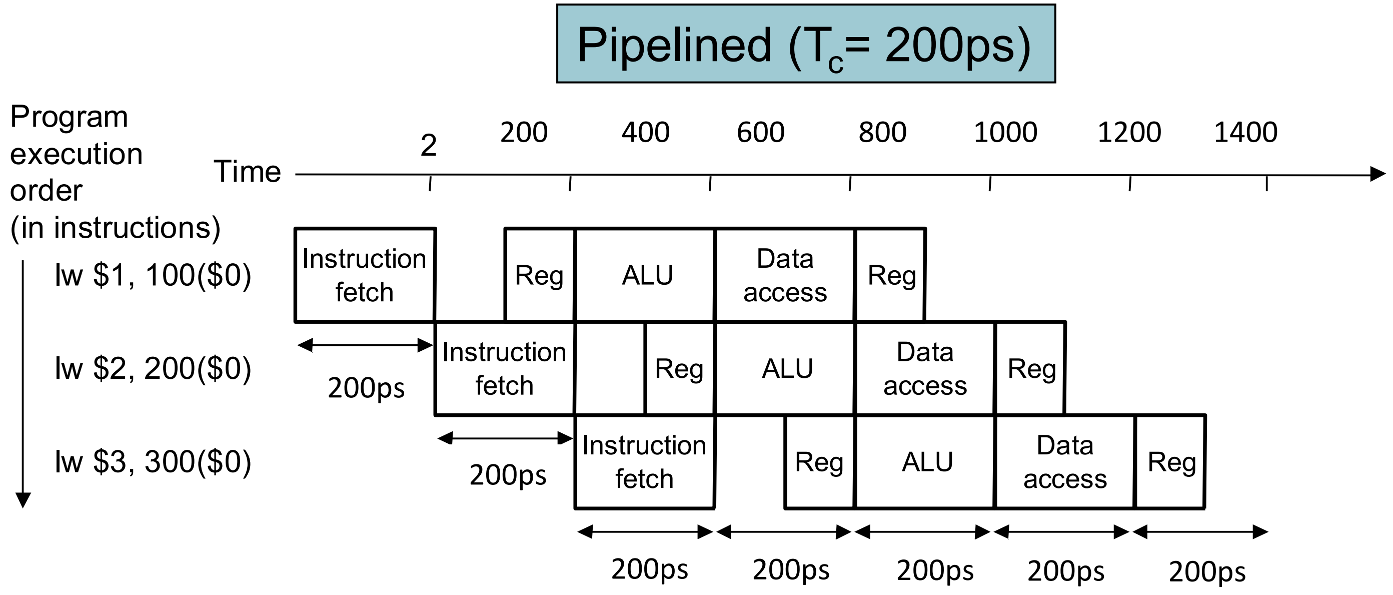
Pipelining Lessons

- Use T_c ("time between completion of instructions") to measure speedup
  - $T_{c,\text{pipelined}} \geq \dfrac{T_{c,\text{single-cycle}}}{\text{Number of stages}}$
  - Equality only achieved if stages are *balanced* (i.e. take the same amount of time)

- If not balanced, speedup is reduced
- Speedup due to increased *throughput*
  - *Latency* for each instruction does not decrease

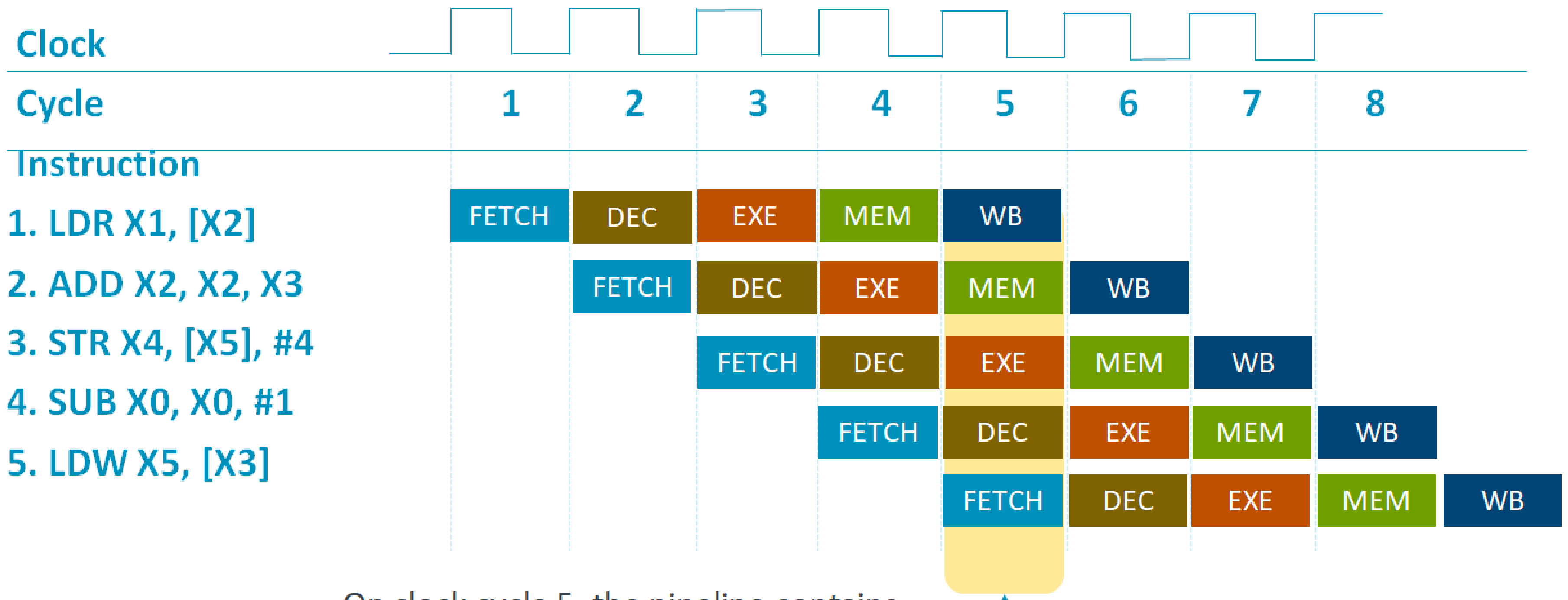
Instruction Pipeline Throughput





5-stage speedup is 4, not 5 as predicted by the ideal model. Why?

The Pipeline in Action



Clock									
Cycle	1	2	3	4	5	6	7	8	
Instruction									
1. LDR X1, [X2]	FETCH	DEC	EXE	MEM	WB				
2. ADD X2, X2, X3		FETCH	DEC	EXE	MEM	WB			
3. STR X4, [X5], #4			FETCH	DEC	EXE	MEM	WB		
4. SUB X0, X0, #1				FETCH	DEC	EXE	MEM	WB	
5. LDW X5, [X3]					FETCH	DEC	EXE	MEM	WB

On clock cycle 5, the pipeline contains all the instructions listed in different pipeline stages

Sequential vs Simultaneous

What happens sequentially, what happens simultaneously?

instruction sequence

add t0, t1, t2

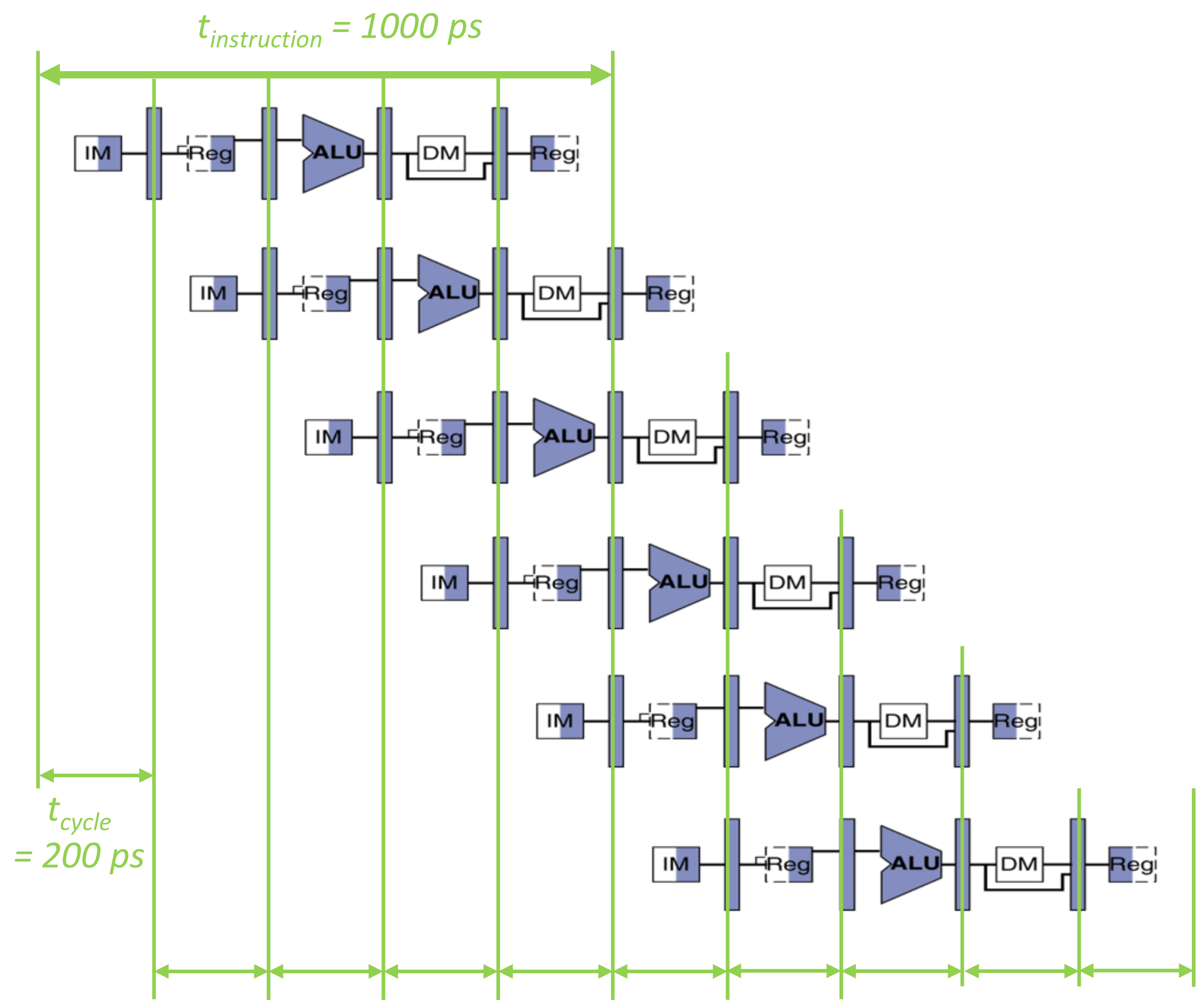
or t3, t4, t5

sll t6, t0, t3

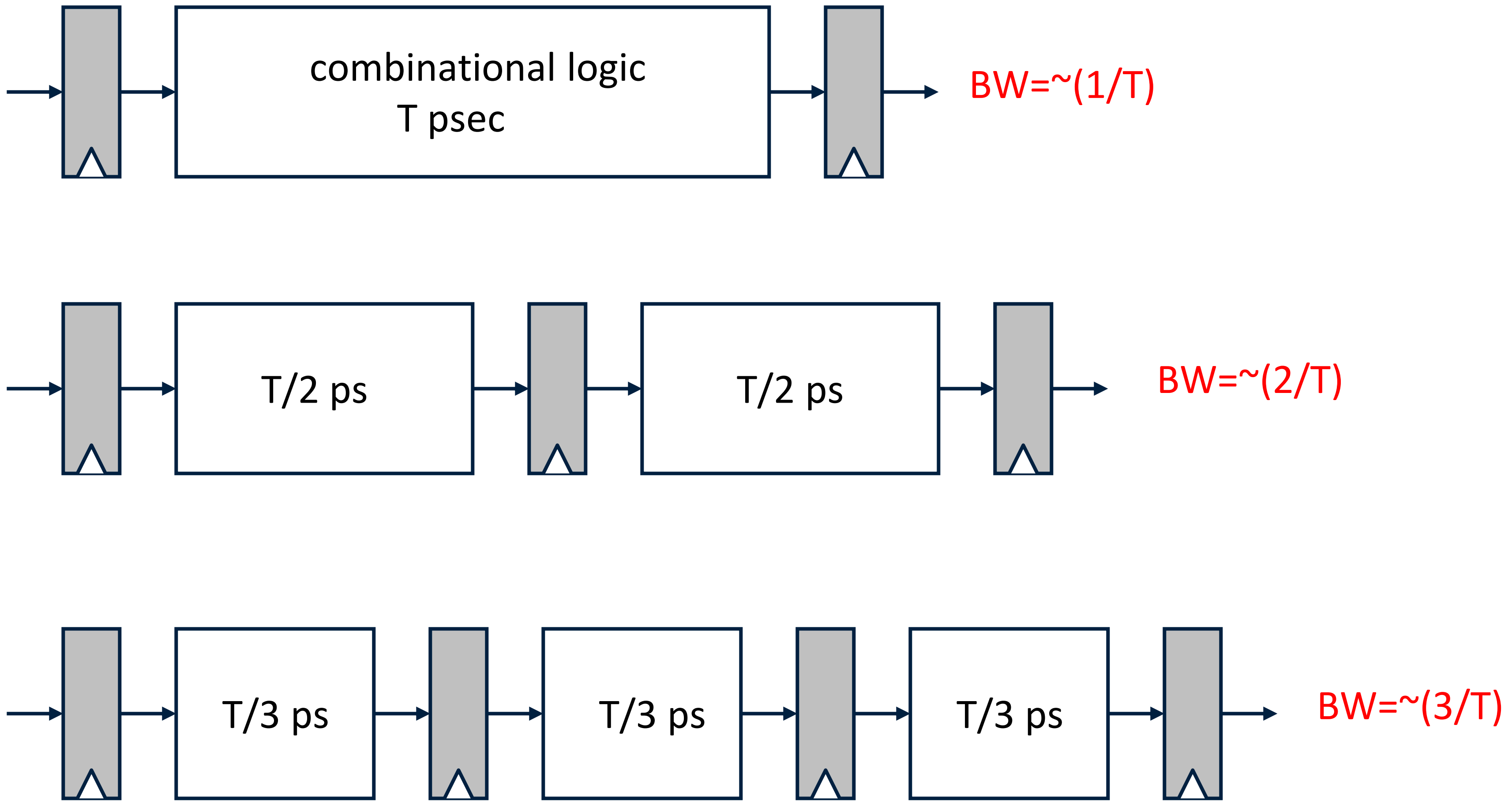
sw t0, 4(t3)

lw t0, 8(t3)

addi t2, t2, 1



Implementing Pipelining



Slide Credit: Prof. Onur Mutulu

Summary

- Pipelining improves performance by *increasing instruction throughput, in contrast to decreasing the execution time of an individual instruction*, but instruction throughput is the important metric because real programs execute billions of instructions.

Recommended Reading

- Computer Organization and Design_ The Hardware Software Interface - David A. Patterson, John L. Hennessy:
  - Chapter-4:
    - 4.5
    - 4.6 -> Graphically representing pipeline

THANK YOU