Computer Systems II

Li Lu
Room 605, CaoGuangbiao Building
li.lu@zju.edu.cn
https://person.zju.edu.cn/lynnluli

Single-Cycle vs. Pipelined Performance

- Example: consider a system with seven instructions
  - load word (lw)
  - store word (sw)
  - add (add)
  - subtract (sub)
  - AND (and)
  - OR (or)
  - branch if equal(beq)

Total Time for Each Instruction

Instruction Class	Instruction Fetch	Register Read	ALU Operation	Data Access	Register Write	Total Time
lw	200ps	100ps	200ps	200ps	100ps	800ps
sw	200ps	100ps	200ps	200ps		700ps
add, sub, and, or	200ps	100ps	200ps		100ps	600ps
beq	200ps	100	200ps			500ps

Question: Consider the Clock Cycle for the Single-Cycle and the Pipeline

Total Time for Each Instruction

Instruction Class	Instruction Fetch	Register Read	ALU Operation	Data Access	Register Write	Total Time
lw	200ps	100ps	200ps	200ps	100ps	800ps
sw	200ps	100ps	200ps	200ps		700ps
add, sub, and, or	200ps	100ps	200ps		100ps	600ps
beq	200ps	100	200ps			500ps

Clock Cycle for Single-Cycle = MAX(instructions total times) = 800ps

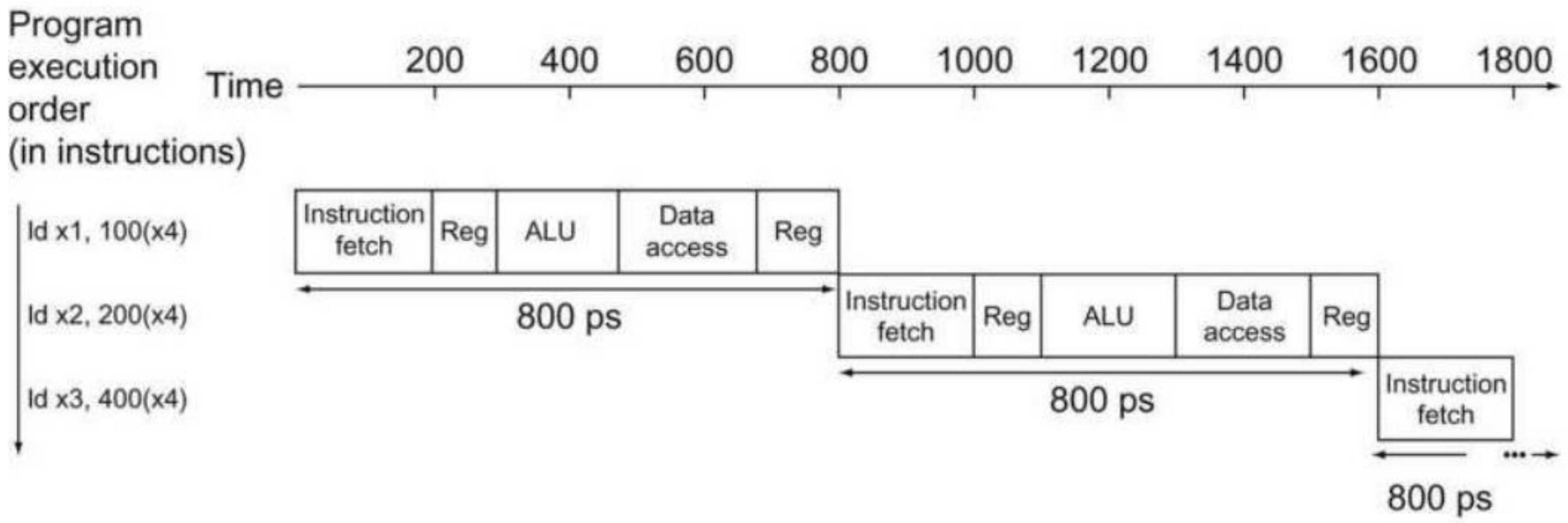
Total Time for Each Instruction

Instruction Class	Instruction Fetch	Register Read	ALU Operation	Data Access	Register Write	Total Time
lw	200ps	100ps	200ps	200ps	100ps	800ps
sw	200ps	100ps	200ps	200ps		700ps
add, sub, and, or	200ps	100ps	200ps		100ps	600ps
beq	200ps	100	200ps			500ps

Clock Cycle for Single-Cycle = MAX(instructions total times) = 800ps

Clock Cycle for Pipelined = MAX(operations times) = 200ps

Single-Cycle, Nonpipelined Execution

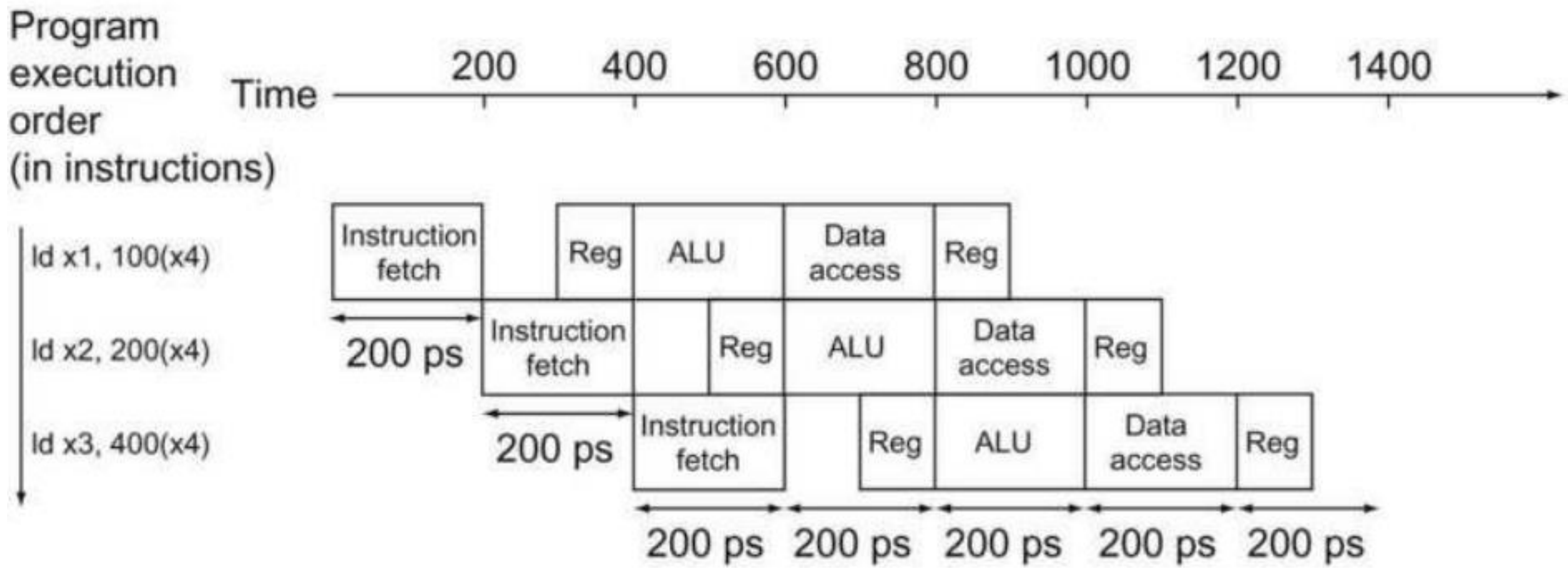


Total execution time for the three instructions = 800ps * 3 = 2400ps

How about Pipeline?

Instruction Class	Instruction Fetch	Register Read	ALU Operation	Data Access	Register Write	Total Time
lw	200ps	100ps	200ps	200ps	100ps	800ps
sw	200ps	100ps	200ps	200ps		700ps
add, sub, and, or	200ps	100ps	200ps		100ps	600ps
beq	200ps	100	200ps			500ps

Pipelined Execution



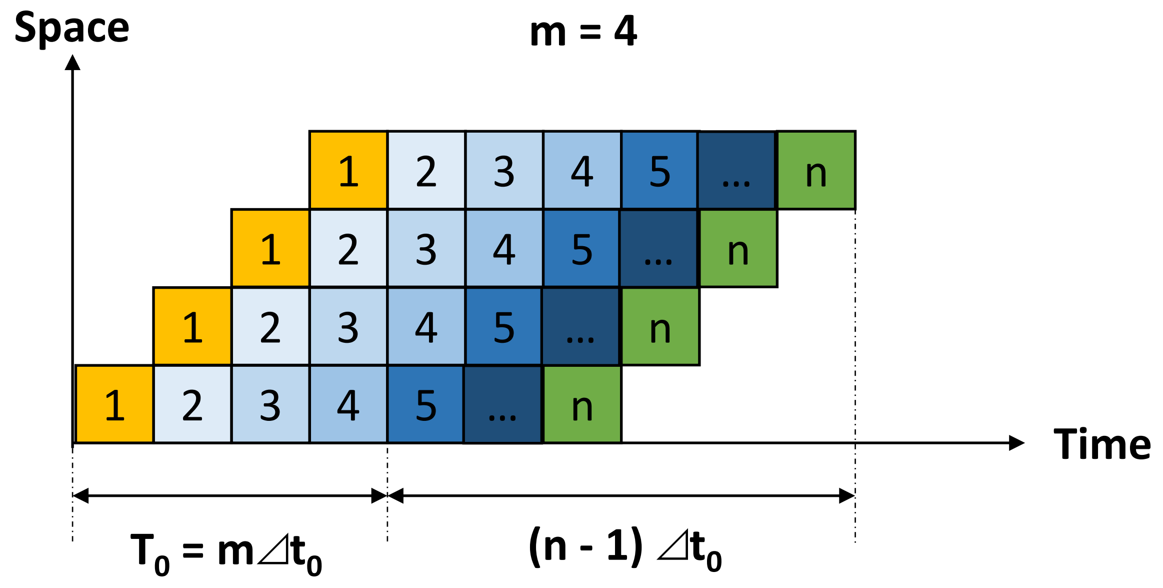
Total execution time for the three instructions = 200ps * 7 = 1400ps

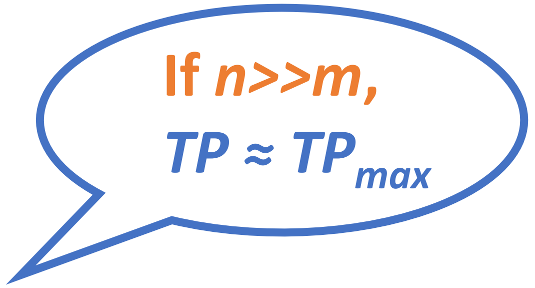
Throughput (TP)

$$TP = \frac{n}{T}$$

$$TP < TP_{max}$$

§2.4 Performance evaluation of pipelining





$$T = (m + n - 1) \times \triangle t_0$$

$$TP = n / (m + n - 1) \triangle t_0$$

$$TP_{max} = 1 / \triangle t_0$$

Throughput (TP)

$$\boldsymbol{TP = \frac{n}{n + m - 1} TP_{max}}$$

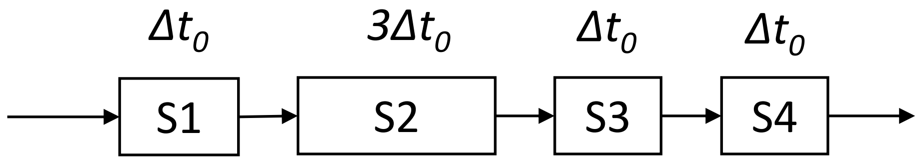
- The actual throughput of the pipeline is **less than** the maximum throughput, which is not only related to the time of each segment, but also related to *m* and *n*.

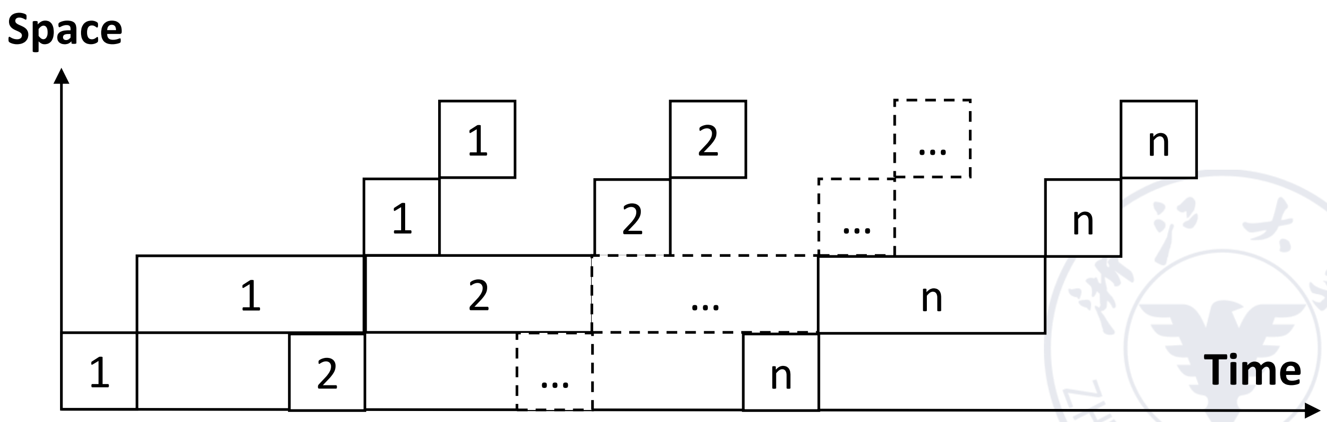
- If $n >> m$, $TP \approx TP_{max}$

Throughput under Practical Case

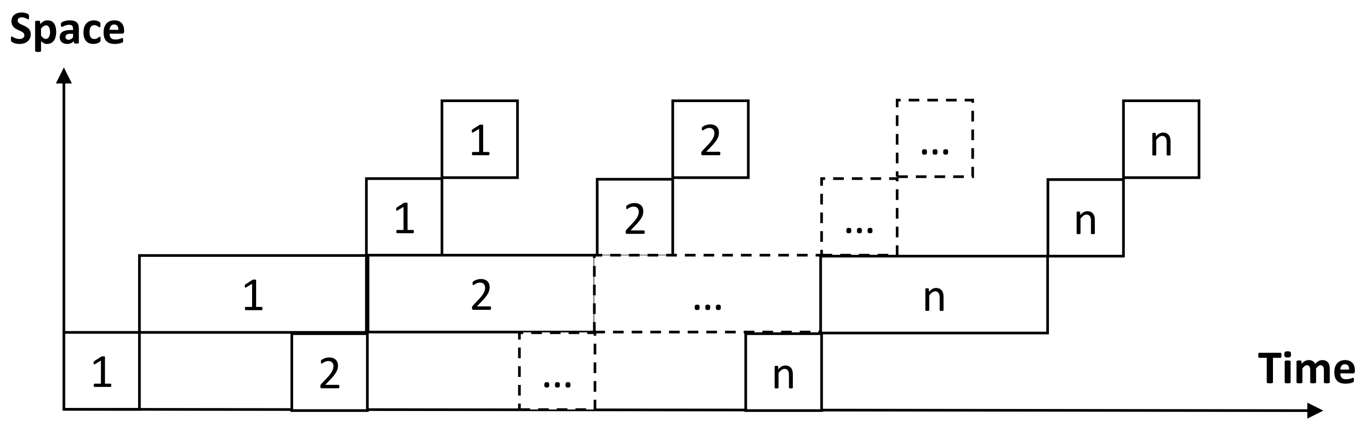
- Suppose the time of segments are different in pipelining,
  - *m = 4*
  - Time of *S1, S3, S4: Δt*
  - Time of *S2: 3Δt* (**Bottleneck**)

The longest segment in the pipelining is called the bottleneck segment.





Throughput under Practical Case



$$TP = \frac{n}{T}$$

$$\boldsymbol{TP} = \frac{\boldsymbol{n}}{\sum_{i=1}^{m} \Delta t_i + (n - 1) max(\Delta t_1, \Delta t_2, \ \dots, \Delta t_m)}$$

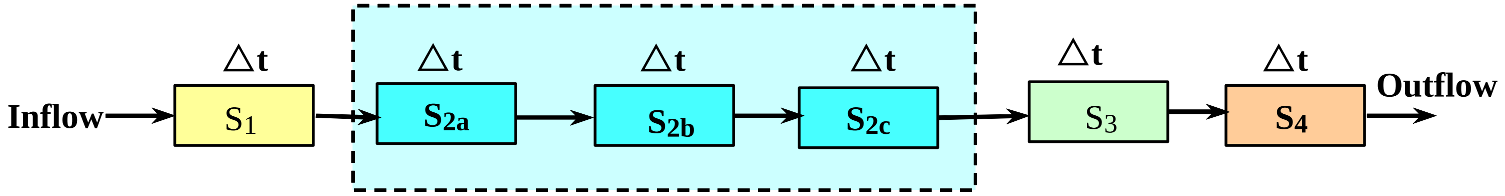
$$= \frac{\boldsymbol{n}}{(m-1)\Delta t + n\, max(\Delta t_1, \Delta t_2, \ \dots, \Delta t_m)}$$

$$\boldsymbol{TP_{max}} = \frac{\boldsymbol{1}}{max(\Delta t_1, \Delta t_2, \ \dots, \Delta t_m)}$$

$$\boldsymbol{TP_{max}} = \frac{\boldsymbol{1}}{3\Delta t}$$

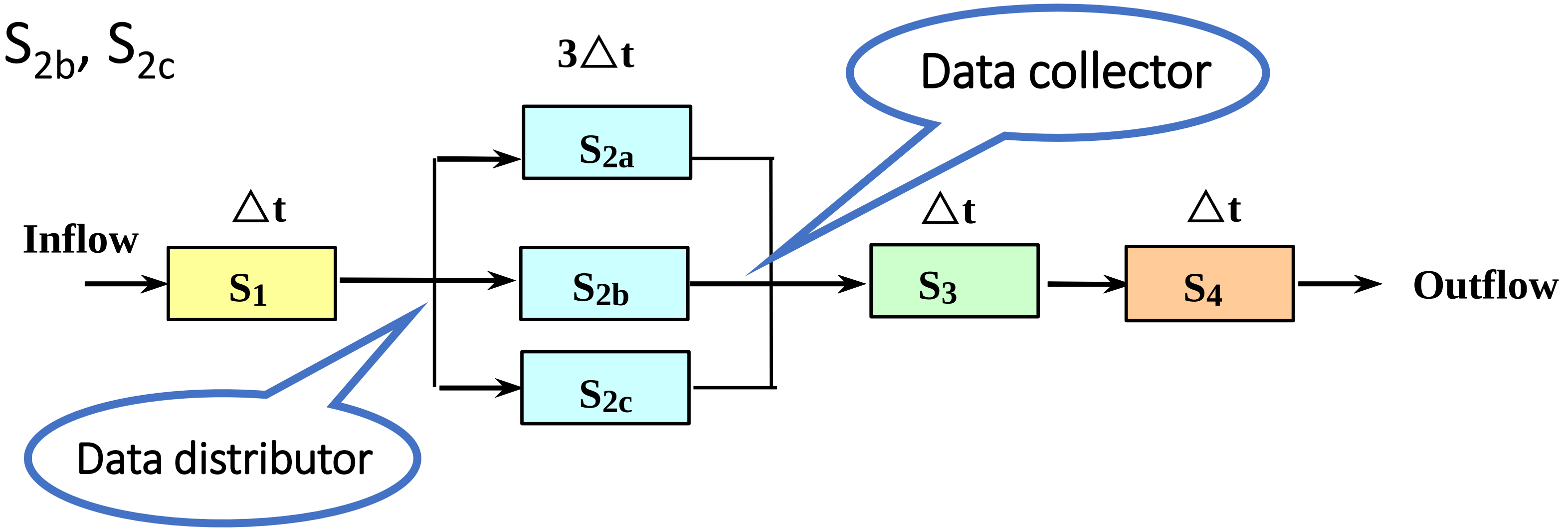
Common methods to solve pipeline bottleneck

- Subdivision
  - Divide S_2 into 3 subsegments： S_{2a}, S_{2b}, S_{2c}

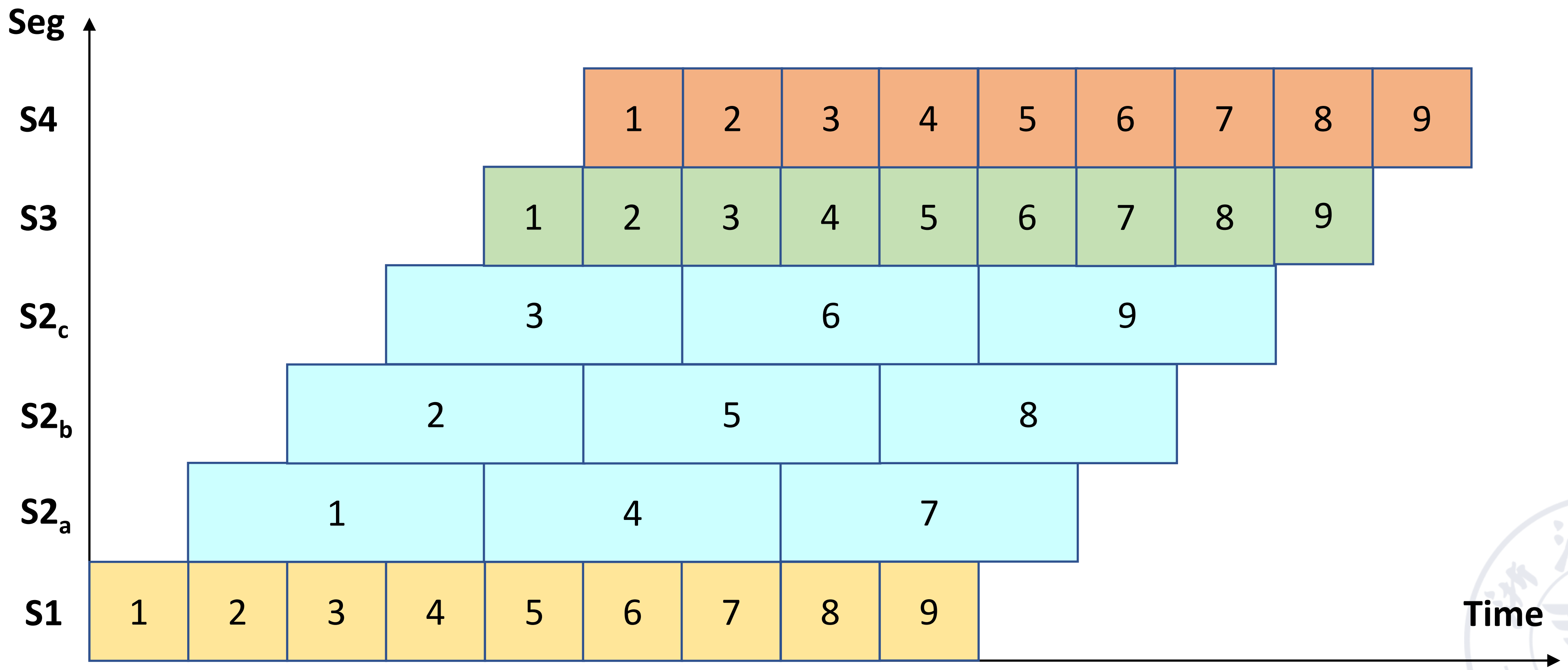


$$TP_{max} = \frac{1}{\Delta t}$$

- Repetition
  - S_2: S_{2a}, S_{2b}, S_{2c}



Time space diagram with repetition bottleneck segment



Speedup (Sp)

In the example (execute 3 instructions):

- Non-pipelined execution time **vs.** Pipelined execution time

  2,400ps **vs.** 1,400ps

What would happen if we increased the number of instructions?

Extend the pervious example to 1,000,003 instructions

- Non-pipelined execution time **vs.** Pipelined execution time

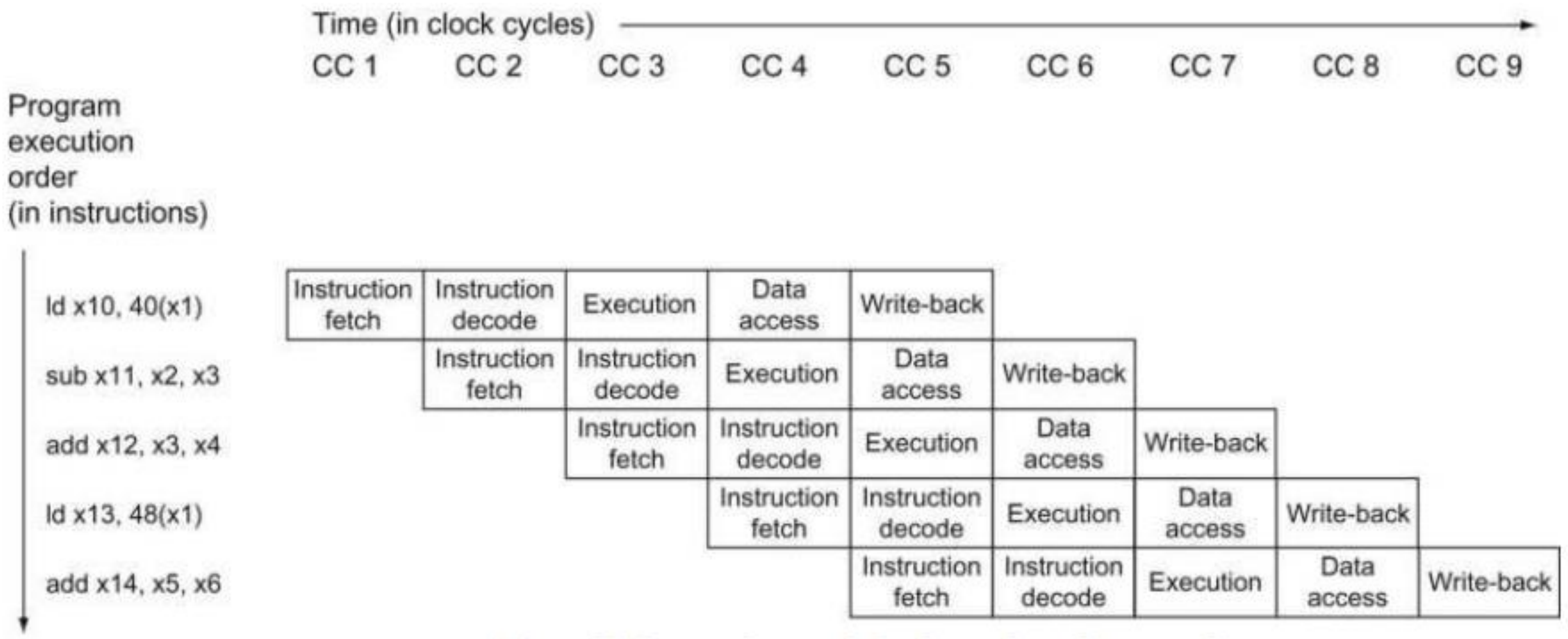
  1,000,000 ×800ps +2,400ps **vs.** 1,000,000 ×200ps +1,400ps

  800,002,400ps **vs.** 200,001,400ps

4:1

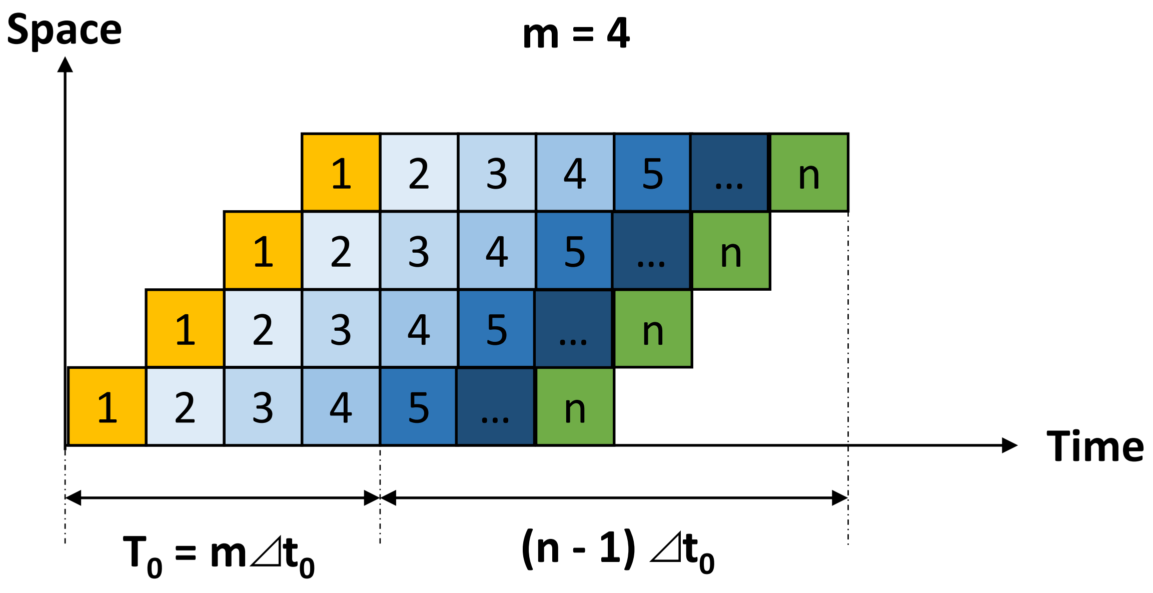
Speedup (Sp)

$$Execution\ Time_{\boldsymbol{pipelined}} = \frac{Execution\ Time_{\boldsymbol{nonpipelined}}}{Number\ of\ Pipestages}$$



The five-stage pipeline is nearly five times faster

Speedup (Sp)

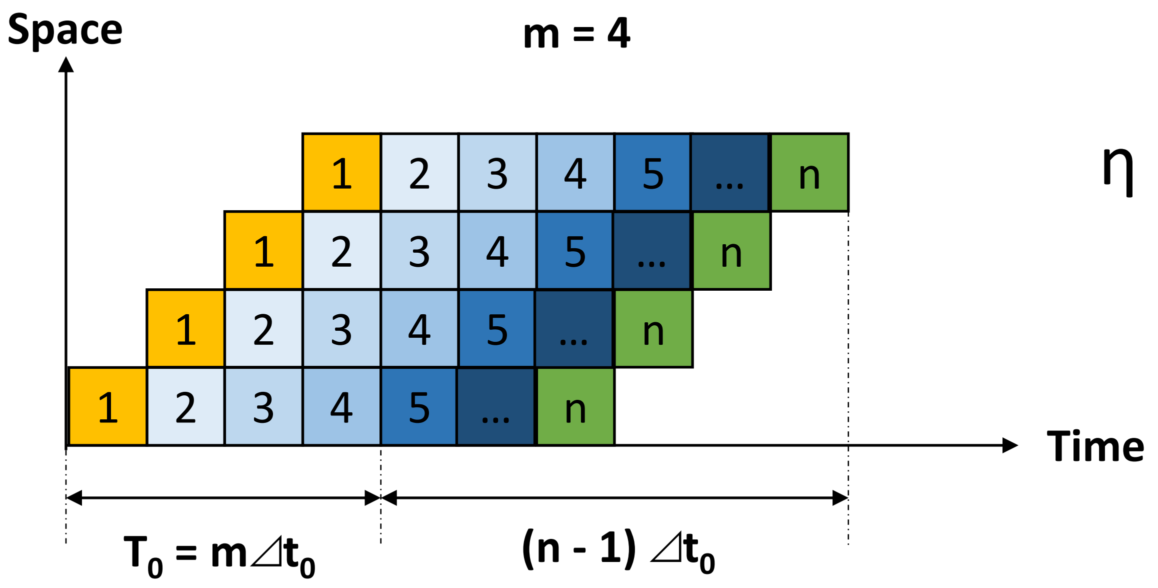


$T = (m + n - 1) \times \Delta t_0$

$$Sp = (n \times m \times \Delta t_0) / (m + n - 1) \Delta t_0$$
$$= (n \times m) / (m + n - 1)$$

If n>>m,
$Sp \approx m$

Efficiency (η)



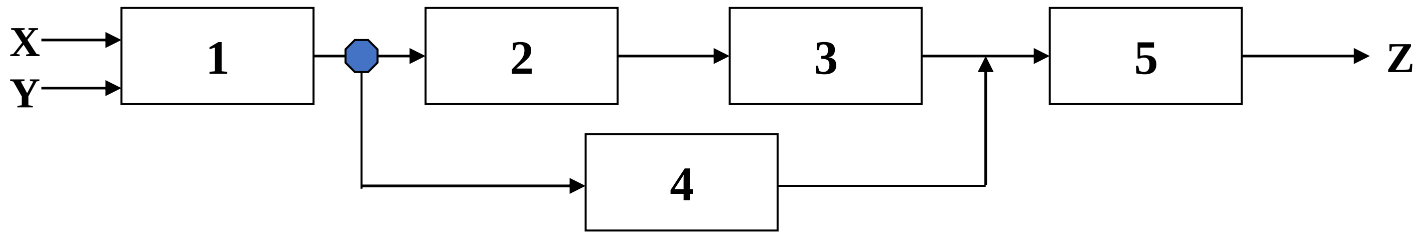
$$T = (m + n - 1) \times \Delta t_0$$

$$\eta = (n \times m \times \Delta t_0) / (m + n - 1) \Delta t_0 \times m$$
$$= n / (m + n - 1)$$

If n>>m,
$\eta \approx 1$

Pipeline Performance

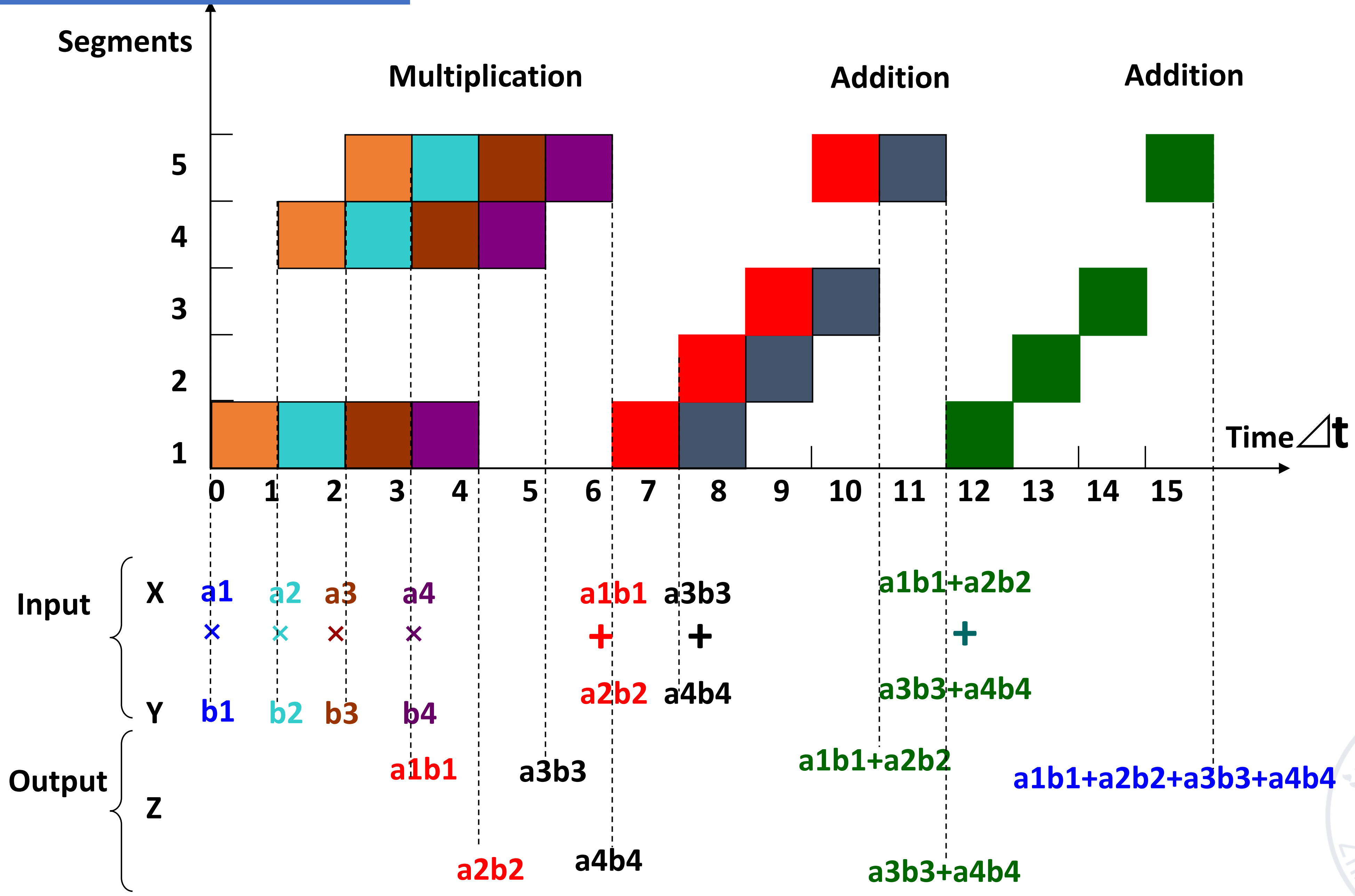
- Vector A(a1, a2,a3,a4)
- Vector B(b1,b2,b3,b4)
- Compute vector dot product (A·B) in the static dual-function pipelining.



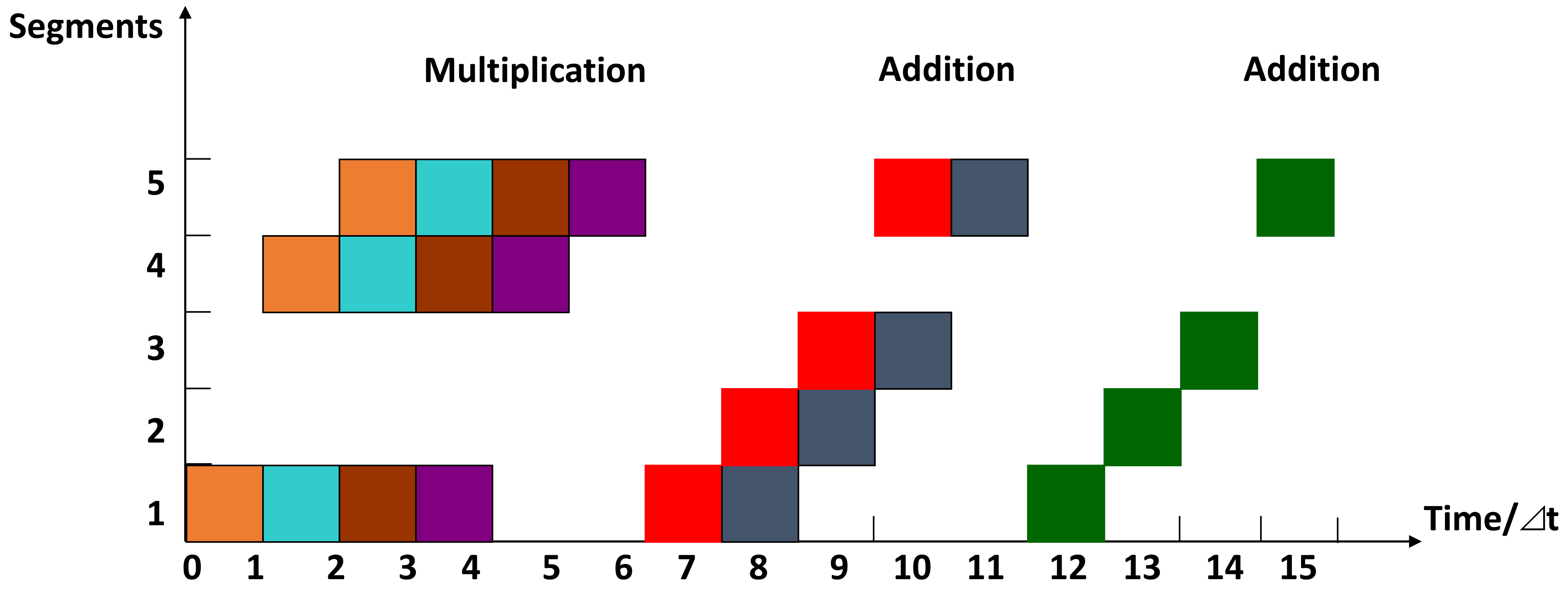
- 1→2→3→5 Addition pipelining
- 1→4→5 Multiplication pipelining
- The time of each segment in the pipelining is ⊿t。

TP? Sp? η?

§2.4 Performance evaluation of pipelining



§2.4 Performance evaluation of pipelining



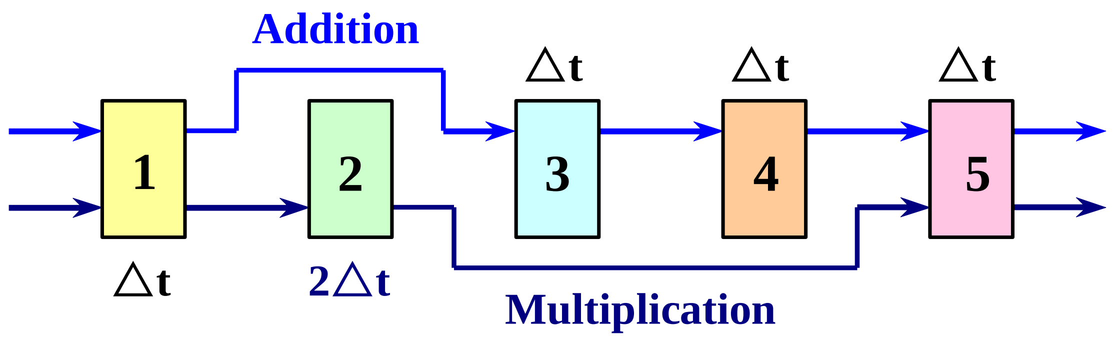
TP=7/(15⊿t)=0.47/⊿t

Sp=(4×3⊿t+3×4⊿t)/(15⊿t)=1.6

η=(3×4⊿t+4×3⊿t)/(5×15⊿t)=32%

Pipeline Performance

- Vector A(a1, a2,a3,a4)
- Vector B(b1,b2,b3,b4)
- Compute vector dot product (A·B) in the dynamic dual-function pipeline-ing

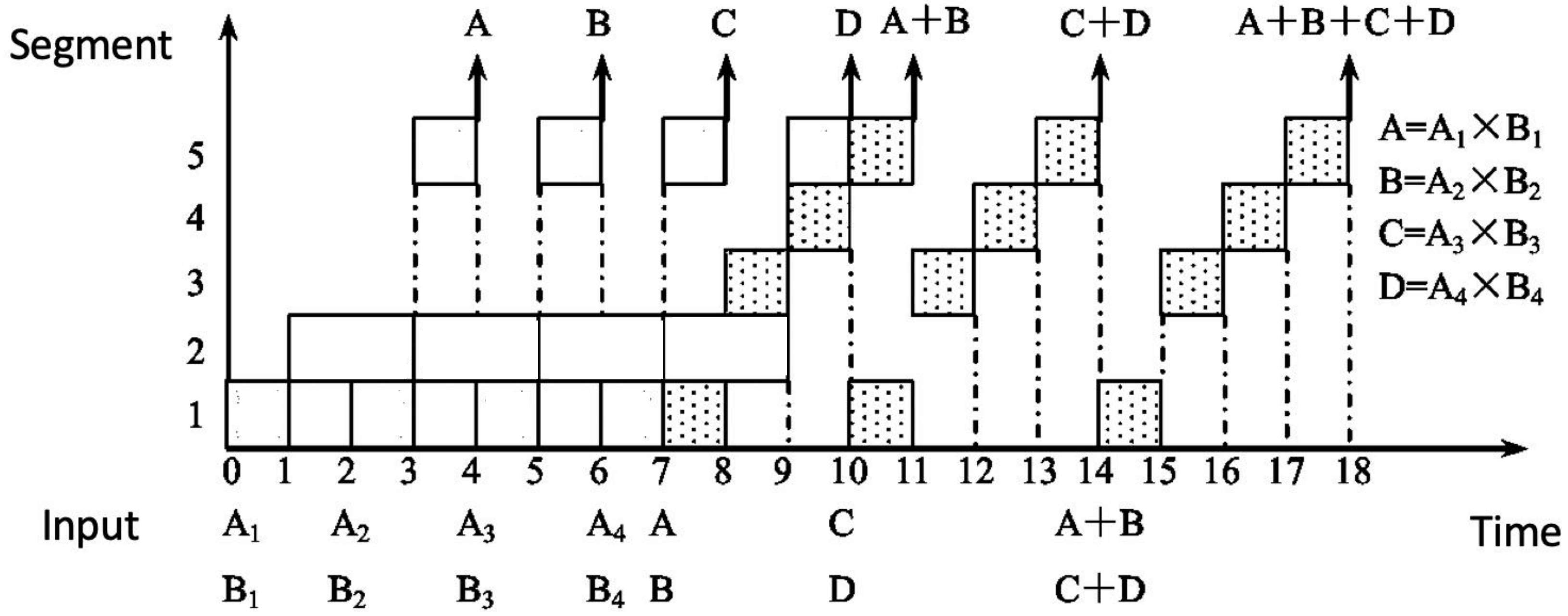


1→3→4→5 Addition pipelining

1→2→5 Multiplication pipelining

TP? Sp? η?

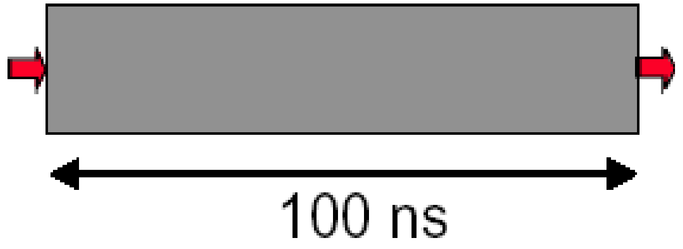
Pipeline Performance

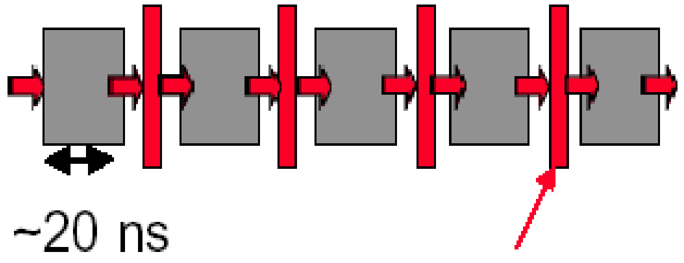


$$TP = \frac{7}{18\Delta t} \qquad S = \frac{28\Delta t}{18\Delta t} \approx 1.56 \qquad E = \frac{4\times4+3\times4}{5\times18} \approx 0.31$$

Discussion

- Why pipelining: overlapped





- Can “launch” a new computation every 100ns in this structure
- Can finish 10^7 computations per second

- Can launch a new computation every 20ns in pipelined structure
- Can finish 5×10^7 computations per second

Discussion

- Why pipelining?
  - The key implementation technique used to make fast CPU: decrease CPU time
  - Improving of Throughput (rather than individual execution time)
  - Improving of efficiency for resources (functional unit)

Discussion

- Ideal Performance for Pipelining

- If the stages are perfectly balanced, the time per instruction on the pipelined processor equal to:

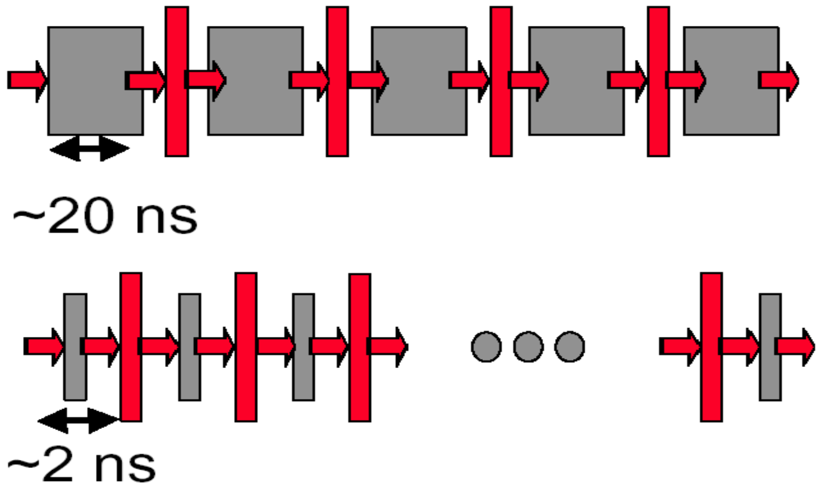
$$\frac{\textit{Time per instruction on unpipelined machine}}{\textit{Number of pipelined stages}}$$

- So, ideal speedup equal to

  Number of pipeline stages

Discussion

- Why not just make a 50-stage pipelining ?



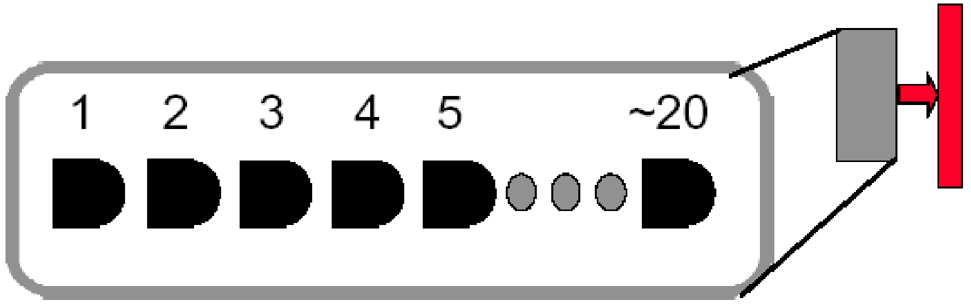
5 stages OK

50 stages NO. Sorry!

- Too many stages:
  - Lots of complications
  - Should take care of possible dependencies among in-flight instructions
  - Control logic is huge

Discussion

- Why not just make a 50-stage pipelining ?
- Those latches are NOT free, they take up area, and there is a real delay to go THRU the latch itself
  - Machine cycle > latch latency + clock skew
- In modern, deep pipeline (10-20 stages), this is a real effect
- Typically see logic “depths” in one pipe stage of 10-20 “gates”



At these speeds, and with this few levels of logic, latch delay is important

Discussion

- What factors affect the efficiency of multi-functional pipeline?

1. When the multi-functional pipeline implements a certain function, there are always some segments for other functions in the idle state
2. In the process of pipelining establishment, some segments to be used are also idle
3. When the segments are not equal, the clock cycle depends on the time of the bottleneck segment
4. When the functions are switch, the pipelining needs to be emptied
5. The output of last operation is the input of the next operation
6. Extra cost: pipelining register delay & overhead of clock skew