Chapter 2

Pipelining

Pipelining?

Pipelining?

You already knew!

Pipelining

- What is pipelining?
- How is the pipelining Implemented?
- What makes pipelining hard to implement?


hello!
HAPPY
cake

Ben

Ben

Peter

Jim

Tom

Ben

Peter

Jim

Tom

Ben

Peter

Jim

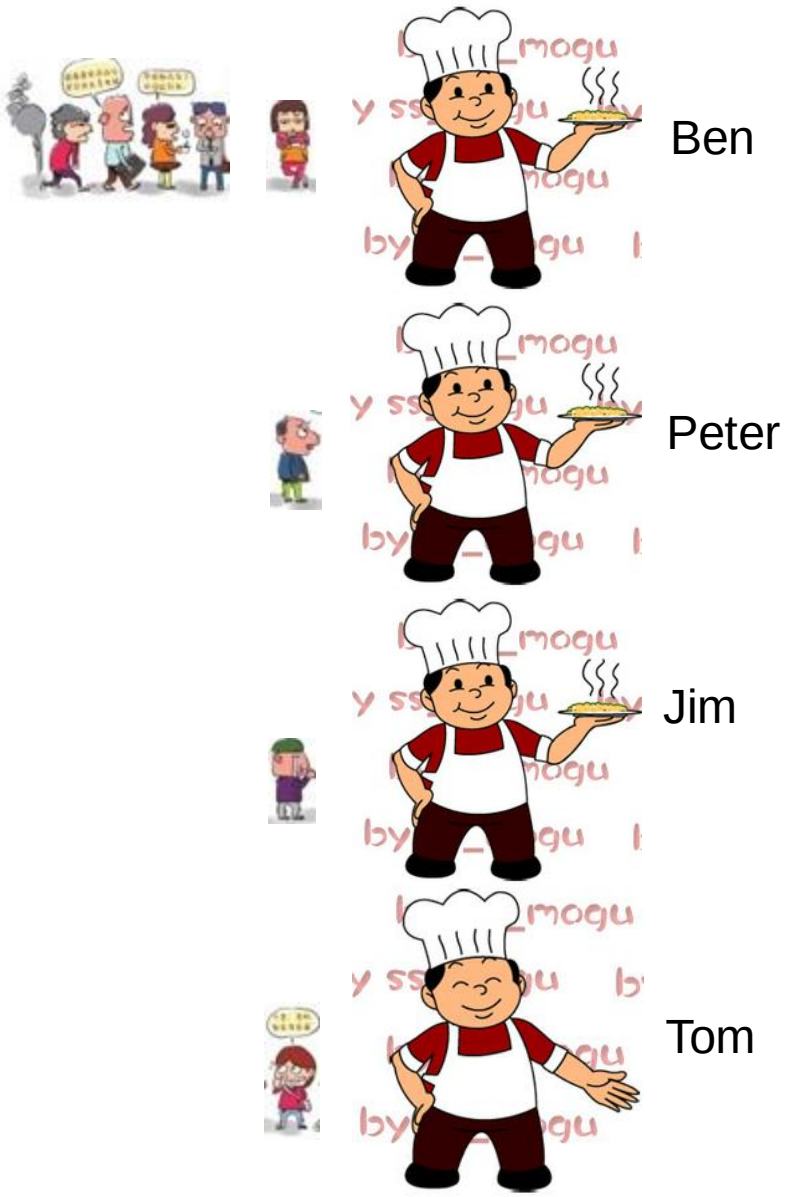
Tom

Ben

Peter

Jim

Tom


Ben
Peter
Jim
Tom

What exactly is pipelining in computer arch?

Speed up the execution of instructions (machine language)

- Shorten the execution time of each instruction
  - More high-speed devices
  - Better calculation methods
  - Improve the parallelism of each microoperation in the instruction
  - reduce the number of beats needed in the interpretation process

- Reduces the execution time of the entire program (machine language)
  - Through the control mechanism, the interpretation of the entire program can be speeded up by means of simultaneous interpretation of two, multiple or even whole programs.

- *Pipelining is an implementation technique whereby multiple instructions are overlapped in execution; it takes advantage of parallelism that exists among the actions needed to execute an instruction.*

- *Today, pipelining is the key implementation technique used to make fast CPUs.*

Pipelining

- *"A technique designed into some computers to increase speed by starting the execution of one instruction before completing the previous one."*

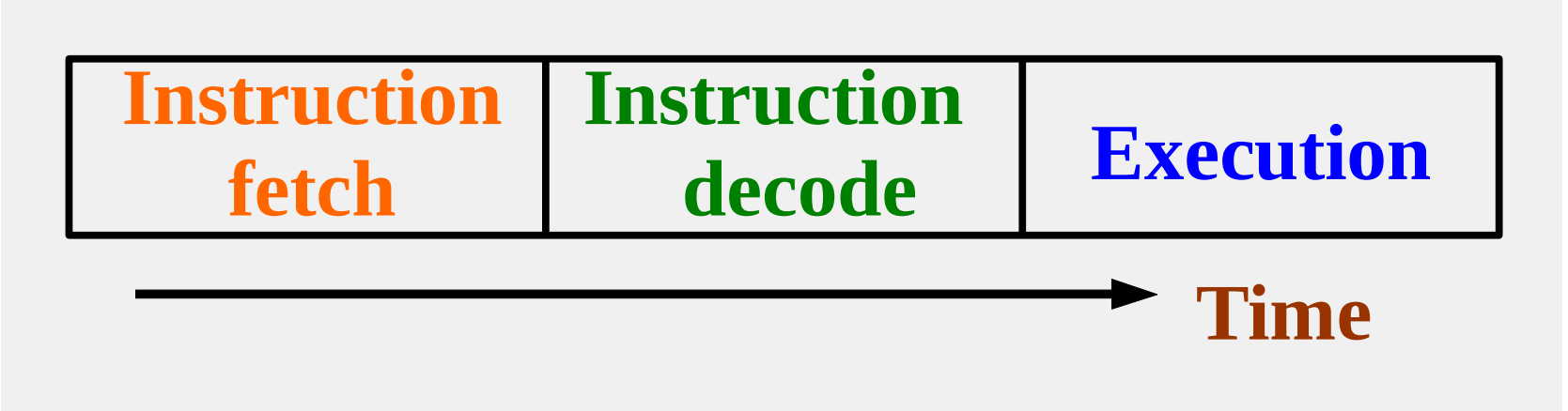
  ----Modern English-Chinese Dictionary

- implementation technique whereby different instructions are overlapped in execution at the same time.
- implementation technique to make fast CPUs

Three modes of execution

- Sequential execution
- Single overlapping execution
- Twice overlapping execution

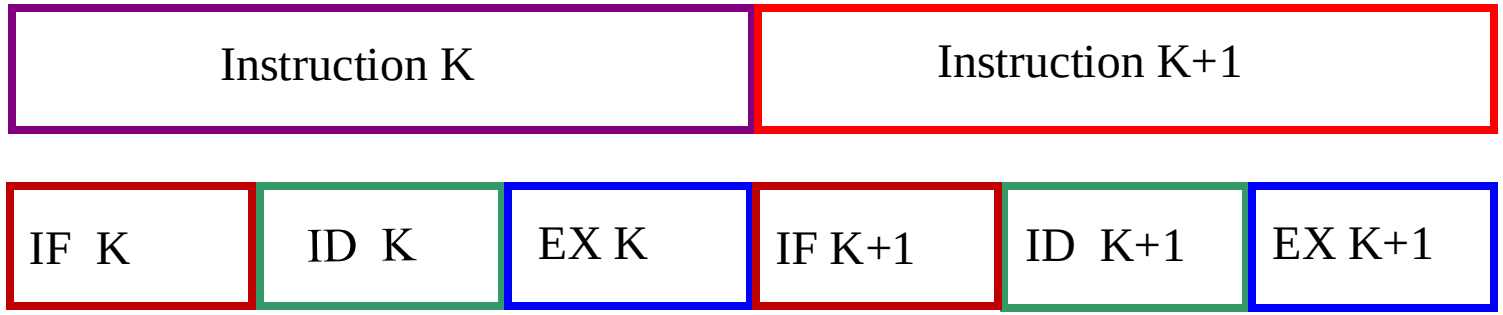
The execution of an instruction is divided into three stages：



The execution of an instruction

Sequential execution

- The execution of the instructions

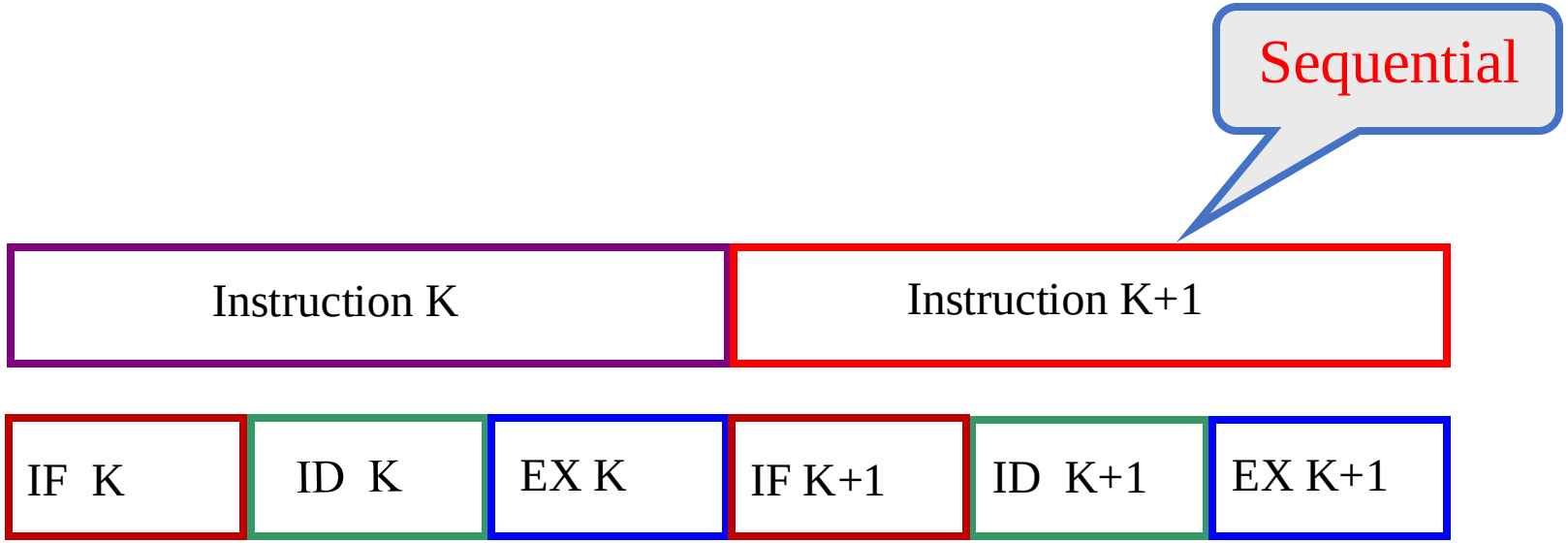


- Instructions are executed sequentially, and each microoperation in the instruction is also executed sequentially.

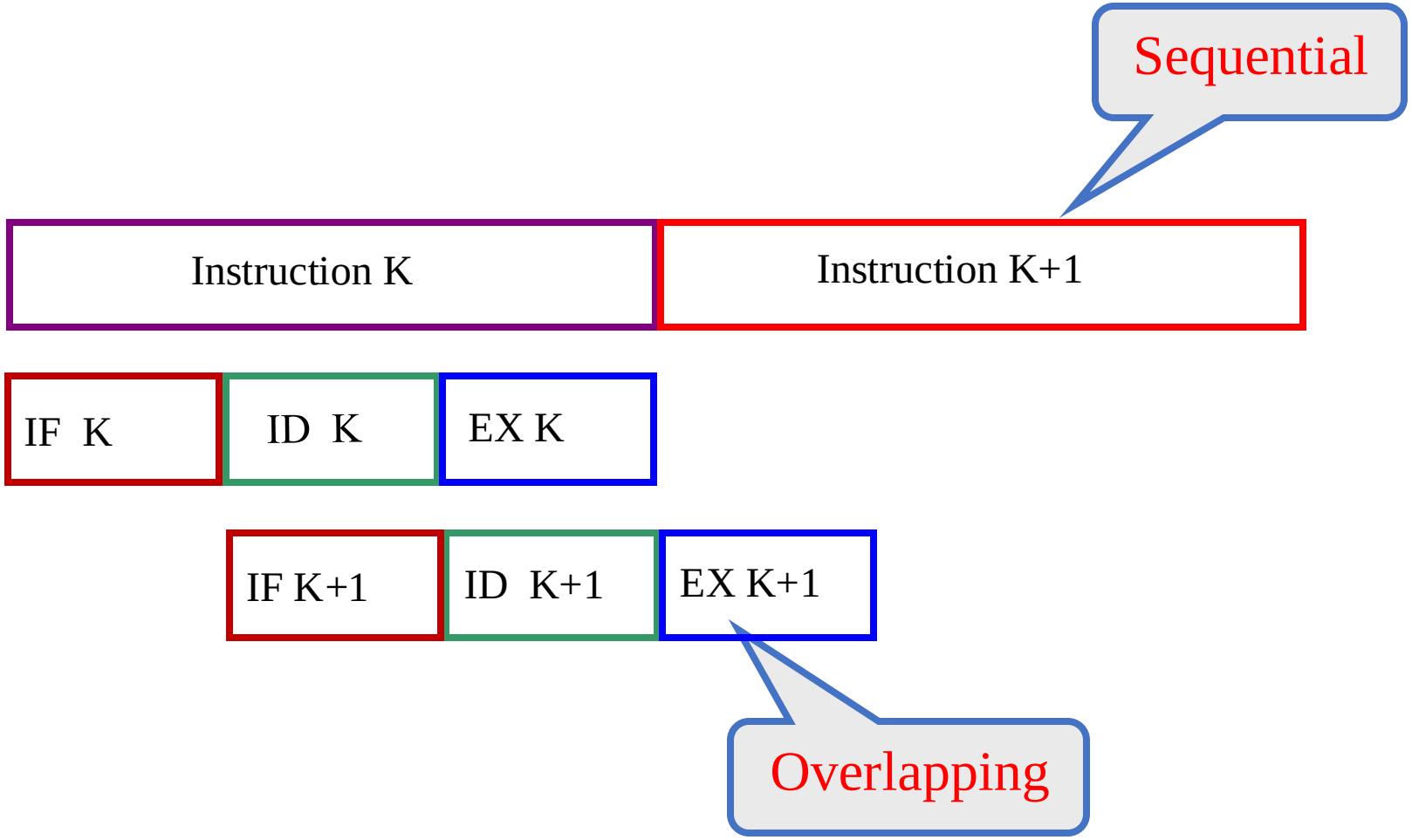
- Execution Time:

$$T = \sum_{i=1}^{n} (t_{IFi} + t_{IDi} + t_{EXi})$$

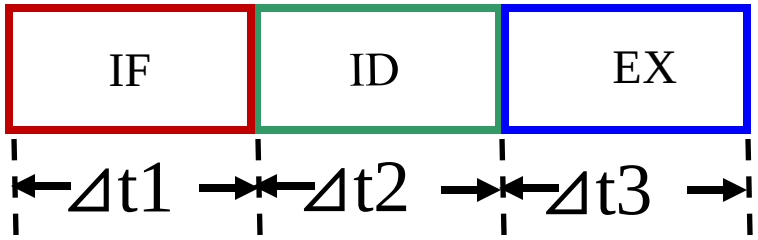
Overlapping execution



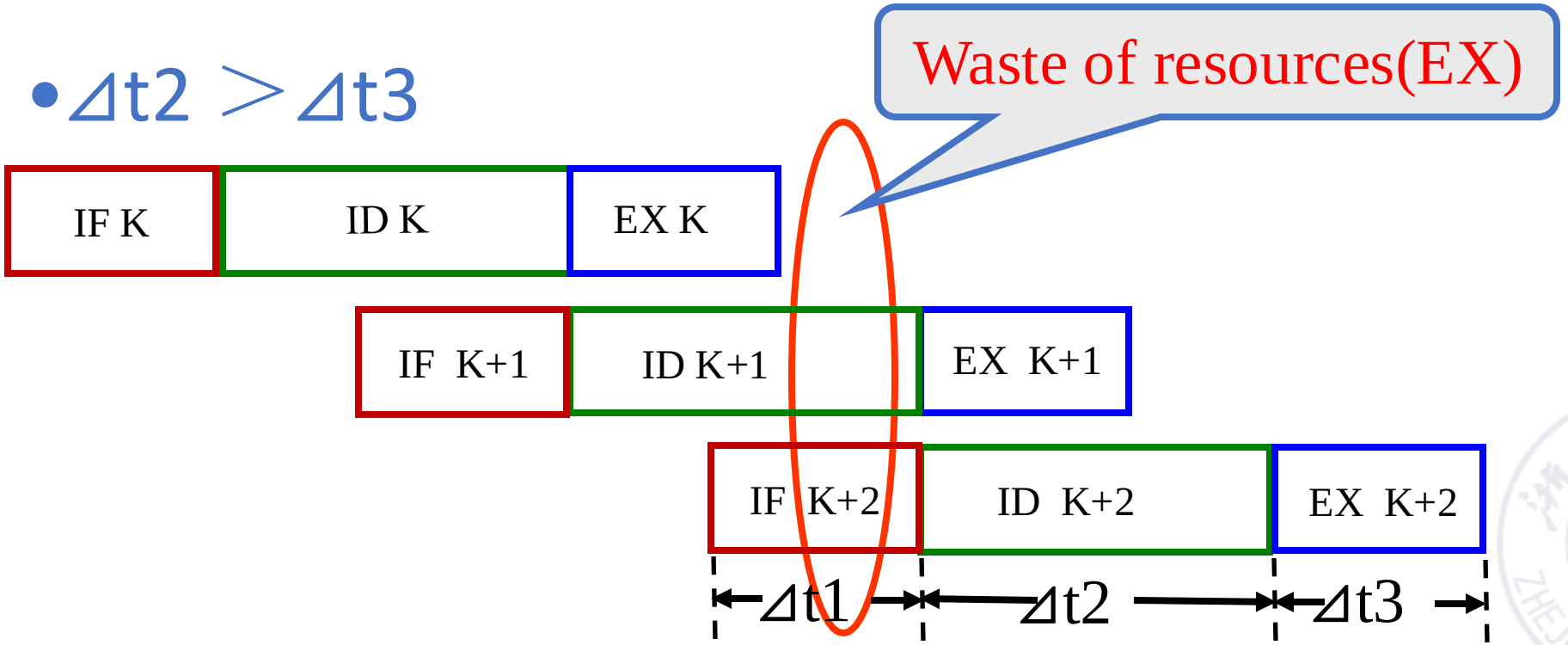
Overlapping execution



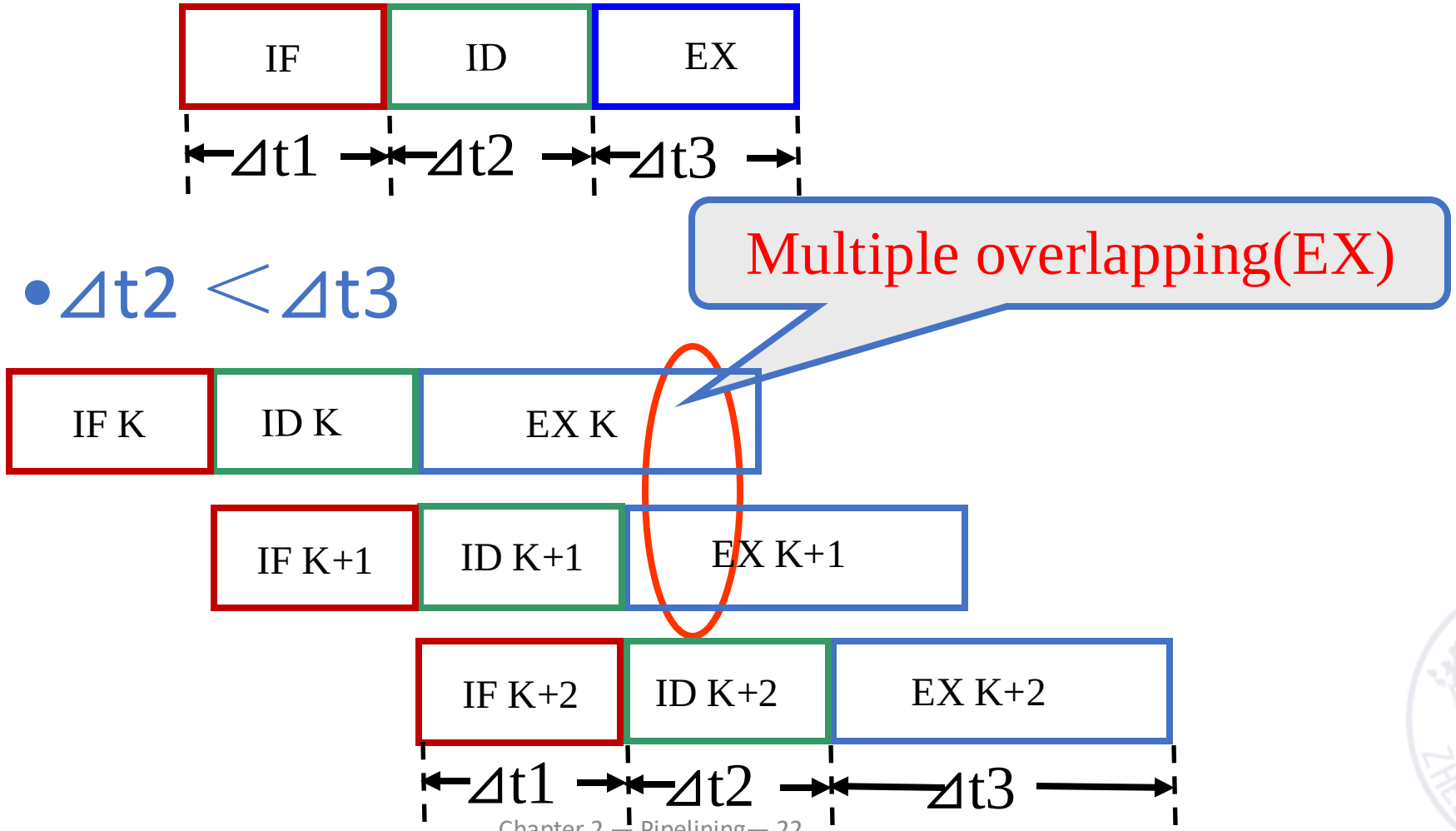
Overlapping execution



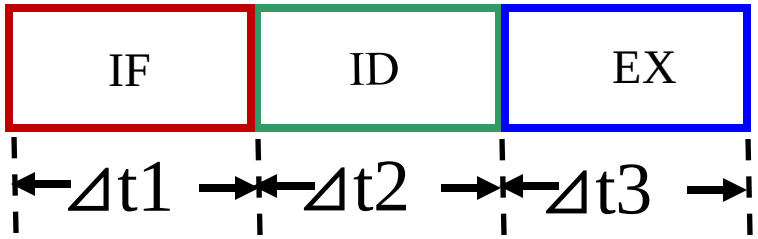
- ⊿t2 ＞⊿t3



Overlapping execution



Overlapping execution



- ⊿t1 =⊿t2 =⊿t3 = ⊿t

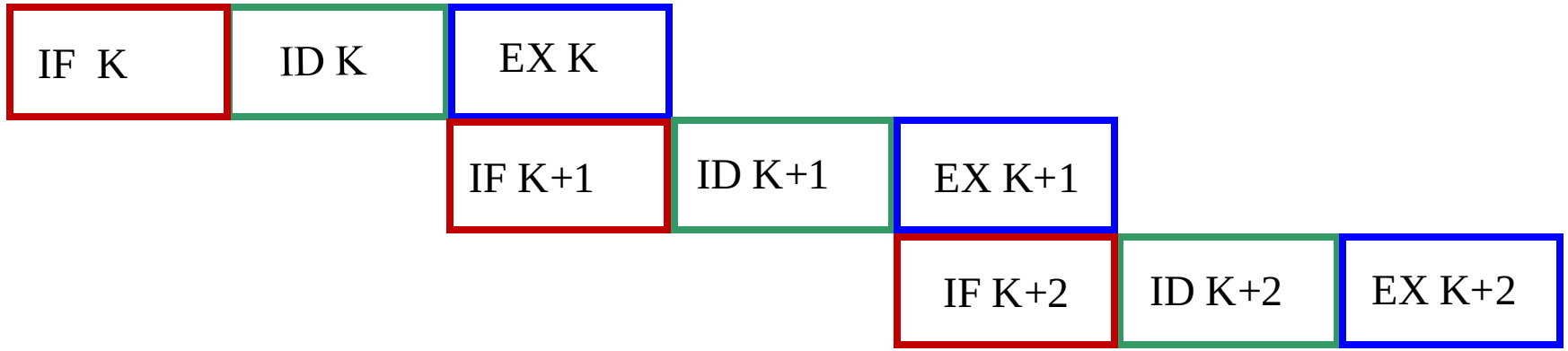
- Execute time for n instructions in sequence

$$T = \sum_{i=0}^{n} (t_{IFi} + t_{IDi} + t_{EXi}) = 3n\Delta t$$

- Sequential execution
  - ✓ Simple control, saving equipment
- Overlapping execution
  - ✓ High-usage of functional unit

Overlapping execution

- Single overlapping execution
- Execute the k instruction at the same time as fetch the k+1 instruction



- Execute time for n instructions by single overlapping execution

$$T = (2n + 1)\Delta t$$

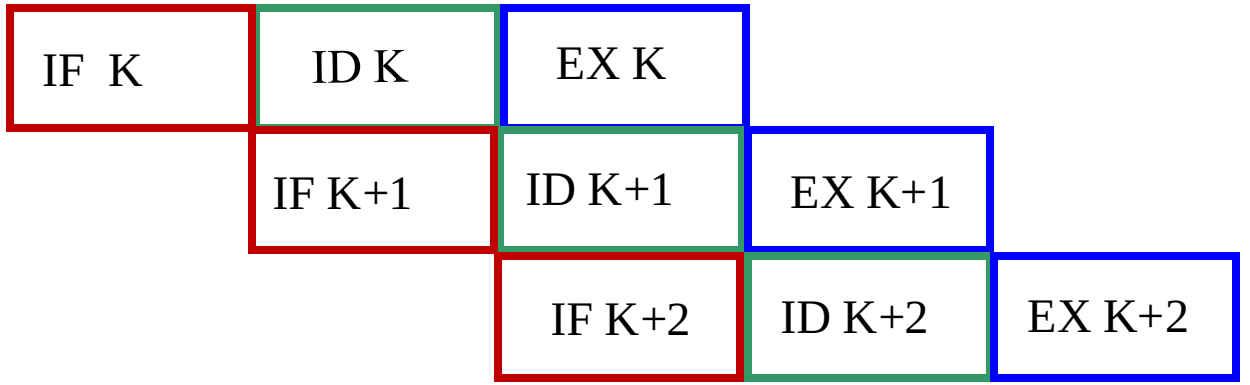
Overlapping execution

- Advantages
  - Execution time was reduced by nearly 1/3.
  - The utilization rate of functional unit is improved obviously.

- Disadvantages
  - Some additional hardware was needed, and the control process became complicated

Overlapping execution

- Twice overlapping execution
- Decode the k instruction at the same time as fetch the k+1 instruction
- Execute the k instruction at the same time as decode the k+1 instruction



- Execute time for n instructions by twice overlapping execution

$$T = (n + 2)\Delta t$$

Twice Overlapping execution

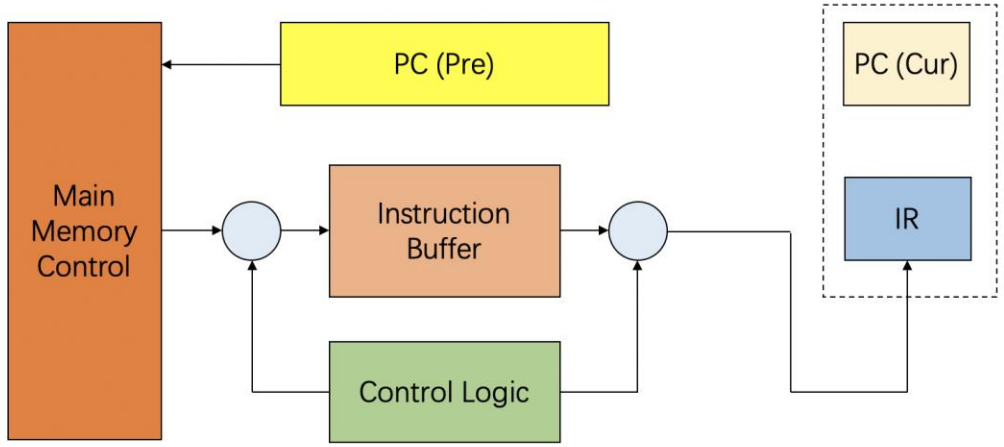
- Advantages
  - Execution time was reduced by nearly 2/3.
  - The utilization rate of functional unit is improved obviously.

- Disadvantages
  - Much more hardware was needed.
  - Separate fetch, decode, and execution components are required.

Overlapping execution

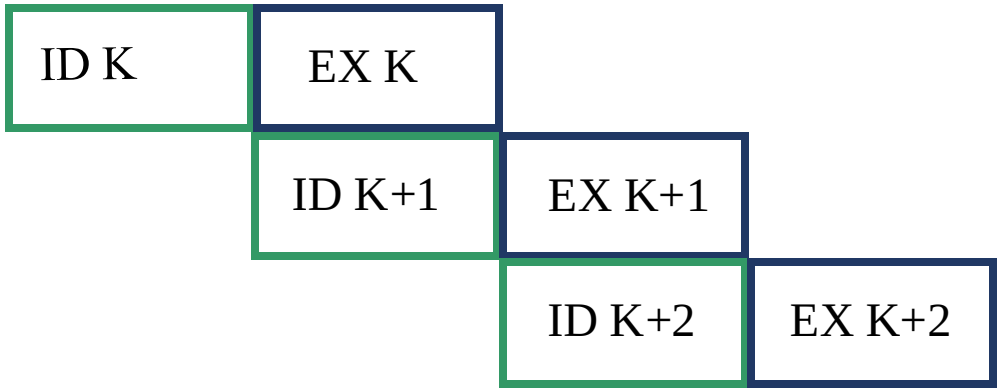
- Conflict in access memory
  - Instruction memory & data memory
  - Instruction cache & data cache (same memory): Harvard structure
  - Multibody cross structure (same memory with limitations)
  - Adding instruction buffer between memory and instruction decode unit

Instruction Buffer Structure



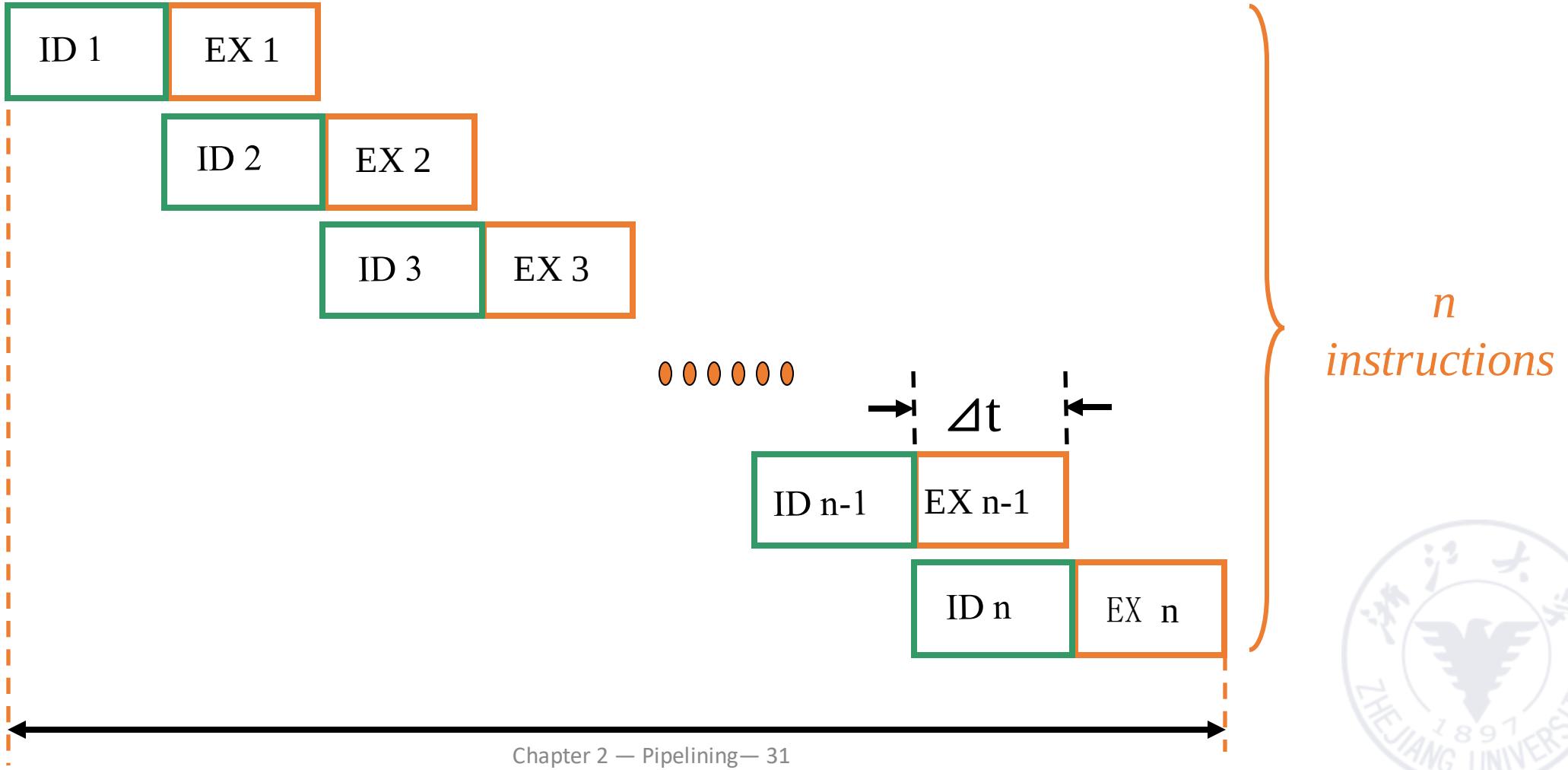
Single Overlapping execution

- If the time of fetching instruction phase is very short, fetch operation can be incorporated into the decode operation. The twice overlapping becomes single overlapping.

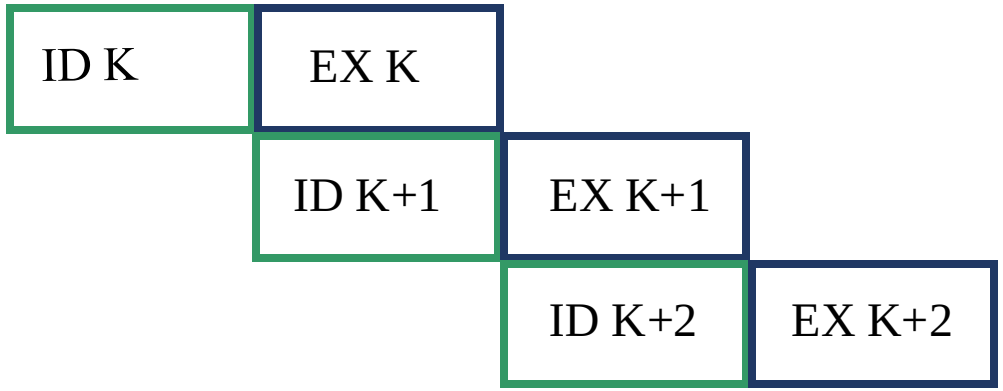


Single Overlapping execution

Single Overlapping execution

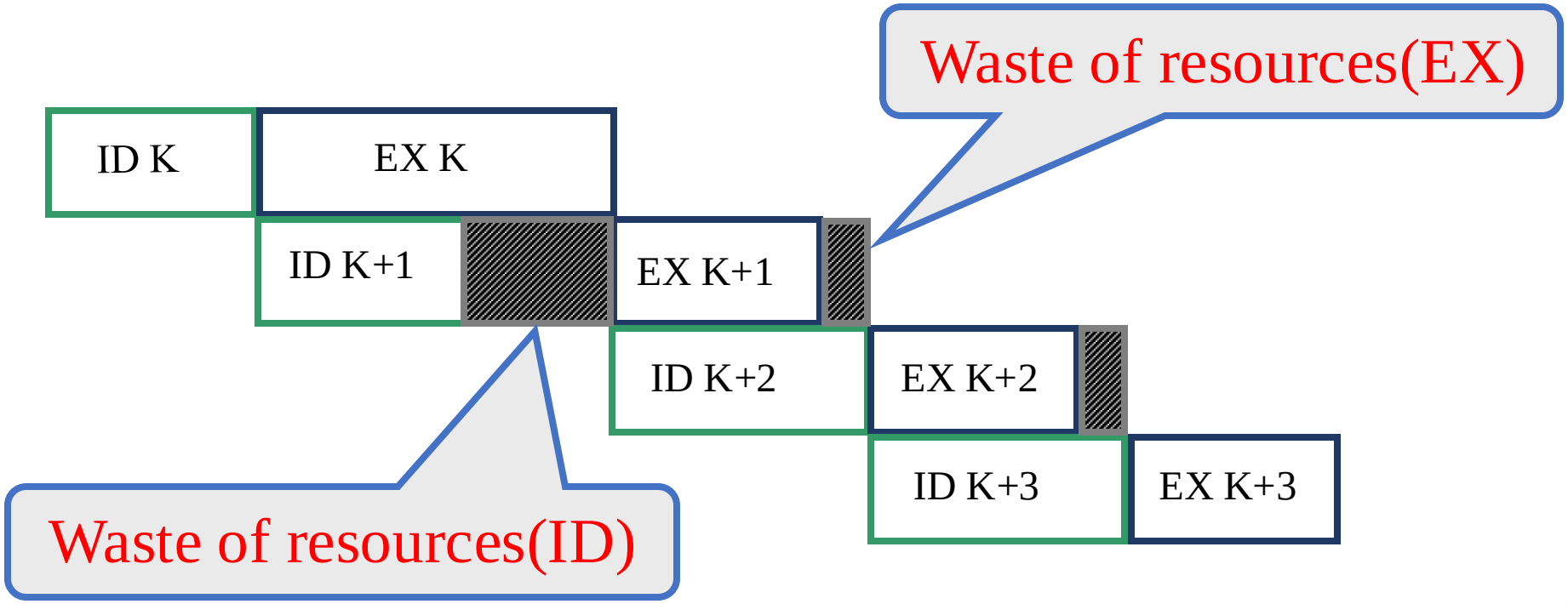


Single Overlapping execution



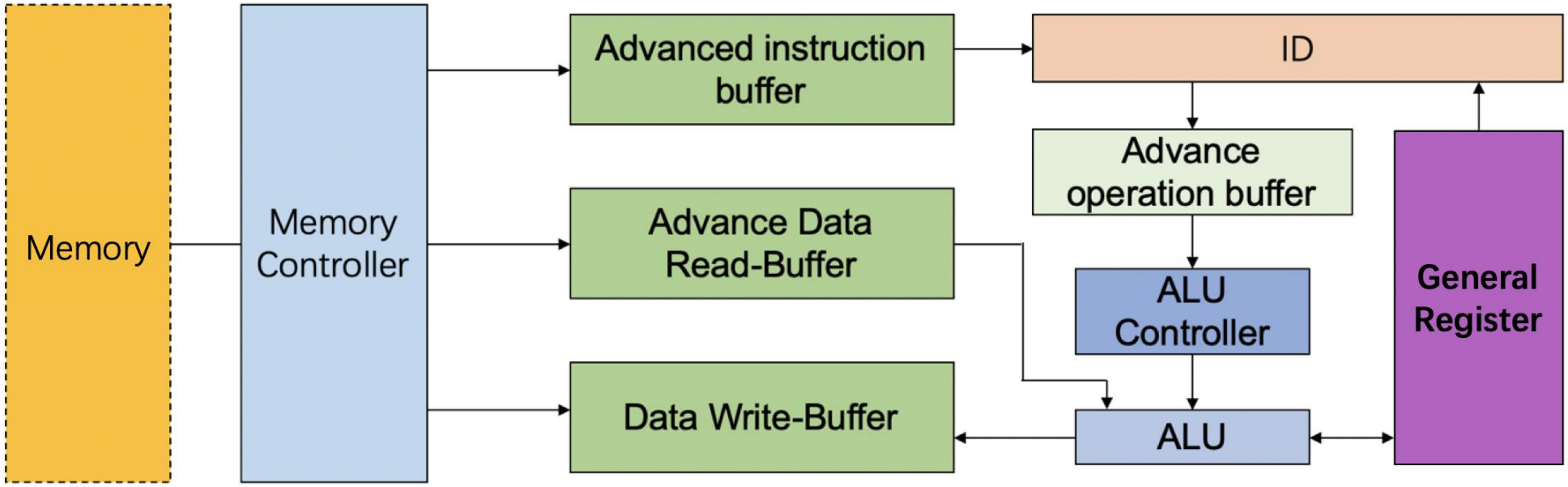
- $\Delta t_{ID} = \Delta t_{EX}$

Single Overlapping execution



- $\Delta t_{ID} \neq \Delta t_{EX}$

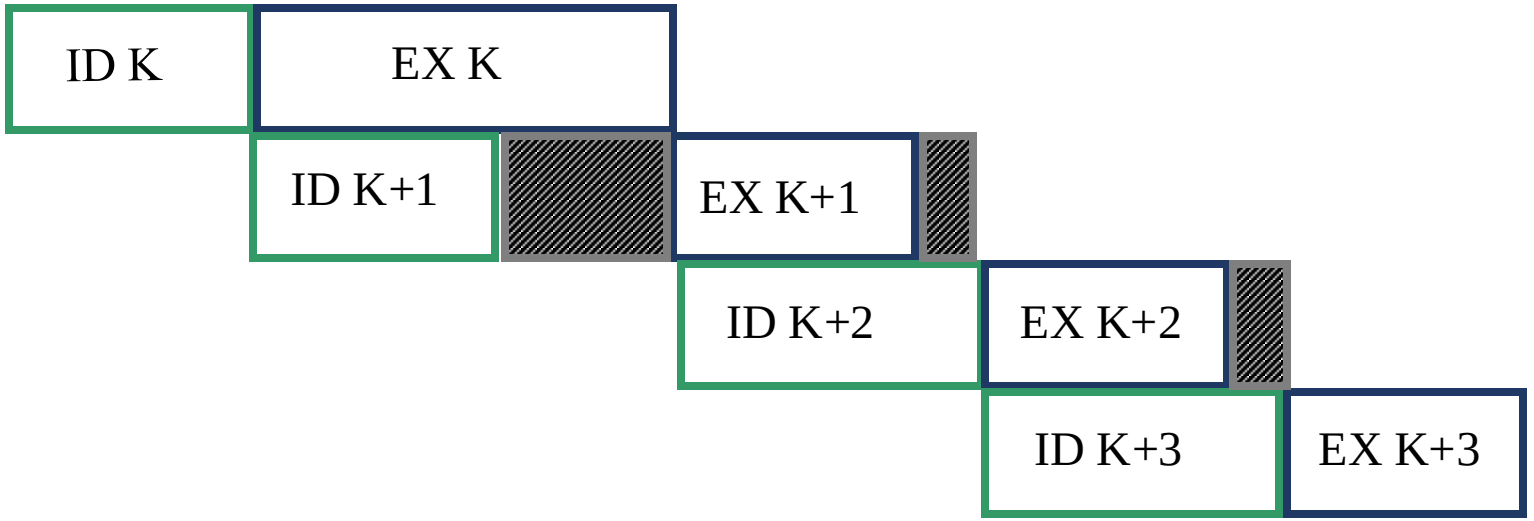
Single Overlapping execution



The structure of processor with advance control

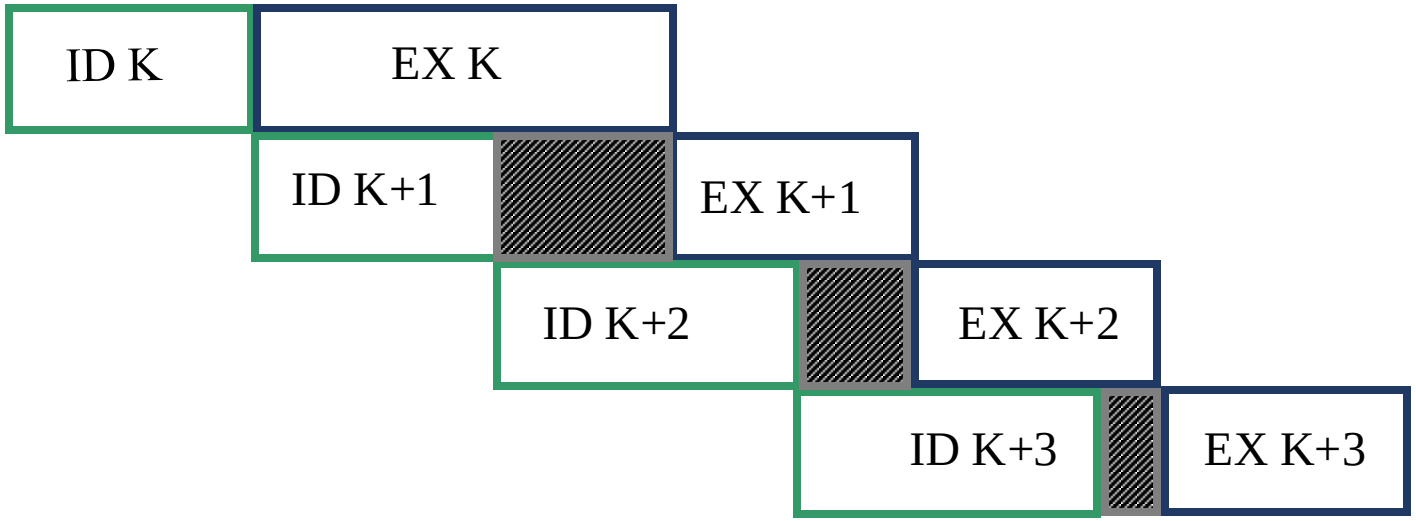
Common features: They work by FIFO, and are composed of a group of several storage units that can be accessed quickly and related control logic.

Single Overlapping execution



- $\Delta t_{ID} \neq \Delta t_{EX}$

Single Overlapping execution with advance control



- $\Delta t_{ID} \neq \Delta t_{EX}$

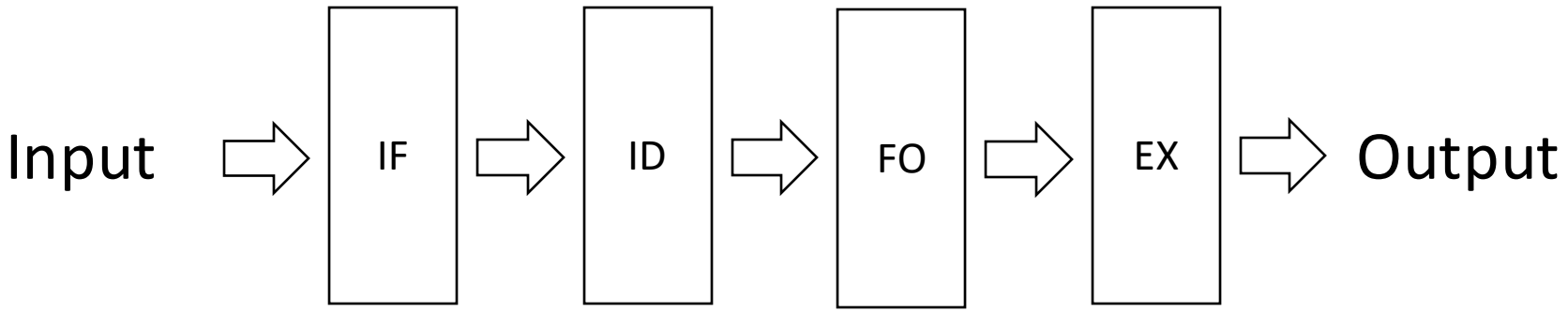
$$T_{Advanced} = t_{ID1} + \sum_{i=1}^{n} t_{EXi} \approx \sum_{i=1}^{n} t_{EXi}$$

What is pipelining ?

- Pipelining: The process of an instruction is divided into m (m > 2) sub processes with equal time, and the process of m adjacent instructions are staggered and overlapped in the same time.

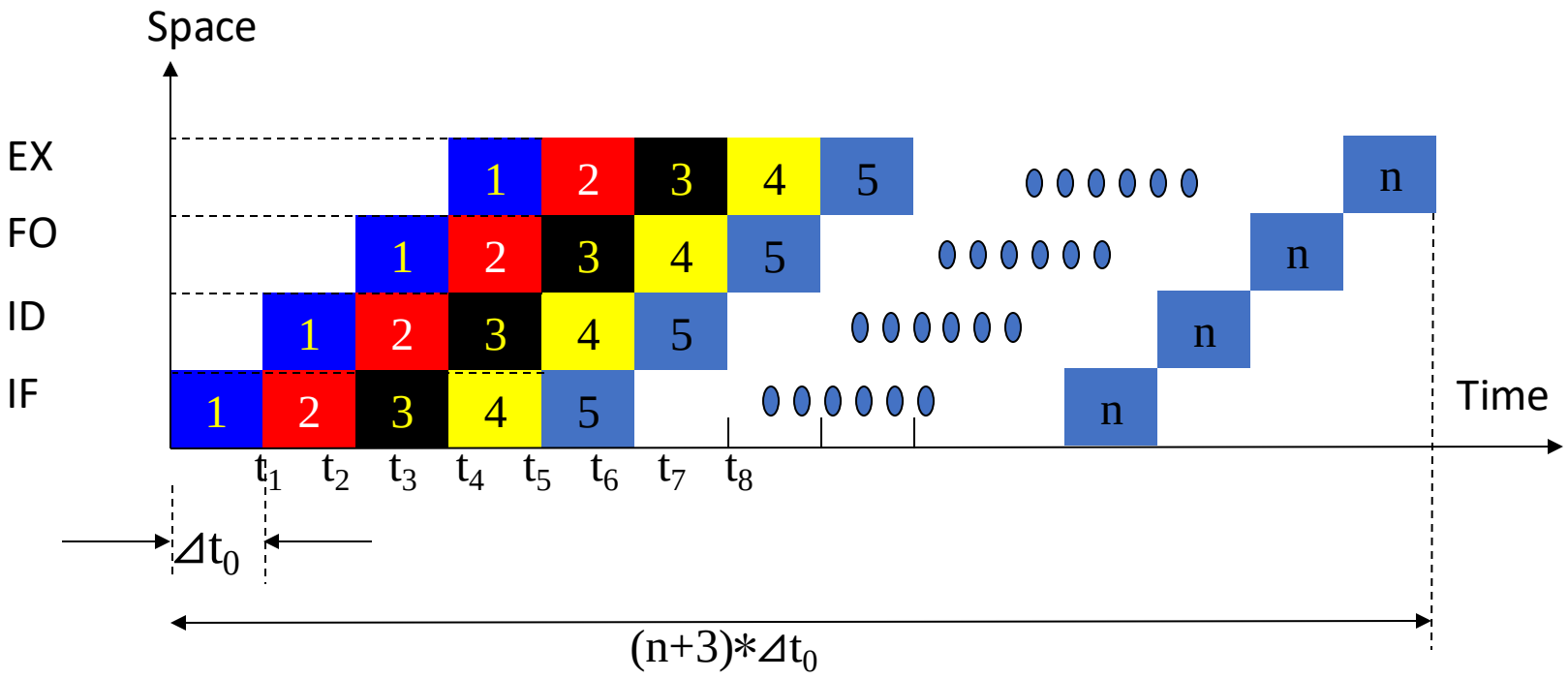
- Pipelining can be regarded as the extension of overlapping execution.
- Each subprocess and its functional components in the pipelining are called stages or segments of the pipelining, which are connected to form a pipelining.
- The number of segments in a pipelining is called the depth of pipelining.

What is pipelining ?



Pipelining

What is pipelining ?



How much faster?

Characteristics of pipelining

- The pipelining divides a process into several sub processes, each of which is implemented by a special functional unit.
- The time of each section in the pipelining should be equal as much as possible, otherwise the pipelining will be blocked and cut off. A longest section will become the bottleneck of the pipelining.
- Every functional part of the pipelining must have a buffer register (latch), which is called pipelining register.

- Function: transfer data between two adjacent sections to ensure the data to be used later, and separate the processing work of each section from each other.

Characteristics of pipelining

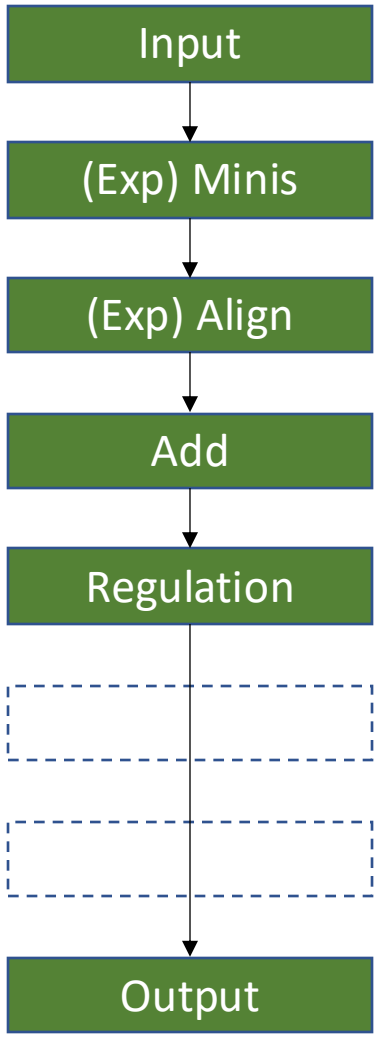
- Pipelining technology is suitable for a large number of repetitive sequential processes. Only when tasks are continuously provided at the input, the efficiency of pipelining can be brought into full play.

- The pipelining needs the pass time and the empty time
  - Pass time: the time for the first task from beginning (entering the pipelining) to ending.
  - Empty time: the time for the last task from entering the pipelining to having the result.

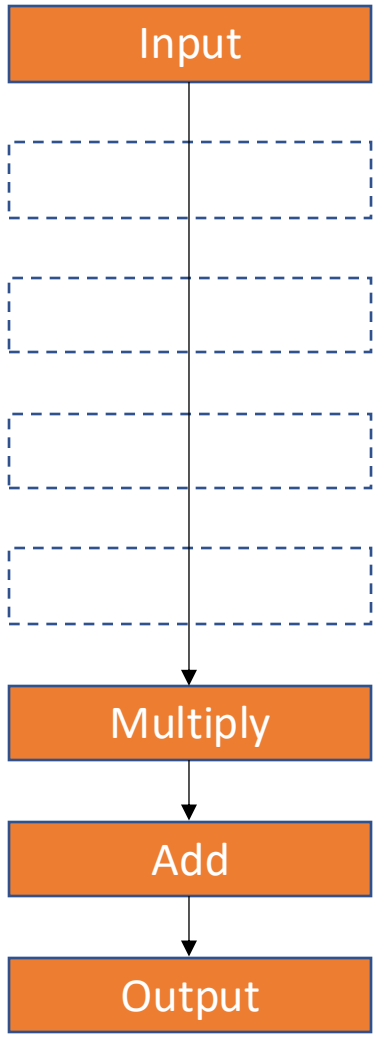
Characteristics of pipelining

- Single function pipelining: only one fixed function pipelining.
- Multi function pipelining: each section of the pipelining can be connected differently for several different functions.

Segmentation



Floating Point Arithmetic



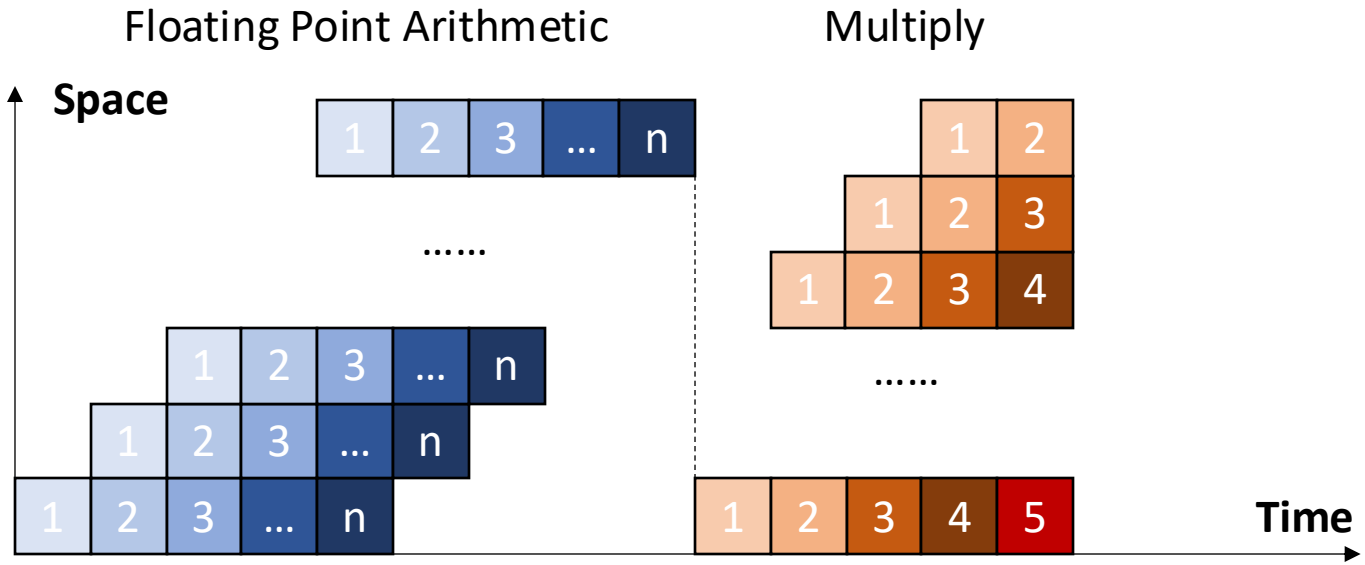
Multiply

Characteristics of pipelining

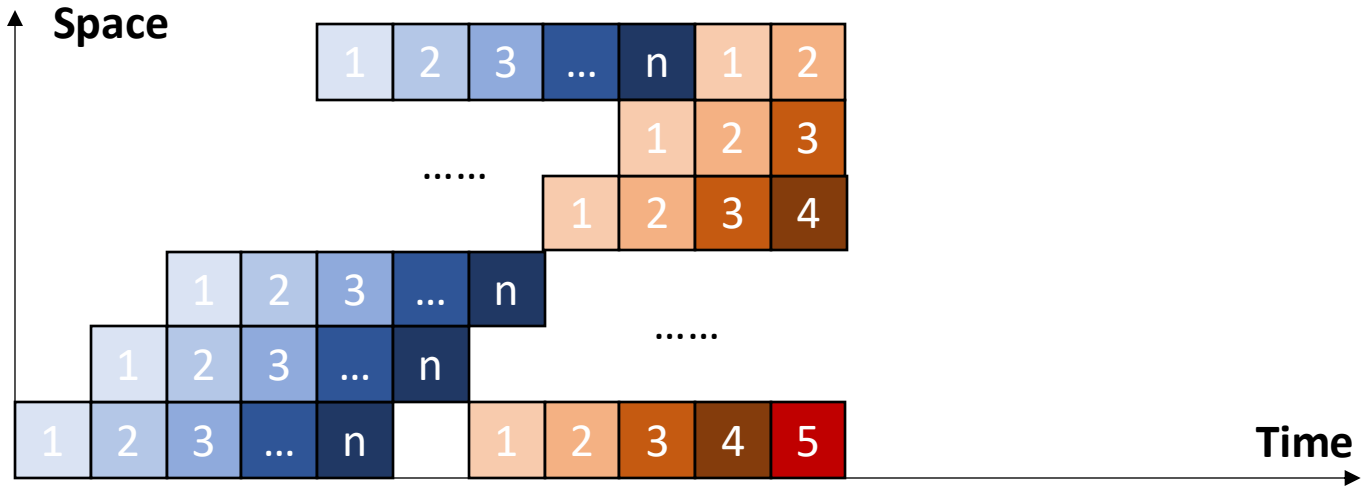
- Static pipelining: In the same time, each segment of the multi-functional pipelining can only work according to the connection mode of the same function.
  - For static pipelining, only the input is a series of the same operation tasks, the efficiency of pipelining can be brought into full play.

- Dynamic pipelining: In the same time, each segment of the multi-functional pipelining can be connected in different ways and perform multiple functions at the same time.
  - It is flexible but with complex control.
  - It can improve the availability of functional units.

Static Pipelining



Dynamic Pipelining

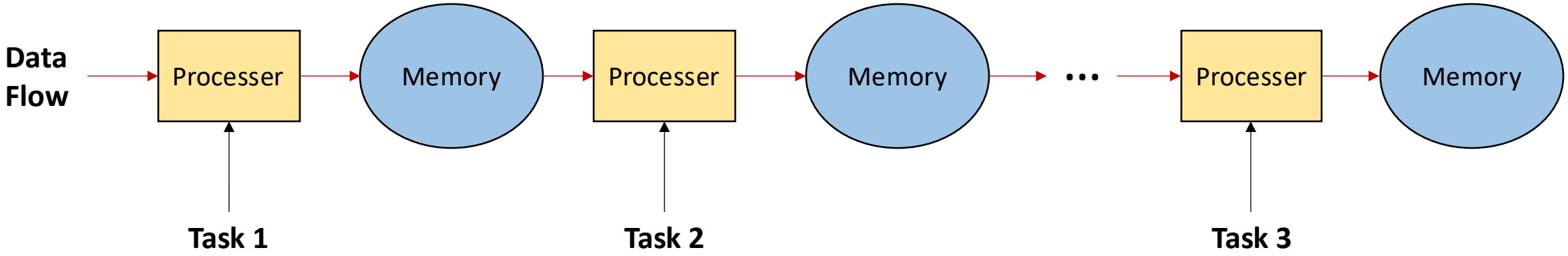


Component level pipelining (in component - operation pipelining) : The arithmetic and logic operation components of the processor are divided into segments, so that various types of operation can be carried out by pipelining.

Processor level pipelining (inter component - instruction pipelining): The interpretation and execution of instructions are implemented through pipelining. The execution process of an instruction is divided into several sub processes, each of which is executed in an independent functional unit.

Inter processor pipelining (inter processor - macro pipelining): It is a serial connection of two or more processors to process the same data stream, and each processor completes a part of the whole task.

§2.3 Classes of pipelining



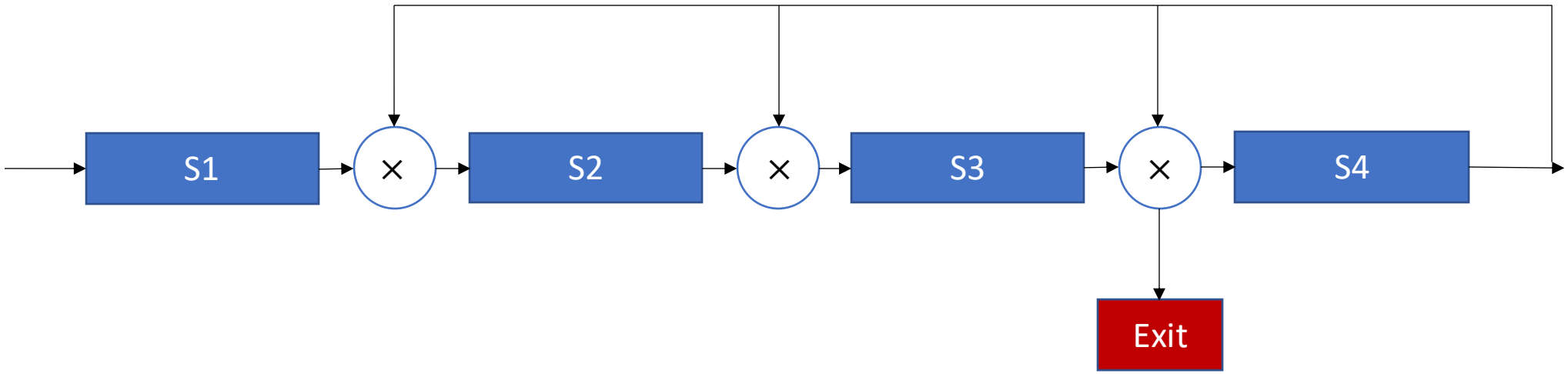
Linear pipelining: Each section of the pipelining is connected serially without feedback loop. When data passes through each segment in the pipelining, each segment can only flow once at most.

Nonlinear pipelining: In addition to the serial connection, there is also a feedback loop in the pipelining.

Scheduling problem of nonlinear pipelining.

Determine when to introduce a new task to the pipelining, so that the task will not conflict with the task previously entering the pipelining.

Nonlinear pipelining



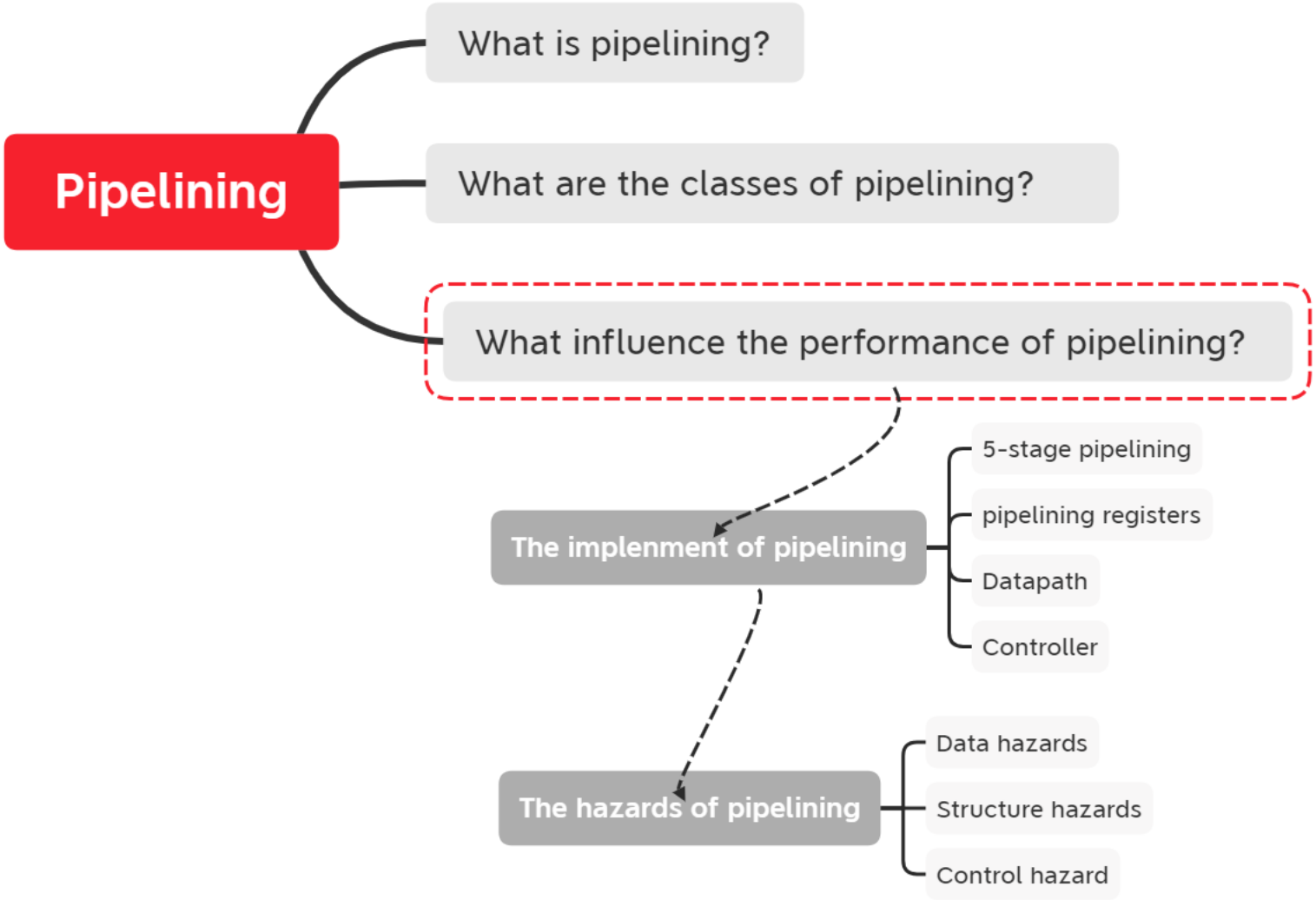
Task: → S_1 → S_2 → S_3 → S_4 → S_2 → S_3 → S_4 → S_3 →

Ordered pipelining: In the pipelining, the outflow order of tasks is exactly the same as the inflow order. Each task flows by sequence in each segment of the pipelining.

Disordered pipelining: In the pipelining, the outflow order of tasks is not the same as the inflow order. The later tasks are allowed completed first.

Scalar processor: The processor does not have vector data representation and vector instructions, and only deal with scalar data through pipelining.

Vector pipelining processor: The processor has vector data representation and vector instructions. It is the combination of vector data representation and pipelining technology.


Pipelining
What is pipelining?
What are the classes of pipelining?
What influence the performance of pipelining?
The implenment of pipelining
5-stage pipelining
pipelining registers
Datapath
Controller
The hazards of pipelining
Data hazards
Structure hazards
Control hazard

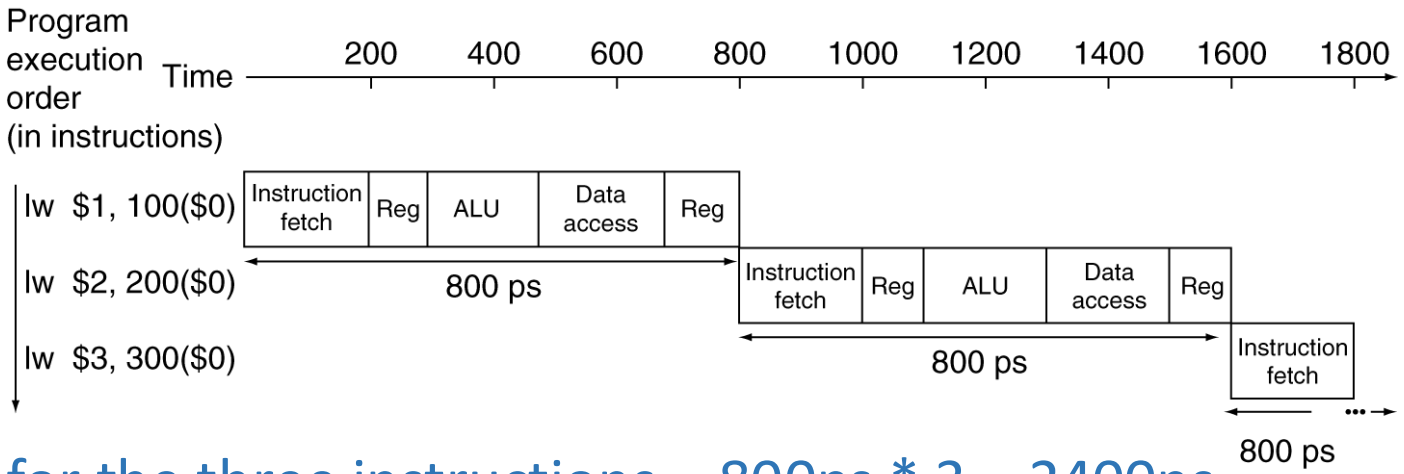
Pipeline Performance

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

Inst	Inst 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-type	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

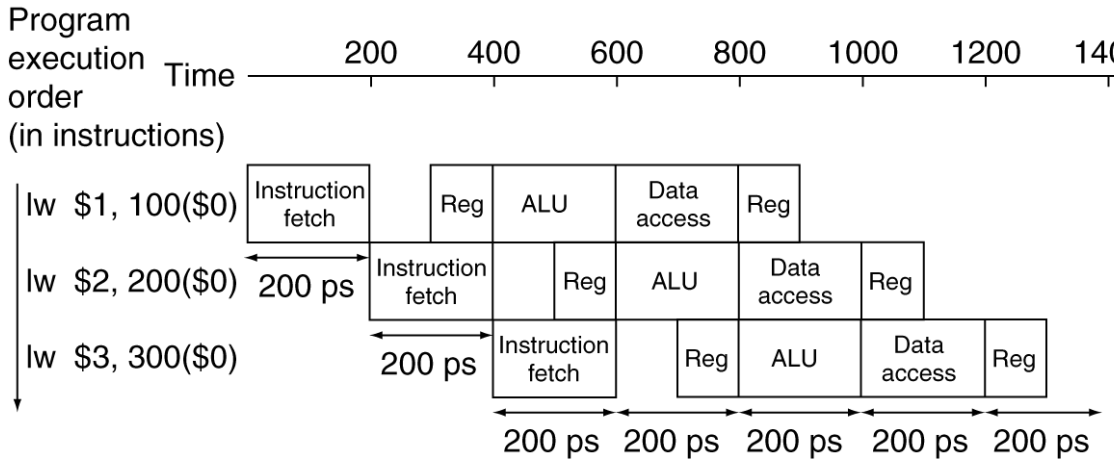
Pipeline Performance

Single-cycle (T_c= 800ps)



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

Pipelined (T_c= 200ps)



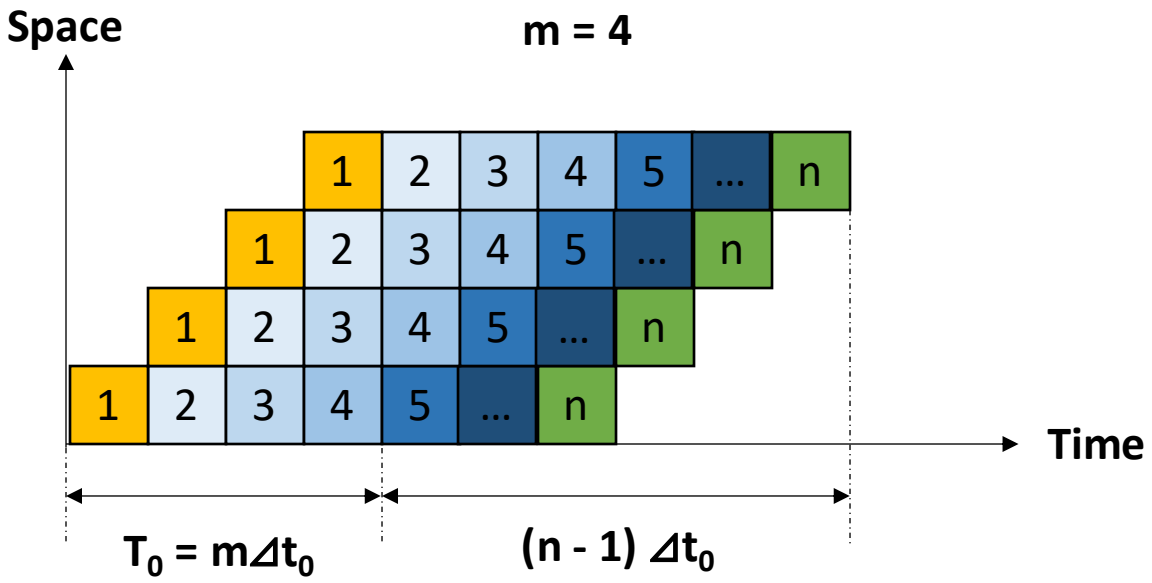
Total execution time for the three instructions = 200ps * 7 = 1400ps

Throughput (TP)

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

$$TP < TP_{max}$$

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



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

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

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

Pipeline Performance

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

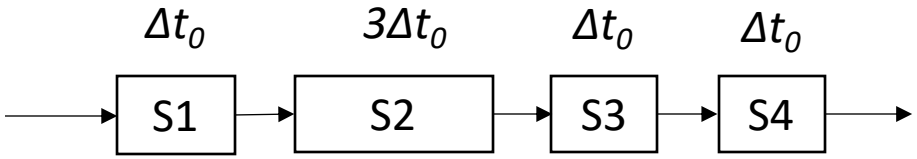
- The actual throughput of the pipeline is less than the maximum throughput, which is not only related to the time of each segment, but also related to m and n.

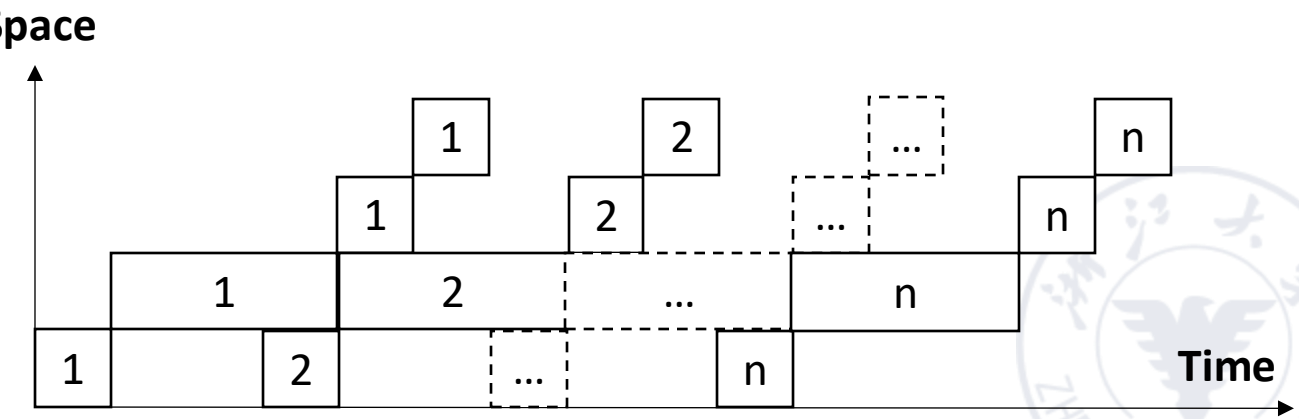
- If $n >> m$, $TP \approx TP_{max}$

Pipeline Performance

- 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.



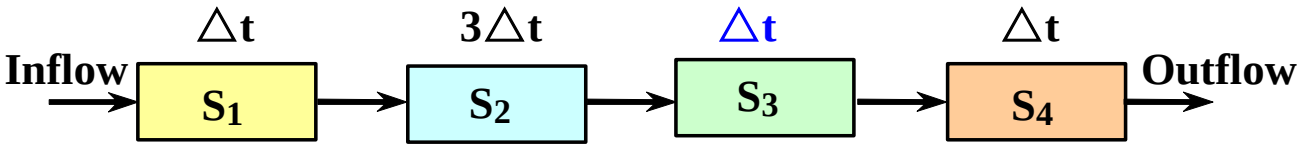


Pipeline Performance

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

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

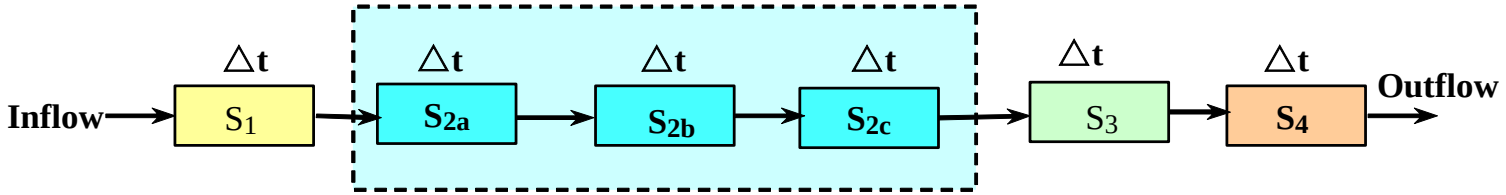
$$TP_{max} = \frac{1}{max(\Delta t_1, \Delta t_2, \ldots, \Delta t_m)}$$



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

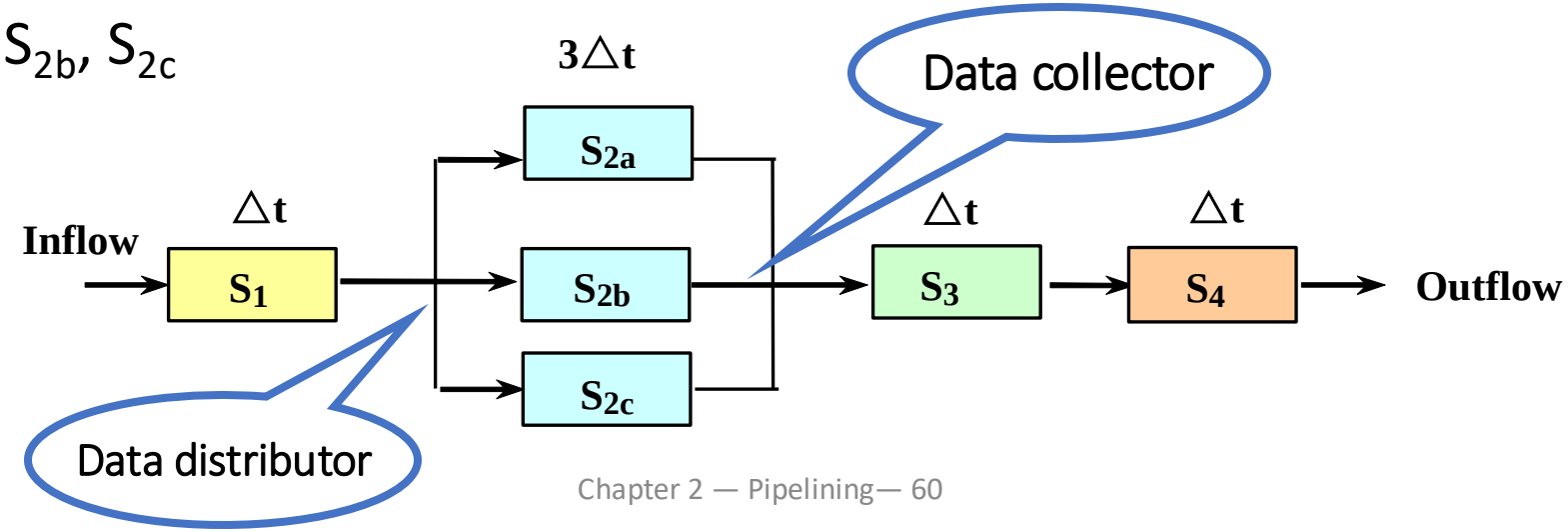
Common methods to solve pipeline bottleneck

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



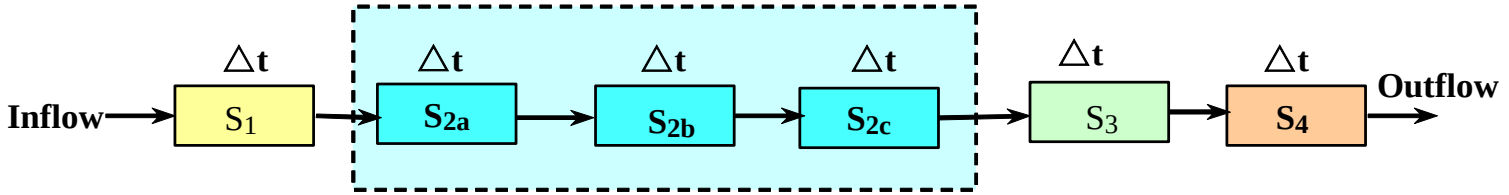
$$TP_{max} = \frac{1}{\Delta t}$$

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



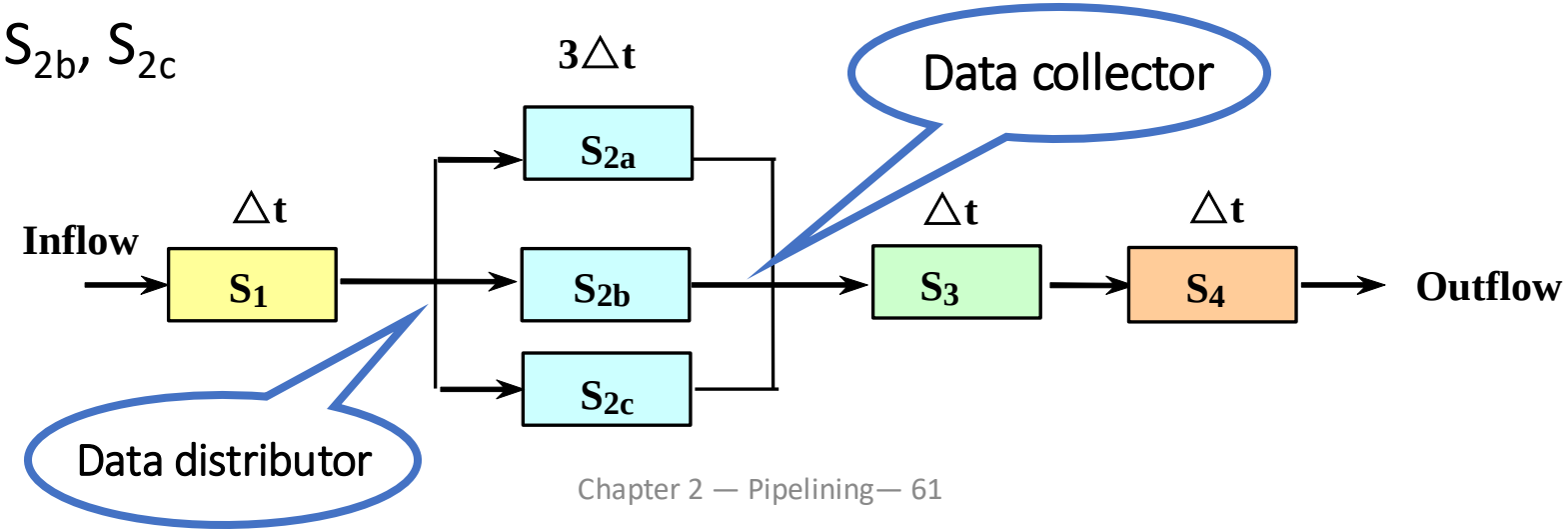
Common methods to solve pipeline bottleneck

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

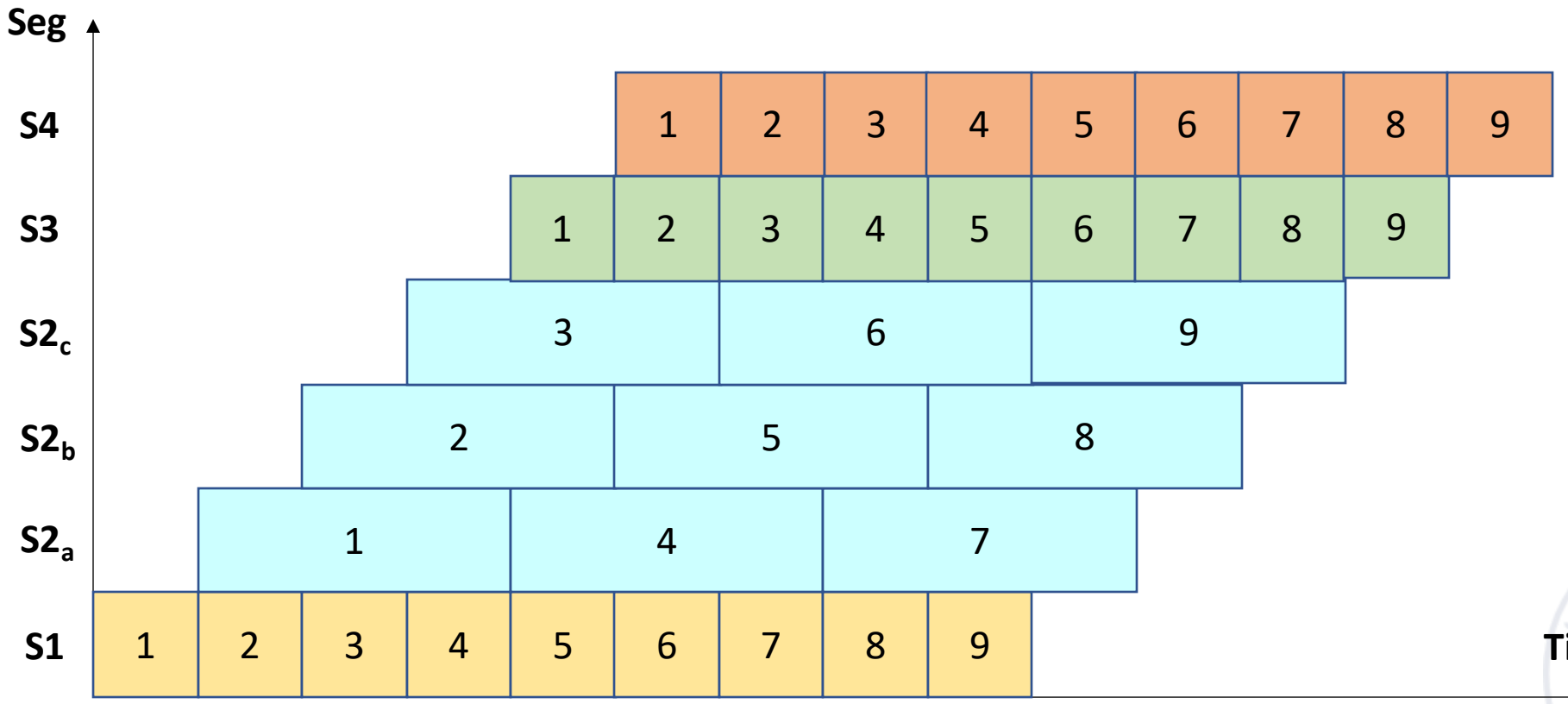


$$TP_{max} = \frac{1}{\Delta t}$$

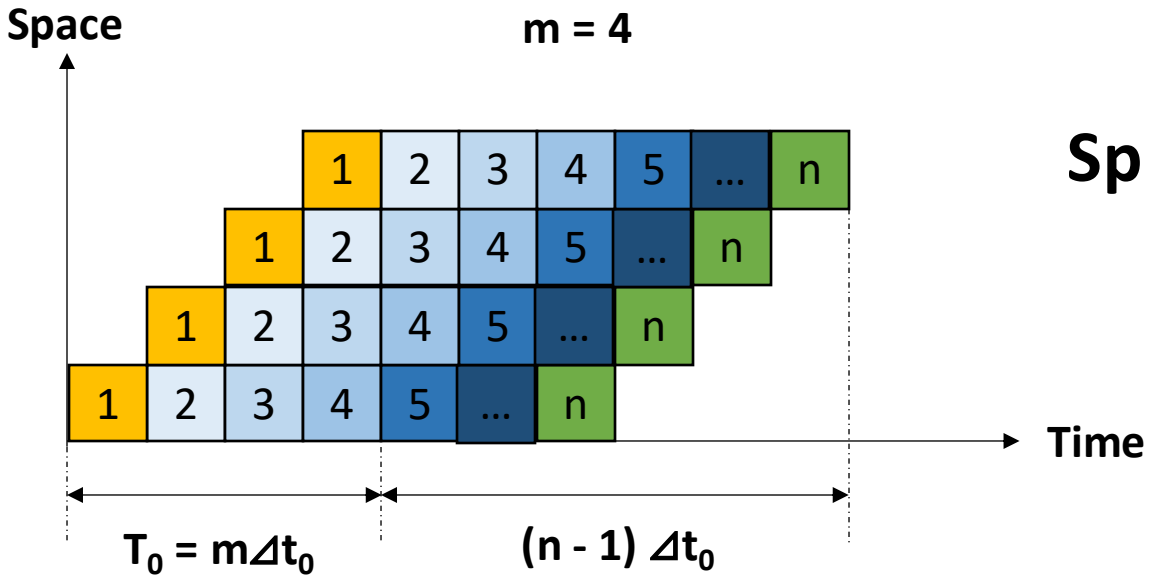
- Repetition
  - S_2: S_{2a}, S_{2b}, S_{2c}



Time space diagram with repetition bottleneck segment



Speedup (Sp)

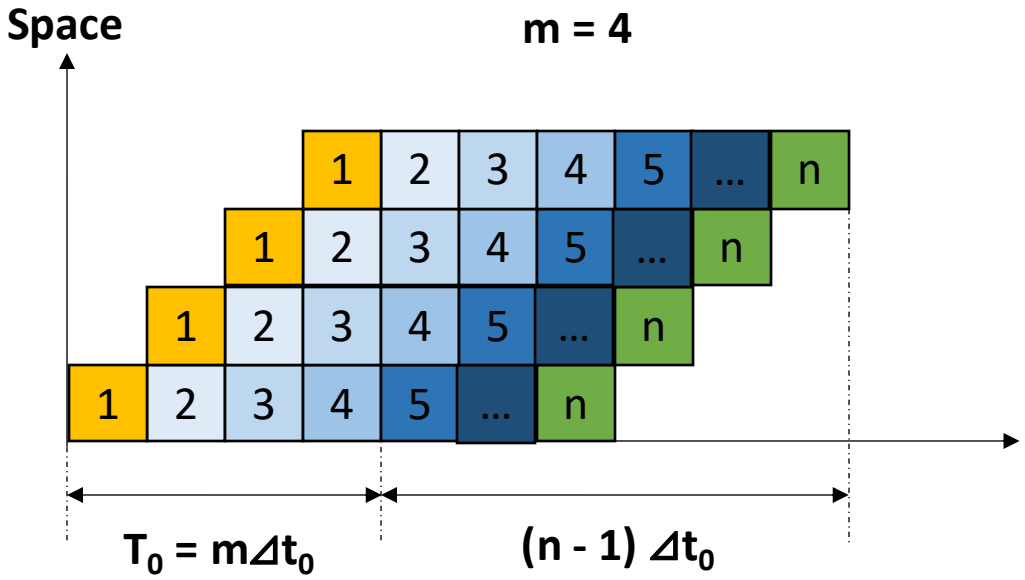


$$T = (m + n - 1) \times \Delta t_0$$

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

If n>>m,
Sp ≈ m

Efficiency (η)



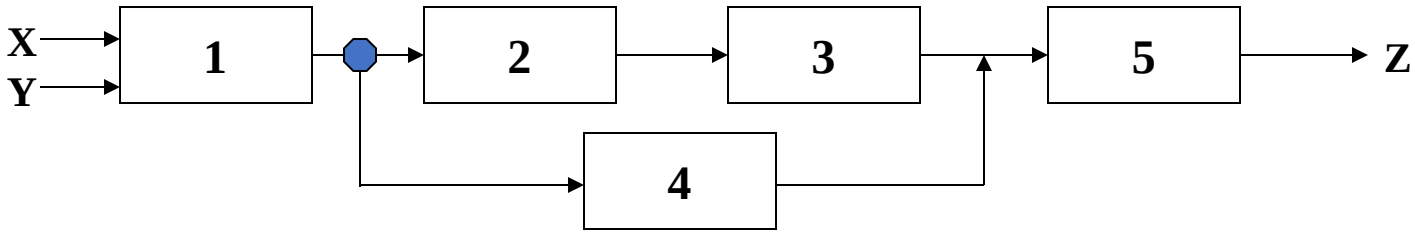
$$T = (m + n - 1) \times \Delta t_0$$

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

If n>>m,
η ≈ 1

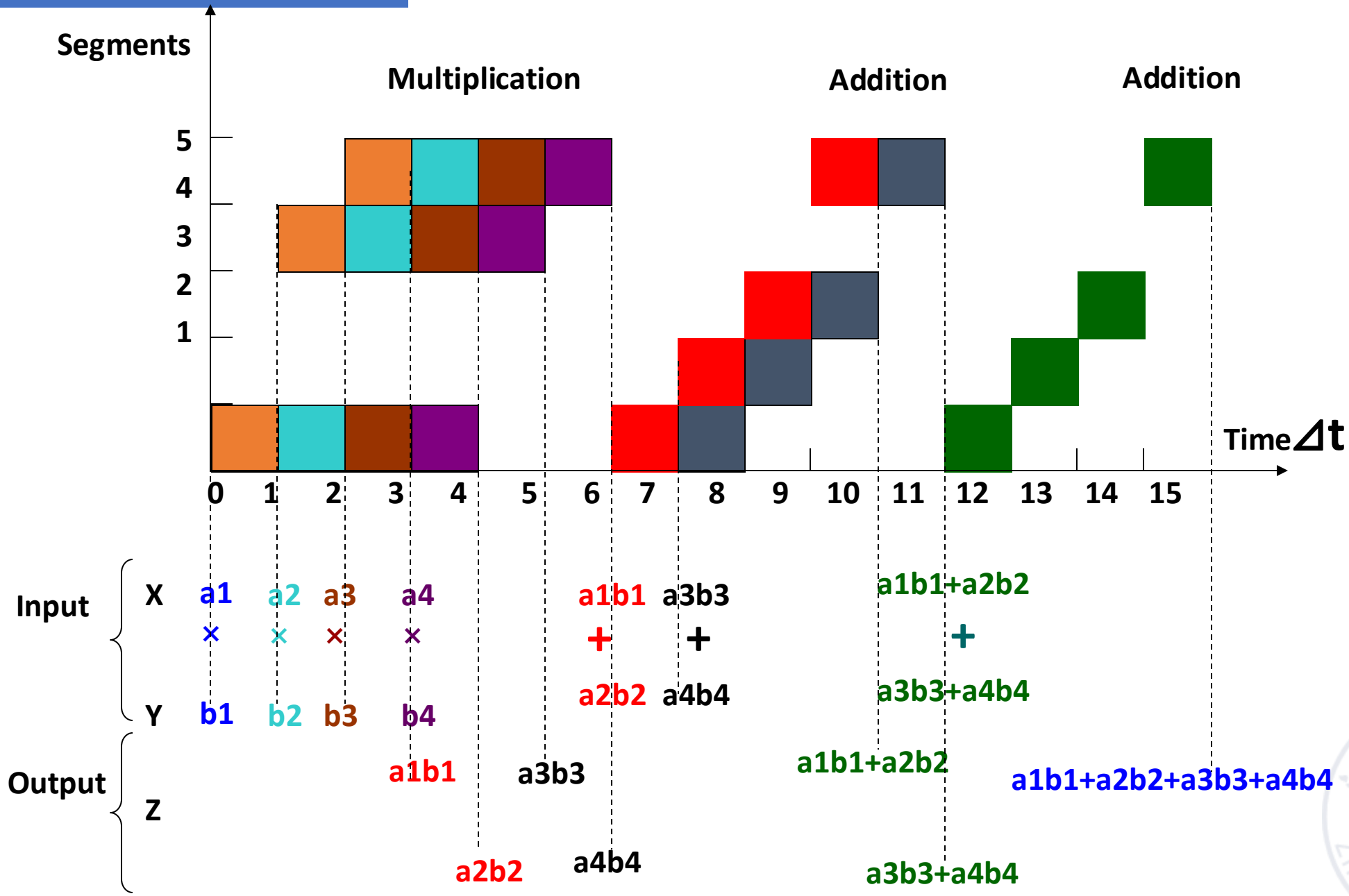
Pipeline Performance

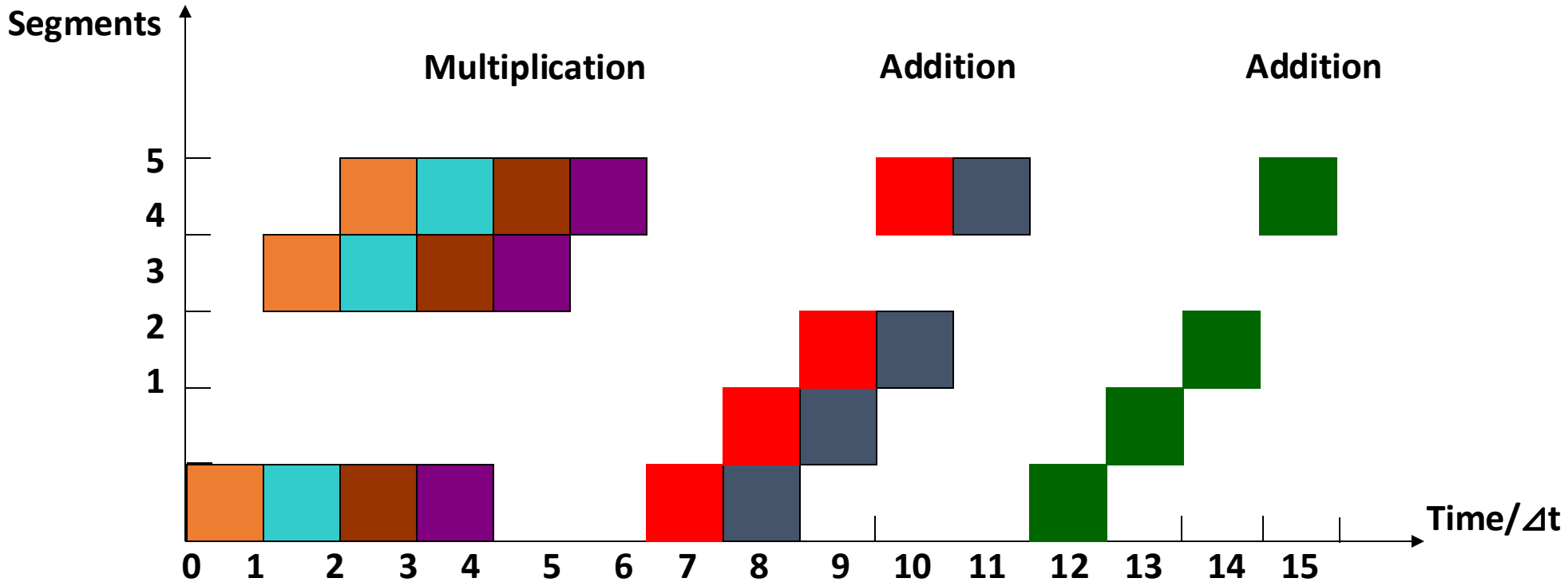
- Vector A(a1, a2,a3,a4)
- Vector B(b1,b2,b3,b4)
- Compute vector dot product (A·B) in the static dual function pipelining.



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

TP? Sp? η?


Segments
Multiplication
Addition
Addition
5
4
3
2
1
Time Δt
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Input
X
Y
a1 a2 a3 a4
b1 b2 b3 b4
a1b1 a3b3
a2b2 a4b4
a1b1+a2b2
a3b3+a4b4
Output
Z
a1b1
a3b3
a2b2
a4b4
a1b1+a2b2
a3b3+a4b4
a1b1+a2b2+a3b3+a4b4



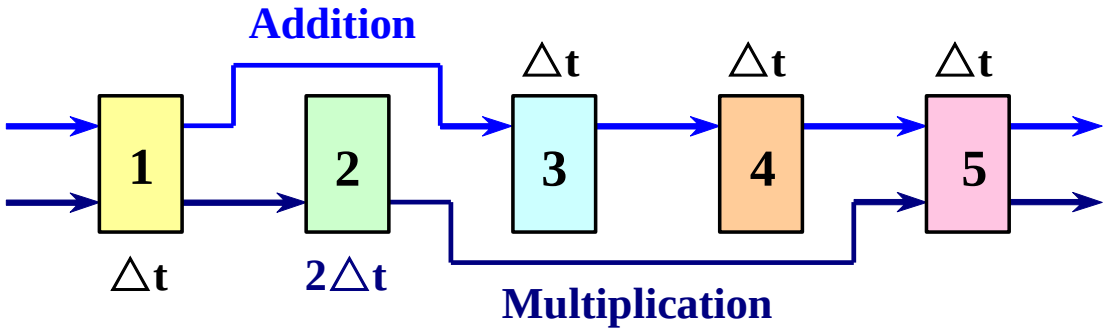
$$TP=7/15\Delta t$$

$$Sp=(4\times 3\Delta t+3\times 4\Delta t)/(15\Delta t)=1.6$$

$$\eta=(3\times 4\Delta t+4\times 3\Delta t)/(5\times 15\Delta t)=32\%$$

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 pipelining.

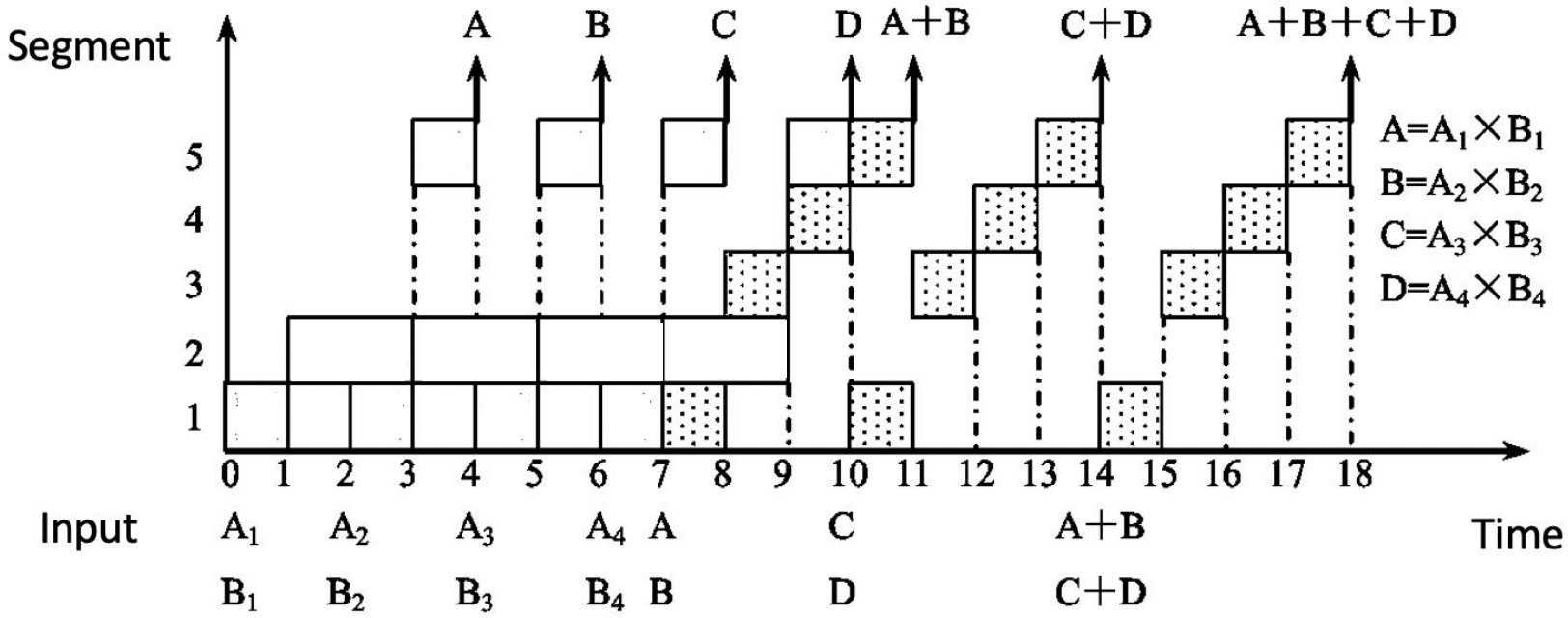


1→3→4→5 Addition pipelining

1→2→5 Multiplication pipelining

TP? Sp? η?

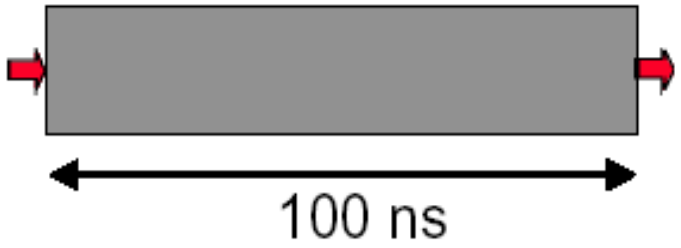
Pipeline Performance

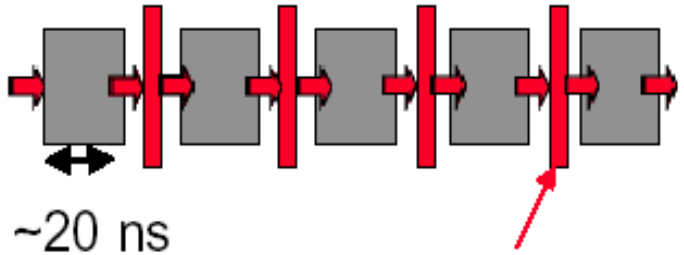


$$TP = \frac{7}{18\Delta t} \qquad S = \frac{28\Delta t}{18\Delta t} \approx 1.56 \qquad E = \frac{4\times 4 + 3\times 4}{5\times 18} \approx 0.31$$

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:

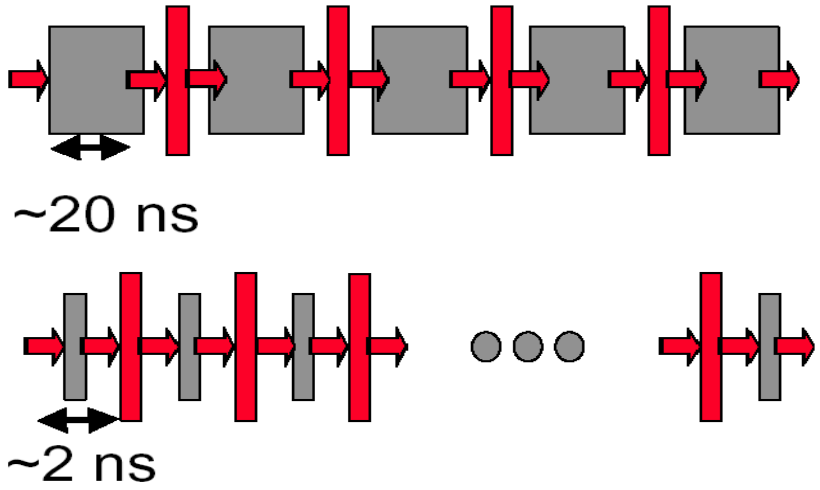
$$\frac{\textit{Time per instruction on unpipelined machine}}{\textit{Number of pipe stages}}$$

- So, ideal speedup equal to

Number of pipe stages.

Discussion

- Why not just make a 50-stage pipelining ?



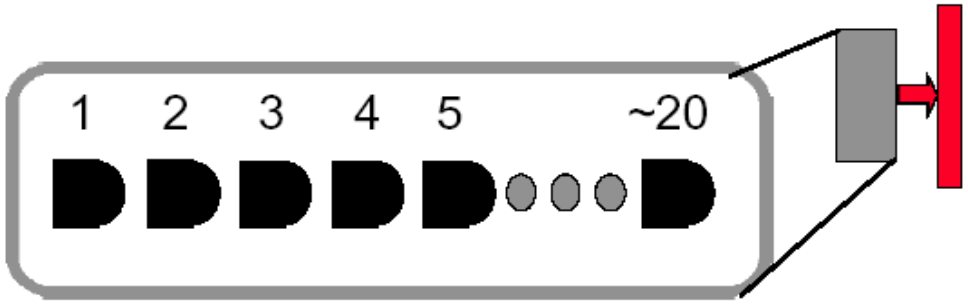
5 stages OK

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 implement 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

An Implementation of Pipelining

RISC-V Pipelining

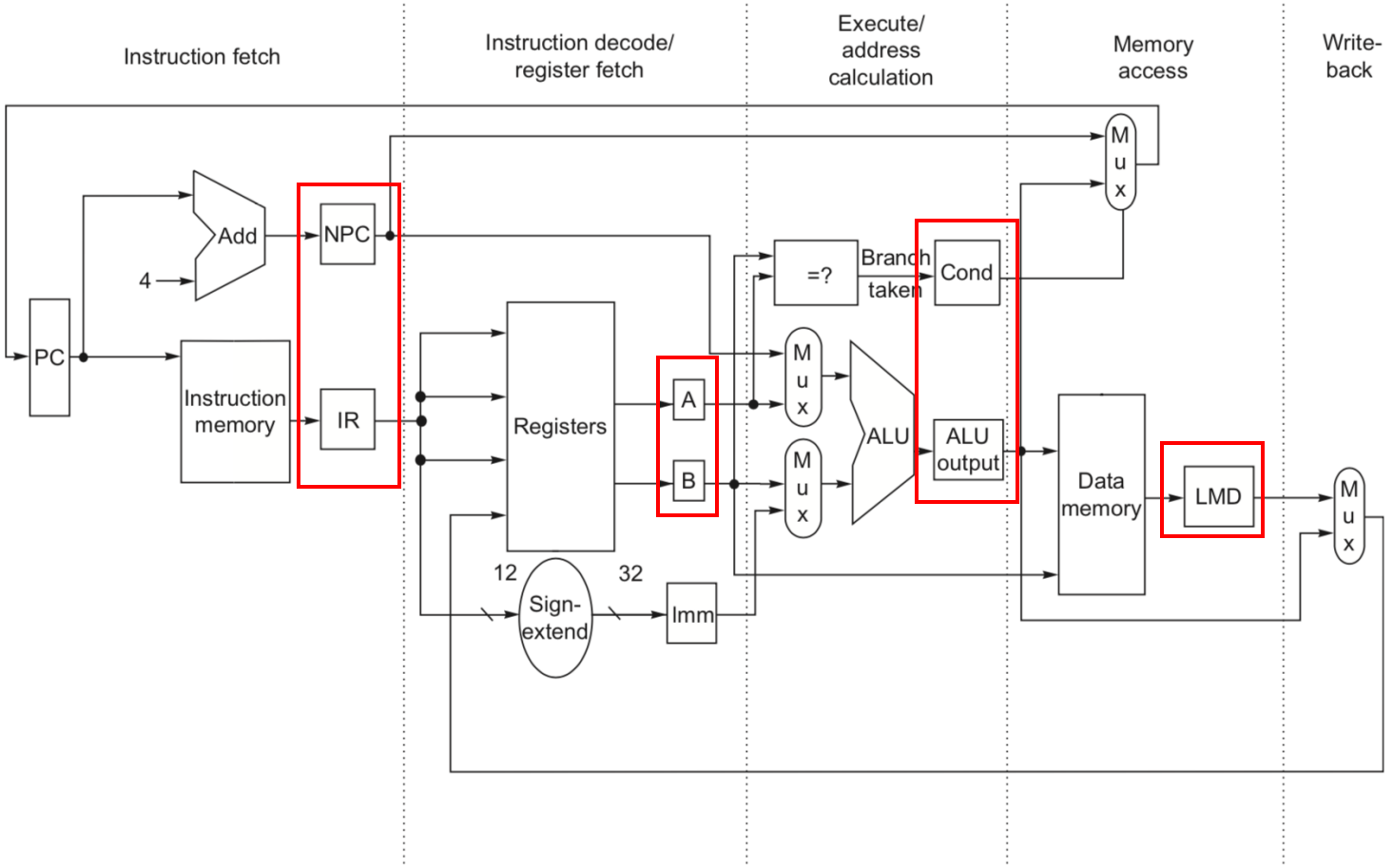
- Five stages, one step per stage

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

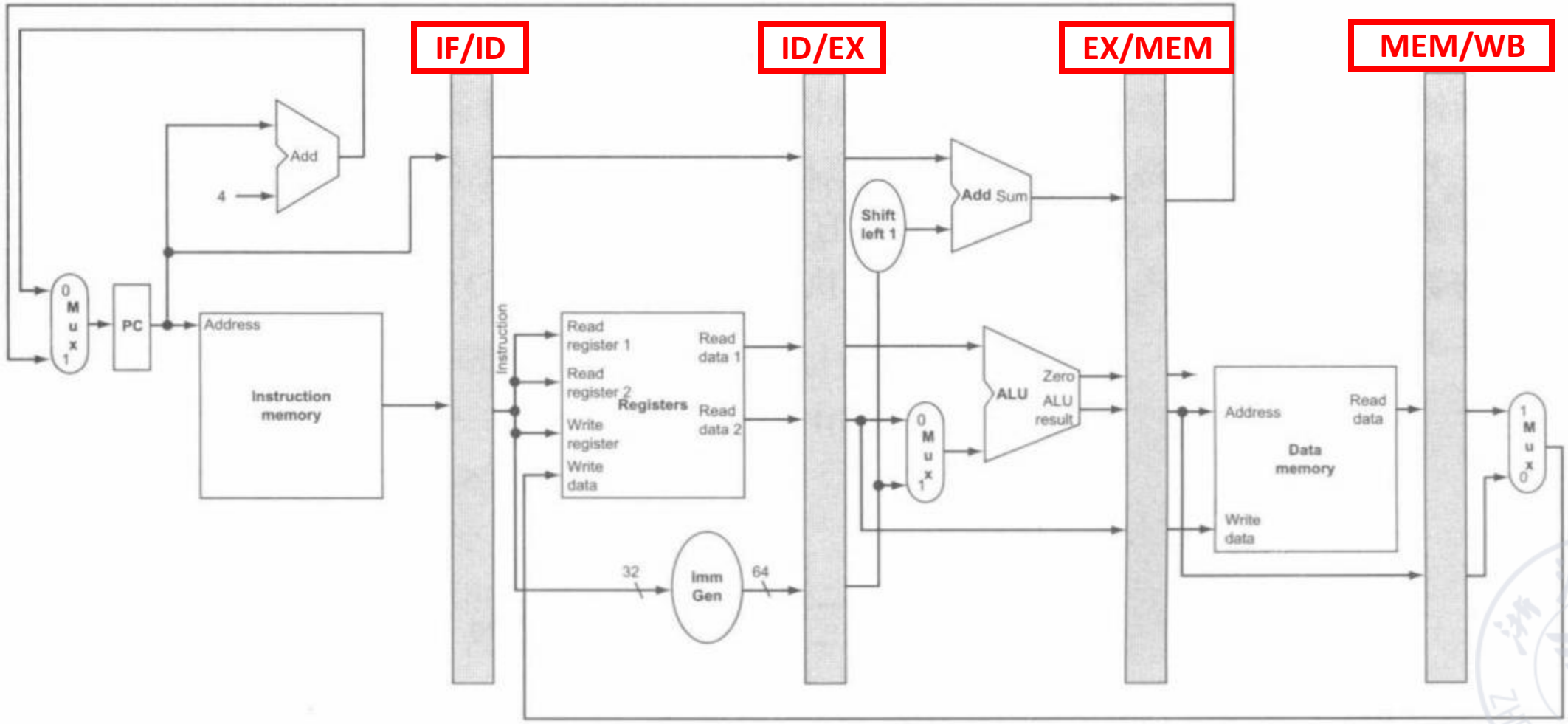
Pipelining and ISA Design

- 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
  - Alignment of memory operands
    - Memory access takes only one cycle

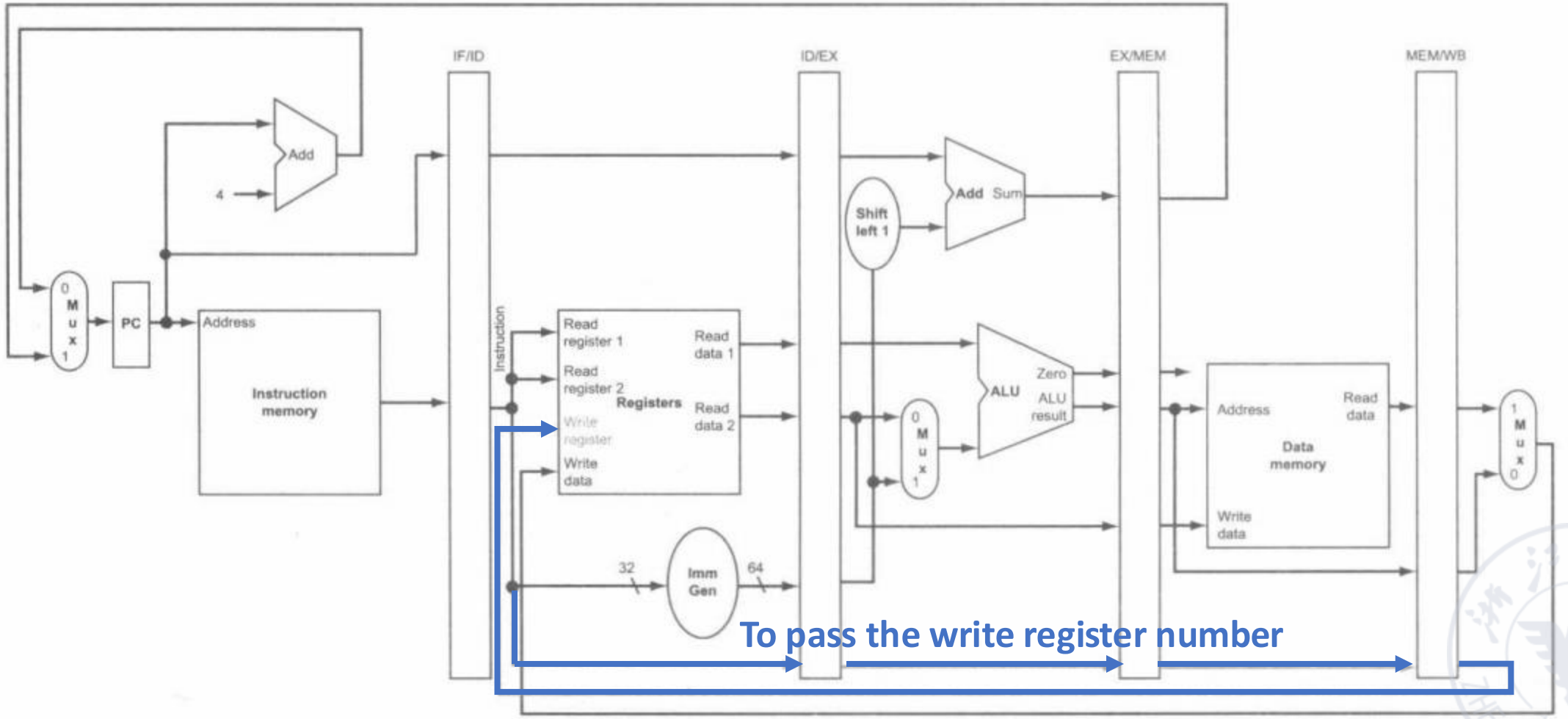
An Implementation of Pipelining



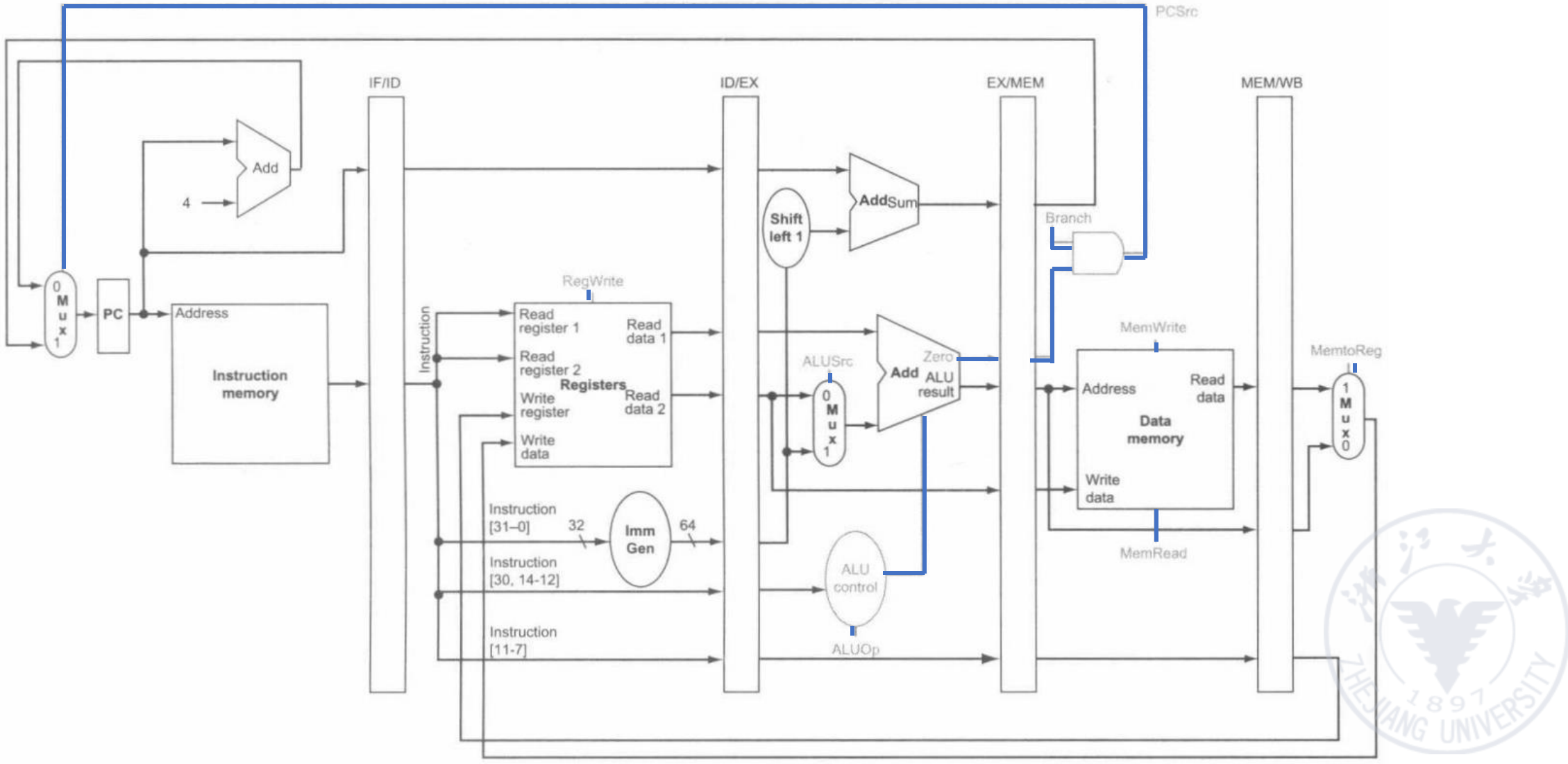
An Implementation of Pipelining



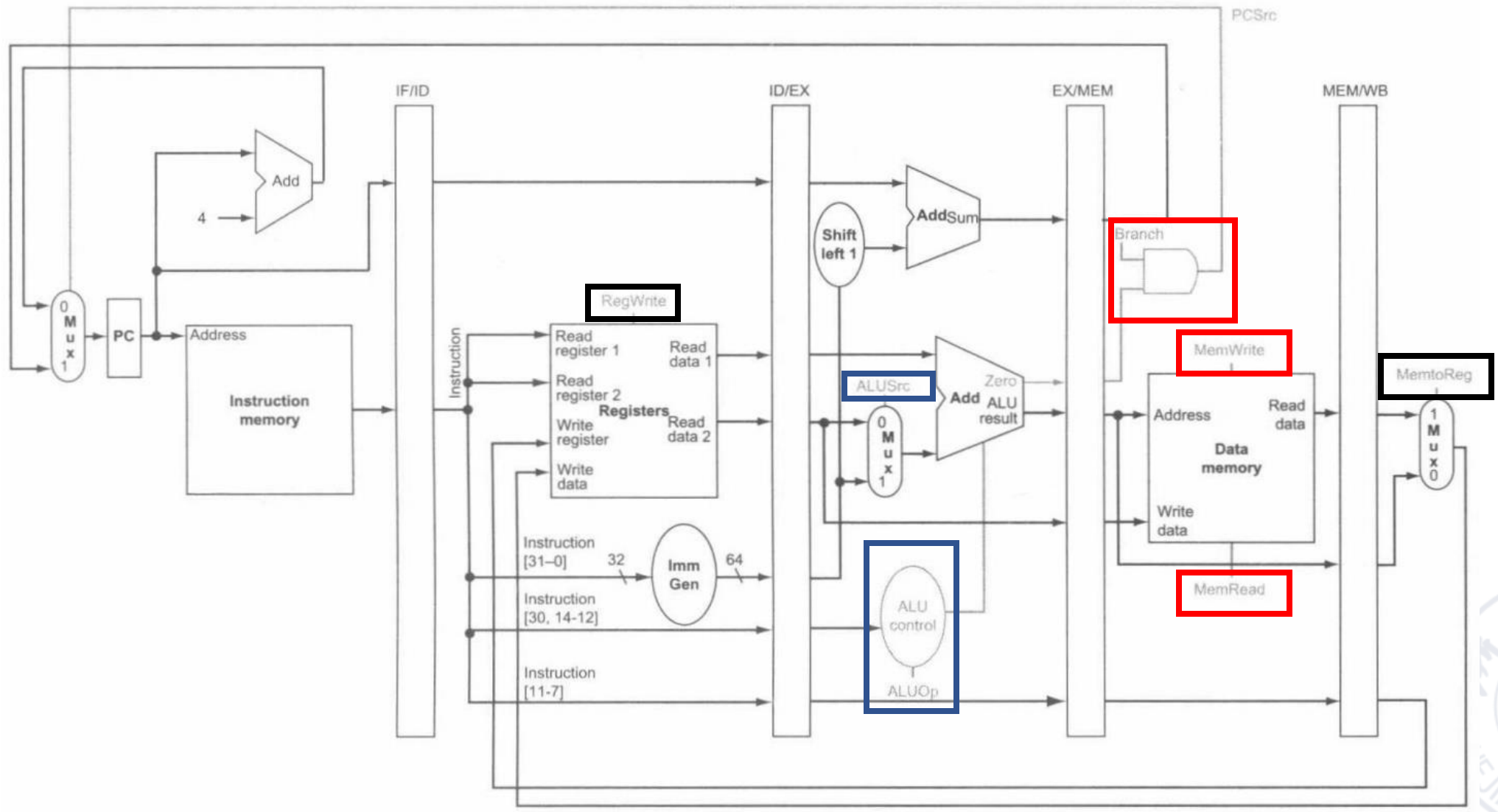
The Corrected Pipelined Datapath to Handle the Load Instruction Properly



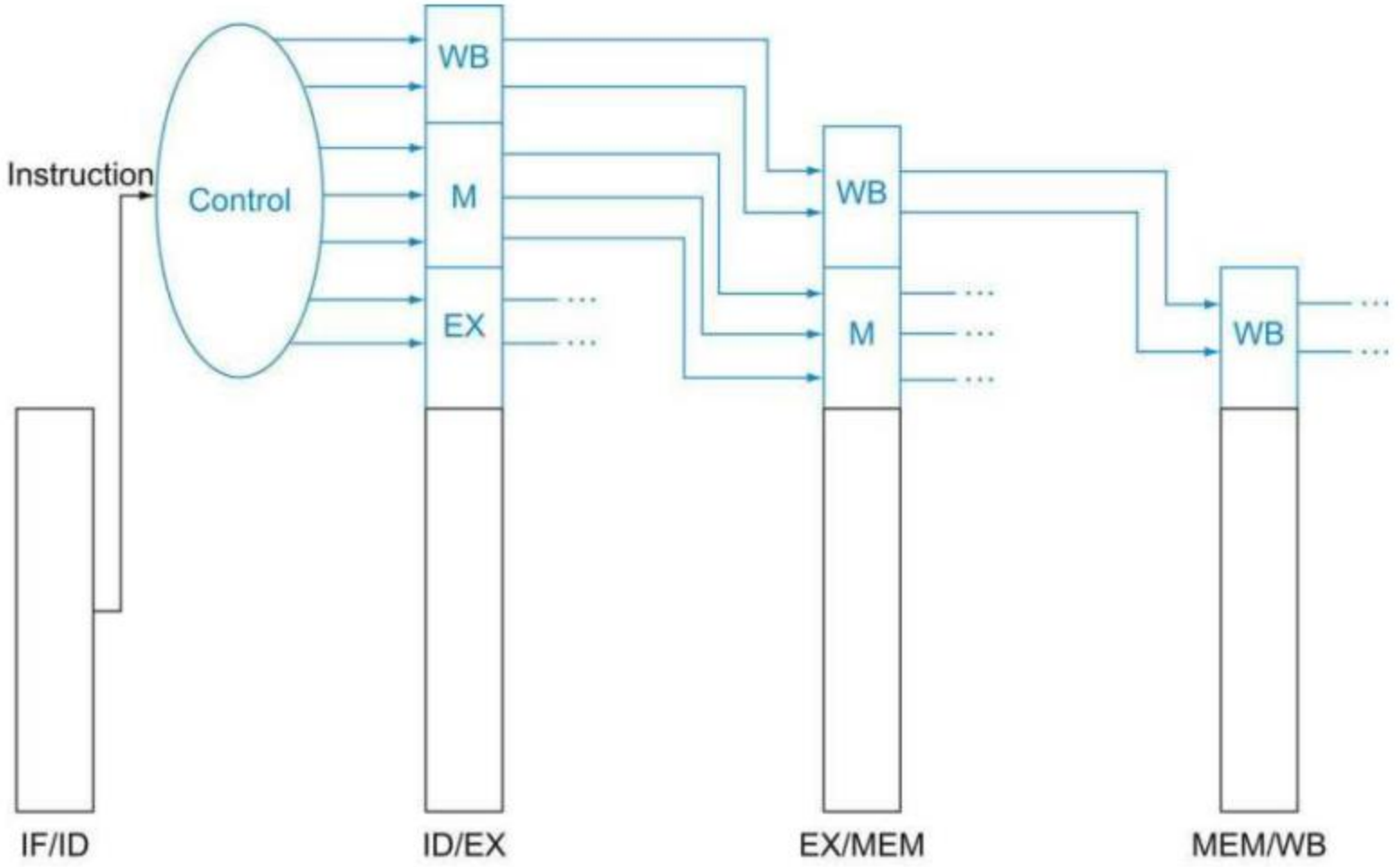
Pipelined Datapath with Control Signals



Review of 7 Control Lines



Extend Pipeline Registers to Include Control Information



Pipelined Datapath with the Control Signals