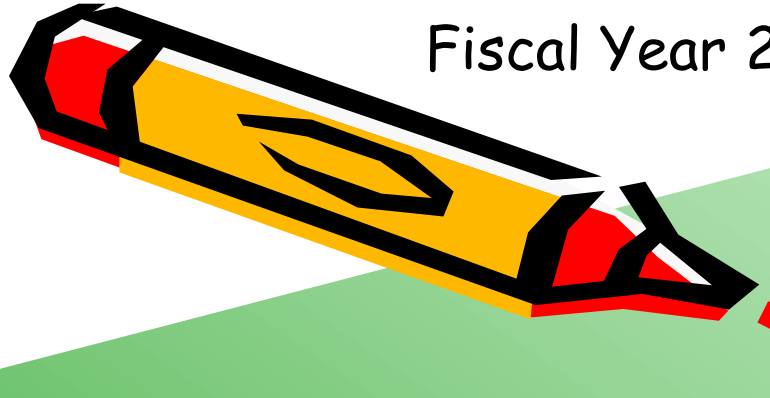
Fiscal Year 2020

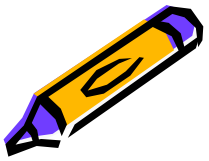
Ver. 2021-01-21a



Course number: CSC.T433
School of Computing,
Graduate major in Computer Science

Advanced Computer Architecture

9. Instruction Level Parallelism: Out-of-order Execution and Multithreading



www.arch.cs.titech.ac.jp/lecture/ACA/
Room No.W936
Mon 14:20-16:00, Thr 14:20-16:00

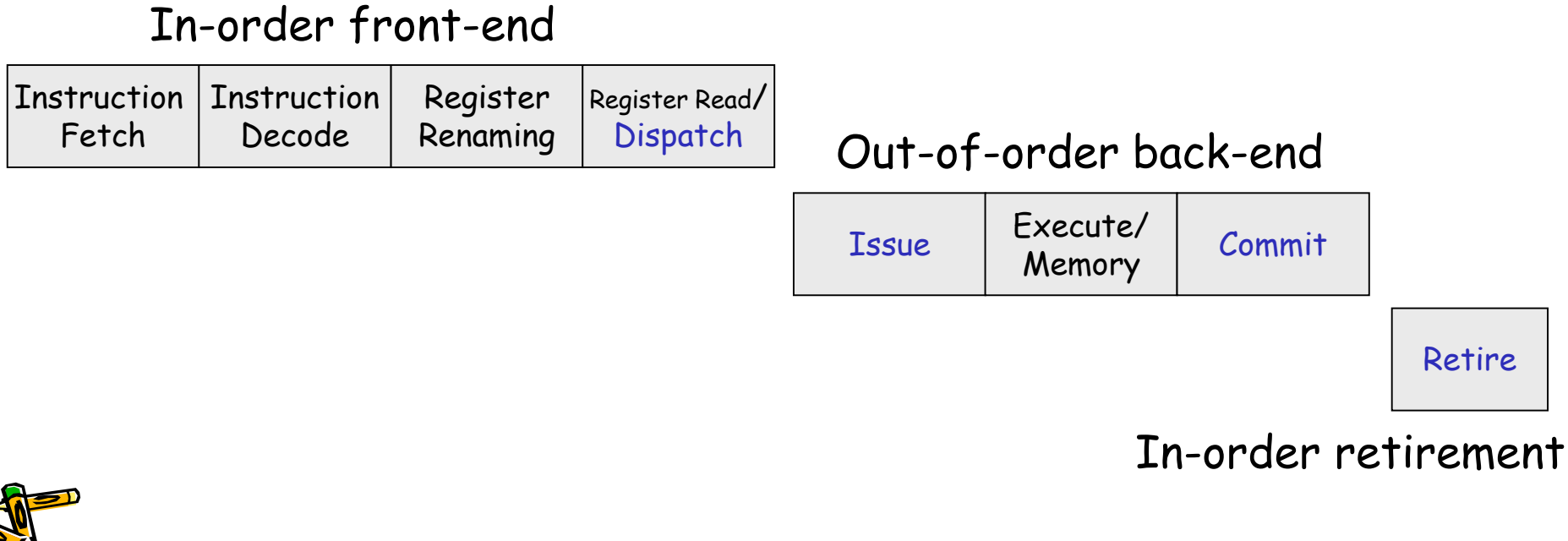
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

Exploiting Instruction Level parallelism (ILP)

- A superscalar processor has to handle some flows efficiently to exploit ILP
  - Control flow
    - To execute *n* instructions per clock cycle, the processor has to fetch at least *n* instructions per cycle.
    - The main obstacles are branch instruction (BNE, BEQ)
    - Another obstacle is instruction cache
  - Register data flow
    - Dynamic scheduling
  - Memory data flow

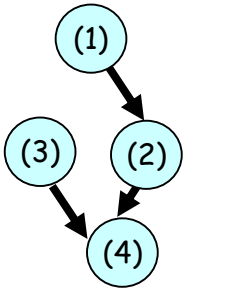
Instruction pipeline of OoO execution processor

- Allocating instructions to instruction window is called dispatch
- Issue or fire wakes up instructions and their executions begin
- In commit stage, the computed values are written back to ROB
- The last stage is called retire or graduate. The result is written back to register file (architectural register file) using a logical register number.

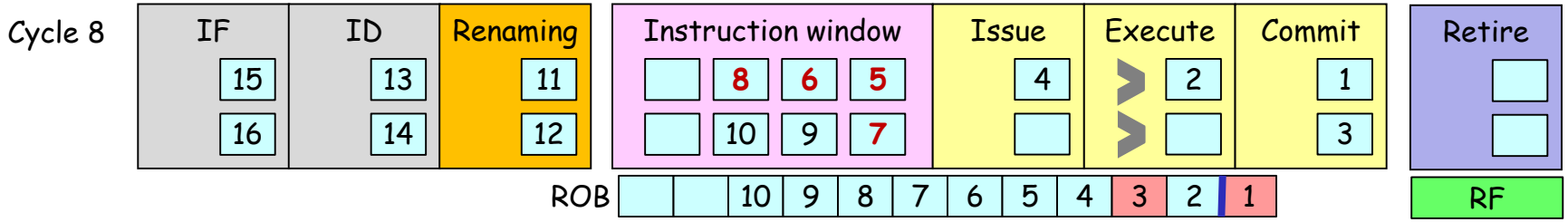


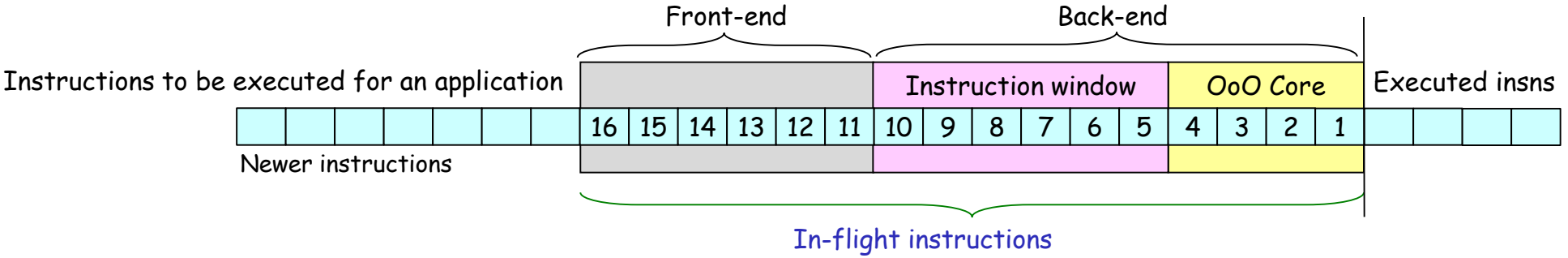
Register dataflow

- In-flight instructions are ones processing in a processor



Data flow graph

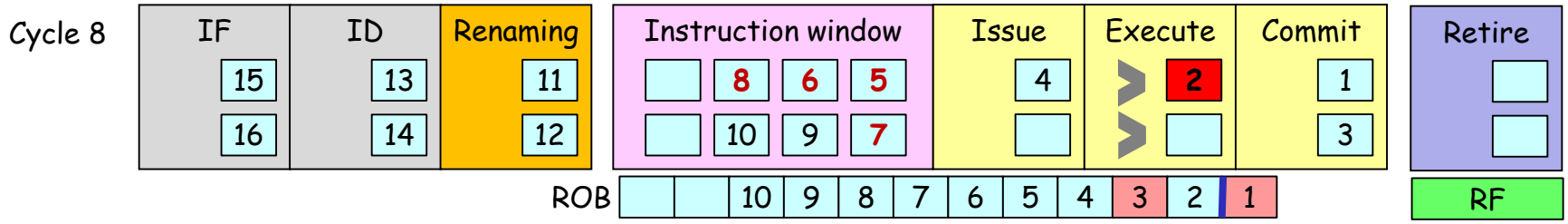


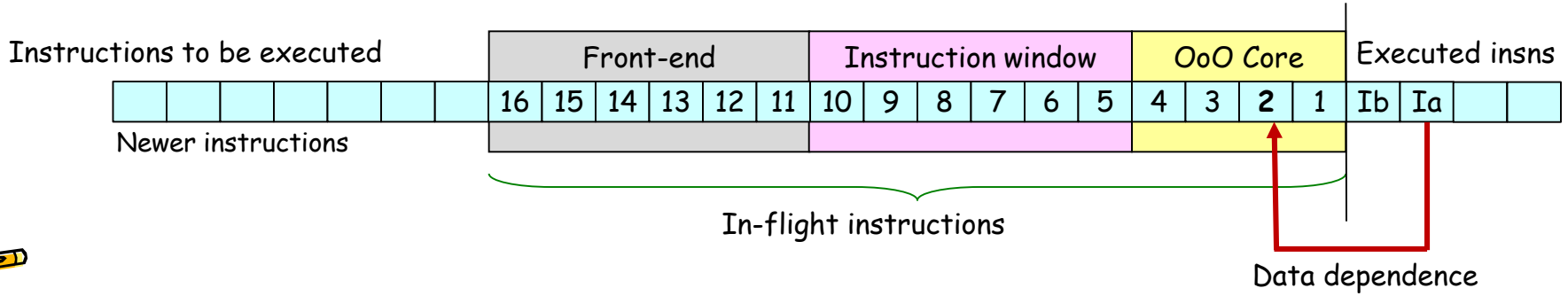


Case 1: Register dataflow from a far previous instn

- One source operand of insn I2 is from a retired instruction Ia.
- Because Ia is retired long ago, the physical destination register has been freed. The tag of the source register can not be renamed at the renaming stage for I2, still having the logical register tag $3.
- Where does the operand $3 of I2 comes from?

```
Ia: add $3,$0,$0
I1: sub p9,$1,$2
I2: add p10,p9,$3
I3: or  p11,$4,$5
I4: and p12,p10,p11
```

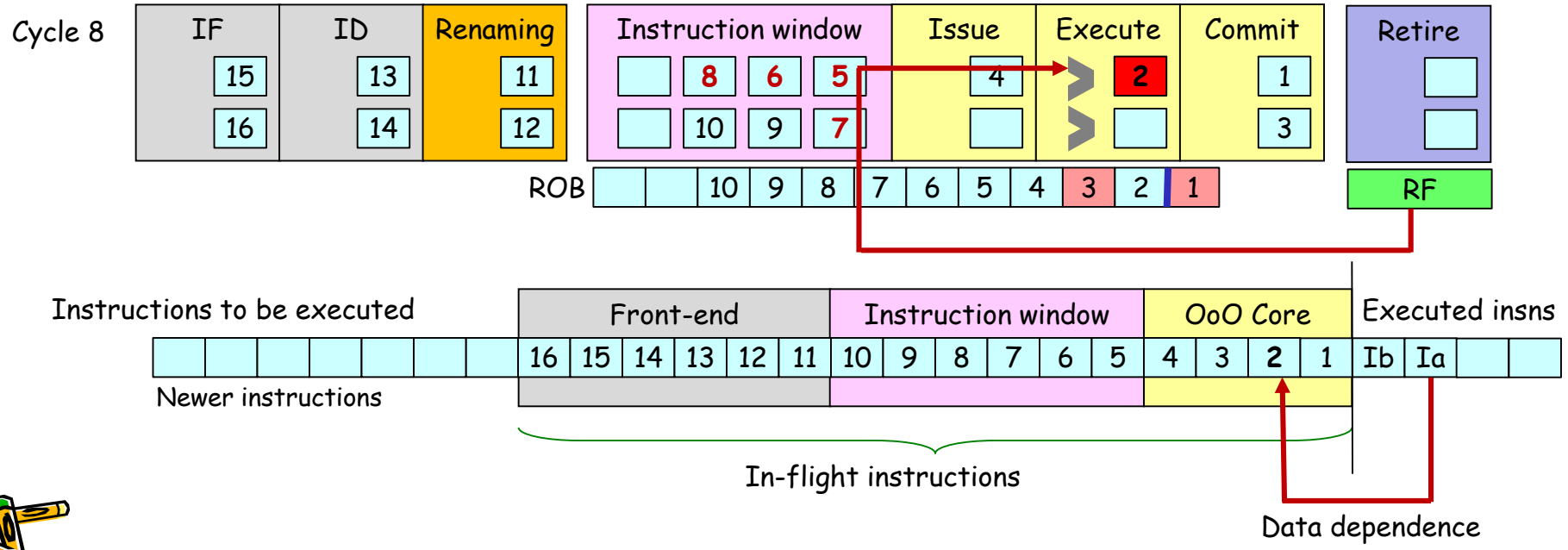




Case 1: Register dataflow from a far previous instn

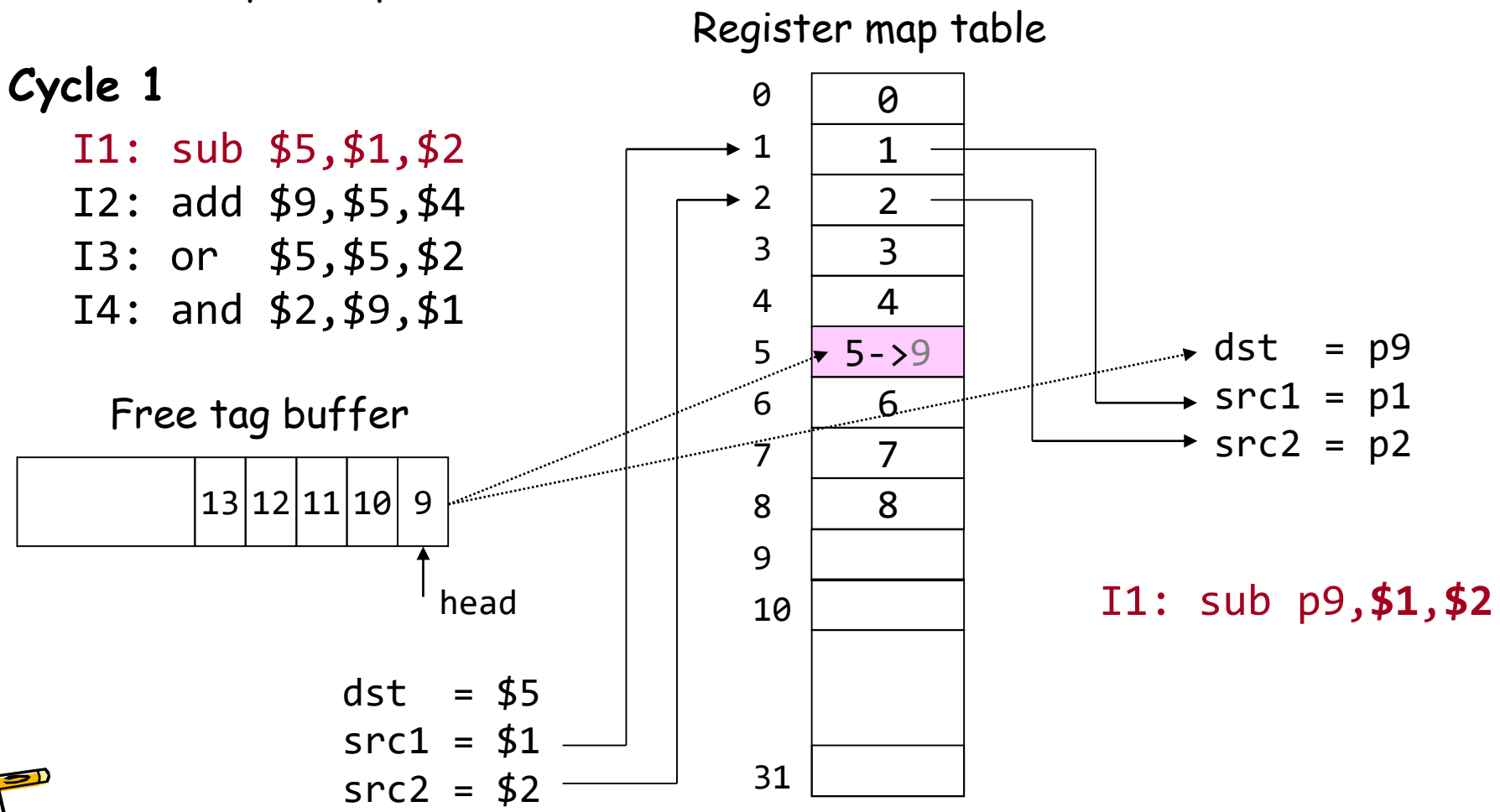
- One source operand of insn I2 is from a retired instruction Ia.
- Because Ia is retired long ago, the physical destination register has been freed. The tag of the source register can not be renamed at the renaming stage for I2, still having the logical register tag $3.
- Where does the operand $3 of I2 comes from?

```
Ia: add $3,$0,$0
I1: sub p9,$1,$2
I2: add p10,p9,$3
I3: or  p11,$4,$5
I4: and p12,p10,p11
```



Register renaming again

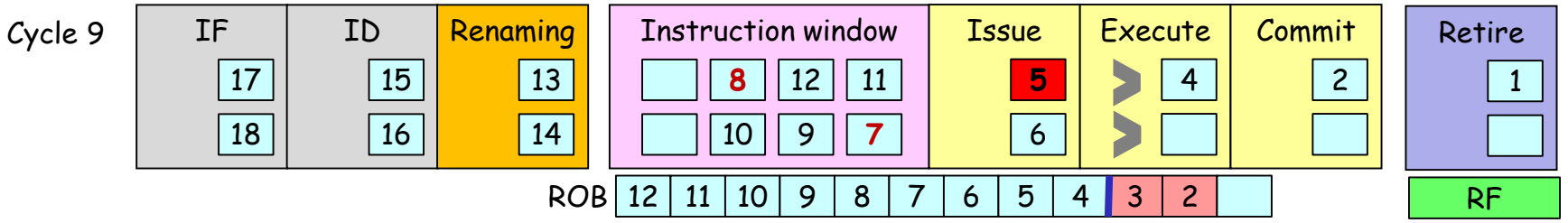
- A processor remembers a set of renamed logical registers.
- If $1 and $2 are not renamed for in-flight instructions, it uses $1 and $2 instead of p1 and p2.

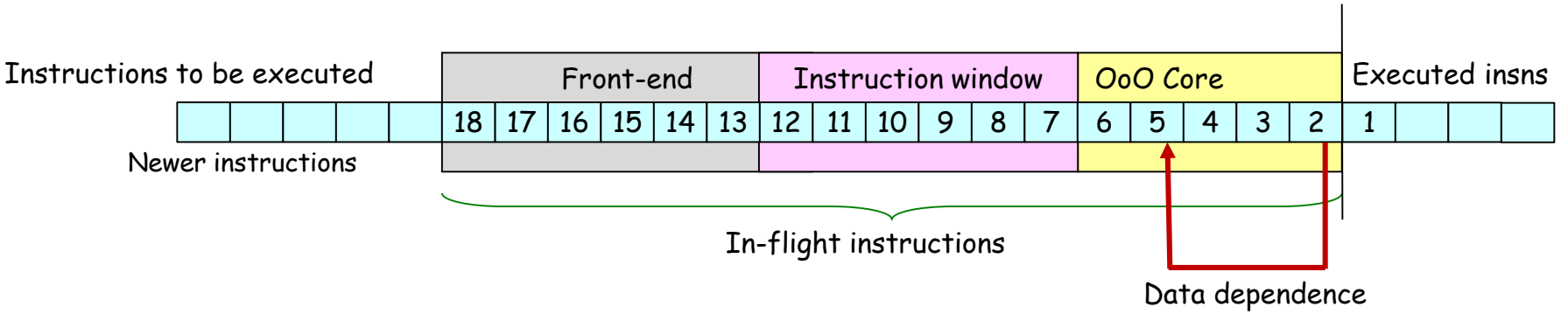


Case 2: Register dataflow

- Assume that one source operand p10 of insn I5 is from I2 which is not retired. The operand is generated a few clock cycles (tens of cycles sometimes) earlier.
- Because I2 is not retired, RF does not have the operand. Because I2 is committed, the operand is stored in ROB.
- Where does the operand of I5 comes from?

```
Ia: add $3,$0,$0
I1: sub p9,$1,$2
I2: add p10,p9,$3
I3: or  p11,$4,$5
I4: and p12,p10,p11
I5: nor p13,p10,p12
```

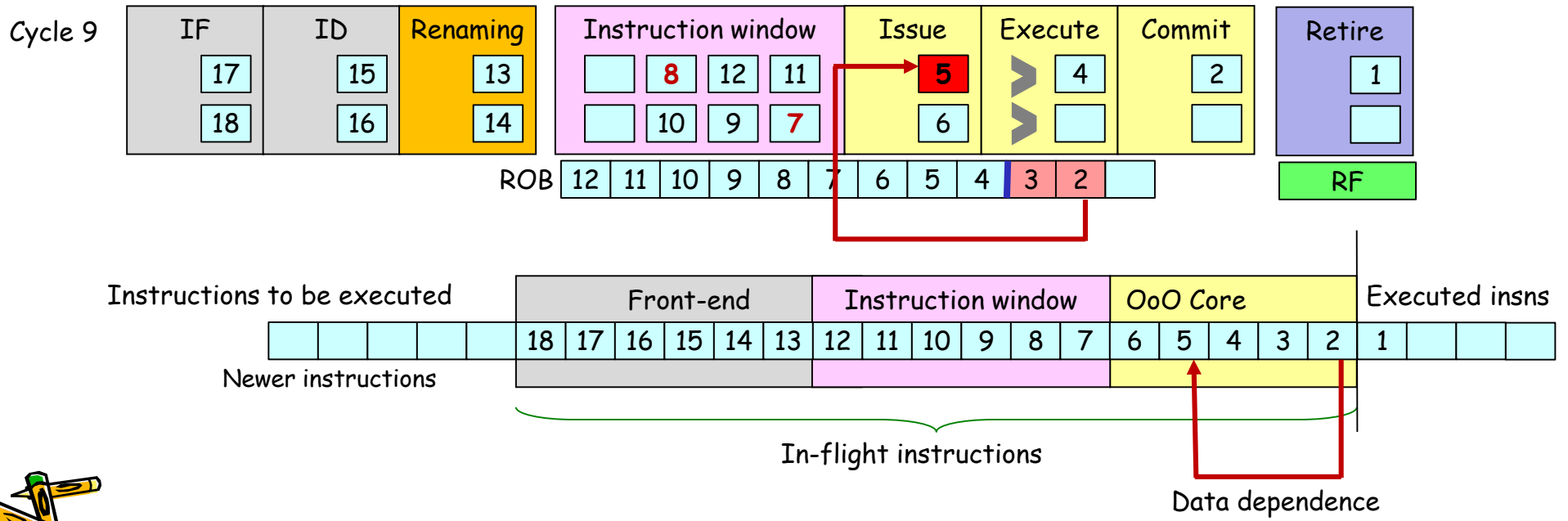




Case 2: Register dataflow from ROB

- Assume that one source operand p10 of insn I5 is from I2 which is not retired. The operand is generated a few clock cycles (tens of cycles sometimes) earlier.
- Because I2 is not retired, RF does not have the operand. Because I2 is committed, the operand is stored in ROB.
- Where does the operand of I5 comes from?

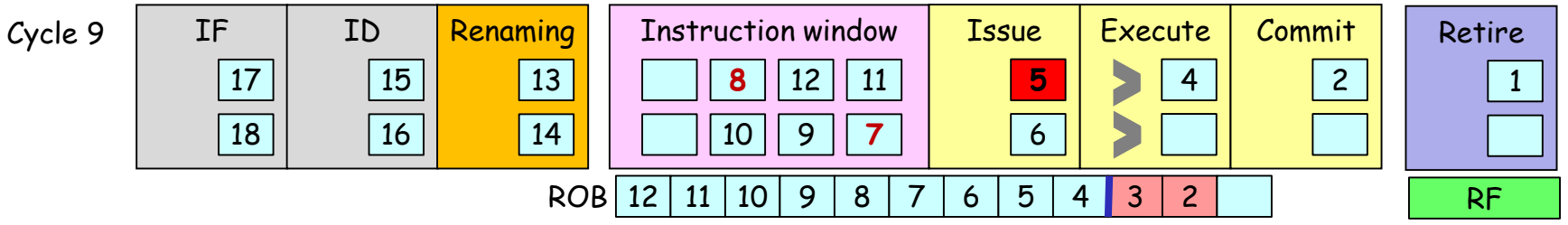
```
Ia: add $3,$0,$0
I1: sub p9,$1,$2
I2: add p10,p9,$3
I3: or  p11,$4,$5
I4: and p12,p10,p11
I5: nor p13,p10,p12
```

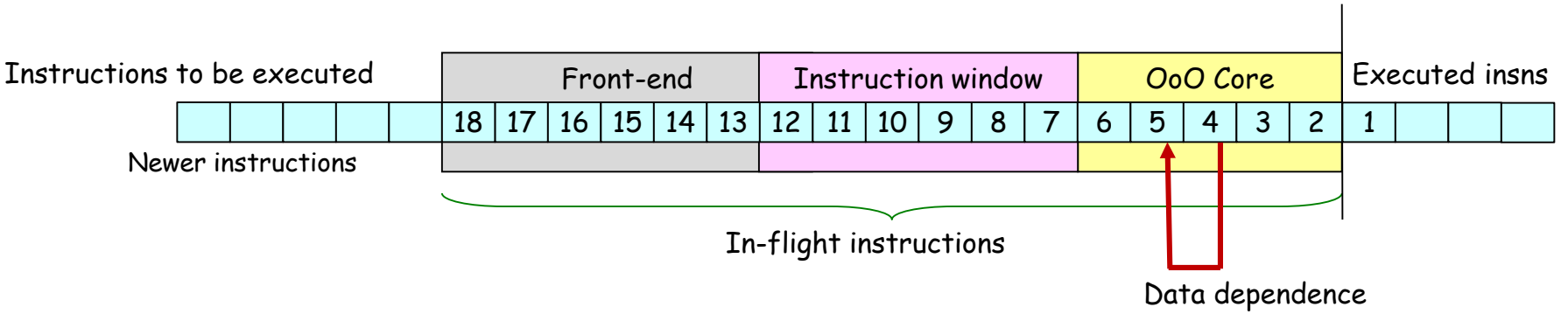


Case 3: Register dataflow

- Assume that the other source operand p12 of insn I5 is from I4 which is not committed. The operand is generated in the previous clock cycle.
- Because I2 is not retired, RF does not have the operand.
  Because I2 is not committed, ROB does not have the operand.
- Where does the operand of I5 comes from?

```
Ia: add $3,$0,$0
I1: sub p9,$1,$2
I2: add p10,p9,$3
I3: or  p11,$4,$5
I4: and p12,p10,p11
I5: nor p13,p10,p12
```

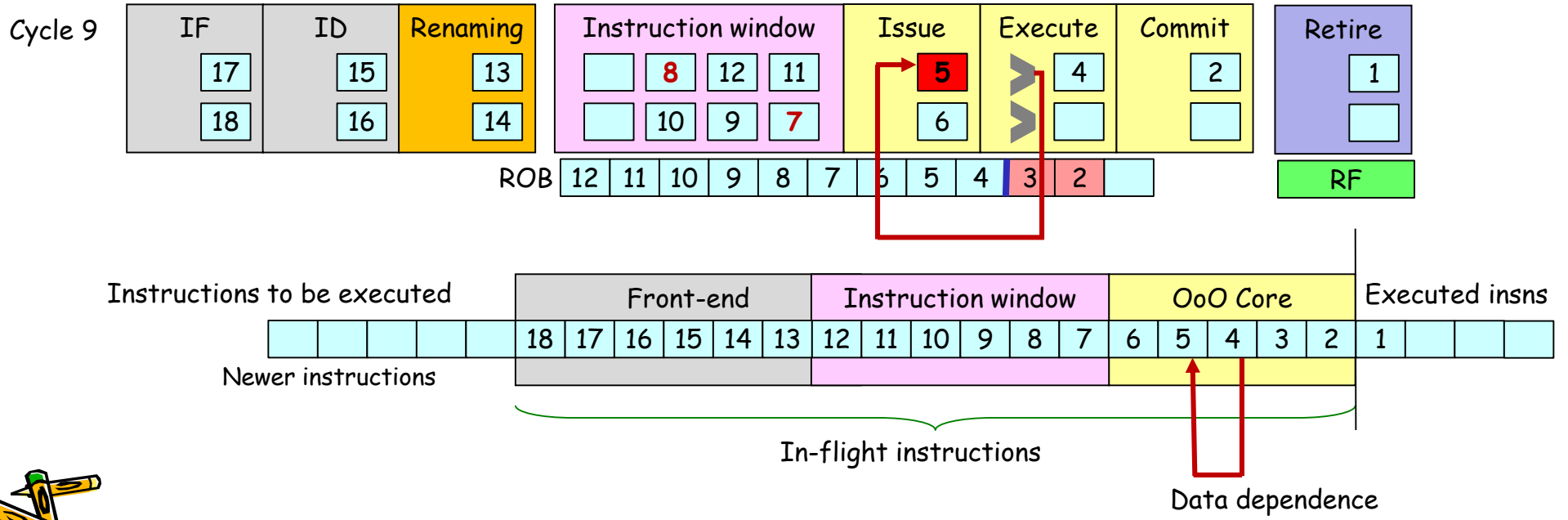




Case 3: Register dataflow from ALUs

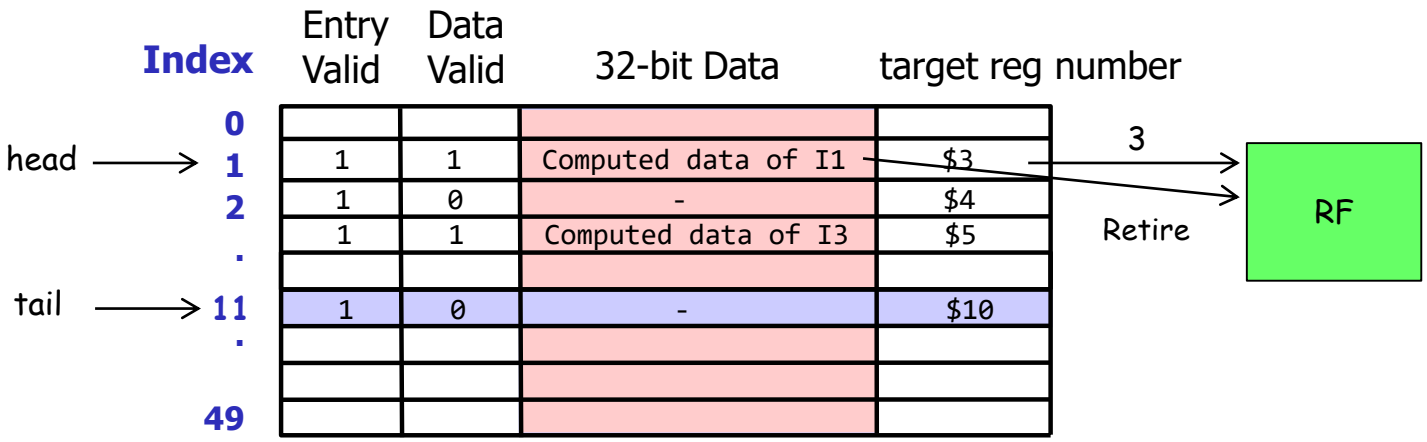
- Assume that the other source operand p12 of insn I5 is from I4 which is not committed. The operand is generated in the previous clock cycle.
- Because I2 is not retired, RF does not have the operand.
  Because I2 is not committed, ROB does not have the operand.
- Where does the operand of I5 comes from?

```
Ia: add $3,$0,$0
I1: sub p9,$1,$2
I2: add p10,p9,$3
I3: or  p11,$4,$5
I4: and p12,p10,p11
I5: nor p13,p10,p12
```

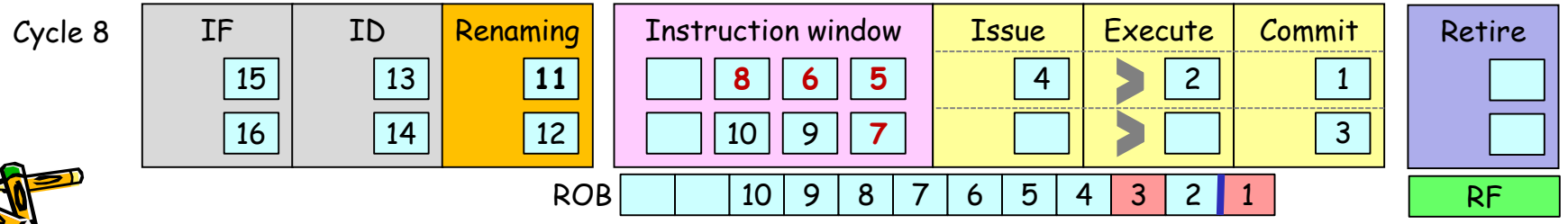


Reorder buffer (ROB), content addressable memory

- Each ROB entry has following fields
  - entry valid bit, data valid bit, data, target register number, etc.
- ROB provides the large physical registers for renaming
  - physical register number is ROB entry number
- The value of a physical register may be data within a matching ROB entry

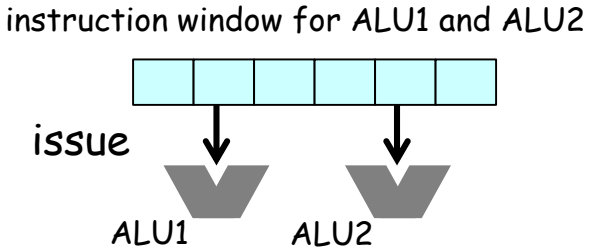



```
I11: add p11,p3,p8 (add $10,$5,$6)
```

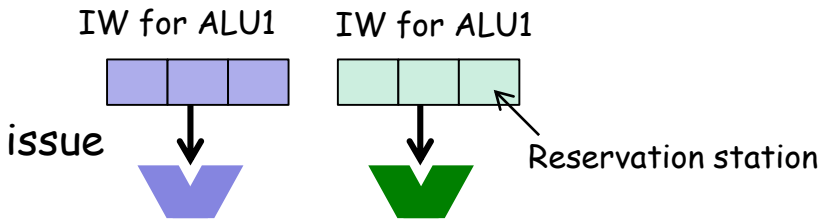


Reservation station (RS)

- To simplify the wakeup and select logic at issue stage, each functional unit (ALU) has own instruction window, an entry for an instruction is called reservation station (RS).
- Each reservation station has
  - valid bit, src1 tag, src1 data, src1 ready, src2 tag, src2 data, src2 ready, destination physical register number (*dst*), operation, …
  - The computed data with its *dst* as tag is broadcasted to all RSs.

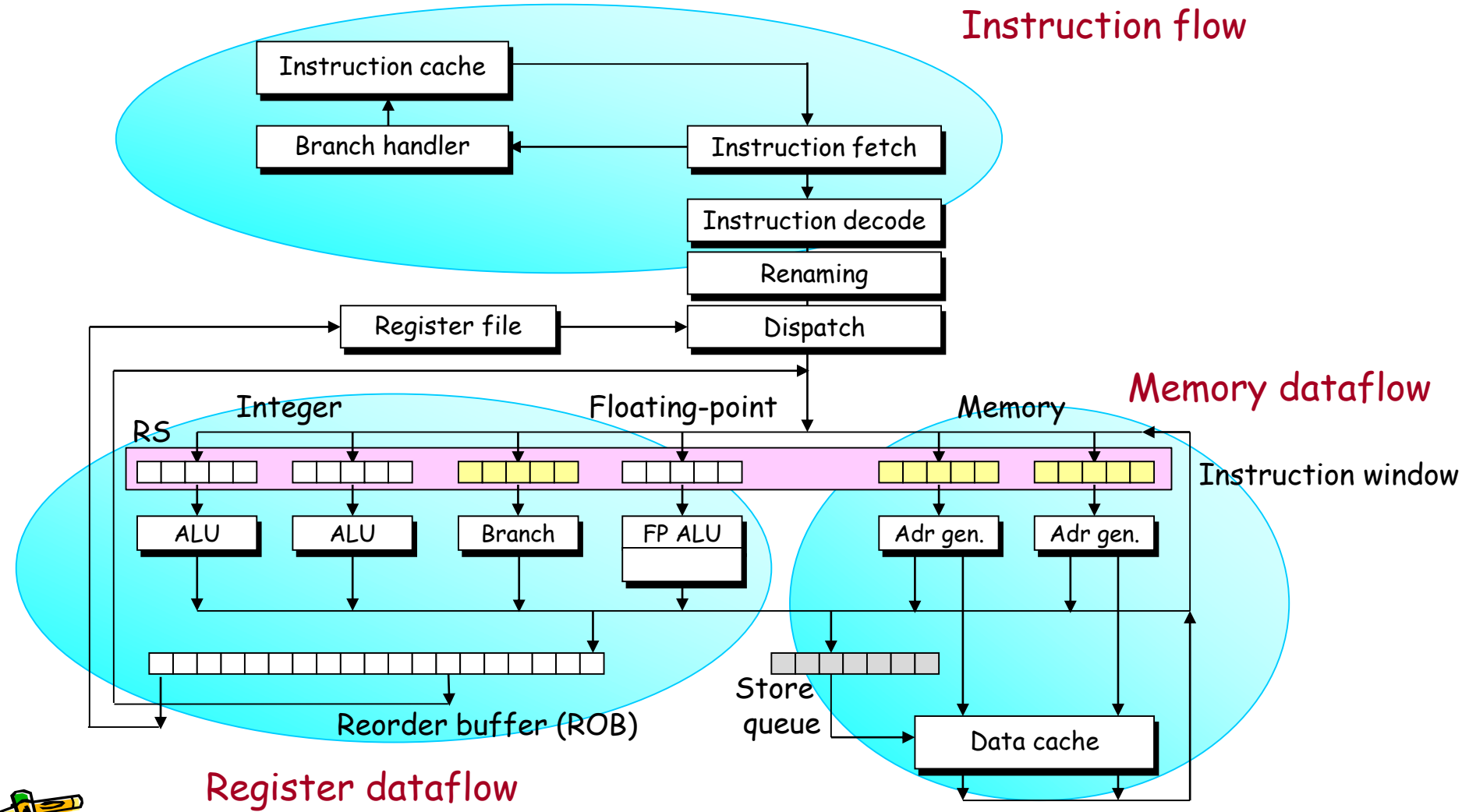


(a) Central instruction window



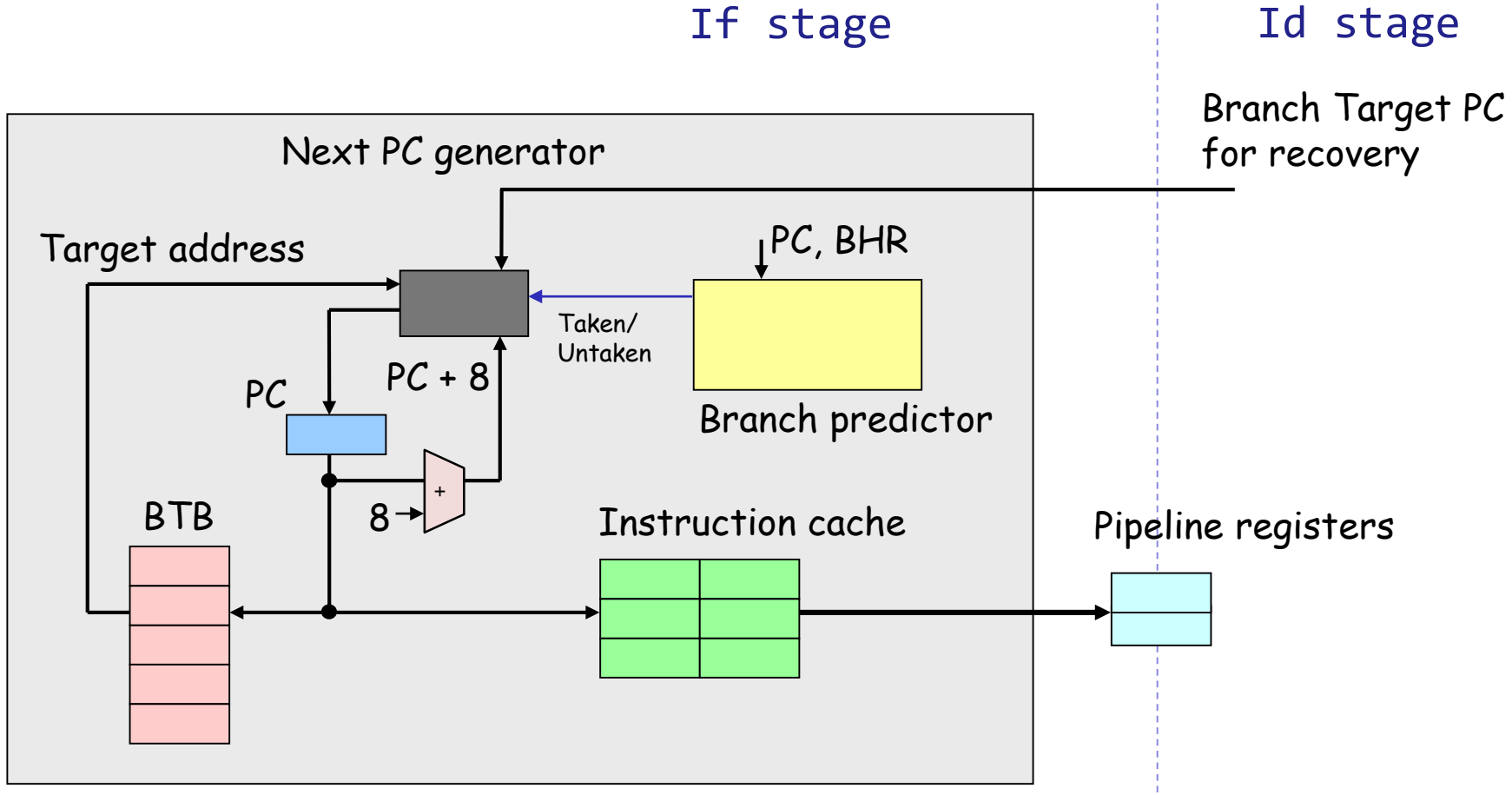
(b) instruction window using RS

Datapath of OoO execution processor



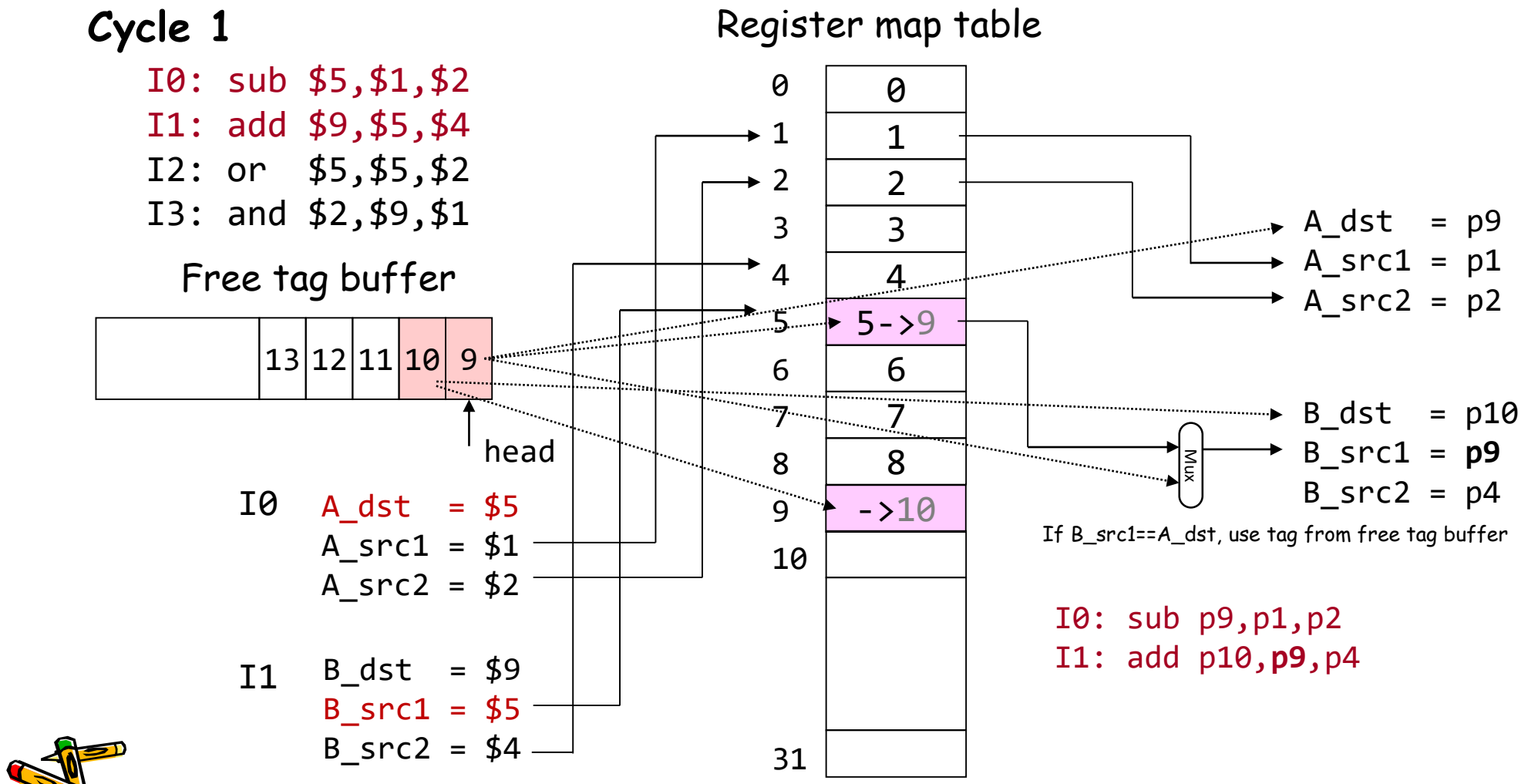
Instruction fetch unit in IF stage

- For high-bandwidth instruction delivery, prediction, and speculation

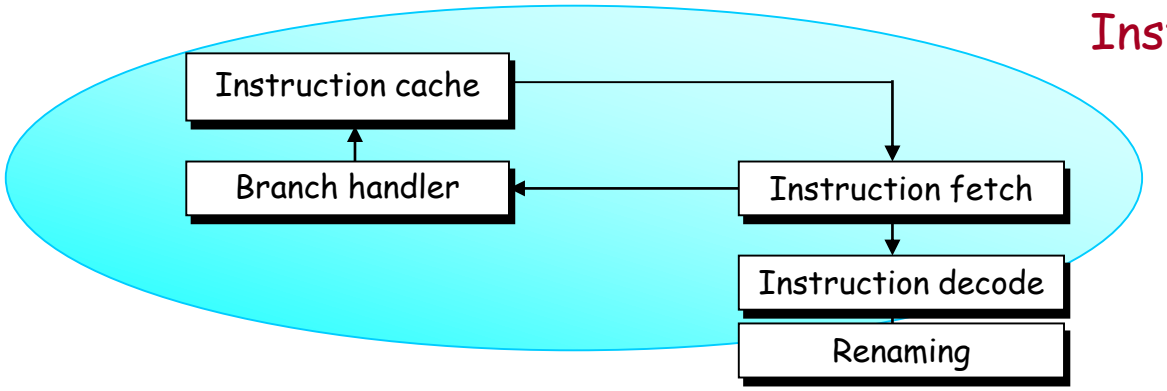


Renaming two instructions per cycle for superscalar

- Renaming instruction I0 and I1

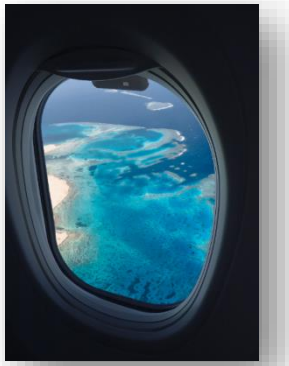


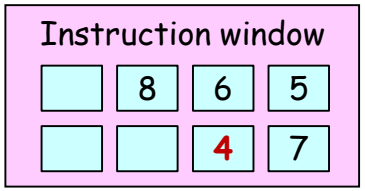
Datapath of OoO execution processor (partially)

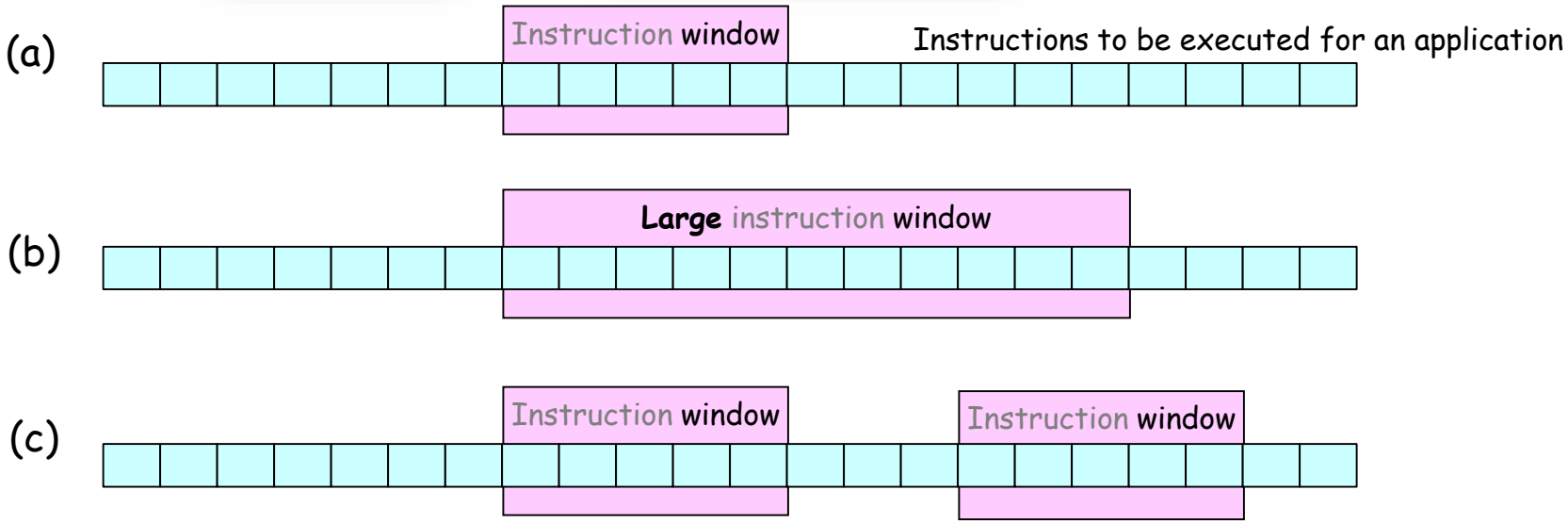


Aside: What is a window?

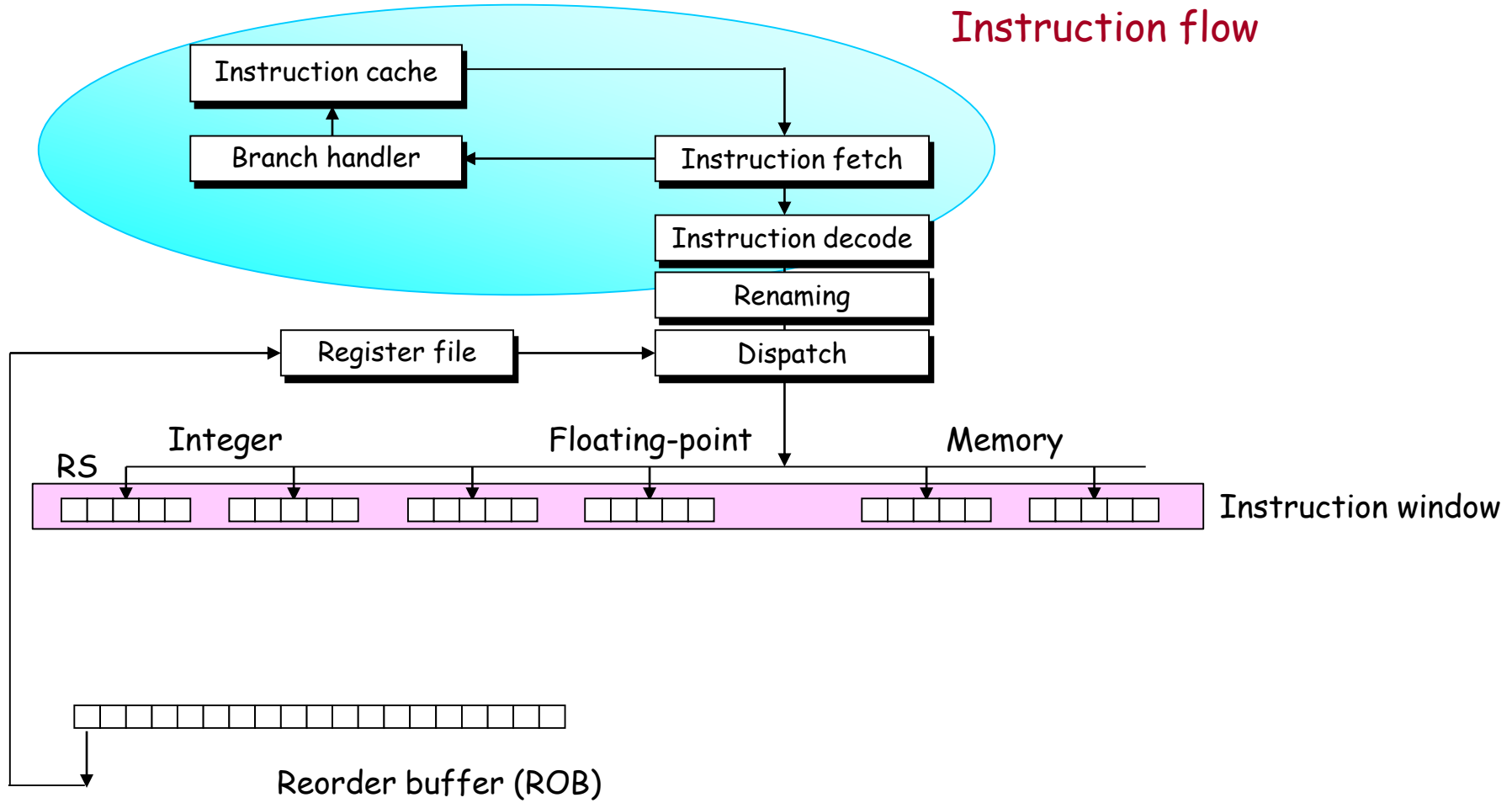
- A window is a space in the wall of a building or in the side of a vehicle, which has glass in it so that light can come in and you can see out. (Collins)



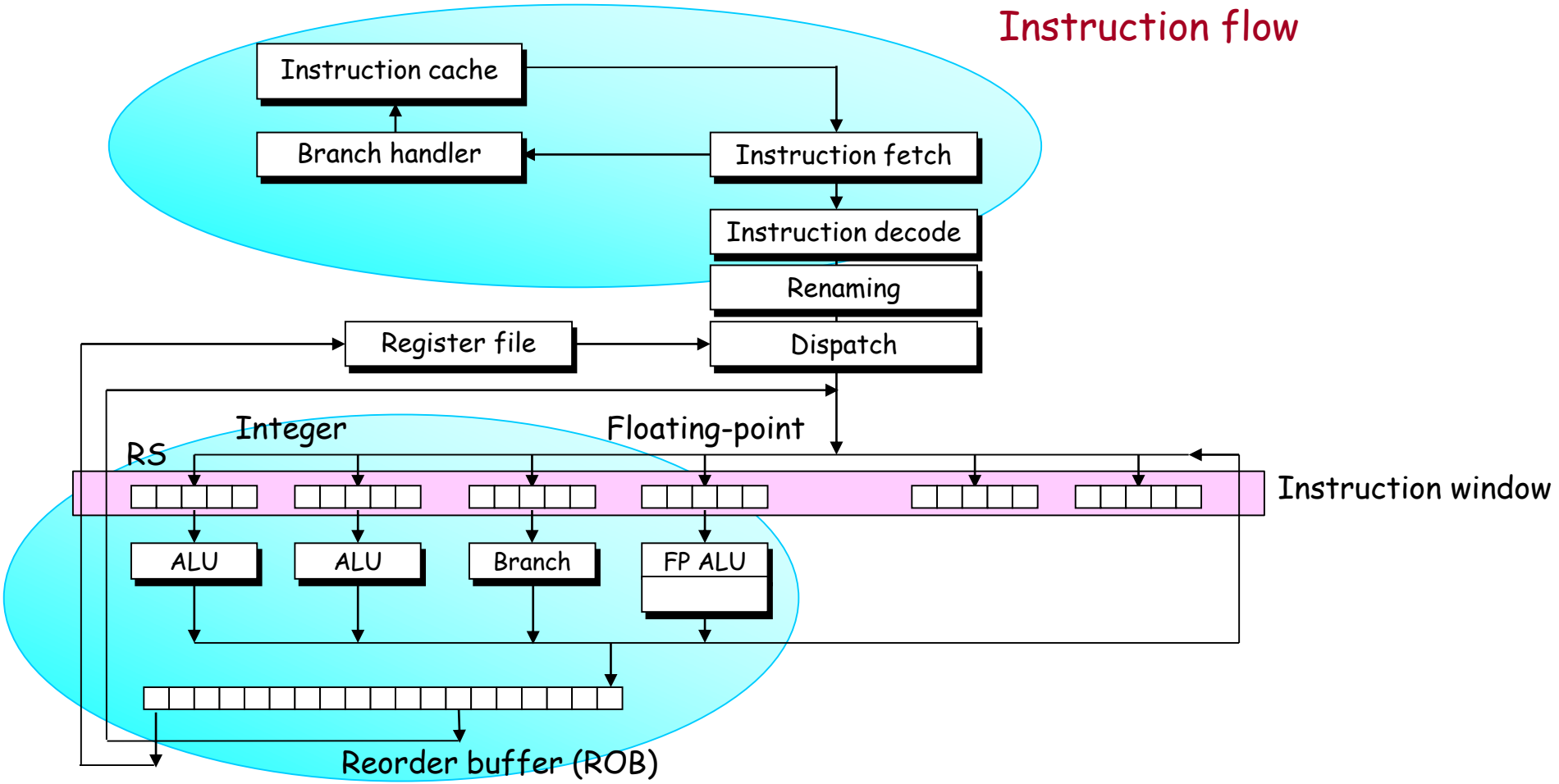




Datapath of OoO execution processor (partially)

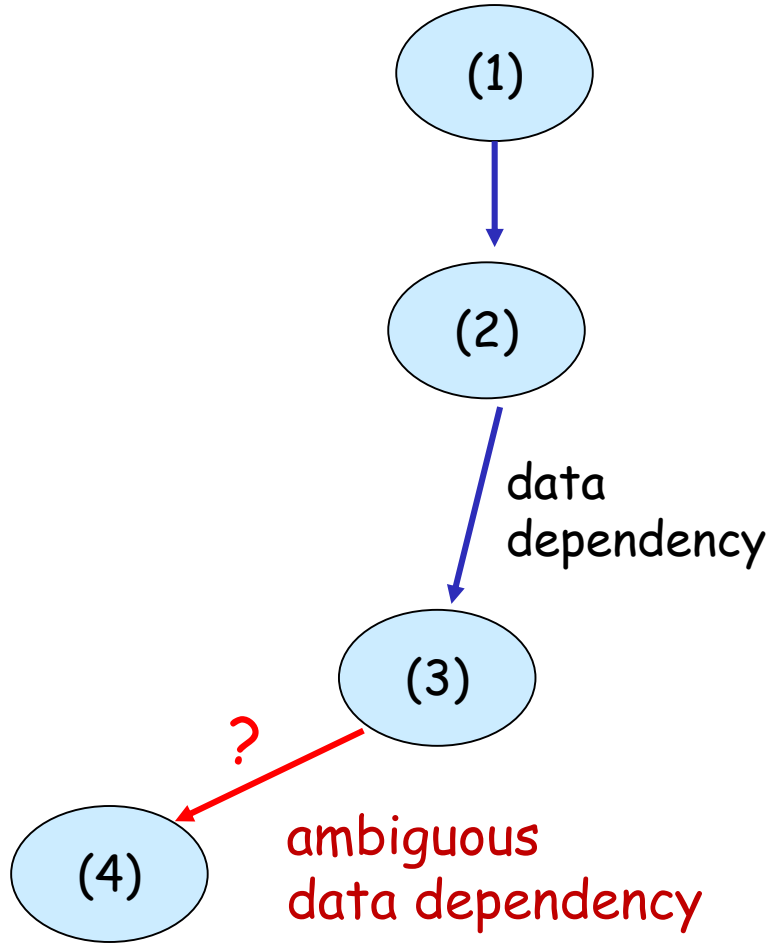


Datapath of OoO execution processor (partially)



Instruction Level Parallelism (ILP)

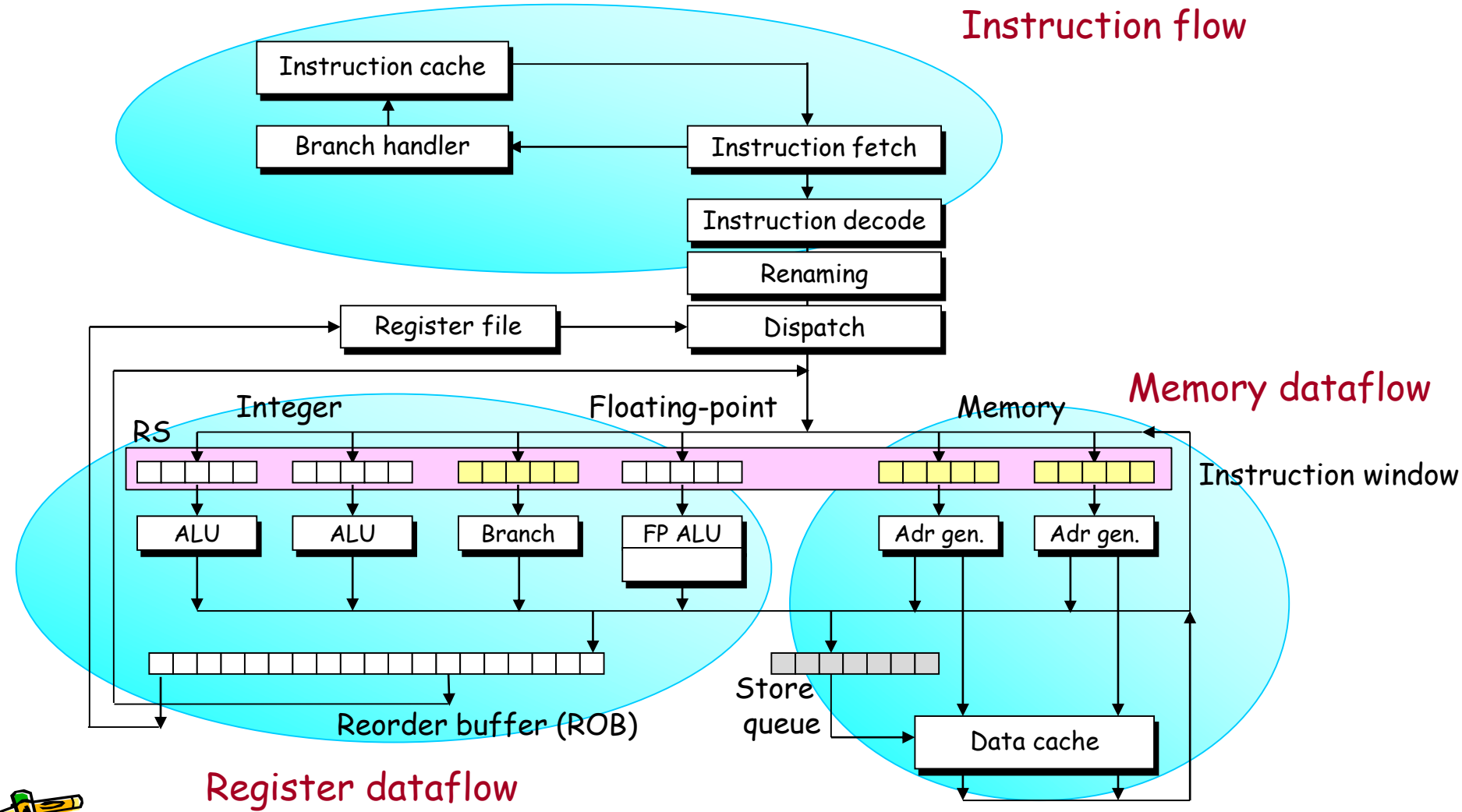
```
lw    $t0, 32($s3)      (1)
add   $t0, $s2, $t0     (2)
sw    $t0, 48($s3)      (3)
        ?
lw    $t1, 32($s4)      (4)
```



Memory dataflow and branches

- The update of a data cache cannot be recovered easily. So, cache update is done at the retire stage in-order manner by using store queue.
  Because of the ambiguous memory dependency, load and store instructions can be executed in-order manner.
  - About 30% (or less) of executed instructions are load and stores.
  - Even if they are executed in-order, IPC of 3 can be achieved.
- Branch instructions are executed in-order manner.
  - About 20% (or less) of executed instructions are jump and branch instructions.
  - Out-or-order branch execution and aggressive miss recovery may cause false recovery (recovery by a branch on the false control path).

Datapath of OoO execution processor



Pollack's Rule

- Pollack's Rule states that microprocessor "performance increase due to microarchitecture advances is roughly proportional to the square root of the increase in complexity". Complexity in this context means processor logic, i.e. its area.

WIKIPEDIA

From multi-core era to many-core era

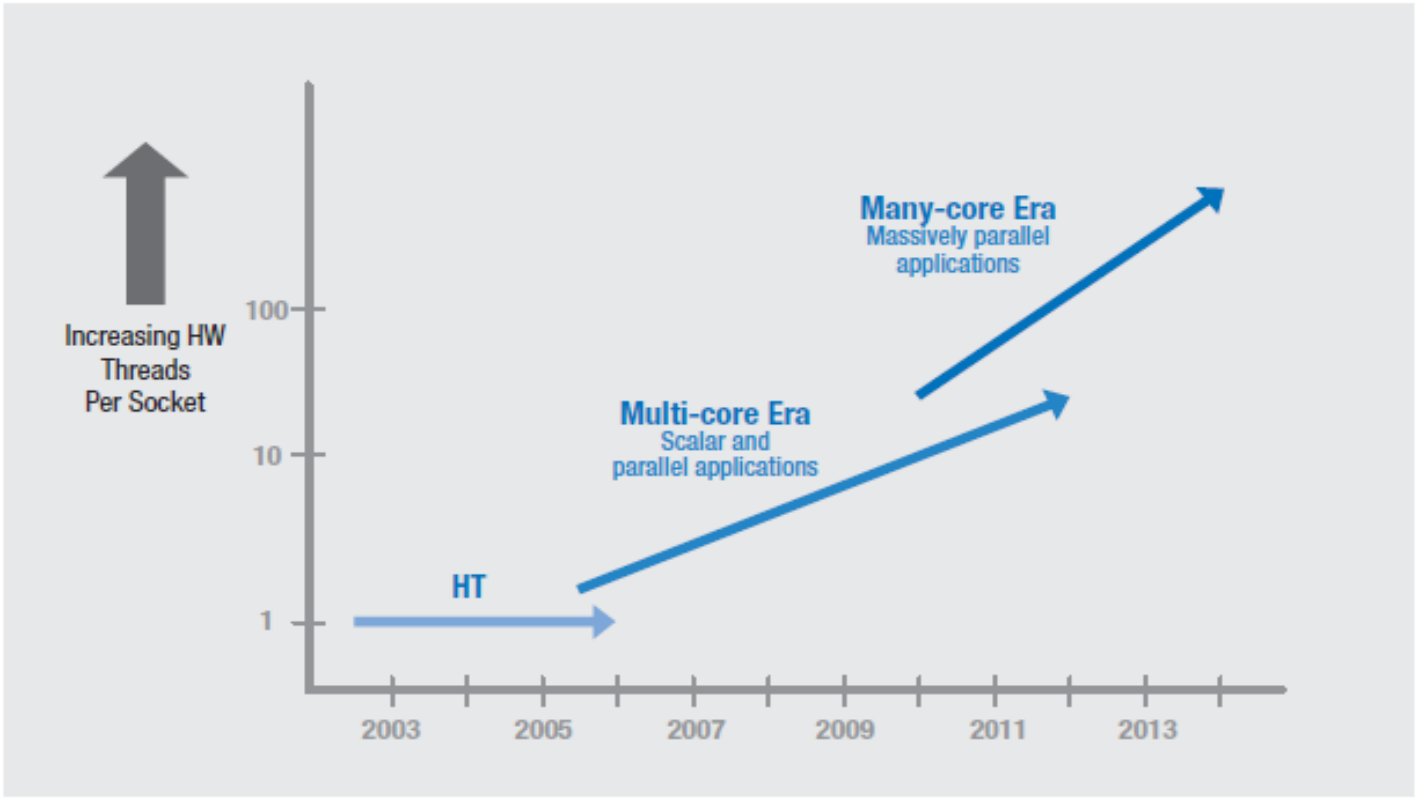
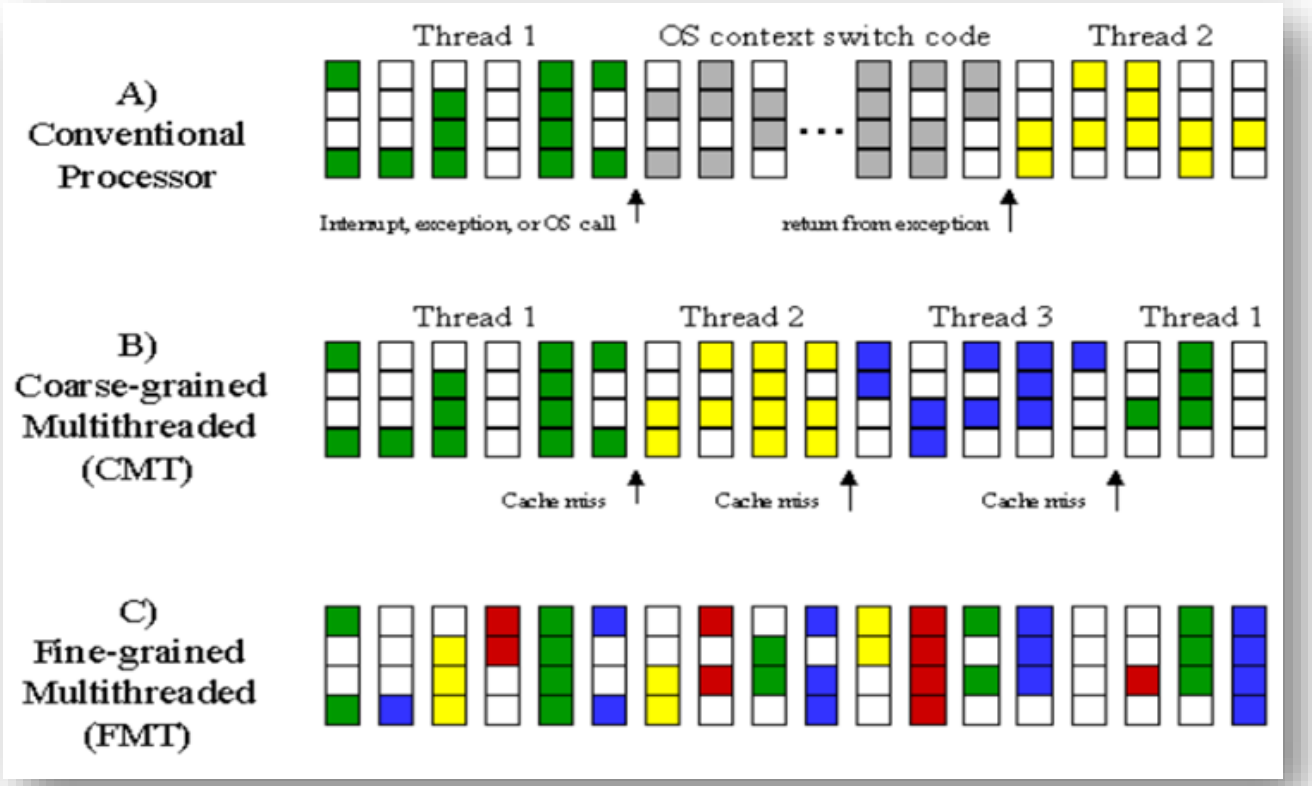


Figure 1: Current and expected eras of Intel® processor architectures

Platform 2015: Intel® Processor and Platform Evolution for the Next Decade, 2005

Multithreading (1/2)

- During a branch miss recovery and access to the main memory by a cache miss, ALUs have no jobs to do and have to be idle.
- Executing multiple independent threads (programs) will mitigate the overhead.
- They are called coarse- and fine-grained multithreaded processors having multiple architecture states.



http://www.realworldtech.com/alpha-ev8-smt/

Multithreading (2/2)

- Simultaneous Multithreading (SMT) can improve hardware resource usage.

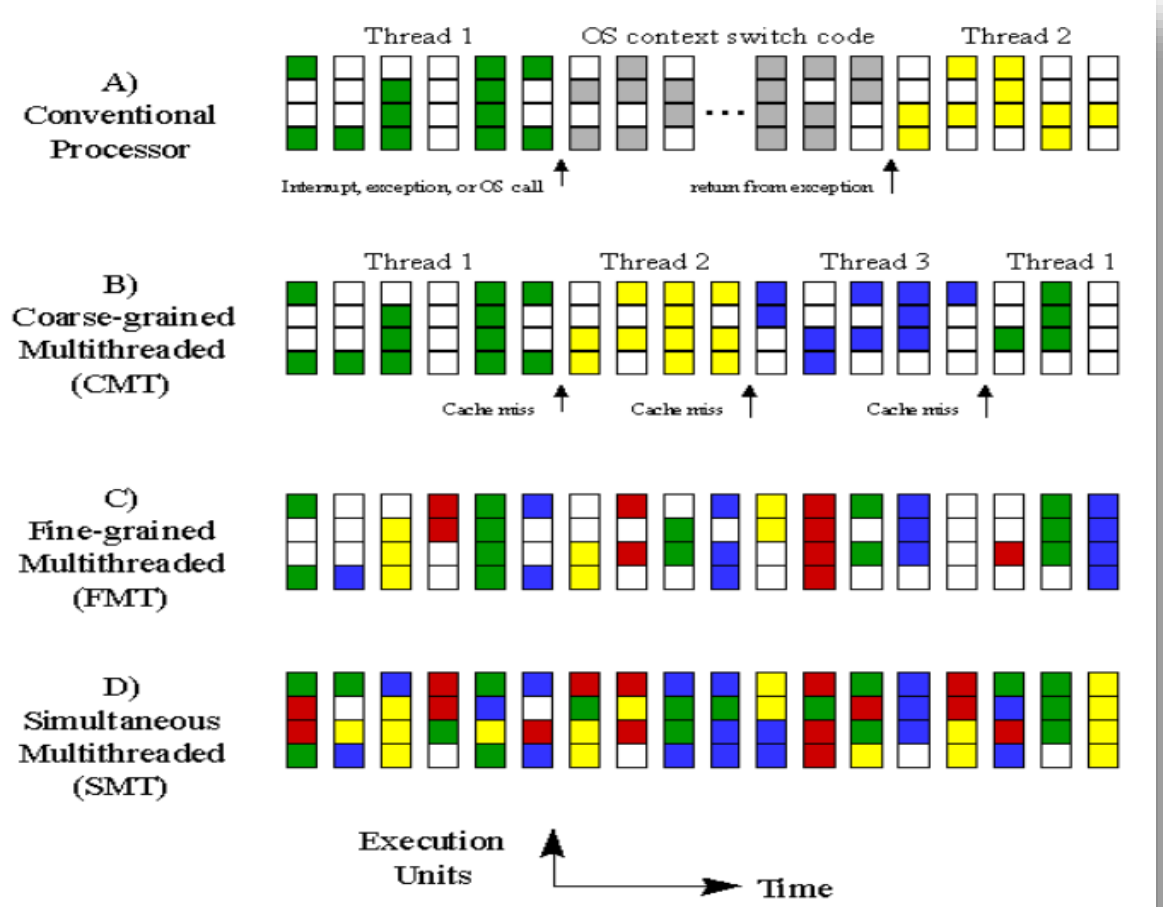


Figure 1. Multithreaded Execution with Increasing Levels of TLP Hardware Support