Complex Pipelines

Miguel Gomez-Garcia

(slides adapted from prior 6.823 offerings)

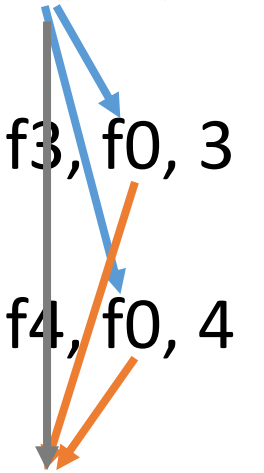
Dependence vs. hazard

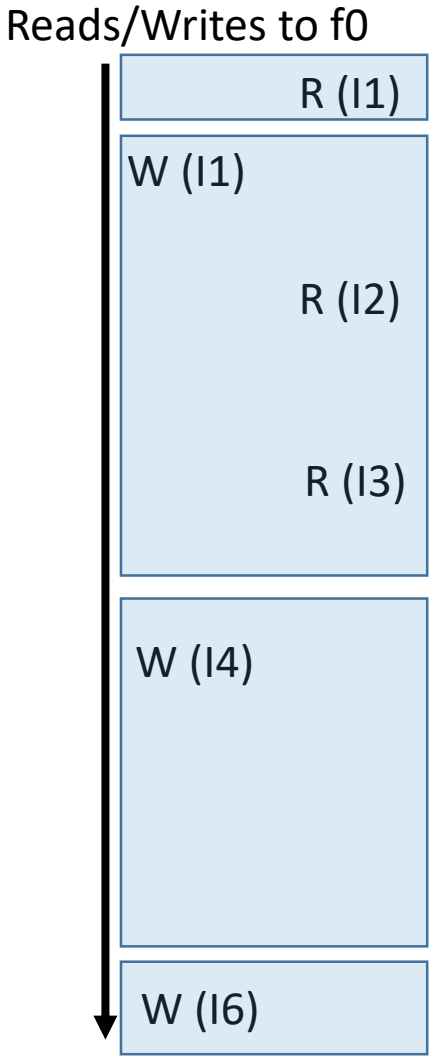
- Dependence is a property of programs
- Whether a dependence results in a hazard is a property of pipeline organizations

Data hazard types

- RAW
- WAR
- WAW

```
I1: ADDI   f0, f0, 0
I2: ADDI   f3, f0, 3
I3: ADDI   f4, f0, 4
I4: ADDI   f0, f5, 1
I5: XOR    f6, f6, f6
I6: ADDI   f0, f7, 1
```



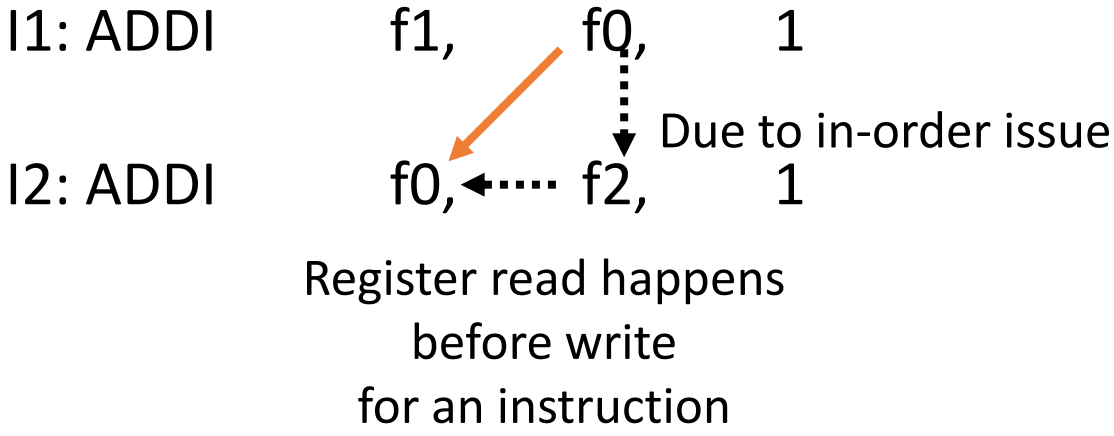


Scoreboard

- A data structure that detects hazards dynamically
- Applicable to both in-order and out-of-order issue
- Why do we need this?
  - Many execution units
  - Variable execution latency
  - Dynamic instruction scheduling

Scoreboard

- Can have many implementations!
- Example: In-order issue
  - WAR cannot happen (if value is latched to functional unit at issue)



  - Can be simplified as Busy[FU#] and WP[reg#] (if WAW resolved conservatively)

Scoreboard

- What strategy does it use to resolve RAW?
  - Stall
- How about bypass?
  - Less beneficial since the register write can happen right after execution finishes
  - Can still be incorporated to allow register read and write to happen in the same cycle

Out of Order (OoO) Execution

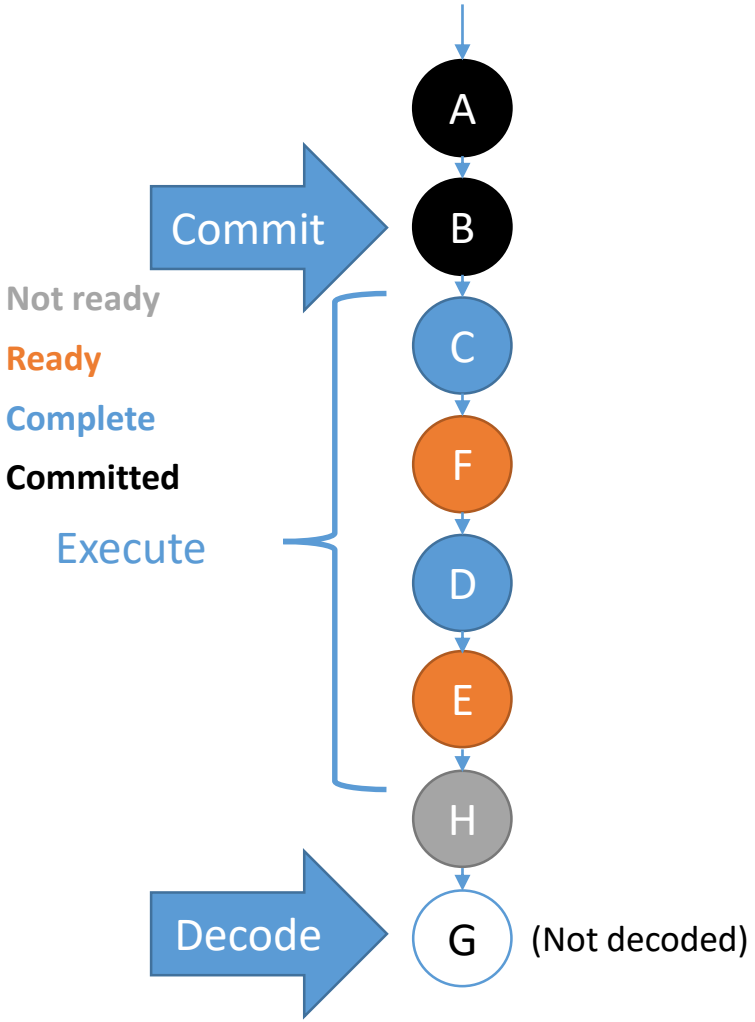
- Why use it in the first place?
  - Stalls of younger instructions prevent dispatch of younger instructions into functional (execution) units.

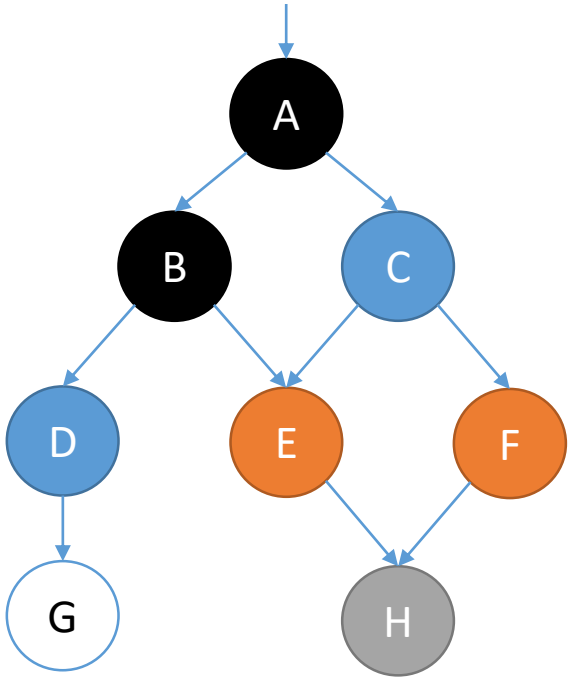
```
MUL   R3 <- R1, R2
ADD   R3 <- R3, R1
ADD   R1 <- R6, R7
MUL   R5 <- R6, R8
ADD   R7 <- R3, R5
```

```
LD    R3 <-R1 (0)
ADD   R3 <- R3, R1
ADD   R1 <- R6, R7
MUL   R5 <- R6, R8
ADD   R7 <- R3, R5
```

  - By eliding false dependences and head-of-line blocking, we are only bound by true data dependences

OoO execution dynamically extracts true dependences





Out-of-order execution

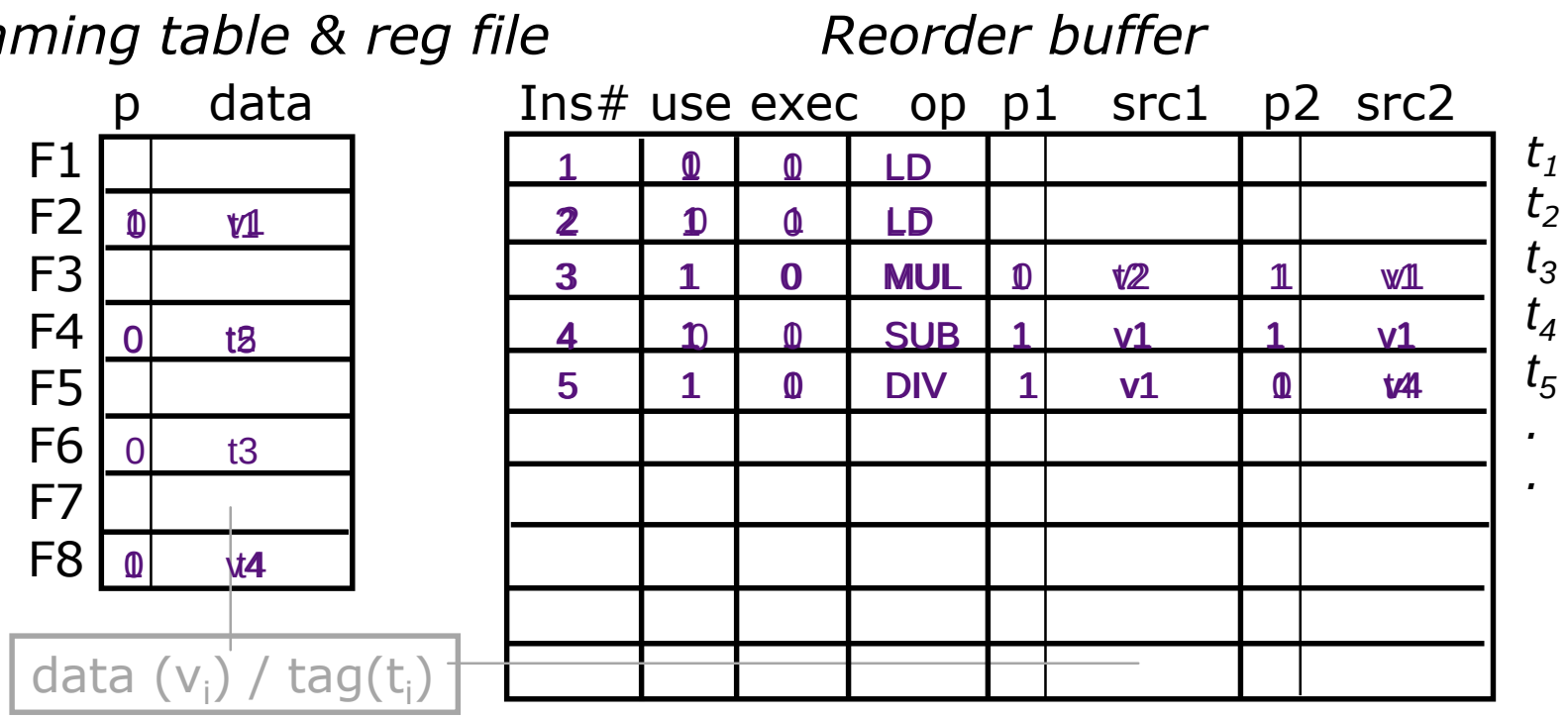
- Want: we want to somehow avoid stalling due to WAR and WAW hazards...
  - Strategy?

  Do something else

  - Technique?

  Register Renaming

Renaming + ROB



- *Insert instruction in ROB*
- *Issue instruction from ROB*
- *Complete instruction*
- *Empty ROB entry*

What happens if I3 raises exception?

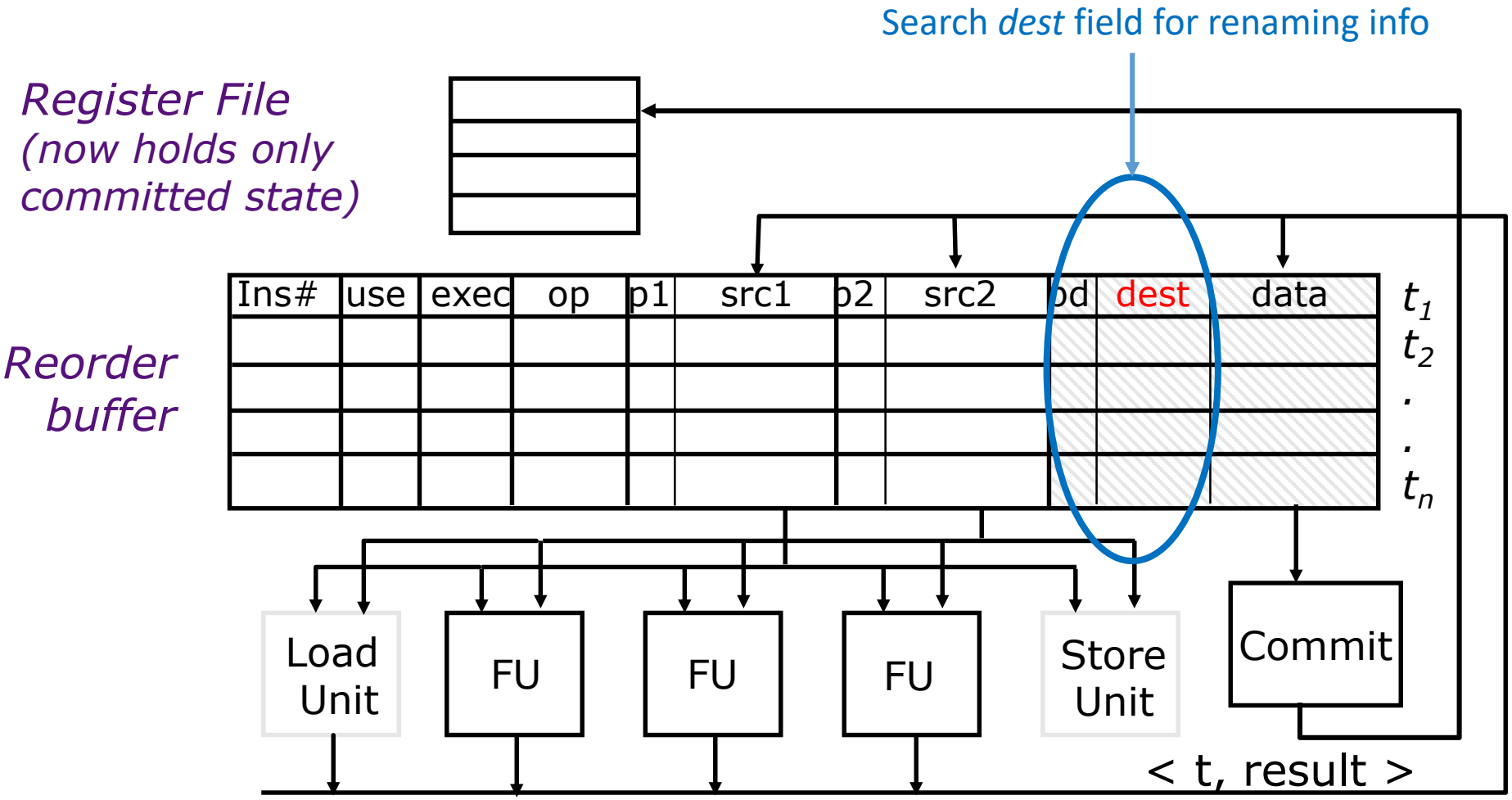
1	LD	F2,	34(R2)	
2	LD	F4,	45(R3)	
3	MULTD	F6,	F4,	F2
4	SUBD	F8,	F2,	F2
5	DIVD	F4,	F2,	F8
6	ADDD	F10,	F6,	F4

Precise Exceptions

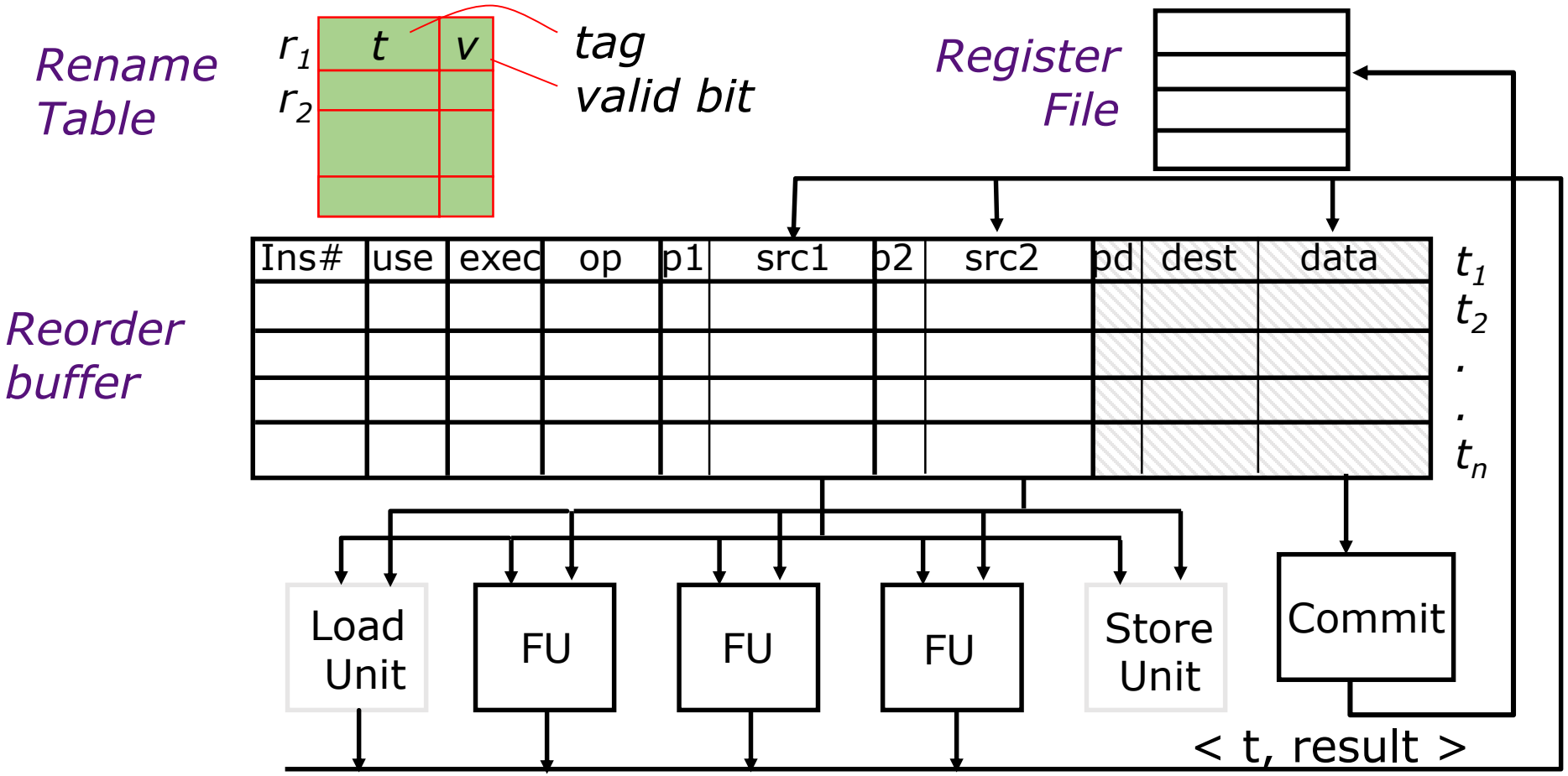
- Issue:
  - Exception ordering is not accurate
  - Instructions that come earlier in program order might not raise an exception until after later instructions have been completed!
- Solution:
  - Introduce in-order commit point!

But where do we store data before commit?

Exception-friendly ROB



Renaming Table



Questions?