Branch Prediction and OoO Techniques

Ryan Lee

(slides adapted from prior 6.823 offerings)

Since Last Time…

1. Branch Prediction
   - Relates to your lab 2!
   - Covered several different schemes, from simple to more complex...

2. Speculation
   - Data-in-ROB vs. unified-register-file
   - Centralized vs. distributed
   - ROB vs. issue queue + commit queue

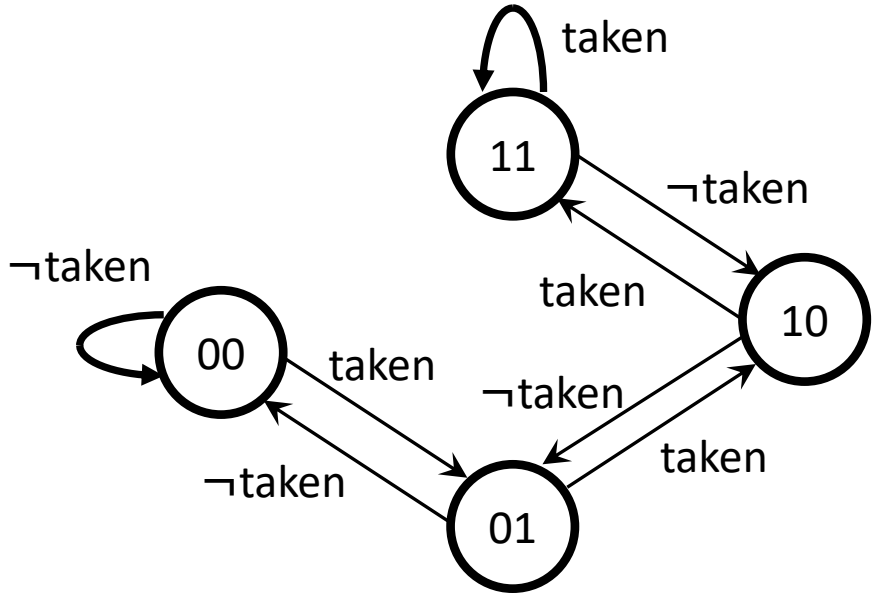
Branch Prediction

Control Flow Dependences. How to handle them?

- Stall: Delay until we know the next PC
- Speculate: Guess next value
- Do something else: Multi-threading

Branch Predictors

- 1-bit predictor
- 2-bit predictor



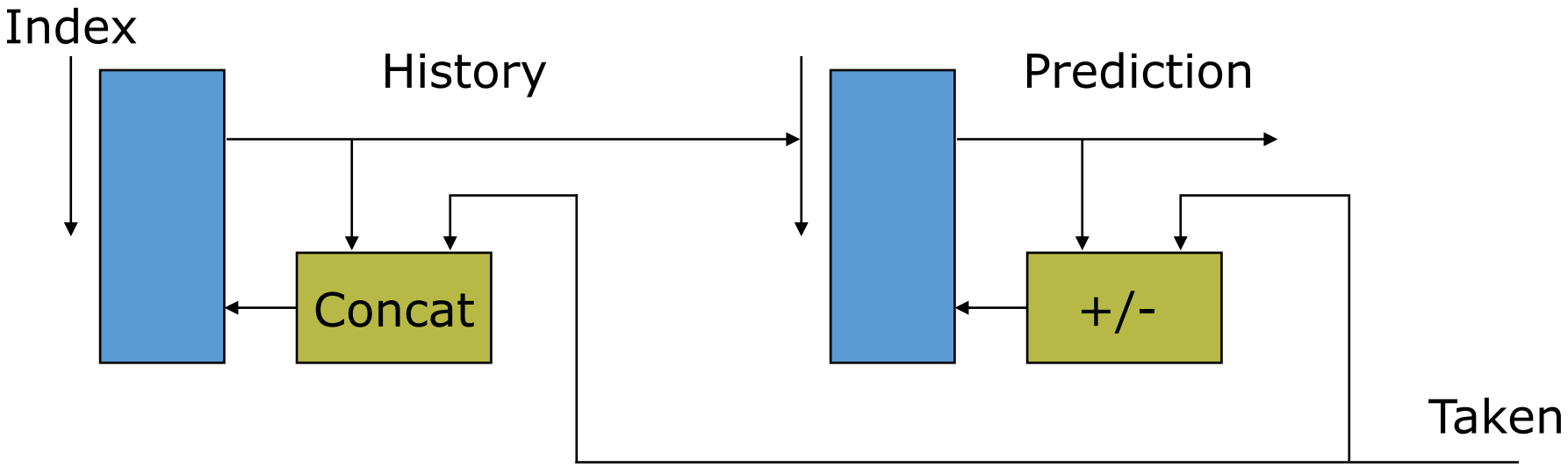
Branch Predictors

Two empirical observations

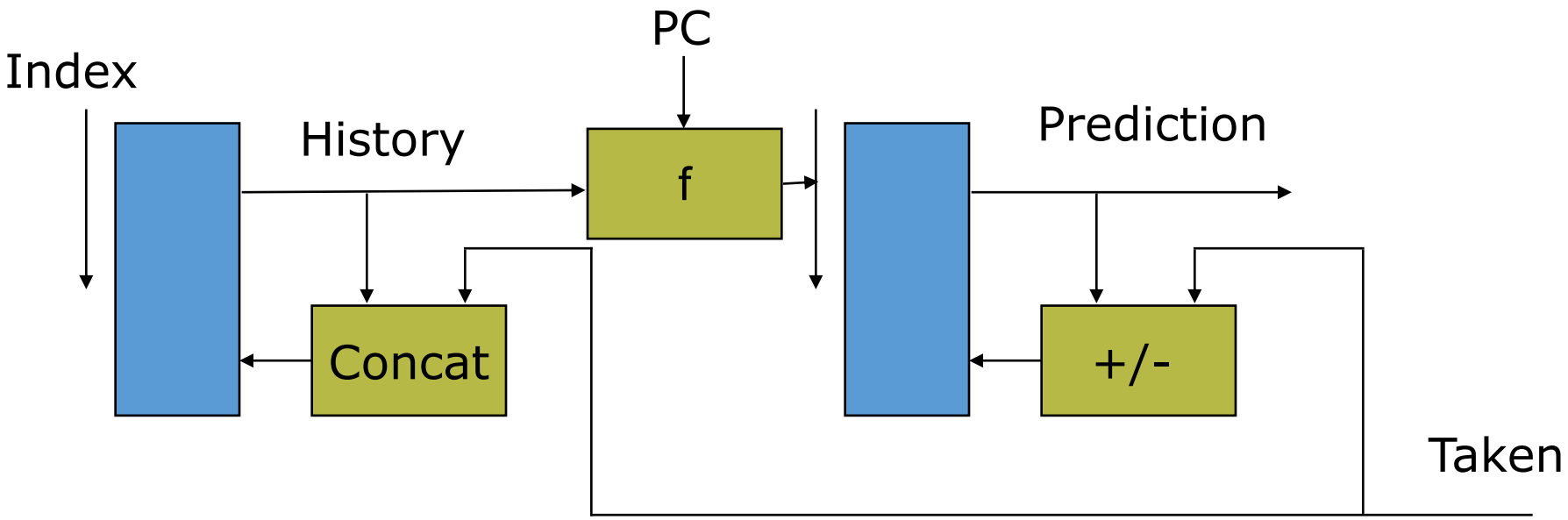
1. A branch's outcome can be correlated with other branches' outcomes
   - Global branch correlation

2. A branch's outcome can be correlated with past outcomes of the same branch
   - Local branch correlation

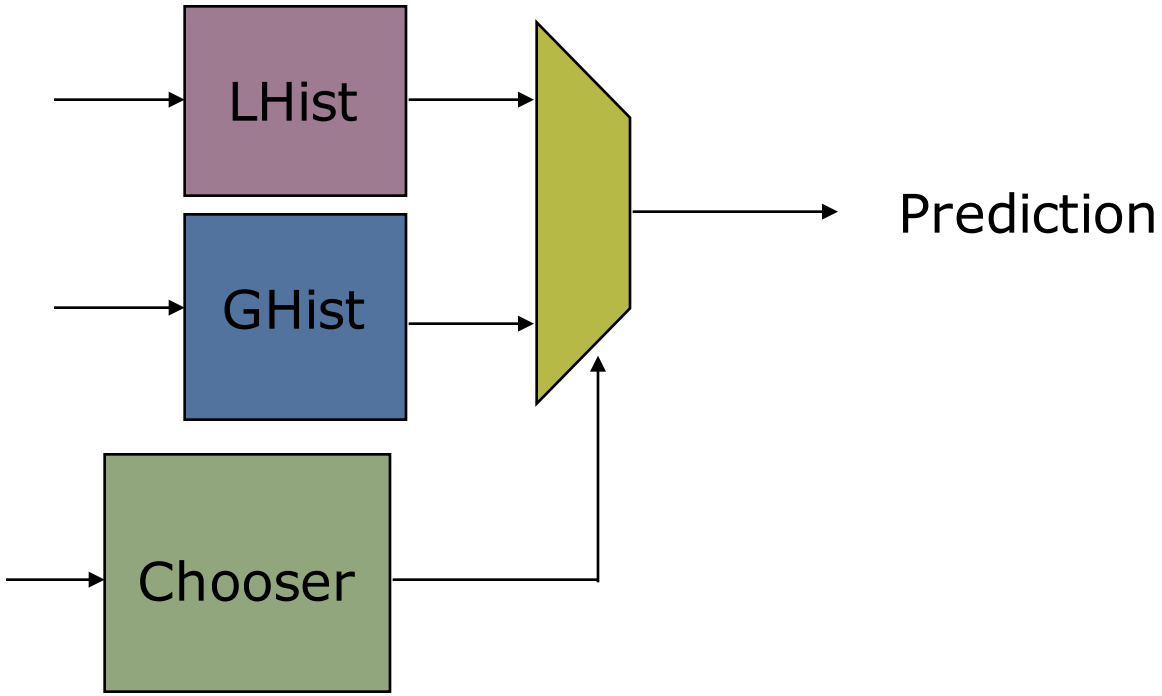
History-based Prediction



Two-level Predictor



Tournament Predictors



Lab 2 Due Oct 22

- Get going early
- Incrementally build more complex predictors
  - 2-bit predictor
  - Local history predictor
  - Tournament predictor
  - and so on
- Recommend researching more advanced predictors for full credit
  - TAGE, Perceptron, etc...

Out of Order (OoO) Execution

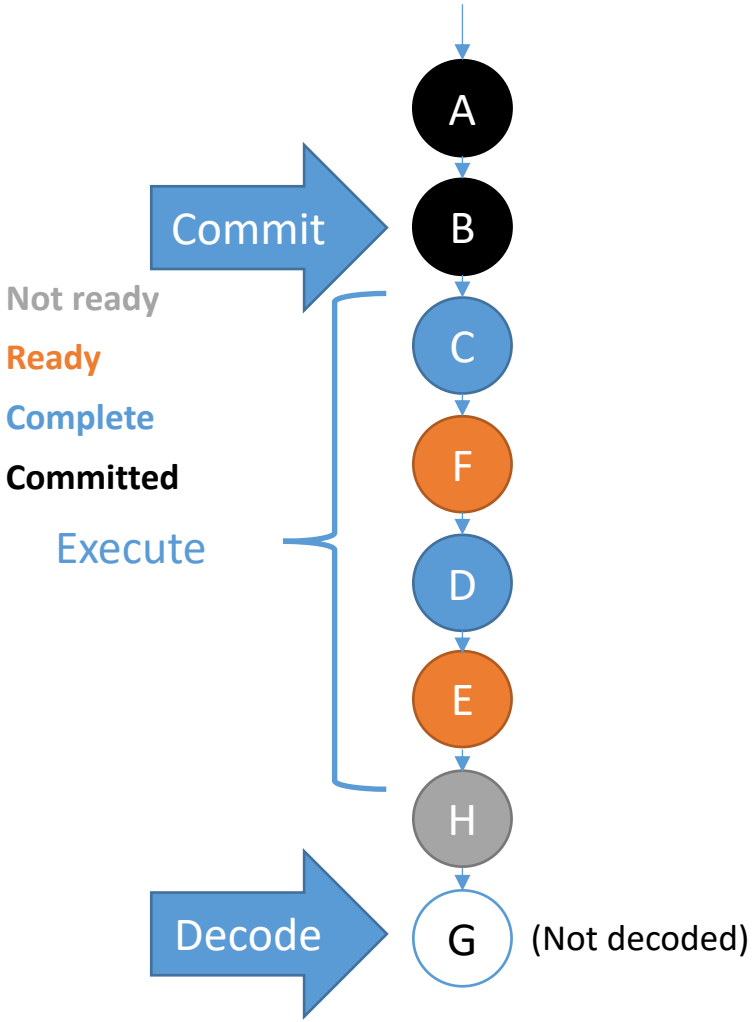
- Why use it in the first place?
  - Stalls of younger instructions prevent dispatch of younger instructions into functional (execution) units.

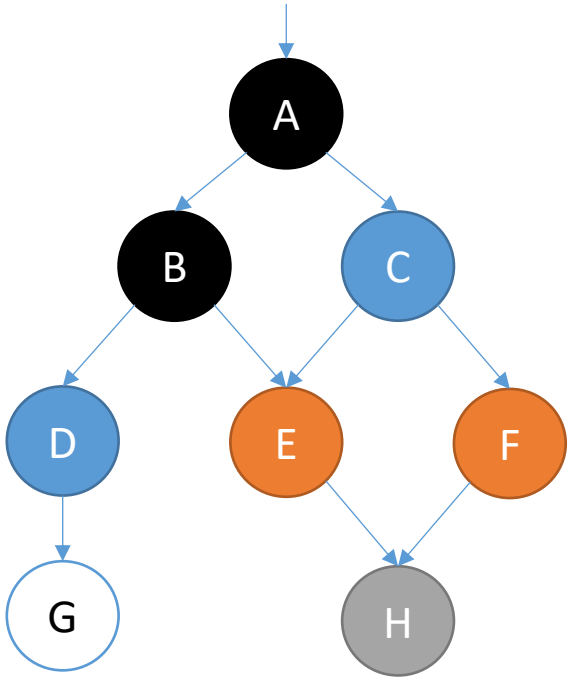
```
MUL   R3 <- R1, R2
ADD   R3 <- R3, R1
ADD   R1 <- R6, R7
MUL   R5 <- R6, R8
ADD   R7 <- R3, R5
```

```
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



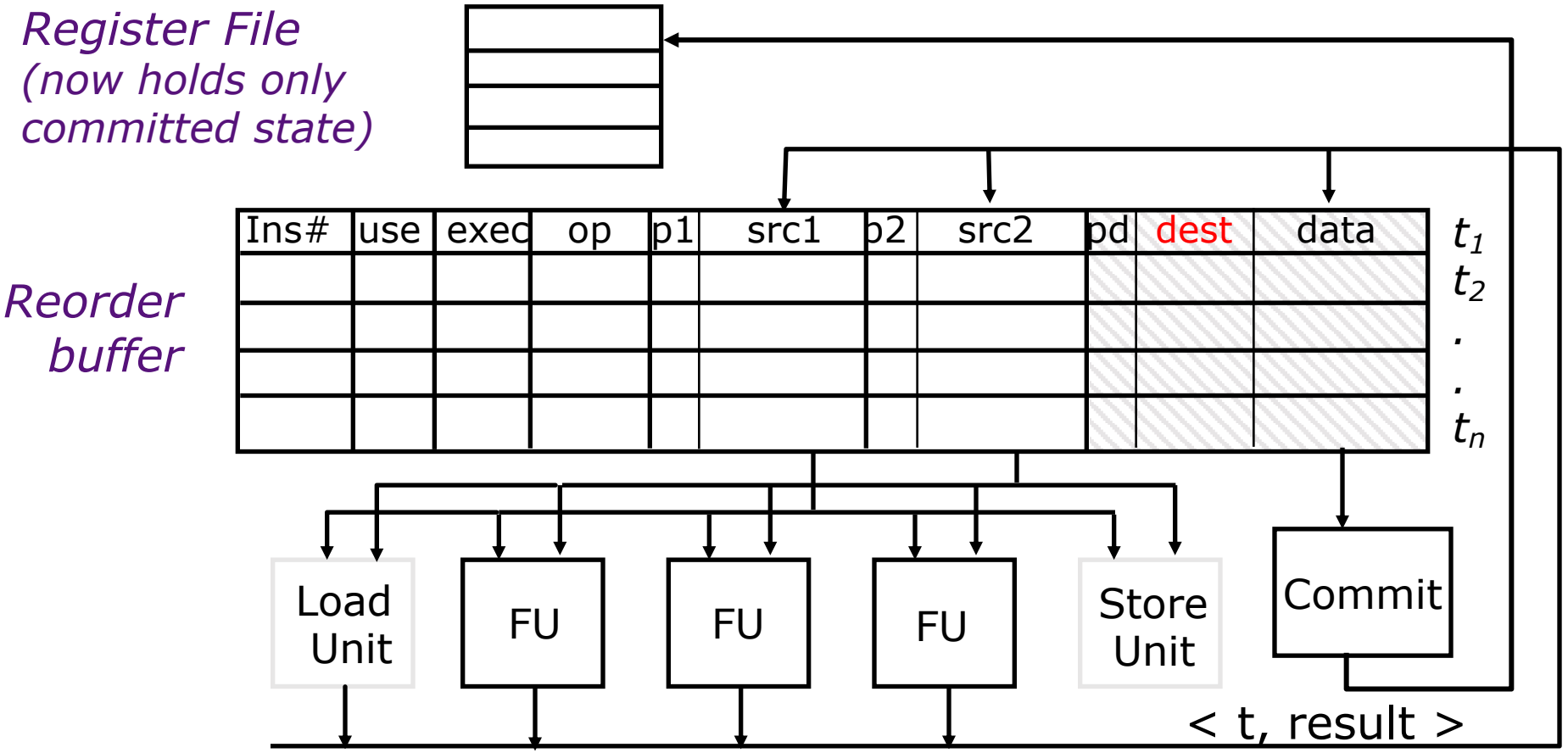


Out-of-order execution

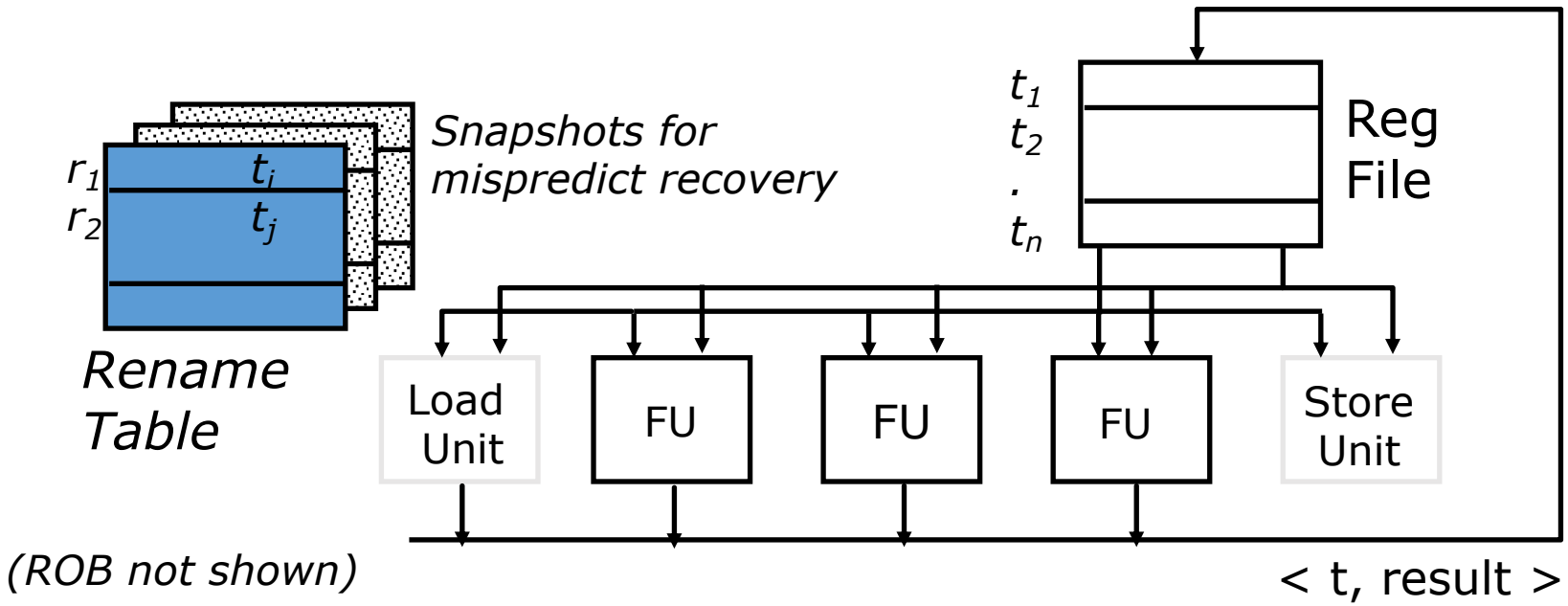
- Data-in-ROB vs. unified register file
  - Key difference: Where do we keep speculative values?
  - A difference in value management

- Value management styles
  - Greedy (Eager): update in place, keep a log, recover from log in commit
  - Lazy: update in buffer, replace old values at commit

Data-in-ROB

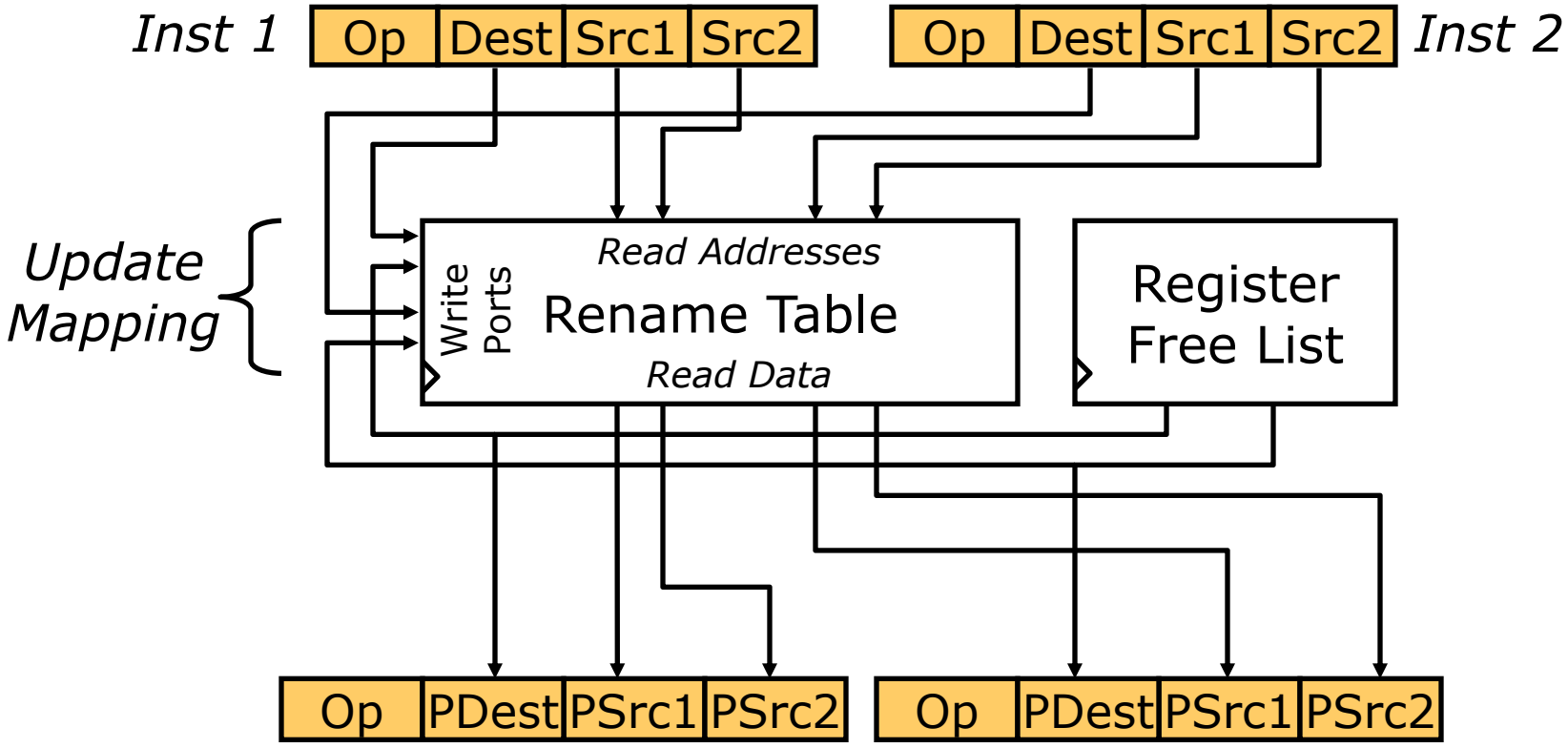

Register File
(now holds only
committed state)
Reorder
buffer
Ins#
use
exec
op
p1
src1
p2
src2
pd
dest
data
t_1
t_2
.
.
t_n
Load
Unit
FU
FU
FU
Store
Unit
Commit
< t, result >

Unified register file



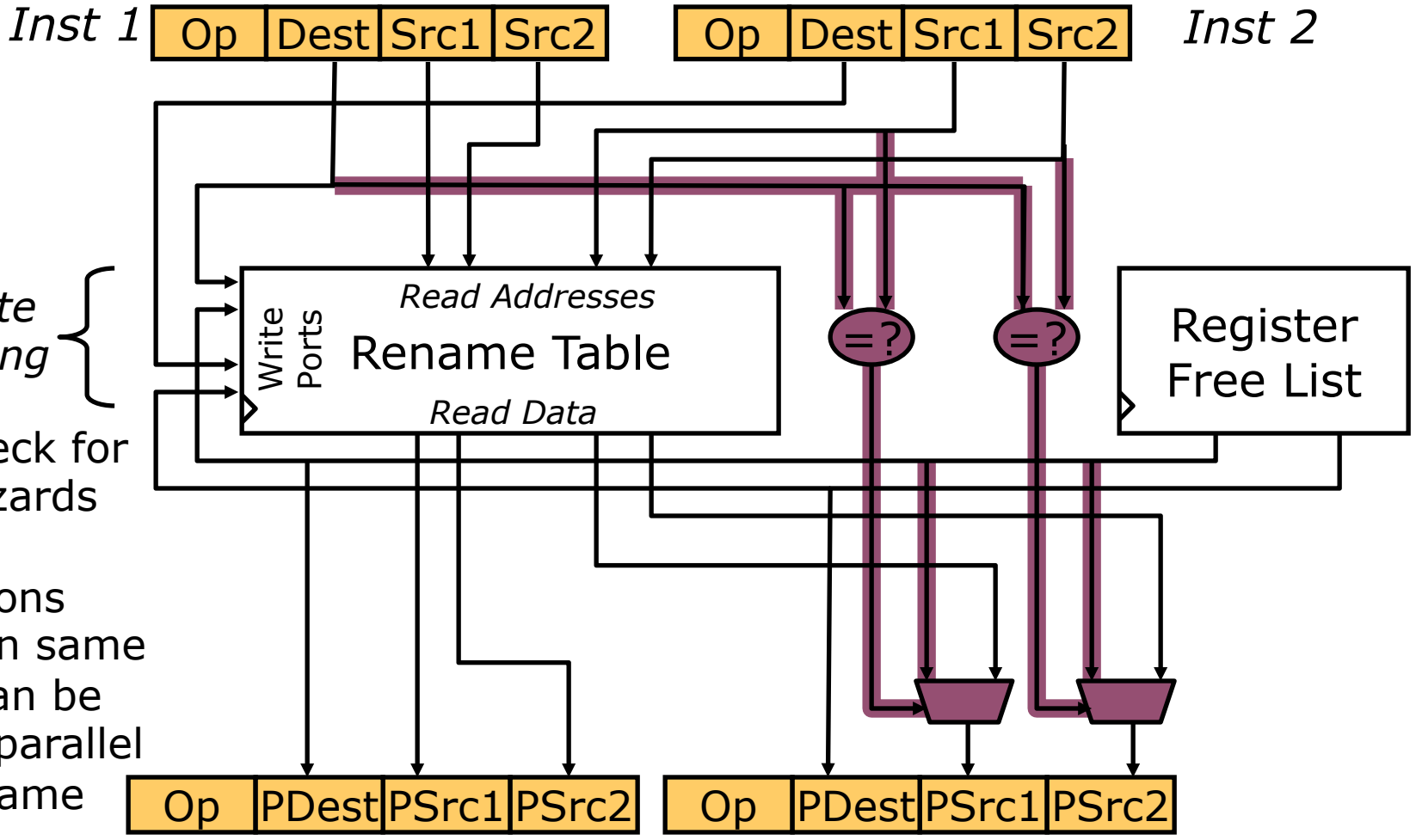
Superscalar Register Renaming

- During decode, instructions allocated new physical destination register
- Source operands renamed to physical register with newest value
- Execution unit only sees physical register numbers



Does this work?

Superscalar Register Renaming



Must check for RAW hazards between instructions issuing in same cycle. Can be done in parallel with rename lookup.

(MIPS R10K renames 4 serially-RAW-dependent insts/cycle)

Split Issue and Commit Queue

- How large should the ROB be?
  - Think Little's Law…
- Can split ROB into issue and commit queues

Issue Queue

use	op	p1	PR1	p2	PR2	tag

Commit Queue

ex	Rd	LPRd	PRd

- Commit queue: Allocate on decode, free on commit
- Issue queue: Allocate on decode, free on dispatch
- Pros: Smaller issue queue → simpler dispatch logic
- Cons: More complex mis-speculation recovery

Questions?