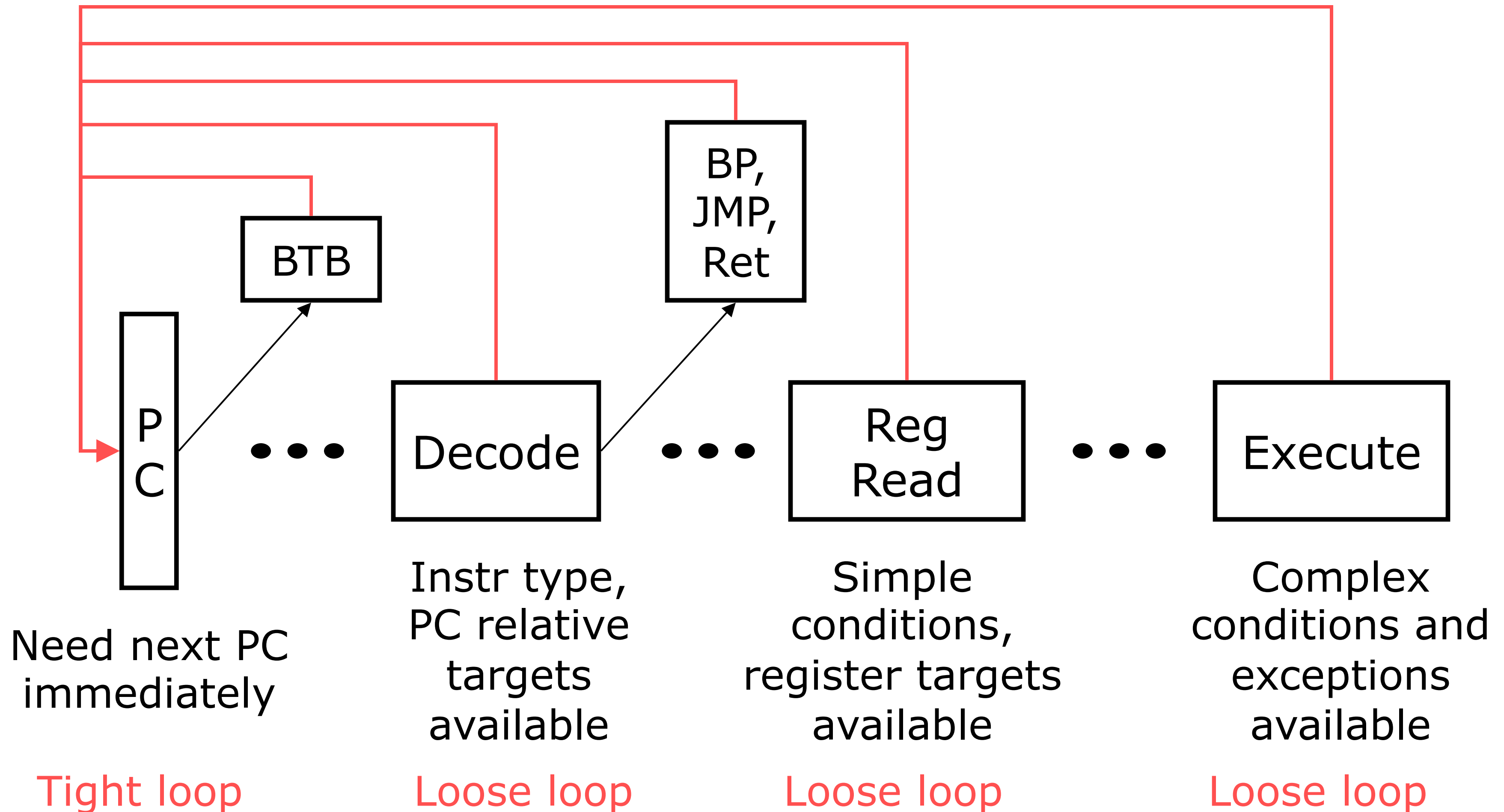
Speculative Execution

Daniel Sanchez
Computer Science and Artificial Intelligence Laboratory
M.I.T.

Overview of branch prediction



Must speculation check always be correct?

Speculative Execution Recipe

1. Proceed ahead despite unresolved dependencies using a prediction for an architectural or micro-architectural value

2. Maintain both old and new values on updates to architectural (and often micro-architectural) state

After speculation check

3. After sure that there was no mis-speculation and there will be no more uses of the old values, discard old values and just use new values

OR

3. In event of mis-speculation, dispose of all new values, restore old values, and re-execute from point before mis-speculation

Why might one use old values?

Value Management Strategies

Greedy (or Eager) Update:

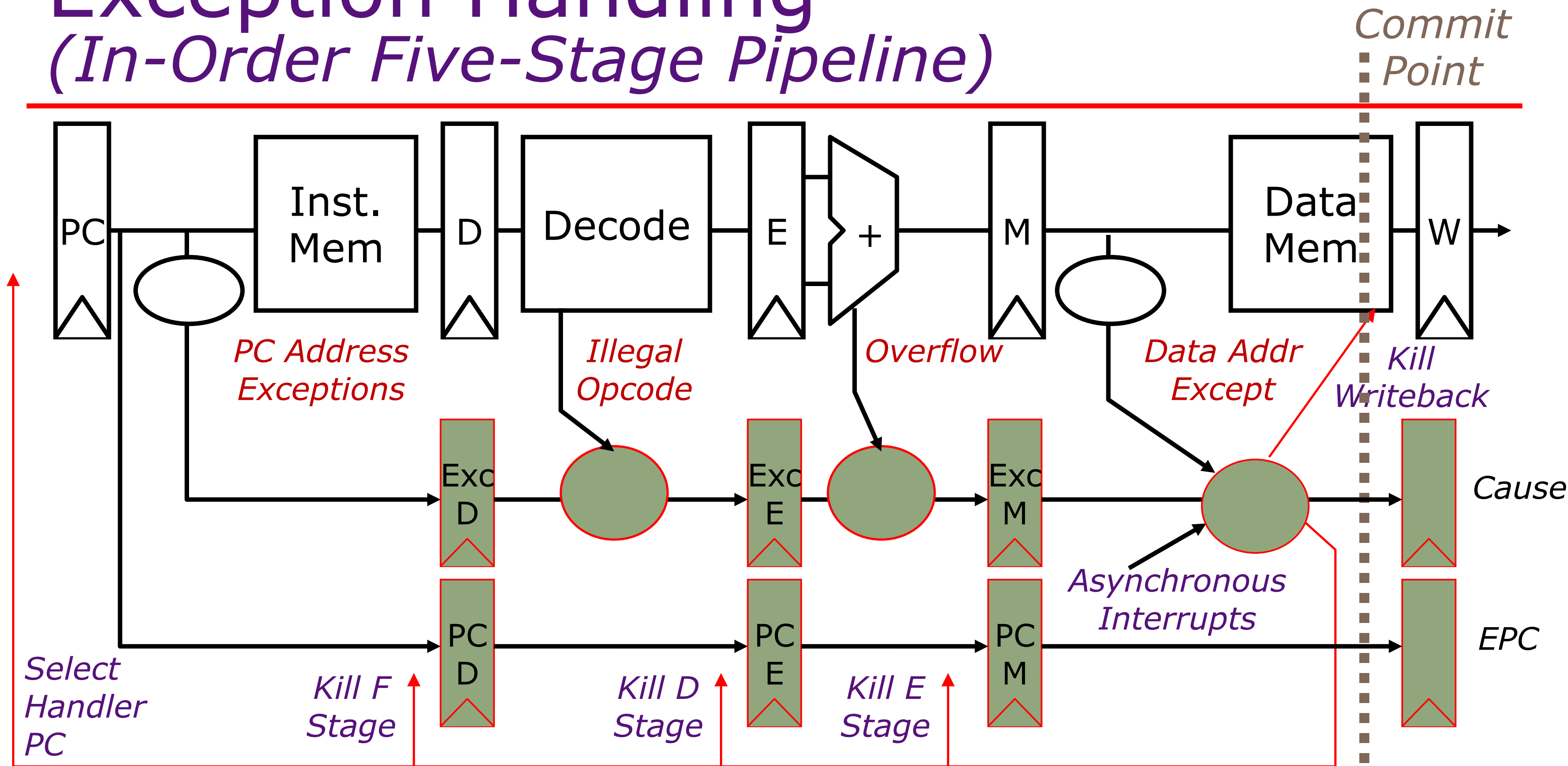
- Update value in place, and
- Provide means to reconstruct old values for recovery
  - often this is a log of old values

Lazy Update:

- Buffer new value, leaving old value in place
- Replace old value only at 'commit' time

Why leave an old value in place?

Exception Handling
(In-Order Five-Stage Pipeline)



Strategy for Registers?

Strategy for PC?

Misprediction Recovery

In-order execution machines:

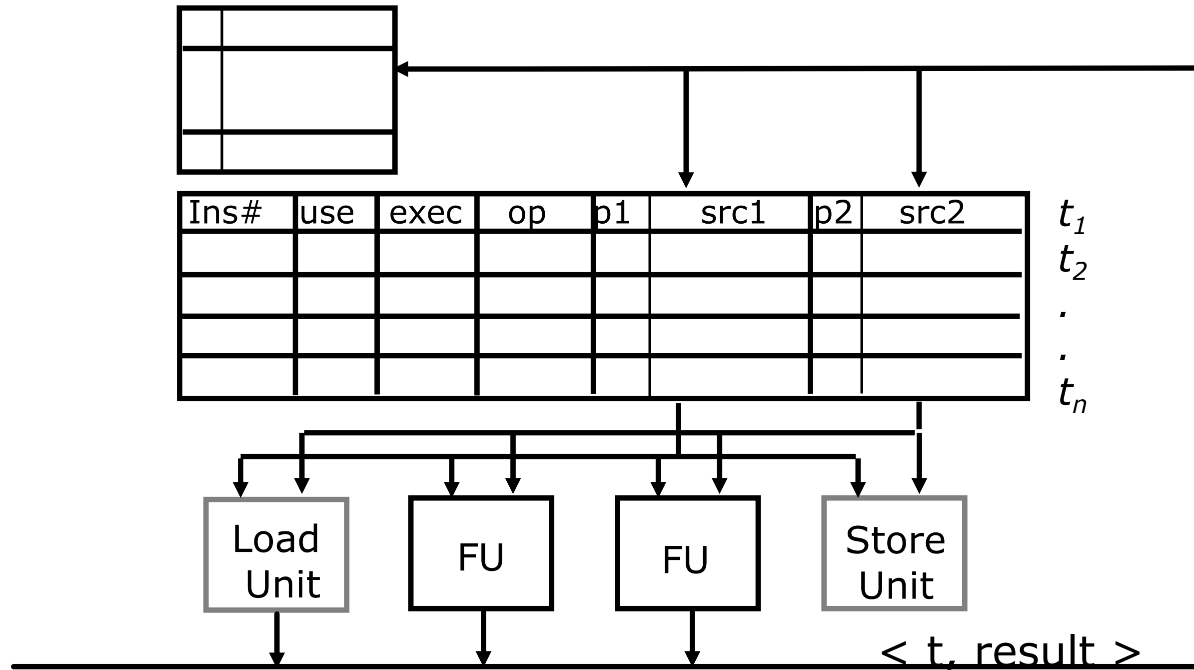
- Guarantee no instruction issued after branch can write-back before branch resolves by keeping values in the pipeline

- Kill all values from all instructions in pipeline behind mispredicted branch

Out-of-order execution?

- Multiple instructions following exception in program order can generate new values before exception resolves

Data-Driven Execution (Tomasulo)

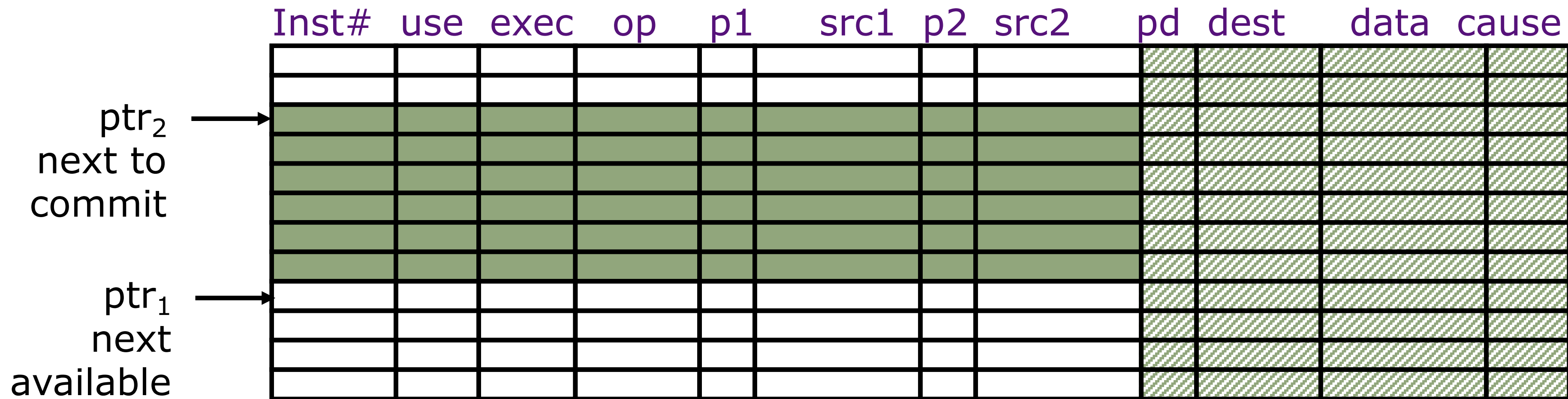


Basic Operation:

- Enter op and tag or data (if known) for each source
- Replace tag with data as it becomes available
- Issue instruction when all sources are available
- Save dest data when operation finishes

Update strategy?

Extensions for Mis-speculation Recovery

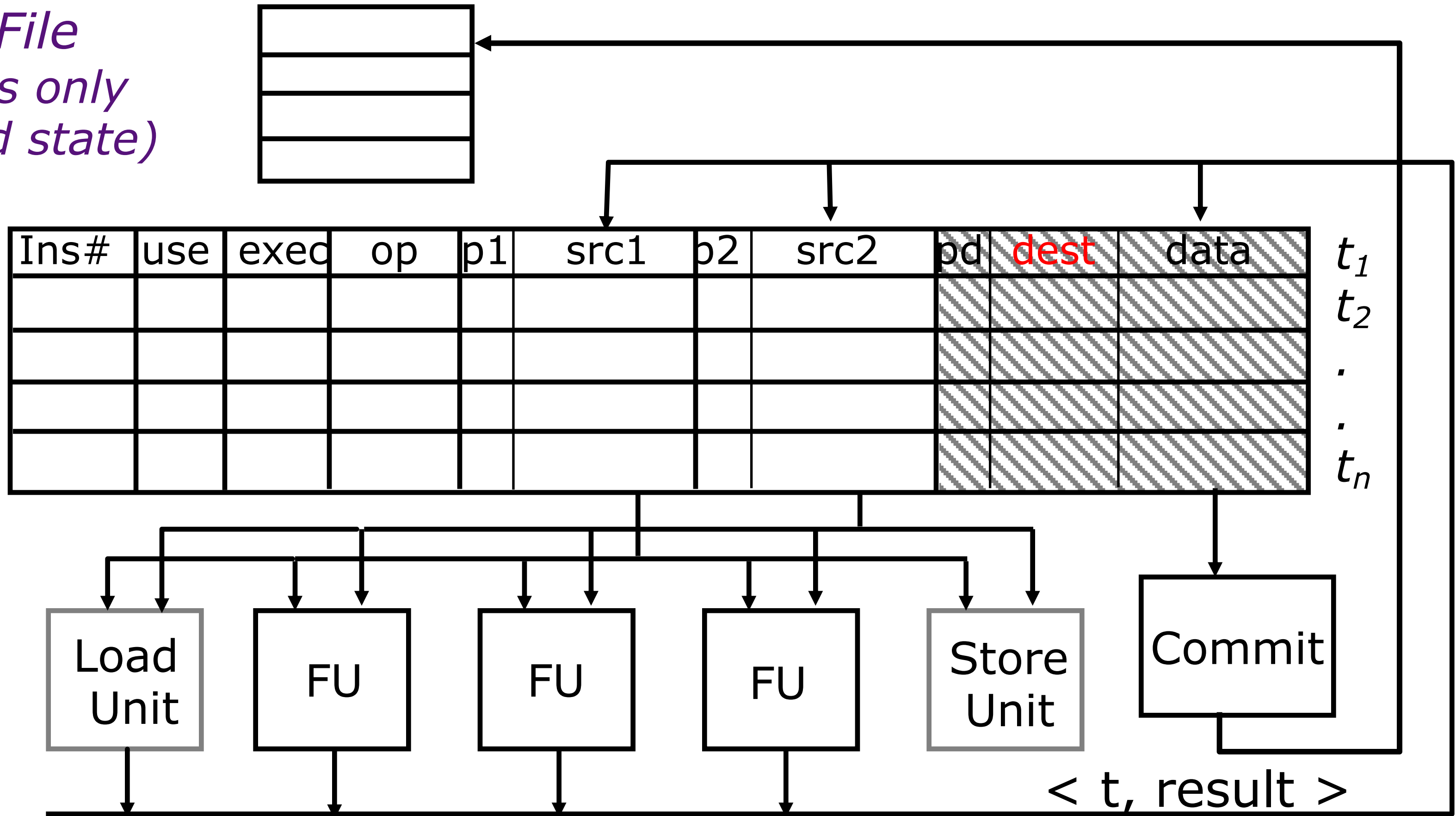


Reorder buffer

- add <pd, dest, data, cause> fields in the instruction template
- commit instructions to reg file and memory in program order ⇒ buffers can be maintained circularly
- on exception, clear reorder buffer by resetting $ptr_1 = ptr_2$
  (stores must wait for commit before updating memory)

What is the update policy of registers?

Rollback and Renaming

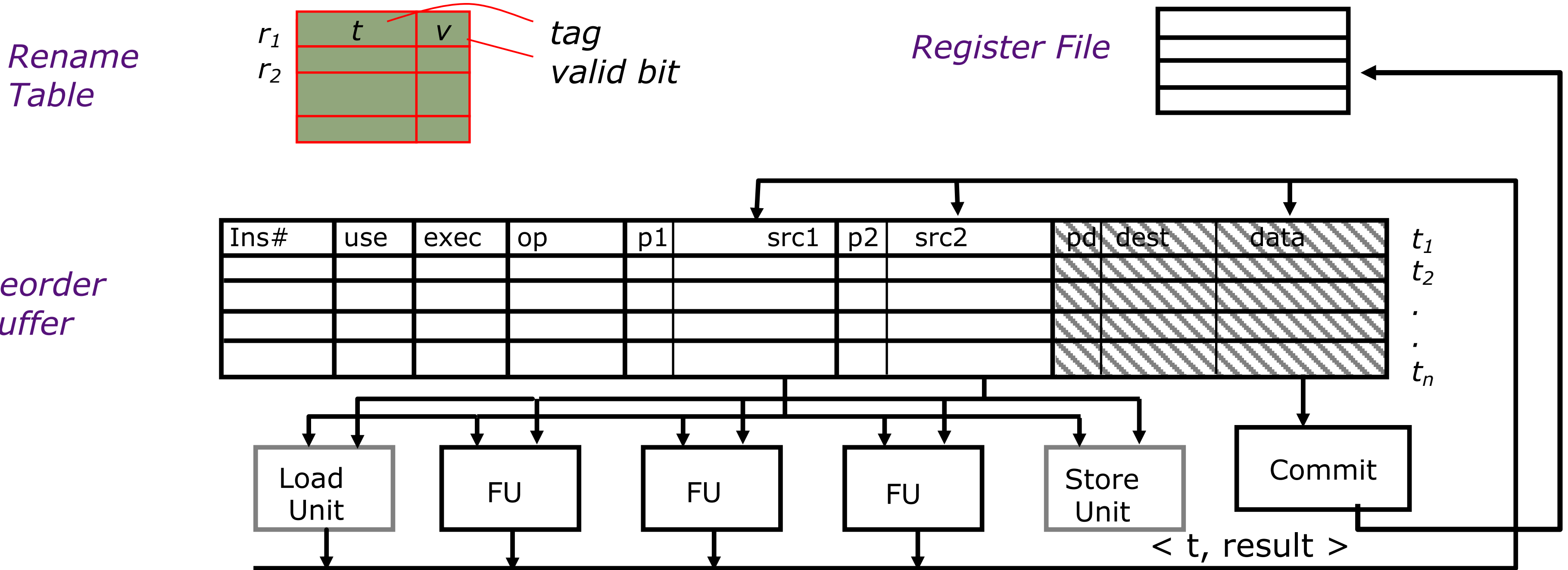


Convert to lazy by holding data in ROB.

But how do we find values before they are committed?

Renaming Table

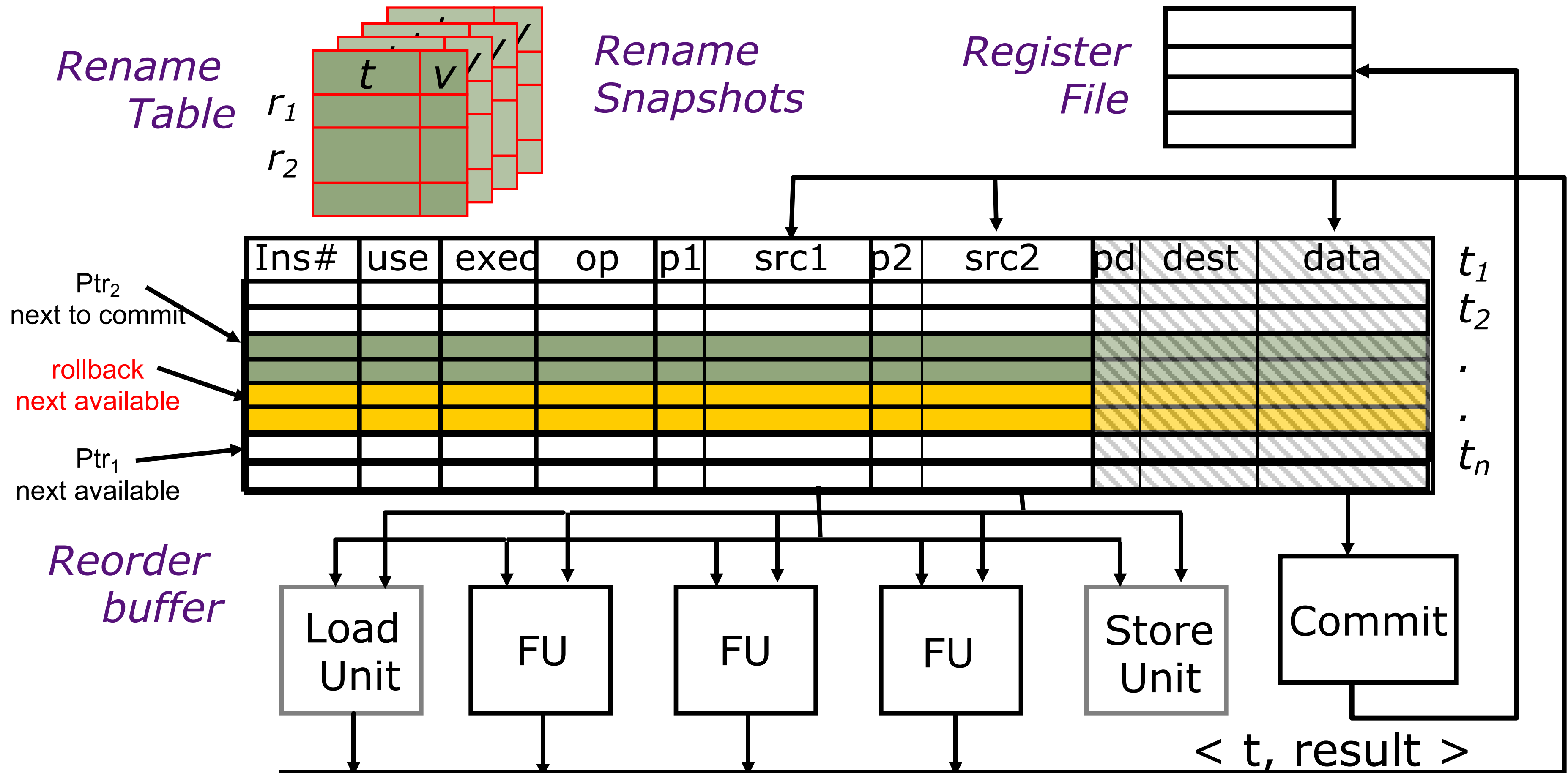
Micro-architectural speculative cache to speed up tag lookup.



What is the update policy of rename table?
What events cause mis-speculation?

How can we respond to mis-speculation on rename table?
After being cleared, when can instructions be added to ROB?

Recovering ROB/Renaming Table



Take snapshot of register rename table at each predicted branch, recover earlier snapshot if branch mispredicted

Map Table Recovery - Snapshots

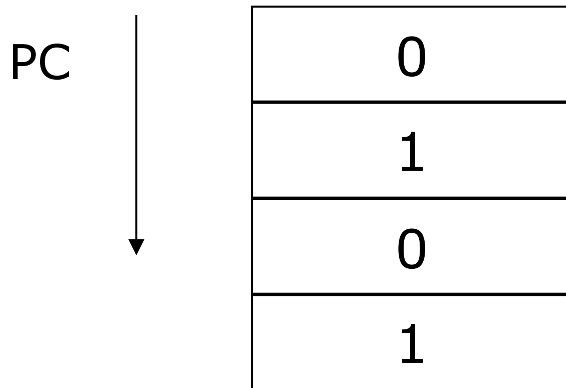
Speculative value management of microarchitectural state

	Reg Map	V	Snap Map	V	Snap Map	V
R0	T20	X	T20	X	T20	X
R1	T73	X	T73	X	T08	
R2	T45	X	T45	X	T45	X
R3	T128		T128		T128	X
⋮	⋮		⋮		⋮	
R30	T54		T54		T54	
R31	T88	X	T88	X	T88	X

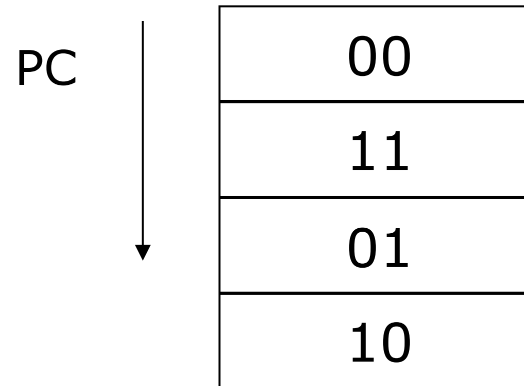
What kind of value management is this?

Branch Predictor: Speculative Value Management

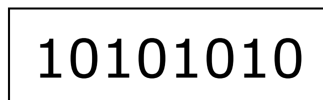
- 1-Bit Counter



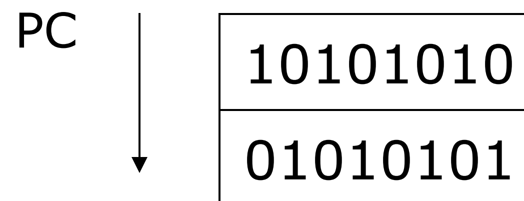
- 2-Bit Counter



- Global History

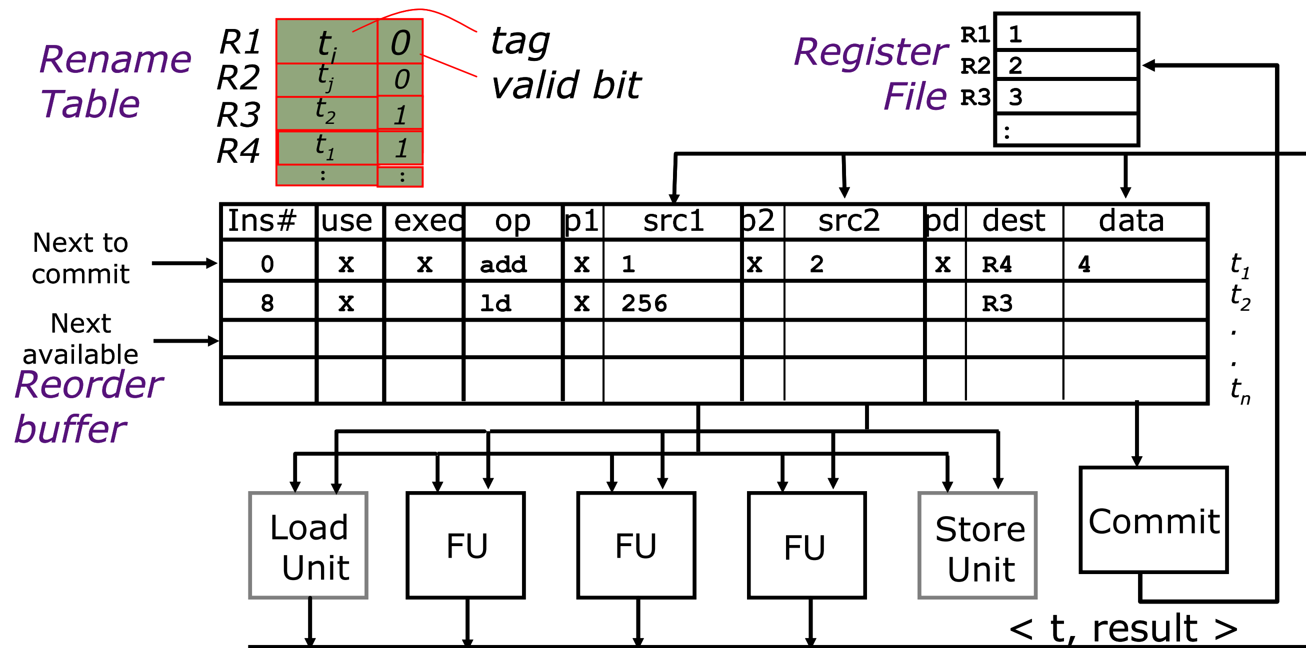


- Local History



O-o-O Execution with ROB

Data-in-ROB design

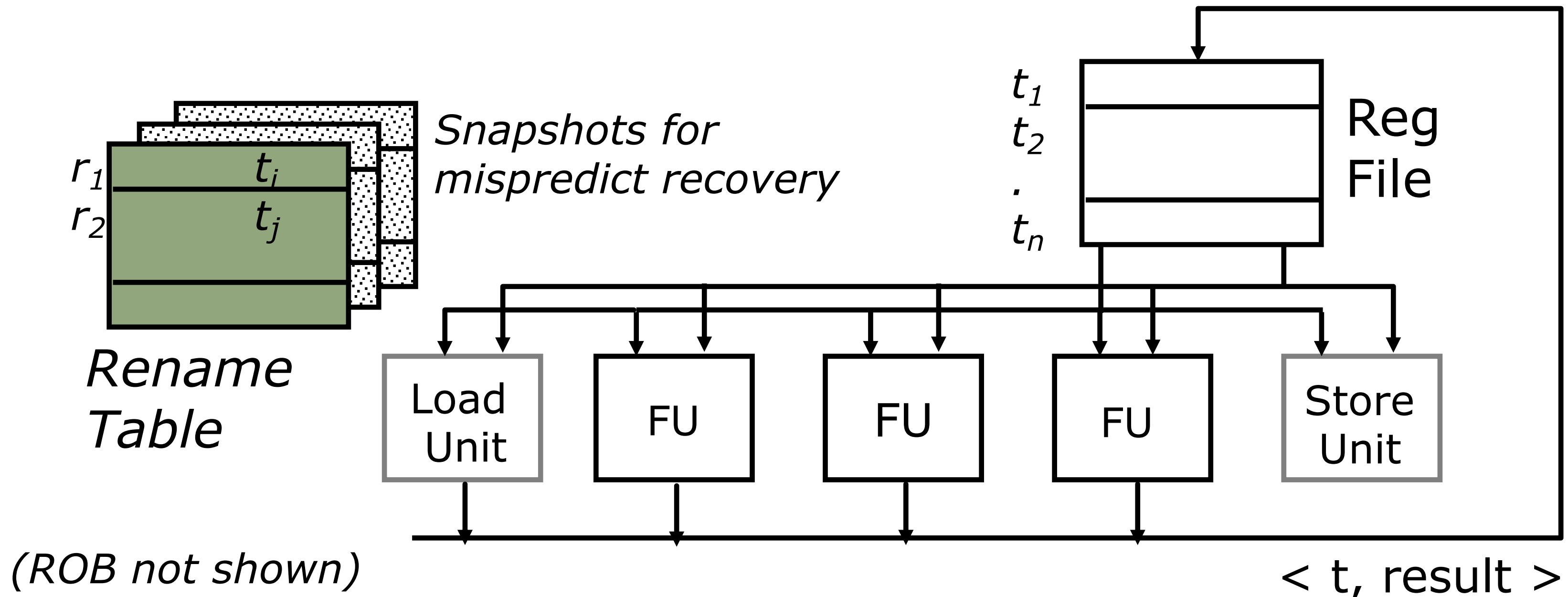


Basic Operation:

- Enter op and tag or data (if known) for each source
- Replace tag with data as it becomes available
- Issue instruction when all sources are available
- Save dest data when operation finishes
- Commit saved dest data when instruction commits

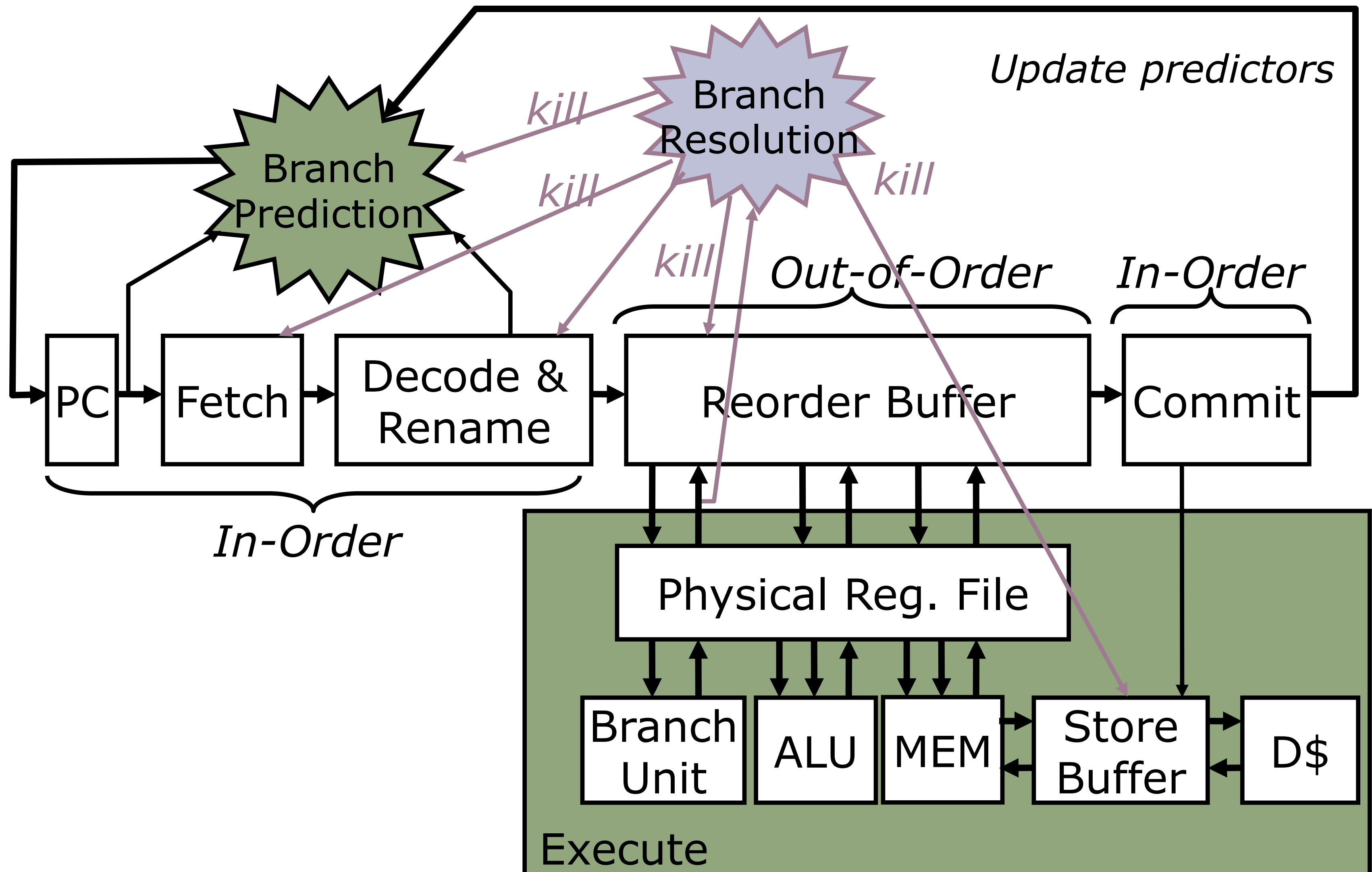
Unified Physical Register File

(MIPS R10K, Alpha 21264, Pentium 4)



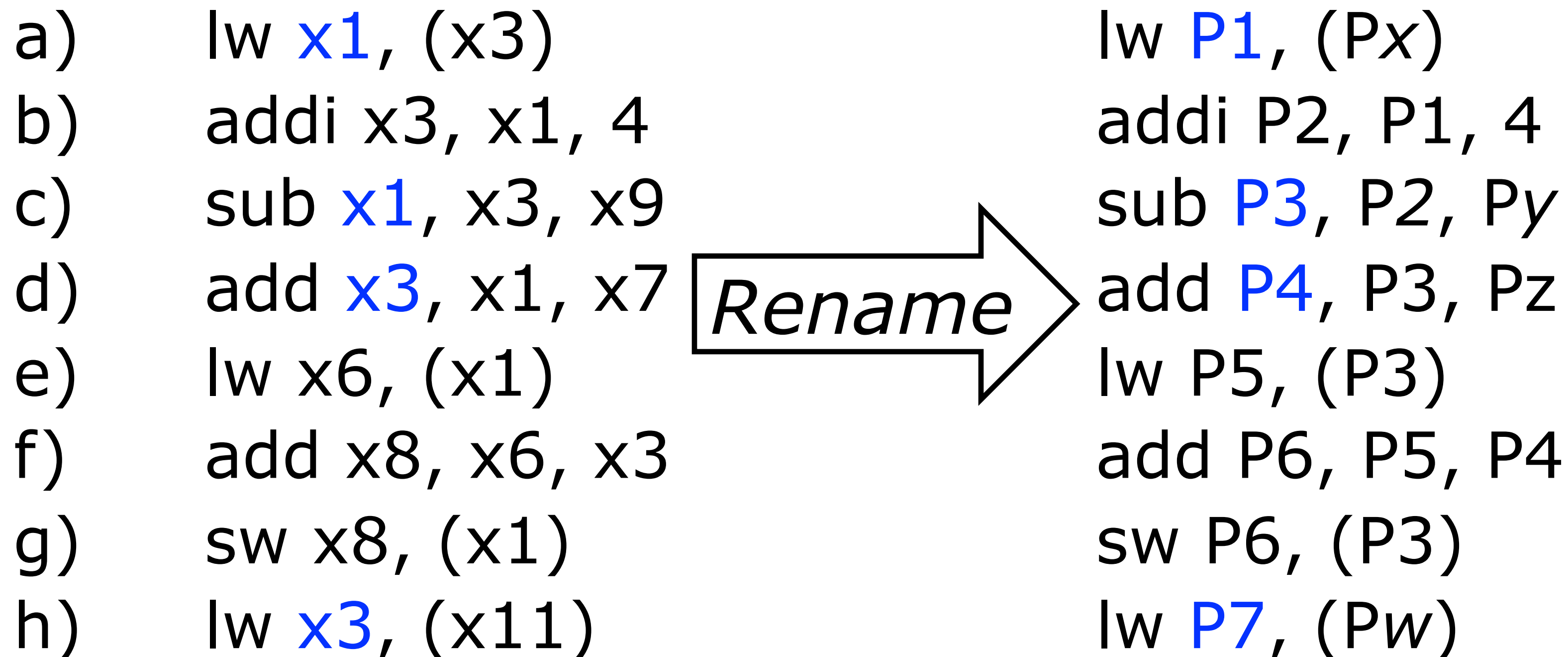
- One regfile for both *committed* and *speculative* values (no data in ROB)
- During decode, instruction result allocated new physical register, source regs translated to physical regs through rename table
- Instruction reads data from regfile at start of execute (not in decode)
- Write-back updates reg. busy bits on instructions in ROB (assoc. search)
- Snapshots of rename table taken at every branch to recover mispredicts
- On exception, renaming undone in reverse order of issue *(MIPS R10000)*

Speculative & Out-of-Order Execution



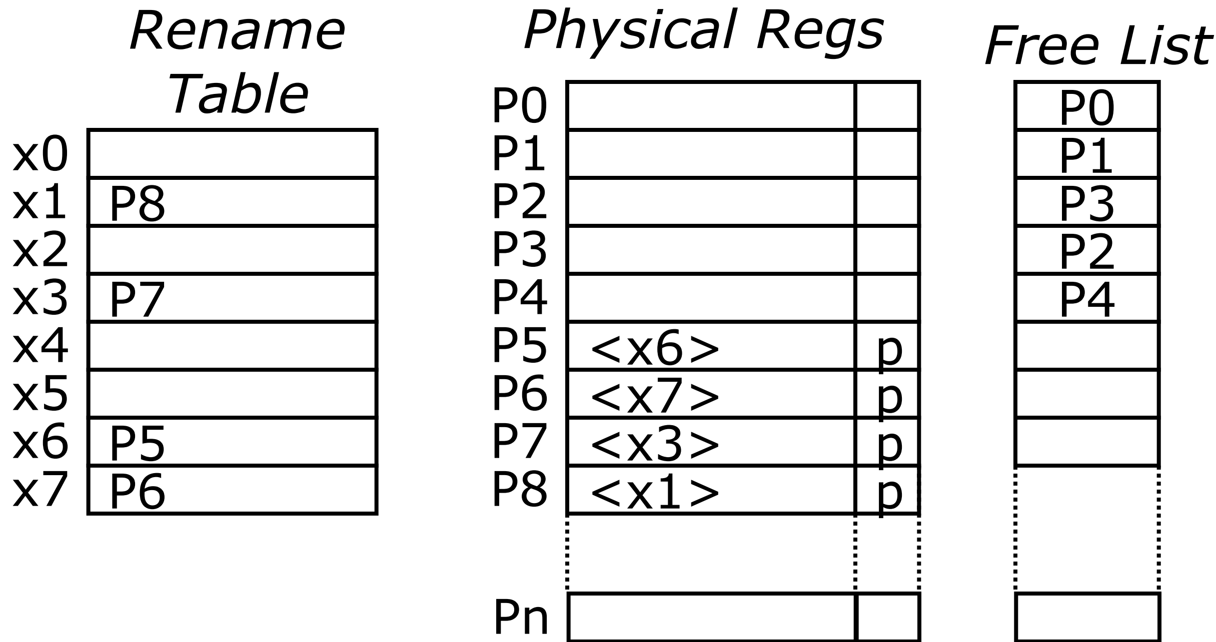
Lifetime of Physical Registers

- Physical regfile holds committed and speculative values
- Physical registers decoupled from ROB entries *(no data in ROB)*



When can we reuse a physical register?

Physical Register Management



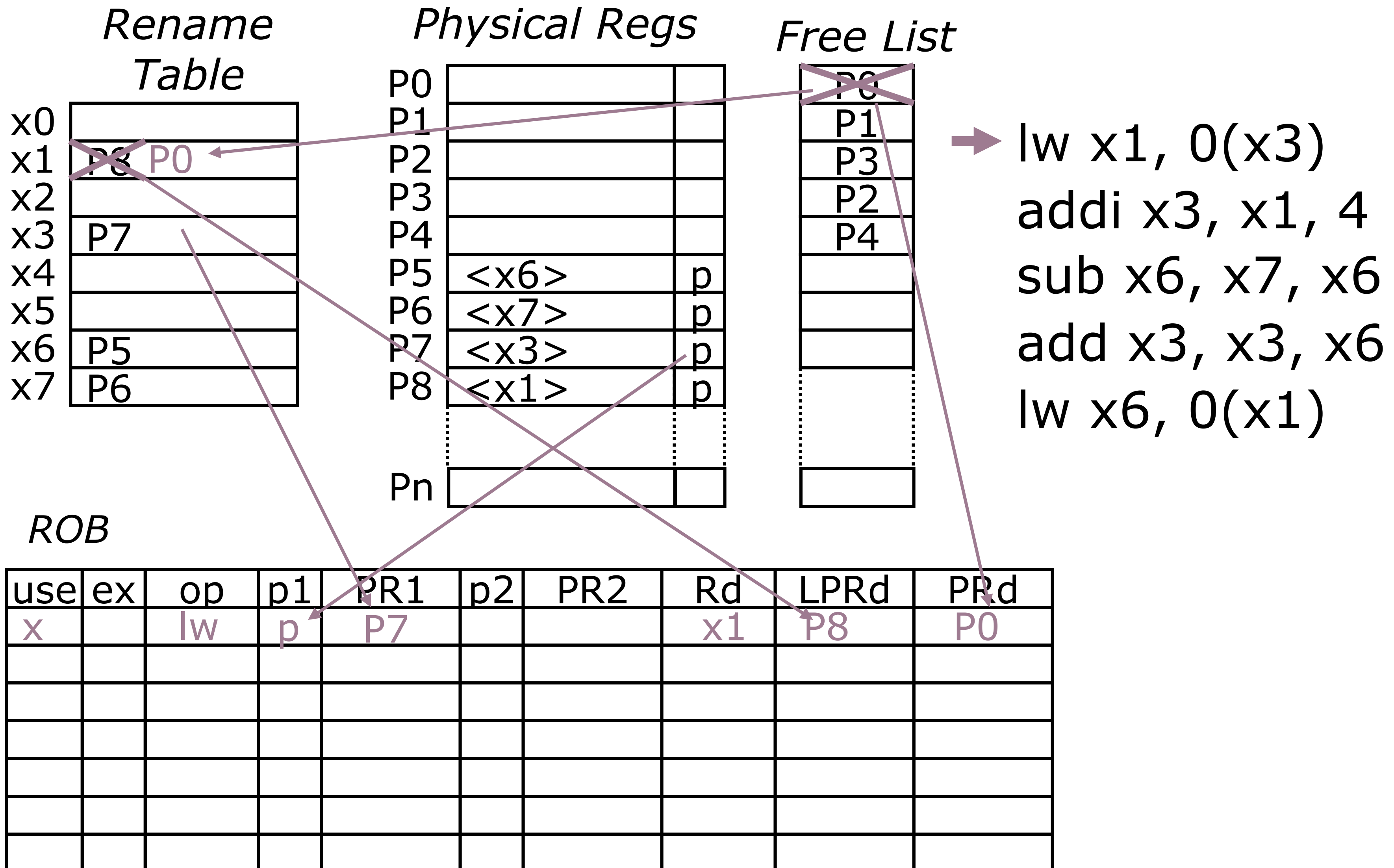

```
lw x1, 0(x3)
addi x3, x1, 4
sub x6, x7, x6
add x3, x3, x6
lw x6, 0(x1)
```

ROB

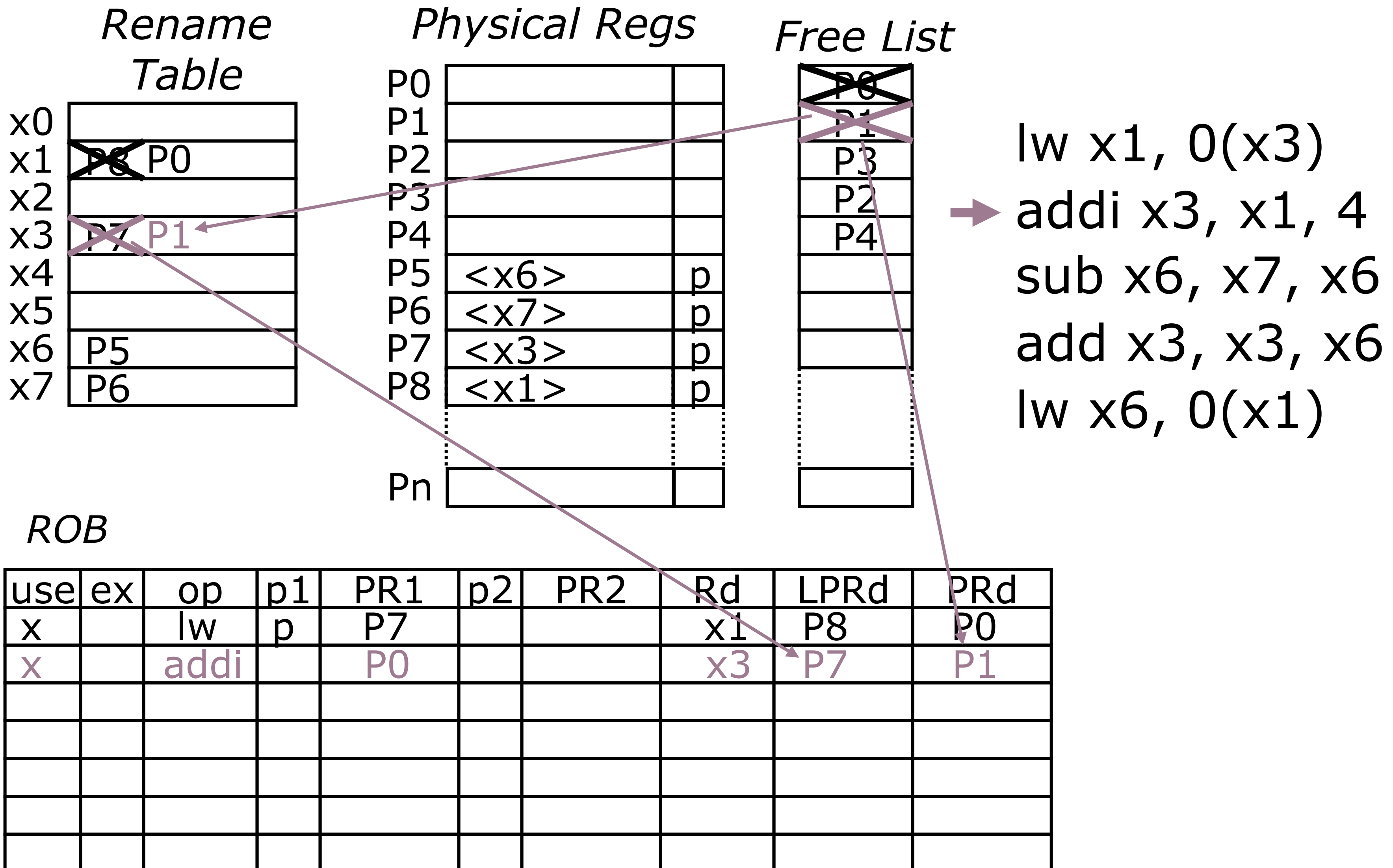
use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd

(LPRd requires third read port on Rename Table for each instruction)

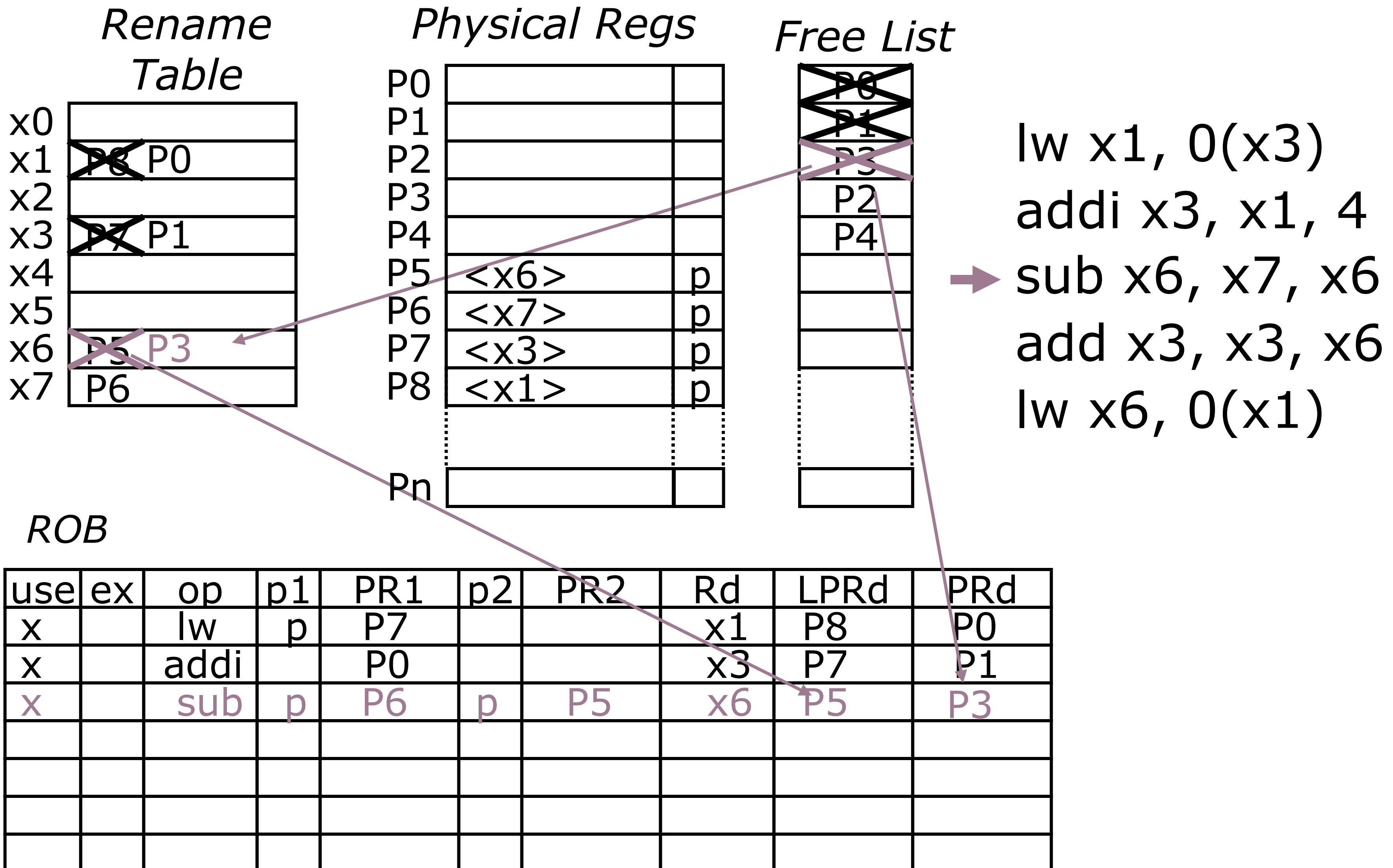
Physical Register Management



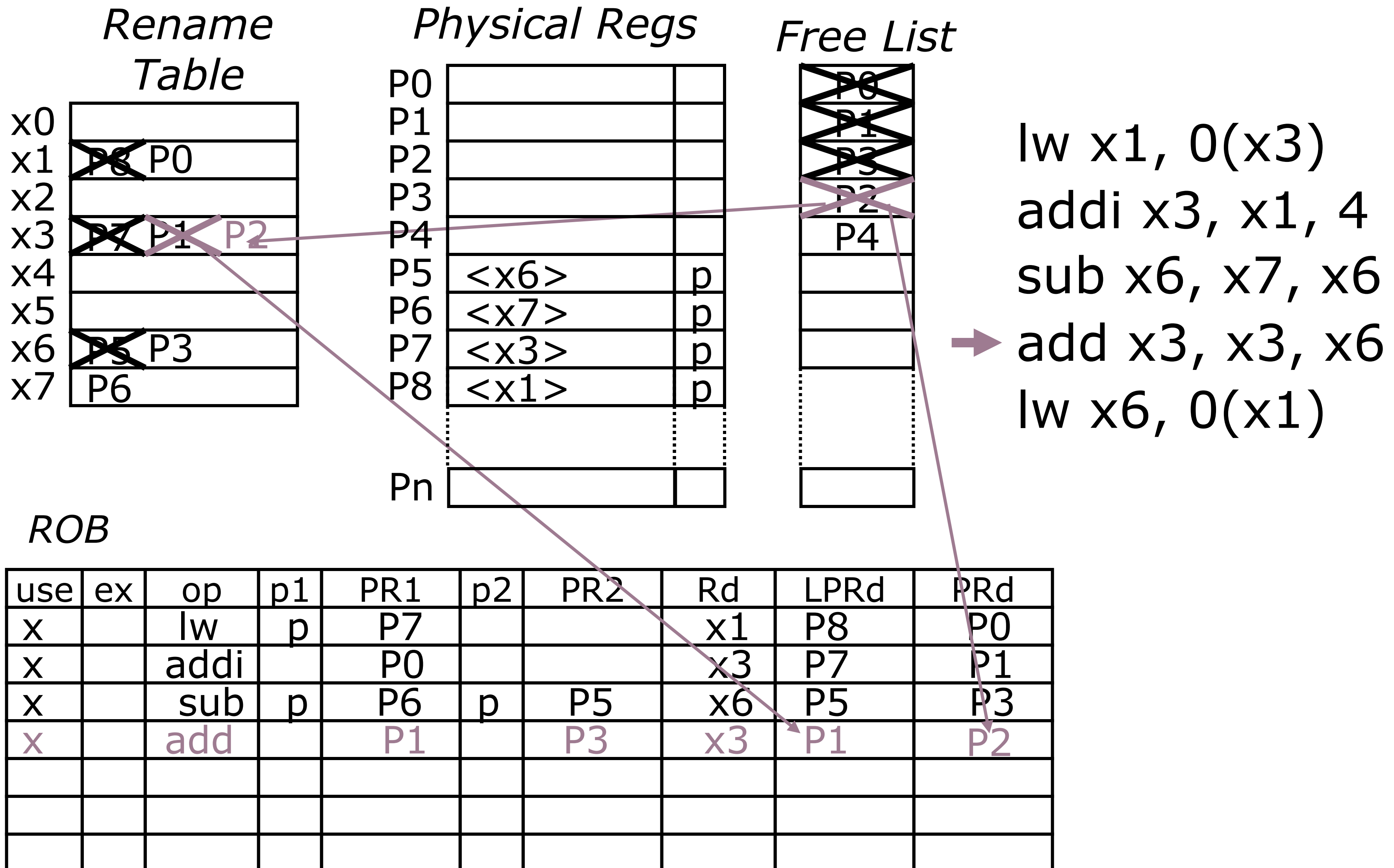
Physical Register Management



Physical Register Management

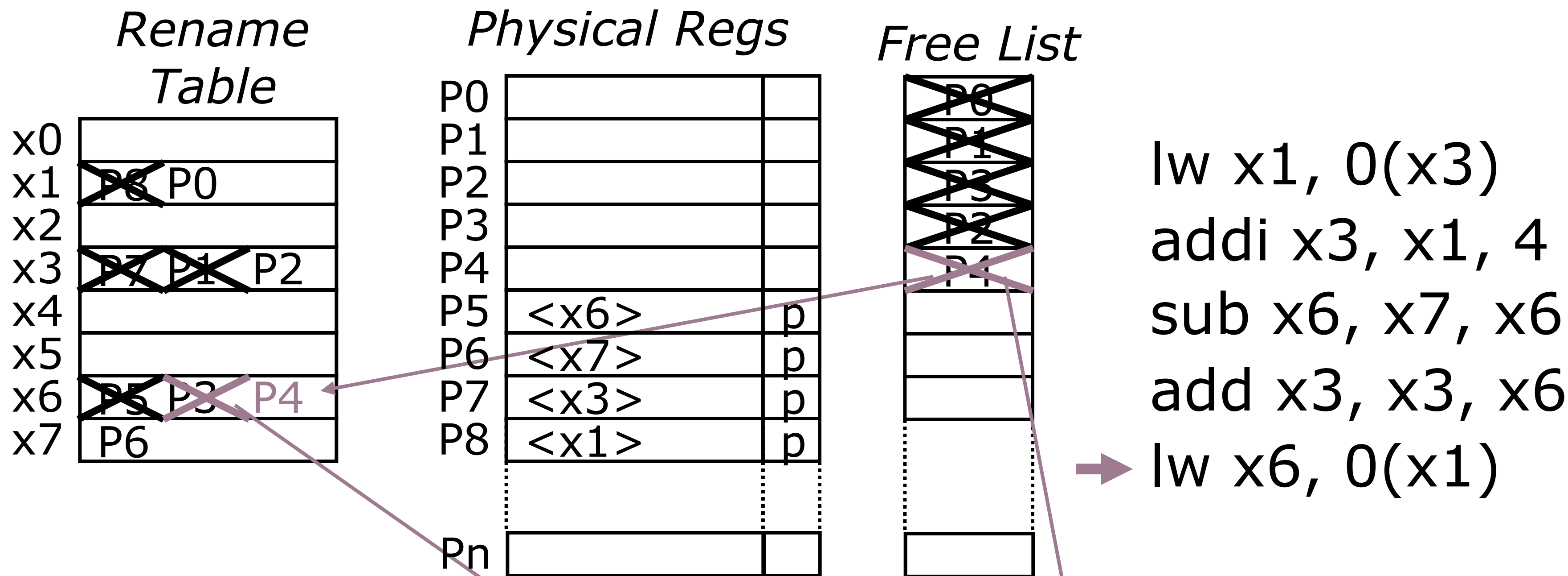


Physical Register Management



use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		lw	p	P7			x1	P8	P0
x		addi		P0			x3	P7	P1
x		sub	p	P6	p	P5	x6	P5	P3
x		add		P1		P3	x3	P1	P2

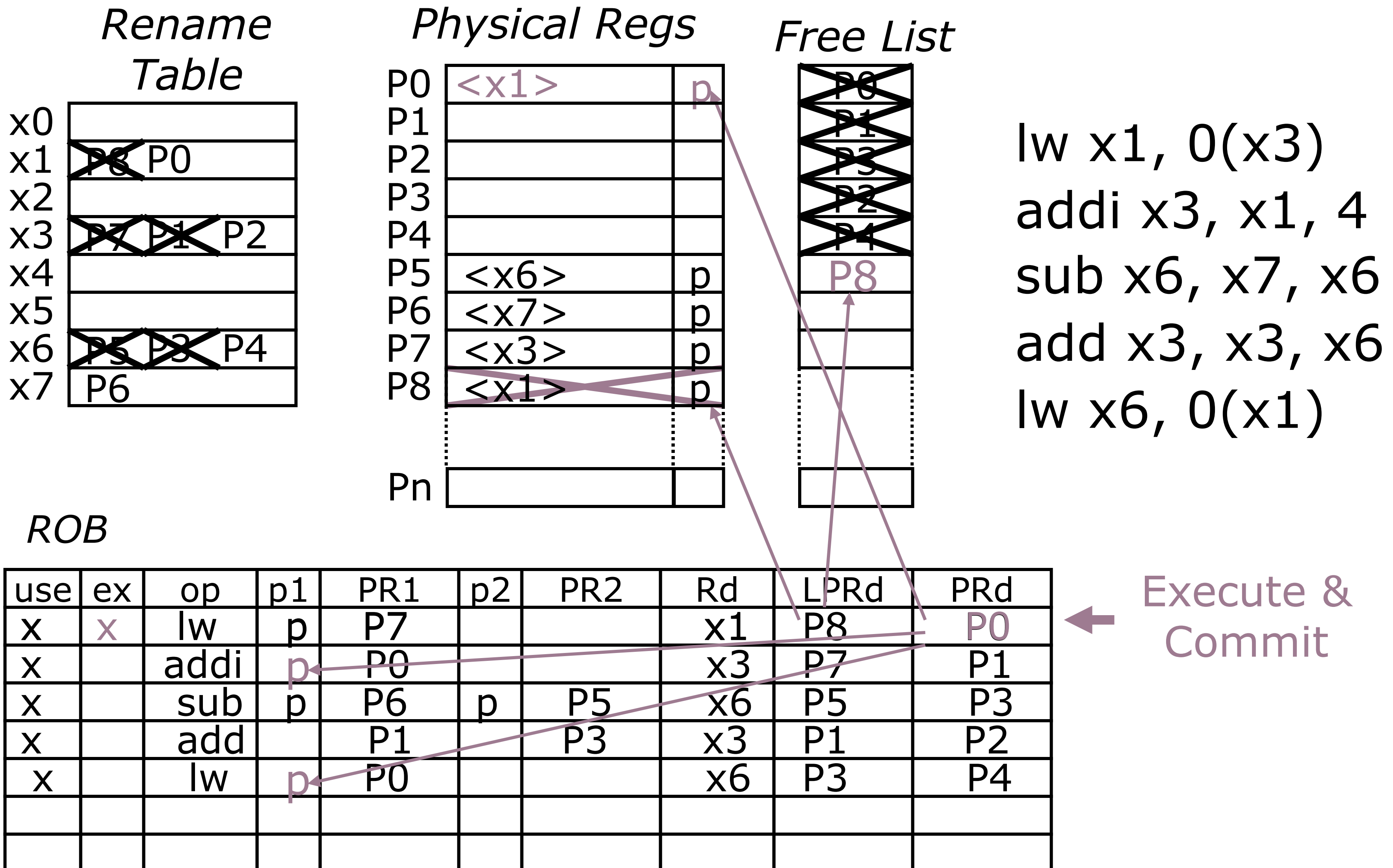
Physical Register Management



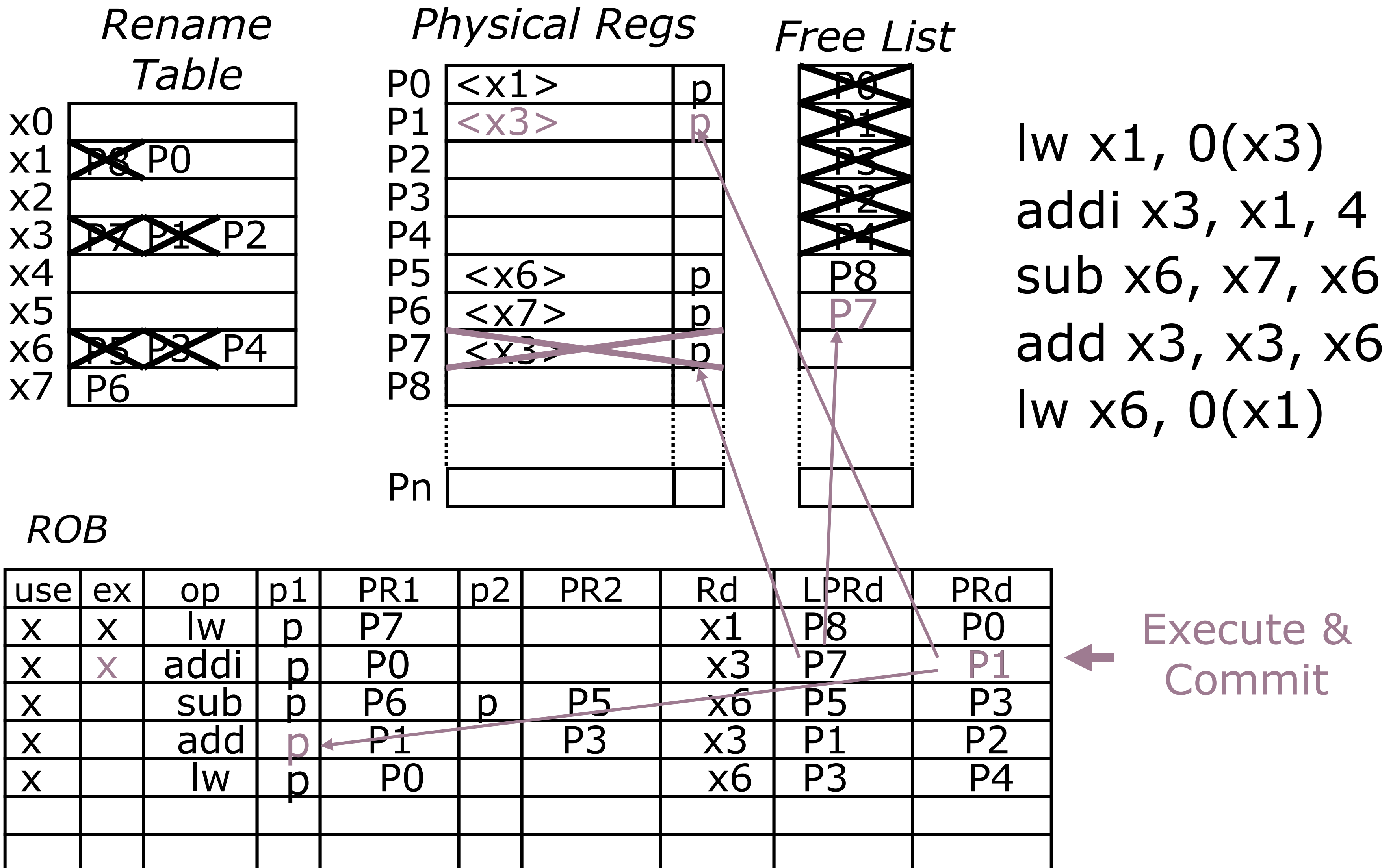
ROB

use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x		lw	p	P7			x1	P8	P0
x		addi		P0			x3	P7	P1
x		sub	p	P6	p	P5	x6	P5	P3
x		add		P1		P3	x3	P1	P2
x		lw		P0			x6	P3	P4

Physical Register Management

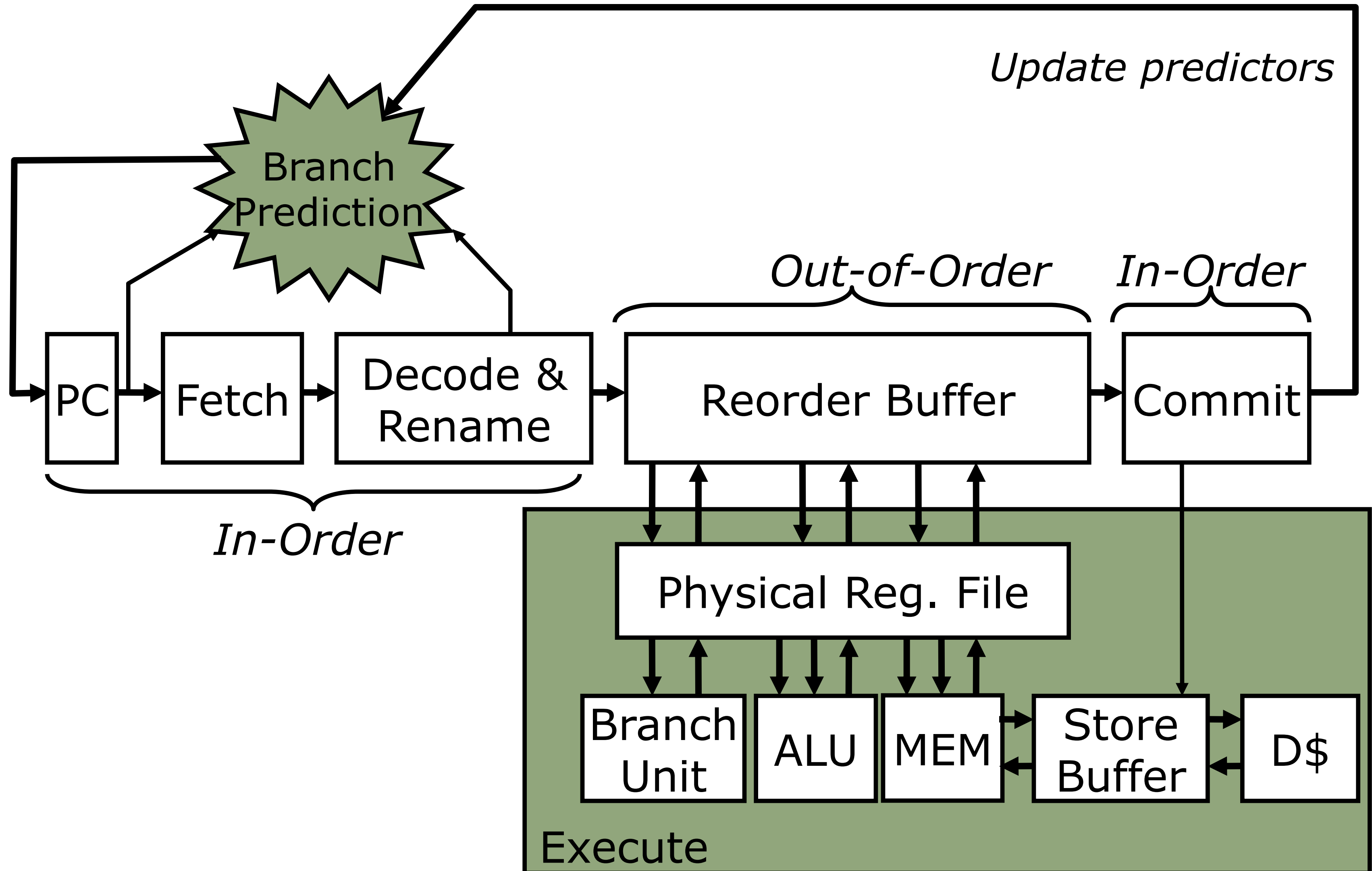


Physical Register Management

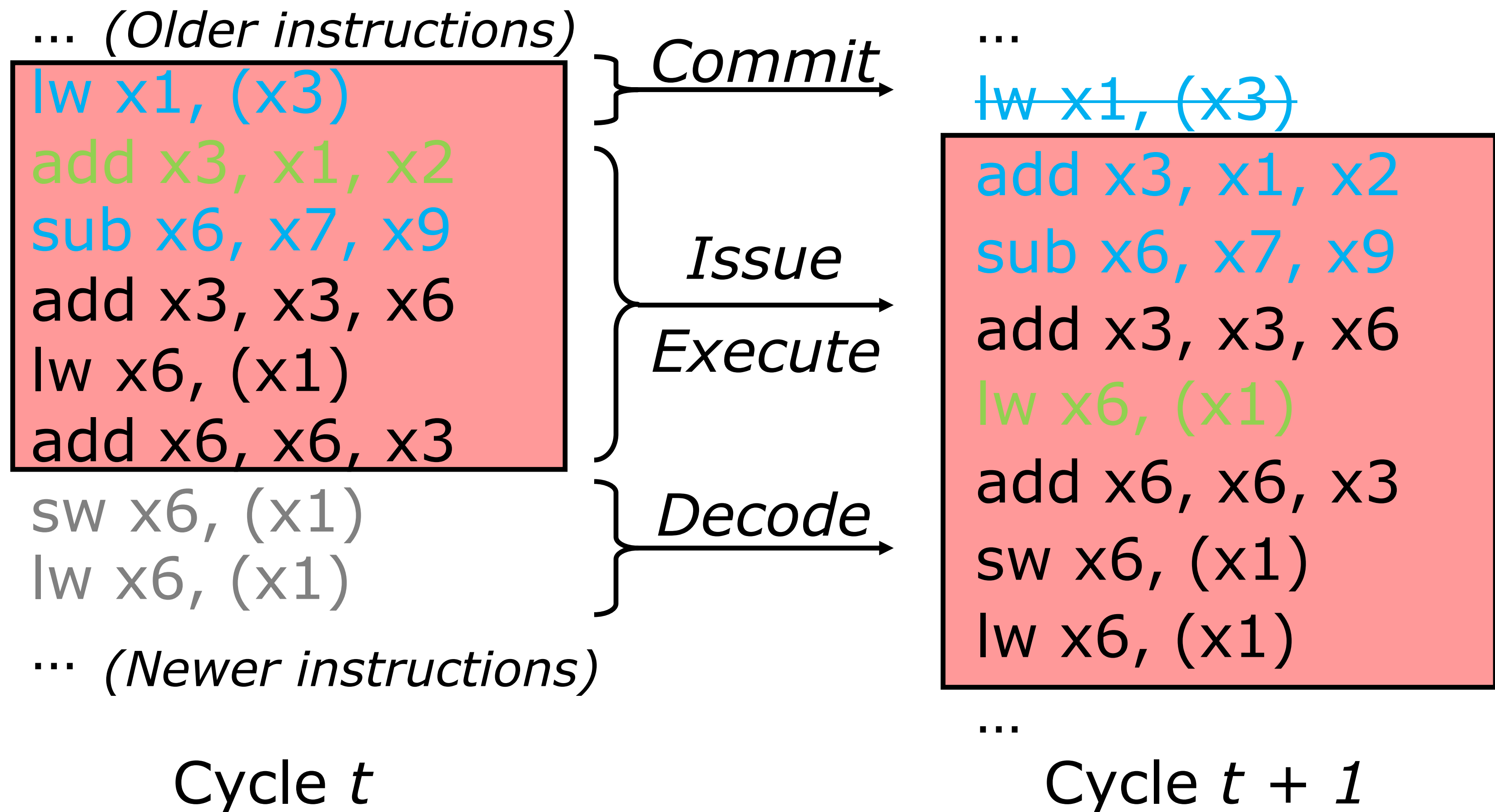


use	ex	op	p1	PR1	p2	PR2	Rd	LPRd	PRd
x	x	lw	p	P7			x1	P8	P0
x	x	addi	p	P0			x3	P7	P1
x		sub	p	P6	p	P5	x6	P5	P3
x		add	p	P1		P3	x3	P1	P2
x		lw	p	P0			x6	P3	P4

Speculative & Out-of-Order Execution



Reorder Buffer Holds Active Instruction Window



Key: predecode, decoded, issued, executed, ~~commited~~

Split Issue and Commit Queues

- 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

Issue Timing

i1	addi x1,x1,1	$Issue_1$	$Execute_1$		
i2	ori x1,x1,1			$Issue_2$	$Execute_2$

How can we issue earlier?

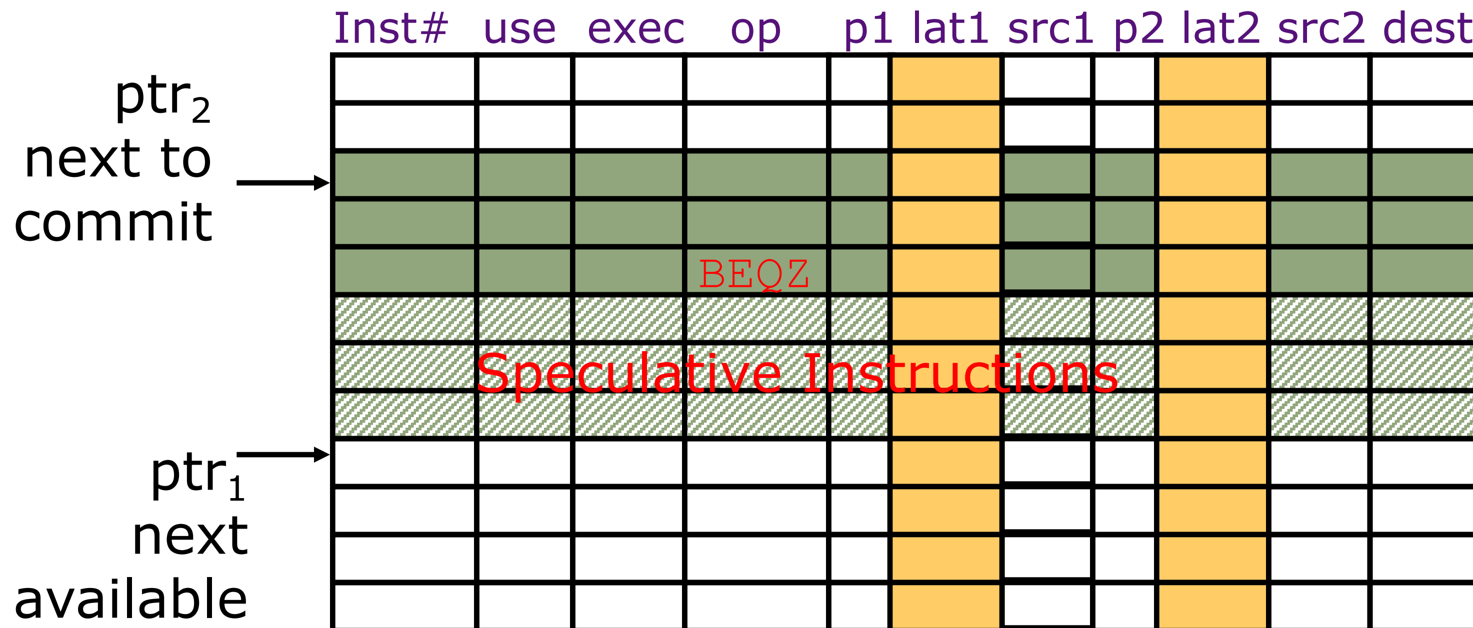
Using knowledge of execution latency (bypass)

i1	lw x1, (x3)	$Issue_1$	$Execute_1$		
i2	addi x1,x1,1		$Issue_2$	$Execute_2$	

What might make this schedule fail?

If execution latency wasn't as expected

Issue Queue with latency prediction

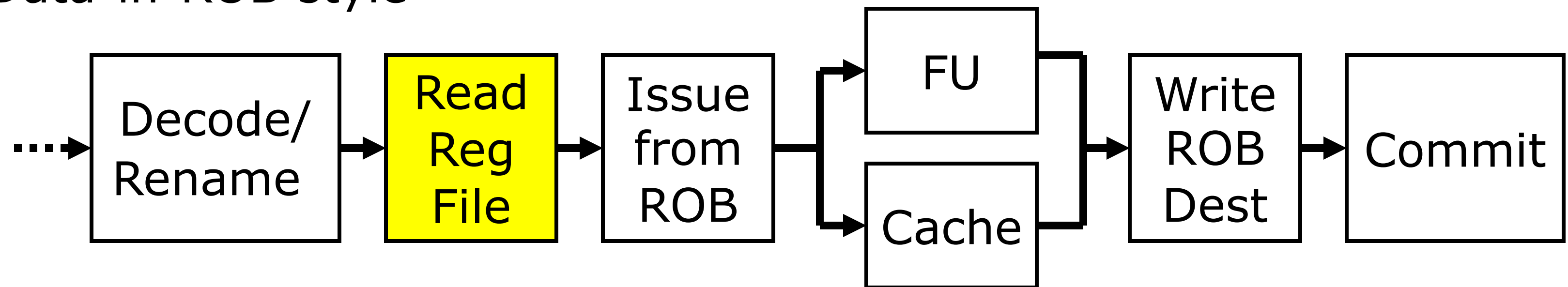


Issue Queue (Reorder buffer)

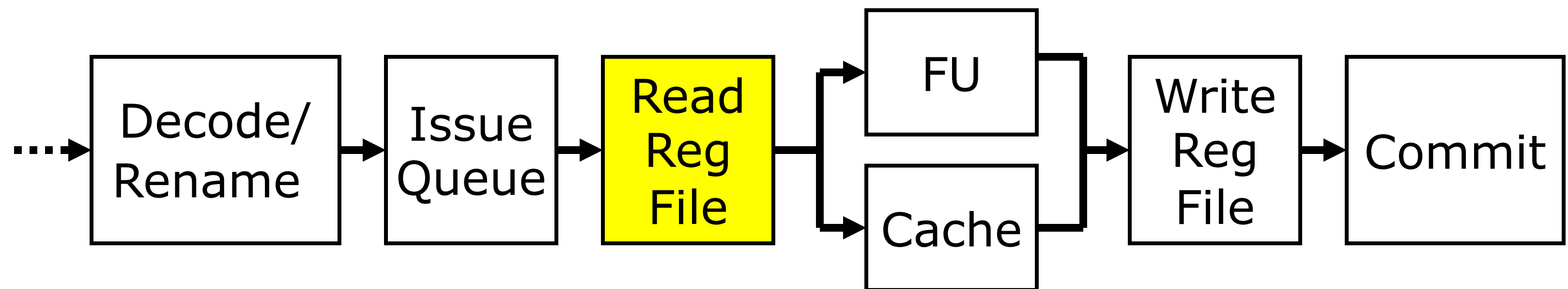
- Fixed latency: latency included in queue entry (‘bypassed’)
- Predicted latency: latency included in queue entry (speculated)
- Variable latency: wait for completion signal (stall)

Data-in-ROB vs. Unified RegFile

Data-in-ROB style



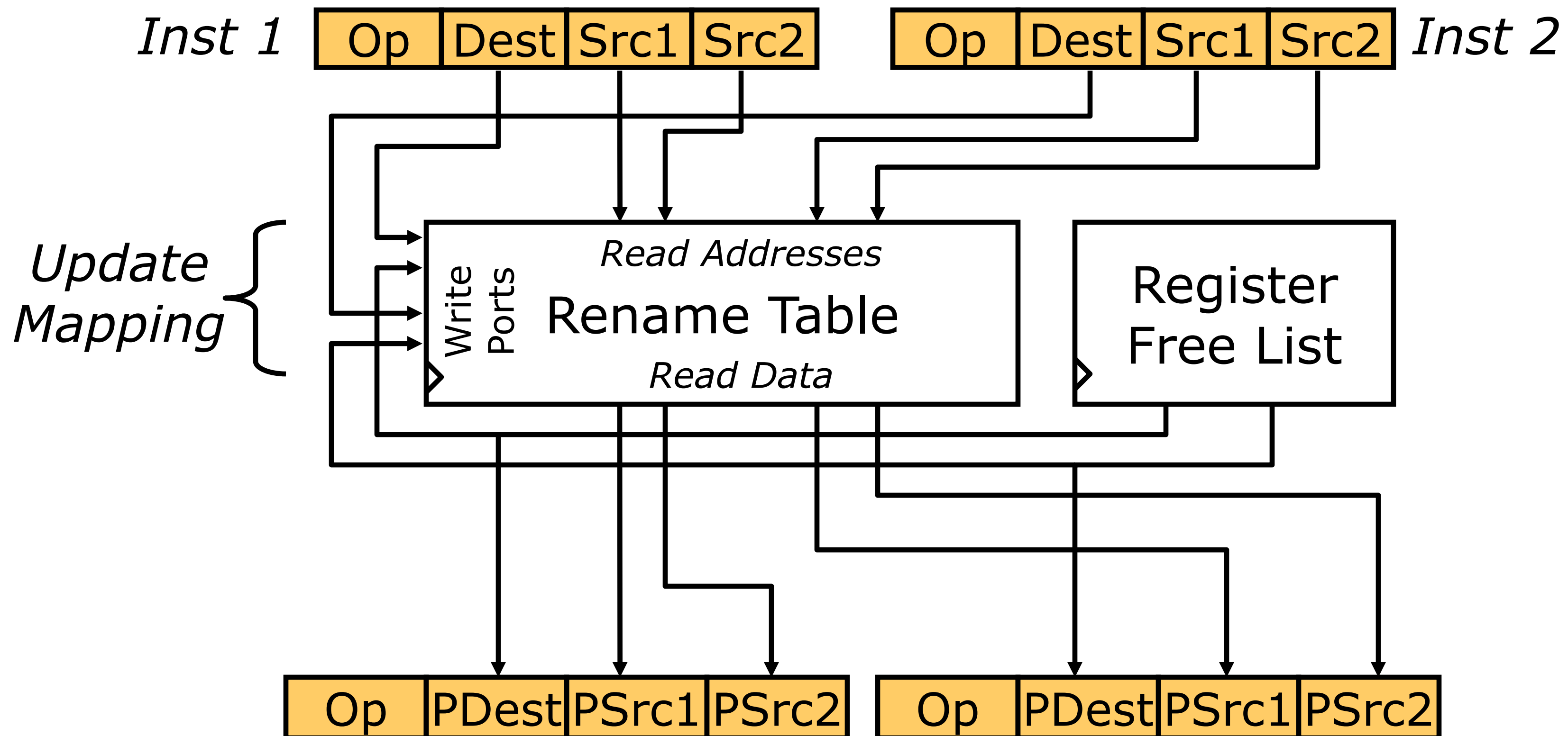
Unified-register-file style



How does issue speculation differ, e.g., on cache miss?

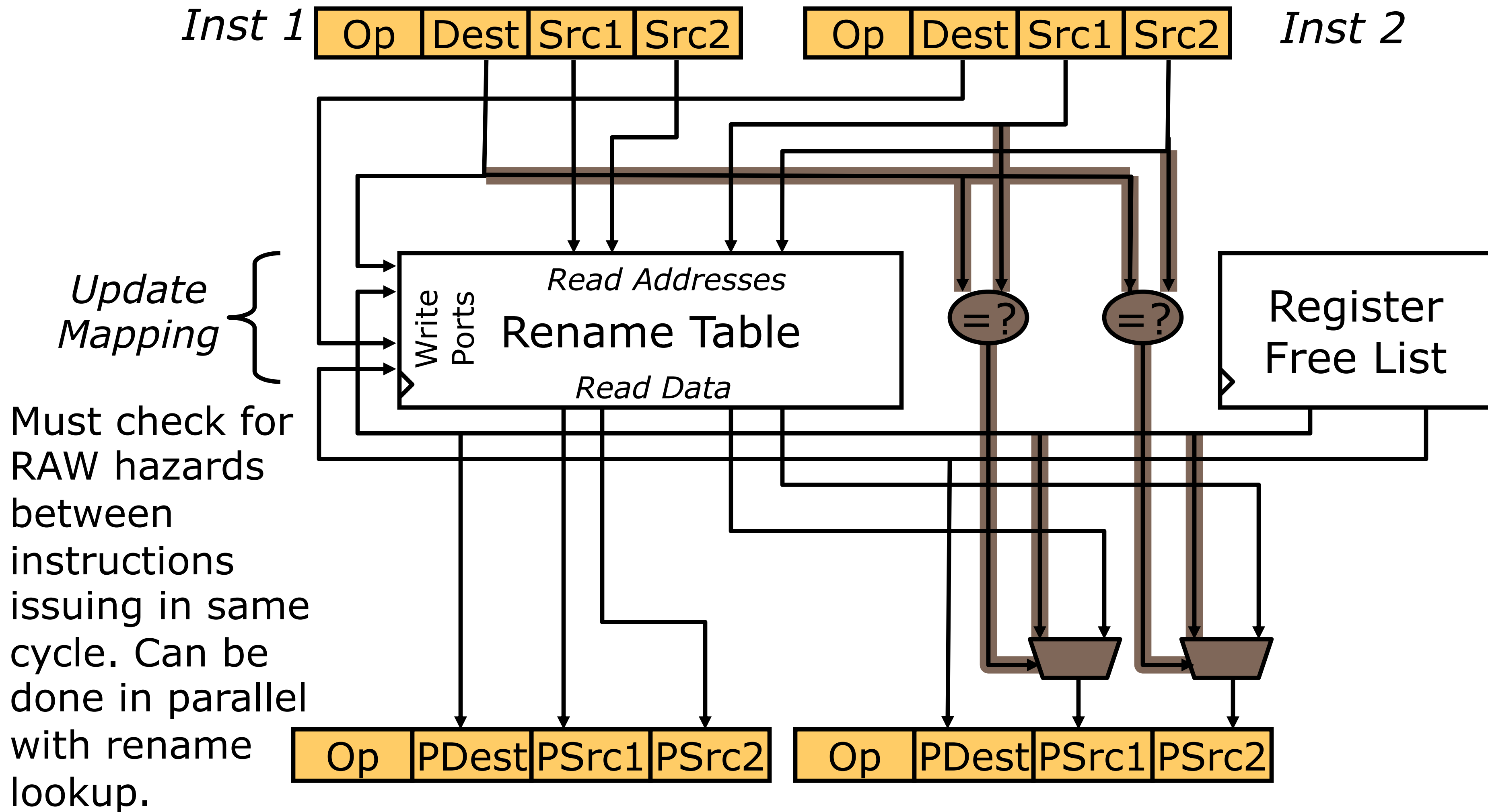
Superscalar Register Renaming

- 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)

Thank you!