Register Read

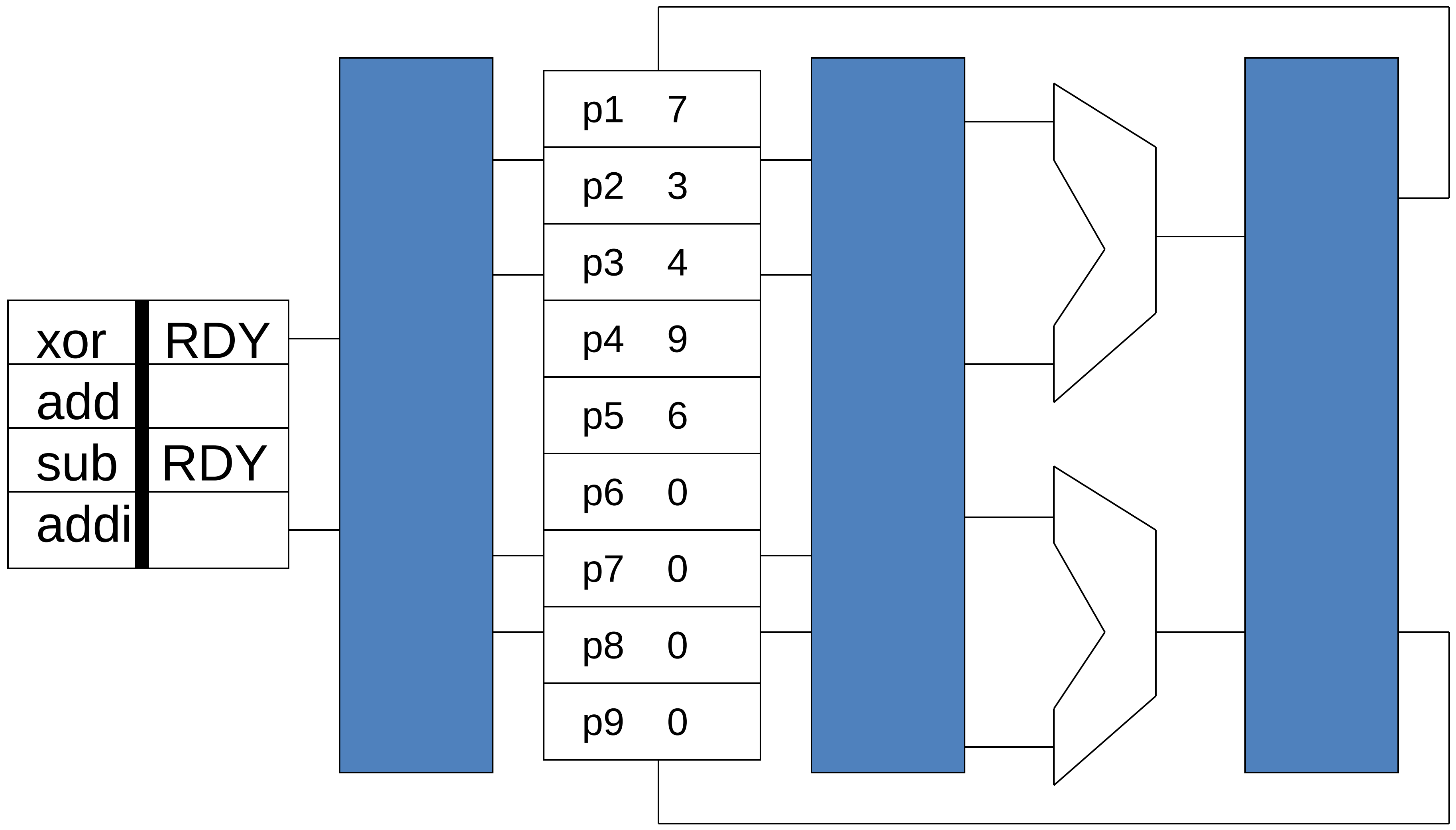
- When do instructions read the register file?

- Option #1: after issue, right before execute
  - (Not done at decode)
  - Read **physical** register (renamed)
  - Or get value via bypassing (based on physical register name)
  - This is Pentium 4, MIPS R10k, Alpha 21264 style

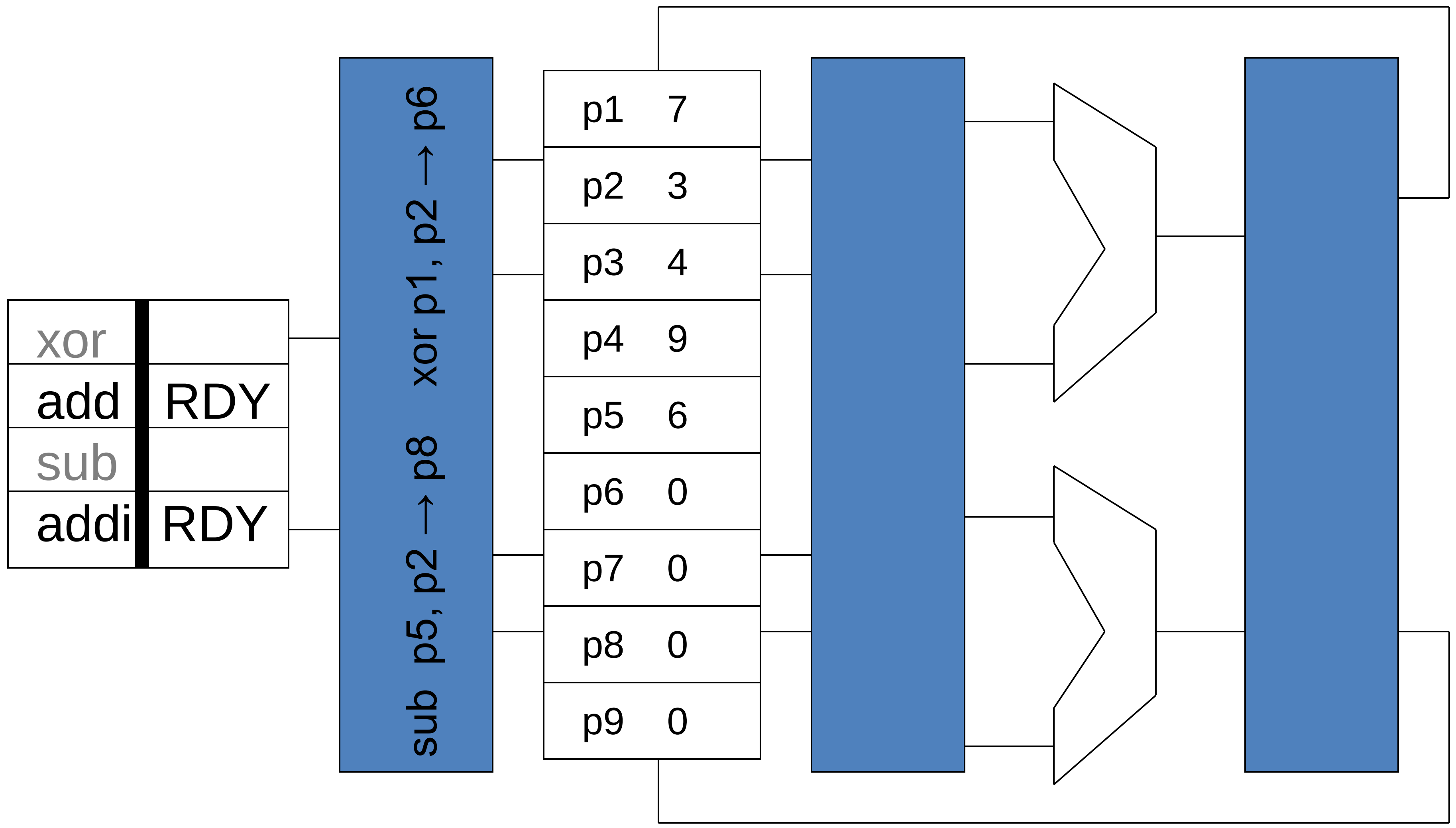
- Physical register file may be large
  - Multi-cycle read

- Option #2: as part of dispatch, keep values in Issue Queue
  - Pentium Pro, Core 2, Core i7

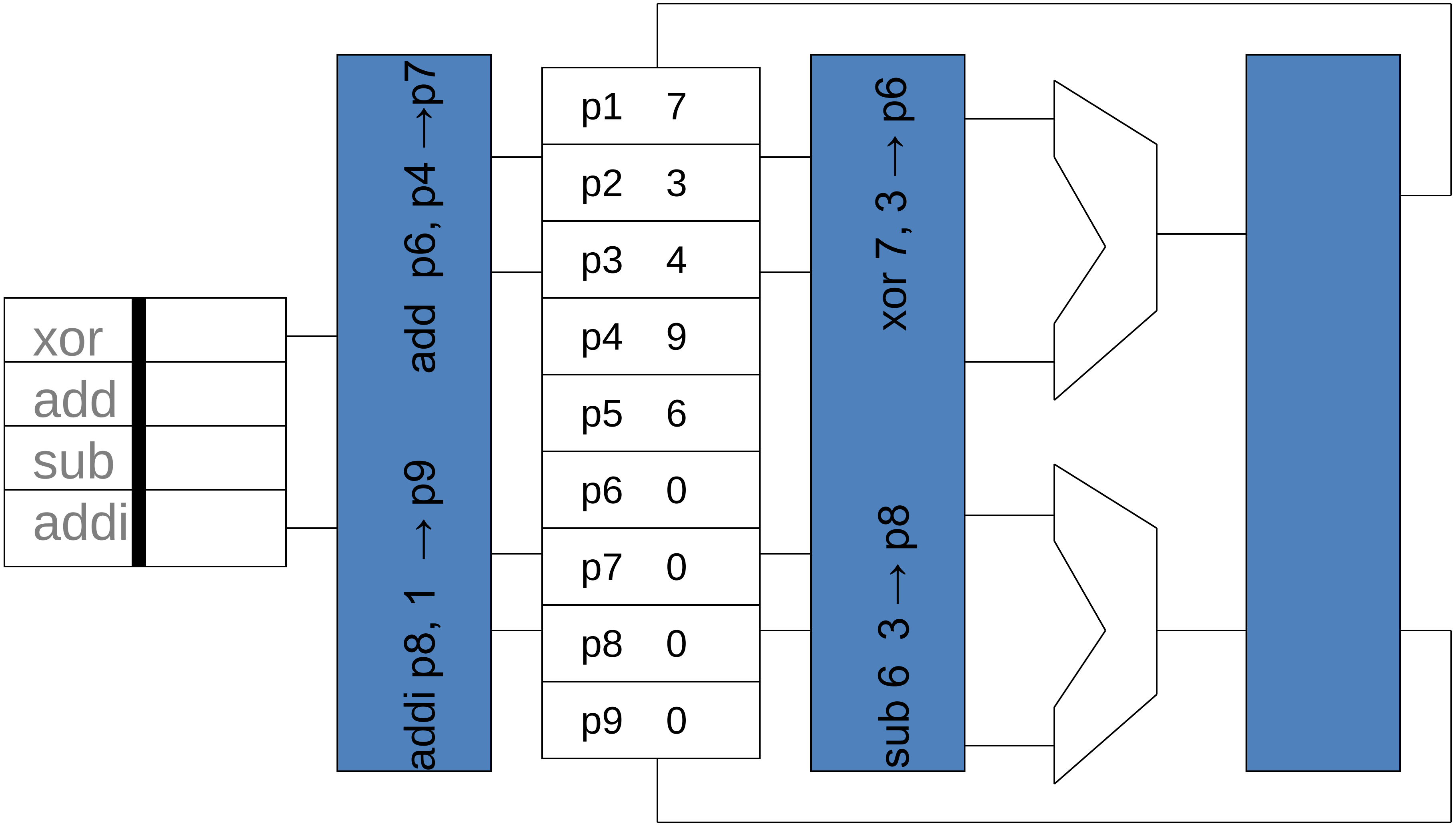
OOO execution (2-wide)



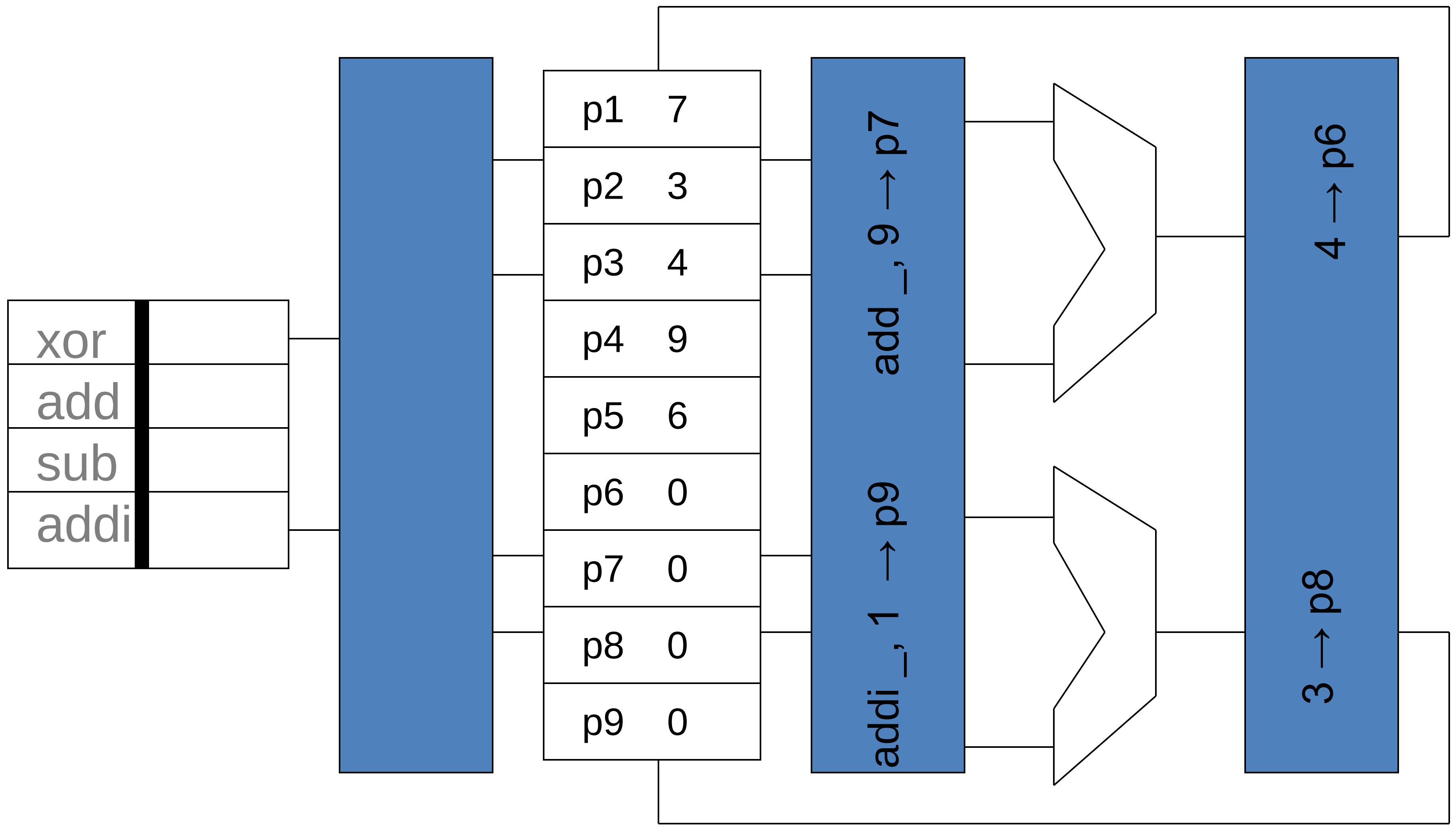
OOO execution (2-wide)



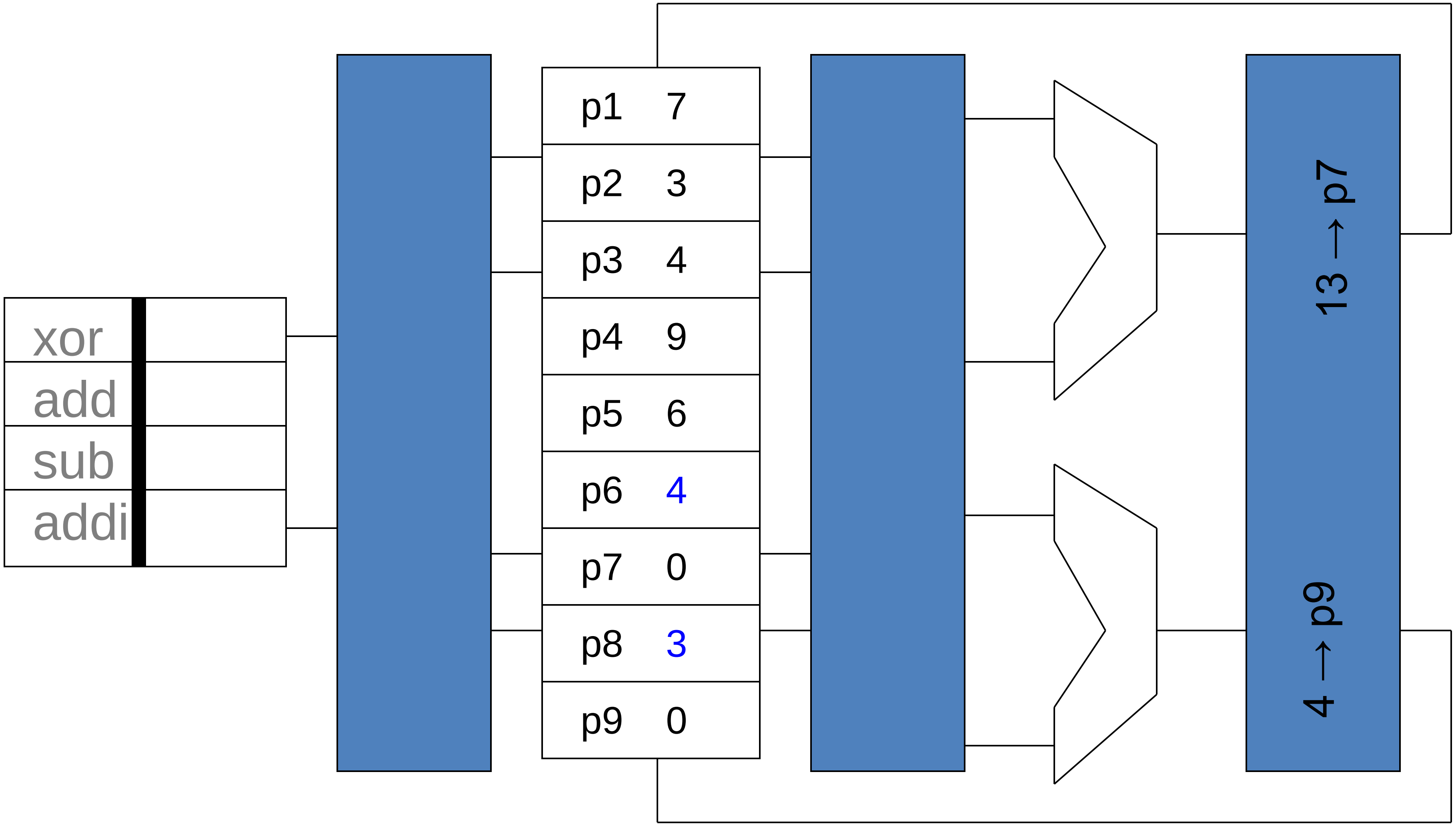
OOO execution (2-wide)



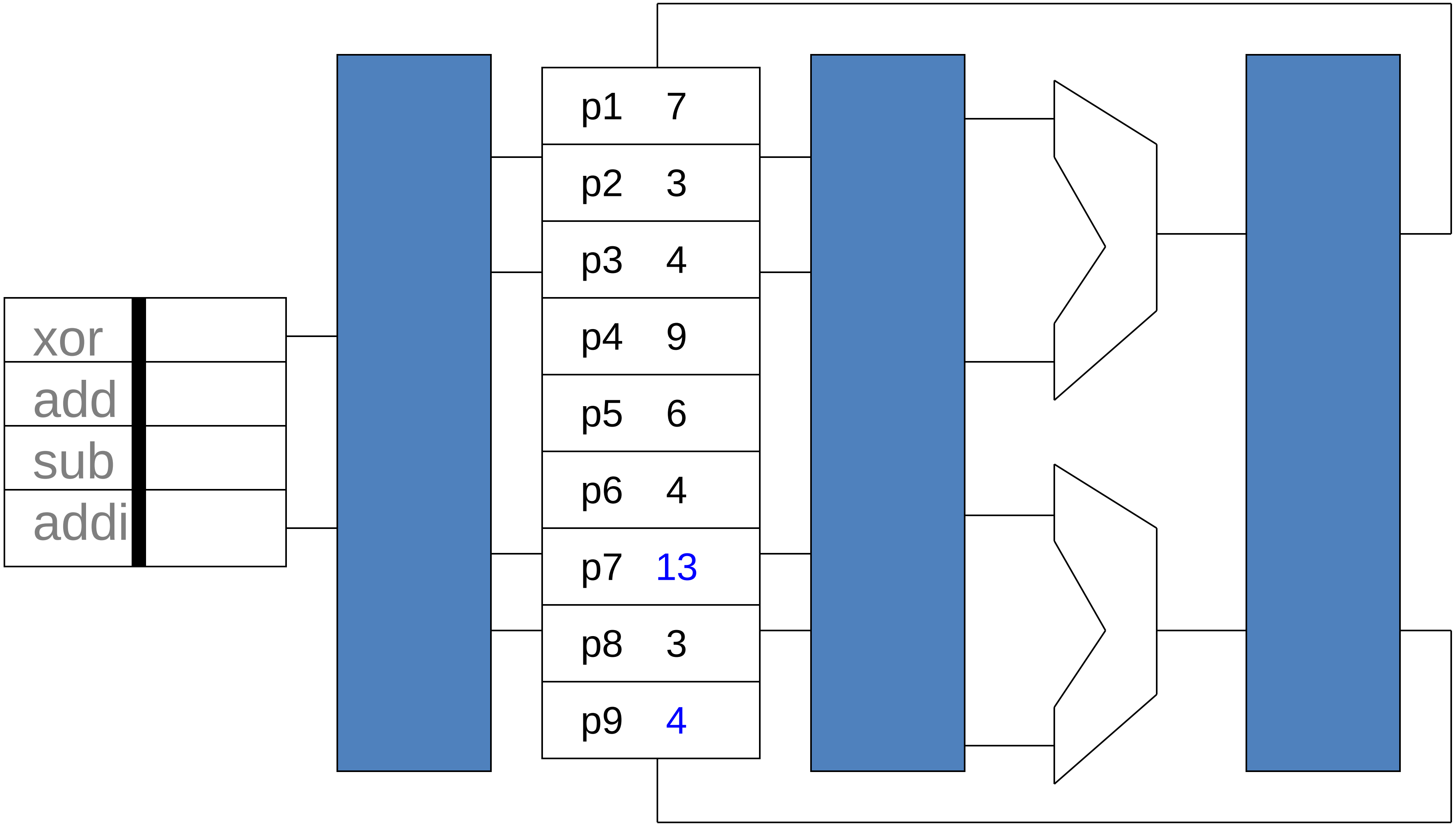
OOO execution (2-wide)



OOO execution (2-wide)

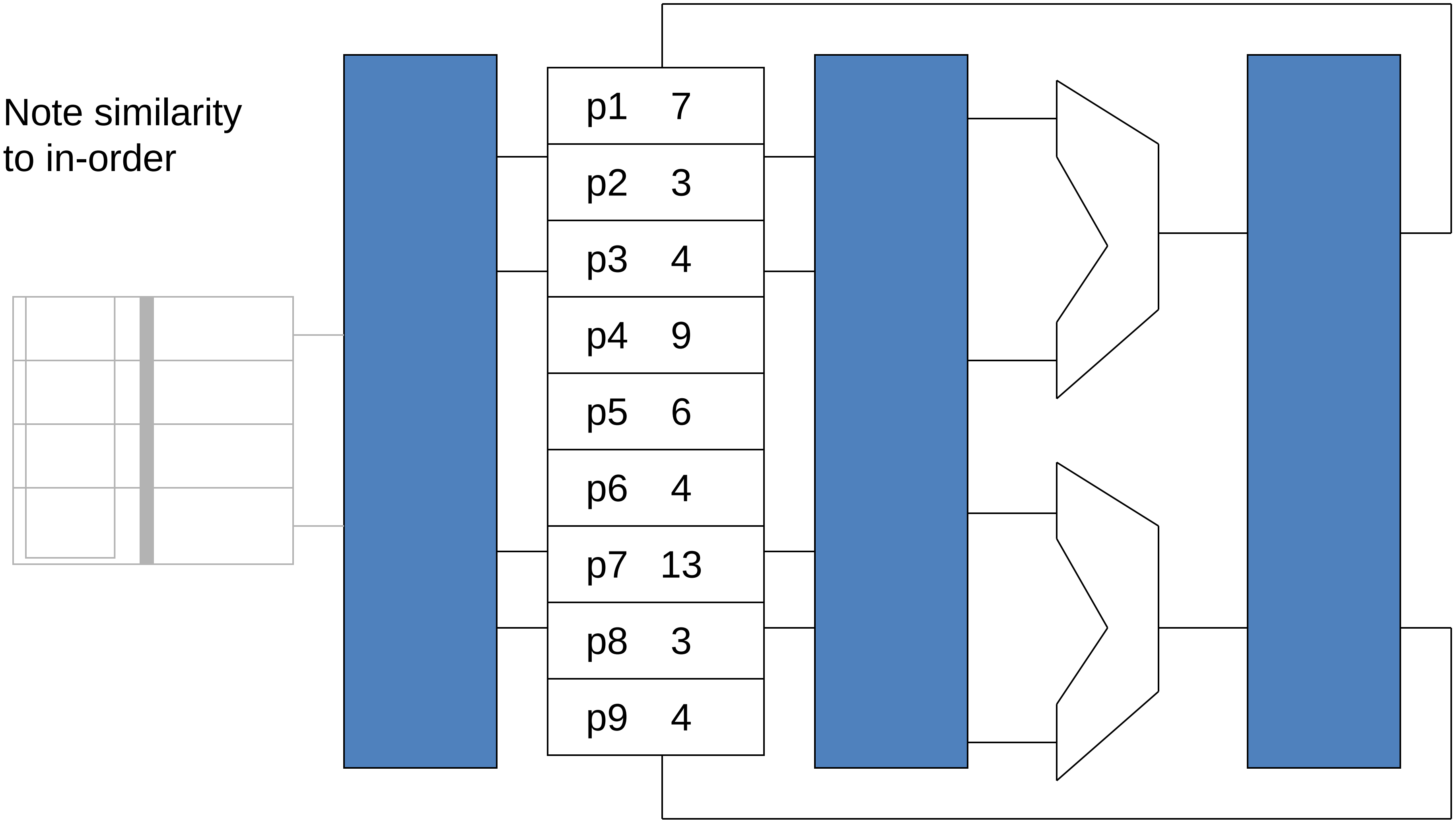


OOO execution (2-wide)



OOO execution (2-wide)

Note similarity to in-order



Multi-cycle operations

- Multi-cycle ops (load, fp, multiply, *etc.*)
  - Wakeup deferred a few cycles
    - Structural hazard?
- Cache misses?
  - Speculative wake-up (assume hit)
  - Cancel exec of dependents
  - Re-issue later
  - Details: complicated, not important

Re-order Buffer (ROB)

- All instructions in order
- Two purposes
    - Misprediction recovery
    - In-order commit
        - Maintain appearance of in-order execution
        - Freeing of physical registers

RENAMING REVISITED

Renaming revisited

- Overwritten register
  - Freed at commit
  - Restore in map table on recovery
    - Branch mis-prediction recovery
  - Also must be read at rename

Renaming example

Original insns

```
xor  r1,r2 → r3
add  r3,r4 → r4
sub  r5,r2 → r3
addi r3,1  → r1
```

r1	p1
r2	p2
r3	p3
r4	p4
r5	p5

Map table

p6
p7
p8
p9
p10

Free-list

Renaming example

<u>Original insns</u>

```
xor  r1,r2 → r3
add  r3,r4 → r4
sub  r5,r2 → r3
addi r3,1  → r1
```

$\longrightarrow$

<u>Renamed insns</u>

```
xor  p1, p2 →
```

<u>Overwritten Reg</u>

[p3]

r1	p1
r2	p2
r3	**p3**
r4	p4
r5	p5

Map table

p6
p7
p8
p9
p10

Free-list

Renaming example

Original insns		Renamed insns	Overwritten Reg
`xor  r1,r2 → r3`	→	`xor  p1, p2 → p6`	[p3]
`add  r3,r4 → r4`			
`sub  r5,r2 → r3`			
`addi r3,1  → r1`			

r1	p1
r2	p2
r3	**p6**
r4	p4
r5	p5

Map table

p7
p8
p9
p10

Free-list

Renaming example

Original insns

```
xor  r1,r2 → r3
add  r3,r4 → r4
sub  r5,r2 → r3
addi r3,1  → r1
```

Renamed insns

```
xor  p1, p2 → p6
add  p6, p4 →
```

Overwritten Reg

```
[p3]
[p4]
```

r1	p1
r2	p2
r3	p6
r4	**p4**
r5	p5

Map table

p7
p8
p9
p10

Free-list

Renaming example

<u>Original insns</u>

```
xor  r1,r2 → r3
add  r3,r4 → r4
sub  r5,r2 → r3
addi r3,1  → r1
```

→

<u>Renamed insns</u>

```
xor  p1, p2 → p6
add  p6, p4 → p7
```

<u>Overwritten Reg</u>

```
[p3]
[p4]
```

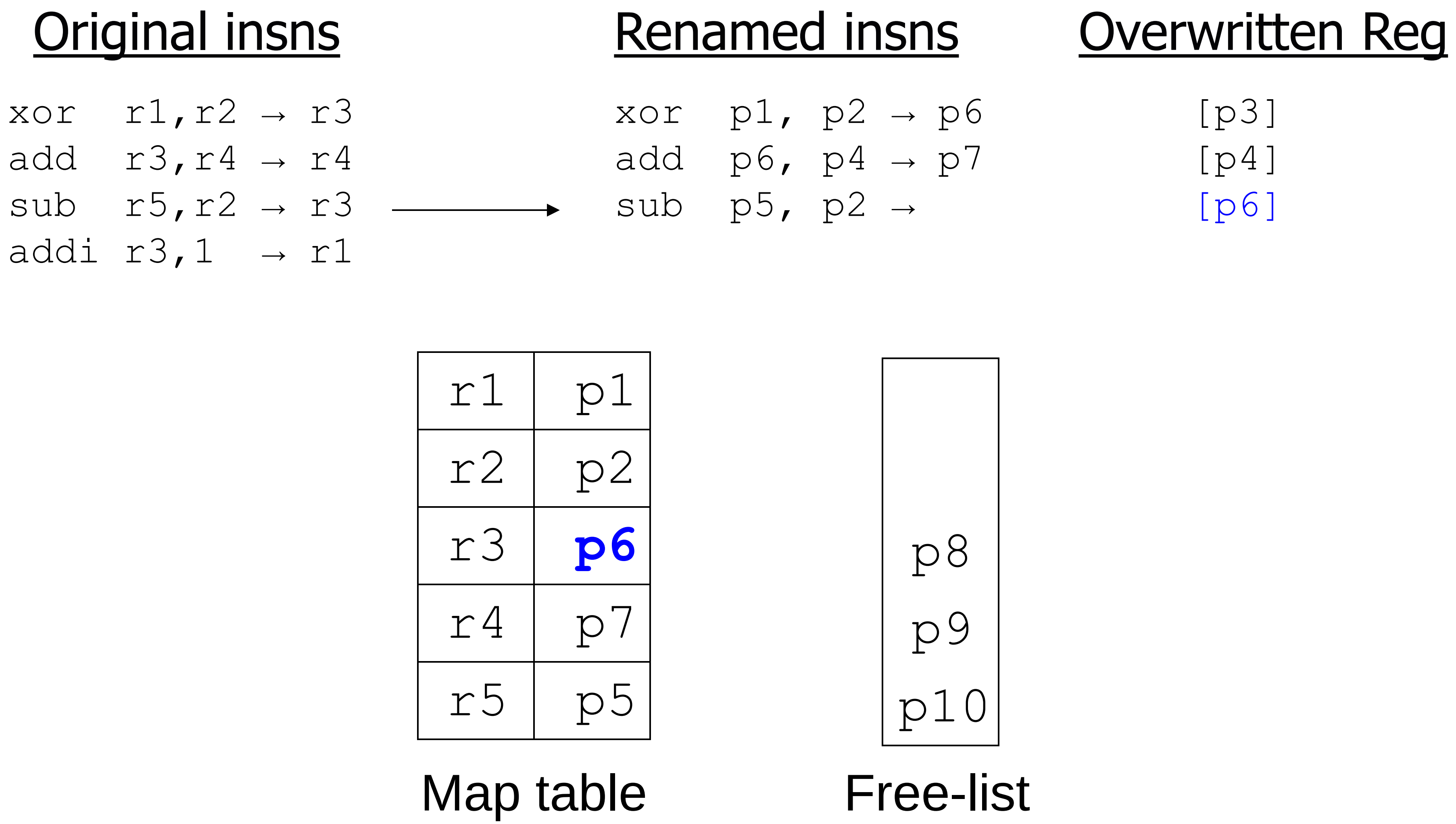
r1	p1
r2	p2
r3	p6
r4	**p7**
r5	p5

Map table

p8
p9
p10

Free-list

Renaming example



Original insns

```
xor  r1,r2 → r3
add  r3,r4 → r4
sub  r5,r2 → r3
addi r3,1  → r1
```

Renamed insns

```
xor  p1, p2 → p6
add  p6, p4 → p7
sub  p5, p2 →
```

Overwritten Reg

```
[p3]
[p4]
[p6]
```

r1	p1
r2	p2
r3	**p6**
r4	p7
r5	p5

Map table

p8
p9
p10

Free-list

Renaming example

Original insns		Renamed insns	Overwritten Reg
`xor  r1,r2 → r3`		`xor  p1, p2 → p6`	`[p3]`
`add  r3,r4 → r4`		`add  p6, p4 → p7`	`[p4]`
`sub  r5,r2 → r3`	⟶	`sub  p5, p2 → p8`	`[p6]`
`addi r3,1  → r1`			

r1	p1
r2	p2
r3	**p8**
r4	p7
r5	p5

Map table

p9
p10

Free-list

Renaming example

Original insns	Renamed insns	Overwritten Reg
`xor  r1,r2 → r3`	`xor  p1, p2 → p6`	[p3]
`add  r3,r4 → r4`	`add  p6, p4 → p7`	[p4]
`sub  r5,r2 → r3`	`sub  p5, p2 → p8`	[p6]
`addi r3,1  → r1`	`addi p8, 1 →`	[p1]

r1	**p1**
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p9
p10

Free-list

Renaming example

Original insns	Renamed insns	Overwritten Reg
`xor  r1,r2 → r3`	`xor  p1, p2 → p6`	[p3]
`add  r3,r4 → r4`	`add  p6, p4 → p7`	[p4]
`sub  r5,r2 → r3`	`sub  p5, p2 → p8`	[p6]
`addi r3,1  → r1`	`addi p8, 1  → p9`	[p1]

r1	**p9**
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p10

Free-list

ROB

- ROB entry holds all info for recover/commit
  - Logical register names
  - Physical register names
  - Instruction types
- Dispatch: insert at tail
  - Full? Stall
- Commit: remove from head
  - Not completed? Stall

Recovery

- Completely remove wrong path instructions
  - Flush from IQ
  - Remove from ROB
  - Restore map table to before misprediction
  - Free destination registers

Recovery example

Original insns	Renamed insns	Overwritten Reg
`bnz r1 loop`	`bnz p1, loop`	[ ]
`xor r1, r2 → r3`	`xor p1, p2 → p6`	[p3]
`add r3, r4 → r4`	`add p6, p4 → p7`	[p4]
`sub r5, r2 → r3`	`sub p5, p2 → p8`	[p6]
`addi r3, 1 → r1`	`addi p8, 1 → p9`	[p1]

r1	p9
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p10

Free-list

Recovery example

Original insns	Renamed insns	Overwritten Reg
`bnz r1 loop`	`bnz p1, loop`	[ ]
`xor r1, r2 → r3`	`xor p1, p2 → p6`	[p3]
`add r3, r4 → r4`	`add p6, p4 → p7`	[p4]
`sub r5, r2 → r3`	`sub p5, p2 → p8`	[p6]
`addi r3, 1 → r1`	`addi p8, 1 → p9`	[p1]

r1	**p1**
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p9
p10

Free-list

Recovery example

Original insns	Renamed insns	Overwritten Reg
`bnz r1 loop`	`bnz p1, loop`	[ ]
`xor r1, r2 → r3`	`xor p1, p2 → p6`	[p3]
`add r3, r4 → r4`	`add p6, p4 → p7`	[p4]
`sub r5, r2 → r3`	`sub p5, p2 → p8`	[p6]

r1	p1
r2	p2
r3	**p6**
r4	p7
r5	p5

Map table

p8
p9
p10

Free-list

Recovery example

Original insns	Renamed insns	Overwritten Reg
`bnz r1 loop`	`bnz p1, loop`	[ ]
`xor r1, r2 → r3`	`xor p1, p2 → p6`	[p3]
`add r3, r4 → r4`	`add p6, p4 → p7`	[p4]

r1	p1
r2	p2
r3	p6
r4	**p4**
r5	p5

Map table

p7
p8
p9
p10

Free-list

Recovery example

Original insns

```
bnz r1 loop
xor r1, r2  → r3
```

Renamed insns

```
bnz p1, loop
xor  p1, p2 → p6
```

Overwritten Reg

```
[  ]
[p3]
```

r1	p1
r2	p2
r3	**p3**
r4	p4
r5	p5

Map table

p6
p7
p8
p9
p10

Free-list

Recovery example

Original insns	Renamed insns	Overwritten Reg
`bnz r1 loop`	`bnz p1, loop`	[ ]

r1	p1
r2	p2
r3	p3
r4	p4
r5	p5

Map table

p6
p7
p8
p9
p10

Free-list

What about stores

- Stores: Write D$, not registers
    - Can we rename memory?
    - Recover in the cache?

No (at least not easily)

- Cache writes unrecoverable
- Stores: only when certain
    - Commit

Commit

Original insns	Renamed insns	Overwritten Reg
`xor r1, r2  → r3`	`xor  p1, p2 → p6`	[p3]
`add r3, r4 → r4`	`add  p6, p4 → p7`	[p4]
`sub r5, r2 → r3`	`sub  p5, p2 → p8`	[p6]
`addi r3, 1 → r1`	`addi p8, 1  → p9`	[p1]

- At commit: instruction becomes architected state
- In-order
- Only when instructions are finished
- Free overwritten register (why?)

Freeing over-written register

Original insns	Renamed insns	Overwritten Reg
`xor r1,r2 → r3`	`xor p1, p2 → p6`	`[p3]`
`add r3,r4 → r4`	`add p6, p4 → p7`	`[p4]`
`sub r5,r2 → r3`	`sub p5, p2 → p8`	`[p6]`
`addi r3,1 → r1`	`addi p8, 1 → p9`	`[p1]`

- Before xor: **`r3`→ `p3`**
- After xor: **`r3`→ `p6`**
  - Insns older than xor reads p3
  - Insns younger than xor read p6 (until next r3-writing instruction)
- At commit of xor, no older instructions exist
  - No one else needs p3 → free it!

Commit Example

Original insns	Renamed insns	Overwritten Reg
`xor r1,r2 → r3`	`xor  p1, p2 → p6`	[p3]
`add r3,r4 → r4`	`add  p6, p4 → p7`	[p4]
`sub r5,r2 → r3`	`sub  p5, p2 → p8`	[p6]
`addi r3,1 → r1`	`addi p8, 1  → p9`	[p1]

Map table

r1	p9
r2	p2
r3	p8
r4	p7
r5	p5

Free-list

p10

Commit Example

Original insns	Renamed insns	Overwritten Reg
`xor r1,r2 → r3`	`xor p1, p2 → p6`	[p3]
`add r3,r4 → r4`	`add p6, p4 → p7`	[p4]
`sub r5,r2 → r3`	`sub p5, p2 → p8`	[p6]
`addi r3,1 → r1`	`addi p8, 1 → p9`	[p1]

Map table

r1	p9
r2	p2
r3	p8
r4	p7
r5	p5

Free-list

p10
p3

Commit Example

Original insns	Renamed insns	Overwritten Reg
`add r3,r4 → r4`	`add  p6, p4 → p7`	`[p4]`
`sub r5,r2 → r3`	`sub  p5, p2 → p8`	`[p6]`
`addi r3,1 → r1`	`addi p8, 1  → p9`	`[p1]`

r1	p9
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p10
p3
p4

Free-list

Commit Example

Original insns	Renamed insns	Overwritten Reg
`sub r5,r2 → r3`	`sub p5, p2 → p8`	`[p6]`
`addi r3,1 → r1`	`addi p8, 1 → p9`	`[p1]`

Map table

r1	p9
r2	p2
r3	p8
r4	p7
r5	p5

Free-list

p10
p3
p4
p6

Commit Example

Original insns	Renamed insns	Overwritten Reg
`addi r3,1 → r1`	`addi p8, 1 → p9`	`[p1]`

r1	p9
r2	p2
r3	p8
r4	p7
r5	p5

Map table

p10
p3
p4
p6
p1

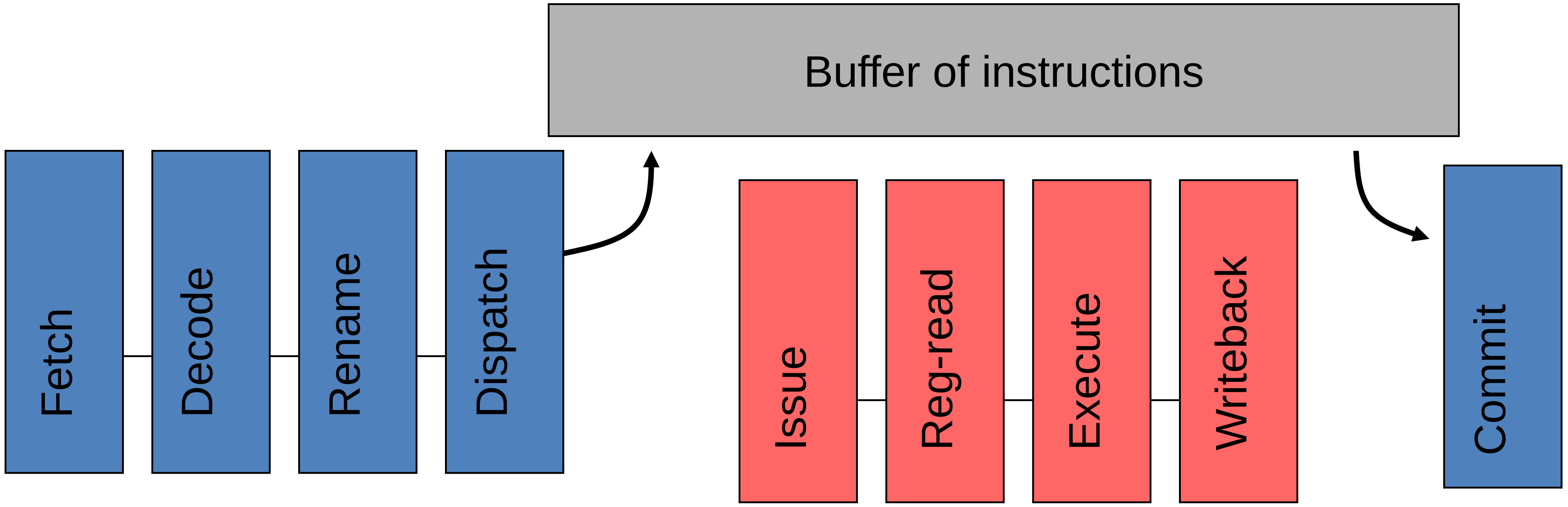
Free-list

Out of order pipeline diagrams

- Standard style: large and cumbersome
- Change layout slightly
  - Columns = stages (dispatch, issue, *etc.*)
  - Rows = instructions
  - Content of boxes = cycles
- For our purposes: issue/exec = 1 cycle
  - Ignore preg read latency, *etc.*
  - Load-use, mul, div, and FP longer

Out of order pipeline diagrams

Instruction	Disp	Issue	WB	Commit
`ld [p1] → p2`				
`add p2, p3 → p4`				
`xor p4, p5 → p6`				
`ld [p7] → p8`				



Out of order pipeline diagrams

Instruction	Disp	Issue	WB	Commit
`ld  [p1] → p2`				
`add p2, p3 → p4`				
`xor p4, p5 → p6`				
`ld [p7] → p8`				

2-wide
Infinite ROB, IQ, Pregs
Loads: 3 cycles

Out of order pipeline diagrams

Instruction	Disp	Issue	WB	Commit
`ld  [p1] → p2`	1			
`add p2, p3 → p4`	1			
`xor p4, p5 → p6`				
`ld [p7] → p8`				

Cycle 1:

- Dispatch 1st ld and add

Out of order pipeline diagrams

Instruction	Disp	Issue	WB	Commit
`ld [p1] → p2`	1	2	5	
`add p2, p3 → p4`	1			
`xor p4, p5 → p6`	2			
`ld [p7] → p8`	2			

Cycle 2:

- Dispatch xor and 2nd ld
- 1st Ld issues -- also note WB cycle while you do this
  (Note: don't issue if WB ports full)

Out of order pipeline diagrams

Instruction	Disp	Issue	WB	Commit
`ld  [p1] → p2`	1	2	5	
`add p2, p3 → p4`	1			
`xor p4, p5 → p6`	2			
`ld [p7] → p8`	2	3	6	

Cycle 3:

- add and xor are not ready
- 2nd load is → issue it

Out of order pipeline diagrams

Instruction	Disp	Issue	WB	Commit
`ld  [p1] → p2`	1	2	5	
`add p2, p3 → p4`	1	5	6	
`xor p4, p5 → p6`	2			
`ld [p7] → p8`	2	3	6	

Cycle 4:

- nothing

Cycle 5:

- add can issue

Out of order pipeline diagrams

Instruction	Disp	Issue	WB	Commit
`ld [p1] → p2`	1	2	5	6
`add p2, p3 → p4`	1	5	6	
`xor p4, p5 → p6`	2	6	7	
`ld [p7] → p8`	2	3	6	

Cycle 6:

- 1st load can commit (oldest instruction & finished)
- xor can issue

Out of order pipeline diagrams

Instruction	Disp	Issue	WB	Commit
`ld [p1] → p2`	1	2	5	6
`add p2, p3 → p4`	1	5	6	7
`xor p4, p5 → p6`	2	6	7	
`ld [p7] → p8`	2	3	6	

Cycle 7:

- add can commit (oldest instruction & finished)

Out of order pipeline diagrams

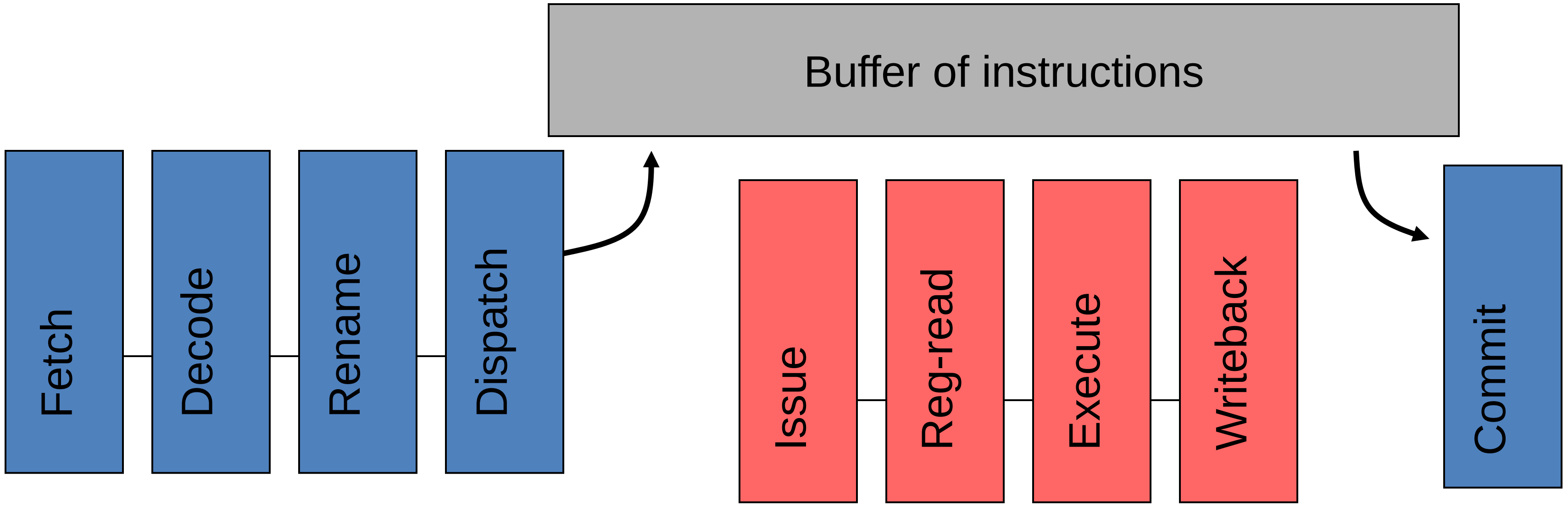
Instruction	Disp	Issue	WB	Commit
`ld [p1] → p2`	1	2	5	6
`add p2, p3 → p4`	1	5	6	7
`xor p4, p5 → p6`	2	6	7	8
`ld [p7] → p8`	2	3	6	8

Cycle 8:

- xor and ld can commit (2-wide: can do both at once)

Out of order pipeline diagrams

Instruction	Disp	Issue	WB	Commit
`ld [p1] → p2`	1	2	5	6
`add p2, p3 → p4`	1	5	6	7
`xor p4, p5 → p6`	2	6	7	8
`ld [p7] → p8`	2	3	6	8


Buffer of instructions
Fetch
Decode
Rename
Dispatch
Issue
Reg-read
Execute
Writeback
Commit