Computer Architecture
ELE 475 / COS 475
Slide Deck 6: Superscalar 3

David Wentzlaff

Department of Electrical Engineering

Princeton University

PRINCETON UNIVERSITY

PRINCETON
School of Engineering and Applied Science

Agenda

- Speculation and Branches
- Register Renaming
- Memory Disambiguation

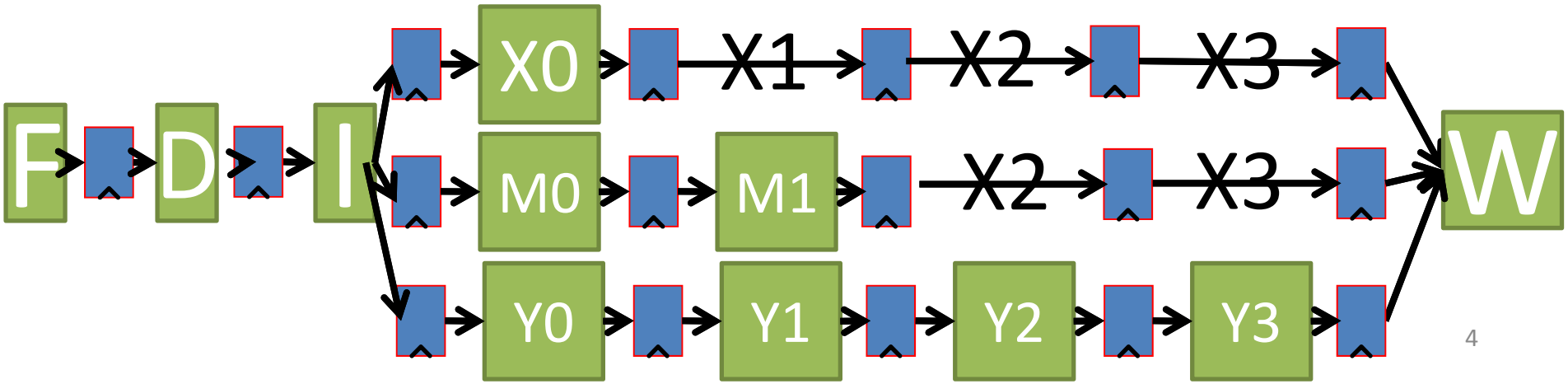
Agenda

- Speculation and Branches
- Register Renaming
- Memory Disambiguation

Speculation and Branches: I4

```
0 MUL    R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W
1 ADDIU  R4, R5, 1     F  D  I  X0 X1 X2 X3 W
2 MUL    R6, R1, R4       F  D  I  I  I  Y0 Y1 Y2 Y3 W
3 BEQZ   R6, Target          F  D  D  D  I  I  I  I  X0 X1 X2 X3 W
4 ADDIU  R8, R9 ,1              F  F  F  D  D  D  D  I  -- -- -- -- --
5 ADDIU  R10,R11,1                        F  F  F  F  D  -- -- -- -- -- --
6 ADDIU  R12,R13,1                                    F  -- -- -- -- -- -- --
T                                                        F  D  I . . .
```

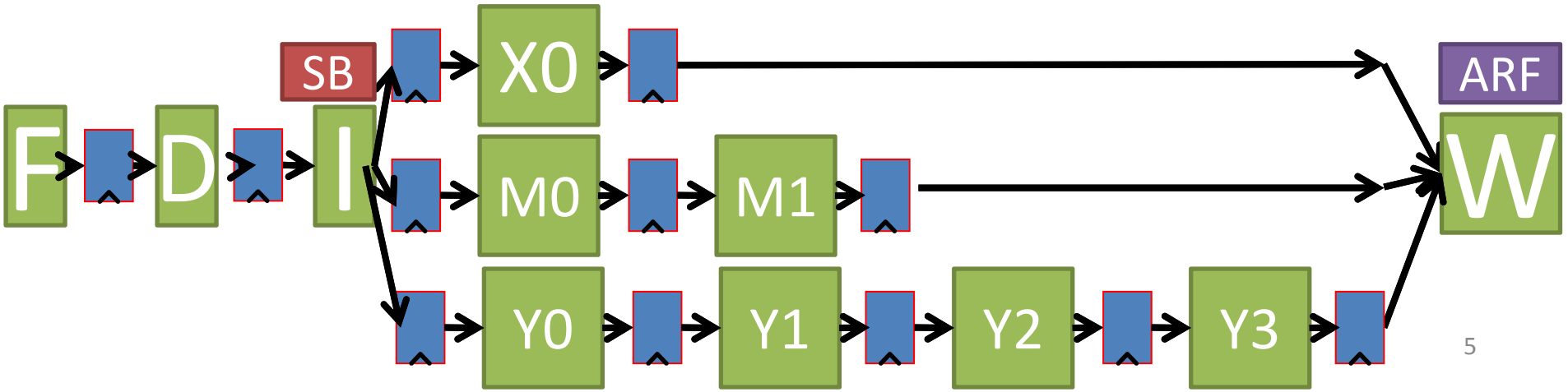
- No Speculative Instructions Commit State



Speculation and Branches: I2O2

```
0 MUL    R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W
1 ADDIU  R4, R5, 1     F  D  I  X0 W
2 MUL    R6, R1, R4       F  D  I  I  I  Y0 Y1 Y2 Y3 W
3 BEQZ   R6, Target          F  D  D  D  I  I  I  I  X0 W
4 ADDIU  R8, R9 ,1              F  F  F  D  D  D  D  I  -- --
5 ADDIU  R10,R11,1                        F  F  F  F  D  -- -- --
6 ADDIU  R12,R13,1                                    F  -- -- -- --
T                                                        F  D  I . . .
```

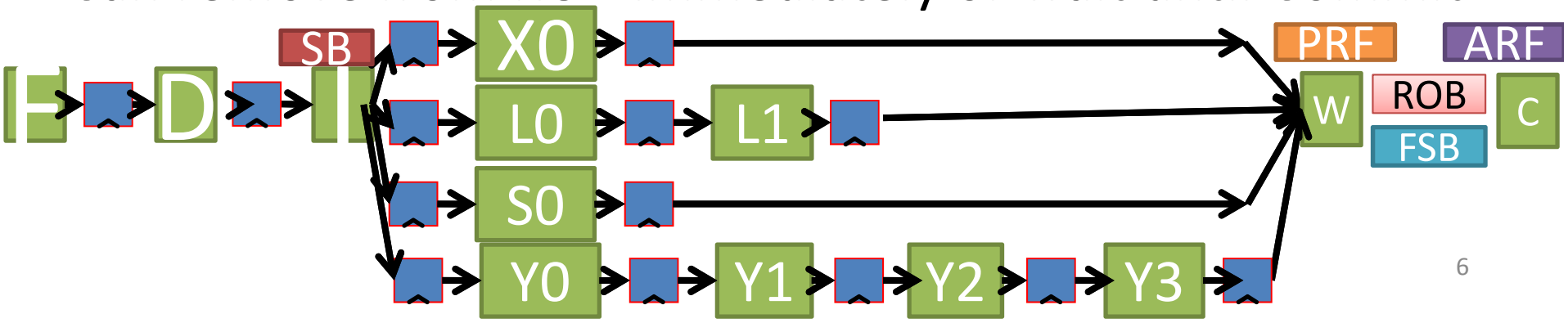
- No Speculative Instructions Commit State



Speculation and Branches: I2OI

```
0 MUL   R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W  C
1 ADDIU R4, R5, 1     F  D  I  X0 W  r        C
2 MUL   R6, R1, R4       F  D  I  I  I  Y0 Y1 Y2 Y3 W  C
3 BEQZ  R6, Target          F  D  D  D  I  I  I  I  X0 W  C
4 ADDIU R8, R9 ,1              F  F  F  D  D  D  D  I  -- -- --
5 ADDIU R10,R11,1                        F  F  F  F  D  -- -- -- --
6 ADDIU R12,R13,1                                    F  -- -- -- -- --
T                                                       F  D  I . . .
```

- Must Squash Instructions in Pipeline after Branch to prevent PRF Write.
- Can remove from ROB immediately or wait until Commit

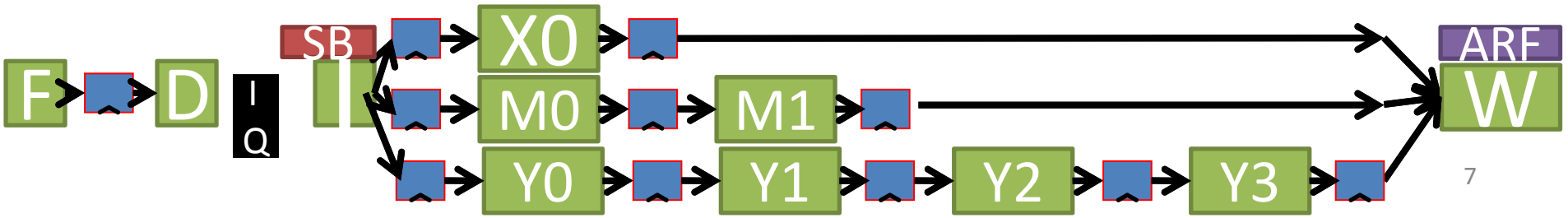


Speculation and Branches: IO3

```
0 MUL    R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W
1 ADDIU  R4, R5, 1     F  D  I  X0 W
2 MUL    R6, R1, R4       F  D  i     I  Y0 Y1 Y2 Y3 W
3 BEQZ   R6, Target          F  D  i                 I  X0 W
4 ADDIU  R8, R9 ,1              F  D  i  I  X0 W
5 ADDIU  R10,R11,1                 F  D  i  I  X0 W
6 ADDIU  R12,R13,1                    F  D  i        I  X0 W
7 ???                                    F  D
8 ???                                       F  D
9 ???                                          F  D
10???                                             F  D
11???                                                F  D
T                                                      F  D  I  .  .  .
```

Speculative Instructions Wrote to ARF

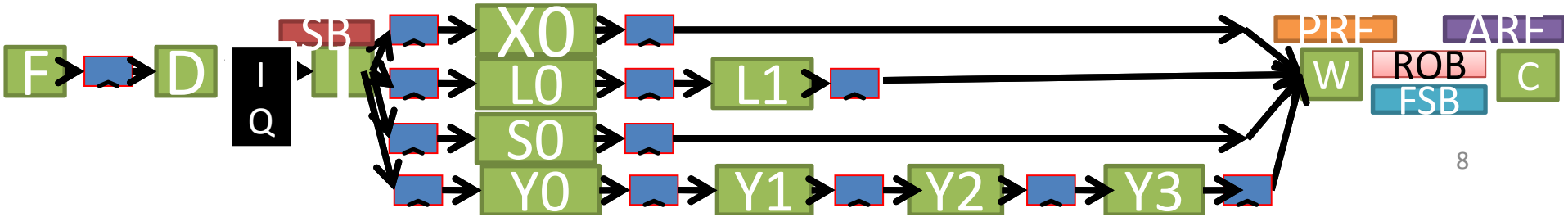
- No Control speculation for IO3
- Could Stall on Branch



Speculation and Branches: IO2I

```
0 MUL    R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W  C
1 ADDIU  R4, R5, 1     F  D  I  X0 W  r        C
2 MUL    R6, R1, R4       F  D  i     I  Y0 Y1 Y2 Y3 W  C
3 BEQZ   R6, Target          F  D  i                 I  X0 W  C
4 ADDIU  R8, R9 ,1              F  D  i  I  X0 W  r  --
5 ADDIU  R10,R11,1                 F  D  i  I  X0 W  --
6 ADDIU  R12,R13,1                    F  D  i        --
7 ???                                    F  D        --
8 ???                                       F  D     --
9 ???                                          F  D  --
10???                                             F  --
11???                                                --  D
T                                                        F  D  I  .  .  .
```

Need to clean up Speculative state In PRF. Needs Selective Rollback

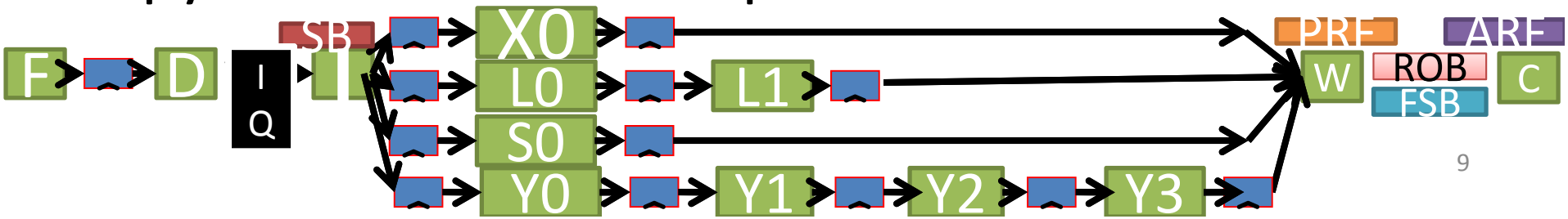


Speculation and Branches: IO2I

```
0 MUL    R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W  C
1 ADDIU  R4, R5, 1     F  D  I  X0 W  r        C
2 MUL    R6, R1, R4       F  D  i     I  Y0 Y1 Y2 Y3 W  C
3 BEQZ   R6, Target          F  D  i                 I  X0 W  C
4 ADDIU  R8, R9 ,1              F  D  i  I  X0 W  r           /
5 ADDIU  R10,R11,1                 F  D  i  I  X0 W  r        /
6 ADDIU  R12,R13,1                    F  D  i        I  X0 /
7 ???                                    F  D              /
8 ???                                       F  D           /
9 ???                                          F  D        /
10???                                             F  D     /
11???                                                F  D  /
12???                                                   F  /
13???                                                      /
T                                                           F  D  I  .  .  .
```

Speculative Instructions Wrote to PRF Not ARF

- Copy ARF to PRF on Mispredict



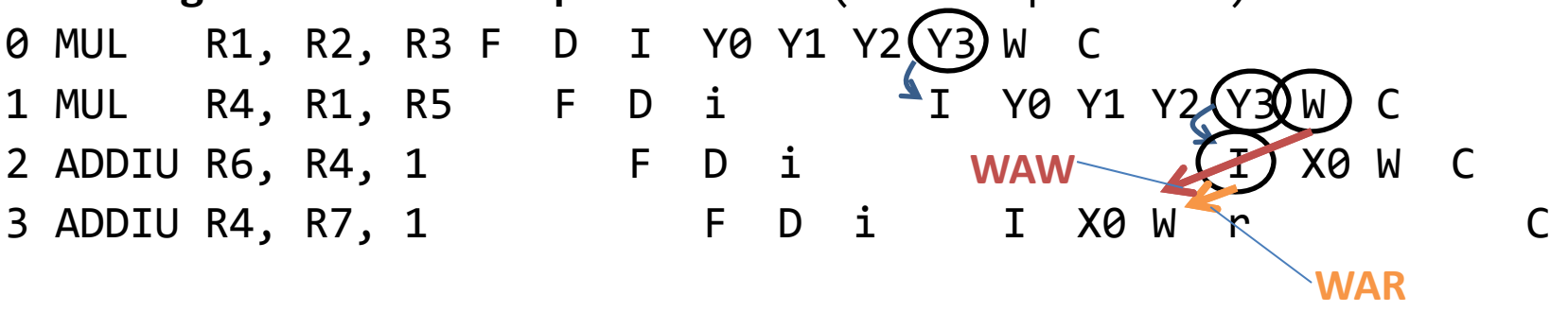
Agenda

- Speculation and Branches
- Register Renaming
- Memory Disambiguation

WAW and WAR "Name" Dependencies

- WAW and WAR are not "True" data dependencies
- RAW is "True" data dependency because reader needs result of writer
- "Name" dependencies exist because we have limited number of "Names" (register specifiers or memory addresses)

Breaking all "Name" Dependencies (Causes problems)



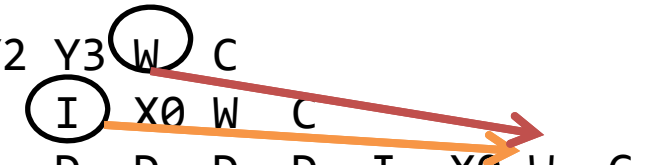
Adding More Registers

Breaking all "Name" Dependencies

```
0 MUL   R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W  C
1 MUL   R4, R1, R5    F  D  i        I  Y0 Y1 Y2 Y3 W  C
2 ADDIU R6, R4, 1        F  D  i              I  X0 W  C
3 ADDIU R4, R7, 1           F  D  i     I  X0 W  r           C
```

IO2I Microarchitecture Conservatively Stalls

```
0 MUL   R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W  C
1 MUL   R4, R1, R5    F  D  i        I  Y0 Y1 Y2 Y3 W  C
2 ADDIU R6, R4, 1        F  D  i              I  X0 W  C
3 ADDIU R4, R7, 1           F  D  D  D  D  D  D  D  D  D  I  X0 W  C
```



Manual Register Renaming. What if we could use more registers? Second R4 Write to R8?

```
0 MUL   R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W  C
1 MUL   R4, R1, R5    F  D  i        I  Y0 Y1 Y2 Y3 W  C
2 ADDIU R6, R4, 1        F  D  i              I  X0 W  C
3 ADDIU R8, R7, 1           F  D  i     I  X0 W  r           C
```

Register Renaming

- Adding more “Names” (registers/memory) removes dependence, but architecture namespace is limited.
  - Registers: Larger namespace requires more bits in instruction encoding. 32 registers = 5 bits, 128 registers = 7 bits.

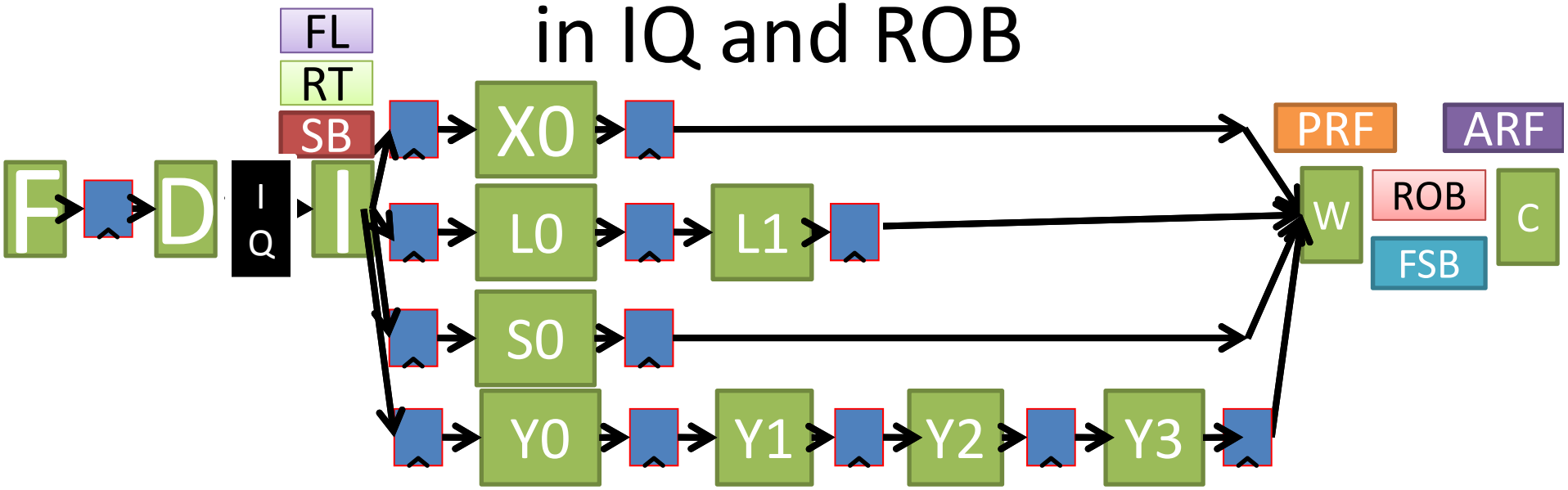
- **Register Renaming:** Change naming of registers in hardware to eliminate WAW and WAR hazards

Register Renaming Overview

- 2 Schemes
  - Pointers in the Instruction Queue/ReOrder Buffer
  - Values in the Instruction Queue/ReOrder Buffer

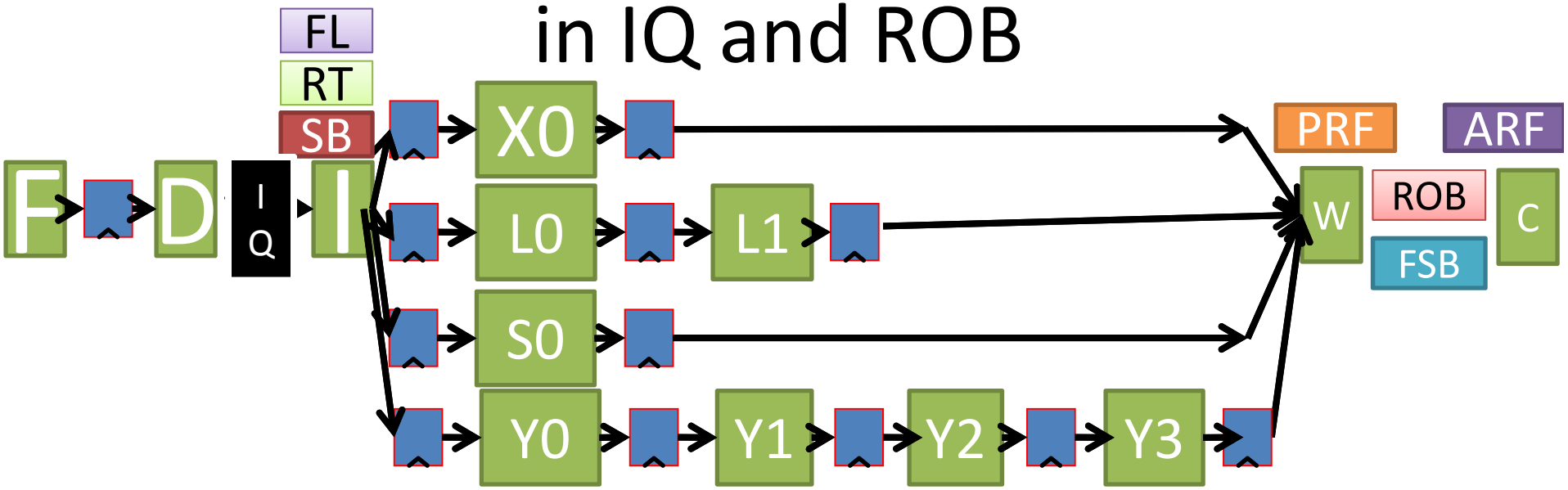
- IO2I Uses pointers in IQ and ROB therefore start with that design.

IO2I: Register Renaming with Pointers in IQ and ROB



- All data structures same as in IO2I Except:
  - Add two fields to ROB
  - Add Rename Table (RT) and Free List (FL) of registers
- Increase size of PRF to provide more register "Names"

IO2I: Register Renaming with Pointers in IQ and ROB



ARF						W
SB		R/W			W	
PRF		R			W	
ROB	R/W				W	R/W
FSB					W	R/W
IQ	W				W	
RT	R/W					W
FL	R/W					W

Modified Reorder Buffer (ROB)

State	S	ST	V	Preg	Areg	Ppreg
--						
P						
F						
P						
P						
F						
P						
P						
--						
--						

State: {Free, Pending, Finished}
S: Speculative
ST: Store bit
V: Destination is valid
Preg: Physical Register File Specifier

Areg: Architectural Register File Specifier
Ppreg: Previous Physical Register

Rename Table (RT)

	P	Preg
R1		
R2		
R3		
…		
R31		

P: Pending, Write to Destination in flight
Preg: Physical Register Architectural Register maps to.

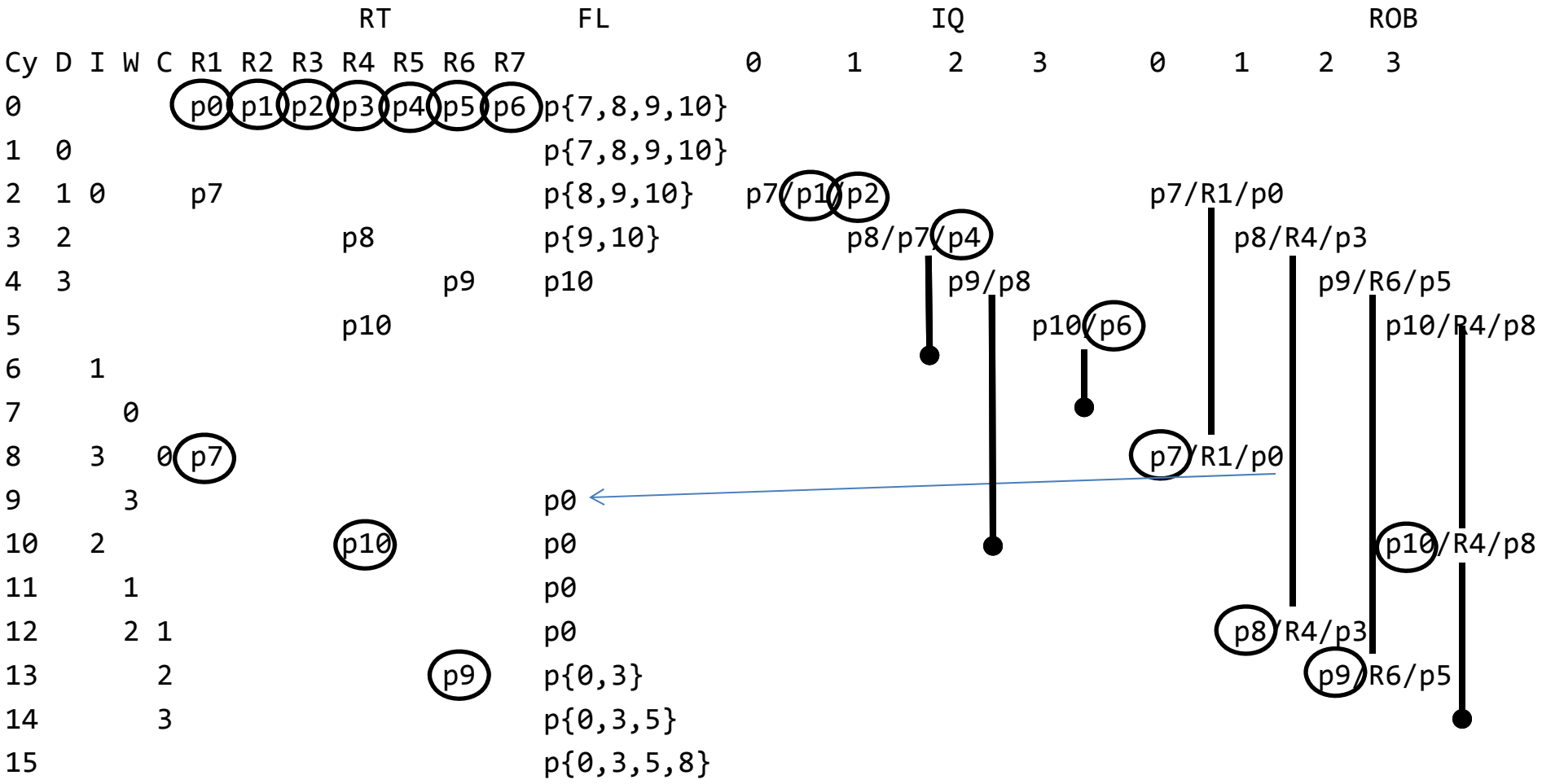
Free List (FL)

	Free
p1	
p2	
p3	
…	
pN	

Free: Register is free for renaming

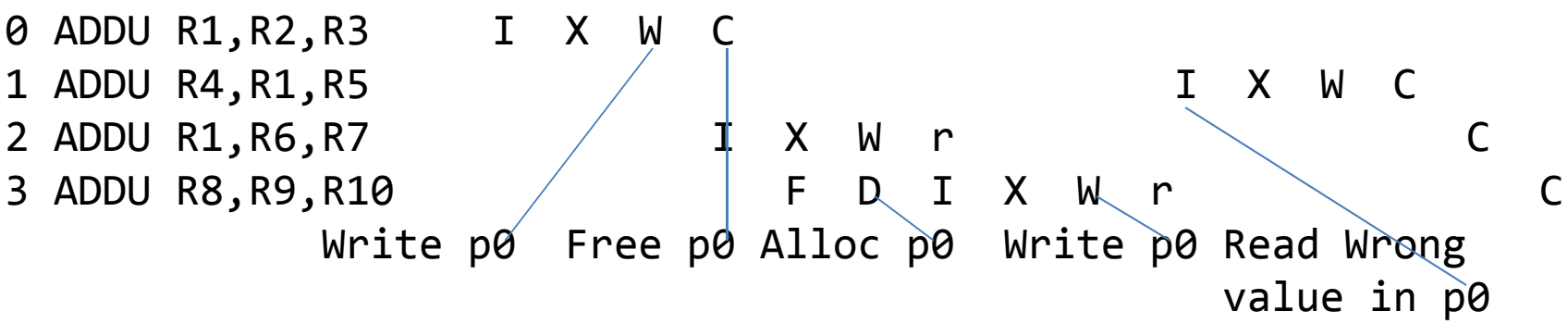
If Free == 0, physical register is in use and cannot be used for renaming

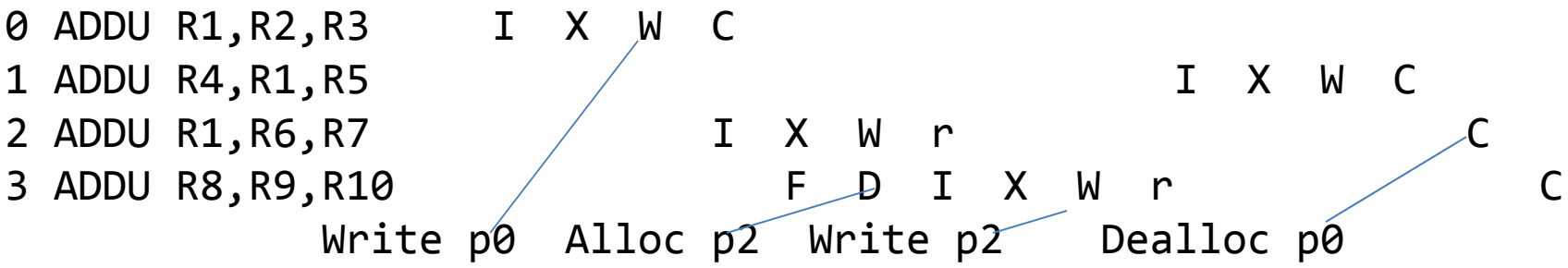
```
                    0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
0 MUL   R1, R2, R3  F  D  I  Y0 Y1 Y2 Y3 W  C
1 MUL   R4, R1, R5     F  D  i        I  Y0 Y1 Y2 Y3 W  C
2 ADDIU R6, R4, 1         F  D  i                 I  X0 W  C
3 ADDIU R4, R7, 1            F  D  i     I  X0 W  r           C
```



Freeing Physical Registers

```
ADDU R1,R2,R3    <-Assume Arch. Reg R1 maps to Phys. Reg p0
ADDU R4,R1,R5
ADDU R1,R6,R7    <-Next write of Arch Reg R1, Mapped to Phys. Reg p1
ADDU R8,R9,R10
```



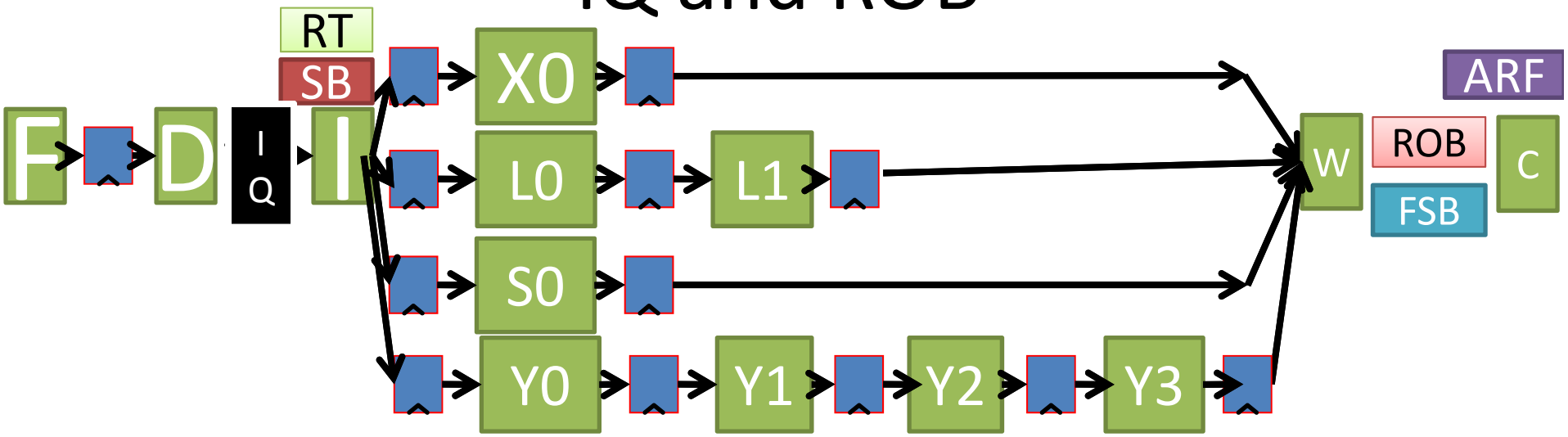


- If Arch. Reg Ri mapped to Phys. Reg pj, we can free pj when the next instruction that writes Ri commits

Unified Physical/Architectural Register File

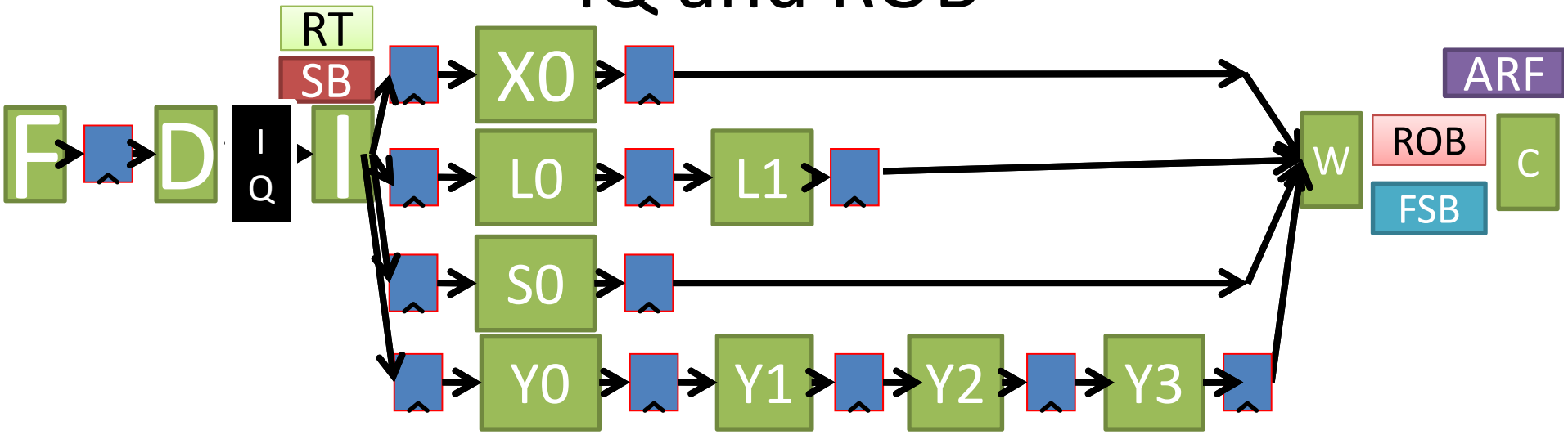
- Combine PRF and ARF into one register file
- Replace ARF with Architectural Rename Table
- Instead of copying **Values**, Commit stage copies Preg pointer into appropriate entry of Architectural Rename Table
- Unified Physical/Architectural Register file can be smaller than separate

IO2I: Register Renaming with Values in IQ and ROB



- All data structures same as previous Except:
  - Modified ROB (Values instead of Register Specifier)
  - Modified RT
  - Modified IQ
  - No FL
  - No PRF, values merged into ROB

IO2I: Register Renaming with Values in IQ and ROB



ARF	R				W
SB		R/W		W	
ROB	R/W			W	R/W
FSB				W	R/W
IQ	W			W	
RT	R/W				W

Modified Reorder Buffer (ROB)

State	S	ST	V	Value	Areg
--					
P					
F					
P					
P					
F					
P					
P					
--					
--					

State: {Free, Pending, Finished}

Areg: Architectural Register File Specifier

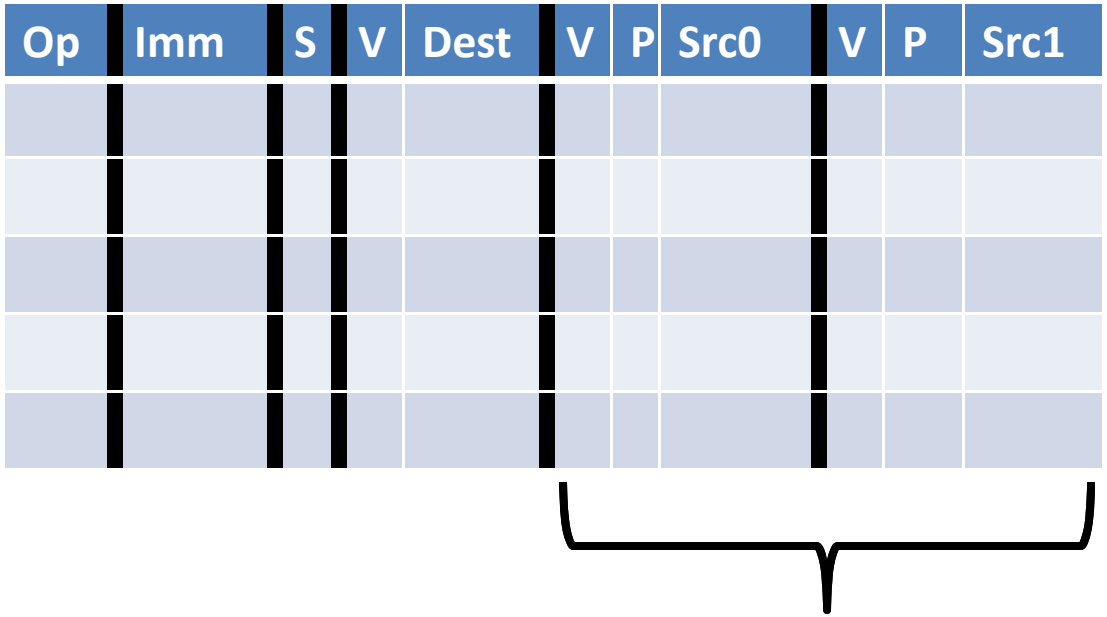
S: Speculative

ST: Store bit

V: Destination is valid

Value: Actual Register Value

Modified Issue Queue (IQ)



Op: Opcode
Imm.: Immediate
S: Speculative Bit
V: Valid (Instruction has corresponding Src/Dest)
P: Pending (Waiting on operands to be produced)

If Pending, Source Field contains index into ROB. Like a Preg identifier

Modified Rename Table (RT)

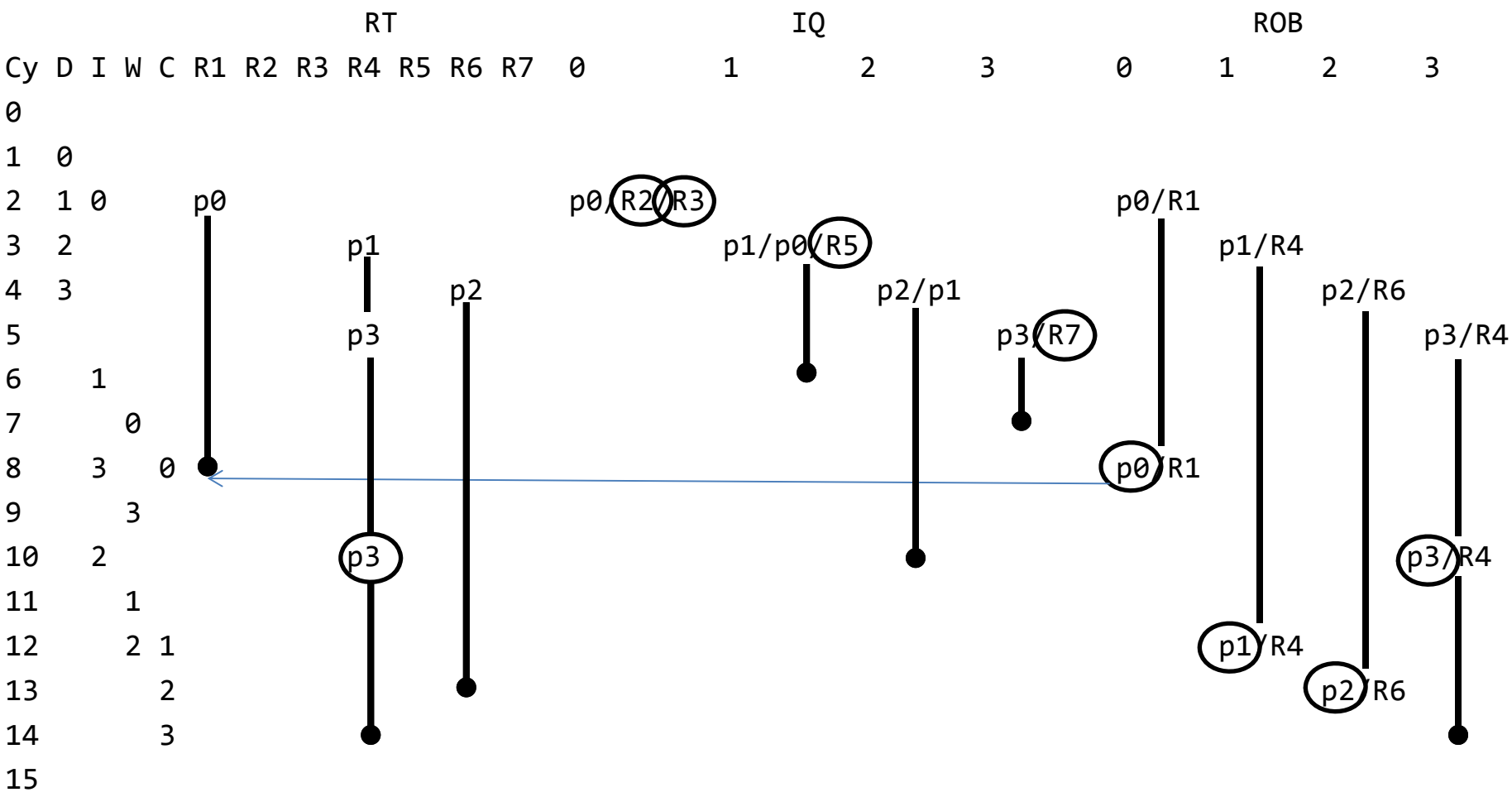
	V	P	Preg
R1			
R2			
R3			
…			
R31			

V: Valid Bit
P: Pending, Write to Destination in flight
Preg: Index into ROB

V:
If V == 0:
Value in ARF is up to date
If V == 1:
Value is in-flight or in ROB

P:
If P == 0:
Value is in ROB
if P == 1:
Value is in flight

```
                    0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
0 MUL   R1, R2, R3  F  D  I  Y0 Y1 Y2 Y3 W  C
1 MUL   R4, R1, R5     F  D  i        I  Y0 Y1 Y2 Y3 W  C
2 ADDIU R6, R4, 1         F  D  i                 I  X0 W  C
3 ADDIU R4, R7, 1            F  D  i     I  X0 W  r           C
```



Agenda

- Speculation and Branches
- Register Renaming
- Memory Disambiguation

Memory Disambiguation

```
st R1, (R2)
ld R3, (R4)
```

When can we execute the load?

In-Order Memory Queue

- Execute all loads and stores in program order

=> Load and store cannot leave ROB for execution until all previous loads and stores have completed execution

- Can still execute loads and stores speculatively, and out-of-order with respect to other instructions

- Need a structure to handle memory ordering…

Conservative OOO Load Execution

```
st R1, (R2)
ld R3, (R4)
```

- Split execution of store instruction into two phases: address calculation and data write
- Can execute load before store, if addresses known and r4 != r2
- Each load address compared with addresses of all previous uncommitted stores *(can use partial conservative check i.e., bottom 12 bits of address)*
- Don't execute load if any previous store address not known

(MIPS R10K, 16 entry address queue)

Address Speculation

```
st R1, (R2)
ld R3, (R4)
```

- Guess that r4 != r2
- Execute load before store address known
- Need to hold all completed but uncommitted load/store addresses in program order
- If subsequently find r4==r2, squash load and *all* following instructions

=> Large penalty for inaccurate address speculation

Memory Dependence Prediction

(Alpha 21264)

```
st r1, (r2)
ld r3, (r4)
```

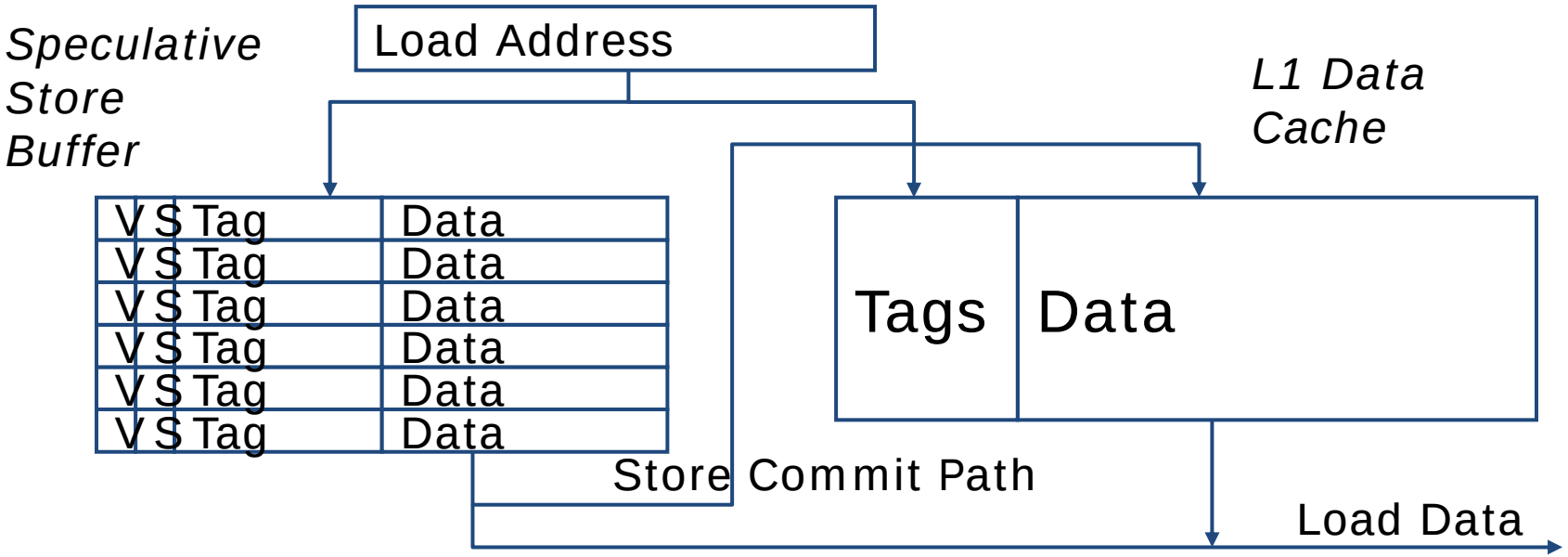
- Guess that r4 != r2 and execute load before store
- If later find r4==r2, squash load and all following instructions, but mark load instruction as *store-wait*
- Subsequent executions of the same load instruction will wait for all previous stores to complete
- Periodically clear *store-wait* bits

Speculative Loads / Stores

Just like register updates, stores should not modify the memory until after the instruction is committed

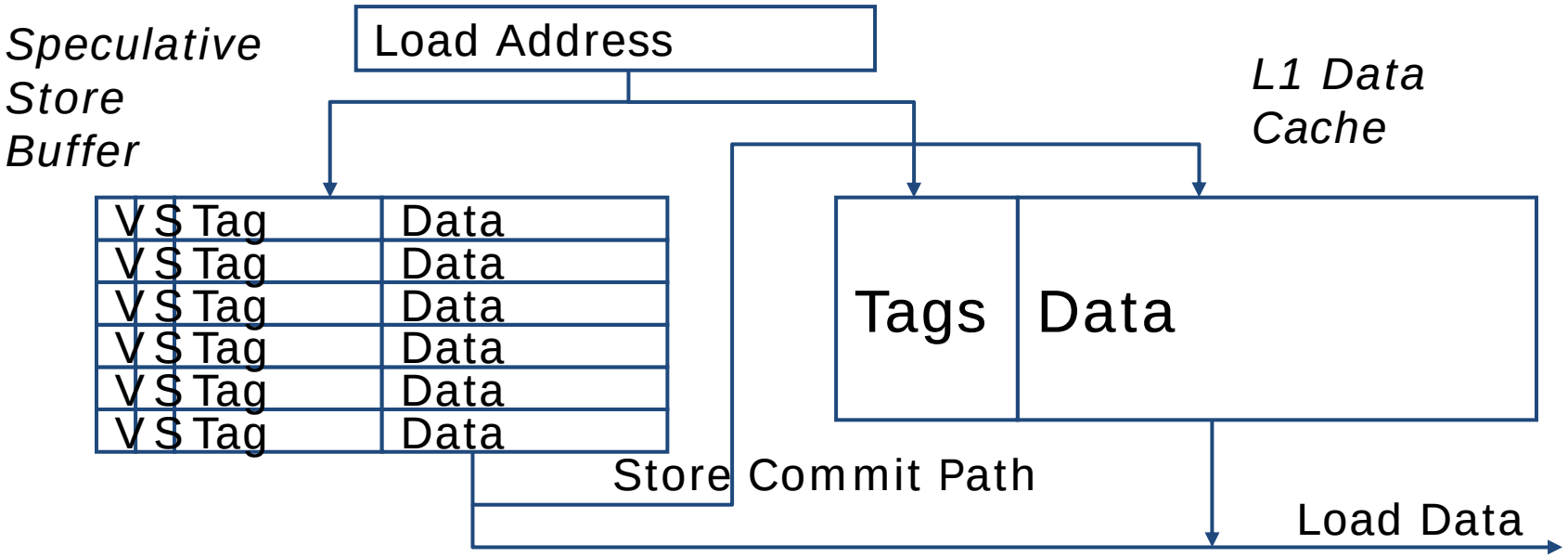
- A speculative store buffer is a structure introduced to hold speculative store data.

Speculative Store Buffer



- On store execute:
  - mark entry valid and speculative, and save data and tag of instruction.
- On store commit:
  - clear speculative bit and eventually move data to cache
- On store abort:
  - clear valid bit

Speculative Store Buffer



- If data in both store buffer and cache, which should we use?
  Speculative store buffer
- If same address in store buffer twice, which should we use?
  Youngest store older than load

Acknowledgements

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)
  - Christopher Batten (Cornell)

- MIT material derived from course 6.823
- UCB material derived from course CS252 & CS152
- Cornell material derived from course ECE 4750