COMP 590-154: Computer Architecture

Speculation and Traps in Out-of-Order Cores

What is wrong with Tomasulo's?

- Branch instructions
  - Need *branch prediction* to guess what to fetch next
  - Need *speculative execution* to "clean up" wrong guesses

- Exceptions and Traps ("software" interrupts)
  - Need to handle uncommon execution cases
    - Jump to a software handler
  - Should follow the insn. on which they were triggered
  - Often referred to as *precise interrupts*

Don't know relative order of instructions in RS

Speculation and Precise Interrupts

- When branch is mis-speculated by predictor
  - Must reset state (e.g,. regs) to time of branch
- Sequential semantics for interrupts
  - All insns. before interrupt should be complete
  - All insns. after interrupt should look as if never started (abort)

- What makes this difficult?
  - Younger insns. finish before branch → must undo writebacks
  - Older insns. not done when young branch resolves → must wait
    - Older insn. takes page fault or divide by zero → forget the branch

Same problem → Same solution

Precise State

- Speculative execution requires
  - (Ability to) abort & restart at every branch
  - Abort & restart at every load (covered in later lecture)
- Synchronous (exception and trap) events require
  - Abort & restart at every load, store, divide, …
- Asynchronous (hardware) interrupts require
  - Abort & restart at every ??

- Real world: bite the bullet
  - Implement abort & restart at every insn.
  - Called *precise state*

Precise State Implementation Options

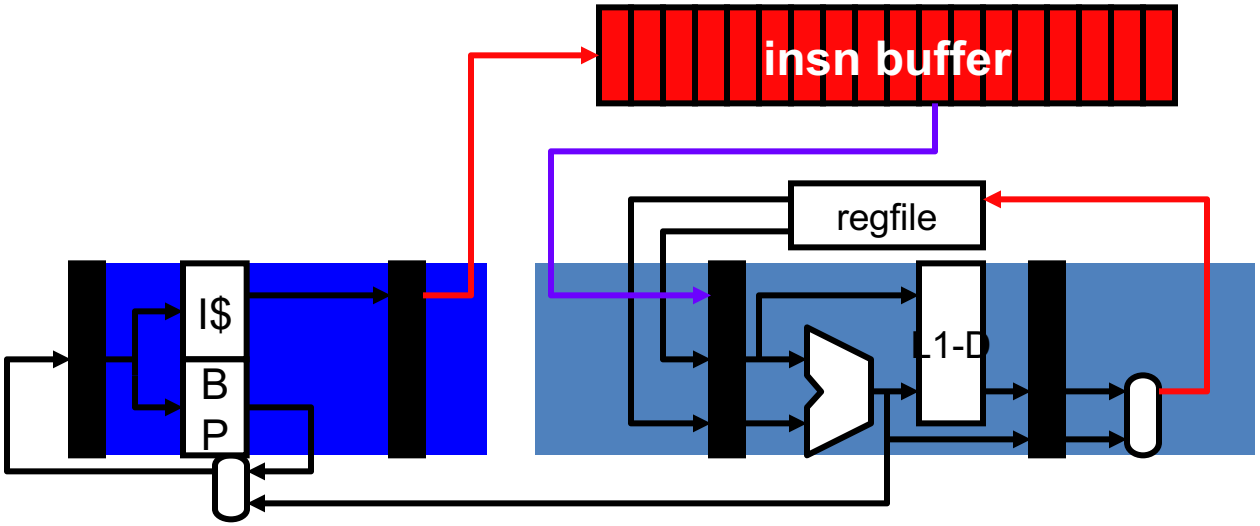
- Imprecise state: ignore the problem!
  - Makes page faults (any restartable exceptions) difficult
  - Makes speculative execution practically impossible
- Force in-order completion (W): stall pipe if necessary
  - Slow (takes away benefit of Out-of-Order)
- Keep track of precise state in hardware
  - Reset current state from precise state when needed

Everything is better in hardware

Our-of-Order Topics

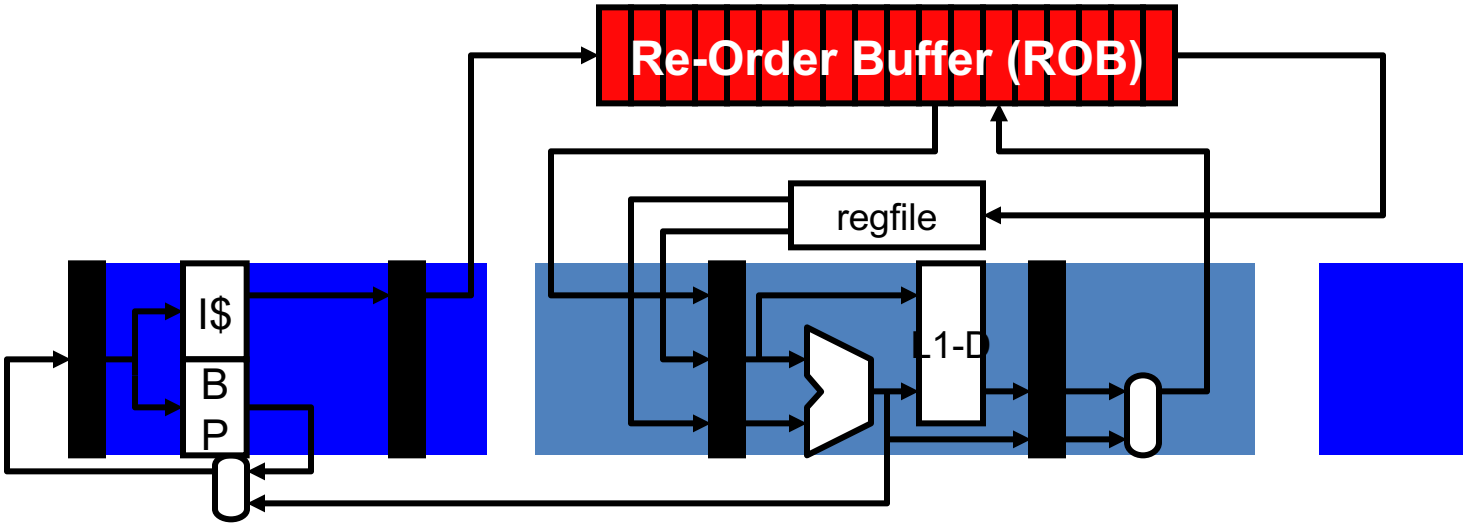
- "*Scoreboarding*"
  - First OoO, no register renaming
- "*Tomasulo's algorithm*"
  - OoO with register renaming
- Handling precise state and speculation
  - P6-style execution (Intel Pentium Pro)
  - R10k-style execution (MIPS R10k)
- Handling memory dependencies

The Problem with Precise State



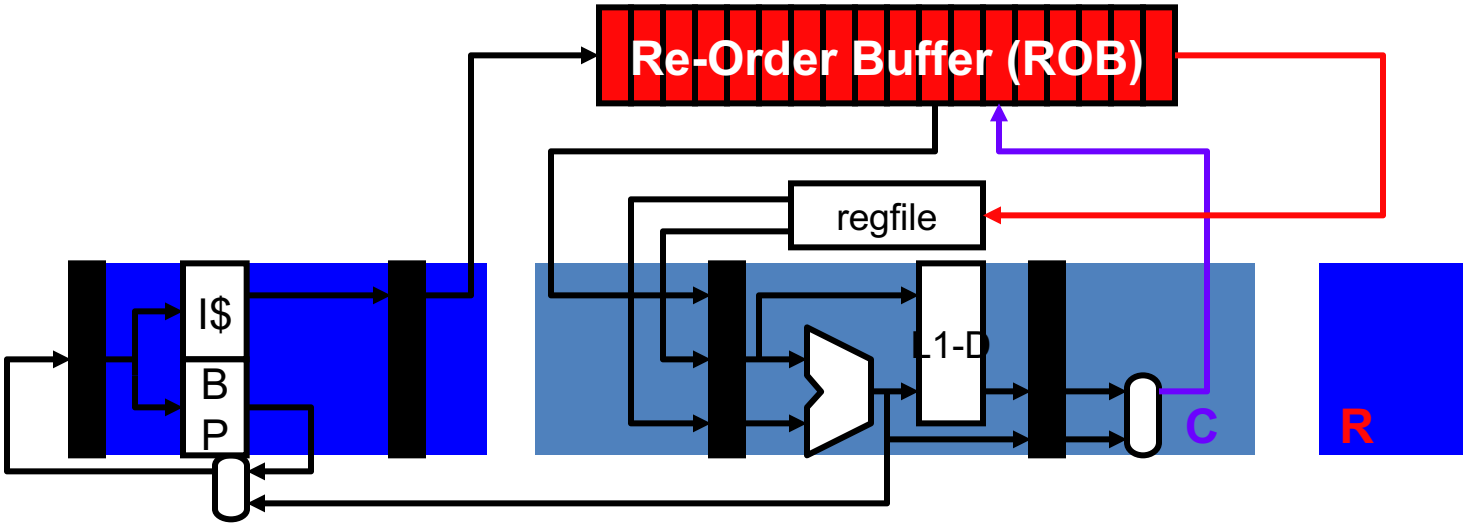
- Problem: writeback combines two functions
  - Forward values to younger insns.: out-of-order is OK
  - Write values to registers: needs to be in order
- Similar solution as for OoO decode
  - Split writeback into two stages

Re-Order Buffer (ROB)



- Insn. buffer → *Re-Order Buffer (ROB)*
  - Buffer completed results en route to register file
  - Can be merged with RS (*RUU*) or separate (common today)
- Split writeback (W) into two stages
  - Why is there no latch between W1 and W2?

Complete and Retire

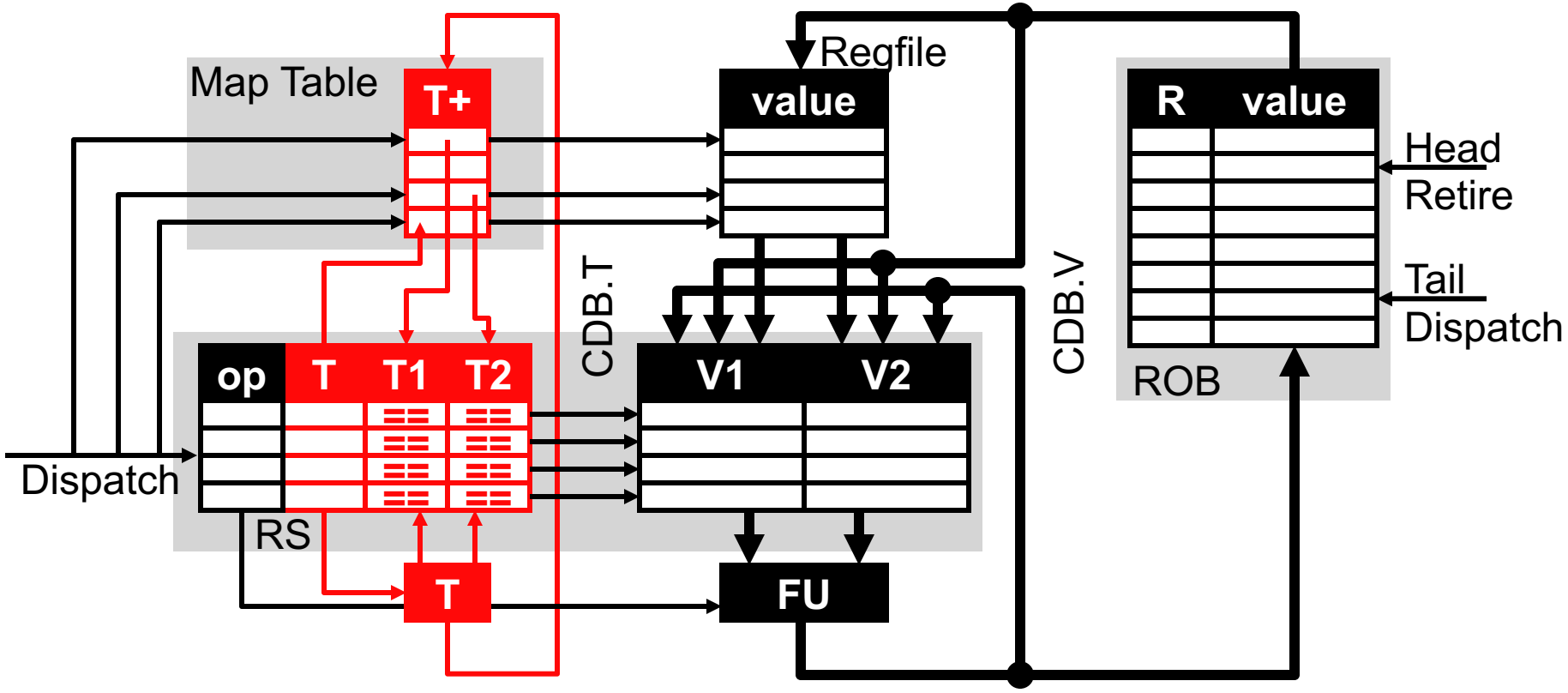


- Complete (**C**): insns. write results into ROB
  - Out-of-order: don't block younger insns.
- *Retire* (**R**): a.k.a. *commit*
  - ROB writes results to register file
  - In order: stall back-propagates to younger insns.

P6 Data Structures

- P6: Start with Tomasulo's algorithm… add ROB
- ROB (separate from RS)
  - **head**, **tail**: pointers maintain sequential order
  - **R**: insn. output register, **V**: insn. output value
- Tags are different
  - Tomasulo: RS# → P6: ROB#
- Map Table is different
  - **T+**: tag + "ready-in-ROB" bit
    - T==0 → Value is ready in register file
    - T!=0 → Value is not ready
    - T!=0+ → Value is ready in the ROB

P6 Data Structures (1/2)



P6 Data Structures (2/2)

ROB							
ht	#	Insn	R	V	S	X	C
	1	`ldf X(r1),f1`					
	2	`mulf f0,f1,f2`					
	3	`stf f2,Z(r1)`					
	4	`addi r1,4,r1`					
	5	`ldf X(r1),f1`					
	6	`mulf f0,f1,f2`					
	7	`stf f2,Z(r1)`					

Map Table	
Reg	T+
f0	
f1	
f2	
r1	

CDB	
T	V

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	no						
4	FP1	no						
5	FP2	no						

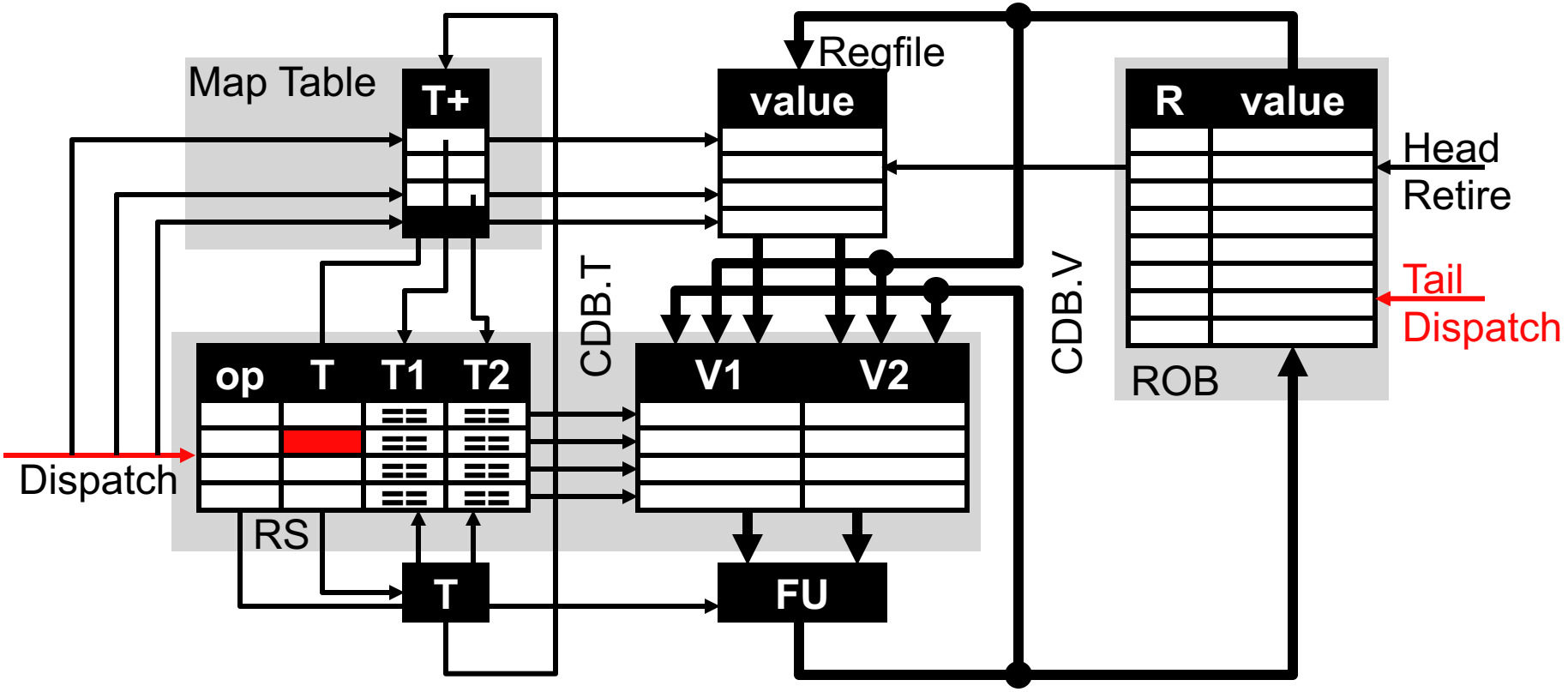
P6 Pipeline

- New pipeline structure: F, **D**, S, **X**, **C**, **R**
  - **D (dispatch)**
    - Structural hazard (ROB/RS) ? **stall**
    - Allocate ROB/RS
    - Set RS tag to ROB#
    - Set Map Table entry to ROB# and clear “ready-in-ROB” bit
    - Read ready registers into RS (from either ROB or Regfile)
  - **X (execute)**
    - Free RS entry
      - No need to wait for W, because tag is from ROB instead of RS

P6 Pipeline

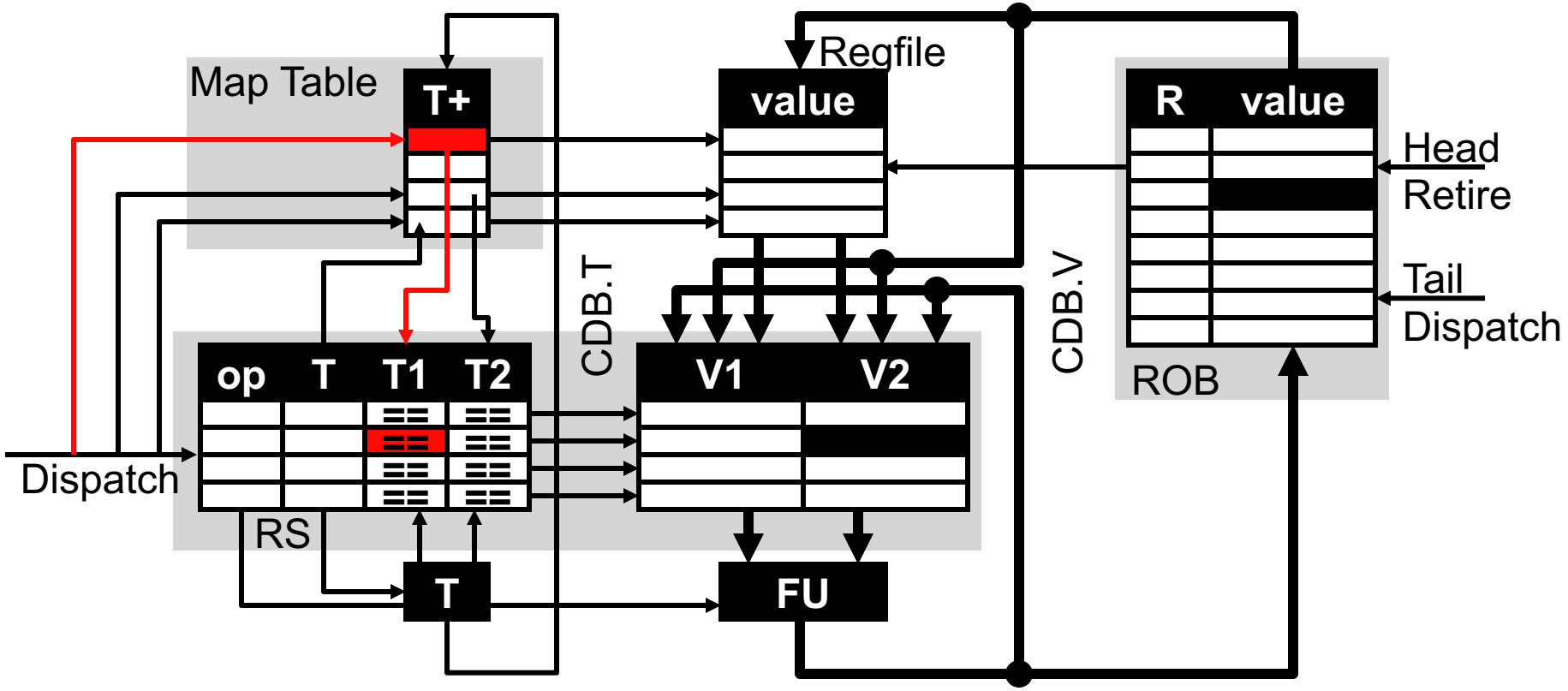
- **C (complete)**
  - Structural hazard (CDB)? **wait**
  - Write value into ROB entry for RS tag
  - If Map Table has same entry, set "ready-in-ROB" bit (+)
- **R (retire)**
  - Insn. at ROB head not complete ? **stall**
  - Handle any exceptions
    - Some go before instruction (branch mispredict, page fault) – why?
    - Some go after instruction (e.g., trap) – why?
  - ROB head value $\rightarrow$ Regfile
  - Free ROB entry

P6 Dispatch (D) (1/2)



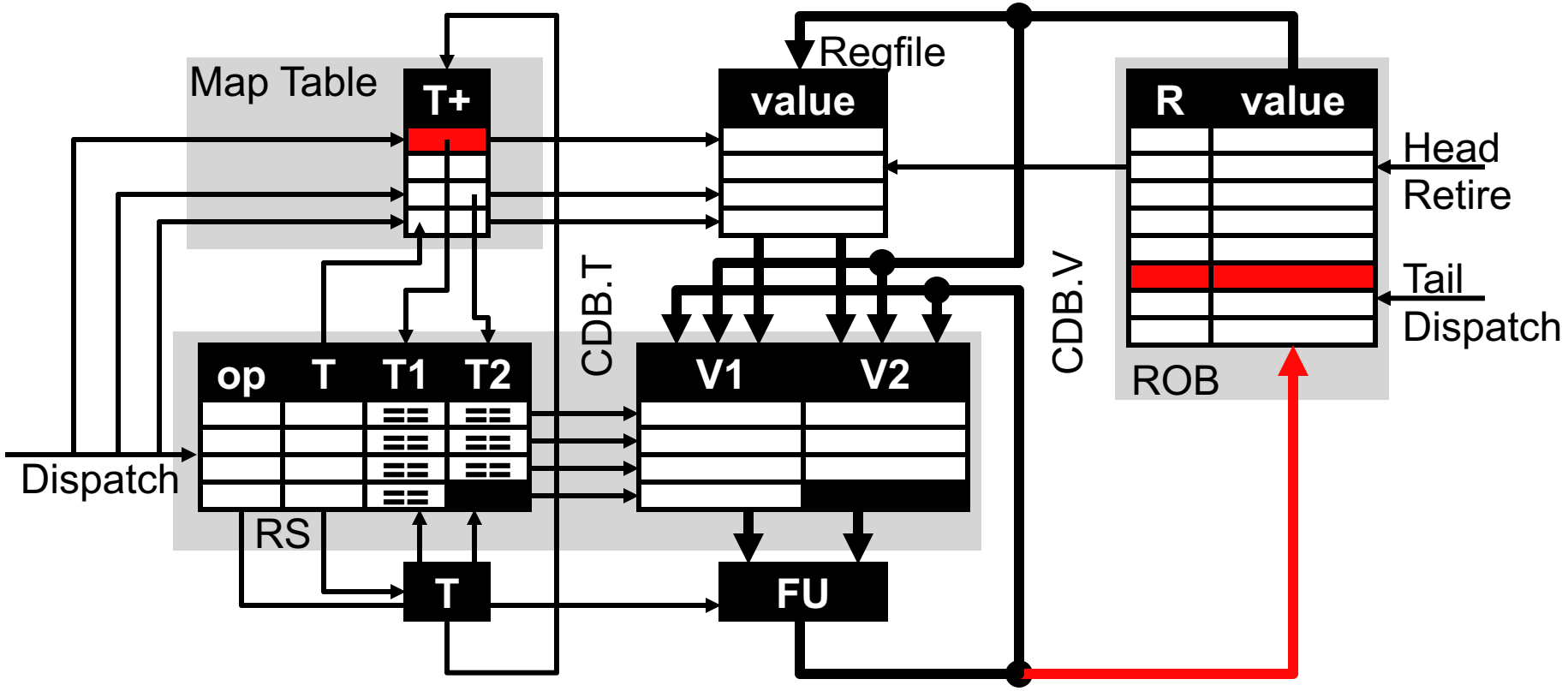
- RS/ROB full ? stall
- Allocate RS/ROB entries, assign ROB# to RS output tag
- Map Table entry set to ROB#, clear “ready-in-ROB”

P6 Dispatch (D) (2/2)



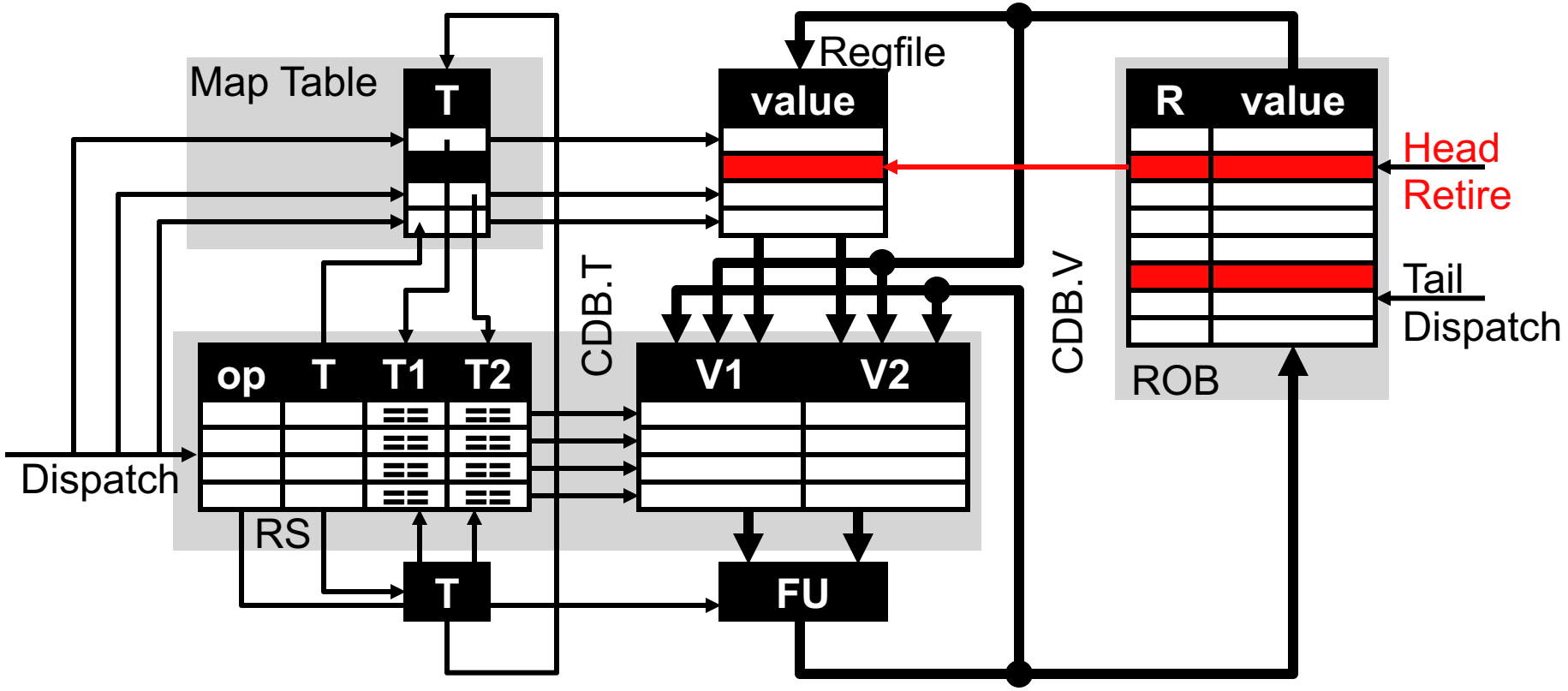
- Read tags for register inputs from Map Table
  - Tag==0 → value from Regfile (not shown)
  - Tag!=0 → Map Table tag to RS, Tag!=0+ → value from ROB

P6 Complete (C)



- CDB busy ? stall : broadcast <value,tag> on CDB
- Result → ROB, if Map Table valid → "ready-in-ROB" bit
- If RS T1 or T2 matches, write CDB.V into RS slot

P6 Retire (R)



- ROB head not complete ? stall : free ROB entry
- Write ROB head result to Regfile
- If still valid, clear Map Table entry

P6: Cycle 1

ROB							
ht	#	Insn	R	V	S	X	C
ht	1	ldf X(r1),f1	f1				
	2	mulf f0,f1,f2					
	3	stf f2,Z(r1)					
	4	addi r1,4,r1					
	5	ldf X(r1),f1					
	6	mulf f0,f1,f2					
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#1
f2	
r1	

CDB	
T	V

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	ROB#1				[r1]
3	ST	no						
4	FP1	no						
5	FP2	no						

set ROB# tag

allocate

P6: Cycle 2

ROB							
ht	#	Insn	R	V	S	X	C
h	1	ldf X(r1),f1	f1		c2		
t	2	mulf f0,f1,f2	f2				
	3	stf f2,Z(r1)					
	4	addi r1,4,r1					
	5	ldf X(r1),f1					
	6	mulf f0,f1,f2					
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#1
f2	ROB#2
r1	

CDB	
T	V

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	ROB#1				[r1]
3	ST	no						
4	FP1	yes	mulf	ROB#2		ROB#1	[f0]	
5	FP2	no						

set ROB# tag

allocate

P6: Cycle 3

ROB							
ht	#	Insn	R	V	S	X	C
h	1	ldf X(r1),f1	f1		c2	c3	
	2	mulf f0,f1,f2	f2				
t	3	stf f2,Z(r1)					
	4	addi r1,4,r1					
	5	ldf X(r1),f1					
	6	mulf f0,f1,f2					
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#1
f2	ROB#2
r1	

CDB	
T	V

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	ROB#3	ROB#2			[r1]
4	FP1	yes	mulf	ROB#2		ROB#1	[f0]	
5	FP2	no						

free
allocate

P6: Cycle 4

ROB							
ht	#	Insn	R	V	S	X	C
h	1	ldf X(r1),f1	f1	[f1]	c2	c3	c4
	2	mulf f0,f1,f2	f2		c4		
	3	stf f2,Z(r1)					
t	4	addi r1,4,r1	r1				
	5	ldf X(r1),f1					
	6	mulf f0,f1,f2					
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#1+
f2	ROB#2
r1	ROB#4

CDB	
T	V
ROB#1	[f1]

ldf finished
1. set "ready-in-ROB" bit
2. write result to ROB
3. CDB broadcast

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	yes	add	ROB#4			[r1]	
2	LD	no						
3	ST	yes	stf	ROB#3	ROB#2			[r1]
4	FP1	yes	mulf	ROB#2		ROB#1	[f0]	CDB.V
5	FP2	no						

allocate

ROB#1 ready
grab CDB.V

P6: Cycle 5

ROB							
ht	#	Insn	R	V	S	X	C
	1	ldf X(r1),f1	f1	[f1]	c2	c3	c4
h	2	mulf f0,f1,f2	f2		c4	c5	
	3	stf f2,Z(r1)					
	4	addi r1,4,r1	r1		c5		
t	5	ldf X(r1),f1	f1				
	6	mulf f0,f1,f2					
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#5
f2	ROB#2
r1	ROB#4

CDB	
T	V

ldf retires

1. write ROB result to regfile

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	yes	add	ROB#4			[r1]	
2	LD	yes	ldf	ROB#5		ROB#4		
3	ST	yes	stf	ROB#3	ROB#2			[r1]
4	FP1	no						
5	FP2	no						

allocate

free

P6: Cycle 6

ROB							
ht	#	Insn	R	V	S	X	C
	1	ldf X(r1),f1	f1	[f1]	c2	c3	c4
h	2	mulf f0,f1,f2	f2		c4	c5+	
	3	stf f2,Z(r1)					
	4	addi r1,4,r1	r1		c5	c6	
	5	ldf X(r1),f1	f1				
t	6	mulf f0,f1,f2	f2				
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#5
f2	ROB#6
r1	ROB#4

CDB	
T	V

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	ROB#5		ROB#4		
3	ST	yes	stf	ROB#3	ROB#2			[r1]
4	FP1	yes	mulf	ROB#6		ROB#5	[f0]	
5	FP2	no						

free

allocate

P6: Cycle 7

ROB							
ht	#	Insn	R	V	S	X	C
	1	ldf X(r1),f1	f1	[f1]	c2	c3	c4
h	2	mulf f0,f1,f2	f2		c4	c5+	
	3	stf f2,Z(r1)					
	4	addi r1,4,r1	r1	[r1]	c5	c6	c7
	5	ldf X(r1),f1	f1		c7		
t	6	mulf f0,f1,f2	f2				
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#5
f2	ROB#6
r1	ROB#4+

CDB	
T	V
ROB#4	[r1]

stall D (no free STore RS)

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	ROB#5		ROB#4		CDB.V
3	ST	yes	stf	ROB#3	ROB#2			[r1]
4	FP1	yes	mulf	ROB#6		ROB#5	[f0]	
5	FP2	no						

ROB#4 ready grab CDB.V

P6: Cycle 8

ROB							
ht	#	Insn	R	V	S	X	C
	1	ldf X(r1),f1	f1	[f1]	c2	c3	c4
h	2	mulf f0,f1,f2	f2	[f2]	c4	c5+	c8
	3	stf f2,Z(r1)			c8		
	4	addi r1,4,r1	r1	[r1]	c5	c6	c7
	5	ldf X(r1),f1	f1		c7	c8	
t	6	mulf f0,f1,f2	f2				
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#5
f2	ROB#6
r1	ROB#4+

CDB	
T	V
ROB#2	[f2]

stall R for addi (in-order)

ROB#2 not in MapTable f2, don't set "ready-in-ROB"

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	ROB#3	ROB#2		[f2]	[r1]
4	FP1	yes	mulf	ROB#6		ROB#5	[f0]	
5	FP2	no						

ROB#2 ready grab CDB.V

P6: Cycle 9

ROB							
ht	#	Insn	R	V	S	X	C
	1	ldf X(r1),f1	f1	[f1]	c2	c3	c4
	2	mulf f0,f1,f2	f2	[f2]	c4	c5+	c8
h	3	stf f2,Z(r1)			c8	c9	
	4	addi r1,4,r1	r1	[r1]	c5	c6	c7
	5	ldf X(r1),f1	f1	[f1]	c7	c8	c9
	6	mulf f0,f1,f2	f2		c9		
t	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#5+
f2	ROB#6
r1	ROB#4+

CDB	
T	V
ROB#5	[f1]

retire mulf

all pipe stages active at once!

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	ROB#7	ROB#6			ROB#4.V
4	FP1	yes	mulf	ROB#6		ROB#5	[f0]	CDB.V
5	FP2	no						

free, re-allocate

ROB#5 ready

grab CDB.V

P6: Cycle 10

ROB							
ht	#	Insn	R	V	S	X	C
	1	`ldf X(r1),f1`	f1	[f1]	c2	c3	c4
	2	`mulf f0,f1,f2`	f2	[f2]	c4	c5+	c8
h	3	`stf f2,Z(r1)`			c8	c9	c10
	4	`addi r1,4,r1`	r1	[r1]	c5	c6	c7
	5	`ldf X(r1),f1`	f1	[f1]	c7	c8	c9
	6	`mulf f0,f1,f2`	f2		c9	c10	
t	7	`stf f2,Z(r1)`					

Map Table	
Reg	T+
f0	
f1	ROB#5+
f2	ROB#6
r1	ROB#4+

CDB	
T	V

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	ROB#7	ROB#6			ROB#4.V
4	FP1	no						
5	FP2	no						

free

P6: Cycle 11

ROB							
ht	#	Insn	R	V	S	X	C
	1	ldf X(r1),f1	f1	[f1]	c2	c3	c4
	2	mulf f0,f1,f2	f2	[f2]	c4	c5	c8
	3	stf f2,Z(r1)			c8	c9	c10
h	4	addi r1,4,r1	r1	[r1]	c5	c6	c7
	5	ldf X(r1),f1	f1	[f1]	c7	c8	c9
	6	mulf f0,f1,f2	f2		c9	c10	
t	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#5+
f2	ROB#6
r1	ROB#4+

retire stf

CDB	
T	V

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	ROB#7	ROB#6			ROB#4.V
4	FP1	no						
5	FP2	no						

Precise State in P6

- Point of ROB is maintaining **precise state**
  - How does that work?
    1. Wait until last good insn. retires, first bad insn. at ROB head
    2. Zero (0) contents of ROB, RS, and Map Table
    3. Start over
  - Works because zero (0) means the right thing…
    - 0 in ROB/RS → entry is empty
    - Tag == 0 in Map Table → register is in Regfile
  - …and because Regfile and L1-D writes take place at R
  - Example: page fault in first **`stf`**

P6: Cycle 9 (with precise state)

ROB							
ht	#	Insn	R	V	S	X	C
	1	ldf X(r1),f1	f1	[f1]	c2	c3	c4
	2	mulf f0,f1,f2	f2	[f2]	c4	c5+	c8
h	3	stf f2,Z(r1)			c8	c9	
	4	addi r1,4,r1	r1	[r1]	c5	c6	c7
	5	ldf X(r1),f1	f1	[f1]	c7	c8	c9
	6	mulf f0,f1,f2	f2		c9		
t	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	
f1	ROB#5+
f2	ROB#6
r1	ROB#4+

CDB	
T	V
ROB#5	[f1]

PAGE FAULT

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	ROB#7	ROB#6			ROB#4.V
4	FP1	yes	mulf	ROB#6		ROB#5	[f0]	CDB.V
5	FP2	no						

P6: Cycle 10 (with precise state)

ROB							
ht	#	Insn	R	V	S	X	C
	1	`ldf X(r1),f1`	f1	[f1]	c2	c3	c4
	2	`mulf f0,f1,f2`	f2	[f2]	c4	c5+	c8
	3	`stf f2,Z(r1)`					
	4	`addi r1,4,r1`					
	5	`ldf X(r1),f1`					
	6	`mulf f0,f1,f2`					
	7	`stf f2,Z(r1)`					

Map Table	
Reg	T+
f0	
f1	
f2	
r1	

CDB	
T	V

faulting insn at ROB head?
CLEAR EVERYTHING
set fetch PC to fault handler

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	no						
4	FP1	no						
5	FP2	no						

P6: Cycle 11 (with precise state)

ROB							
ht	#	Insn	R	V	S	X	C
	1	`ldf X(r1),f1`	f1	[f1]	c2	c3	c4
	2	`mulf f0,f1,f2`	f2	[f2]	c4	c5+	c8
ht	3	`stf f2,Z(r1)`					
	4	`addi r1,4,r1`					
	5	`ldf X(r1),f1`					
	6	`mulf f0,f1,f2`					
	7	`stf f2,Z(r1)`					

Map Table	
Reg	T+
f0	
f1	
f2	
r1	

CDB	
T	V

PF handler done?
CLEAR EVERYTHING
iret fetch PC to faulting insn.

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	ROB#3			[f4]	[r1]
4	FP1	no						
5	FP2	no						

P6: Cycle 12 (with precise state)

ROB							
ht	#	Insn	R	V	S	X	C
	1	`ldf X(r1),f1`	f1	[f1]	c2	c3	c4
	2	`mulf f0,f1,f2`	f2	[f2]	c4	c5+	c8
h	3	`stf f2,Z(r1)`			c12		
t	4	`addi r1,4,r1`	r1				
	5	`ldf X(r1),f1`					
	6	`mulf f0,f1,f2`					
	7	`stf f2,Z(r1)`					

Map Table	
Reg	T+
f0	
f1	
f2	
r1	ROB#4

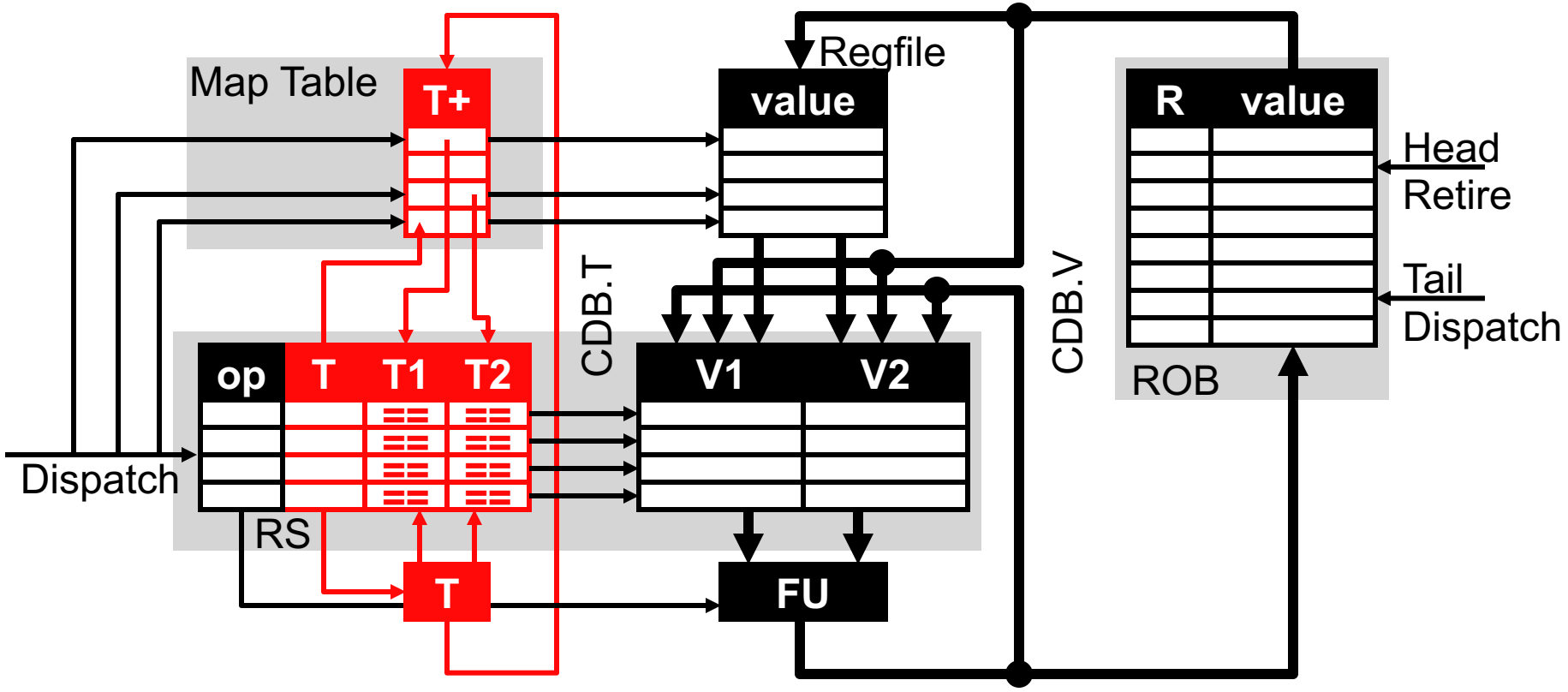
CDB	
T	V

Reservation Stations								
#	FU	busy	op	T	T1	T2	V1	V2
1	ALU	yes	addi	ROB#4			[r1]	
2	LD	no						
3	ST	yes	stf	ROB#3			[f4]	[r1]
4	FP1	no						
5	FP2	no						

P6 Performance

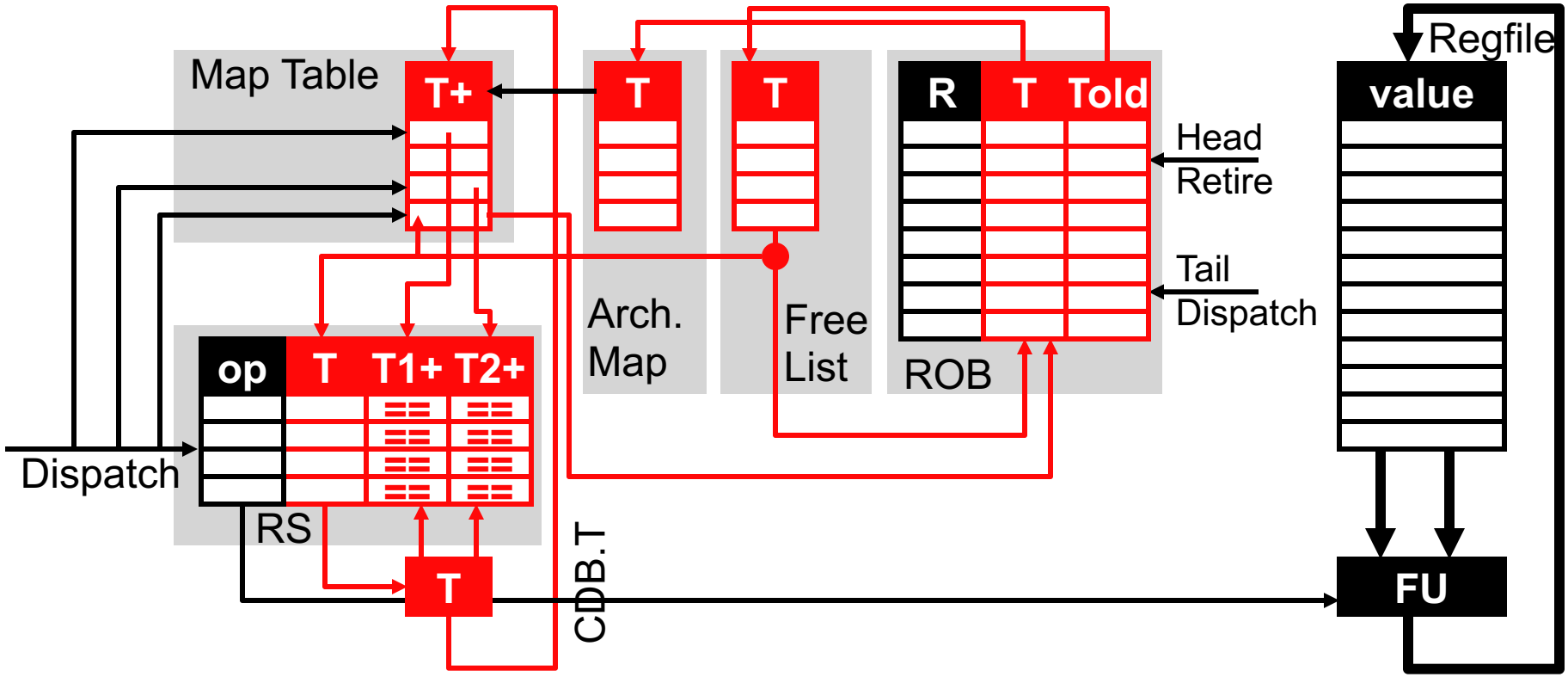
- In other words: what is the cost of precise state?
  - \+ In general: same performance as "plain" Tomasulo
    - ROB is not a performance device
    - Maybe a little better (RS freed earlier $\rightarrow$ fewer struct hazards)
  - – Unless ROB is too small
    - In which case ROB struct hazards become a problem
  - – Rules of thumb for ROB size
    - At least N (width) * number of pipe stages between D and R
    - At least N * $t_{hit\text{-}L2}$
    - Can add a factor of 2 to both if you want
    - What is the rationale behind these?

The Problem with P6



- Problem for high performance implementations
  - Too much **value movement** (Regfile/ROB→RS→ROB→Regfile)
  - Multi-input muxes, long buses, slow clock

MIPS R10K: Alternative Implementation



- One big **physical register file** holds all data - no copies
  - \+ Register file close to FUs → small and fast data path
  - – ROB and RS "on the side" used only for control and tags

Register Renaming in R10K

- Architectural register file? Gone
- Physical register file holds all values
  - #physical registers = #architectural registers + #ROB entries
  - Map architectural registers to physical registers
  - No WAW or WAR hazards (physical regs. replace RS values)
- Fundamental change to map table
  - Mappings cannot be 0 (no architectural register file)
- Explicit free list tracks unallocated physical regs.
  - ROB returns physical regs. to free list

Register Renaming

- Example
  - Names: r1,r2,r3 Locations: p1,p2,p3,p4,p5,p6,p7
  - Original: r1→p1, r2→p2, r3→p3, p4–p7 are "free"

MapTable

r1	r2	r3
p1	p2	p3
p4	p2	p3
p4	p2	p5
p4	p2	p6

FreeList

p4,p5,p6,p7
p5,p6,p7
p6,p7
p7

Original insns.

```
add r2,r3,r1
sub r2,r1,r3
mul r2,r3,r3
div r1,4,r1
```

Renamed insns.

```
add p2,p3,p4
sub p2,p4,p5
mul p2,p5,p6
div p4,4,p7
```

Register Renaming

- Example
  - Names: r1,r2,r3 Locations: p1,p2,p3,p4,p5,p6,p7
  - Original: r1→p1, r2→p2, r3→p3, p4–p7 are "free"

MapTable

r1	r2	r3
p1	p2	p3
p4	p2	p3
p4	p2	p5
p4	p2	p6
p7	p2	p6

FreeList

p4,p5,p6,p7
p5,p6,p7
p6,p7
p7

Original insns.
add r2,r3,r1
sub r2,r1,r3
mul r2,r3,r3
div r1,4,r1
add r1,r3,r2

Renamed insns.
add p2,p3,p4
sub p2,p4,p5
mul p2,p5,p6
div p4,4,p7
add p7,p6,???

- Question: ~~how is the last~~ ***add*** renamed?
  - We are out of free physical registers
  - Real question: how/when are physical registers freed?

Physical Register Reclamation

- P6
  - No need to free speculative ("in-flight") values explicitly
  - Temporary storage comes with ROB entry
- R10K
  - Can't free physical regs. when insn. retires
    - Younger insns. likely depend on it
  - But…
    - Can free physical reg. previously mapped to same logical reg.
    - Why?

Freeing Registers in R10K

MapTable

r1	r2	r3
p1	p2	p3
p4	p2	p3
p4	p2	p5
p4	p2	p6
p7	p2	p6

FreeList

p4,p5,p6,p7
p5,p6,p7
p6,p7
p7
p1

Original insns.

```
add r2,r3,r1
sub r2,r1,r3
mul r2,r3,r3
div r1,4,r1
add r1,r3,r2
```

Renamed insns.

```
~~add p2,p3,p4~~
sub p2,p4,p5
mul p2,p5,p6
div p4,4,p7
add p7,p6,p1
```

- When ***add*** retires, free p1
- When ***sub*** retires, free p3
- When ***mul*** retires, free p5
- When ***div*** retires, free p4

Always OK to free ***old*** mapping

R10K Data Structures

- New tags (again)
  - P6: ROB# → R10K: PR#
- ROB
  - **T**: physical register corresponding to insn's logical output
  - **Told**: physical register previously mapped to insn's logical output
- RS
  - **T**, **T1**, **T2**: output, input physical registers
- Map Table
  - **T+**: PR# (never empty) + "ready" bit
- Architectural Map Table
  - **T**: PR# (never empty)
- Free List
  - **T**: PR#

No values in ROB, RS, or on CDB

R10K Data Structures

ROB							
ht	#	Insn	T	Told	S	X	C
	1	`ldf X(r1),f1`					
	2	`mulf f0,f1,f2`					
	3	`stf f2,Z(r1)`					
	4	`addi r1,4,r1`					
	5	`ldf X(r1),f1`					
	6	`mulf f0,f1,f2`					
	7	`stf f2,Z(r1)`					

Map Table	
Reg	T+
f0	PR#1+
f1	PR#2+
f2	PR#3+
r1	PR#4+

Arch. Map	
Reg	T+
f0	PR#1
f1	PR#2
f2	PR#3
r1	PR#4

Free List
PR#5,PR#6,
PR#7,PR#8

CDB
T

Reservation Stations						
#	FU	busy	op	T	T1	T2
1	ALU	no				
2	LD	no				
3	ST	no				
4	FP1	no				
5	FP2	no				

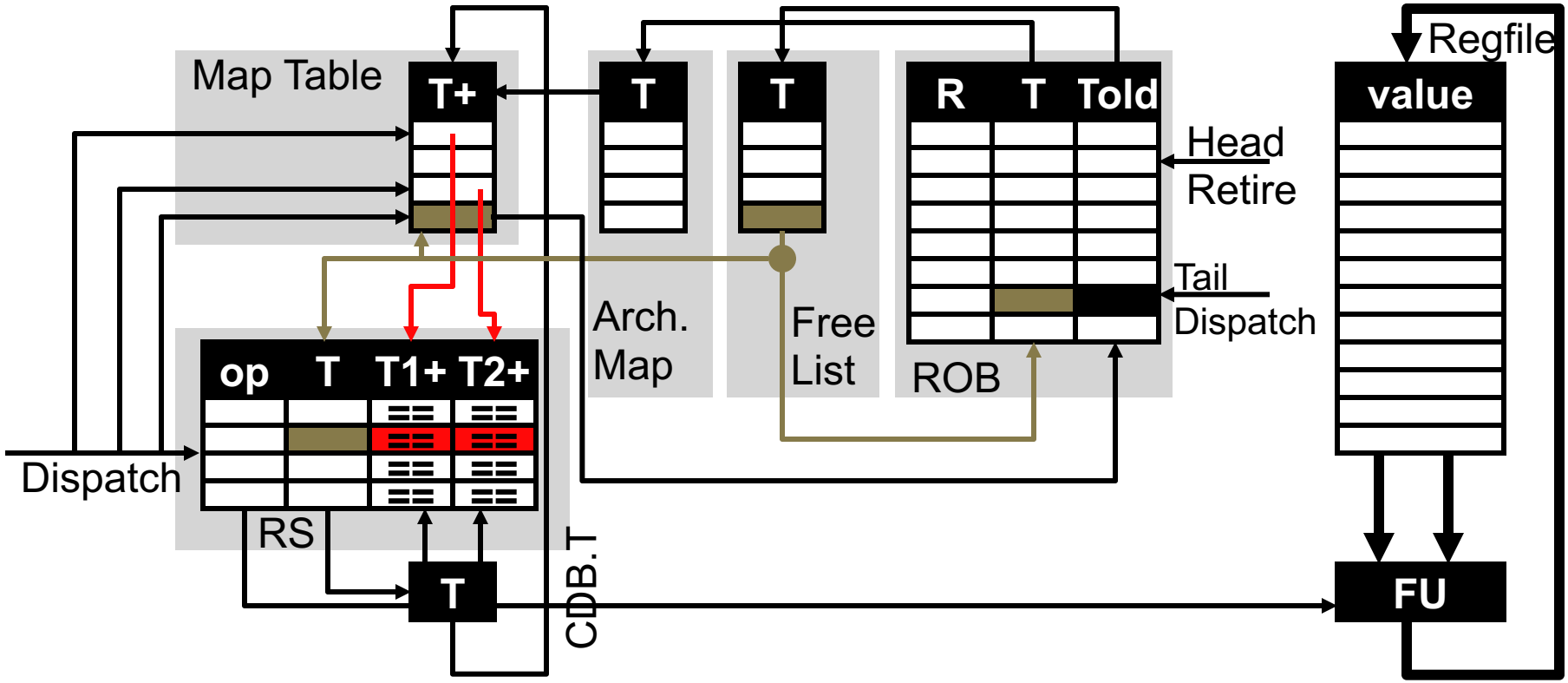
Notice I: no values anywhere

Notice II: MapTable is never empty

R10K Pipeline

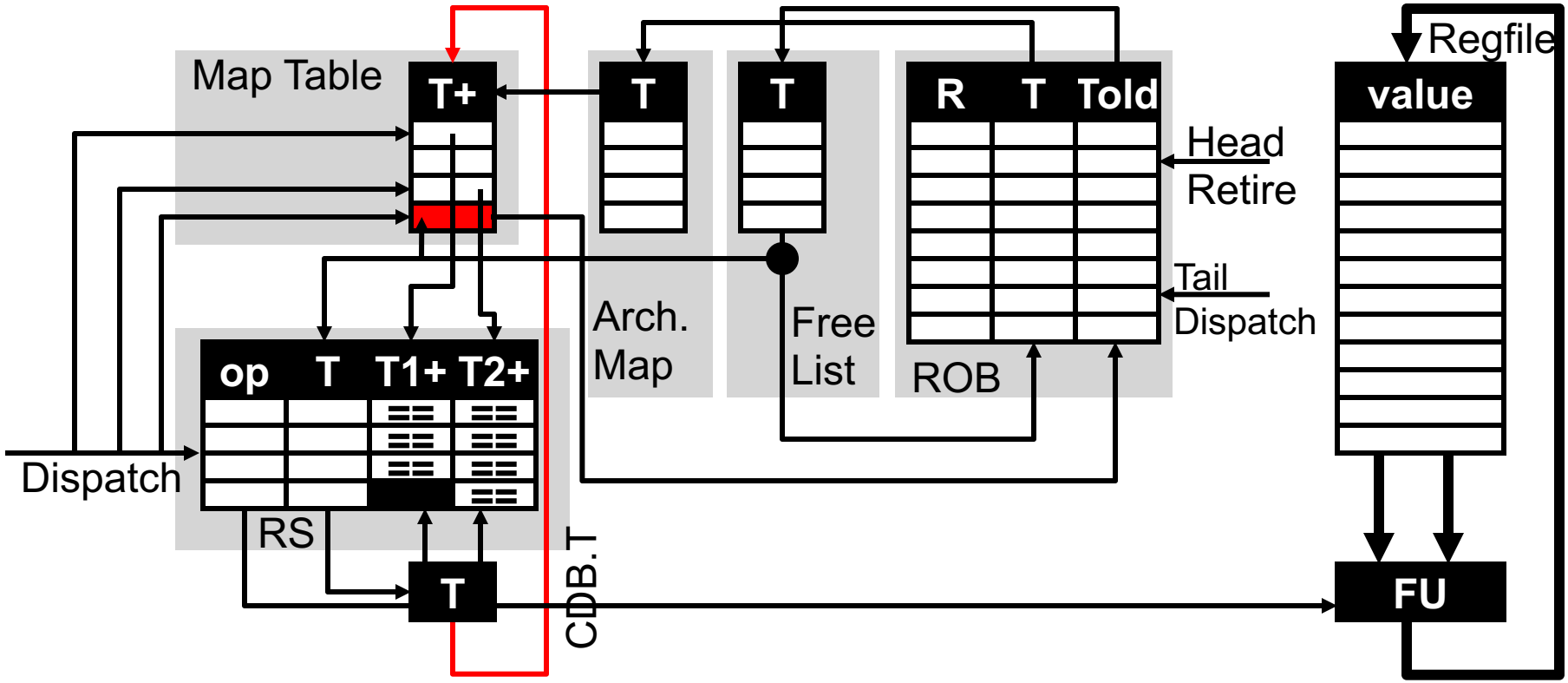
- R10K pipeline structure: F, **D**, S, X, **C**, **R**
  - **D (dispatch)**
    - Structural hazard (RS, ROB, **physical registers**) ? stall
    - Allocate RS, ROB, and new physical register (T)
    - **Record previously mapped physical register (Told)**
  - **C (complete)**
    - Write destination physical register
  - **R (retire)**
    - ROB head not complete ? stall
    - Handle any exceptions
    - Free ROB entry
    - **Free previous physical register (Told)**
    - **Record committed physical register (T)**

R10K Dispatch (D)



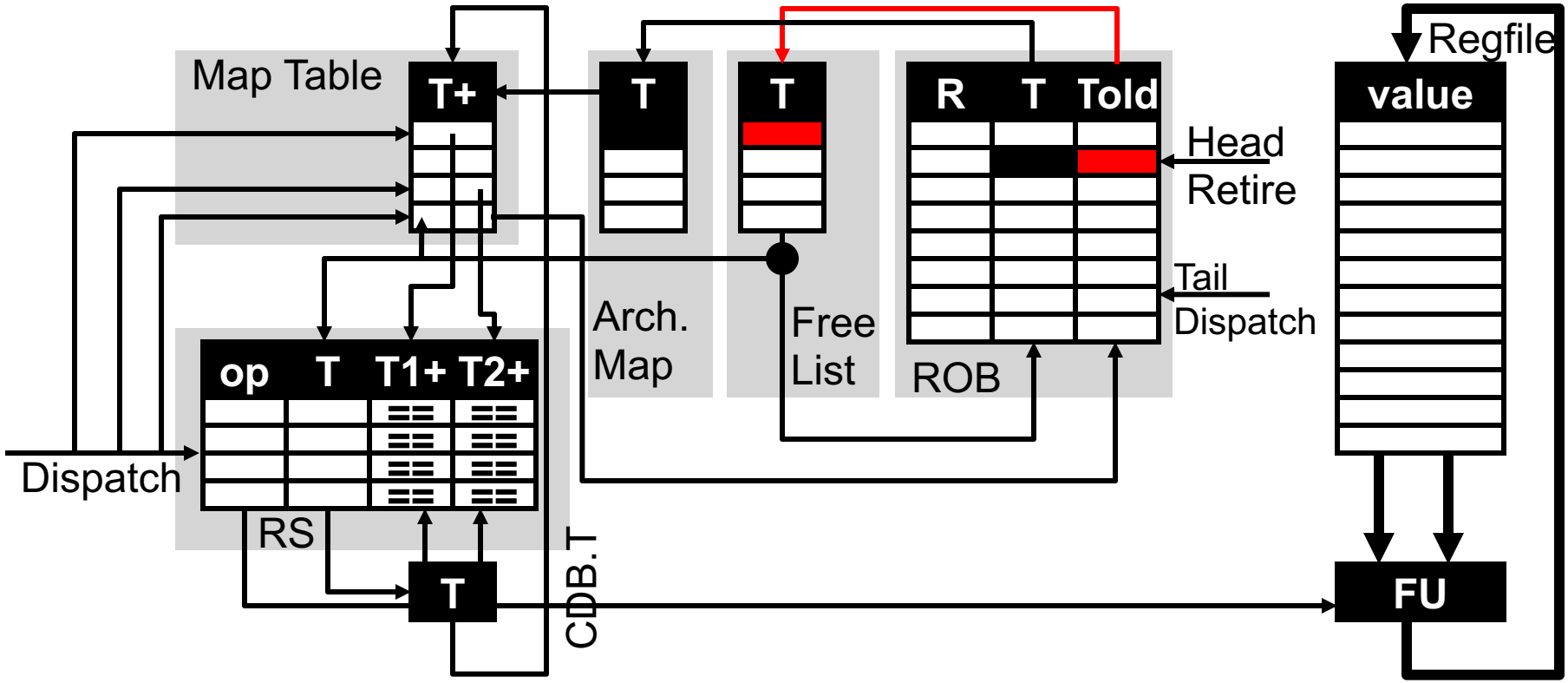
- Read preg (physical register) tags for input registers, store in RS
- Read preg tag for output register, store in ROB (Told)
- Allocate new preg (free list) for output reg, store in RS, ROB, Map Table

R10K Complete (C)



- Set insn's output register ready bit in map table
- Set ready bits for matching input tags in RS

R10K Retire (R)



- Return Told of ROB head to free list
- Record T of ROB head in architectural map table

R10K: Cycle 1

ROB							
ht	#	Insn	T	Told	S	X	C
ht	1	ldf X(r1),f1	PR#5	PR#2			
	2	mulf f0,f1,f2					
	3	stf f2,Z(r1)					
	4	addi r1,4,r1					
	5	ldf X(r1),f1					
	6	mulf f0,f1,f2					
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	PR#1+
f1	PR#5
f2	PR#3+
r1	PR#4+

Arch. Map	
Reg	T+
f0	PR#1
f1	PR#2
f2	PR#3
r1	PR#4

Free List
PR#5,PR#6,
PR#7,PR#8

CDB
T

Reservation Stations						
#	FU	busy	op	T	T1	T2
1	ALU	no				
2	LD	yes	ldf	PR#5		PR#4+
3	ST	no				
4	FP1	no				
5	FP2	no				

Allocate new preg (PR#5) to f1

Remember old preg mapped to f1 (PR#2) in ROB

R10K: Cycle 2

ROB							
ht	#	Insn	T	Told	S	X	C
h	1	ldf X(r1),f1	PR#5	PR#2	c2		
t	2	mulf f0,f1,f2	PR#6	PR#3			
	3	stf f2,Z(r1)					
	4	addi r1,4,r1					
	5	ldf X(r1),f1					
	6	mulf f0,f1,f2					
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	PR#1+
f1	PR#5
f2	PR#6
r1	PR#4+

Arch. Map	
Reg	T+
f0	PR#1
f1	PR#2
f2	PR#3
r1	PR#4

Free List
PR#6,PR#7, PR#8

CDB
T

Reservation Stations						
#	FU	busy	op	T	T1	T2
1	ALU	no				
2	LD	yes	ldf	PR#5		PR#4+
3	ST	no				
4	FP1	yes	mulf	PR#6	PR#1+	PR#5
5	FP2	no				

Allocate new preg (PR#6) to f2

Remember old preg mapped to f3 (PR#3) in ROB

R10K: Cycle 3

ROB							
ht	#	Insn	T	Told	S	X	C
h	1	ldf X(r1),f1	PR#5	PR#2	c2	c3	
	2	mulf f0,f1,f2	PR#6	PR#3			
t	3	stf f2,Z(r1)					
	4	addi r1,4,r1					
	5	ldf X(r1),f1					
	6	mulf f0,f1,f2					
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	PR#1+
f1	PR#5
f2	PR#6
r1	PR#4+

Arch. Map	
Reg	T+
f0	PR#1
f1	PR#2
f2	PR#3
r1	PR#4

Free List
PR#7,PR#8

CDB
T

Reservation Stations						
#	FU	busy	op	T	T1	T2
1	ALU	no				
2	LD	no				
3	ST	yes	stf		PR#6	PR#4+
4	FP1	yes	mulf	PR#6	PR#1+	PR#5
5	FP2	no				

Stores are not allocated pregs

Free

R10K: Cycle 4

ROB							
ht	#	Insn	T	Told	S	X	C
h	1	ldf X(r1),f1	PR#5	PR#2	c2	c3	c4
	2	mulf f0,f1,f2	PR#6	PR#3	c4		
	3	stf f2,Z(r1)					
t	4	addi r1,4,r1	PR#7	PR#4			
	5	ldf X(r1),f1					
	6	mulf f0,f1,f2					
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	PR#1+
f1	PR#5+
f2	PR#6
r1	PR#7

Arch. Map	
Reg	T+
f0	PR#1
f1	PR#2
f2	PR#3
r1	PR#4

Free List
PR#7,PR#8

CDB
T
PR#5

Reservation Stations						
#	FU	busy	op	T	T1	T2
1	ALU	yes	addi	PR#7	PR#4+	
2	LD	no				
3	ST	yes	stf		PR#6	PR#4+
4	FP1	yes	mulf	PR#6	PR#1+	PR#5+
5	FP2	no				

ldf completes
set MapTable ready bit

Match PR#5 tag from CDB & issue

R10K: Cycle 5

ROB							
ht	#	Insn	T	Told	S	X	C
	1	ldf X(r1),f1	PR#5	PR#2	c2	c3	c4
h	2	mulf f0,f1,f2	PR#6	PR#3	c4	c5	
	3	stf f2,Z(r1)					
	4	addi r1,4,r1	PR#7	PR#4	c5		
t	5	ldf X(r1),f1	PR#8	PR#5			
	6	mulf f0,f1,f2					
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	PR#1+
f1	PR#8
f2	PR#6
r1	PR#7

Arch. Map	
Reg	T+
f0	PR#1
f1	PR#5
f2	PR#3
r1	PR#4

Free List
PR#8,PR#2

CDB
T

Reservation Stations						
#	FU	busy	op	T	T1	T2
1	ALU	yes	addi	PR#7	PR#4+	
2	LD	yes	ldf	PR#8		PR#7
3	ST	yes	stf		PR#6	PR#4+
4	FP1	no				
5	FP2	no				

Free

ldf retires
Return PR#2 to free list
Record PR#5 in Arch map

Precise State in R10K

- Precise state is more difficult in R10K
  - Physical registers are written out-of-order (at C)
  - Roll back the Map Table, Arch Table, Free List
    - "free" written registers and "restore" old ones
- Two ways of restoring Map Table and Free List
  - Option I: serial rollback using T, T_{old} ROB fields
    - ± Slow, but simple
  - Option II: single-cycle restoration from some checkpoint
    - ± Fast, but checkpoints are expensive
  - Modern processor compromise: **make common case fast**
    - Checkpoint only (low-confidence) branches (frequent rollbacks)
    - Serial recovery for page-faults and interrupts (rare rollbacks)

R10K: Cycle 5 (with precise state)

ROB							
ht	#	Insn	T	Told	S	X	C
	1	ldf X(r1),f1	PR#5	PR#2	c2	c3	c4
h	2	mulf f0,f1,f2	PR#6	PR#3	c4	c5	
	3	stf f2,Z(r1)					
	4	addi r1,4,r1	PR#7	PR#4	c5		
t	5	ldf X(r1),f1	PR#8	PR#5			
	6	mulf f0,f1,f2					
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	PR#1+
f1	PR#8
f2	PR#6
r1	PR#7

CDB
T

Free List
PR#8,PR#2

Reservation Stations						
#	FU	busy	op	T	T1	T2
1	ALU	yes	addi	PR#7	PR#4+	
2	LD	yes	ldf	PR#8		PR#7
3	ST	yes	stf		PR#6	PR#4+
4	FP1	no				
5	FP2	no				

undo insns 3-5
(doesn't matter why)
use serial rollback

R10K: Cycle 6 (with precise state)

ROB							
ht	#	Insn	T	Told	S	X	C
	1	ldf X(r1),f1	PR#5	PR#2	c2	c3	c4
h	2	mulf f0,f1,f2	PR#6	PR#3	c4	c5	
	3	stf f2,Z(r1)					
t	4	addi r1,4,r1	PR#7	PR#4	c5		
	5	ldf X(r1),f1	~~PR#8~~	PR#5			
	6	mulf f0,f1,f2					
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	PR#1+
f1	PR#5+
f2	PR#6
r1	PR#7

CDB
T

Free List
PR#2, PR#8

Reservation Stations						
#	FU	busy	op	T	T1	T2
1	ALU	yes	addi	PR#7	PR#4+	
2	LD	no				
3	ST	yes	stf		PR#6	PR#4+
4	FP1	no				
5	FP2	no				

undo ldf (ROB#5)
1. free RS
2. free T (PR#8), return to FreeList
3. restore MT[f1] to Told (PR#5)
4. free ROB#5

insns may execute during rollback (not shown)

R10K: Cycle 7 (with precise state)

ROB							
ht	#	Insn	T	Told	S	X	C
	1	ldf X(r1),f1	PR#5	PR#2	c2	c3	c4
h	2	mulf f0,f1,f2	PR#6	PR#3	c4	c5	
t	3	stf f2,Z(r1)					
	4	addi r1,4,r1	PR#7	PR#4	c5		
	5	ldf X(r1),f1					
	6	mulf f0,f1,f2					
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	PR#1+
f1	PR#5+
f2	PR#6
r1	PR#4+

CDB
T

Free List
PR#2,PR#8, PR#7

Reservation Stations						
#	FU	busy	op	T	T1	T2
1	ALU	no				
2	LD	no				
3	ST	yes	stf		PR#6	PR#4+
4	FP1	no				
5	FP2	no				

undo addi (ROB#4)
1. free RS
2. free T (PR#7), return to FreeList
3. restore MT[r1] to Told (PR#4)
4. free ROB#4

R10K: Cycle 8 (with precise state)

ROB							
ht	#	Insn	T	Told	S	X	C
	1	ldf X(r1),f1	PR#5	PR#2	c2	c3	c4
ht	2	mulf f0,f1,f2	PR#6	PR#3	c4	c5	
	3	stf f2,Z(r1)					
	4	addi r1,4,r1					
	5	ldf X(r1),f1					
	6	mulf f0,f1,f2					
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	PR#1+
f1	PR#5+
f2	PR#6
r1	PR#4+

CDB
T

Free List
PR#2,PR#8, PR#7

Reservation Stations						
#	FU	busy	op	T	T1	T2
1	ALU	no				
2	LD	no				
3	ST	no				
4	FP1	no				
5	FP2	no				

undo stf (ROB#3)
1. free RS
2. free ROB#3
3. no registers to restore/free
4. how is L1-D write undone?

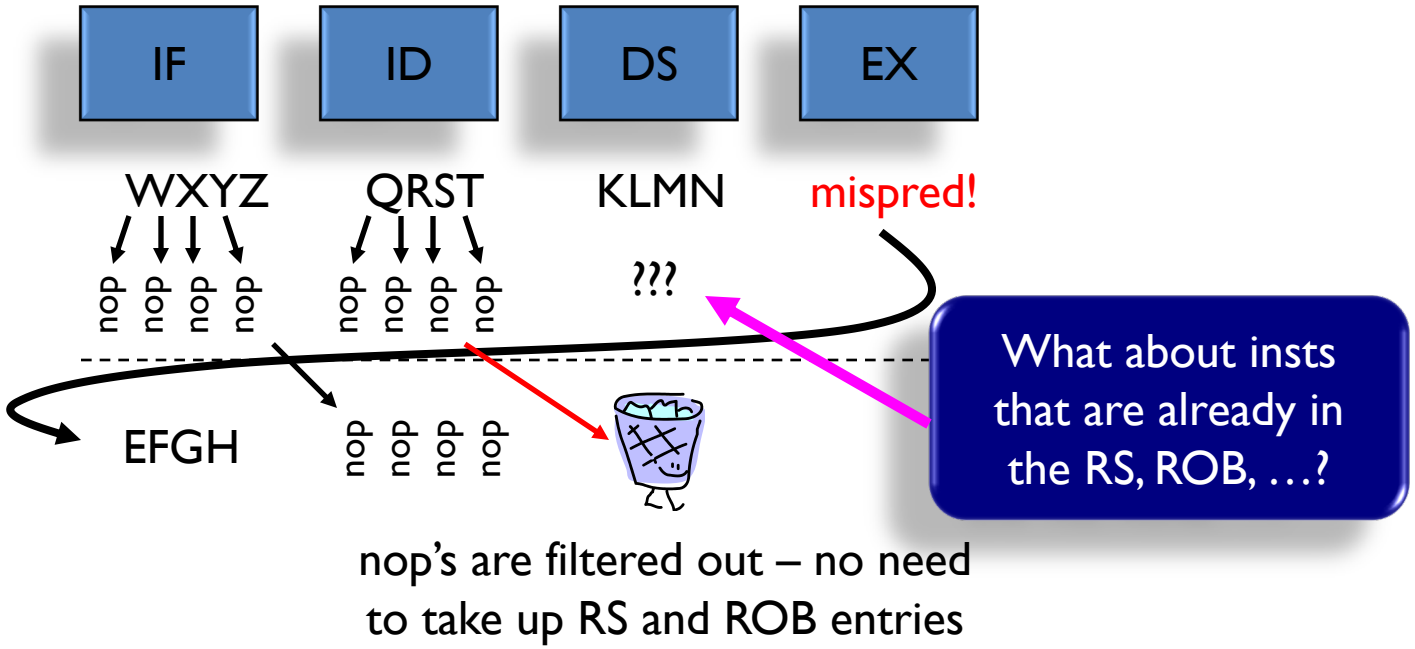
P6 vs. R10K (Renaming)

Feature	P6	R10K
Value storage	ARF,ROB,RS	PRF
Register read	@D: ARF/ROB → RS	@S: PRF → FU
Register write	@R: ROB → ARF	@C: FU → PRF
Speculative value free	@R: automatic (ROB)	@R: overwriting insn
Data paths	ARF/ROB → RS RS → FU FU → ROB ROB → ARF	PRF → FU FU → PRF
Precise state	Simple: clear everything	Complex: serial/checkpoint

- R10K-style became popular in late 90's, early 00's
  - E.g., MIPS R10K (duh), DEC Alpha 21264, Intel Pentium4
- P6-style is making a comeback
  - Why? Frequency (power) is on the retreat, simplicity is important

Speculation Recovery

- Squashing instructions in front-end pipeline

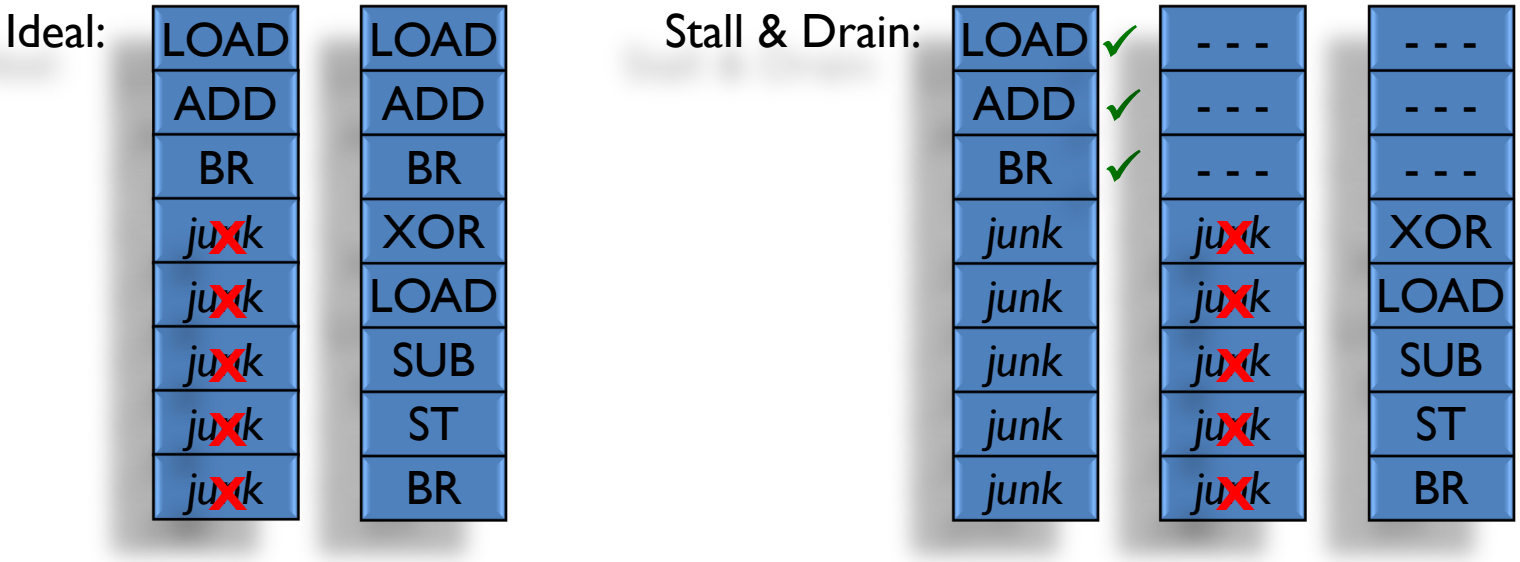


Stall and Drain (1/2)

- Squash in-order front-end (as before)
- Stall dispatch (no new instructions → ROB, RS)
- Let OoO engine execute as usual
- Let commit operate as usual except:
  - Check for the mispredicted branch
    - Cannot commit any instructions after it
      - Any insns. in pipeline are on the wrong path
    - Flush the OoO engine
    - Allow dispatch to continue

Stall and Drain (2/2)

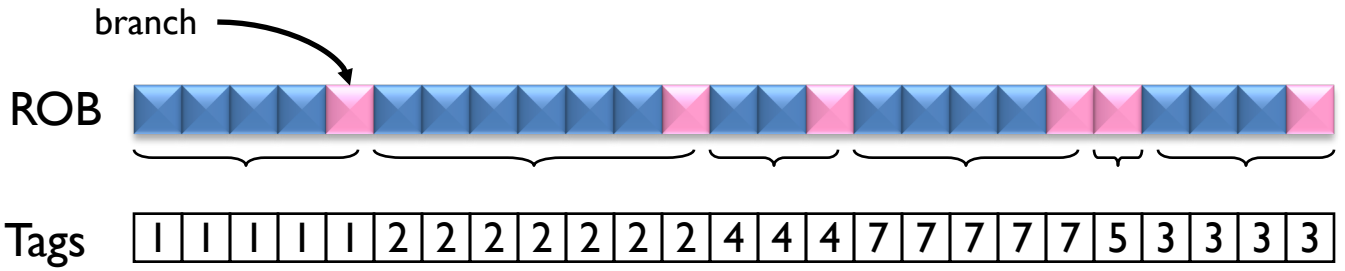
- Delays recovery until BR retires



Simple to build, but low performance

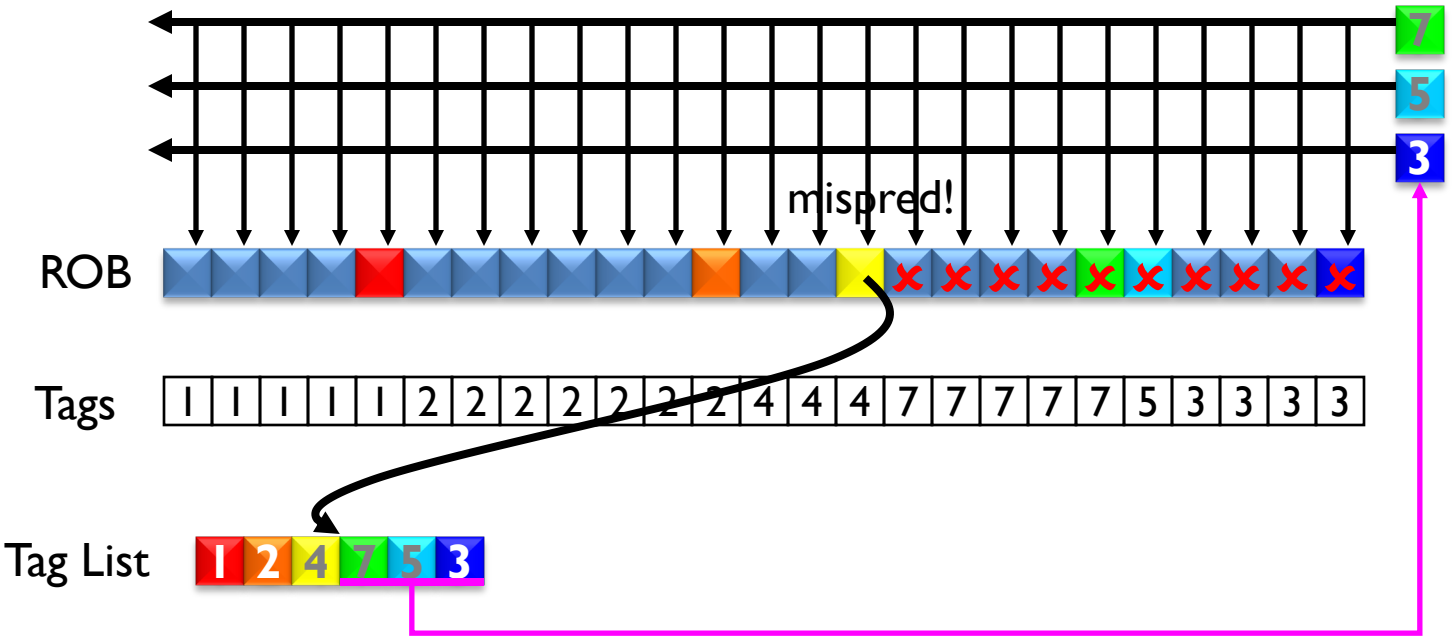
Branch Tags/Colors (1/2)

- Each insn. assigned the “current” branch tag
- Each predicted branch allocates a new branch tag
  - Newly allocated tag becomes current



(Tags might not necessarily be in any particular order)

Branch Tags/Colors (2/2)

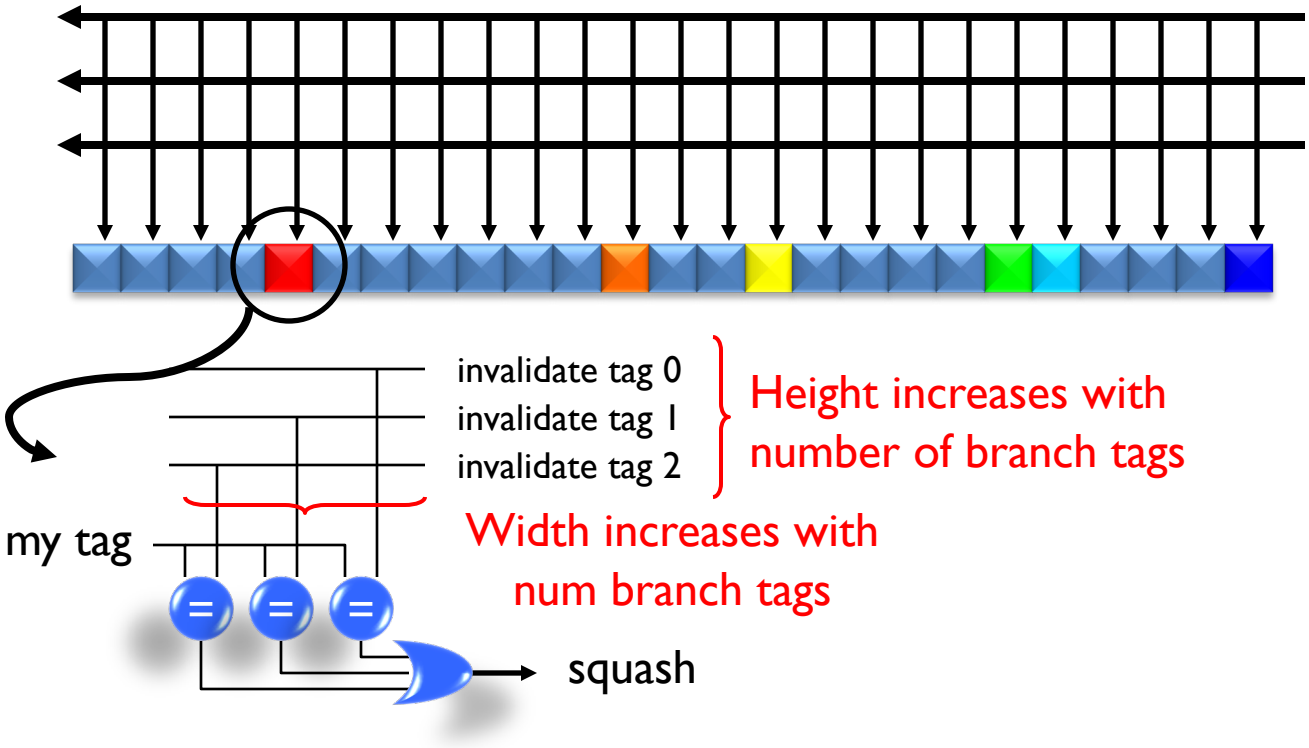


Branch Tags for RS, overkill for ROB

- ROB keeps insns. in program order
- Squash all insns. ***after*** mispredicted branch

- Tagging/coloring useful for RS
  - Insns. in RS are in arbitrary order
  - May be organized into multiple sets of RSs

Hardware Complexity



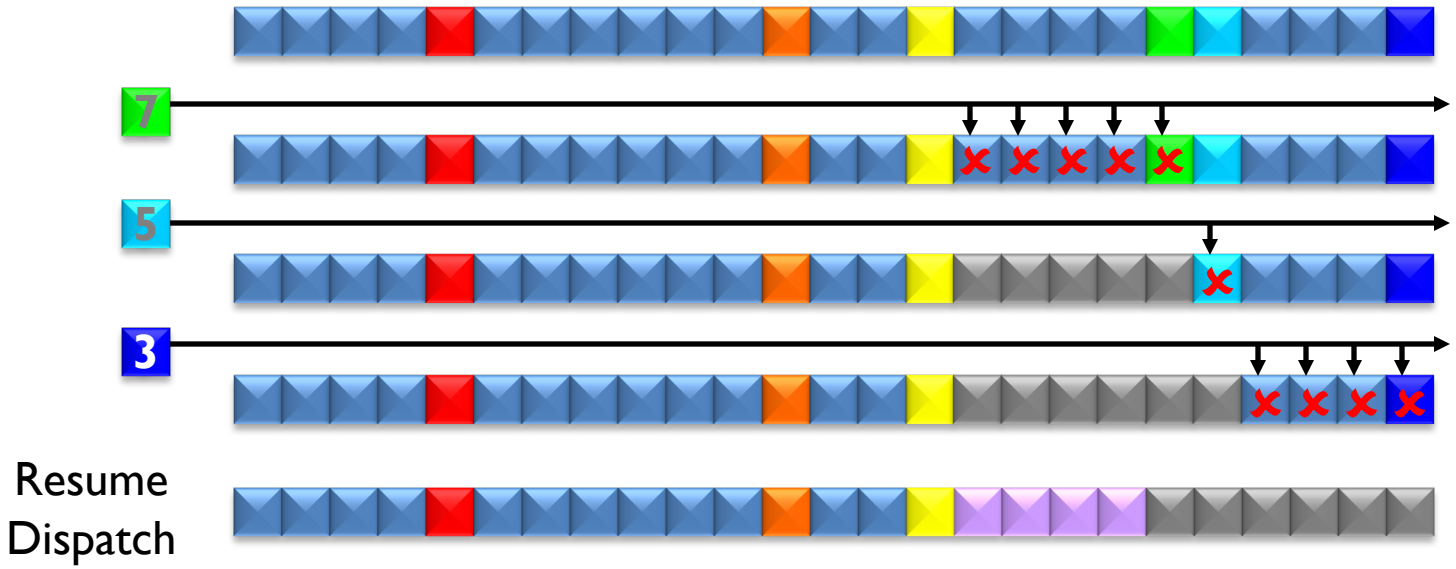
Area overhead is *quadratic* in tag count

Squash Simplifications (1/2)

- For *n*-entry ROB, could have *n* different branches
- In practice, only a fraction of insns. are branches
  - Limit to $k < n$ tags instead
  - If $k+1^{st}$ branch is fetched, stall dispatch (structural hazard)

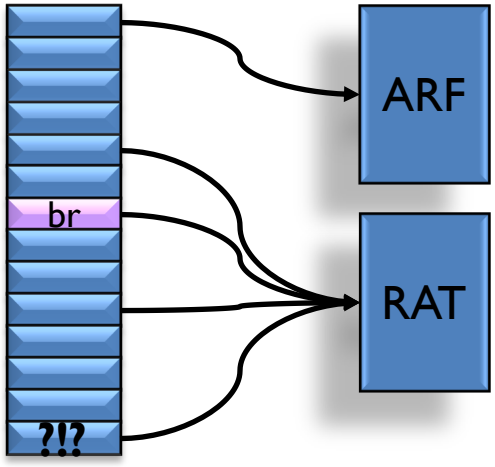
Squash Simplifications (2/2)

- For *k* tags, need to broadcast all younger tags
  - Results in O(k^2) overhead
- Limit to few (e.g., one) broadcast per cycle



Can fetch and decode while squashing in back-end

Register Speculation Recovery



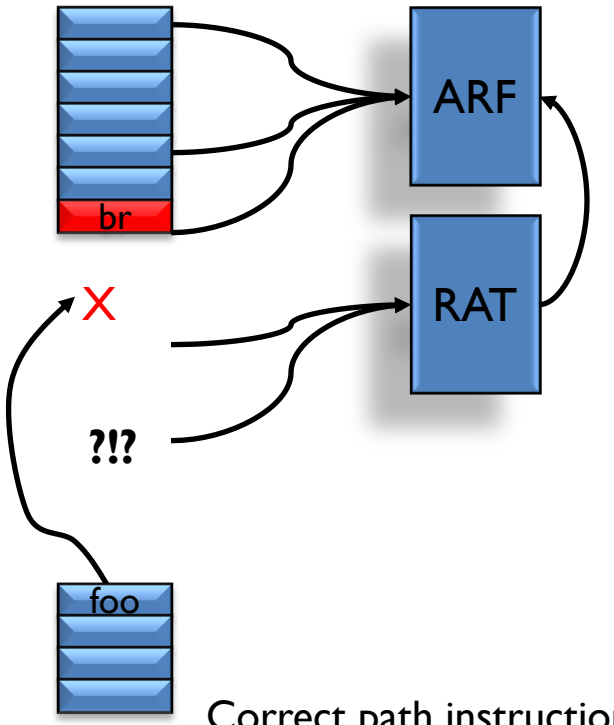
ARF state corresponds to state prior to oldest non-committed instruction

As instructions are processed, the RAT corresponds to the register mapping after the most recently renamed instruction

On a branch misprediction, wrong-path instructions are flushed from the machine

RAT left in invalid state

Solution 1: Stall and Drain



Allow all instructions to execute and commit; ARF corresponds to last comitted instruction

ARF now corresponds to the state right before the next instruction to be renamed (foo)

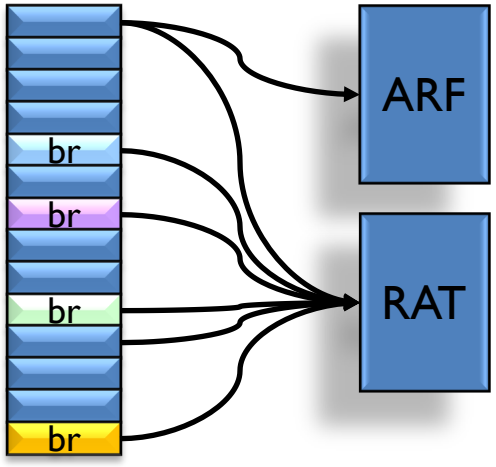
Reset RAT so that all mappings refer to the ARF

Resume renaming the new correct-path instructions from fetch

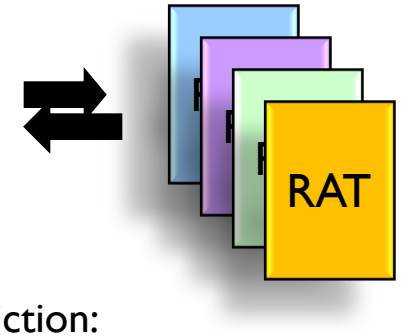
Correct path instructions from fetch; can't rename because RAT is wrong

Simple to build, but low performance

Solution 2: Checkpointing (1/2)



At each branch, make a copy of the RAT
(register mapping at the time of the branch)



Checkpoint
Free Pool

On a misprediction:

1. flush wrong-path instructions
2. deallocate RAT checkpoints
3. recover RAT from checkpoint
4. resume renaming

Squash tags/colors can be same as checkpoints

Solution 2: Checkpointing (2/2)

- No need to stall front-end
  - Need to “flash copy” RAT
    - Both for making checkpoints and recovering
  - Need to recover wrong-path checkpoints
- More hardware
  - Need one checkpoint per branch
  - What if the code has all branches?
    - Stall front-end when out of branch colors/checkpoints

Solution 3: Undo List

- Each ROB entry tracks two physical registers
  - Its destination register
  - The previous physical register mapping
    - Required for R10K-style OoO anyway
- Walk ROB backwards, applying the old mappings
  - Low overhead: don't need full copies of the RAT
  - Slower: need to walk the ROB
  - Flexibility: can recover to any instruction
- Can combine with checkpointing
  - Checkpoint low-confidence branches; walk ROB for others