Dynamic Scheduling II

- so far: dynamic scheduling (out-of-order execution)
  - Scoreboard
  - Tomasulo's algorithm
  - register renaming: removing artificial dependences (WAR/WAW)
- now: out-of-order execution + precise state
- advanced topic: dynamic load scheduling
- PentiumII vs. Pentium4
- limits of ILP

Readings

H+P

- chapter 2

Research Papers

- Pentium4
- Complexity-Effective Superscalar
- Checkpoint Processing and Recovery

Superscalar + Out-of-Order + Speculation

superscalar + out-of-order + speculation

- three concepts that work well (best?) when used together
- CPI >= 1?
  - overcome with superscalar
- superscalar increases hazards?
  - overcome with dynamic scheduling
- RAW dependences still a problem?
  - overcome with a large instruction window
- branches a problem for filling large window?
  - overcome with speculation

Speculation and Precise Interrupts

Q: why are we discussing these together?

- sequential (von Neumann) semantics for interrupts
  - all instructions before interrupt should be complete
  - all instructions after interrupt should look as if never started (abort)
  - basically, we also want the same thing for a mis-predicted branch
- what makes precise interrupts hard?
  - out-of-order completion ⇒ must undo post-interrupt writebacks
  - in-order pipe ⇒ no post-branch writebacks before branch completes
  - out-of-order pipe ⇒ *can* happen

A: with out-of-order pipe, precise interrupts and mis-speculation recovery are same problem ⇒ same solution

Solution: Precise State

- speculative execution requirements
  - ability to abort & restart at every branch
- precise synchronous interrupt requirements
  - ability to abort & restart at every load, store, FP divide, ??
- precise asynchronous interrupt requirements
  - ability to abort & restart at every ??
- just bite the bullet
  - implement ability to abort & restart at every instruction
  - called "precise state"

Ways to Implement Precise State

- force in-order completion (WB): stall pipe if necessary
  – slow
- precise state in software
  – even slower - would require a trap for every misprediction
- precise state in hardware: save recovery info internally
  + everything is better in hardware

The Problem with Precise State

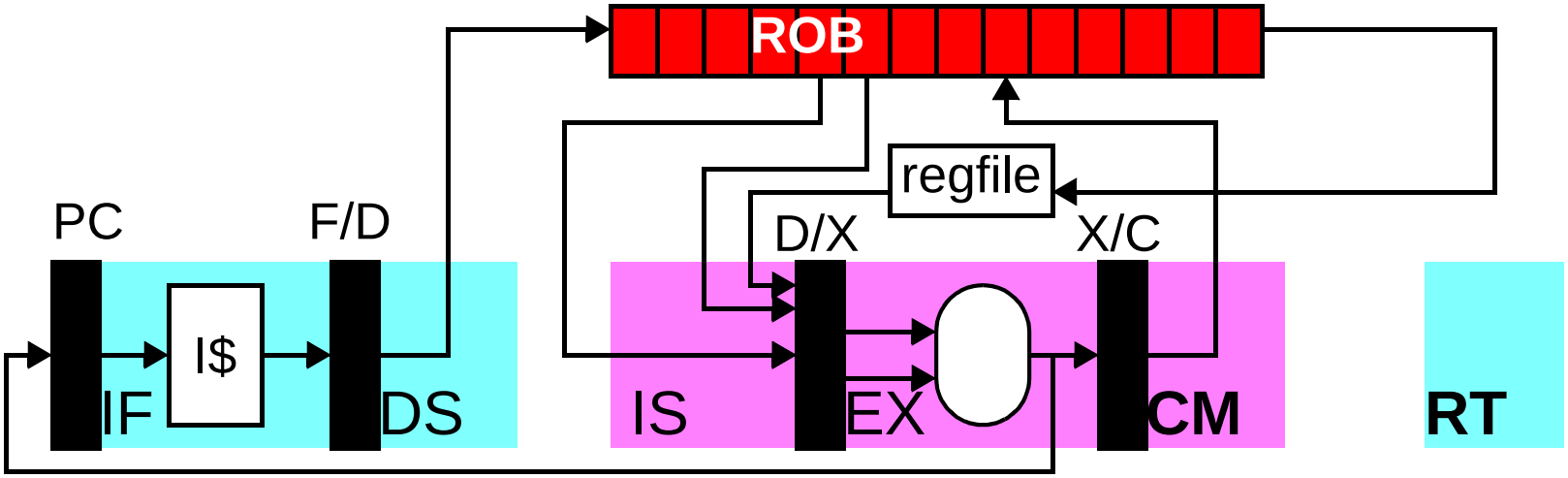
problem is in the writeback stage (WB)

- mixes two things together that should be separate
- (1) broadcasts values to RS, forwards to other instructions
  - OK for this to be out-of-order
- (2) writes values to registers
  - would like this to be in-order

solution to every functionality problem? add a level of indirection

- have already seen this for out-of-order execution
  - split ID into in-order DS and out-of-order IS
  - separate using instruction buffer (scoreboard, reservation stations)

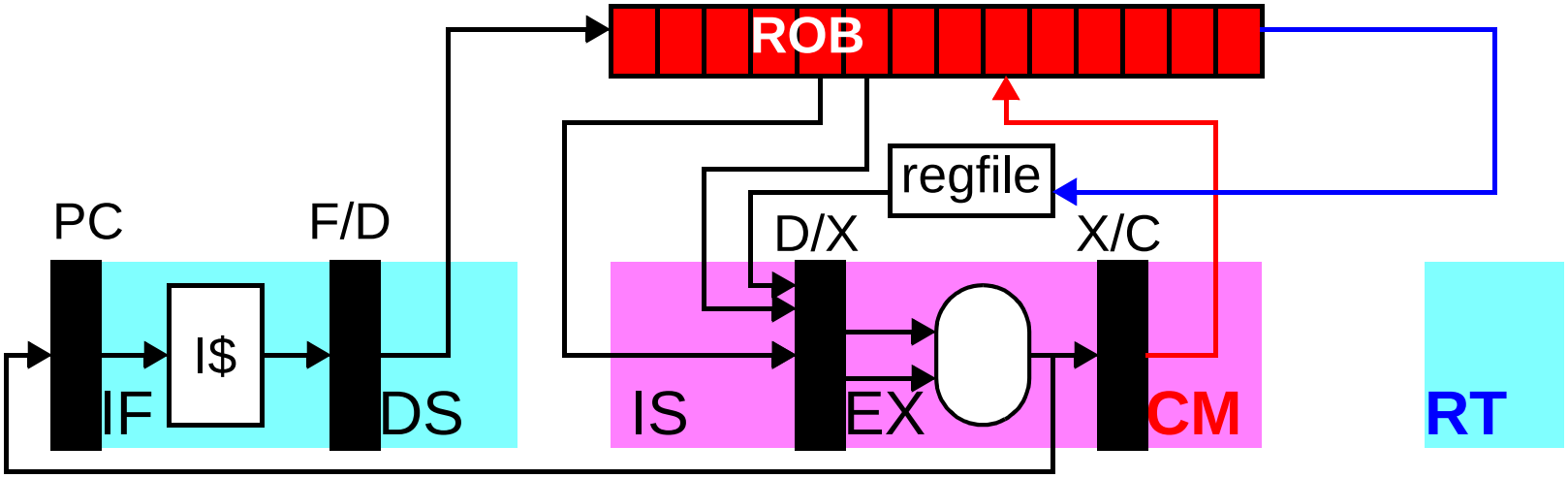
Re-Order Buffer (ROB)



instruction buffer ⇒ re-order buffer (ROB)

- buffers completed results en route to register file and D$
  - may be combined with RS or separate (combined in the picture)
- split writeback (WB) into two stages: Complete and Retire

Complete and Retire



- *CM (complete)*
  - completed values write results to ROB out-of-order
  - out-of-order stage
- *RT (retire, but sometimes called "commit" or "graduate")*
  - ROB writes results to register file in-order
  - in-order stage ⇒ hazards result in stalls

Memory Ordering Buffer (MOB)

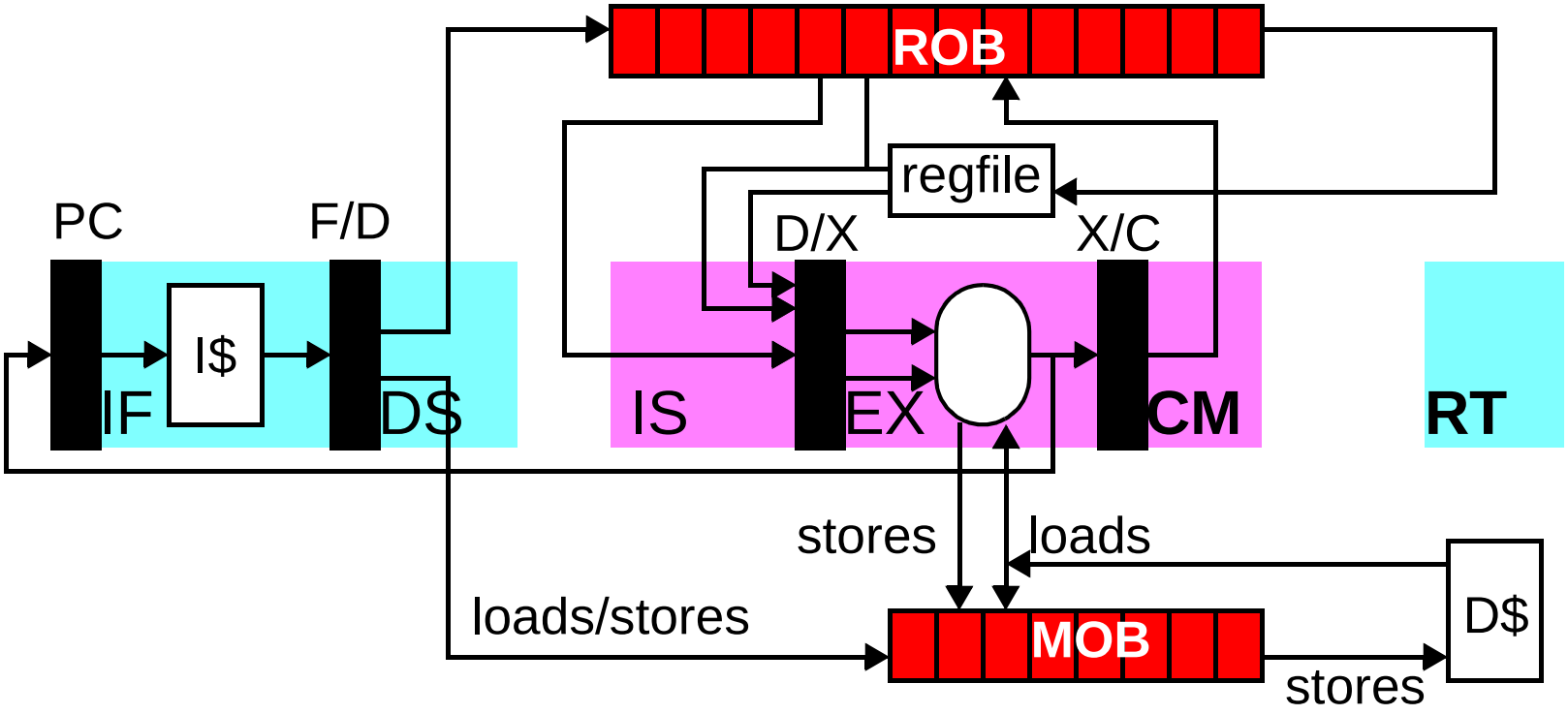
ROB makes register writes in-order, but what about stores?

- same as before (i.e., to D$ in MEM stage)?
  - bad idea! imprecise memory worse than imprecise registers
  - must do same trick for stores

Memory Ordering Buffer (MOB)

- a.k.a. store buffer, store queue, load/store queue (LSQ)
- completed (but not retired) stores write to MOB
- to retire store, write head of MOB to D$
- loads look at MOB and D$ in parallel
  - forward from MOB if matching store (i.e. to same address)

ROB+MOB

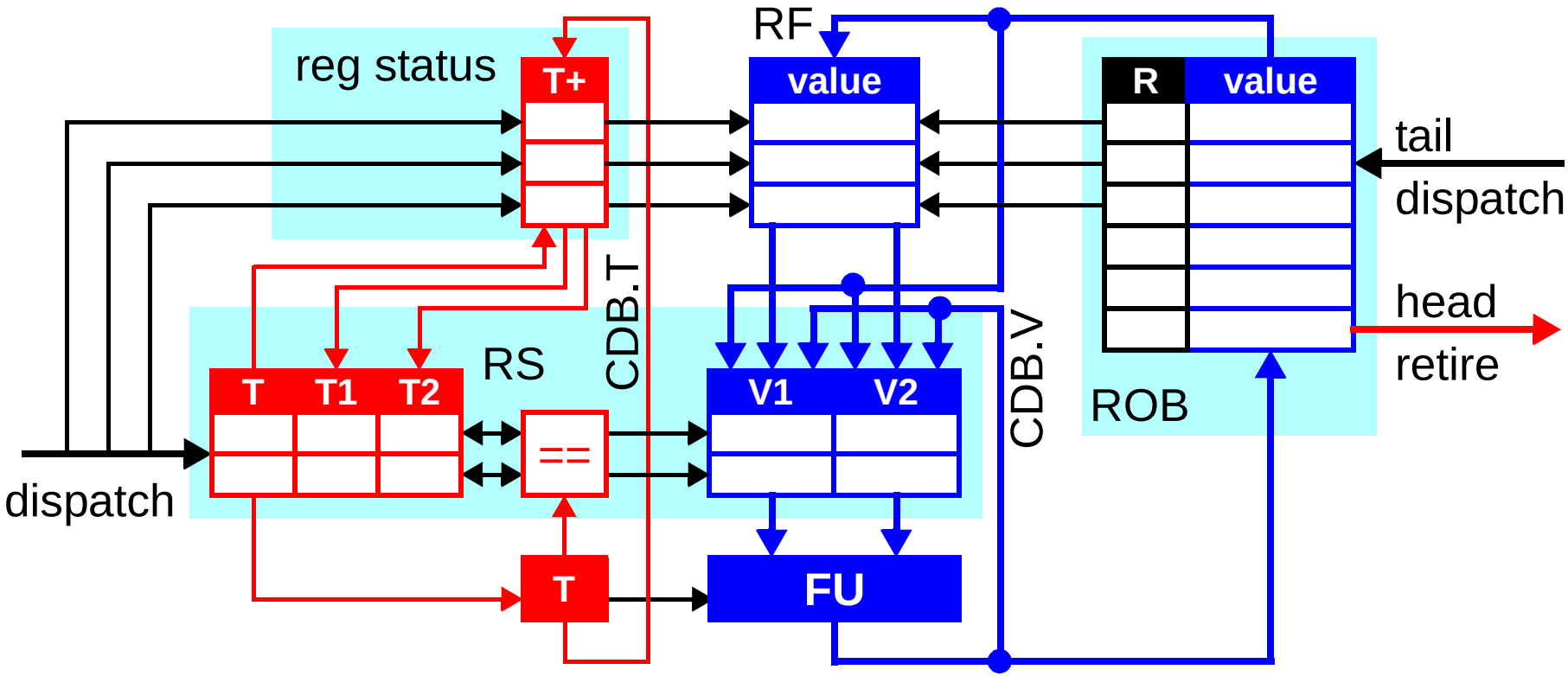


modulo some gross simplifications, this picture is almost realistic!

Tomasulo+ROB

- add ROB to Tomasulo's algorithm
  - combined ROB and RS are called RUU (or Sohi's method)
    - RUU = register update unit
  - separate ROB and RS are called P6-style (Intel P6 = Pentium Pro)
- our example: Simple-P6
  - separate ROB and RS
  - same RS organization as before: 1 ALU, 1 load, 1 store, 2 3-cycle FP

P6-style Organization



- instruction fields and ready bits
- tags
- values

P6 Data Structures

- RS are the same as before
- ROB
  - **head**, **tail**: to keep sequential order
  - **R**: output register of instruction, **V**: output value of instruction
- tags are different
  - was: RS# $\rightarrow$ now: ROB#
- register status table is different
  - **T+**: tag + "ready-in-ROB" bit
  - tag == 0 $\Rightarrow$ result ready in register file
  - tag != 0 $\Rightarrow$ result not ready
  - tag != 0 + $\Rightarrow$ result ready in ROB

P6 Data Structures

hd tl	ROB + MOB							
	#	instruction	R	V	addr	IS	EX	CM
	1	`ldf f0,X(r1)`						
	2	`mulf f4,f0,f2`						
	3	`stf f4,Z(r1)`						
	4	`add r1,r1,#8`						
	5	`ldf f0,X(r1)`						
	6	`mulf f4,f0,f2`						
	7	`stf f4,Z(r1)`						

Reg. Status	
reg	T+
f0	
f2	
f4	
r1	

CDB	
V	T

RS								
#	FU	busy	op	T	V1	V2	T1	T2
1	ALU	No						
2	load	No						
3	store	No						
4	FP1	No						
5	FP2	No						

P6 Pipeline

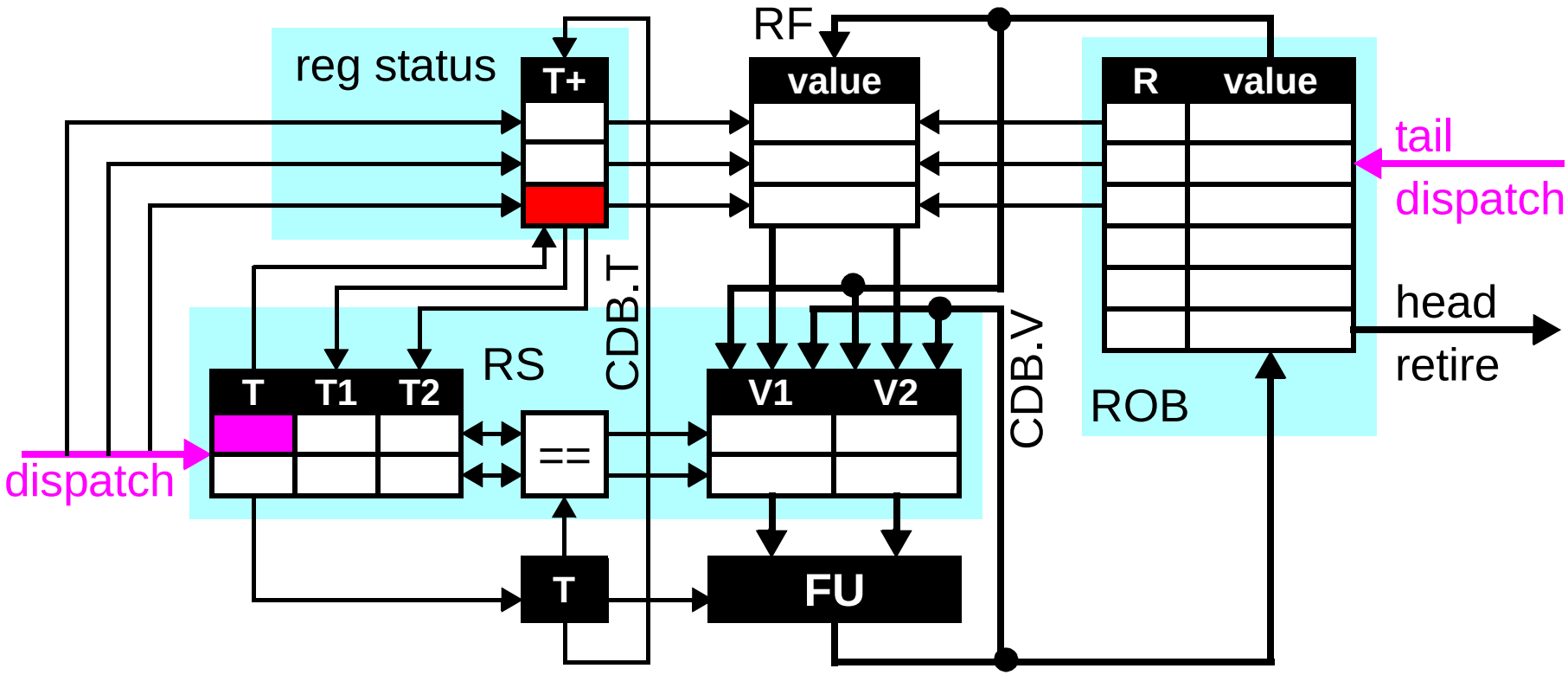
new pipeline structure: IF, *DS*, IS, *EX*, *CM*, *RT*

- *DS (dispatch)*
  - (RS/ROB/MOB full) ? (stall) :
  - {allocate RS/ROB/MOB entries,
    set RS tag to ROB#,
    set register status entry to ROB# with “ready-in-ROB” bit off,
    read ready registers into RS}
- *EX (execute)*
  - free RS entry
  - used to be done at WB
  - can be earlier now because RS# are not tags

P6 Pipeline

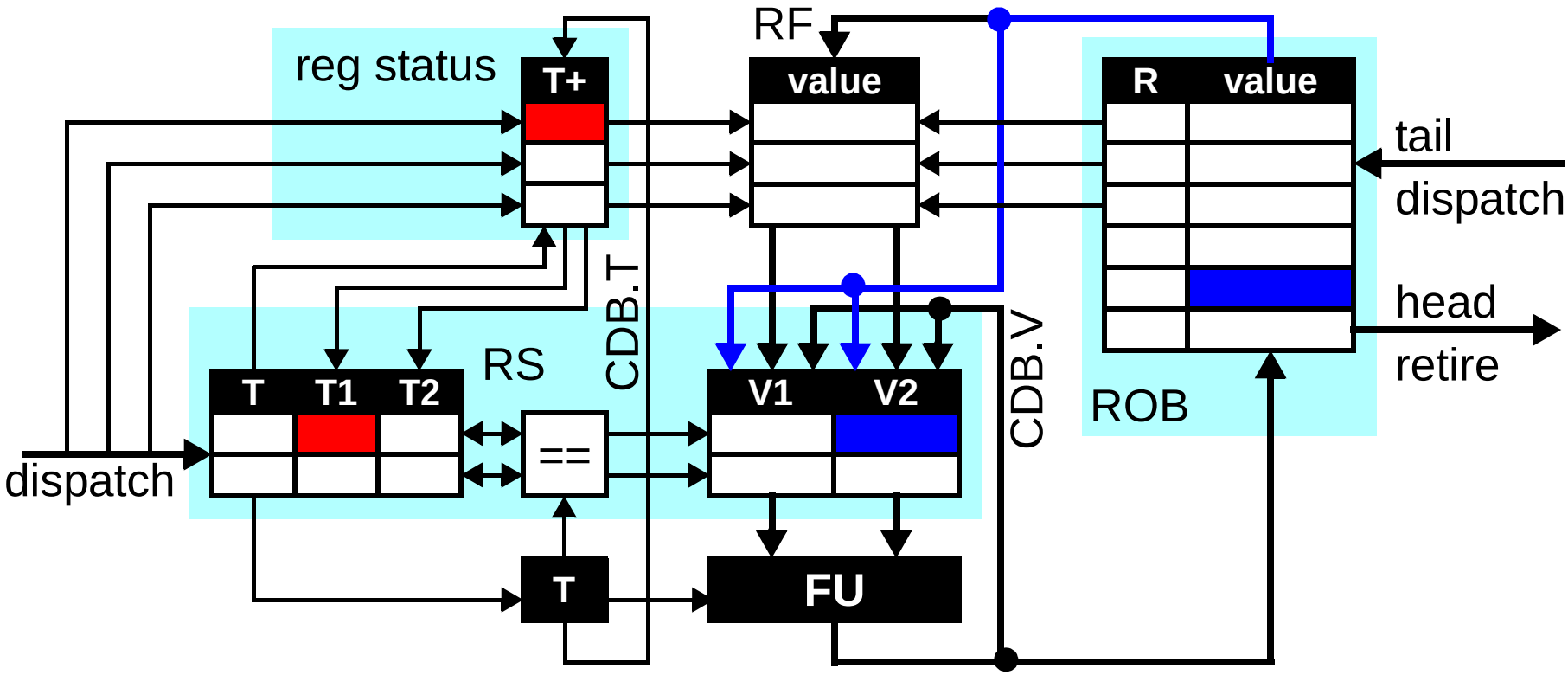
- *CM (complete)*
  - (CDB not available) ? (wait) :
  - {write value into ROB entry indicated by RS tag,
    mark ROB entry complete,
    mark register status entry “ready-in-ROB” bit (+)}
- *RT (retire, commit, graduate)*
  - (ROB head not complete) ? (stall) :
  - {write ROB head result to register file,
    if store, then write MOB head to D$,
    handle any exceptions,
    free ROB/MOB entries}

P6: Dispatch (DS) part I



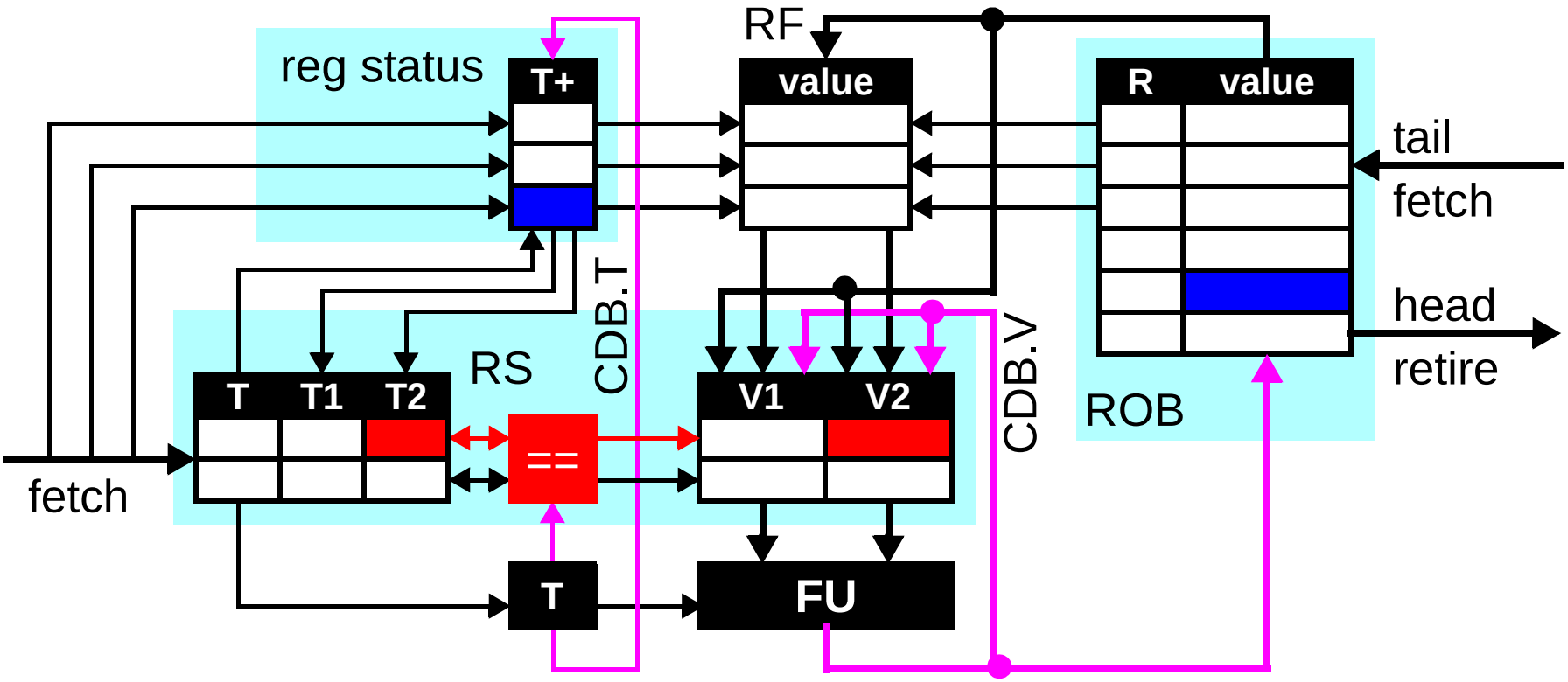
- stall if RS or ROB or MOB is full
  - allocate RS+ROB entries (assign ROB# to RS output tag)
  - set register status entry to ROB# and “ready-in-ROB” bit to 0

P6: Dispatch (DS) part II



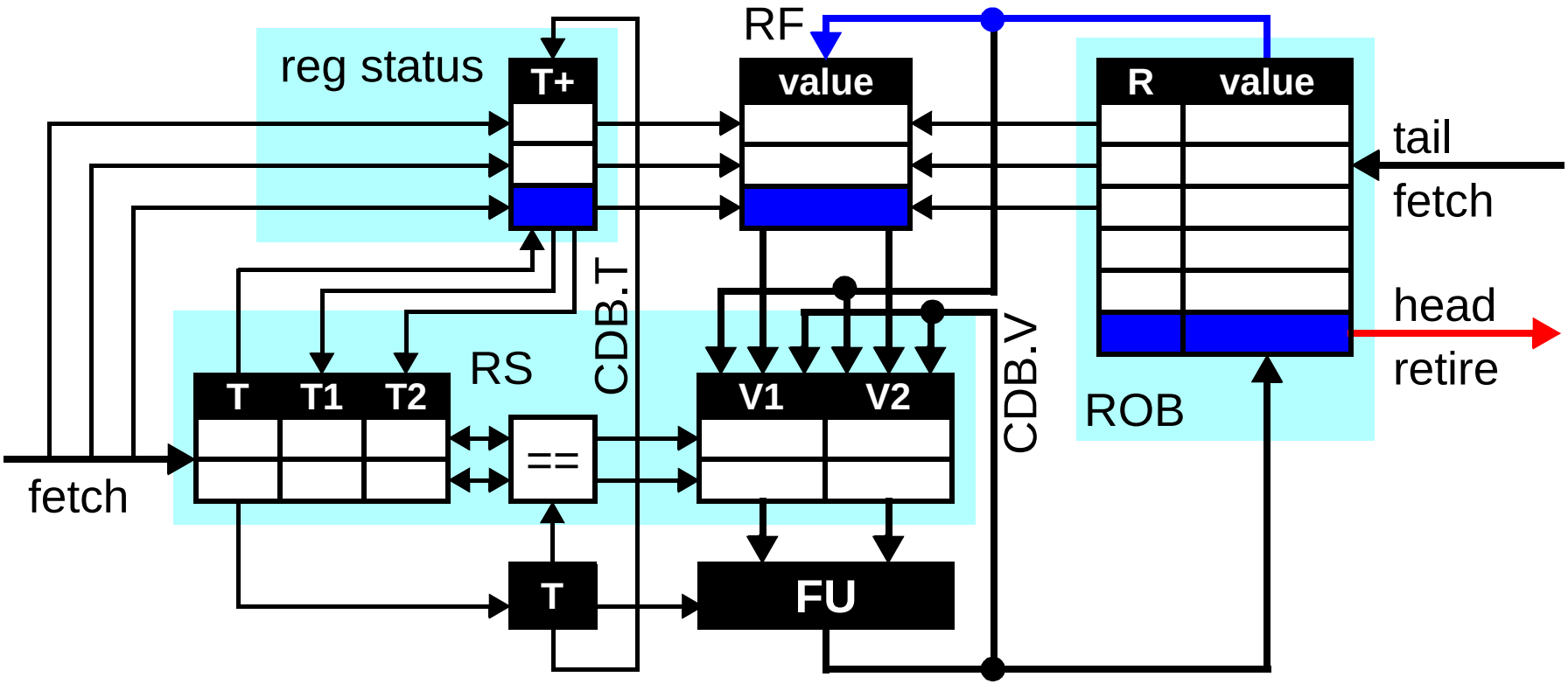
- read tags for register inputs from register status table
  - if tag==0: copy value from RF (not shown)
  - if tag!=0: copy tag to RS
  - if tag!=0 +: copy value from ROB

P6: Complete (CM)



- wait for CDB
  - broadcast <result,tag> on CDB
  - write result into ROB, set reg. status "ready-in-ROB" bit (+)
  - match tags, write CDB.V into RS of dependent instructions

P6: Retire (RT)



- stall until instruction at ROB head has completed
  - write ROB head result to reg-file (D$ if store), clear reg. status entry
  - free ROB entry

P6 Example: Cycle 1

hd tl	ROB + MOB							
	#	instruction	R	V	addr	IS	EX	CM
ht	1	`ldf f0,X(r1)`	f0		&X[0]			
	2	`mulf f4,f0,f2`						
	3	`stf f4,Z(r1)`						
	4	`add r1,r1,#8`						
	5	`ldf f0,X(r1)`						
	6	`mulf f4,f0,f2`						
	7	`stf f4,Z(r1)`						

Reg. Status	
reg	T+
f0	ROB#1
f2	
f4	
r1	

CDB	
V	T

before ROB, this was RS#2

RS								
#	FU	busy	op	T	V1	V2	T1	T2
1	ALU	No						
2	load	Yes	ldf	ROB#1		REG[r1]		
3	store	No						
4	FP1	No						
5	FP2	No						

allocate RS
set reg. status

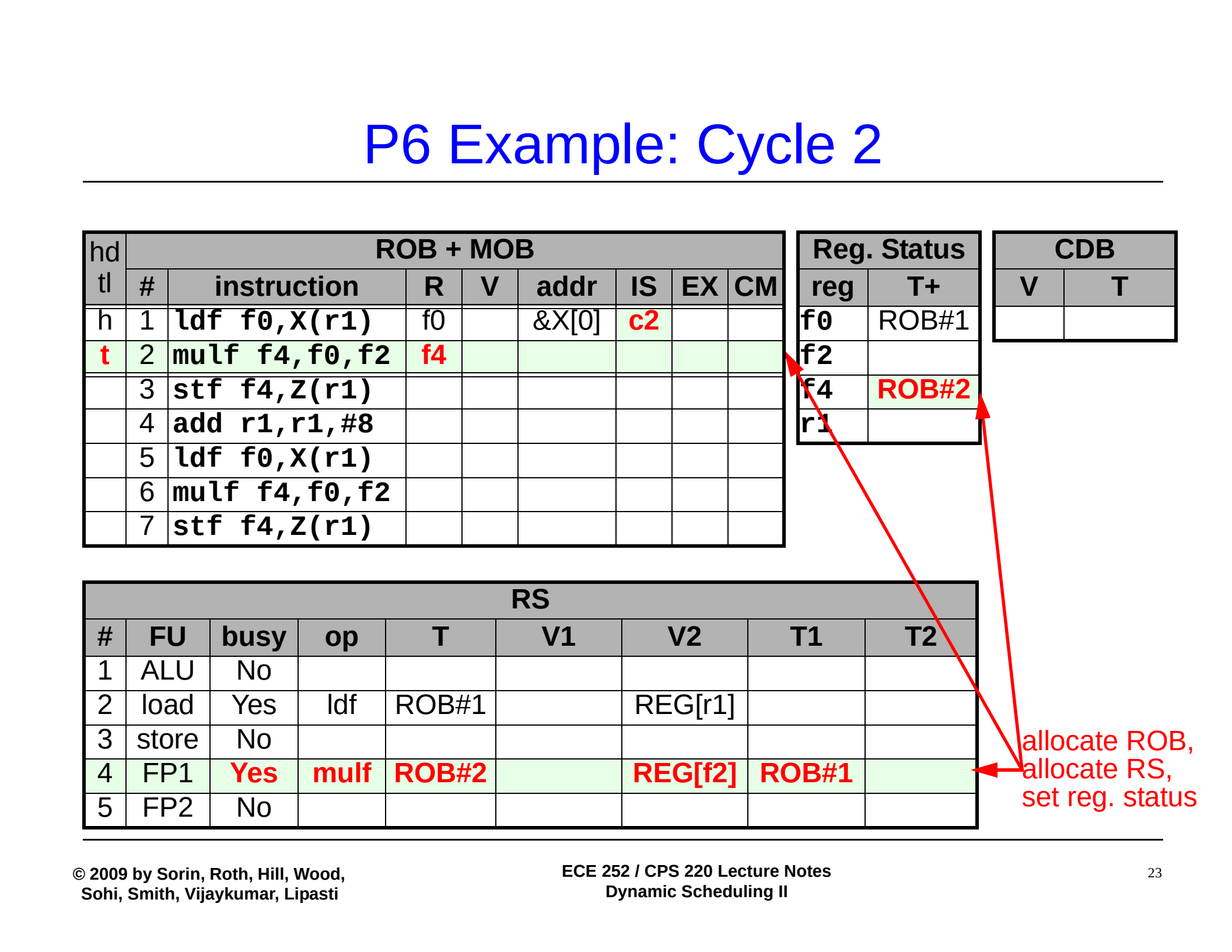
P6 Example: Cycle 2

hd / tl	#	ROB + MOB instruction	R	V	addr	IS	EX	CM
h	1	`ldf f0,X(r1)`	f0		&X[0]	c2		
t	2	`mulf f4,f0,f2`	f4					
	3	`stf f4,Z(r1)`						
	4	`add r1,r1,#8`						
	5	`ldf f0,X(r1)`						
	6	`mulf f4,f0,f2`						
	7	`stf f4,Z(r1)`						

Reg. Status	
reg	T+
f0	ROB#1
f2	
f4	ROB#2
r1	

CDB	
V	T

RS								
#	FU	busy	op	T	V1	V2	T1	T2
1	ALU	No						
2	load	Yes	ldf	ROB#1		REG[r1]		
3	store	No						
4	FP1	Yes	mulf	ROB#2		REG[f2]	ROB#1	
5	FP2	No						



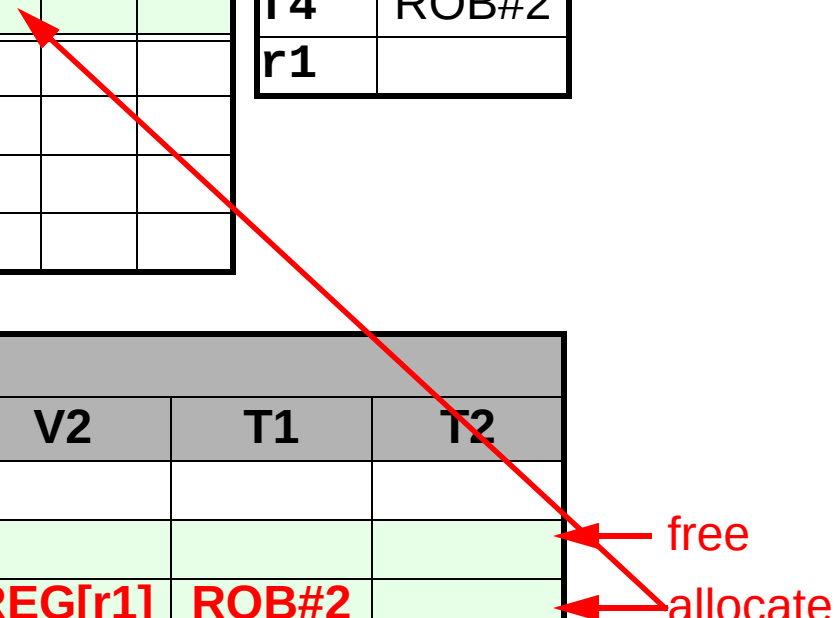
P6 Example: Cycle 3

hd tl	#	ROB + MOB instruction	R	V	addr	IS	EX	CM
h	1	`ldf f0,X(r1)`	f0		&X[0]	c2	**c3**	
	2	`mulf f4,f0,f2`	f4					
t	3	`stf f4,Z(r1)`			**&Z[0]**			
	4	`add r1,r1,#8`						
	5	`ldf f0,X(r1)`						
	6	`mulf f4,f0,f2`						
	7	`stf f4,Z(r1)`						

Reg. Status	
reg	T+
`f0`	ROB#1
`f2`	
`f4`	ROB#2
`r1`	

CDB	
V	T

RS								
#	FU	busy	op	T	V1	V2	T1	T2
1	ALU	No						
2	load	**No**						
3	store	**Yes**	**stf**	**ROB#3**		**REG[r1]**	**ROB#2**	
4	FP1	Yes	mulf	ROB#2		REG[f2]	ROB#1	
5	FP2	No						



P6 Example: Cycle 4

hd tl	ROB + MOB							
	#	instruction	R	V	addr	IS	EX	CM
h	1	`ldf f0,X(r1)`	f0	[f0]	&X[0]	c2	c3	c4
	2	`mulf f4,f0,f2`	f4			c4		
	3	`stf f4,Z(r1)`			&Z[0]			
t	4	`add r1,r1,#8`	r1					
	5	`ldf f0,X(r1)`						
	6	`mulf f4,f0,f2`						
	7	`stf f4,Z(r1)`						

Reg. Status	
reg	T+
f0	ROB#1+
f2	
f4	ROB#2
r1	ROB#4

CDB	
V	T
[f0]	ROB#1

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

RS								
#	FU	busy	op	T	V1	V2	T1	T2
1	ALU	Yes	add	ROB#4	REG[r1]			
2	load	No						
3	store	Yes	stf	ROB#3		REG[r1]	ROB#2	
4	FP1	Yes	mulf	ROB#2	CDB.V	REG[f2]	~~ROB#1~~	
5	FP2	No						

allocate

f0 ready
grab from CDB

P6 Example: Cycle 5

hd tl	ROB + MOB							
	#	instruction	R	V	addr	IS	EX	CM
	1	`ldf f0,X(r1)`	f0	[f0]	&X[0]	c2	c3	c4
h	2	`mulf f4,f0,f2`	f4			c4	c5	
	3	`stf f4,Z(r1)`			&Z[0]			
	4	`add r1,r1,#8`	r1			c5		
t	5	`ldf f0,X(r1)`	f0					
	6	`mulf f4,f0,f2`						
	7	`stf f4,Z(r1)`						

Reg. Status	
reg	T+
f0	ROB#5
f2	
f4	ROB#2
r1	ROB#4

CDB	
V	T

retire, write ROB result into regfile

RS								
#	FU	busy	op	T	V1	V2	T1	T2
1	ALU	Yes	add	ROB#4	REG[r1]			
2	load	Yes	ldf	ROB#5				ROB#4
3	store	Yes	stf	ROB#3		REG[r1]	ROB#2	
4	FP1	No						
5	FP2	No						

allocate

free

P6 Example: Cycle 6

hd tl	ROB + MOB							
	#	instruction	R	V	addr	IS	EX	CM
	1	`ldf f0,X(r1)`	f0	[f0]	&X[0]	c2	c3	c4
h	2	`mulf f4,f0,f2`	f4			c4	c5+	
	3	`stf f4,Z(r1)`			&Z[0]			
	4	`add r1,r1,#8`	r1			c5	**c6**	
	5	`ldf f0,X(r1)`	f0					
t	6	`mulf f4,f0,f2`	**f4**					
	7	`stf f4,Z(r1)`						

Reg. Status	
reg	T+
`f0`	ROB#5
`f2`	
`f4`	**ROB#6**
`r1`	ROB#4

CDB	
V	T

RS								
#	FU	busy	op	T	V1	V2	T1	T2
1	ALU	**No**						
2	load	Yes	ldf	ROB#5				ROB#4
3	store	Yes	stf	ROB#3		REG[r1]	ROB#2	
4	FP1	No						
5	FP2	**Yes**	**mulf**	**ROB#6**		**REG[f2]**	**ROB#5**	

← free

← allocate

P6 Example: Cycle 7

hd tl	ROB + MOB							
	#	instruction	R	V	addr	IS	EX	CM
	1	`ldf f0,X(r1)`	f0	[f0]	&X[0]	c2	c3	c4
h	2	`mulf f4,f0,f2`	f4			c4	c5+	
	3	`stf f4,Z(r1)`			&Z[0]			
	4	`add r1,r1,#8`	r1	**[r1]**		c5	c6	**c7**
	5	`ldf f0,X(r1)`	f0		&X[1]	**c7**		
t	6	`mulf f4,f0,f2`	f4					
	7	`stf f4,Z(r1)`						

Reg. Status	
reg	T+
`f0`	ROB#5
`f2`	
`f4`	ROB#6
`r1`	ROB#4**+**

CDB	
V	T
[r1]	**ROB#4**

add finished
write result into ROB, CDB

stall DS, no free store RS

RS								
#	FU	busy	op	T	V1	V2	T1	T2
1	ALU	No						
2	load	Yes	ldf	ROB#5	**CDB.V**		~~**ROB#4**~~	
3	store	Yes	stf	ROB#3		REG[r1]	ROB#2	
4	FP1	No						
5	FP2	Yes	mulf	ROB#6		REG[f2]	ROB#5	

r1 ready
grab from CDB

P6 Example: Cycle 8

hd tl	ROB + MOB							
	#	instruction	R	V	addr	IS	EX	CM
	1	`ldf f0,X(r1)`	f0	[f0]	&X[0]	c2	c3	c4
h	2	`mulf f4,f0,f2`	f4	**[f4]**		c4	c5+	**c8**
	3	`stf f4,Z(r1)`			&Z[0]	**c8**		
	4	`add r1,r1,#8`	r1	[r1]		c5	c6	c7
	5	`ldf f0,X(r1)`	f0		&X[1]	c7	**c8**	
t	6	`mulf f4,f0,f2`	f4					
	7	`stf f4,Z(r1)`						

Reg. Status	
reg	T+
`f0`	ROB#5
`f2`	
`f4`	ROB#6
`r1`	ROB#4+

CDB	
V	T
[f4]	**ROB#2**

stall RT

stall DS, no free store RS

RS								
#	FU	busy	op	T	V1	V2	T1	T2
1	ALU	No						
2	load	**No**						
3	store	Yes	stf	ROB#3	**CDB.V**	REG[r1]	~~**ROB#2**~~	
4	FP1	No						
5	FP2	Yes	mulf	ROB#6		REG[f2]	ROB#5	

free

f4 ready
grab from CDB

P6 Example: Cycle 9

hd tl	ROB + MOB							
	#	instruction	R	V	addr	IS	EX	CM
	1	`ldf f0,X(r1)`	f0	[f0]	&X[0]	c2	c3	c4
	2	`mulf f4,f0,f2`	f4	[f4]		c4	c5+	c8
h	3	`stf f4,Z(r1)`			&Z[0]	c8	**c9**	
	4	`add r1,r1,#8`	r1	[r1]		c5	c6	c7
	5	`ldf f0,X(r1)`	f0		&X[1]	c7	c8	**c9**
	6	`mulf f4,f0,f2`	f4			**c9**		
t	7	`stf f4,Z(r1)`			**&Z[1]**			

Reg. Status	
reg	T+
f0	ROB#5+
f2	
f4	ROB#6
r1	ROB#4+

CDB	
V	T
[f0]	ROB#5

stall RT

read from ROB not reg. file (+)

RS								
#	FU	busy	op	T	V1	V2	T1	T2
1	ALU	No						
2	load	No						
3	store	**Yes**	**stf**	**ROB#7**		**ROB#4.V**	**ROB#6**	
4	FP1	No						
5	FP2	Yes	mulf	ROB#6	**CDB.V**	REG[f2]	~~**ROB#5**~~	

free (ROB#3)
allocate (ROB#7)

f0 ready
grab from CDB

P6 Example: Cycle 10

hd tl	ROB + MOB							
	#	instruction	R	V	addr	IS	EX	CM
	1	`ldf f0,X(r1)`	f0	[f0]	&X[0]	c2	c3	c4
	2	`mulf f4,f0,f2`	f4	[f4]		c4	c5+	c8
h	3	`stf f4,Z(r1)`			&Z[0]	c8	c9	**c10**
	4	`add r1,r1,#8`	r1	[r1]		c5	c6	c7
	5	`ldf f0,X(r1)`	f0		&X[1]	c7	c8	c9
	6	`mulf f4,f0,f2`	f4			c9	**c10**	
t	7	`stf f4,Z(r1)`						

Reg. Status	
reg	T+
`f0`	ROB#5+
`f2`	
`f4`	ROB#6
`r1`	ROB#4+

stall RT

CDB	
V	T

RS								
#	FU	busy	op	T	V1	V2	T1	T2
1	ALU	No						
2	load	No						
3	store	Yes	stf	ROB#7		ROB#4.V	ROB#6	
4	FP1	No						
5	FP2	**No**						

free

P6 Example: Cycle 11

hd tl	ROB + MOB							
	#	instruction	R	V	addr	IS	EX	CM
	1	`ldf f0,X(r1)`	f0	[f0]	&X[0]	c2	c3	c4
	2	`mulf f4,f0,f2`	f4	[f4]		c4	c5+	c8
	3	`stf f4,Z(r1)`			&Z[0]	c8	c9	c10
h	4	`add r1,r1,#8`	r1	[r1]		c5	c6	c7
	5	`ldf f0,X(r1)`	f0		&X[1]	c7	c8	c9
	6	`mulf f4,f0,f2`	f4			c9	c10	
t	7	`stf f4,Z(r1)`						

Reg. Status	
reg	T+
`f0`	ROB#5+
`f2`	
`f4`	ROB#6
`r1`	ROB#4+

CDB	
V	T

retire stf

RS								
#	FU	busy	op	T	V1	V2	T1	T2
1	ALU	No						
2	load	No						
3	store	Yes	stf	ROB#7		ROB#4.V	ROB#6	
4	FP1	No						
5	FP2	No						