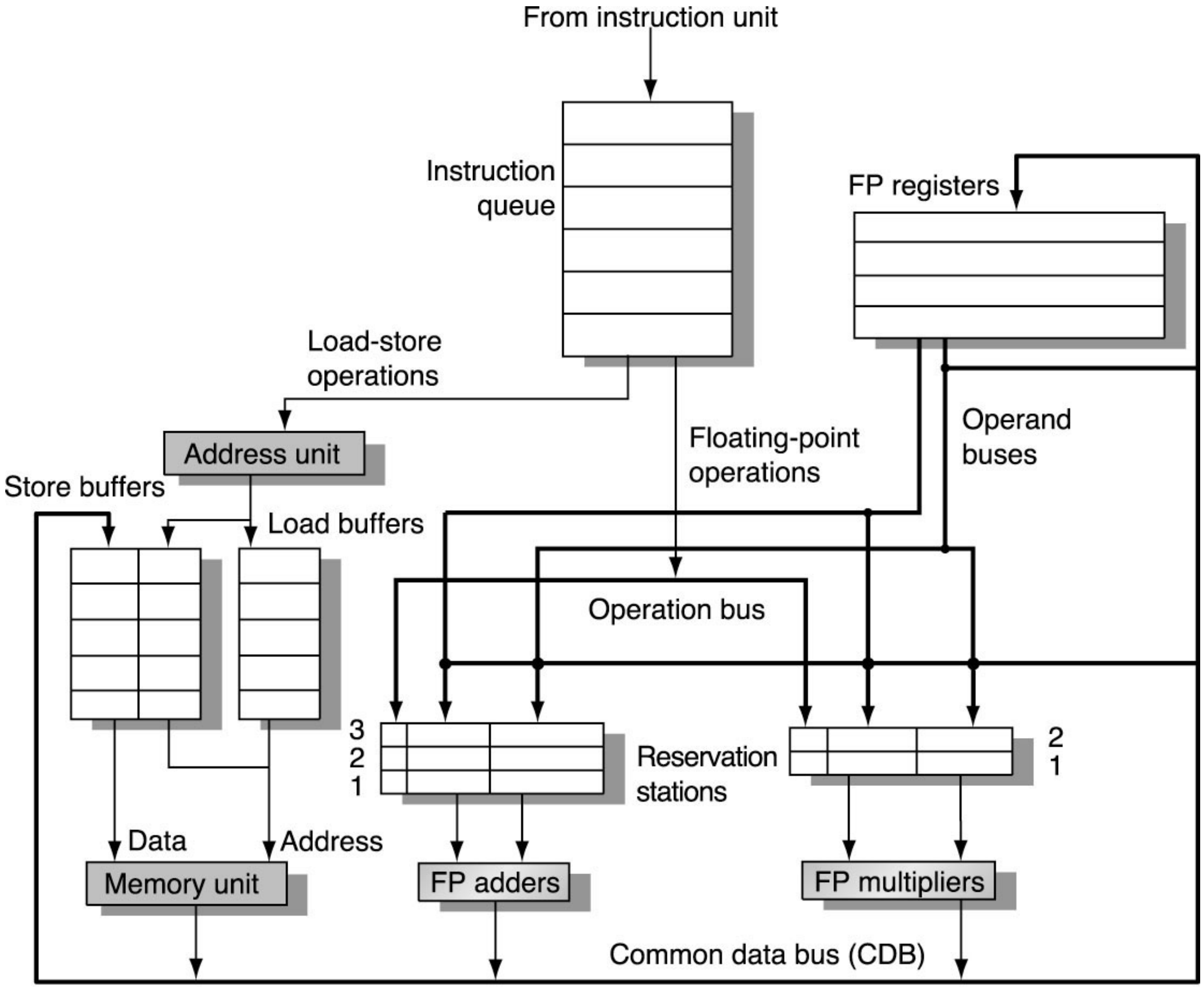
Tomasulo

Scoreboard Summary

- Drawbacks of scoreboard:
  - has to stall completely on WAW hazards
    - can’t overlap loop iterations!

Tomasulo Architecture



Tomasulo

- For IBM 360/91 about 3 years after CDC 6600
- Goal: High Performance without special compilers
- New features
  - HW renaming of registers to avoid WAR and WAW hazards
    - overlap loop iterations
  - registers in instructions are replaced by pointers to reservations station buffer
  - reservation stations
    - control is distributed to the functional units instead of the scoreboard
  - Common Data Bus broadcasts results to all reservation stations
  - Load and Stores are treated as reservation stations with memory disambiguation

~~Three~~ Four Phases of Tomasulo’s

(Text crunches several stages together)

1. **Dispatch** — get instruction from instr queue
   If reservation station X is free, the instruction is dispatched
   - reservation station becomes busy
   - read inputs from register file (RF)
   - read register status table (“RST”) for each input register
   - if (“RST”) has an entry “Y” for the input register, then the fetched RF value is not correct, and the input will come over the CDB from Y.
   - register status table (“RST”) updated to map output register to reservation station X.

2. **Issue** — wait for input operands
   “Snoop” CDB bus for inputs, if needed.
   If both operands are ready, arbitrate for functional unit versus other reservations stations assigned to same unit.

3. **Execute** — compute on the functional unit

4. **Write result** — finish execution
   Write result on Common Data Bus to all awaiting reservation stations;
   If RST still maps destination register to res-station X:
   - write result to RF.
   - clear RST entry.

Tomasulo Control Components

Reservation Station Related

Busy—Indicates whether the reservation station is busy or not

Op — Operation to perform in the unit (e.g., + or –)

Vj, Vk — Value of source operands

Qj, Qk — Reservation stations producing source operands Vj, Vk

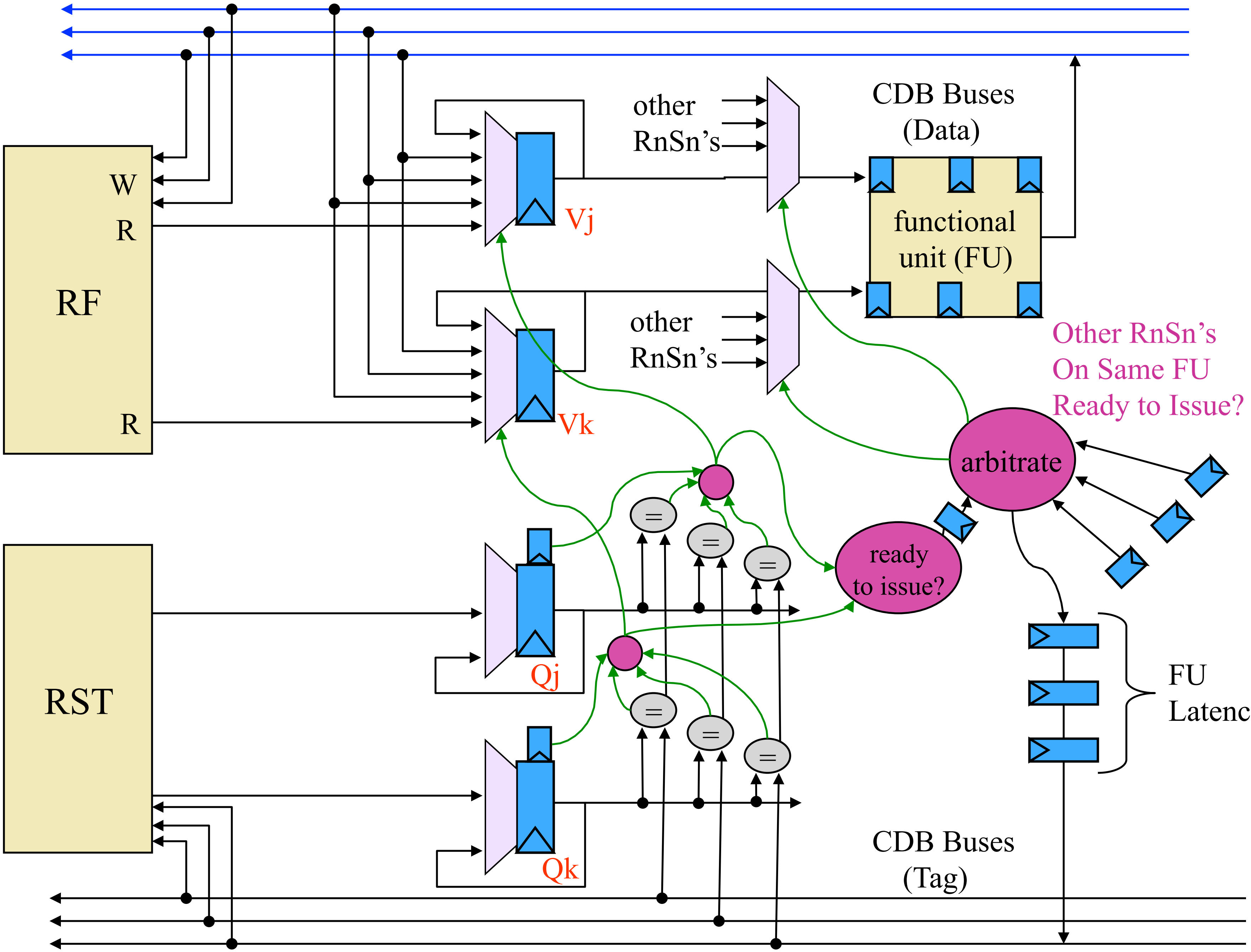
Rj, Rk — Flags indicating when Vj, Vk are ready

Register Related

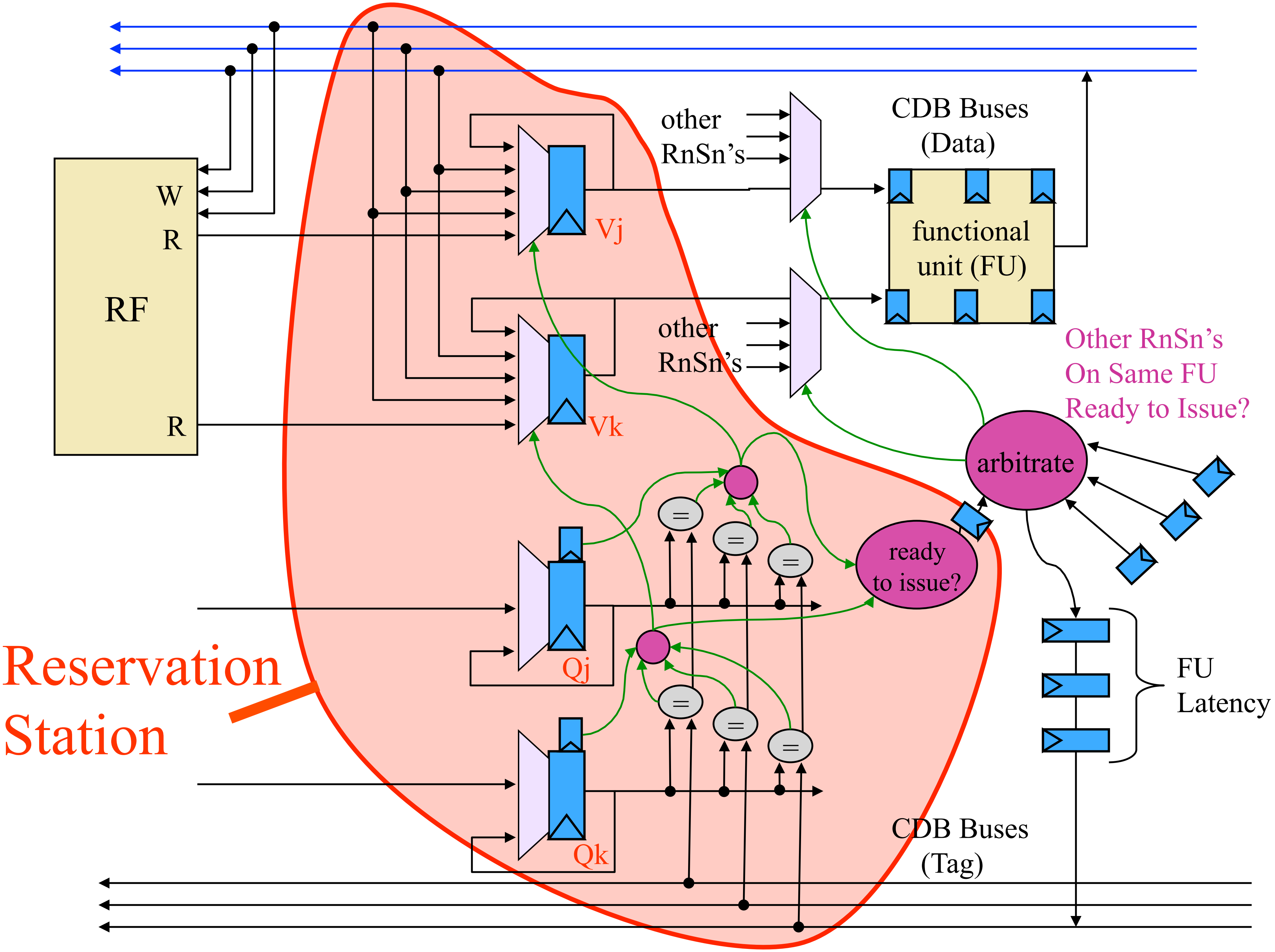
Register status table - Indicates which reservation station that contains the youngest instruction that will write to each register, if a write is outstanding. Blank when no pending instructions will write that register.

Assumptions for these slides

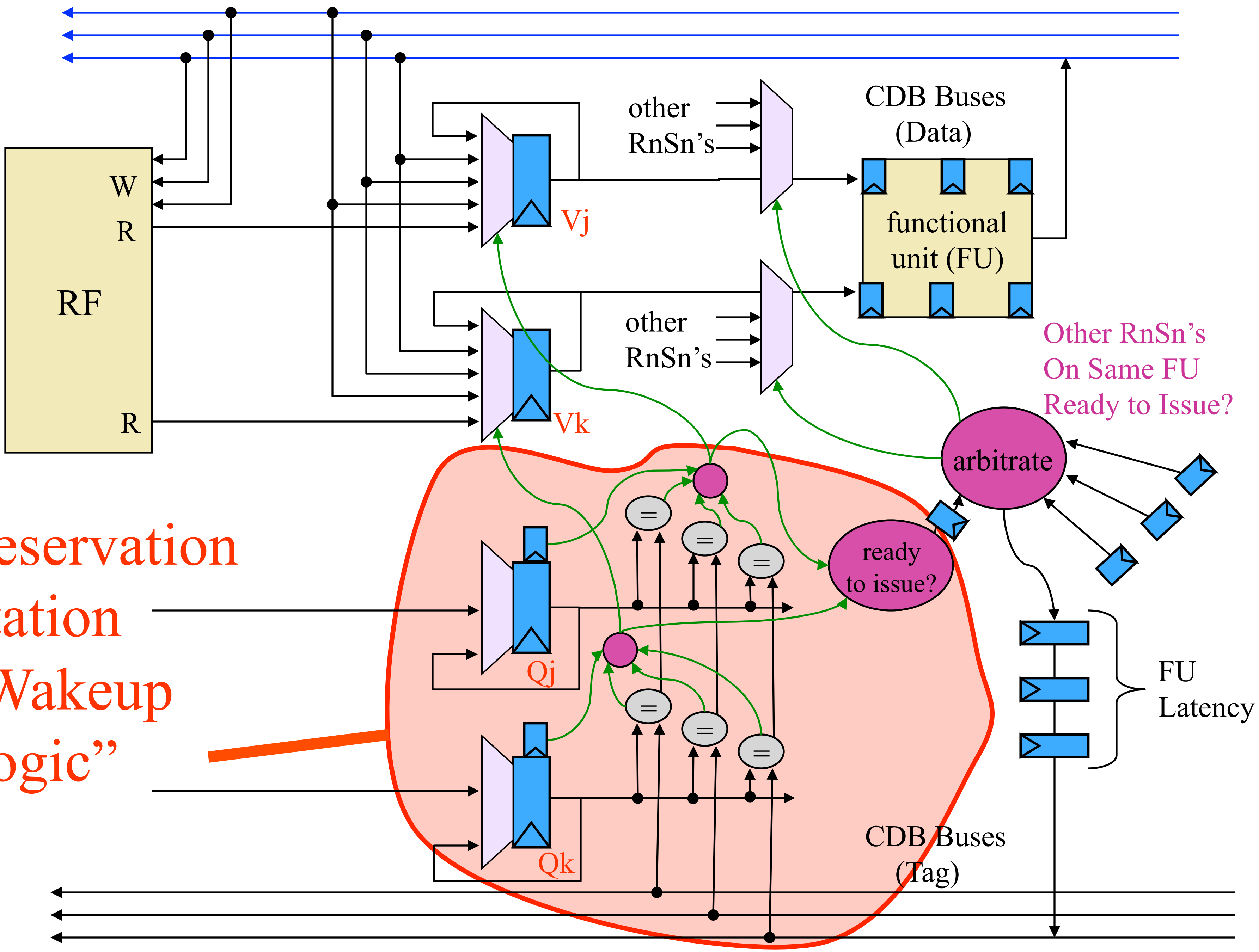
- LD = 1 cycle for EA calculation, 2 cycles for array access
- Add/Mul/Div = 2 cycles (in real life, Div >> Add/Mul)



W
R
RF
R
other
RnSn's
Vj
other
RnSn's
Vk
CDB Buses
(Data)
functional
unit (FU)
Other RnSn's
On Same FU
Ready to Issue?
arbitrate
ready
to issue?
=
RST
Qj
Qk
FU
Latency
CDB Buses
(Tag)

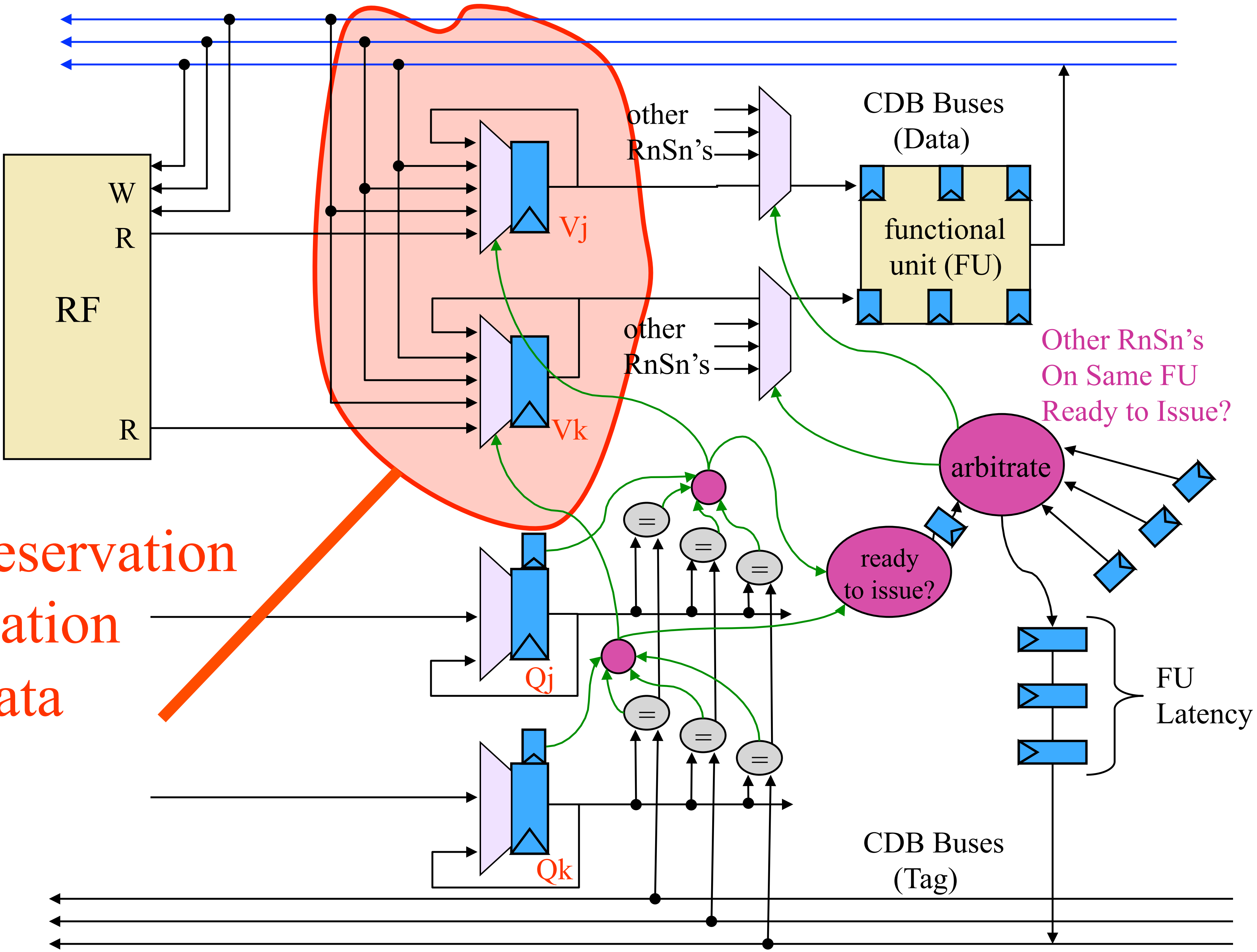


CDB Buses
(Data)
RF
W
R
R
other
RnSn's
other
RnSn's
Vj
Vk
Qj
Qk
functional
unit (FU)
Other RnSn's
On Same FU
Ready to Issue?
arbitrate
ready
to issue?
=
FU
Latency
Reservation
Station
CDB Buses
(Tag)

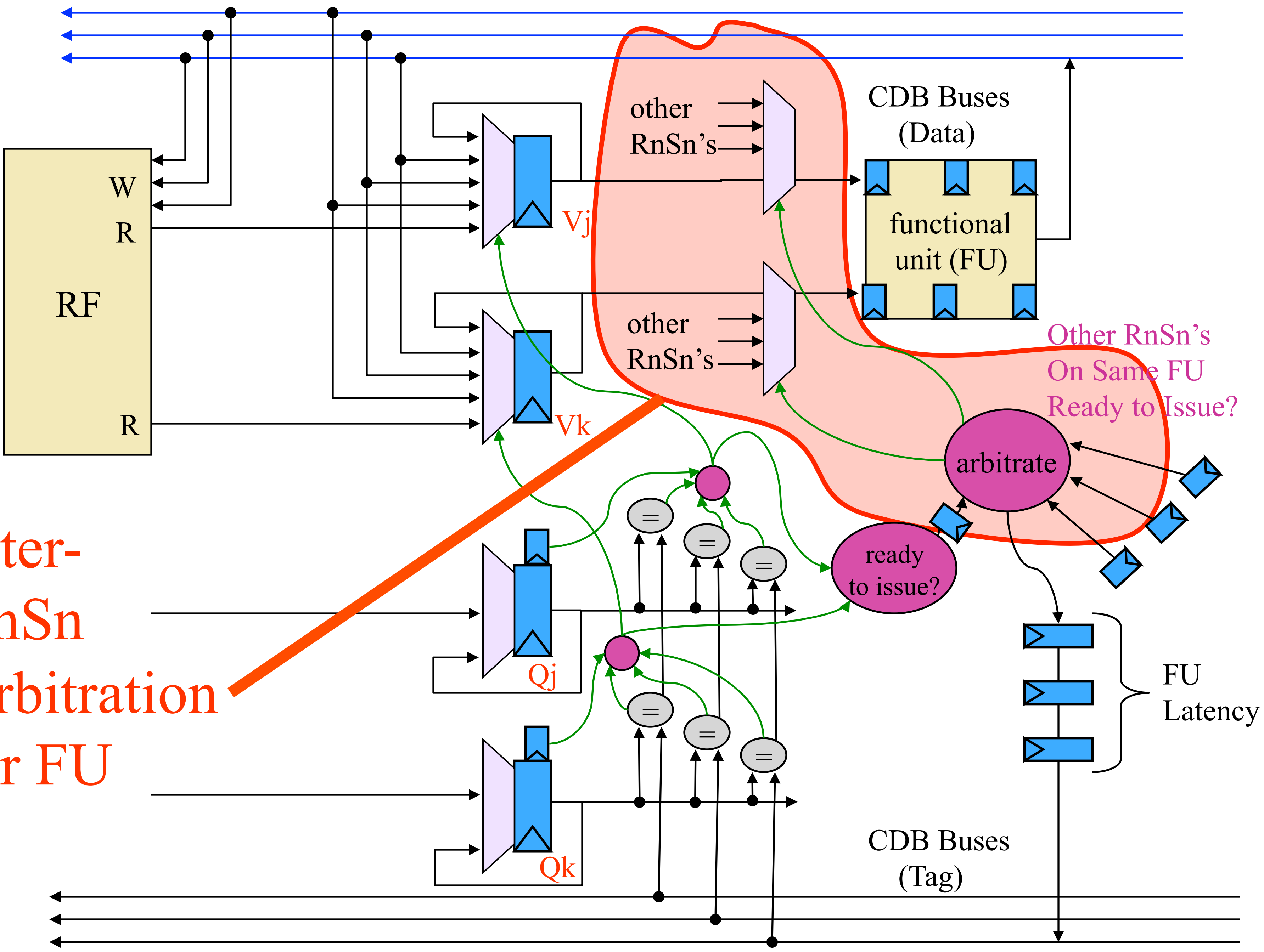


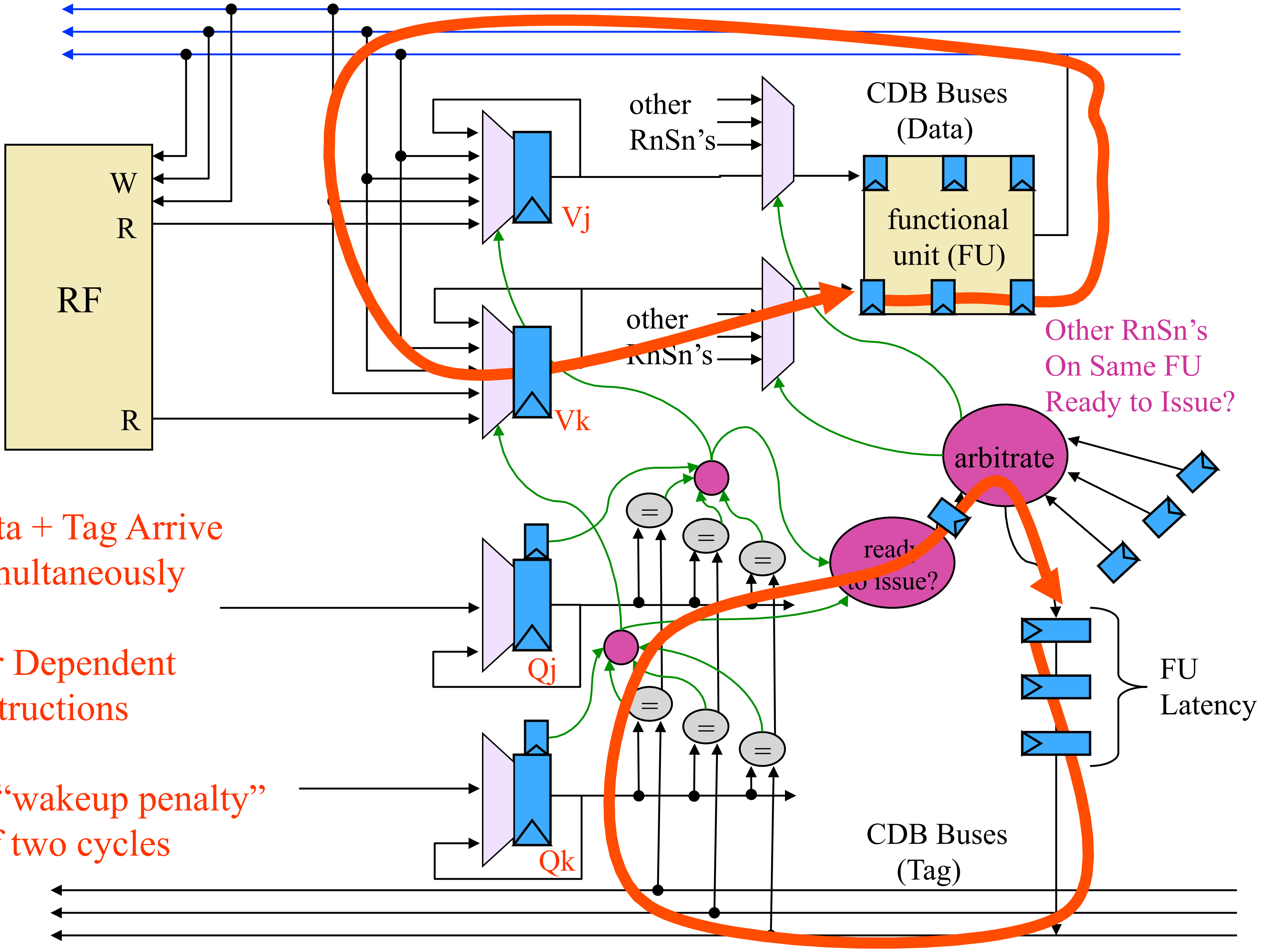
Reservation
Station
"Wakeup
Logic"

Reservation Station Data



Inter-RnSn Arbitration for FU



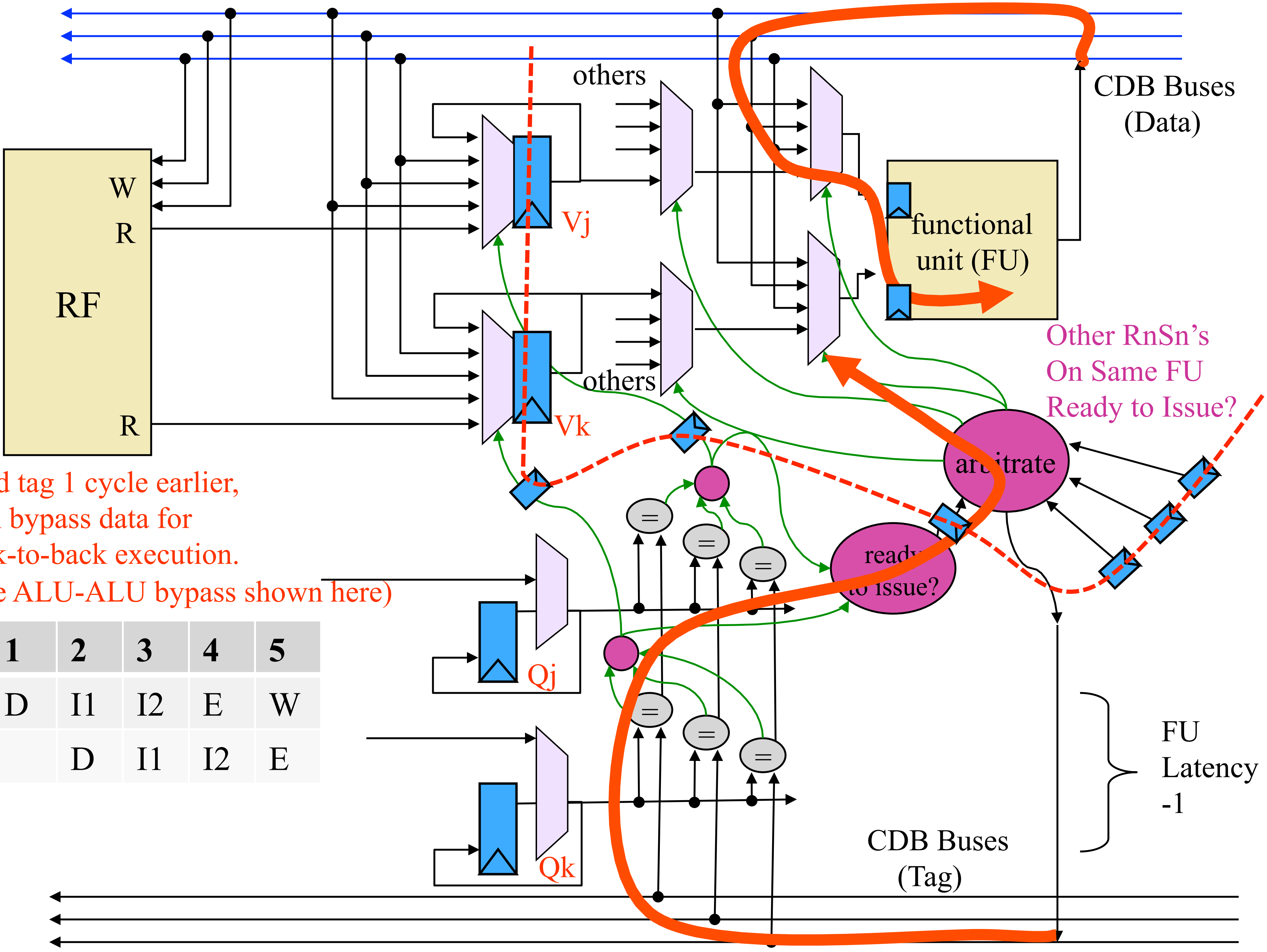


Data + Tag Arrive Simultaneously

For Dependent Instructions

-- "wakeup penalty" of two cycles

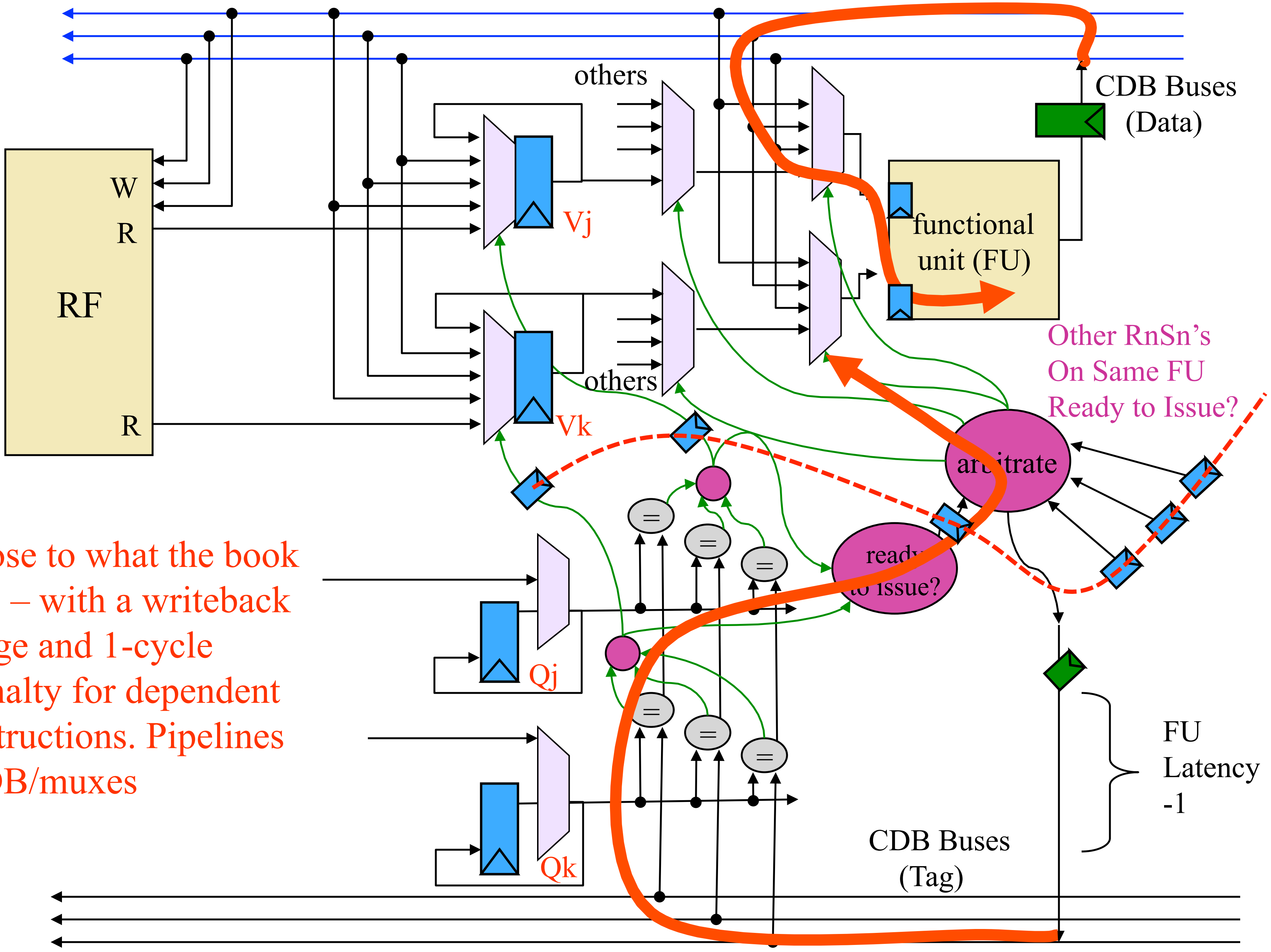
Pipelining the wakeup logic for zero-penalty back-to-back dependent operations: an example



Send tag 1 cycle earlier, then bypass data for back-to-back execution.
(free ALU-ALU bypass shown here)

	1	2	3	4	5
+	D	I1	I2	E	W
+		D	I1	I2	E

Pipelining the wakeup logic for single cycle-penalty back-to-back dependent operations: an example



Close to what the book has – with a writeback stage and 1-cycle penalty for dependent instructions. Pipelines CDB/muxes

```
1   L.D     F6,     32(R2)        D
2   L.D     F2,     44(R3)
3   MUL.D   F0,   F2,   F4
4   SUB.D   F6,   F2,   F6
5   DIV.D   F10,  F0,   F6
6   ADD.D   F6,   F8,   F2
```

Cycle 1

Reservation Stations

Name	Op	Vj	Vk	Qj	Qk	A	(Instr #)
Load1	L.D	32	[R2]				1
Load2							
Add1							
Add2							
Add3							
Mult1							
Mult2							

Register Status Table

F0	F2	F4	F6	F8	F10	..	F30
			Load1				

```
1   L.D     F6,     32(R2)       D    I
2   L.D     F2,     44(R3)            D
3   MUL.D   F0,   F2,   F4
4   SUB.D   F6,   F2,   F6
5   DIV.D   F10,  F0,   F6
6   ADD.D   F6,   F8,   F2
```

Cycle 2

Reservation Stations

Name	Op	Vj	Vk	Qj	Qk	A	(Instr #)
Load1	L.D	32	[R2]				1
Load2	L.D	44	[R3]				2
Add1							
Add2							
Add3							
Mult1							
Mult2							

Register Status

F0	F2	F4	F6	F8	F10	..	F30
	Load2		Load1				

Cycle 3

1	L.D	F6,	32(R2)		D	I	EA
2	L.D	F2,	44(R3)			D	I
3	MUL.D	F0,	F2,	F4			D
4	SUB.D	F6,	F2,	F6			
5	DIV.D	F10,	F0,	F6			
6	ADD.D	F6,	F8,	F2			

Reservation Stations

Name	Op	Vj	Vk	Qj	Qk	A	(Instr #)
Load1	L.D	32	[R2]			"[R2]+32"	1
Load2	L.D	44	[R3]				2
Add1							
Add2							
Add3							
Mult1	MUL.D		[F4]	Load2			3
Mult2							

Register Status

F0	F2	F4	F6	F8	F10	..	F30
Mult1	Load2		Load1				

Cycle 4

1	L.D	F6,	32(R2)		D	I	EA	L1
2	L.D	F2,	44(R3)			D	I	EA
3	MUL.D	F0,	F2,	F4			D	I
4	SUB.D	F6,	F2,	F6				D
5	DIV.D	F10,	F0,	F6				
6	ADD.D	F6,	F8,	F2				

Reservation Stations

Name	Op	Vj	Vk	Qj	Qk	A	(Instr #)
Load1	L.D	32	[R2]			“[R2]+32”	1
Load2	L.D	44	[R3]			“[R3]+44”	2
Add1	SUB.D			Load2	Load1		4
Add2							
Add3							
Mult1	MUL.D		[F4]	Load2			3
Mult2							

Register Status

F0	F2	F4	F6	F8	F10	..	F30
Mult1	Load2		~~Load1~~/Add1				

Cycle 5

1	L.D	F6,	32(R2)		D	I	EA	L1	L2
2	L.D	F2,	44(R3)			D	I	EA	L1
3	MUL.D	F0,	F2,	F4			D	I	I
4	SUB.D	F6,	F2,	F6				D	I
5	DIV.D	F10,	F0,	F6					D
6	ADD.D	F6,	F8,	F2					

Reservation Stations

Name	Op	Vj	Vk	Qj	Qk	A	(Instr #)
Load1	L.D	32	[R2]			“[R2]+32”	1
Load2	L.D	44	[R3]			“[R3]+44”	2
Add1	SUB.D			Load2	Load1		4
Add2							
Add3							
Mult1	MUL.D		[F4]	Load2			3
Mult2	DIV.D			Mult1	Add1		5

Register Status

F0	F2	F4	F6	F8	F10	..	F30
Mult1	Load2		Add1		Mult2		

Cycle 6

1	L.D	F6,	32(R2)		D	I	EA	L1	L2	W
2	L.D	F2,	44(R3)			D	I	EA	L1	L2
3	MUL.D	F0,	F2,	F4			D	I	I	I
4	SUB.D	F6,	F2,	F6				D	I	I
5	DIV.D	F10,	F0,	F6					D	I
6	ADD.D	F6,	F8,	F2						D

Reservation Stations

Name	Op	Vj	Vk	Qj	Qk	A	(Instr #)
Load1	~~L.D~~	~~32~~	~~[R2]~~			~~“[R2]+32”~~	~~1~~
Load2	L.D	44	[R3]			“[R3]+44”	2
Add1	SUB.D		[Load1]	Load2	~~Load1~~		4
Add2	ADD.D	[F8]			Load2		6
Add3							
Mult1	MUL.D		[F4]	Load2			3
Mult2	DIV.D			Mult1	Add1		5

Register Status

F0	F2	F4	F6	F8	F10	..	F30
Mult1	Load2		~~Add1~~/Add2		Mult2		

Cycle 7

1	L.D	F6,	32(R2)		D	I	EA	L1	L2	W	
2	L.D	F2,	44(R3)			D	I	EA	L1	L2	W
3	MUL.D	F0,	F2,	F4			D	I	I	I	I
4	SUB.D	F6,	F2,	F6				D	I	I	I
5	DIV.D	F10,	F0,	F6					D	I	I
6	ADD.D	F6,	F8,	F2						D	I

Reservation Stations

Name	Op	Vj	Vk	Qj	Qk	A	(Instr #)
Load1							
Load2	~~L.D~~	~~44~~	~~[R3]~~			~~"[R3]+44"~~	~~2~~
Add1	SUB.D	[Load2]	[Load1]	~~Load2~~			4
Add2	ADD.D	[F8]	[Load2]		~~Load2~~		6
Add3							
Mult1	MUL.D	[Load2]	[F4]	~~Load2~~			3
Mult2	DIV.D			Mult1	Add1		5

Register Status

F0	F2	F4	F6	F8	F10	..	F30
Mult1	~~Load2~~		Add2		Mult2		

Cycle 8

1	L.D	F6,	32(R2)		D	I	EA	L1	L2	W		
2	L.D	F2,	44(R3)			D	I	EA	L1	L2	W	
3	MUL.D	F0,	F2,	F4			D	I	I	I	I	M1
4	SUB.D	F6,	F2,	F6				D	I	I	I	A1
5	DIV.D	F10,	F0,	F6					D	I	I	I
6	ADD.D	F6,	F8,	F2						D	I	I

lost
on
arbitration

Reservation Stations

Name	Op	Vj	Vk	Qj	Qk	A	(Instr #)
Load1							
Load2							
Add1	SUB.D	[Load2]	[Load1]				4
Add2	ADD.D	[F8]	[Load2]				6
Add3							
Mult1	MUL.D	[Load2]	[F4]				3
Mult2	DIV.D			Mult1	Add1		5

Register Status

F0	F2	F4	F6	F8	F10	..	F30
Mult1			Add2		Mult2		

Cycle 9

1	L.D	F6,	32(R2)		D	I	EA	L1	L2	W			
2	L.D	F2,	44(R3)			D	I	EA	L1	L2	W		
3	MUL.D	F0,	F2,	F4			D	I	I	I	I	M1	M2
4	SUB.D	F6,	F2,	F6				D	I	I	I	A1	A2
5	DIV.D	F10,	F0,	F6					D	I	I	I	I
6	ADD.D	F6,	F8,	F2						D	I	I	A1

Reservation Stations

Name	Op	Vj	Vk	Qj	Qk	A	(Instr #)
Load1							
Load2							
Add1	SUB.D	[Load2]	[Load1]				4
Add2	ADD.D	[F8]	[Load2]				6
Add3							
Mult1	MUL.D	[Load2]	[F4]				3
Mult2	DIV.D			Mult1	Add1		5

Register Status

F0	F2	F4	F6	F8	F10	..	F30
Mult1			Add2		Mult2		

Cycle 10

1	L.D	F6,	32(R2)		D	I	EA	L1	L2	W				
2	L.D	F2,	44(R3)			D	I	EA	L1	L2	W			
3	MUL.D	F0,	F2,	F4			D	I	I	I	I	M1	M2	W
4	SUB.D	F6,	F2,	F6				D	I	I	I	A1	A2	W
5	DIV.D	F10,	F0,	F6					D	I	I	I	I	I
6	ADD.D	F6,	F8,	F2						D	I	I	A1	A2

not cleared!

Reservation Stations

Name	Op	Vj	Vk	Qj	Qk	A	(Instr #)
Load1							
Load2							
Add1	~~SUB.D~~	~~[Load2]~~	~~[Load1]~~				~~4~~
Add2	ADD.D	[F8]	[Load2]				6
Add3							
Mult1	~~MUL.D~~	~~[Load2]~~	~~[F4]~~				~~3~~
Mult2	DIV.D	[Mult1]	[Add1]	~~Mult1~~	~~Add1~~		5

Register Status

F0	F2	F4	F6	F8	F10	..	F30
~~Mult1~~			Add2		Mult2		

Cycle 11

1	L.D	F6,	32(R2)		D	I	EA	L1	L2	W					
2	L.D	F2,	44(R3)			D	I	EA	L1	L2	W				
3	MUL.D	F0,	F2,	F4			D	I	I	I	I	M1	M2	W	
4	SUB.D	F6,	F2,	F6				D	I	I	I	A1	A2	W	
5	DIV.D	F10,	F0,	F6					D	I	I	I	I	I	M1
6	ADD.D	F6,	F8,	F2						D	I	I	A1	A2	W

Reservation Stations

Name	Op	Vj	Vk	Qj	Qk	A	(Instr #)
Load1							
Load2							
Add1							
Add2	~~ADD.D~~	~~[F8]~~	~~[Load2]~~				~~6~~
Add3							
Mult1							
Mult2	DIV.D	[Mult1]	[Add1]				5

Register Status

F0	F2	F4	F6	F8	F10	..	F30
			~~Add2~~		Mult2		

Cycle 12

1	L.D	F6,	32(R2)		D	I	EA	L1	L2	W						
2	L.D	F2,	44(R3)			D	I	EA	L1	L2	W					
3	MUL.D	F0,	F2,	F4			D	I	I	I	I	M1	M2	W		
4	SUB.D	F6,	F2,	F6				D	I	I	I	A1	A2	W		
5	DIV.D	F10,	F0,	F6					D	I	I	I	I	I	M1	M2
6	ADD.D	F6,	F8,	F2						D	I	I	A1	A2	W	

Reservation Stations

Name	Op	Vj	Vk	Qj	Qk	A	(Instr #)
Load1							
Load2							
Add1							
Add2							
Add3							
Mult1							
Mult2	DIV.D	[Mult1]	[Add1]				5

Register Status

F0	F2	F4	F6	F8	F10	..	F30
					Mult2		

Cycle 13

1	L.D	F6,	32(R2)		D	I	EA	L1	L2	W							
2	L.D	F2,	44(R3)			D	I	EA	L1	L2	W						
3	MUL.D	F0,	F2,	F4			D	I	I	I	I	M1	M2	W			
4	SUB.D	F6,	F2,	F6				D	I	I	I	A1	A2	W			
5	DIV.D	F10,	F0,	F6					D	I	I	I	I	I	M1	M2	W
6	ADD.D	F6,	F8,	F2						D	I	I	A1	A2	W		

Reservation Stations

Name	Op	Vj	Vk	Qj	Qk	A	(Instr #)
Load1							
Load2							
Add1							
Add2							
Add3							
Mult1							
Mult2	~~DIV.D~~	~~[Mult1]~~	~~[Add1]~~				~~5~~

Register Status

F0	F2	F4	F6	F8	F10	..	F30
					~~Mult2~~		

Cycle 14

1	L.D	F6,	32(R2)		D	I	EA	L1	L2	W							
2	L.D	F2,	44(R3)			D	I	EA	L1	L2	W						
3	MUL.D	F0,	F2,	F4			D	I	I	I	I	M1	M2	W			
4	SUB.D	F6,	F2,	F6				D	I	I	I	A1	A2	W			
5	DIV.D	F10,	F0,	F6					D	I	I	I	I	I	M1	M2	W
6	ADD.D	F6,	F8,	F2						D	I	I	A1	A2	W		

Reservation Stations

Name	Op	Vj	Vk	Qj	Qk	A	(Instr #)
Load1							
Load2							
Add1							
Add2							
Add3							
Mult1							
Mult2							

Register Status

F0	F2	F4	F6	F8	F10	..	F30

Benefits of Tomasulo Dynamic Scheduling

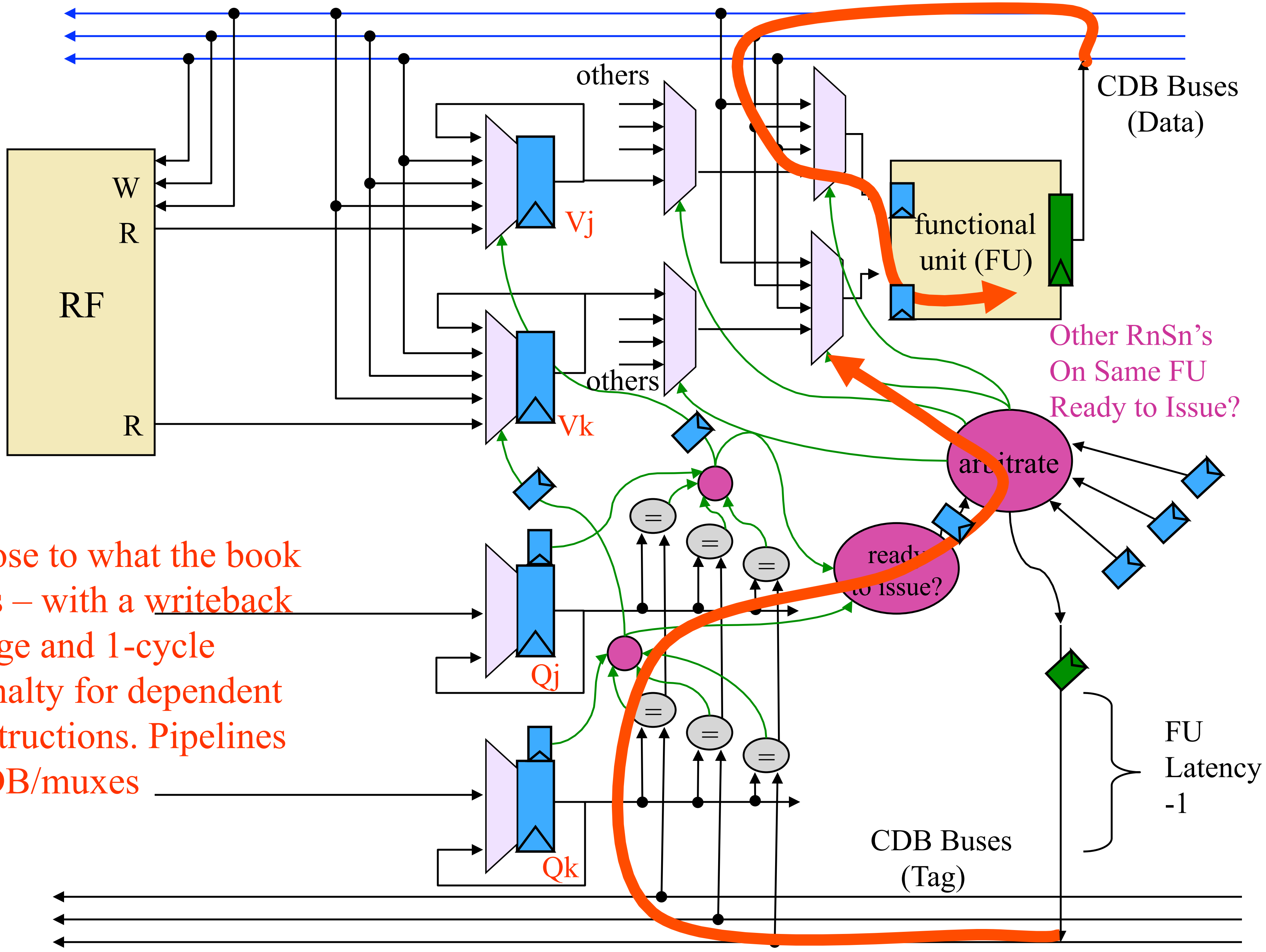
- Remember our “3-day waiting period” for hardware
- Complex hardware structures have high costs:
  - design time
  - verification time
  - silicon area
  - possibly lower frequency
- What does Tomasulo dynamic scheduling buy us versus smart compilation?
  - compilers can easily schedule instructions within a basic block
  - benefits of dynamic scheduling come from:
    - ☺ rescheduling instructions on the fly to respond to cache misses and any other variable-latency instructions
    - limited - scheduling across branches (and loop edges)
    - ☺ rescheduling instructions on the fly to respond to memory dependencies (or the lack there-of)

Allowing Out-of-Order Memory Accesses

- Phase 1: Resolve memory addresses *in order* (thus the semantics are slightly different than a reservation station).

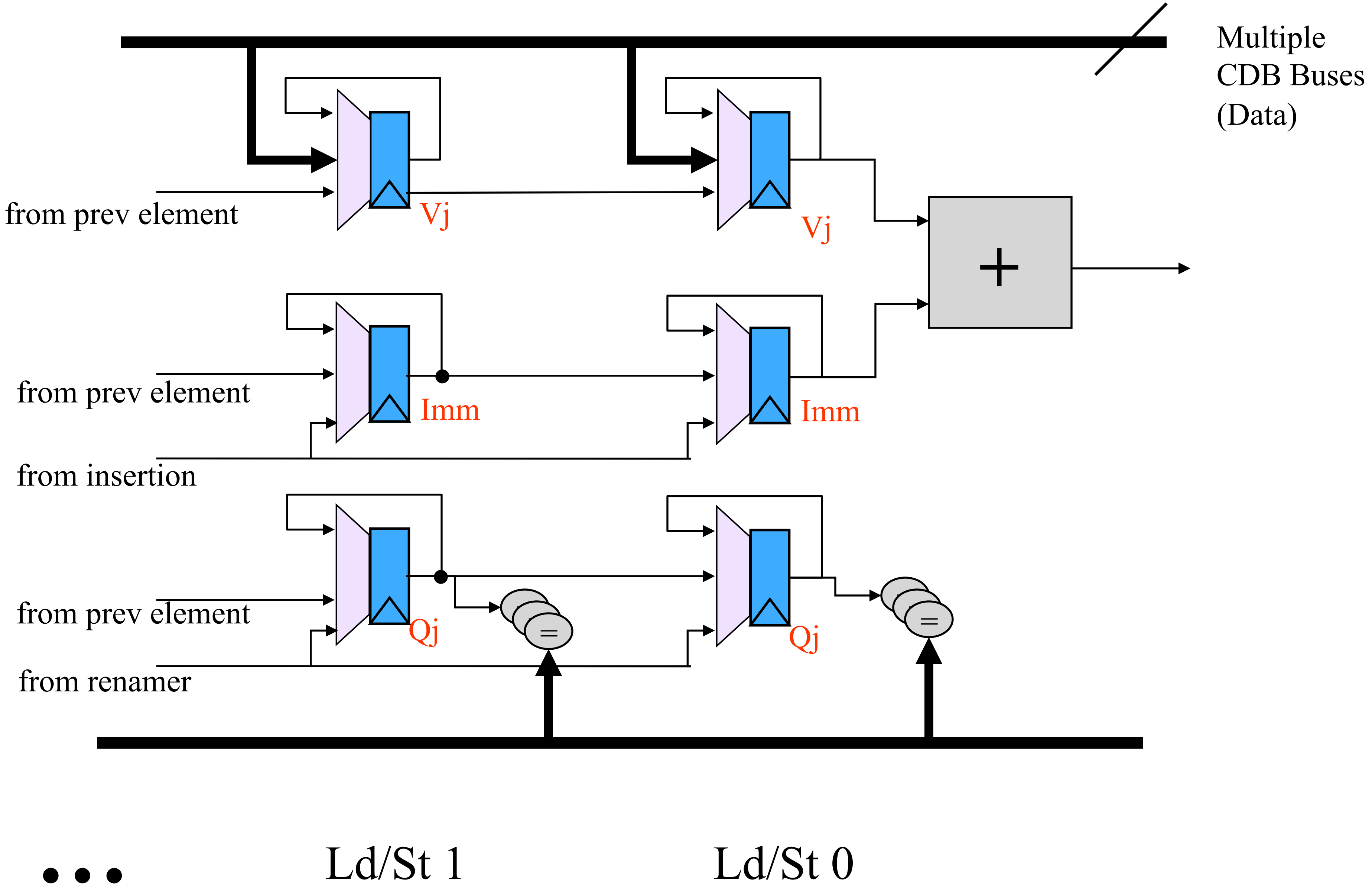
- Phase 2a: A load operation may proceed if no older stores to the same address are in Phase 2. (In-order processing of Phase 1 guarantees that all older addresses have passed Phase 2.)

- Phase 2b: A store operation may proceed if no older loads or stores to the same address are in flight.



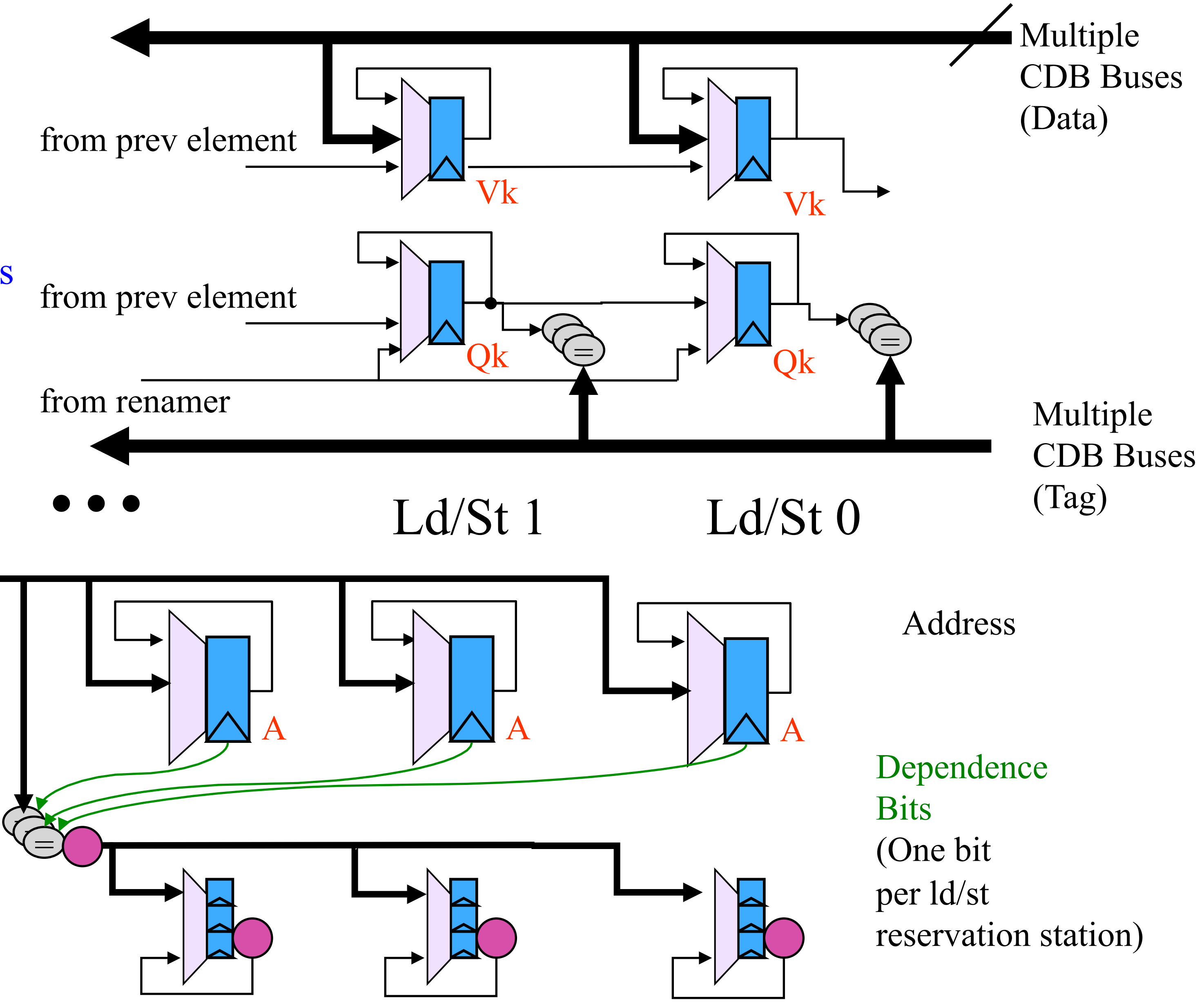
Close to what the book has – with a writeback stage and 1-cycle penalty for dependent instructions. Pipelines CDB/muxes

Hardware Sketch for Tomasulo Load-Store Queue
Phase 1: Resolve addresses in-order



Hardware Sketch for Tomasulo Load-Store Queue
Phase 2: Wait for inputs/deps

For stores, the Qk’s for data have been sitting here snooping the CDB while the address was being processed in-order with other LD/STs.



Each “A” register matches a corresponding reservation station.

Dependence Bits
(One bit per ld/st reservation station)

Rules for dependence bits:

Load-is-dependent(A, Other.A, Other.IsStore) →

(A == Other.A) && Other.IsStore

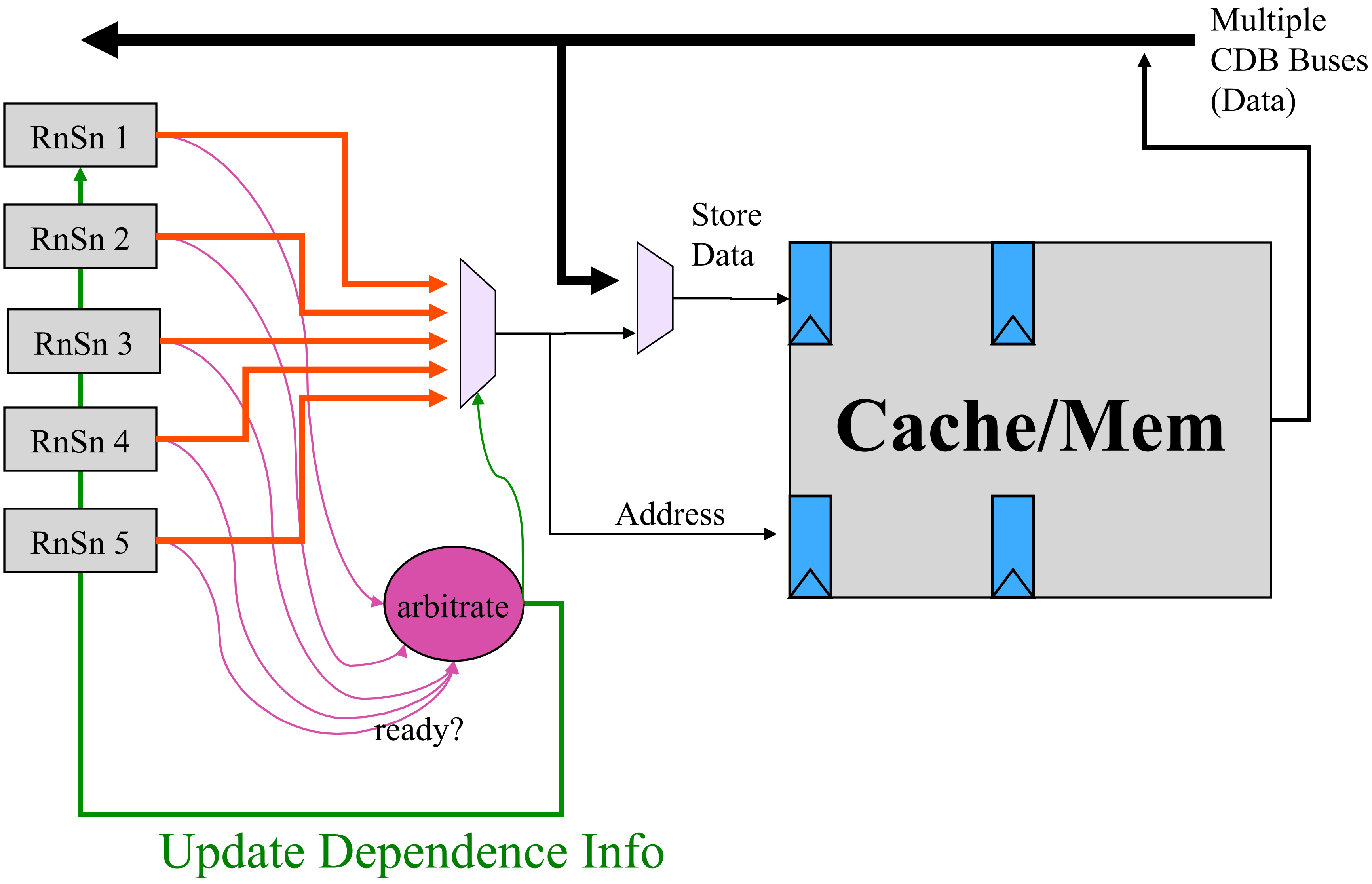
Stall on RAW, Ignores RAR

Store-is-dependent(A, Other.A, Other.IsStore) →

(A == Other.A)

Stall on WAR, WAW

Hardware Sketch for Tomasulo Load-Store Queue
Phase 3: Execute out-of-order, update deps.



Allowing Scheduling Across Branches

- Limited ability to schedule instructions across branches – only allowed if the branch is known to be taken.

- In 360/91, FP instructions are queued up separately from integer instructions. Presumably, FP instructions are only queued up after branches have been resolved in integer part of the machine. In other words, instructions are already passed the point of no-return once they reach the instruction queue.

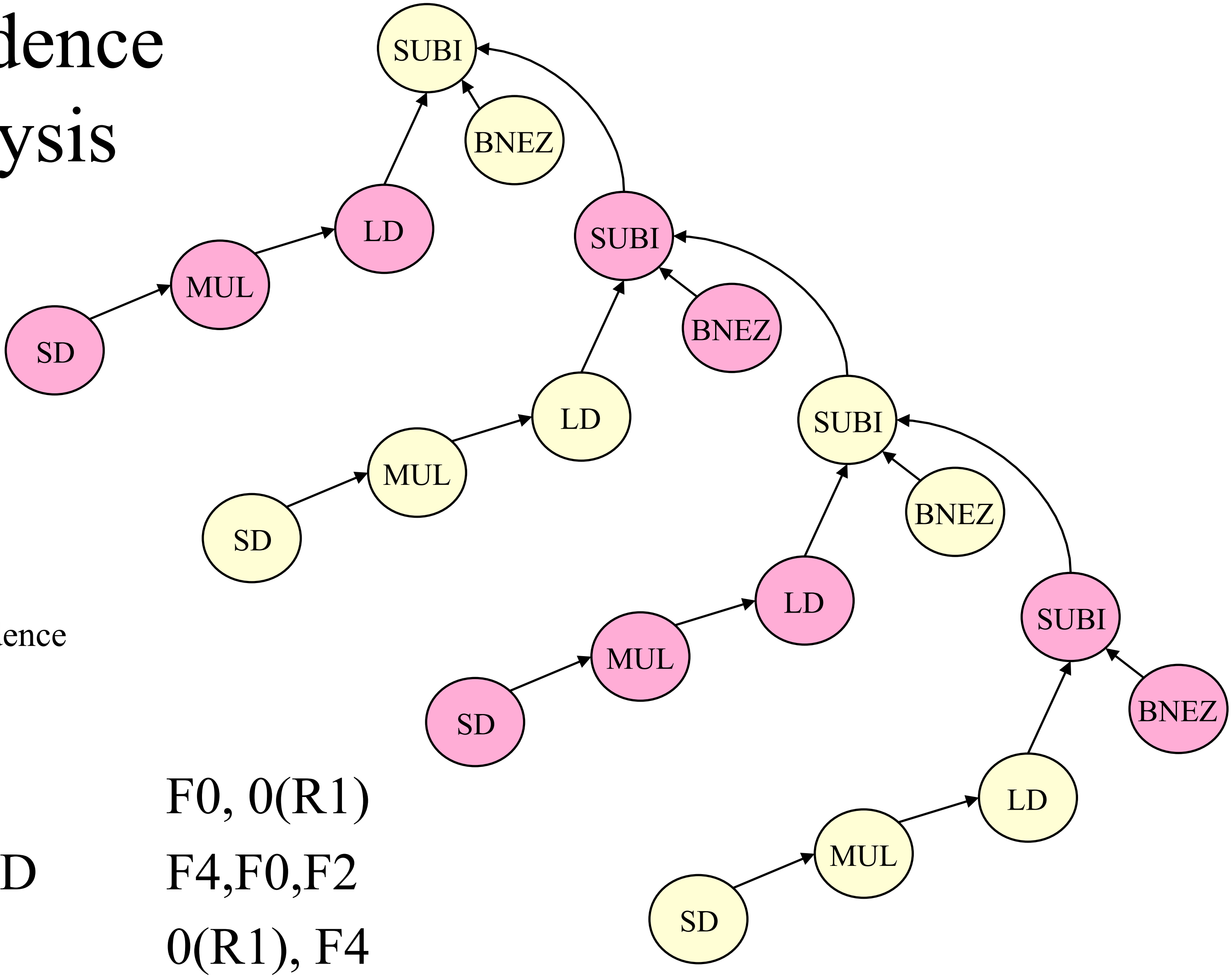
- Reasonable, but there is a benefit only if the integer part runs ahead of the floating point part.

  → Could the compiler handle this case?

Tomasulo Loop Example

```
Loop:  LD        F0, 0(R1)
       MULTD     F4,F0,F2
       SD        0(R1), F4
       SUBI      R1,R1,#8
       BNEZ      R1, Loop
```

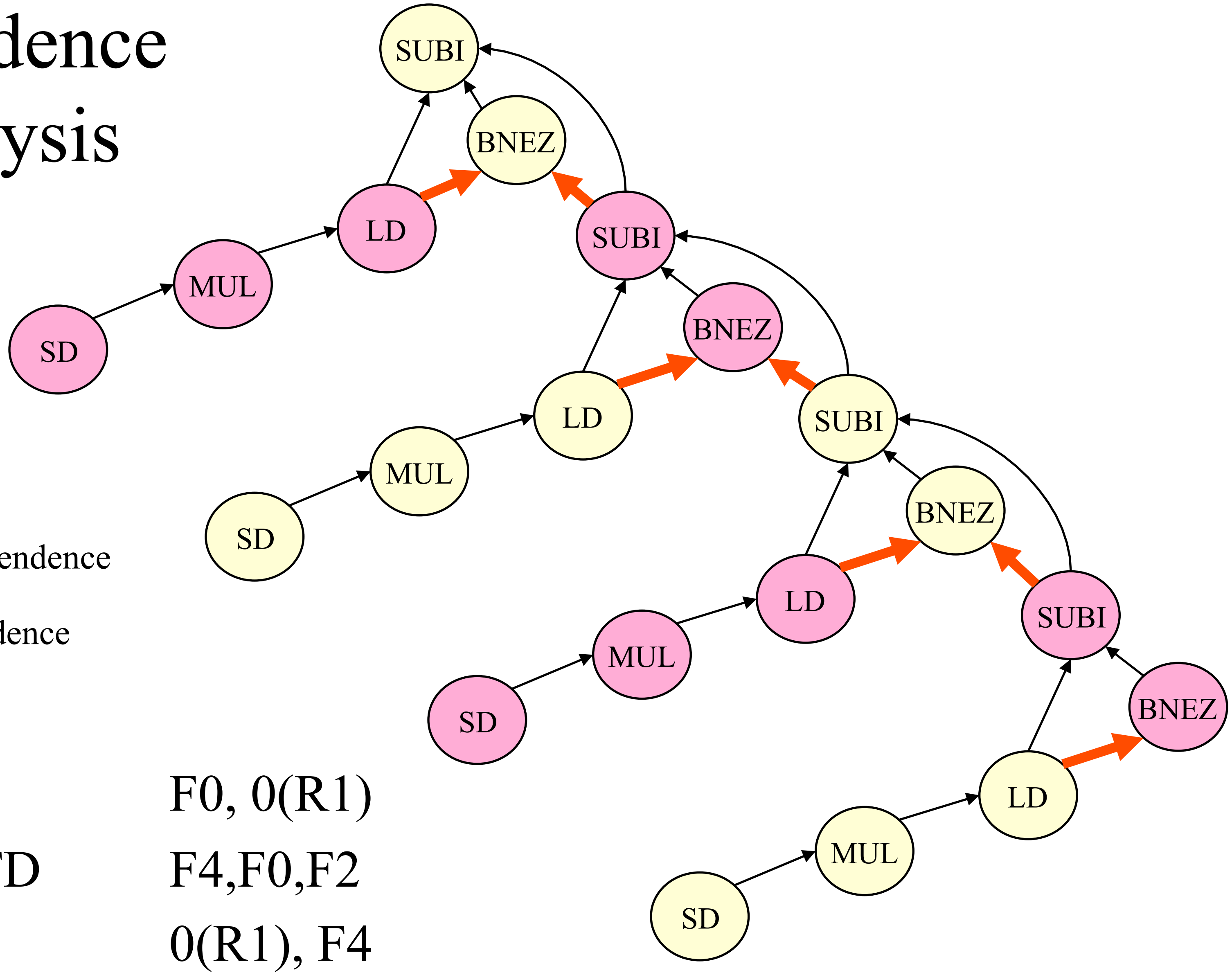
Dependence Analysis



→ Data Dependence

```
Loop:   LD          F0, 0(R1)
        MULTD       F4,F0,F2
        SD          0(R1), F4
        SUBI        R1,R1,#8
        BNEZ        R1, Loop
```

Dependence Analysis

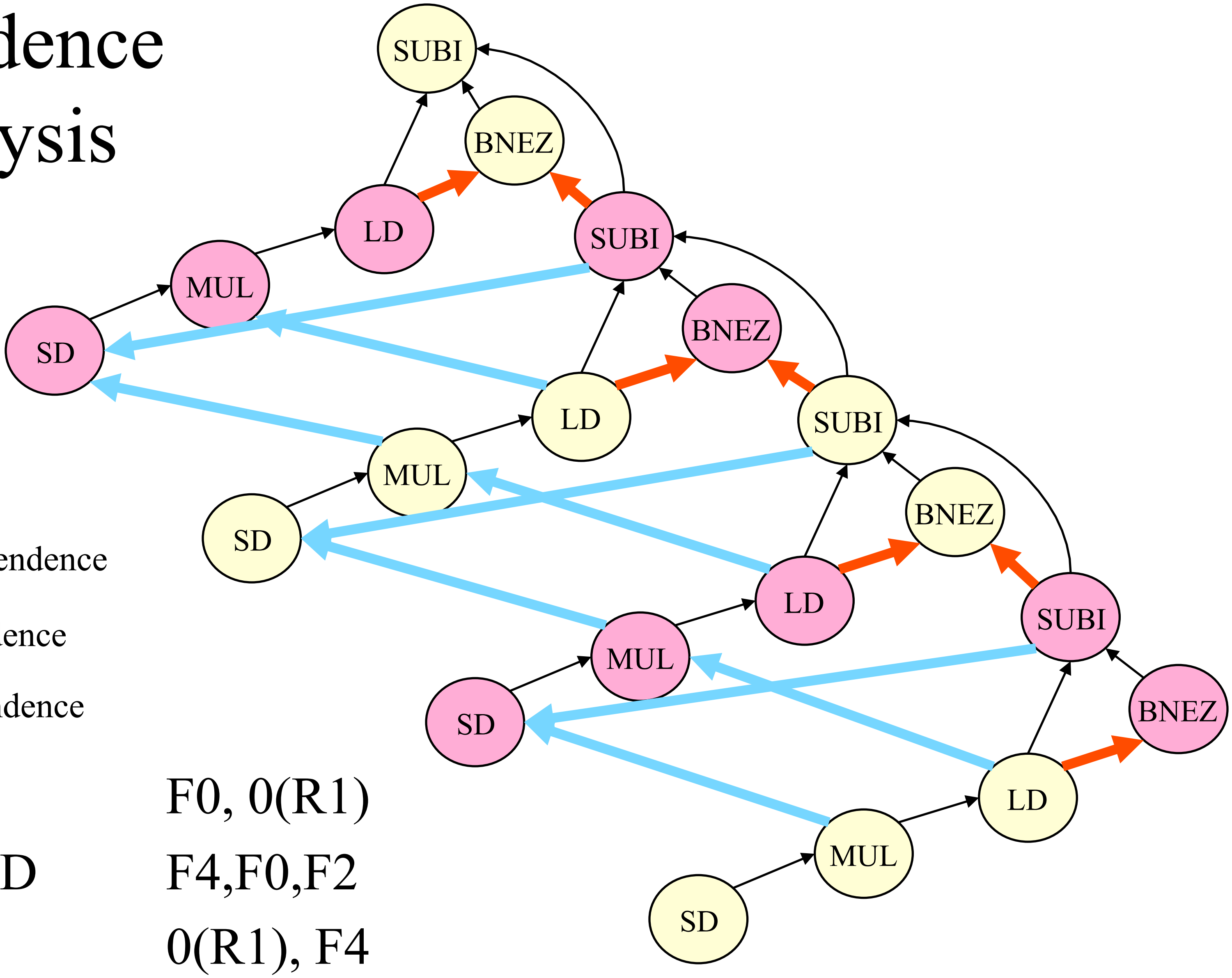


Control Dependence

Data Dependence

```
Loop:   LD          F0, 0(R1)
        MULTD       F4,F0,F2
        SD          0(R1), F4
        SUBI        R1,R1,#8
        BNEZ        R1, Loop
```

Dependence Analysis



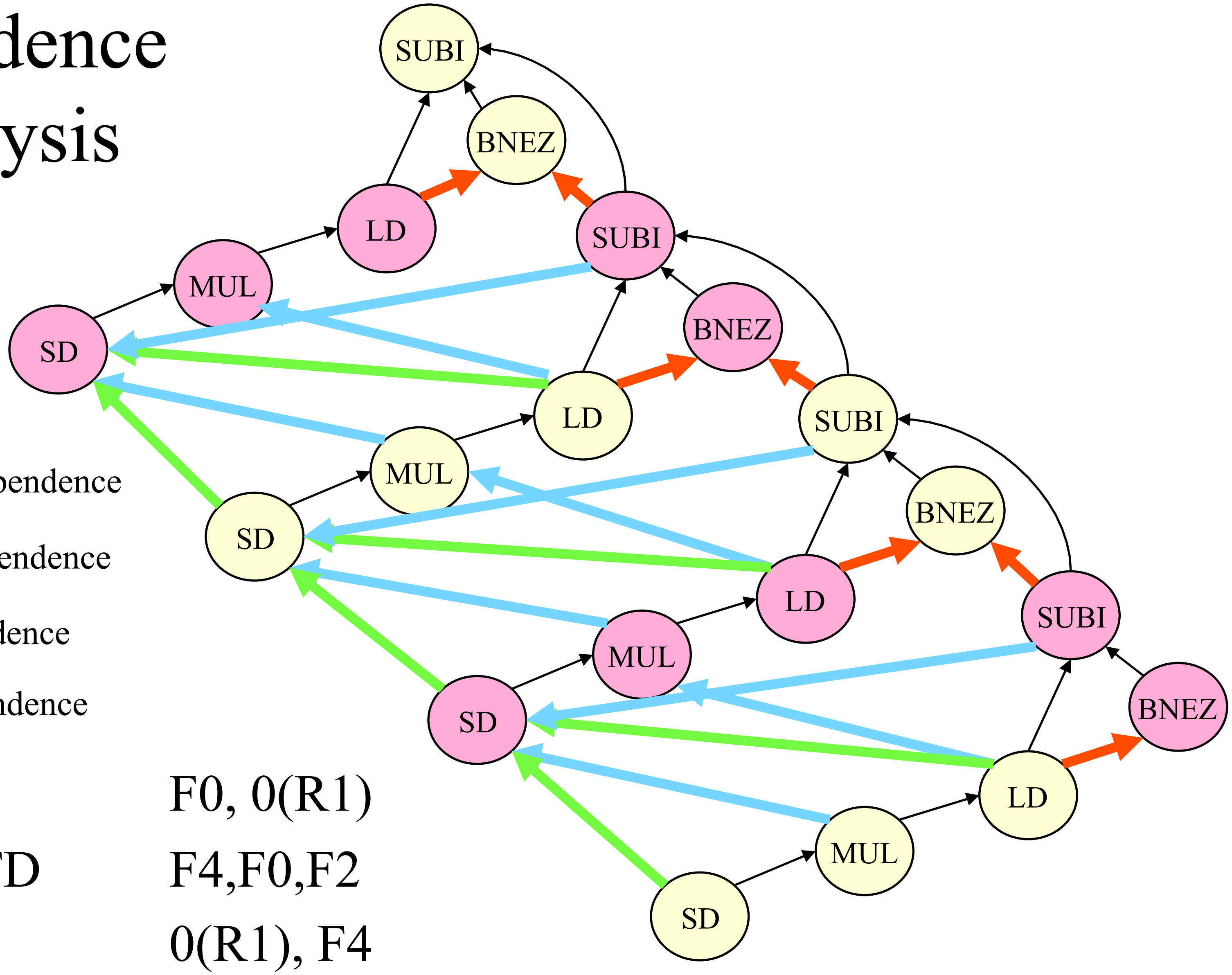
🟥 Control Dependence

⬛ Data Dependence

🟦 Name Dependence

```
Loop:  LD       F0, 0(R1)
       MULTD    F4,F0,F2
       SD       0(R1), F4
       SUBI     R1,R1,#8
       BNEZ     R1, Loop
```

Dependence Analysis



🟩 Memory Dependence

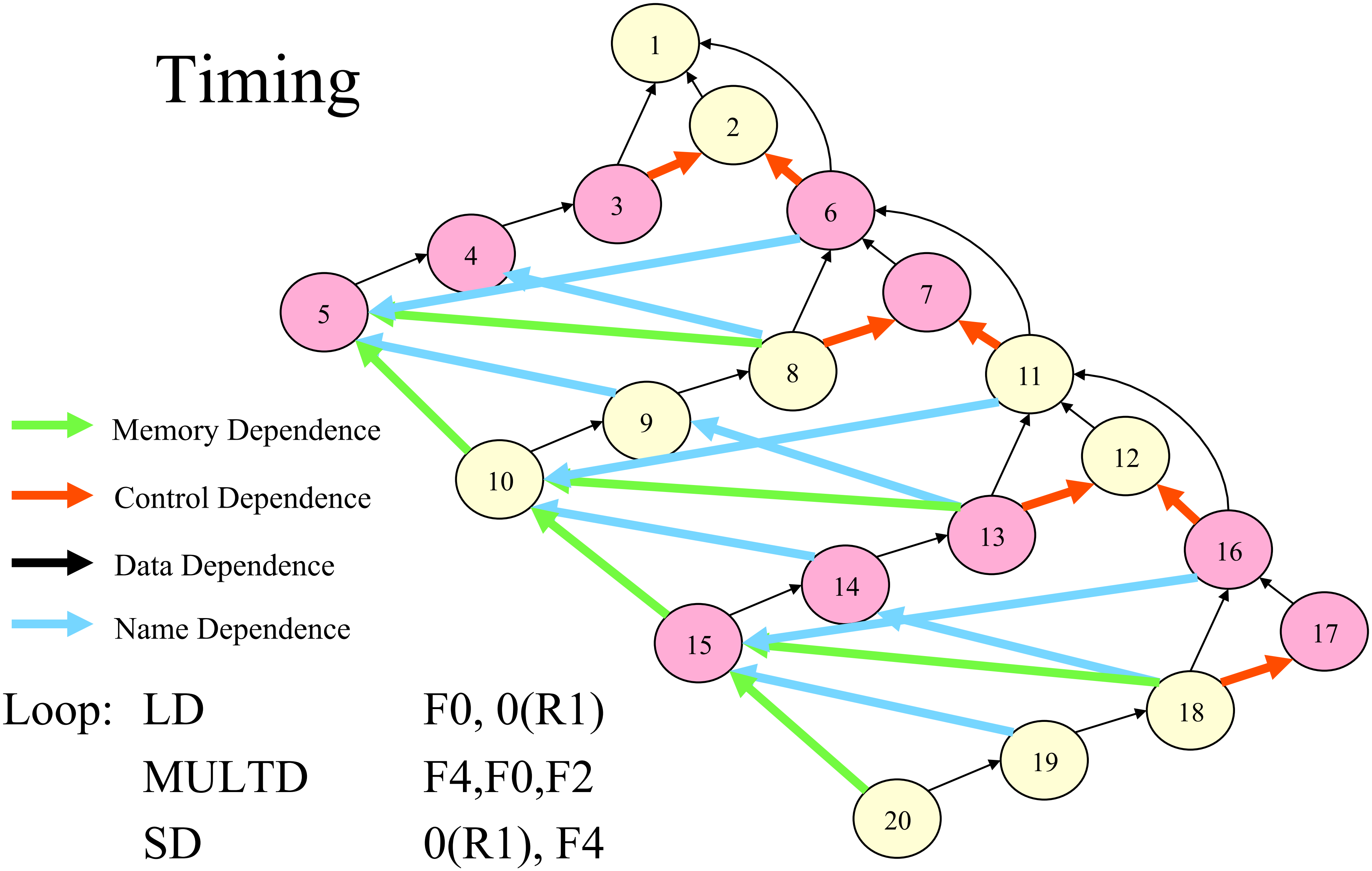
🟥 Control Dependence

⬛ Data Dependence

🟦 Name Dependence

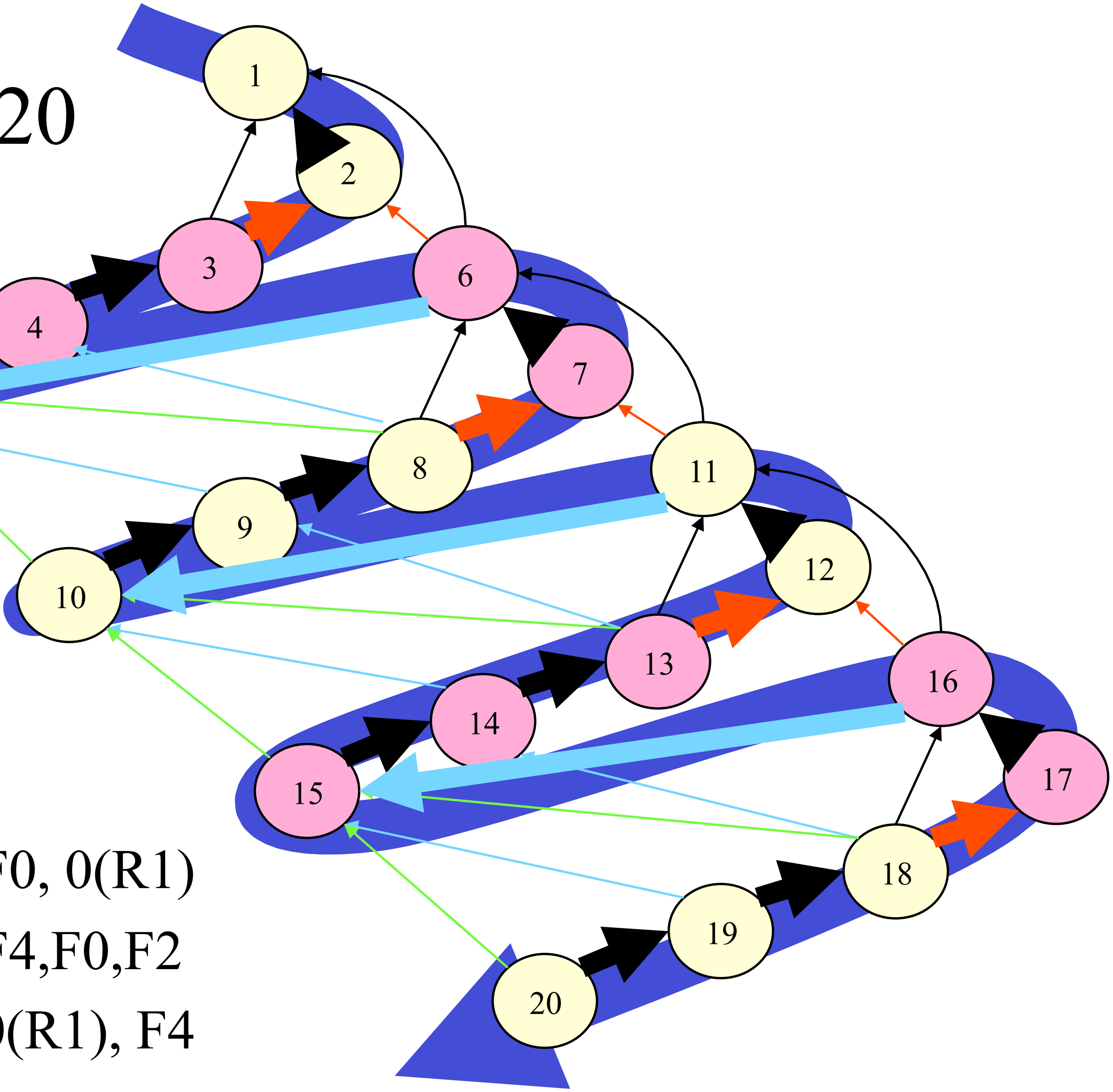
```
Loop:  LD          F0, 0(R1)
       MULTD       F4,F0,F2
       SD          0(R1), F4
       SUBI        R1,R1,#8
       BNEZ        R1, Loop
```

Timing



```
Loop:  LD        F0, 0(R1)
       MULTD     F4,F0,F2
       SD        0(R1), F4
       SUBI      R1,R1,#8
       BNEZ      R1, Loop
```

Critical Path = 20



Memory Dependence

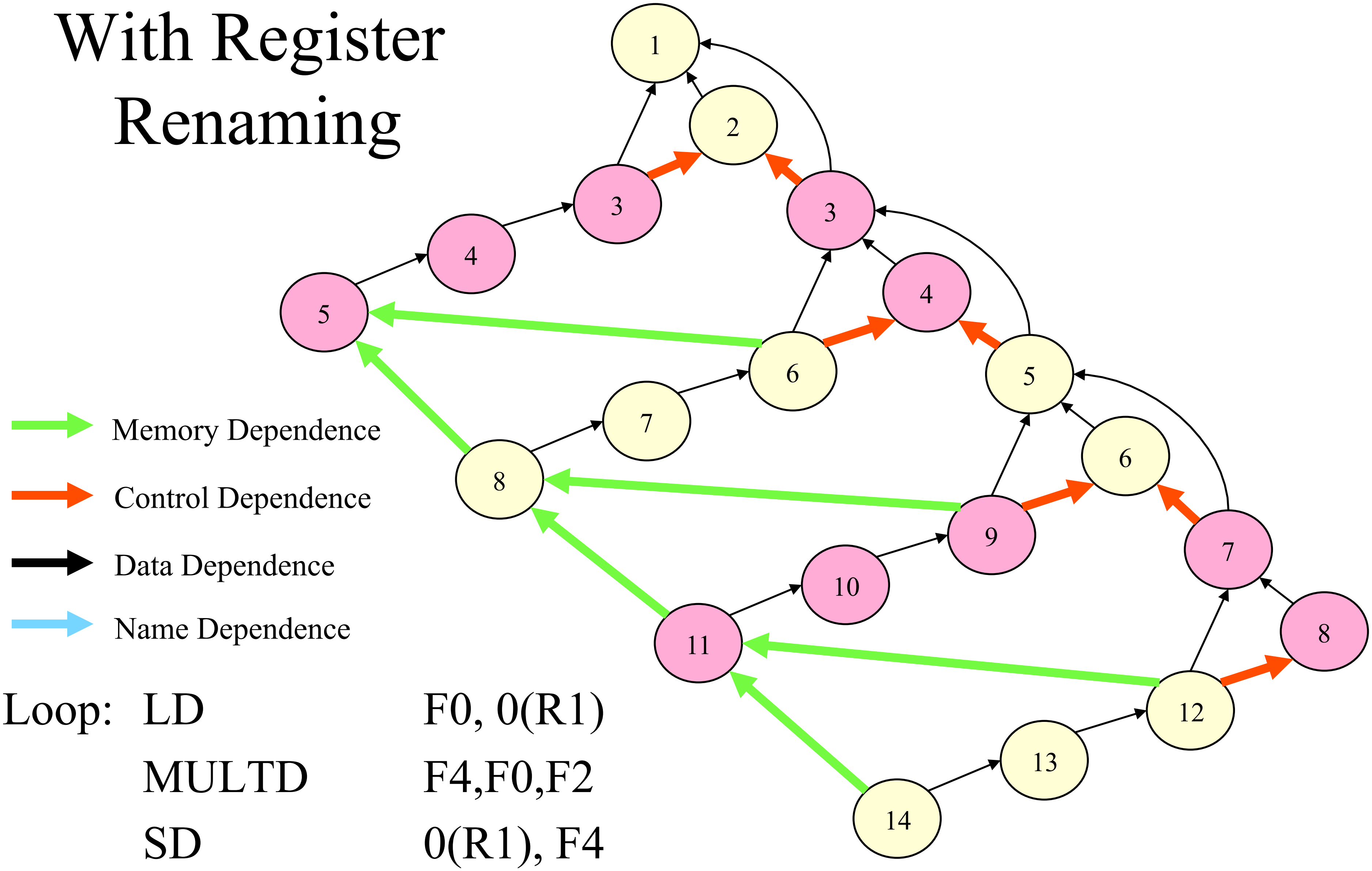
Control Dependence

Data Dependence

Name Dependence

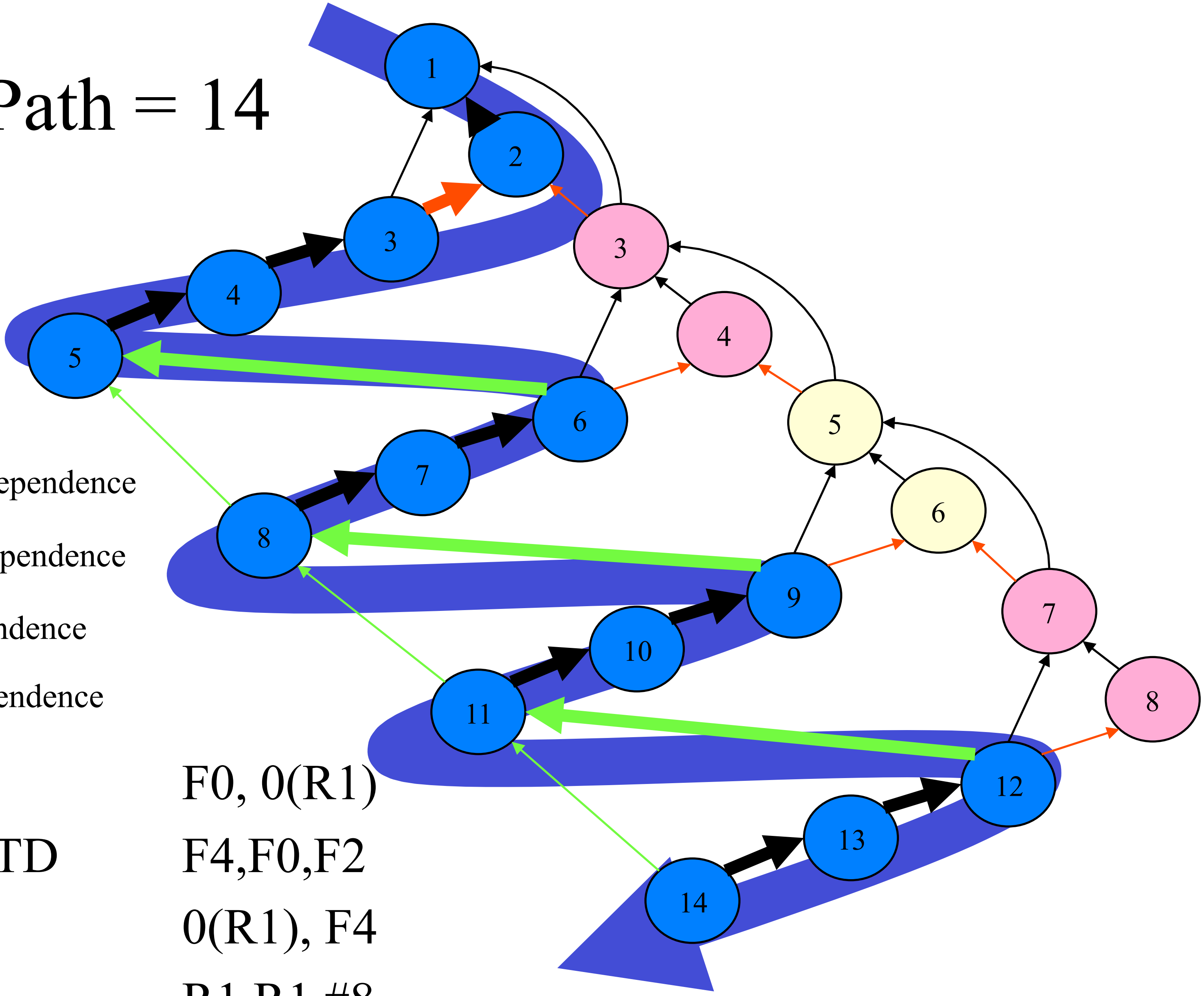
```
Loop:  LD        F0, 0(R1)
       MULTD     F4,F0,F2
       SD        0(R1), F4
       SUBI      R1,R1,#8
       BNEZ      R1, Loop
```

With Register Renaming



```
Loop:  LD          F0, 0(R1)
       MULTD       F4,F0,F2
       SD          0(R1), F4
       SUBI        R1,R1,#8
       BNEZ        R1, Loop
```

Critical Path = 14



🟩 Memory Dependence

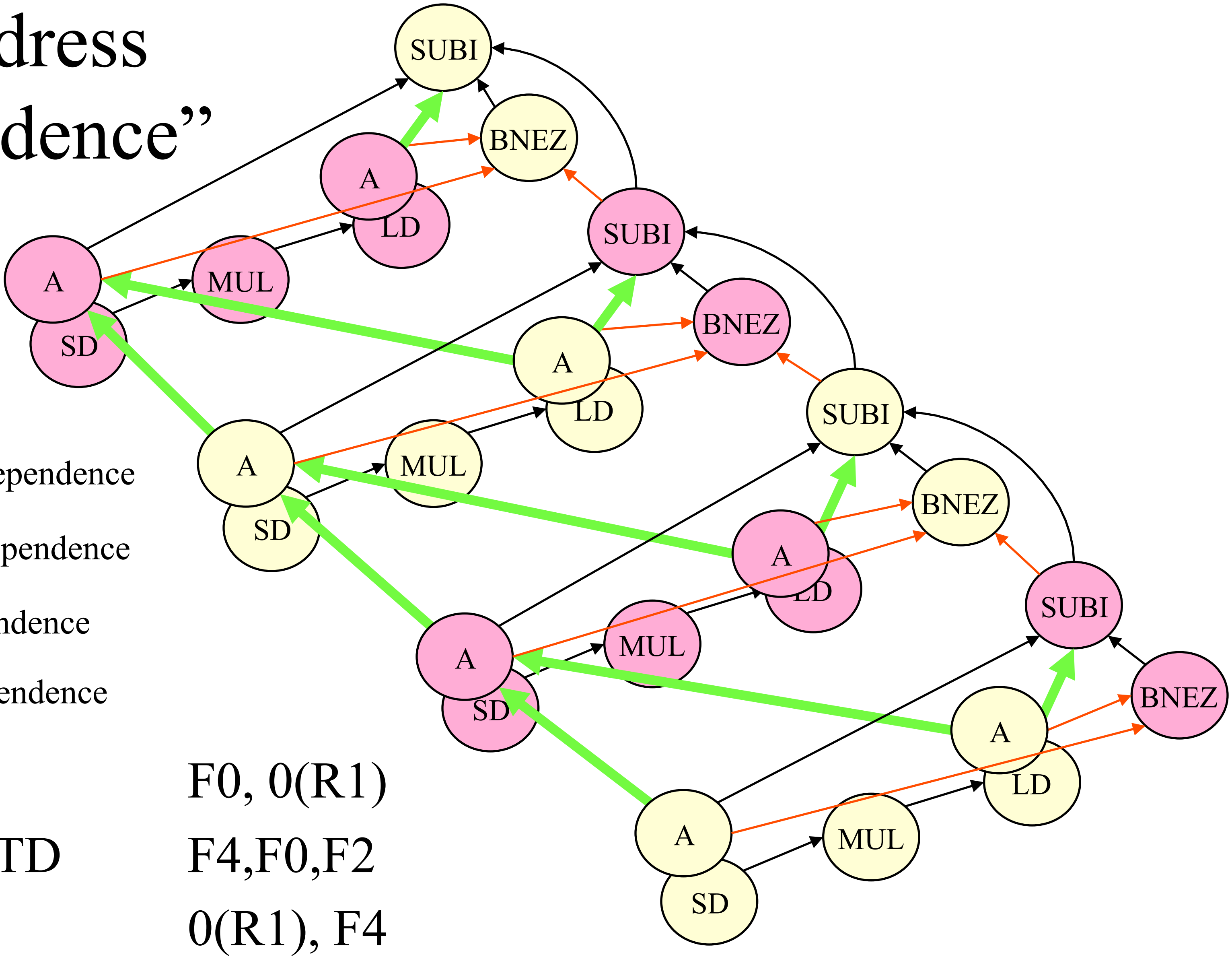
🟥 Control Dependence

⬛ Data Dependence

🟦 Name Dependence

```
Loop:  LD       F0, 0(R1)
       MULTD    F4,F0,F2
       SD       0(R1), F4
       SUBI     R1,R1,#8
       BNEZ     R1, Loop
```

“Address Dependence”



🟩 Address Dependence

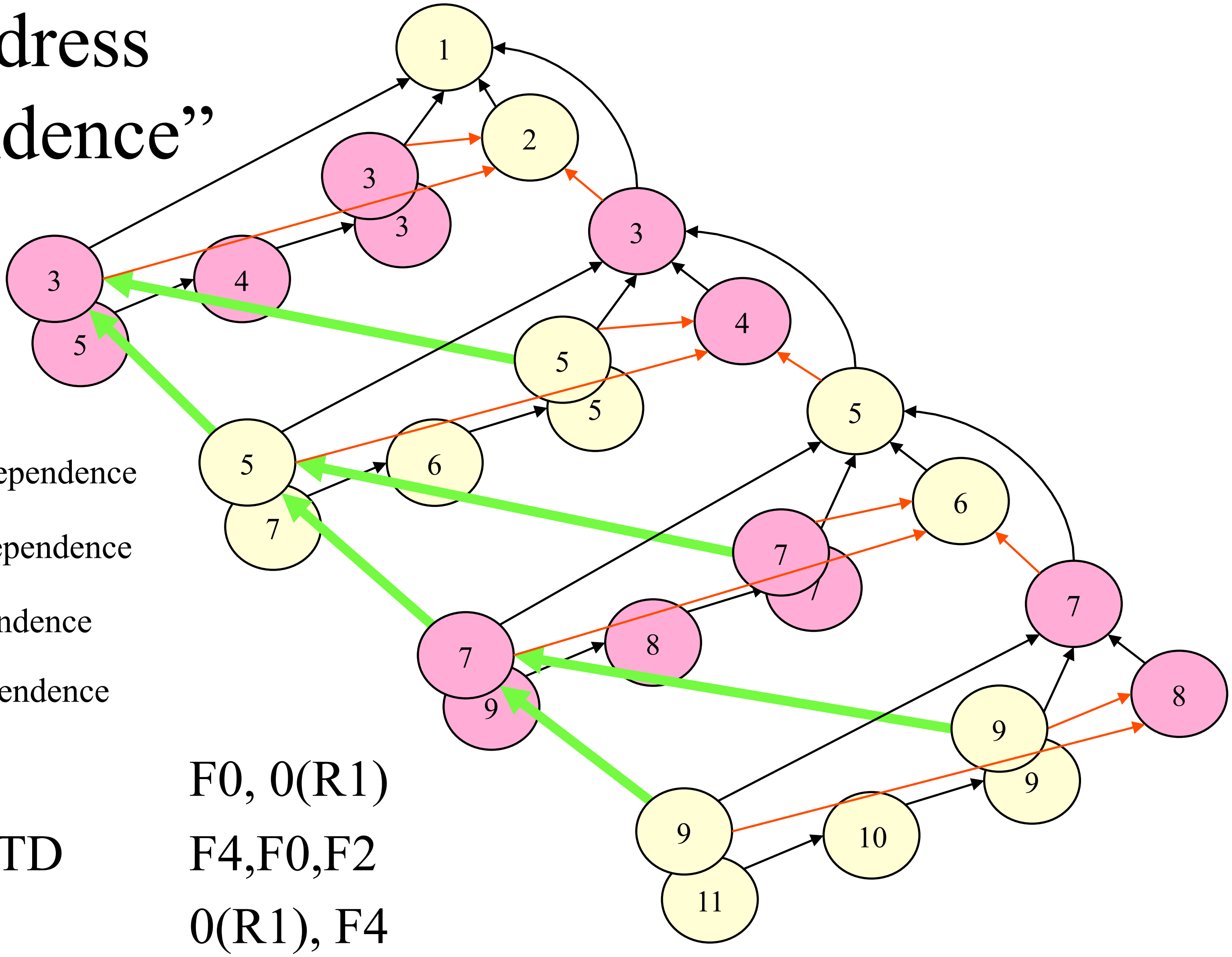
🟥 Control Dependence

→ Data Dependence

🟦 Name Dependence

```
Loop:   LD          F0, 0(R1)
        MULTD       F4,F0,F2
        SD          0(R1), F4
        SUBI        R1,R1,#8
        BNEZ        R1, Loop
```

“Address Dependence”



→ Address Dependence

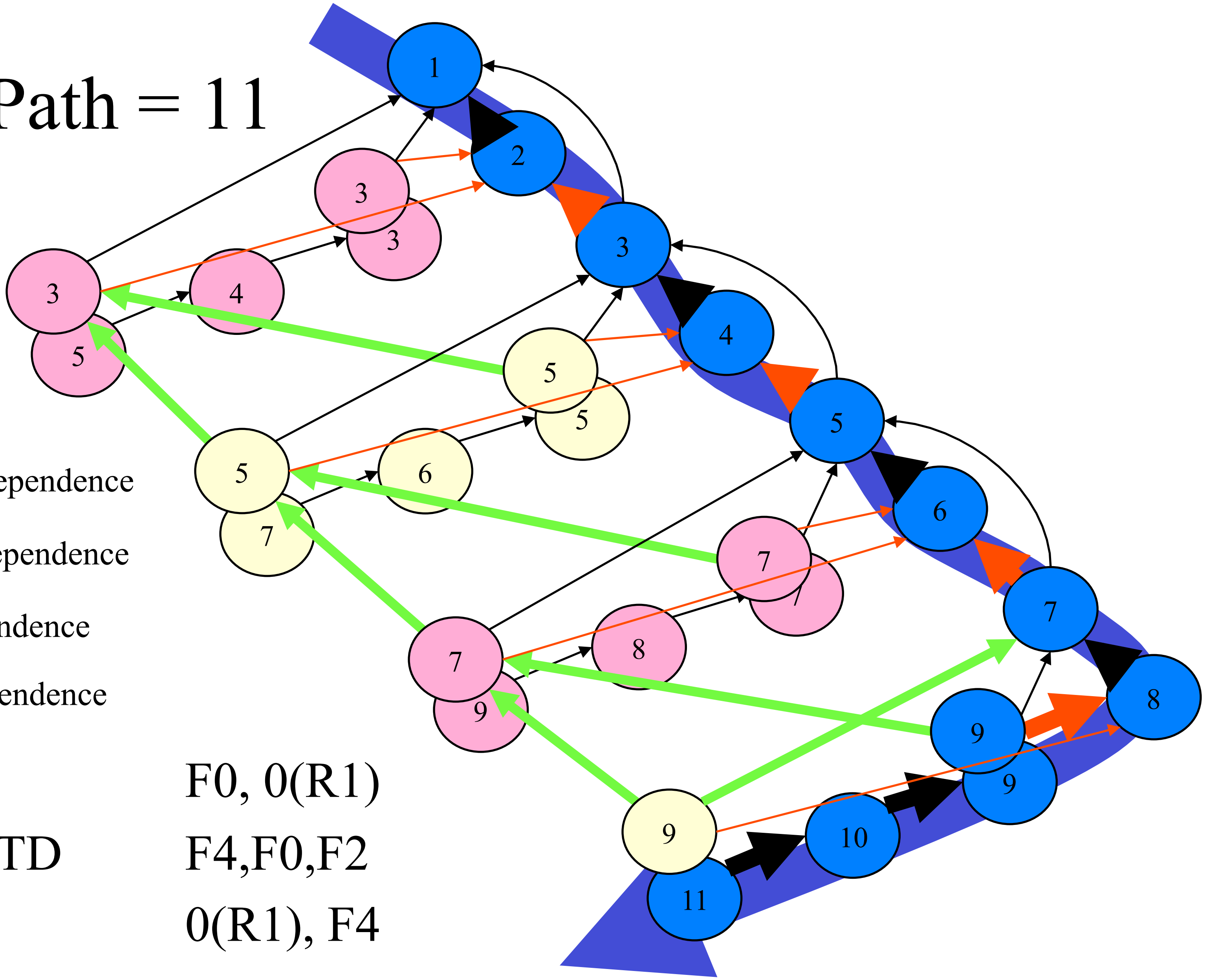
→ Control Dependence

→ Data Dependence

→ Name Dependence

```
Loop:  LD       F0, 0(R1)
       MULTD    F4,F0,F2
       SD       0(R1), F4
       SUBI     R1,R1,#8
       BNEZ     R1, Loop
```

Critical Path = 11



→ Address Dependence

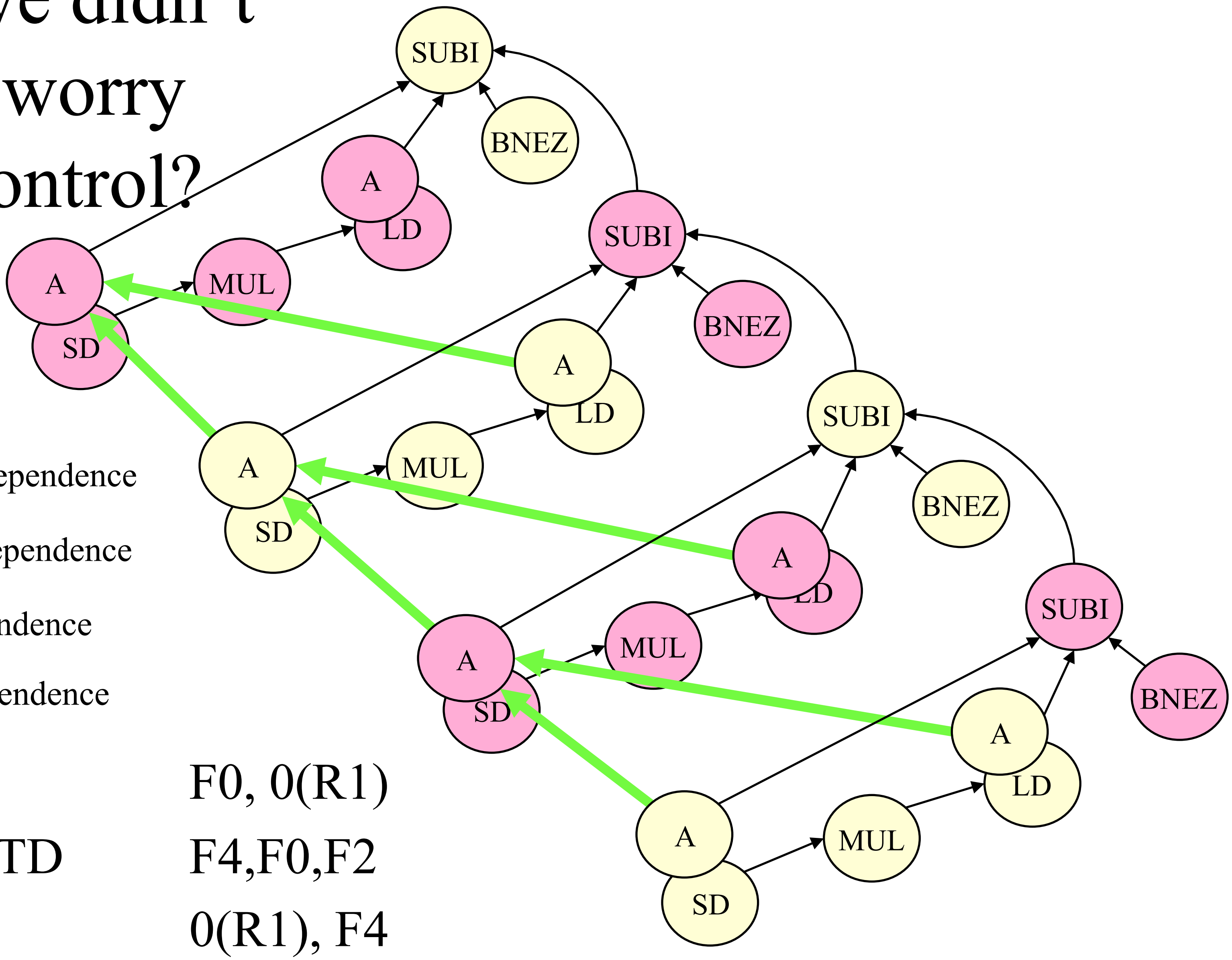
→ Control Dependence

→ Data Dependence

→ Name Dependence

```
Loop:  LD       F0, 0(R1)
       MULTD    F4,F0,F2
       SD       0(R1), F4
       SUBI     R1,R1,#8
       BNEZ     R1, Loop
```

What if we didn't have to worry about control?



🟩 Address Dependence

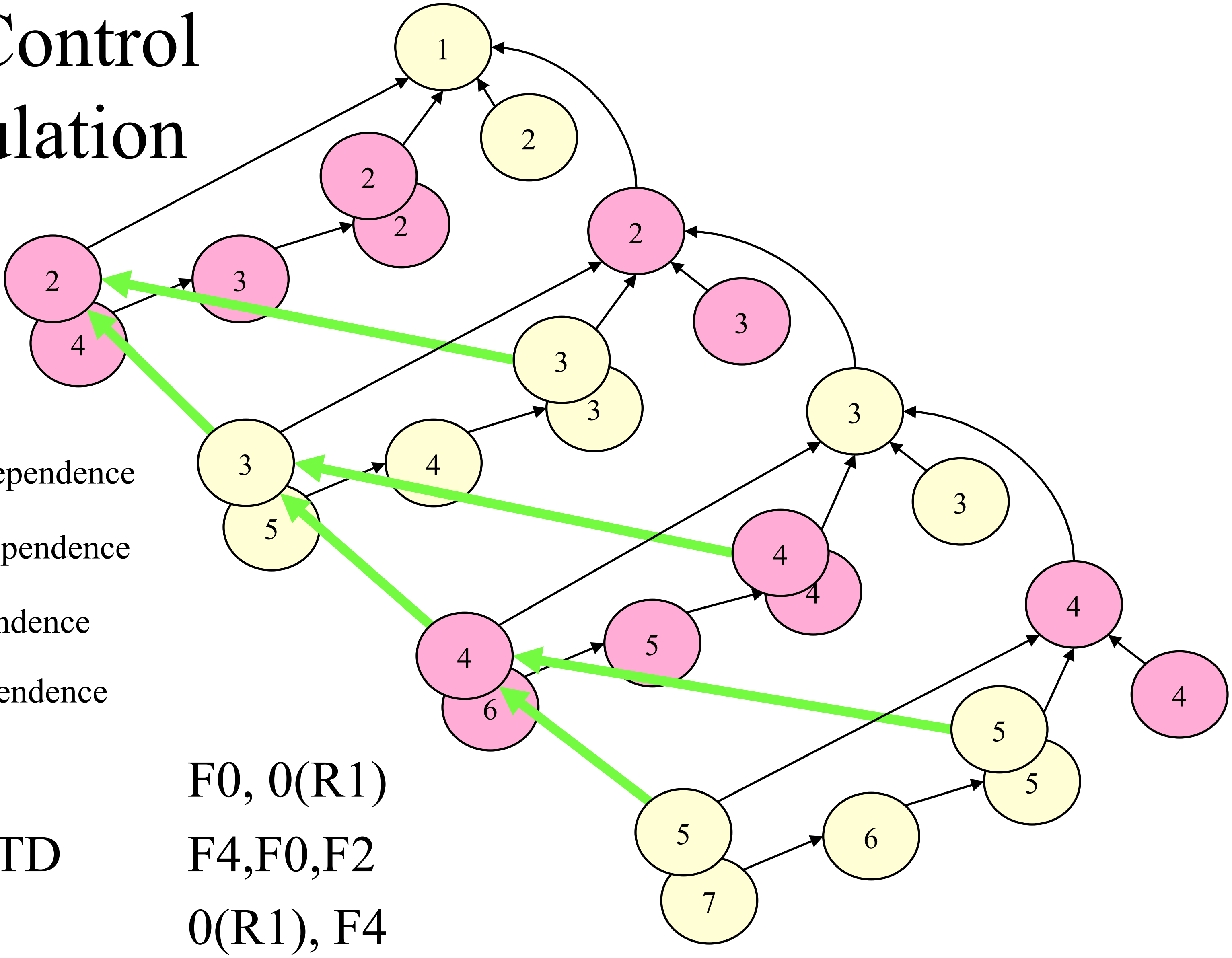
🟥 Control Dependence

■ Data Dependence

🟦 Name Dependence

```
Loop:  LD        F0, 0(R1)
       MULTD     F4,F0,F2
       SD        0(R1), F4
       SUBI      R1,R1,#8
       BNEZ      R1, Loop
```

With Control Speculation



→ Address Dependence

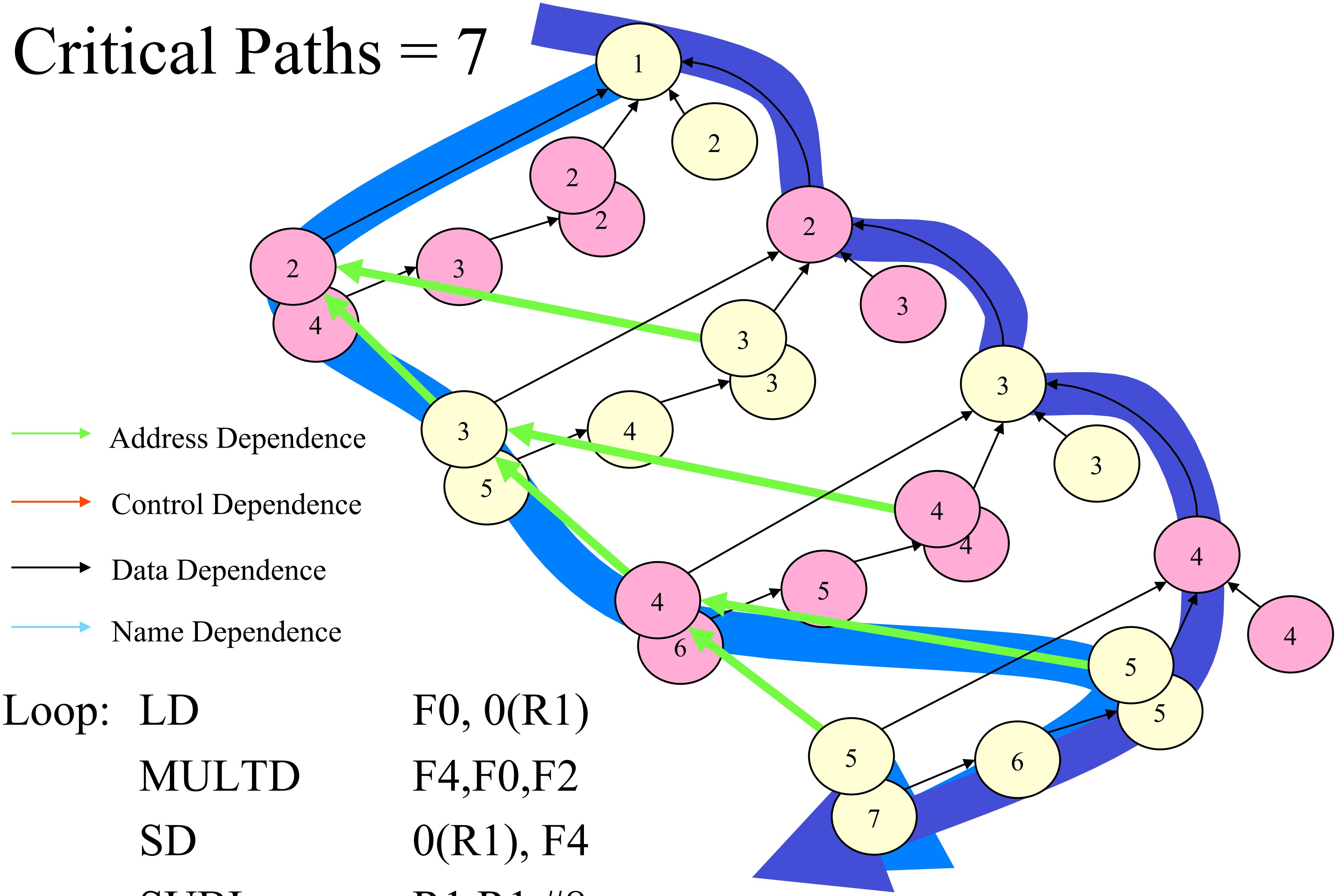
→ Control Dependence

→ Data Dependence

→ Name Dependence

```
Loop:  LD        F0, 0(R1)
       MULTD     F4,F0,F2
       SD        0(R1), F4
       SUBI      R1,R1,#8
       BNEZ      R1, Loop
```

Critical Paths = 7



```
Loop:  LD        F0, 0(R1)
       MULTD     F4,F0,F2
       SD        0(R1), F4
       SUBI      R1,R1,#8
       BNEZ      R1, Loop
```

Why Can Tomasulo Overlap Iterations of Loops?

- Register renaming
  - –Multiple iterations use different physical destinations for registers (dynamic loop unrolling).
  - –Replace static register names from code with dynamic register “pointers”
  - –Effectively increases size of register file
- Crucial: integer unit must “get ahead” of floating point unit so that we can issue multiple iterations
- Other idea: Tomasulo building “DataFlow” graph.

Key Points to Dynamic Scheduling

- Dynamic scheduling is instruction scheduling in hardware
  - allows out-of-order execution

- Register renaming eliminates WAW and WAR dependencies

- To get cross-iteration parallelism, need to eliminate WAR and WAW dependencies.

- Dynamic scheduling can do things SW scheduling cannot
  - execute instructions past instructions with unpredictable stalls
  - disambiguate may-alias memory references
  - rename across loop boundaries without code duplication

Progression of Execution Techniques

Issue	Dynamic Scheduling	Renaming	Speculative Execution	Completion/ Exceptions	Machine
In-order	No	Limited	Limited	In-order/ Precise	5-stage MIPS
In-order	No	No	No	Out-of-order/ Precise	US 1
In-order	Yes	Yes	No	Out-of-order/ Imprecise	Tomasulo
In-order	Yes	Yes	Yes	In-order/ Precise	Tomasulo + Reorder Buffer

* * *