6.823 SPring15
Quiz 4 Review (L20-L24)

TA: Po-An Tsai, Hsin-Jung Yang

Transactional memory

- Atomicity (all or nothing)
  - At commit, all memory writes take effect at once
  - On abort, none of the writes appear to take effect

- Isolation
  - No other code can observe writes before commit

- Serializability
  - Transactions seem to commit in a single serial order
  - The exact order is not guaranteed

Data Management Policy

1. Eager versioning (undo-log based)
   - Update memory location directly
   - Maintain undo info in a log
   - Fast commits
   - Slow aborts

2. Lazy versioning (write-buffer based)
   - Buffer data until commit in a write buffer
   - Update actual memory locations at commit
   - Fast aborts
   - Slow commits

Conflict Detection

- 1. Pessimistic detection

  Check for conflicts during loads or stores

- 2. Optimistic detection

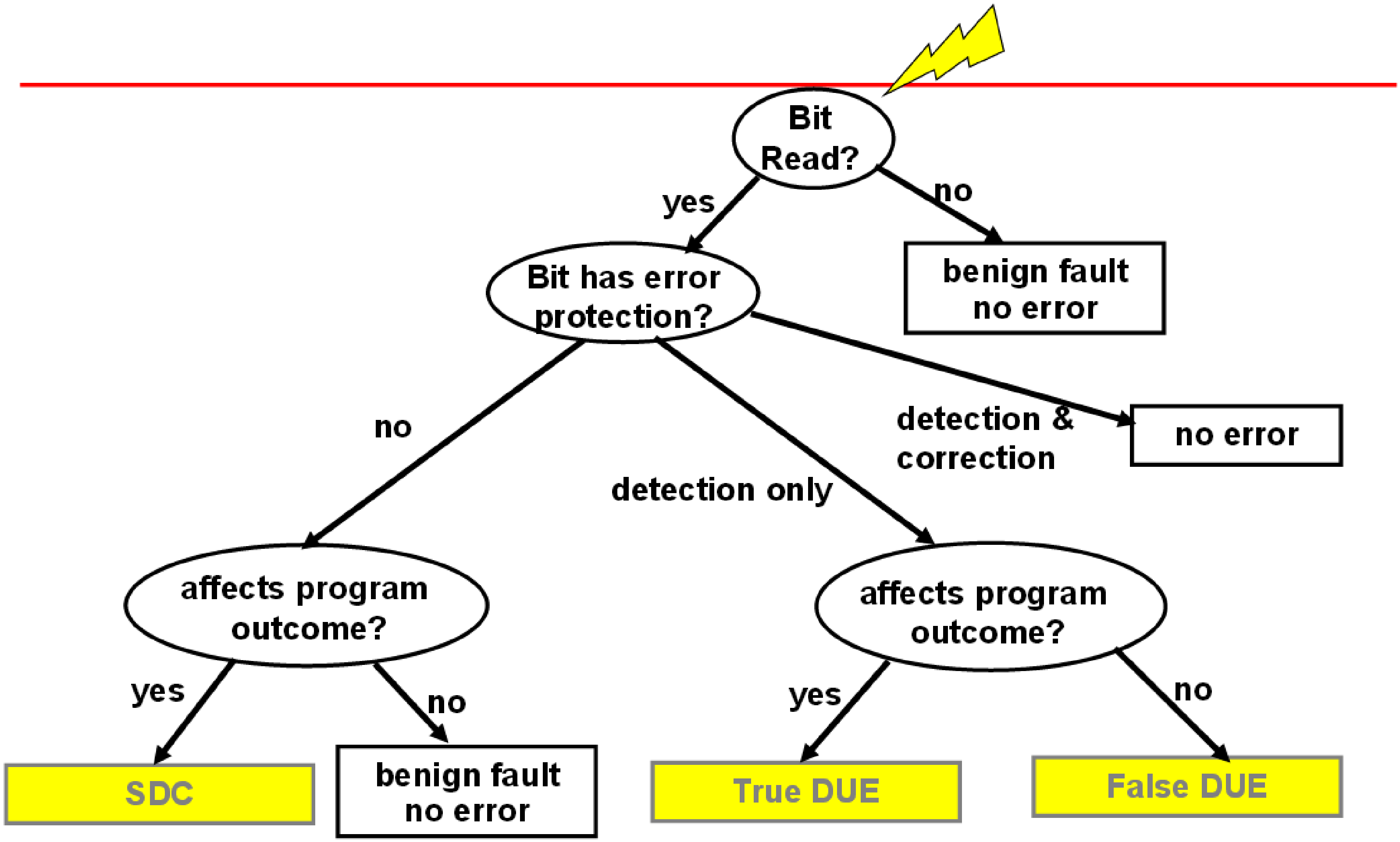
  Detect conflicts when a transaction attempts to commit

Reliability

- What happens if a bit flip
  - DUE : Detected Unrecoverable Error
  - SDC : Silent Data Corruption

- What structures are important
  - ACE : Architecturally Correct Execution
  - AVF : Architectural Vulnerability Factor

DUE and SDC

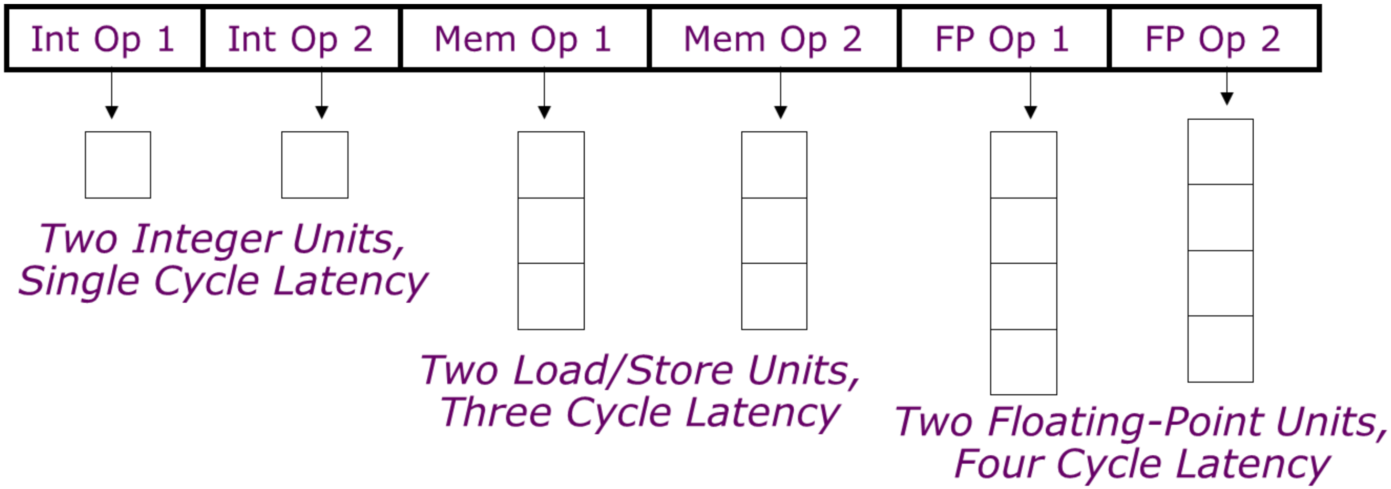


Bit Read?
yes
no
benign fault no error
Bit has error protection?
no
detection only
detection & correction
no error
affects program outcome?
yes
no
SDC
benign fault no error
affects program outcome?
yes
no
True DUE
False DUE

ACE and AVF

- Do the questions to make sure you understand how to calculate or argue for them

VLIW: Very Long Instruction Word



- Software (compiler) packs independent instructions into a large instruction
- Deterministic latency for all operations
  - Latency measured in 'instructions'
- Hardware does not perform dependency checks
  - Software adds explict NOP instructions

Loop Execution

```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

Compile

```
loop:  ld f1, 0(r1)
       add r1, 8
       fadd f2, f0, f1
       sd f2, 0(r2)
       add r2, 8
       bne r1, r3, loop
```

Schedule

loop:

Int1	Int 2	M1	M2	FP+	FPx
add r1		ld			
				fadd	
add r2	bne	sd			

How many FP ops/cycle?

1 fadd / 8 cycles = 0.125

- How to get denser packing?
  - Loop unrolling
  - Software pipelining

Loop Unrolling

- Unroll the inner loop to perform M iterations at once
  - To get more independent instructions
- Need to handle the case where the total iteration is not a multiple of M

```
loop:  ld f1, 0(r1)
       ld f2, 8(r1)
       ld f3, 16(r1)
       ld f4, 24(r1)
       add r1, 32
       fadd f5, f0, f1
       fadd f6, f0, f2
       fadd f7, f0, f3
       fadd f8, f0, f4
       sd f5, 0(r2)
       sd f6, 8(r2)
       sd f7, 16(r2)
       sd f8, 24(r2)
       add r2, 32
       bne r1, r3, loop
```

Schedule →

	Int1	Int 2	M1	M2	FP+	FPx
loop:			ld f1			
			ld f2			
			ld f3			
	add r1		ld f4		fadd f5	
					fadd f6	
					fadd f7	
					fadd f8	
			sd f5			
			sd f6			
			sd f7			
	add r2	bne	sd f8			

Unroll 4 iterations:

How many FP ops/cycle?

4 fadds / 11 cycles = 0.36

Software Pipelining

- Execute different iterations in parallel

```
loop:  ld f1, 0(r1)
       ld f2, 8(r1)
       ld f3, 16(r1)
       ld f4, 24(r1)
       add r1, 32
       fadd f5, f0, f1
       fadd f6, f0, f2
       fadd f7, f0, f3
       fadd f8, f0, f4
       sd f5, 0(r2)
       sd f6, 8(r2)
       sd f7, 16(r2)
       add r2, 32
       sd f8, -8(r2)
       bne r1, r3, loop
```

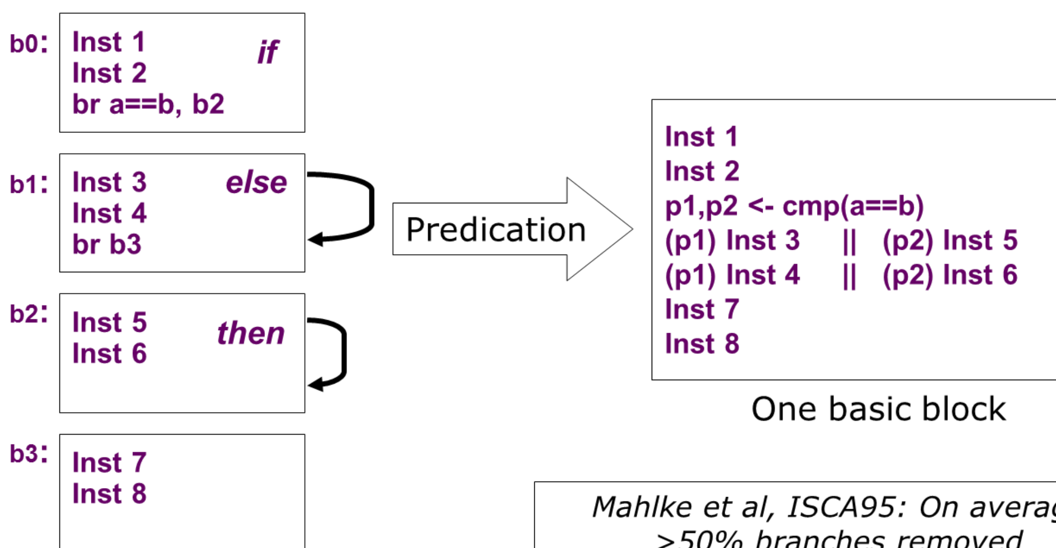
	Int1	Int 2	M1	M2	FP+	FPx
prolog			ld f1			
			ld f2			
			ld f3			
	add r1		ld f4			
			ld f1		fadd f5	
			ld f2		fadd f6	
			ld f3		fadd f7	
	add r1		ld f4		fadd f8	
iterate loop:			ld f1	sd f5	fadd f5	
			ld f2	sd f6	fadd f6	
		add r2	ld f3	sd f7	fadd f7	
	add r1	bne	ld f4	sd f8	fadd f8	
epilog				sd f5	fadd f5	
				sd f6	fadd f6	
		add r2		sd f7	fadd f7	
		bne		sd f8	fadd f8	
				sd f5		

Unroll 4 iterations and then do software pipelining:

How many FP ops/cycle? 4 fadds / 4 cycles = 1

Predicated Execution

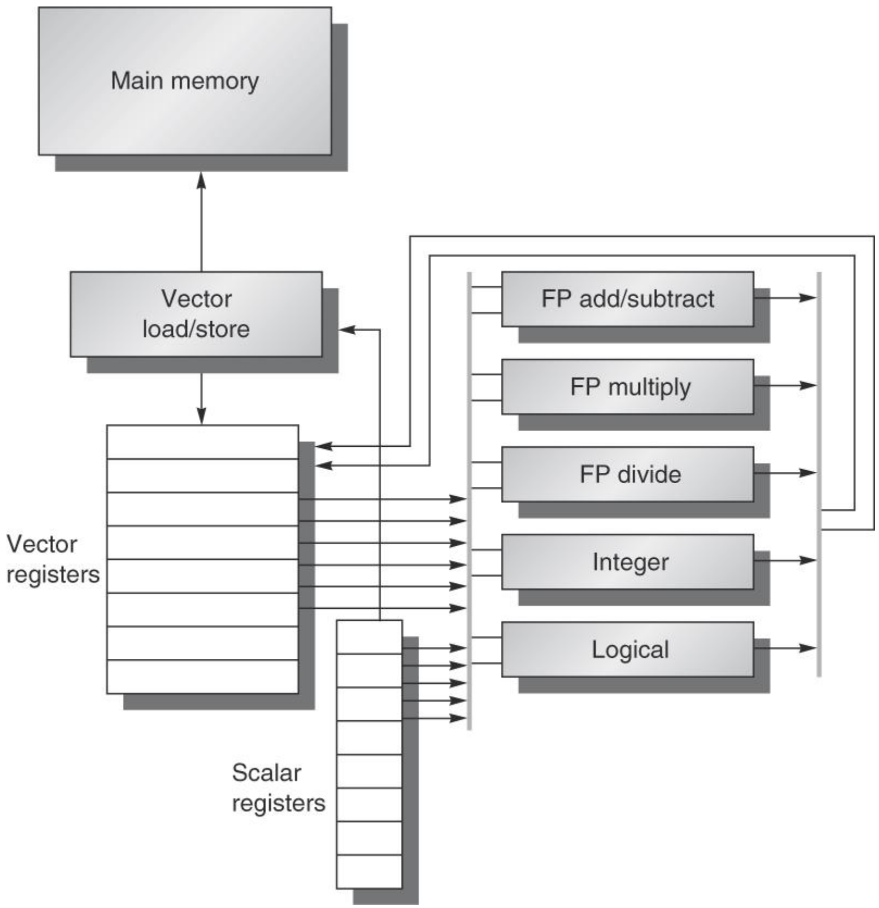
- Eliminate hard to predict branches with predicated execution
- Example: (p1) ADD r1, r2, r3
  - Execute (commit) add if the predicate register p1 is true



Four basic blocks

Mahlke et al, ISCA95: On average >50% branches removed

Vector Machine

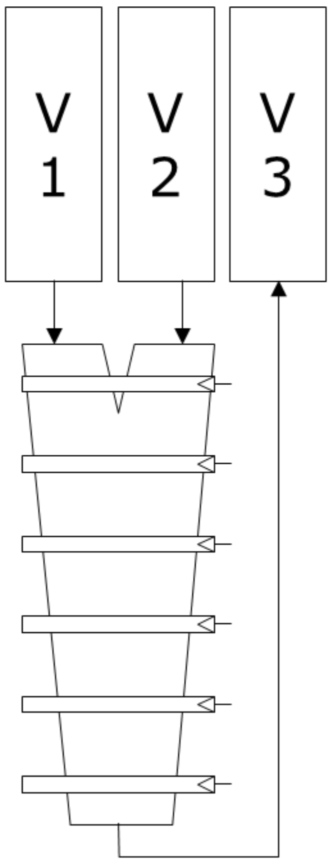


The basic structure of a vector architecture

- **Single Instruction Multiple Data (SIMD)**
  - Data level parallelism (DLP)
  - Single instruction for multiple loop iterations
  - Vector Length Register (VLR)
  - Conditional execution using the vector mask (VM) register

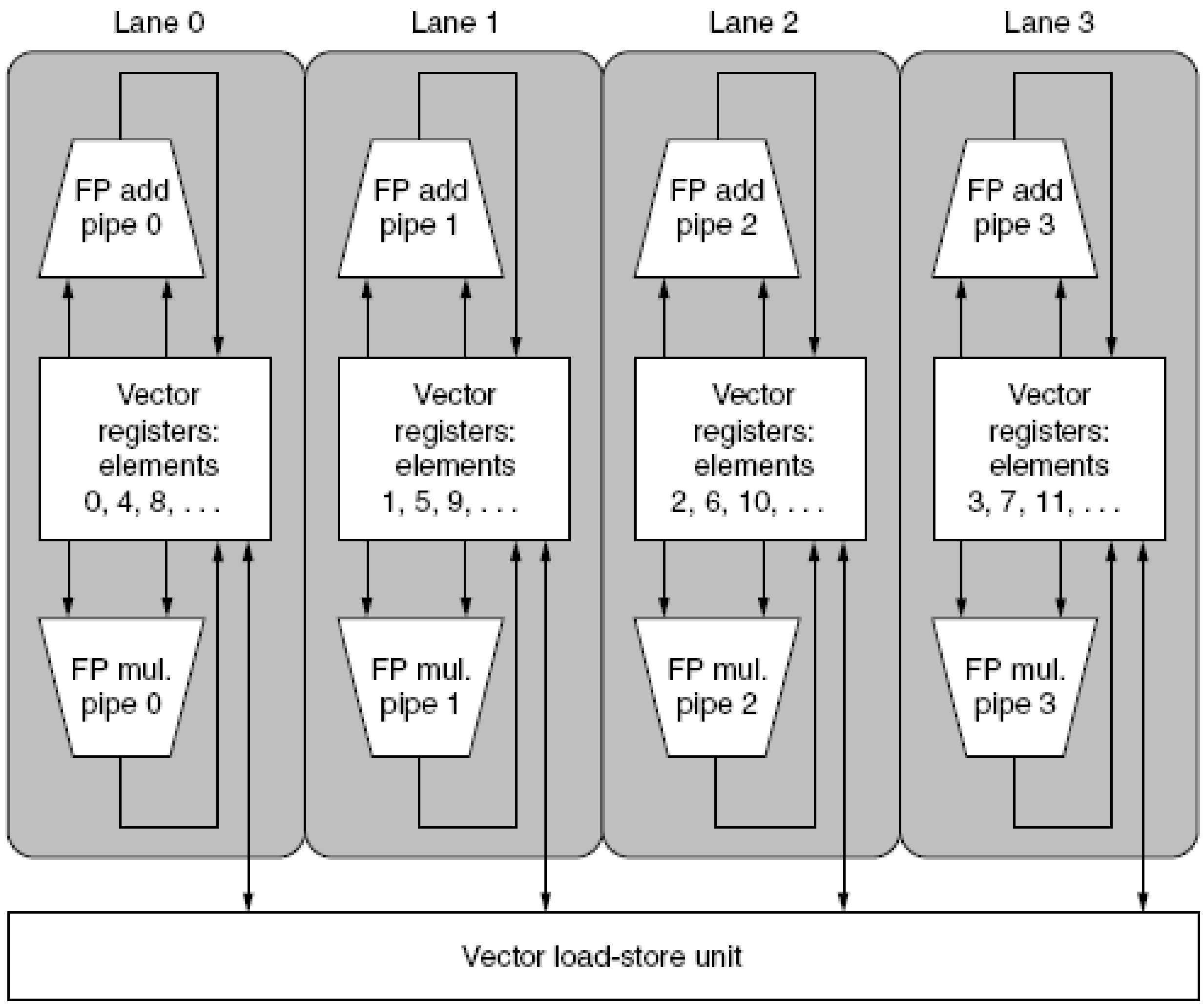
Vector Arithmetic Execution

- Use deep pipeline (=> fast clock) to execute element operations
- Simplifies control of deep pipeline because elements in vector are independent (=> no hazards!)



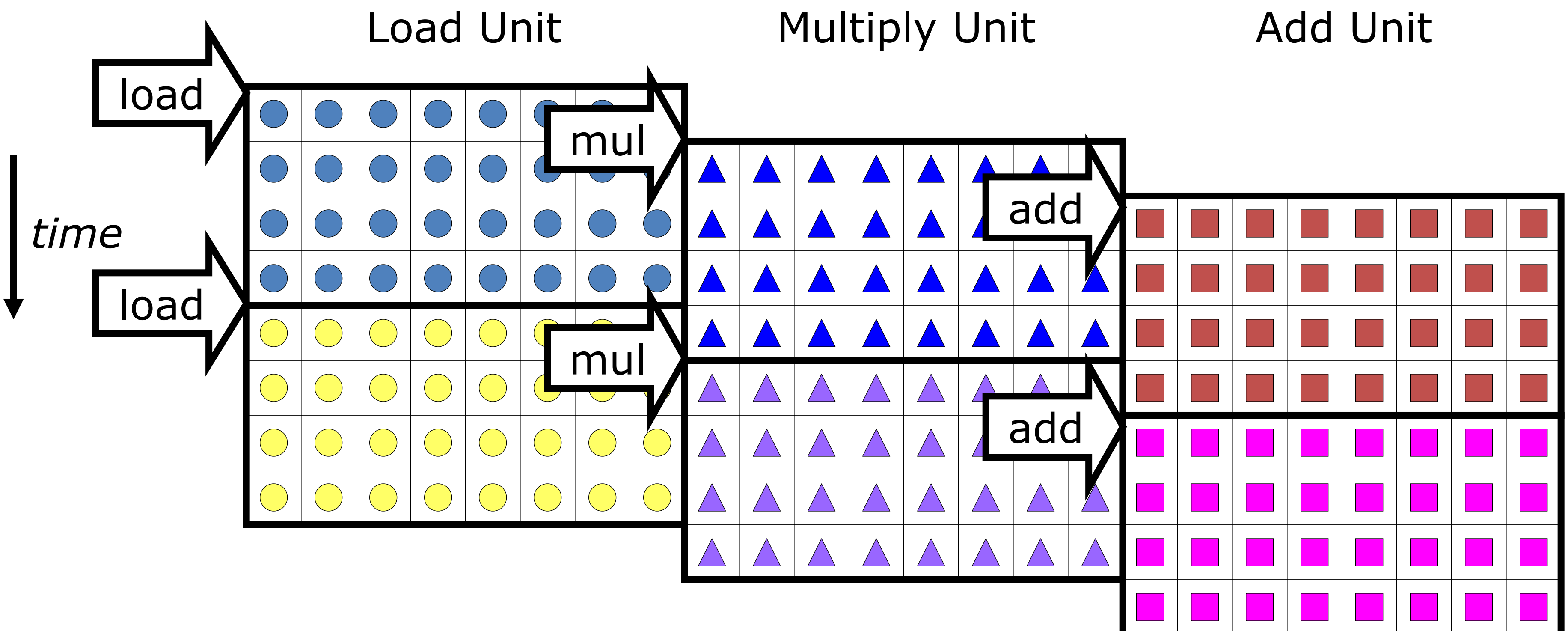
V3 <- v1 * v2

Multiple Lanes



Vector Instruction Parallelism

- Can overlap the execution of multiple vector instructions

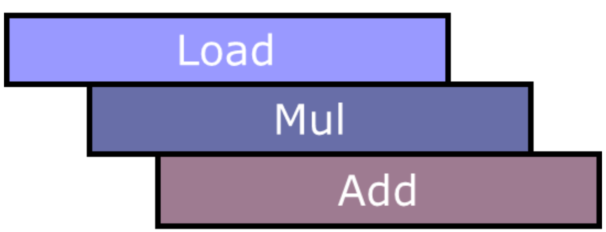


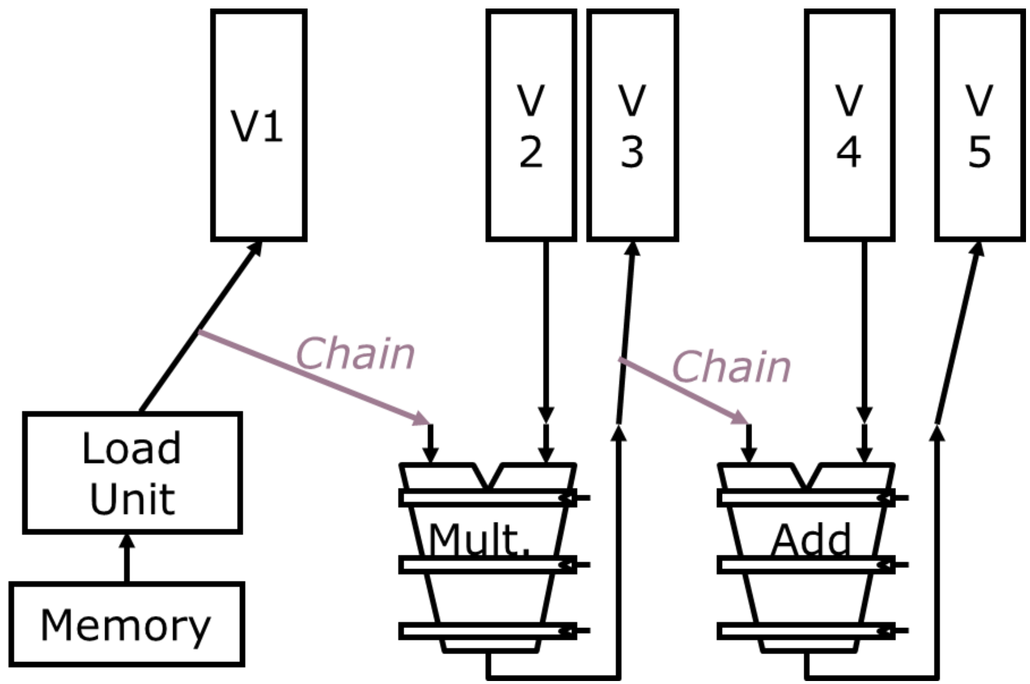
6 SIMD instructions executing on 8 lanes
with 32-element vector registers

Vector Chaining

- Allow a vector operation to start as soon as the individual elements of its vector source operand become available

```
LV   v1
MULV v3,v1,v2
ADDV v5, v3, v4
```

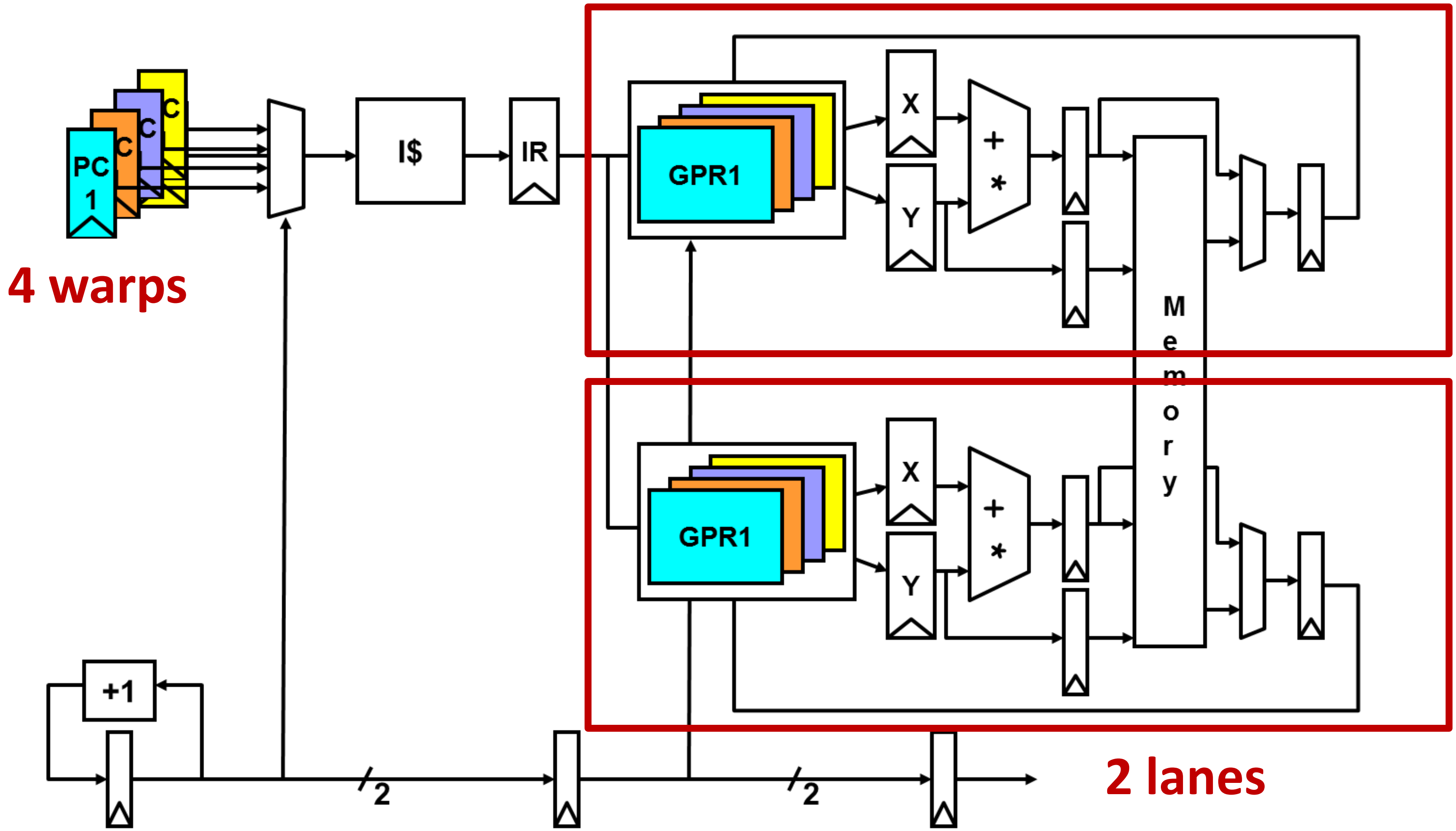




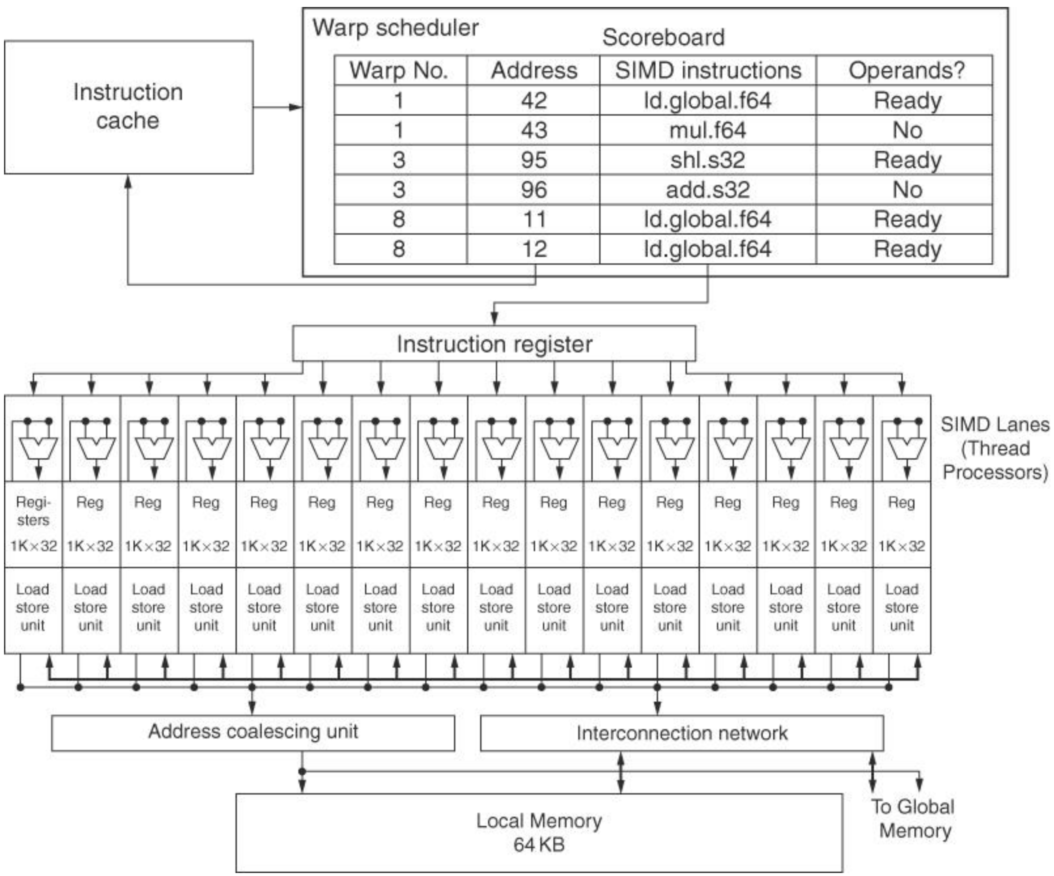
Graphical Processing Units (GPU)

- GPU consists of many multithreaded SIMD processors
- **Terminology**
  - **Thread:** a CUDA thread, which is associated with each data element
  - **Warp:** a traditional tread that contains SIMD instructions executed on a SIMD processor. Each SIMD instruction operates with multiple (CUDA) threads (ex: 32 threads/warp).
  - **Programming abstractions:**
    - **Thread block:** A group of threads (ex: 512 threads) mapped to a SIMD processor to execute a vectorized loop.
    - **Grid:** A grid has multiple thread blocks that can be mapped to different SIMD processors and execute independently.

Multithreaded SIMD Processor



Multithreaded SIMD Processor



Example:

- 16 physical lanes
- 10s of warps with 32 threads per warp
- Warp scheduler issues an SIMD instruction (for a specific warp) when its elements (threads in the warp) are all ready

GPU vs Vector Machine

- **Similarities:**
  - Works well with data-level parallel problems
  - Mask registers
  - Multiple lanes
  - Large register files

- **Differences:**
  - GPU uses multithreading to hide memory latency
  - GPU has many functional units, as opposed to a few deeply pipelined units like a vector processor

The end

Good luck!! ☺