CS 152 Computer Architecture and Engineering

Lecture 16: Vector Computers

Krste Asanovic
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~krste
http://inst.cs.berkeley.edu/~cs152

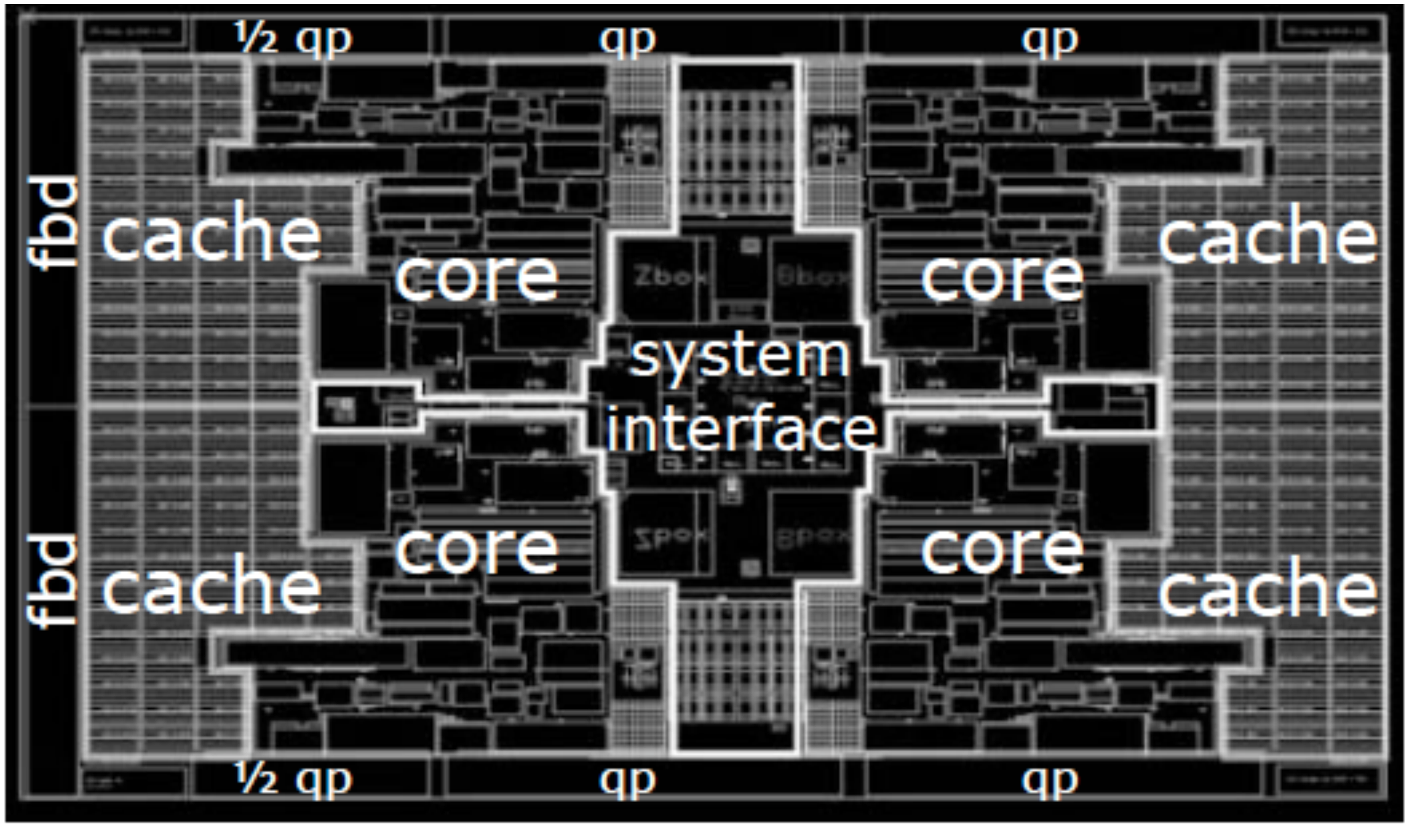
Last Time Lecture 15: VLIW

- In a classic VLIW, compiler is responsible for avoiding all hazards -> simple hardware, complex compiler. Later VLIWs added more dynamic hardware interlocks
- Use loop unrolling and software pipelining for loops, trace scheduling for more irregular code
- Static scheduling difficult in presence of unpredictable branches and variable latency memory

Intel EPIC IA-64

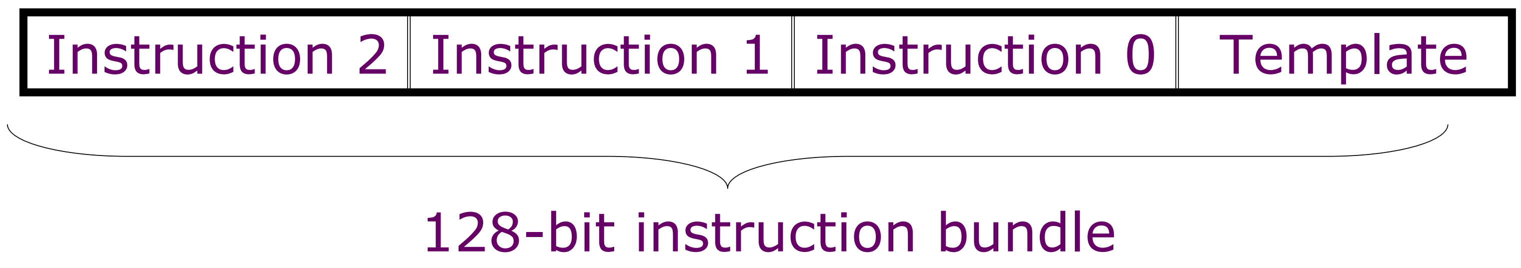
- EPIC is the style of architecture (cf. CISC, RISC)
  - Explicitly Parallel Instruction Computing
- IA-64 is Intel's chosen ISA (cf. x86, MIPS)
  - IA-64 = Intel Architecture 64-bit
  - An object-code compatible VLIW
- Itanium (aka Merced) is first implementation (cf. 8086)
  - First customer shipment expected 1997 (actually 2001)
  - McKinley, second implementation shipped in 2002
  - Recent version, Tukwila 2008, quad-cores, 65nm (not shipping until 2010?)

Quad Core Itanium "Tukwila" *[Intel 2008]*

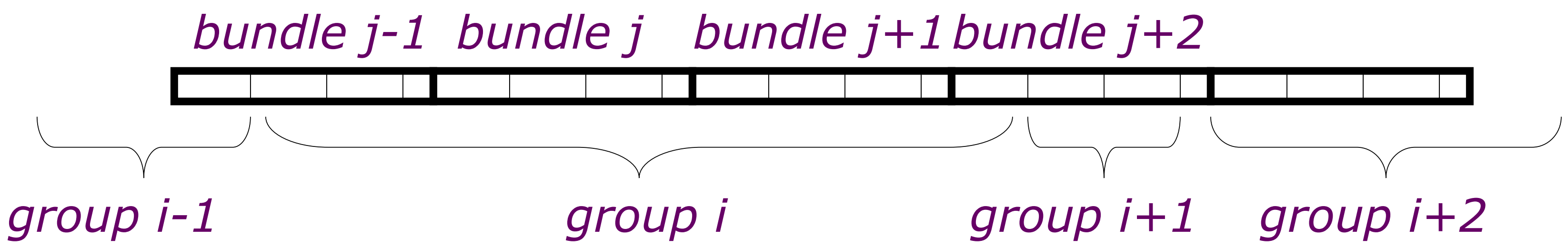


- 4 cores
- 6MB $/core, 24MB $ total
- ~2.0 GHz
- 698mm^2 in 65nm CMOS!!!!!
- 170W
- Over 2 billion transistor

IA-64 Instruction Format



- Template bits describe grouping of these instructions with others in adjacent bundles
- Each group contains instructions that can execute in parallel



IA-64 Registers

- 128 General Purpose 64-bit Integer Registers
- 128 General Purpose 64/80-bit Floating Point Registers
- 64 1-bit Predicate Registers

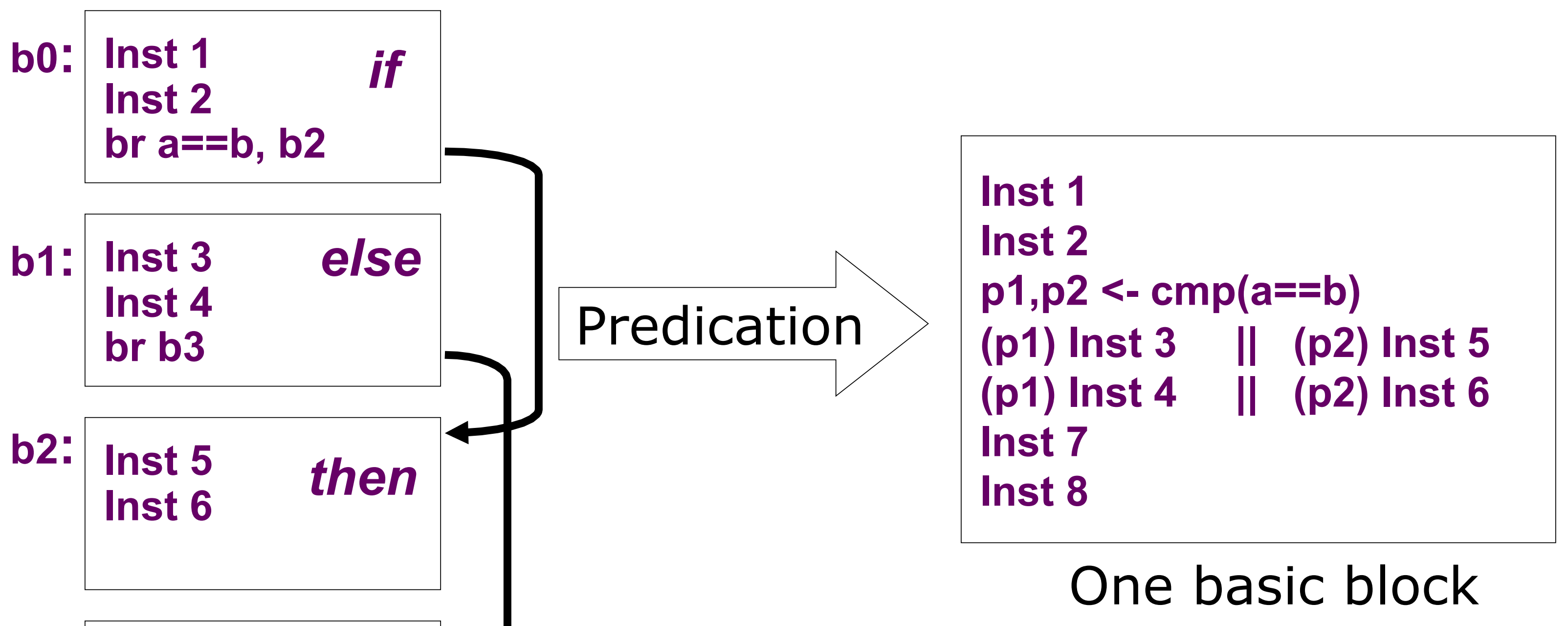
- GPRs rotate to reduce code size for software pipelined loops

IA-64 Predicated Execution

Problem: Mispredicted branches limit ILP

Solution: Eliminate hard to predict branches with predicated execution

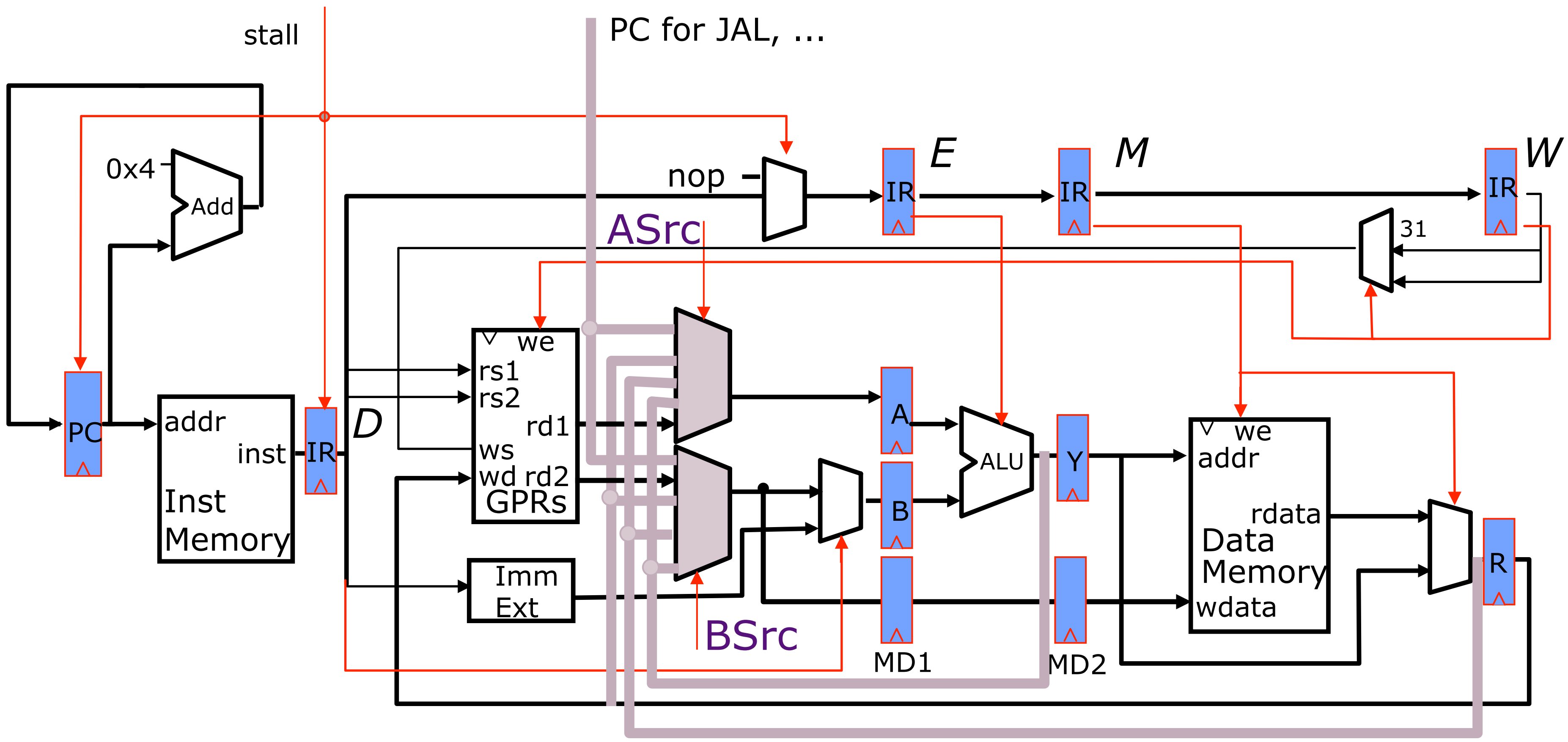
- Almost all IA-64 instructions can be executed conditionally under predicate
- Instruction becomes NOP if predicate register false



Four basic blocks

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

Fully Bypassed Datapath

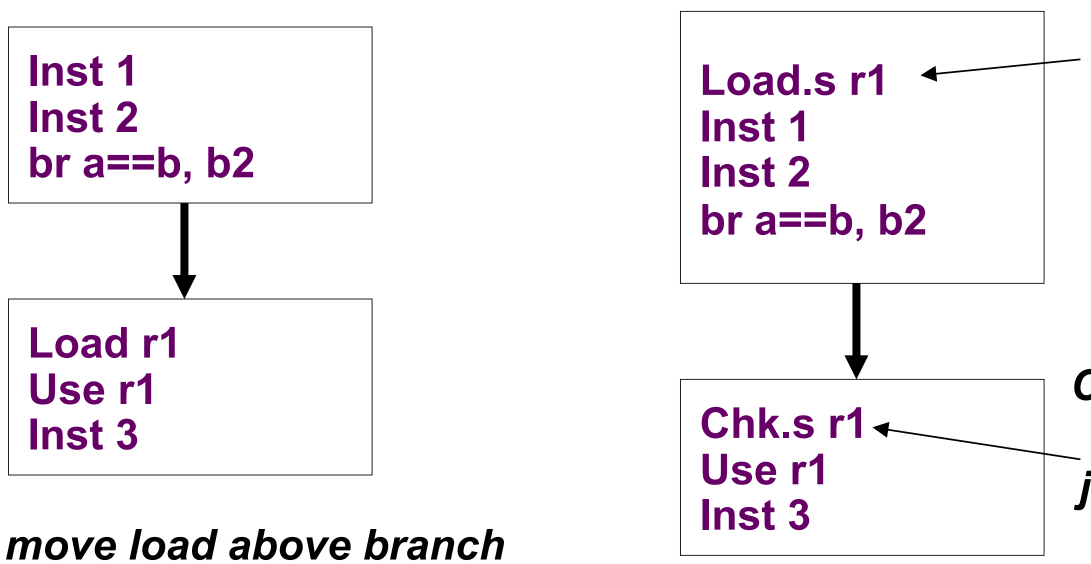


Where does predication fit in?

IA-64 Speculative Execution

Problem: Branches restrict compiler code motion

Solution: Speculative operations that don't cause exceptions



```
Inst 1
Inst 2
br a==b, b2
```

```
Load r1
Use r1
Inst 3
```

Can't move load above branch because might cause spurious exception

```
Load.s r1
Inst 1
Inst 2
br a==b, b2
```

Speculative load never causes exception, but sets "poison" bit on destination register

```
Chk.s r1
Use r1
Inst 3
```

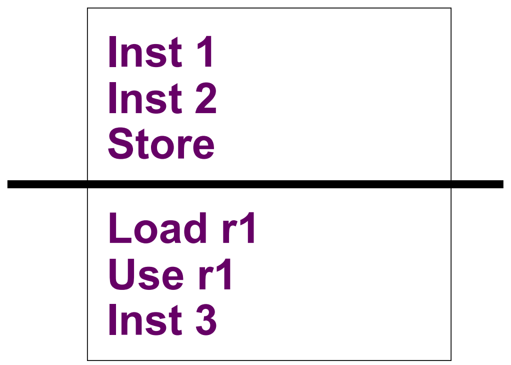
Check for exception in original home block jumps to fixup code if exception detected

Particularly useful for scheduling long latency loads early

IA-64 Data Speculation

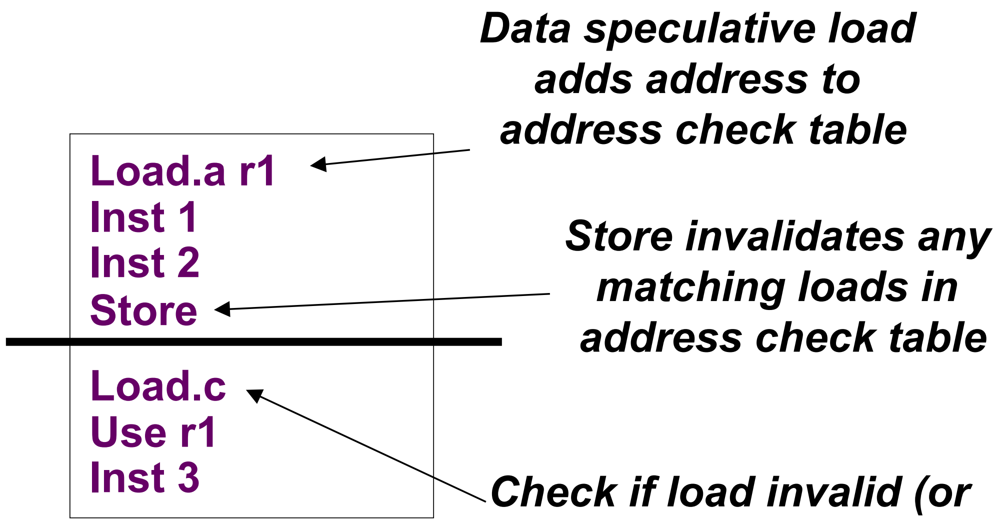
Problem: Possible memory hazards limit code scheduling

Solution: Hardware to check pointer hazards



```
Inst 1
Inst 2
Store
--------
Load r1
Use r1
Inst 3
```

Can't move load above store because store might be to same address



```
Load.a r1
Inst 1
Inst 2
Store
--------
Load.c
Use r1
Inst 3
```

Data speculative load adds address to address check table

Store invalidates any matching loads in address check table

Check if load invalid (or missing), jump to fixup code if so

Requires associative hardware in address check table

Limits of Static Scheduling

- Unpredictable branches
- Variable memory latency (unpredictable cache misses)
- Code size explosion
- Compiler complexity

Despite several attempts, VLIW has failed in general-purpose computing arena.

Successful in embedded DSP market.

Supercomputers

Definition of a supercomputer:

- Fastest machine in world at given task
- A device to turn a compute-bound problem into an I/O bound problem
- Any machine costing $30M+
- Any machine designed by Seymour Cray

CDC6600 (Cray, 1964) regarded as first supercomputer

Supercomputer Applications

Typical application areas

- Military research (nuclear weapons, cryptography)
- Scientific research
- Weather forecasting
- Oil exploration
- Industrial design (car crash simulation)
- Bioinformatics
- Cryptography

All involve huge computations on large data sets

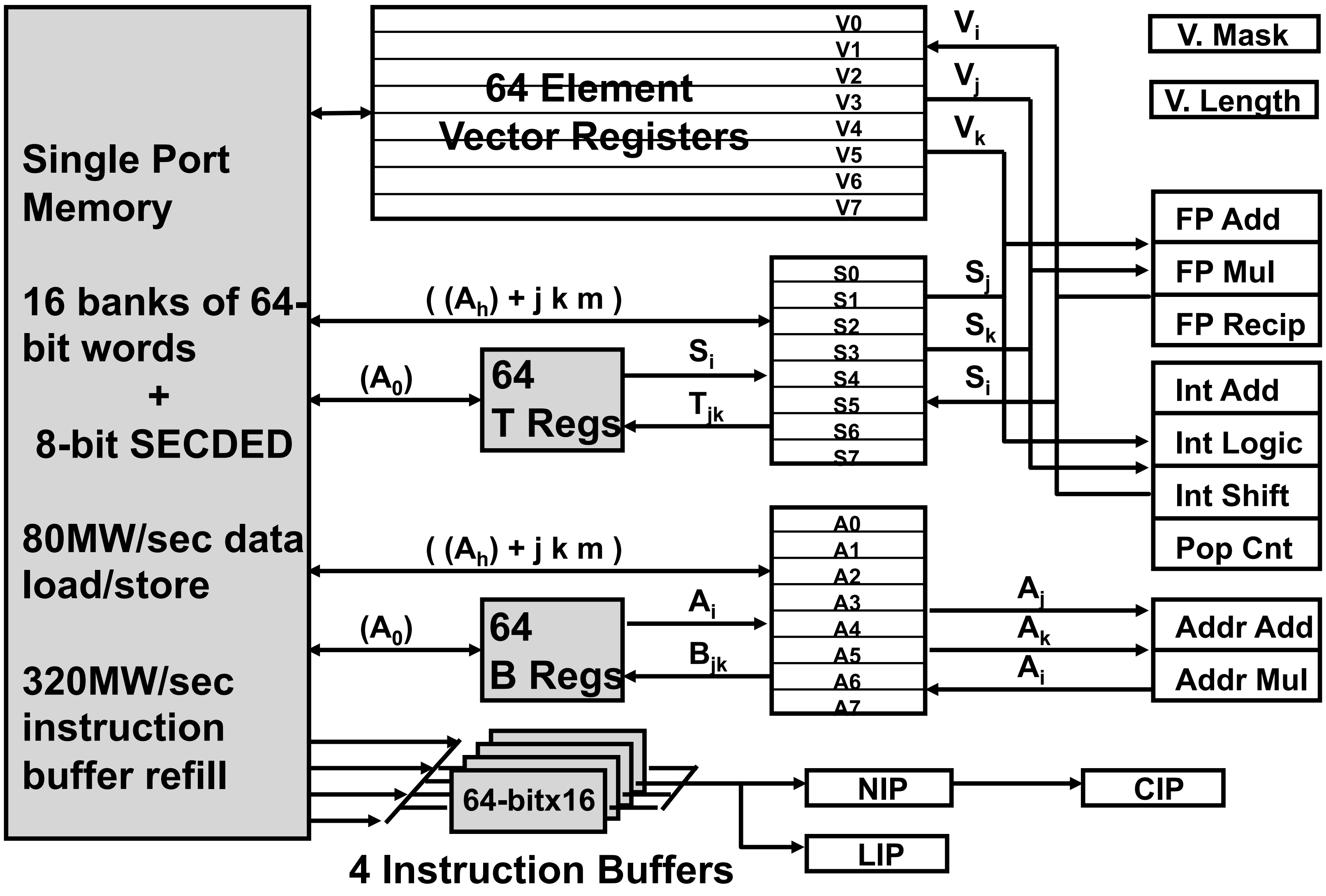
In 70s-80s, Supercomputer ≡ Vector Machine

Vector Supercomputers

Epitomized by Cray-1, 1976:

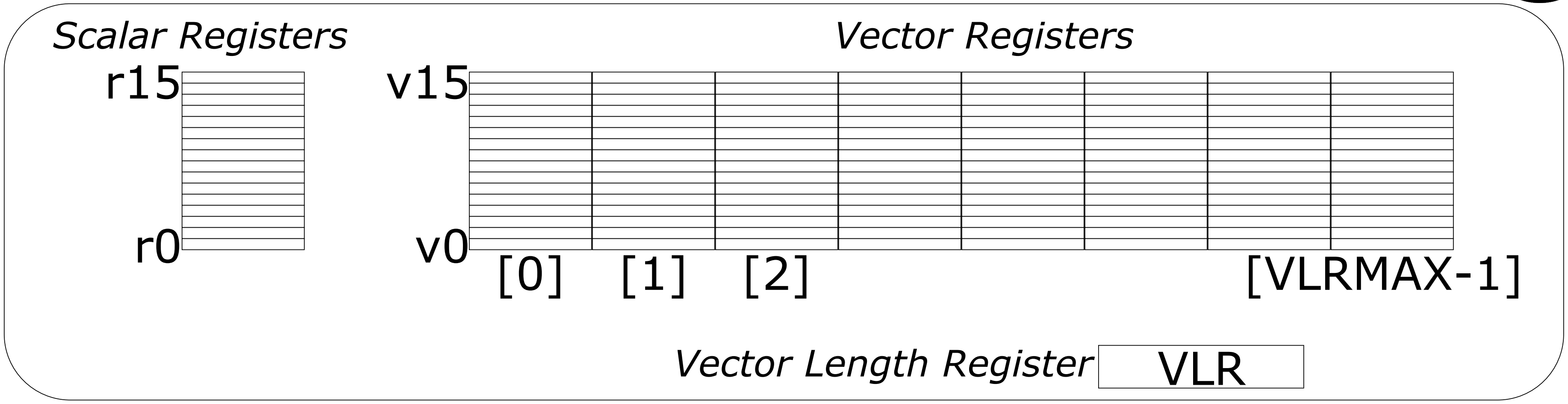
- Scalar Unit
  - Load/Store Architecture
- Vector Extension
  - Vector Registers
  - Vector Instructions
- Implementation
  - Hardwired Control
  - Highly Pipelined Functional Units
  - Interleaved Memory System
  - No Data Caches
  - No Virtual Memory

Cray-1 (1976)



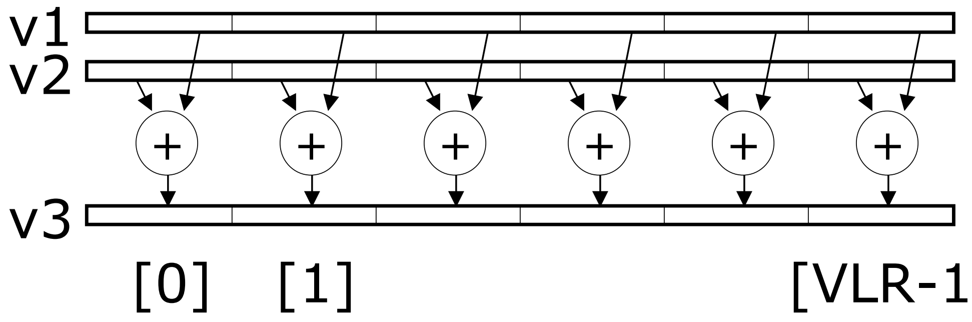
memory bank cycle **50 ns** ***processor cycle*** **12.5 ns (80MHz)**

Vector Programming Model



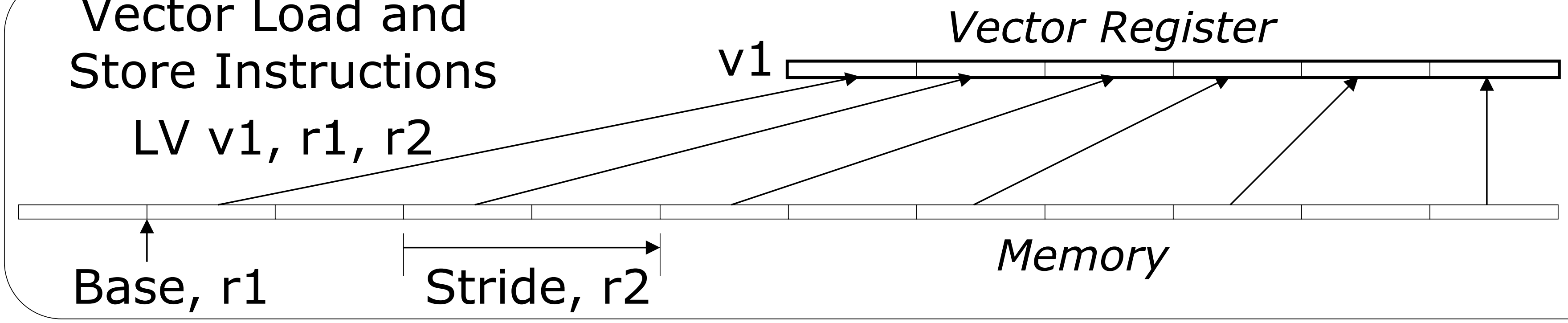
Vector Arithmetic Instructions

ADDV v3, v1, v2



Vector Load and Store Instructions

LV v1, r1, r2



Vector Code Example

```
# C code
for (i=0; i<64; i++)
  C[i] = A[i] + B[i];
```

```
# Scalar Code
  LI R4, 64
loop:
  L.D F0, 0(R1)
  L.D F2, 0(R2)
  ADD.D F4, F2, F0
  S.D F4, 0(R3)
  DADDIU R1, 8
  DADDIU R2, 8
  DADDIU R3, 8
  DSUBIU R4, 1
  BNEZ R4, loop
```

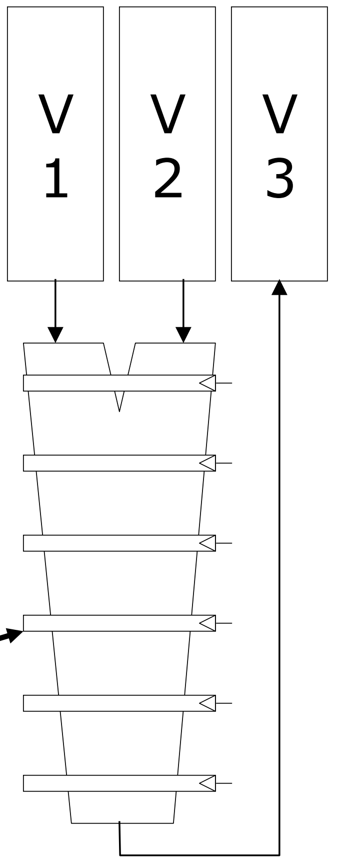
```
# Vector Code
  LI VLR, 64
  LV V1, R1
  LV V2, R2
  ADDV.D V3, V1, V2
  SV V3, R3
```

Vector Instruction Set Advantages

- Compact
  - one short instruction encodes N operations
- Expressive, tells hardware that these N operations:
  - are independent
  - use the same functional unit
  - access disjoint registers
  - access registers in same pattern as previous instructions
  - access a contiguous block of memory (unit-stride load/store)
  - access memory in a known pattern (strided load/store)
- Scalable
  - can run same code on more parallel pipelines (*lanes*)

Vector Arithmetic Execution

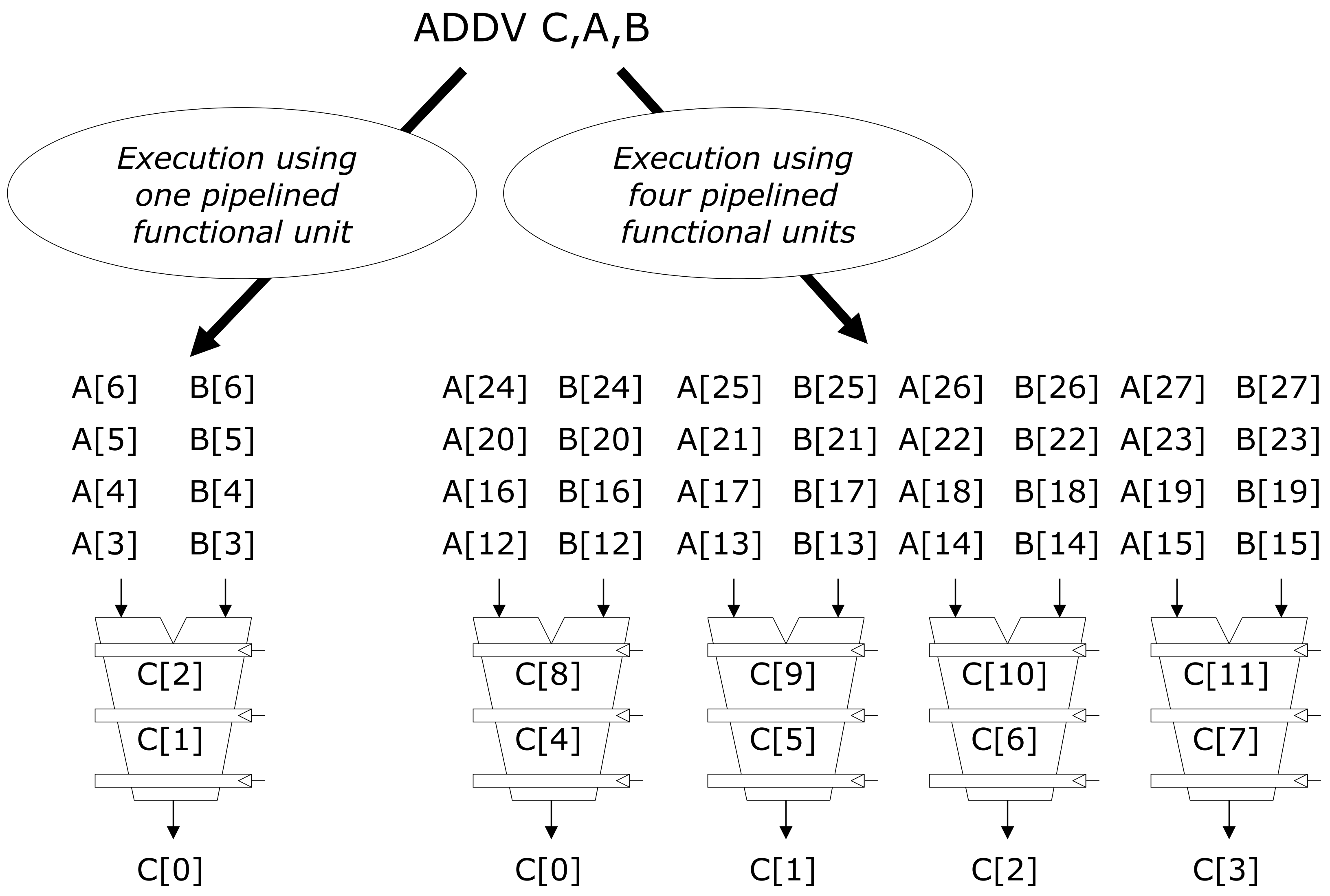
- Use deep pipeline (=> fast clock) to execute element operations
- Simplifies control of deep pipeline because elements in vector are independent (=> no hazards!)



Six stage multiply pipeline

V3 <- v1 * v2

Vector Instruction Execution

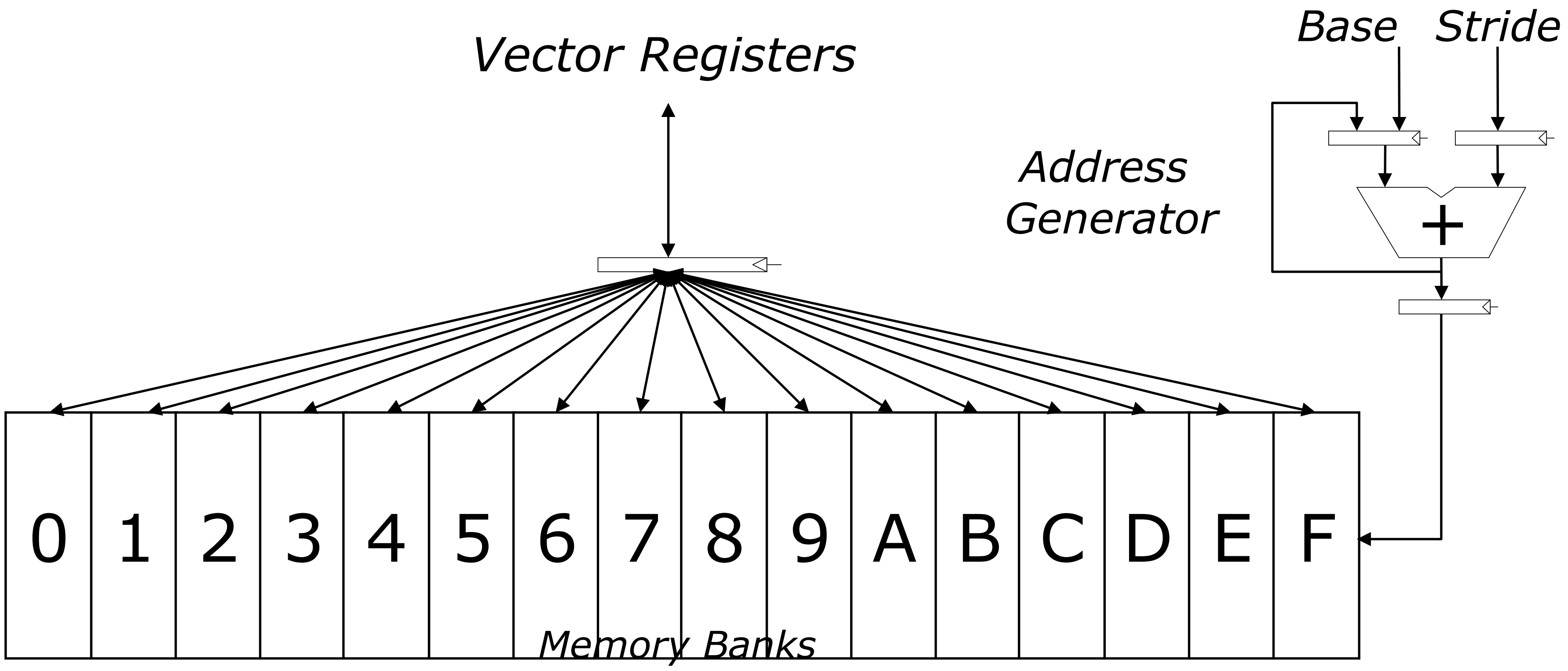


ADDV C,A,B
Execution using one pipelined functional unit
Execution using four pipelined functional units
A[6] B[6]
A[5] B[5]
A[4] B[4]
A[3] B[3]
C[2]
C[1]
C[0]
A[24] B[24] A[25] B[25] A[26] B[26] A[27] B[27]
A[20] B[20] A[21] B[21] A[22] B[22] A[23] B[23]
A[16] B[16] A[17] B[17] A[18] B[18] A[19] B[19]
A[12] B[12] A[13] B[13] A[14] B[14] A[15] B[15]
C[8] C[9] C[10] C[11]
C[4] C[5] C[6] C[7]
C[0] C[1] C[2] C[3]

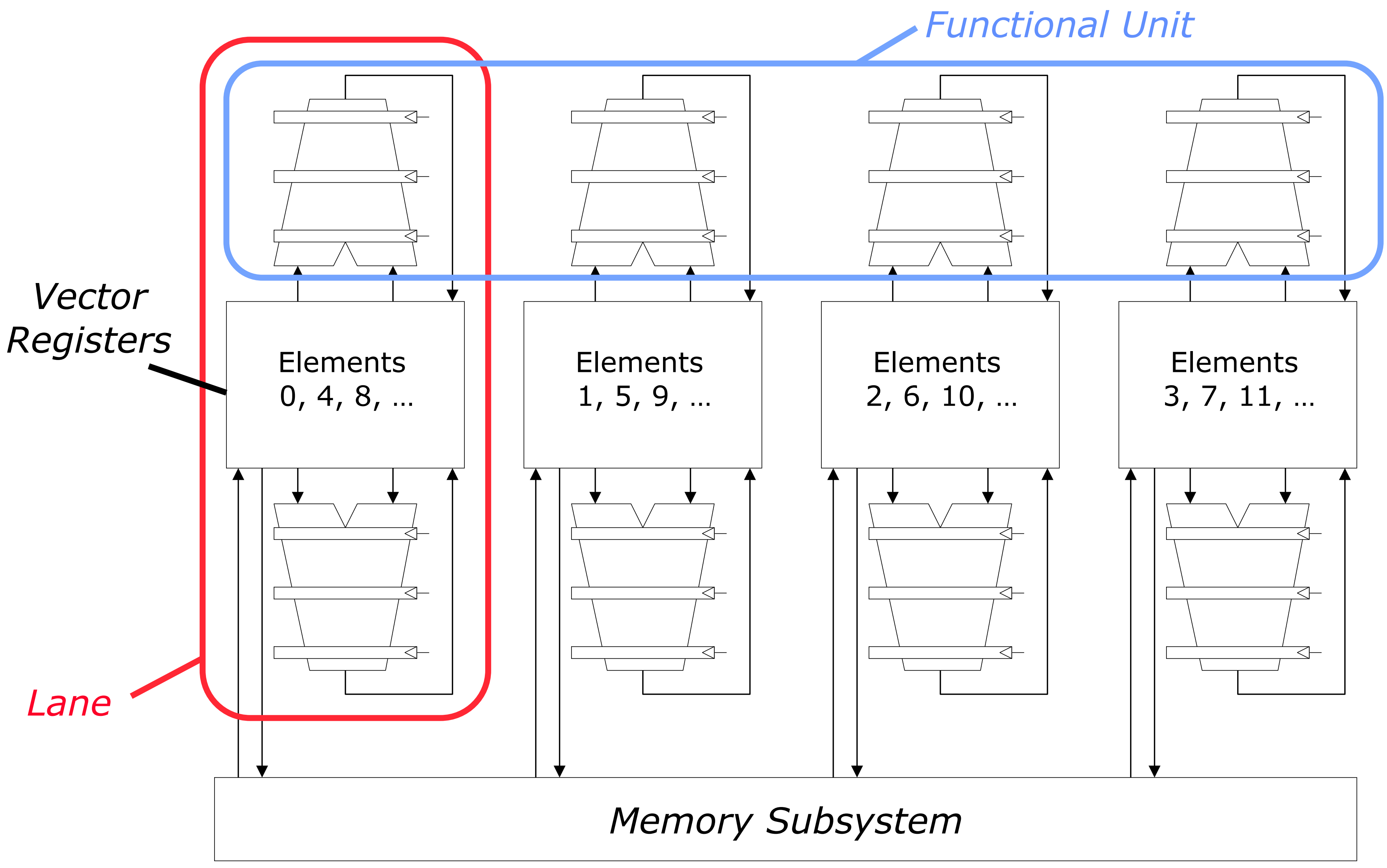
Vector Memory System

Cray-1, 16 banks, 4 cycle bank busy time, 12 cycle latency

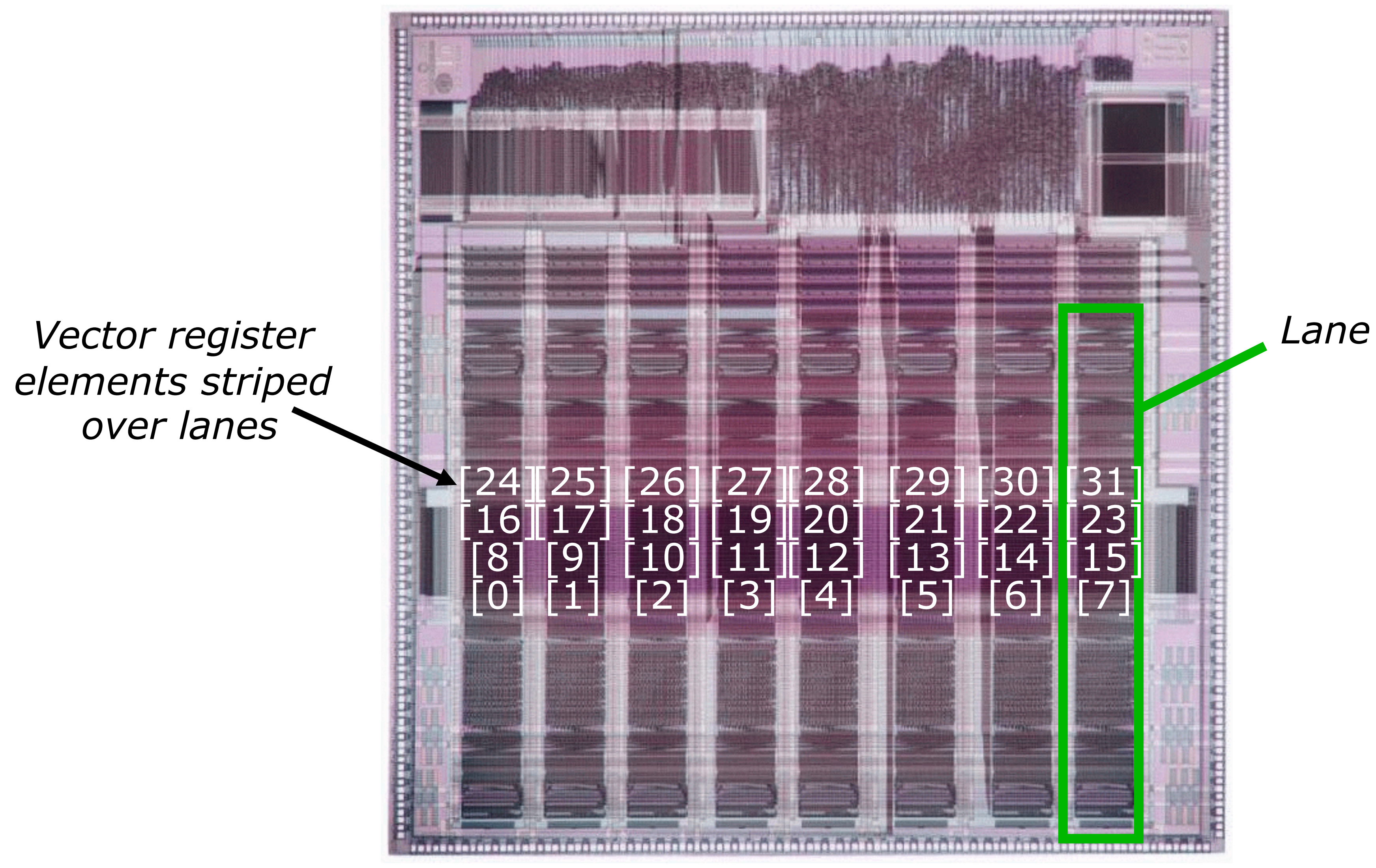
- *Bank busy time*: Time before bank ready to accept next request



Vector Unit Structure



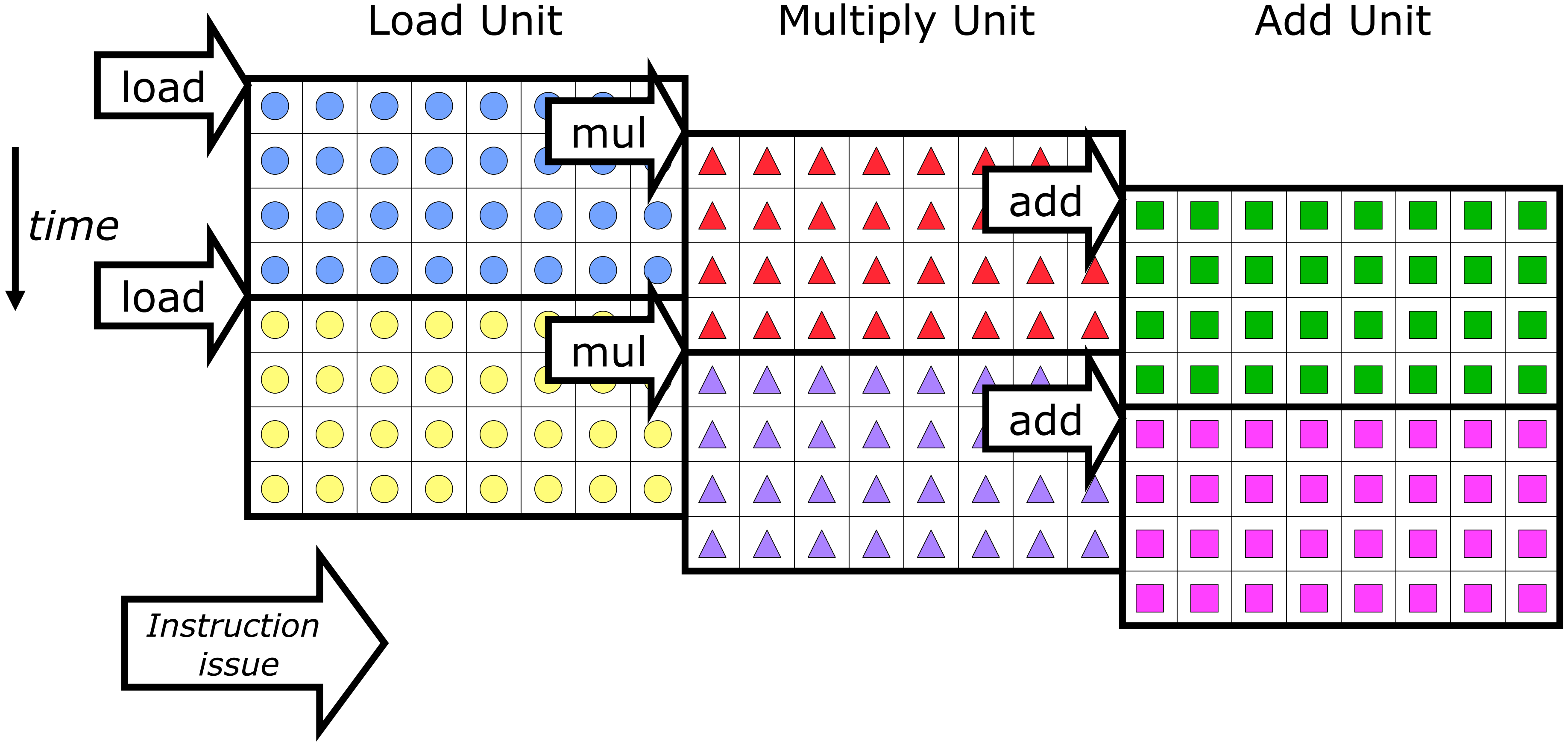
T0 Vector Microprocessor (UCB/ICSI, 1995)



Vector Instruction Parallelism

Can overlap execution of multiple vector instructions

- example machine has 32 elements per vector register and 8 lanes



Complete 24 operations/cycle while issuing 1 short instruction/cycle

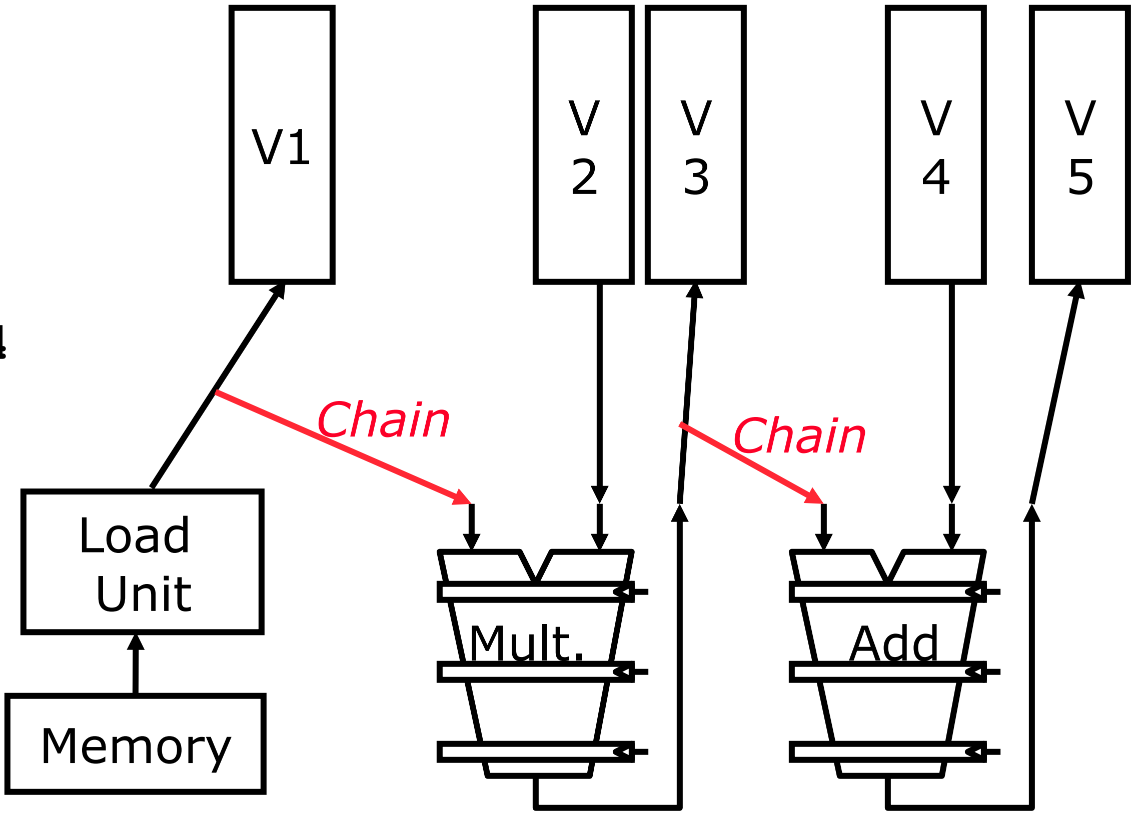
CS152 Administrivia

- Quiz 5, **Thursday** April 23

Vector Chaining

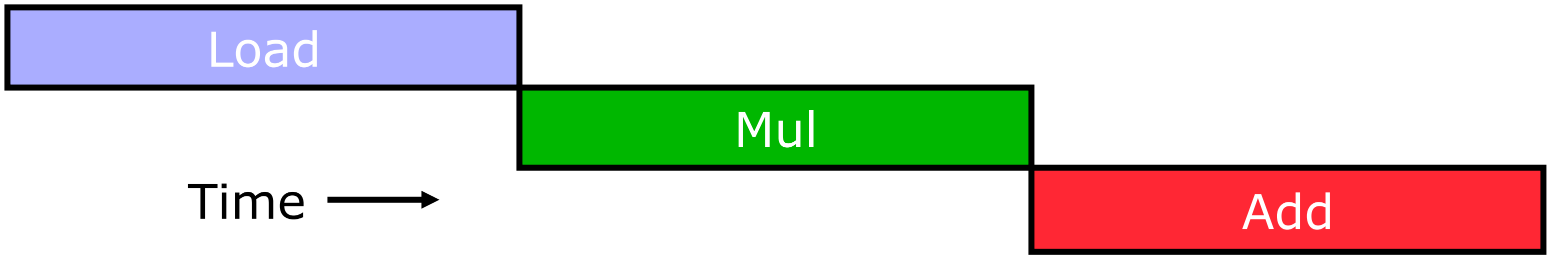
- Vector version of register bypassing
  - introduced with Cray-1

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

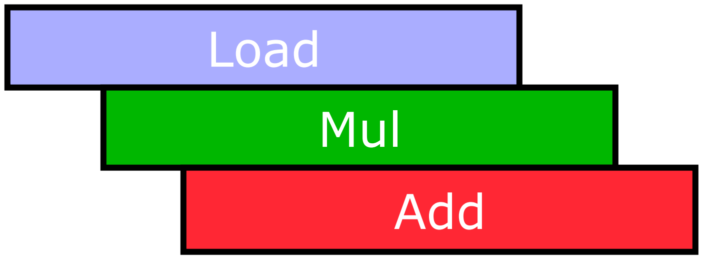


Vector Chaining Advantage

- Without chaining, must wait for last element of result to be written before starting dependent instruction



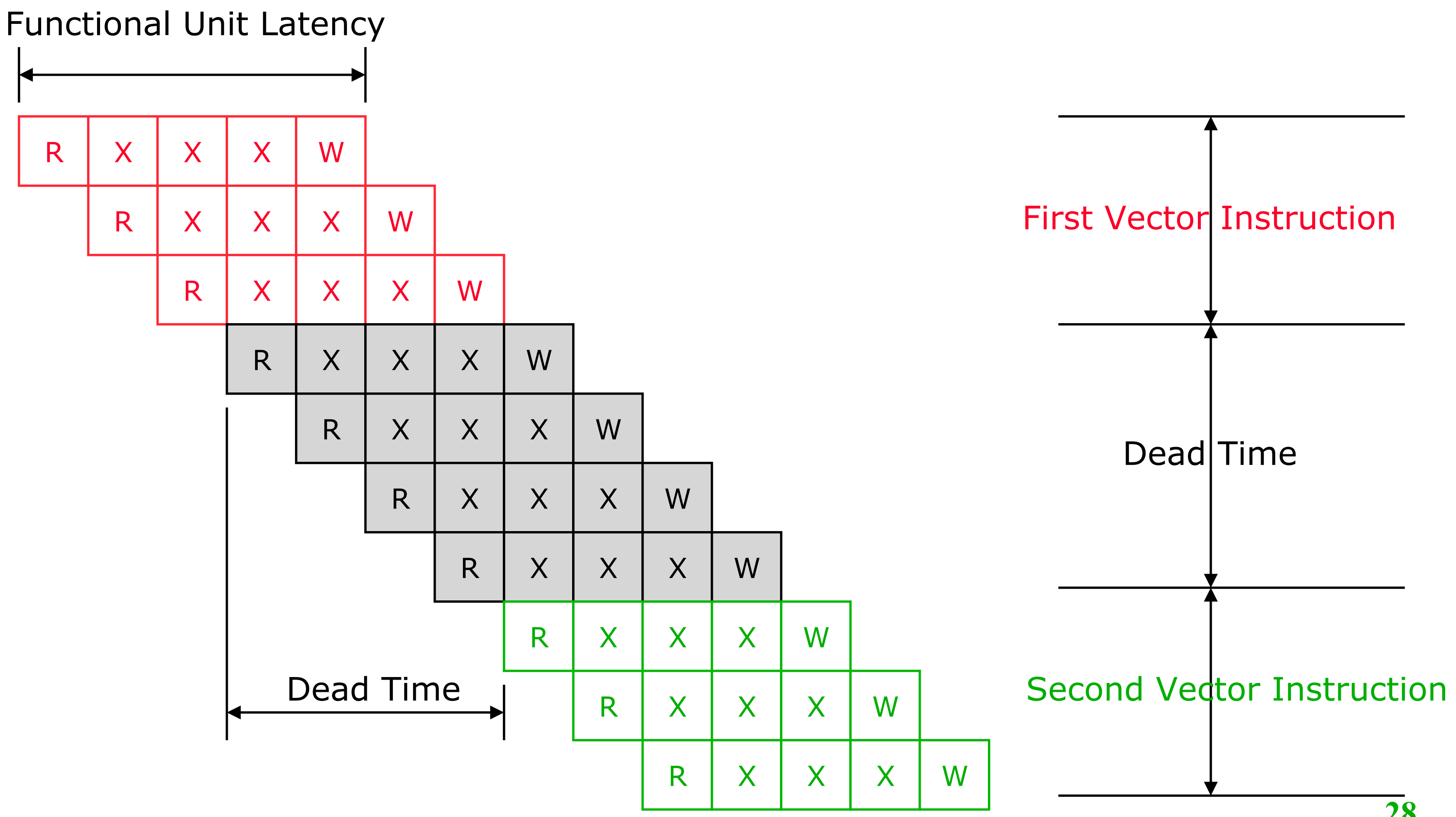
- With chaining, can start dependent instruction as soon as first result appears



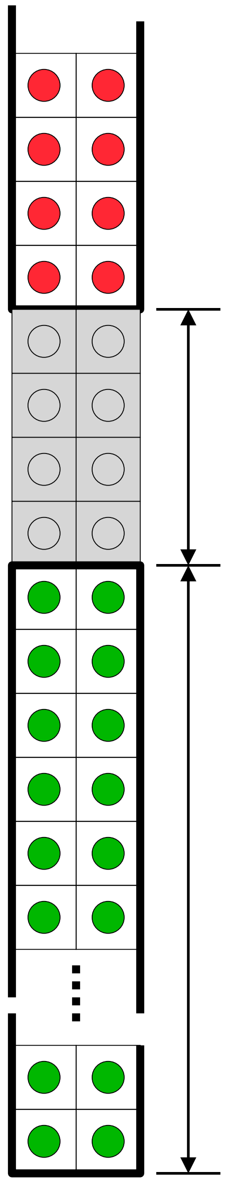
Vector Startup

Two components of vector startup penalty

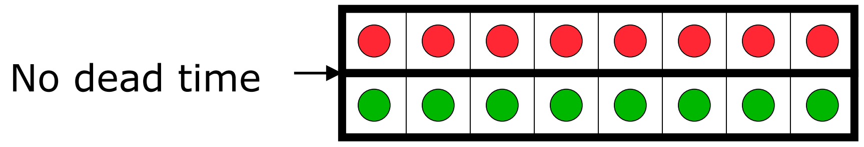
- functional unit latency (time through pipeline)
- dead time or recovery time (time before another vector instruction can start down pipeline)



Dead Time and Short Vectors



Cray C90, Two lanes
4 cycle dead time
Maximum efficiency 94% with 128 element vectors



T0, Eight lanes
No dead time
100% efficiency with 8 element vectors

Vector Memory-Memory versus Vector Register Machines

- Vector memory-memory instructions hold all vector operands in main memory
- The first vector machines, CDC Star-100 (‘73) and TI ASC (‘71), were memory-memory machines
- Cray-1 (’76) was first vector register machine

Example Source Code

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

Vector Memory-Memory Code

```
ADDV C, A, B
SUBV D, A, B
```

Vector Register Code

```
LV V1, A
LV V2, B
ADDV V3, V1, V2
SV V3, C
SUBV V4, V1, V2
SV V4, D
```

Vector Memory-Memory vs. Vector Register Machines

- Vector memory-memory architectures (VMMA) require greater main memory bandwidth, why?
  - All operands must be read in and out of memory
- VMMAs make if difficult to overlap execution of multiple vector operations, why?
  - Must check dependencies on memory addresses
- VMMAs incur greater startup latency
  - Scalar code was faster on CDC Star-100 for vectors < 100 elements
  - For Cray-1, vector/scalar breakeven point was around 2 elements

⇒ *Apart from CDC follow-ons (Cyber-205, ETA-10) all major vector machines since Cray-1 have had vector register architectures*

(we ignore vector memory-memory from now on)

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)

- MIT material derived from course 6.823
- UCB material derived from course CS252