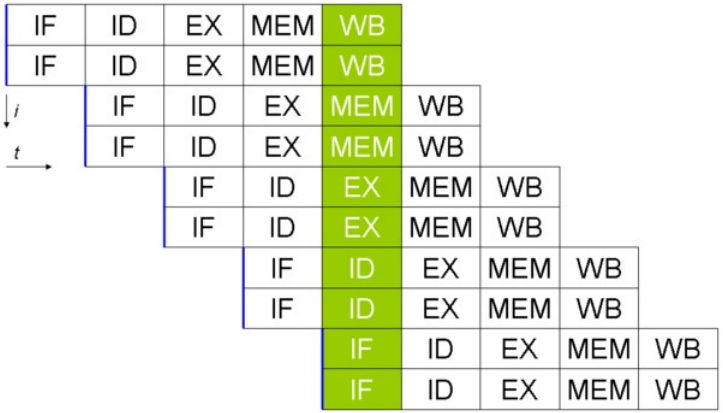
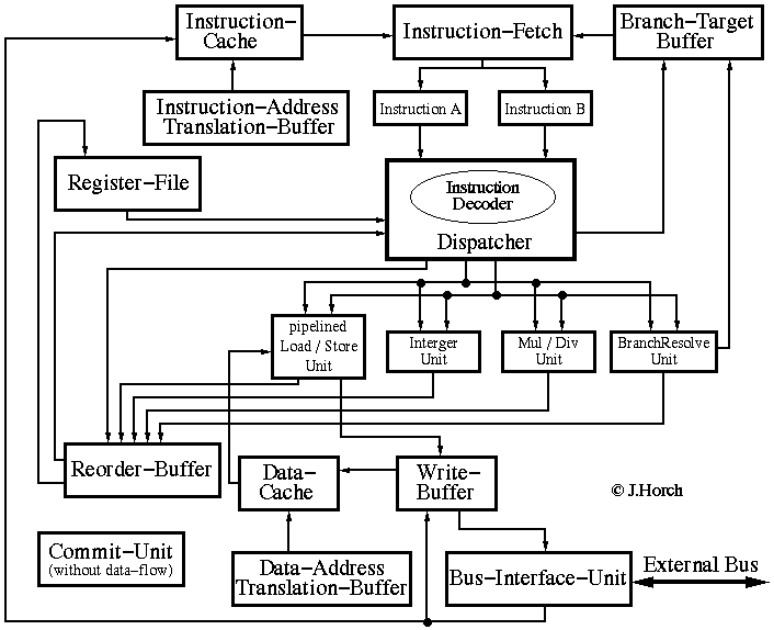
History of Parallel Architectures

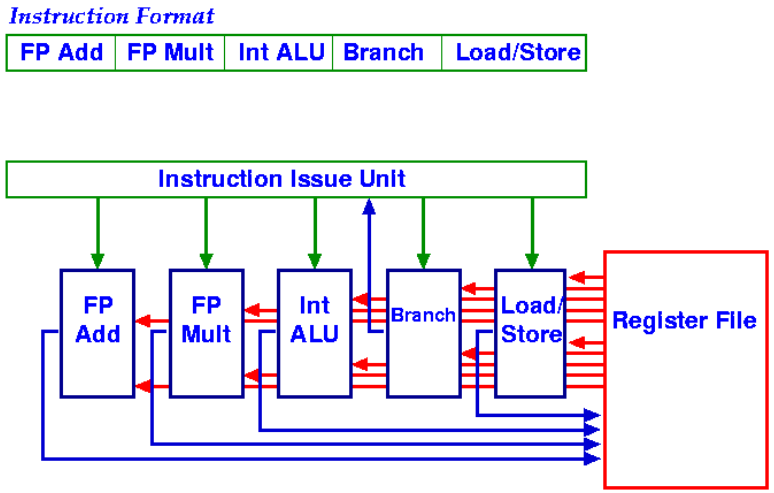
Instruction Level Parallelism



Superscalar



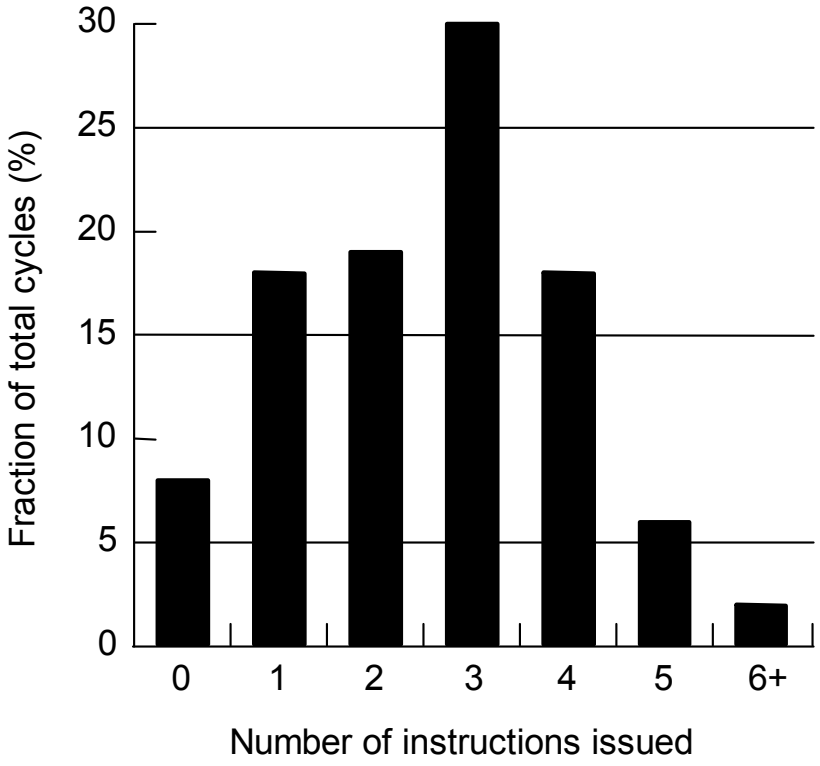
VLIW

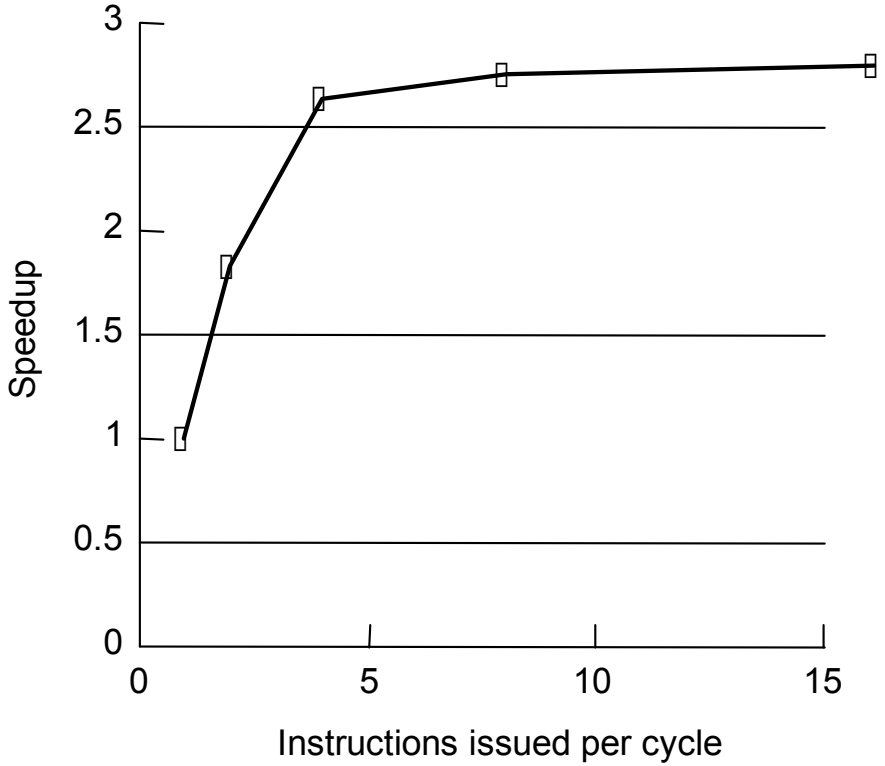


Architectural Trends: ILP

- Reported speedups for superscalar processors
  - Horst, Harris, and Jardine [1990] 1.37
  - Wang and Wu [1988] 1.70
  - Smith, Johnson, and Horowitz [1989] 2.30
  - Murakami et al. [1989] 2.55
  - Chang et al. [1991] 2.90
  - Jouppi and Wall [1989] 3.20
  - Lee, Kwok, and Briggs [1991] 3.50
  - Wall [1991] 5
  - Melvin and Patt [1991] 8
  - Butler et al. [1991] ..... 17+
- Large variance due to difference in
  - application domain investigated (numerical versus non-numerical)
  - capabilities of processor modeled

ILP Ideal Potential

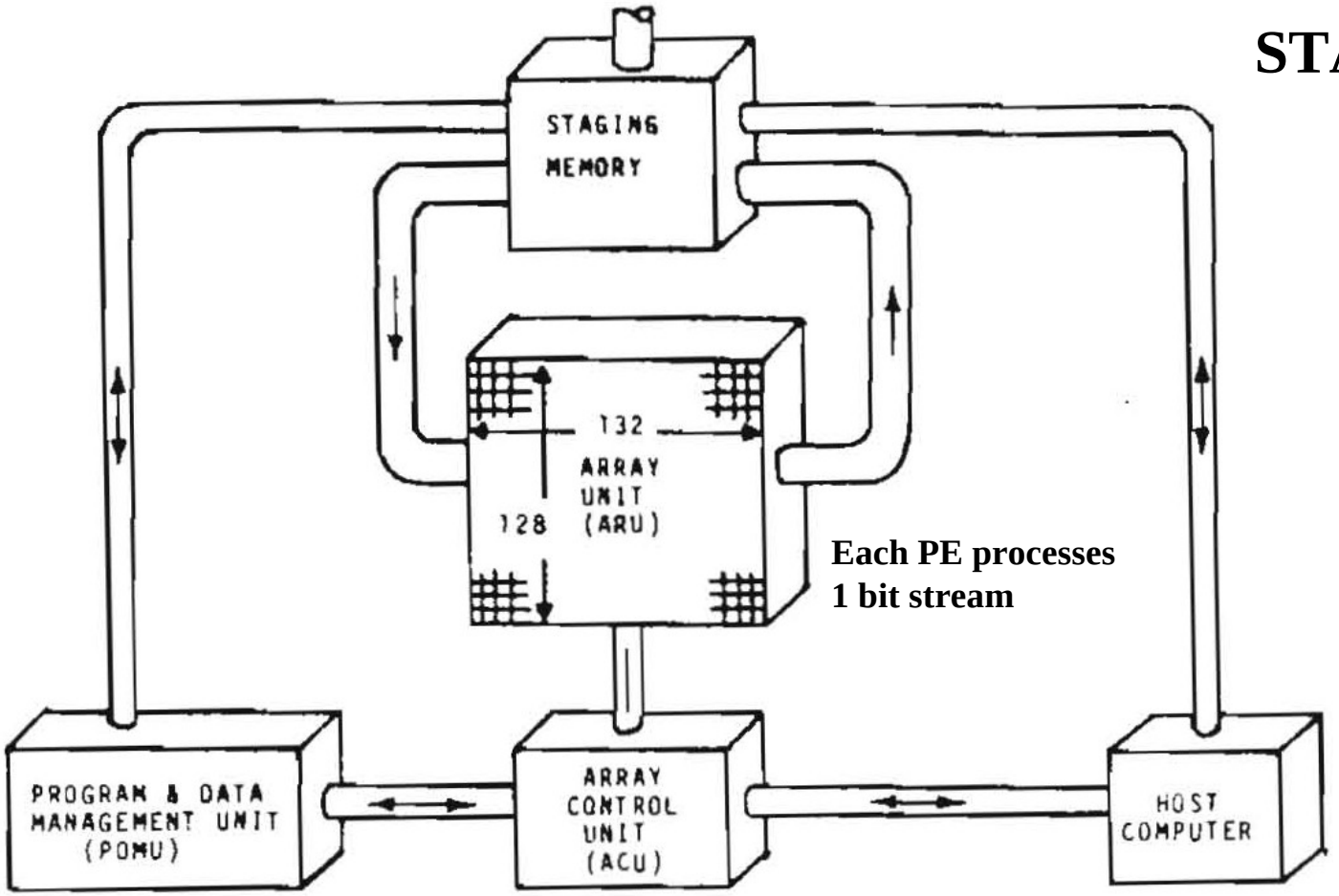




- Infinite resources and fetch bandwidth, perfect branch prediction and renaming
  - real caches and non-zero miss latencies

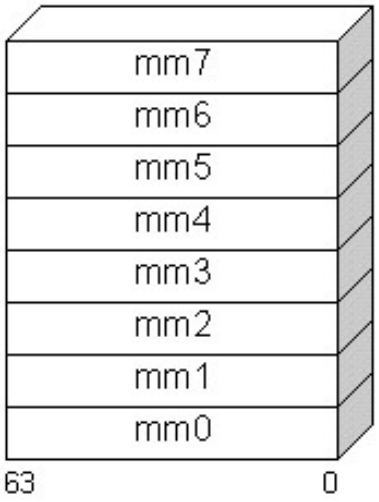
Bit Serial Processing Systems (1970s)

STARAN

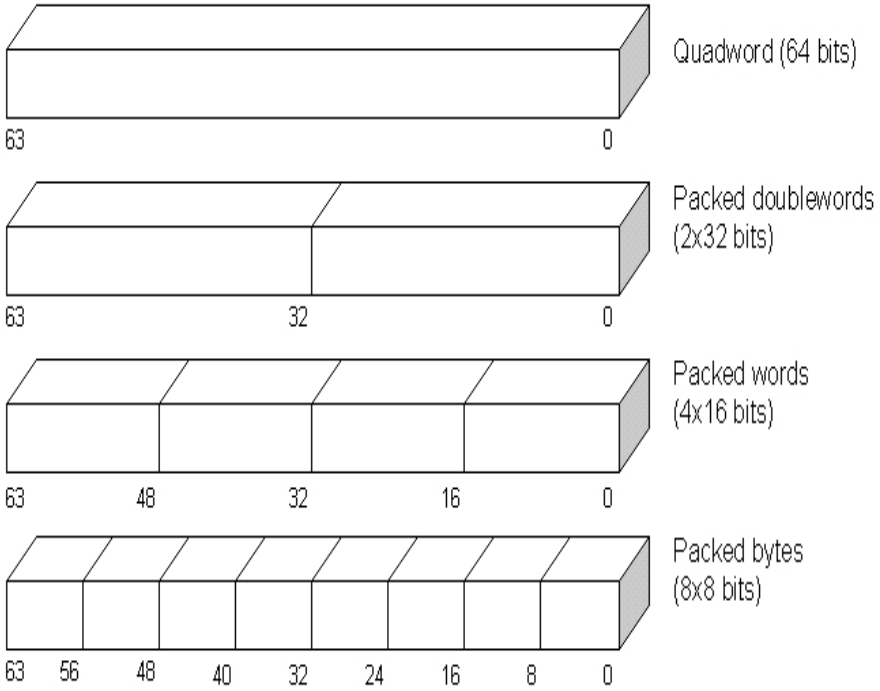


Groups of Bits (e.g. Intel MMX)

MMX Registers



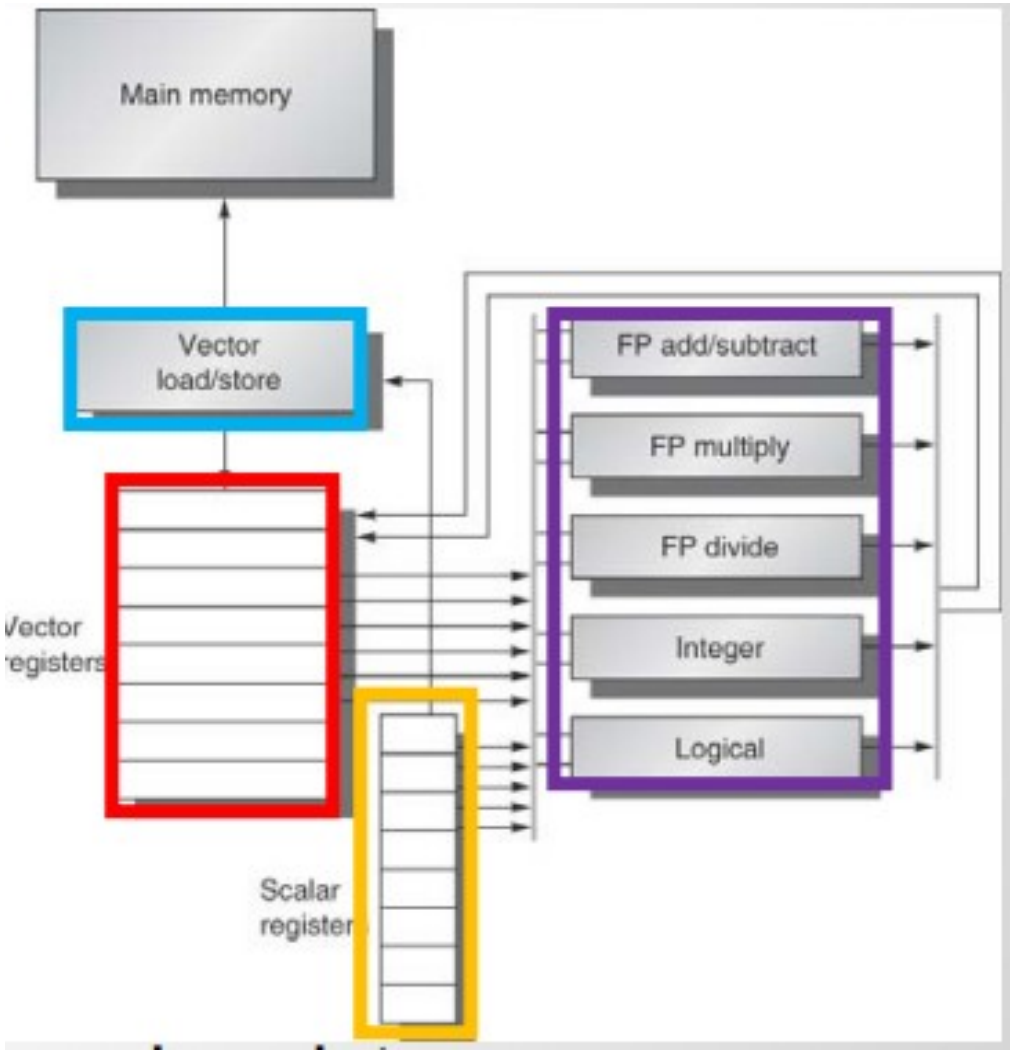
Interpretable as



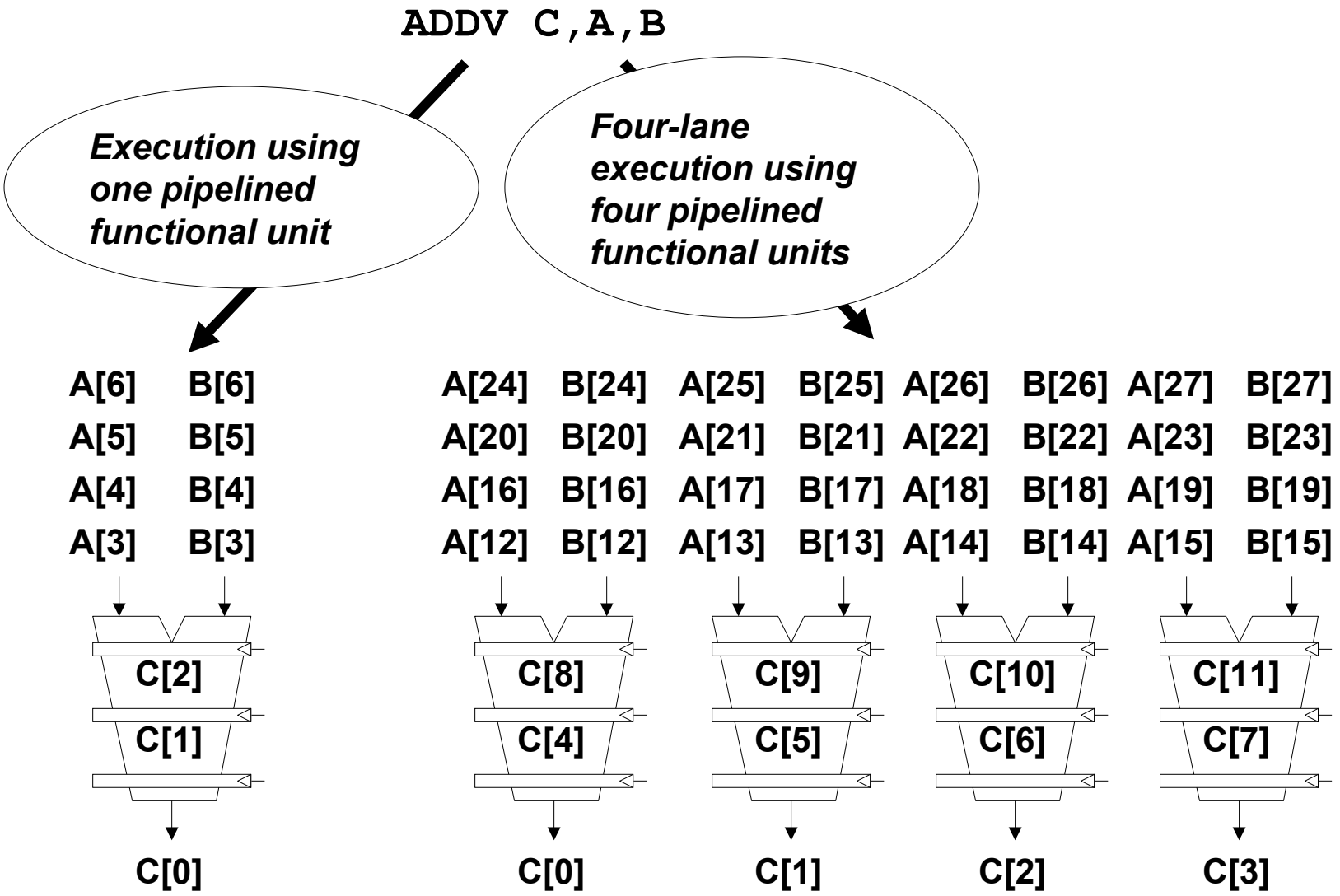
OP <src>, <dest>

<src> OP <dest> ⟹ **<dest>**

Vector Machines (e.g. Cray)



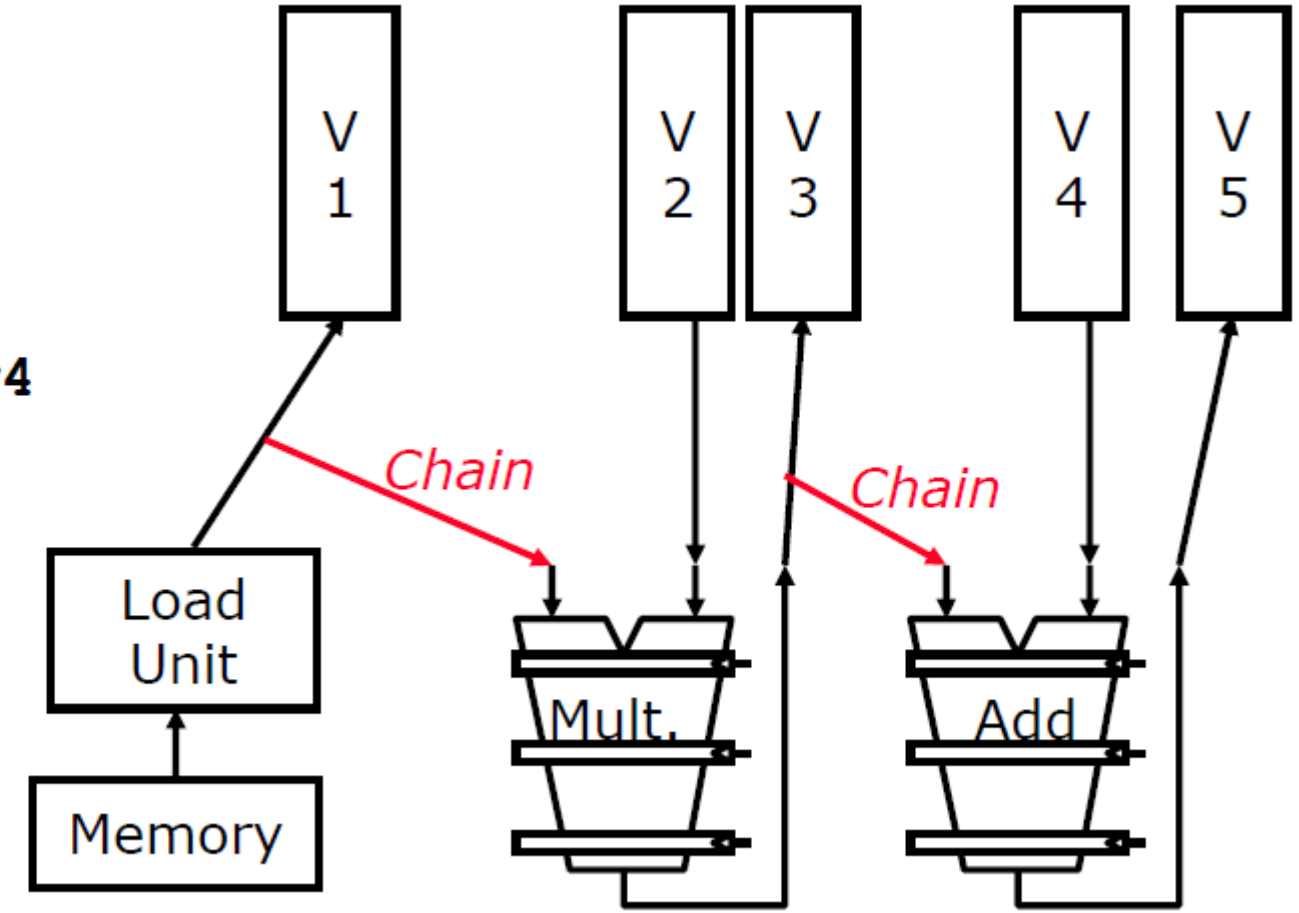
Multiple Lanes



Coming at a cost of High Memory Bandwidth Needs!

Vector Chaining (Cray-1)

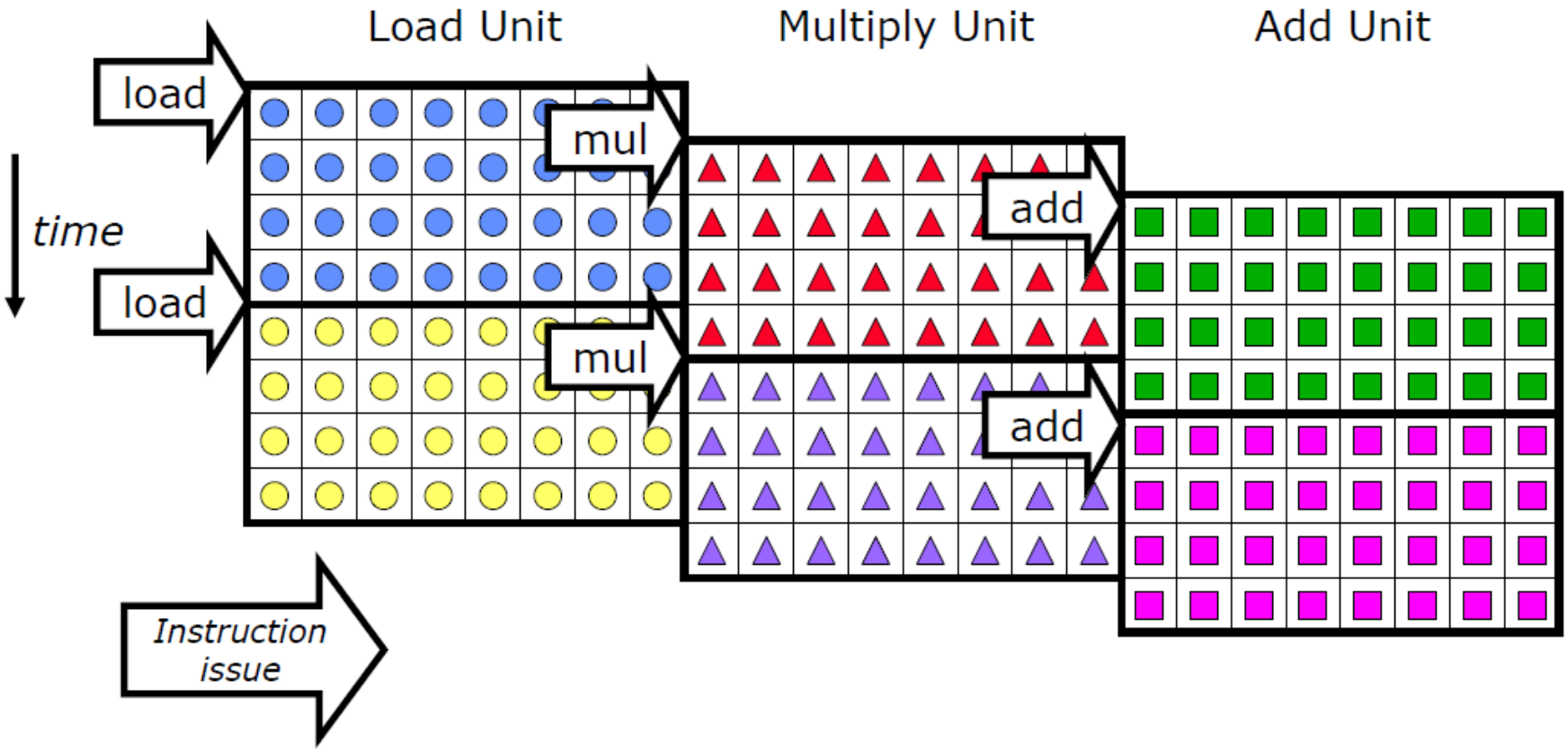
```
LV    v1
MULV  v3,v1,v2
ADDV  v5,  v3,  v4
```



Vector Instruction Parallelism

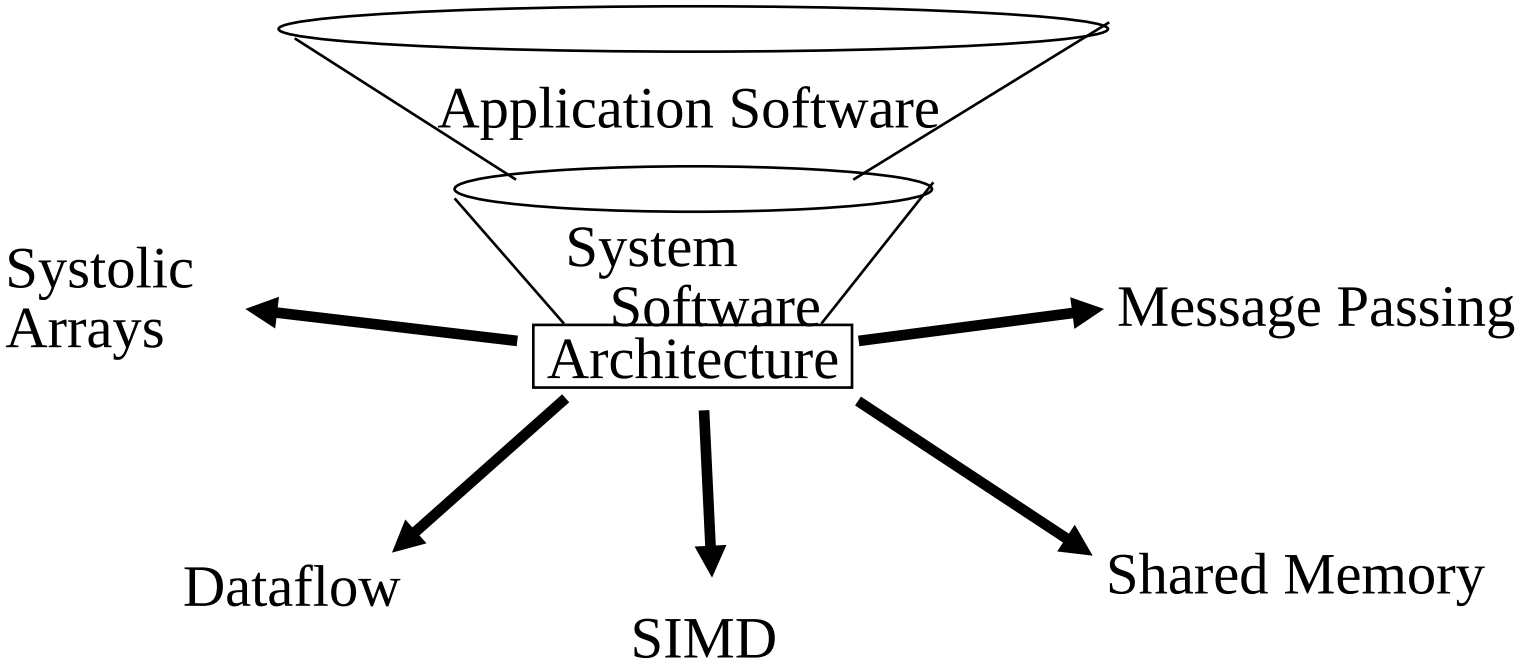
Can overlap execution of multiple vector instructions

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



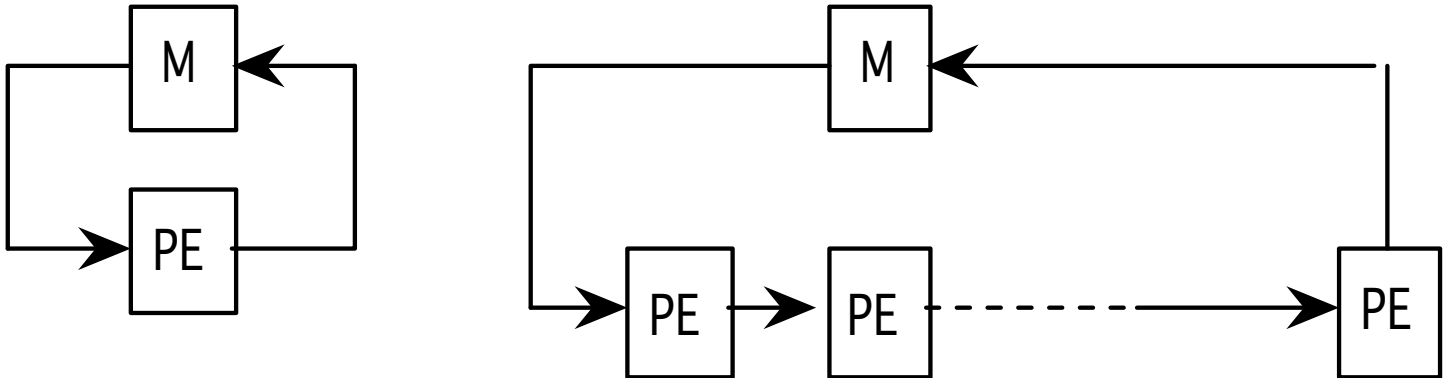
Complete 24 operations/cycle while issuing 1 short instruction/cycle

True Multiprocessors



Systolic Architectures

- Replace single processor with array of regular processing elements
- Orchestrate data flow for high throughput with less memory access



Different from pipelining

- Nonlinear array structure, multidirection data flow, each PE may have (small) local instruction and data memory

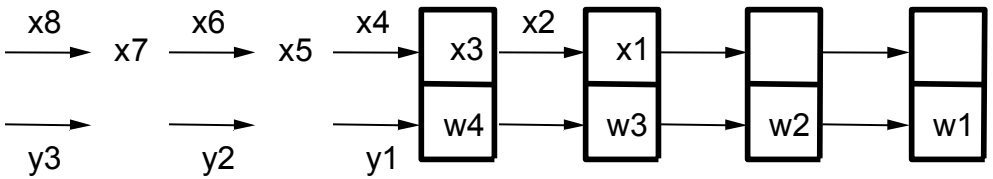
Initial motivation: VLSI enables inexpensive special-purpose chips

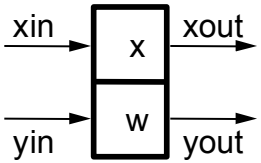
Represent algorithms directly by chips connected in regular pattern

Systolic Arrays (contd.)

Example: Systolic array for 1-D convolution

$$y(i) = w1 \times x(i) + w2 \times x(i+1) + w3 \times x(i+2) + w4 \times x(i+3)$$





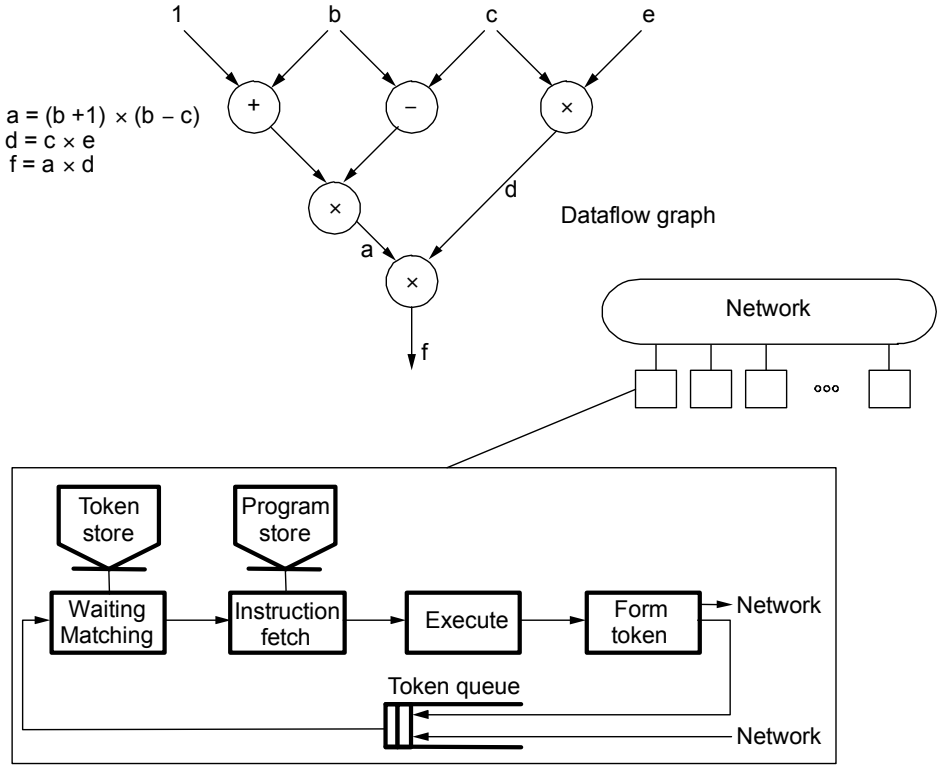
xout = x
x = x in
yout = yin + w × xin

- Practical realizations (e.g. iWARP) use quite general processors
  - Enable variety of algorithms on same hardware
- But dedicated interconnect channels
  - Data transfer directly from register to register across channel
- Specialized, and same problems as SIMD
  - General purpose systems work well for same algorithms (locality etc.)

Dataflow Architectures

Represent computation as a graph of essential dependences

- Logical processor at each node, activated by availability of operands
- Message (tokens) carrying tag of next instruction sent to next processor
- Tag compared with others in matching store; match fires execution



Flynn's Taxonomy

Single Instruction Single Data (SISD)

Single Instruction Multiple Data (SIMD)

Multiple Instruction Single Data (MISD)

Multiple Instruction Multiple Data (MIMD)

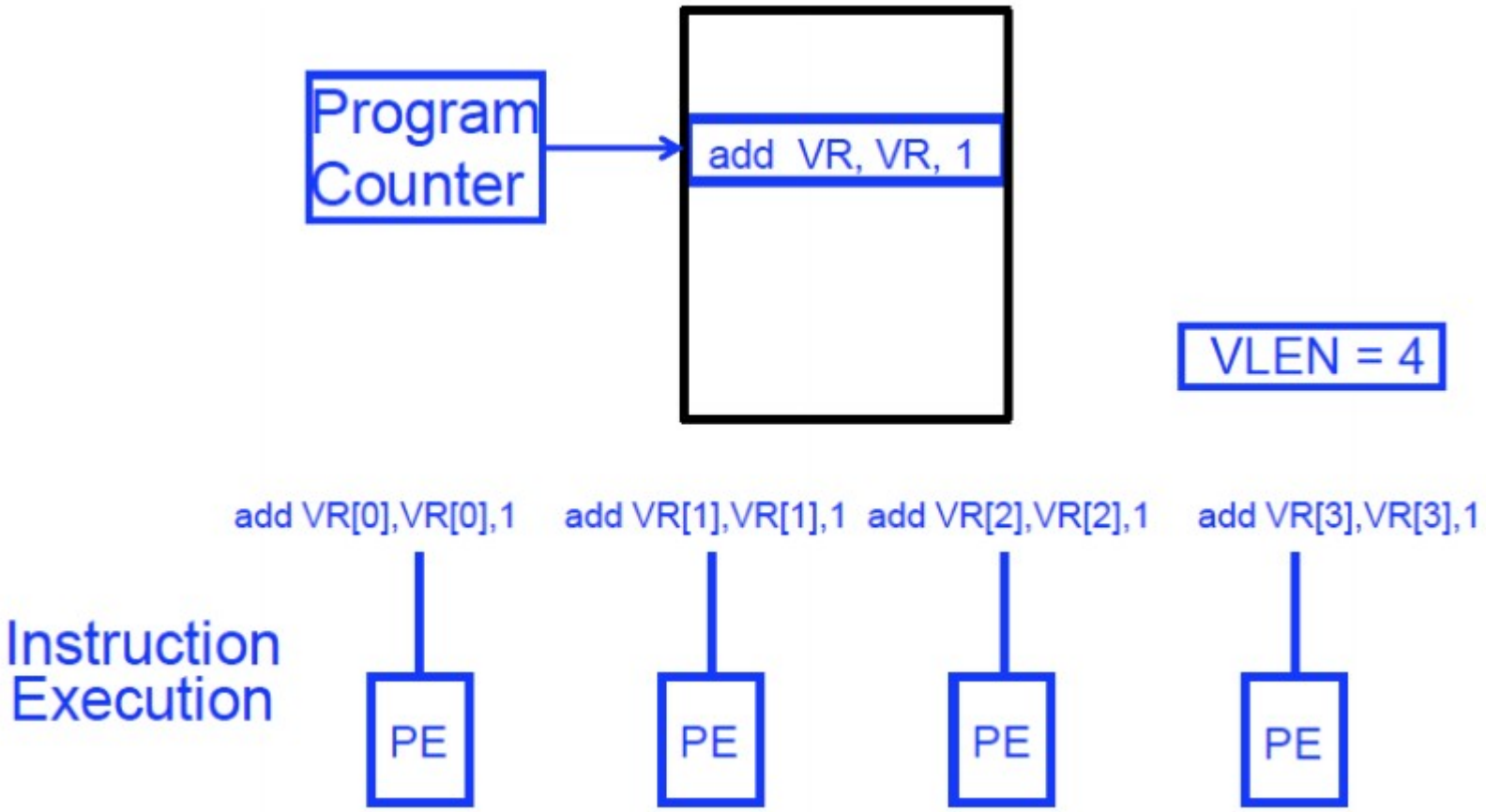
From a practical viewpoint,

SIMD (single PC)

MIMD (multiple PCs)

More commonly used is SPMD (Single Program Multiple Data – single program with multiple PCs).

SIMD (e.g. MasPar, CM-2)



Problems with SIMD

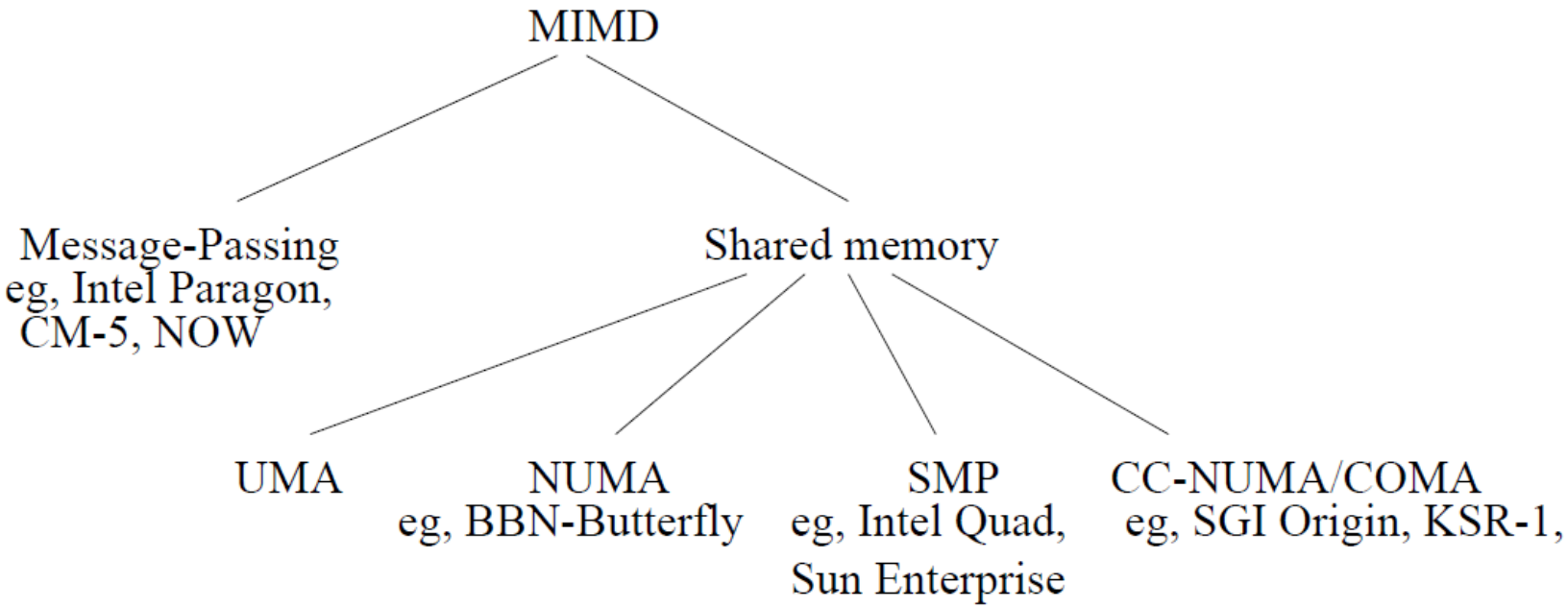
Everyone has to do the same instruction

Everyone has to complete current instruction before proceeding to next.

Costly to broadcast same instruction to thousands of processors.

MIMD

No broadcast, each can execute its own instruction at its own pace!



Programming Model

What programmer uses in coding applications

Specifies communication and synchronization

Examples:

- *Shared address space*: like bulletin board
- *Message passing*: like letters or phone calls, explicit point to point
- *Data parallel*: more regimented, global actions on data
  - Implemented with shared address space or message passing

Note that the model can be directly support in (i) Hardware, or (ii) OS or (iii) User libraries

Shared Address Space Architectures

Any processor can directly reference any memory location

- Communication occurs implicitly as result of loads and stores

Convenient:

- Location transparency
- Similar programming model to time-sharing on uniprocessors
  - Except processes run on different processors
  - Good throughput on multiprogrammed workloads

Naturally provided on wide range of platforms

- History dates at least to precursors of mainframes in early 60s
- Wide range of scale: few to hundreds of processors

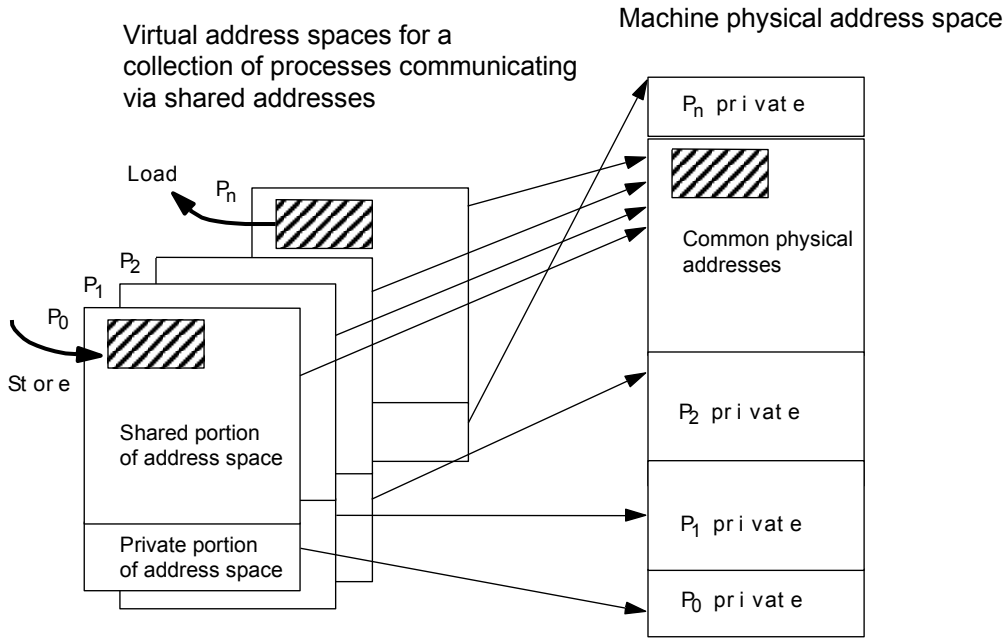
Popularly known as *shared memory* machines or model

- Ambiguous: memory may be physically distributed among processors

Shared Address Space Model

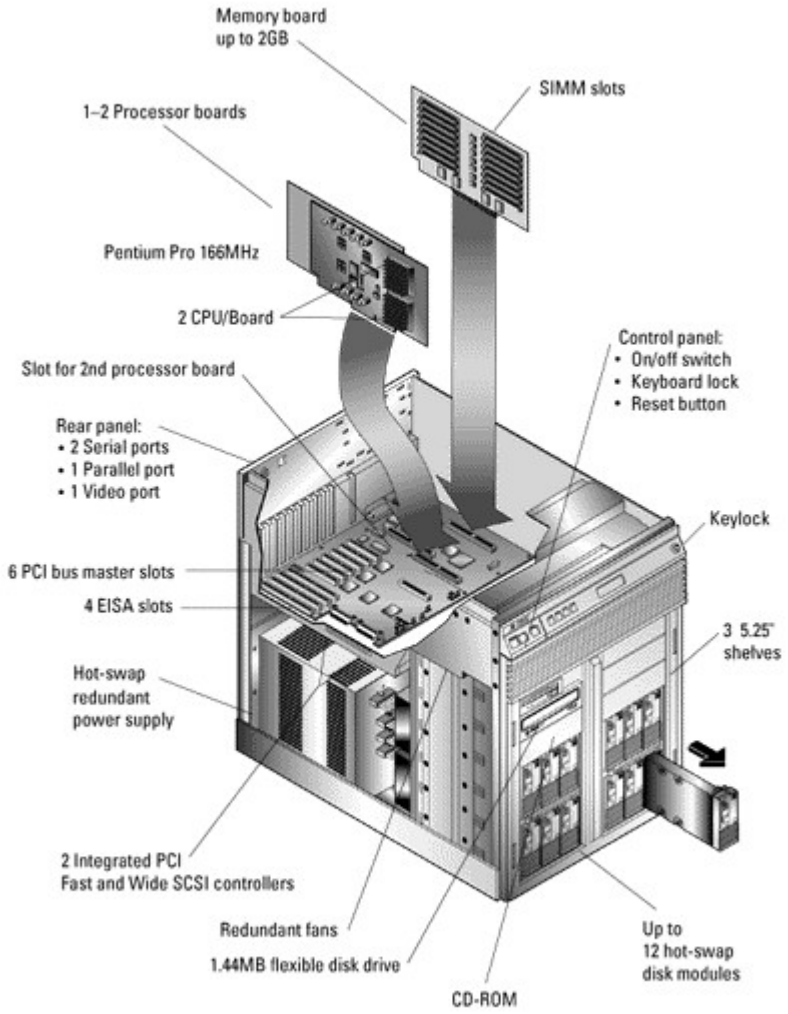
Process: virtual address space plus one or more threads of control

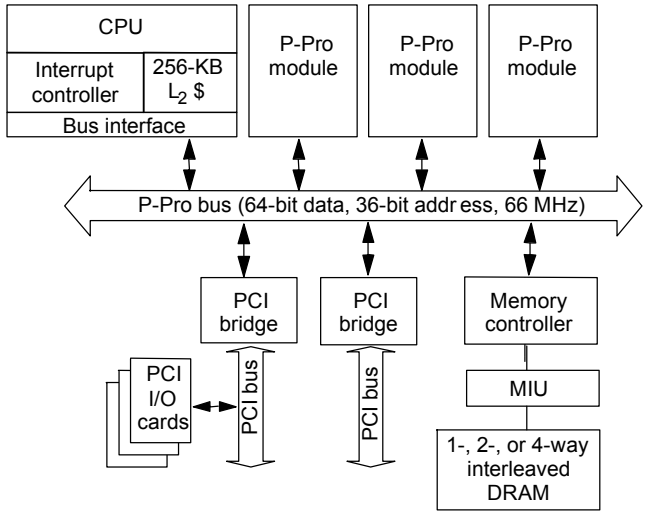
Portions of address spaces of processes are shared



- Writes to shared address visible to other threads (in other processes too)
- Natural extension of uniprocessors model: conventional memory operations for comm.; special atomic operations for synchronization
- OS uses shared memory to coordinate processes

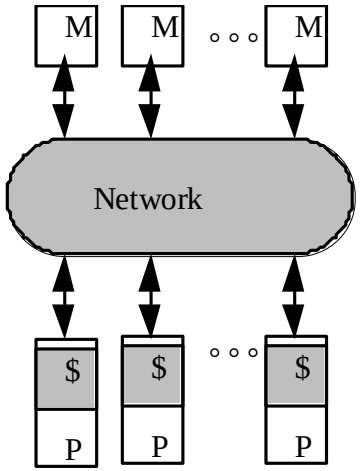
Example: Intel Pentium Pro Quad (UMA)



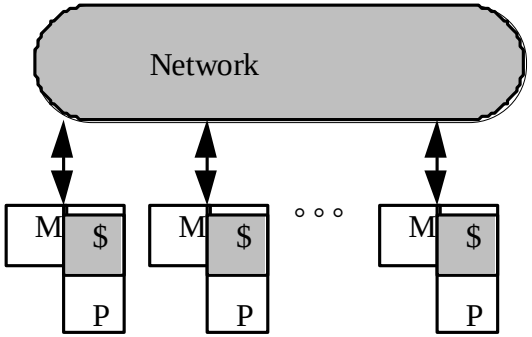


- All coherence and multiprocessing glue in processor module
- Highly integrated, targeted at high volume
- Low latency and bandwidth

Scaling Up



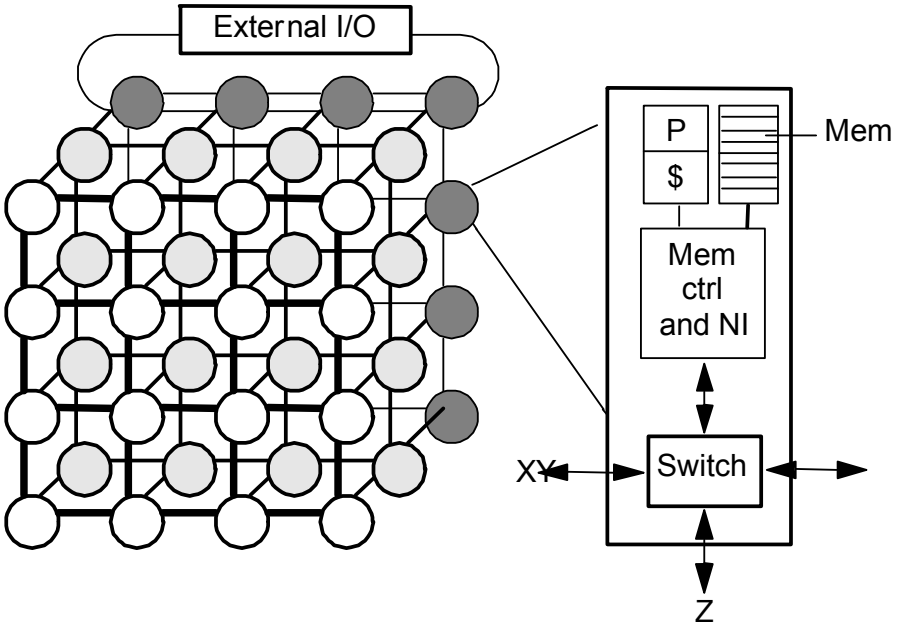
"Dance hall" (Still UMA)



Distributed "Shared "memory (NUMA)

- Problem is interconnect: cost (crossbar) or bandwidth (bus)
- Dance-hall: bandwidth still scalable, but lower cost than crossbar
  - latencies to memory uniform, but uniformly large (e.g. BBN Butterfly)
- Distributed memory or non-uniform memory access (NUMA)
  - Construct shared address space out of simple message transactions across a general-purpose network (e.g. read-request, read-response)
- Caching shared (particularly nonlocal) data? (CC-NUMA)

Example: Cray T3E (NUMA)



- Scale up to 1024 processors, 480MB/s links
- Memory controller generates comm. request for nonlocal references
- No hardware mechanism for coherence (SGI Origin etc. provide this)

Message Passing Architectures

Complete computer as building block, including I/O

- Communication via explicit I/O operations

Programming model: directly access only private address space (local memory), comm. via explicit messages (send/receive)

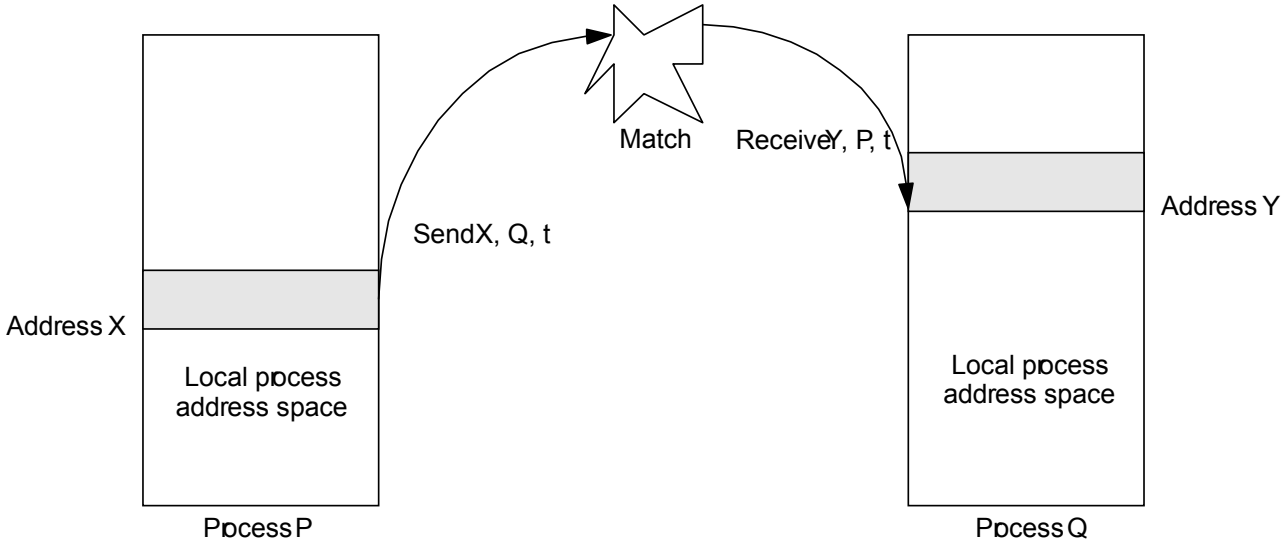
High-level block diagram similar to distributed-memory SAS

- But comm. integrated at IO level, needn't be into memory system
- Like networks of workstations (clusters), but tighter integration
- Easier to build than scalable SAS

Programming model more removed from basic hardware operations

- Library or OS intervention

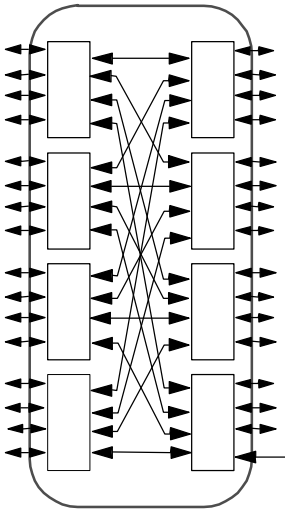
Message-Passing Abstraction

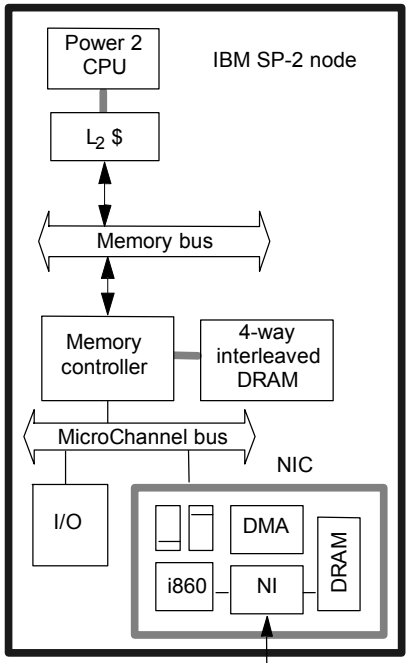


- Send specifies buffer to be transmitted and receiving process
- Recv specifies sending process and application storage to receive into
- Memory to memory copy, but need to name processes
- Optional tag on send and matching rule on receive
- User process names local data and entities in process/tag space too
- In simplest form, the send/recv match achieves pairwise synch event
  - Other variants too
- Many overheads: copying, buffer management, protection

Example: IBM SP-2

General interconnection network formed from 8-port switches

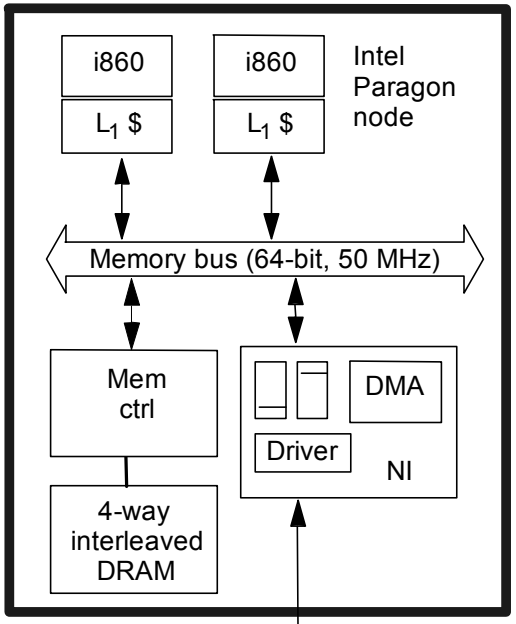




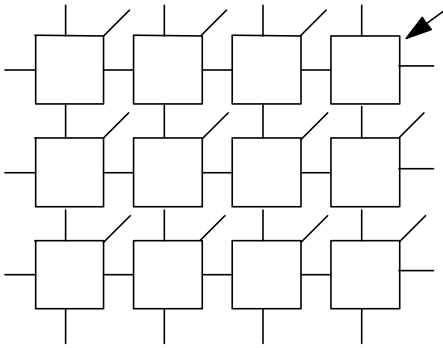
- Made out of essentially complete RS6000 workstations
- Network interface integrated in I/O bus (bw limited by I/O bus)

Example Intel Paragon

Sandia's Intel Paragon XP/S-based Super computer



8 bits,
175 MHz,
bidirectional



2D grid network
with processing node
attached to every switch