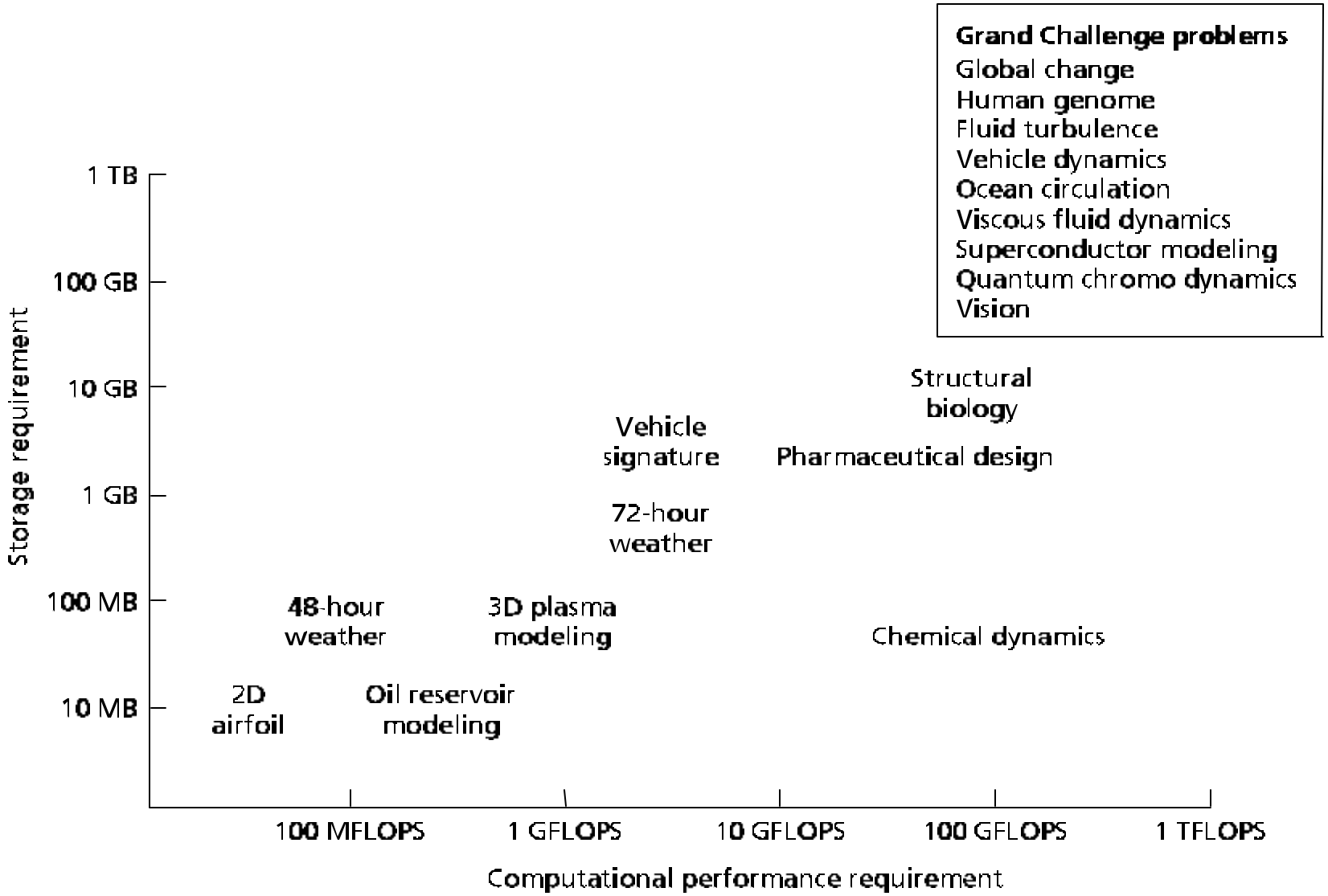
Parallel Computer Architecture

- **A parallel computer is a collection of processing elements that cooperate to solve large problems fast**
- **Broad issues involved:**
  - **Resource Allocation:**
    - **Number of processing elements (PEs).**
    - **Computing power of each element.**
    - **Amount of physical memory used.**
  - **Data access, Communication and Synchronization**
    - **How the elements cooperate and communicate.**
    - **How data is transmitted between processors.**
    - **Abstractions and primitives for cooperation.**
  - **Performance and Scalability**
    - **Performance enhancement of parallelism: Speedup.**
    - **Scalabilty of performance to larger systems/problems.**

The Need And Feasibility of Parallel Computing

- **Application demands: More computing cycles:**
  - ***Scientific computing*: CFD, Biology, Chemistry, Physics, ...**
  - ***General-purpose computing*: Video, Graphics, CAD, Databases, Transaction Processing, Gaming…**
  - **Mainstream multithreaded programs, are similar to parallel programs**
- **Technology Trends**
  - **Number of transistors on chip growing rapidly**
  - **Clock rates expected to go up but only slowly**
- **Architecture Trends**
  - **Instruction-level parallelism is valuable but limited**
  - **Coarser-level parallelism, as in MPs, the most viable approach**
- **Economics:**
  - **Today's microprocessors have multiprocessor support eliminating the need for designing expensive custom PEs**
  - **Lower parallel system cost.**
  - **Multiprocessor systems to offer a cost-effective replacement of uniprocessor systems in mainstream computing.**

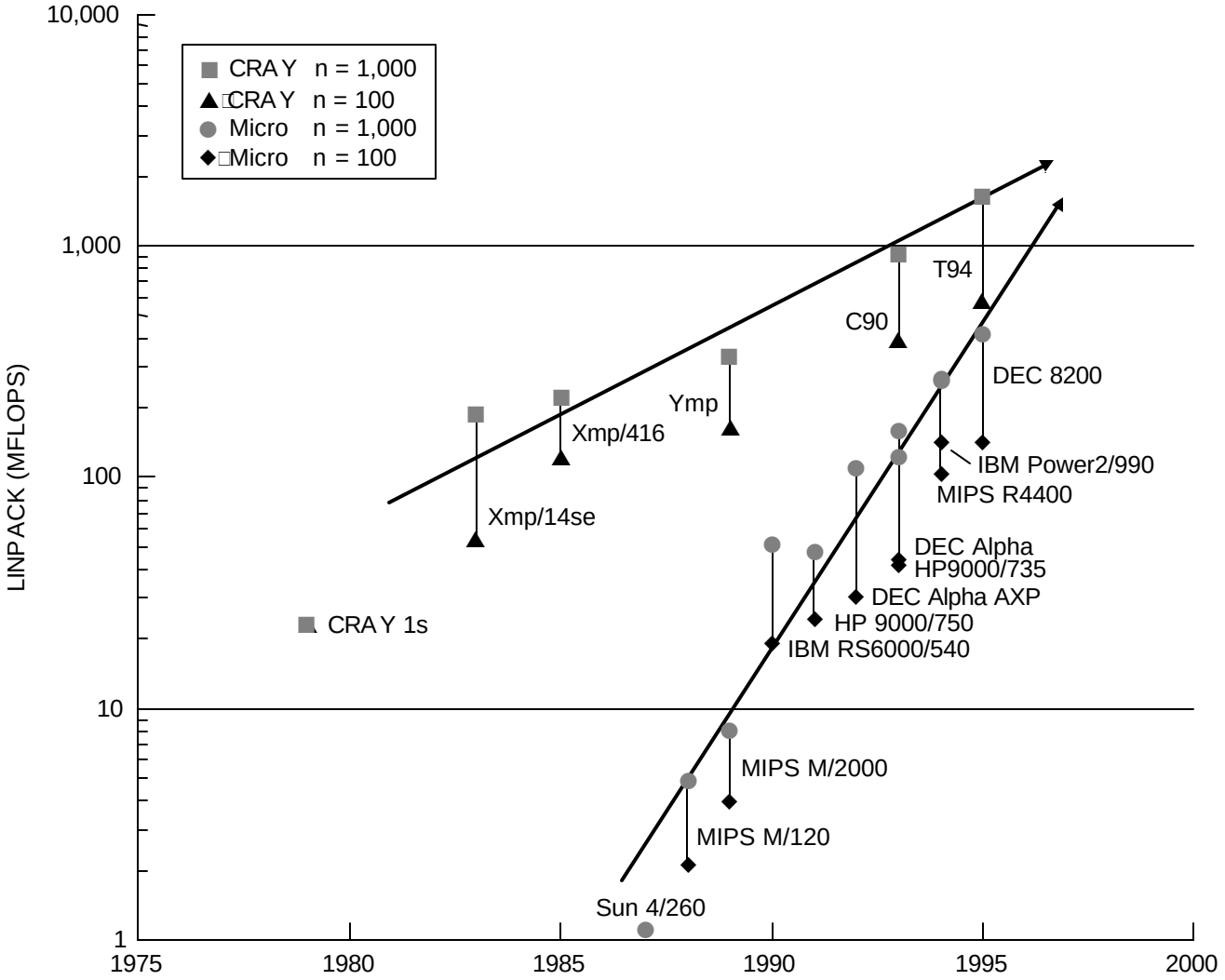
Scientific Computing Demand



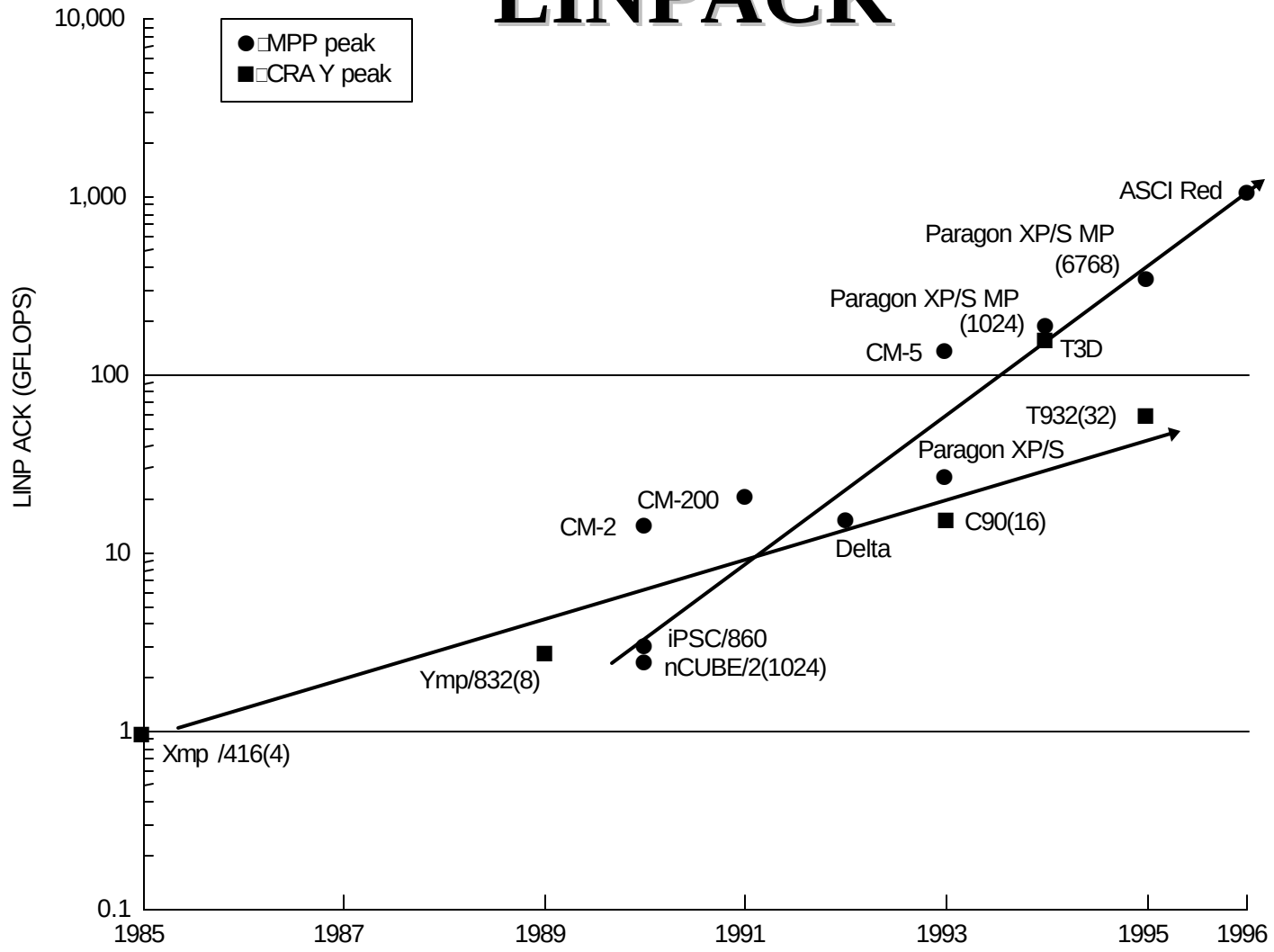
Scientific Supercomputing Trends

- **Proving ground and driver for innovative architecture and advanced techniques:**
  - **Market is much smaller relative to commercial segment**
  - **Dominated by vector machines starting in 70s**
  - **Meanwhile, microprocessors have made huge gains in floating-point performance**
    - **High clock rates.**
    - **Pipelined floating point units.**
    - **Instruction-level parallelism.**
    - **Effective use of caches.**
- ***Large-scale multiprocessors replace vector supercomputers***
  - **Well under way already**

Raw Uniprocessor Performance: LINPACK

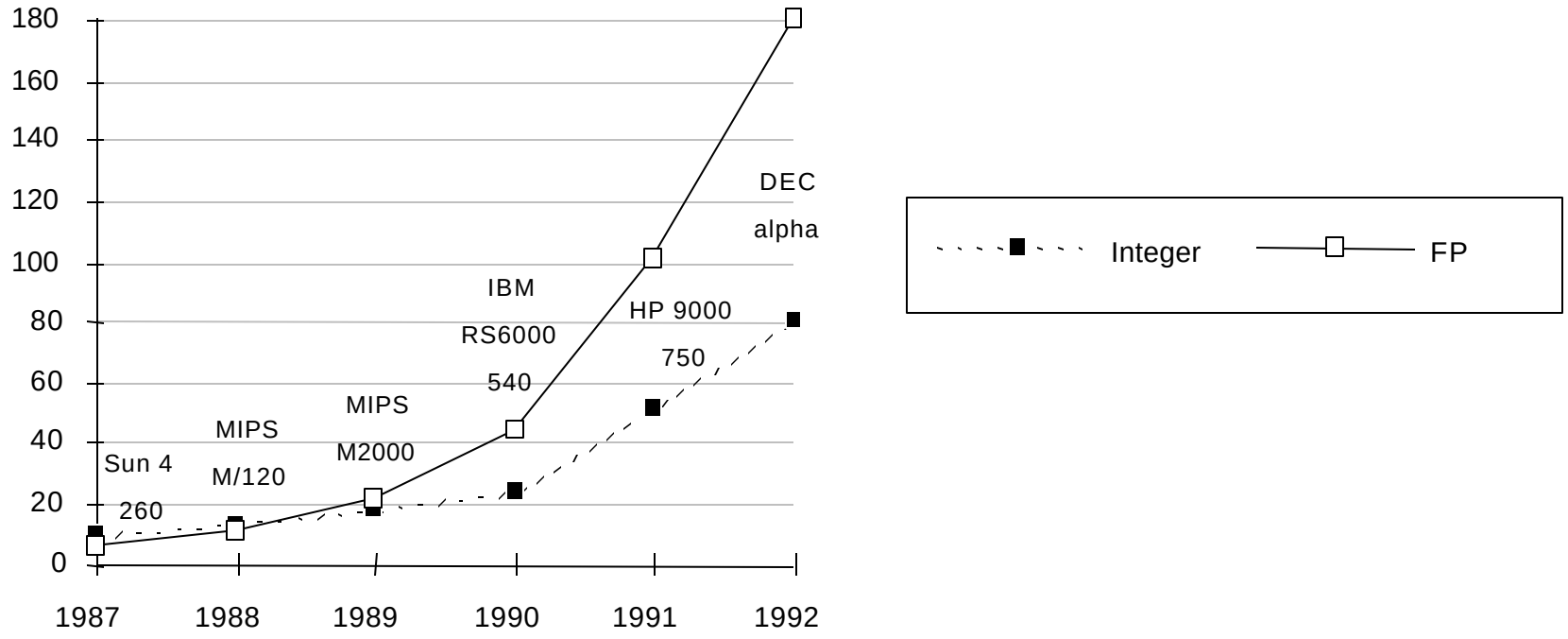


Raw Parallel Performance: LINPACK

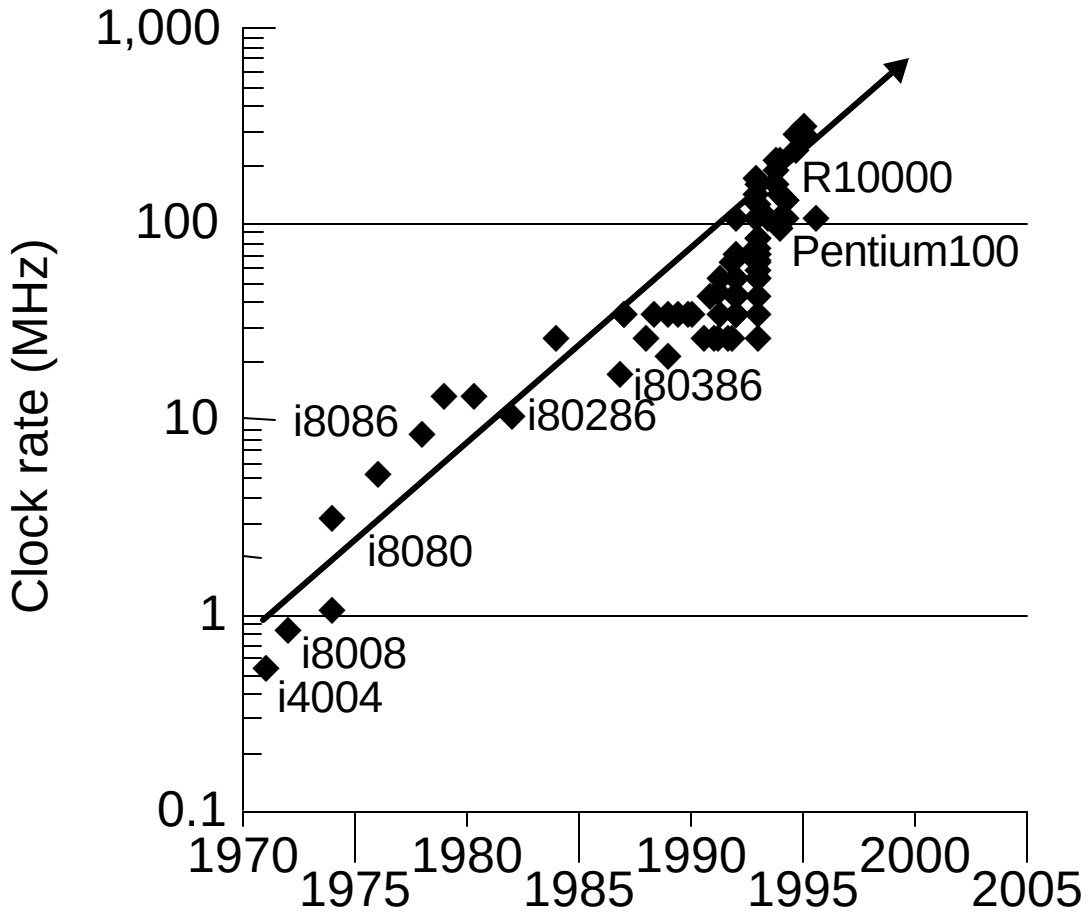


General Technology Trends

- ***Microprocessor performance*** **increases 50% - 100% per year**
- ***Transistor count*** **doubles every 3 years**
- ***DRAM size*** **quadruples every 3 years**

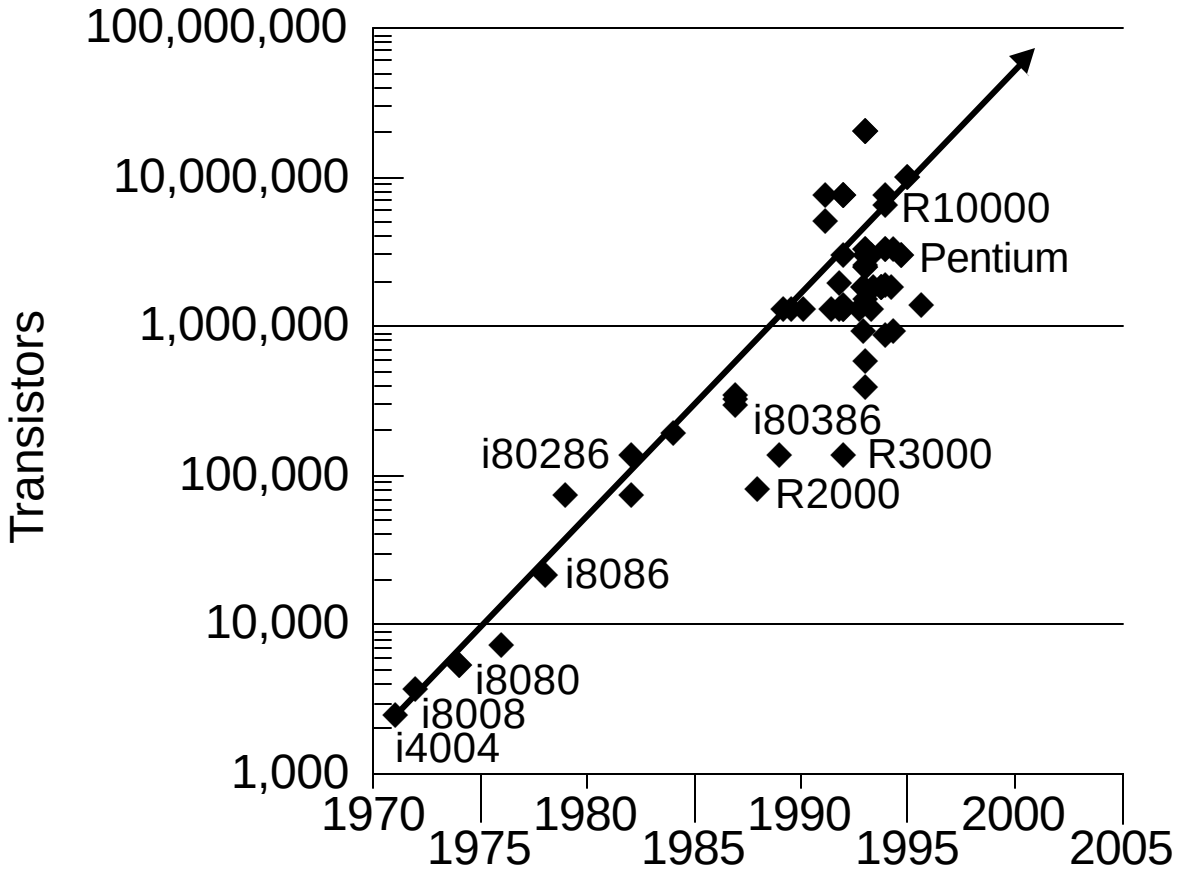


Clock Frequency Growth Rate



- Currently increasing 30% per year

Transistor Count Growth Rate



- 100 million transistors on chip by early 2000's A.D.
- Transistor count grows much faster than clock rate
  - Currently 40% per year

System Attributes to Performance

- Performance benchmarking is program-mix dependent.
- Ideal performance requires a perfect machine/program match.
- Performance measures:
  - Cycles per instruction (CPI)
  - Total CPU time = $T = C \times t = C / f = I_c \times CPI \times t$
    $= I_c \times (p + m \times k) \times t$

    I_c = Instruction count t = CPU cycle time

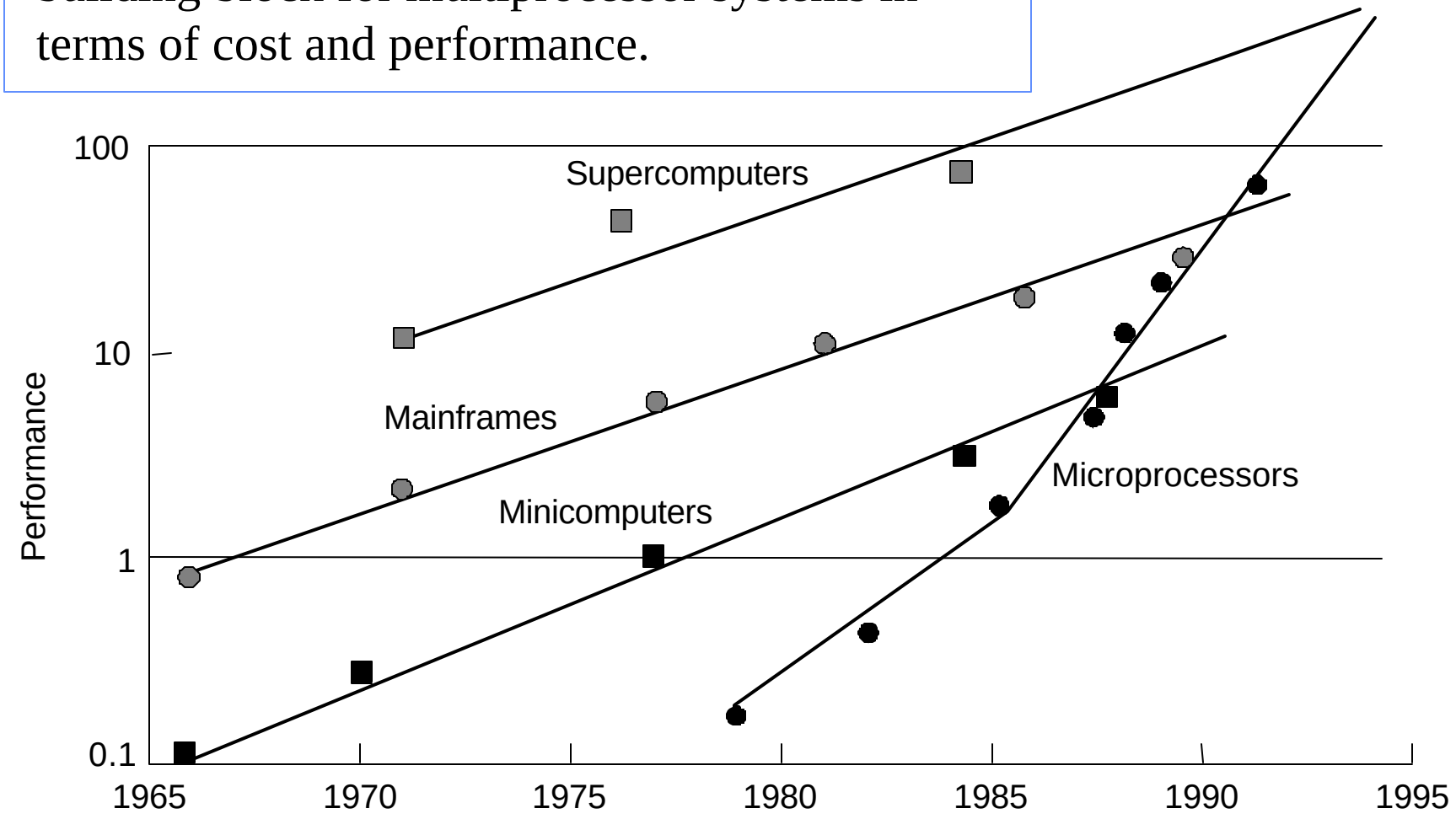
    p = Instruction decode cycles

    m = Memory cycles k = Ratio between memory/processor cycles

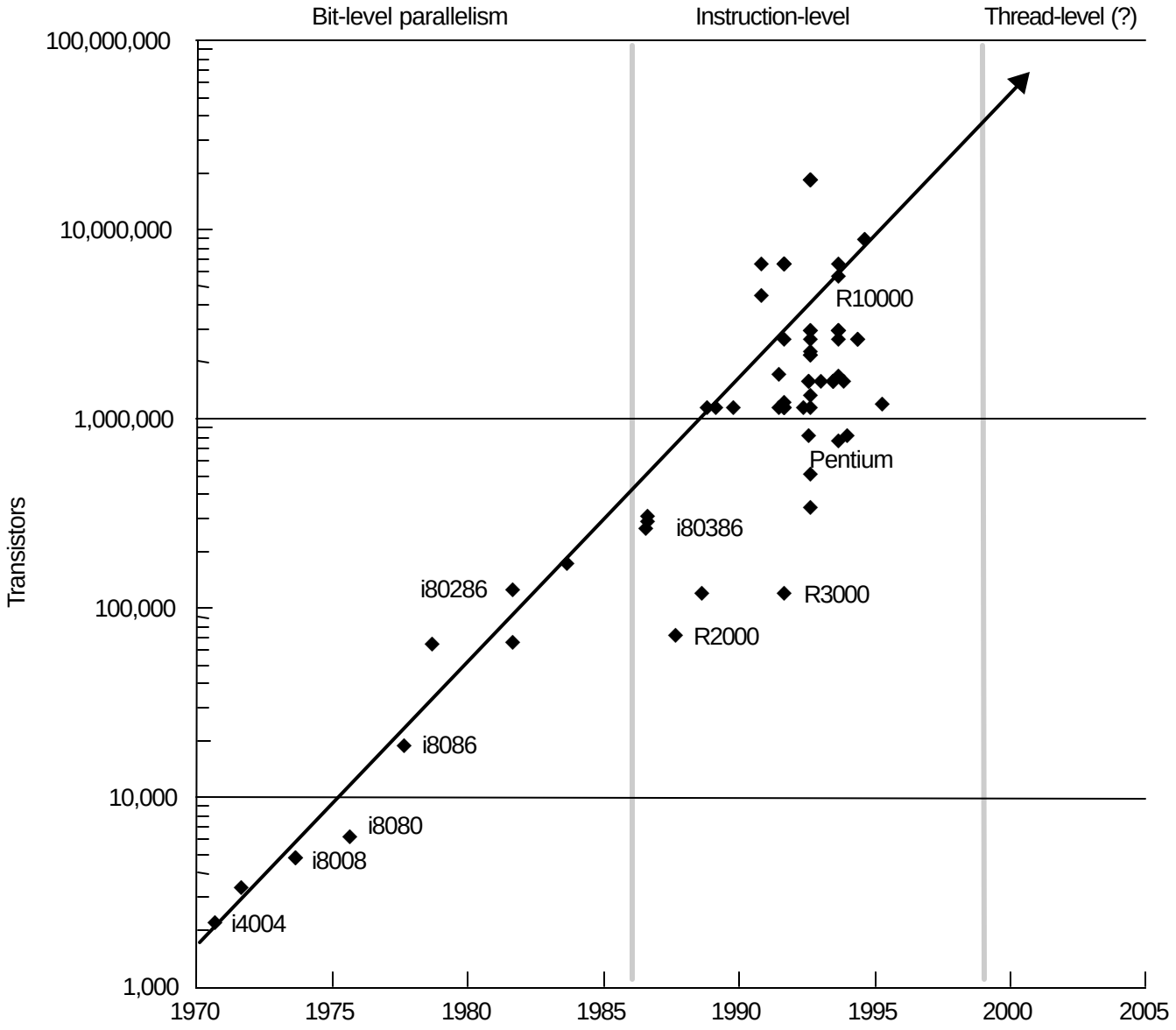
    C = Total program clock cycles f = clock rate
  - MIPS Rate = $I_c / (T \times 10^6) = f / (CPI \times 10^6) = f \times I_c / (C \times 10^6)$
  - Throughput Rate: $W_p = f / (I_c \times CPI) = (MIPS) \times 10^6 / I_c$
- Performance factors: (I_c, p, m, k, t) are influenced by: instruction-set architecture, compiler design, CPU implementation and control, cache and memory hierarchy.

CPU Performance Trends

The microprocessor is currently the most natural building block for multiprocessor systems in terms of cost and performance.



Parallelism in Microprocessor VLSI Generations



The Goal of Parallel Computing

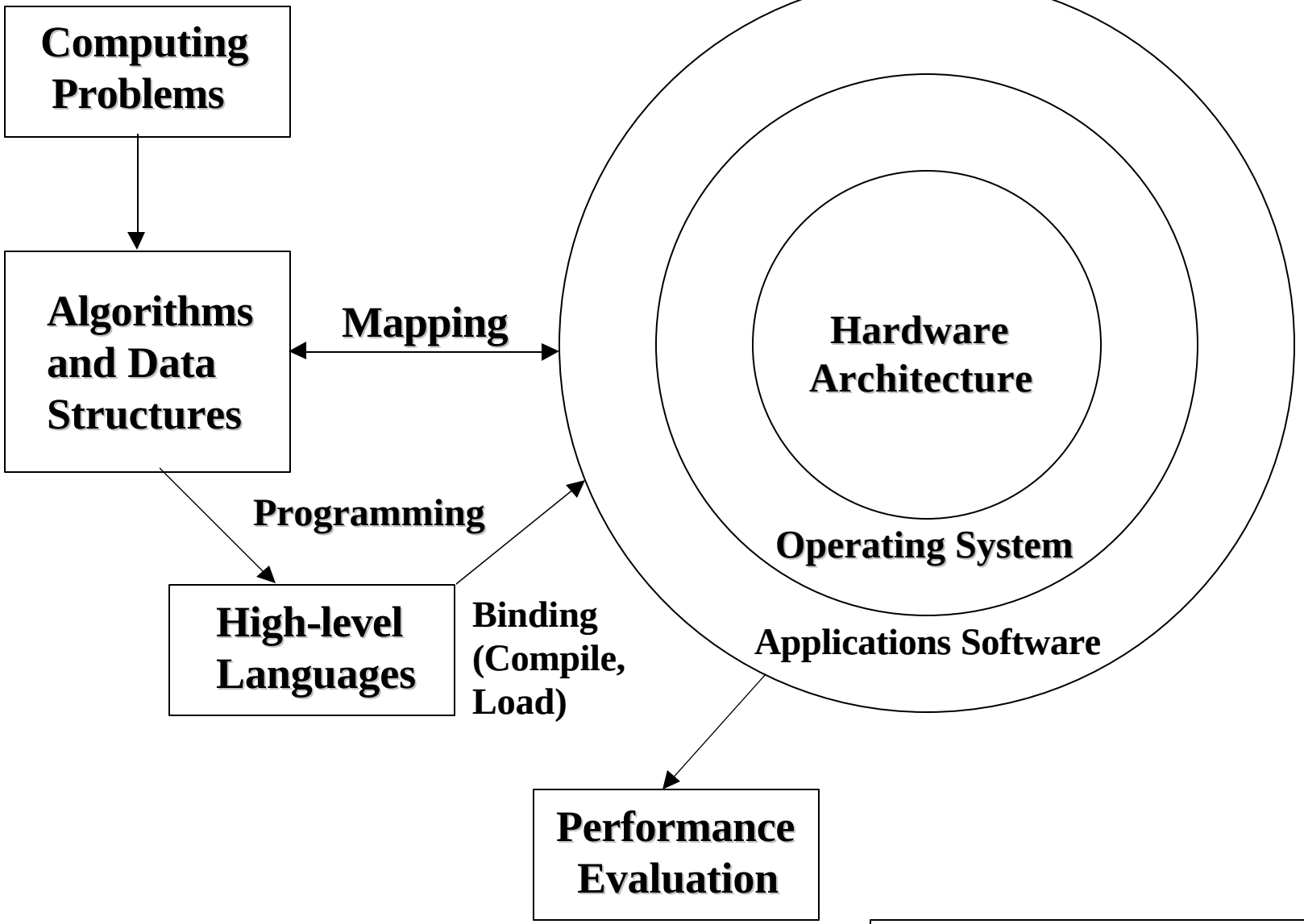
- **Goal of applications in using parallel machines: Speedup**

$$\textit{Speedup (p processors)} = \frac{\textit{Performance (p processors)}}{\textit{Performance (1 processor)}}$$

- **For a fixed problem size (input data set), performance = 1/time**

$$\textit{Speedup}_{\textit{fixed problem}}\textit{ (p processors)} = \frac{\textit{Time (1 processor)}}{\textit{Time (p processors)}}$$

Elements of Modern Computers



Elements of Modern Computers

1 **Computing Problems:**

- **Numerical Computing: Science and technology numerical problems demand intensive integer and floating point computations.**
- **Logical Reasoning: Artificial intelligence (AI) demand logic inferences and symbolic manipulations and large space searches.**

2 **Algorithms and Data Structures**

- **Special algorithms and data structures are needed to specify the computations and communication present in computing problems.**
- **Most numerical algorithms are deterministic using regular data structures.**
- **Symbolic processing may use heuristics or non-deterministic searches.**
- **Parallel algorithm development requires interdisciplinary interaction.**

Elements of Modern Computers

3 **Hardware Resources**

- **Processors, memory, and peripheral devices form the hardware core of a computer system.**
- **Processor instruction set, processor connectivity, memory organization, influence the system architecture.**

4 **Operating Systems**

- **Manages the allocation of resources to running processes.**
- **Mapping to match algorithmic structures with hardware architecture and vice versa: processor scheduling, memory mapping, interprocessor communication.**
- **Parallelism exploitation at: algorithm design, program writing, compilation, and run time.**

Elements of Modern Computers

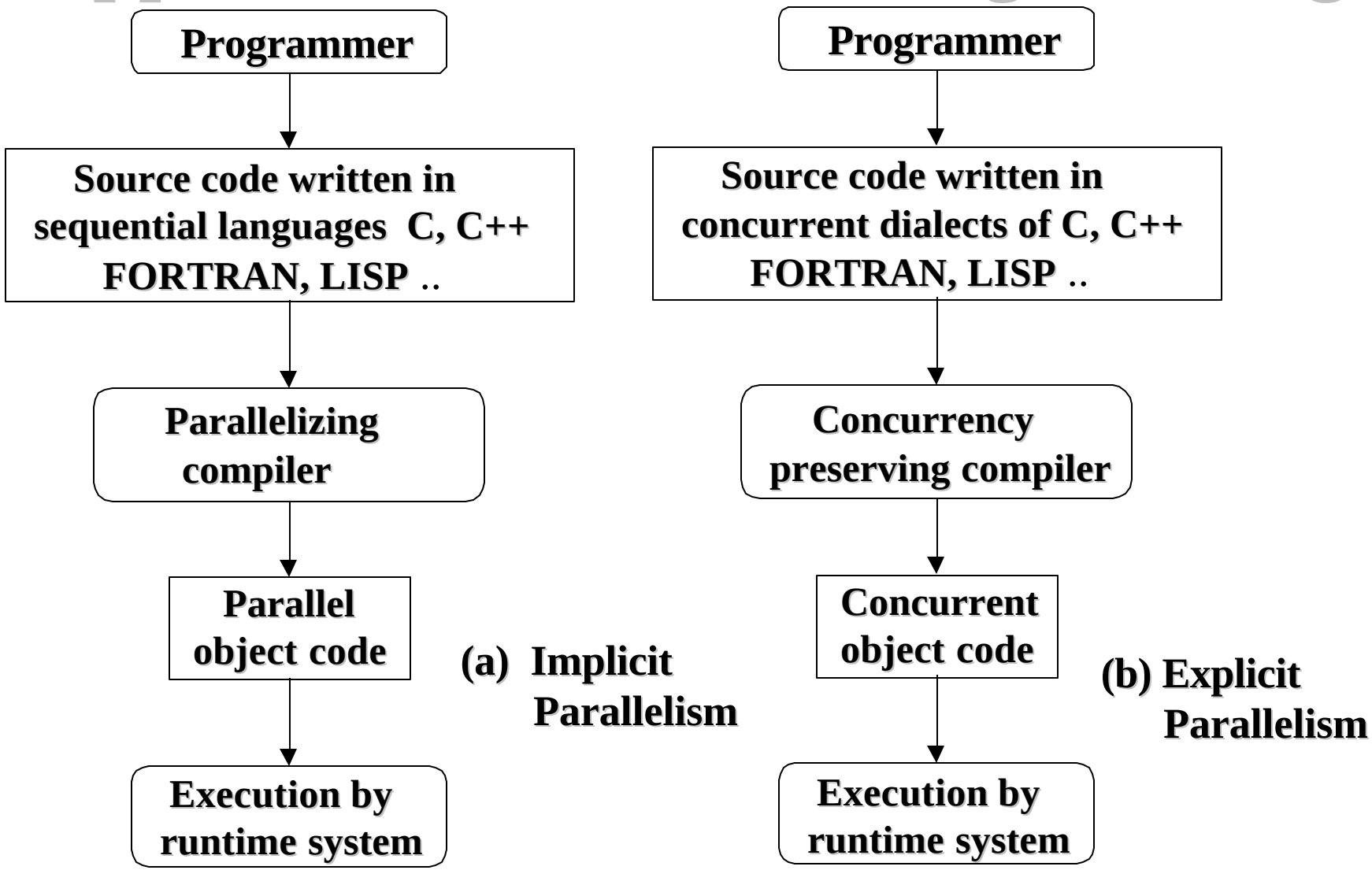
5 System Software Support

- Needed for the development of efficient programs in high-level languages (HLLs.)
- Assemblers, loaders.
- Portable parallel programming languages
- User interfaces and tools.

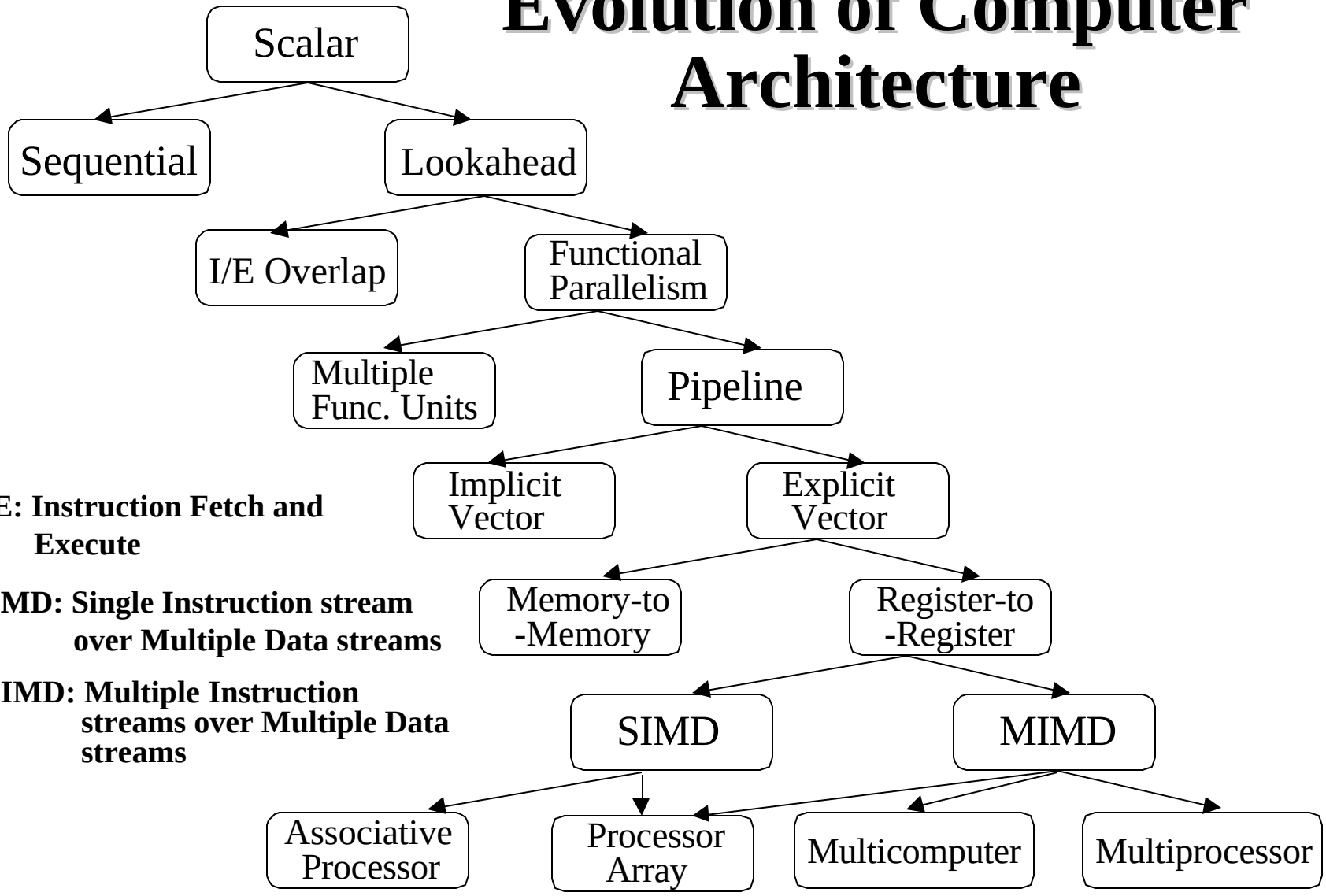
6 Compiler Support

- Preprocessor compiler: Sequential compiler and low-level library of the target parallel computer.
- Precompiler: Some program flow analysis, dependence checking, limited optimizations for parallelism detection.
- Parallelizing compiler: Can automatically detect parallelism in source code and transform sequential code into parallel constructs.

Approaches to Parallel Programming



Evolution of Computer Architecture



I/E: Instruction Fetch and Execute

SIMD: Single Instruction stream over Multiple Data streams

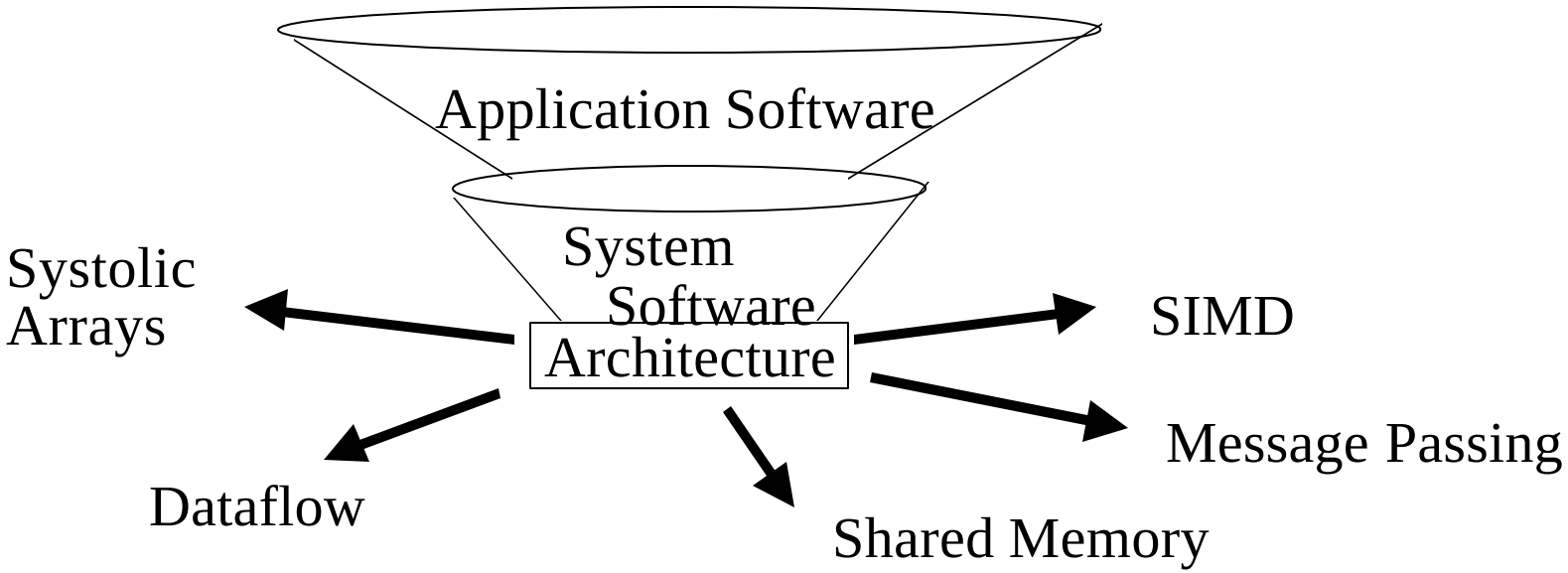
MIMD: Multiple Instruction streams over Multiple Data streams

Massively Parallel Processors (MPPs)

Parallel Architectures History

Historically, parallel architectures tied to programming models

- **Divergent architectures, with no predictable pattern of growth.**



Programming Models

- **Programming methodology used in coding applications**
- **Specifies communication and synchronization**

- **Examples:**
  - **Multiprogramming:**

    No communication or synchronization at program level
  - ***Shared memory address space:***
  - ***Message passing*:**

    Explicit point to point communication
  - ***Data parallel*:**

    More regimented, global actions on data
    - **Implemented with shared address space or message passing**

Flynn's 1972 Classification of Computer Architecture

- **Single Instruction stream over a Single Data stream (SISD): Conventional sequential machines.**
- **Single Instruction stream over Multiple Data streams (SIMD): Vector computers, array of synchronized processing elements.**
- **Multiple Instruction streams and a Single Data stream (MISD): Systolic arrays for pipelined execution.**
- **Multiple Instruction streams over Multiple Data streams (MIMD): Parallel computers:**
  - **Shared memory multiprocessors.**
  - **Multicomputers: Unshared distributed memory, message-passing used instead.**

Flynn's Classification of Computer Architecture

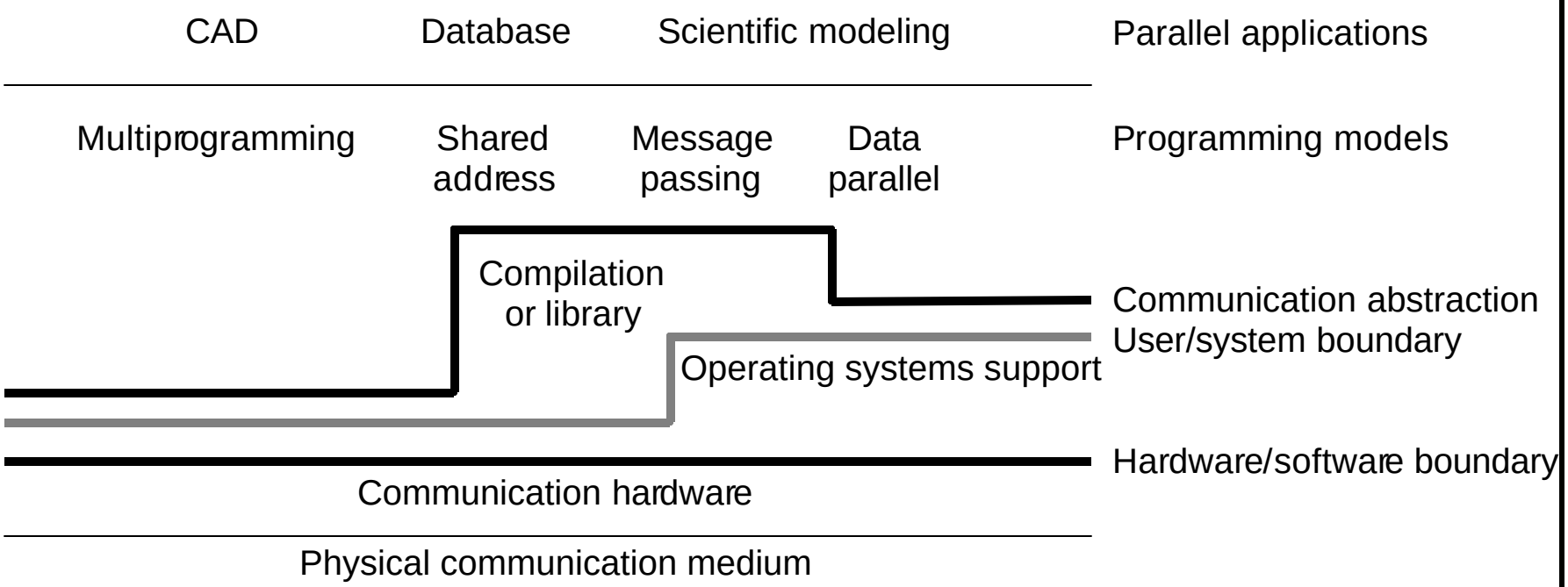
Fig. 1.3 page 12 in

Advanced Computer Architecture: Parallelism, Scalability, Programmability, Hwang, 1993.

Current Trends In Parallel Architectures

- **The extension of "computer architecture" to support communication and cooperation:**
  - **OLD: Instruction Set Architecture**
  - **NEW: *Communication Architecture***
- **Defines:**
  - **Critical abstractions, boundaries, and primitives (interfaces)**
  - **Organizational structures that implement interfaces (hardware or software)**
- **Compilers, libraries and OS are important bridges today**

Modern Parallel Architecture Layered Framework

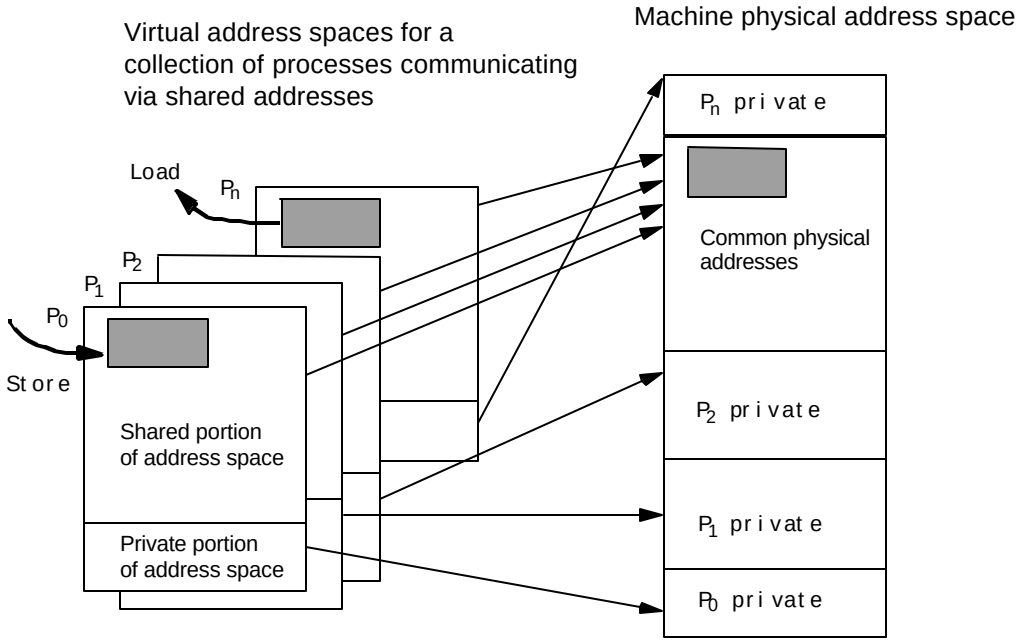


Shared Address Space Parallel Architectures

- **Any processor can <u>directly</u> reference any memory location**
  - **Communication occurs implicitly as result of loads and stores**
- **Convenient:**
  - **Location transparency**
  - **Similar programming model to time-sharing in uniprocessors**
    - **Except processes run on different processors**
    - **Good throughput on multiprogrammed workloads**
- **Naturally provided on a wide range of platforms**
  - **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 (SAS) Model

- **Process: virtual address space plus one or more threads of control**
- **Portions of address spaces of processes are shared**



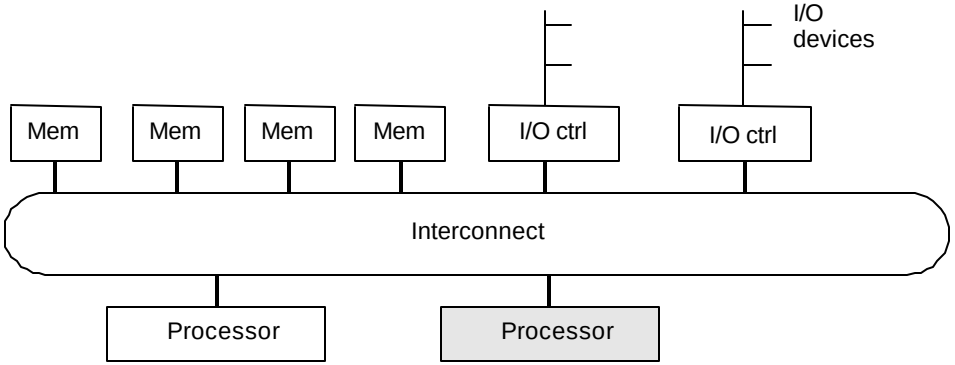
- **Writes to shared address visible to other threads (in other processes too)**
- **Natural extension of the uniprocessor model:**
  - **Conventional memory operations used for communication**
- **Special atomic operations needed for synchronization**
- **OS uses shared memory to coordinate processes**

Models of Shared-Memory Multiprocessors

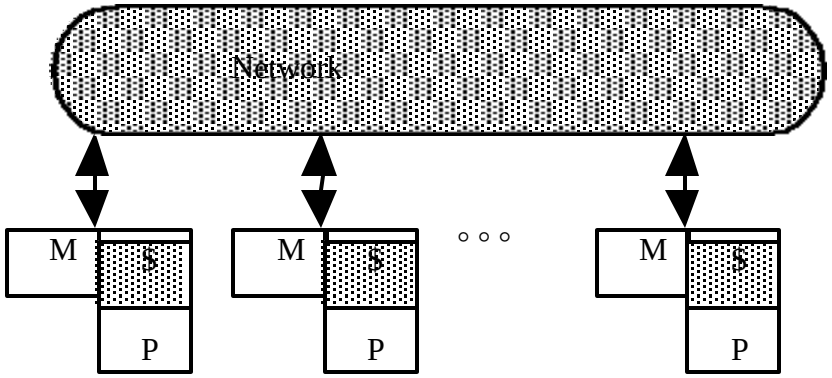
- **The Uniform Memory Access (UMA) Model:**
  - **The physical memory is shared by all processors.**
  - **All processors have equal access to all memory addresses.**
- **Distributed memory or Nonuniform Memory Access (NUMA) Model:**
  - **Shared memory is physically distributed locally among processors.**
- **The Cache-Only Memory Architecture (COMA) Model:**
  - **A special case of a NUMA machine where all distributed main memory is converted to caches.**
  - **No memory hierarchy at each processor.**

Models of Shared-Memory Multiprocessors

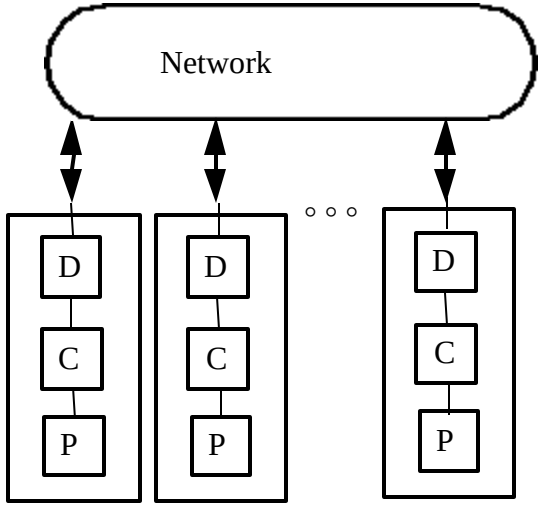
Uniform Memory Access (UMA) Model



Interconnect:
Bus, Crossbar, Multistage network
P: Processor
M: Memory
C: Cache
D: Cache directory

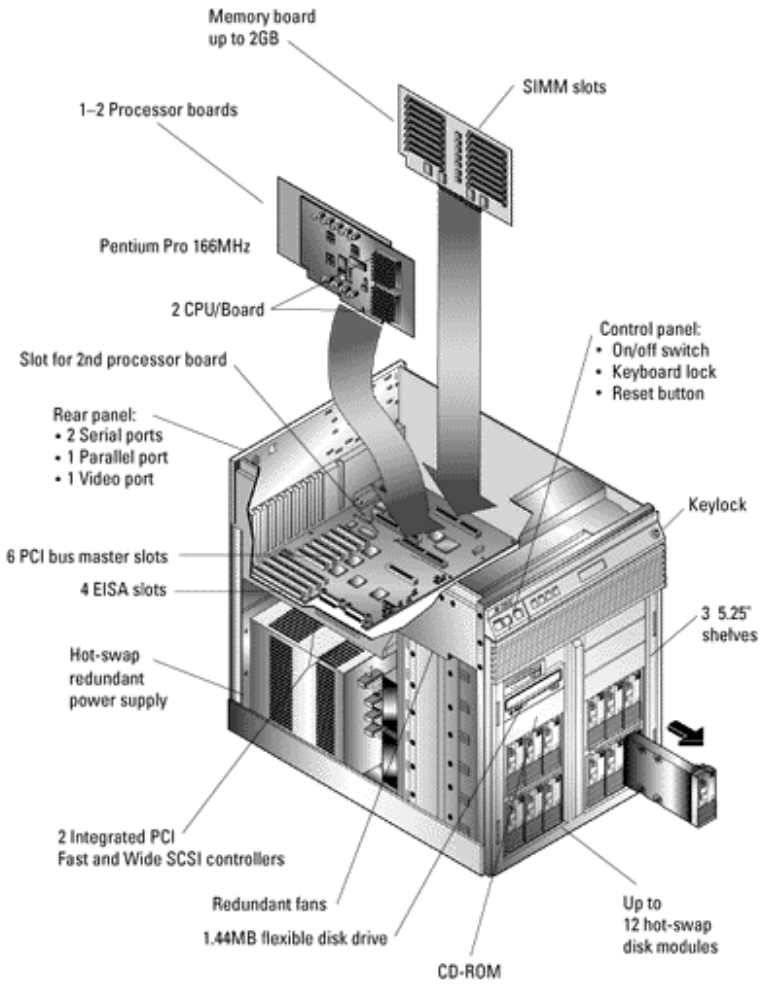


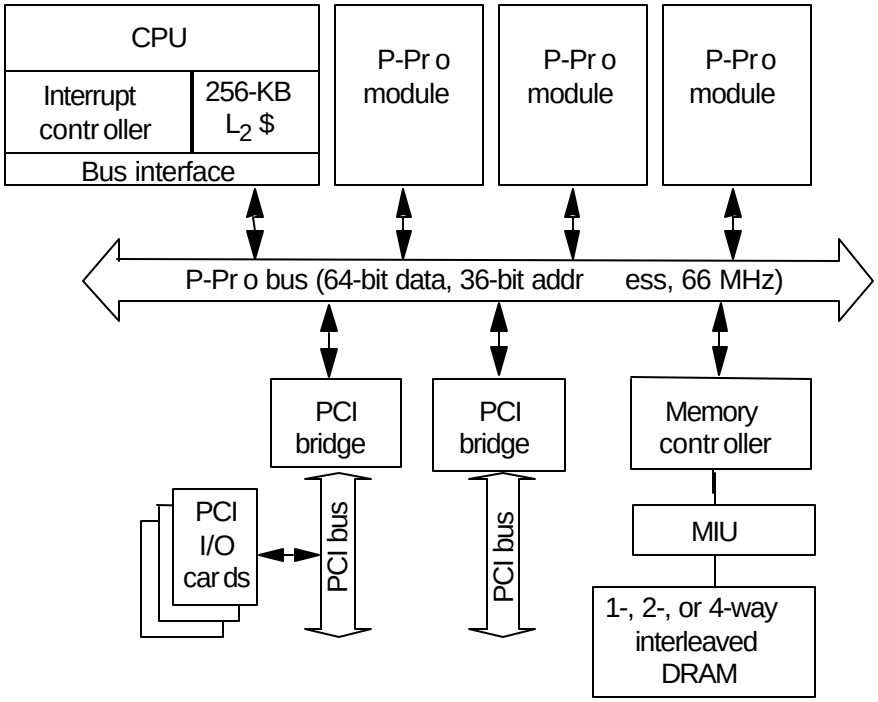
Distributed memory or
Nonuniform Memory Access (NUMA) Model



Cache-Only Memory Architecture (COMA)

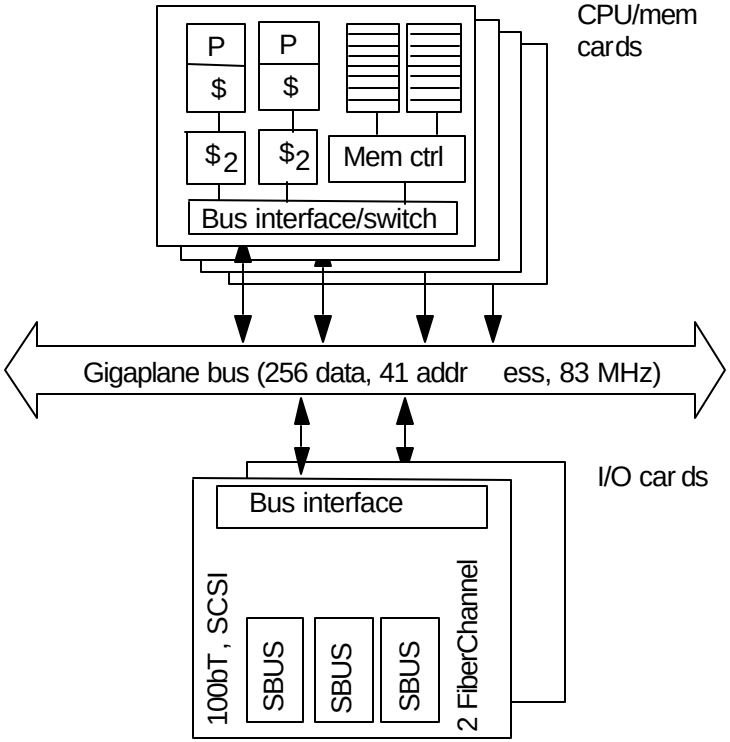
Uniform Memory Access Example: Intel Pentium Pro Quad





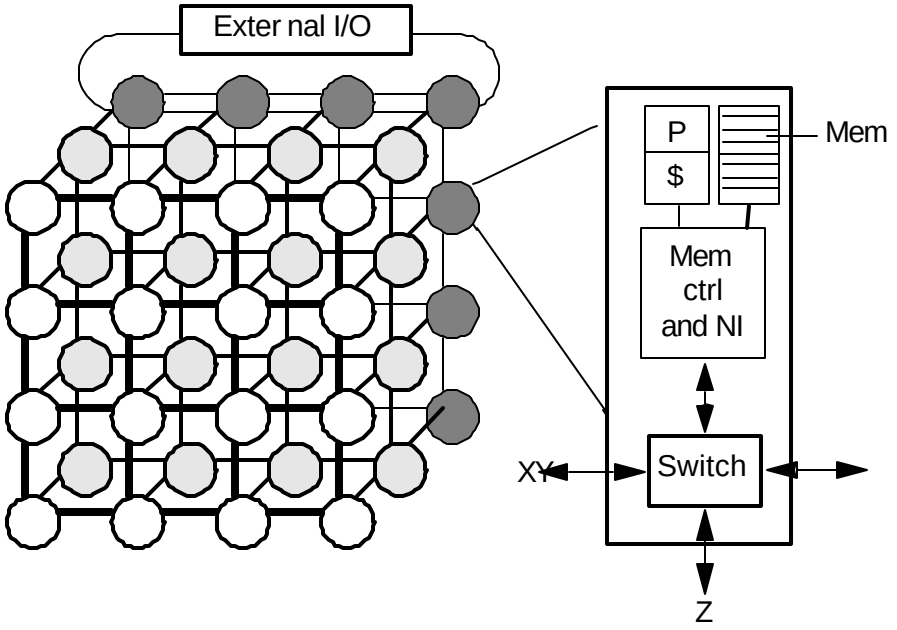
- **All coherence and multiprocessing glue in processor module**
- **Highly integrated, targeted at high volume**
- **Low latency and bandwidth**

Uniform Memory Access Example: SUN Enterprise



- 16 cards of either type: processors + memory, or I/O
- All memory accessed over bus, so symmetric
- Higher bandwidth, higher latency bus

Distributed Shared-Memory Multiprocessor System Example: Cray T3E

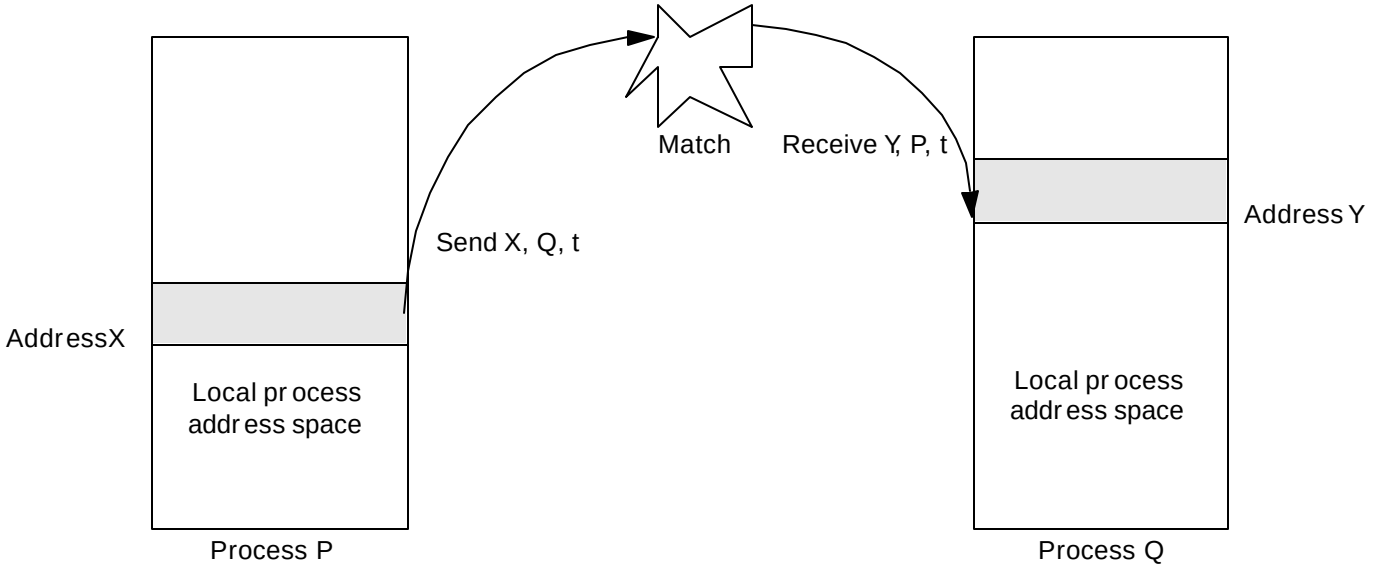


- **Scale up to 1024 processors, 480MB/s links**
- **Memory controller generates communication requests for nonlocal references**
- **No hardware mechanism for coherence (SGI Origin etc. provide this)**

Message-Passing Multicomputers

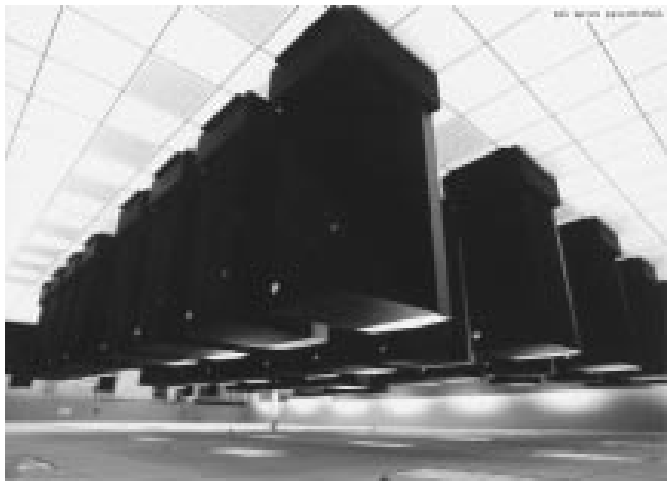
- Comprised of multiple autonomous computers (nodes).
- Each node consists of a processor, local memory, attached storage and I/O peripherals.
- Programming model more removed from basic hardware operations
- Local memory is only accessible by local processors.
- A message passing network provides point-to-point static connections among the nodes.
- Inter-node communication is carried out by message passing through the static connection network
- Process communication achieved using a message-passing programming environment.

Message-Passing Abstraction

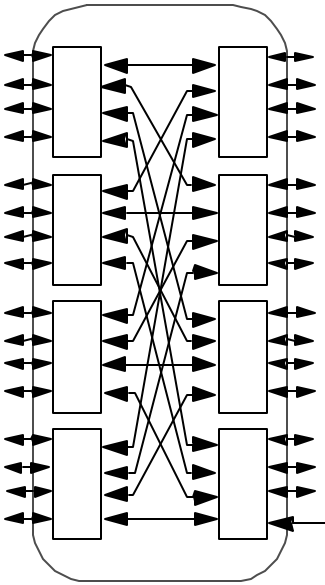


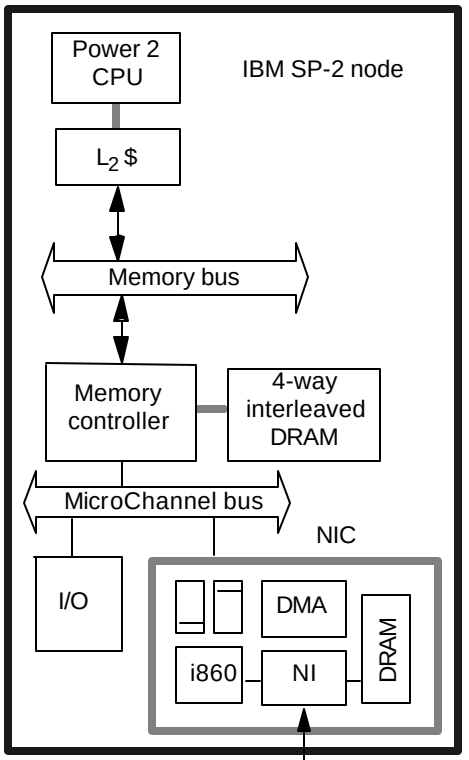
- **Send specifies buffer to be transmitted and receiving process**
- **Recv specifies sending process and application storage to receive into**
- **Memory to memory copy, but need to name processes**
- **Optional tag on send and matching rule on receive**
- **User process names local data and entities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

Message-Passing Example: IBM SP-2



General interconnection network formed from 8-port switches

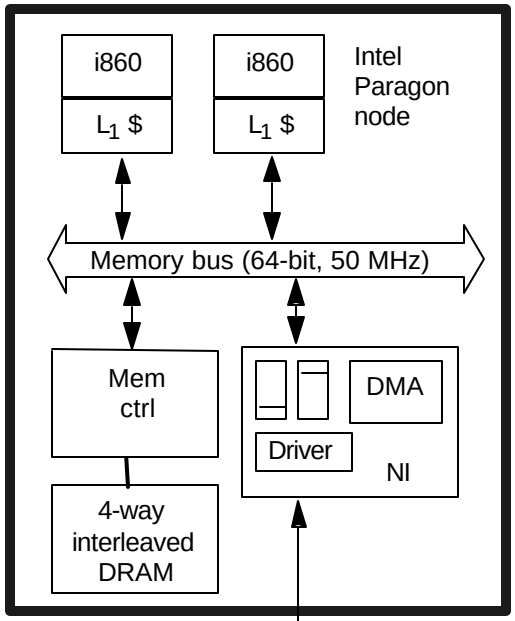


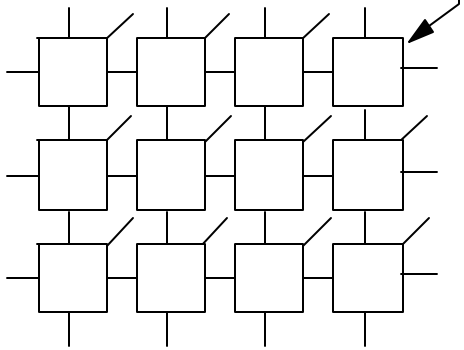


- **Made out of essentially complete RS6000 workstations**
- **Network interface integrated in I/O bus (bandwidth limited by I/O bus)**

Message-Passing Example: Intel Paragon

Sandia's Intel Paragon XP/S-based Supercomputer





8 bits, 175 MHz, bidirectional

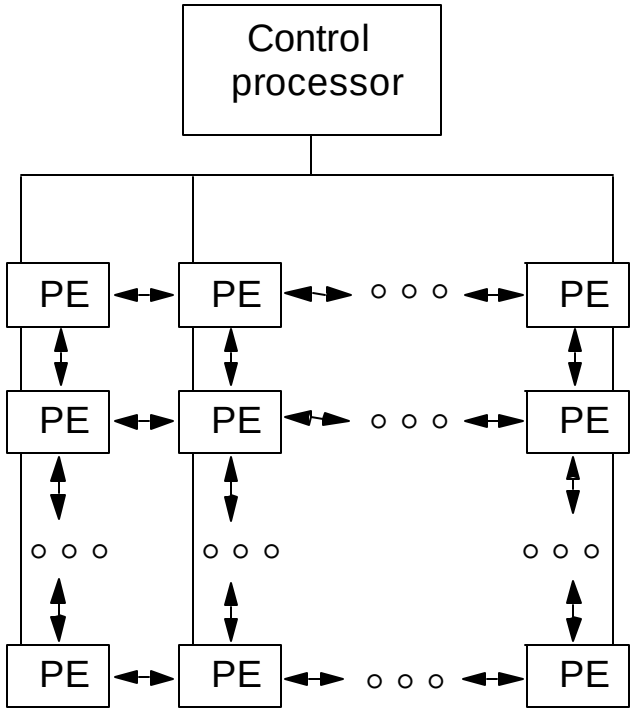
2D grid network with processing node attached to every switch

Message-Passing Programming Tools

- **Message-passing programming libraries include:**
  - **Message Passing Interface (MPI):**
    - **Provides a standard for writing concurrent message-passing programs.**
    - **MPI implementations include parallel libraries used by existing programming languages.**
  - **Parallel Virtual Machine (PVM):**
    - **Enables a collection of heterogeneous computers to used as a coherent and flexible concurrent computational resource.**
    - **PVM support software executes on each machine in a user-configurable pool, and provides a computational environment of concurrent applications.**
    - **User programs written for example in C, Fortran or Java are provided access to PVM through the use of calls to PVM library routines.**

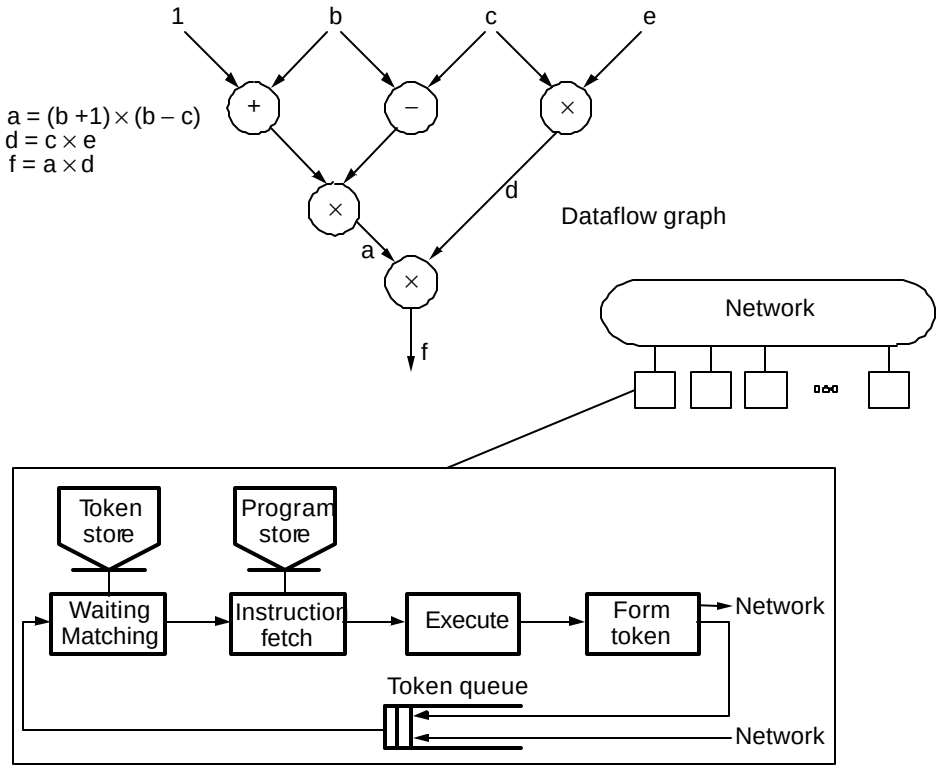
Data Parallel Systems SIMD in Flynn taxonomy

- **Programming model**
  - **Operations performed in parallel on each element of data structure**
  - **Logically single thread of control, performs sequential or parallel steps**
  - **Conceptually, a processor is associated with each data element**
- **Architectural model**
  - **Array of many simple, cheap processors each with little memory**
    - **Processors don't sequence through instructions**
  - **Attached to a control processor that issues instructions**
  - **Specialized and general communication, cheap global synchronization**
- **Example machines:**
  - **Thinking Machines CM-1, CM-2 (and CM-5)**
  - **Maspar MP-1 and MP-2,**



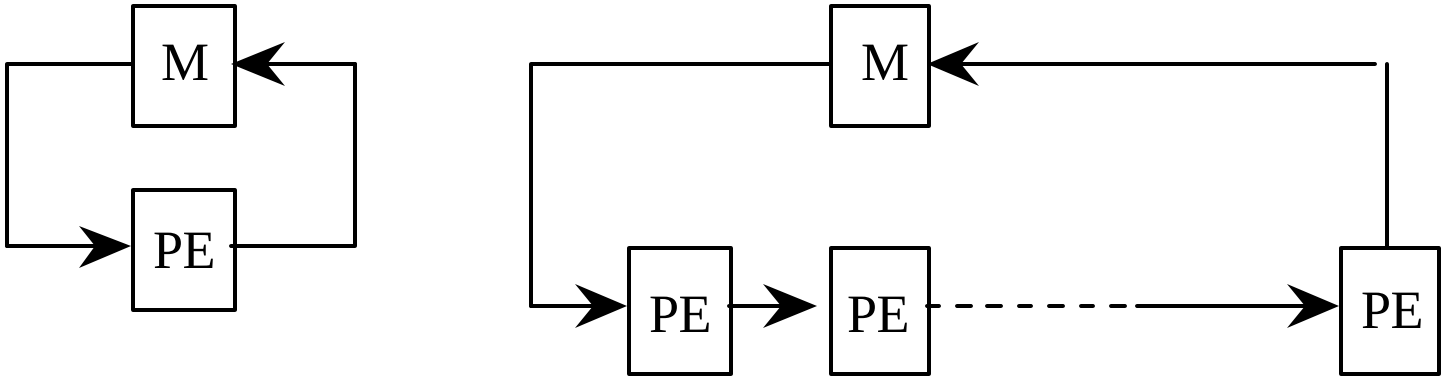
Dataflow Architectures

- **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**



Systolic Architectures

- Replace single processor with an 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
- Different from SIMD: each PE may do something different
- Initial motivation: VLSI enables inexpensive special-purpose chips
- Represent algorithms directly by chips connected in regular pattern