CMSC 611: Advanced Computer Architecture

Parallel Systems

Parallel Computers

Definition: “A parallel computer is a collection of processing elements that cooperate and communicate to solve large problems fast.”

- Almasi and Gottlieb, Highly Parallel Computing ,1989

Parallel machines are expected to have a bigger role in the future since:

- Microprocessors are likely to remain dominant in the uniprocessor arena and the logical way to extend the performance is by connecting multiple microprocessors
- It is not expected that the microprocessor technology will keep the pace of performance improvement given the increased level of complexity
- There has been steady progress in software development for parallel architectures in recent years

* Slide is a courtesy of Dave Patterson

Questions about parallel computers:

How large a collection?

How powerful are processing elements?

How do they cooperate and communicate?

How are data transmitted?

What type of interconnection?

What are HW and SW primitives for programmers?

Does it translate into performance?

Level of Parallelism

Bit-level parallelism

- ALU parallelism: 1-bit, 4-bits, 8-bit, ...

Instruction-level parallelism (ILP)

- Pipelining, Superscalar, VLIW, Out-of-Order execution

Process/Thread-level parallelism

- Divide job into parallel tasks

Job-level parallelism

- Independent jobs on one computer system

Applications

Scientific Computing

- Nearly Unlimited Demand (Grand Challenge):
- Successes in some real industries:
  - Petroleum: reservoir modeling
  - Automotive: crash simulation, drag analysis, engine
  - Aeronautics: airflow analysis, engine, structural mechanics
  - Pharmaceuticals: molecular modeling

App	Perf (GFLOPS)	Memory (GB)
48 hour weather	0.1	0.1
72 hour weather	3	1
Pharmaceutical design	100	10
Global Change, Genome	1000	1000

* Slide is a courtesy of Dave Patterson

Commercial Applications

Transaction processing

File servers

Electronic CAD simulation

Large WWW servers

WWW search engines

Graphics

- Graphics hardware
- Render Farms

Framework

Extend traditional computer architecture with a communication architecture

- abstractions (HW/SW interface)
- organizational structure to realize abstraction efficiently

Programming Model:

- Multiprogramming: lots of jobs, no communication
- Shared address space: communicate via memory
- Message passing: send and receive messages
- Data Parallel: several agents operate on several data sets simultaneously and then exchange information globally and simultaneously (shared or message passing)

Communication Abstraction:

- Shared address space: e.g., load, store, atomic swap
- Message passing: e.g., send, receive library calls
- Debate over this topic (ease of programming, scaling)
  → many hardware designs 1:1 programming model

Taxonomy of Parallel Architecture

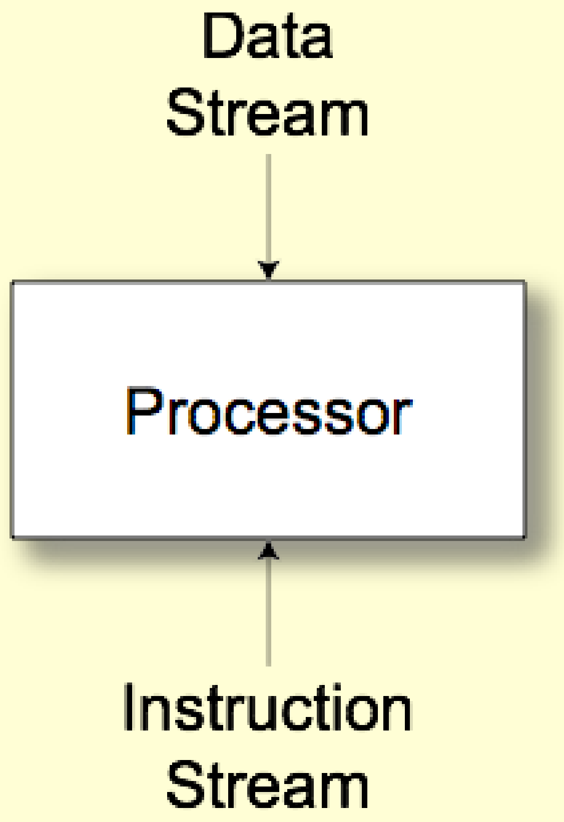
Flynn Categories

- SISD (Single Instruction Single Data)
- MISD (Multiple Instruction Single Data)
- SIMD (Single Instruction Multiple Data)
- MIMD (Multiple Instruction Multiple Data)

* Slide is a courtesy of Dave Patterson

SISD

Uniprocessor

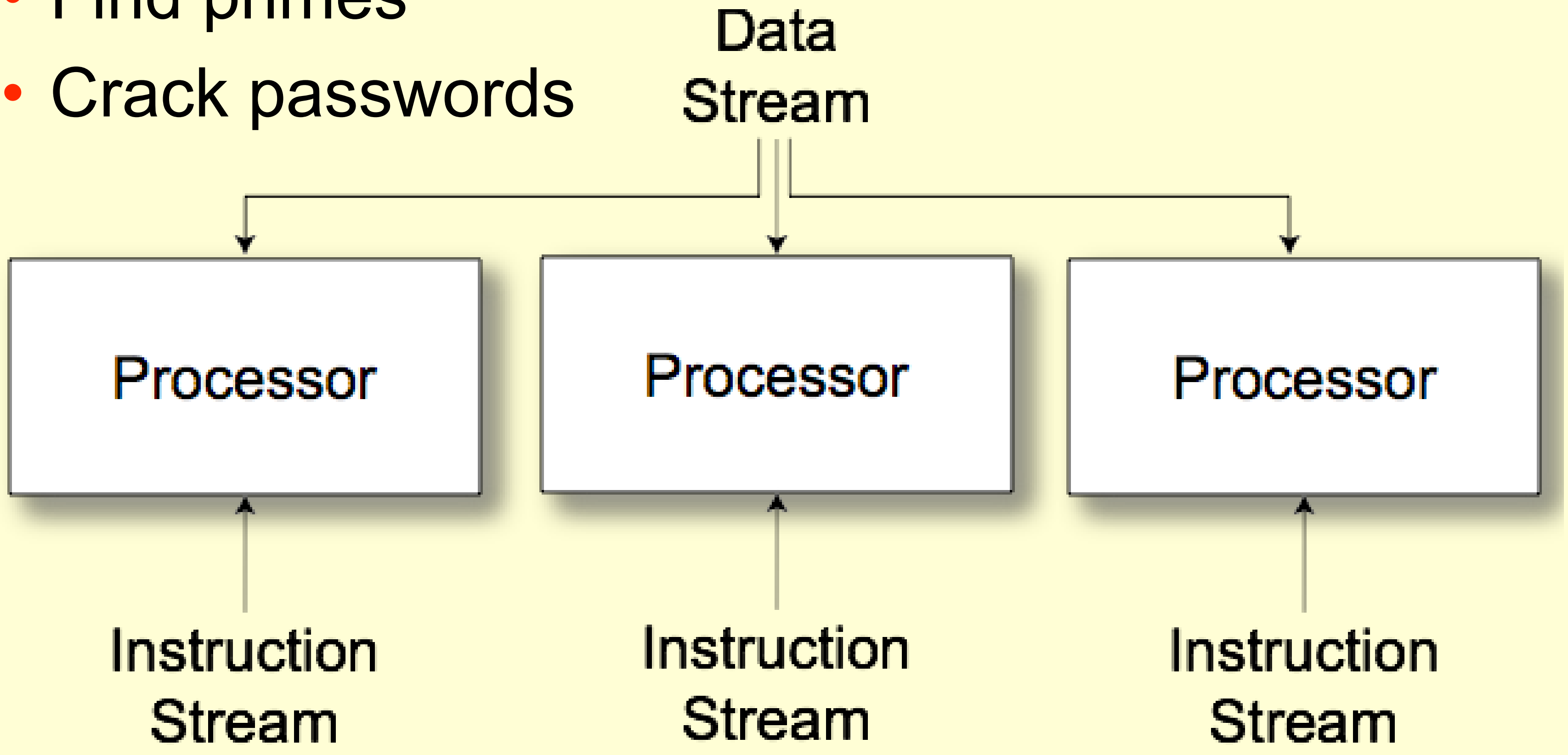


MISD

No commercial examples

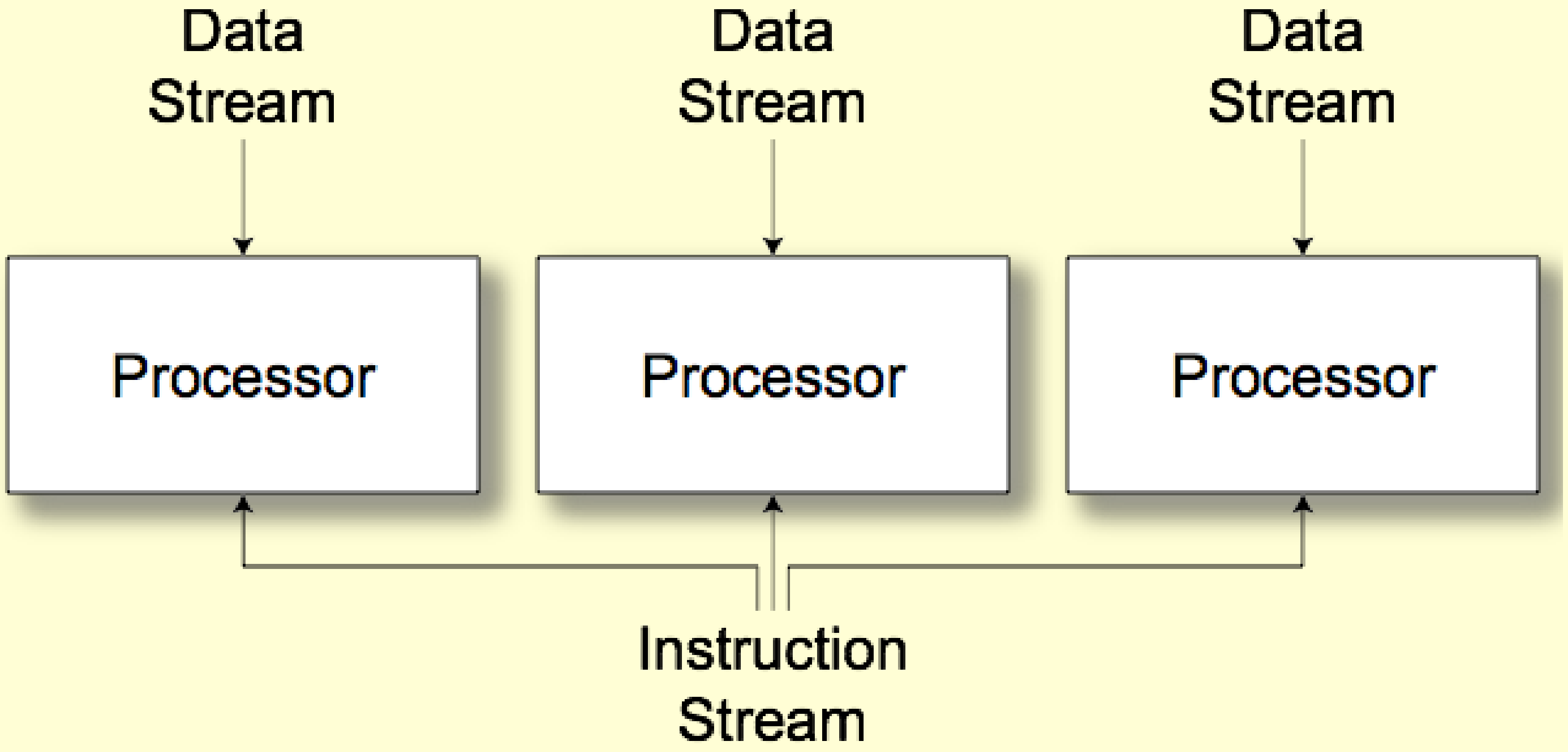
Apply same operations to a set of data

- Find primes
- Crack passwords



SIMD

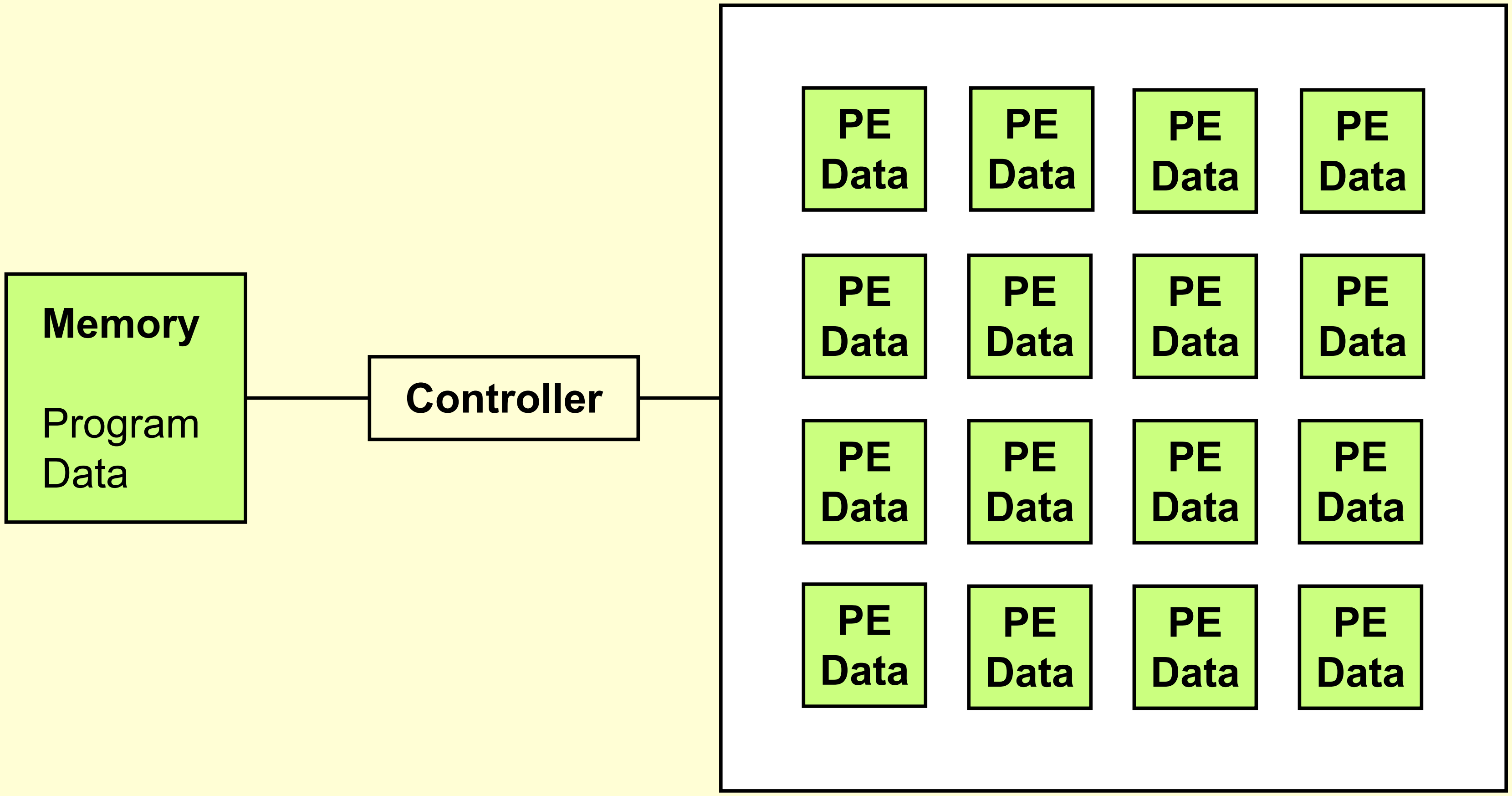
Vector/Array computers



SIMD Arrays

Performance keys

- Utilization
- Communication



Data Parallel Model

Operations performed in parallel on each element of a large regular data structure, such as an array

- One Control Processor broadcast to many processing elements (PE) with condition flag per PE so that can skip

For distributed memory architecture data is distributed among memories

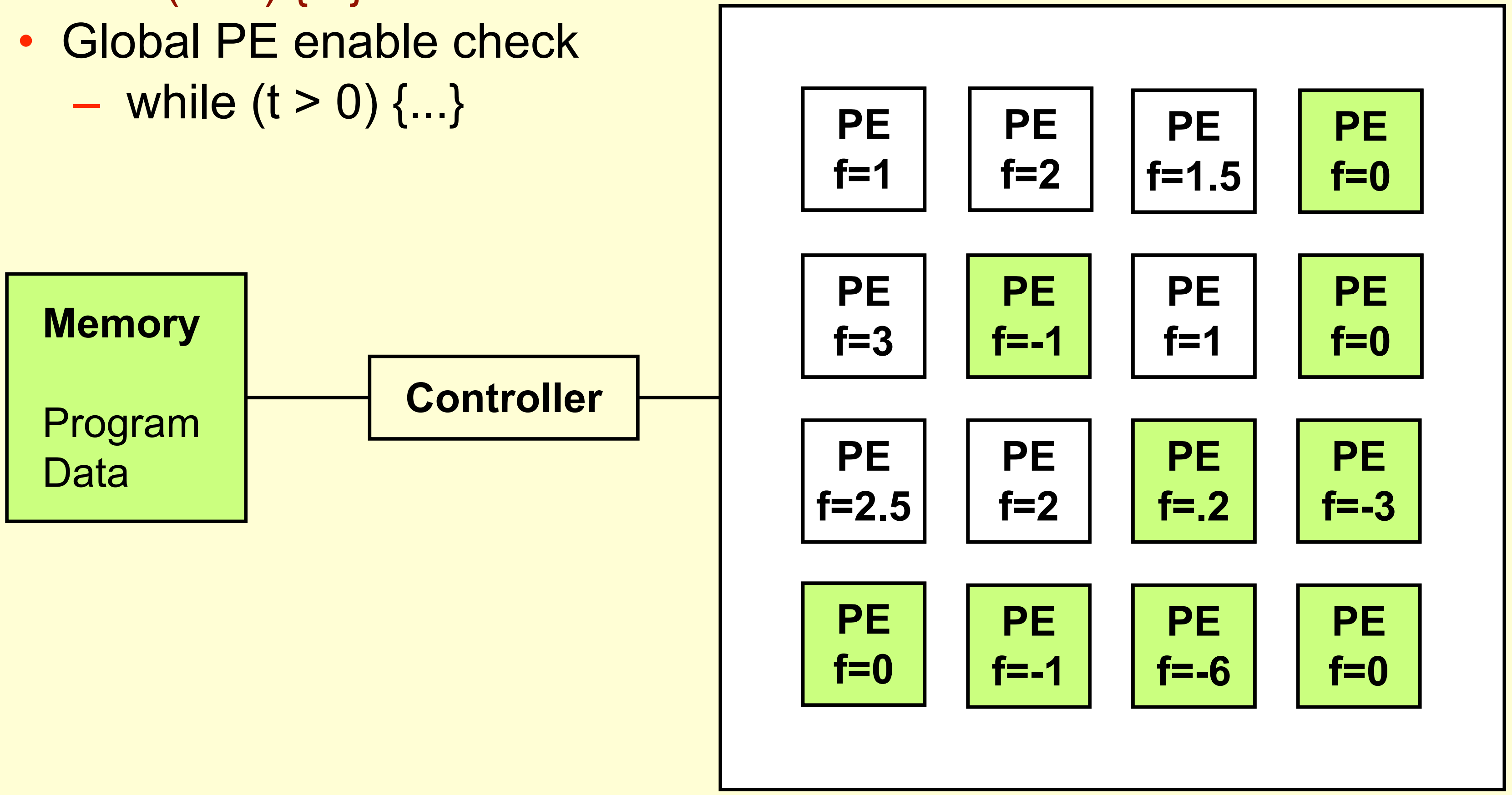
- Data parallel model requires fast global synchronization
- Data parallel programming languages lay out data to processor
- Vector processors have similar ISAs, but no data placement restriction

* Slide is a courtesy of Dave Patterson

SIMD Utilization

Conditional Execution

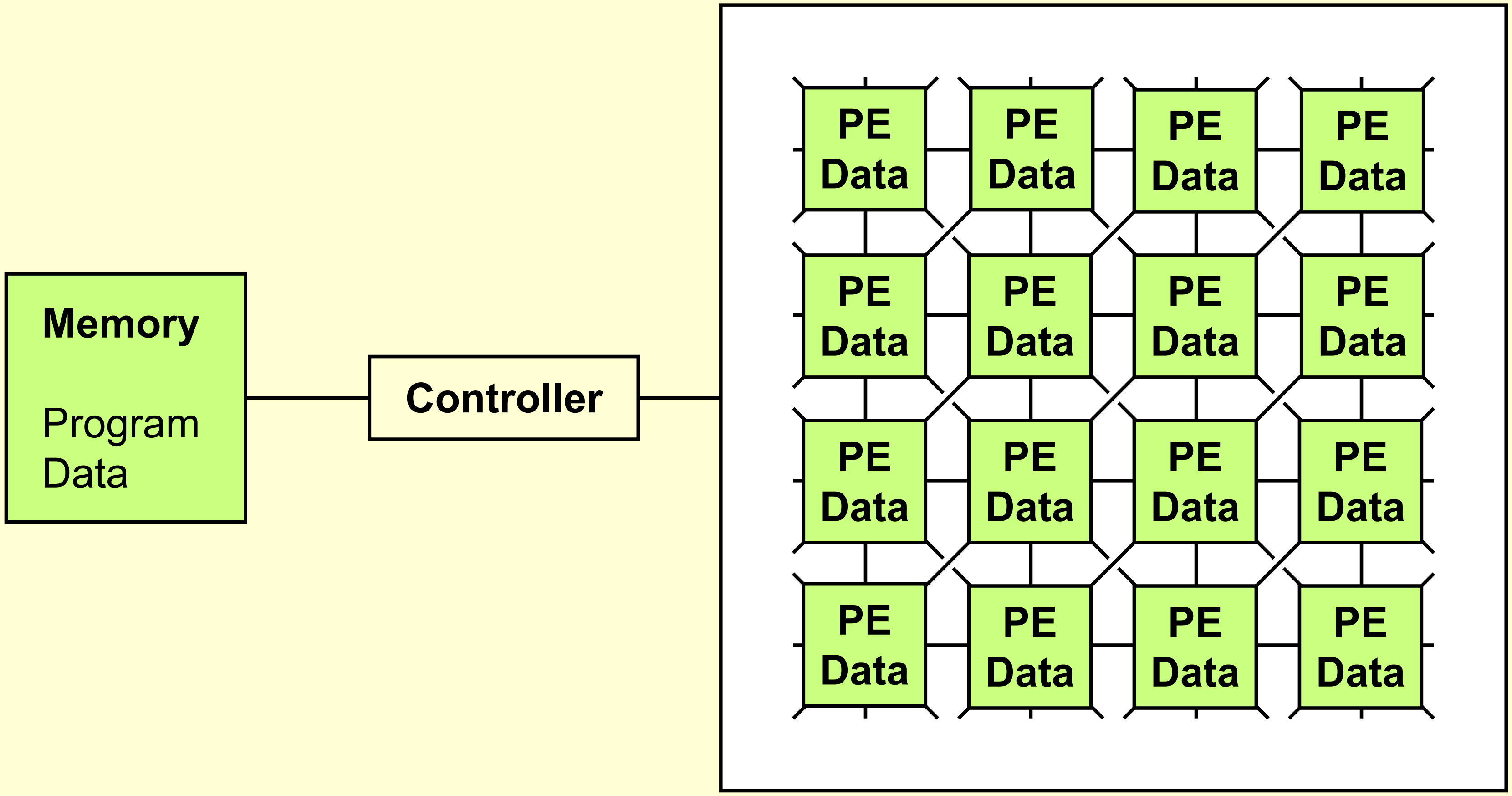
- PE Enable
  - if (f<.5) {...}
- Global PE enable check
  - while (t > 0) {...}



Communication: MasPar MP1

Fast local X-net

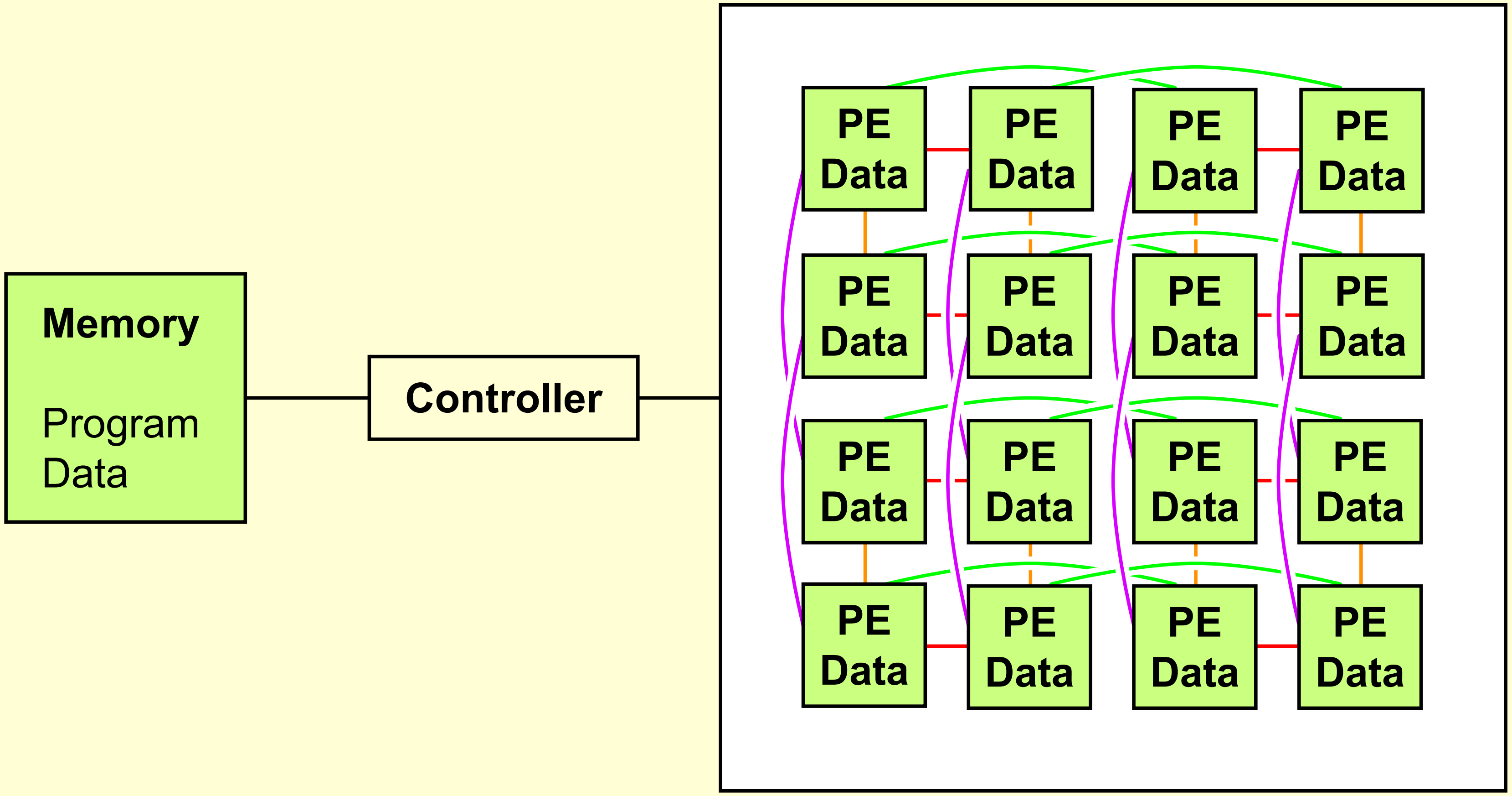
Slow global routing



Comunication: CM2

Hypercube local routing

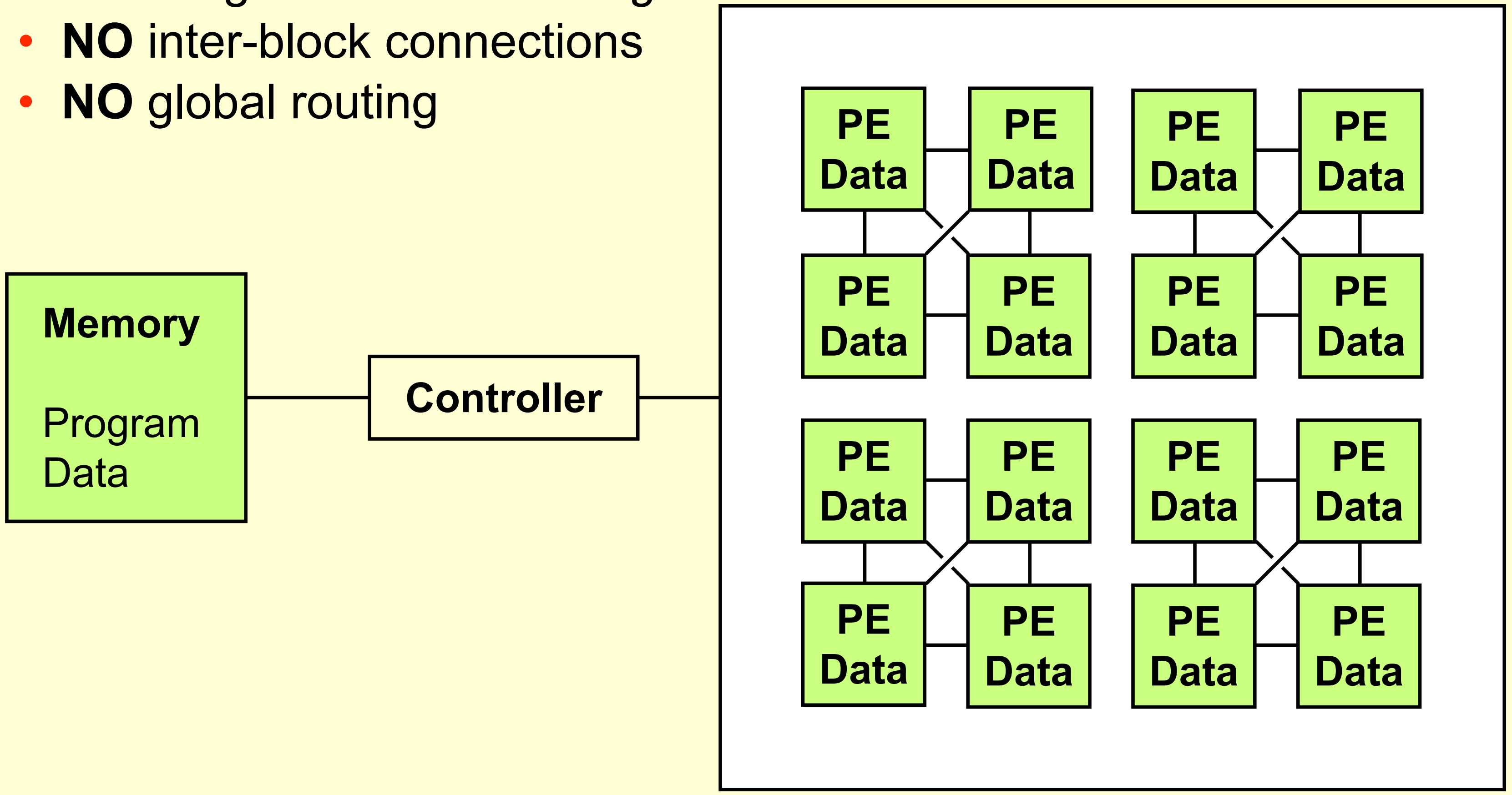
Wormhole global routing



Communication: PixelFlow

Dense connections within block

- Single *swizzle* operation collects one word from each PE in block
  - Designed for *antialiasing*
- **NO** inter-block connections
- **NO** global routing

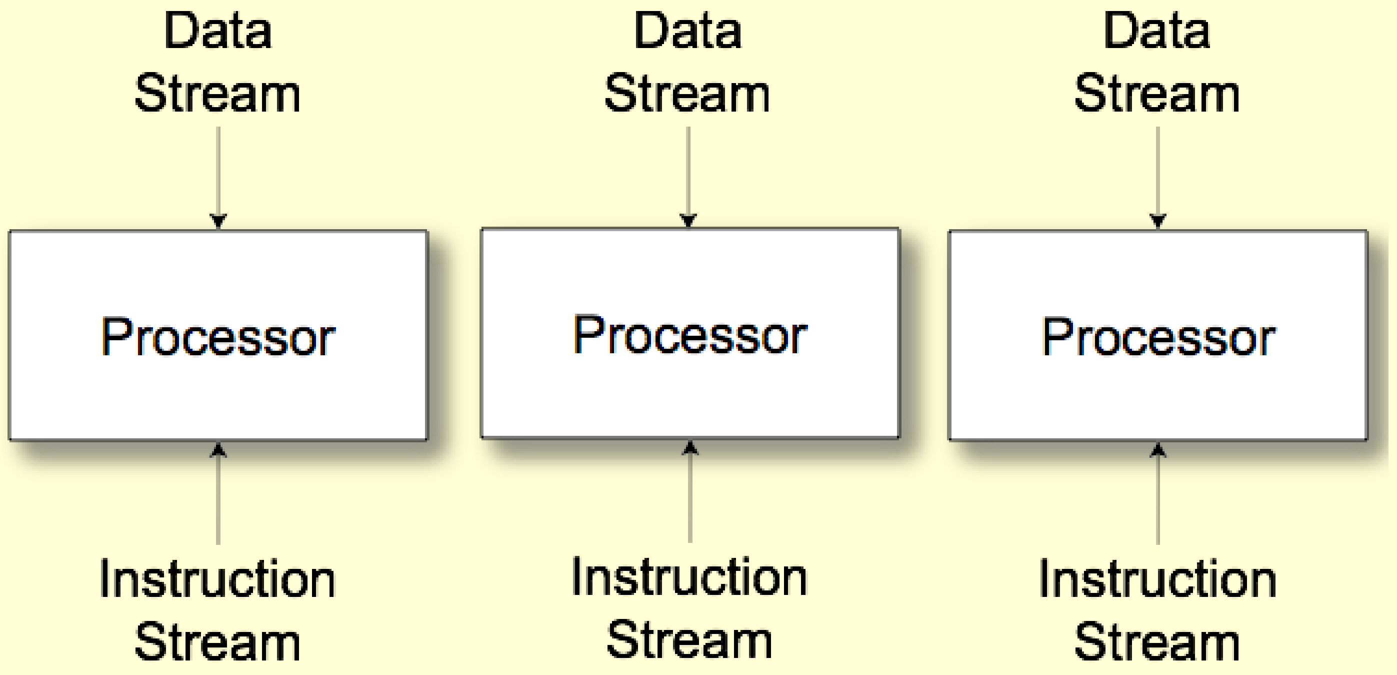


MIMD

Message Passing

Shared memory/distributed memory

- Uniform Memory Access (UMA)
- Non-Uniform Memory Access (NUMA)



Can support either SW model on either HW basis

Message passing

Processors have private memories, communicate via messages

Advantages:

- Less hardware, easier to design
- Focuses attention on costly non-local operations

Message Passing Model

Each PE has local processor, data, (I/O)

- Explicit I/O to communicate with other PEs
- Essentially NUMA but integrated at I/O vs. memory system

Free run between Send & Receive

- Send + Receive = Synchronization between processes (event model)
  - Send: local buffer, remote receiving process/port
  - Receive: remote sending process/port, local buffer

History of message passing

Early machines

- Local communication
- Blocking send & receive

Later: DMA with non-blocking sends

- DMA for receive into buffer until processor does receive, and then data is transferred to local memory

Later still: SW libraries to allow arbitrary communication

Example

IBM SP-2, RS6000 workstations in racks

- Network Interface Card has Intel 960
- 8X8 Crossbar switch as communication building block
- 40 MByte/sec per link

Shared Memory

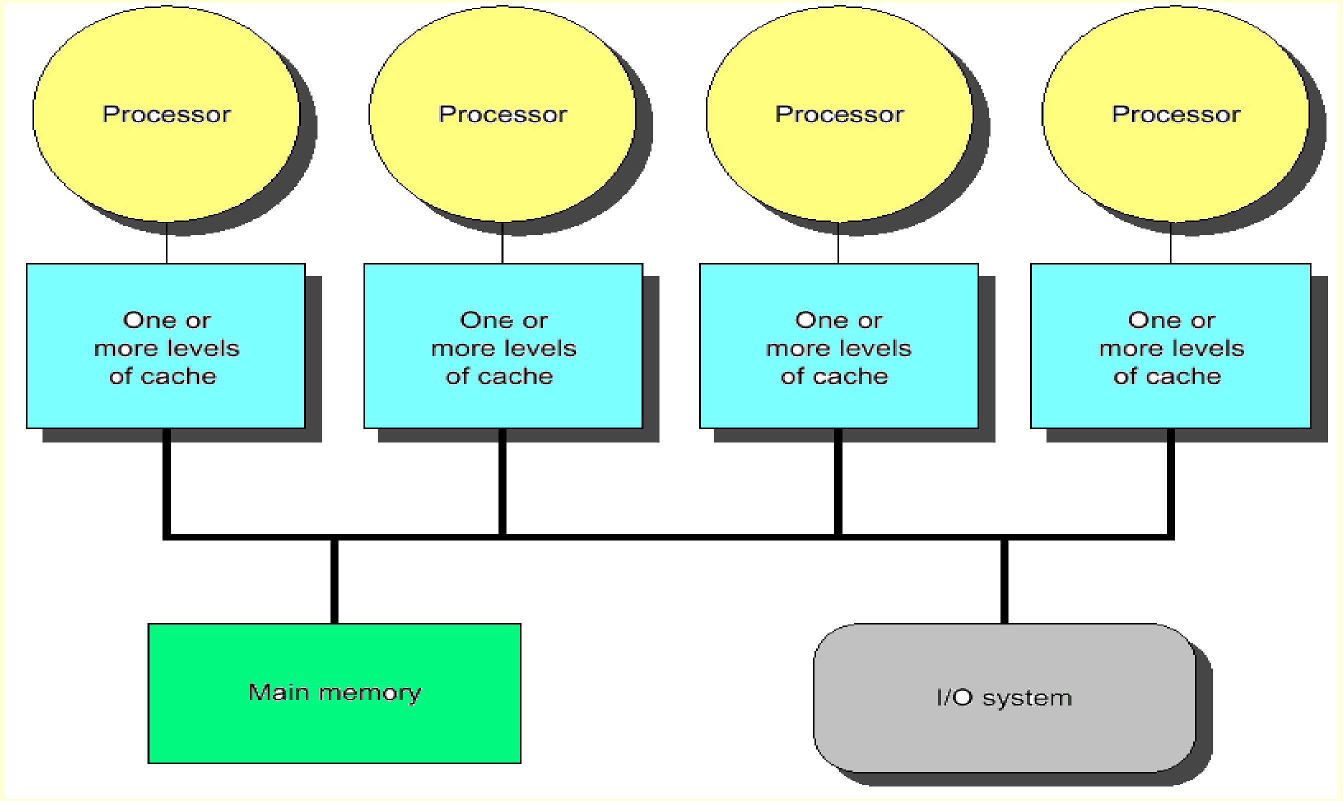
Processors communicate with shared address space

Easy on small-scale machines

Advantages:

- Model of choice for uniprocessors, small-scale multiprocessor
- Ease of programming
- Lower latency
- Easier to use hardware controlled caching
- Difficult to handle node failure

Centralized Shared Memory

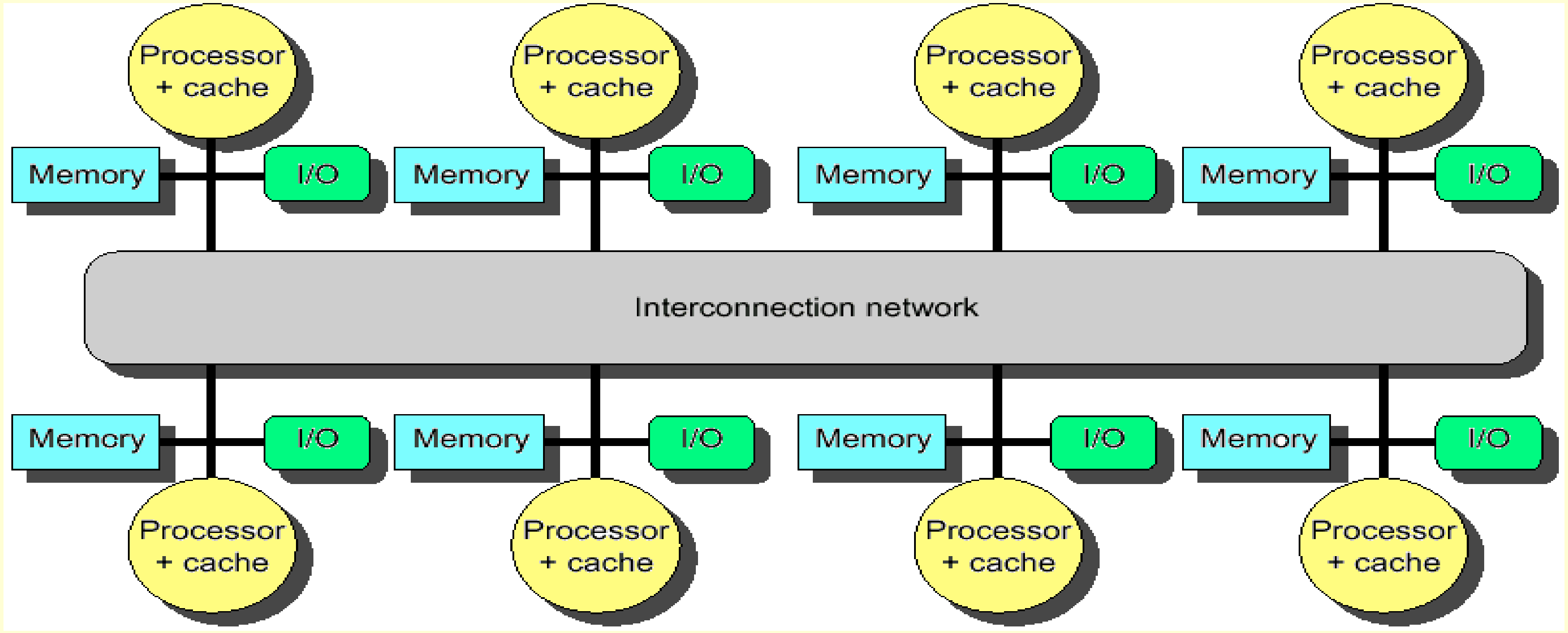

Processor
Processor
Processor
Processor
One or more levels of cache
One or more levels of cache
One or more levels of cache
One or more levels of cache
Main memory
I/O system

Processors share a single centralized (UMA) memory through a bus interconnect

Feasible for small processor count to limit memory contention

Centralized shared memory architectures are the most common form of MIMD design

Distributed Memory



Uses physically distributed (NUMA) memory to support large processor counts (to avoid memory contention)

Advantages

- Allows cost-effective way to scale the memory bandwidth
- Reduces memory latency

Disadvantage

- Increased complexity of communicating data

Shared Address Model

Physical locations

- Each PE can name every physical location in the machine

Shared data

- Each process can name all data it shares with other processes

Data transfer

- Use load and store, VM maps to local or remote location
- Extra memory level: cache remote data
- Significant research on making the translation transparent and scalable for many nodes
  - Handling data consistency and protection challenging
  - Latency depends on the underlying hardware architecture (bus bandwidth, memory access time and support for address translation)
  - Scalability is limited given that the communication model is so tightly coupled with process address space

* Slide is a courtesy of Dave Patterson

Data Parallel Languages

SIMD programming

- PE point of view
- Data: shared or per-PE
  - What data is distributed?
  - What is shared over PE subset
  - What data is broadcast with instruction stream?
- Data layout: shape [256][256]d;
- Communication primitives
- Higher-level operations
  - Prefix sum: [i]r = $\sum_{j \le i}$ [j]d
    - 1,1,2,3,4 → 1,1+1=2,2+2=4,4+3=7,7+4=11

Single Program Multiple Data

Many problems do not map well to SIMD

- Better utilization from MIMD or ILP

Data parallel model ⇒ *Single Program Multiple Data* (SPMD) model

- All processors execute identical program
- Same program for SIMD, SISD or MIMD
- Compiler handles mapping to architecture

Three Fundamental Issues

1: Naming: how to solve large problem fast

- what data is shared
- how it is addressed
- what operations can access data
- how processes refer to each other

Choice of naming affects code produced by a compiler

- Just remember and load address or keep track of processor number and local virtual address for message passing

Choice of naming affects replication of data

- In cache memory hierarchy or via SW replication and consistency

* Slide is a courtesy of Dave Patterson

Naming Address Spaces

Global physical address space

- any processor can generate, address and access it in a single operation

Global virtual address space

- if the address space of each process can be configured to contain all shared data of the parallel program
  - memory can be anywhere: virtual address translation handles it

Segmented shared address space

- locations are named <process number, address> uniformly for all processes of the parallel program

Three Fundamental Issues

2: Synchronization: To cooperate, processes must coordinate

- Message passing is implicit coordination with transmission or arrival of data
- Shared address → additional operations to explicitly coordinate: e.g., write a flag, awaken a thread, interrupt a processor

* Slide is a courtesy of Dave Patterson

Three Fundamental Issues

3: Latency and Bandwidth

Bandwidth

- Need high bandwidth in communication
- Cannot scale, but stay close
- Match limits in network, memory, and processor
- Overhead to communicate is a problem in many machines

Latency

- Affects performance, since processor may have to wait
- Affects ease of programming, since requires more thought to overlap communication and computation

Latency Hiding

- How can a mechanism help hide latency?
- Examples: overlap message send with computation, pre-fetch data, switch to other tasks

Three Graphics Examples

Pixel-Planes 4

- 512x512 SIMD array (full screen)

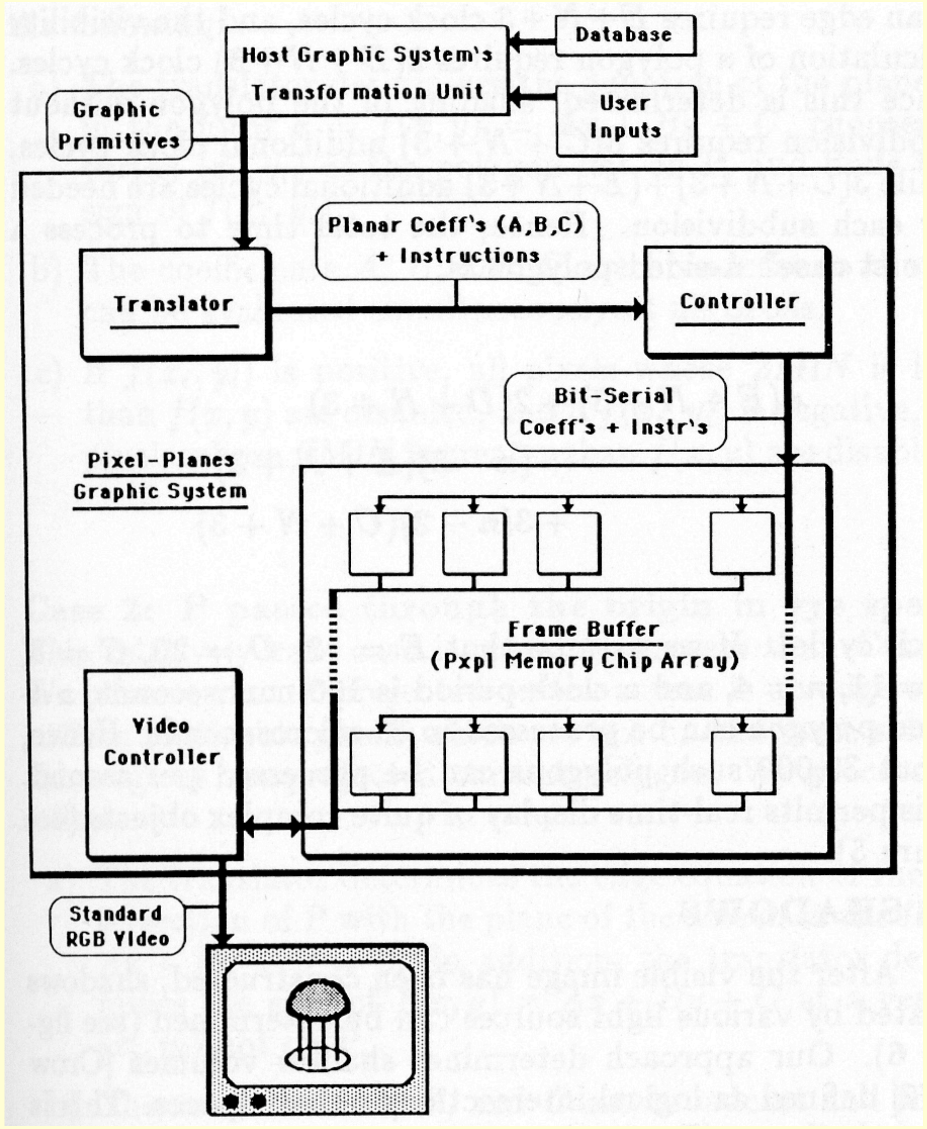
Pixel-Planes 5

- Message-passing
- ~40 i860 CPUs
- ~20 128x128 SIMD arrays (~80 tiles/screen)

Pixel-Flow

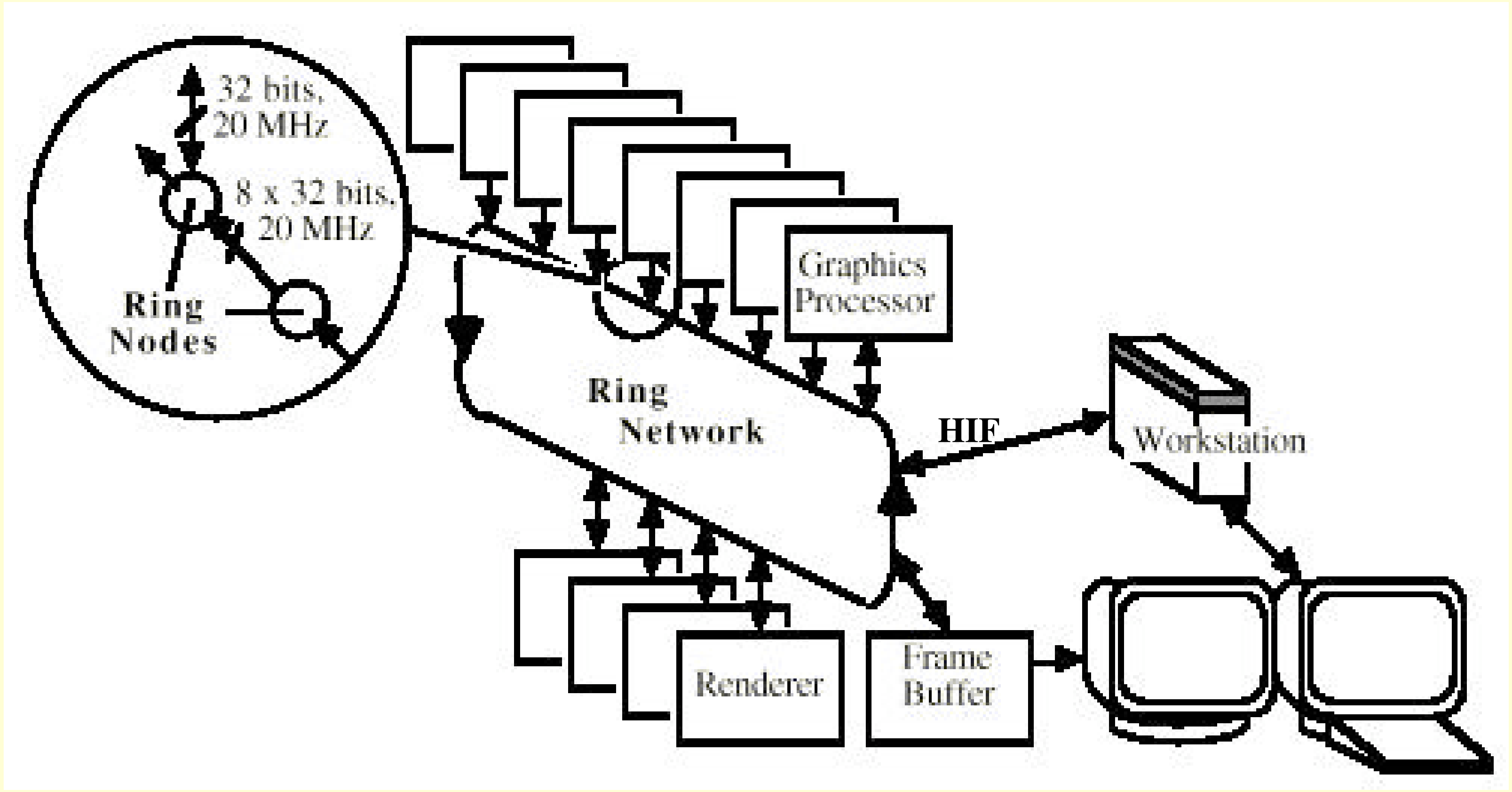
- Message-passing
- ~35 nodes, each with
  - 2 HP-PA 8000 CPUs
  - 128x64 SIMD array (~160 tiles/screen)

Pixel-Planes 4


Database
Host Graphic System's Transformation Unit
User Inputs
Graphic Primitives
Planar Coeff's (A,B,C) + Instructions
Translator
Controller
Bit-Serial Coeff's + Instr's
Pixel-Planes Graphic System
Frame Buffer (Pxpl Memory Chip Array)
Video Controller
Standard RGB Video

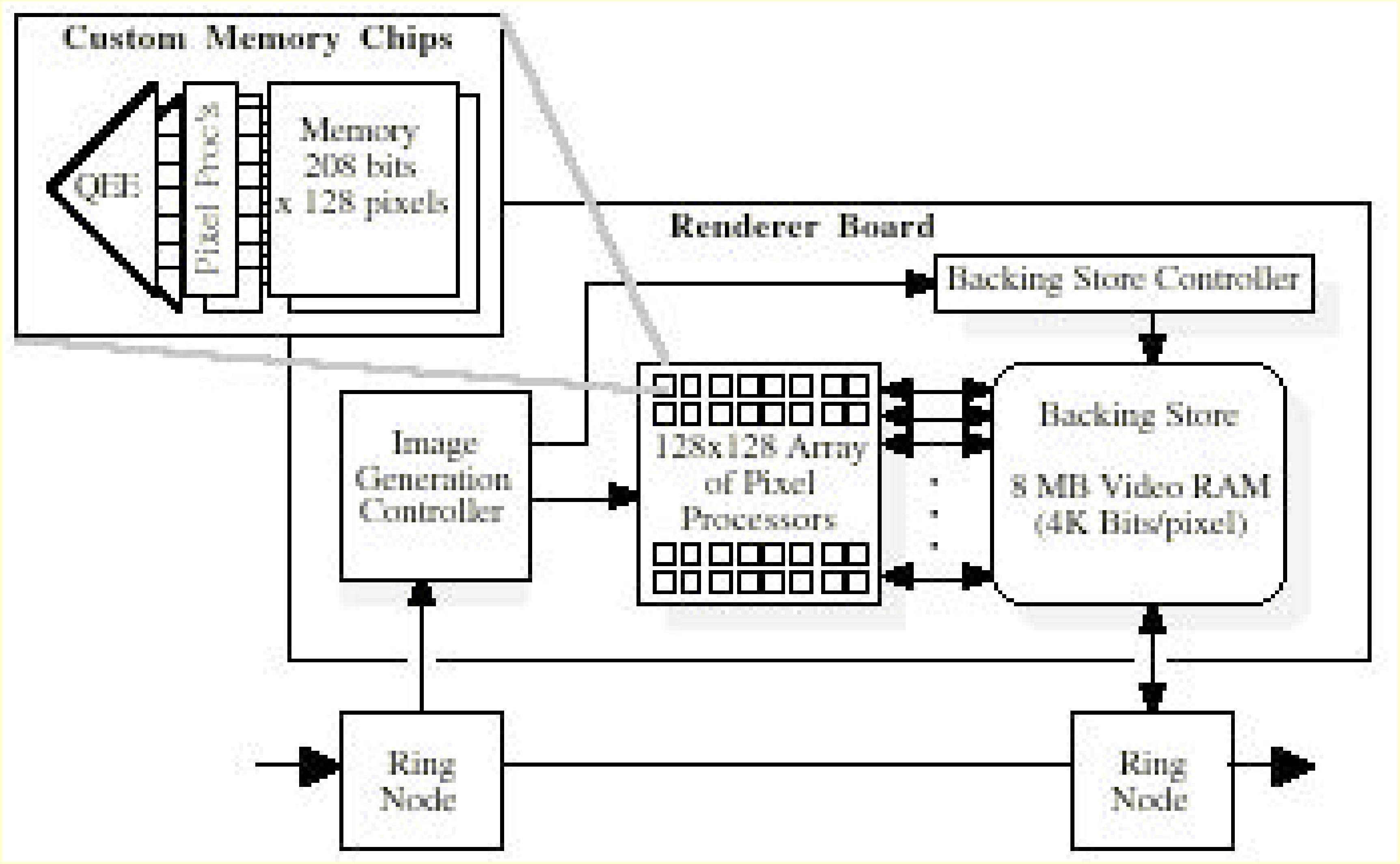
Fuchs, et al., "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes", SIGGRAPH 1985

Pixel-Planes 5



Fuchs, et al., "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor Enhanced Memories", SIGGRAPH 89

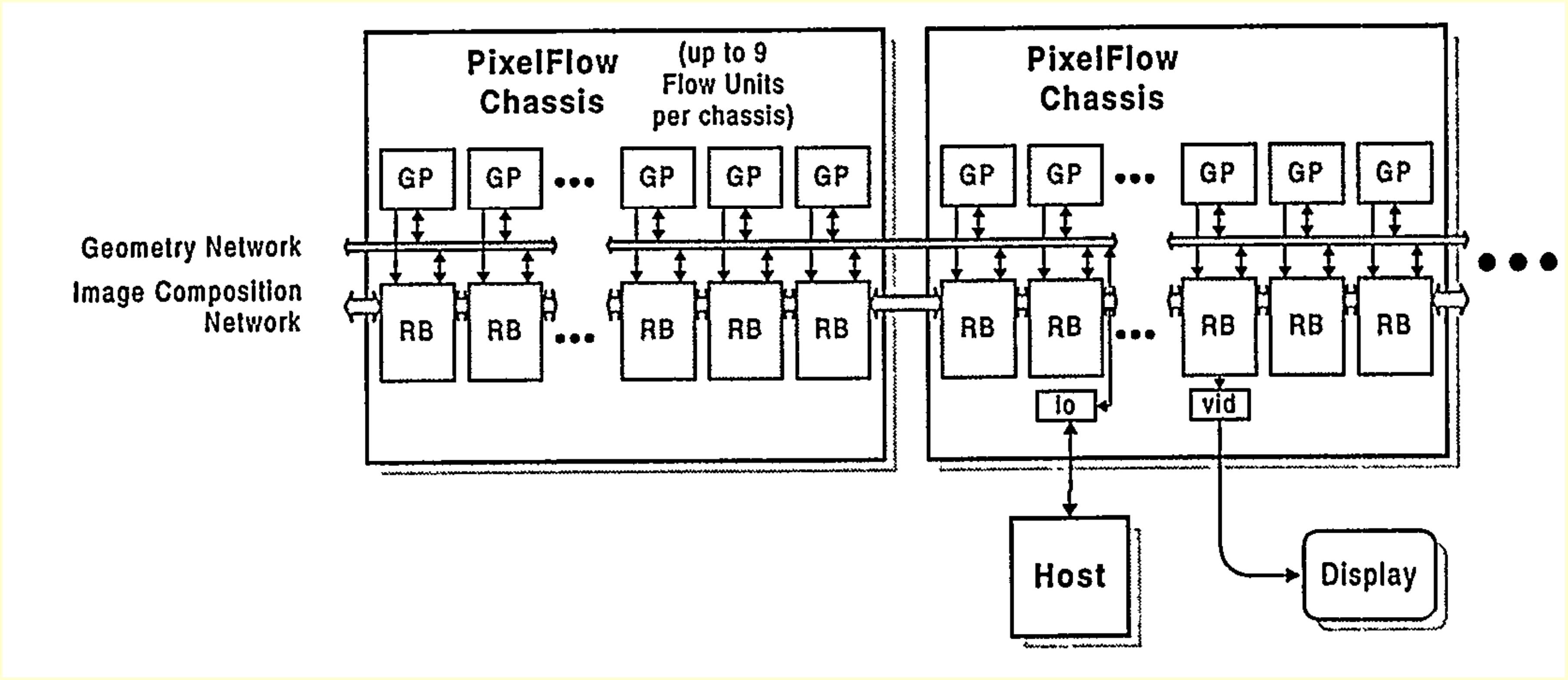
Pixel-Planes 5



Custom Memory Chips
QEE
Pixel Proc's
Memory 208 bits x 128 pixels
Renderer Board
Backing Store Controller
Image Generation Controller
128x128 Array of Pixel Processors
Backing Store
8 MB Video RAM (4K Bits/pixel)
Ring Node
Ring Node

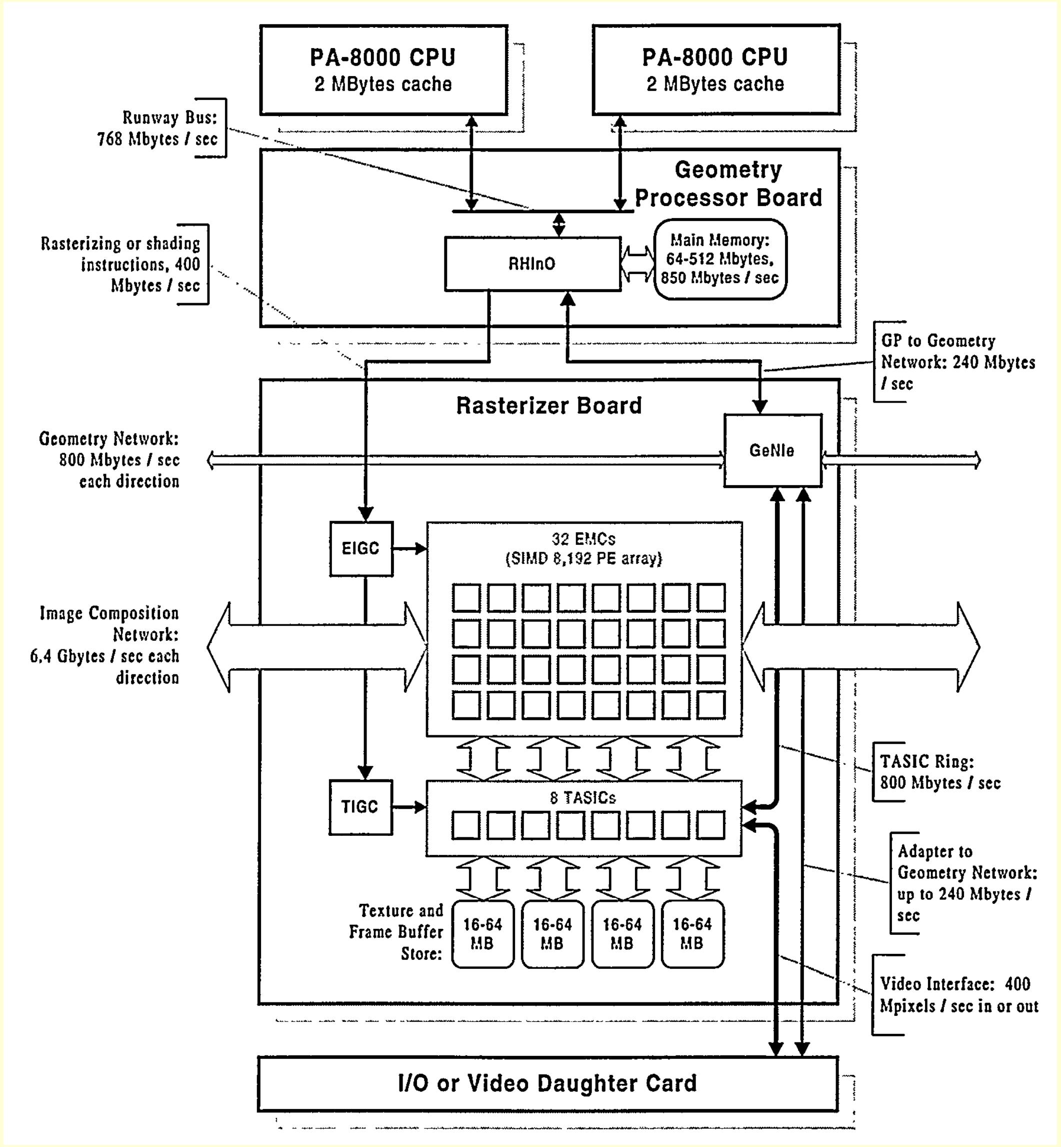
Fuchs, et al., "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor Enhanced Memories", SIGGRAPH 89

Pixel-Flow



Eyles, et al., "PixelFlow: The Realization", Graphics Hardware 1997

Pixel-Flow


PA-8000 CPU
2 MBytes cache
PA-8000 CPU
2 MBytes cache
Runway Bus: 768 Mbytes / sec
Geometry Processor Board
RHInO
Main Memory: 64-512 Mbytes, 850 Mbytes / sec
Rasterizing or shading instructions, 400 Mbytes / sec
GP to Geometry Network: 240 Mbytes / sec
Rasterizer Board
GeNIe
Geometry Network: 800 Mbytes / sec each direction
EIGC
32 EMCs (SIMD 8,192 PE array)
Image Composition Network: 6.4 Gbytes / sec each direction
TASIC Ring: 800 Mbytes / sec
TIGC
8 TASICs
Adapter to Geometry Network: up to 240 Mbytes / sec
Texture and Frame Buffer Store:
16-64 MB
16-64 MB
16-64 MB
16-64 MB
Video Interface: 400 Mpixels / sec in or out
I/O or Video Daughter Card

Eyles, et al., "PixelFlow: The Realization", Graphics Hardware 1997