Explicit Parallel Platforms

- Explicit Parallelism, Task Parallelism
- Mostly in the order of >> 10
- Requires active involvement of the programmer and /or compiler (no free lunch)
- Requires additional program constructs
- Requires new programming paradigms

Amdahl's Law

Given a computation of which a fraction of **q** cannot be parallelized.

Then the **maximal speedup** with **P** processors is limited to:

$$S_P = T / (q T + (1-q) T / P)$$

with **T** the sequential time.

So, with q = 0.01 max speedup <= 100 regardless of P

q = 0.05 max speedup <= 20 regardless of P

q = 0.10 max speedup <= 10 regardless of P

Flynn's Taxonomy

- Processing units in parallel computers either operate under the **centralized control** of a single control unit or work **independently**.
- If there is a **single control unit** that dispatches the same instruction to various processors (that work on different data), the model is referred to as single instruction stream, multiple data stream (**SIMD**).
- If **each processor has its own control control unit**, each processor can execute different instructions on different data items. This model is called multiple instruction stream, multiple data stream (**MIMD**).

SIMD Processors

- The **same instruction on different processors (functional units)**. Execution is **tightly synchronized**.
- Some of the earliest parallel computers such as the **Illiac IV, MPP, DAP, CM-2, and MasPar MP-1** belonged to this class of machines.
- Variants of this concept have found use in **co-processing** units such as the **MMX** units in Intel processors and **GPU**'s like NVIDIA.
- SIMD relies on the **regular structure of computations** (such as those in image processing).
- It is often necessary to selectively turn off operations on certain data items. For this reason, most SIMD programming paradigms allow for an ``**activity mask**'', which determines if a processor should participate in a computation or not.

MIMD Processors

- In contrast to SIMD processors, **MIMD processors** can execute **different programs on different processors**.
- A variant of this, called single program multiple data streams (**SPMD**) executes **the same program on different processors**, but allows for different instructions to be executed on each processor (if/case stmts)
- It is easy to see that **SPMD and MIMD are closely related** in terms of programming flexibility and underlying architectural support.
- Examples of such platforms include current generation **Sun Ultra Servers, SGI Origin Servers, multiprocessor PCs, workstation clusters, and the IBM SP**.

SIMD-MIMD Comparison

- **SIMD** computers require **less hardware** than MIMD computers (single control unit).
- However, since **SIMD** processors are tightly synchronized and therefore specially designed, they tend to be **expensive** and have **long design cycles**. (NVIDIA forms an exception to this, WHY?)
- In contrast, platforms supporting the **MIMD/SPMD paradigm** **can be built from inexpensive off-the-shelf components** with relatively little effort in a short amount of time.
- **Not all applications are naturally suited to SIMD processors**.
- **MIMD/SPMD** platforms have relatively **large communication overhead**, therefore ask for **large grain parallelism**.

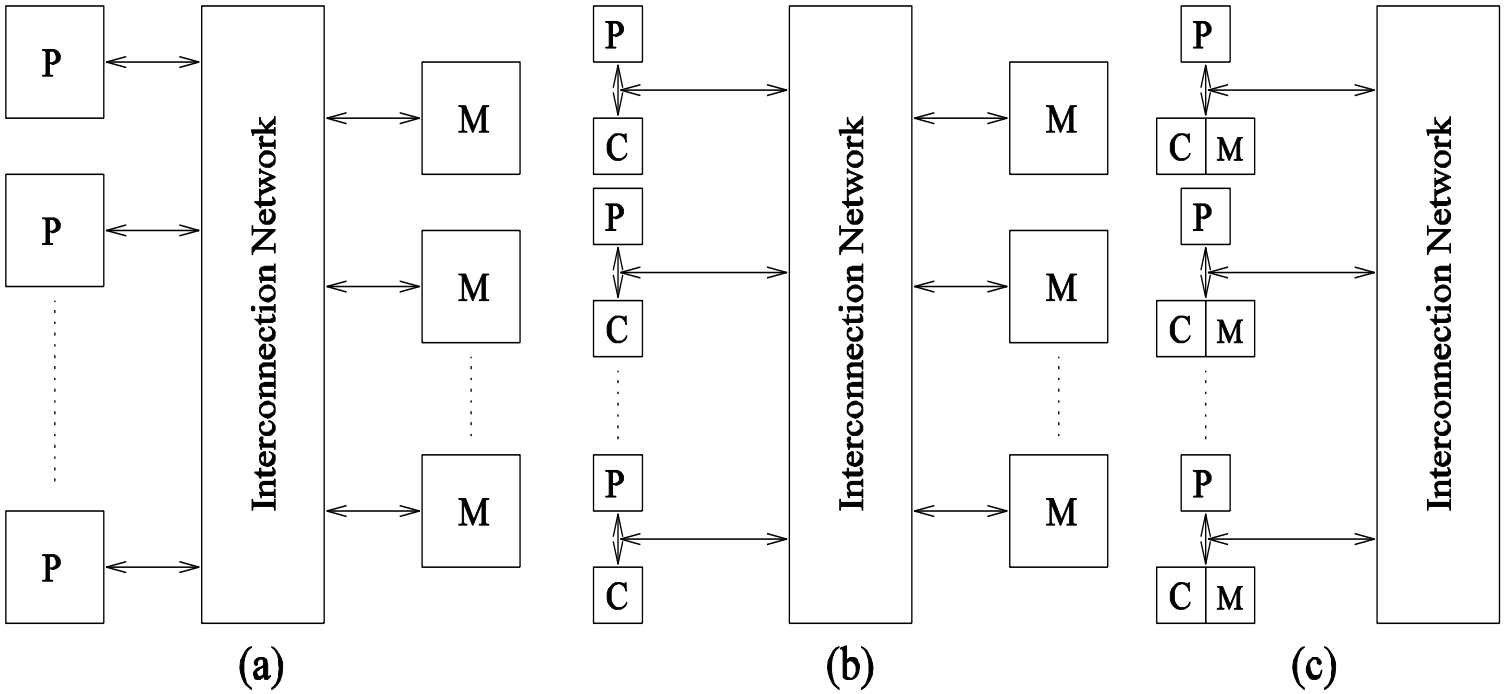
Communication Model of Parallel Platforms

- There are two primary forms of data exchange between parallel tasks - accessing a **shared data space** and **exchanging messages**.
- Platforms that provide a shared data space are called **shared-address-space machines** or multiprocessors.
- Platforms that support messaging are also called **message passing platforms** or multi-computers.

Shared-Address-Space Platforms

- Part (or all) of the memory is accessible to all processors.
- Processors interact by modifying data objects stored in this shared-address-space.
- If the time taken by a processor to access any memory word in the system global is identical, the platform is classified as a **uniform memory access machine** (**UMA**). If this is not the case then we refer to a **non-uniform memory access machine** (**NUMA**).

NUMA and UMA Shared-Address-Space Platforms



(a) Uniform-memory access shared-address-space computer;

(b) Uniform-memory-access shared-address-space computer with caches and memories;

(c) Non-uniform-memory-access shared-address-space computer with local memory only.

Programming Consequences

- In contrast to UMA platforms, **NUMA machines require locality** from underlying algorithms for performance.
- **Programming Shared-Address-Space platforms is easier** since reads and writes are implicitly visible to other processors.
- However, **read-write data to shared data must be coordinated**.
- Caches in such machines require coordinated access to multiple copies. This leads to the **cache coherence** problem.
- A weaker model of these machines provides an address map, but not coordinated access. These models are called **non cache coherent shared address space machines**.

Message-Passing Platforms

- These platforms comprise of a set of processors and their **own (exclusive) memory**.
- Naturally examples are clustered workstations and non-shared-address-space multi-computers.
- These platforms are programmed using (variants of) **send** and **receive primitives**.
- Libraries such as **MPI (Message Passing Interface)** for Distributed Memory Platforms and **PVM (Parallel Virtual Machine)** for Parallel and Shared Memory Platforms provide such primitives. **OpenMP** is an **API** for Shared Memory Platforms based on multithreading.

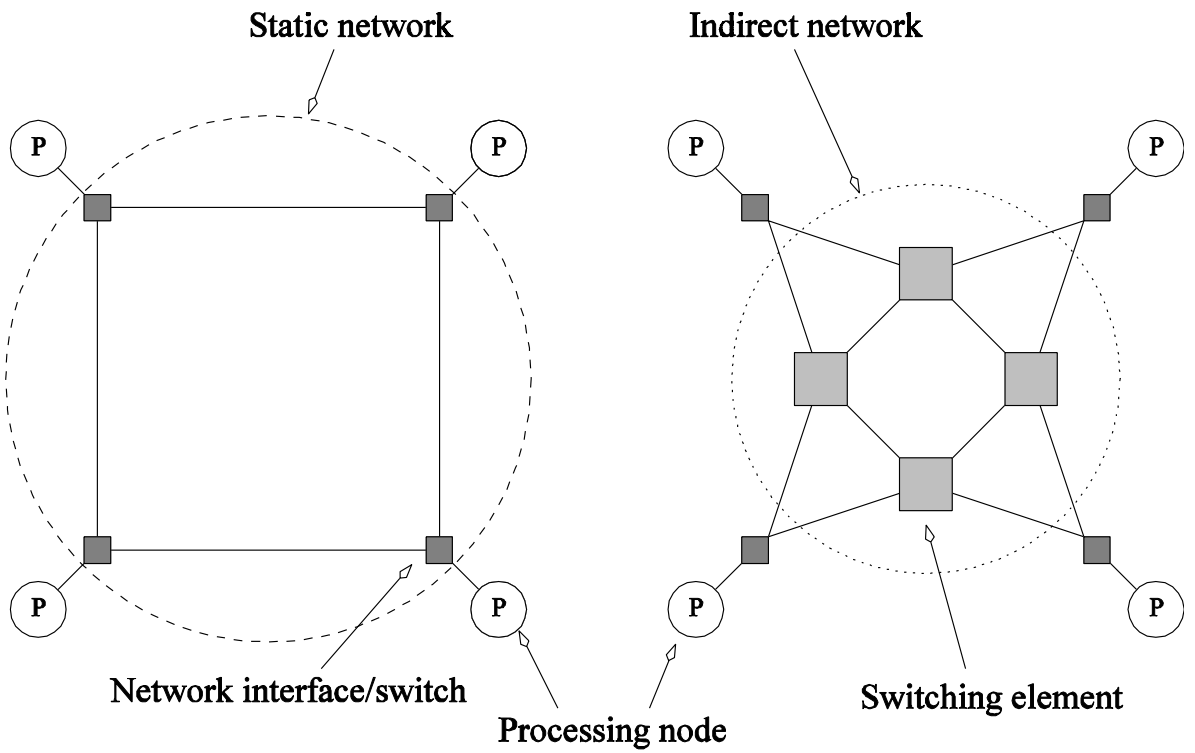
Message Passing
vs.
Shared Memory Platforms

- **Message passing requires little hardware support, other than a network.**
- **Shared Memory platforms can easily emulate message passing. The reverse is more difficult to do (in an efficient manner).**

Interconnection Networks for Parallel Computers

- Interconnection networks carry data between processors and to memory.
- Interconnects are made of switches and links (wires, fiber).
- Interconnects are classified as **static** or **dynamic**.
- **Static networks** consist of **point-to-point** communication **links** among processing nodes and are also referred to as **direct** networks.
- **Dynamic networks** are built using **switches** and communication links. Dynamic networks are also referred to as **indirect** networks.

Static and Dynamic Interconnection Networks



Classification of interconnection networks: (a) **a static network**; and (b) **a dynamic network**.

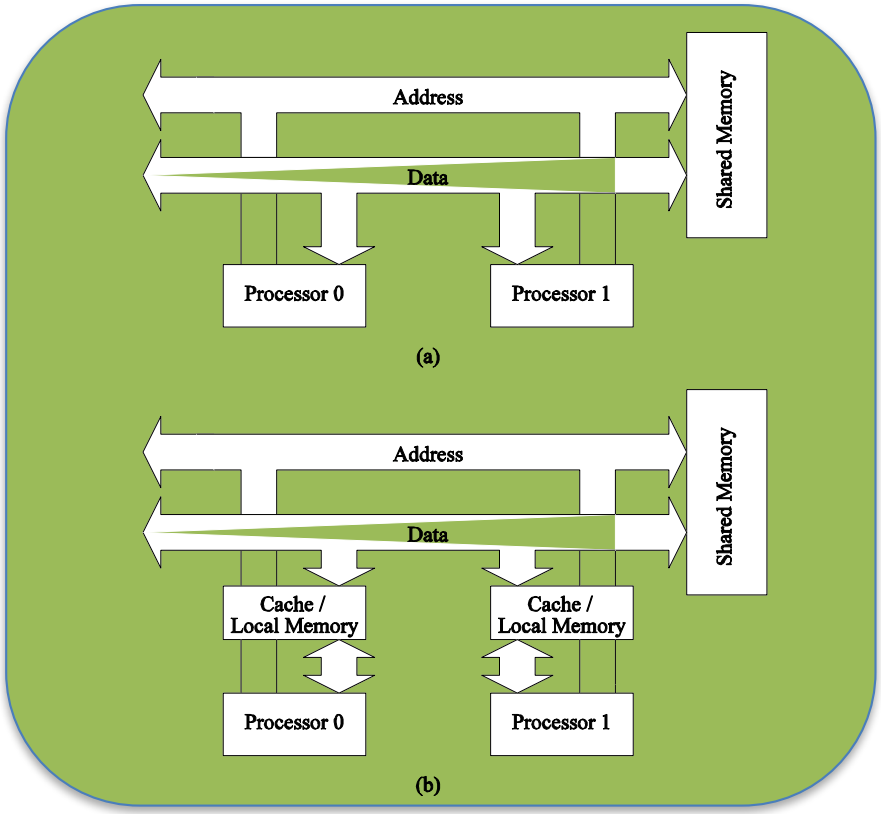
Network **Topologies**

- A **variety of network topologies** have been proposed and implemented.
- These topologies tradeoff performance for cost.
- **Commercial machines often implement hybrids of multiple topologies for reasons of packaging, cost, and available components**.

Network Topologies: Buses

- Some of the simplest and earliest parallel machines used **buses**.
- All processors access a **common bus** for exchanging data.
- The **distance** between any two nodes is **O(1)** in a bus. The bus also provides a convenient broadcast media.
- However, the **bandwidth of the shared bus is a major bottleneck**.
- Typical bus based machines are **limited to dozens of nodes**. **Sun (Cray) servers** and **Intel Core based shared-bus multiprocessors** are examples of such architectures.

Network Topologies: Buses

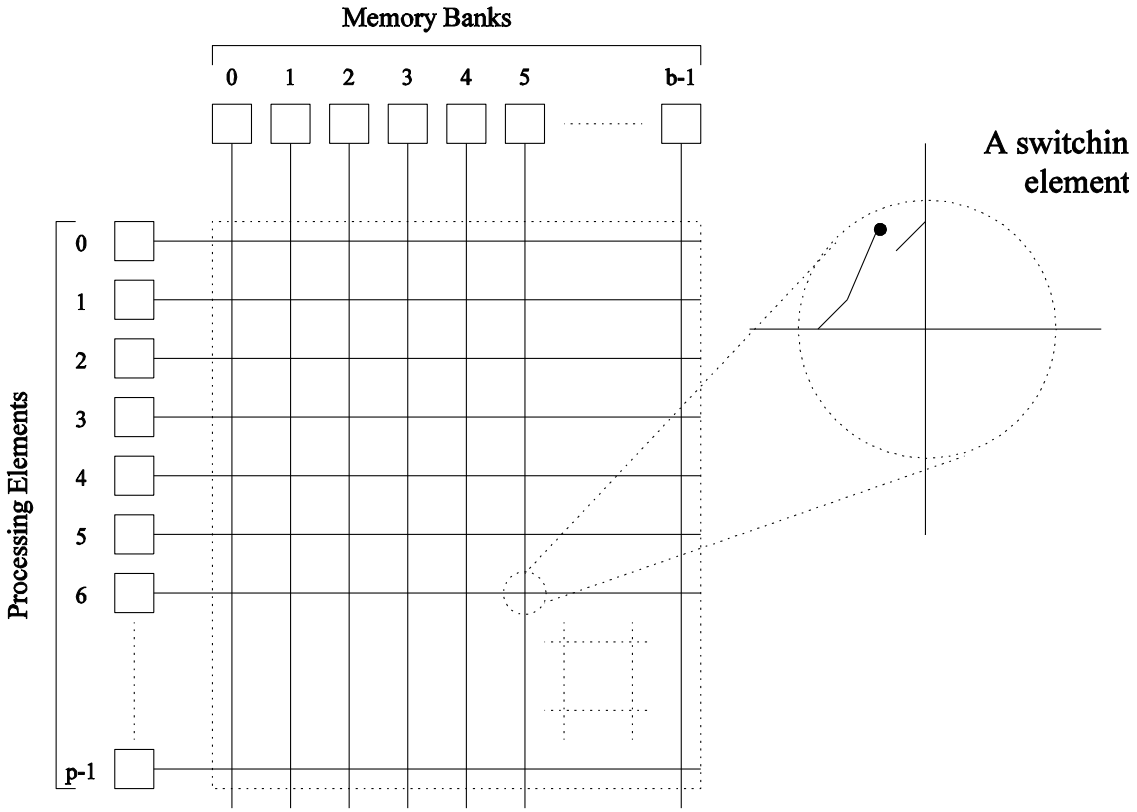


Bus-based interconnects (a) with no local caches; (b) with local memory/caches.

Since much of the data accessed by processors is local to the processor, a local memory can improve the performance.

Network Topologies: Crossbars

A crossbar network uses an *p*×*m* grid of switches to connect *p* inputs to m outputs in a **non-blocking** manner.



A completely non-blocking crossbar network connecting *p* processors to b memory banks.

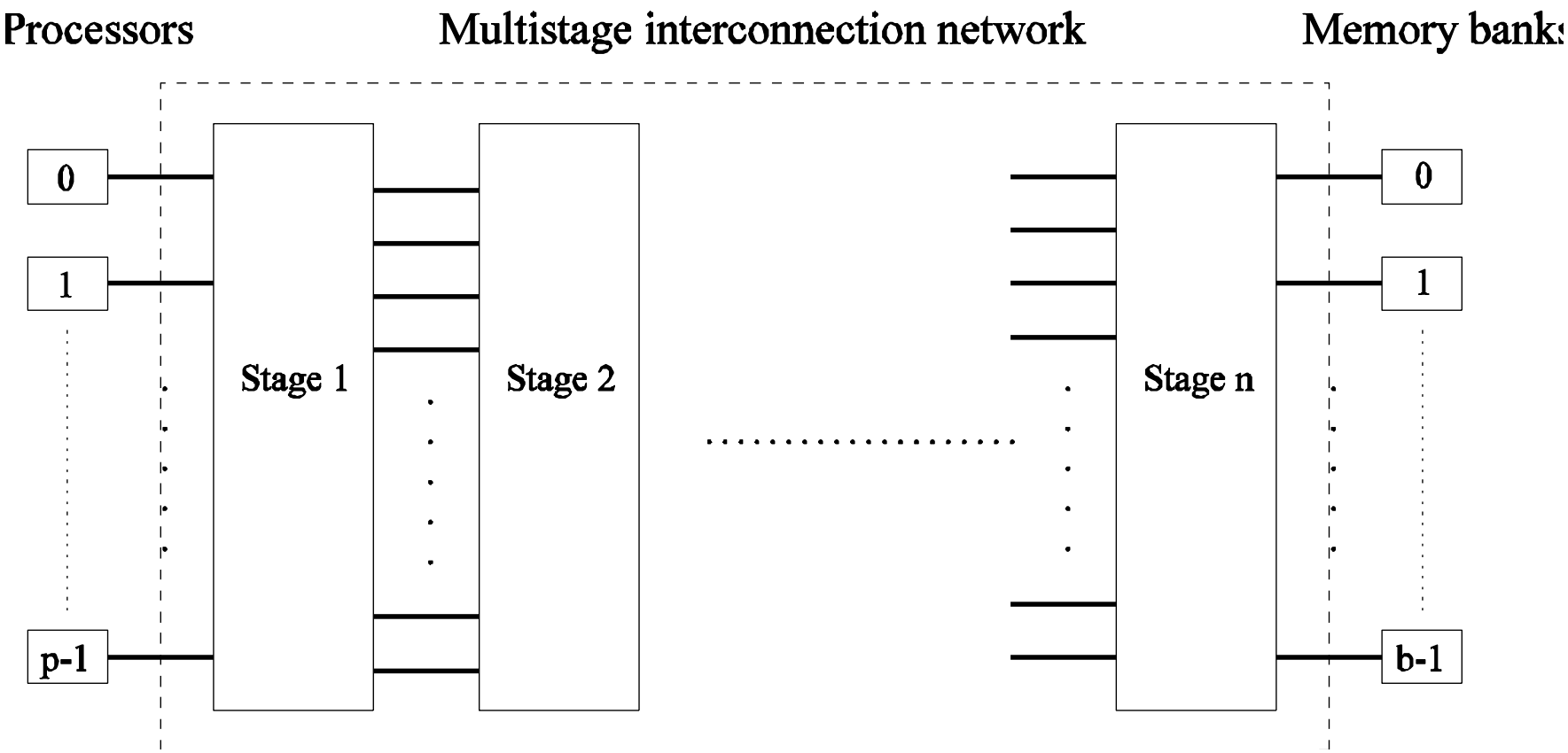
Network Topologies: Crossbars

- The cost of a crossbar of p processors grows as $O(p^2)$.
- This is generally **difficult to scale** for large values of p.
- Examples of machines that employ crossbars include the **Sun Ultra HPC 10000** and the **Fujitsu VPP500**.

Network Topologies: Multistage Networks

- **Crossbars** have **excellent performance scalability** but **poor cost scalability**.
- **Buses** have **excellent cost scalability**, but **poor performance scalability**.
- **Multistage interconnects** strike a **compromise** between these extremes.

Network Topologies: Multistage Networks



The schematic of a typical multistage interconnection network.

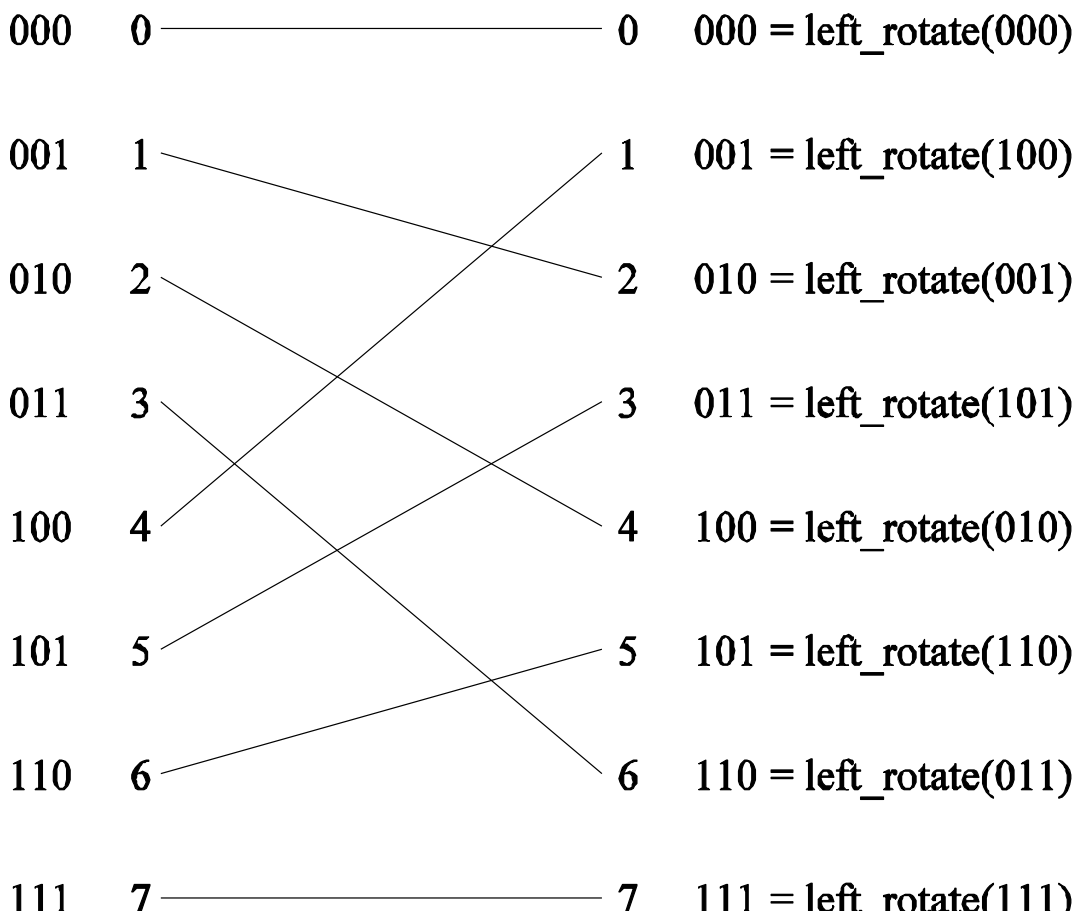
Network Topologies: Multistage Omega Network

- One of the most commonly used multistage interconnects is the **Omega network**.
- This network consists of **`log p`** stages, where **`p`** is the number of inputs/outputs.
- At each stage, input *i* is connected to output *j*:

$$j = \begin{cases} 2i, & 0 \leq i \leq p/2 - 1 \\ 2i + 1 - p, & p/2 \leq i \leq p - 1 \end{cases}$$

Network Topologies: **Omega Network**

Each stage of the Omega network implements a **perfect shuffle** as follows:



A perfect shuffle interconnection for eight inputs and outputs.

Network Topologies:
Multistage Omega Network

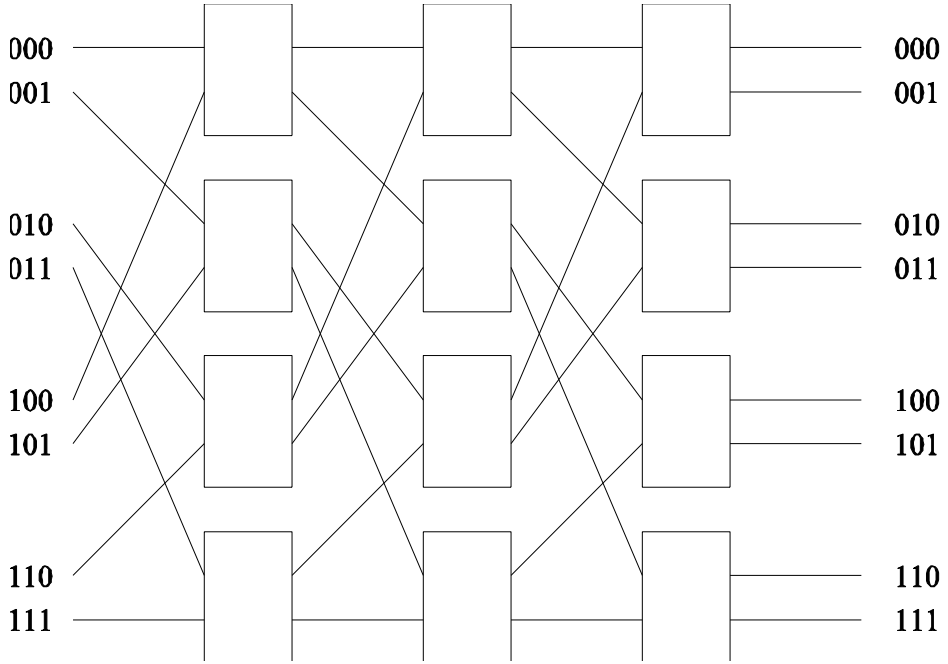
- The perfect shuffle patterns are connected using 2×2 switches.
- The switches operate in two modes: **crossover** or **pass-through** (switch bit position or not).

(a) (b)

Two switching configurations of the 2 × 2 switch:
(a) **Pass-through**; (b) **Cross-over**.

Network Topologies:
Multistage Omega Network

A complete Omega network with the perfect shuffle interconnects and switches can be illustrated as follows:

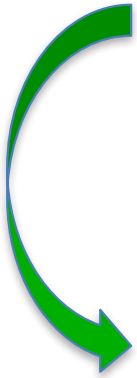


A complete omega network Ω_8 connecting 8 inputs and eight outputs.

An omega network Ω_n has `n/2 * log n` switching nodes (`log n` stages).

Routing in Omega Network

Connecting $X_1X_2X_3X_4$ to $Y_1Y_2Y_3Y_4$



$X_1X_2X_3X_4$ -> $X_2X_3Y_4X_1$ -> $X_2X_3Y_4Y_1$ -> $X_3Y_4Y_1X_2$ ->

PS Sw PS Sw

$X_3Y_4Y_1Y_2$ -> $Y_4Y_1Y_2X_3$ -> $Y_4Y_1Y_2Y_3$ -> $Y_1Y_2Y_3Y_4$ ->

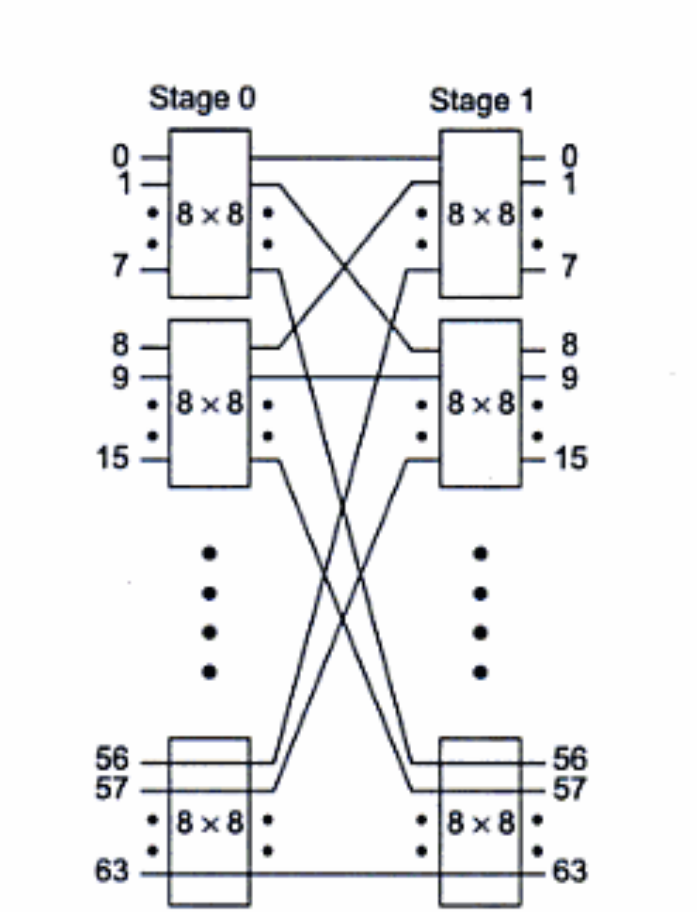
PS Sw PS Sw

$Y_1Y_2Y_3Y_4$

Note that one can also skip the first **PS** and start by substituting X_4 by Y_4, then X_1 by Y_1, etc.

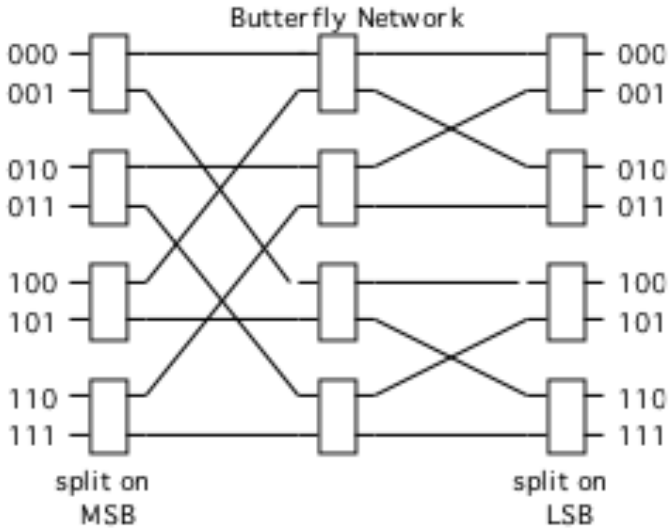
So **the Omega network is equivalent to the Omega network without the first Perfect Shuffle!!!!!**

Network Topologies:
the Butterfly Network



(a) A two-stage 64 × 64 Butterfly switch network built with 16 8 × 8 crossbar switches and eight-way shuffle interstage connections

Two Stages;



A variation of The Omega network

In Fact: **The following networks are equivalent**

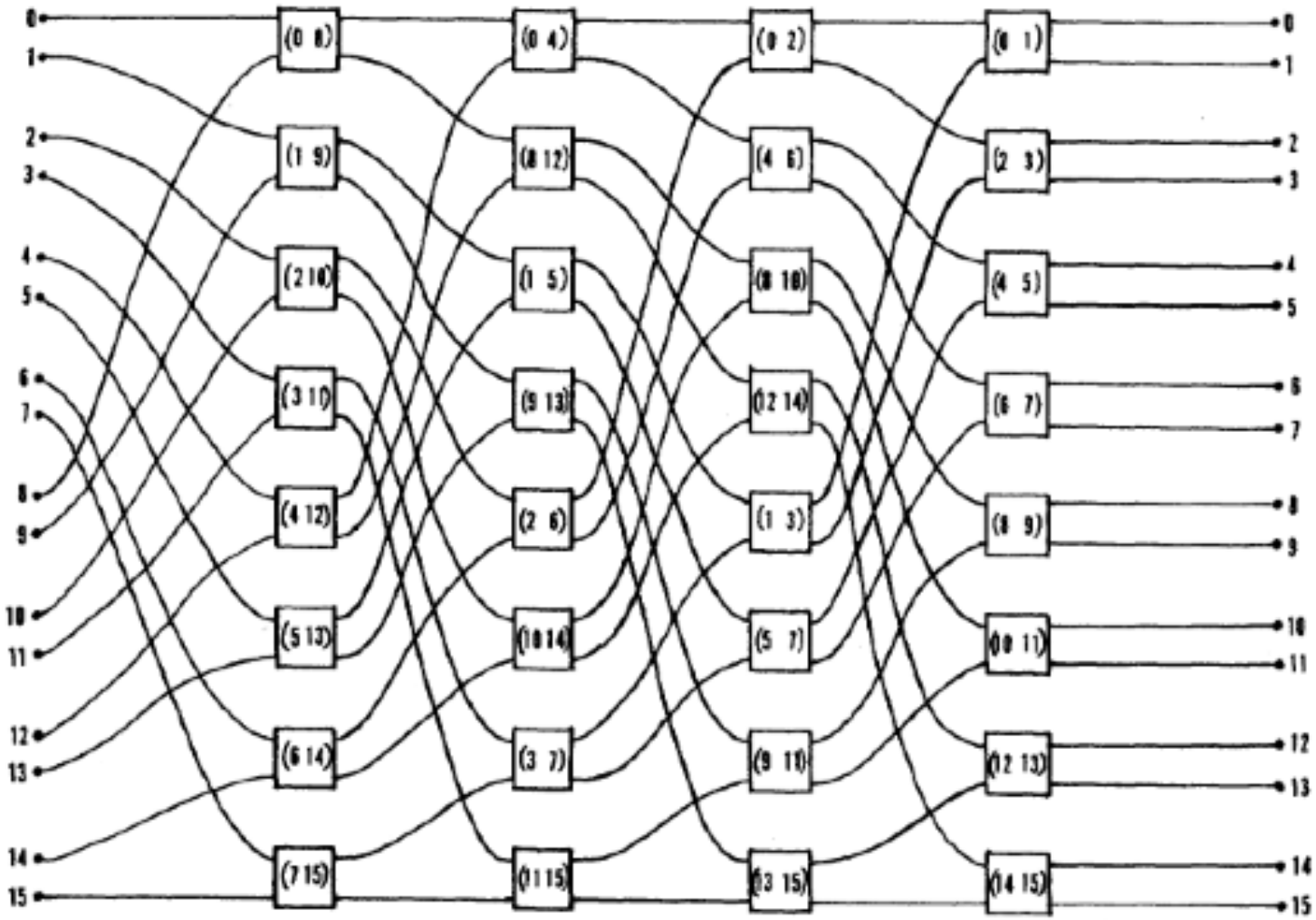


Fig. 1. A 16-input Omega network.

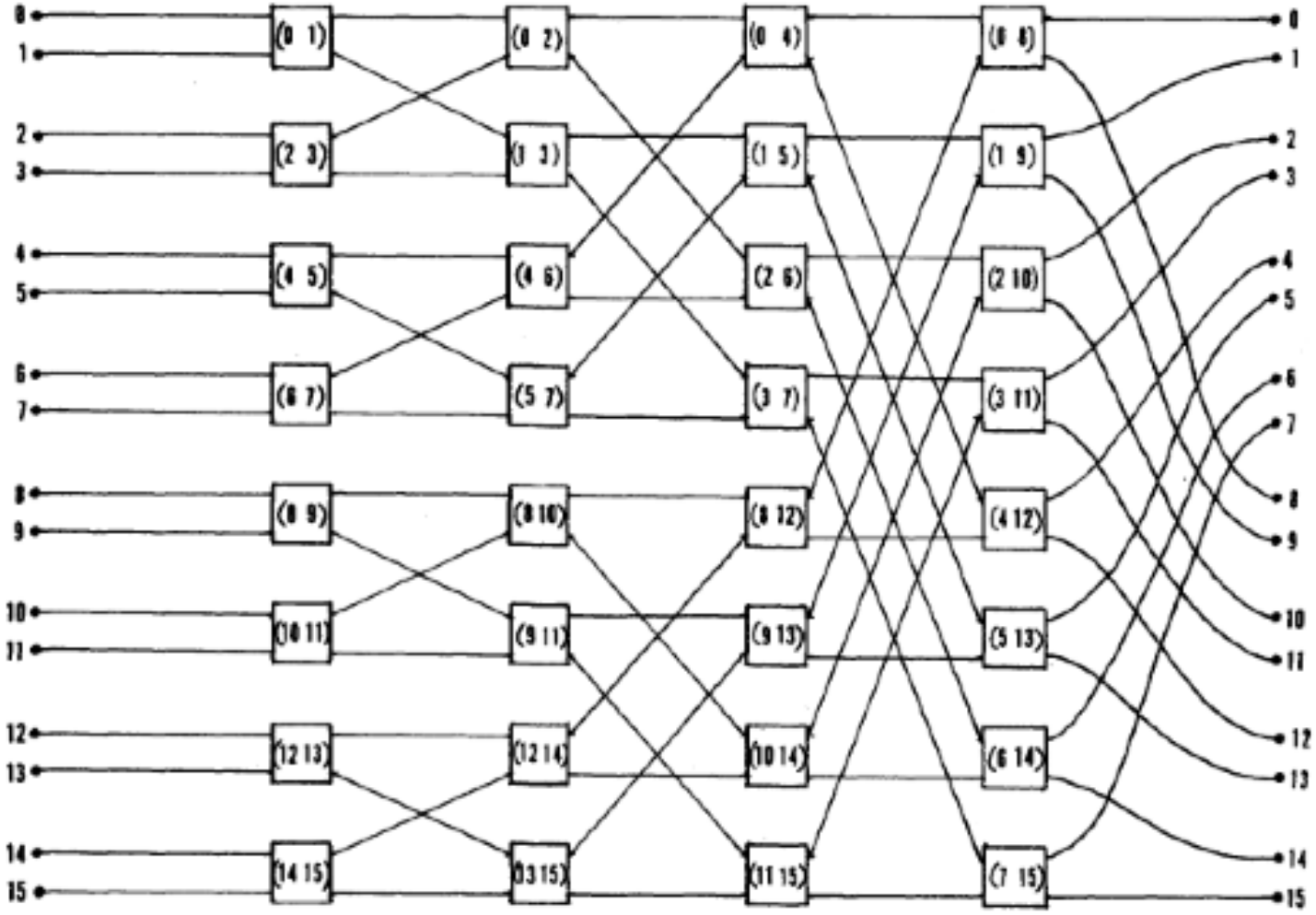


Fig. 2. An indirect binary 4-cube network.

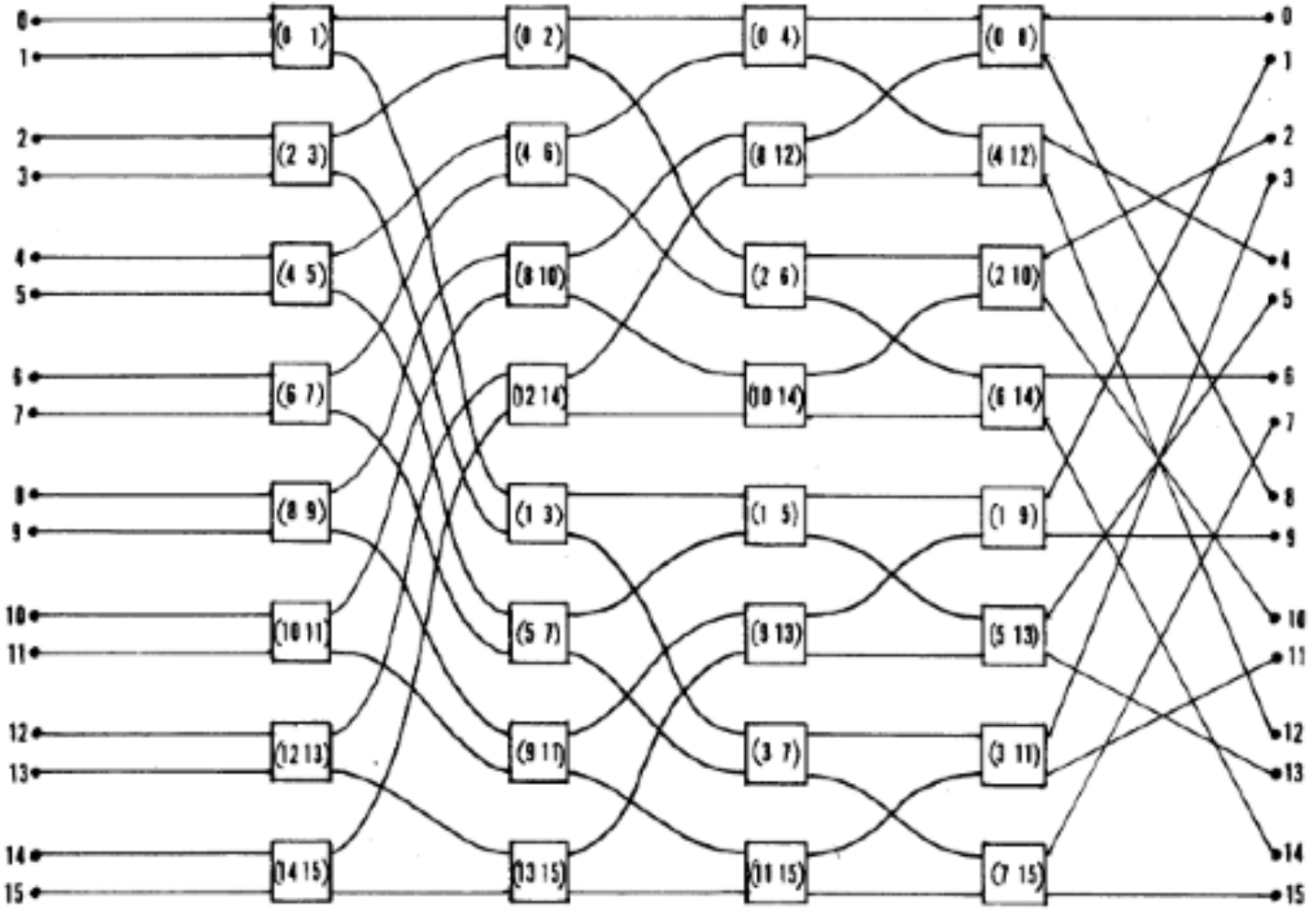


Fig. 3. A 16-input *R*-network.

Relationship with FFT

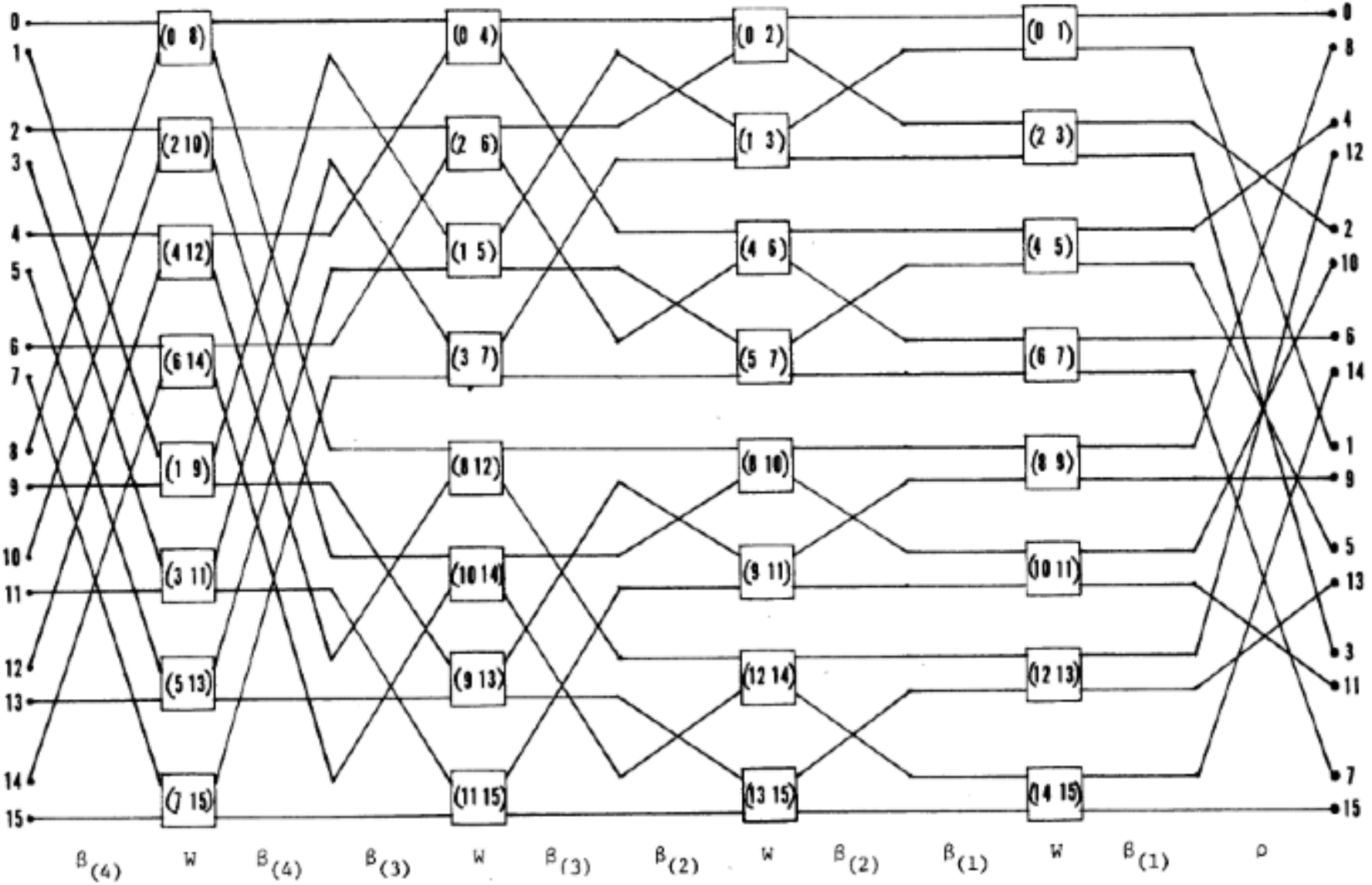


Fig. 4. Traditional FFT algorithm structure.

Routing Properties

- **Clos/Benes showed that $R_N R_N^{-1}$ can realize any permutation**. Proof is based on Hall's marriage theorem:

FYI

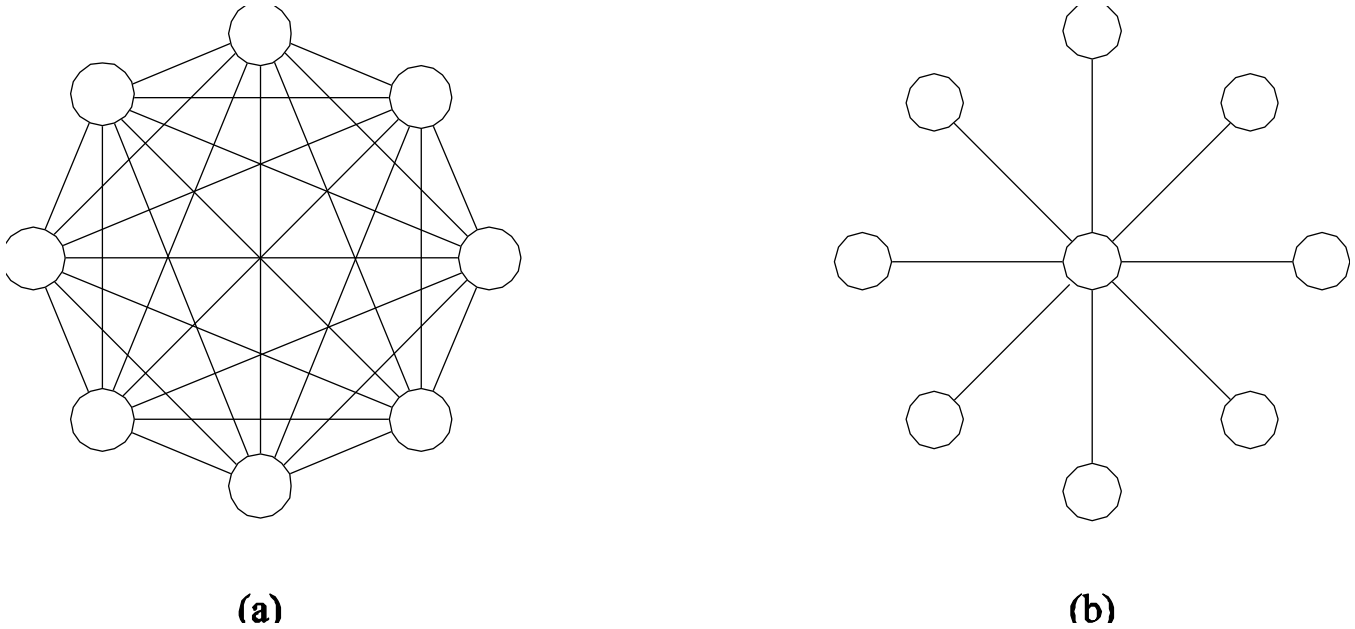
Imagine two groups; one of *n* men, and one of *n* women. For each woman, there is a subset of the men, any one of which she would happily marry; and a man would be happy to marry a woman who wants to marry him. Consider whether it is possible to pair up (in marriage) the men and women so that every person is happy. If we let A_i be the set of men that the *i*-th woman would be happy to marry, then the marriage theorem states that each woman can happily marry a man if and only for any subset of the women, the number of men whom at least one of the women would be happy to marry, be at least as big as the number of women in that subset. It is obvious that this condition is *necessary*, as if it does not hold, there are not enough men to share among the women. What is interesting is that it is also a *sufficient* condition.

- **Ω_N is equivalent with R_N^{-1}, so $\Omega_N^{-1}\Omega_N$ can also realize any permutation. Non-blocking!!!!!**
- **This is not the case for $\Omega_N\Omega_N$.**
- **$\Omega_N\Omega_N\Omega_N$ can also realize any permutations. Proof based on counting arguments, actual routing is very complicated**.

Network Topologies:
Completely Connected Network

- Each processor is connected to every other processor.
- The number of links in the network scales as $O(p^2)$.
- While the **performance scales very well**, the hardware is not realizable for large values of p.
- In this sense, **these networks are static counterparts of crossbars**.

Network Topologies: **Completely Connected** and **Star Connected Networks**



(a) A completely-connected network of eight nodes;
(b) a star connected network of nine nodes.

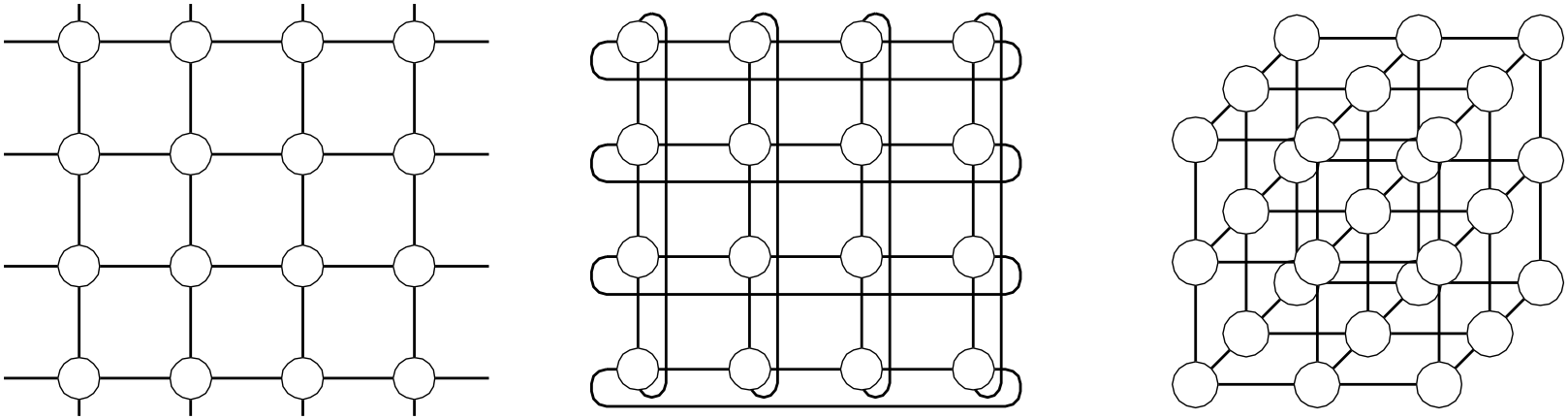
Network Topologies:
Star Connected Network

- Every node is connected only to a **common node at the center**.
- Distance between any pair of nodes is $O(1)$. However, **the central node becomes a bottleneck**.
- In this sense, **star connected networks are static counterparts of buses**.

Network Topologies: **Linear Arrays**, **Meshes**, and ***k-d* Meshes**

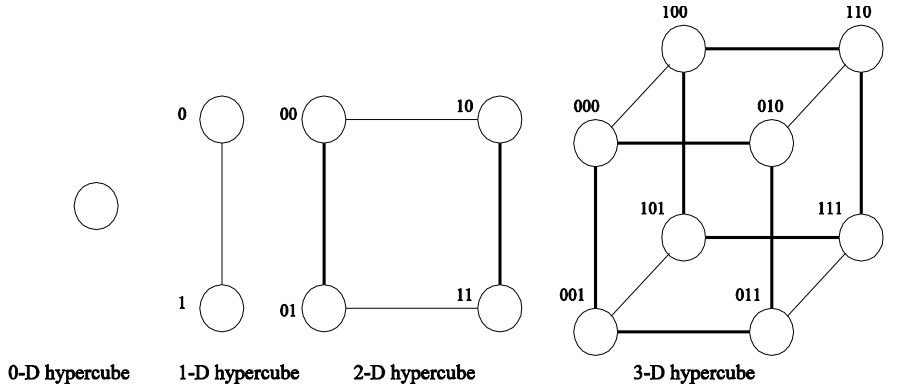
- In a **linear array**, each node has **two neighbors**, one to its left and one to its right. If the nodes at either end are connected, we refer to it as a 1-D **torus** or a **ring**.
- A generalization to **2 dimensions** has nodes with **4 neighbors**, to the north, south, east, and west.
- A further generalization to ***d* dimensions** has nodes with ***2d* neighbors**.
- A **special case** of a ***d*-dimensional mesh** is a **hypercube**. Here, ***d* = *log p***, where ***p* is the total number of nodes**.

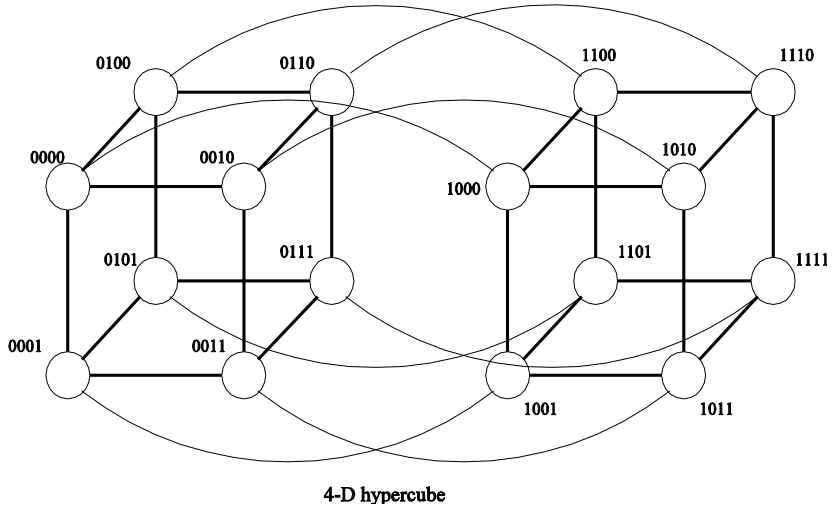
Network Topologies:
Two- and **Three Dimensional Meshes**



Two and three dimensional meshes: (a) **2-D mesh with no wraparound**; (b) **2-D mesh with wraparound link (2-D torus)**; and (c) **a 3-D mesh with no wraparound**.

Network Topologies: **Hypercubes** and their Construction



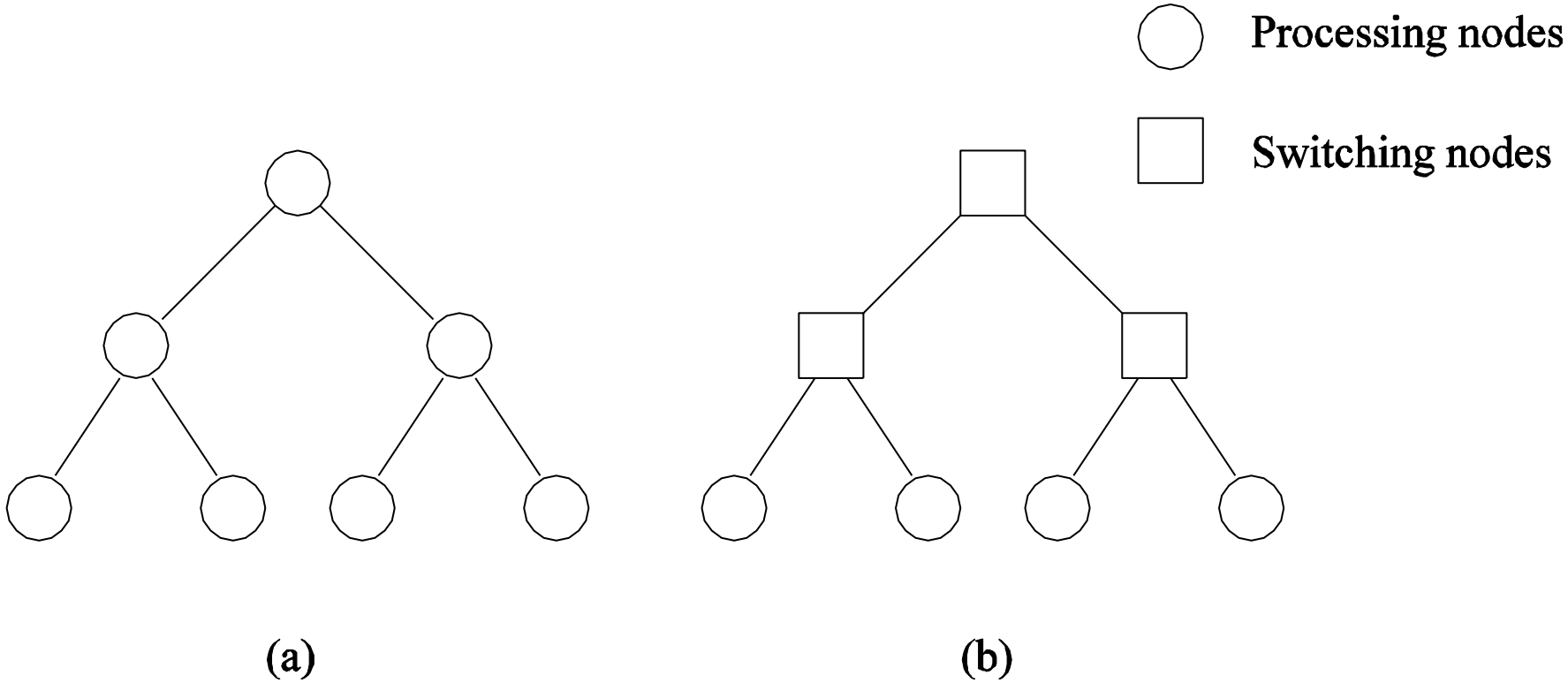


Construction of hypercubes from hypercubes of lower dimension.

Properties of Hypercubes

- The **distance between any two nodes** is at most ***log p***.
- Each node has ***log p*** **neighbors**.
- The **distance** between two nodes is given by **the number of bit positions at which the two nodes differ**, and therefore is limited to ***log p***.

Network Topologies: **Tree-Based Networks**

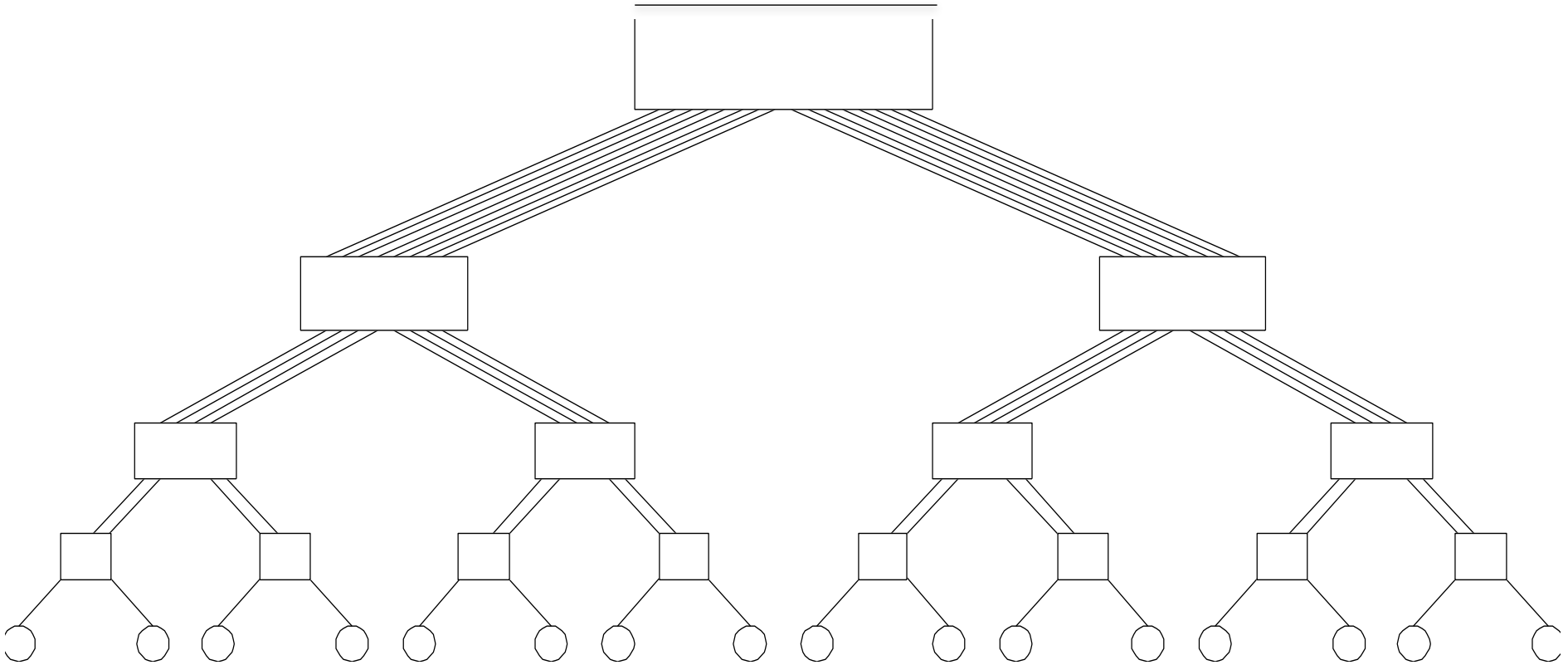


Complete binary tree networks: (a) **a static tree network**; and (b) **a dynamic tree network**.

Network Topologies: Tree Properties

- The distance between any two nodes is no more than ***2 log p***.
- Links **higher up** the tree potentially carry **more traffic** than those at the lower levels.
- For this reason, a variant called a **fat-tree**, fattens the links as we go up the tree.
- Trees can be laid out in **2D** with no wire crossings in **$\Omega(\sqrt{n} \log n)$** space area.

Network Topologies: **Fat Trees**



A fat tree network of 16 processing nodes. **Bandwidth each times doubles when going up one level**.

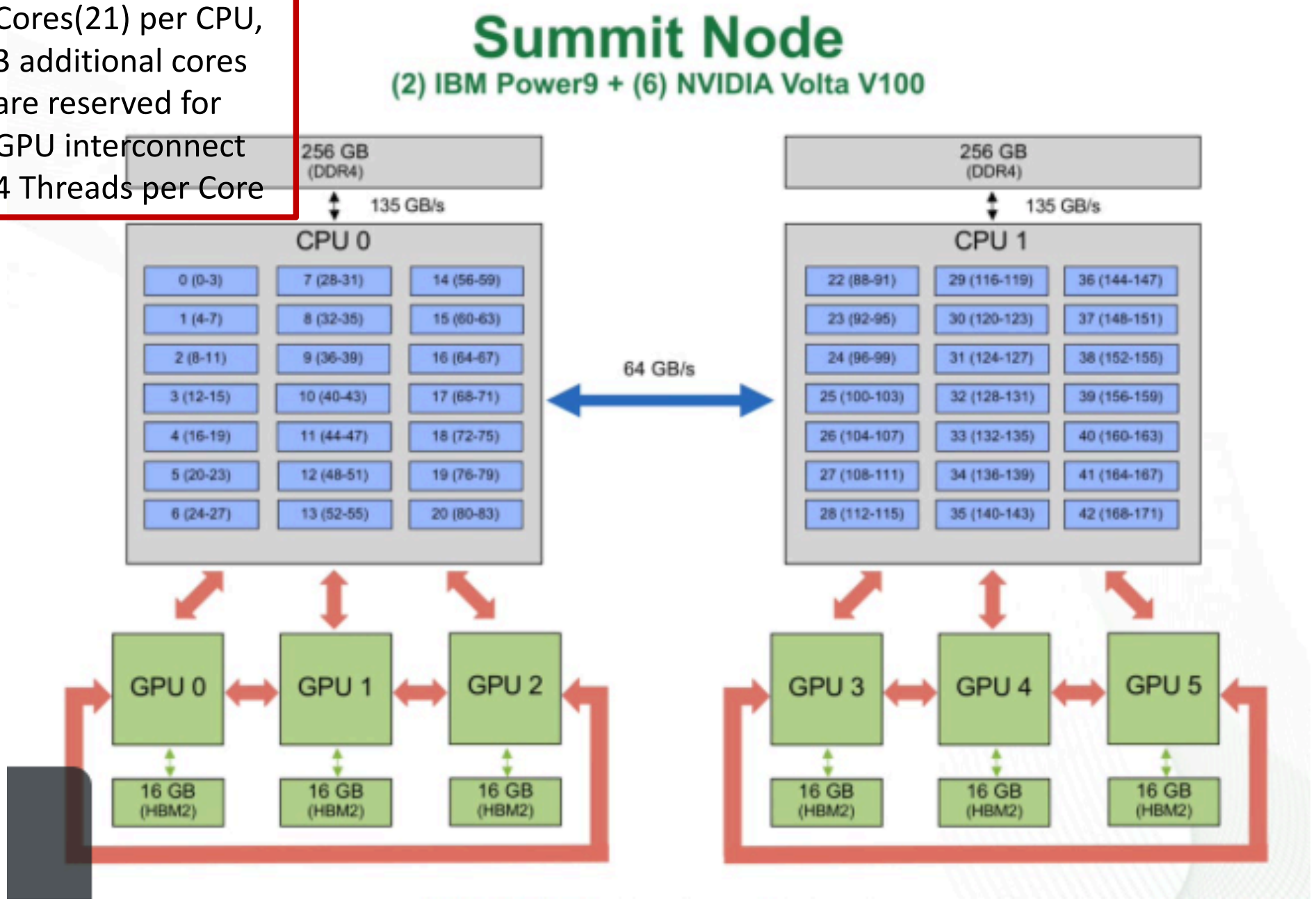
Routing Mechanisms for Interconnection Networks

How does one compute the route that a message takes from source to destination?

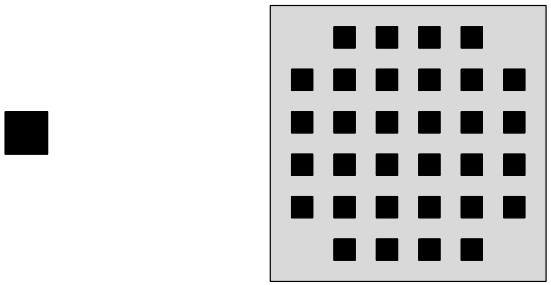
- Routing must prevent deadlocks - for this reason, we use dimension-ordered or e-cube routing.
- Routing must avoid hot-spots - for this reason, two-step routing is often used. In this case, a message from source *s* to destination *d* is first sent to a randomly chosen intermediate processor *i* and then forwarded to destination *d*.

Case Studies: SUMMIT (**4608 nodes**)

Cores(21) per CPU,
3 additional cores
are reserved for
GPU interconnect
4 Threads per Core

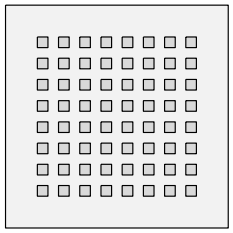


Case Studies:
The IBM Blue-Gene Architecture



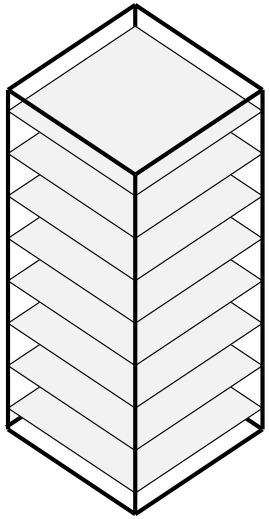
(a) CPU (1GF)

(b) Chip (32 GF)

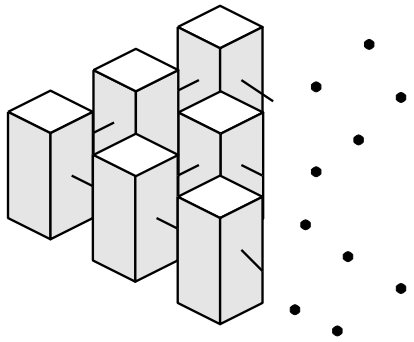


(c) Board (2 TF)

64*32 GF = 2 TF



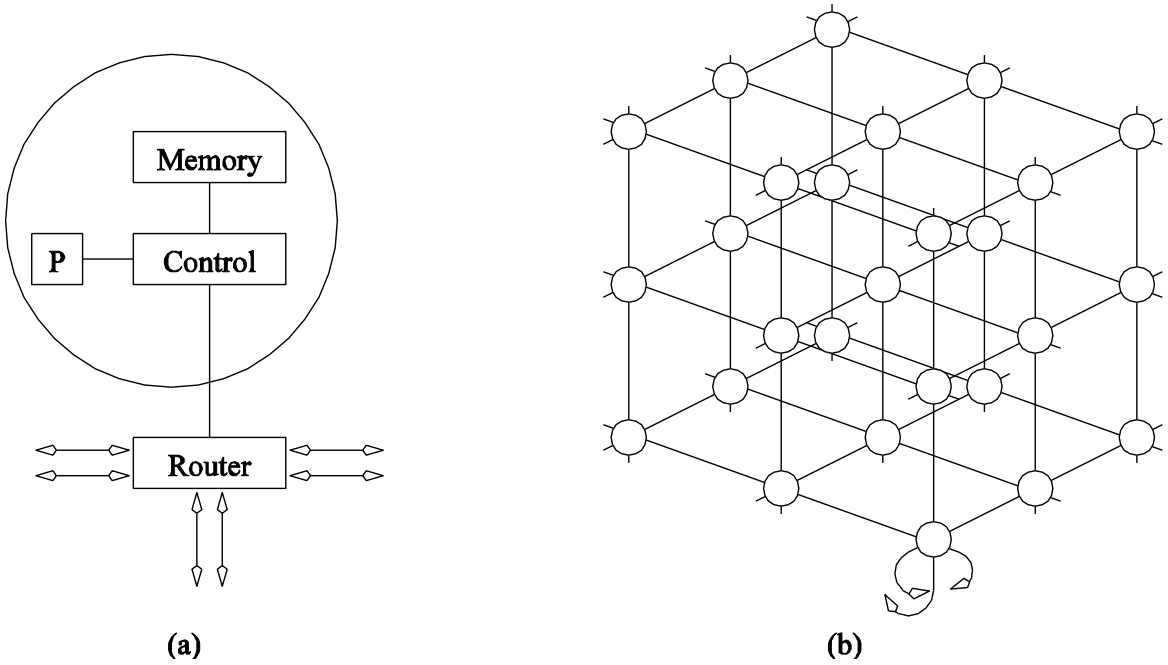
(d) Tower (16 TF)



(e) Blue Gene (1 PF)

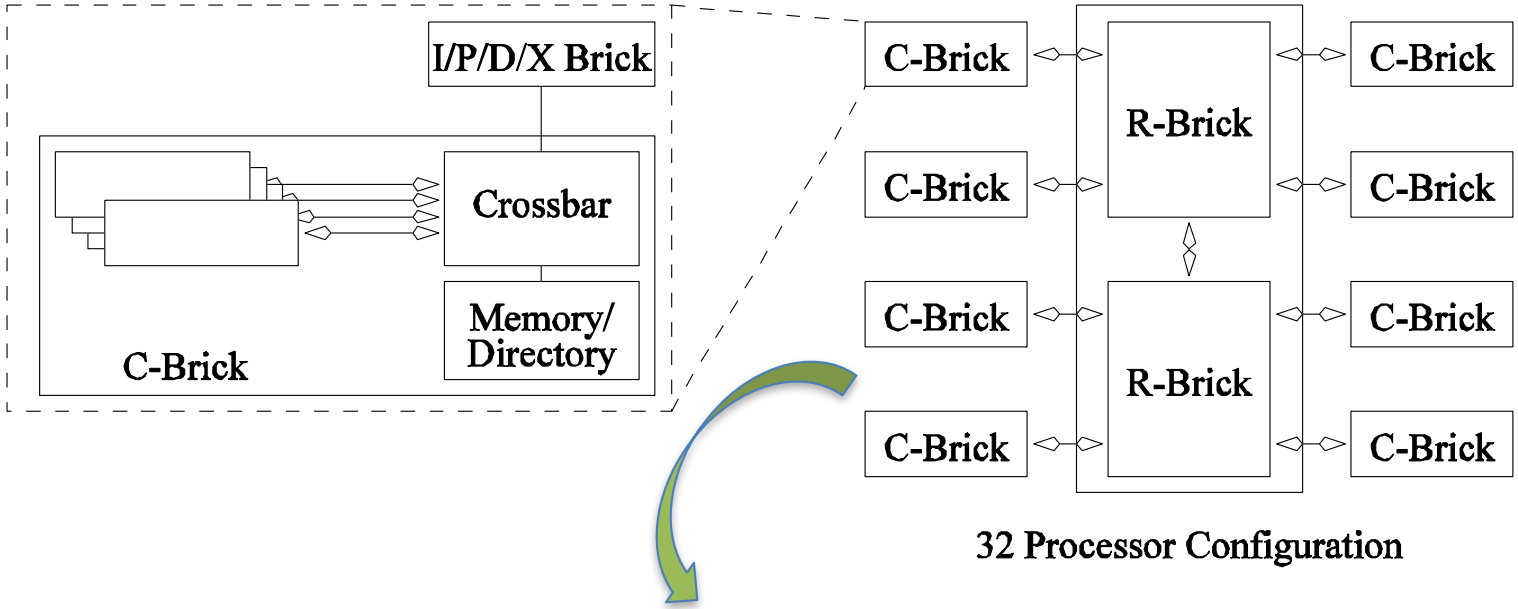
64X16 TF = 1 PF

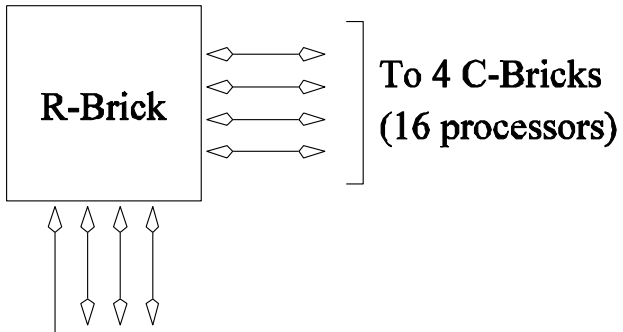
Case Studies:
The Cray T3E Architecture



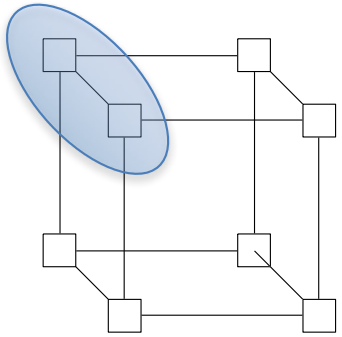
Interconnection network of the Cray T3E:
(a) node architecture; (b) network topology.

Case Studies:
The SGI Origin 3000 Architecture

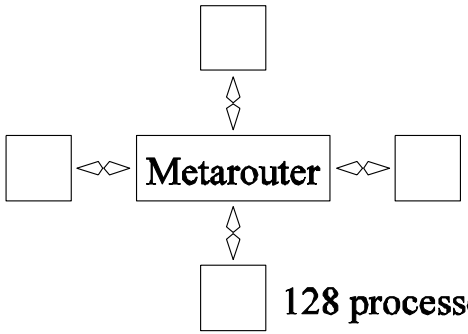




128 Processor Configuration

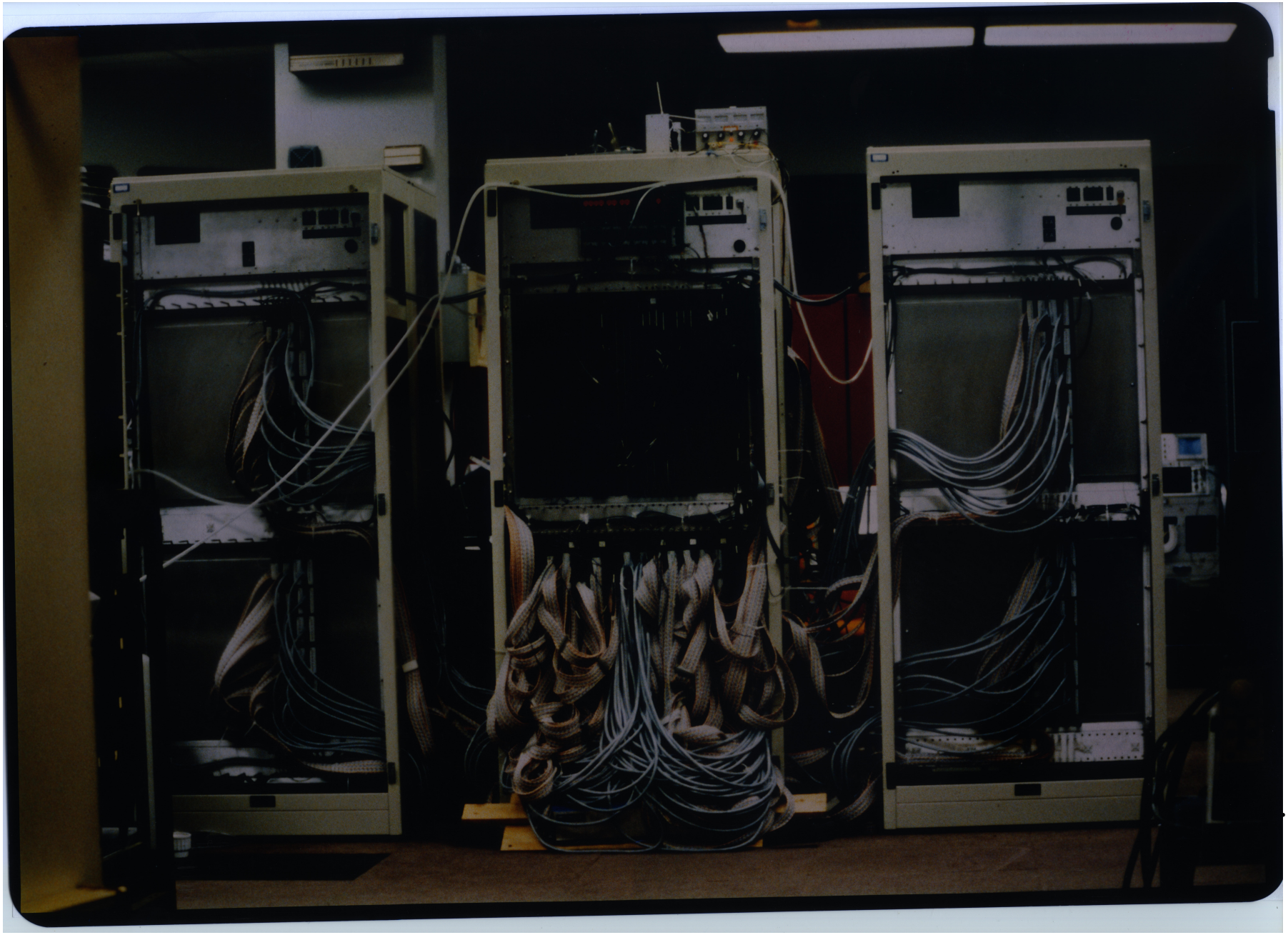


1 R-Brick, 4 C-Bricks, and
16 processors at each vertex.



512 Processor Configuration

The Cedar Architecture



MasPar MP 1

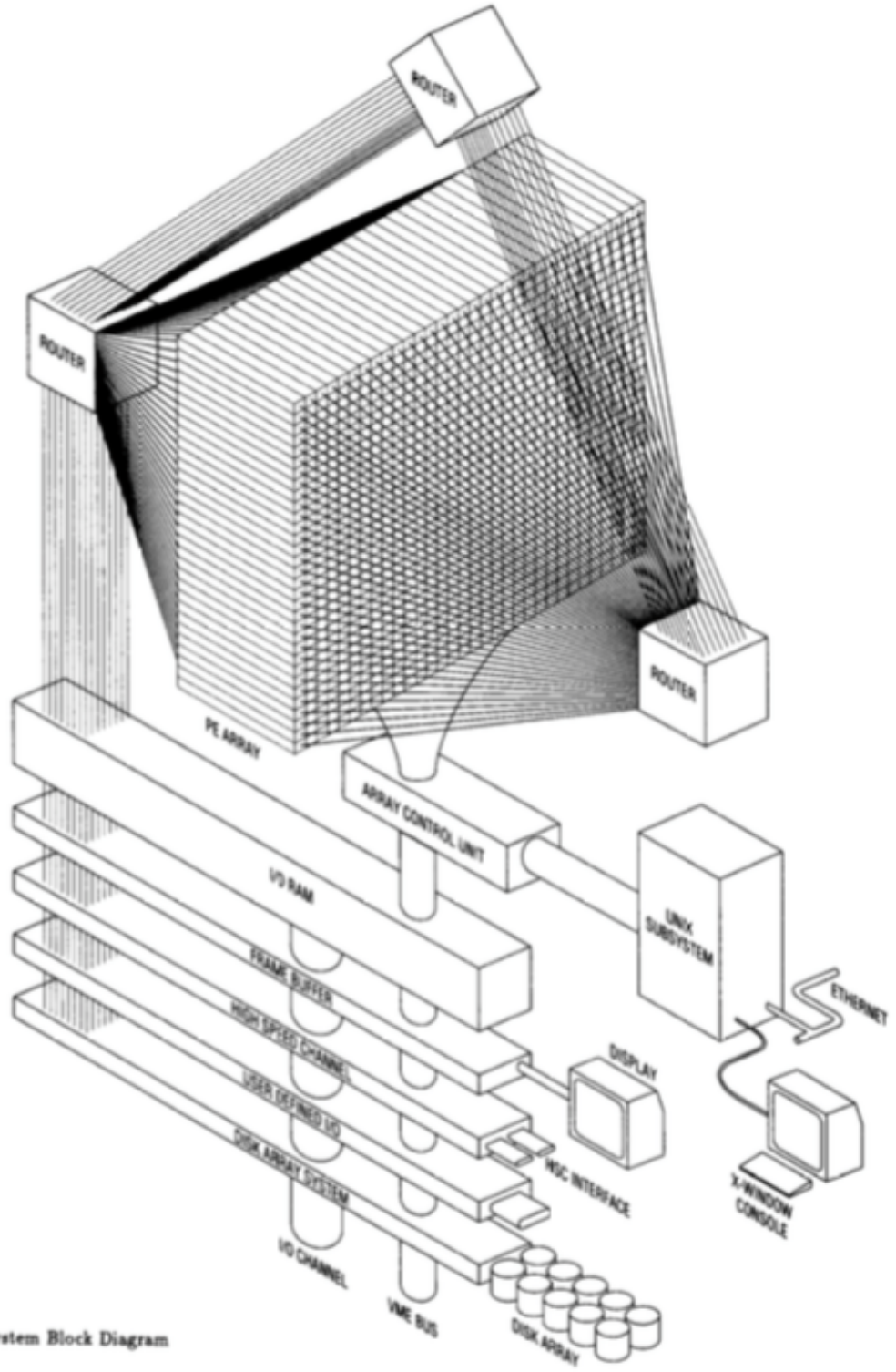


Figure 1: MP-1 System Block Diagram