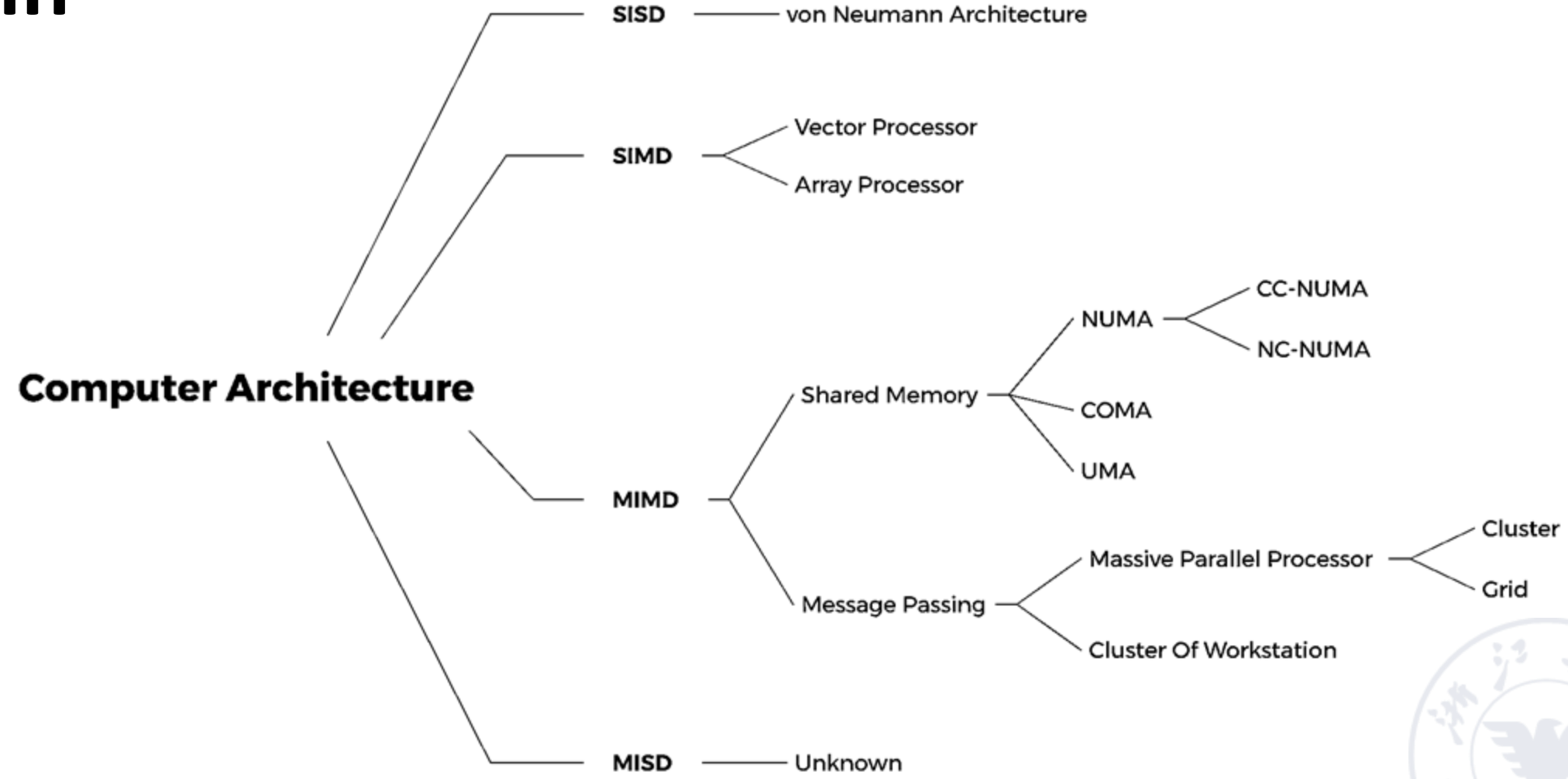
Chapter 5

DLP and TLP

Flynn


Computer Architecture
SISD
von Neumann Architecture
SIMD
Vector Processor
Array Processor
MIMD
Shared Memory
NUMA
CC-NUMA
NC-NUMA
COMA
UMA
Message Passing
Massive Parallel Processor
Cluster
Grid
Cluster Of Workstation
MISD
Unknown

Vector Processor & Scalar Processor

- A pipeline processor, in which the vector data representation and the corresponding vector instructions are set, is called the **vector processor**.

- A pipeline processor that does not have vector data representation and corresponding vector instructions is called a **scalar processor**.

Vector Processor & Scalar Processor

- The particularity of vector pipeline:
  The elements in the vector are rarely correlated during operations.

- Improper vector handling can also cause related problem and frequent function switching.

- Problems to be solved by vector pipeline processing:
  How to deal with vectors and arrays to maximize the effectiveness of the pipeline.

Vector Processor & Scalar Processor

- (1) Horizontal processing method:
  - Vector calculations are performed horizontally from left to right in a row.
- (2) Vertical processing method:
  - The vector calculation is performed vertically from top to bottom in a column manner.
- (3) Vertical and horizontal processing method (group processing method):
  - A combination of horizontal processing and vertical processing.

Vector Processor & Scalar Processor

Example: D = A × (B + C) A, B, C, D — vector of length N

1. Horizontal processing method

- Vector calculations are performed horizontally from left to right in a row-wise manner.
  - First calculate: $d_1 \leftarrow a_1 \times (b_1 + c_1)$
  - Recalculate: $d_2 \leftarrow a_2 \times (b_2 + c_2)$
  - …
  - Final calculation: $d_N \leftarrow a_N \times (b_N + c_N)$
- Compose a loop program for processing.
  - $k_i \leftarrow b_i + c_i$
  - $d_i \leftarrow a_i \times k_i$
- Data related: N times Function switching: 2N times

Vector Processor & Scalar Processor

Example: $D = A \times (B + C)$ A, B, C, D — vector of length N

1. Horizontal processing method

Problems with horizontal processing:

- When calculating each component, RAW correlation occurs, and the pipeline efficiency is low.
- If a static multi-functional pipeline is used, the pipeline must be switched frequently; the throughput of the pipeline is lower than that of sequential serial execution.
- The horizontal processing method is not suitable for vector processors.

Vector Processor & Scalar Processor

Example: D = A × (B + C) A, B, C, D — vector of length N

2. Vertical processing method

Problems with Vertical processing:

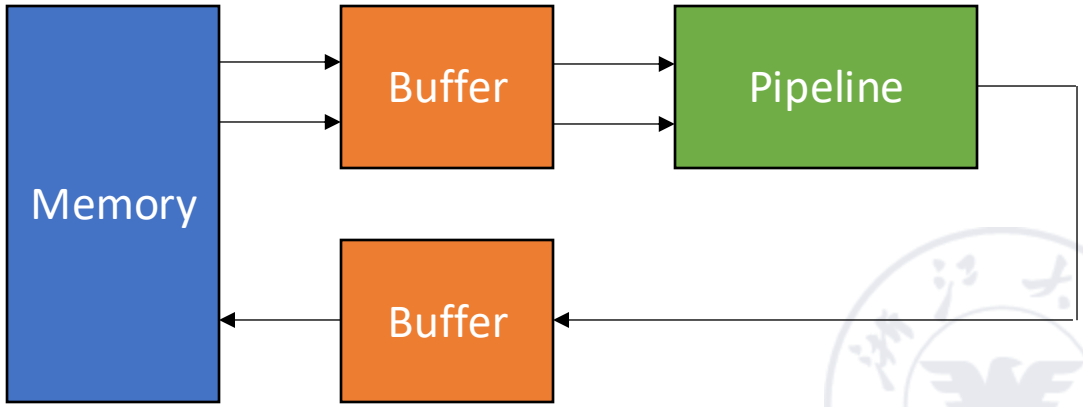
$$K \leftarrow B + C$$
$$D \leftarrow A \times K$$

Data related: 1 times Function switching: 2 times

Vector Processor & Scalar Processor

- Requirements for processor structure: memory-memory structure
- The source vector and destination vector of the vector instruction are stored in the memory, and the intermediate result of the operation needs to be sent back to the memory.

- Memory-memory operation pipeline
  - For example: STAR-100, CYBER-205



Vector Processor & Scalar Processor

3. Vertical and horizontal (grouping) processing method

- Divide the vector into several groups, process them in a vertical manner, and process each group in turn.
- For the above example, let:
  - $N = S \times n + r$
  - Where N is the length of the vector, S is the number of groups, n is the length of each group, and r is the remainder.
  - If the remaining r numbers are also treated as a group, there are a total of S 1 groups.

Vector Processor & Scalar Processor

3. Vertical and horizontal (grouping) processing method

- The calculation process is:
  - Calculate the first group: $d_{1\sim n} \leftarrow a_{1\sim n} \times (b_{1\sim n} + c_{1\sim n})$
  - Calculate the second group: $d_{(n+1)\sim 2n} \leftarrow a_{(n+1)\sim 2n} \times (b_{(n+1)\sim 2n} + c_{(n+1)+2n})$
  - …
  - Final calculation the last group: $d_{(s*n+1)\sim N} \leftarrow a_{(s*n+1)\sim N} \times (b_{(s*n+1)\sim N} + c_{(s*n+1)\sim N})$

Data related: S+1 times Function switching: 2(S+1) times

Vector Processor & Scalar Processor

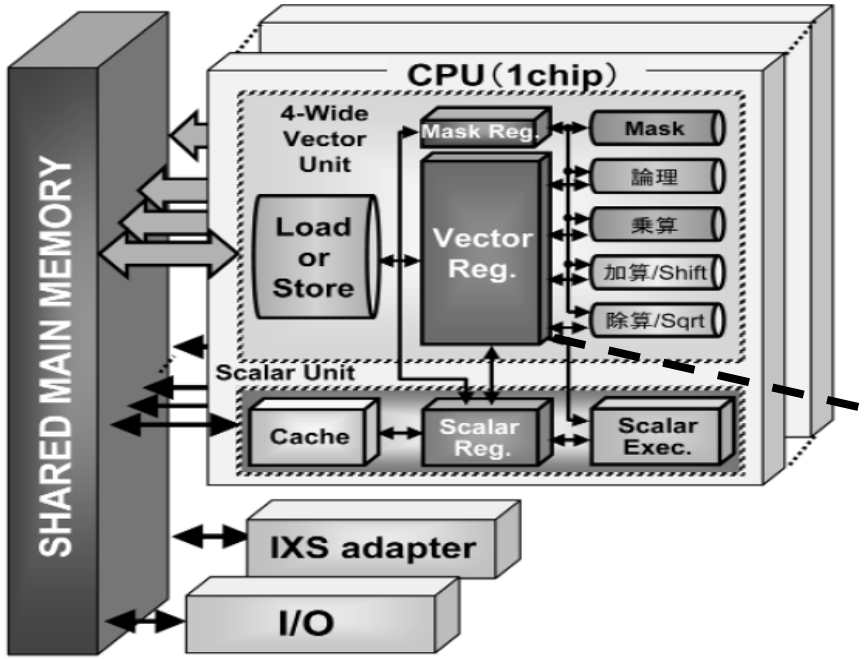
- Requirements for processor structure: register-register structure

- Set up vector registers that can be accessed quickly to store source vectors, destination vectors, and intermediate results, so that the input and output ends of the arithmetic components are connected with the vector registers to form a register-register type operation pipeline.

Vector Processor & Scalar Processor

- Typical register-vector processor with register structure

American CRAY-1, my country's YH-1 supercomputer, GRAP-3, earth simulator



Vector processor SX-8
for earth simulator

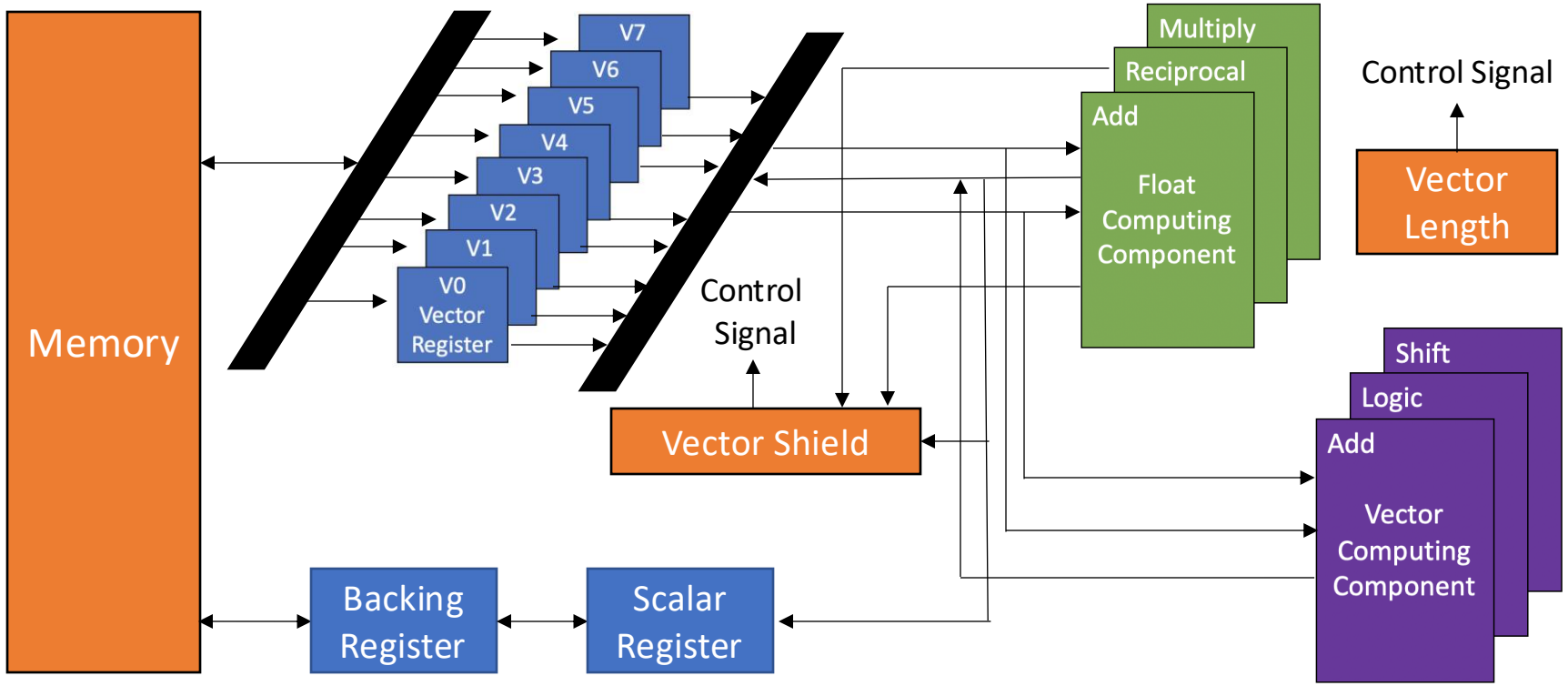
72 vector registers
256 elements

Vector Processor & Scalar Processor

- The structure of the vector pipeline processing machine varies from machine to machine.

- Take the vector pipeline processing part in CRAY-1 as an example to introduce some structural features of the register-register-oriented vector pipeline processing machine.

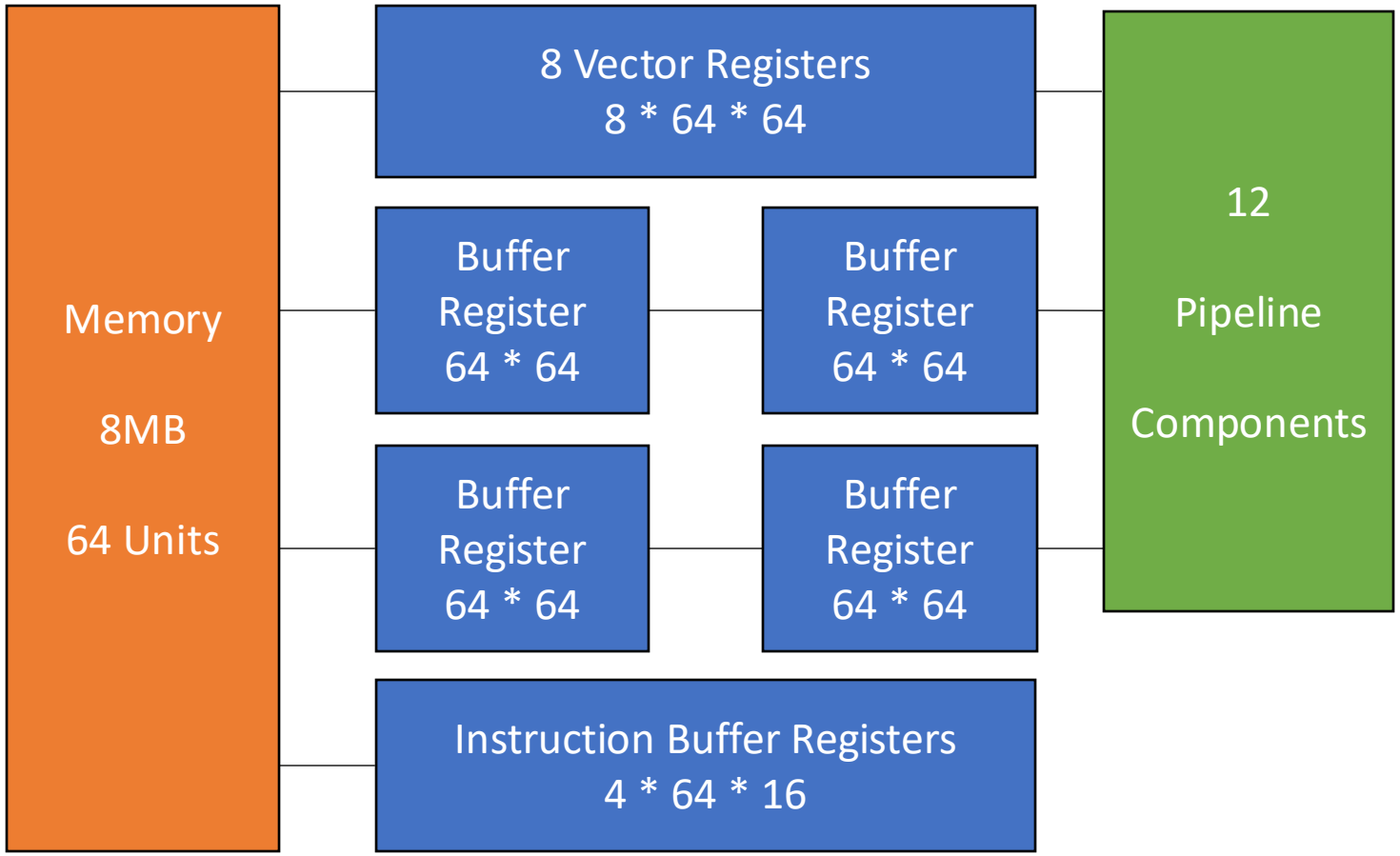
CRAY-1



CRAY-1
American CRAY Company
100 million FLOPS
Clock Period: 12.5 ns

There are 12 single-function pipelines that can work in parallel, which can perform various operations of address, vector, and scalar in a pipeline.

CRAY-1


Memory
8MB
64 Units
8 Vector Registers
8 * 64 * 64
Buffer Register 64 * 64
Buffer Register 64 * 64
Buffer Register 64 * 64
Buffer Register 64 * 64
Instruction Buffer Registers
4 * 64 * 16
12
Pipeline
Components

Features of CRAY-1 Vector Processor

- Each vector register Vi has a separate bus connected to 6 vector functional units.
- Each vector function unit also has a bus that returns the result of the operation to the vector register bus.
- As long as there is no Vi conflict and functional conflict, each Vi and each functional unit can work in parallel, which greatly speeds up the processing of vector instructions.

Features of CRAY-1 Vector Processor

- Vi conflict: The source vector or result vector of each vector instruction working in parallel uses the same Vi.

- Writing and reading data related:
  - $V0 \leftarrow V1 + V2$
  - $V3 \leftarrow V0 \times V4$

- Reading data related
  - $V0 \leftarrow V1 + V2$
  - $V3 \leftarrow V1 \times V4$

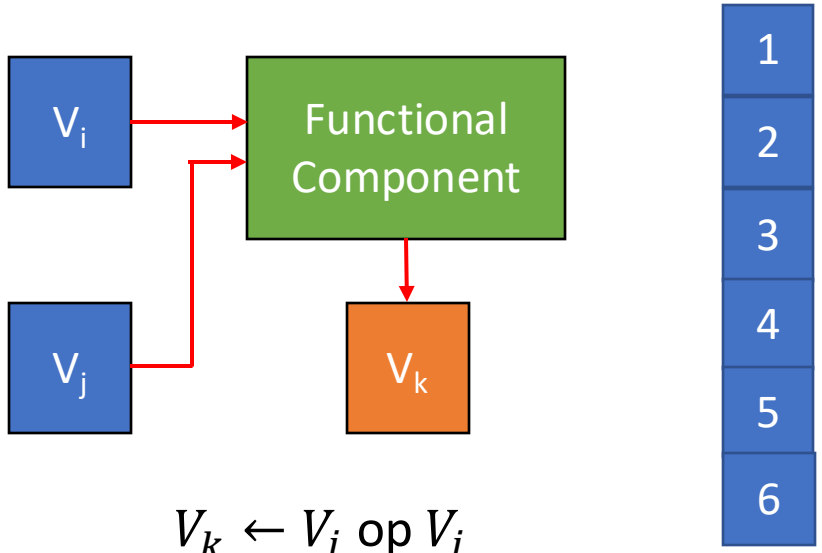
Features of CRAY-1 Vector Processor

- Functional conflict: Each vector instruction working in parallel must use the same functional unit.

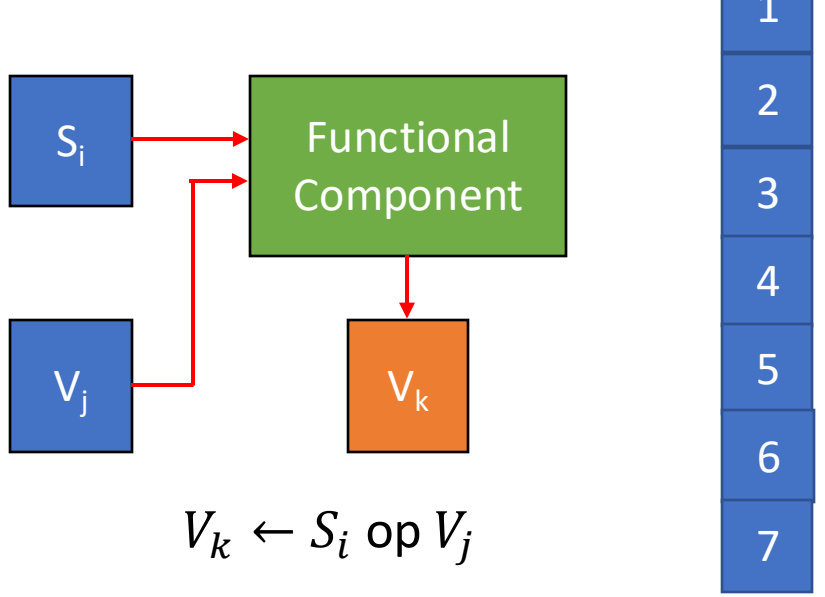
- For example: (all need to use multiplication feature)
  - $V3 \leftarrow V1 \times V2$
  - $V5 \leftarrow V4 \times V6$

- After the last component of the first vector instruction is executed and the float multiplication function is released, the second vector starts to execute.

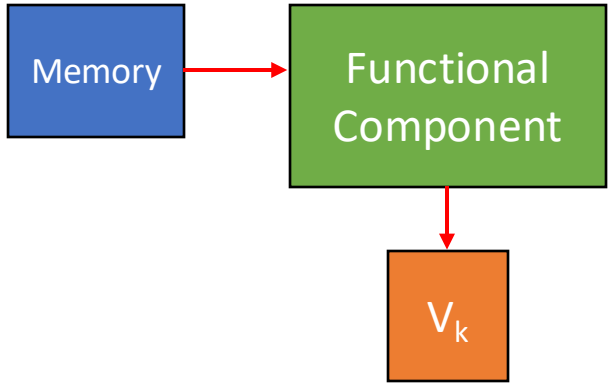
Instruction Types of CRAY-1



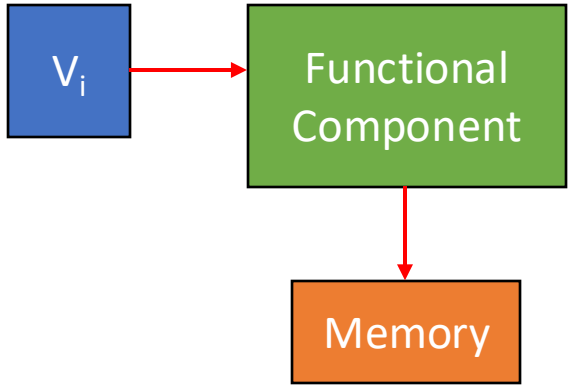
$V_k \leftarrow V_i$ op V_j



$V_k \leftarrow S_i$ op V_j



$V_k \leftarrow Memory$



$Memory \leftarrow V_i$

Improve the Performance of Vector Processor

- Set up multiple functional components and make them work in parallel.
- Use vector chaining technology to speed up the execution of a string of vector instructions.
- Adopt recycling mining technology to speed up recycling processing.
- Using a multi-processor system to further improve the performance.

Improve the Performance of Vector Processor

1. Set up multiple functional units and make them work in parallel.

- These components can work in parallel, and each work in a pipeline manner, thus forming multiple parallel operation pipelines.

For example:

- CRAY-1 vector processor has 4 groups of 12 single-function pipeline components:
  - Vector components: vector addition, shift, logic operation
  - Floating point components: float addition, multiplication, reciprocal calculation
  - Scalar components: scalar addition, shift, logic operation, number "1"/count
  - Address calculation components: integer addition, multiplication

Improve the Performance of Vector Processor

2. Use vector chaining technology to speed up the execution of a string of vector instructions.

- Link feature: It has two related instructions that are written first and then read. In the case of no conflicts between functional components and source vector conflicts, functional components can be linked for pipeline processing to achieve the purpose of speeding up execution.

- The essence of the link feature: the result of introducing the idea of pipeline orientation to the vector execution process.

Improve the Performance of Vector Processor

Example: Use vector chaining technology to perform vector operations on CRAY-1

- D=A×(B+C)

- Assuming that the vector length is N ≤ 64, the vector elements are floating numbers, and the vectors B and C have been stored in V0 and V1.
- Draw a link diagram and analyze the different execution times in the case of non-link execution and link execution.
- Solution: Use the following three vectors to complete the above operation:
  - V3 ← memory // access vector A
  - V2 ← V0 + V1 // Vector B and Vector C perform floating point addition
  - V4 ← V2 × V3 // Floating point multiplication, the result is stored in V4

Improve the Performance of Vector Processor

Hypothesis: It takes one beat to send vector data elements to the vector function unit and to store the result in the vector register, and one beat time to send data from the memory to the fetch function unit.

V3 ← memory

V2 ← V0 + V1

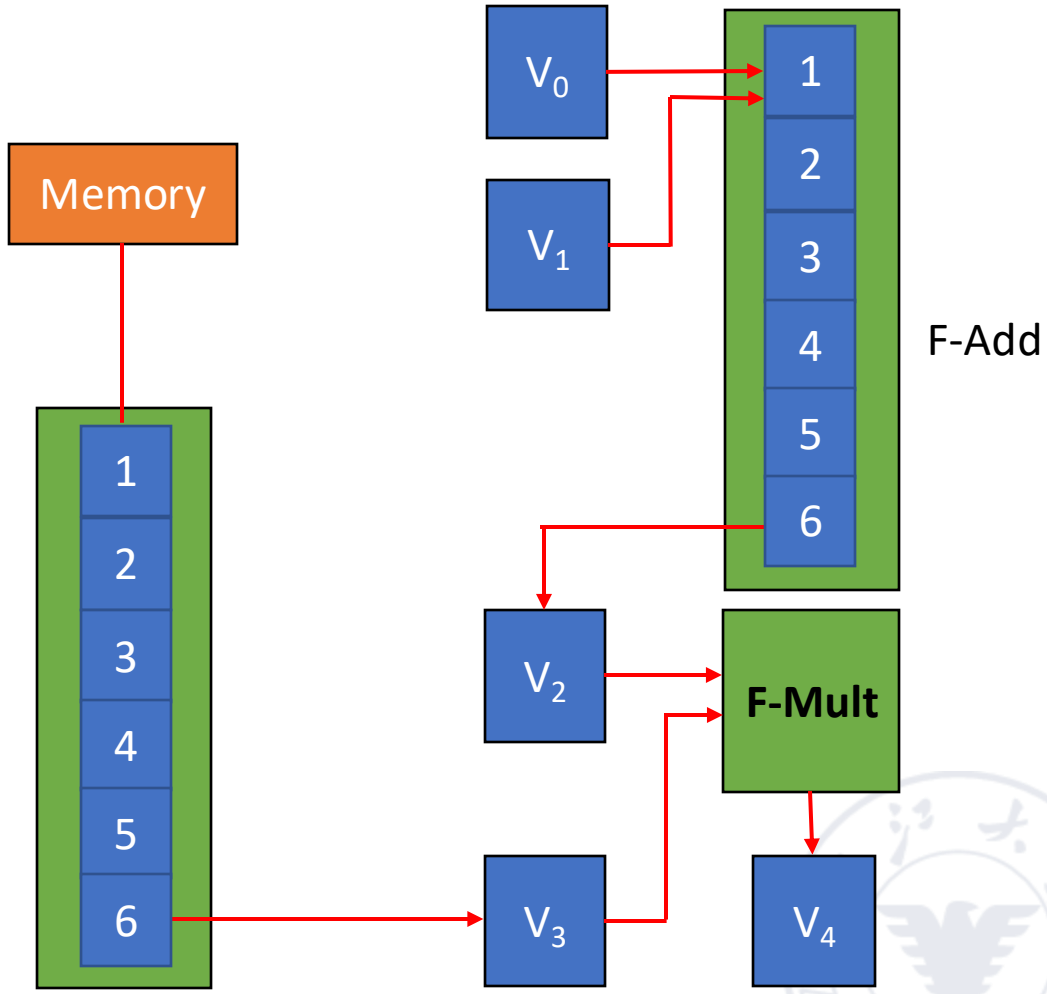
V4 ← V2 × V3

No Conflict
Parallelizable

RAW but
No functional conflict
Not parallelizable
Linkable

Improve the Performance of Vector Processor

- Why 8 beats?
- It takes 1 beat of transmission delay to start the memory fetch, send the element to the functional part and send the calculation result to Vi.



Improve the Performance of Vector Processor

- Calculate D=A·(B+C), A, B, C, D are all N elements Vector, vector length N≤64, complete operation with 3 instructions,
  - (1). $V3 \leftarrow A$
  - (2). $V2 \leftarrow V0 + V1$
  - (3). $V4 \leftarrow V2 * V3$

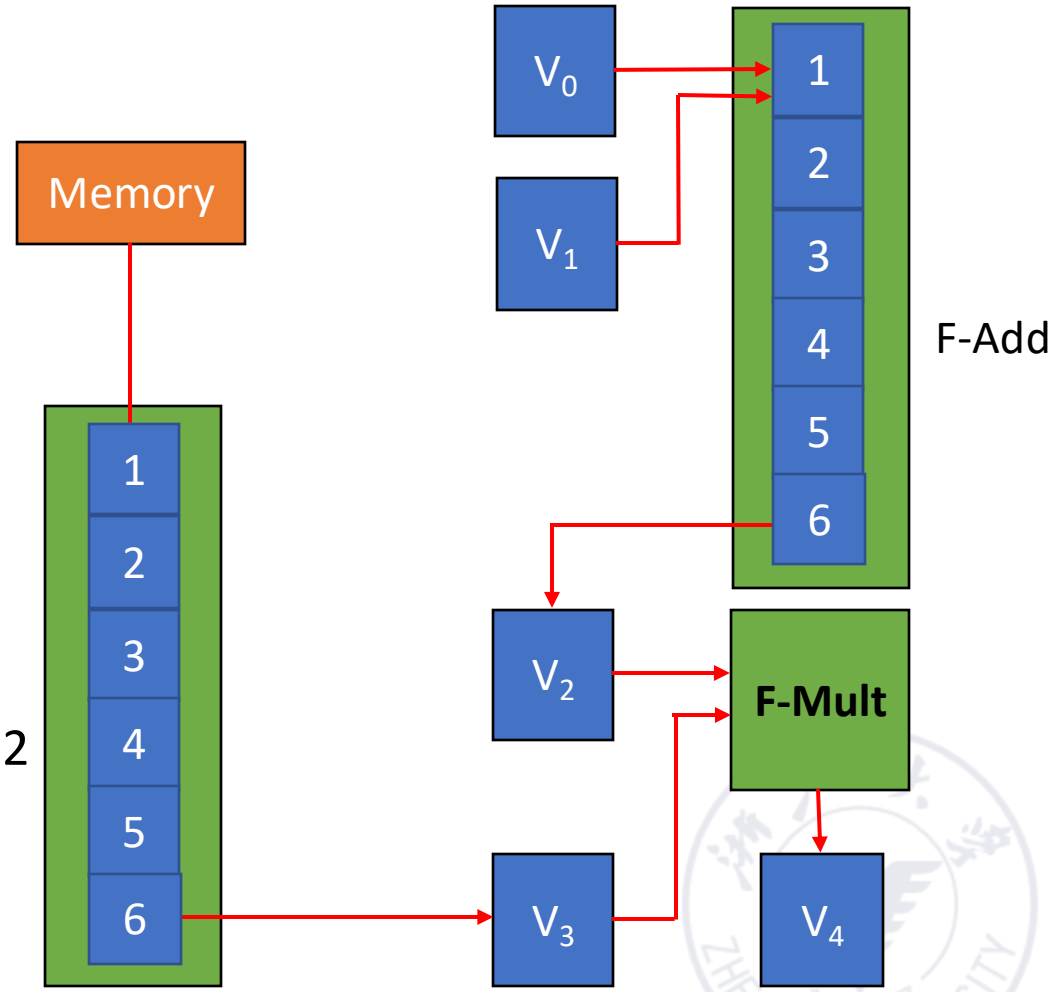
- Calculate how many shots each of the following three methods will take to get all the results?
  - (1), (2), (3)Serial execution;
  - After (1) and (2) are executed in parallel, execute (3);
  - Use vector chaining technology.

Improve the Performance of Vector Processor

- Let the vector length be N
  - (1). $V3 \leftarrow A$
  - (2). $V2 \leftarrow V0 + V1$
  - (3). $V4 \leftarrow V2 * V3$

- (1) The execution time using serial method is:

[(1+6+1)+N-1] + [(1+6+1)+N-1] + [(1+7+1)+N-1] = 3N + 22

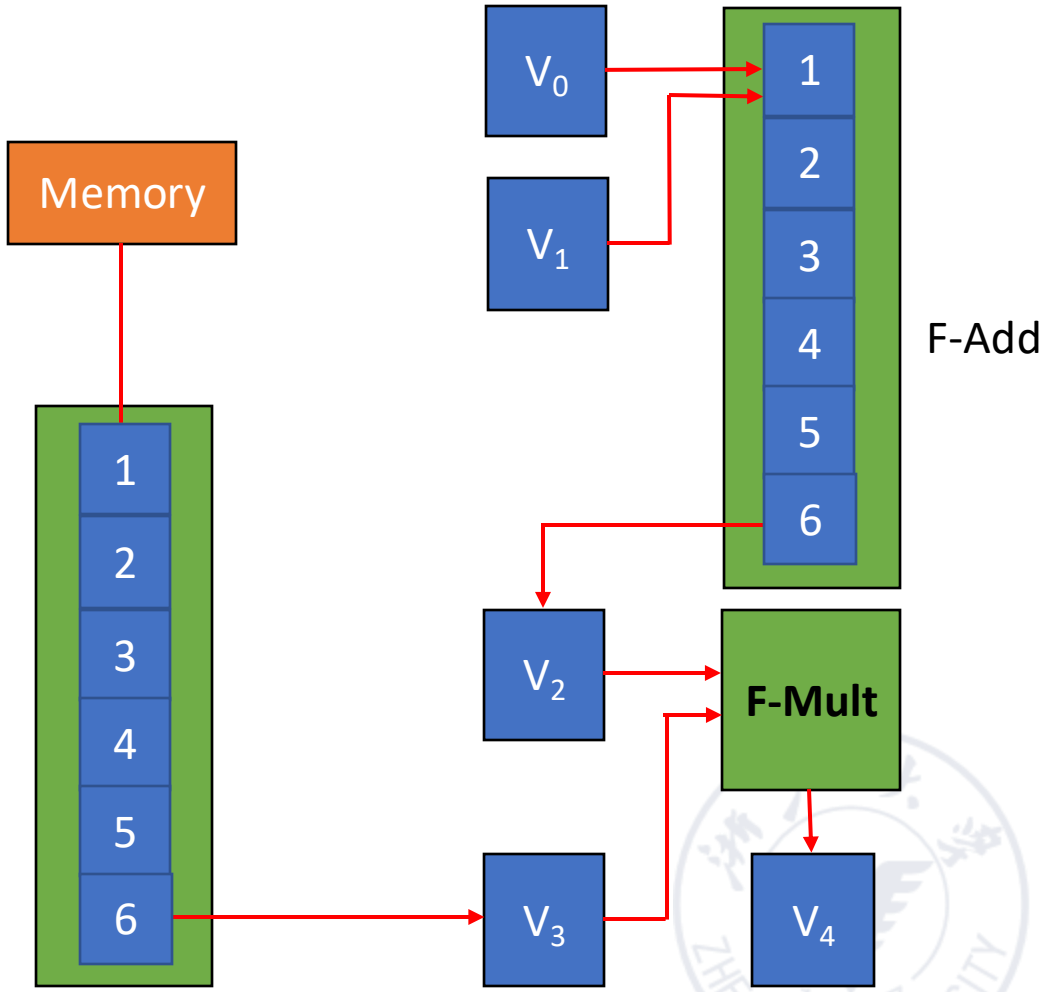


Improve the Performance of Vector Processor

- Let the vector length be N
  - (1). $V3 \leftarrow A$
  - (2). $V2 \leftarrow V0 + V1$
  - (3). $V4 \leftarrow V2 * V3$

- (2) The first two instructions are parallel, and the third is serial. The execution time is:

max{[(1+6+1)+N-1], [(1+6+1)+N-1]} + [(1+7+1)+N-1] =
2N + 15



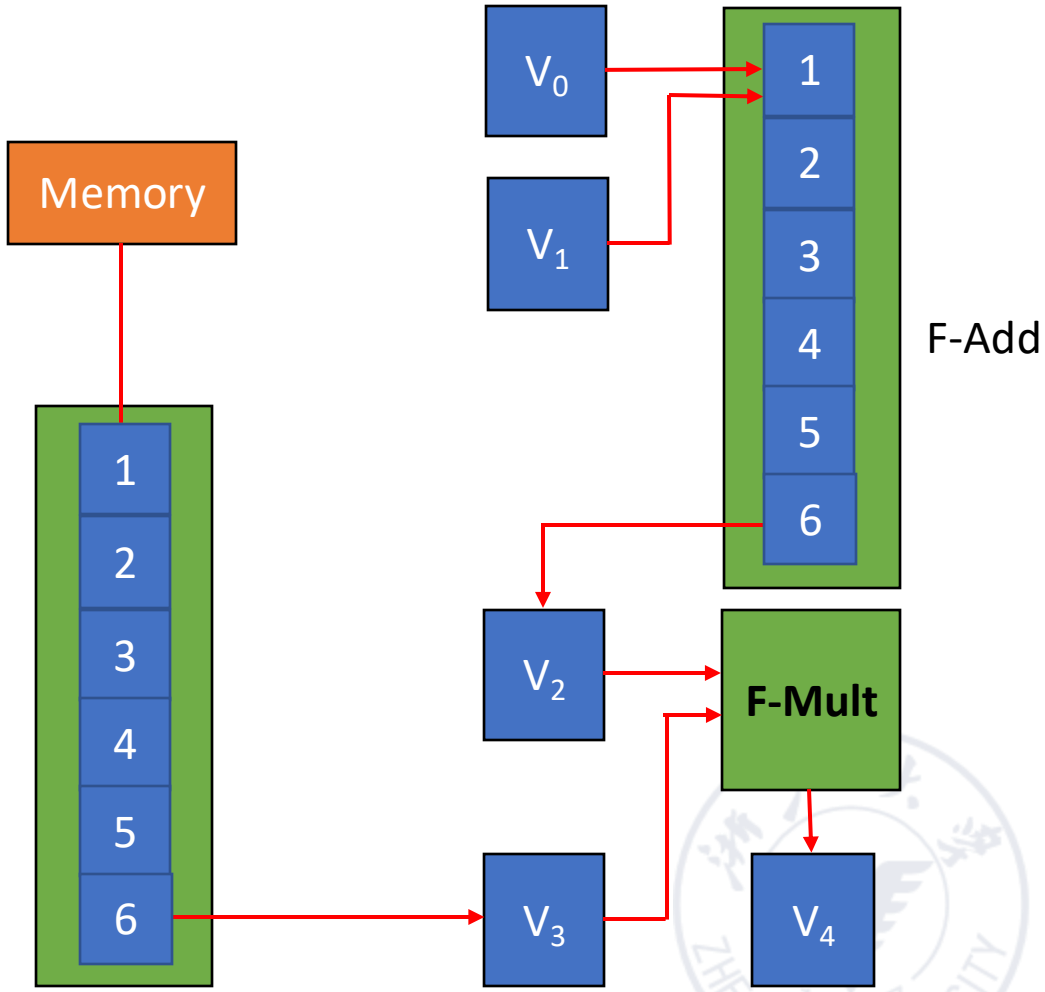
Improve the Performance of Vector Processor

- Let the vector length be N
  - (1). $V3 \leftarrow A$
  - (2). $V2 \leftarrow V0 + V1$
  - (3). $V4 \leftarrow V2 * V3$

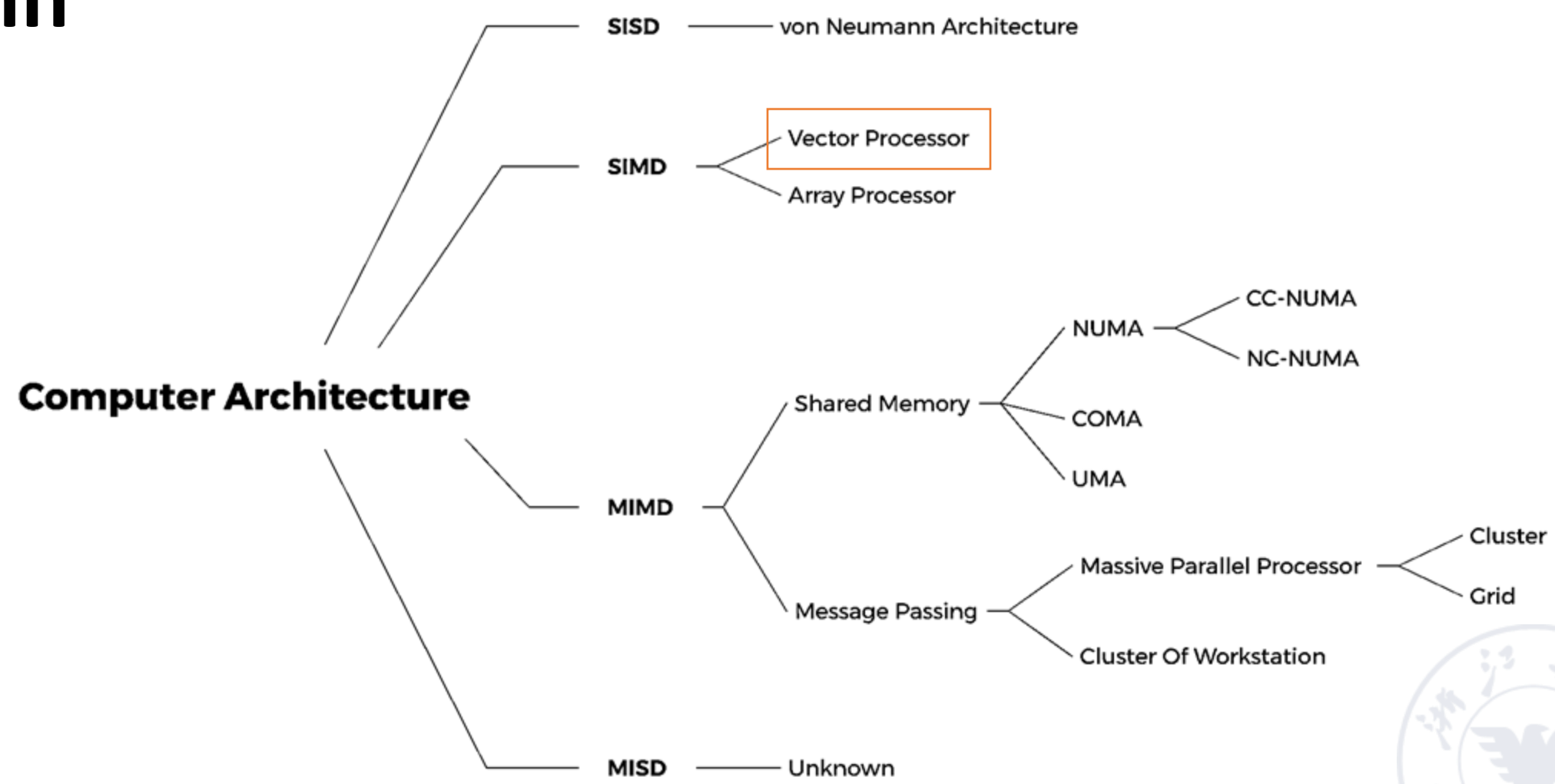
- (3) Use vector chaining technology.

  The execution time is:

max{(1+6+1), (1+6+1)} + (1+7+1)+N-1 = N + 16



Flynn


Computer Architecture
SISD
von Neumann Architecture
SIMD
Vector Processor
Array Processor
MIMD
Shared Memory
NUMA
CC-NUMA
NC-NUMA
COMA
UMA
Message Passing
Massive Parallel Processor
Cluster
Grid
Cluster Of Workstation
MISD
Unknown

Improve the Performance of Vector Processor

- Set up multiple functional components and make them work in parallel.
- Use link technology to speed up the execution of a string of vector instructions.
- Adopt recycling mining technology to speed up recycling processing.
- Using a multi-processor system to further improve the performance.

Improve the Performance of Vector Processor

3. Segmented Vector

What should I do if the length of the vector is greater than the length of the vector register?

- When the length of the vector is greater than the length of the vector register, the long vector must be divided into segments of fixed length, and then processed in loops, and only one vector segment is processed in each loop.

- The aboved is segmented vector technology.
  - It is controlled by system hardware and software and is transparent to programmers.

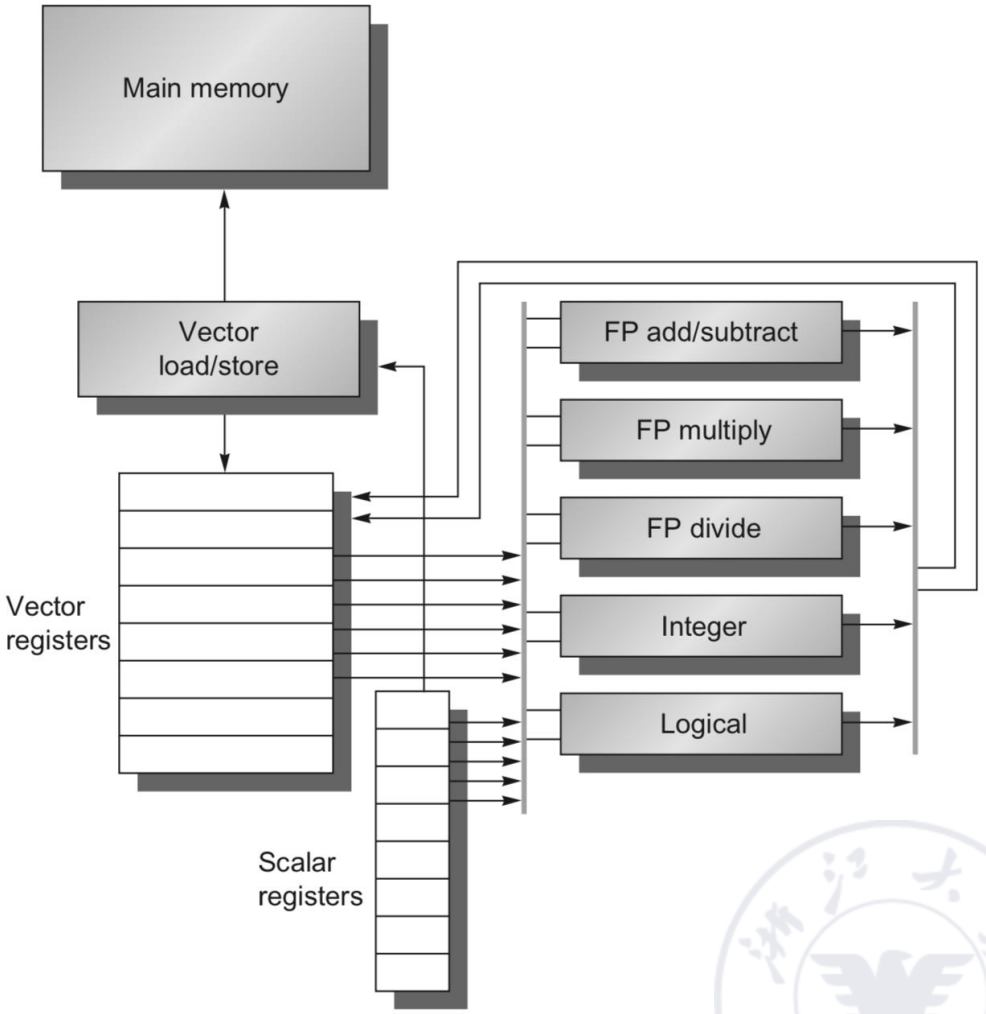
Improve the Performance of Vector Processor

4. Adopt multi-processor system
Many new vector processor systems have adopted a multi-processor system structure. E.g.

- CRAY-2
  - Contains 4 vector processors
  - Floating point calculation speed can reach up to 1800MFLOPS
- CRAY Y-MP, C90
  - Can contain up to 16 vector processors

RV64V

- Loosely based on Cray-1
- 32 62-bit vector registers
  - Register file has 16 read ports and 8 write ports
- Vector functional units
  - Fully pipelined
  - Data and control hazards are detected
- Vector load-store unit
  - Fully pipelined
  - One word per clock cycle after initial latency
- Scalar registers
  - 31 general-purpose registers
  - 32 floating-point registers



Example: DAXPY (Double Precision a*X plus Y)

RISC-V (scalar):

```
        fld    f0,a          # Load scalar a
        addi   x28,x5,#256   # Last address to load
Loop:   fld    f1,0(x5)      # Load X[i]
        fmul.d f1,f1,f0      # a × X[i]
        fld    f2,0(x6)      # Load Y[i]
        fadd.d f2,f2,f1      # a × X[i] + Y[i]
        fsd    f2,0(x6)      # Store into Y[i]
        addi   x5,x5,#8      # Increment index to X
        addi   x6,x6,#8      # Increment index to Y
        bne    x28,x5,Loop   # Check if done
```

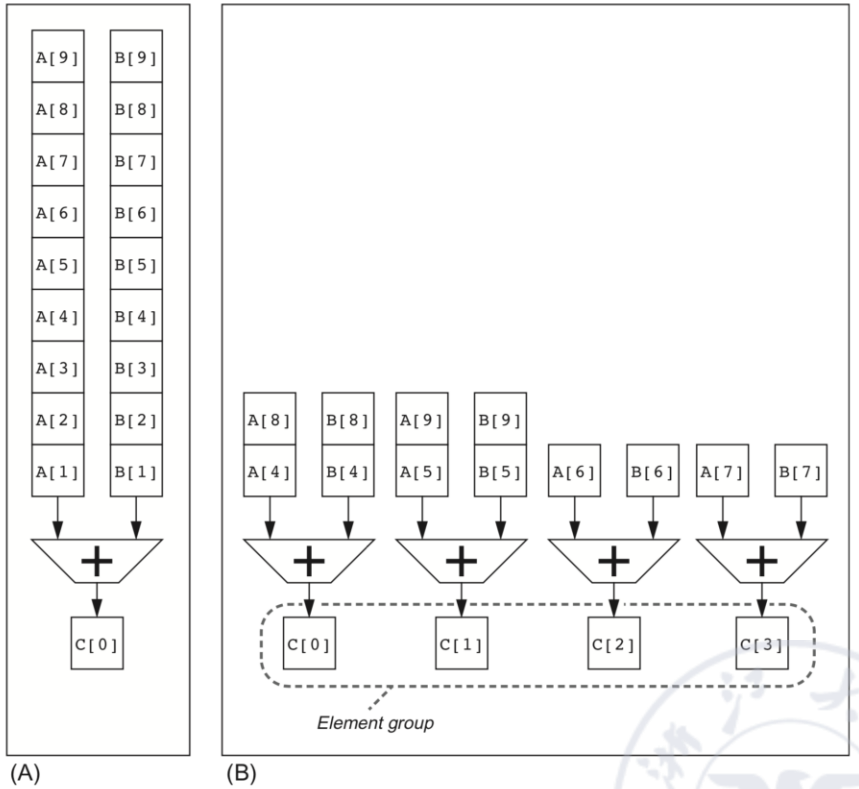
RV64V code:

```
vsetdcfg  4*FP64      # Enable 4 DP FP vregs
fld       f0,a        # Load scalar a
vld       v0,x5       # Load vector X
vmul      v1,v0,f0    # Vector-scalar mult
vld       v2,x6       # Load vector Y
vadd      v3,v1,v2    # Vector-vector add
vst       v3,x6       # Store the sum
vdisable              # Disable vector regs
```

Multiple Lanes: Beyond One Element per Clock Cycle

- Element *n* of vector register *A* is “hardwired” to element *n* of vector register *B*
  - Allows for multiple hardware lanes

- The RV64V instruction set has the property that all vector arithmetic instructions only allow element N of one vector register to take part in operations with element N from other vector registers.
  - This dramatically simplifies the design of a highly parallel vector unit, which can be structured as multiple parallel *lanes*.



Gather-Scatter: Handling Sparse Matrices in Vector Architectures

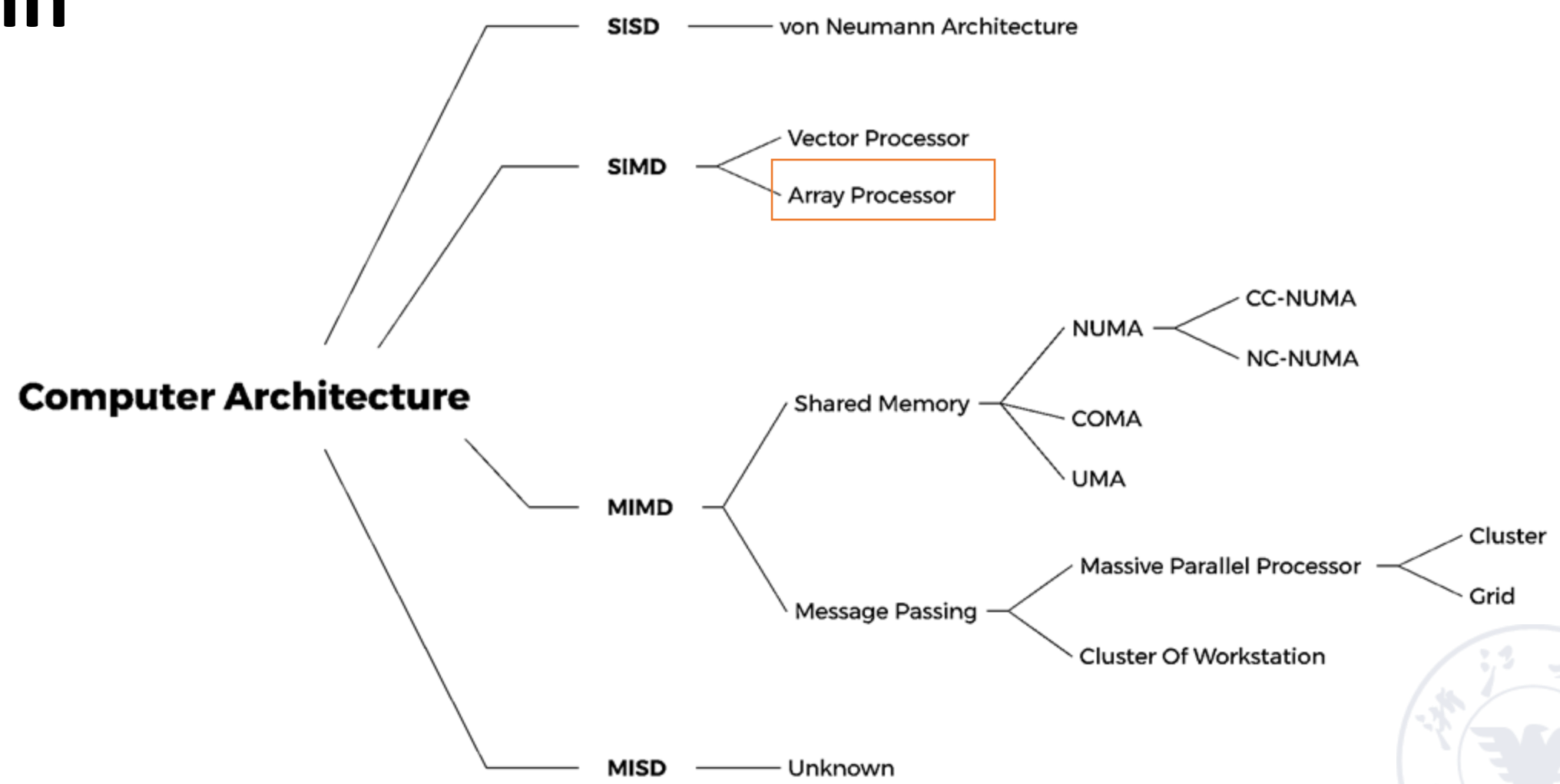
- Consider:

```
for (i = 0; i < n; i=i+1)
        A[K[i]] = A[K[i]] + C[M[i]];
```

- This code implements a sparse vector sum on the arrays A and C, using index vetors K and M to designate the nonzero elements of A and C.
- Use index vector:

```
vsetdcfg        4*FP64          # 4 64b FP vector registers
vld             v0, x7          # Load K[]
vldx            v1, x5, v0      # Load A[K[]]
vld             v2, x28         # Load M[]
vldi            v3, x6, v2      # Load C[M[]]
vadd            v1, v1, v3      # Add them
vstx            v1, x5, v0      # Store A[K[]]
vdisable                        # Disable vector registers
```

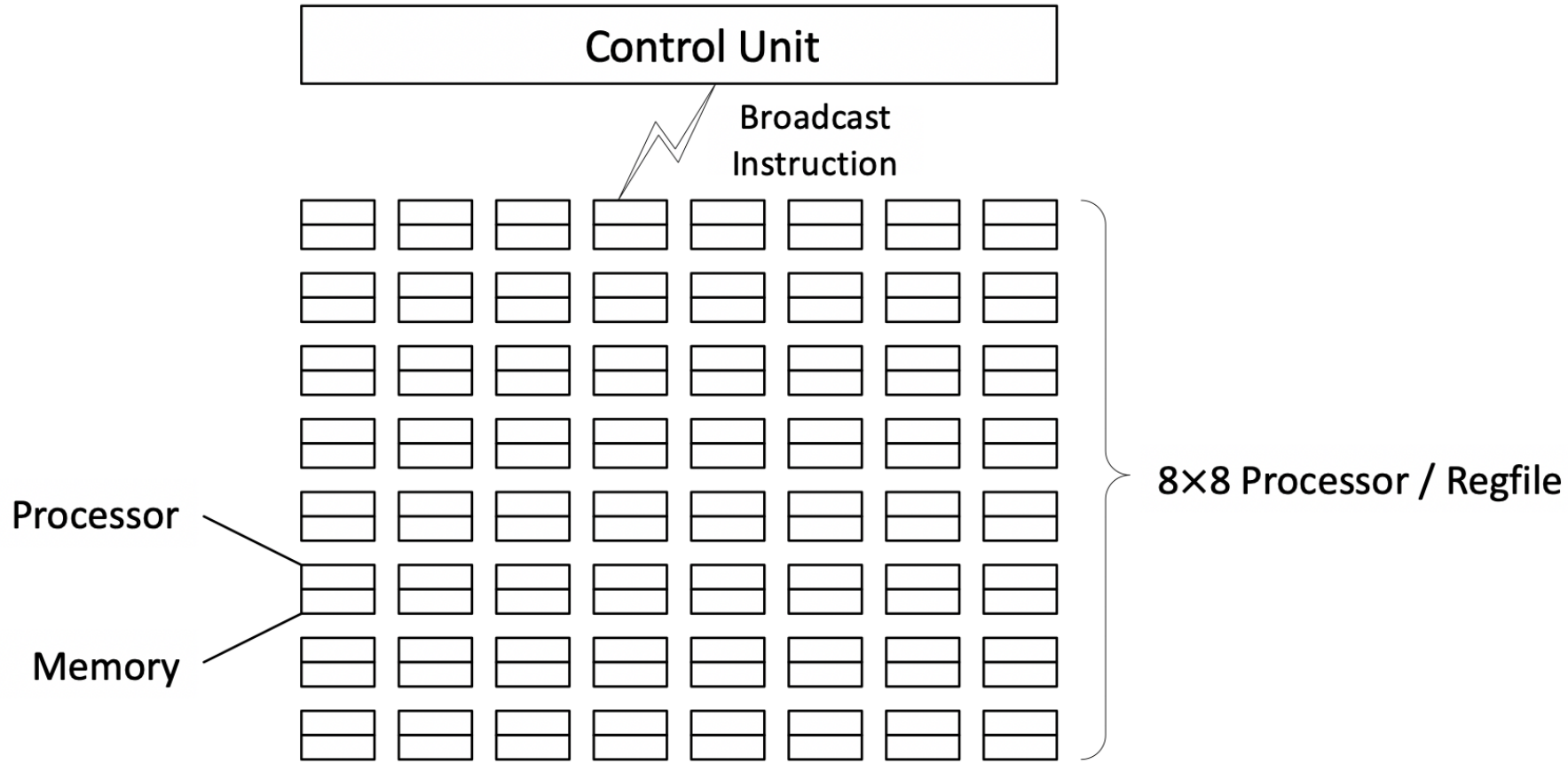
Flynn


Computer Architecture
SISD
von Neumann Architecture
SIMD
Vector Processor
Array Processor
MIMD
Shared Memory
NUMA
CC-NUMA
NC-NUMA
COMA
UMA
Message Passing
Massive Parallel Processor
Cluster
Grid
Cluster Of Workstation
MISD
Unknown

Array Processor

- N processing elements PE_0 to PE_{N-1} are repeatedly set.
- Interconnect in a certain way to form an array.
- Under the control of a single control unit, the operations specified by the same instruction are completed in parallel for the data allocated by the processing units.
- Array processors are sometimes called parallel processors.

ILLIAC IV

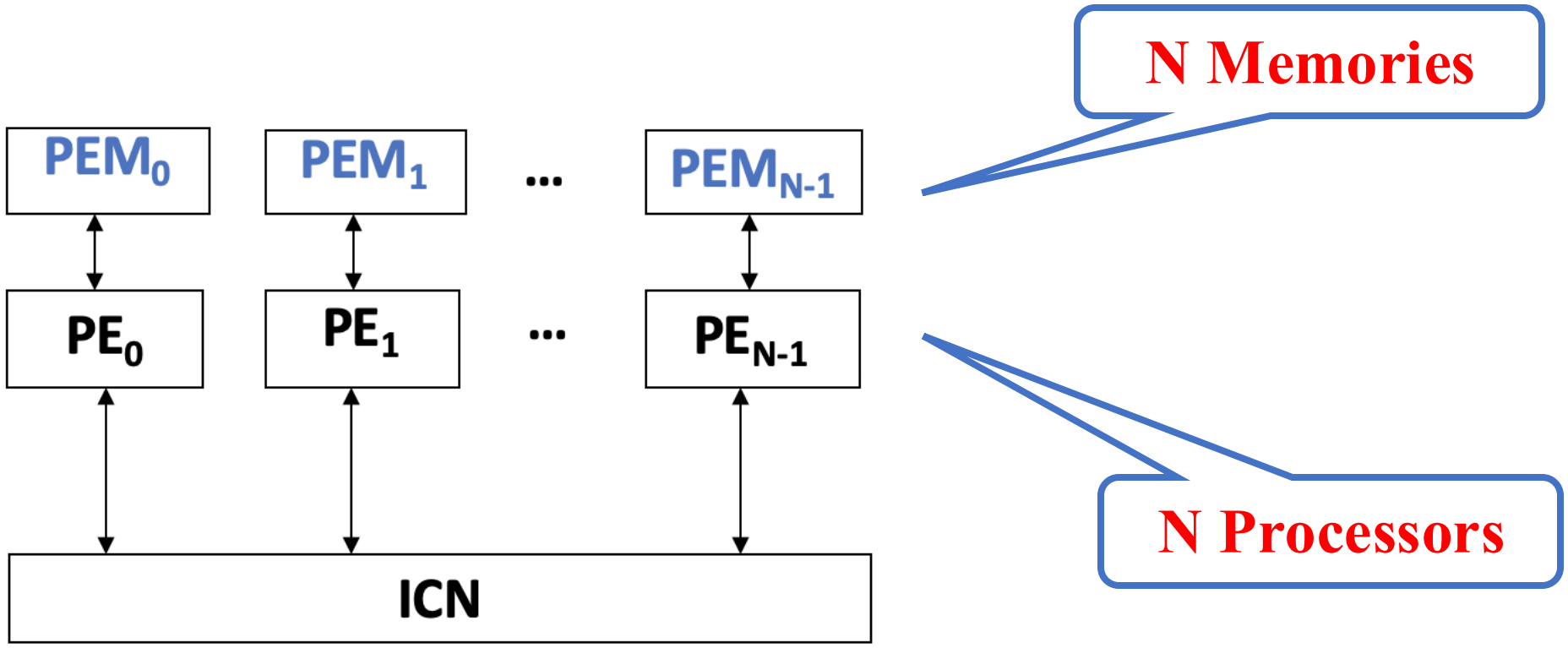

Control Unit
Broadcast
Instruction
Processor
Memory
8×8 Processor / Regfile

Basic Structure

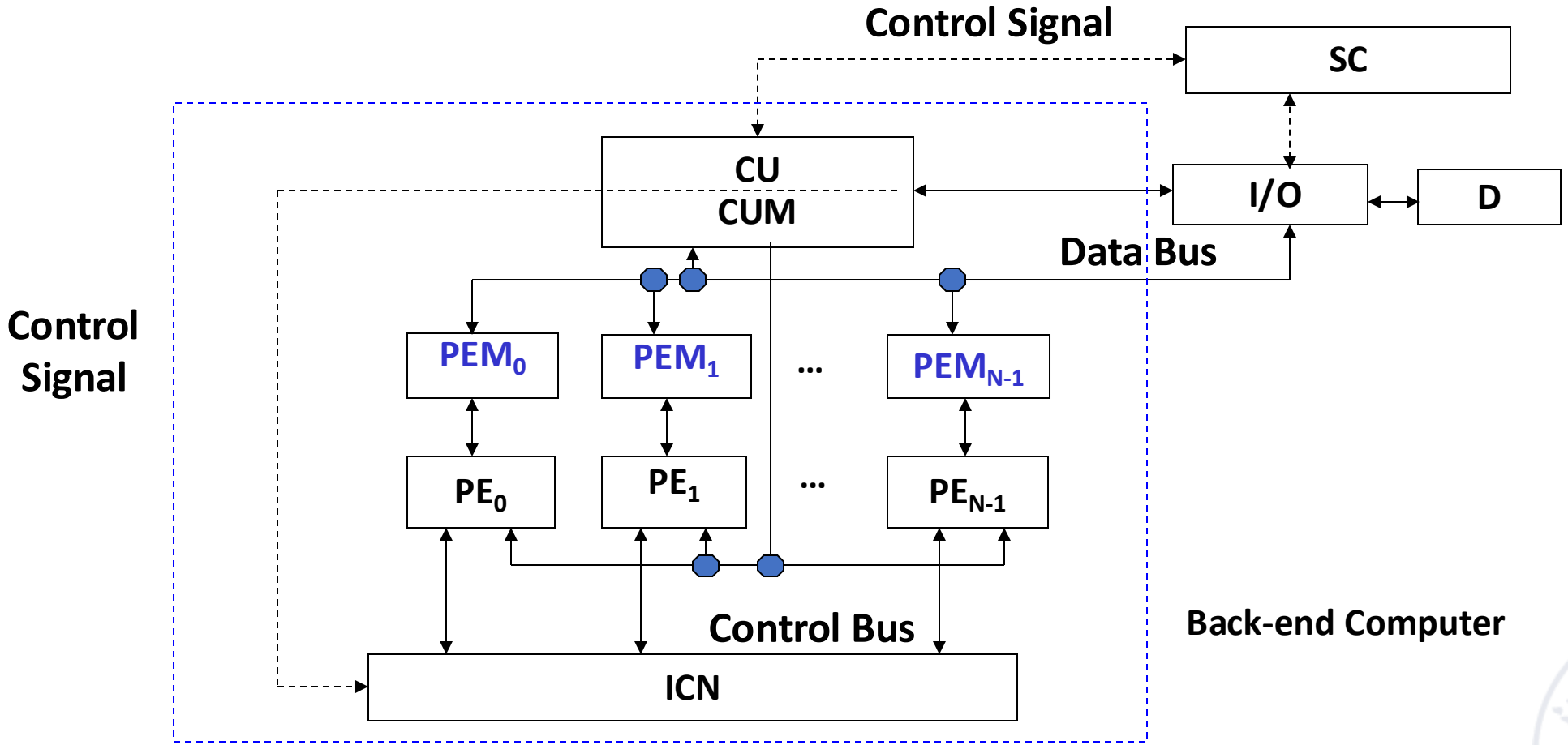
According to the composition of the memory in the system, the array processor can be divided into two basic structures:

- Distributed memory
- Centralized shared memory

Distributed memory

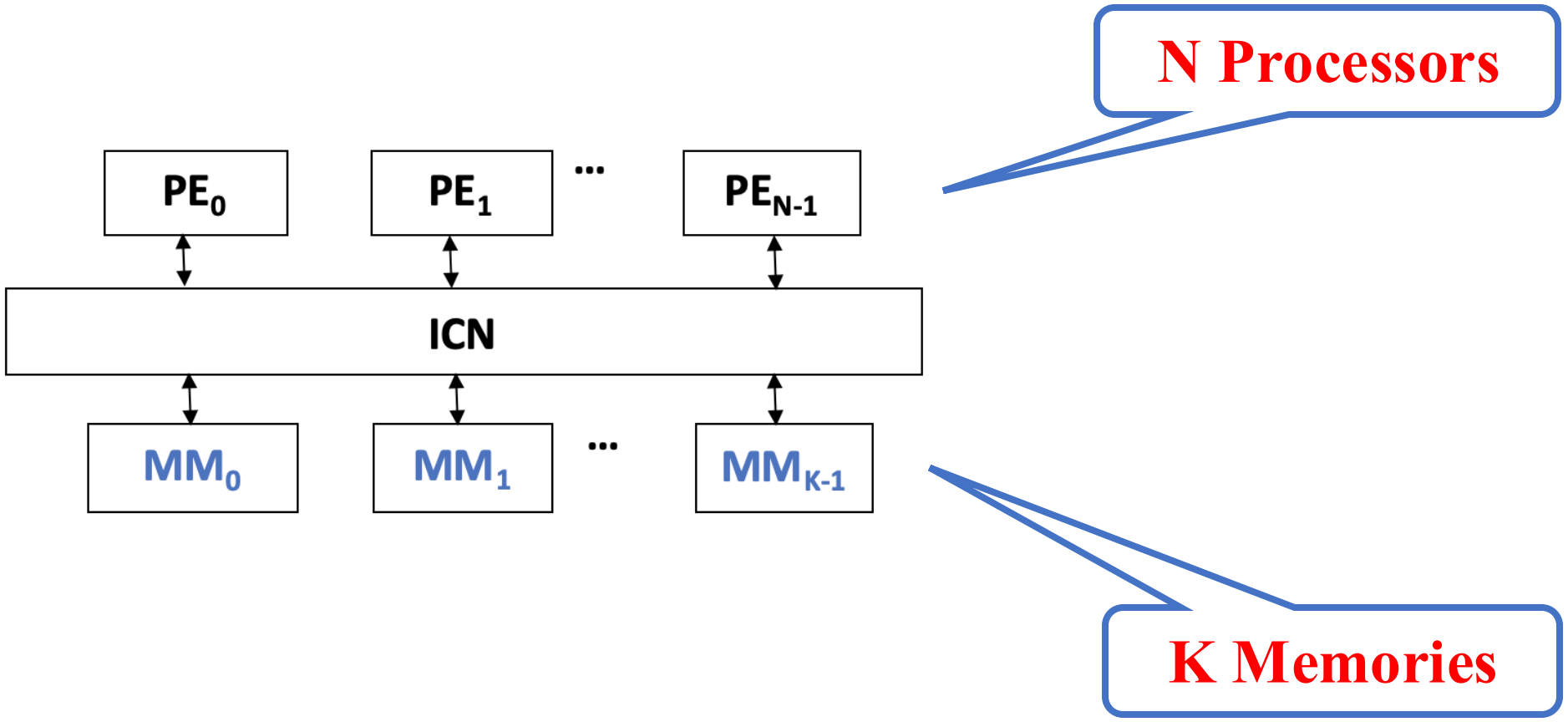


Distributed memory

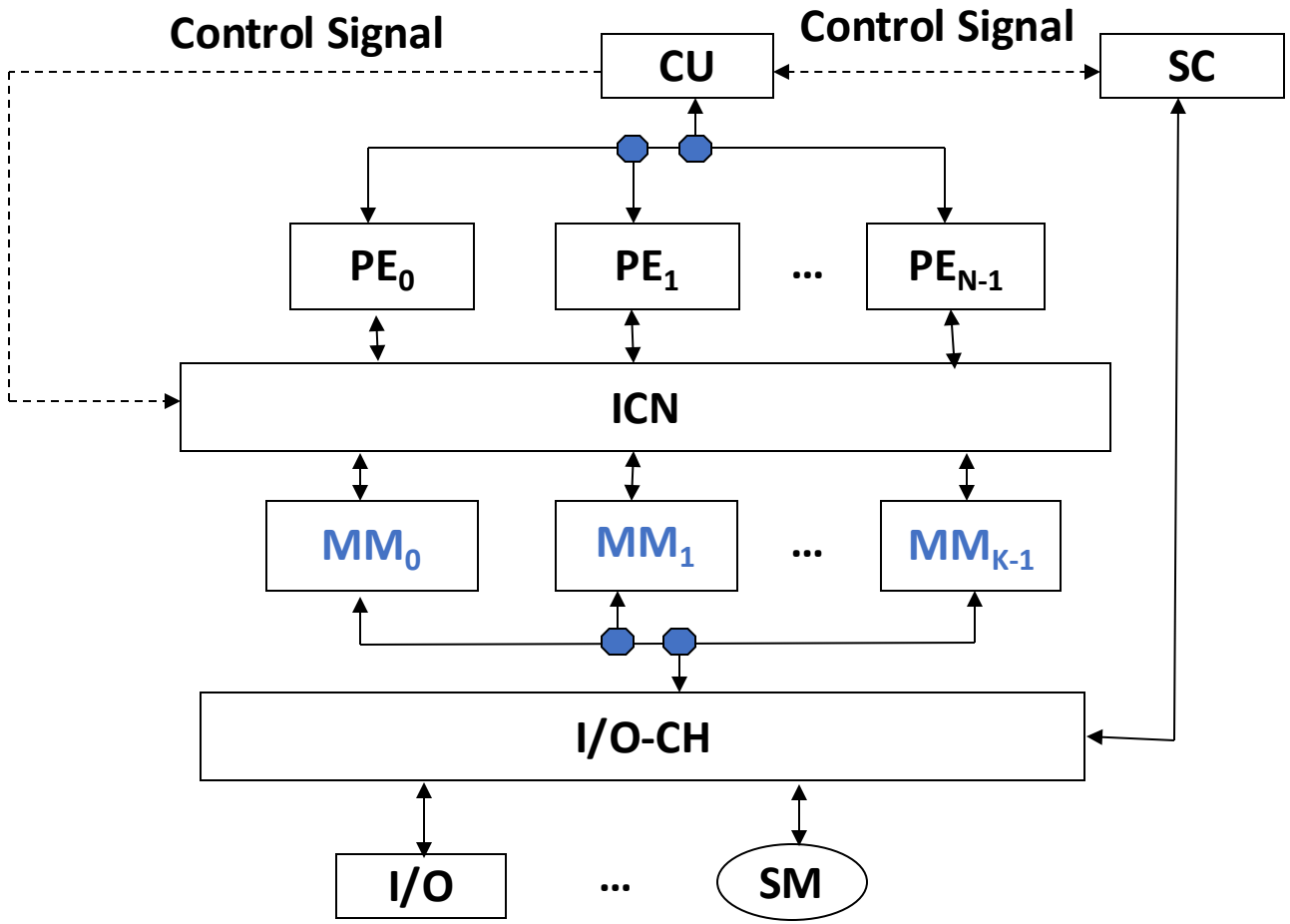


Distributed memory configuration is the mainstream of SIMD array processor.

Centralized shared memory



Centralized shared memory



The difference with distributed memory type:

1. the distribution of memory is different

2. the role of interconnection network is different

Question

- If a direct connection path is required between n processing units, how many pairs of connections are required?

$$P = C_n^2 = \frac{n(n-1)}{2}$$

- Direct path ➔ difficult to achieve
- Indirect path ➔ try to achieve

Parallel computer design

- The communication architecture of the parallel computer is the core of the system
  - The underlying interconnection network;
  - Communication support provided by upper-level languages, software toolkits, compilers, and operating systems.
- The design problem of parallel computer system
  - Interconnection network of parallel computer systems
  - Performance problems of parallel computer systems
  - The software problems of parallel computer systems

Parallel computer design

- The interconnection network is inside the parallel computer system
- Definition: A network composed of switching units according to a certain topology and control mode to realize the interconnection between multiple processors or multiple functional components within a computer system.
- There are many similarities between computer networks in terms of working principles, concepts and terminology. The interconnection networks in some parallel computer systems are just high-speed Ethernet and ATM networks.

Interconnection network

- The interconnection network generally consists of the following five parts:

CPU, memory, interface, link and switch node

  - Interface: It is a device that obtains information from CPU and memory and sends information to another CPU and memory. Typical devices are network interface cards.
  - Link: A physical channel to transmit data bits. The link can be a cable, twisted pair or optical fiber, it can be serial or parallel, and each link has its maximum bandwidth. The link can be simplex half-duplex and full-duplex, the clock mechanism used by the link can be synchronous or asynchronous.
  - Switch node: It is the information exchange and control station of the interconnected network. It is a device with multiple input ports and multiple output ports which is able to perform data buffer storage and path selection.

Some key points

- Topology of interconnection network
  - Static topology
  - Dynamic topology
- Timing mode of interconnection network
  - Synchronization system: Use a unified clock. Such as SIMD array processor
  - Asynchronous system: No uniform clock. Each processor in the system works independently
- Exchange method of interconnection network
  - Circuit switching
  - Packet switching
- Control Strategy of interconnection network
  - Centralized control mode: have a global controller
  - Distributed control mode: no global controller

Classification of interconnection network

- Static network
  - Static Networks refer to networks with fixed connection paths between nodes and this connection remains unchanged during program execution.

- Dynamic network
  - Dynamic Networks are composed of switches, which can dynamically change the connection status according to the requirements of the application. Such as bus, crossbar switch, multi-level switching network, etc.

Goal of interconnection network

- Through a limited number of connection methods, any two PEs can achieve information transmission in one step or a few steps to complete a certain problem-solving algorithm.

Goal of interconnection network

- Single-stage interconnection network: There are only a limited number of connections at the only level to realize information transmission between any two processing units.

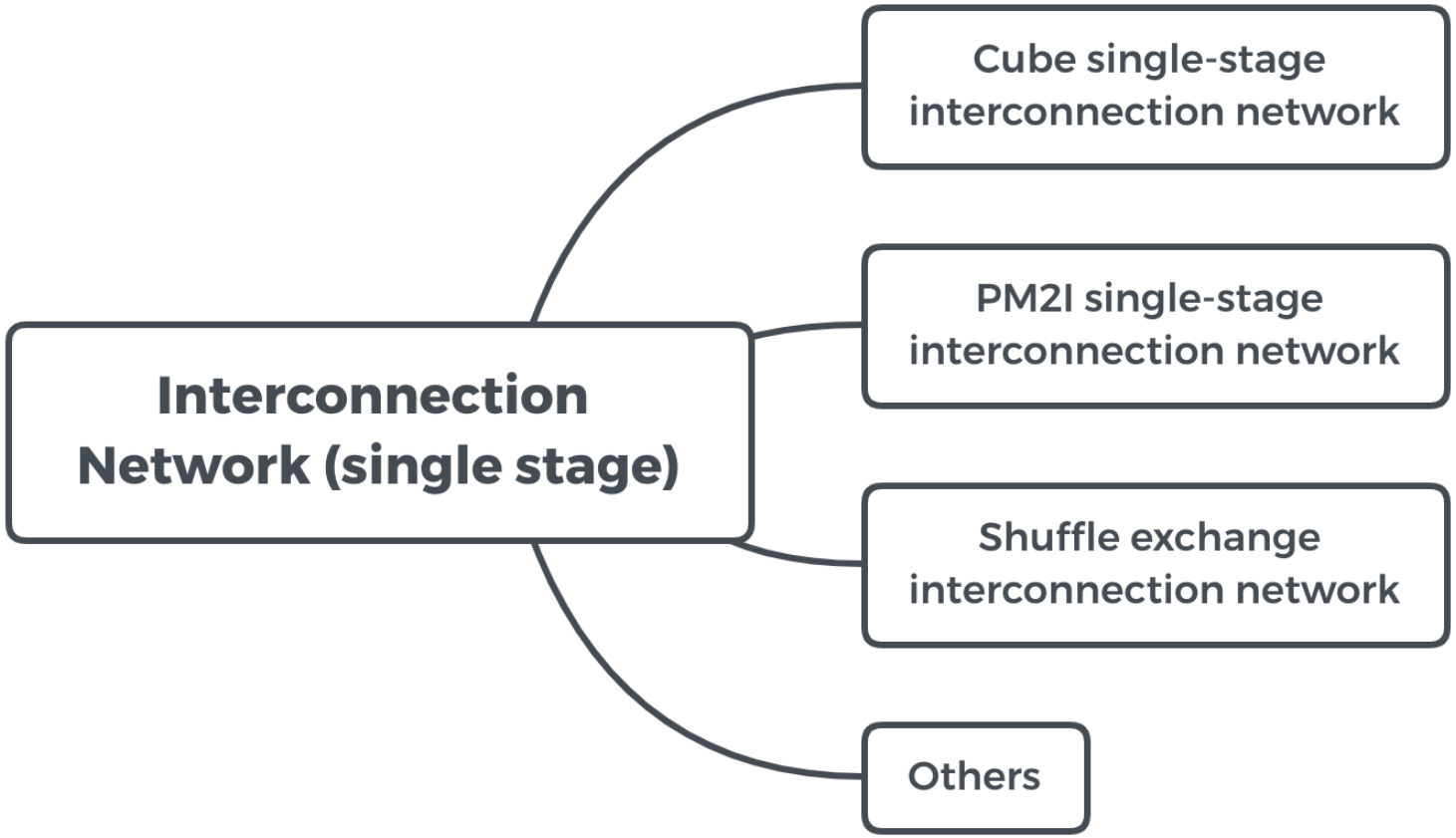
- Multi-stage interconnection network: It is composed of multiple single-level networks in series to realize the connection between any two processing units.

Goal of interconnection network

- For all *N* input terminals (*0, 1, ..., j, ..., N-1*) of the interconnection network, there is a functional correspondence between the input terminal *j* and the output terminal *f(j)*.

- The input *j* and the output *f(j)* generally use binary coding, and the corresponding function law can be found from the binary coding of the two. This corresponding law is the interconnection function.

Single-stage interconnection network



Cube single-stage interconnection network

- The *N* inputs and outputs are encoded with *n*-bit ($n = log_2 N$) binary code $P_{n-1}...P_i...P_1P_0$.

- There are *n* different interconnection functions:

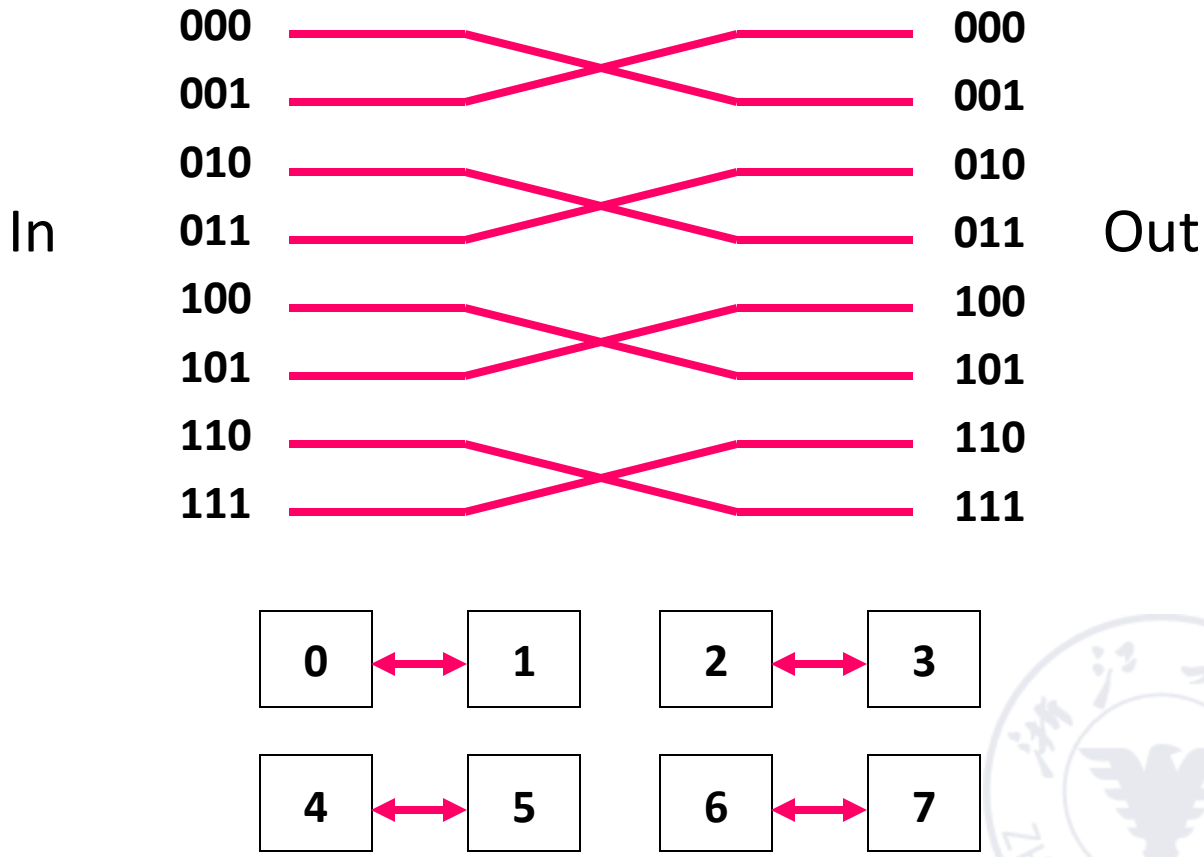
$$Cube_i(P_{n-1}\cdots P_i\cdots P_1P_0) = P_{n-1}\cdots \overline{P_i}\cdots P_1P_0$$

Cube single-stage interconnection network

Example:

- In a cube single-stage interconnection network with 8 PEs, try to find out its interconnection functions.

- $\boldsymbol{cube_0(X_2X_1X_0) = (X_2X_1\overline{X_0})}$

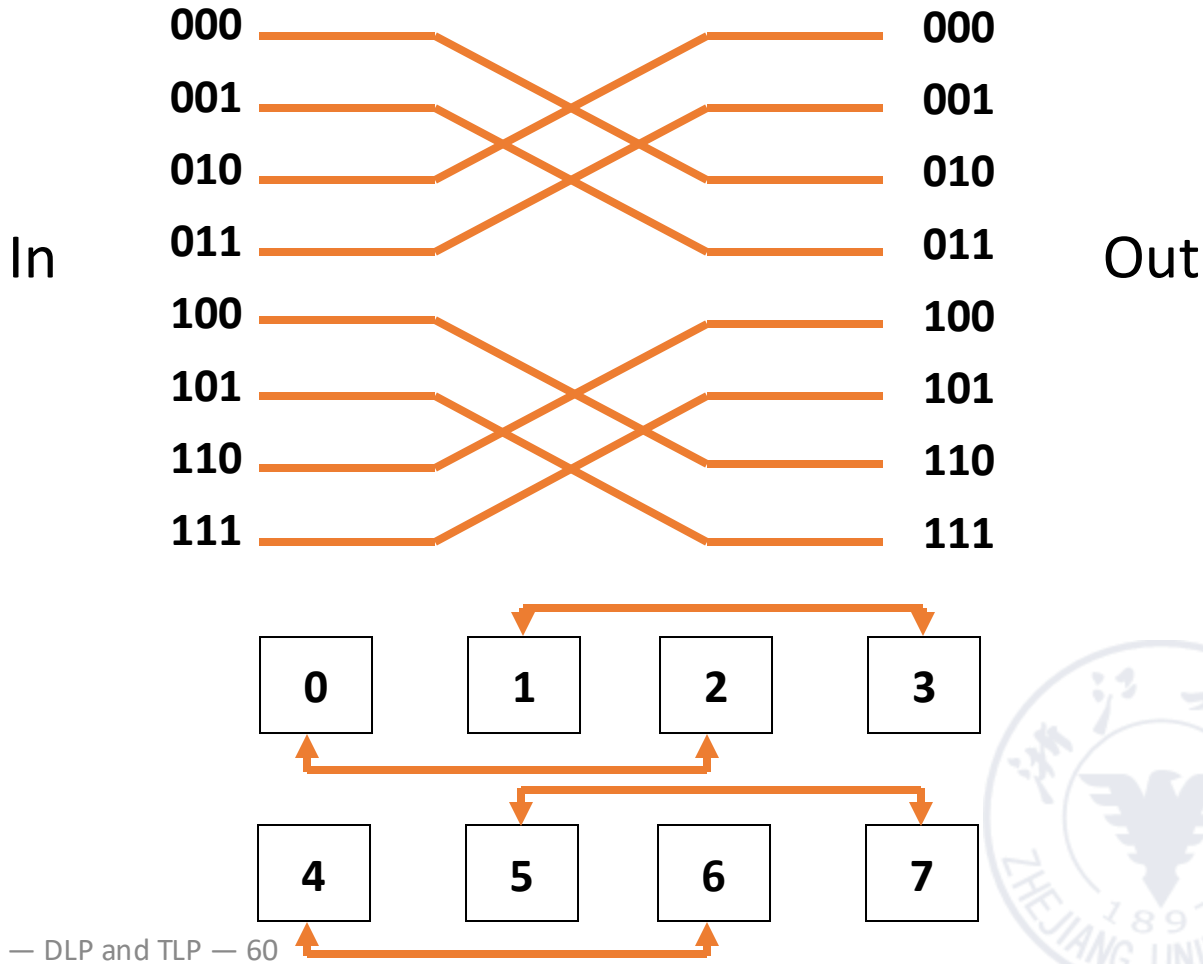


Cube single-stage interconnection network

Example:

- In a cube single-stage interconnection network with 8 PEs, try to find out its interconnection functions.

- $\boldsymbol{cube_1(X_2X_1X_0) = (X_2\overline{X_1}X_0)}$

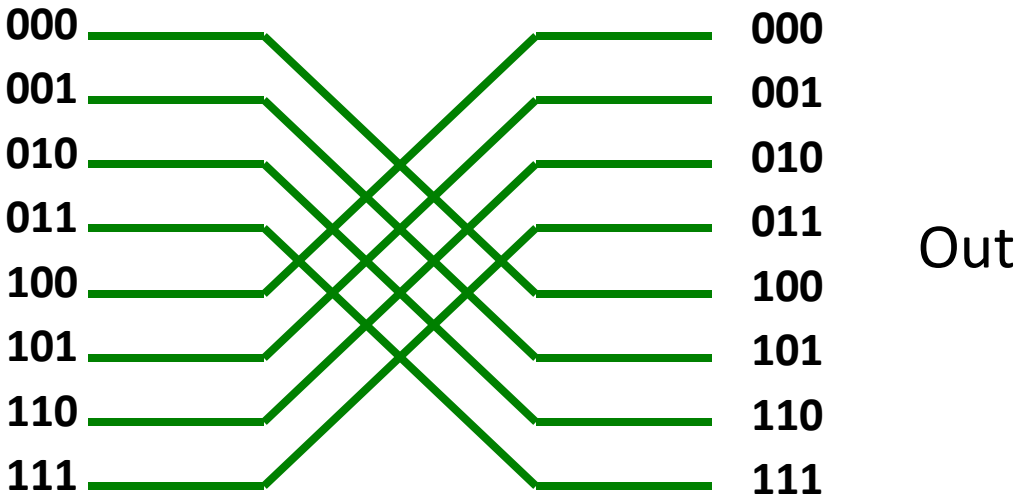


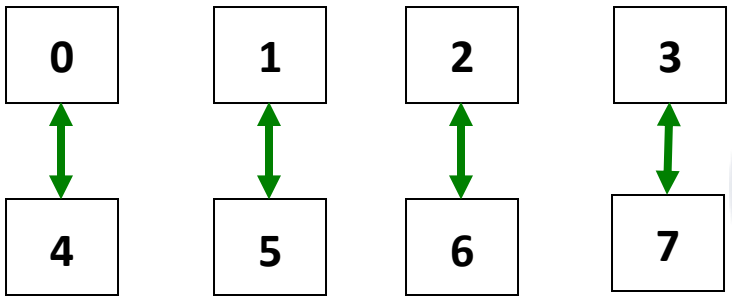
Cube single-stage interconnection network

Example:

- In a cube single-stage interconnection network with 8 PEs, try to find out its interconnection functions.

- $\boldsymbol{cube_2(X_2X_1X_0) = (\overline{X_2}X_1X_0)}$





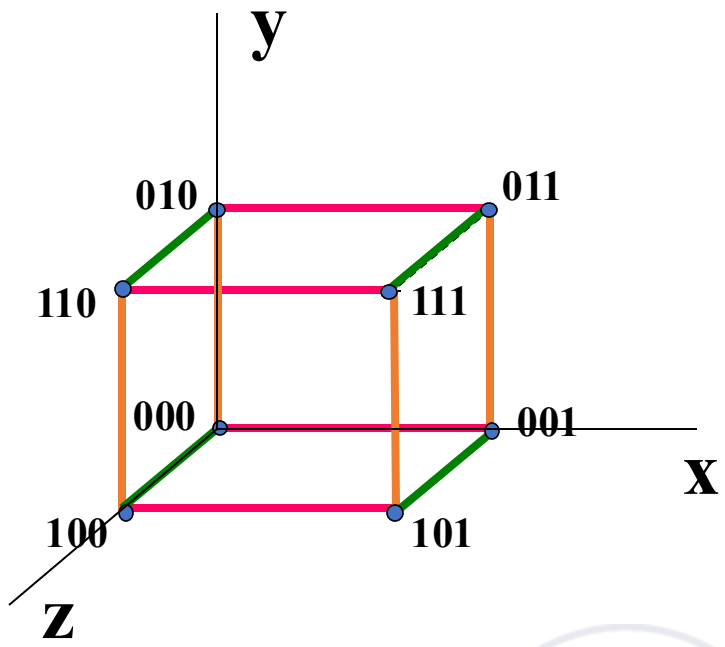
Cube single-stage interconnection network

- The 3D cube can transfer data between any two processing elements at most 3 times.

$Cube_0$

$Cube_1$

$Cube_2$



Hyper cube interconnection network

- When the dimension ***n > 3***, it is called a hyper cube network.

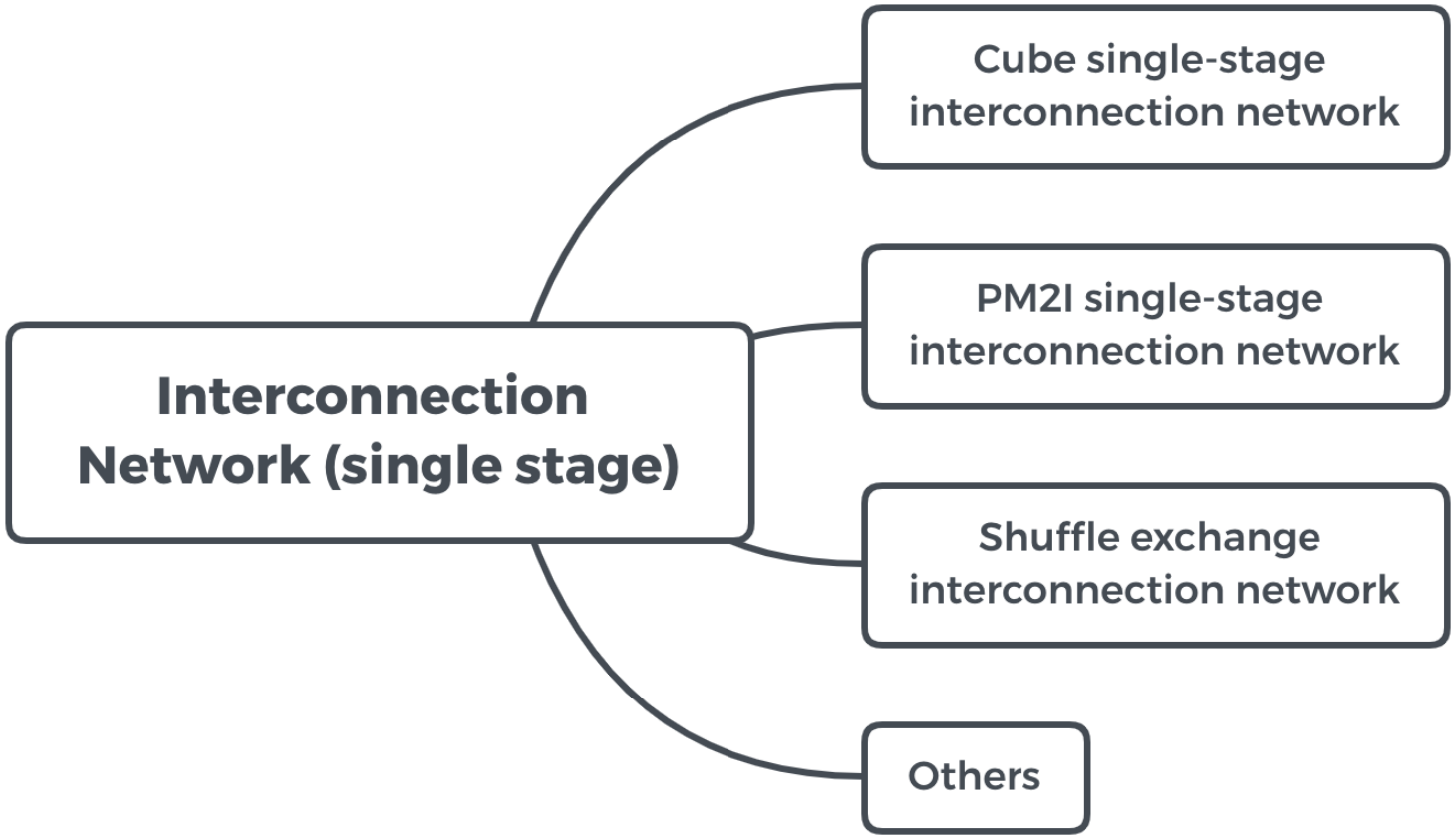
$$Cube_i(P_{n-1}\cdots P_i\cdots P_1P_0) = P_{n-1}\cdots \overline{P_i}\cdots P_1P_0$$

- ***0 ≤ I ≤ n-1***, P_i is the i-th bit of the binary code of the input port number.

The maximum distance of a single-stage n-dimensional cube network is n.

➔ After n transfers at most, data transfer between any two PEs can be realized.

Single-stage interconnection network



Hyper cube interconnection network

- When the dimension ***n > 3***, it is called a hyper cube network.

$$Cube_i(P_{n-1}\cdots P_i\cdots P_1P_0) = P_{n-1}\cdots \overline{P_i}\cdots P_1P_0$$

- ***0 ≤ I ≤ n-1***, P_i is the i-th bit of the binary code of the input port number.

The maximum distance of a single-stage n-dimensional cube network is n.

➔ After n transfers at most, data transfer between any two PEs can be realized.

PM2I single-stage interconnection network

- PM2I (Plus Minus 2^i) single-stage network

- Interconnection Function(2n):

$$\boldsymbol{PM2_{+i}(j) = \left(j + 2^i\right) mod\ N}\ and\ \boldsymbol{PM2_{-i}(j) = \left(j - 2^i\right) mod\ N}$$

- N is the number of interconnection network nodes

$$\boldsymbol{0 \le j \le N - 1, 0 \le i \le log_2 N - 1}$$

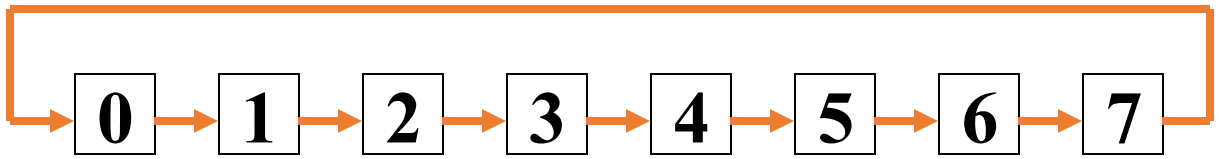
PM2I single-stage interconnection network

Example: ***N = 8***

- Interconnection Function(***i = 0***):

$$PM2_{+0}(j) = \left(j + 2^0\right) mod\ 8$$

- The topology of the interconnection network



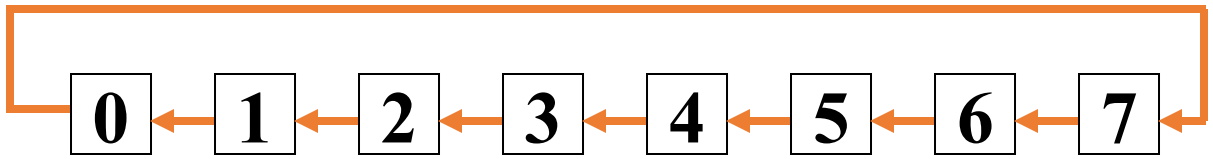
PM2I single-stage interconnection network

Example: ***N = 8***

- Interconnection Function(***i = 0***):

$$PM2_{-0}(j) = \left(j - 2^0\right) mod\ 8$$

- The topology of the interconnection network



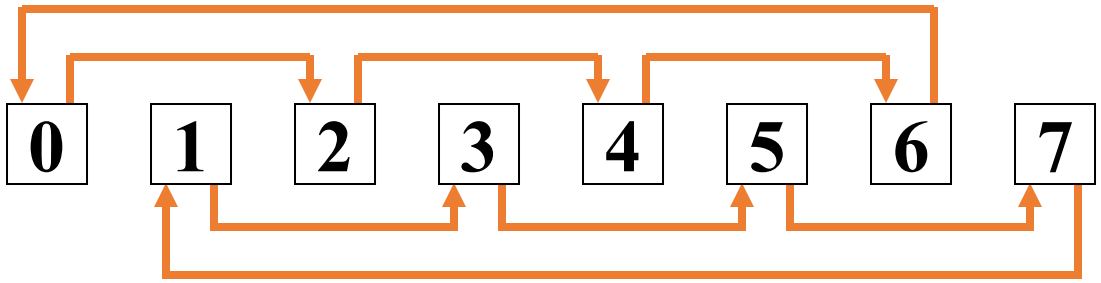
PM2I single-stage interconnection network

Example: ***N = 8***

- Interconnection Function(***i = 1***):

$$PM2_{+1}(j) = \left(j + 2^1\right) mod\ N$$

- The topology of the interconnection network



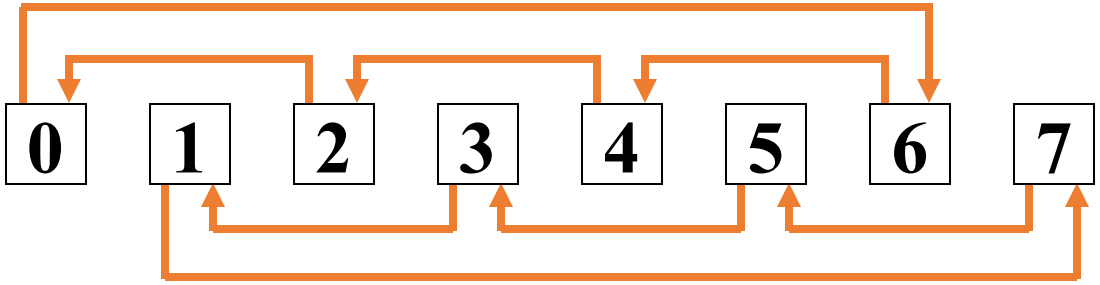
PM2I single-stage interconnection network

Example: ***N = 8***

- Interconnection Function(***i = 1***):

$$PM2_{-1}(j) = \left(j - 2^1\right) mod\ N$$

- The topology of the interconnection network



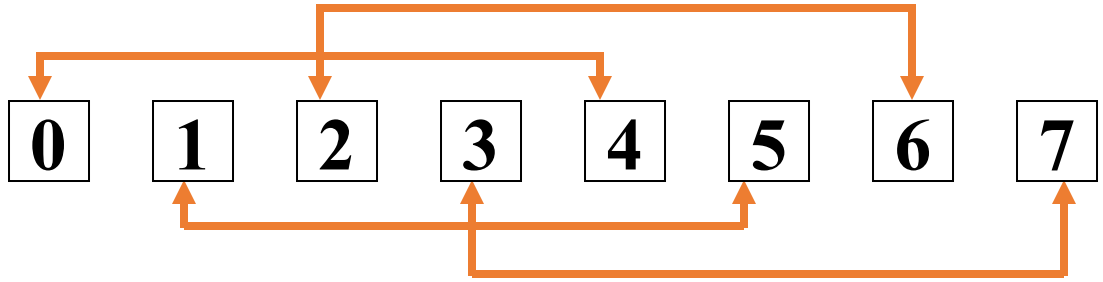
PM2I single-stage interconnection network

Example: ***N = 8***

- Interconnection Function(***i = 2***):

$$PM2_{\pm 2}(j) = \left(j \pm 2^2\right) mod\ N$$

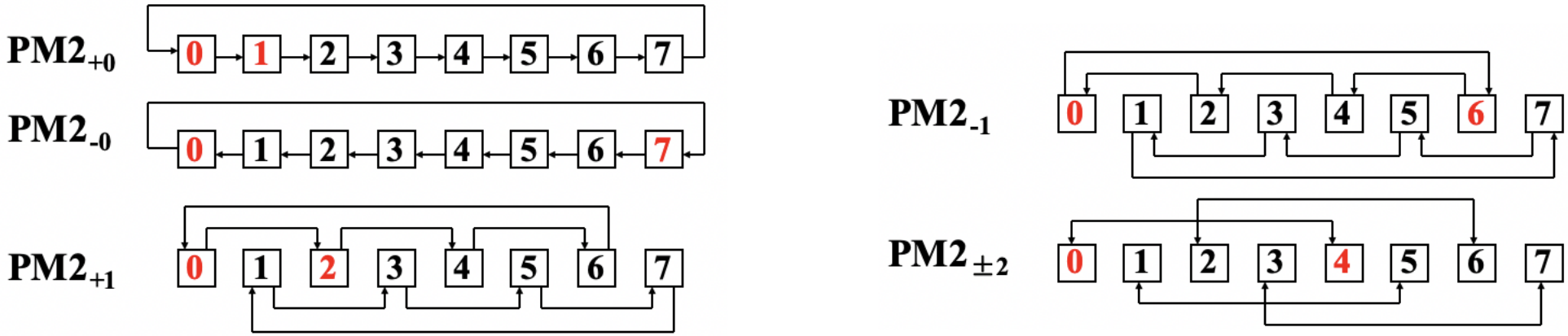
- The topology of the interconnection network



PM2I single-stage interconnection network

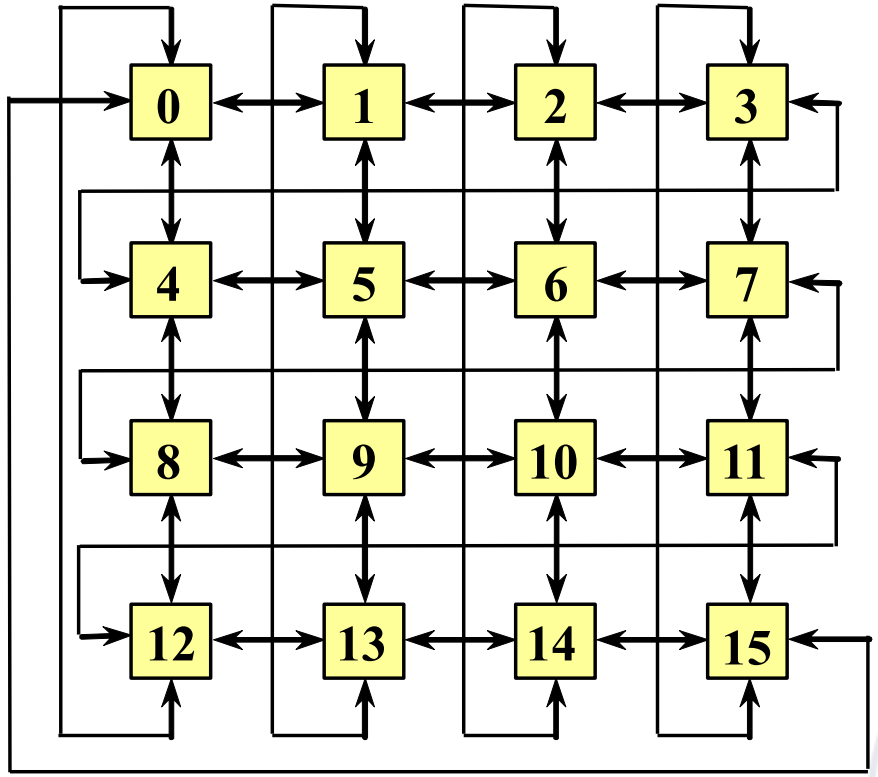
Example: ***N = 8***

- Node 0 can reach nodes 1, 2, 4, 6, 7 in 1 step
- Node 0 can reach nodes 3, 5 in 2 steps

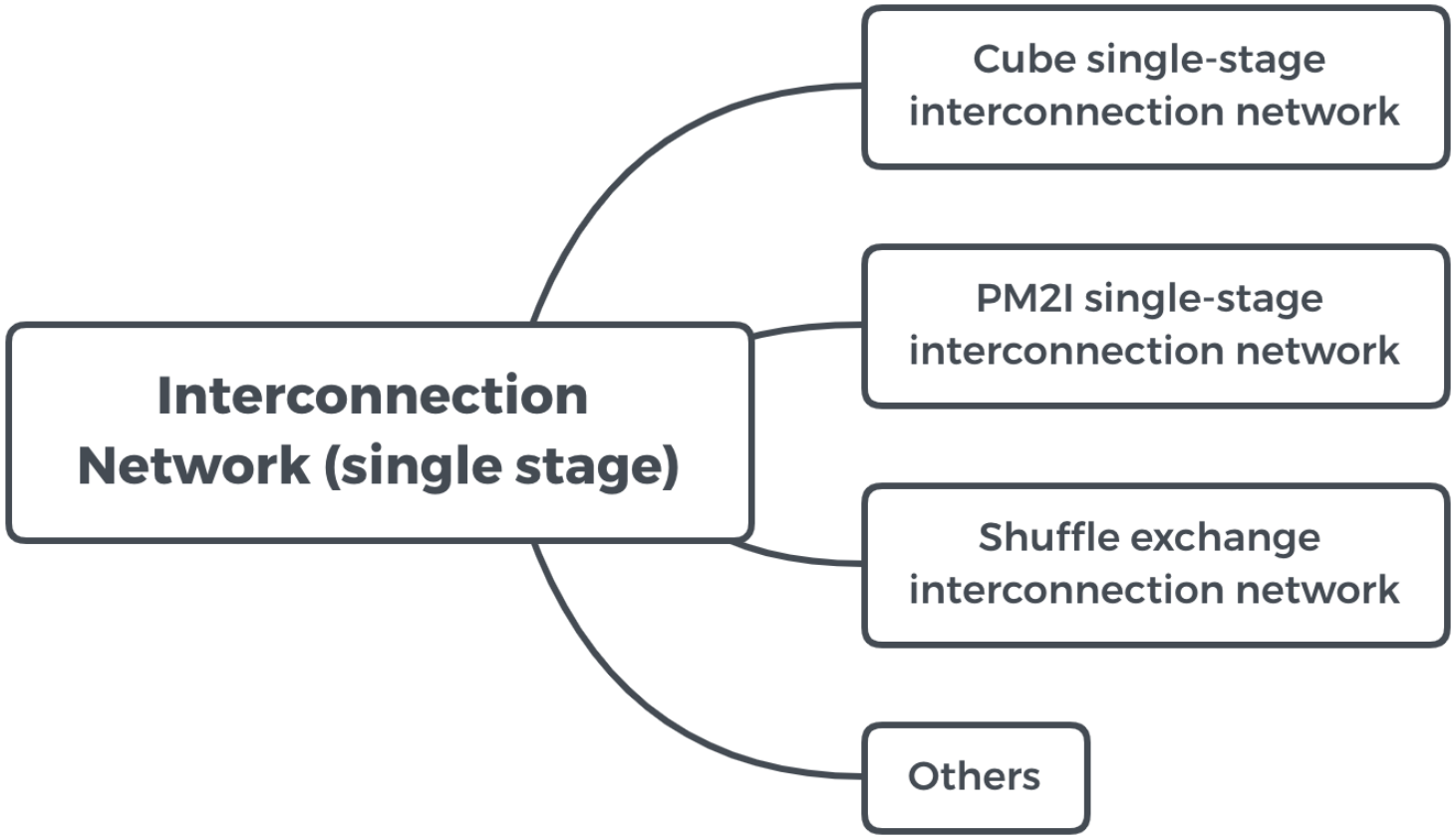


ILLIAC Ⅳ array processor

- It uses $PM2_{\pm 0}$ and $PM2_{\pm n/2}$ to form its interconnection network to realize the up, down, left and right interconnection between the PEs.



Single-stage interconnection network



Shuffle exchange network

- Composed of two parts: **Shuffle** + **Exchange**

- N-dimension shuffle function:

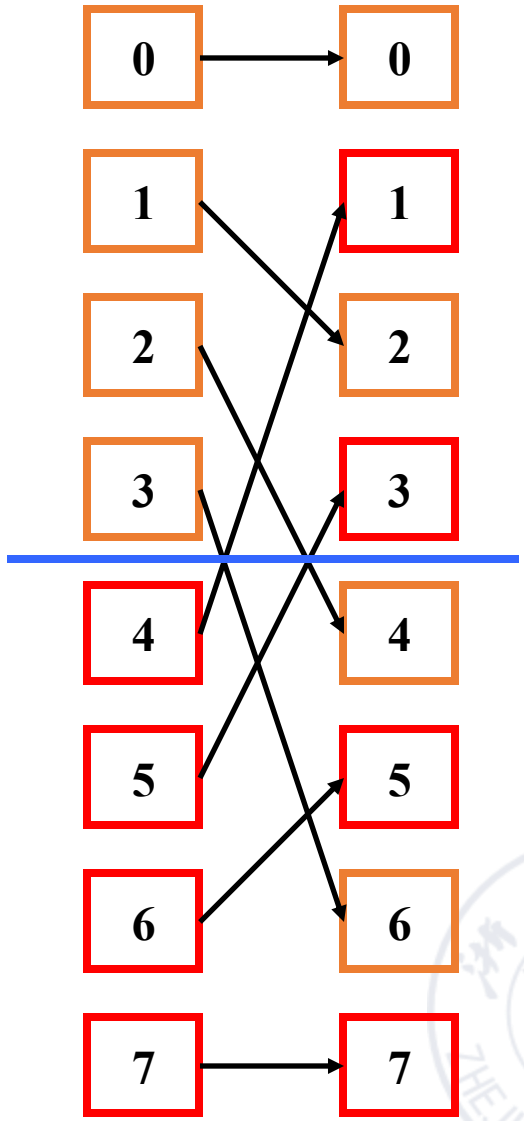
$$\boldsymbol{shuffle}(P_{n-1}P_{n-2}\cdots P_1P_0) = P_{n-2}\cdots P_1P_0P_{n-1}$$

$$n = \log_2 N, P_{n-1}P_{n-2}\ldots P_1P_0 \textit{ are binary codes for the input number}$$

Example: 8 PEs

Shuffle function: $\boldsymbol{shuffle}(P_2P_1P_0) = P_1P_0P_2$

Initial Code	Before Shuffle	After 1st Shuffle	Connected PE
0	000	000	0
1	001	010	2
2	010	100	4
3	011	110	6
4	100	001	1
5	101	011	3
6	110	101	5
7	111	111	7



Example: 8 PEs

Shuffle function: $\boldsymbol{shuffle}(P_2P_1P_0) = P_1P_0P_2$

Initial Code	Before Shuffle	After 1st Shuffle	Connected PE	After 2nd Shuffle	Connected PE
0	000	000	0	000	0
1	001	010	2	100	4
2	010	100	4	001	1
3	011	110	6	101	5
4	100	001	1	010	2
5	101	011	3	110	6
6	110	101	5	011	3
7	111	111	7	111	7

Example: 8 PEs

Shuffle function: $\boldsymbol{shuffle}(\boldsymbol{P_2P_1P_0}) = \boldsymbol{P_1P_0P_2}$

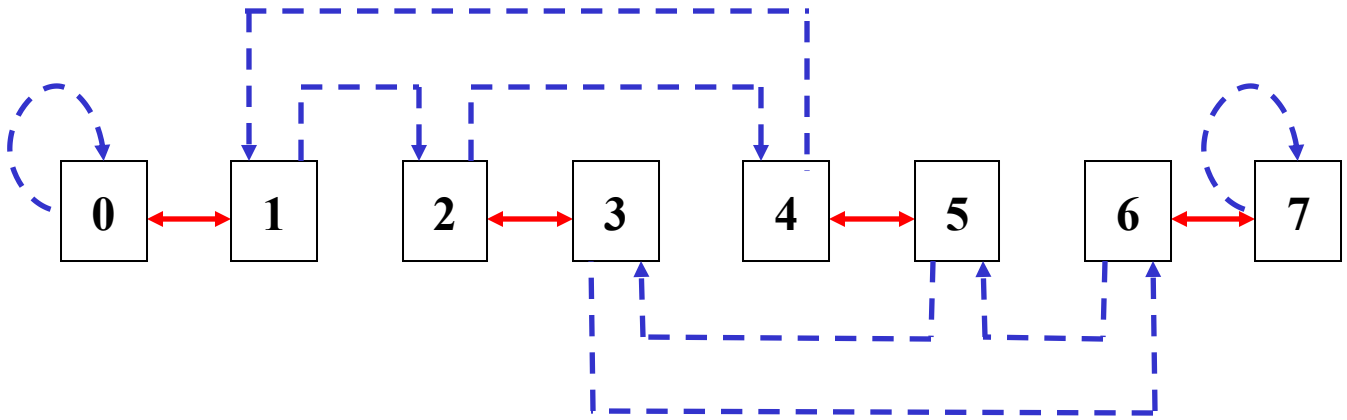
Initial Code	Before Shuffle	After 2nd Shuffle	Connected PE	After 3rd Shuffle	Connected PE
0	000	000	0	000	0
1	001	100	4	001	1
2	010	001	1	010	2
3	011	101	5	011	3
4	100	010	2	100	4
5	101	110	6	101	5
6	110	011	3	110	6
7	111	111	7	111	7

Shuffle exchange network

An important feature of the Shuffle function:

- After ***N*** times of shuffling, all ***N*** PEs are restored to the initial arrangement order.

$$shuffle(P_2P_1P_0) = P_1P_0P_2$$



3 exchanges and 2 shuffles

Shuffle exchange network

The maximum distance of shuffle exchange network

- From the nodes numbered all “0” to all “1”, the data transmission needs to go through at most:

n exchanges and n-1 shuffles

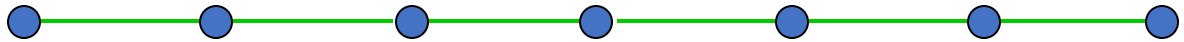
- Maximum distance: 2n-1

Features of single-stage interconnection networks

- Simple structure and low cost
- Flexible connection to meet the needs of algorithms and applications
- The number of transfer steps is small, and the array operation speed is improved
- Regularity and modularity are better to enhance the scalability of the system
- Facilitate large-scale integration

Linear array

- For a linear array of N nodes, there are ***N-1*** links, the diameter is ***N-1*** (the maximum distance between any two points), the degree is ***2***, asymmetry, and the equal division width is ***1***.
- When N is large, the communication efficiency is very low.

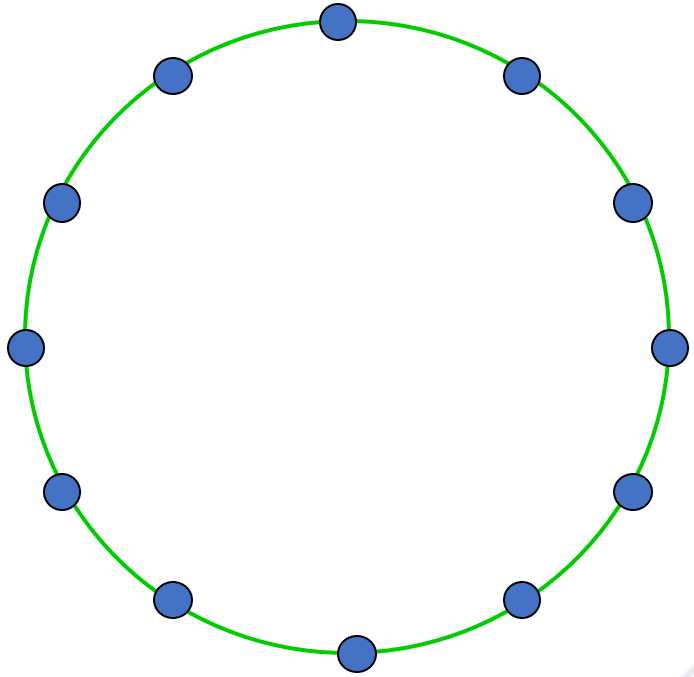


Circular array

- For a ring of N nodes, consider the data transfer direction of adjacent nodes:

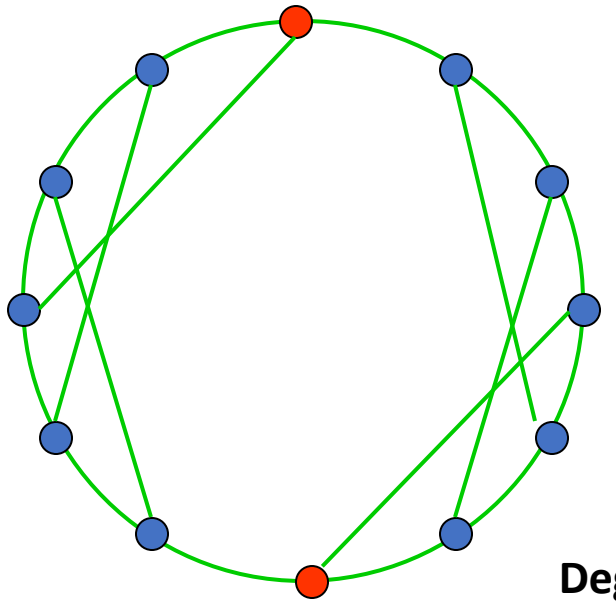
- Two-way ring: The number of links is ***N***, the diameter is $\lfloor N/2 \rfloor$, the degree is ***2***, symmetrical, and the equal width is ***2***.

- Unidirectional ring: The number of links is ***N***, the diameter is ***N-1***, the degree is ***2***, symmetrical, and the equal width is ***2***.

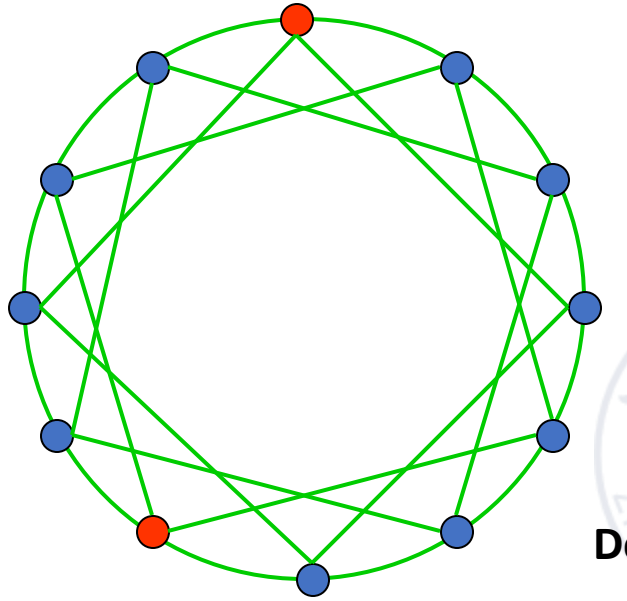


Loop with chord array

- For the two-way loop with chord with 12 nodes in the figure
- Node degree is 3: The number of links is 18
- Node degree is 4: The number of links is 24



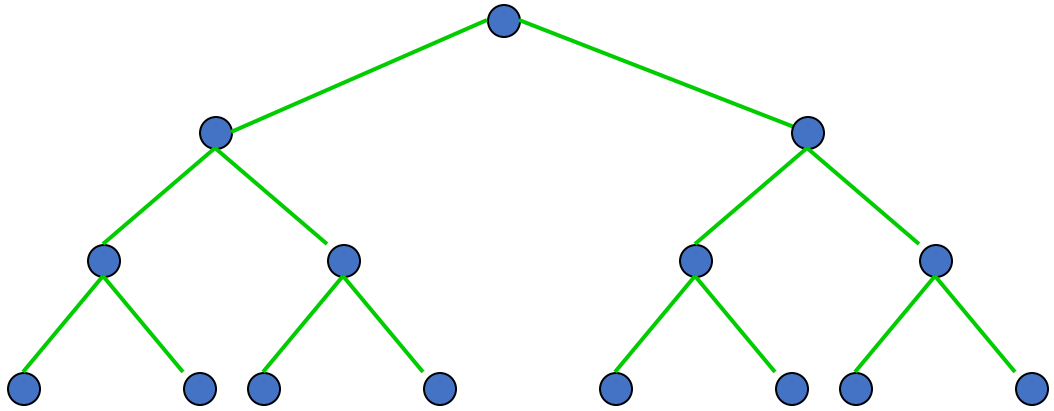
Degree = 3



Degree = 4

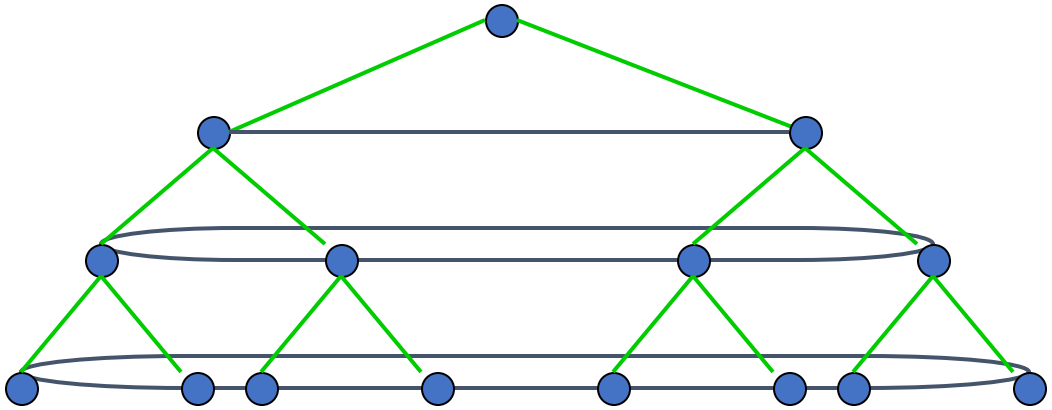
Tree array

- A K-level complete binary tree should have ***N = 2^K-1*** nodes, the maximum node degree is ***3***, and the diameter is ***2(K-1)*** (that is, any leaf node on the right to any leaf node on the left), asymmetrical, equal width is ***1***.

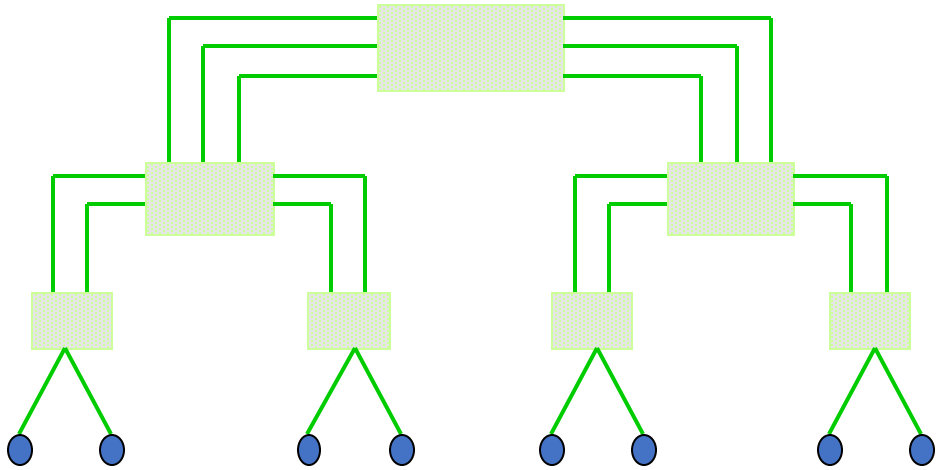


K = 4

Extensions of tree array



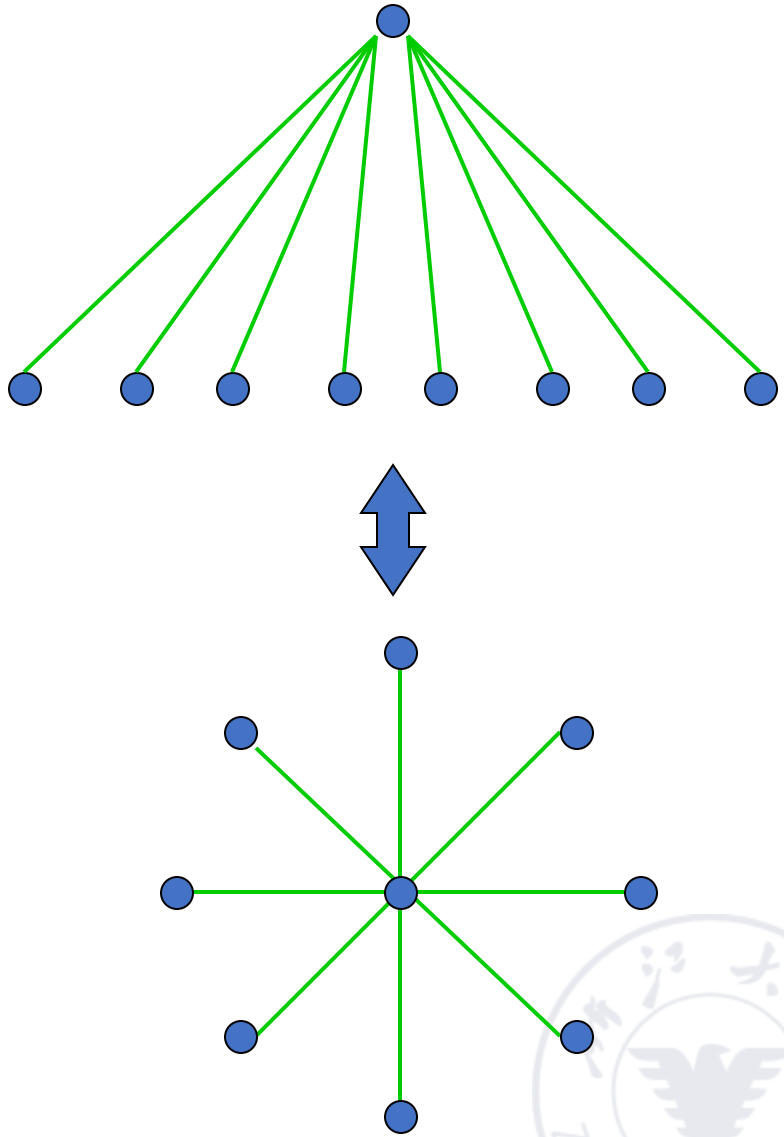
Tree with loop



Binary fat tree

Star array

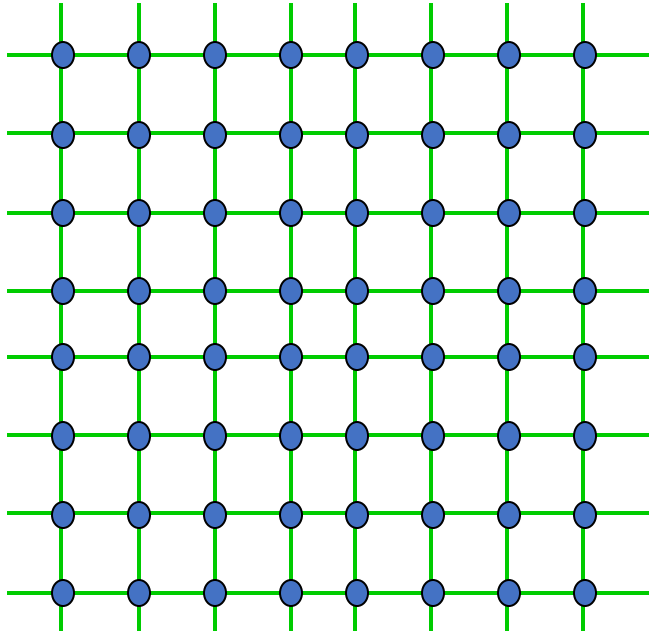
- The star array is actually a two-level tree (as shown on the right).
- A star network with N nodes has ***N-1*** links, a diameter of ***2***, a maximum node degree of ***N-1***, asymmetrical, and an equal division width of ***1***.



Grid

- An ***r*r*** network with N nodes has ***2N-2r*** links, a diameter of ***2(r-1)***, a node degree of ***4***, asymmetrical, and an equal division width of ***r***.

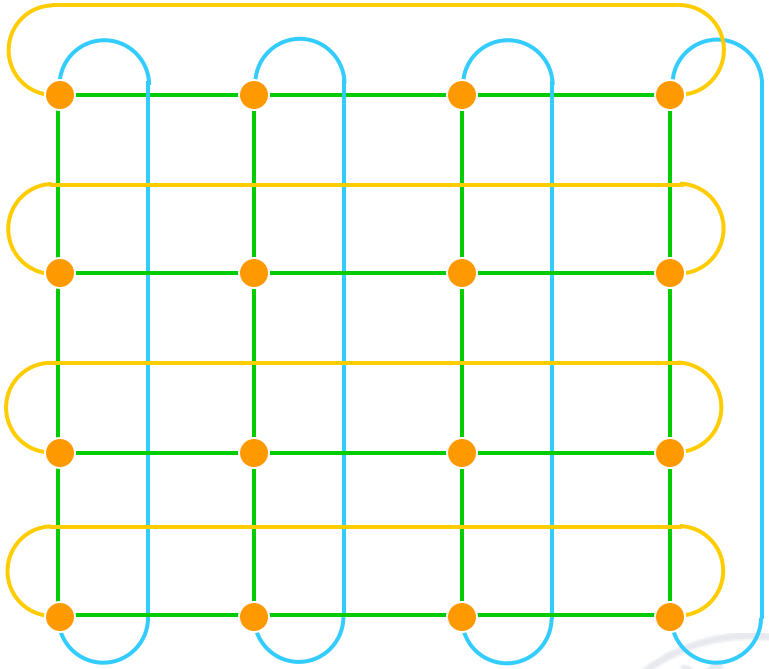
- $\boldsymbol{r = \sqrt{N}}$



2D torus

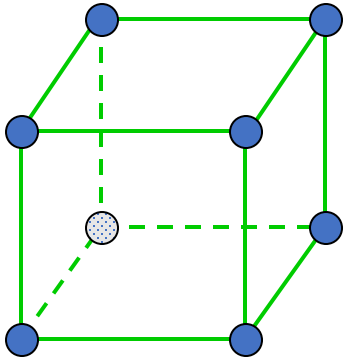
- An ***r*r*** network with N nodes has ***2N*** links, a diameter of $2\lfloor r/2 \rfloor$, a node degree of ***4***, asymmetrical.

- $\boldsymbol{r = \sqrt{N}}$

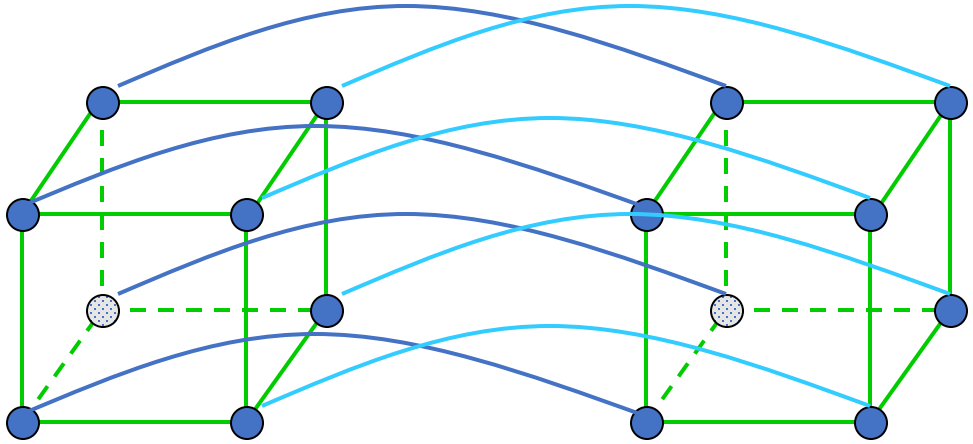


Hypercube

- An n-cube consists of $N = 2^n$ nodes, which are distributed in n dimensions, with two nodes in each dimension. The diameter is n, the degree is n, symmetrical.



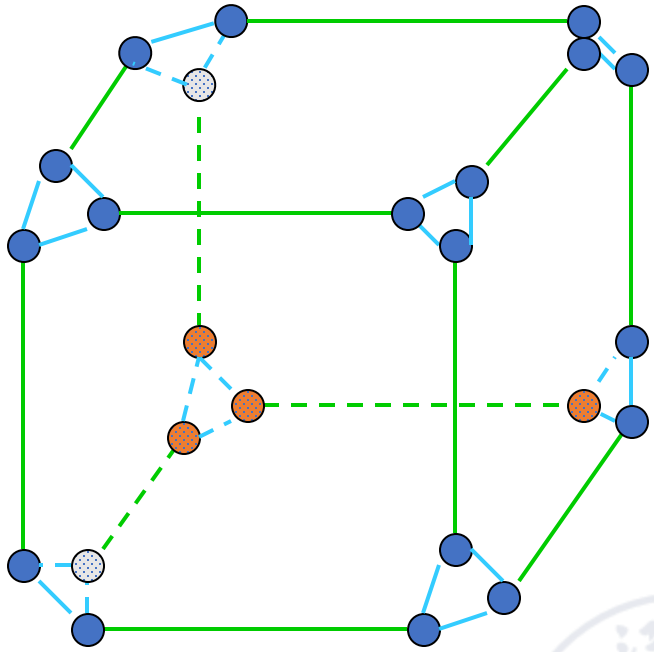
3-cube



4-cube

Cube with loop

- An n-cube with loop is composed of $N = 2^n$ node rings, each node ring is a ring with n nodes, so the total number of nodes is $n2^n$, the node degree is 3, and it is symmetric.



Summary

	Scale	Degree	Diameter	Width	Symmetry	Link
Linear	*N*	*2*	*N-1*	*1*	*No*	*N-1*
Circular	*N*	*2*	$\lfloor N/2 \rfloor$	*1*	*Yes*	*N*
Binary tree	*N*	*3*	$2(\lceil logN \rceil - 1)$	*1*	*No*	*N-1*
Star	*N*	*N-1*	*2*	$N/2$	*No*	*N-1*
Grid	*N*	*4*	$2(\sqrt{N} - 1)$	$\sqrt{N}$	*No*	$2(N-\sqrt{N})$
2D torus	*N*	*4*	$2\lfloor \sqrt{N}/2 \rfloor$	$2\sqrt{N}$	*Yes*	*2N*
Hypercube	$N = 2^n$	*N*	*n*	*N/2*	*Yes*	*nN/2*
Cube with loop	$N = k2^k$	*3*	$2k-1+\lceil k/2 \rceil$	$N/2^k$	*Yes*	*3N/2*

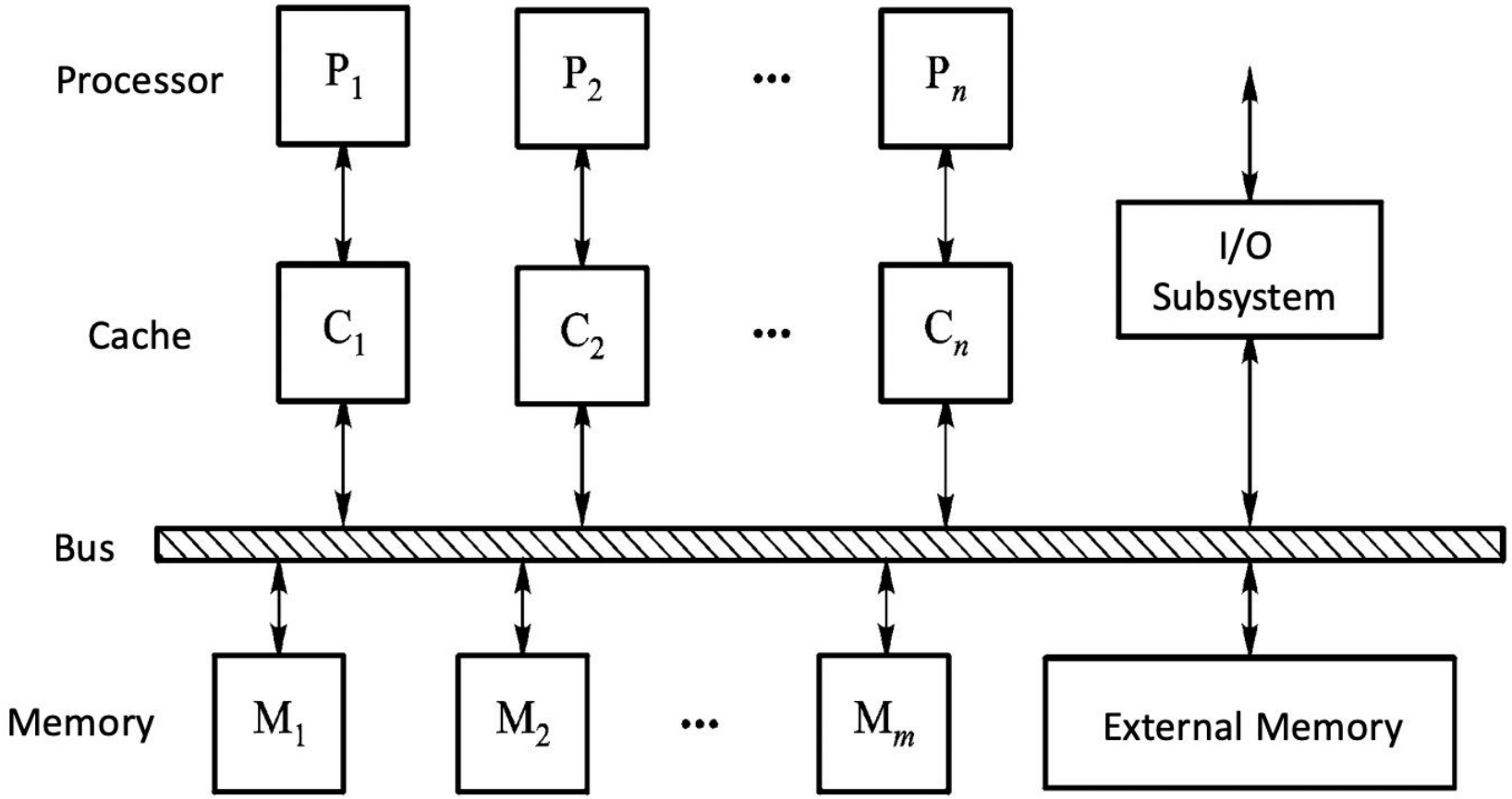
Network characteristics

- The connection in the dynamic network is not fixed and can be changed as needed during the execution of the program.
- The switching elements of the network are active, and the link can be reconstructed by setting the state of these switches.
- Only the switching elements on the network boundary can be connected to the processor.
- Dynamic networks mainly include buses, crossbar switches, and multi-stage interconnection networks.

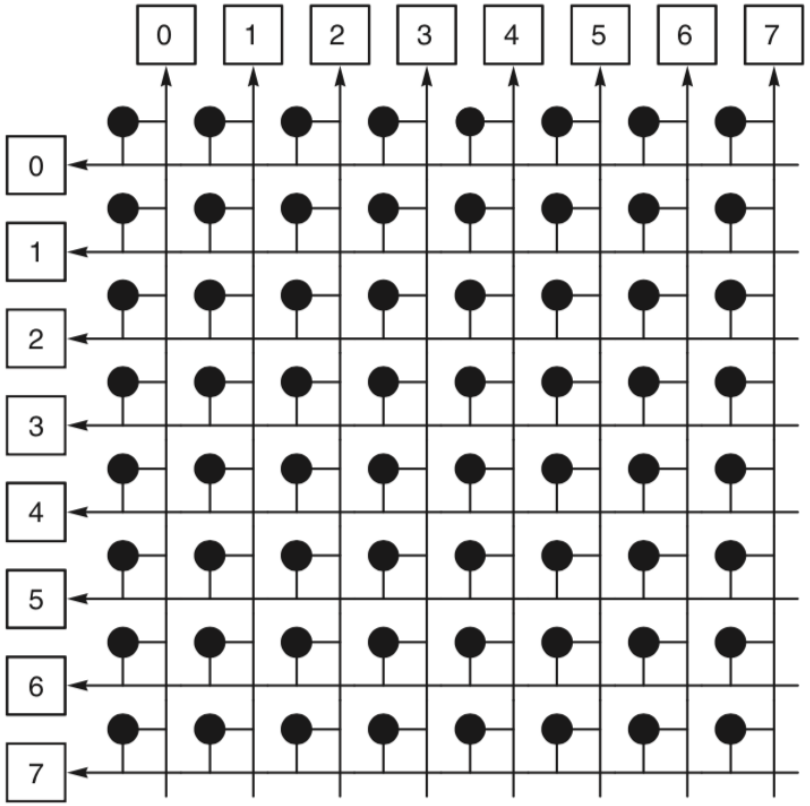
- Bus
  - The bus is actually a set of wires and sockets that connect peripheral devices such as processors, memories, and I/O.
  - It can only be used to transfer data between a pair of source and destination at a certain time.
  - When there are multiple pairs of source and destination requests to use the bus, bus arbitration is required. When the number of CPUs is large, the bus contention is serious (<=32).

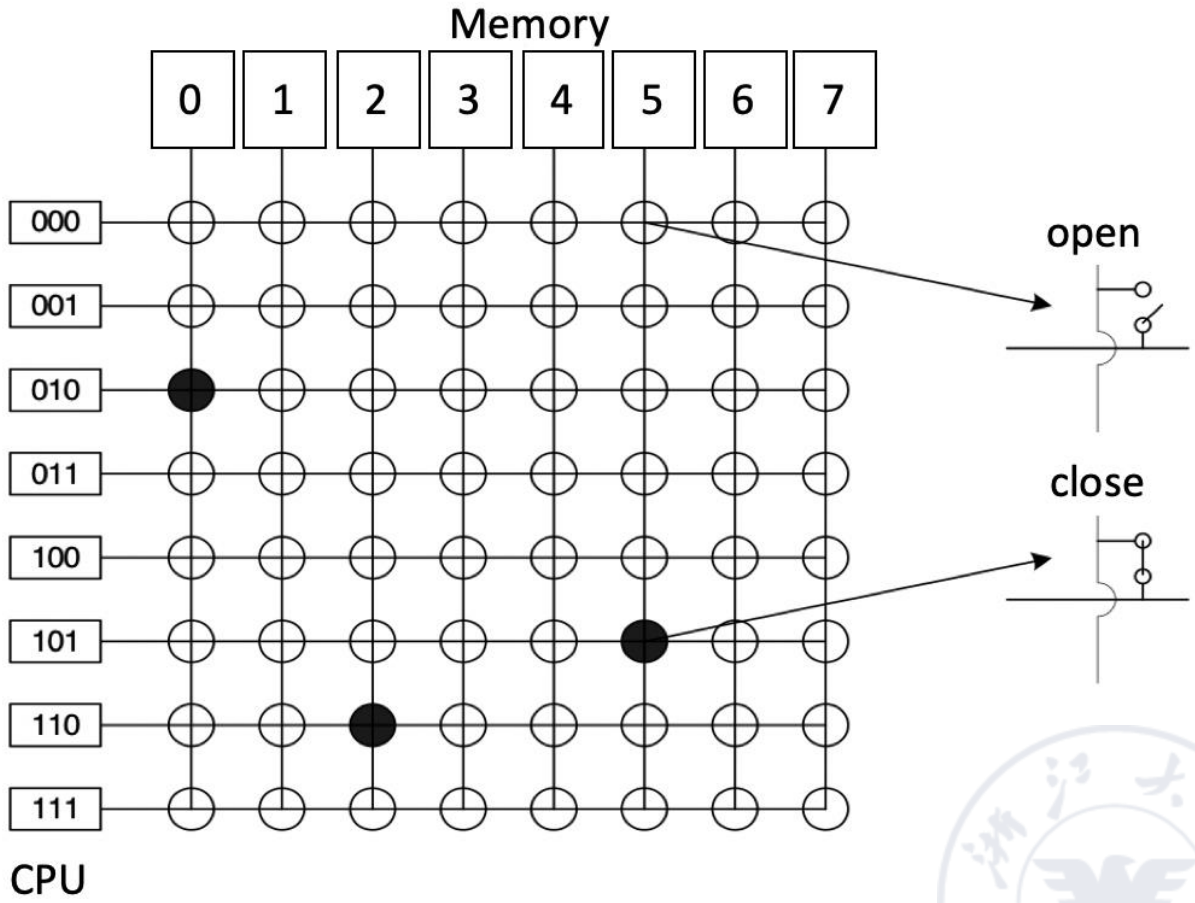
- The difference between linear array and bus
  - Linear array: Allows different source and destination nodes to concurrently use different parts of the system.
  - Bus: Realize time-division characteristics by switching many nodes connected to it. Only one pair of nodes are transmitting data at the same time.

A multiprocessor system connected by bus



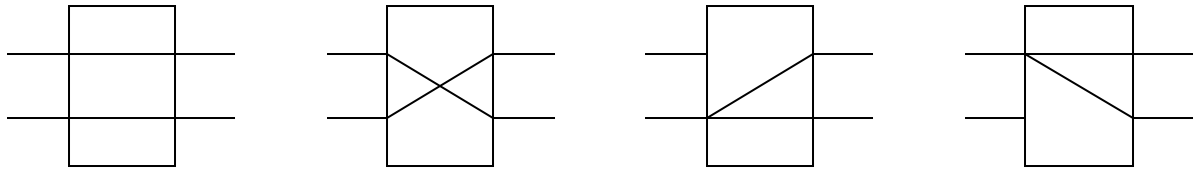
Crosspoint switches





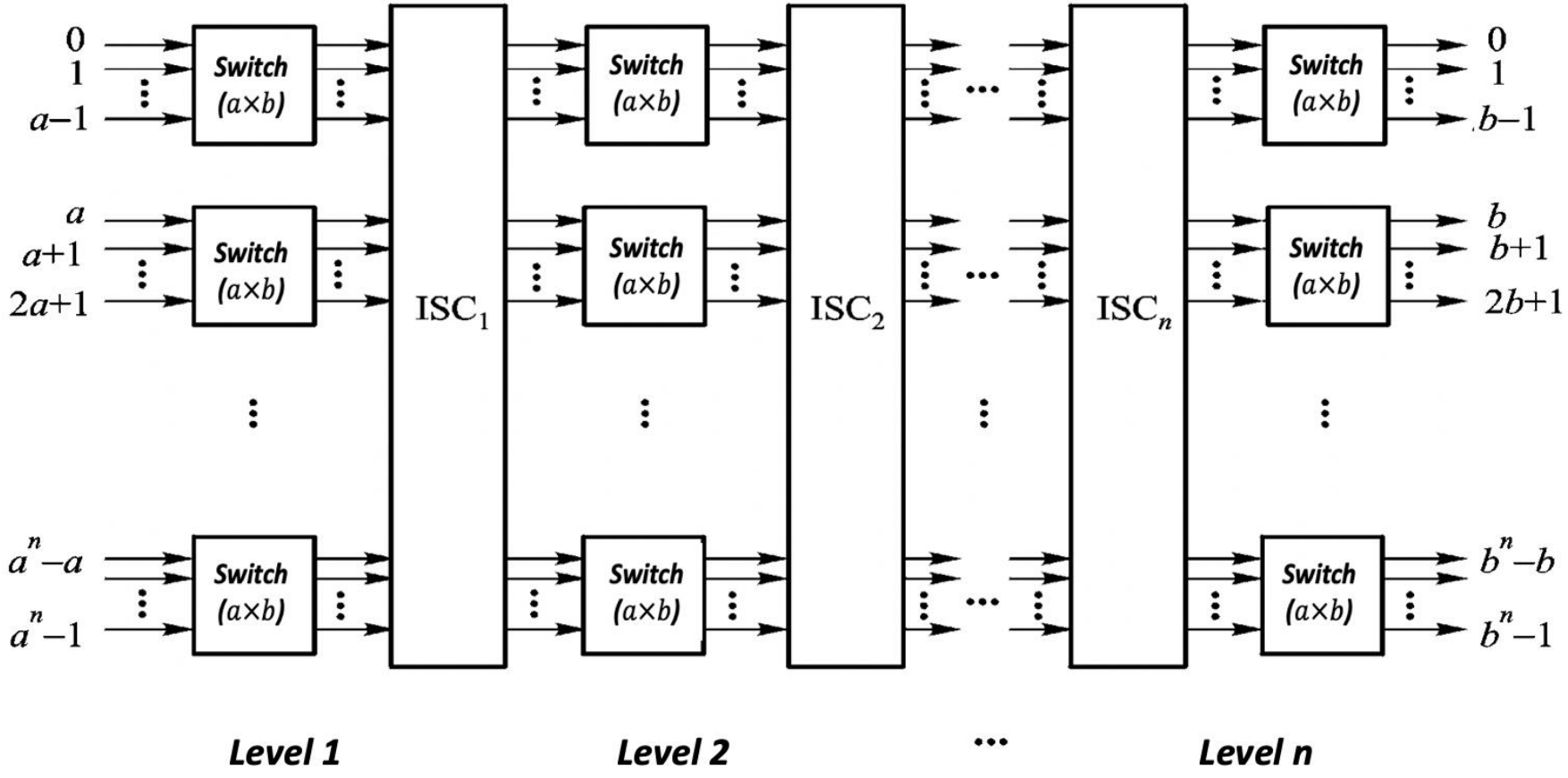
Multi-stage interconnection network

- Switch unit: a switch unit with **m** inputs and **m** outputs is recorded as a switch unit of **m×m**, m = 2k. Common ones are **2×2**, **4×4**, **8×8** and so on.
- According to the function of the switch unit, **2×2** can be divided into two-function and four-function switches.



Module Size	Legal Status	Exchange Connection
2×2	2	2
4×4	256	24
8×8	16,777,216	40,320
N×N	N^N	N!

Multi-stage interconnection network



Multi-stage interconnection network

In order to realize the connection between any PEs, you can use:

- Circular interconnection network:
  - A single-stage network can be used to cycle through multiple times
- Multi-level interconnection network:
  - Connect multiple single-stage interconnection networks in series
- Multi-level circular interconnection network:
  - Based on multi-level interconnection network, recycle through multiple times

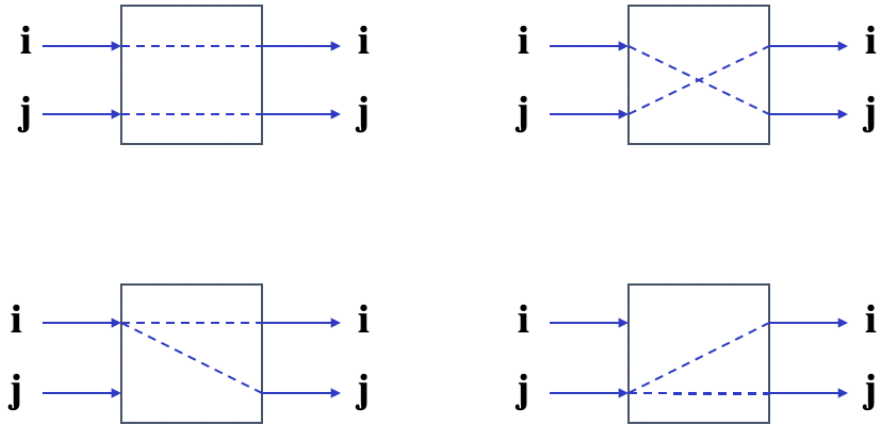
Multi-stage interconnection network

The differences between different multi-level networks are:

- Switch function
- Switch control method
- Topology

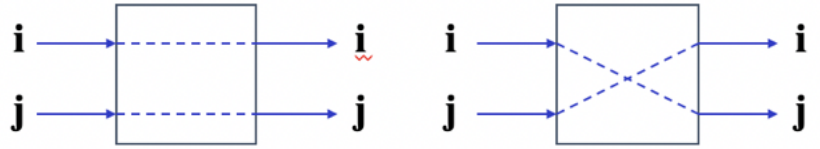
Switching unit

- A switching unit with two inputs and two outputs is the basic component of various multi-level interconnection networks.
- Regardless of input and output, the one located above are represented by ***i***, and the one located below are represented by ***j***.
- The statuses of switching unit:
  - Straight
  - Exchange
  - Upper broadcast
  - Lower broadcast

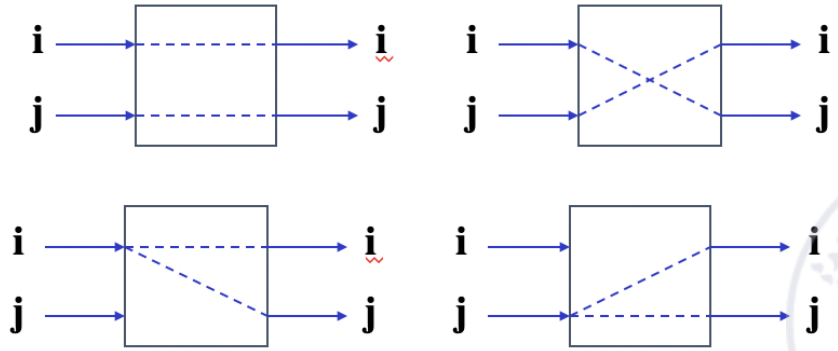


Multi-stage interconnection network

- Two-function switching unit
  - Straight and exchange unit only

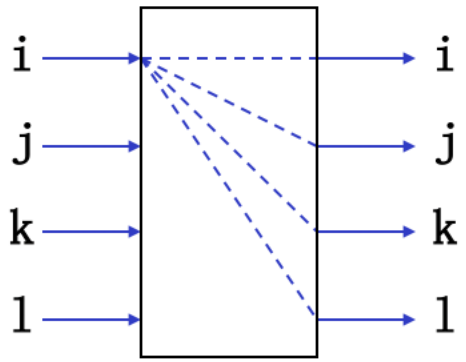


- Four-function switching unit
  - Include four basic functions

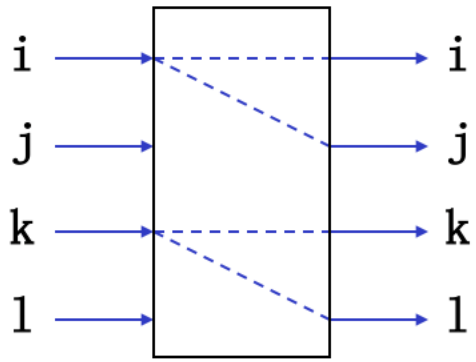


Multi-stage interconnection network

- Multi-end switching unit
  - Add broadcast and multicast module



Broadcast



Multicast

Multi-stage interconnection network

- Topology: The mode in which the input/output ends of the switches at all levels are connected to each other.
  - Multi-stage cube
  - Multi-stage shuffle exchange
  - Multi-stage PM2I
  - The combination of the above three networks

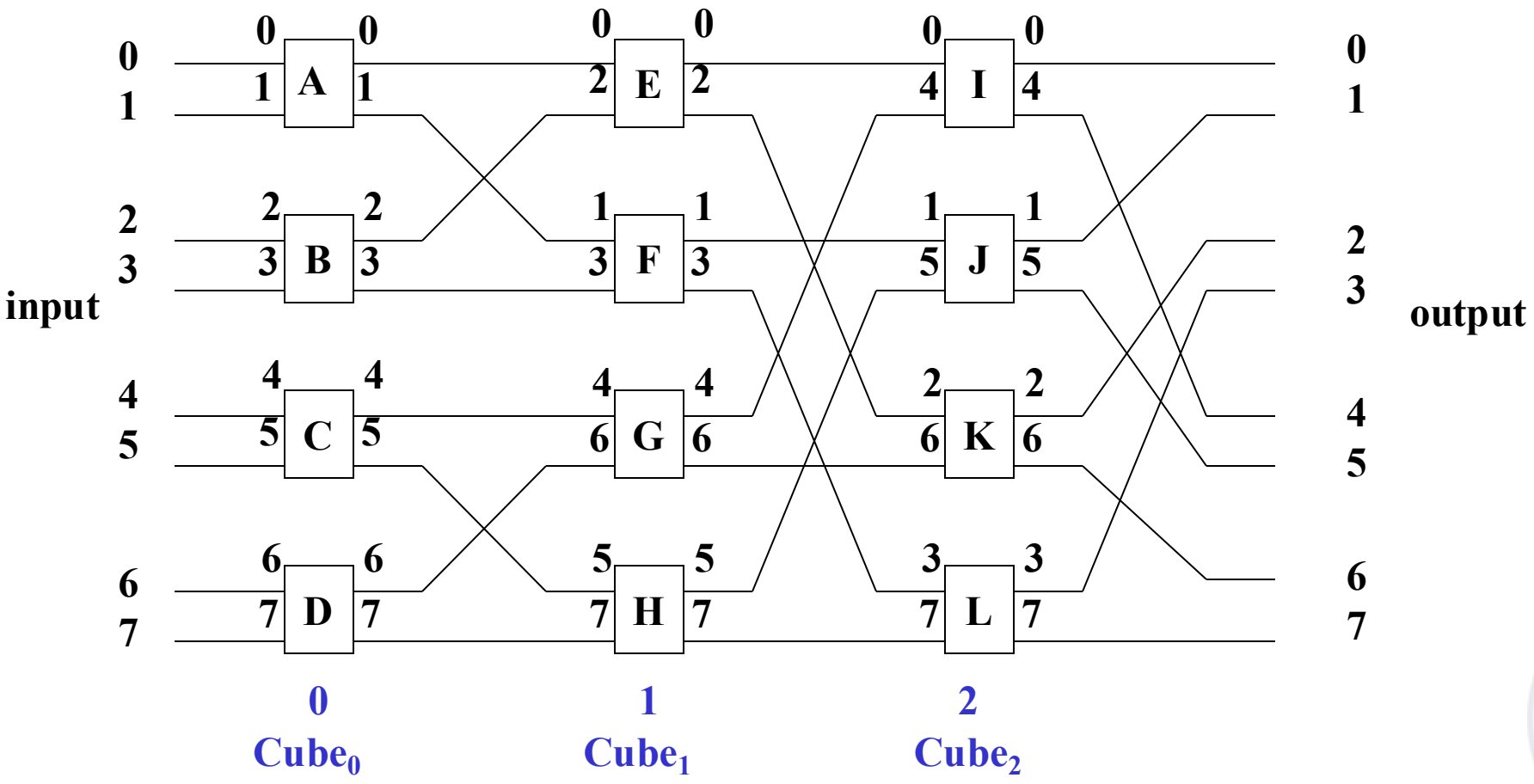
Characteristics of multi-stage cube interconnection network

- Switch unit: two-function switch unit
- Control mode: stage, part stage and unit control
- Topology: cube structure

Multi-stage cube interconnection network

- N-unit multi-stage cube network topology diagram
  - From $n = \log_2 N$, find the number n of the multi-stage cube network
  - The stage number is set from input to output as 0, 1, ..., n-1
  - Each stage draws N/2 two-function switch units
  - Let the two input/output terminals of all the i^{th} stage switch units be numbered according to the $Cube_i$ relationship
  - Connect the ends of the switches with the same number on each stage

Three-stage cube interconnection network



Multi-stage cube interconnection network

The multi-level cube network can be divided into:

- Switched network
- Mobile number network

} STARAN Network

- Indirect binary n-cube network

Multi-stage cube interconnection network

Flip Network

- The multi-stage cube network that adopts the stage control mode is called the switching network, which can only realize the function of the switching function

Exchange function

- Exchange a group of elements symmetrically. If a group of elements contains 2^1 elements, that is, all the k^{th} elements in the group are exchanged with the $[2^l-(k+1)]^{th}$ elements.

N = 8		Stage control signal ($K_2K_1K_0$), K_i: i^{th} stage (0=connect, 1=exchange)							
		000	001	010	011	100	101	110	111
No.	0	0	1	2	3	4	5	6	7
	1	1	0	3	2	5	4	7	6
	2	2	3	0	1	6	7	4	5
	3	3	2	1	0	7	6	5	4
	4	4	5	6	7	0	1	2	3
	5	5	4	7	6	1	0	3	2
	6	6	7	4	5	2	3	0	1
	7	7	6	5	4	3	2	1	0
Function		8 × 1	4×2	4×2 + 2× 4	2× 4	2× 4 + 1×8	4×2 + 1×4 + 1× 8	4× 2 + 1×8	1×8
		i	$Cube_0$	$Cube_1$	$Cube_0$ +Cube 1	$Cube_2$	$Cube_0$+ $Cube_2$	$Cube_1$ +$Cube_2$	$Cube_0$+ $Cube_1$+ $Cube_2$

Multi-stage cube interconnection network

Flip Network

- The multi-stage cube network that adopts the stage control mode is called the switching network, which can only realize the function of the switching function

Exchange function

- Exchange a group of elements symmetrically. If a group of elements contains 2^1 elements, that is, all the k^{th} elements in the group are exchanged with the $[2^l-(k+1)]^{th}$ elements.

Multi-stage cube interconnection network

The parallel processor has 16 processors. To achieve the function equivalent of first 4 groups of 4-element exchanges, then 2 groups of 8-element exchanges, and again a set of 16-element exchanges, the requirements are:

- Write the general formula of the interconnection function implemented between the processors
- Draw the corresponding multi-stage network topology diagram
- Mark the status of each stage of switch unit

Multi-stage cube interconnection network

- Input number sequence:

0 1 2 3 4 5 6 7 8 9 A B C D E F

Exchange ↓ 4 group * 4 elements

3 2 1 0 7 6 5 4 B A 9 8 F E D C

Exchange ↓ 2 group * 8 elements

4 5 6 7 0 1 2 3 C D E F 8 9 A B

Exchange ↓ 1 group * 16 elements

B A 9 8 F E D C 3 2 1 0 7 6 5 4

Multi-stage cube interconnection network

- Interconnection function:

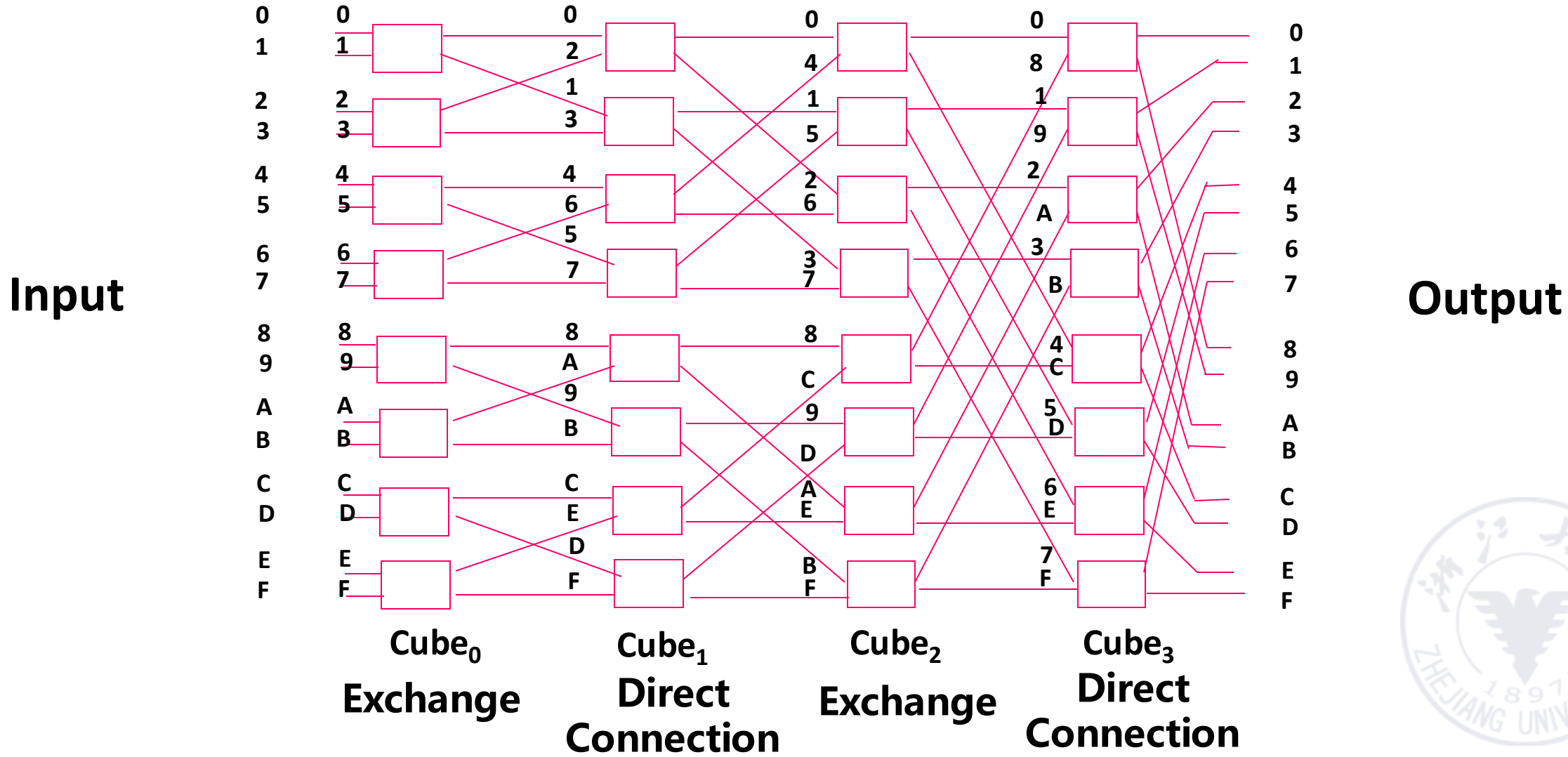
  (0 B) (1 A) (2 9) (3 8)

  (4 F) (5 E) (6 D) (7 C)

- General formula of the interconnection function :

$$f(P_3P_2P_1P_0) = \overline{P_3}P_2\overline{P_1}P_0$$

$$Cube(P_3P_2P_1P_0) = \overline{P_3}P_2\overline{P_1}P_0$$



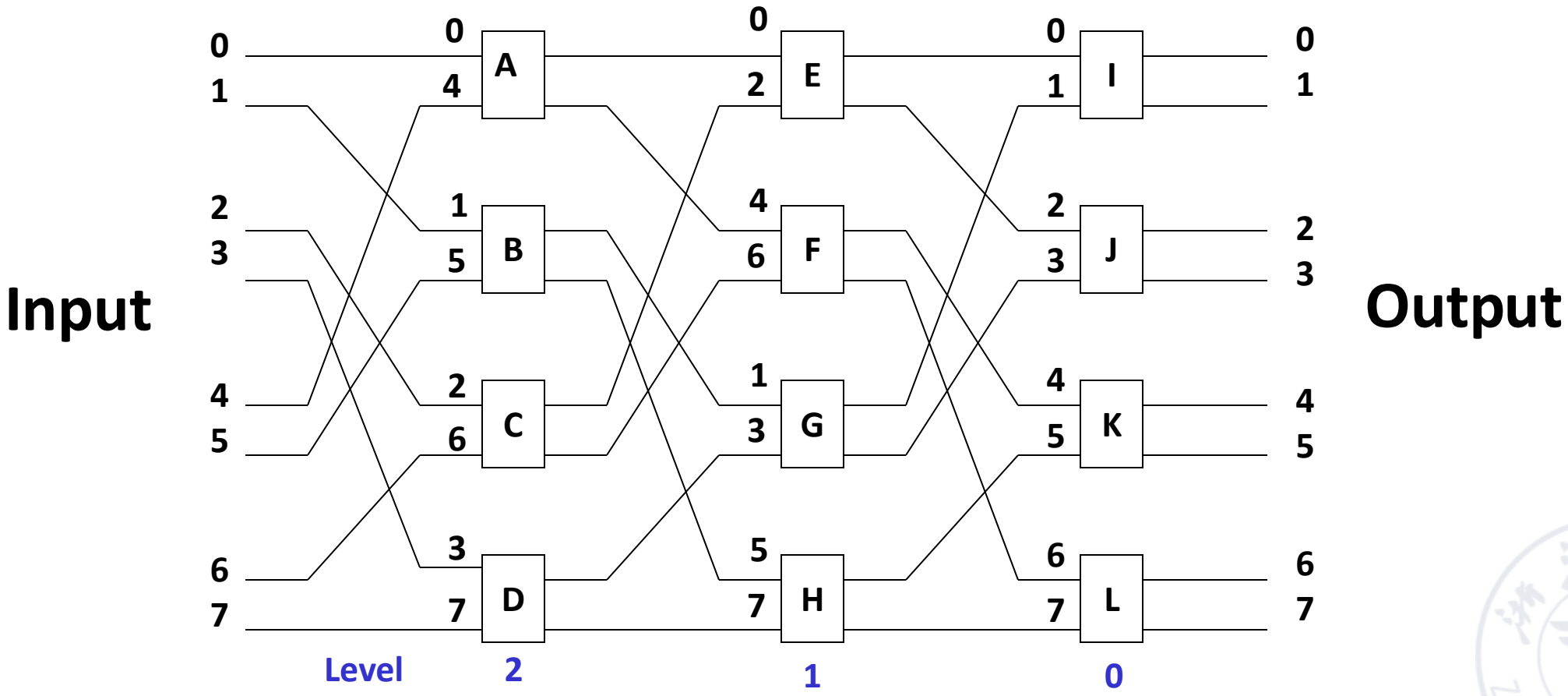
Multi-stage shuffle exchange network

- Multi-level shuffle exchange network is also called **Omega network**.

Features:

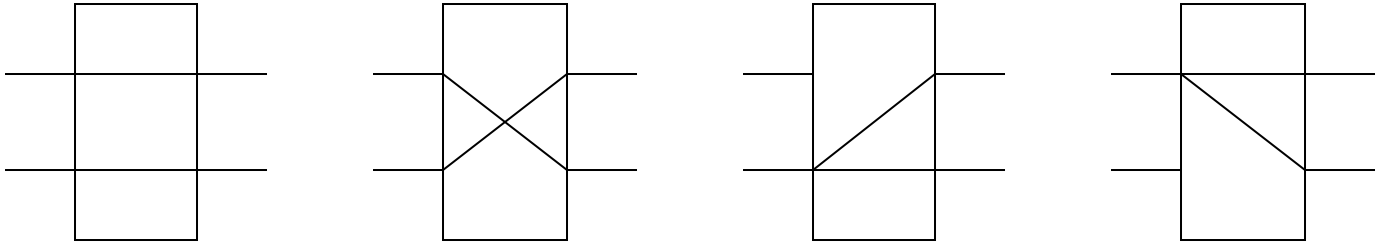
- The switch function has four functions
- The topological structure is shuffled topology followed by a four-function switch
- Control mode is unit control

N = 8 Omega Network

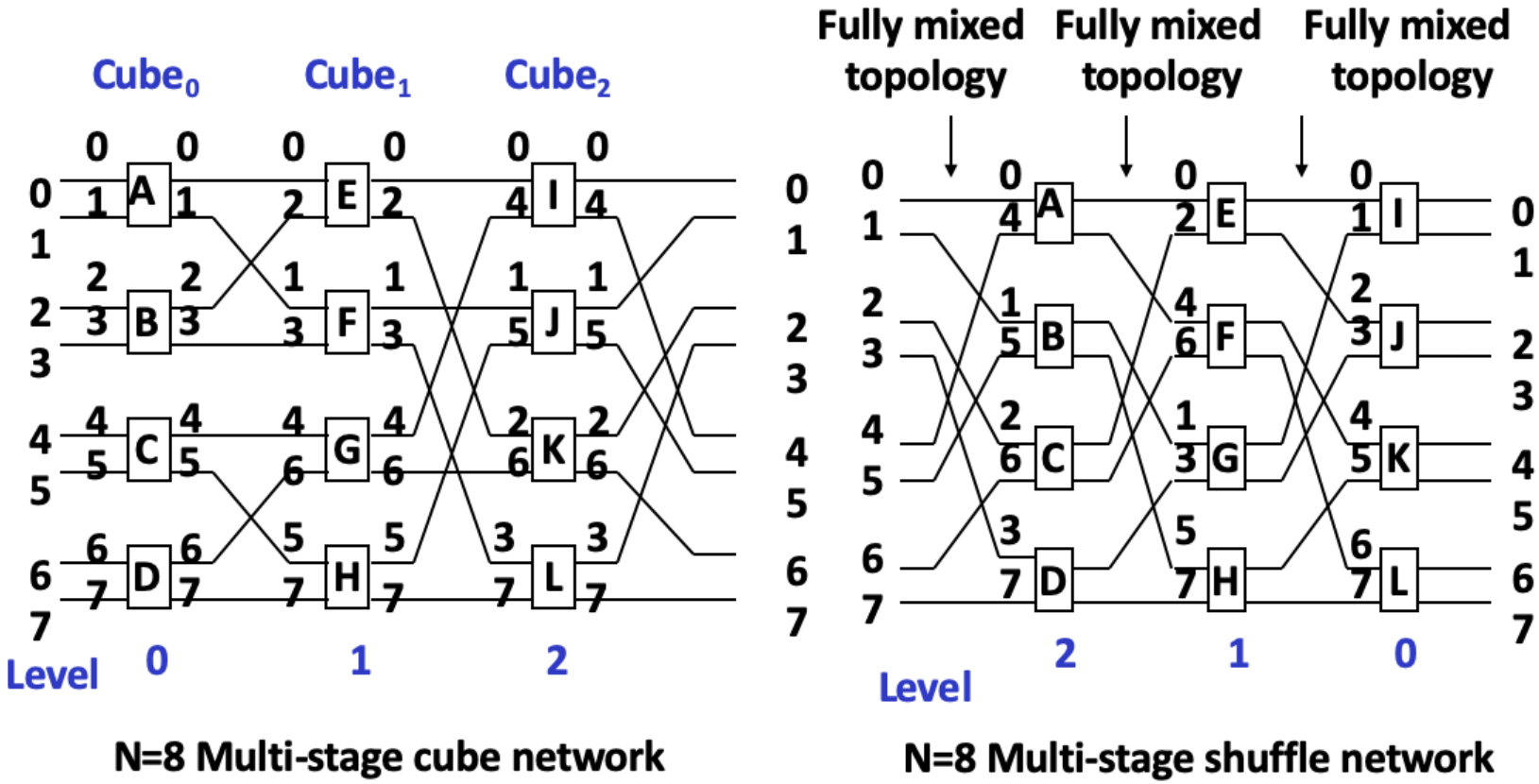


Multi-stage shuffle exchange network

1. Number of stages n = $\log_2 N$
2. The stage number from input to output is n-1,..., 1, 0
3. Number of units N/2
4. Shuffle topology + four-function switch



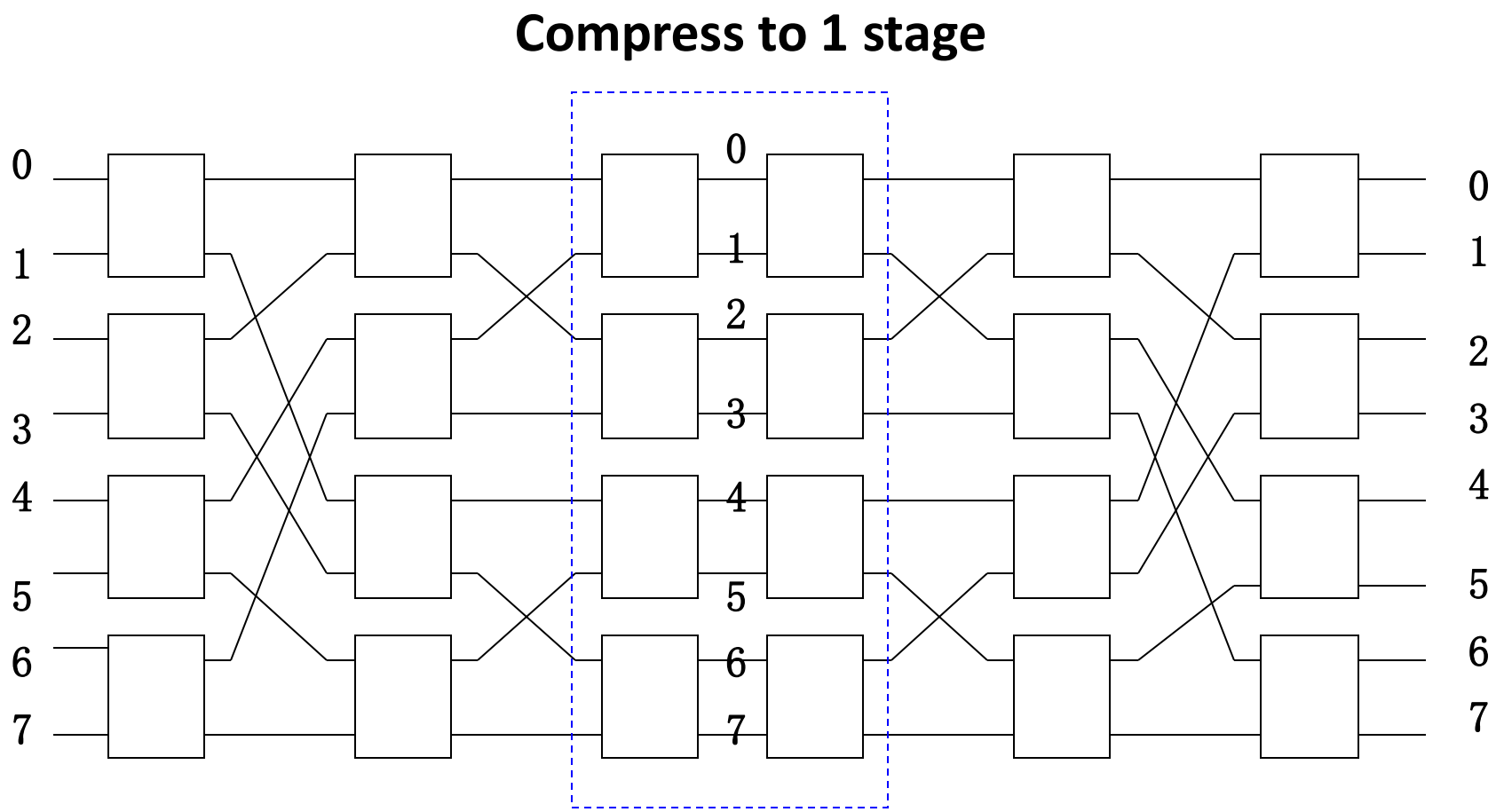
If the switching unit of the Omega network is restricted to only use the "direct connect" and ”exchange" functions, it becomes the inverse network of the n-cube network.



N=8 Multi-stage cube network

N=8 Multi-stage shuffle network

Example of multi-stage full array network (Benes network)



Omega network and n-cube network

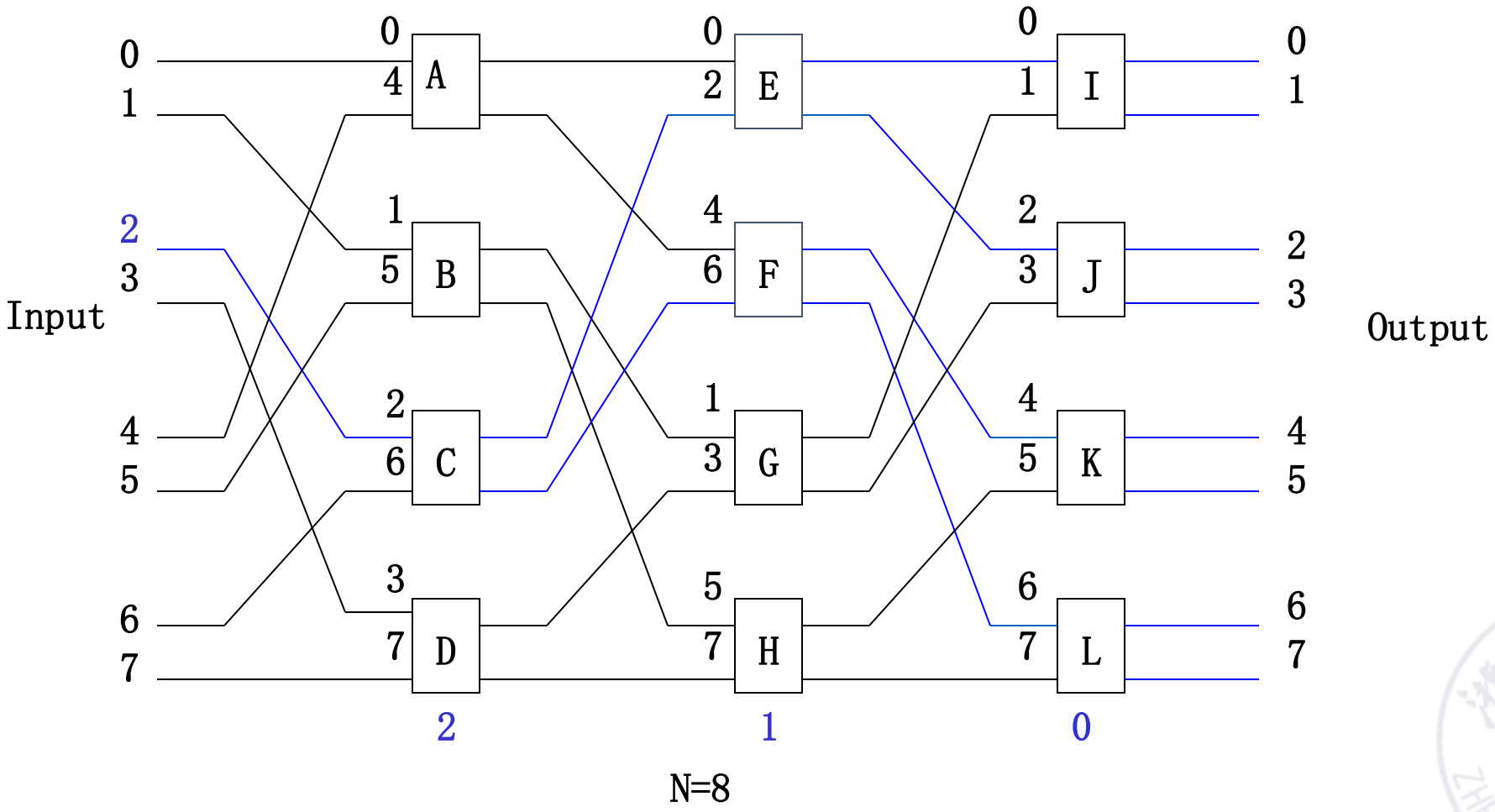
The difference between Omega network and n-cube network:

- 1. The level of Omega network data flow: n-1, n-2,..., 1, 0.
  The level of n-cube network data flow: 0, 1,..., n-1.

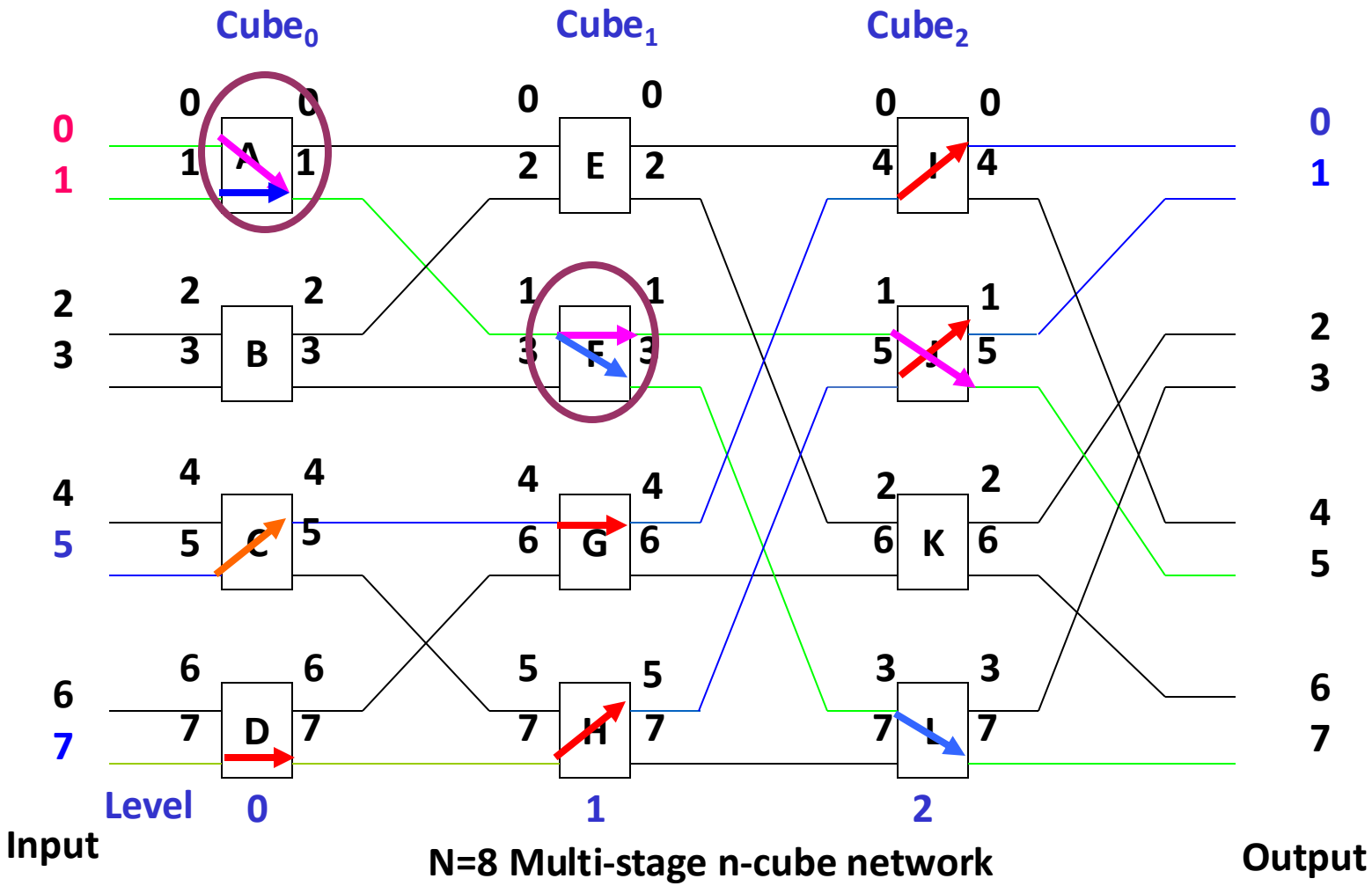
- 2. The Omega network uses a four-function exchange unit.
  The n-cube network uses a two-function exchange unit.

- 3. Omega network can realize one-to-many broadcasting function.
  N-cube network cannot achieve.

Omega network

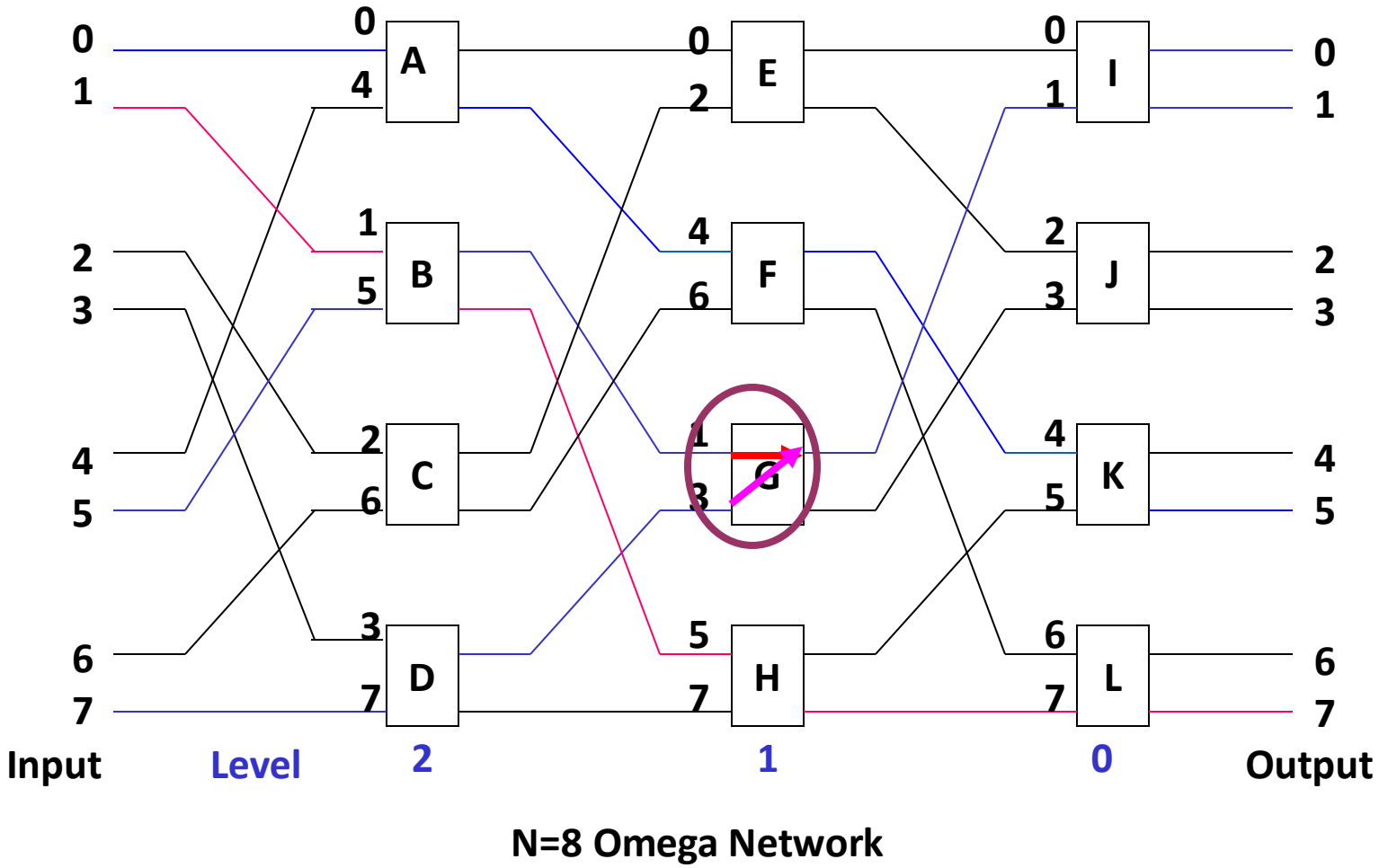


n-cube network



If the connection of **5 to 0** and **7 to 1** can be realized at the same time, can the connection of **0 to 5** and **1 to 7** be realized at the same time?

Omega network and n-cube network



N=8 Omega Network

The connection of **5 to 0** and **7 to 1** can be realized at the same time, but the connection of **0 to 5** and **1 to 7** cannot be realized at the same time!

Comparison of dynamic interconnection networks

	Bus System	Multi-stage Network	Crosspoint Switches
Bandwidth	O(w/n) to O(w)	O(w) to O(nw)	O(w) to O(nw)
Complexity of Link	O(w)	$O(nw\log_k n)$	$O(n^2w)$
Complexity of Switch	O(n)	$O(n\log_k n)$	$O(n^2)$
Connection and Path Finding	One to one once	Broadcast and exchange to some extents	Fully exchange, on
Explanation	n processors on bus, the bus bandwidth is w bits	n*n MIN with k*k switches, the bandwidth is w bits	n*n crosspoint switches with w-bit bandwidth

Advantages of SIMD

- SIMD architectures can exploit significant data-level parallelism for:
  - Matrix-oriented scientific computing
  - Media-oriented image and sound processors

- SIMD is more energy efficient than MIMD
  - Only needs to fetch one instruction per data operation
  - Makes SIMD attractive for personal mobile devices

- SIMD allows programmer to continue to think sequentially

Graphical Processing Units

- Basic idea:
  - *Heterogeneous* execution model
    - CPU is the *host*, GPU is the *device*
  - Develop a C-like programming language for GPU
  - Unify all forms of GPU parallelism as *CUDA thread*
  - Programming model is "Single Instruction Multiple Thread"

Programming the GPU: CUDA

- CUDA: Compute Unified Device Architecture
- NVIDIA decided that the unifying theme of all these forms of parallelism is the *CUDA Thread*. We call the hardware that executes a whole block of threads a *multithreaded SIMD Processor*.
- Actually, **GPUs are really just multithreaded SIMD Processors**.

Example: DAXPY

- C code:

```
// Invoke DAXPY
daxpy(n, 2.0, x, y);
// DAXPY in C
void daxpy(int n, double a, double *x, double *y)
{
          for (int i = 0; i < n; ++i)
               y[i] = a*x[i] + y[i];
}
```

- CUDA code:

```
// Invoke DAXPY with 256 threads per Thread Block
__host__
int nblocks = (n+ 255) / 256;
   daxpy<<<nblocks, 256>>>(n, 2.0, x, y);
// DAXPY in CUDA
__global__
void daxpy(int n, double a, double *x, double *y)
{
   int i = blockIdx.x*blockDim.x + threadIdx.x;
   if (i < n) y[i] = a*x[i] + y[i];
}
```

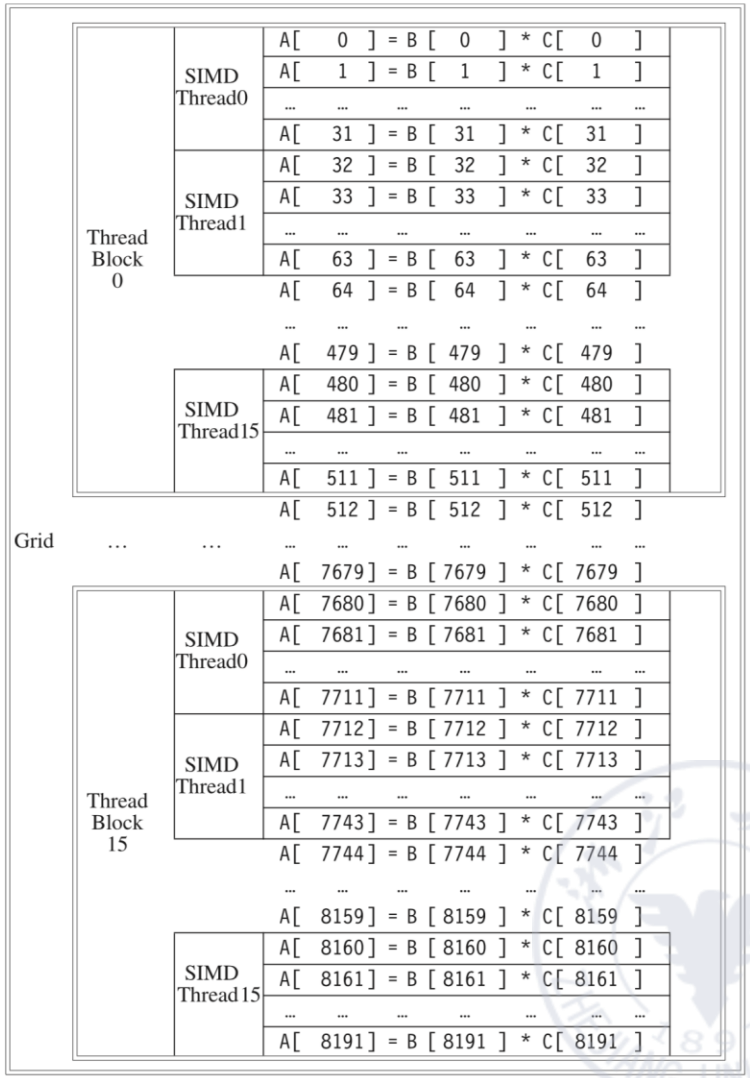
CPU (Host)

GPU (Device)

Grid, Thread Blocks and Threads

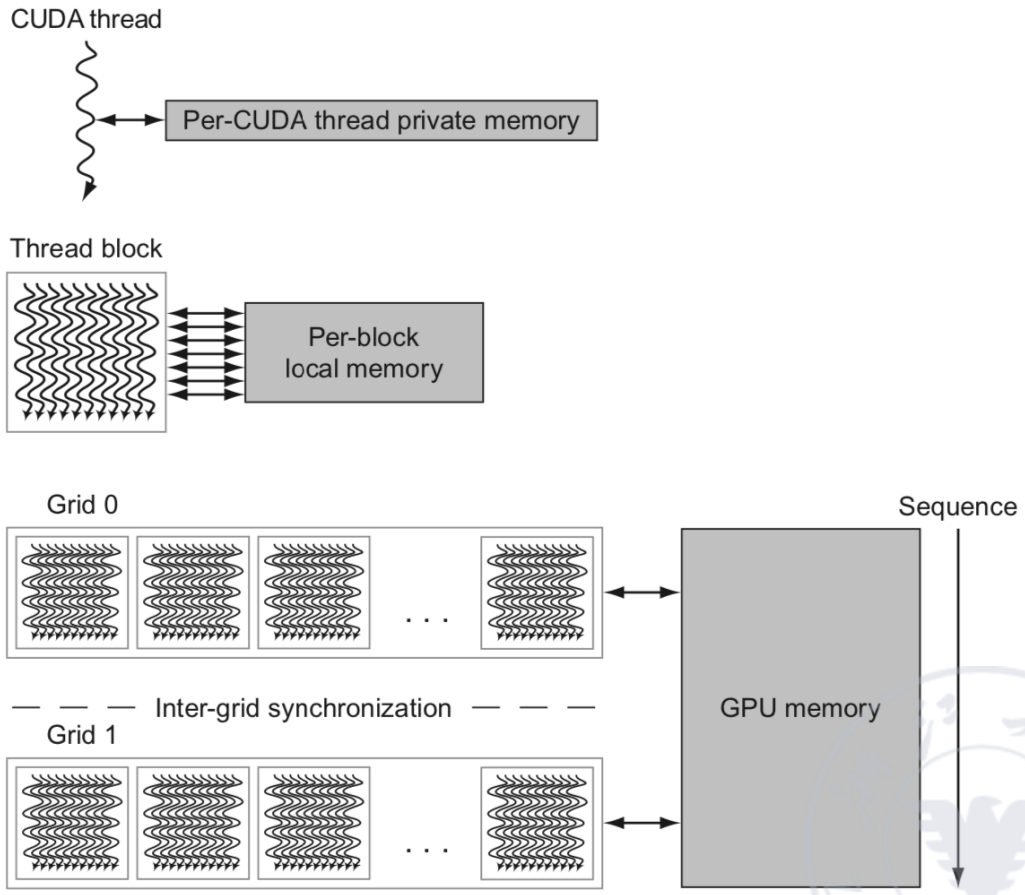
- A *thread* is associated with each data element
- Threads are organized into *blocks*
- Blocks are organized into a *grid*

- GPU hardware handles thread management, not applications or OS

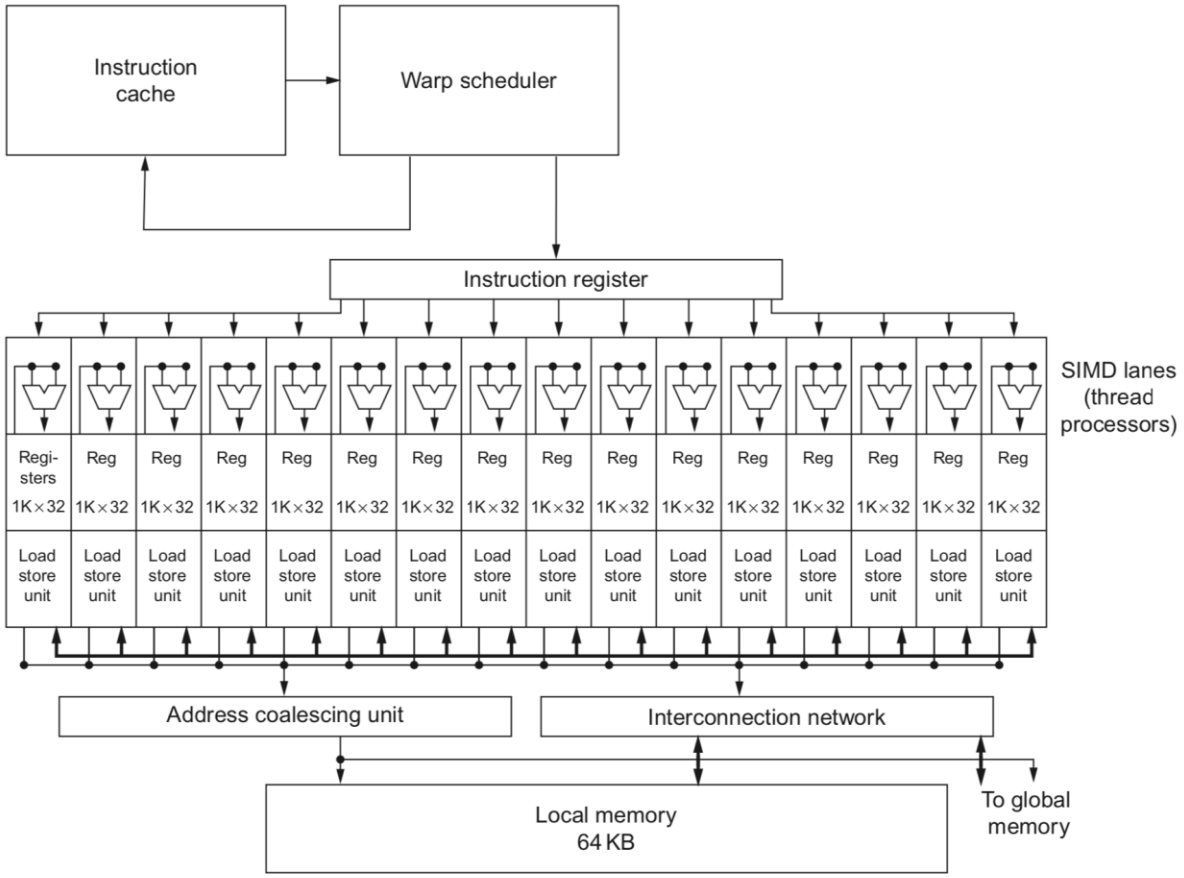


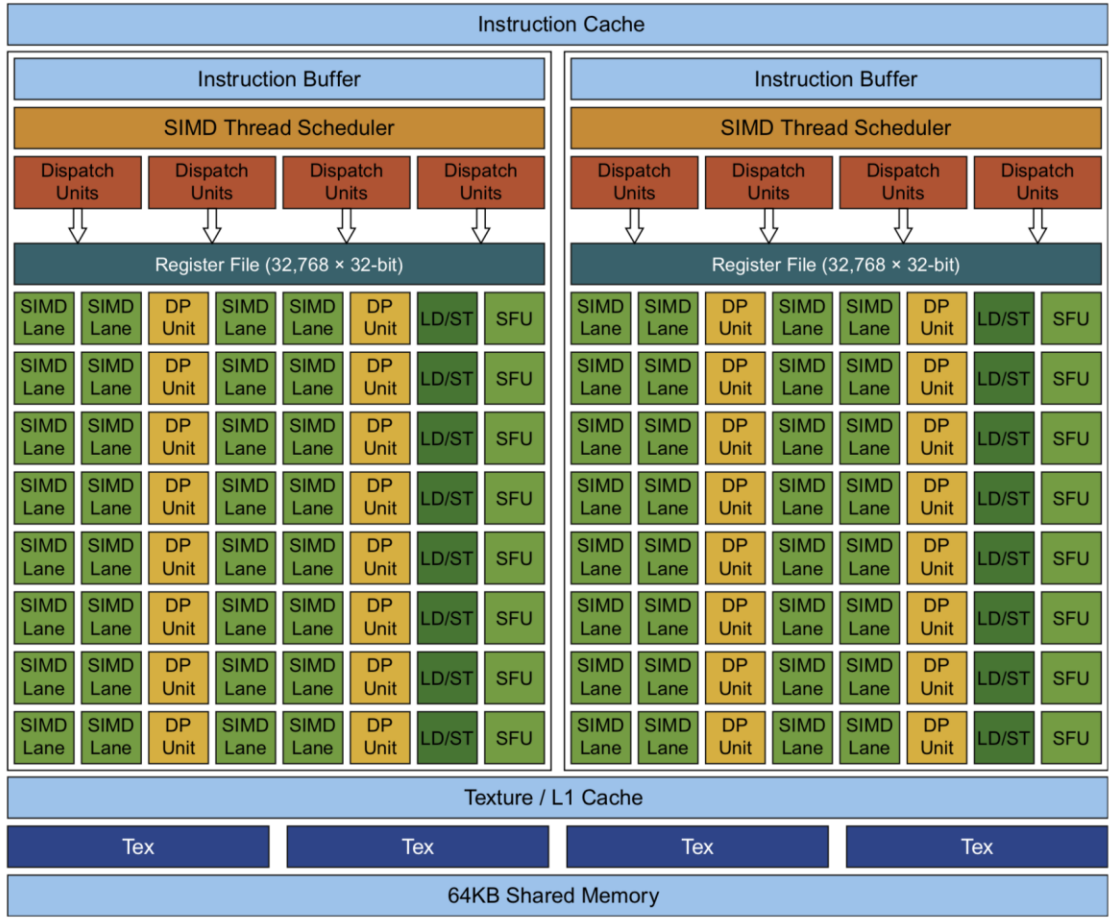
GPU memory structures

- **GPU memory** is shared by all *Grids* (vectorized loops).
- **Local memory** is shared by all threads of SIMD instructions within *a Thread Block* (body of a vectorized loop).
- **Private memory** is private to a single *CUDA Thread*.

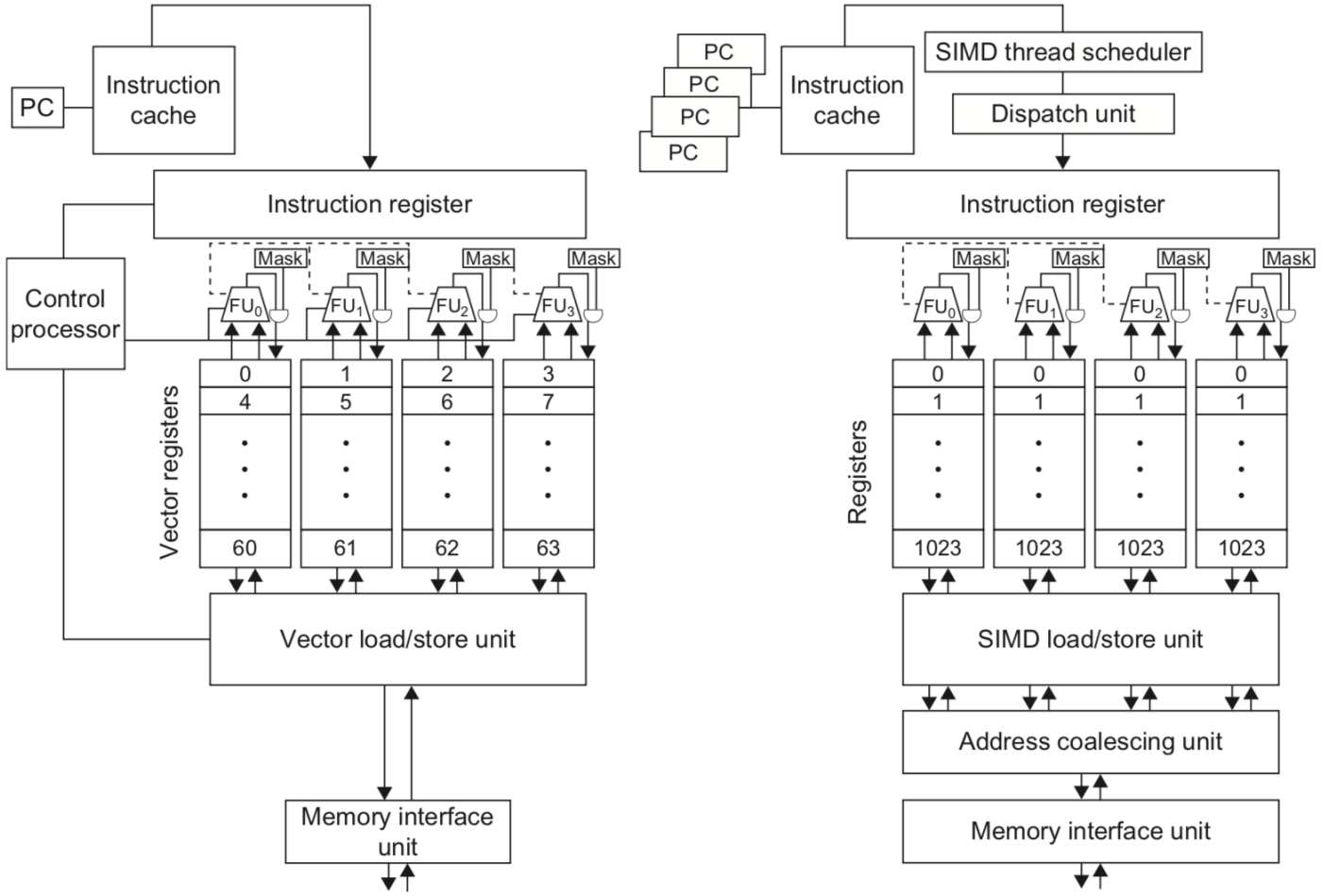


GPU Organization and an example (Pascal GPU)





NVIDIA GPU (right) and vector machine (left)



NVIDIA GPU and vector machines

- Similarities to vector machines:
  - Works well with data-level parallel problems
  - Scatter-gather transfers
  - Mask registers
  - Large register files

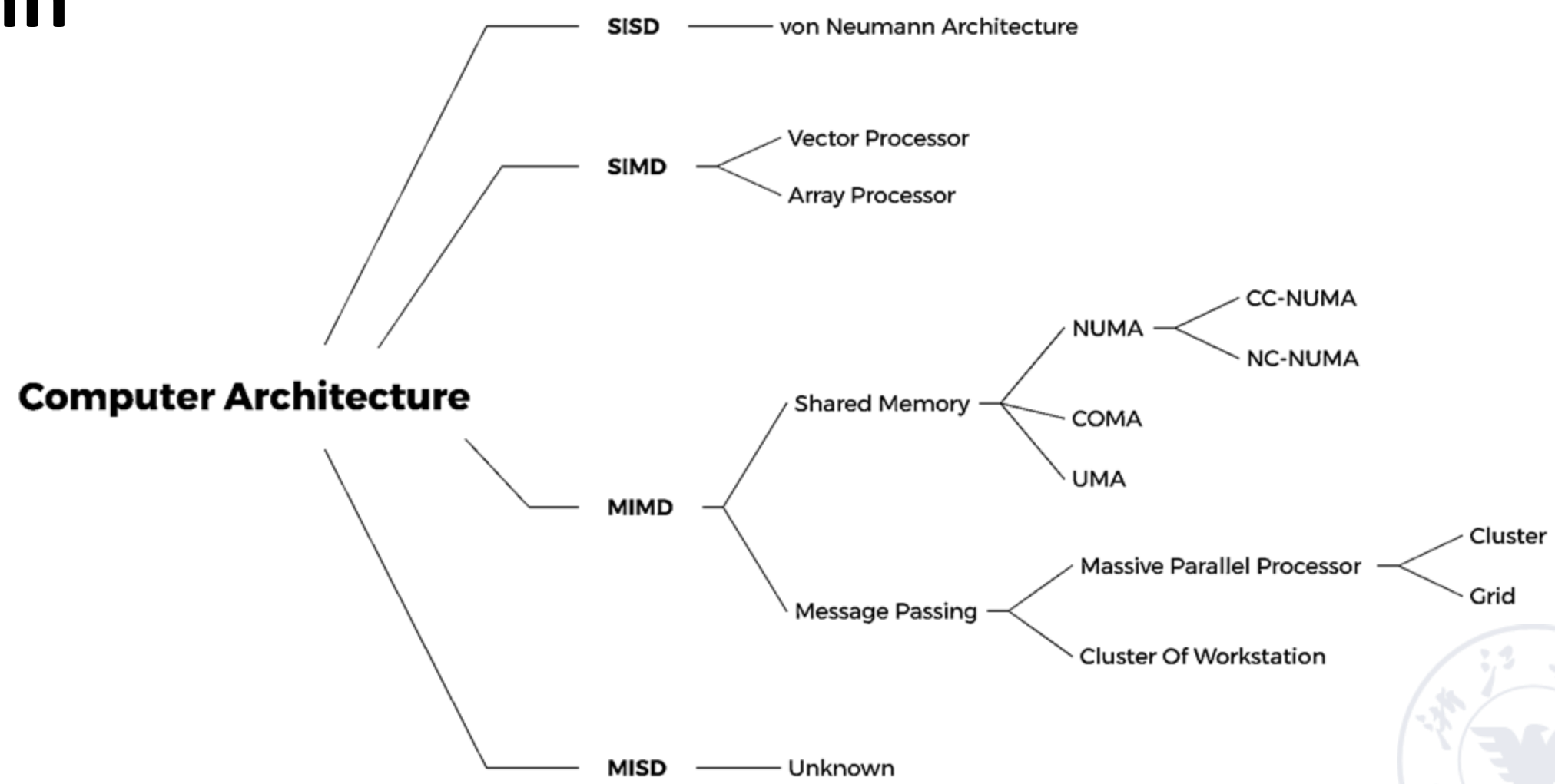
- Differences:
  - No scalar processor
  - Uses multithreading to hide memory latency
  - Has many functional units, as opposed to a few deeply pipelined units like a vector processor

Loop-Level Parallelism

- Loops in programs are the fountainhead of many of the types of parallelism

- Finding and manipulating loop-level parallelism is critical to exploiting both DLP and TLP, as well as the more aggressive static ILP approaches (e.g., VLIW) .

Flynn

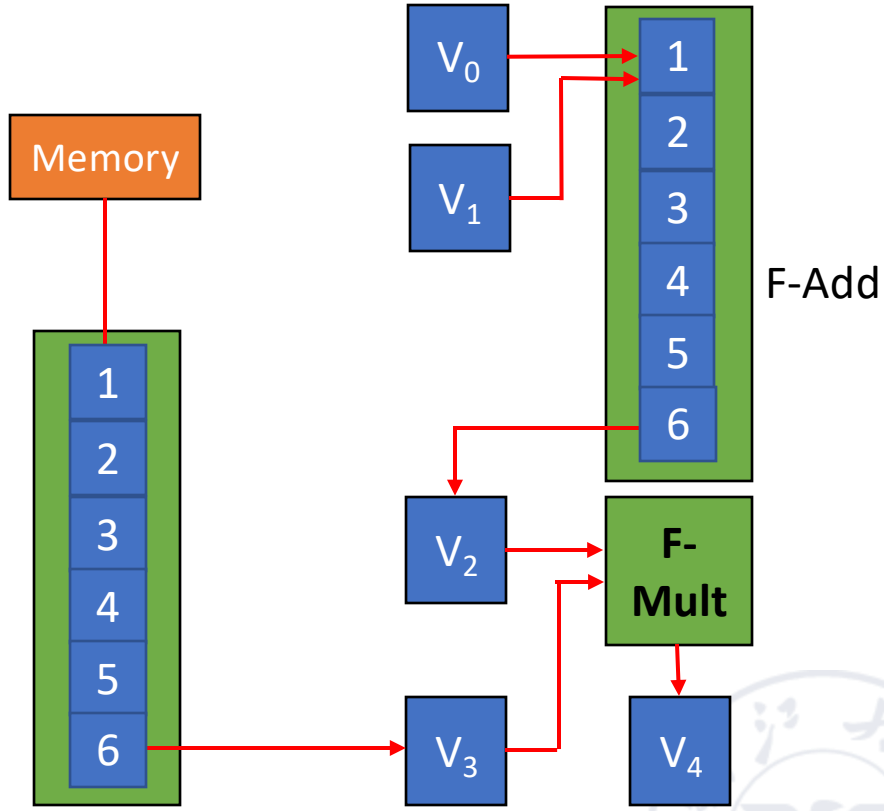

Computer Architecture
SISD
von Neumann Architecture
SIMD
Vector Processor
Array Processor
MIMD
Shared Memory
NUMA
CC-NUMA
NC-NUMA
COMA
UMA
Message Passing
Massive Parallel Processor
Cluster
Grid
Cluster Of Workstation
MISD
Unknown

Vector Processor & Vector Chaining Technology

- (1). $V3 \leftarrow A$
- (2). $V2 \leftarrow V0 + V1$
- (3). $V4 \leftarrow V2 * V3$

- Use vector chaining technology.
- The execution time is:

max{(1+6+1), (1+6+1)} + (1+7+1)+N-1 = N + 16



Loop-Level Parallelism

- Focuses on determining whether data accesses in later iterations are dependent on data values produced in earlier iterations
  - Loop-carried dependence

- Example 1:

```
for (i=999; i>=0; i=i-1)
        x[i] = x[i] + s;
```

- No loop-carried dependence

Loop-Level Parallelism

- Example 2:

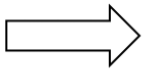
```
for (i=0; i<100; i=i+1) {
        A[i+1] = A[i] + C[i];        /* S1 */
        B[i+1] = B[i] + A[i+1];      /* S2 */
}
```

- S1 and S2 use values computed by S1 in previous iteration
- S2 uses value computed by S1 in same iteration

Loop-Level Parallelism

- Example 3:

```
for (i=0; i<100; i=i+1) {
        A[i] = A[i] + B[i];     /* S1 */
        B[i+1] = C[i] + D[i]; /* S2 */
}
```



- S1 uses value computed by S2 in previous iteration but dependence is not circular so loop is parallel.

Loop-Level Parallelism

- Practice:

```
for (i=0; i<100; i=i+1) {
        Y[i] = X[i]/c;          /* S1 */
        X[i] = X[i] +c;         /* S2 */
        Z[i] = Y[i] +c;         /* S3 */
        Y[i] = c- Y[i];         /* S4 */
}
```

```
for (i=0; i<100; i=i+1) {
        T[i] = X[i]/c;          /* S1 */
        P[i] = X[i] +c;         /* S2 */
        Z[i] = T[i] +c;         /* S3 */
        Y[i] = c-T[i];          /* S4 */
}
```

Vector Chaining Technology

- Practice:

Assuming there is a Vector machine, having 2 load/store units with 10 clock cycles function unit time, 1 multiplier with 7 clock cycles function unit time, and 1 adder with 4 clock cycles function unit time. All the vector function units are fully pipelined that can start a new operation on every clock cycle. There is a code sequence as following, the vector length is 64.

```
•   Vld     v0, x5          ; load vector X
•   Vmul    v1, v0, f0      ; vector –scalar multiply
•   Vld     v2, x6          ; Load vector Y
•   Vadd    v3, v1, v2      ; vector –vector add
•   Vst     v3, x6          ; store the sum
```

- How many clock cycles to execute the code on a vector machine without chaining technique?
- If the above code running on a vector machine with chaining technique, how many conveys will this vector sequence take? Please showing the layout of the conveys.
- If an element of a result vector going from a function unit to a chaining register need 1 clock cycle, and vice versa, how many clock cycles to execute the code using chaining technique?