

Interconnection networks

Intensive Computation

Annalisa Massini

2017/2018

References

- *Computer Architecture: A Quantitative Approach*
5th Edition, Appendix F *Interconnection Networks Ch. F.4*
Hennessy Patterson
Slides: Timothy Mark Pinkston and José Duato
- *Advanced Computer Architecture and Parallel Processing*
H. El-Rewini, M. Abd-El-Barr, John Wiley and Sons, 2005
- *Parallel computing for real-time signal processing and control*
– Ch. 2 *Parallel Architectures*
M. O. Tokhi, M. A. Hossain, M. H. Shaheed, Springer, 2003

Criteria for classification

- **Multiprocessors interconnection networks** (INs) can be classified based on a number of criteria:
 - Mode of Operation (Synchronous vs. Asynchronous)
 - Control Strategy (Centralized vs. Decentralized)
 - Switching Techniques (Packet switching vs. Circuit switching)
 - Topology (Static Vs. Dynamic)

Mode of operation

- According to the mode of operation, INs are classified as **synchronous** versus **asynchronous**
- In **synchronous** mode of operation:
 - a **single global clock** is used by all components in the system such that the whole system is operating in a lock-step manner
- **Asynchronous** mode of operation:
 - Does not require a global clock
 - **Handshaking** signals are used instead in order to coordinate the operation of asynchronous systems
- While **synchronous systems** tend to be **slower** compared to asynchronous systems, they are race and **hazard-free**

Control strategy

- According to the control strategy, INs can be classified as *centralized* versus *decentralized*
- In **centralized** control systems:
 - a single central control unit is used to oversee and control the operation of the components of the system
- In **decentralized** control:
 - the control function is distributed among different components in the system
- The function and reliability of the central control unit can become the bottleneck in a centralized control system
- For example, while the *crossbar* is a centralized system, the *multistage interconnection networks* are decentralized

Switching techniques

- Interconnection networks can be classified according to the switching mechanism as **circuit switching** versus **packet switching networks**
- In the **circuit switching mechanism**:
 - A complete path has to be established prior to the start of communication between a source and a destination
 - The established path will remain in existence during the whole communication period

Switching techniques

- Interconnection networks can be classified according to the switching mechanism as *circuit switching* versus *packet switching networks*
- In a *packet switching mechanism*:
 - Communication between a source and destination takes place via messages that are divided into smaller entities, called packets
 - On their way to the destination, packets can be sent from a node to another in a store-and-forward manner until they reach their destination

Topology

- An **interconnection network topology** is a *mapping function* from the *set of processors and memories* onto the same set of processors and memories
- In other words, the topology describes how to connect processors and memories to other processors and memories
- For example:
 - A **fully connected topology** is a mapping in which each processor is connected to all other processors in the computer
 - A **ring** topology is a mapping that connects processor k to its neighbors, processors $(k - 1)$ and $(k + 1)$

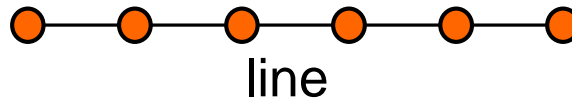
Topology

- In general, interconnection networks can be classified as *static* versus *dynamic* networks
- In *static networks*:
 - direct fixed links are established among nodes to form a fixed network
- In *dynamic networks*:
 - connections are established as needed
- Switching elements are used to establish connections among inputs and outputs
- Depending on the switch settings, different interconnections can be established

Static Networks

Linear Network

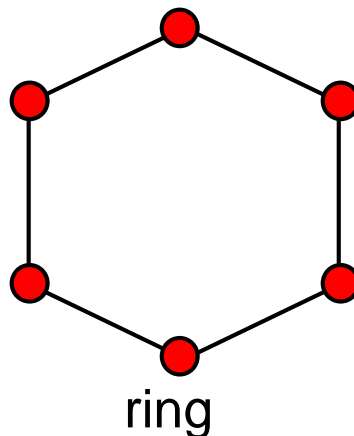
- Every node, except the nodes at the two ends, in this configuration is directly connected to two other nodes
- To connect n nodes in this configuration $n-1$ buses are required and the maximum internodes distance is $n-1$



Static Networks

Ring Interconnection Network

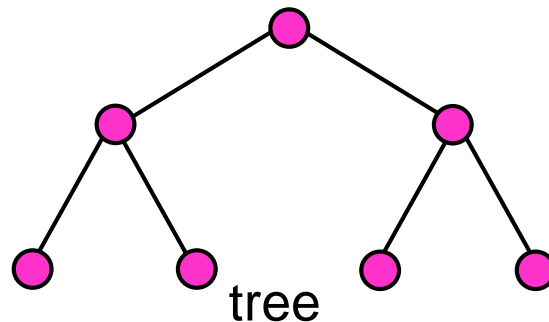
- n buses are required to connect n nodes
- *the maximum* internodes distance is $n / 2$
- Several commercial machines have been designed using ring networks (e.g. Hewlett-Packard's Exemplar V2600 and Kendal Square Research's KSR-2)



Static Networks

Tree Interconnection Network

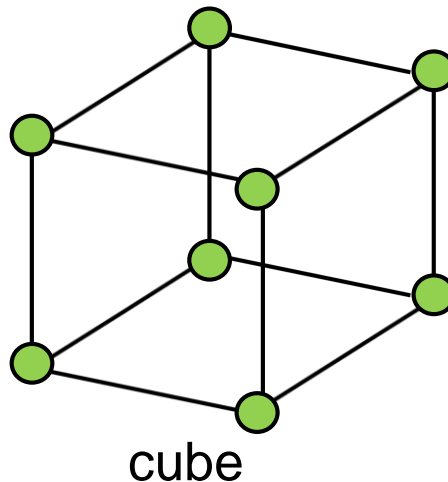
- In the tree structure (*n -level tree*) any intermediate node acts as a medium to establish communication between its parents and children
- Communication can be established between any two nodes in the structure
- The root node can be the bottleneck



Static Networks

Hypercube Interconnection Network

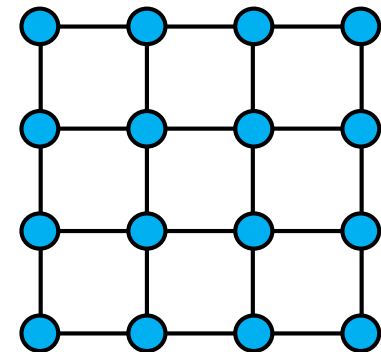
- An n -dimensional hypercube can connect 2^n nodes
- The **nodes** are labelled using **binary addresses**
- Addresses of the two neighboring nodes differ by one bit
- Many commercial multiprocessors (especially NUMA multiprocessors) have used hypercube interconnections



Static Networks

Mesh and Torus Interconnection Network

- **Mesh** is used to connect large numbers of nodes
- It is an alternative to hypercube in large multiprocessors
- To formulate a mesh structure with n nodes, $2(n - \sqrt{n})$ buses are required
- The maximum internodes distance is $2(\sqrt{n} - 1)$
- A **torus** is obtained by using wraparound connections between the nodes at opposite edges

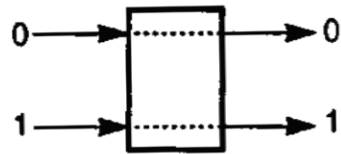


array or mesh

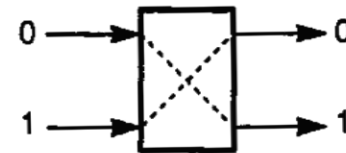
Dynamic Networks

- Connections in a dynamic network are established on the fly as needed
- Dynamic networks can be classified based on interconnection scheme as **bus-based** or **switch-based**
- **Bus-based** networks can further be classified as single bus or multiple buses
- **Switch-based** can be classified according to the structure of the interconnection network:
 - single-stage
 - multistage
 - crossbar networks

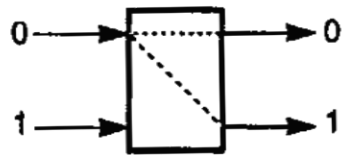
2 × 2 Switches



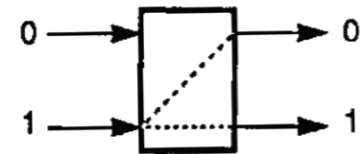
(a) Straight



(b) Crossover



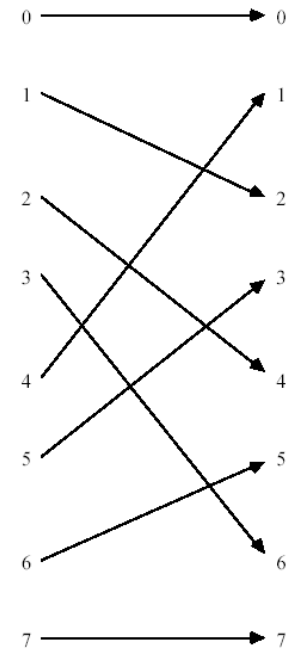
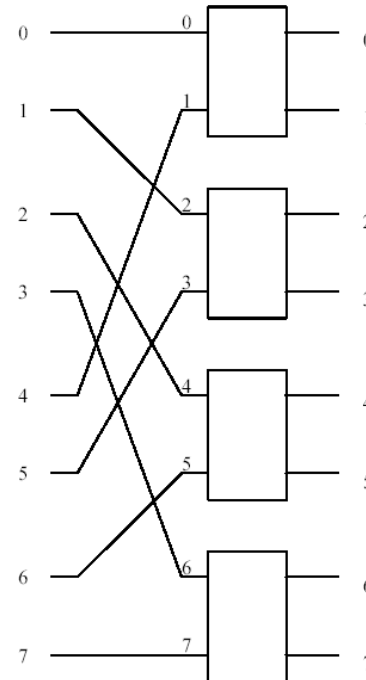
(c) Upper broadcast



(d) Lower broadcast

Single-stage networks

- Single stage **Shuffle-Exchange** IN (left)
- Perfect *shuffle* mapping function (right)
- *Perfect shuffle operation*: cyclic shift 1 place left, e.g. 101 --> 011
- *Exchange operation*: invert least significant bit, e.g. 101 --> 100



Multistage Interconnection Networks

- The capability of single stage networks is limited
- If we **cascade** enough of them together, they form a **Multistage Interconnection Network** (MIN)
- Switches can perform their own routing or can be controlled by a central router

Multistage Interconnection Networks

- **Nonblocking**

- A network is (strictly) **nonblocking** if it can connect any idle input to any idle output regardless of what other connections are currently in process

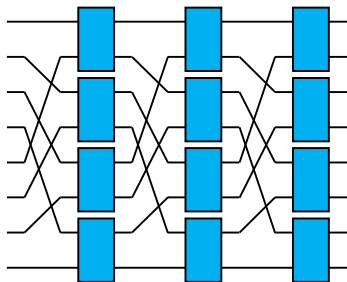
- **Rearrangeable nonblocking**

- Network able to establish all possible connections between inputs and outputs by rearranging its existing connections

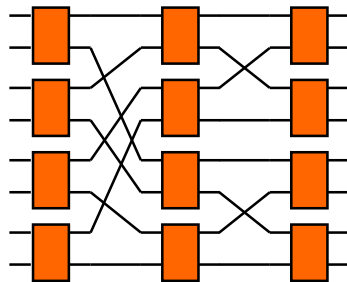
- **Blocking**

- A network is **blocking** if it can perform many, but not all, possible connections between terminals

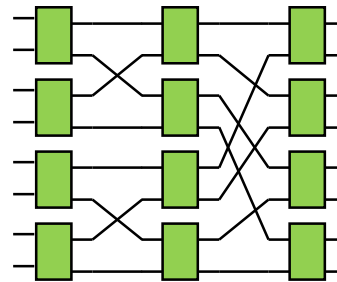
- Example: log N stage networks such as Omega, Baseline, Butterfly, ...



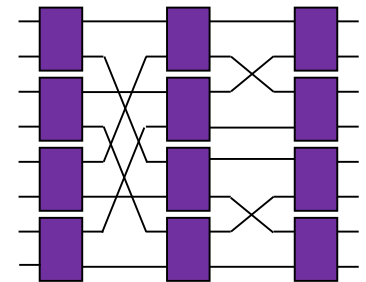
Omega



Baseline



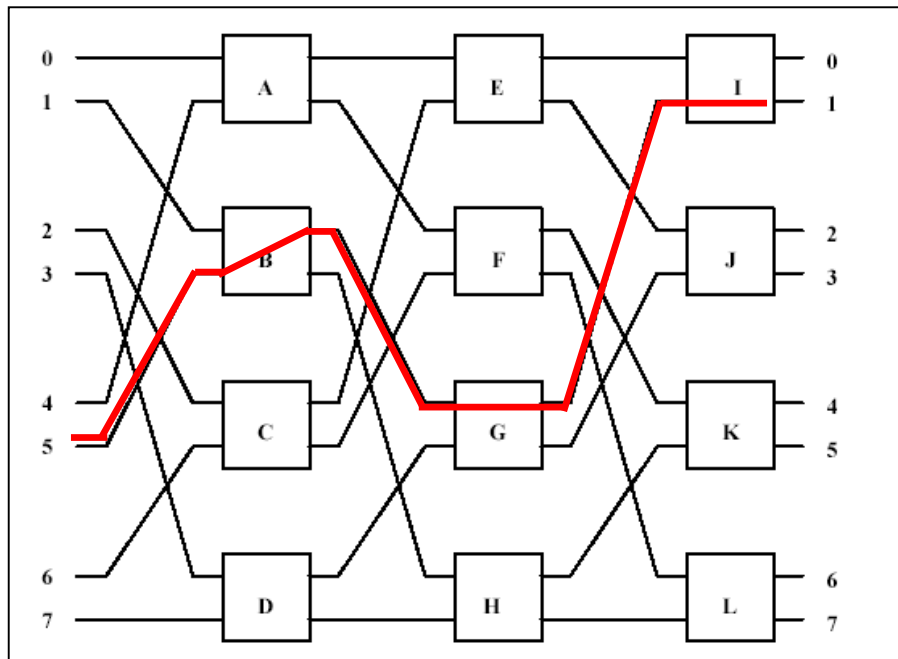
Reverse Baseline



Butterfly

Omega networks

- A MIN using 2×2 switches and a perfect shuffle interconnect pattern between the stages
- There is one unique path from each input to each output
- No redundant paths \rightarrow no fault tolerance, blocking

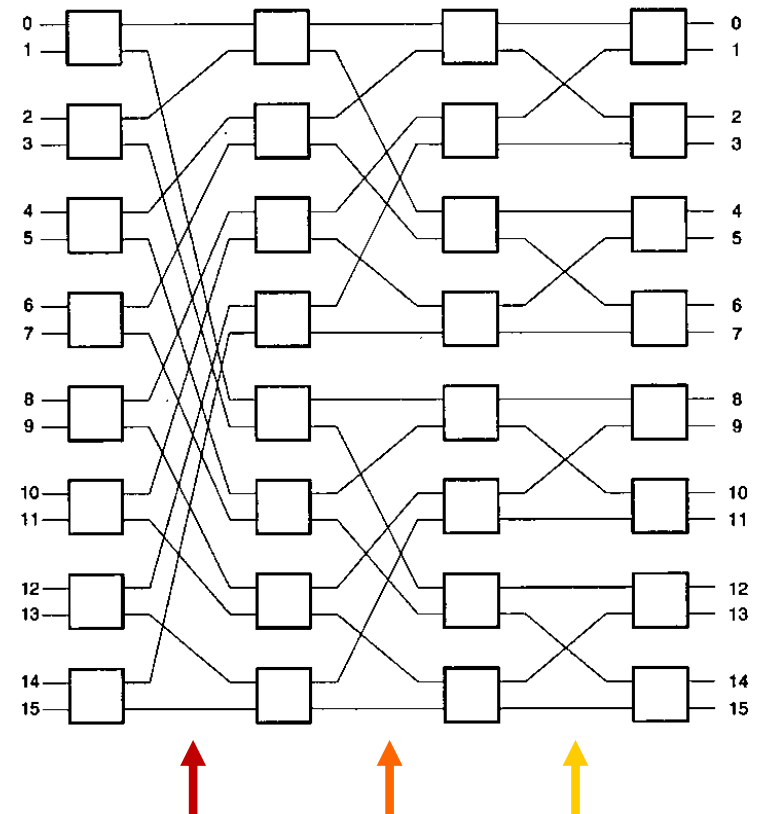
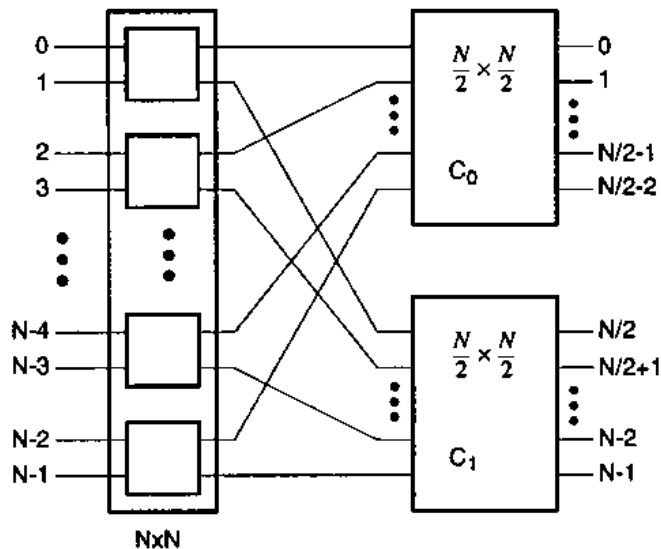


Example

- Connect input 101 to output 001
- **Self routing:**
 - Use the bits of the destination address for dynamically selecting a path
- Routing:
 - 0 means use upper output
 - 1 means use lower output

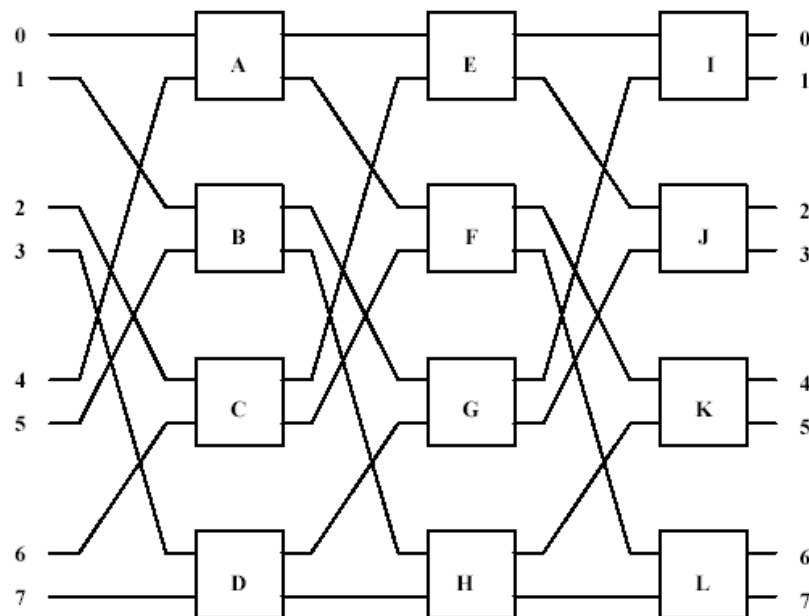
Baseline networks

- The baseline network can be generated recursively
- The **first stage** $N \times N$, the **second** $(N/2) \times (N/2)$ twice, the **third**...



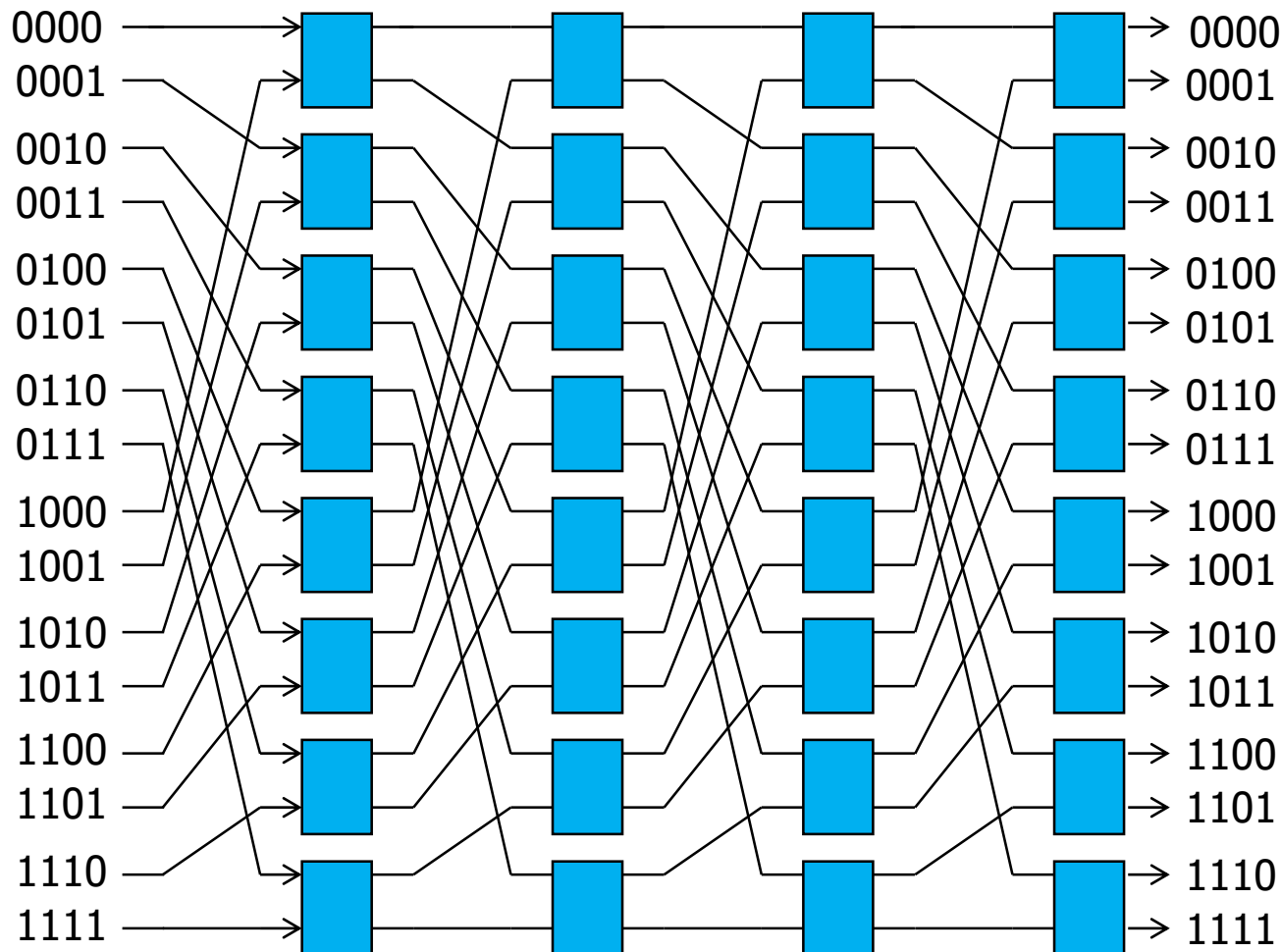
Omega networks

- $\log_2 N$ **stages** of 2×2 **switches**
- $N/2$ **switches per stage**
- $S = (N/2) \log_2(N)$ total number of switches
- Number of **permutations** in an Omega network 2^S



Network Topology

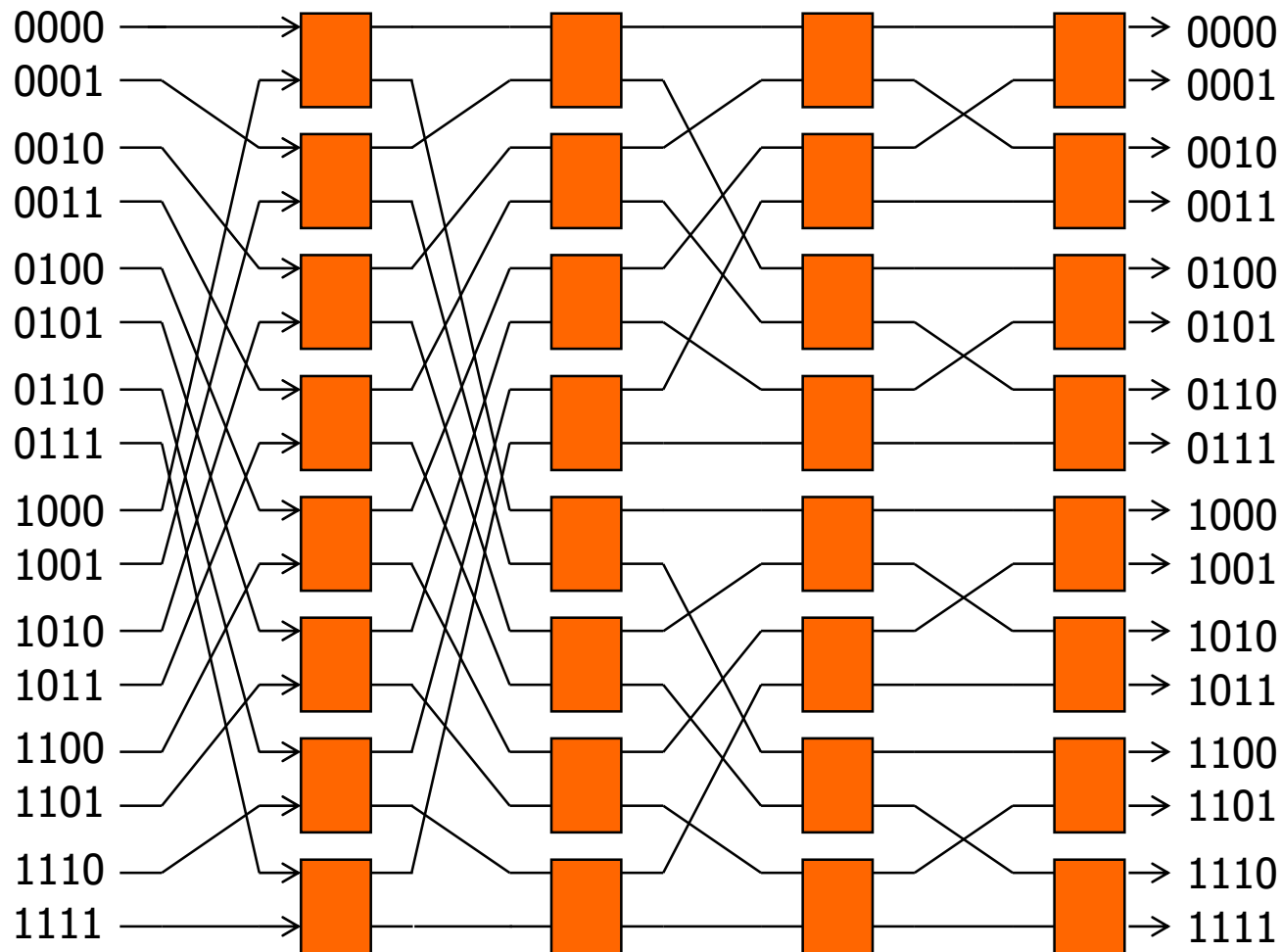
- Multistage interconnection networks (MINs)*



*4 stage
Omega
network*

Network Topology

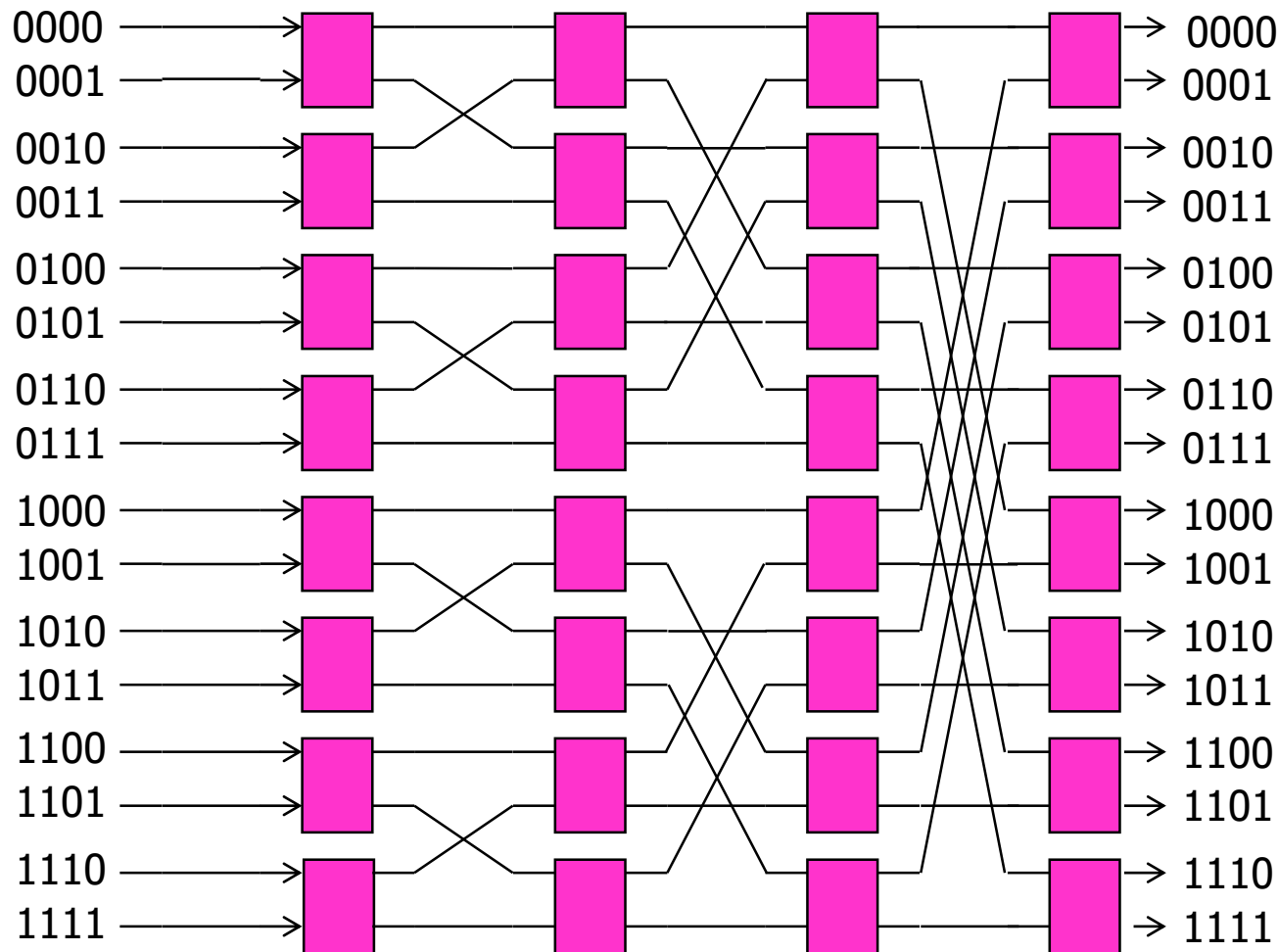
- Multistage interconnection networks (MINs)*



*4 stage
Baseline
network*

Network Topology

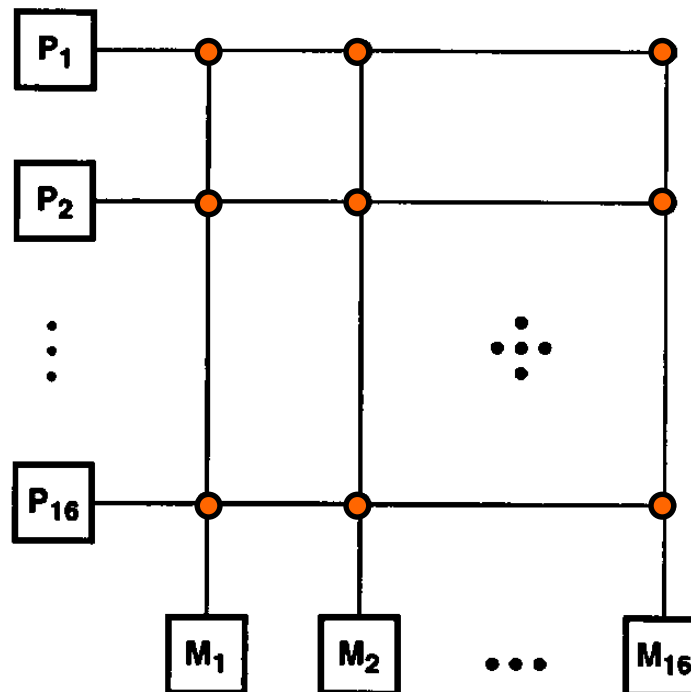
- Multistage interconnection networks (MINs)*



*4 stage
Reverse
Butterfly
network*

Crossbar Network

- Each junction is a **switching component** – connecting the row to the column
- Can only have one connection in each column



Crossbar Network

- The **major advantage** of the crossbar switch is its **speed**
- In one clock, a connection can be made between source and destination
- Because of its **complexity** (*number of switching components*), the **cost** of the crossbar switch can become the dominant factor for a large multiprocessor system
- Crossbars can be used to implement the $a \times b$ switches used in MIN's, so that each crossbar is small, and costs are kept down
- *Blocking* only if the destination is in use

COMPARISON OF NETWORK TOPOLOGIES

Comparison of Interconnection Networks

- Intuitively, one network topology is more desirable than another if it is:
 - More **efficient**
 - More **convenient**
 - More **regular** (i.e. easy to implement)
 - More **expandable** (i.e. highly modular)
 - Unlikely to experience bottlenecks
- Clearly ***no one interconnection network maximizes all these criteria***
- Some tradeoffs are needed

Comparison of Interconnection Networks

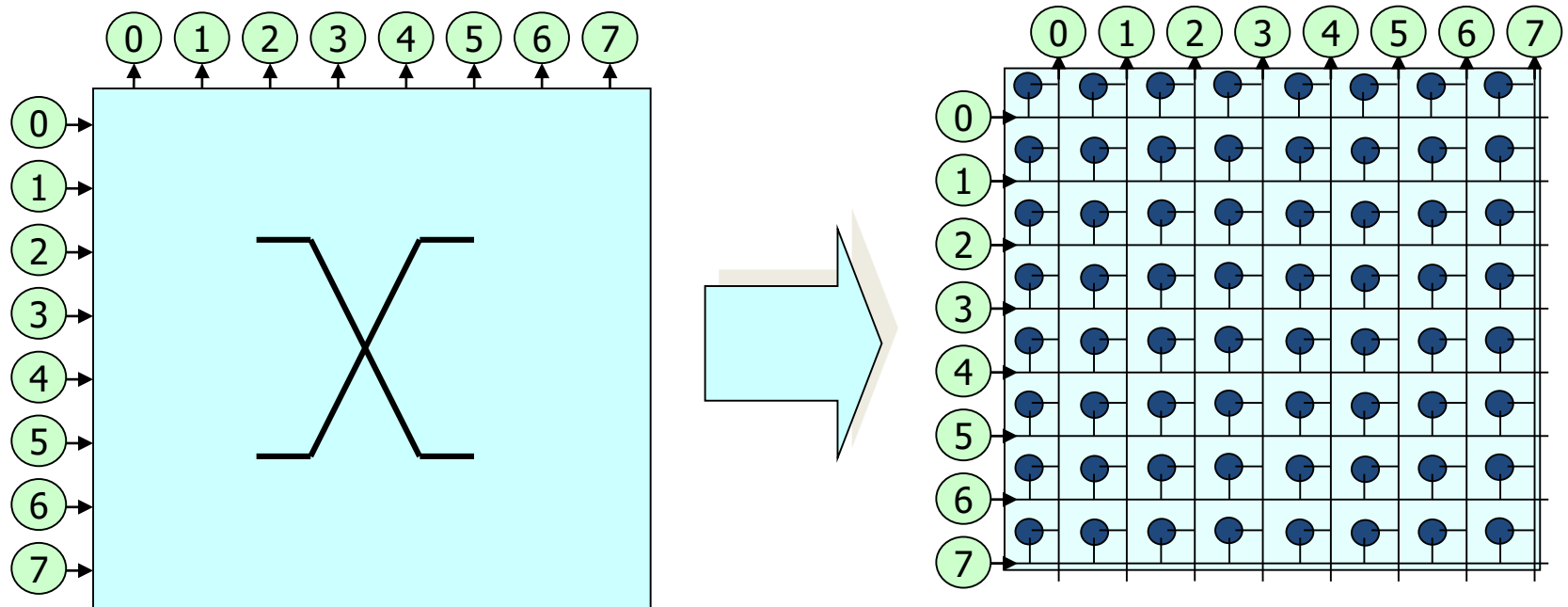
Standard criteria

- **Node degree** d - the number of edges incident on a node
 - In degree/Out degree
- **Network Diameter** D of a network is the maximum shortest path between any two nodes
- **Network bisection width** Minimum number of links to be cut for a network to be divided into two halves
- **Symmetry** The network looks the same from any node
- **Scalability** The network is *scalable* if it is expandable with scalable performance when the machine resources are increased

Network Topology

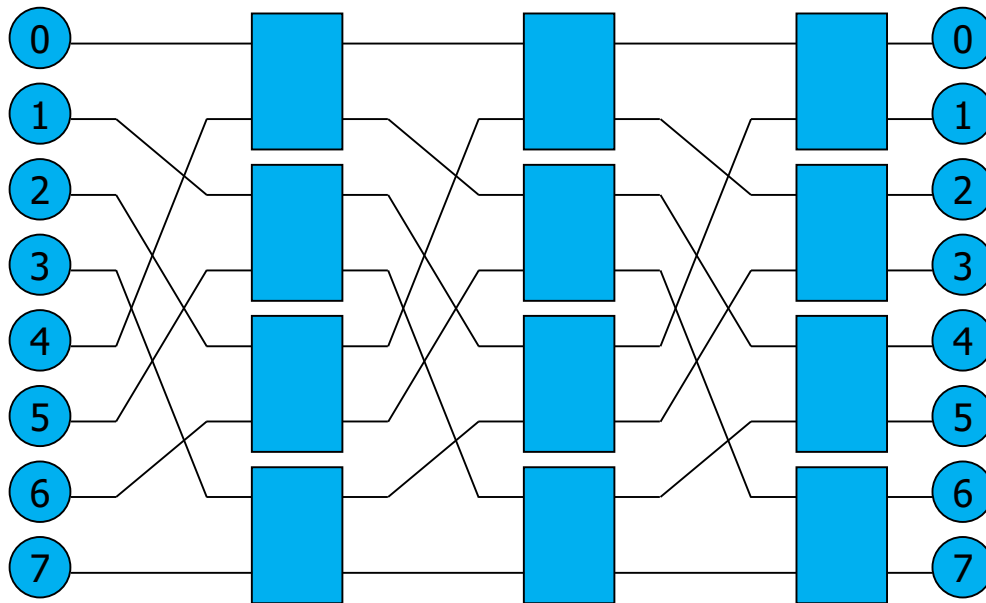
- **Crossbar network**

- Crosspoint switch complexity increases quadratically with the number of crossbar input/output ports, N , i.e., grows as $O(N^2)$
- Has the property of being *non-blocking*



Network Topology

- **Multistage interconnection networks (MINs)**
 - Crossbar split into several stages consisting of smaller crossbars
 - Complexity grows as $O(N \times \log N)$, where N is # of end nodes
 - Inter-stage connections represented by a set of permutation functions



Omega
topology,
perfect-shuffle
exchange

Network Topology

- Multistage interconnection networks (MINs)
 - MINs interconnect N input/output ports using $k \times k$ switches
 - $\log_k N$ switch stages, each with N/k switches
 - $N/k(\log_k N)$ total number of switches
 - **Example** Compute the switch and link costs of interconnecting 4096 nodes using a crossbar relative to a MIN, assuming that switch cost grows quadratically with the number of input/output ports (k).

Consider the following values of k :

- MIN with 2×2 switches
- MIN with 4×4 switches
- MIN with 16×16 switches

Network Topology

Multistage interconnection networks (MINs)

- **Example** Compute the switch and link costs $N=4096$ nodes

$$\text{cost}(\text{crossbar})_{\text{switches}} = 4096^2$$

$$\text{cost}(\text{crossbar})_{\text{links}} = 8192$$

$$\text{relative_cost}(2 \times 2)_{\text{switches}} = 4096^2 / (2^2 \times 4096/2 \times \log_2 4096) = 170$$

$$\text{relative_cost}(2 \times 2)_{\text{links}} = 8192 / (4096 \times (\log_2 4096 + 1)) = 2/13 = 0.1538$$

$$\text{relative_cost}(4 \times 4)_{\text{switches}} = 4096^2 / (4^2 \times 4096/4 \times \log_4 4096) = 170$$

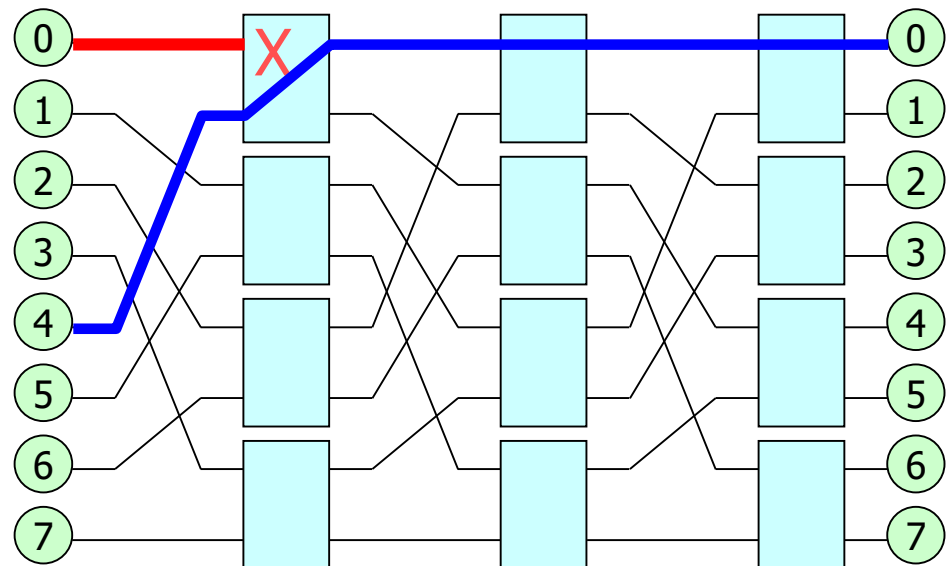
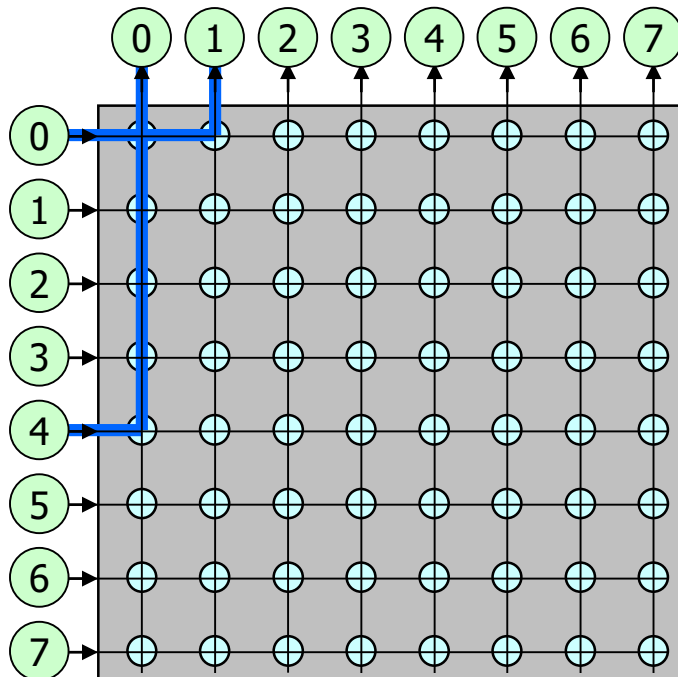
$$\text{relative_cost}(4 \times 4)_{\text{links}} = 8192 / (4096 \times (\log_4 4096 + 1)) = 2/7 = 0.2857$$

$$\text{relative_cost}(16 \times 16)_{\text{switches}} = 4096^2 / (16^2 \times 4096/16 \times \log_{16} 4096) = 85$$

$$\text{relative_cost}(16 \times 16)_{\text{links}} = 8192 / (4096 \times (\log_{16} 4096 + 1)) = 2/4 = 0.5$$

Network Topology

- **Cost reduction** in MIN switch \rightarrow **performance reduction**
 - The MIN is **blocking**
 - Paths from different sources to different destinations can require to set a switch straight and cross at the same time (or to share the same link)
 - Consider the requests $0 \rightarrow 1$ and $1 \rightarrow 4$



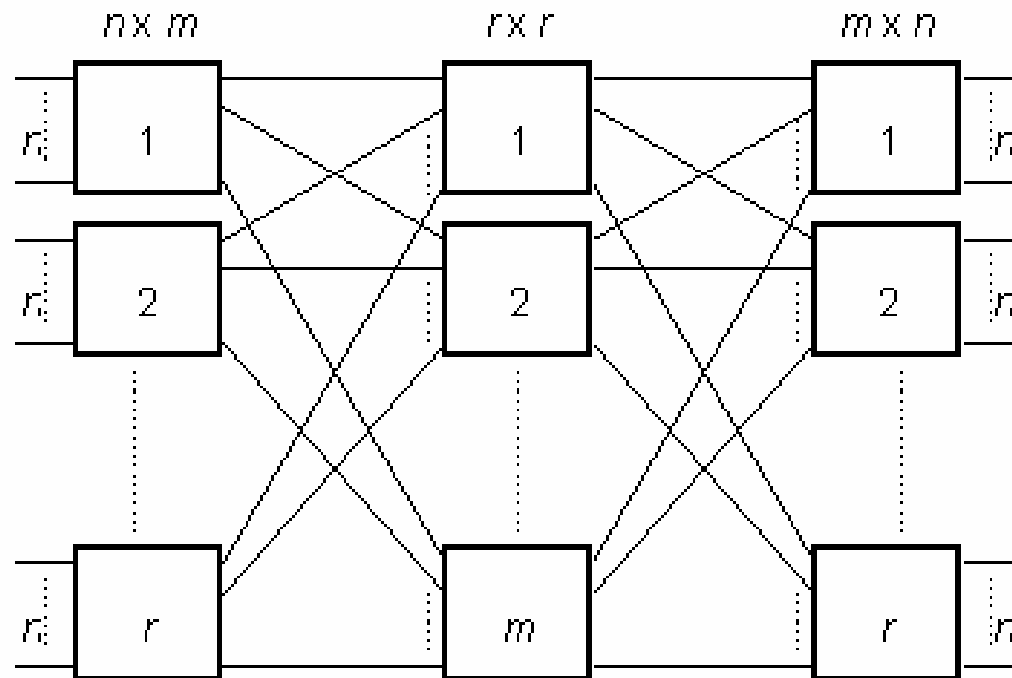
Network Topology

- To reduce blocking in MINs → *Provide alternative paths*
 - **Use larger switches** (can equate to using more switches)
 - **Clos network**: minimally three stages (non-blocking)
 - A larger switch in the middle of two other switch stages provides enough alternative paths to avoid all conflicts
 - **Use more switches**
 - Add $\log_k N - 1$ stages, mirroring the original topology
 - *Rearrangeably non-blocking*
 - Allows for non-conflicting paths
 - Doubles network *hop count (distance)*, *d*
 - Centralized control can rearrange established paths
 - **Benes topology**: $2(\log_2 N) - 1$ stages (rearrangeable non-blocking)
 - Recursively applies the three-stage Clos network concept to the middle-stage set of switches to reduce all switches to 2×2

CLOS NETWORK

Clos network

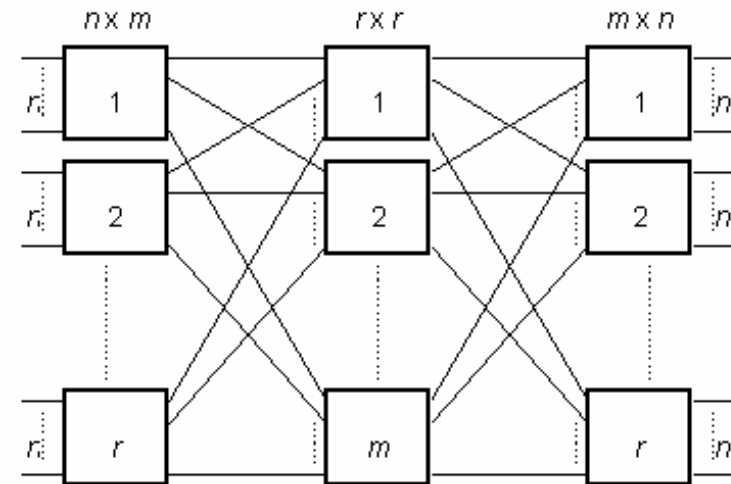
- Clos network is a *multistage switching network*
- Clos networks have **three stages** - the *ingress stage*, *middle stage*, and the *egress stage* - made up of **crossbars**



Clos network

Clos networks are defined by three integers n , m , and r

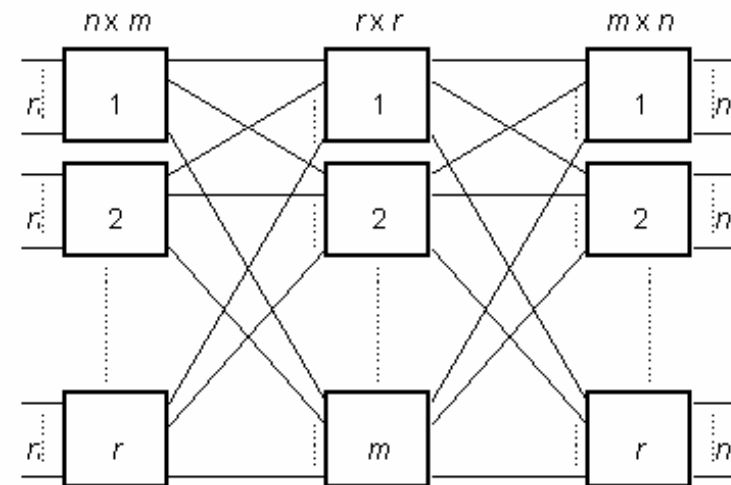
- n is the number of
 - input of each (of the r) ingress stage crossbar switches
 - output of each (of the r) egress stage crossbar switches
- r is the number of
 - crossbar switches in the ingress stage
 - crossbar switches in the egress stage
 - input and output of switches in the middle stage crossbar switches
- m is the number of
 - middle stage crossbar switches
 - output of each (of the r) ingress stage crossbar switches
 - input of each (of the r) egress stage crossbar switches



Clos network

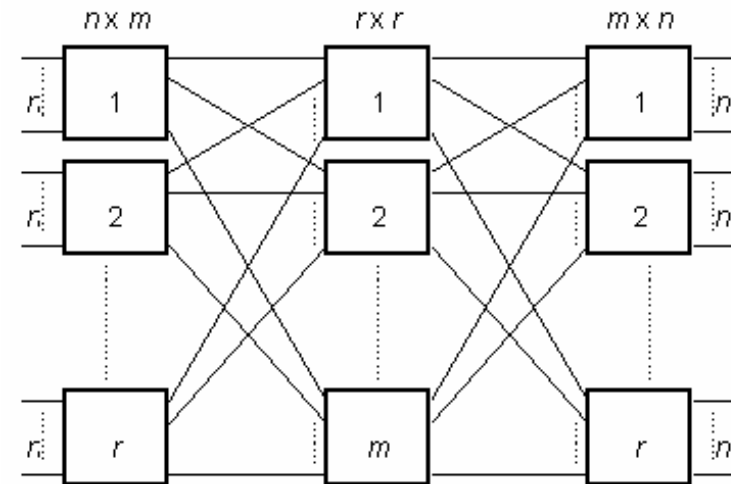
Thus:

- The **ingress stage** has **r switches** $n \times m$
- The **middle stage** has **m switches** - $r \times r$
- The **egress stage** has **r switches** - $m \times n$
- **Each middle stage switch** is connected exactly once to **each ingress stage switch** and to **each egress stage switch**



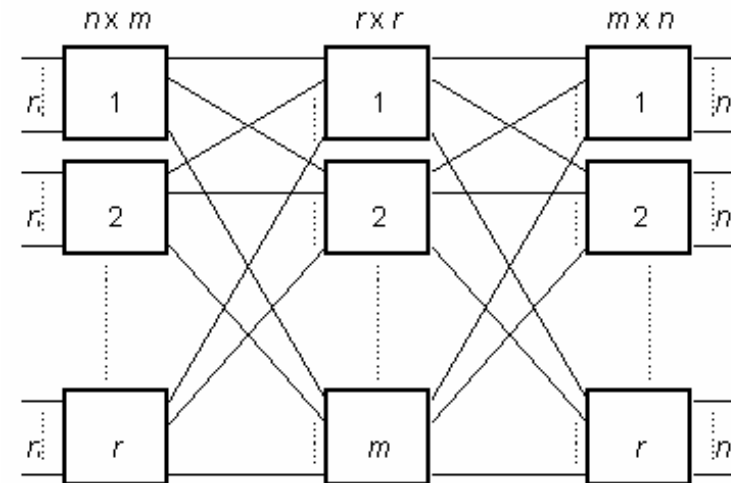
Clos network

- Each call entering an **ingress crossbar** can be routed through **any** of the available **middle stage crossbar**, to the relevant **egress crossbar** switch
- A middle stage crossbar is **available** for a new call if **both the link connecting the ingress switch to the middle stage switch, and the link connecting the middle stage switch to the egress switch, are free**



Clos network

- The **advantage** of Clos network is that connection between a **large number of input and output** ports can be made by using only **small-sized switches**



Strict-sense nonblocking Clos networks

- If $m \geq 2n-1$, the Clos network is *strict-sense nonblocking* (Clos paper 1953)
- This means that an **unused input on an ingress switch** can **always** be connected to an **unused output on an egress switch**, *without having to re-arrange existing calls*

Strict-sense nonblocking Clos networks

- Assume that:
 - there is a free terminal on the input of an ingress switch, and
 - this has to be connected to a free terminal on a particular egress switch
- In the worst case:
 - $n-1$ other calls are active on the ingress switch in question, and
 - $n-1$ other calls are active on the egress switch in question
- Assume, also in the worst case, that:
 - each of these calls passes through a different middle-stage switch
- Hence, in the worst case:
 - $2n-2$ of the middle stage switches are unable to carry the new call
- Therefore, to ensure strict-sense nonblocking operation, another middle stage switch is required, making a total of $2n-1$

Rearrangeably nonblocking Clos networks

- If $m \geq n$, the Clos network is *rearrangeably nonblocking*
- This means that an **unused input on an ingress** switch can **always** be connected to an **unused output on an egress** switch, but for this to take place, **existing calls may have to be rearranged** by assigning them to different middle stage switches in the Clos network
- To prove this, it is sufficient to consider $m = n$, with the Clos network fully utilised; that is, $r \times n$ calls in progress

Rearrangeably nonblocking Clos networks

- The proof shows how any permutation of these $r \times n$ input terminals onto $r \times n$ output terminals may be broken down into smaller permutations which may each be implemented by the individual crossbar switches in a Clos network with $m = n$
- The proof uses **Hall's marriage theorem**
 - Suppose there are r boys and r girls
 - The theorem states that if every subset of k boys (for each k such that $0 \leq k \leq r$) between them know k or more girls, then each boy can be paired off with a girl that he knows
 - This is a (obvious) necessary condition for pairing to take place; and it is also sufficient

Rearrangeably nonblocking Clos networks

- In the context of a Clos network, each boy represents an ingress switch, and each girl represents an egress switch
- A boy is said to know a girl if the corresponding ingress and egress switches carry the same call
- Each set of k boys must know at least k girls because k ingress switches are carrying $k \times n$ calls and these cannot be carried by less than k egress switches

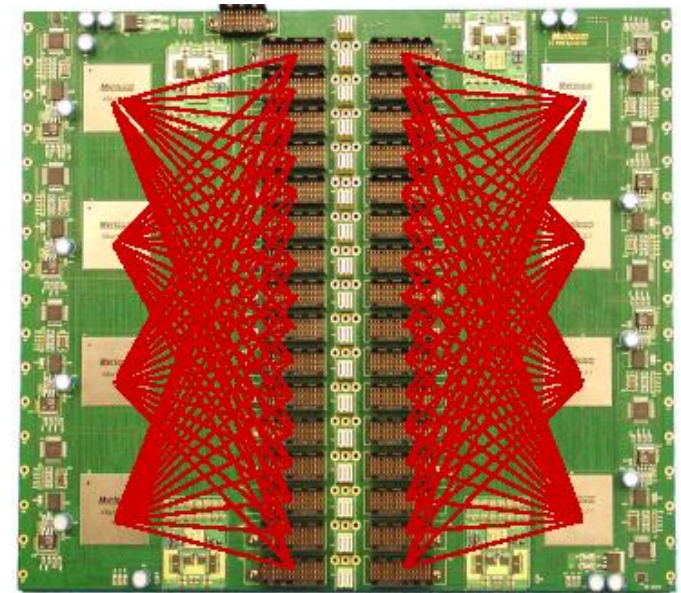
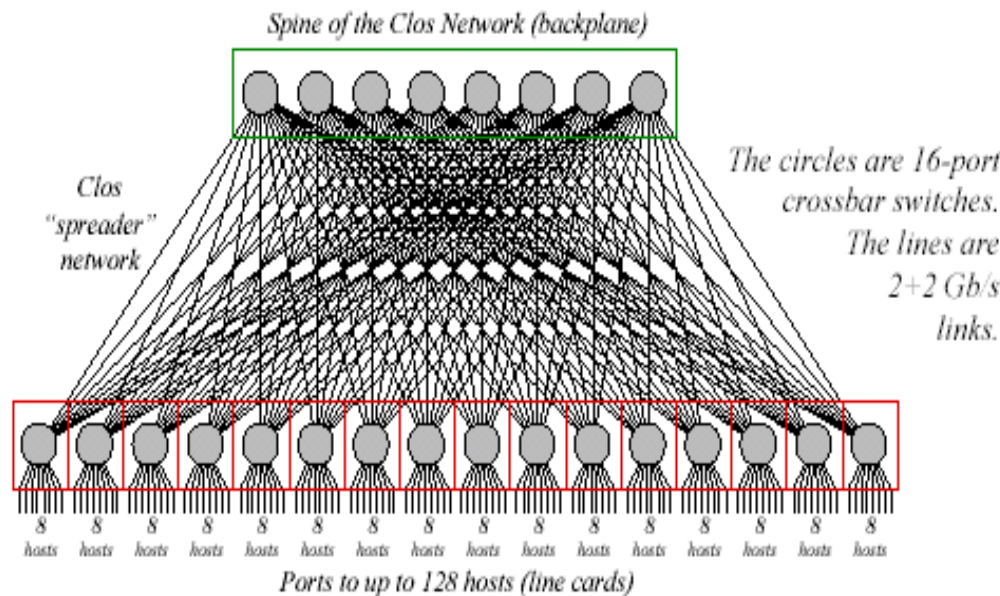
Rearrangeably nonblocking Clos networks

- Hence each ingress switch can be paired off with an egress switch that carries the same call, via a one-to-one mapping
- These r calls can be carried by one middle-stage switch
- If this middle-stage switch is now removed from the Clos network, m is reduced by 1, and we are left with a smaller Clos network
- The process then repeats itself until $m = 1$, and every call is assigned to a middle-stage switch

Network Topology

- Myrinet-2000 Clos Network for 128 hosts

<http://myri.com>



↑
Backplane of the
M3-E128 Switch

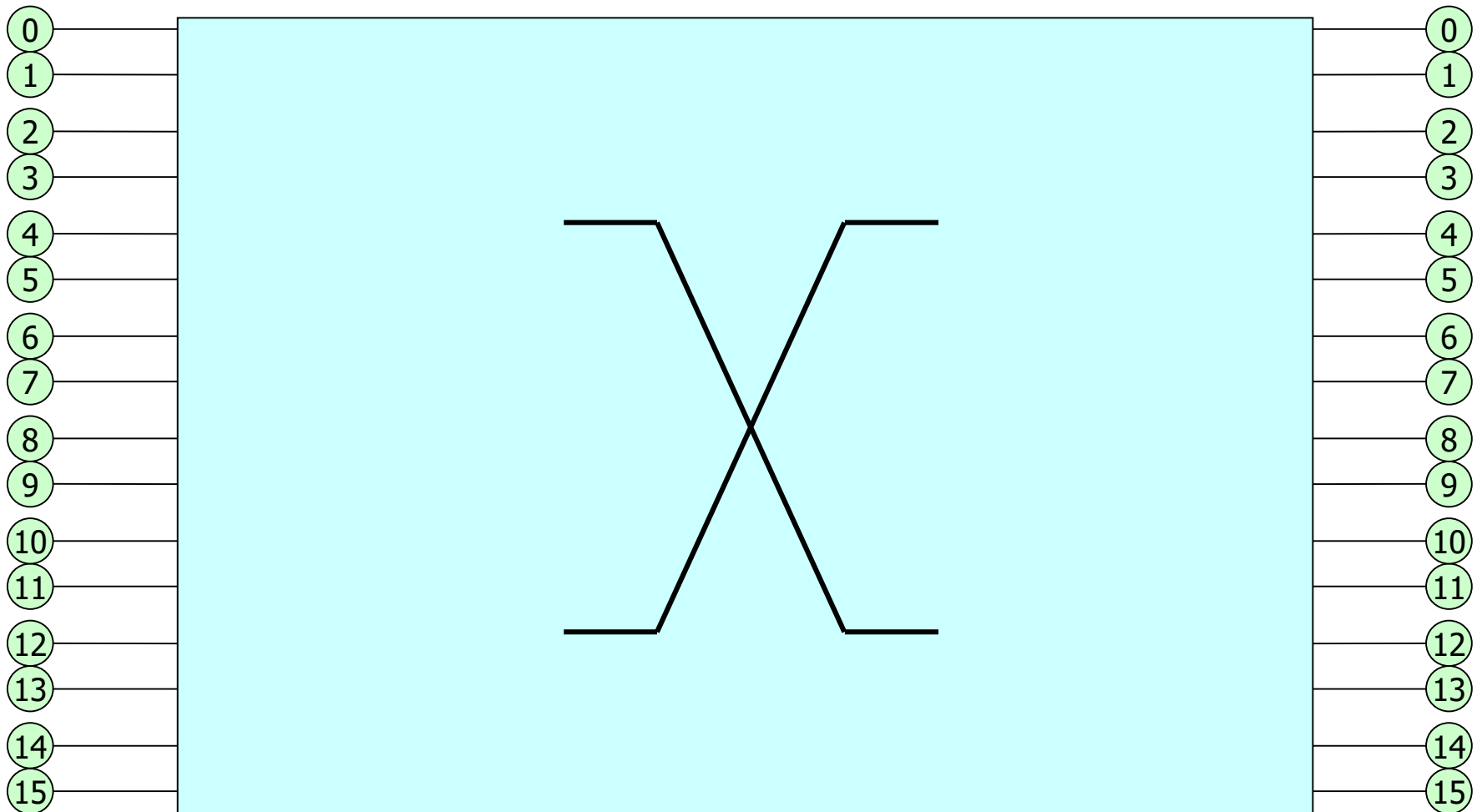


←
M3-SW16-8F fiber
line card (8 ports)

BENES NETWORK

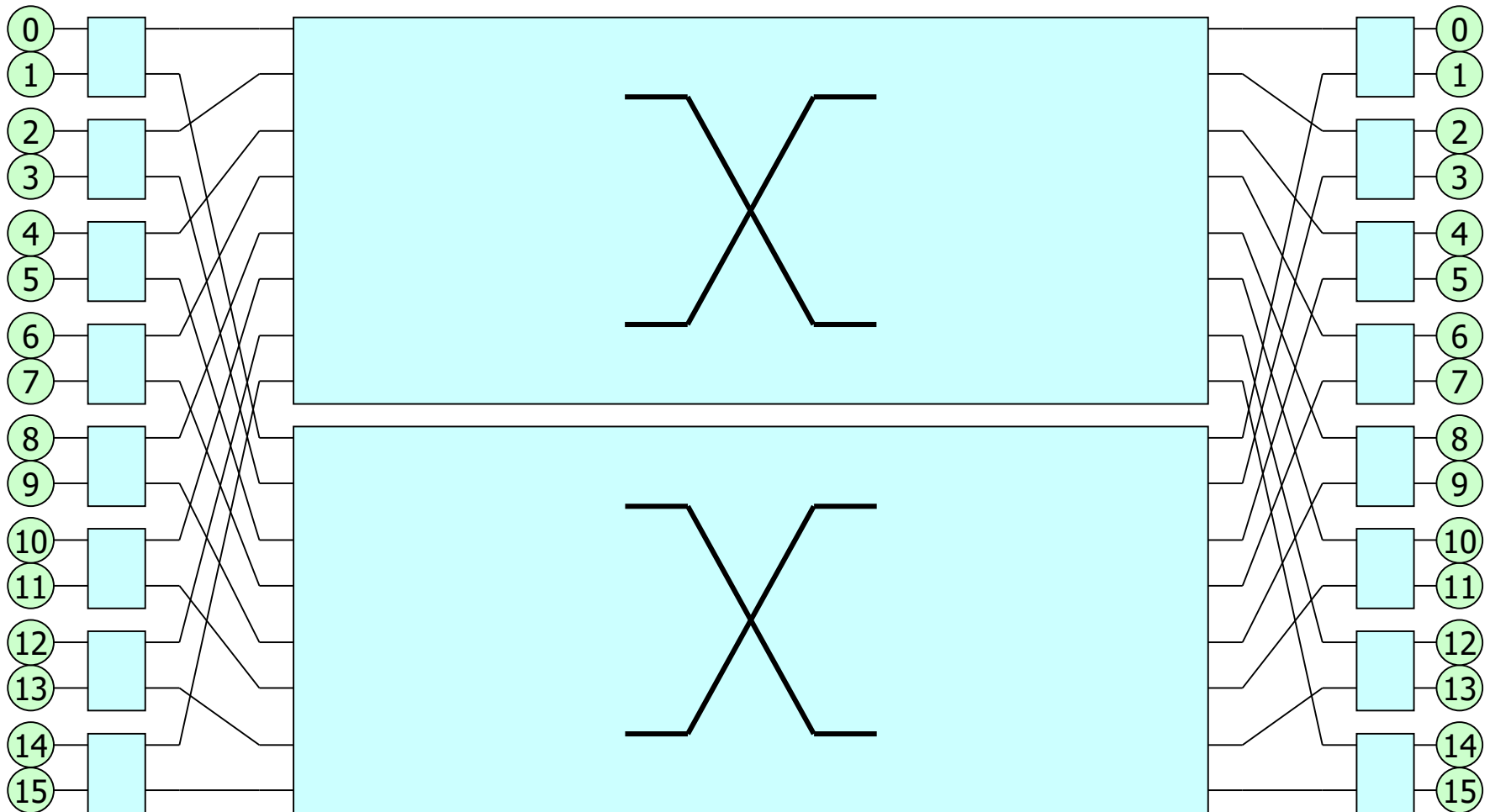
Benes Network

16 port **Crossbar** network



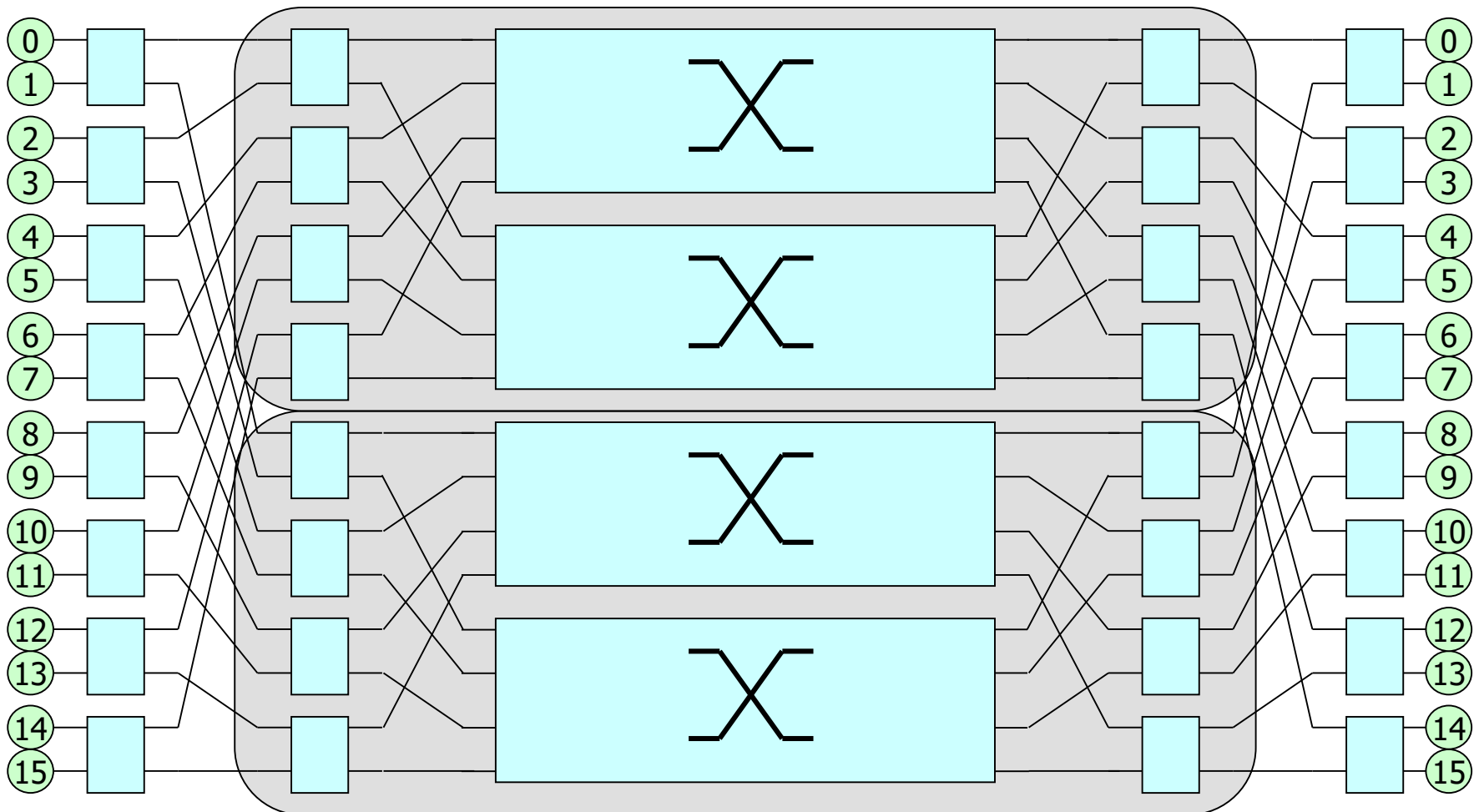
Benes Network

16 port, **3 stage Clos** network



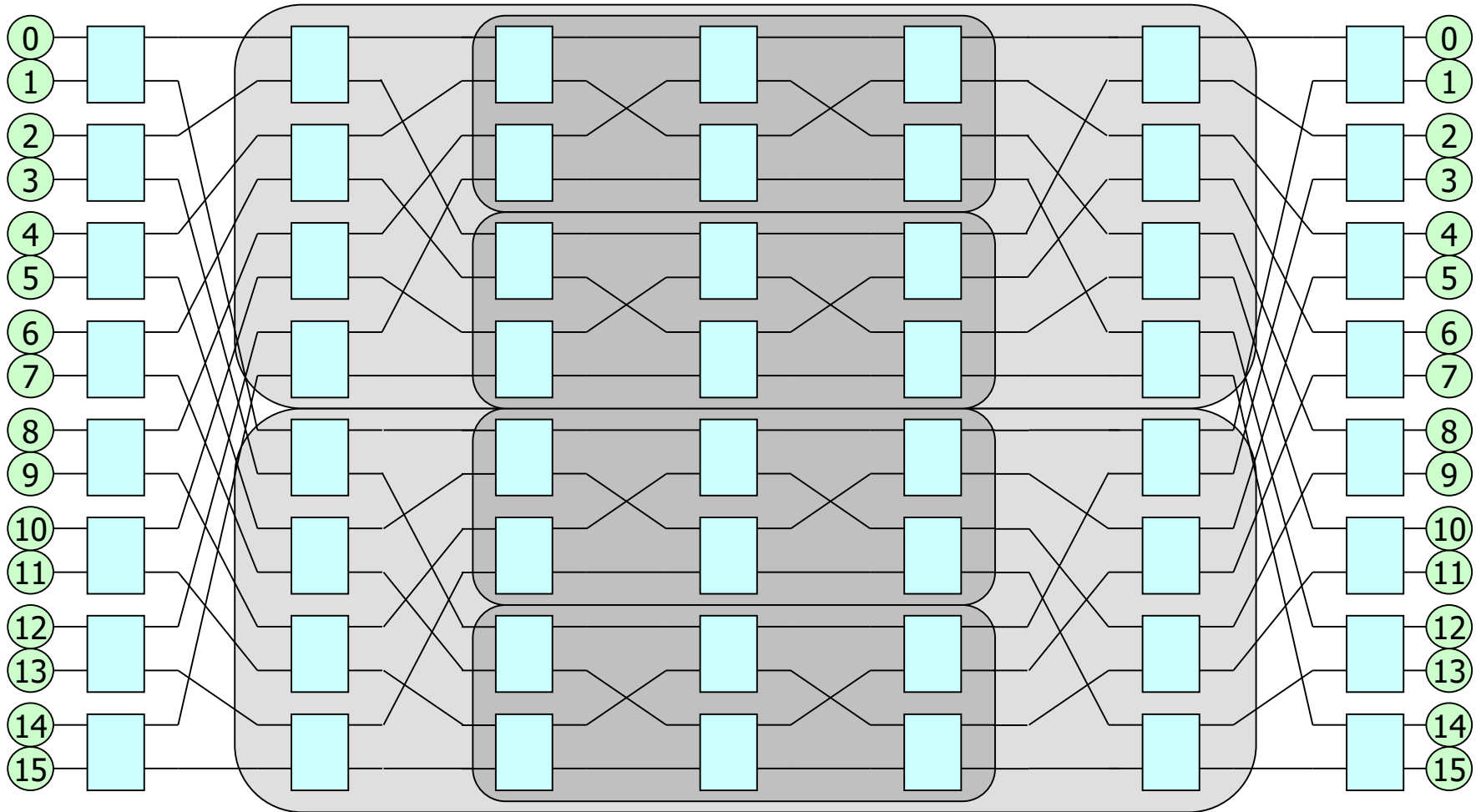
Benes Network

16 port, 5 stage Clos network



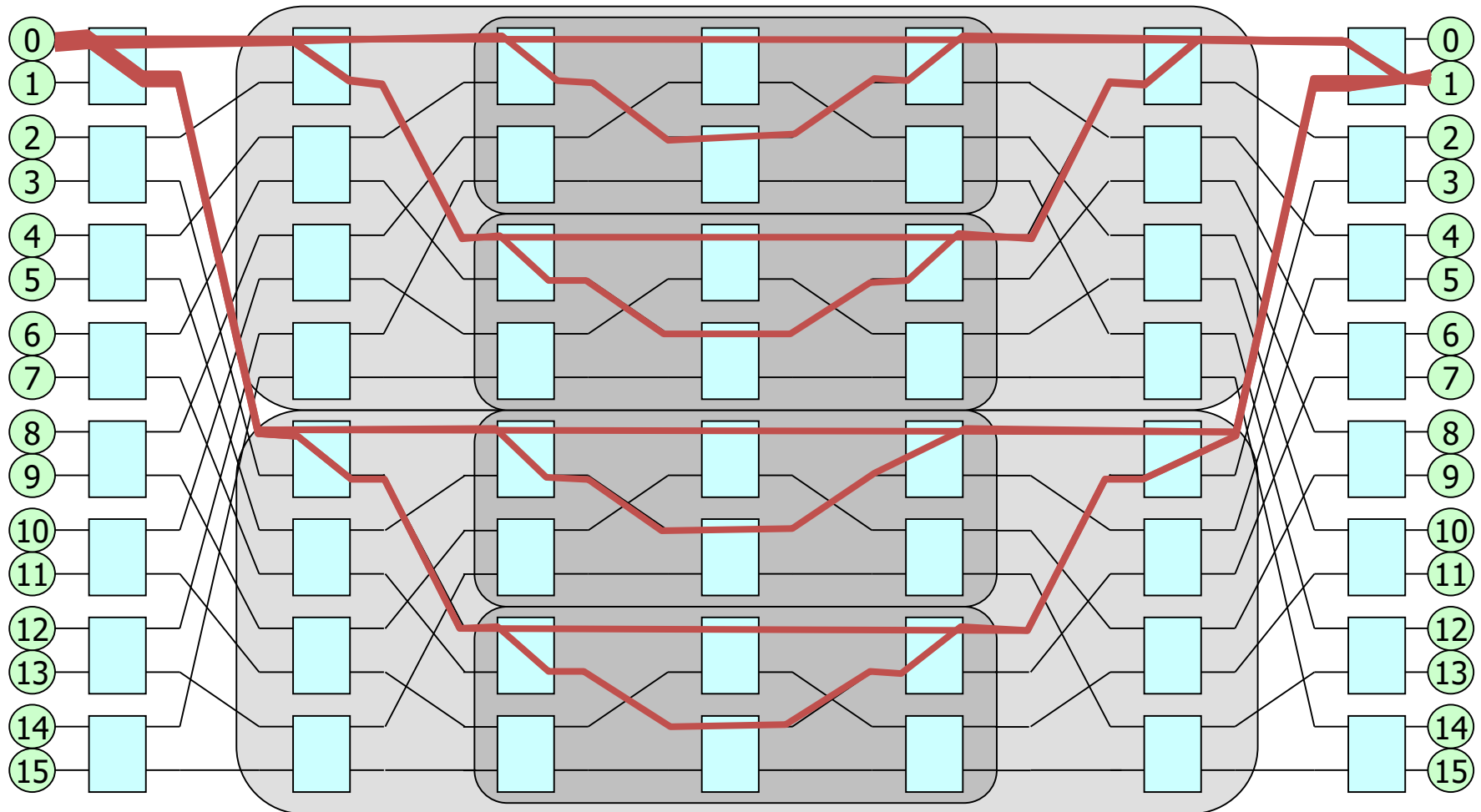
Benes Network

16 port, **7 stage Clos** network = **Benes topology**



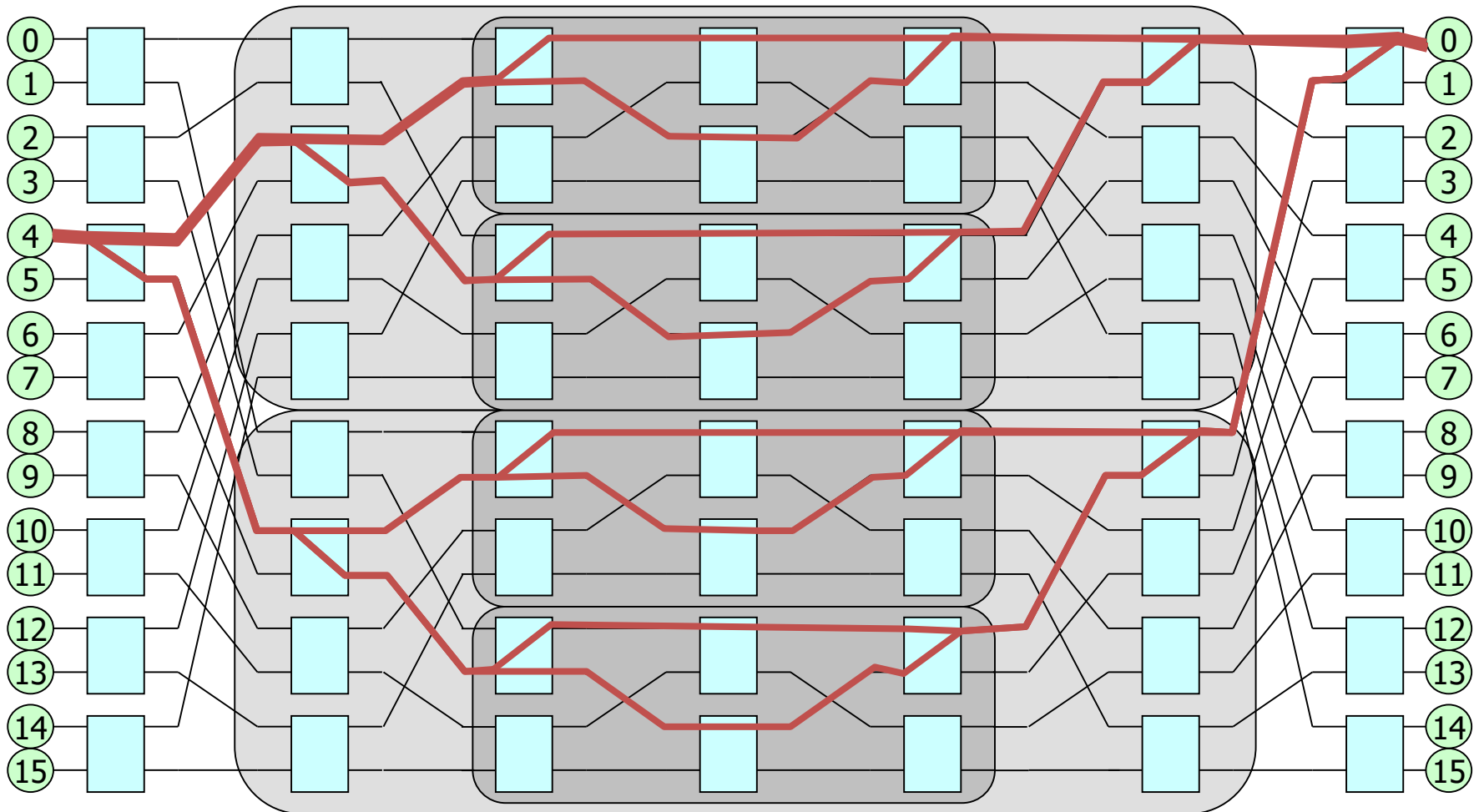
Benes Network

Alternative paths from 0 to 1 in a 16 port **Benes topology**



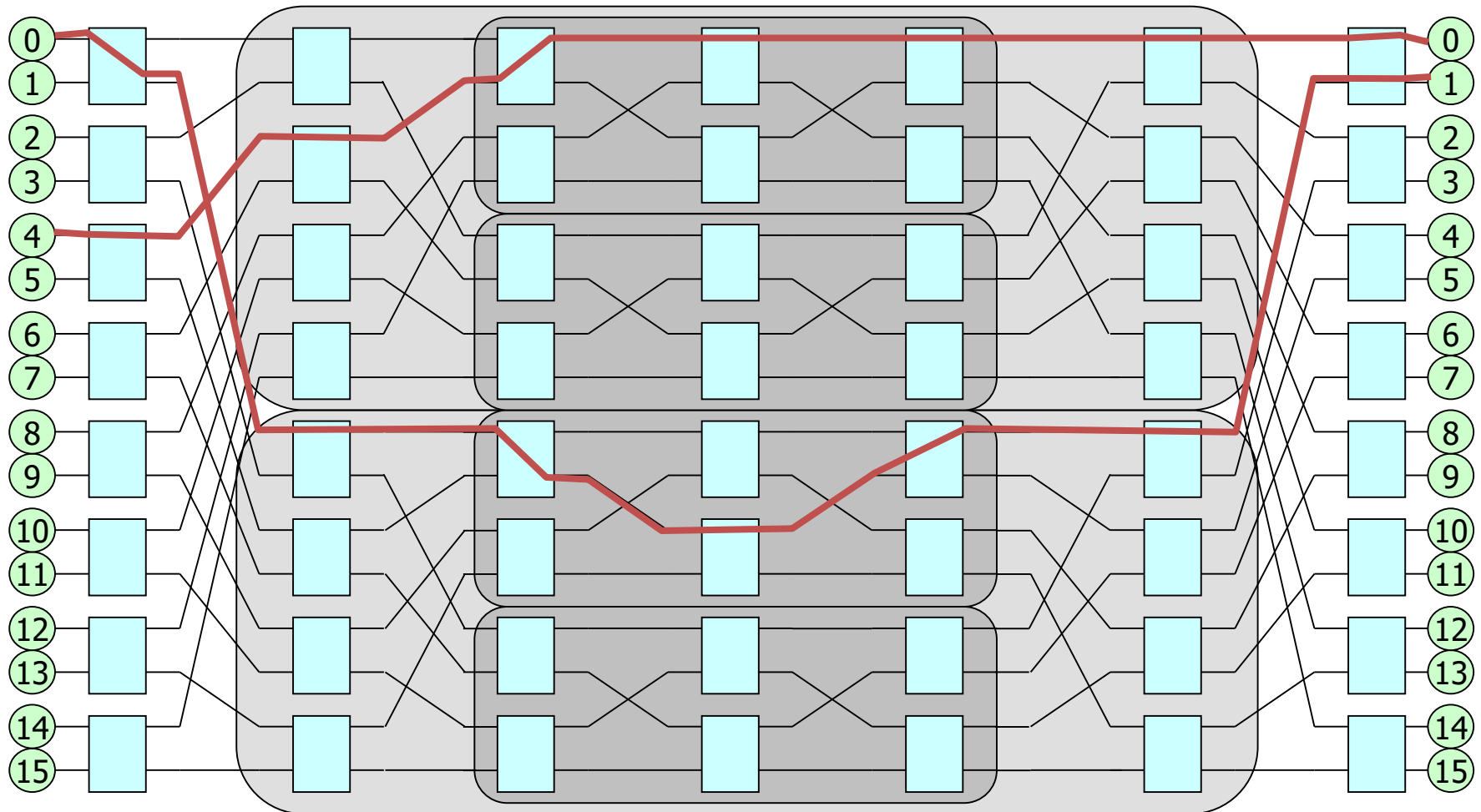
Benes Network

Alternative paths from 4 to 0 in a 16 port **Benes topology**



Benes Network

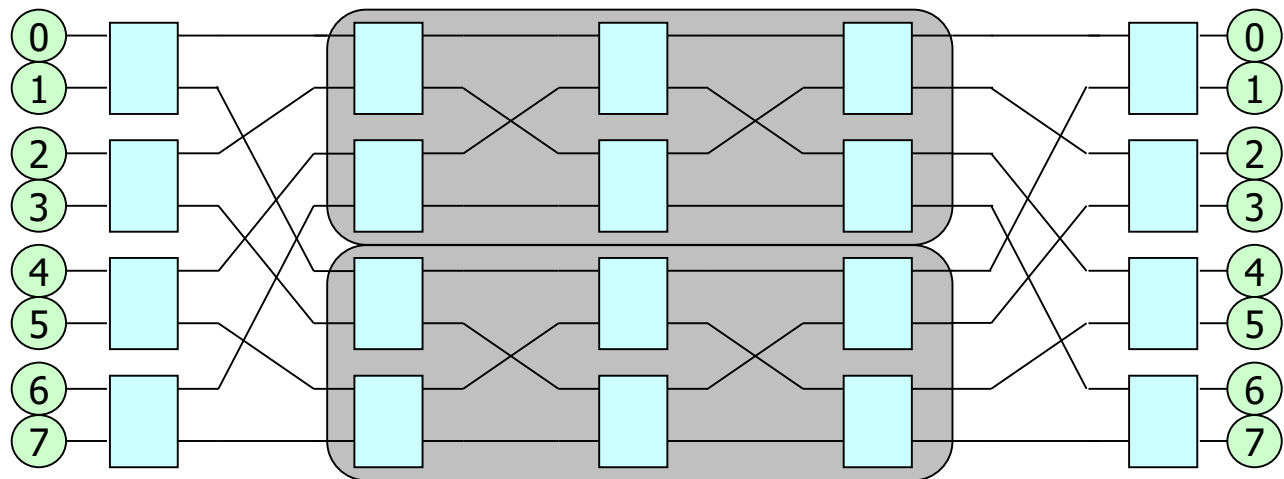
Contention free, paths 0 to 1 and 4 to 1 in a 16 port **Benes topology**



Looping algorithm

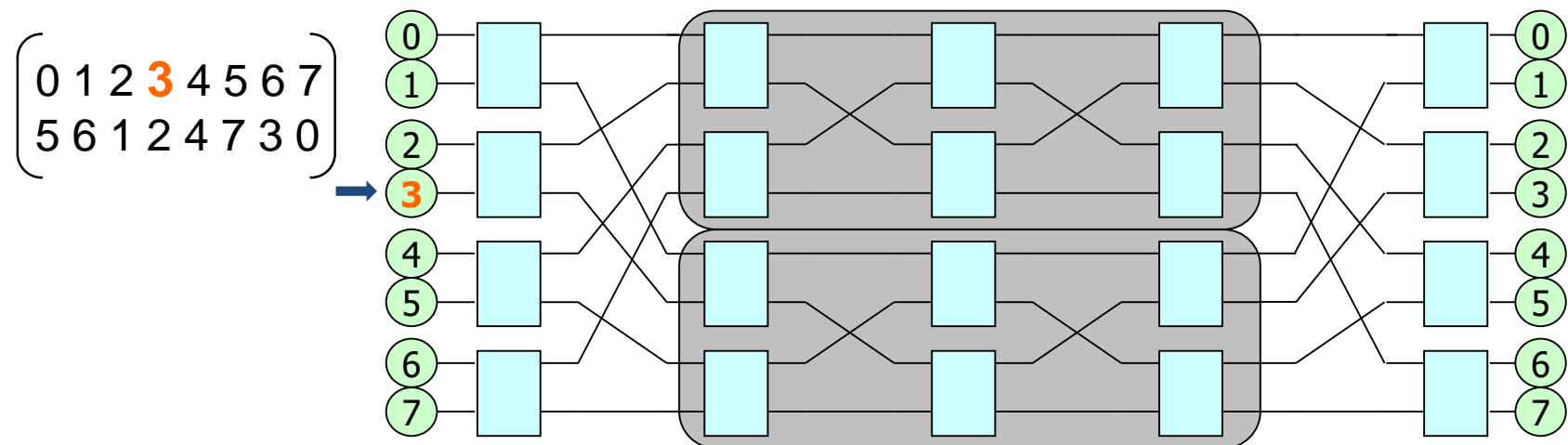
Realizing permutations on a Benes network

- ▶ Start from **arbitrarily chosen input** by arbitrarily setting the corresponding switch
- ▶ **Connect** the input to the requested output
- ▶ **Connect back** the other output of the switch in the last stage to the corresponding input
- ▶ The algorithm follows this procedure, **looping back and forth** between inputs and outputs, until the original switch is reached
- ▶ If there are inputs not connected, the algorithm starts again from a free input



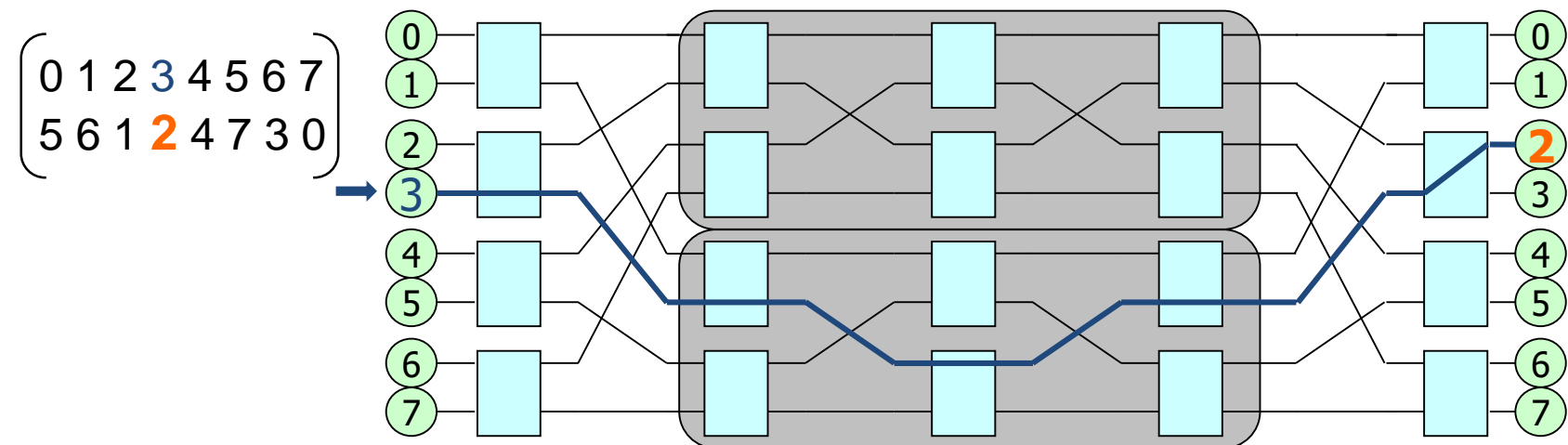
Looping algorithm

- ▶ **Example on a Benes network of size N=8**
 - ▶ The algorithm starts from an arbitrarily chosen
 - ▶ The input is connected to the requested output
 - ▶ The other output of the switch in the last stage is connected to the corresponding input
 - ▶ The algorithm follows this procedure, looping back and forth between inputs and outputs, until the original switch is reached
 - ▶ If there are inputs not connected, the algorithm starts again from a free input



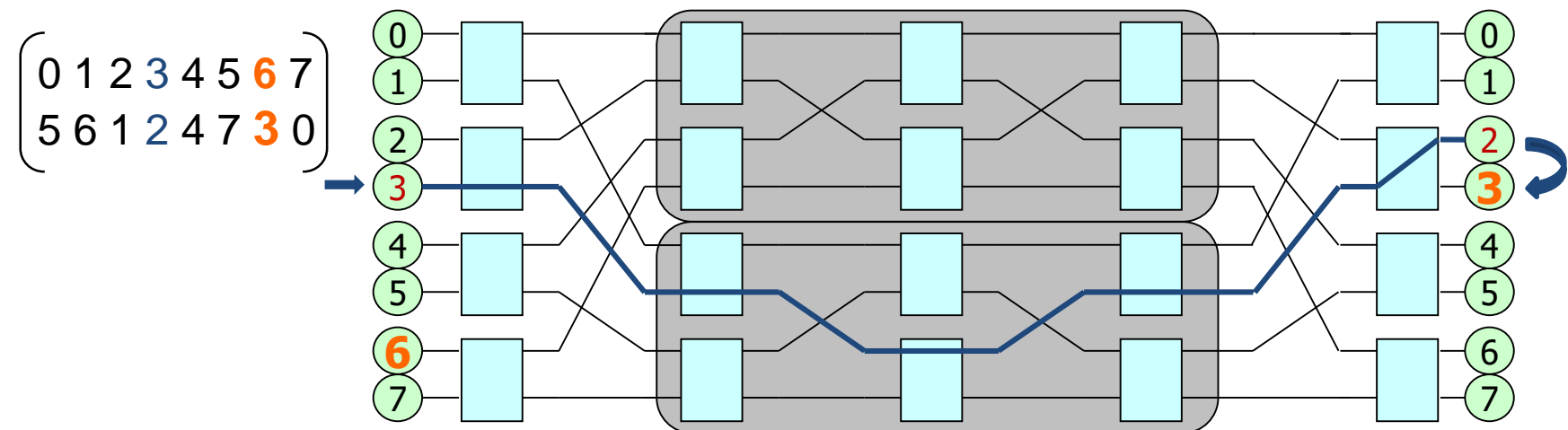
Looping algorithm

- ▶ **Example on a Benes network of size N=8**
 - ▶ The algorithm starts from an arbitrarily chosen
 - ▶ The input is connected to the requested output
 - ▶ The other output of the switch in the last stage is connected to the corresponding input
 - ▶ The algorithm follows this procedure, looping back and forth between inputs and outputs, until the original switch is reached
 - ▶ If there are inputs not connected, the algorithm starts again from a free input



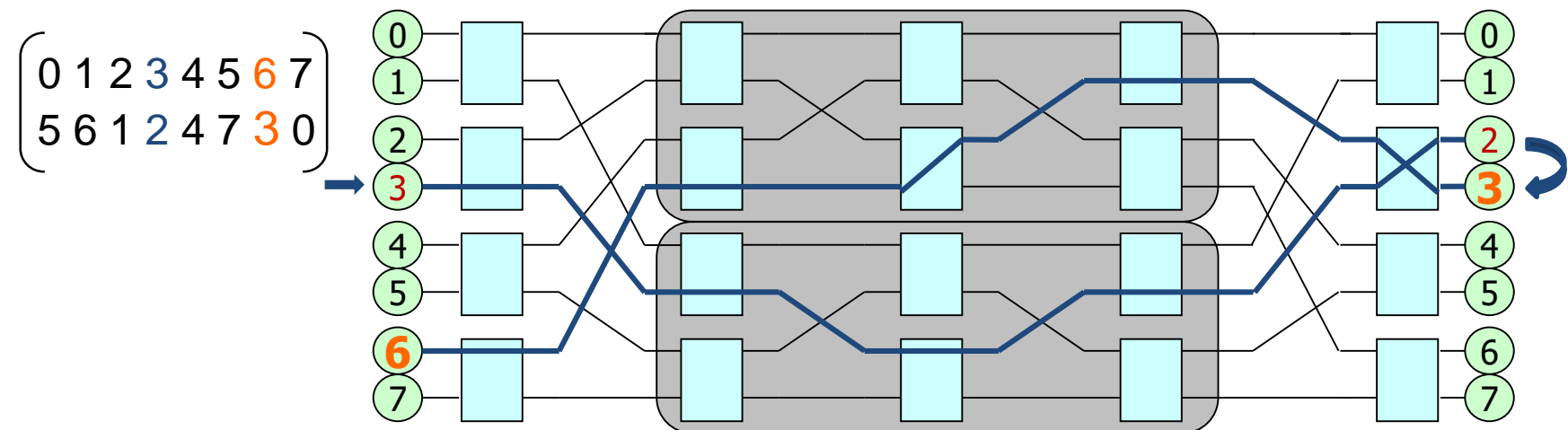
Looping algorithm

- ▶ Example on a Benes network of size $N=8$
 - ▶ The algorithm starts from an arbitrarily chosen
 - ▶ The input is connected to the requested output
 - ▶ The other output of the switch in the last stage is connected to the corresponding input
 - ▶ The algorithm follows this procedure, looping back and forth between inputs and outputs, until the original switch is reached
 - ▶ If there are inputs not connected, the algorithm starts again from a free input



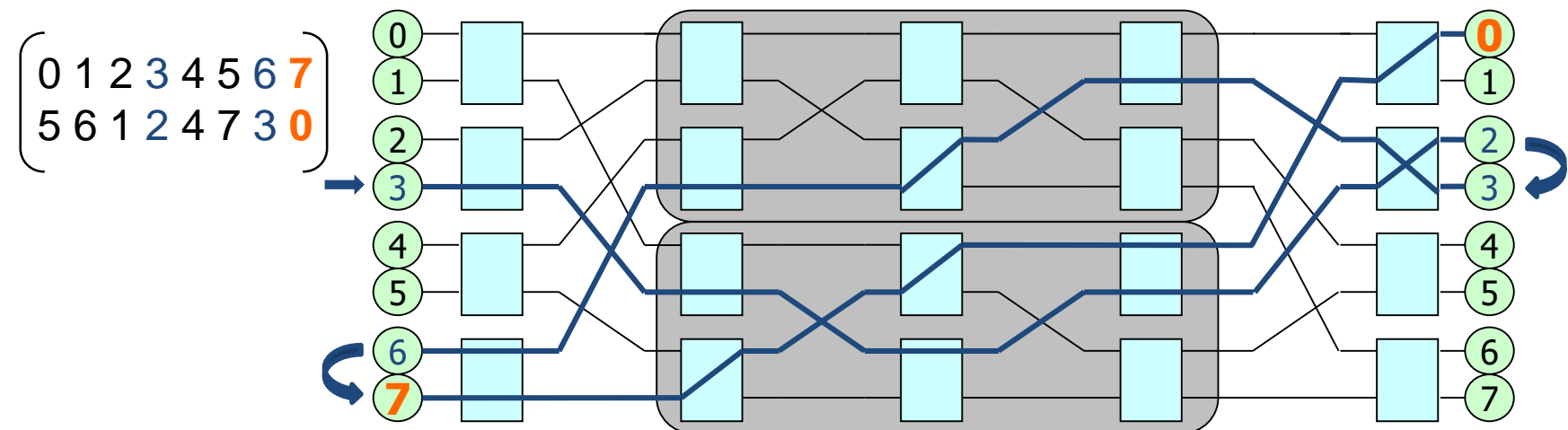
Looping algorithm

- ▶ Example on a Benes network of size $N=8$
 - ▶ The algorithm starts from an arbitrarily chosen
 - ▶ The input is connected to the requested output
 - ▶ The other output of the switch in the last stage is connected to the corresponding input
 - ▶ The algorithm follows this procedure, looping back and forth between inputs and outputs, until the original switch is reached
 - ▶ If there are inputs not connected, the algorithm starts again from a free input



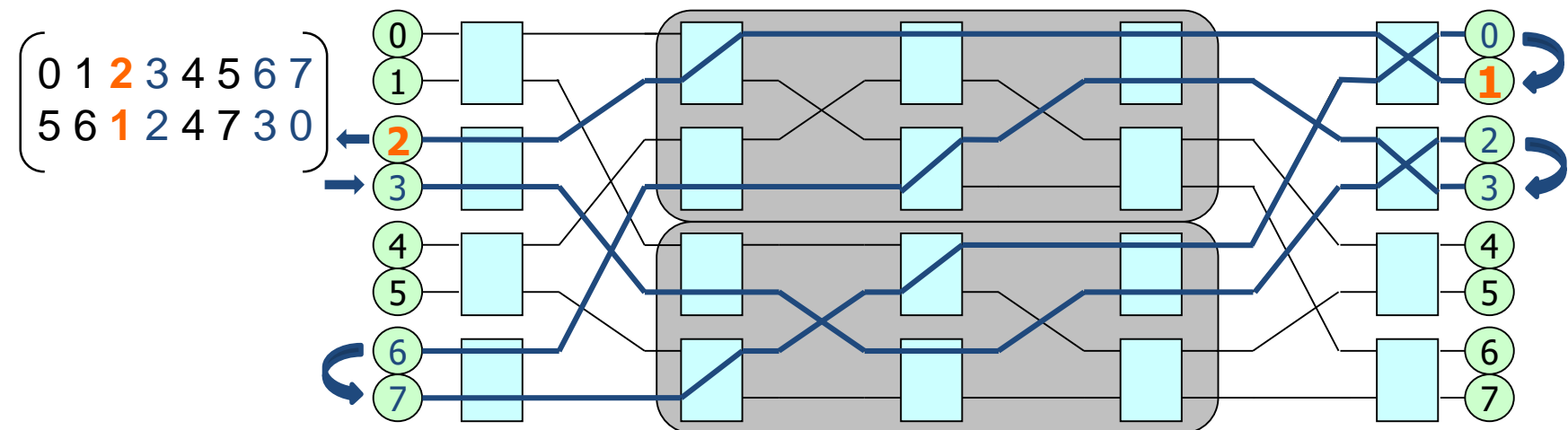
Looping algorithm

- ▶ Example on a Benes network of size $N=8$
 - ▶ The algorithm starts from an arbitrarily chosen
 - ▶ The input is connected to the requested output
 - ▶ The other output of the switch in the last stage is connected to the corresponding input
 - ▶ The algorithm follows this procedure, looping back and forth between inputs and outputs, until the original switch is reached
 - ▶ If there are inputs not connected, the algorithm starts again from a free input



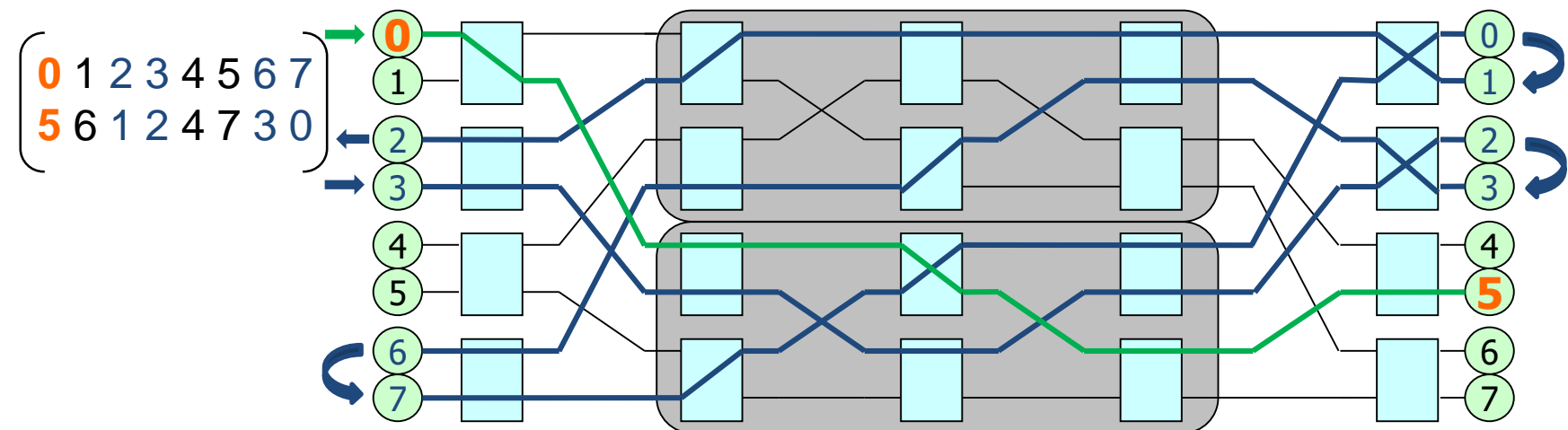
Looping algorithm

- ▶ Example on a Benes network of size $N=8$
 - ▶ The algorithm starts from an arbitrarily chosen
 - ▶ The input is connected to the requested output
 - ▶ The other output of the switch in the last stage is connected to the corresponding input
 - ▶ The algorithm follows this procedure, looping back and forth between inputs and outputs, until the original switch is reached
 - ▶ If there are inputs not connected, the algorithm starts again from a free input



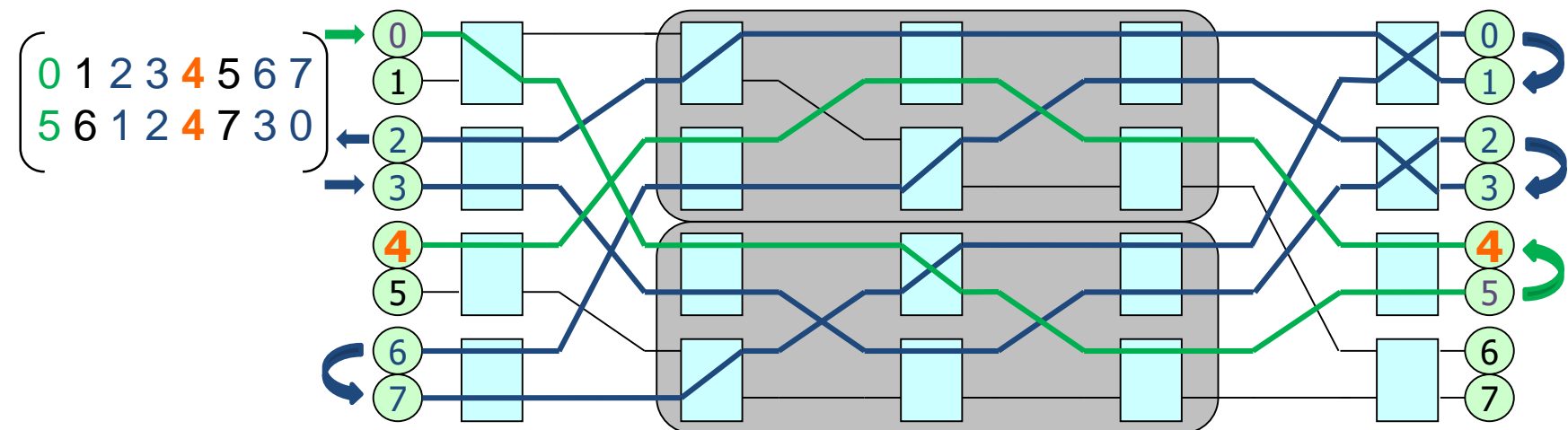
Looping algorithm

- ▶ Example on a Benes network of size $N=8$
 - ▶ The algorithm starts from an arbitrarily chosen
 - ▶ The input is connected to the requested output
 - ▶ The other output of the switch in the last stage is connected to the corresponding input
 - ▶ The algorithm follows this procedure, **looping back and forth** between inputs and outputs, until the original switch is reached
 - ▶ If there are inputs not connected, the algorithm starts again from a free input



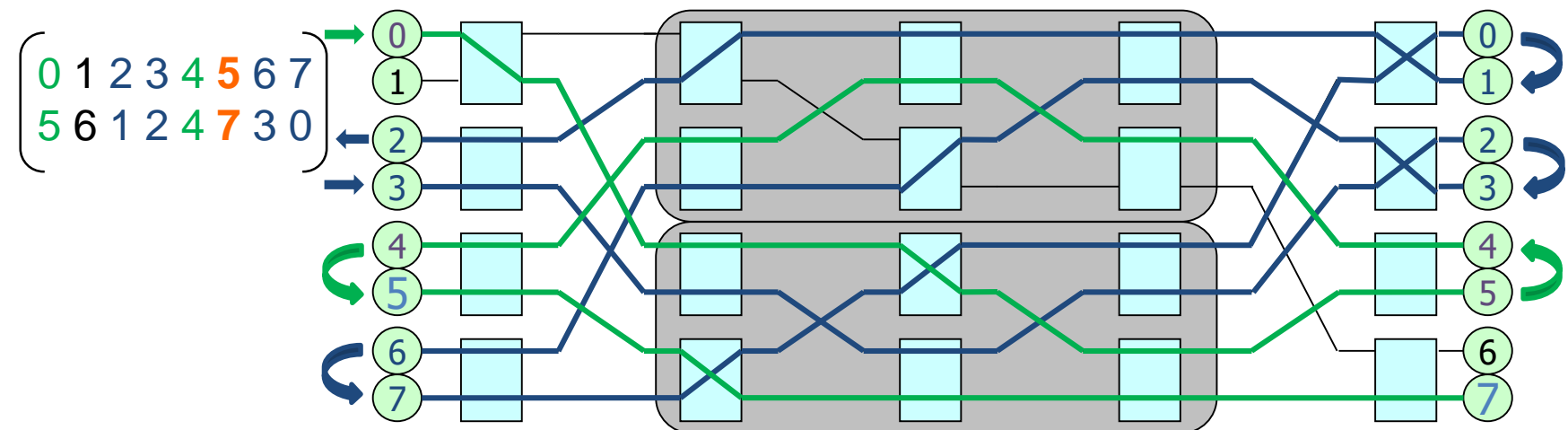
Looping algorithm

- ▶ Example on a Benes network of size $N=8$
 - ▶ The algorithm starts from an arbitrarily chosen
 - ▶ The input is connected to the requested output
 - ▶ The other output of the switch in the last stage is connected to the corresponding input
 - ▶ The algorithm follows this procedure, **looping back and forth** between inputs and outputs, until the original switch is reached
 - ▶ If there are inputs not connected, the algorithm starts again from a free input



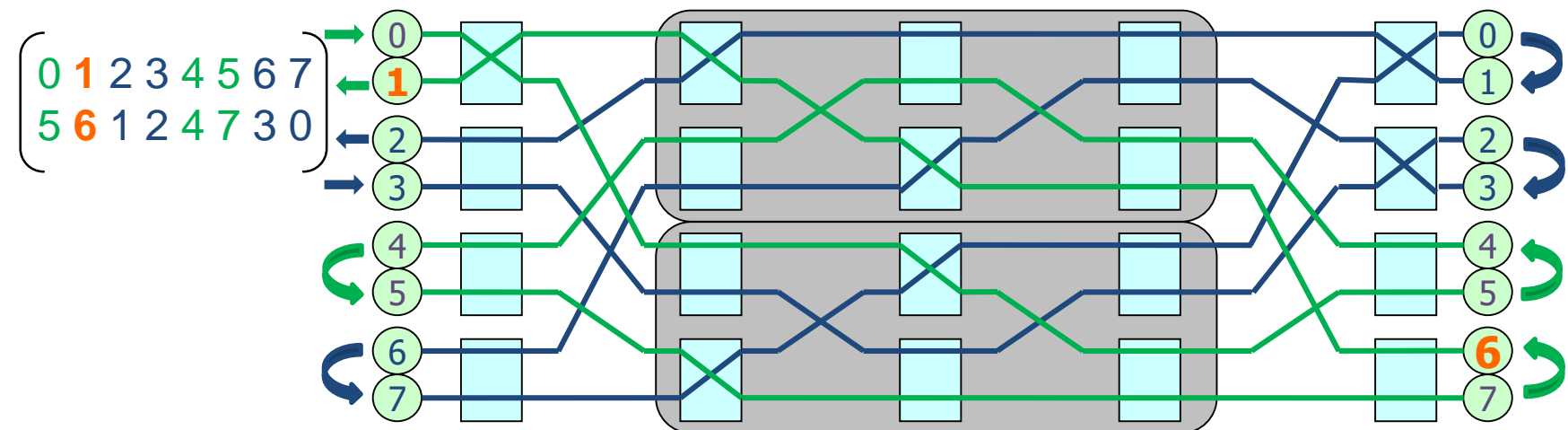
Looping algorithm

- ▶ Example on a Benes network of size $N=8$
 - ▶ The algorithm starts from an arbitrarily chosen
 - ▶ The input is connected to the requested output
 - ▶ The other output of the switch in the last stage is connected to the corresponding input
 - ▶ The algorithm follows this procedure, **looping back and forth** between inputs and outputs, until the original switch is reached
 - ▶ If there are inputs not connected, the algorithm starts again from a free input



Looping algorithm

- ▶ Example on a Benes network of size $N=8$
 - ▶ The algorithm starts from an arbitrarily chosen
 - ▶ The input is connected to the requested output
 - ▶ The other output of the switch in the last stage is connected to the corresponding input
 - ▶ The algorithm follows this procedure, **looping back and forth** between inputs and outputs, until the original switch is reached
 - ▶ If there are inputs not connected, the algorithm starts again from a free input



LOG N STAGE MIN EQUIVALENCE

T. Calamoneri, A. Massini - *Efficient Algorithms for Checking the Equivalence of Multistage Interconnection Networks*

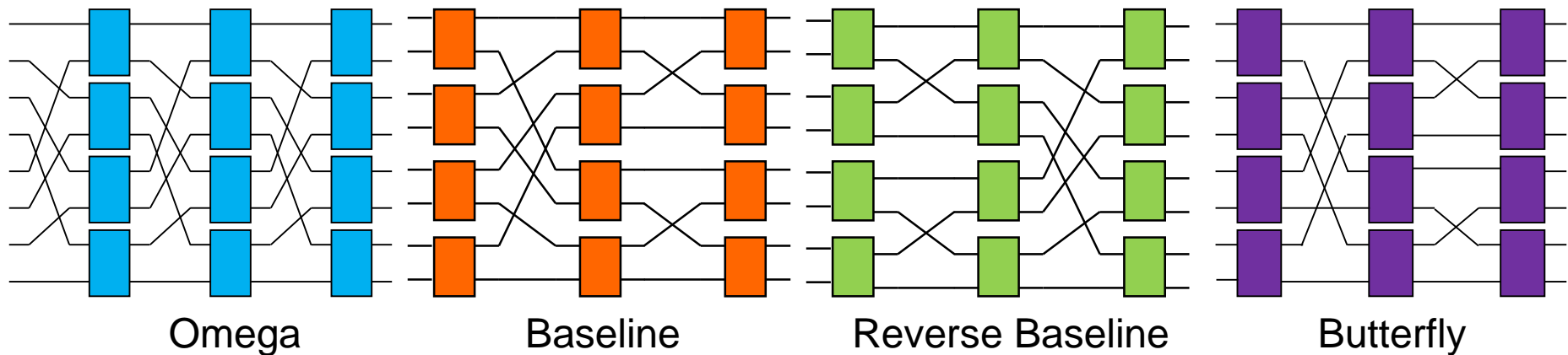
Journal of Parallel and Distributed Computing, 64, 135 - 150, 2004

Topological and functional equivalence

- There are two different concepts of equivalence:
 - **Topological equivalence**: isomorphism
 - **Functional equivalence**: capability of always performing the same set of assignments
- Topological equivalence and functional equivalence are different:
 - All **rearrangeable** MINs are **functionally equivalent** (because they can realize all the permutations) though not necessarily topologically equivalent
 - **Not rearrangeable** MINs **can be topologically equivalent** but not functionally equivalent, as in the case of $\log N$ stage MINs

Topological equivalence

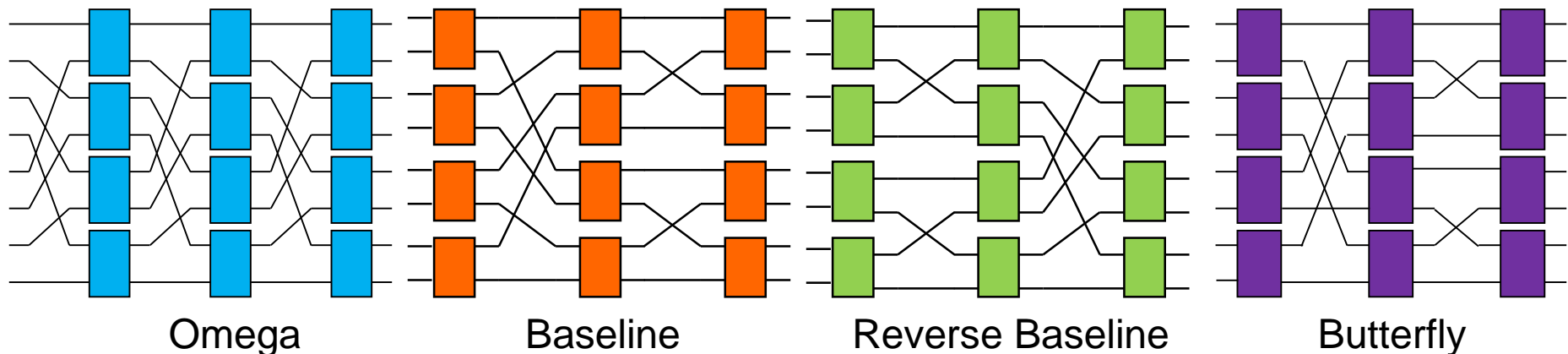
- Networks are **topologically equivalent** if one network can be easily reproduced from the other networks by simply rearranging nodes at each stage \rightarrow isomorphism



Topological equivalence

Bermond, Fourneau and Jean-Marie (1987) give the characterization of MINs topologically equivalent to the *Reverse Baseline* network. It is based on:

- the *Banyan property*
 - A MIN has the Banyan property if and only if for any input and any output there exists a unique path connecting them, passing through each stage once



Topological equivalence

Bermond, Fourneau and Jean-Marie (1982) give the characterization of MINs topologically equivalent to the Reverse Baseline network

It is based on:

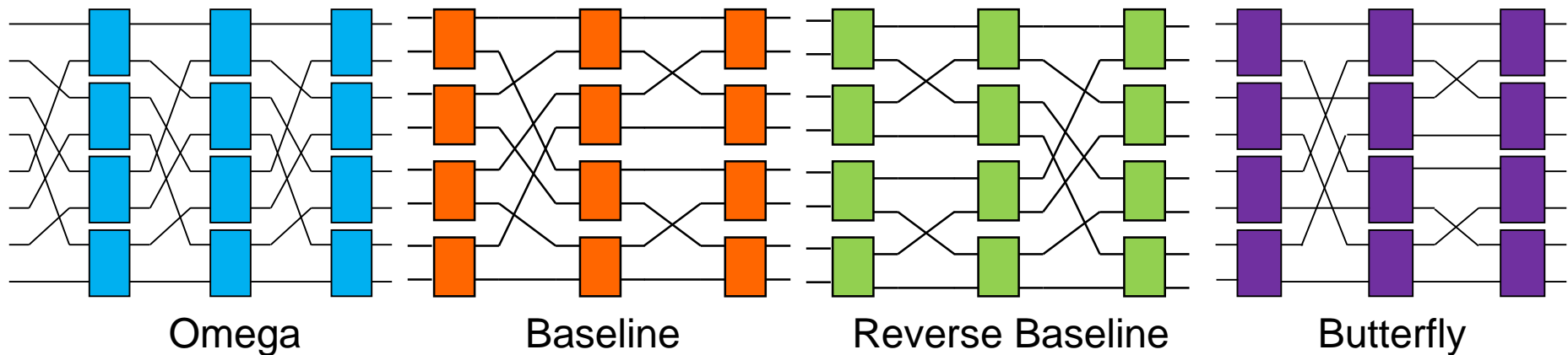
- the ***$P(*, *)$ property***

- ***Property $P(i, j)$*** An N-MIN has property $P(i, j)$ for $1 \leq i \leq j \leq \log N$ if the subgraph $G_{i,j}$ induced by the nodes of the stage from i to j has exactly $2^{\log N - 1 - j + i}$ connected components
- ***Property $P(*, *)$*** An N-MIN has property $P(*, *)$ if and only if it satisfies $P(i, j)$ for every ordered pair i, j such that $1 \leq i \leq j \leq \log N$

Topological equivalence

Bermond, Fourneau and Jean-Marie (1982) give the characterization of MINs topologically equivalent to the Reverse Baseline network

Theorem All the MINs satisfying the Banyan Property and $P(*, *)$ are *topologically equivalent* to the Reverse Baseline



Topological equivalence

- Another way to prove the equivalence of log N stage MINs Calamoneri and Massini (2004)
- It is based on the Layered Cross Product Even and Litman (1992)
 - An **l-layered graph**, $G = (V_1, V_2, \dots, V_l, E)$ consists of l layers of nodes, V_i is the set of nodes in layer i , where $1 \leq i \leq l$; E is a set of edges connecting nodes of two adjacent layers
 - The **Layered Cross Product**, $G = G' \otimes G''$, of two l -layered graphs $G' = (V'_1, V'_2, \dots, V'_l, E')$ and $G'' = (V''_1, V''_2, \dots, V''_l, E'')$ is an l -layered graph $G = (V_1, V_2, \dots, V_l, E)$ where V_i is the cartesian product of V'_i and V''_i , $1 \leq i \leq l$, and an edge $\langle (u', u''), (v', v'') \rangle$ belongs to E if and only if $\langle u', v' \rangle \in E'$ and $\langle u'', v'' \rangle \in E''$. G' and G'' are called the first and second factor of G , respectively

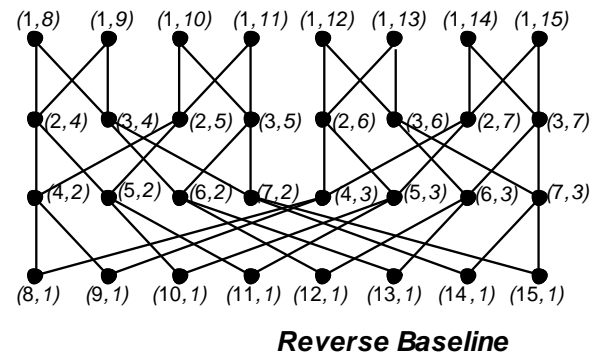
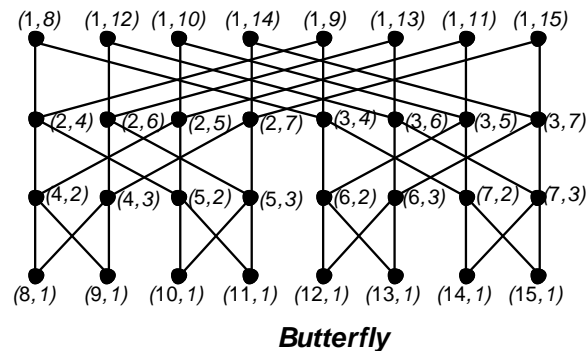
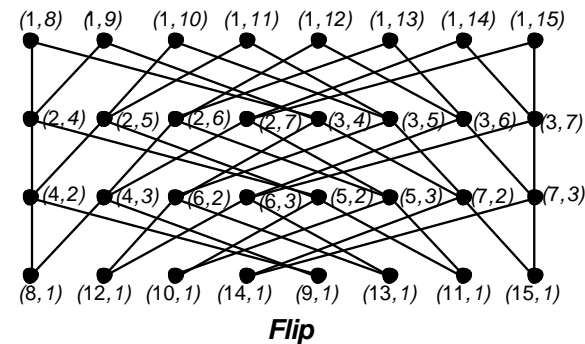
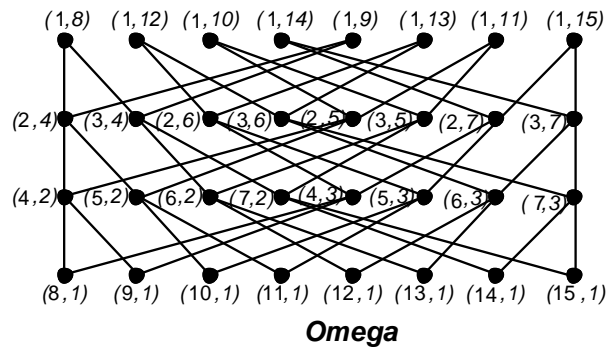
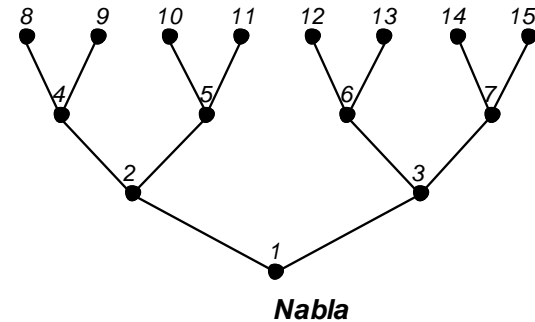
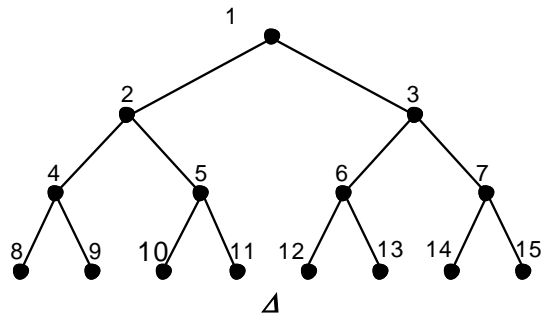
Topological equivalence

- The operation of *decomposition in factors* is the inverse operation of the LCP
- **Theorem** Let G' and G'' be two s stage MINs, and let G' decomposable as $G'_1 \otimes G'_2$. Then G'' is topologically equivalent to G' if and only if G'' can be decomposed as $G'_1 \otimes G'_2$
- **Corollary** Given two N-MINs $G' = G'_1 \otimes G'_2$ and $G'' = G''_1 \otimes G''_2$, they are topologically equivalent if their factors are topologically equivalent

Topological equivalence

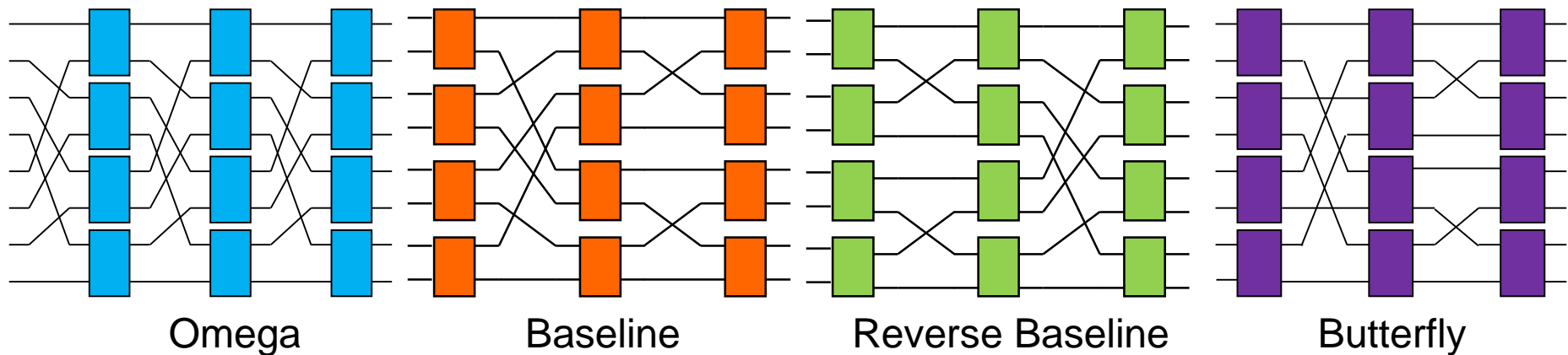
- **Lemma** A MIN G satisfies the Banyan and $P(*, *)$ properties if and only if it can be decomposed as $\Delta \otimes \nabla$, where Δ and ∇ denote binary trees with the root on the top and in the bottom, respectively
- **Theorem** A MIN G is decomposable as $\Delta \otimes \nabla$ if and only if G is topologically equivalent to the Reverse Baseline

Topological equivalence



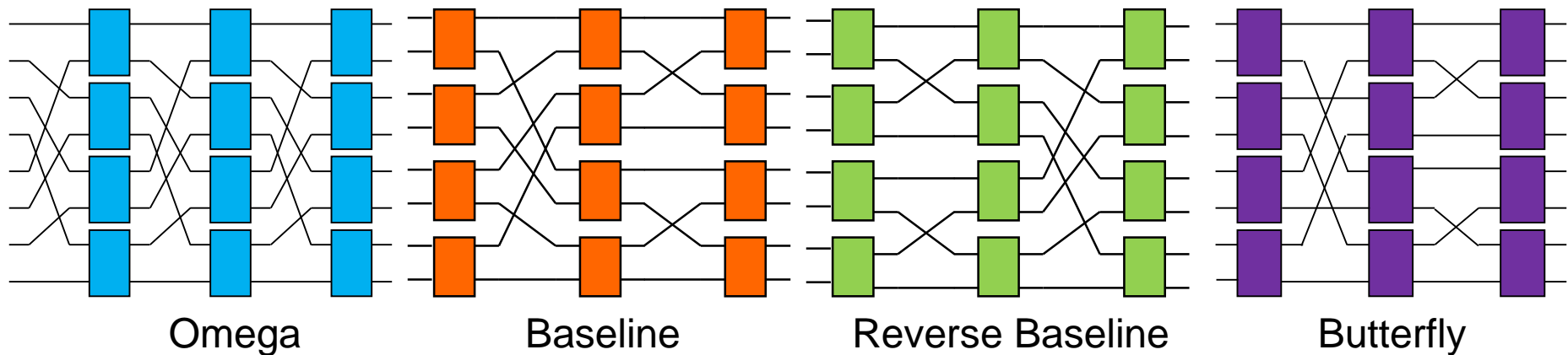
Topological equivalence

- MINs consisting of $\log N$ stages such as Omega, Flip (Reverse Omega), Baseline and Reverse Baseline, Butterfly and Reverse Butterfly are all equivalent networks
- They have attractive features, but **they are not rearrangeable**



Topological equivalence

- For this reason, MINs obtained by concatenating two $\log N$ stage MINs with the center stage overlapped, have been intensively studied
- Indeed, $2 \log N - 1$ is the theoretically minimum number of stages required for obtaining rearrangeable multistage interconnection networks



2 LOGN-1 STAGE MIN EQUIVALENCE

T. Calamoneri, A. Massini - *Efficient Algorithms for Checking the Equivalence of Multistage Interconnection Networks*

Journal of Parallel and Distributed Computing, 64, 135 - 150, 2004

2logN-1 stage MIN equivalence

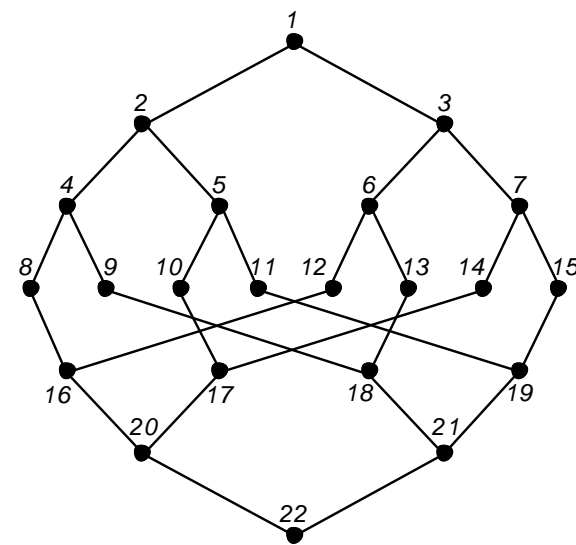
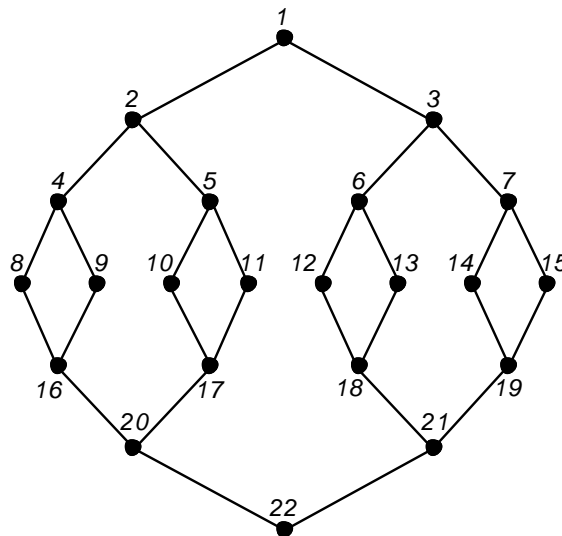
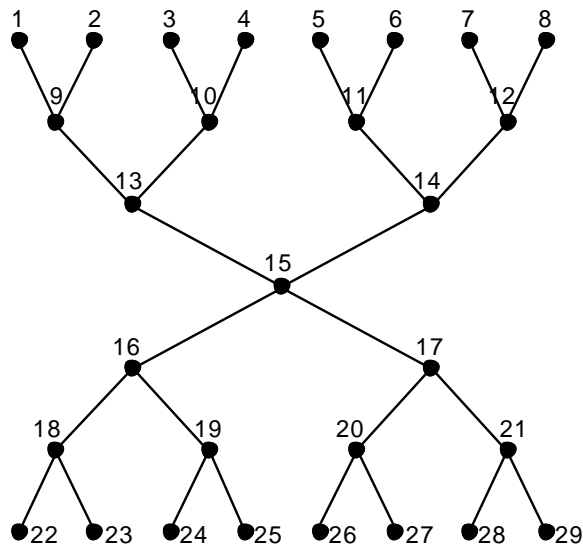
- The popular $(2 \log N - 1)$ stage Benes network is rearrangeable and the Looping algorithm provides a method and a proof for its rearrangeability
- **Unfortunately** the Looping algorithm can be used only on $(2 \log N - 1)$ stage **symmetric** MINs with **recursive structure** such as Baseline-Reverse Baseline and Butterfly-Reverse Butterfly networks
- Looping algorithm does not work on the Omega-Omega⁻¹ or Double Baseline even if they are equivalent to the Benes network

2logN-1 stage MIN equivalence

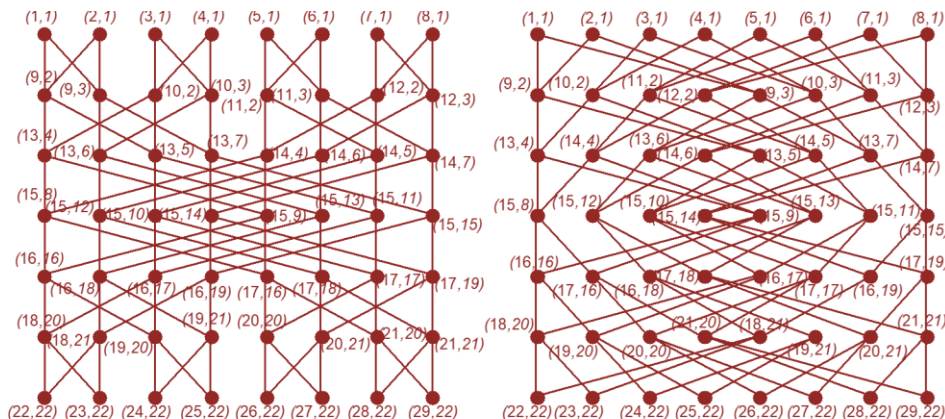
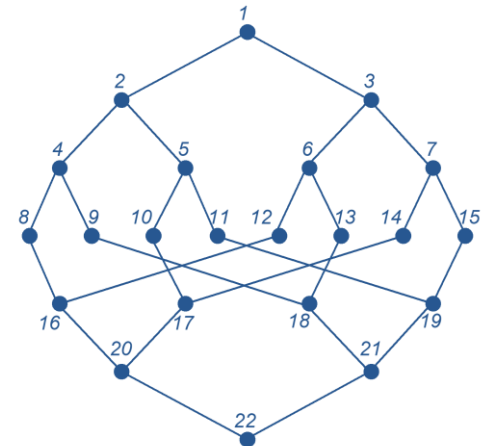
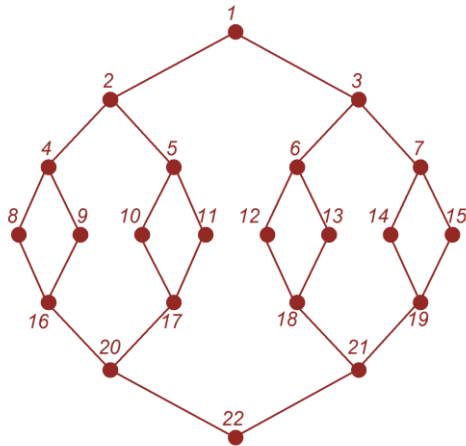
- It is typical to concatenate all the combinations of pairs of networks among Butterfly, Omega, Flip, Baseline, their reverses, etc. to obtain a new N-MIN
- Both the two log N stage MINs constituting a (2log N- 1) stage MIN can be decomposed as LCP of $\Delta \otimes \nabla$
- As a consequence, we obtain that the factors of (2log N- 1) stage MIN are the concatenation of a Δ and a ∇ (roots merging) and of a ∇ and a Δ (leaves merging), r

2logN-1 stage MIN equivalence

- It is **obvious** how to merge the last layer of a ∇ with the first layer of a Δ
- But there are **many ways** of merging the last layer of a Δ and the first layer of a ∇ respectively

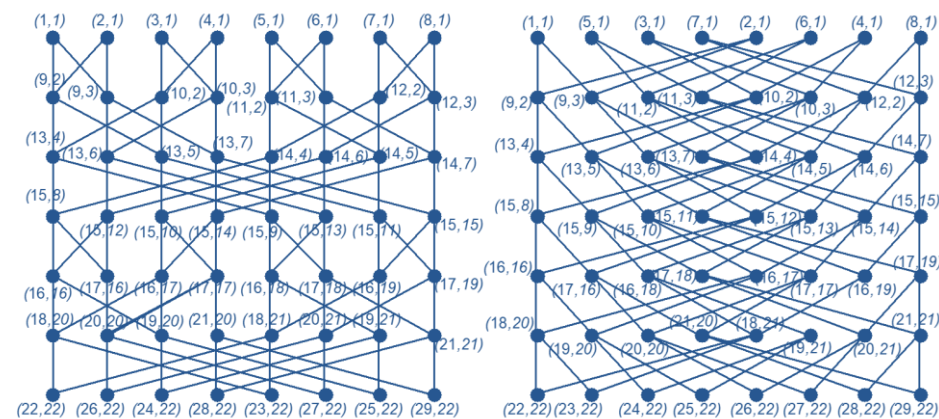


2logN-1 stage MIN equivalence



A reverse Butterfly and a Butterfly

A Flip and a Omega



Two reverse Butterflies

Two Omega

$2\log N - 1$ stage MIN equivalence

- **Theorem** The number of **distinct equivalence classes** of $(2 \log N - 1)$ MINs is **$(\log N - 1)!$**
- We can represent these classes representing the MINs using **butterfly stages**
- In particular we can represent the first half of the MIN as a butterfly and the second half by a permutation of butterfly stages (that are: $\log N - 1$)

Classes for N=16

