ADVANCED ARCHITECTURES

Interconnection Networks

Annalisa Massini *Lecture 17-18*

2024-2025

References

- *Computer Architecture: A Quantitative Approach* 5th Edition

  J.L. Hennessy, D.A. Patterson - Morgan Kaufmann, 2011

  - Appendix F *Interconnection Networks Ch. F.4 – T. M. Pinkston and J. Duato*

- *Advanced Computer Architecture and Parallel Processing*

  H. El-Rewini, M. Abd-El-Barr - John Wiley and Sons, 2005

- *Parallel computing for real-rime signal processing and control*

  M. O. Tokhi, M. A. Hossain, M. H. Shaheed - Springer, 2003

  – Ch. 2 *Parallel Architectures*

INTRODUCTION

Advanced Computer Architecture and Parallel Processing

H. El-Rewini, M. Abd-El-Barr - John Wiley and Sons, 2005

Section 4.4 – Graphics

Computer Architecture - A Quantitative Approach, Fifth Edition

J.L. Hennessy D.A. Patterson

Appendix F – Interconnection Networks

Classification based on type of interconnections

- Flynn’s classification of computer architectures is general

- In particular, **MIMD class** comprises very different **architectures**, generally consisting of multiple PEs and memory modules
- Architectures in the **MIMD class** can be described using a classification based on interconnection network
- The various **interconnecting communication networks** used in parallel architectures include: **linear, shared single bus, shared multiple bus, crossbar, ring, mesh, star, tree, hypercube and complete graph**

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 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

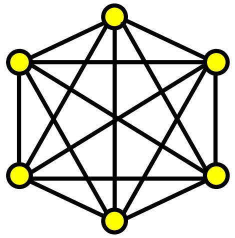
Switching techniques

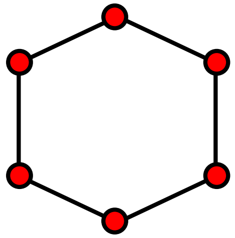
- Interconnection networks can be classified according to the switching mechanism as ***circuit switching*** versus ***packet switching networks***

- 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
  - Packet switching tends to use the network resources more efficiently, but suffers from variable packet delays

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
  - A ***ring*** topology is a mapping that connects processor k to its neighbors, processors (k - 1) and (k + 1)


fully connected


ring

Topology

According to the *topology*, INs can be classified as ***static*** versus ***dynamic*** networks

- In **static networks**
  - Direct fixed links are established among nodes to form a fixed network
  - All connections are formed when the system is designed
- 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
- Nearly all multiprocessor systems can be distinguished by their interconnection network topology

Topology

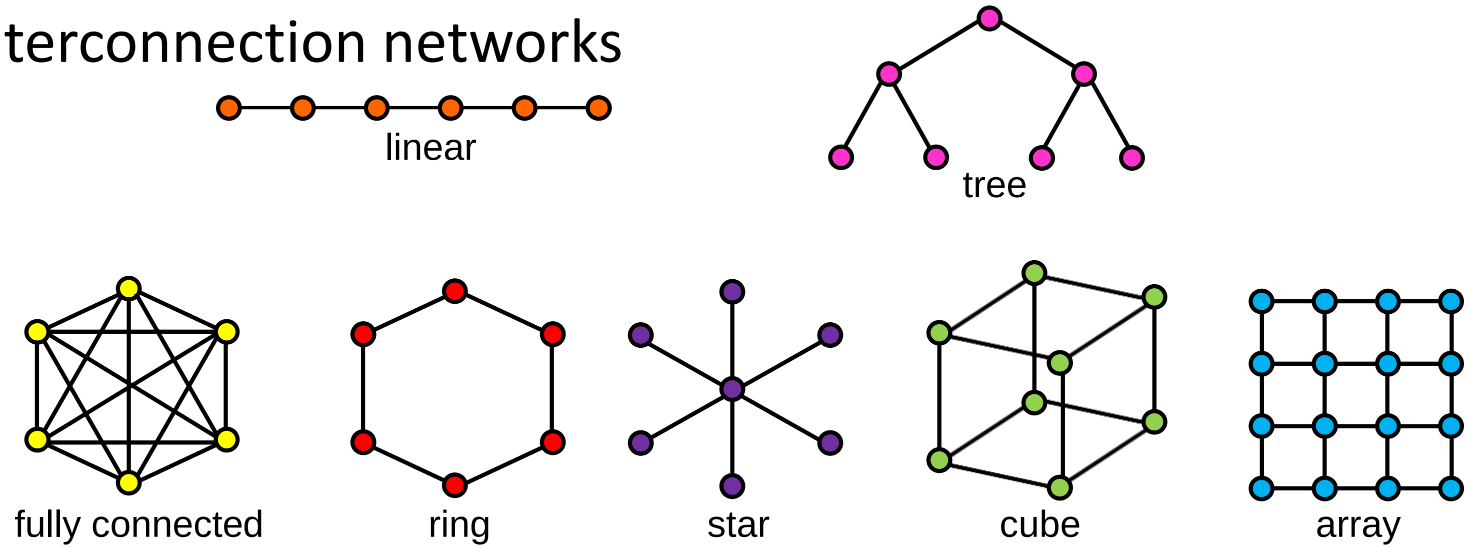
- The number of **hops** in a path from source to destination node is equal to the number of point-to-point links a message must traverse to reach its destination

- In either static or dynamic networks, a single message may have to hop through intermediate processors on its way to its destination

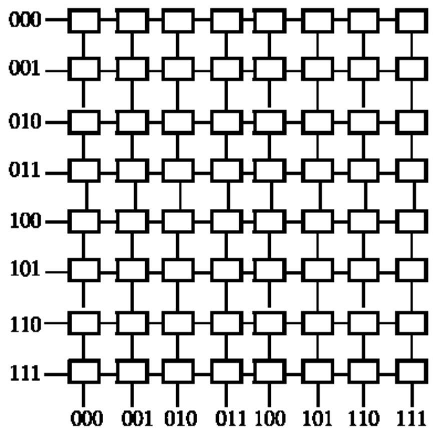
- Therefore, the ultimate **performance** of an interconnection network is greatly influenced by the **number of hops** taken to traverse the network

Static and dynamic networks

- **Static** interconnection networks



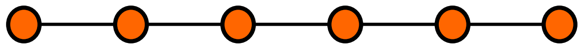
- **Dynamic** interconnection networks are **reconfigurable** under system control and include: *shared single bus, shared multiple bus, multistage interconnection networks and crossbar*



Static Networks

Linear Network

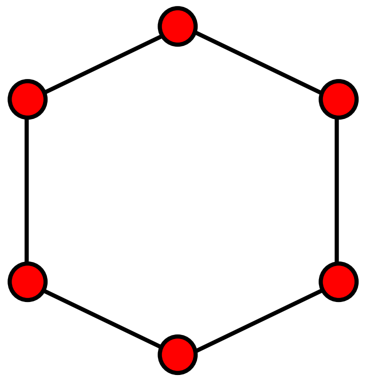
- Every node, except the nodes at the two ends, in this configuration is directly connected to two other nodes
- To connect *n nodes in this configuration n – 1 buses are required and the maximum* internodes distance is *n – 1*


line

Static Networks

Ring Interconnection Network

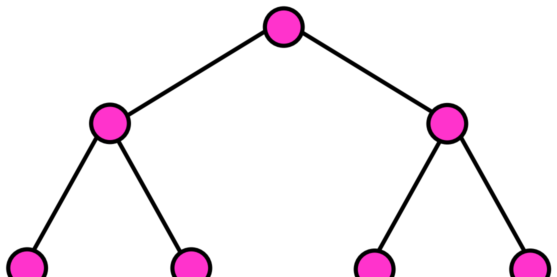
- *n* buses are required to connect *n* nodes
- *The maximum* internodes distance is *n / 2*
- Several commercial machines have been designed using ring networks (e.g. *Hewlett-Packard's Exemplar V2600* and *Kendal Square Research's KSR-2*)


ring

Static Networks

Tree Interconnection Network

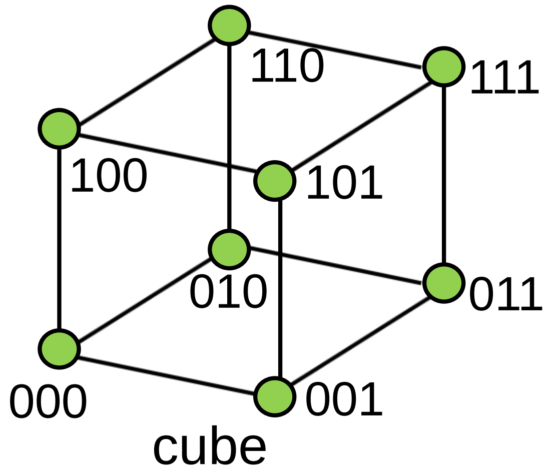
- In the tree structure any intermediate node acts as a medium to establish communication between its parents and children
- *The maximum* internodes distance – in case of a binary tree – is $2\log\left(\frac{n+1}{2}\right)$ where n is the number of nodes of the tree
- The root node can be the bottleneck


tree

Static Networks

Hypercube Interconnection Network

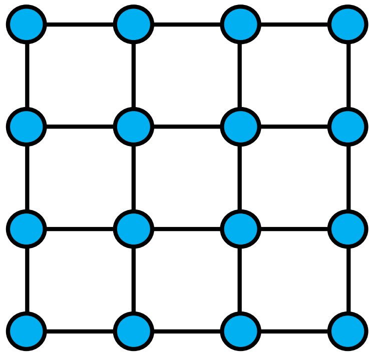
- An *n-dimensional hypercube can connect 2^n nodes*
- The **nodes** are labelled using **binary addresses** and addresses of the two neighboring nodes differ by one bit
- *The maximum* internodes distance is *n*
- Many commercial multiprocessors (especially NUMA multiprocessors) have used hypercube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

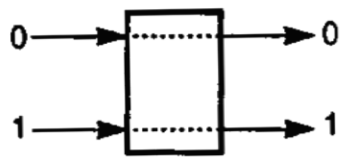

array or mesh

Dynamic Networks

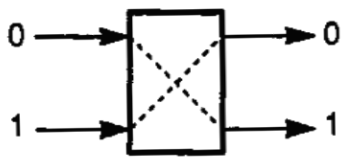
- Connections in a dynamic network are established *on the fly as needed*
- Dynamic networks can be classified based on interconnection scheme as **bus-based** or **switch-based**
- **Bus-based** networks can further be classified as single bus or multiple buses
- **Switch-based** can be classified according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

Dynamic Networks

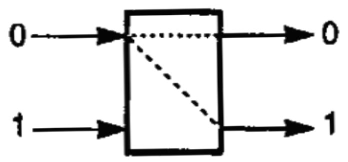
- **Switch-based** networks include switches with a given number of input and outputs, providing different functions
- **Example**: possible functions for a 2 × 2 switch



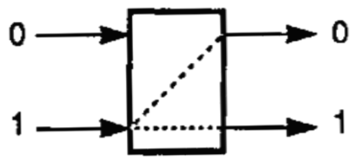
(a) Straight



(b) Crossover



(c) Upper broadcast



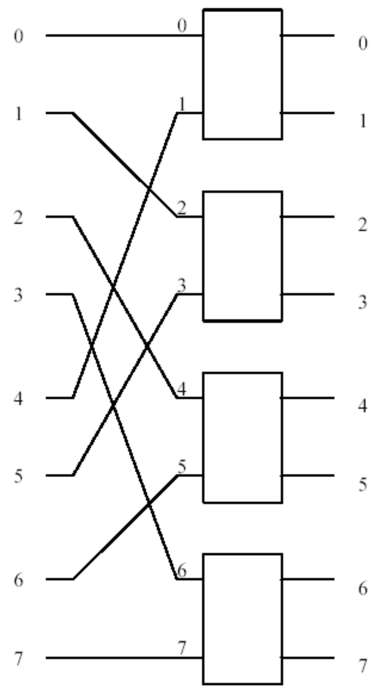
(d) Lower broadcast

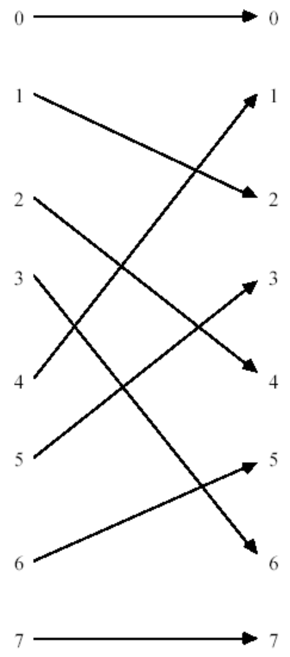
Single-stage networks

- An example of single stage network is the **Shuffle-Exchange** IN (***left***)

- *Perfect shuffle operation*: cyclic shift 1 place left, e.g., 101 --> 011
- The perfect ***shuffle*** mapping function is realized by the links (***right***)

- *Exchange operation*: invert least significant bit, e.g., 101 --> 100
- Exchange is realized by the switches



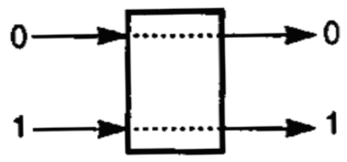


Multistage Interconnection Networks

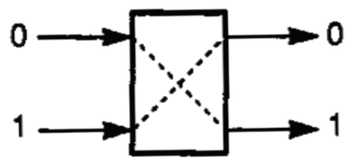
- The capability of single stage networks is limited

- If we **cascade** enough stages together, they form a **Multistage Interconnection Network** (MIN)

- Switches can perform their own routing or can be controlled by a central router
- Most used functions that switches can assume are ***straight*** and ***cross*** state



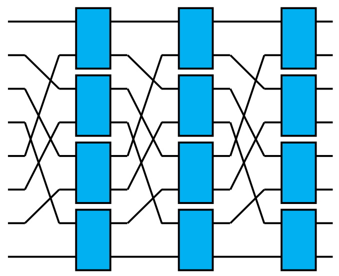
(a) Straight

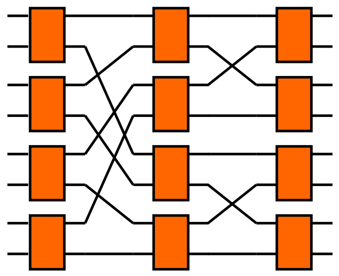


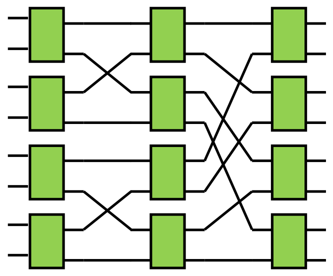
(b) Crossover

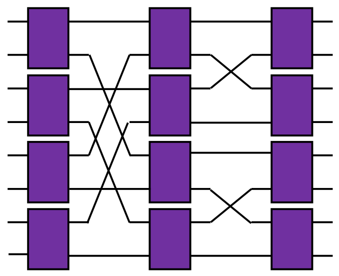
Multistage Interconnection Networks

- ***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

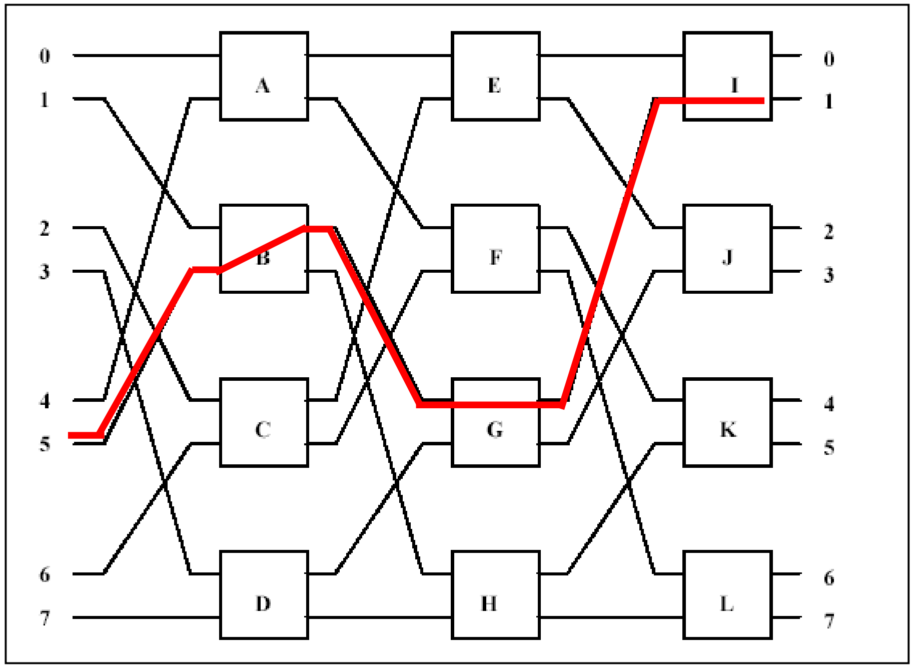

Baseline


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 → no fault tolerance, blocking

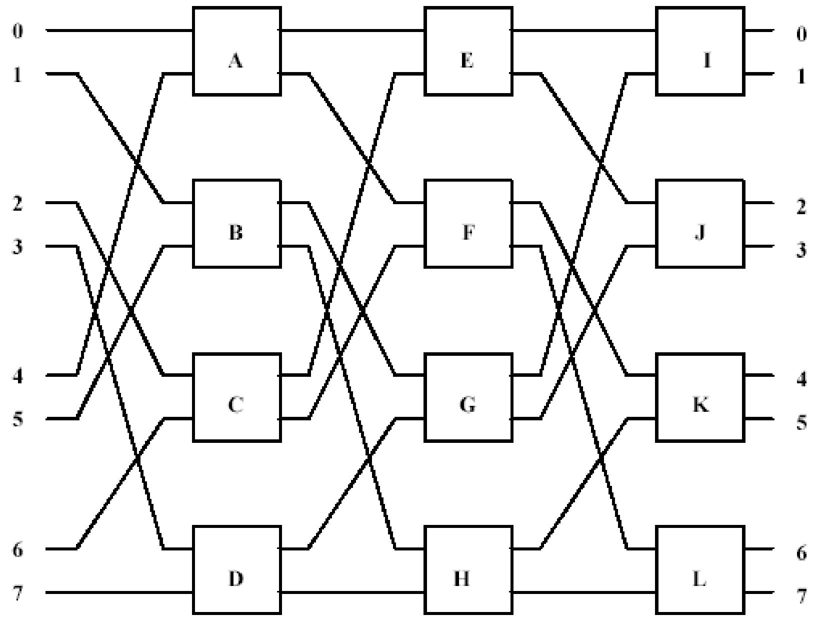


Example

- 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

Omega networks

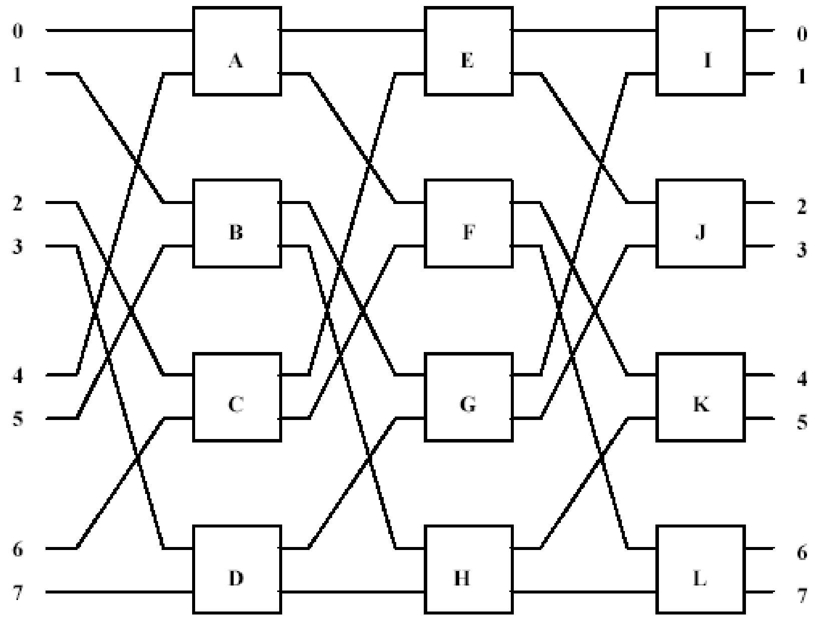
- $\log_2 N$ **stages** of 2 × 2 **switches**
- N/2 **switches per stage**
- $S = (N/2) \log_2(N)$ total number of switches
- Number of **permutations** realized by an Omega network 2^S (much less than N! = total number of permutations)



Omega networks

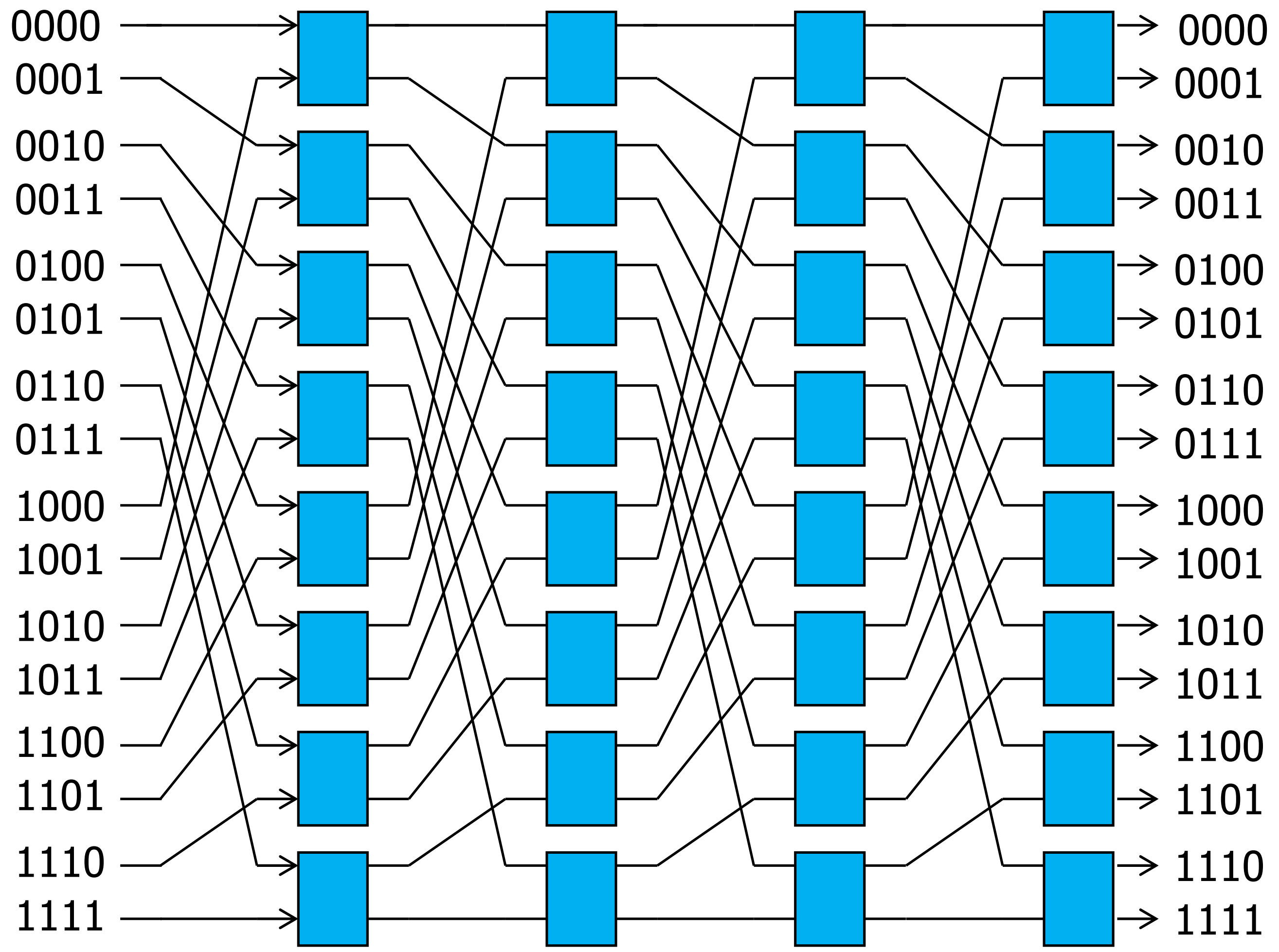
- $\log_2 N$ **stages of 2 × 2 switches**
- N/2 **switches per stage**
- $S = (N/2) \log_2(N)$ total number of switches
- Number of **permutations** by an Omega network 2^S

This is valid for all multistage interconnection networks with $\log_2 N$ stages and 2x2 switches



Omega networks

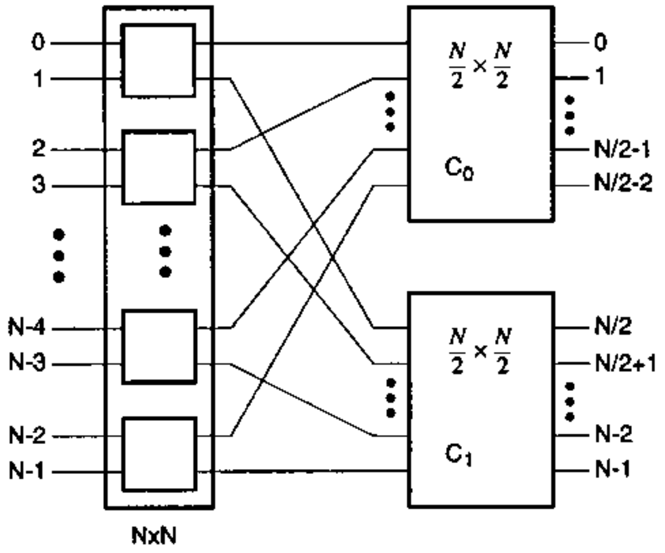
- ***Multistage interconnection networks (MINs)***

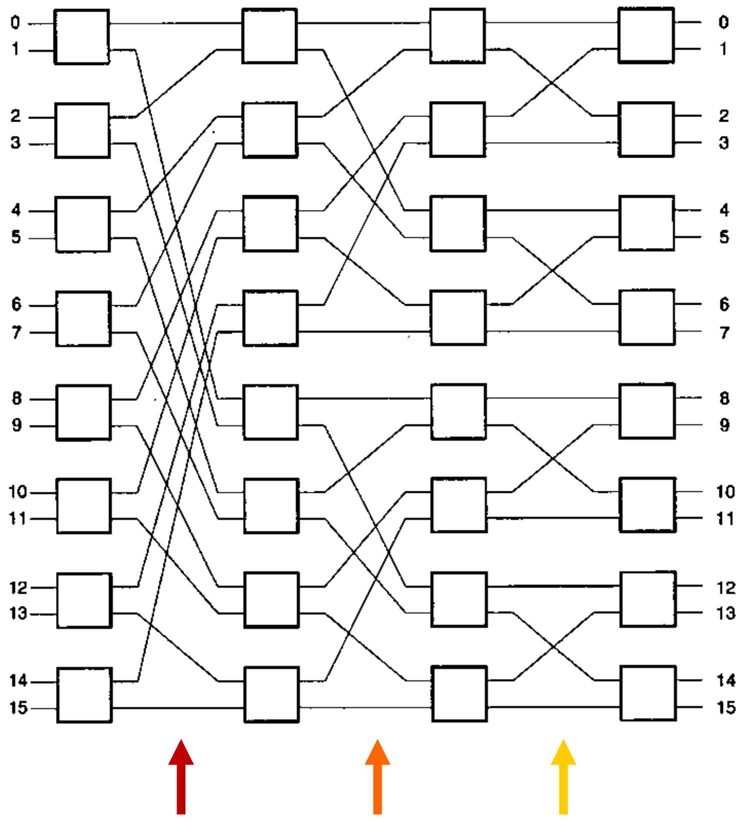


4 stage
Omega
network

Baseline networks

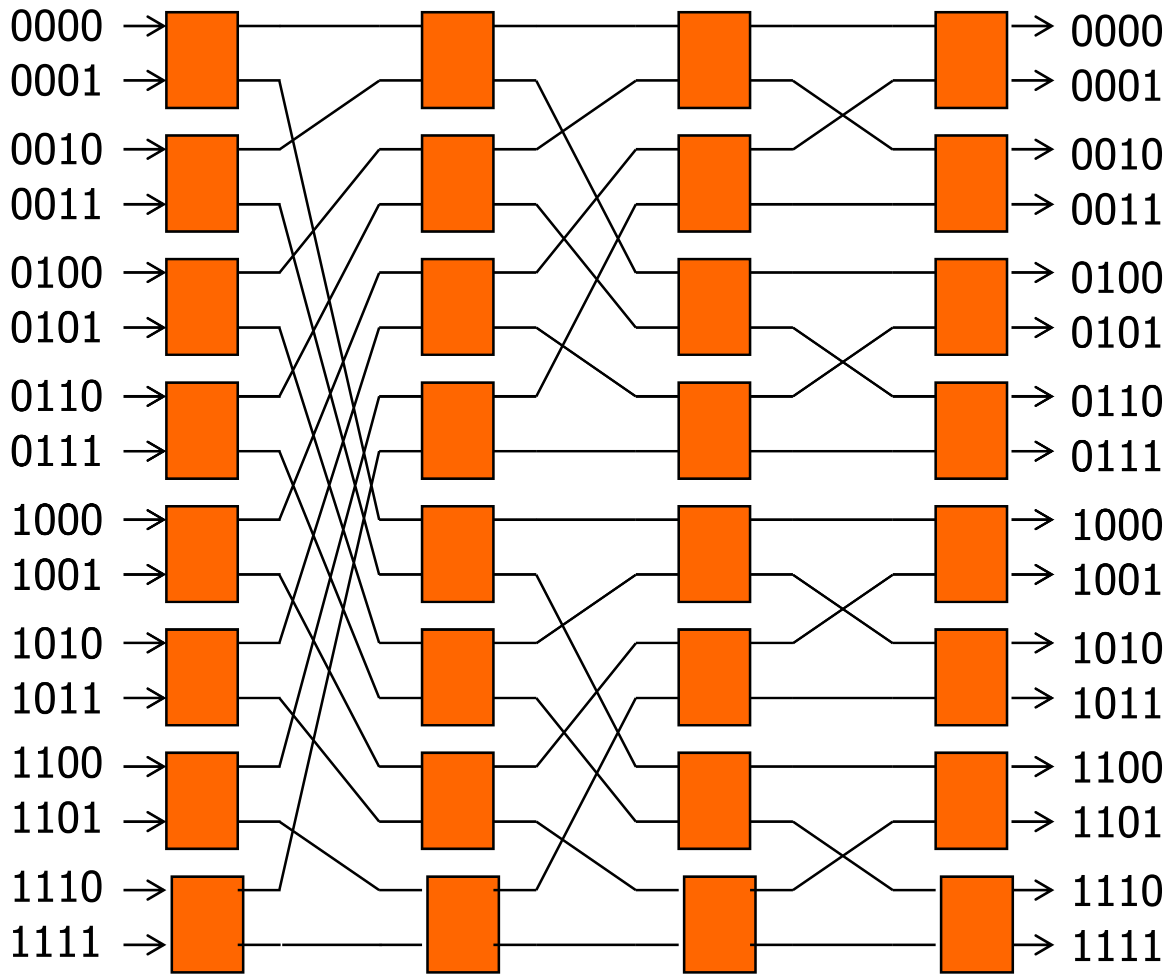
- The baseline network can be generated recursively
- The **first stage** N × N, the **second** (N/2) × (N/2) twice, the **third**…
- Stages are *shuffle*





Baseline networks

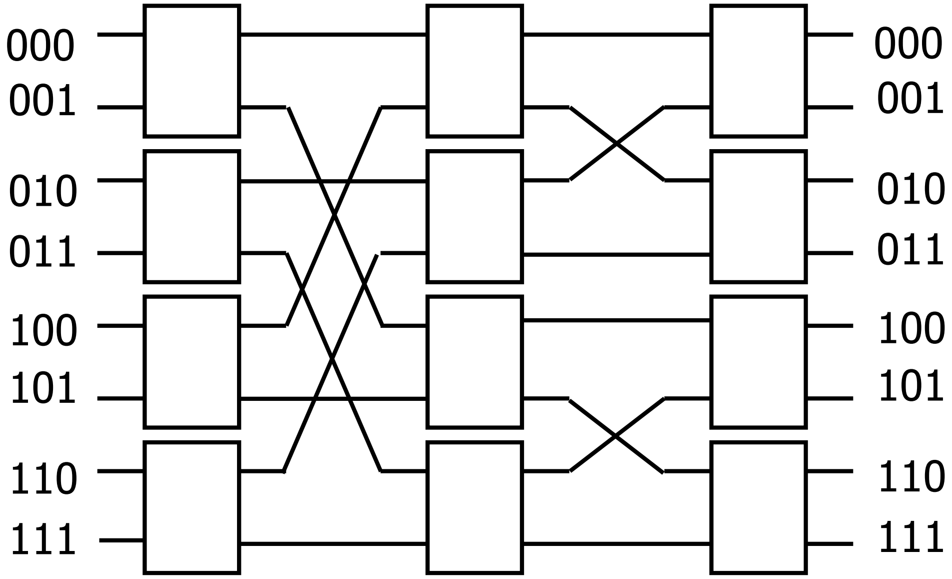
- ***Multistage interconnection networks (MINs)***



4 stage Baseline network

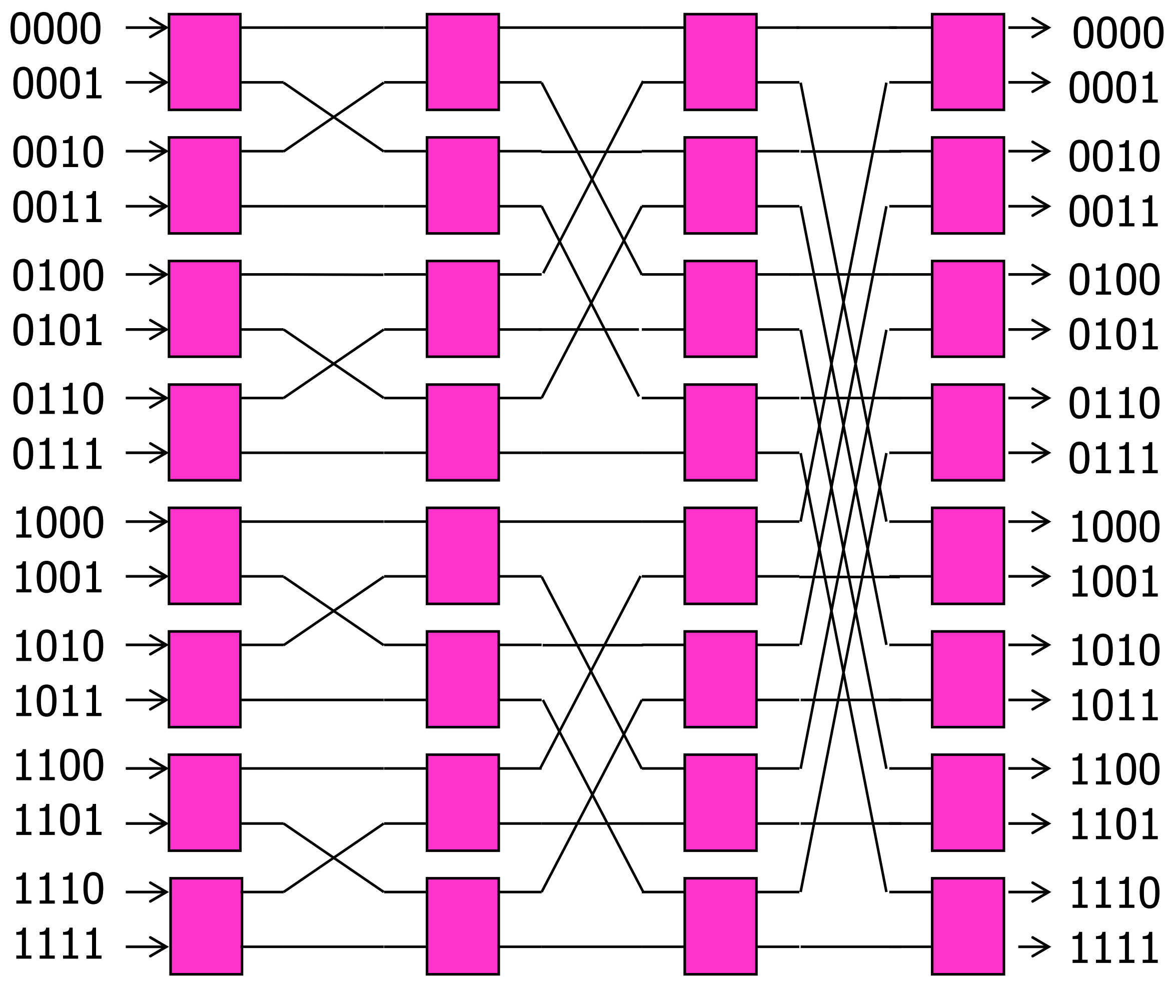
Butterfly networks

- The Butterfly network can be generated using *Butterfly stages*
- A **Butterfly edge stage** of size l, with l = 1, 2, …, n-1, is an edge stage where any node v in the left node stage is connected to node v' in the right node stage if and only if
  - $v = v'$ → **straight edge**
  - node v and v' differ only in the l-th bit → **cross edge**



Butterfly network

- *Multistage interconnection networks (MINs)*

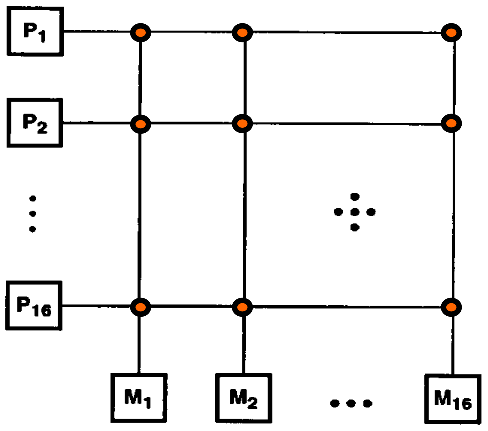


4 stage Reverse Butterfly network

Crossbar Network

- The crossbar contains a switching element (SE) at the intersection of any two horizontal or vertically lines
- Each junction is a **switching component**
- In general for an N x N crossbar
  - The network complexity, measured in terms of the number of switching points, is O(N^2)
  - The time complexity, measured in terms of the input to output delay, is O(1)

- The crossbar is a nonblocking network that allows multiple input–output connection patterns (permutations)
- For large multiprocessor system the network complexity becomes prohibitive



Crossbar Network

In summary

- 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×b* switches used in MIN’s, so that each crossbar is small, and costs are kept down
- Crossbar is a **nonblocking** network (it is *blocking* only if the destination is already 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

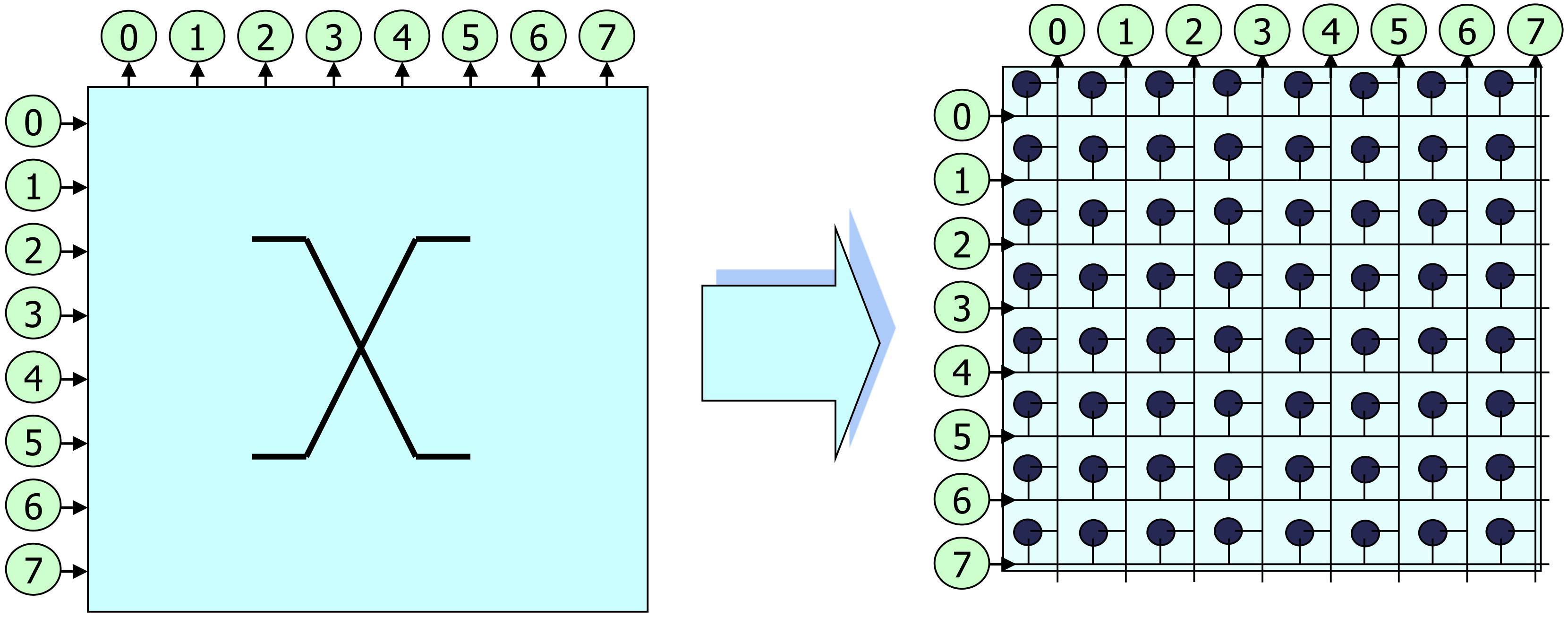
Comparison of Interconnection Networks

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

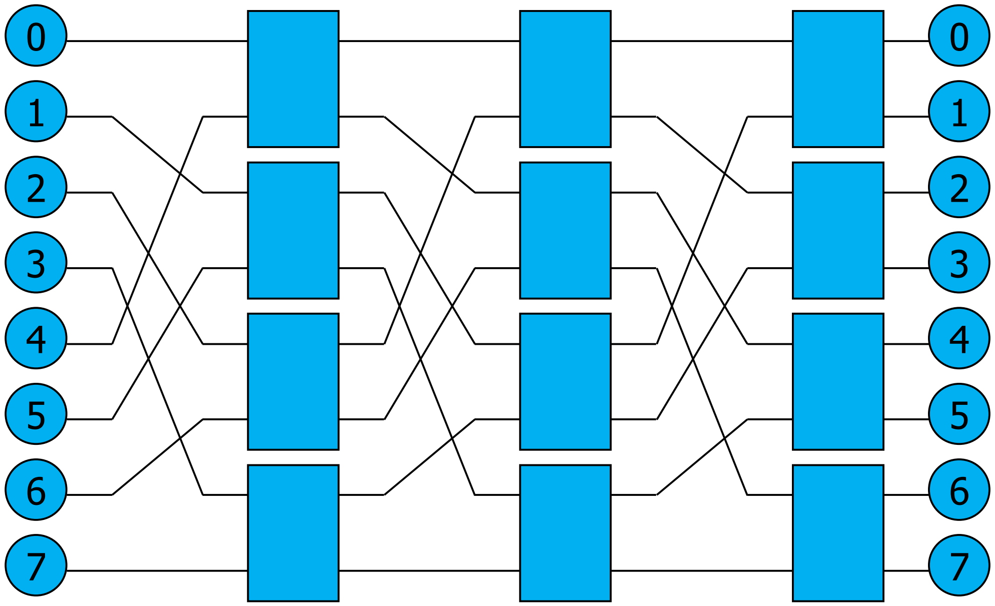
Comparison of Interconnection Networks

- ***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*



Comparison of Interconnection Networks

- ***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 input/output nodes
  - Inter-stage connections represented by a set of permutation functions



Omega topology, perfect-shuffle exchange

Comparison of Interconnection Networks

- Multistage interconnection networks (MINs)
  - MINs interconnect ***N* input/output ports** using **$k \times k$ switches**
    - $\log_k N$ switch stages, each with $\frac{N}{k}$ switches
    - $\frac{N}{k}\log_k N$ total number of switches

Example

- Compute the **switch** and **link** costs of interconnecting 4096 nodes using a crossbar relative to a MIN, that is **cost(crossbar)/cost(MIN)**
- Assume that switch cost grows quadratically with the number of input/output ports (*k*).

  Consider the following values of *k*:
  - *k*= 2 - MIN with 2 x 2 switches
  - *k*= 4 - MIN with 4 x 4 switches
  - *k*=16 - MIN with 16 x 16 switches

Comparison of Interconnection Networks

- **Crossbar**

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

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

Comparison of Interconnection Networks

- **Crossbar**

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

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

- **Relative cost of MINs**

  $\text{relative\_cost}(2 \times 2)_{\textit{switches}} = 4096^2 / (2^2 \times 4096/2 \times \log_2 4096) = 512/3 = \mathit{170}$

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

Comparison of Interconnection Networks

- **Crossbar**

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

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

- **Relative cost of MINs**

  $\text{relative\_cost}(2 \times 2)_{\textit{switches}} = 4096^2 / (2^2 \times 4096/2 \times \log_2 4096) = 512/3 = \mathit{170}$

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

  $\text{relative\_cost}(4 \times 4)_{\textit{switches}} = 4096^2 / (4^2 \times 4096/4 \times \log_4 4096)) = 512/3 = \mathit{170}$

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

Comparison of Interconnection Networks

- **Crossbar**

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

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

- **Relative cost of MINs**

  $\text{relative\_cost}(2 \times 2)_{\textit{switches}} = 4096^2 / (2^2 \times 4096/2 \times \log_2 4096) = 512/3 = \mathit{170}$

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

  $\text{relative\_cost}(4 \times 4)_{\textit{switches}} = 4096^2 / (4^2 \times 4096/4 \times \log_4 4096)) = 512/3 = \mathit{170}$

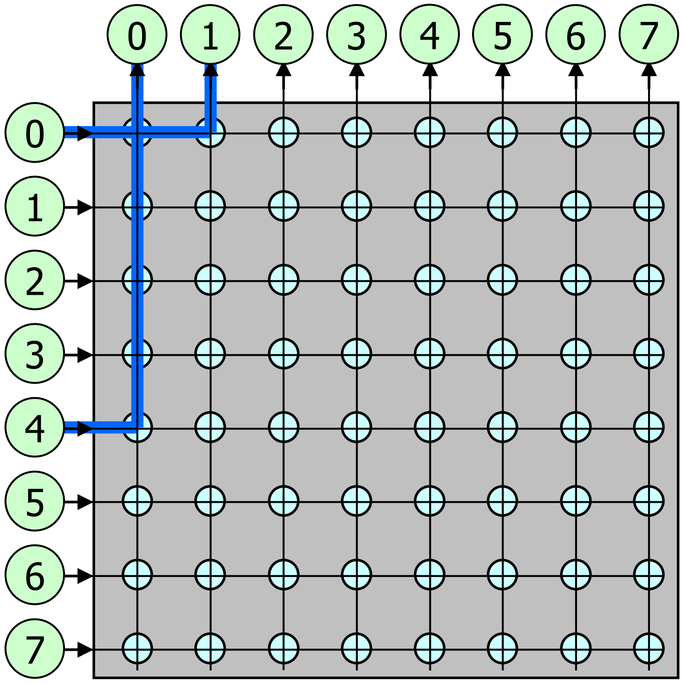
  $\text{relative\_cost}(4 \times 4)_{\textit{links}} = 8192 / (4096 \times (\log_4 4096 + 1)) = 2/7 = \mathit{0.2857}$

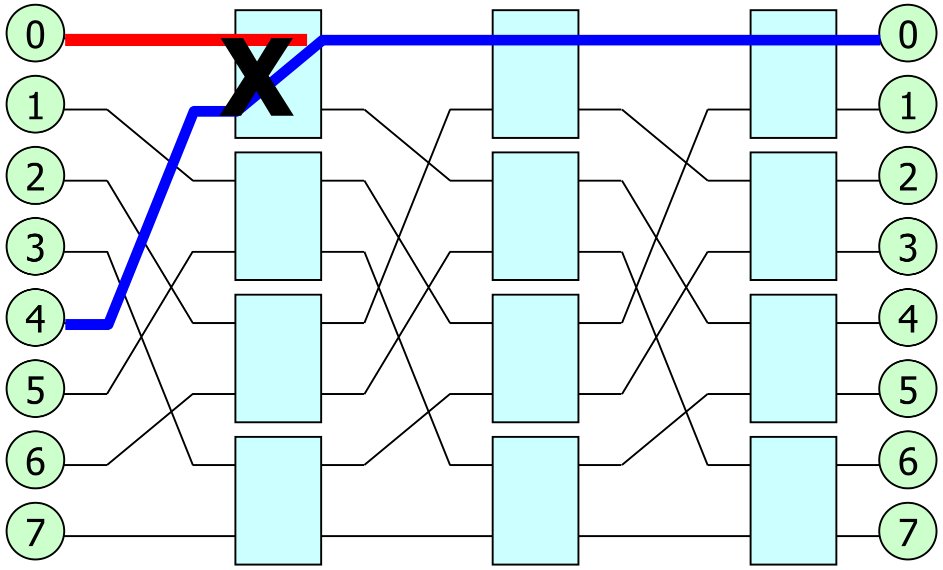
  $\text{relative\_cost}(16 \times 16)_{\textit{switches}} = 4096^2 / (16^2 \times 4096/16 \times \log_{16} 4096) = \mathit{85}$

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

Comparison of Interconnection Networks

- ***Cost** reduction* in MIN switch → ***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 two requests 0→1 and 4→0





Comparison of Interconnection Networks

- 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

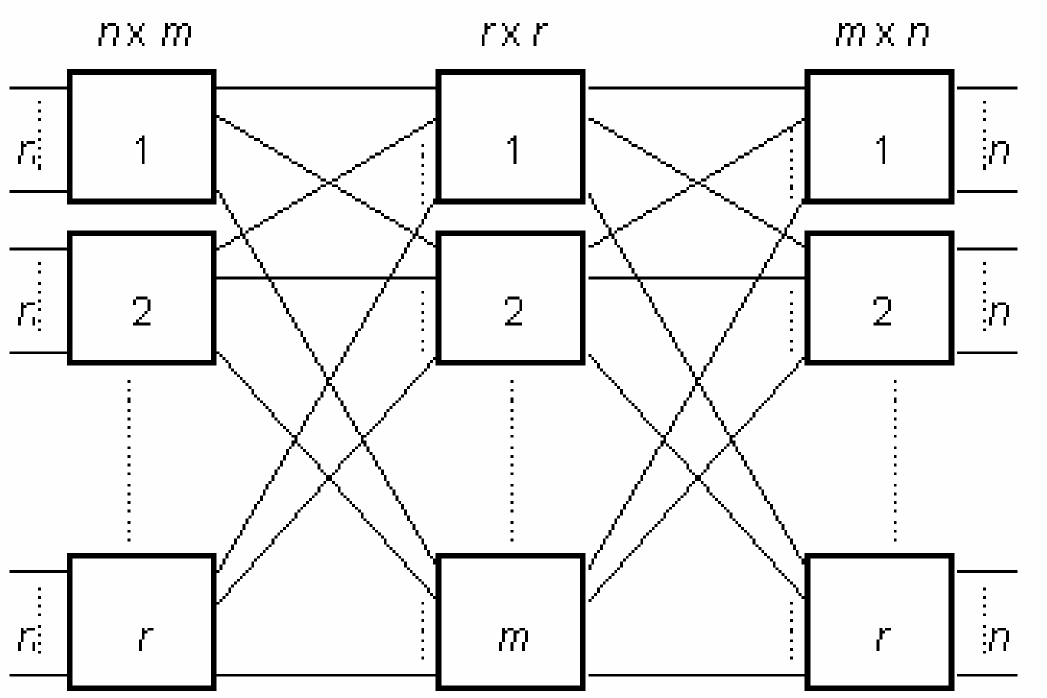
Comparison of Interconnection Networks

- 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 x 2

CLOS NETWORK

Clos network

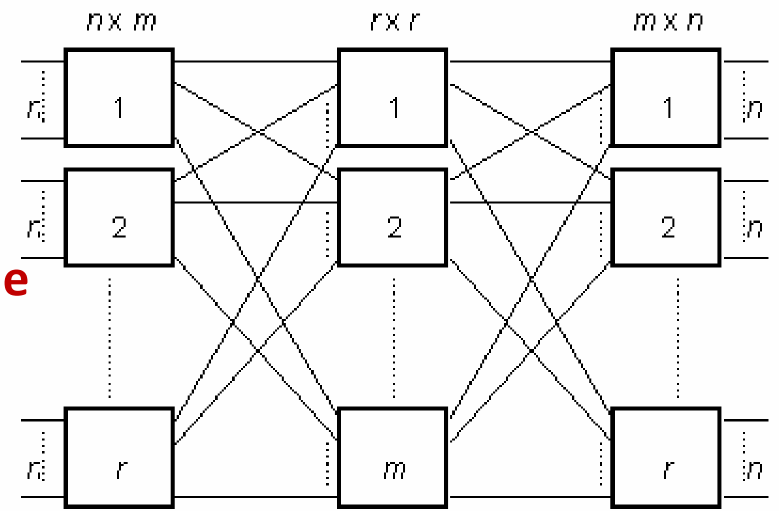
- Clos network is a ***multistage switching network***
- Clos networks have **three stages** - the *ingress stage*, *middle stage*, and the *egress stage* - made up of **crossbars**



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

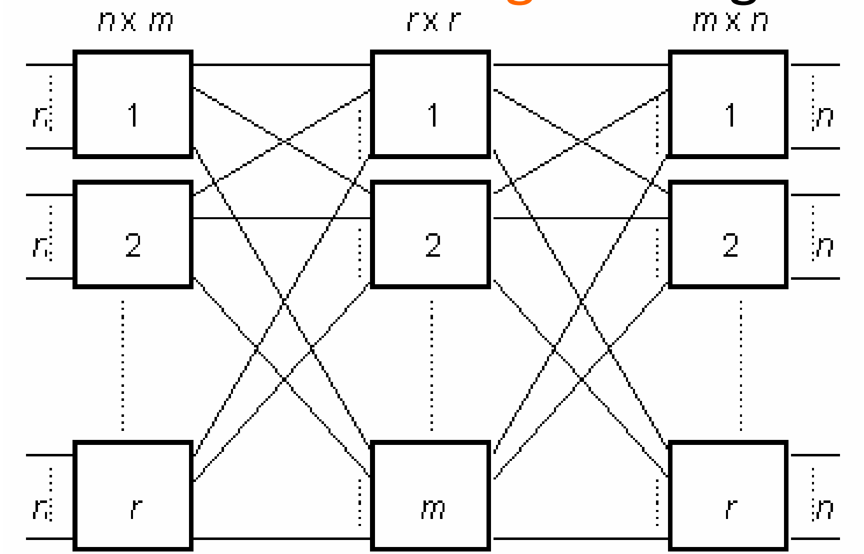


Clos network

Thus:

- The ingress stage has ***r*** switches *n* x *m*
- The middle stage has ***m*** switches - *r* x *r*
- The egress stage has ***r*** switches - *m* x *n*

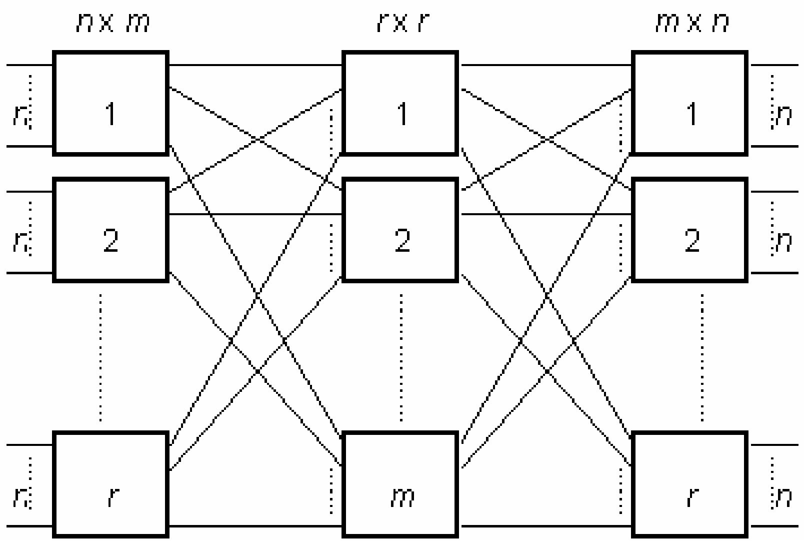
- Each middle stage switch is connected exactly once to each ingress stage switch and to each egress stage switch



Clos network

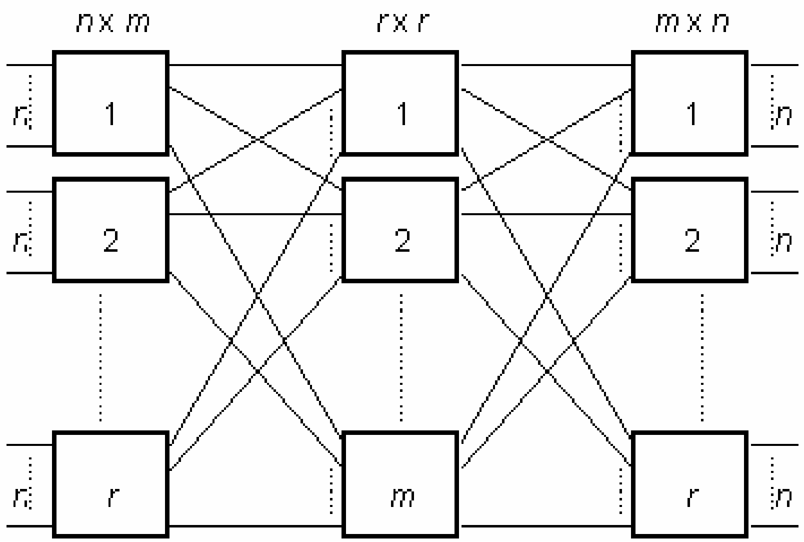
- Each request entering an **ingress crossbar** can be routed through ***any*** of the available **middle stage crossbar** to the relevant **egress crossbar** switch

- A middle stage crossbar is **available** for a new request 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 (out of n) on the input of an ingress switch
  - this has to be connected to a free terminal on a particular egress switch
- In the worst case:
  - $n-1$ other requests are active on the ingress switch in question
  - $n-1$ other requests are active on the egress switch in question
- Assume, also in the worst case, that:
  - each of these requests 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 utilized, that is $r \times n$ requests 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 $0 \le k \le r$) 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*×*n* calls and these cannot be carried by less than *k* egress switches

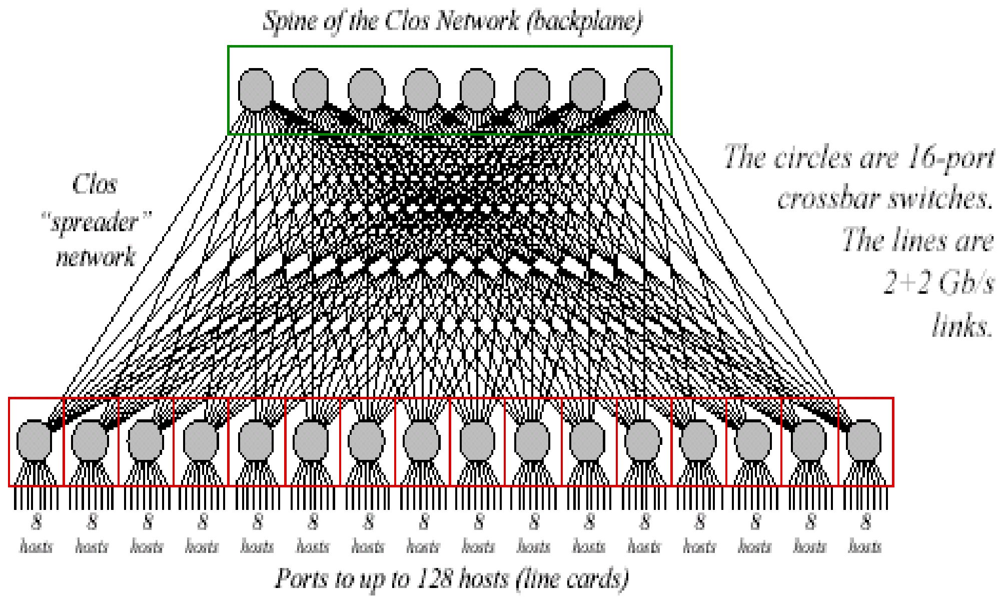
Rearrangeably nonblocking Clos networks

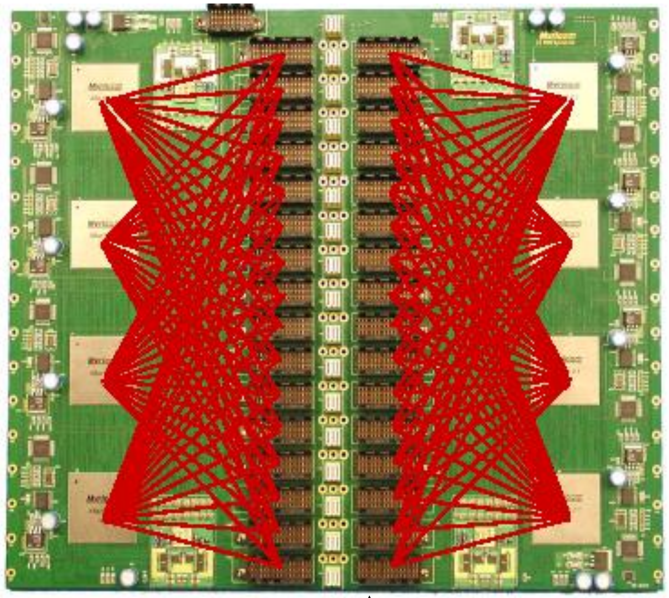
- Hence each ingress switch can be paired off with an egress switch that carries the same call, via a one-to-one mapping
- These r calls can be carried by one **middle-stage** switch
- If this middle-stage switch is now removed from the Clos network, m is reduced by 1, and we are left with a smaller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


Spine of the Clos Network (backplane)
Clos "spreader" network
The circles are 16-port crossbar switches. The lines are 2+2 Gb/s links.
8 hosts
Ports to up to 128 hosts (line cards)



Backplane of the M3-E128 Switch

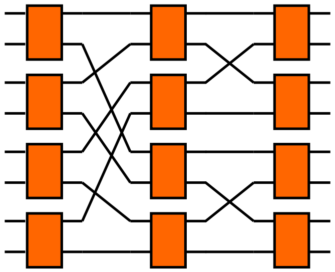
M3-SW16-8F fiber line card (8 ports)

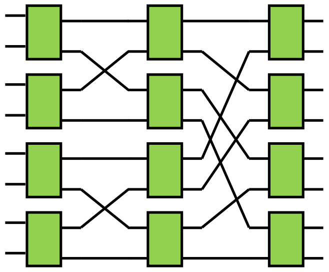
BENES NETWORK

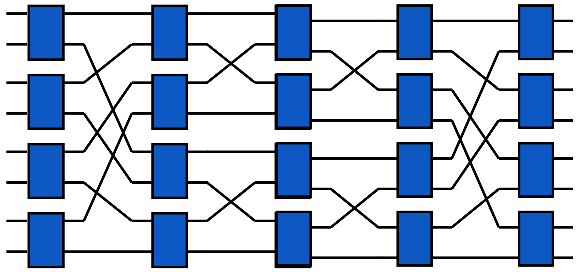
Benes Network

The ***Benes topology***

- Consists of 2 $\log_2 N$ - 1 stages
- Is **rearrangeable non-blocking**
- Can be obtained **recursively** applying the three-stage Clos network concept to the middle-stage set of switches to reduce all switches to 2 x 2 switches
- Corresponds to the **concatenation** of a **baseline** and a **reverse baseline**

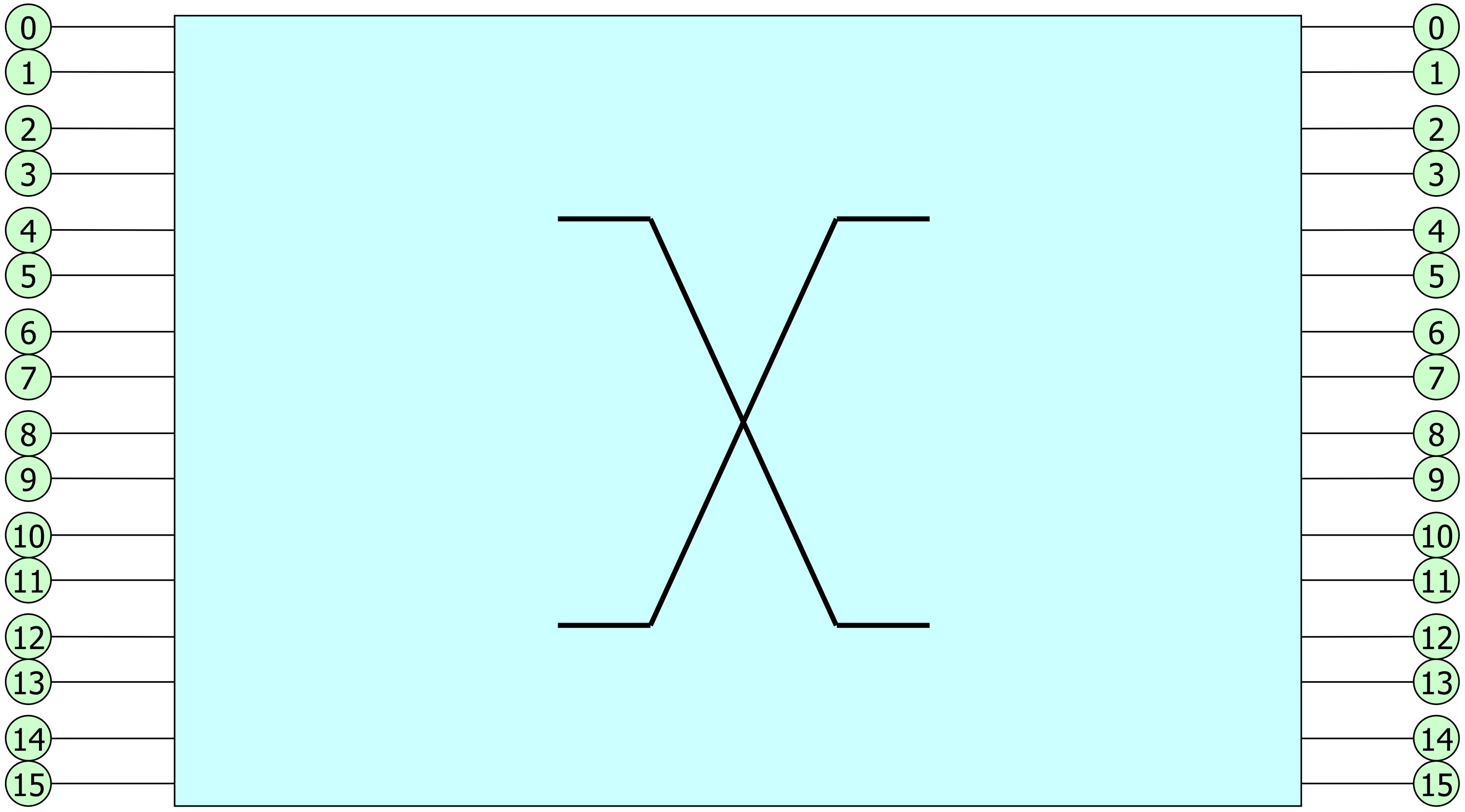

Baseline


Reverse Baseline

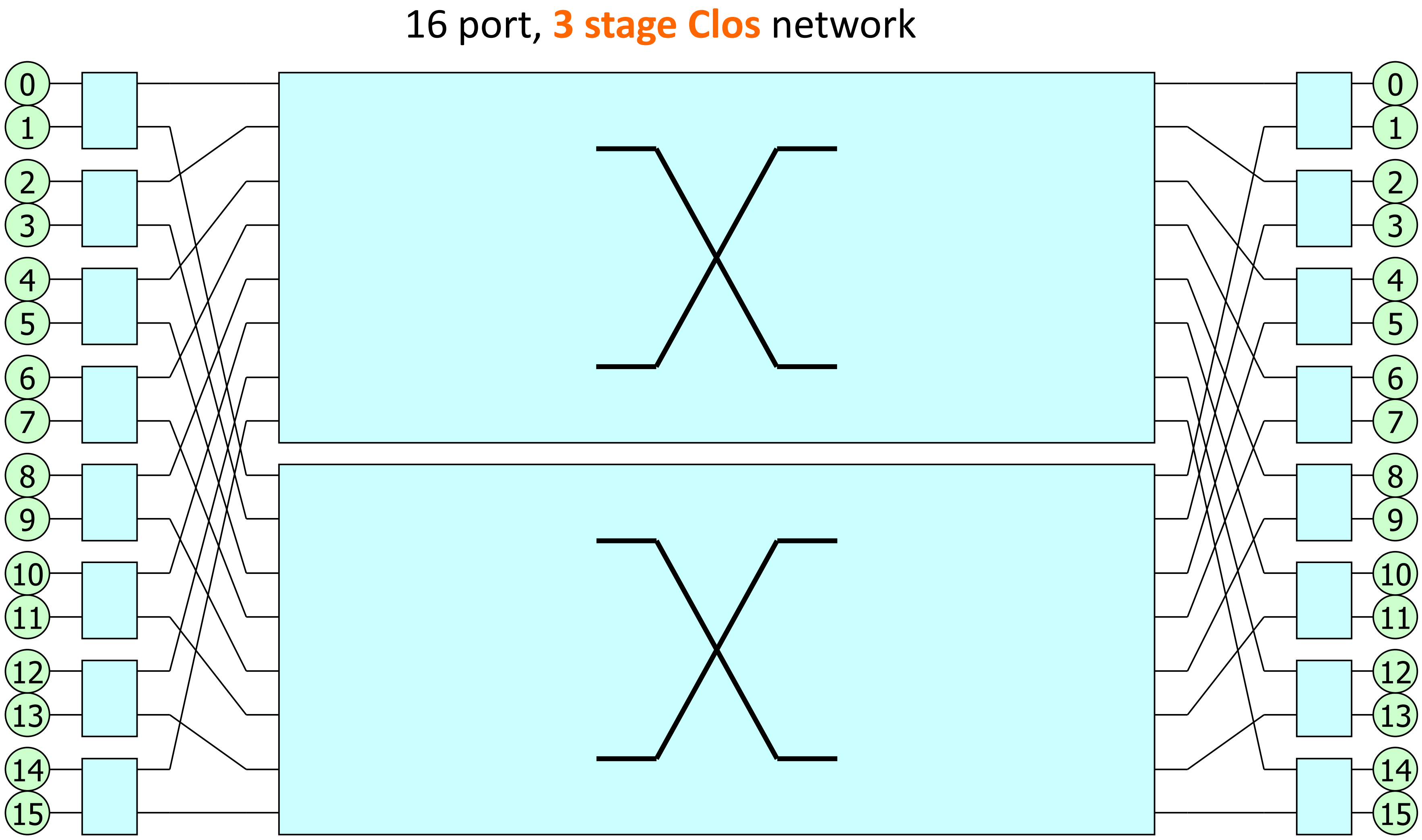

Benes

Benes Network

16 port **Crossbar** network

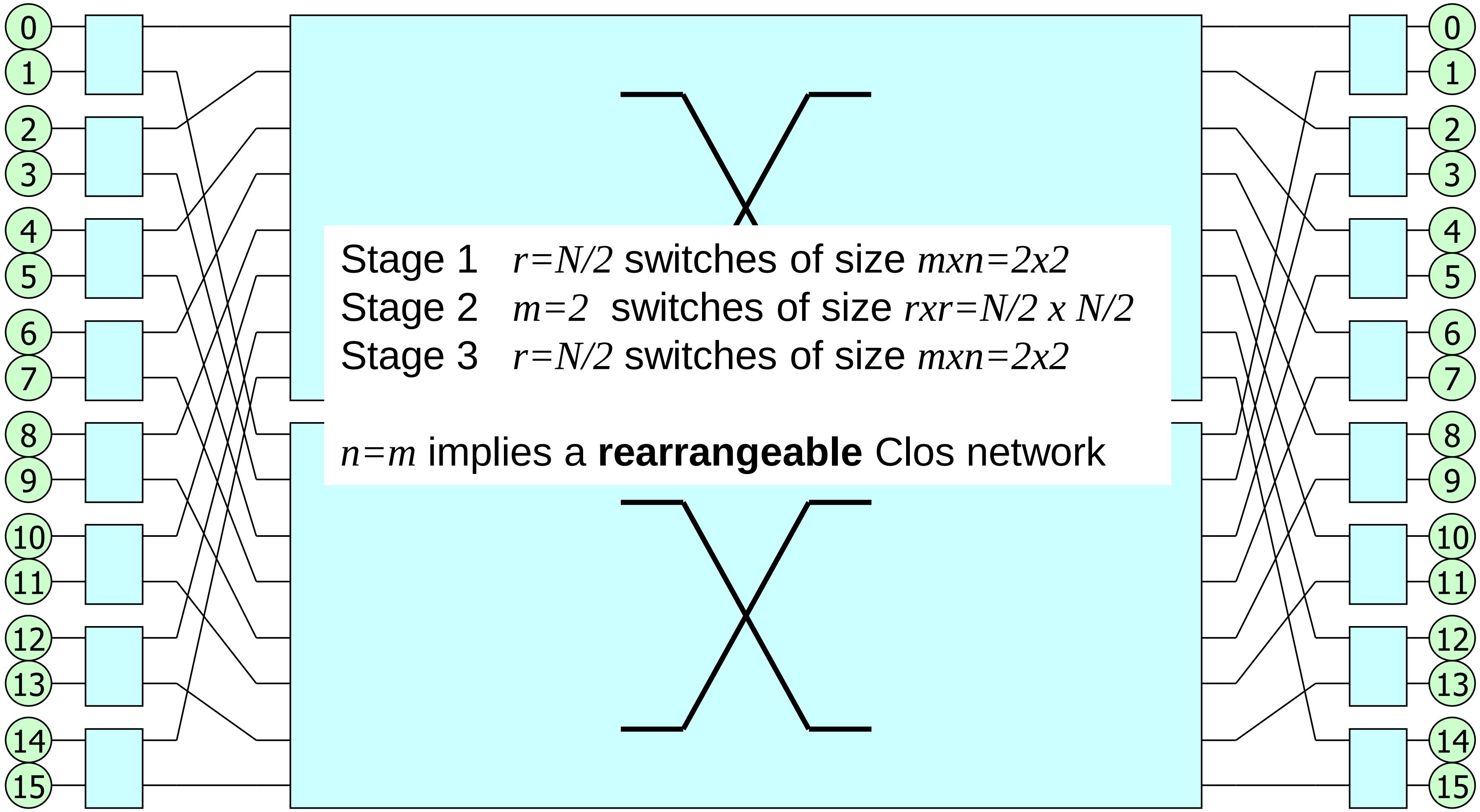


Benes Network



Benes Network

16 port, **3 stage Clos** network



Stage 1 $r=N/2$ switches of size $mxn=2x2$

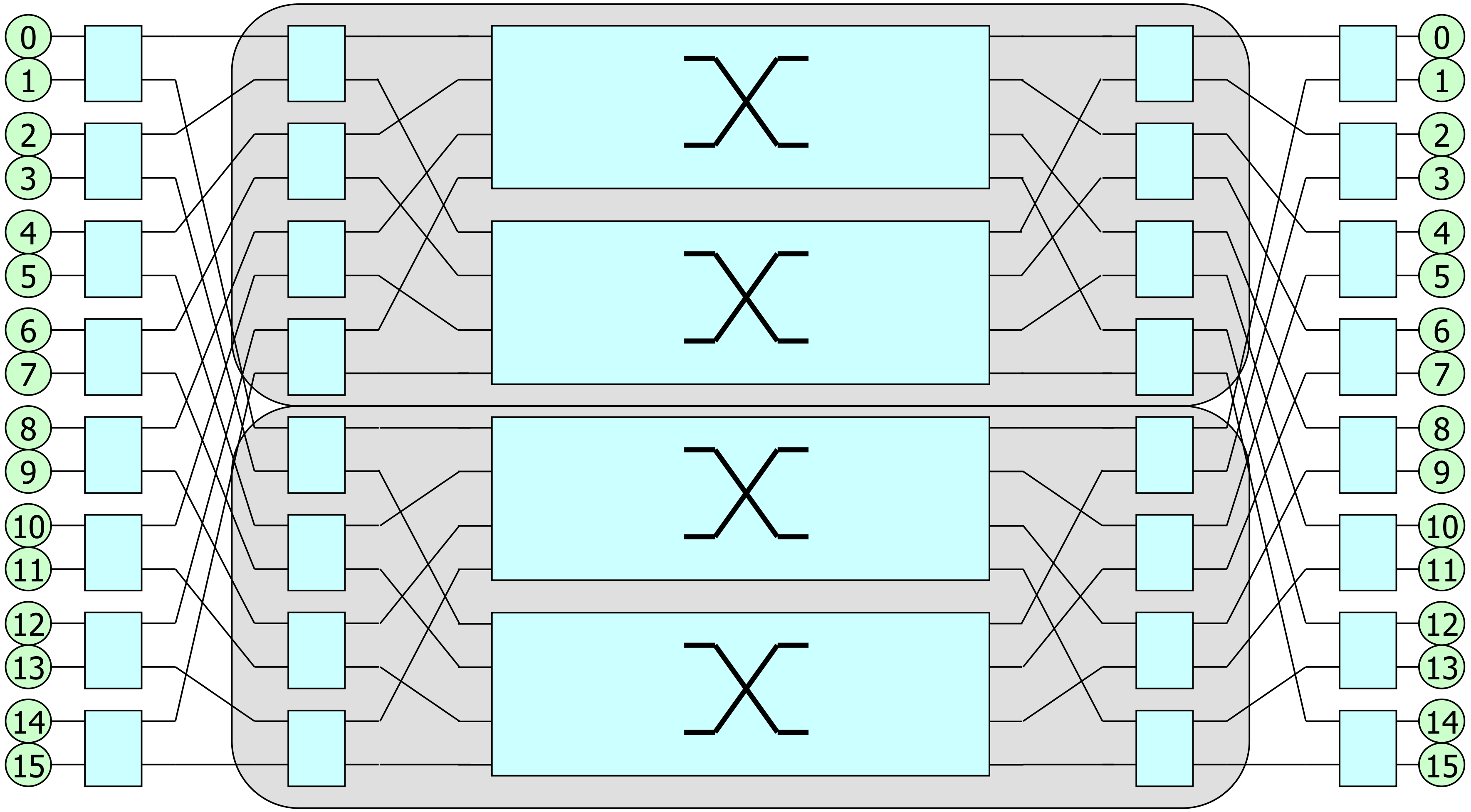
Stage 2 $m=2$ switches of size $rxr=N/2 \times N/2$

Stage 3 $r=N/2$ switches of size $mxn=2x2$

$n=m$ implies a **rearrangeable** Clos network

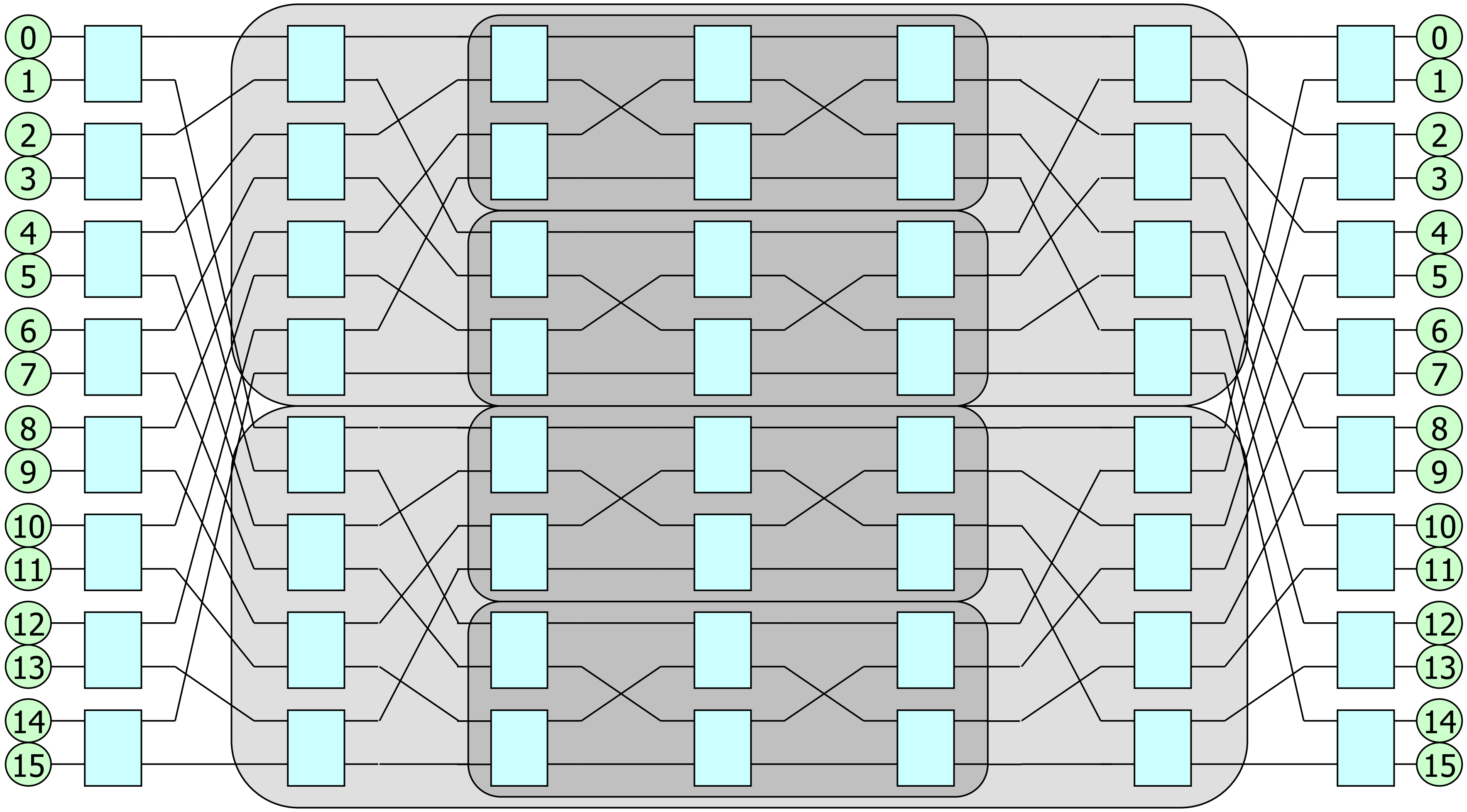
Benes Network

16 port, **5 stage Clos** network



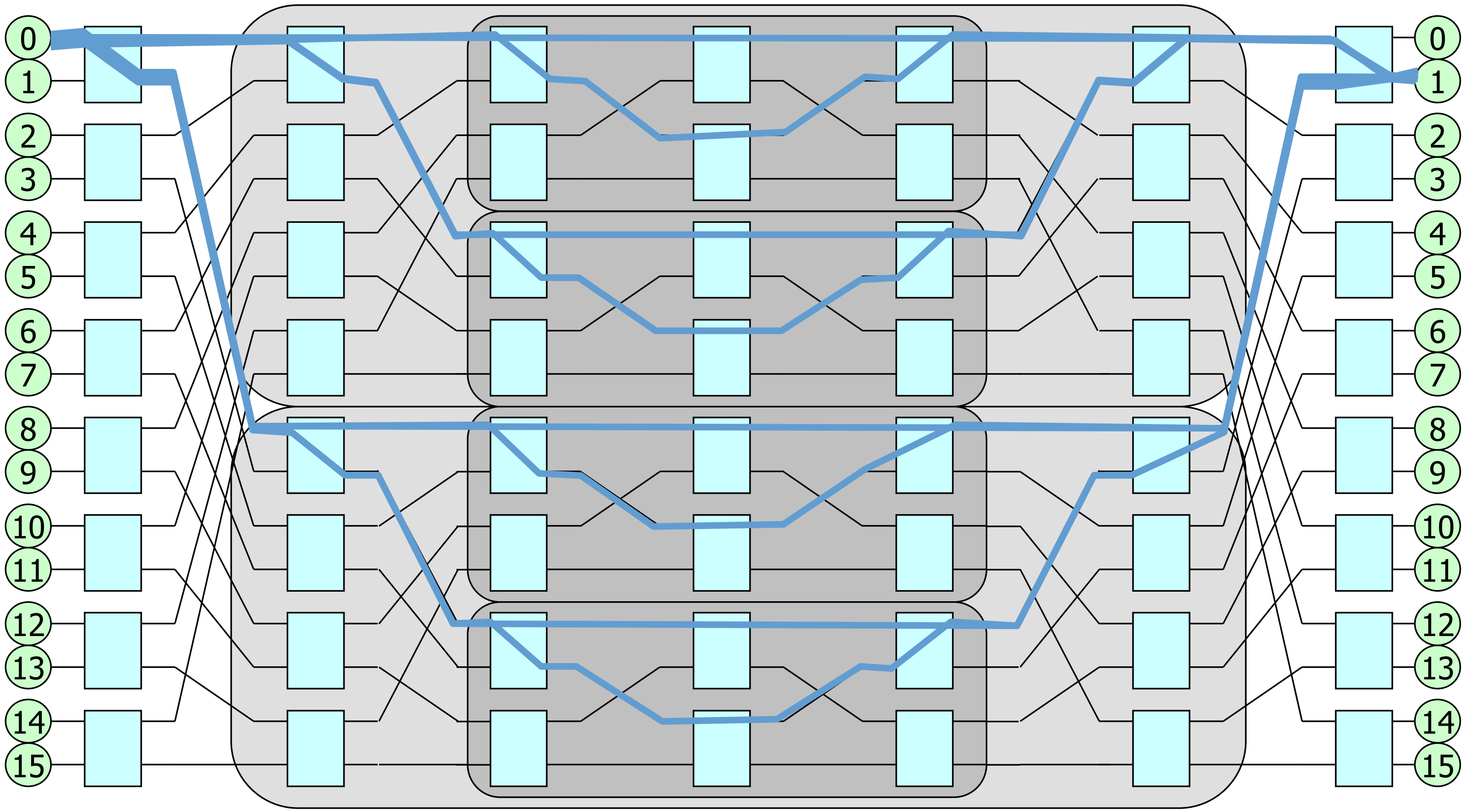
Benes Network

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



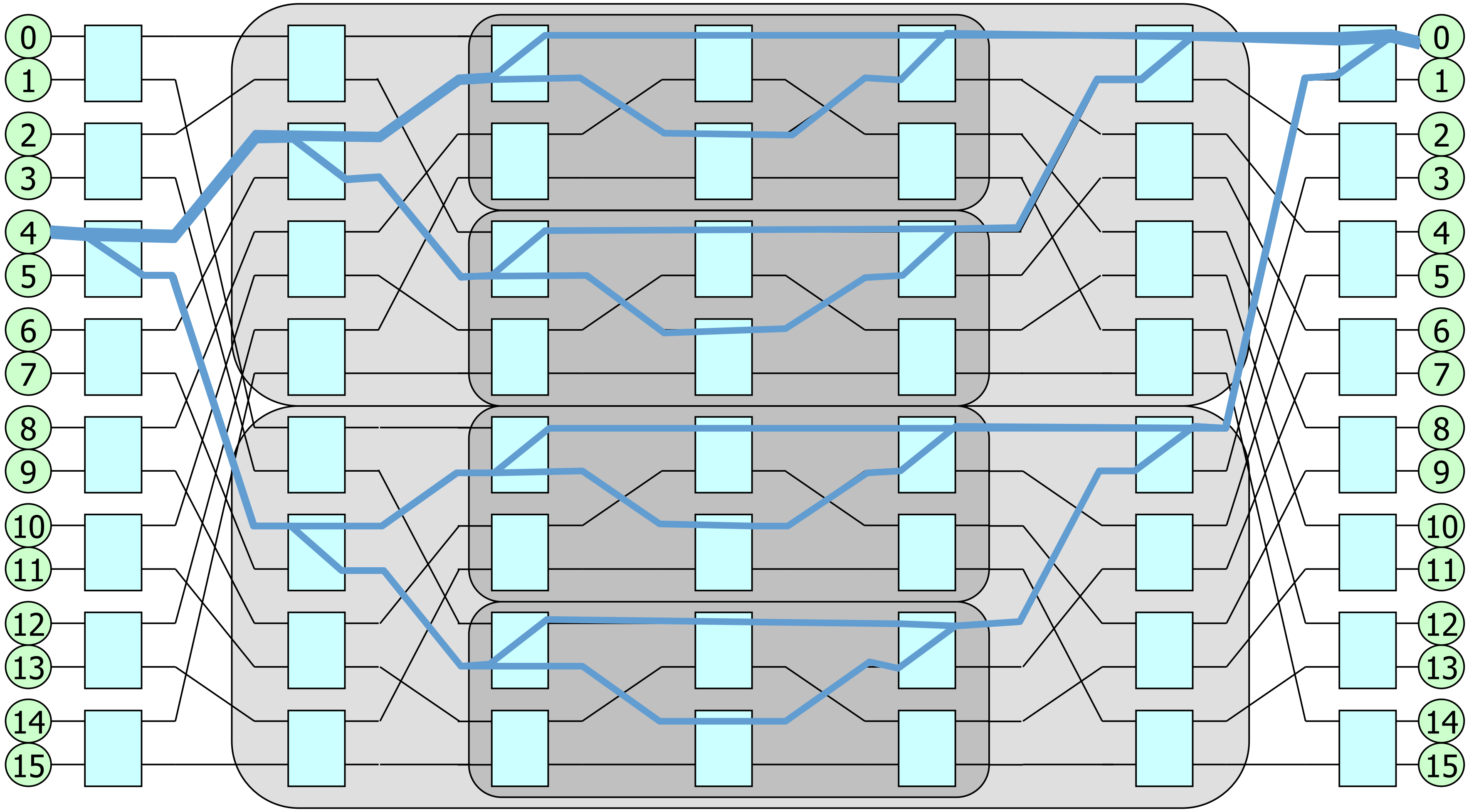
Benes Network

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



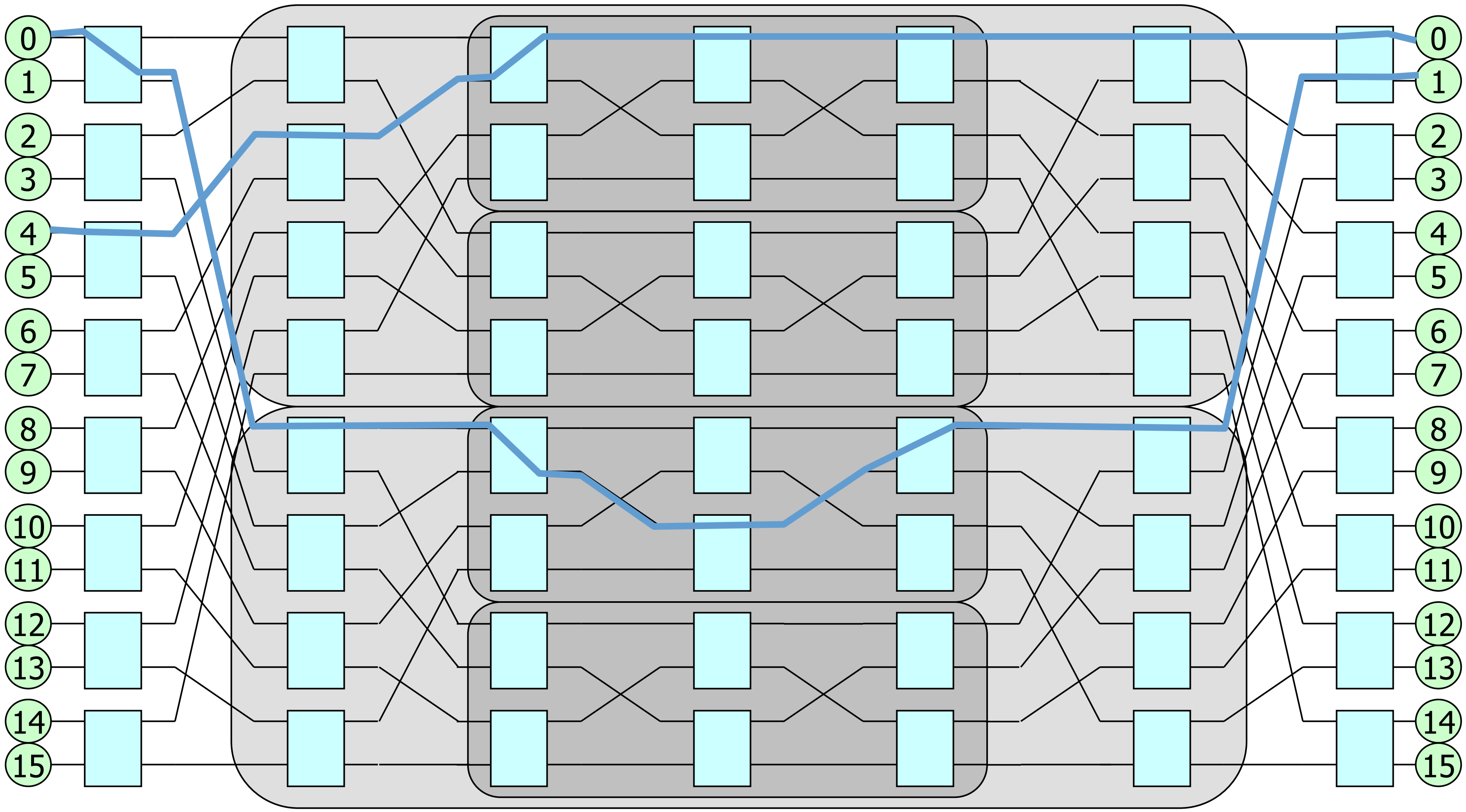
Benes Network

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



Benes Network

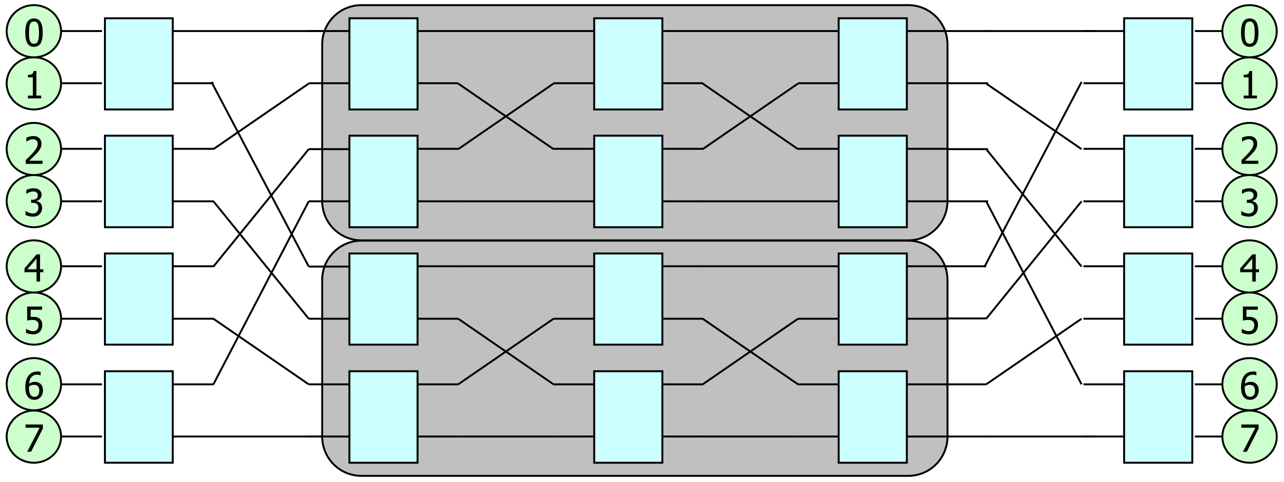
Contention free, paths 0 to 1 and 4 to 1 in a 16 port **Benes topology**



Looping algorithm

Realizing permutations on a Benes network

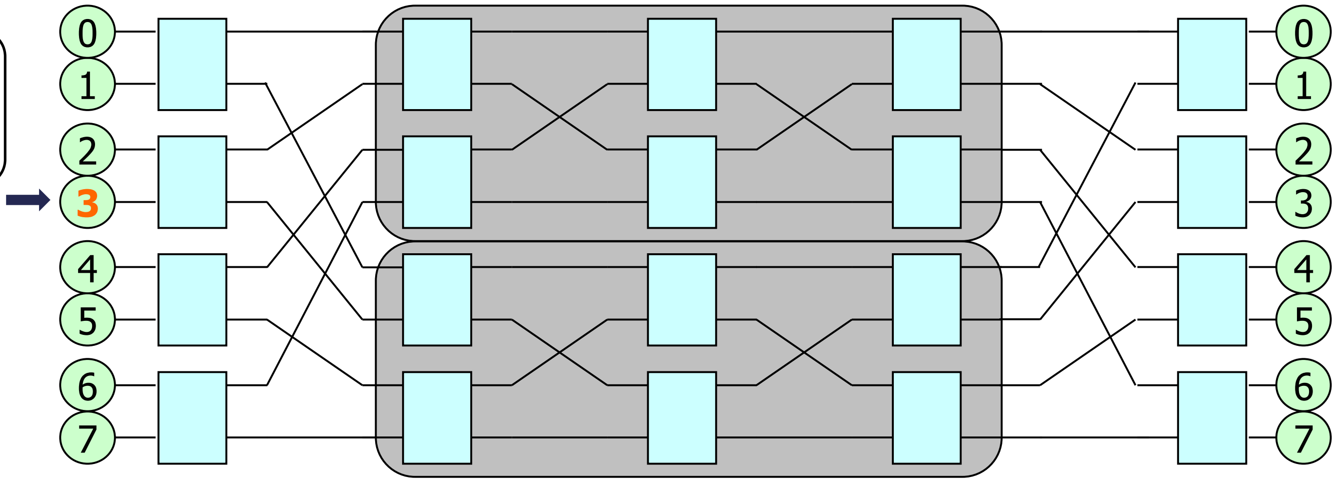
- Start from arbitrarily chosen input by arbitrarily setting the corresponding switch
- Connect the input to the requested output
- Connect back the other output of the switch in the last stage to the corresponding input
- The algorithm follows this procedure, **looping back and forth** between inputs 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

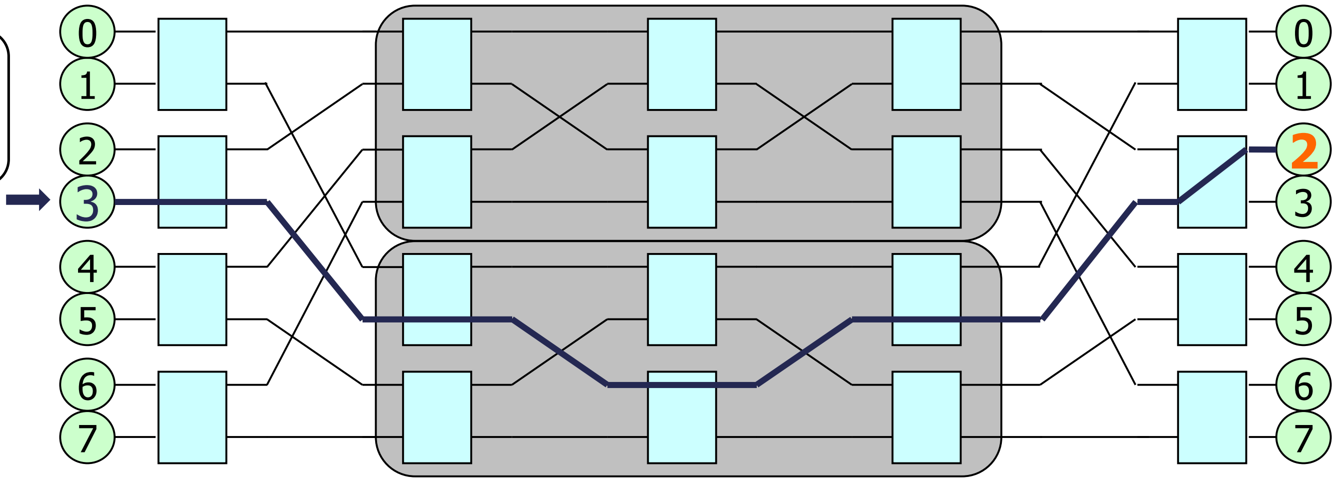
$$\begin{pmatrix} 0 & 1 & 2 & \mathbf{3} & 4 & 5 & 6 & 7 \\ 5 & 6 & 1 & 2 & 4 & 7 & 3 & 0 \end{pmatrix}$$



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

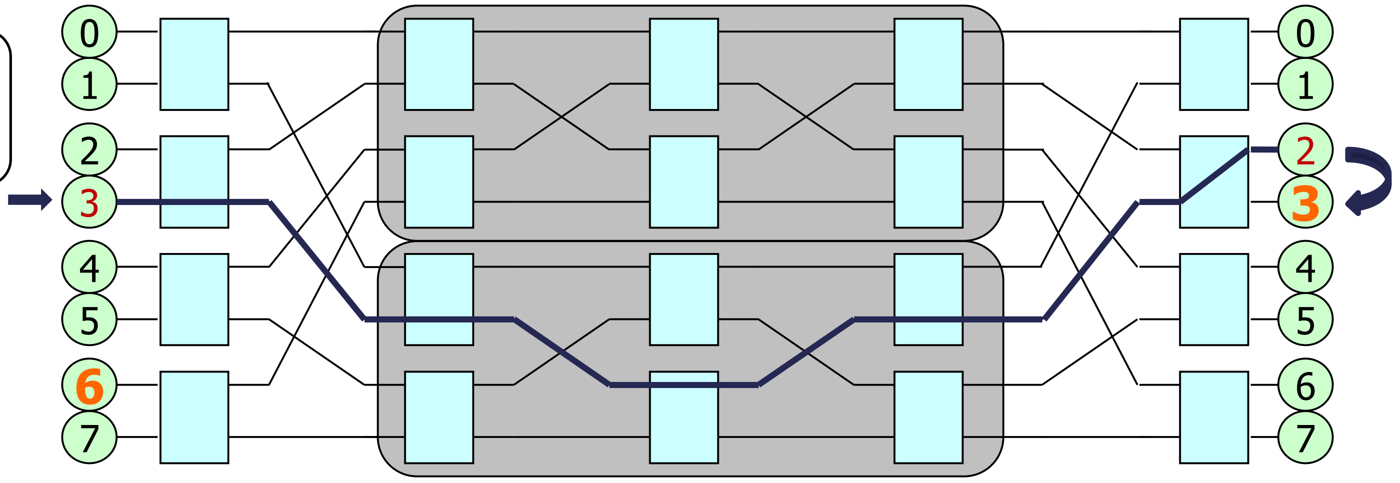
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 1 & 2 & 4 & 7 & 3 & 0 \end{pmatrix}$$



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

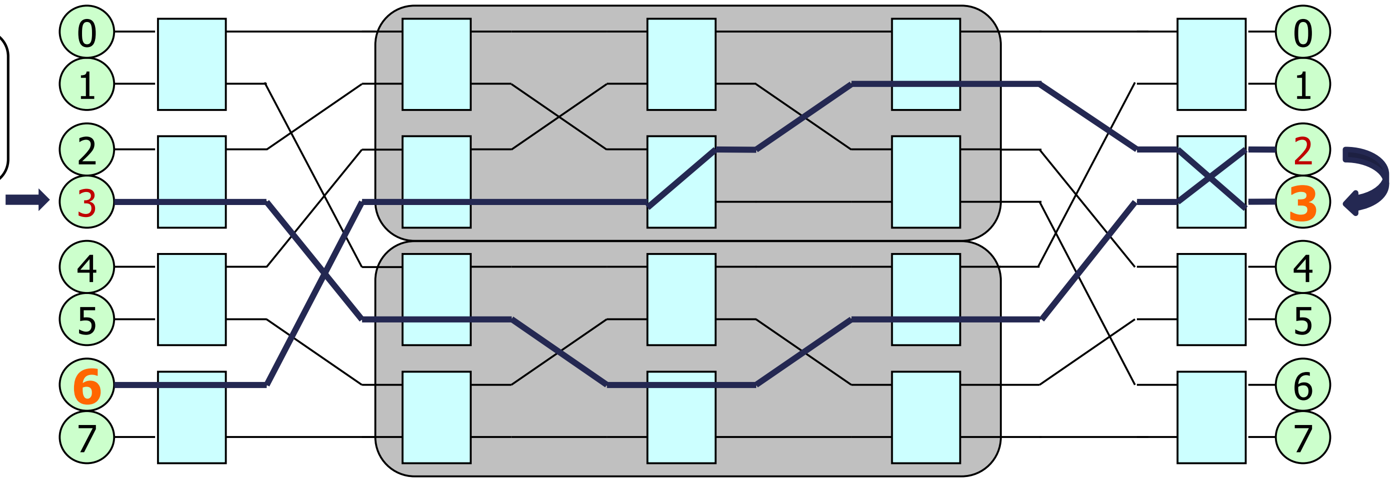
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 1 & 2 & 4 & 7 & 3 & 0 \end{pmatrix}$$



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

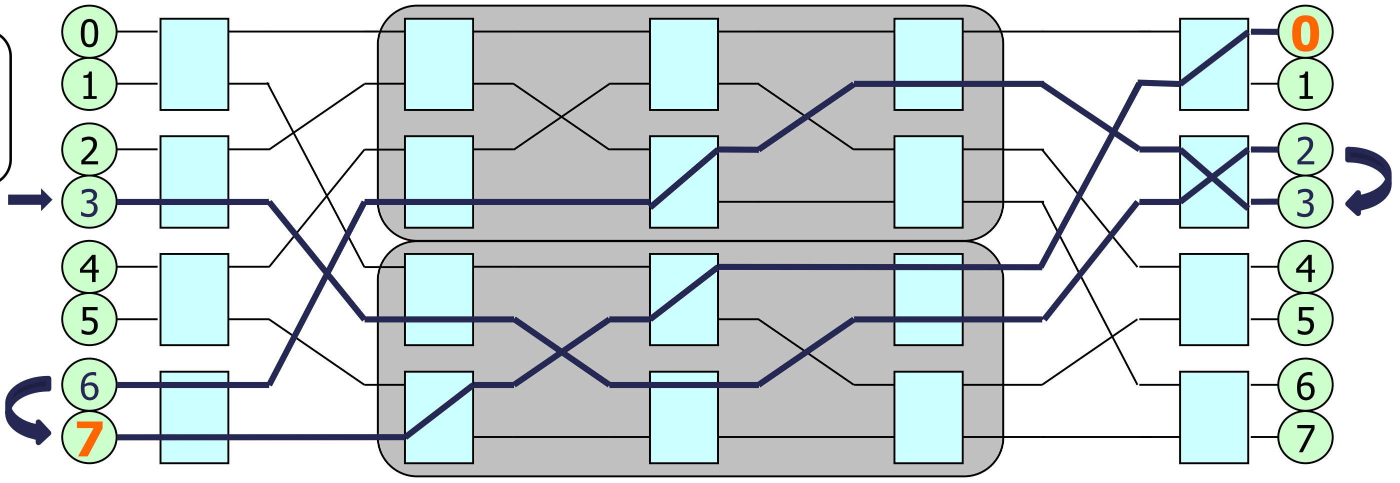
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 1 & 2 & 4 & 7 & 3 & 0 \end{pmatrix}$$



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

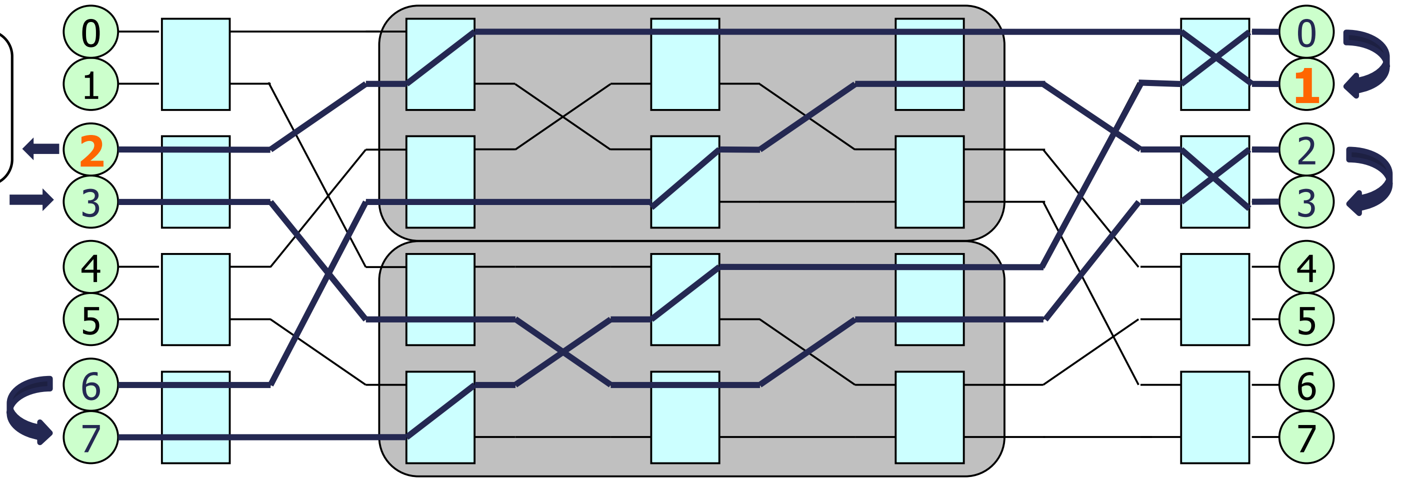
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 1 & 2 & 4 & 7 & 3 & 0 \end{pmatrix}$$



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

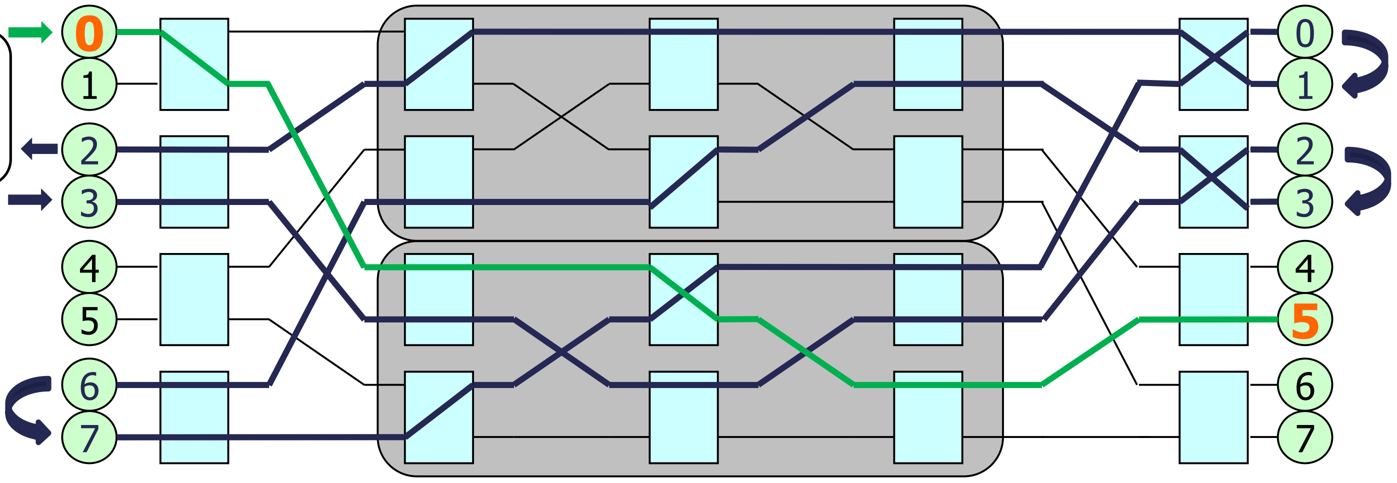
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 1 & 2 & 4 & 7 & 3 & 0 \end{pmatrix}$$



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

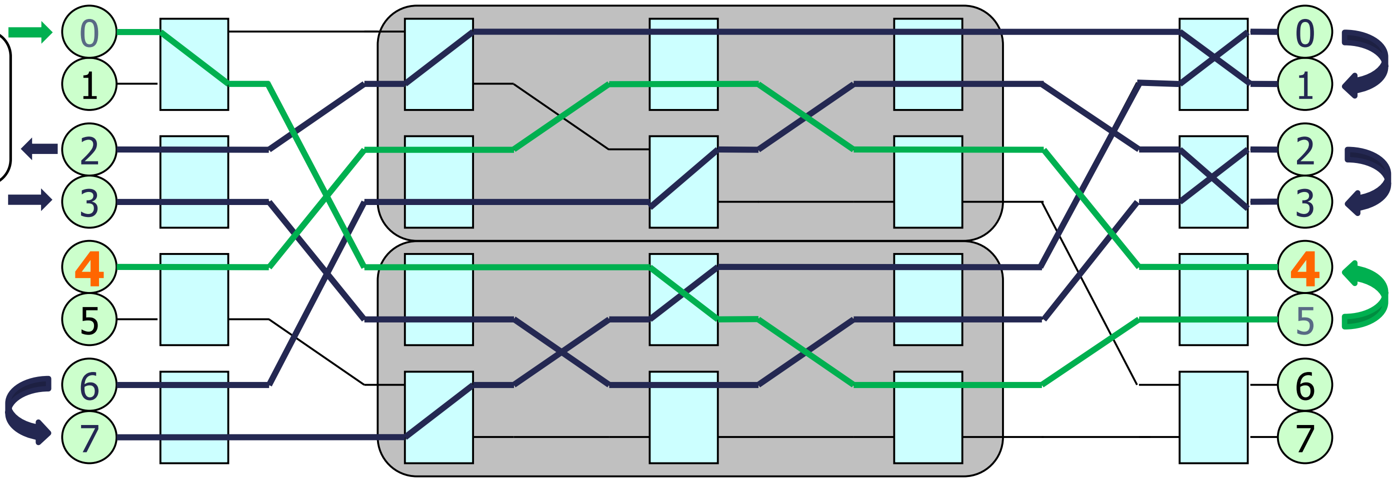
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 1 & 2 & 4 & 7 & 3 & 0 \end{pmatrix}$$



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

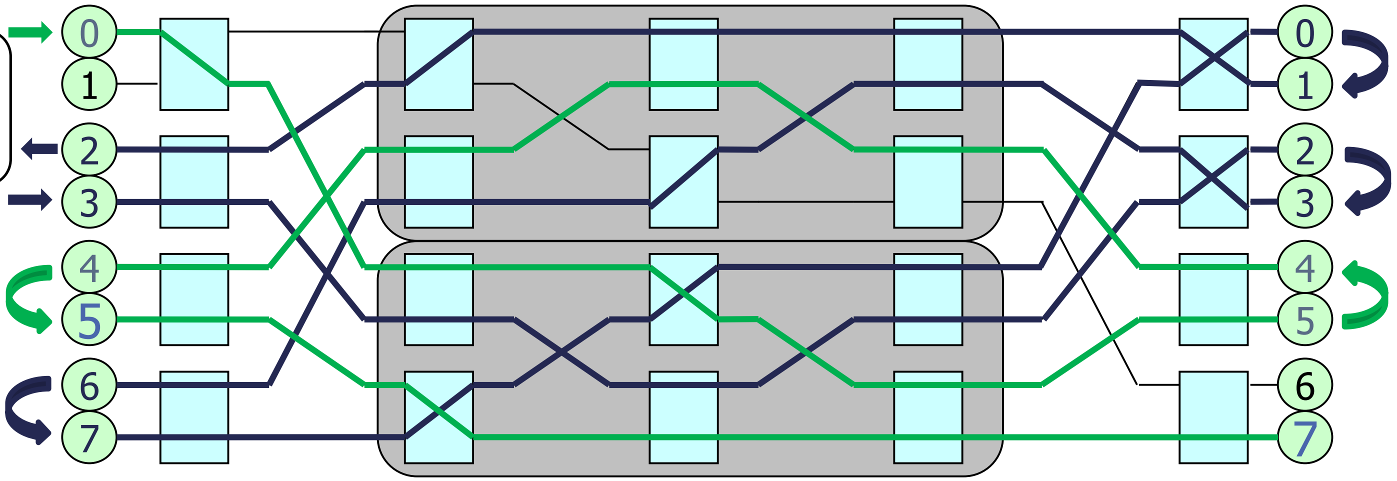
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 1 & 2 & 4 & 7 & 3 & 0 \end{pmatrix}$$



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

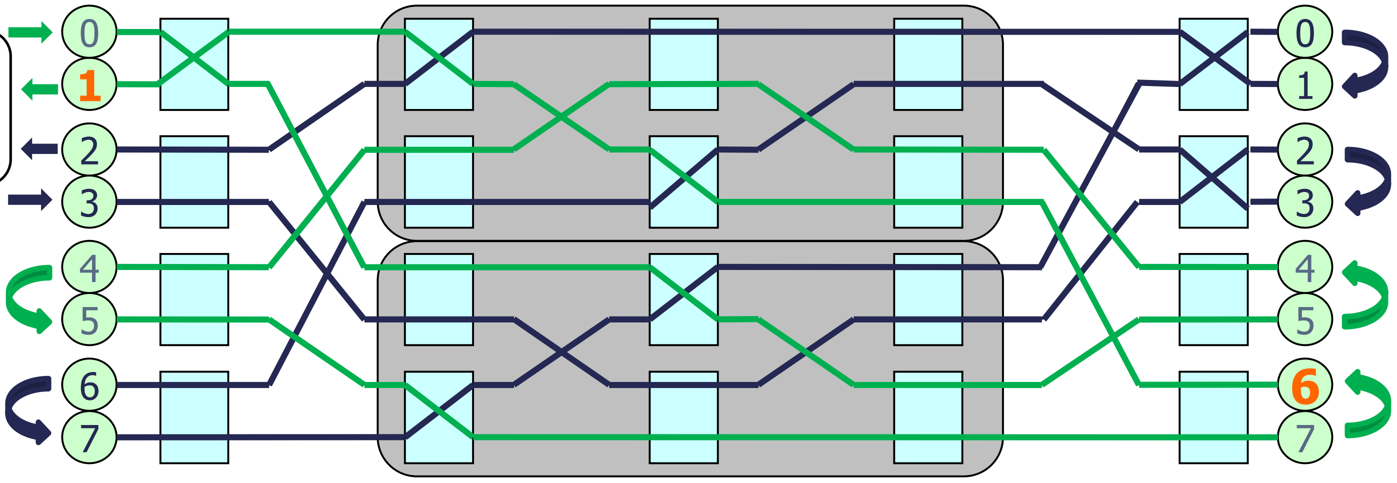
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 1 & 2 & 4 & 7 & 3 & 0 \end{pmatrix}$$



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

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 1 & 2 & 4 & 7 & 3 & 0 \end{pmatrix}$$



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

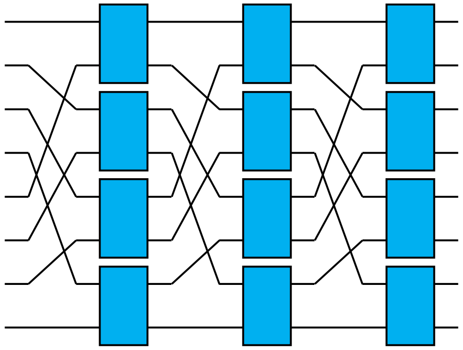
Topological and functional equivalence

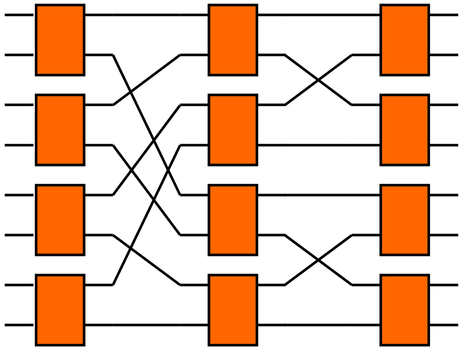
- There are two different concepts of equivalence:
  - **Topological** equivalence: isomorphism
  - **Functional** equivalence: capability of always performing the same set of assignments (communication requests)

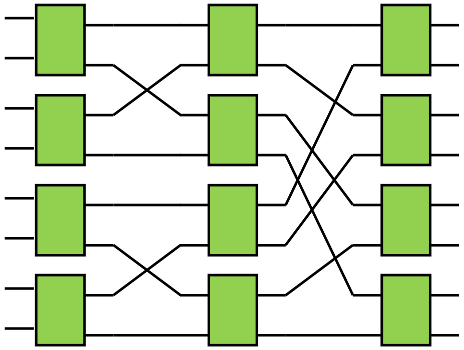
- Topological equivalence and functional equivalence are different:
  - All **rearrangeable** MINs (e.g., Clos and Benes) are functionally equivalent (because they can realize all the permutations) though not necessarily topologically equivalent
  - **Not rearrangeable** MINs can be topologically equivalent but not functionally equivalent, as in the case of log N stage MINs

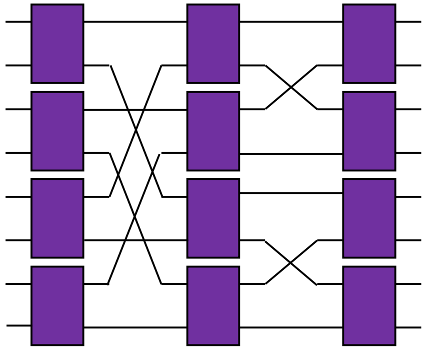
Topological equivalence

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


Omega


Baseline

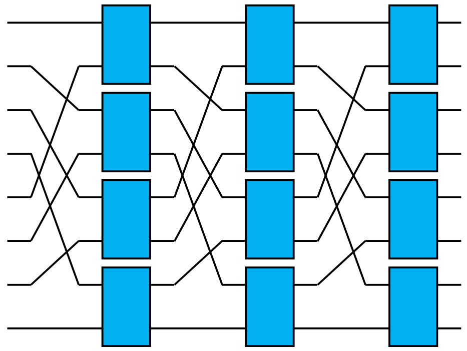

Reverse Baseline

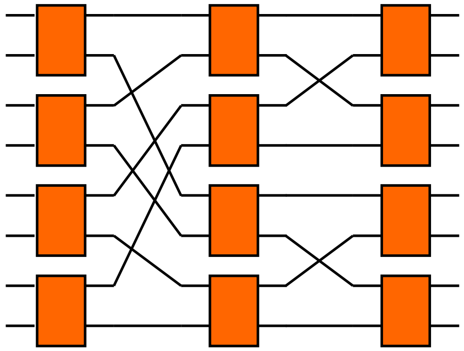

Butterfly

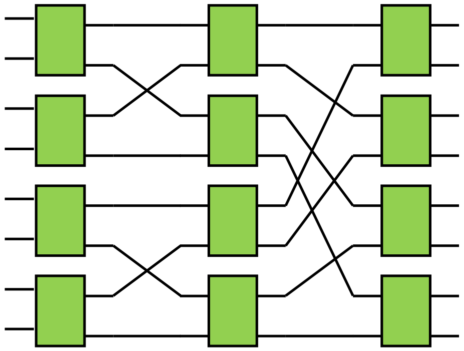
Topological equivalence

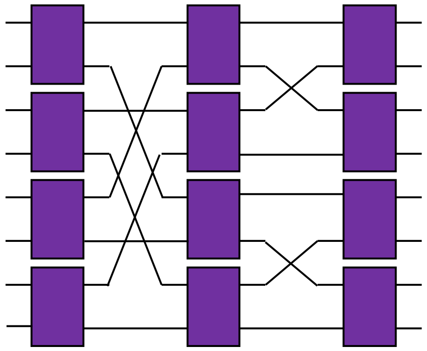
Bermond, Fourneau and Jean-Marie (1987) gave the properties to show the topological equivalence of MINs to the ***Reverse Baseline:*** the Banyan property and the $P(*, *)$ property

- ***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


Omega

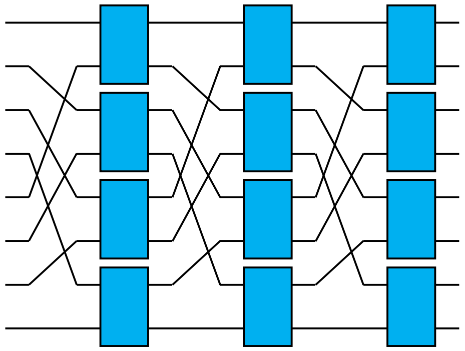

Baseline

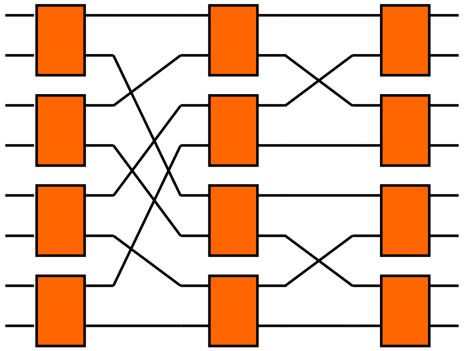

Reverse Baseline

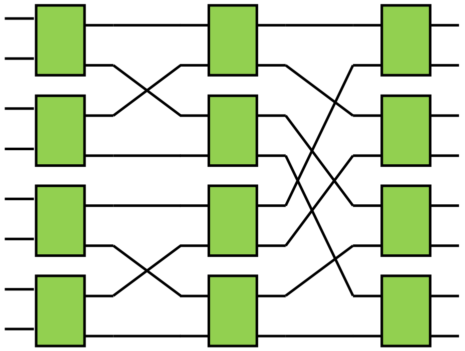

Butterfly

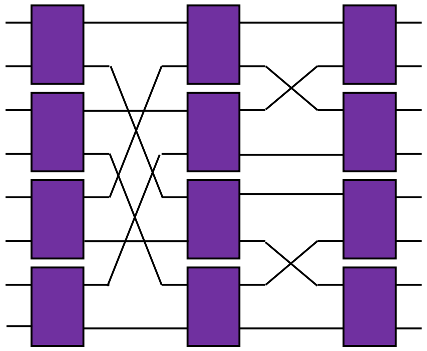
Topological equivalence

- *P(∗, ∗) property*
  - ***Property P(i,j)*** An N-MIN has property P(i, j) for 1 ≤ i ≤ j ≤ log N if the subgraph Gi,j induced by the nodes of the stage from i to j has exactly 2log 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 ≤ i ≤ j ≤ log N


Omega


Baseline

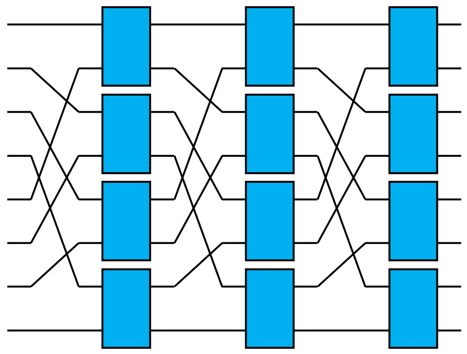

Reverse Baseline

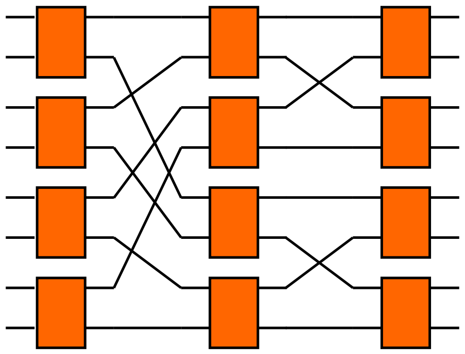

Butterfly

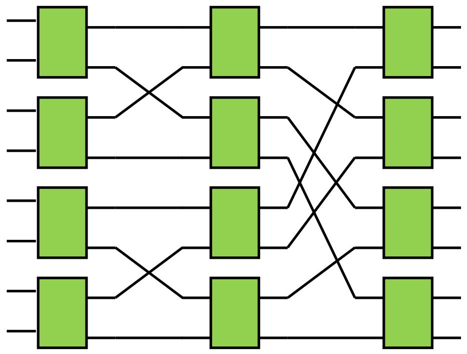
Topological equivalence

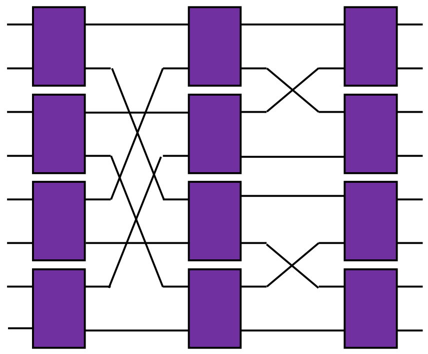
The topological equivalence to the Reverse Baseline network is stated in the following theorem

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


Omega


Baseline


Reverse Baseline


Butterfly

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** (LCP) defined by Even and Litman (1992)
  - An **l-layered graph**, $G = (V_1, V_2, \ldots, V_l, E)$ 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 $G' = (V'_1, V'_2, \ldots, V'_l, E')$ and $G'' = (V''_1, V''_2, \ldots, V''_l, E'')$ is an l-layered graph $G = (V_1, V_2, \ldots, V_l, E)$ where:
    - V_i is the cartesian product of V'_i and V''_i, $1 \leq i \leq l$
    - an edge $<(u', u''),(v', v'')>$ belongs to E if and only if $<u', v'> \in E'$ and $<u'', v''> \in E''$
    - 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

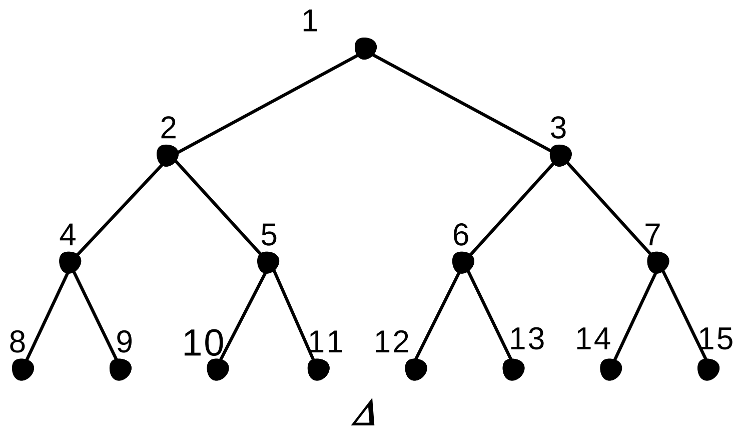
The following theorem and corollary state the ***topological equivalence between MINs***

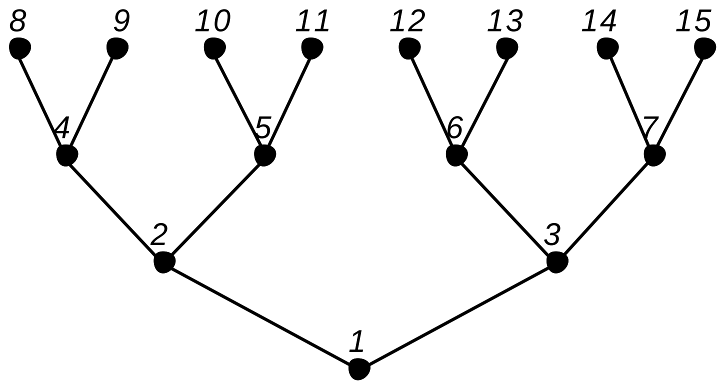
- **Theorem** Let G′ and G″ be two *s* stage MINs, and let G′ decomposable as $G'_1 \otimes G'_2$. Then G″ is topologically equivalent to G′ if and only if G″ can be decomposed as $G'_1 \otimes G'_2$
- **Corollary** Given two N-MINs $G' = G'_1 \otimes G'_2$ and $G'' = G''_1 \otimes G''_2$, they are topologically equivalent if their factors are topologically equivalent

- A particularly interesting decomposition is based on **binary trees**

Topological equivalence

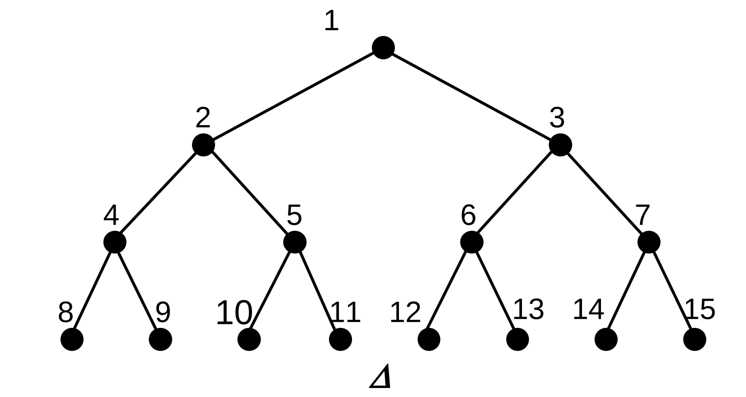
- **Lemma** A MIN G satisfies the Banyan and P(∗, ∗) properties if and only if it can be decomposed as $\Delta \otimes \nabla$, where Δ and ∇ denote binary trees with the root on the top and in the bottom, respectively



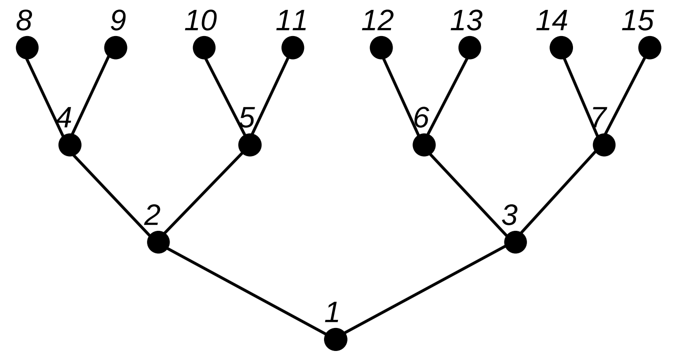


Nabla

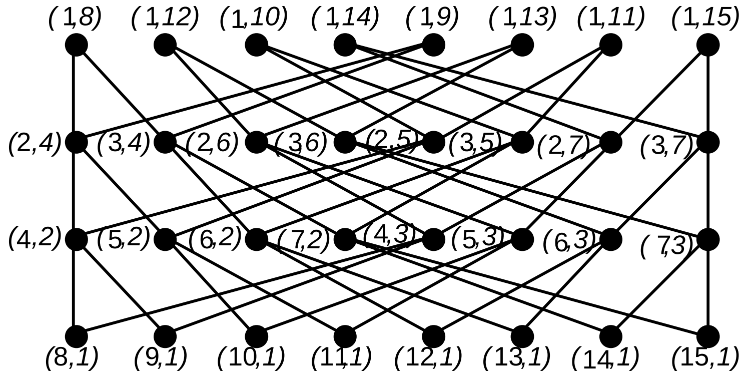
- **Theorem** A MIN G is decomposable as $\Delta \otimes \nabla$ if and only if G is topologically equivalent to the Reverse Baseline



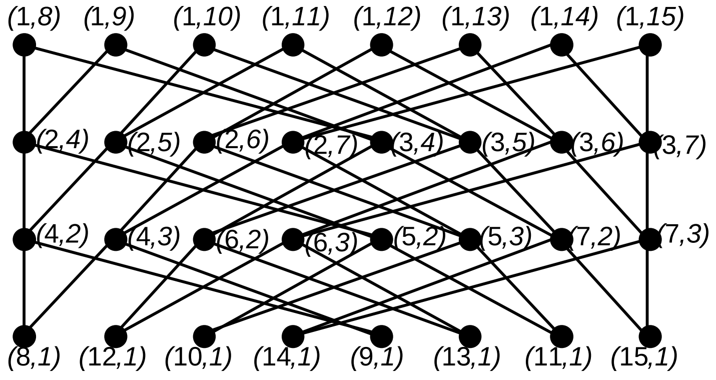
Δ



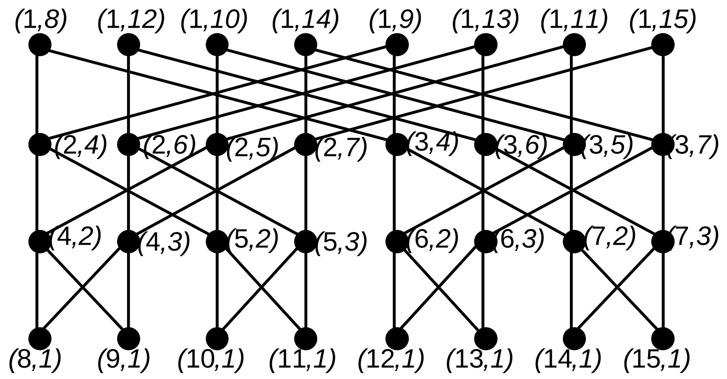
Nabla



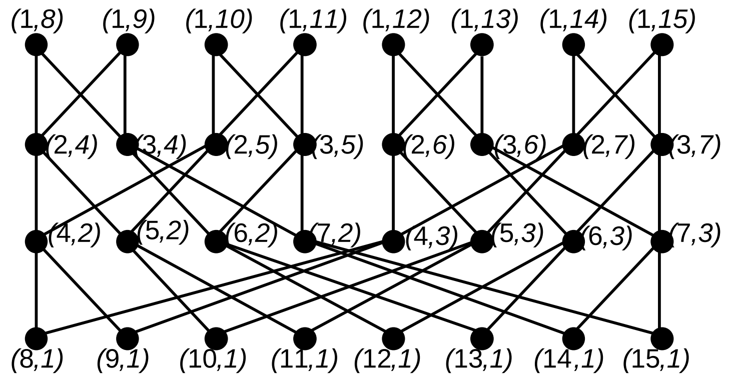
Omega



Flip



Butterfly

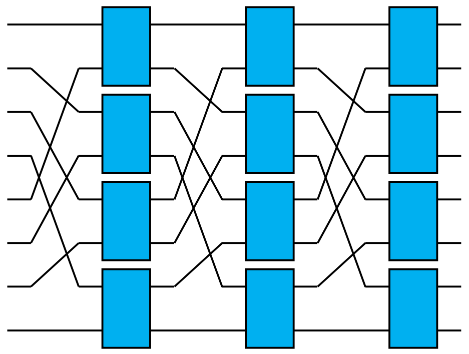


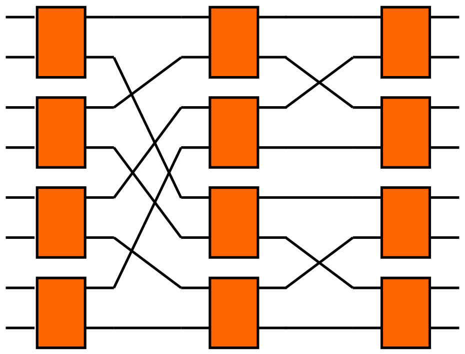
Reverse Baseline

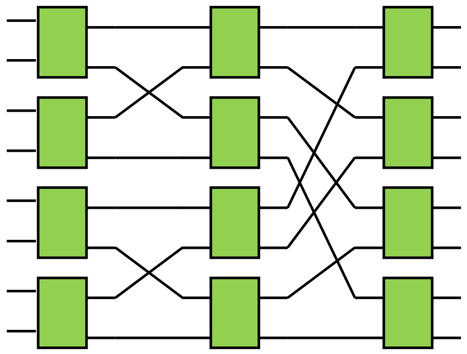
Topological equivalence

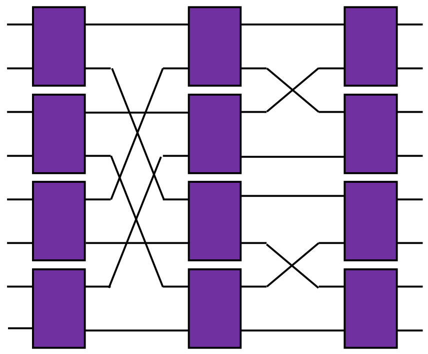
- MINs consisting of log N stages such as Omega, Flip (Reverse Omega), Baseline and Reverse Baseline, Butterfly and Reverse Butterfly are all equivalent networks

- They have attractive features, but they are **not rearrangeable**


Omega

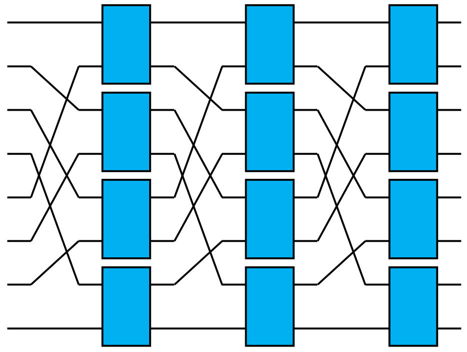

Baseline

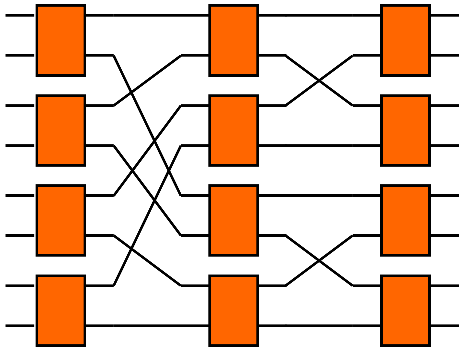

Reverse Baseline

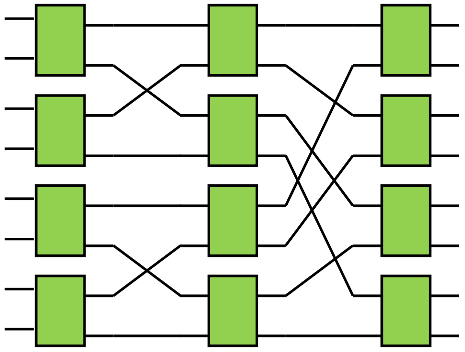

Butterfly

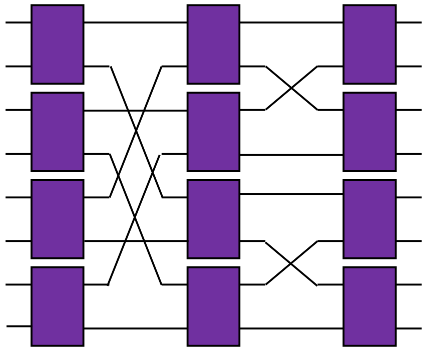
Topological equivalence

- 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**


Omega


Baseline


Reverse Baseline


Butterfly

2 LOG N-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

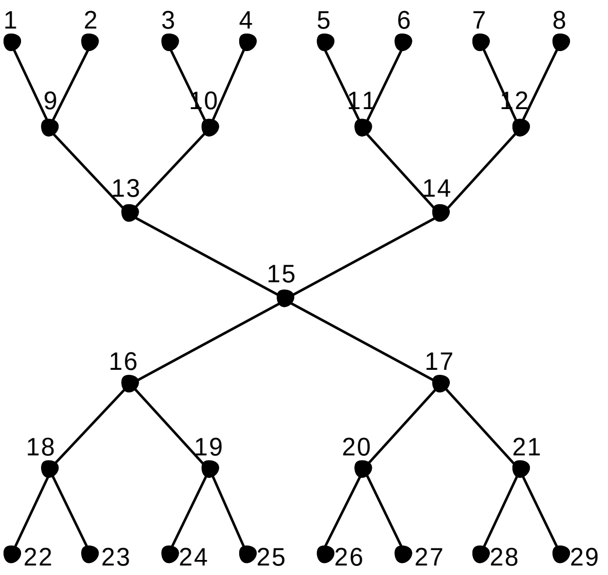
- Concatenating pairs of log N stage networks such as Butterfly, Omega, Flip, Baseline, their reverses, and so on, many different N-MINs can be obtained

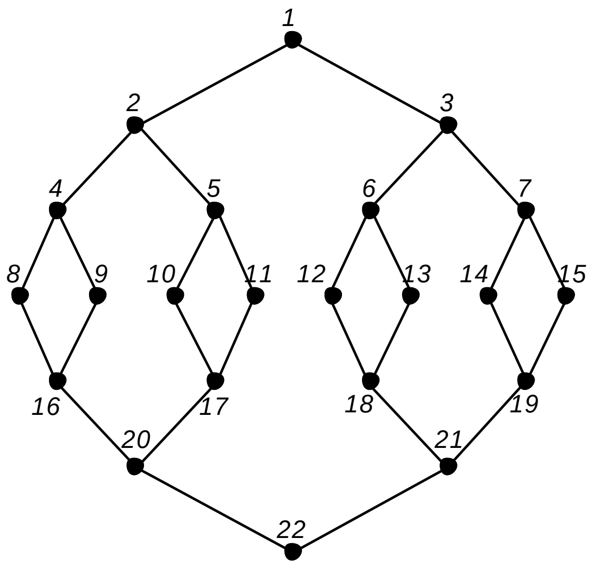
- Both the two log N stage MINs constituting a (2log N- 1) stage MIN can be decomposed as LCP of $\Delta \otimes \nabla$

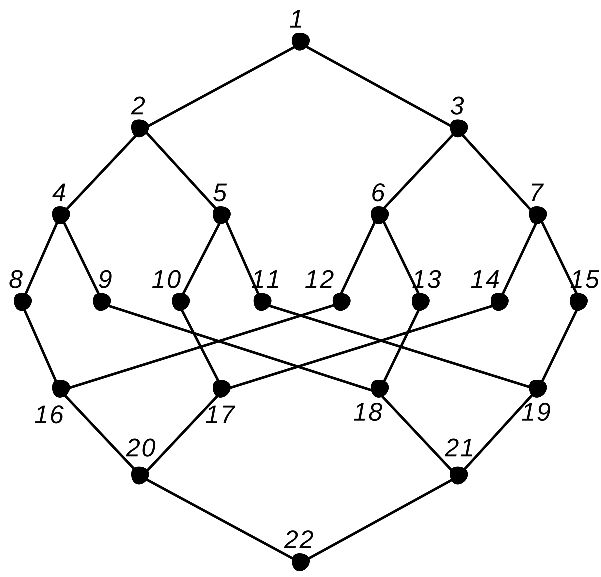
- As a consequence, we obtain that the factors of (2log N- 1) stage MIN are the concatenation of a Δ and a ∇ (leaves merging) and of a ∇ and a Δ (roots merging)

2logN-1 stage MIN equivalence

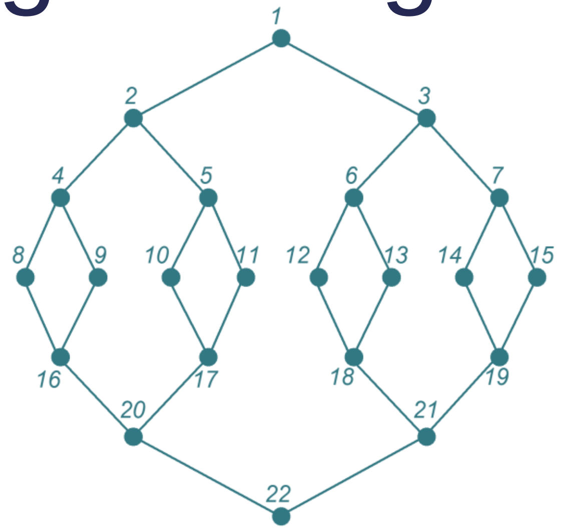
- It is obvious how to merge the last layer of a ∇ with the first layer of a Δ
- But there are **many ways** of merging the last layer of a Δ and the first layer of a ∇ respectively

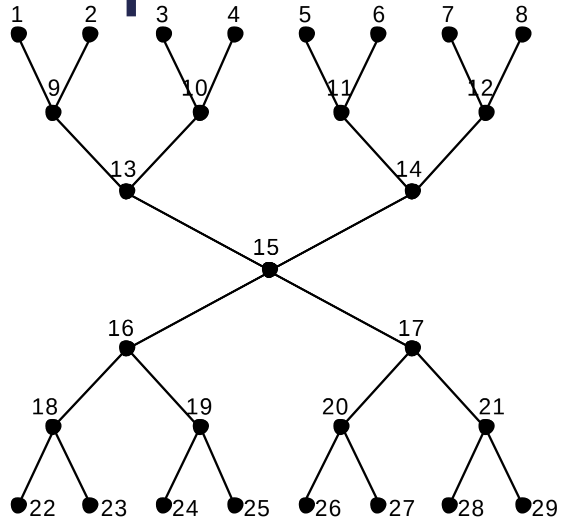


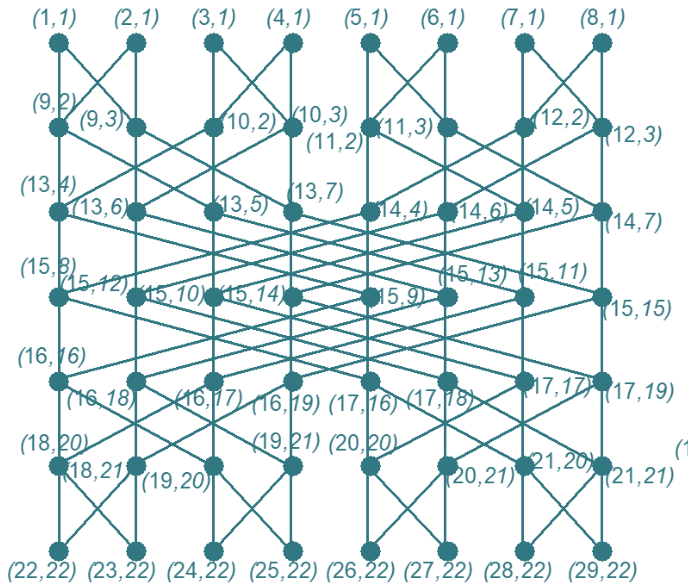




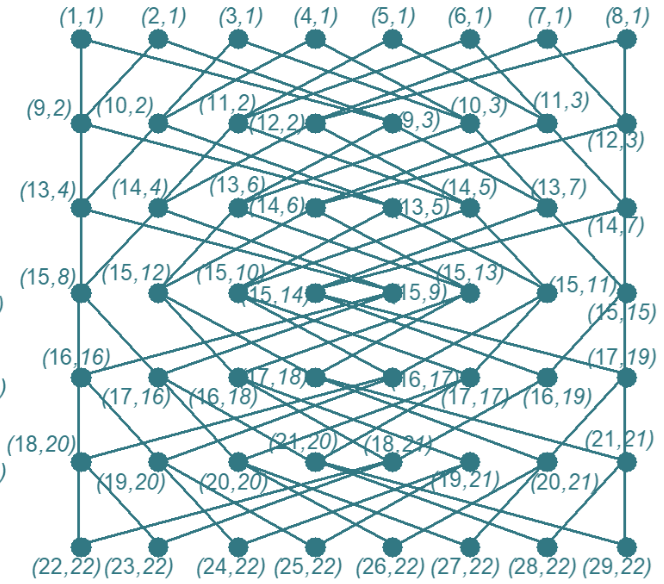
2logN-1 stage MIN equivalence





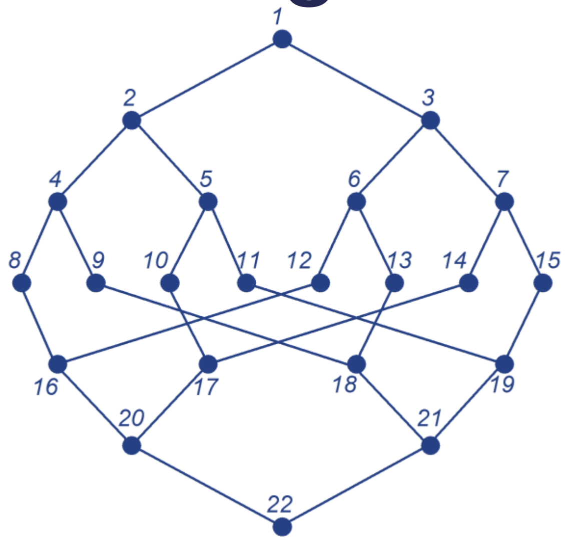


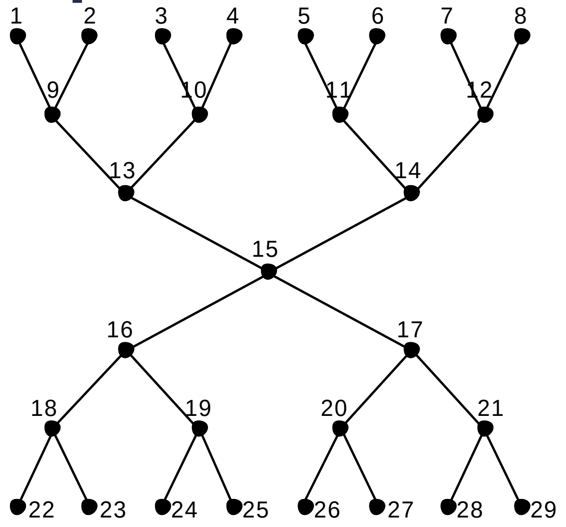
A reverse Butterfly and a Butterfly

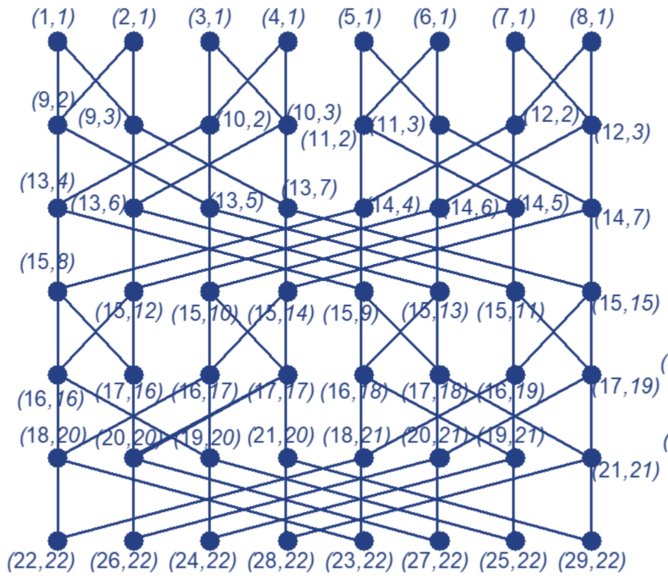


A Flip and a Omega

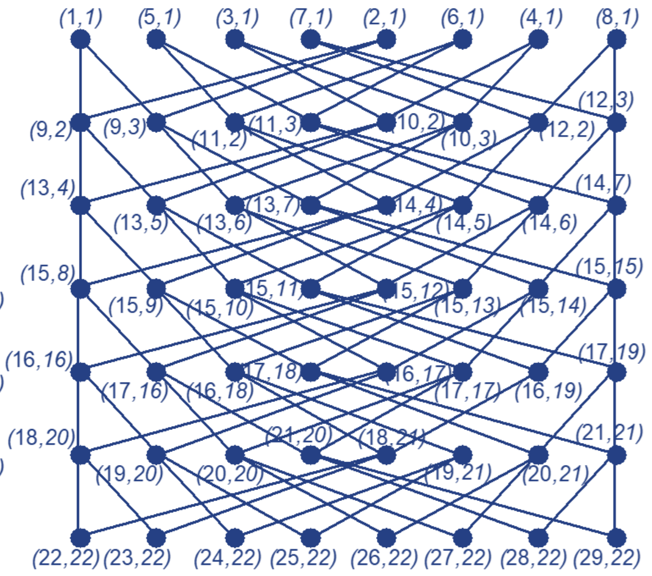
2logN-1 stage MIN equivalence







Two reverse Butterflies



Two Omega

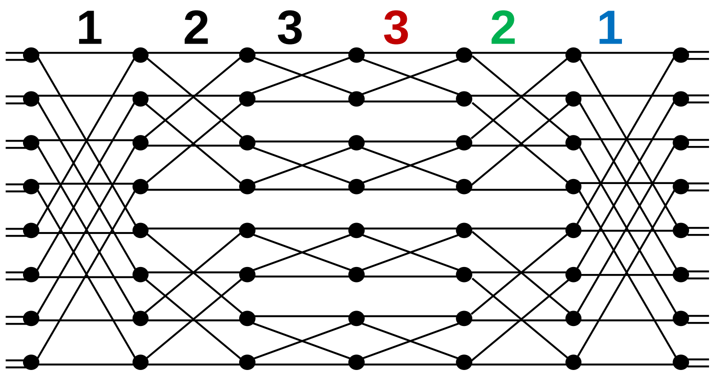
2logN-1 stage MIN equivalence

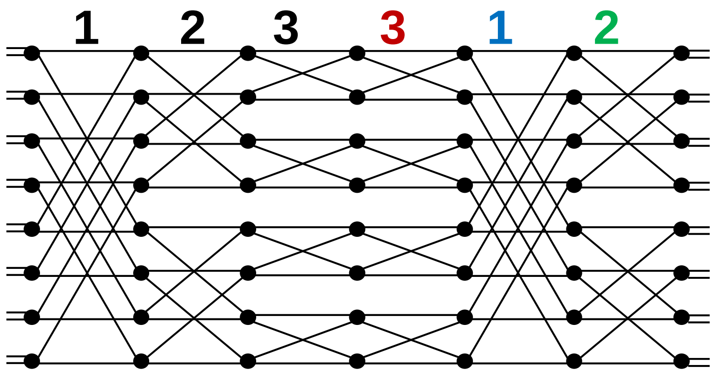
- **Theorem** The number of distinct equivalence classes of (2 logN - 1) MINs is (log N – 1)!

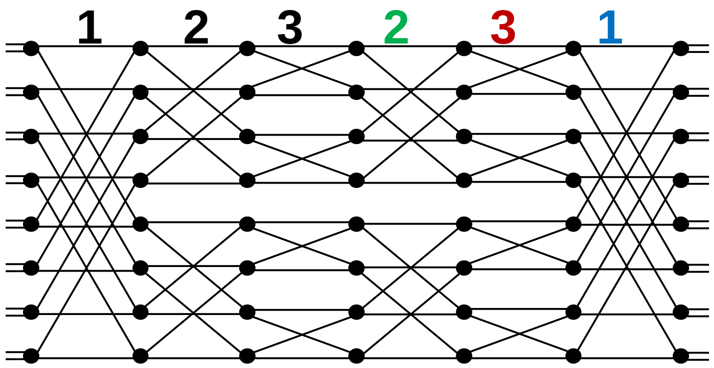
- We can represent these classes representing the MINs using **Butterfly stages**

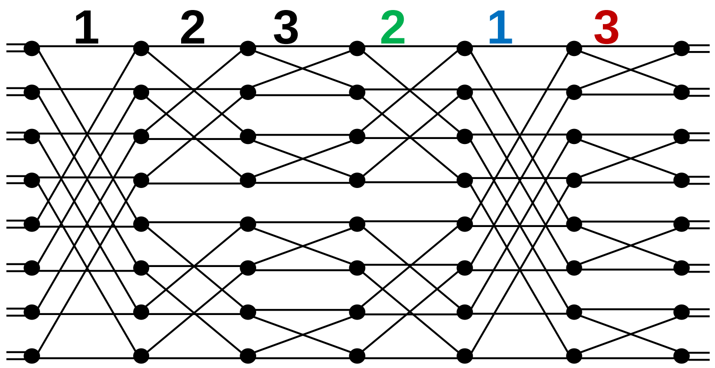
- In particular we can represent the first half of the MIN as a Butterfly and the second half by a permutation of Butterfly stages, that are log N -1, thus giving (log N – 1)! different ways to arrange them

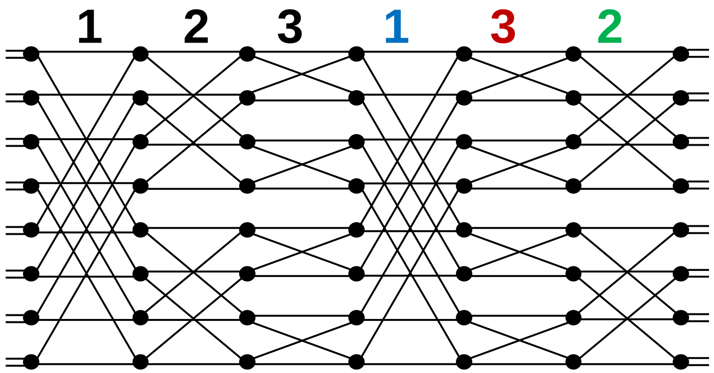
Classes for N=16

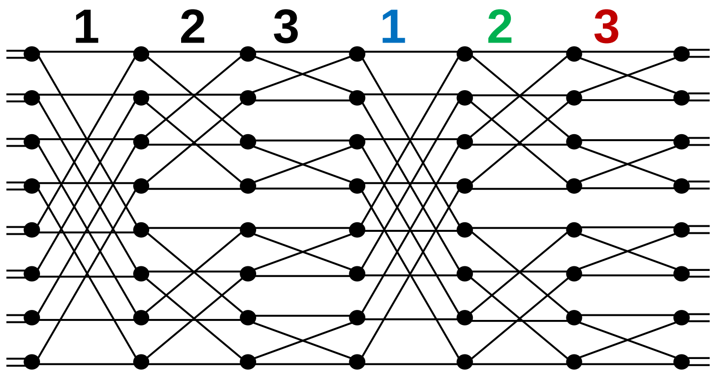












ALL-TO-ALL COMMUNICATION ON MINS

- A. Massini - All-to-all personalized communication on multistage interconnection networks - Discrete Applied Mathematics, 128, 435 - 446, 2003
- D. Izzi, A. Massini - Optimal all-to-all personalized communication on Butterfly networks through a reduced Latin square – Int. Conf. HPCC/DSS/SmartCity, 2020

All-to-all communication

- In **all-to-all communication** every processor in a processor group sends a message to all other processors in the group

- According to type of message to be sent, all-to-all communications can be classified in:
  - **all-to-all broadcast** communication - every processor sends the **same message** to all other processors
  - **all-to-all personalized** communication - each processor sends a **distinct message** to every other processor

- All-to-all personalized communication problem has been extensively studied for many networks topologies, in particular many results have been reported for meshes, tori and MINs

log N stage MINs

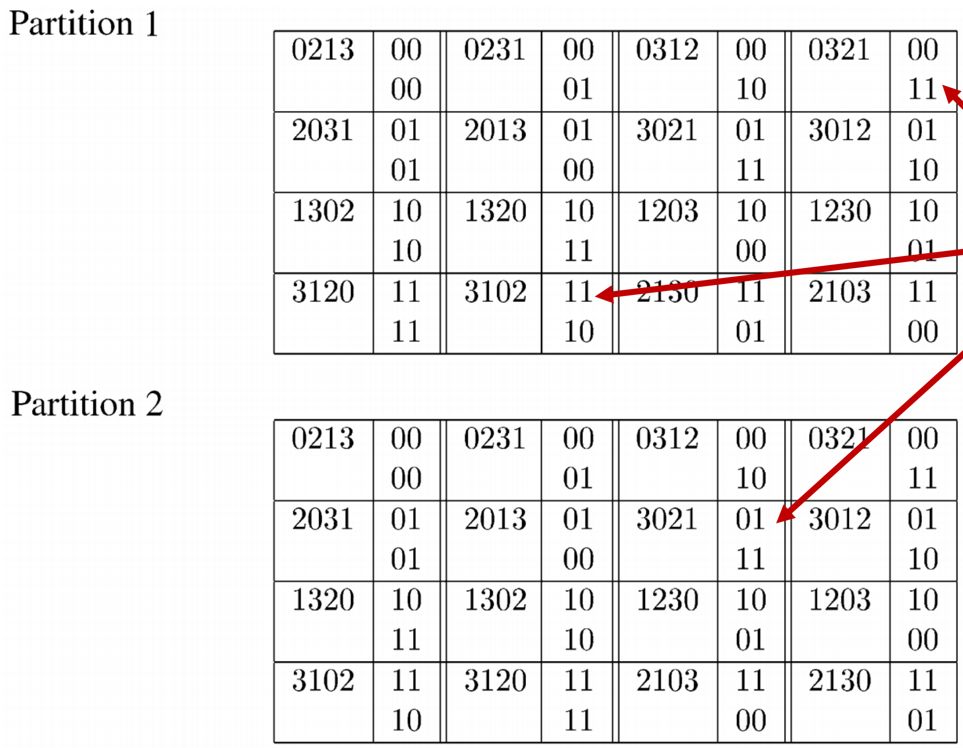
- Yang and Wang (2000)
  - developed a method for decomposing the all-to-all personalized exchange (ATAPE) patterns into some permutations which are realizable on *log N* **multistage networks**
  - presented an optimal algorithm with O(*N*) time complexity
- Namely, the all-to-all personalized communication is realized by routing a set of *N* permutations forming a **Latin Square**
- Yang and Wang's algorithm exploits the decomposition of an admissible permutation in basic permutations (interstage permutations and stage permutations)

log N stage MINs

- Despite Baseline, Omega and indirect binary-cube networks are equivalent, in Yang and Wang’s paper each network is discussed separately, and the **algorithm depends on the network topology**, being based on interstage permutations
- The Latin Square is generated by means of an **off-line algorithm** run at the time the network is built
- This realization is **specific** for the size of the network and provides only one particular Latin Square
- The requirement to realize the complete exchange with this method is to **keep in memory a matrix of size N × N** containing the destination tags for the N permutations

log N stage MINs

- Anyway, it is possible to do better
- We can prove that the set of **admissible permutations** for a MIN **can be partitioned in sets that are Latin Squares**, and the partitioning can be done in different ways



Partition 1

0213	00 00	0231	00 01	0312	00 10	0321	00 11
2031	01 01	2013	01 00	3021	01 11	3012	01 10
1302	10 10	1320	10 11	1203	10 00	1230	10 01
3120	11 11	3102	11 10	2130	11 01	2103	11 00

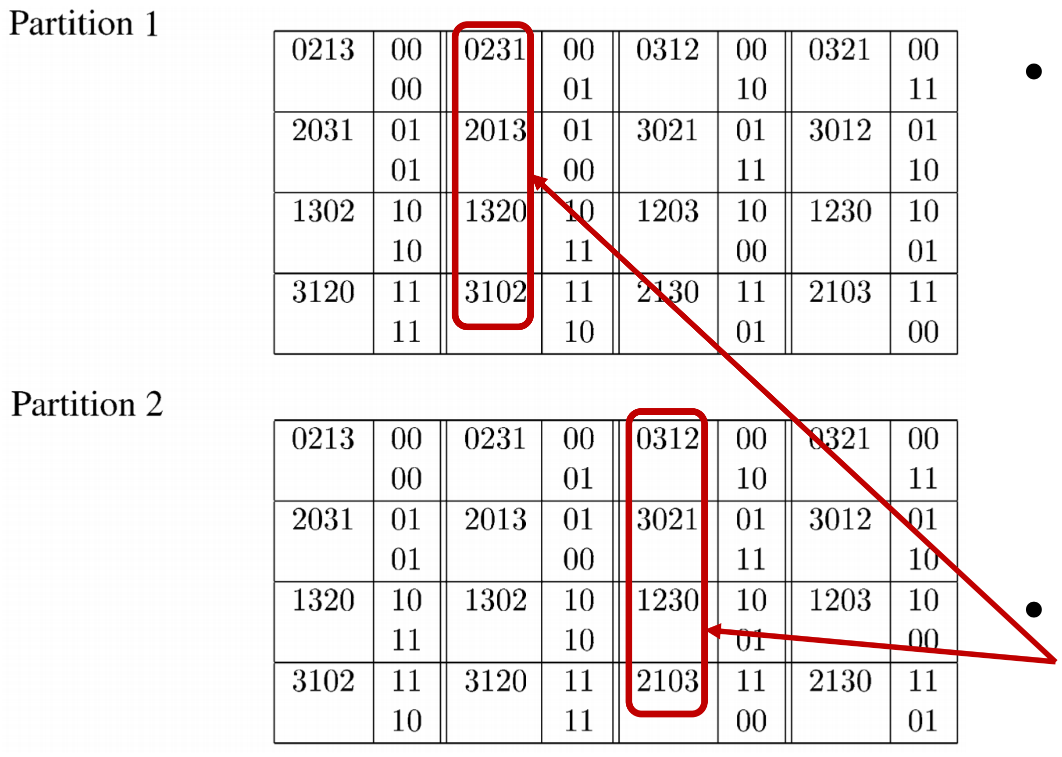
Partition 2

0213	00 00	0231	00 01	0312	00 10	0321	00 11
2031	01 01	2013	01 00	3021	01 11	3012	01 10
1320	10 11	1302	10 10	1230	10 01	1203	10 00
3102	11 10	3120	11 11	2103	11 00	2130	11 01

- In each cell of the tables, we see the binary matrix representing the network configuration (right) that produces the associated permutation (left)

log N stage MINs

- Anyway, it is possible to do better
- We can prove that the set of **admissible permutations** for a MIN **can be partitioned in sets that are Latin Squares**, and the partitioning can be done in different ways



Partition 1

0213	00 00	0231	00 01	0312	00 10	0321	00 11
2031	01 01	2013	01 00	3021	01 11	3012	01 10
1302	10 10	1320	10 11	1203	10 00	1230	10 01
3120	11 11	3102	11 10	2130	11 01	2103	11 00

Partition 2

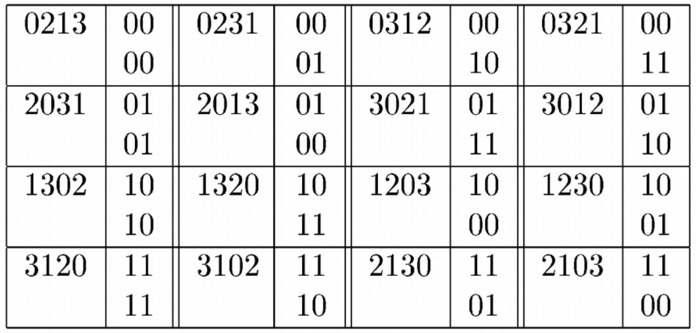
0213	00 00	0231	00 01	0312	00 10	0321	00 11
2031	01 01	2013	01 00	3021	01 11	3012	01 10
1320	10 11	1302	10 10	1230	10 01	1203	10 00
3102	11 10	3120	11 11	2103	11 00	2130	11 01

- In each cell of the tables, we see the binary matrix representing the network configuration (right) that produces the associated permutation (left)
- Each column of the two tables is a Latin Square

log N stage MINs

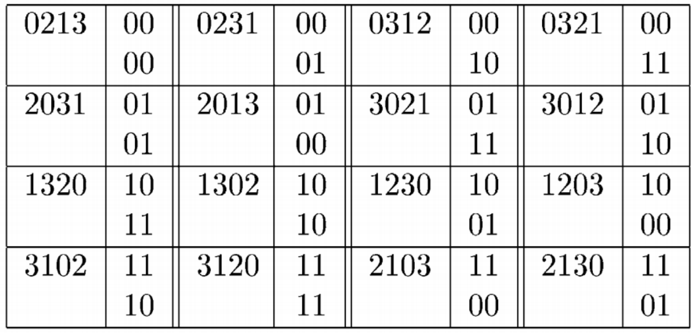
- Different partitions can be obtained for the set of admissible permutations of any size MINs
- The idea is to **start from any (admissible) permutation** and then **produce the other N-1 permutations of the Latin Square**

Partition 1



0213	00 00	0231	00 01	0312	00 10	0321	00 11
2031	01 01	2013	01 00	3021	01 11	3012	01 10
1302	10 10	1320	10 11	1203	10 00	1230	10 01
3120	11 11	3102	11 10	2130	11 01	2103	11 00

Partition 2



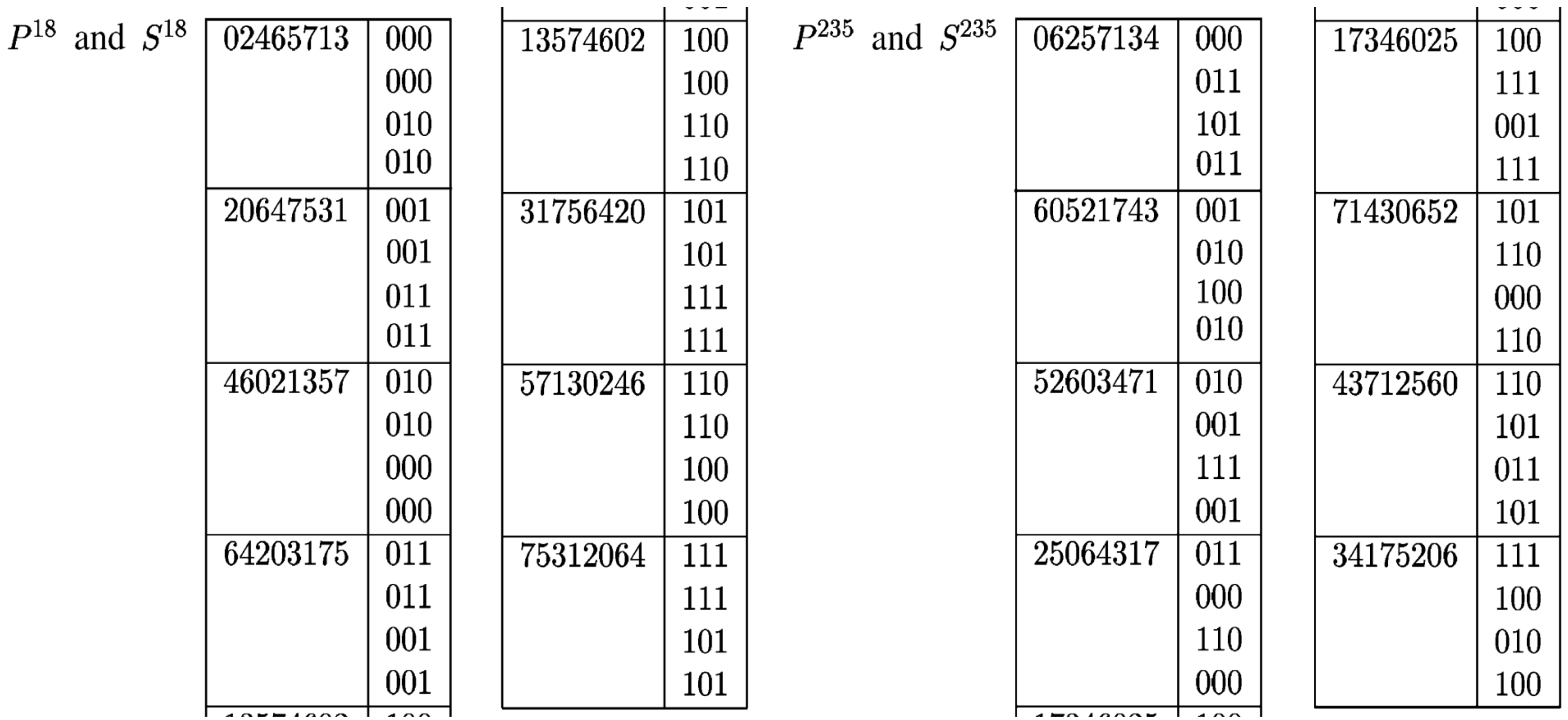
0213	00 00	0231	00 01	0312	00 10	0321	00 11
2031	01 01	2013	01 00	3021	01 11	3012	01 10
1320	10 11	1302	10 10	1230	10 01	1203	10 00
3102	11 10	3120	11 11	2103	11 00	2130	11 01

The simplest way to obtain a Latin Square is:

- Consider the switch configuration C_0 with **all switches set to straight**
- The **k-th switch configuration**, k=1, 2, …, N-1, is obtained using the **log N bits binary representation of k**
- i.e., **switch configuration k** is obtained from C_0 applying the **XOR row by row** with binary representation of k

log N stage MINs

- Two different **Latin Squares** and the corresponding **switch settings** for a Butterfly of size N=8



P^{18} and S^{18}

02465713	000	13574602	100
	000		100
	010		110
	010		110
20647531	001	31756420	101
	001		101
	011		111
	011		111
46021357	010	57130246	110
	010		110
	000		100
	000		100
64203175	011	75312064	111
	011		111
	001		101
	001		101

P^{235} and S^{235}

06257134	000	17346025	100
	011		111
	101		001
	011		111
60521743	001	71430652	101
	010		110
	100		000
	010		110
52603471	010	43712560	110
	001		101
	111		011
	001		101
25064317	011	34175206	111
	000		100
	110		010
	000		100

log N stage MINs

- This method **does not need a pre-computation** and **does not require to store the matrix of permutations to be realized**, because an explicit computation of permutations belonging to a Latin Square is not necessary

- With this method we can obtain a partition of the set P of admissible permutations for a MIN in sets P^l where $l = 0, \ldots, N^{(N-2)} - 1$, which can be proved being Latin Squares

- If we start from the switch configuration having all switches set to straight, the set of switch configurations and the corresponding Latin squares obtained applying the method are called ***canonical***, and correspond to Partition 1 of the example

2 log N – 1 stage MINs

- Unfortunately, log N stage MINs are not rearrangeable, and can realize only a subset of the N! possible permutations
- If a **particular Latin square** is desired, it can be realized using MINs either with more than log N stages or without the unique path property
- We take into consideration the **2 log N - 1 stage MIN** obtained concatenating **two Butterflies network**
- Note that **rearrangeability** is proved only for the Benes network (concatenation of a Baseline and a Reverse Baseline) through the looping algorithm, as well as for the equivalent networks
- However, double MIN networks have many useful properties and can be exploited for applications requiring specific sets of permutations

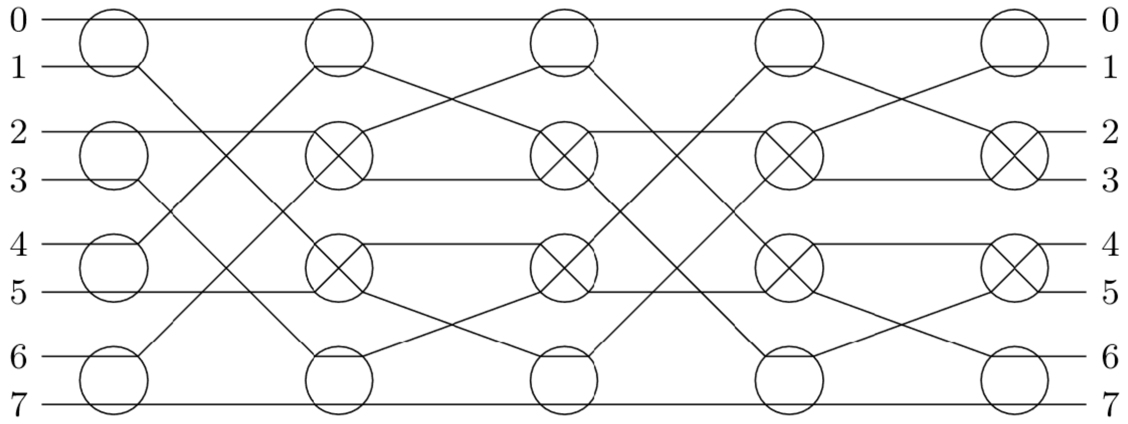
2 log N – 1 stage MINs

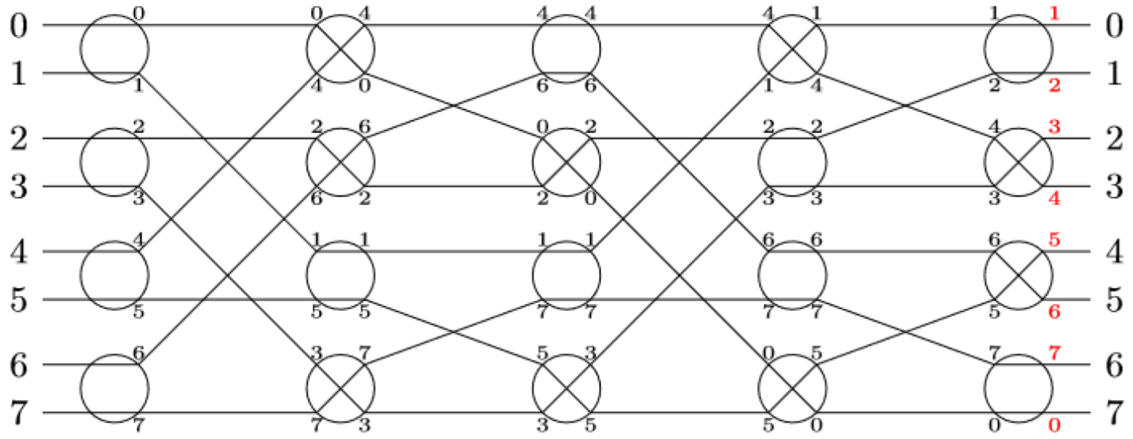
In the paper *Optimal all-to-all personalized communication on Butterfly networks through a reduced Latin square*

- We propose a method for all-to-all personalized communication based on the Latin square consisting of the **identity permutation and all N - 1 right rotations**, denoted Right Latin Square (RLS)
- We describe how to obtain the set of permutations belonging to RLS on any size Butterfly-Butterfly
- We formally prove the **feasibility of the set of permutation belonging to RLS on any size Butterfly-Butterfly**
- Our method can be easily utilized on the **flattened butterfly** network used in Data Centers

2 log N – 1 stage MINs

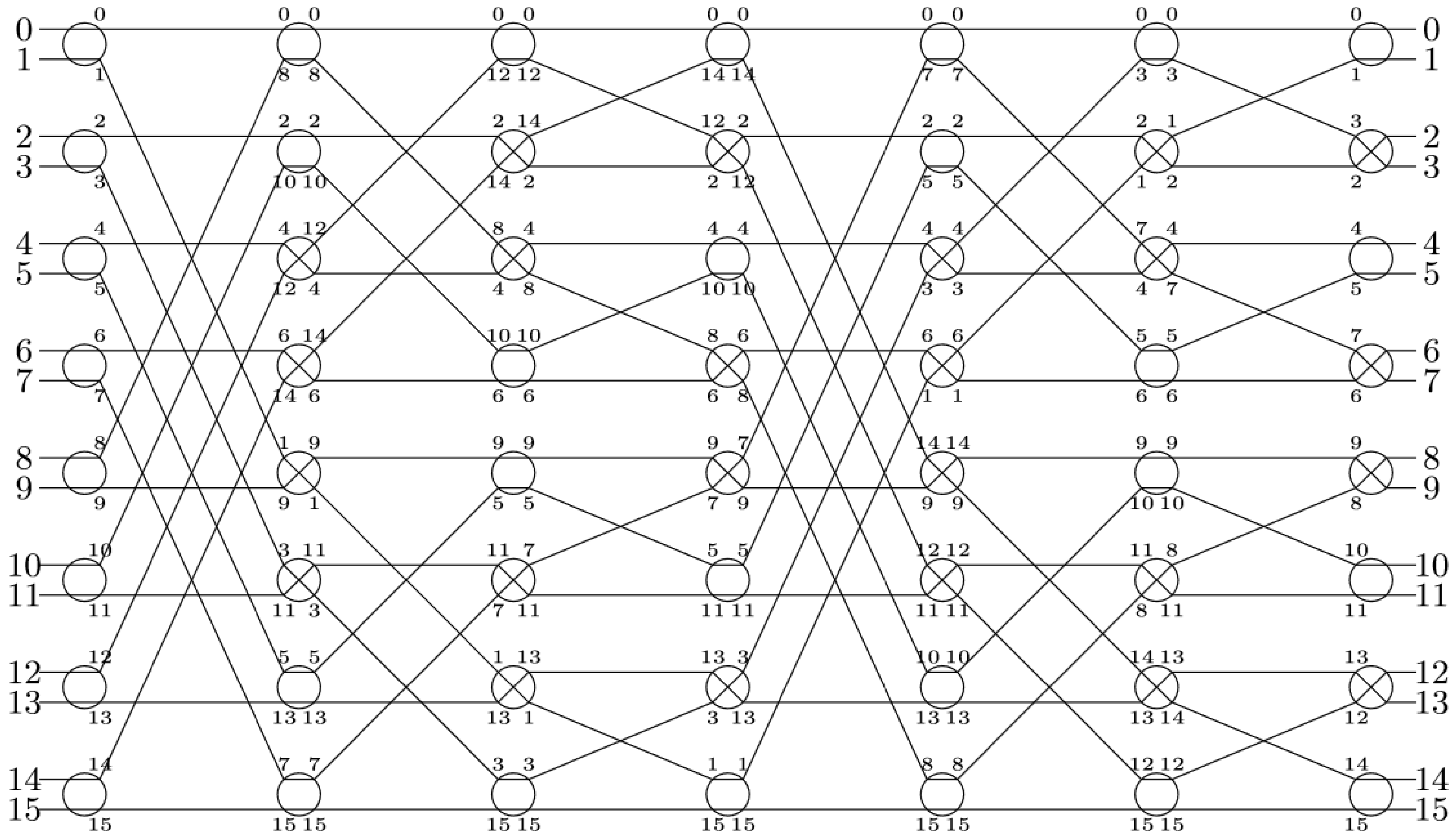
- The idea is to find the switch setting for the **identity** and the **first rotation**, and then derive all other - even and odd - rotations





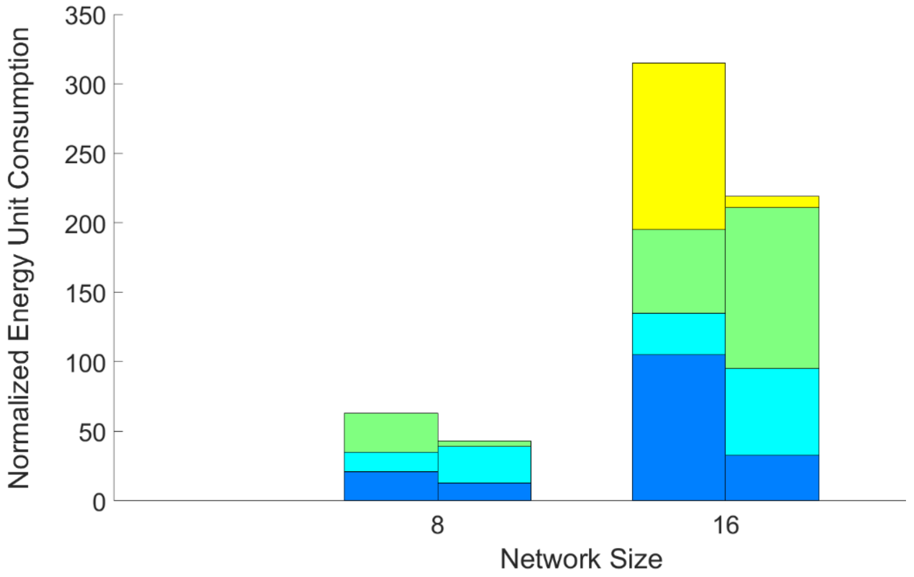
2 log N – 1 stage MINs

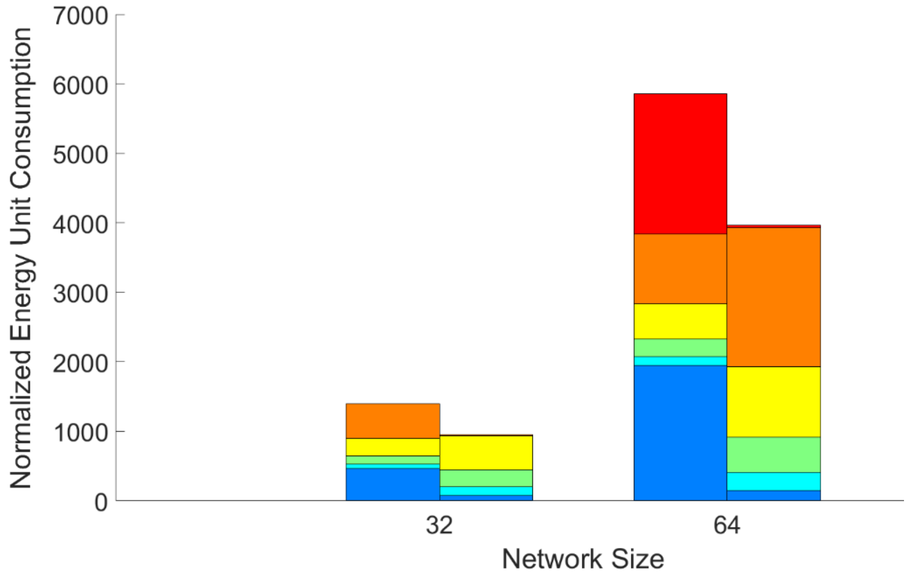
Switch setting for the identity on a **Butterfly-Butterfly** of size **N=16**



Energy consumption

- We measure the energy consumption to realize the Canonical Latin square counting the changes of the switch states
  - Namely when switches pass from straight to cross and vice versa
  - Changes are counted considering the rotations routed in pipeline fashion
  - We compare the cases when the Latin Square is realized in natural order (left) and when even and odd permutations are made separately (right)

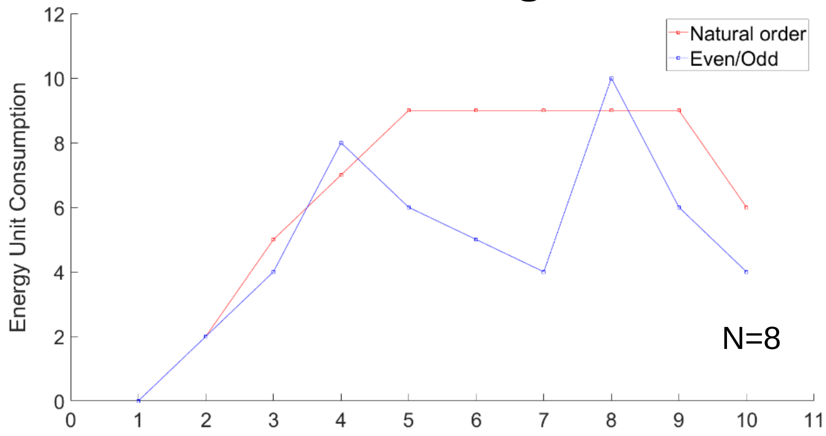


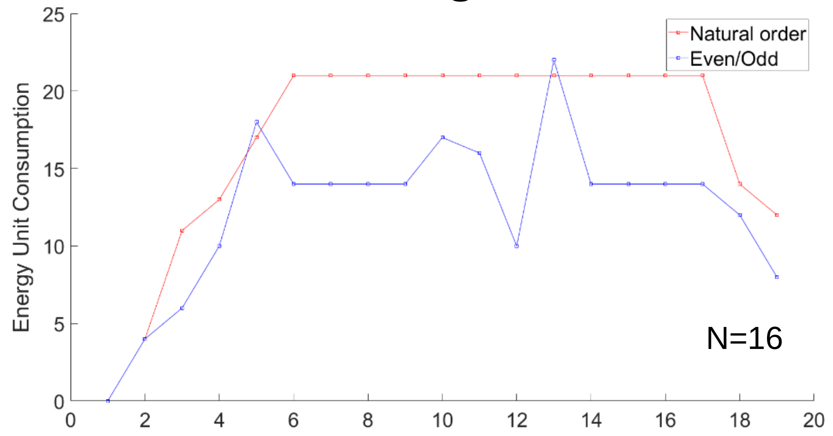


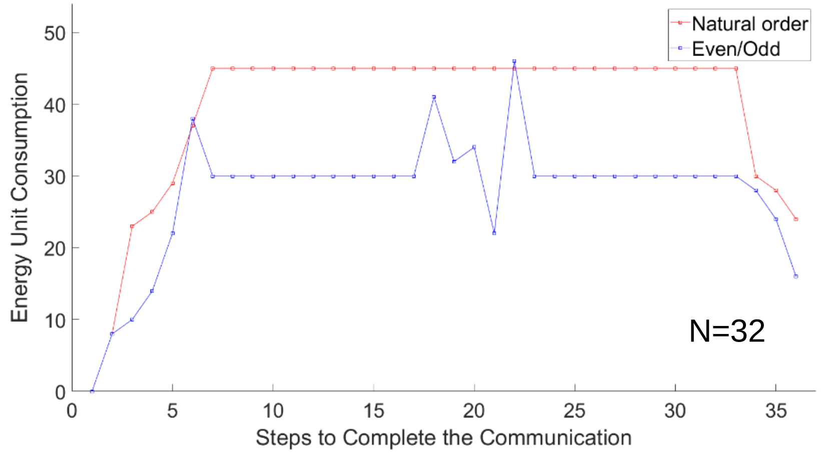
2 log N – 1 stage MINs

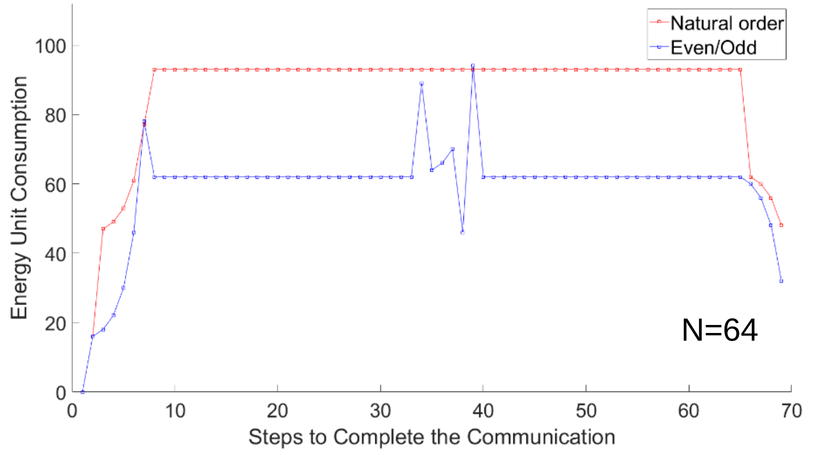
Variation of energy consumption for rotations routed in pipeline

- For example, during the step 3, the identity rotation is on stage 2, the first rotation is on stage 1 and the second rotation is on the stage 0









FAT TREES

S. R. Öhring, M. Ibel,Sajal K. Das, M. J. Kumar **On generalized fat trees**, Proc. 9th Int. Parallel Processing Symp., 1995

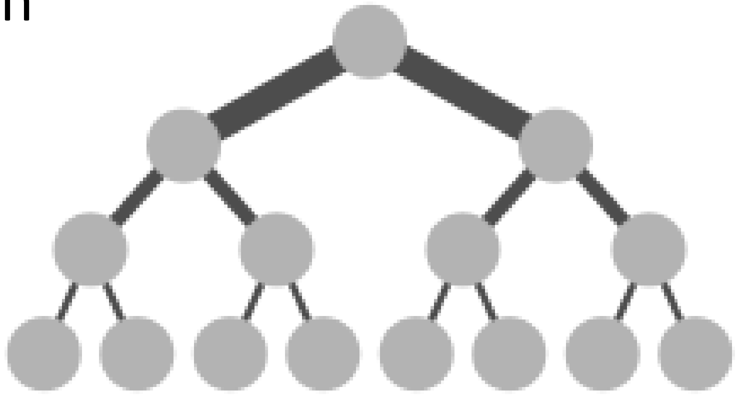
https://api.semanticscholar.org/CorpusID:26494144

B. Prisacari, G. Rodriguez, C. Minkenberg, T. Hoefler, **Fast Pattern-Specific Routing for Fat Tree Networks**, ACM Trans. Archit. Code Optim., 2013

https://api.semanticscholar.org/CorpusID:14743429

Fat trees

- The **fat tree network** is a universal network for efficient communicationinvented by Charles E. Leiserson of the Massachusetts Institute of Technology in 1985
- In a fat tree, the links get **fatter** (thicker) towards the top of the tree, and switch in the root of the tree has most links compared to any other switch

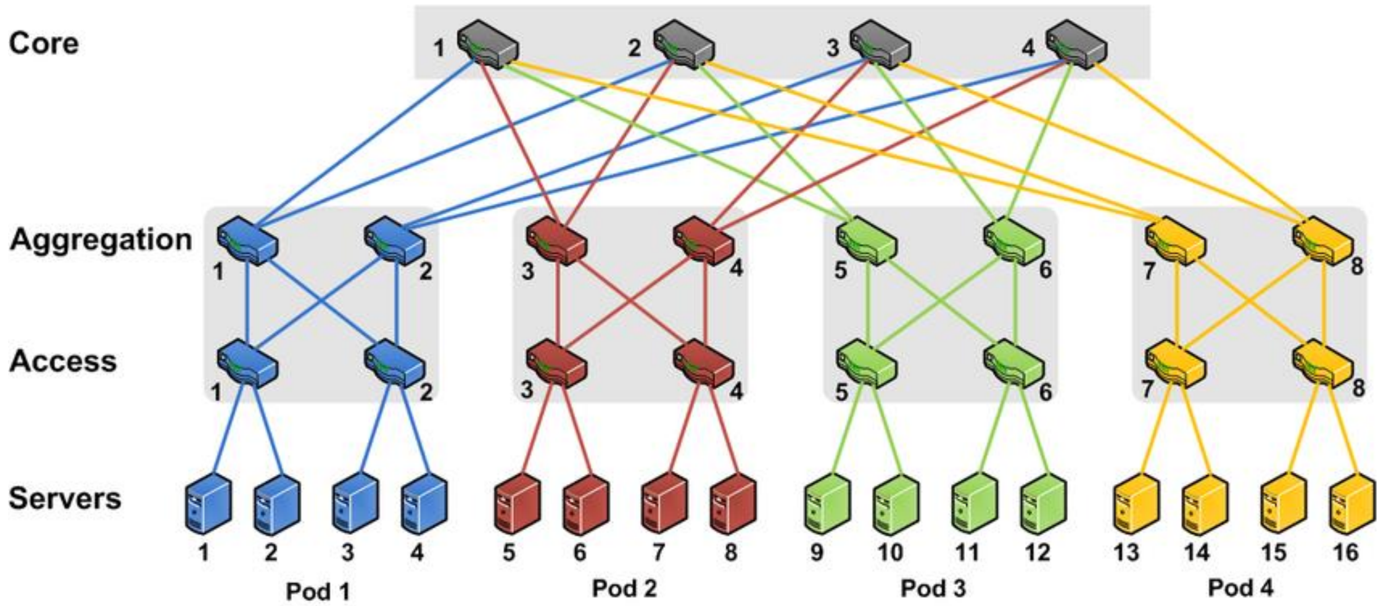


- k-ary n-trees, the type of fat-trees commonly used in most high-performance networks, where initially formalized in 1997

Fat trees

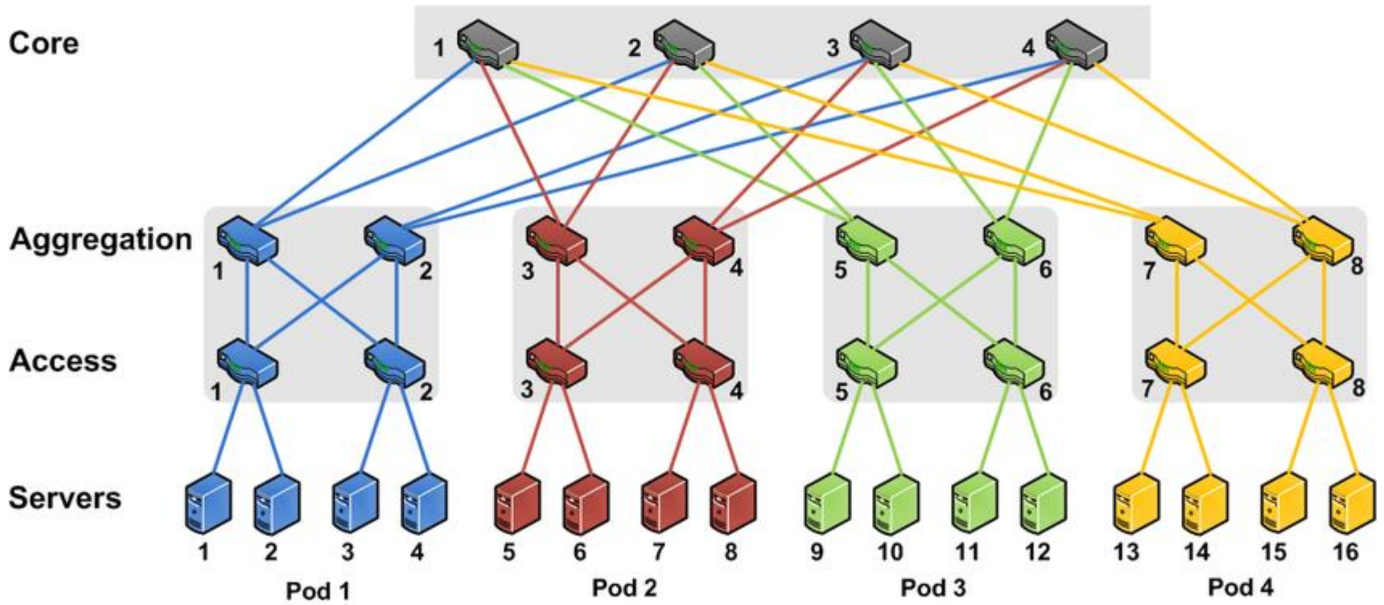
In the fat tree structure topology

- Leaf nodes are physical (heterogeneous) servers or computers
- All other nodes are switches arranged in layers
  - The top layer (level-0) of switches is called **Core layer**
  - The second layer of switches is called **Aggregation layer**
  - The third layer of switches is called **Edge (or Access) Layer**



Fat trees

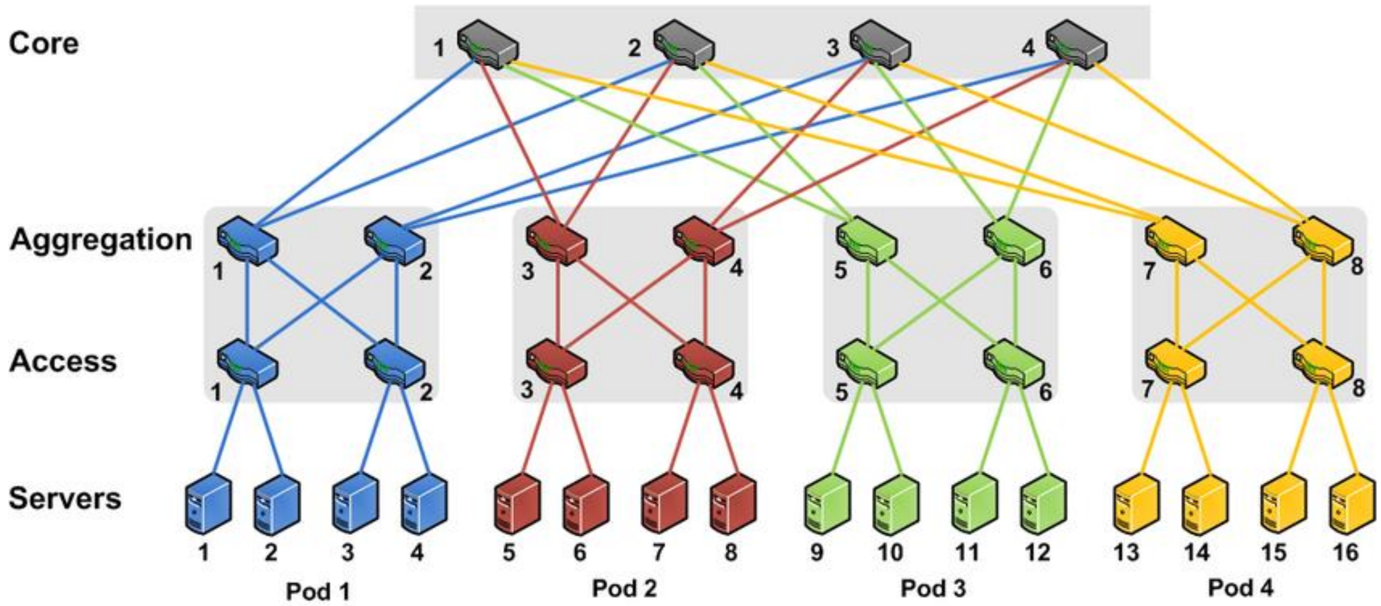
- There is a controversy whether such topologies should be called **fat-trees** or rather **(folded) Clos networks**
- However, the term fat-tree is widely used
- Fat trees are very versatile and are used both in Data Center and as On-Chip-Networks



Fat trees

Three-layer topology of **the k-ary fat tree**

- each **pod** consists of $(k/2)^2$ servers and 2 layers of k/2 k-port switches
- each **edge switch** connects to k/2 servers and k/2 aggregation switches
- each **aggregation switch** connects to k/2 edge and k/2 core switches
- $(k/2)^2$ **core switches**: each connects to k pods



Generalized Fat Tree

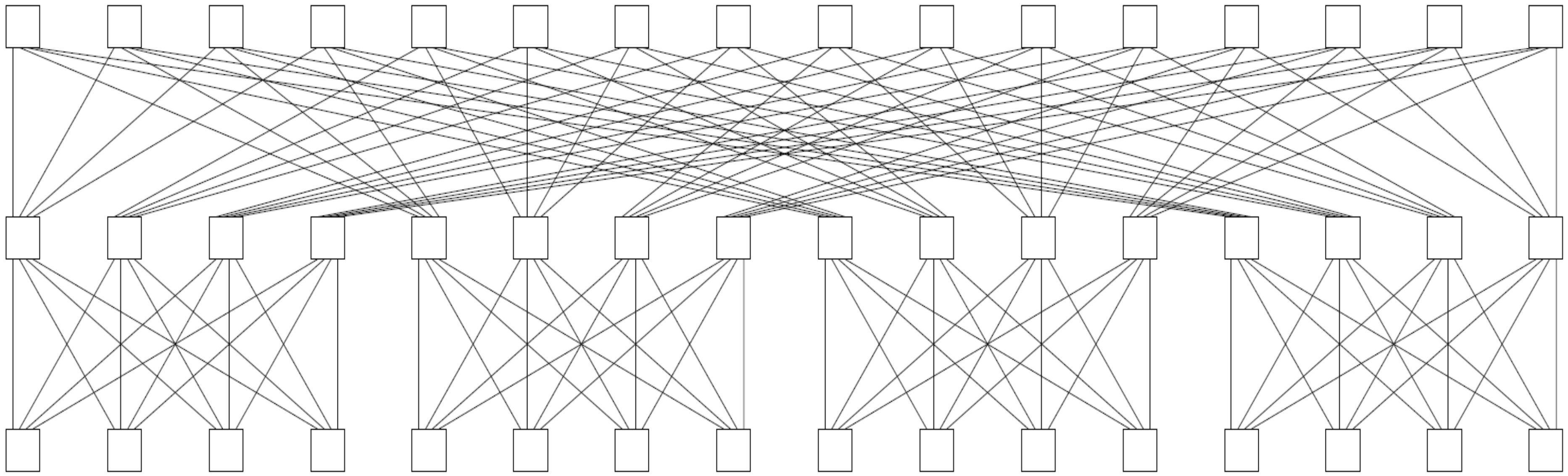
- A Generalized Fat-Tree (GFT), or a k–ary Fat-Tree, is an undirected graph $GFT(h, m, w)$
  - h represents the height of the tree
  - w^h is the number of root nodes (level h)
  - m^h is the number of nodes on level 1
  - m^{h+1} is the number of leaves (level 0)
  - Non root nodes have w parents
  - (Non leaves) nodes have m children
  - m is the number of identical copies of sub-trees at each step

  - Notice that some definitions use the height h differently

Fat trees

Example - Generalized Fat-Tree $GFT(h,m,w) = GFT(3,4,4)$

- $h = 3$ height of the tree
- $m = 4$ number of identical copies $GFT(2,4,4)$
- $w^h = 16$ number of root nodes (top level) and $m^{h+1} = 64$ leaves (not shown)
- $w = 4$ parents for non root nodes and $m = 4$ children for non leaves nodes



Generalized Fat Tree

- A Generalized Fat-Tree $GFT(h+1,m,w)$ can be **recursively** built with m copies GFT^{j} of $GFT(h,m,w)$
  - where $0 \leq j \leq m-1$
  - w^{h} nodes are added on top of the copies, labeled as $(h,k+j \cdot w^{h})$ for $0 \leq k \leq w^{h}-1$
  - the set of nodes is $V_h = \{(l,1) | 0 \leq l \leq h \land 0 \leq i \leq m^{h-1} \cdot w^{l}-1\}$
  - there exist an edge between two nodes labeled as (h,a) and $(h+1,b)$ belonging to $GFT(h,m,w)$ and $GFT(h+1,m,w)$ respectively, if and only if $a \bmod w^{h} = \left\lfloor \frac{b}{w} \right\rfloor$

Generalized Fat Tree

An Extended Generalized Fat-Tree (XGFT) is an undirected graph $XGFT(h; m_1, \dots, m_h; w_1, \dots, w_h)$

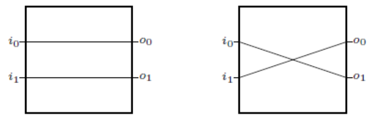
- h represents the height of the tree
- $m_1 \cdot m_2 \cdots m_h$ number of leaves
- $w_1 \cdot w_2 \cdots w_h$ number of roots
- Nodes on level i are $m_{i+1} \cdots m_h \cdot w_1 \cdots wi$
- Non root nodes in level i have w_{i+1} parents
- Non leaves nodes in level i have $m_{\ i}$ children
- From level i to level 1 we have m_{i+1} copies of $XGFT(i; m_1, \dots, m_i; w_1, \dots, w_i)$

- **Exercise** Draw the XGFT(2; 4, 4; 1, 2)

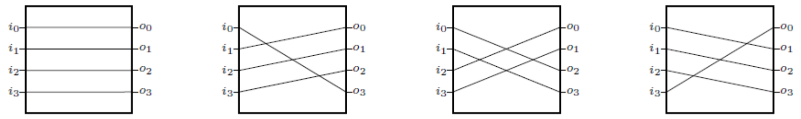
Fat trees

- The **all-to-all personalized communication** can be realized on a GFT using the same approach used for the Butterfly-Butterfly
- To this end, it is necessary to define the switch configurations needed for the realization of the Latin Square
- Let us assume that $m = w$, namely a complete Fat-Tree with the same number of switches in any stage
- Switches have size $m \times m$, and can assume $m!$ possible states, but a subset of m states suffices

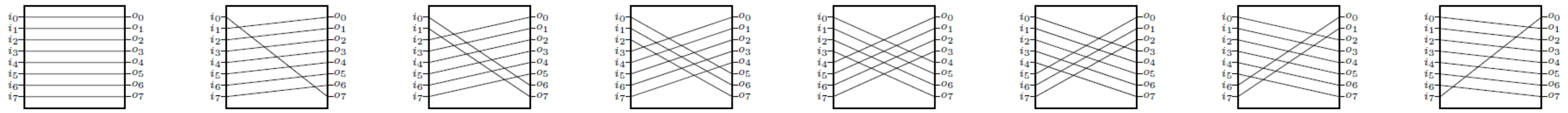
$m = 2$



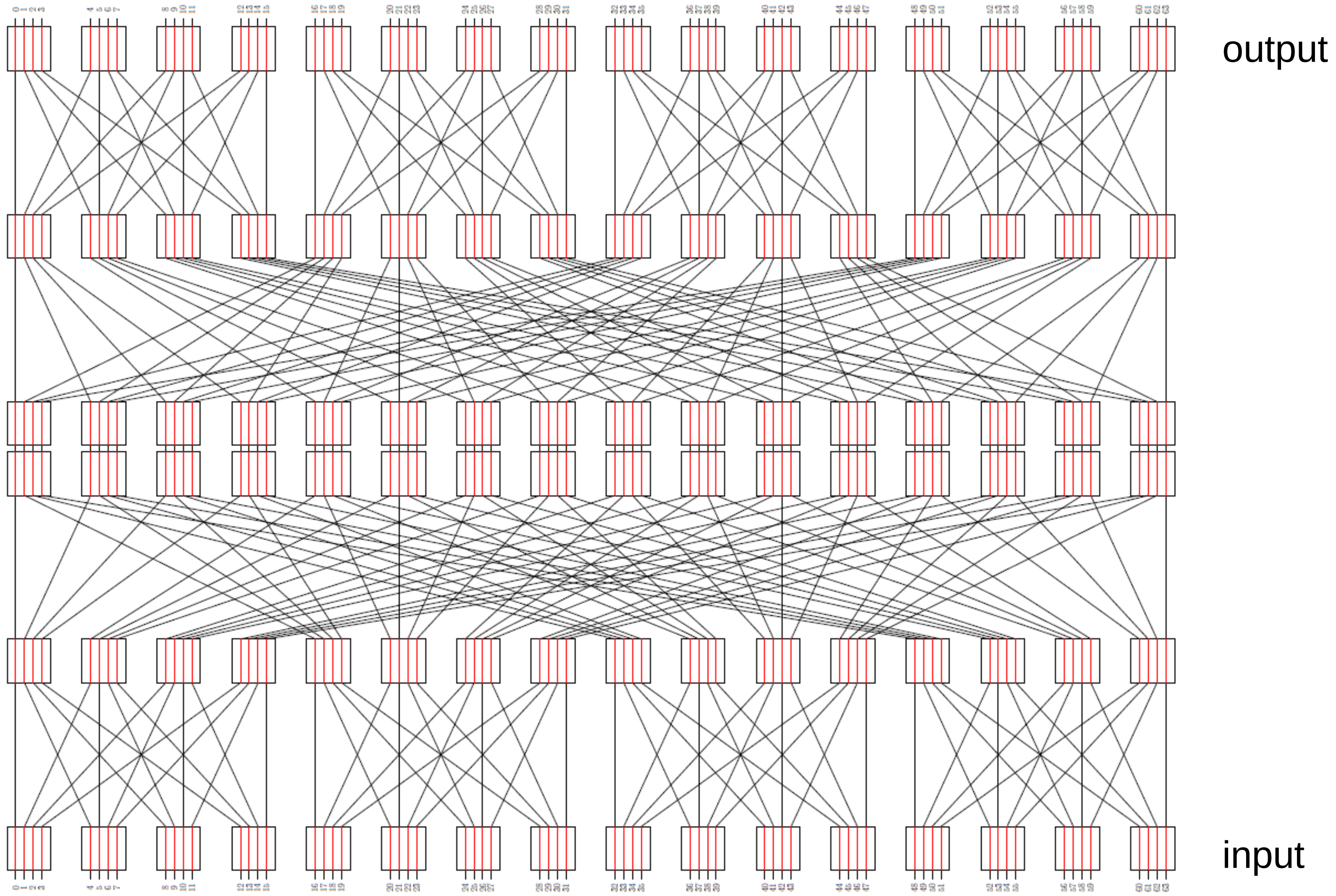
$m = 4$



$m = 8$

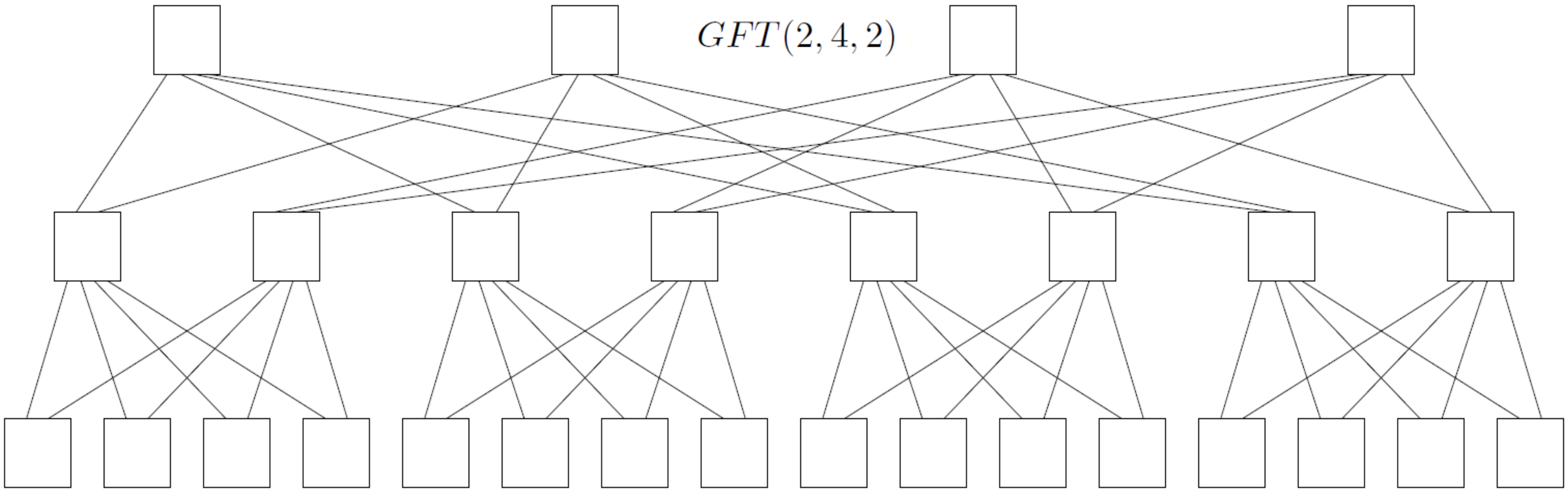


Identity permutation on Fat trees



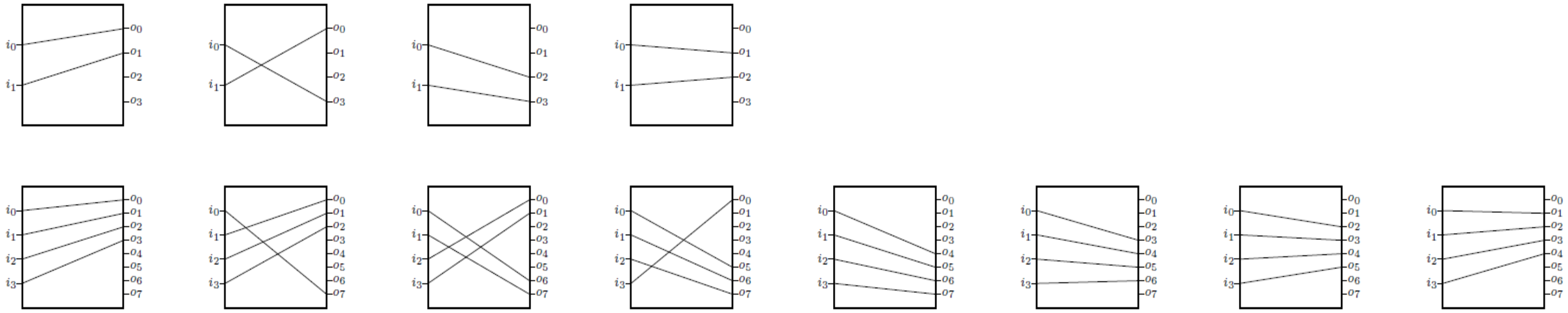
Slimmed Fat trees

- Anyway, a complete Fat-Tree is expensive and Data Centers usually employ cheaper networks
- An interesting variation is represented by **Slimmed Fat-Trees** where $\boldsymbol{m > w}$
- The number of nodes is different at each layer and the number of edges does not allow the contemporary route of every input

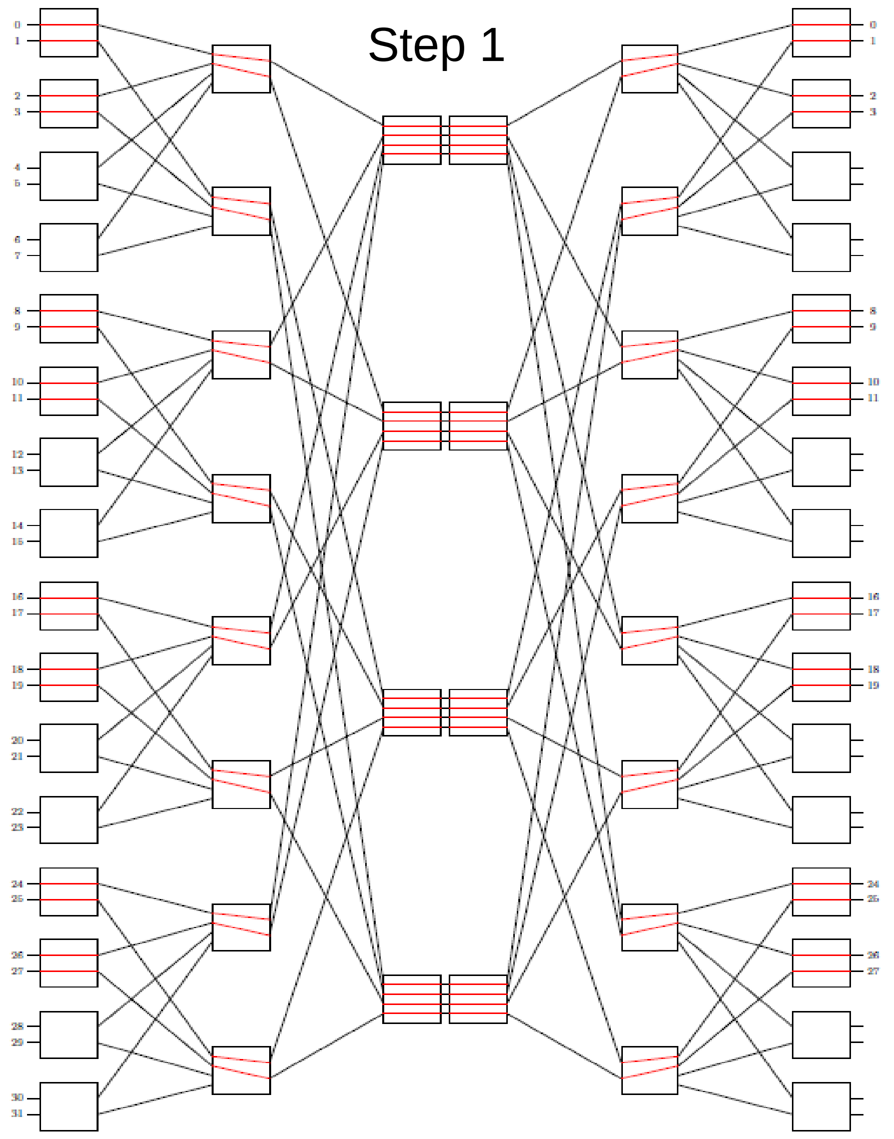


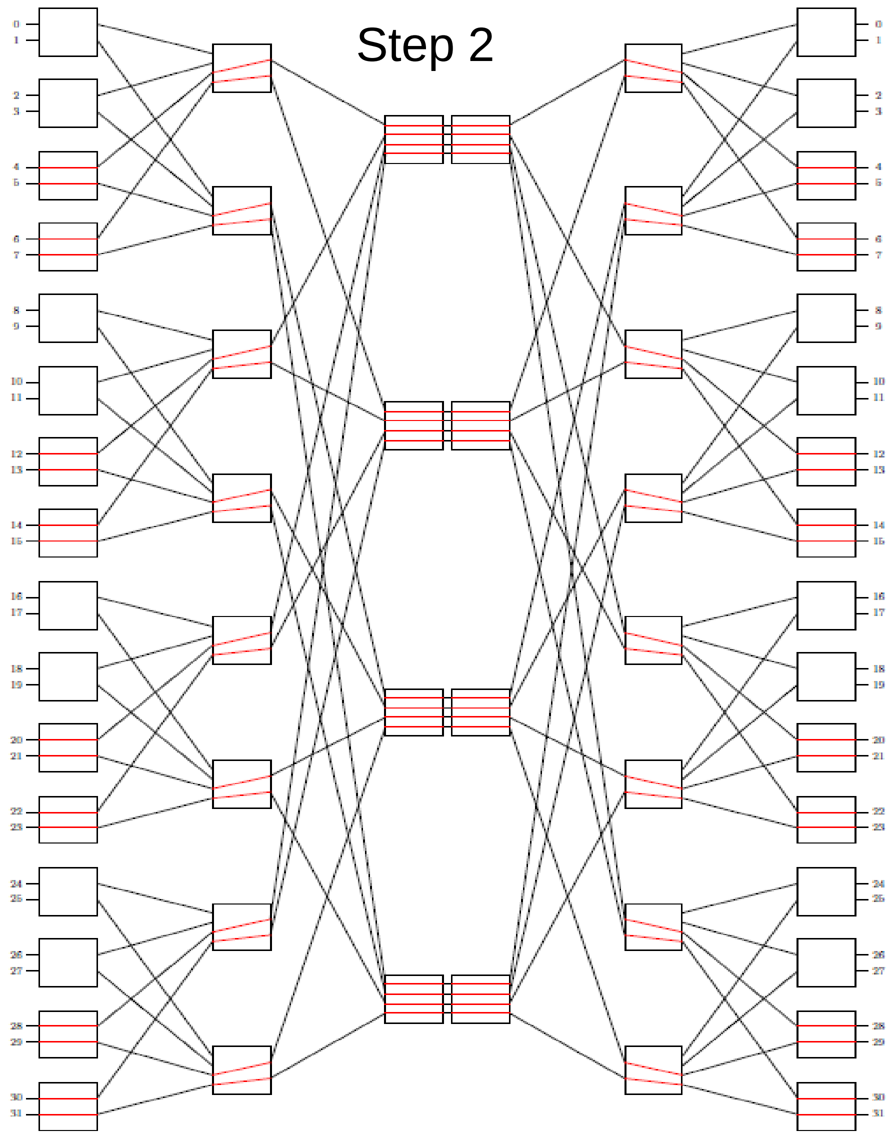
Slimmed Fat trees

- In this case, the **all-to-all communication** can be realized in more steps, routing subsets of inputs, namely **in $\left\lfloor \frac{m}{w} \right\rfloor$ steps**
- For switches of size $m \times w$, we have to define the **switch configurations needed**



Identity on Slimmed Fat trees for ATAPE


Step 1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31


Step 2
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31