Advanced Parallel Architecture

Annalisa Massini - 2016/2017

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-rime signal processing and control* – Ch. 2 *Parallel Architectures*
  M. O. Tokhi, M. A. Hossain, M. H. Shaheed, Springer, 2003

Interconnection Networks

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:
  - **Network diameter** Max. number of hops necessary to link up two most distant processors
  - **Maximum-Degree of PEs** max number of links to/from one PE
  - **Minimum-Degree of PEs** min number of links to/from one PE
  - **Network bisection width** Minimum number of links to be cut for a network to be into two halves
  - **Symmetry** The network looks the same from any node
  - **Scalability** The network is expandable with scalable performance when the machine resources are increased

Network Topology: Evolution

- One switch suffices to connect a small number of devices
  - Number of switch ports limited by VLSI technology, power consumption, packaging, and other such *cost* constraints
- A ***fabric*** of interconnected switches (i.e., *switch fabric* or *network fabric*) is needed when the number of devices is much larger
  - The *topology* must make a path(s) available for every pair of devices

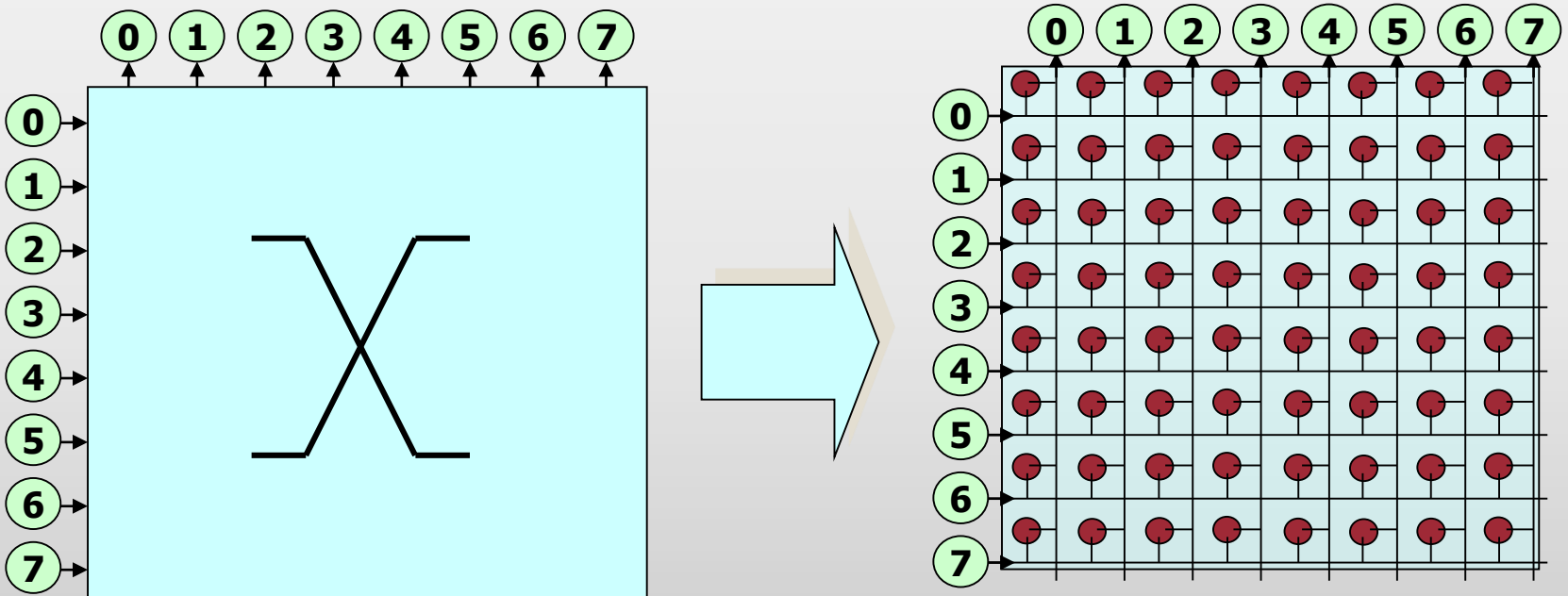
Network Topology: Evolution

- Several tens of topologies proposed, but less than a dozen used

- 1970s and 1980s
  - Topologies were proposed to reduce *hop count*
- 1990s
  - Pipelined transmission and switching techniques
  - Packet latency became decoupled from hop count
- 2000s
  - Topology still important (especially OCNs, SANs) when *N* is high
  - Topology impacts performance and has a major impact on cost

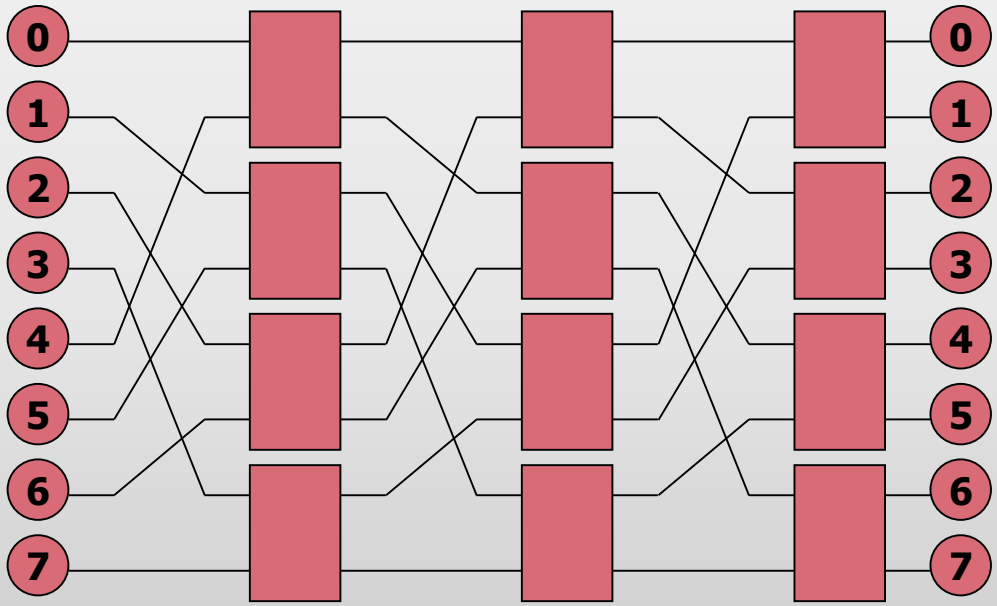
Network Topology

- ***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

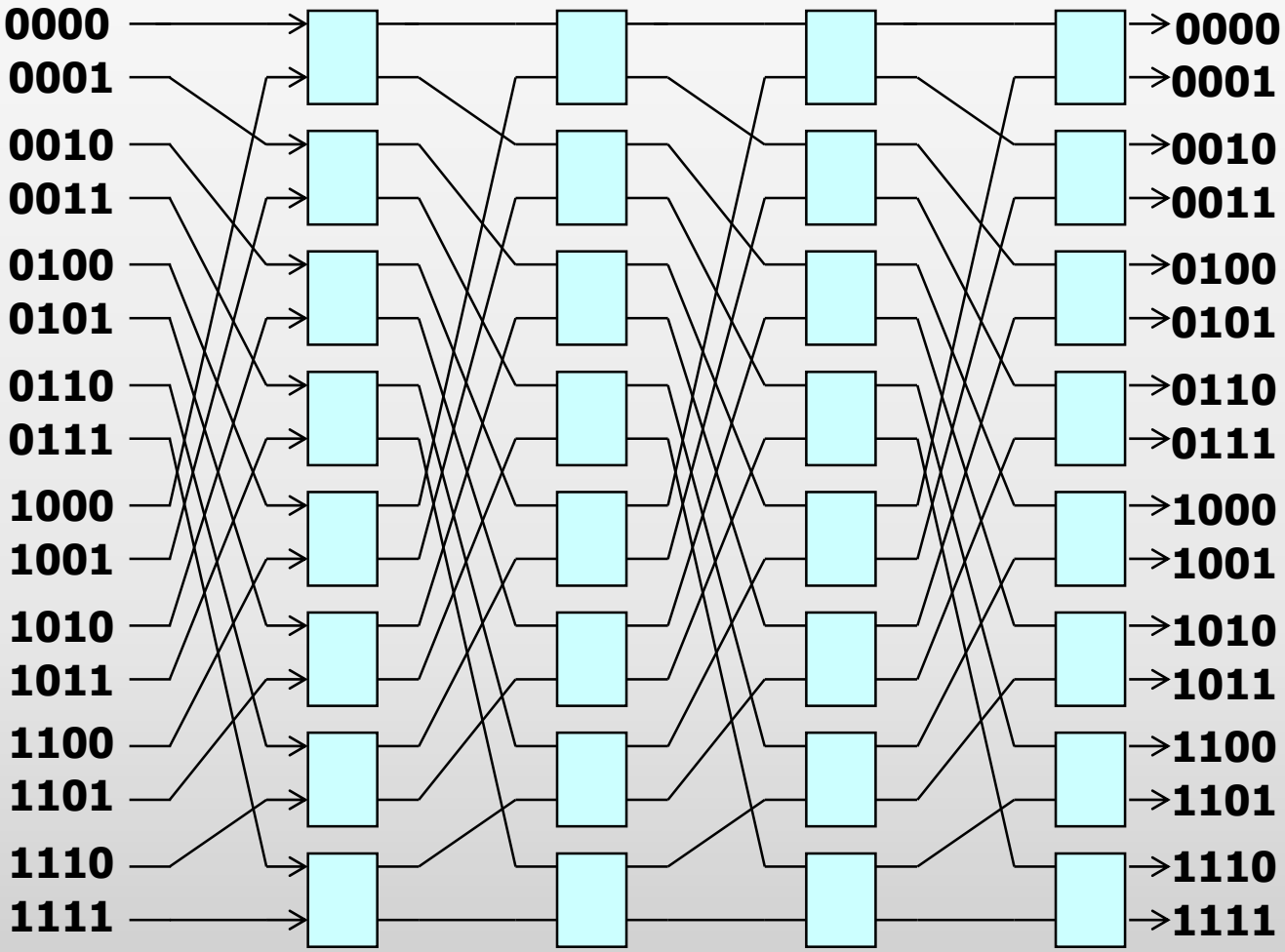
- ***Multistage interconnection networks (MINs)***
  - Crossbar split into several stages consisting of smaller crossbars
  - Complexity grows as O($N \times \log N$), where N is # of end nodes
  - Inter-stage connections represented by a set of permutation functions



Omega topology, perfect-shuffle exchange

Network Topology

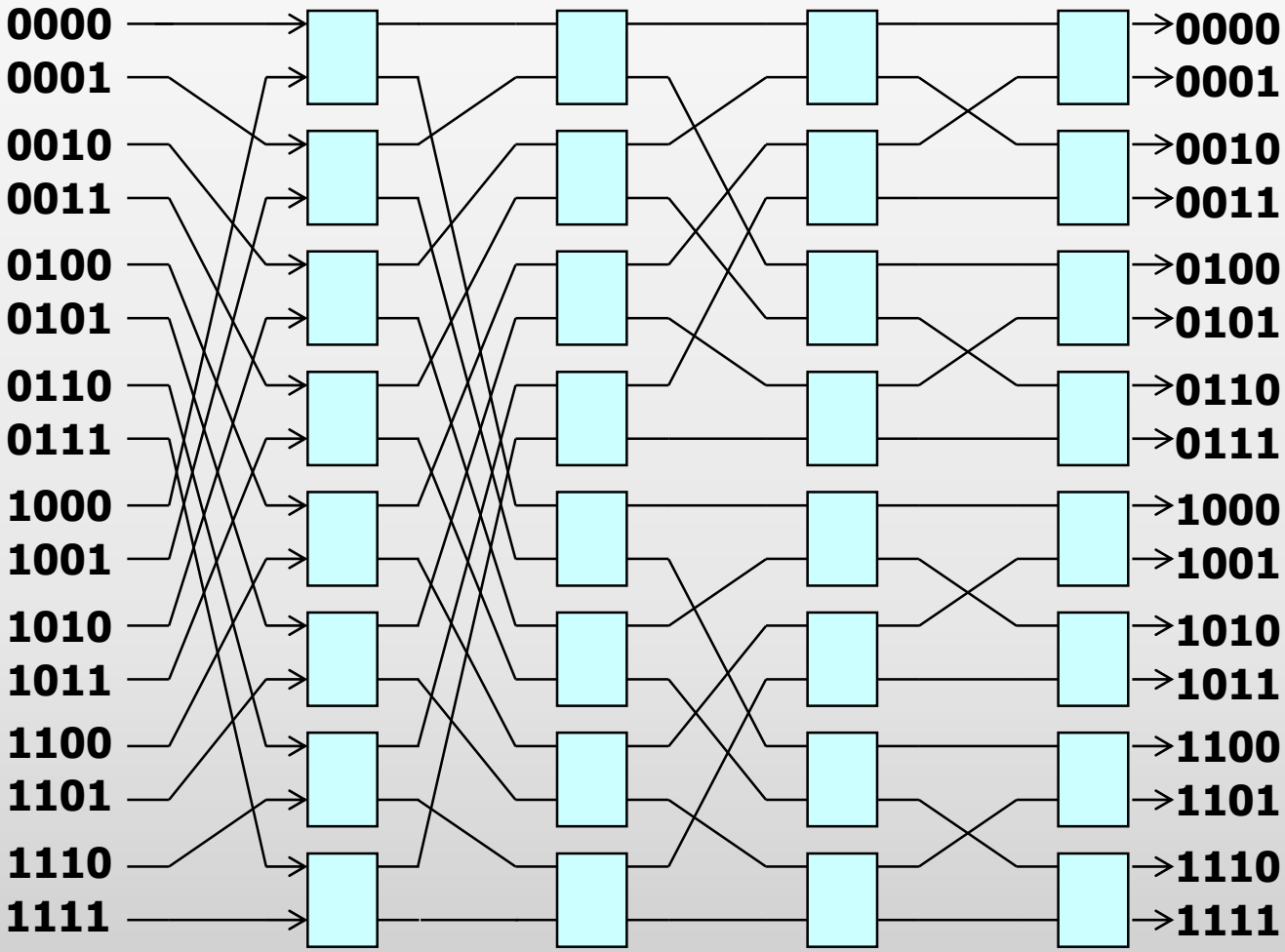
- ***Multistage interconnection networks (MINs)***



4 stage ***Omega*** *network*

Network Topology

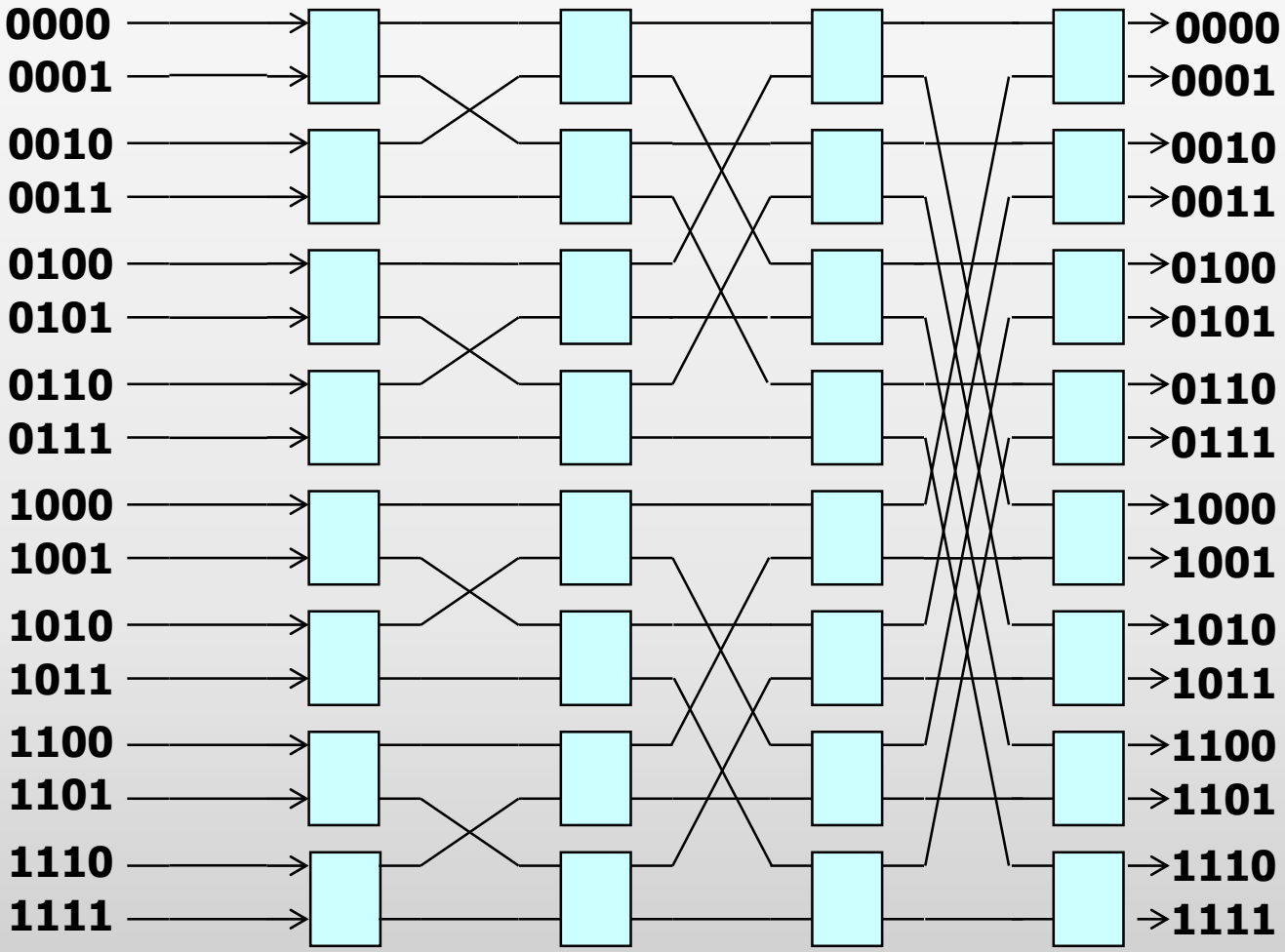
- ***Multistage interconnection networks (MINs)***



4 stage ***Baseline*** *network*

Network Topology

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



4 stage ***Reverse Butterfly*** *network*

Network Topology

- Multistage interconnection networks (MINs)
  - MINs interconnect *N* input/output ports using *k* x *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 x 2 switches
    - MIN with 4 x 4 switches
    - MIN with 16 x 16 switches

Network Topology

- Multistage interconnection networks (MINs)
  - ***Example:*** Compute the switch and link costs N=4096 nodes

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

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

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

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

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

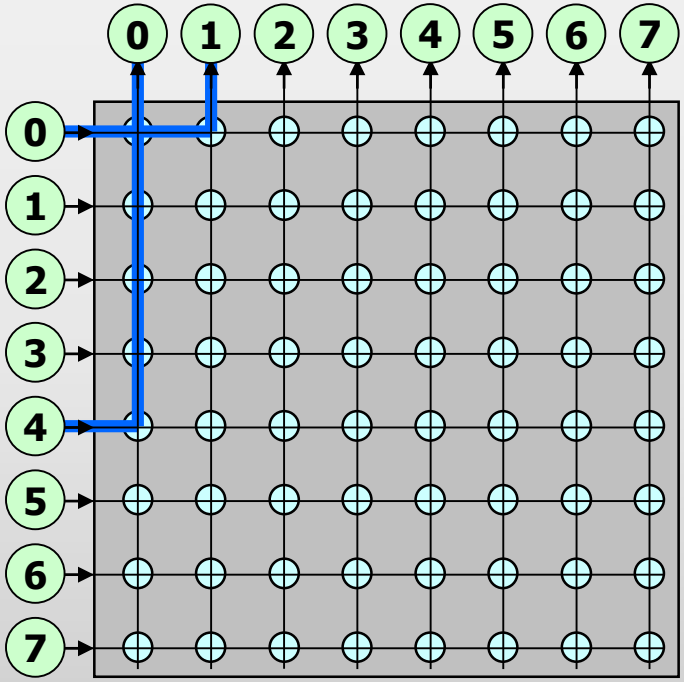
$$relative\_cost(4 \times 4)_{links} = 8192 / (4096 \times (\log_4 4096 + 1)) = 2/7 = 0.2857$$

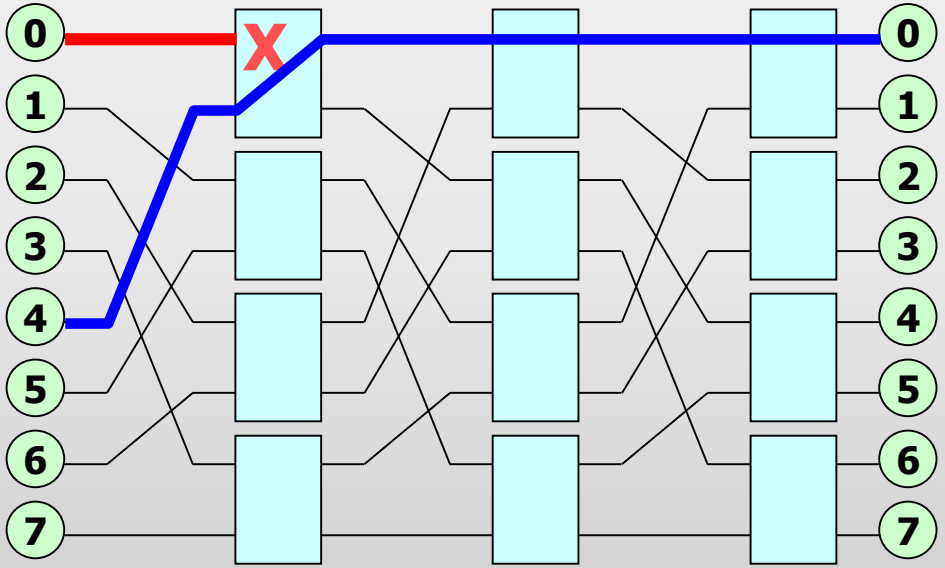
$$relative\_cost(16 \times 16)_{switches} = 4096^2 / (16^2 \times 4096/16 \times \log_{16} 4096) = 85$$

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

Network Topology

- Reduction in MIN switch *cost* → *performance* reduction
  - Network has the property of being *blocking*
  - Paths from different sources to different destinations share one or more links





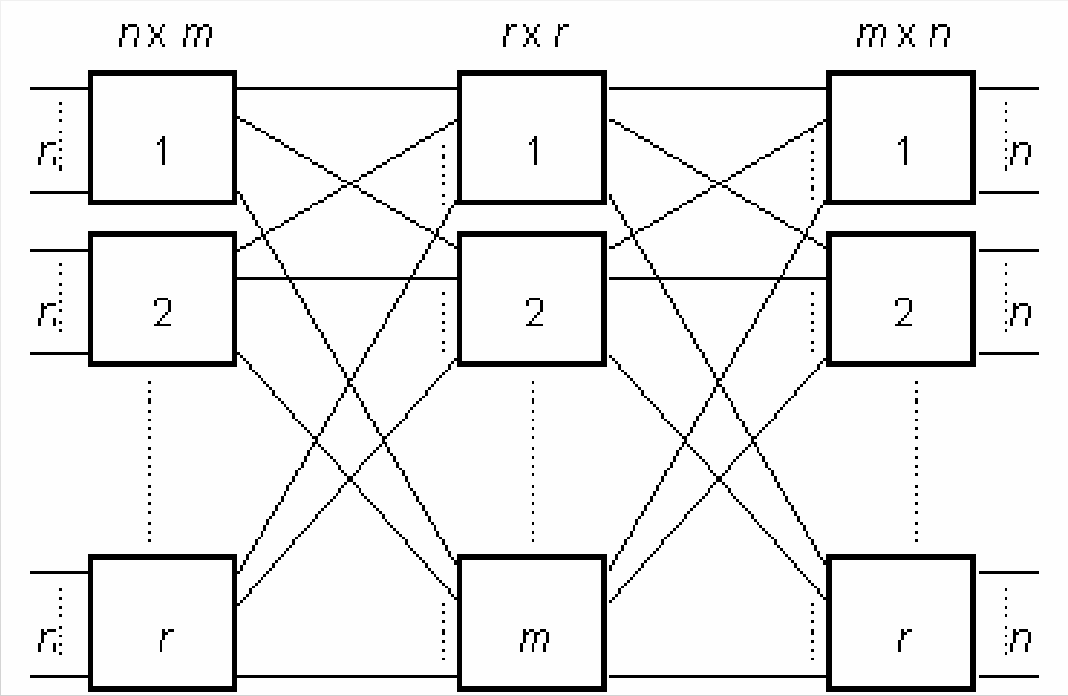
Network Topology

- 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 (rearrangeably 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

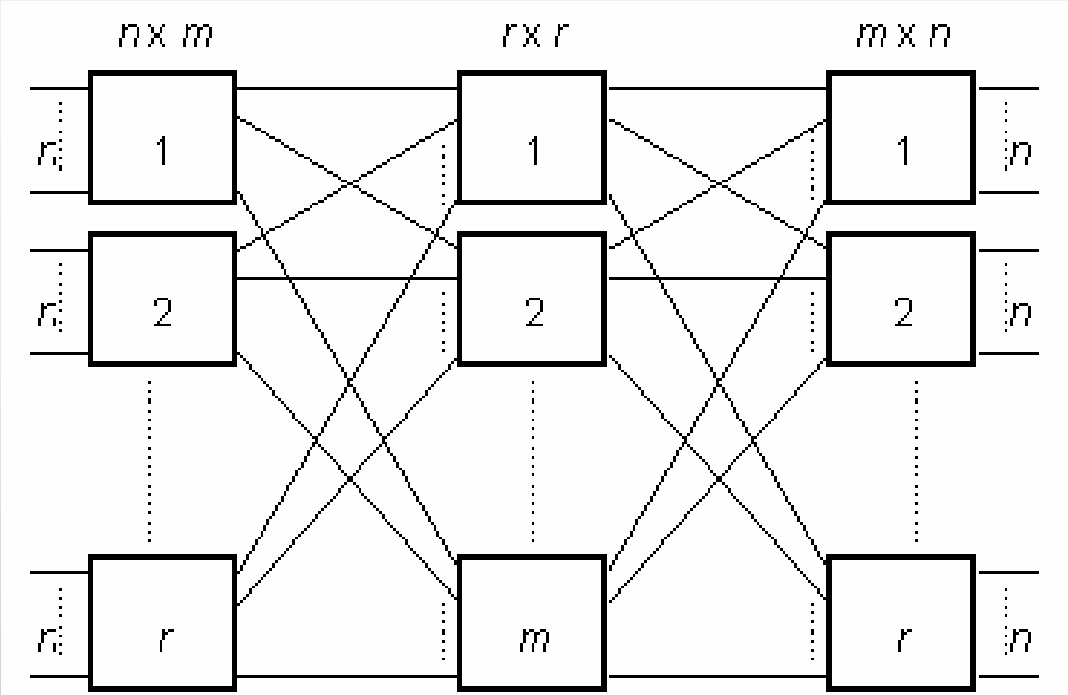
Clos network

- 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



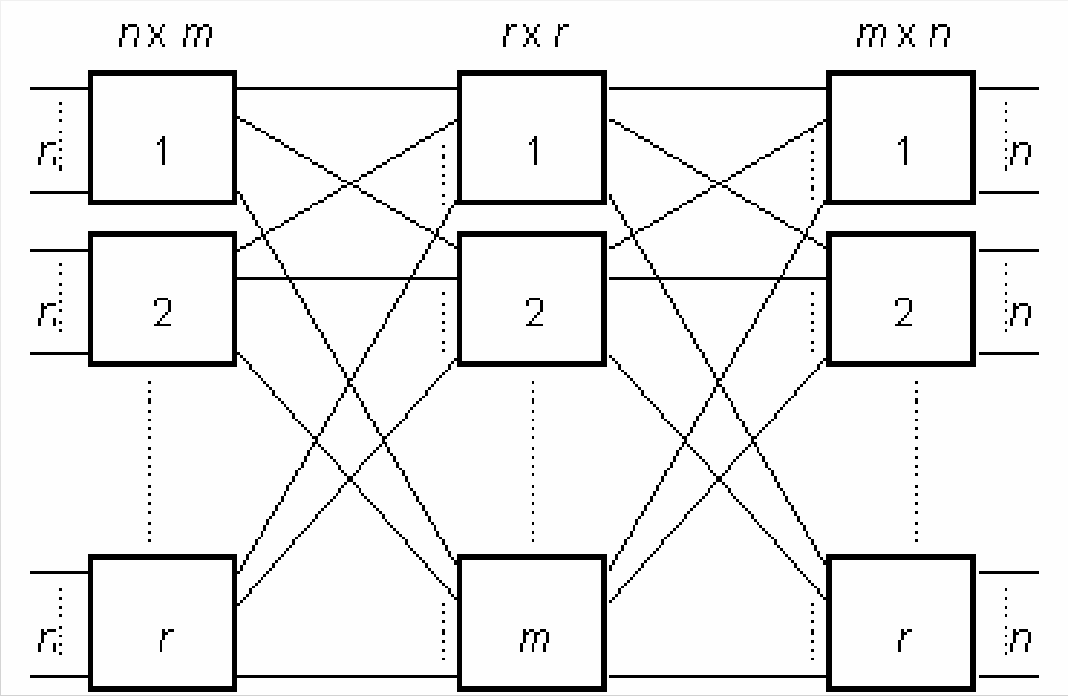
Clos network

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



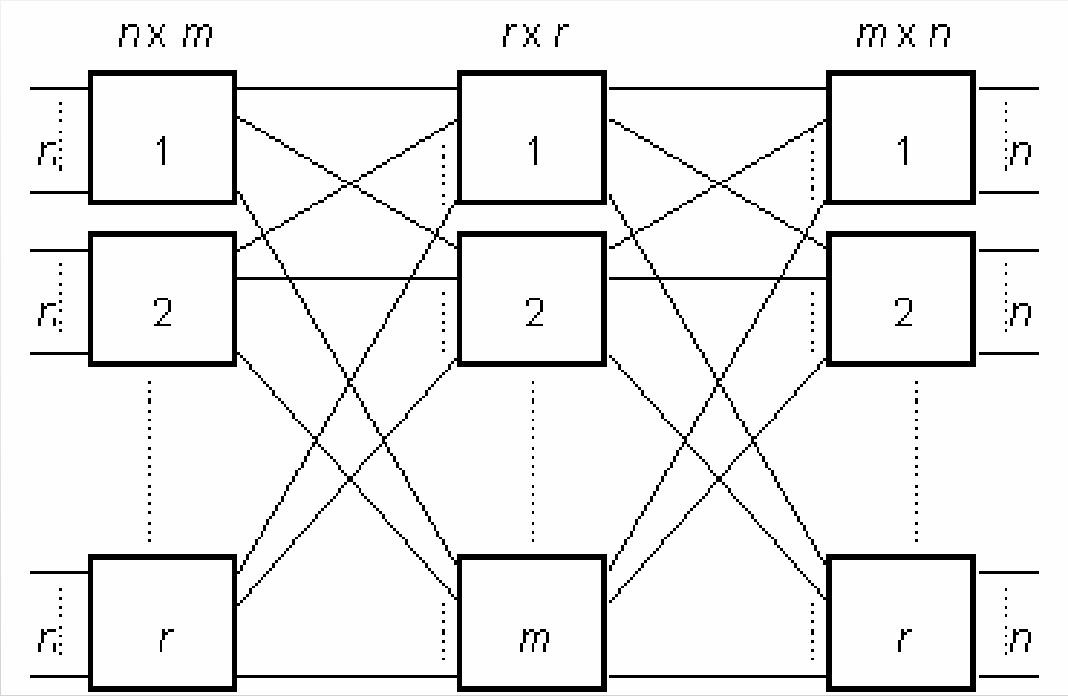
Clos network

- 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



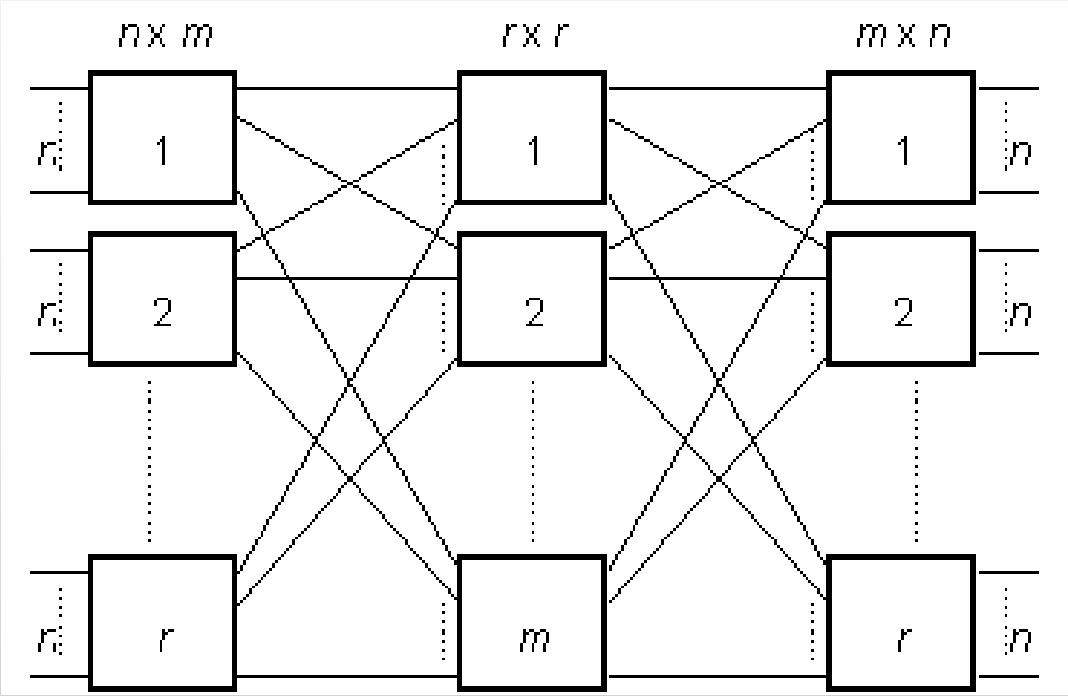
Clos network

- Clos networks are defined by three integers ***n*, *m*, and *r***
- *n* represents the number of sources which feed into each of *r* ingress stage crossbar switches



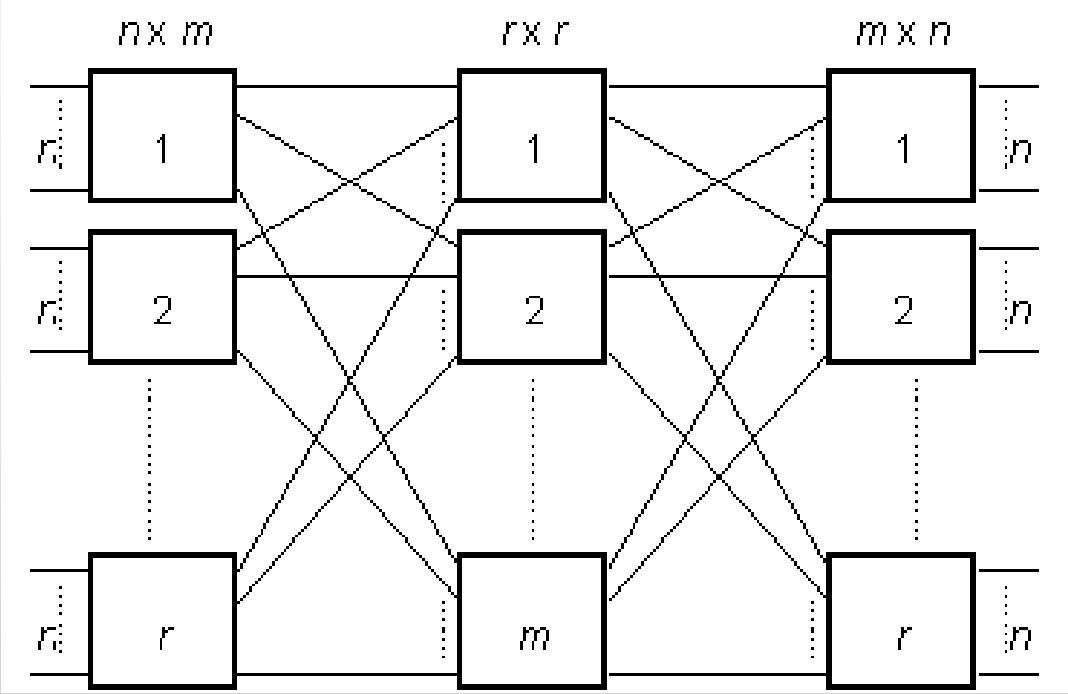
Clos network

- Each ingress stage crossbar switch has *m* outlets, and there are *m* middle stage crossbar switches
- There is exactly one connection between each ingress stage switch and each middle stage switch



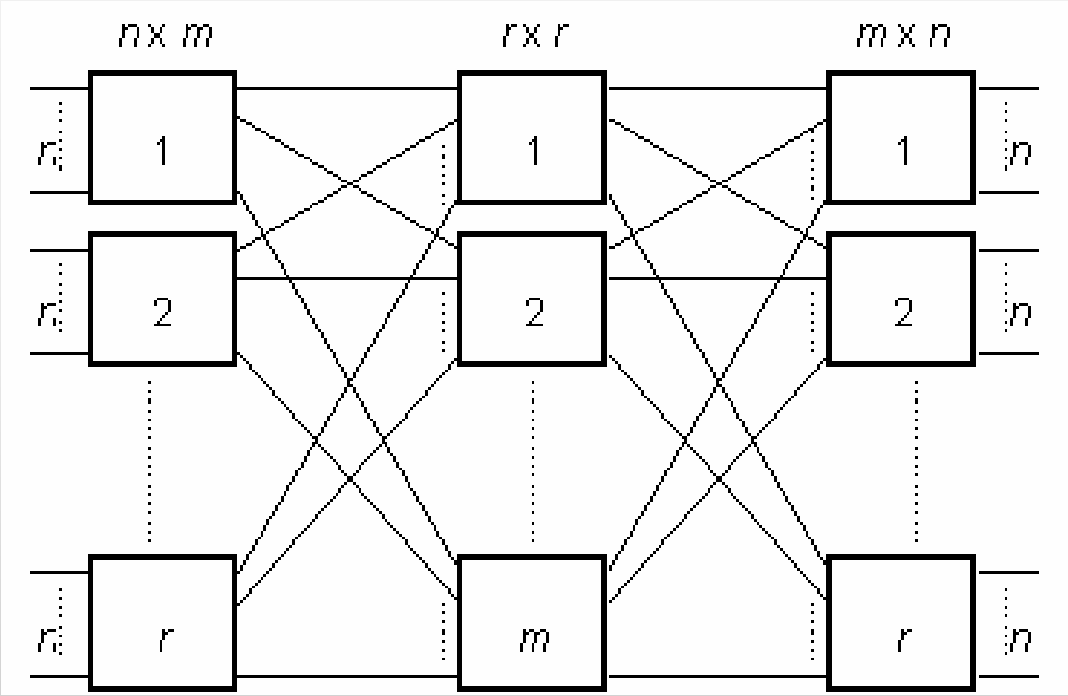
Clos network

- There are *r* egress stage switches, each with *m* inputs and *n* outputs
- Each middle stage switch is connected exactly once to each egress stage switch



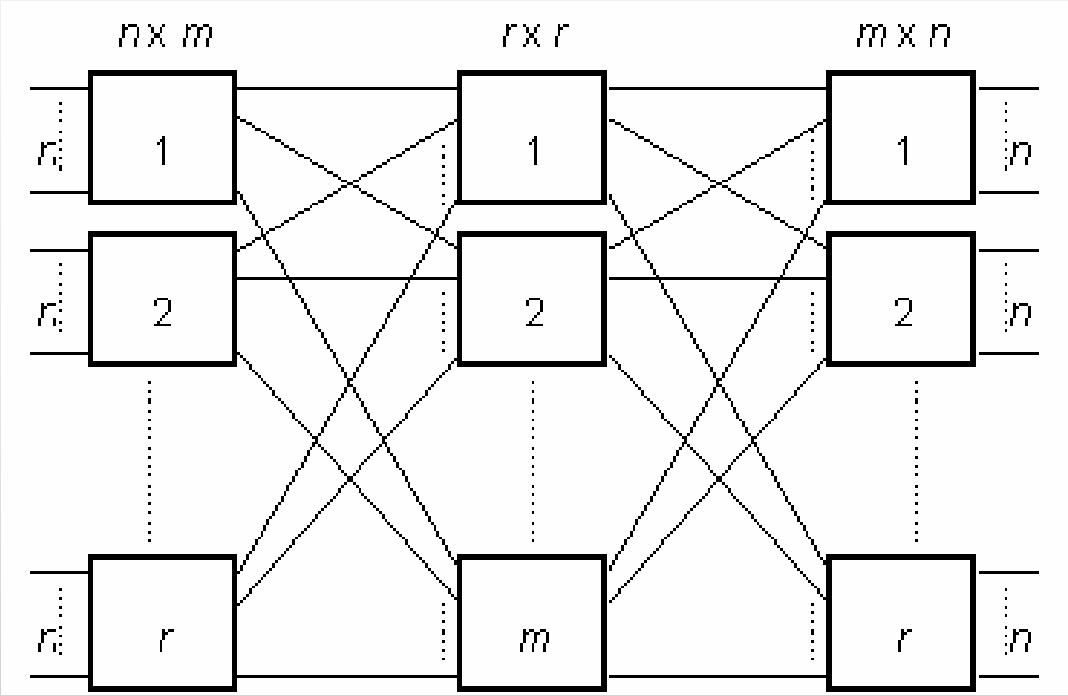
Clos network

- Thus
  - the ingress stage has r switches- - n inputs and m outputs
  - The middle stage has m switches - r inputs and r outputs
  - The egress stage has r switches - m inputs and n outputs



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's 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*

- 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

Strict-sense nonblocking Clos networks

- 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 ≥ 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 centre 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

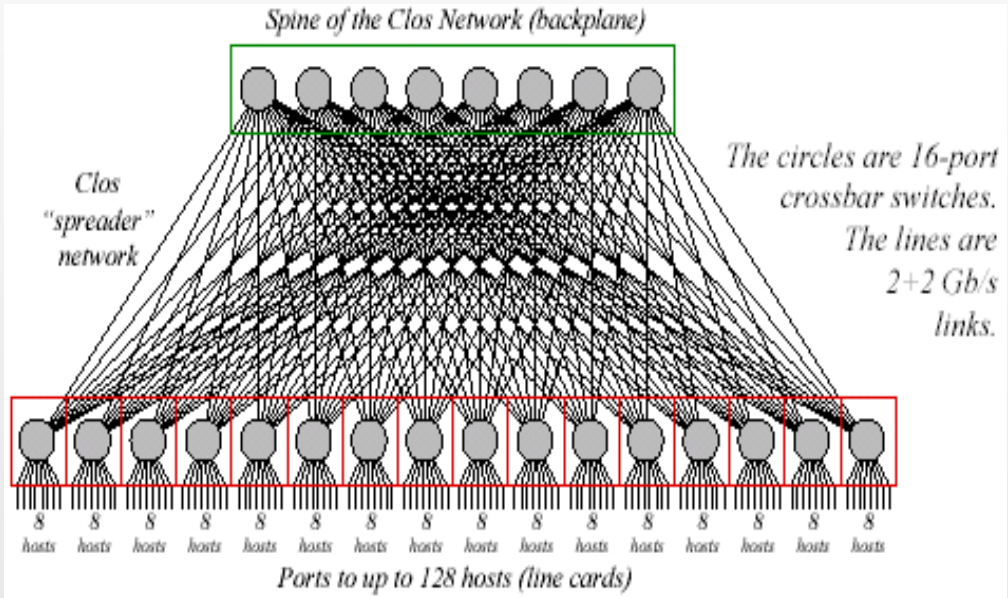
- In the context of a Clos network, each boy represents an ingress switch, and each girl represents an egress switch
- A boy is said to know a girl if the corresponding ingress and egress switches carry the same call
- Each set of k boys must know at least k girls because k ingress switches are carrying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

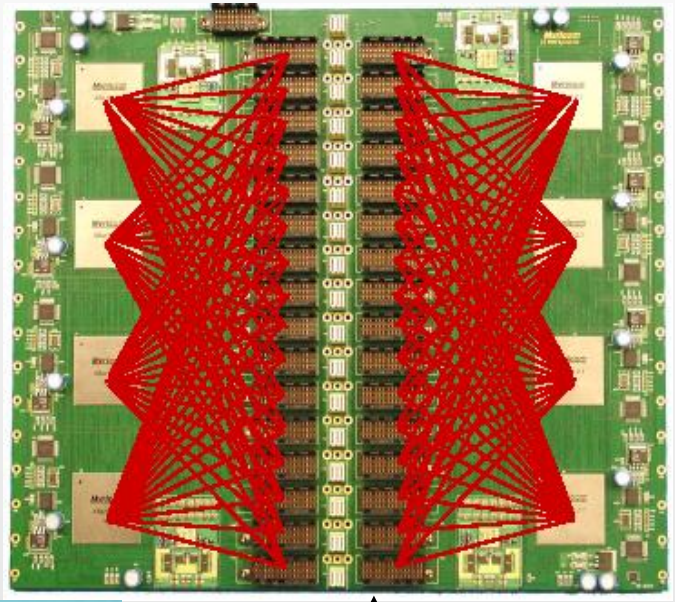
Rearrangeably nonblocking Clos networks

- 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





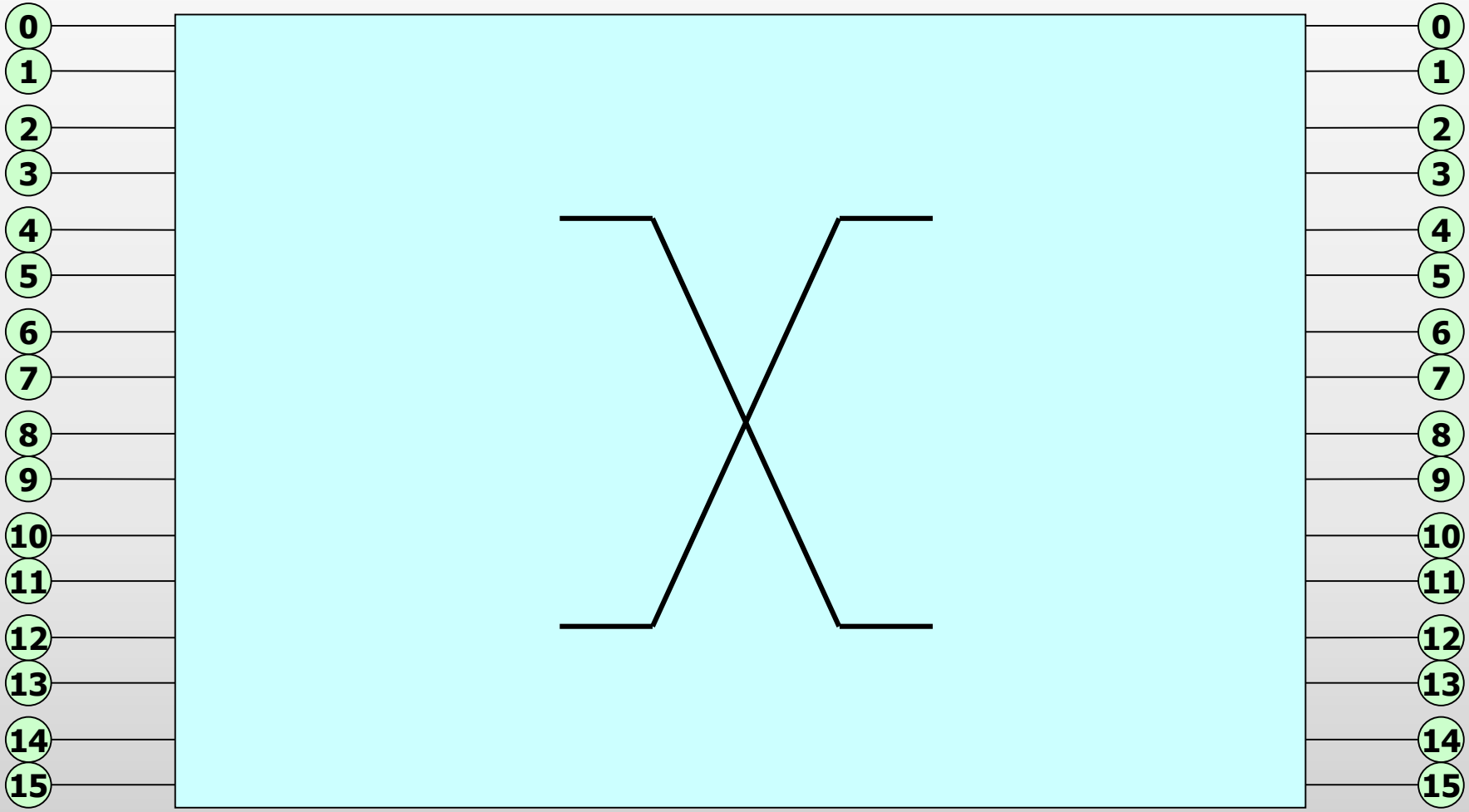
Backplane of the M3-E128 Switch

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

Benes Network

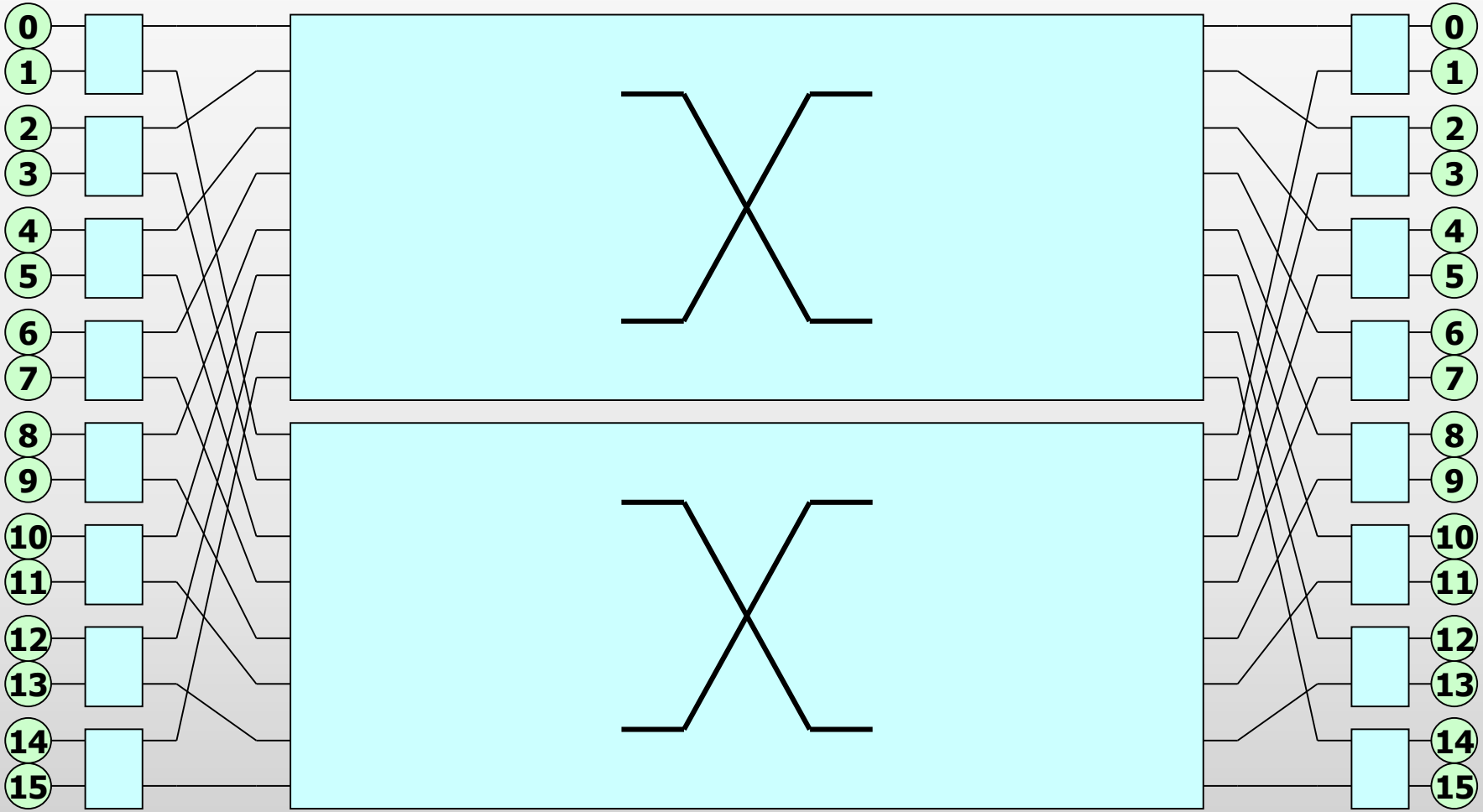
Benes Network

16 port *Crossbar* network



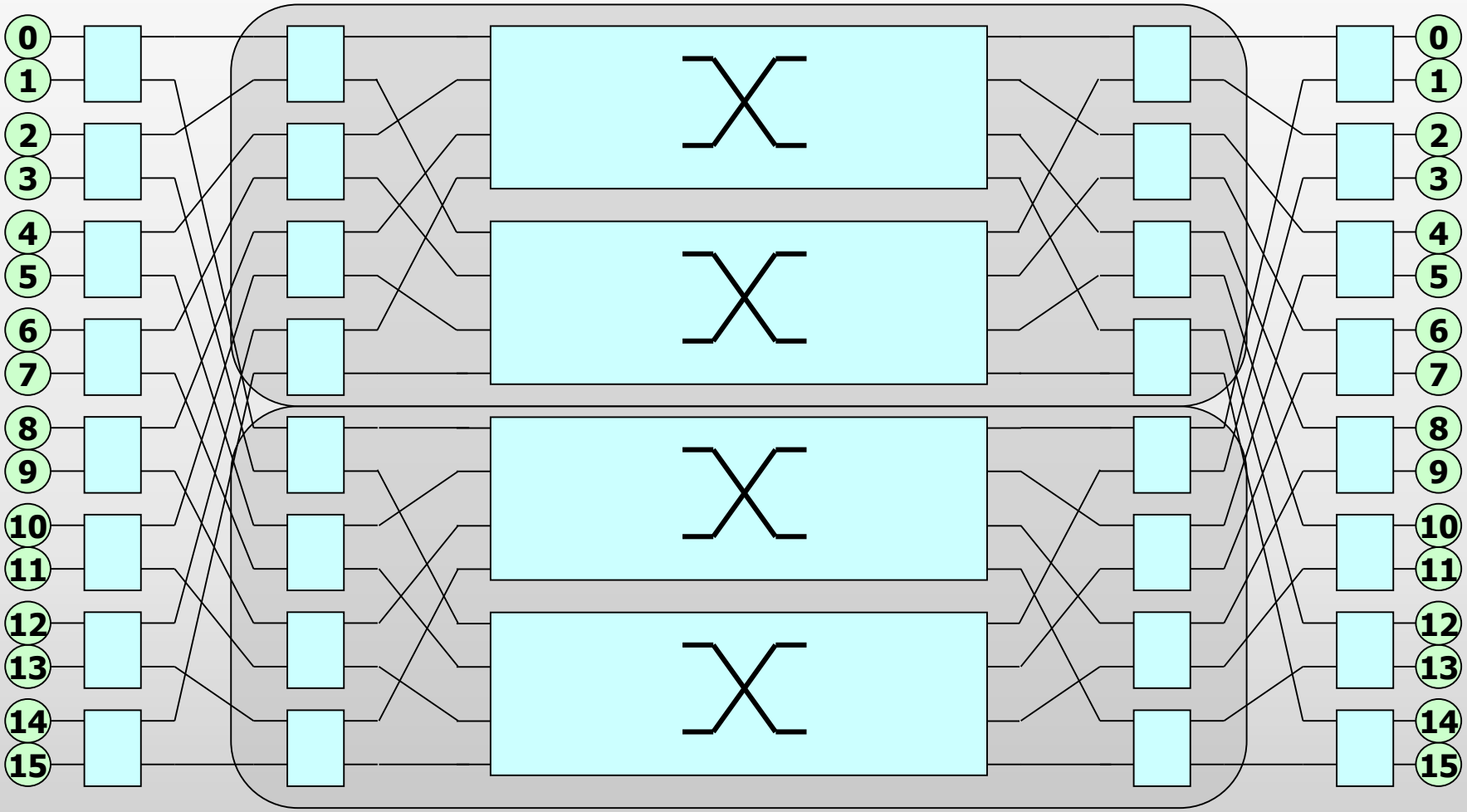
Benes Network

16 port, *3 stage Clos* network



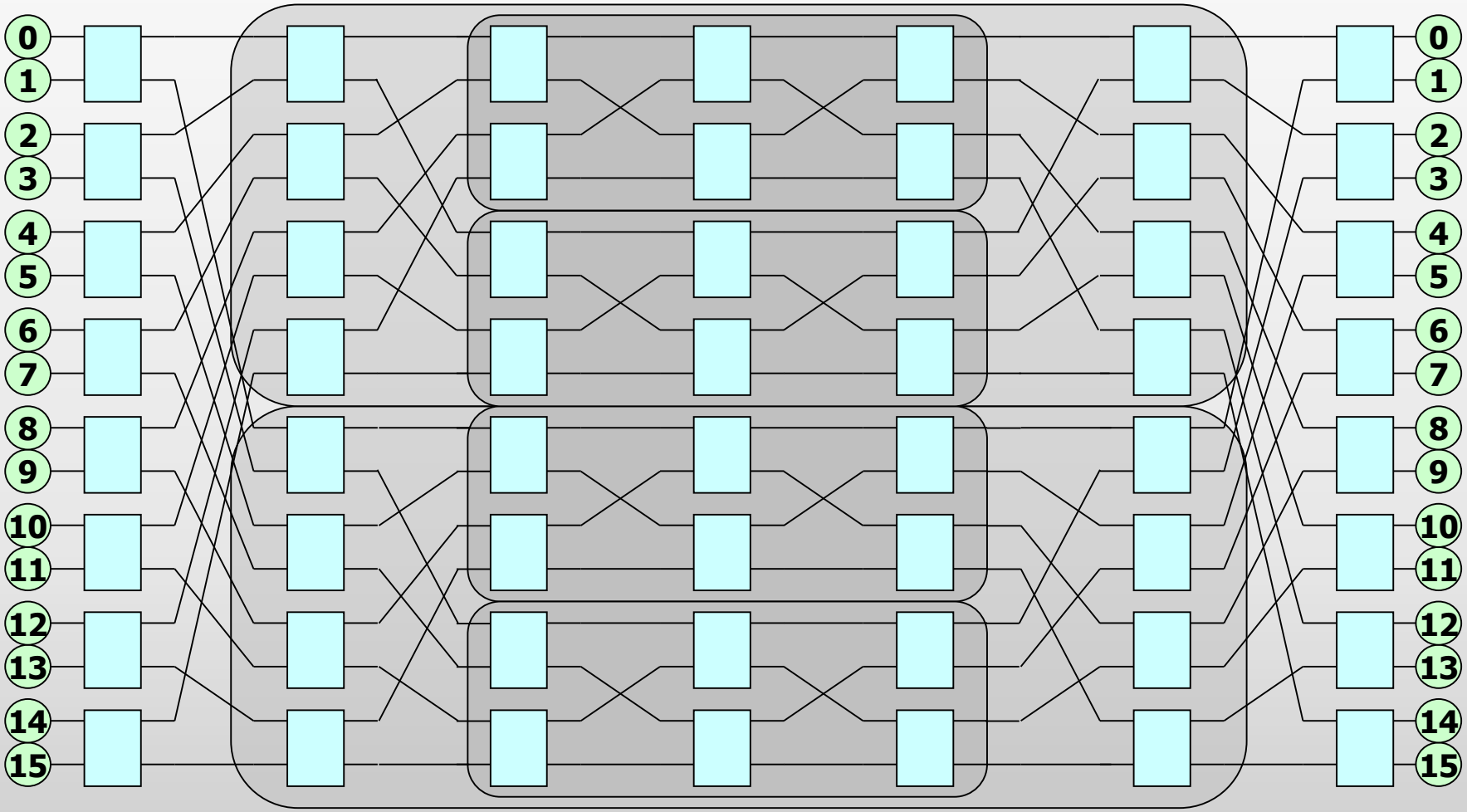
Benes Network

16 port, *5 stage Clos* network



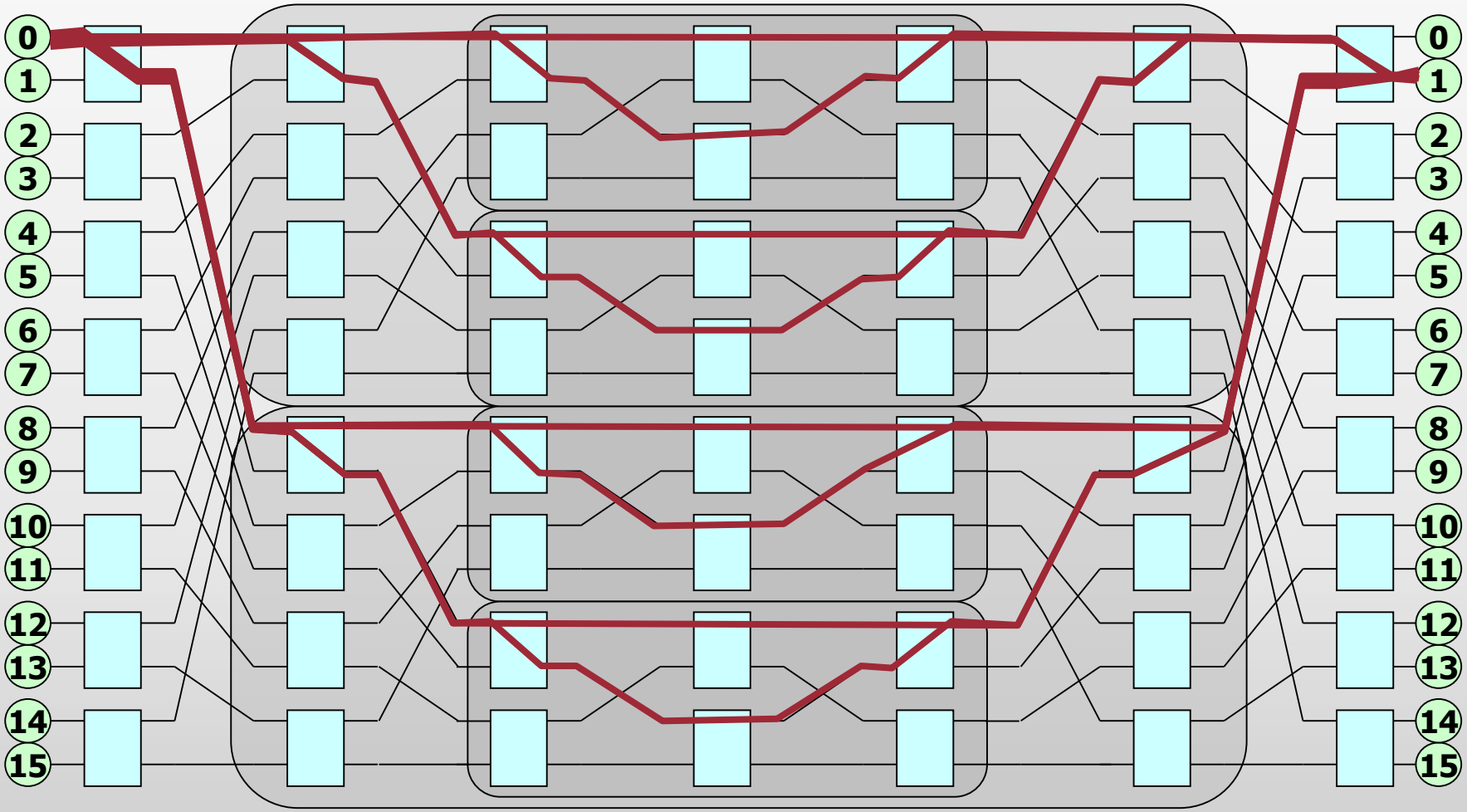
Benes Network

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



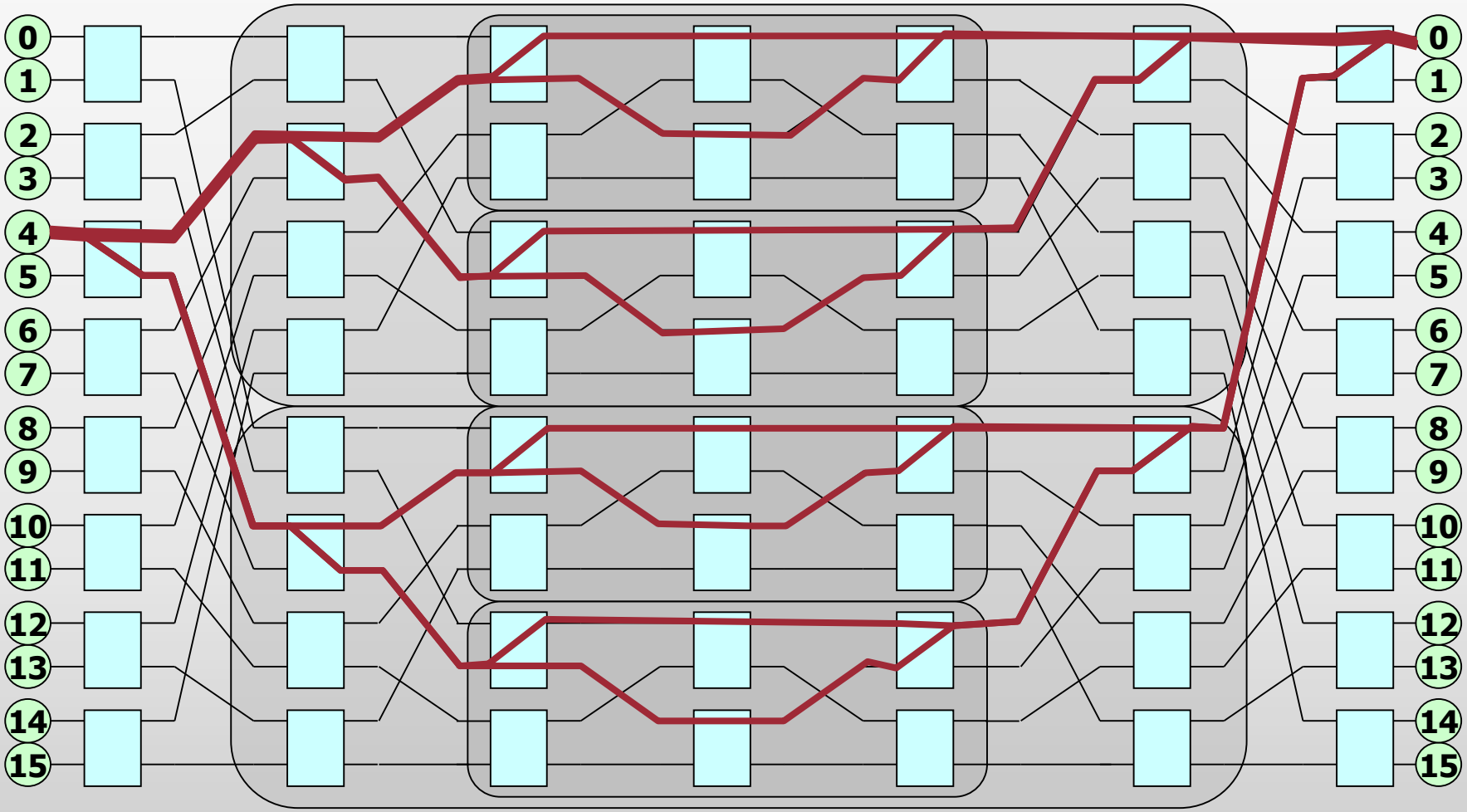
Benes Network

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



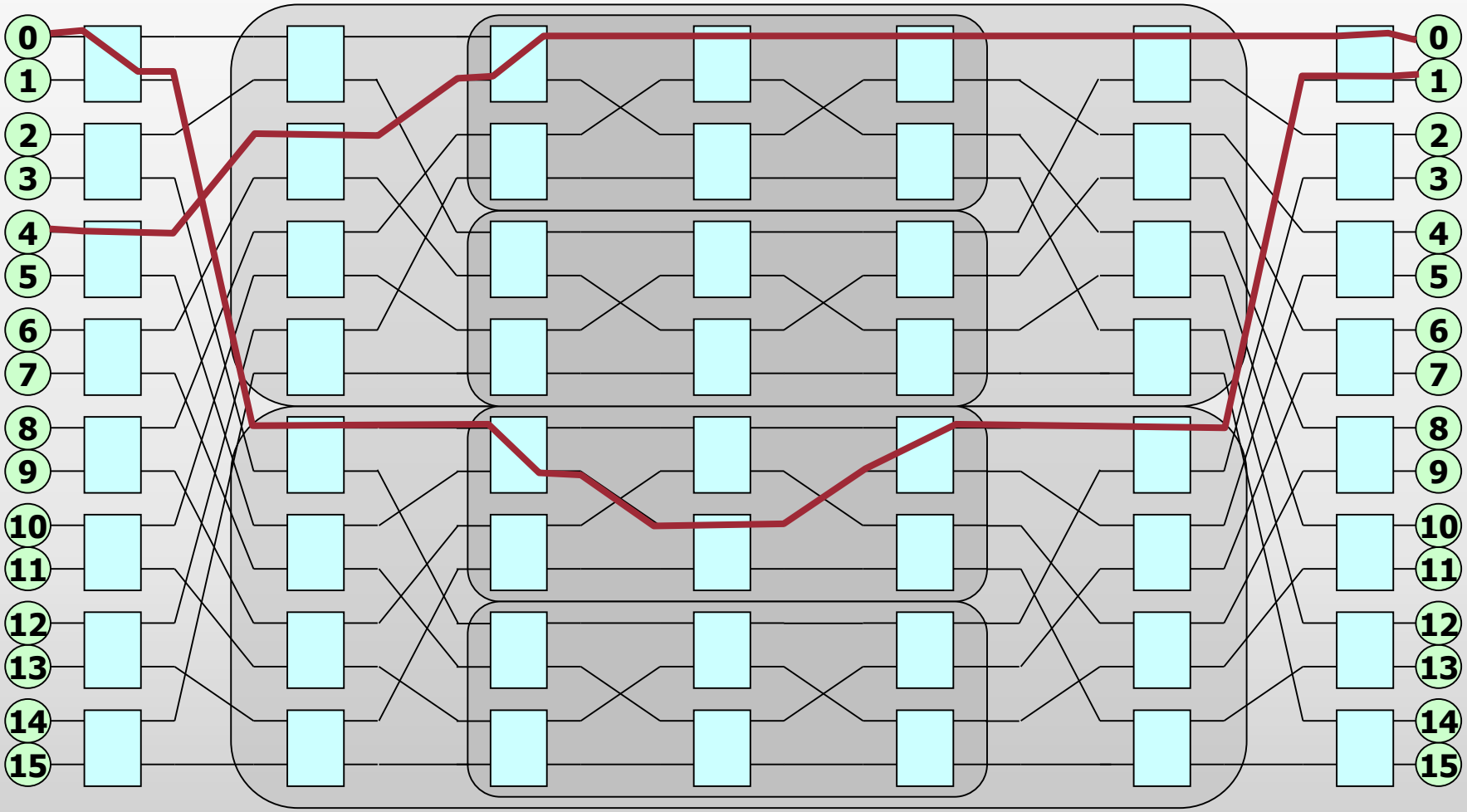
Benes Network

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



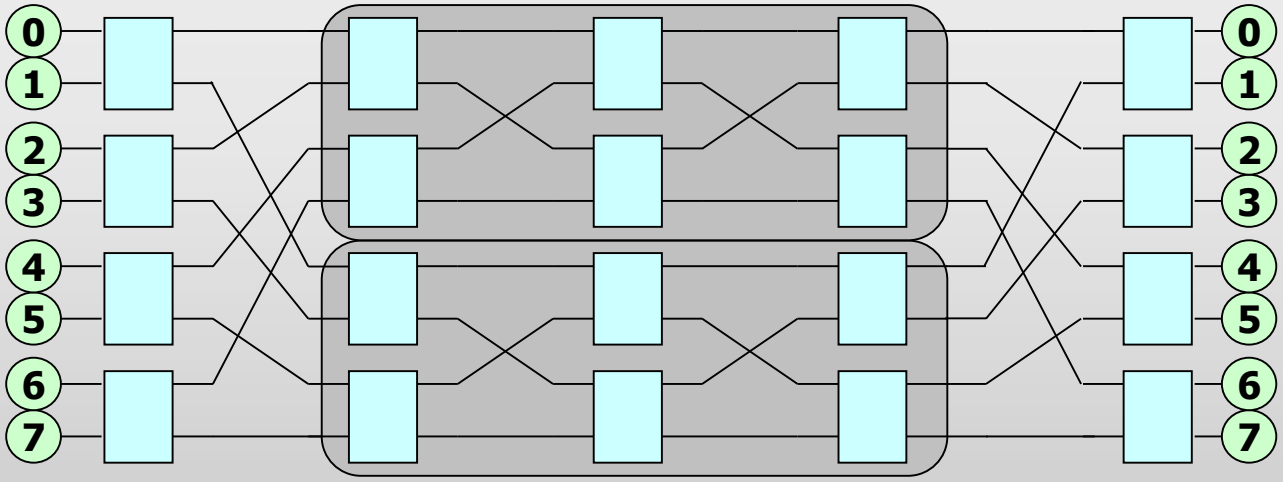
Benes Network

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



Looping algorithm

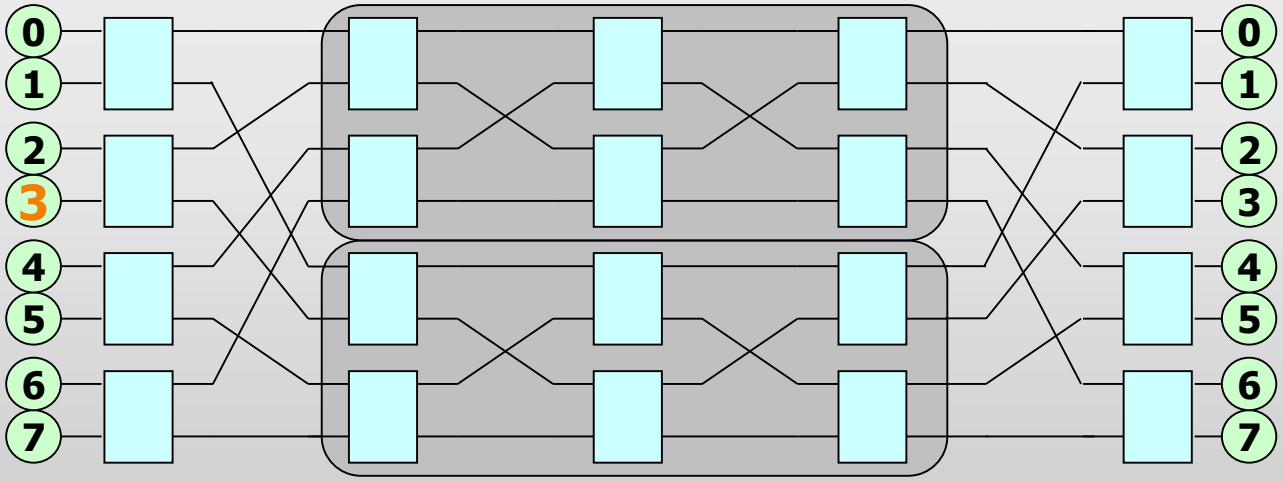
- Realizing permutations on a Benes network
  - The algorithm starts from arbitrarily chosen input by arbitrarily setting the corresponding switch
  - The input is connected to the requested output
  - The other output of the switch in the last stage is connected back to the corresponding input
  - The 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

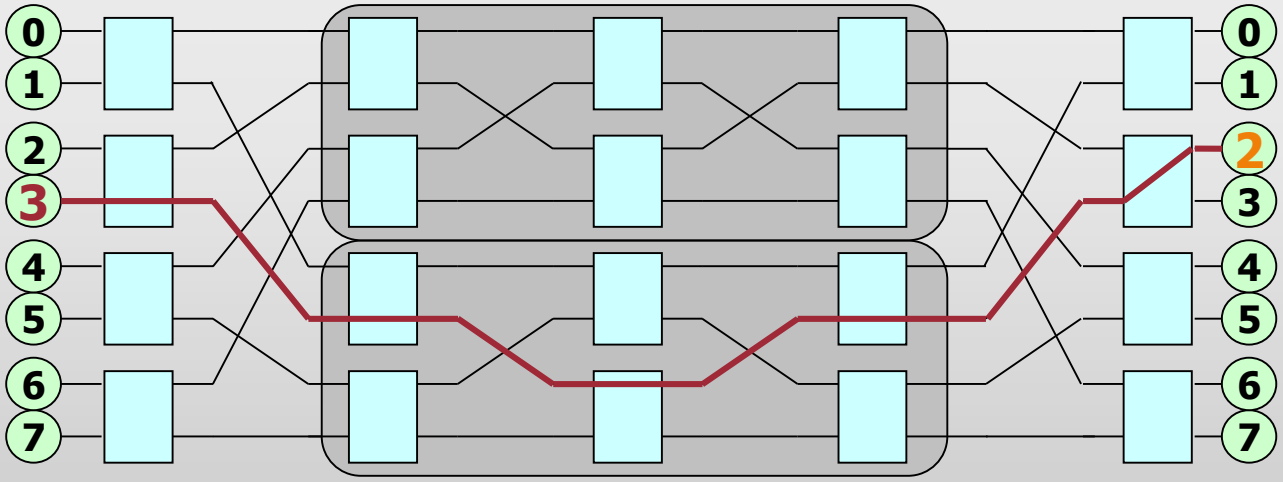
0	1	2	3	4	5	6	7
5	6	1	2	4	7	3	0



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

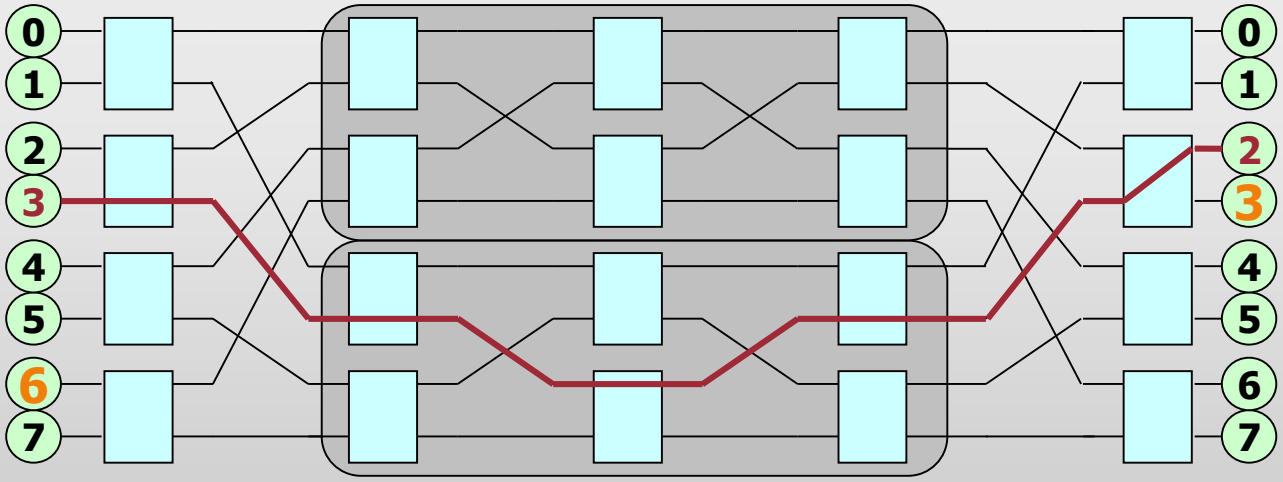
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 1 & 2 & 4 & 7 & 3 & 0 \end{pmatrix}$$



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

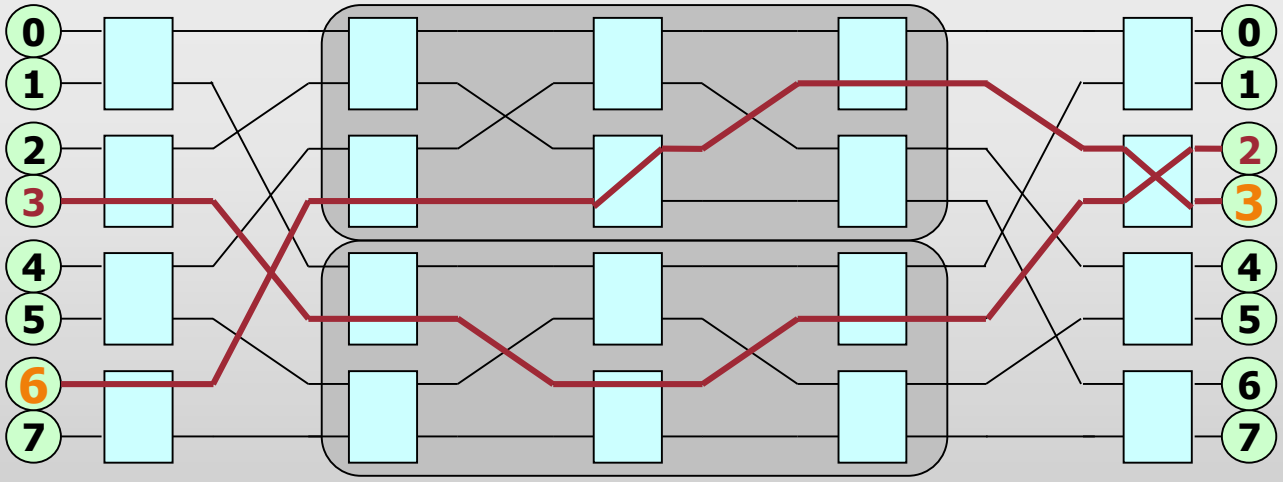
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 1 & 2 & 4 & 7 & 3 & 0 \end{pmatrix}$$



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

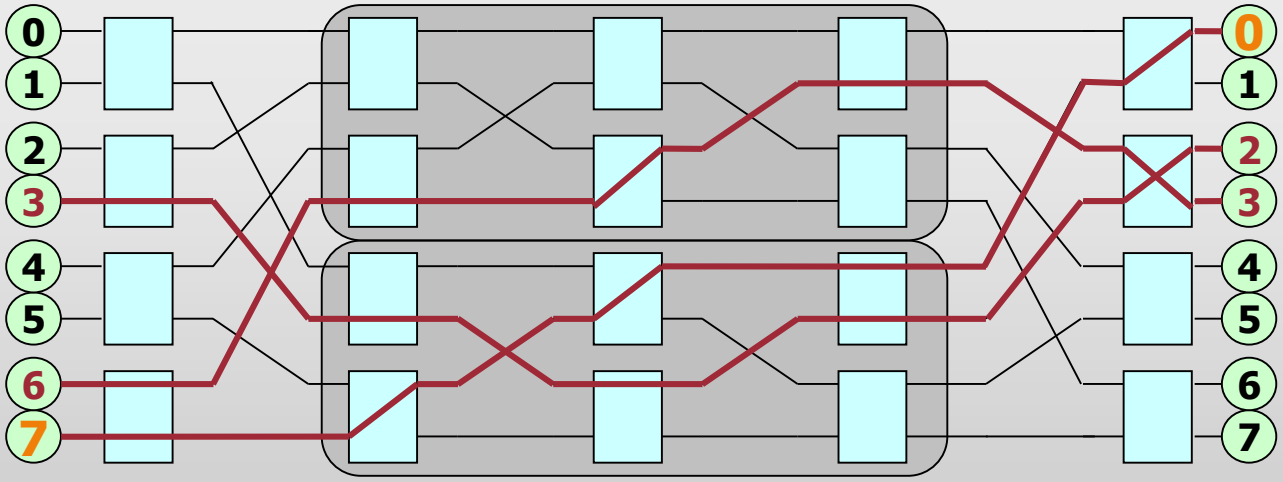
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 1 & 2 & 4 & 7 & 3 & 0 \end{pmatrix}$$



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

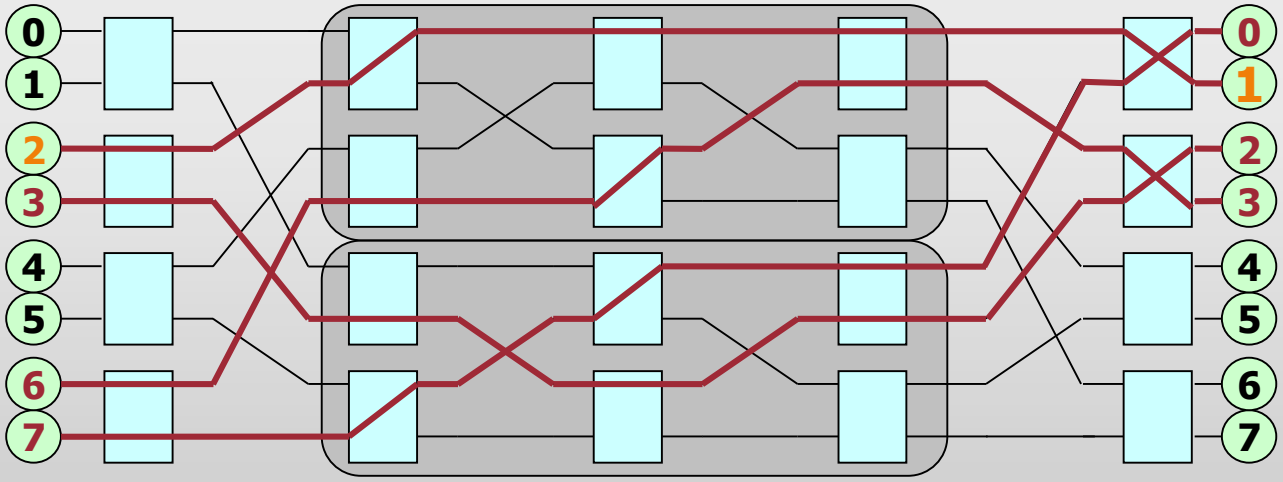
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 1 & 2 & 4 & 7 & 3 & 0 \end{pmatrix}$$



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

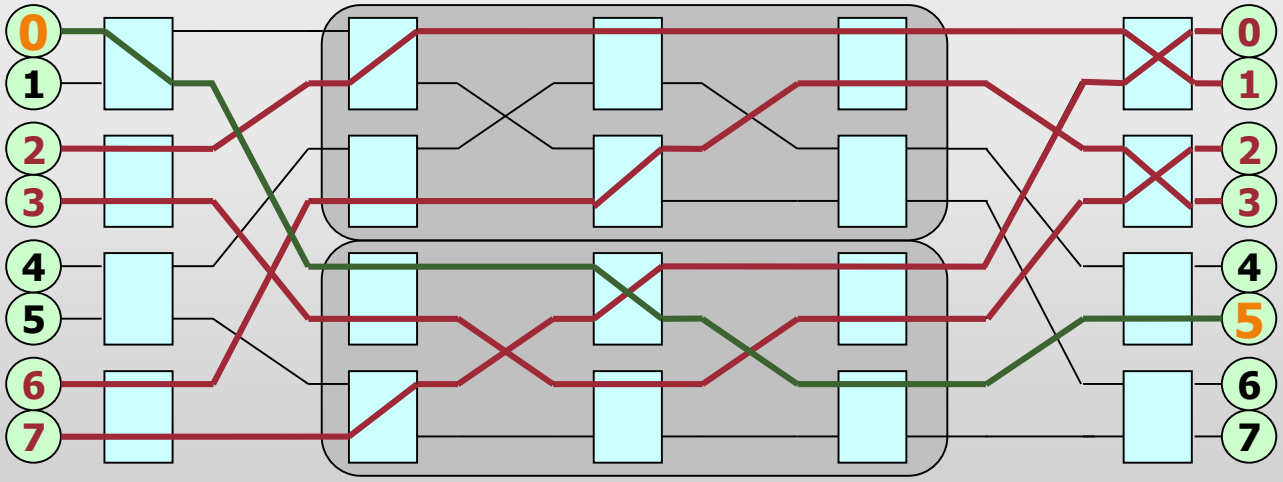
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 1 & 2 & 4 & 7 & 3 & 0 \end{pmatrix}$$



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

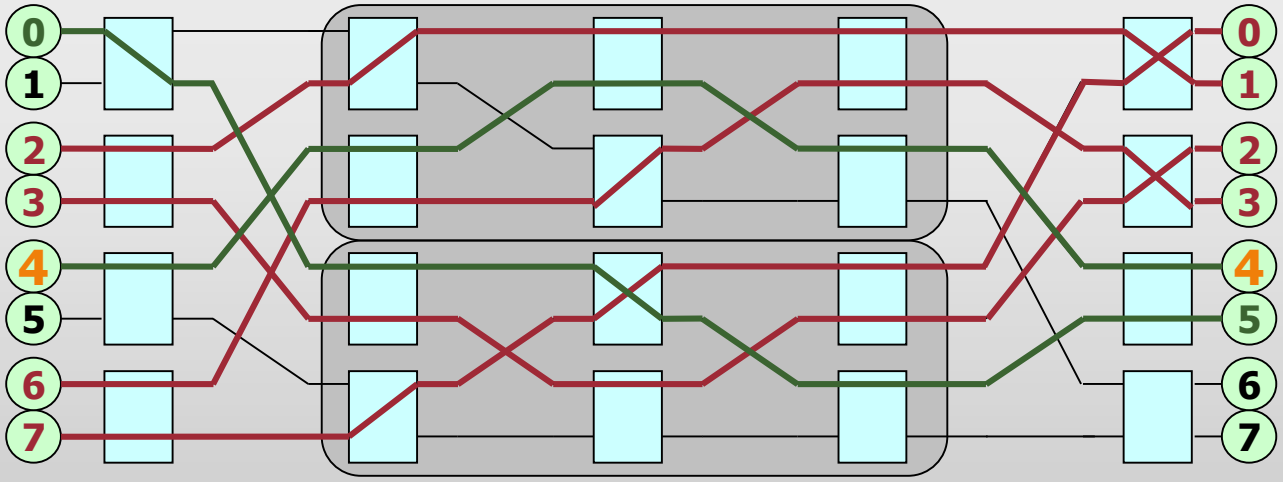
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 1 & 2 & 4 & 7 & 3 & 0 \end{pmatrix}$$



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

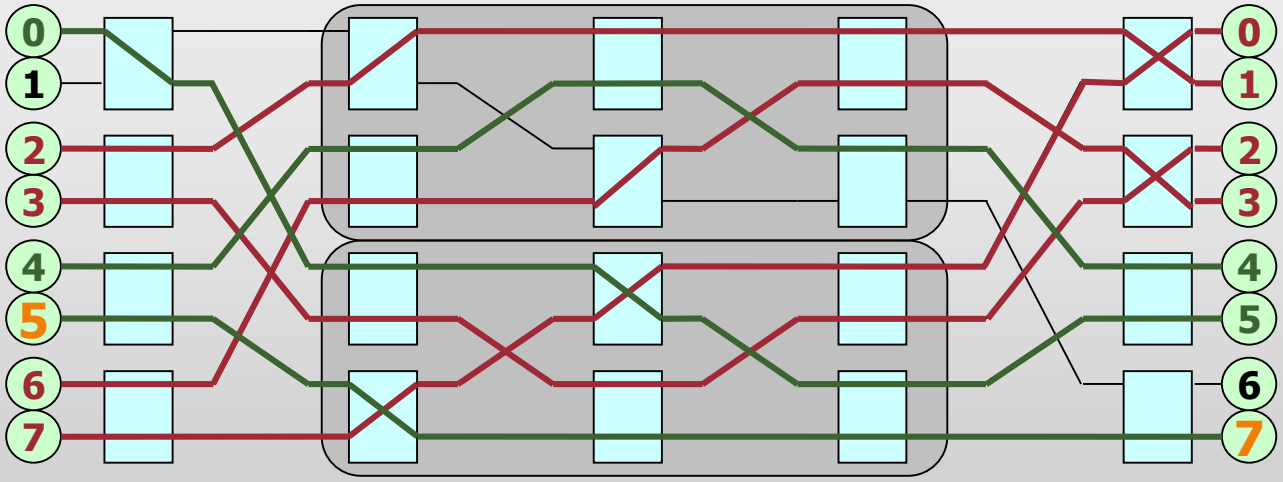
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 1 & 2 & 4 & 7 & 3 & 0 \end{pmatrix}$$



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

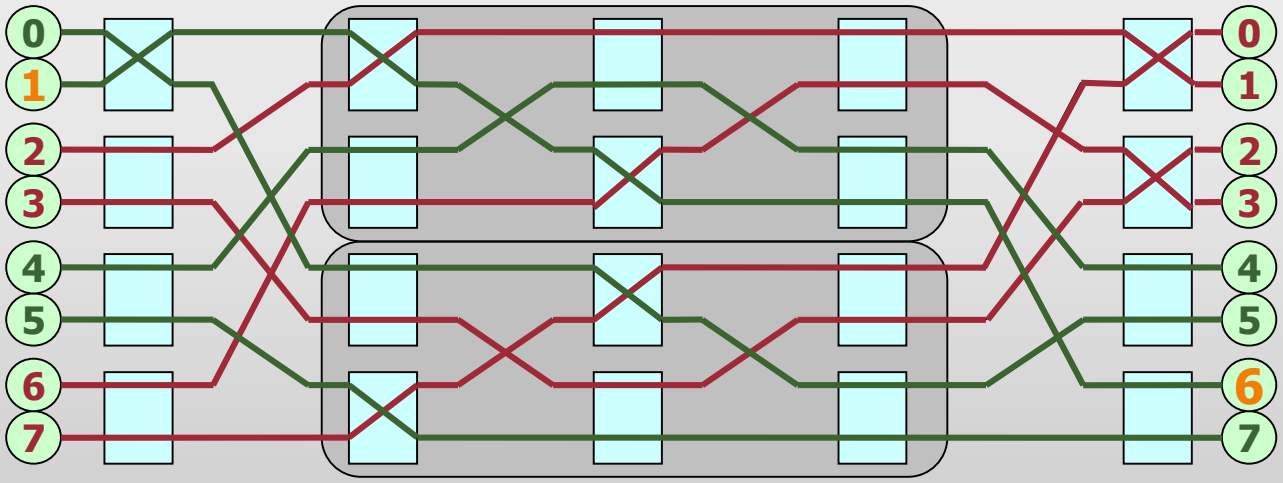
$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 5 & 6 & 1 & 2 & 4 & 7 & 3 & 0 \end{pmatrix}$$



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 (and Layered Cross Product)

Topological and functional equivalence

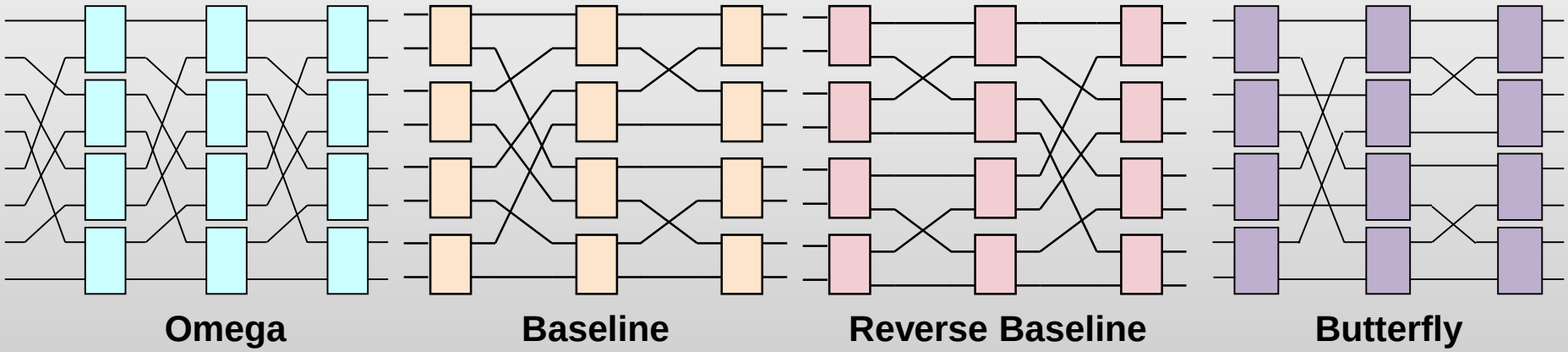
- There are two different concepts of equivalence:
  - Topological equivalence: isomorphism
  - Functional equivalence: capability of always performing the same set of assignments

- Topological equivalence is different from functional equivalence:
  - All rearrangeable MINs are functionally equivalent though not necessarily topologically equivalent
  - Not rearrangeable N-MINs could be topologically equivalent but not functionally equivalent

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

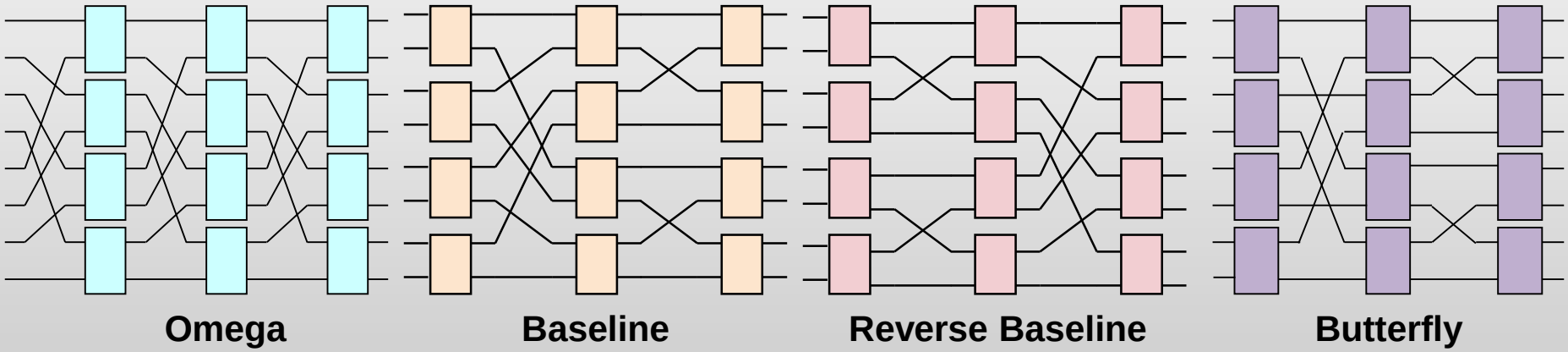
Bermond, Fourneau and Jean-Marie (1982) give the characterization of MINs topologically equivalent to the Reverse Baseline network. It is based on:

- 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 Gi,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) – is based on the Layered Cross Product Even and Litman (1992)
  - An **l-layered graph**, G = (V_1, V_2, . . . , 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'⊗G'', of two l-layered graphs G' = (V'_1, V'_2, . . . , V'_l, E') and G'' = (V''_1, V''_2, . . . , V''_l, E'') is an l-layered graph G = (V_1, V_2, . . . , V_l , E) where V_i is the cartesian product of V'_i and V''_i , $1 \leq i \leq l$, and an edge <(u', u''),(v', v'')> belongs to E if and only if <u' , v'> ∈ E' and <u'' , v''> ∈ 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

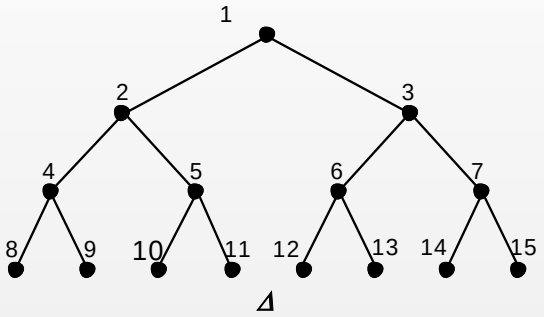
- **Theorem** Let G′ and G′′ be two *s* stage MINs, and let G′ decomposable as $G'_1 \otimes G'_2$. Then G′′ is topologically equivalent to G′ if and only if G′′ can be decomposed as $G'_1 \otimes G'_2$
- **Corollary** Given two N-MINs $G' = G'_1 \otimes G'_2$ and $G'' = G''_1 \otimes G''_2$, they are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

Topological equivalence

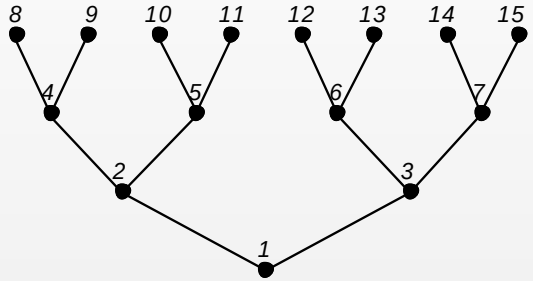
- **Lemma** A MIN G satisfies the Banyan and P(∗, ∗) properties if and only if it can be decomposed as $\Delta \otimes \nabla$, where Δ and ∇ denote binary trees with the root on the top and in the bottom, respectively

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

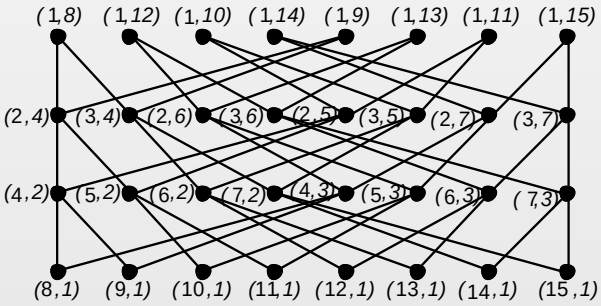
Topological equivalence



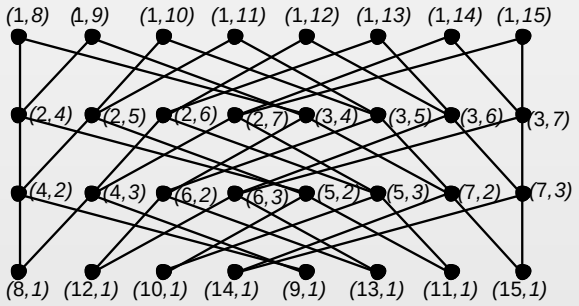
Δ



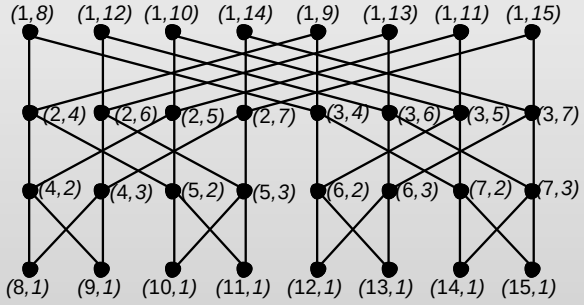
Nabla



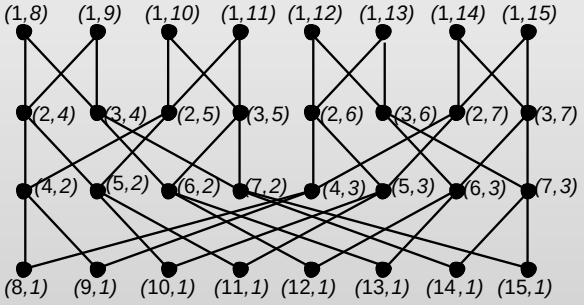
Omega



Flip



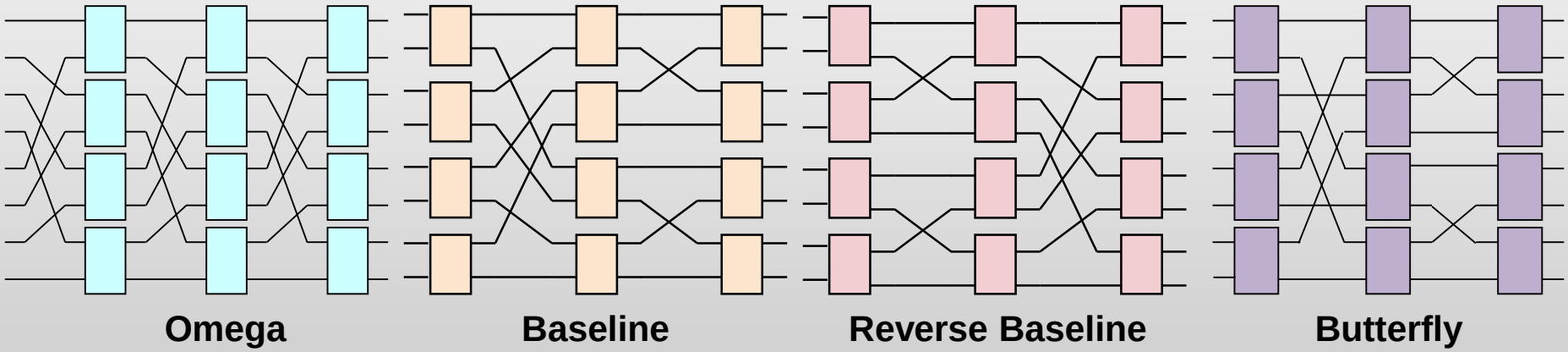
Butterfly



Reverse Baseline

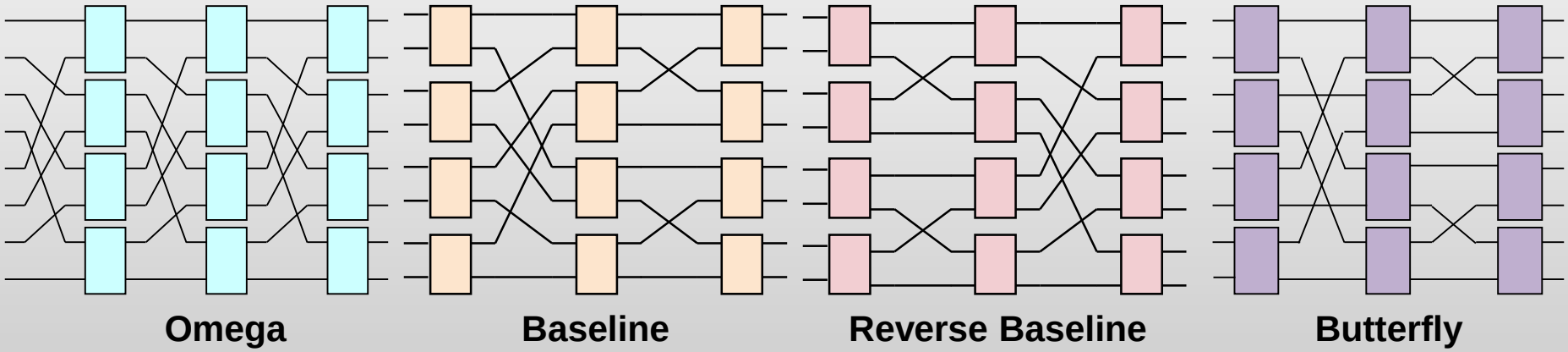
Topological equivalence

- 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



Topological equivalence

- For this reason, MINs obtained by concatenating two logN 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



2logN-1 stage MIN equivalence

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

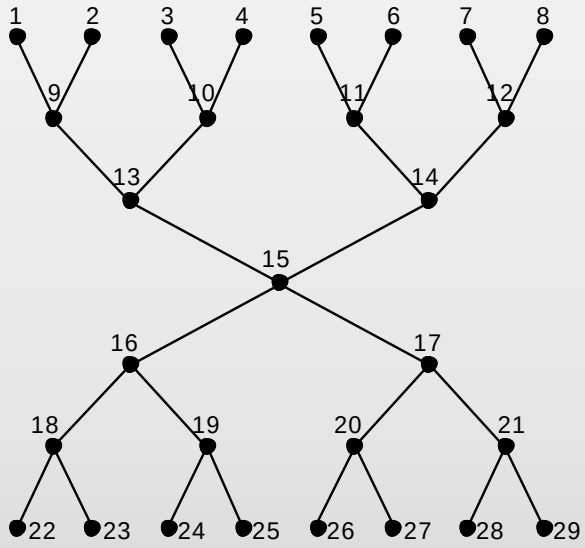
- Looping algorithm does not work on the Omega-Omega^{-1} or Double Baseline even if they are equivalent to the Benes network

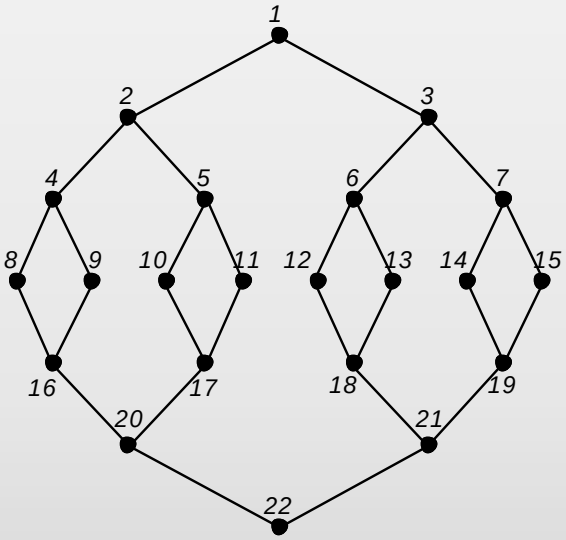
2logN-1 stage MIN equivalence

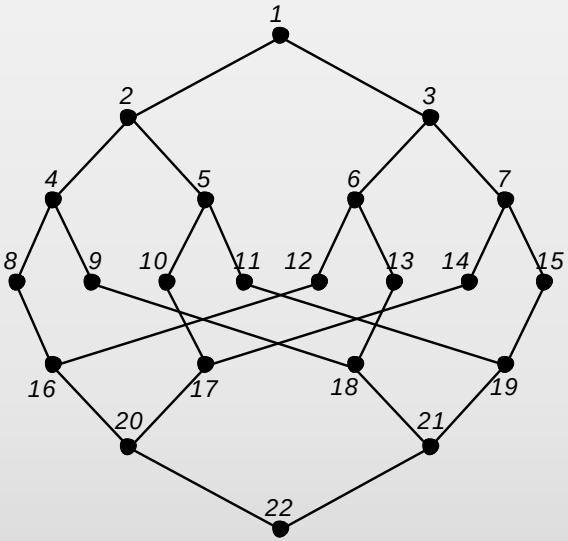
- It is typical to concatenate all the combinations of pairs of networks among Butterfly, Omega, Flip, Baseline, their reverses, etc. to obtain a new N-MIN
- Both the two log N stage MINs constituting a (2log N- 1) stage MIN can be decomposed as LCP of $\Delta \otimes \nabla$
- As a consequence, we obtain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

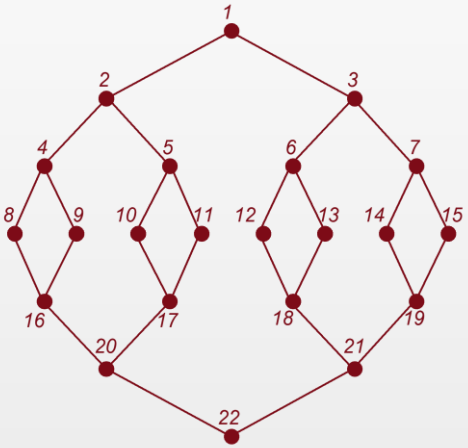
- It is obvious how to merge the last layer of a ∇ with the first layer of a Δ, but there are many ways of merging the last layer of a Δ and the first layer of a ∇ respectively

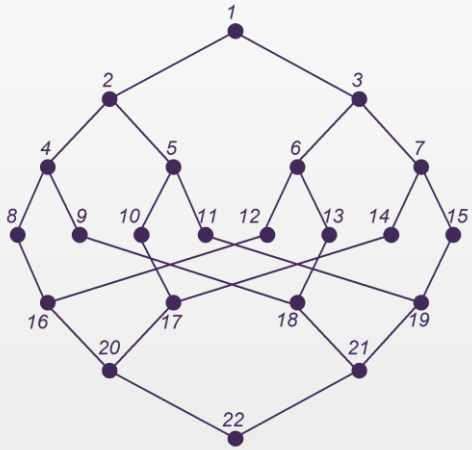


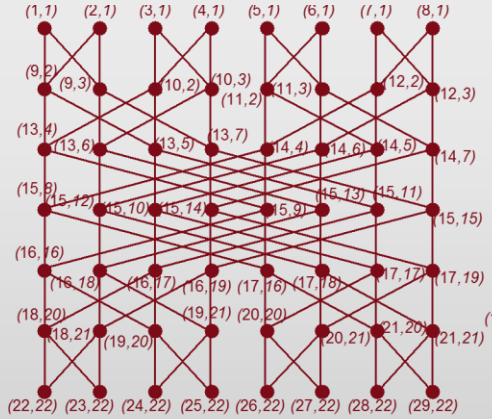




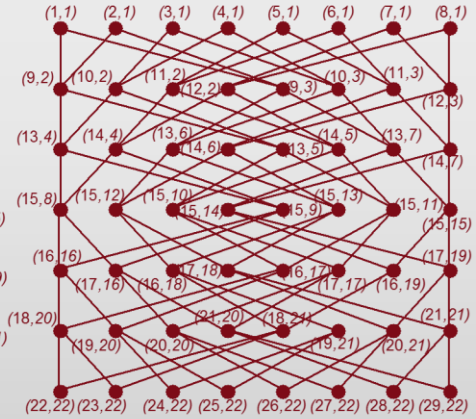
2logN-1 stage MIN equivalence



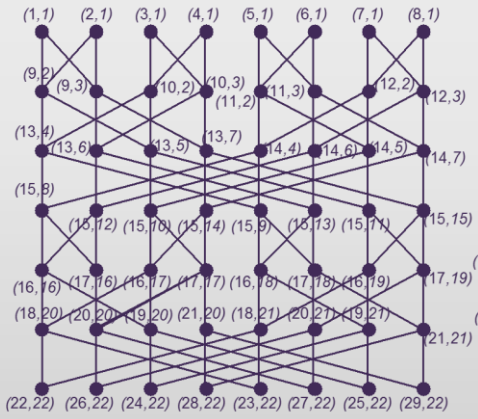




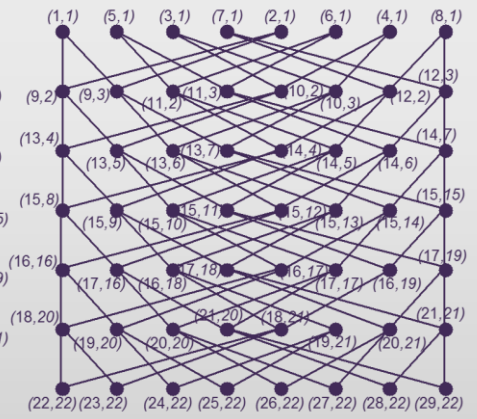
A reverse Butterfly and a Butterfly



A Flip and a Omega



Two reverse Butterflies



Two Omega

2logN-1 stage MIN equivalence

- **Theorem** The number of distinct equivalence classes of (2 logN - 1) MIN s 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