

# Packet Switch Architecture

3. Output Queueing Architectures
4. Input Queueing Architectures
5. Switching Fabrics
6. Flow and Congestion Control in Sw. Fabrics

*Manolis Katevenis, Nikolaos Chrysos*

FORTH and Univ. of Crete, Greece

<http://archvlsi.ics.forth.gr/~kateveni/534>

# 5. Switching Fabrics

## Table of Contents:

- **5.0 Introduction**
  - multistage netw., bisection b/w, non-blocking perf., routing categories
- **5.1 Inverse Multiplexing (Adaptive / Multipath Routing)**
  - byte-sliced switches, recursive definition of the Benes network
  - load distribution & balancing, packet ordering & resequencing
- **5.2 Scalable Non-Blocking Switching Fabrics**
  - Banyan ( $k$ -ary  $n$ -flies), Benes, Clos –  $O(N \cdot \log N)$  cost & lower bound
  - fat trees ( $k$ -ary  $n$ -trees) – controlled blocking, locality of traffic
  - fully-connected networks – flat networks
  - Dragonfly networks – few global links and small diameter
- **5.3 What about Scalable Scheduling?**
  - buffers in multi-stage fabrics
  - scheduling bufferless Clos networks, load-balanced switches
  - self-routing fabrics, sorting networks: bad solution
  - fabrics with small internal buffers and flow control: good solution

## 5. Switching Fabrics

- **What are switching fabrics (or multi-stage interconnection networks)?**
  - a network consisting of multiple smaller switches interconnected via channels (point-to-point links)
    - switches are usually crossbars, each one impl. in a single chip/board
- **Why switching fabrics?**
  - crossbars do not scale to large port counts
    - $N^2$  crosspoints
    - I/O chip bandwidth (# pins / HSS cores & power limitations)
- **Where are they deployed?**
  - inside large routers/switches
    - multi-chip/multi-chassis routers
    - single-chip switches (internally multi-stage)
  - inside modern datacenters and high-performance computers
  - inside chip multi-processors (Networks-On-Chip)

## 5.0 Switching Fabrics: terminology

- **Network = nodes + channels**
  - node= terminal or switch, channel= connection (link) between 2 nodes
- **Path** = a set of channels  $\{c_1, c_2, \dots, c_n\} : d_{c_i} = s_{c_{i+1}}, \text{ for } i \text{ in } 1 \dots (n-1)$
- **Hop count** of path: the number of channels traversed in the path
- **Connected network**: path exists between any pair of terminals
- **Minimal path** from node  $x$  to node  $y$  = the path with the smallest hop count connecting node  $x$  to node  $y$
- **Network diameter** = the largest hop count over all pairs of terminals

## 5.0 Switching Fabrics: performance

- **Typically sub-optimal performance (compared to xbars)**
  - “ideally, we would like to connect all processors in a datacenter using a single flat (crossbar-like) network”
- **Challenges**
  - full / high tput irrespective of traffic pattern/orientation (routing)
  - fairness (scheduling)
  - flow isolation (congestion control)
  - equidistant paths ?
    - same latency irrespective to which ports communicate
- **Recent trend: datacenters networks → flattened datacenter fabrics**
  - replace previous “slim” datacenter nets w. high-performance fabrics

# 5.0 Non-Blocking Switching Fabrics

- **Non-blocking fabrics/ networks**

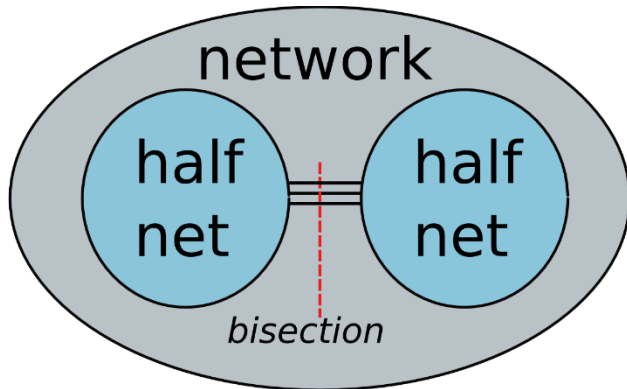
- can route any input-output permutation
  - necessary condition: at least  $N!$  states  $\rightarrow \geq \log_2(N!)$  crosspoints
- feasible traffic for network ports (for each port : sum load  $\leq 1$ )  $\rightarrow$  feasible for internal links as well
  - necessary condition: full bisection bandwidth

- **Strictly vs. rearrangeably non-blocking networks**

- if netw. currently “switches” connections {1-0, 2-1, 0-3}
  - adding 3-2 does not require rerouting connections (strictly non-blocking)
  - adding 3-2 may require rerouting existing connections (rearrangeably non-blocking)

*Typically, a network with path diversity ( $\geq 1$  paths for port-pair flows) becomes non-blocking only if appropriate routing is applied*

## 5.0 Bisection Bandwidth



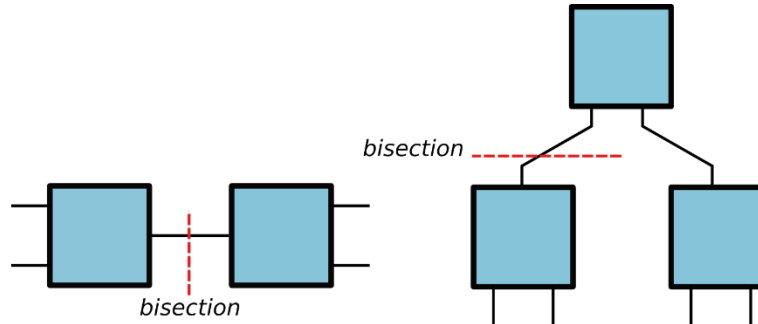
### **Full bisection bandwidth network**

- in each direction, the bisection has the same capacity as  $N/2$  ports

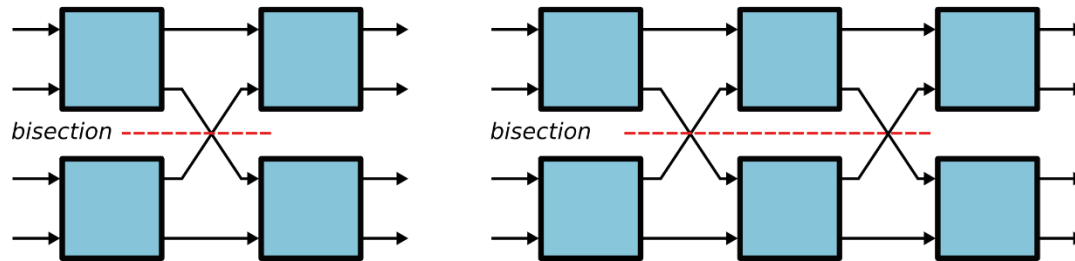
- **A bisection is a set of channels that partitions:**
  - nodes into two ~ equal groups:  $|N1| \leq |N2| + 1$ ,  $|N2| \leq |N1| + 1$
  - terminals nodes into two ~equal groups :  $|n1| \leq |n2| + 1$ ,  $|n2| \leq |n1| + 1$
- **Bisection bandwidth = minimum bandwidth over all bisections**
  - implementation cost (global wiring)
  - non-blocking performance (if no “full bisection” then the network is blocking)
    - however, full bisection does not guarantee non-blocking performance (routing)

# 5.0 Bisection Bandwidth: examples

*Bidirectional networks*

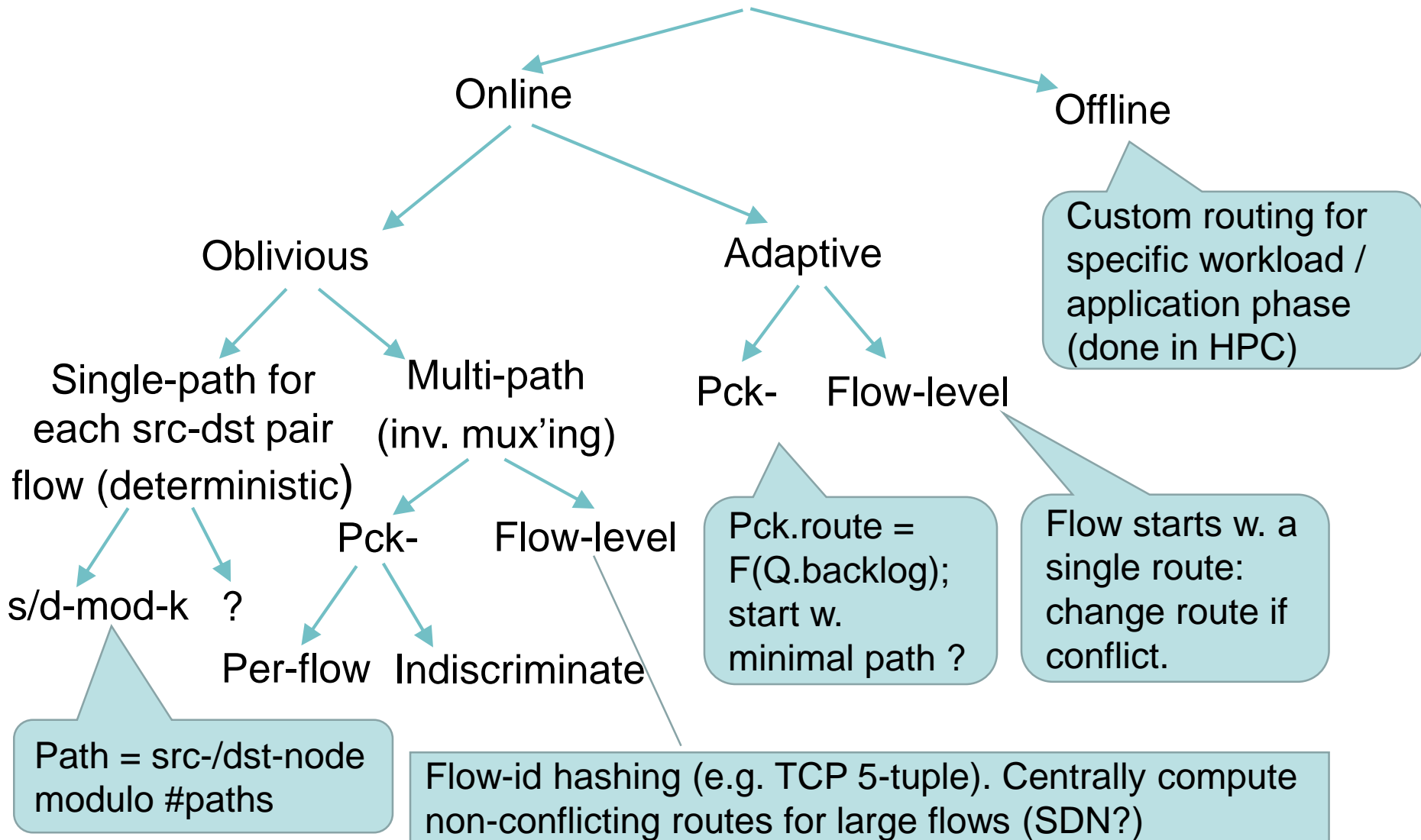


*Unidirectional networks*





# Routing Strategies: a taxonomy



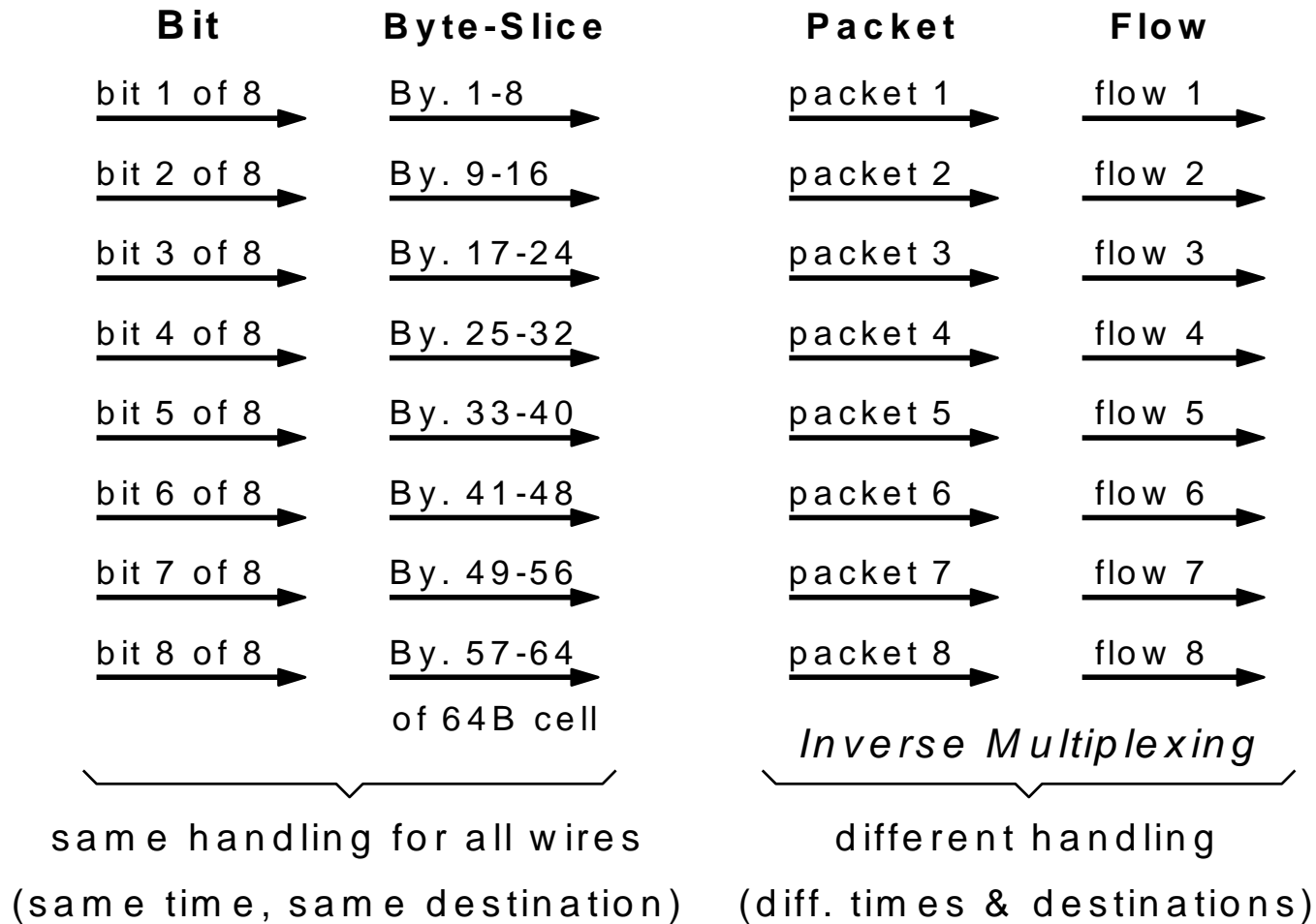
# Additional routing categories

- Source (or explicit) routing
  - path computed at source & carried in packet header
- Self-routing (network)
  - path computed gradually at network nodes using header bits
    - $k$ -ary  $n$ -flies,  $k$ -ary  $n$ -trees, and Benes/Clos can operate as self-routing networks (but usually more sophisticated decisions at stages where multiple paths available)
- Routing tables (e.g. Ethernet, IP, Infiniband)
  - arbitrary routing, computed based on path costs or other metric
    - distributed (e.g. IP BGP, Ethernet flooding/learning) or central (e.g. SDN)
    - convergence time too long for microsecond-sensitive app's
- Deflection routing: avoid link conflicts (used in some bufferless nets)
- Valiant routing : src  $\rightarrow$  random *intermediate* dest  $\rightarrow$  dest
  - load balances traffic on internal links  $\rightarrow$  avoids hotspots in adversarial patterns
    - tput independent of spatial distribution of traffic pattern; tput of minimal depends..
  - but each packet traverses two times more links
    - extra latency at low loads
    - extra load on internal links for balanced (e.g. all-to-all) patterns

## 5.1 Inverse Multiplexing

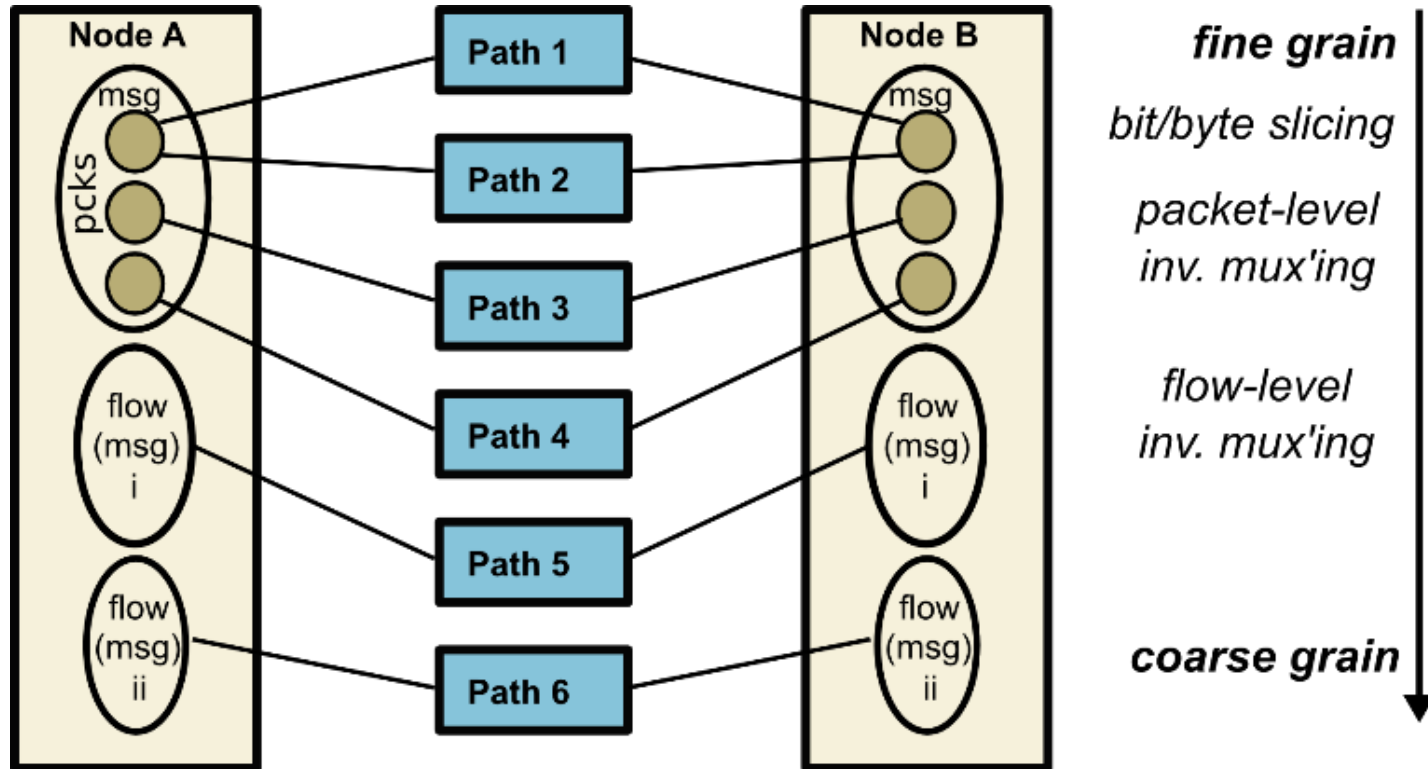
- **What is it?**
  - a (multi-path) routing strategy that spreads the load (packets/flows) equally among all available paths
    - a technique for scalable, non-blocking switching fabrics
- **Generalization of bit/byte slicing**
  - break packets into (“headerless”) slices; forwarded slices synchronously via parallel wires/links or even subnets (Tiny-Tera)
    - same idea: high-tput buffer from many lower-throughput buffers
  - perfect load balancing (equal load on all links, ignoring padding ovrhd) but not practical for distributed implementation (synchronous subnets, central control)
- **“inverse-multiplex” fragments of packets? yes, but header ovrhd**
  - practical only for large packets; done inside some (internally multipath) routers working on “fixed-size” (e.g. 256-byte) segments

# 5.1 Parallelism for High-Throughput: Inverse Multiplexing



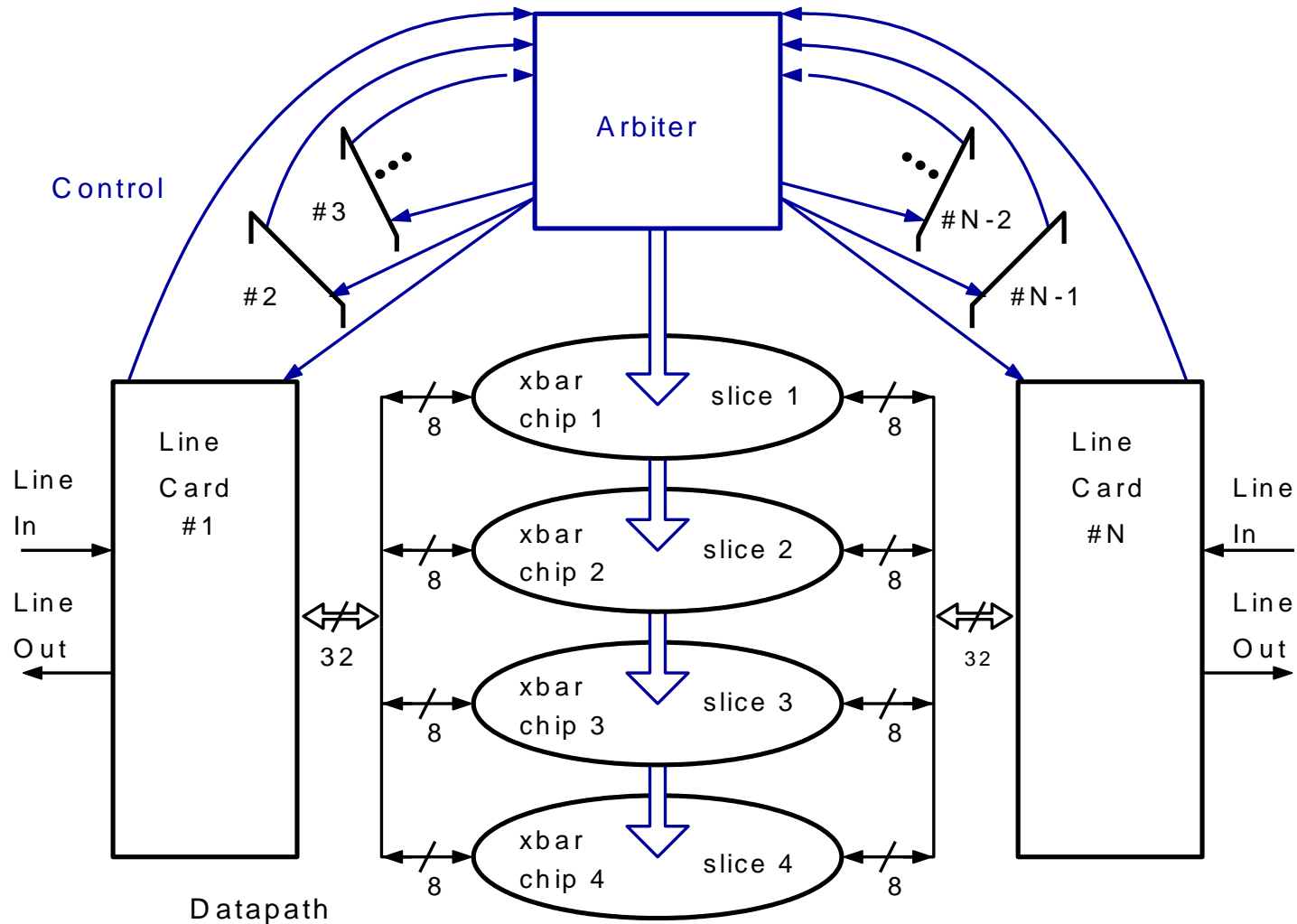
- Parallel wires or network routes for scaling (virtual) “link” throughput up
- Easy: central control, synchronized; Difficult: distributed control, asynch.

## 5.1 Inverse Multiplexing: granularity of load balancing



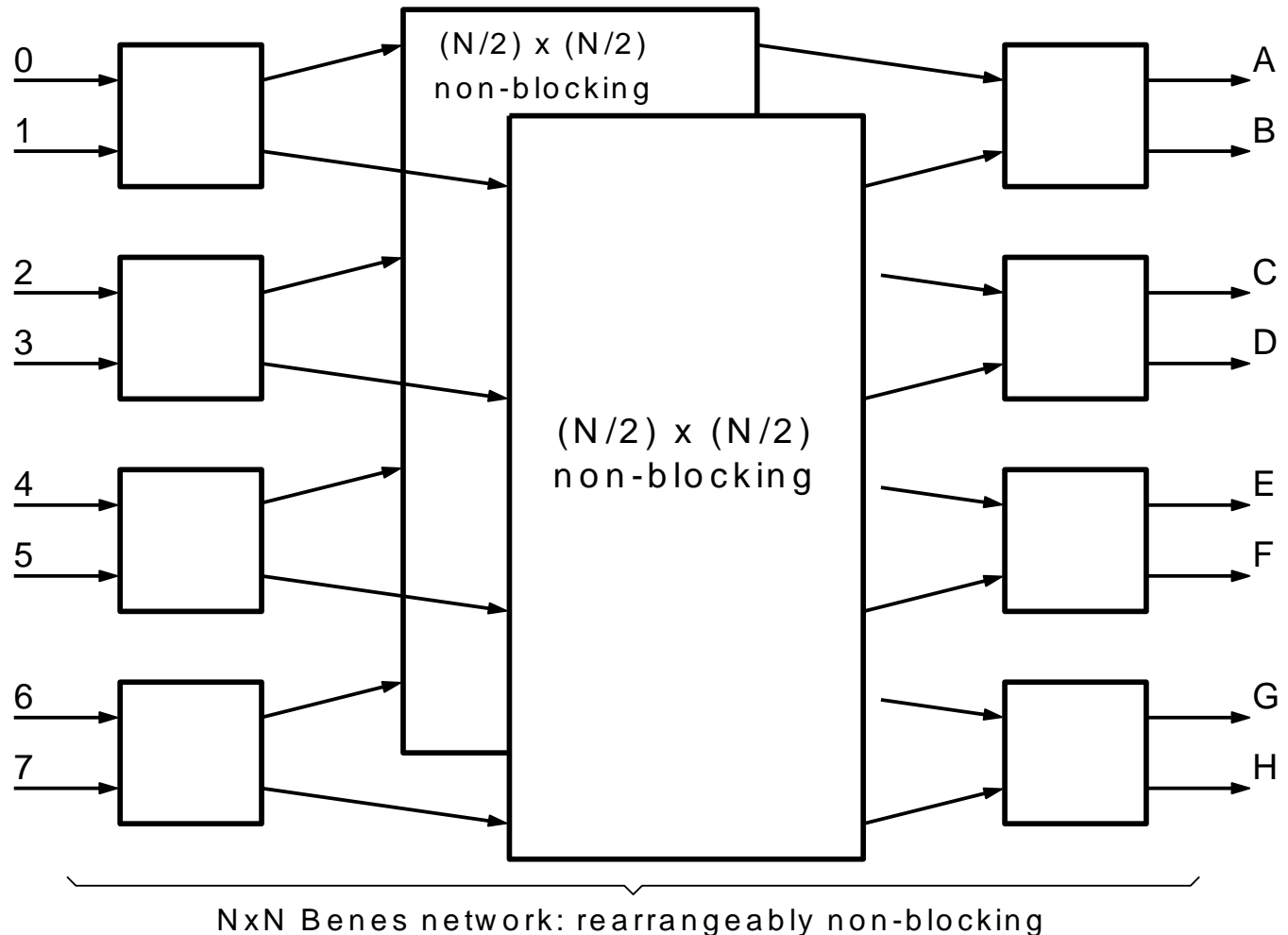
- Fine grain: equalize load on a small time scale
- Coarse grain: danger of overshooting paths (& filling up buffers → delay)

# 5.1 Byte-Slicing: Tiny Tera & other commercial chips



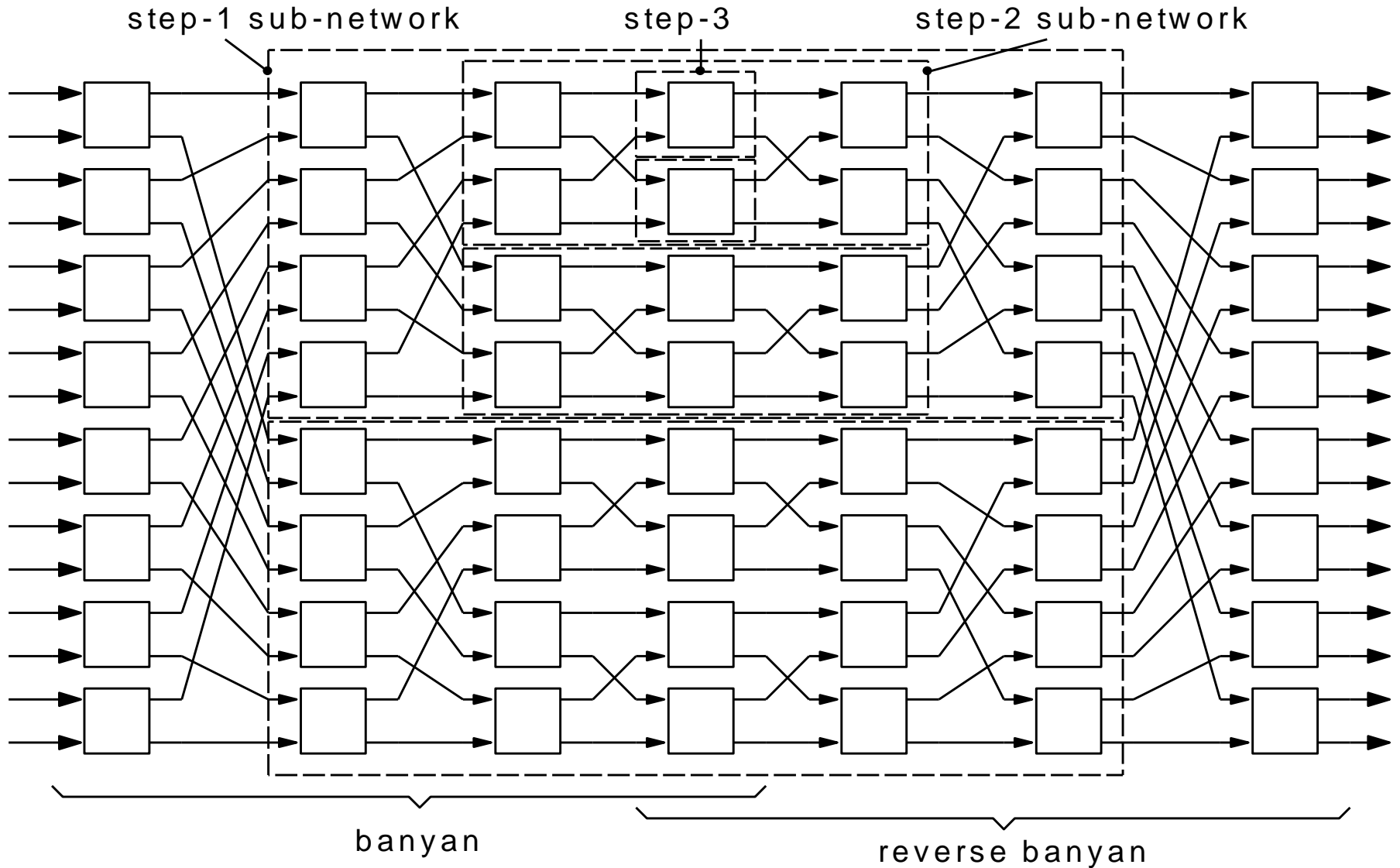
Mckeown e.a.: "Tiny Tera: a Packet Switch Core", IEEE Micro, Jan.-Feb.'97

## 5.2.1 Benes Fabric: Recursive Definition



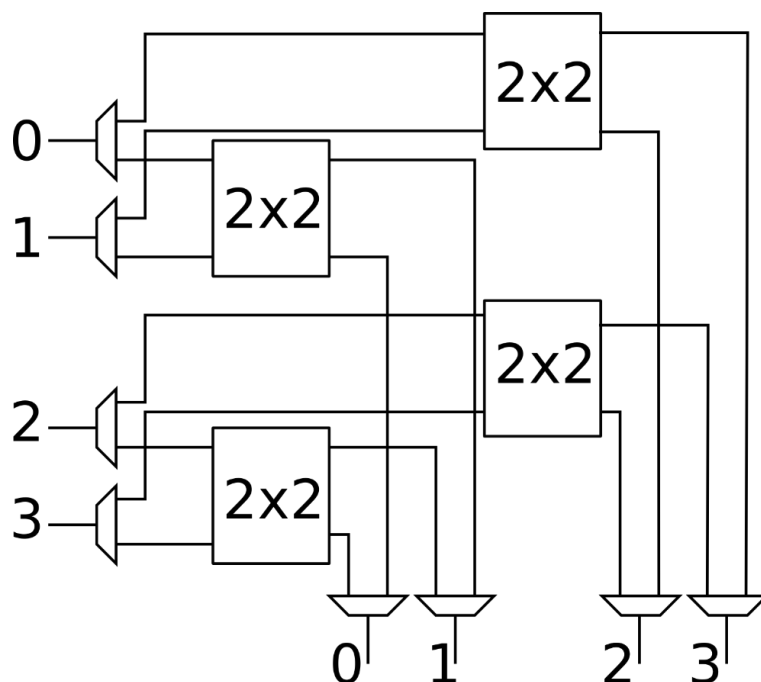
- Goal: reduce switch radix from  $N \times N$  to  $(N/2) \times (N/2)$ : combine ports in pairs
  - Port-pairs require links of twice the throughput: use inverse multiplexing
- ⇒ Use two switches, of half the radix each, in parallel to provide req'd thruput

# Full Construction of 16×16 Benes out of 2×2 Switches



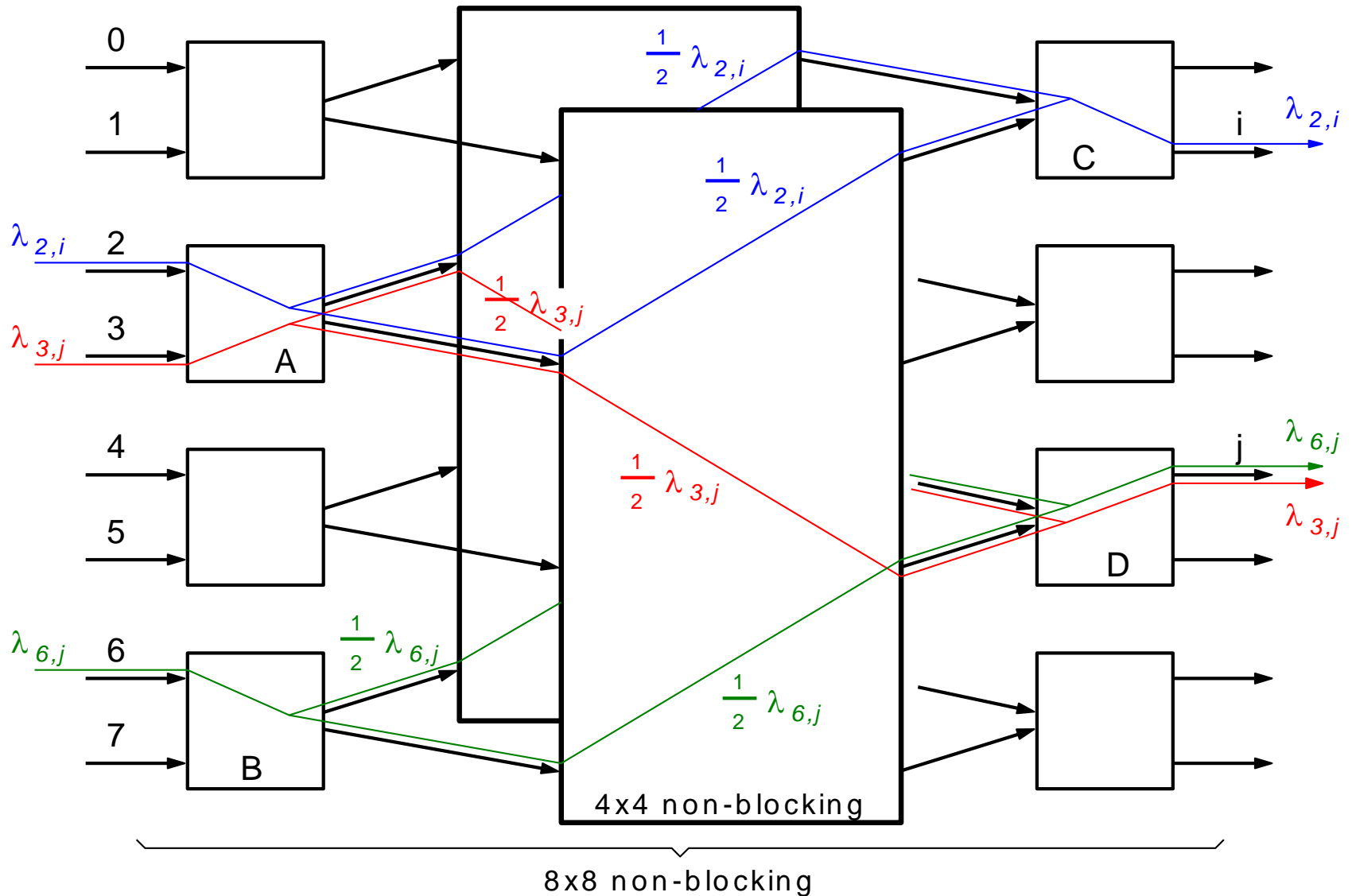


# Hierarchical Crossbars: single-path non-blocking netw.



- No path diversity ... but the network is non-blocking
- $N^2$  crosspoints ... but smaller crossbars  $\rightarrow$  can be implemented in separate chips or chip tiles
- YARC (crossbar) 64x64 switch by Cray Inc., uses 64, 8x8 xbar tiles
  - Scott, Steve, e.a. “The blackwidow High-Radix Clos Network.” ACM SIGARCH Computer Architecture News. vol. 34, no. 2, 2006.

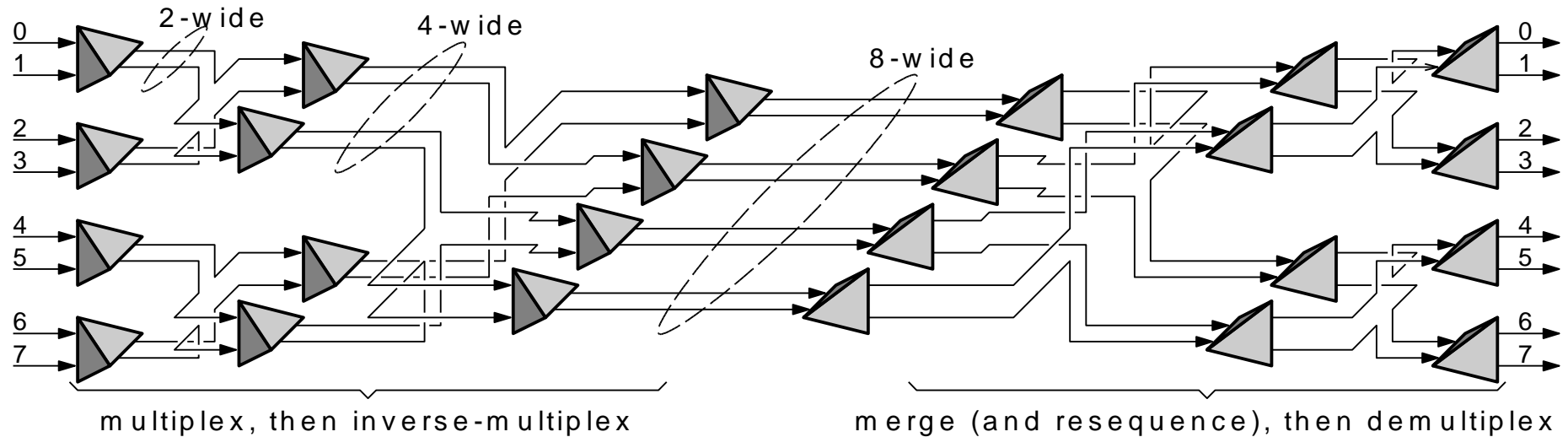
# Inverse Multiplexing for Non-Blocking Operation



# Per-Flow Inverse Mux'ing for Non-Blocking Operation

- Prove that overall  $N \times N$  network is non-blocking, i.e. *any* feasible external traffic  $\Rightarrow$  feasible rates on all internal links
- All traffic entering switch A is feasible, hence of aggregate rate  $\leq 1+1 = 2$ ; it is split into two halves  $\Rightarrow$  each of rate  $\leq 1$   
 $\Rightarrow$  traffic entering each  $(N/2) \times (N/2)$  subnetwork is feasible
- It does not suffice to balance (equalize) the *aggregate* load out of switch A – must equally distribute *individual* (end-to-end) flows – *per-flow* inverse multiplexing
  - $\Rightarrow$  each of  $\lambda_{2,i}$ ;  $\lambda_{3,j}$ ;  $\lambda_{6,j}$  is individually split in two equal halves
  - $\Rightarrow$  the sum of  $\lambda_{3,j} + \lambda_{6,j}$  is also split in two equal halves
- All traffic exiting switch D is feasible, hence of aggregate rate  $\leq 1+1 = 2$ ; it enters D in two equal halves  $\Rightarrow$  each of rate  $\leq 1$   
 $\Rightarrow$  traffic exiting each  $(N/2) \times (N/2)$  subnetwork is also feasible

## Conceptual View of 8x8 Benes: Virtual Parallel Links using Inverse Multiplexing



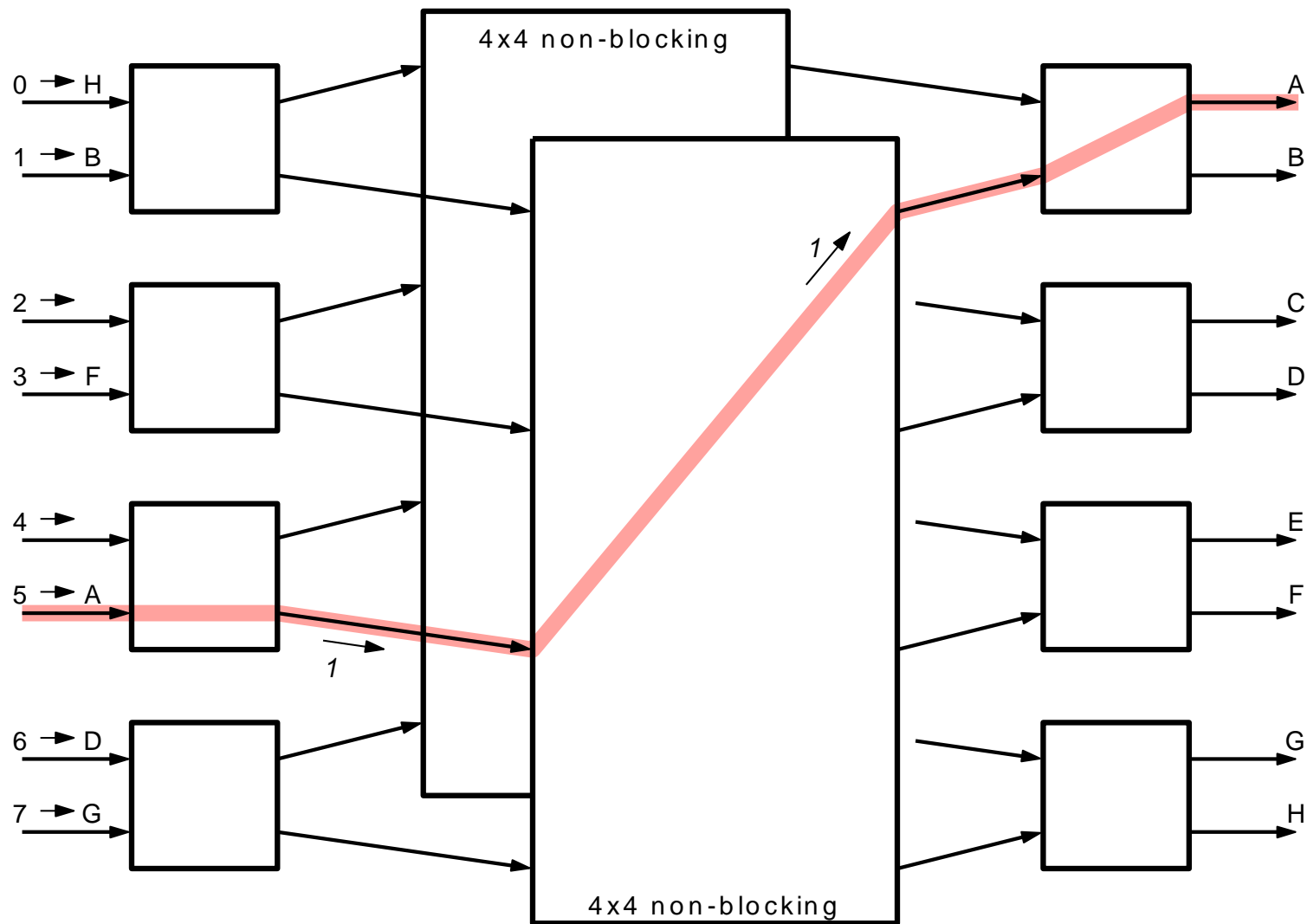
## Methods to implement (per-flow) Inverse Multiplexing

- Per-Flow Round-Robin, at packet granularity
  - for each flow, circularly and per-packet alternate among routes
  - requires maintaining per-flow state
  - danger of synchronized RR pointers: pck bursts to same route
  - alternative: arbitrary route selection, provided the (per-flow) imbalance counter has not exceeded upper bound value

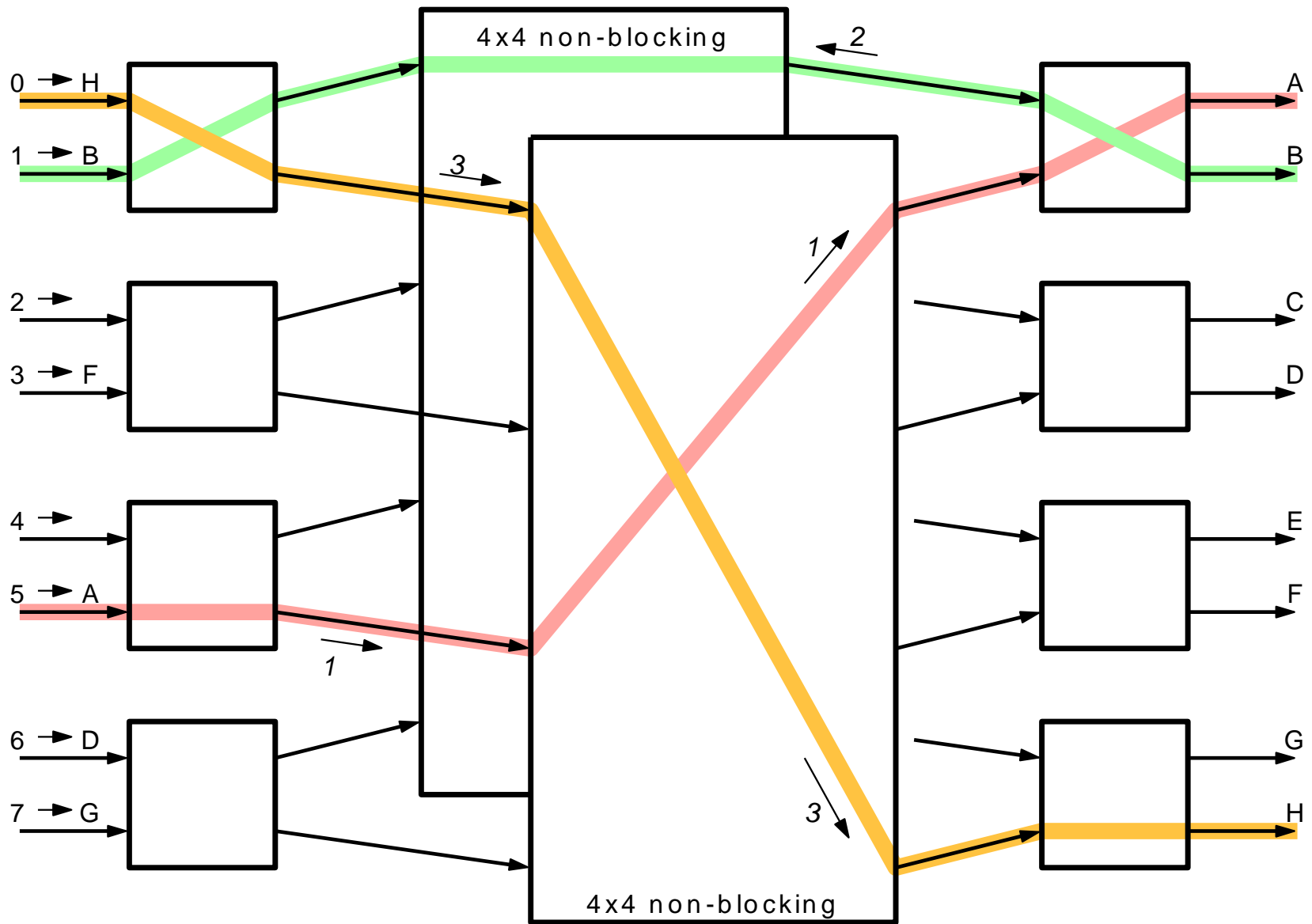
## Methods to implement (per-flow) inverse multiplexing (continued)

- Adaptive Routing, at packet granularity – usu. Indiscriminate
  - chose the route with least-occupied buffer (max. credits)
  - + does not maintain or use per-flow state
  - per-flow load balancing only “after-the-fact”, when buffers fill up
- Randomized Route Selection, at packet granularity
  - + does not require maintaining per-flow state
  - load balancing is approximate, and long-term
- **Packet Resequencing** (when needed): major cost of inv.mux'ng
  - Chiussi, Khotimsky, Krishnan: IEEE GLOBECOM'98
- Hashed Route Selection at entire Flow Granularity
  - route selection based on hash function of flow ID
  - + all packets of given flow through same route  $\Rightarrow$  in-order delivery
  - poor load balancing when small number of flows

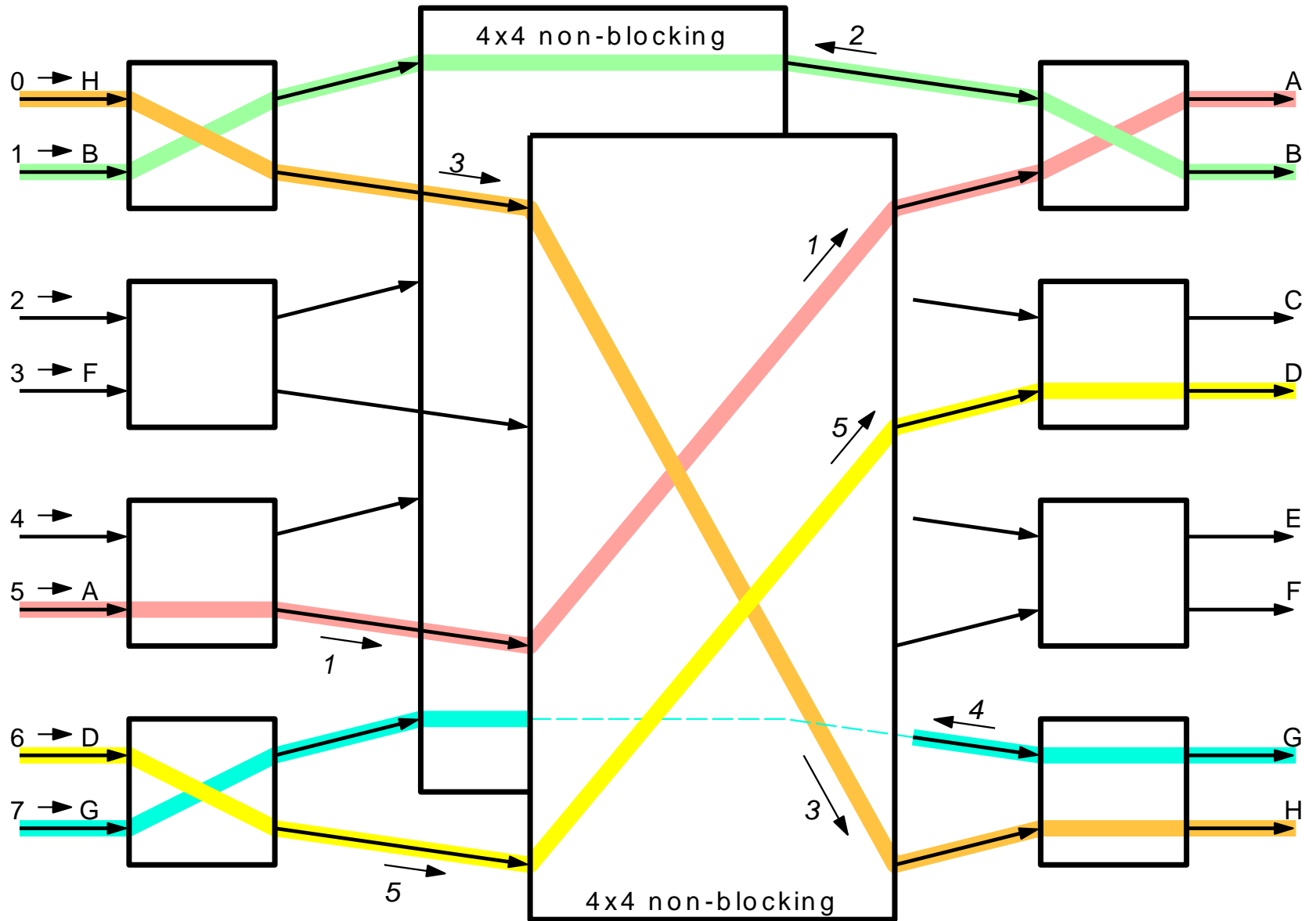
# Benes Net under Telephony-Ckt Connection Requests



- Circuit Connections: Start from an input, use one of the subnets

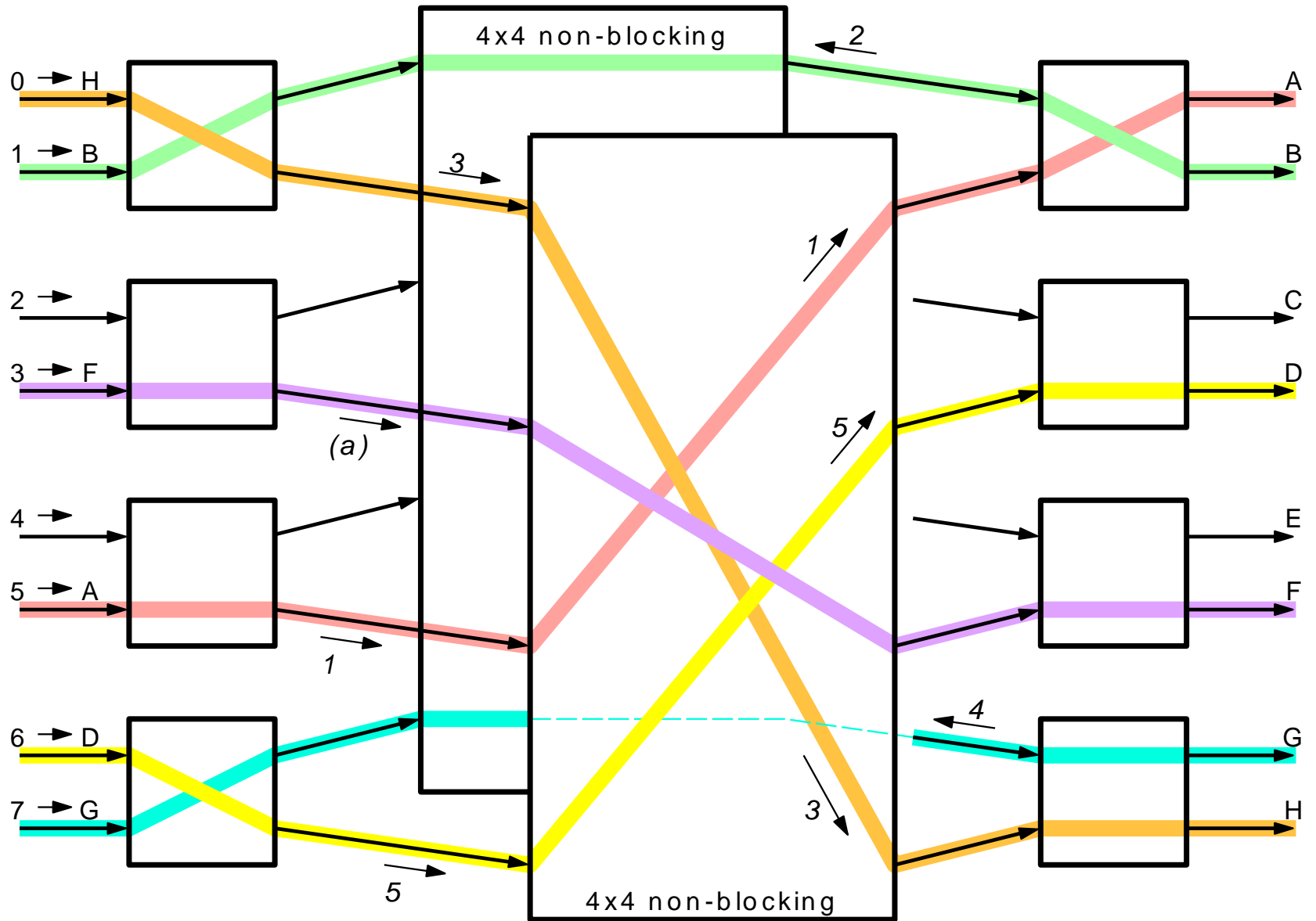


- Continue from the brother port of the output, then the brother of the input



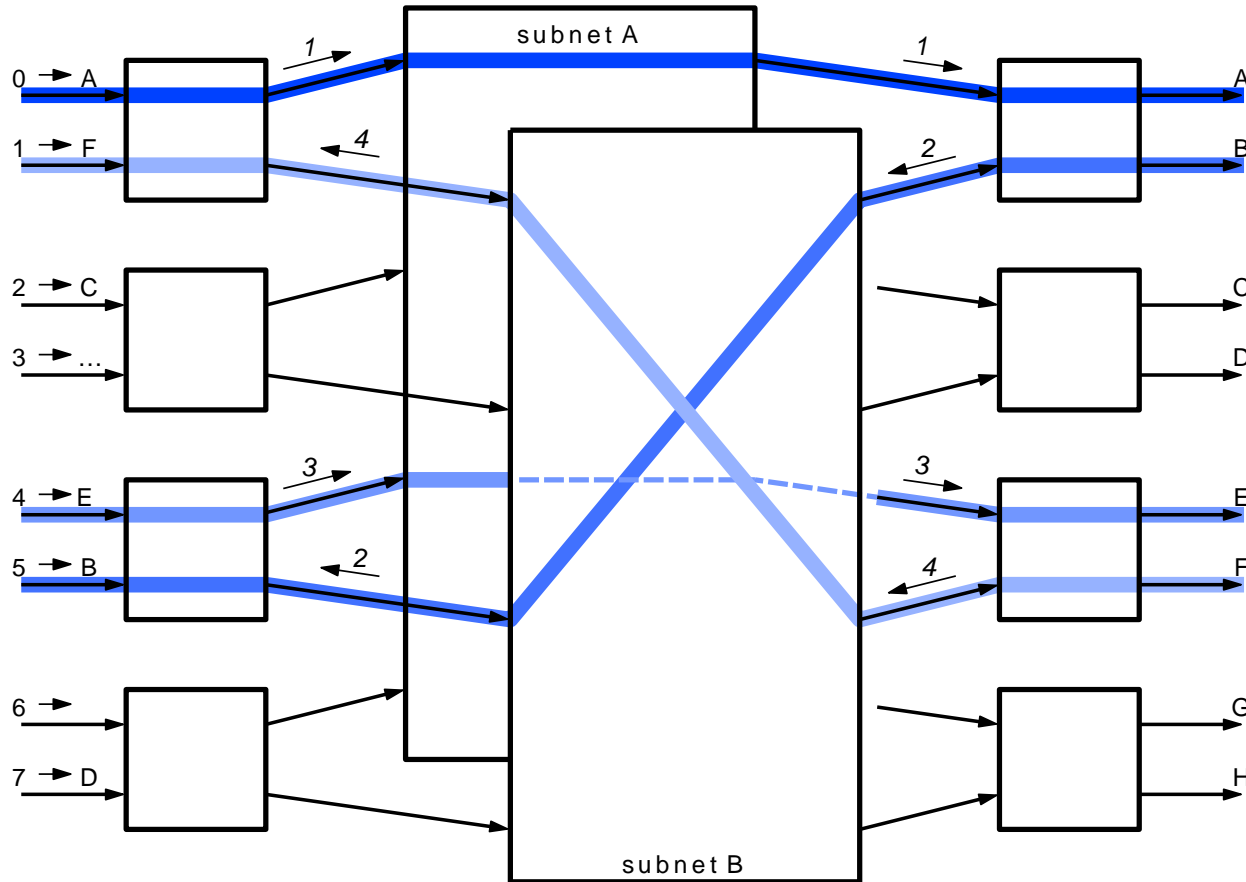
- Keep “threading” output and input switches, till closing or no-connection





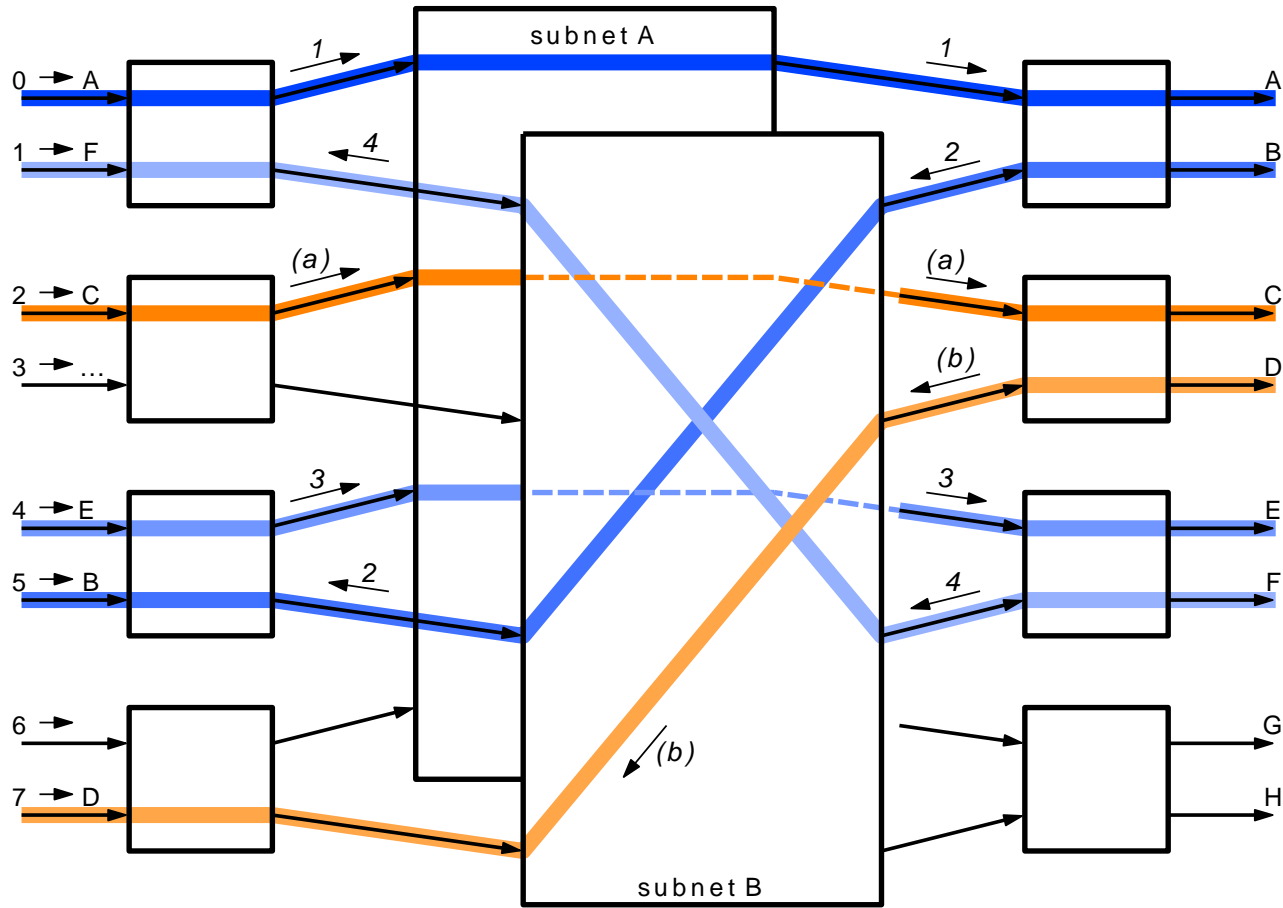
- Start a new “thread” (a) from an unconnected input, till completing all conn.

# (A) Thread termination on input side (1 of 2)



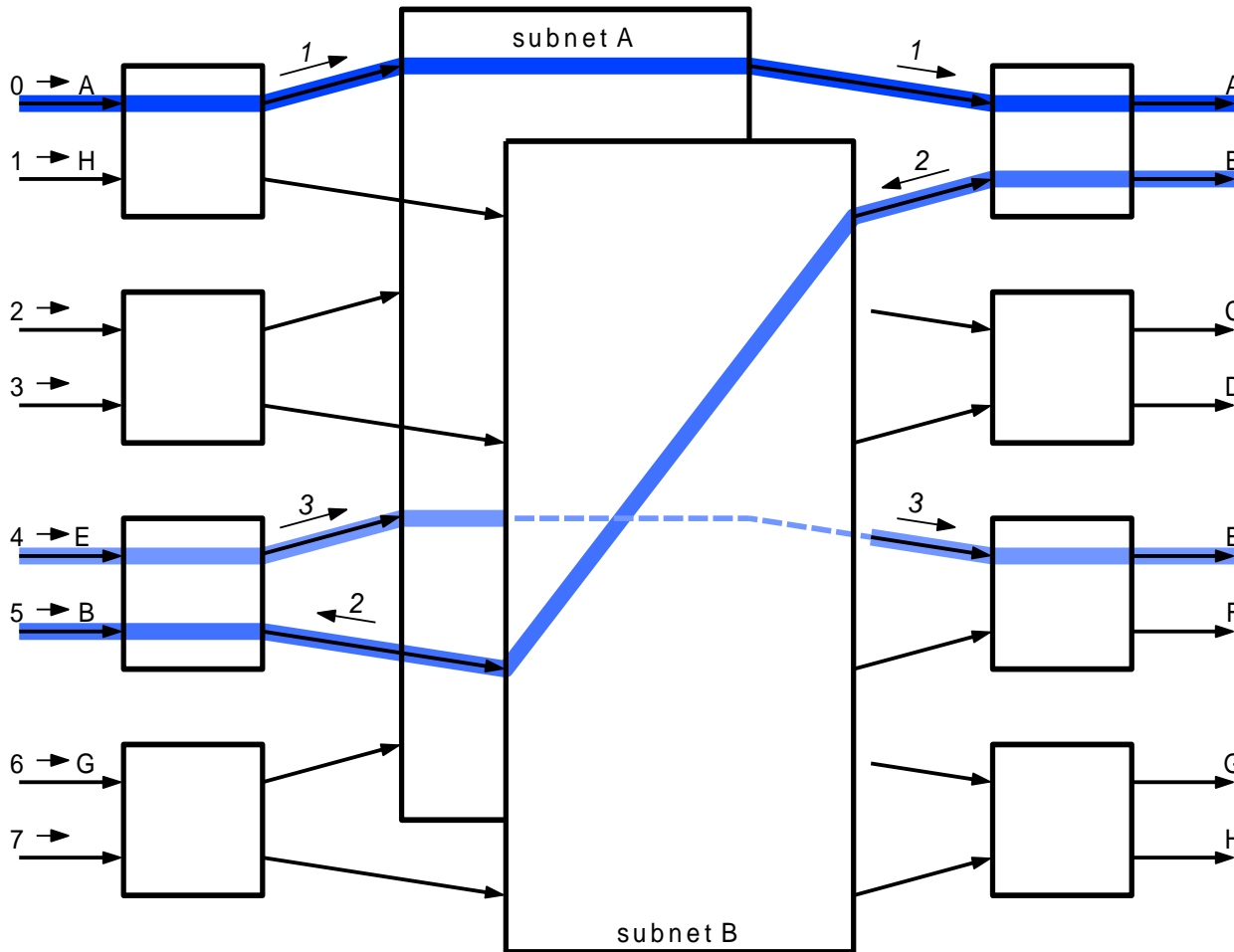
- Threads always start on the input side
- If a thread terminates on the input side:
  - all touched output switches are completely connected
  - concerning touched input switches:
    - (1) if thread closes, all are complete,
    - ...

# (A) Thread termination on input side (2 of 2)



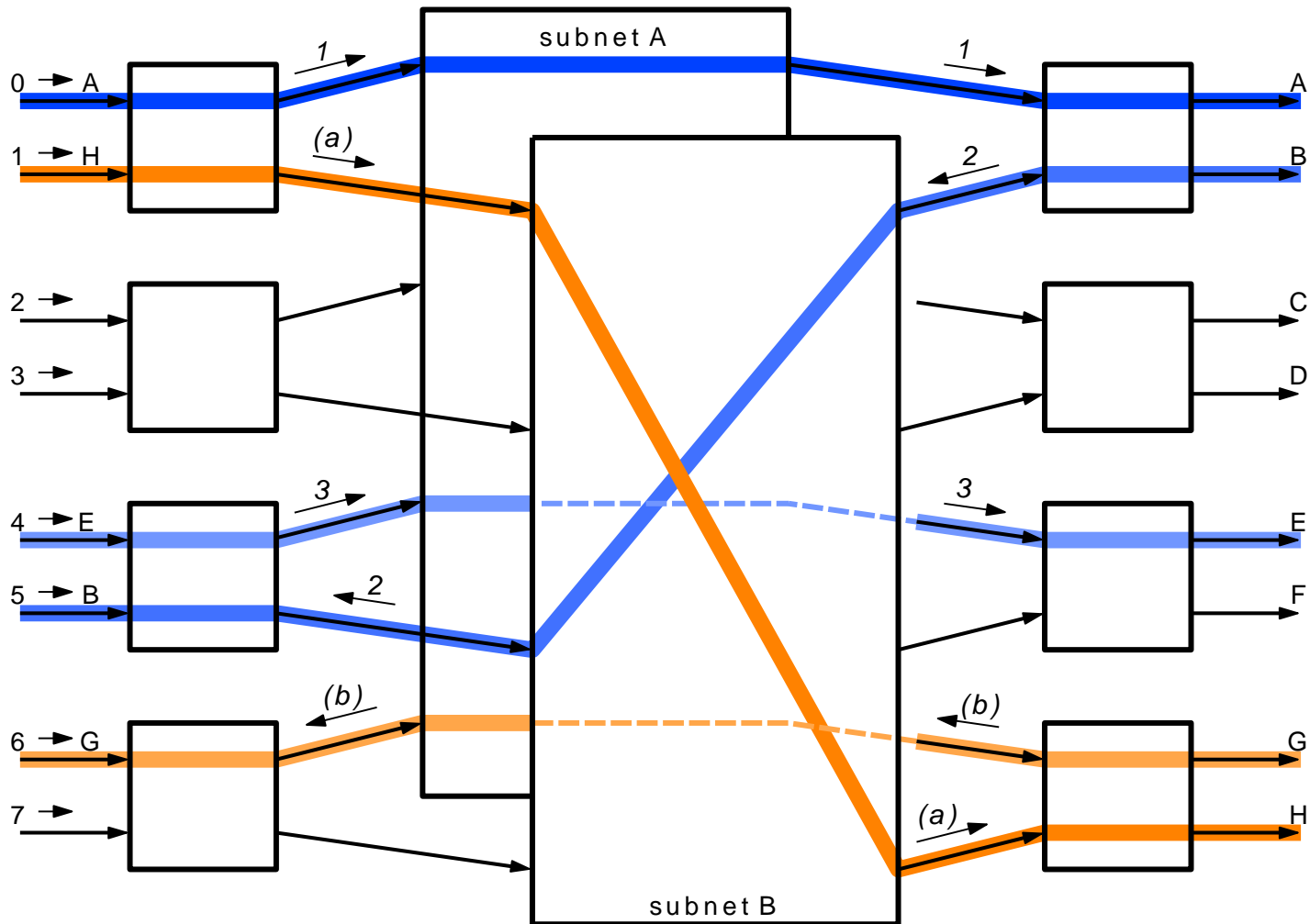
- Threads always start on the input side
- If a thread terminates on the input side:
  - all touched output switches are completely connected
  - concerning touched input switches:
    - (1) if thread closes (4), all are complete,
    - (2) if thread terminates on half-used input (b): all touched input switches are complete, except the first one, which is half-covered by this thread

## (B) Thread termination on output side



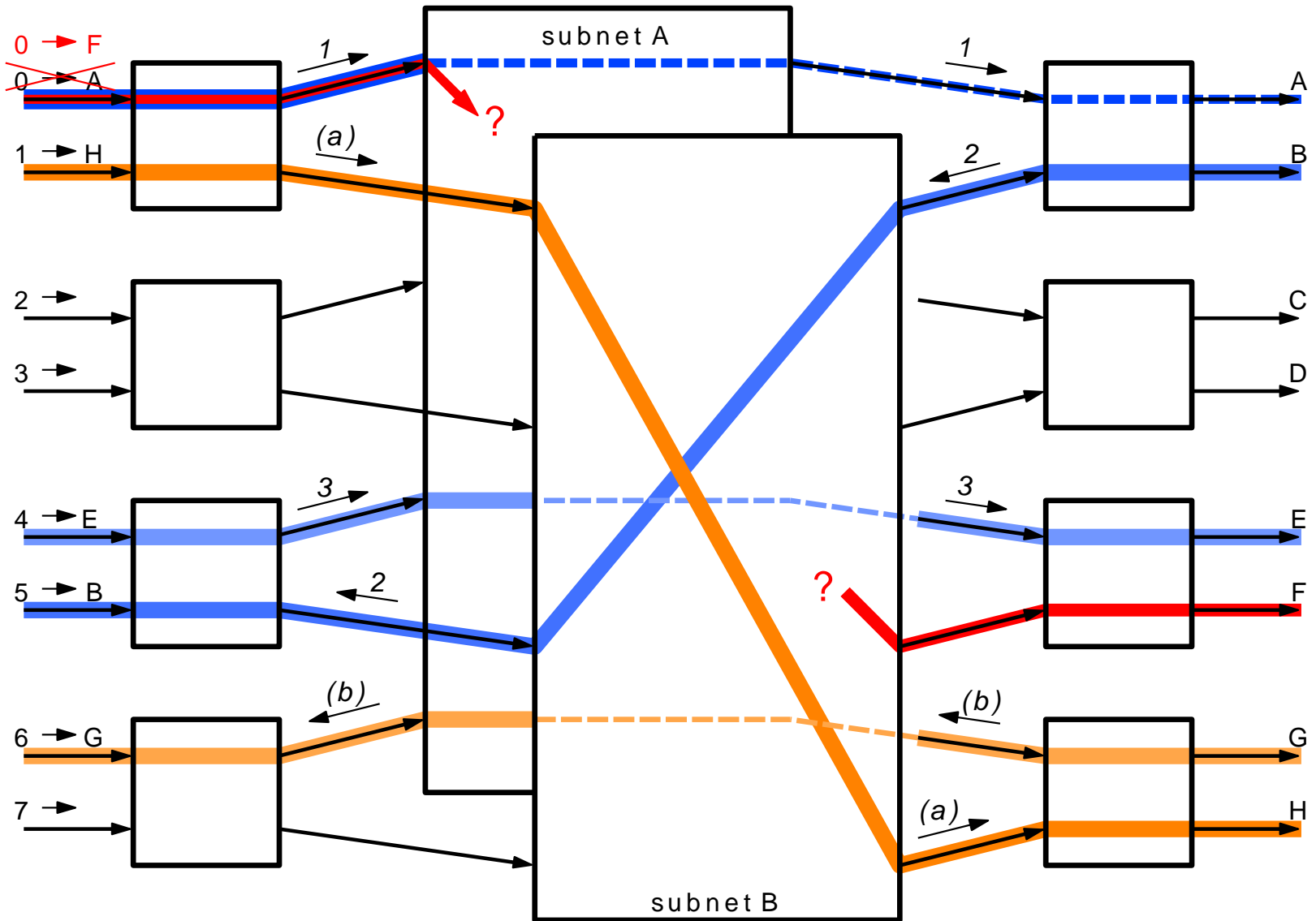
- Threads always start on the input side
- If a thread terminates on the output side:
  - all touched output switches are completely connected
  - the first touched input switch is half-covered

## (C) Completing half-covered input switches



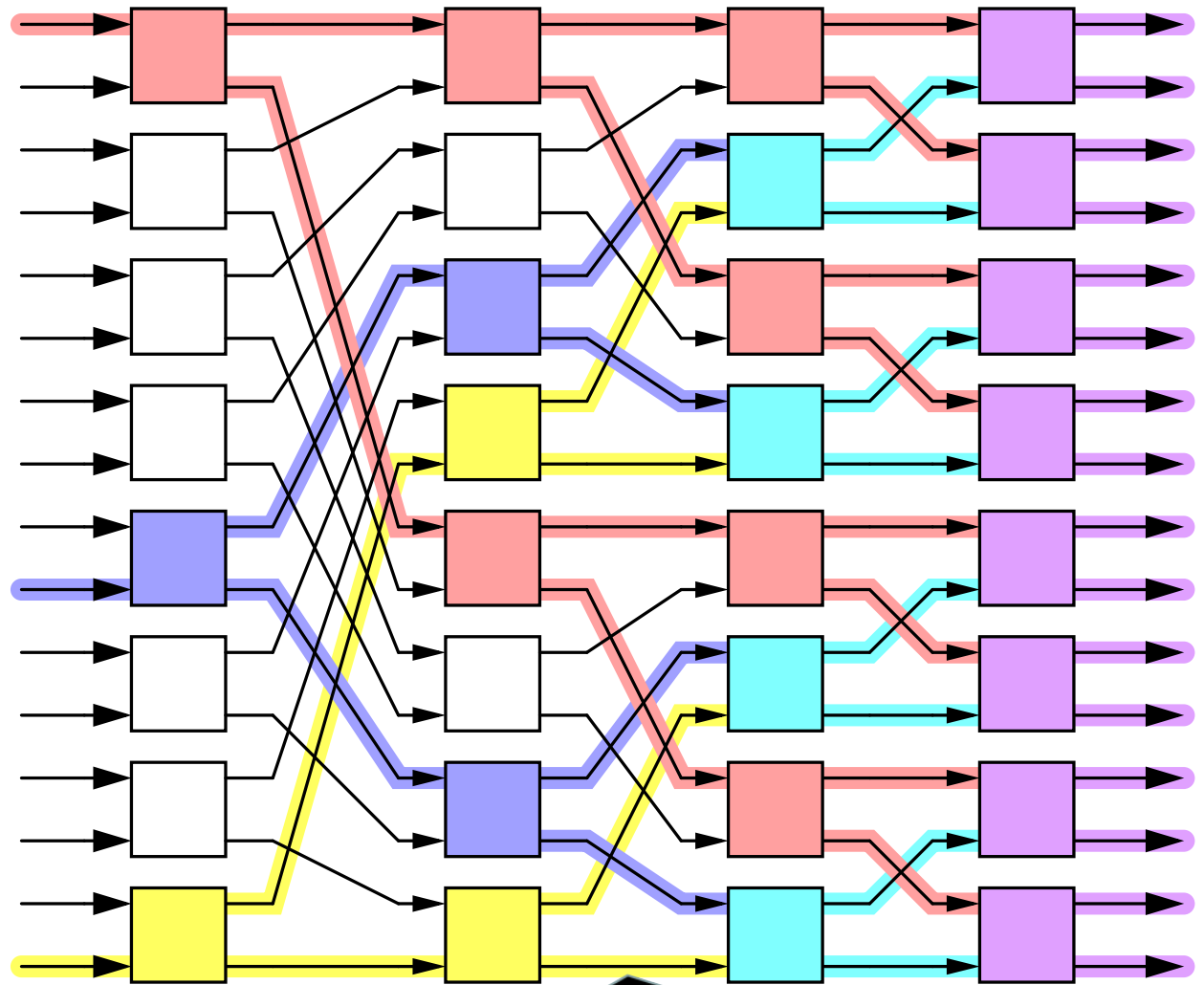
- New threads always start from a half-covered input switch, if there is one  
 $\Rightarrow$  all threads cover all out-sw's they touch, in-sw's are covered in sequence

# Benes Fabric: *Rearrangeably* Non-Blocking



## 5.2.2 The Banyan (Butterfly) Network

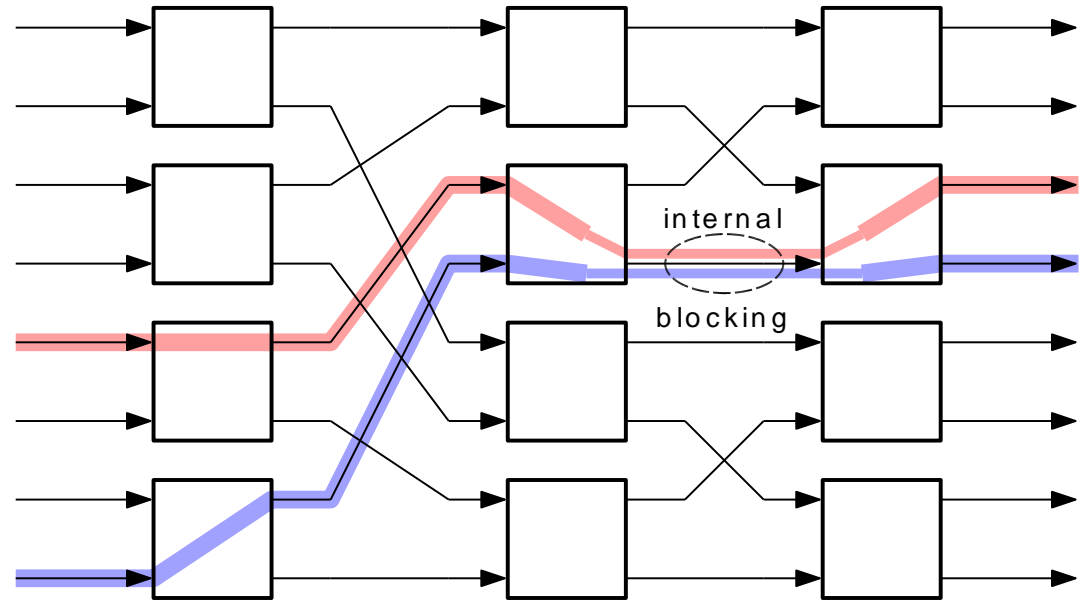
- Single route from given input to given output
- Each input is the root of a tree leading to all outputs
- Trees share nodes
- (Similarly, outputs are roots of trees feeding each from all inputs)
- for  $N \times N$  network made of  $2 \times 2$  sw.:
- $\log_2 N$  stages, of
- $N/2$  sw. per stage



Up to 4 port-pair flows may use this link --  $\sqrt{N}$  in general

# The banyan network is internally blocking

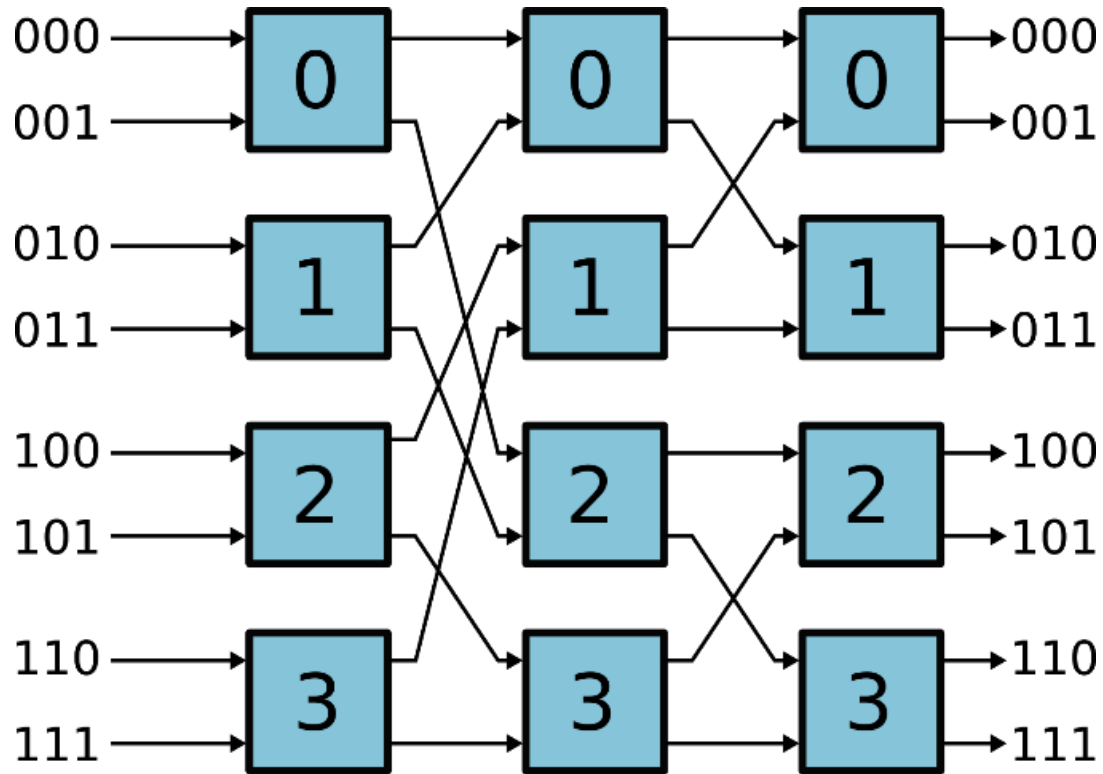
- Consider circuits: each  $\lambda_{i,j}$  is either 1 or 0: single connection per port – “telephony” style
- There are  $N!$  such circuit connection patterns for a  $N \times N$  network – each is a permutation of the numbers  $(1, 2, \dots, N)$



- Any network containing  $(N/2) \cdot \log_2 N$  or less  $2 \times 2$  switches (like the banyan does) has to be internally blocking, because it can only be placed into less than  $N!$  states, hence cannot route all  $N!$  existing sets of con. req's
- Each  $2 \times 2$  switch can be placed in 2 different states; a network containing  $(N/2) \cdot \log_2 N$  such switches can be placed into  $2^{(N/2) \cdot \log_2 N} = N^{(N/2)}$  different states;  $N^{(N/2)} = N \cdot (N/2)^{(N/2)-1} \cdot 2^{(N/2)-1} < N \cdot [(N-1) \cdot \dots \cdot (N/2+1)] \cdot [(N/2) \cdot \dots \cdot 2] = N! \Rightarrow$  not enough states

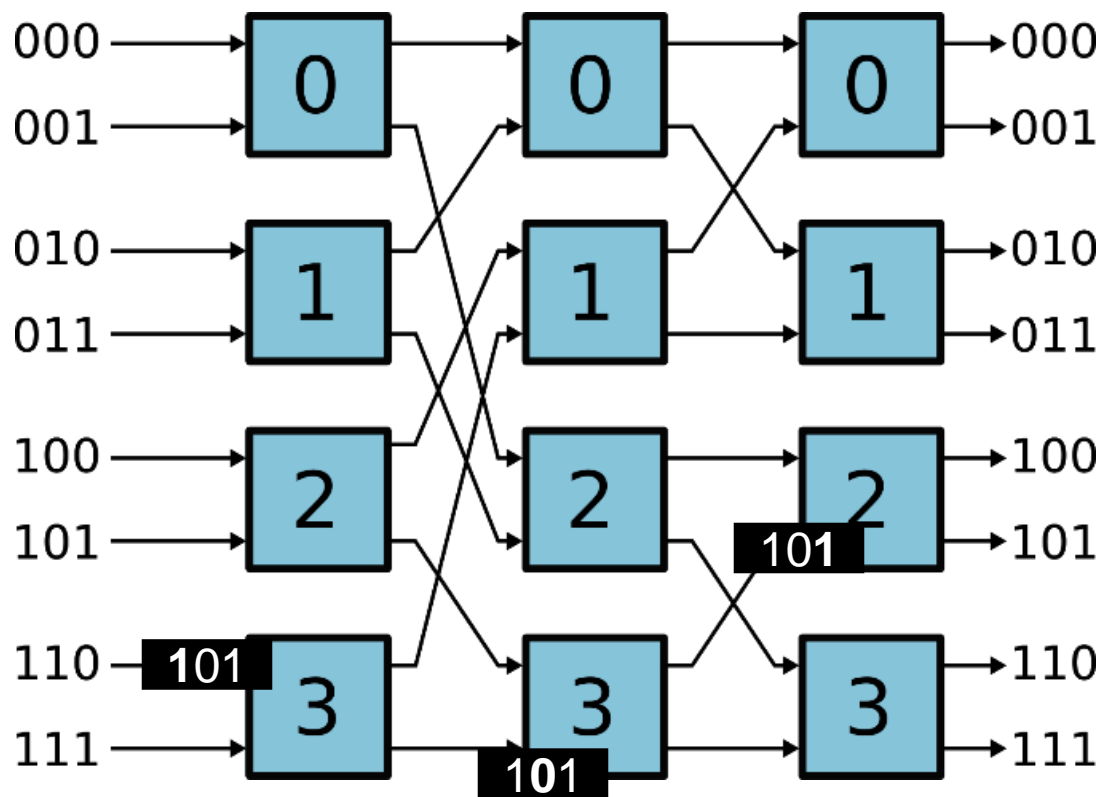


## Butterfly (or $k$ -ary $n$ -fly) Network



- $k$  = switch radix = number of switch ports
- $n$  = number of stages
- Total number of ports =  $k^n$ 
  - frequently called “banyan networks”

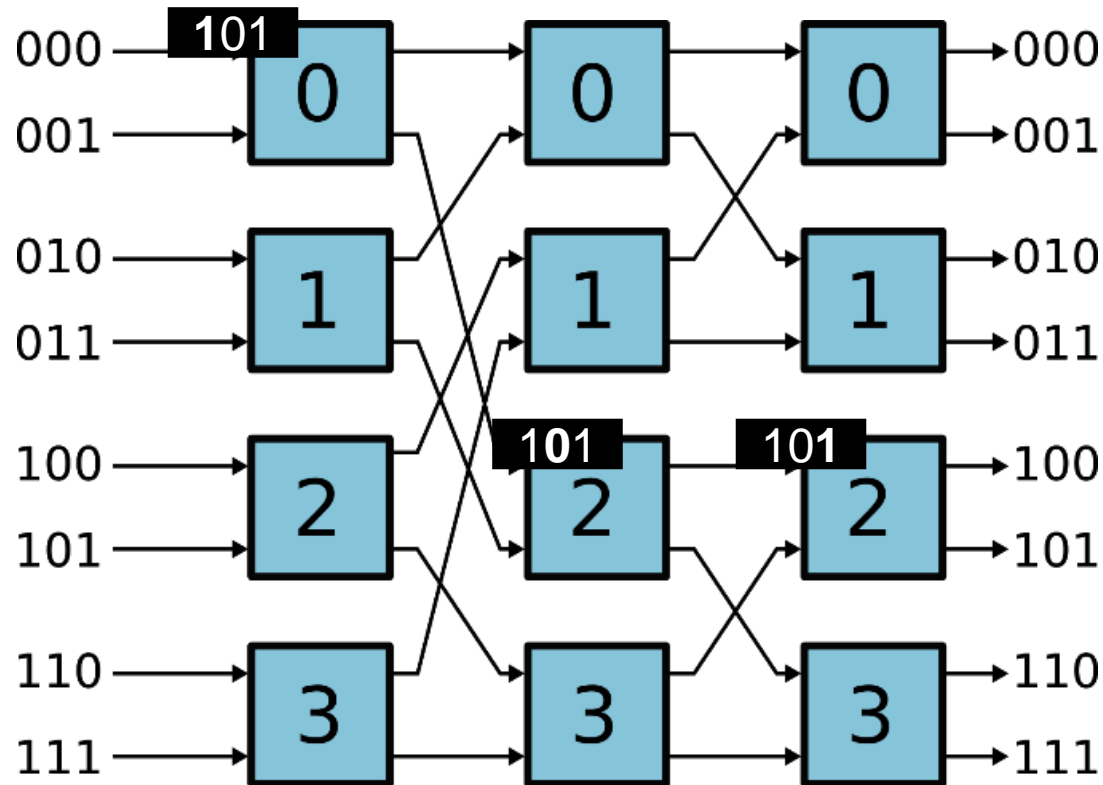
## Butterfly Networks Are Self-Routing



**Variant 1**

- $\log_2(N)$  stages,  $\log_2(N)$  bits in destination ID
- Each stage uses one destination bit for routing purposes
  - if 0 route up, if 1 route down
- No need for routing tables: packets are self-routed

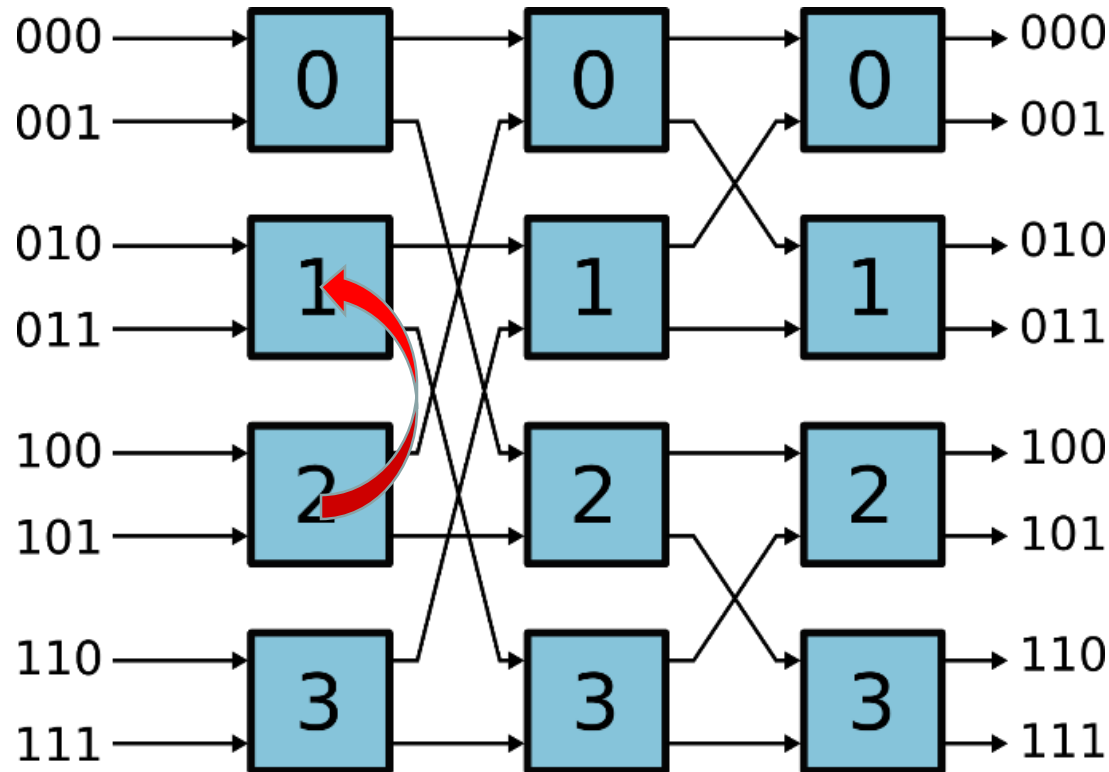
# Routing in Butterfly Networks



**Variant 1**

- $\log_2(N)$  stages,  $\log_2(N)$  bits in destination ID
- Each stage uses one destination bit for routing purposes
  - “0” route up, “1” route down
- No need for routing tables: packets are self-routed

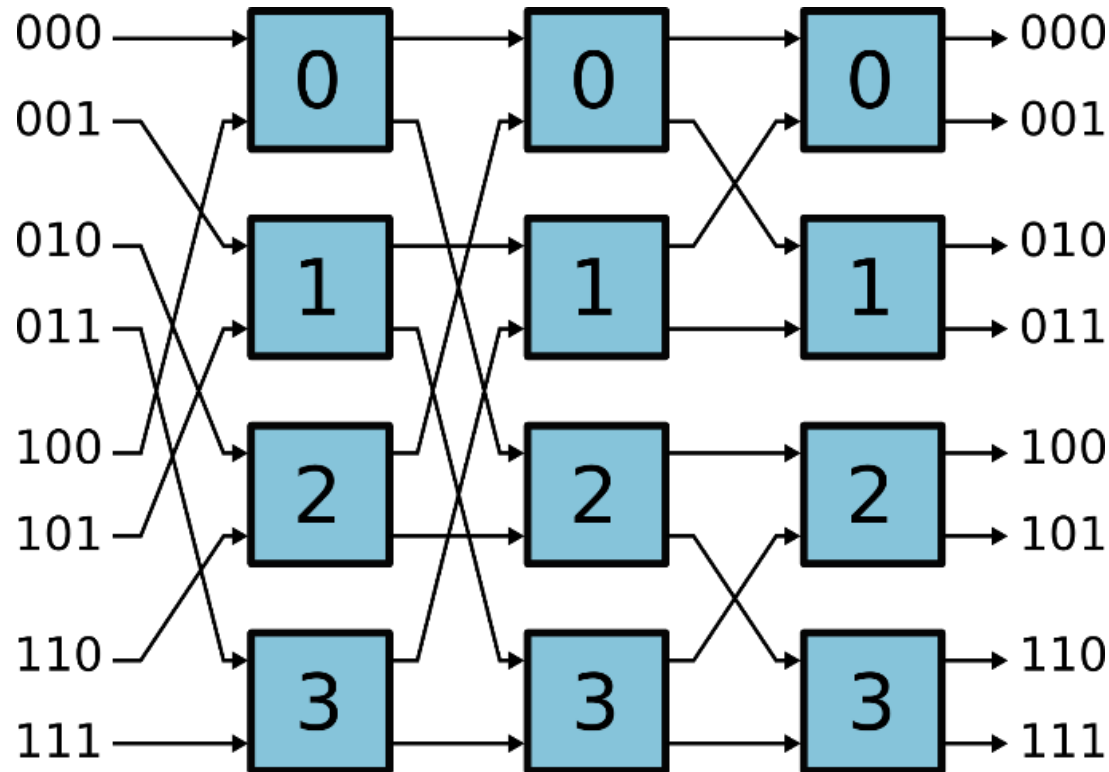
# Banyan & Butterfly Are Isomorphic



**Variant 2**

- Topologically equivalent network (isomorphic)
  - interchange 1<sup>st</sup>-stage nodes 1 and 2 → variant 1
    - do not move inputs (left side) of 1<sup>st</sup> stage nodes
    - equivalently, move inputs together with 1<sup>st</sup> stage nodes, and then shuffle them

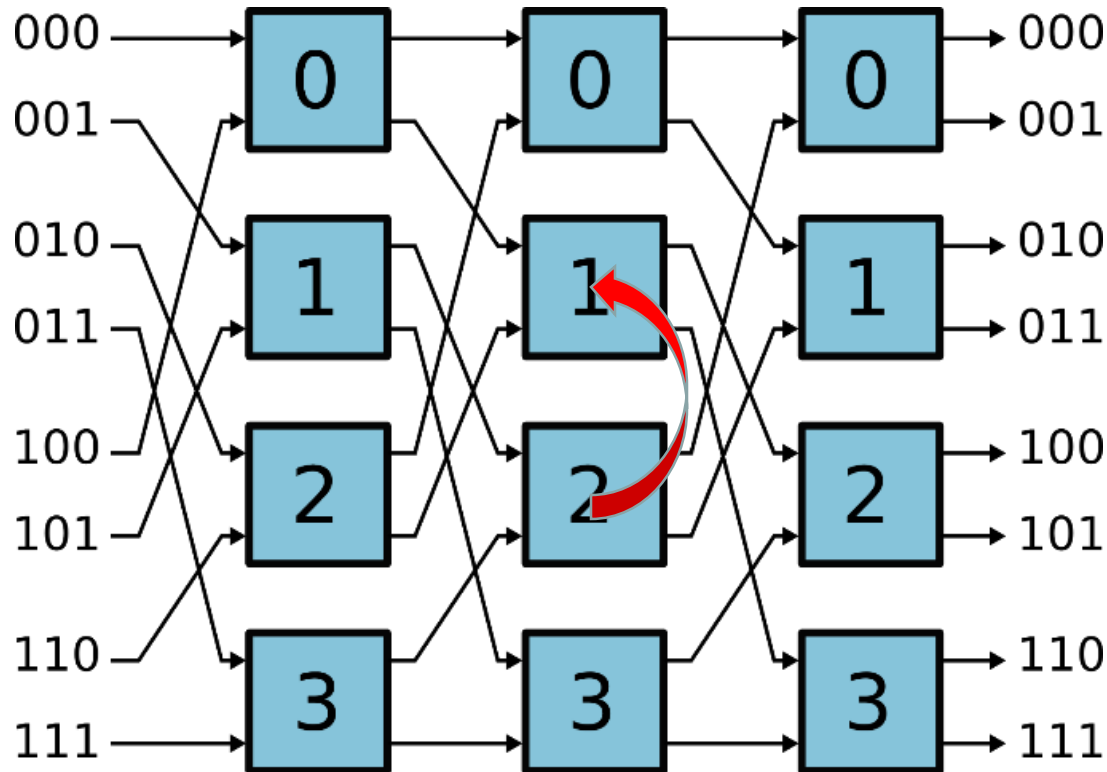
## Shuffling The Input Ports



**Variant 3**

- Interchange inputs using the perfect shuffle
- Perfect shuffle - bitwise operation: shift left by 1, e.g. 100 → 001
  - “cards of the “lower” deck perfectly interleaved with those of the upper one”
- Can route any “monotonically increasing” permutation

# The Omega Network



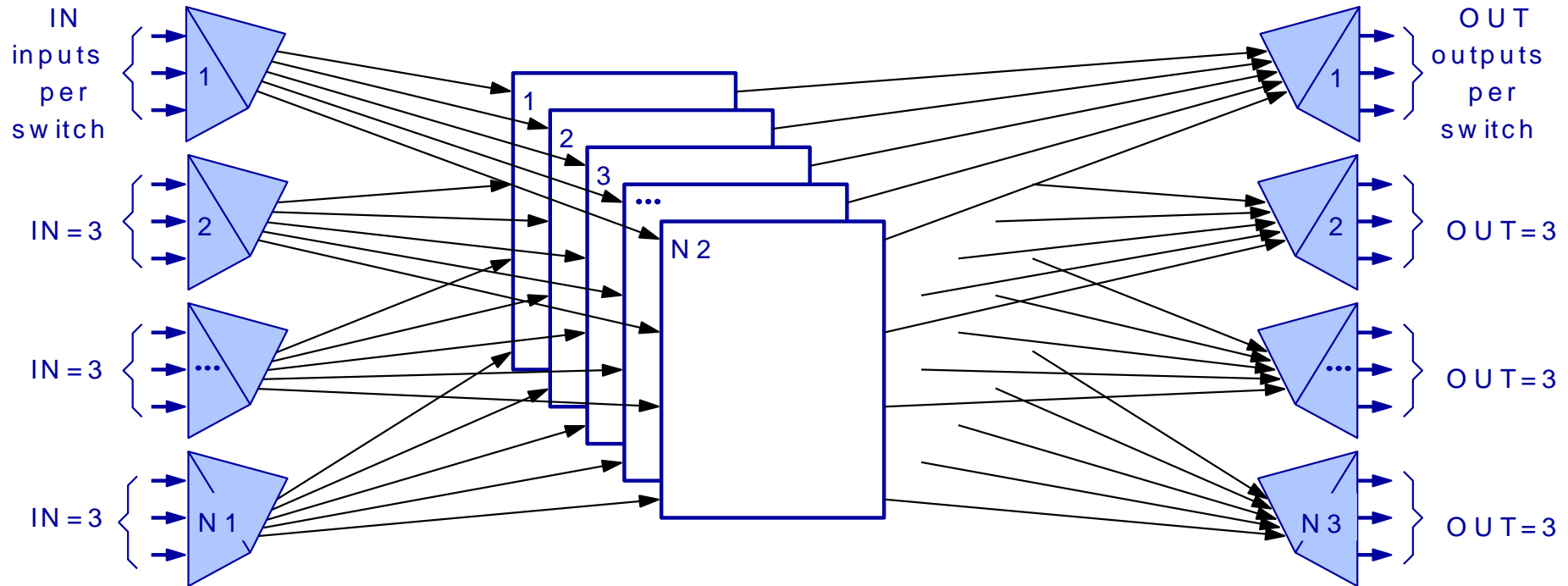
**Variant 4**

- The outputs of one stage are connected to the inputs of the next using the perfect shuffle permutation (circular shift to the left by one)
  - interchange 2<sup>nd</sup>-stage nodes 1 and 2 to obtain variant 3
    - move inputs of nodes (routers) as well

## Which is the lowest-cost non-blocking fabric?

- $N \times N$  Benes network, made of  $2 \times 2$  switches:
    - $2 \cdot (\log_2 N) - 1$  stages (2 banyans back-to-back, 1 shared stage)
    - $N/2$  switches per stage  $\Rightarrow$  total switches =  $N \cdot (\log_2 N) - N/2$
    - number of states that the Benes network can be in =  $2^{\#\text{switches}} = 2^{N \cdot (\log_2 N) - N/2} = (2^{\log_2 N})^N / 2^{N/2} = N^N / 2^{N/2} = [N \cdot \dots \cdot N] \cdot [(N/2) \cdot \dots \cdot (N/2)] > N \cdot (N-1) \cdot \dots \cdot 2 \cdot 1 = N! \Rightarrow$  Benes has more states than the minimum required for a net to be non-blocking
    - Benes was seen to be non-blocking: (i) circuits and the “threading” algorithm, (ii) packets and inverse multiplexing
    - “rearrangeably” non-blocking: in a partially connected network, making a new connection may require re-routing existing ones
  - Impossible for any network with about half the switches of the Benes (e.g. banyan) to be non-blocking (# of states)
- $\Rightarrow$  Benes is probably the lowest-cost practical non-blocking fabric

## 5.2.3 Clos Networks (generalization of Benes nets)



5-parameter Network:  $(IN, N_1, N_2, N_3, OUT)$

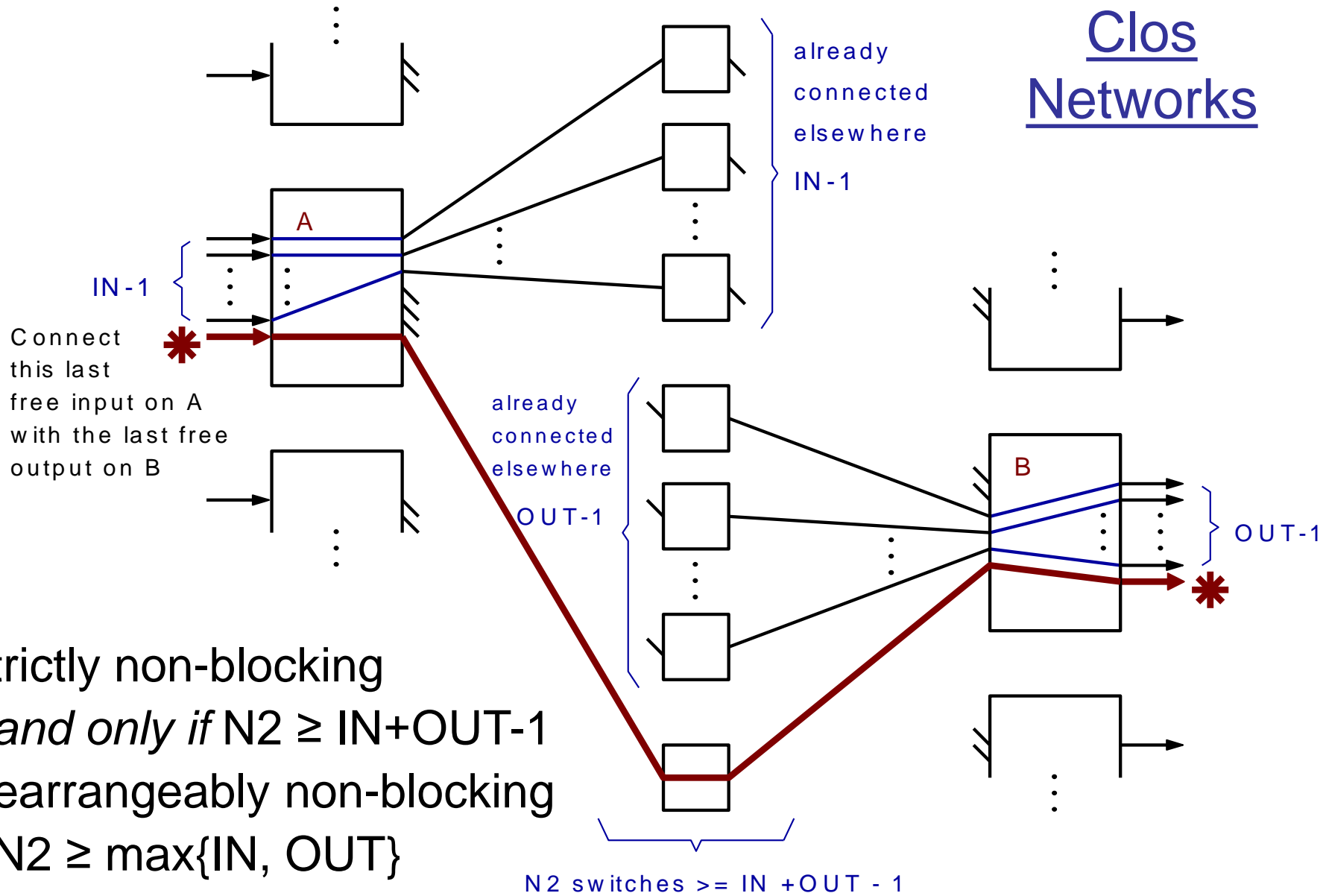
this example: the  $(3, 4, 5, 4, 3)$  Clos Network

usually:  $IN = OUT$ , and  $N_1 = N_3$

other times,  $IN=IN_1=N_2=N_3=OUT = \text{sqrt}(\text{number of Clos ports})$



# Clos Networks



Connect this last free input on A with the last free output on B

already connected elsewhere  
IN-1

already connected elsewhere  
OUT-1

OUT-1

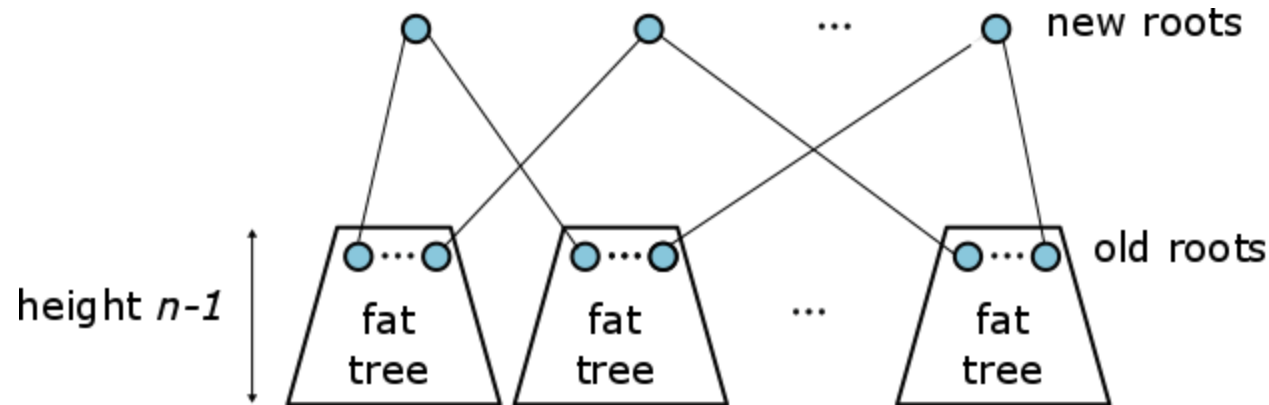
$N_2 \text{ switches} \geq IN + OUT - 1$

- Strictly non-blocking  
*if and only if*  $N_2 \geq IN + OUT - 1$
- Rearrangeably non-blocking  
*if*  $N_2 \geq \max\{IN, OUT\}$

## 5.2.4 Fat Trees: recursive definition

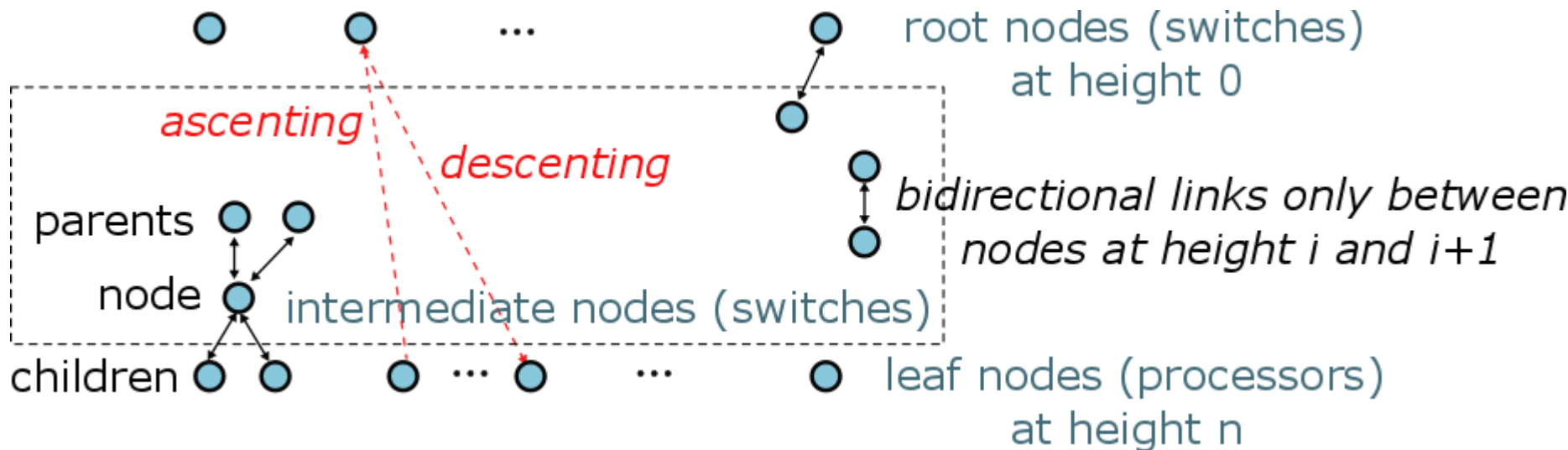
● root

- A fat tree of height 0 consists of a single vertex  
– the root of the tree



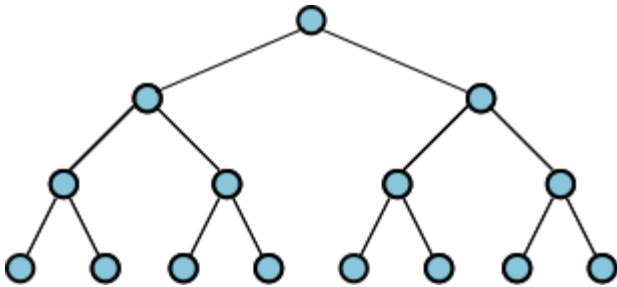
- If we have a set of (sub) fat trees of height  $n-1$  and we *arbitrarily* connect their roots to a set of a new (vertices) roots  $\rightarrow$  fat-tree of height  $n$

## 5.2.4 Fat Trees: properties & routing

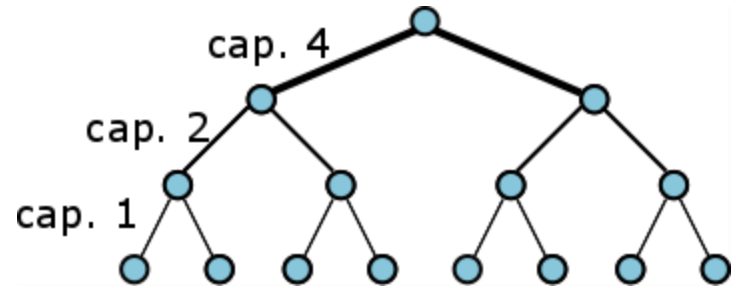


- At least one path from each leaf node to every root
  - Bidirectional
- Non-minimal routing: route up an arbitrary root node, then route down to destination
- Minimal routing : route up to closest common ancestor, then down

## 5.2.4 Fat Trees: single-root trees

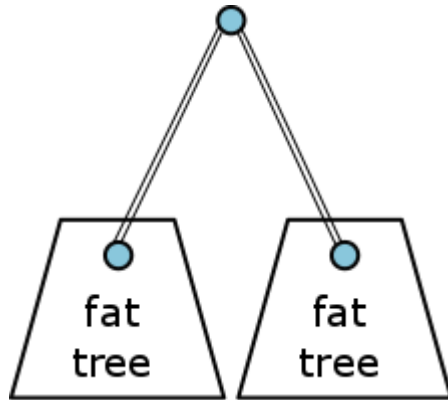


- (ultra) Slim trees
  - poor bisection bandwidth
  - constant switch radix

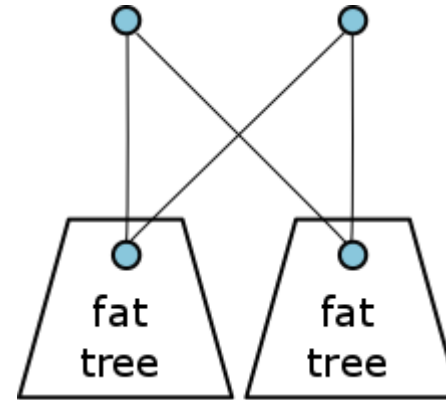


- (Fully-fledged) Fat trees
  - full bisection bandwidth
    - capacity to/from children = capacity to/from parents
  - switch radix increases as we move towards the root

## 5.2.4 Fat Trees: single root versus multiple roots



- Single root fat trees
  - the radix of switches increase



- Same bisection bandwidth w. lower radix switches
  - Can be built with constant radix switches

## 5.2.4 Multi-root trees: $k$ -ary $n$ -trees



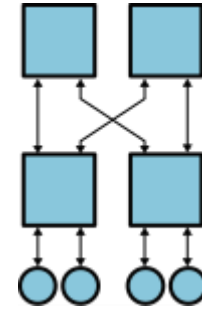
- Switches have  $2k$  ports (but root nodes may have less)
  - 2-ary 0-tree

## 5.2.4 Multi-root trees: $k$ -ary $n$ -trees



- Switches have  $2k$  ports (but root nodes may have less)
  - 2-ary 1-tree

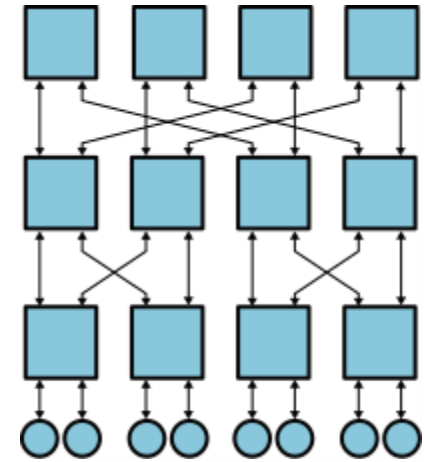
## 5.2.4 Multi-root trees: $k$ -ary $n$ -trees



- Switches have  $2k$  ports (but root nodes may have less)
  - 2-ary 2-tree



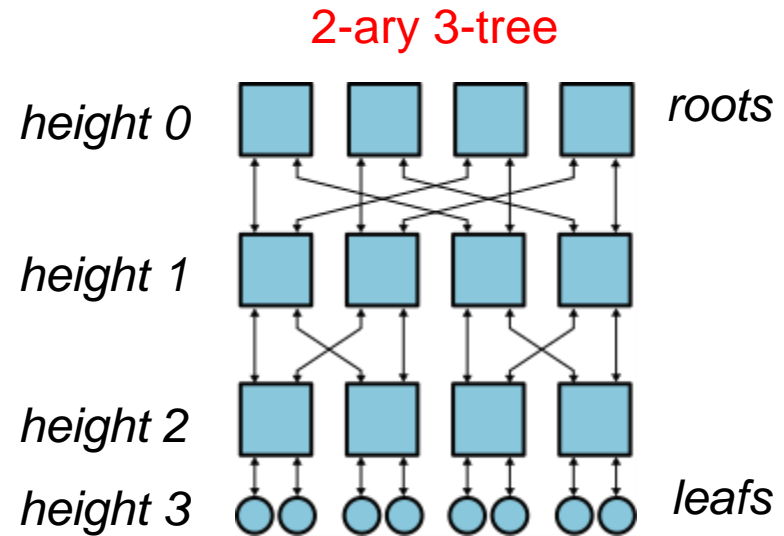
## 5.2.4 Multi-root trees: $k$ -ary $n$ -trees



- Switches have  $2k$  ports (but root nodes may have less)
  - 2-ary 3-tree

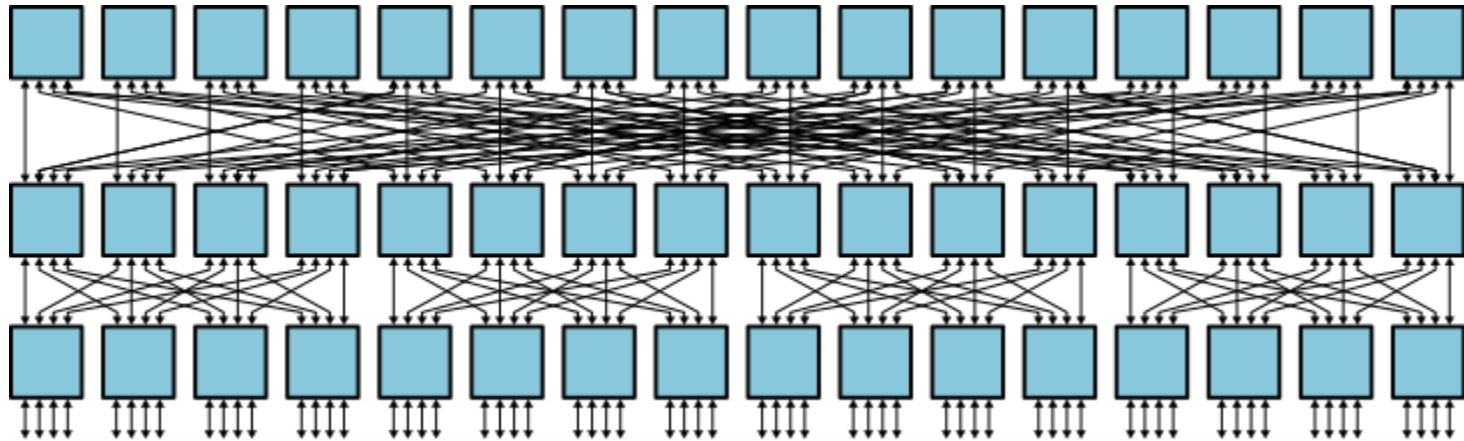
## 5.2.4 Multi-root trees: $k$ -ary $n$ -trees

- $2k \times 2k$  switches, tree height  $n$
- $k^n$  leaf nodes (processors)
- $(n-1) k^{n-1}$  switches in total
  - $k^{n-1}$  switches per stage
- $nk^n$  bidirectional links in total



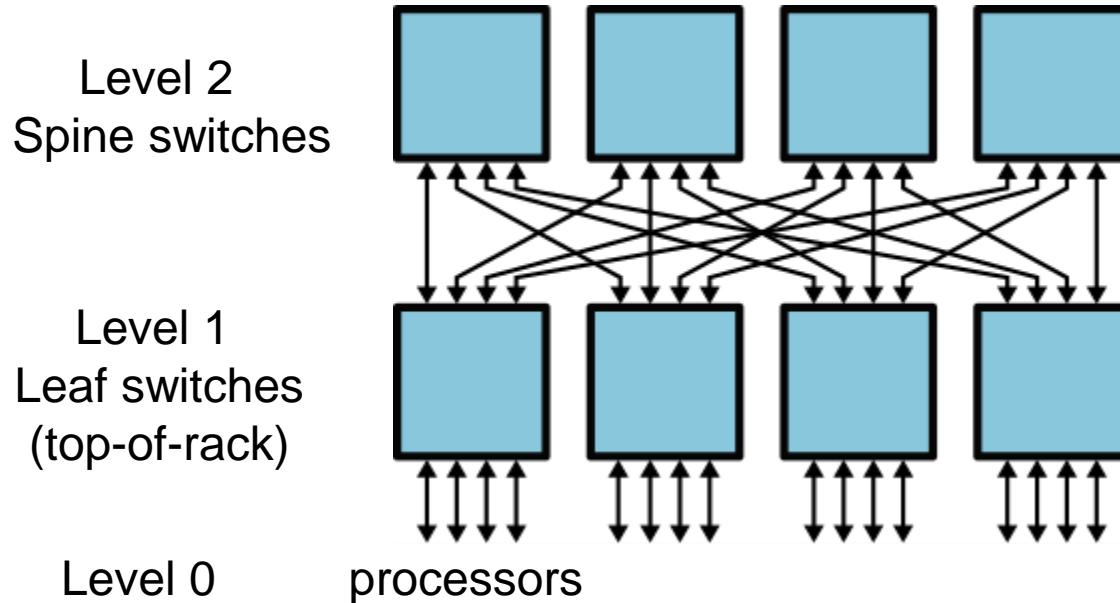
- **Relationship with banyan**
  - $k$ -ary  $n$ -tree = bidirectional  $k$ -ary  $n$ -fly
  - transforming a fly into a tree, the radix of switches doubles

## 5.2.4 Multi-root trees: $k$ -ary $n$ -trees



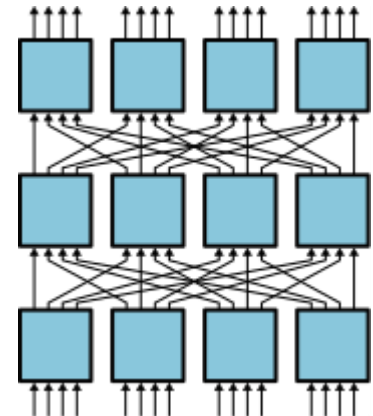
- 4-ary 3-tree
  - 64 ports, 8 ports per switch, 64x3x2 unidirectional links
- 4-ary 3-fly
  - 64 ports, 4 ports per switch, 64x4 unidirectional links

## 5.2.4 Spine-leaf (folded-Clos)

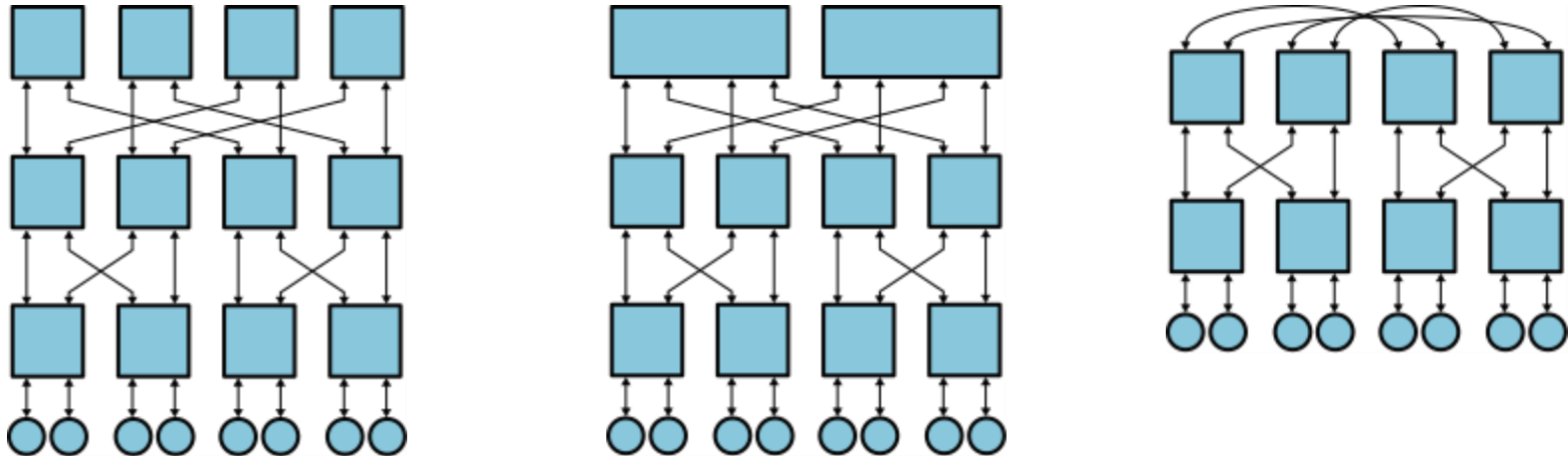


*Spine nodes are the roots of the fat tree, and leaf nodes are the switches at height 1 – in  $k$ -ary  $n$ -tree terminology, leaf nodes are the processors*

- Spine-leaf network is a 2-level fat-tree (*4-ary 2-tree*)
  - all leaf switches are connected to every spine switch
- Spine-leaf = folded three-stage Clos network
  - but minimal paths (shortcuts) do not exist in Clos



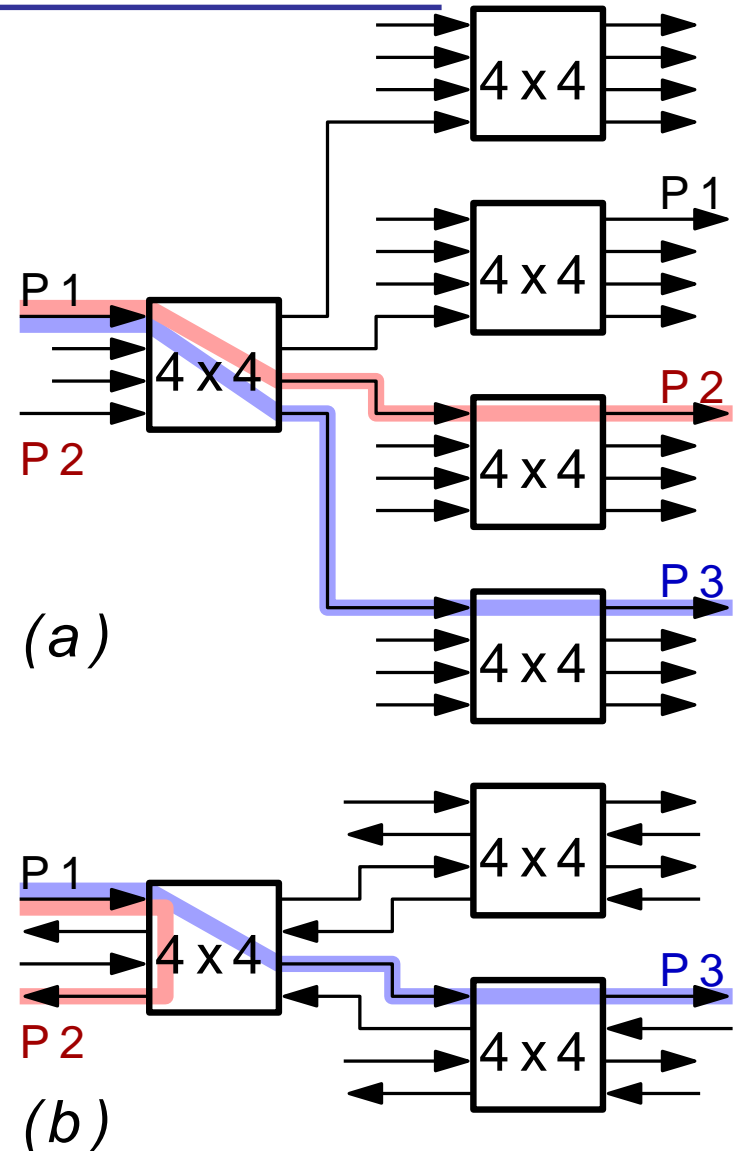
## 5.2.4 Fat Trees: equivalent networks



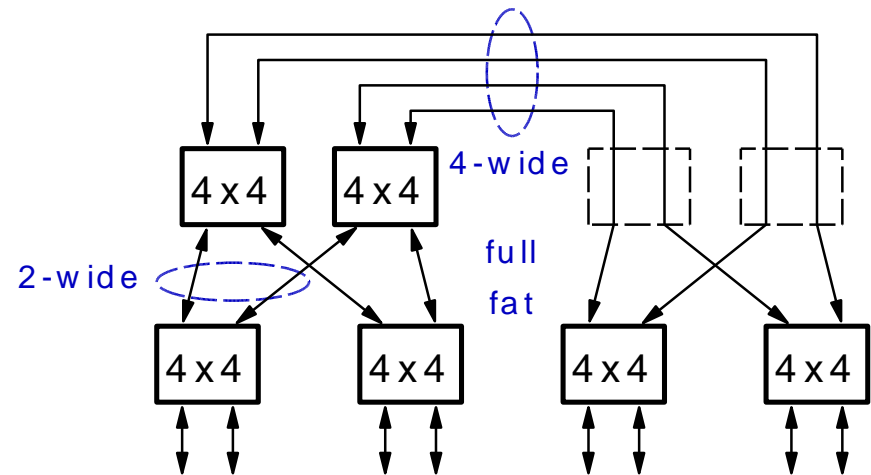
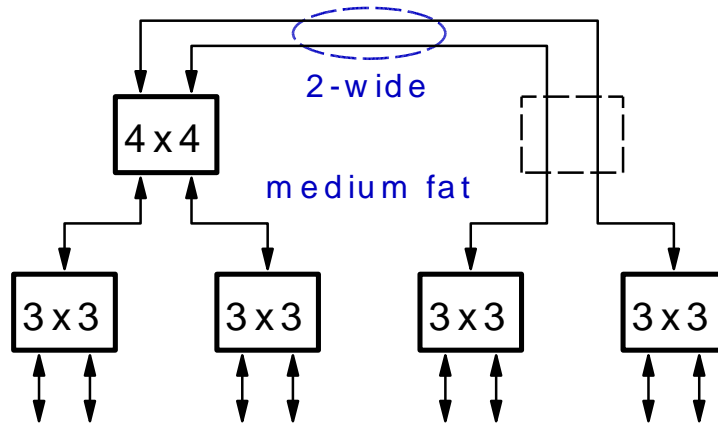
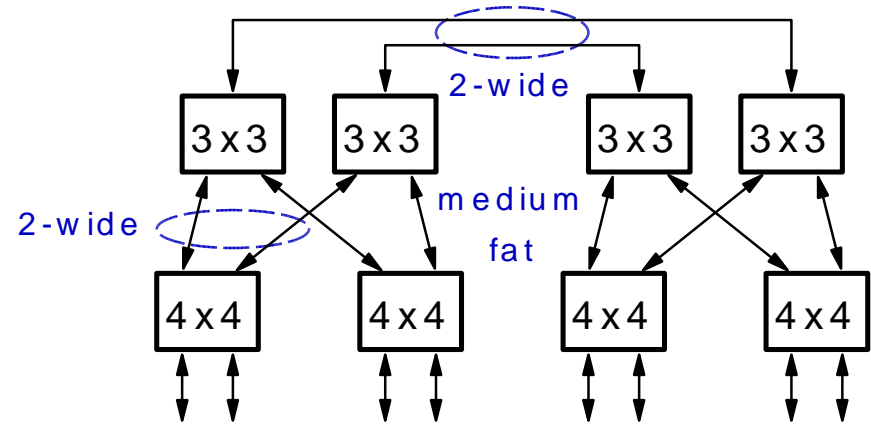
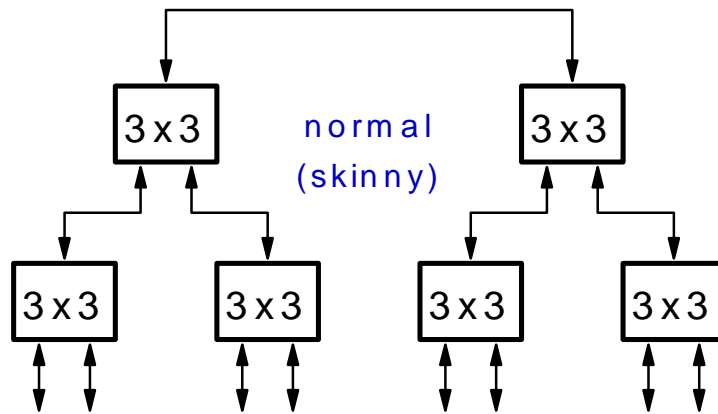
- All built using 4x4 switches
- All have same (full) bisection bandwidth
  - same number of wires in total
- All have same number of paths per port pair
- But different number of switches
  - savings on number of root switches

# Switch Radix, Hop Count, Network Diameter

- Most of our examples used unidirectional links – fig. (a)
  - “indirect” nets have ports at edges.
- Most practical interconnects use bidirectional links – fig. (b)
  - “direct” nets provide external ports on all switches.
- If some destinations are reachable at reduced hop count (P2 in (b)), that is at the expense of the total number of destinations reachable at a given hop count – or larger network diameter.
- Energy consumption to cross the net critically depends on the number of chip-to-chip hops, because chip power is dominated by I/O pin driver consum.

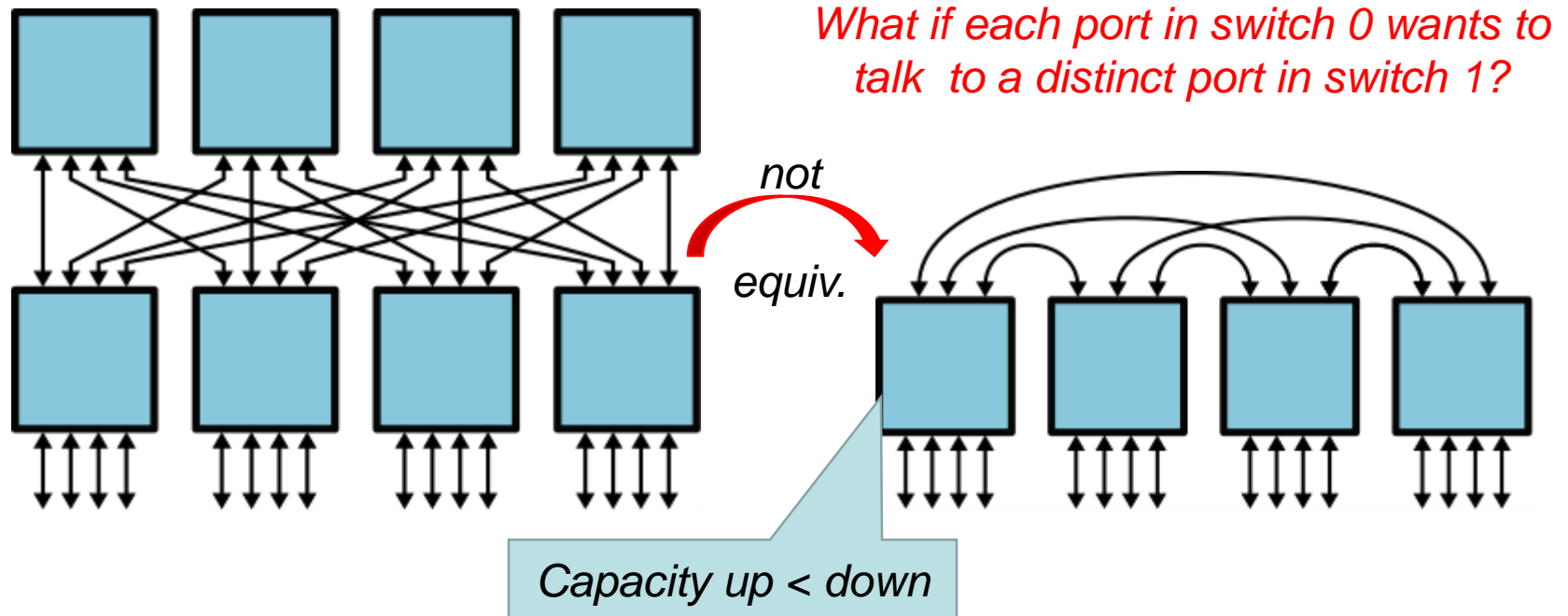


## 5.2.4 Fat Trees: customizable local versus global traffic



- Customizable percent fat – configurable amounts of internal blocking
- Bidirectional links, like most practical interconnects
- Skinny trees support local traffic – Full-fat tree is like folded Benes

## 5.2.4 Cannot eliminate the roots of a 2-level fat-tree

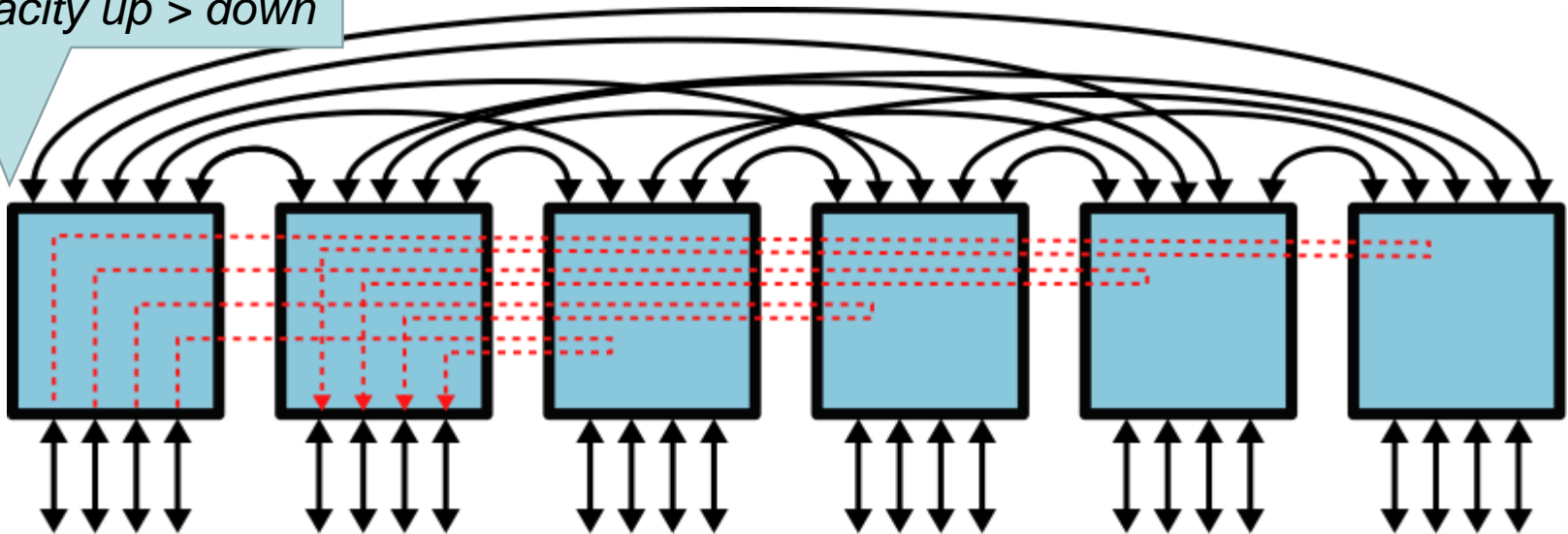


- 4-ary 2-tree
  - bisection bandwidth = 8 links
- Completely-connected netw. of switches
  - a bidi link between every switch pair (“4 per 2” links)
  - bisection b/w = 4 unidi. links for 8 src’ing ports...not enough



## 5.2.5 A fully-connected network

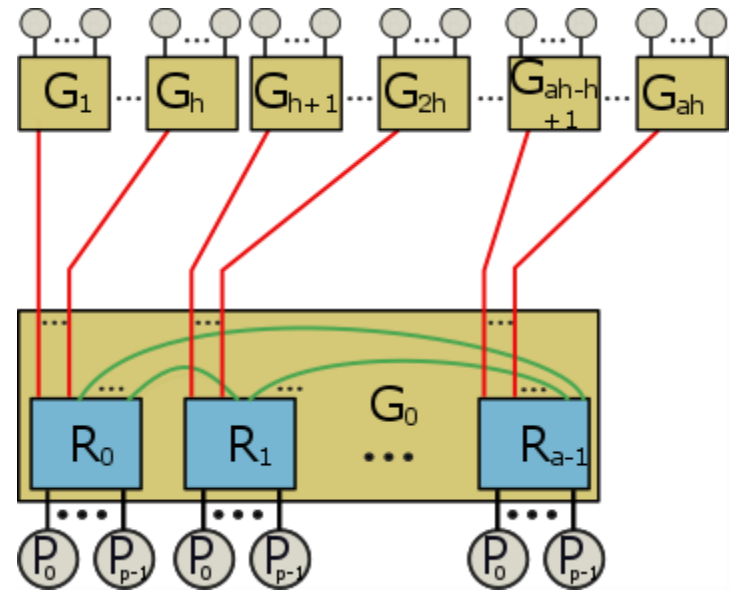
Capacity up > down



- Number of links = “6 per 2” =  $6! / ((6-2)! * 2!) = 15$  (bidi.) or 30 (unidi.)
- Bisection b/w = 9 unidi. links for 12 src'ing ports (better)
- Switch 0 can “talk” to 1, if we use one extra hop (**Valiant** routing)
  - but occupying 8 (almost 1/4 of the total) unidir. links in total
  - if all (6) switches do the same, they need  $6 \times 8 = 48$  unidi. links, and we have 30  $\rightarrow$  tput  $\sim 30 / 48$ , better than minimal routing (tput 1/4)
- However, for uniformly-destined (all-to-all) traffic, tput of Valiant routing  $\sim 30 / 48$  -- **worse** than minimal routing (tput 1)

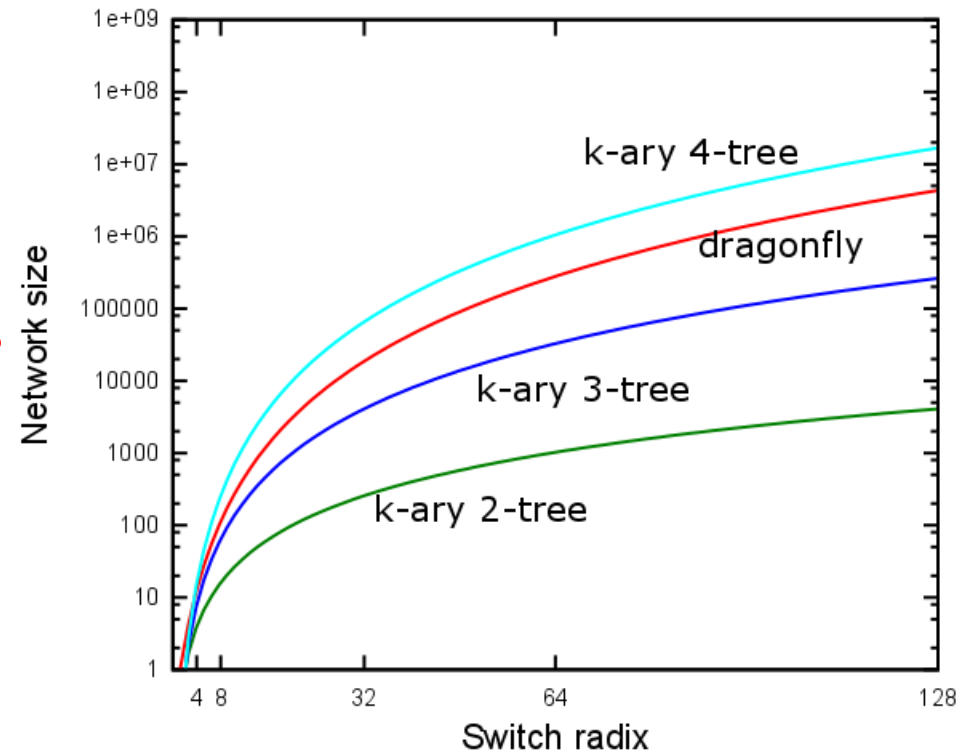
## 5.2.6 Dragonfly (bidirectional) networks

- Groups (supernodes) has **a** switches
  - Each switch (or node) **p** links to ports
  - Switches in same group full-connected (**+a-1 local links** / switch)
  - Each switch **+h** links to other groups
    - groups fully-connected **global links**
  - Switches have **k = p+h+a-1** ports
  - **N = ap(ah+1)** ports, **n = ah+1** groups
- 
- Dragonfly tries to minimize the number of expensive global links while maintaining a small diameter (critical for supercomputers)
  - Minimal routing: local + global + local
    - just one global link  $\rightarrow$  few (1 E/O + 1 O/E ) power-hungry signal conversions -- global links are long and thus optical (not electrical)
  - Demand on local links 2x than on global or port links in all-to-all traffic
    - selecting  $a \geq 2h$ ,  $a \geq 2p$  balances the load on all links under all-to-all traffic



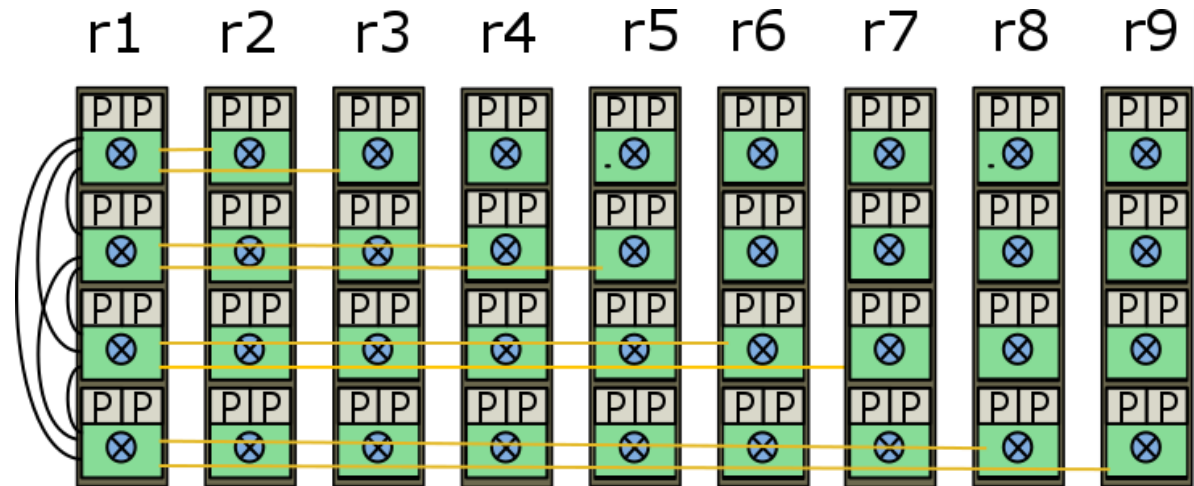
## 5.2.6 Dragonfly networks versus fat-tree

- Dragonfly (common config.)
  - $a = 2h = 2p$
  - $k = 4p - 1, n = 2p^2 + 1, N = 4p^4 + 2p^2$
  - size  $N = (k+1)^4 / 64 + (k+1)^2 / 8$
  - **bisection b/w  $\sim p^4$  links for  $2p^4$  ports**
- $k$ -ary  $n$ -tree
  - size  $N = (k/2)^n$
  - **$N/2$  links for  $N/2$  ports**
- Hop count comparison
  - **5 vs. 4 (2-tree) vs. 6 (3-tree) vs. 8 (4-tree)**
- Hop count (only global links)
  - **1 vs. 2 (2-tree) vs. 4 (3-tree) vs. 6 (4-tree)**
- Number of unidirectional global links (for network size  $N$ ) comparison:
  - **$N$  (dragonfly) vs.  $2N$  (2-tree) vs.  $4N$  (3-tree) vs.  $6N$  (4-tree)**

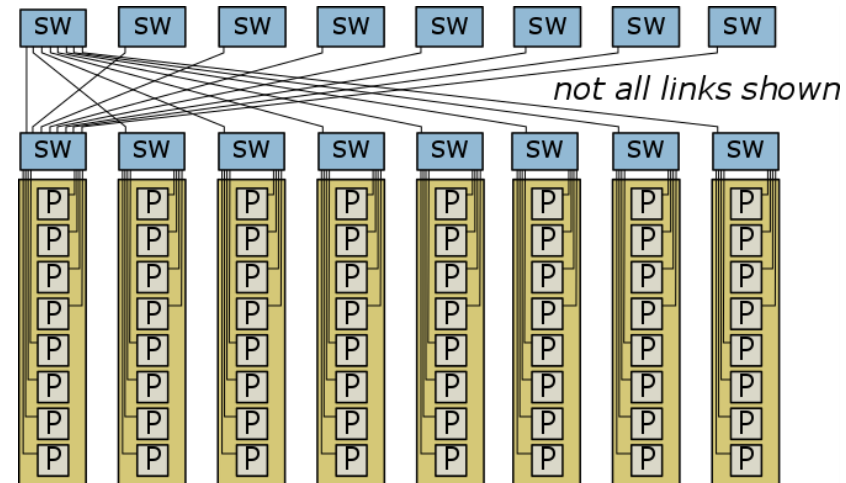


## 5.2.6 Server rack networks

- 72-port Dragonfly
  - $p=2, a=4, h=2$
  - 36, 7x7 switches
  - 9 x 8 servers
  - 72 global links



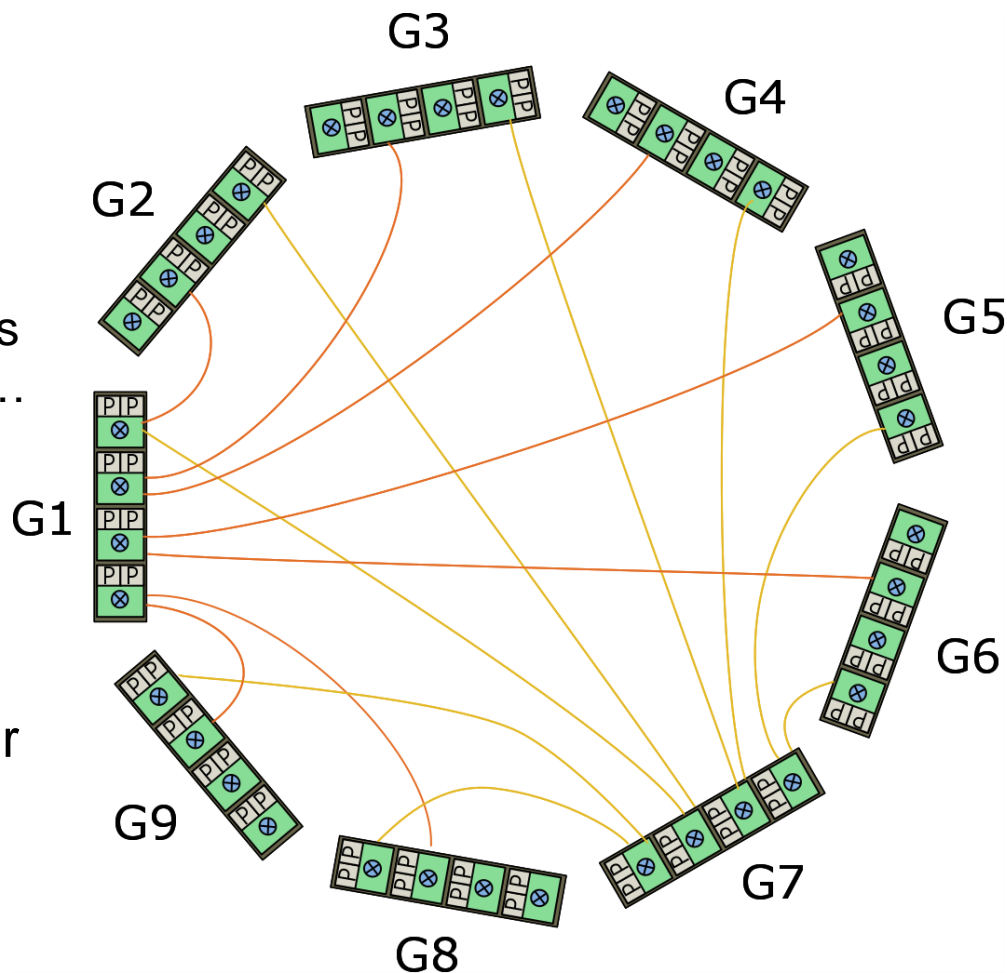
- 64-port 2-level fat-tree  
(8-ary 2-tree or spine-leaf)
  - 16, 8x8 switches
  - 8x8 servers
  - 128 global links



Global links implemented w. expensive optical links (electronic cables < 10 meter)

## 5.2.6 System-level Valiant routing in Dragonflies

- Minimal routing:
  - local + global + local
  - when G1 talks to G7
    - $2p^2$  ports clash on one link
  - G1 has  $2p^2$  global outgoing links (when  $p=h$ ) but minimal uses 1...
- Valiant routing w. 1 random intermediate group
  - local+global+local+global+local
  - full tput for G1 talking to G7
- But for uniform, minimal is better
  - tput 1 (if  $a/2 \geq p, h$  and  $h \geq p$ )
  - Valiant uses two global unidi. links / packet
    - $\rightarrow \sim \text{tput} = \frac{1}{2} h/p$
- How to adaptively select between the two?



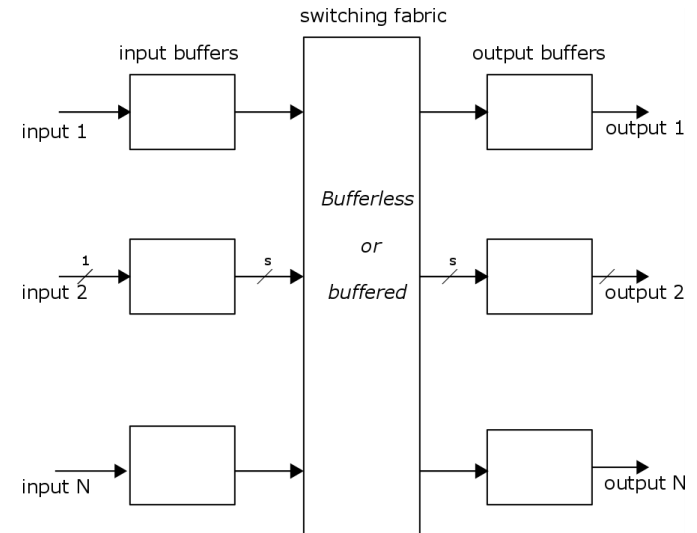
72-port Dragonfly:  $p=2, a=4, h=2$   
 (global links of G1 & G7 shown)

## 5.3 Towards Scalable Switches

- Buffer throughput limitation  $\Rightarrow$  use input queueing or CIOQ
- Input queued crossbar scalability limited primarily by:
  - quadratic cost growth rate,  $O(N^2)$ , of crossbar
  - scheduler complexity & efficiency, i.e. solving the output contention (congestion management) problem
- To solve the crossbar cost  $\Rightarrow$  use switching fabrics
- To solve the scheduler / contention / congestion problem:
  - (sorting / self-routing networks – bad solution)
  - Switching Fabrics with Small Internal Buffers, large input VOQ's, and Internal Backpressure (Flow Control)

## 5.3.1 Buffer Organization in Switching Fabrics

- Packet switched networks & fabrics
  - buffers to resolve contention



- Bufferless fabrics : buffers at ports but not internally
  - OQ: buffers only at outputs (expensive)
  - IQ : buffers only at inputs (not scalable scheduling, poor performance)
  - CIOQ: buffers at both inputs & outputs
- Buffered fabrics : internal buffers (in addition to port buffers)
  - gradual contention resolution + better performance
  - preferred nowadays : cables dictate cost, on-chip buffers are cheap

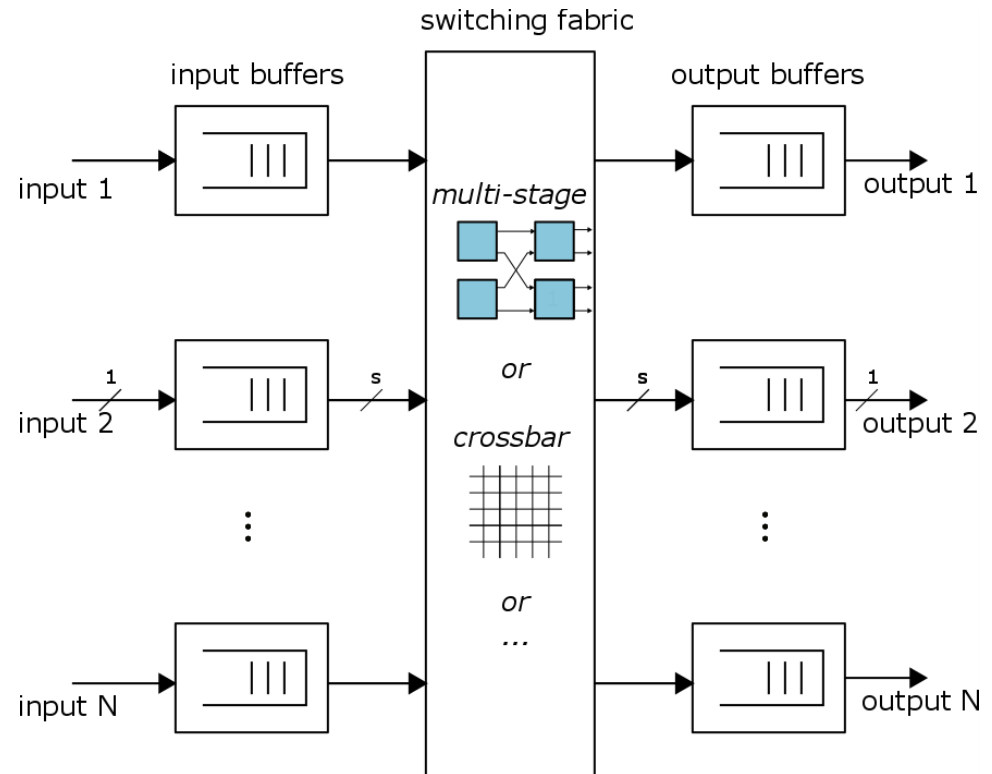
## 5.3.1 Speedup In Switching Fabrics

- Internal speedup often used to improve the performance of CIOQ
  - expensive for off-chip switching fabrics
    - (fabric-internal **off-chip links run faster** than ports)*
      - difficult to increase chip I/O bandwidth
      - power consumption dictated by chip I/O bandwidth
    - less expensive for on-chip switches and networks
      - (make wider datapath)*
        - e.g. inside a single-chip crossbar or Clos
- Input speedup
  - input buffer read tput / input buffer write tput (= line tput)
- Output speedup
  - output buffer write tput / output buffer read tput (= line tput)



## 5.3.1 Modern CIOQ Switching Fabrics

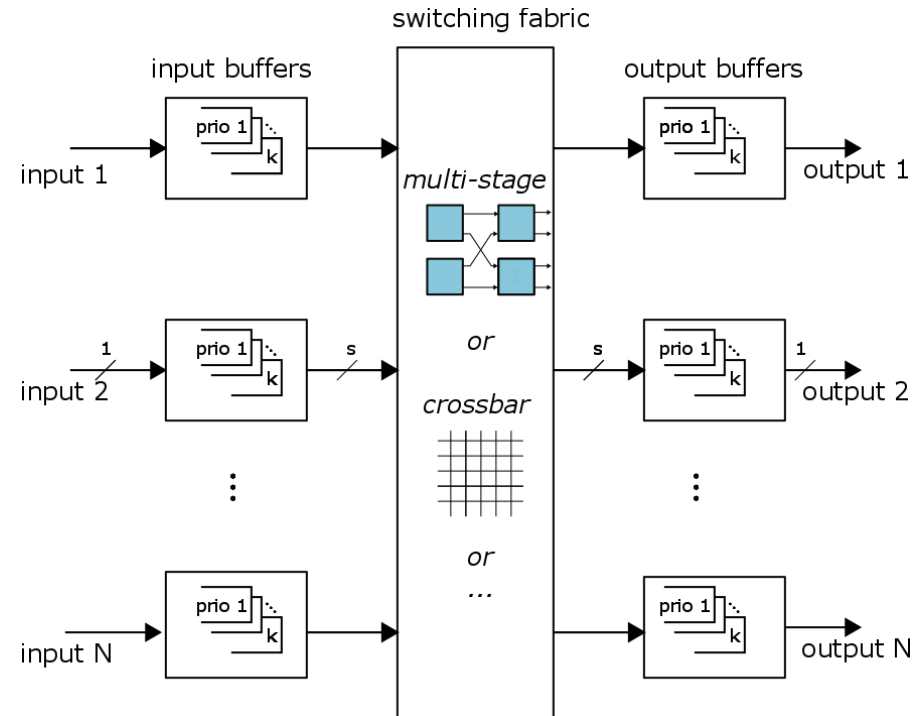
- Buffers at inputs & outputs
- Internal speedup
  - internal links & switches run  $s$  times faster than ports



- **Single FIFO queue per input / output**
  - simple scheduling (one candidate per input)
  - but first-in-first-out service and HOL block, simple

## 5.3.1 CIOQ Switching Fabrics + Priorities

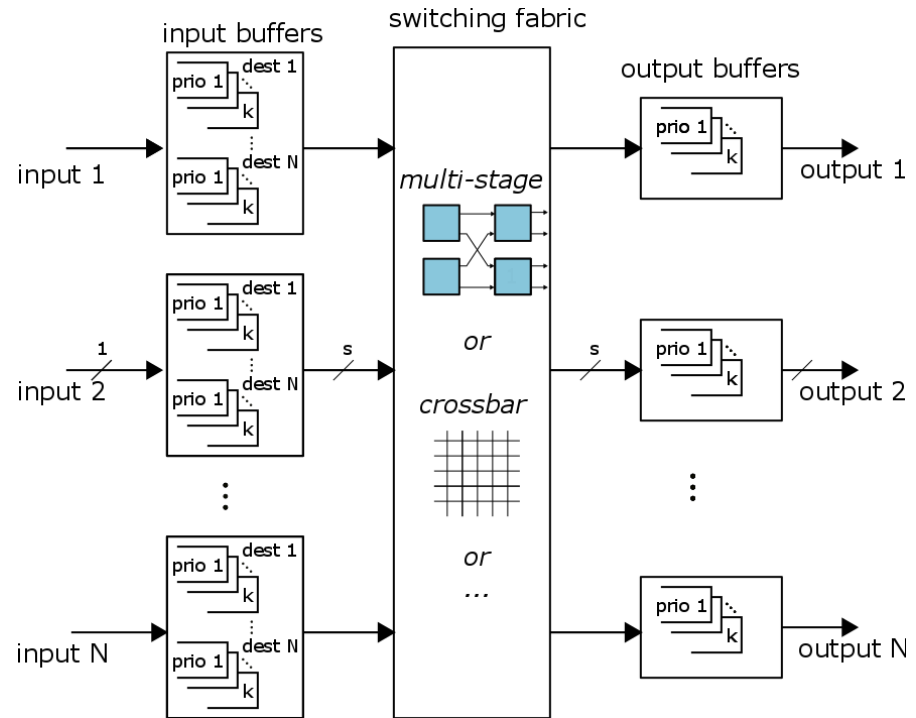
- Buffers at inputs & outputs
- Internal speedup



- **Private queues per priority-level (service class)**
  - 2-16 priority levels
  - typically separate buffers per priority-level
  - only recently implemented in Ethernet

# 5.3.1 CIOQ Switching Fabrics + Input VOQs

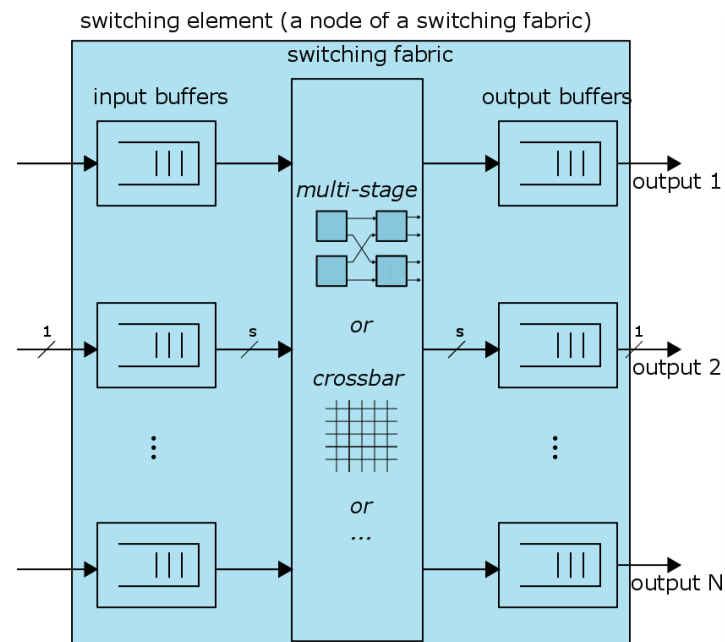
- Buffers at inputs & outputs



- Private input queues per output (VOQs)
  - tens to thousands of destinations
  - separate input buffers per VOQ?
  - only inside router boxes → neither Ethernet nor Infiniband

## 5.3.1 How Do Switching Nodes Look Like?

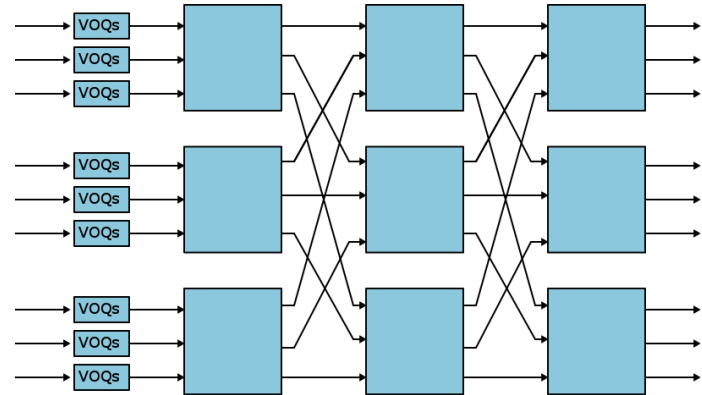
- Switching elements (or nodes)  
*(building blocks of multi-stage switching fabrics)*
  - single chip switch (in a board)
  - switch/route box in a data center or supercomputer



- Node  $\leftrightarrow$  network
  - recursive definition of networks
- Modern switching nodes are CIOQ switch chips
  - priority levels + local VOQs

## 5.3 Scheduling in Bufferless Clos Networks

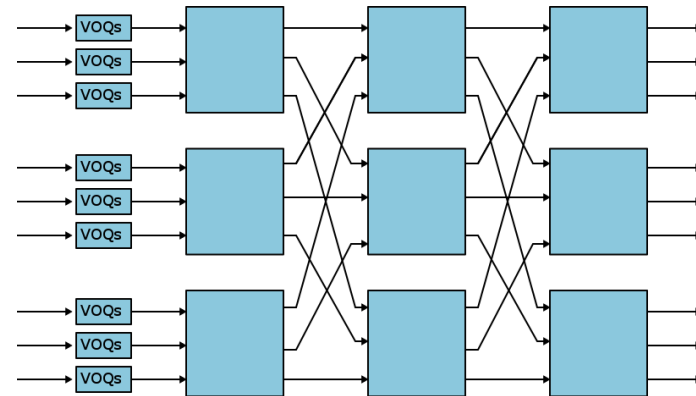
- VOQs at inputs
  - separate queue at each input for each fabric output
- No speedup



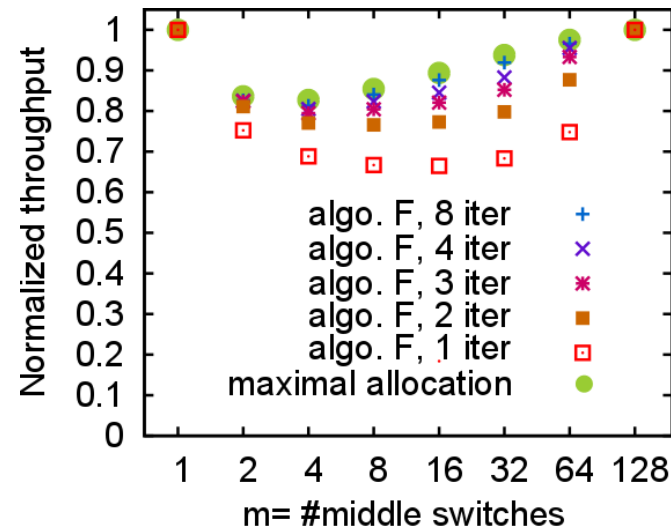
- At each time-slot (packet time)
  - 1) packet scheduling among non-empty VOQs
    - bipartite graph matching (inputs/outputs)
      - each input with one at most output
      - each output with one at most input
  - 2) route assignments for selected packets
    - no two packets from same 1<sup>st</sup> or 3<sup>rd</sup> stage module use same route (color)

## 5.3 Iterative route assignments: non-backtracking algo

- Input:  $N$  packets (1 per port)
- Output: non-conflicting routes for a subset of packet
- For  $i$  in 1 to num\_iterations
  - for each packet
    - output module selects a random locally available route
    - if route also available at input module  $\rightarrow$  reserve route
- Converges to maximal route assignment
  - new edges can be added only if we rearrange existing ones



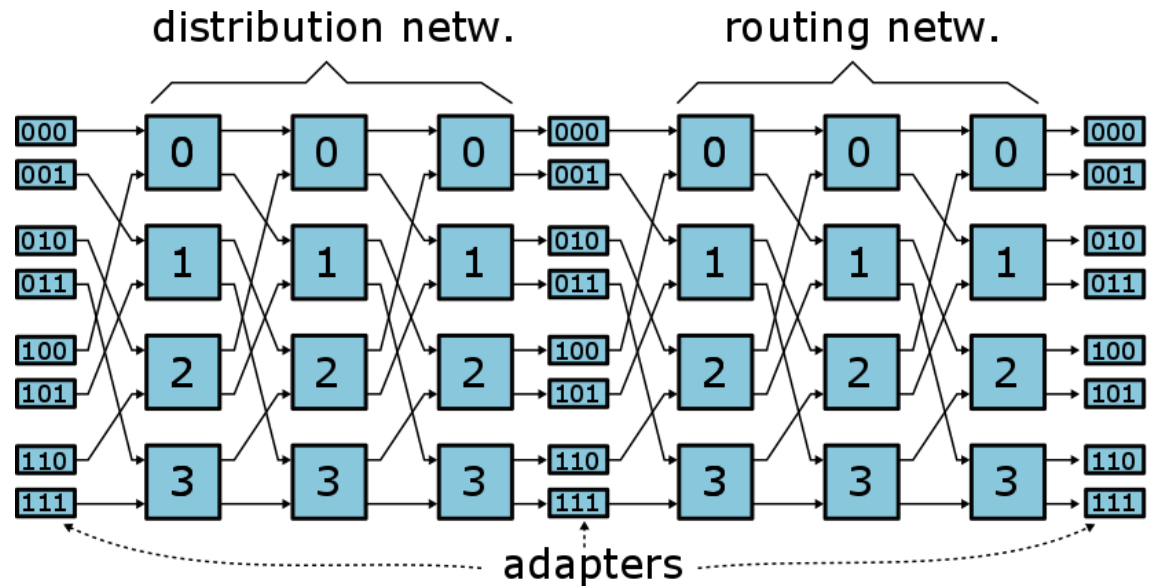
$N=128$ , num of 1/3-stage modules =  $N / m$



## 5.3 Load-Balanced (Birkhoff-von Neumann) switches

### At time-slot $t$ :

- input adapter  $i$  connected to intermediate  $(i+t) \bmod N$
- intermediate adapter  $j$  connected to dest  $(j+t) \bmod N$



- Valiant routing
- Permutations chosen so that no conflicts in distribution/routing networks
- Buffers/queues only at intermediate adapters (VOQs)
  - ~ shared-memory switch
- Simple distributed control/scheduling -  $O(1)$  complexity
- But out-of-order (OOO) delivery at dests ... AND  $O(N)$  packet latency even at low loads

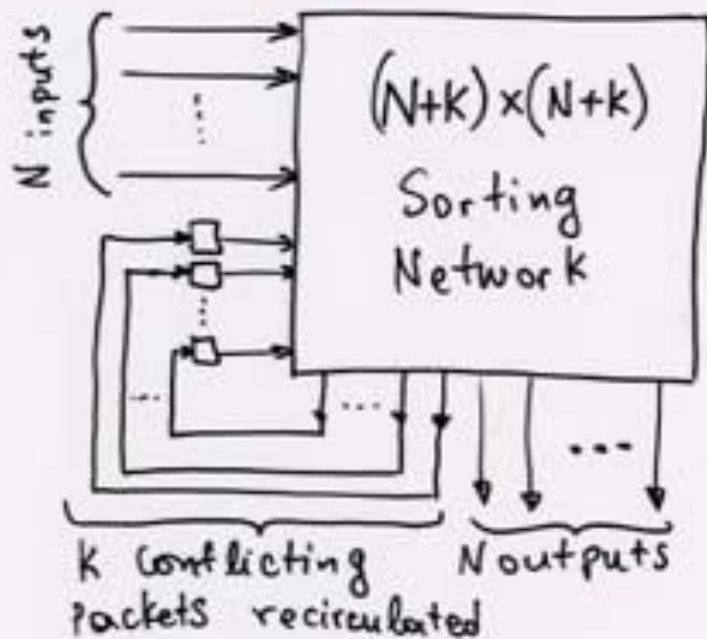
[intentionally left blank]



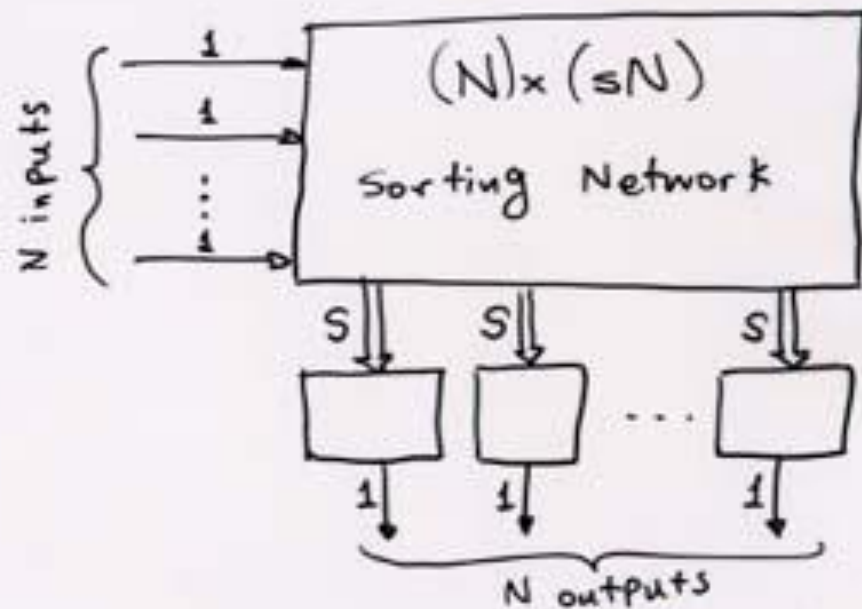
## Central Scheduler is Impractical for large N

### Solution 1: Sorting Networks w. Distributed Control (see ch.5 for details)

- all incoming packets allowed in – no central scheduling
- conflicting packets appropriately steered – distributed control



- uses communication paths as buffer memory  
... too expensive

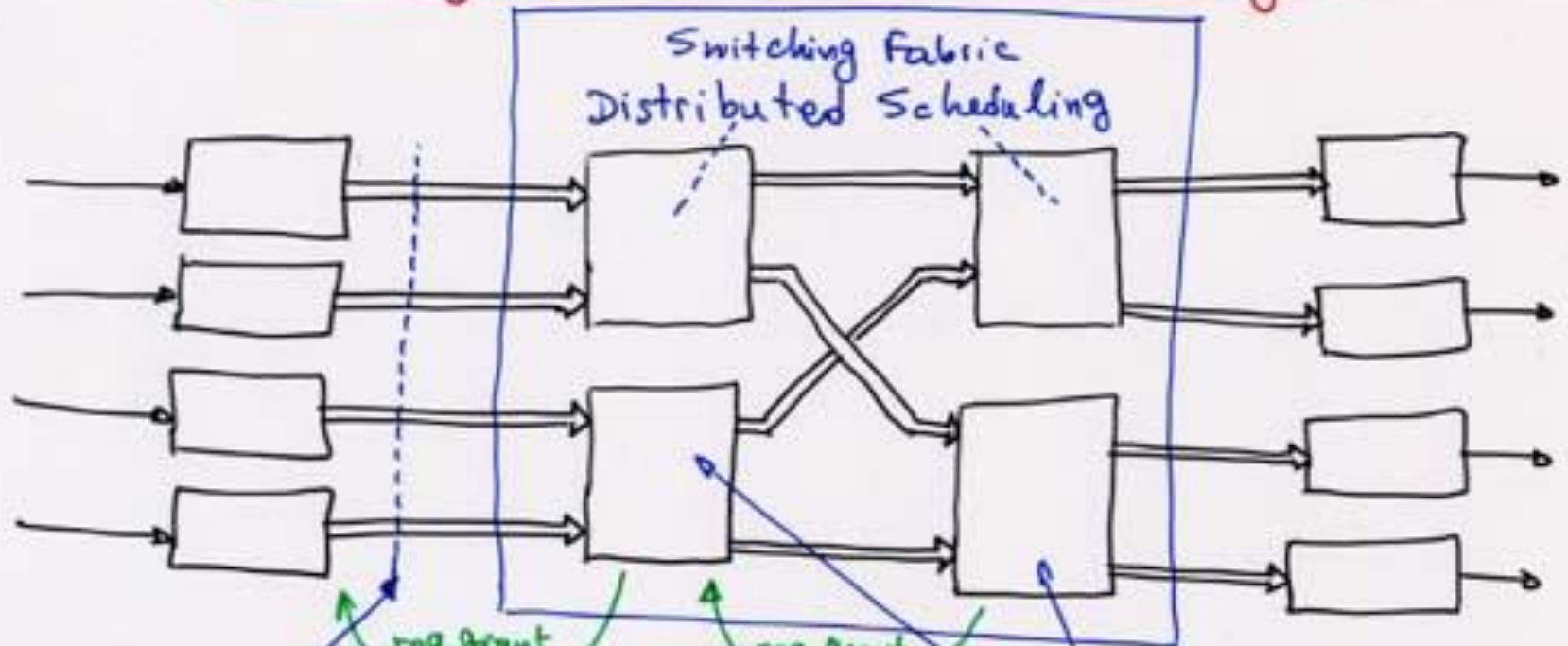


- Knock-Out Style  
but different Sw. fabric

- Sorting Networks are quite large... not too practical

# Central Scheduler is Impractical for large N

## Solution 2: Switching Fabrics with Internal Buffering & Backpressure



the traffic here may have packets that are short-term-conflicting in the switching fabric, but are long-term-non-conflicting in the fabric

small internal buffers  
handled by these  
owing to backpressure and distributed scheduling