Operating Systems
Multi Computer Systems

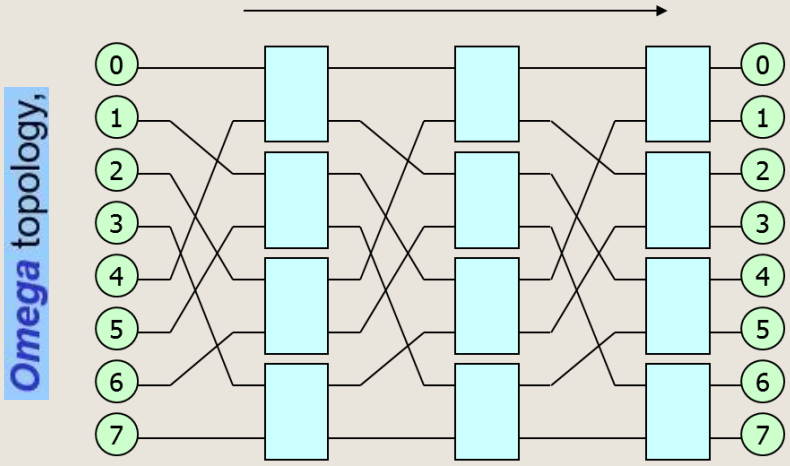
Shyan-Ming Yuan
CS Department, NCTU
smyuan@gmail.com
Office hours: Wed. 8~10 (EC442)

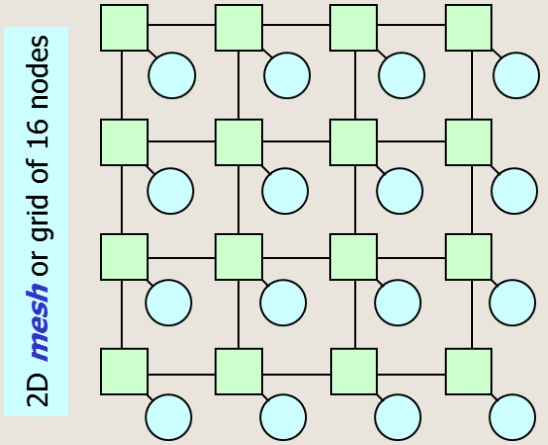
Multi Computers (MCs)

- Multiprocessors share a common memory
  - It is very expensive to build a MP with huge size.
- Multi computers are closely coupled computers that do not physically share any memory.
  - MCs are cluster of computers and networks or clusters of workstations (NOWs or COWs)
  - MCs can grow to a very large number.
- A MC consists of
  - Processing nodes contains CPU, memory and network interface
  - I/O nodes contains device controller and network interface
  - High speed interconnection network for message passing
    - There are many topologies – e.g. grid, hypercube, torus, …
    - It can be packet switched or circuit switched.

Interconnection Networks (INs)

- Direct or Indirect Networks
  - Nodes sit "inside" (direct) or "outside" (indirect) the network.
  - Mesh is a direct network.
    - Every node is both a processing node and a network switch.
  - Omega switch is an indirect network.
    - Processing nodes are connected by dedicated switching nodes.
    - All messages between any pair of processing nodes must be routed through switching nodes.



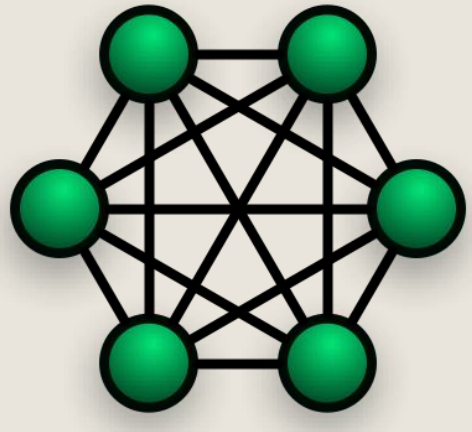


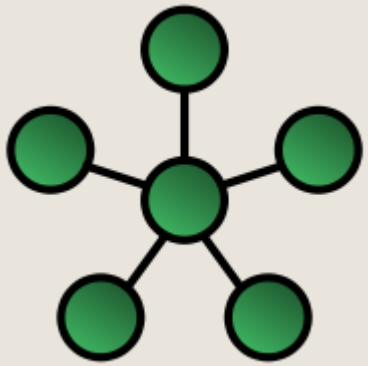
Properties of INs

- Degree (a): the number of incident nodes of node a
- Distance (a, b): shortest path between nodes a and b
- Diameter: maximal distance between all pairs of nodes
- Bisection width: the smallest number of links can disconnect the network into two equal halves (± 1).
- Average distance: average distance over all pairs of nodes
- Cost: resources needed to construct the topology
  - Number of links and switches and so on
- Symmetricity: uniform vs hot-spot traffic pattern
- Fault tolerance: how many faults that a topology can bear to continue providing message service

Direct INs

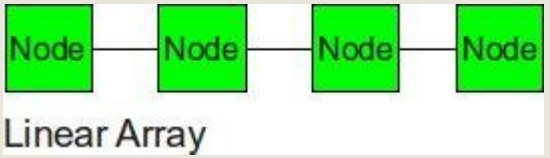
- Completely connected networks
  - Diameter (1),
  - Average distance (1),
  - Bisection width and Cost are $O(N^2)$
- Star-connected networks
  - Diameter (2),
  - Average distance (2),
  - Bisection width and Cost are O(N)
- Linear array and Ring
- Mesh and Torus
- Hypercube and more



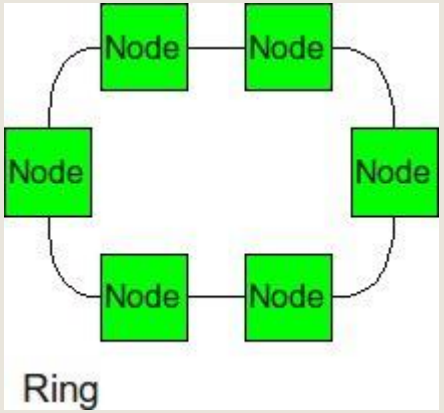


Linear Array and Ring

- Linear Array:
  - Diameter: N-1 (an array of N nodes)
  - Average Distance: (N+1)/3
  - Bisection Width: 1
  - Cost: O(N)

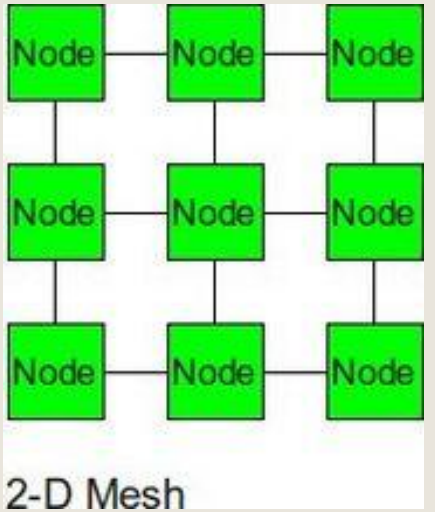


- Ring (1D Torus):
  - Diameter: N/2 (N is even), (N-1)/2 (N is odd)
  - Average Distance: ~ N/4
  - Bisection Width: 2
  - Cost: O(N)

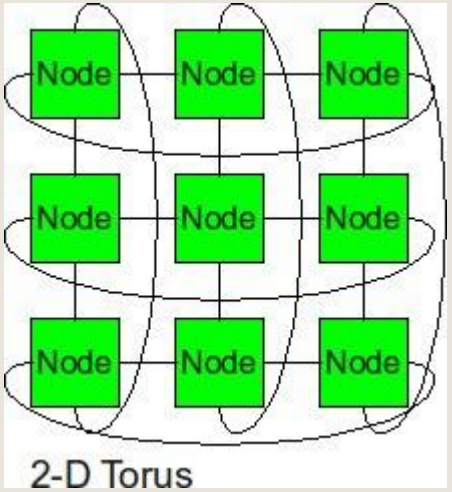


2D Mesh and 2D Torus

- Mesh (2D array):
  - Diameter: $2(\sqrt[2]{N}-1)$ for a mesh of n nodes
  - Average Distance: $O(\sqrt{N})$, $(2x2\sqrt{N})/3$
  - Bisection Width: $O(\sqrt{N})$, $\sqrt{N}+1$
  - Cost: O(N)



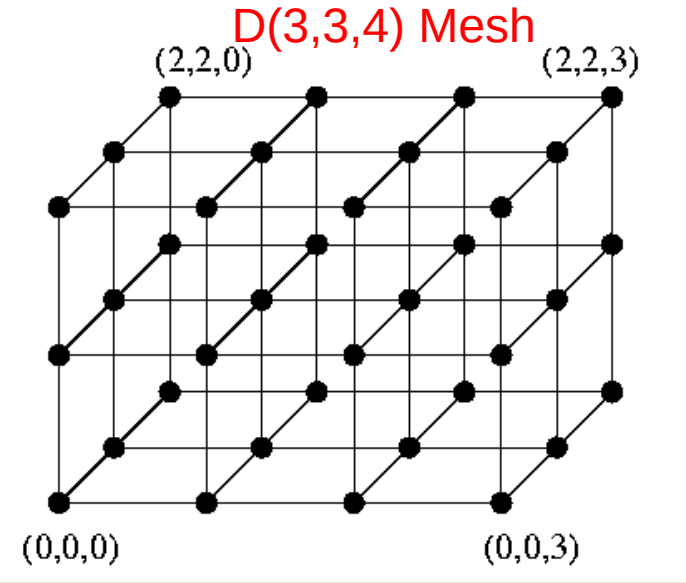
- 2D Torus:
  - Performance of mesh is very sensitive to placement of nodes.
  - Torus can avoid this problem
  - Torus has smaller diameter ($\sqrt{N}$) and average distance ($(1/2)\sqrt{N}$) than mesh.
  - The Cost and Bisection width are higher than mesh.



Multidimensional Mesh

- D-dimensional mesh:
  - N = k_{d-1} x k_{d-2} x … x k_0 nodes
    - Each node is described by

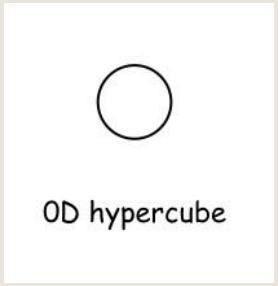
    a d-vector of coordinates (x_{d-1}, …, x_0)

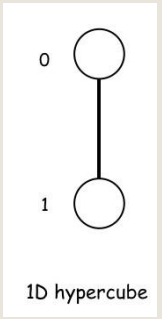


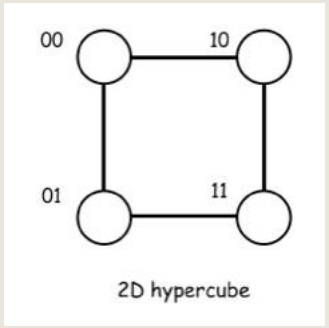
- D-dimensional k-ary mesh: $N=k^d$ and $k=\sqrt[d]{N}$
  - Each node is described by a d-vector of radix k coordinate.
  - Diameter: $d(\sqrt[d]{N}-1)$
  - Average distance: $d \times 2(\sqrt[d]{N})/3$
  - Bisection width: $O(\sqrt[d]{N^{(d-1)}})$

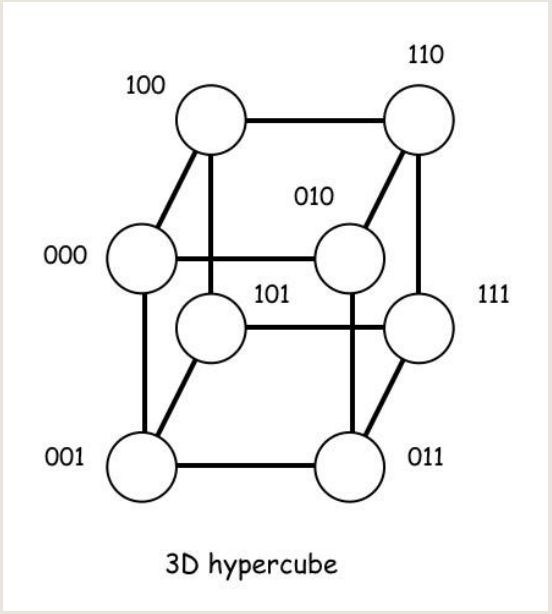
Hypercubes

- An N-node hypercube is a $\log_2 N$ dimensional 2-array.
  - It consists of $N=2^n$ nodes, where $n=\log_2 N$.
    - Each node is described by n-vector of binary coordinate.
    - Each node is connected to all other nodes whose numbers differ from it in only one bit position.
    - If n=3, a node (1,0,0) is connected to nodes (0,0,0), (1,1,0), and (1,0,1)
  - Diameter: $n=\log_2 N$
  - Average Distance: $O(\log_2 N)$, $(1/2)\log_2 N$
  - Bisection Width: O(N), (1/2)N



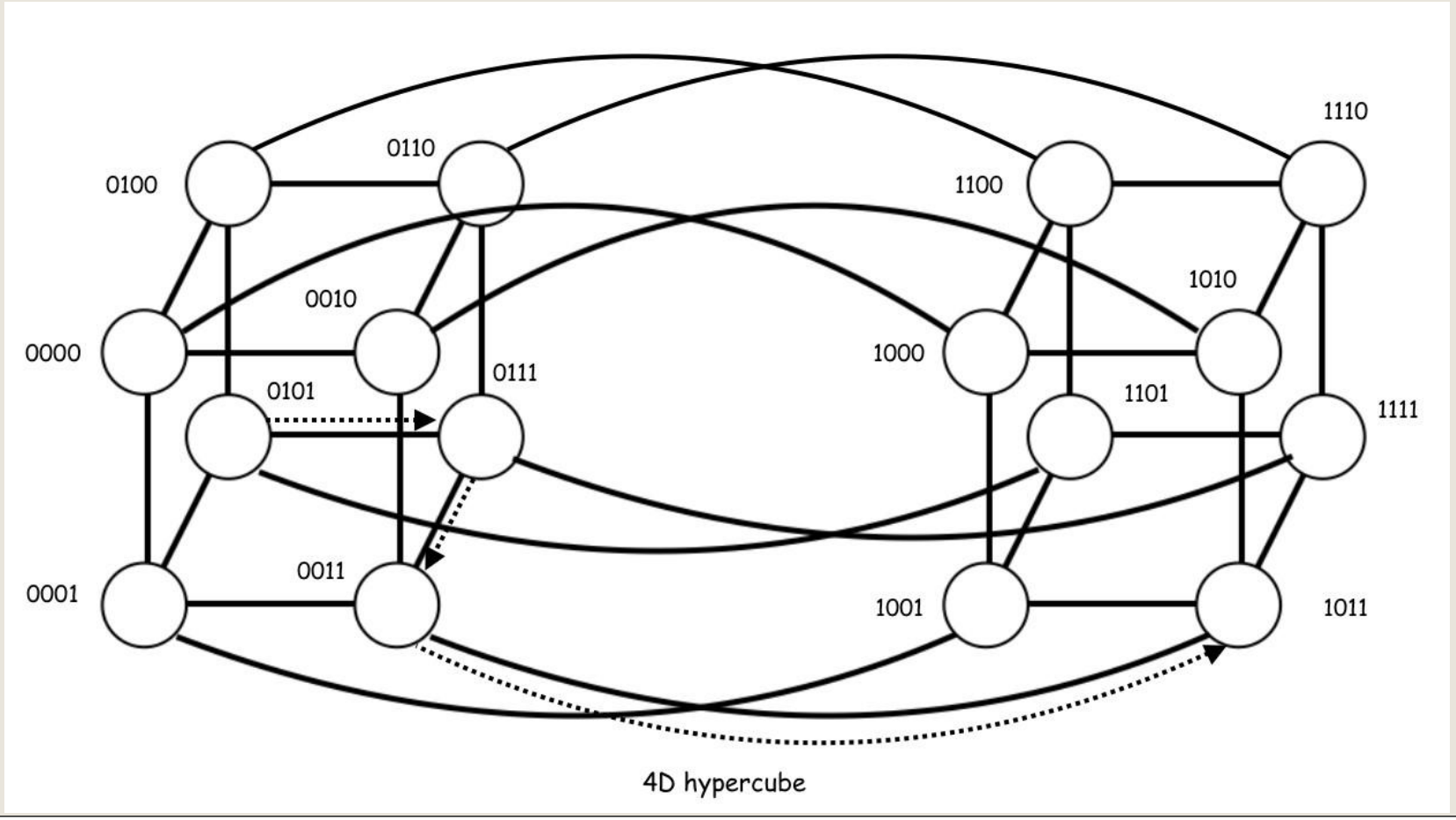






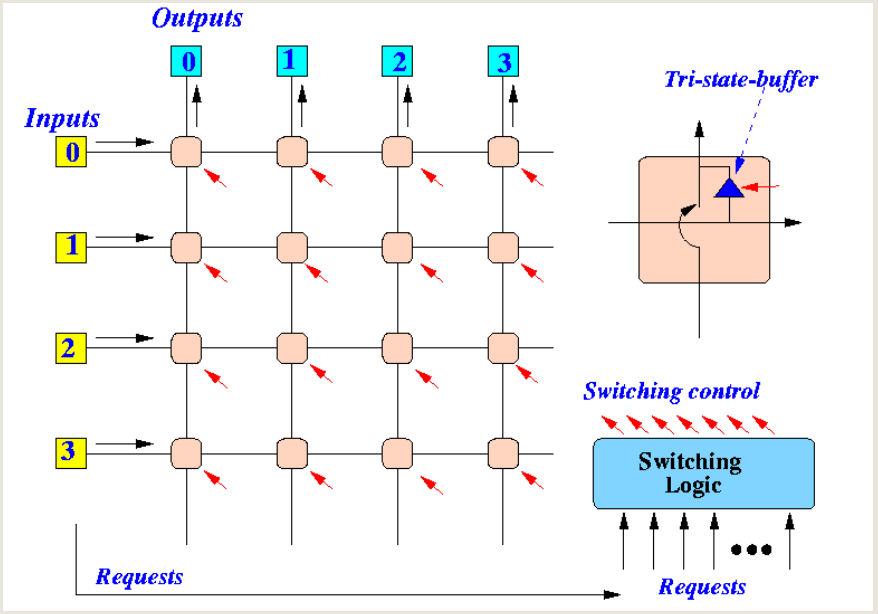
The 4D Hypercube

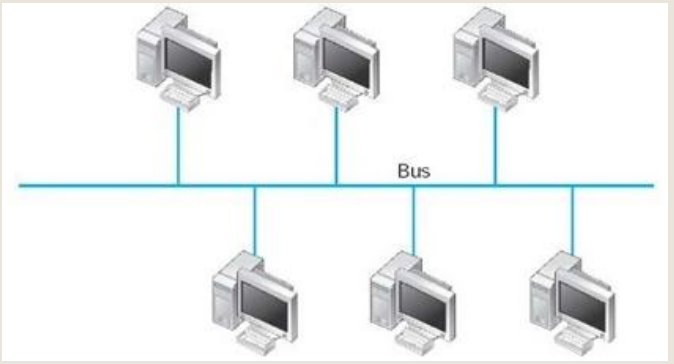
- A 4D hypercube can be constructed by connecting corresponding nodes of two 3D hypercubes.

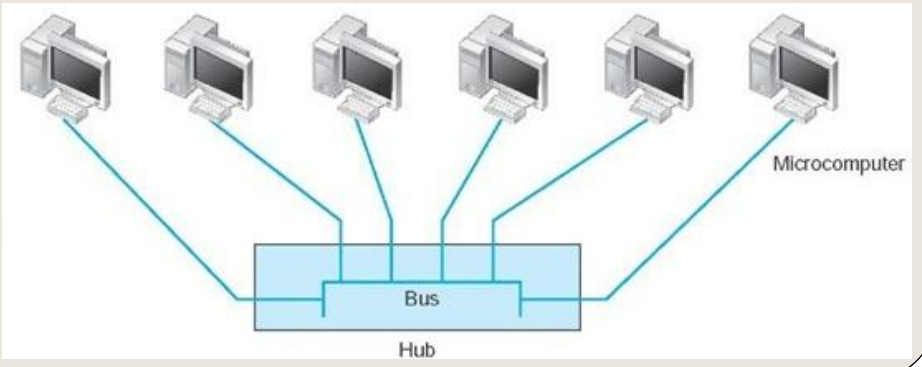


Indirect INs

- Bus based networks.
  - Ethernet Bus, Ethernet Hub, and Ethernet Switch
  - Diameter (2), Average distance (2), Cost (constant)
- Crossbar switch.
  - Cost: $O(N^2)$

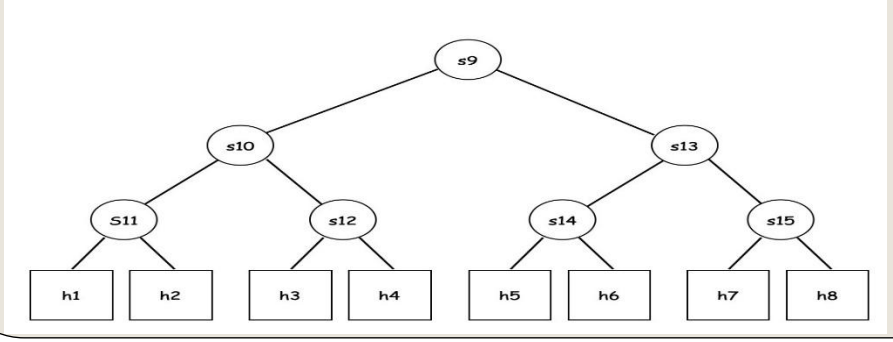


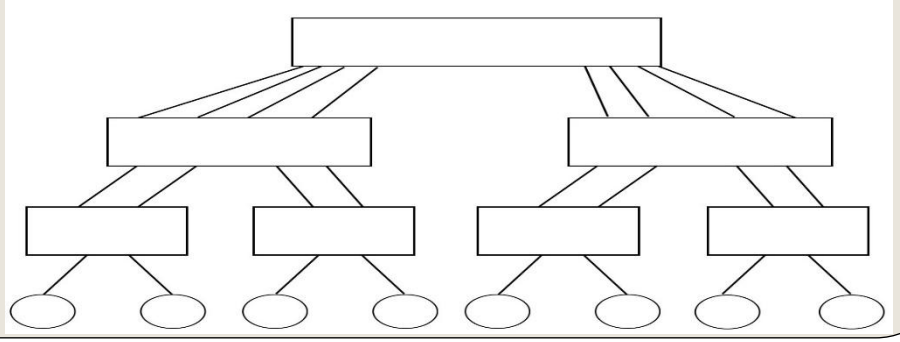




Tree Topologies

- Tree is a planar and hierarchical topology.
  - Processing Nodes are placed as the leaves and all other intermediate nodes are switches.
  - Diameter: $2\log_2 N$ for binary tree, N is the number of processing nodes
  - Average Distance: $O(\log_2 N)$ for binary tree
  - Bisection Width: 1 for binary tree
  - The upper layer switching node may become bottle necks.
- Fat trees can avoid the bottleneck problem.
  - It doubles the bandwidth of upper layer switches.



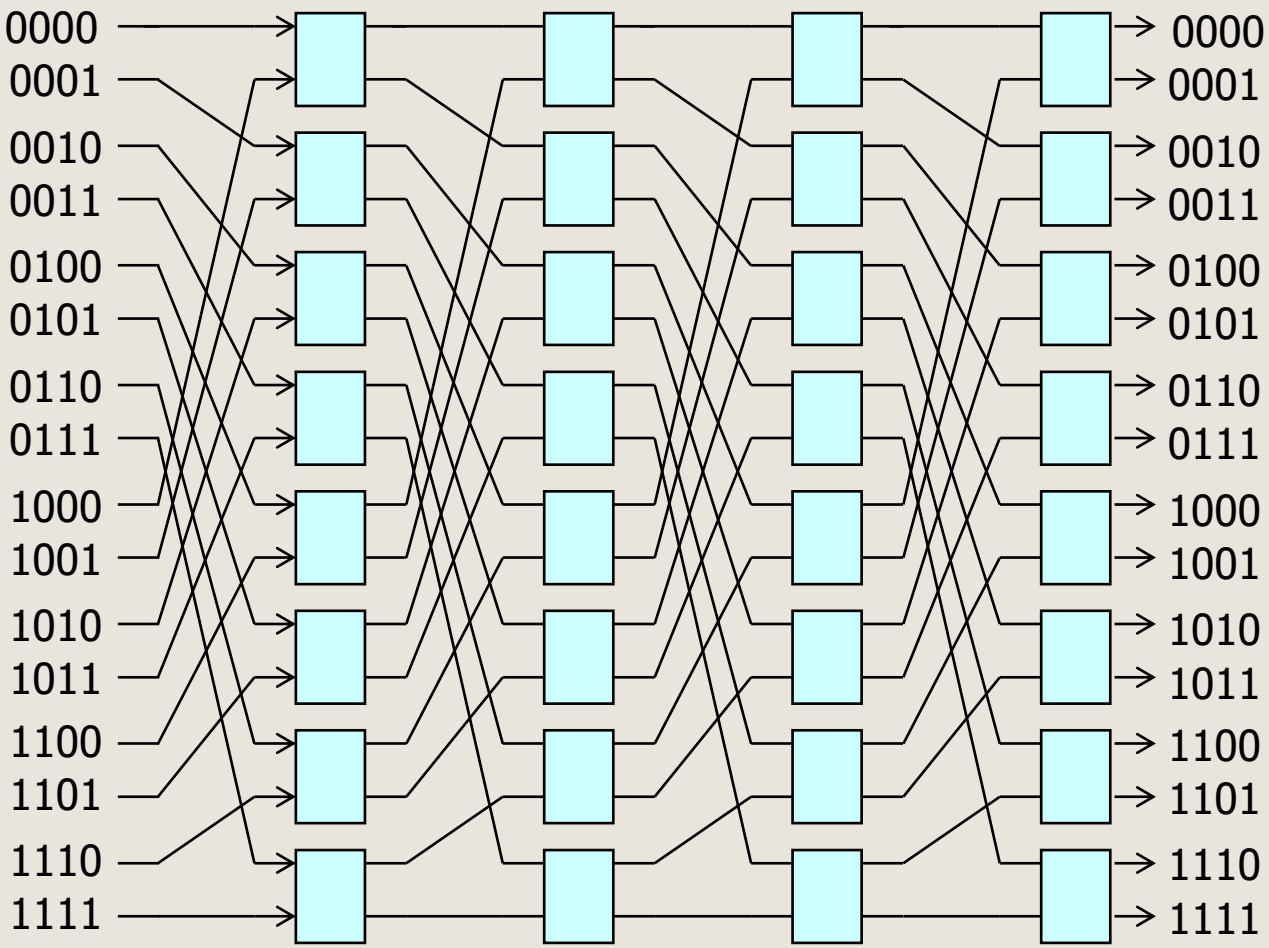


Multistage INs (MINs)

- Split a big crossbar into several stages of smaller crossbars.
- Inter-stage crossbars are connected by a particular pattern.
  - The pattern may be represented by a permutation function.
- In general, a MIN can be constructed by interconnecting N inputs to N outputs with k x k crossbars (kkCBs).
  - There are $\log_k N$ stages and each stage has N/k kkCBs.
  - The total number of kkCBs are $N/k(\log_k N)$.
  - The total number of links are $N(\log_k N+1)$.

The Omega MIN

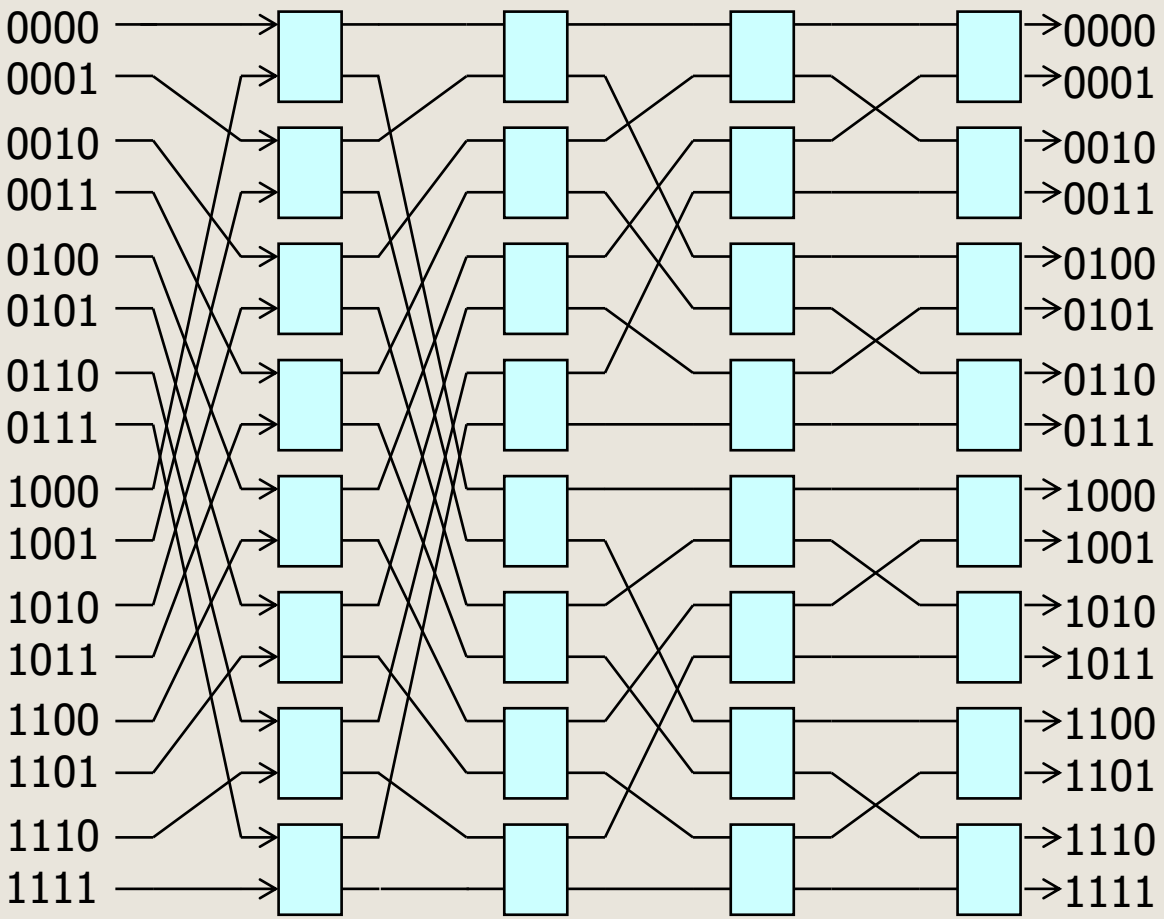
- Perfect Shuffle between stages



16 port, *4 stage **Omega** network*

The Baseline MIN

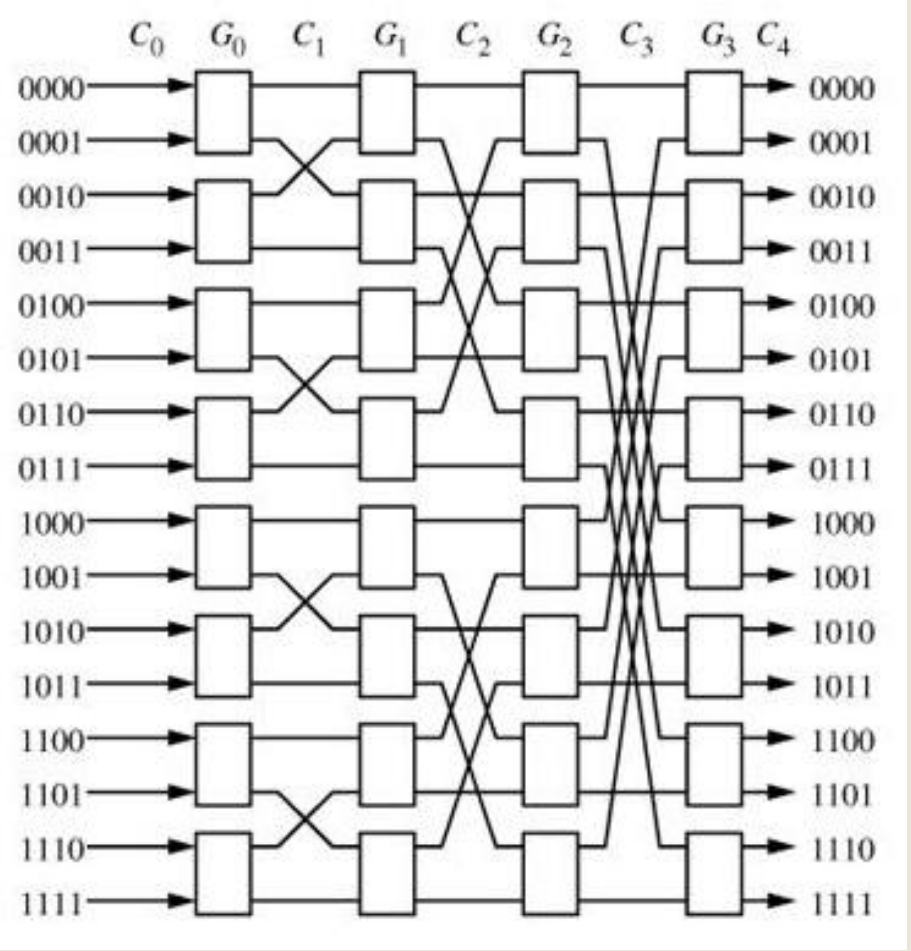
- 1st stage N x N
  Perfect shuffle
- 2nd stage N x N
  Inverse shuffle
- 3rd stage N/2 x N/2
  Inverse shuffle
- ….



16 port, *4 stage* ***Baseline*** *network*

The Butterfly MIN

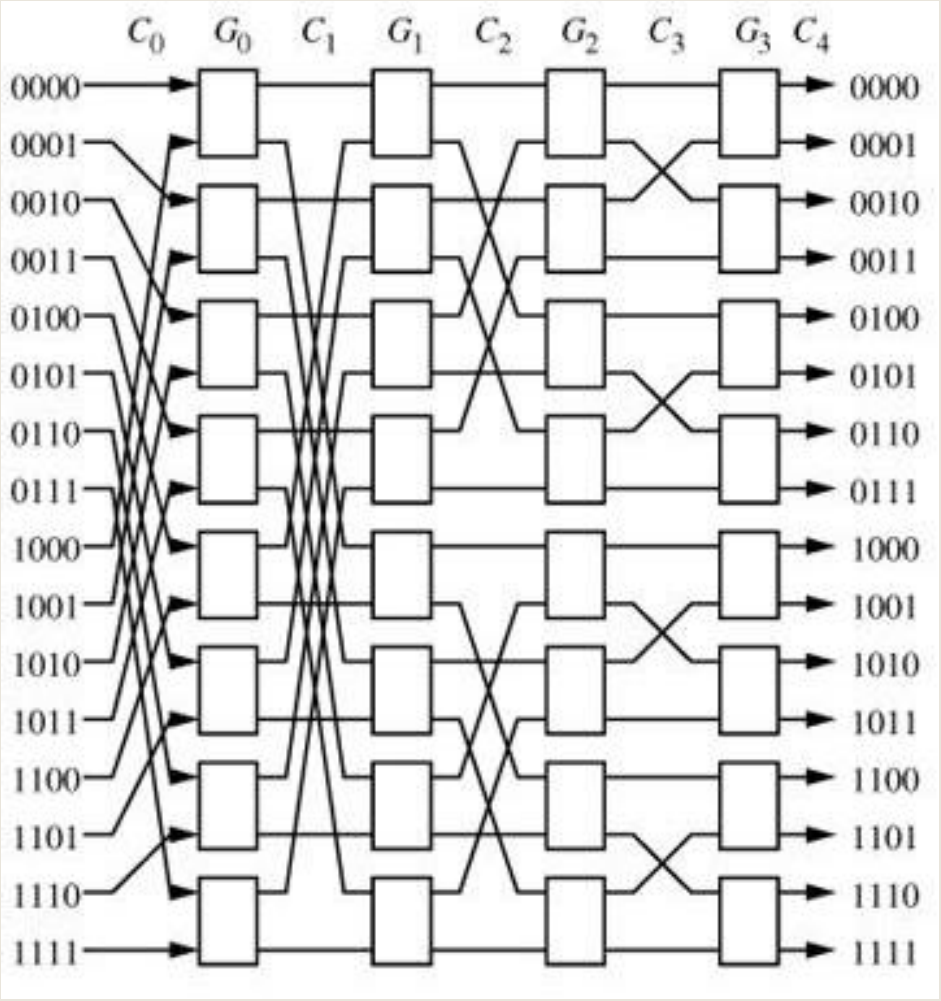
- 2nd stage 2 x 2 cross over
- 3rd stage 4 x 4 cross over
- ….



16 port, *4 stage* ***Butterfly*** *network*

The Cube MIN

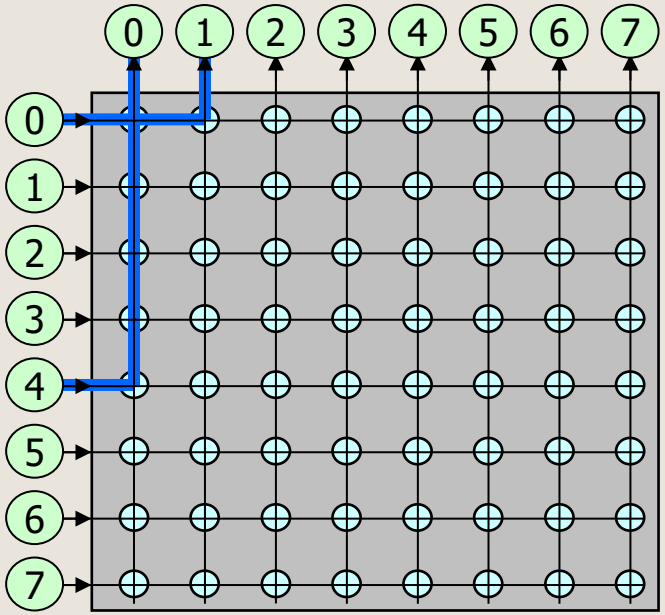
- 1st stage perfect shuffle
- 2nd stage N x N cross over
- 3rd stage N/2 x N/2 cross over
- …



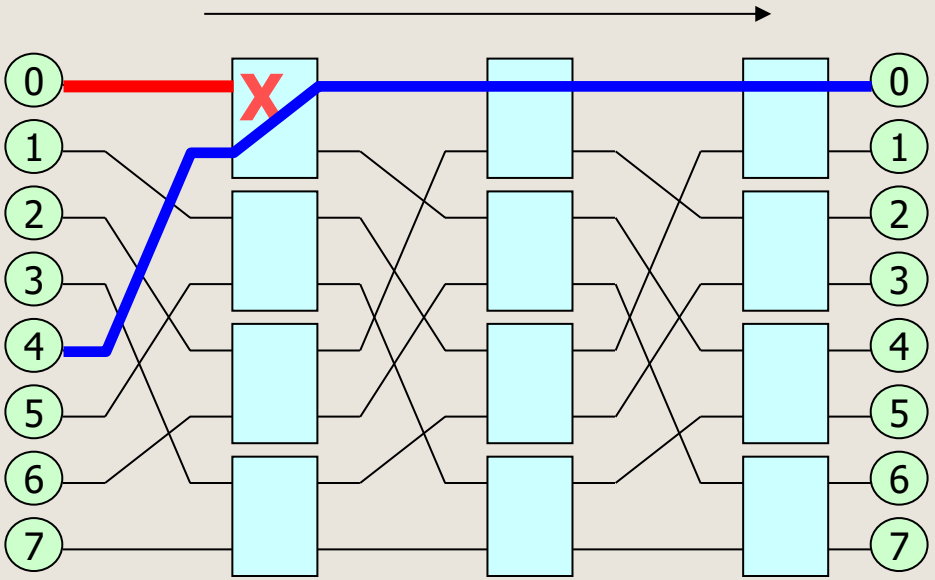
16 port, *4 stage **Cube** network*

Blocking vs Non-blocking

- In general, MINs have the property of being blocking.
  - Contention may occur on network links or switches.
  - Paths from different sources to different destinations may share the same link or the same switch.



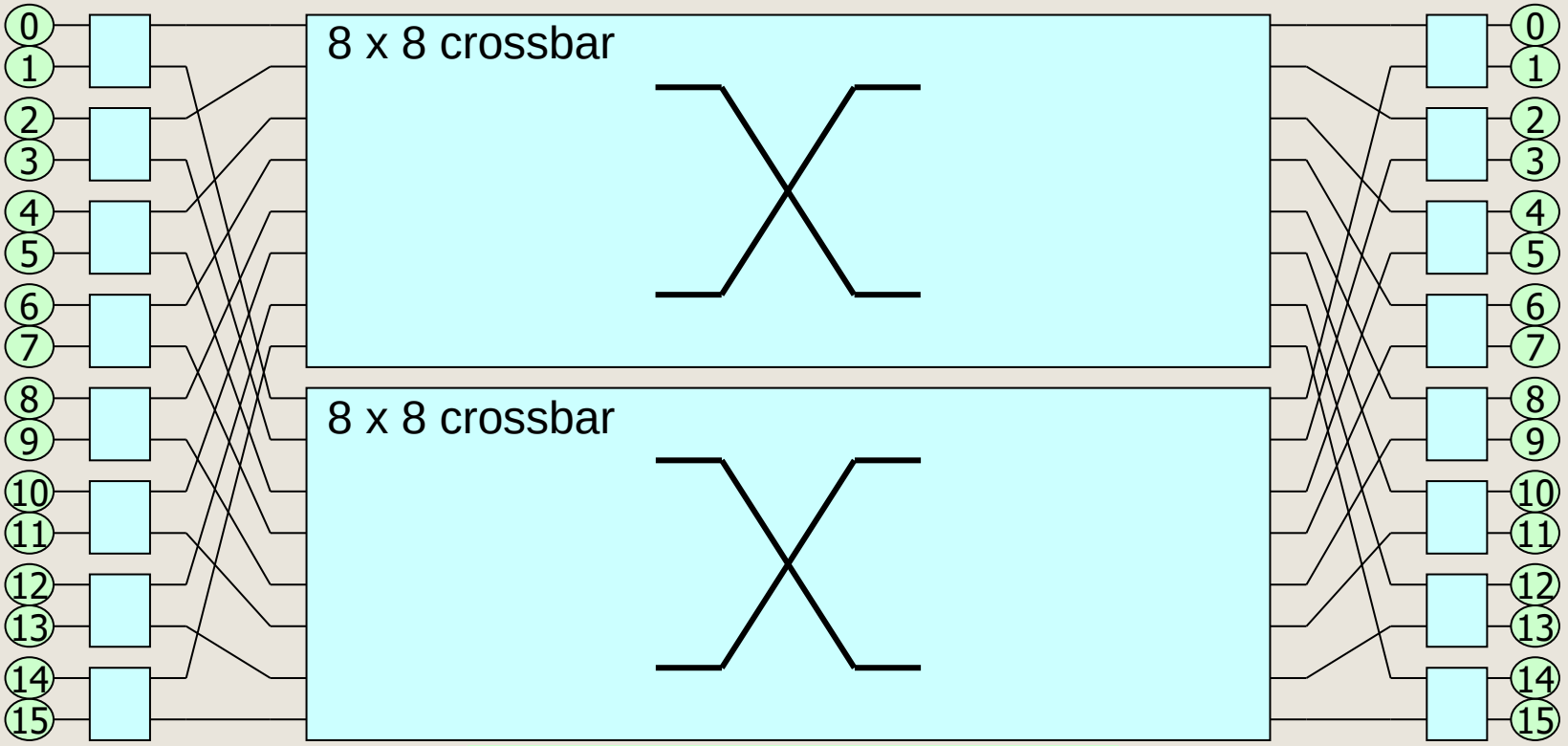
Crossbar is non-blocking



Omega is blocking

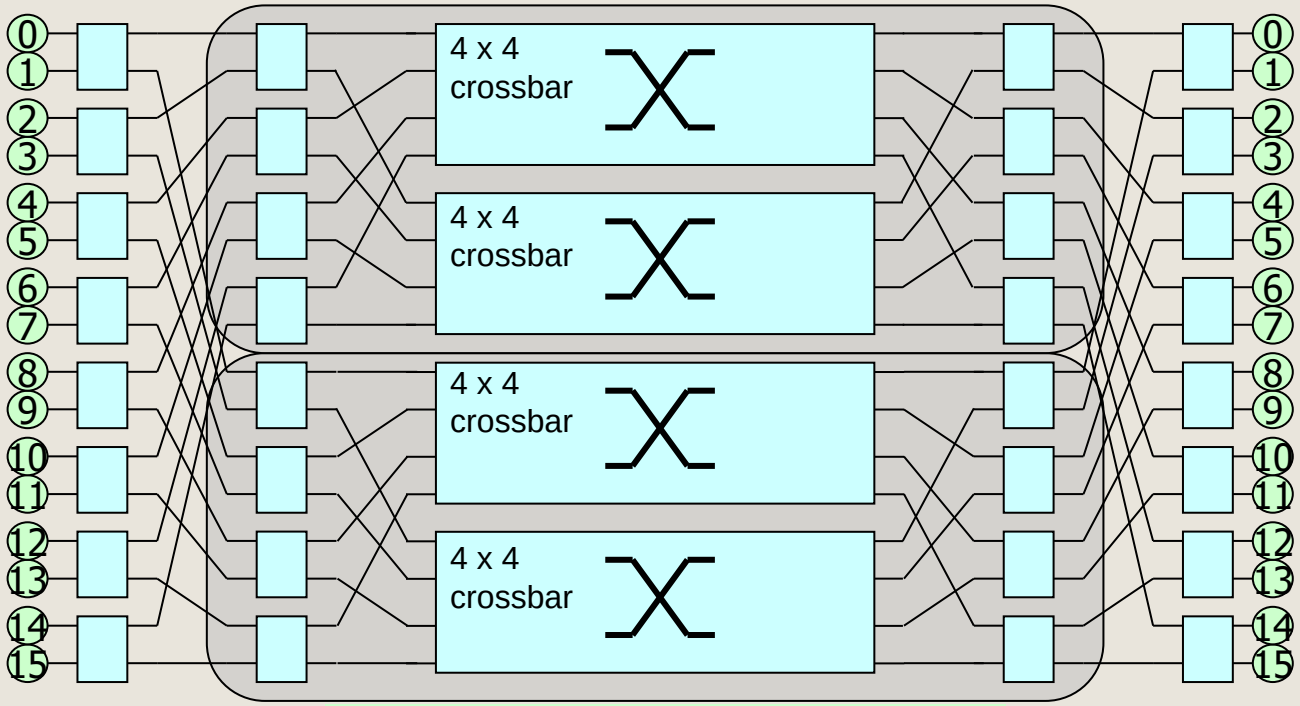
The Non-Blocking MINs

- An N x N clos network is a 3 stage MIN which has two N/2 x N/2 crossbars in the middle stage and N/2 2 x 2 crossbars in both 1st and 3rd stages.



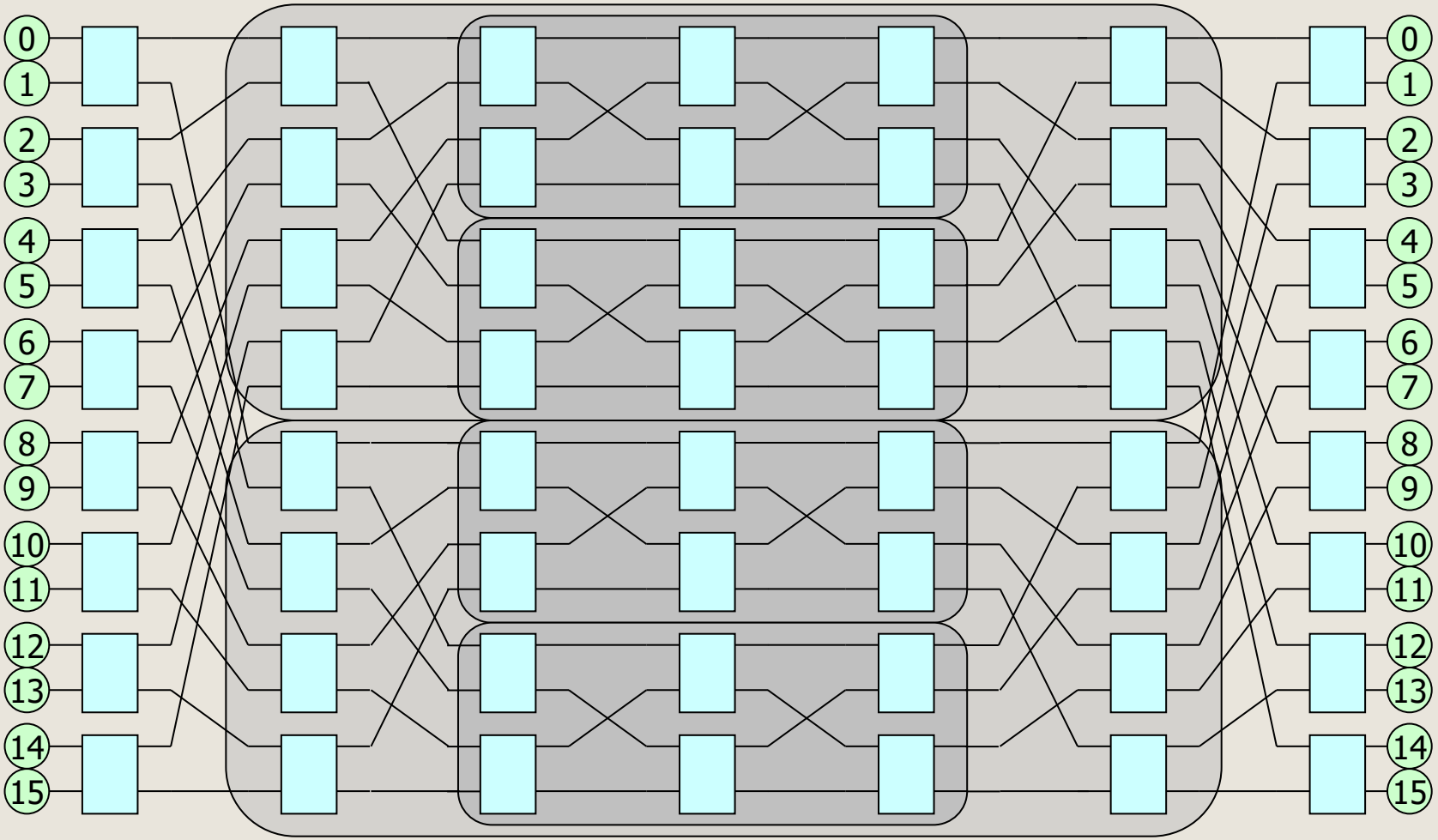
The Benes MIN

- Benes MIN is another non-blocking MIN.
  - It is constructed by recursively replacing crossbars of the middle stage of a clos network to a smaller 3-stage clos until all switches are 2 x 2 crossbars.



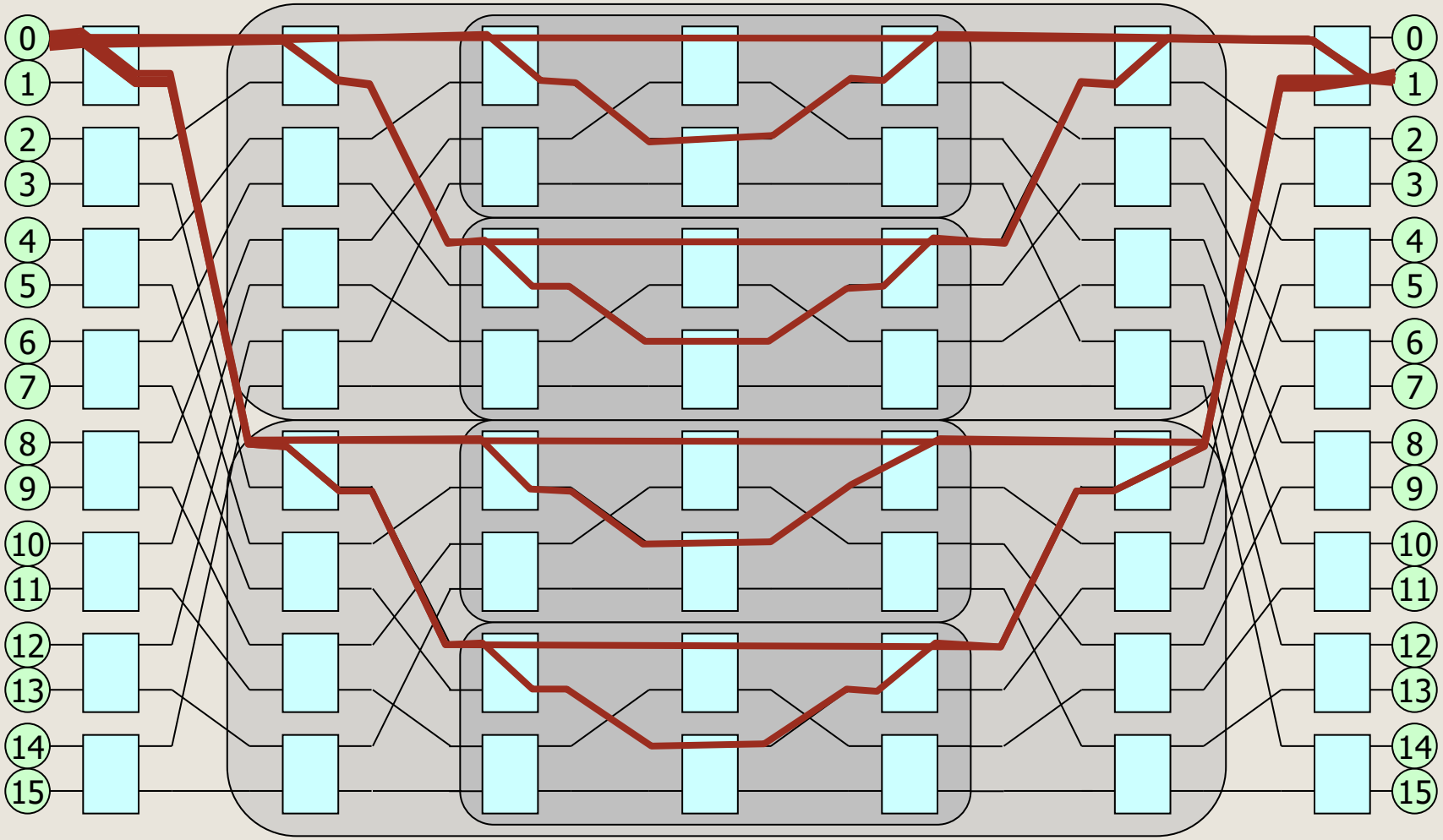
16 port, ***5-stage Clos*** network

The Benes MIN



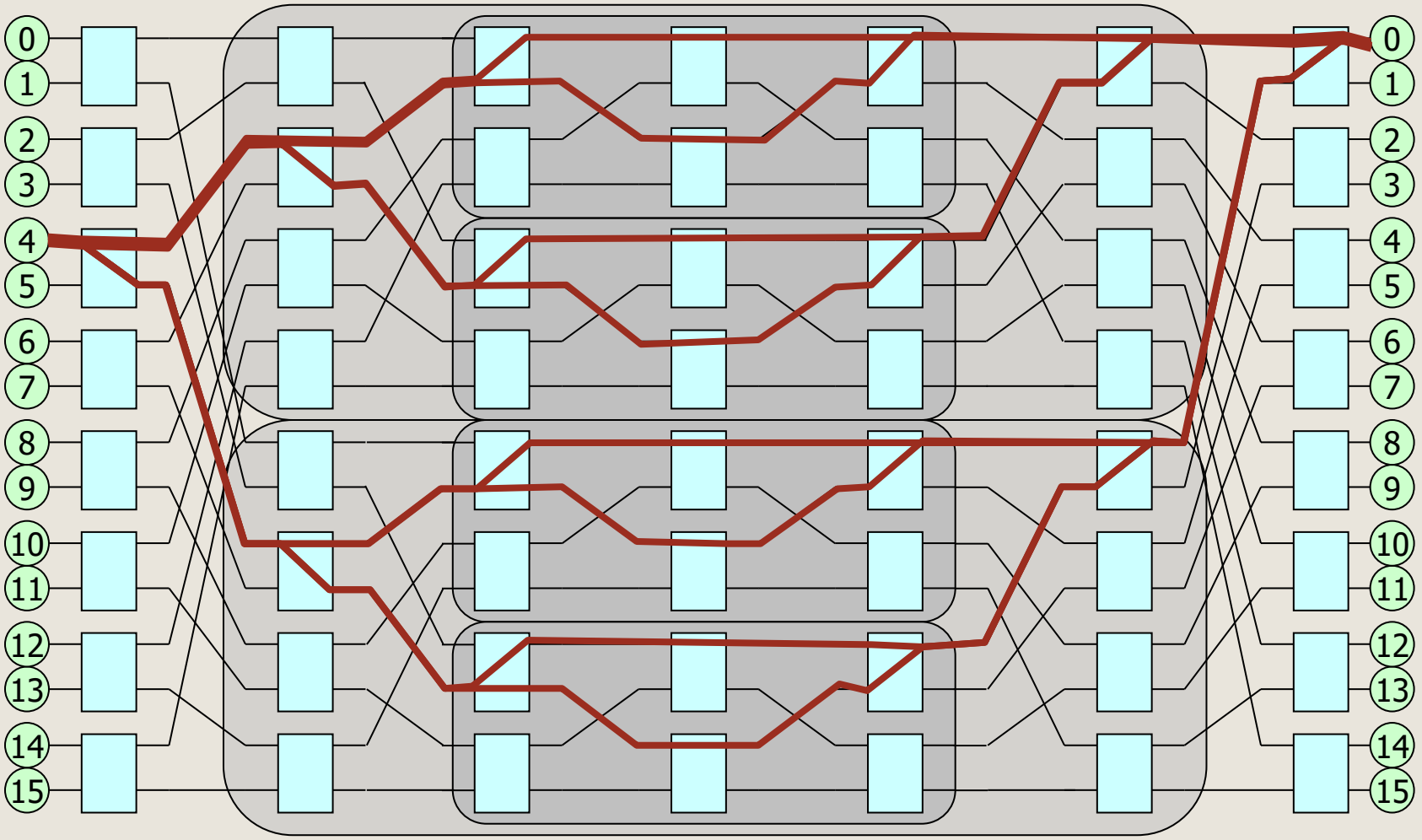
16 port, ***7 stage Clos*** network = ***Benes topology***

The Benes MIN



There are 8 alternative paths from node 0 to node 1 in Benes MIN

The Benes MIN



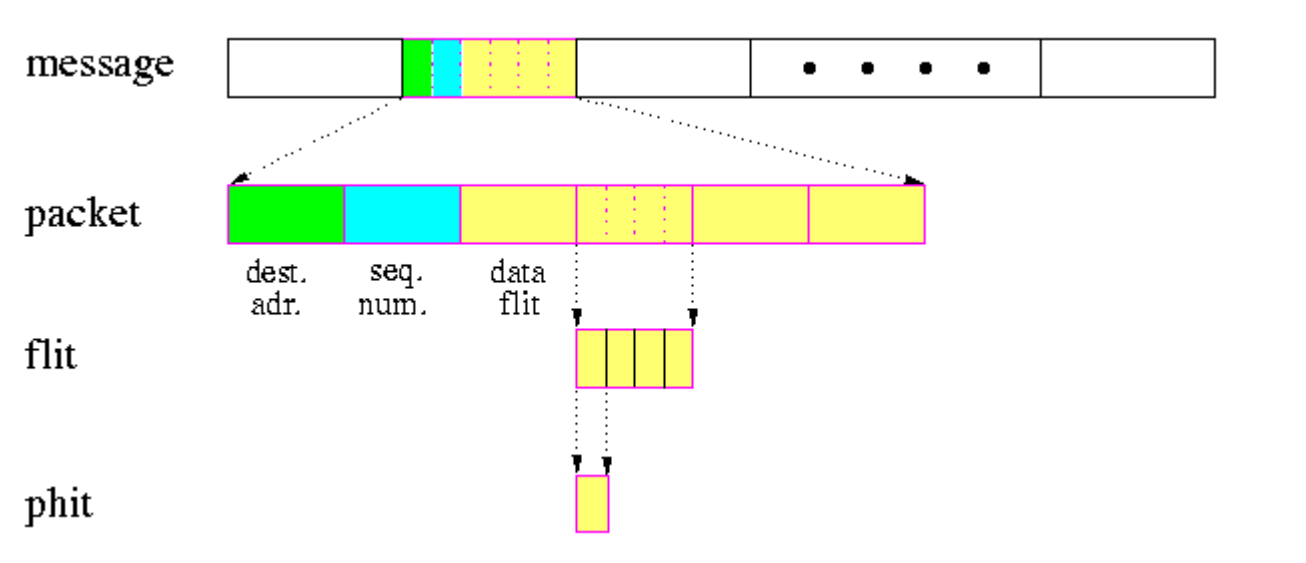
There are 8 alternative paths from node 4 to node 0 in Benes MIN

Clos and Benes are Non-Blocking

- It is easy to show that 3-stage clos network is non-blocking.
  - Since the two crossbars in the middle stage are non-blocking and they do not share links, the 3-stage clos is non-blocking.
- Since a benes network only replaces each large crossbar by a 3-stage clos network and a 3-stage clos network is non-blocking, the benes network is also non-blocking.
- An N x N benes has $2(\log_2 N) - 1$ stages which is about the twice of other blocking MINs.

Switching Schemes

- Message is the unit of communication from the programmer's perspective. Its size is limited only by the user memory space.
- Packet is the fixed-size smallest unit of communication containing routing information and sequencing information in its header. Its size is in the order of hundreds or thousands of bytes or words. It consists of *header flits* and *data flits*.

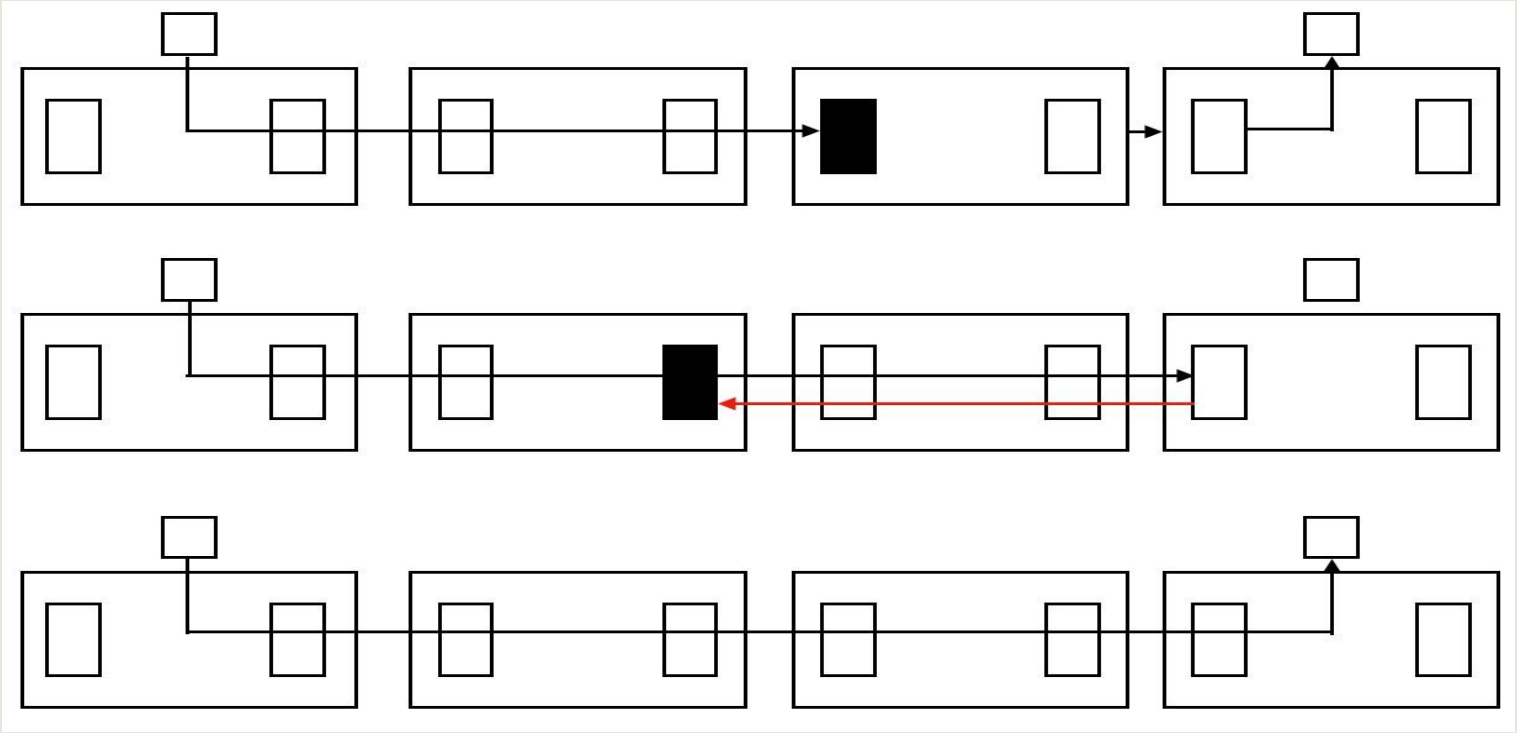


Circuit Switching

- The communication between a source and destination has two phases: circuit establishment and message transmission.
- A physical path from the source to the destination is reserved *prior* to the transmission of data.
  - The source sends a routing probe which contains the destination address and some control information to the destination.
  - The probe reserves physical links when it is transmitted through intermediate switches to the destination.
  - Then an ACK flit is sent back to the source from the destination.
- On reception of the ACK, the sender transmits the whole message at the full bandwidth of the path.
  - The path (all reserved links) is released either by destination node or by the last bits of the message.

Circuit Switching

- In (a), the probe progresses to the destination switch.
- In (b), the ACK is sent back to the source.
- In (c), the whole message is transmitted along the established circuit using its full bandwidth.

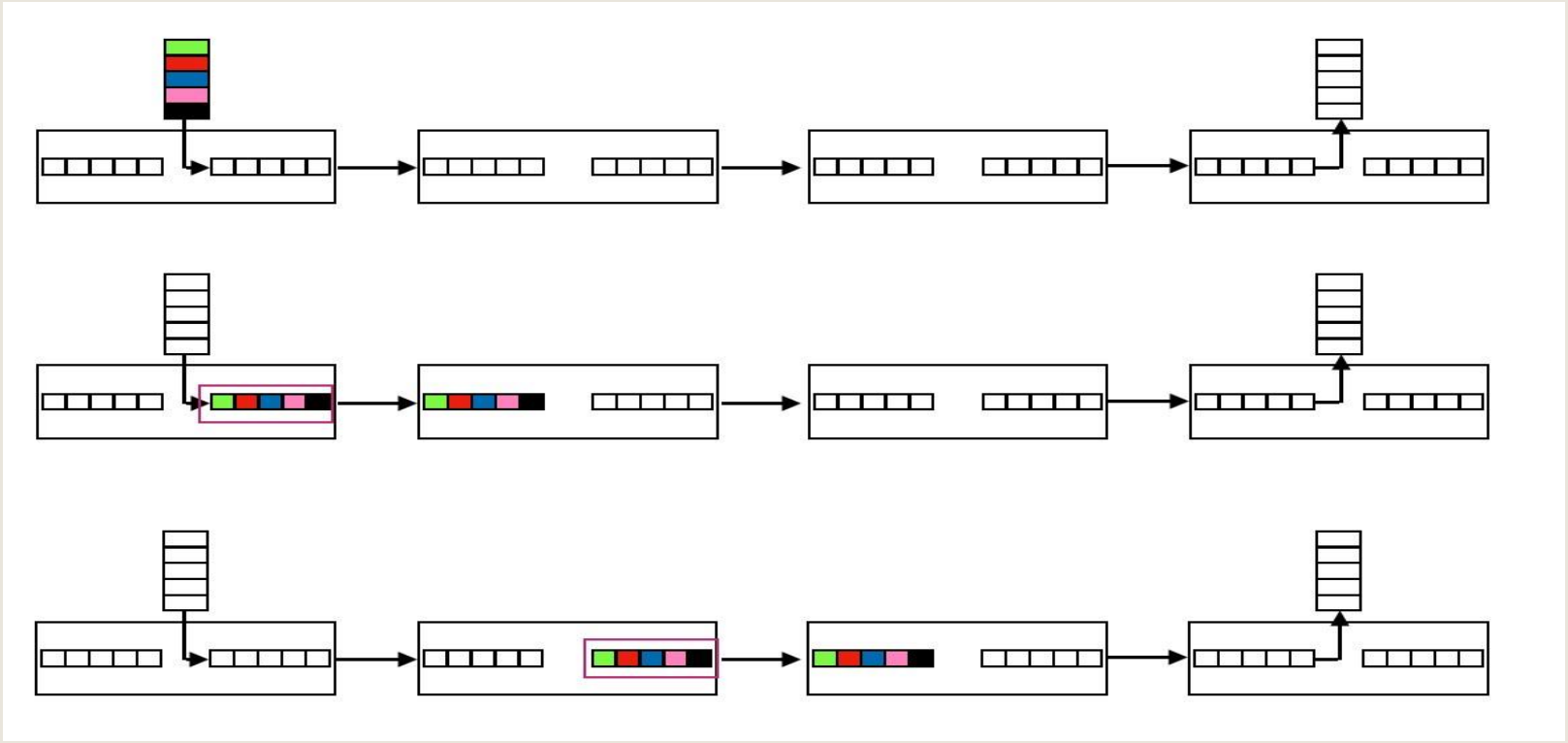


Store and Forward (SF) Packet Switching

- Each message is split into packets.
  - Every packet consists of flits, starting with the header flit.
- Every channel has input and output buffers
  - The size of each buffer can hold at least one entire packet.
- Every packet is individually routed from the source to the destination through intermediate switches.
  - The whole packet must be copied from an output buffer of the current switch to an input buffer of the next switch.
  - Routing decisions are made by each intermediate switch only after the whole packet was completely buffered in its input buffer.

SF Packet Switching

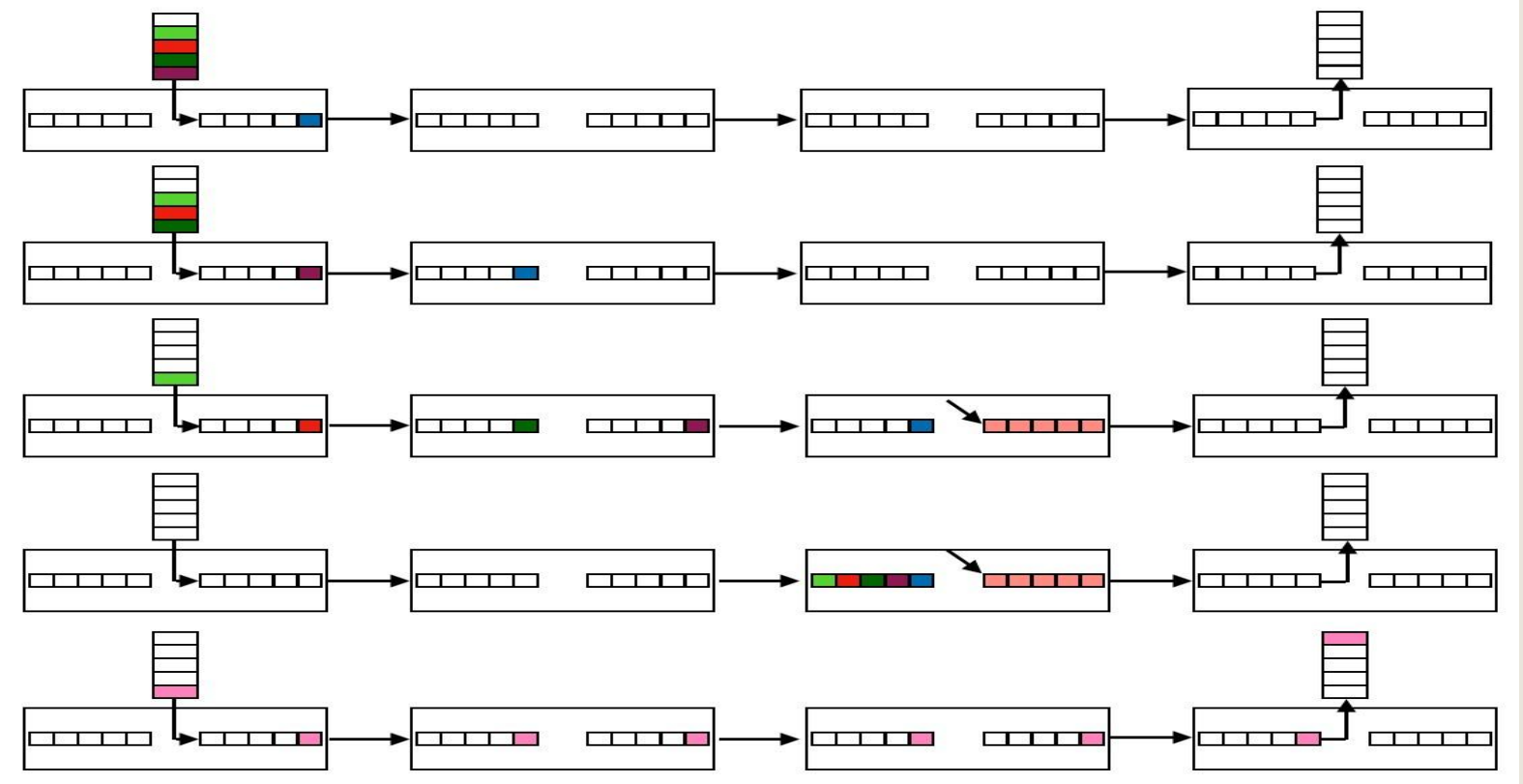
- **In (a), routing decision is being made in the 1st switch.**
- **In (b), the packet is copied to the input buffer of the 2nd switch after it has been copied to the output buffer of the 1st switch.**
- **In (c), the packet is copied to the input buffer of the 3rd switch after it has been copied to the output buffer of the 2nd switch.**



Cut Through (CT) Packet Switching

- Each message is split into packets and each switch have buffers for the whole packet as in SF packet switching.
  - The incoming header flit is cut through into the next switch as soon as the routing decision was made and the output channel is free.
    - Every further flit is buffered whenever it reaches a switch, but it is also cut-through to the next switch if the output channel is free.
  - If the header cannot proceed, it waits in the current switch.
    - All the data flits will continue to flow into the current switch.
    - This will release the channels occupied by data flits.
  - In case of contention free, a packet is effectively pipelined through successive switches as a loose chain of flits.
    - All the buffers along the routing path are blocked for remaining data flits.

CT Packet Switch

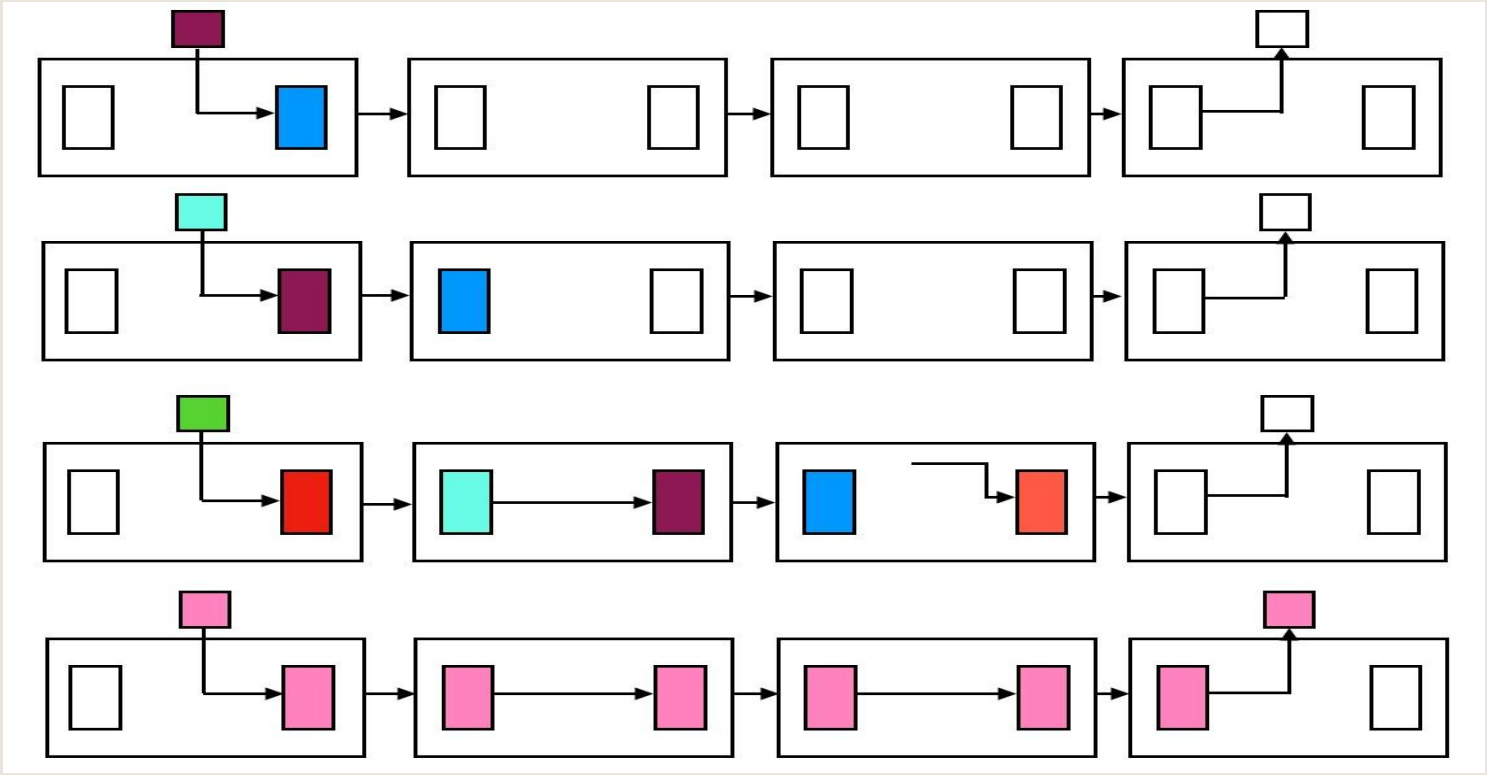


(a) The header flit is in the output buffer of the 1st switch.
(b) The header cut through into the 2nd switch while data flits are following its path.
(c) The header cut through into a switch where its output buffer was occupied.
(d) The flits of whole packet are buffered in the current switch and releasing all allocated links.
(e) Flits are pipelined and moved to the destination.

Wormhole (WH) Packet Switching

- The packets are split to flits which are snaked along the switches exactly in the same pipeline way as in conflict-free CT switching.
- The main difference between WH and CT switching are:
  - WH switches do not have buffers for the whole packets.
  - Every switch has small buffers for one or a few flits only.
- The header flit again builds a path in the network.
  - The sequence of buffers and links occupied by flits of a given packet forms the wormhole.
- If the header cannot proceed due to busy output channels, the whole chain of flits gets stalled.
  - They occupy buffers in switches on the path and block other traffics.

WH Packet Switching



(a) The header flit is copied in the output buffer of 1st switch after making routing decision.

(b) The header flit is transferred to the 2nd switch and other flits are following it.

(c) The header flit arrived into a switch with busy output channel and the whole chain of flits along the path got stalled. They block all corresponding channels.

(d) In case of conflict-free, pipeline of flits establishes the wormhole across switches on the path.

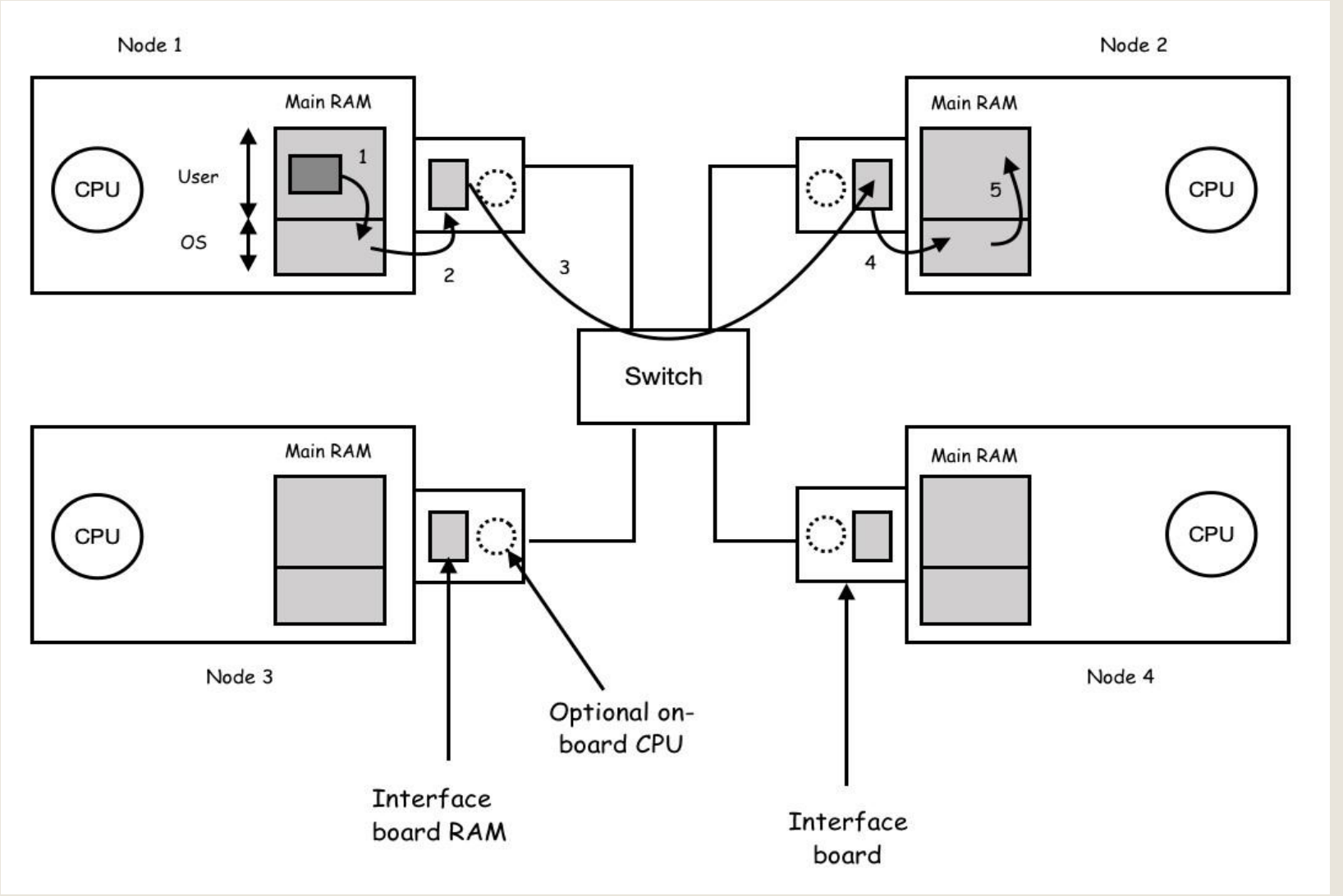
Network Interfaces for MCs

- In general, all MCs have network interface boards (NIBs) to connect processing nodes into the IN.
- The NIB usually contains its own RAM for buffering incoming and outgoing messages (packets.)
- The NIB may have one or more DMA channels or a special CPU on board.
  - DMA channels are for block data transfer between NIB RAM and main memory.
  - The CPU (network processor) is used to offload main CPU from
    - reliable transmission and multicasting
    - compression/decompression and encryption/decryption

Communication Among Computers

- Since there is no shared memory in MC, processes on separate computers communicate by messages.
  - Message copying is the major barrier for achieving high performance.
  - A message is delivered through the following phases.
    - A message is first moved to the buffer of the sender NIC.
    - It is then moved across the IN.
    - Finally, it is copied into the buffer of the receiver NIC.
  - Additional copies may be needed between user and kernel space.
    - Map NIC into user space may reduce copies but it creates some additional synchronization problem.
- Using 2 NIBs for each computer may reduce data copies.
  - One for OS and the other one for user processes.

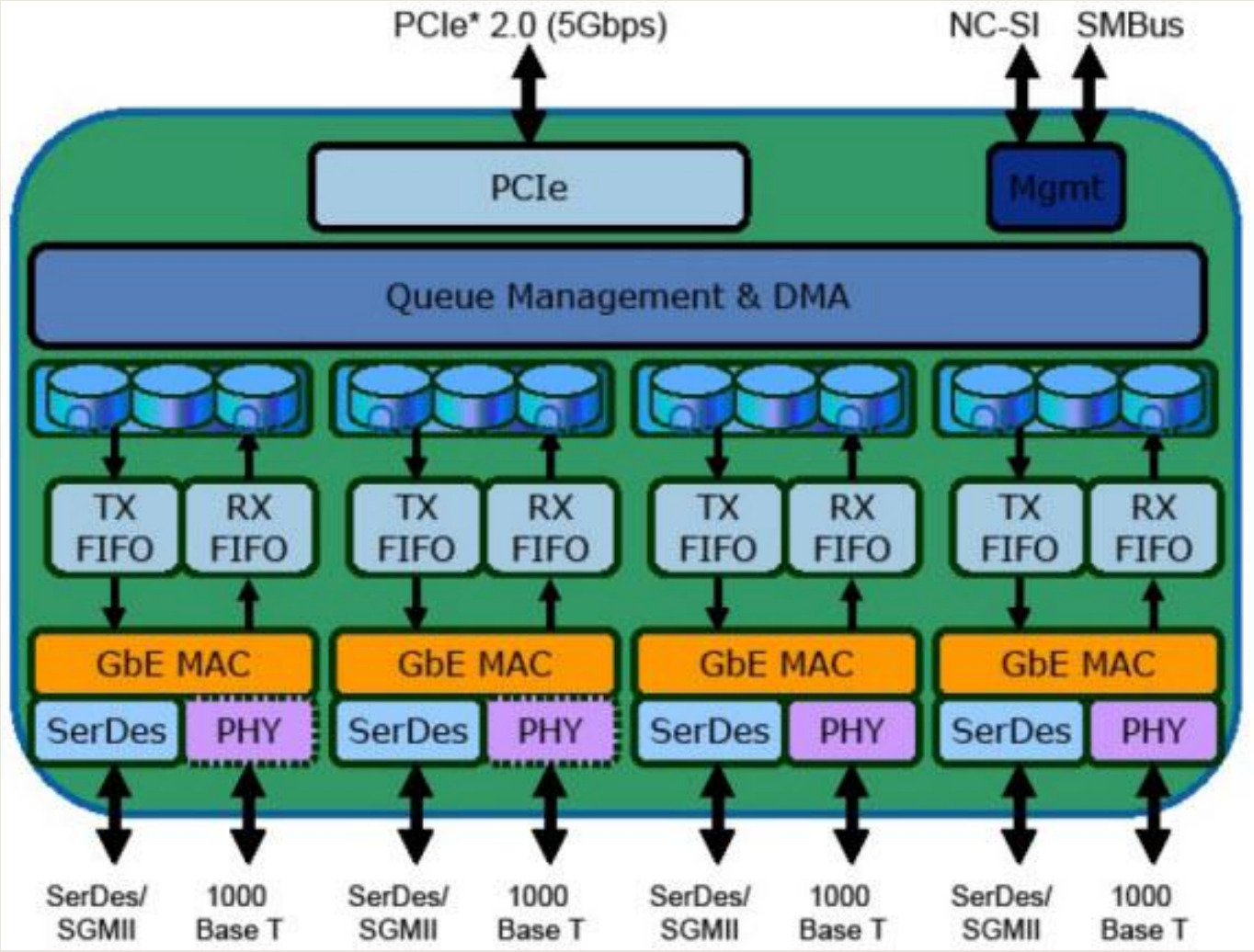
Data Copying in Comm. for MC



Multi-queued NIB

- Some NIBs support multiple queues.
  - Packets of different users are handled by different queues.
- Intel I350 series is a typical example.
  - It has up to 4 Ports.
    - Each port has 8 sending queues and 8 receiving queues.
  - It also supports core affinity such that packets of the same TCP flow can be handled by the same core.
  - It uses a hashing logic to hash the IP addresses and port numbers such that packets with the same hash value are handled by the same queue.

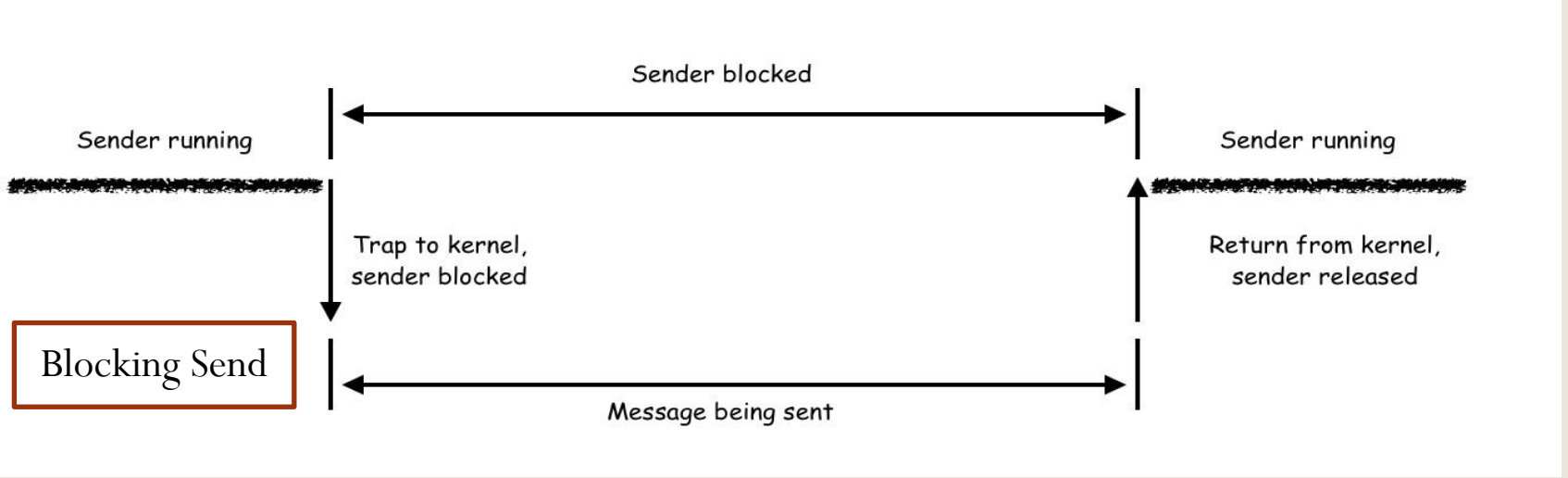
The Intel I350

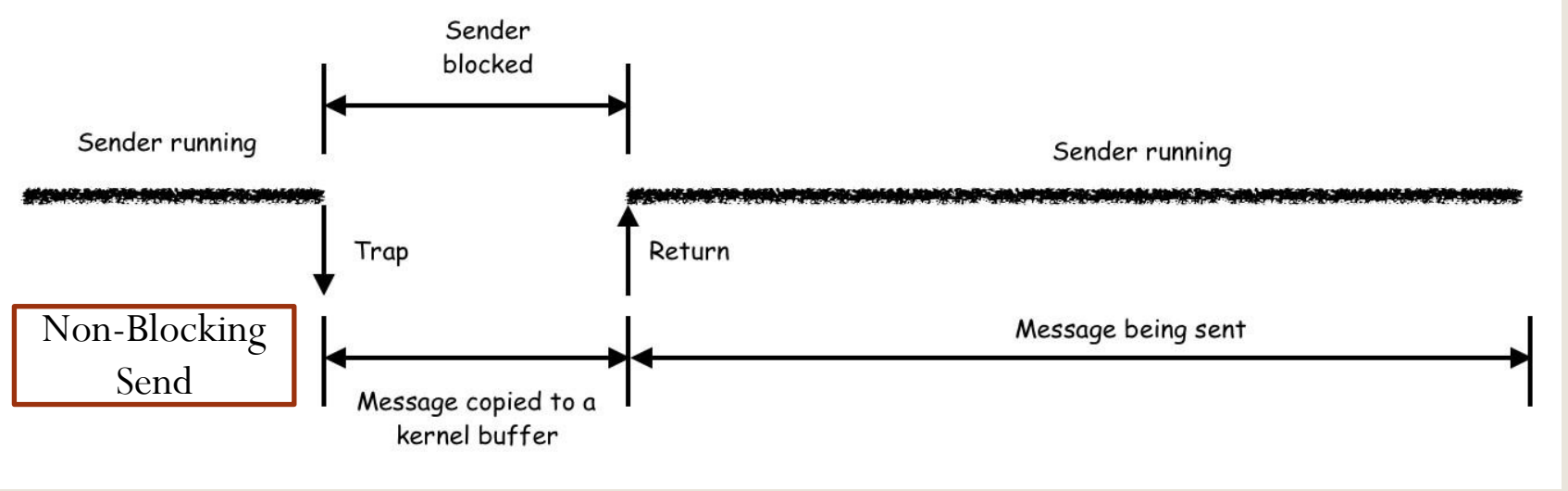


Message Passing in MC

- The typical messaging operations are
  - send (destID, &mptr)
    - It sends the message pointed to by the mptr to a process identified by destID.
  - receive (sendID, &mptr)
    - On arriving a message, the message is copied to the buffer pointed to by the mptr and the receive call is returned.
    - The sendID specifies the process to which the caller is listening.
- Blocking Calls
  - The caller of the send is blocked until the message has been sent.
  - The caller of the receive is blocked until a desired message arrives.

Blocking and Non-blocking Sends





Blocking and Non-blocking Sends

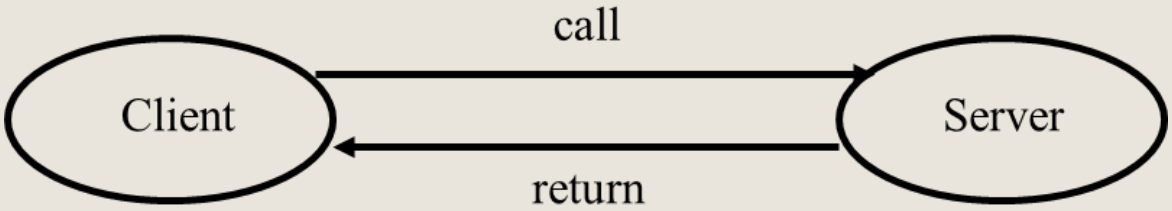
- Blocking send: a sender waits until the message is sent
  - Disadvantage: Process has to wait
- Non-blocking send: the call returns to sender immediately
  - The buffer must be protected until the message is sent.
  - Interrupt sender upon completion of message transmission.
  - Possible ways of handling non-blocking send:
    - Copy the message into kernel buffer.
    - Mark the page of the message buffer as read-only and perform copy-on-write on the marked page.
- In general, the blocking call is the most convenient.
  - In multithreaded system, a blocked thread will not affect other threads to continue their works.

Blocking and Non-blocking Receives

- Blocking receive: the caller is blocked until a message arrives
  - It is good for multithreaded system.
- Non-blocking receive: it tells the kernel where the buffer is and returns immediately.
  - Needs a way to tell the receiver the arrival of a message.
    - Interrupt the receiver when a message arrives.
    - The caller polls the status of the buffer periodically and retrieves the message when the state changes.
      - Needs poll() and get_message() operations
    - On receiving a message, a pop-up thread is created to handle the message.
    - Process the arriving active message inside the interrupt handler.
      - It only works in a completely trusted environment.

Remote Procedure Call (RPC)

- Message passing model is flexible, but it is not a natural programming model
- Procedure call is a more natural way for programming
- Basic idea of remote procedure call (RPC)
  - A server exports a set of procedures that can be called by client programs.
  - A client makes a local call to the exported remote procedure.
    - The local procedure call is converted into a message exchanging with the remote server process.



The RPC model

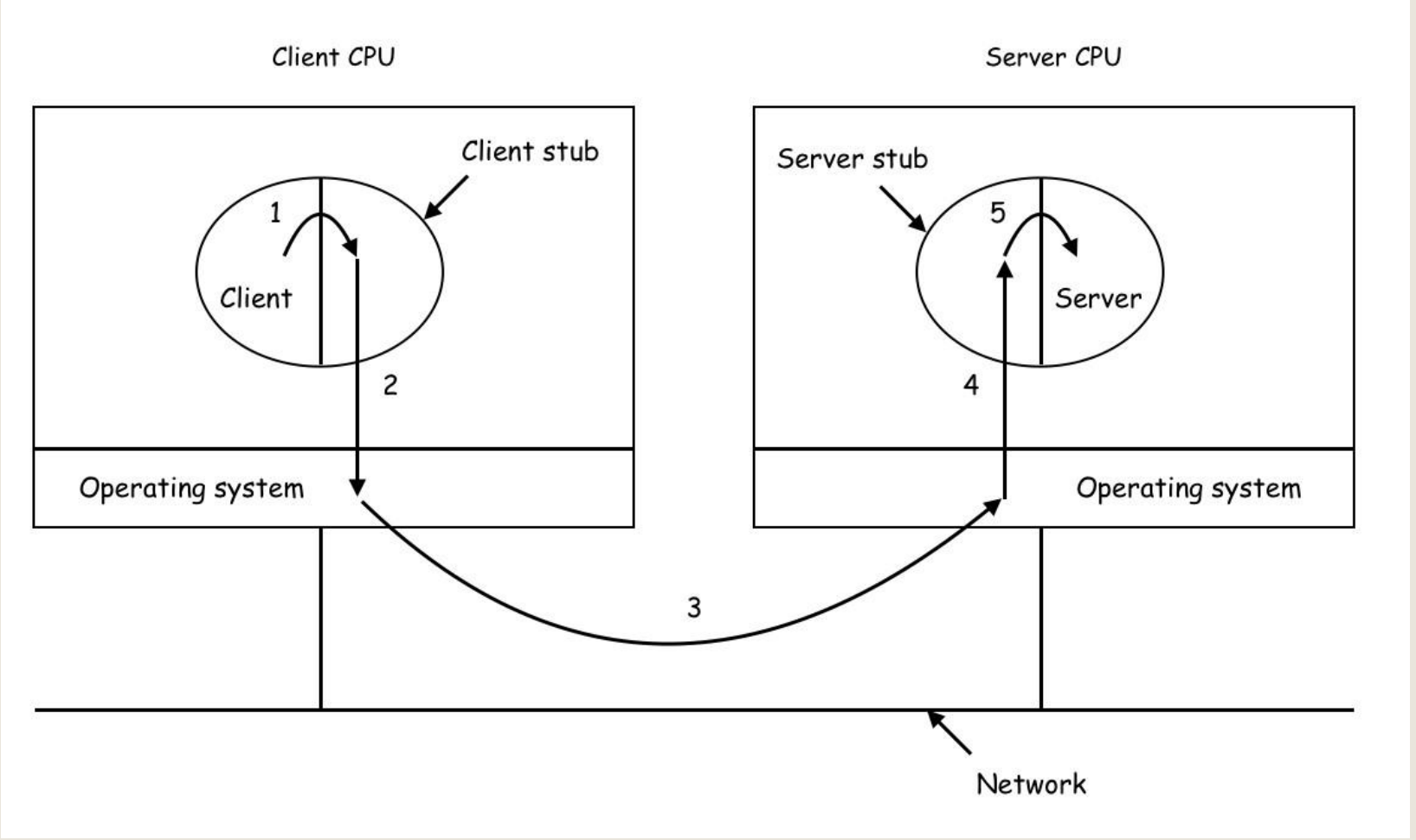
- Use procedure call as a model for process communications
  - It offers a good programming abstraction to hide low-level communication details
- Goal: make RPC look as much like local call as possible
- Many issues:
  - how do we make this invisible to the programmer?
  - what are the semantics of parameter passing?
  - how is binding done (locating the server)?
  - how do we support heterogeneity (OS, arch., language)?
  - how to deal with failures?

Steps in Making an RPC

- There are 3 modules on both end of an RPC system:
  - a user program (client or server)
  - a set of stub procedures (client side and sever side)
  - an RPC runtime support module
- Steps in making an RPC
  - A client invokes a library routine called the client stub
    - It may contain parameters
  - The client stub generates a message to be sent
    - The parameters are marshaled as the message body
  - The client side kernel sends the message to the server
  - The server side kernel calls the server stub
  - The server stub invokes the service procedure after
    - it un-marshals the message into parameters

The Data Flow of an RPC

The RPC return traces the same path in the other direction.



Issues in RPC Marshaling

- The byte order issues:
  - different data representation (ASCII, UNICODE, EBCDIC)
  - big-endian versus little-endian
  - alignments, etc.
- The parameter passing issues:
  - passing pointer:
    - copy on demand (useless for complex structures)
  - passing unbounded array:
    - size of an array in C can be undefined in compiling time
  - unbounded number of parameters:
    - printf in C has different number of parameters
  - using global variables as parameters

Failures in RPC

- Cannot locate the server
  - server down
  - version mismatch
  - other exceptions
- Request message is lost
- Reply message is lost
- Server crashes after receiving a request
- Client crashes after sending a request

Message Lost Handling

- Request message is lost
  - use timer and resend the request message
- Reply message is lost
  - use timer and resend the request message
  - server needs to handle duplicated request message unless the RPC function is idempotent
- Make all requests idempotent
  - redefine read (fd, buf, n) to read (fd, buf, pos, n)
  - deposit (money) is not possible to make it idempotent
- Assign a sequence number to each request and keep track

Semantics of Node Crash Handling

- Do nothing and leave it up to the user
- At least once:
  - On successful return, the request is executed at lease once.
  - Only good for idempotent functions
- At most once:
  - After crash recovery, a request is executed at most once.
  - Need to suppress duplicated requests
    - Client provides each request an unique sequence number
    - Server saves results for each request
- Exactly once:
  - Each request has an unique sequence number
  - Both client and server log requests and replies

Shared Memory vs. Message Passing

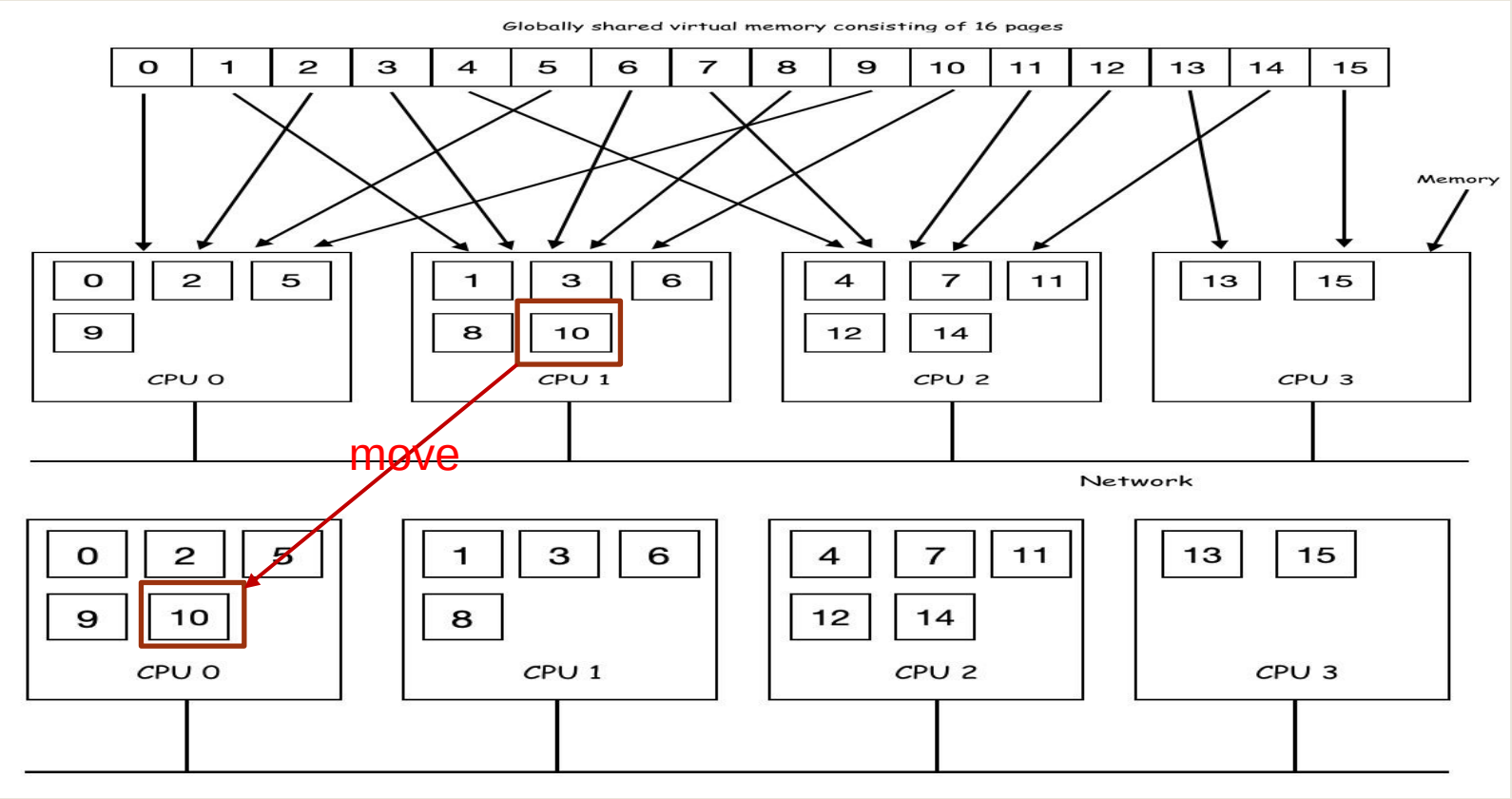
- Message passing:
  - Information exchanged are well defined and managed
- Shared memory:
  - Processes exchange information by shared variables
    - Since there is no explicit comm. primitives (send and receive) needed in programming, it is believed to be easier to design and write parallel programs by using shared memory.
  - It is easy to port a cooperative (multithreaded or multi-processed) application from a centralized system to a MC system.
  - When scaling the MC system by either adding more memory to each node or adding more nodes, there is no need to rewrite codes

Distributed Shared Memory (DSM)

- A method of allowing processes on different computers to share regions of virtual memory
  - Implementation is essentially paging over the network
- Backing file rests in globally accessible storage
- Read-only pages can be replicated to improve performance
- Writeable pages are managed by the following ways:
  - Single copy only
    - The last writer keeps the copy.
  - Multiple copies
    - Initially, all pages are read-only.
    - To write a page, a node first invalidate all other copies.
    - It then writes the local copy and propagates to all other copies.

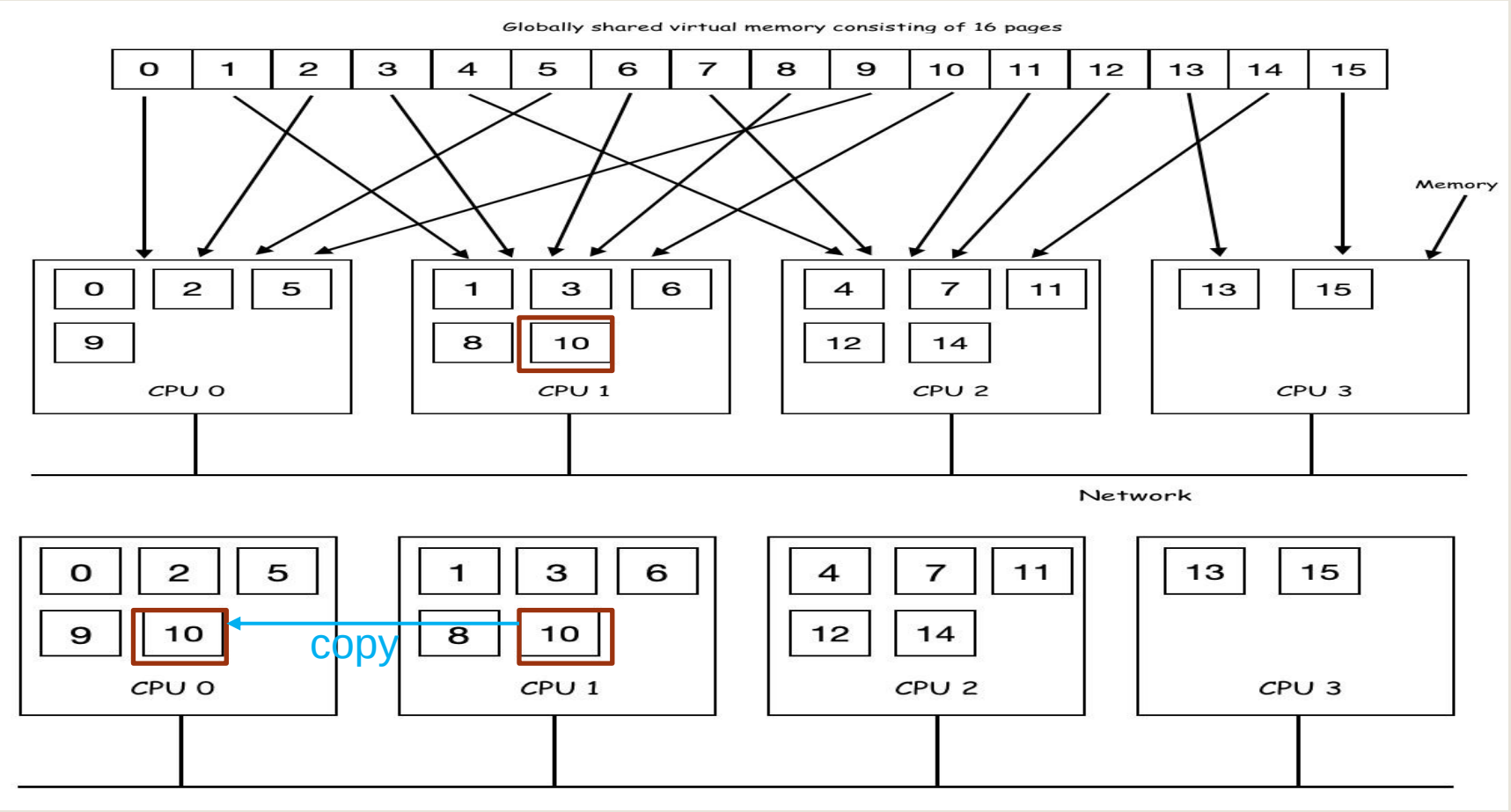
A DSM of 4 Machines

- Pages of the VM are distributed among four machines.
- In single copy only DSM, the page 10 is moved to CPU 0 after it is referenced by CPU 0.



A DSM of 4 Machines

- In multiple copy DSM, the page 10 is duplicated to CPU0 after it is referenced by CPU0.



Properties of DSM

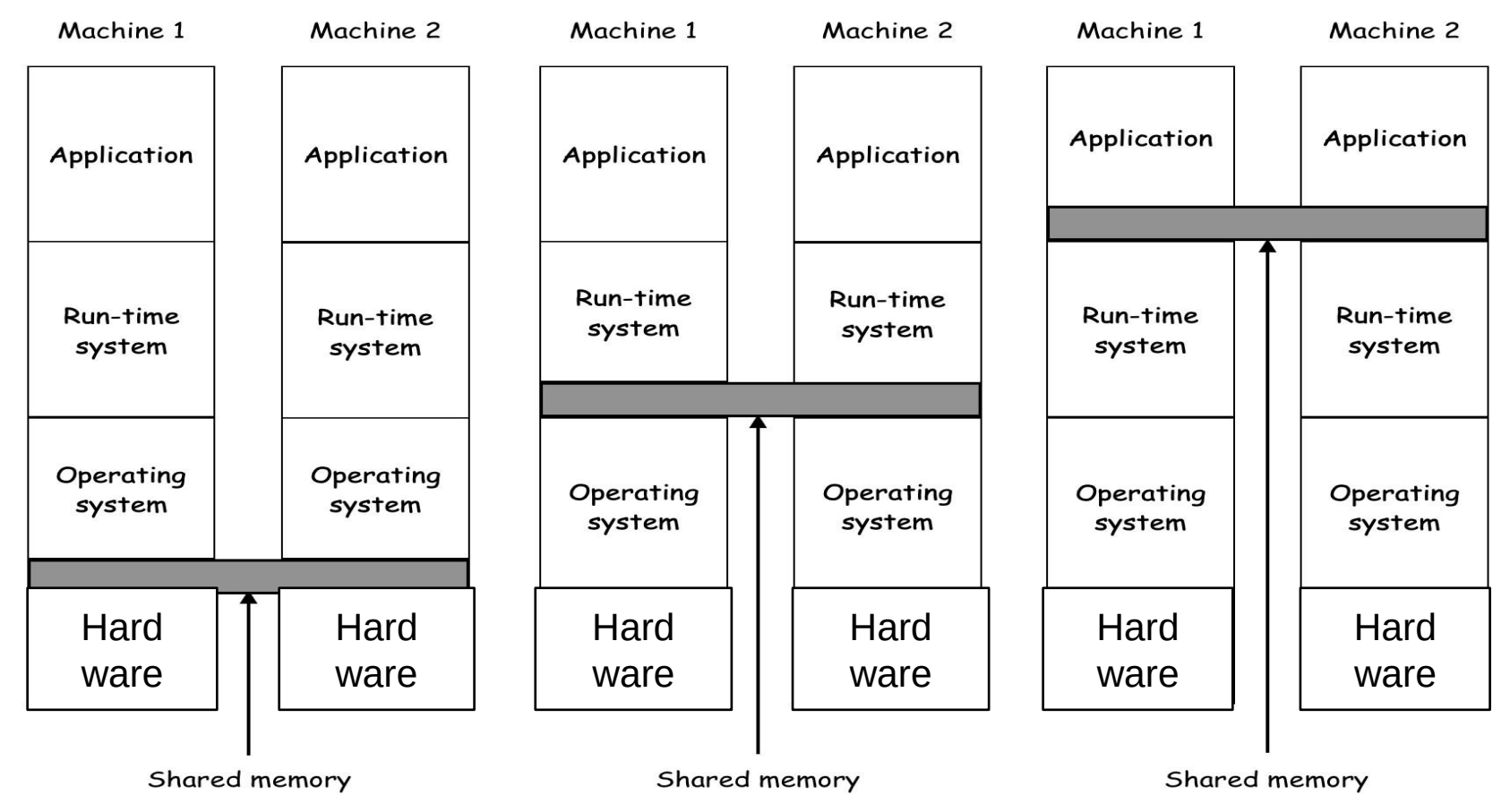
- Data in shared address space is accessed as the virtual memory of a centralized system.
  - A mapping manager is used to map the shared address space to the physical address space.
- Complex data structure can be passed by reference.
- Paging over network takes advantage of locality and may reduce communication overhead.
- It makes programs written for a MP become portable and scalable to a larger MC with DSM.

DSM Implementation Issues

- In virtual memory (VM), OS hides the fact that pages may reside in main memory or on disk.
- In MPs, there is a single shared memory (possibly virtual) accessible by multiple CPUs.
  - There may be multiple caches, but cache coherency protocols hide this from applications.
- Major goal of DSM: make shared data concurrently accessible
  - In DSM, each machine has its own RAM, but the VM is shared, so pages can reside in any RAM or on disk.
    - The location of shared data must be keep track
  - On page fault, OS can fetch the page from a remote RAM
    - The comm. delays and cache coherency protocol overhead must be minimized

DSM Implementation Types

- There are various layers where DSM can be implemented.



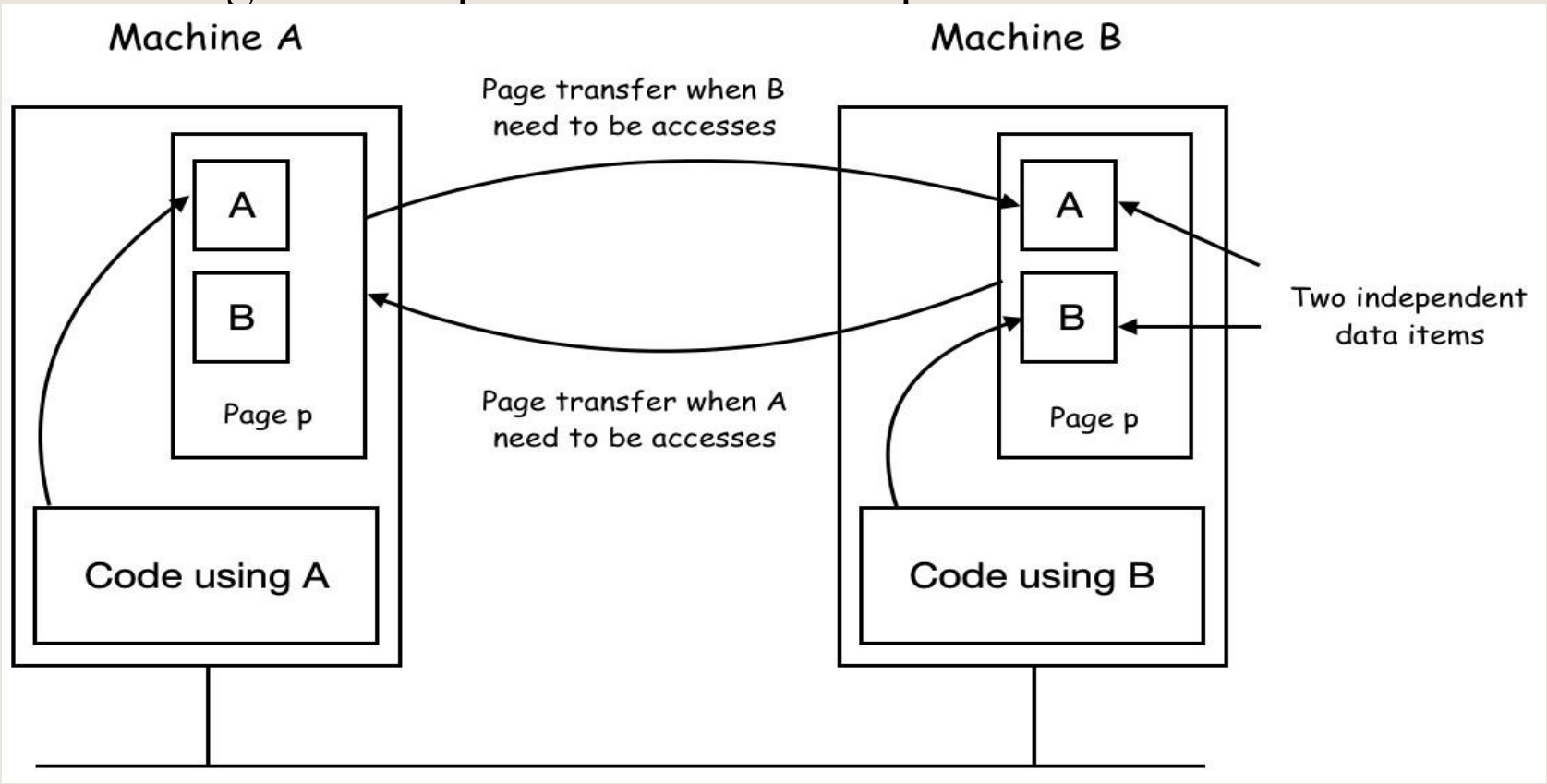
(a) The hardware (b) The OS (c) User-level software

DSM Implementation Details

- Every computer has its own page-table
- If accessed page is not in local RAM, a message is sent to lookup where it resides, and the page is fetched
  - A lookup algorithm for locating pages
  - A migration algorithm for paging and coherence control
- Replication is used to reduce the traffic
  - The read-replication algorithm
    - As in cache coherence for multiple caches in a MP, a page can reside on multiple nodes with read-only flag set.
    - All other copies must be invalidated before a page can be written
  - The full-replication algorithm
    - Using a global lock, a CPU must acquire the lock before the actual write.
    - On releasing the lock, changes are propagated to all other copies.

False Sharing in DSM

- There may exist false sharing of a page between two independent processes.
- Needs a good compiler to reduce the problem.



MC Scheduling

- In a MP system , an idle CPU may select any ready process to run.
- Since there is no shared memory in a MC system, each node has its own set of processes and scheduler.
  - When a new process is created, it is assigned to a particular node and stay there until it is finished.
    - The decision is usually based on load balancing.
- With a coordinator on one particular node, gang scheduling may be possible for a MC system.
  - All nodes must synchronize in the start of each time slot.

Load Balancing in MCs

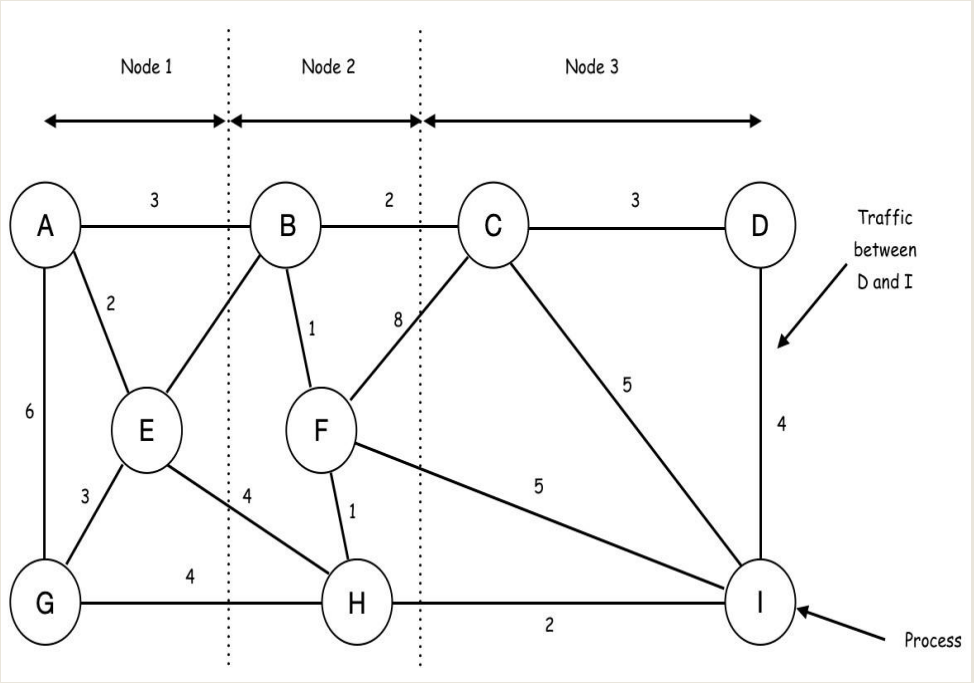
- In a MC system, the most important activity is how processes can be assigned to nodes in an effective way.
  - Processor allocation (PA) algorithms are used to deal with it.
- Major factors concerned by a PA algorithm include
  - Estimated CPU and memory requirements of a process
  - Estimated comm. traffic between any pair of processes
- Possible goals of PA algorithms are
  - Minimizing wasted CPU cycles due to lack of works
  - Minimizing total comm. bandwidth
  - Ensuring fairness to users and processes

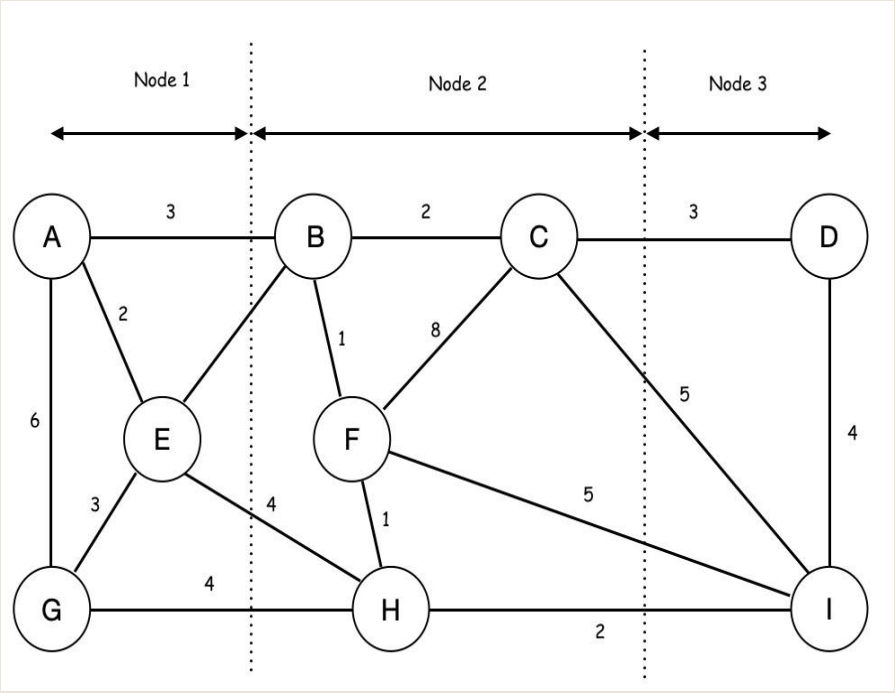
A Deterministic PA Algorithm

- Assumptions:
  - Known CPU and memory requirement for each process
  - Known average traffic between each pair of processes
- A system is represented as a weighted graph.
  - Each vertex represents a process
  - Each arc represents the traffic between two processes
- The PA is then reduced to find an effective way for cutting the graph into k disjoint subgraphs subject to some constraints.
  - In each feasible solution, arcs among subgraphs represents real network traffic.
  - Thus, the goal is to find a feasible solution with minimized network traffic.

Two Feasible Partitions

- The left solution has 13+17=30 units of network traffic.
- The right solution has 13+15=28 units of network traffic.



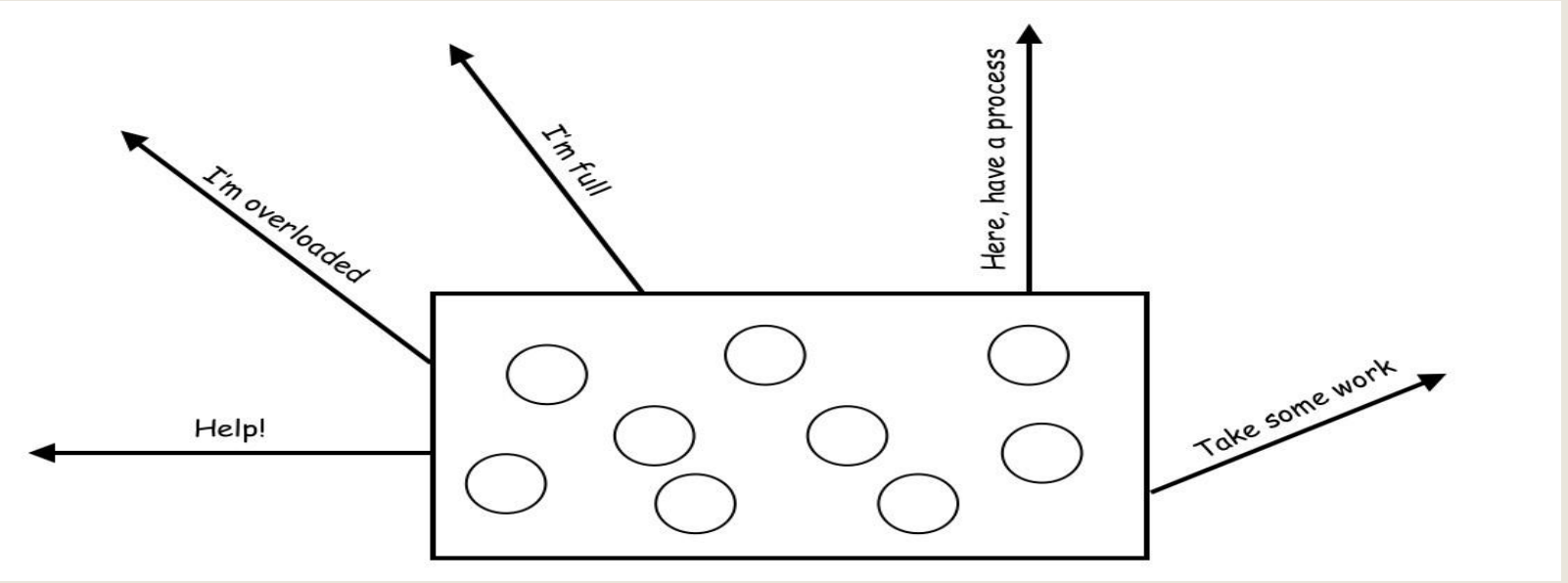


Heuristic Load Balancing Algorithms

- Finding optimal PA is computationally expensive
  - NP-hard (must try all possible combinations in the worst case)
  - Greedy heuristics may perform well
- Dynamic adaptation schemes:
  - Sender Initiated Schemes
    - A new process is always assigned to the node which creates it.
    - If the node is overloaded, it probes others for help.
    - On receiving a probe, if a node is under loaded, it accepts the new process.
  - Receiver Initiated Schemes
    - If a node is under loaded, it probes others for process migration.
    - On receiving a probe, if a node is overloaded, it migrate some processes to the probing node.

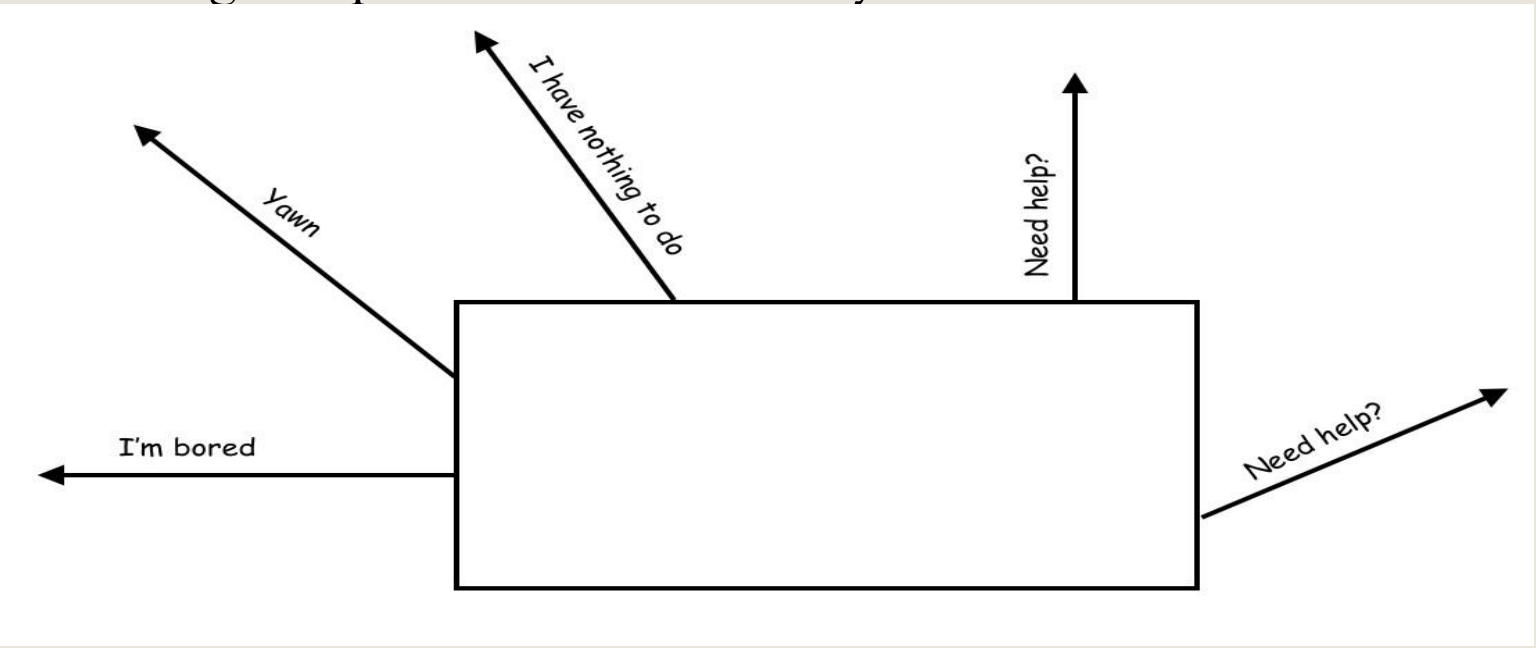
Sender Initiated Load Balancing

- A new process is always assigned to the node which creates it.
- If the node is overloaded, it randomly probes a node for help.
  - If the probed node is under loaded, it accepts the new process.
  - Otherwise, the probing node repeatedly to probe for help until an upper limit N.
- It does not perform well in heavily loaded situation.



Receiver Initiated Load Balancing

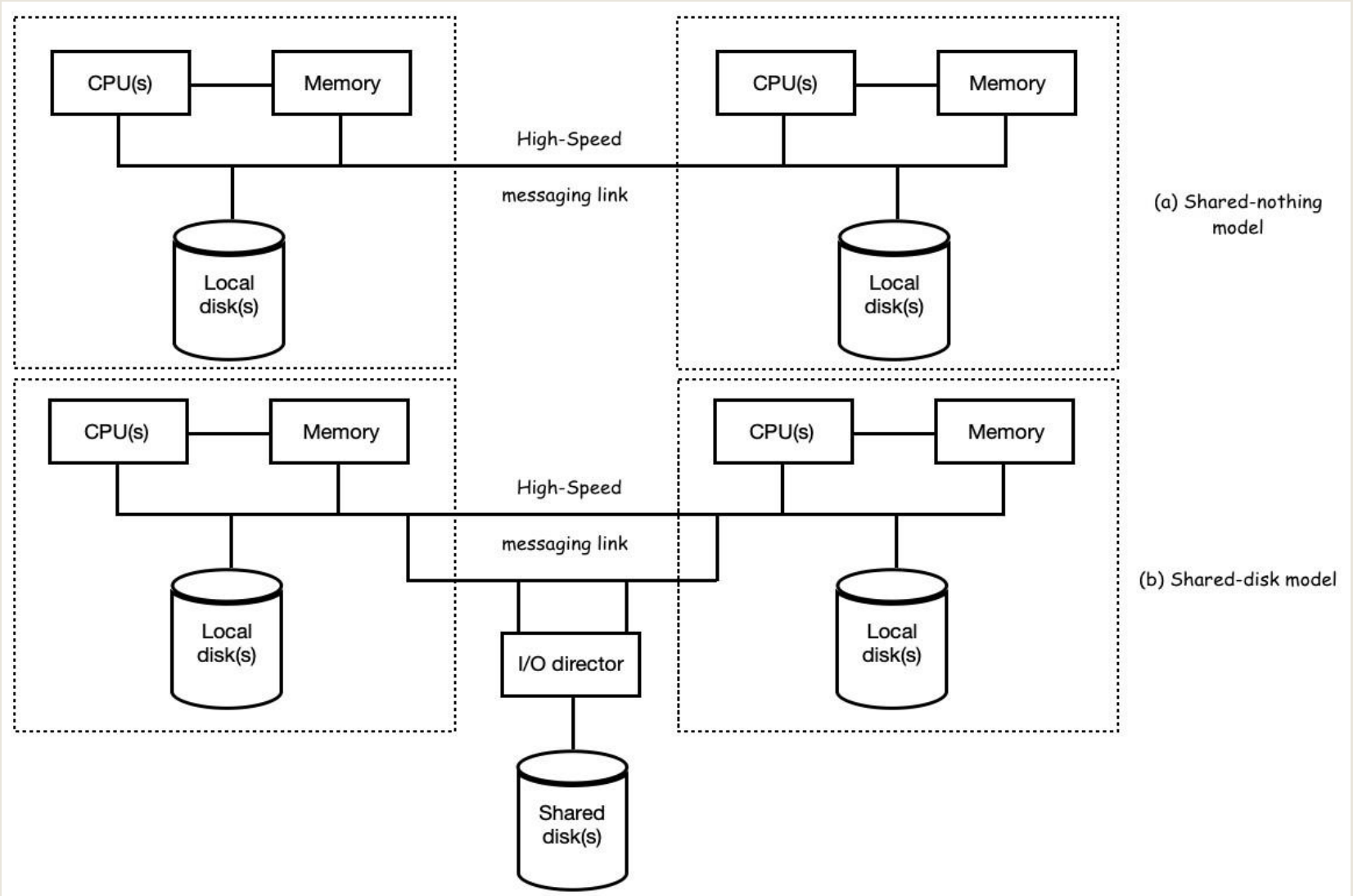
- Whenever a process finishes, the node checks whether it is under loaded. If so, it randomly probes a node for work.
  - If the probed node is overloaded, it migrates a process to the probing node.
  - Otherwise, the probing node repeatedly probes for work until reaches an upper limit N.
- It does not degrade performance in heavily loaded situation.



The Cluster Systems

- A common architecture of Multi Computer systems.
- A computer cluster consists of a set of tightly connected computers that work together such that it appears as a single system to users and applications
  - These computers are commonly connected through a fast LAN.
  - Each computer has its own CPU, private main memory, I/O facilities, and has its own operating system.
    - Since the processors on one computer cannot directly access the memory on the others, "message passing" is the most common way for inter process communications in a cluster.
- In general, there are Shared-nothing and Shared-disk models.

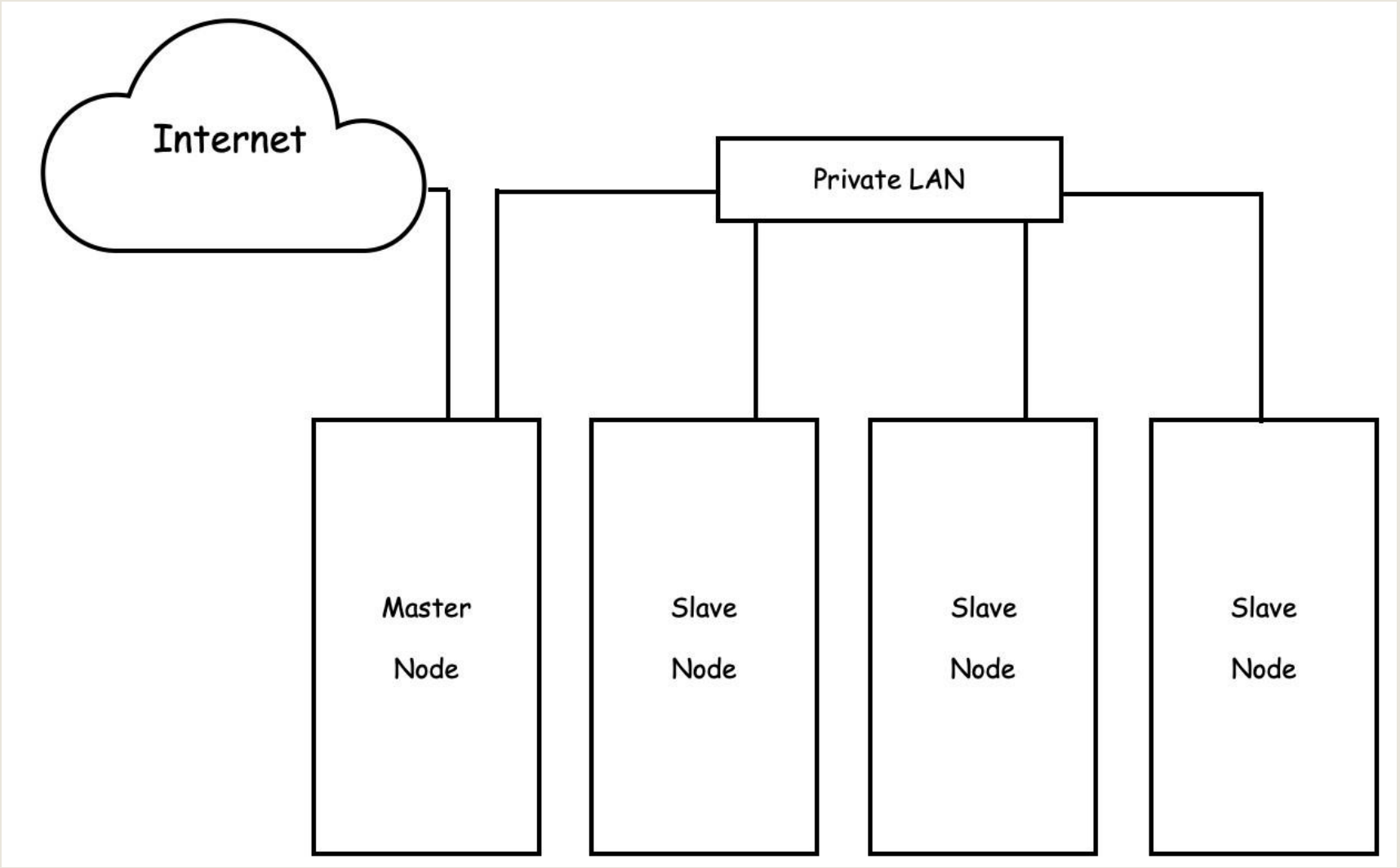
Two Types of Computer Clusters



The Beowulf Cluster

- Donald Becker of NASA assembled this cluster in 1994.
  - It is a set of off-shell computers connected through a private LAN.
  - It consists of two types of computers.
    - A master computer is used to interact with users and outside world.
    - A group of slave computers.
  - A task is submitted to the master of a Beowulf cluster.
    - The master computer then breaks the task into small subtasks.
    - It sends each subtask to a separate slave to compute.
    - As nodes finish their subtasks, the master continually sends remaining subtasks to them until the entire task has been computed.
- MPI (Message Passing Interface) and PVM (Parallel Virtual Machine) are the commonly used mechanisms for IPC in a Beowulf cluster.

The Topology of a Beowulf Cluster



More on Beowulf Cluster, MPI, PVM

- Thomas Lawrence Sterling, "Beowulf Cluster Computing with Linux," MIT 2002
- Richard S. Morrison, "Cluster Computing: Architectures, Operating Systems, Parallel Processing & Programming Languages," http://electro.fisica.unlp.edu.ar/arq/downloads/Papers/Clusters/Linux%20Cluster%20Computing%20-%20Apr%202003.pdf
- Official site of MPI, http://www.mpi-forum.org/
- Official site of MPICH, https://www.mpich.org/
- Official site of PVM, http://www.csm.ornl.gov/pvm/
- Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek and Vaidyalingam S. Sunderam,"PVM: A Users' Guide and Tutorial for Networked Parallel Computing," http://www.netlib.org/pvm3/book/pvm-book.html
- Sanath Kumar, "Building a Beowulf Cluster in just 13 steps," https://www.linux.com/community/blogs/133-general-linux/9401
- Mike Perry, "Building Linux Beowulf Clusters," http://fscked.org/writings/clusters/cluster.html