

**Lecture 18:**

# **Interconnection Networks**

---

**Parallel Computer Architecture and Programming  
CMU 15-418/15-618, Spring 2015**

**Credit: many of these slides were created by Michael Papamichael  
This lecture is partially based on Prof. Mutlu's 18-742 lecture slides**

# Tunes

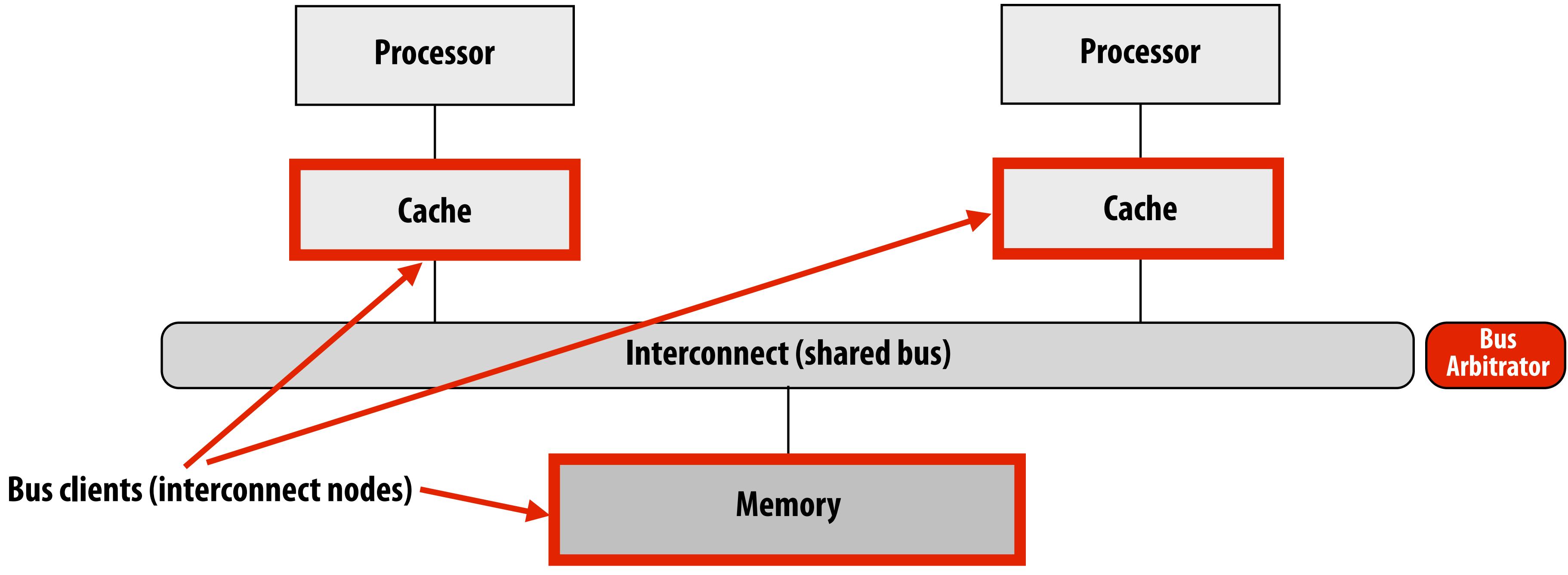
## Kaiser Chiefs

**Never Miss A Beat  
(Off With Their Heads)**

*“Not many people know this about me, but I am a big fan of circuit switching.”*

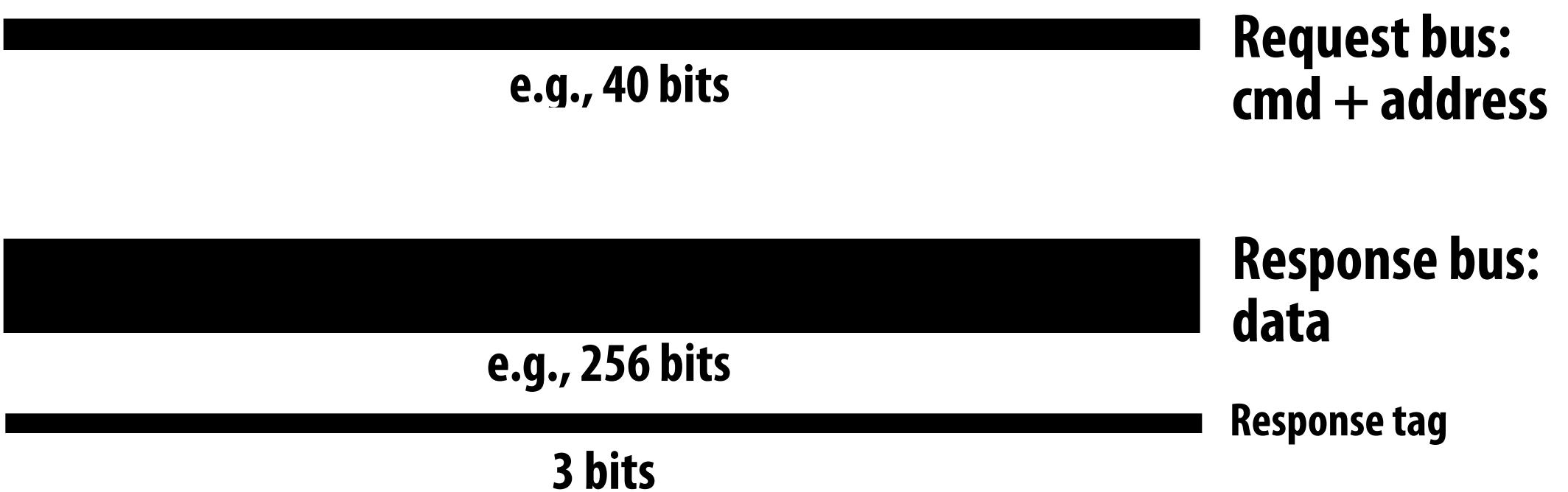
*- Ricky Wilson*

# Basic system design from previous lectures

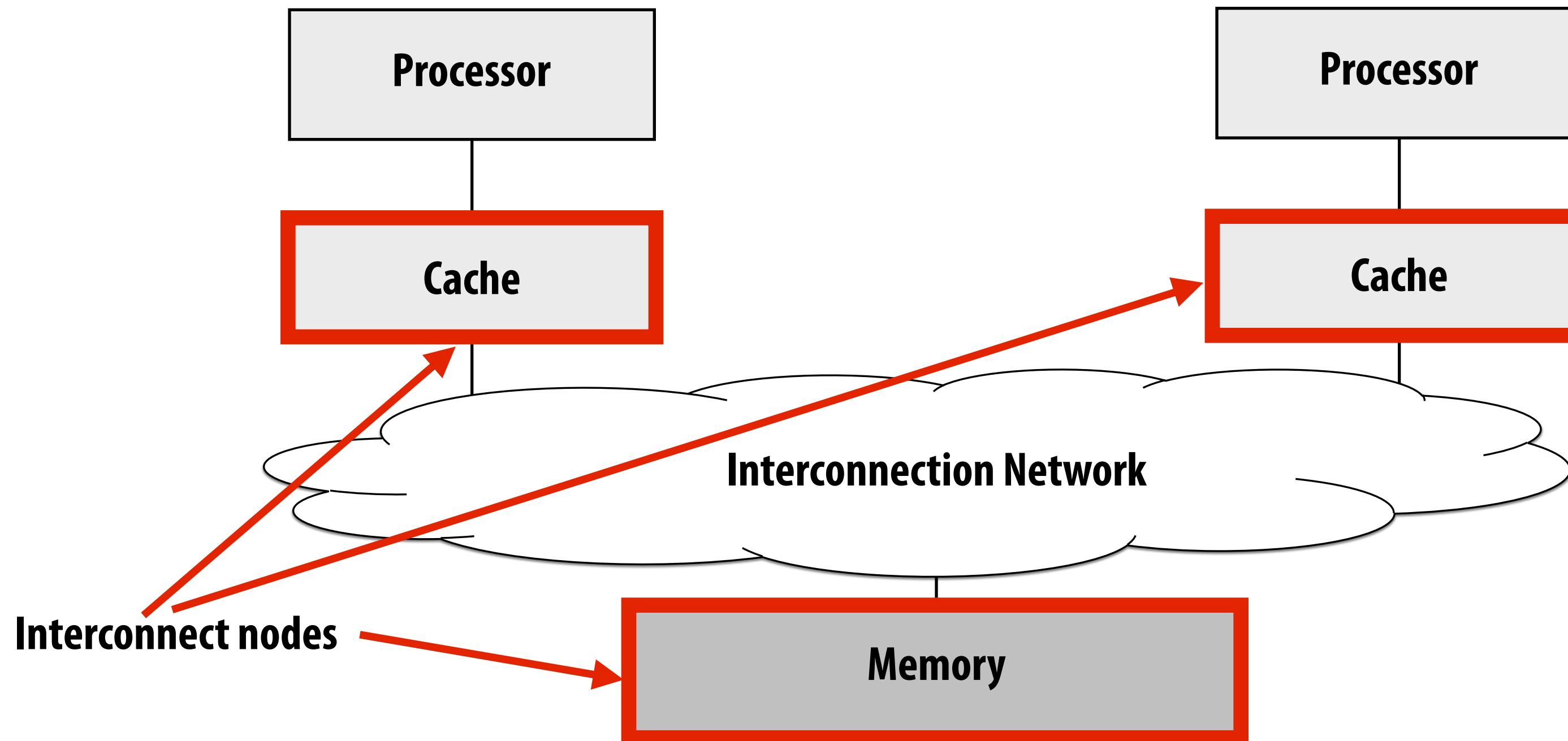


**Bus interconnect:**

**All nodes connected by a shared set of wires**



# Today: modern interconnect designs



**Today's topics: the basic ideas of building a high-performance interconnection network in a parallel processor.  
(think: "a network-on-a-chip")**

# What are interconnection networks used for?

- To connect:

- Processor cores with other cores
- Processors and memories
- Processor cores and caches
- Caches and caches
- I/O devices

# Why is the design of the interconnection network important?

## ■ System scalability

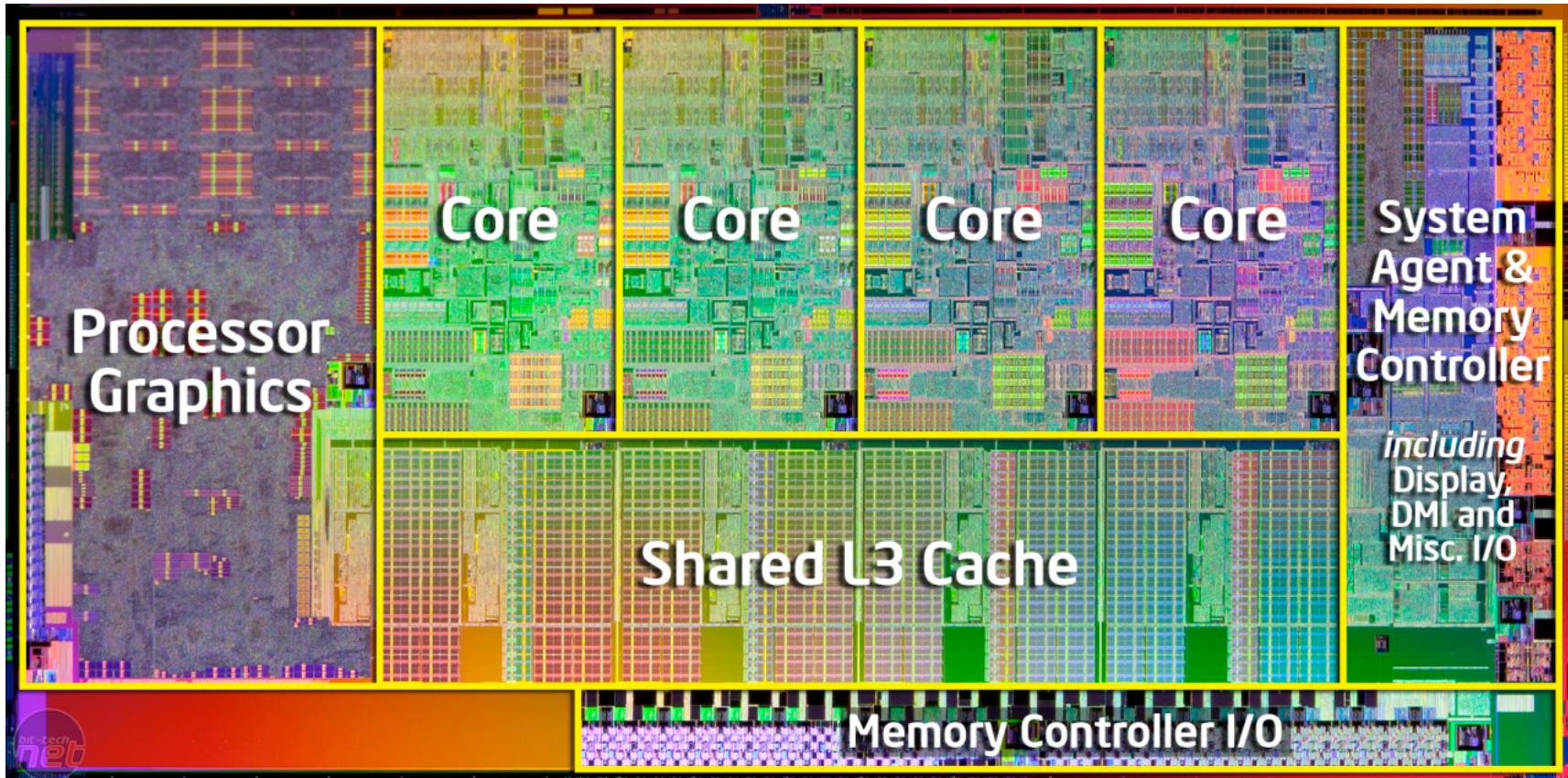
- How large of a system can be built?
- How easy is it to add more nodes (e.g., cores)

## ■ System performance and energy efficiency

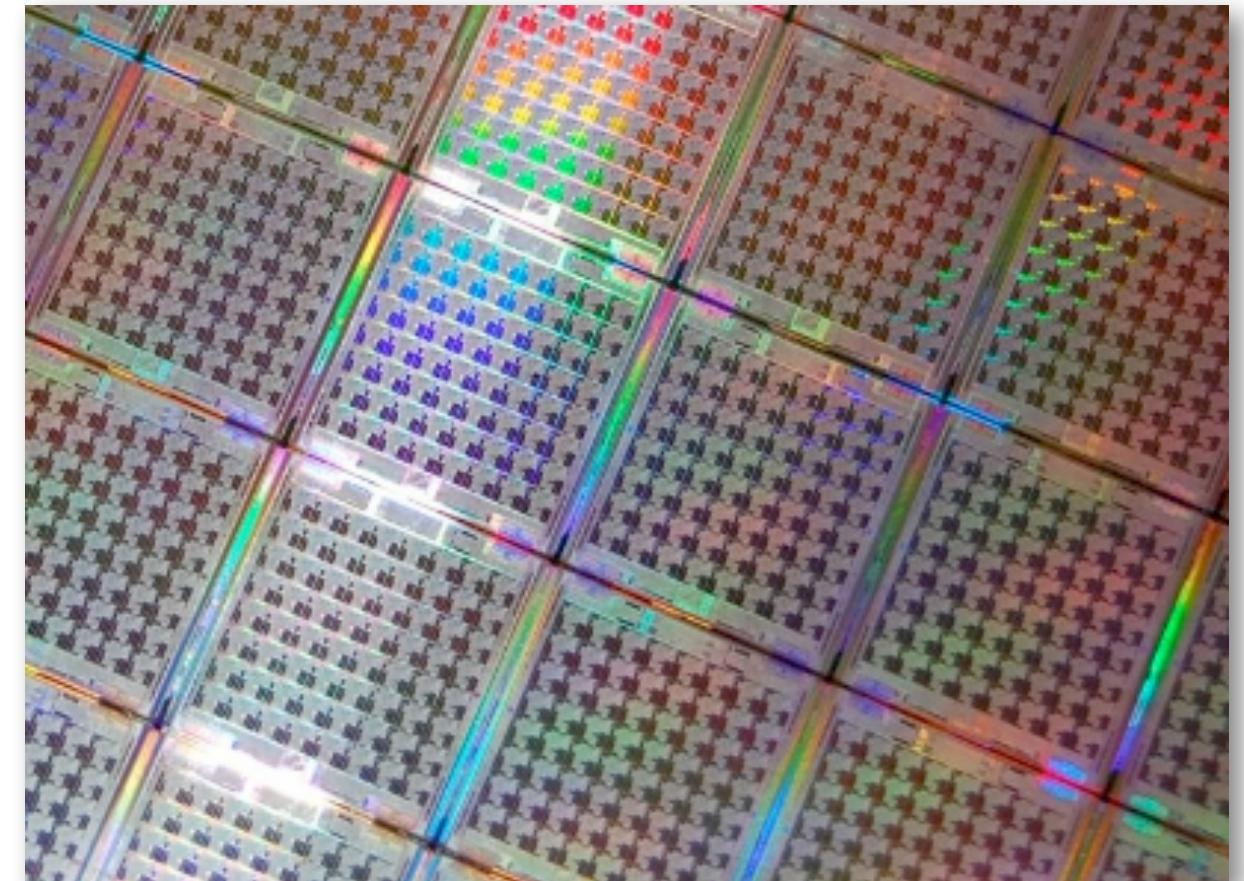
- How fast can cores, caches, memory communicate
- How long is latency to memory?
- How much energy is spent on communication?

# Increasing core counts

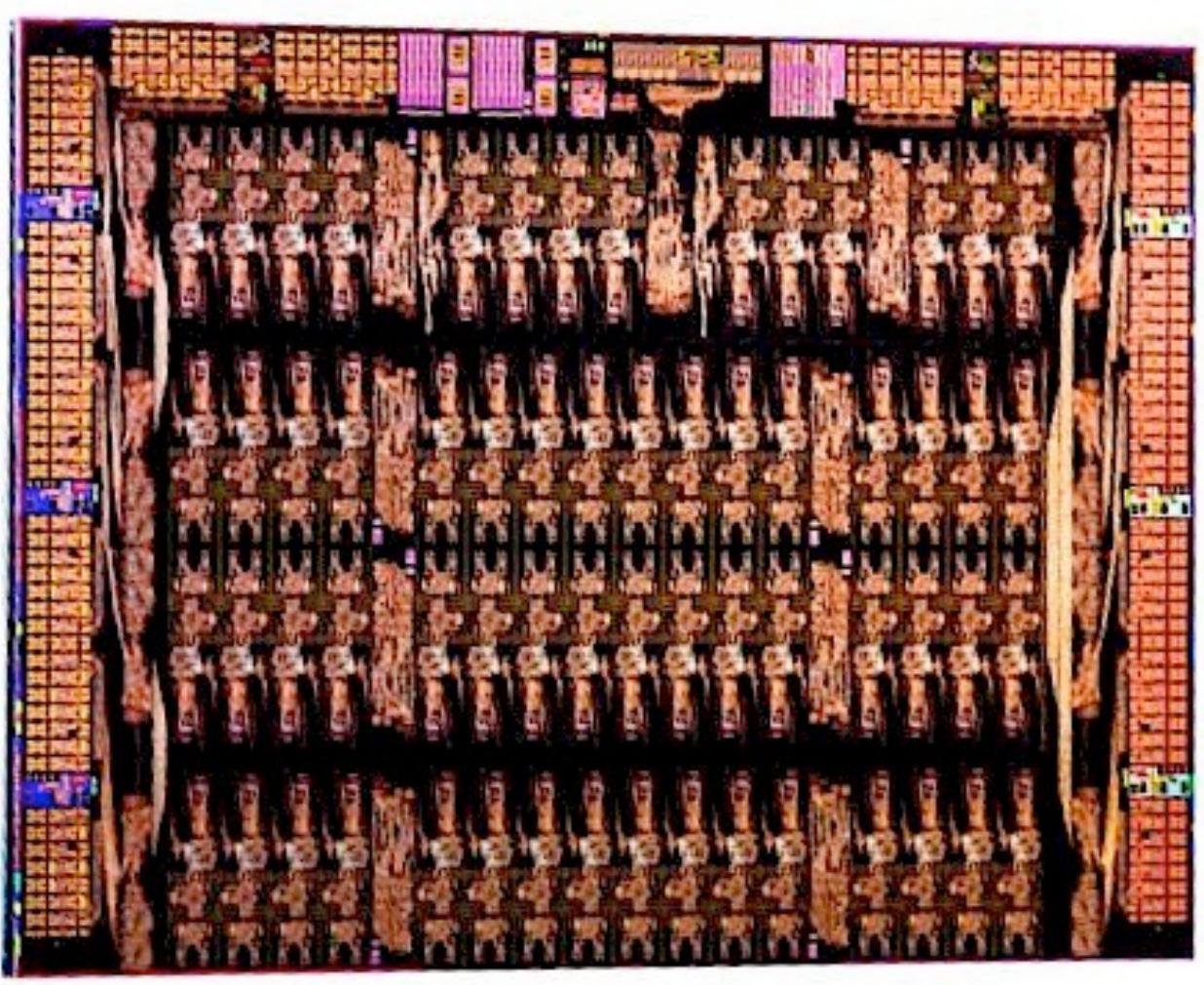
Scalability of on-chip interconnection network becomes increasingly important



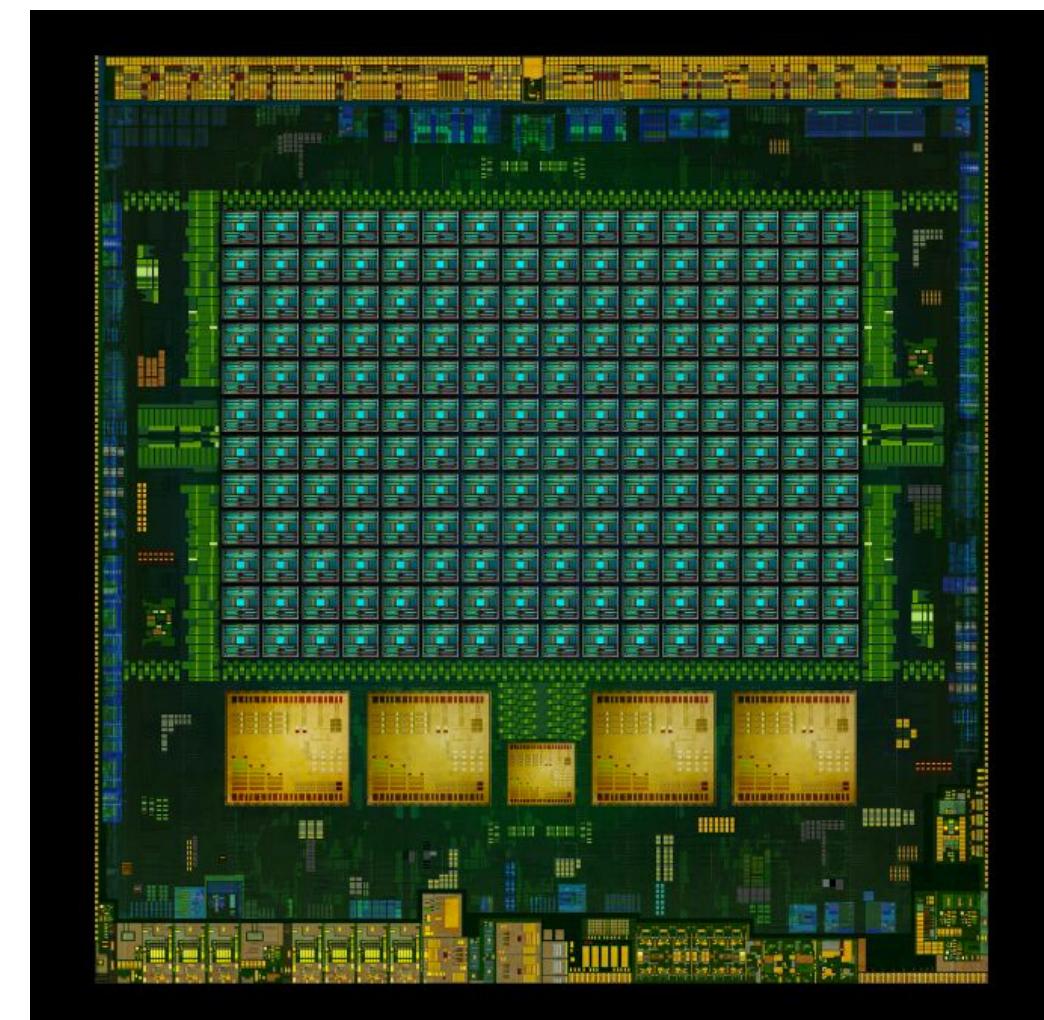
Intel core i7 (4-CPU cores, + GPU)



Tilera GX 64-core chip



Intel Xeon Phi (60-core x86)

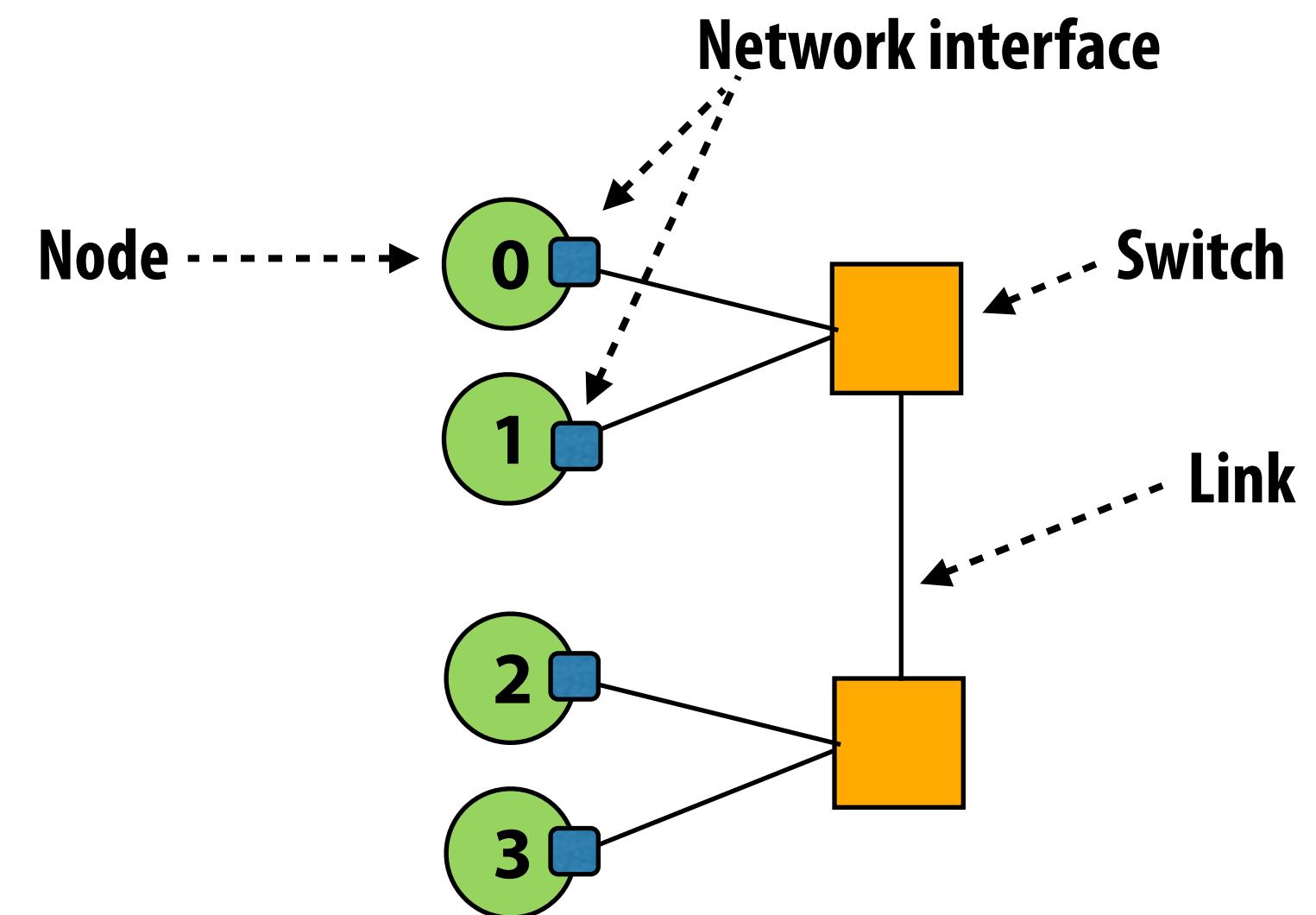


Tegra K1: 4 + 1 ARM cores + GPU cores

# **Interconnect terminology**

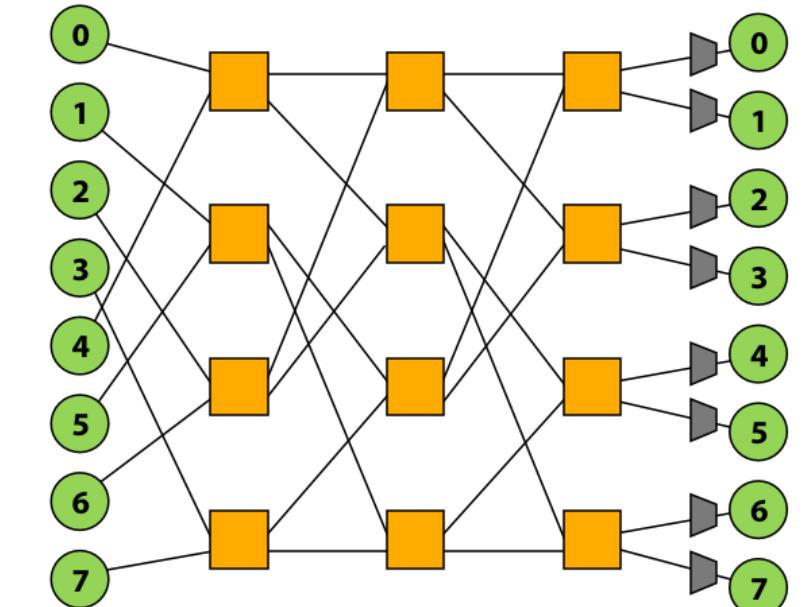
# Terminology

- **Network node:** a network endpoint connected to a router/switch
  - Examples: processor caches, the memory controller
- **Network interface:**
  - Connects nodes to the network
- **Switch/router:**
  - Connects a fixed number of inputs to a fixed number of outputs
- **Link:**
  - A bundle of wires carrying a signal



# Design issues

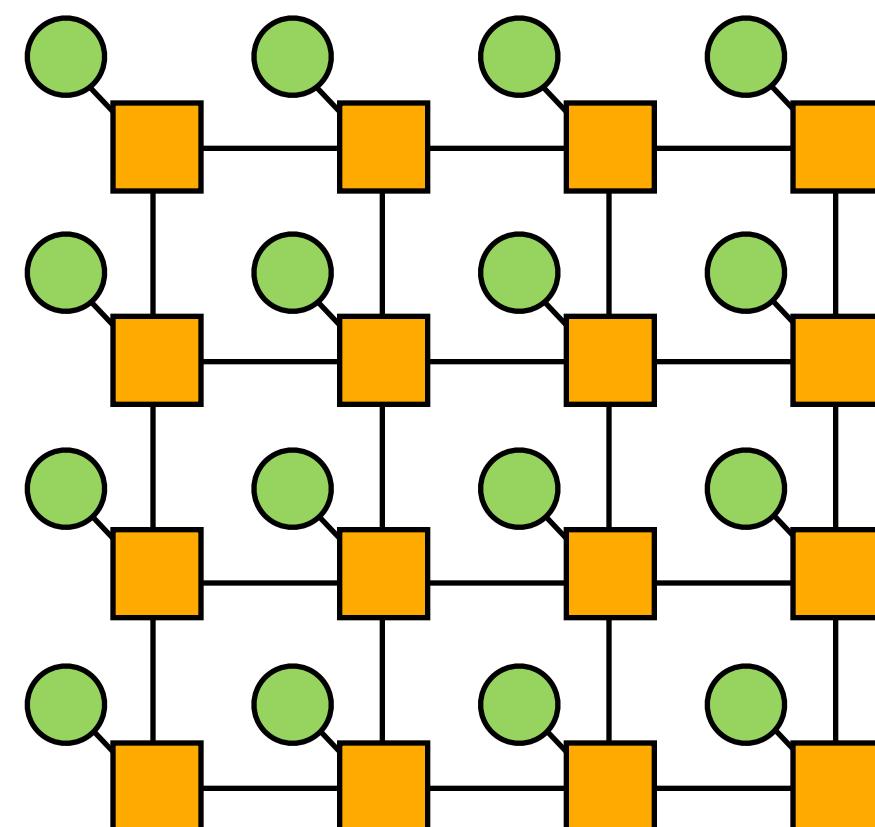
- **Topology: how switches are connected via links**
  - Affects routing, throughput, latency, complexity/cost of implementation
- **Routing: how a message gets from its source to its destination in the network**
  - Can be static (messages take a predetermined path) or adaptive based on load
- **Buffering and flow control**
  - What data is stored in the network? packets, partial packets? etc.
  - How does the network manage buffer space?



# Properties of interconnect topology

- Regular or irregular topology
  - Regular is topology is a regular graph (e.g., ring, mesh)
- Routing distance
  - Number of links (“hops”) along a route
- Diameter: the maximum routing distance
- Average distance: average routing distance over all valid routes

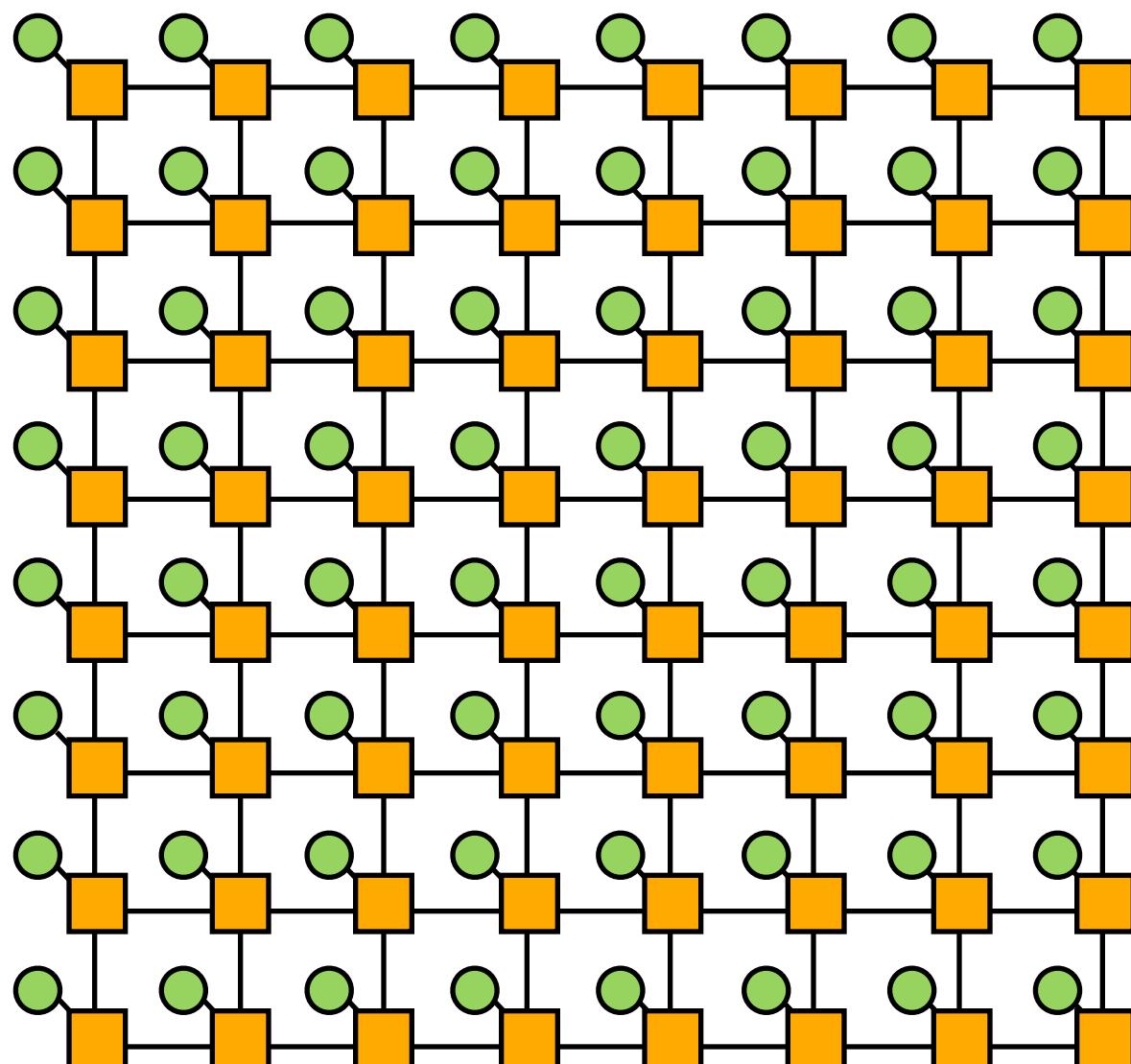
Example:  
diameter = 6



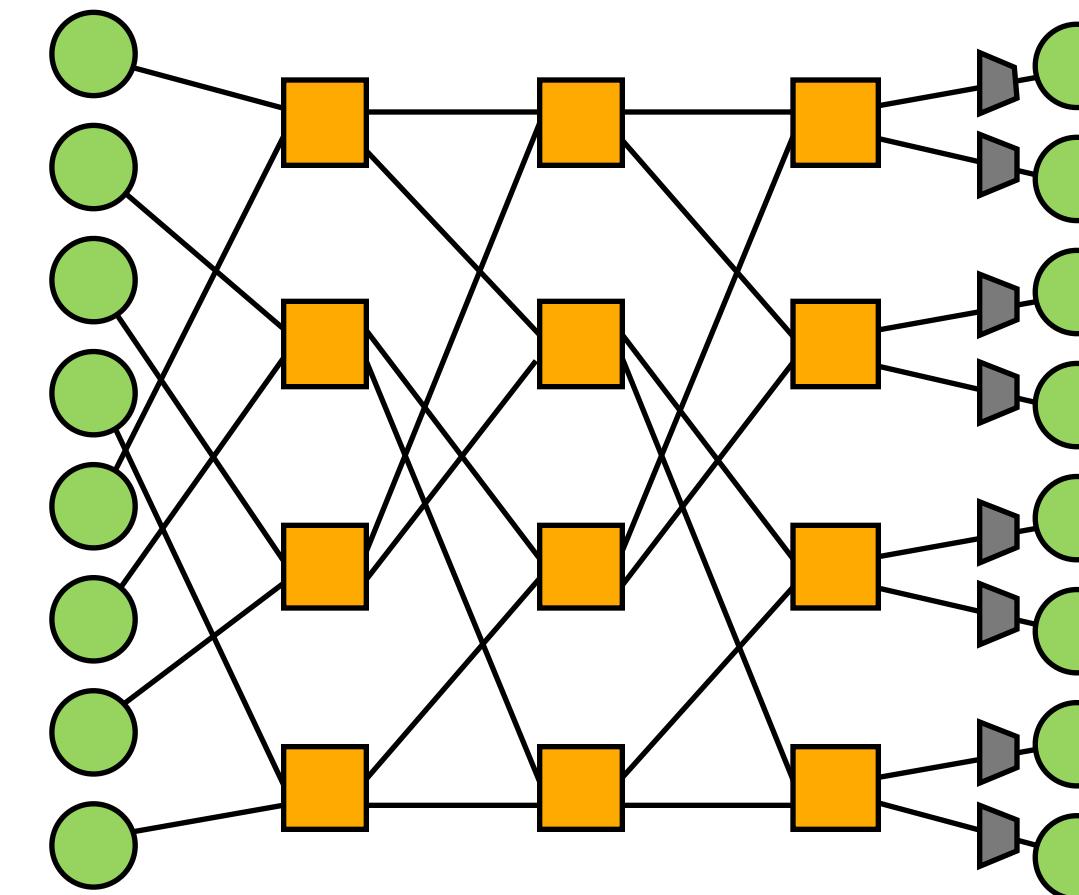
# Properties of interconnect topology

## ■ Direct vs. indirect networks

- **Direct network: endpoints sit “inside” the network**
- **e.g., mesh is direct network: every node is both an endpoint and a switch**



**Direct network**



**Indirect network**

# Properties of an interconnect topology

## ■ Bisection bandwidth:

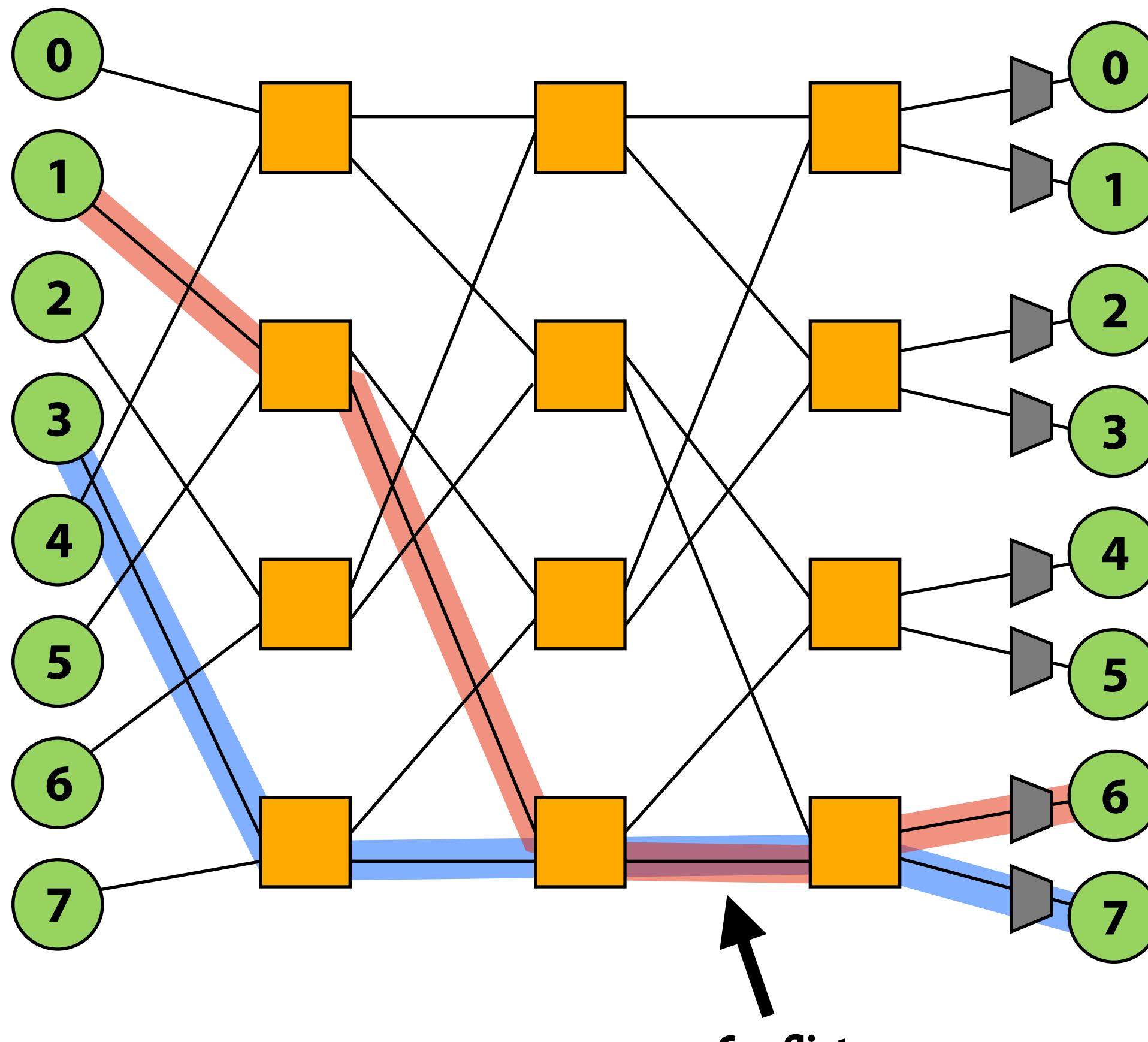
- Common metric of performance for recursive topologies
- Cut network in half, sum bandwidth of all severed links
- Can be misleading: does not account for switch and routing efficiencies

## ■ Blocking vs. non-blocking:

- If connecting any pairing of nodes is possible, network is non-blocking (otherwise, it's blocking)

# Blocking vs. non-blocking example

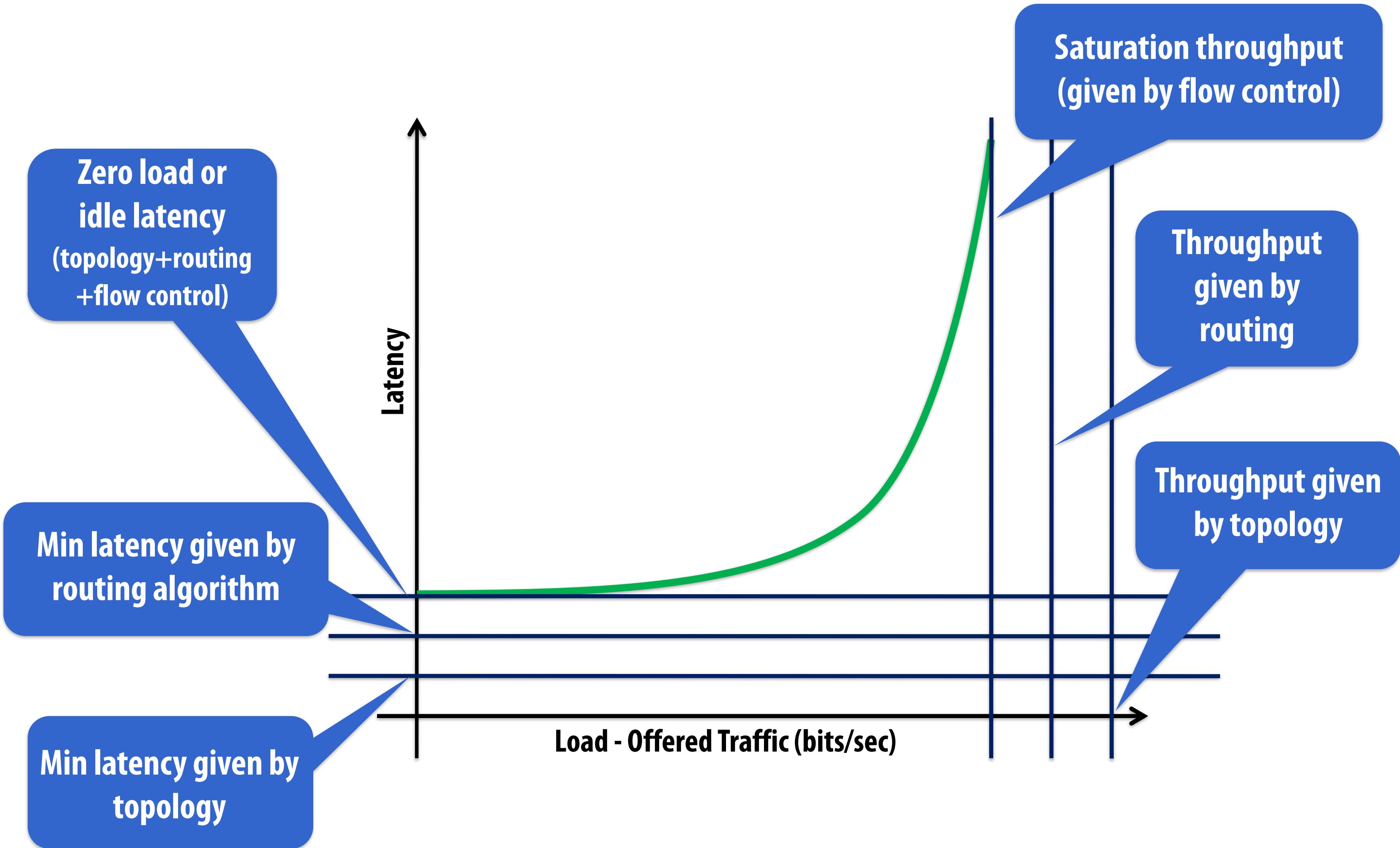
- Is this network blocking or non-blocking?
  - Consider simultaneous messages from 0-to-1 and 3-to-7.
  - Consider simultaneous messages from 1-to-6 and 3-to-7. Blocking!!!



Note: in this network illustration, each node is drawn twice for clarity (at left and at right)

# Load-latency behavior of network

General rule: latency increases with load (throughput)



# **Interconnect topologies**

# Many possible network topologies

- Bus
- Crossbar
- Ring
- Tree
- Omega
- Hypercube
- Mesh
- Torus
- Butterfly
- ...

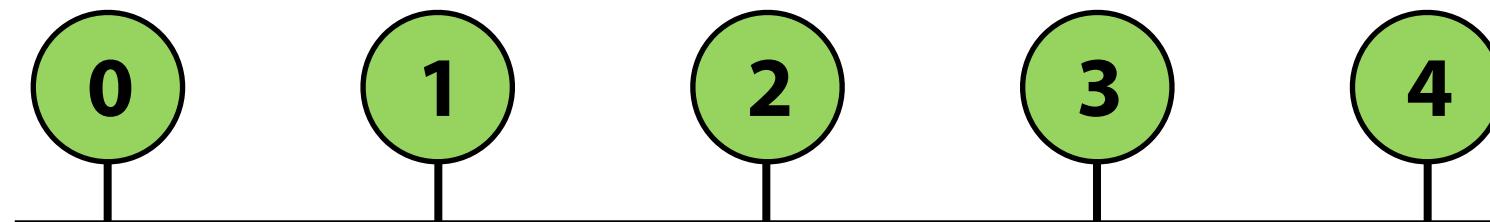
# A bus interconnect

## ■ Good:

- Simple design
- Cost effective for a small number of nodes
- Easy to implement coherence (via snooping)

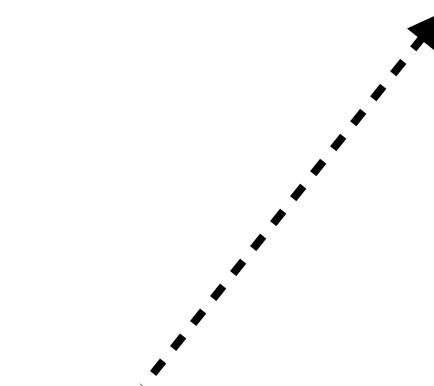
## ■ Bad:

- Contention: all nodes contend for shared bus
- Limited bandwidth: all nodes communicate over same wires
- High electrical load = low frequency, high power

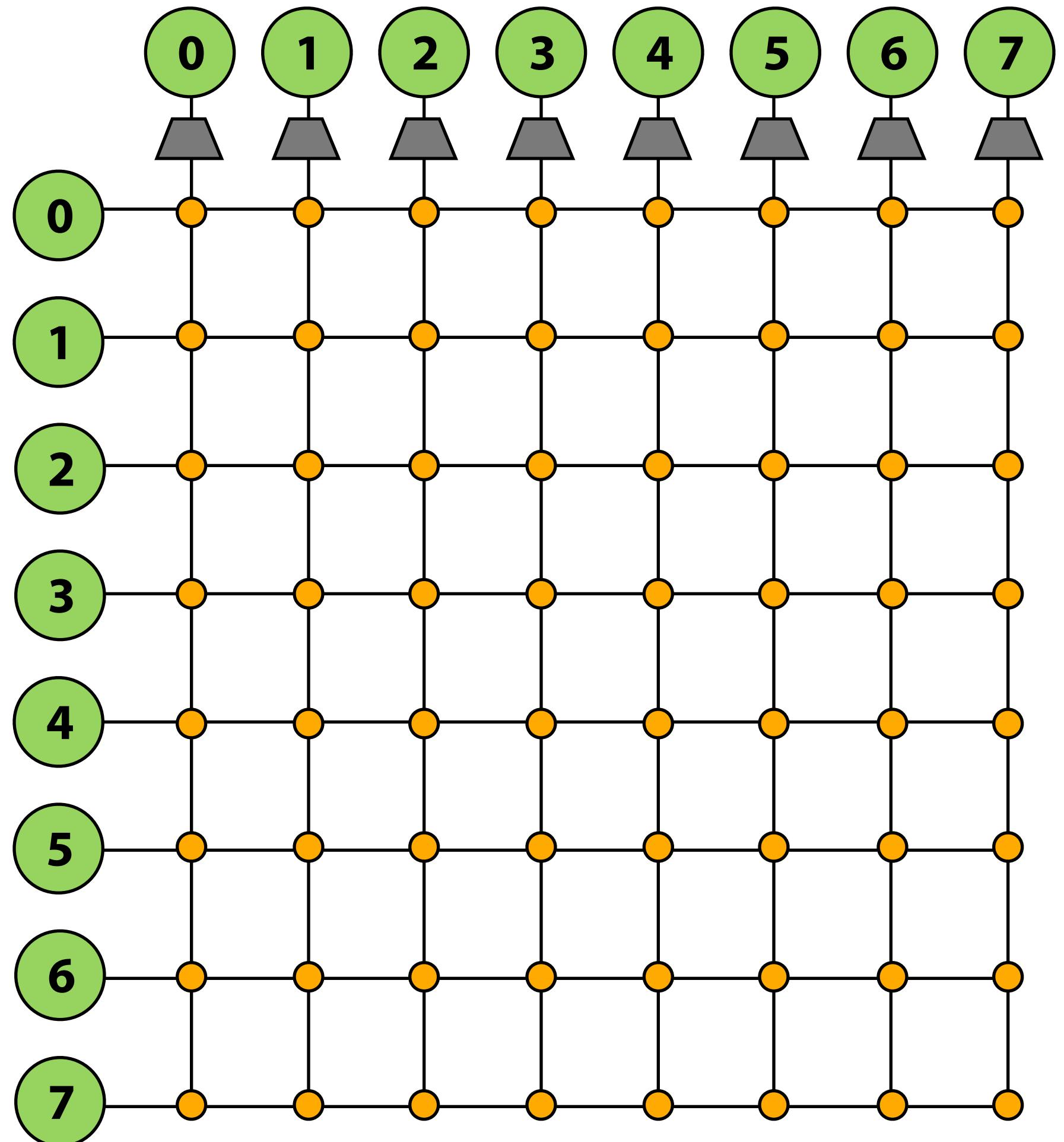


# Crossbar interconnect

- Every node is connected to every other node (non-blocking, indirect)
- Good:
  - **$O(1)$  latency and high bandwidth**
- Bad:
  - **Not scalable:  $O(N^2)$  switches**
  - **High cost**
  - **Difficult to arbitrate at scale**

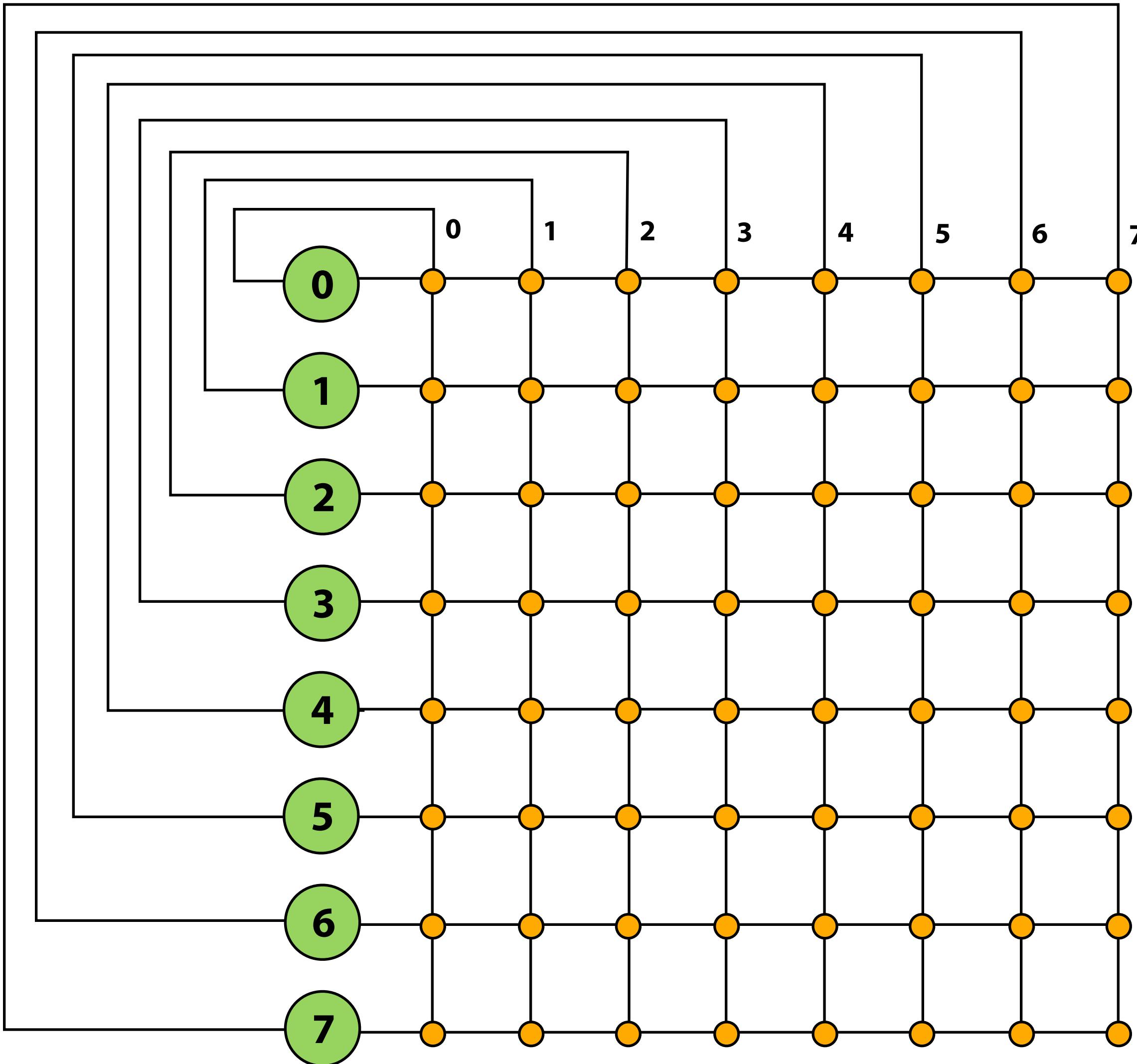


Crossbar scheduling algorithms / efficient hardware implementations are still active research areas.

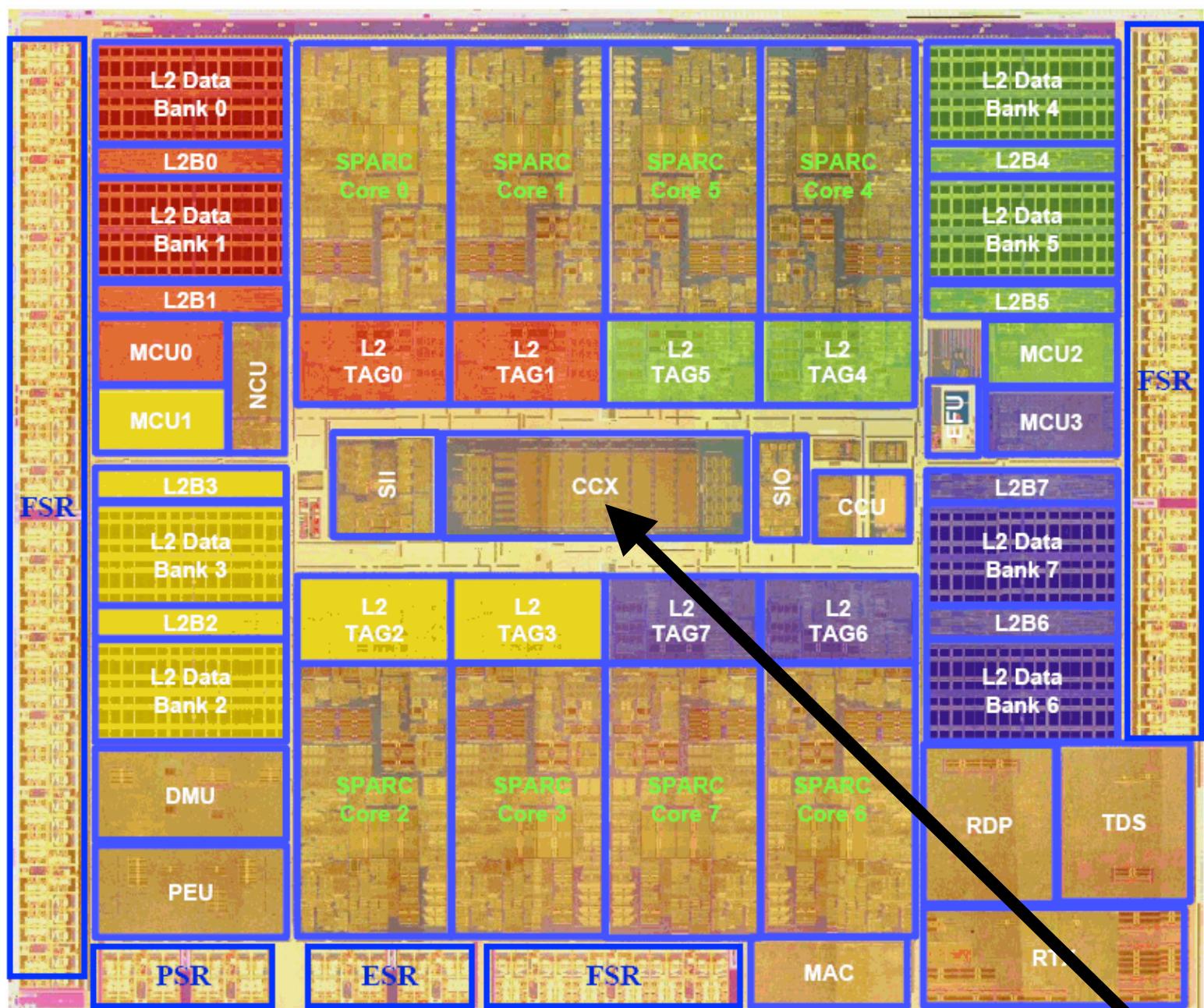


# Crossbar interconnect

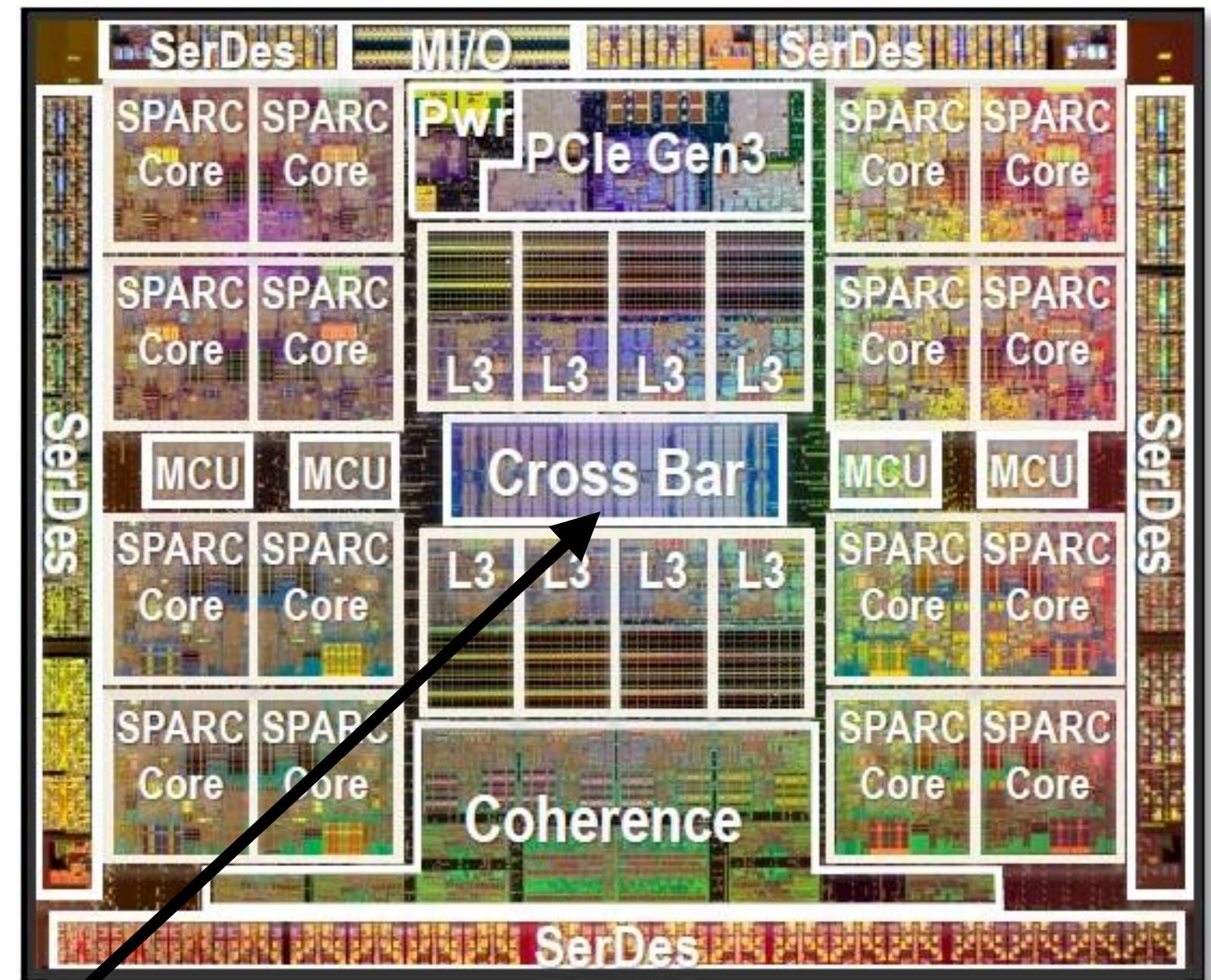
(Here is a more verbose illustration than that on previous slide)



# Crossbars are using in Sun/Oracle processors



Sun SPARC T2 (8 cores, 8 L2 cache banks)

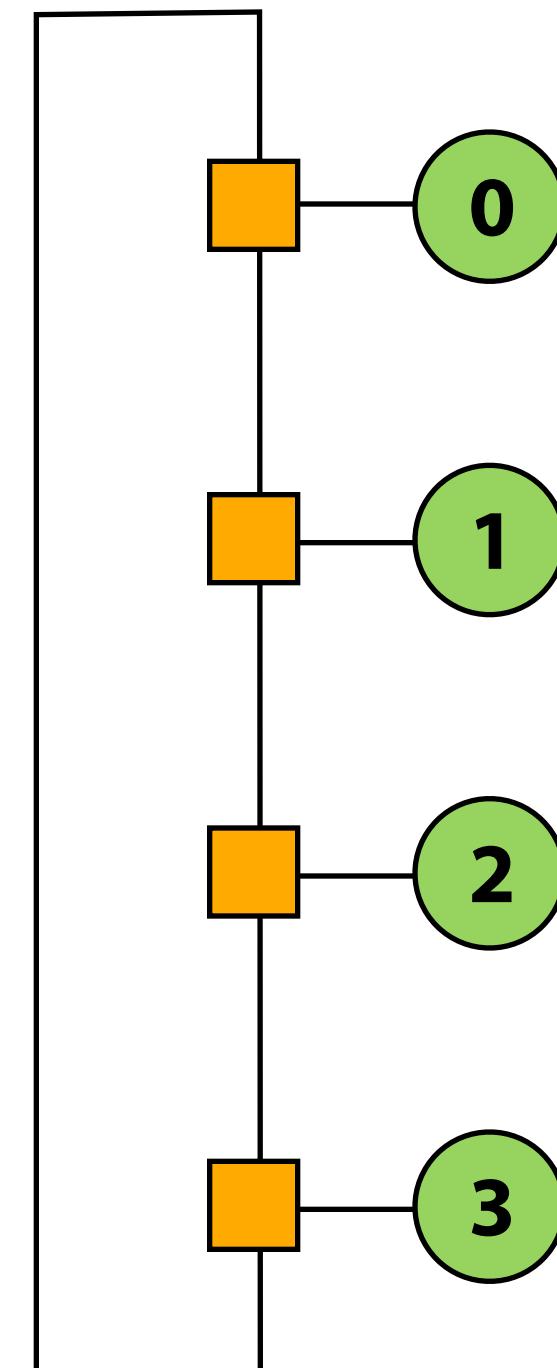


Oracle SPARC T5 (16 cores, 8 L3 cache banks)

Note that crossbar (CCX) occupies about the same chip area as a core

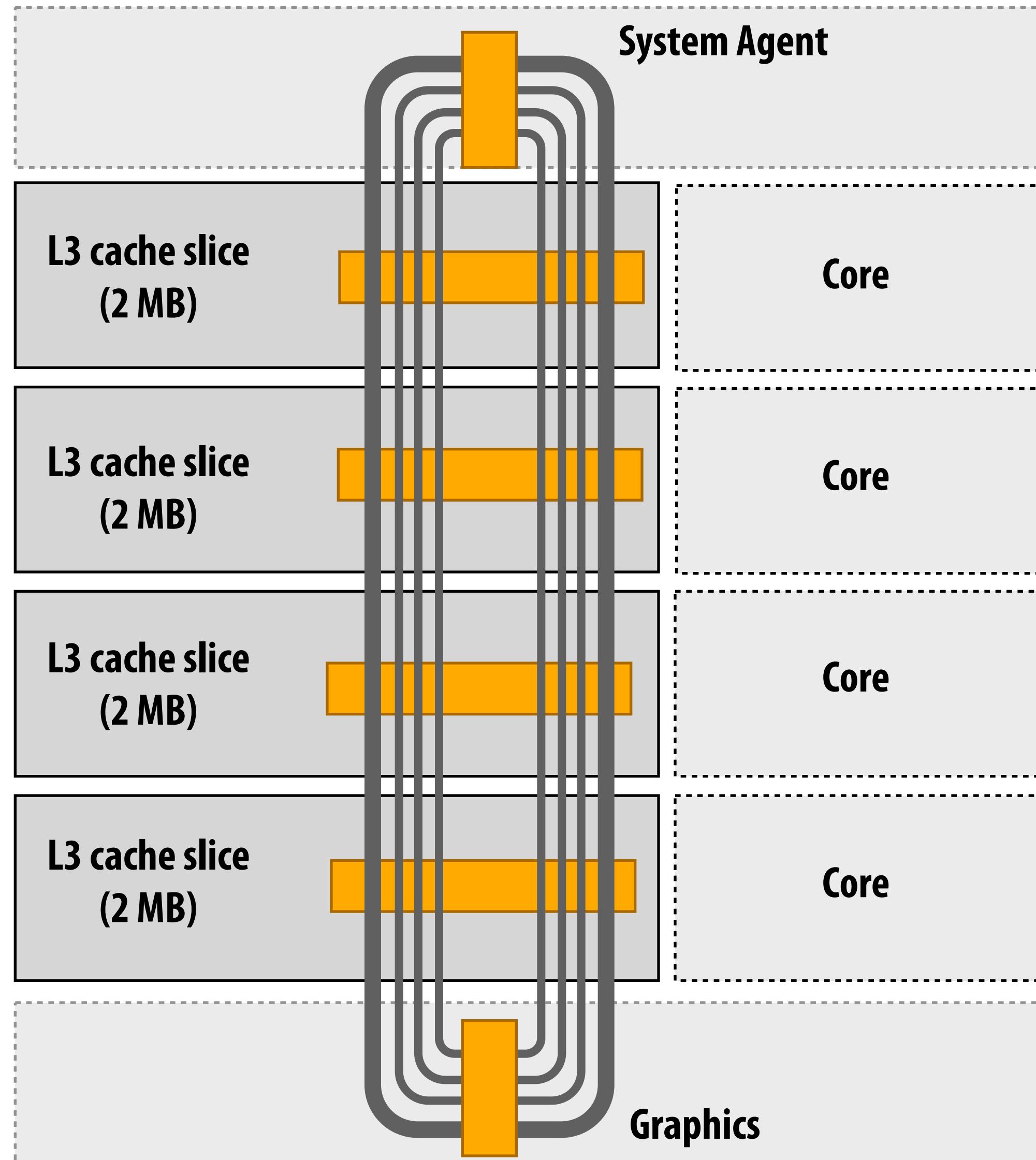
# Ring

- **Good:**
  - **Simple**
  - **Cheap:  $O(N)$  cost**
- **Bad:**
  - **High latency:  $O(N)$**
  - **Bisection bandwidth remains constant as nodes are added (scalability issue)**
- **Used in recent Intel architectures**
  - **Core i7, Xeon Phi**
- **Also used in IBM CELL Broadband Engine (9 cores)**



# Intel's ring interconnect

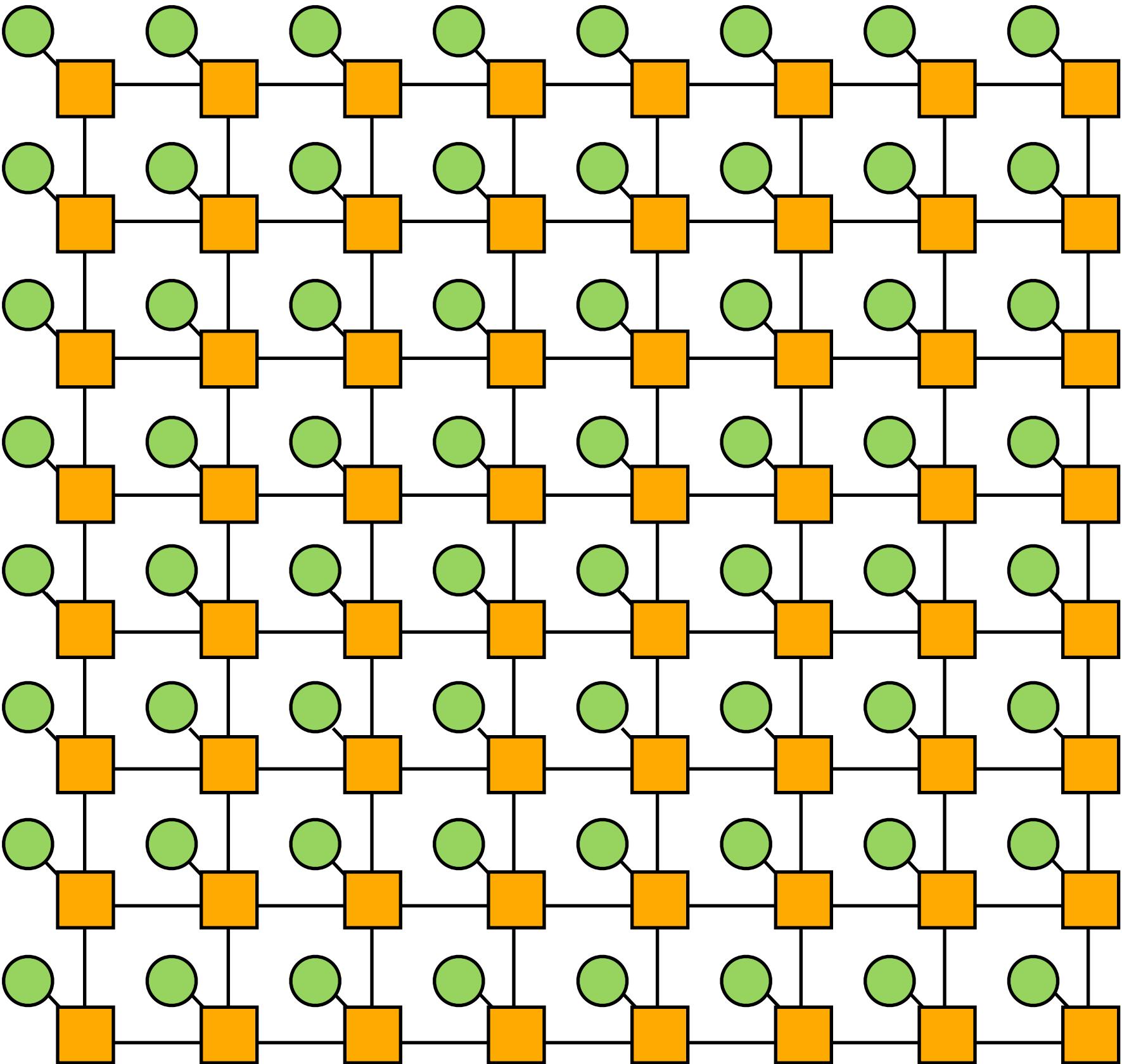
Introduced in Sandy Bridge microarchitecture



- **Four rings**
  - request
  - snoop
  - ack
  - data (32 bytes)
- **Six interconnect nodes: four “slices” of L3 cache + system agent + graphics**
- **Each bank of L3 connected to ring bus twice**
- **Theoretical peak BW from cores to L3 at 3.4 GHz is approx. 435 GB/sec**
  - When each core is accessing its local slice

# Mesh

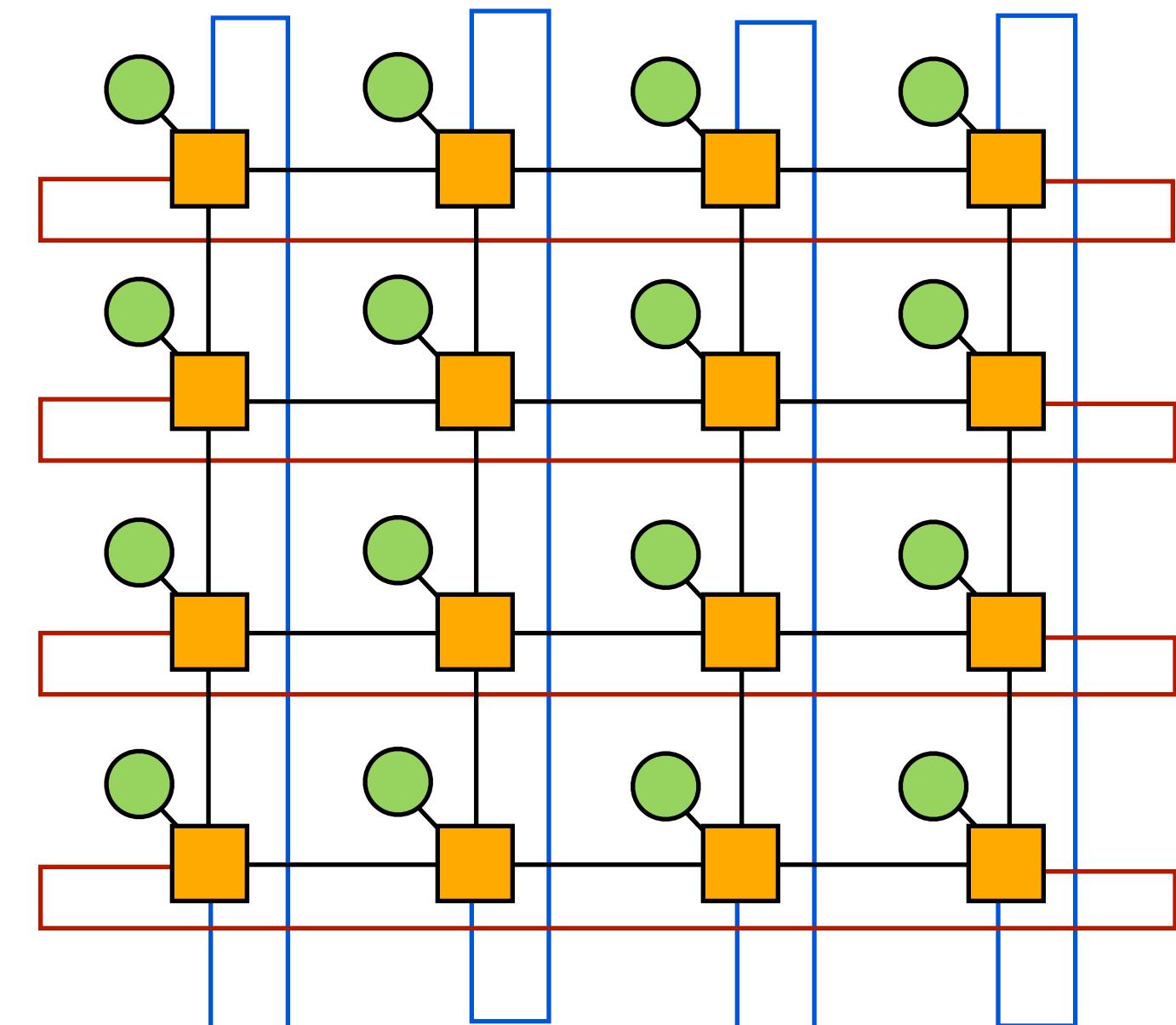
- Direct network
- Echoes locality in grid-based applications
- $O(N)$  cost
- Average latency:  $O(\sqrt{N})$
- Easy to lay out on chip: fixed-length links
- Path diversity: many ways for message to travel from one node to another
- Used by:
  - Tilera processors
  - Prototype Intel chips



2D Mesh

# Torus

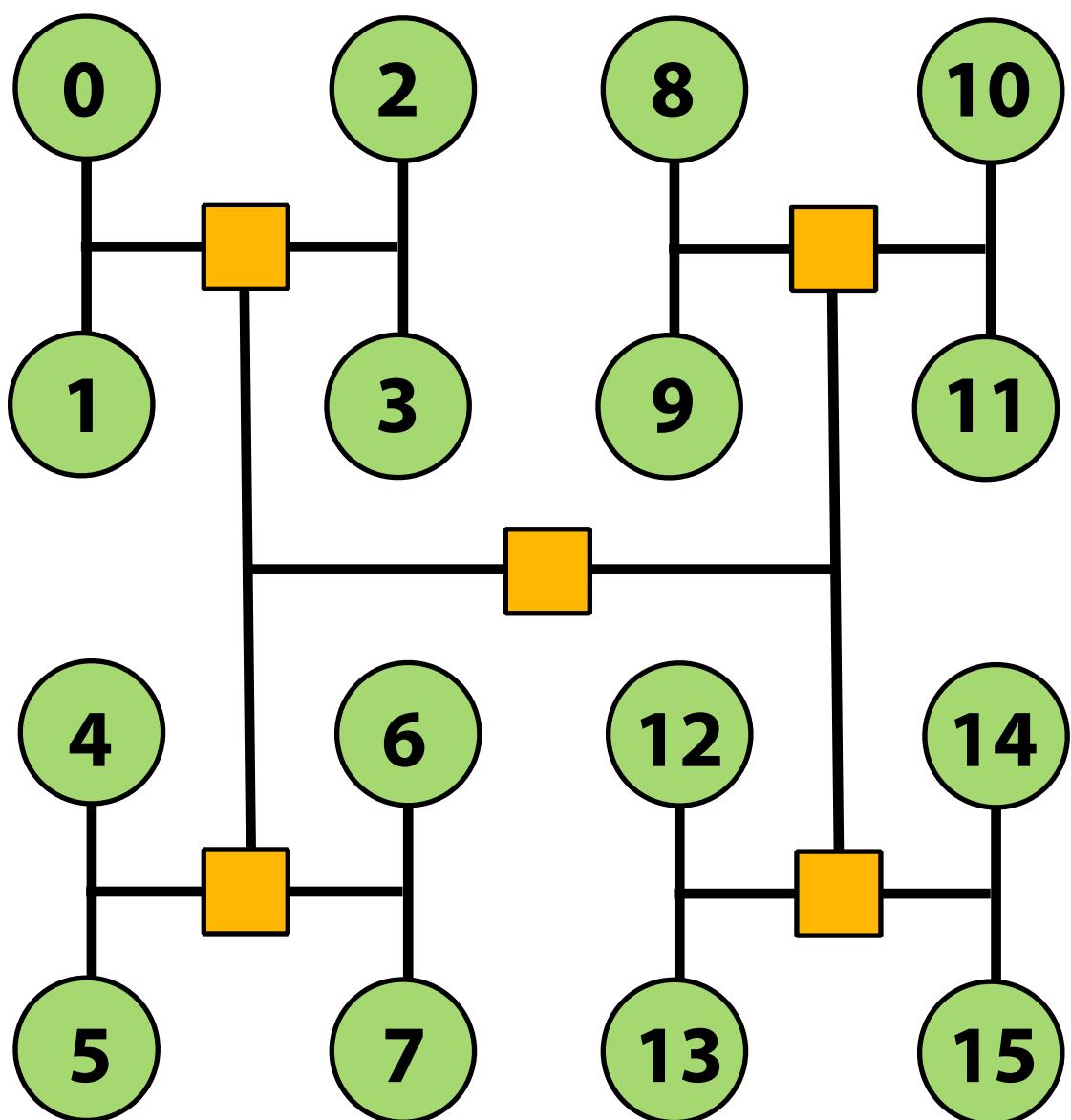
- Characteristics of mesh topology are different based on whether node is near edge or middle of network (torus topology introduces new links to avoid this problem)
- Still  $O(N)$  cost, but higher cost than 2D grid
- Higher path diversity and bisection BW than mesh
- Higher complexity
  - Difficult to layout on chip
  - Unequal link lengths



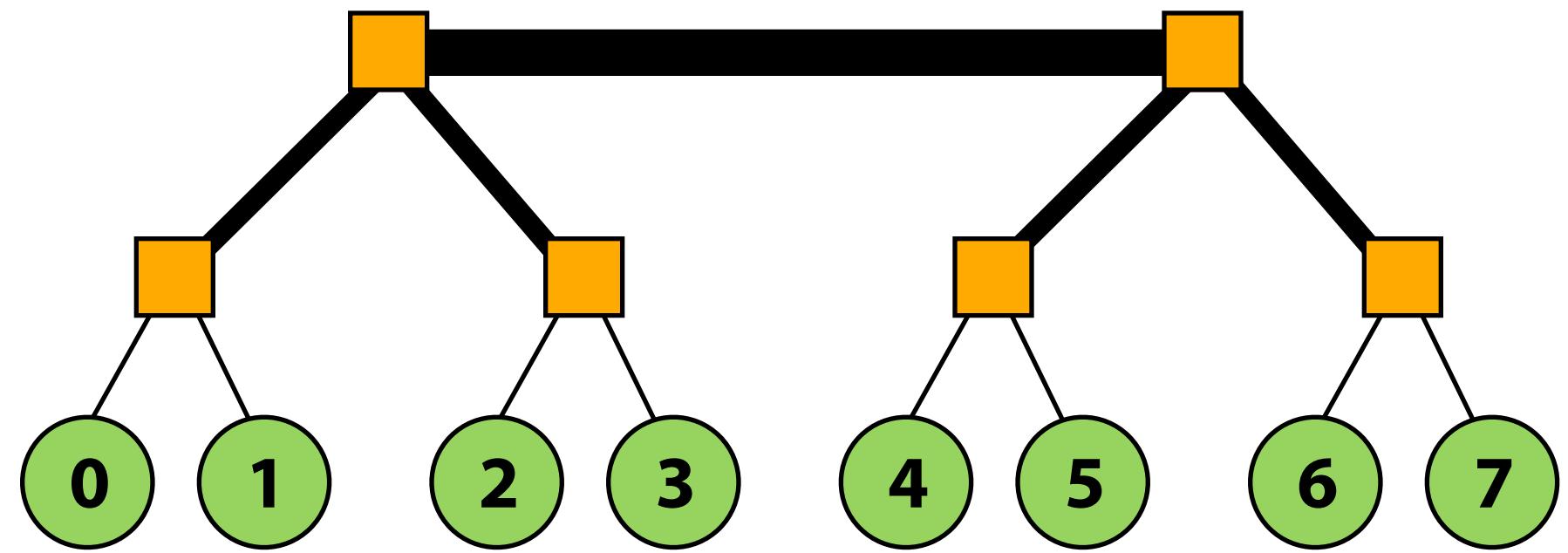
2D Torus

# Trees

- Planar, hierarchical topology
- Like mesh/torus, good when traffic has locality
- Latency:  $O(\lg N)$
- Use “fat trees” to alleviate root bandwidth problem (higher bandwidth links near root)

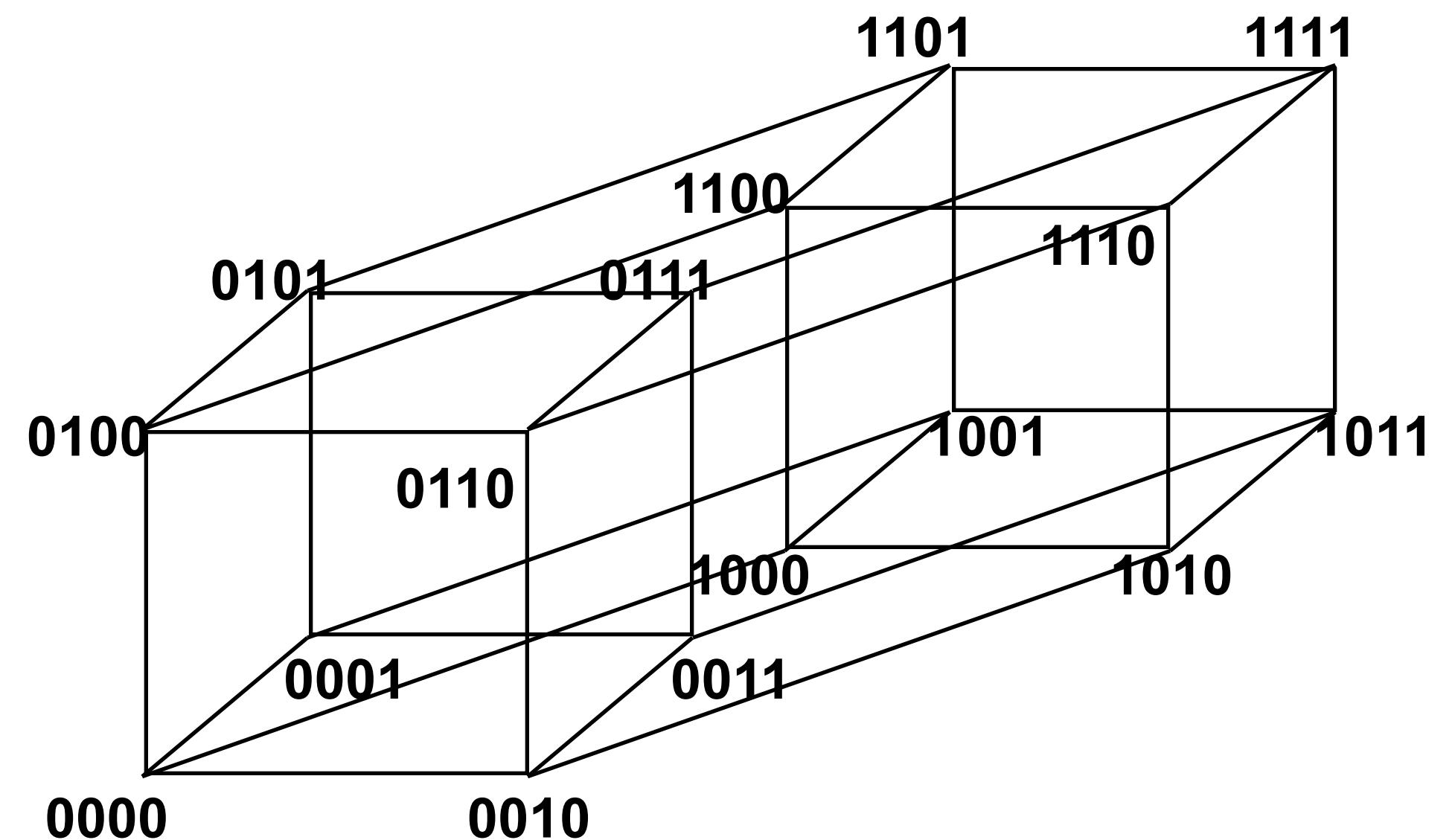
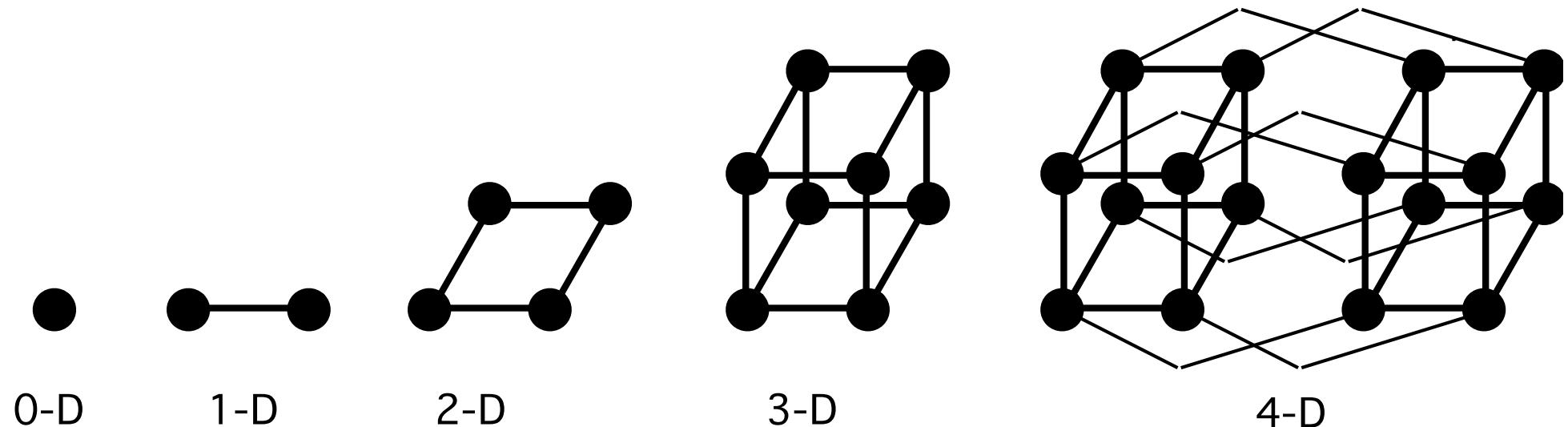


H-Tree



Fat Tree

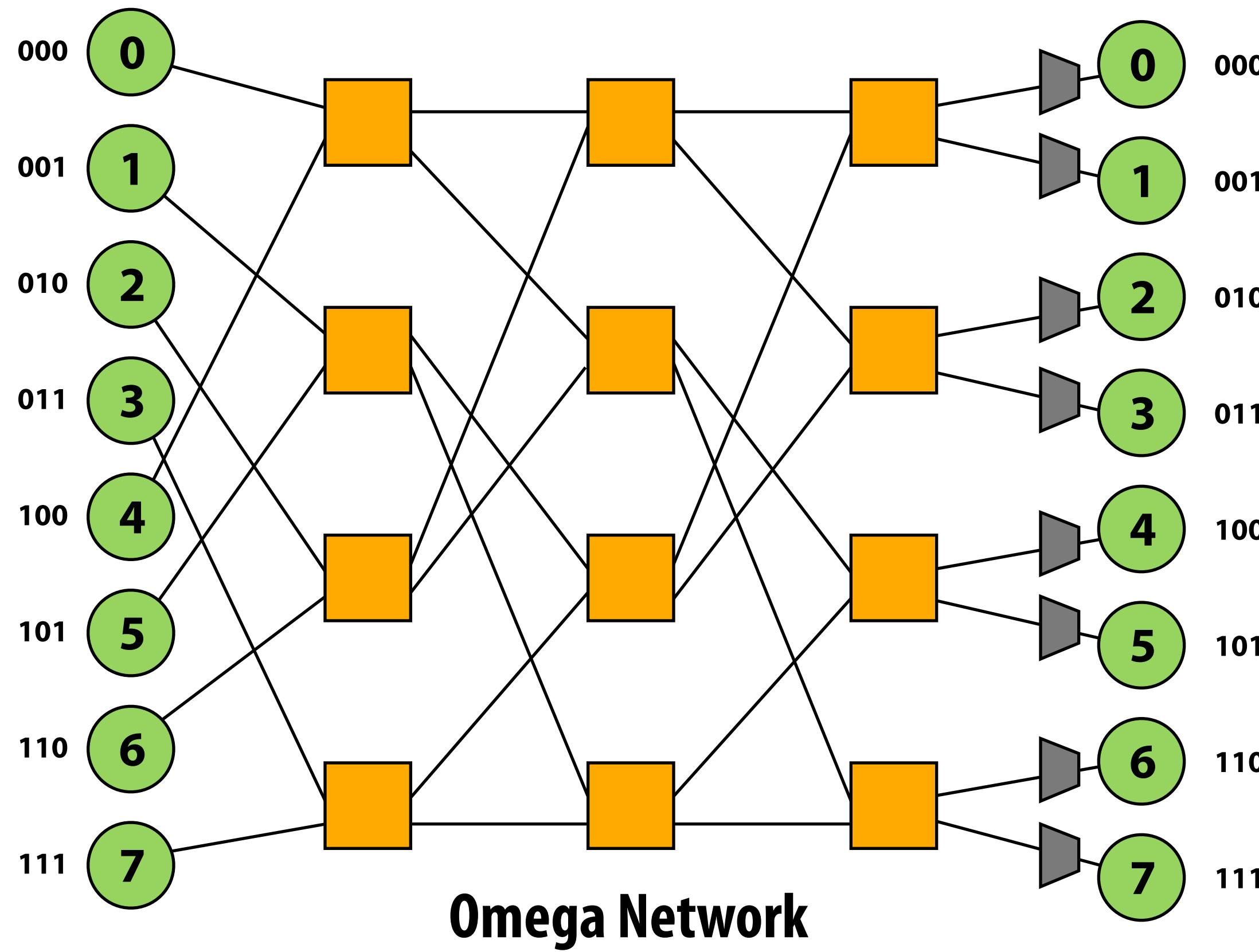
- **Low latency:  $O(\lg N)$**
- **Radix:  $O(\lg N)$**
- **Number of links  $O(N \lg N)$**



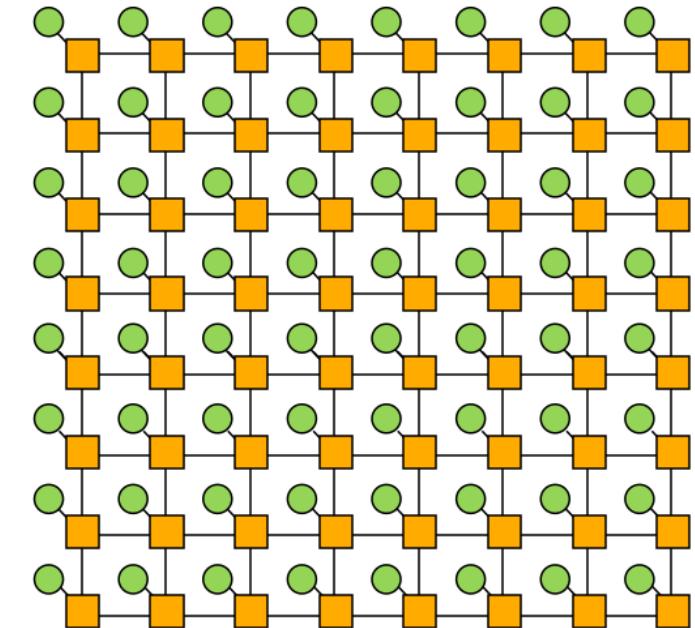
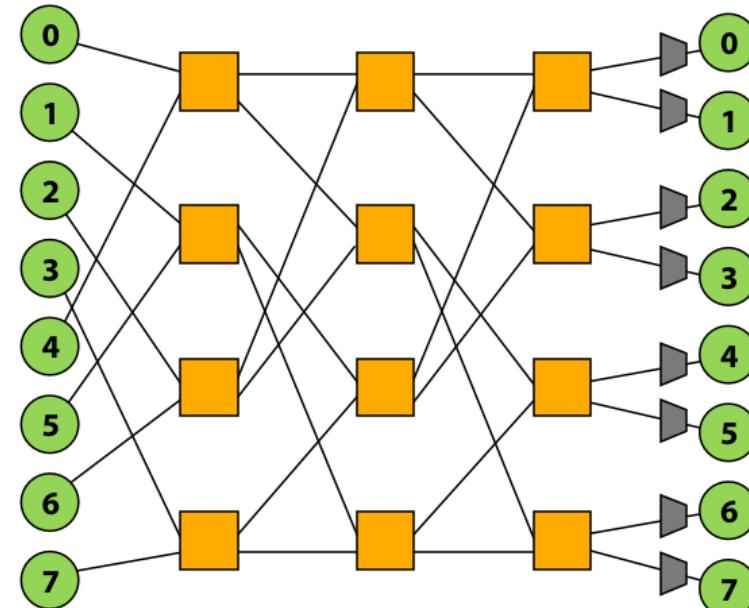
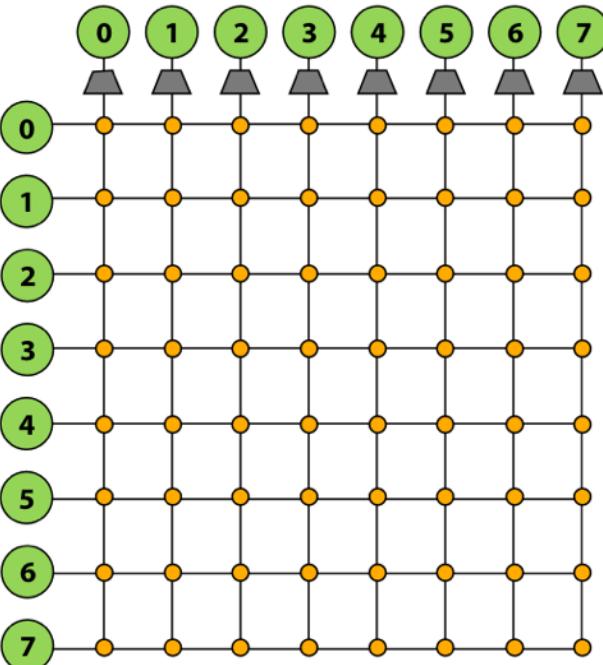
- **6D hypercube used in 64-core Cosmic Cube computer developed at Caltech in the 80s.**

# Multi-stage logarithmic

- Indirect network with multiple switches between terminals
- Cost:  $O(N \lg N)$
- Latency:  $O(\lg N)$
- Many variations: Omega, butterfly, Clos networks, etc...



# Review: network topologies



**Topology**

**Crossbar**

**Multi-stage log.**

**Mesh**

**Direct/Indirect**

**Indirect**

**Indirect**

**Direct**

**Blocking/  
Non-blocking**

**Non-blocking**

**Blocking**  
(one discussed in class  
is, others are not)

**Blocking**

**Cost**

**$O(N^2)$**

**$O(N \lg N)$**

**$O(N)$**

**Latency**

**$O(1)$**

**$O(\lg N)$**

**$O(\sqrt{N})$**   
**(average)**

# **Buffering and flow control**

# Circuit switching vs. packet switching

- **Circuit switching sets up a full path (acquires all resources) between sender and receiver prior to sending a message**

- Establish route (reserve links) then send data
- Higher bandwidth transmission (no per-packet link mgmt overhead)
- Does incur overhead to set up/tear down path
- Reserving links can result in low utilization



- **Packet switching makes routing decisions per packet**

- Route each packet individually (possibly over different network links)
- Opportunity to use link for a packet whenever link is idle
- Overhead due to dynamic switching logic during transmission
- No setup/tear down overhead



# Granularity of communication

## ■ Message

- Unit of transfer between network clients (e.g., cores, memory)
- Can be transmitted using many packets

## ■ Packet

- Unit of transfer for network
- Can be transmitted using multiple flits (will discuss later)

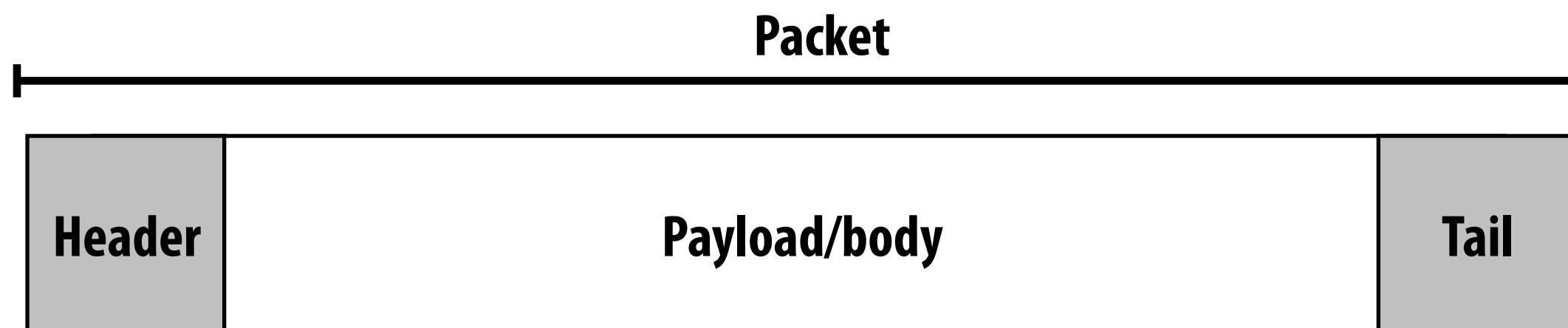
## ■ Flit (flow control digit)

- Packets broken into smaller units called “flits”
- Flit: (“flow control digit”) a unit of flow control in the network
- Flits become minimum granularity of routing/buffering

# Packet format

- A packet consists of:

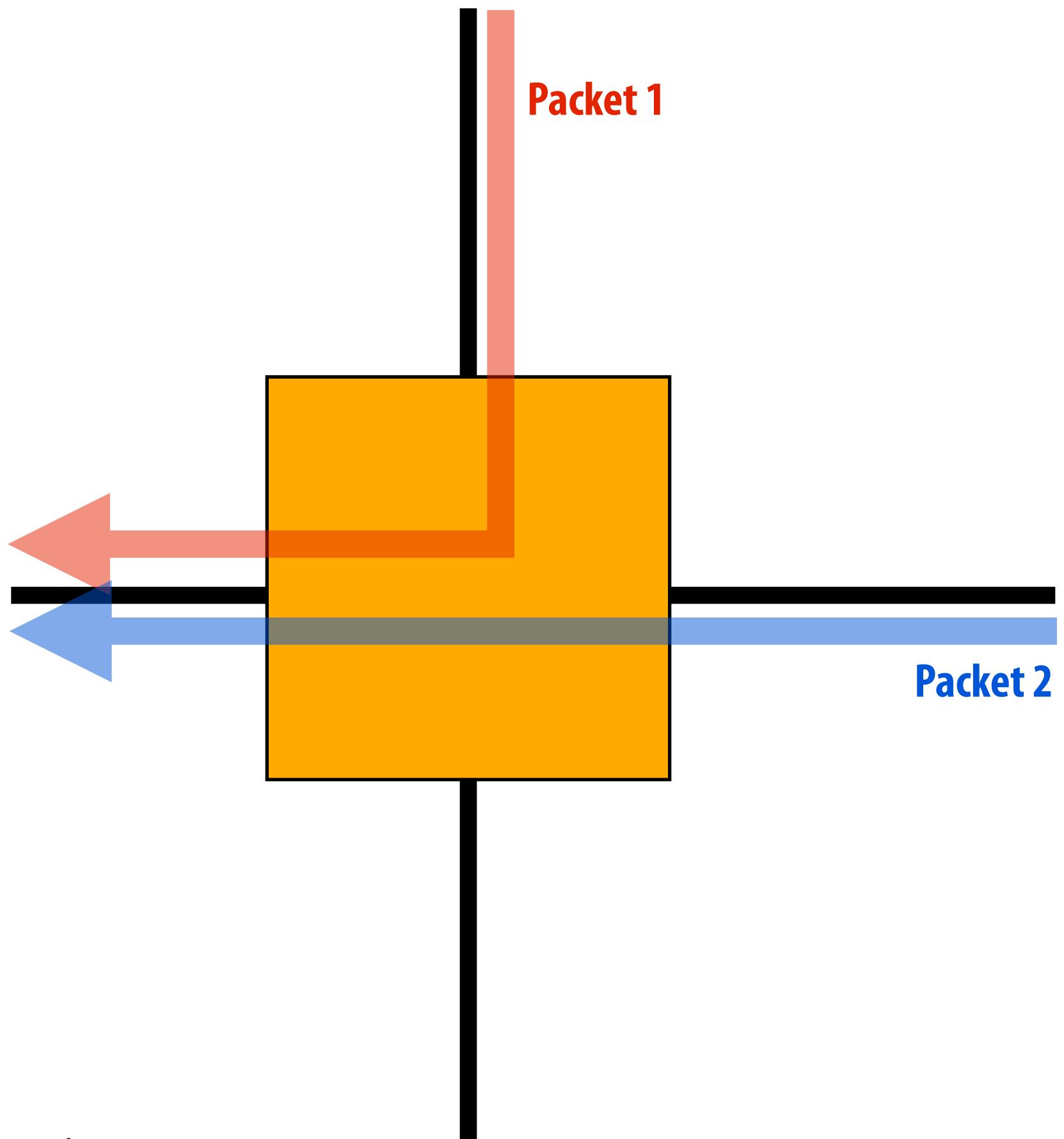
- Header:
  - Contains routing and control information
  - At start of packet to router can start forwarding early
- Payload/body: contains data to be sent
- Tail
  - Contains control information, e.g., error code
  - Generally located at end of packet so it can be generated “on the way out” (sender computes checksum, appends it to end of packet)



# Handling contention

**Scenario: two packets need to be routed onto the same outbound link at the same time**

- Options:
  - Buffer one packet, send it over link later
  - Drop one packet
  - Reroute one packet (deflection)
- In this lecture: we only consider buffering \*

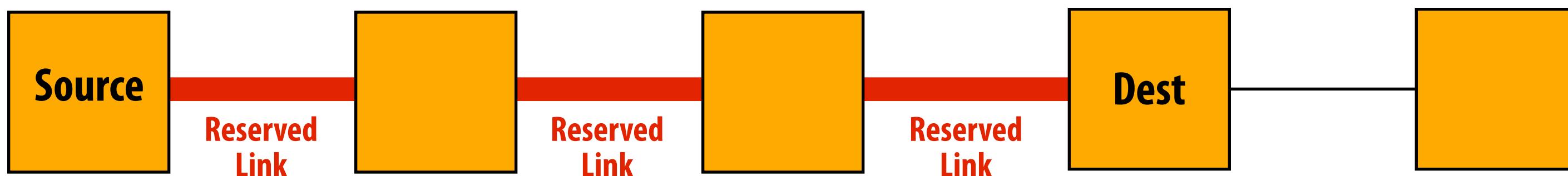


\* But recent research has looked at using bufferless networks with deflection routing as a power efficient interconnect for chip multiprocessors.

# Circuit-switched routing

## ■ High-granularity resource allocation

- Main idea: pre-allocate all resources (links across multiple switches) along entire network path for a message (“setup a flow”)



## ■ Costs

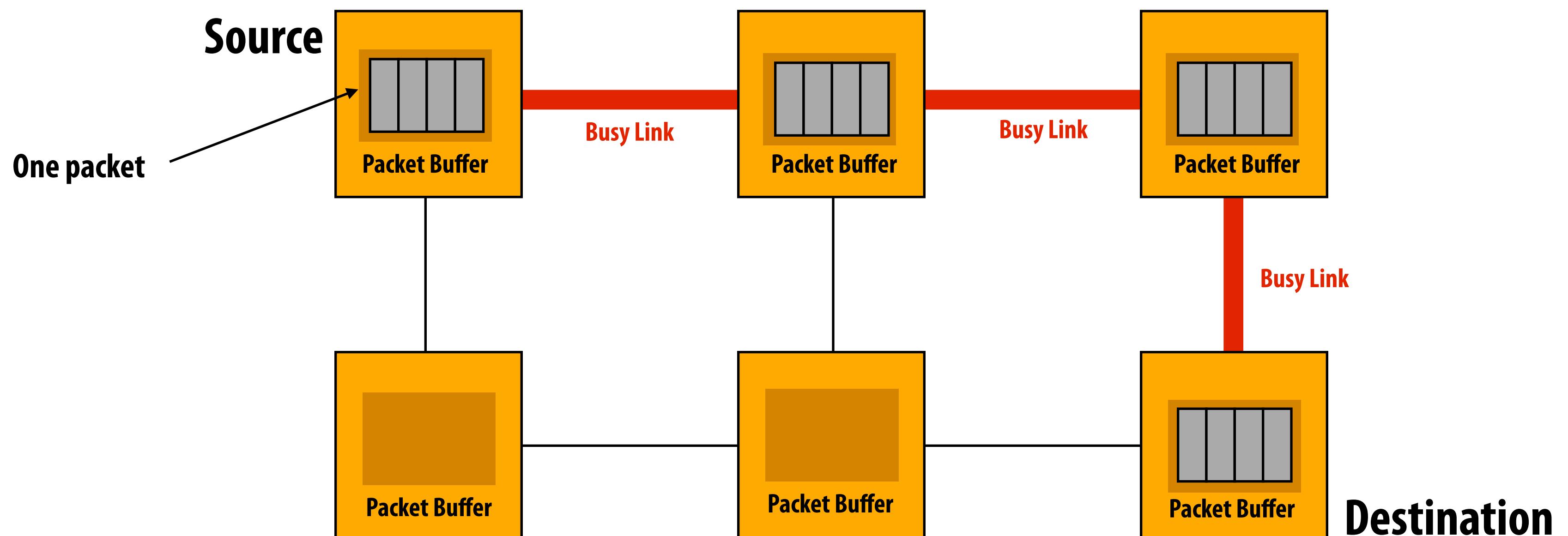
- Needs setup phase (“probe”) to set up the path (and to tear it down and release the resources when message complete)
- Lower link utilization. Transmission of two messages cannot share same link (even if some resources on a preallocated path are no longer utilized during a transmission)

## ■ Benefits

- No contention during transmission due to preallocation, so no need for buffering
- Arbitrary message sizes (once path is set up, send data until done)

# Store-and-forward (packet-based routing)

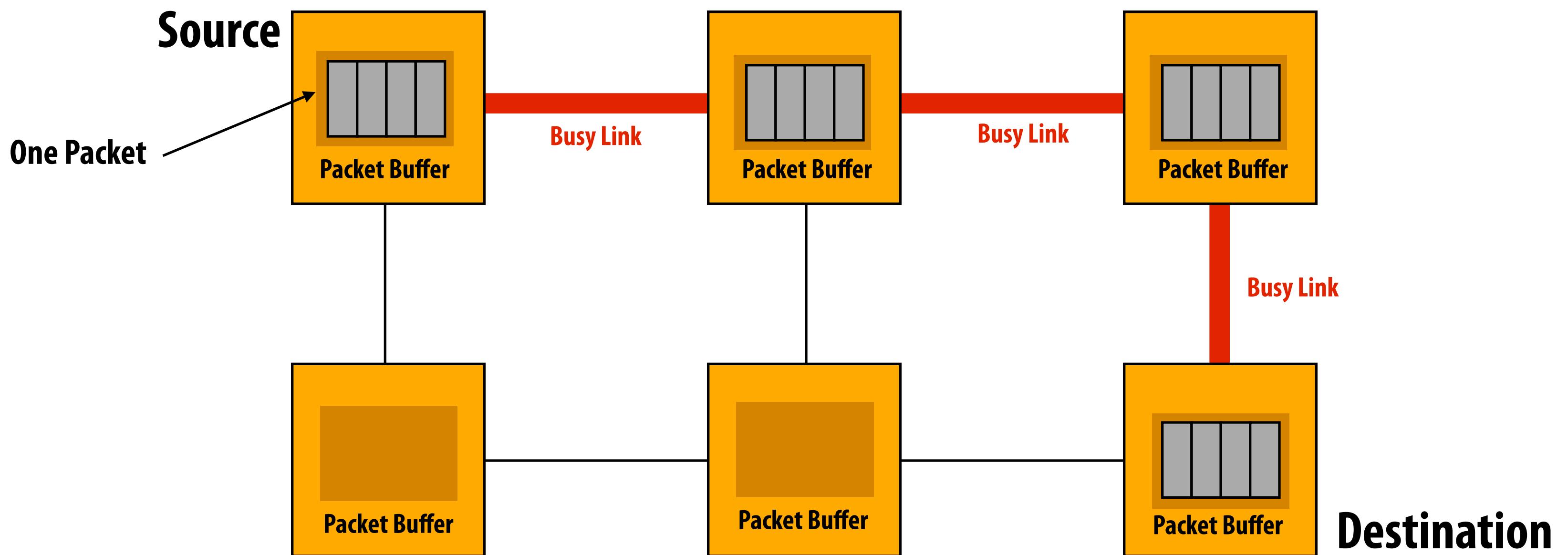
- Packet copied entirely into network switch before moving to next node
- Flow control unit is an entire packet
  - Different packets from the same message can take different routes, but all data in a packet is transmitted over the same route
- Requires buffering for entire packet in each router
- High per-packet latency (latency = packet transmission time on link  $\times$  network distance)



Note to students: in lecture this slide was animated and the final build shown here is not illustrative of store-and-forward routing concept (please refer to lecture video)

# Cut-through flow control (also packet-based)

- Switch starts forwarding data on next link as soon as packet header is received (header determines how much link bandwidth packet requires, where to route)
- Result: reduced transmission latency
  - Cut-through routing reduces to store-and-forward under high contention. Why?



Store and forward solution from previous slide: 3 hops x 4 units of time to transmit packet over a single link = 12 units of time

Cut-through solution: 5 steps of latency for head of packet to get to destination + 3 units of time for rest of packet = 8 units of time

Note to students: in lecture this slide was animated and the final build shown here is not illustrative of the cut-through routing concept (please refer to lecture video)

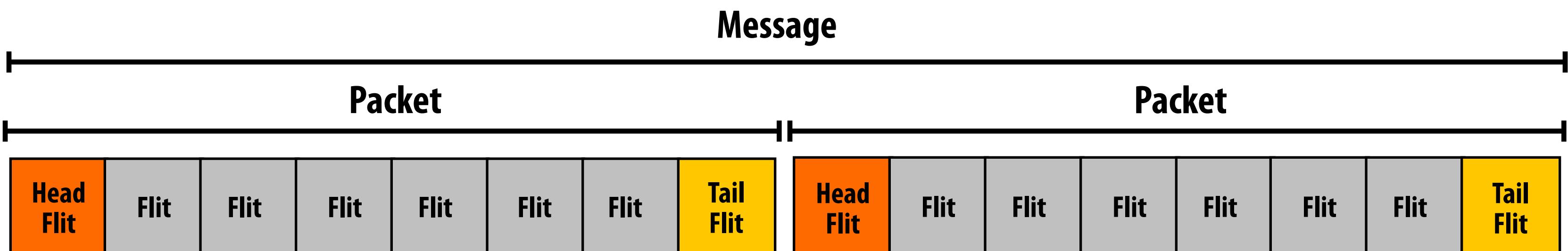
# Cut-through flow control

- If output link is blocked (cannot transmit head), transmission of tail can continue
  - Worst case: entire message is absorbed into a buffer in a switch (cut-through float control degenerates to store-and-forward in this case)
  - Requires switches to have buffering for entire packet, just like store-and-forward

# Wormhole flow control

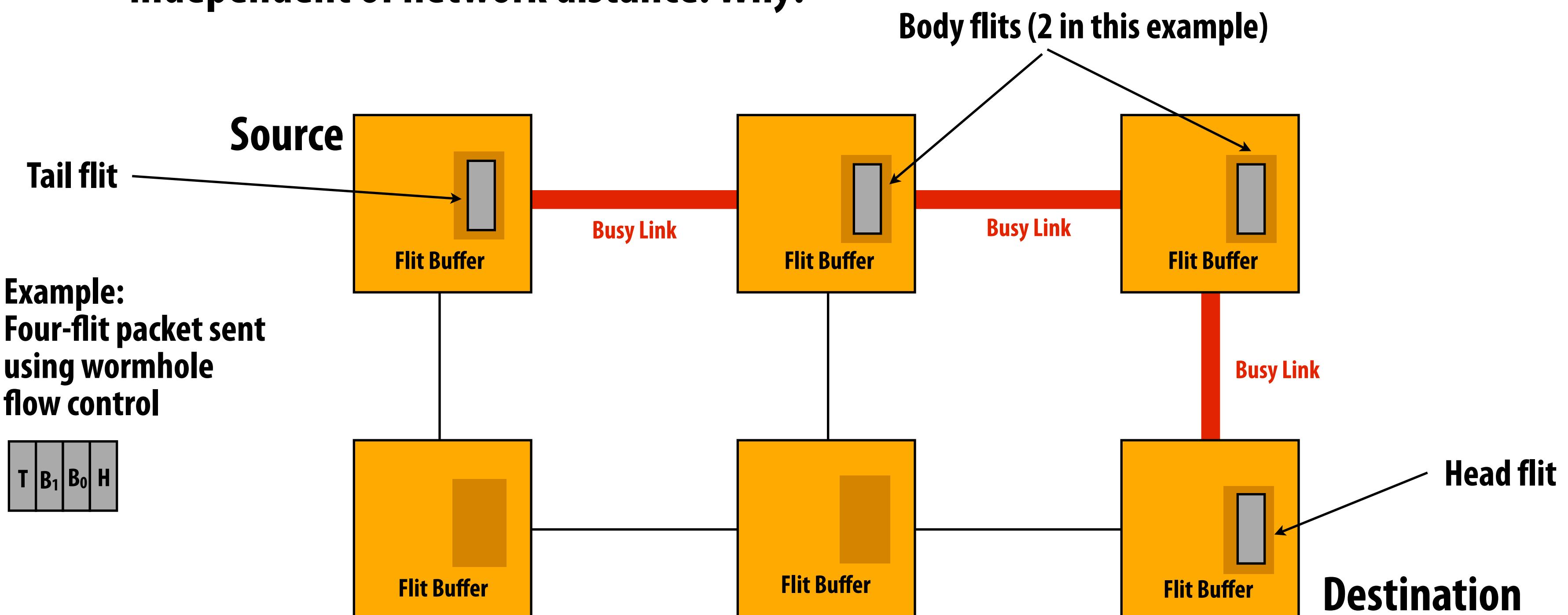
## ■ Flit (flow control digit)

- Packets broken into smaller units called “flits”
- Flit: (“flow control digit”) a unit of flow control in the network
- Flits become minimum granularity of routing/buffering
  - Recall: up until now, packets were the granularity of transfer AND flow control and buffering (store-and-forward, cut-through routing)



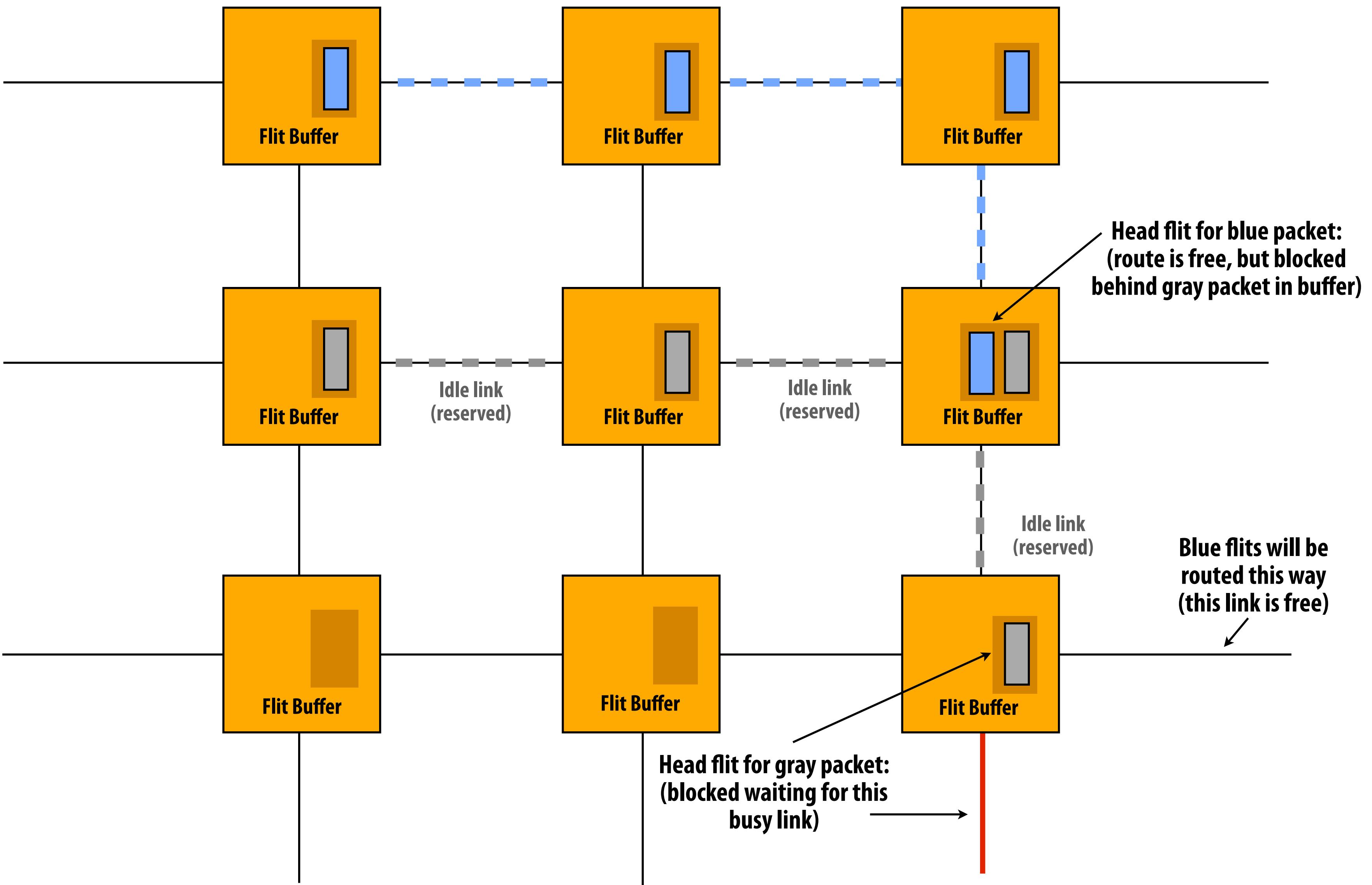
# Wormhole flow control

- Routing information only in head flit
  - Body flits follows head, tail flit flows body
  - If head flit blocks, rest of packet stops
  - Completely pipelined transmission
    - For long messages, latency is almost entirely independent of network distance. Why?



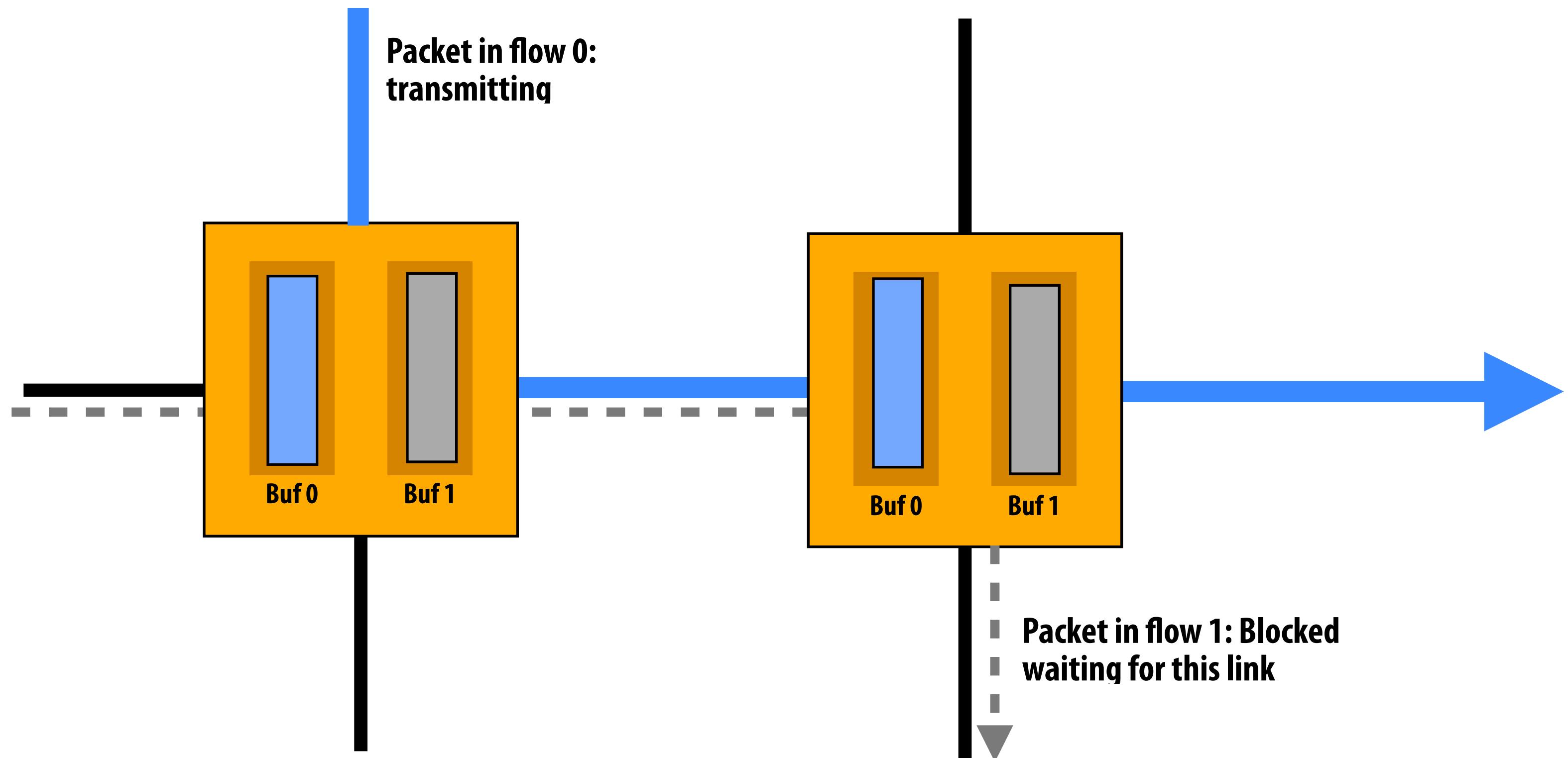
**Note to students: in lecture this slide was animated and the final build shown here is not illustrative of wormhole routing concept (please refer to lecture video)**

# Problem: head-of-line blocking



# Virtual channel flow control

- Multiplex multiple operations over single physical channel
- Divide switch's input buffer into multiple buffers sharing a single physical channel
- Reduces head-of-line blocking



# Other uses of virtual channels

## ■ Deadlock avoidance

- Can be used to break cyclic dependency of resources
- Prevent cycles by ensuring requests and responses use different virtual channels
- “Escape” VCs: retain at least one virtual channel that uses deadlock-free routing

## ■ Prioritization of traffic classes

- Provide quality-of-service guarantees
- Some virtual channels have higher priority than others

# Current research topics

- **Energy efficiency of interconnections**
  - **Interconnect can be energy intensive (~35% of total chip power in MIT RAW research processor)**
  - **Bufferless networks**
  - **Other techniques: turn on/off regions of network, use fast and slow networks**
- **Prioritization and Quality of service guarantees**
  - **Prioritize packets to improve multi-processor performance (e.g., some applications may be more sensitive to network performance than others)**
  - **Throttle endpoints (e.g., cores) based on network feedback**
- **New/emerging technologies**
  - **Die stacking (3D chips)**
  - **Photonic networks-on-chip (use optical waveguides instead of wires)**
  - **Reconfigurable devices (FPGAs): create custom interconnects tailored to application (see CMU projects: CONNECT, CoRAM, Shrinkwrap)**

# Summary

- **The performance of the interconnection network in a modern multi-processor is absolutely critical to overall system performance**
  - **Buses do not scale to many nodes**
  - **Historically interconnect was off-chip network connecting sockets, boards, racks**
  - **Today, all these issues apply to the design of on-chip networks**
- **Network topologies differ in performance, cost, complexity tradeoffs**
  - **e.g., crossbar, ring, mesh, torus, multi-stage network, fat tree, hypercube**
- **Challenge: efficiently routing data through network**
  - **Interconnect is a precious resource (communication is expensive!)**
  - **Flit-based flow control: fine-grained flow control to make good use of available link bandwidth**
  - **If interested, much more to learn about (not discussed in this class): ensuring quality-of-service, prioritization, reliability, deadlock, livelock, etc.**