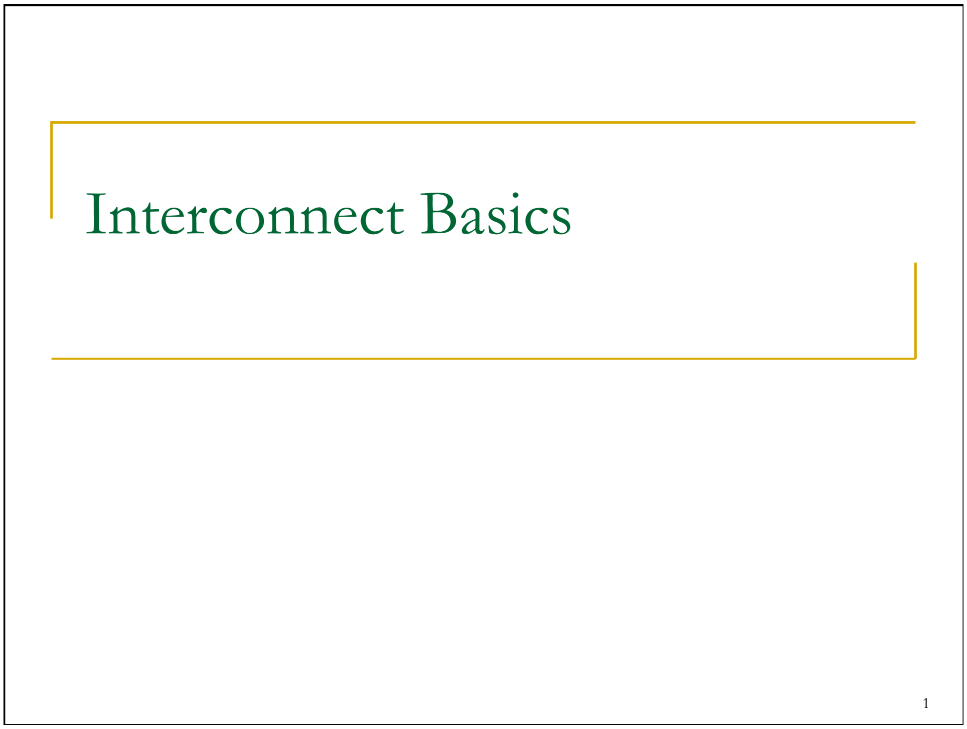
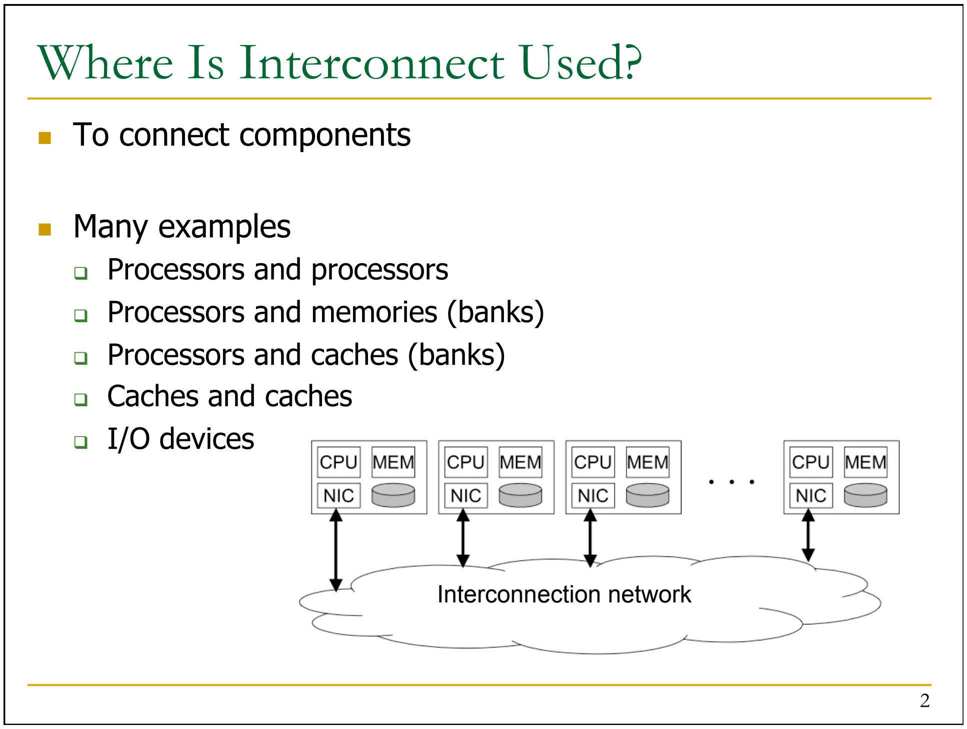
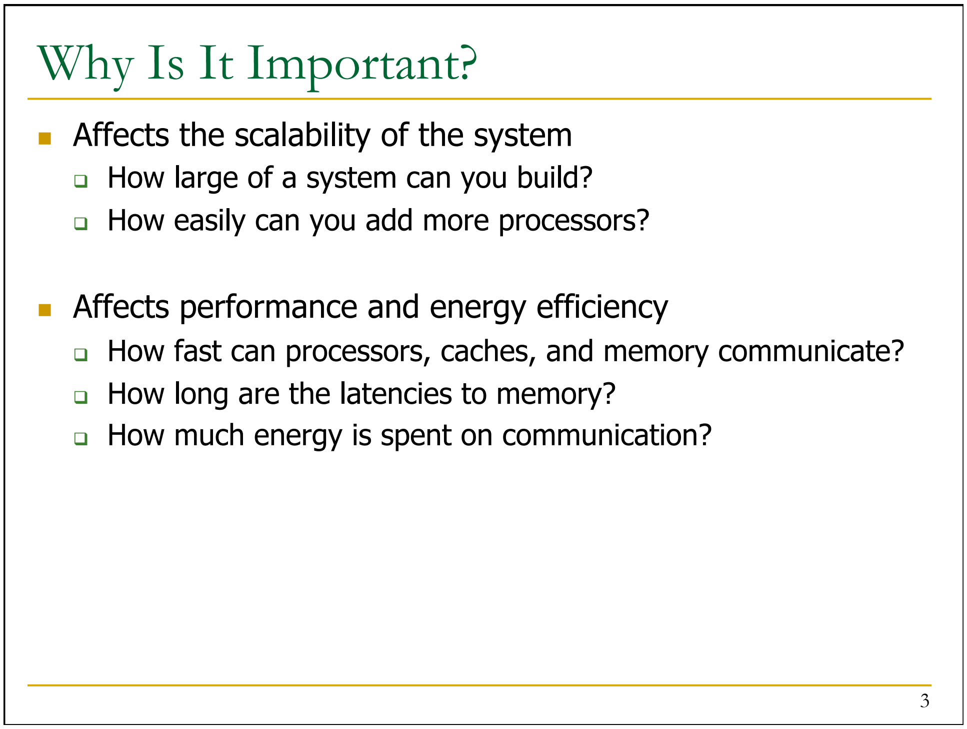


Interconnect Basics



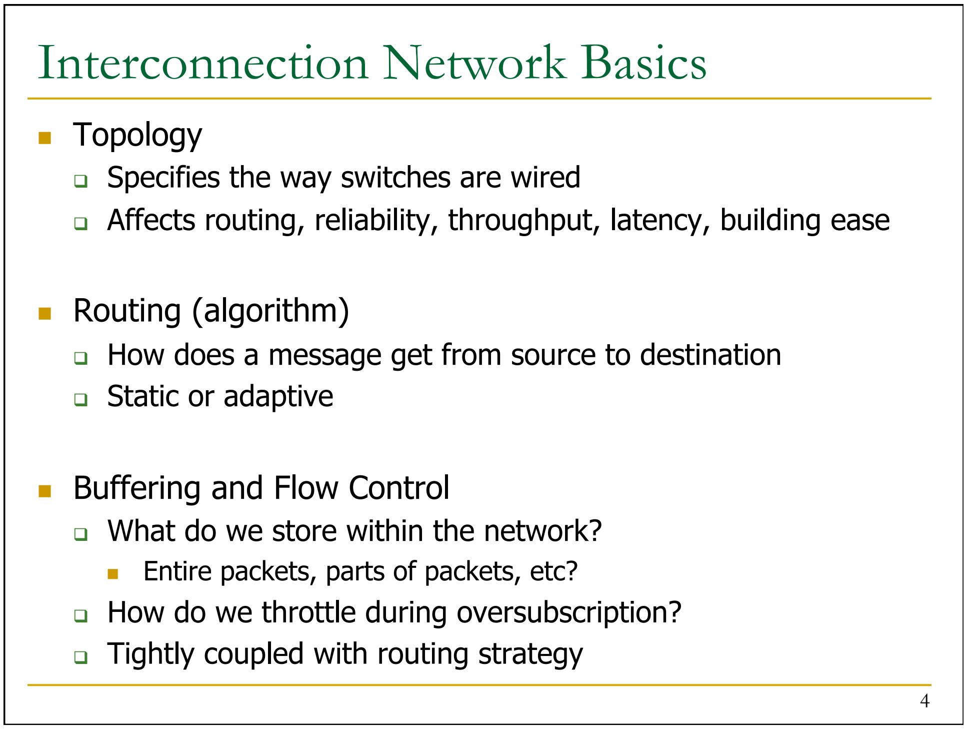
Where Is Interconnect Used?

- To connect components
- Many examples
  - Processors and processors
  - Processors and memories (banks)
  - Processors and caches (banks)
  - Caches and caches
  - I/O devices



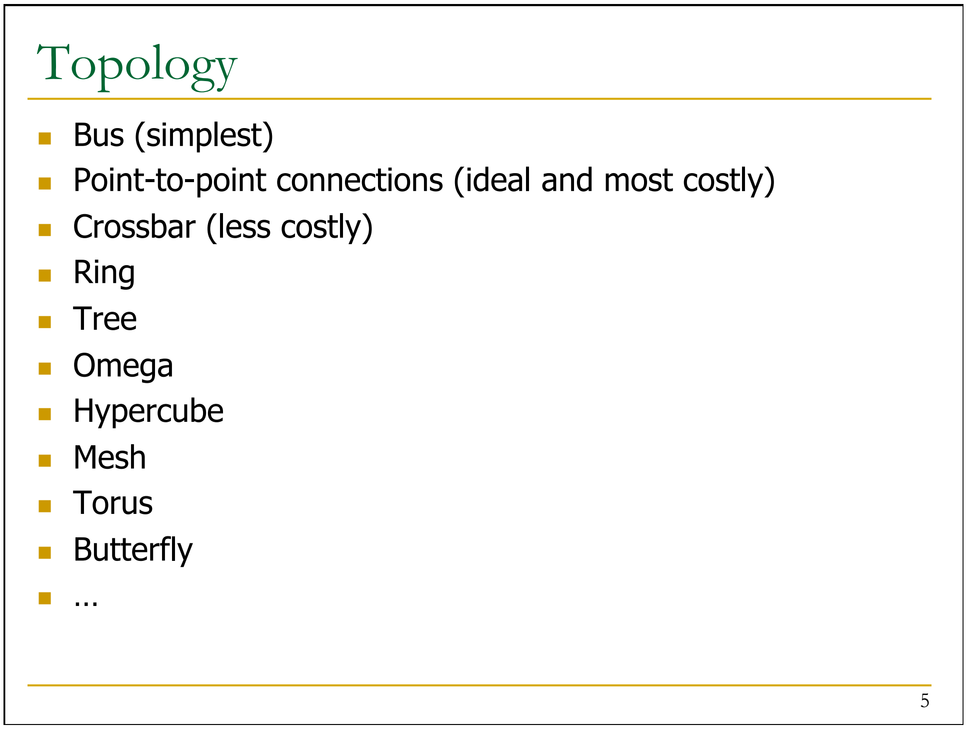
Why Is It Important?

- Affects the scalability of the system
  - How large of a system can you build?
  - How easily can you add more processors?
- Affects performance and energy efficiency
  - How fast can processors, caches, and memory communicate?
  - How long are the latencies to memory?
  - How much energy is spent on communication?



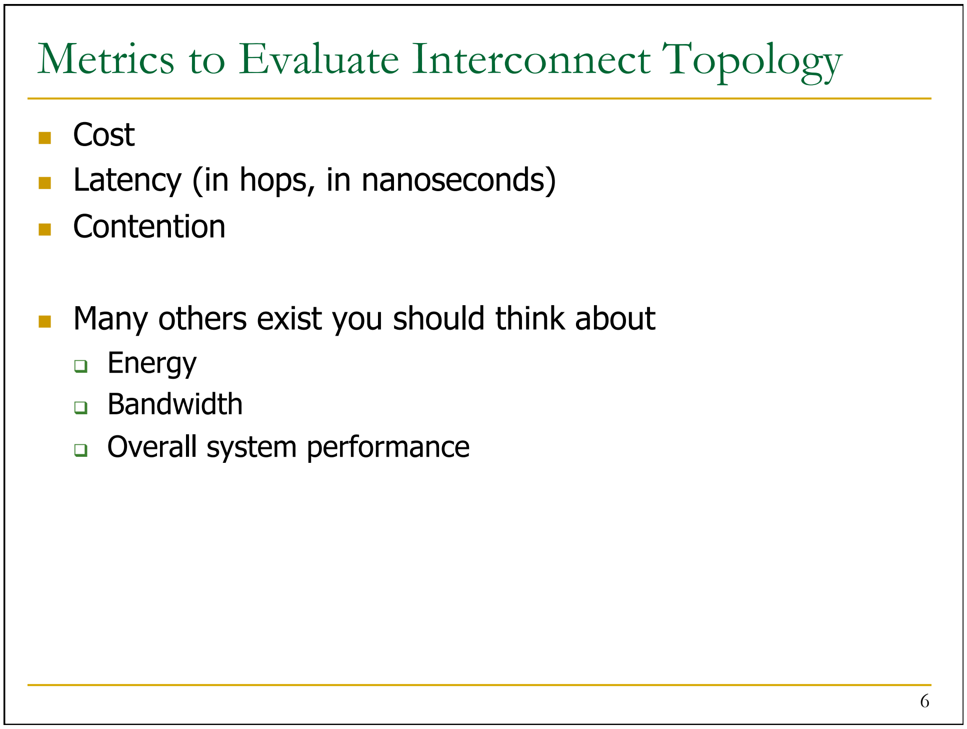
Interconnection Network Basics

- Topology
  - Specifies the way switches are wired
  - Affects routing, reliability, throughput, latency, building ease
- Routing (algorithm)
  - How does a message get from source to destination
  - Static or adaptive
- Buffering and Flow Control
  - What do we store within the network?
    - Entire packets, parts of packets, etc?
  - How do we throttle during oversubscription?
  - Tightly coupled with routing strategy



Topology

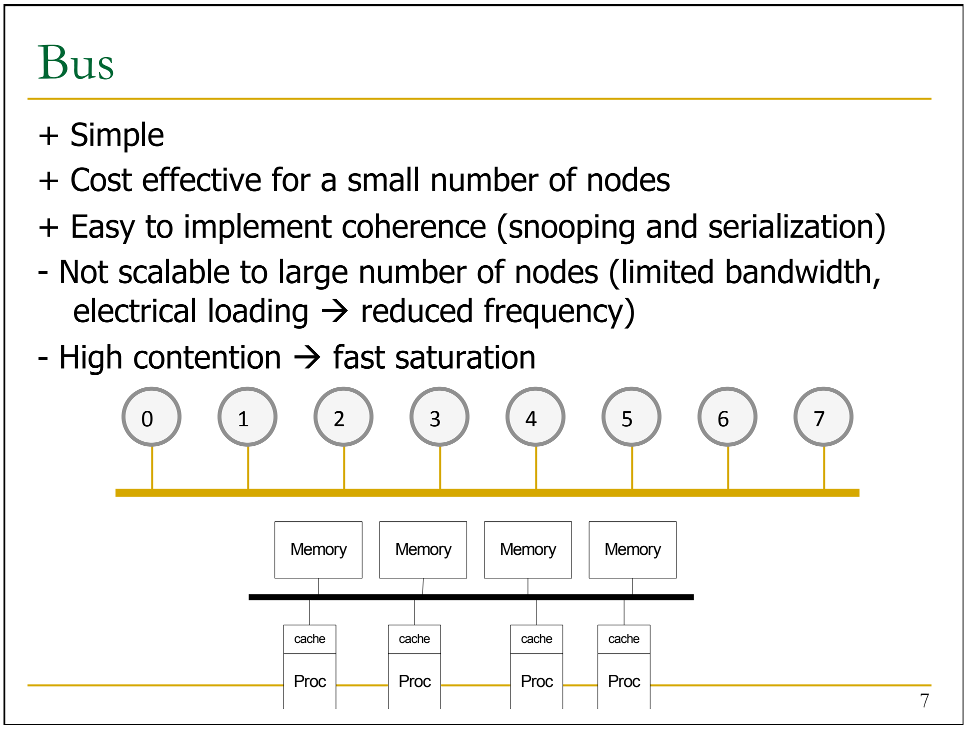
- Bus (simplest)
- Point-to-point connections (ideal and most costly)
- Crossbar (less costly)
- Ring
- Tree
- Omega
- Hypercube
- Mesh
- Torus
- Butterfly
- …



Metrics to Evaluate Interconnect Topology

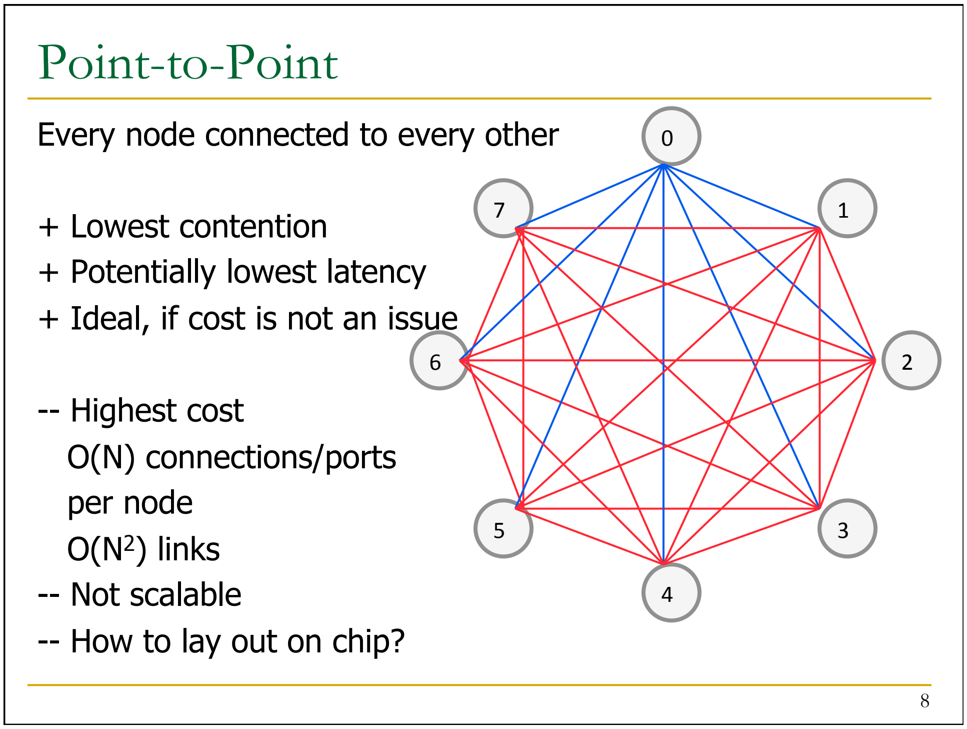
- Cost
- Latency (in hops, in nanoseconds)
- Contention

- Many others exist you should think about
  - Energy
  - Bandwidth
  - Overall system performance



Bus

+ Simple
+ Cost effective for a small number of nodes
+ Easy to implement coherence (snooping and serialization)
- Not scalable to large number of nodes (limited bandwidth, electrical loading → reduced frequency)
- High contention → fast saturation



Point-to-Point

Every node connected to every other

+ Lowest contention
+ Potentially lowest latency
+ Ideal, if cost is not an issue

-- Highest cost
   O(N) connections/ports per node
   O(N^2) links
-- Not scalable
-- How to lay out on chip?

Crossbar

- Every node connected to every other (non-blocking) except one can be using the connection at any given time
- Enables concurrent sends to non-conflicting destinations
- Good for small number of nodes

+ Low latency and high throughput

- Expensive

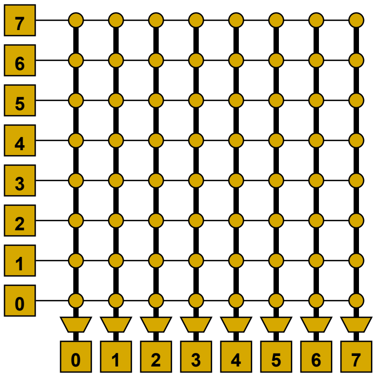
- Not scalable → $O(N^2)$ cost

- Difficult to arbitrate as N increases

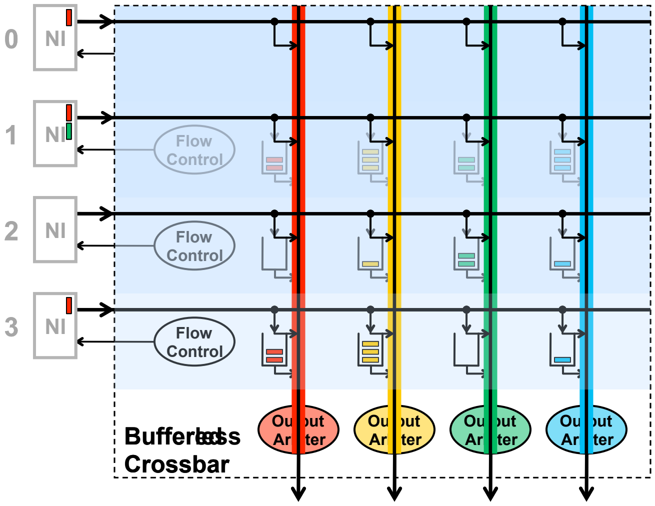
Used in core-to-cache-bank networks in

- IBM POWER5

- Sun Niagara I/II



Buffered Crossbar



+ Simpler arbitration/ scheduling

+ Efficient support for variable-size packets

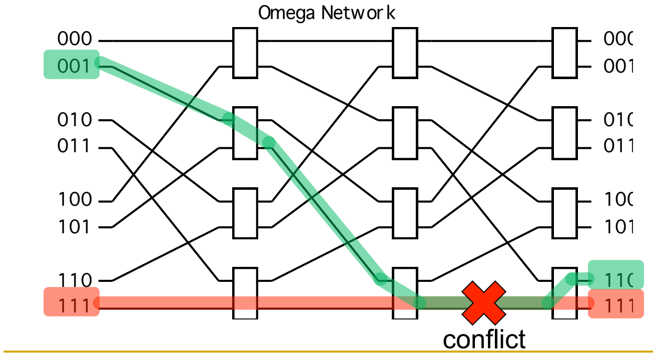
- Requires N^2 buffers

Can We Get Lower Cost than A Crossbar?

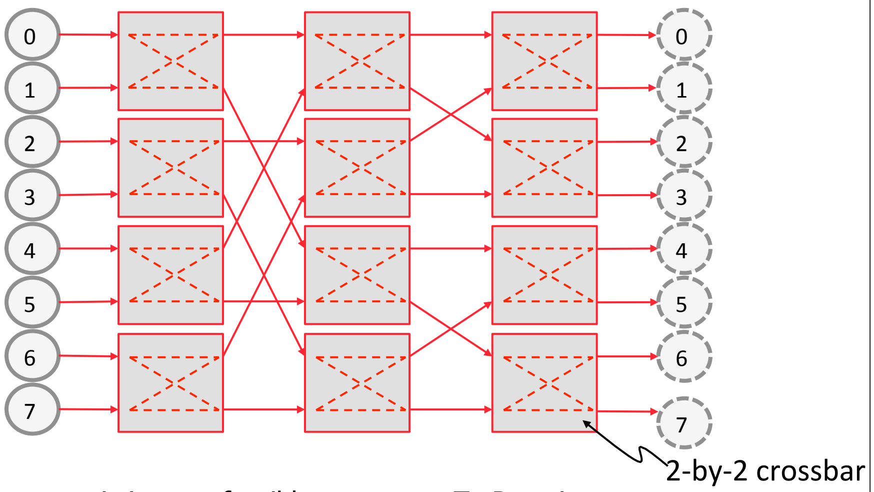
- Yet still have low contention?
- Idea: Multistage networks

Multistage Logarithmic Networks

- Idea: Indirect networks with multiple layers of switches between terminals/nodes
- Cost: O(NlogN), Latency: O(logN)
- Many variations (Omega, Butterfly, Benes, Banyan, …)
- Omega Network:

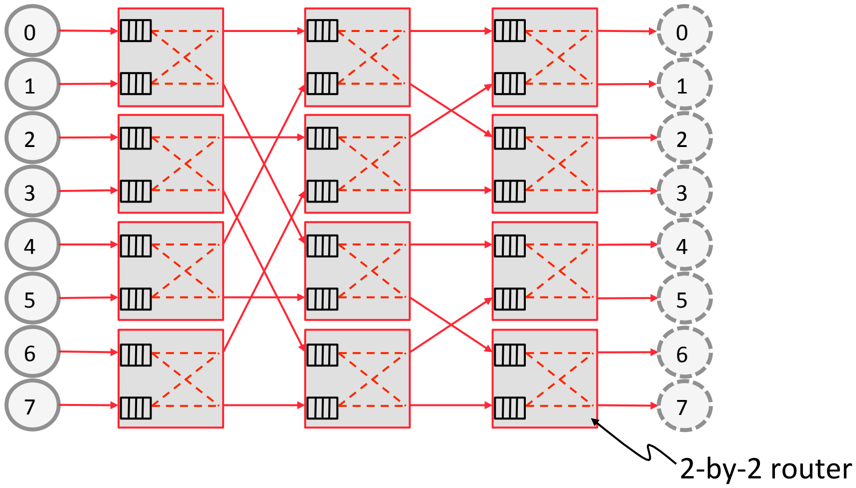


Multistage Circuit Switched

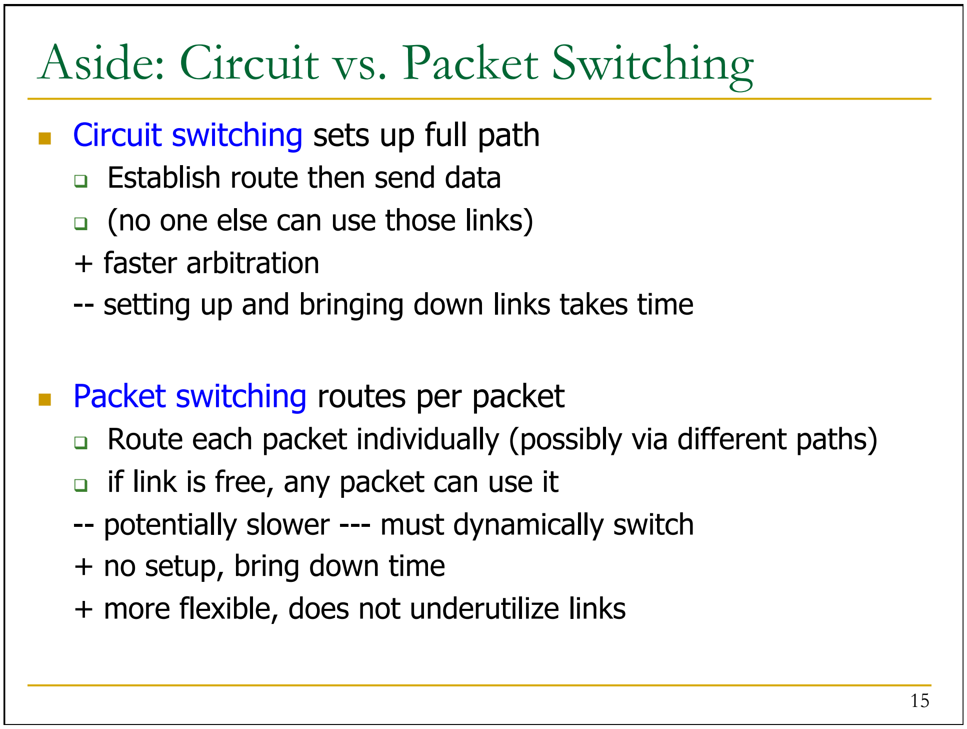


- More restrictions on feasible concurrent Tx-Rx pairs
- But more scalable than crossbar in cost, e.g., O(N logN) for Butterfly

Multistage Packet Switched



- Packets “hop” from router to router, pending availability of the next-required switch and buffer



Aside: Circuit vs. Packet Switching

- Circuit switching sets up full path
  - Establish route then send data
  - (no one else can use those links)

  + faster arbitration

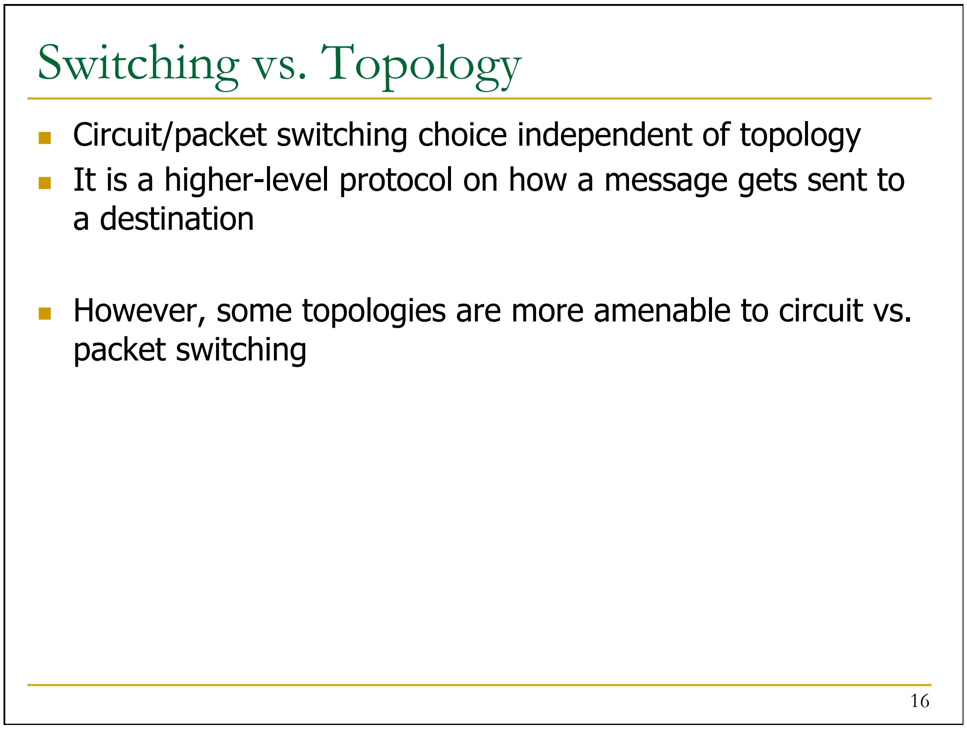
  -- setting up and bringing down links takes time

- Packet switching routes per packet
  - Route each packet individually (possibly via different paths)
  - if link is free, any packet can use it

  -- potentially slower --- must dynamically switch

  + no setup, bring down time

  + more flexible, does not underutilize links

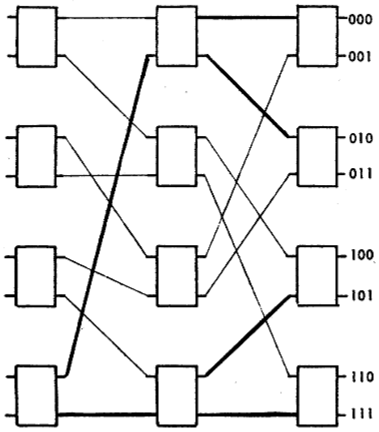


Switching vs. Topology

- Circuit/packet switching choice independent of topology
- It is a higher-level protocol on how a message gets sent to a destination
- However, some topologies are more amenable to circuit vs. packet switching

Another Example: Delta Network

- Single path from source to destination
- Does not support all possible permutations
- Proposed to replace costly crossbars as processor-memory interconnect



8x8 Delta network

Another Example: Omega Network

- Single path from source to destination
- All stages are the same

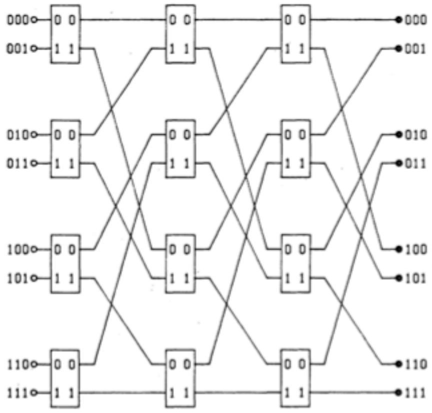


Fig. 2. Omega-network ($N = 8$).

Ring

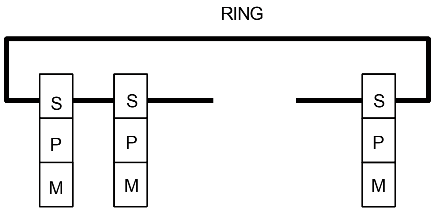
+ Cheap: O(N) cost

- High latency: O(N)

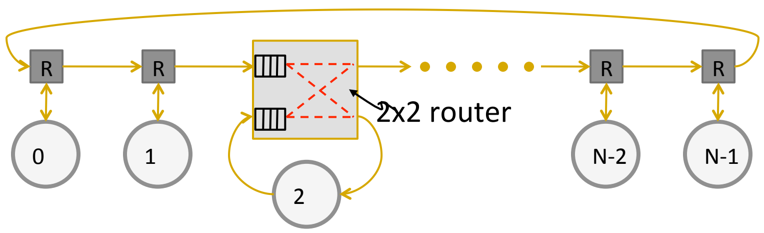
- Not easy to scale

  - Bisection bandwidth remains constant

Used in Intel Haswell, Intel Larrabee, IBM Cell, many commercial systems today



Unidirectional Ring

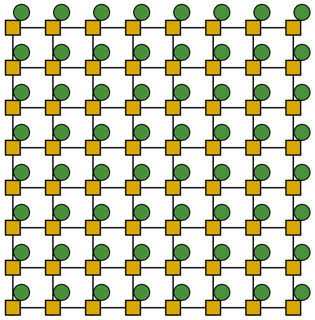


- Simple topology and implementation
  - Reasonable performance if N and performance needs (bandwidth & latency) still moderately low
  - O(N) cost
  - N/2 average hops; latency depends on utilization

Mesh

- O(N) cost
- Average latency: O(sqrt(N))
- Easy to layout on-chip: regular and equal-length links
- Path diversity: many ways to get from one node to another

- Used in Tilera 100-core
- And many on-chip network prototypes

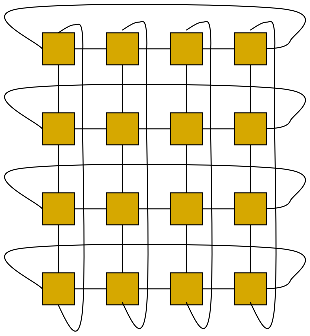


Torus

- Mesh is not symmetric on edges: performance very sensitive to placement of task on edge vs. middle
- Torus avoids this problem

+ Higher path diversity (and bisection bandwidth) than mesh

- Higher cost
- Harder to lay out on-chip
  - Unequal link lengths



Trees

Planar, hierarchical topology

Latency: O(logN)

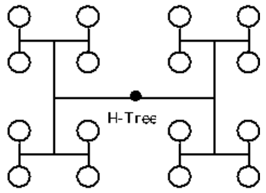
Good for local traffic

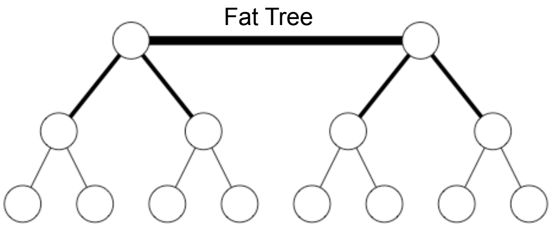
+ Cheap: O(N) cost

+ Easy to Layout

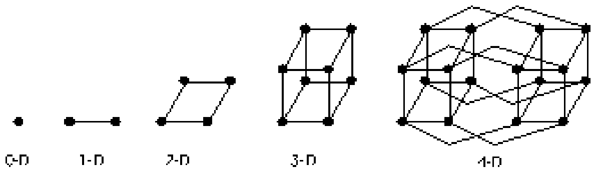
- Root can become a bottleneck

  Fat trees avoid this problem (CM-5)





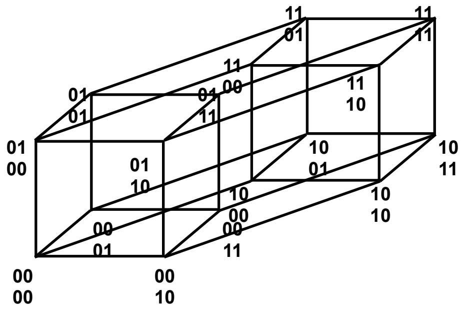
Hypercube



- Latency: O(logN)
- Radix: O(logN)
- #links: O(NlogN)

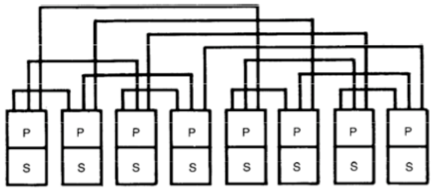
+ Low latency

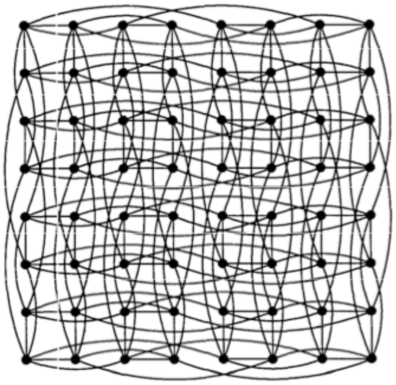
- Hard to lay out in 2D/3D



Caltech Cosmic Cube

- 64-node message passing machine
- Seitz, “The Cosmic Cube,” CACM 1985.

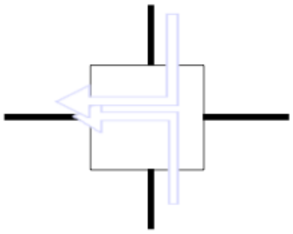




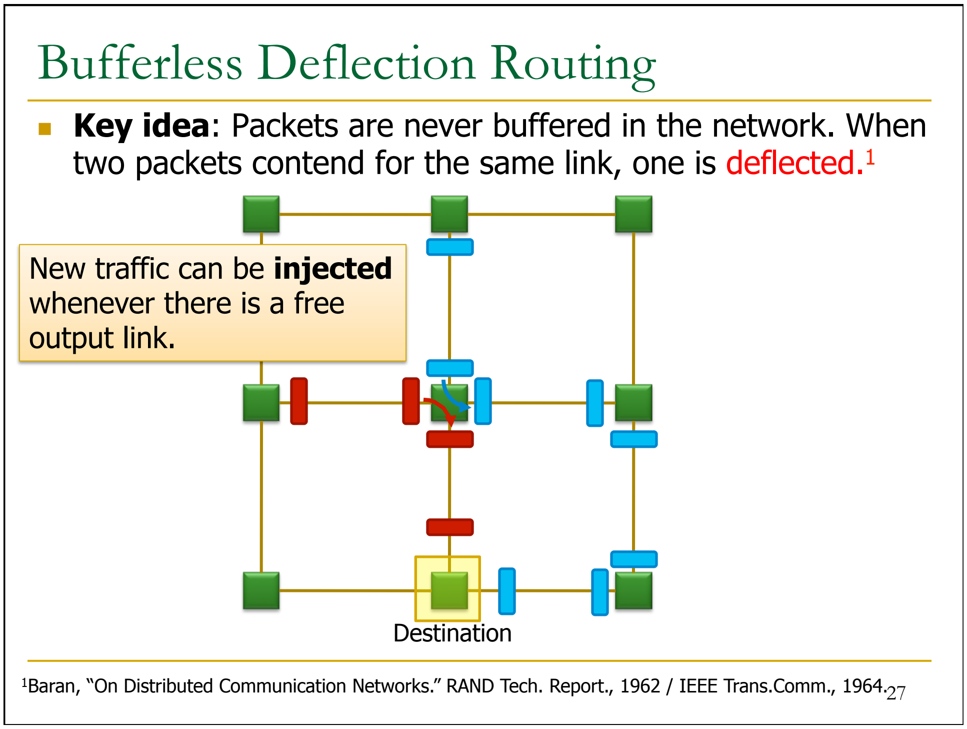
A hypercube connects $N = 2^n$ small computers, called nodes, through point-to-point communication channels in the Cosmic Cube. Shown here is a two-dimensional projection of a six-dimensional hypercube, or binary 6-cube, which corresponds to a 64-node machine.

FIGURE 1. A Hypercube (also known as a binary cube or a Boolean *n*-cube)

Handling Contention



- Two packets trying to use the same link at the same time
- What do you do?
  - Buffer one
  - Drop one
  - Misroute one (deflection)
- Tradeoffs?

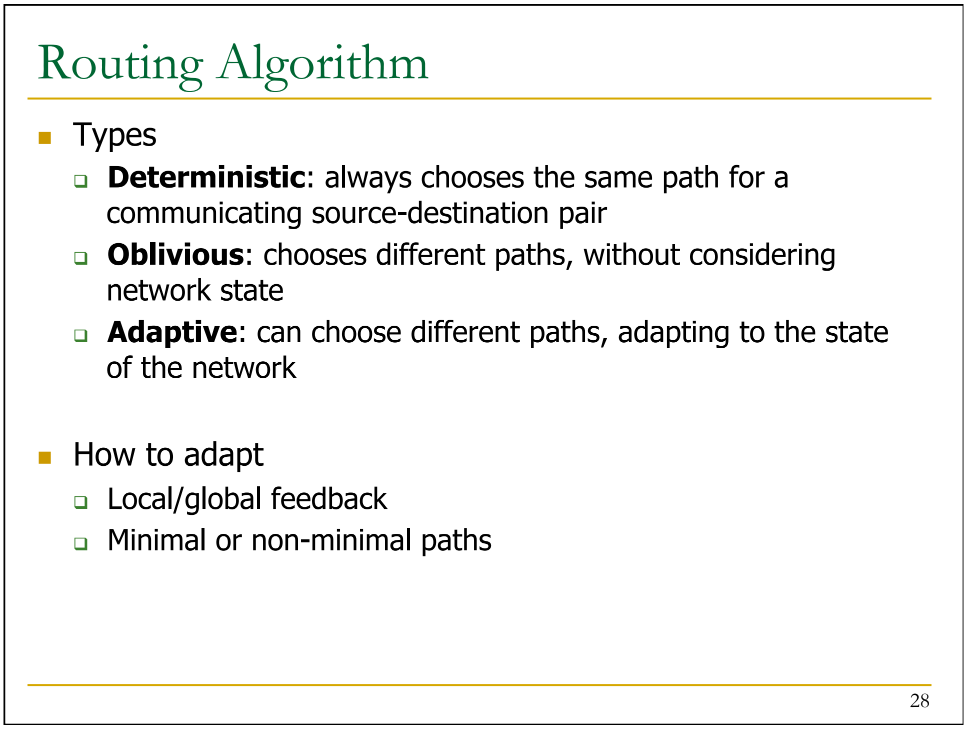


Bufferless Deflection Routing

- **Key idea**: Packets are never buffered in the network. When two packets contend for the same link, one is deflected.[1]

New traffic can be **injected** whenever there is a free output link.

[1] Baran, "On Distributed Communication Networks." RAND Tech. Report., 1962 / IEEE Trans.Comm., 1964.



Routing Algorithm

- Types
  - **Deterministic**: always chooses the same path for a communicating source-destination pair
  - **Oblivious**: chooses different paths, without considering network state
  - **Adaptive**: can choose different paths, adapting to the state of the network
- How to adapt
  - Local/global feedback
  - Minimal or non-minimal paths

Deterministic Routing

- All packets between the same (source, dest) pair take the same path
- Dimension-order routing
  - E.g., XY routing (used in Cray T3D, and many on-chip networks)
  - First traverse dimension X, then traverse dimension Y

+ Simple

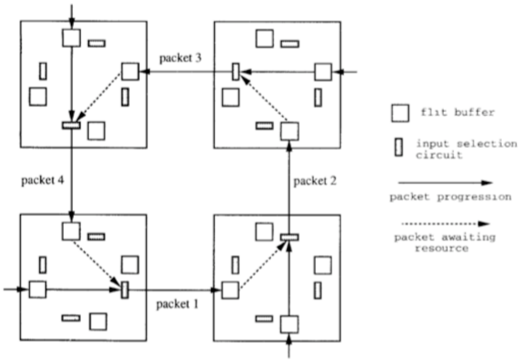
+ Deadlock freedom (no cycles in resource allocation)

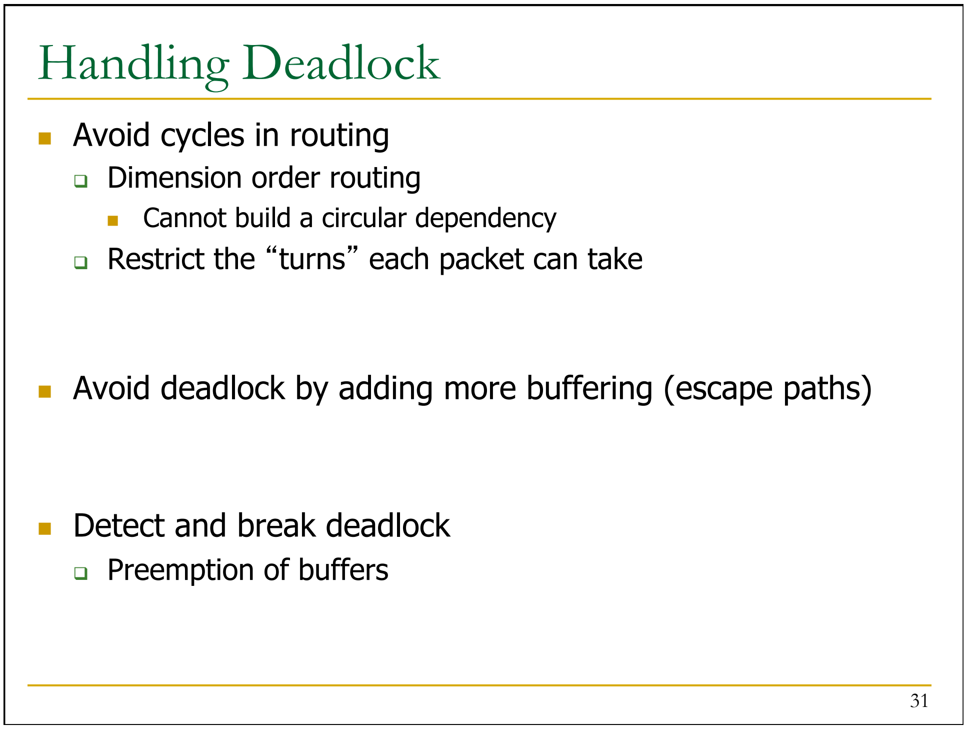
- Could lead to high contention

- Does not exploit path diversity

Deadlock

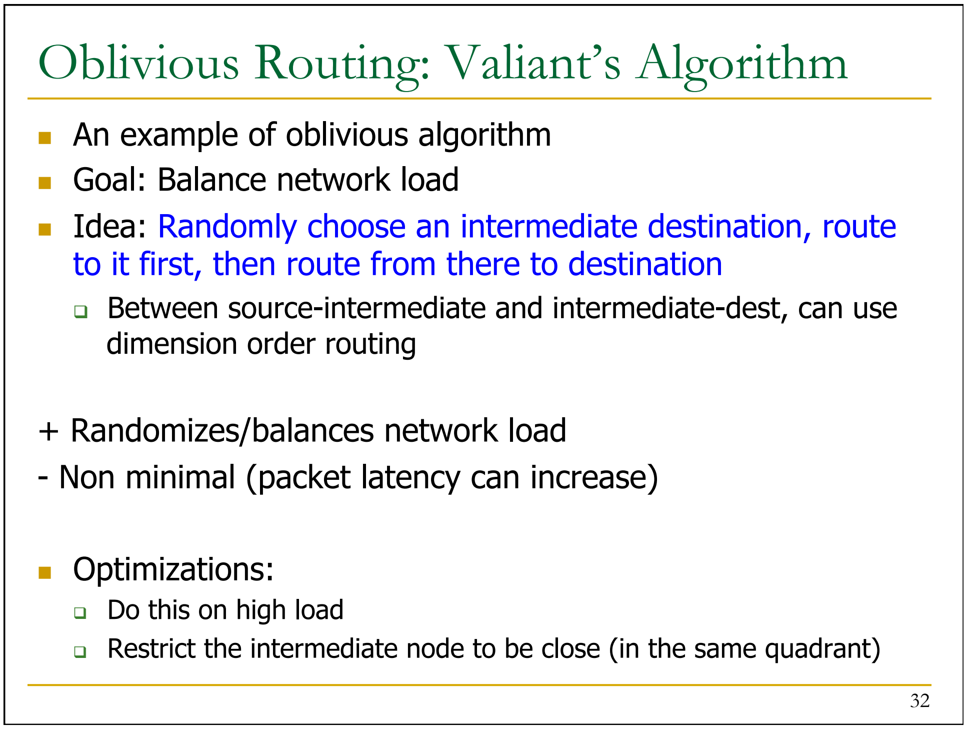
- No forward progress
- Caused by circular dependencies on resources
- Each packet waits for a buffer occupied by another packet downstream





Handling Deadlock

- Avoid cycles in routing
  - Dimension order routing
    - Cannot build a circular dependency
  - Restrict the “turns” each packet can take
- Avoid deadlock by adding more buffering (escape paths)
- Detect and break deadlock
  - Preemption of buffers



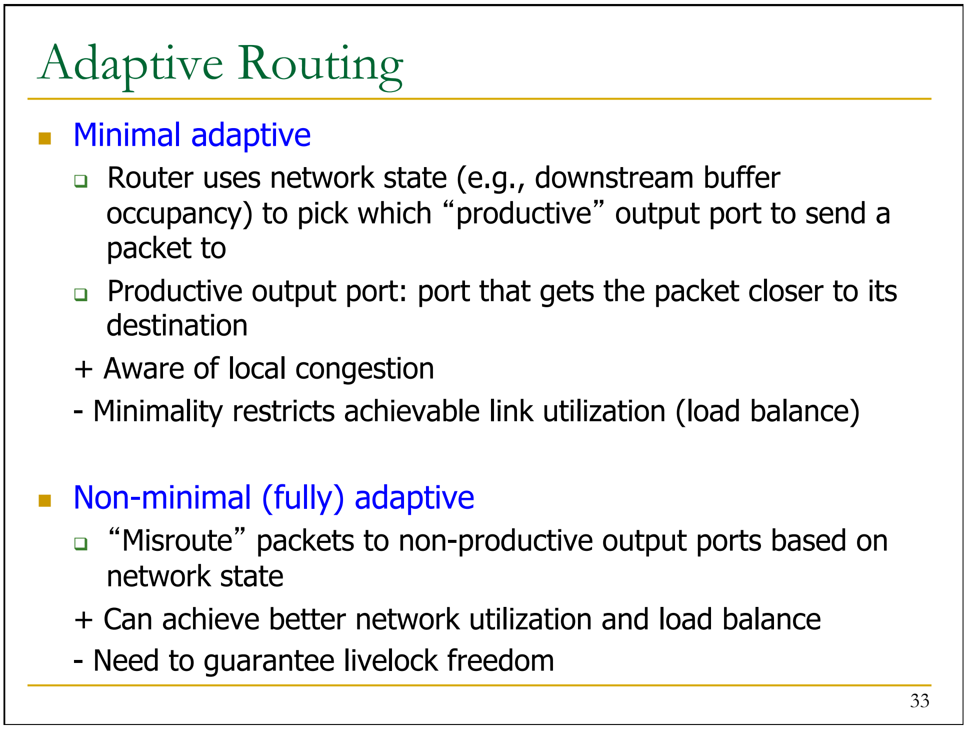
Oblivious Routing: Valiant’s Algorithm

- An example of oblivious algorithm
- Goal: Balance network load
- Idea: Randomly choose an intermediate destination, route to it first, then route from there to destination
  - Between source-intermediate and intermediate-dest, can use dimension order routing

+ Randomizes/balances network load

- Non minimal (packet latency can increase)

- Optimizations:
  - Do this on high load
  - Restrict the intermediate node to be close (in the same quadrant)



Adaptive Routing

- Minimal adaptive
  - Router uses network state (e.g., downstream buffer occupancy) to pick which "productive" output port to send a packet to
  - Productive output port: port that gets the packet closer to its destination

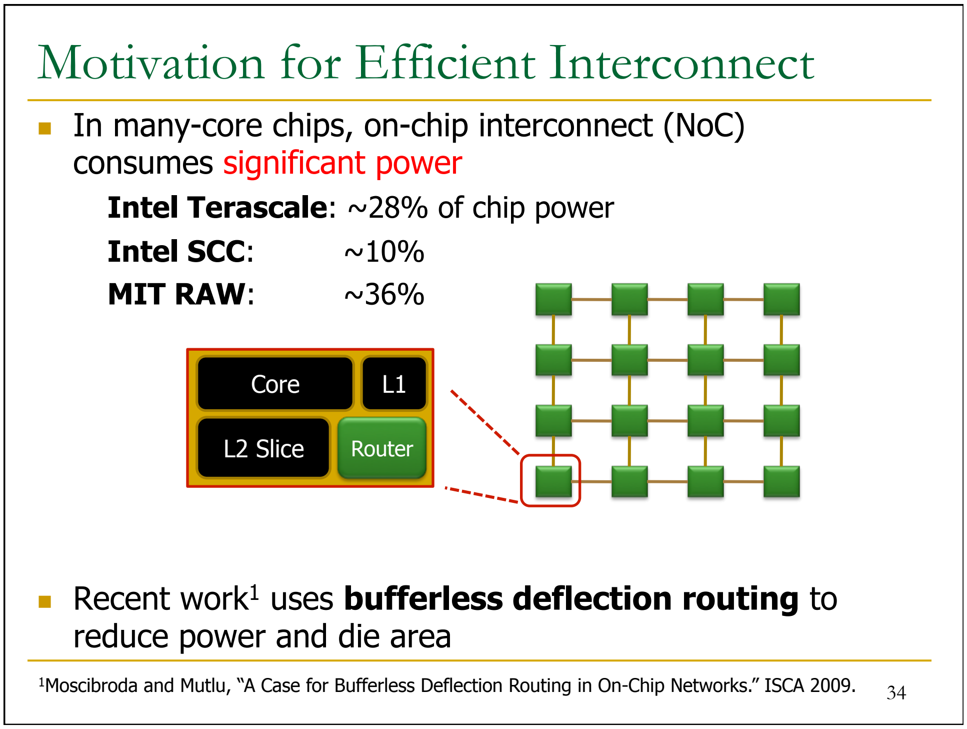
  + Aware of local congestion

  - Minimality restricts achievable link utilization (load balance)

- Non-minimal (fully) adaptive
  - "Misroute" packets to non-productive output ports based on network state

  + Can achieve better network utilization and load balance

  - Need to guarantee livelock freedom



Motivation for Efficient Interconnect

- In many-core chips, on-chip interconnect (NoC) consumes significant power

  Intel Terascale: ~28% of chip power

  Intel SCC: ~10%

  MIT RAW: ~36%

Core
L1
L2 Slice
Router

- Recent work[1] uses **bufferless deflection routing** to reduce power and die area

[1] Moscibroda and Mutlu, "A Case for Bufferless Deflection Routing in On-Chip Networks." ISCA 2009.