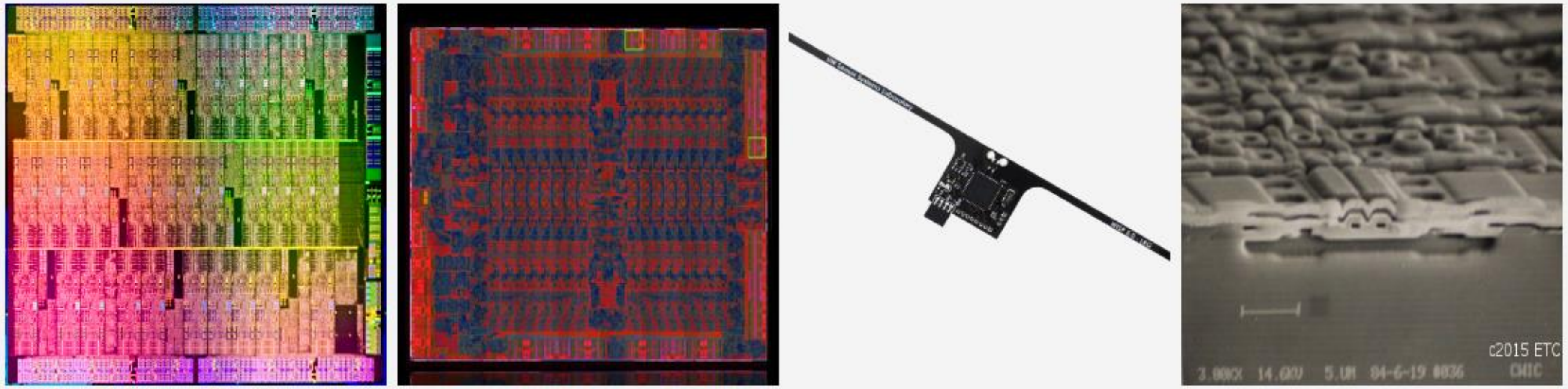
18-344: Computer Systems and the Hardware-Software Interface

Fall 2023



Course Description

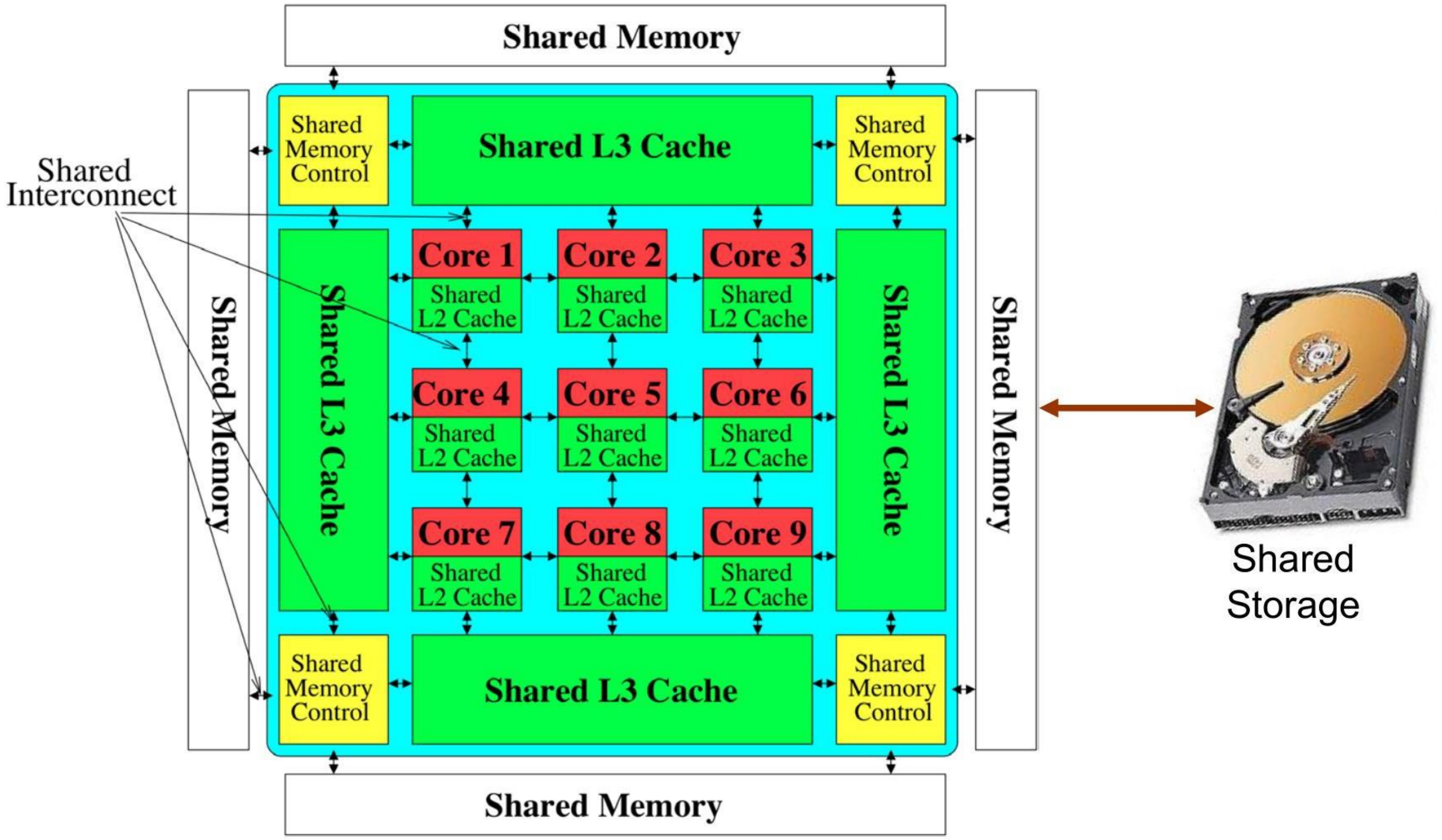
Lecture 20: On-Chip Interconnect

This course covers the design and implementation of computer systems from the perspective of the hardware software interface. The purpose of this course is for students to understand the relationship between the operating system, software, and computer architecture. Students that complete the course will have learned operating system fundamentals, computer architecture fundamentals, compilation to hardware abstractions, and how software actually executes from the perspective of the hardware software/boundary. The course will focus especially on understanding the relationships between software and hardware, and how those relationships influence the design of a computer system's software and hardware. The course will convey these topics through a series of practical, implementation-oriented lab assignments.

Credit: Lightly updated from Phillip Gibbons and Omar Multu, 15-740-F18

On-chip Interconnect

On-chip interconnect in a multicore



Where is interconnect used?

To connect components

Processor-to-processor

Processor-to-cache

Cache-to-cache

Cache-to-memory

I/O-to-memory

Etc.

Why is interconnect important?

Affects **scalability** of the system

- How large a system can you build?
- How easily can you add more processors/caches?

Affects performance & energy efficiency

- How fast can processors, caches, memories communicate? (longer than cache access)
- How much energy is spent on communication? (10-35%)

Interconnect basics

Topology

- How switches are wired to each other
- Affects routing, reliability, throughput, latency, cost

Routing (algorithm)

- How does a message get from source to destination?
- Static vs adaptive

Buffering and flow control

- What do we store within the network? (Packets, headers, …?)
- How do we throttle when oversubscribed?
- Tightly coupled with routing

Interconnect topologies

Bus (simplest)

Point-to-point (ideal and most costly)

Crossbar (less costly)

Ring

Mesh

Tree

Omega

Hypercube

Torus

Butterfly

…

Interconnect metrics

Cost (area)

Latency (hops, cycles, nanoseconds)

Contention

Energy

Bandwidth (“bisection” b/w)

End-to-end system performance

Bus

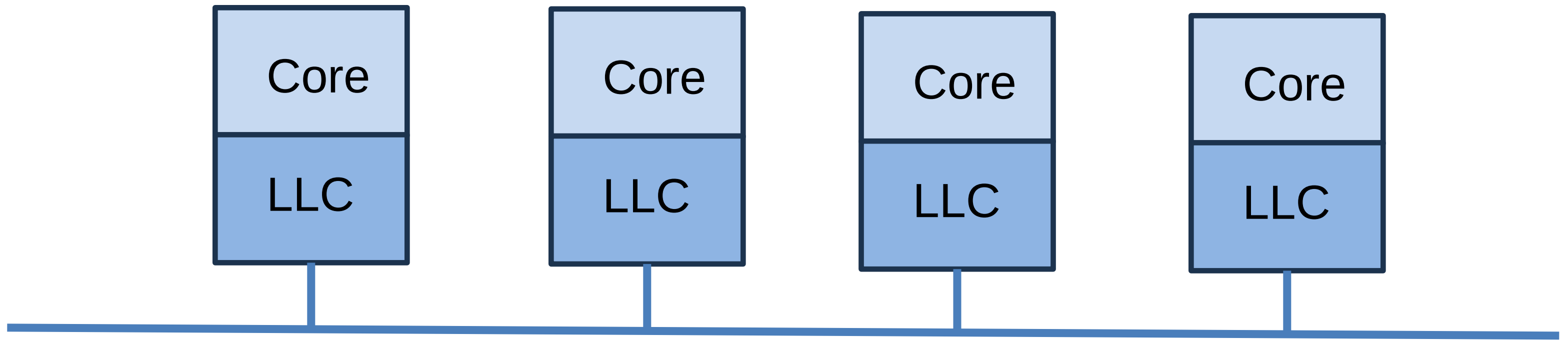
+ Simple

+ Cost-effective for small number of nodes

+ Easy to implement coherence (global broadcast)

- Poor scalability (electrical limitations)

- High contention



Point-to-point

Every node connected directly to every other

+ Lowest contention

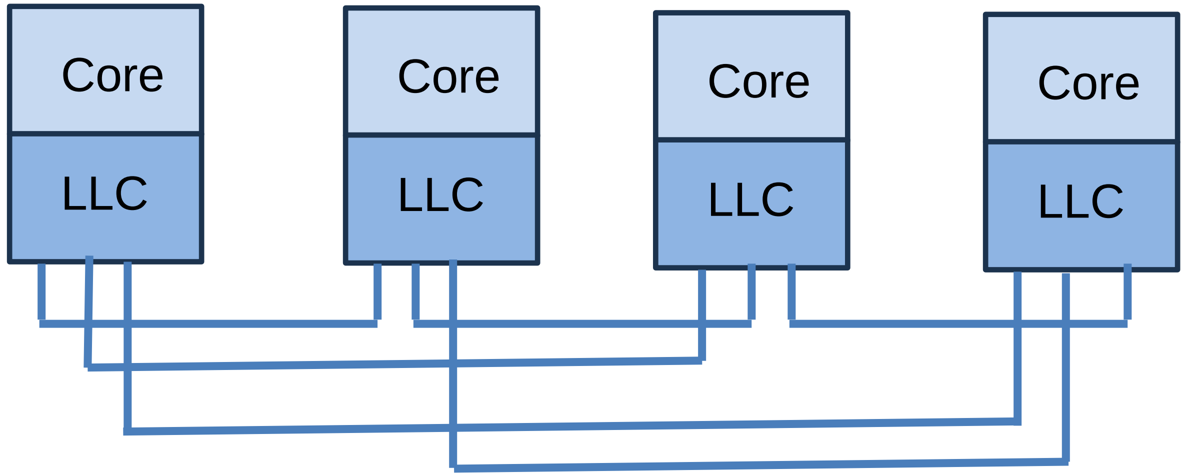
+ Lowest latency (maybe—wire length, wasted area)

+ Ideal except for cost

- Highest cost
  - $O(N^2)$ links

- Not scalable

- Physical layout??



Crossbar

Every node connected to every other, but only one at a time

Concurrent communication to different destinations

Good with few nodes

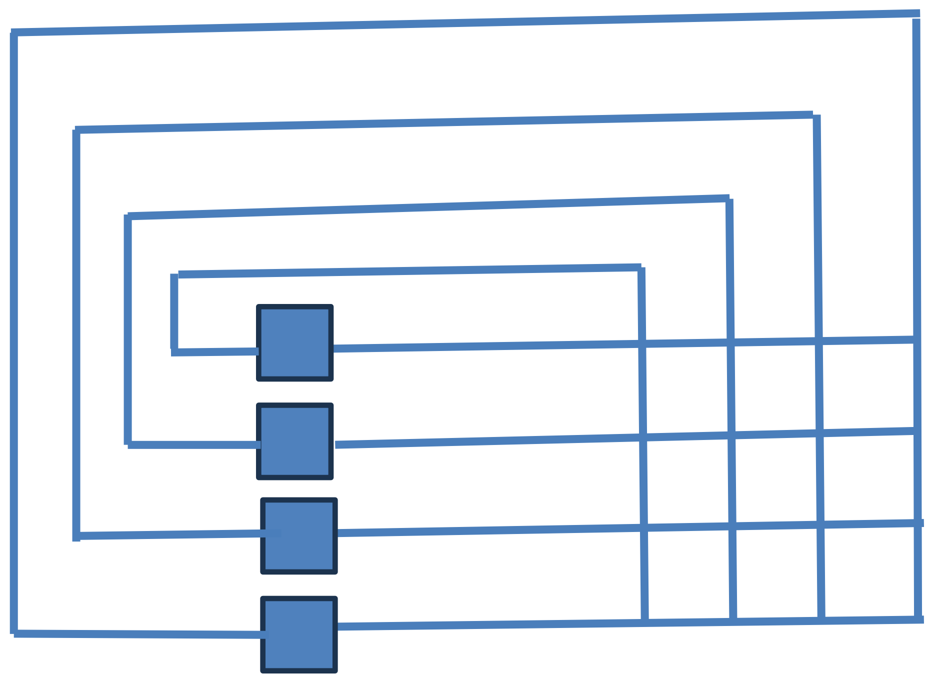
+ Low latency & high throughput

- Expensive

- Doesn’t scale -- $O(N^2)$ switches

- Difficult to arbitrate with many nodes

Used in many designs (e.g., Sun UltraSPARC T1)



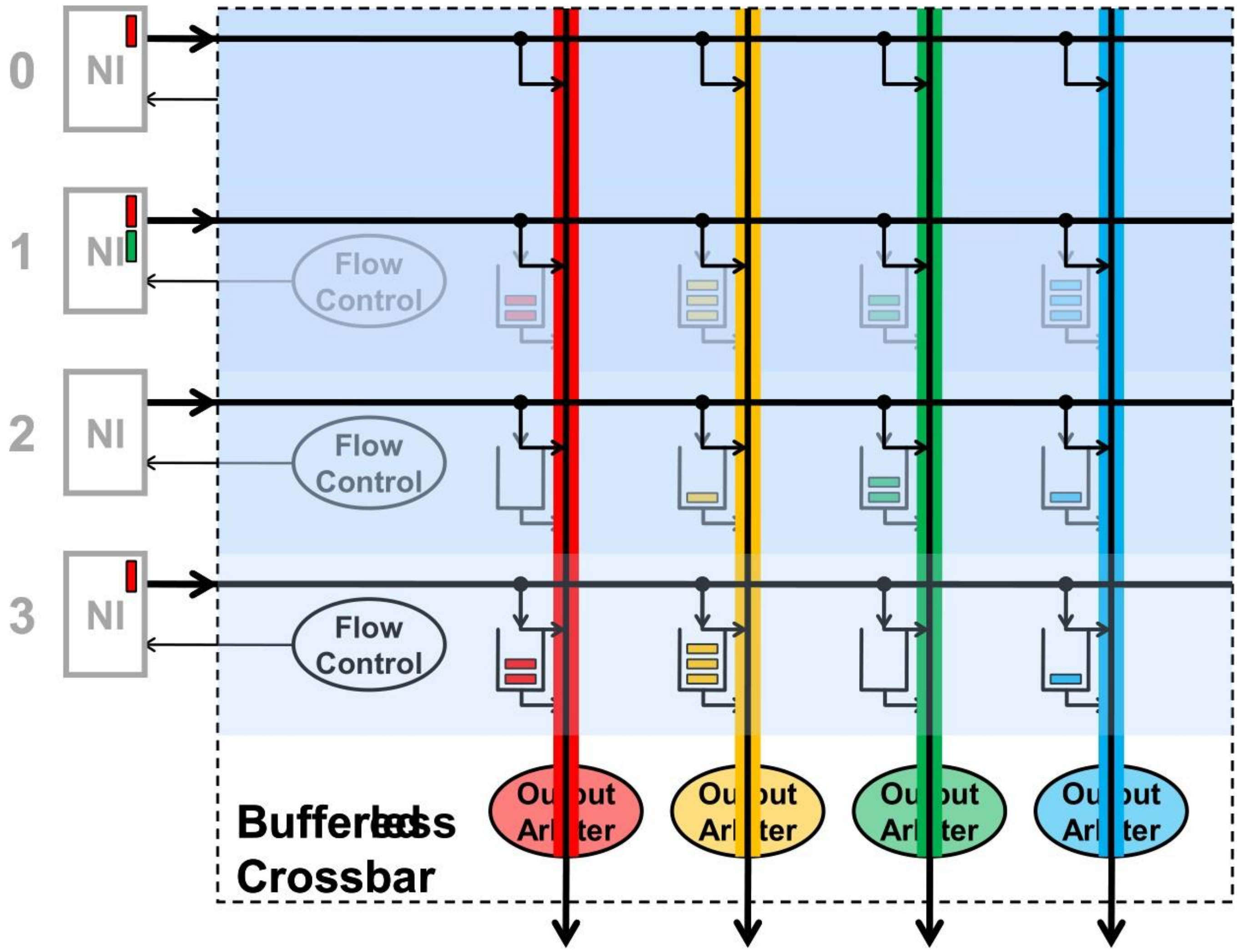
Buffered crossbar

+ Simpler arbitration & scheduling

+ Efficient support for variable sized packets

- Requires $O(N^2)$ buffers

Can we scale the interconnect without contention?

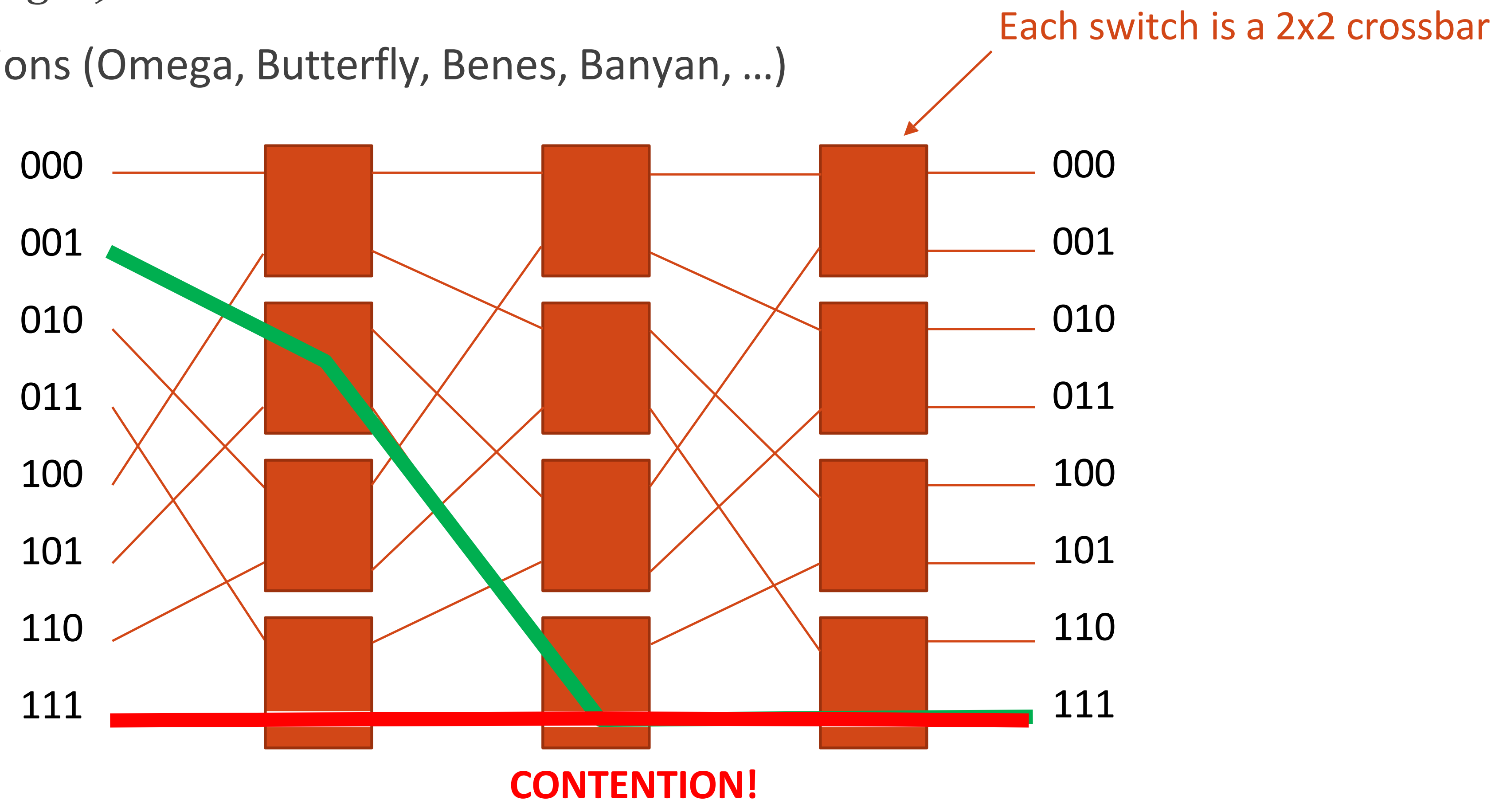


Multistage networks

Idea: $\log N$ switches between nodes

+ Cost $O(N \log N)$

Many variations (Omega, Butterfly, Benes, Banyan, ...)

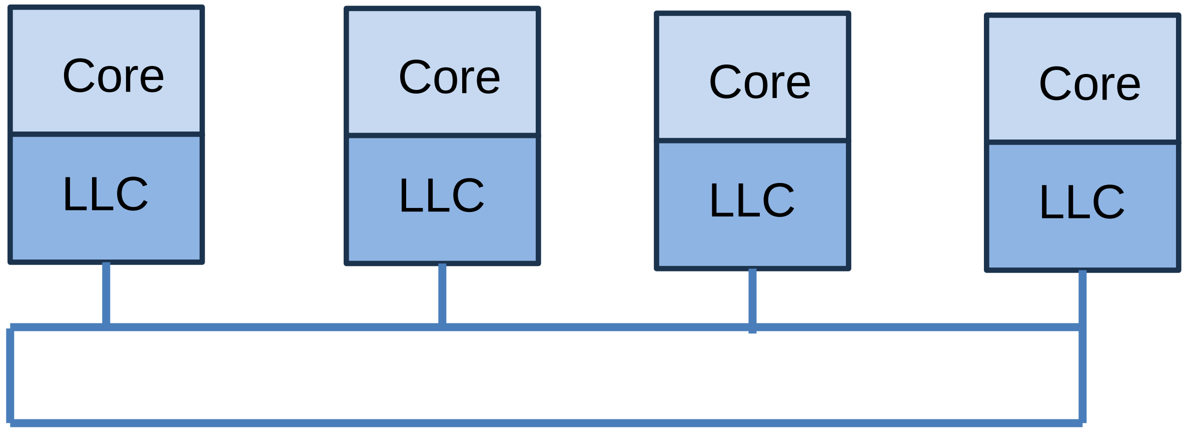


Handling contention

Two packets try to use same link at the same time

What do you do?

- Buffer one
- Drop one
- Misroute one (deflection)



Let's assume buffering for now

Ring

Unidirectional or bidirectional

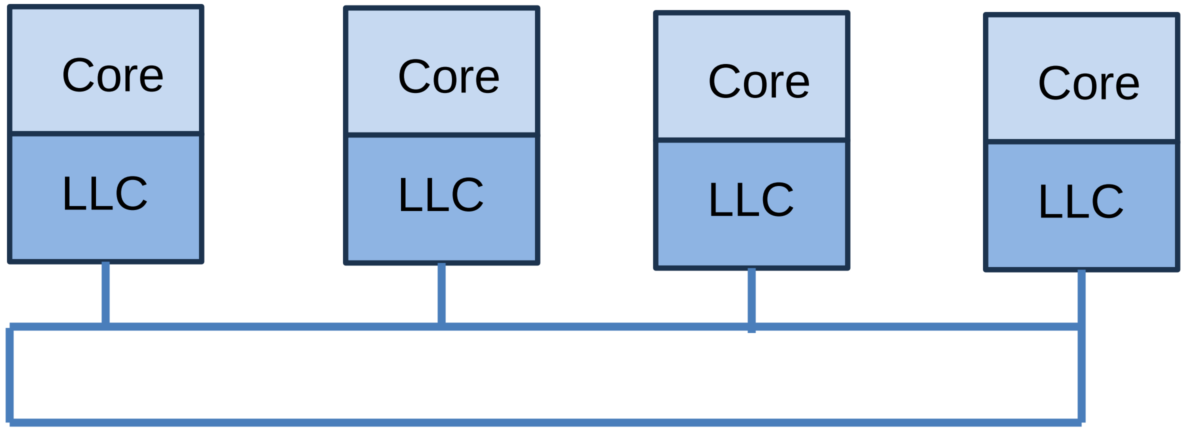
+ Cheap $O(N)$ switches

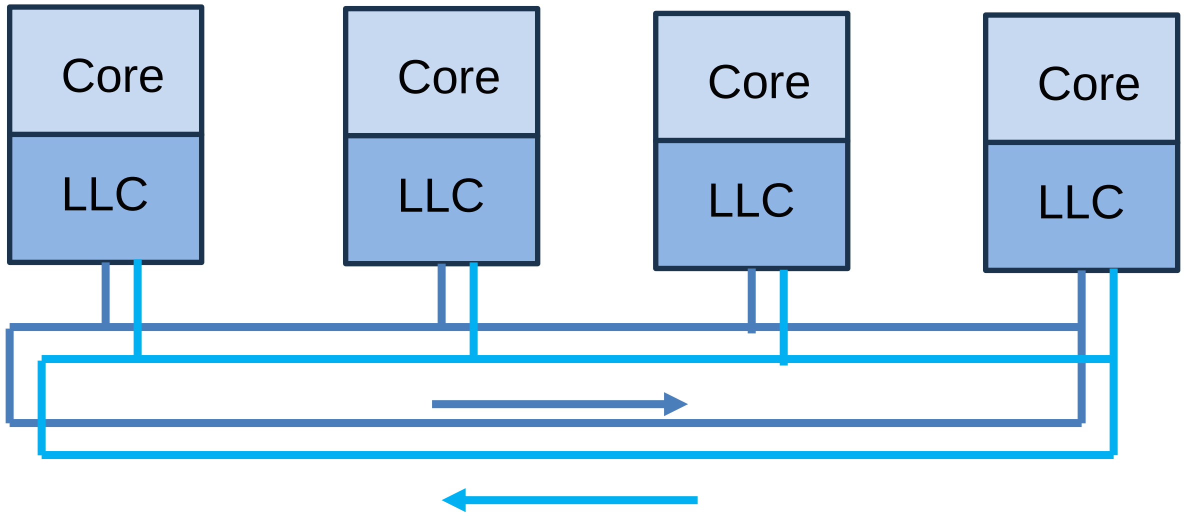
+ Simple switches ➔ Low hop latency

- High latency $O(N)$

- Not scalable; **bisection bandwidth** is constant

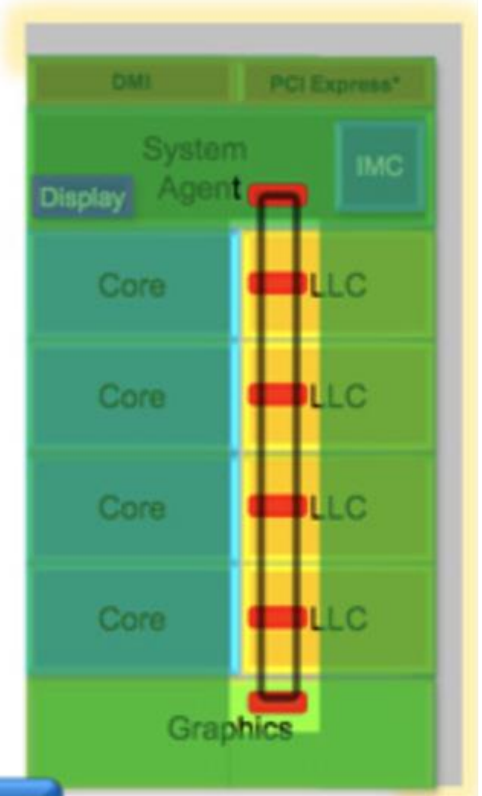
Used in many commercial systems today; recently Intel switched to “ring of rings” topology





Scalable Ring On-die Interconnect

- **Ring-based** interconnect between Cores, Graphics, Last Level Cache (LLC) and System Agent domain
- Composed of **4 rings**
  - 32 Byte *Data* ring, *Request* ring, *Acknowledge* ring and *Snoop* ring
  - Fully pipelined at core frequency/voltage: bandwidth, latency and power scale with cores
- Massive ring **wire routing** runs over the LLC with no area impact
- Access on ring always picks the **shortest path** – minimize latency
- **Distributed arbitration**, sophisticated ring protocol to handle coherency, ordering, and core interface
- **Scalable to servers** with large number of processors



High Bandwidth, Low Latency, Modular

IDF2010

Credit: https://www.anandtech.com/show/3922/intels-sandy-bridge-architecture-exposed/4

2D Mesh

+ $O(N)$ cost

+ $O(\sqrt{N})$ average latency

+ Natural physical layout

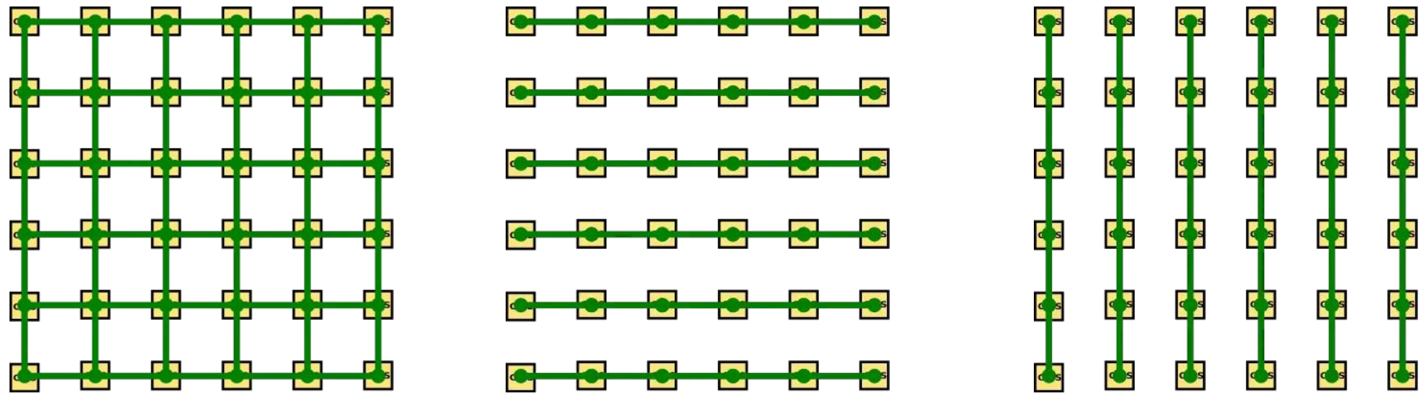
+ Path diversity: Many routes between most sources & destinations
  - Potentially lower contention

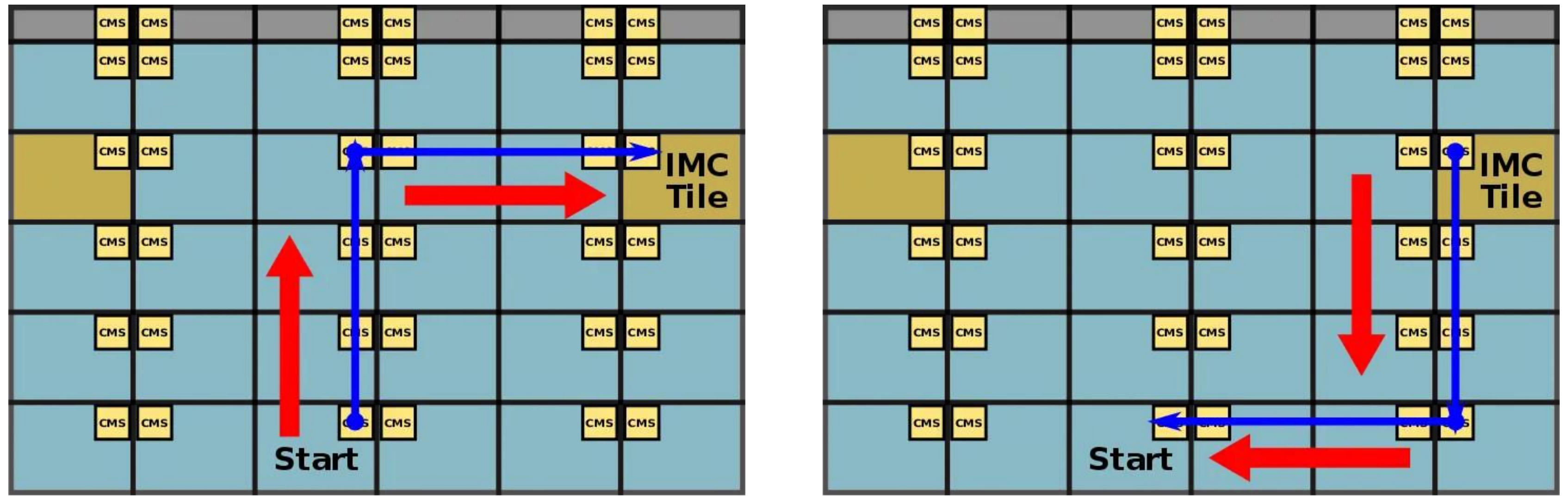
+ Decent bisection bandwidth

- More complex routers ➔ Higher hop latency

Used in Tilera 100-core chip & most research prototypes

Intel Xeon Phi / Skylake Server

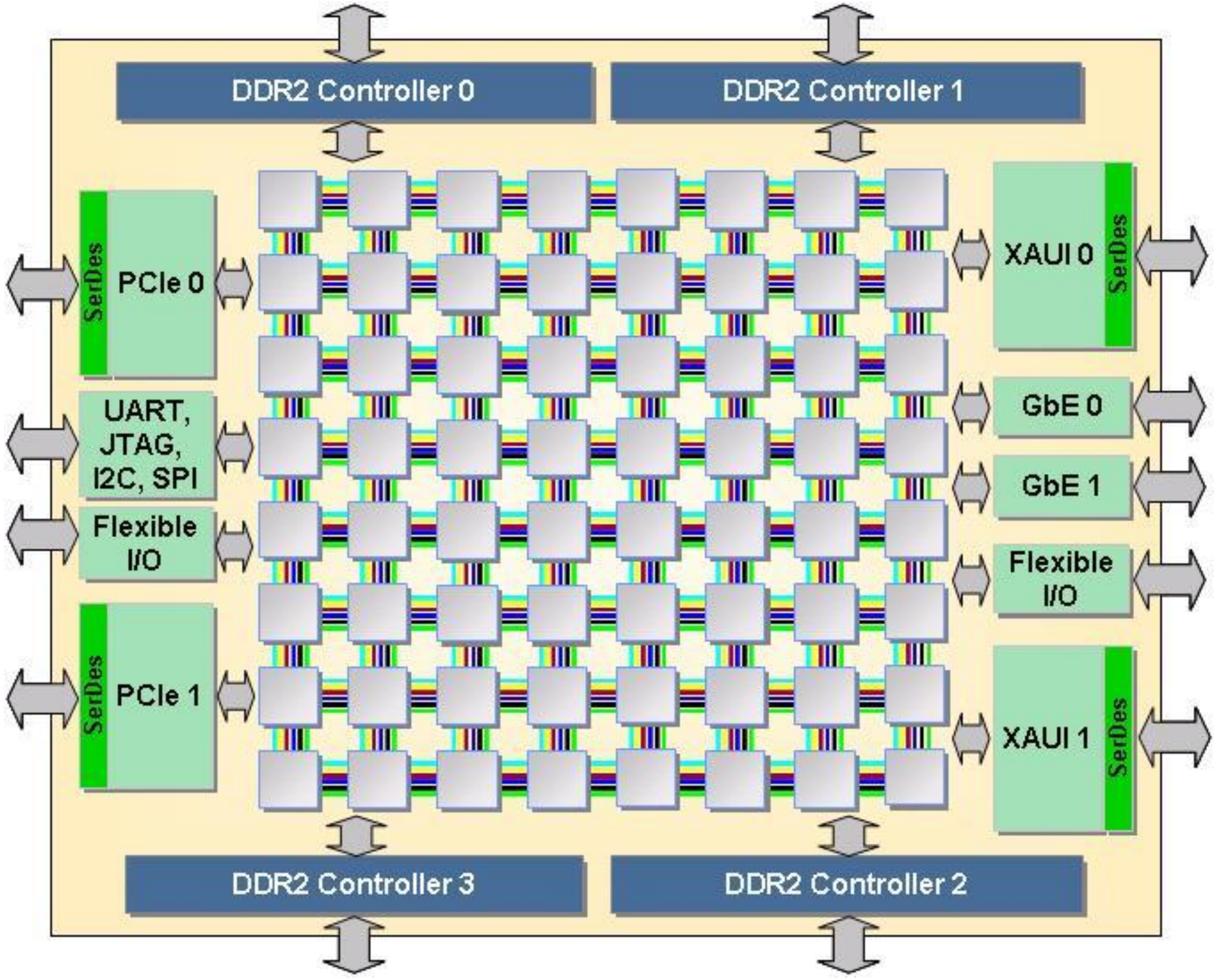




- Vertical and horizontal half rings
- Route first vertically, then horizontally
- Return path may be different

Credit: https://en.wikichip.org/wiki/intel/mesh_interconnect_architecture

Tilera Pro Mesh Interconnect



Credit: https://en.wikipedia.org/wiki/TILEPro64

Trees

Planar, hierarchical topology

+ $O(\log N)$ latency

+ $O(N)$ cost

+ Easy to layout

- Root is bottleneck; constant bisection bandwidth

Trees common for local communication; e.g., banks of single cache

Bisection bandwidth mitigated by “fat trees”, at add’l cost

- Idea: Make bandwidth at the top “fatter”
- Idea: Replicate root node, randomize routing (Used in Thinking Machines CM-5 circa 1991)

Flow control methods

Circuit switching

Packet switching

- Store and forward
- Virtual cut-through
- Wormhole

Circuit switching

Pre-allocate resources across multiple switches

Requires “probe” ahead of message

+ No need for buffering

+ No contention (after circuit established)

+ Handles arbitrary message sizes

- Low link utilization

- Delay to set up circuit

Store and forward

Copy entire packet between switches

+ Simple

- High per-packet latency

- Requires big buffers / small messages

Virtual cut-through

Start forwarding as soon as header is received

+ Dramatic reduction in latency vs store and forward

- Still buffers entire message in worst case: requires large buffers / small messages

Wormhole

Break packets into much smaller “flits”

Pipeline delivery: Each flit (“flow control digit”) follows its predecessor through network

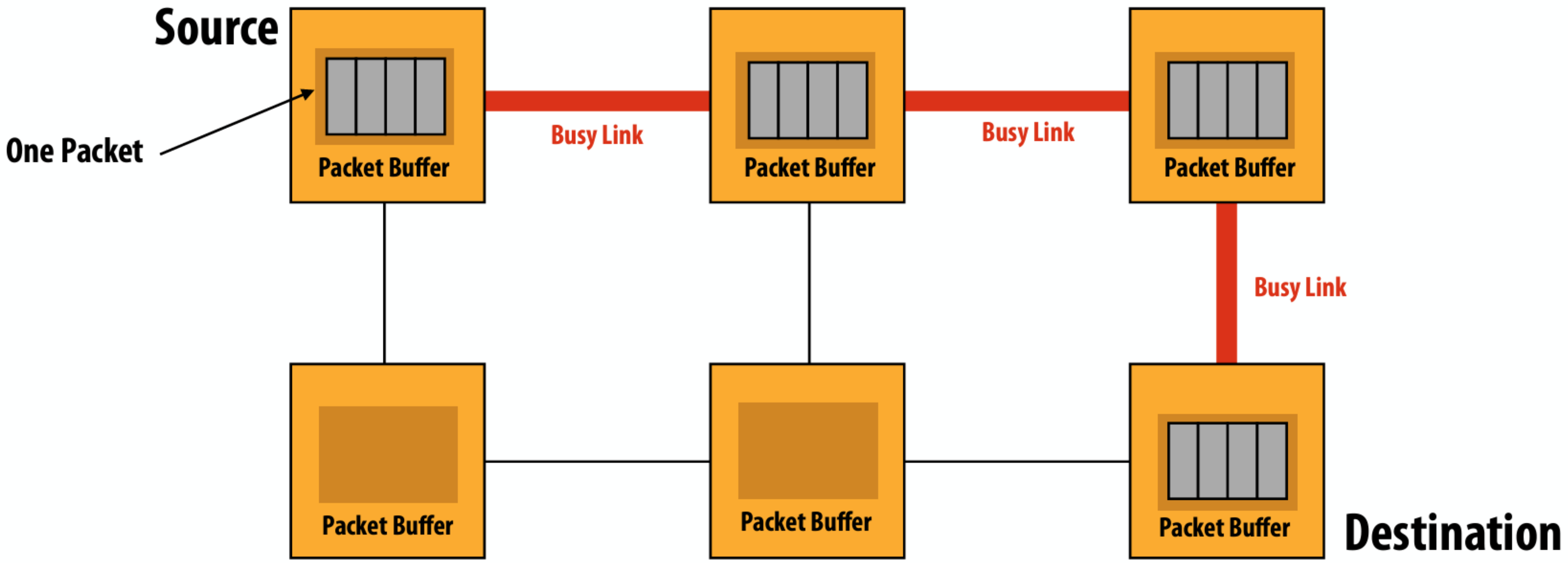
If head is blocked, rest of packet waits in earlier switches

+ No large buffering in network

+ Latency independent of distance for large messages

- Head-of-line blocking

Wormhole



Credit: 15-418, Spring 2023, Lecture 14

Routing algorithms

Deterministic: Simplest, high contention

- Dimension-order (e.g., XY)
- Deadlock-free

Oblivious: Simple, mitigates contention

- Valiant’s algorithm: Route deterministically via a random node
- Balances network load, adds latency
- Optimization: Use only at high load

Adaptive: Complex, most efficient

- Minimal adaptive: Always route closer to destination on least-contended port
- Fully adaptive: “Misroute” packets to optimize overall network load
  - Must guard against livelock
  - How to coordinate overall network state?