TU WIEN Informatics

Advanced Computer Architecture

F2 Networks-on-Chip (NoCs)

Daniel Mueller-Gritschneder

Sources

- **Principles and Practices of Interconnection Networks**
  Authors: William James Dally, Brian Patrick Towles
  ISBN: 978-0-08-049780-8

- Slides inspired by the „On-Chip Networks I/II“ (L-15/L-16) lectures of Ryan Lee and Tushar Krishna: http://csg.csail.mit.edu/6.5900/lecnotes.html

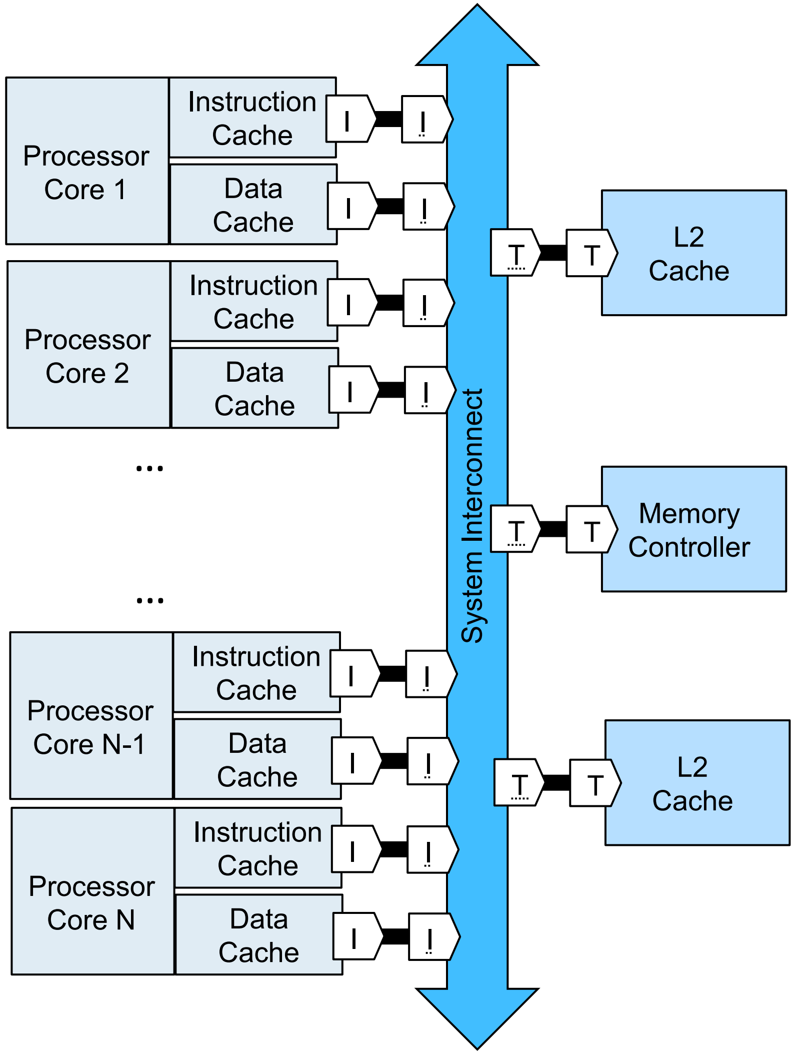
F2.1 Introduction to NoCs

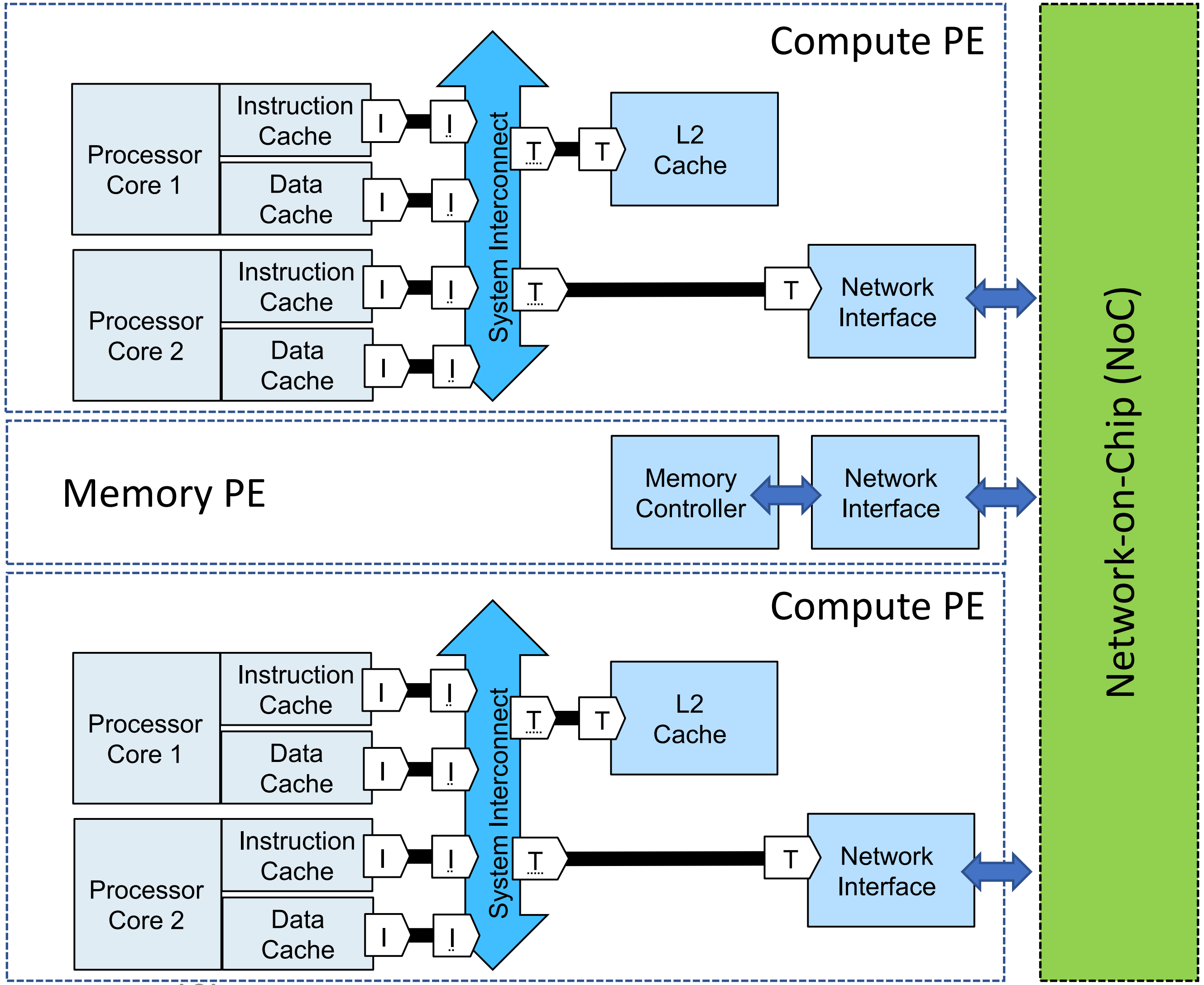
Motivation

- Need for scalability and reduced cost
  - Avoid long interconnects/delays caused by increased system complexity
  - Reduce wiring overhead caused by increasing number of system components
- Performance demands
  - Goal: high bandwidth and low latency
  - Concurrent communication required due to increased traffic
- Solution: Network-on-Chip (NoC)
  - Move from bus to network (small-scale networks on chip-/system-level)
    - Larger-scale networks in later lectures
  - Broadcast can be avoided, but still possible via multiple messages (when required)
  - Serialization achievable, e.g., by forcing the same path or via sequence numbers

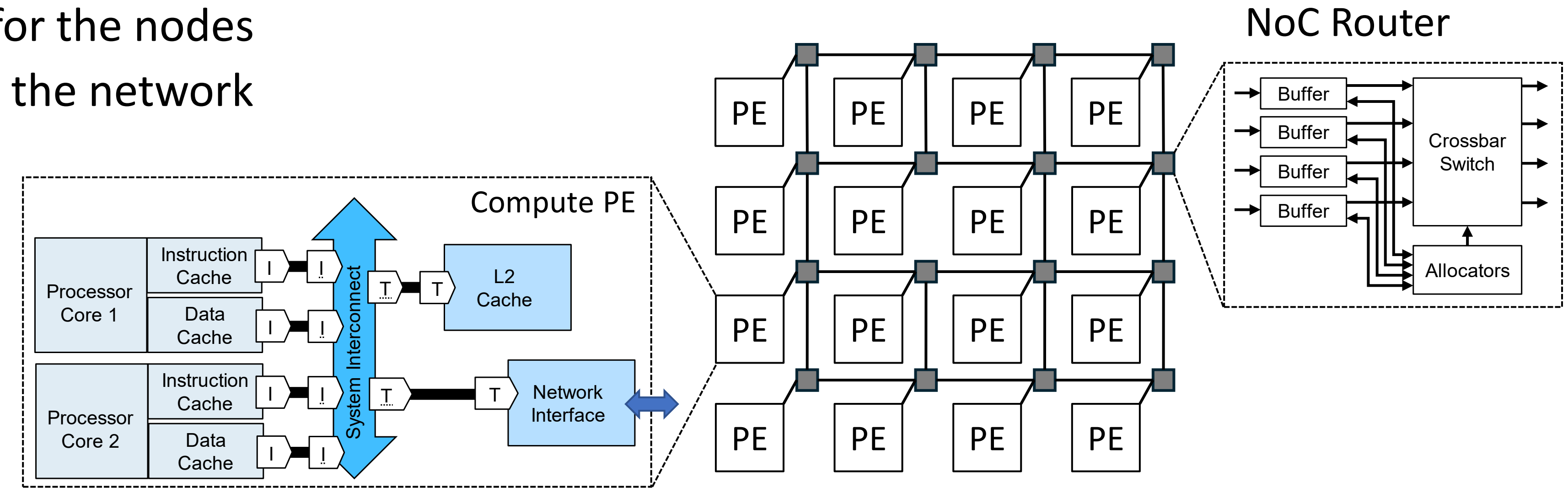
Motivation: Scalability

- Scalability: How to connect hundreds of processor cores / memory interfaces?





- Objective: Connect nodes with each other via routers and wires, so that messages can be sent from source to destination
- Building blocks:
  - Node: any component, e.g., processor, memory, or a combination of them
  - Network interface: module connecting a node to the network
  - Router: forwards data from inputs to outputs (network interfaces or other routers)
  - Link: physical set of wires, e.g., connecting two routers
  - Channel: logical connection between routers
  - Message: unit of transfer for the nodes
  - Packet: unit of transfer for the network



Design

- Topology: What is the connection pattern of the nodes?
- Routing: Which path should a message take?
- Flow control: Which network resources are granted to a message over time?

- Traffic analogy
  - Topology: defines roadmap, i.e., streets and intersections
  - Routing: steering of the car, i.e., where to turn at each intersection
  - Flow control: traffic light control, i.e., when a car can advance over the next part of the road

F2.2 NoC Topologies

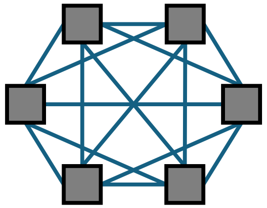
Topology

- Topology: arrangement of nodes and channels
  - Determines e.g., number of hops, number of alternative paths, cost

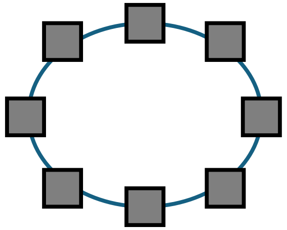
- Properties for comparison
  - Degree: number of links at each node
  - Distance: number of links in the shortest route
  - Diameter: maximum distance between any two nodes
  - Bisection bandwidth: available bandwidth from one partition to the other, when cutting the network into two equal parts (minimum for multiple possible cuts)

Topology

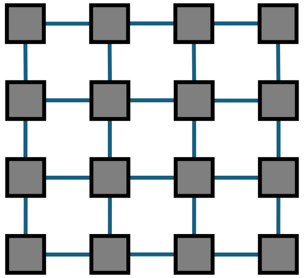
- ***Direct networks:*** each terminal node is associated with a router; routers are sources/sinks *and* switches for traffic from other nodes



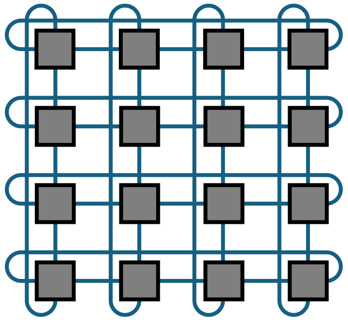
Fully Connected



Ring

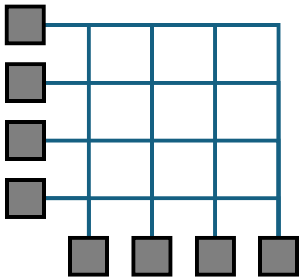


Mesh

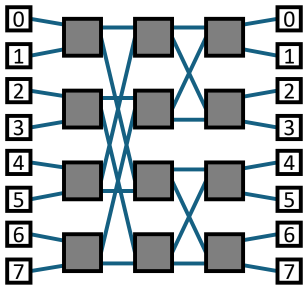


Torus

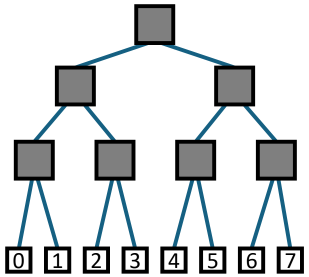
- ***Indirect networks:*** terminal nodes are connected via intermediate stages of switch nodes; terminal nodes are sources/sinks, intermediate nodes only switch traffic



Crossbar



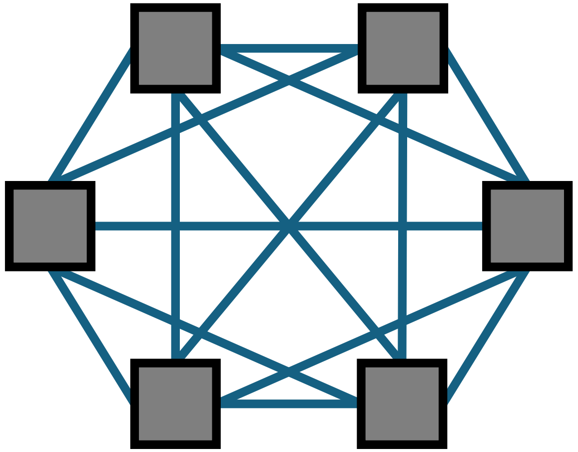
Butterfly



Tree

Fully Connected Networks

- Every node connected to every other node with a direct link
- *N* nodes, *N·(N-1)/2* links
- Degree: *N-1*
- Diameter: *1*
- Bisection width: $\lfloor N/2 \rfloor \cdot \lceil N/2 \rceil$



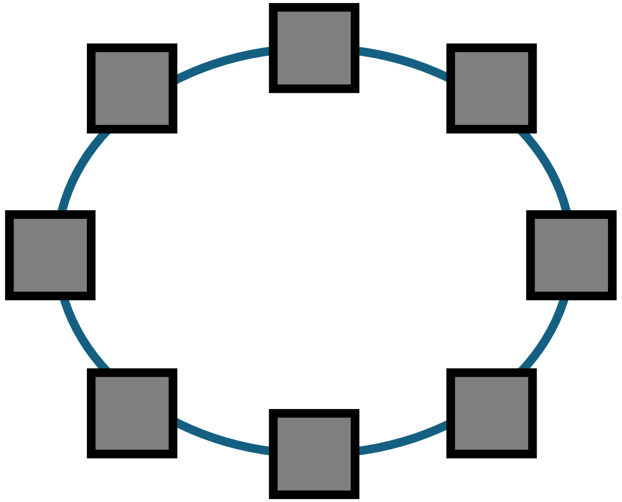
Fully Connected

- Pros: high fault tolerance, low contention, low latency
- Cons: high costs for large N, limited scalability

Ring (*k*-ary 1-cube)

- Each node connected to two other nodes
- *N* nodes, *N* links
- Degree: *2*
- Diameter: $\lfloor N/2 \rfloor$
- Bisection width: *2*

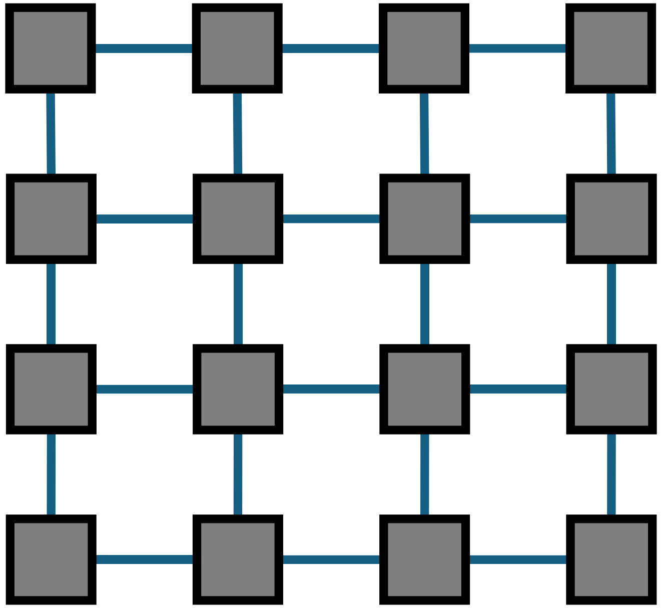
- Pros: simple, low link costs
- Cons: high latency for large N, limited path diversity



Ring

Mesh

- k-ary n-cube: $N=k^n$ nodes in a regular n-dimensional grid
  - k nodes in each dimension
  - Links between nearest neighbors
- For n=2 (i.e., $k \times k$ grids)
  - $N=k^2$ nodes, $2k \cdot (k-1)$ links
  - Degree: *4*
  - Diameter: *2k-2*
  - Bisection width: *k*

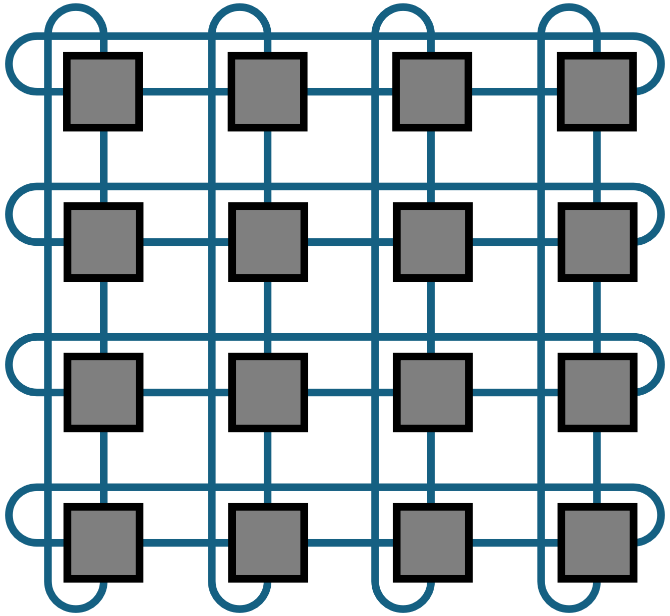


Mesh
(here: 4-ary 2-cube)

- Pros: path diversity, regular and equal-length links
- Cons: large diameter, asymmetric (higher demand for center links)

Torus

- *k*-ary *n*-cube: $N=k^n$ nodes in a regular *n*-dimensional grid
  - *k* nodes in each dimension
  - Links between nearest neighbors, adds wrap-around links at the edges compared to mesh
- For n=2 (i.e., $k \times k$ grids)
  - $N=k^2$ nodes, *2N* links
  - Degree: *4*
  - Diameter: *k*
  - Bisection width: *2k*



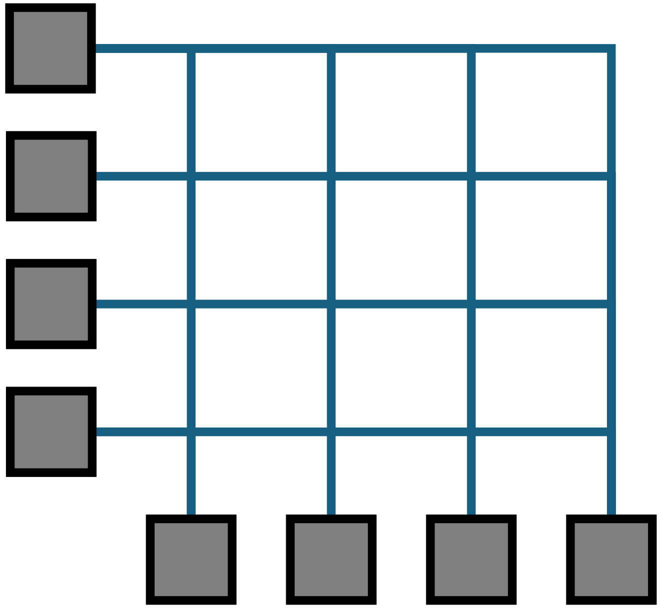
Torus

- Pros: avoids asymmetry and improves path diversity compared to mesh
- Cons: unequal link lengths and higher cost compared to mesh

Crossbar

- Connects n inputs to m outputs via $n \times m$ switches
- Switches enable concurrent communication between disjoint input/output pairs without blocking
- $N = n \cdot m$ nodes, $n \cdot m$ links
- Diameter: *1*

- Pros: non-blocking, latency (for small n, m)
- Cons: high cost, limited scalability

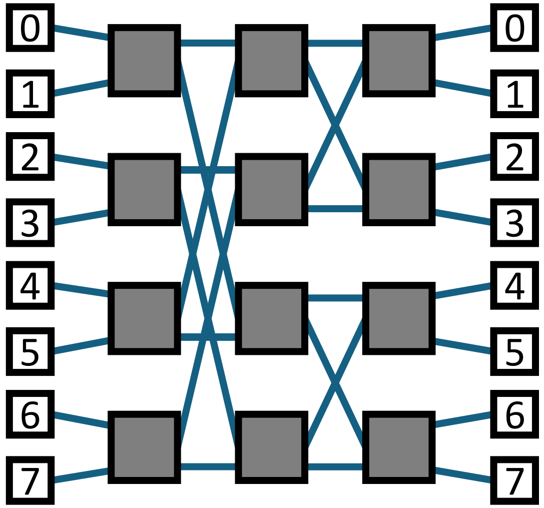


Crossbar

Butterfly

- *k*-ary *n*-flies: k^n nodes connected via *n* stages of k^{n-1} intermediate $k \times k$ switches
  - k: switch degree
  - n: number of stages of switches

- Pros: lower cost compared to crossbar
- Cons: blocking, lack of path diversity, locality not exploitable

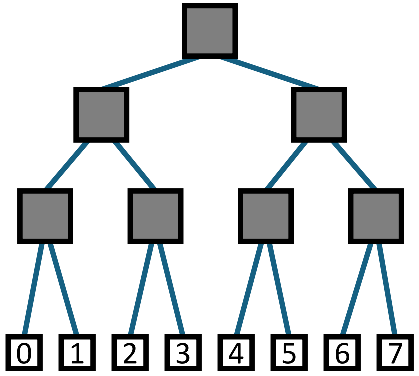


Butterfly
(2-ary 3-fly)

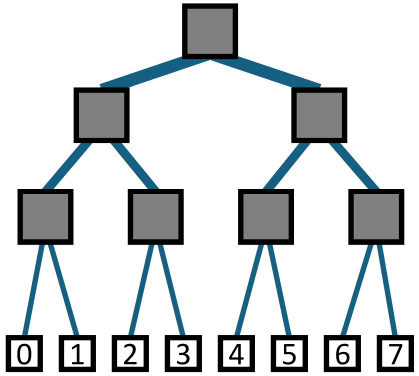
Trees

- k-ary tree with N nodes and $log_k N$ stages
- Nodes are the leaves of the tree, switches at intermediate stages
- Messages are sent up to common ancestor, then sent down to destination

- Pro: simple, cheap
- Cons: Bottleneck towards root
  - Alternative: Fat tree, where links between switches closer to the root are increased



Tree

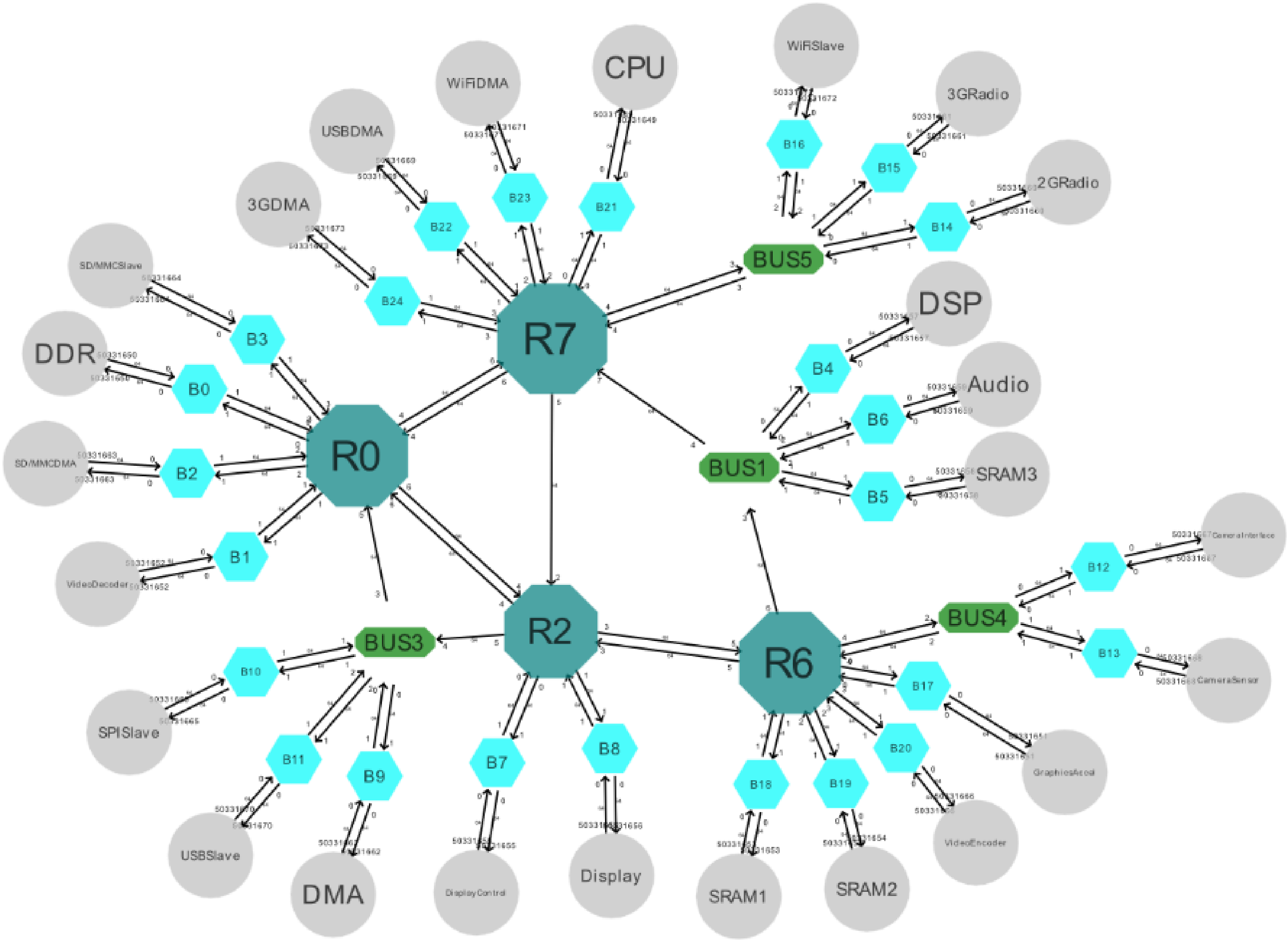


Fat tree

Application-Specific Network-on-Chip Architectures

- Custom tailored NoC topology for chips with very unbalanced traffic demand for different PEs

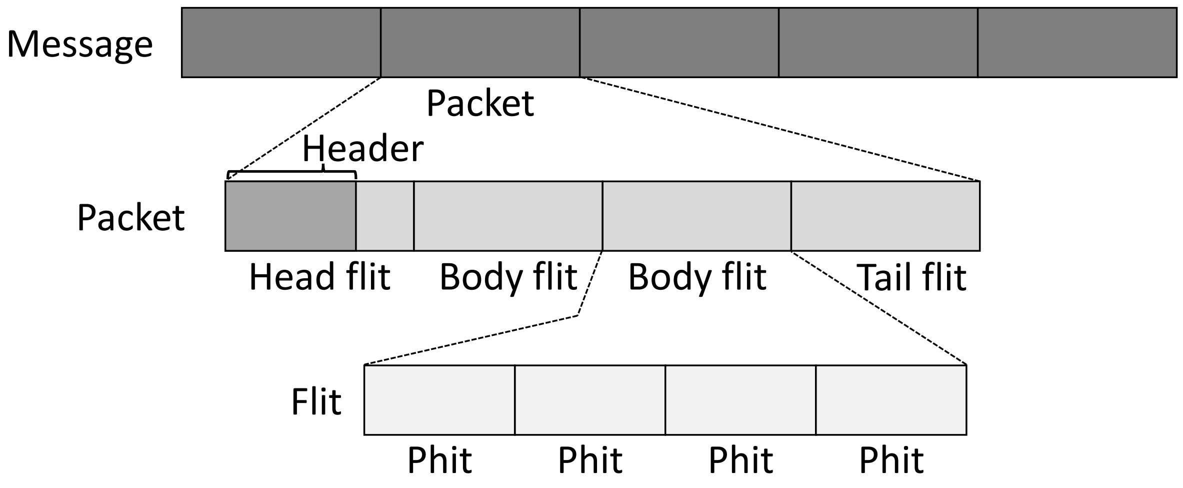
- Example: NoC for a 3G Modem Chip (2014)



F2.3 NoC Messages

Messages

- Message: logically continuous group of bits, may be arbitrarily long
- Packet: basic unit of routing and sequencing, restricted maximum length
  - Consists of header + segment of a message
- Flit (flow control digit): basic unit of bandwidth and storage allocation
  - Contain no separate routing/sequencing information and therefore follow the same path in-order
  - Subdivision allows for low overhead (large packets) and fine-grained resource utilization (small flits)
- Phit (physical transfer digit): information transferred over a channel in a single clock cycle



Flow Control vs. Routing

- Flow control: Allocates resources (channels, control state, buffers) to packets
  - Alternative view: resolve contention during packet transmission
  - Contention: What happens if two packets want to use the same channel at the same time?

- Routing: Selects the path a packet takes from source to destination
  - Determines how well the potential of the given topology is exploited
  - Should balance load across network channels

F2.4 NoC Flow Control

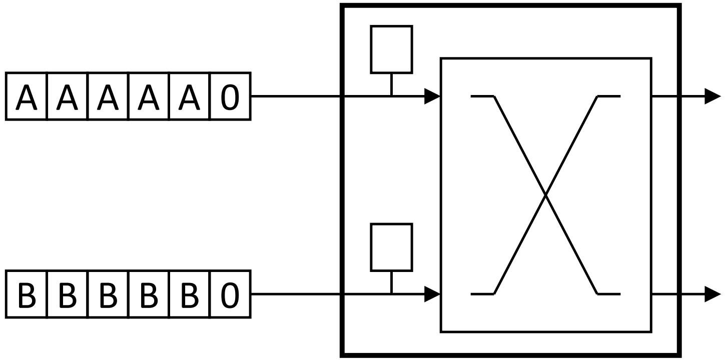
Flow Control

- Bufferless
  - Dropping
  - Misrouting
  - Circuit switching
- Buffered
  - Store-and-forward
  - Cut-through
  - Wormhole
  - Virtual channel

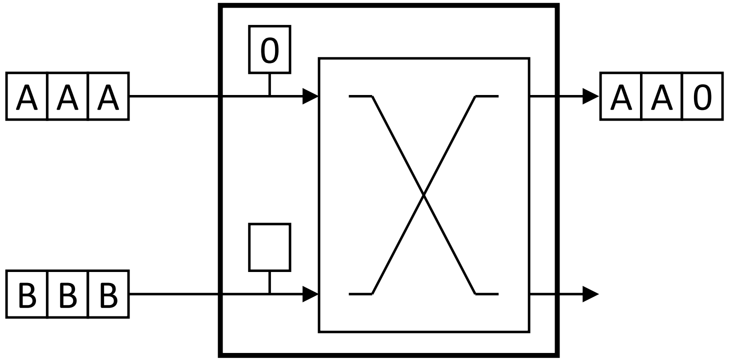
Bufferless Flow Control: Dropping

- Competing packets: No buffers available, therefore drop “losing” packets, “winning” packet is allowed to proceed
- Example:

Two packets A and B arriving, both requesting channel 0



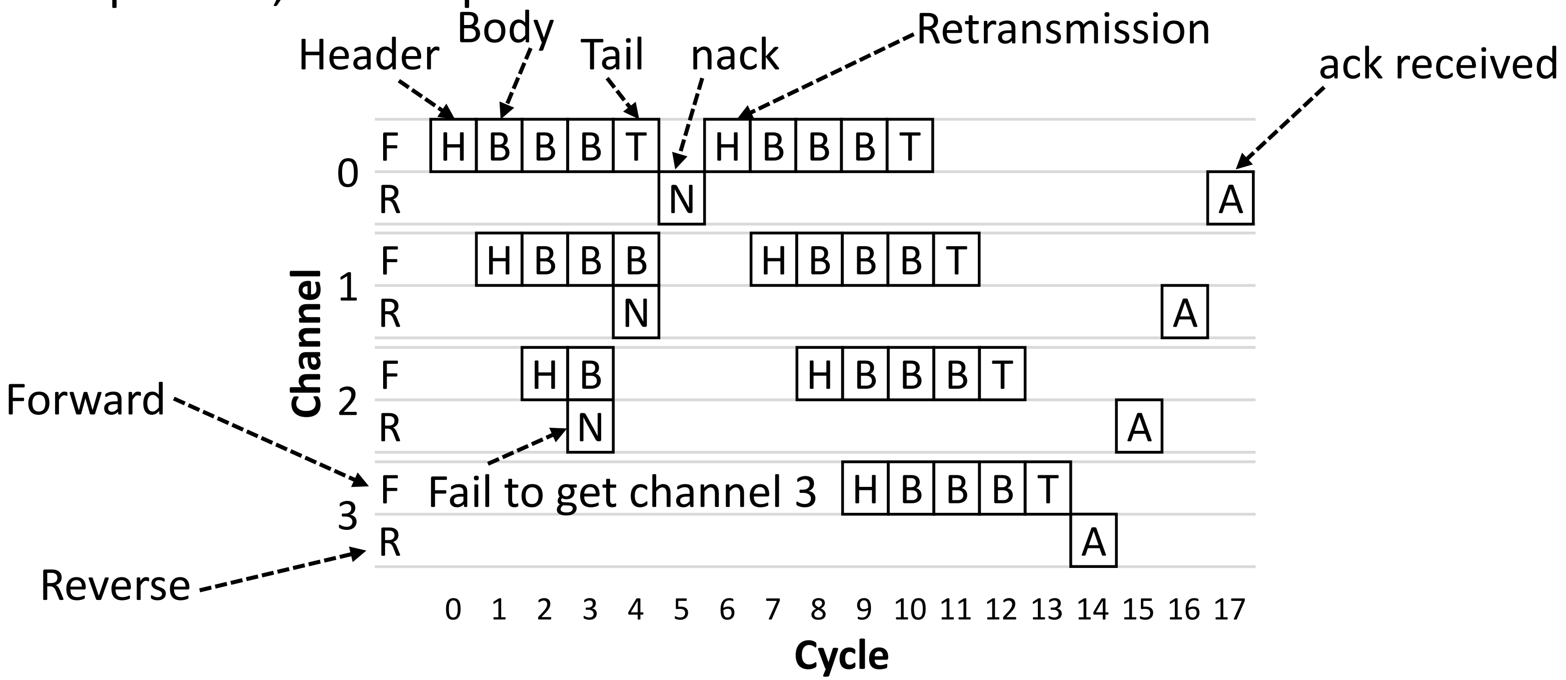
Packet A “wins”, B is dropped and must be retransmitted from source



  - Complete effort already invested in packet B is lost
  - Source needs to be informed to about successful transmission or need for retransmission

Bufferless Flow Control: Dropping

- Time-space diagram with negative acknowledgements (nacks)
  - Example: five-flit packets, four-hop route

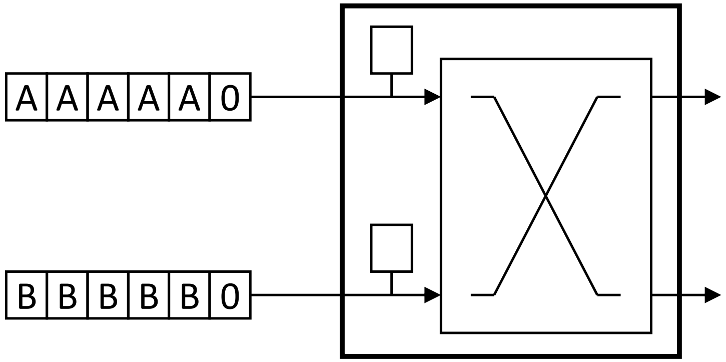


- Alternative: no nacks, resend packet if ack is not received before a timeout

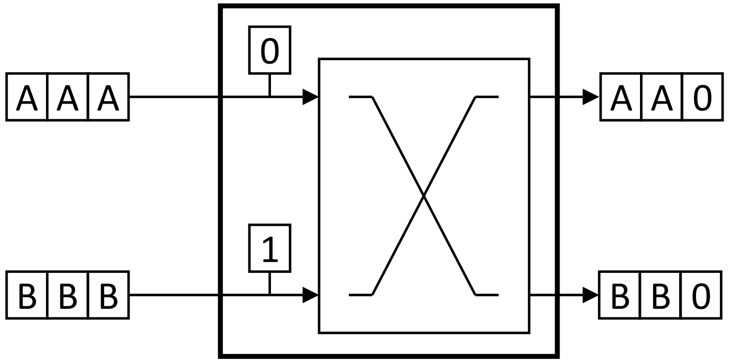
- Dropping: simple, wastes resources

- Competing packets: No buffers available, therefore misroute “losing” packets, “winning” packet gets the requested channel
- Example:

Two packets A and B arriving, both requesting channel 0



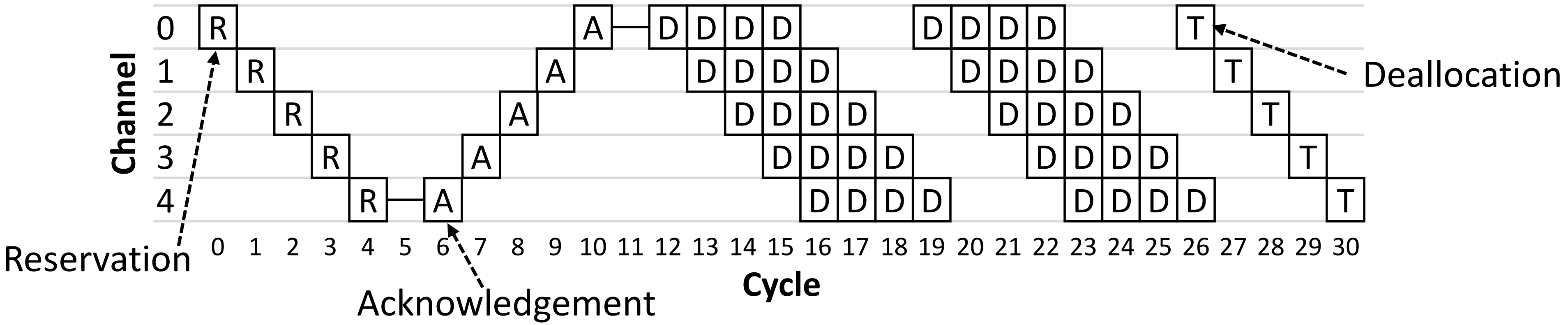
Packet A “wins”, B is misrouted to channel 1



  - Requires sufficient path diversity
  - Routing needs to ensure that packet reaches its destination despite misrouting
- Misrouting: no packet dropping, packets sent in wrong direction, livelock possible (need to guarantee forward progress)

Bufferless Flow Control: Circuit Switching

- First allocate channels to build a circuit from source to destination, then send packets along the circuit, deallocate circuit after packets are sent
- Example: four-flit packets, five-hop route
  - 1. Send request (R) to destination allocating channels along the way
  - 2. Destination returns acknowledgement (A) to source
  - 3. Data flits (D) are sent
  - 4. Tail flit (T) deallocates the channel



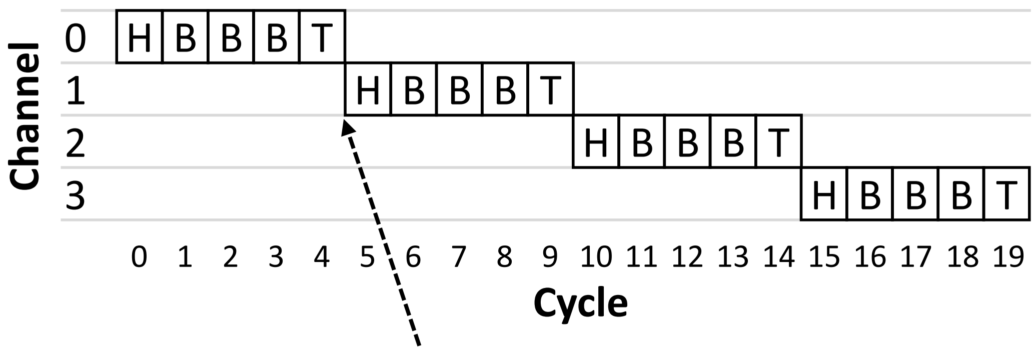
- Circuit switching: simple, high latency, high overhead for circuits with short duration

Buffered Flow Control

- Buffers allow to store data while waiting for the following channel
  - Without buffers data arriving at cycle *i* had to be transmitted at cycle *i+1* (or dropped)
- Flow control now needs to allocate channels *and* buffers
  - Allocation at packet or flit granularity
  - Packet granularity: store-and-forward, cut-through
  - Flit granularity: wormhole

Buffered Flow Control: Store-and-forward (Packet-based)

- Each node waits until packet is received completely before transmission to the next node
- Need to allocate channel and sufficient buffer space for the packet in the next node
- Example: five-flit packet, four-hop route without contention

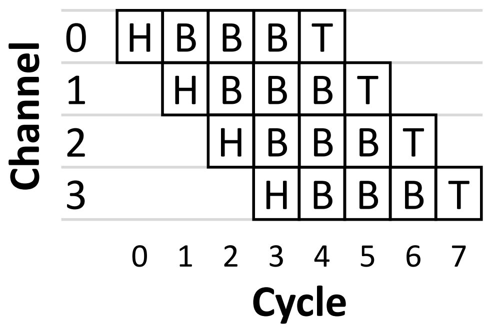


Could also be transmitted later if channel/buffer space is not available

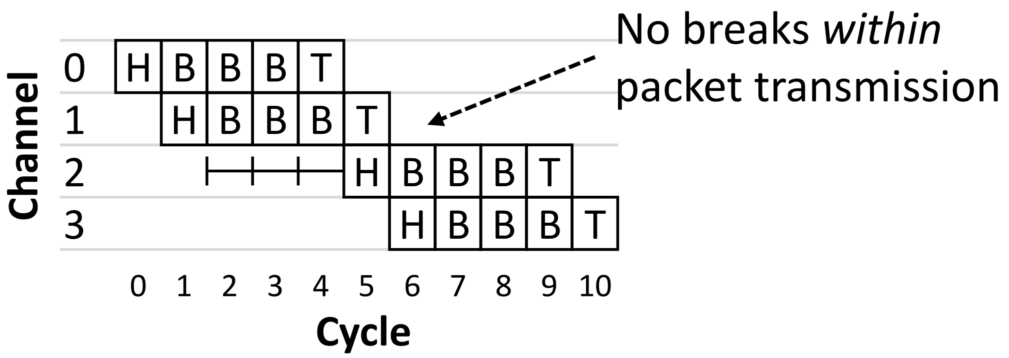
- Store-and-forward: channels not held idle, only small buffers required, high latency due to serialization

Buffered Flow Control: Cut-through (Packet-based)

- Flits are forwarded as soon as they are received *and* the following channel and buffer space is acquired (allocation still at packet granularity)
- Avoids waiting for receiving the complete packet before transmission
- Example: five-flit packet, four-hop route without/with contention



No contention



Three-cycle contention before channel 2

- Cut-through: high channel utilization, low latency, inefficient use of buffer storage and long contention latency due to packet-based allocation

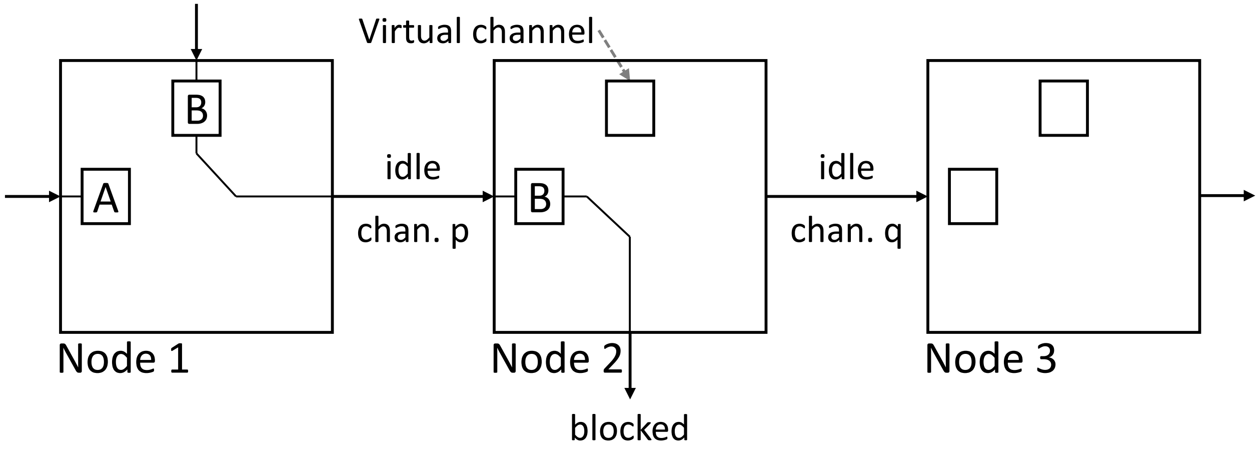
Buffered Flow Control: Wormhole (Flit-based)

- Similar to cut-through, but allocates channels and buffers to flits instead of packets
  - Head flit requests channel state (virt. channel) for the *packet*, buffer for one flit and channel for one flit
  - Body flits use virtual channel to follow head flit, request buffer for one flit and channel for one flit
  - Tail flit treated like body flit, but additionally releases virtual channel
- Blocking might occur as the single virtual channel belongs to a packet, while buffers are allocated to flits
  - Channel set to idle if buffer cannot be acquired (it cannot be used by other packet)
- Wormhole: Saves buffer space, may block a channel mid-packet

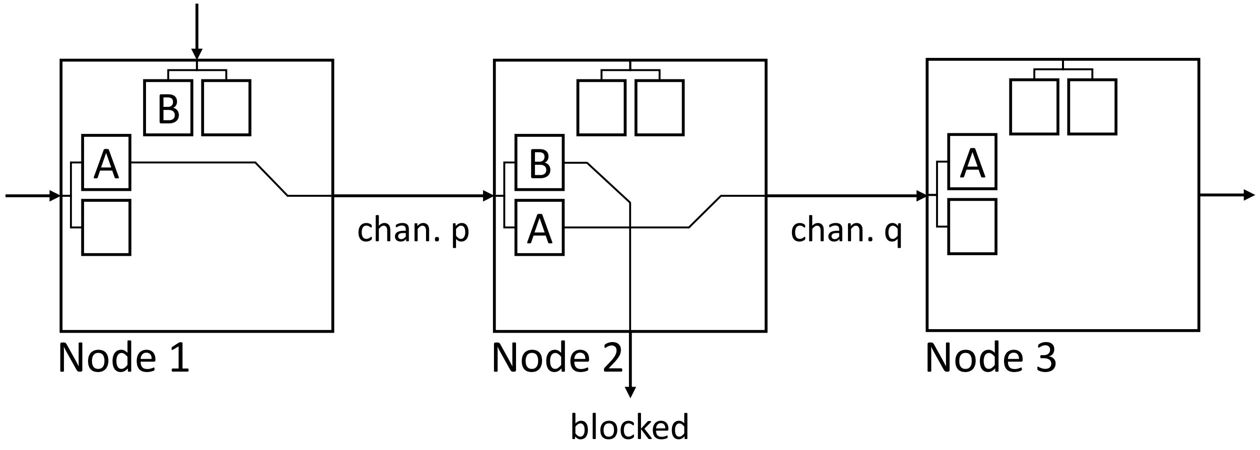
- Improvement: virtual-channel flow control
  - Associate multiple virtual channels (channel state and flit buffers) with single physical channel
  - Other packets can use channel when one packet is blocked
  - Competition for transmitting flits over single physical channel
  - Reduces blocking, more complex routers

Buffered Flow Control: Wormhole vs. Virtual-channel

- Wormhole flow control: When B blocks, channel p and q are idle



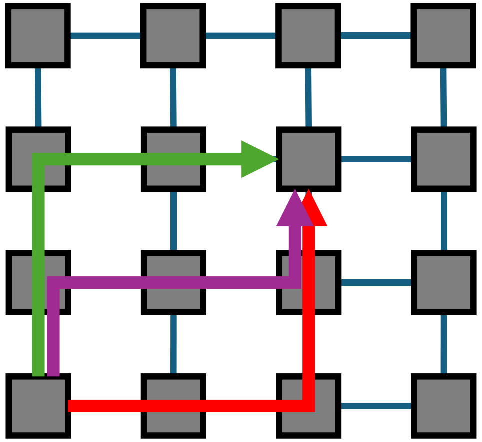
- Virtual-channel flow control: A can use channel p and q using a second virtual channel



F2.5 NoC Routing

Routing

- Selects the path a packet takes from source to destination in a given topology
- Determines how well the potential of the given topology is exploited



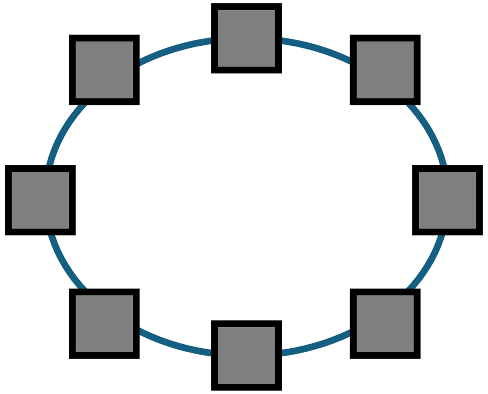
- Balance load across the network channels to avoid hotspots and contention
  - Difficult, particularly with non-uniform traffic patterns causing load misbalances

Routing Algorithms

- Properties
  - Minimal or non-minimal
    - Minimal: select shortest paths
    - Non-minimal: not limited to shortest paths only
  - Oblivious or adaptive
    - Oblivious: select route without considering information about current network state
      - Deterministic: Subset of oblivious; always select same path between source and destination
    - Adaptive: select route based on current network state
- Design aspects
  - Table-based or algorithmic
    - Table-based: Table lookup of the entire route (source-table routing) or at each node along the route (node-table routing)
    - Algorithmic: Compute route using an algorithm usually implemented via combinational logic
  - Deadlocks
    - Situations where packets cannot make progress as they are waiting on one another to release resources

Routing Example

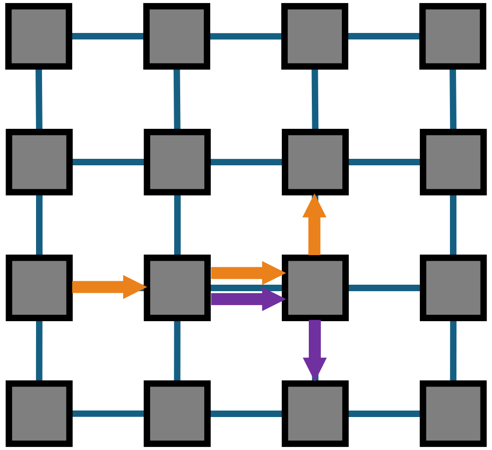
- Routing decision in ring network: clockwise or counter-clockwise?



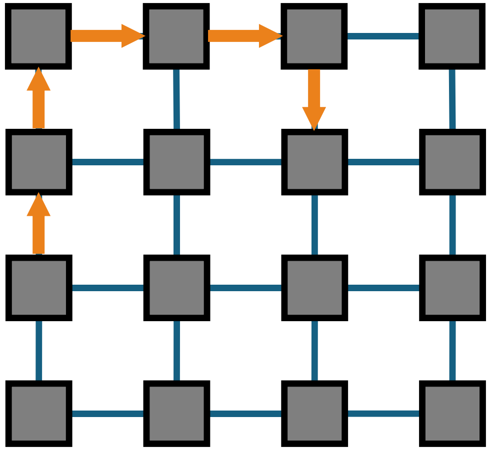
- Potential routing algorithms
  - Greedy (deterministic, minimal): always pick the shortest direction
  - Uniform random (oblivious, non-minimal): randomly pick a direction with equal probability
  - Weighted random (oblivious, non-minimal): randomly pick a direction with a higher weight for shorter direction
  - Adaptive (adaptive, non-minimal): pick direction based on load of the local channels

Dimension-order Routing

- First move towards x-dimension, then move towards y-dimension (XY)
  - To increase the clarity, we will focus on 2D meshes in the following
- Example: 2D Mesh



Dimension-order routing:
Deterministic and minimal

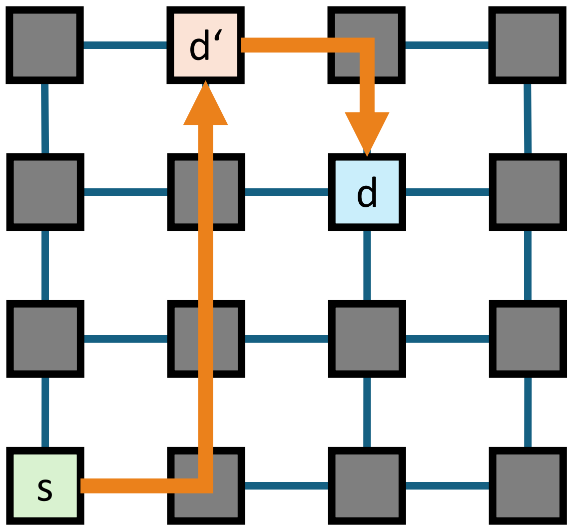


Alternate route:
non-minimal

- Dimension-order routing: simple, minimal, can cause load imbalance, doesn‘t exploit path diversity

Valiant‘s Algorithm

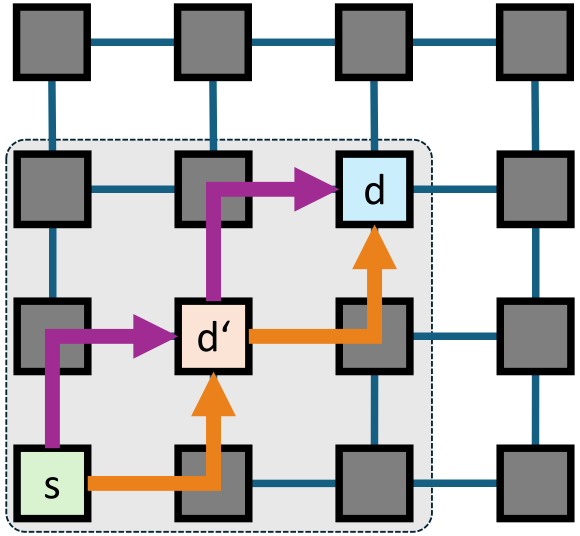
- Packet from source *s* to destination *d* is routed via an intermediate node *d‘*
  - Randomly select intermediate node *d‘*
  - Phase I: Route packet from *s* to *d‘*
  - Phase II: Route packet from *d‘* to *d*
  - Use arbitrary routing algorithm for Phase I+II, e.g., dimension order routing for tori and meshes



- Can use arbitrary routing algorithm for the two phases
  - For tori and meshes: Dimension-order routing as appropriate choice
- Valiant‘s Algorithm: Randomizes traffic, balances network load, non-minimal, doesn‘t exploit locality

Valiant‘s Algorithm

- Minimal version of Valiant‘s algorithm for k-ary n-cubes:
  - Restrict intermediate node: *d‘* lies in minimal quadrant between *s* and *d* (subnetwork with *s* and *d* as corner nodes)
  - Randomly selects among minimal routes
- Steps:
  - Identify quadrant
  - Select intermediate node *d‘* from quadrant
  - Route from *s* to *d‘*
  - Route from *d‘* to *d*



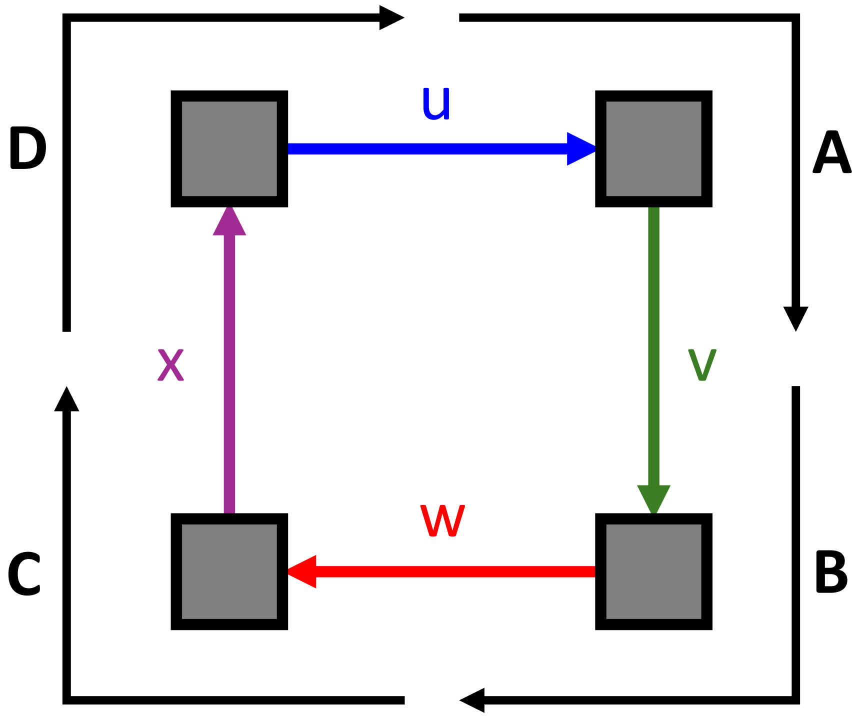
- With dimension-order routing (either XY or YX): Doesn‘t use all paths
  - Idea: Select randomly whether to use XY or YX (but: deadlock problem arises)

- Preserves locality, improves load balancing (compared to deterministic routing)

Deadlocks

- Deadlock: Situation where packets cannot make progress as they are waiting on each other to release resources (buffers or channels)

- Example:
  - Nodes: 0, 1, 2, 3; Channels: u, v, w, x
  - A holds *u* and waits for *v*
  - B holds *v* and waits for *w*
  - C holds *w* and waits for *x*
  - D holds *x* and waits for *u*

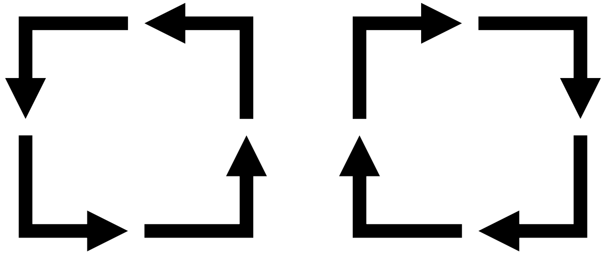


- Observation: Cycles pose a problem

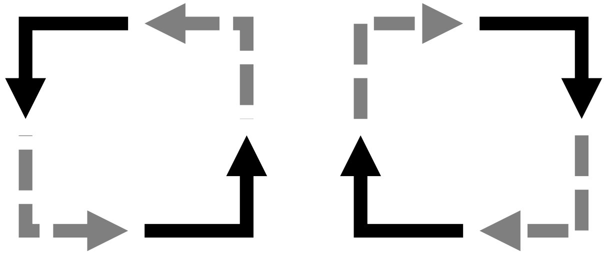
Deadlock Avoidance: Restrict Routing

- Dimension Order Routing (k-ary n-meshes)
  - E.g., first x then y (we have seen this approach already)
  - Deadlock-free, but restricts path diversity

- Turn Model: Focuses on the turns allowed and the cycles they can form
  - 2D mesh: 8 possible turns forming two abstract cycles

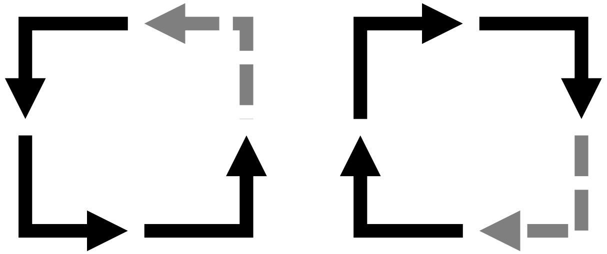


  - XY Routing removes four turns (prevents deadlocks)

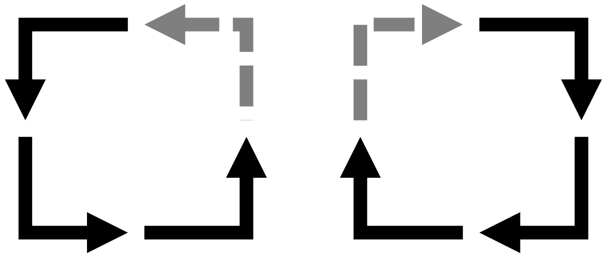


Deadlock Avoidance: Restrict Routing

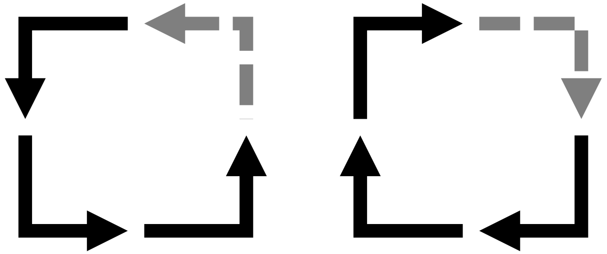
- Turn Model: Focuses on the turns allowed and the cycles they can form
  - Removing one (carefully selected) turn from each abstract cycle also prevents deadlocks



west-first: traveling west only allowed at the start

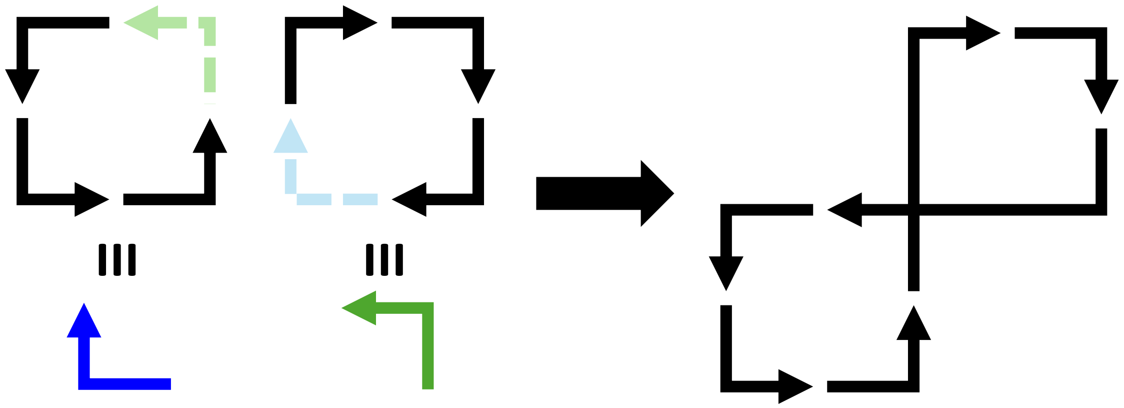


north-last: traveling north only allowed as last direction



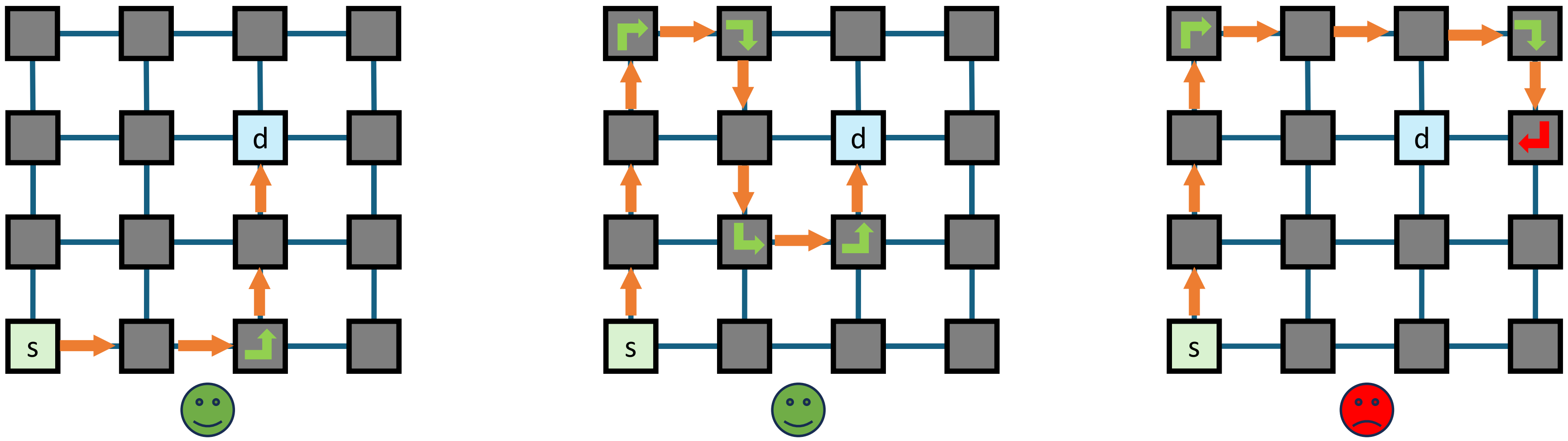
negative-first: traveling first west and south, then east and north

  - Removing any two turns does not prevent deadlocks

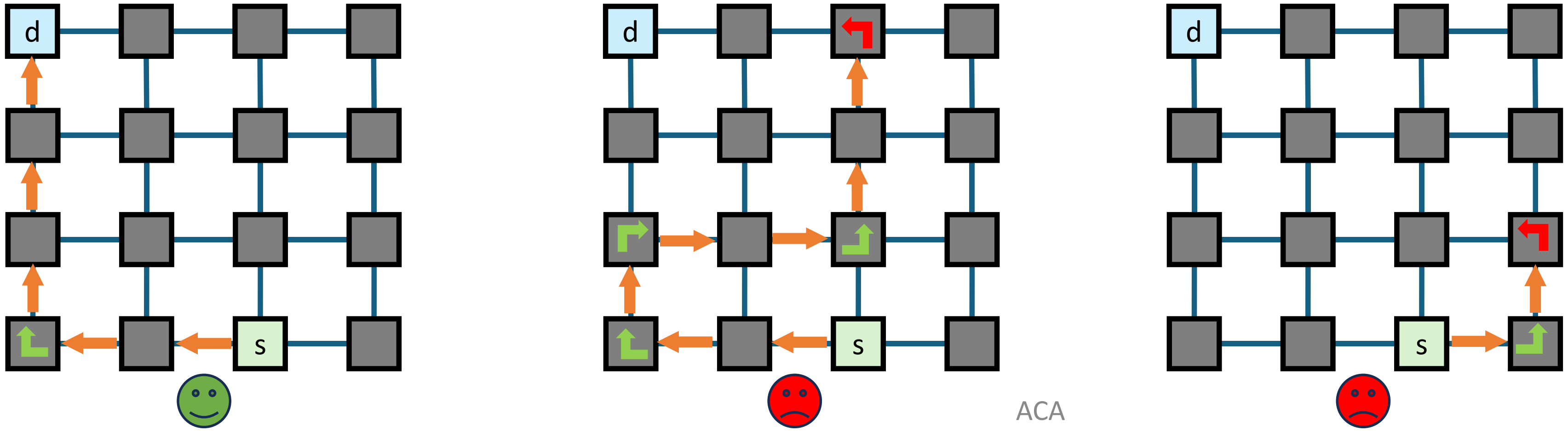


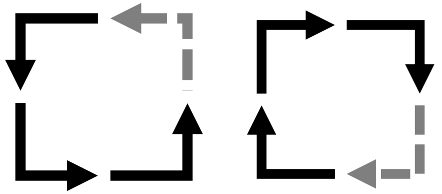
Examples: West-First

- Example 1

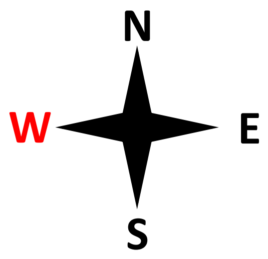


- Example 2

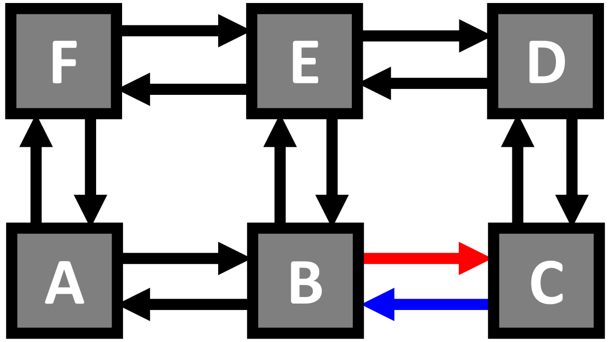




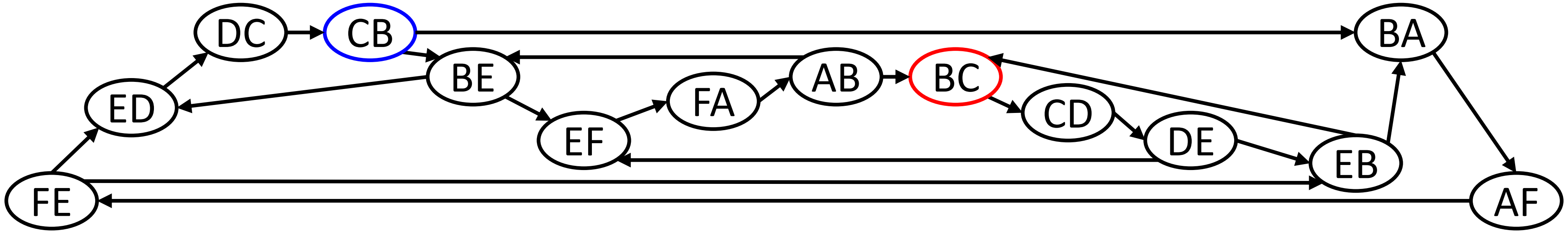
west-first: traveling west only allowed at the start



- Network topology:

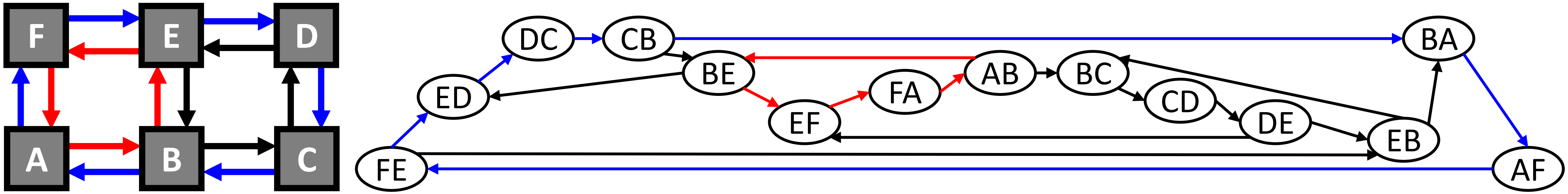


- Channel Dependence Graph:
  - One vertex for each channel
  - Edges denote dependences
    - Dependence exists if it is possible for channel *i* to wait for channel *i+1*
    - 180° turns not allowed (e.g., AB → BA)

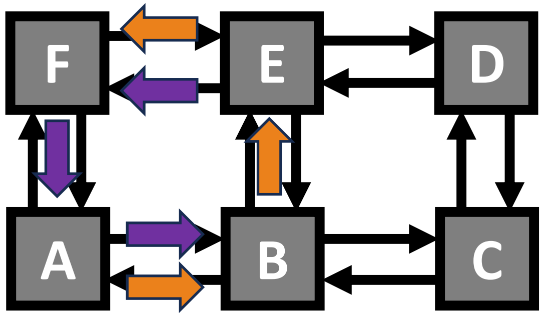


Cycles in the CDG

- Channel Dependence Graph may contain cycles



- Route through AB, BE, EF and route through EF, FA, AB → Deadlock

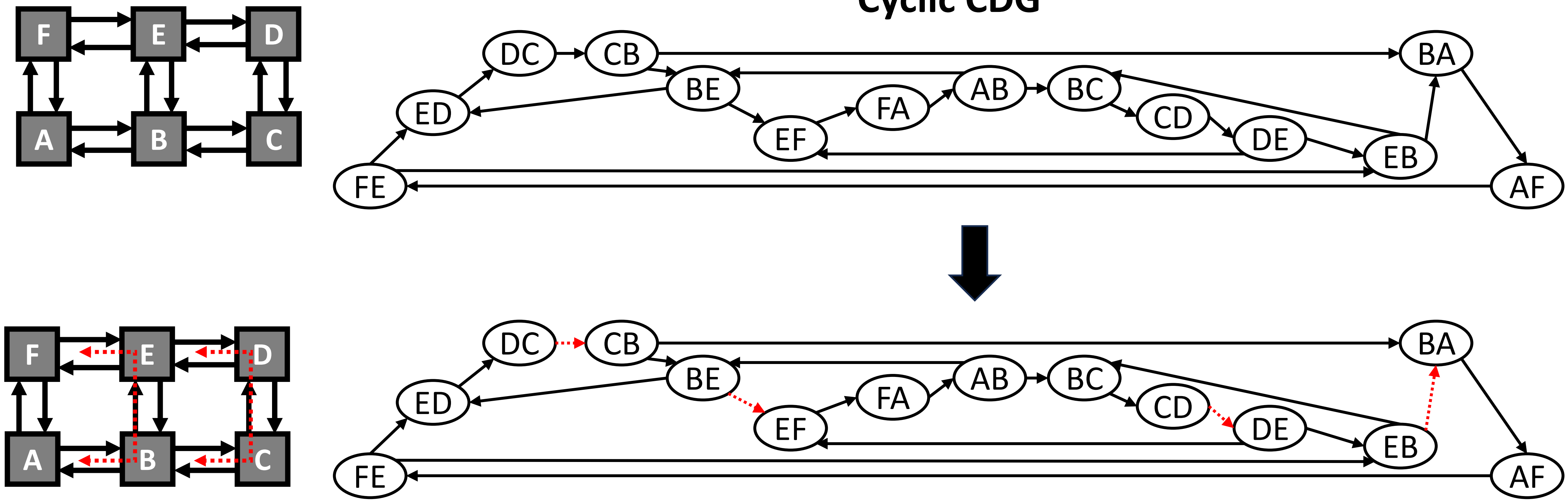


→ *Remove selected edges in the CDG*

Acyclic CDG

- Example: Remove Edges in the CDG (West-first turn model)

Cyclic CDG



Acyclic CDG

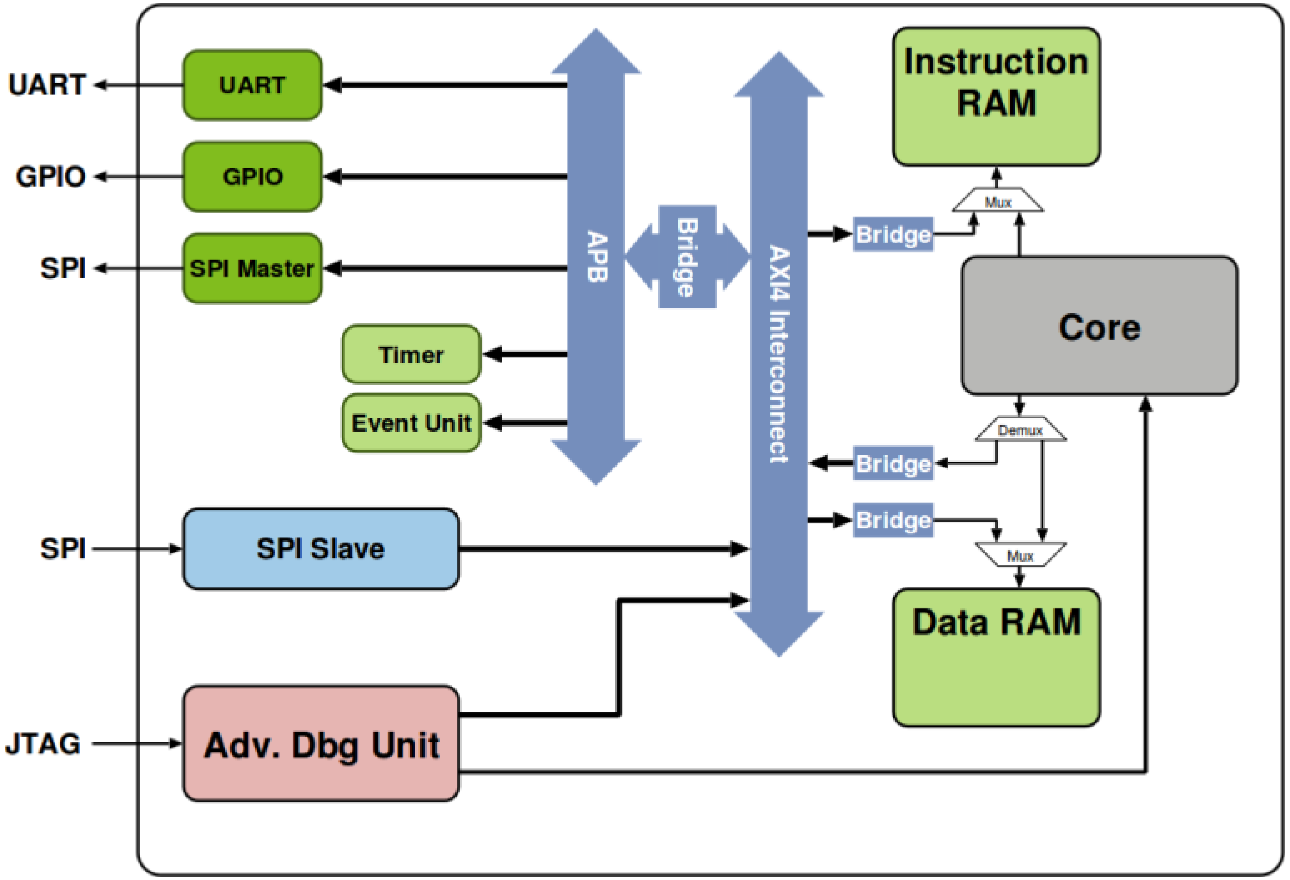
A look at real Systems-on-Chip

Optional, not relevant for exam

PULP 2016, PULP 2022, SpiNNaker2

Simple SoC Architecture for IoT / Wearables – Example - PULPino 2016

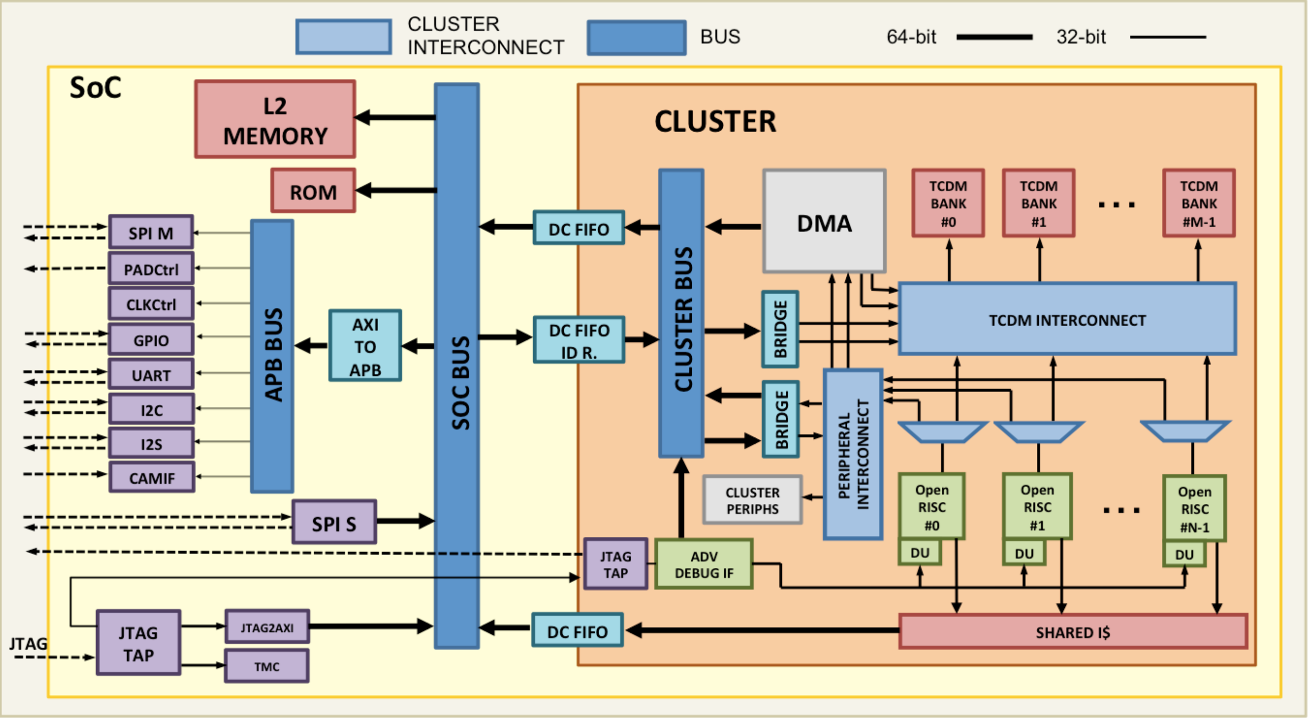
- SoC: System-on-chip
- PULPino Architecture 2016:
  All memories are on the same chip as the processor core
- SoC Modules:
  - Processor Core
  - Instruction memory
  - Data memory
  - Input/output devices: UARR, SPI, GPIO
  - Timer
  - Programming and Debug Devices: SPI Slave and Debug Unit
  - Connected by **on-chip interconnect:** AHB, AXI4

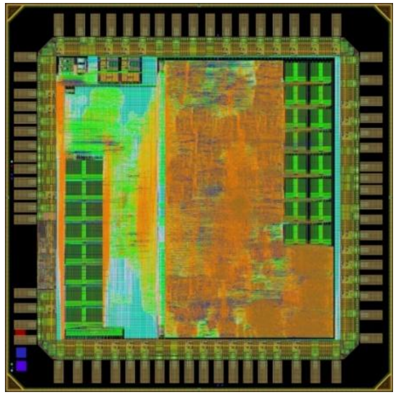


Source: CNX Software
https://www.cnx-software.com/2016/04/06/pulpino-open-source-risc-v-mcu-is-designed-for-iot-and-wearables/

Complex Multi-core SoC – Example PULP 2022

- More complex architecture
- Different **On-chip Interconnects**
- DMA: Direct Memory Access – Module to offload data movements from the CPU
- Multi-Core with shared caches
- All these modules are physically integrated in one integrated circuit (IC).

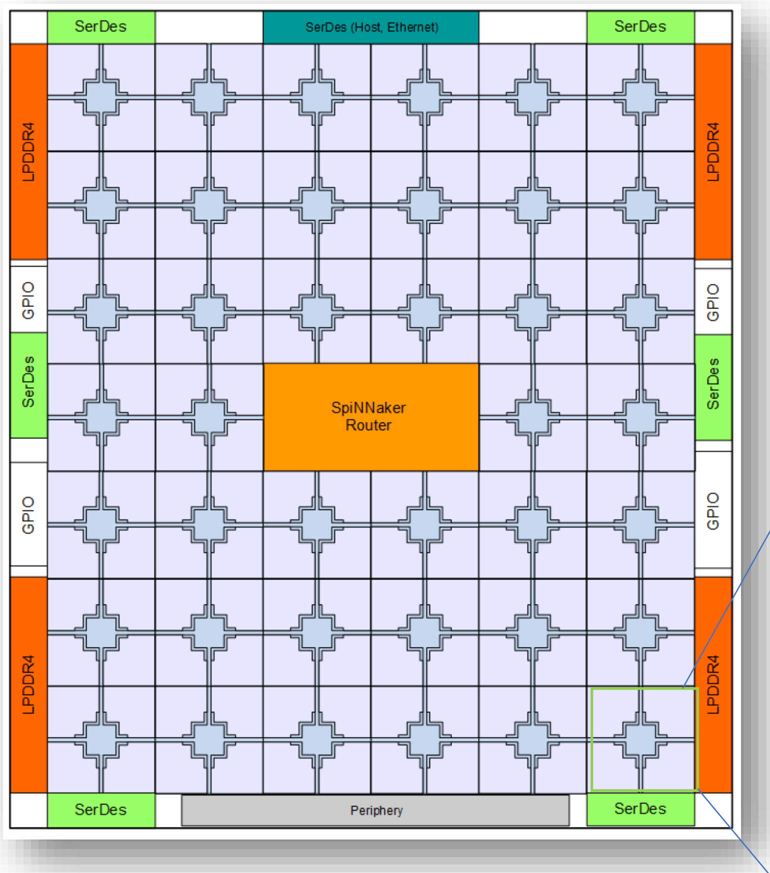




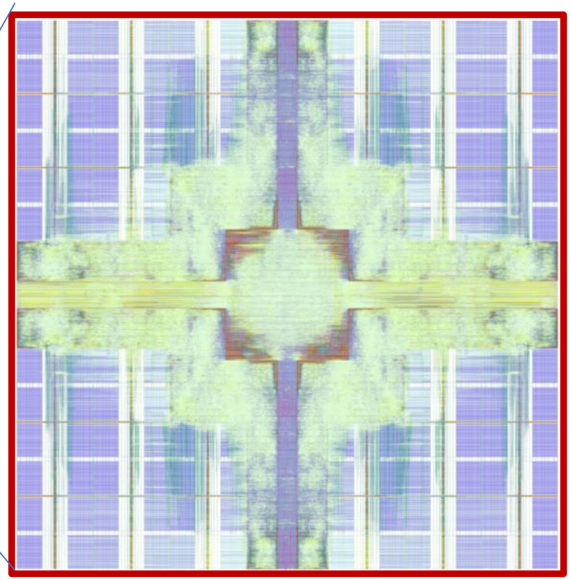
Source: https://iis-projects.ee.ethz.ch/index.php/PULP

SpiNNaker2 Chip

- Brain-inspired Chip designed for Spiking Neural Netwoks (SNNs)



Mesh NoC



Source: SpinnCloud

Summary

Conclusion

- Bus-based On-chip Interconnect
- Network on-Chip
- Next Sessions: Specialized Cores

Thank you for your attention