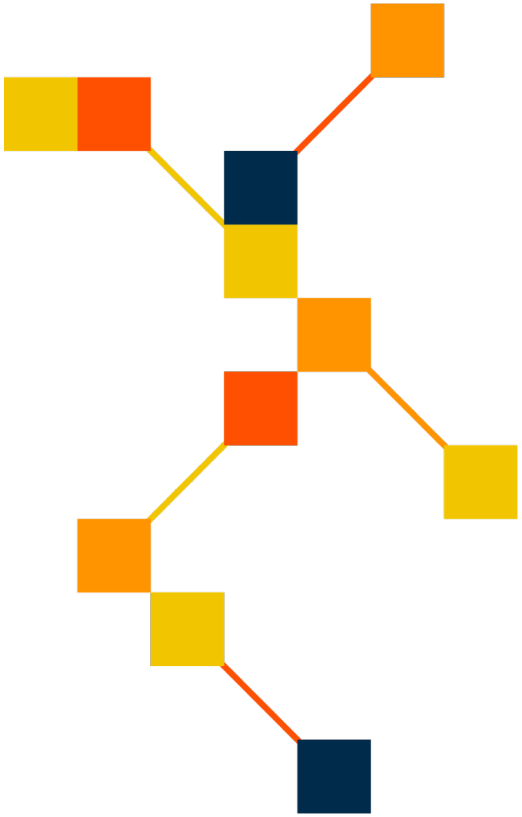


Your SoC, Your Topology

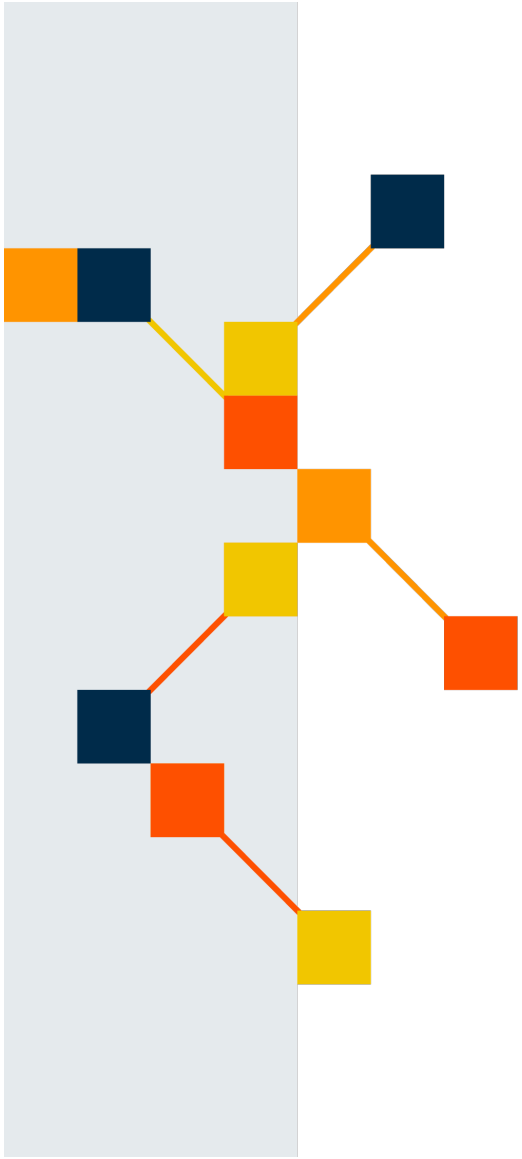
Interconnects used within SoCs

Ami Pathak, Senior Staff Field Application Engineer
Matt Mangan, Director of Corporate Application Engineering

ARTERIS IP

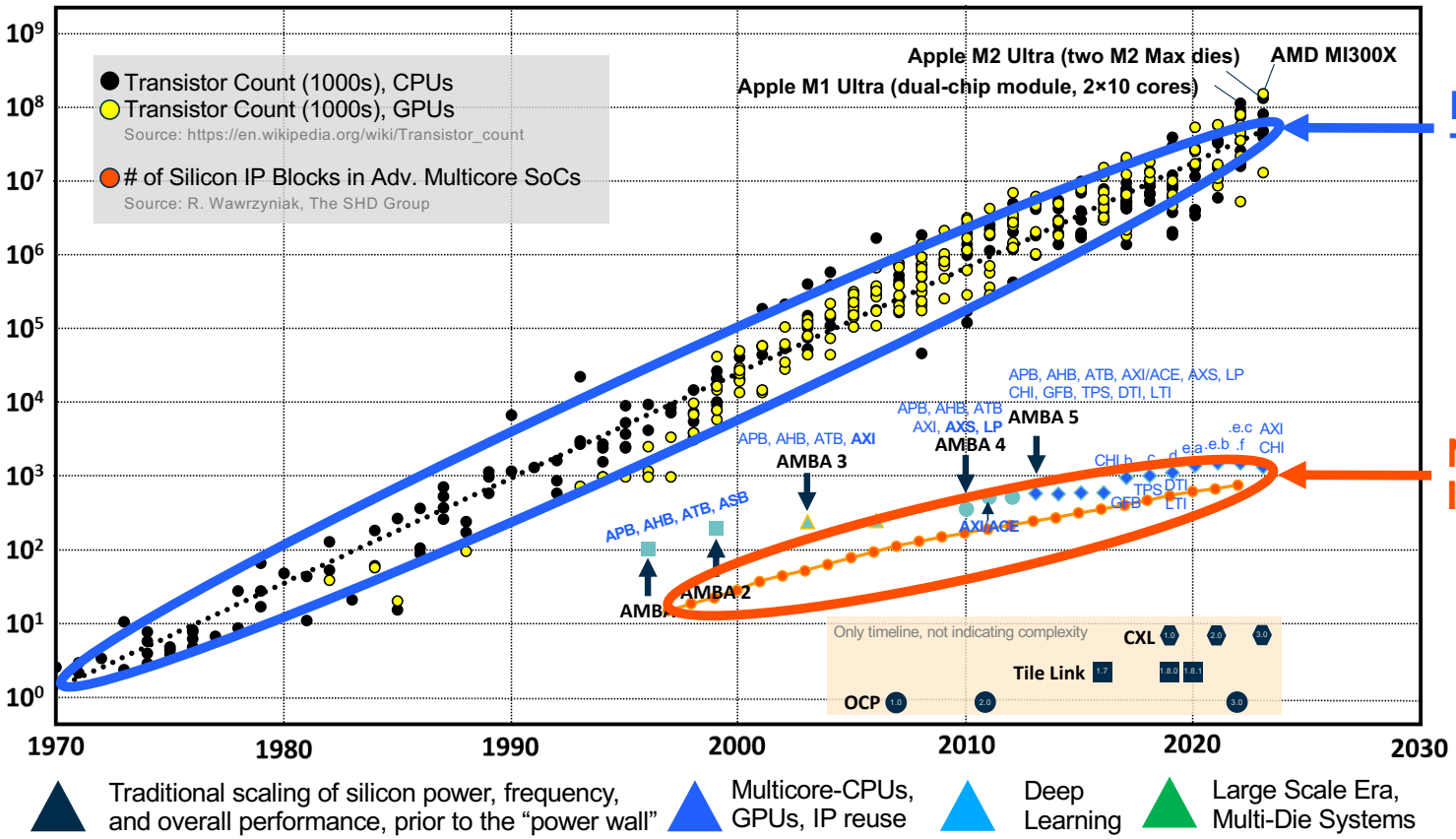
Agenda

1. Types of interconnects within SoCs
2. But why the transition to NoC interconnects?
3. NoC topology optimization part 1 – Adaptation to layout
4. NoC topology optimization part 2 – Quality of service
5. Summary



Types of interconnects in SoCs

50+ Years of Exponential Technology Scaling



Source: Frank Schirrmeister, Arteris, based on https://rb.gy/qmfcn, https://rb.gy/pgdop, The SHD Group, Public Protocol Specs

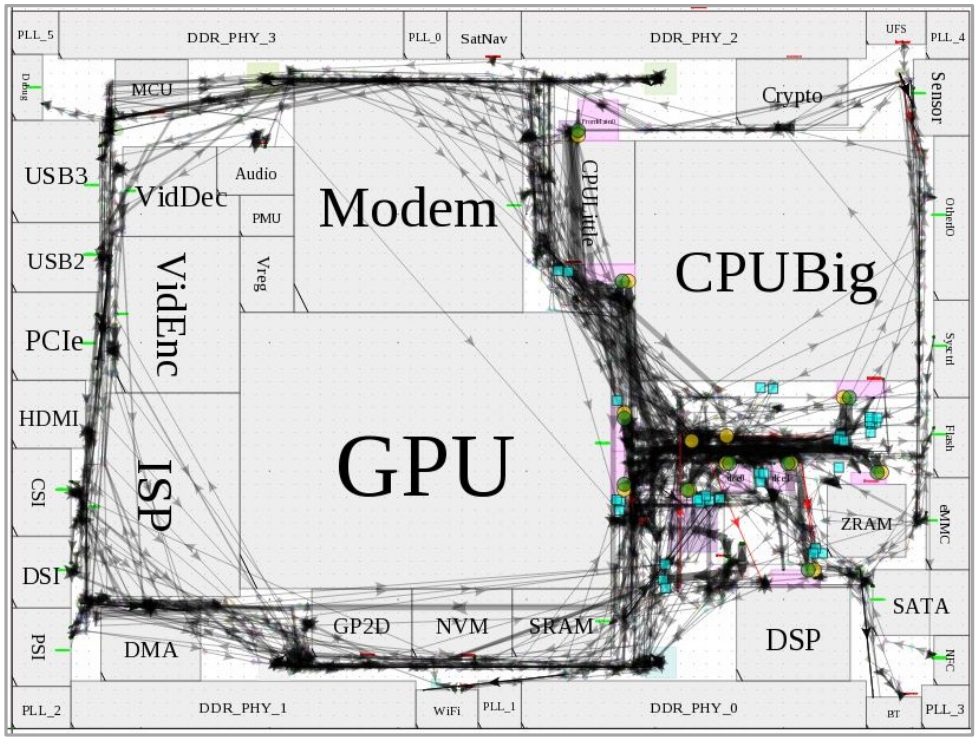
ARTERIS IP

Bigger chips, more transistors, and more IPs = Better SoCs… right?

Yes, except not without interconnect challenges!

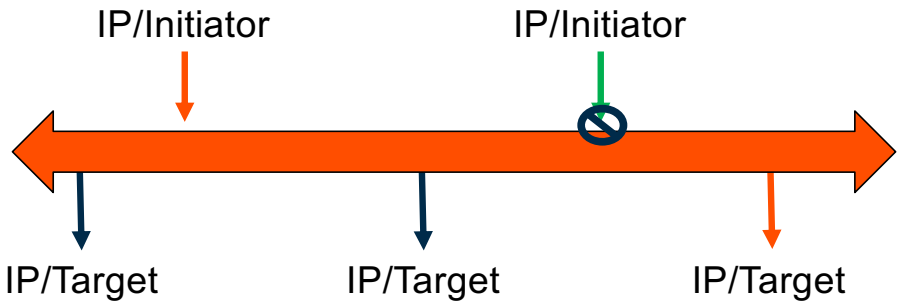
- The SoC interconnect implements the connections between all the IPs
- Interconnect is the only IP that traverses the chip
- Has the longest wires of any IPs
  - Span long distances
  - Congestion points
- Carries most of the interesting data
- Changes in response to Architecture ECOs
- Must not be the bottleneck of the SoC performance
- Often the last IP to be frozen – time to backend closure becomes schedule-critical!

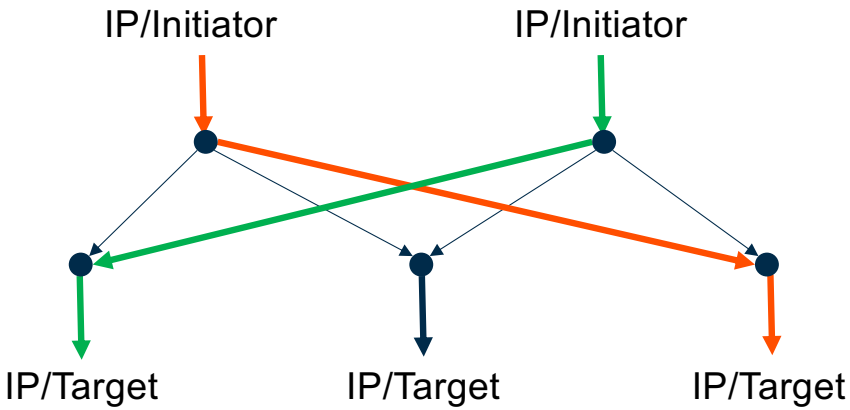
How to design it just right, without overdesign?



ARTERIS IP

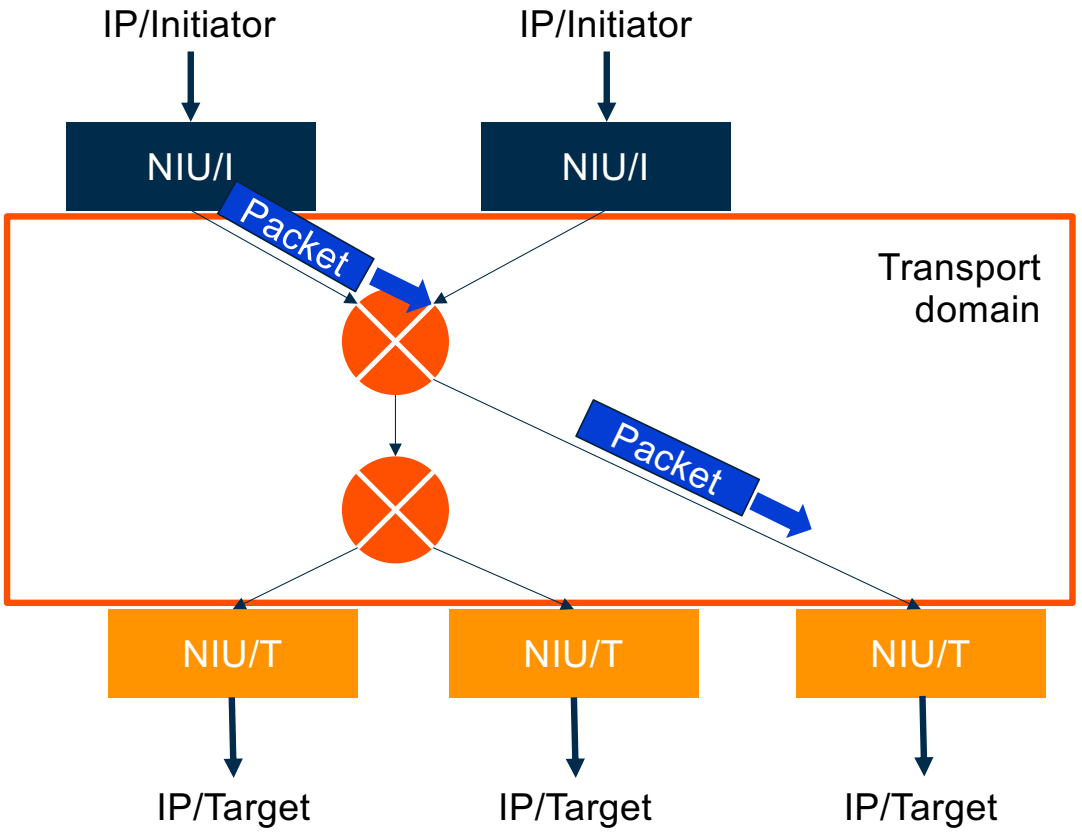
Buses and Crossbars



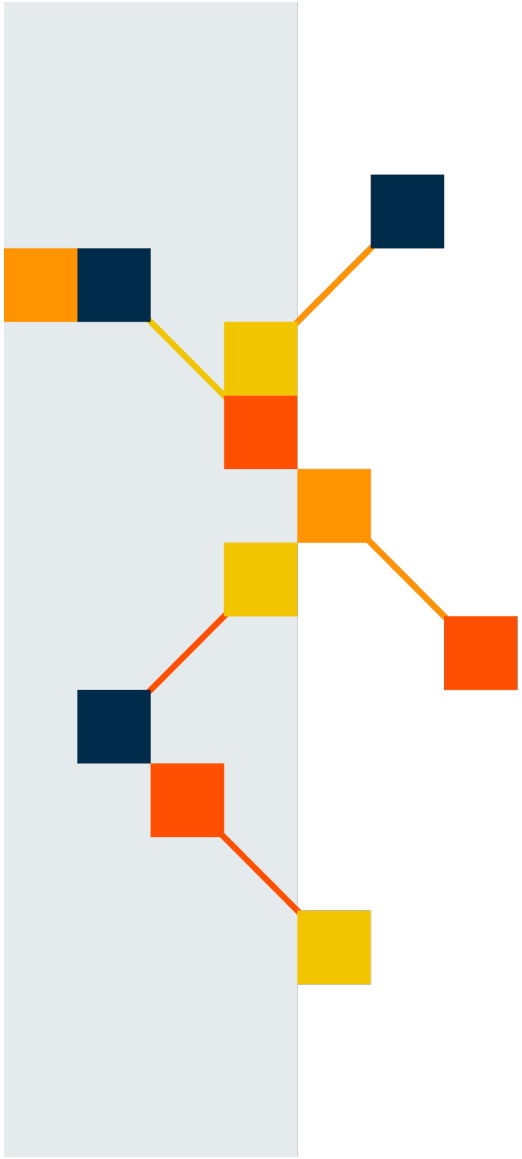


- Buses are:
  - Single, shared communication resources
    - Everything is serialized
  - Broadcast from selected source to all destination
  - Arbiter selects which source owns the bus
  - Address decode selects which destination shall listen to request
  - example: PCI
- Crossbars are:
  - Point to point communication resources
    - Can have multiple communications in parallel
  - Switches manage traffic
    - Routing to destination
    - Arbitration in case of conflicts

Network on Chip



- Network On Chip function
  - Transports **requests** from **Initiators** to **Targets** and responses back
  - Initiators: CPU, GPU, DMA, …
  - Targets: SRAM, DDR, registers…
- Component breakdown
  - The **transport** is built using **switches**
  - The switches arrangement is the **topology**
  - Often, the **transport uses its own protocol**
    - Fine tuned for distance spanning, efficiency
    - Transport atoms are called **packets**

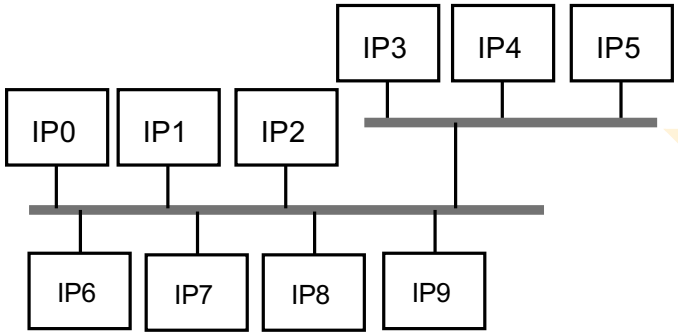


But why the move to NoC interconnects?

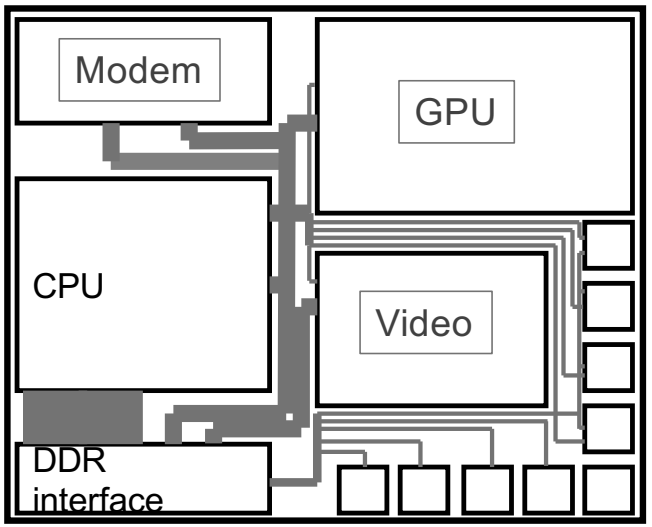
NoCs help solve SoC implementation challenges

- System on Chips present unique implementation challenges for the communication between IP modules
- Network on Chip technology addresses all of them

High-level view

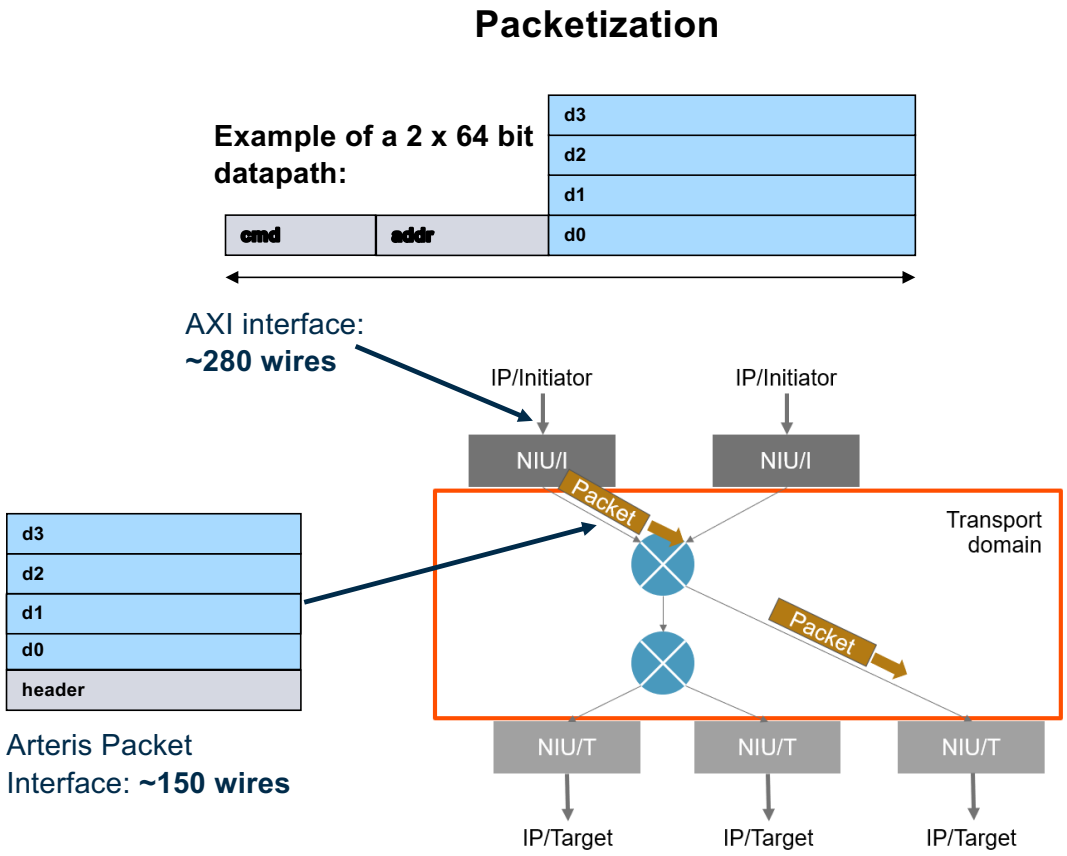


Physical view



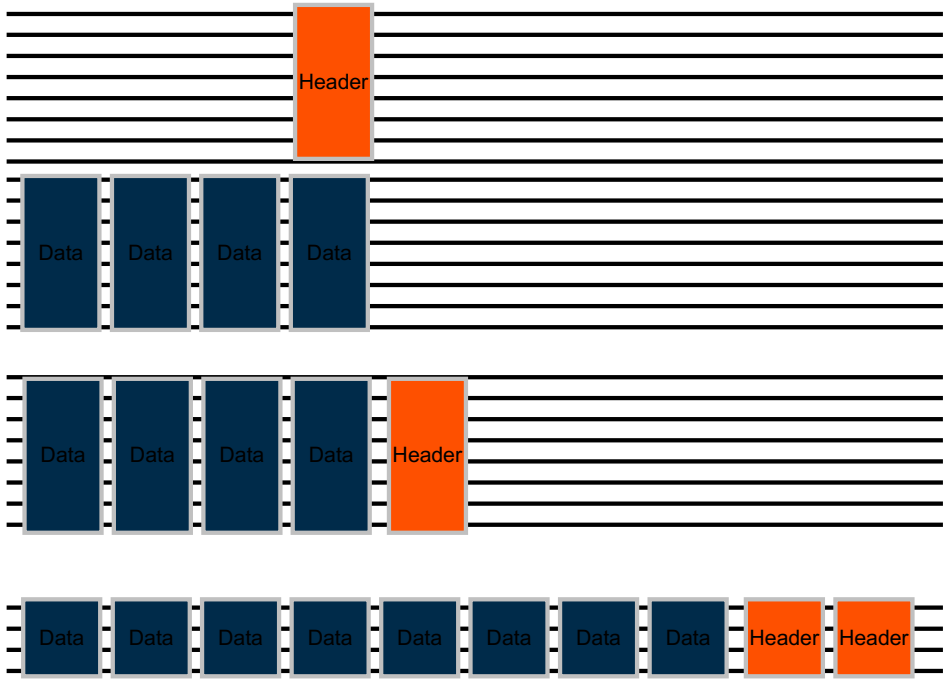
Arteris NoC Packetization

- Reduce routing congestion
  - Packetization – fewer wires
- Ease timing closure
  - Fine grain pipeline stage insertion
- Reduce die area
  - Simpler datapath logic
- Reduce active power
  - Unit-level dynamic clock gating
- Decouple sockets from transport
  - Easier integration

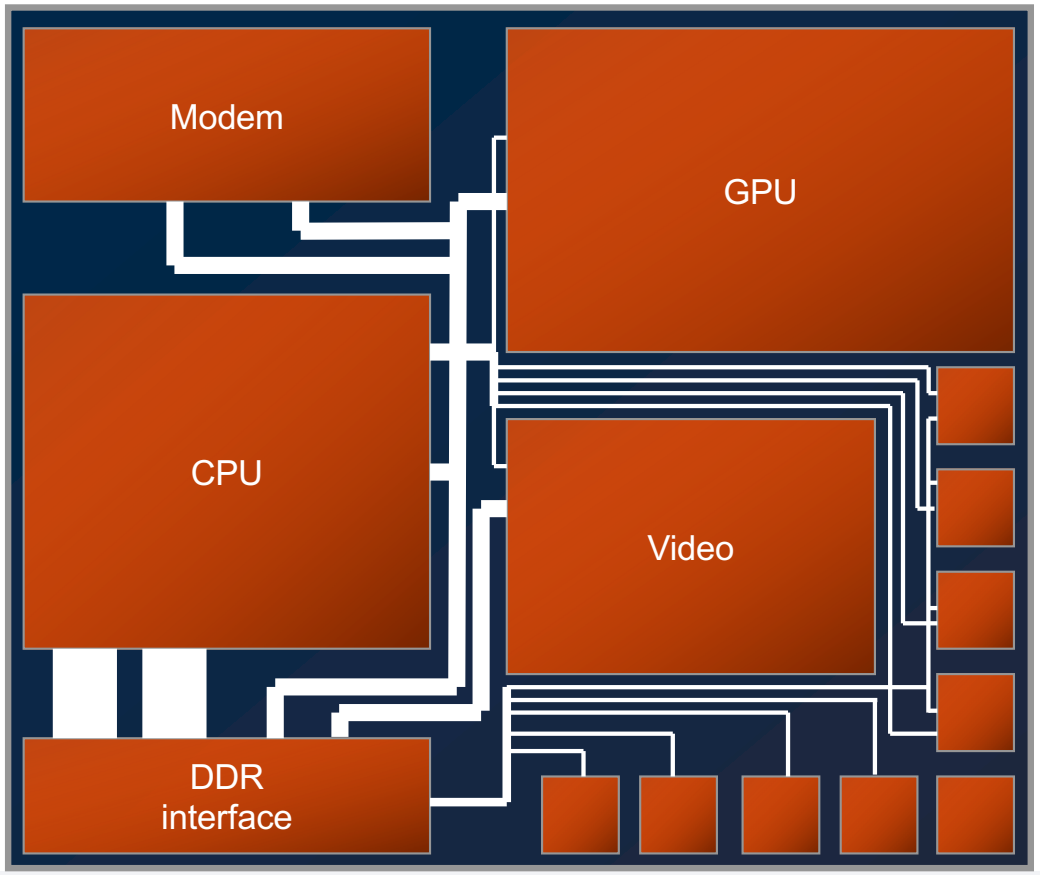


Serialization

More activity on fewer wires
- Less datapath logic
- Less place & route congestion



Flexible Serialization: Metal Reduction

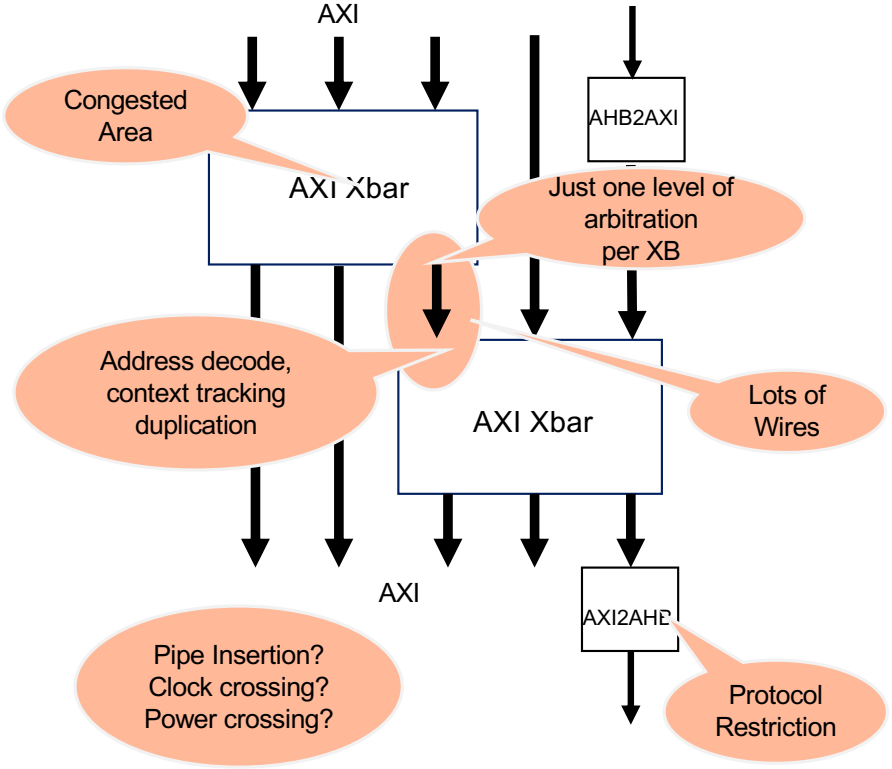


wide links are short

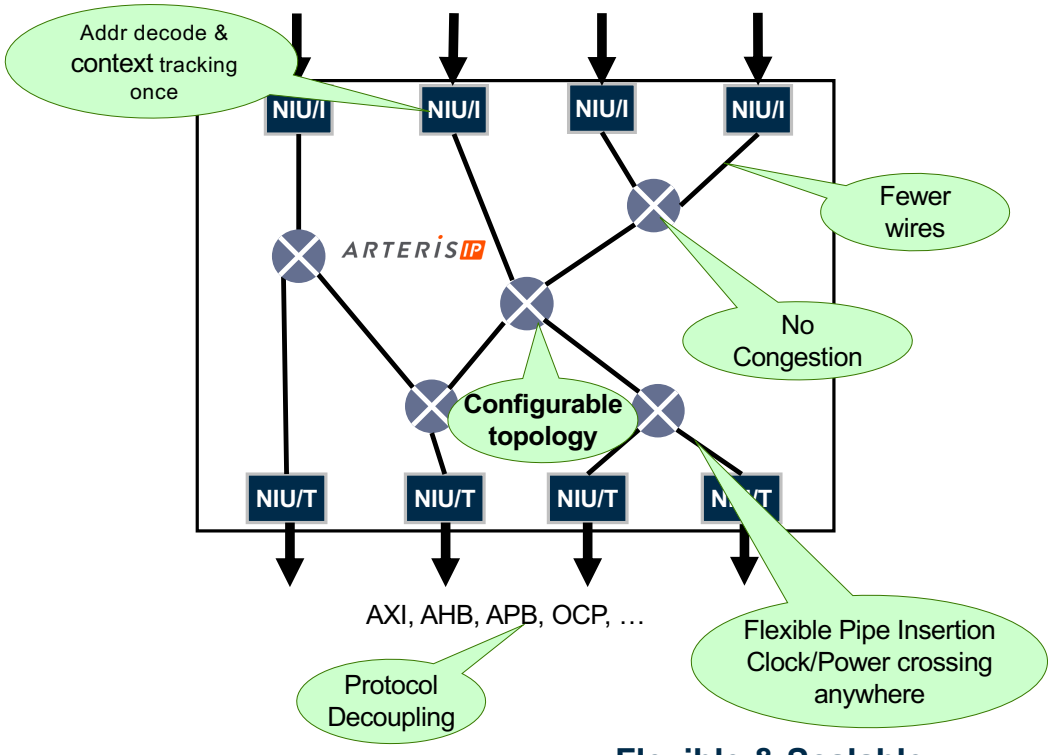
long links are narrow

Better than cascaded crossbars

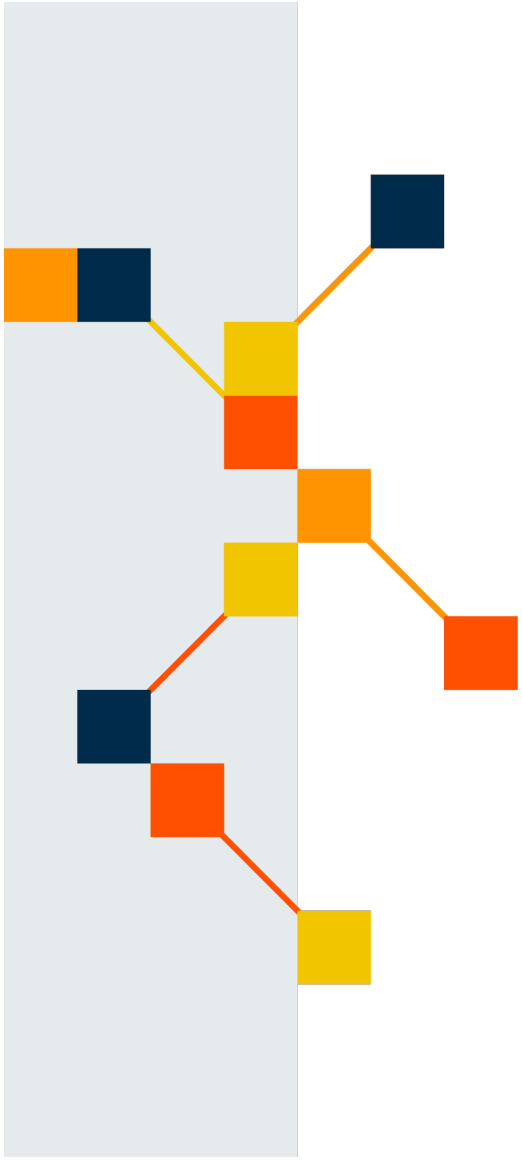
Cascaded crossbar architecture + bridges



NoC transport based architecture



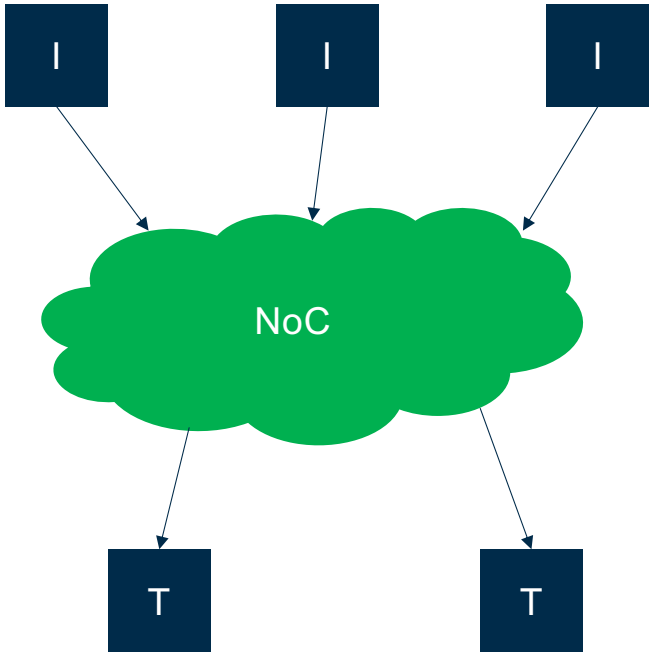
Flexible & Scalable



NoC topology optimization

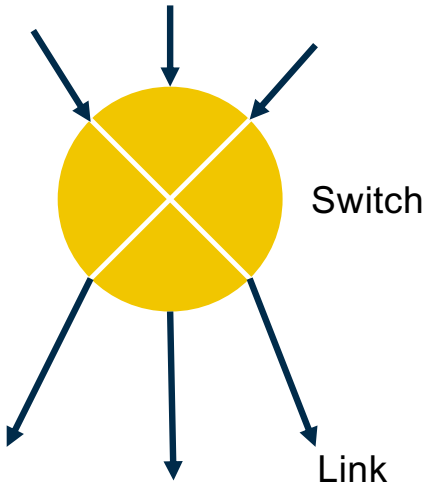
Part 1: Adaptation to layout

How many ways to implement an NoC?



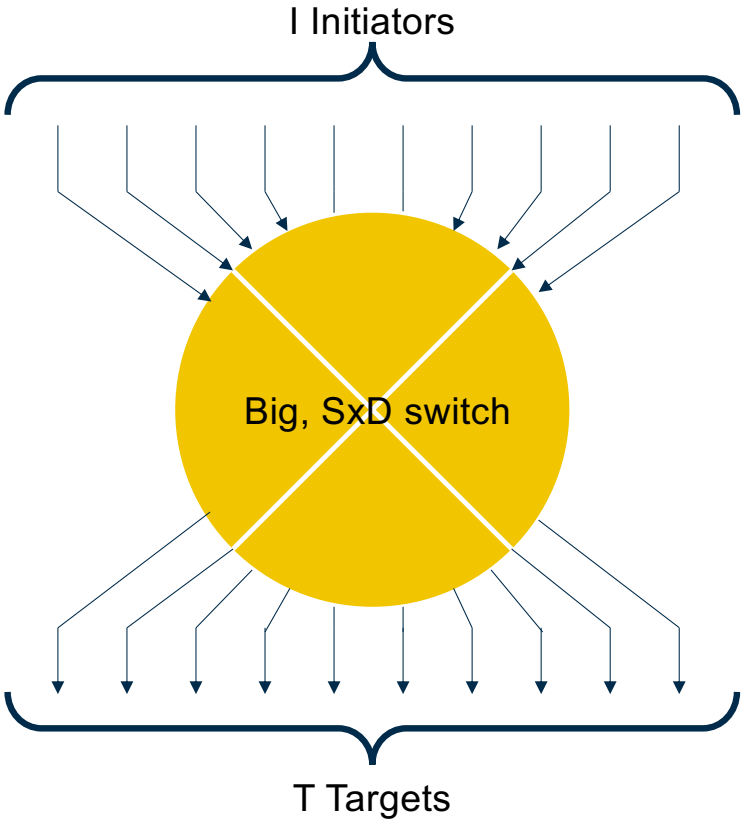
- The problem:
  - We need to connect I initiators to T targets using a set of switches and links
  - Without lack of generality let's assume all I need to communicate with all T
  - What is the optimal interconnect implementation, knowing:
    - The amount of traffic between each I and each T?
    - The location on the chip of all I and all T?
  - Optimal interconnect implementation means minimize:
    - **Amount of logic in switches**
    - **Amount of wiring between switches**
- **Solving this problem is finding an optimal topology!**

Definitions – Switch & Link



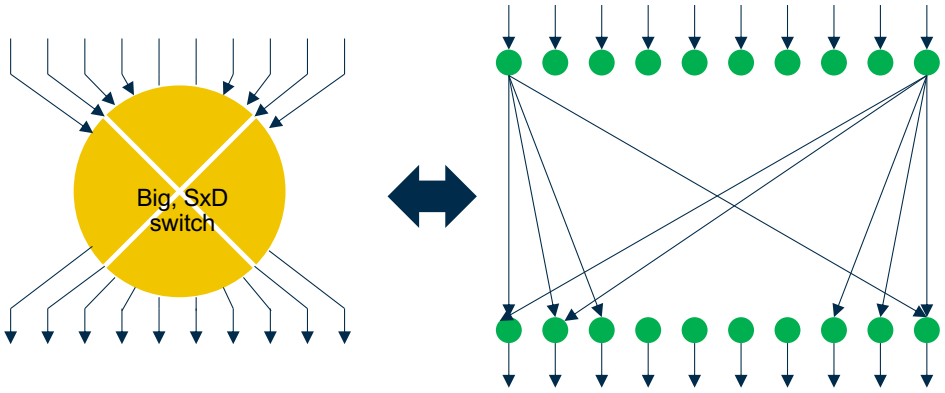
- A **switch** is a logic function that can route the traffic from I inputs to O outputs
  - It performs routing: Mapping a request from its input ports to an output port according to a mapping function
  - It performs arbitration: Resolution of conflicts when multiple input ports need to access the same output port
  - More details on the switch structure later
    - For now it's a black box
- A **link** connects a switch output port to the next switch input port
  - Protocol converters at the edge of the network are connected to switches using links as well
- Switches and links carries **packets**

Why everything is not just a big switch?



- Simplest solution: Use a I x T switch!
- Problem: It does not scale well!

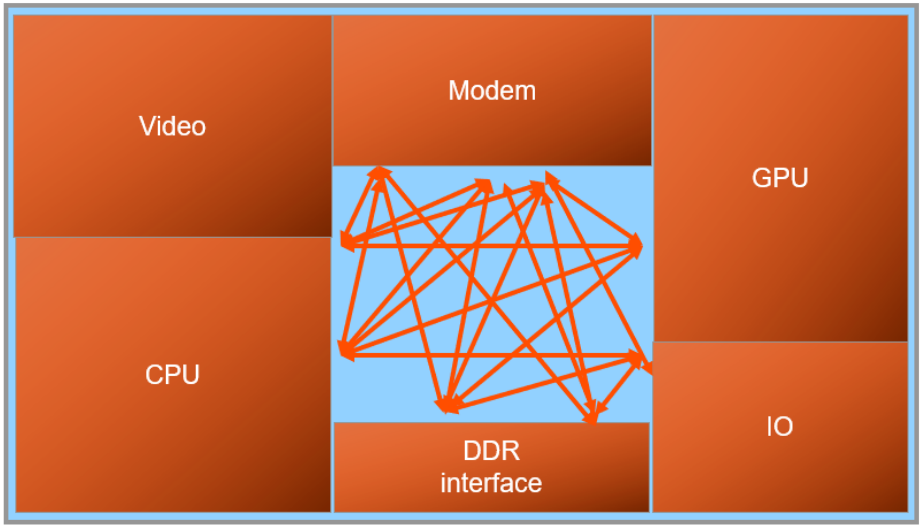
Cost of a “big” switch



- A I x T switch can be decomposed in:
  - I switches of size 1 X T
    - Also called “splitters”
  - T switches of size I x 1
    - Also called “mergers”
  - I x T wires
- This topology is called a **crossbar**
- Examples:
  - A 10 x 10 crossbar has:
    - 10 switches 1x10; 10 switches 10x1; 10x10 = 100 wires
  - A 20 x 20 crossbar has:
    - 20 switches 1x20, 20 switches 20x1, 20x20 = 400 wires
- The cost of a crossbar is **quadratic**

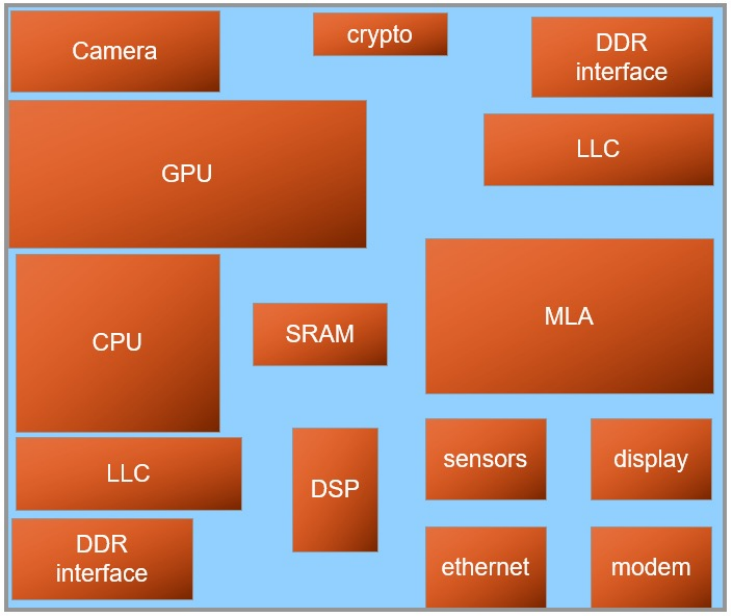
Big Switches and layouts

Big cross bars can be in the middle…



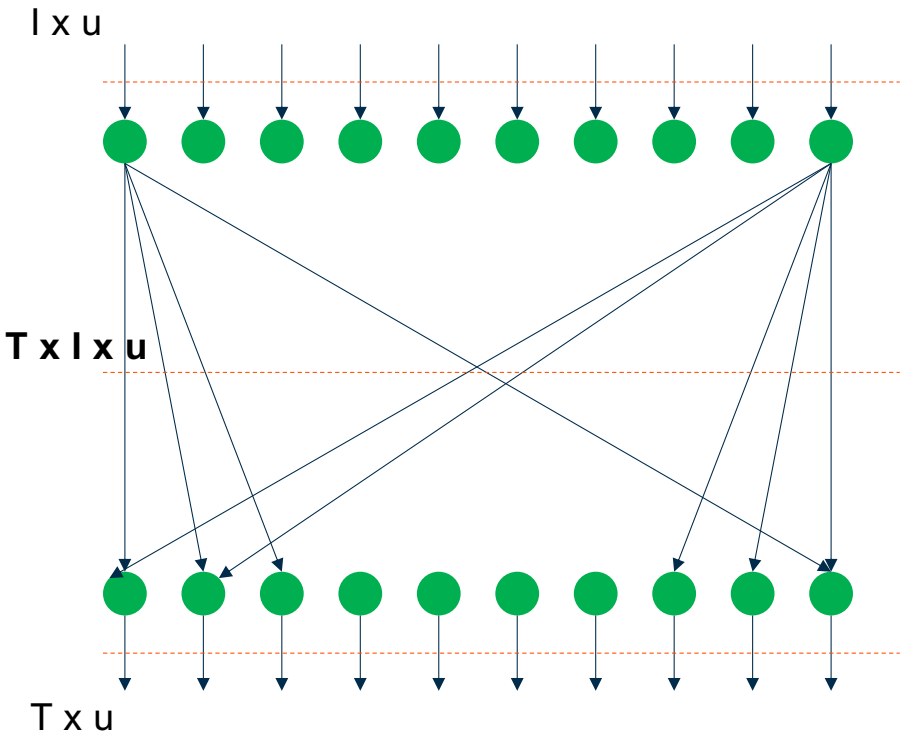
OK (kind of)

But nowadays there's usually something in the middle…



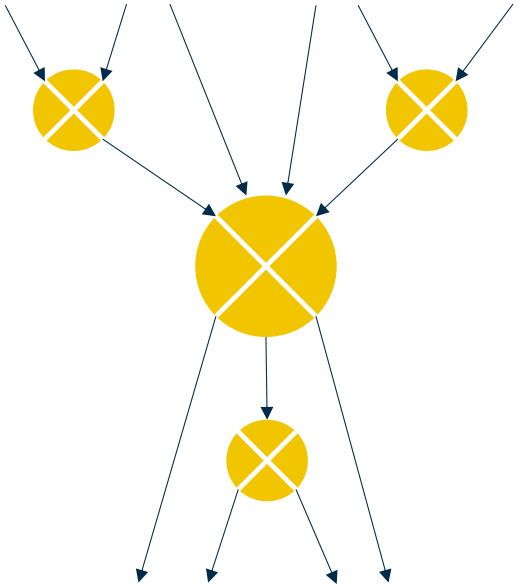
Don't even try

Do we really need all those wires though?



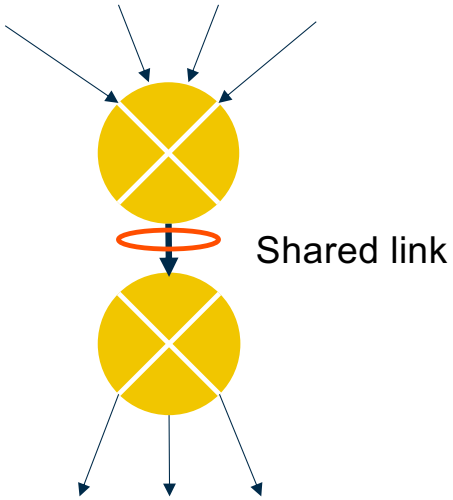
- **Cross-sectional bandwidth** - how much bandwidth goes through links when the interconnect is cut “in the middle”?
- If we say that each initiator I and each target T has the same link width and run at the same clock
  - Carries a bandwidth of “u”
- Network must transport min(I x u, T x u)
- But cross-section bw is I x T x u
- I x T x u >> min(I x u, T x u)
- **Conclusion: A huge majority of crossbar wires do nothing!**

Reducing the size of the big crossbar



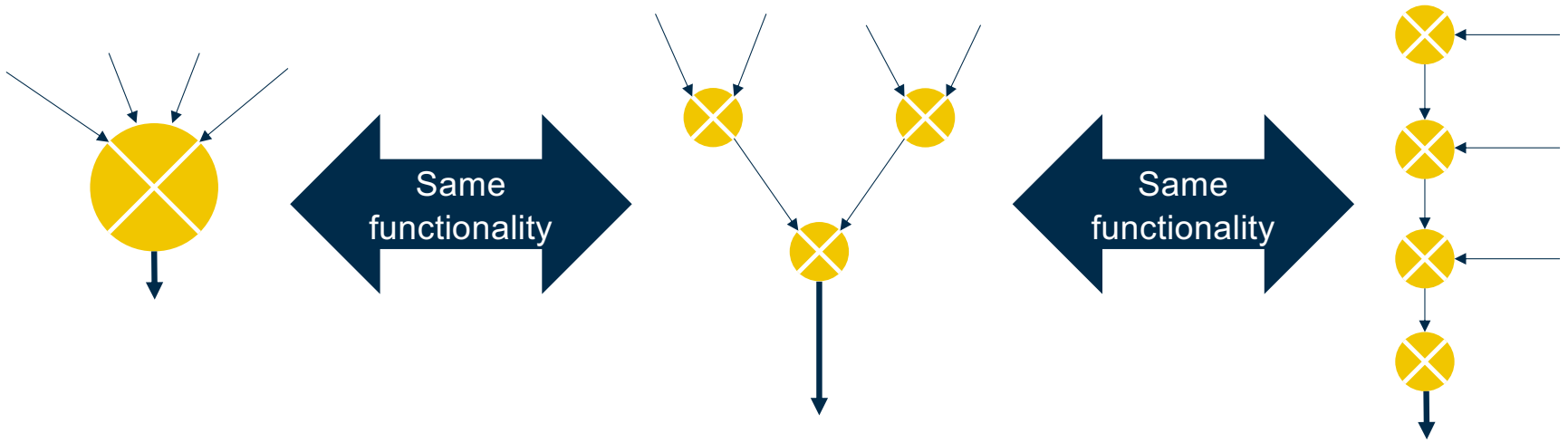
- **Combining traffic from masters as early as possible**
- **Keeping traffic to slaves combined as long as possible**
- Goal: Reduce the dimension of the crossbar
  - Gain is fast since crossbar cost is quadratic
  - A 14x14 crossbar is ½ the size of a 20x20!
    - Combining 6 sources, and 6 destinations, already half the size!
- Of course, reduction in size also comes with a reduction of the maximum throughput
  - Paths that were parallel before becomes serialized
  - This is perfectly acceptable for **most** applications!

Minimizing the size of the big crossbar



- What's the smallest topology using this approach?
  - Combine all intiators into one
  - Distribute the result to all destinations
  - This topology is called a **shared link**
- All packets serialized on a single link
  - Becomes the bottleneck for performance
- In the area/performance tradeoff, it's the smallest area / lower performance ratio

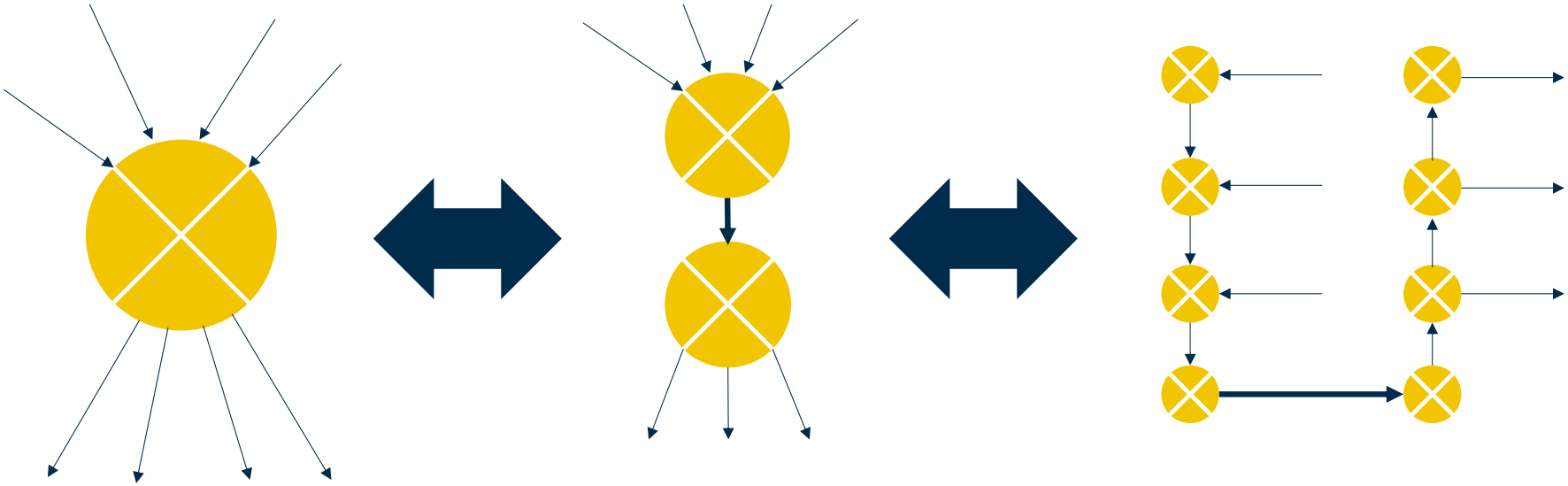
Adaptation to layout – basic idea



Centralized merger:
Everything done in a central place

Distributed merger:
Function spatially distributed → Easier to implement!

Adapt to layout – cont.

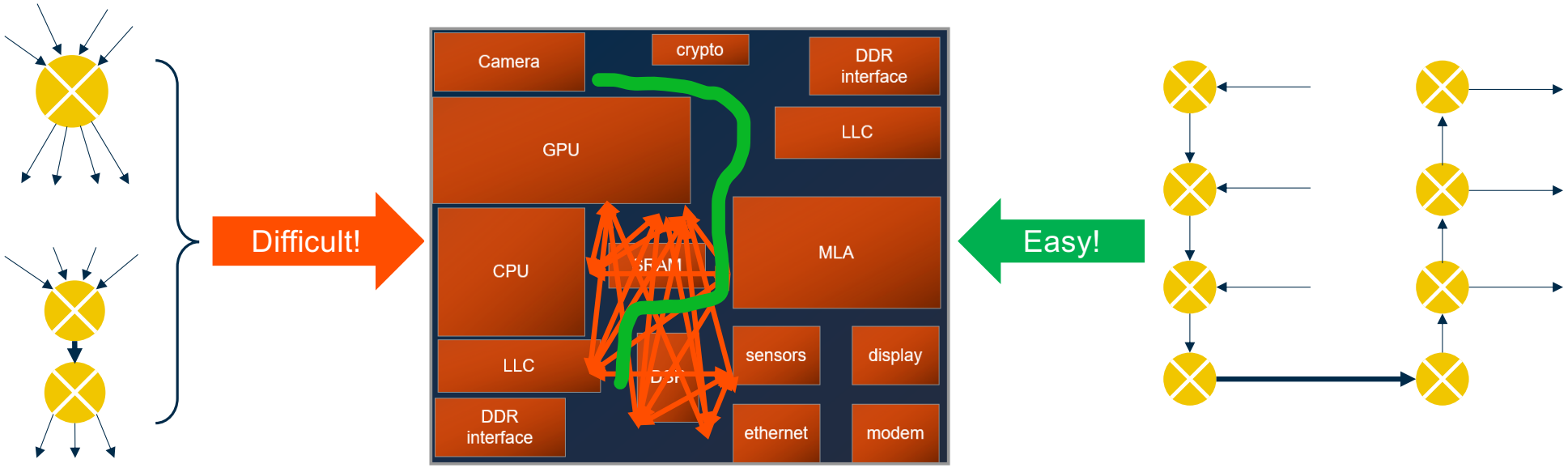


Centralized implementation
Biggest

Centralized implementation
Smallest

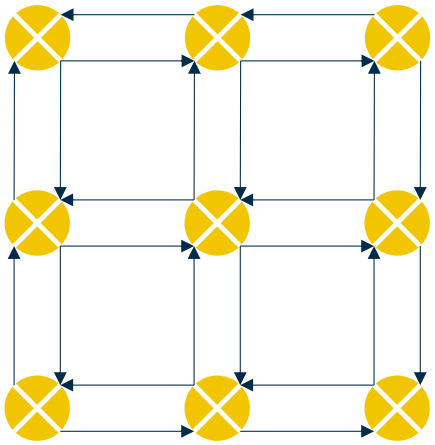
Distributed implementation

Benefit of distributed topologies

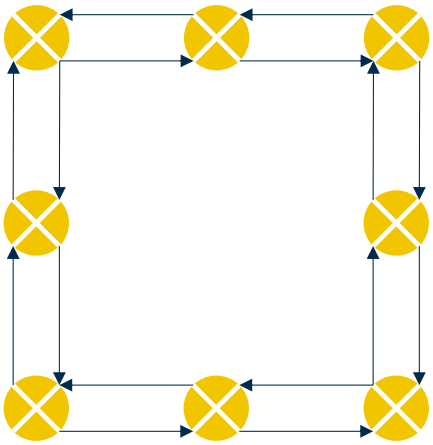


A distributed implementation is well adapted to most layouts

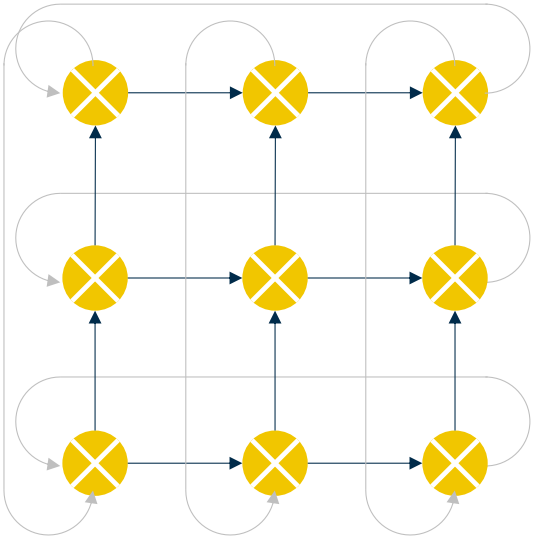
What about those other topologies?



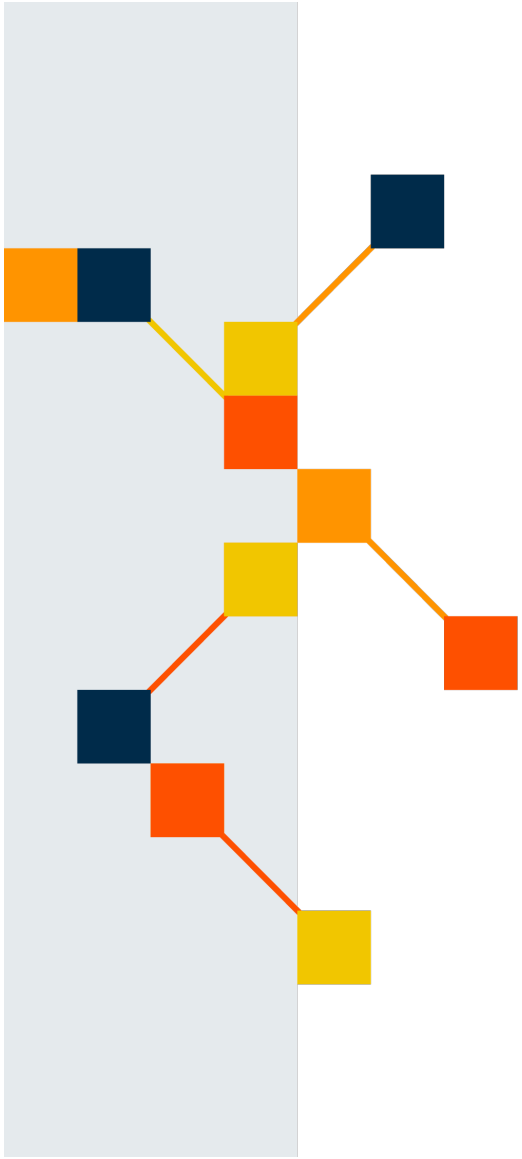
a) Mesh topology



b) Ring topology



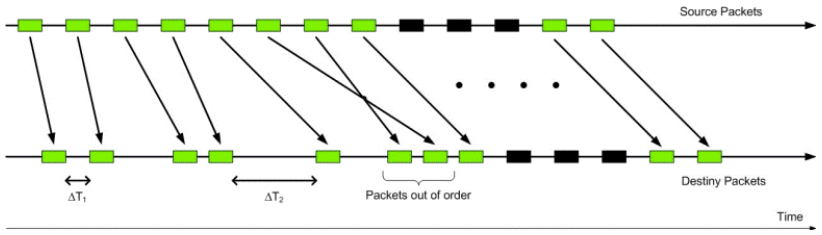
c) Torus topology



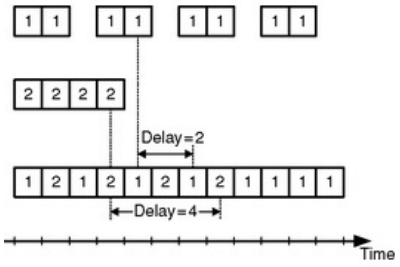
NoC topology optimization

Part 2: Quality of service

When will my stuff arrive? Some definitions…



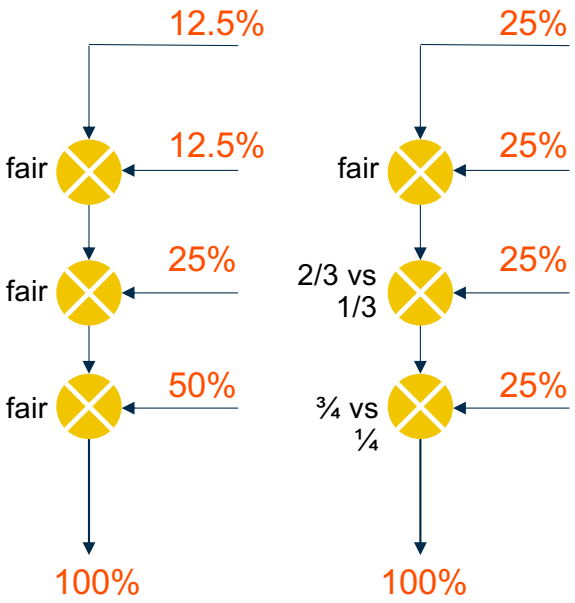
Jitter



Burstiness

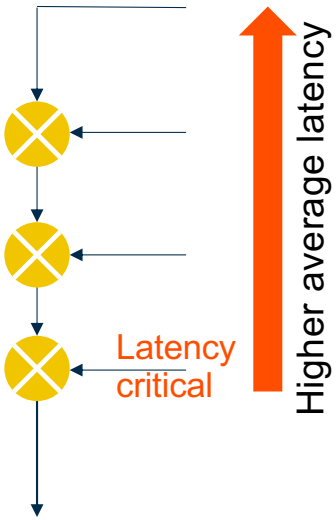
- **Bandwidth**
  - Measures the **amount of data** (typ. bytes) transferred between a initiator and a target over a **period of time** (typically 1 second)
- **Latency**
  - Measures the amount of time (typically in seconds or cycles) taken between two events (e.g., a request event and a response event)
- **Service Jitter**
  - Statistical distribution of the latencies as viewed by a master
- **Burstiness**
  - Irregularities in the flow

Bandwidth fairness

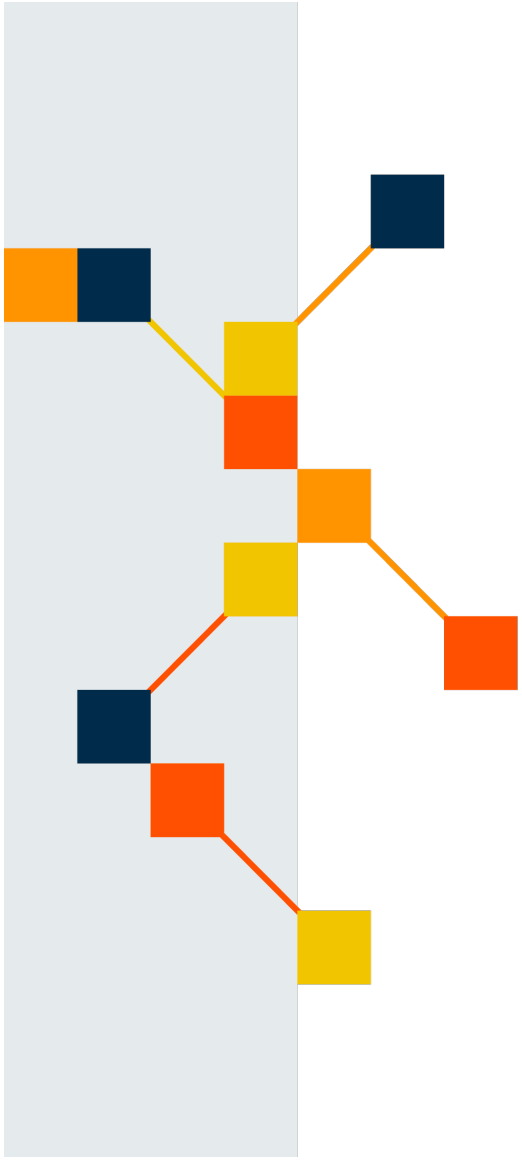


- Cascaded arbiters in distributed switch topologies can lead to great differences in how initiators can access a target
- How do NoCs handle this? Depends on the NoC ☺
  - Adaptive QoS mechanisms can mitigate this problem by regulating packet priority levels over time
  - Weighted fairness arbitration can also mitigate this effect
    - Can become tricky if state must be maintained over time so that the arbiter “remembers” the past to allow the weighting to apply even if there is no continuous contention.
- Configuration and effectiveness depends on the communication performance requirements of the agents within the system!

Latency control



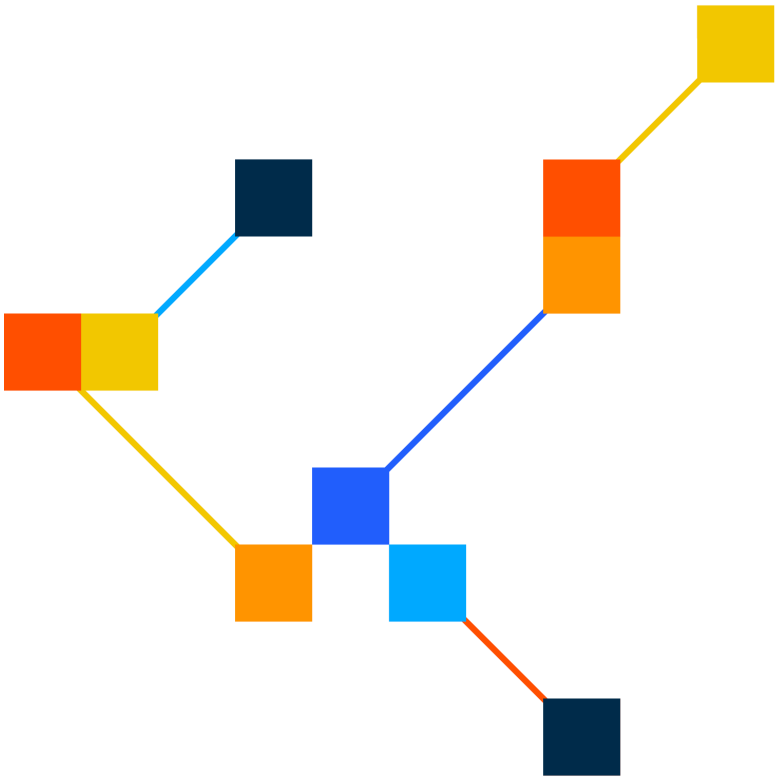
- Topology
  - Topology is the first means of controlling latency
  - The less the amount of contention, and buffers on the path, the lower the average latency
  - Attention: Deep buffers are bad for low average latency!
- Use of Virtual Channels
  - Special “high priority” VCs lower latency
  - High priority traffic “jumps” over the rest
- And other QoS mechanisms…
  - Lots of variation in solutions used across different NoCs



Summary

Your SoC, Your (**NoC**) Topology!

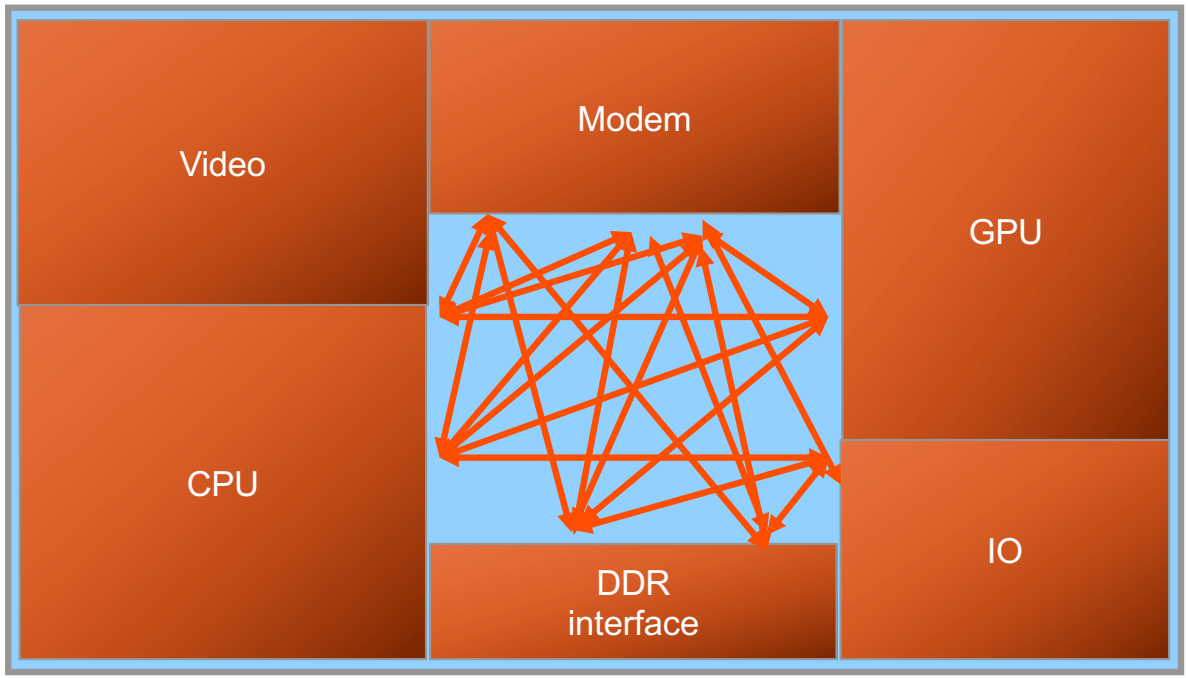
- SoCs are getting more complex as more IP is integrated on chip
- This additional complexity puts more demand on on-chip interconnects to handle the communication between all the IP
- NoC based interconnects help manage the complexity by enabling NoC designers to optimize the topology to the specific physical and performance constraints of their SoC
- **We've just scratched the surface on NoCs** – if you are interested in learning more about them, feel free to connect with us at Arteris!

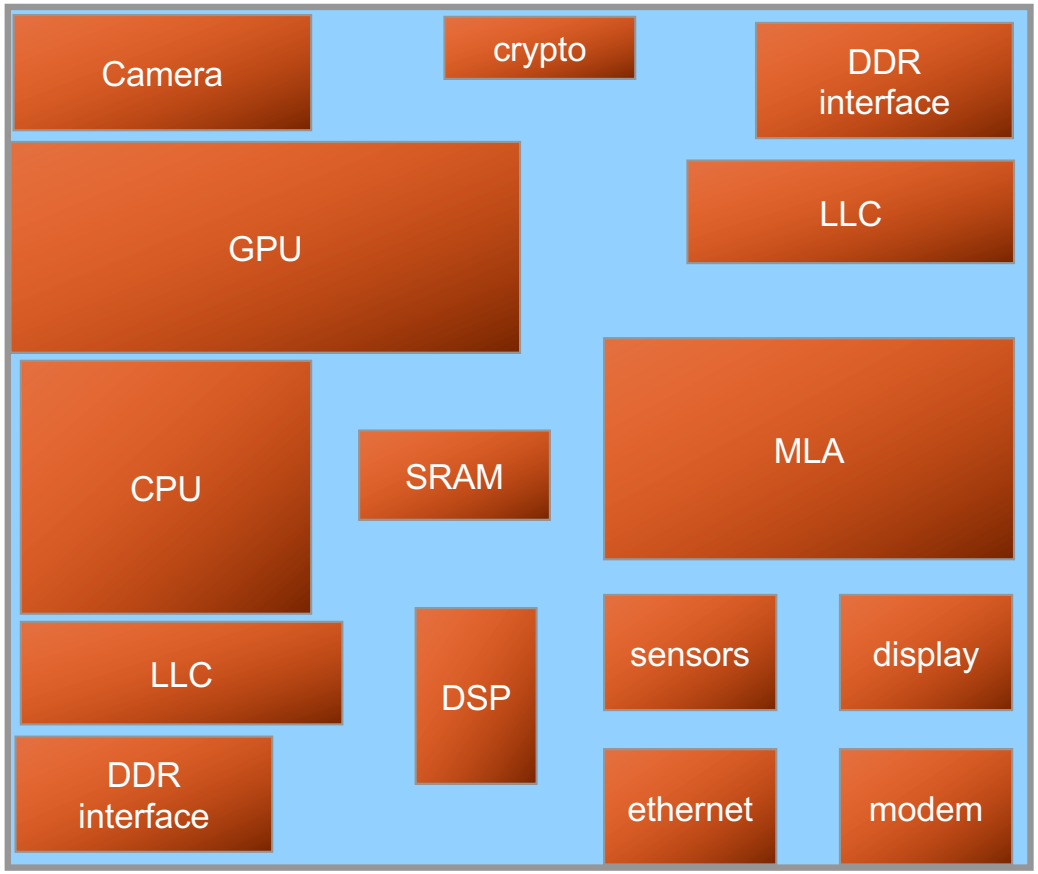


ARTERIS IP

THANK YOU

- Q&A


Video
Modem
GPU
CPU
IO
DDR
interface


Camera
crypto
DDR interface
GPU
LLC
CPU
SRAM
MLA
LLC
DSP
sensors
display
DDR interface
ethernet
modem