The Case for Embedding Networks-on-Chip in FPGA Architectures

Vaughn Betz
University of Toronto

With special thanks to
Mohamed Abdelfattah, Andrew Bitar
and Kevin Murray

Overview

- Why do we need a new system-level interconnect?
- Why an embedded NoC?
- How does it work?
- How efficient is it?

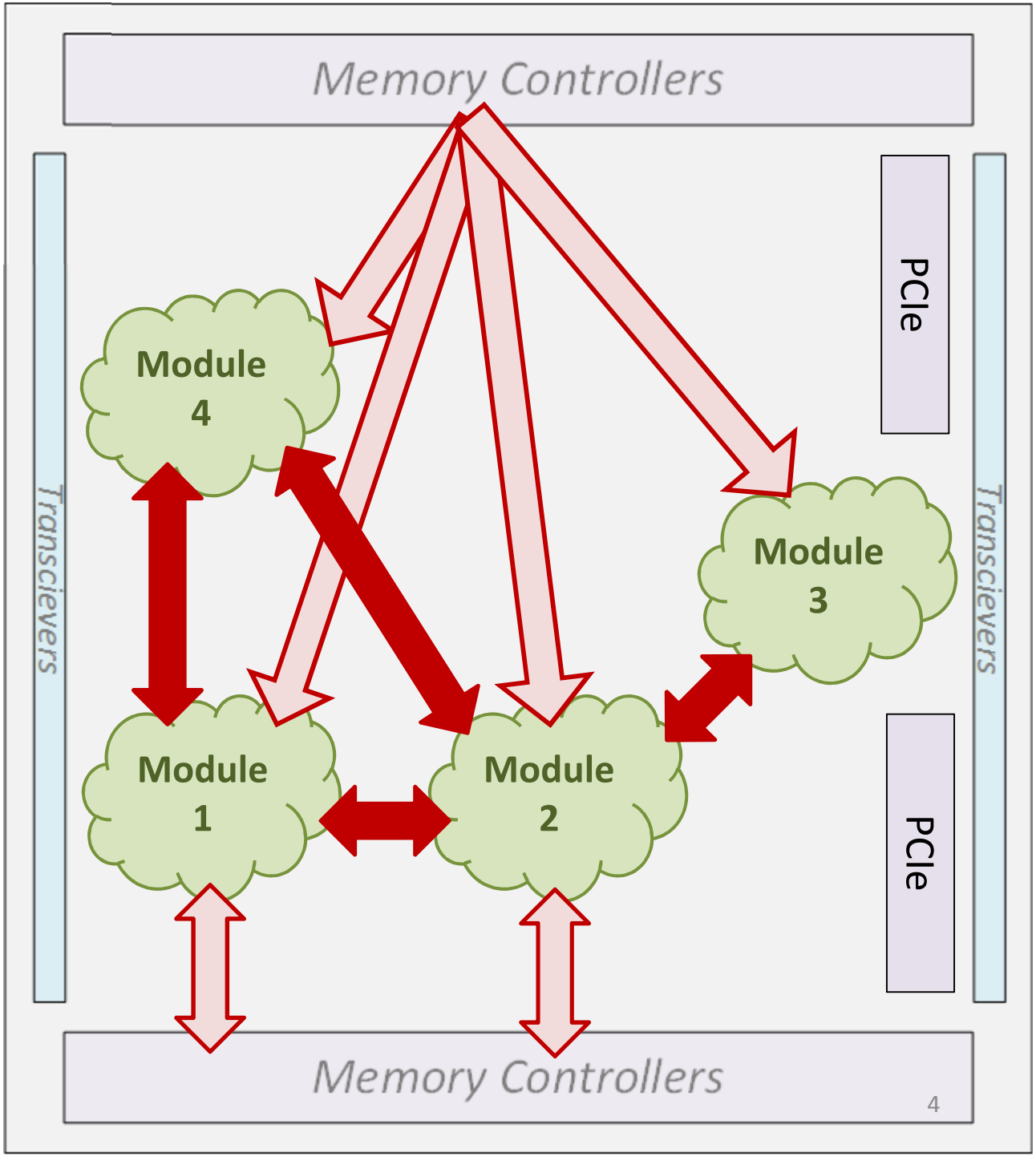
- Future trends and an embedded NoC
  - Data center FPGAs
  - Silicon interposers
  - Registered routing
  - Kernels → massively parallel accelerators

Why Do We Need a System Level Interconnect?

And Why a NoC?

Challenge

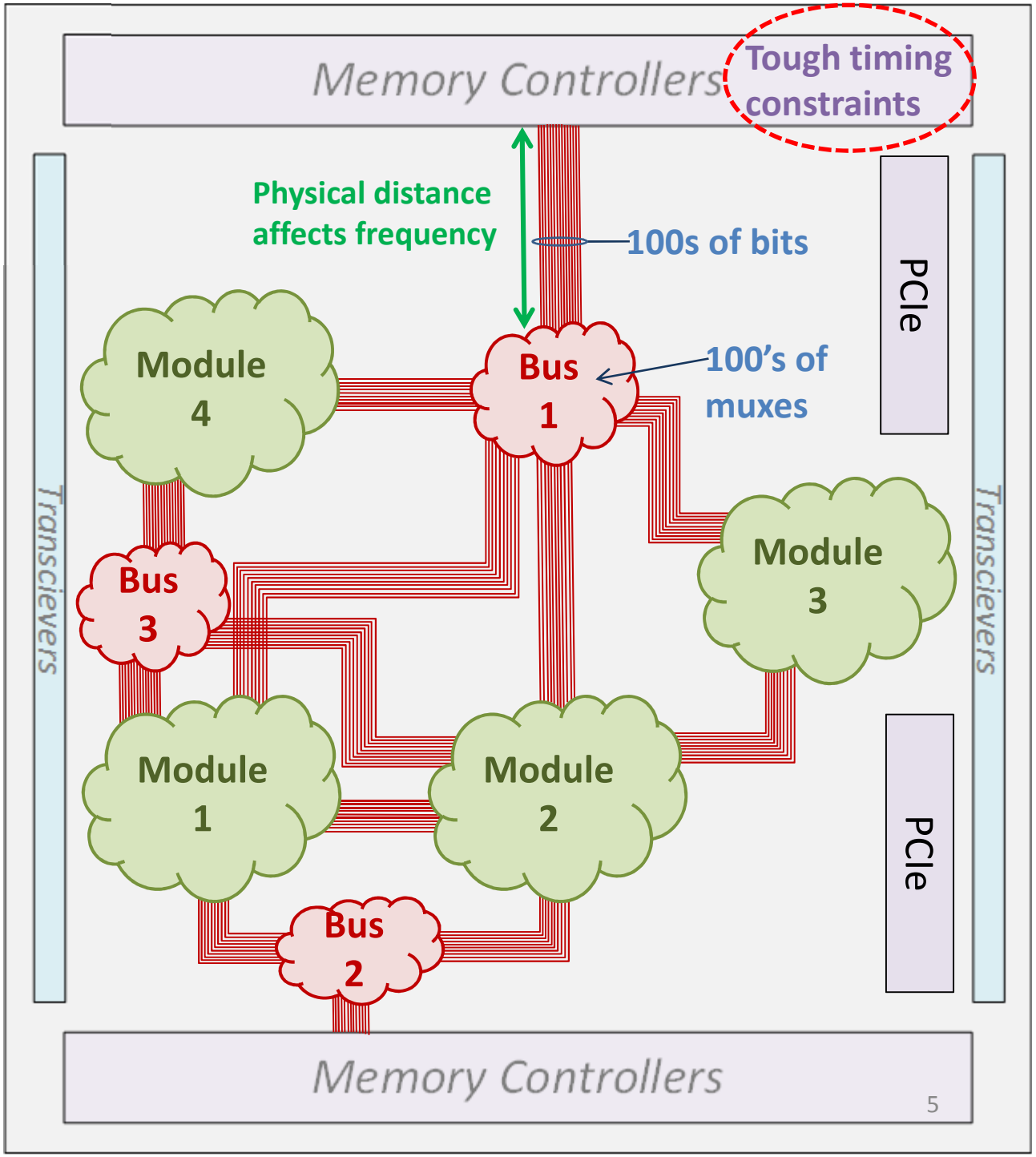
- Large Systems
- High bandwidth hard blocks & compute modules
- High on-chip communication



Today

Design soft bus per set of connections

- Wide links from single bit-programmable interconnect
- Muxing/arbitration/buffers on wide datapaths → big
- *Somewhat* unpredictable frequency → re-pipeline
- Buses are *unique* to the application → design time
- System-level buses → *costly*

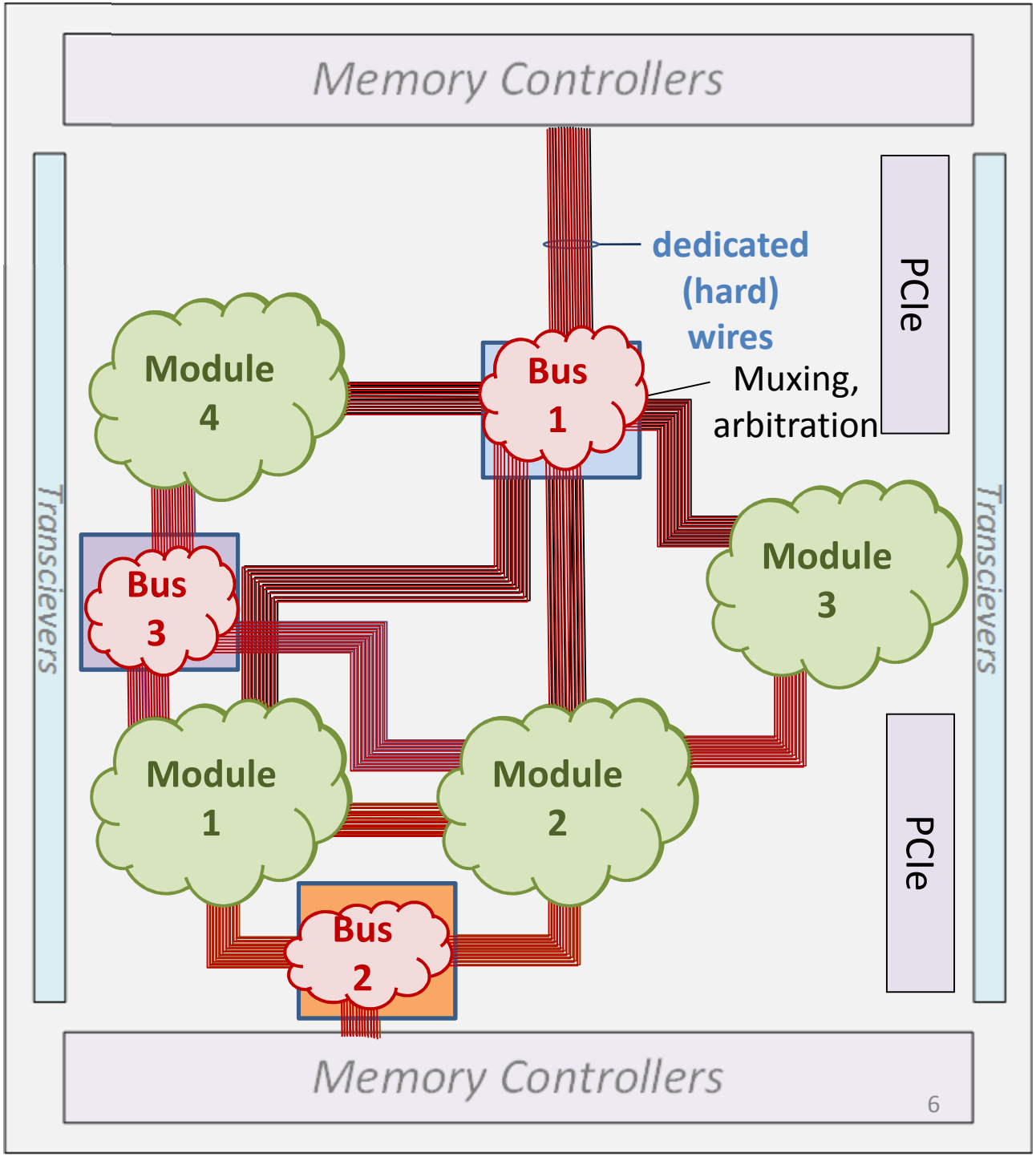


Hard Bus?

System-level interconnect in most designs? ✓

Costly in area & power? ✓

Usable by many designs?



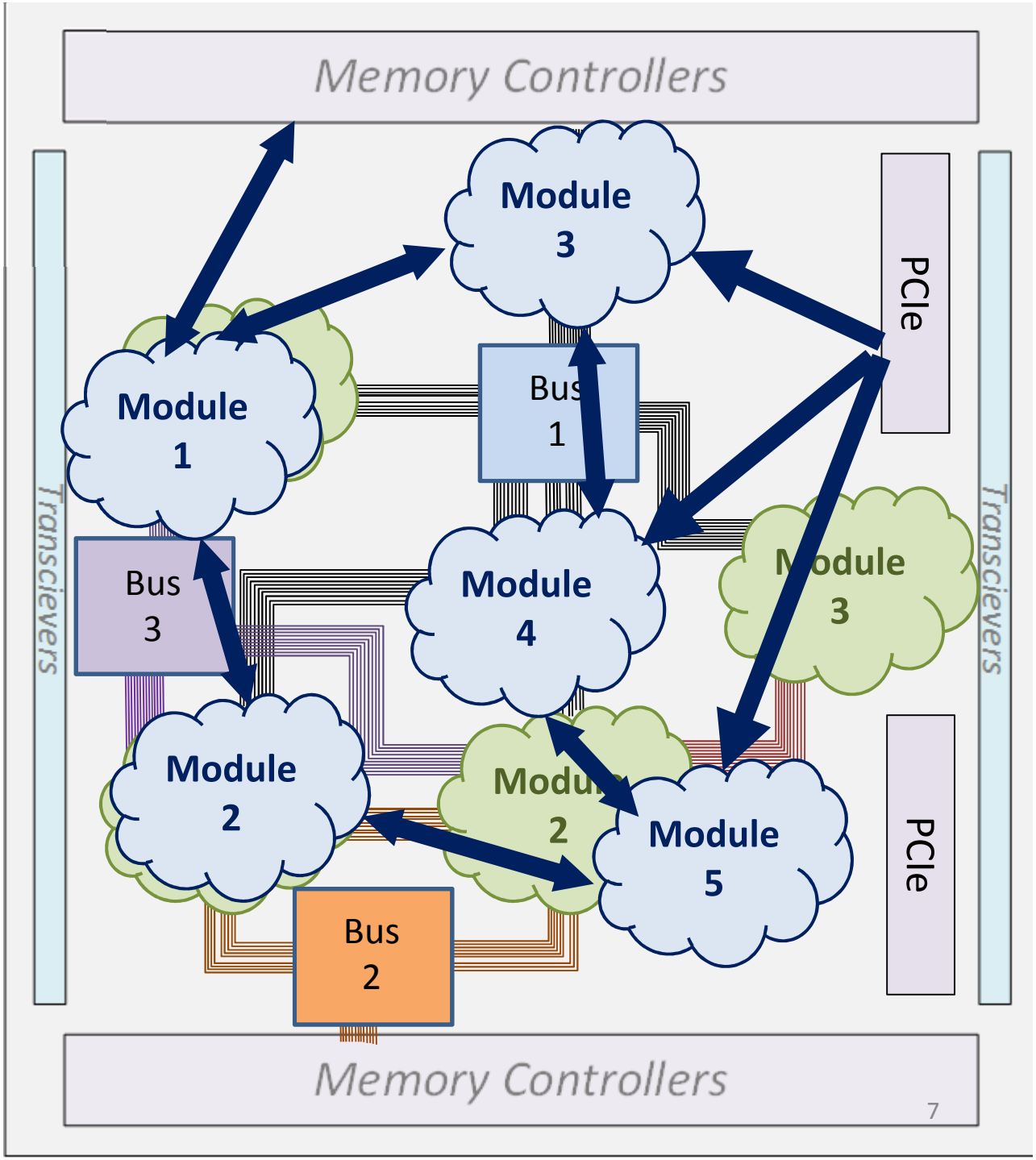
Hard Bus?

System-level interconnect in most designs? ✓

Costly in area & power? ✓

Usable by many designs? ✗

Not Reusable!
Too design specific



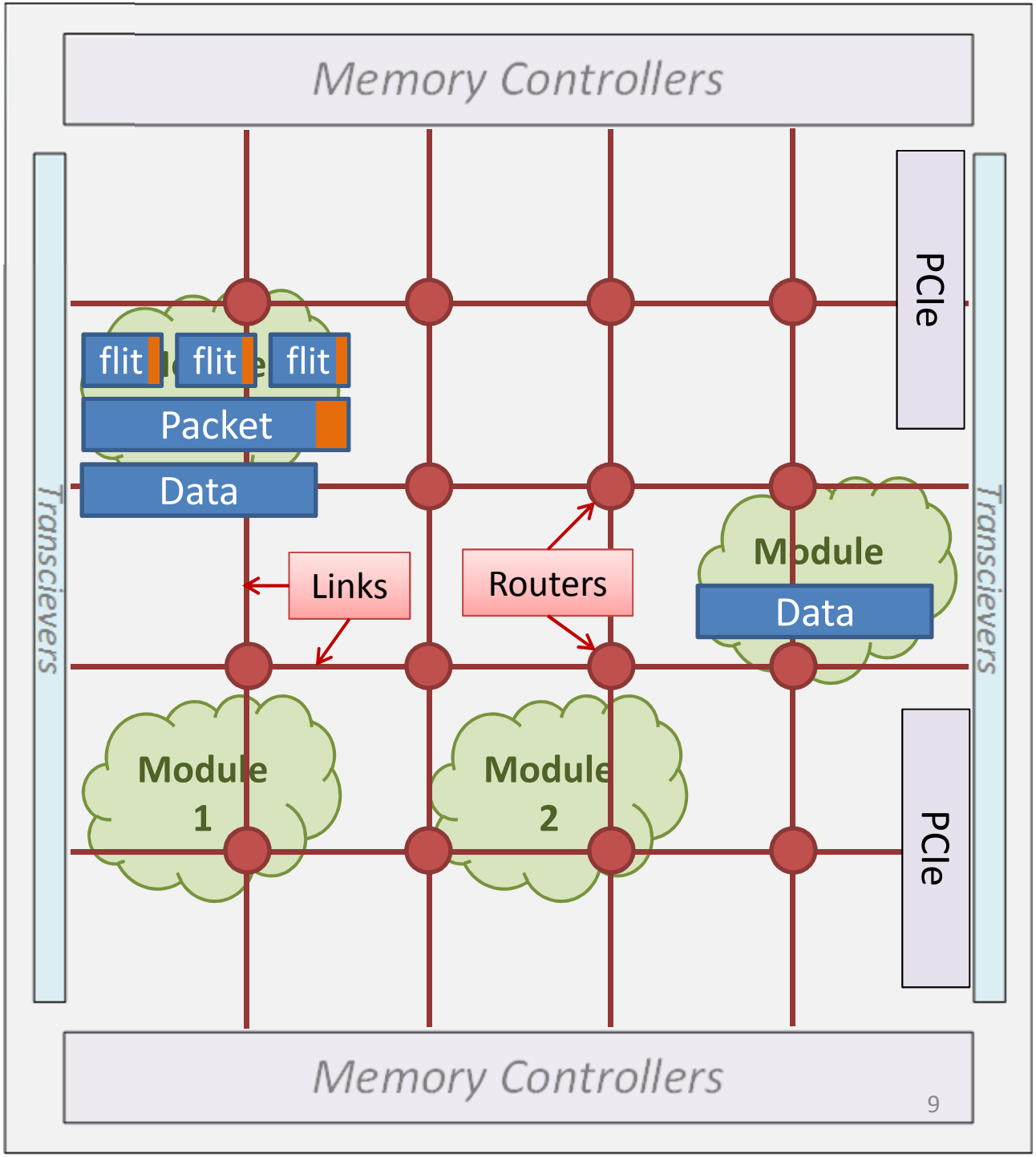
Needed: A More General System-Level Interconnect

1. Move data between **arbitrary** end-points
2. **Area efficient**
3. High bandwidth → match on-chip & **I/O bandwidths**

Network-on-Chip (NoC)

Embedded NoC

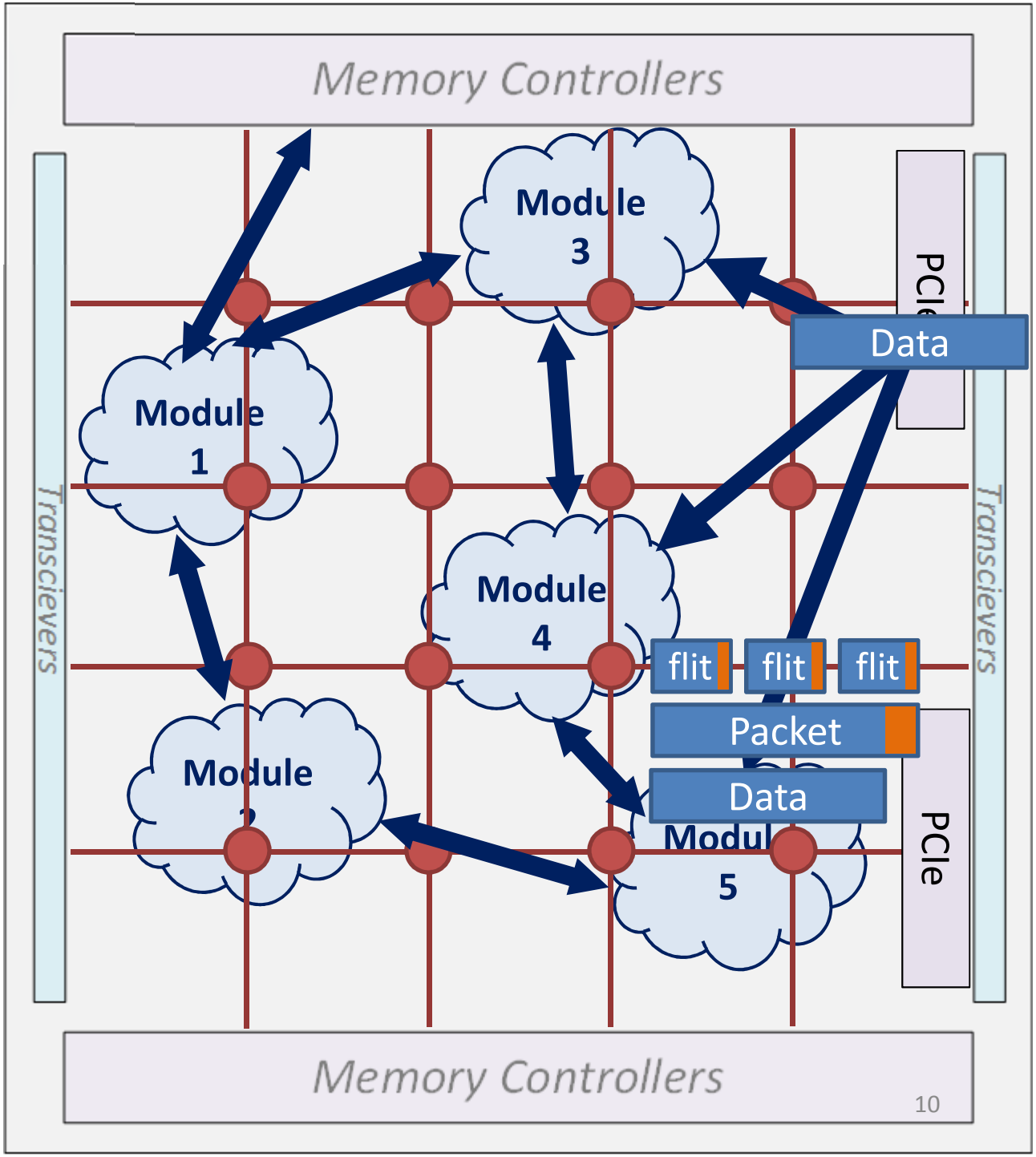
NoC=*complete* interconnect
- Data transport
- Switching
- Buffering



Embedded NoC

NoC=*complete* interconnect
- Data transport
- Switching
- Buffering

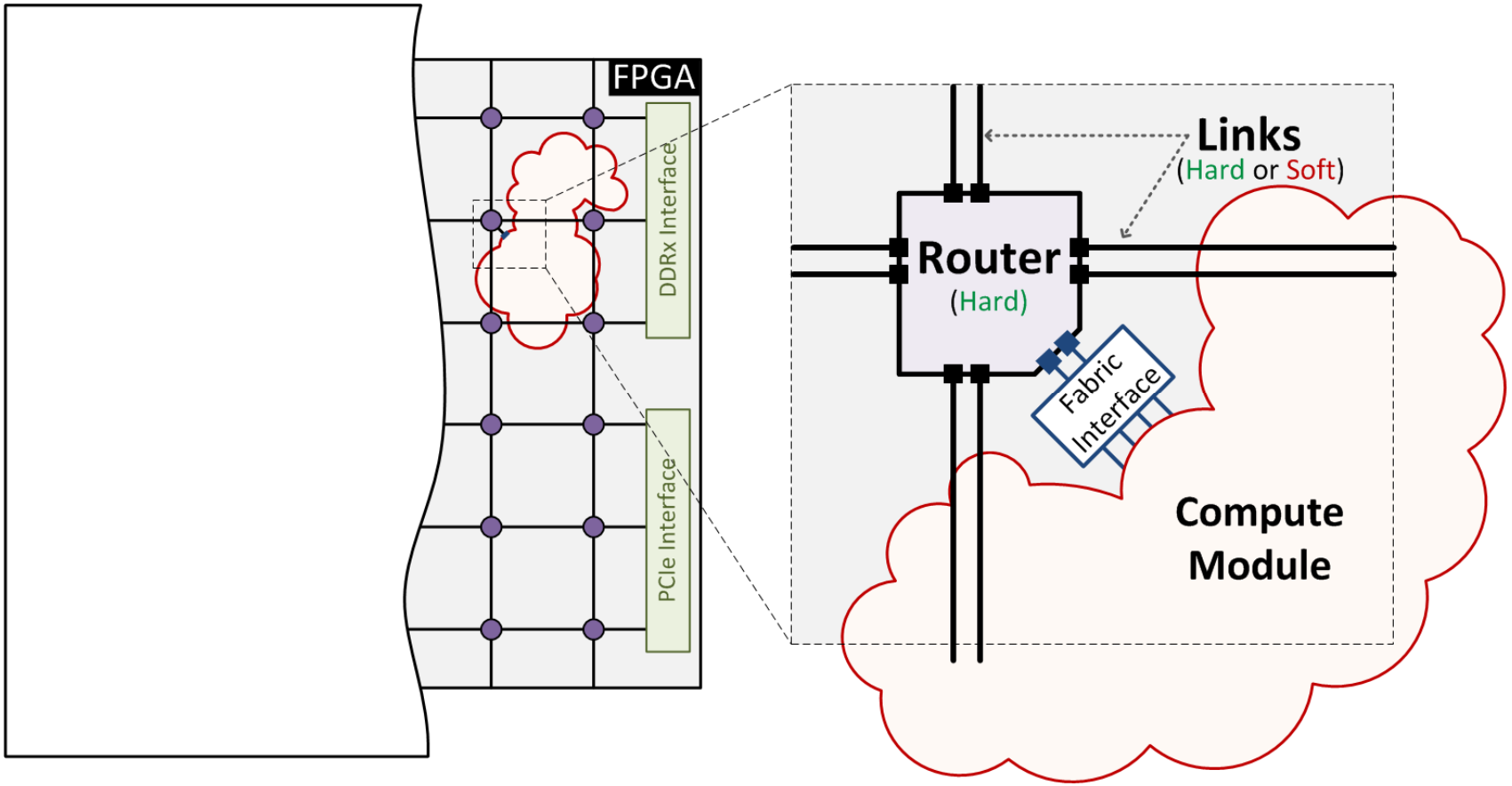
1. Moves data between arbitrary end points? ✓



Embedded NoC Architecture

How Do We Build It?

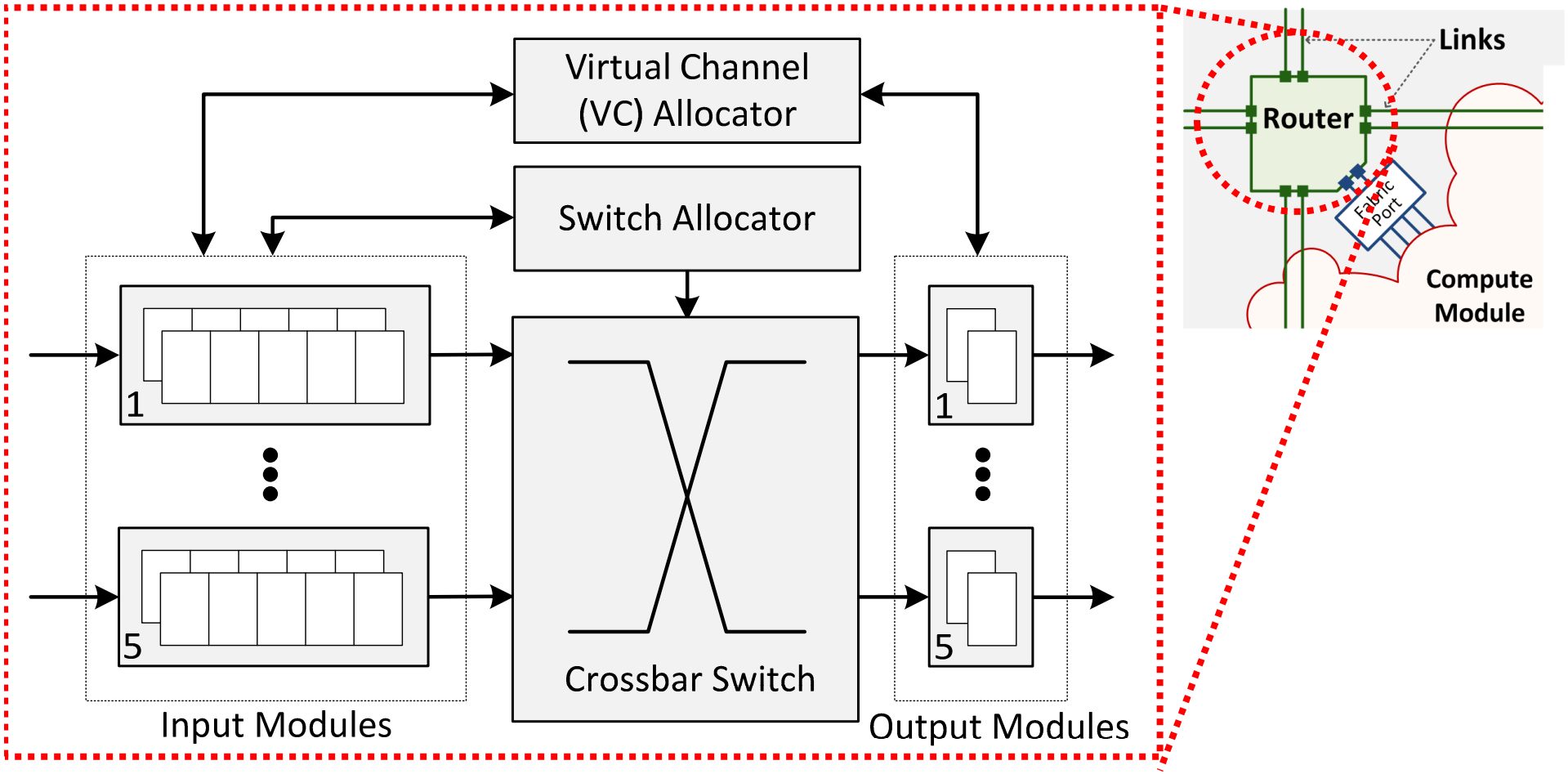
Routers, Links and Fabric Ports



- No hard boundaries

  → Build any size compute modules in fabric
- **Fabric interface:** flexible interface to compute modules

Router



- Full featured virtual channel router [D. Becker, Stanford PhD, 2012]

Must We Harden the Router?

- Tested: 32-bit wide ports, 2 VCs, 10 flit deep buffers
- 65 nm TSMC process standard cells vs. 65 nm Stratix III

	Soft	Hard
Area	4.1 mm^2 **(1X)**	0.14 mm^2 **(30X)**
Speed	166 MHz **(1X)**	943 MHz **(5.7X)**

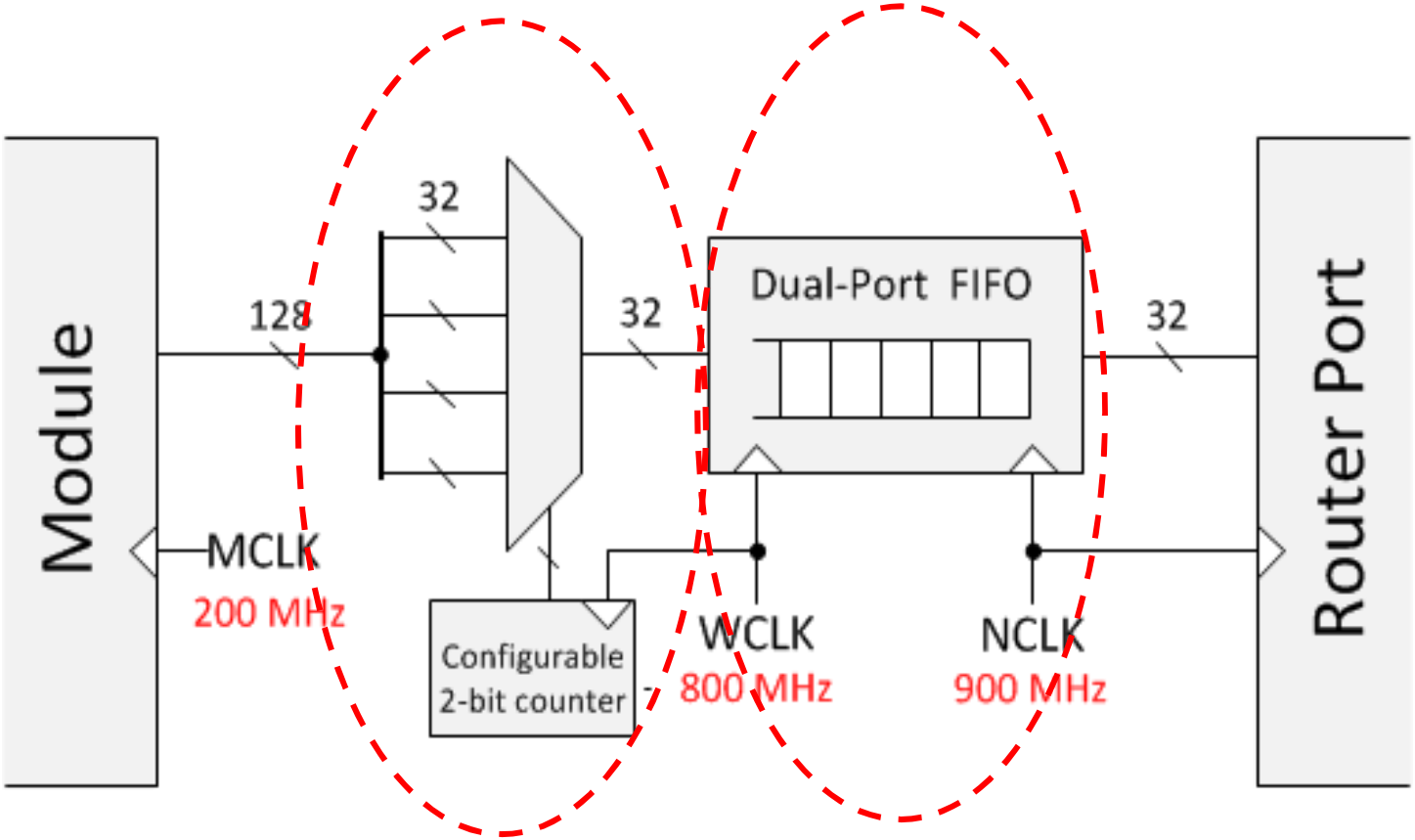
Hard: 170X throughput per area!

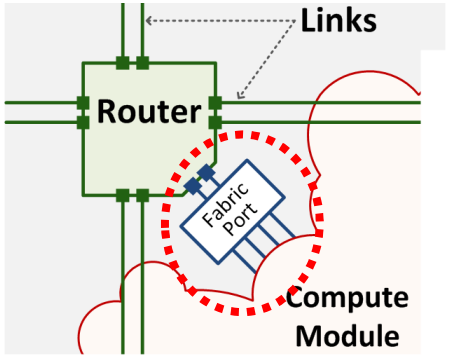
Harden the Routers?

- FPGA-optimized soft router?
  - [CONNECT, Papamichale & Hoe, FPGA 2012] and [Split/Merge, Huan & Dehon, FPT 2012]
    - ~2-3X throughput / area improvement with reduced feature set
  - [Hoplite, Kapre & Gray, FPL 2015]
    - Larger improvement with very reduced features / guarantees
- Not enough to close 170X gap with hard
- Want ease of use → full featured

Hard Routers

Fabric Interface

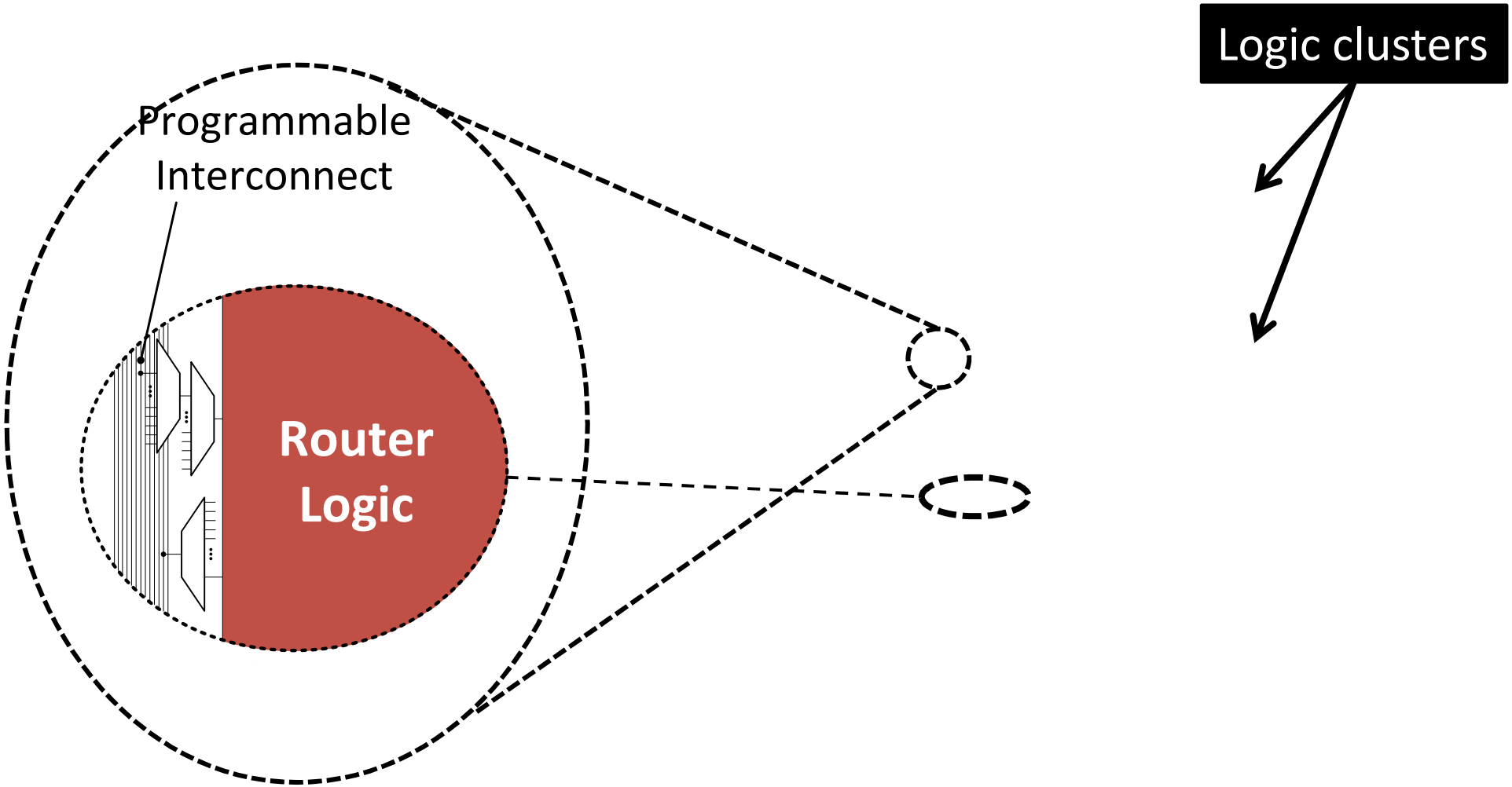




- 200 MHz module, 900 MHz router?
- Configurable time-domain mux / demux: match bandwidth
- Asynchronous FIFO: cross clock domains

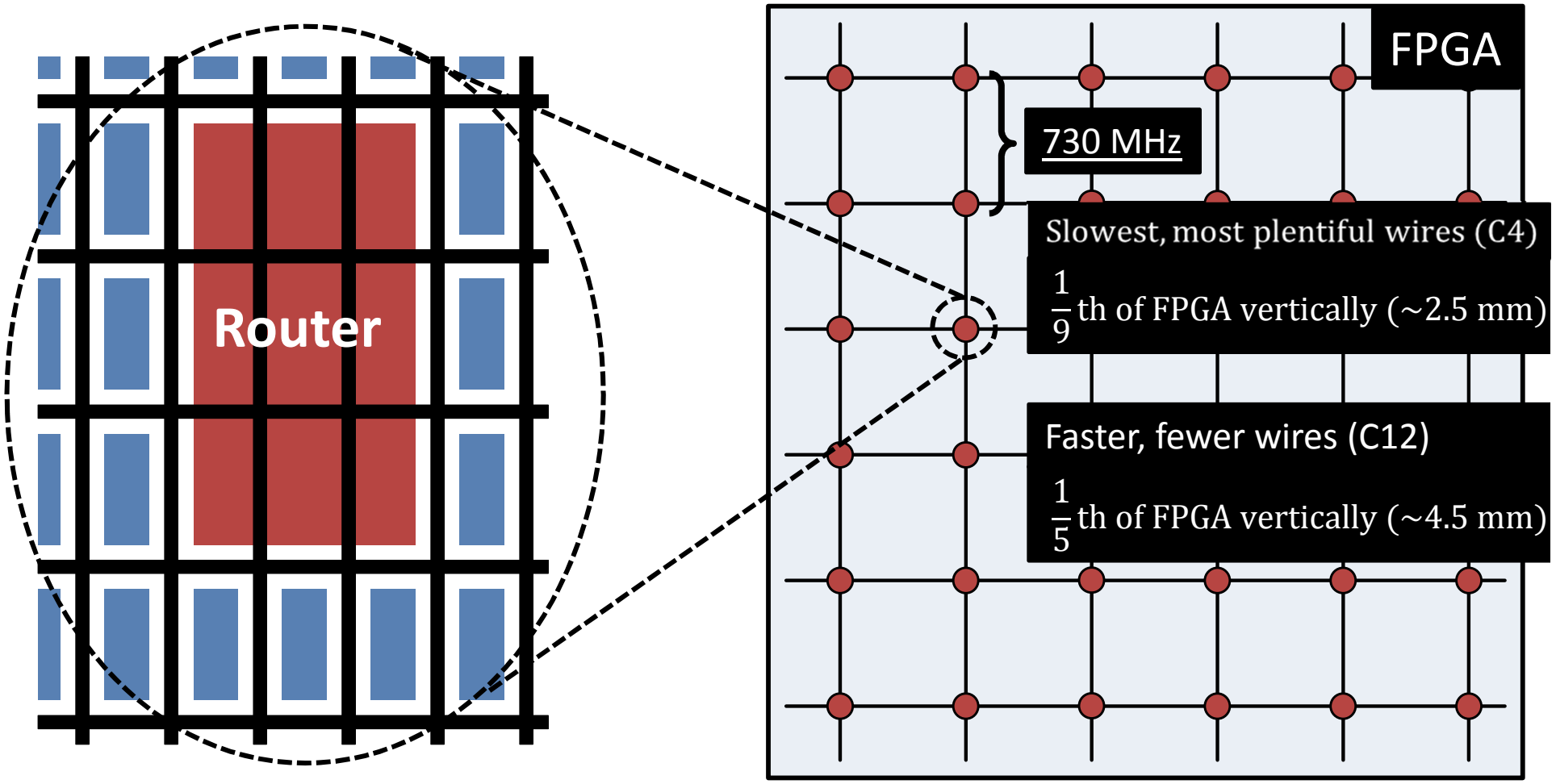
→ Full NoC bandwidth, w/o clock restrictions on modules

Hard Routers/Soft Links



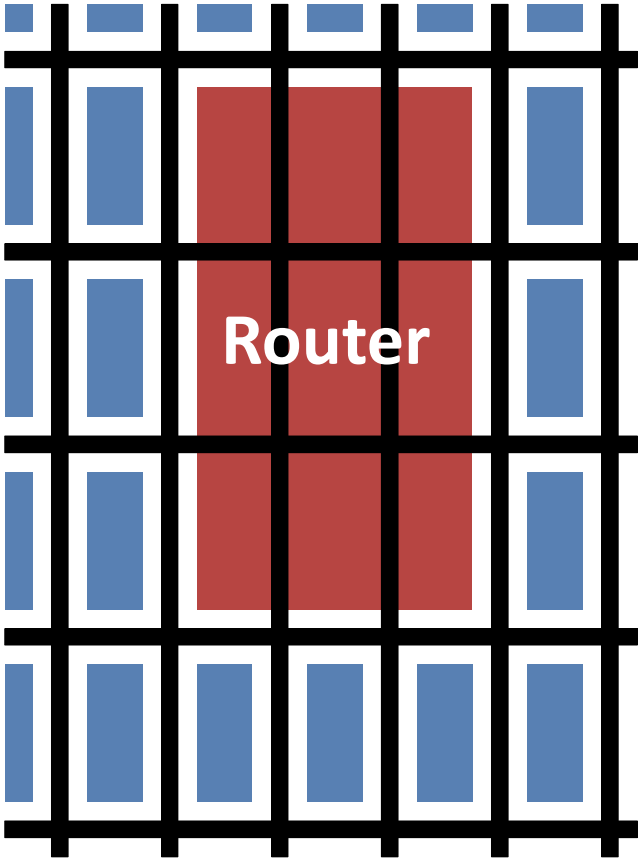
- Same I/O mux structure as a logic block – 9X the area
- Conventional FPGA interconnect between routers

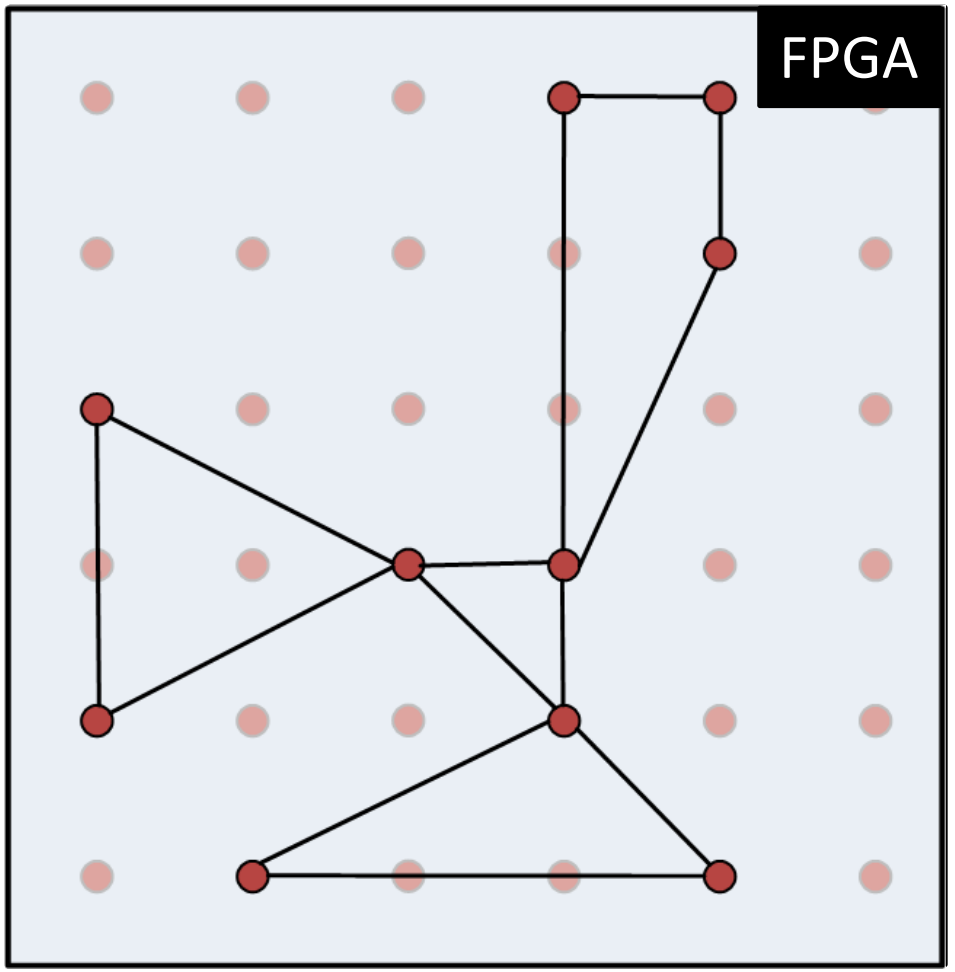
Hard Routers/Soft Links



- Same I/O mux structure as a logic block – 9X the area
- Conventional FPGA interconnect between routers

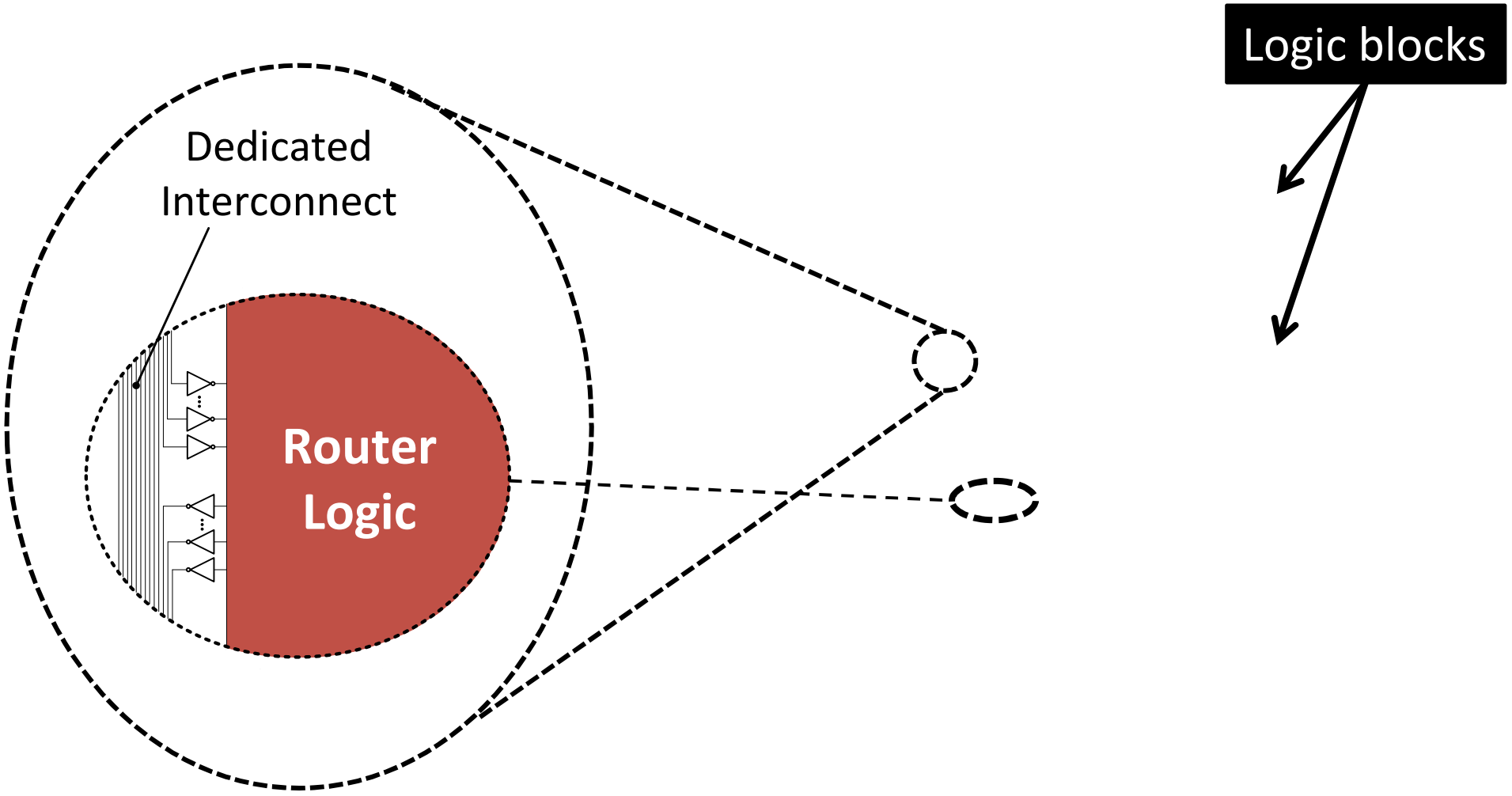
Hard Routers/Soft Links





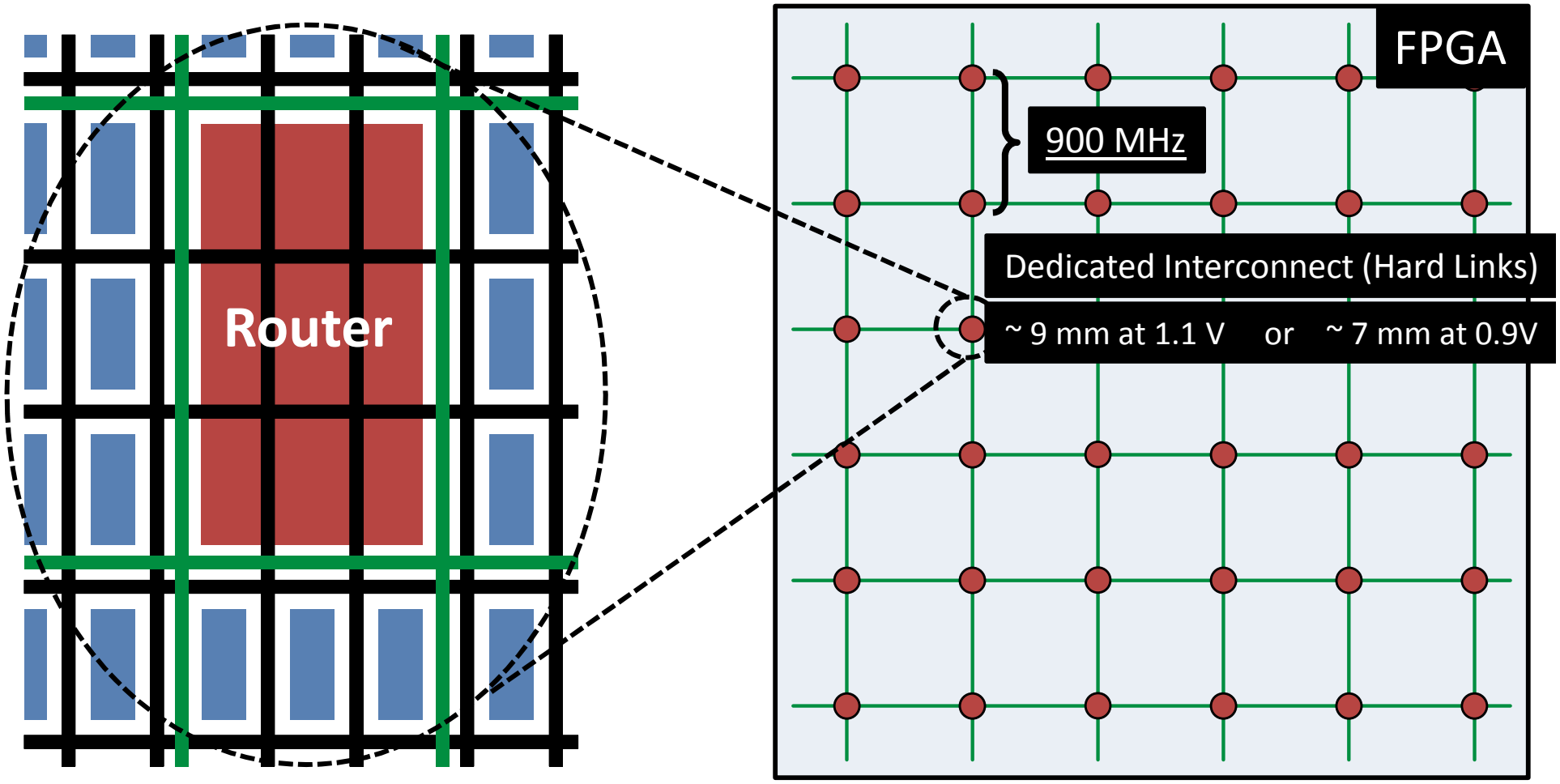
Assumed a mesh → Can form any topology

Hard Routers/Hard Links



- Muxes on router-fabric interface only – 7X logic block area
- Dedicated interconnect between routers → Faster/Fixed

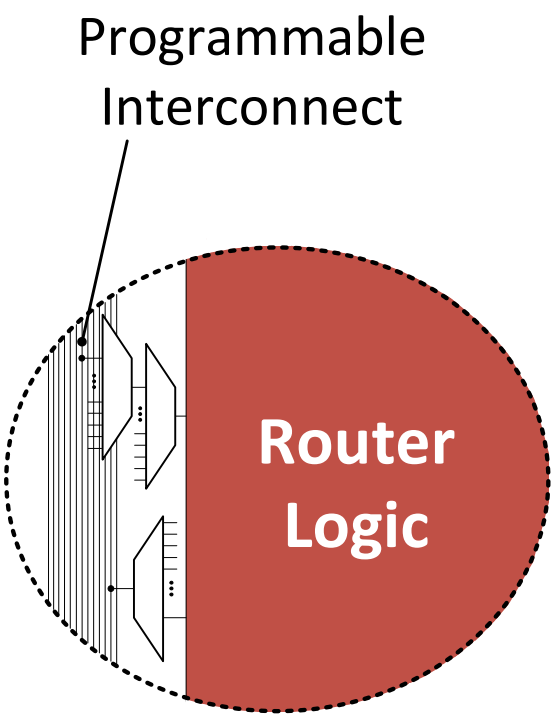
Hard Routers/Hard Links

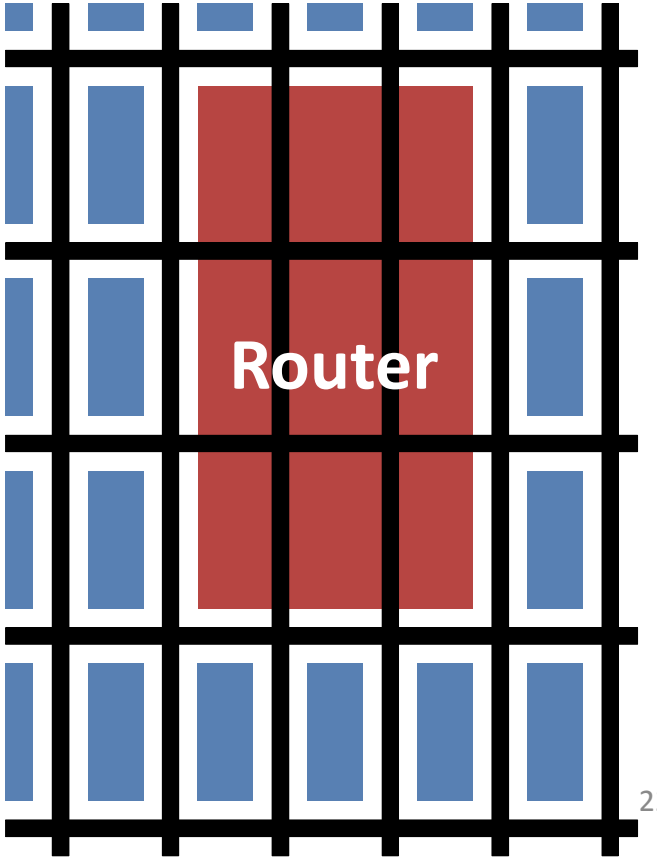


- Muxes on router-fabric interface only – 7X logic block area
- Dedicated interconnect between routers → Faster/Fixed

Hard NoCs

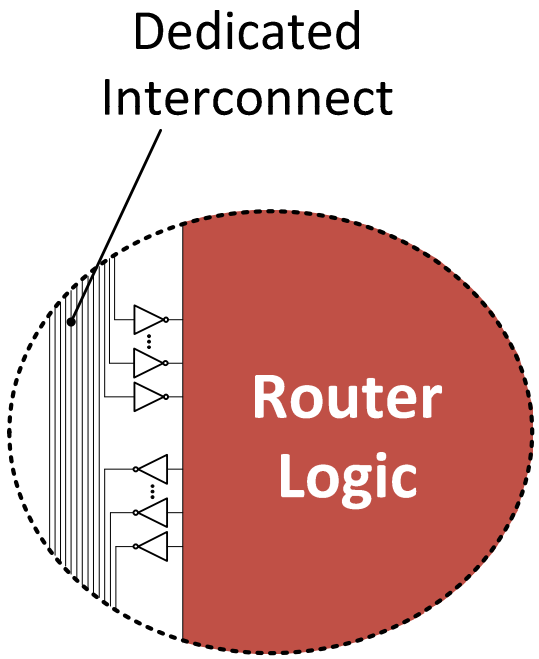
	Soft	Hard (+ Soft Links)	Hard (+ Hard Links)
Area	4.1 mm^2 **(1X)**		
Speed	166 MHz **(1X)**		
Power	**--**		

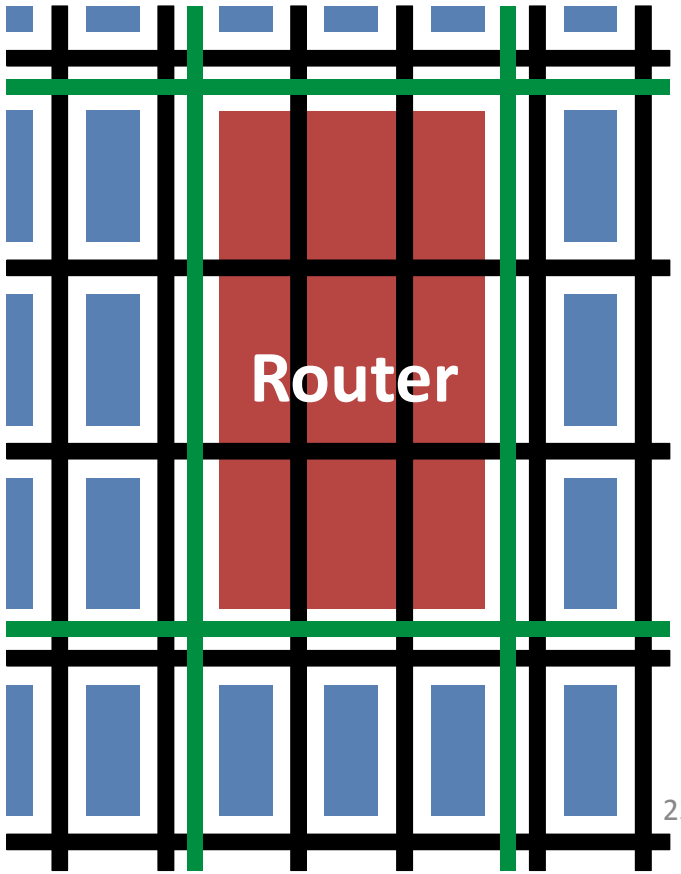




Hard NoCs

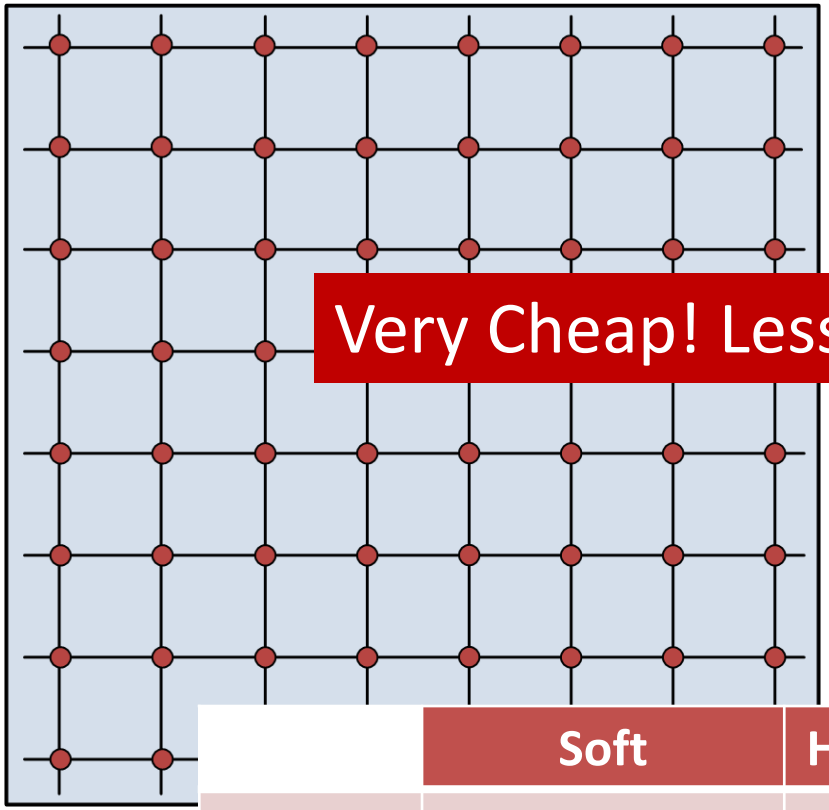
	Soft	Hard (+ Soft Links)	Hard (+ Hard Links)
Area	4.1 mm^2 **(1X)**	0.18 mm^2 = 9 LABs **(22X)**	0.15 mm^2 =7 LABs **(27X)**
Speed	166 MHz **(1X)**	730 MHz **(4.4X)**	943 MHz **(5.7X)**
Power	**--**	**(9X less)**	**(11X – 15X less)**





2. Area Efficient? ✓

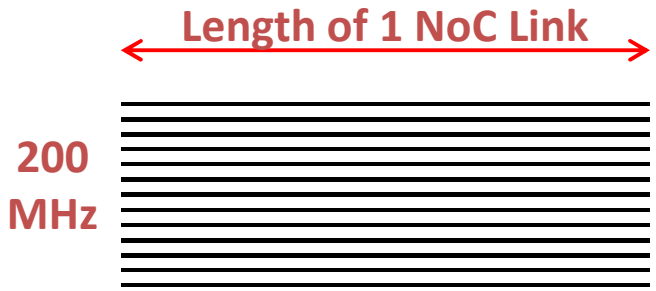
64-node, 32-bit wide NoC on Stratix V



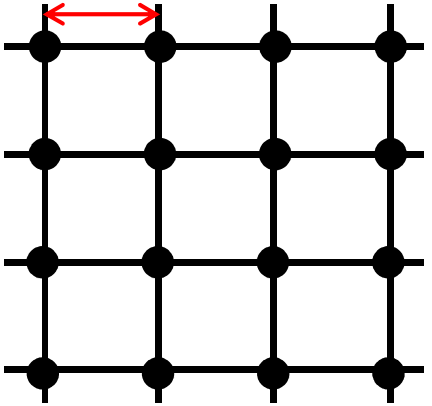
Very Cheap! Less than cost of 3 soft nodes

	Soft	Hard (+ Soft Links)	Hard (+ Hard Links)
Area	~12,500 LABs	576 LABs	448 LABs
%LABs	33 %	1.6 %	1.3%
%FPGA	12 %	0.6 %	0.45%

Power Efficient?



Compare to best case FPGA interconnect: point-to-point link

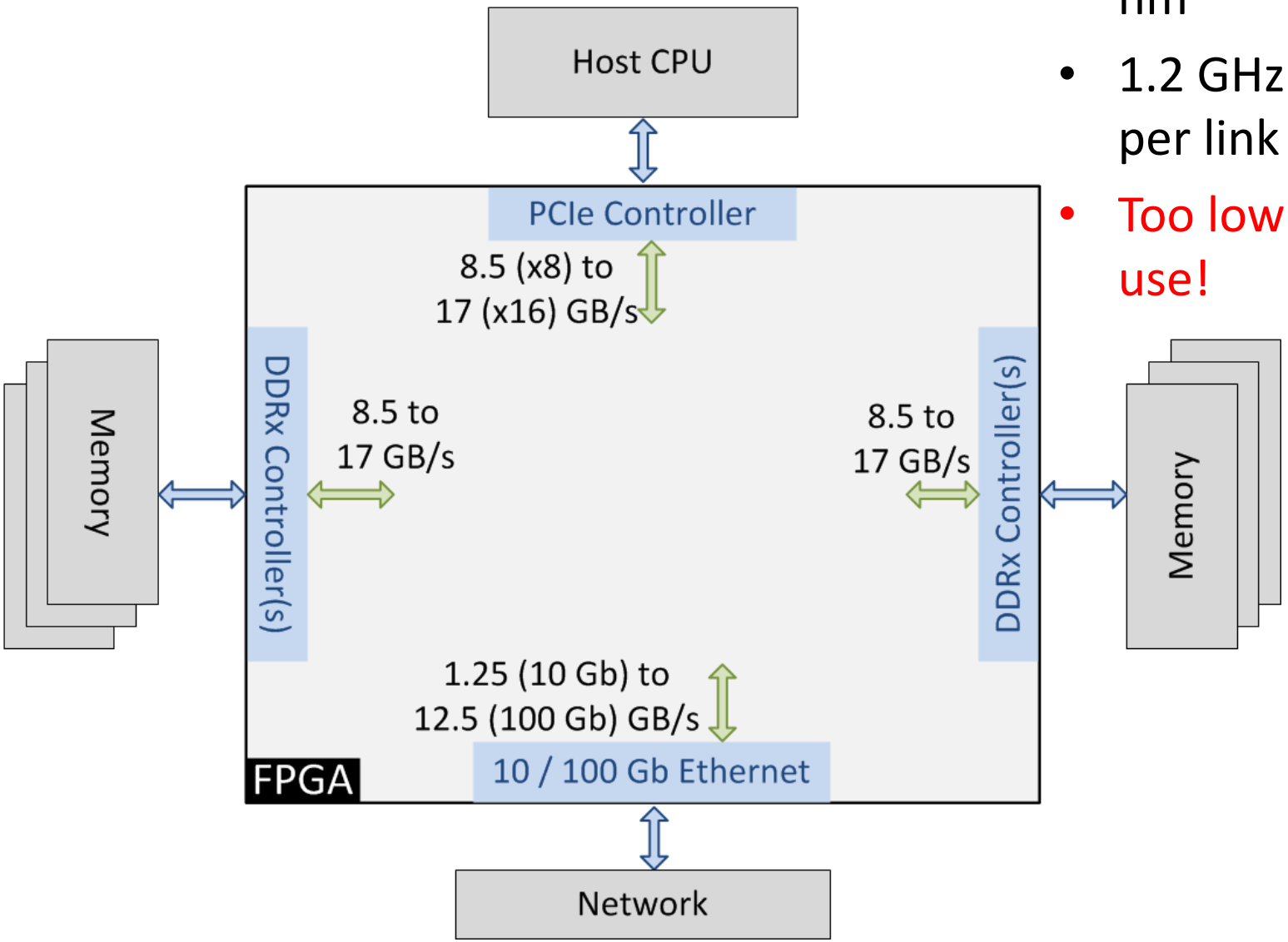


Interconnect	Description	Total Power (W)	Actual Aggregate BW (GB/s)	Energy per Data (mJ/GB)
FPGA Interconnect	10000 Wires	1.2	250	4.7

Hard and Mixed NoCs → Power Efficient

3. Match I/O Bandwidths?

- 32-bit wide NoC @ 28 nm
- 1.2 GHz → 4.8 GB/s per link
- Too low for easy I/O use!



3. Match I/O Bandwidths? ✓

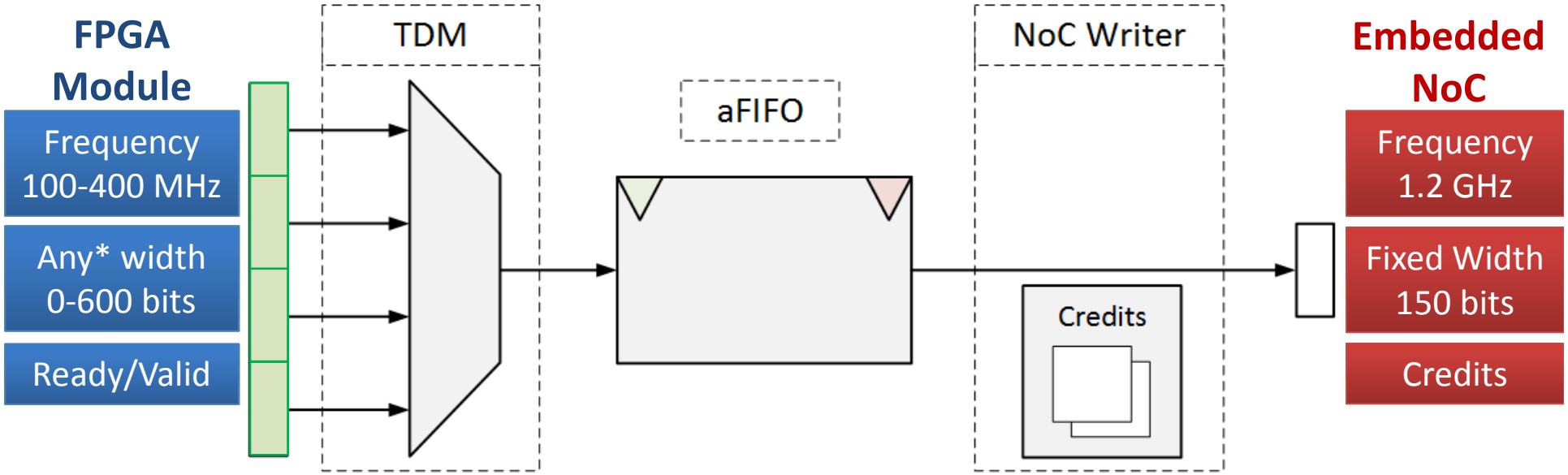
- Need higher-bandwidth links
  - 150 bits wide @ 1.2 GHz

  →22.5 GB/s per link
  - Can carry full I/O bandwidth on one link
- Want to keep cost low
  - Much easier to justify adding to an FPGA if cheap
    - E.g. Stratix I: 2% of die size for DSP blocks
    - First generation: not used by most customers, but 2% cost OK
  - Reduce number of nodes: 64 → 16
    - 1.3% of core area for a large Stratix V FPGA

NoC Usage & Application Efficiency Studies

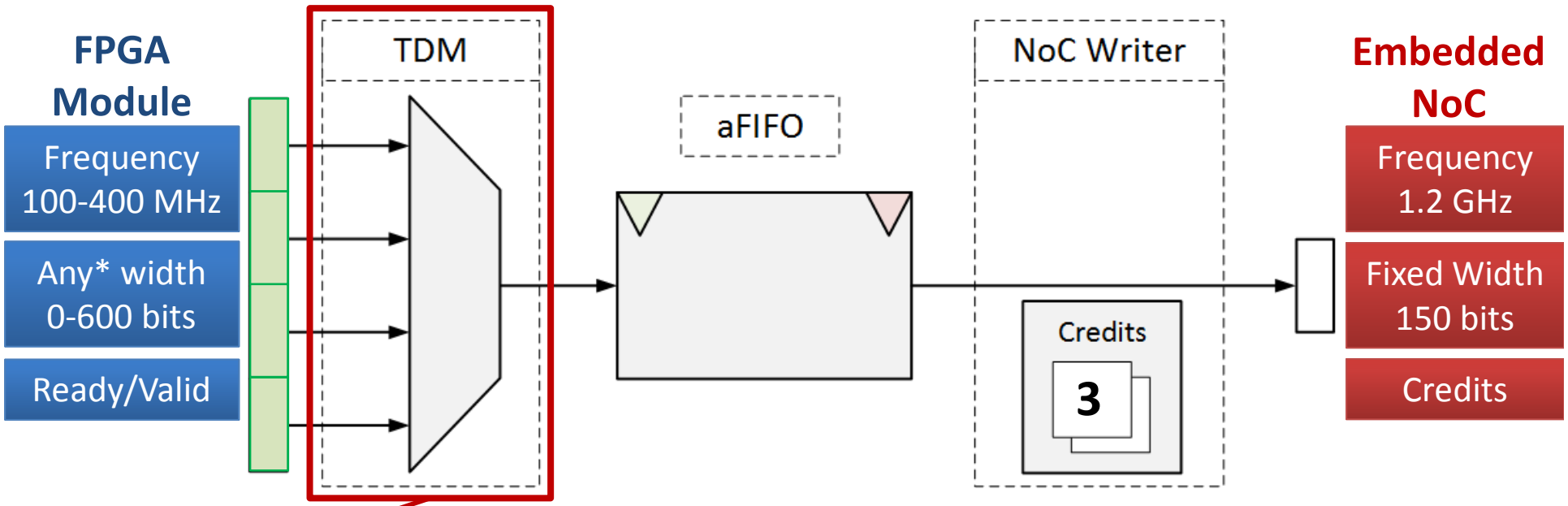
How Do We Use It?

FabricPort In



Width	# flits
0-150 bits	1
150-300 bits	2
300-450 bits	3
450-600 bits	4

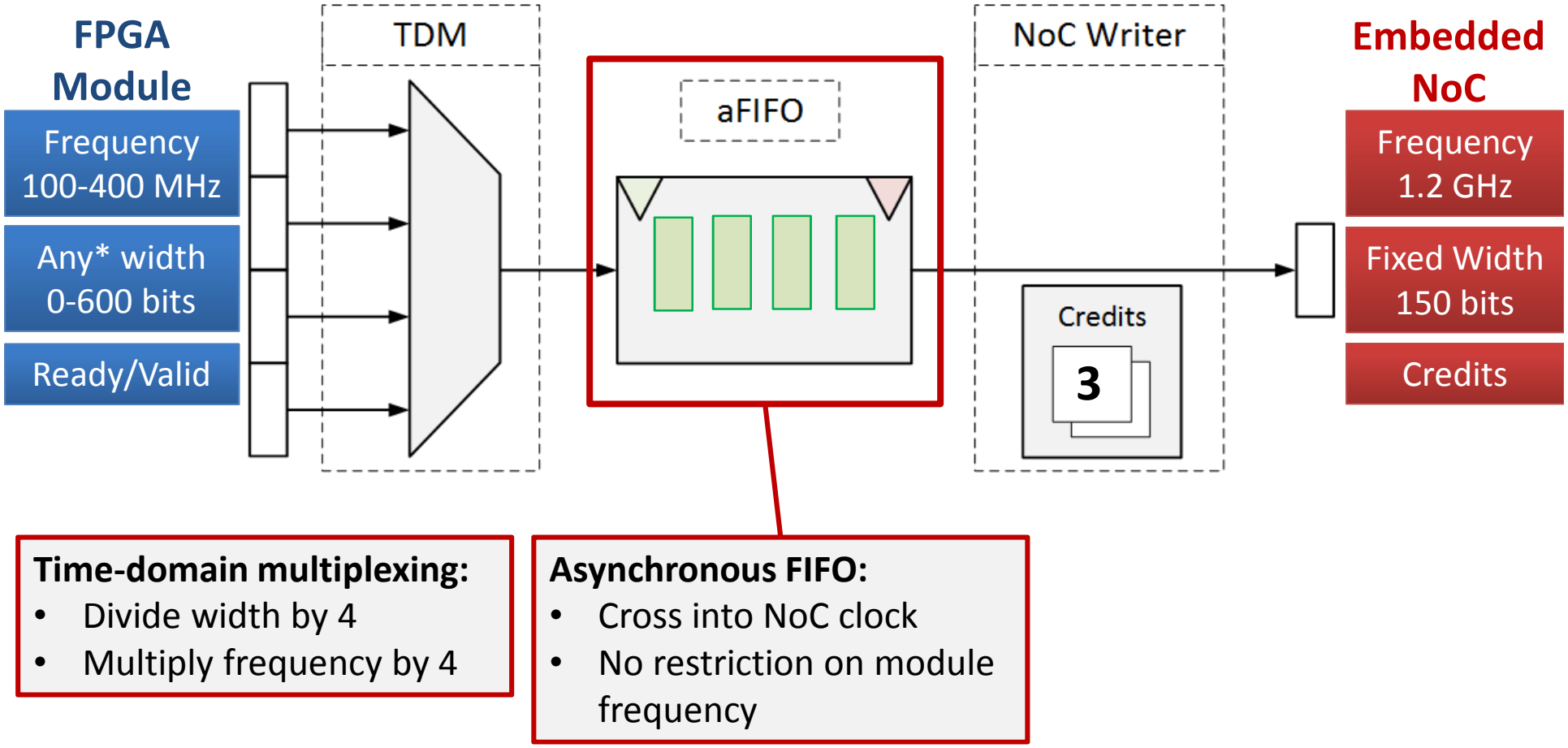
FabricPort In



Time-domain multiplexing:

- Divide width by 4
- Multiply frequency by 4

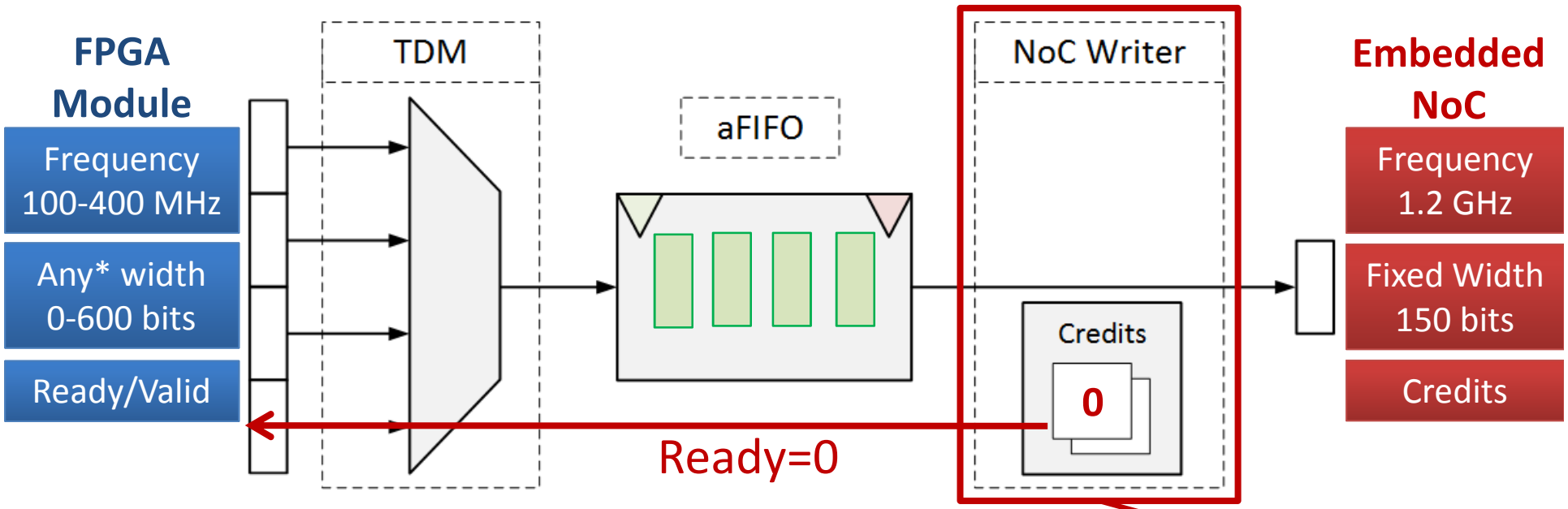
FabricPort In



Time-domain multiplexing:
- Divide width by 4
- Multiply frequency by 4

Asynchronous FIFO:
- Cross into NoC clock
- No restriction on module frequency

FabricPort In



Time-domain multiplexing:
- Divide width by 4
- Multiply frequency by 4

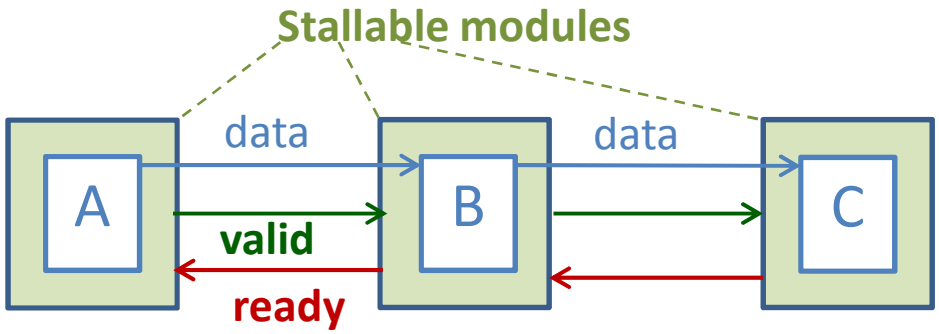
Asynchronous FIFO:
- Cross into NoC clock
- No restriction on module frequency

NoC Writer:
- Track available buffers in NoC Router
- Forward flits to NoC
- Backpressure

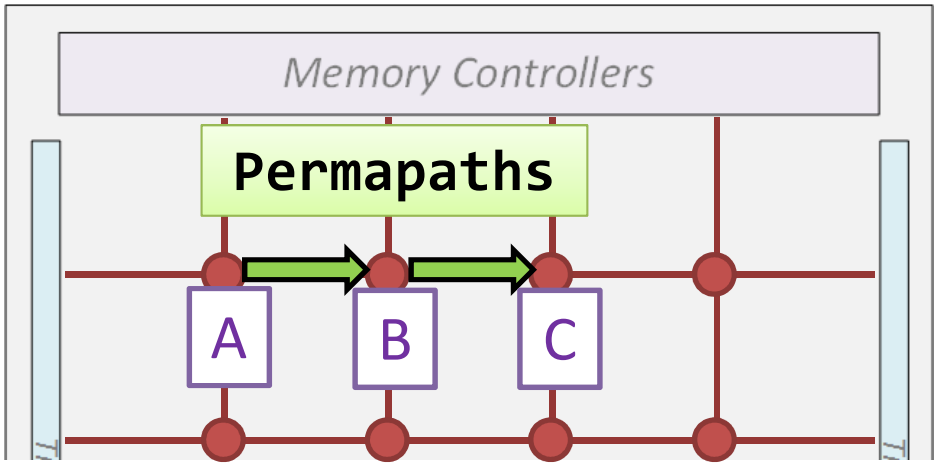
Input interface: flexible & *easy* for designers → little soft logic

Designer Use

- NoC has non-zero, usually variable latency
- Use on latency-insensitive channels



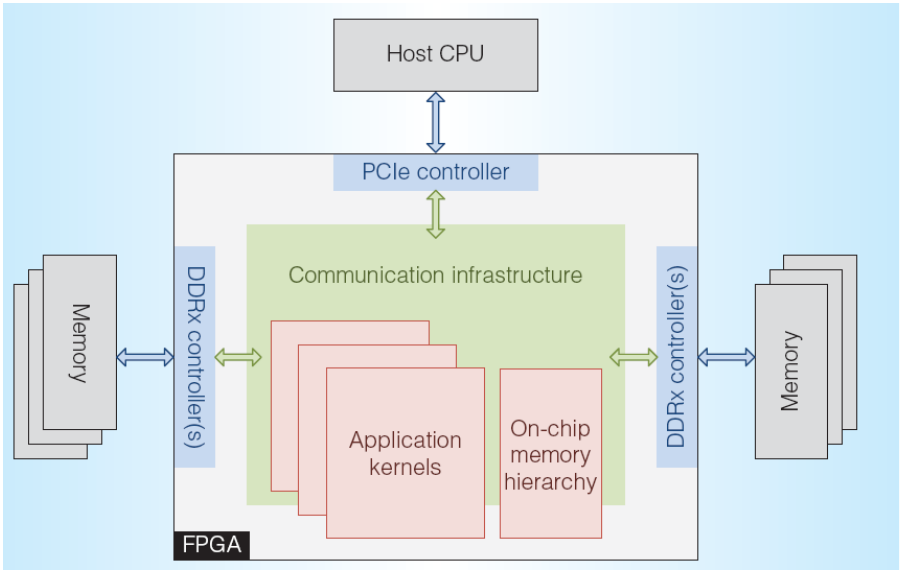
- With restrictions, usable for fixed-latency communication
  - Pre-establish and reserve paths
  - "Permapaths"



How Common Are Latency-Insensitive Channels?

- Connections to I/O
  - DDRx, PCIe, …
  - Variable latency
- Between HLS kernels
  - OpenCL channels / pipes
  - Bluespec SV
  - …
- Common design style between larger modules
  - And any module can be converted to use [Carloni et al, TCAD, 2001]

Widely used at system level, and use likely to increase



Packet Ordering

Multiprocessors

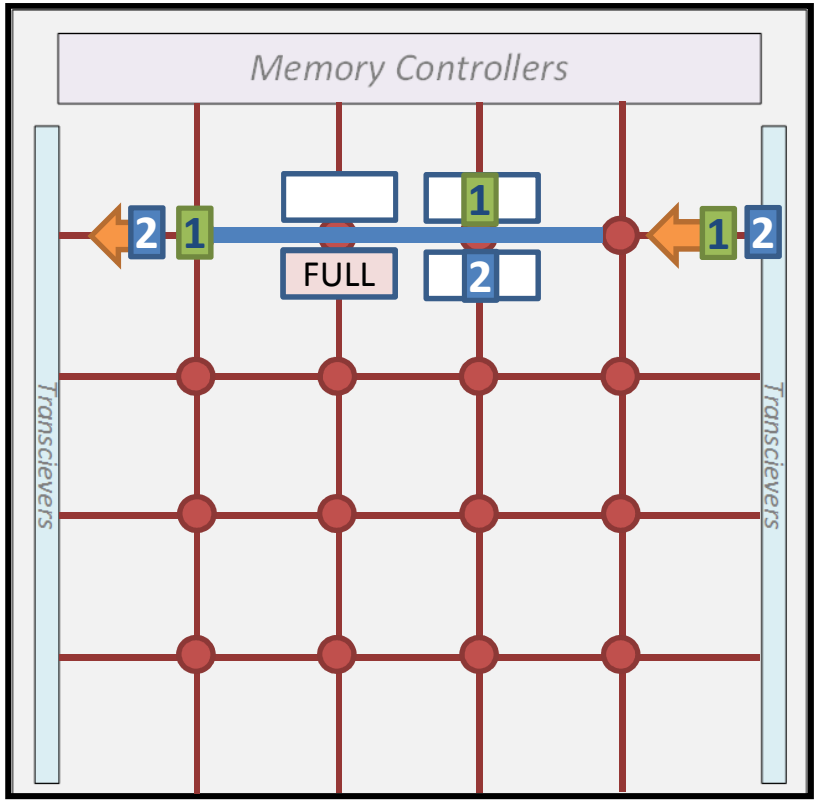
- Memory mapped
- Packets arrive out-of-order
  - Fine for cache lines
  - Processors have re-order buffers

FPGA Designs

- Mostly streaming
- Cannot tolerate reordering
  - Hardware expensive and difficult

RULE All packets with same src/dst must take **same NoC path**

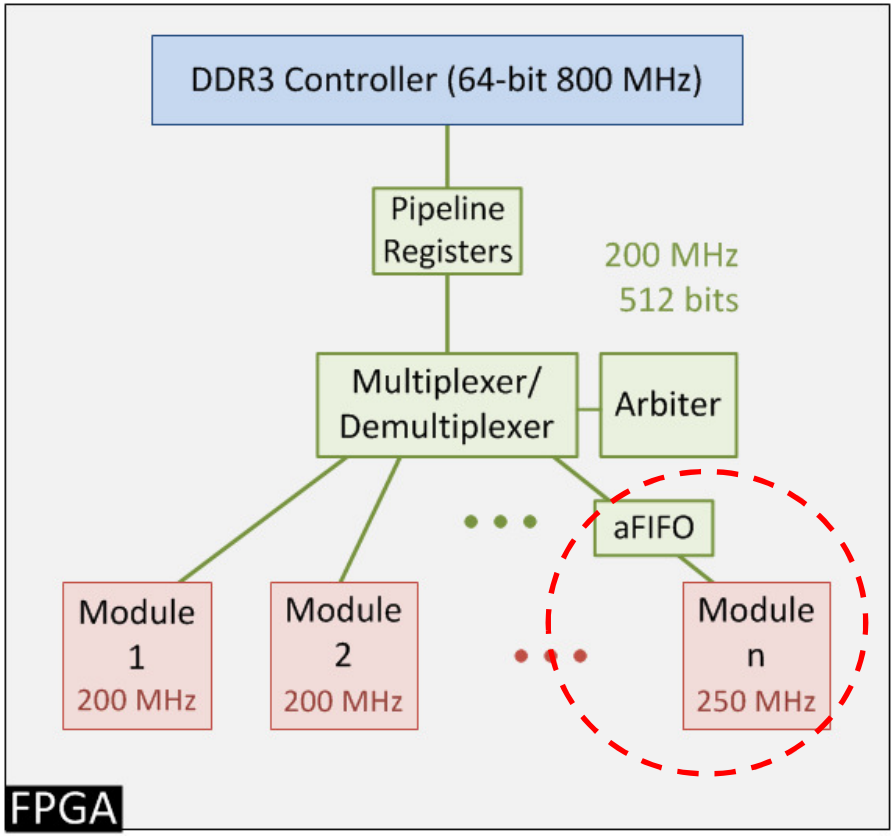
RULE All packets with same src/dst must take **same VC**

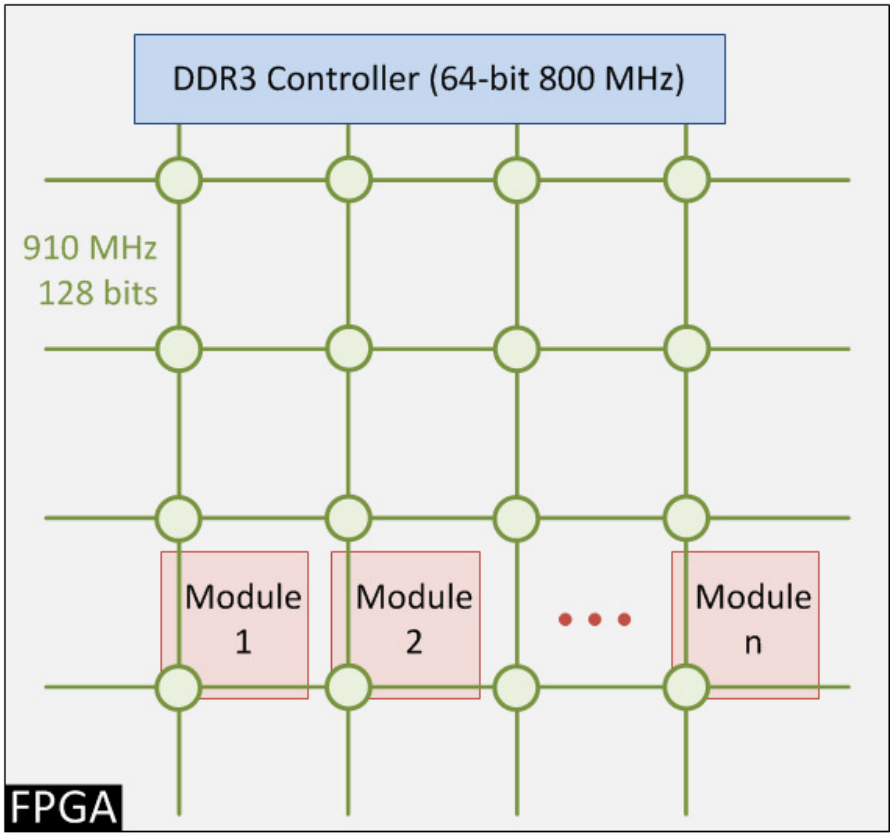


Application Efficiency Studies

How Efficient Is It?

1. Qsys vs. NoC

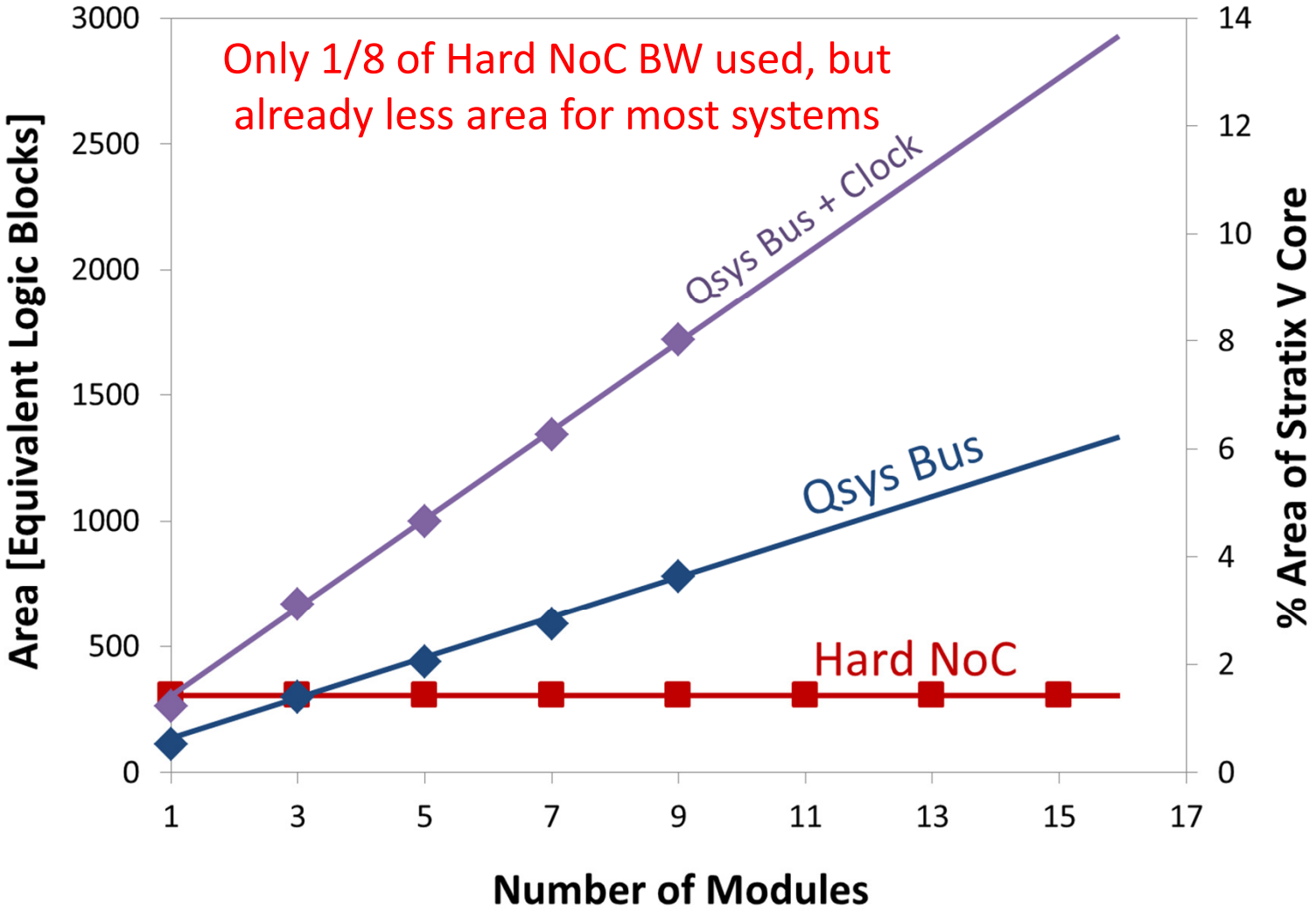




qsys: build logical bus from fabric

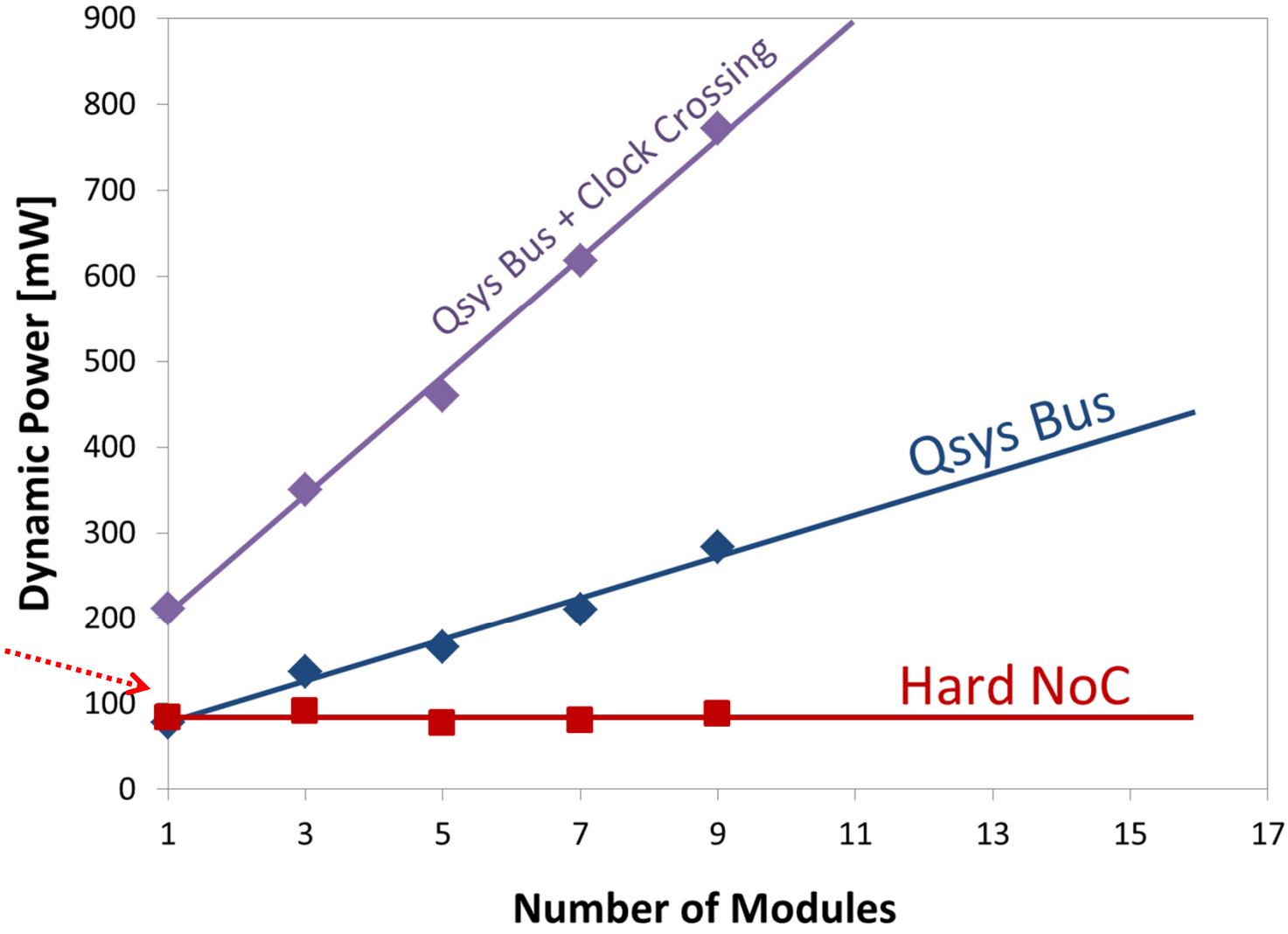
NoC: 16-nodes, hard routers & links

Area Comparison



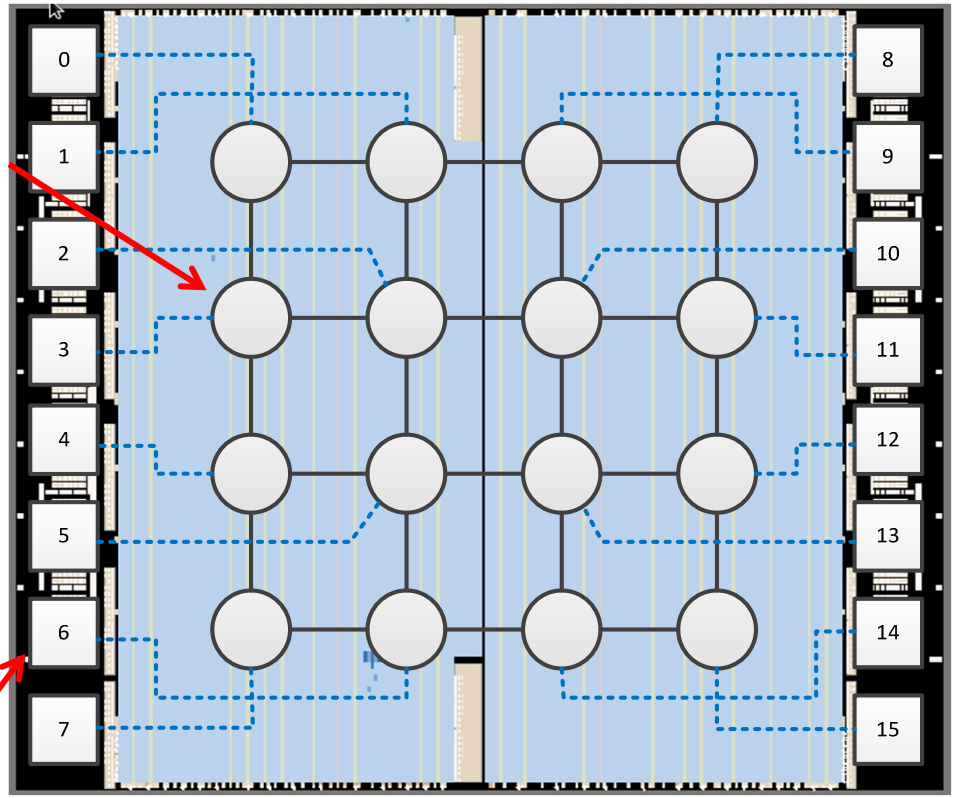
Power Comparison

Hard NoC saves power for even simplest systems



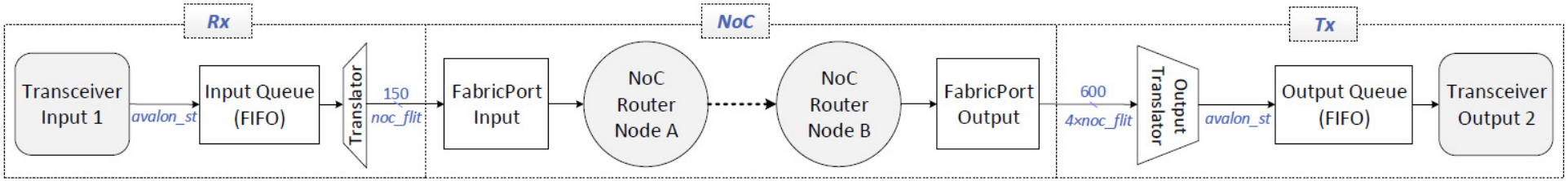
2. Ethernet Switch

- FPGAs with transceivers: commonly manipulating / switching packets
- e.g. 16x16 Ethernet switch, @ 10 Gb/s per channel



- NoC is the crossbar
- Plus buffering, distributed arbitration & back-pressure
- Fabric inspects packet headers, performs more buffering, …

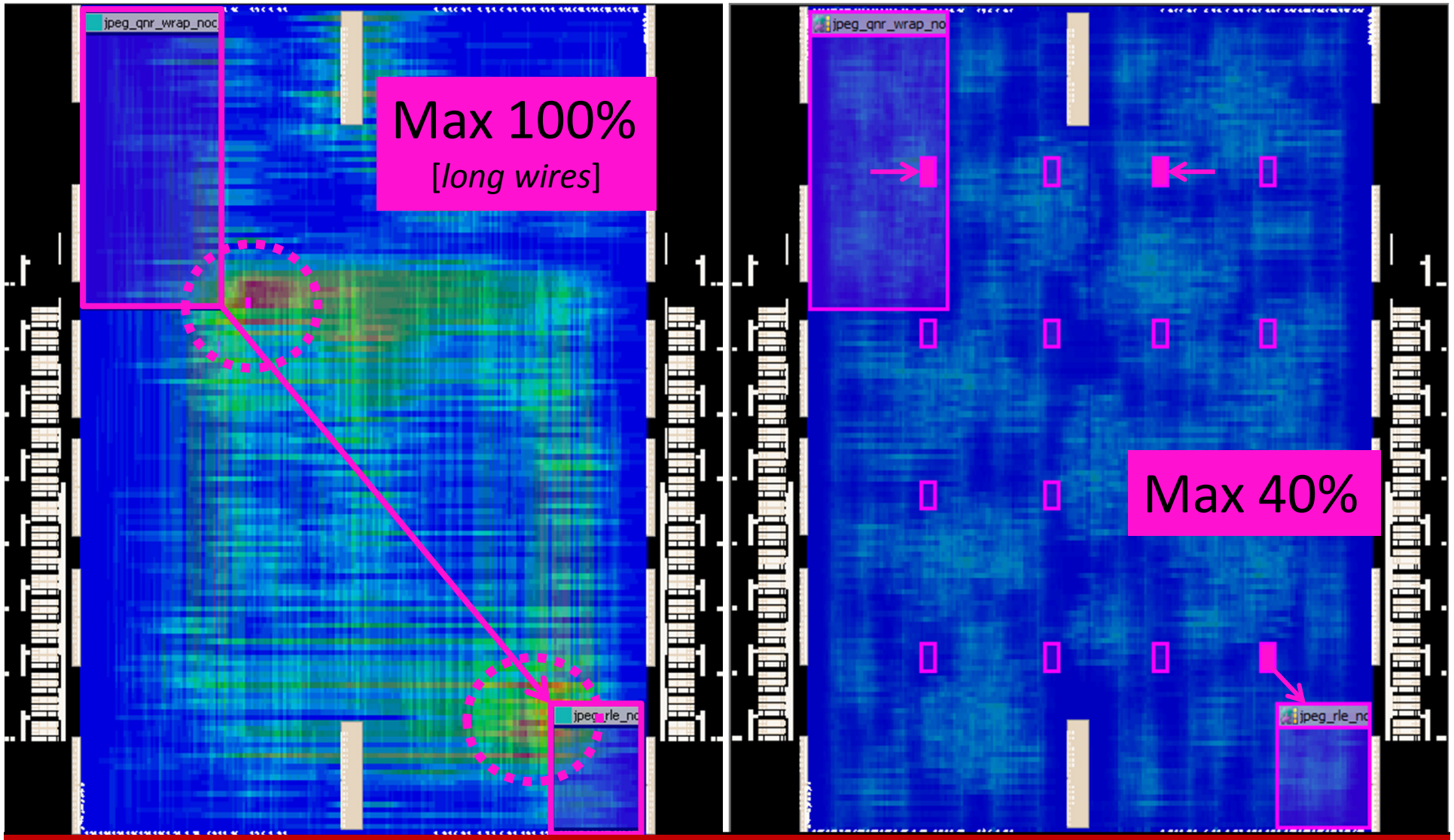
Ethernet Switch Efficiency



	NoC-Based Switch	Memory-Based Switch [Dai & Zhu]
Area Consumption (% of Stratix V device)	3%	8%
Supported Bandwidth	819 Gb/s	160 Gb/s
Latency (at 75% injection rate)	350 ns	2125 ns

- 14X more efficient!
- Latest FPGAs: ~2 Tb/s transceiver bandwidth → need good switches

3. Parallel JPEG (Latency Sensitive)



- NoC makes performance more predictable
- NoC doesn't produce wiring hotspots & saves long wires

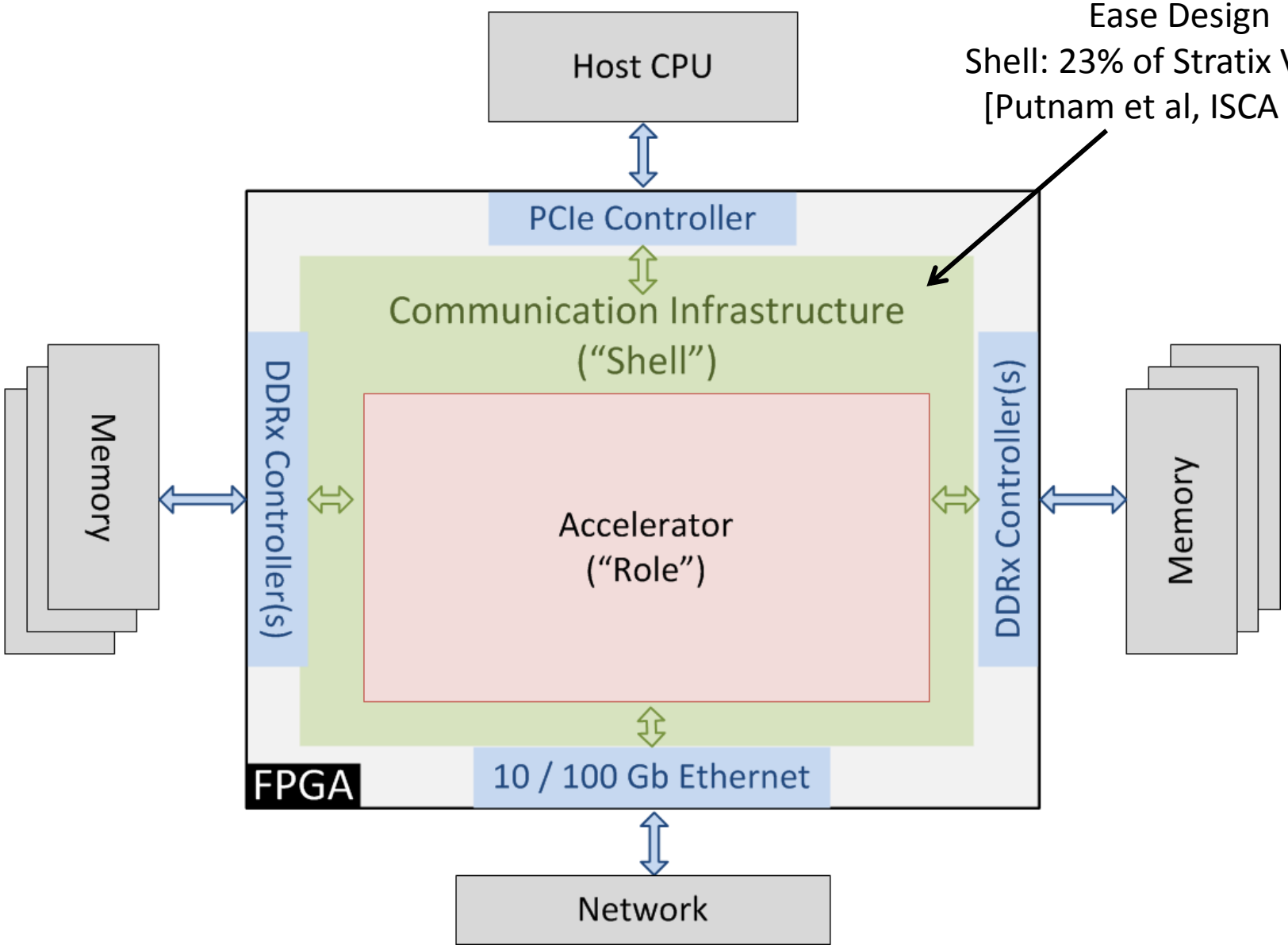
Future Trends and Embedded NoCs

Speculation Ahead!

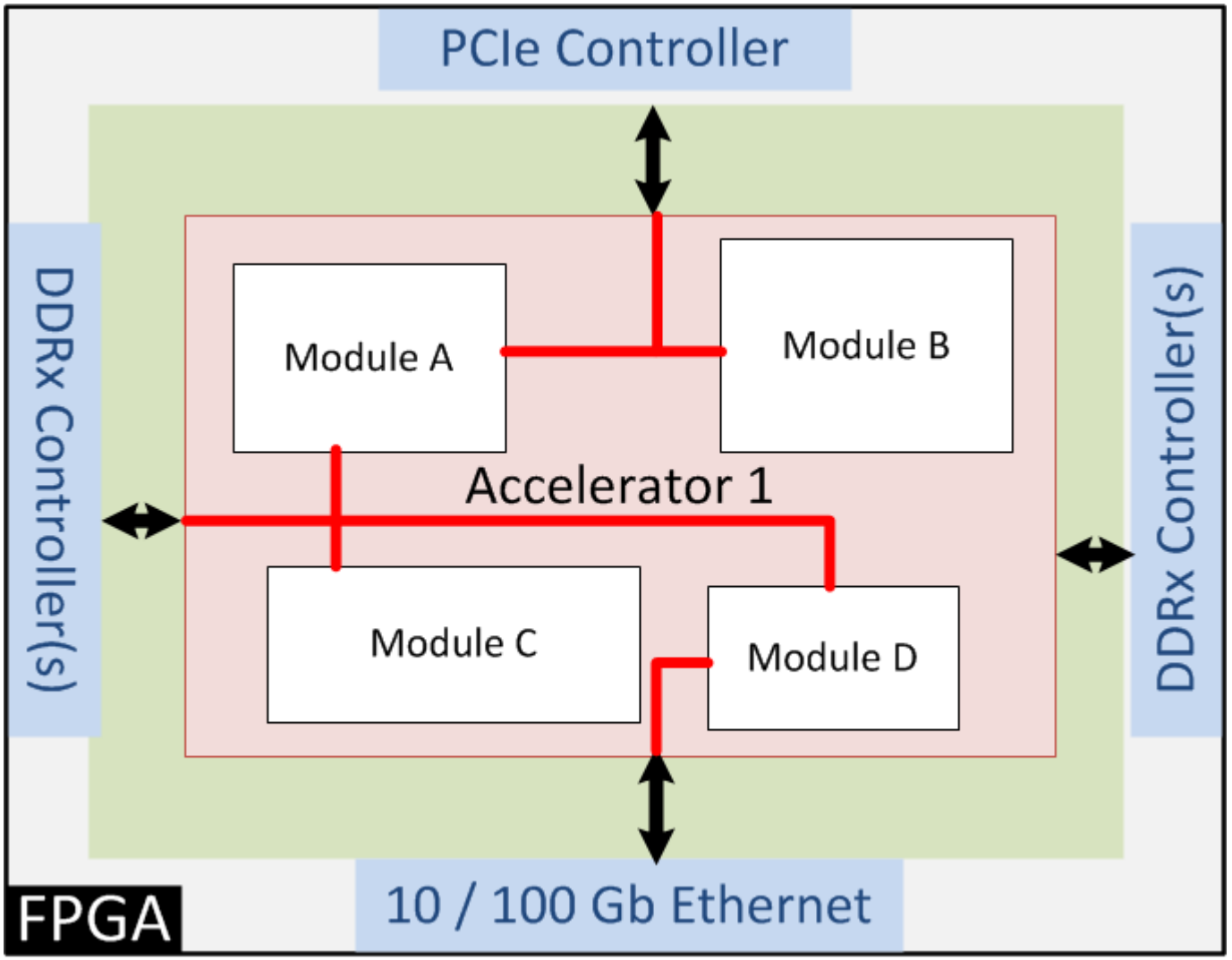
1. Embedded NoCs and the Datacenter

Datacenter Accelerators

Microsoft Catapult: Shell & Role to Ease Design
Shell: 23% of Stratix V FPGA
[Putnam et al, ISCA 2014]

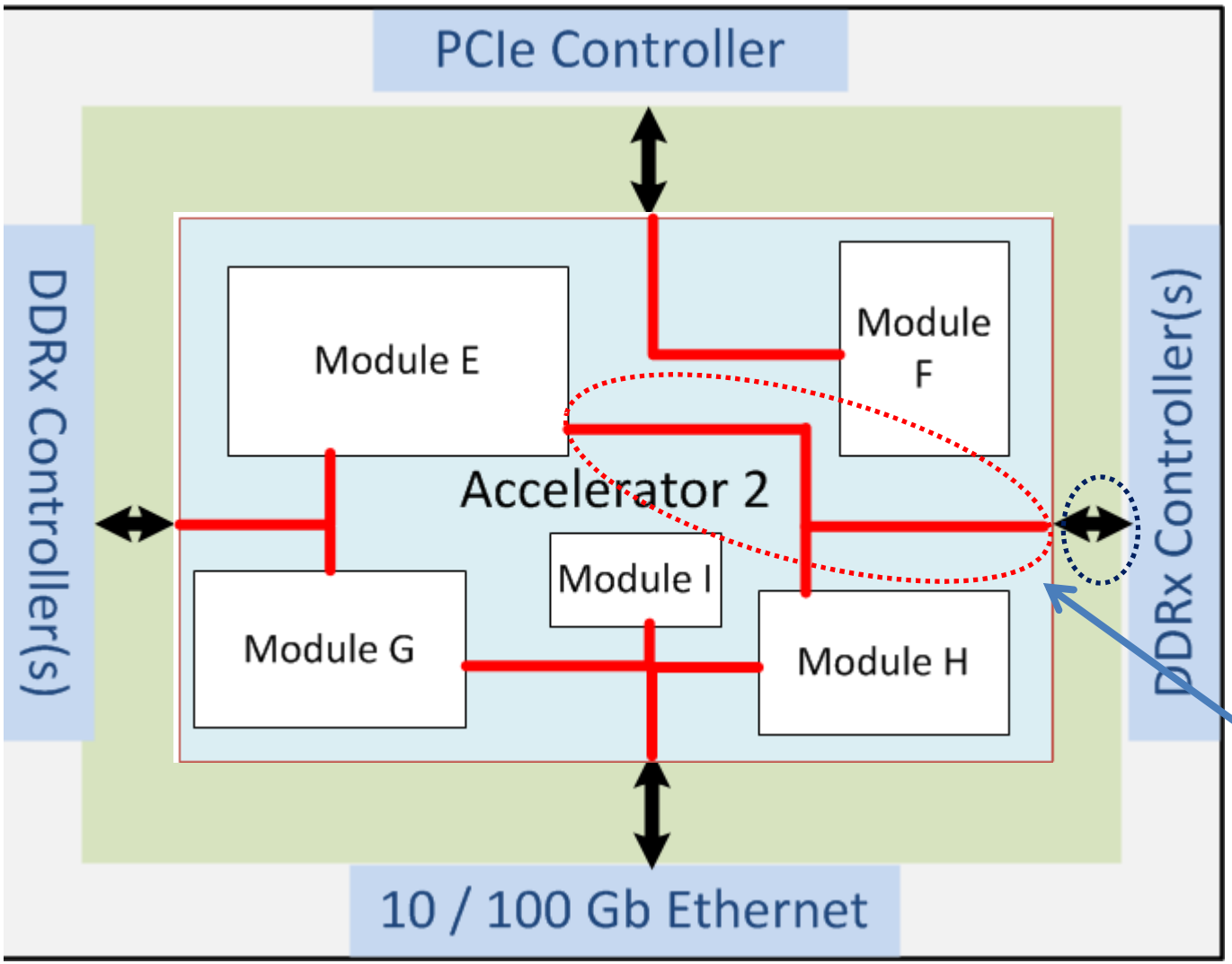


Datacenter “Shell”: Bus Overhead



- Buses to I/Os in shell & role
- Divided into two parts to ease compilation (shell portion locked down)

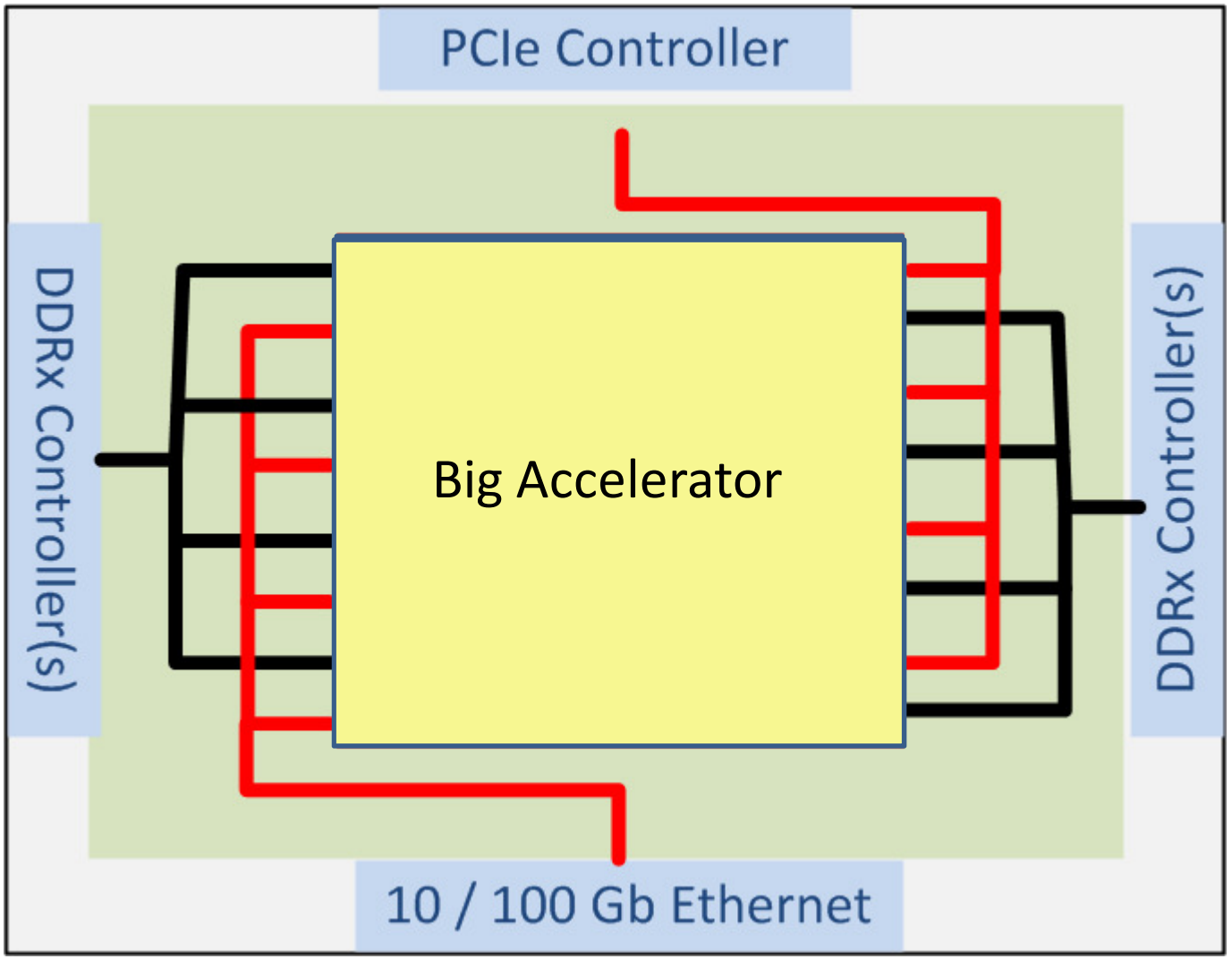
Datacenter “Shell”: Swapping Accelerators



- Partial reconfig of role only → swap accelerator w/o taking down system
- Overengineer shell buses for most demanding accelerator

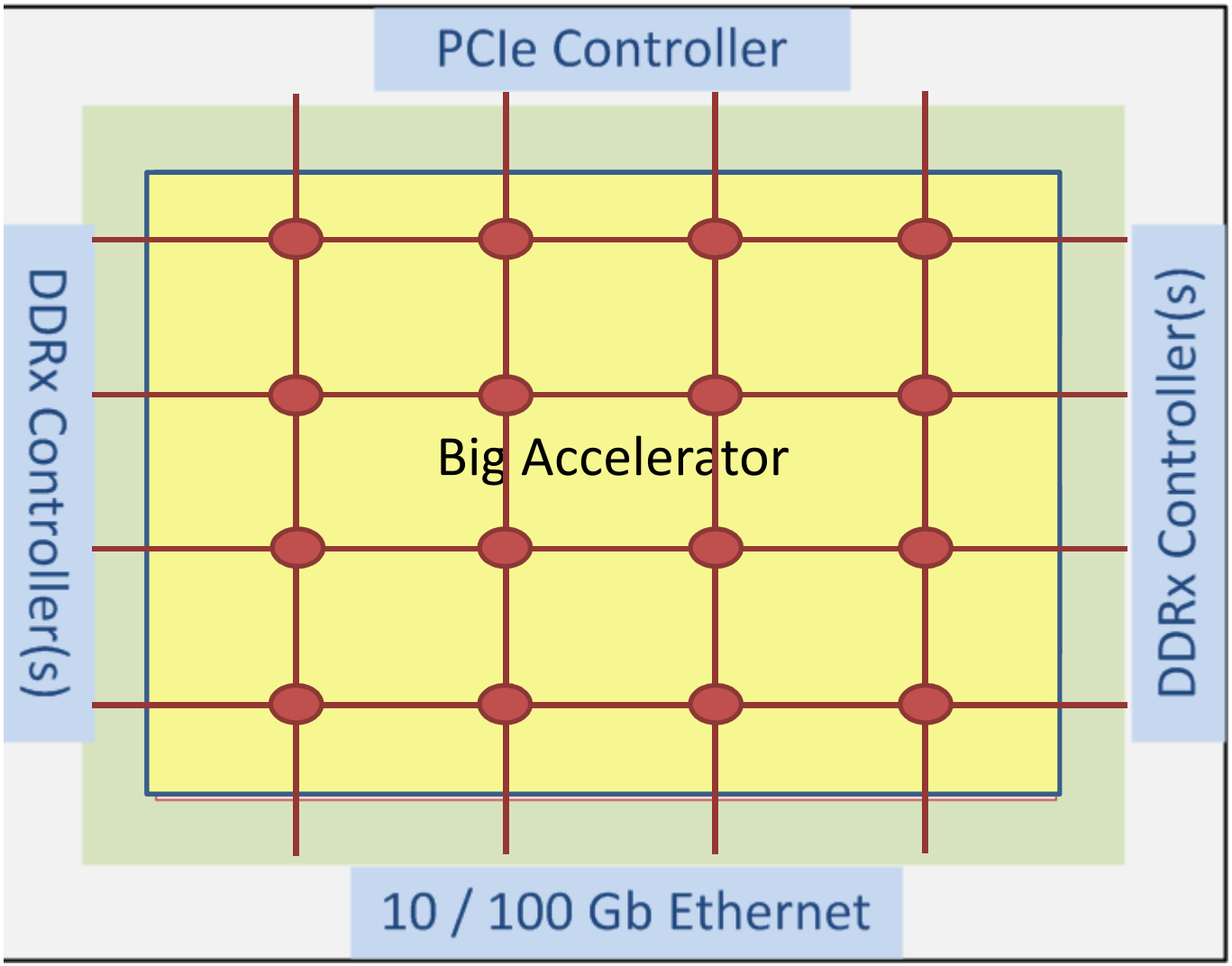
Two separate compiles → lose some optimization of bus

More Swappable Accelerators



- Allows more virtualization
- But shell complexity increases
- Less efficient
- Wasteful for one big accelerator

Shell with an Embedded NoC



- Efficient for more cases (small or big accelerators)
- Data brought into accelerator, not just to edge with locked bus

2. Interposer-Based FPGAs

Xilinx: Larger Fabric with Interposers

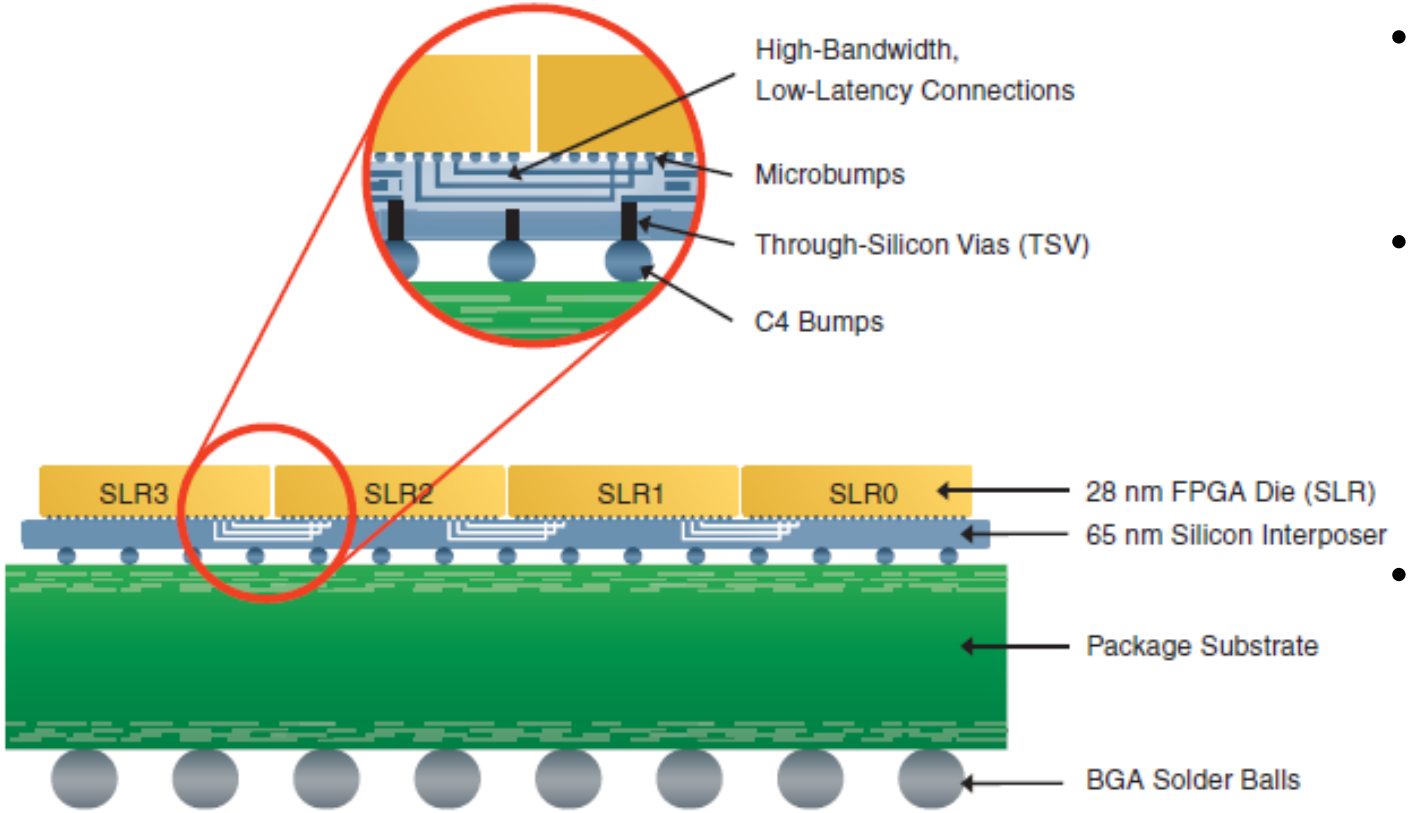


Figure: Xilinx, SSI Technology White Paper, 2012

- Create a larger FPGA with interposers
- 10,000 connections between dice (23% of normal routing)
- Routability good if > 20% of normal wiring cross interposer [Nasiri et al, TVLSI, to appear]

Interposer Scaling

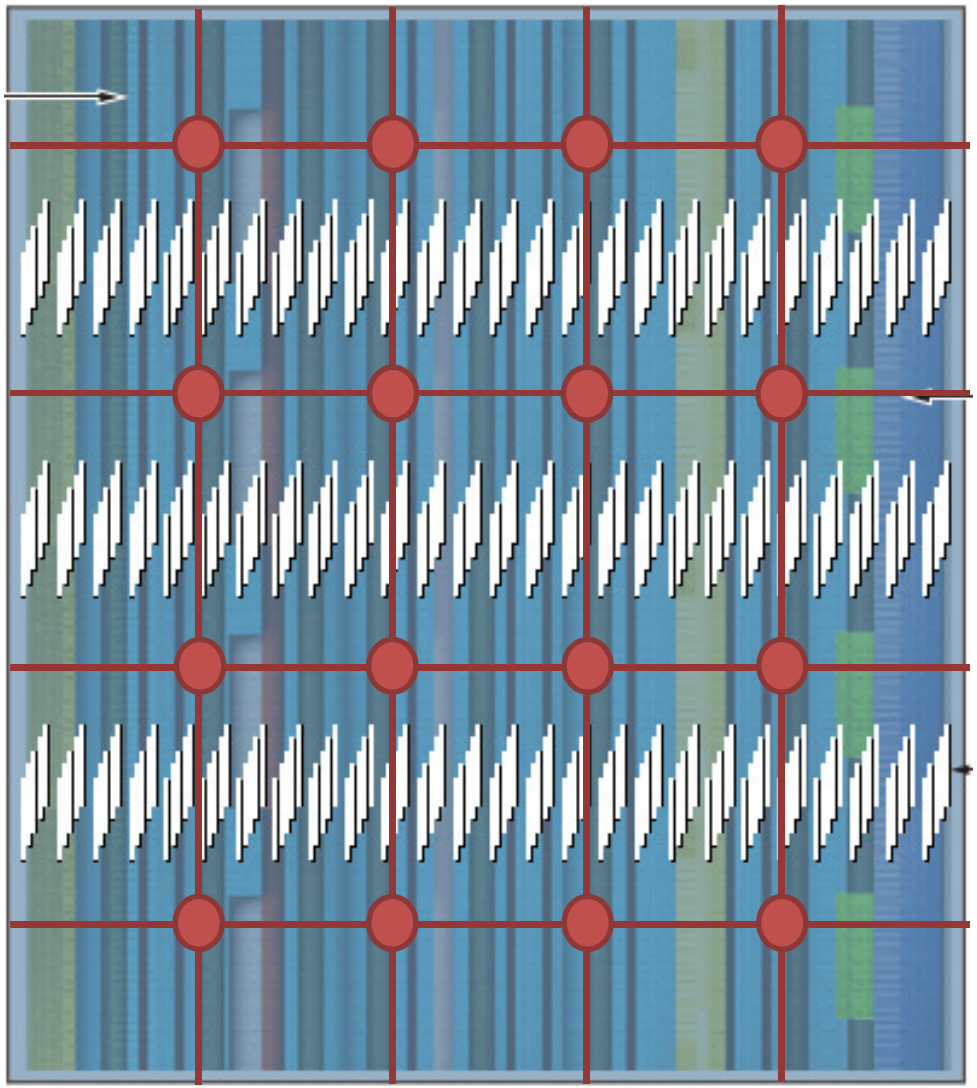


Figure: Xilinx, SSI Technology White Paper, 2012

- Concerns about how well microbumps will scale
- Will interposer routing bandwidth remain >20% of within-die bandwidth?
- Embedded NoC: naturally multiplies routing bandwidth (higher clock rate on NoC wires crossing interposer)

Altera: Heterogeneous Interposers

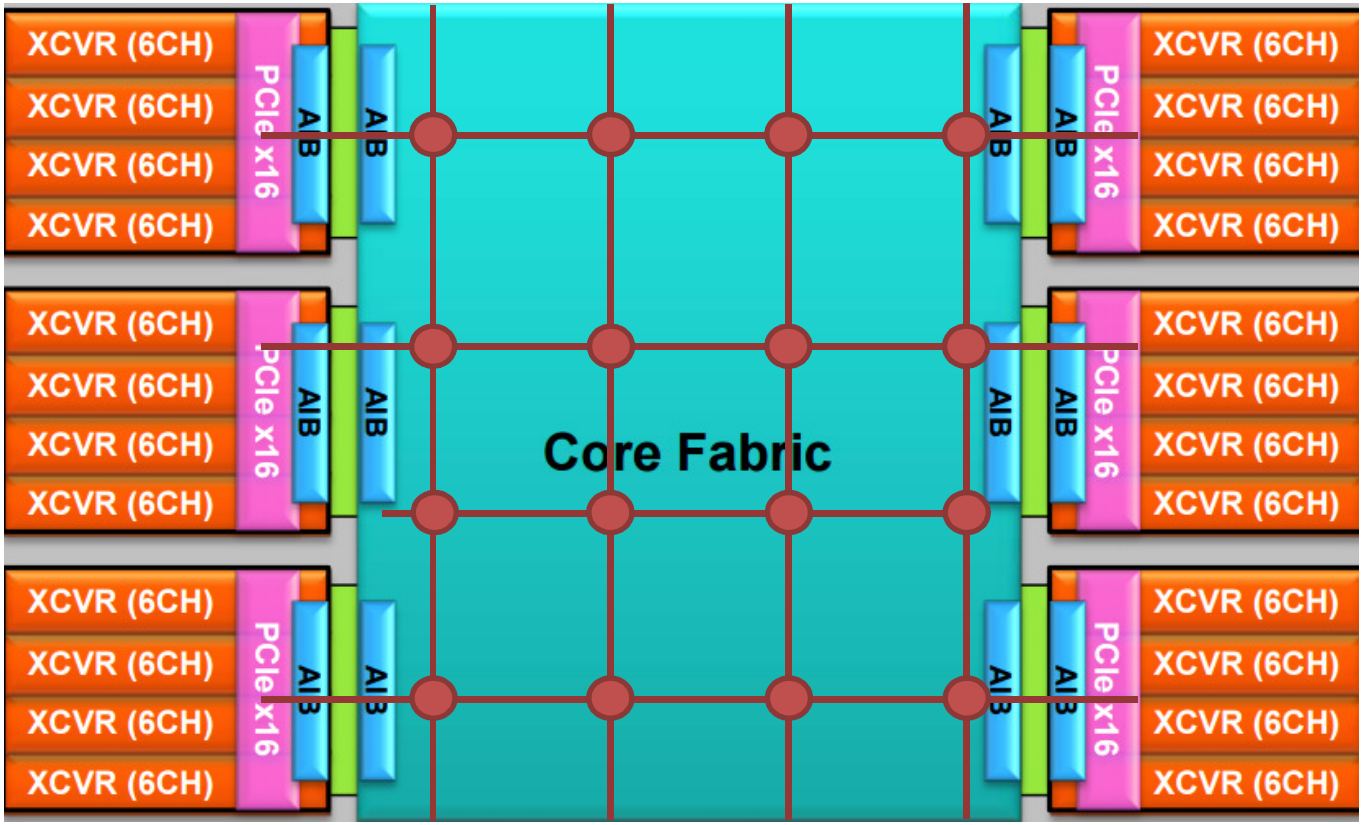


Figure: Mike Hutton, Altera Stratix 10, FPL 2015

- Custom wiring interface to each unique die
  - PCIe/transceiver, high-bandwidth memory
- NoC: standardize interface, allow TDM-ing of wires
- Extends system level interconnect beyond one die

3. Registered Routing

Registered Routing

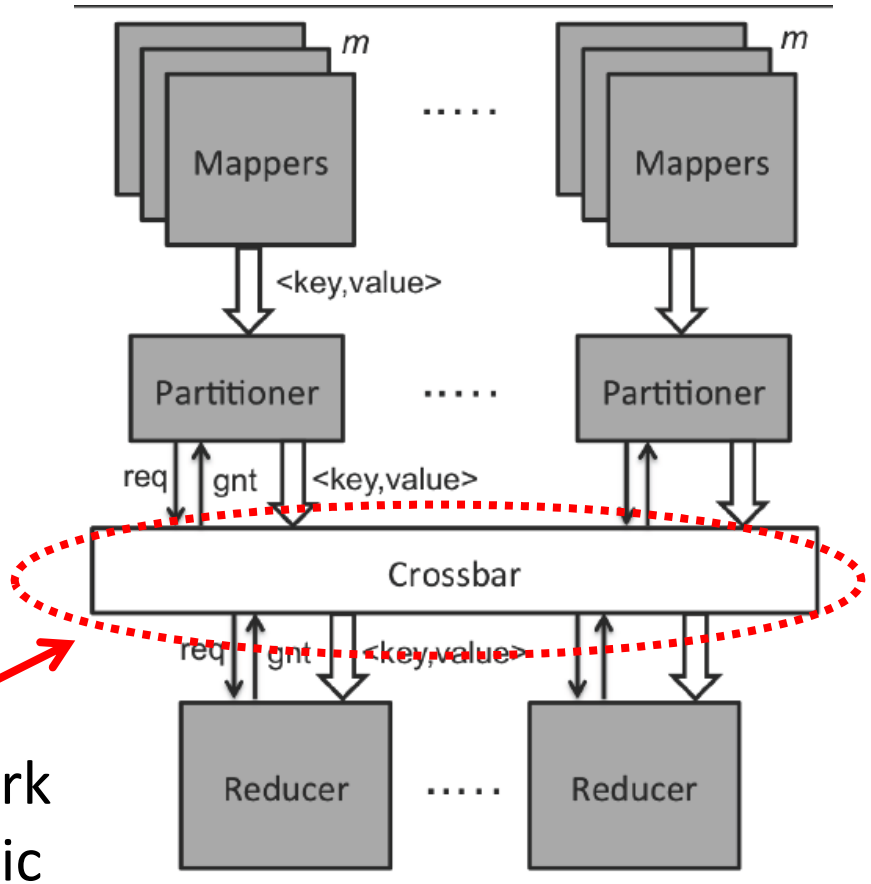
- Stratix 10 includes a pulse latch in each routing driver
  - Enables deeper interconnect pipelining
  - Obviates need for a new system-level interconnect?
- I don't think so
  - Makes it easier to run wires faster
  - But still not:
    - Switching, buffering, arbitration (complete interconnect)
    - Pre-timing closed
    - Abstraction to compose & re-configure systems
- Pushes more designers to latency-tolerant techniques
  - Which helps match the main NoC programming model

4. Kernels → Massively Parallel Accelerators

Crossbars for Design Composition

Map – Reduce and FPGAs

- [Ghasemi & Chow, MASc thesis, 2015]
- Write map & reduce kernel
- Use Spark infrastructure to distribute data & kernels across many CPUs
- Do same for FPGAs?



Between chips → network
Within a chip → soft logic
Consumes lots of soft logic and limits routable design to ~30% utilization!

Can We Remove the Crossbar?

- Not without breaking Map-Reduce/Spark abstraction!
  - The automatic partitioning / routing / merging of data is what makes Spark easy to program
  - Need a crossbar to match the abstraction and make composability easy
- NoC: efficient, distributed crossbar
  - Allows us to efficiently compose kernels
  - Can use crossbar abstraction within chips (NoC) and between chips (datacenter network)

Wrap Up

Wrap Up

- Adding NoCs to FPGAs
  - Enhances efficiency of system level interconnect
  - Enables new abstractions (crossbar composability, easily-swappable accelerators)
- NoC abstraction can cross interposer boundaries
  - Interesting multi-die systems
- My belief:
  - Special purpose box → datacenter
  - ASIC-like flow → composable flow
  - Embedded NoCs help make this happen

Future Work

- CAD System for Embedded NoCs
  - Automatically create lightweight soft logic to connect to fabric port (translator)
    - According to designer's specified intent
  - Choose best router to connect each compute module
  - Choose when to use NoC vs. soft links
- Then map more applications, using CAD