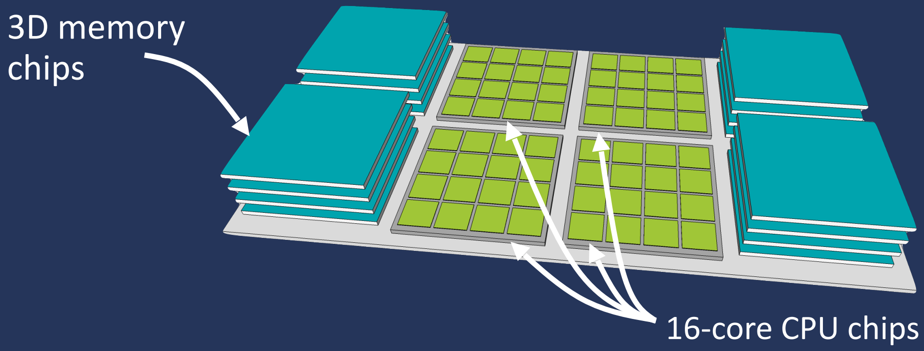
UNIVERSITY OF TORONTO | Engineering

Architecting Chiplet-Based Systems

Natalie Enright Jerger
Canada Research Chair in Computer Architecture
enright@ece.utoronto.ca
www.eecg.toronto.edu/~enright

UNIVERSITY OF TORONTO | Engineering

Architecting Chiplet-Based Systems



Natalie Enright Jerger
Canada Research Chair in Computer Architecture
enright@ece.utoronto.ca
www.eecg.toronto.edu/~enright

Executive Summary

Why build chiplet-based systems?

How to build chiplet-based systems?

Where do we go from here?

Executive Summary

Why build chiplet-based systems?

- End of technology scaling
- Rise of heterogeneity
- Demands of big data

How to build chiplet-based systems?

Where do we go from here?

Executive Summary

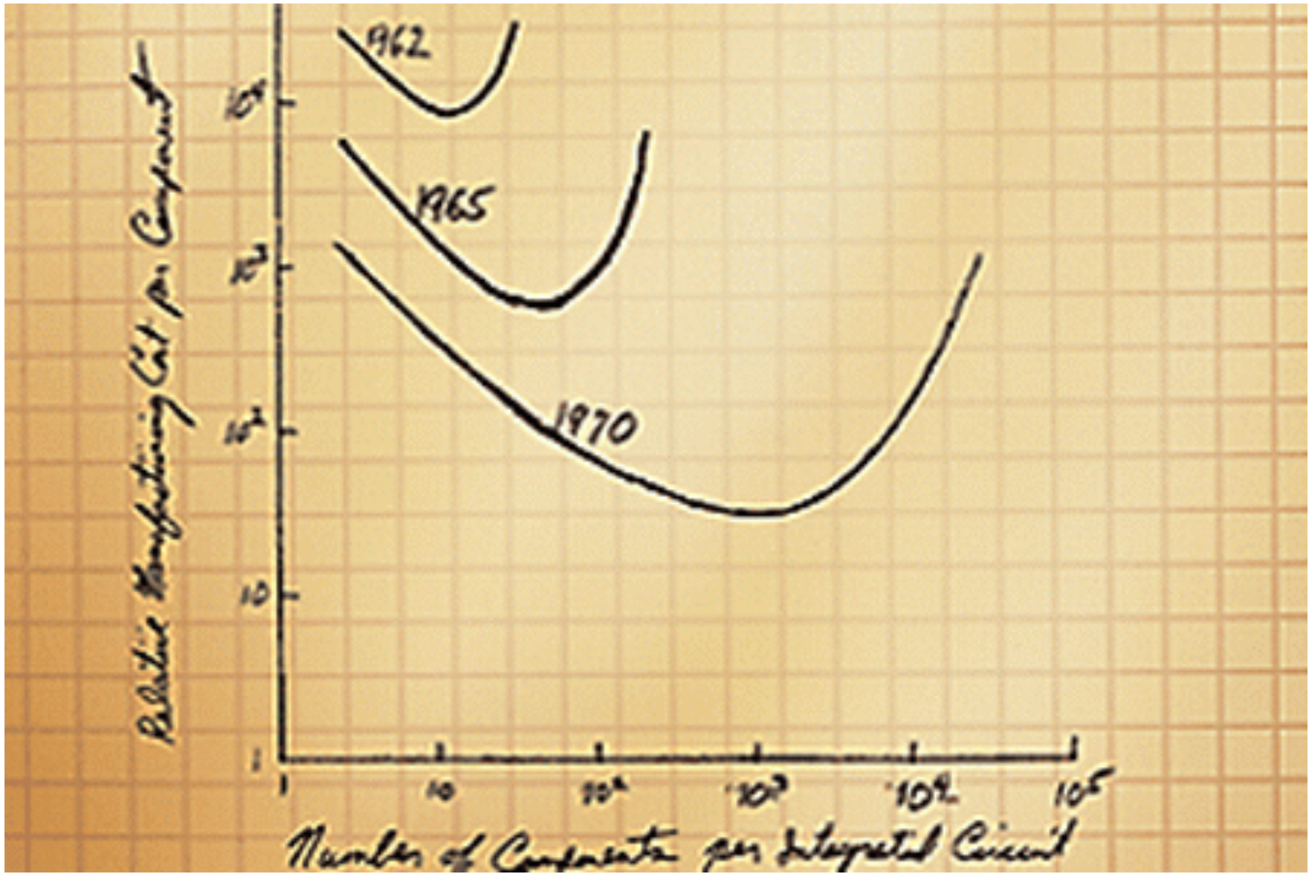
Why build chiplet-based systems?

How to build chiplet-based systems?

- Reintegrate with hybrid topologies
- Deadlock-free routing for independent, modular design

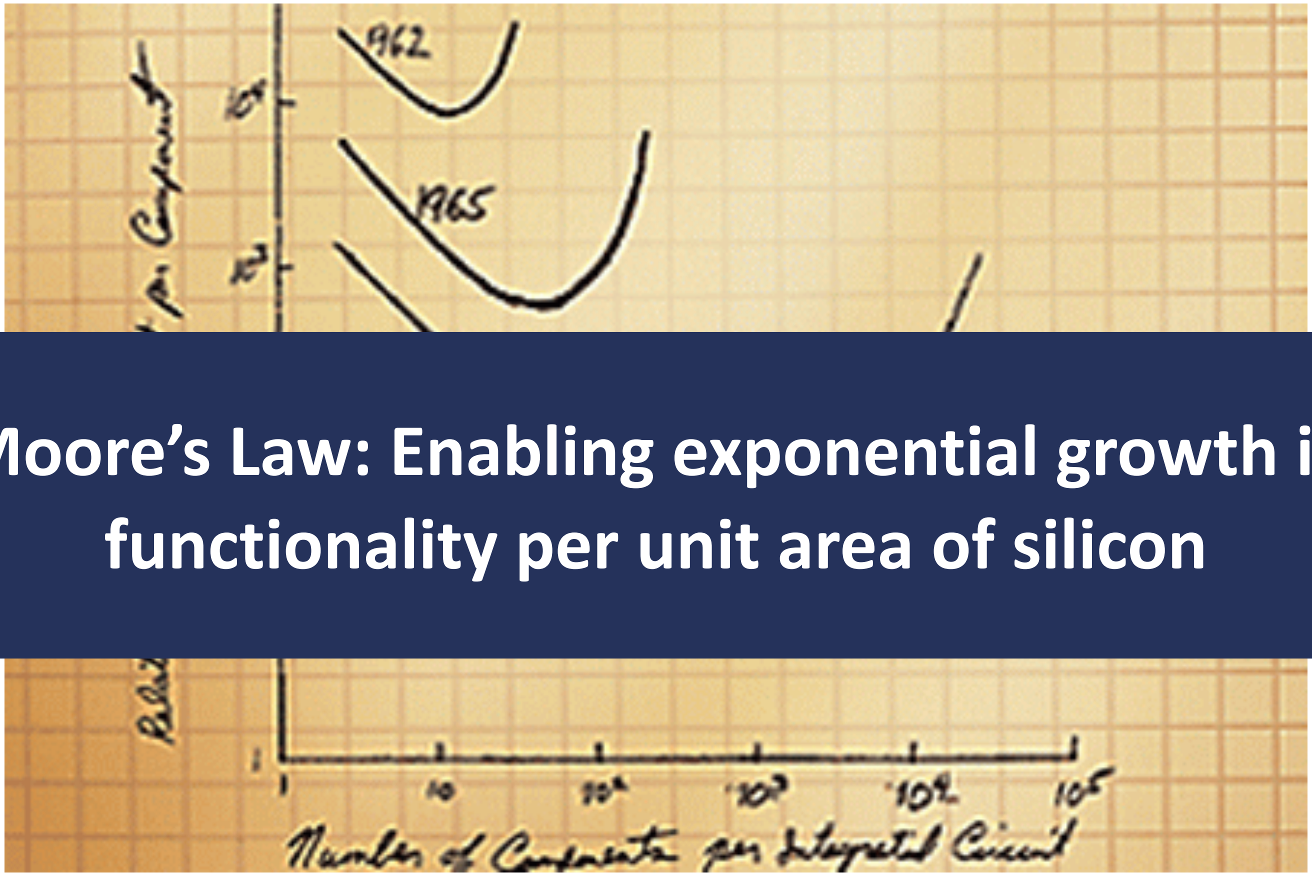
Where do we go from here?

Challenges: End of Scaling



Source: G.E. Moore, Electronics 1965

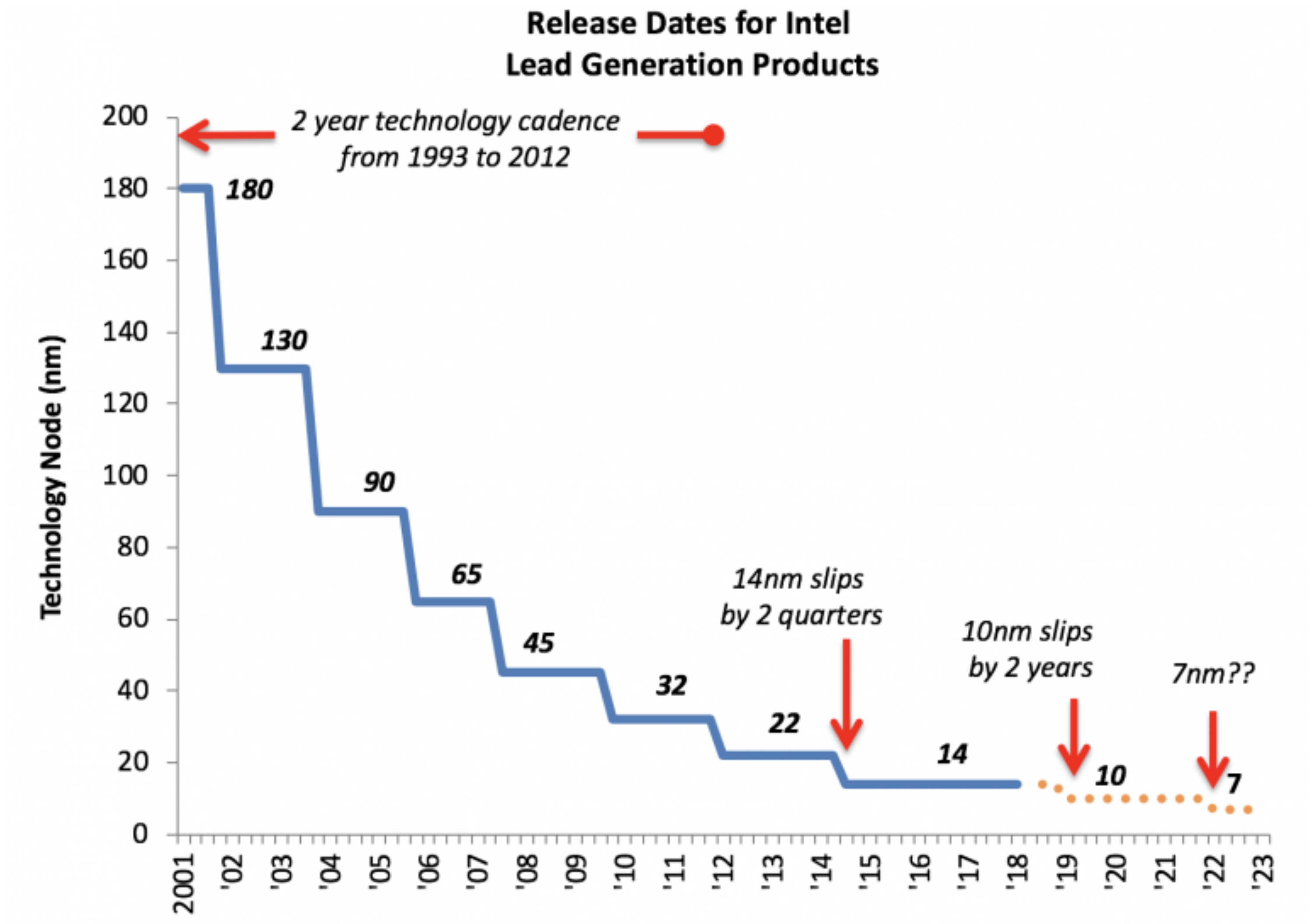
Challenges: End of Scaling



Moore's Law: Enabling exponential growth in functionality per unit area of silicon

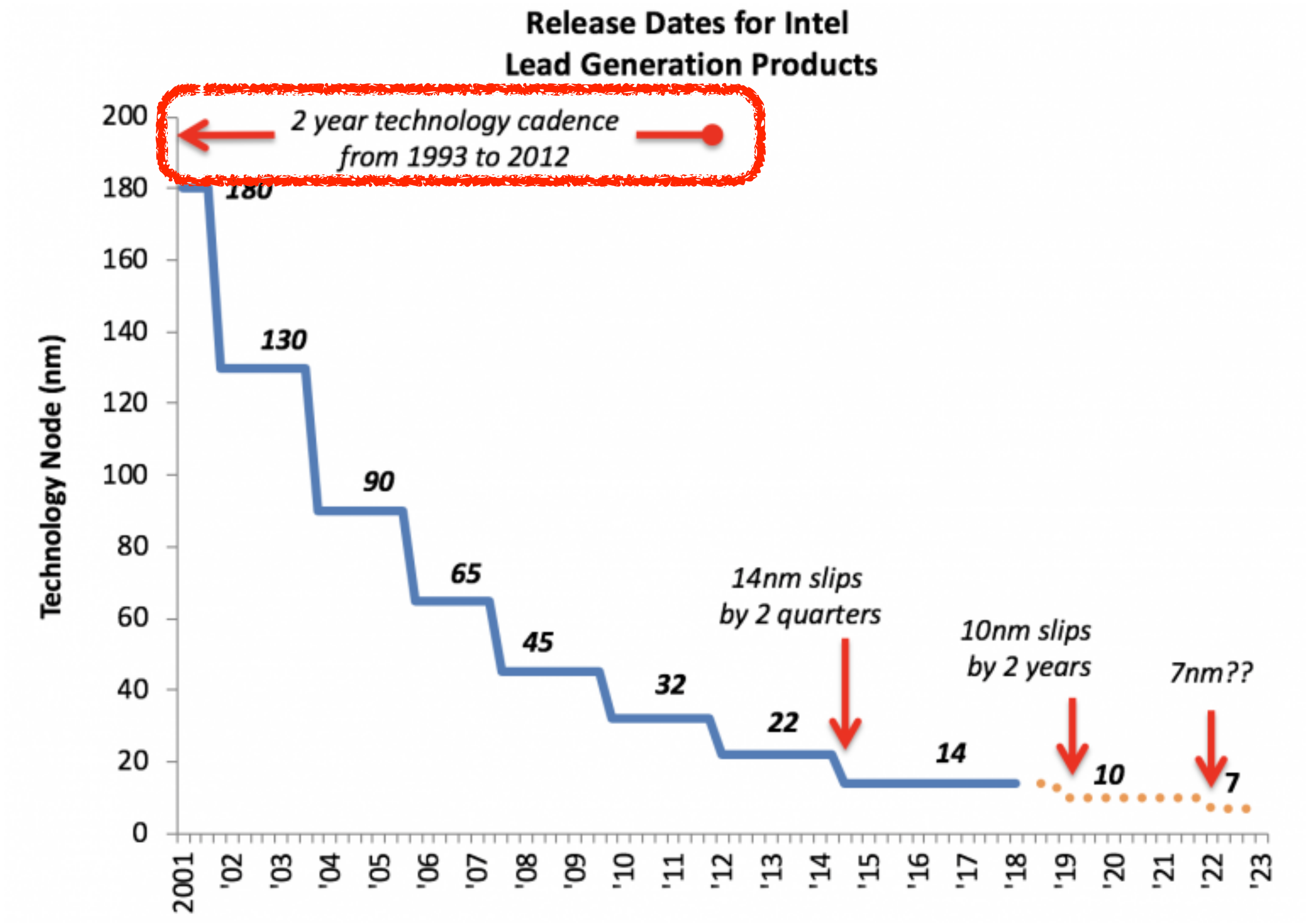
Source: G.E. Moore, Electronics 1965

Challenges: End of Scaling



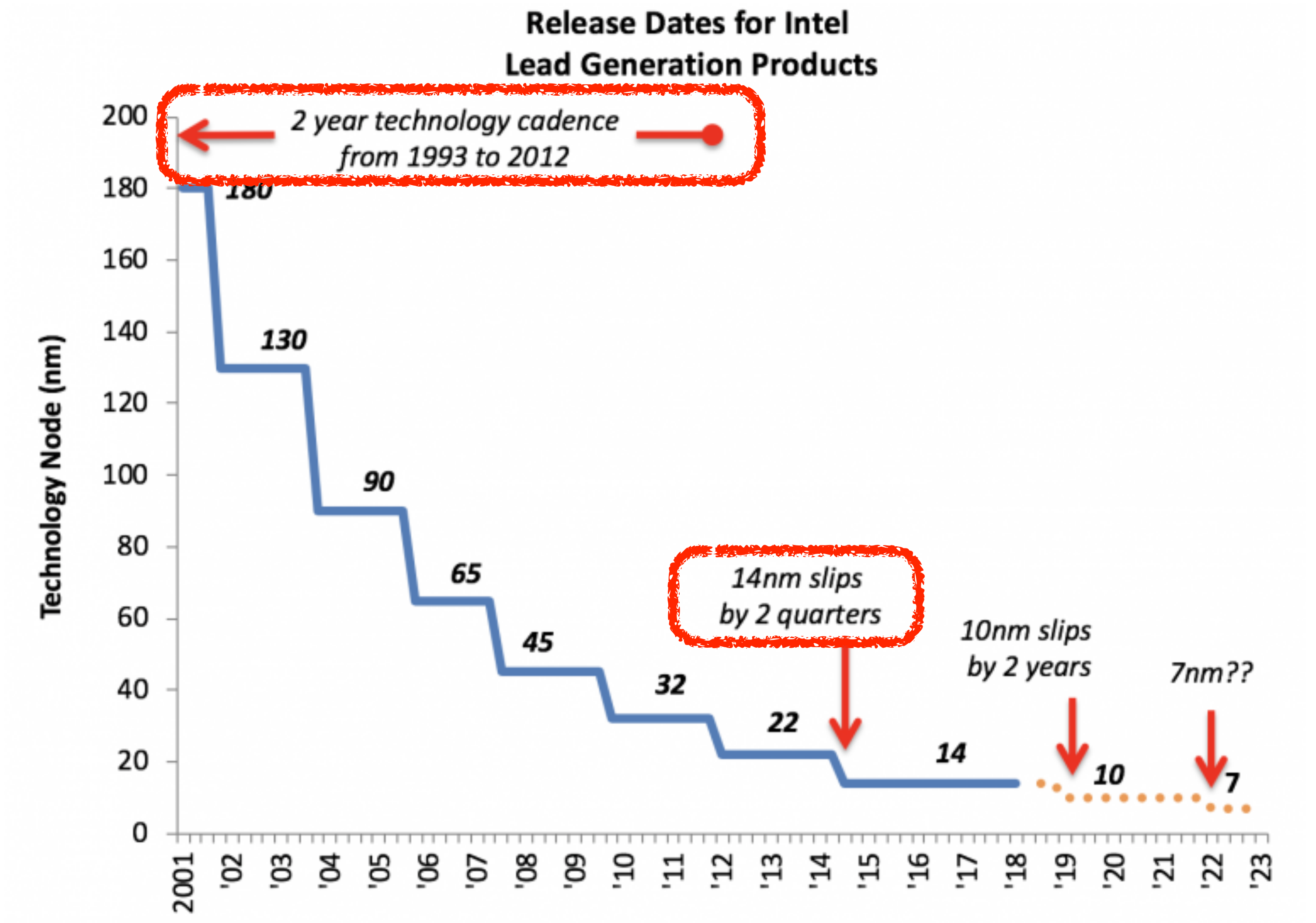
Source: David Brooks

Challenges: End of Scaling



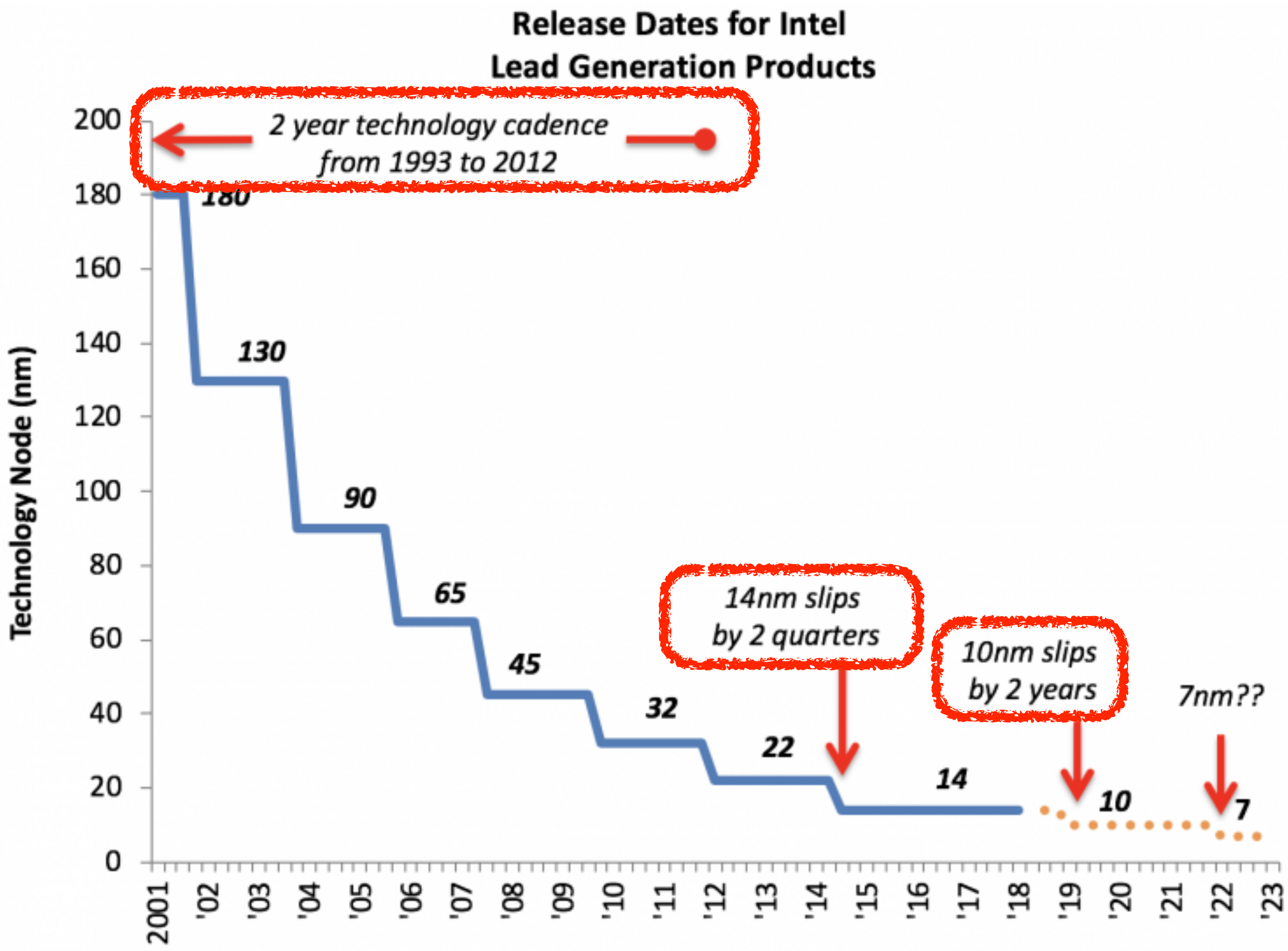
Source: David Brooks

Challenges: End of Scaling



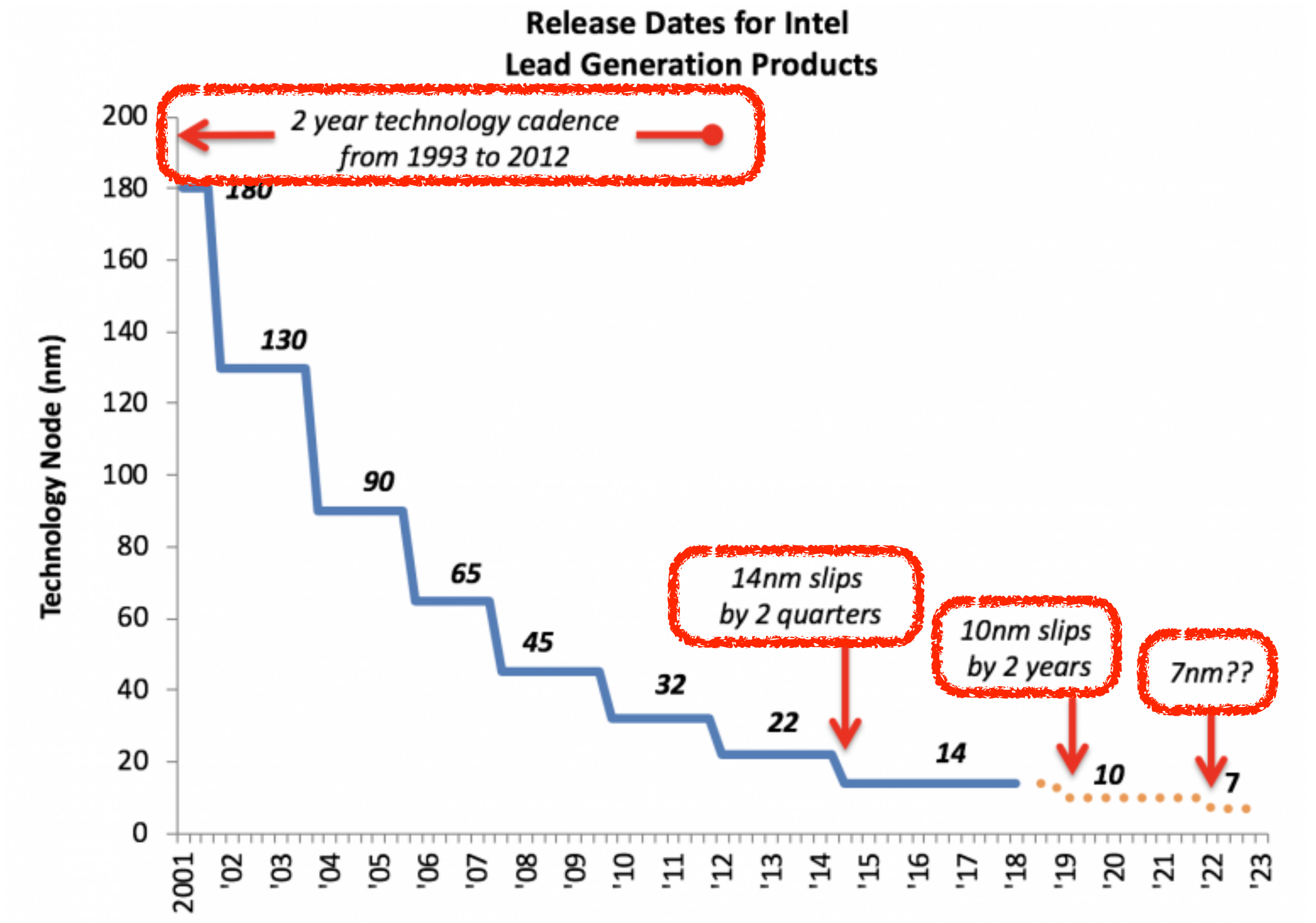
Source: David Brooks

Challenges: End of Scaling



Source: David Brooks

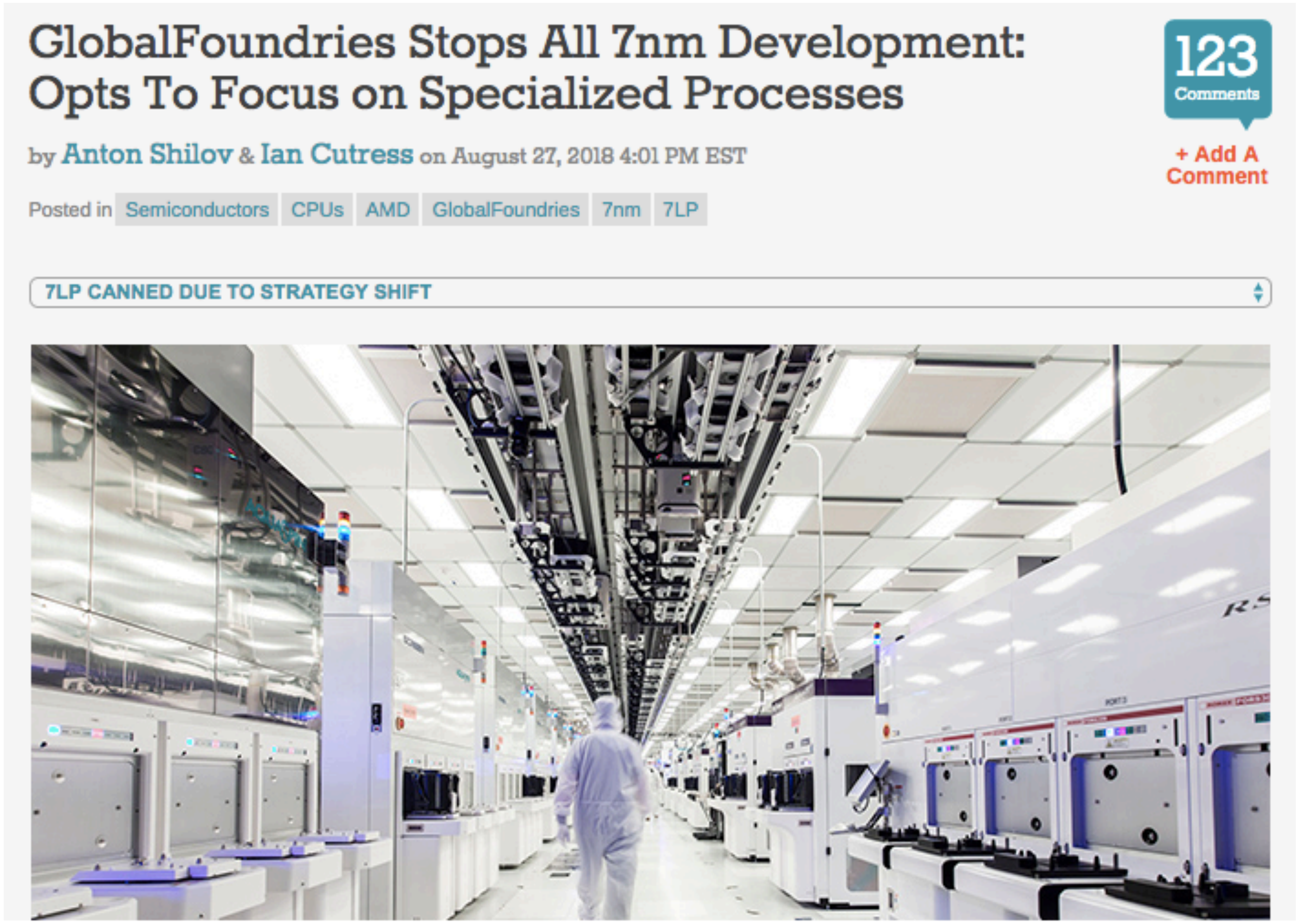
Challenges: End of Scaling



Source: David Brooks

Challenges: End of Scaling

Challenges: End of Scaling

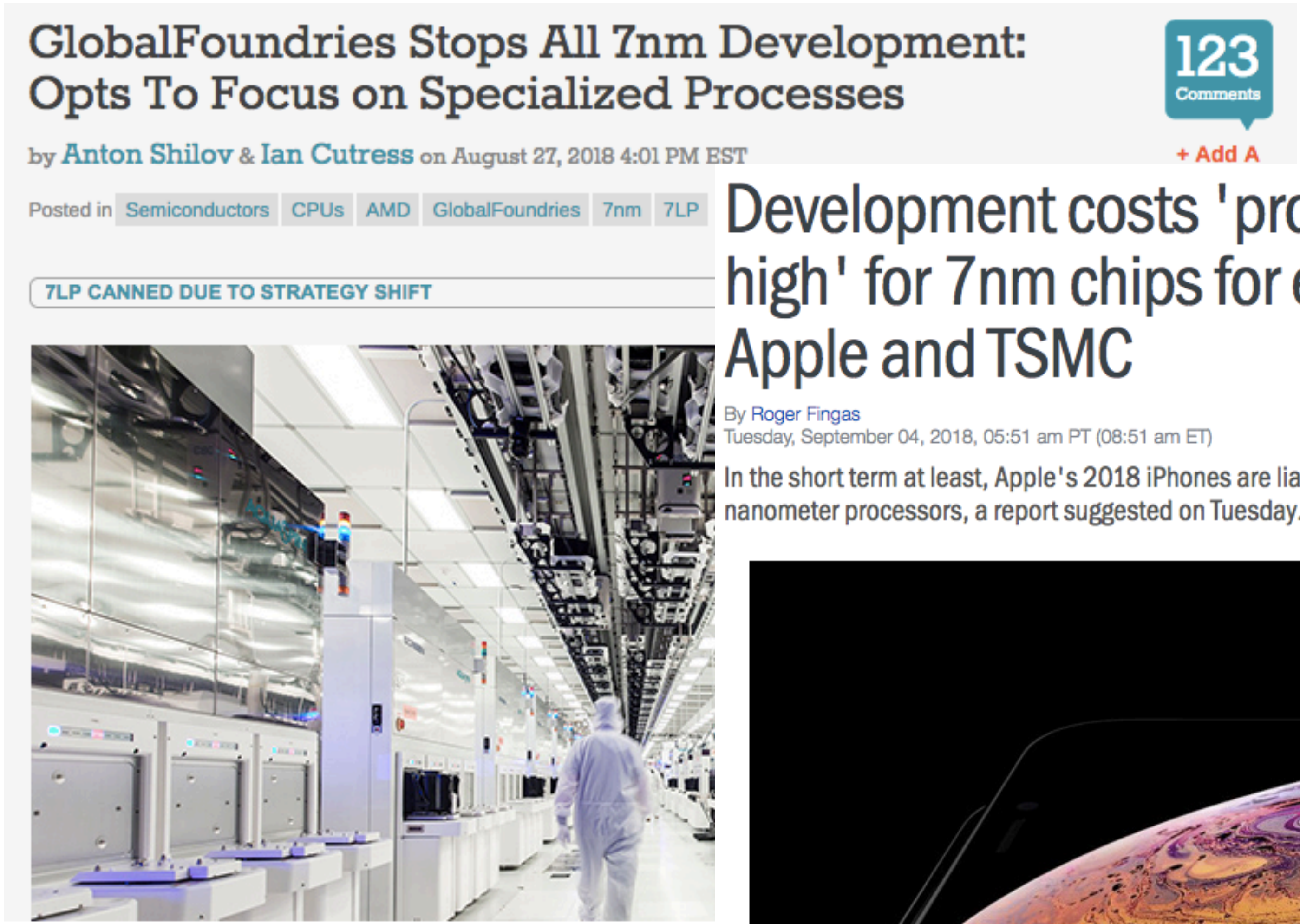


GlobalFoundries Stops All 7nm Development: Opts To Focus on Specialized Processes

by Anton Shilov & Ian Cutress on August 27, 2018 4:01 PM EST

Posted in Semiconductors CPUs AMD GlobalFoundries 7nm 7LP

Challenges: End of Scaling



GlobalFoundries Stops All 7nm Development: Opts To Focus on Specialized Processes

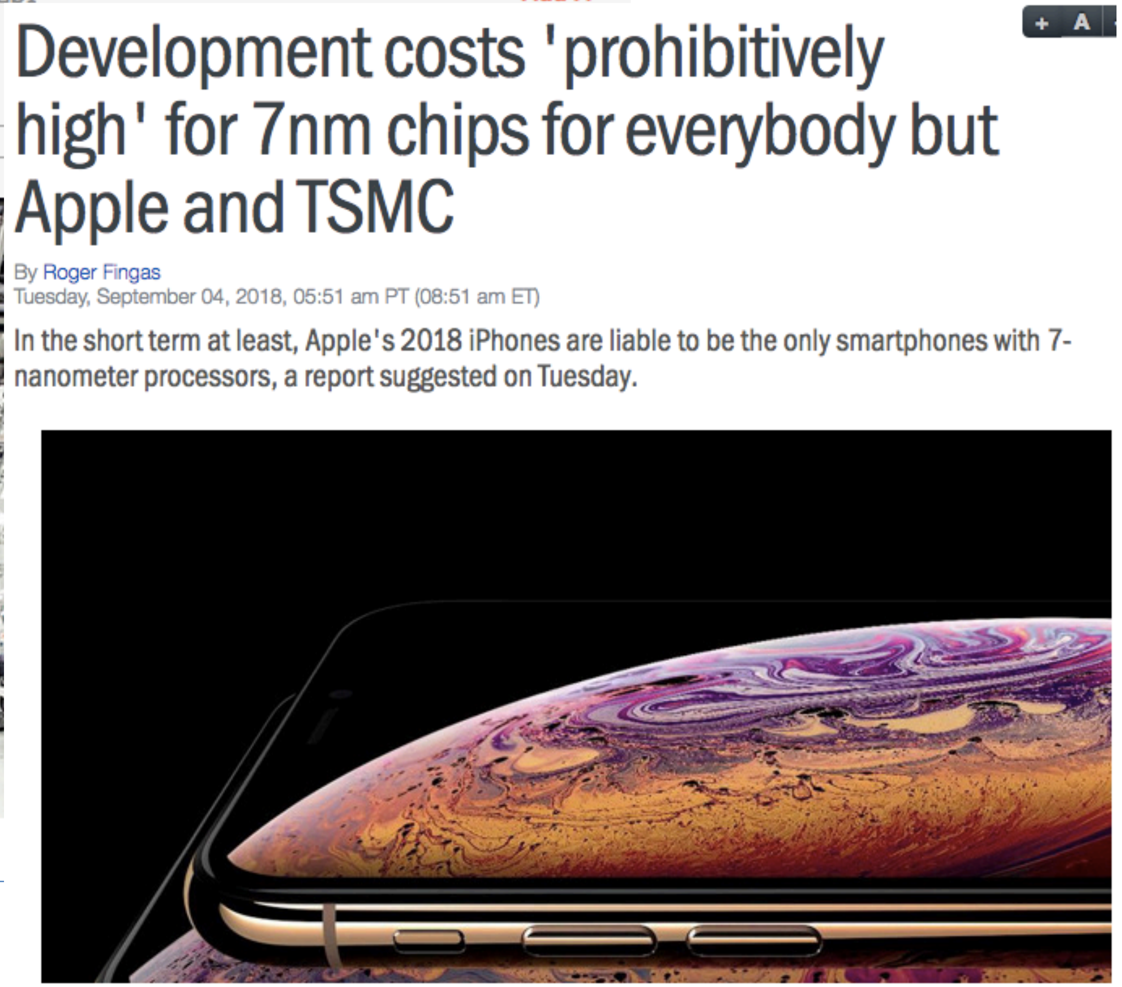
123 Comments

by Anton Shilov & Ian Cutress on August 27, 2018 4:01 PM EST

+ Add A

Posted in Semiconductors CPUs AMD GlobalFoundries 7nm 7LP

7LP CANNED DUE TO STRATEGY SHIFT

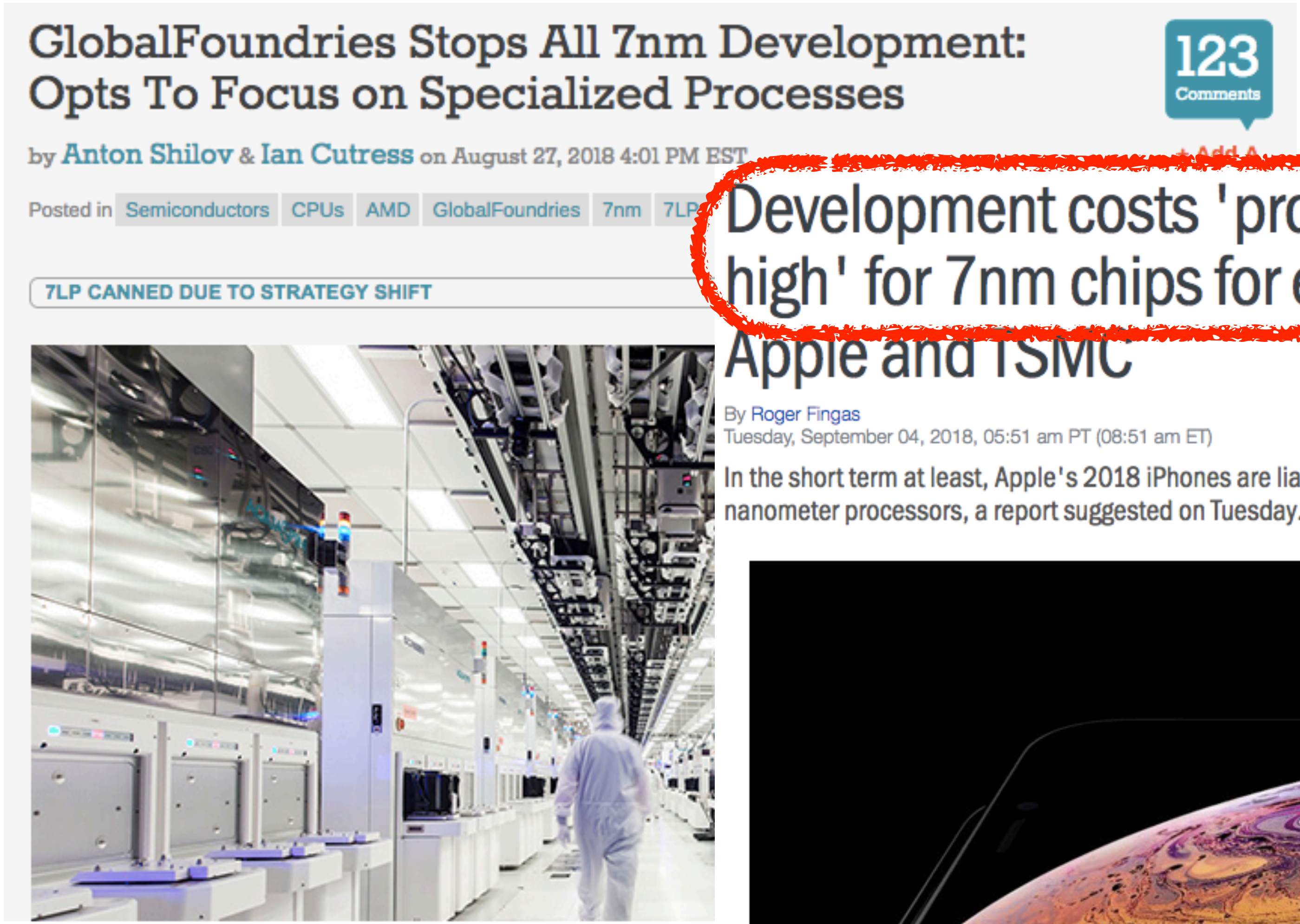


Development costs 'prohibitively high' for 7nm chips for everybody but Apple and TSMC

By Roger Fingas
Tuesday, September 04, 2018, 05:51 am PT (08:51 am ET)

In the short term at least, Apple's 2018 iPhones are liable to be the only smartphones with 7-nanometer processors, a report suggested on Tuesday.

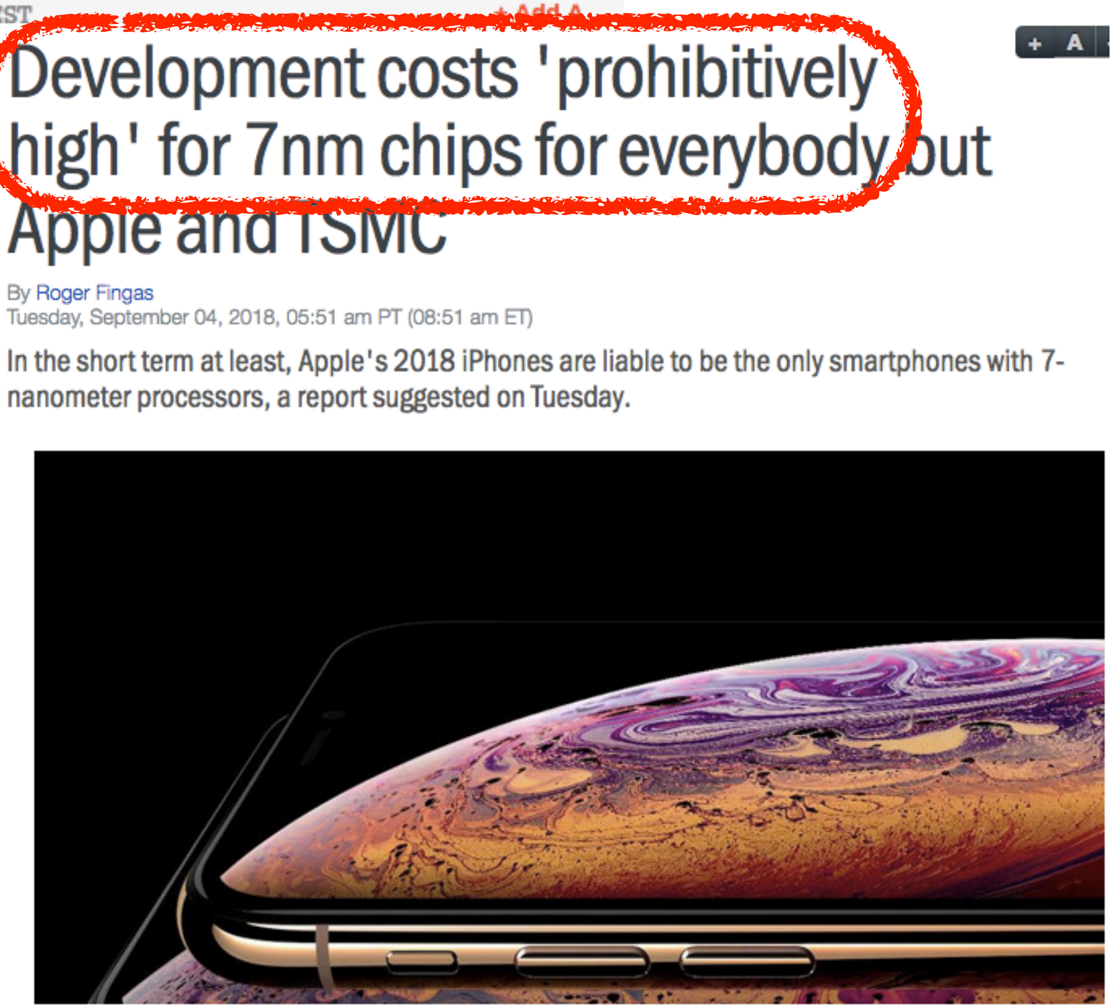
Challenges: End of Scaling



GlobalFoundries Stops All 7nm Development: Opts To Focus on Specialized Processes

by Anton Shilov & Ian Cutress on August 27, 2018 4:01 PM EST

7LP CANNED DUE TO STRATEGY SHIFT



Development costs 'prohibitively high' for 7nm chips for everybody but Apple and TSMC

By Roger Fingas
Tuesday, September 04, 2018, 05:51 am PT (08:51 am ET)

In the short term at least, Apple's 2018 iPhones are liable to be the only smartphones with 7-nanometer processors, a report suggested on Tuesday.

Challenges: End of Scaling

GlobalFoundries Stops All 7nm Development: Opts To Focus on Specialized Processes

123 Comments

by Anton Shilov & Ian Cutress on August 27, 2018 4:01 PM EST

Posted in Semiconductors CPUs AMD GlobalFoundries 7nm 7LP

7LP CANNED DUE TO STRATEGY SHIFT

Development costs 'prohibitively high' for 7nm chips for everybody but

Moving forward, limited ability to integrate more transistors on a chip

Need: Novel approaches to increase integration *affordably*

Challenges: The Rise of Heterogeneity

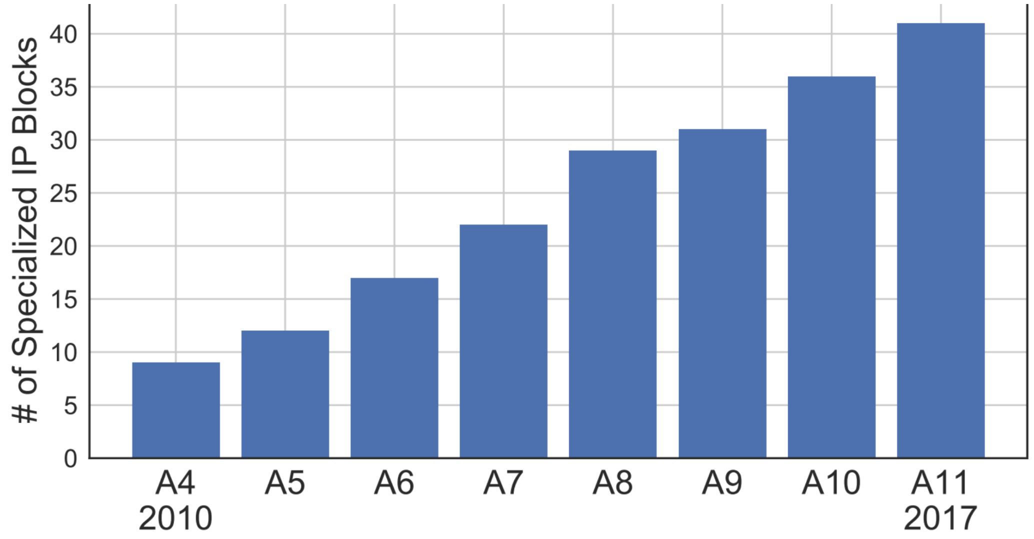
End of Dennard scaling

- Need power efficient alternatives to general purpose computing

Challenges: The Rise of Heterogeneity

End of Dennard scaling

Need power efficient alternatives to general purpose computing

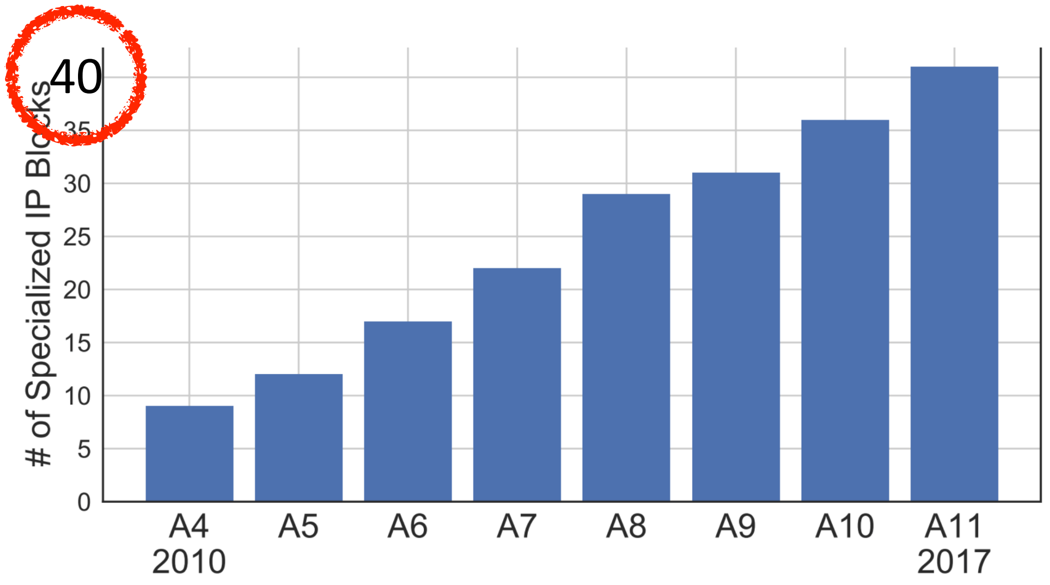


Source: David Brooks

Challenges: The Rise of Heterogeneity

End of Dennard scaling

Need power efficient alternatives to general purpose computing

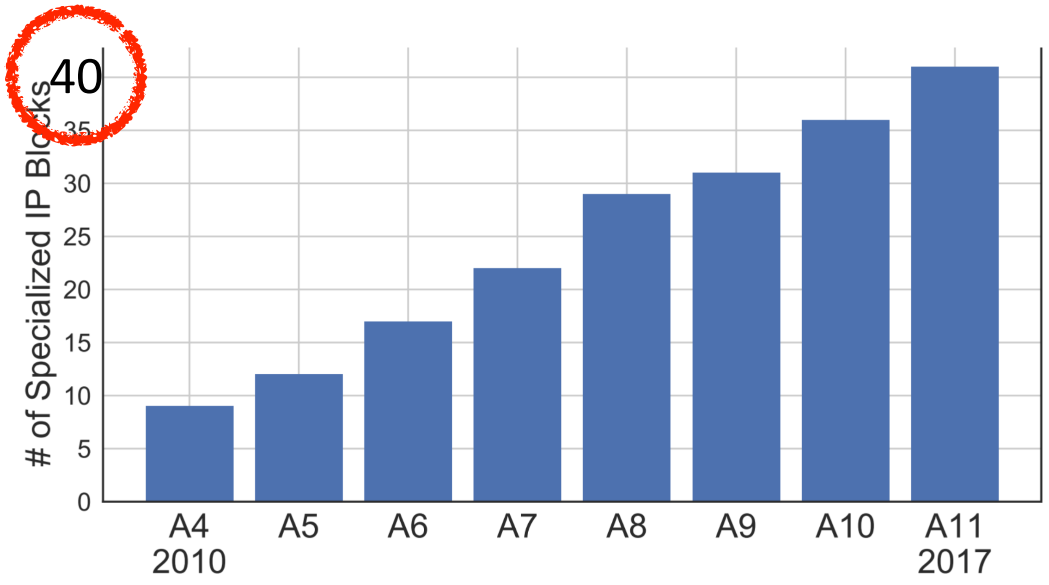


Source: David Brooks

Challenges: The Rise of Heterogeneity

End of Dennard scaling

Need power efficient alternatives to general purpose computing



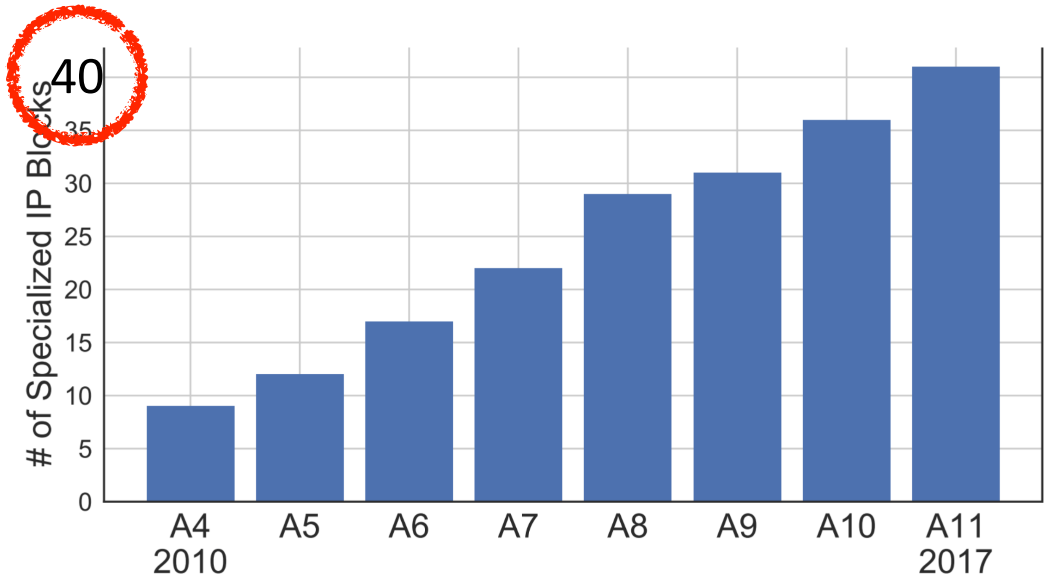
Source: David Brooks

Not just Machine Learning

Challenges: The Rise of Heterogeneity

End of Dennard scaling

Need power efficient alternatives to general purpose computing

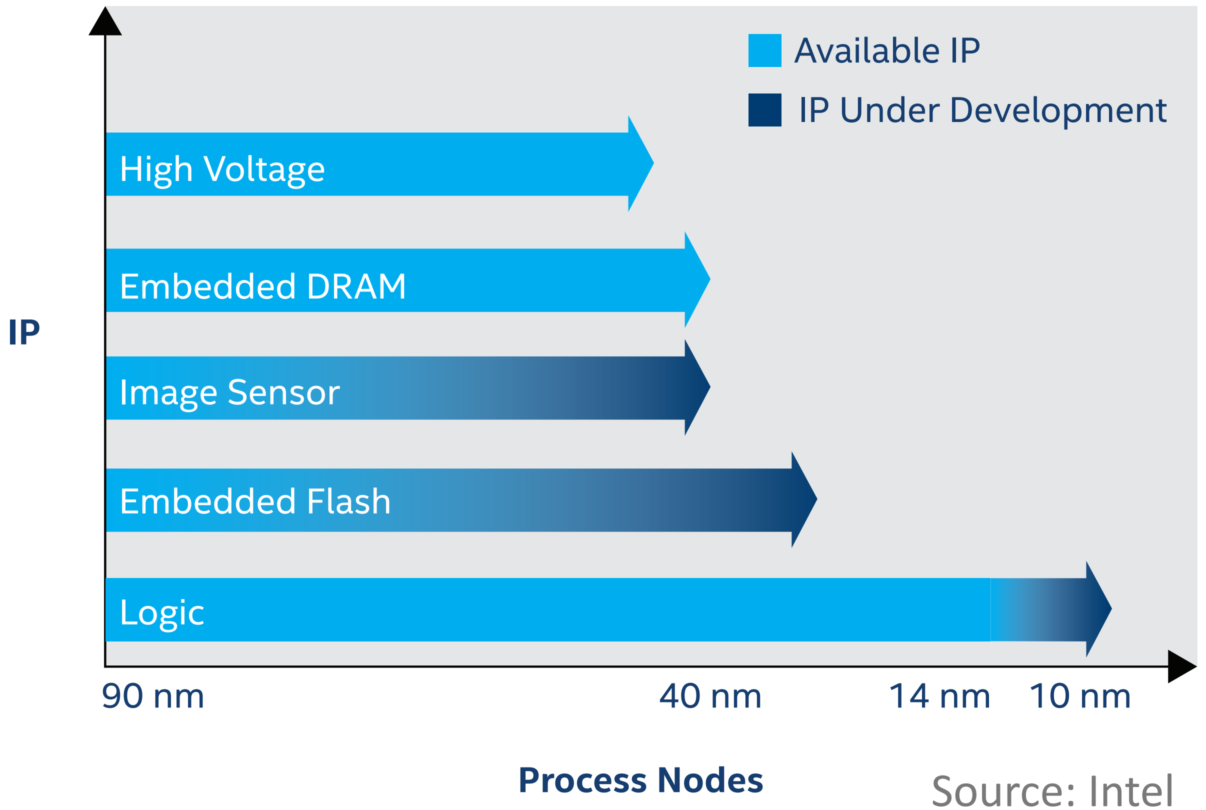


Source: David Brooks

Not just Machine Learning

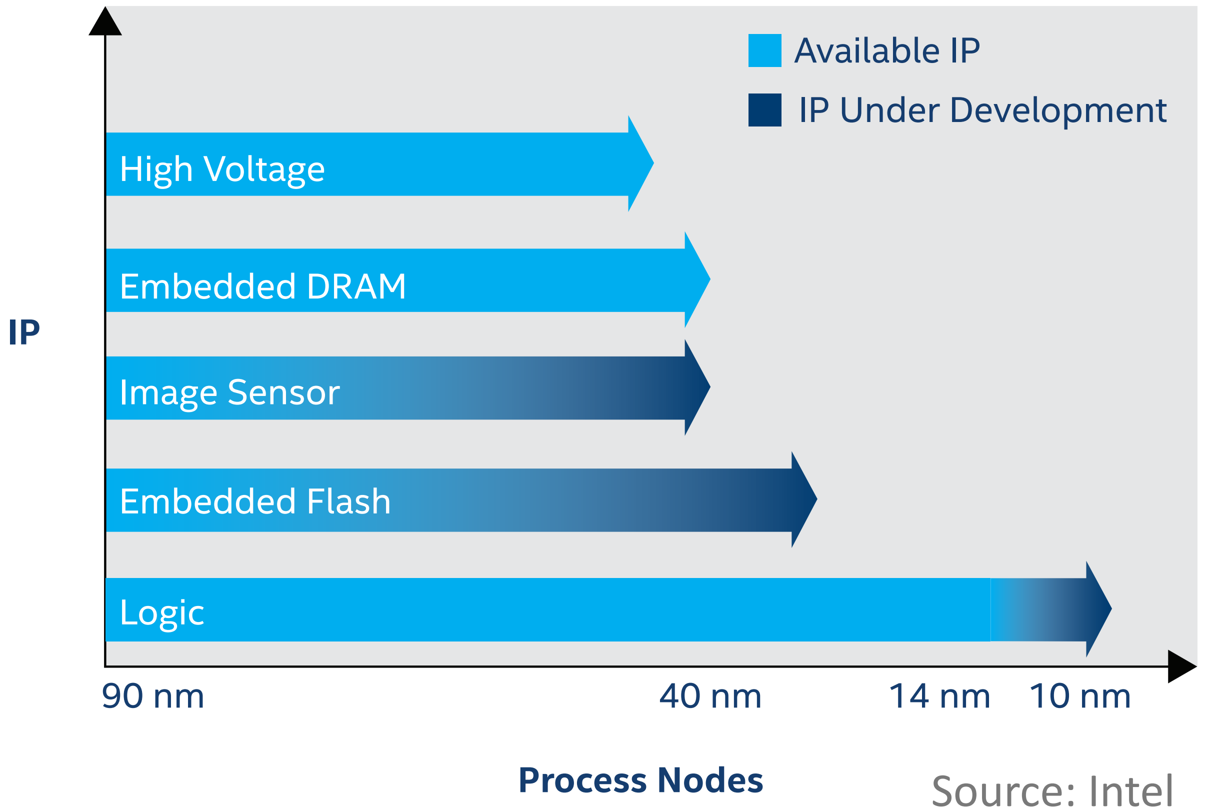
SoC integration challenges for datacentres, cellphones

Challenges: Rise of Heterogeneity



Source: Intel

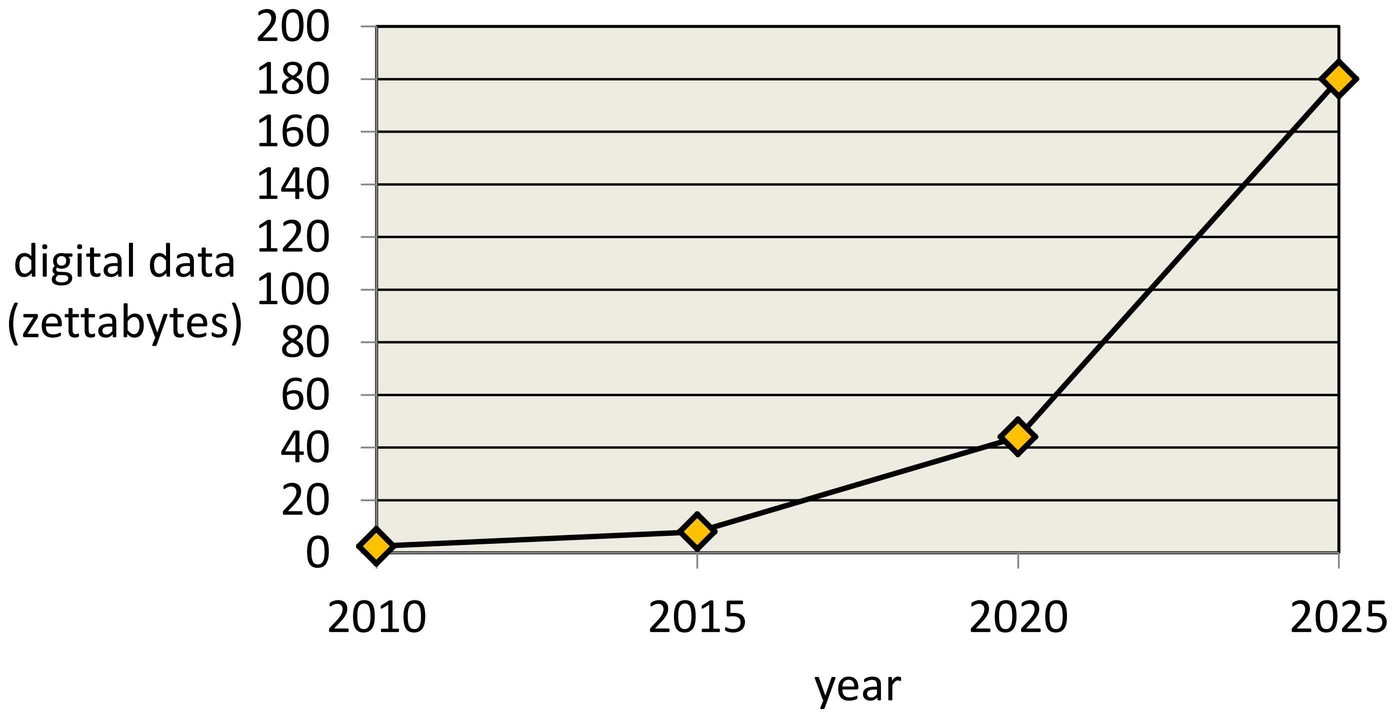
Challenges: Rise of Heterogeneity



Source: Intel

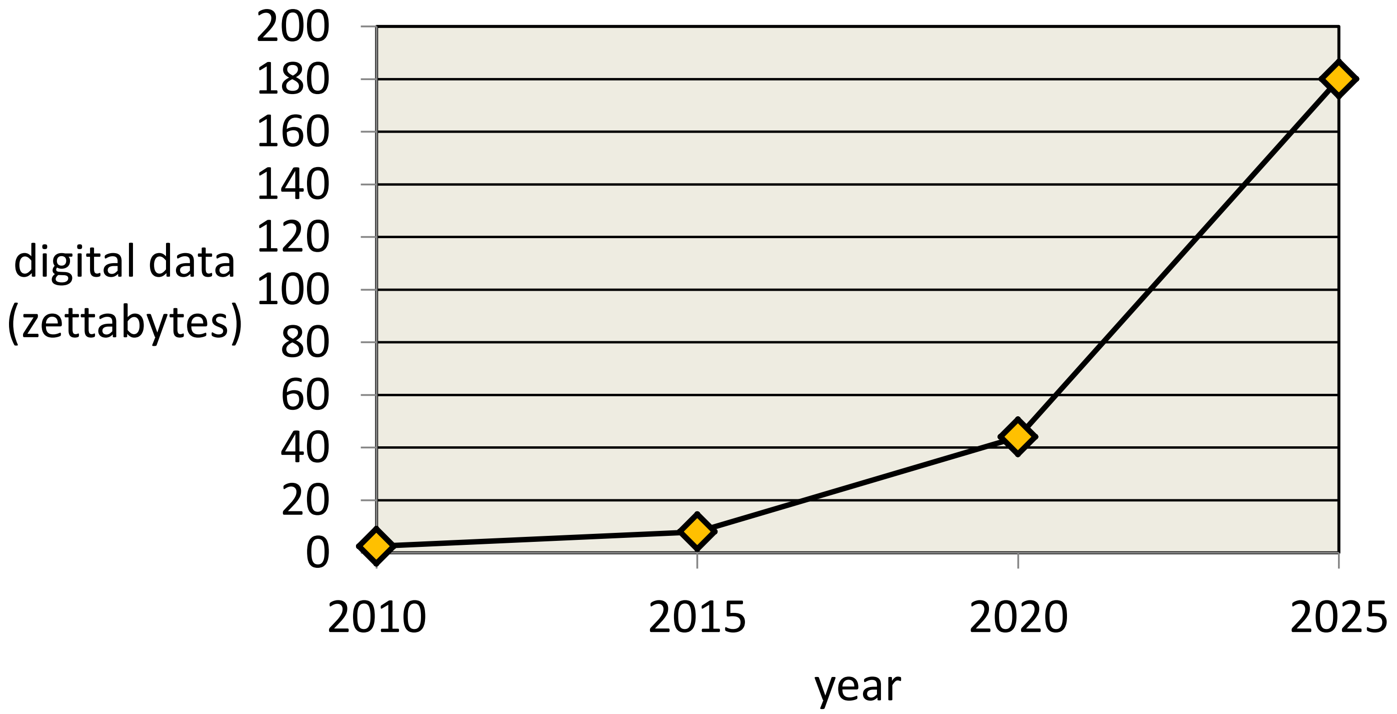
Heterogeneous manufacturing processes for different IP

Challenges: Big Data



Source: International Data Corporation, 2016

Challenges: Big Data



Workloads increasingly memory and communication bound
Need to integrate lots of memory!

What do we need?

What do we need?

A means to continue integrating more functionality

What do we need?

A means to continue integrating more functionality

A means to deal with IP and manufacturing heterogeneity

What do we need?

A means to continue integrating more functionality

A means to deal with IP and manufacturing heterogeneity

A means to enable greater memory integration and efficient communication

What do we need?

A means to continue integrating more functionality

A means to deal with IP and manufacturing heterogeneity

A means to enable greater memory integration and efficient communication

All while combating skyrocketing manufacturing costs

What do we need?

A means to continue integrating more functionality

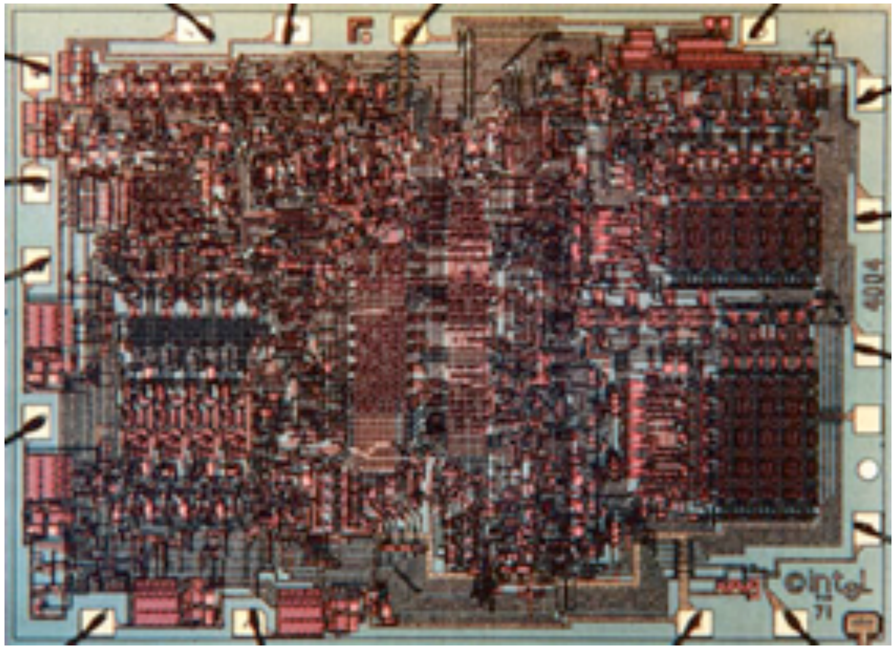
A means to deal with IP and manufacturing hetero

But first…

A mea and efficient communication

All while combating skyrocketing manufacturing costs

Walk down memory lane (1971)



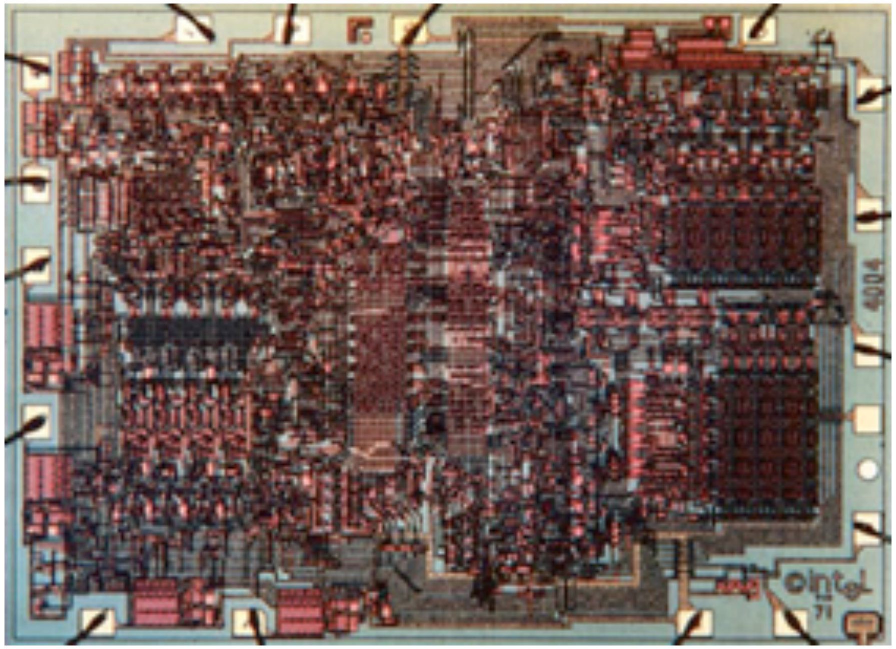
Intel introduces 4004

1st commercial microprocessor

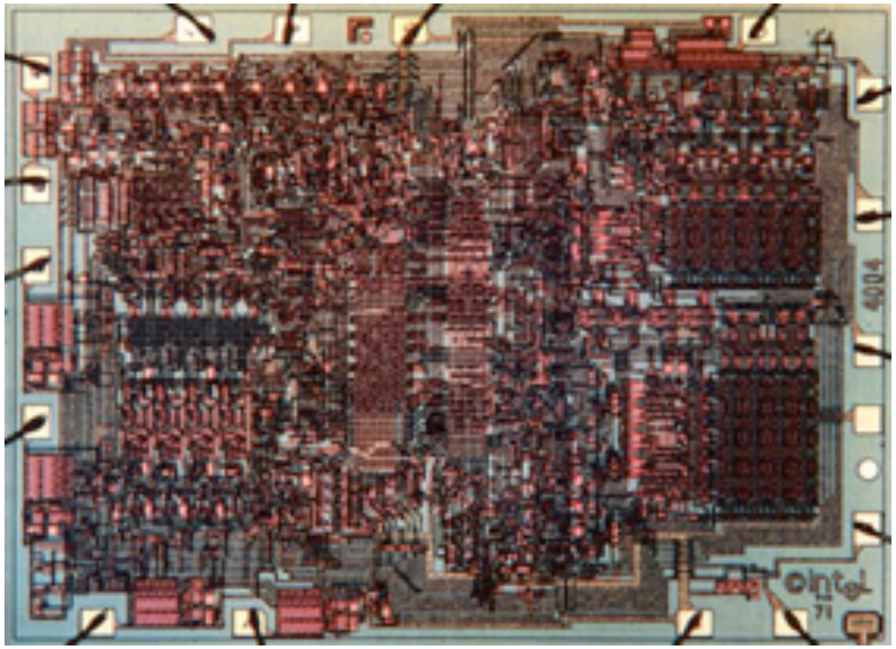
2300 transistors

13mm^2

Walk down memory lane (1971)

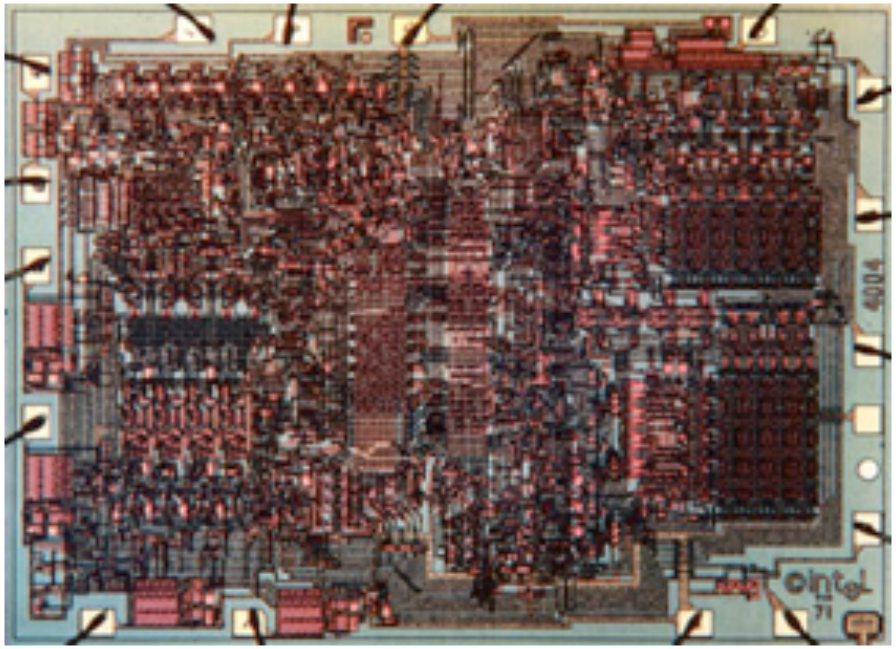


Walk down memory lane (1971)



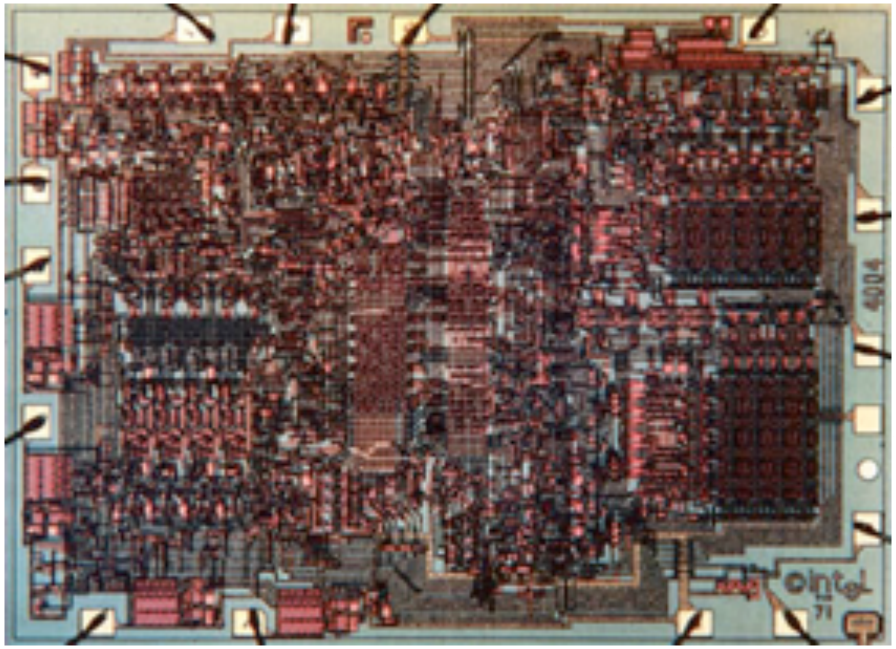
😁 Everything on one chip

Walk down memory lane (1971)



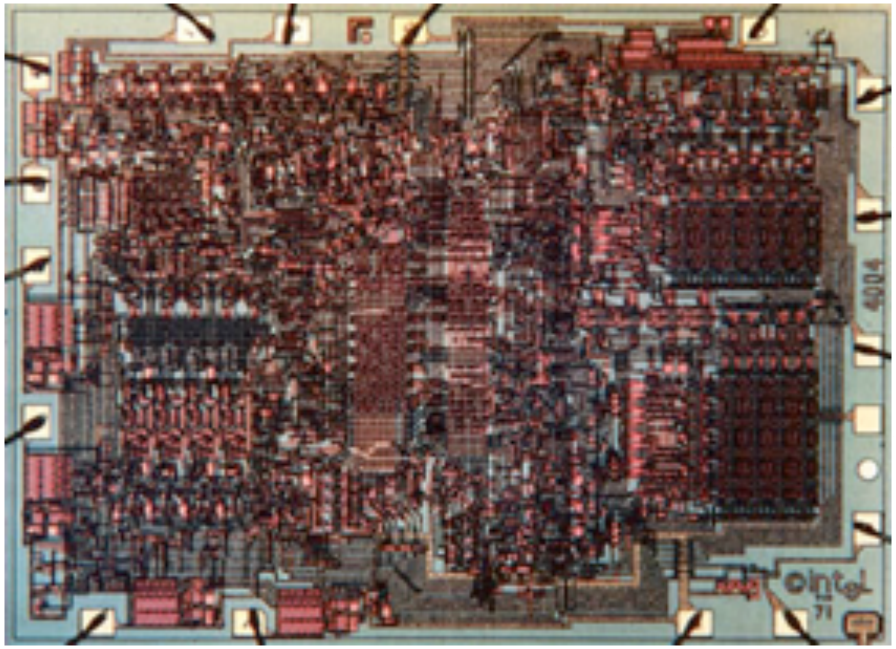
- 😁Everything on one chip
- 😁No more slow chip crossings

Walk down memory lane (1971)



- 😁Everything on one chip
- 😁No more slow chip crossings
- 😁Cheaper manufacturing!

Walk down memory lane (1971)



- 😁Everything on one chip
- 😁No more slow chip crossings
- 😁Cheaper manufacturing!

A sea change for the computer industry

Disintegrate chips into chiplets (2020)

Large *Cost-Effective* SoCs through Disintegration

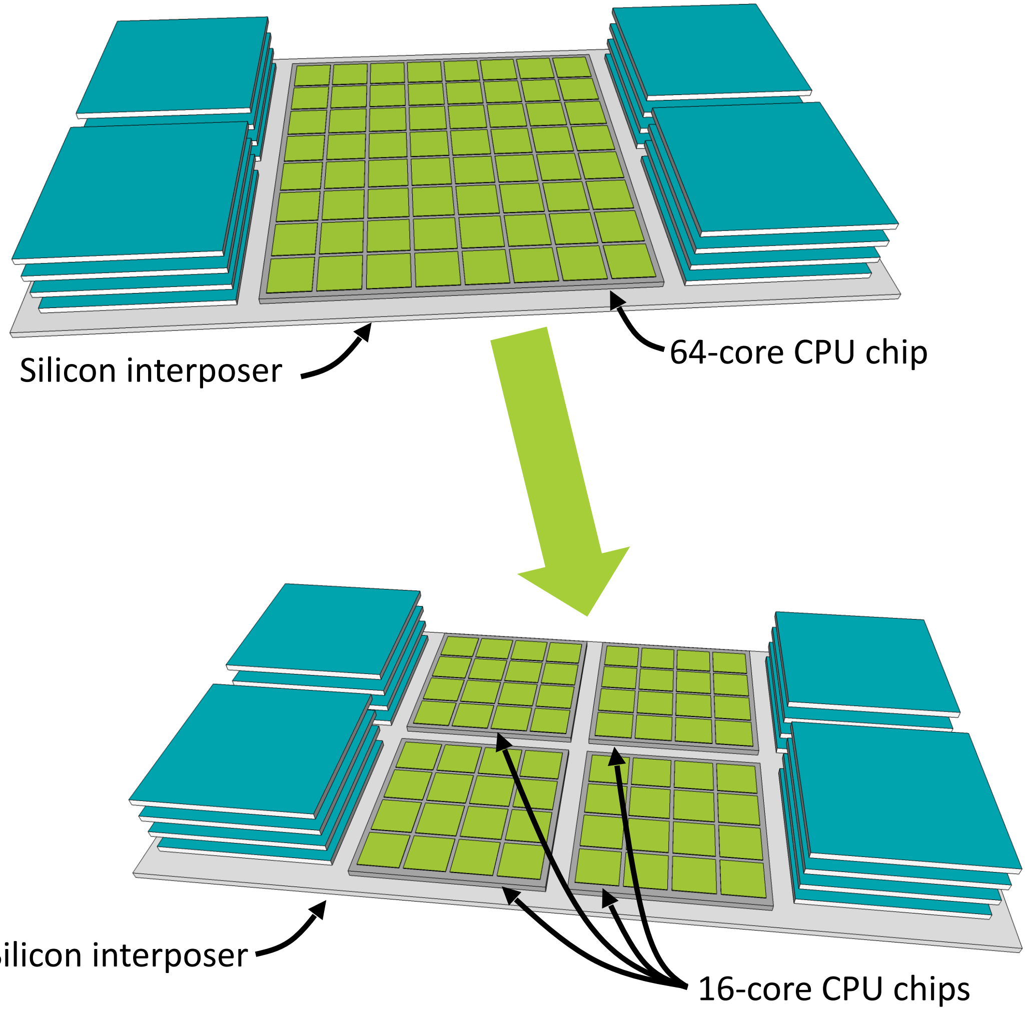
Why disintegrate?

Want more functionality, but…

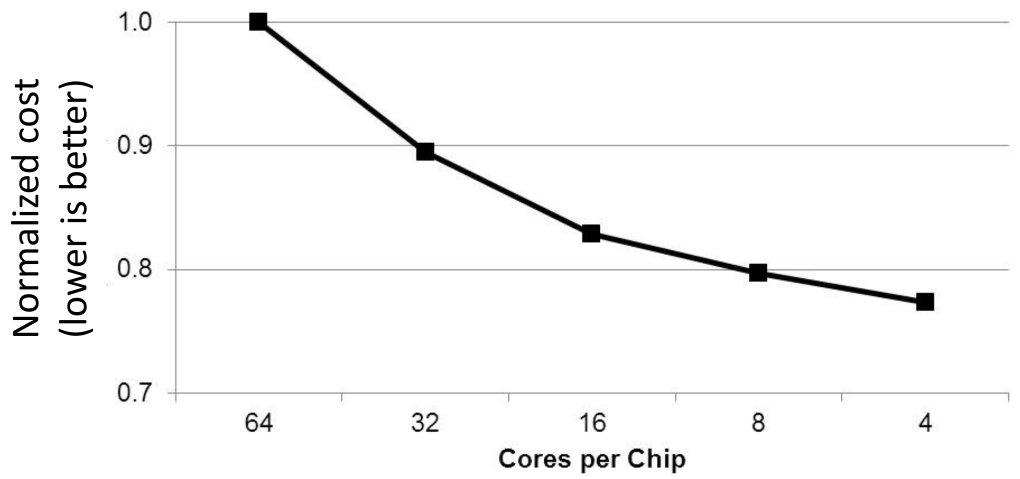
Big chips are expensive

Break (disintegrate) into several smaller pieces

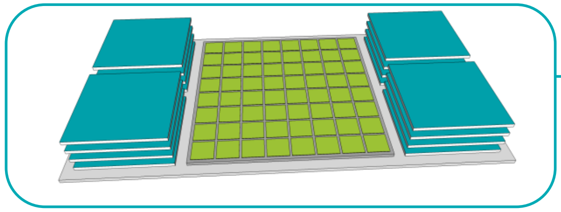
Cheaper to manufacture

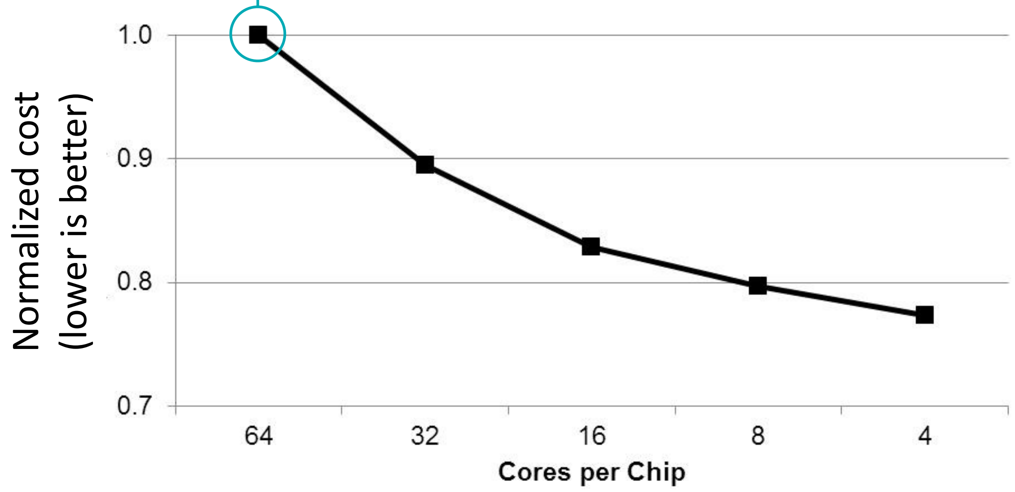


Disintegration → Cheaper SoCs

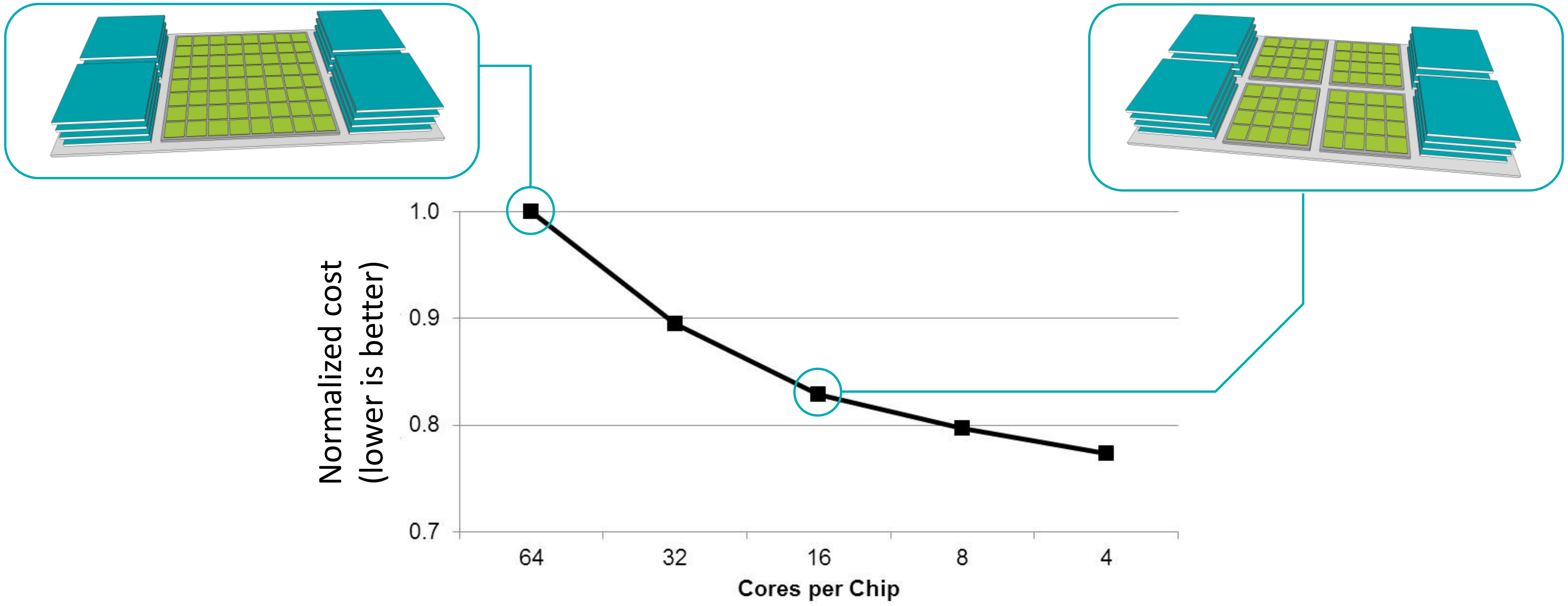


Disintegration → Cheaper SoCs

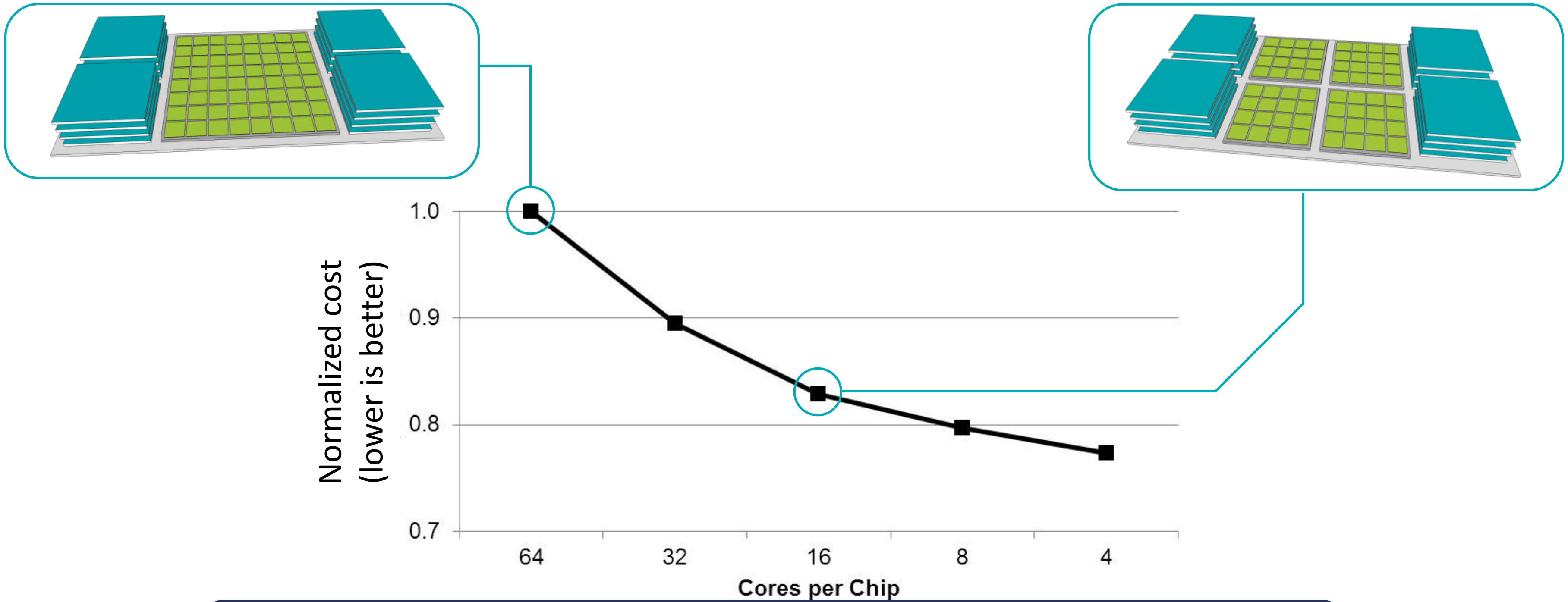




Disintegration → Cheaper SoCs

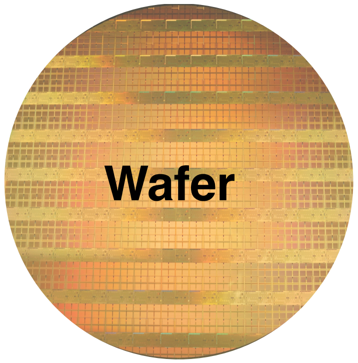


Disintegration → Cheaper SoCs

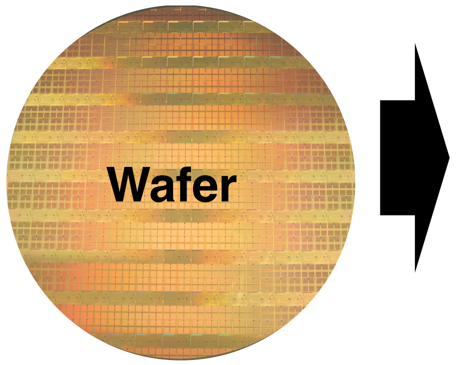


Disintegrated SoCs have potential for reducing costs of large chips while maintaining functionality

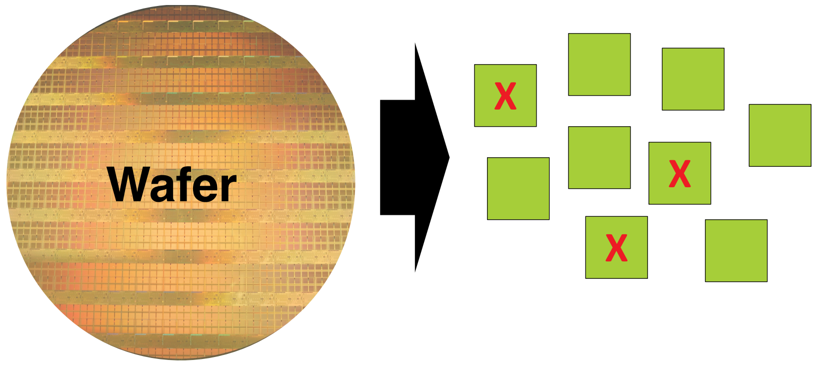
Cost Argument: High-Level Idea



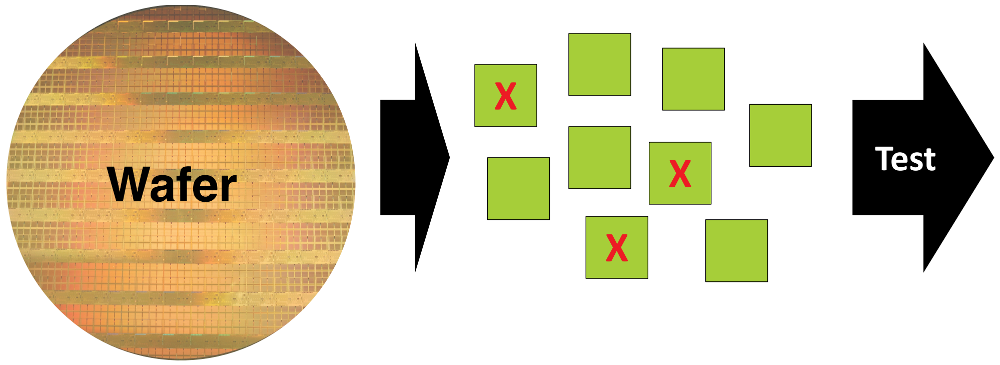
Cost Argument: High-Level Idea



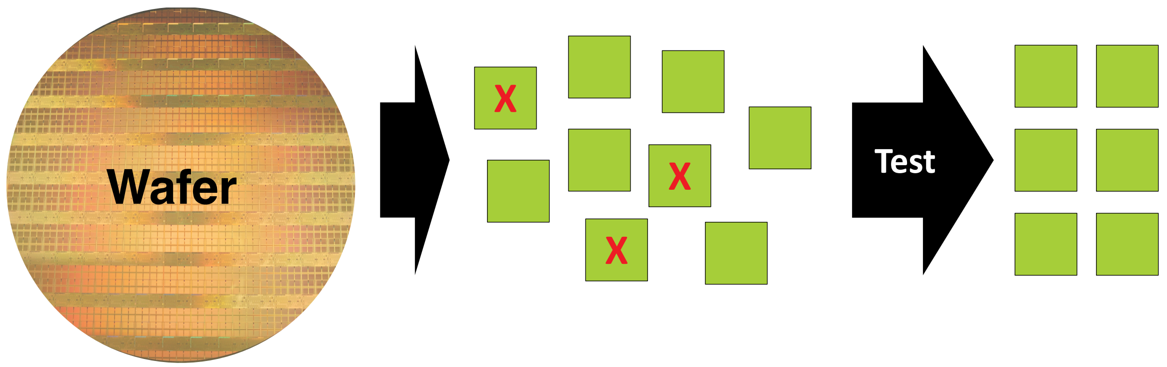
Cost Argument: High-Level Idea



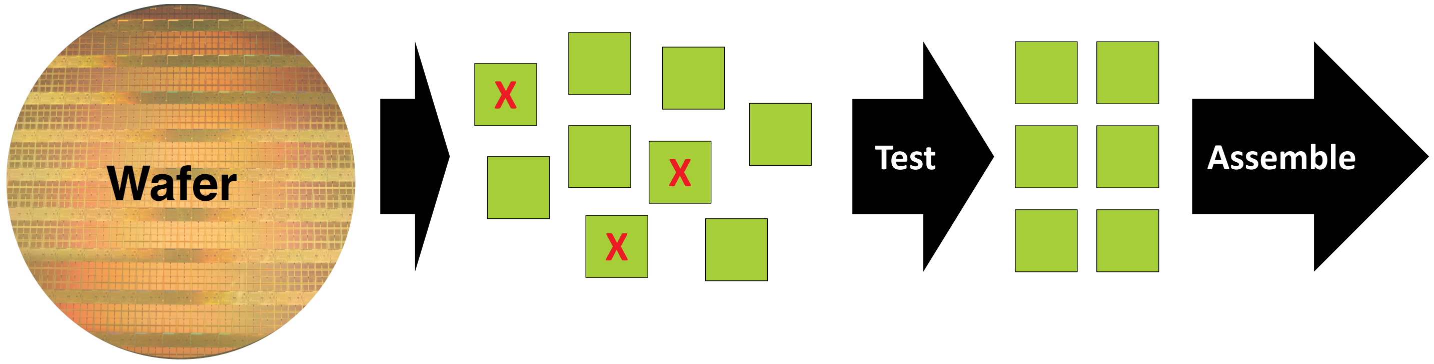
Cost Argument: High-Level Idea



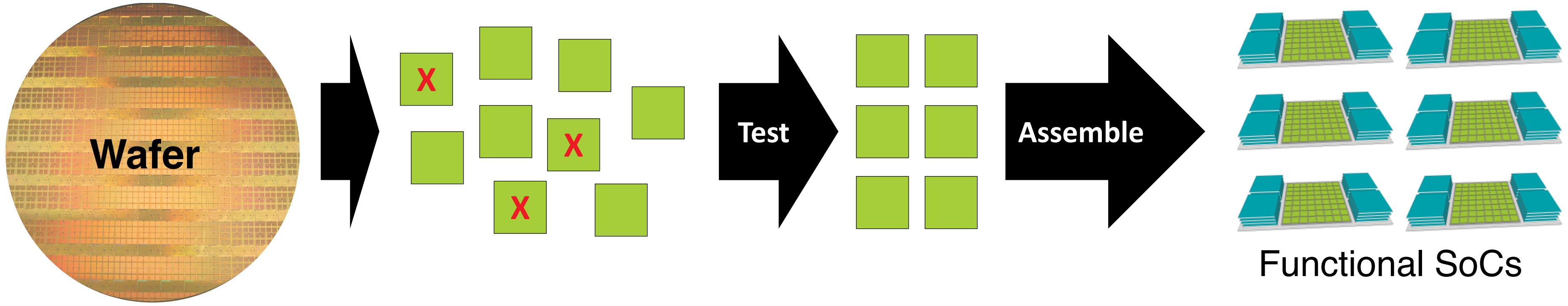
Cost Argument: High-Level Idea



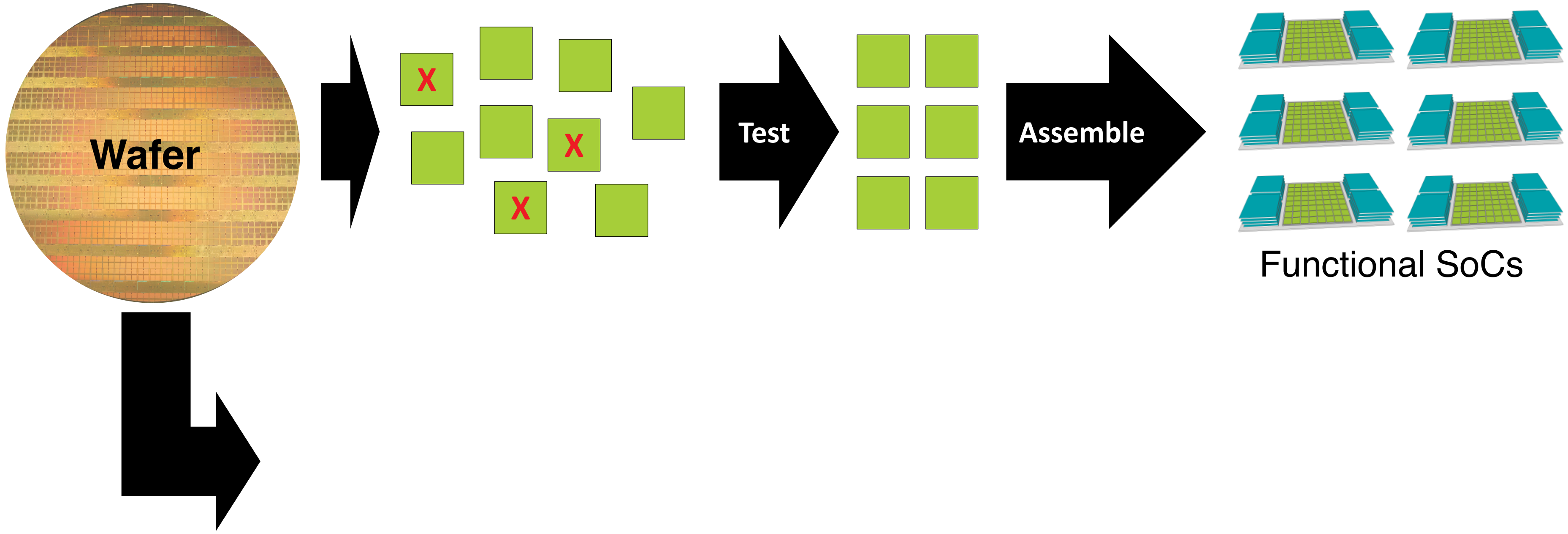
Cost Argument: High-Level Idea



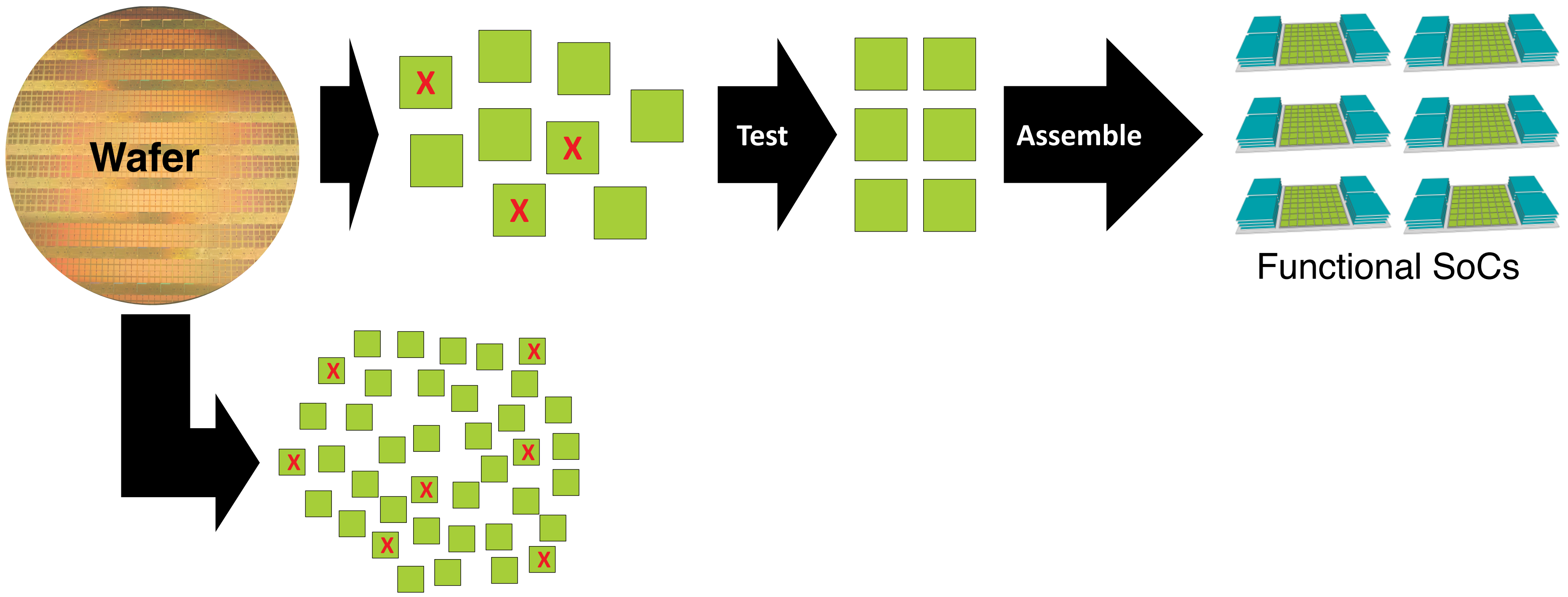
Cost Argument: High-Level Idea



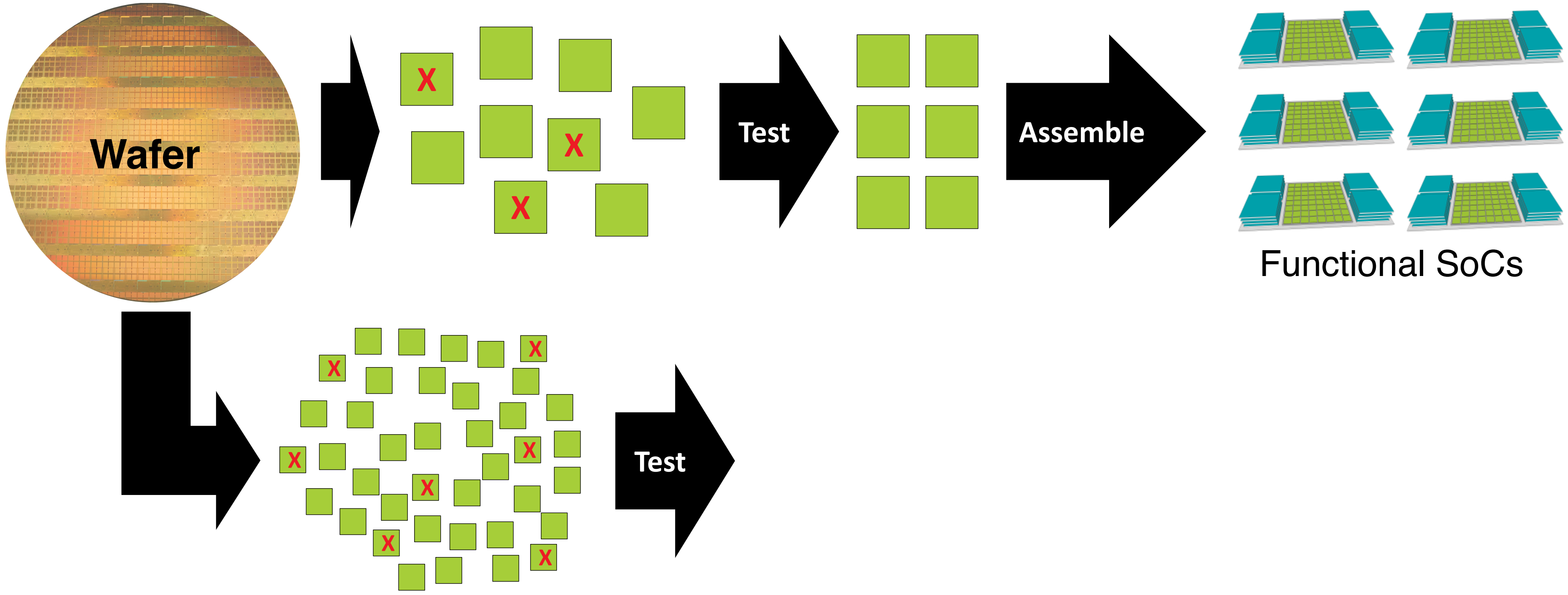
Cost Argument: High-Level Idea



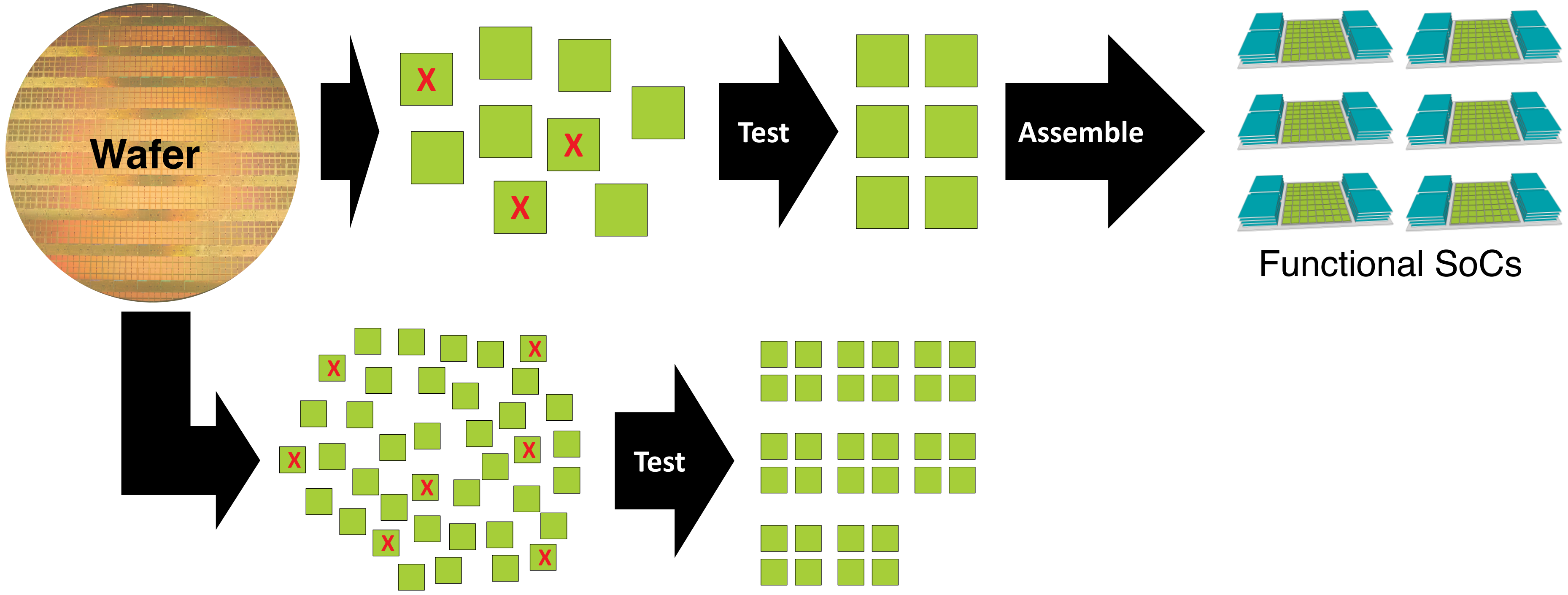
Cost Argument: High-Level Idea



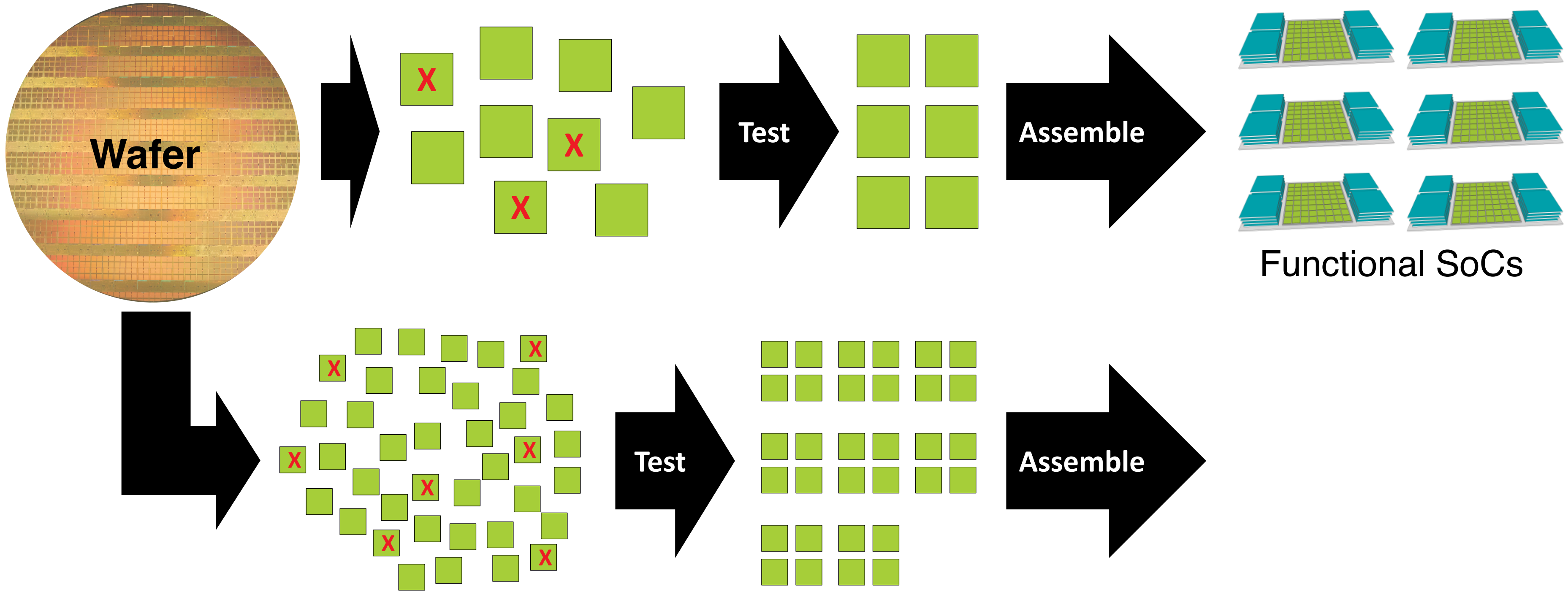
Cost Argument: High-Level Idea



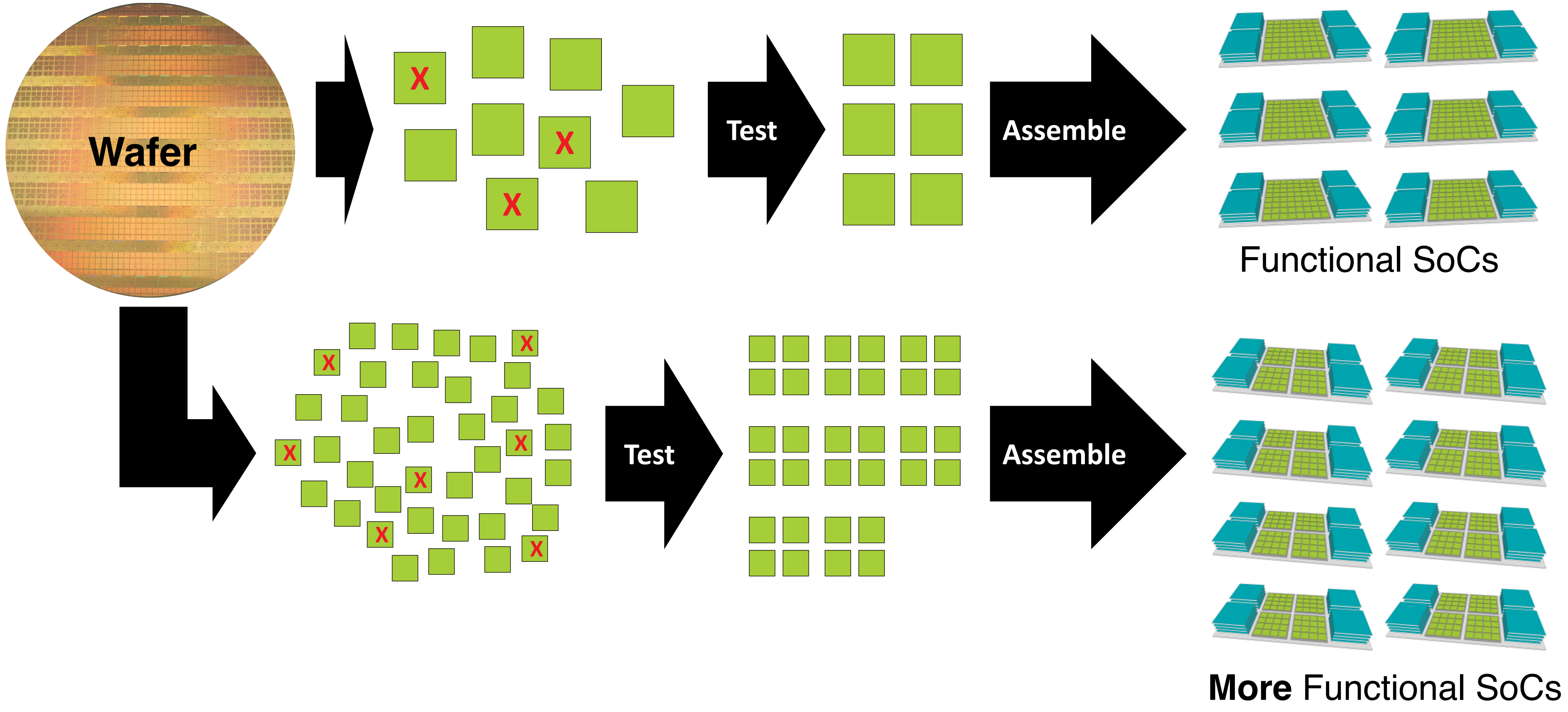
Cost Argument: High-Level Idea



Cost Argument: High-Level Idea



Cost Argument: High-Level Idea



Cheaper Chips + Larger Profit Margins

Cheaper Chips + Larger Profit Margins

Process variations lead to different maximum operating frequencies

Cheaper Chips + Larger Profit Margins

Process variations lead to different maximum operating frequencies

Sort chips before assembly to improve speed binning

Cheaper Chips + Larger Profit Margins

Process variations lead to different maximum operating frequencies

Sort chips before assembly to improve speed binning

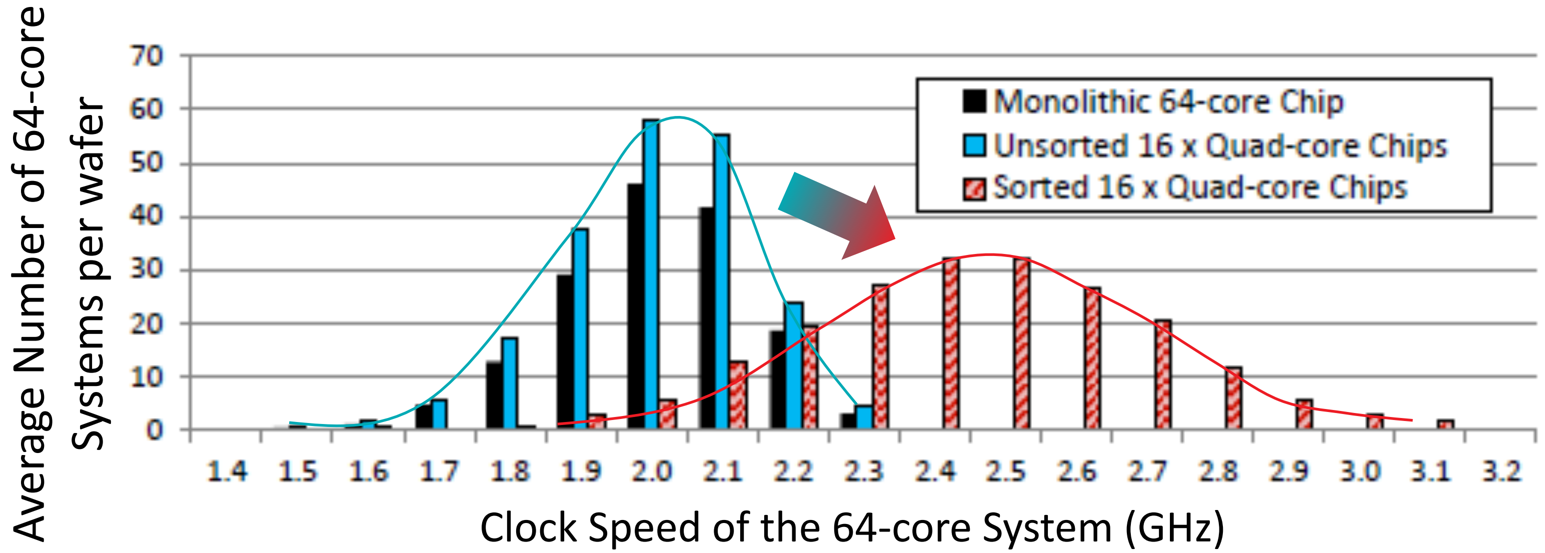
Within die variations hurt performance of large monolithic chips

Cheaper Chips + Larger Profit Margins

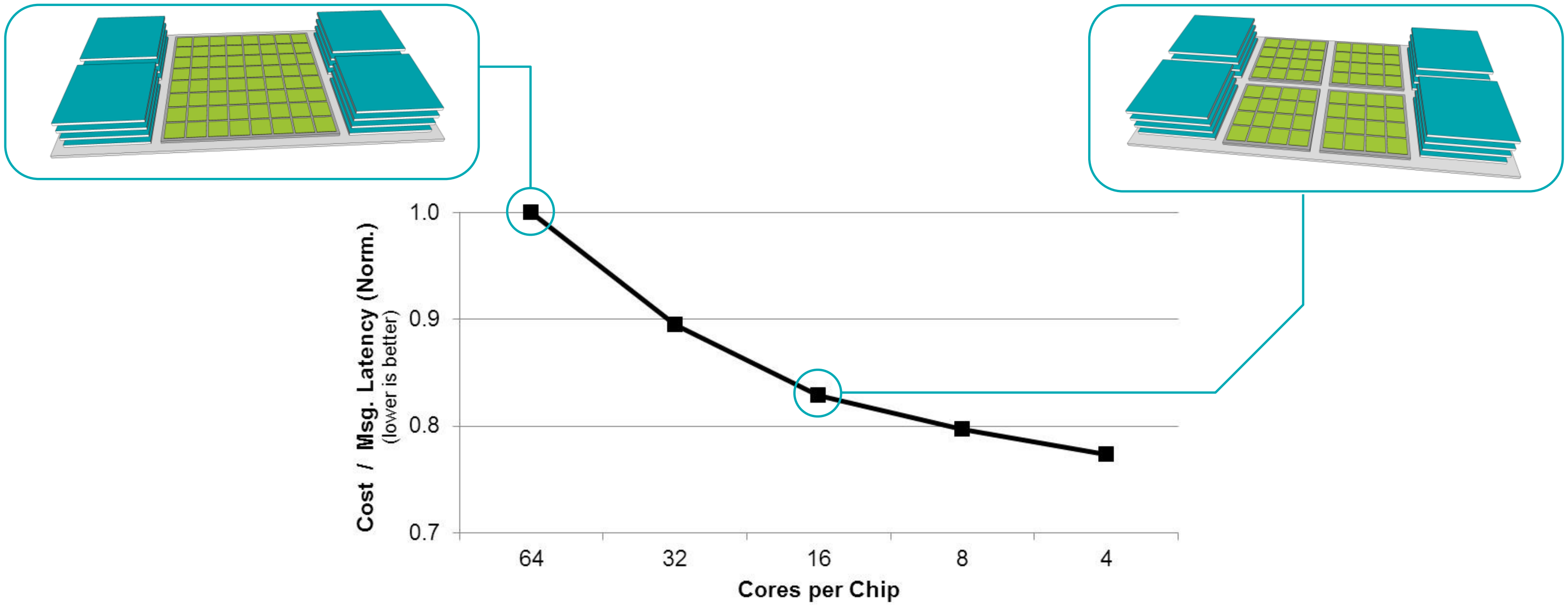
Process variations lead to different maximum operating frequencies

Sort chips before assembly to improve speed binning

Within die variations hurt performance of large monolithic chips

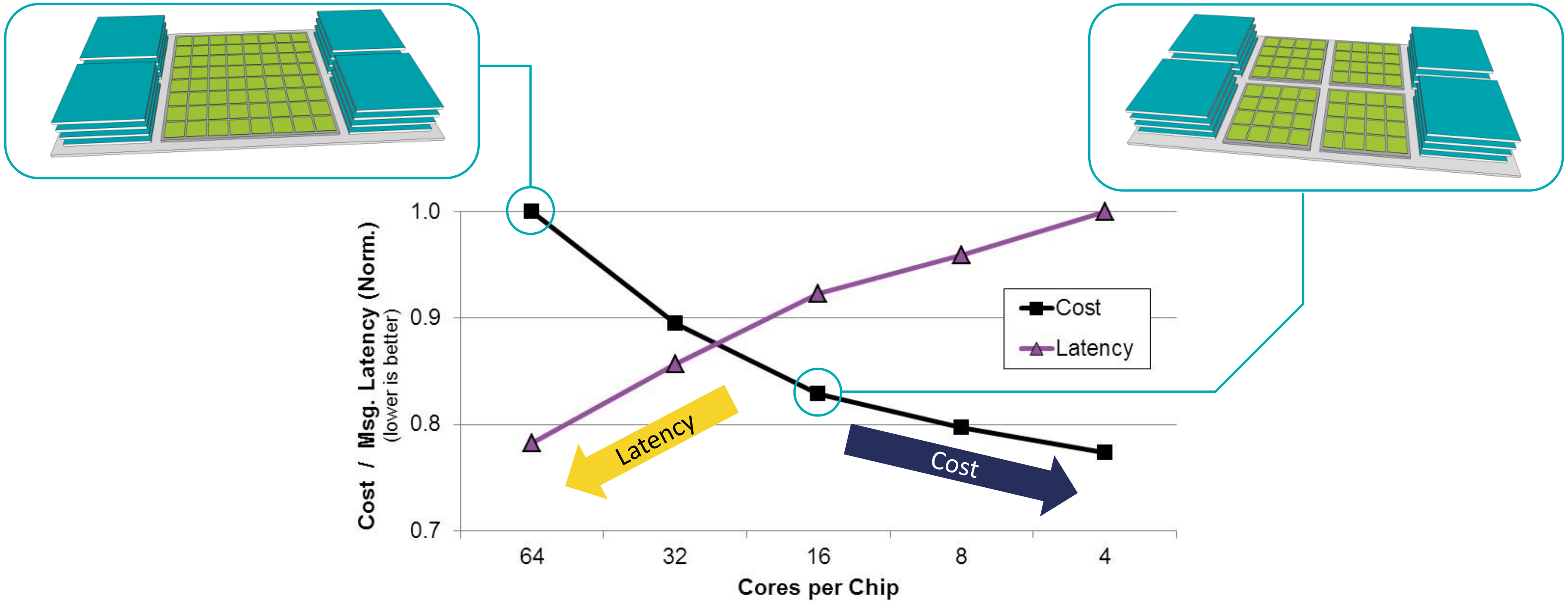


Fragmented Architecture



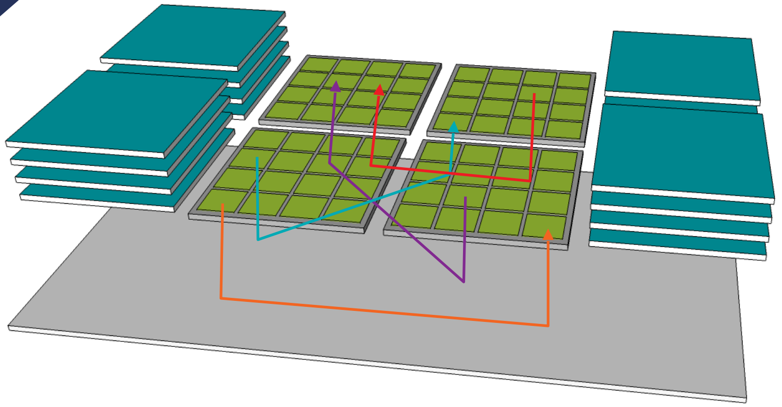
Disintegrated SoCs have potential for reducing costs of large chips

Fragmented Architecture



Disintegrated SoCs have potential for reducing costs of large chips

But performance degrades with disintegration granularity



How to integrate chiplets?

What are we looking for when reintegrating?

What are we looking for when reintegrating?

Enable small/simple chiplets

What are we looking for when reintegrating?

Enable small/simple chiplets

High bandwidth/low latency connections between chiplets

What are we looking for when reintegrating?

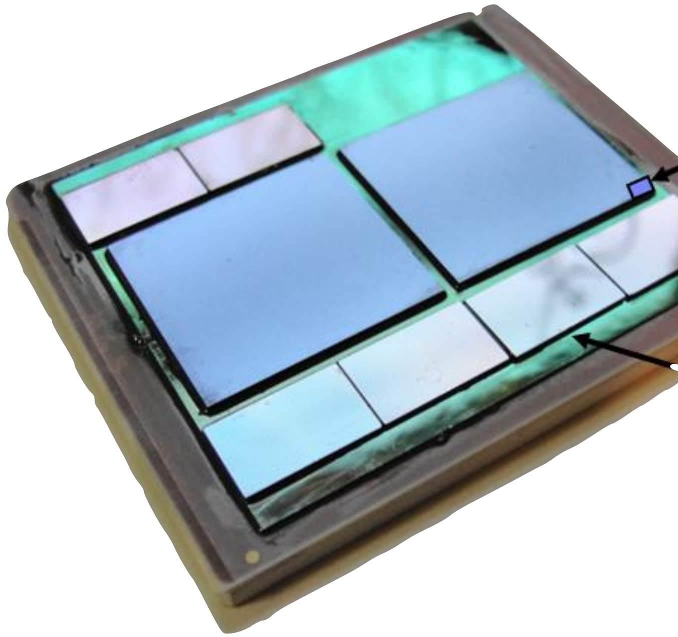
Enable small/simple chiplets

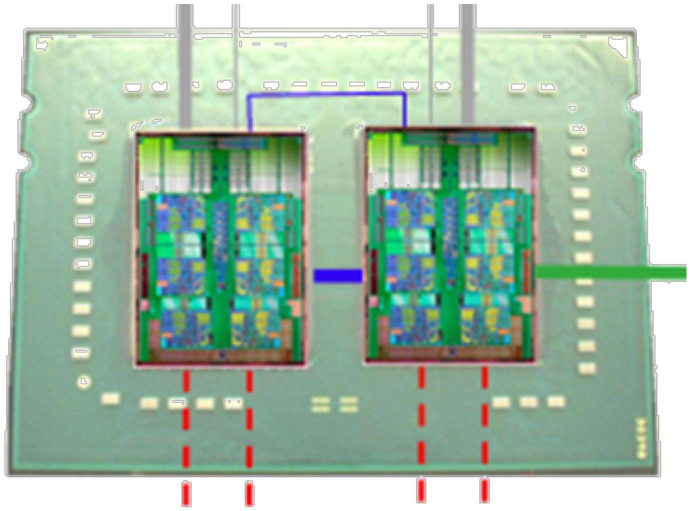
High bandwidth/low latency connections between chiplets

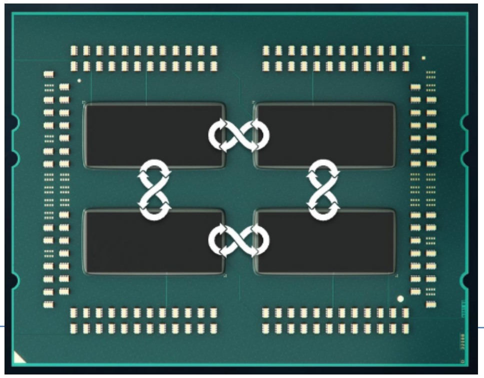
Ease of manufacturing

How to integrate?

Multi-chip modules (MCM)



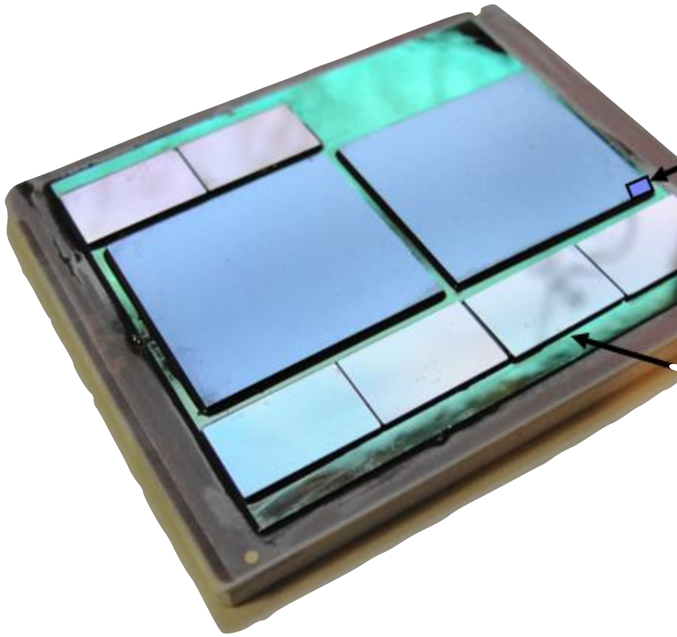


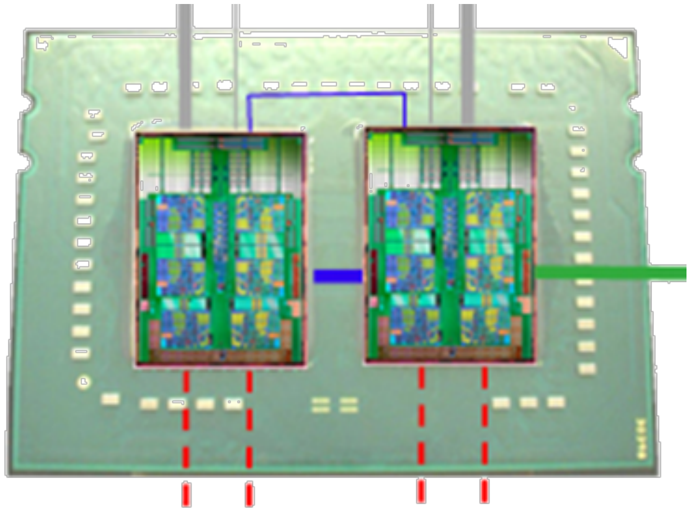


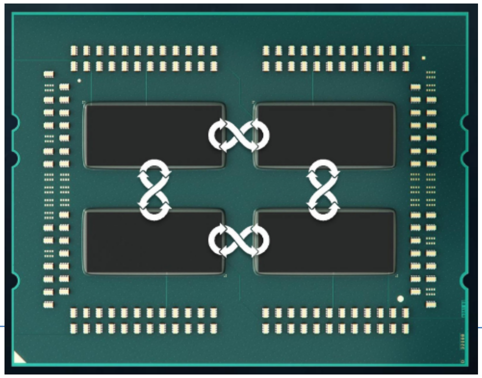
How to integrate?

Multi-chip modules (MCM)

😁 Avoids pin limitations of multi-package solutions



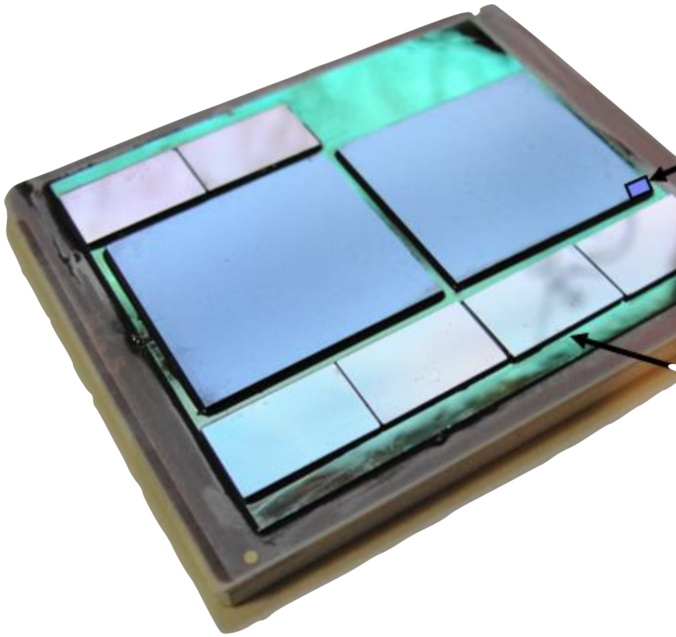


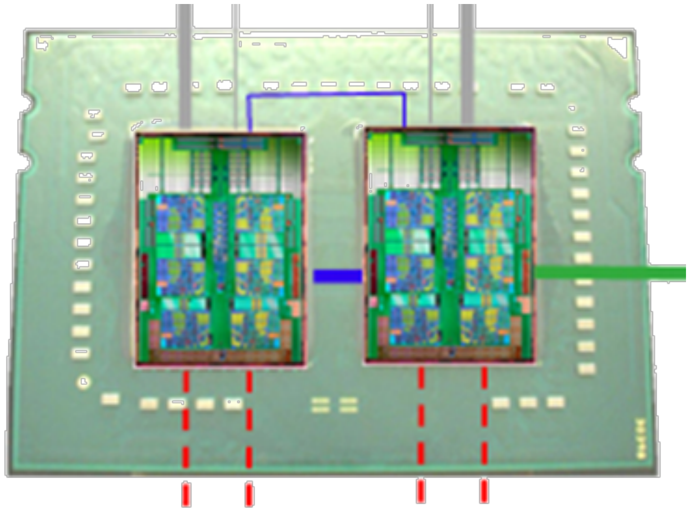


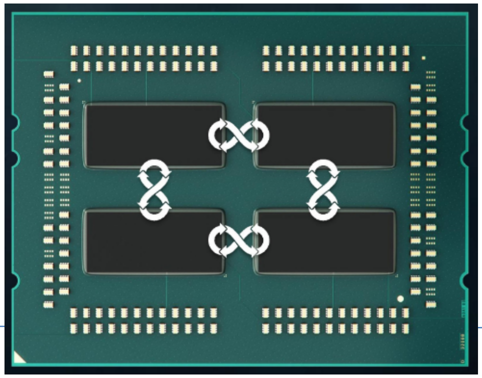
How to integrate?

Multi-chip modules (MCM)

- 😁 Avoids pin limitations of multi-package solutions
- 😢 Bandwidth/Latency constraints of C4 bumps and substrate



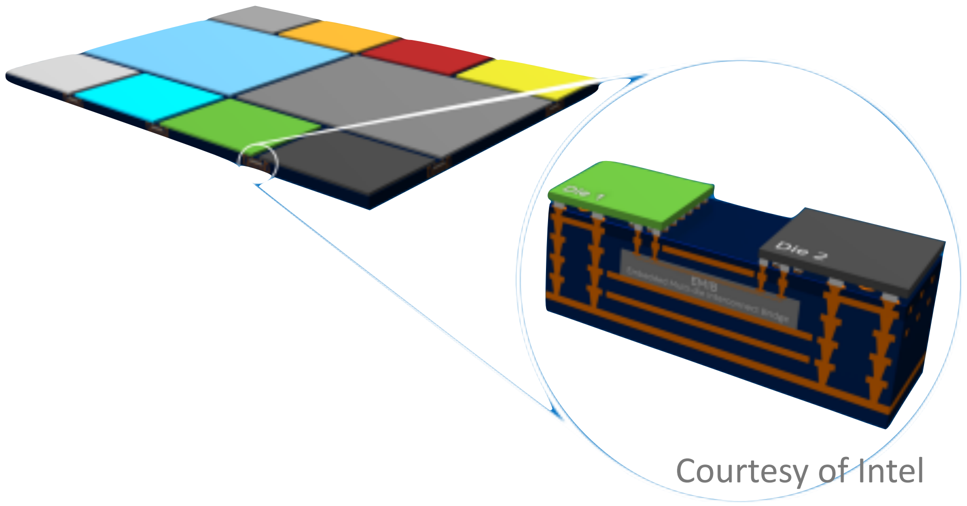




How to integrate?

Multi-chip modules (MCM)

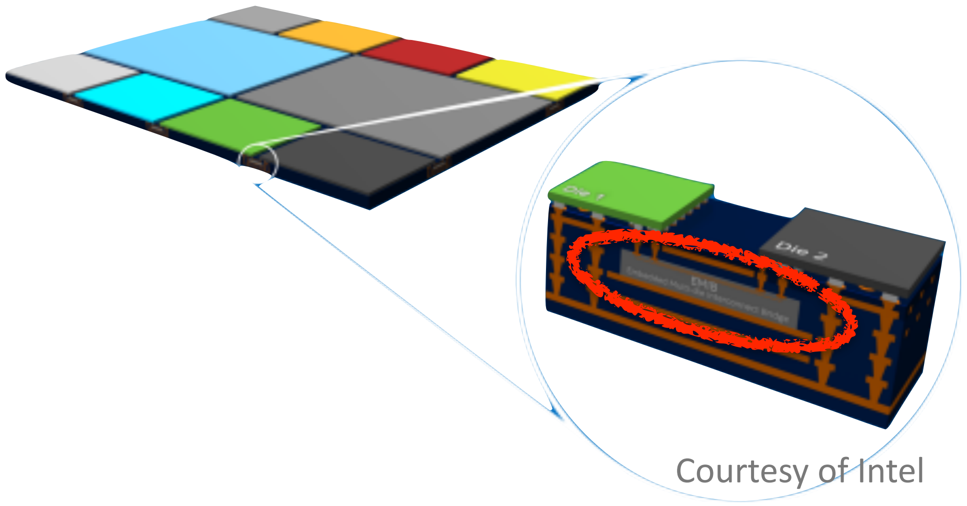
Embedded Multi-Chip Interconnect Bridge (EMIB)



How to integrate?

Multi-chip modules (MCM)

Embedded Multi-Chip Interconnect Bridge (EMIB)

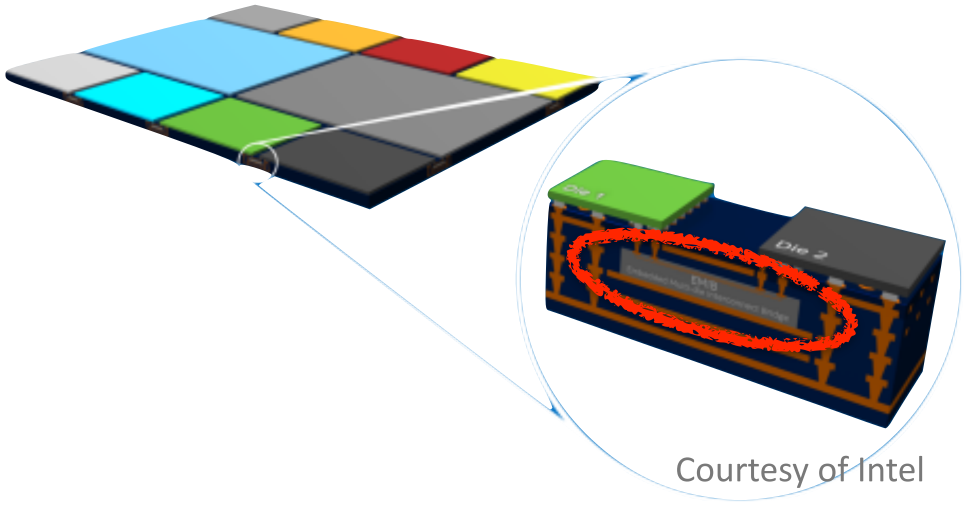


How to integrate?

Multi-chip modules (MCM)

Embedded Multi-Chip Interconnect Bridge (EMIB)

- 😁 Small/simple chiplets

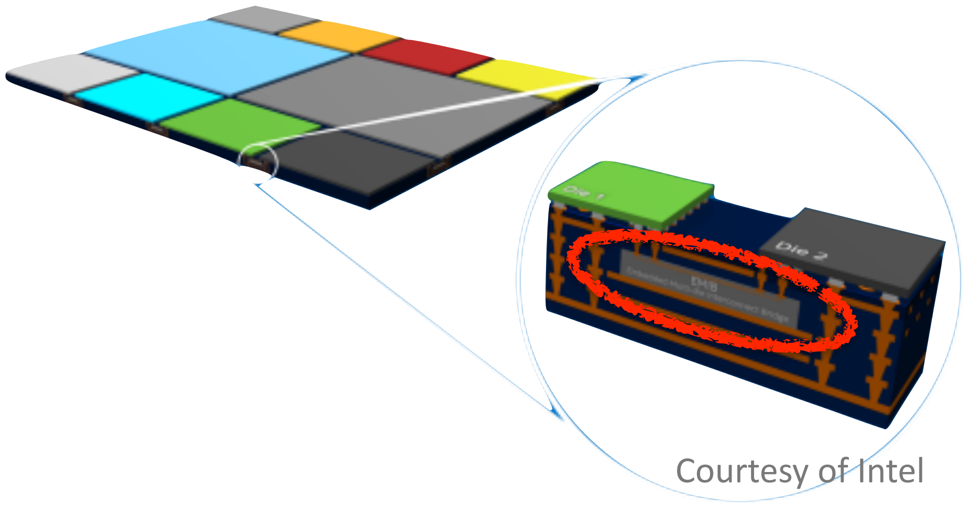


How to integrate?

Multi-chip modules (MCM)

Embedded Multi-Chip Interconnect Bridge (EMIB)

- 😁 Small/simple chiplets
- 😁 Small bridge die



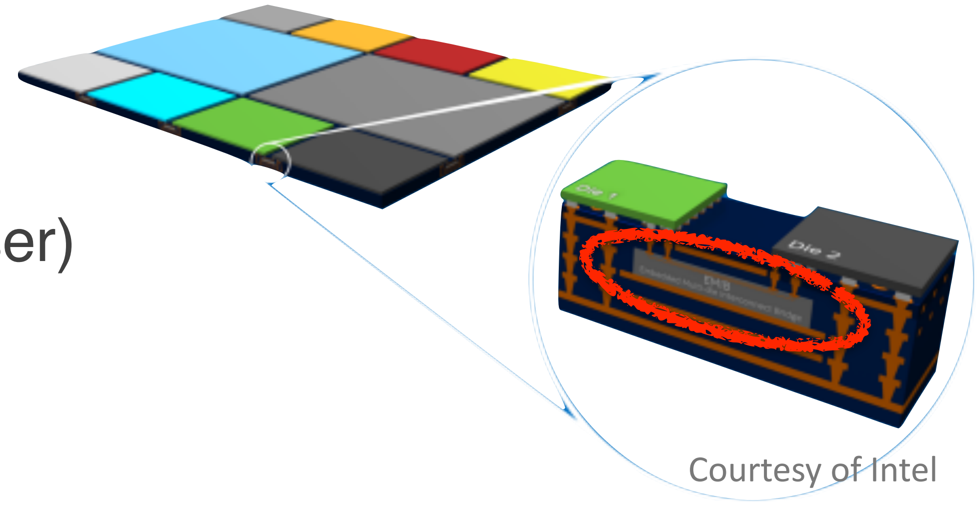
Courtesy of Intel

How to integrate?

Multi-chip modules (MCM)

Embedded Multi-Chip Interconnect Bridge (EMIB)

- 😁 Small/simple chiplets
- 😁 Small bridge die
- 😁 Avoids large die (interposer)

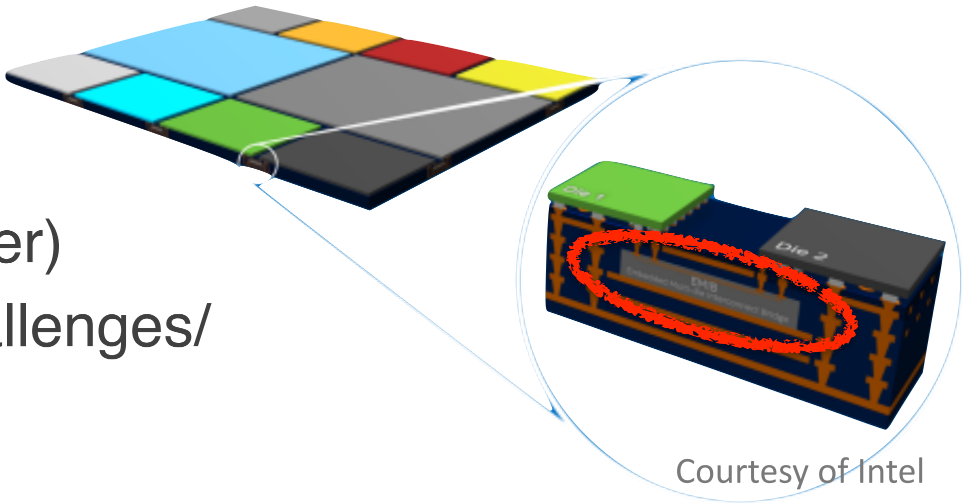


How to integrate?

Multi-chip modules (MCM)

Embedded Multi-Chip Interconnect Bridge (EMIB)

- 😁 Small/simple chiplets
- 😁 Small bridge die
- 😁 Avoids large die (interposer)
- 😁 Avoids manufacturing challenges/ costs

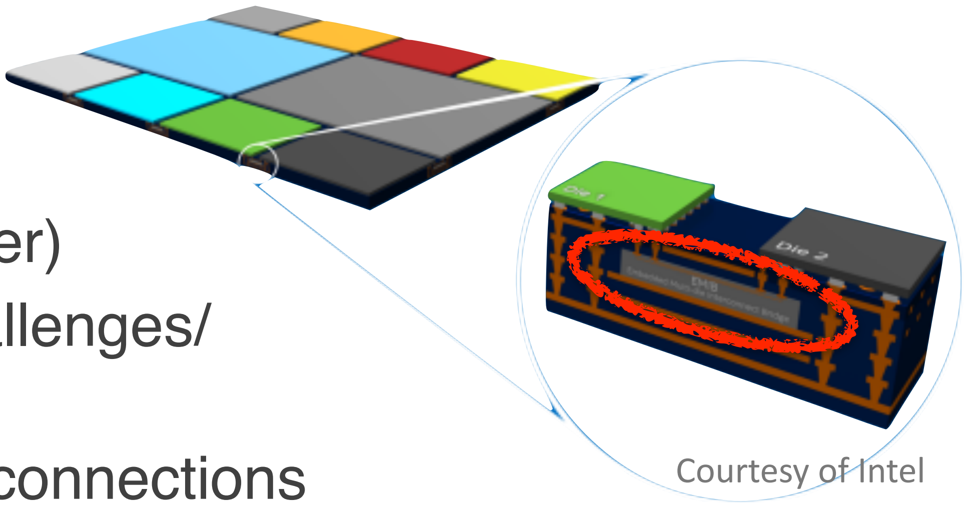


How to integrate?

Multi-chip modules (MCM)

Embedded Multi-Chip Interconnect Bridge (EMIB)

- 😁 Small/simple chiplets
- 😁 Small bridge die
- 😁 Avoids large die (interposer)
- 😁 Avoids manufacturing challenges/ costs
- 😢 Only offers point-to-point connections



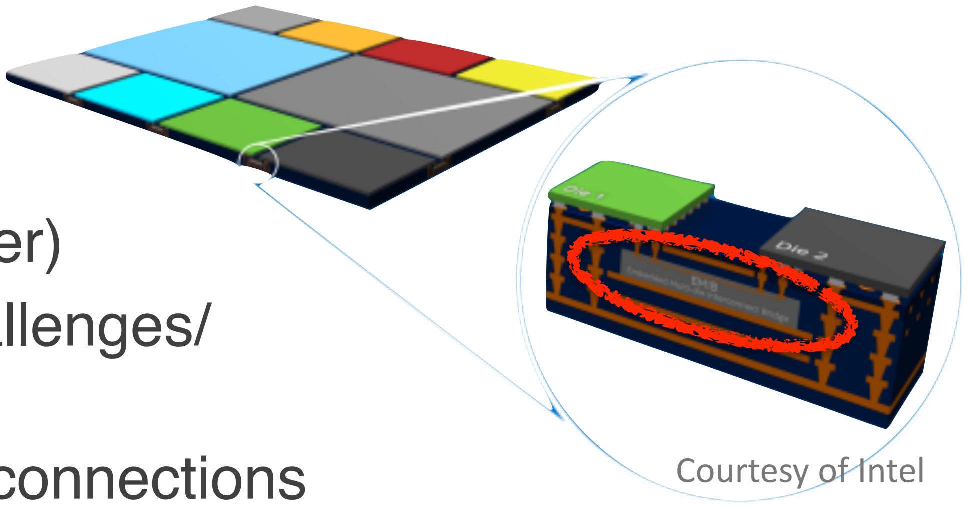
Courtesy of Intel

How to integrate?

Multi-chip modules (MCM)

Embedded Multi-Chip Interconnect Bridge (EMIB)

- 😁 Small/simple chiplets
- 😁 Small bridge die
- 😁 Avoids large die (interposer)
- 😁 Avoids manufacturing challenges/ costs
- 😢 Only offers point-to-point connections
- 😢 Misses opportunity to offload some functionality to interposer



Courtesy of Intel

How to integrate?

Multi-chip modules (MCM)

Embedded Multi-Chip Interconnect Bridge (EMIB)

Silicon Interposer (2.5D)

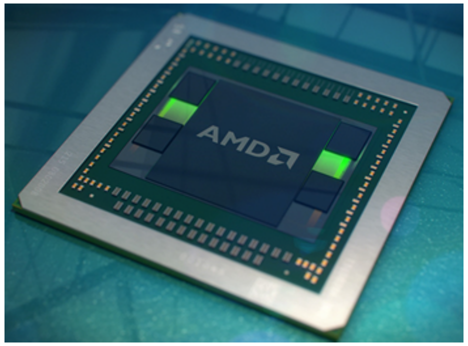
How to integrate?

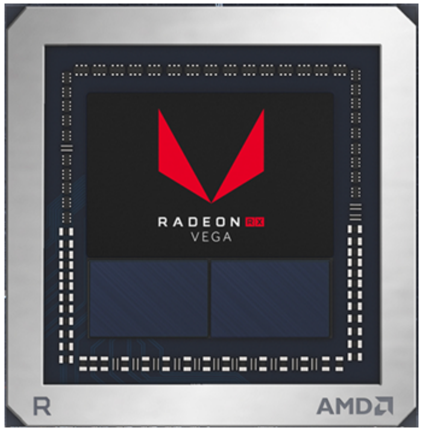
Multi-chip modules (MCM)

Embedded Multi-Chip Interconnect Bridge (EMIB)

Silicon Interposer (2.5D)

- 😁 Technology maturation (high volume passive interposer production — 3 years)



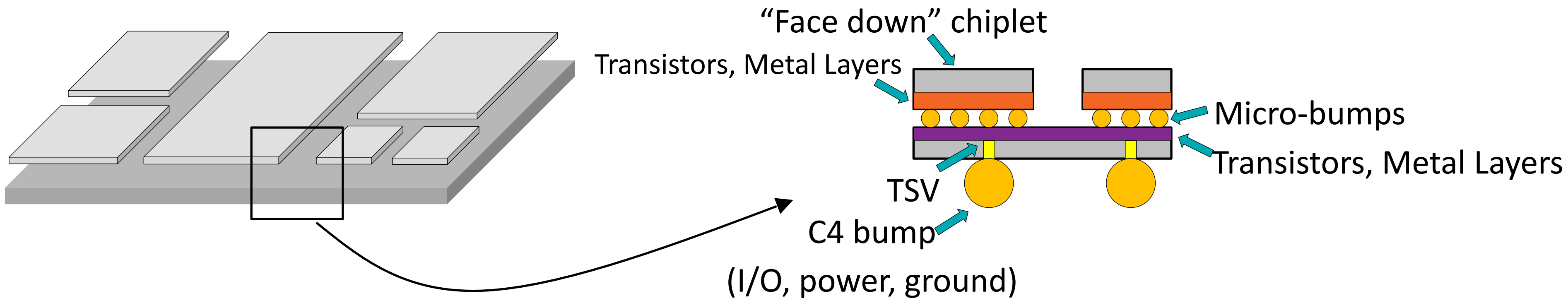


How to integrate?

Multi-chip modules (MCM)

Embedded Multi-Chip Interconnect Bridge (EMIB)

Active Silicon Interposer (2.5D)



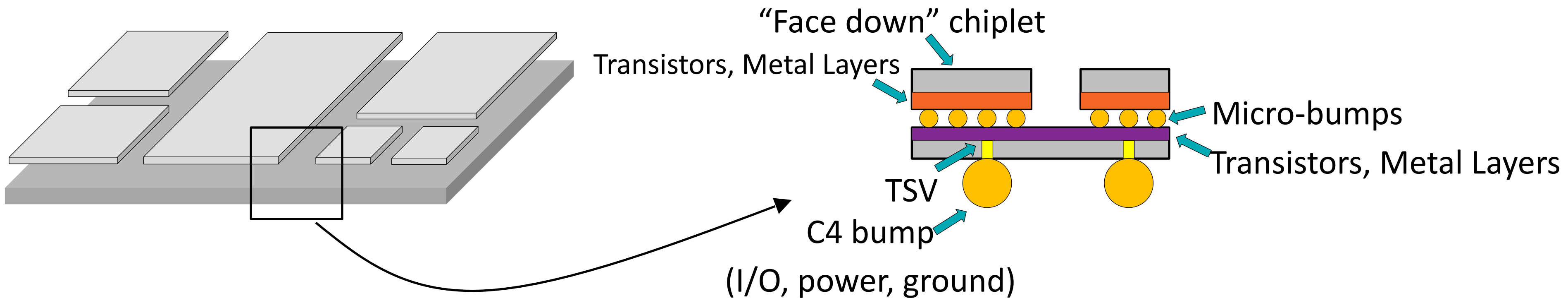
How to integrate?

Multi-chip modules (MCM)

Embedded Multi-Chip Interconnect Bridge (EMIB)

Active Silicon Interposer (2.5D)

- 😁 Simple, small chiplets



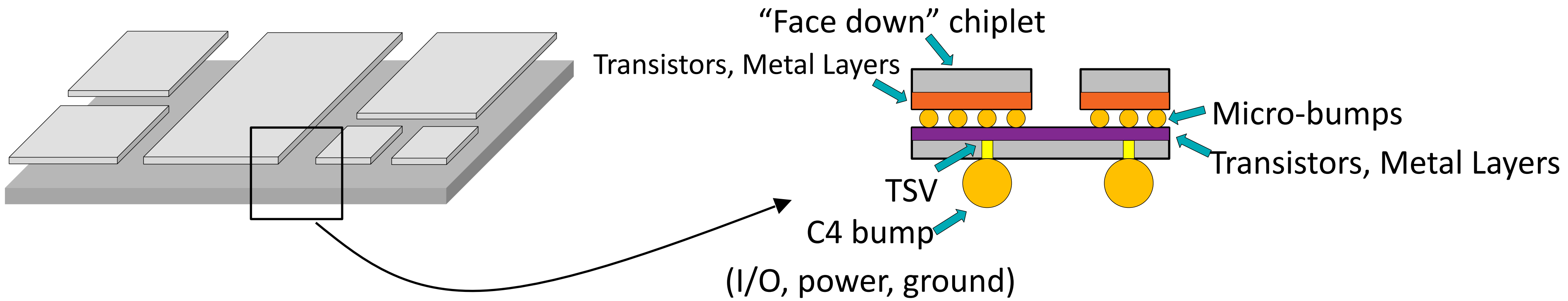
How to integrate?

Multi-chip modules (MCM)

Embedded Multi-Chip Interconnect Bridge (EMIB)

Active Silicon Interposer (2.5D)

- 😁 Simple, small chiplets
- 😁 Move SoC functionality into interposer



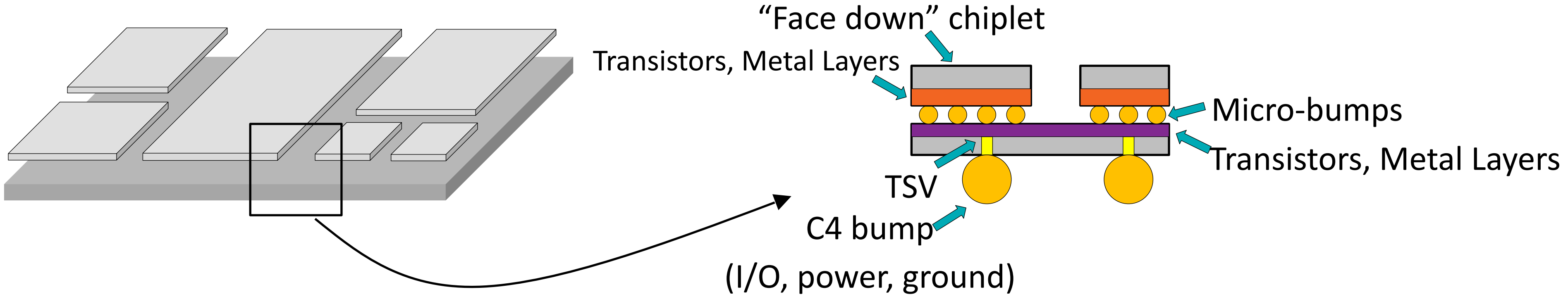
How to integrate?

Multi-chip modules (MCM)

Embedded Multi-Chip Interconnect Bridge (EMIB)

Active Silicon Interposer (2.5D)

- 😁 Simple, small chiplets
- 😁 Move SoC functionality into interposer
- 😁 Implement in older technology node



Interposers: An enabling integration technology

Interposers: An enabling integration technology

Facilitates modular SoC design

Interposers: An enabling integration technology

Facilitates modular SoC design

But what about…

Interposers: An enabling integration technology

Facilitates modular SoC design

But what about…

- Cost — Aren't interposers expensive?

Interposers: An enabling integration technology

Facilitates modular SoC design

But what about…

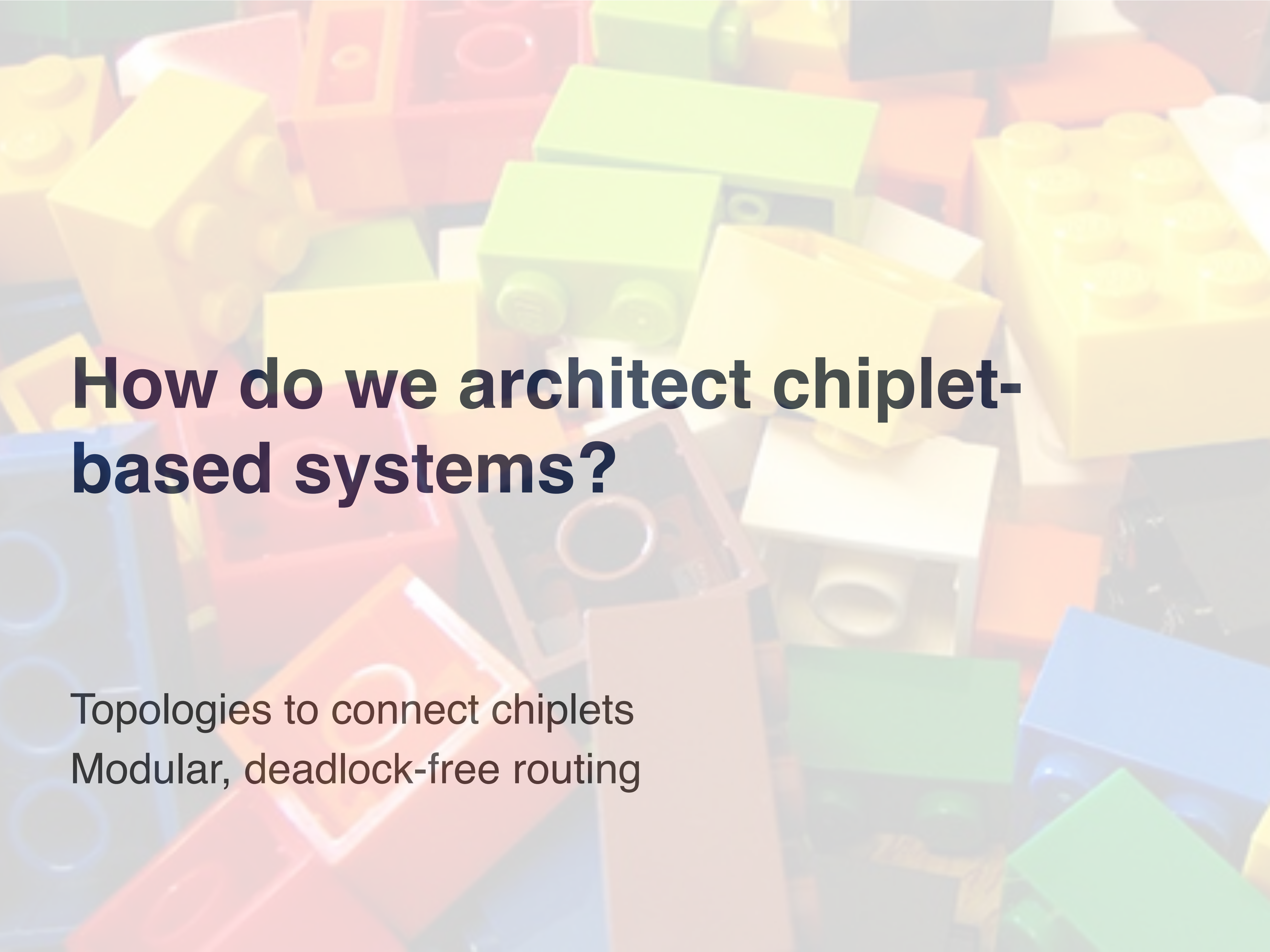
- Cost — Aren’t interposers expensive?
- Communication — How do we reintegrate?

Interposers: An enabling integration technology

Facilitates modular SoC design

But what about…

- Cost — Aren’t interposers expensive?
- Communication — How do we reintegrate?
- How to maximize modularity while reintegrating?

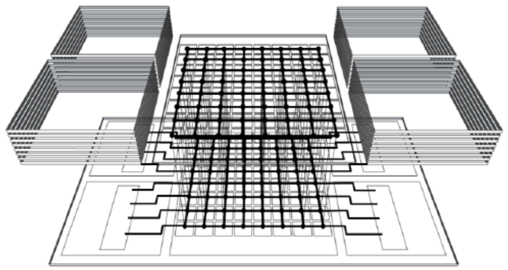


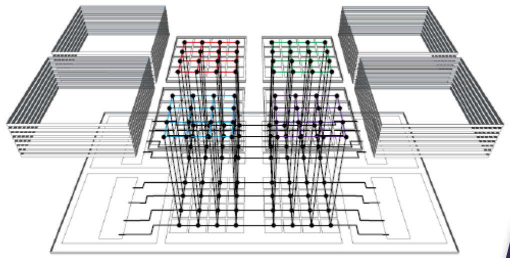
How do we architect chiplet-based systems?

Topologies to connect chiplets

Modular, deadlock-free routing

Network-on-Chip on Interposer to Reintegrate

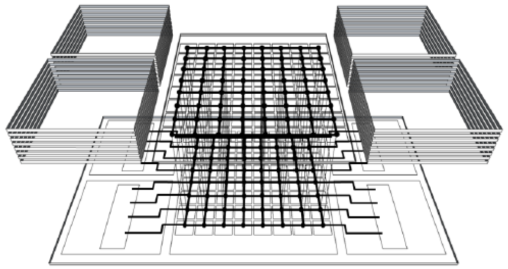


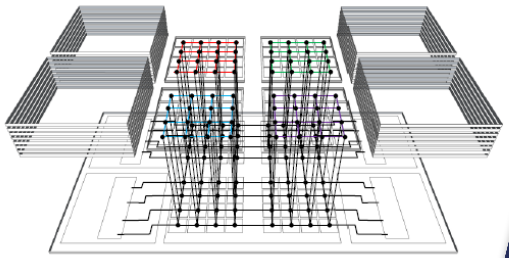


Q1: How do you build a NoC on the interposer?

Q2: What type of NoC should you build?

Network-on-Chip on Interposer to Reintegrate

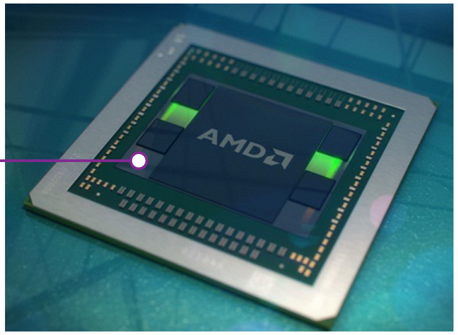




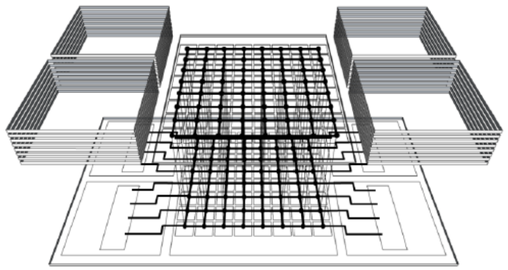
Q1: How do you build a NoC on the interposer?

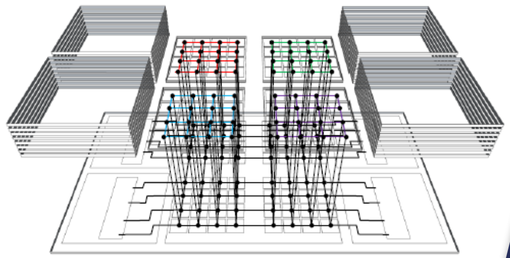
Q2: What type of NoC should you build?

Q1: … current interposers are passive!



Network-on-Chip on Interposer to Reintegrate

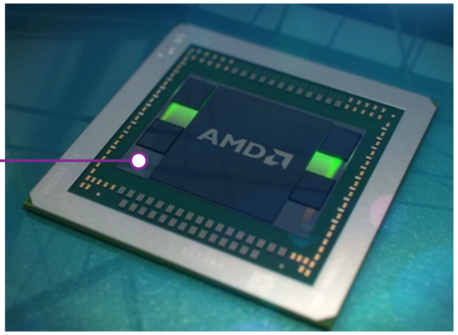




Q1: How do you build a NoC on the interposer?

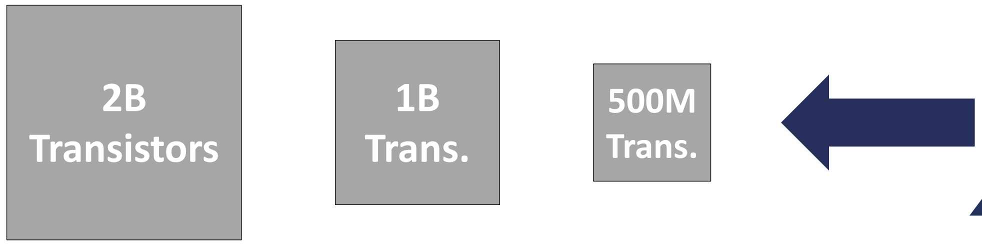
Q2: What type of NoC should you build?

Q1: … current interposers are passive!



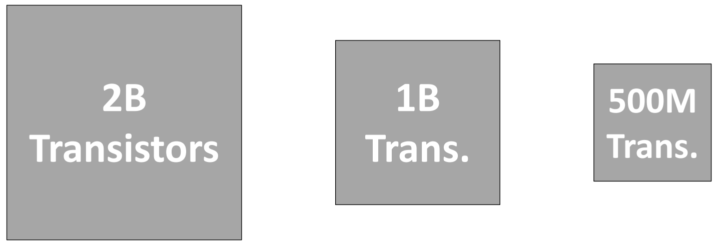
An active interposer is a huge chip, which should have horrible yield, no?!?

Minimally Active Interposers

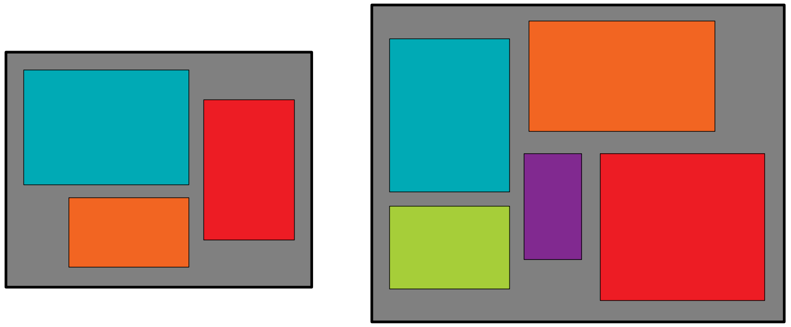


Conventional chips:
Fewer transistors → smaller chip

Minimally Active Interposers

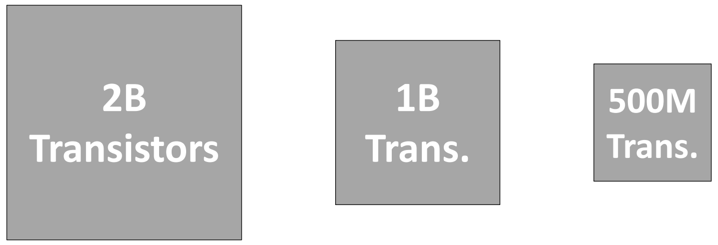


Conventional chips:
Fewer transistors → smaller chip

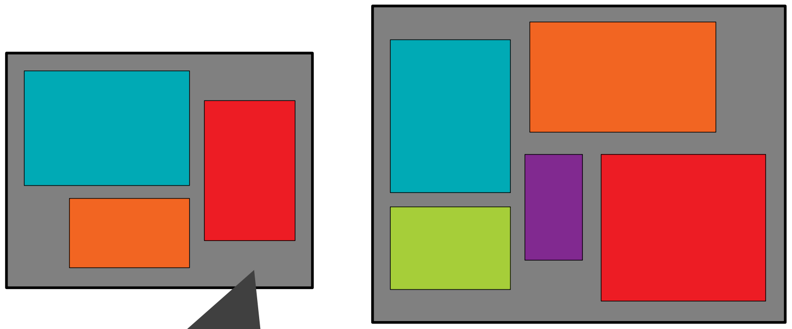


Interposer size depends on chips stacked on top of it

Minimally Active Interposers



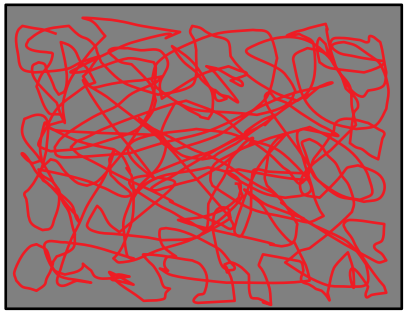
Conventional chips:
Fewer transistors → smaller chip

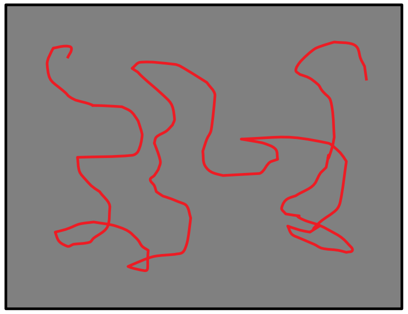


Interposer size depends on chips stacked on top of it

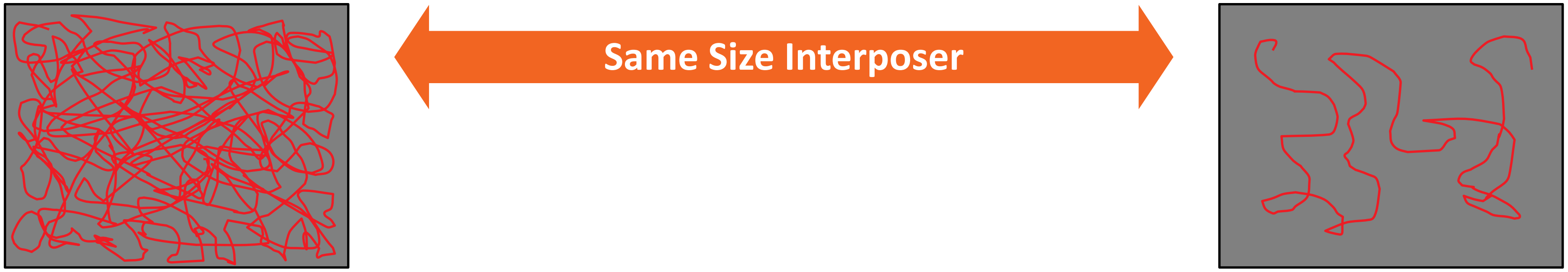
Zero or billion transistors, interposer size is the same

Minimally Active Interposers

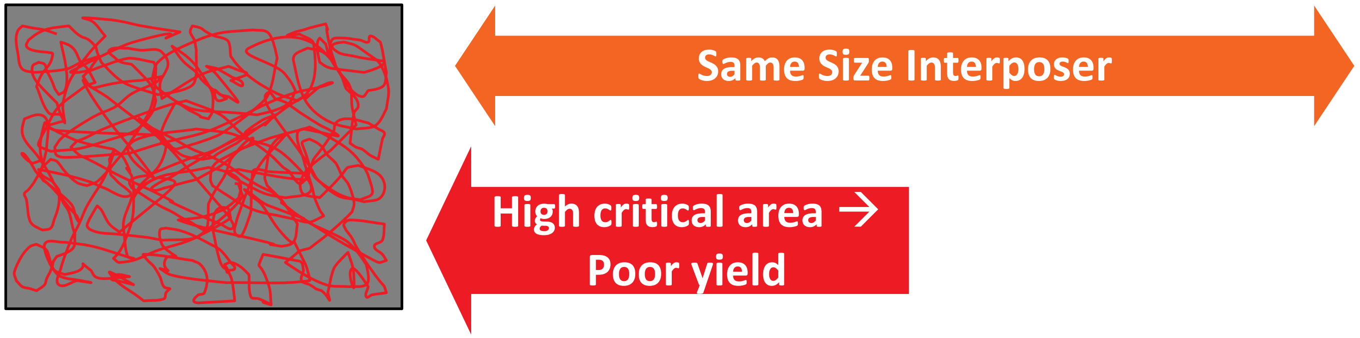




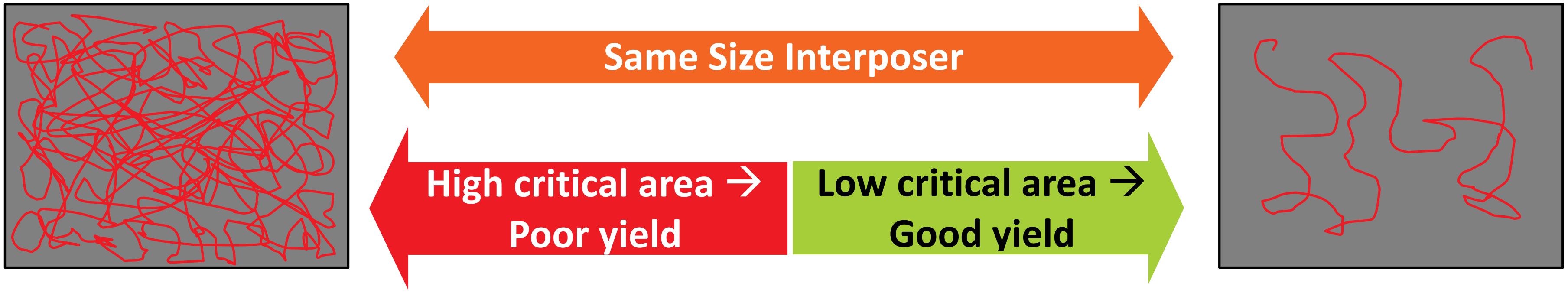
Minimally Active Interposers



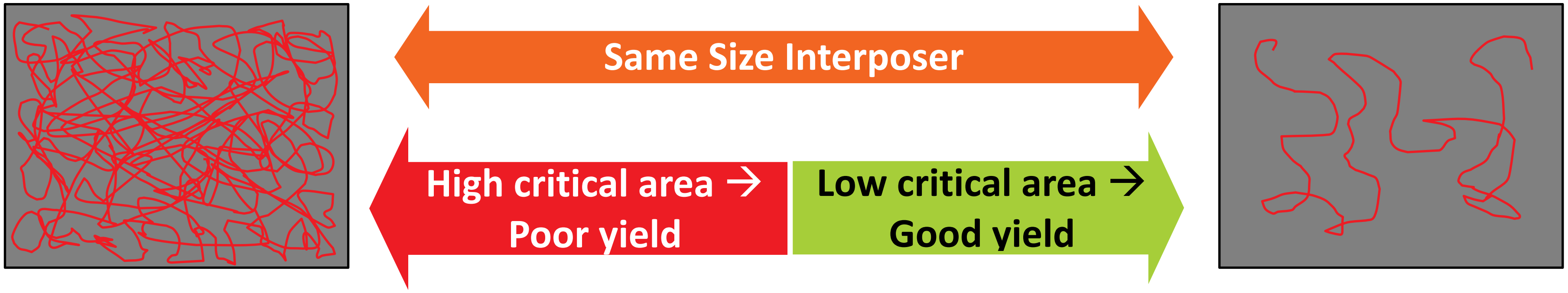
Minimally Active Interposers



Minimally Active Interposers

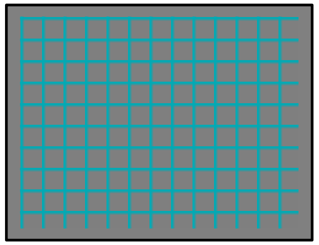

Same Size Interposer
High critical area → Poor yield
Low critical area → Good yield

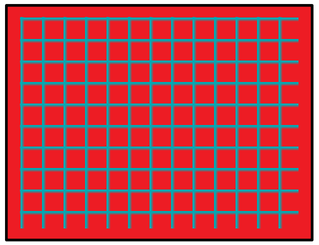
Minimally Active Interposers


Same Size Interposer
High critical area → Poor yield
Low critical area → Good yield

Chip yield impacted by defects in ***critical areas*** (e.g., contaminant in white space is fine)

Minimally Active Interposers for Large SoCs

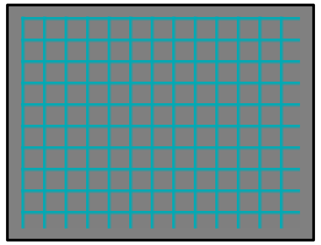


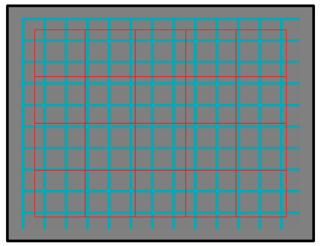


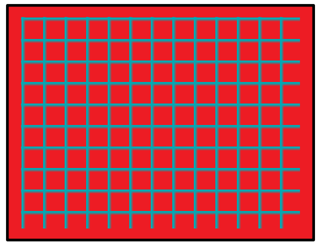
	24mm x 36mm interposer		
Defect Density	**Low**	**Medium**	**High**
Passive Interposer	98.5%	95.5%	92.7%
Active Interposer 1%	98.4%	95.4%	92.5%
Active Interposer 10%	98.0%	94.2%	90.7%
Fully-Active Interposer	87.2%	68.5%	55.6%

*Modelled, not real yield rates

Minimally Active Interposers for Large SoCs



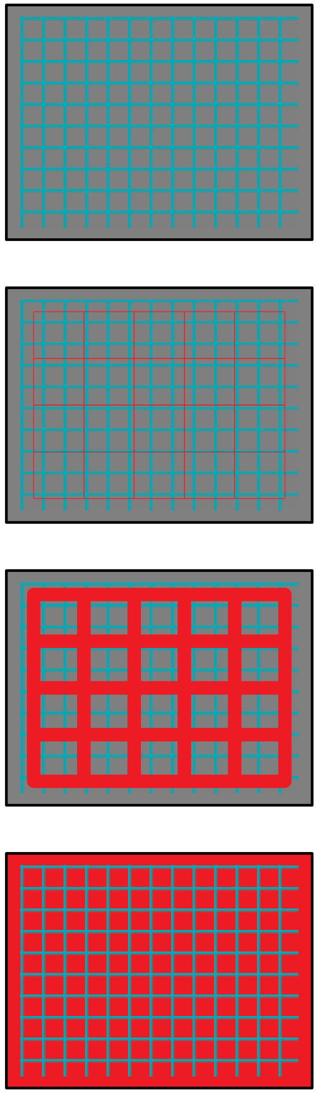




	24mm x 36mm interposer		
Defect Density	**Low**	**Medium**	**High**
Passive Interposer	98.5%	95.5%	92.7%
Active Interposer 1%	98.4%	95.4%	92.5%
Active Interposer 10%	98.0%	94.2%	90.7%
Fully-Active Interposer	87.2%	68.5%	55.6%

*Modelled, not real yield rates

Minimally Active Interposers for Large SoCs

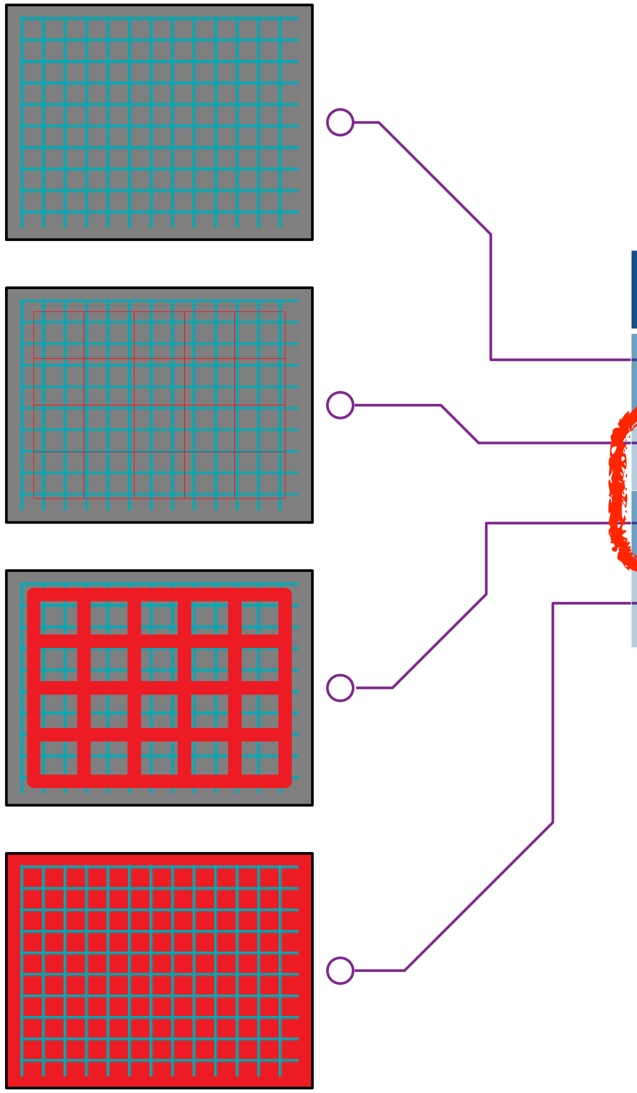


24mm x 36mm interposer

Defect Density	Low	Medium	High
Passive Interposer	98.5%	95.5%	92.7%
Active Interposer 1%	98.4%	95.4%	92.5%
Active Interposer 10%	98.0%	94.2%	90.7%
Fully-Active Interposer	87.2%	68.5%	55.6%

*Modelled, not real yield rates

Minimally Active Interposers for Large SoCs



	24mm x 36mm interposer		
Defect Density	**Low**	**Medium**	**High**
Passive Interposer	98.5%	95.5%	92.7%
Active Interposer 1%	98.4%	95.4%	92.5%
Active Interposer 10%	98.0%	94.2%	90.7%
Fully-Active Interposer	87.2%	68.5%	55.6%

*Modelled, not real yield rates

Minimally Active Interposers for Large SoCs

Minimally Active Interposers for Large SoCs

Active interposer is not free…

Minimally Active Interposers for Large SoCs

Active interposer is not free…

… But appears practical if used judiciously

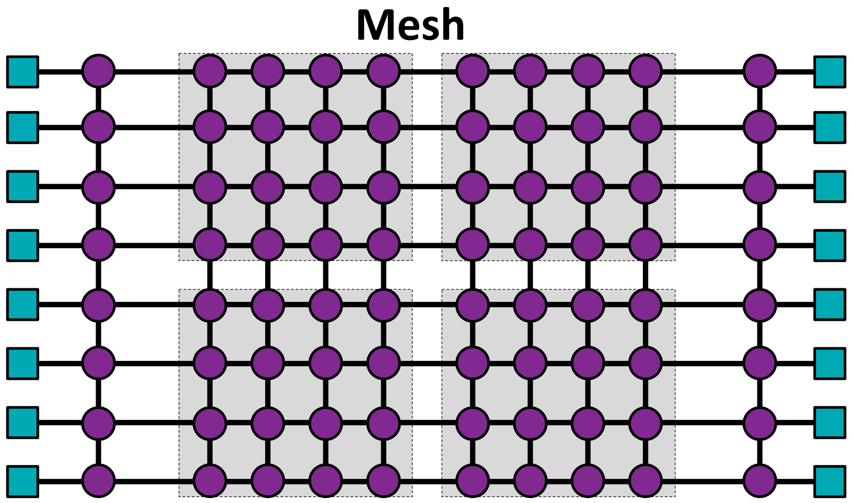
Minimally Active Interposers for Large SoCs

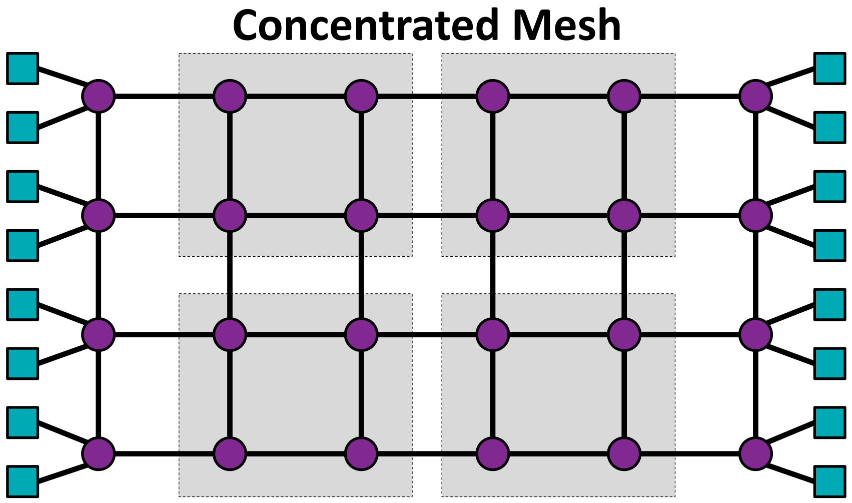
Active interposer is not free…

… But appears practical if used judiciously

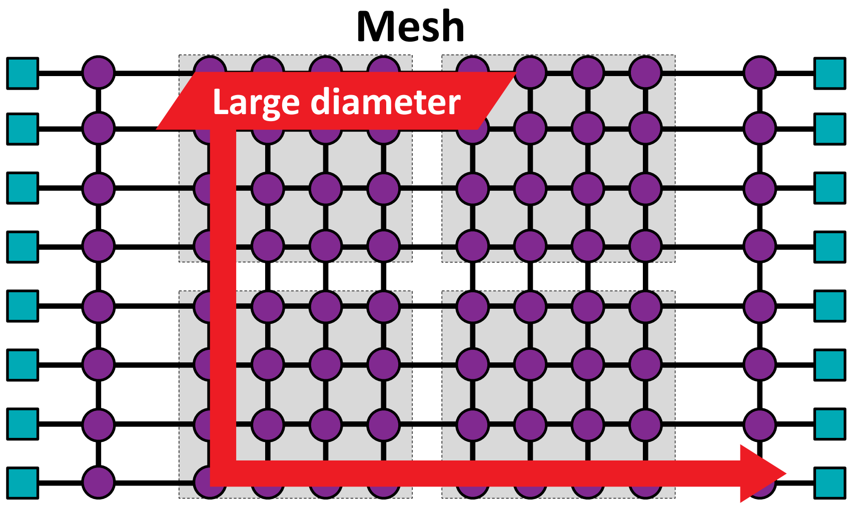
So how should we design our NoC on interposer?

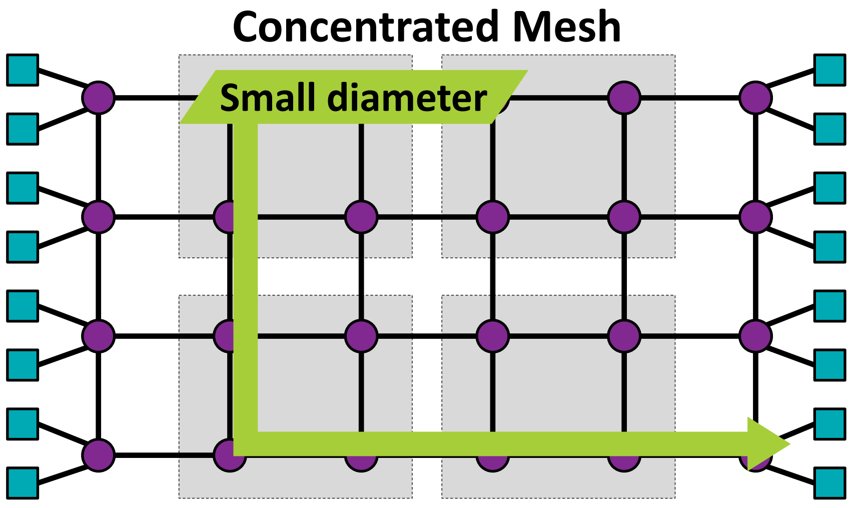
NoC Goals: Diameter vs Bisection Bandwidth



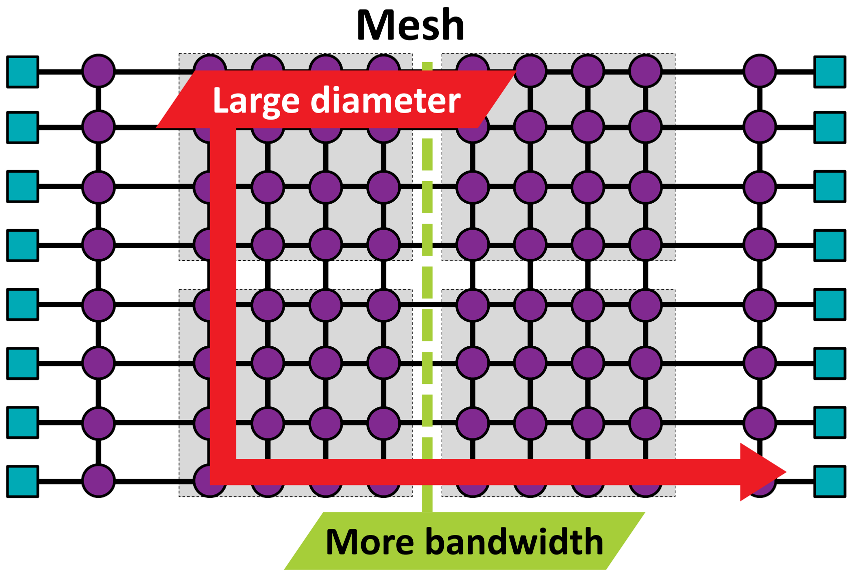


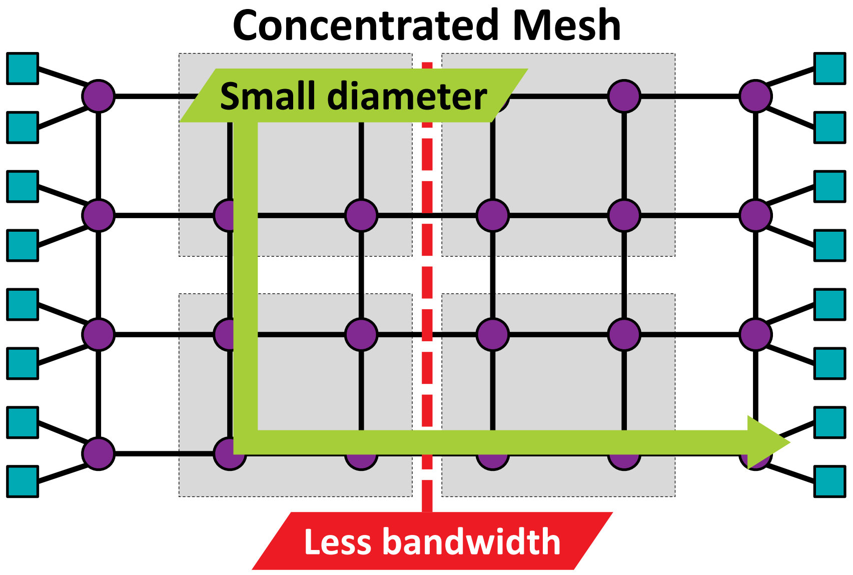
NoC Goals: Diameter vs Bisection Bandwidth



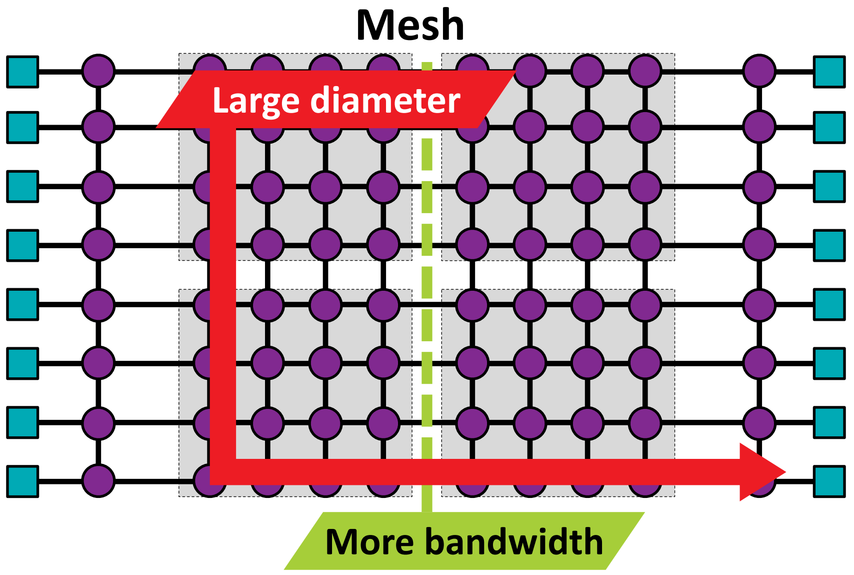


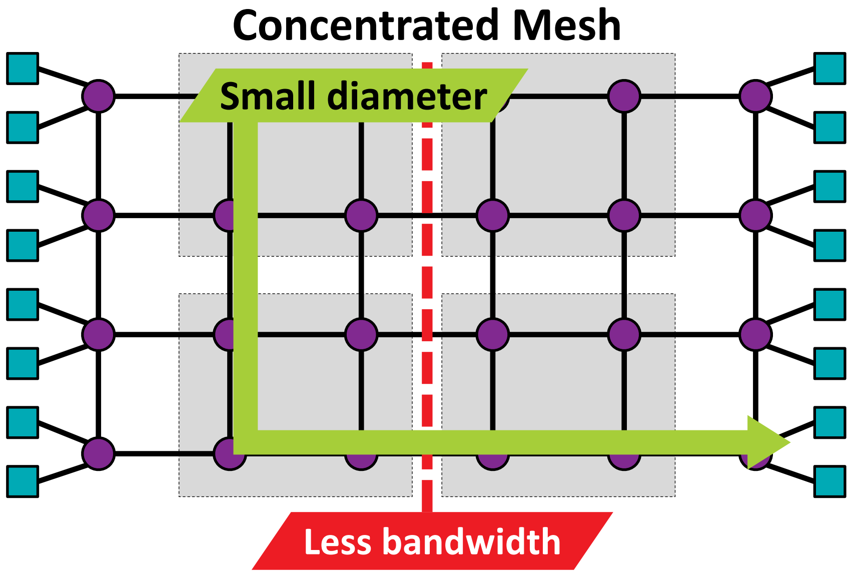
NoC Goals: Diameter vs Bisection Bandwidth

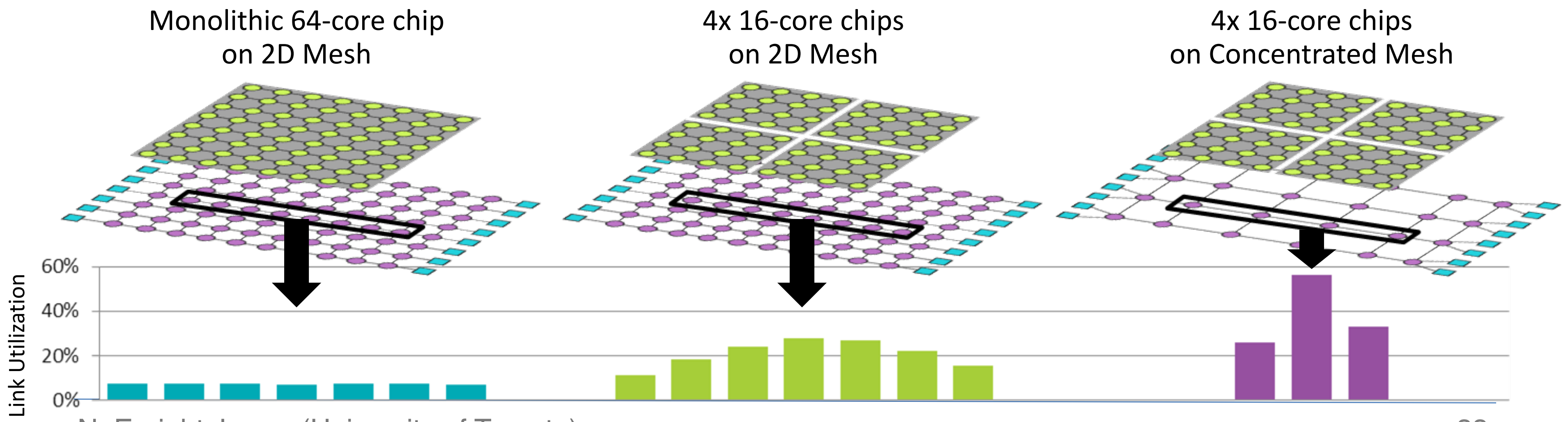




NoC Goals: Diameter vs Bisection Bandwidth



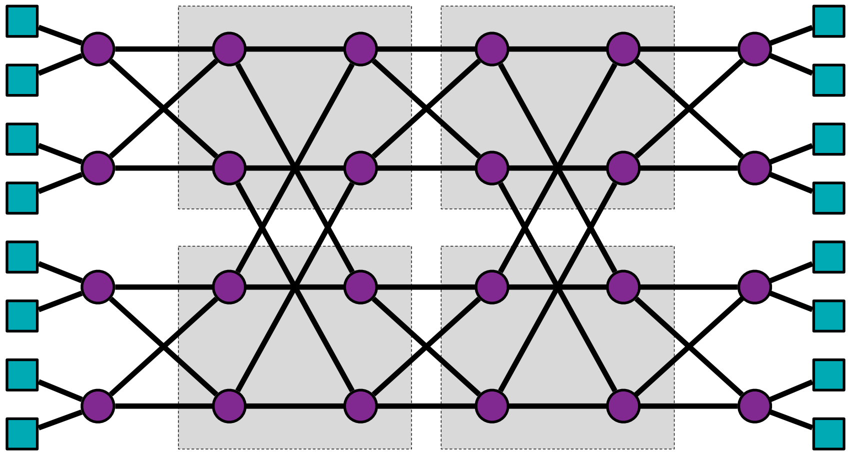




The ButterDonut

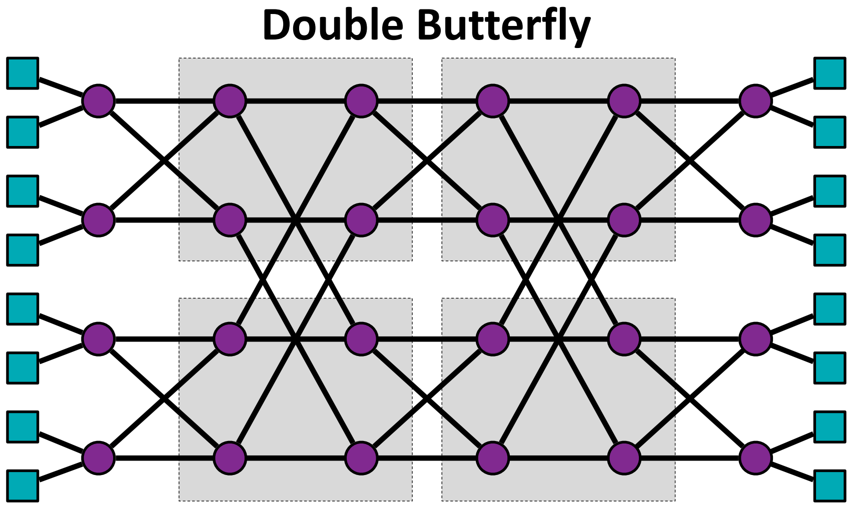
The ButterDonut

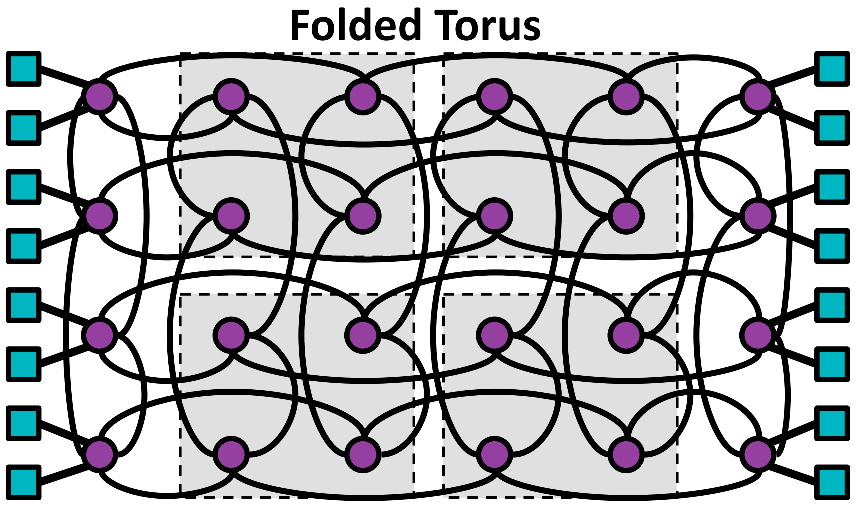
Double Butterfly



Optimized for E-W traffic (cores →memory)

The ButterDonut

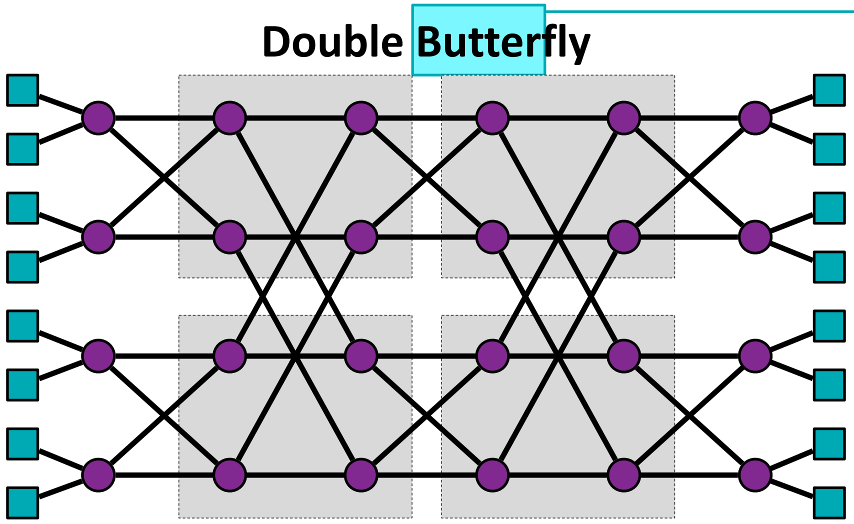




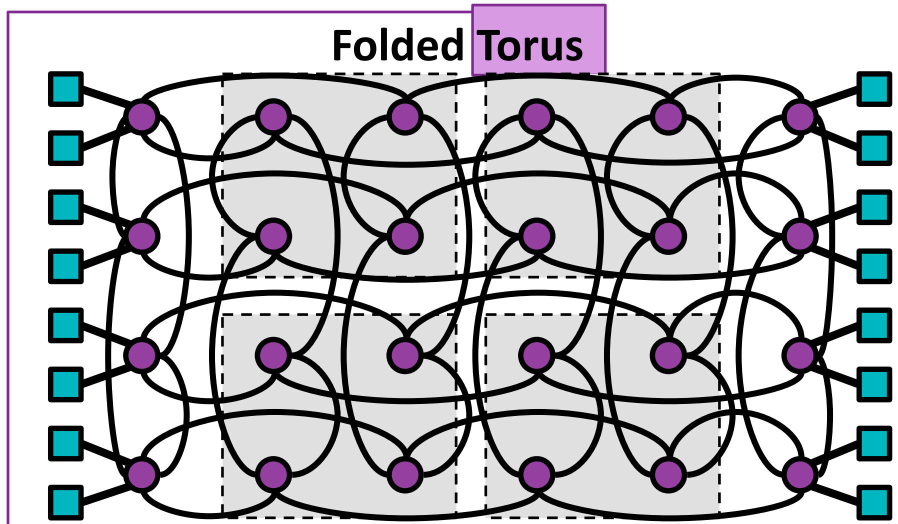
Optimized for E-W traffic (cores →memory)

Balanced, Fewer hops, +20% links

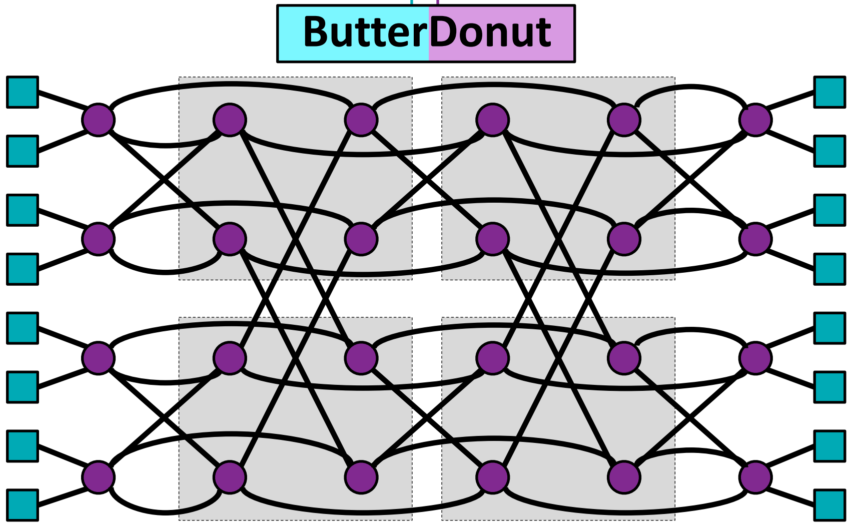
The ButterDonut



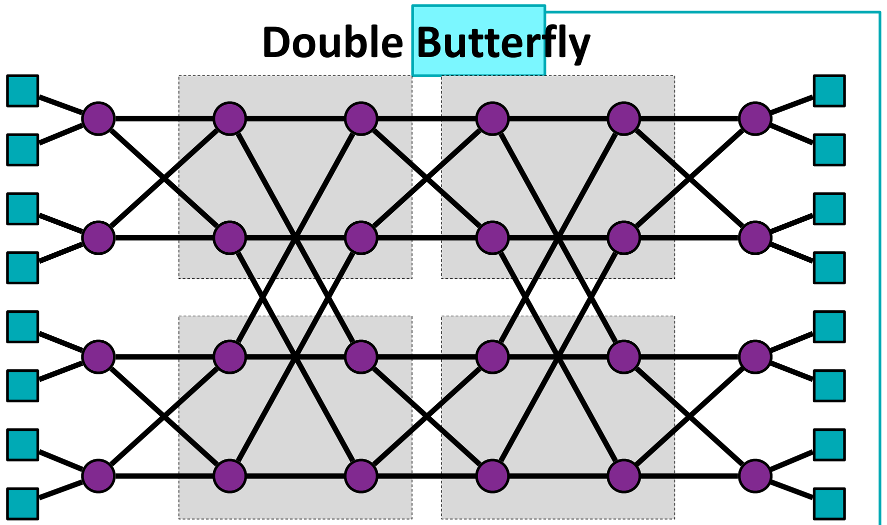
Optimized for E-W traffic (cores →memory)



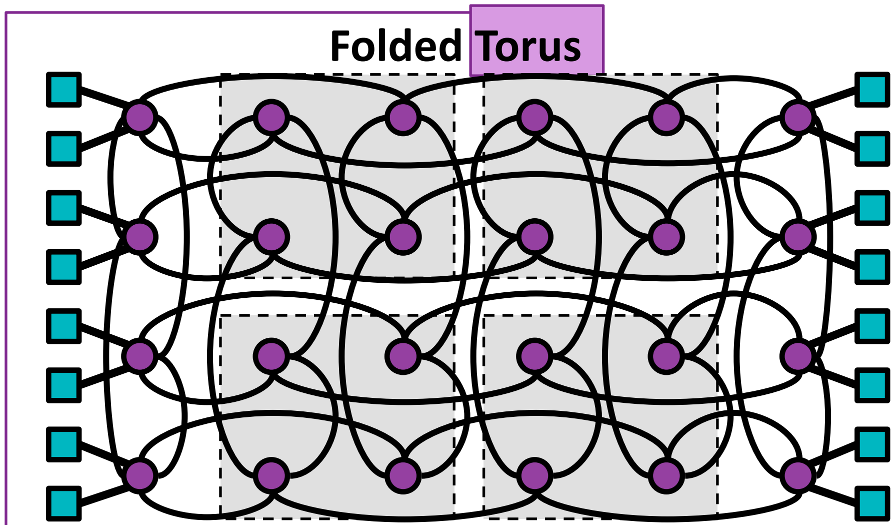
Balanced, Fewer hops, +20% links



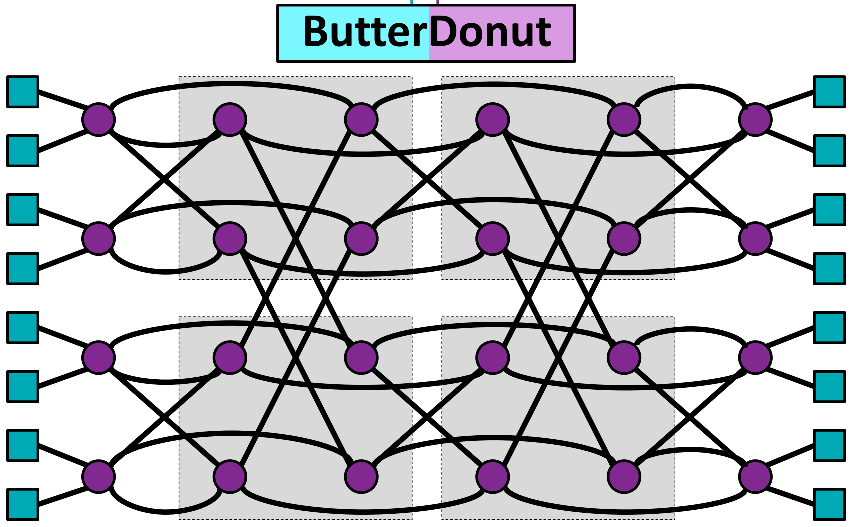
The ButterDonut



Optimized for E-W traffic (cores →memory)



Balanced, Fewer hops, +20% links



Smaller diameter

Lower Avg. Hop Count

Higher Bisection BW

+10% links vs. DButterfly

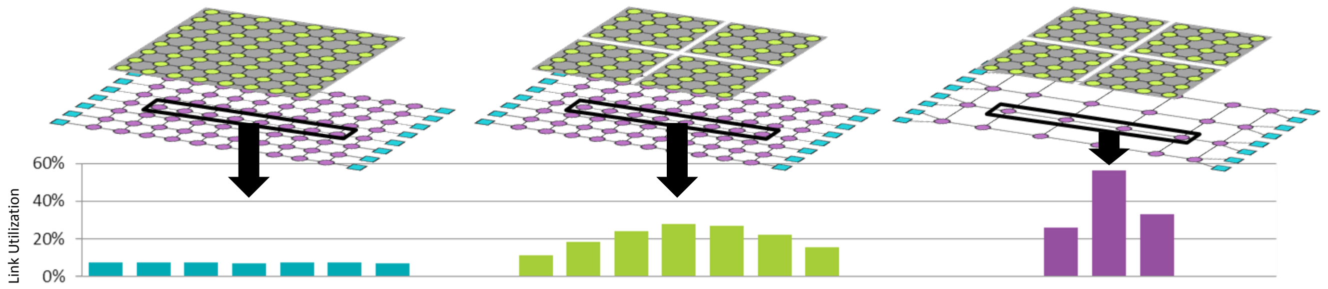
Interposer router misalignment

Interposer router misalignment

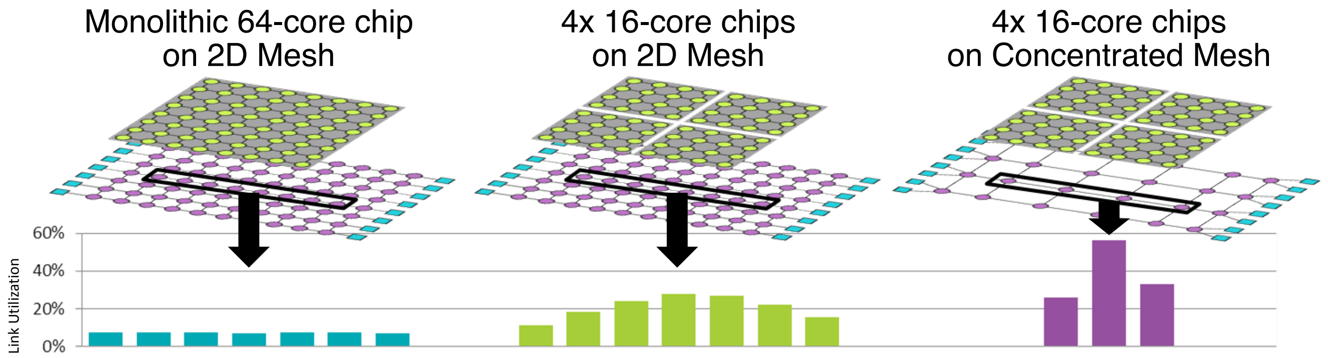
Monolithic 64-core chip on 2D Mesh

4x 16-core chips on 2D Mesh

4x 16-core chips on Concentrated Mesh

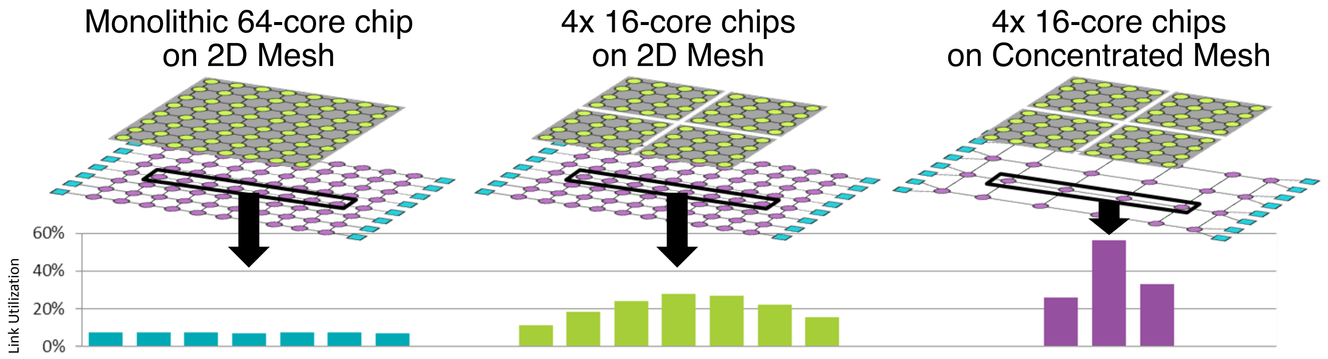


Interposer router misalignment

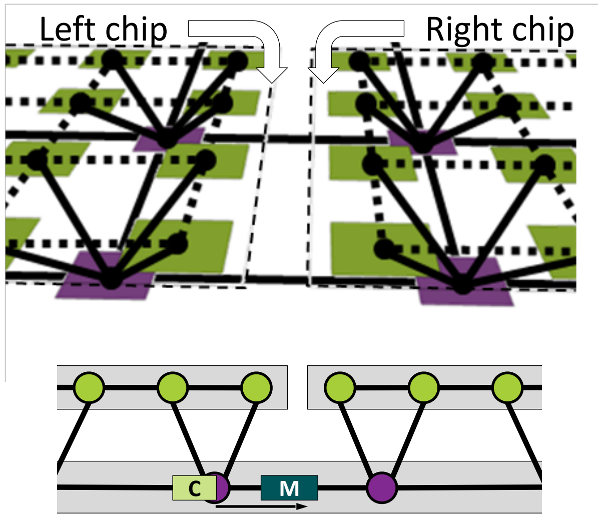


Bisection links are the primary bottleneck

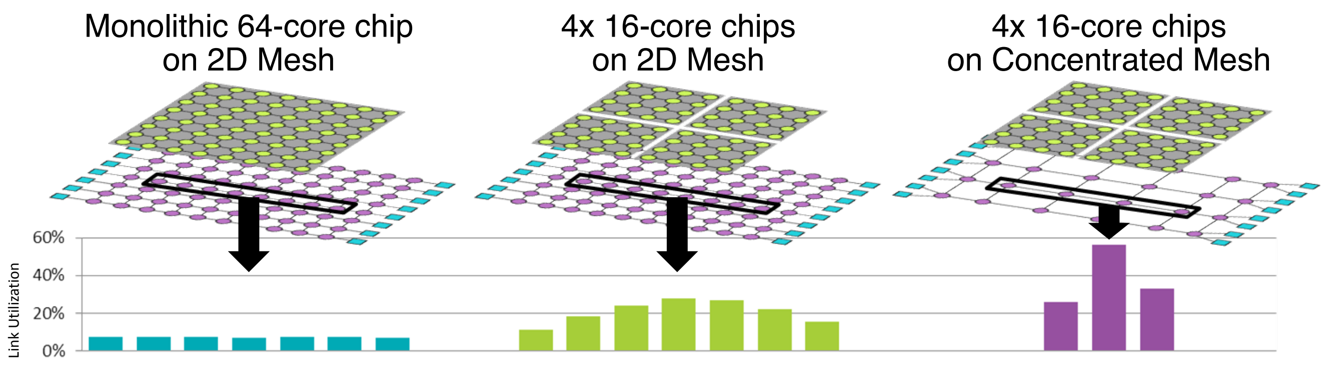
Interposer router misalignment



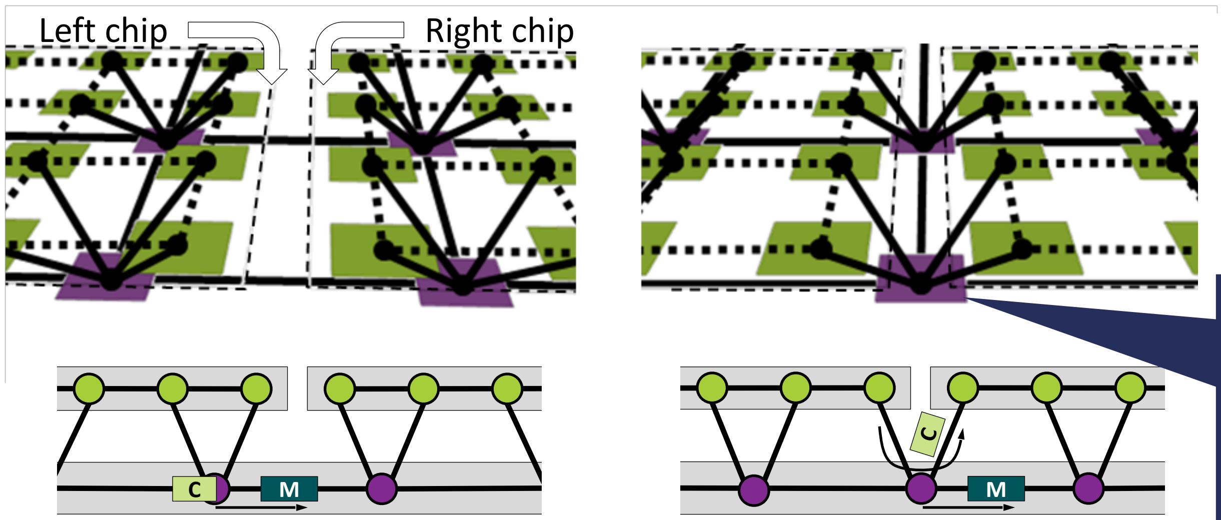
Bisection links are the primary bottleneck



Interposer router misalignment



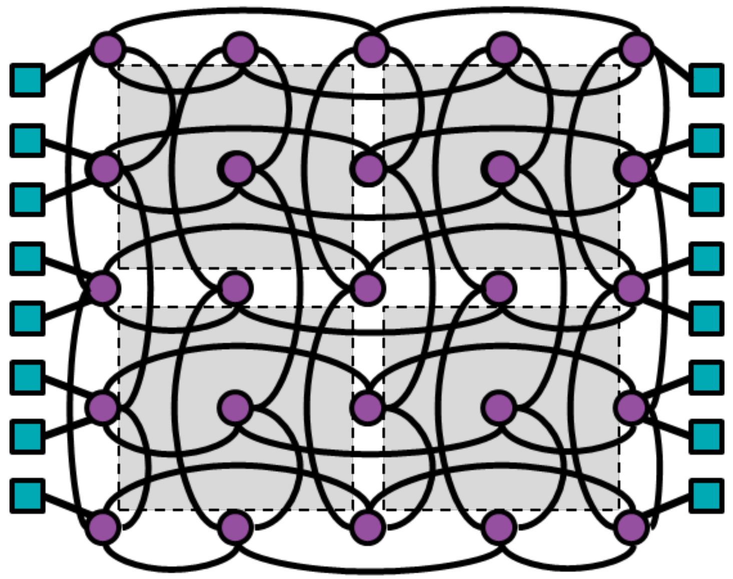
Bisection links are the primary bottleneck



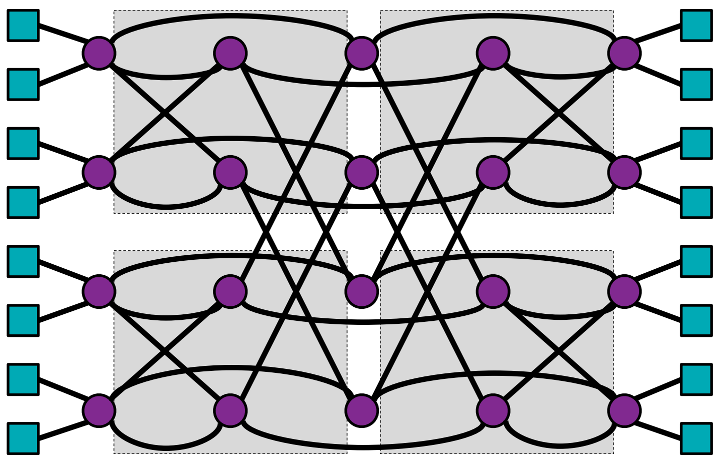
“Misaligned” router placed in between chips

Misaligned Topologies

Folded Torus(X+Y)

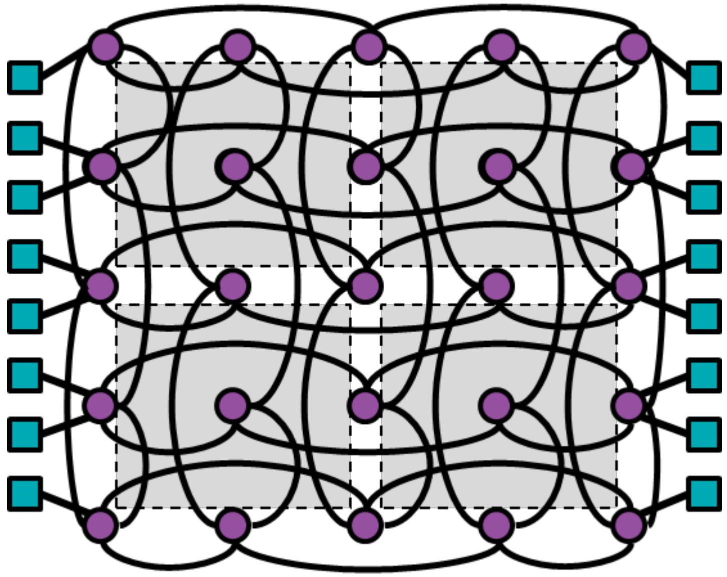


Misaligned ButterDonut(X)

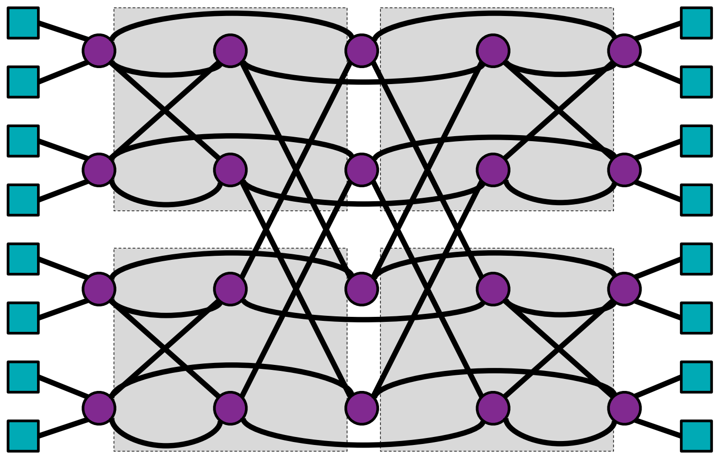


Misaligned Topologies

Folded Torus(X+Y)



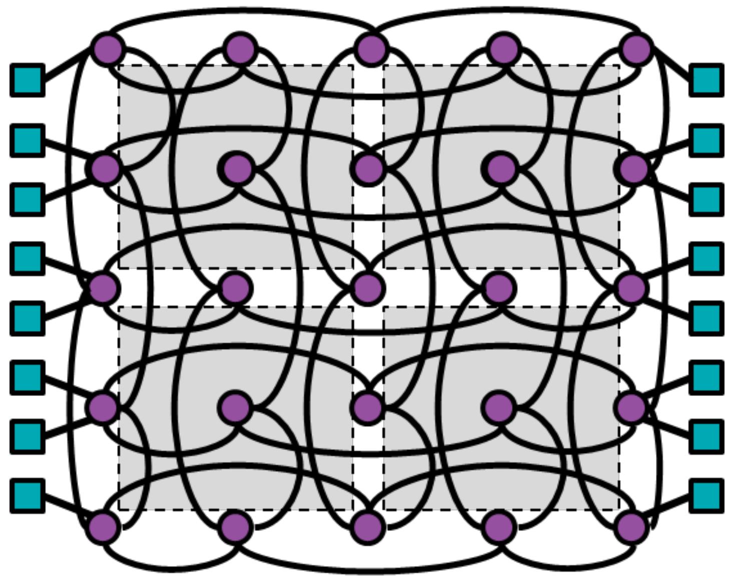
Misaligned ButterDonut(X)



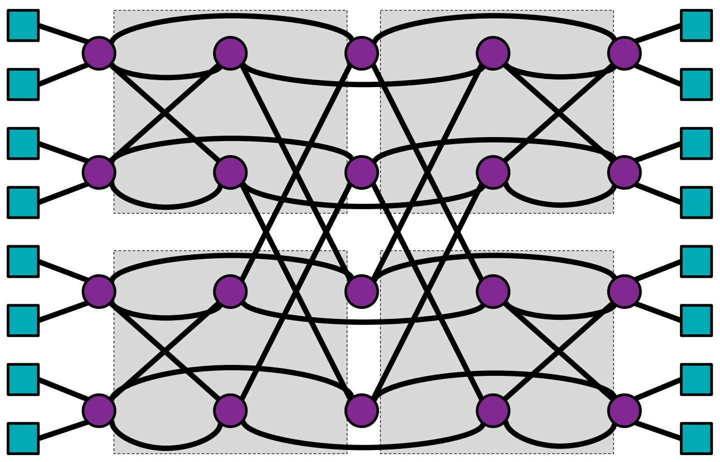
😁 Small # routers

Misaligned Topologies

Folded Torus(X+Y)



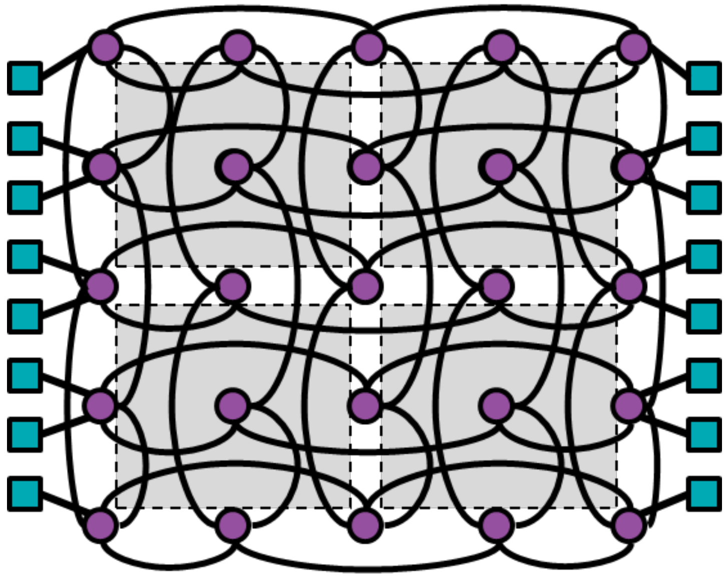
Misaligned ButterDonut(X)



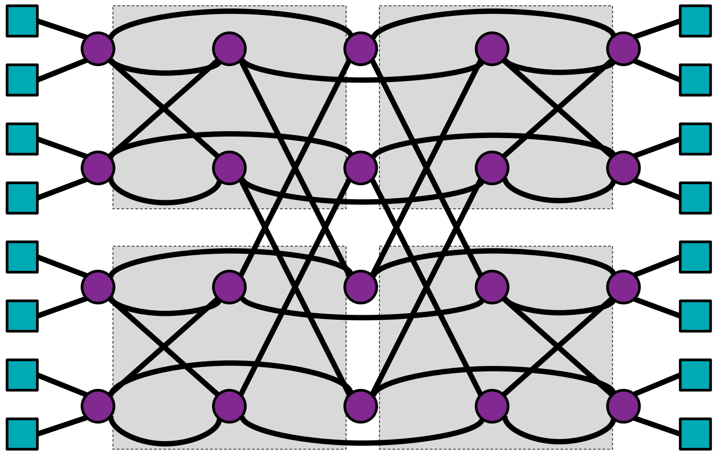
- 😁 Small # routers
- 😁 Small # links*

Misaligned Topologies

Folded Torus(X+Y)



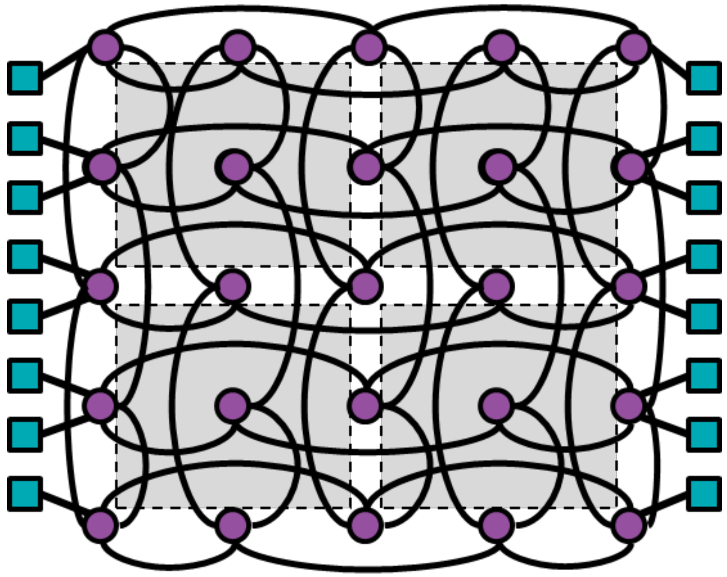
Misaligned ButterDonut(X)



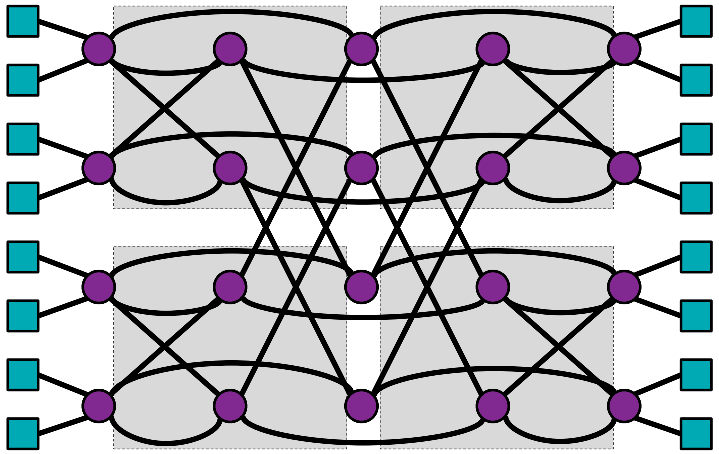
- 😁 Small # routers
- 😁 Small # links*
- 😁 Small diameter

Misaligned Topologies

Folded Torus(X+Y)



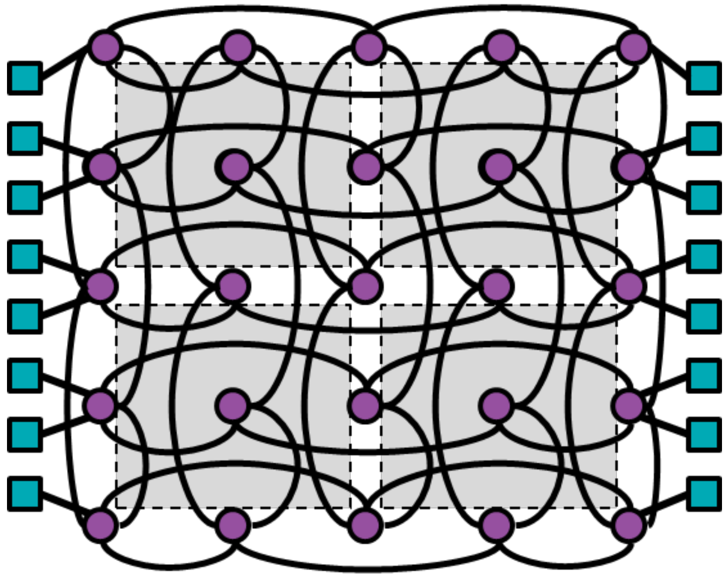
Misaligned ButterDonut(X)



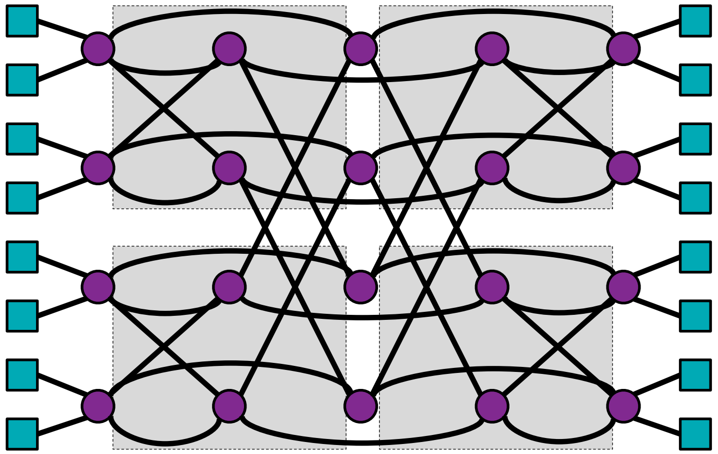
- 😁 Small # routers
- 😁 Small # links*
- 😁 Small diameter
- 😁 Low average hop count

Misaligned Topologies

Folded Torus(X+Y)

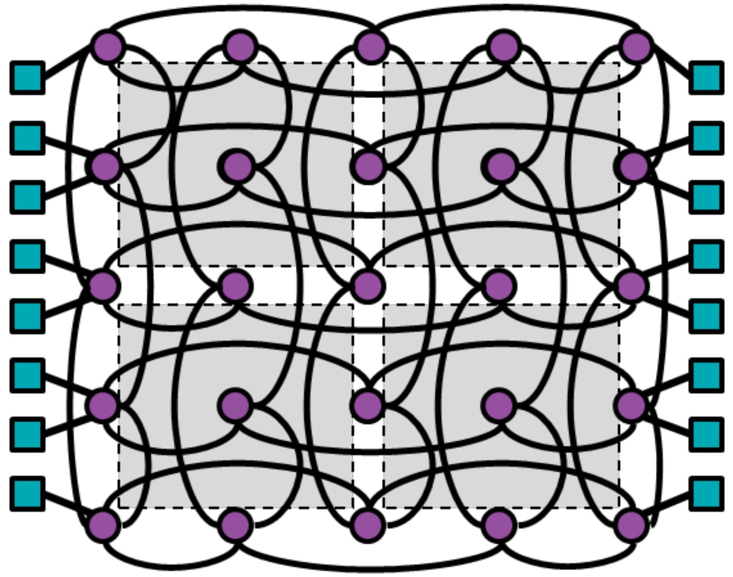


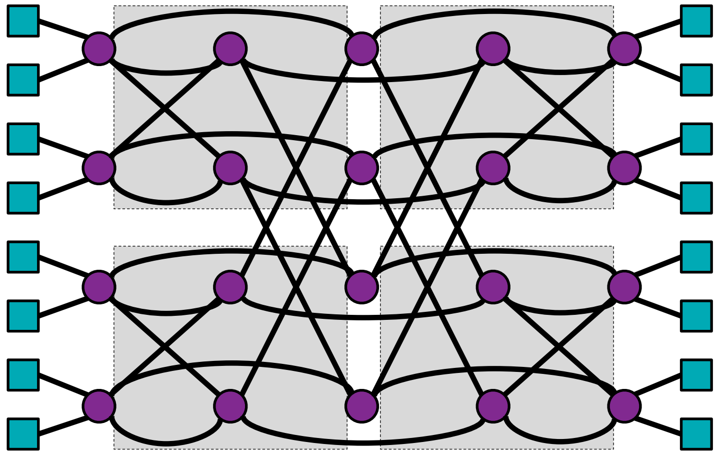
Misaligned ButterDonut(X)



- 😁 Small # routers
- 😁 Small # links*
- 😁 Small diameter
- 😁 Low average hop count
- 😁 Large bisection bandwidth

Misaligned Topologies

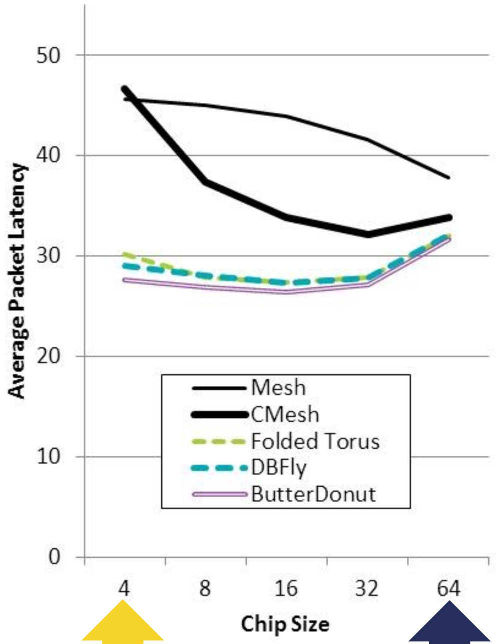




- 😁 Small # routers
- 😁 Small # links*
- 😁 Small diameter
- 😁 Low average hop count
- 😁 Large bisection bandwidth

Hybrid topology + misalignment gets you best of everything (almost)

Main takeaway: Disintegration is promising

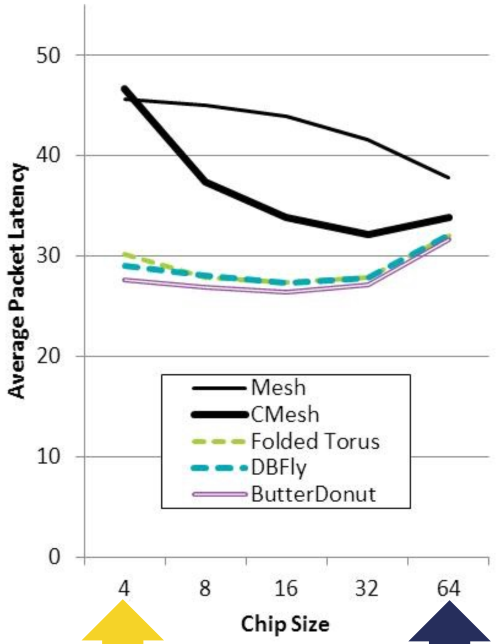


16x Quad-core chips

1x 64-core chip

Main takeaway: Disintegration is promising

Can design an interposer NoC topology to overcome disintegration-induced fragmentation of SoC

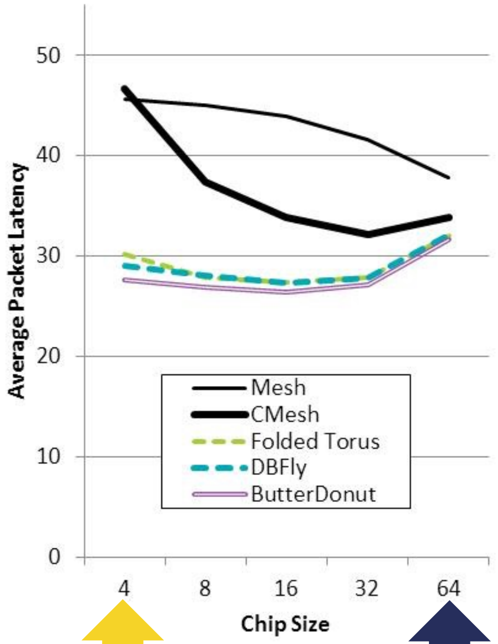


16x Quad-core chips

1x 64-core chip

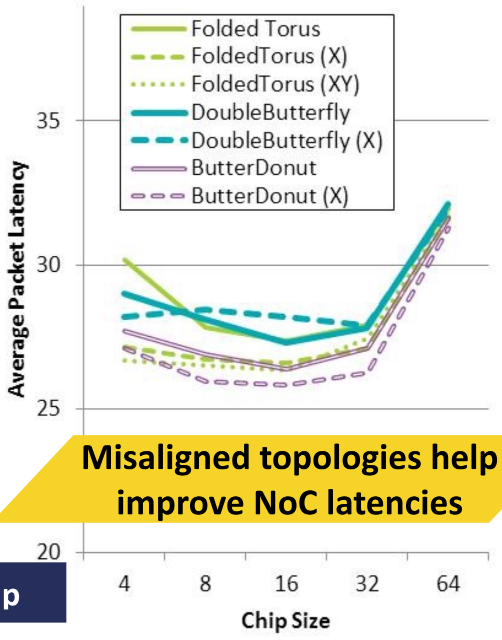
Main takeaway: Disintegration is promising

Can design an interposer NoC topology to overcome disintegration-induced fragmentation of SoC



16x Quad-core chips

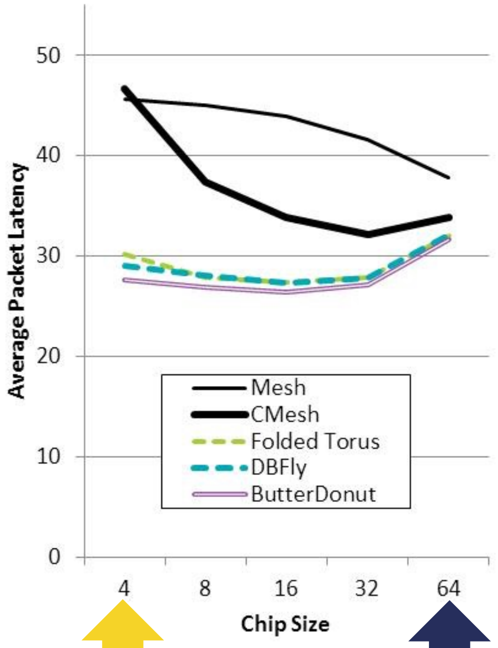
1x 64-core chip



Misaligned topologies help improve NoC latencies

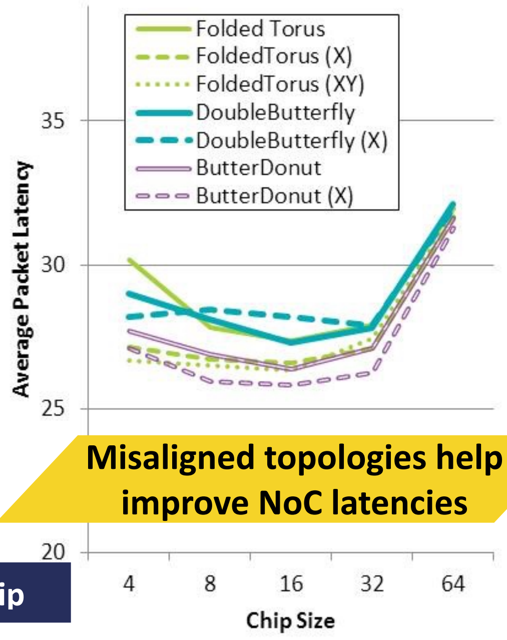
Main takeaway: Disintegration is promising

Can design an interposer NoC topology to overcome disintegration-induced fragmentation of SoC

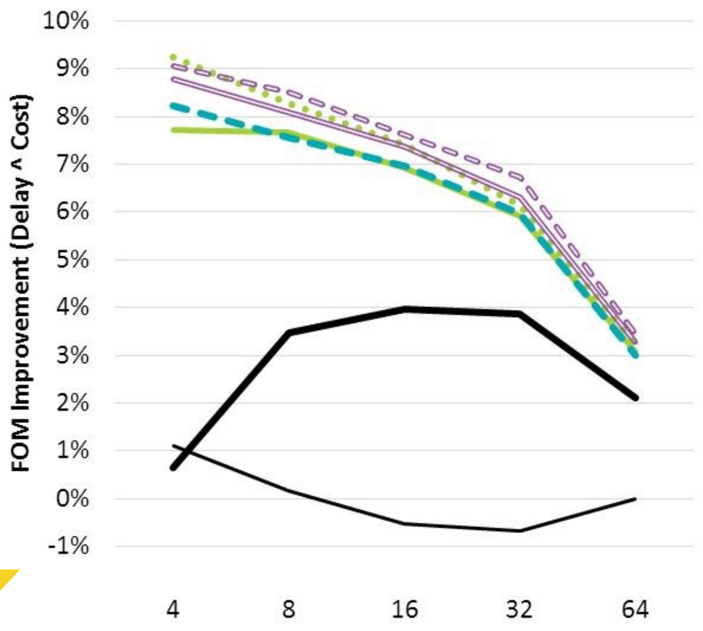


16x Quad-core chips

1x 64-core chip

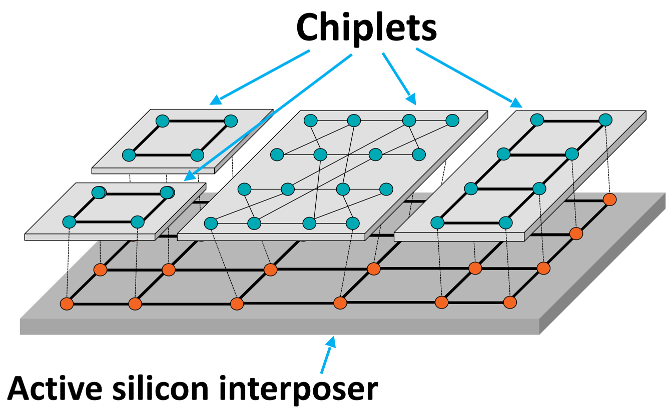


Misaligned topologies help improve NoC latencies



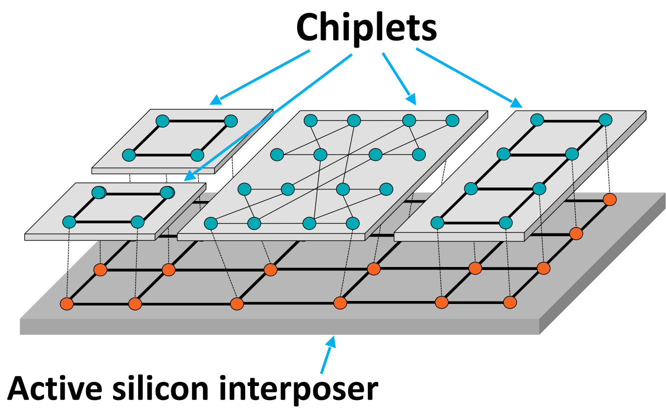
You <u>can</u> go too far…

Disintegration is promising… but how to route?



Disintegration is promising… but how to route?

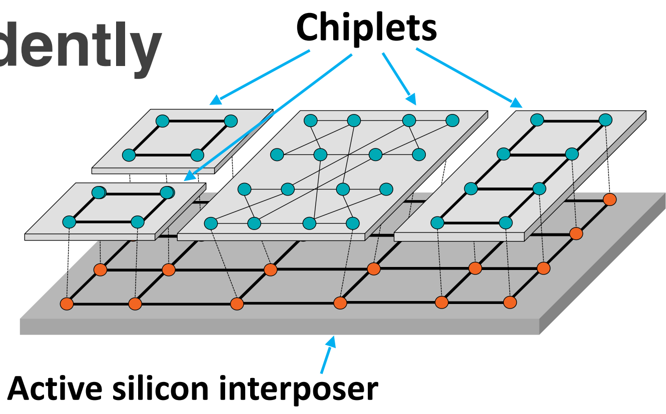
Now we have a NoC that spans both chiplets and interposer



Disintegration is promising… but how to route?

Now we have a NoC that spans both chiplets and interposer

Each chiplet may be designed independently



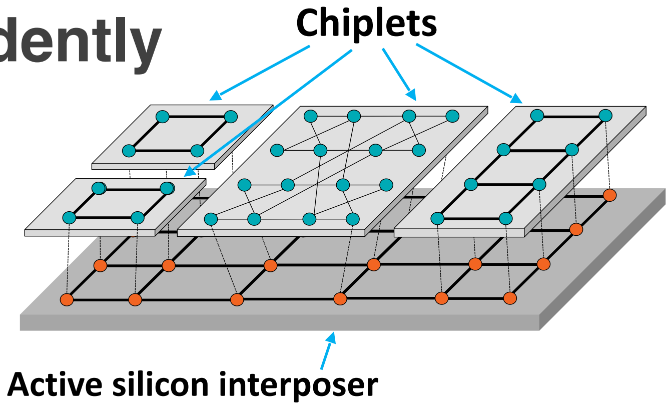
Disintegration is promising… but how to route?

Now we have a NoC that spans both chiplets and interposer

Each chiplet may be designed independently

Goals:

- Free to choose NoC topology on chiplet
- Free to choose local routing algorithm within chiplet (deadlock free)



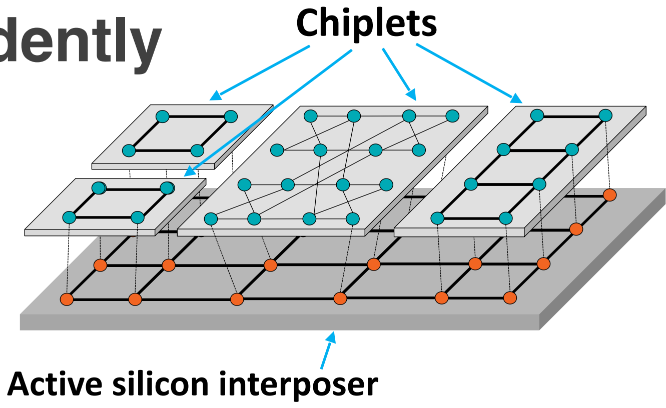
Disintegration is promising… but how to route?

Now we have a NoC that spans both chiplets and interposer

Each chiplet may be designed independently

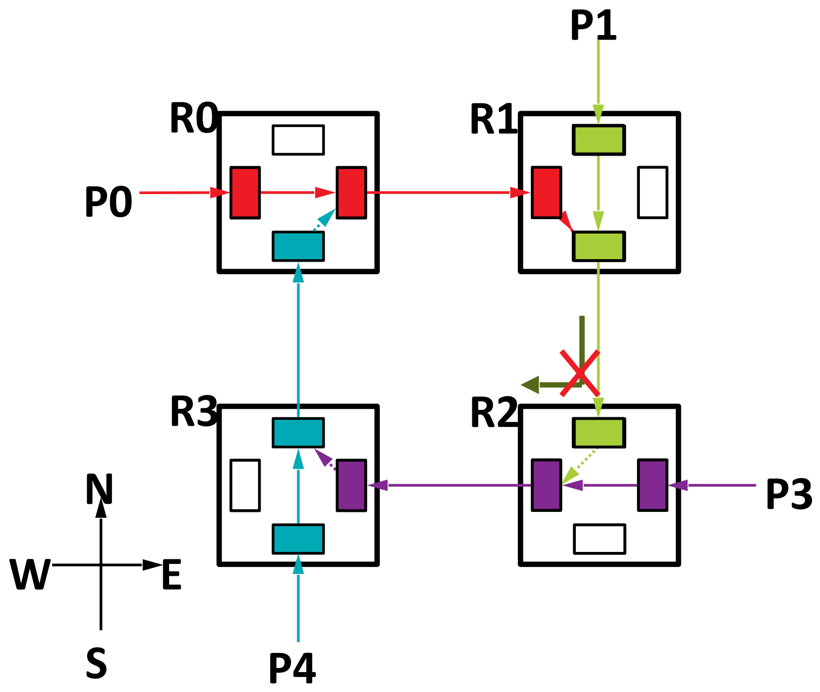
Goals:

- Free to choose NoC topology on chiplet
- Free to choose local routing algorithm within chiplet (deadlock free)



Problem: Even though NoCs for chiplets and interposer are individually deadlock free, how do you ensure the final composed system is still correct?

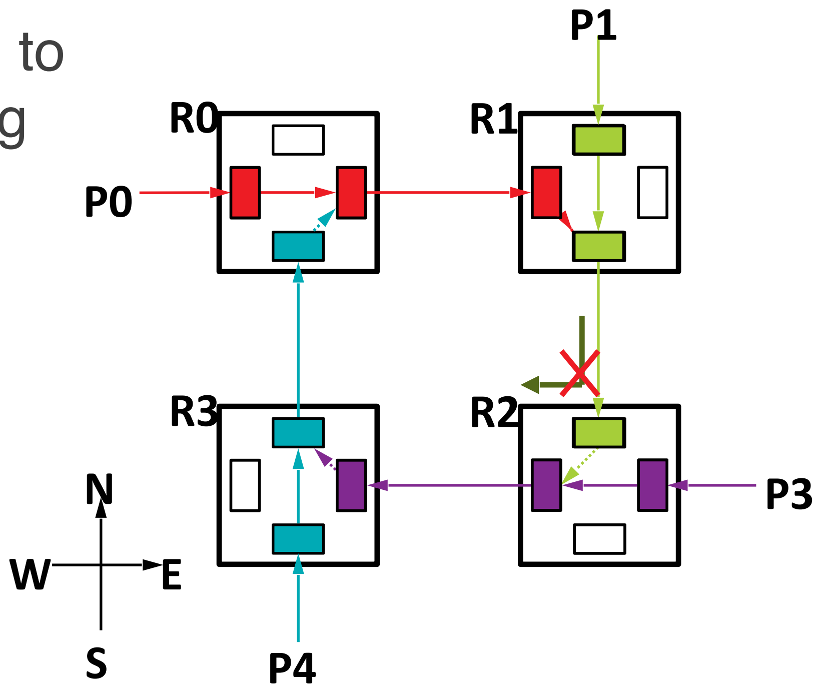
Deadlock primer



Deadlock primer

Deadlock

Can occur when packets are allowed to hold some resources while requesting others



Deadlock primer

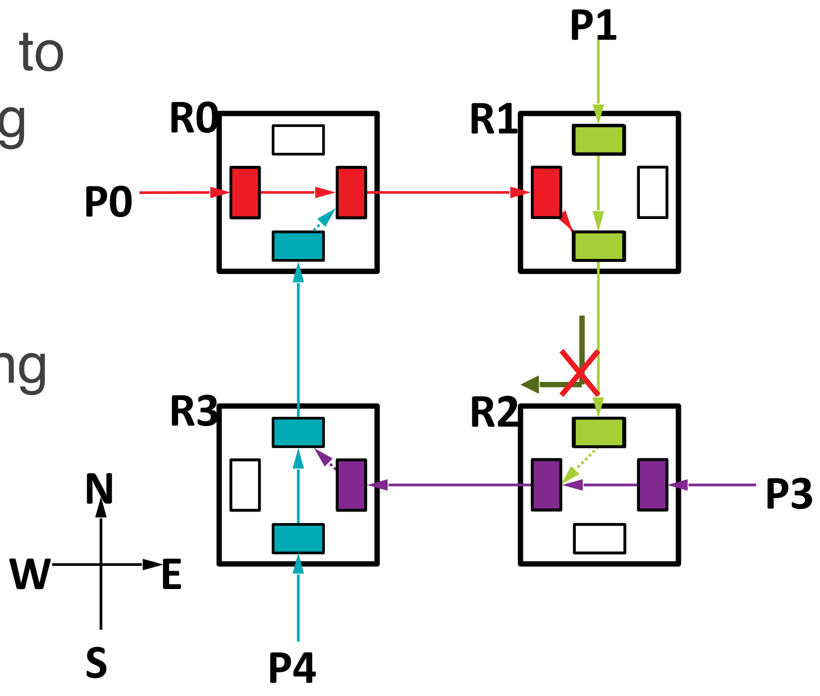
Deadlock

Can occur when packets are allowed to hold some resources while requesting others

Deadlock avoidance

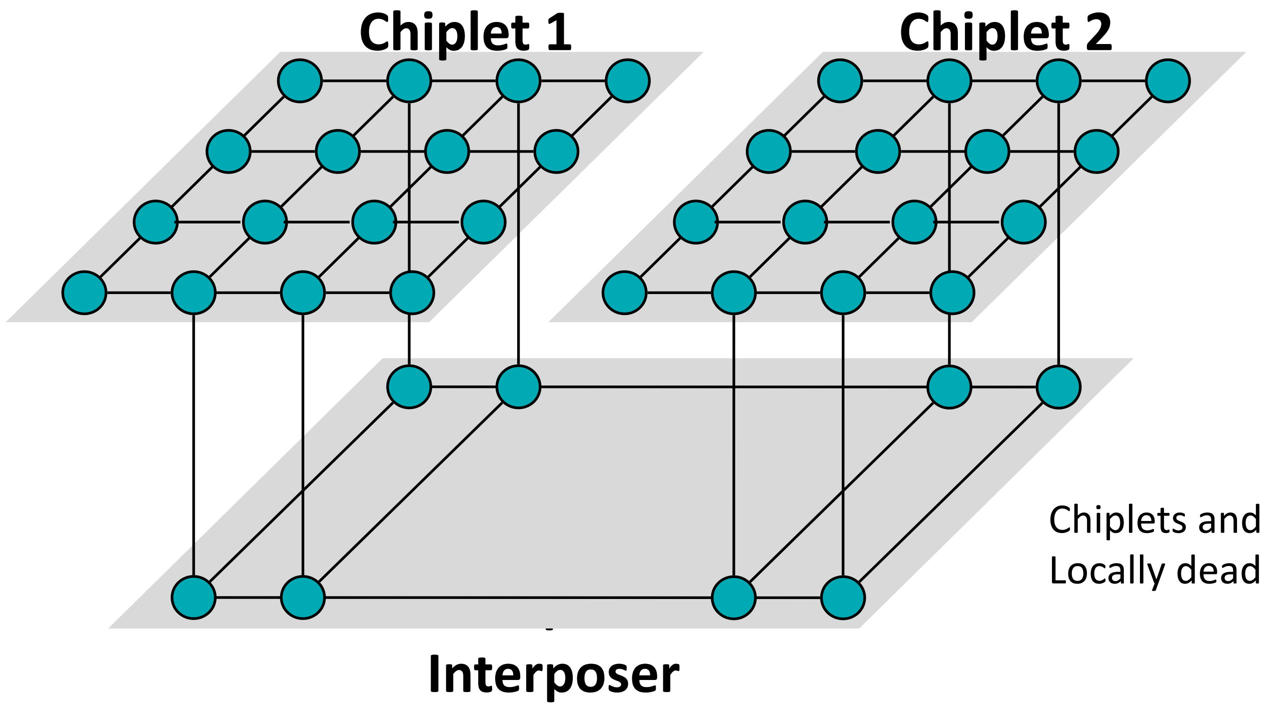
Avoid dependency cycles from forming

Example: Turn restrictions



Deadlock in Chiplet-based Systems

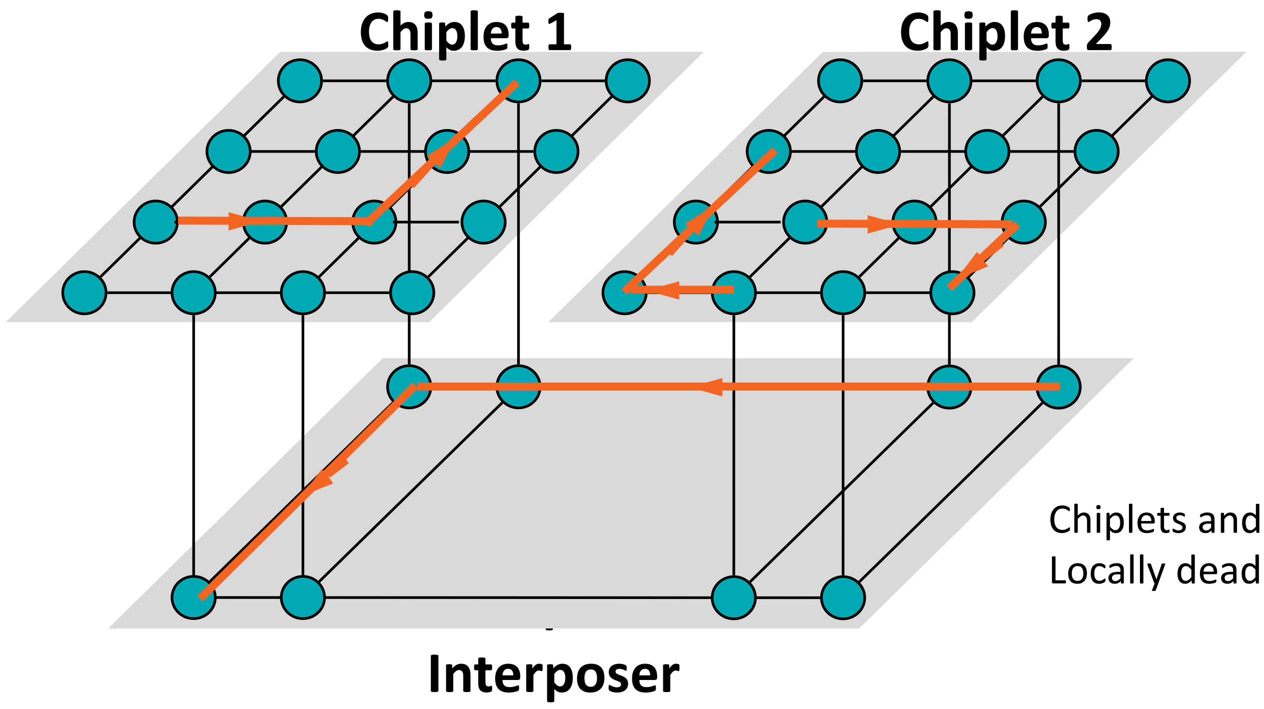
Deadlocks can occur even if individual chiplets are deadlock free



Chiplets and interposer use X-Y routing.
Locally deadlock free.

Deadlock in Chiplet-based Systems

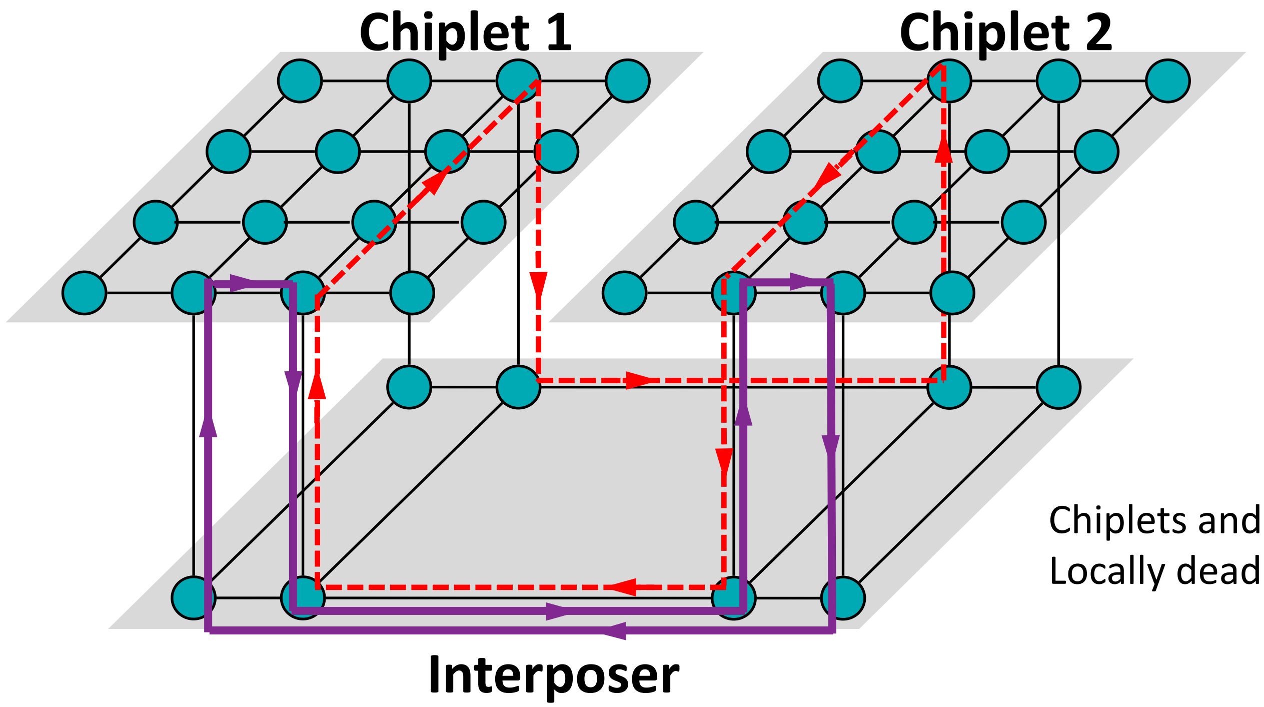
Deadlocks can occur even if individual chiplets are deadlock free



Chiplets and interposer use X-Y routing.
Locally deadlock free.

Deadlock in Chiplet-based Systems

Deadlocks can occur even if individual chiplets are deadlock free



Chiplets and interposer use X-Y routing.
Locally deadlock free.

Chiplet Composability Challenges

Chiplet Composability Challenges

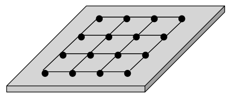
Scalable analysis

- Analyze entire composition of NoCs and all possible paths
- Global channel dependency graph (CDG)

Chiplet Composability Challenges

Scalable analysis

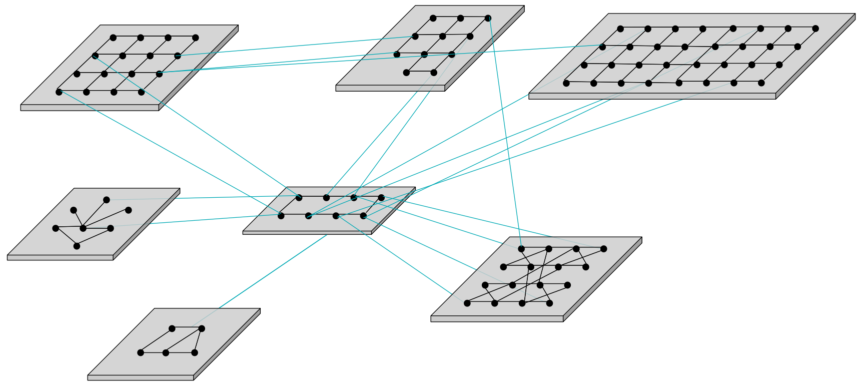
- Analyze entire composition of NoCs and all possible paths
- Global channel dependency graph (CDG)



Chiplet Composability Challenges

Scalable analysis

- Analyze entire composition of NoCs and all possible paths
- Global channel dependency graph (CDG)



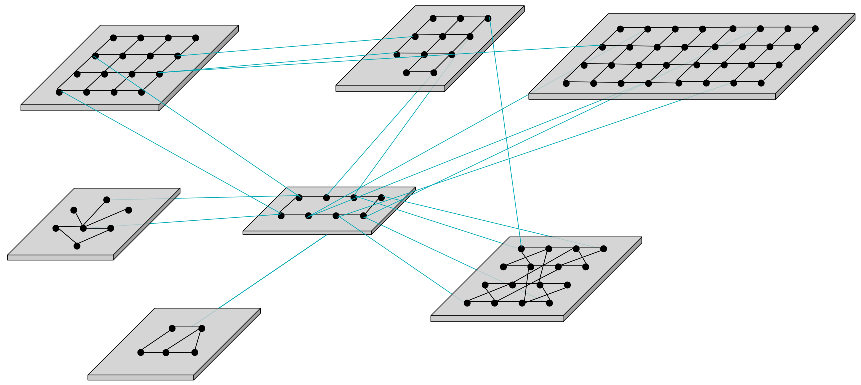
Chiplet Composability Challenges

Scalable analysis

- Analyze entire composition of NoCs and all possible paths
- Global channel dependency graph (CDG)

Local optimized chiplets

- Allow local optimization independent of final SoC organization



Chiplet Composability Challenges

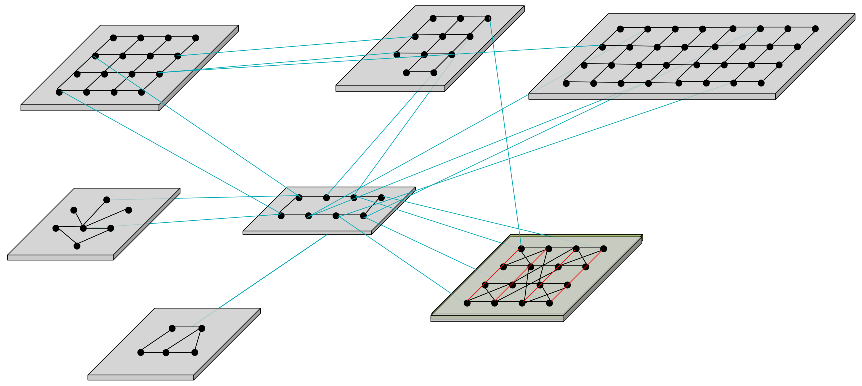
Scalable analysis

Analyze entire composition of NoCs and all possible paths

Global channel dependency graph (CDG)

Local optimized chiplets

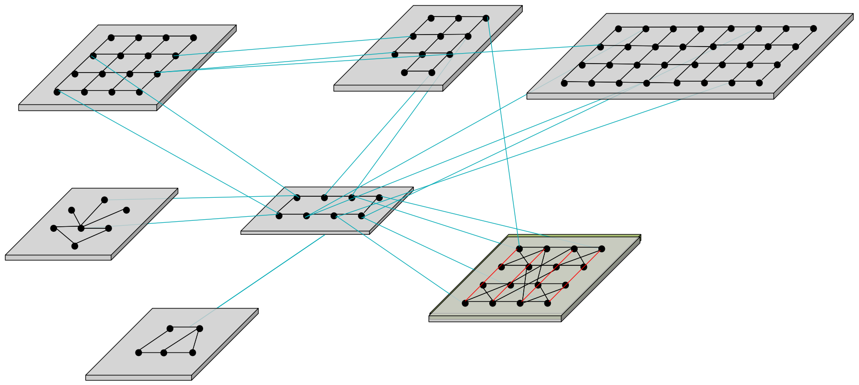
Allow local optimization independent of final SoC organization



Chiplet Composability Challenges

Scalable analysis

- Analyze entire composition of NoCs and all possible paths
- Global channel dependency graph (CDG)



Local optimized chiplets

- Allow local optimization independent of final SoC organization

Lack info on other chiplets in system

- 3rd party may not want to share
- Other chiplets may not have been designed/finalized yet
- Global CDG might NOT be available

Chiplet Composability Challenges

Scalable analysis

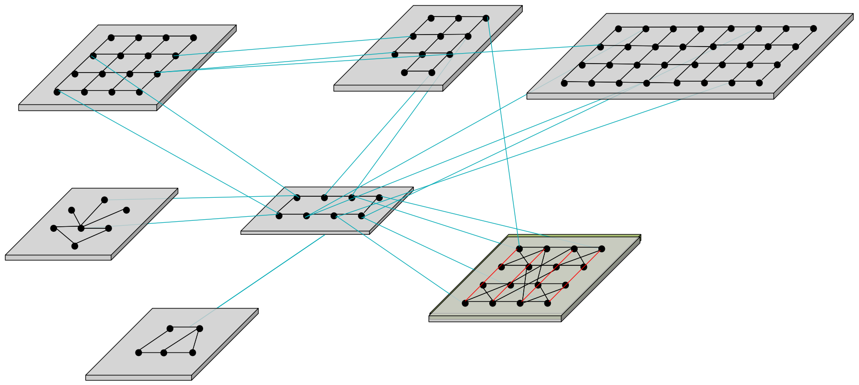
- Analyze entire composition of NoCs and all possible paths
- Global channel dependency graph (CDG)

Local optimized chiplets

- Allow local optimization independent of final SoC organization

Lack info on other chiplets in system

- 3rd party may not want to share
- Other chiplets may not have been designed/finalized yet
- Global CDG might NOT be available



Chiplet Composability Challenges

Scalable analysis

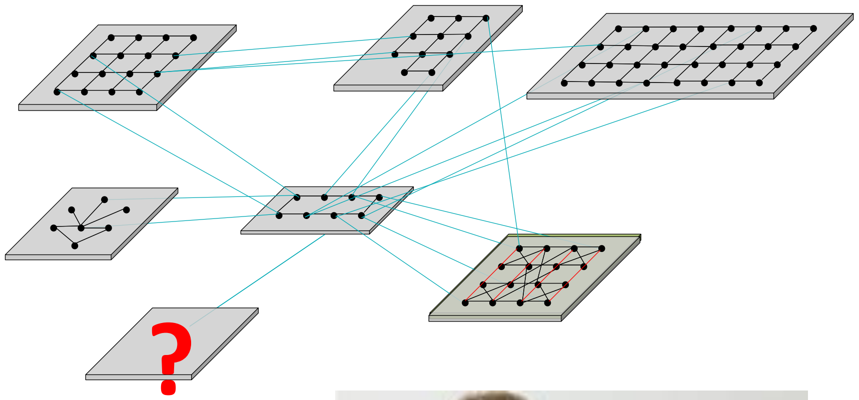
- Analyze entire composition of NoCs and all possible paths
- Global channel dependency graph (CDG)

Local optimized chiplets

- Allow local optimization independent of final SoC organization

Lack info on other chiplets in system

- 3rd party may not want to share
- Other chiplets may not have been designed/finalized yet
- Global CDG might NOT be available



Chiplet Composability Challenges

Analysis scalability

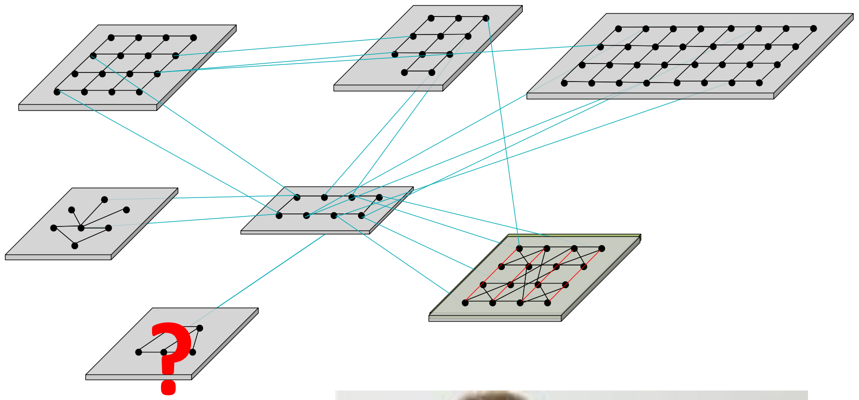
- Analyze entire composition of NoCs and all possible paths
- Global channel dependency graph (CDG)

Local optimized chiplets

- Allow local optimization independent of final SoC organization

Lack info on other chiplets in system

- 3rd party may not want to share
- Other chiplets may not have been designed/finalized yet
- Global CDG might NOT be available



Chiplet Composability Challenges

Analysis scalability

- Analyze entire composition of NoCs and all possible paths
- Global channel dependency graph (CDG)

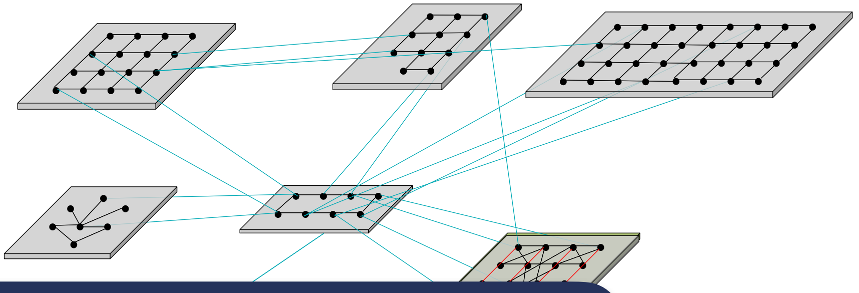
Local optimized chiplets

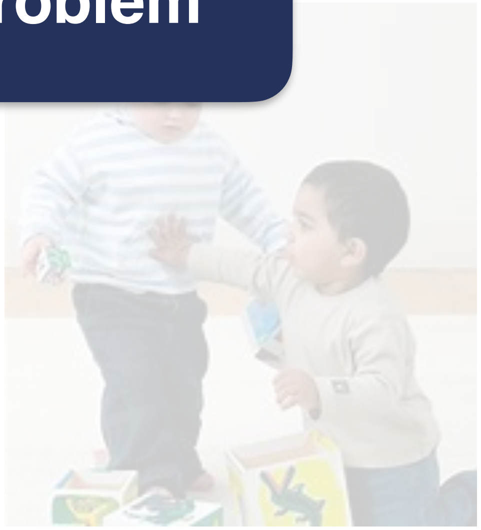
- Allo
- final

Chiplet composability is a HARD problem

Lack info on other chiplets in system

- 3rd party may not want to share
- Other chiplets may not have been designed/finalized yet
- Global CDG might NOT be available





Chiplet Composability Challenges

Analysis scalability

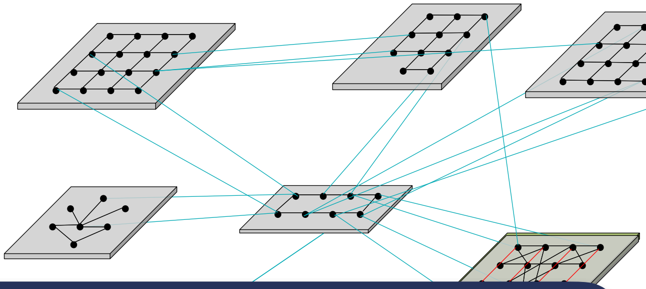
- Analyze entire composition of NoCs and all possible paths
- Global channel dependency graph (CDG)

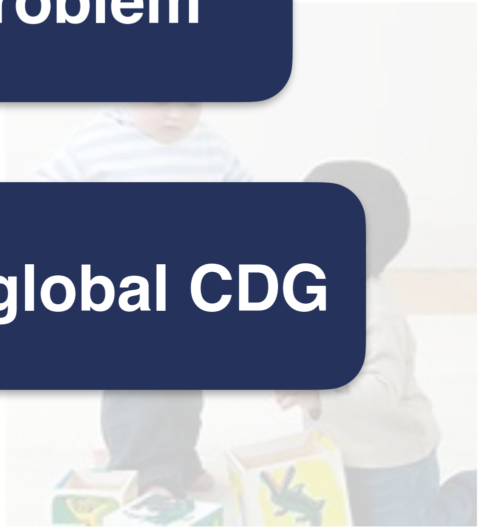
Local optimized chiplets

- Allo
- fina

Lack info on other chiplets in sy

- designed/finalized yet
- Global CDG might NOT be available



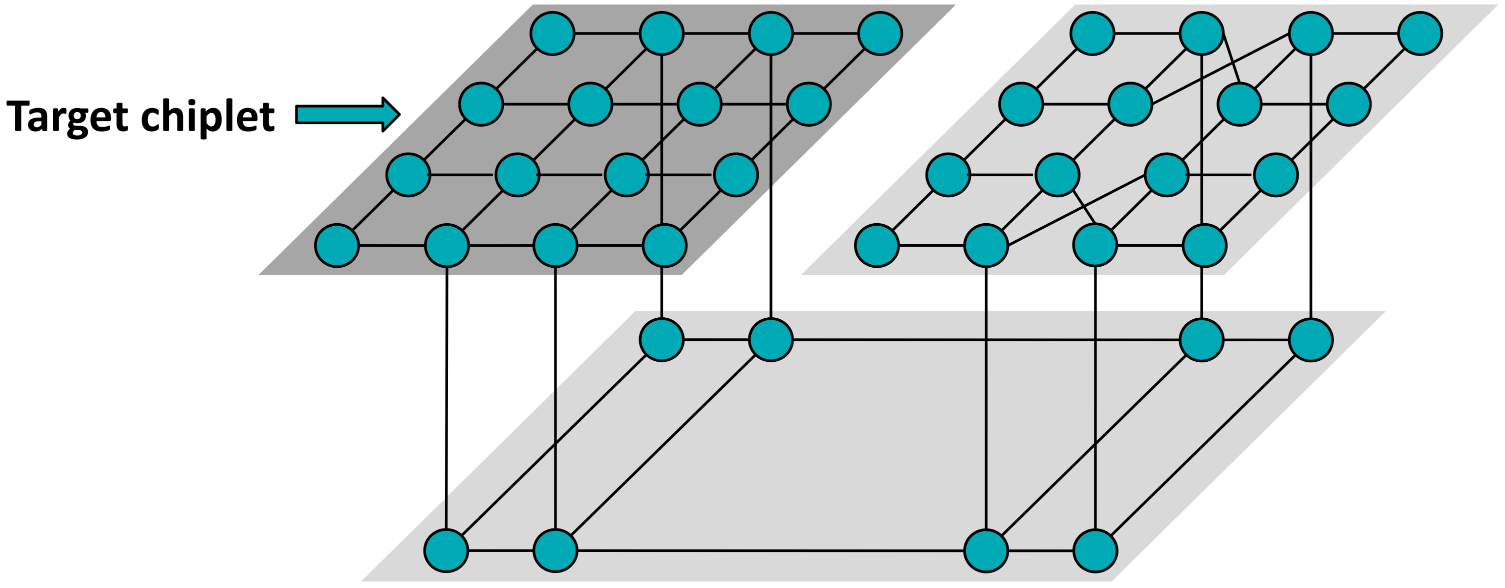


Chiplet composability is a HARD problem

Need a composable approach without global CDG

Our Methodology: Composable Routing

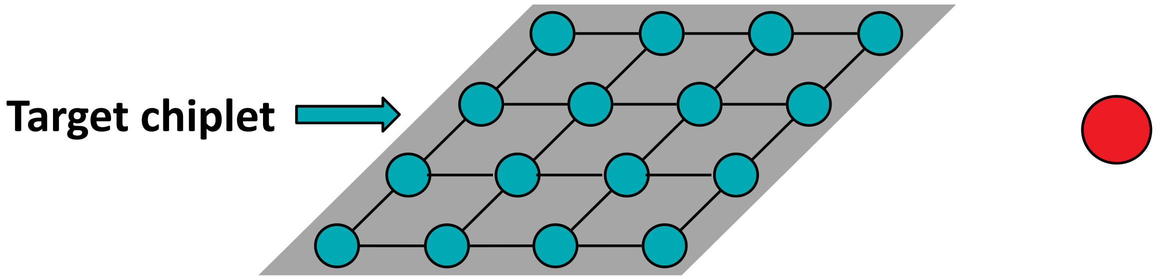
Step 1: Abstract node



Our Methodology: Composable Routing

Step 1: Abstract node

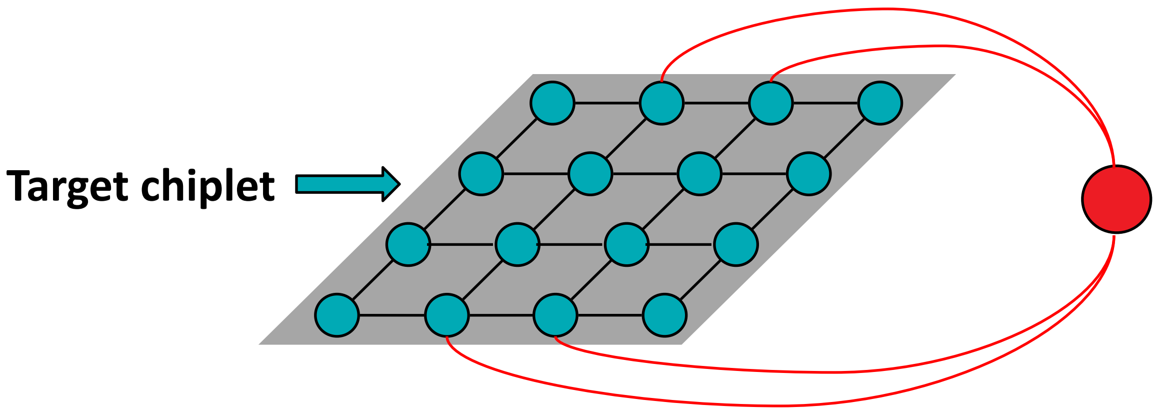
Abstract rest of the system with a single node (key insight)



Our Methodology: Composable Routing

Step 1: Abstract node

- Abstract rest of the system with a single node (key insight)
- Connect the chiplet to the abstract node

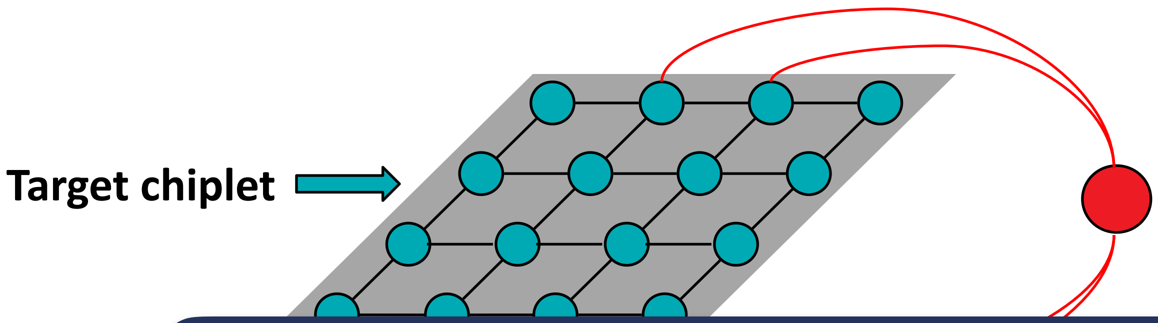


Our Methodology: Composable Routing

Step 1: Abstract node

Abstract rest of the system with a single node (key insight)

Connect the chiplet to the abstract node

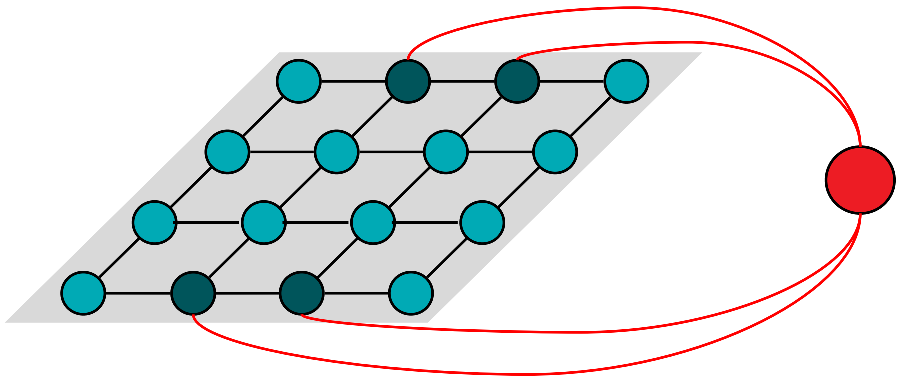


The new system must be deadlock free

Our Methodology: Composable Routing

Step 2: Turn restrictions

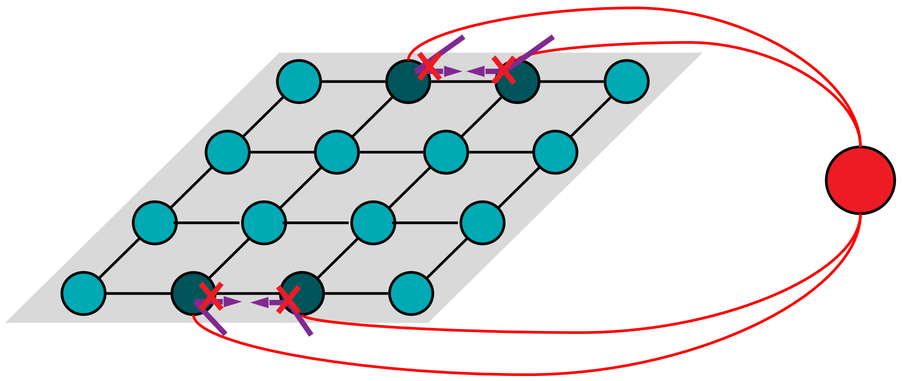
Apply turn restrictions **only at boundary nodes**



Our Methodology: Composable Routing

Step 2: Turn restrictions

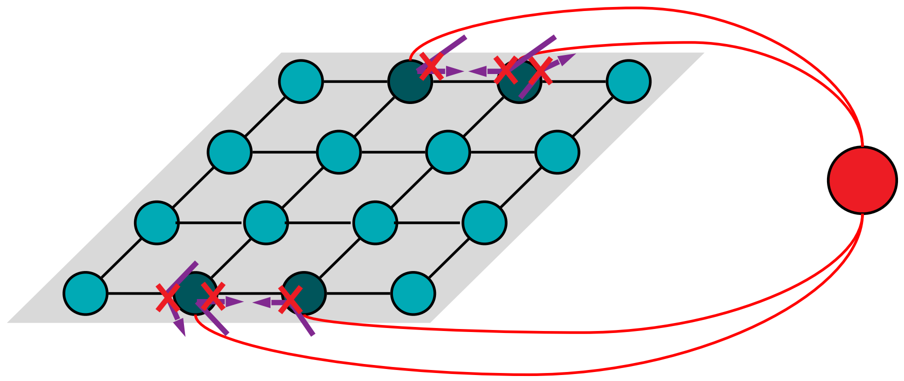
- Apply turn restrictions **only at boundary nodes**
  - Inbound turn restrictions



Our Methodology: Composable Routing

Step 2: Turn restrictions

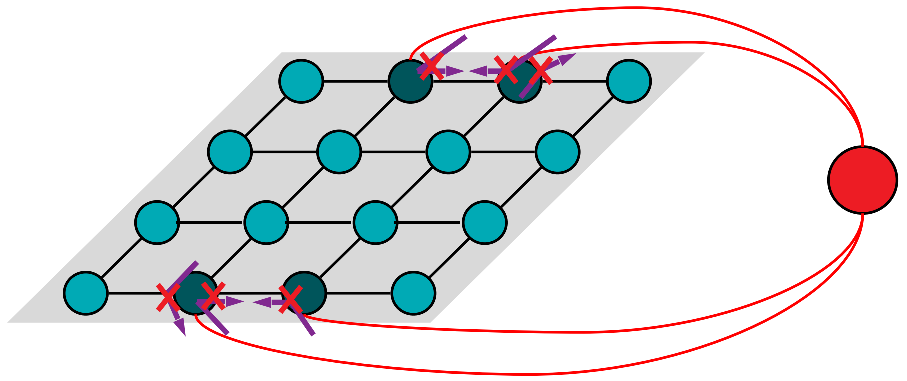
- Apply turn restrictions **only at boundary nodes**
  - Inbound turn restrictions
  - Outbound turn restrictions



Our Methodology: Composable Routing

Step 2: Turn restrictions

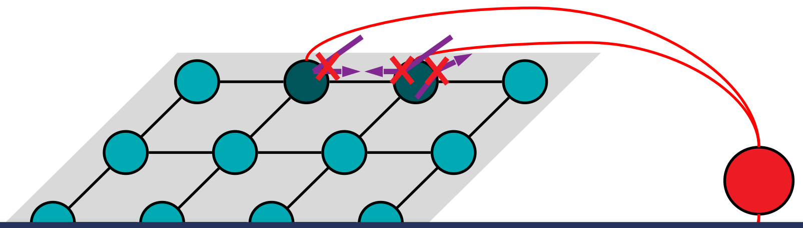
- Apply turn restrictions **only at boundary nodes**
  - Inbound turn restrictions
  - Outbound turn restrictions
- Program chiplet routing tables for outbound messages



Our Methodology: Composable Routing

Step 2: Turn restrictions

- Apply turn restrictions **only at boundary nodes**
  - Inbound turn restrictions
  - Outbound turn restrictions
- Program chiplet routing tables for outbound messages

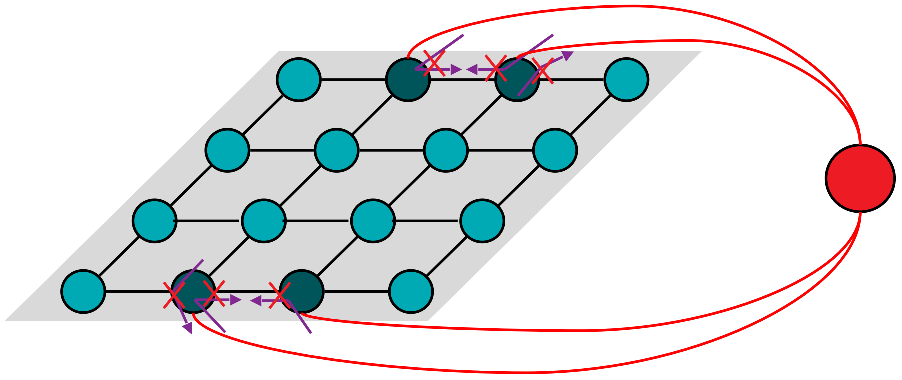


Messages must be routed through the correct boundary nodes

Our Methodology: Composable Routing

Step 3: Reachability

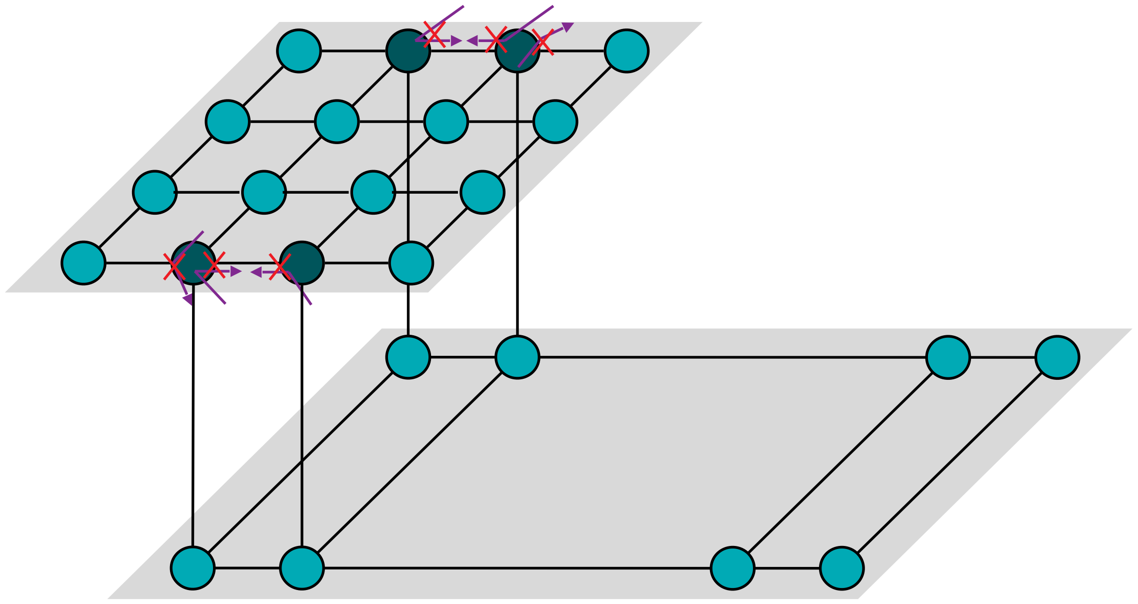
Propagate inbound reachabilities to the interposer (system integrator)



Our Methodology: Composable Routing

Step 3: Reachability

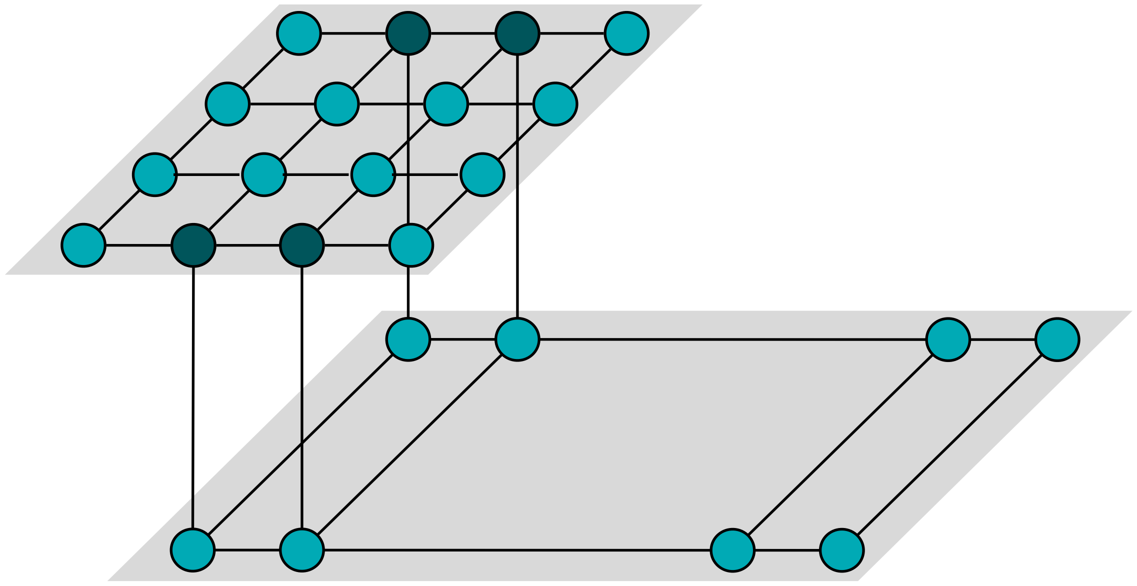
Propagate inbound reachabilities to the interposer (system integrator)



Our Methodology: Composable Routing

Step 3: Reachability

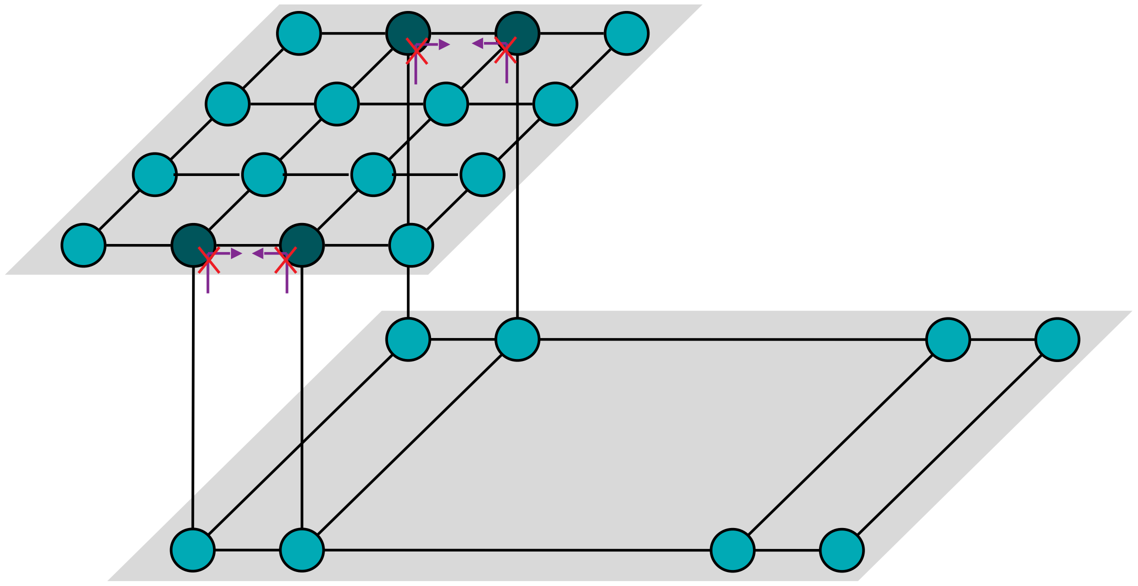
Propagate inbound reachabilities to the interposer (system integrator)



Our Methodology: Composable Routing

Step 3: Reachability

Propagate inbound reachabilities to the interposer (system integrator)

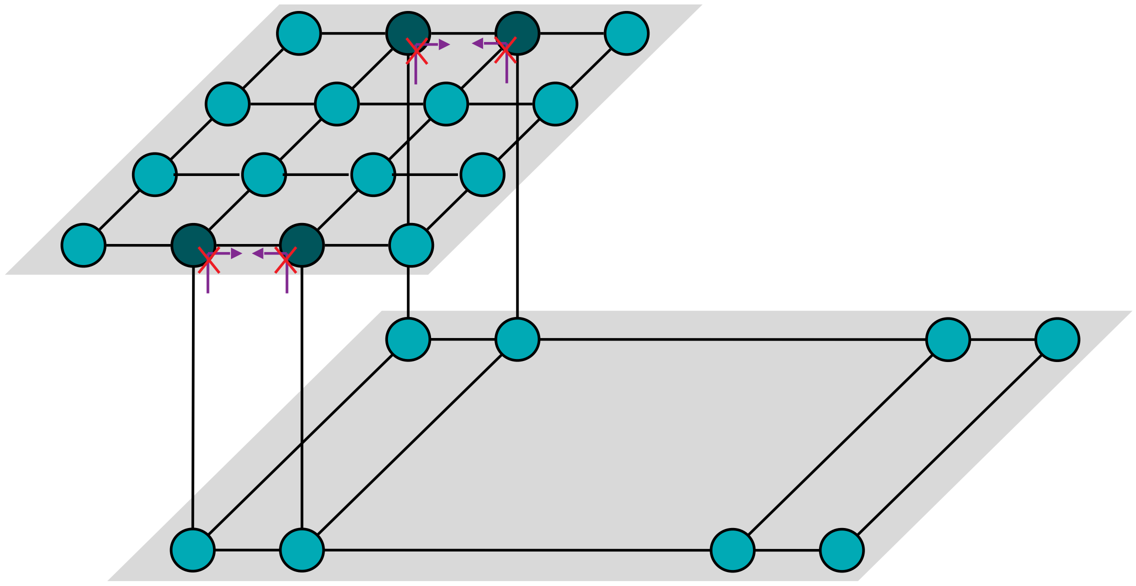


Our Methodology: Composable Routing

Step 3: Reachability

Propagate inbound reachabilities to the interposer (system integrator)

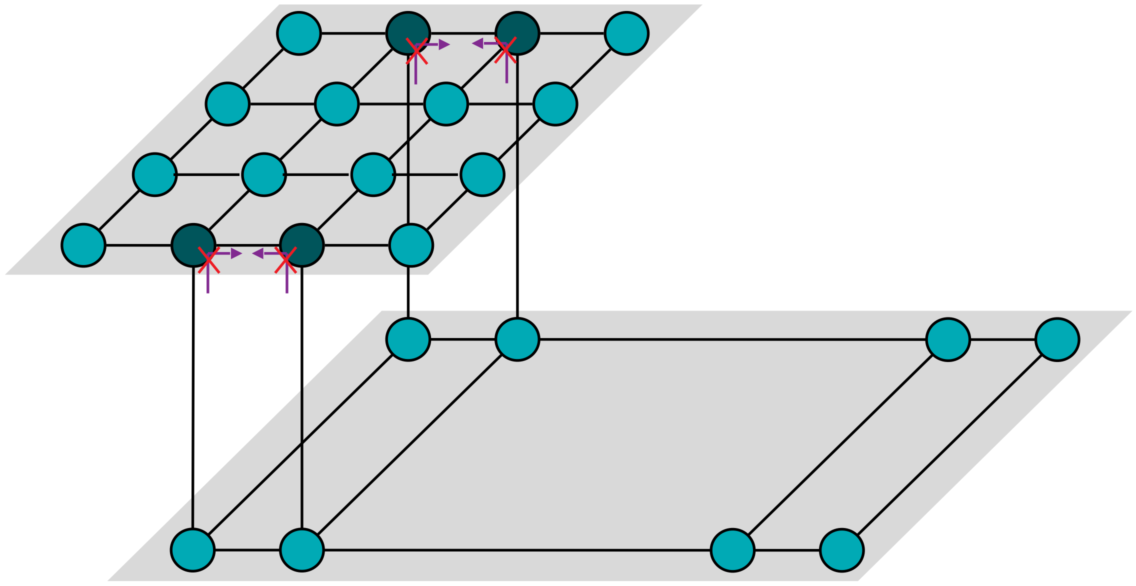
Program interposer routing tables at integration



Our Methodology: Composable Routing

Step 3: Reachability

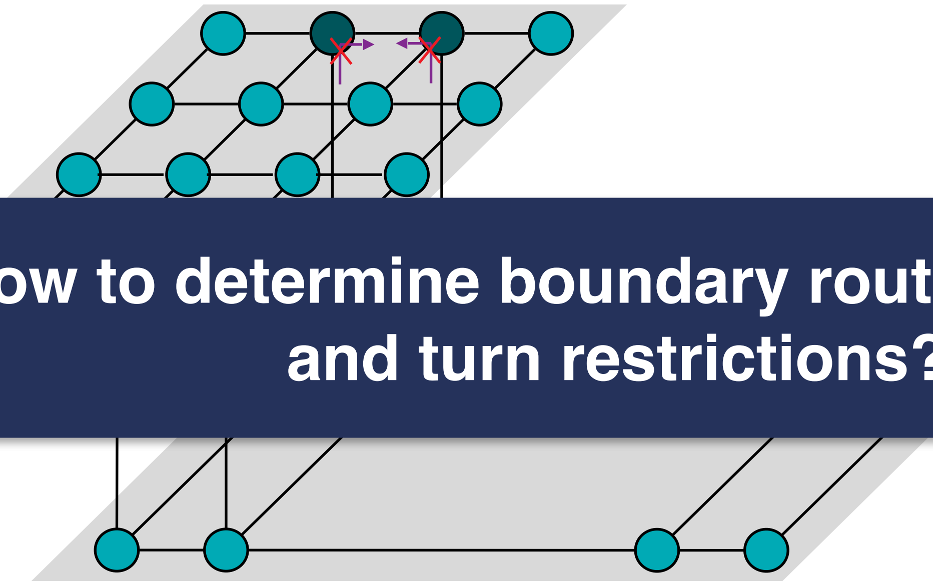
- Propagate inbound reachabilities to the interposer (system integrator)
- Program interposer routing tables at integration
- Interposer NoC must be deadlock-free by itself



Our Methodology: Composable Routing

Step 3: Reachability

- Propagate inbound reachabilities to the interposer (system integrator)
- Program interposer routing tables at integration
- Interposer NoC must be deadlock-free by itself



How to determine boundary router locations and turn restrictions?

Our Methodology: Composable Routing

Boundary router placement

- Physical constraints
- Load balancing
- Route distance

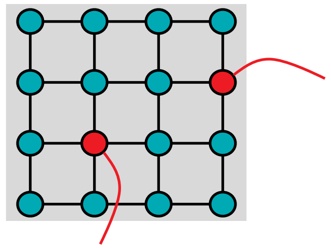
Turn restriction

- Distance to/from boundary node
- Load balance

Our Methodology: Composable Routing

Boundary router placement

- Physical constraints
- Load balancing
- Route distance



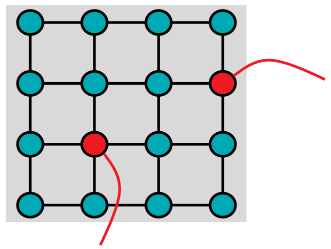
Turn restriction

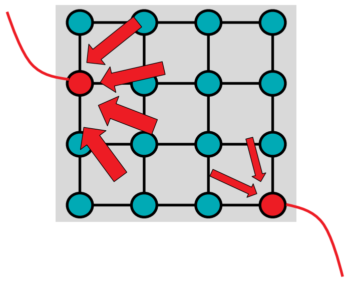
- Distance to/from boundary node
- Load balance

Our Methodology: Composable Routing

Boundary router placement

- Physical constraints
- Load balancing
- Route distance





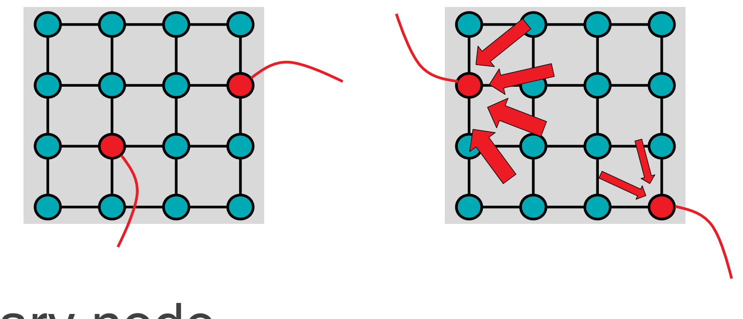
Turn restriction

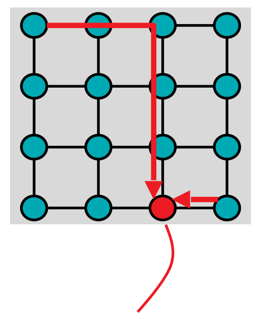
- Distance to/from boundary node
- Load balance

Our Methodology: Composable Routing

Boundary router placement

- Physical constraints
- Load balancing
- Route distance





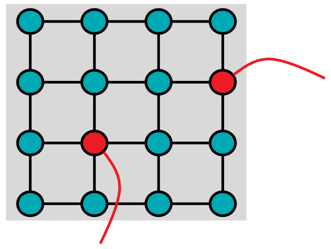
Turn restriction

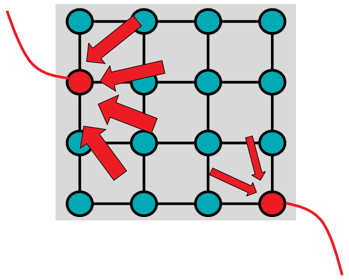
- Distance to/from boundary node
- Load balance

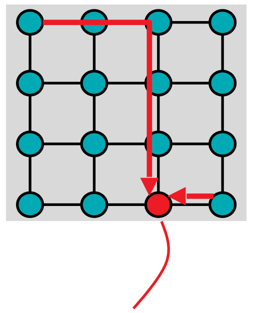
Our Methodology: Composable Routing

Boundary router placement

- Physical constraints
- Load balancing
- Route distance







Turn restriction

- Distance to/from boundary node
- Load balance

Objective function

- Distance
- Reachability

Our Methodology: Composable Routing

Boundary router placement

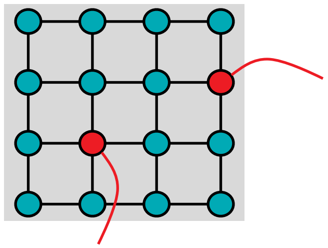
- Physical constraints
- Load balancing
- Route distance

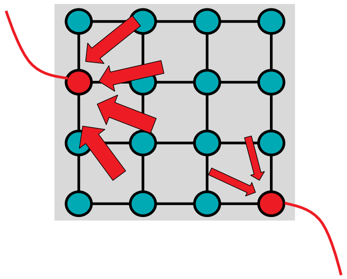
Turn restriction

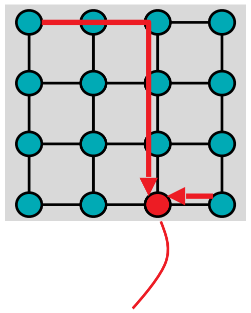
- Distance to/from boundary node
- Load balance

Objective function

- Distance
- Reachability



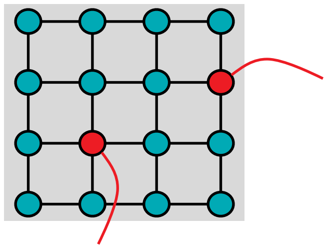


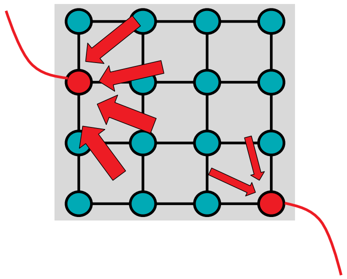


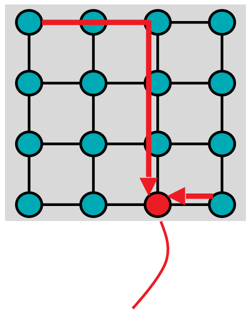
Our Methodology: Composable Routing

Boundary router placement

- Physical constraints
- Load balancing
- Route distance







Turn restriction

- Distance to/from boundary node
- Loa

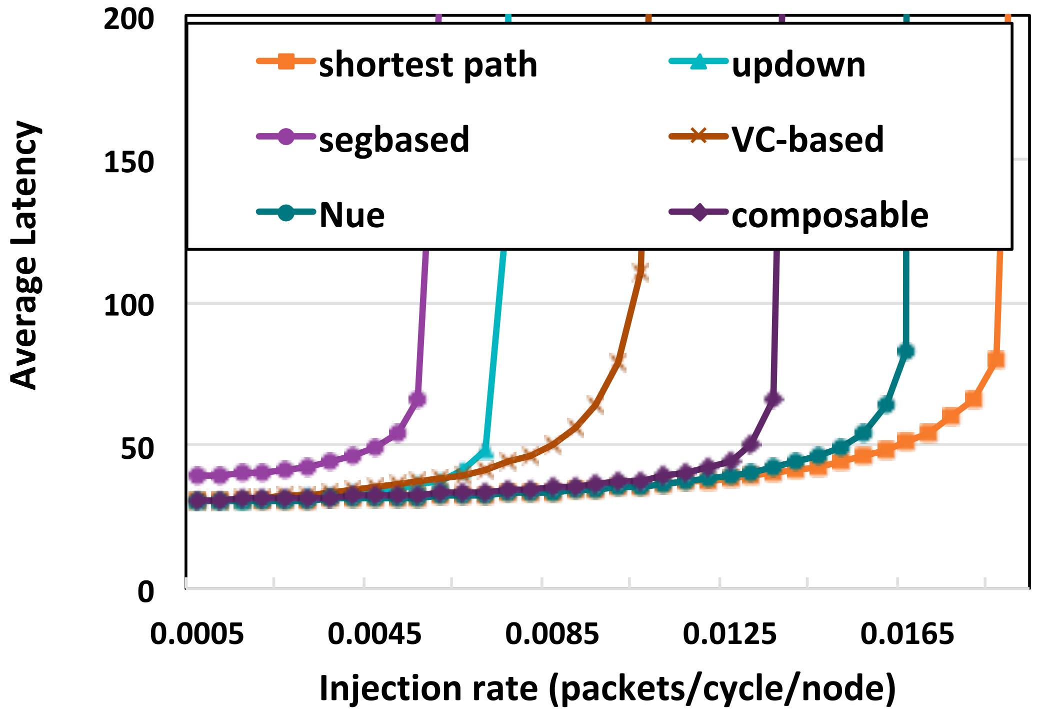
Obje

- Dist
- Rea

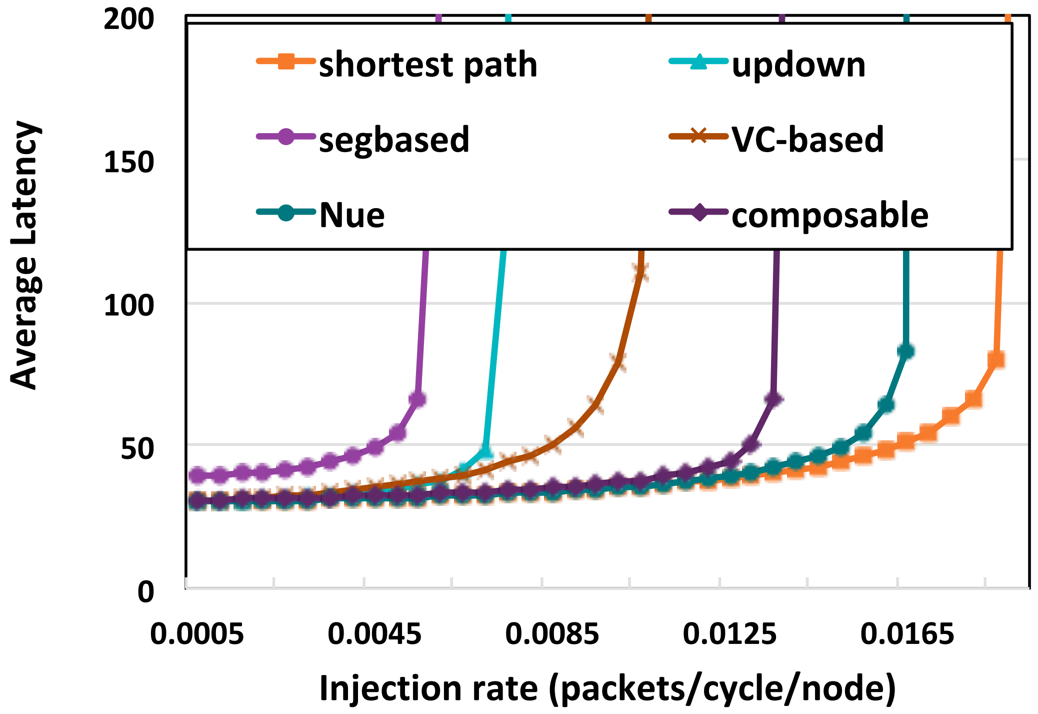
Our objective: minimize

$$\frac{\textit{average distance}}{\textit{average reachability}}$$

Composable Routing Take-aways

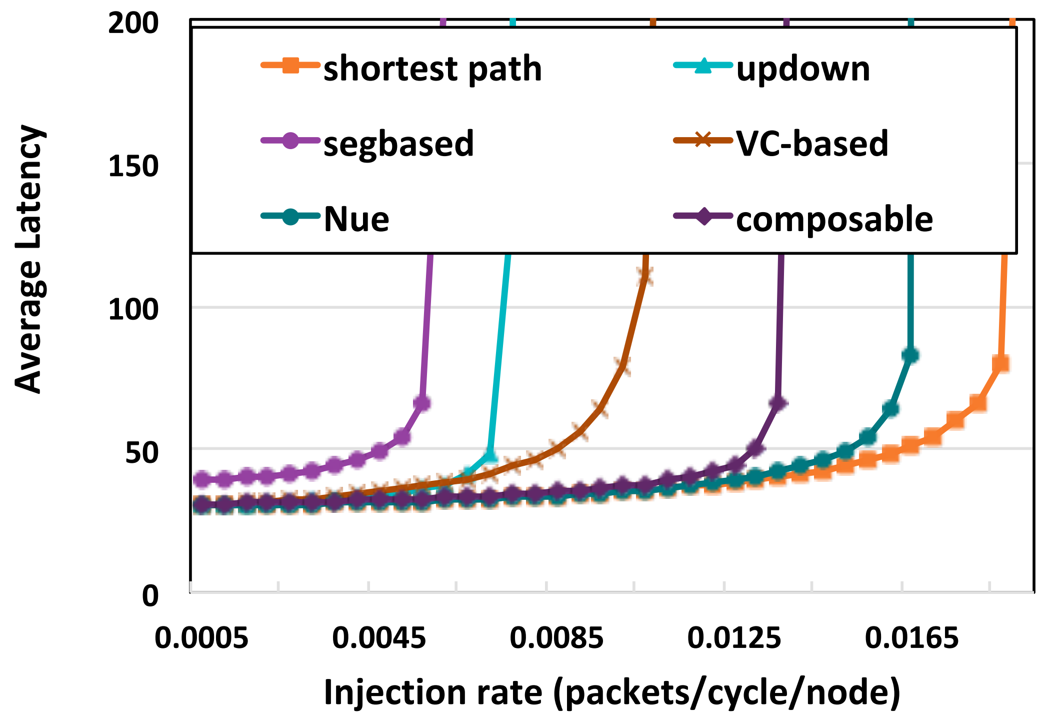


Composable Routing Take-aways



Does not require a CDG

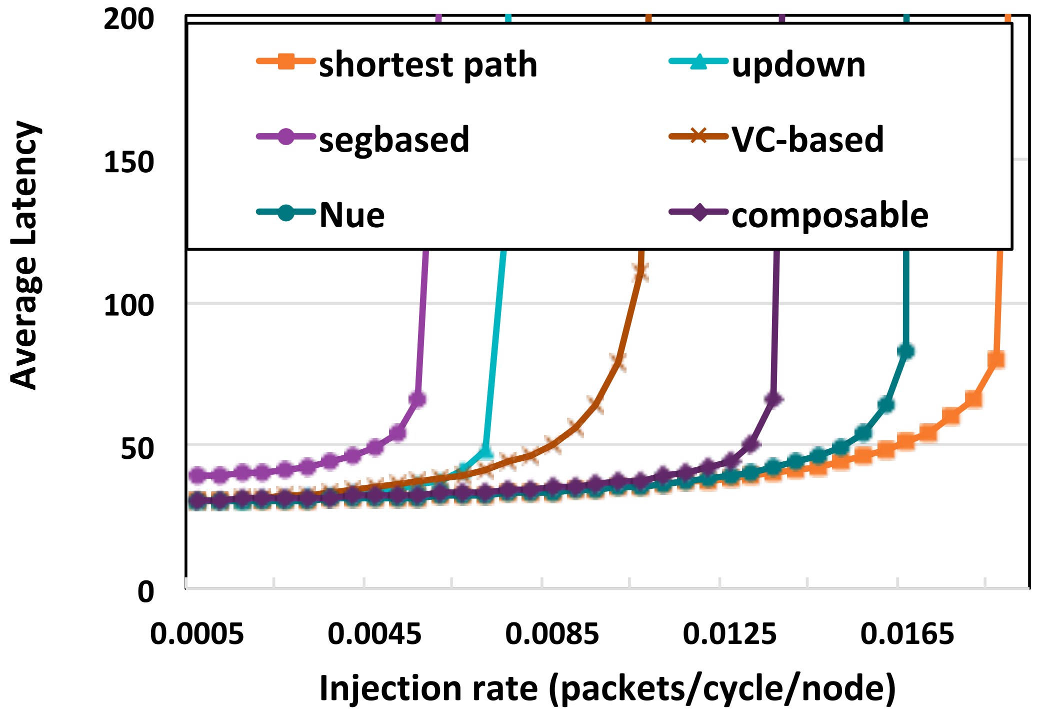
Composable Routing Take-aways



Does not require a CDG

Outperforms most prior work

Composable Routing Take-aways

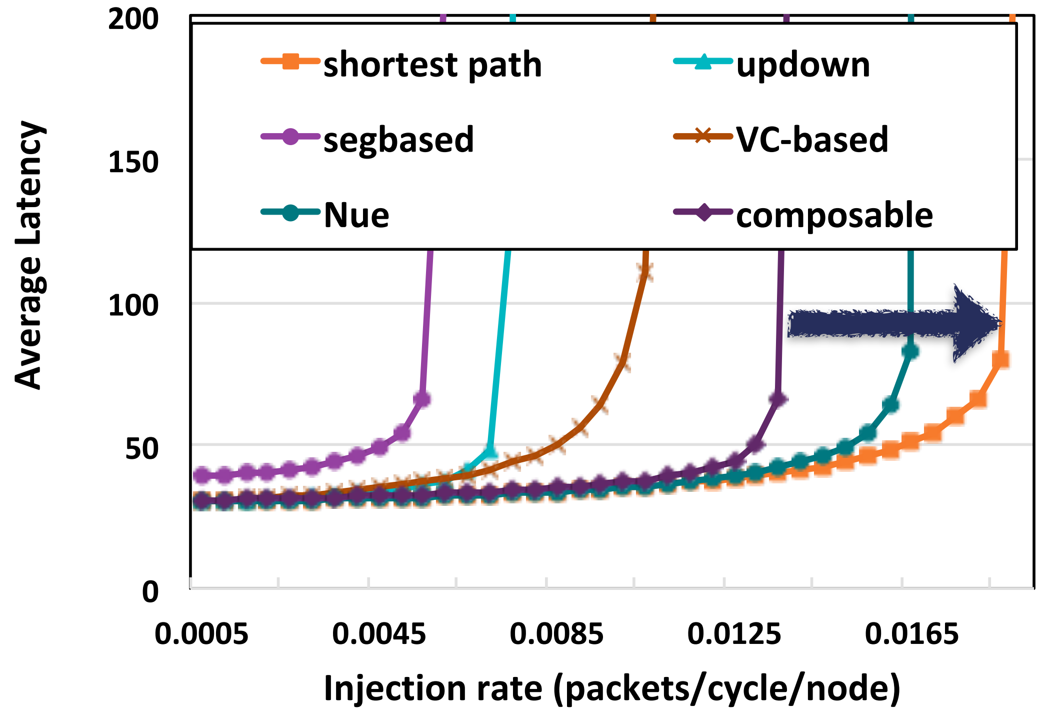


Does not require a CDG

Outperforms most prior work

Room for improvement: load imbalance and head-of-line blocking

Composable Routing Take-aways



Does not require a CDG

Outperforms most prior work

Room for improvement: load imbalance and head-of-line blocking

What are the open challenges and opportunities?

Challenges: Active Interposer

Challenges: Active Interposer

Passive interposers currently in fashion

- Can manufacture minimally active interposer with reasonable cost

Challenges: Active Interposer

Passive interposers currently in fashion

- Can manufacture minimally active interposer with reasonable cost

Challenges: Active Interposer

Passive interposers currently in fashion

- Can manufacture minimally active interposer with reasonable cost

Opportunity to offload more functionality to interposer

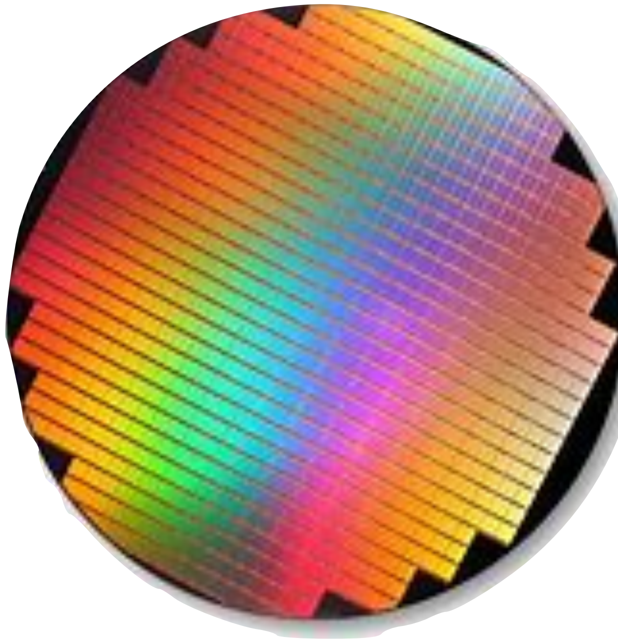
- System monitoring, security features, auxiliary compute devices

Challenges: Process

Challenges: Process

Die-to-die variations in re-integrated SoC

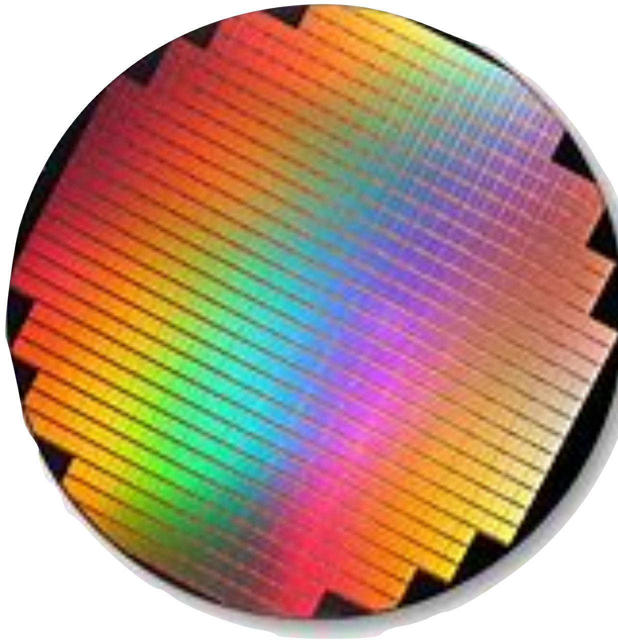
- Additional timing or voltage margins
- Less efficient, lower performance



Challenges: Process

Die-to-die variations in re-integrated SoC

- Additional timing or voltage margins
- Less efficient, lower performance



Challenges: Process

Die-to-die variations in re-integrated SoC

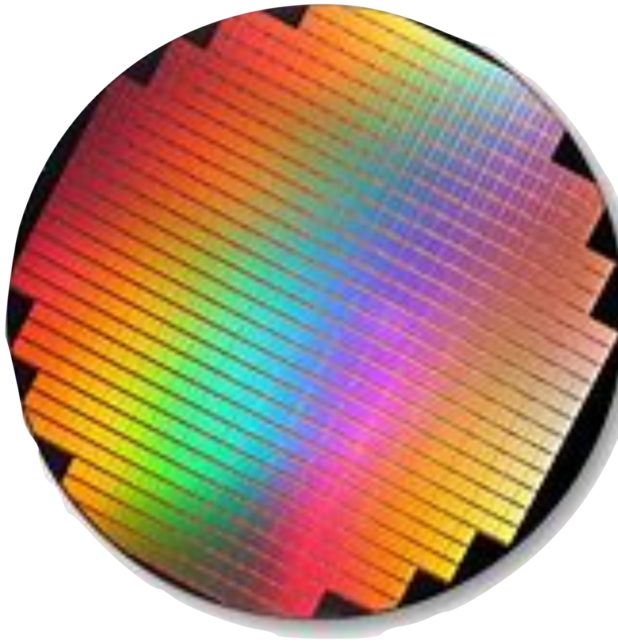
Additional timing or voltage margins

Less efficient, lower performance

Independent clock domains

DVFS management

Mitigate overheads of clock crossings



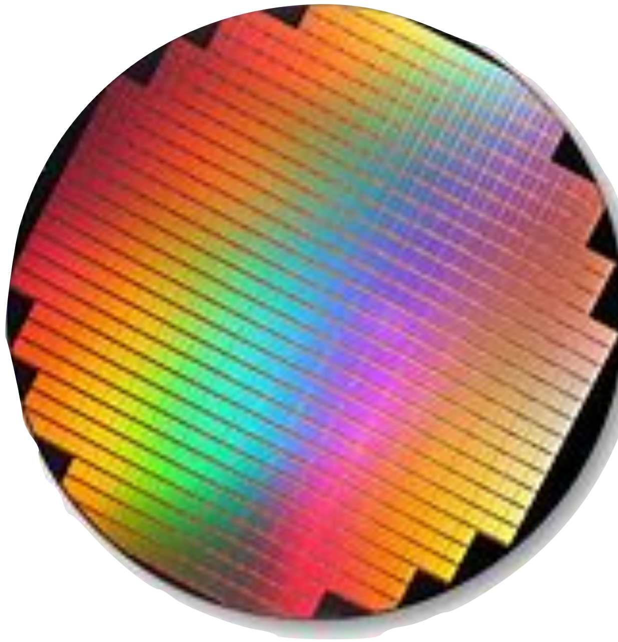
Challenges: Process

Die-to-die variations in re-integrated SoC

- Additional timing or voltage margins
- Less efficient, lower performance

Independent clock domains

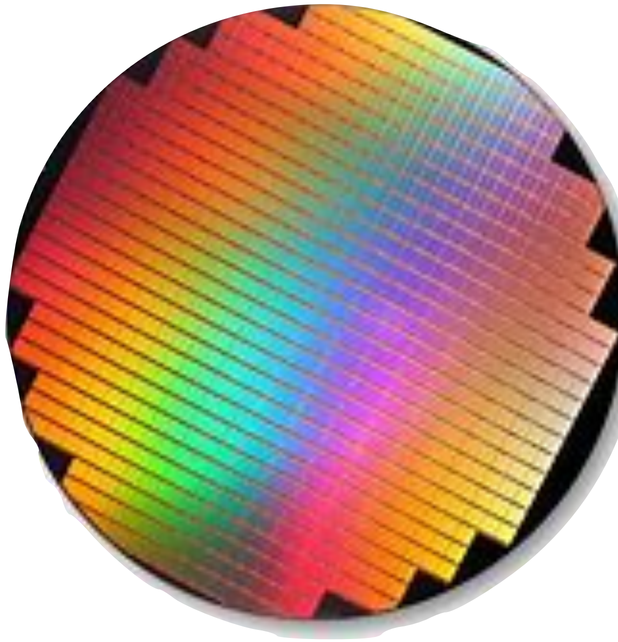
- DVFS management
- Mitigate overheads of clock crossings



Challenges: Process

Die-to-die variations in re-integrated SoC

- Additional timing or voltage margins
- Less efficient, lower performance



Independent clock domains

- DVFS management
- Mitigate overheads of clock crossings

3D NoC spanning multiple process technologies?

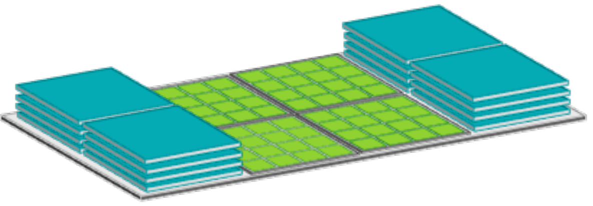
Opportunity: Chiplet organization

Opportunity: Chiplet organization

Alternative chiplet placements?

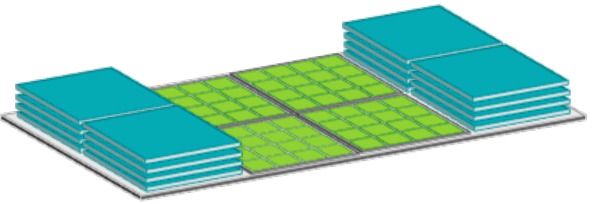
Opportunity: Chiplet organization

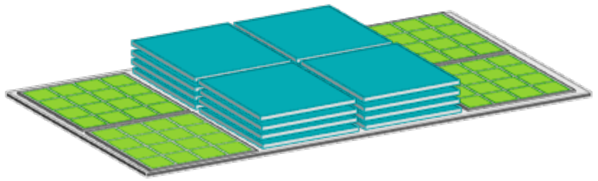
Alternative chiplet placements?



Opportunity: Chiplet organization

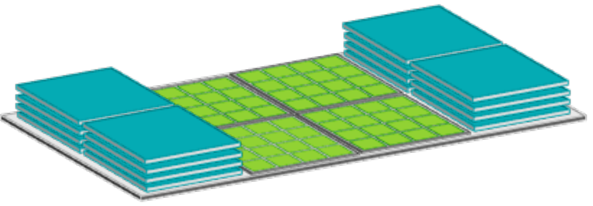
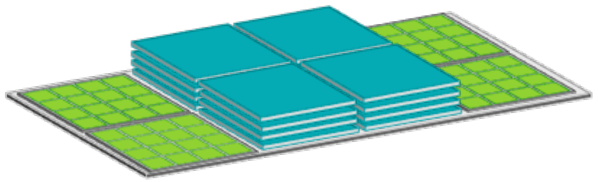
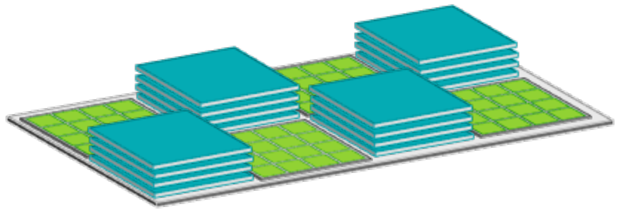
Alternative chiplet placements?





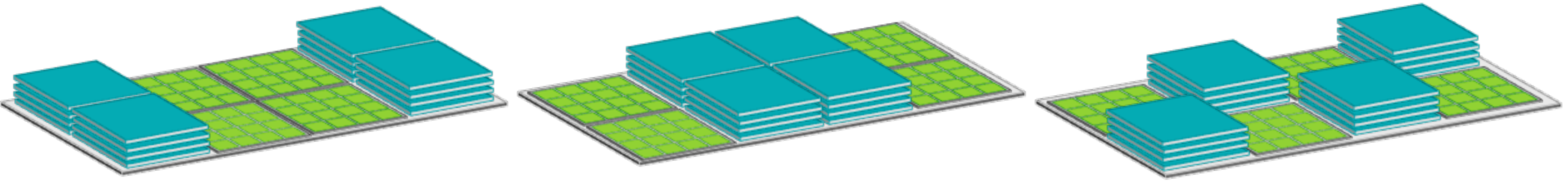
Opportunity: Chiplet organization

Alternative chiplet placements?

Opportunity: Chiplet organization

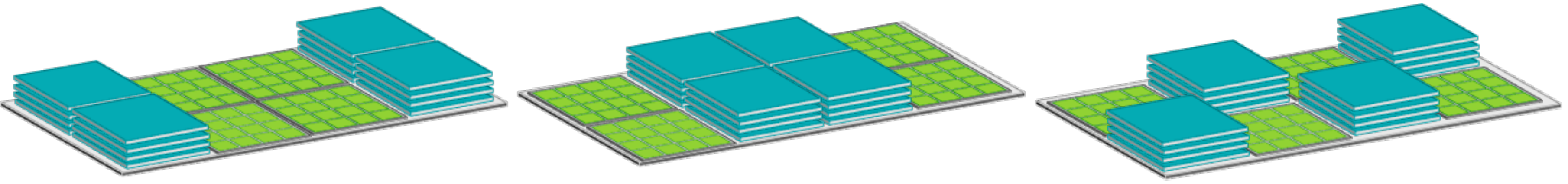
Alternative chiplet placements?



Change NoC traffic patterns — new bottlenecks, new opportunities

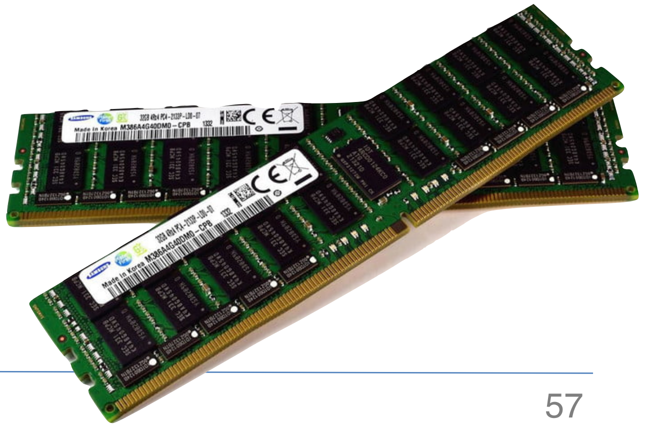
Opportunity: Chiplet organization

Alternative chiplet placements?



Change NoC traffic patterns — new bottlenecks, new opportunities

Interaction between in-package memory stacks and external memories?



Opportunity: Heterogeneous SoCs

Opportunity: Heterogeneous SoCs

End of scaling

- Rise of accelerators to provide performance, power efficiency, security

Opportunity: Heterogeneous SoCs

End of scaling

- Rise of accelerators to provide performance, power efficiency, security

Mix and match

- Not all systems need every flavour of accelerator

Opportunity: Heterogeneous SoCs

End of scaling

- Rise of accelerators to provide performance, power efficiency, security

Mix and match

- Not all systems need every flavour of accelerator

Additional challenges

- Interfaces
- QoS
- Coherence

Conclusions

Conclusions

Disintegrate chips

Build cost-effective LARGE SoCs

Conclusions

Disintegrate chips

- Build cost-effective LARGE SoCs

Reintegrate with an active silicon interposer

- Minimal active area to reduce cost
- Novel NoC topologies to improve performance

Conclusions

Disintegrate chips

- Build cost-effective LARGE SoCs

Reintegrate with an active silicon interposer

- Minimal active area to reduce cost
- Novel NoC topologies to improve performance

Ensure composability

- Deadlock-free routing that allows chiplets to be optimized independently

Conclusions

Disintegrate chips

- Build cost-effective LARGE SoCs

Reintegrate with an active silicon interposer

- Minimal active area to reduce cost
- Novel NoC topologies to improve performance

Ensure composability

- Deadlock-free routing that allows chiplets to be optimized independently

Open questions and opportunities for research!

Acknowledgements

Graduate Students

Ajay Kannan, Zimo Li

Collaborators at AMD

Gabriel Loh, Jieming Yin, Onur Karyiran, Matthew Poremba, Zhifeng Lin, Muhammad Shoaib Bin Altaf, Yasuko Eckert

Funding

UofT, Natural Science and Engineering Research Council

Thanks

Natalie Enright Jerger
enright@ece.utoronto.ca