Scalar Operand Networks for Tiled Microprocessors

Michael Taylor

Raw Architecture Project
MIT CSAIL

(now at UCSD)

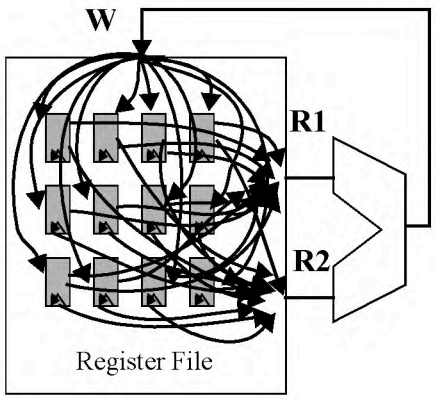
Until 3 years ago – computer architects have been using the N-way superscalar to encapsulate the ideal for a parallel processor…

- nearly "perfect" but not attainable

	Superscalar
"PE"->"PE" communication	Free
exploitation of parallelism	Implicit
Clean semantics	Yes
scalable	No
power efficient	No

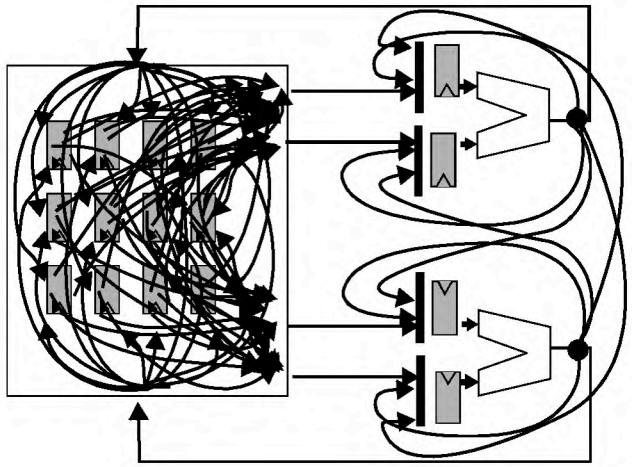
(or VLIW)

(hw scheduler or compiler)



mul $2,$3,$4

add $6,$5,$2



What's great about superscalar microprocessors?

→ It's the networks!

Fast low-latency tightly-coupled networks

(0-1 cycles of latency,

no occupancy)

-For the lack of a better name

let's call them *Scalar Operand Networks (SONs)*

- Can we incorporate the benefits of superscalar communication + multicore scalability

-Can we build *Scalable* Scalar Operand Networks?

(I agree with Jose: "We need low-latency tightly-coupled … network interfaces" – Jose Duato, OCIN, Dec 6, 2006)

The industry shift toward Multicore
- attainable but hardly ideal

	Superscalar	Multicore
"PE"->"PE" communication	Free	Expensive
exploitation of parallelism	Implicit	Explicit
Clean semantics	Yes	No
scalable	No	Yes
power efficient	No	Yes

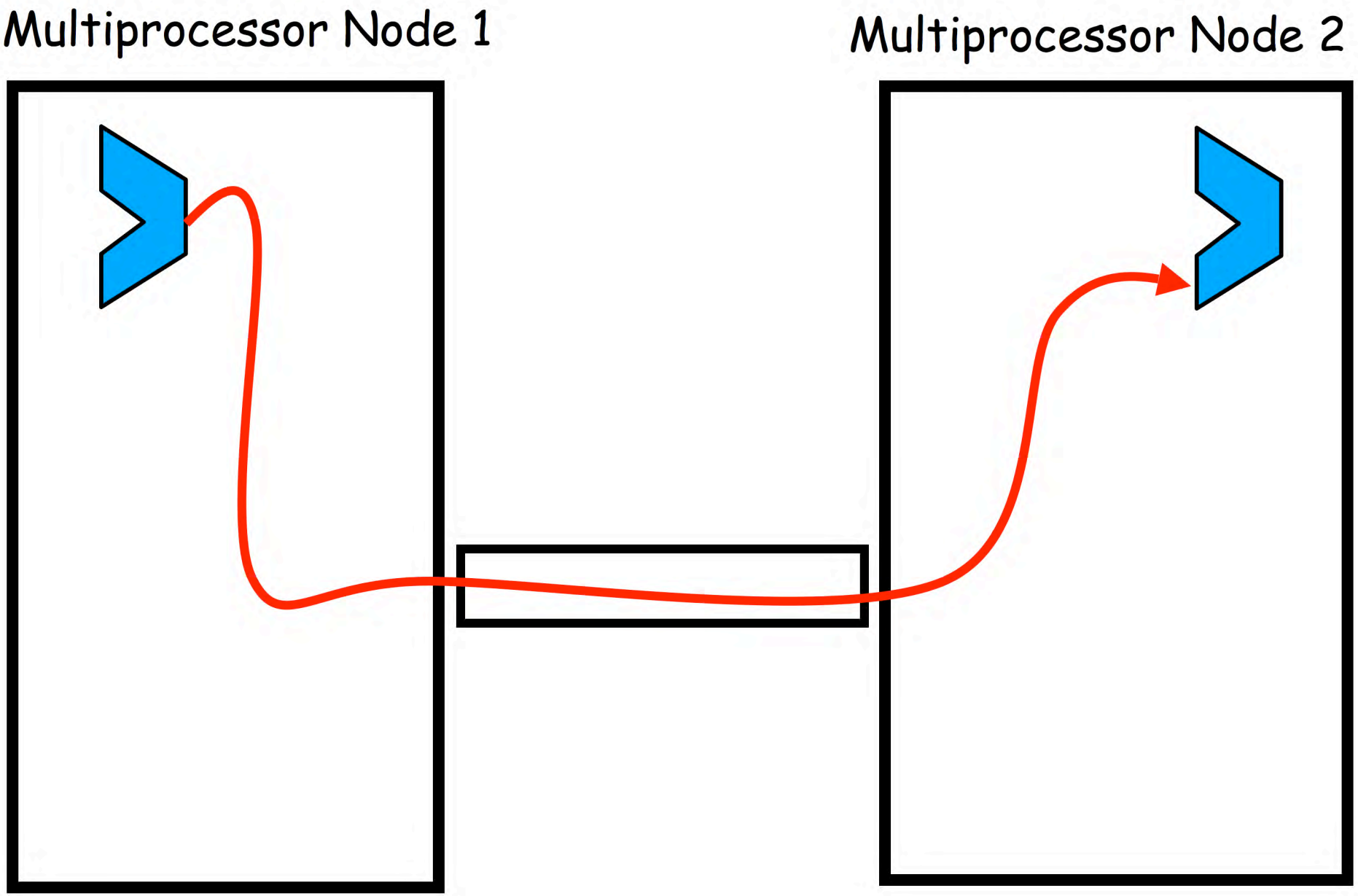
What we'd like – neither superscalar nor multicore

	Superscalar	Multicore
"PE"->"PE" communication	Free	Expensive
exploitation of parallelism	Implicit	Explicit
Clean semantics	Yes	No
scalable	No	Yes
power efficient	No	Yes

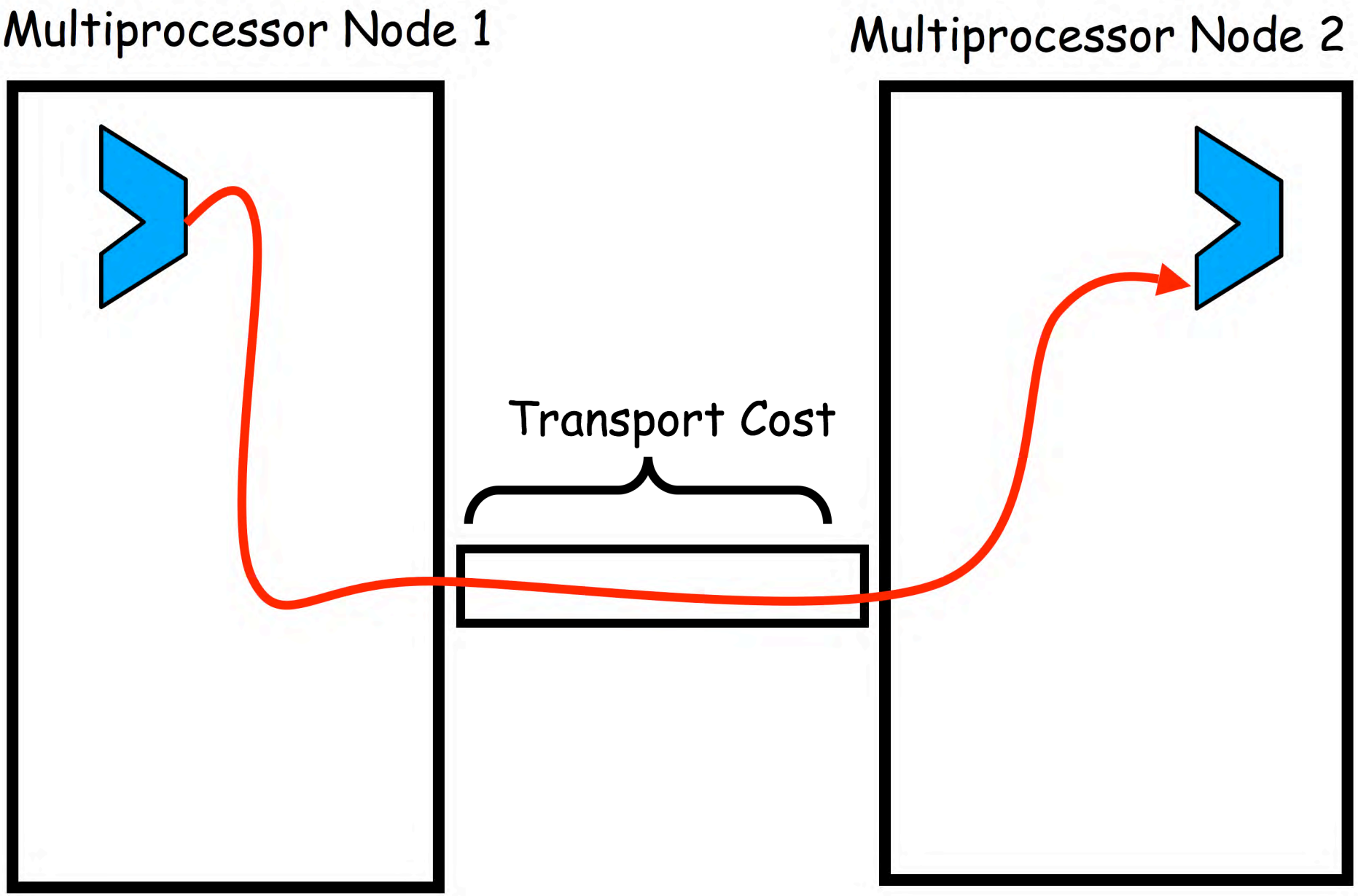
Superscalars have fast networks and great usability

Multicore has great scalability and efficiency

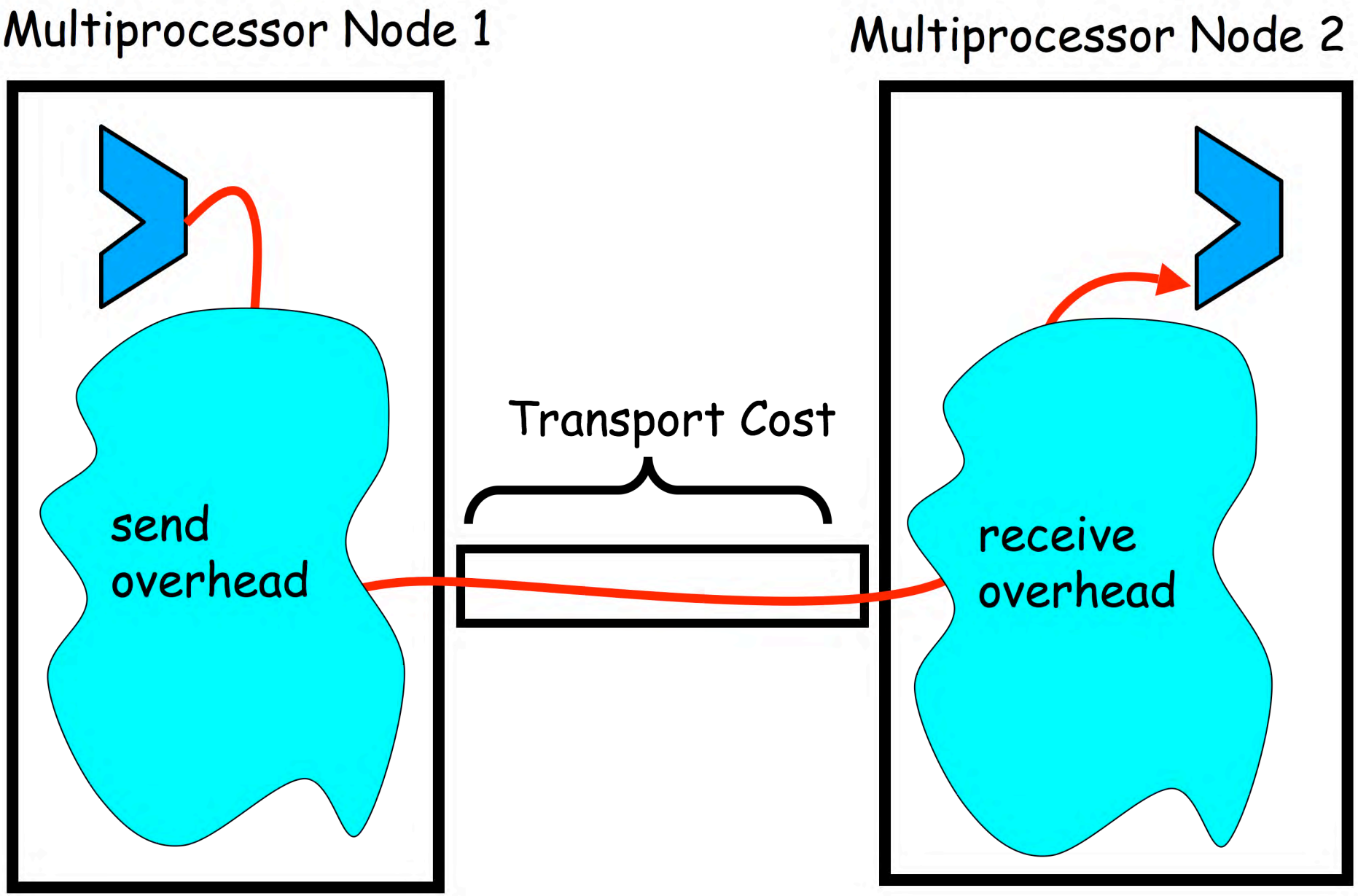
Why communication is expensive on multicore



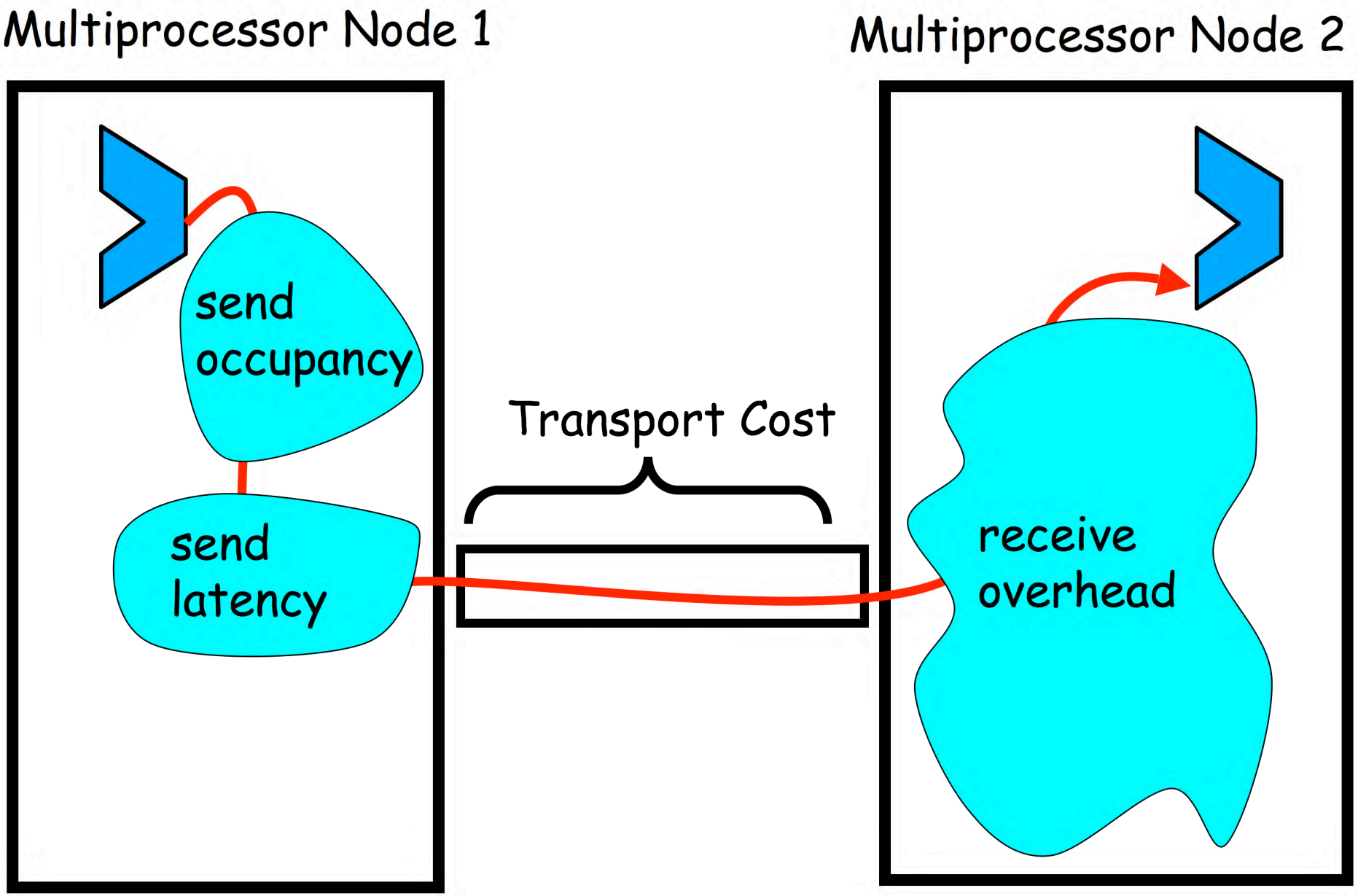
Why communication is expensive on multicore



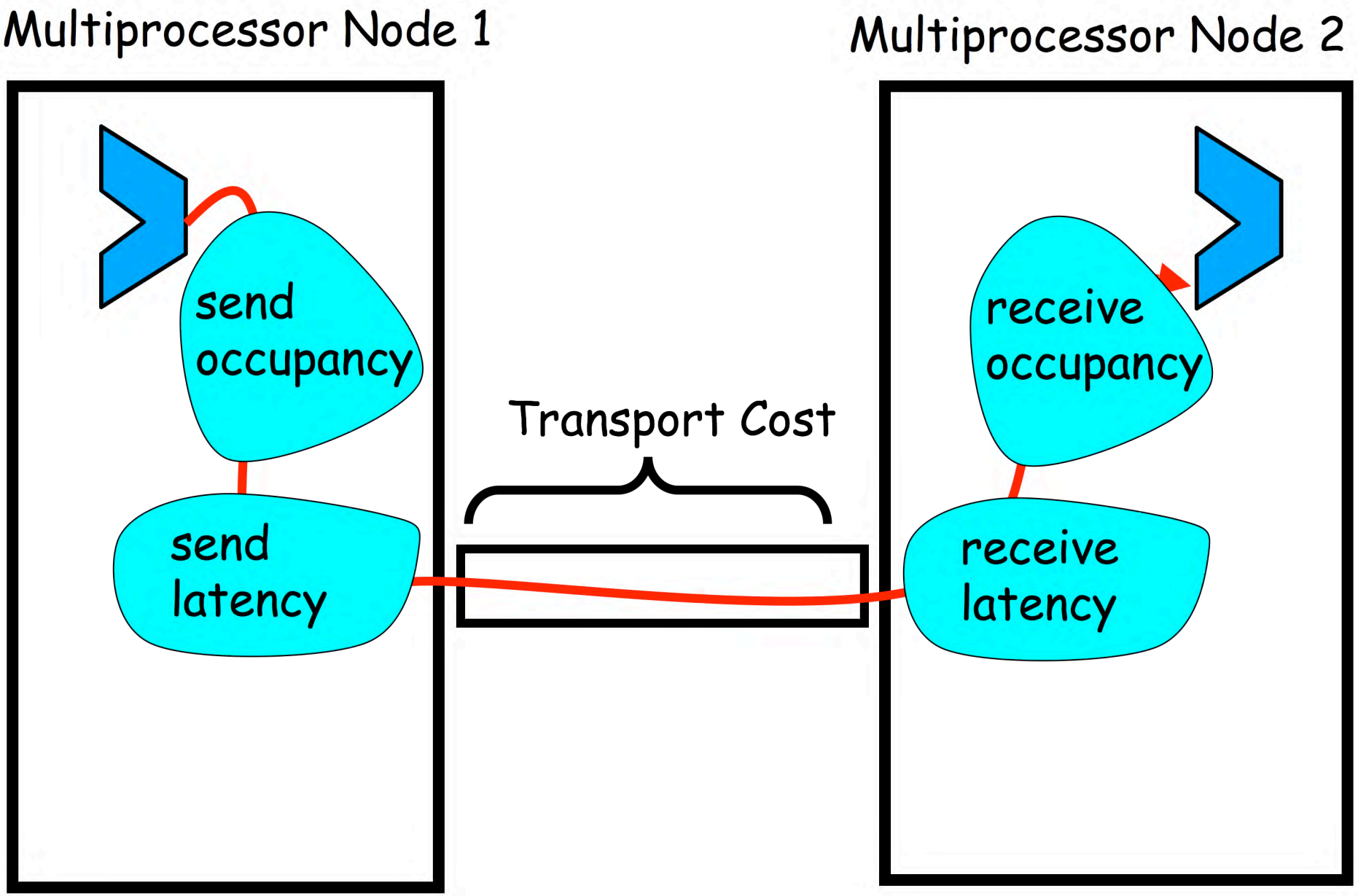
Why communication is expensive on multicore



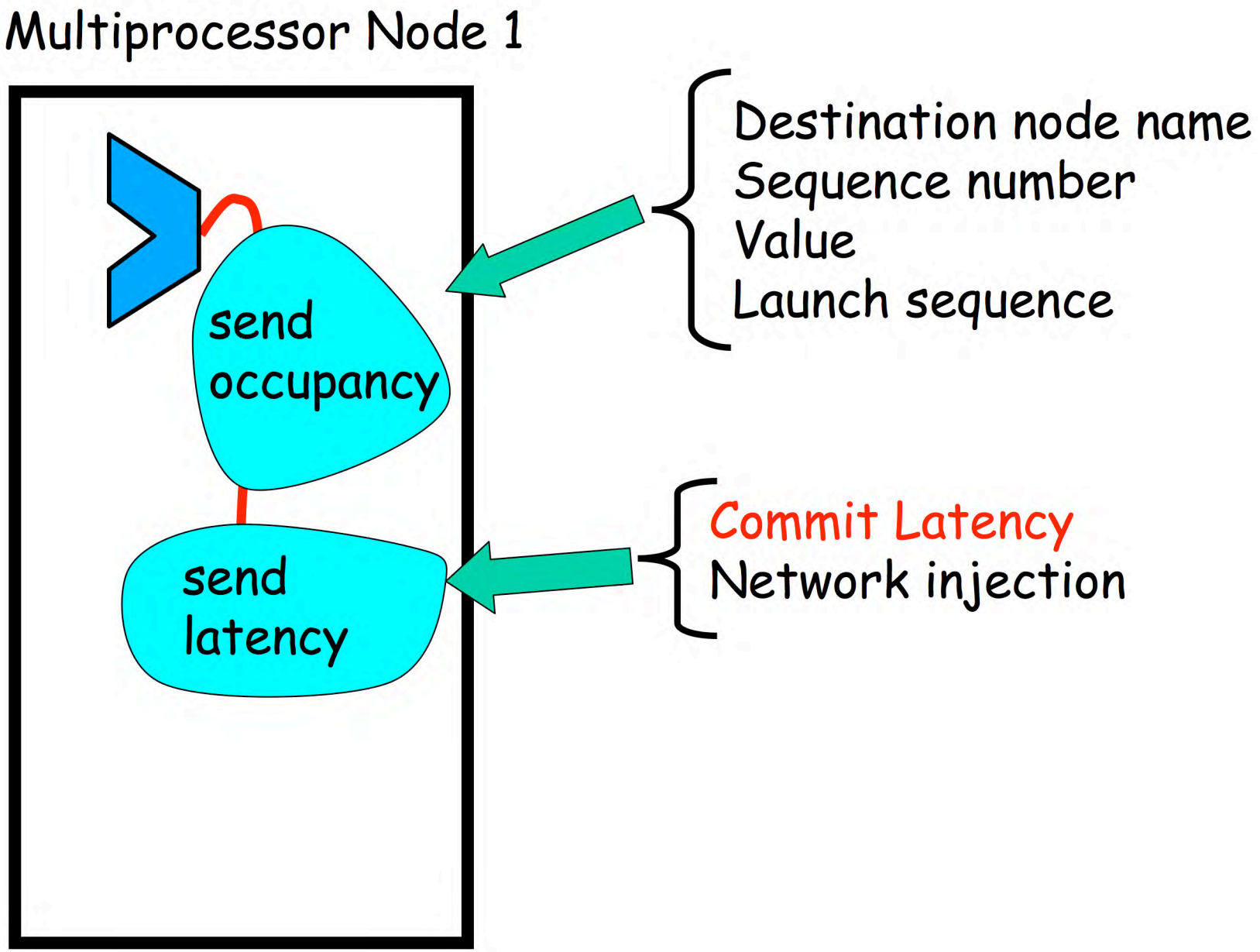
Why communication is expensive on multicore



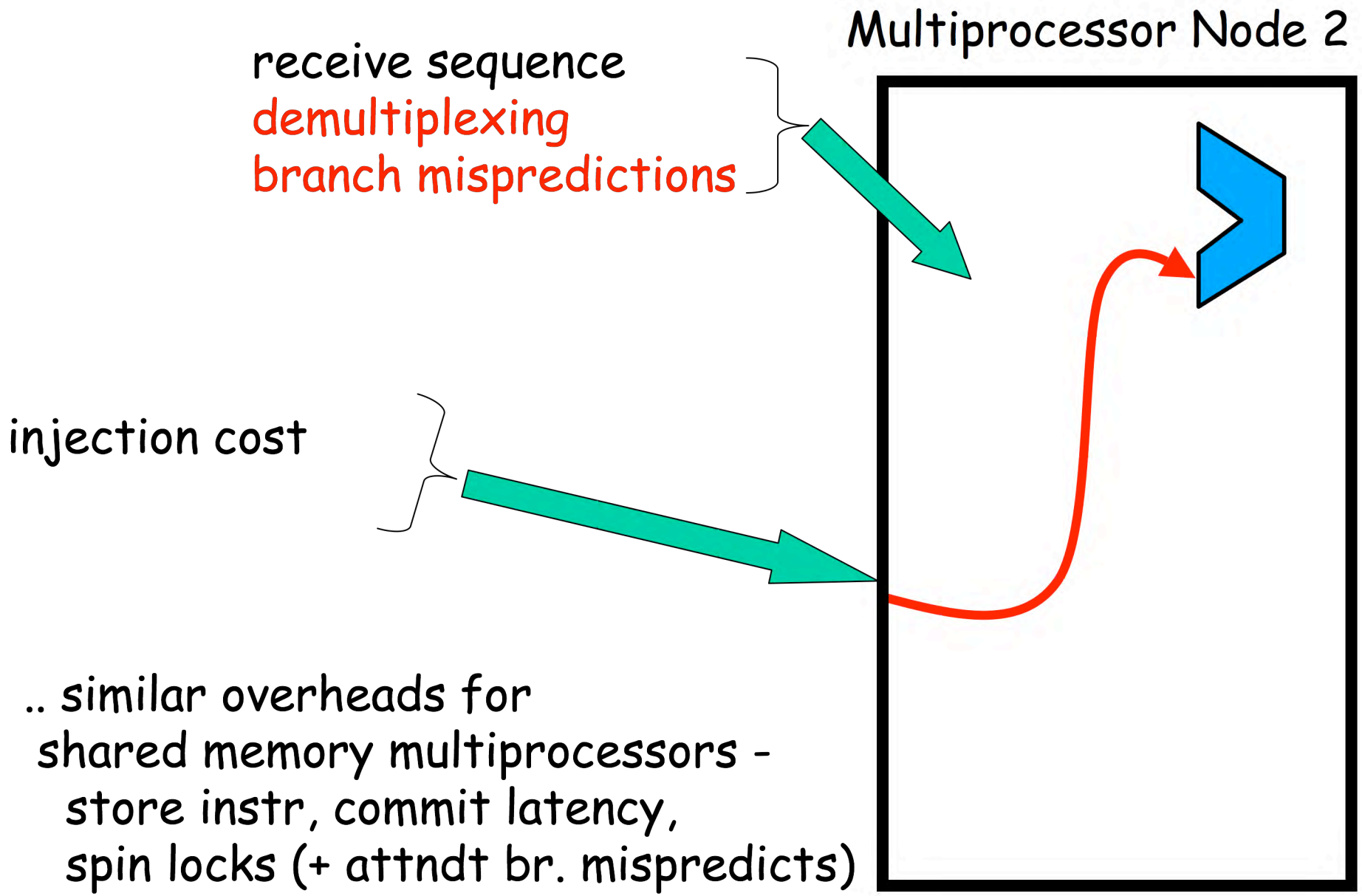
Why communication is expensive on multicore



Multiprocessor SON Operand Routing

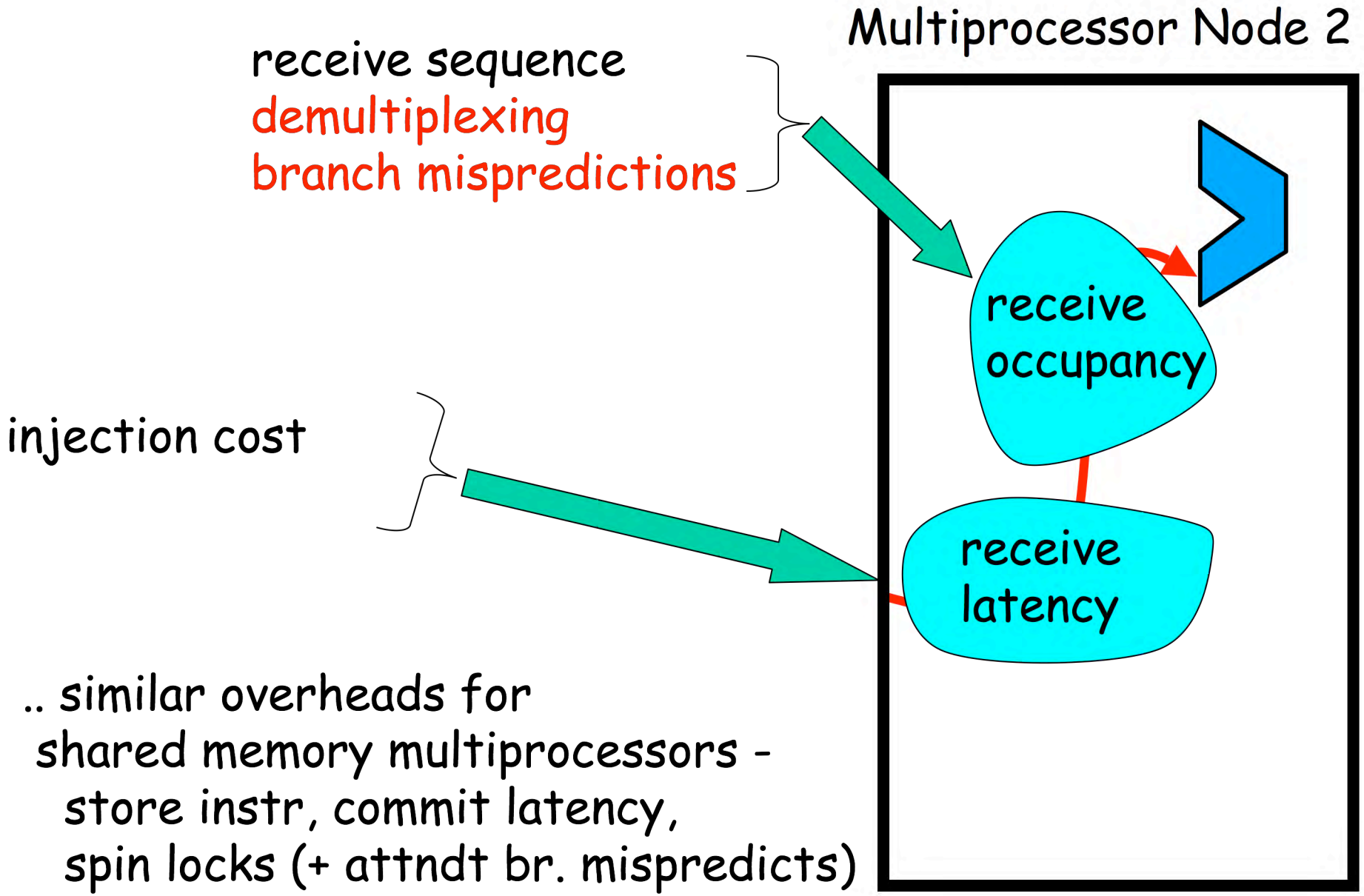


Multiprocessor SON Operand Routing



.. similar overheads for
shared memory multiprocessors -
store instr, commit latency,
spin locks (+ attndt br. mispredicts)

Multiprocessor SON Operand Routing



.. similar overheads for
shared memory multiprocessors -
store instr, commit latency,
spin locks (+ attndt br. mispredicts)

Defining a figure of merit for scalar operand networks

5-tuple <SO, SL, NHL, RL, RO>:

Send Occupancy

Send Latency

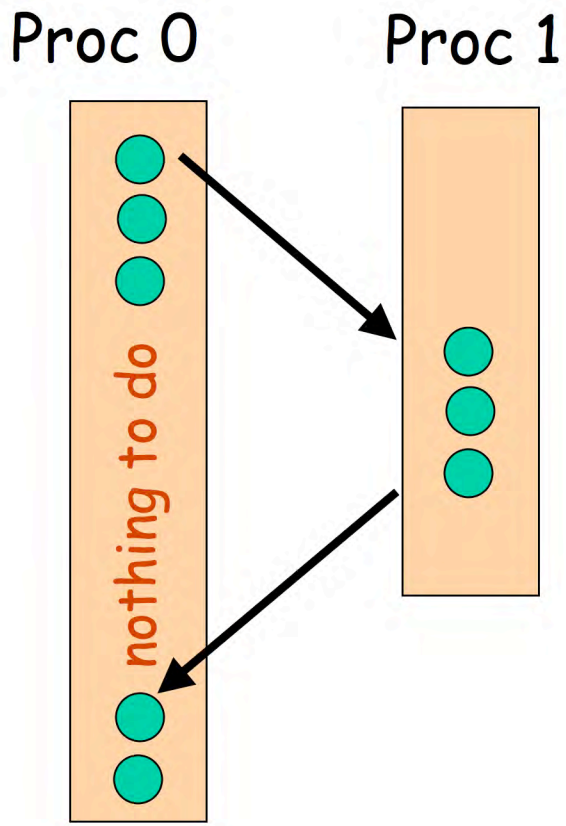
Network Hop Latency

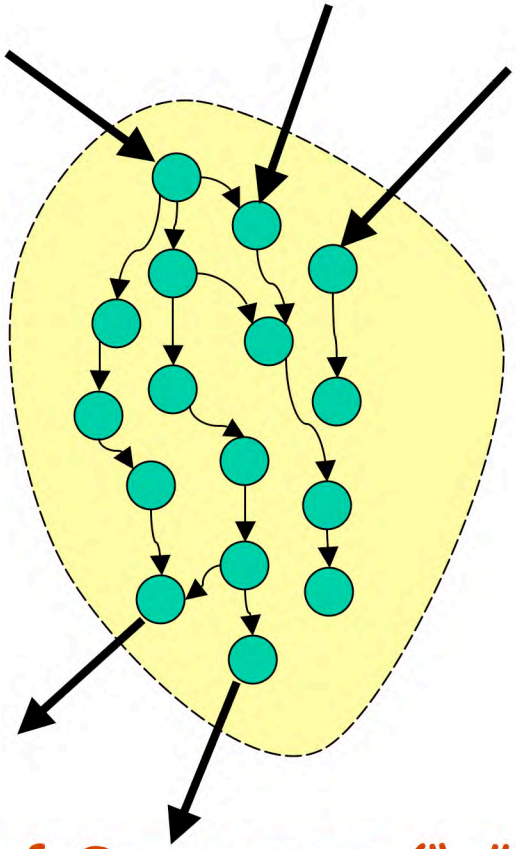
Receive Latency

Receive Occupancy

We can use this metric to quantitatively differentiate SONs from existing multiprocessor networks…

Tip: Ordering follows timing of message from sender to receiver





Impact of Occupancy ("o" = so+ro)

if (o * "surface area" > "volume")

→ not worth it to offload:
overhead too high
(parallelism too fine-grained)

Impact of Latency

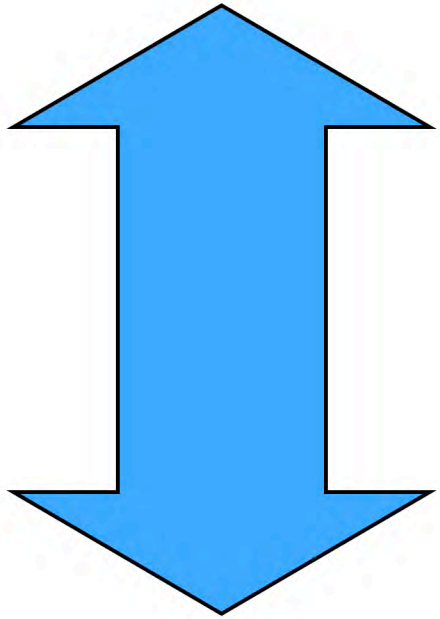
The lower the latency,
the less work needed to keep
myself busy waiting for answer

→ not worth it to offload:
could have done it myself faster
(not enough parallelism to hide latency)

The interesting region

Power4
(on-chip)

<2, 14, 0, 14,4>



Superscalar
(not scalable)

< 0, 0, 0, 0, 0>

Tiled Microprocessors (or "Tiled Multicore")

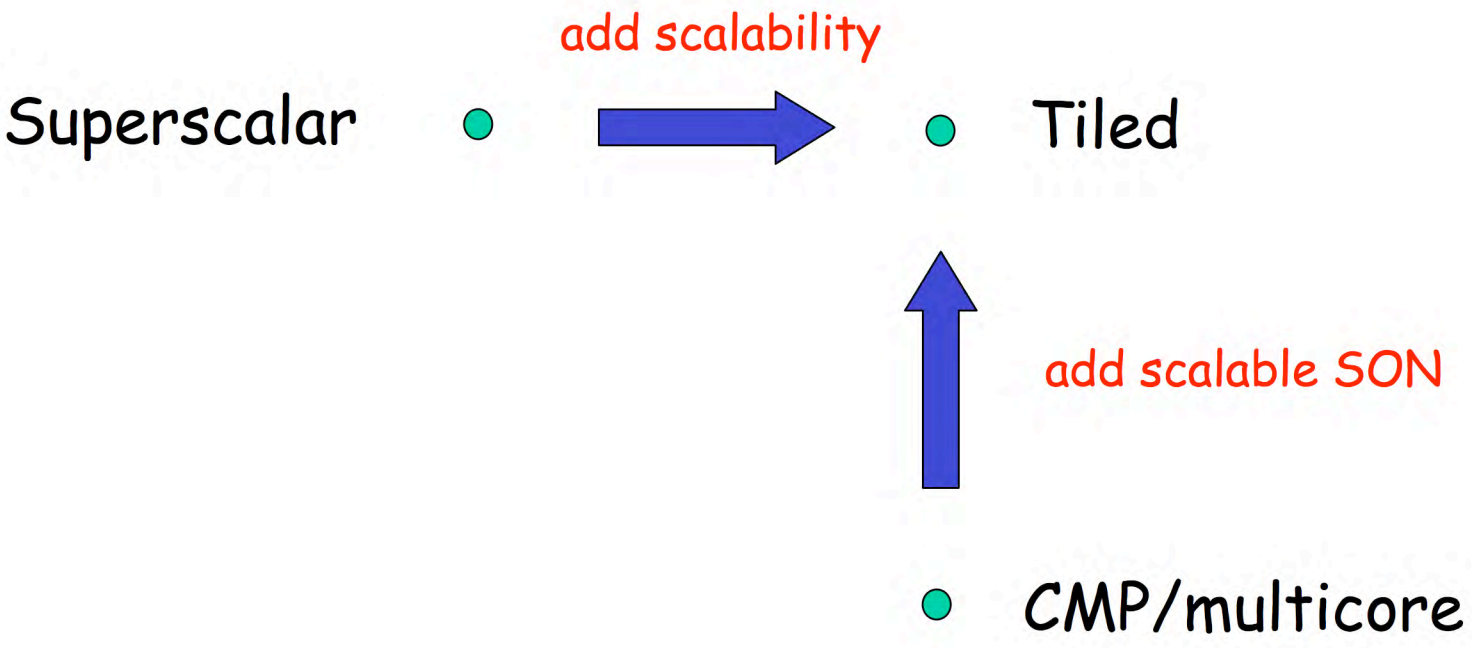
(w/ scalable SON)

	Superscalar	Multicore	Tiled Multicore
PE-PE communication	Free	Expensive	Cheap
exploitation of parallelism	Implicit	Explicit	Both
scalable	No	Yes	Yes
power efficient	No	Yes	Yes

Tiled Microprocessors (or "Tiled Multicore")

	Superscalar	Multicore	Tiled Multicore
Alu-Alu communication	Free	Expensive	Cheap
exploitation of parallelism	Implicit	Explicit	Both
scalable	No	Yes	Yes
power efficient	No	Yes	Yes

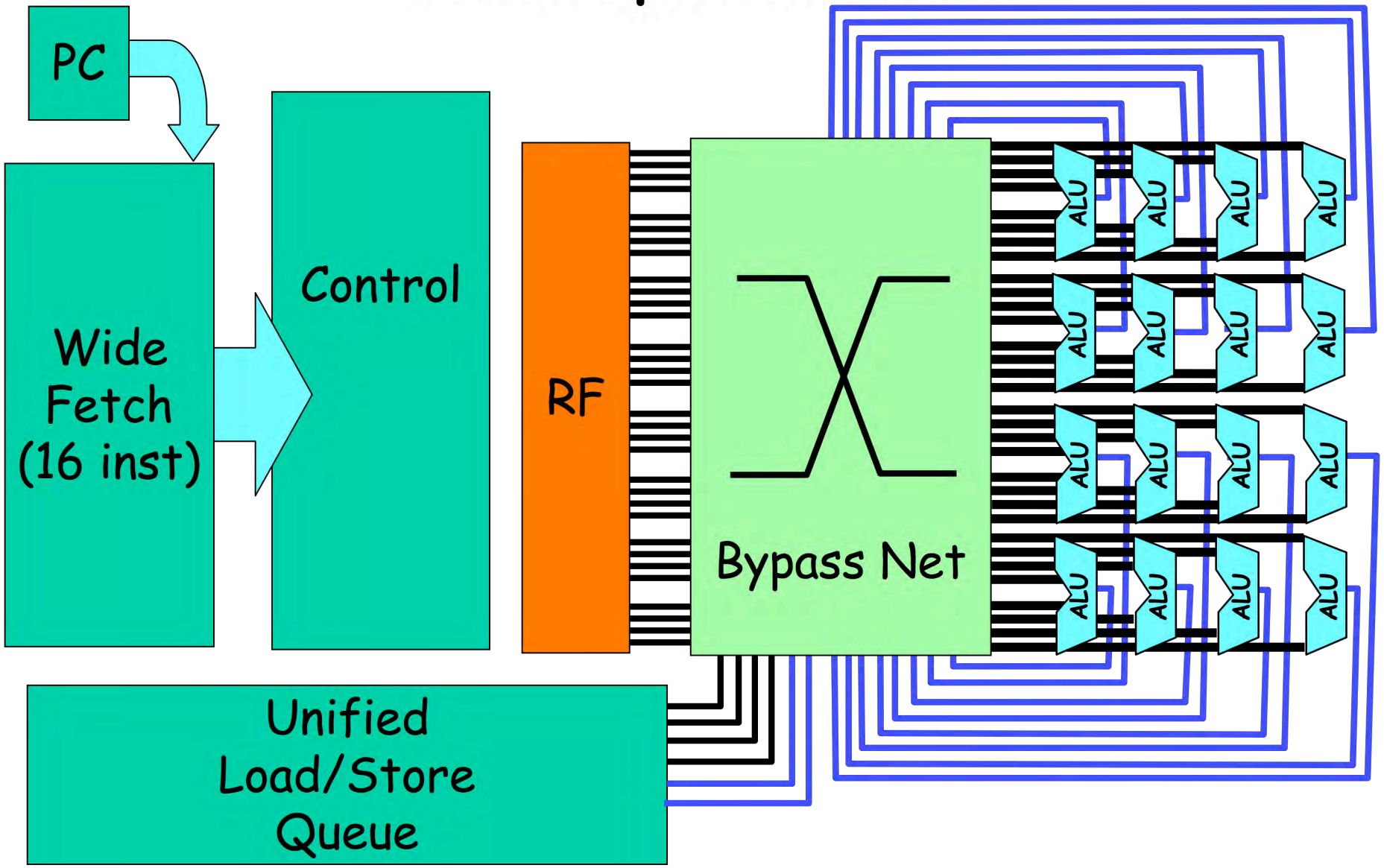
Transforming from multicore or superscalar to tiled



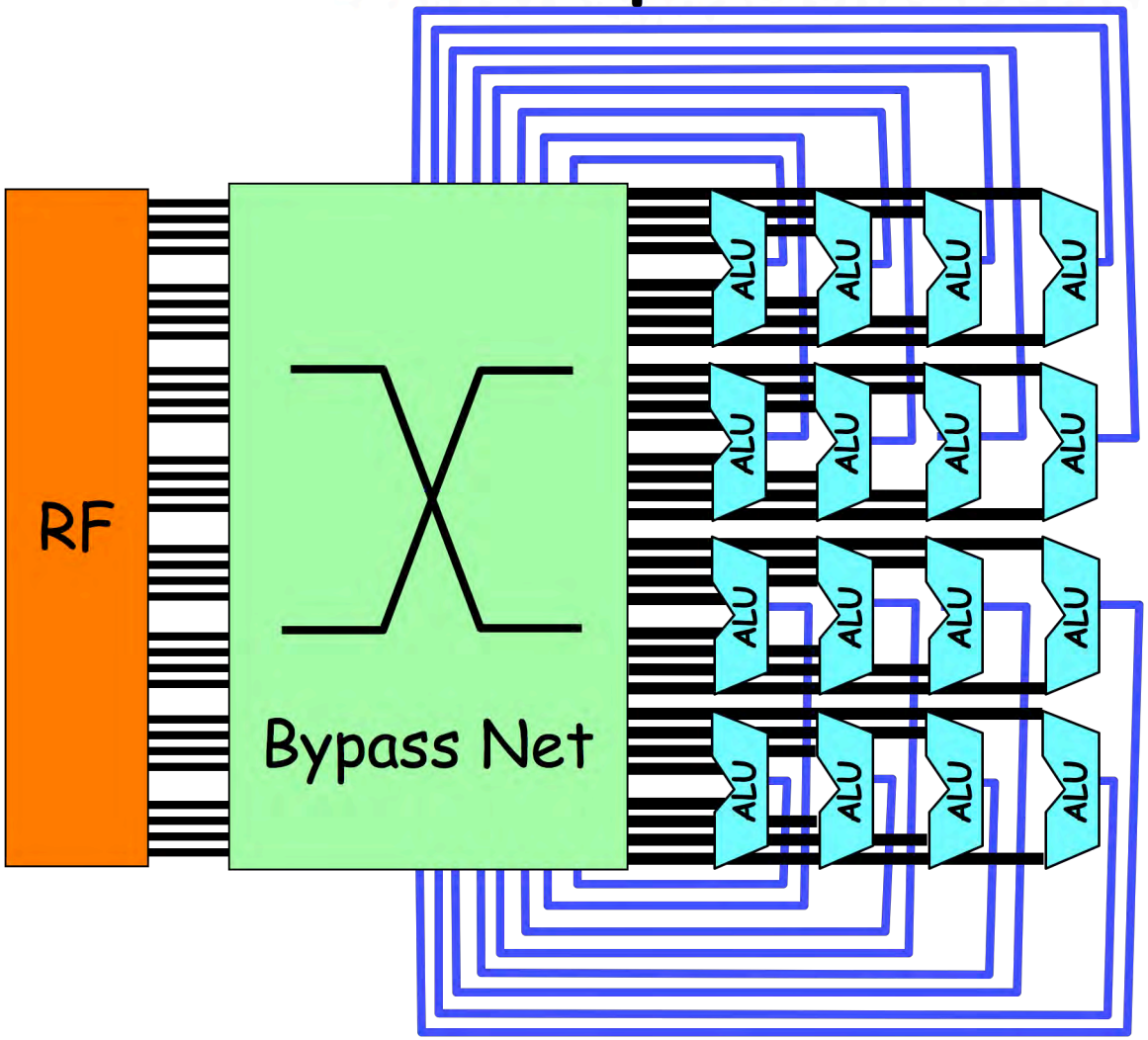
The interesting region

Power4 (on-chip)	<2, 14, 0, 14,4>
Raw	< 0, 0, 1, 2, 0>
Tiled "Famous Brand 2"	< 0, 0, 1, 0, 0>
Superscalar (not scalable)	< 0, 0, 0, 0, 0>

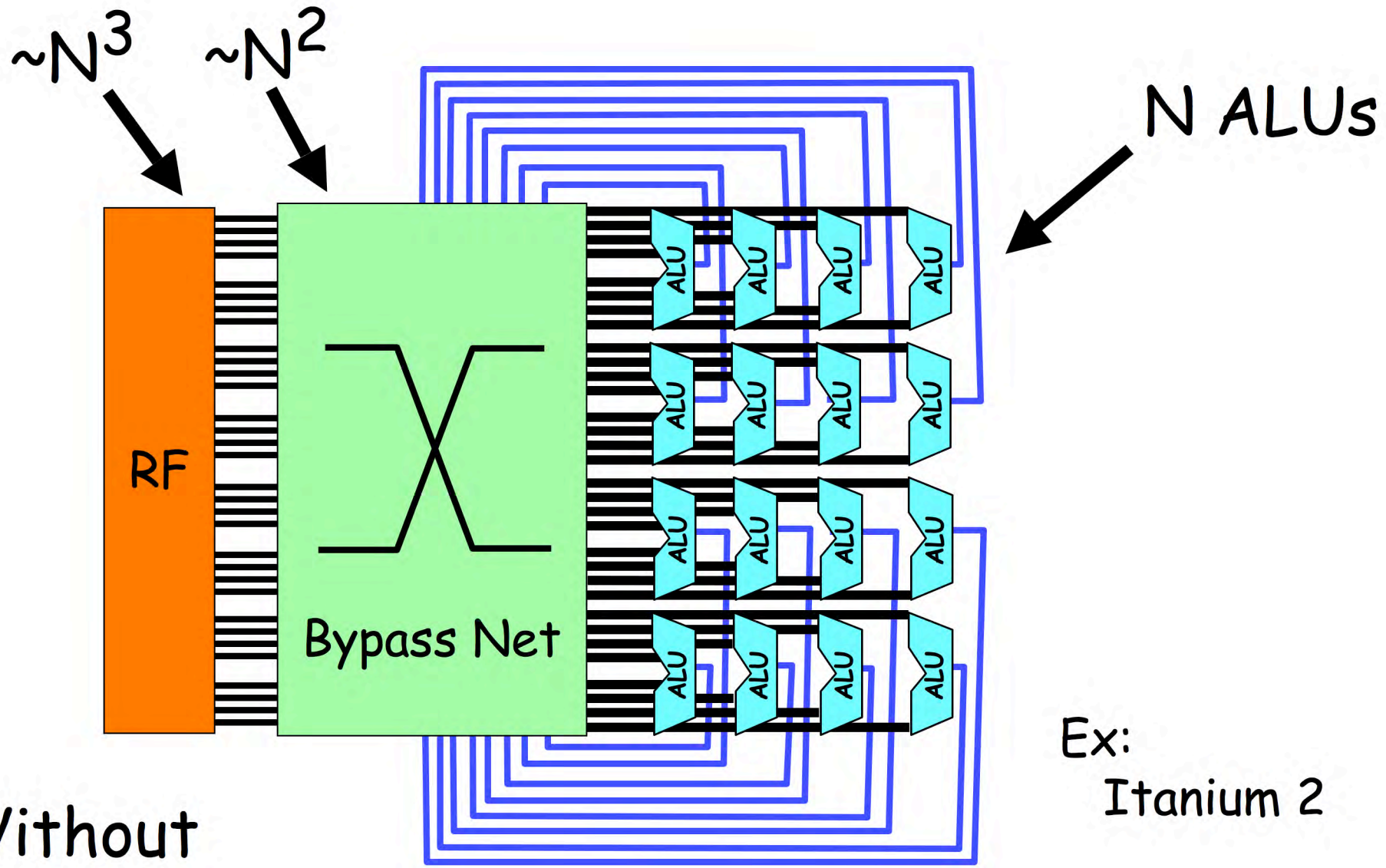
Scalability Problems in Wide Issue Microprocessors



Scalability Problems in Wide Issue Microprocessors



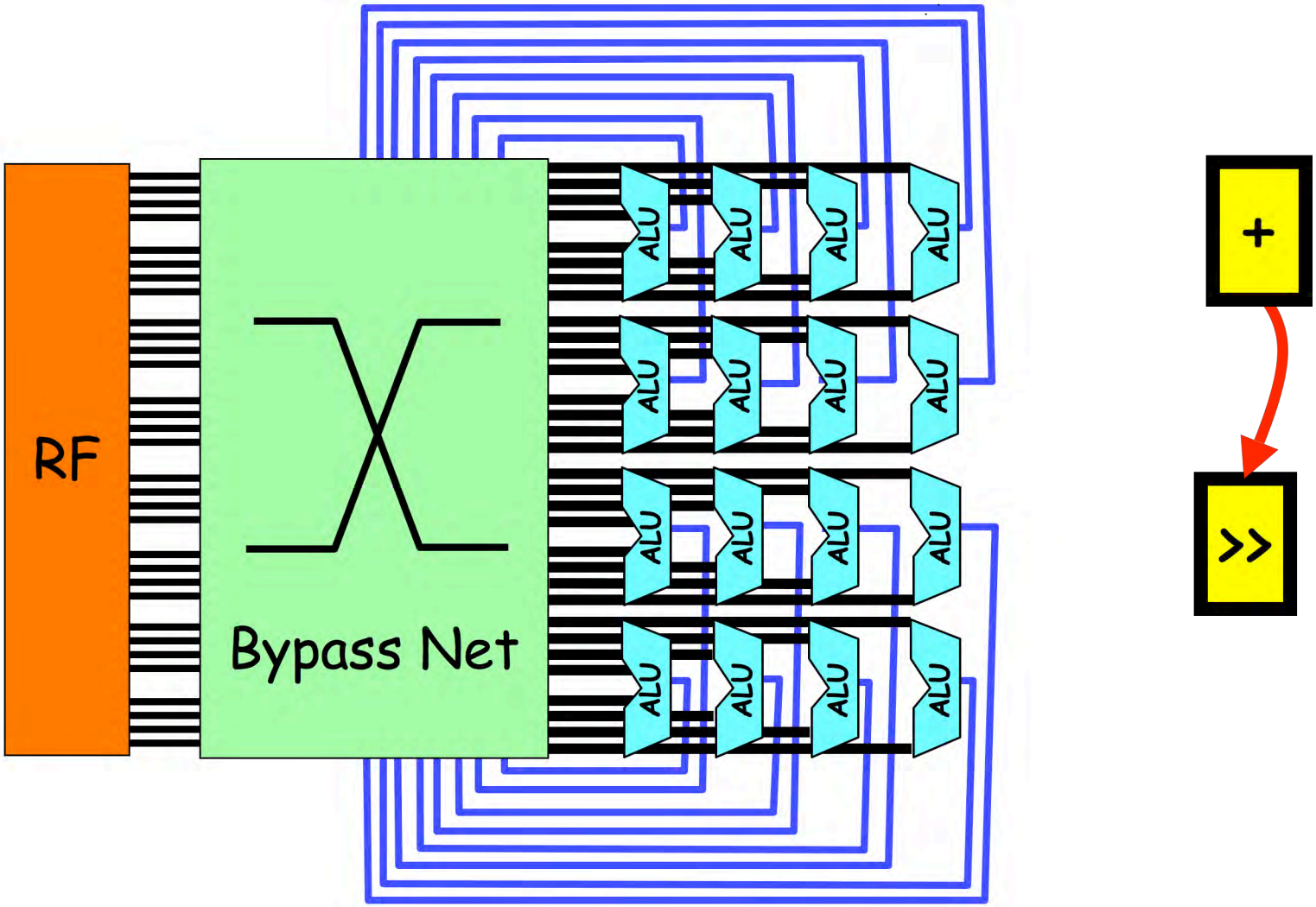
Area and Frequency Scalability Problems



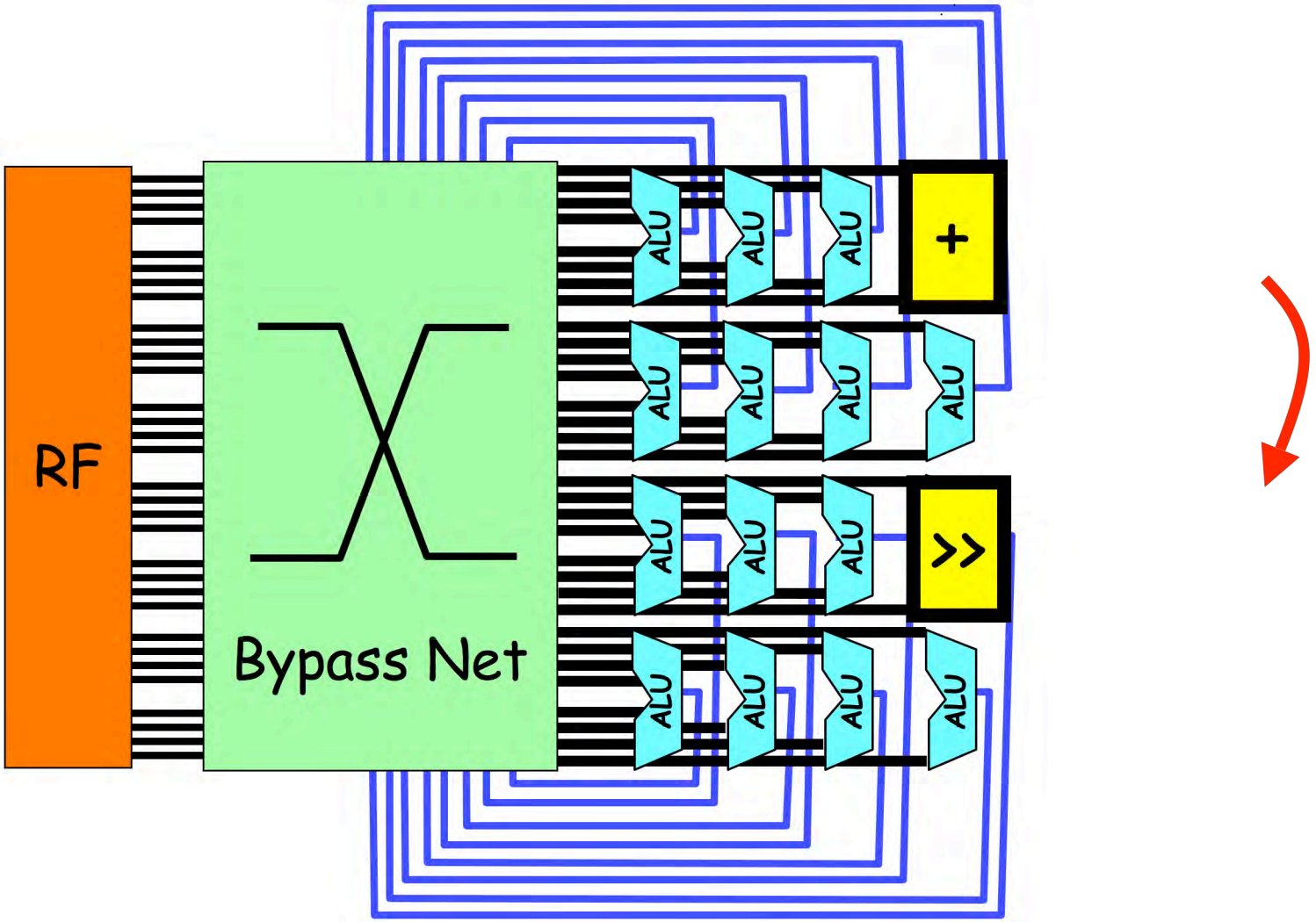
Ex:
Itanium 2

Without modification, freq decreases linearly or worse.

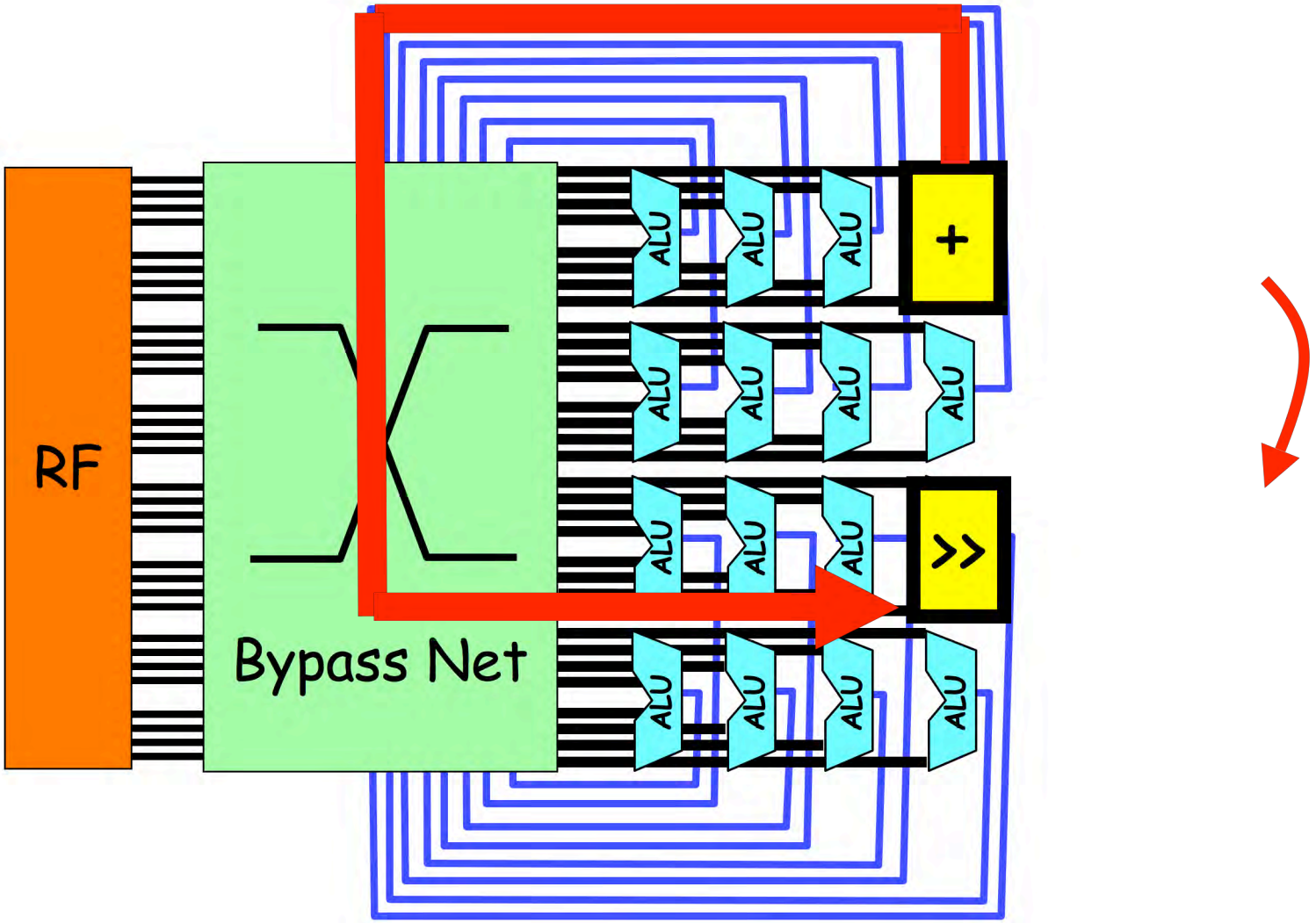
Operand Routing is Global



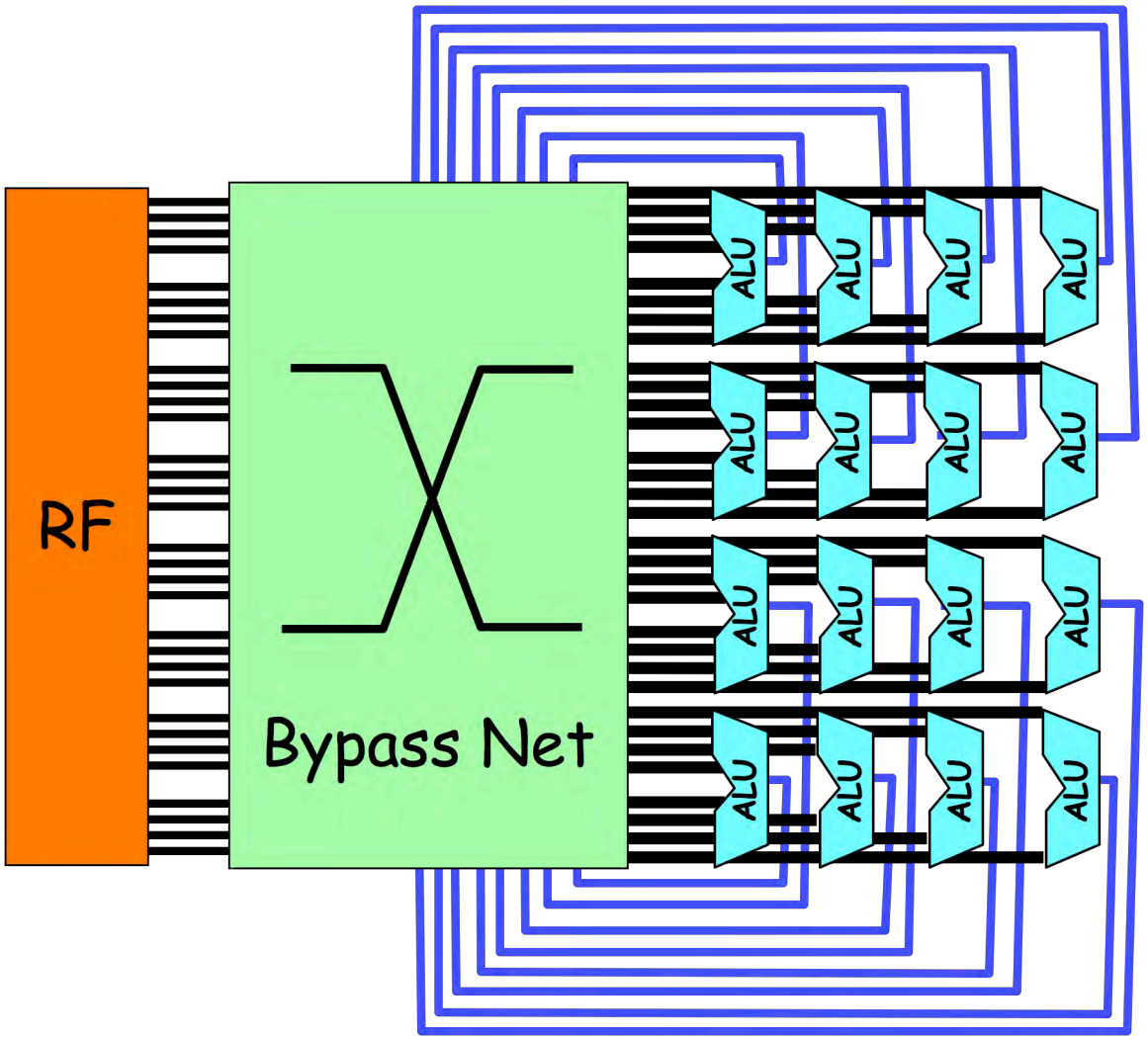
Operand Routing is Global



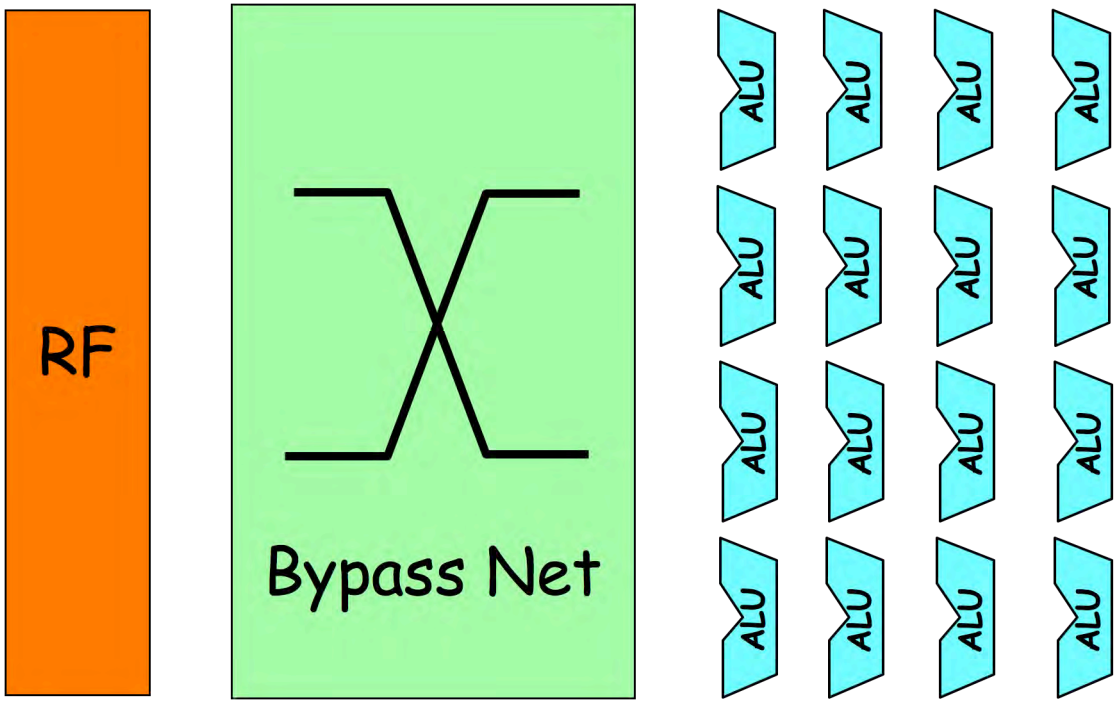
Operand Routing is Global



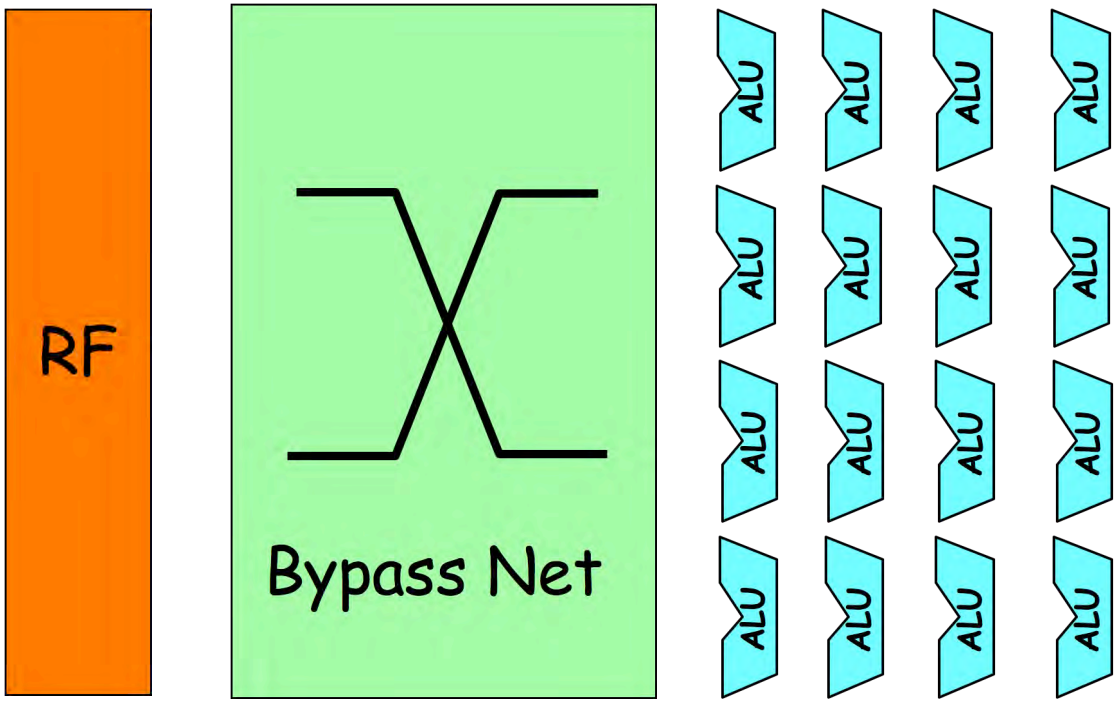
Idea: Make Operand Routing Local



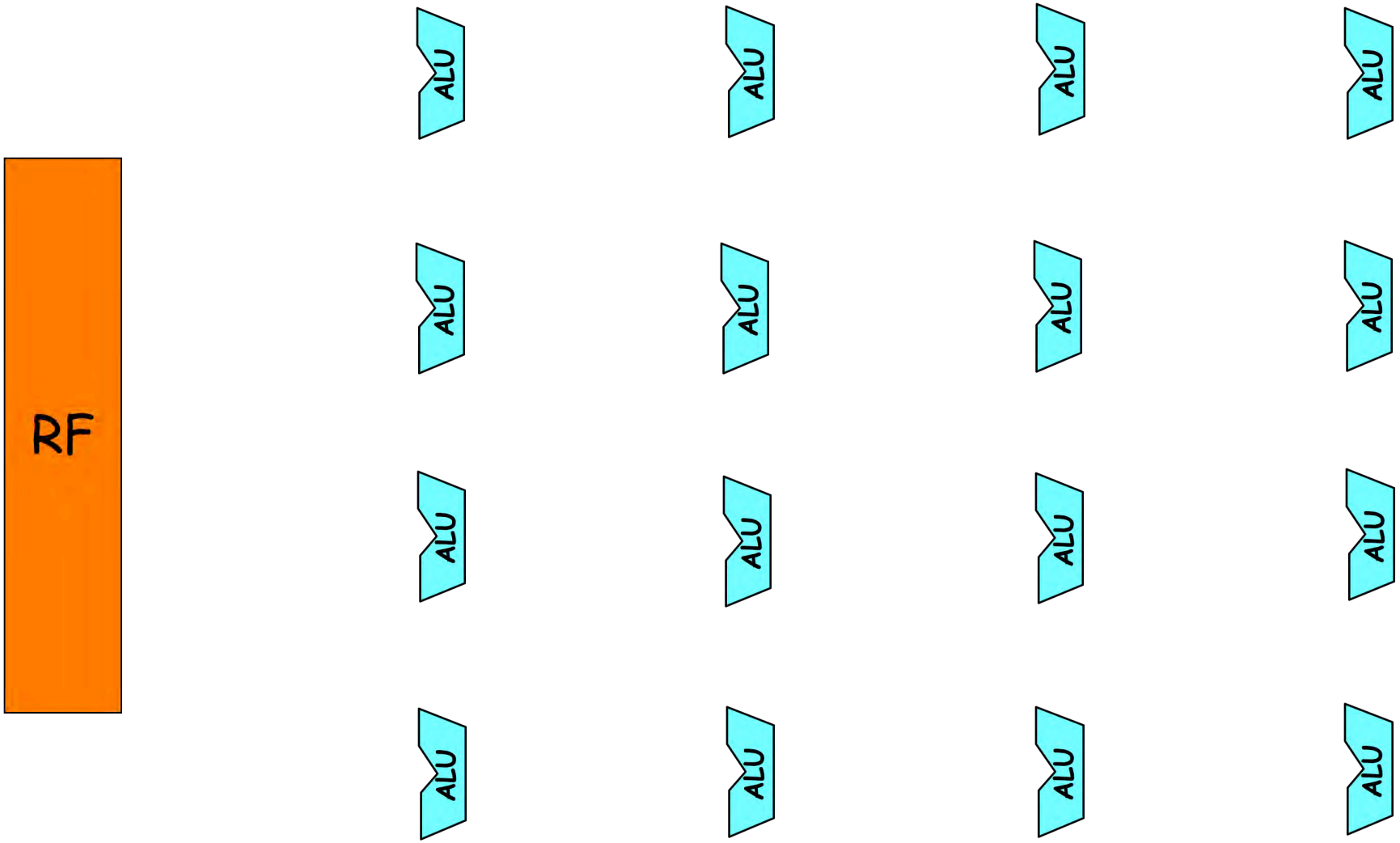
Idea: Make Operand Routing Local



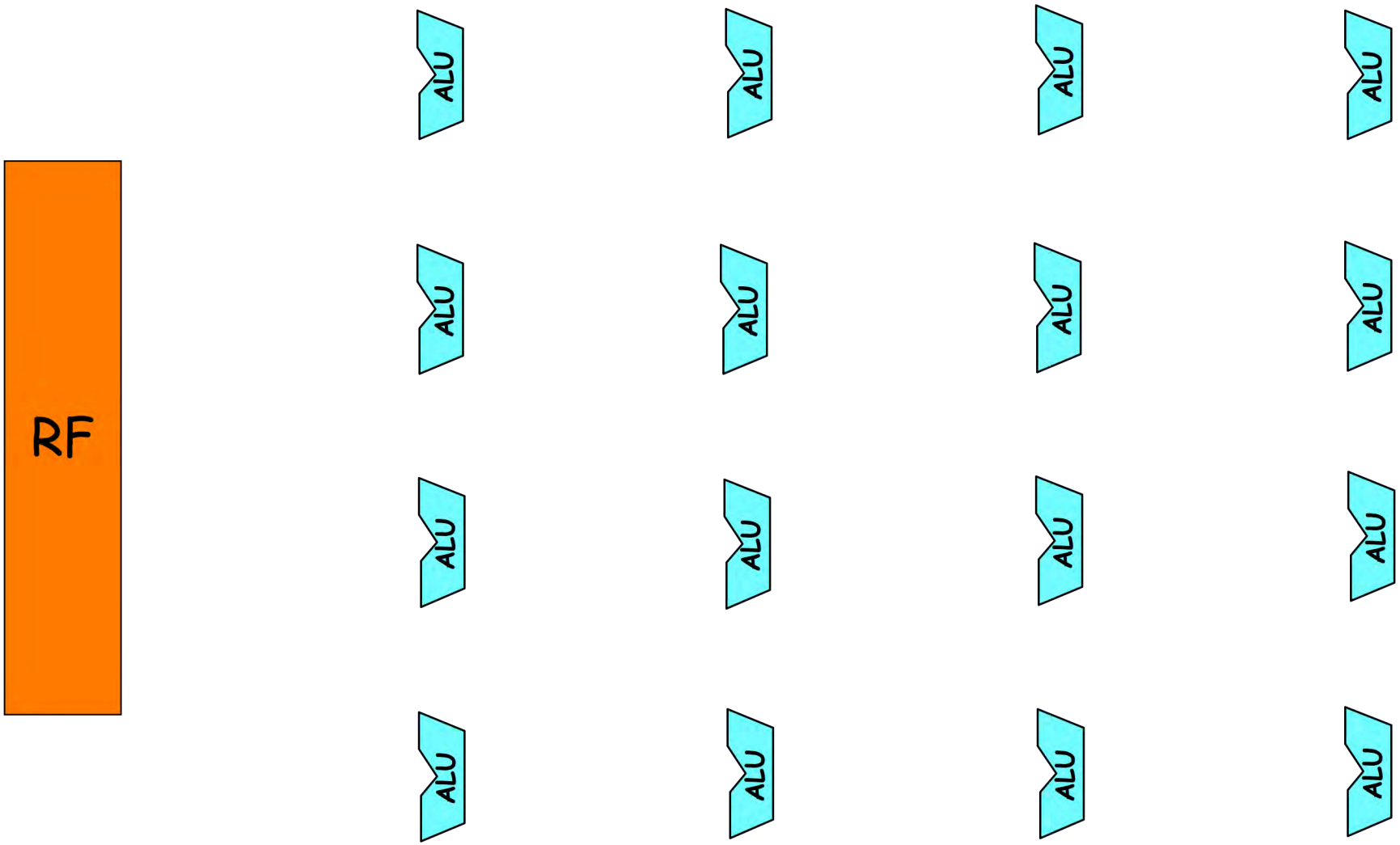
Idea: Exploit Locality



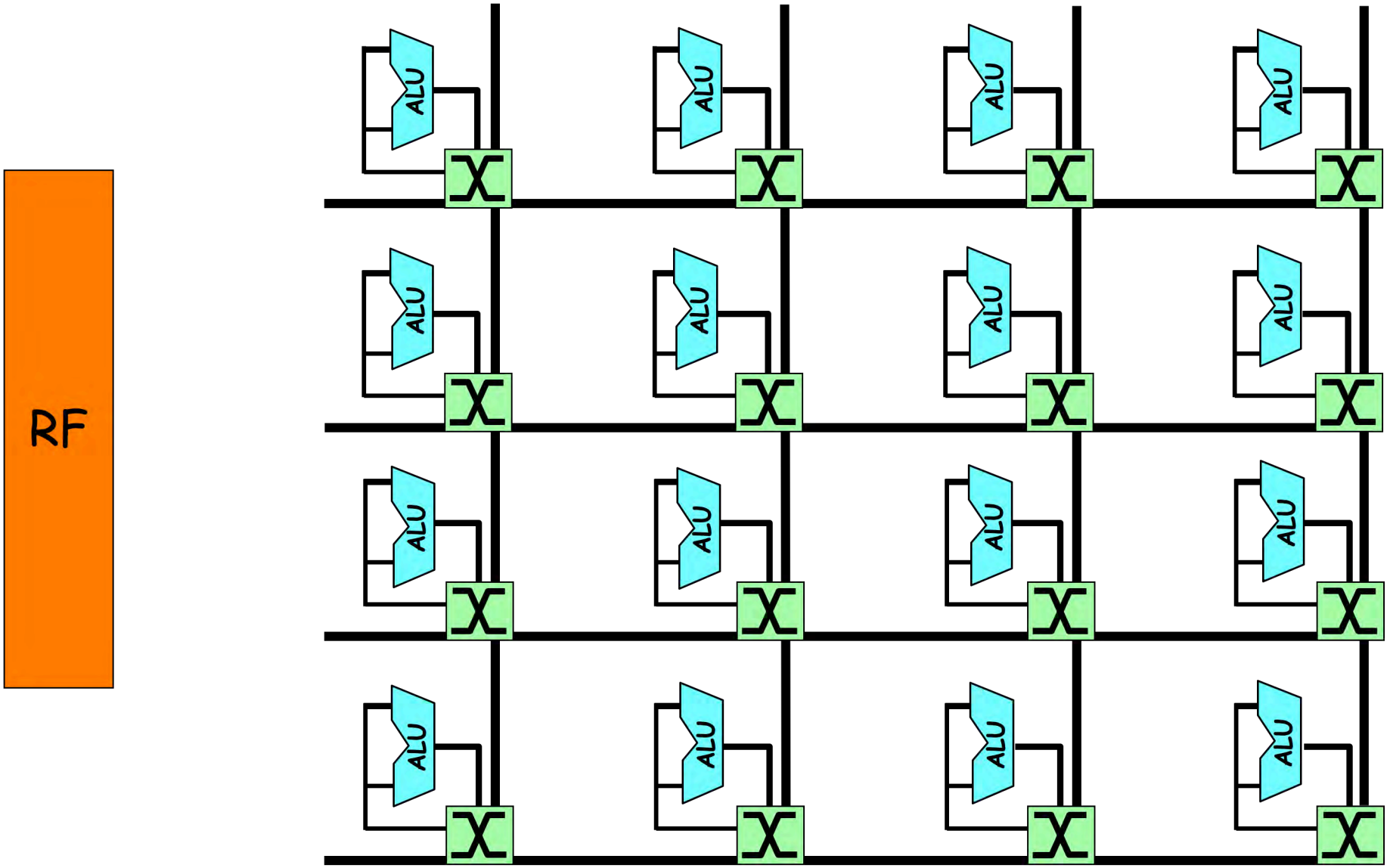
Idea: Exploit Locality



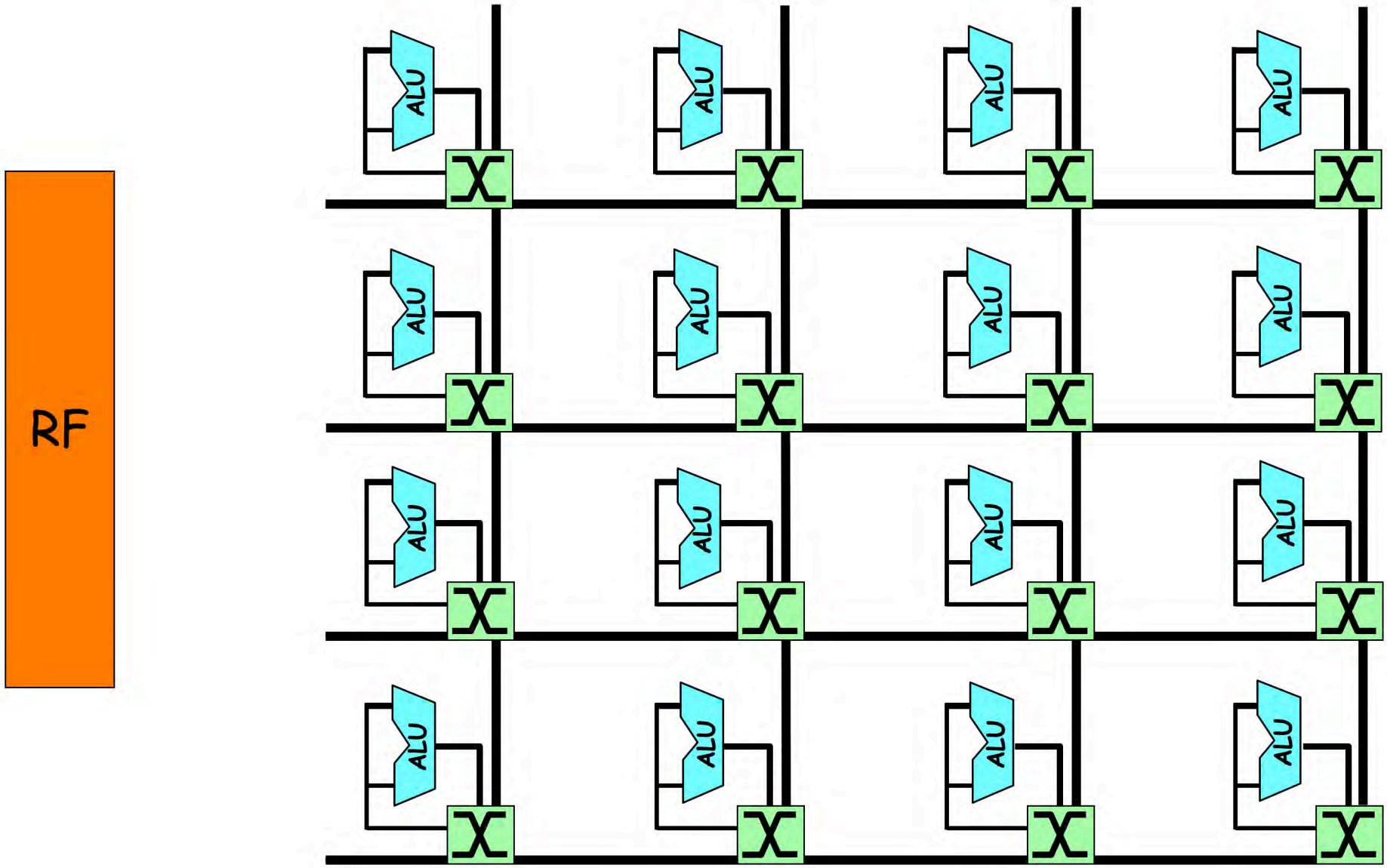
Replace the crossbar with a point-to-point, pipelined, routed scalar operand network.



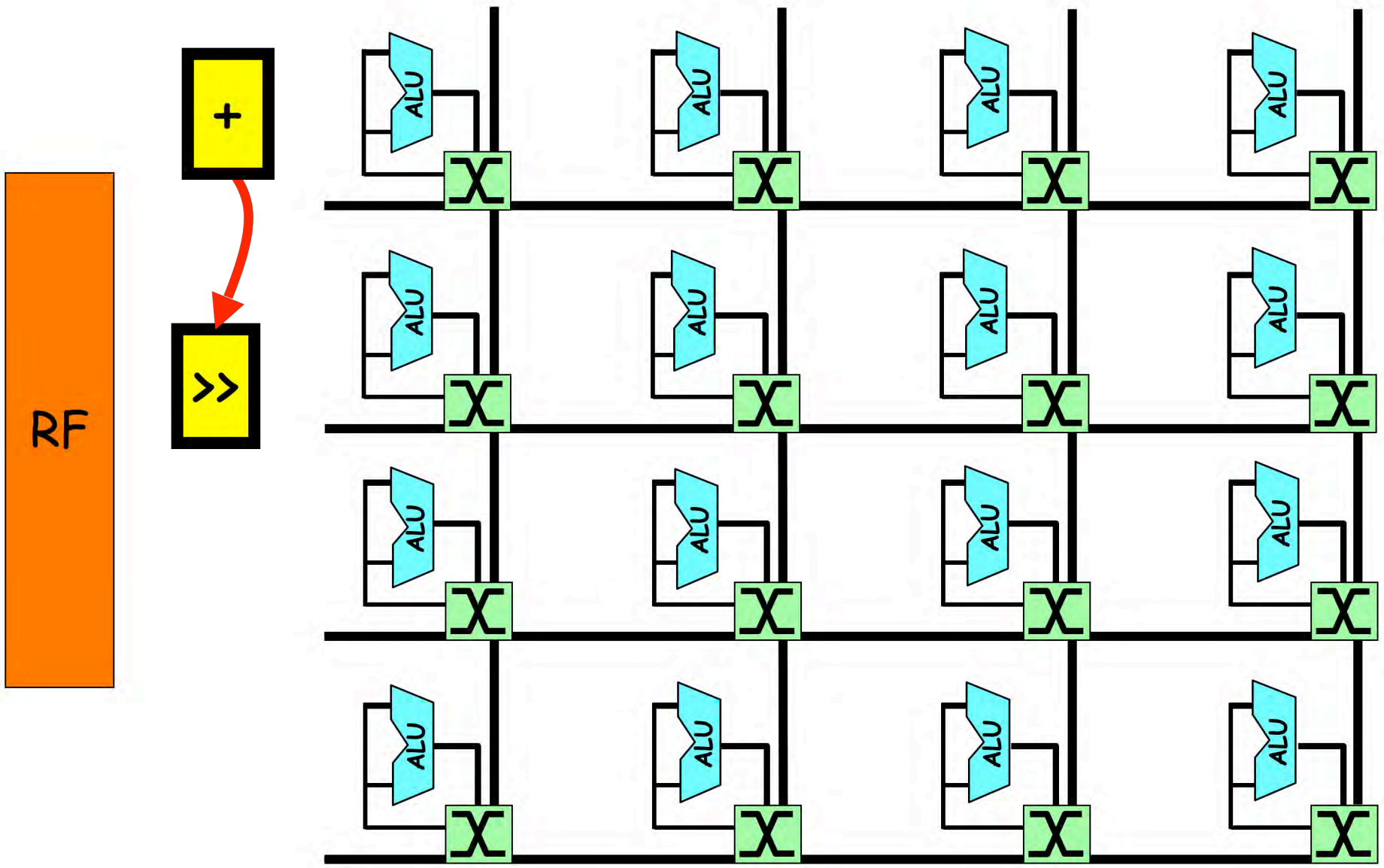
Replace the crossbar with a point-to-point, pipelined, routed scalar operand network.



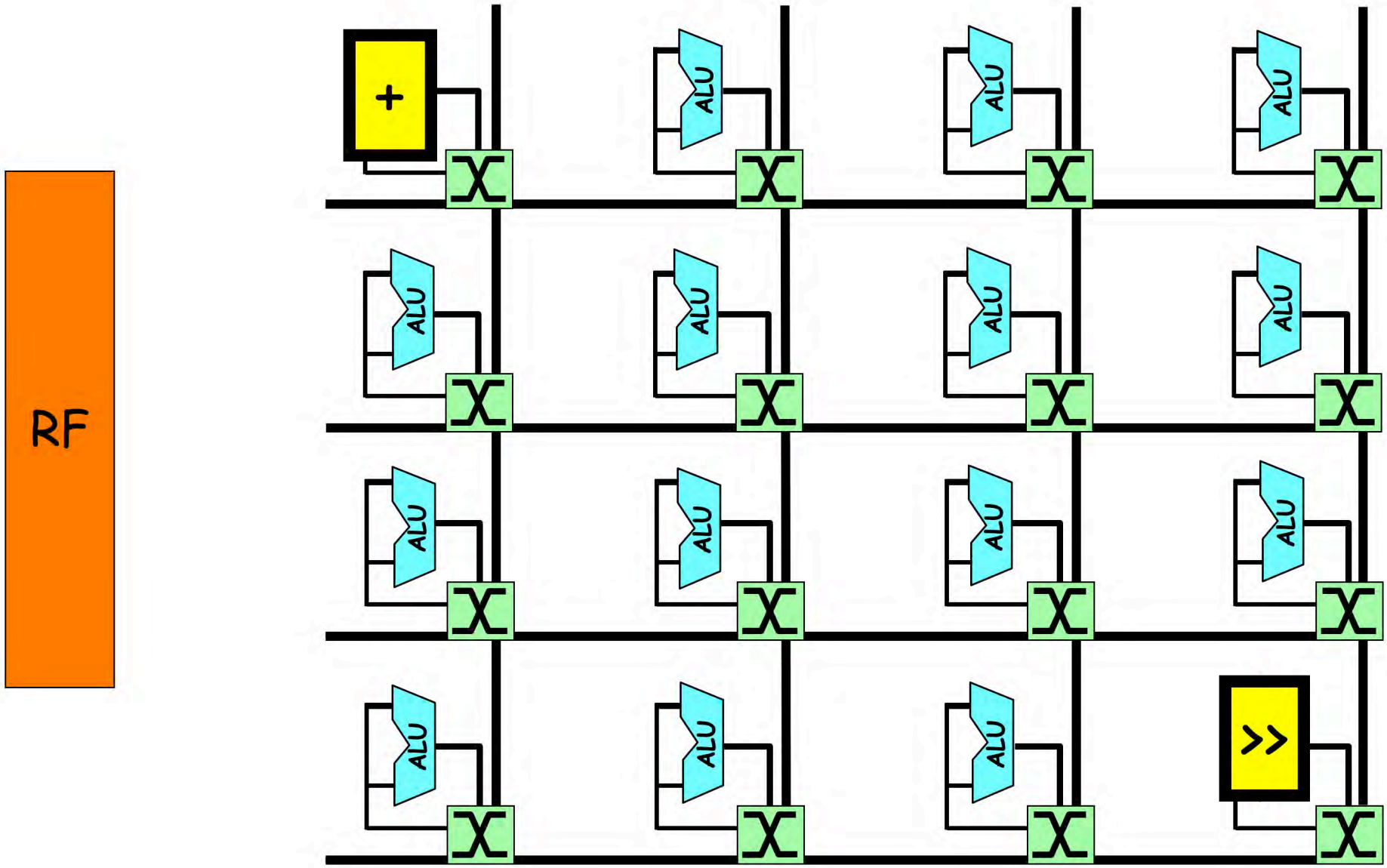
Replace the crossbar with a point-to-point, pipelined, routed scalar operand network.



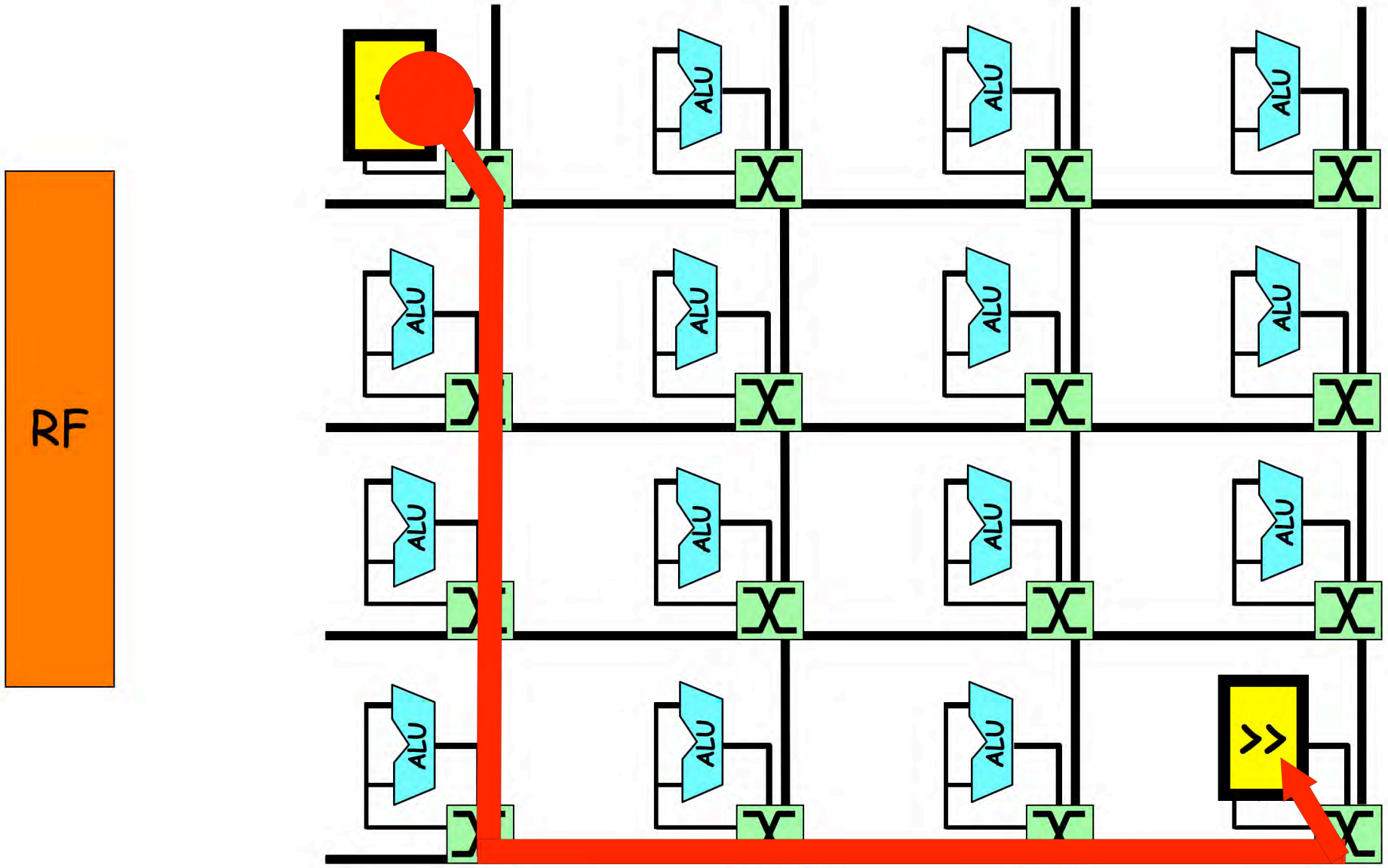
Replace the crossbar with a point-to-point, pipelined, routed scalar operand network.



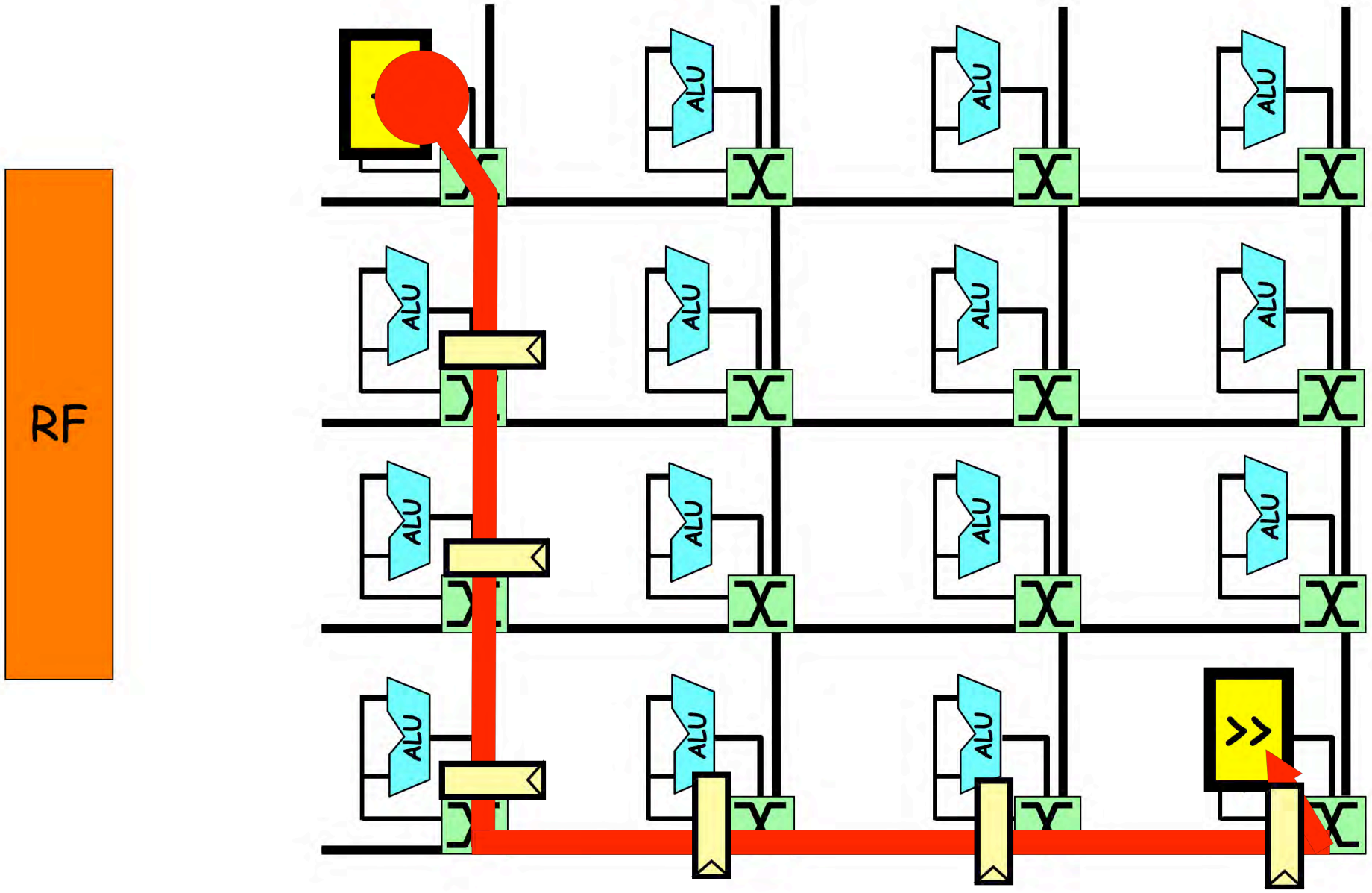
Replace the crossbar with a point-to-point, pipelined, routed scalar operand network.



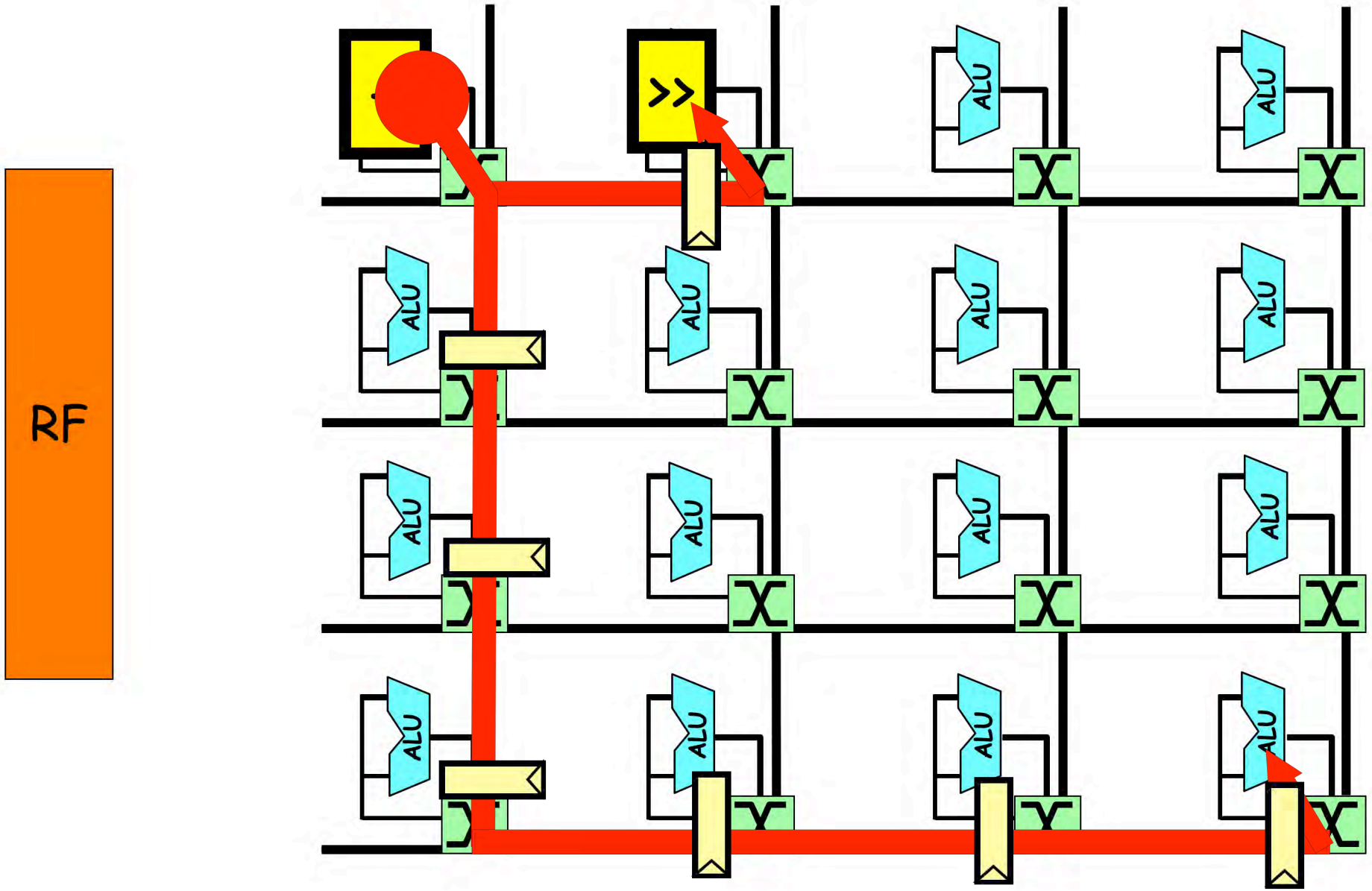
Replace the crossbar with a point-to-point, pipelined, routed scalar operand network.



Replace the crossbar with a point-to-point, pipelined, routed scalar operand network.



Replace the crossbar with a point-to-point, pipelined, routed scalar operand network.



Operand Transport Scaling - Bandwidth and Area

For N ALUs and $N^{\frac{1}{2}}$ bisection bandwidth:

as in conventional superscalar

	Un-pipelined crossbar bypass	Point-to-Point Routed Mesh Network
Local BW	~ $N^{\frac{1}{2}}$	~ N
Area	~ N^2	~ N

Operand Transport Scaling - Bandwidth and Area

For N ALUs and $N^{\frac{1}{2}}$ bisection bandwidth:

as in conventional superscalar

	Un-pipelined crossbar bypass	Point-to-Point Routed Mesh Network
Local BW	~ $N^{\frac{1}{2}}$	~ N
Area	~ N^2	~ N

Scales as 2-D VLSI

Operand Transport Scaling - Bandwidth and Area

For N ALUs and $N^{\frac{1}{2}}$ bisection bandwidth:

as in conventional superscalar

	Un-pipelined crossbar bypass	Point-to-Point Routed Mesh Network
Local BW	~ $N^{\frac{1}{2}}$	~ N
Area	~ N^2	~ N

Scales as 2-D VLSI

We can route more operands per unit time if we are able to map communicating instructions nearby.

Operand Transport Scaling - Latency

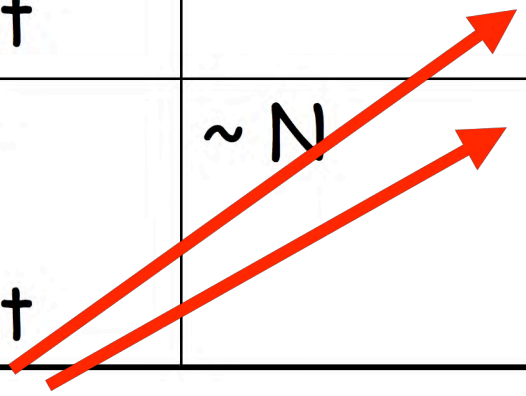
Time for operand to travel between instructions mapped to different ALUs.

	Un-pipelined crossbar	Point-to-Point Routed Mesh Network
Non-local Placement	~ N	~ $N^{\frac{1}{2}}$
Locality-Driven Placement	~ N	~ 1

Operand Transport Scaling - Latency

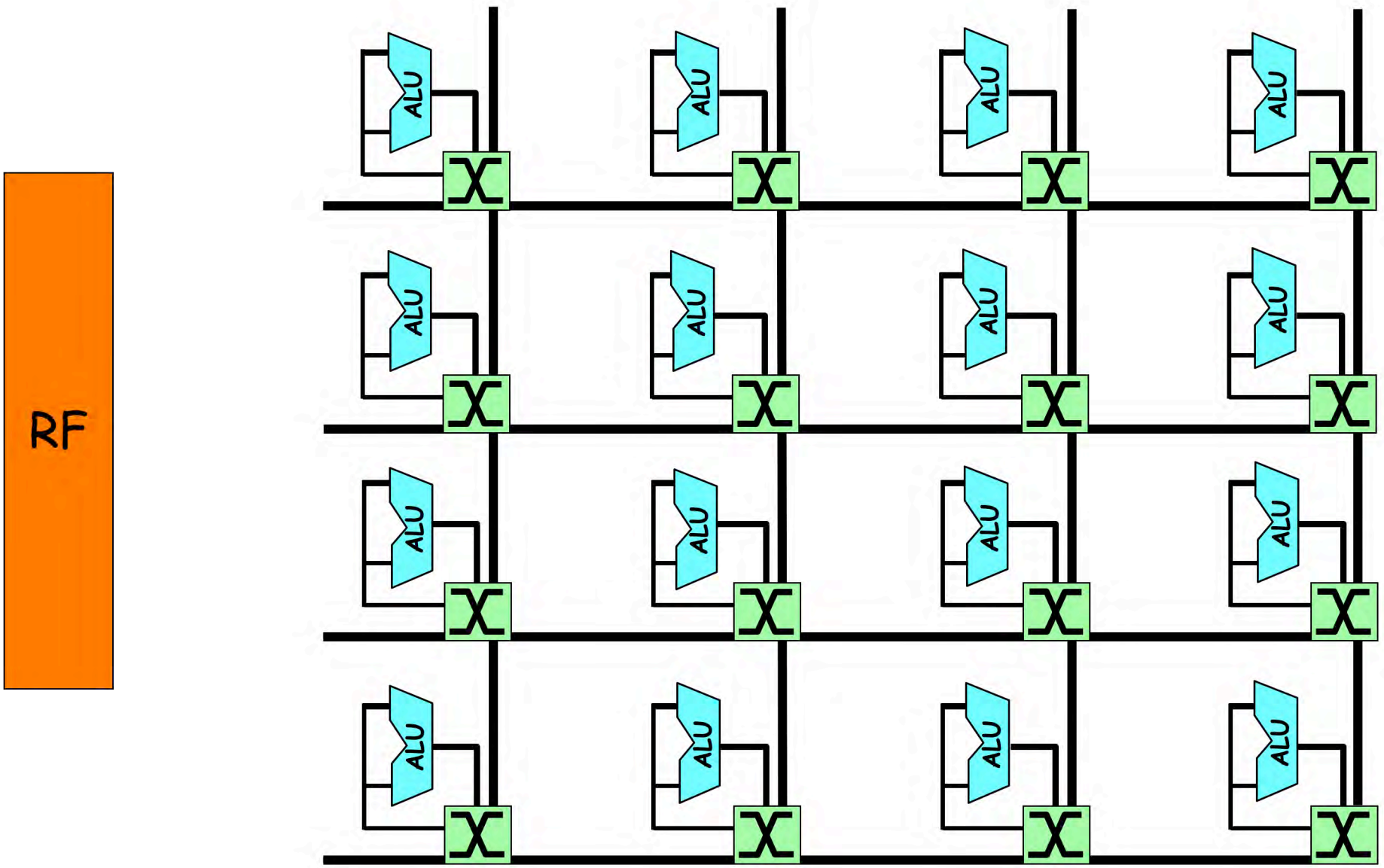
Time for operand to travel between instructions mapped to different ALUs.

	Un-pipelined crossbar	Point-to-Point Routed Mesh Network
Non-local Placement	~ N	~ $N^{\frac{1}{2}}$
Locality-Driven Placement	~ N	~ 1

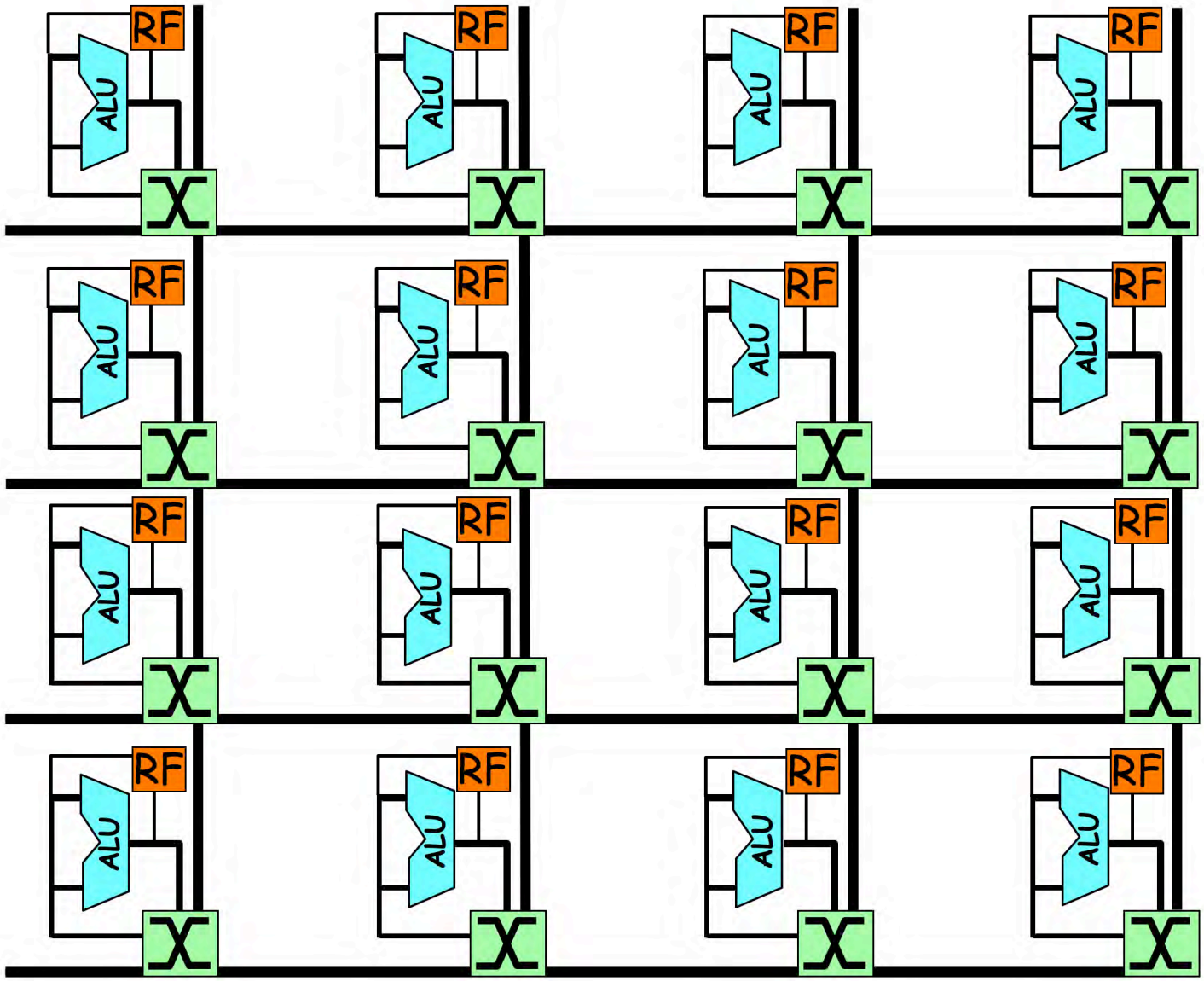


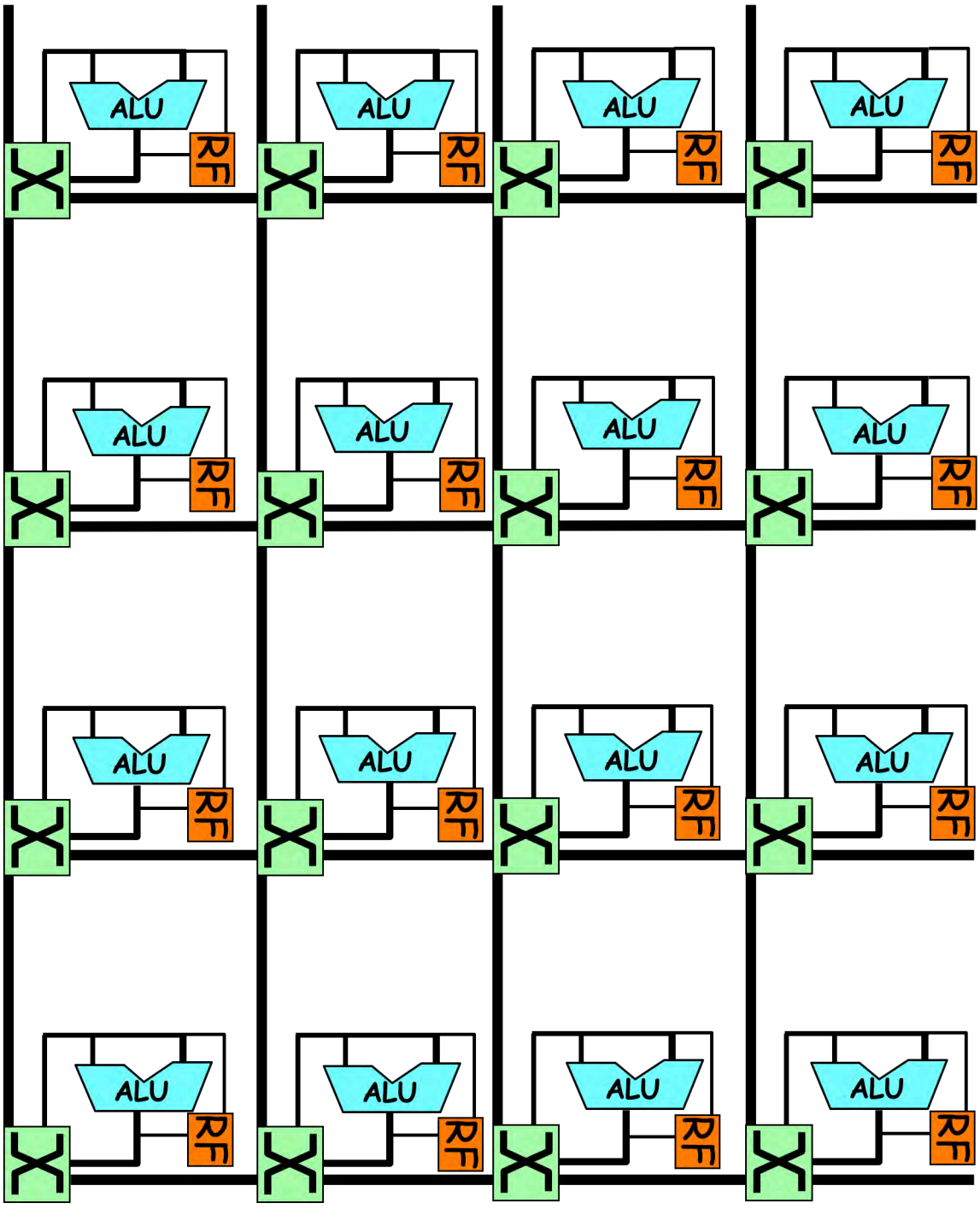
Latency bonus if we map communicating instructions nearby so communication is local.

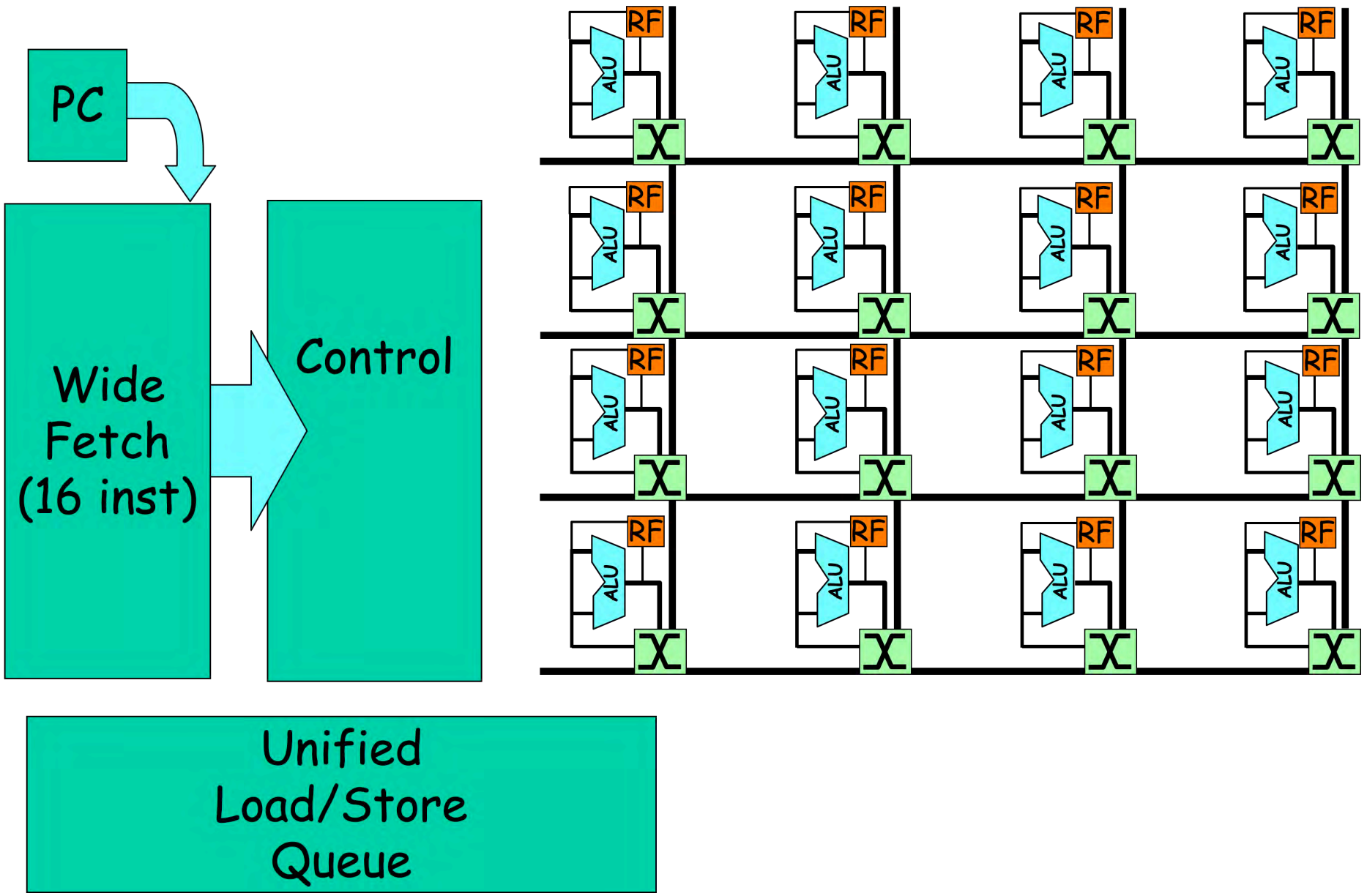
Distribute the Register File

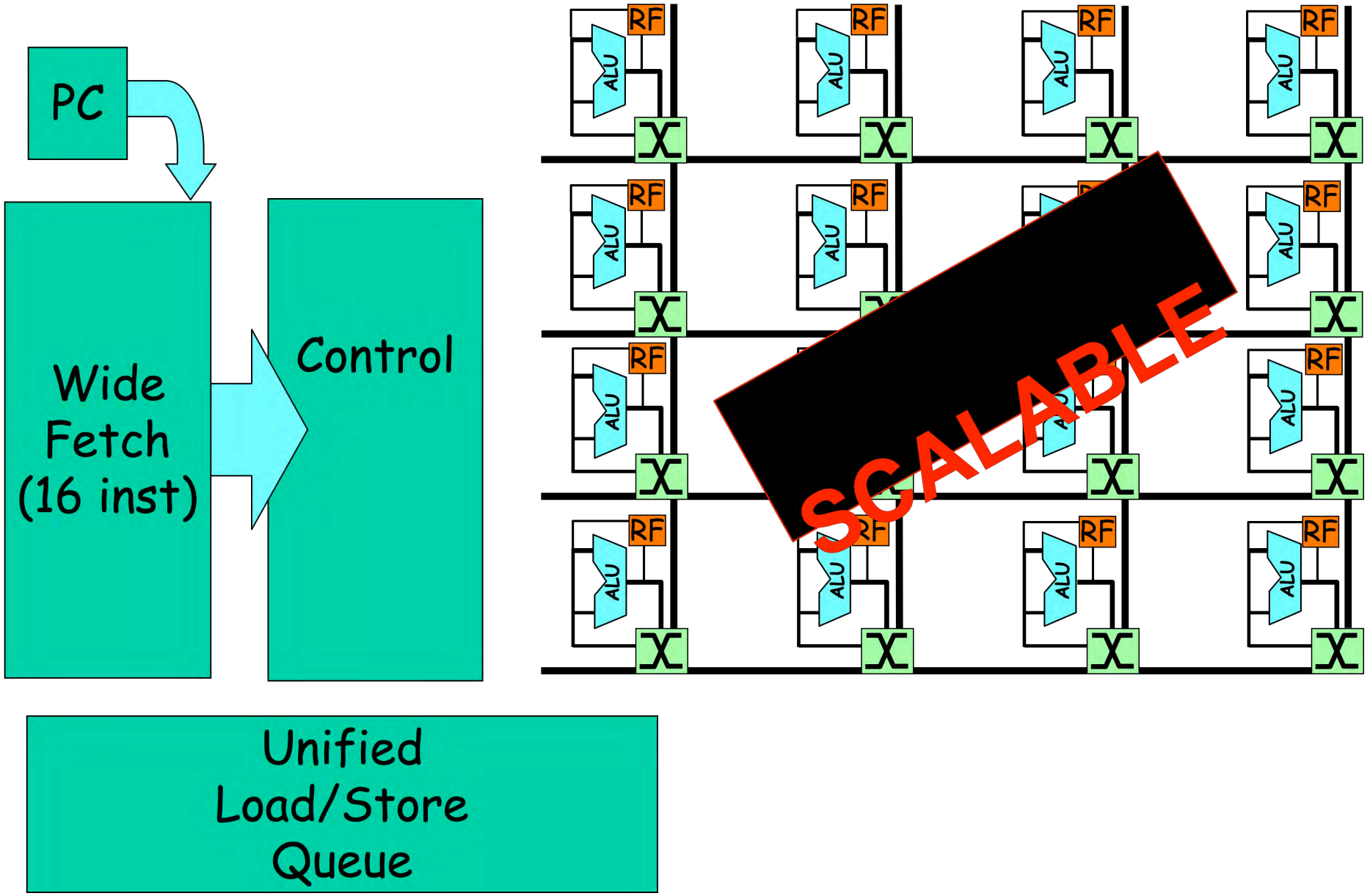


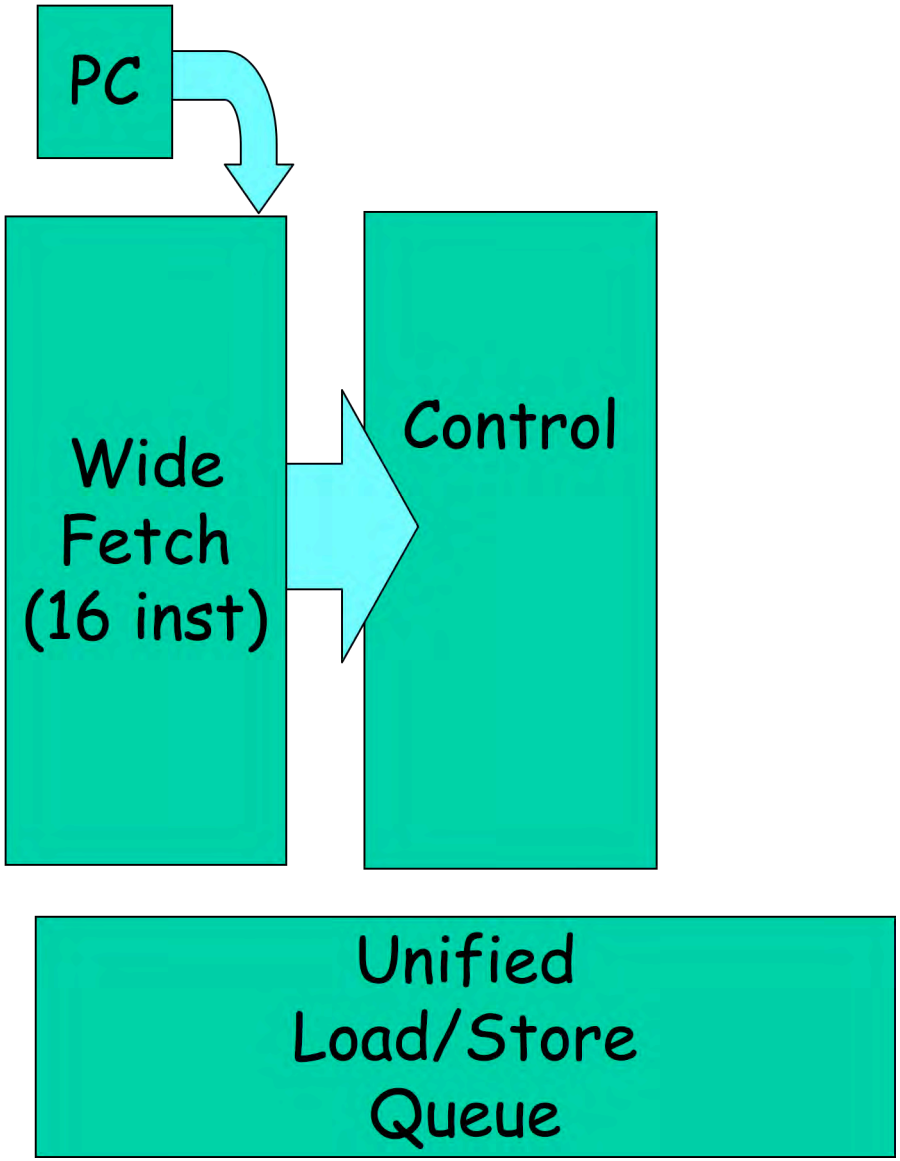
Distribute the Register File



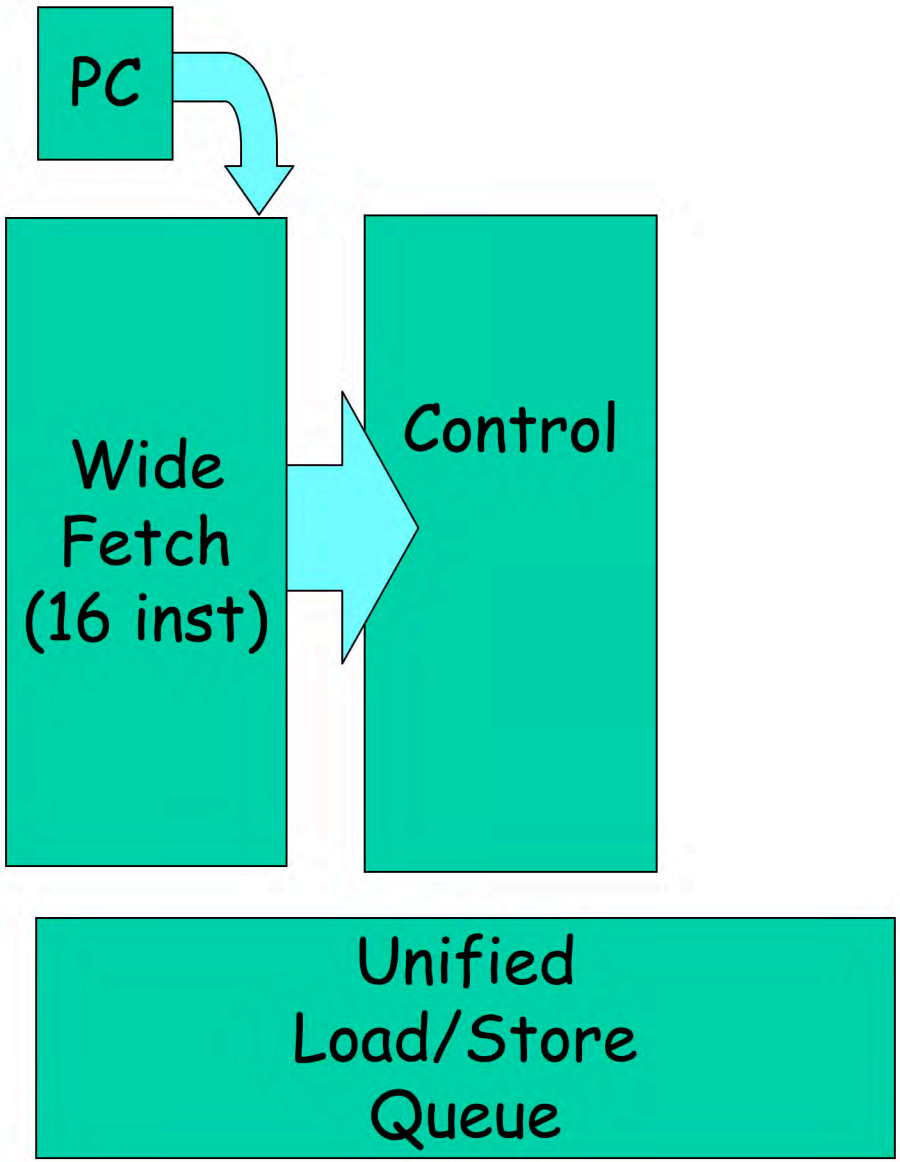

ALU
RF


PC
Wide Fetch (16 inst)
Control
RF
ALU
Unified Load/Store Queue

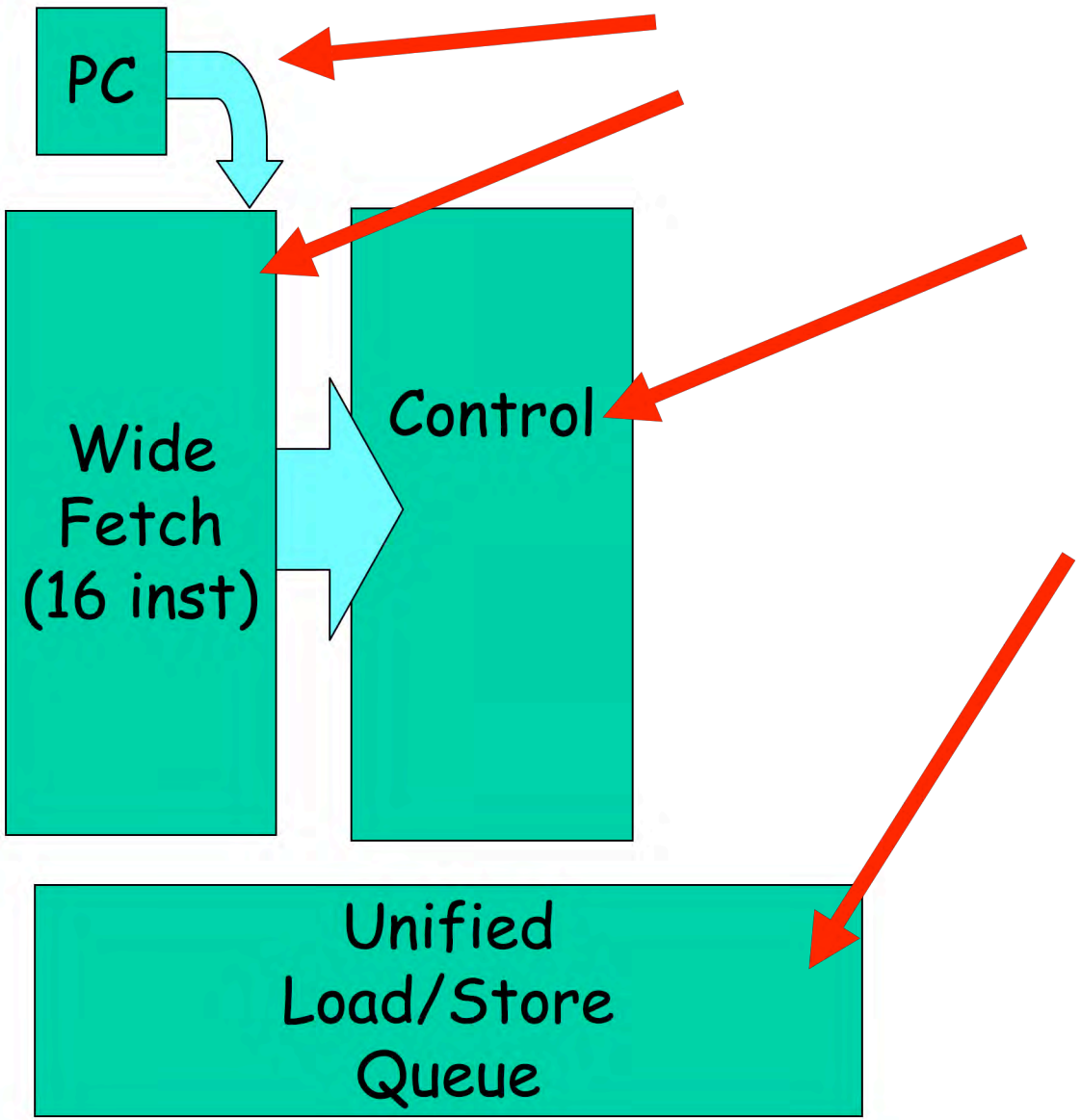

PC
Wide Fetch (16 inst)
Control
RF
ALU
SCALABLE
Unified Load/Store Queue


PC
Wide
Fetch
(16 inst)
Control
Unified
Load/Store
Queue

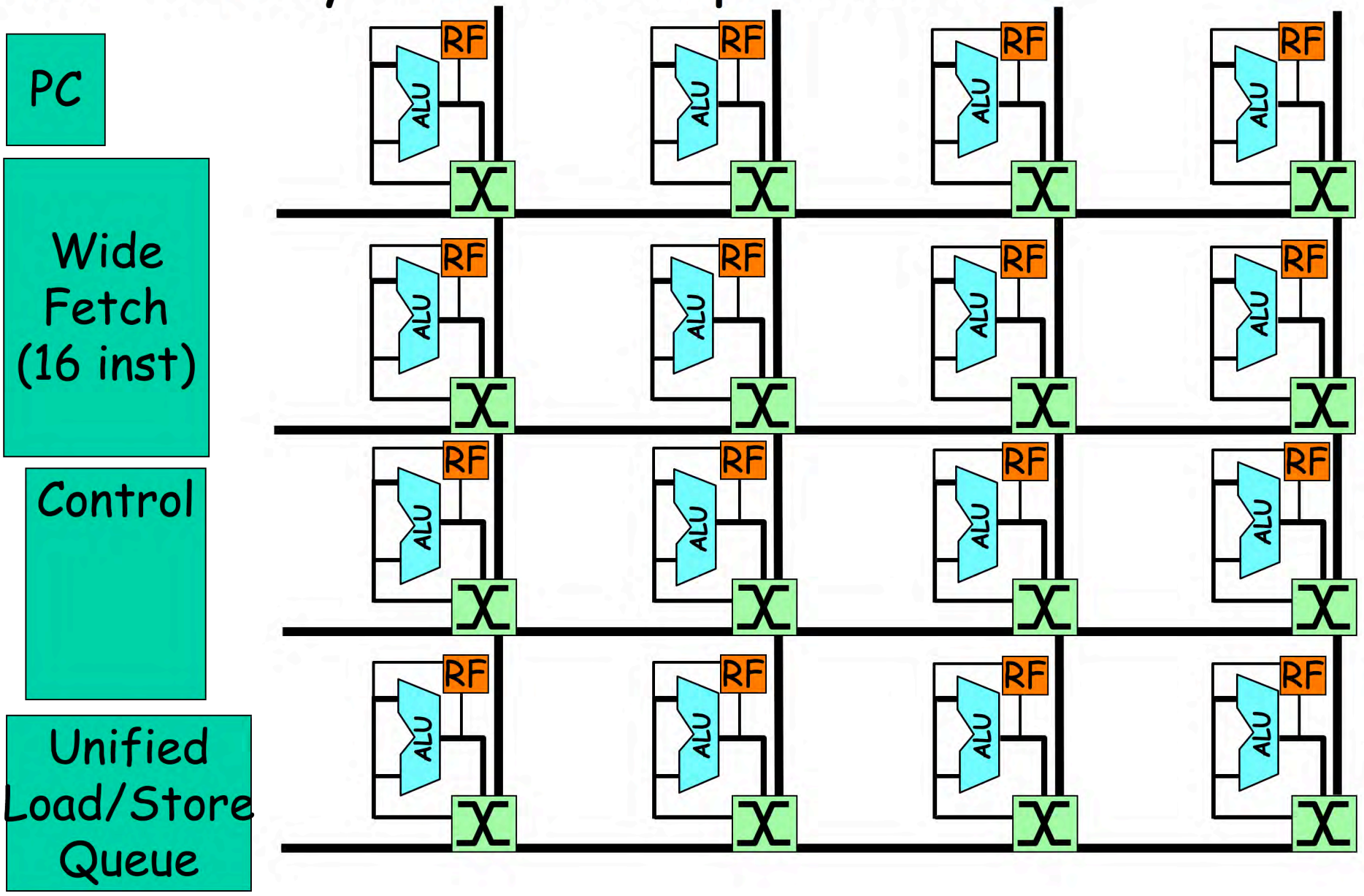
More Scalability Problems



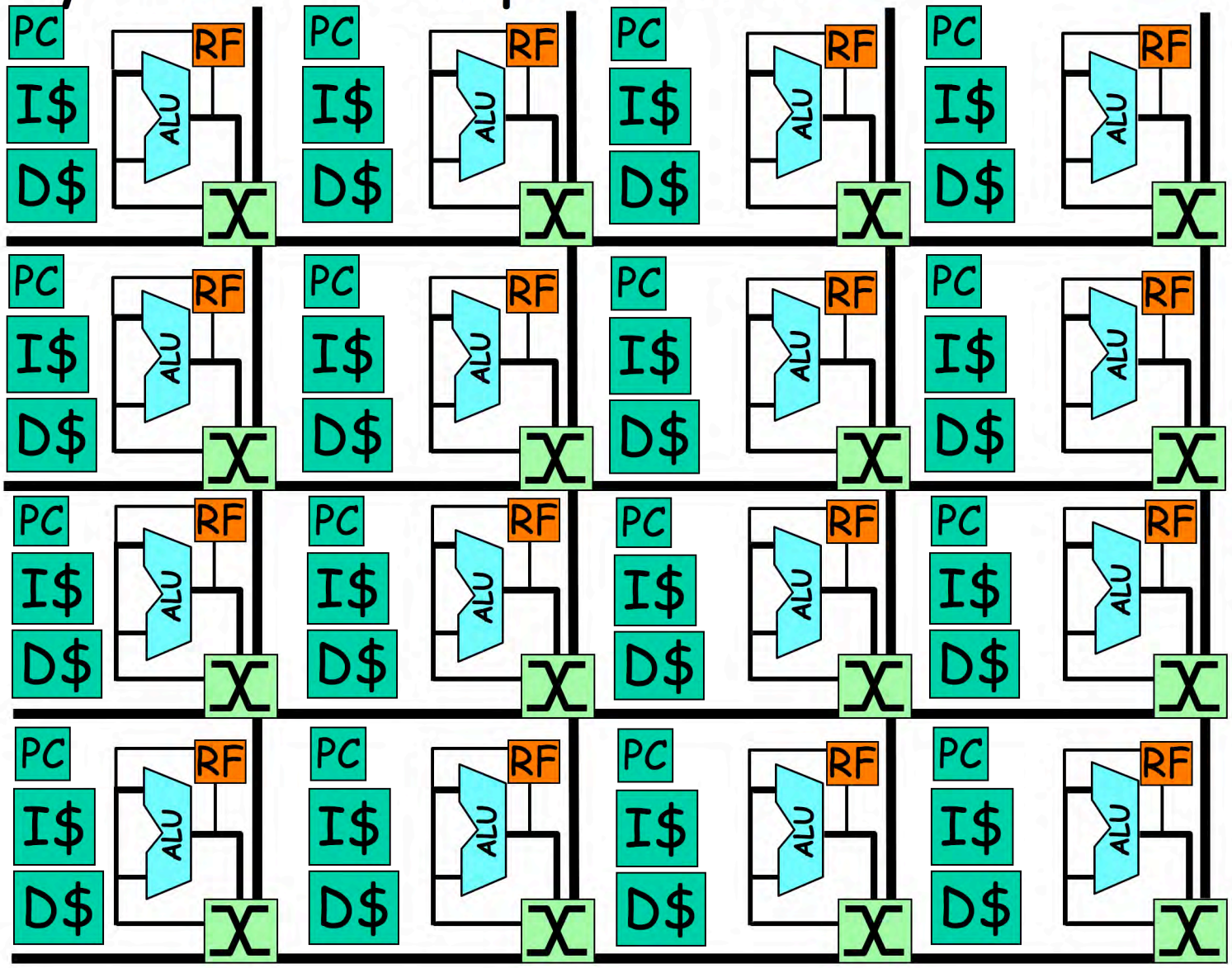
More Scalability Problems



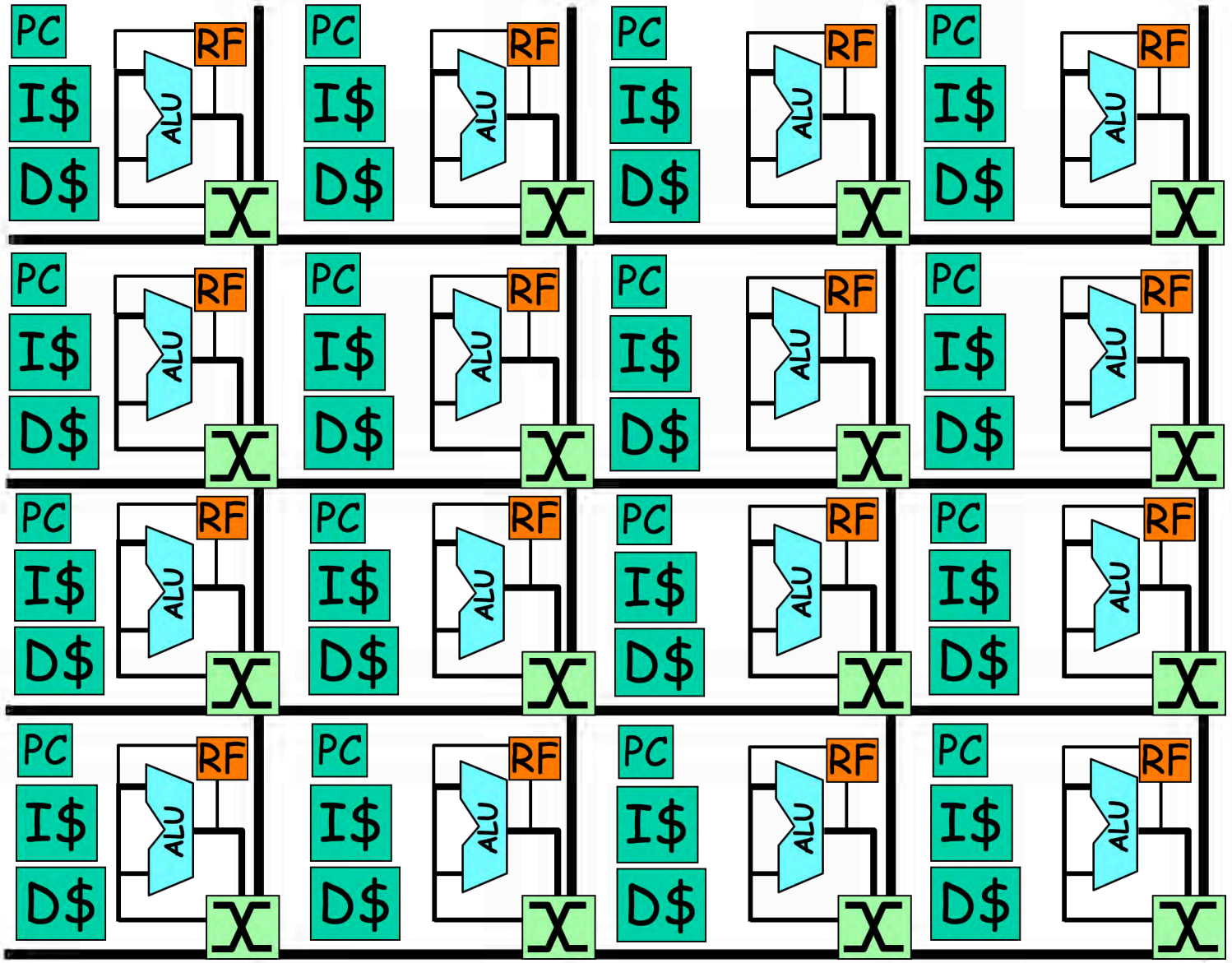
Distribute the rest:
Raw – a Fully-Tiled Microprocessor



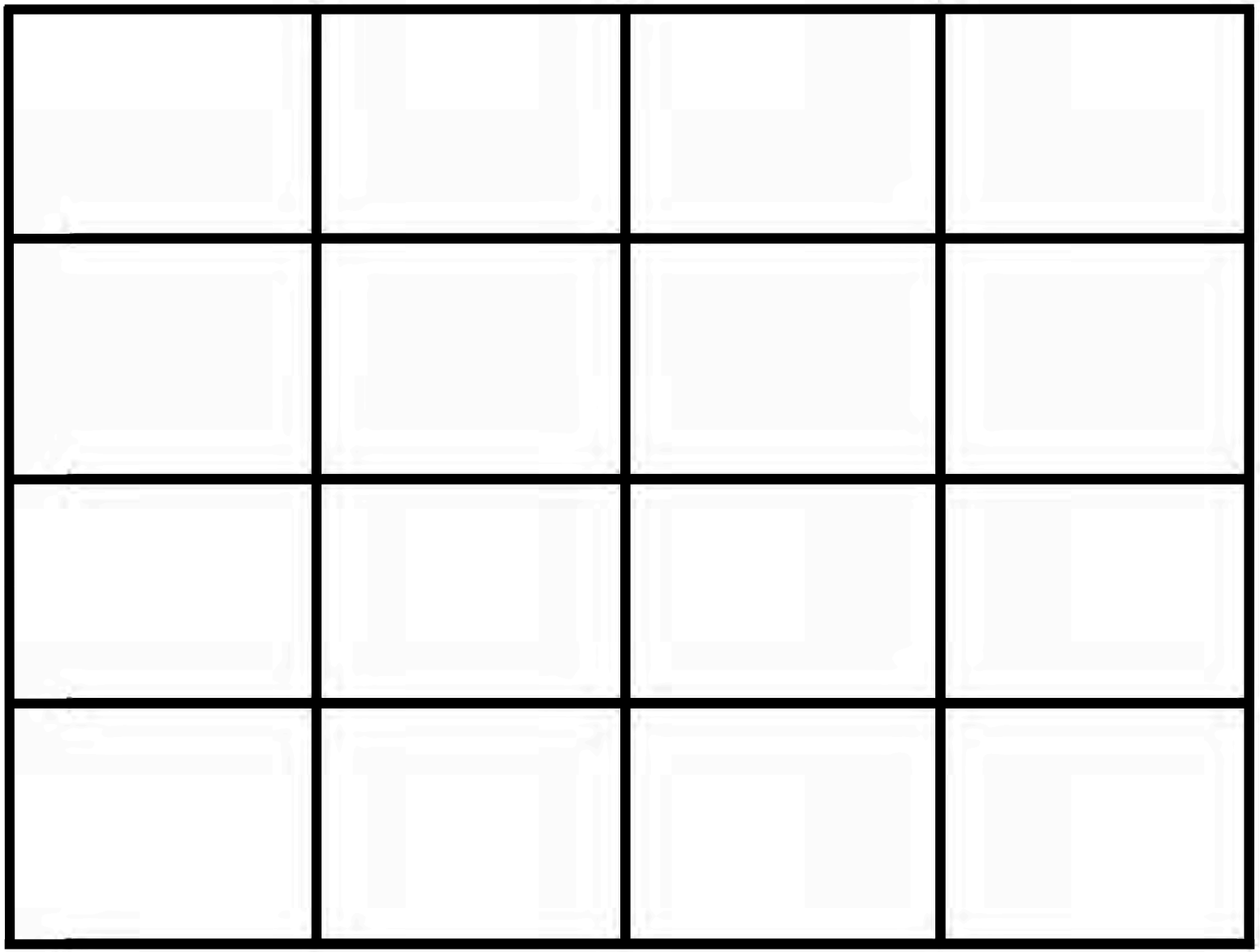
Distribute the rest:
Raw – a Fully-Tiled Microprocessor



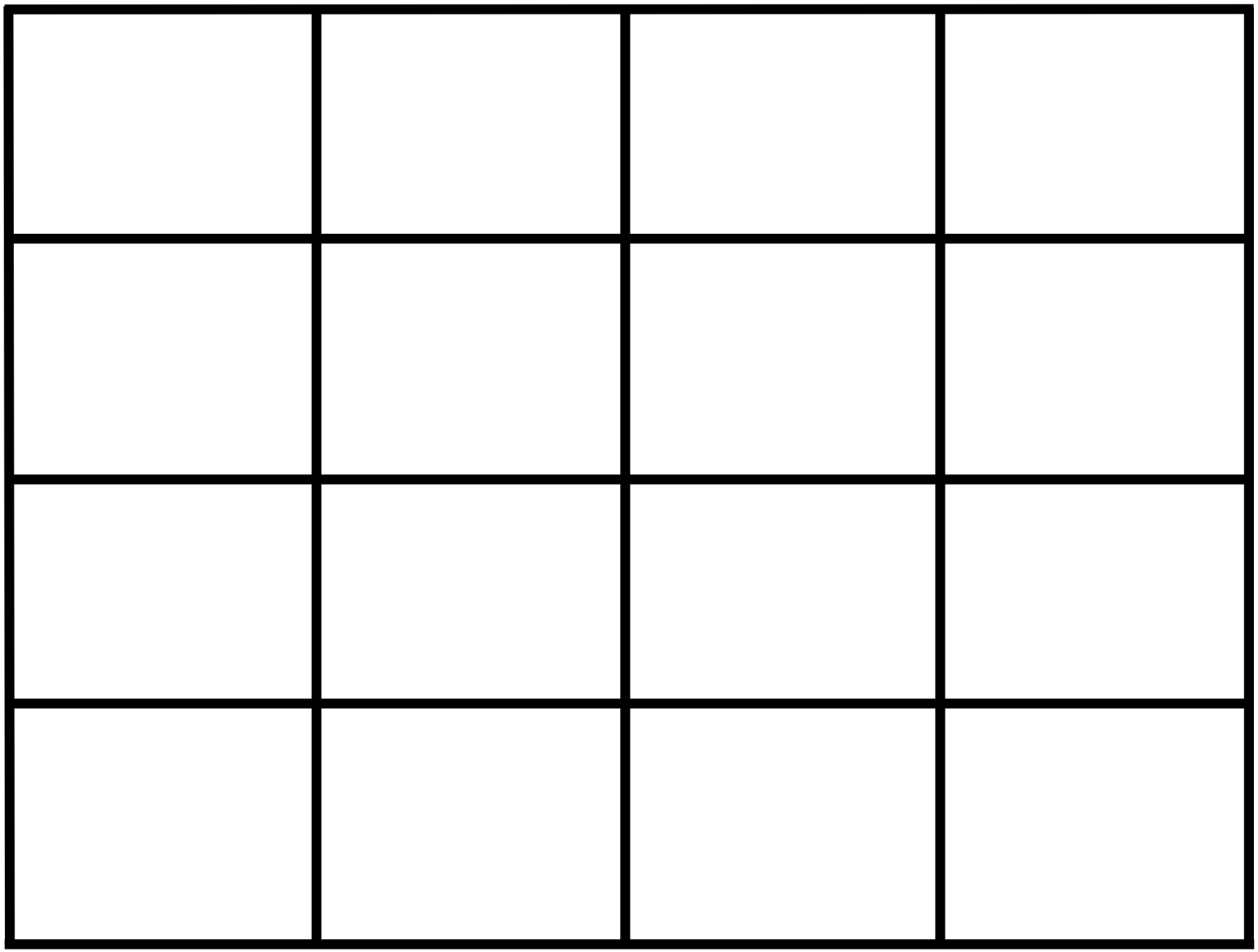
Tiles!



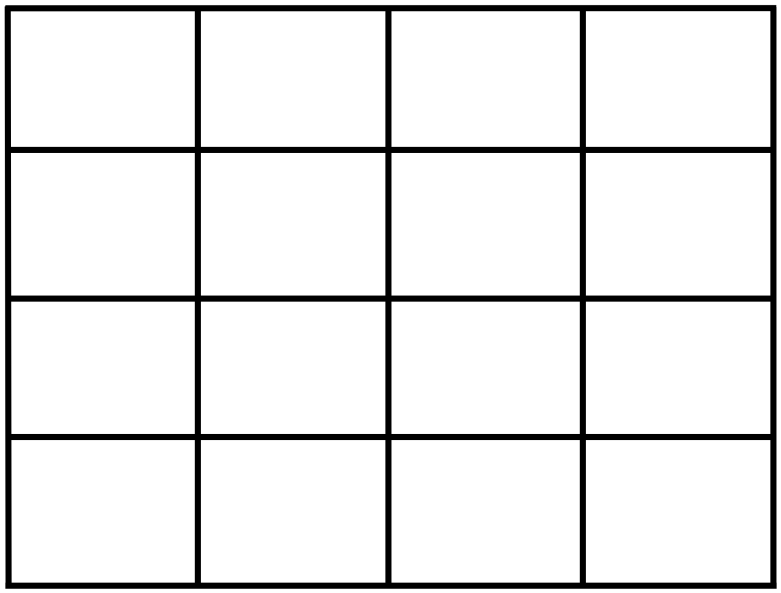
Tiles!



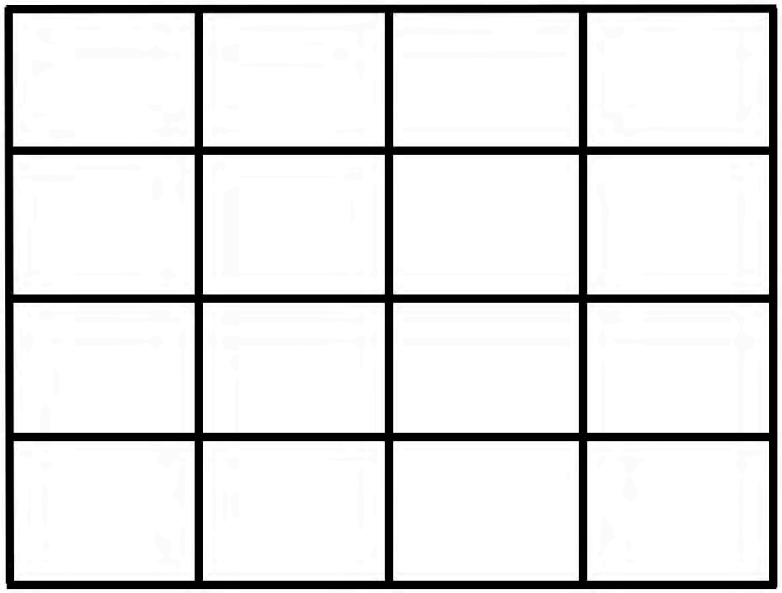
Tiles!



Tiles!



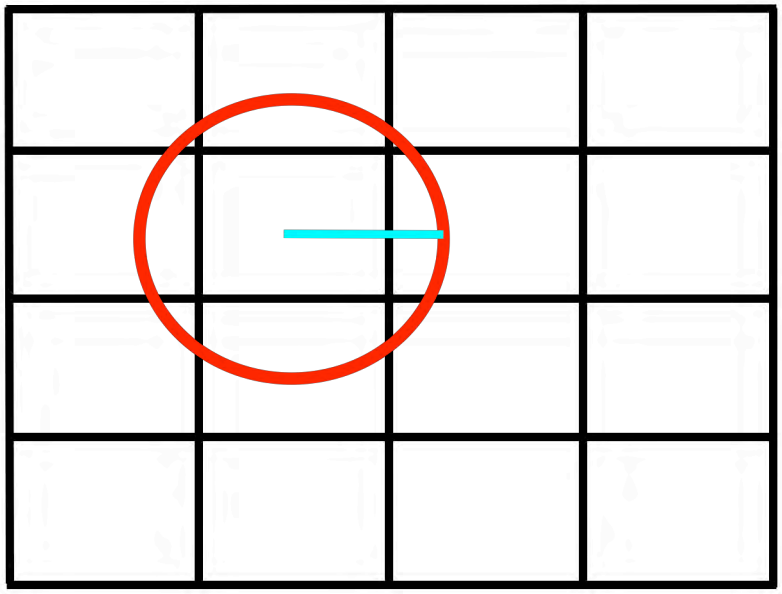
Tiled Microprocessors



-fast inter-tile communication through SON

-easy to scale (same reasons as multicore)

Tiled Microprocessors



-fast inter-tile communication through SON

-easy to scale (same reasons as multicore)

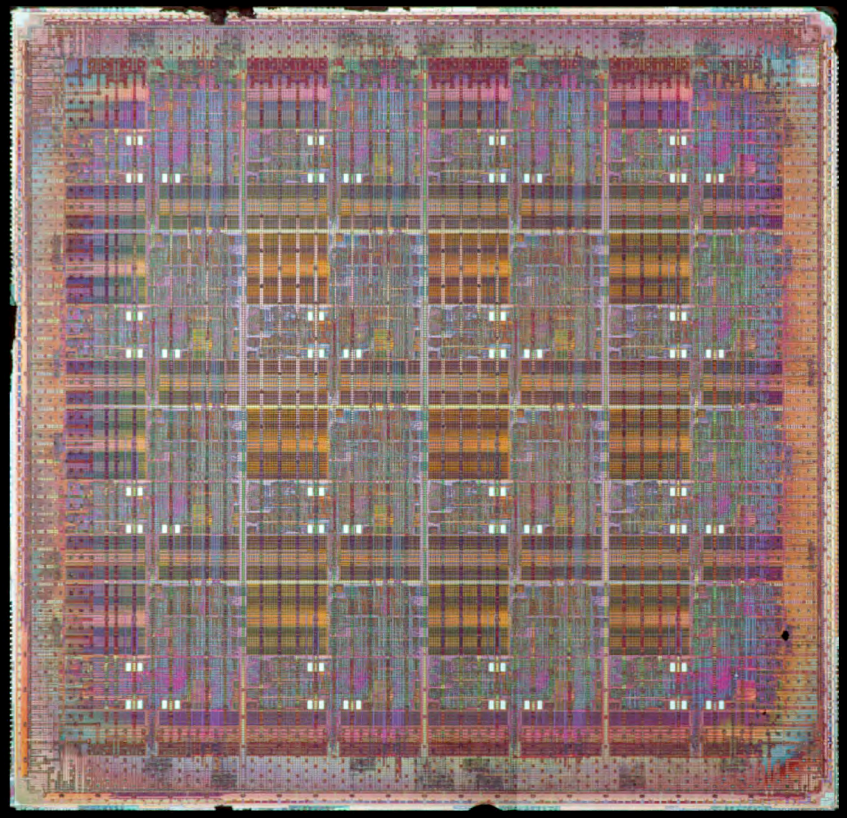
Outline

1. Scalar Operand Network and Tiled Microprocessor intro

2. Raw Architecture + SON

3. VLSI implementation of Raw,
   a scalable microprocessor with a scalar operand network.

Raw Microprocessor



Tiled scalable microprocessor
Point-to-point pipelined networks

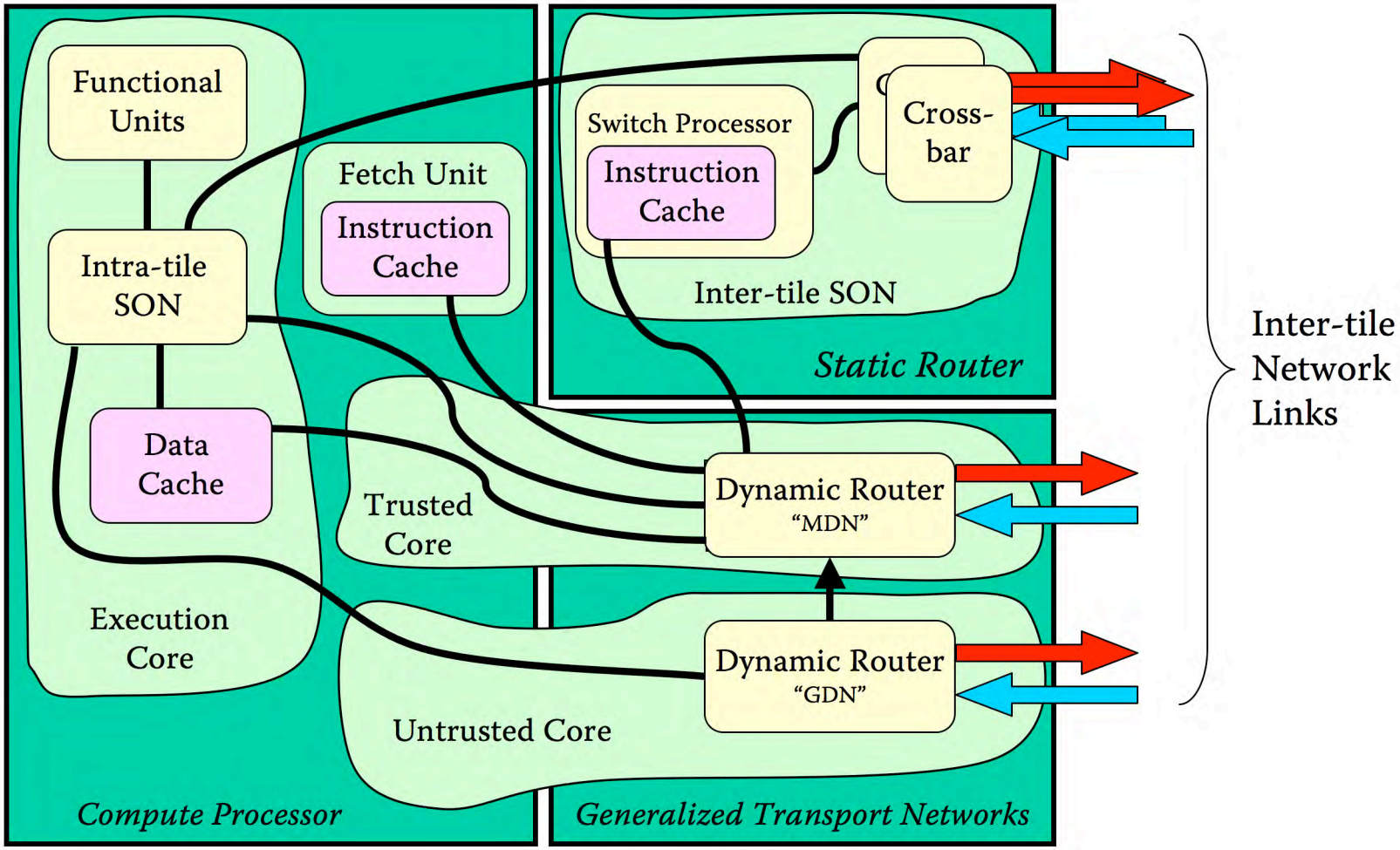
16 tiles, 16 issue

Each 4 mm x 4mm tile:

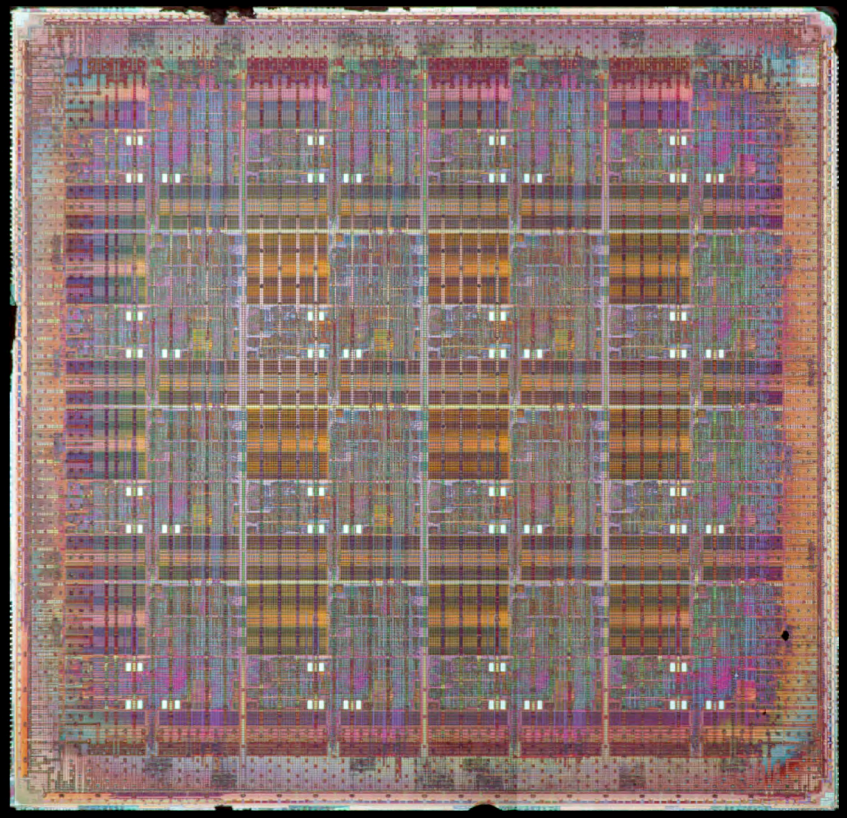
MIPS-style compute processor
- single-issue 8-stage pipe
- 32b FPU
- 32K D Cache, I Cache

4 on-chip networks
- two for operands
- one for cache misses
- one for message passing

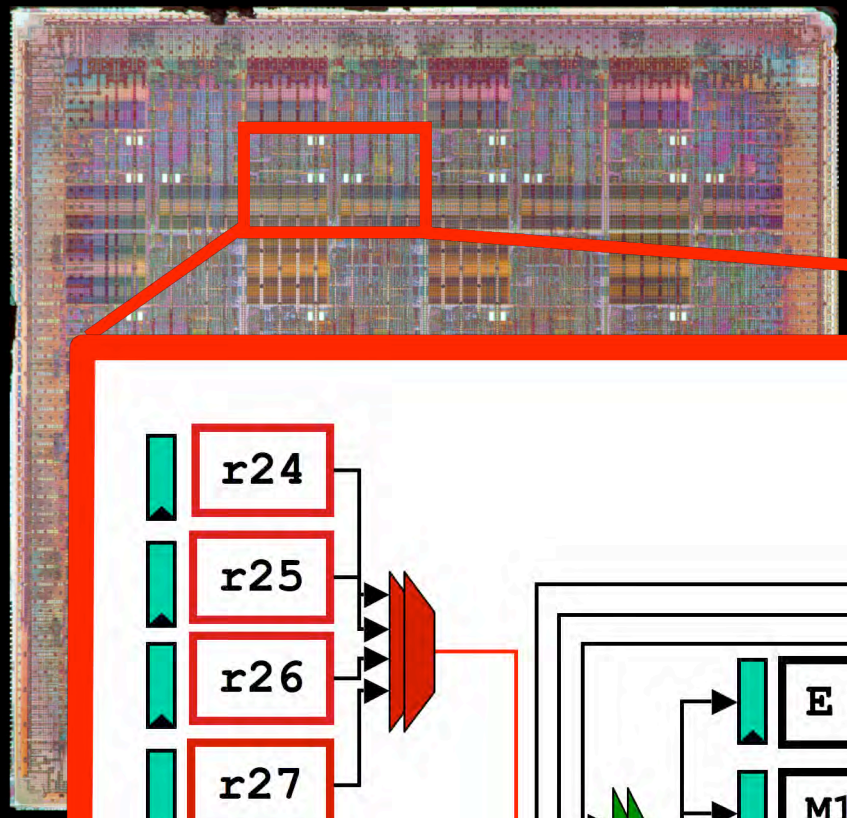
Raw Microprocessor Components



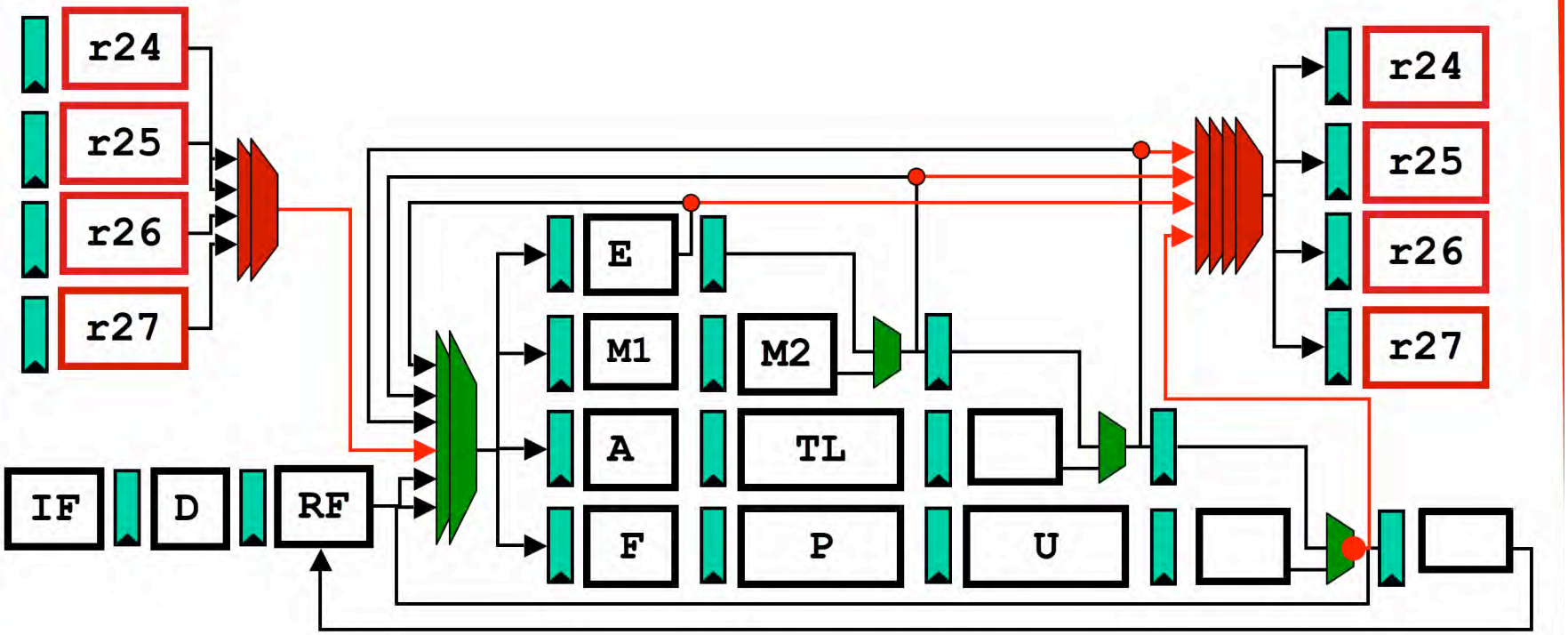
Raw Compute Processor Internals



Raw Compute Processor Internals

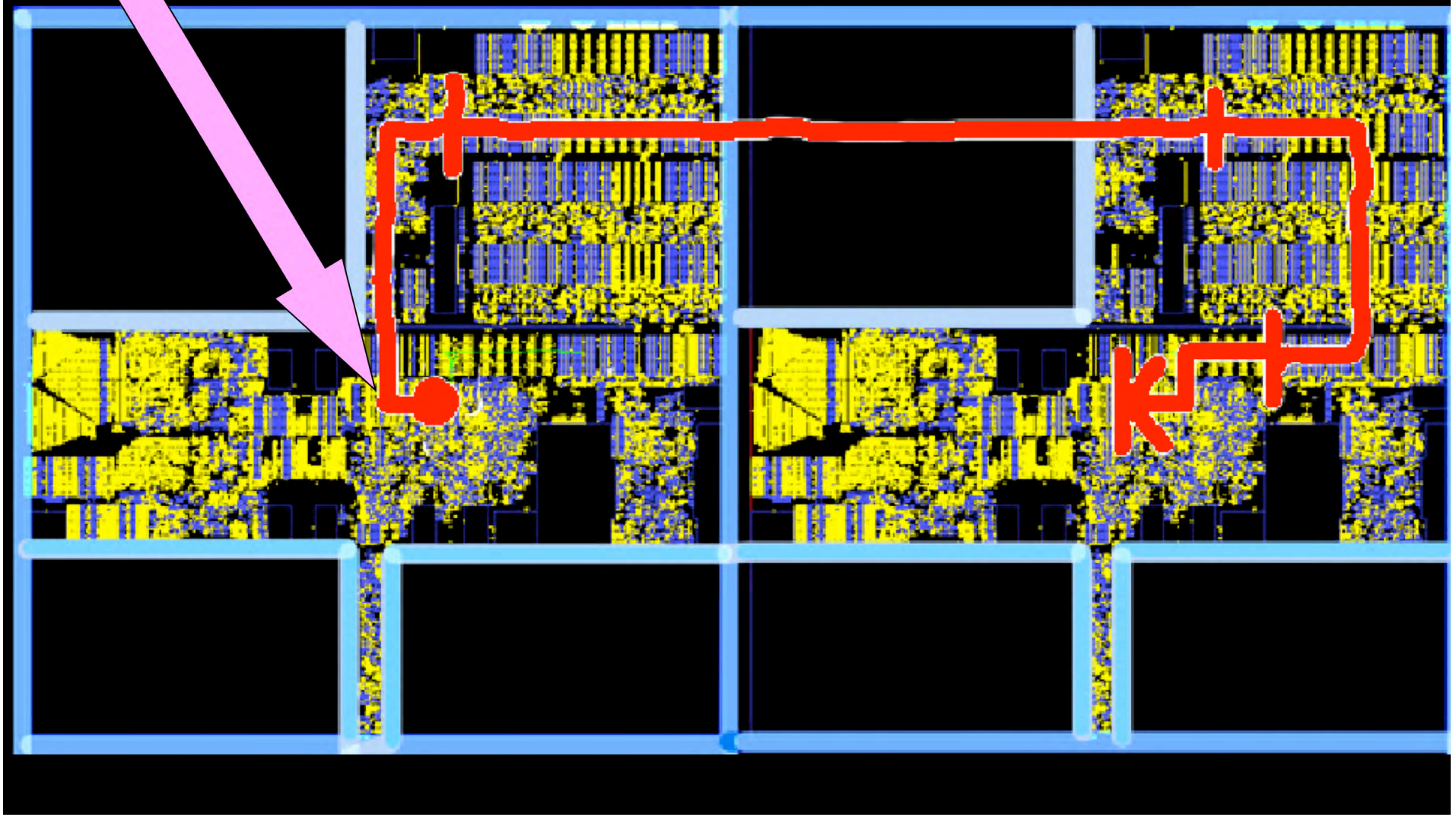


Ex: fadd r24, r25, r26



Tile-Tile Communication

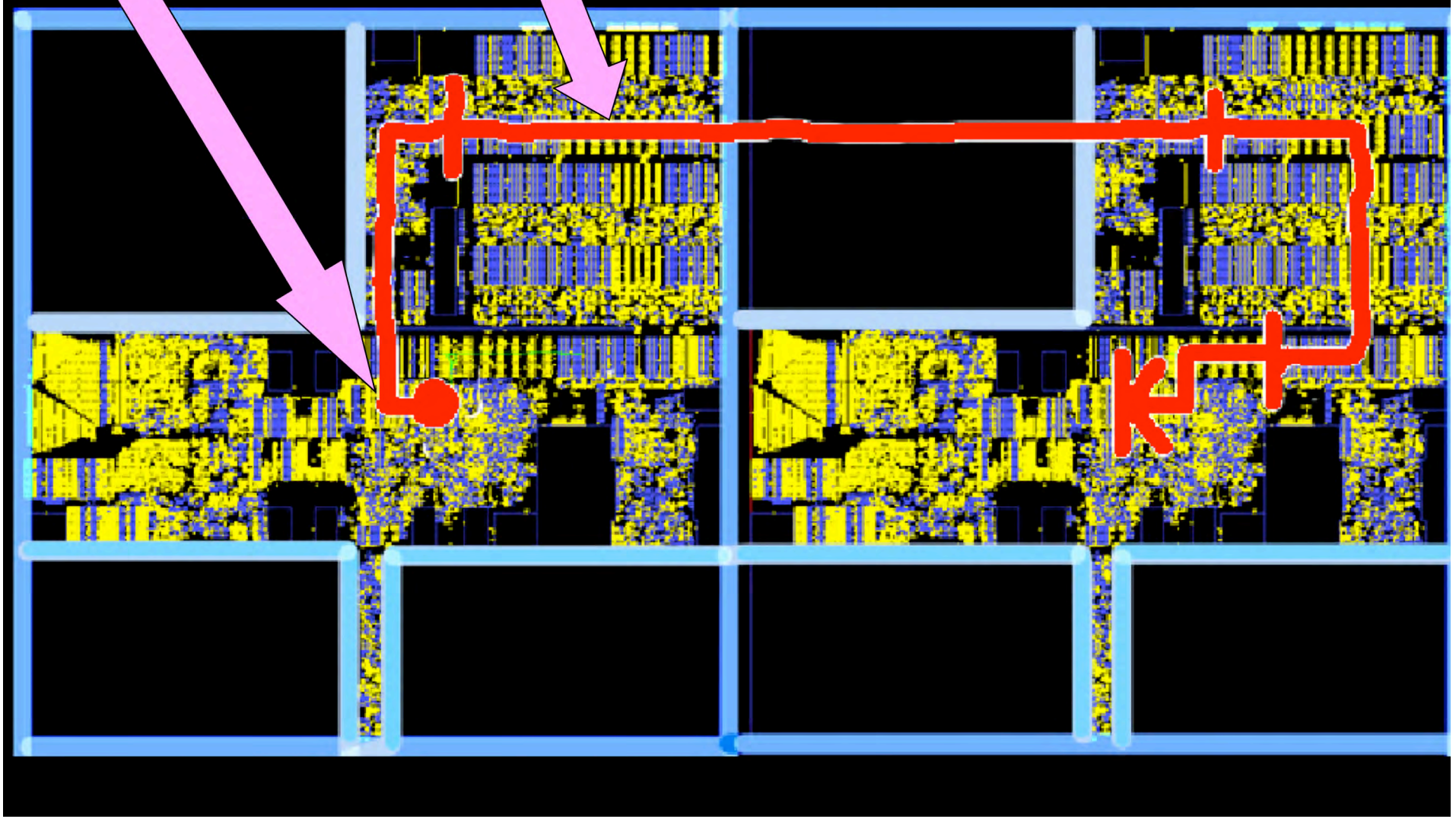
add $25,$1,$2



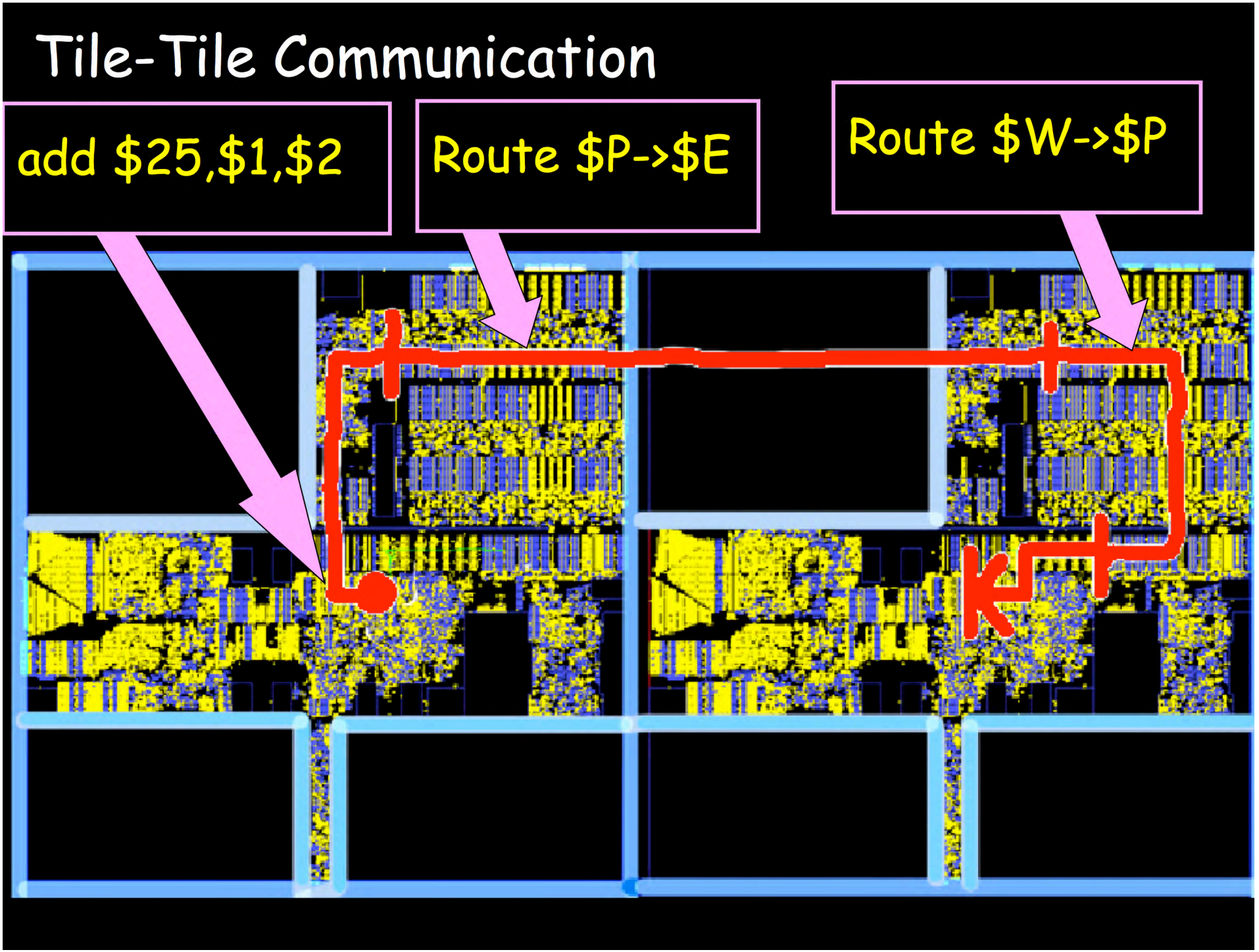
Tile-Tile Communication

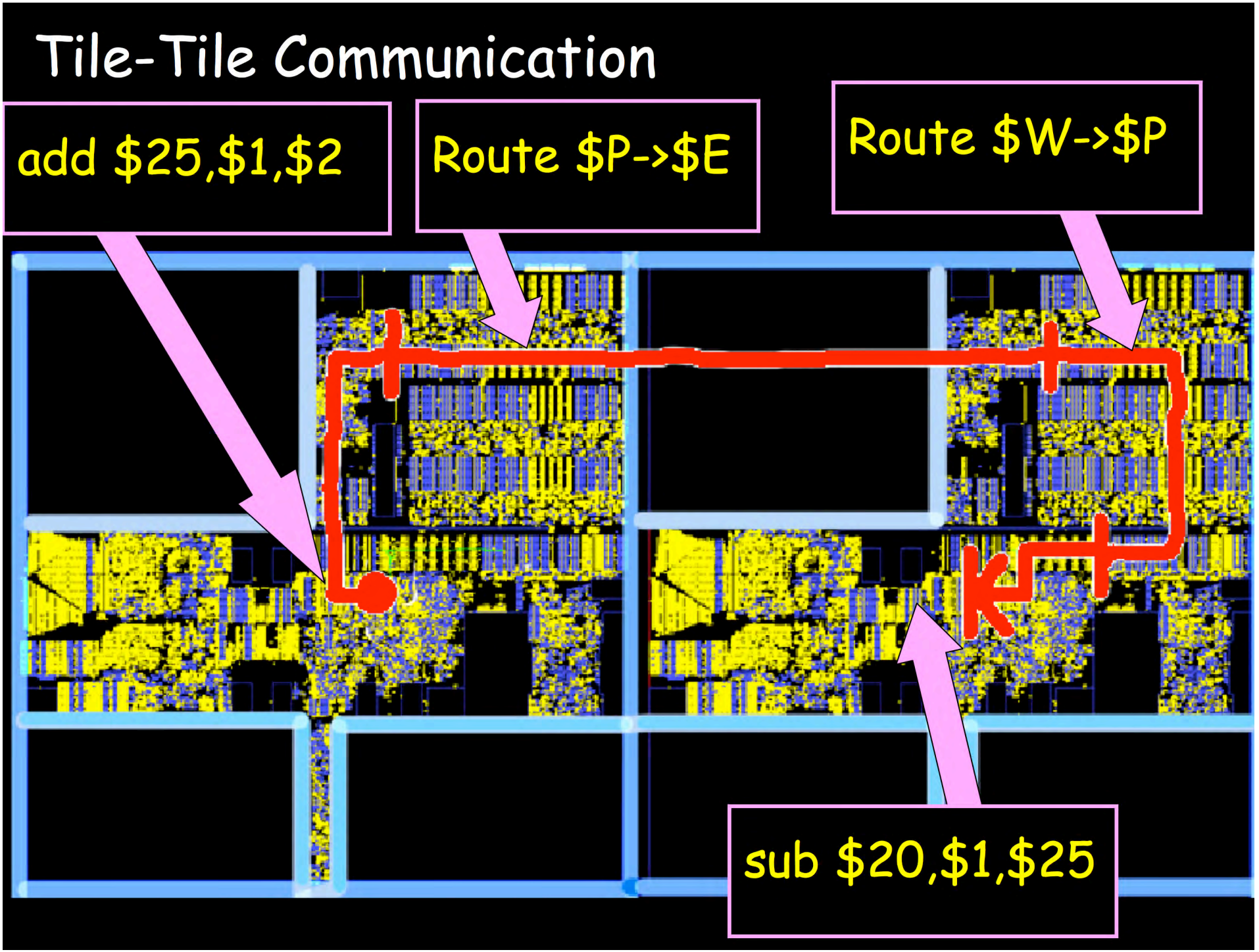
add $25,$1,$2

Route $P->$E



Tile-Tile Communication

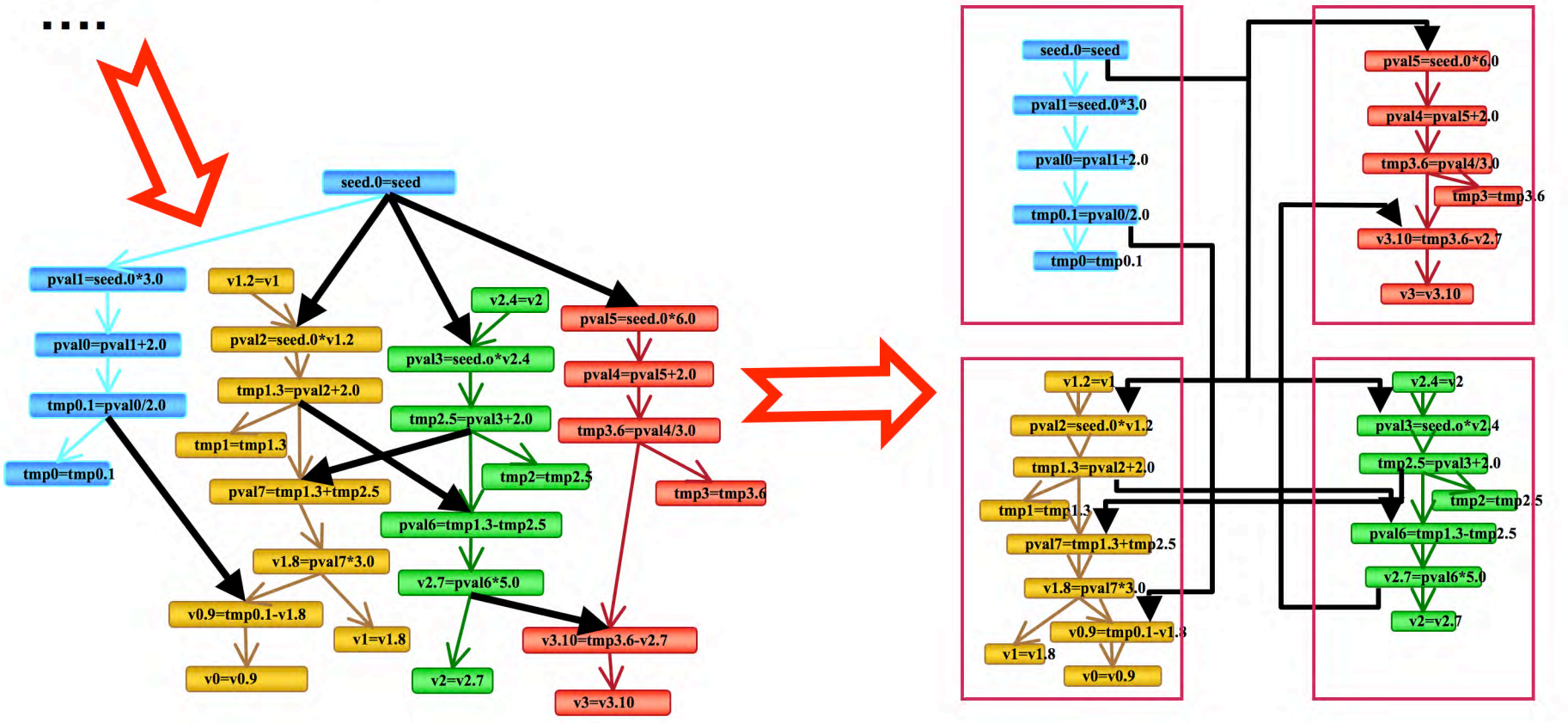



Tile-Tile Communication
add $25,$1,$2
Route $P->$E
Route $W->$P
sub $20,$1,$25

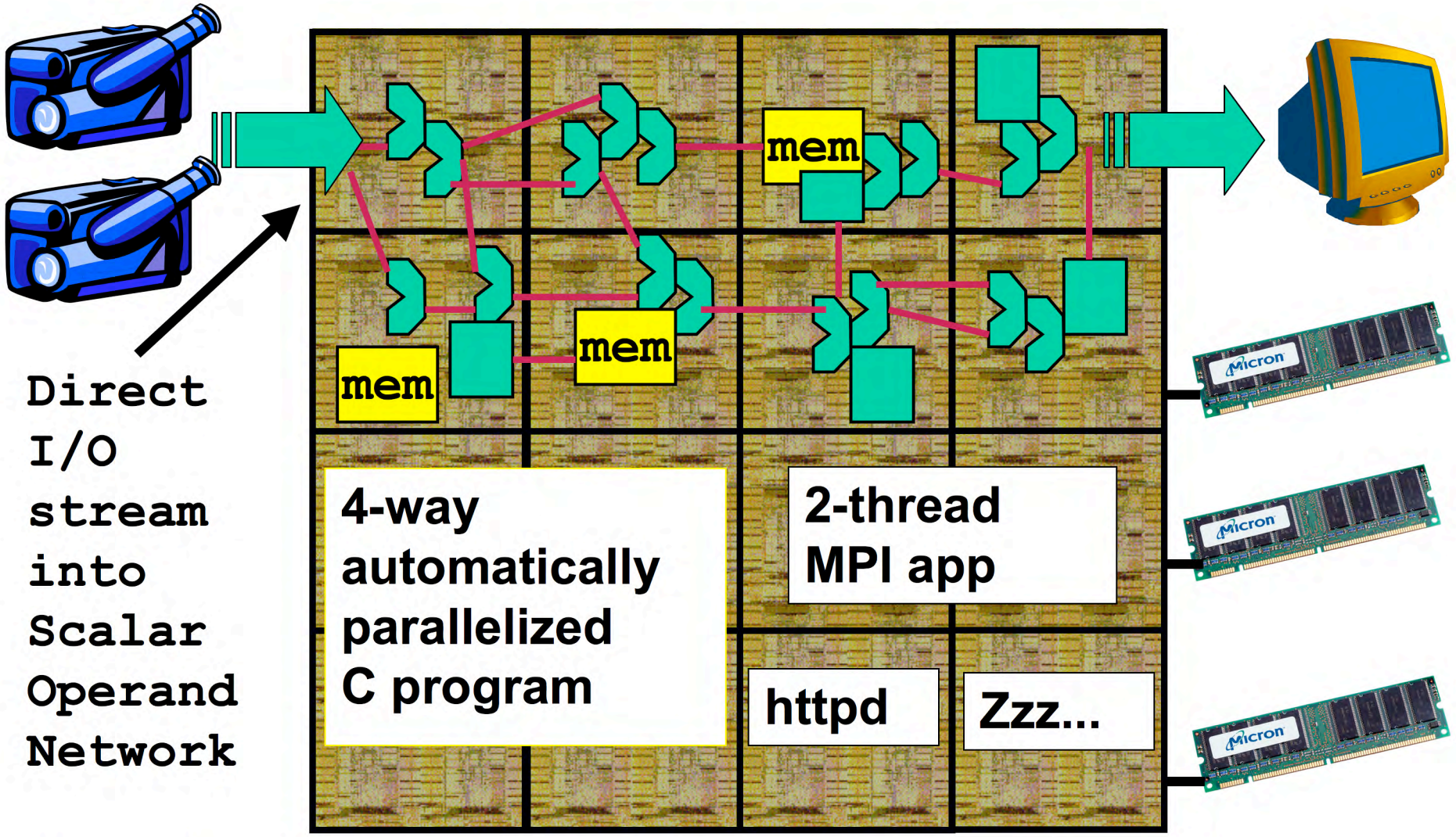
Compilation

tmp3 = (seed*6+2)/3
v2 = (tmp1 - tmp3)*5
v1 = (tmp1 + tmp2)*3
v0 = tmp0 - v1
….

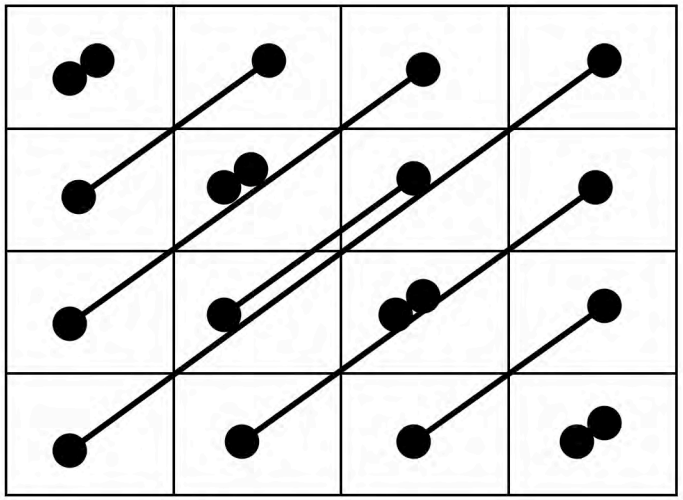
RawCC assigns instructions to the tiles, maximizing locality. It also generates the static router instructions that transfer operands between tiles.

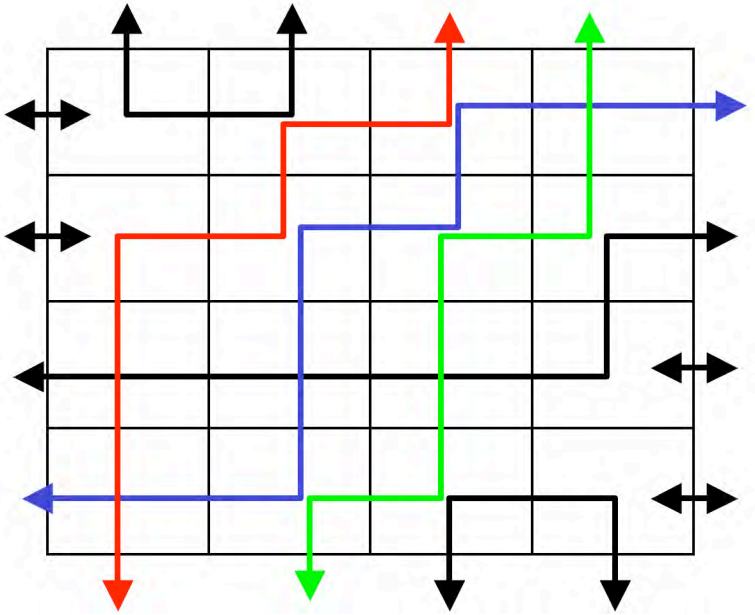


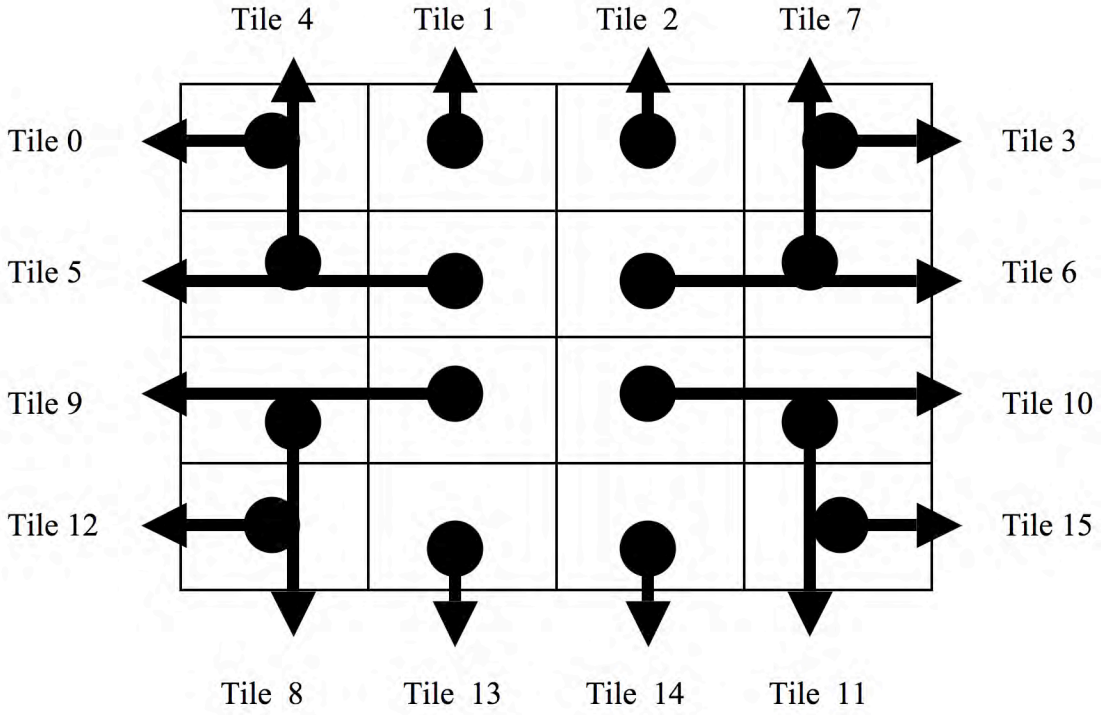
One cycle in the life of a tiled micro



An application uses only as many tiles as needed to exploit the parallelism intrinsic to that application…



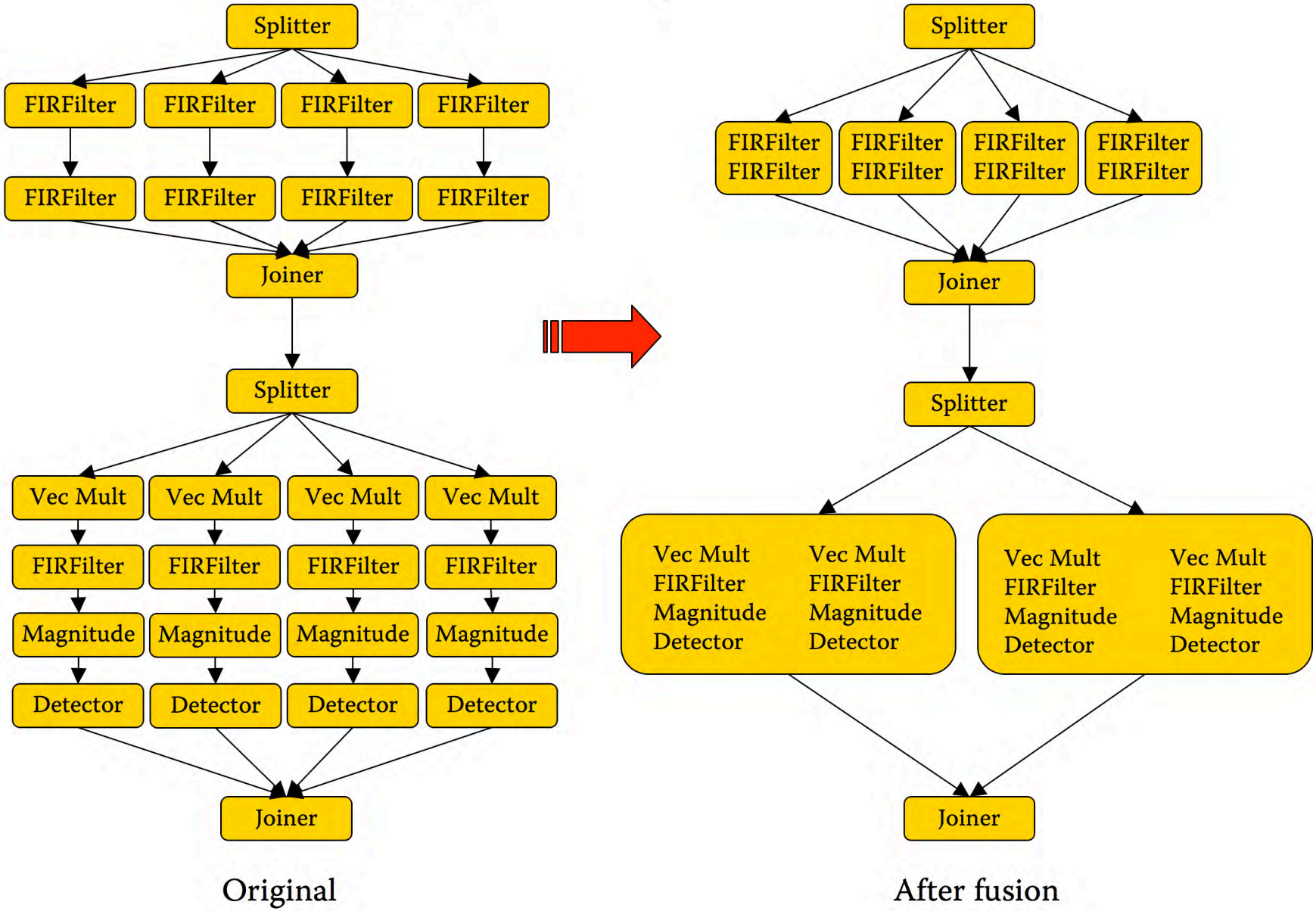


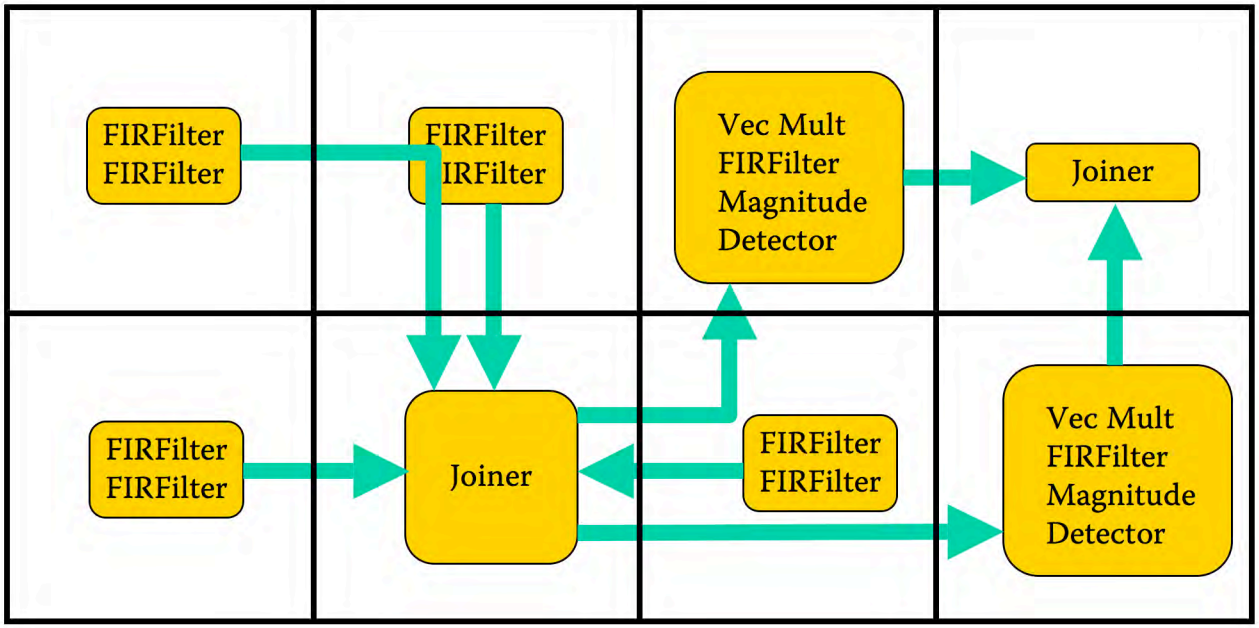


One Streaming Application on Raw

very different traffic patterns than RawCC-style parallelization

Auto-Parallelization Approach #2: Streamit Language + Compiler

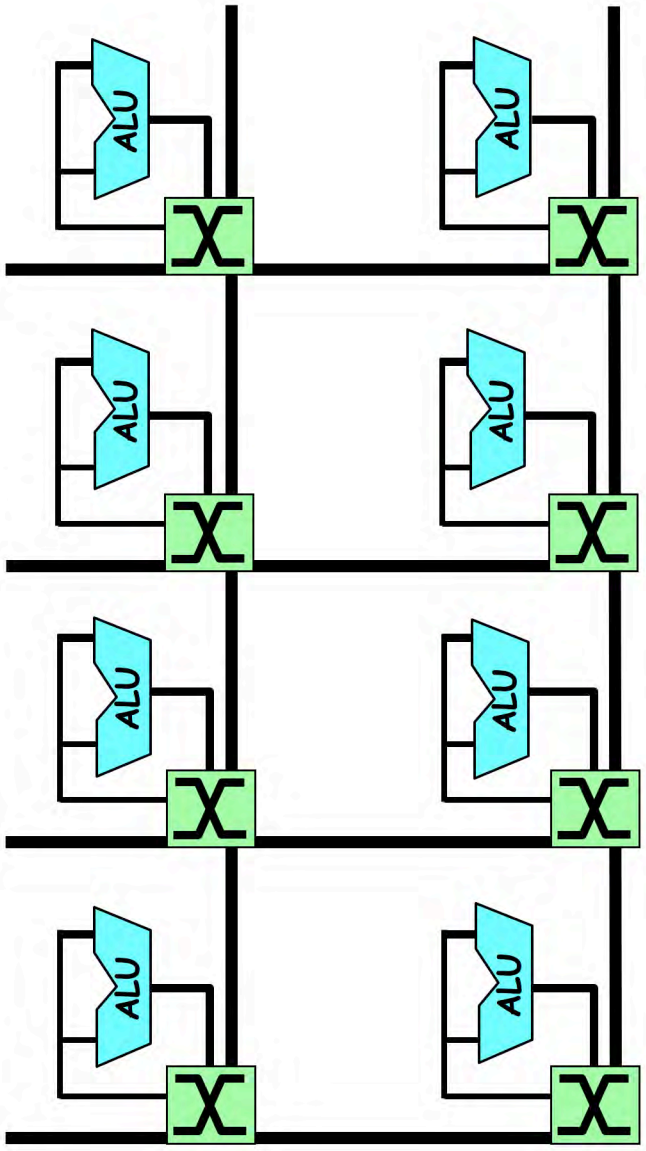




End Results – auto-parallelized by MIT Streamit to 8 tiles.

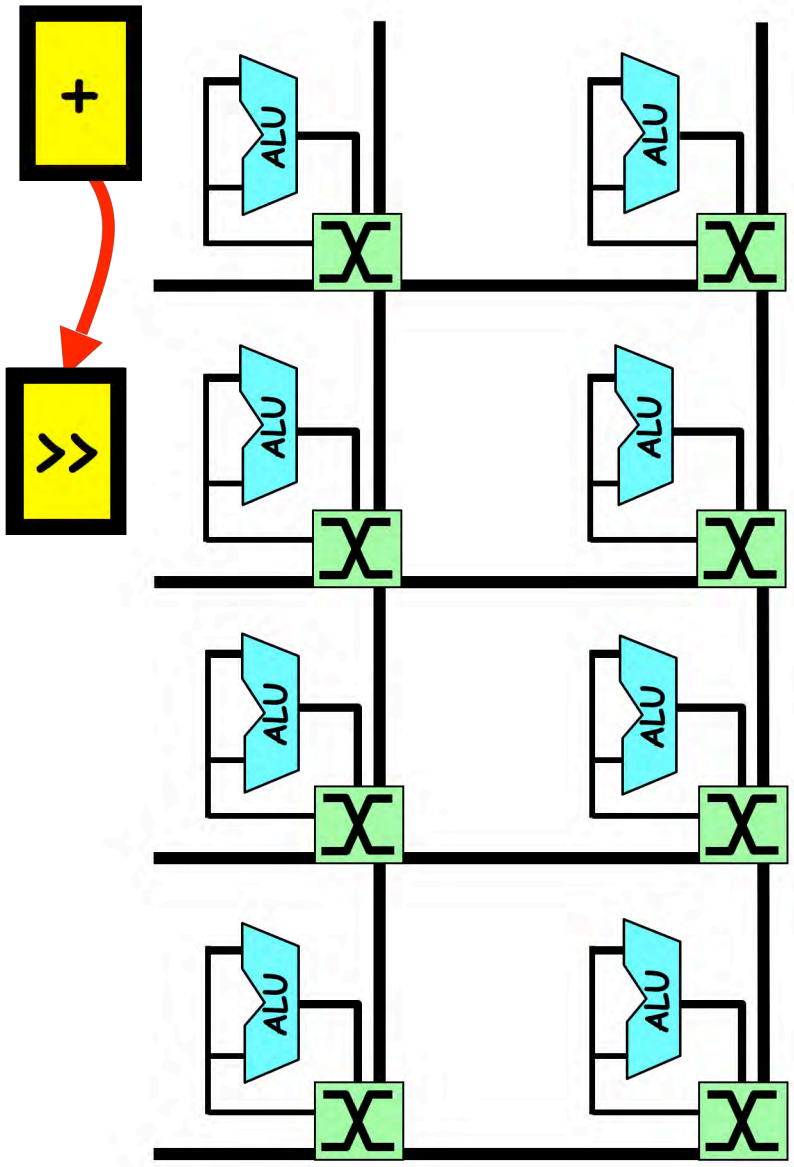
AsTrO Taxonomy: Classifying SON diversity

Assignment (Static/Dynamic)



AsTrO Taxonomy: Classifying SON diversity

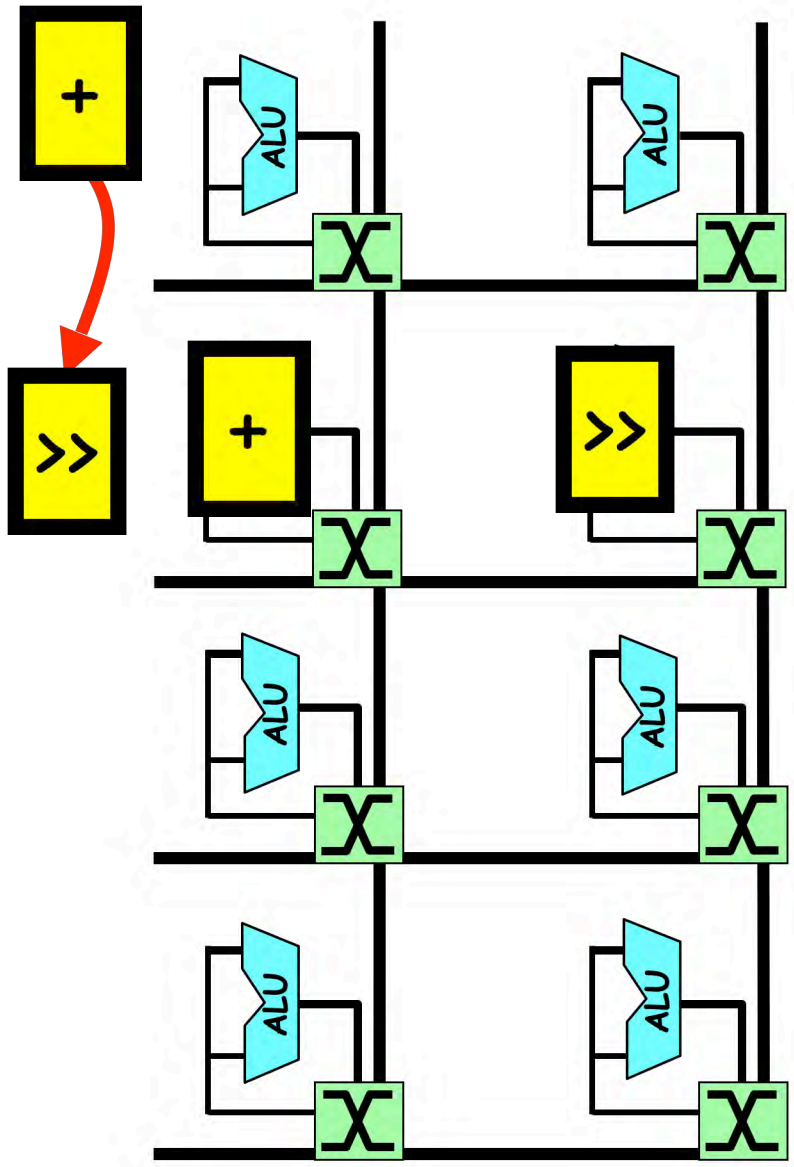
Assignment (Static/Dynamic)



AsTrO Taxonomy: Classifying SON diversity

Assignment (Static/Dynamic)

Is instruction assignment to ALUs predetermined?

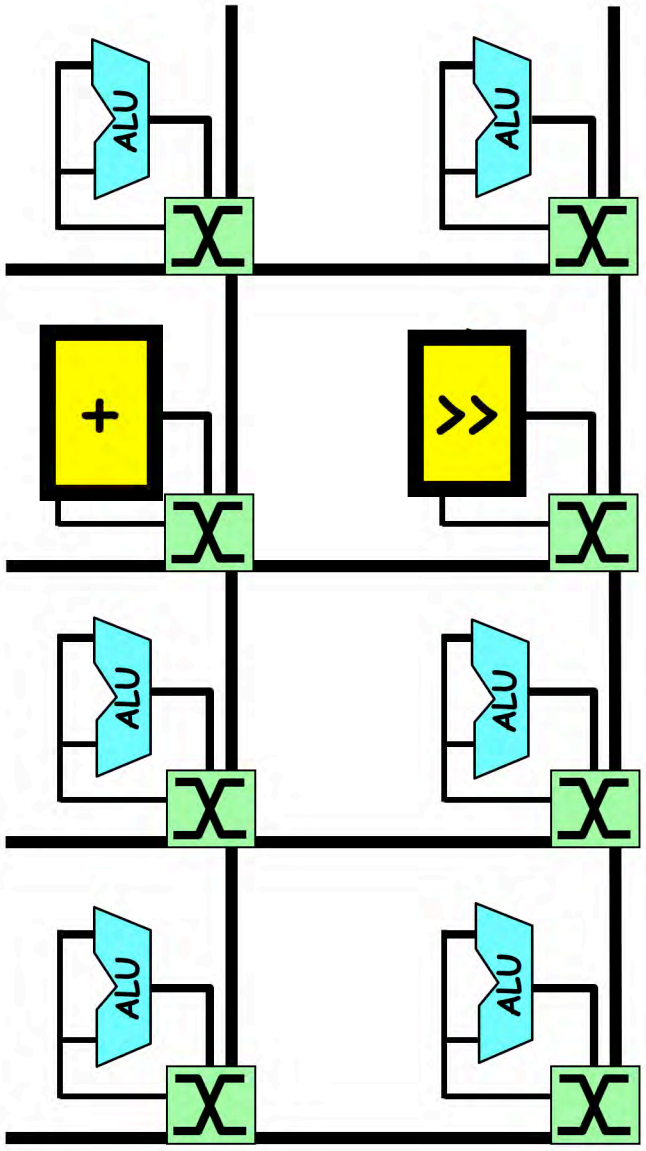


AsTrO Taxonomy: Classifying SON diversity

Assignment (Static/Dynamic)

Is instruction assignment to ALUs predetermined?

Transport (Static/Dynamic)

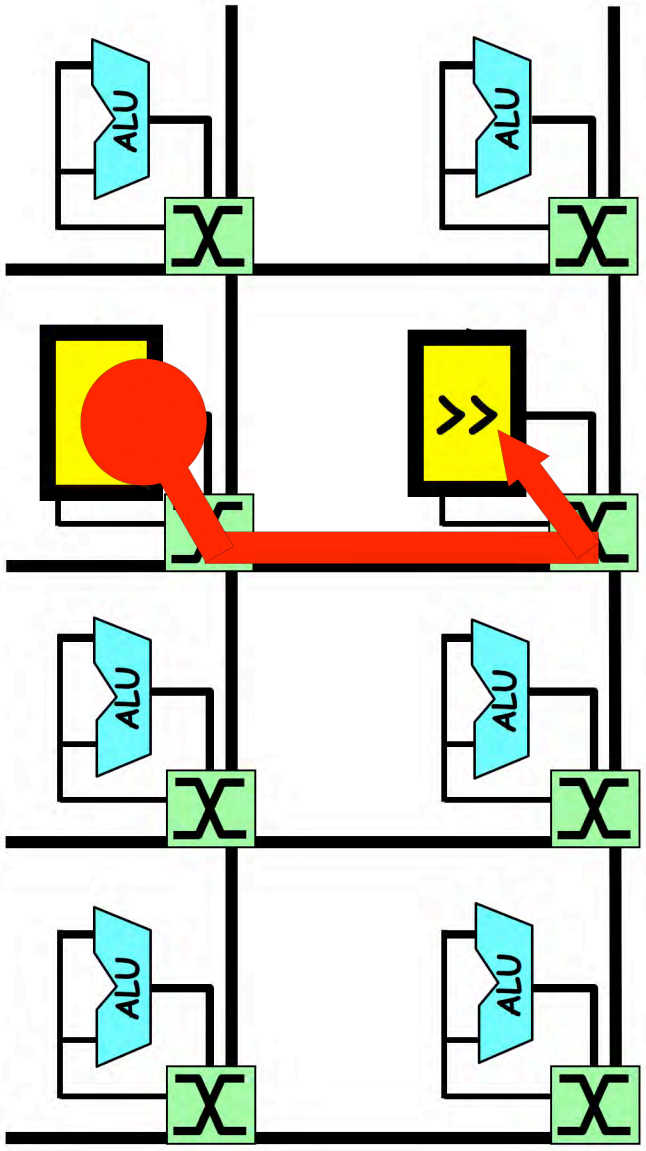


AsTrO Taxonomy: Classifying SON diversity

Assignment (Static/Dynamic)

Is instruction assignment to ALUs predetermined?

Transport (Static/Dynamic)



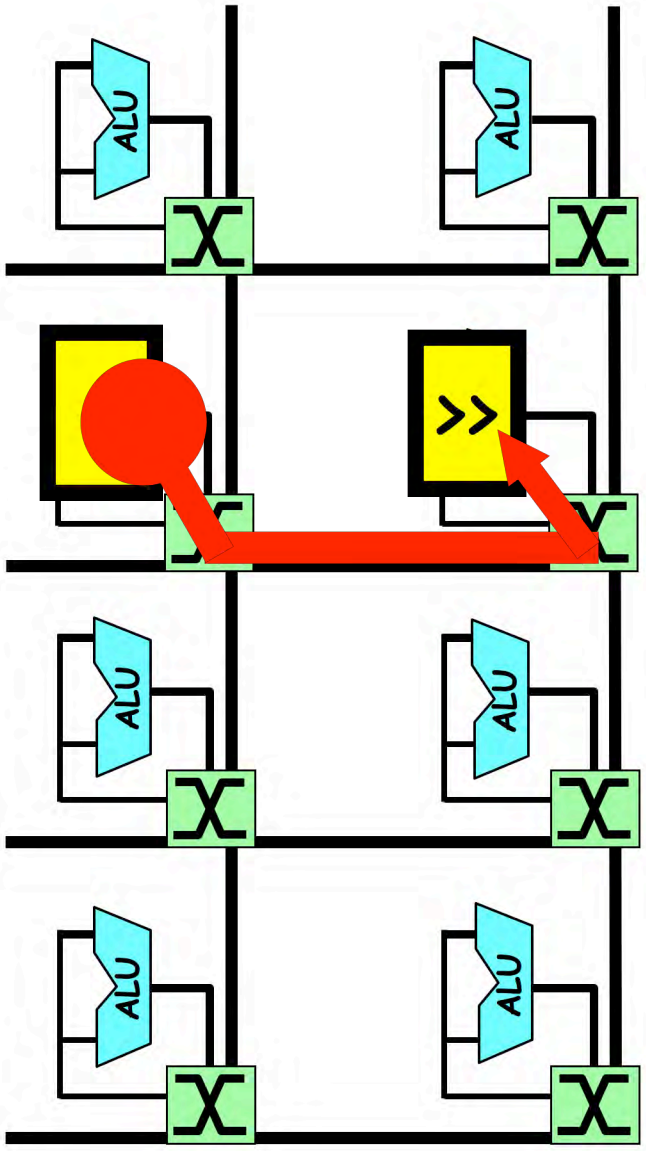
AsTrO Taxonomy: Classifying SON diversity

Assignment (Static/Dynamic)

Is instruction assignment to ALUs predetermined?

Transport (Static/Dynamic)

Are operand routes predetermined?



AsTrO Taxonomy: Classifying SON diversity

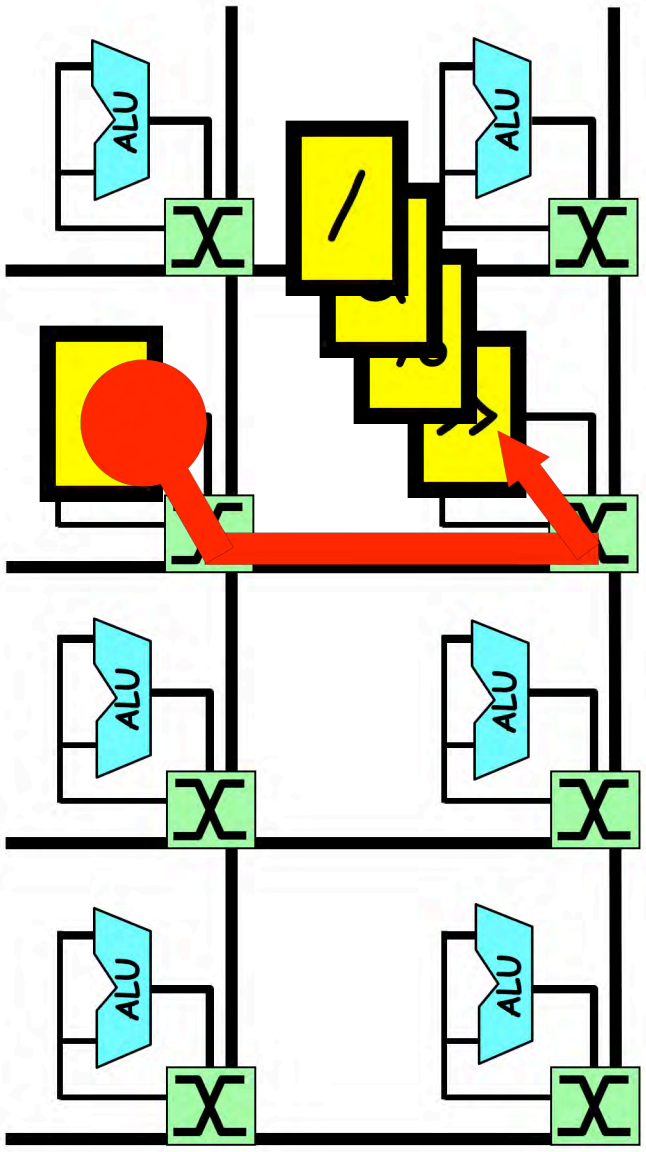
Assignment (Static/Dynamic)

Is instruction assignment to ALUs predetermined?

Transport (Static/Dynamic)

Are operand routes predetermined?

Ordering (Static/Dynamic)



AsTrO Taxonomy: Classifying SON diversity

Assignment (Static/Dynamic)

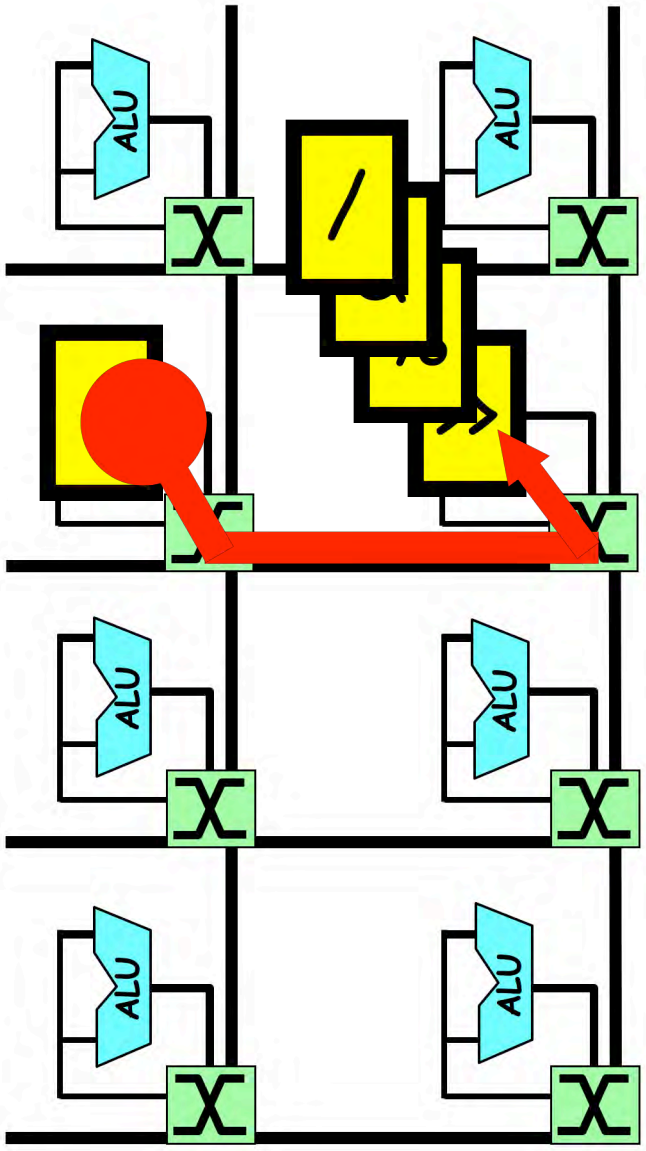
Is instruction assignment to ALUs predetermined?

Transport (Static/Dynamic)

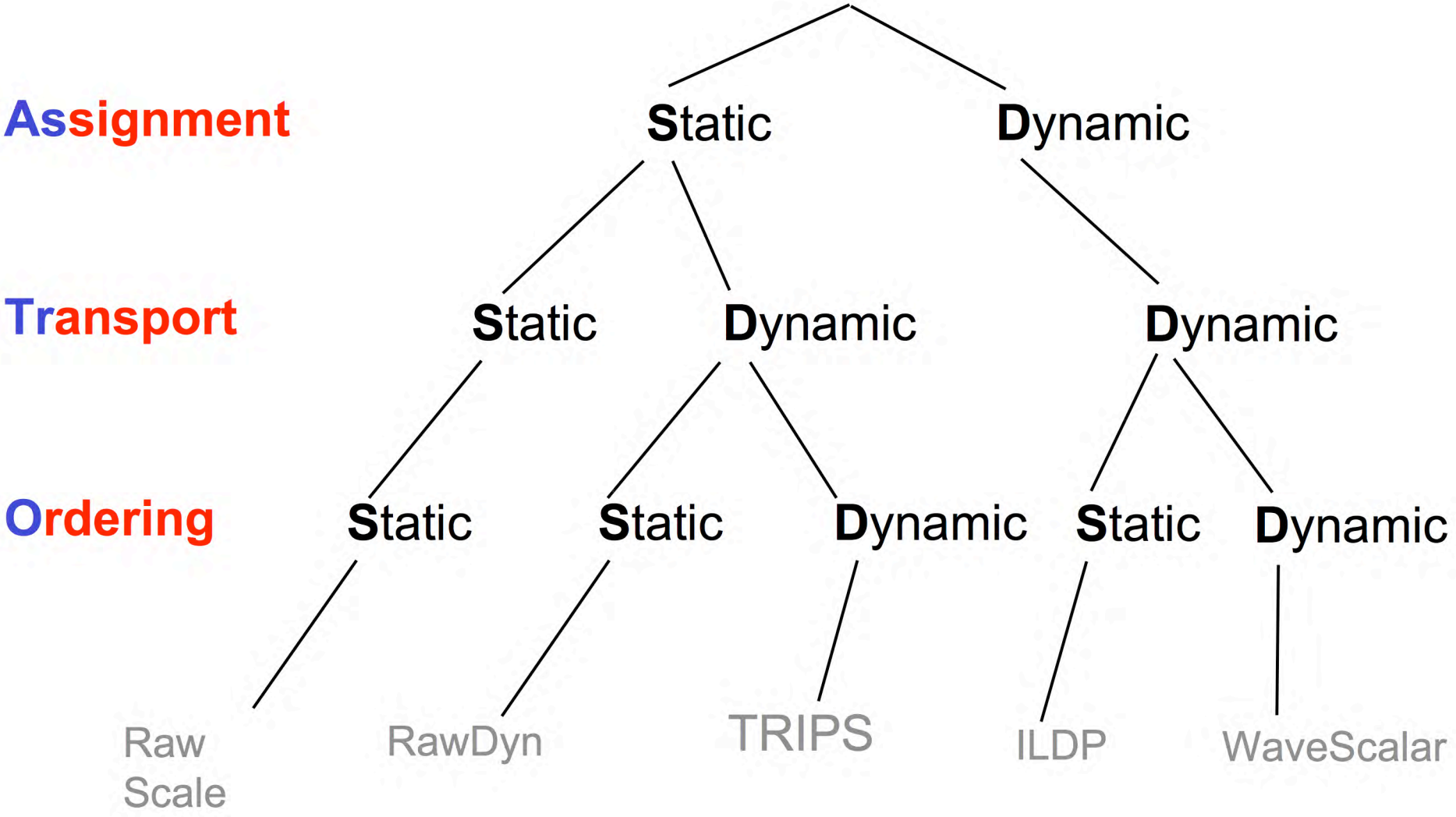
Are operand routes predetermined?

Ordering (Static/Dynamic)

Is the execution order of instructions assigned to a node predetermined?



Microprocessor SON diversity using AsTrO taxonomy



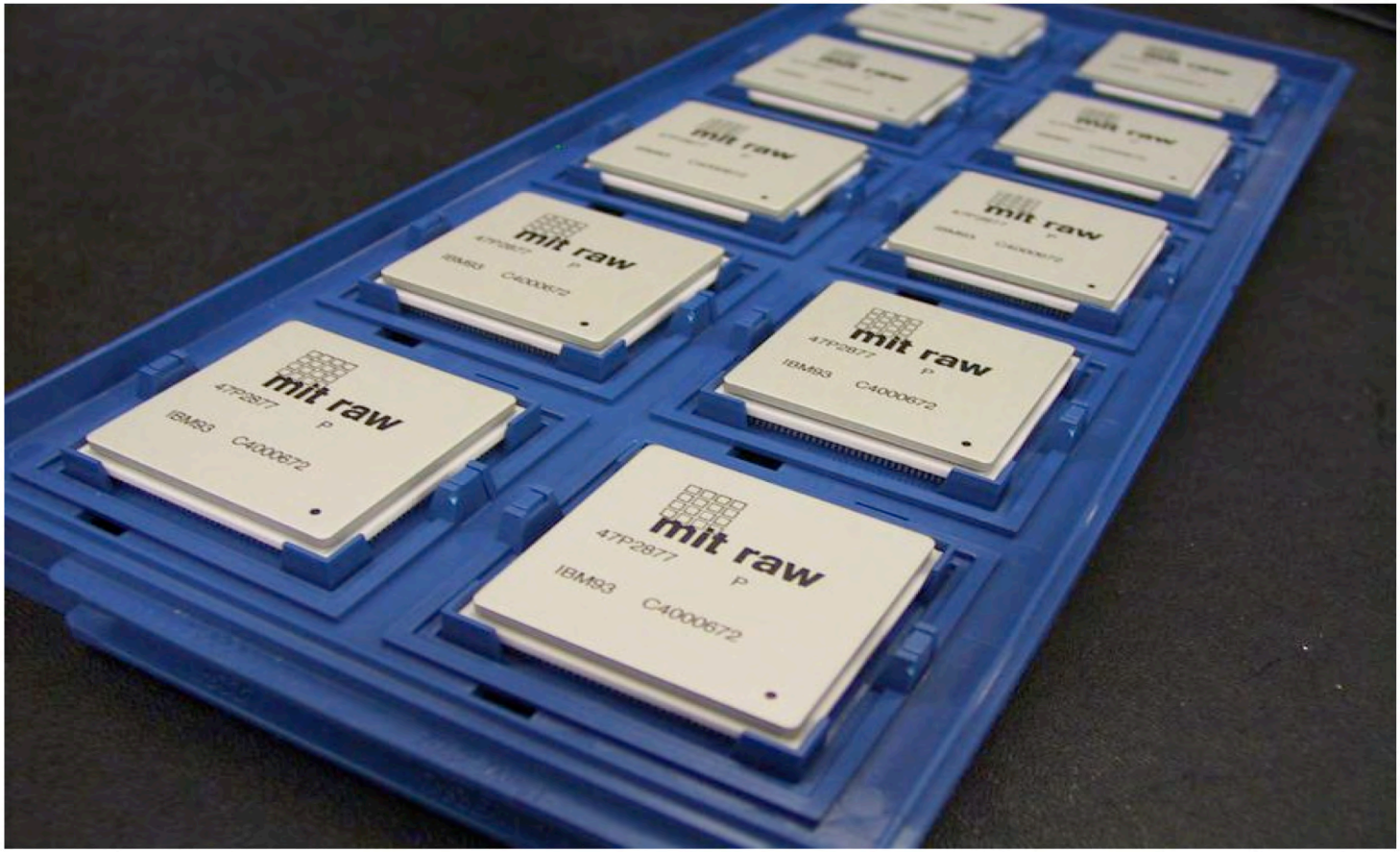
Outline

1. Scalar Operand Network and Tiled Microprocessor intro

2. Raw Architecture + SON

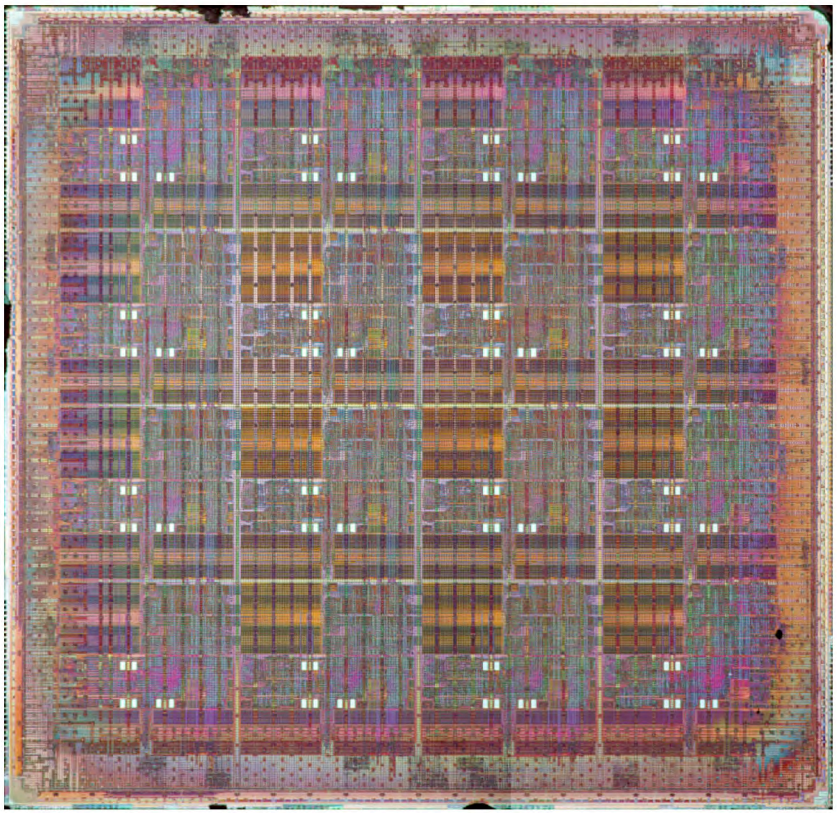
3. VLSI implementation of Raw,
   a scalable microprocessor with a scalar operand network.

Raw Chips



October 02

Raw



16 tiles (16 issue)
180 nm ASIC (IBM SA-27E)
~100 million transistors
1 million gates

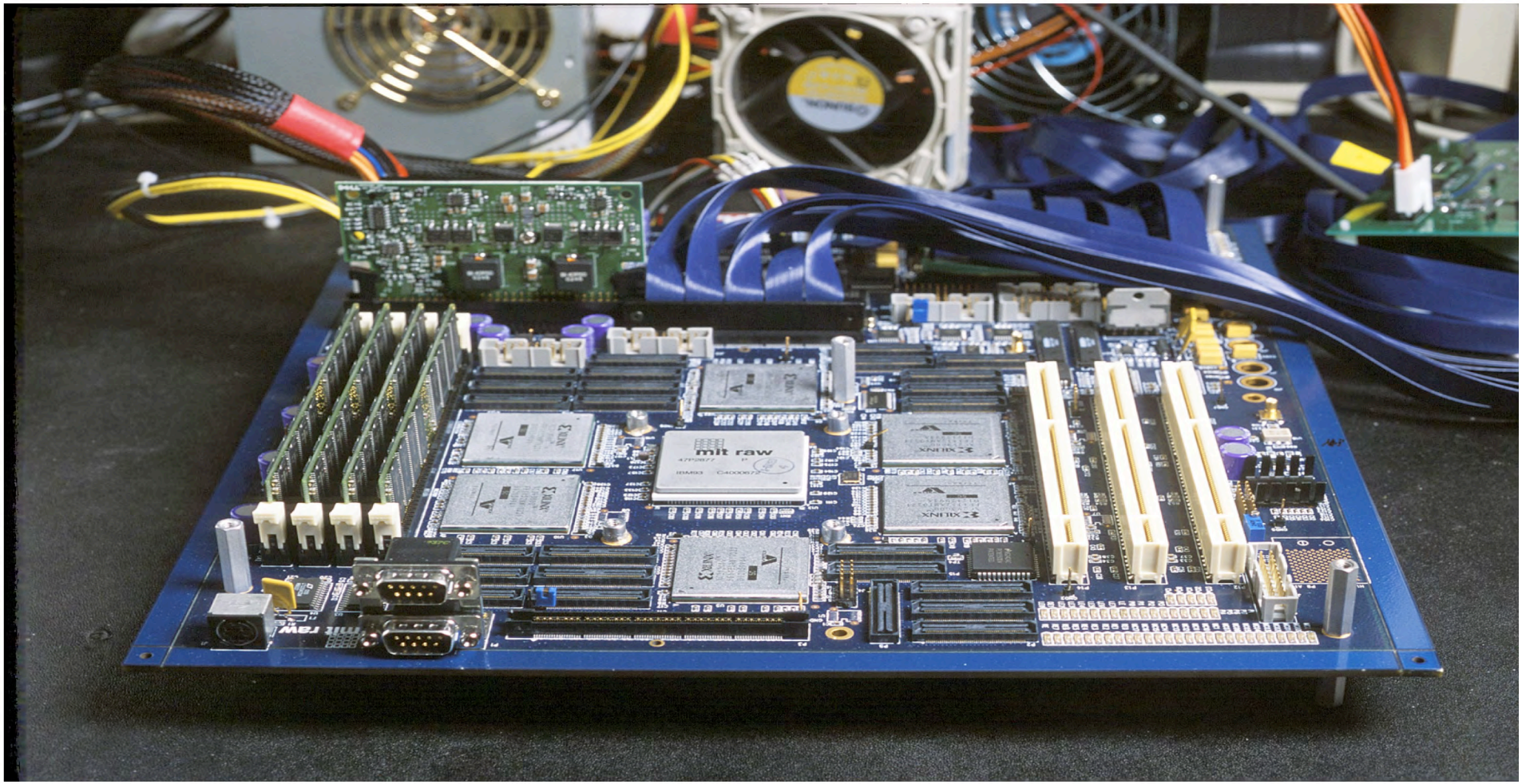
3-4 years of development
1.5 years of testing
200K lines of test code

Core Frequency:
425 MHz @ 1.8 V
500 MHz @ 2.2 V

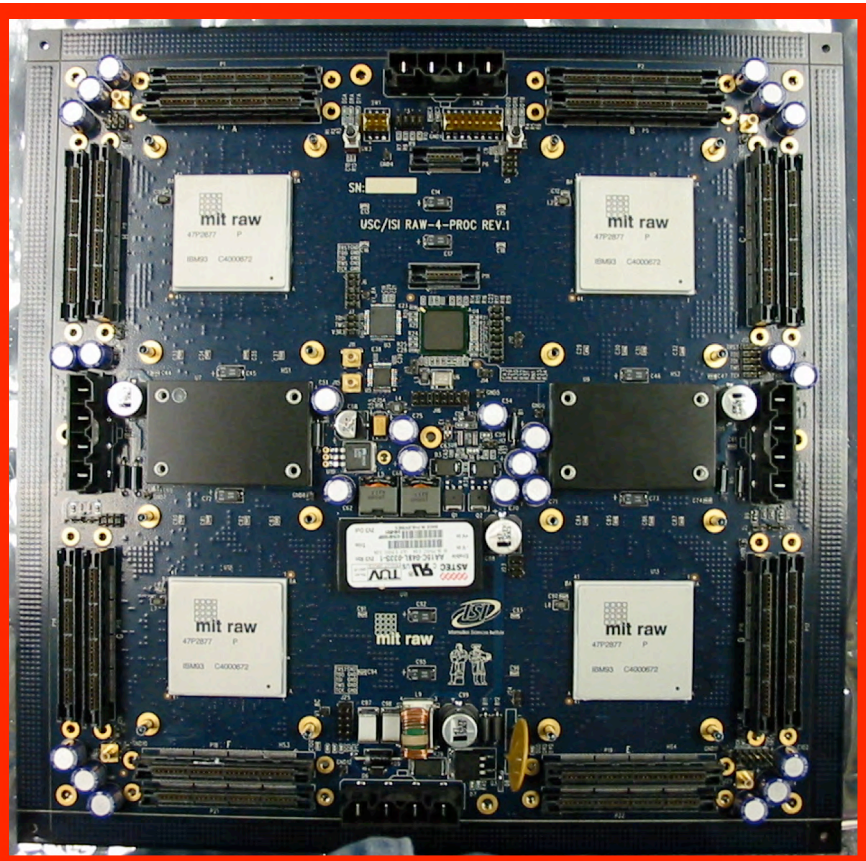
18W average power

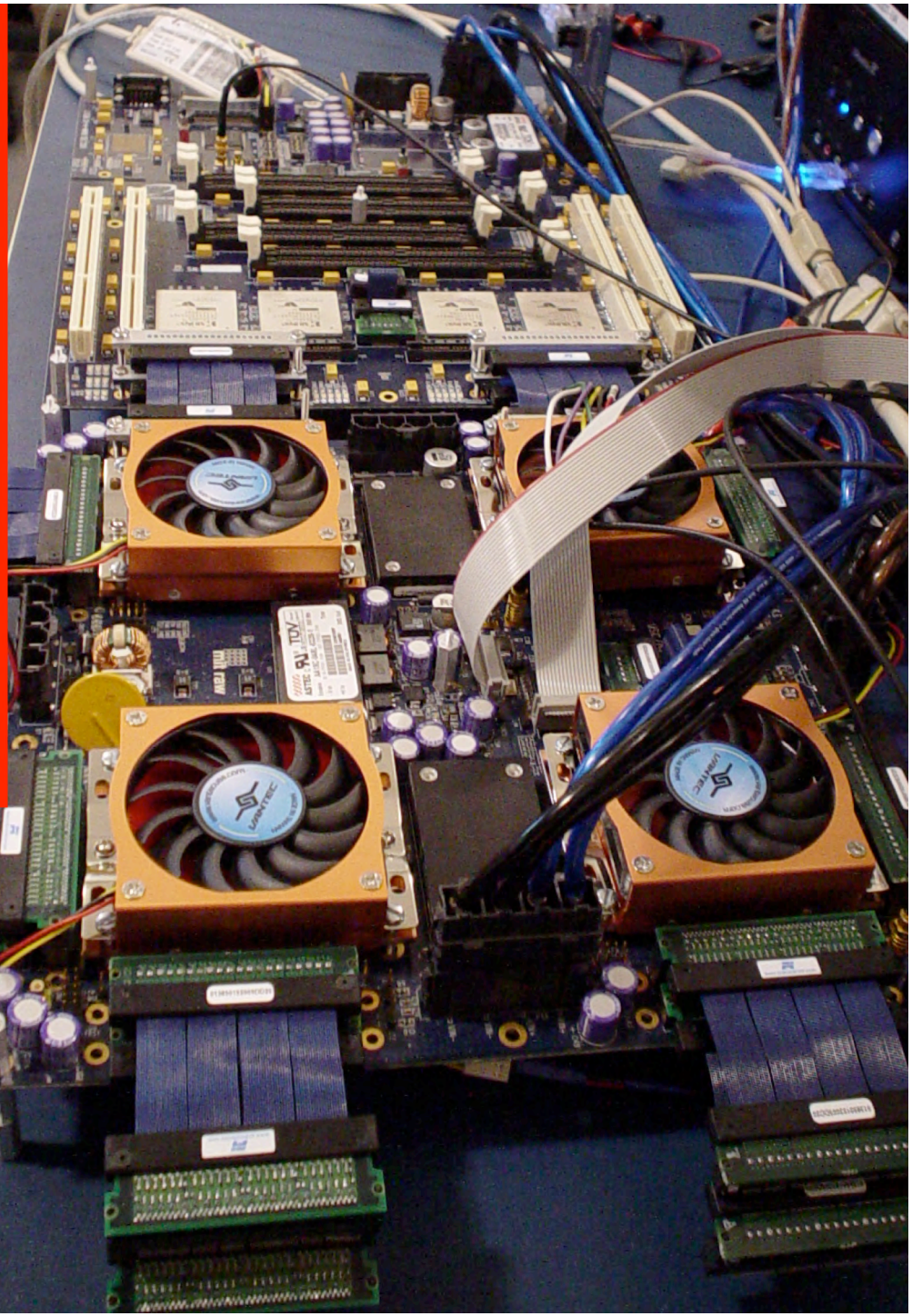
Frequency competitive with IBM-implemented PowerPCs in same process.

Raw motherboard

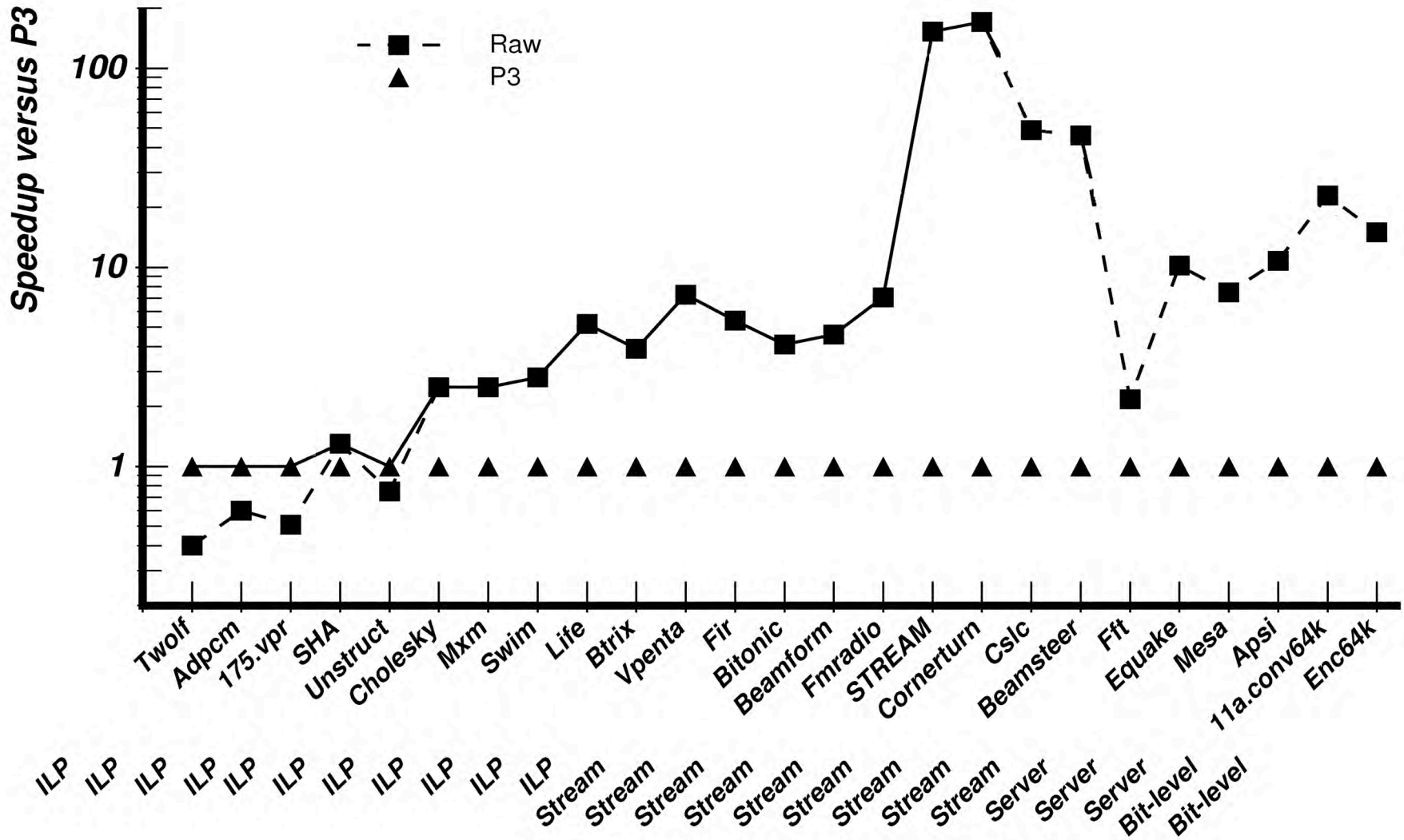


Support Chipset implemented in FPGA


SN:
USC/ISI RAW-4-PROC REV.1
mit raw
mit raw
mit raw
mit raw
mit raw

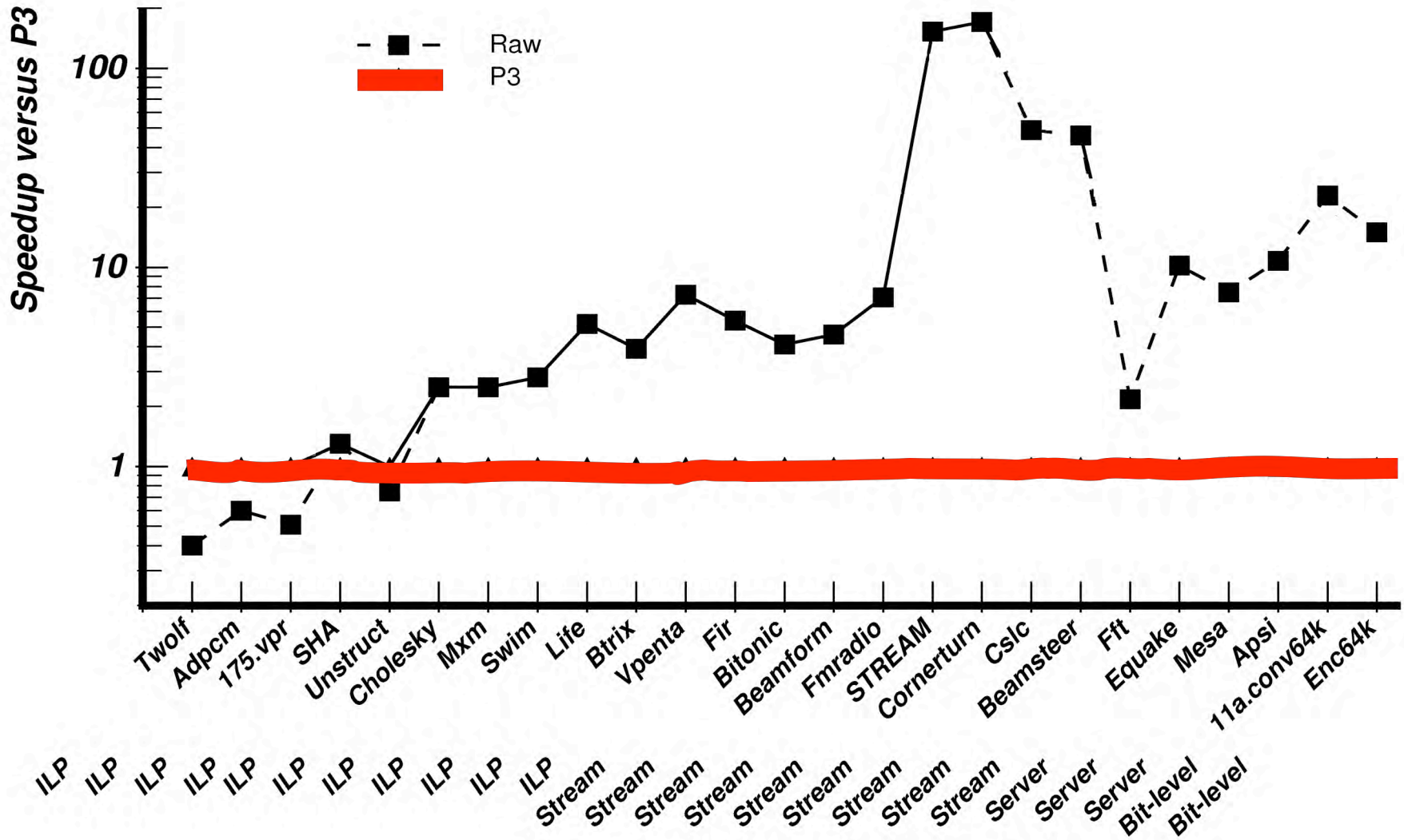


A Scalable Microprocessor in Action



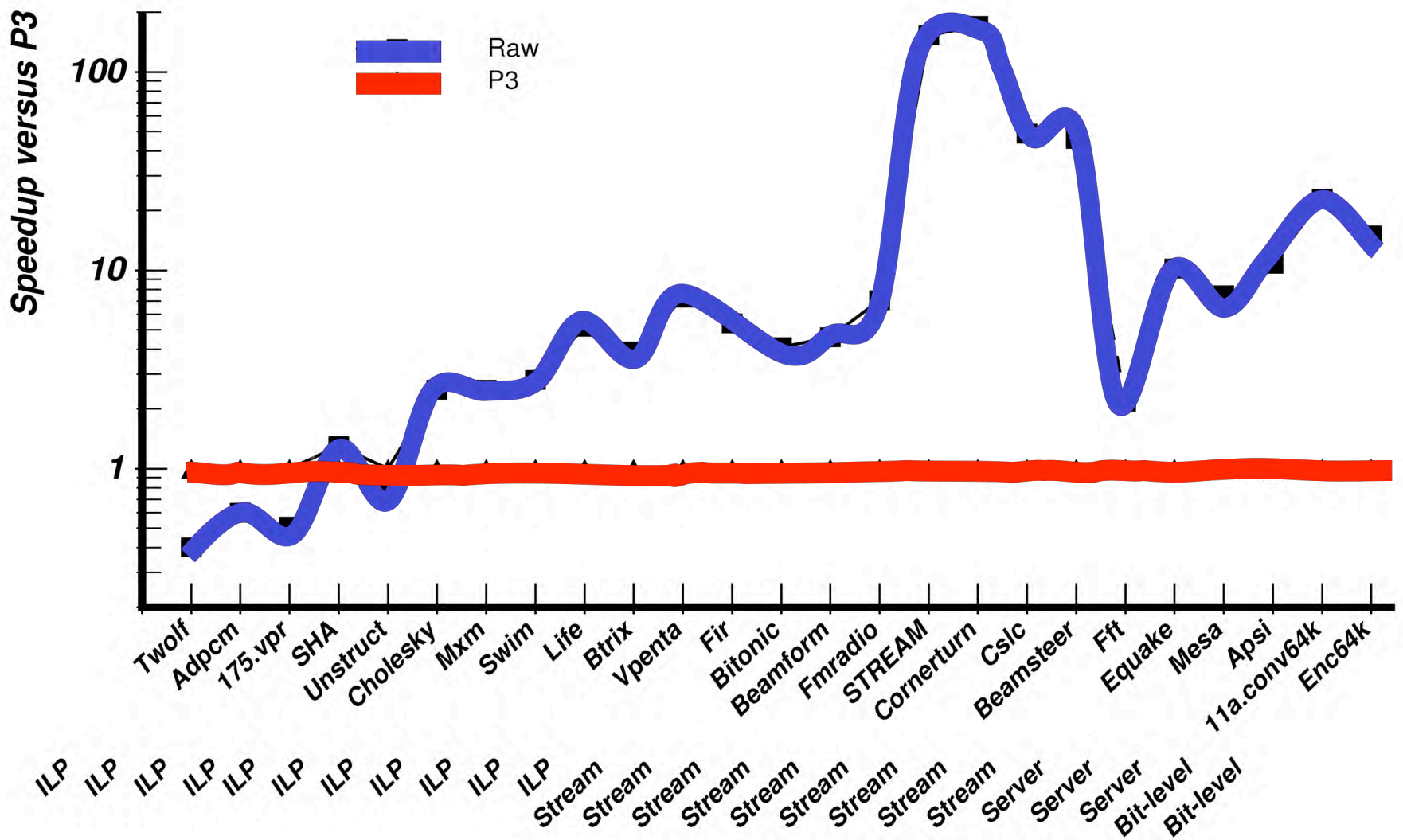
[Taylor et al, ISCA '04]

A Scalable Microprocessor in Action



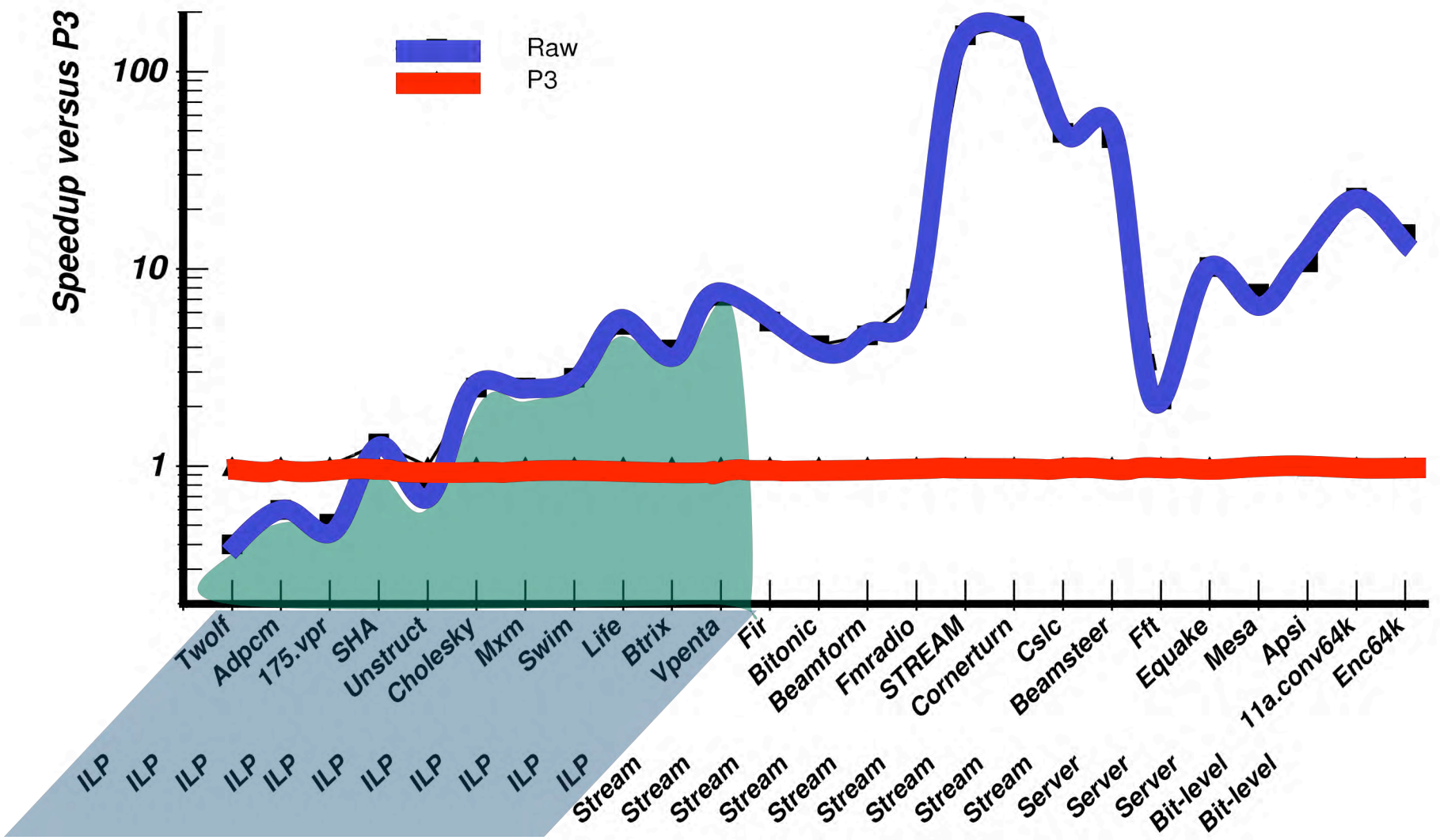
[Taylor et al, ISCA '04]

A Scalable Microprocessor in Action



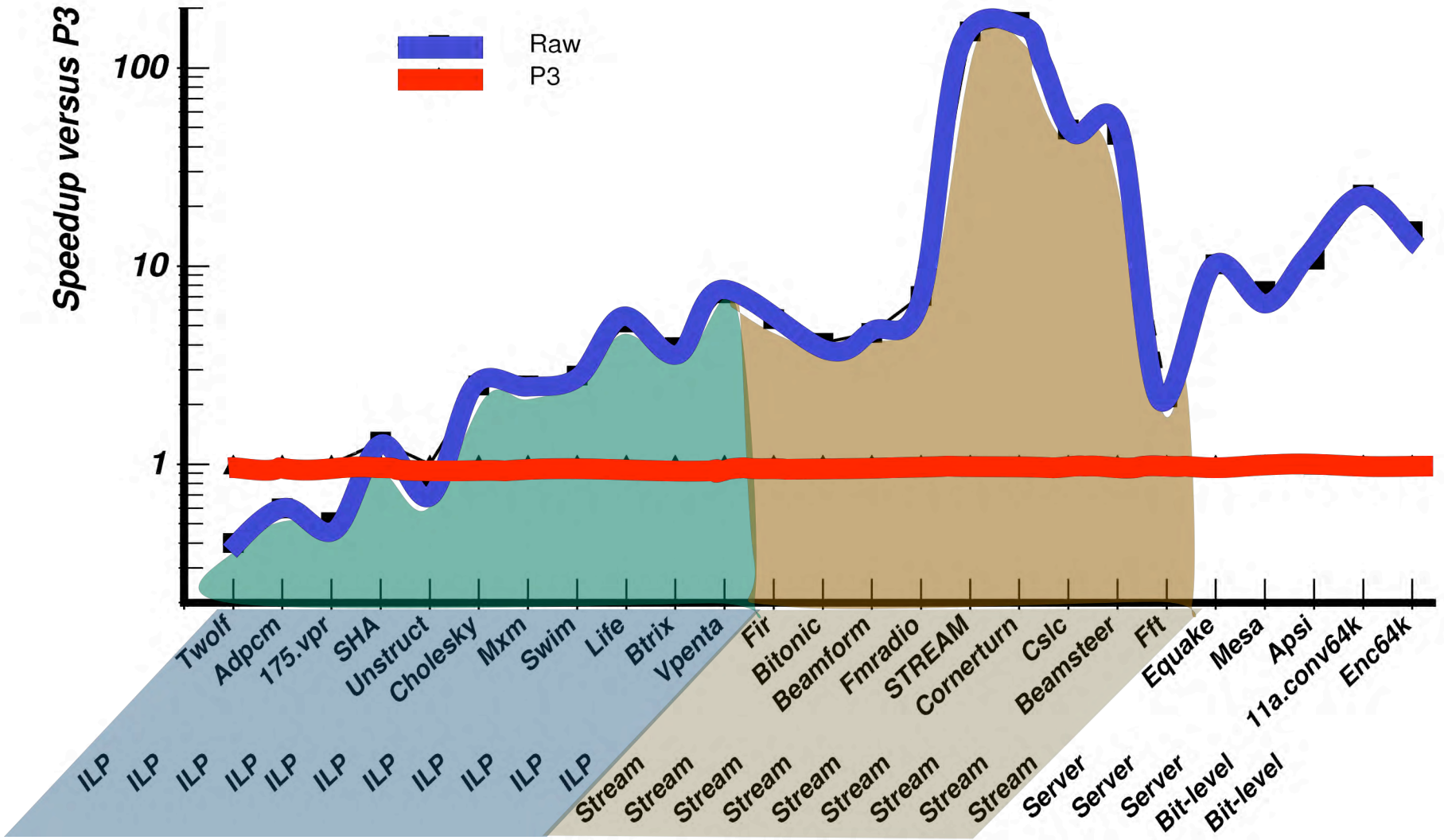
[Taylor et al, ISCA '04]

A Scalable Microprocessor in Action



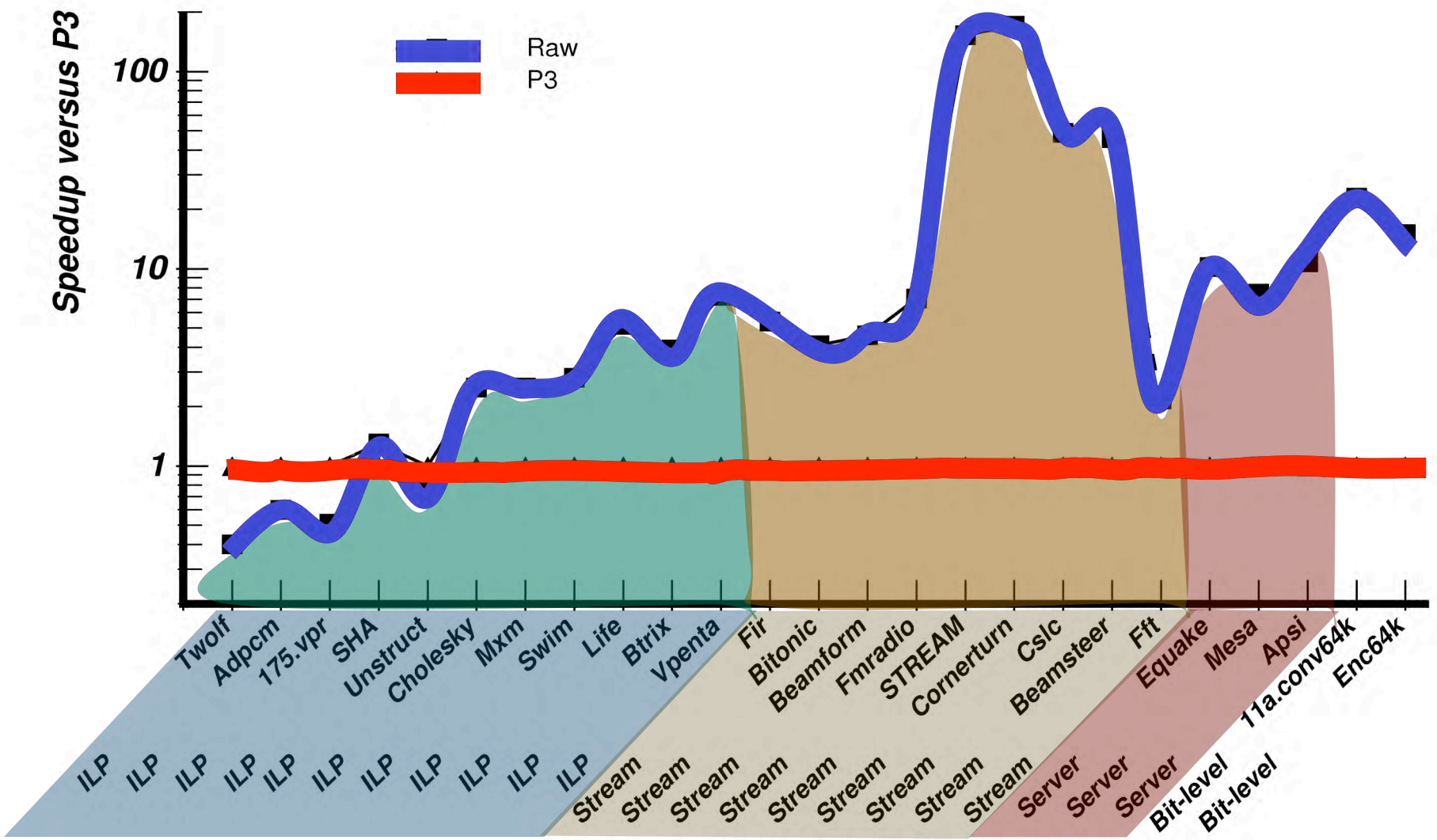
[Taylor et al, ISCA '04]

A Scalable Microprocessor in Action



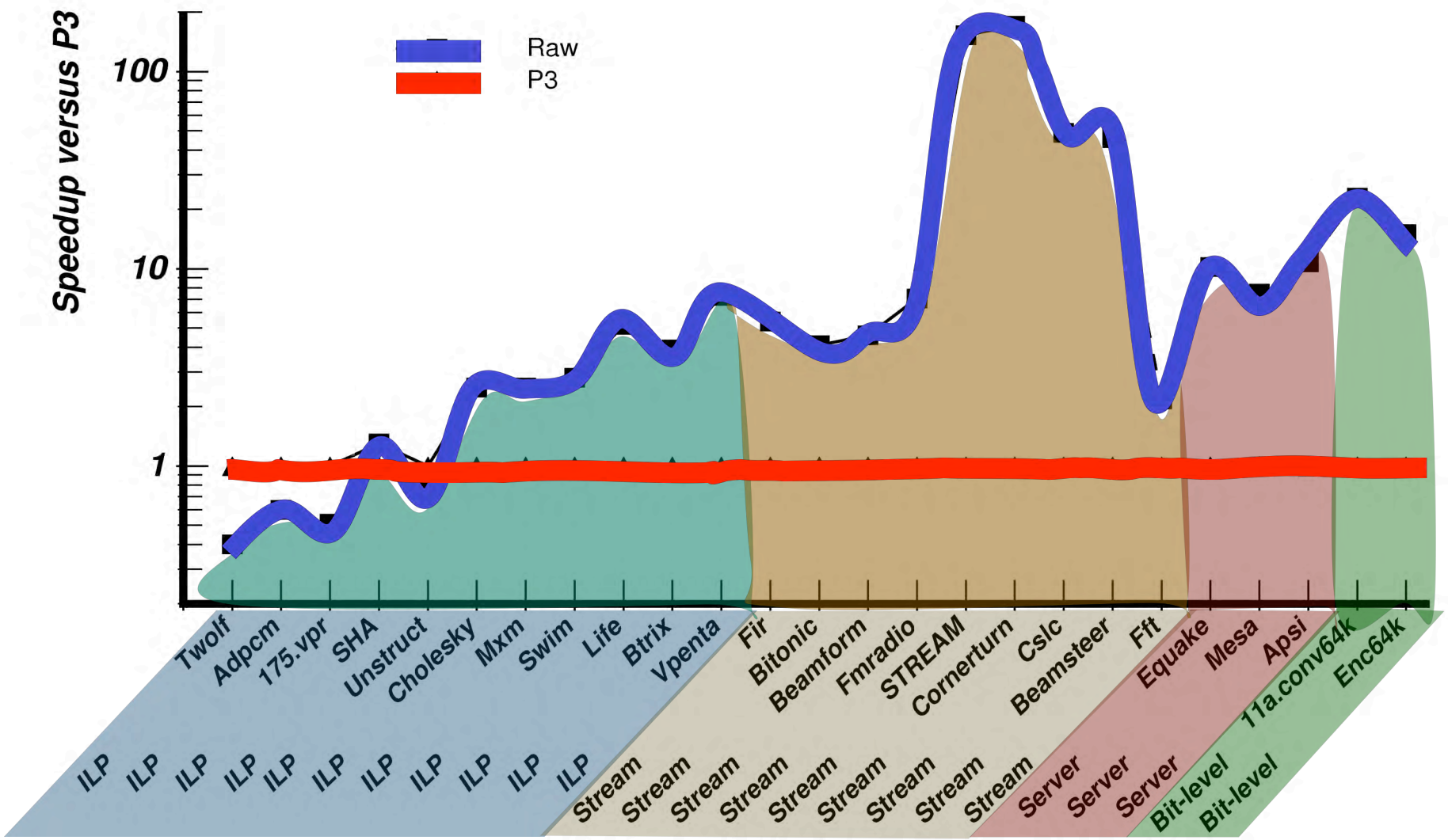
[Taylor et al, ISCA '04]

A Scalable Microprocessor in Action



[Taylor et al, ISCA '04]

A Scalable Microprocessor in Action



[Taylor et al, ISCA '04]

Conclusions

Scalability problems in general purpose processors can be addressed by tiling resources across a scalable, low-latency, low-occupancy scalar operand network (SON).

These SONs can be characterized by a 5-tuple and the AsTrO classification.

The 180 nm 16-issue Raw prototype shows the feasibility of the approach is feasible. 64+-issue is possible in today's VLSI processes.

Multicore machines could benefit by adding inter-node SON for cheap communication.