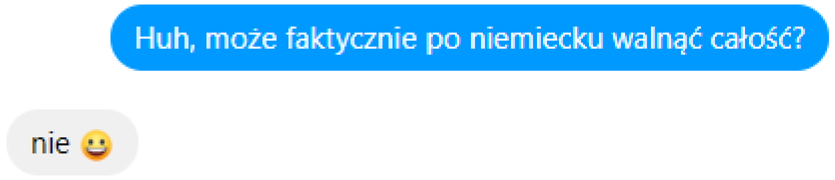
CHIPPER

Ein kurzer Vortrag über pufferlose Ablenkrouter

Zbigniew Drozd



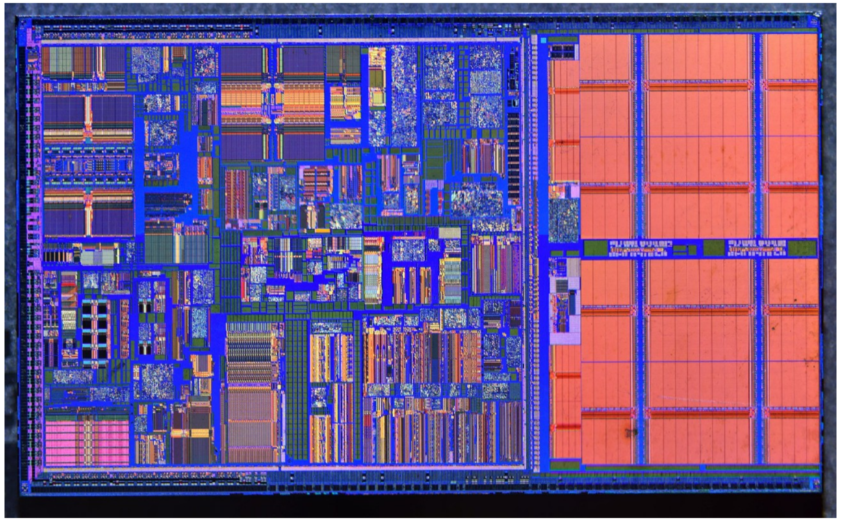
Huh, może faktycznie po niemiecku walnąć całość?

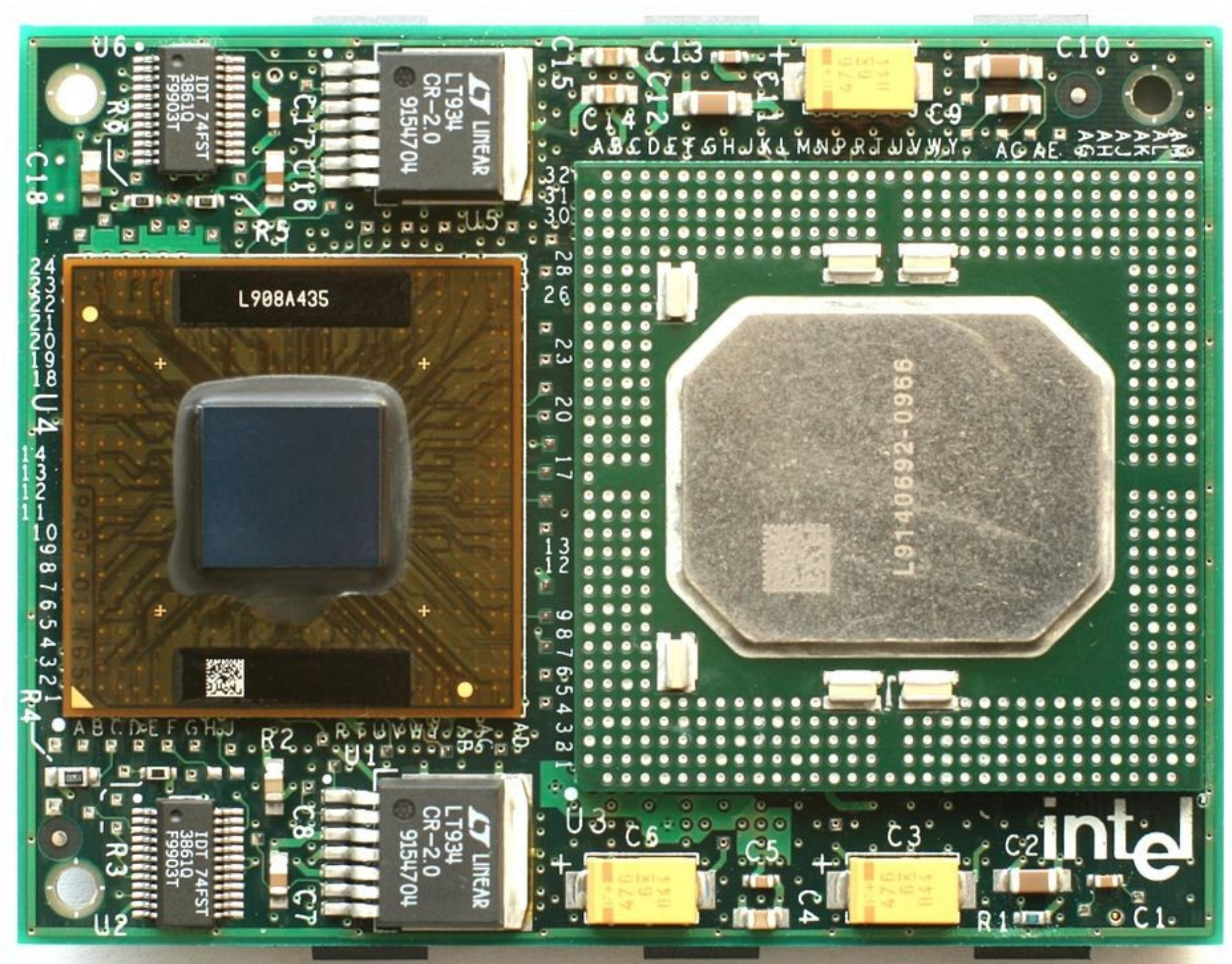
nie 😀

The presentation plan:

- People screaming in opposition of the presentation being in German.
- Quick talk about the simpler times, AMD Athlon 64 X2 3800+
- The use of interconnects today
- Multicore processors, AMD and Intel microarchitectures
- Multicore processors with absurd number of cores
- US8531943
- Problems with internal routers
- CHIPPER

Quick (pre)history of processors



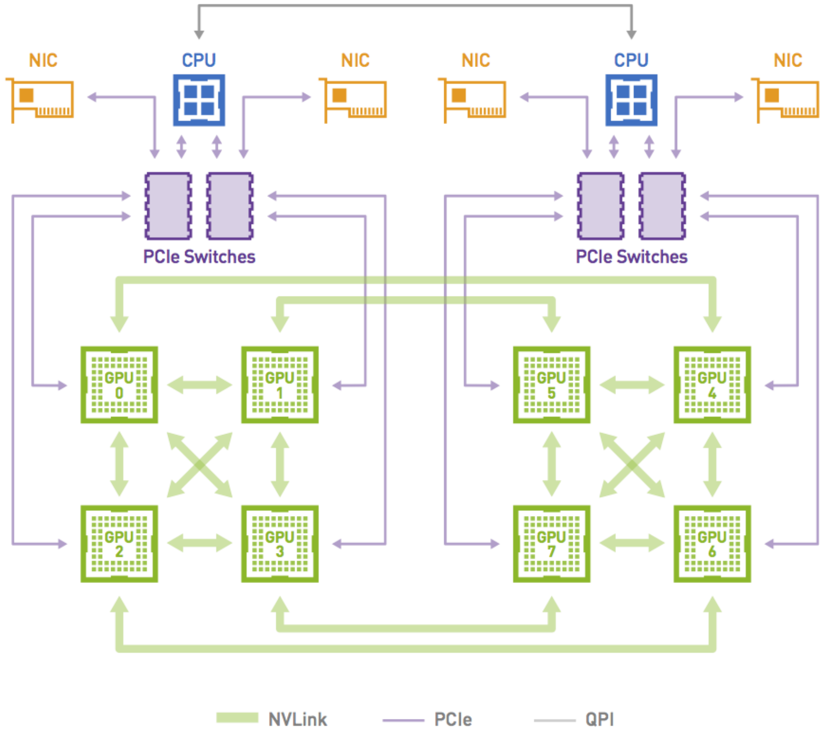


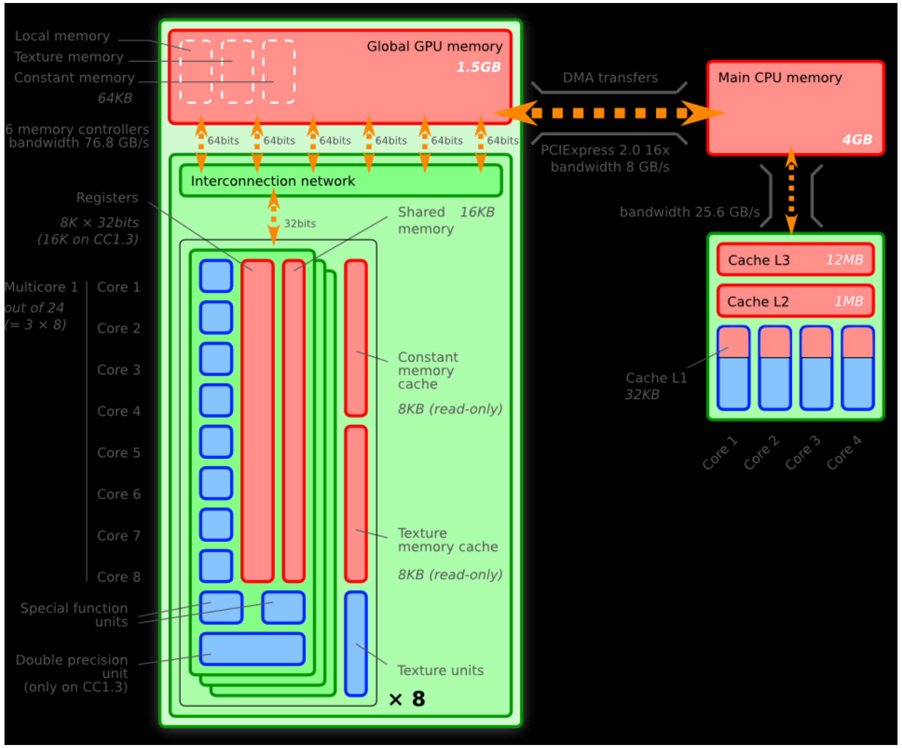
Pentium II Overdrive.
Deschutes core on the left,
512 KB of L2 cache on the right.

Please take a moment to appreaciate the fact that we live in a time when L3 is integrated into the die.

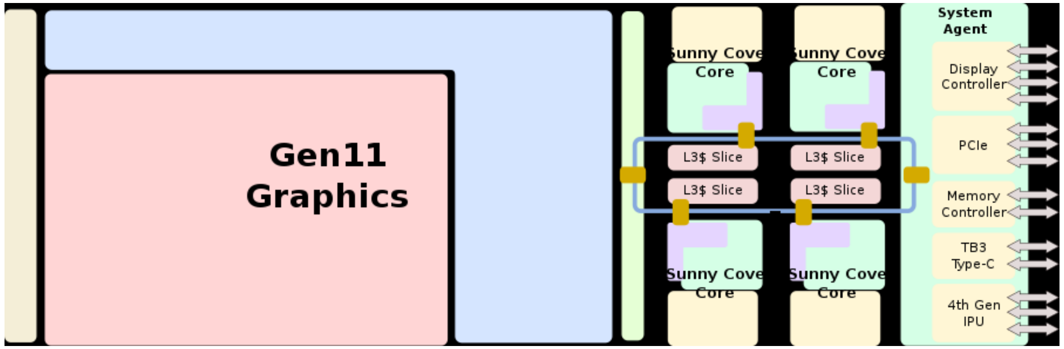
Interconnects today

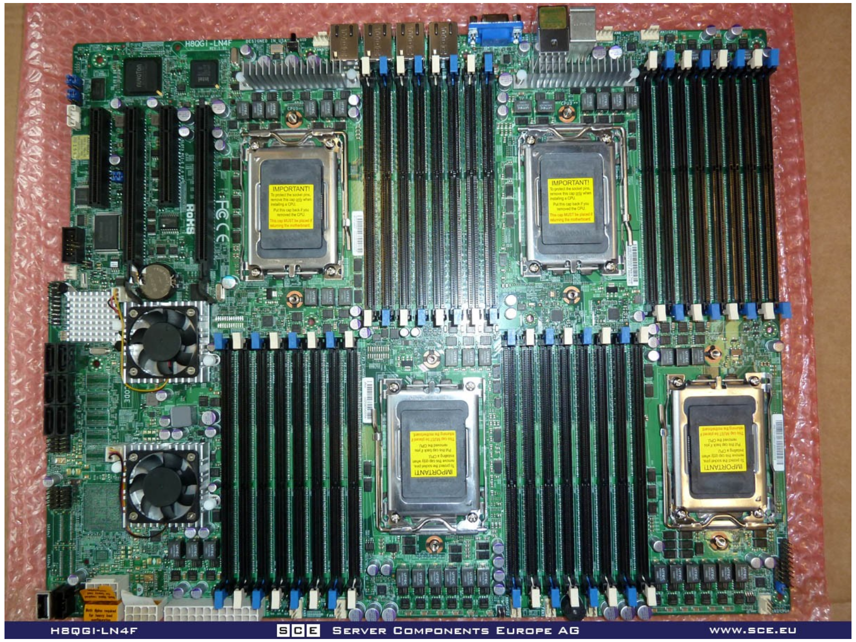
Nvidia

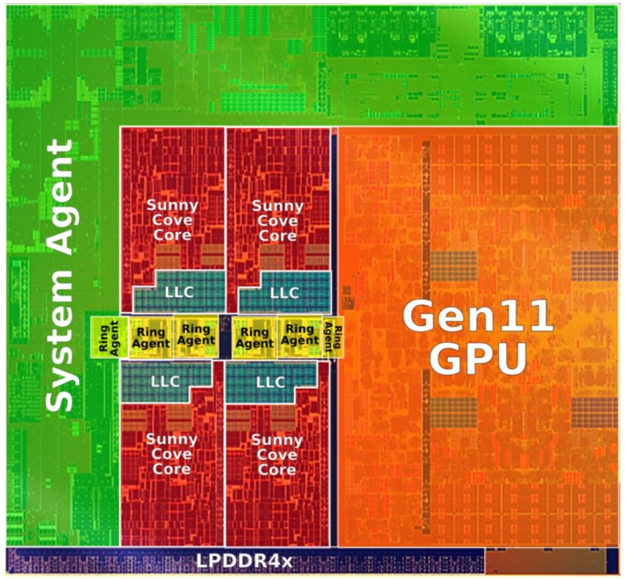

NIC
CPU
NIC
NIC
CPU
NIC
PCIe Switches
PCIe Switches
GPU 0
GPU 1
GPU 5
GPU 4
GPU 2
GPU 3
GPU 7
GPU 6
NVLink
PCIe
QPI

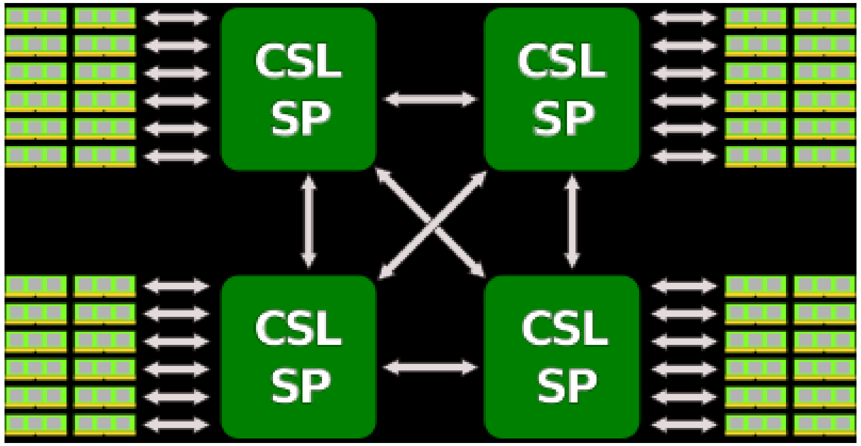

Local memory
Texture memory
Constant memory
64KB
Global GPU memory
1.5GB
6 memory controllers
bandwidth 76.8 GB/s
64bits
Interconnection network
Registers
8K × 32bits
(16K on CC1.3)
32bits
Shared 16KB
memory
Multicore 1
out of 24
(= 3 × 8)
Core 1
Core 2
Core 3
Core 4
Core 5
Core 6
Core 7
Core 8
Constant
memory
cache
8KB (read-only)
Texture
memory cache
8KB (read-only)
Special function
units
Double precision
unit
(only on CC1.3)
Texture units
× 8
DMA transfers
PCIExpress 2.0 16x
bandwidth 8 GB/s
Main CPU memory
4GB
bandwidth 25.6 GB/s
Cache L3 12MB
Cache L2 1MB
Cache L1
32KB
Core 1
Core 2
Core 3
Core 4

Intel

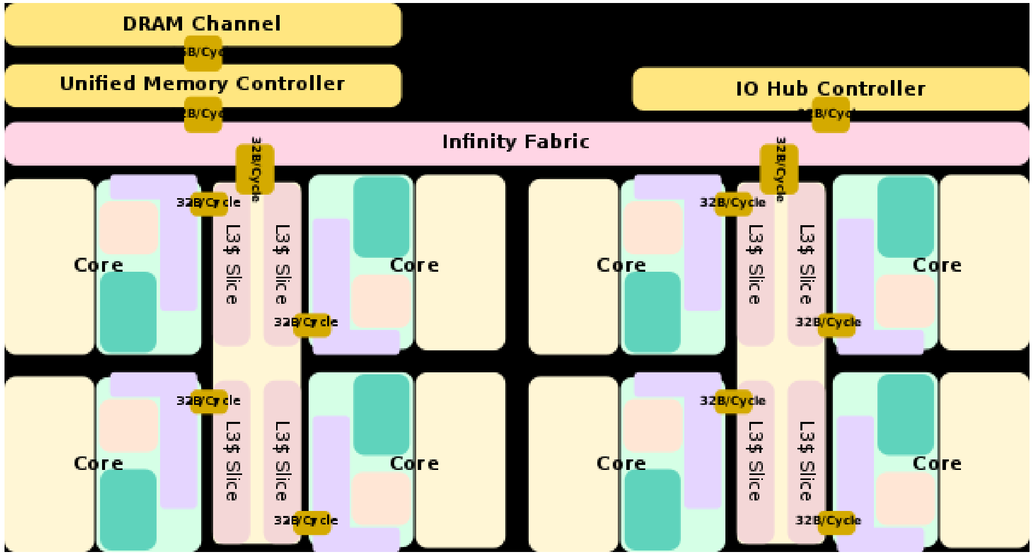

Gen11 Graphics
Sunny Cove Core
Sunny Cove Core
L3$ Slice
L3$ Slice
L3$ Slice
L3$ Slice
Sunny Cove Core
Sunny Cove Core
System Agent
Display Controller
PCIe
Memory Controller
TB3 Type-C
4th Gen IPU

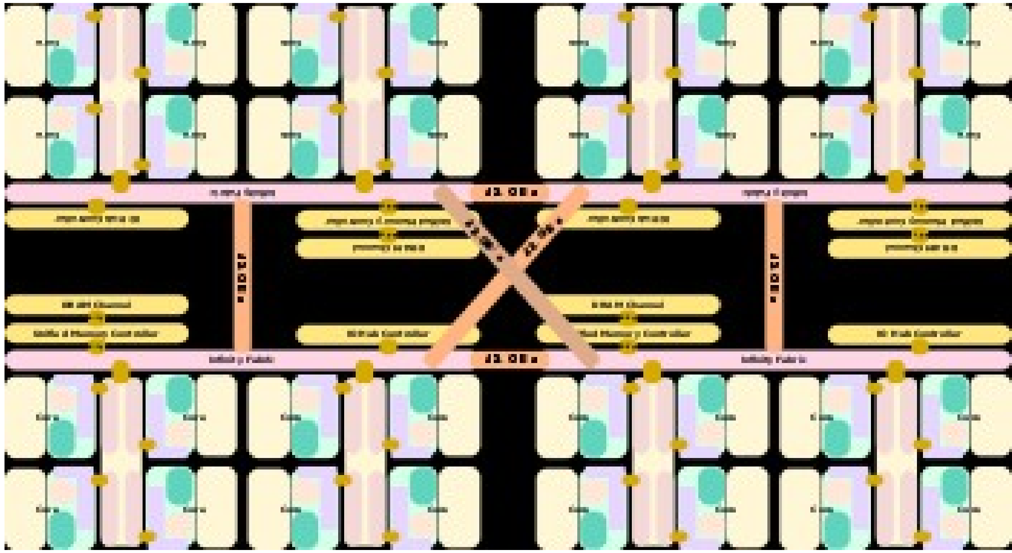

H8QGi-LN4F
SCE Server Components Europe AG
www.sce.eu


System Agent
Sunny Cove Core
Sunny Cove Core
LLC
LLC
Ring Agent
Ring Agent
Ring Agent
Ring Agent
Ring Agent
Ring Agent
LLC
LLC
Sunny Cove Core
Sunny Cove Core
Gen11 GPU
LPDDR4x


CSL SP
CSL SP
CSL SP
CSL SP

AMD

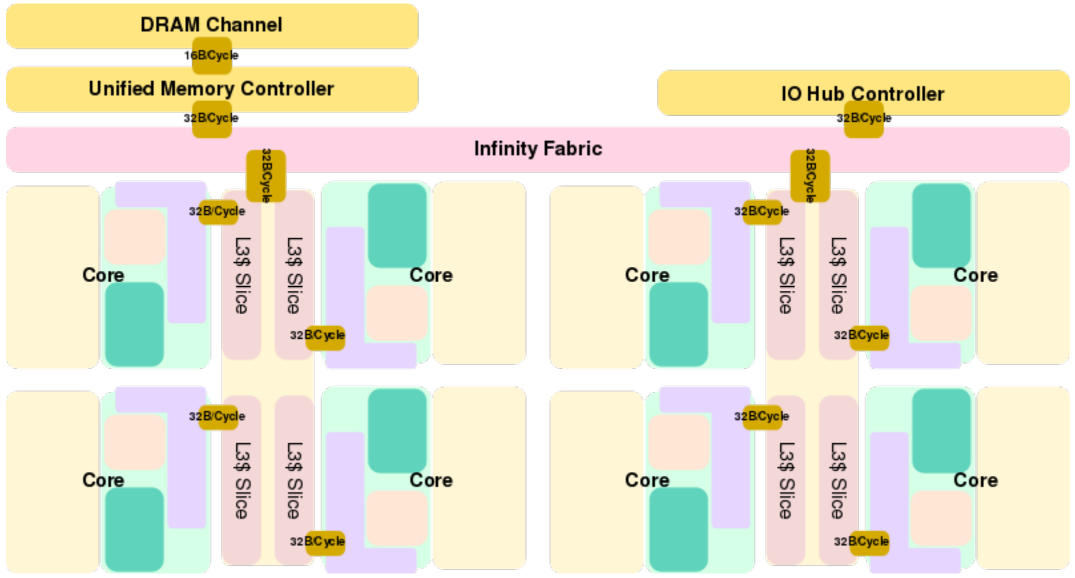

DRAM Channel
Unified Memory Controller
IO Hub Controller
Infinity Fabric
32B/Cycle
Core
L3$ Slice

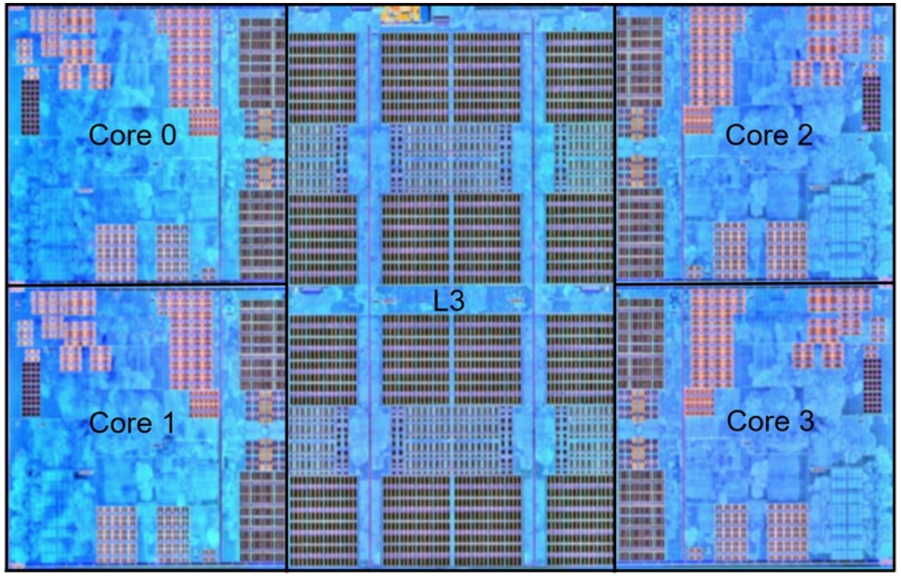


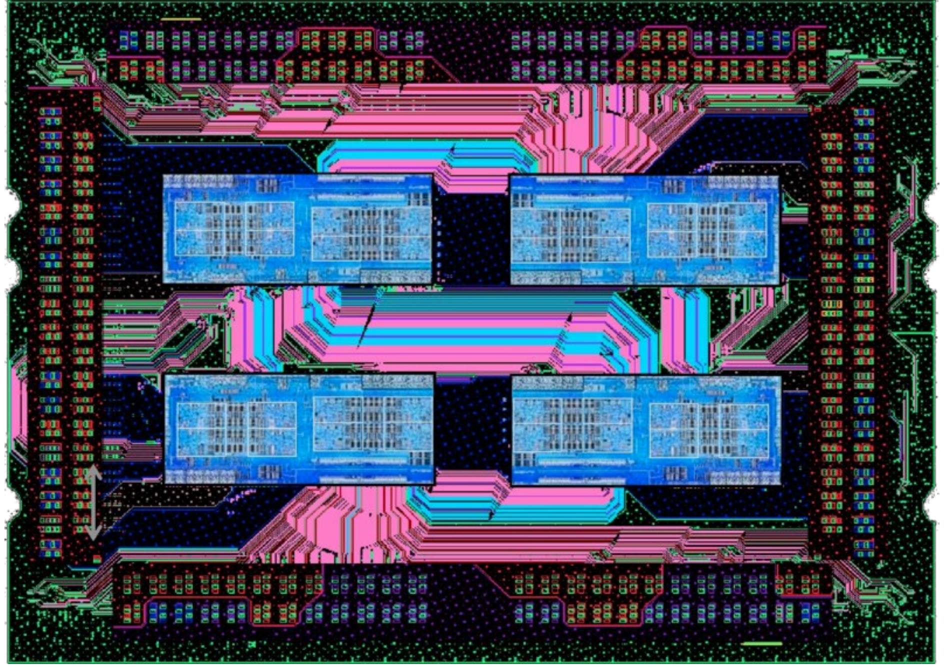
What the heck is SMBus?

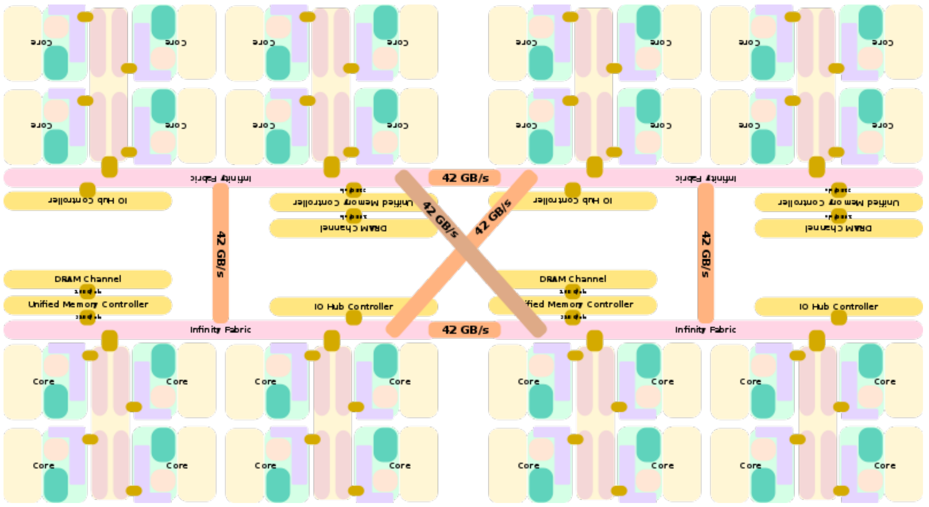
Multicore processors

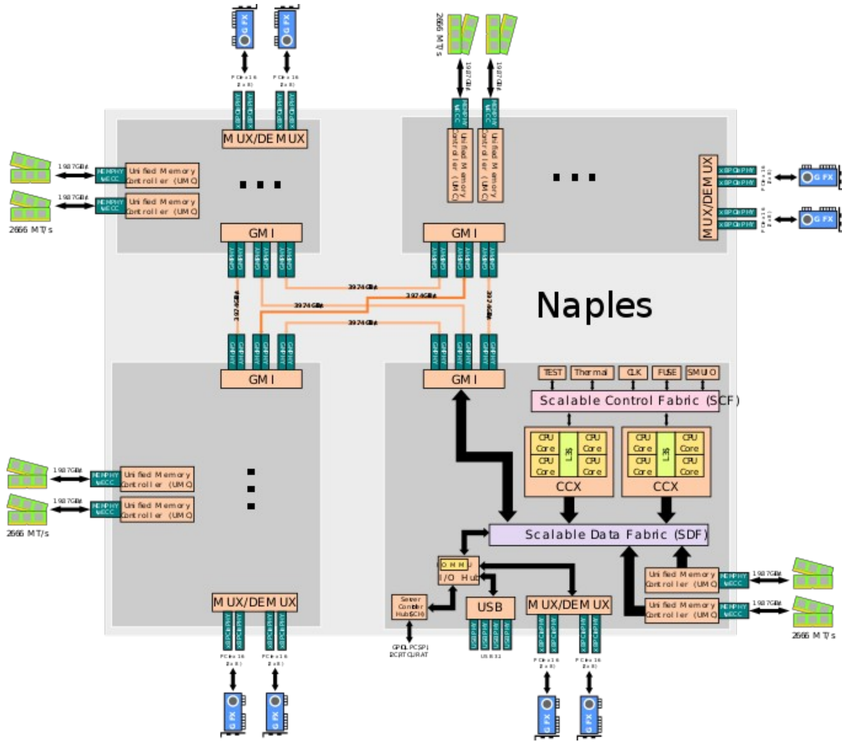
AMD Infinity Fabric








Core
Infinity Fabric
IO Hub Controller
Unified Memory Controller
DRAM Channel
42 GB/s
42 GB/s
42 GB/s
42 GB/s
42 GB/s
42 GB/s


Naples
MUX/DEMUX
GMI
Unified Memory Controller (UMC)
2666 MT/s
19.87 GB/s
39.74 GB/s
PCIe x16
TEST
Thermal
CLK
FUSE
SMUIO
Scalable Control Fabric (SCF)
CPU Core
L3
CCX
Scalable Data Fabric (SDF)
I/O Hub
USB
Server Controller Hub (SCH)

Intel uuuh

- Petition for Intel to chill the fuck out and stop conceaving a microarchitecture every 2.3 nS
- On wikichip 69 pages of intel microarchitectures exist, with additional 19 for GPU’s

Intel uuuh

- Petition for Intel to chill the fuck out and stop conceaving a microarchitecture every 2.3 nS
- On wikichip 69 pages of intel microarchitectures exist, with additional 19 for GPU’s
- Yet no Intel GPU’s exist (at least for commercial sale)

- 19 micro architectures that are actually interesting
- 12 of them have any description of interconnects
- Falling into one of three cathegories

But first, a game

But first, a game

Airmont, Chiovano, Knights Ferry, Palm cove, Whiskey lake,

Montvale, Rocket lake, Ice Lake, Polaris,

Alder lake, Coffe lake, Lakefield, Granite rapids,

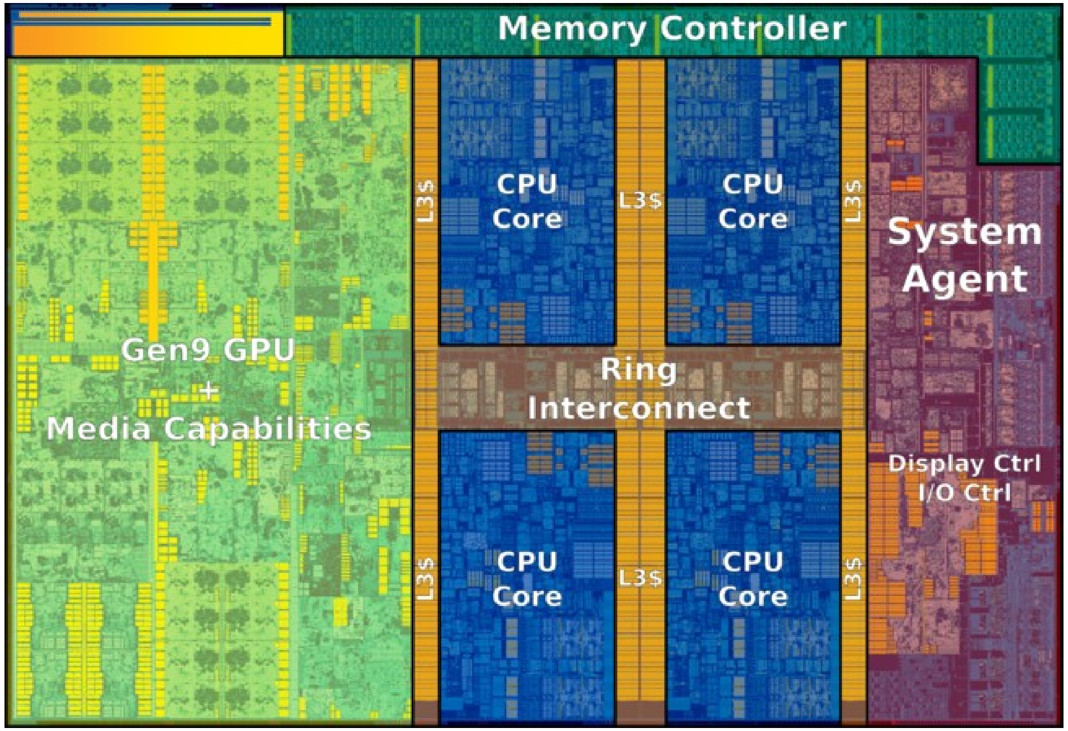
Rock creek, Willow cove, Snow ridge, Haswell,

Cannon lake, Knights hill, Sapphire Rapids,

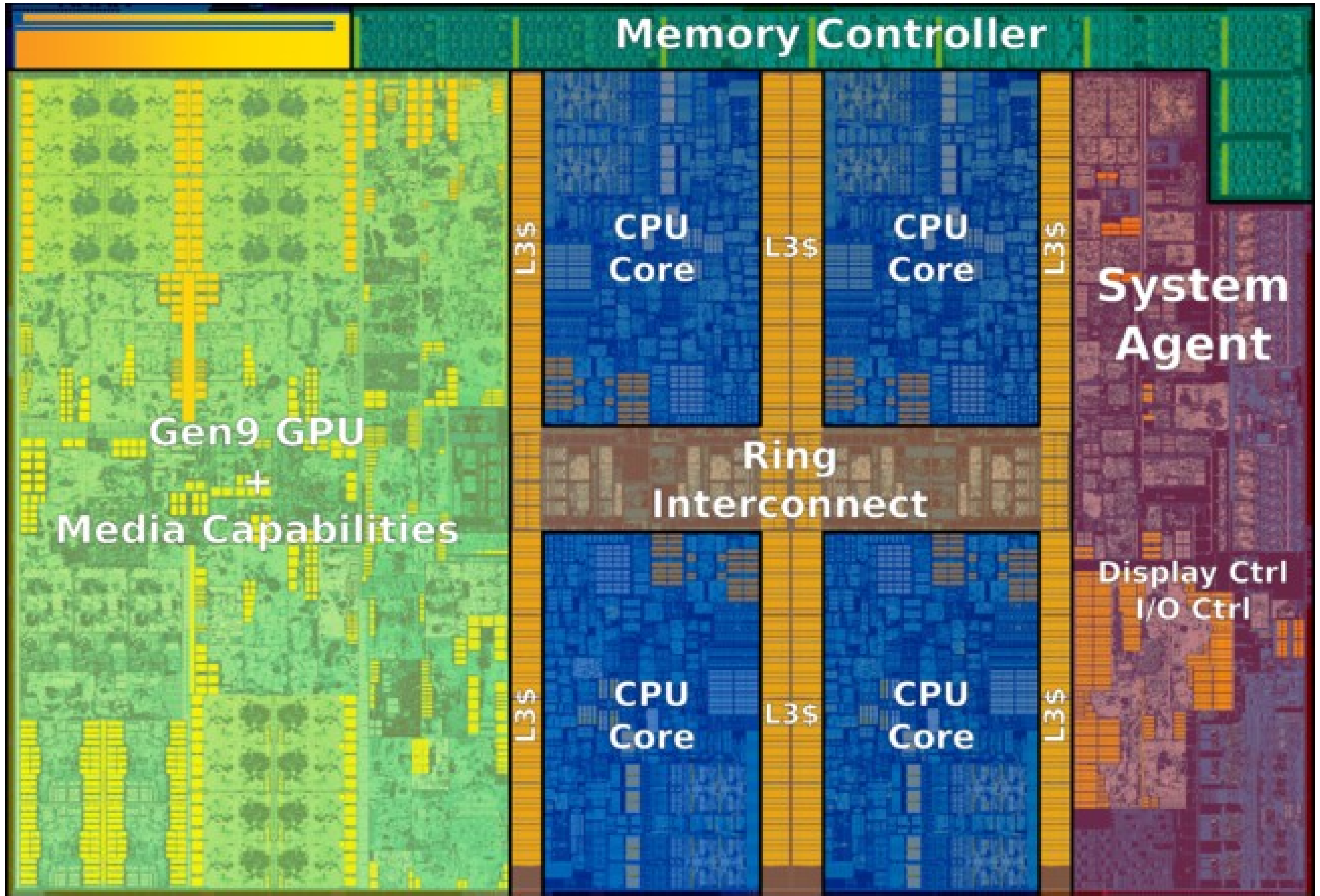
Tiger lake, Tukwila, Saltwell, Kaby Lake, Meteor Lake

Intel – Agents

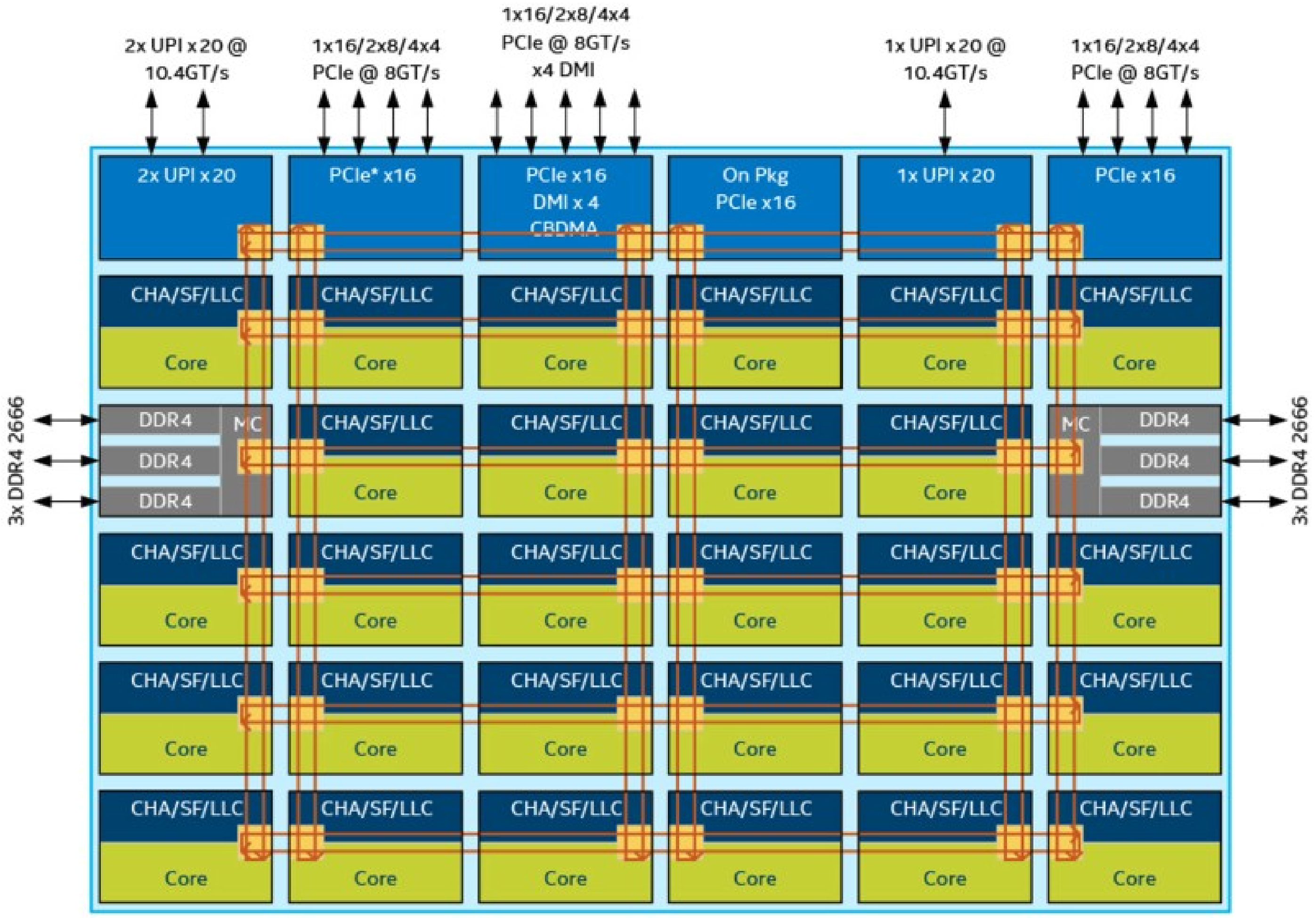
- System agent
- Ring agent



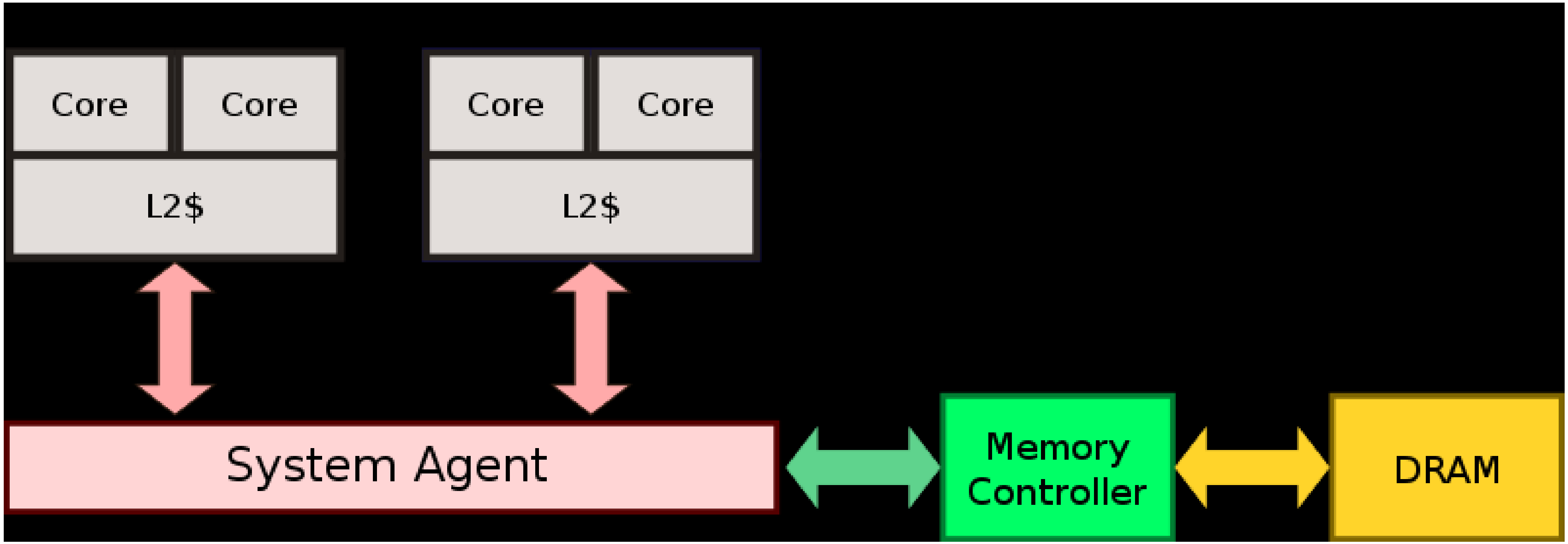
Intel – Ring interconnect



Intel – mesh interconnect

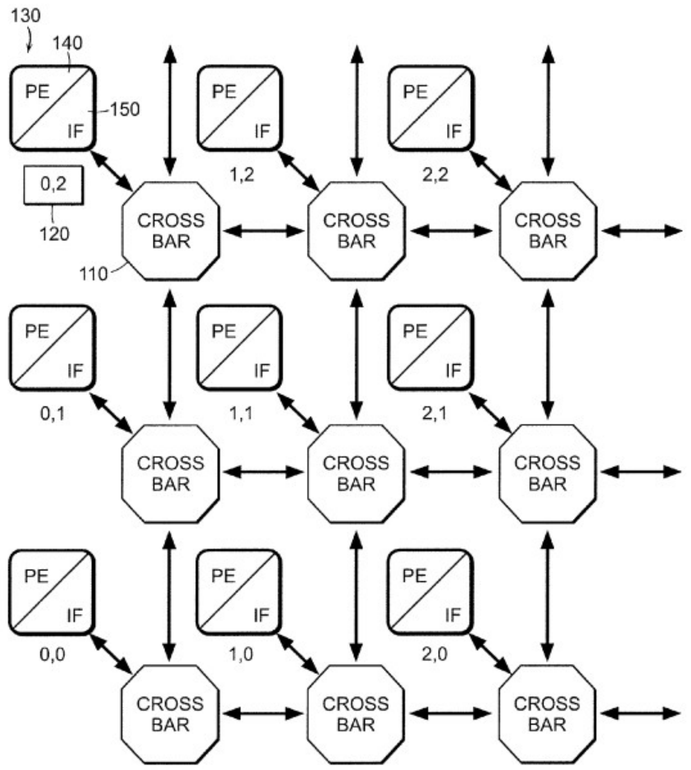


Intel – slapping cores directly to system agent



High count core processors

US8531943



Epiphany-V: A 1024 processor 64-bit RISC System-On-Chip

Andreas Olofsson

(give it a read, it’s an awesome paper about chip design)

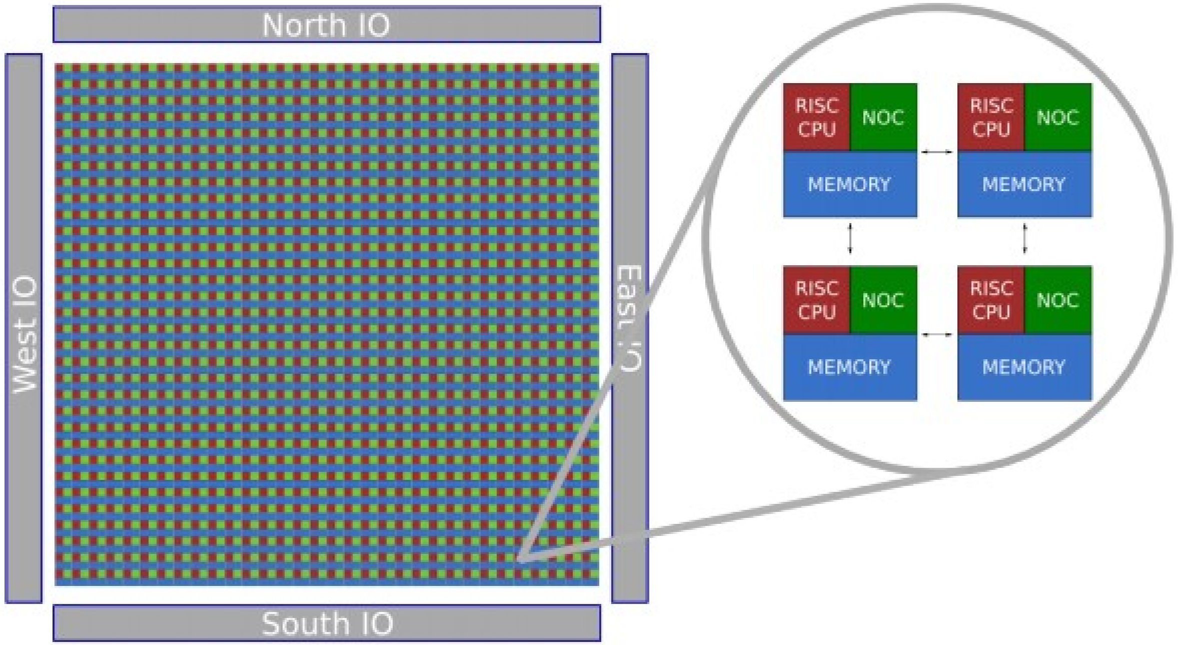


Figure 1: Epiphany-V Overview

III.B Memory Architecture

The Epiphany 64-bit memory map is split into 1 Billion 1MB memory regions, with 30 bits dedicated to x,y,z addressing. The complete Epiphany memory map is flat, distributed, and shared by all processors in the system. Each individual memory region can be used by a single processor or aggregated as part of a shared memory pool. The Epiphany architecture uses multi-banked software-managed scratch-pad memory at each processor node. On every clock cycle, a processor node can:

- Fetch 8 bytes of instructions
- Load/store 8 bytes of data
- Receive 8 bytes from another processor in the system
- Send 8 bytes to another processor in the system

III.C Network-On-Chip

The Epiphany-V mesh Network-on-Chip (“emesh”) consists of three independent 136-bit wide mesh networks. Each one of the three NOCs serve different purposes:

- rmesh: Read request packets
- cmesh: On-chip write packets
- xmesh: Off-chip write packets

Epiphany NOC packets are 136 bits wide and transferred between neighboring nodes in one and a half clock cycles. Packets consist of 64 bits of data, 64 bits of address, and 8 bits of control. Read requests puts a second 64-bit address in place of the data to indicate destination address for the returned read data.

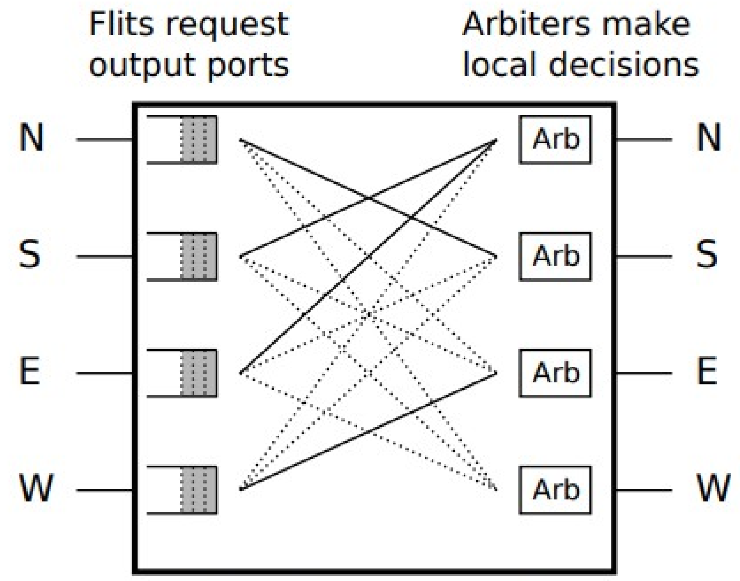
Network-On-Chip routing follows a few simple, static rules. At every hop, the router compares its own coordinate address with the packet’s destination address. If the column addresses are not equal, the packet gets immediately routed to the south or north; otherwise, if the row addresses are not equal, the packet gets routed to the east or west; otherwise the packet gets routed into the hub node.

Each routing node consists of a round robin five direction arbiter and a single stage FIFO. Single cycle transaction push-back enables network stalling without packet loss.

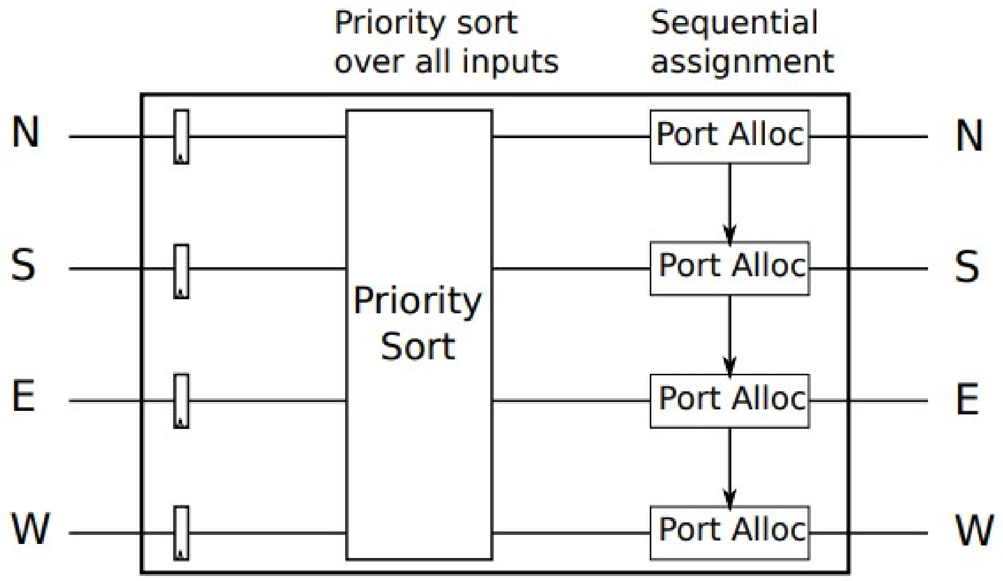
How much do interconnects cost?

So, why should we go bufferless?

- Buffers take up lots of space. If you get rid of them, the routers are smaller.
- Interconnects consume ~40% of the chip’s power while contributing nothing to the computation. Power used by the interconnect is basicly wasted, so we should minimise it.
- Desighning IC’s is hard. Manufacturing them, even harder. So, since bufferless networks are simpler, we reduce the development costs, and reduce the chances for a mistake.



(a) Buffered router port allocator



(b) Bufferless deflection router port allocator

Figure 1: Port allocator structures: deflection routing requires more complex logic with a longer critical path.

Injection and ejection

Since each router has N inputs and N outputs data injection to the network is non-trivial

Deadlock prevention

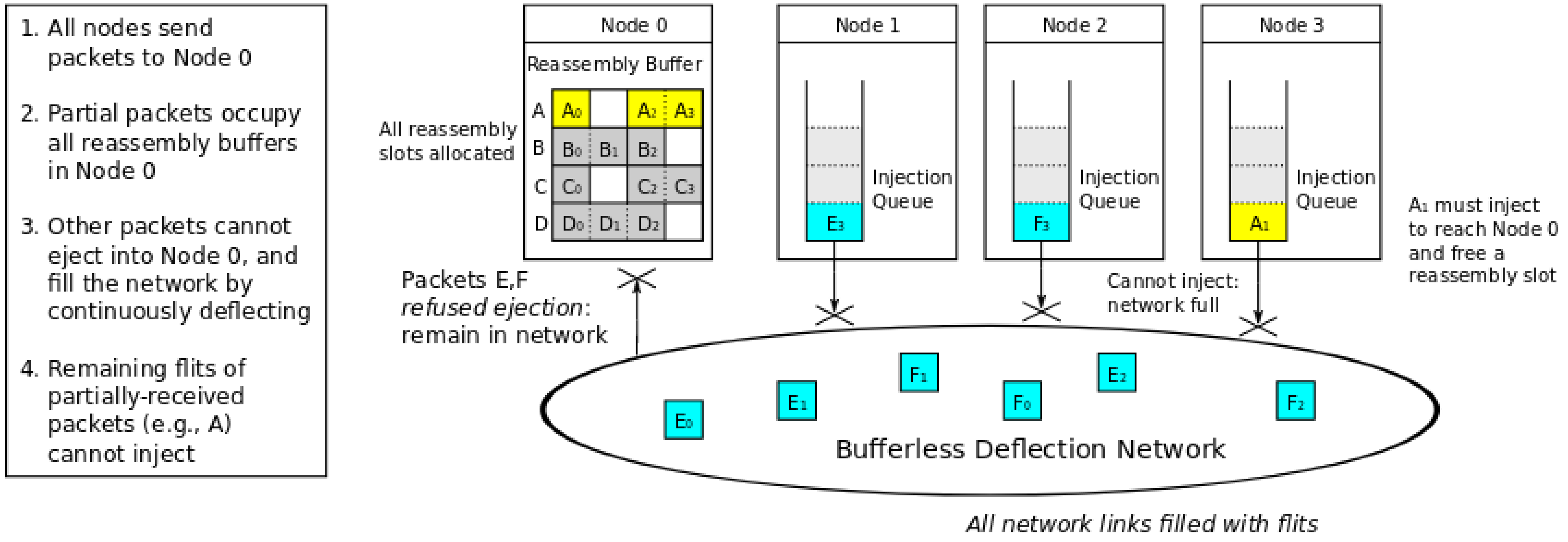

1. All nodes send packets to Node 0
2. Partial packets occupy all reassembly buffers in Node 0
3. Other packets cannot eject into Node 0, and fill the network by continuously deflecting
4. Remaining flits of partially-received packets (e.g., A) cannot inject
Node 0
Reassembly Buffer
A
B
C
D
A0
A2
A3
B0
B1
B2
C0
C2
C3
D0
D1
D2
All reassembly slots allocated
Packets E,F refused ejection: remain in network
Node 1
Node 2
Node 3
Injection Queue
Injection Queue
Injection Queue
E3
F3
A1
A1 must inject to reach Node 0 and free a reassembly slot
Cannot inject: network full
E0
E1
F1
F0
E2
F2
Bufferless Deflection Network
All network links filled with flits

Figure 2: Deadlock due to reassembly-buffer overflow.

CHIPPER router architecture

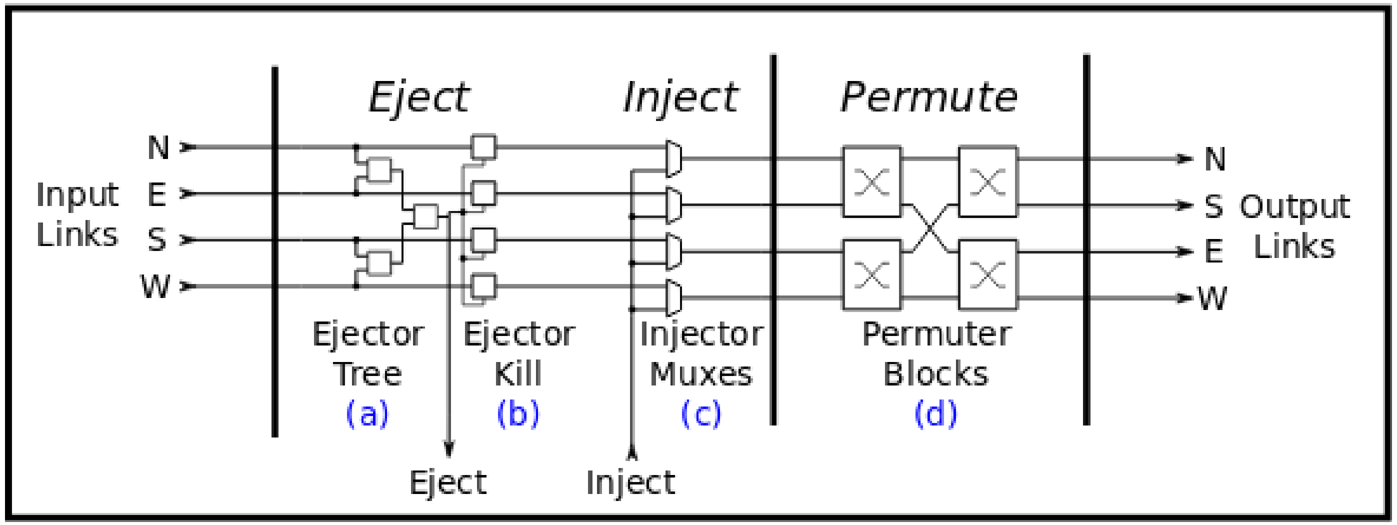
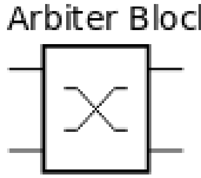
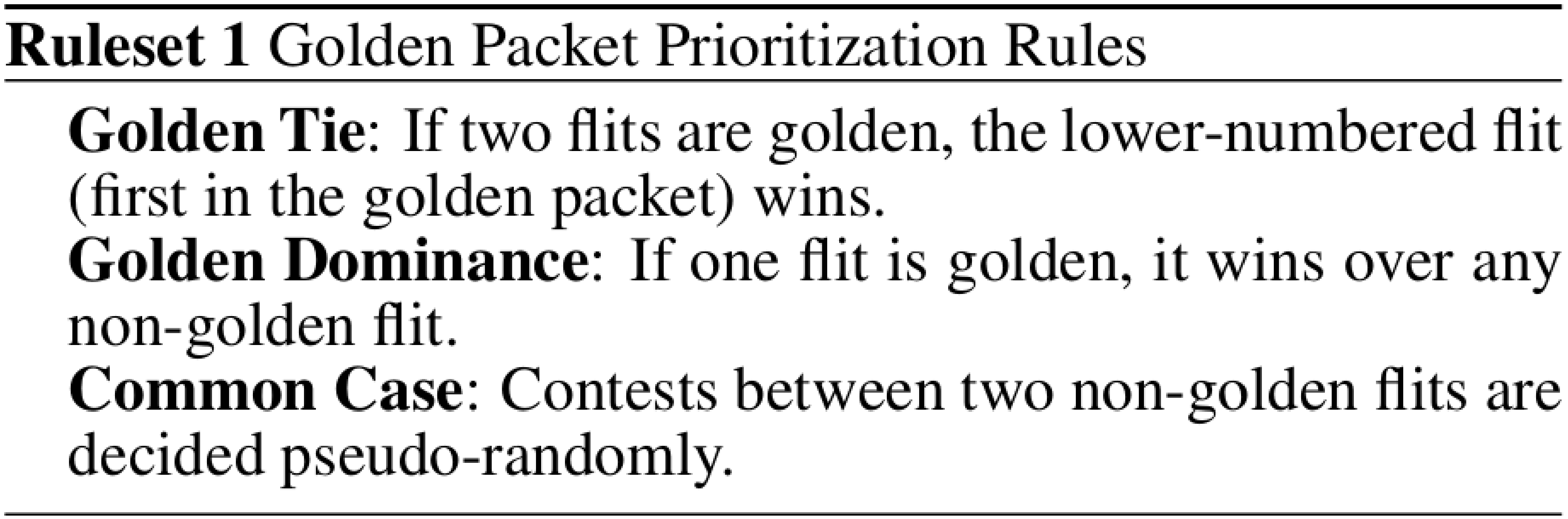


Figure 3: CHIPPER architecture: a permutation network replaces the traditional arbitration logic and crossbar.



1. Pick winner by priority (GP)
2. Send winner to desired port
3. Send loser to remaining port

But what about the livelocks?



Ruleset 1 Golden Packet Prioritization Rules

Golden Tie: If two flits are golden, the lower-numbered flit (first in the golden packet) wins.
Golden Dominance: If one flit is golden, it wins over any non-golden flit.
Common Case: Contests between two non-golden flits are decided pseudo-randomly.

Couple more words about the reassembly buffers

Solutions for deadlocks

Retransmit once

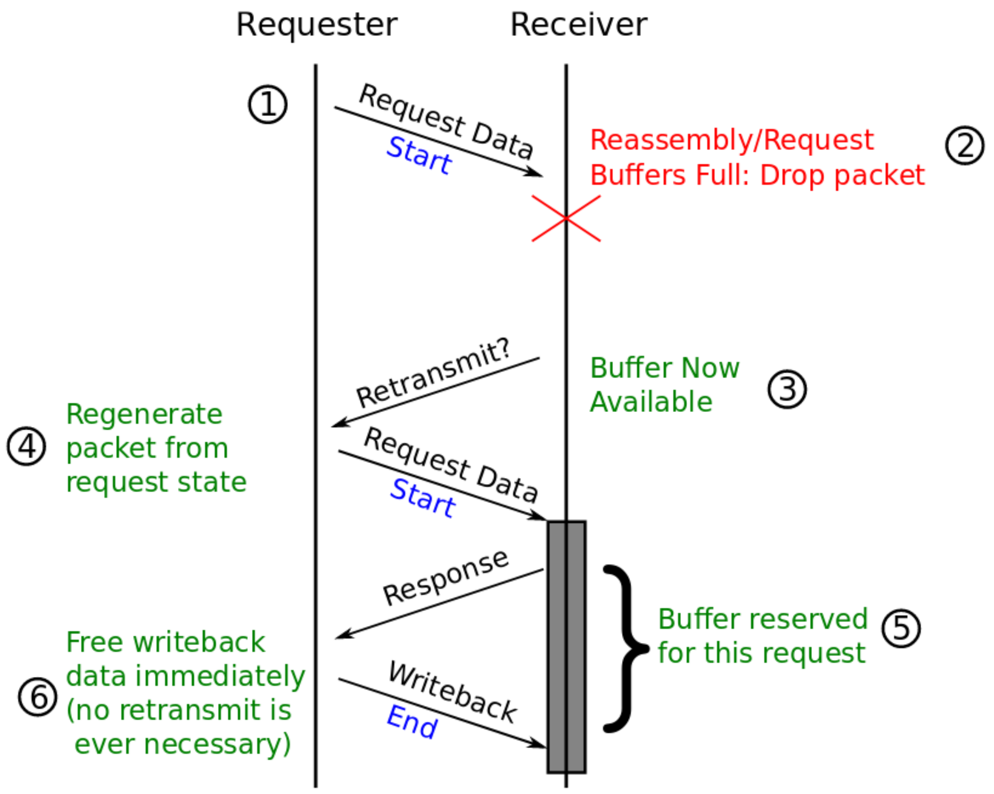
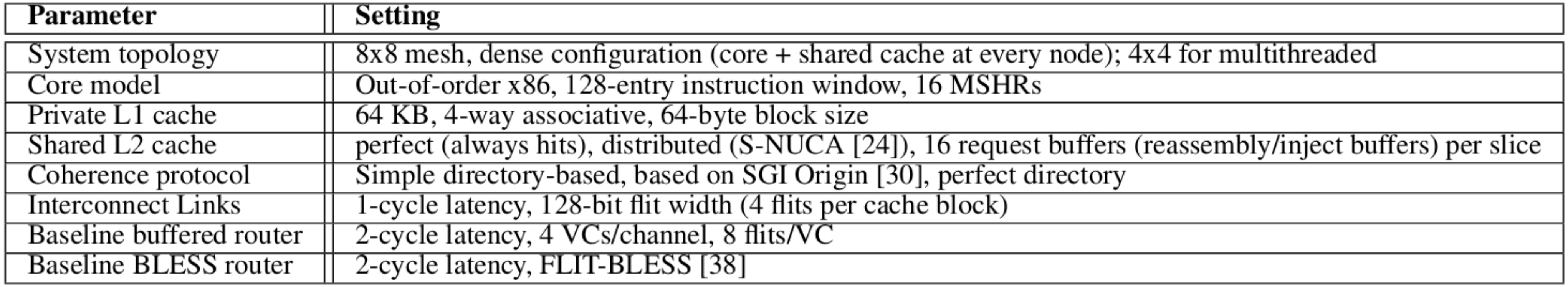


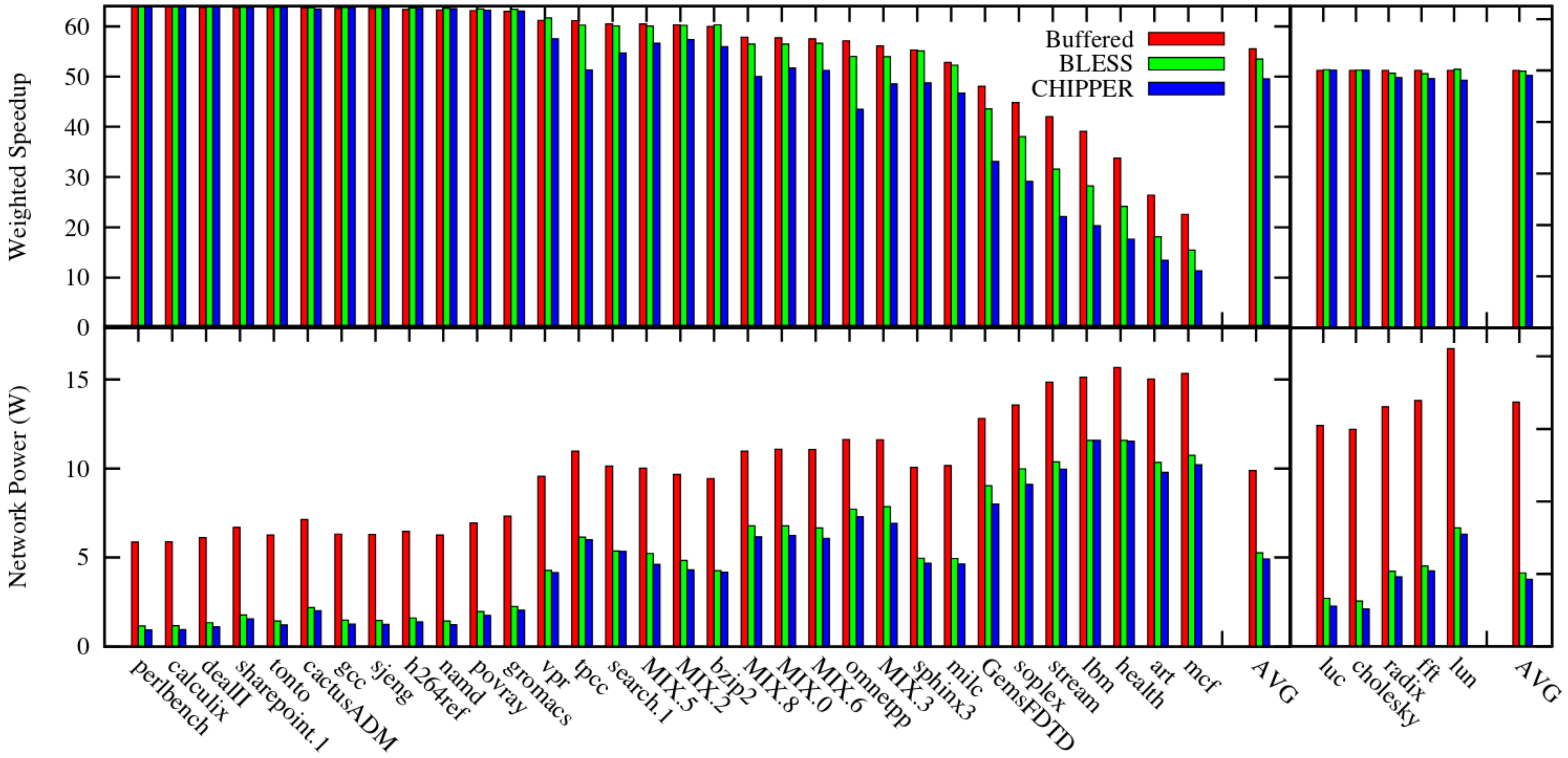
Figure 4: Retransmit-Once flow control scheme.

Evaluation

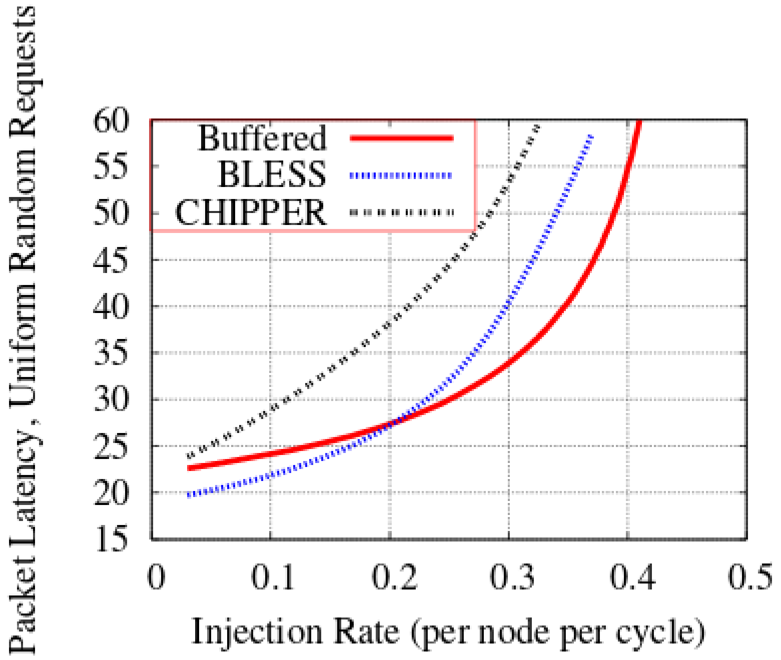


Parameter	Setting
System topology	8x8 mesh, dense configuration (core + shared cache at every node); 4x4 for multithreaded
Core model	Out-of-order x86, 128-entry instruction window, 16 MSHRs
Private L1 cache	64 KB, 4-way associative, 64-byte block size
Shared L2 cache	perfect (always hits), distributed (S-NUCA [24]), 16 request buffers (reassembly/inject buffers) per slice
Coherence protocol	Simple directory-based, based on SGI Origin [30], perfect directory
Interconnect Links	1-cycle latency, 128-bit flit width (4 flits per cache block)
Baseline buffered router	2-cycle latency, 4 VCs/channel, 8 flits/VC
Baseline BLESS router	2-cycle latency, FLIT-BLESS [38]

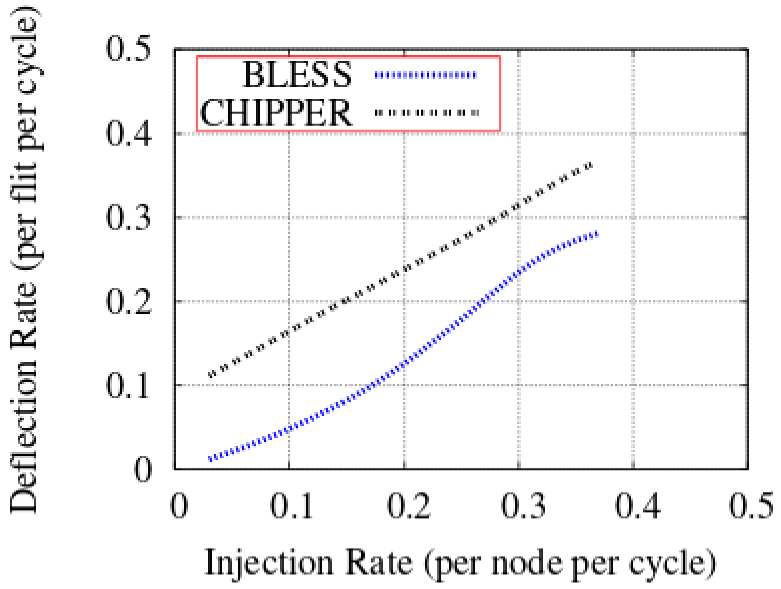
Table 1: System parameters used in our evaluation.


Buffered
BLESS
CHIPPER
Weighted Speedup
Speedup (normalized)
Network Power (W)
Network Power (W)
perlbench
calculix
dealII
sharepoint.1
tonto
cactusADM
gcc
sjeng
h264ref
namd
povray
gromacs
vpr
tpcc
search.1
MIX.5
MIX.2
bzip2
MIX.8
MIX.0
MIX.6
omnetpp
MIX.3
sphinx3
milc
GemsFDTD
soplex
stream
lbm
health
art
mcf
AVG
luc
cholesky
radix
fft
lun
AVG

Locality aware data mapping



(a) Total latency



(b) Deflection rate

	Buffered	BLESS	CHIPPER	% Δ Buffered $\rightarrow$ CHIPPER	% Δ BLESS $\rightarrow$ CHIPPER
Area	480174 μm^2	311059 μm^2	306165 μm^2	36.2% reduction	1.6% reduction
Timing (crit path)	1.88ns	2.68 ns	1.90 ns	1.1% increase	29.1% reduction

Table 2: Hardware cost comparisons for a single router in a 65nm process.

Further reading

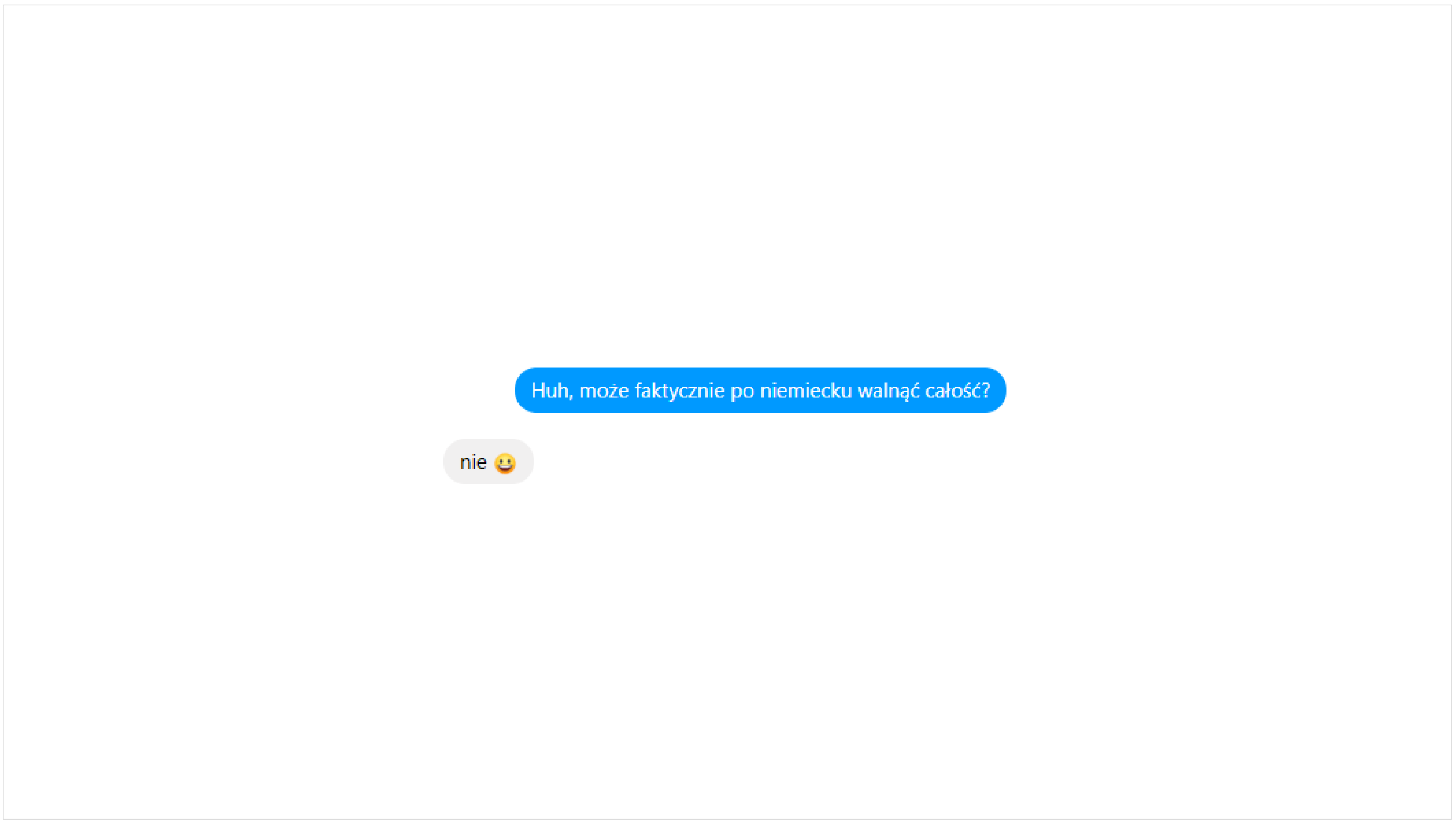
en.wikichip.org ← lots of informations on processors, architectures and beautyfull pictures of dies

CHIPPER

Ein kurzer Vortrag über pufferlose Ablenkrouter

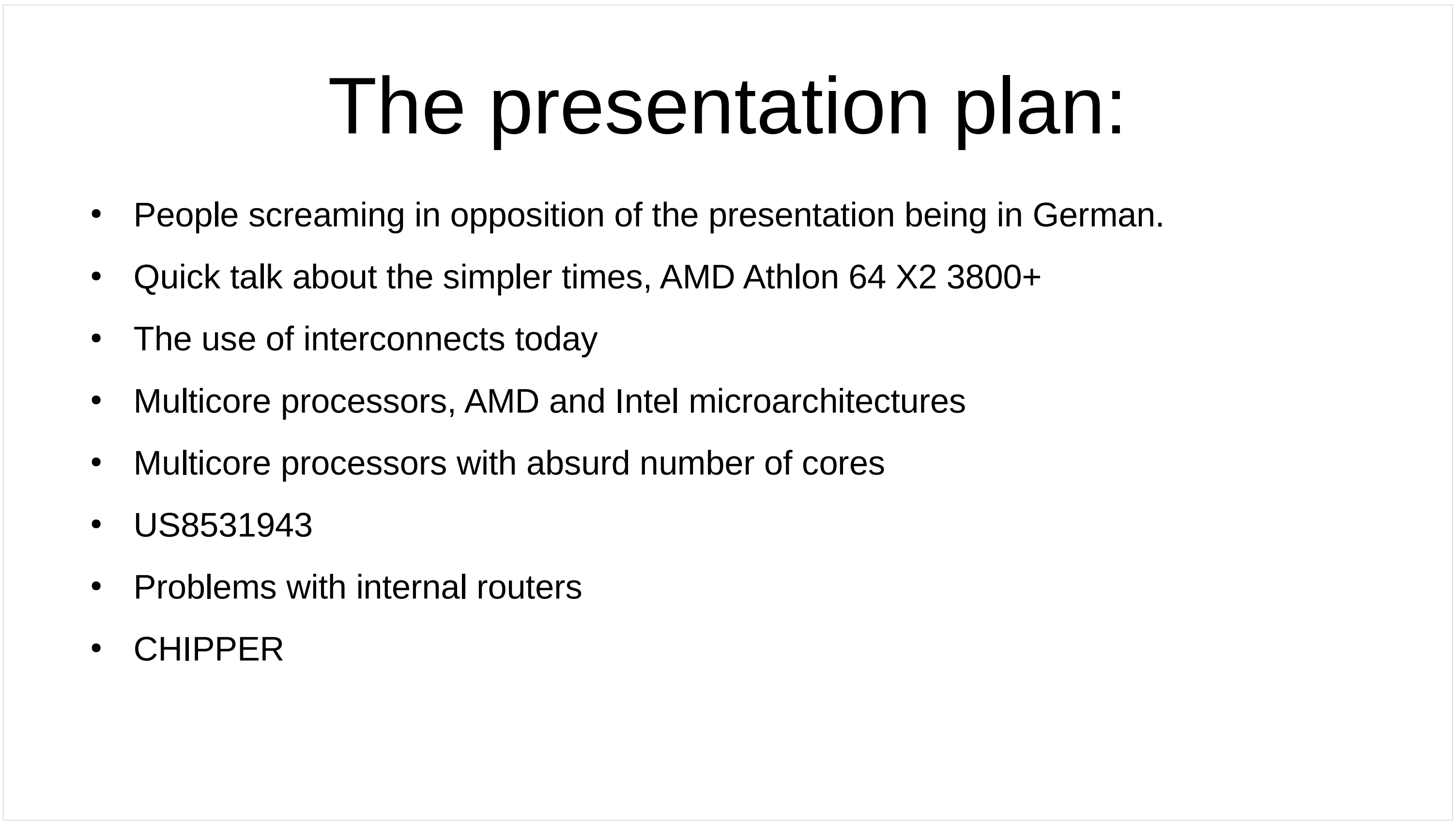
Zbigniew Drozd

Hallo, Ich heisse Zbyszek, und heute sprechen wir uber die pufferlose ablenkrouteren in eingebettete prozessoren.



Huh, może faktycznie po niemiecku walnąć całość?

nie 😀


The presentation plan:

- People screaming in opposition of the presentation being in German.
- Quick talk about the simpler times, AMD Athlon 64 X2 3800+
- The use of interconnects today
- Multicore processors, AMD and Intel microarchitectures
- Multicore processors with absurd number of cores
- US8531943
- Problems with internal routers
- CHIPPER

Alright, let’s get to the main presentation. Today we will try to cover the immense topic of interconnects and implementation of bufferless routers.

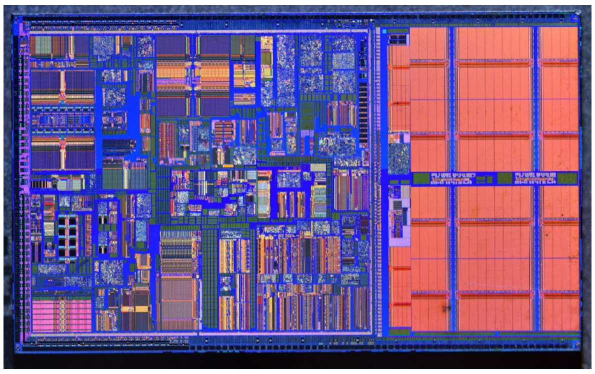
First, we will start with how people decided to ruin their lives by implementing multicore processors, in a bit of a history lesson.

Then, we will talk about how the interconnects are used today, to show that this is not some expensive server technology accesible only by chosen people with lots of money.

Next we will delve into microarchitectures of intel and amd, and see how exactly are the interconnects implemented. Now, disclaimer, this seminar focuses on low level implementations. So we will not care about what exactly is being sent, or why. We will have a sender, a receaver, and a packet containing data, that will travel from A to B.

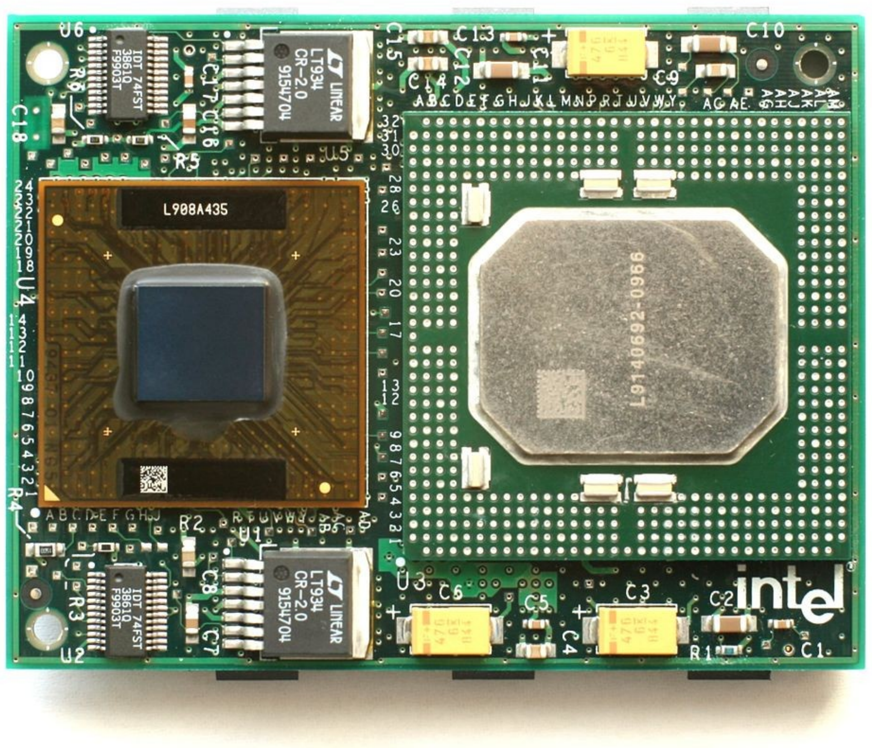
Then, we will talk about how people deal with processors with stupid number of cores. A case study of epiphany V, a 1024 core risc processor.

Quick (pre)history of processors



<quick rant about how earlier days were simpler, as there were only single core processors>

But then, in 2005 AMD has introduced the first dual core CPU the Athlon 64 X2 3800+. This is where our story begins, as at this point we started to have problems with connecting multiple chips together.



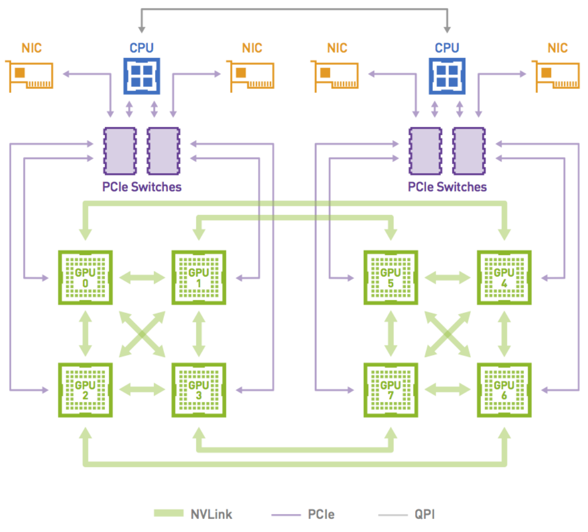
Pentium II Overdrive.
Deschutes core on the left,
512 KB of L2 cache on the right.

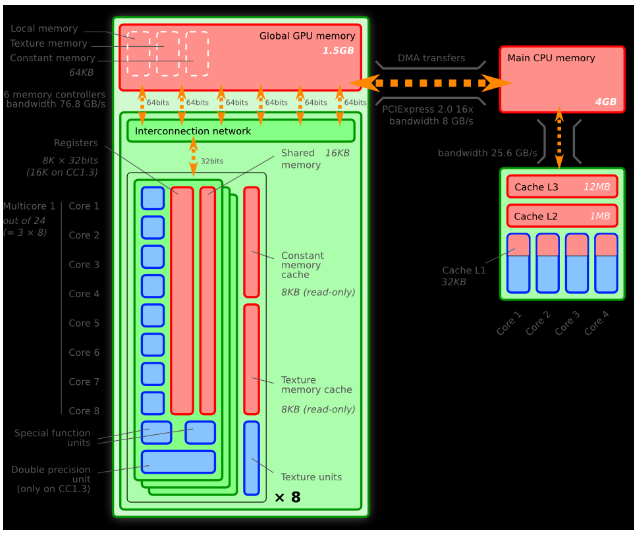
Please take a moment to appreaciate the fact that we live in a time when L3 is integrated into the die.

- As a fun fact, here is the bleeding edge of the 90’s. Notice how the L2 is outside the processor.
- Also, notice how the horrible nameing schemes vere present even 20 years ago.

Interconnects today

Nvidia

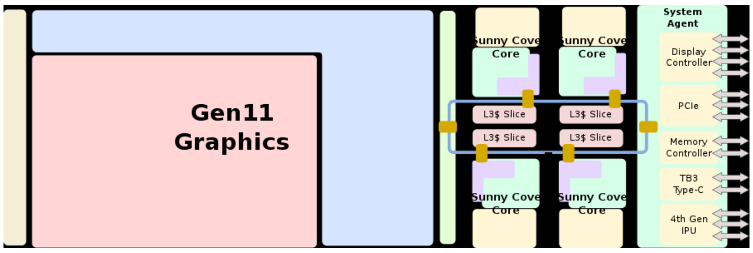


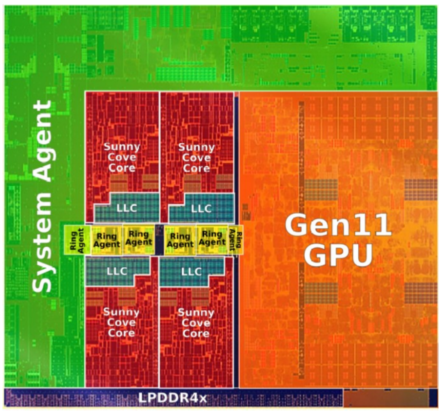


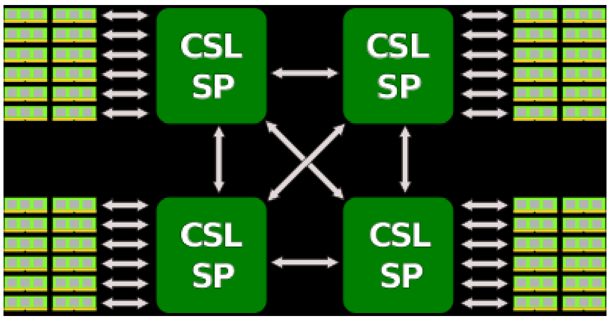
In this case study we will go really quickly over some HPC companies to show that interconnects are present in every single area of computeing.
Here we have nvidia, with their interconnects.

First example we have NVLink. This technology is not for the feint of heart, as prices of servers (whoose diagram is on the left are as high as a new car). But on the right, we can see a simplified graph of a gpu, with the interconnect marked as a green rectangle. Every GPU must have such an interconnect to shuffle data between processing cores.

Intel

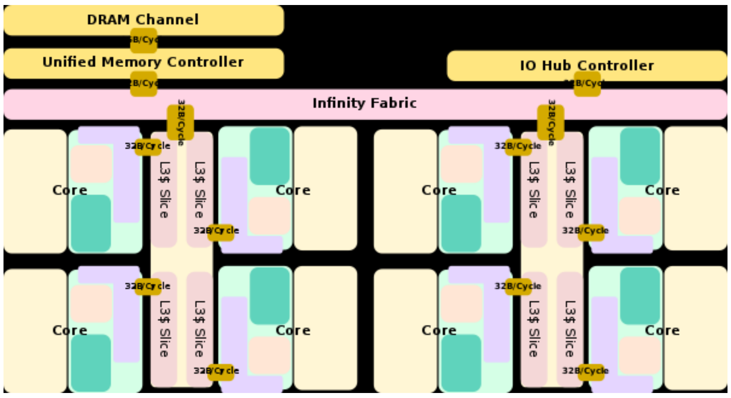


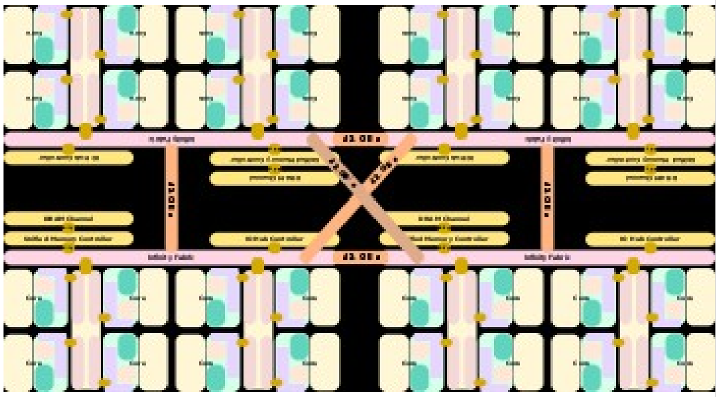




Next, intel. And similarly, we have two examples. On the right, is a four cpu motherboard, with intel’s way of connecting them. Same story, equivalent to a price of a car. But on the left side, we can see insides and a diagram of an ice lake processor, that is accesable to a mere mortal. This suprised me, as the processor uses a ring interconnect (so not „all to all” as I have thought). This is a brief slide about intel, we will talk later about the specific implementations of interconnects that they have conceaved during the years.

AMD





Now, AMD with their Zen architecture, and Infinity fabric interconnect. This one is actually quite interesting, as it leverages chiplets (so processing cores on separate dies)

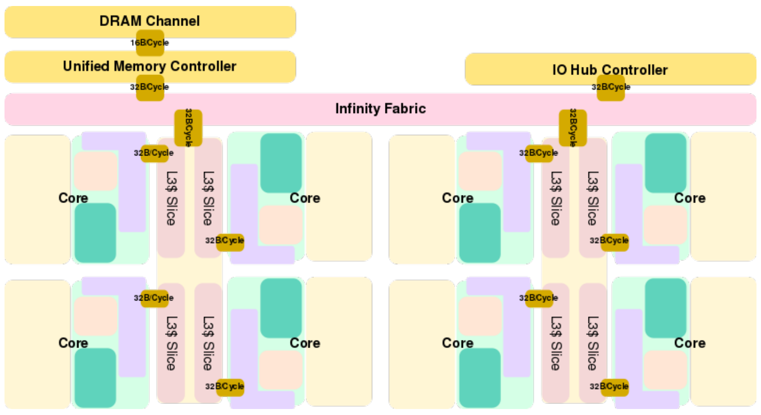
What the heck is SMBus?

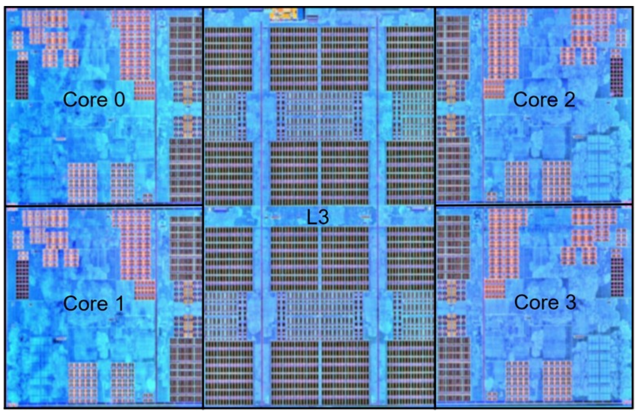
Not every interconnect is as fast as we have covered previously. There are actually interconnects in embedded systems, than enable master to slave communication. There is even one on the motherboard of your computer. It’s called SMBus, and it derives from I2C, a two wire protocol that is known to everyone who had done anything with electronics.

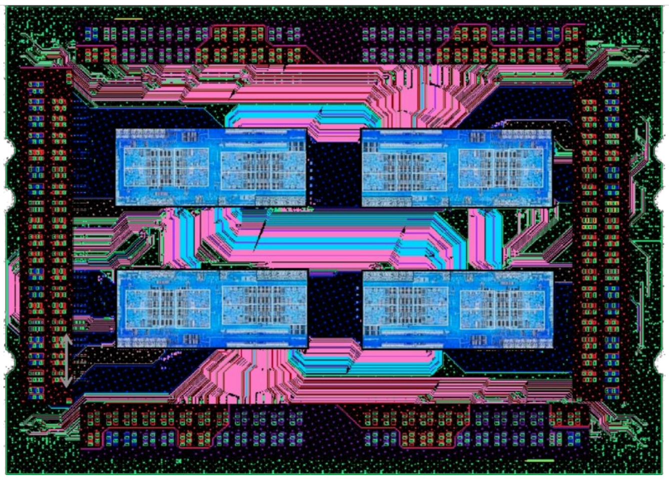
Multicore processors

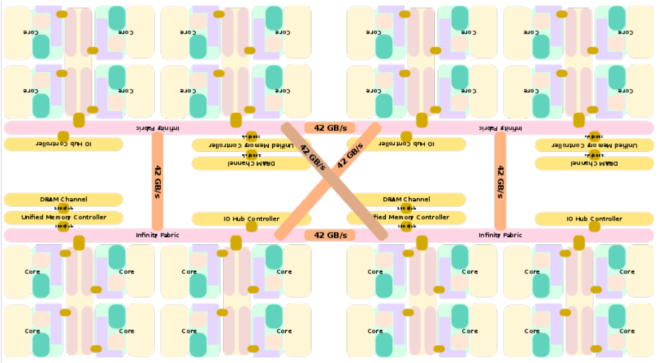
Ok, now that we have skimmed over the different kinds of applications of interconnects, we will focus on CPU’s (sorry nvidia) and solutions that are kept inside a single chip (we are giving AMD a pass, and stick with their chiplets for the next hour)

AMD Infinity Fabric

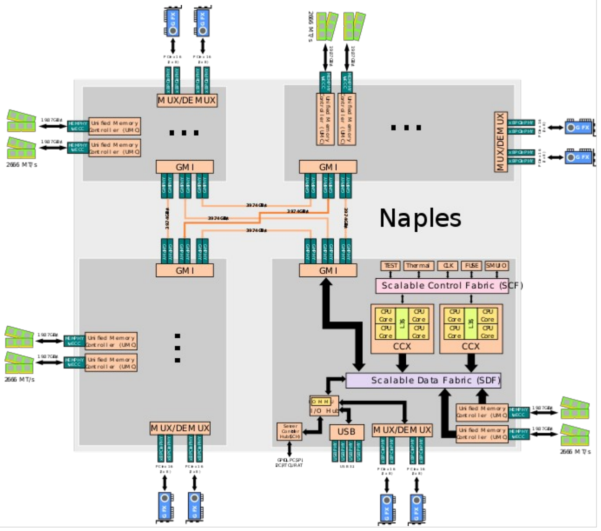








Core
Infinity Fabric
42 GB/s
IO Hub Controller
Unified Memory Controller
DRAM Channel



Naples
MUX/DEMUX
GMI
Unified Memory Controller (UMC)
2666 MT/s
19.87 GB/s
39.74 GB/s
GFX
TEST
Thermal
CLK
FUSE
SMUIO
Scalable Control Fabric (SCF)
CPU Core
L3$
CCX
Scalable Data Fabric (SDF)
IOMMU
I/O Hub
Server Controller Hub (SCH)
USB

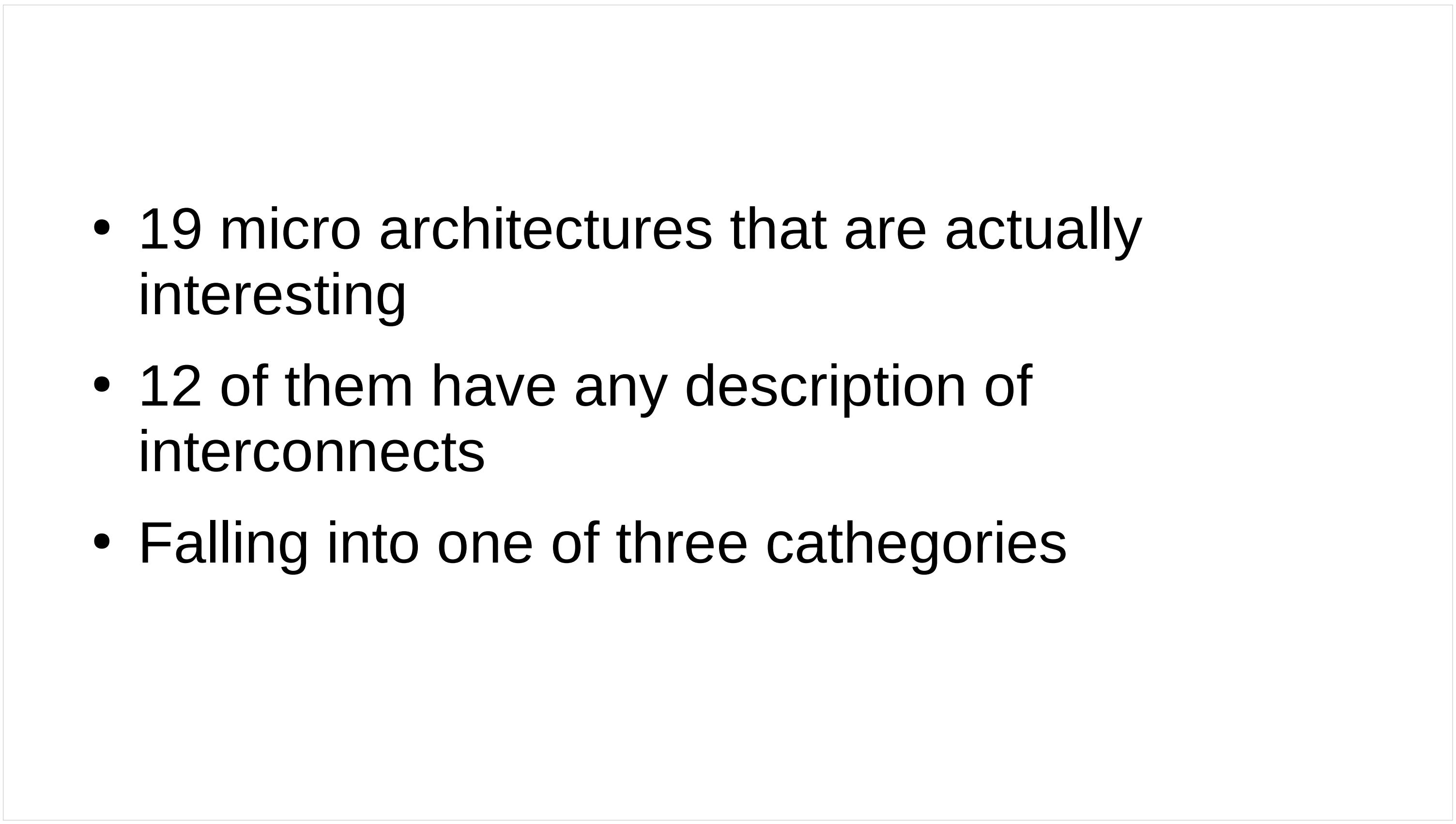
Intel uuuh

- Petition for Intel to chill the fuck out and stop conceaving a microarchitecture every 2.3 nS
- On wikichip 69 pages of intel microarchitectures exist, with additional 19 for GPU’s

So, intel processors.
I have a petition for Intel, to chill out and stop creating a new microarchitecture every 2.3 nS. On wikichip (that will be the source for the following slides, there are 69 pages talking about intel cpu microarchitectures)

Intel uuuh

- Petition for Intel to chill the fuck out and stop conceaving a microarchitecture every 2.3 nS
- On wikichip 69 pages of intel microarchitectures exist, with additional 19 for GPU’s
- Yet no Intel GPU’s exist (at least for commercial sale)



- 19 micro architectures that are actually interesting
- 12 of them have any description of interconnects
- Falling into one of three cathegories

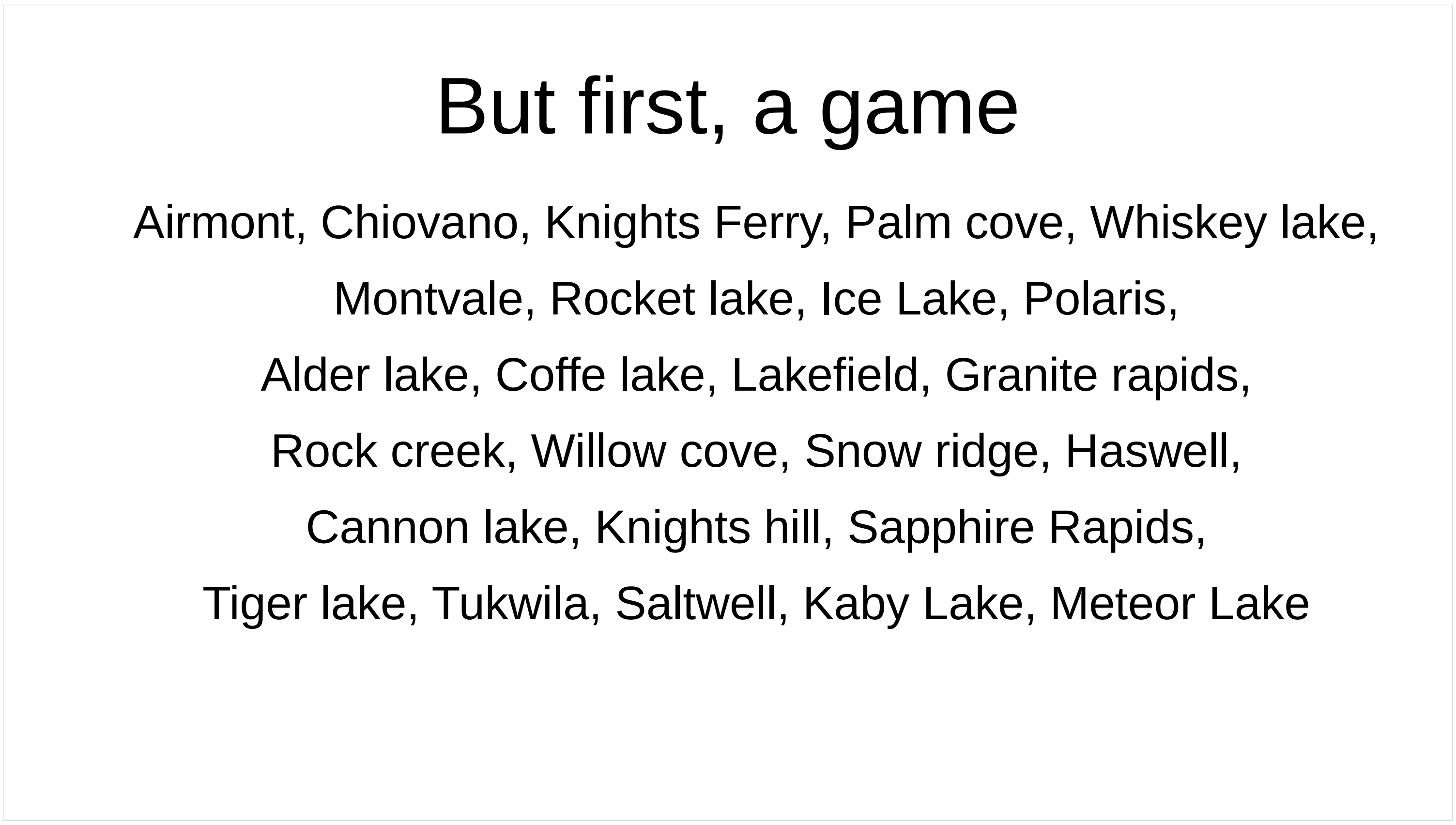
So, we have 19 mircro architectures that are well documented (either because closed source, or people not giving a shit, as those microarchitectures are similar to each other)

12 of them have any description of the interconnects, that fall into one of three cathegories:

> ring
> mesh
> slapping cores directly to the system agent.

But first, a game

But first, a game. I will now display some names. Part of them are intel architectures, some of them are made up, and others are geological places from nature preserves in america. Try to guess what are they)



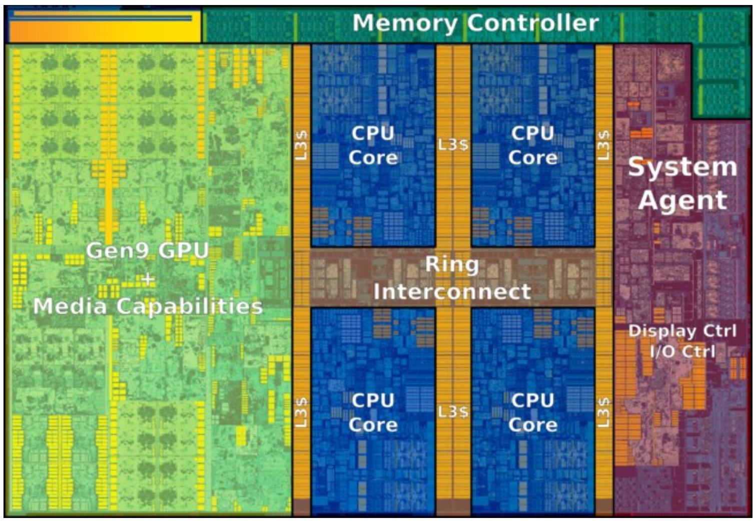
But first, a game

Airmont, Chiovano, Knights Ferry, Palm cove, Whiskey lake,
Montvale, Rocket lake, Ice Lake, Polaris,
Alder lake, Coffe lake, Lakefield, Granite rapids,
Rock creek, Willow cove, Snow ridge, Haswell,
Cannon lake, Knights hill, Sapphire Rapids,
Tiger lake, Tukwila, Saltwell, Kaby Lake, Meteor Lake

Yeah, they are all intel microarchitectures.

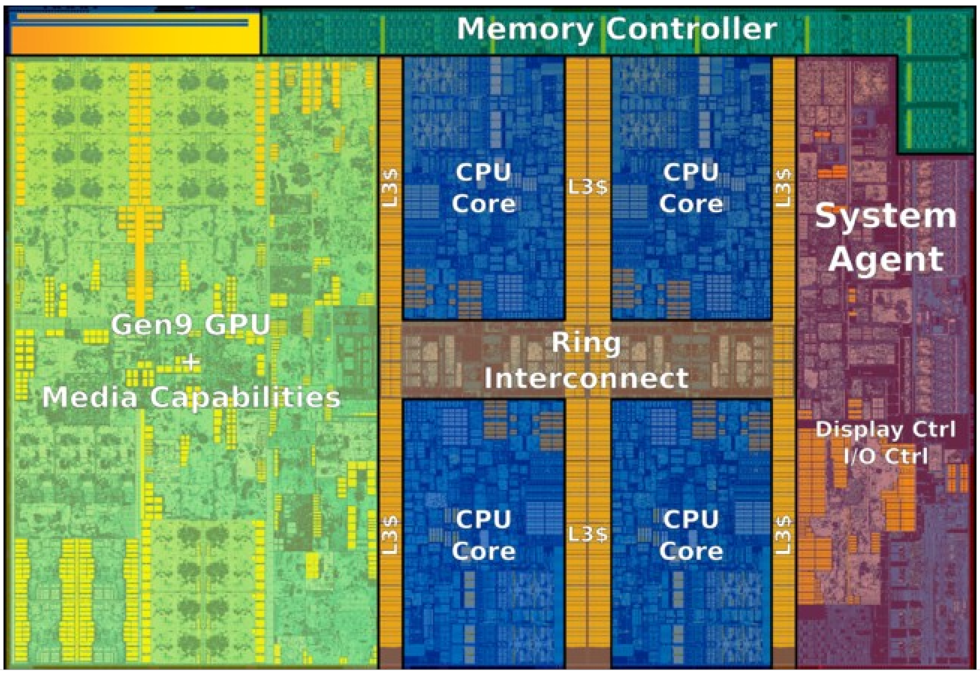
Intel – Agents

- System agent
- Ring agent



So, before we go any further, we should explain what system agent and ring agent is. To put it simply System agent contains Image Processing Unit (somehow different from the GPU), Display Engine (no clue what this is), I/O bus (pcie, memory controller). This is included in the seminar, as not many people probably know about it’s existence. Ring agent – possibly intel’s name for the networking part of the core. Not sure on this one tho.

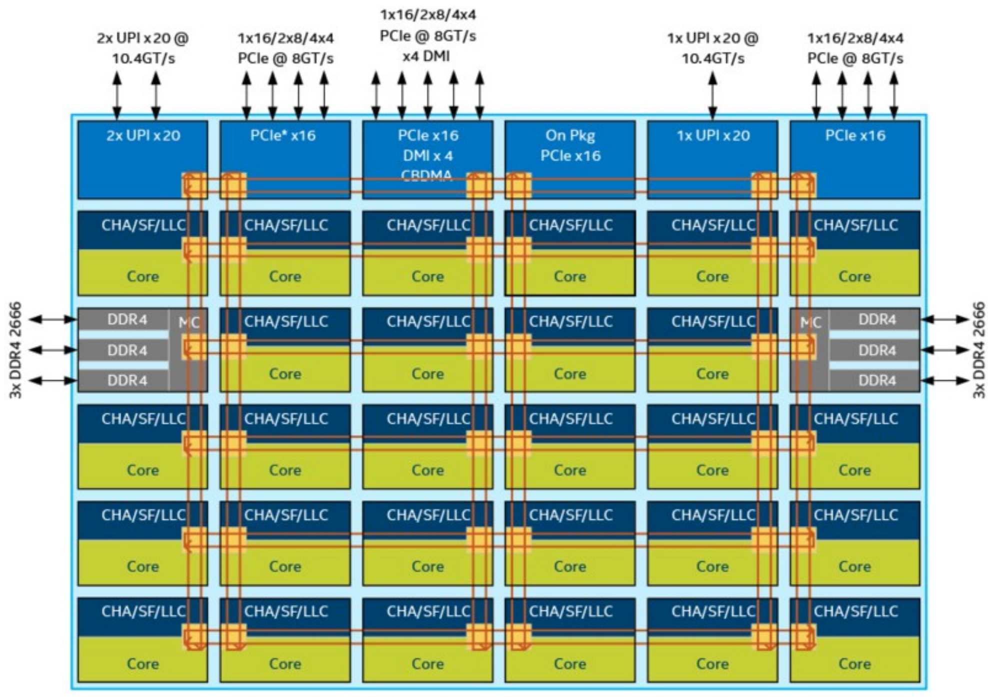
Intel – Ring interconnect



The ring interconnect is the most popular interconnect in mid to high range intel CPU's these days. It features a ring built up of ring agents (routers) that handle transmitting data around the ring. The interconnect also joins the GPU and system agent.

The interconnect is actually composed of four rings
> Data
> Request
> Acknowlegde
> Snoop

Intel – mesh interconnect

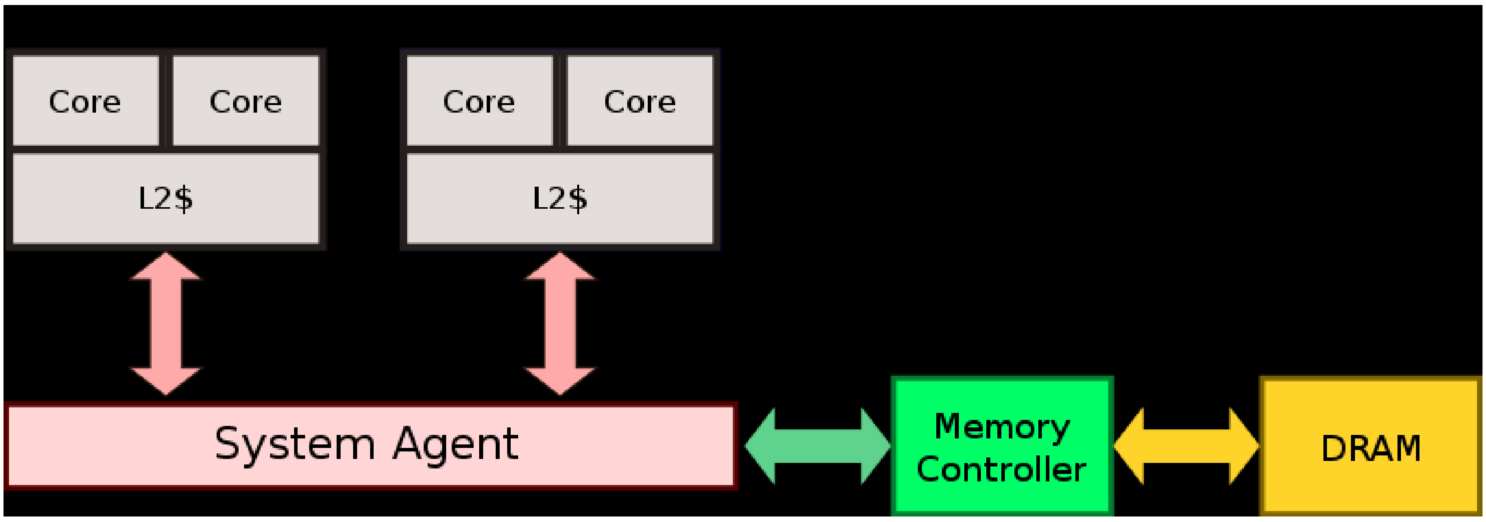


This interconnect is used in Cascade lake, Skylake and Polaris. First two are real products, the last one is a showcase for multicore processing.

What is mesh architecture you might ask? Mesh is acheaved by placing vertical and horisontal bidirectional halfrings.

Here, the architecture of a skylake processor.

Intel – slapping cores directly to system agent



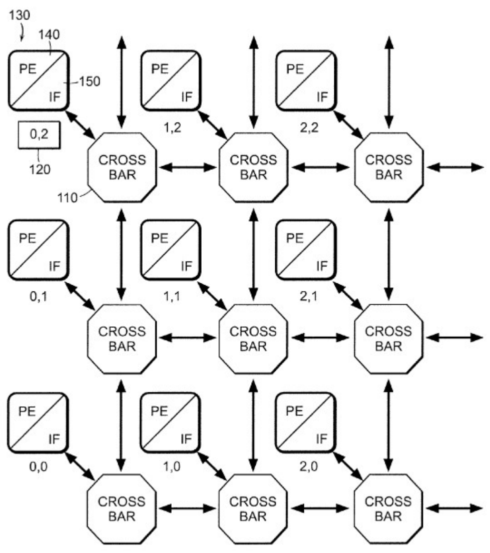
This is in Silvermont (and probably other architectures) processors. These are mobile chips, so such a solution makes sense, we are not doing heavy computing tasks on a smartphone.

These were in 1,2,4,8 core configurations

High count core processors

Ok. So, we have talked about commercial CPU’s with up to 72 cores. But what if we wanted to go bigger? Surely, those solutions are scalable, right? They will work up to, say, 1024 cores, right?

US8531943



Epiphany-V: A 1024 processor 64-bit RISC System-On-Chip

Andreas Olofsson

(give it a read, it’s an awesome paper about chip design)

Now, we will be doing a case study of Epiphany-V. This is a processor designed by a single person, funded by DARPA, that you sadly can’t buy :/

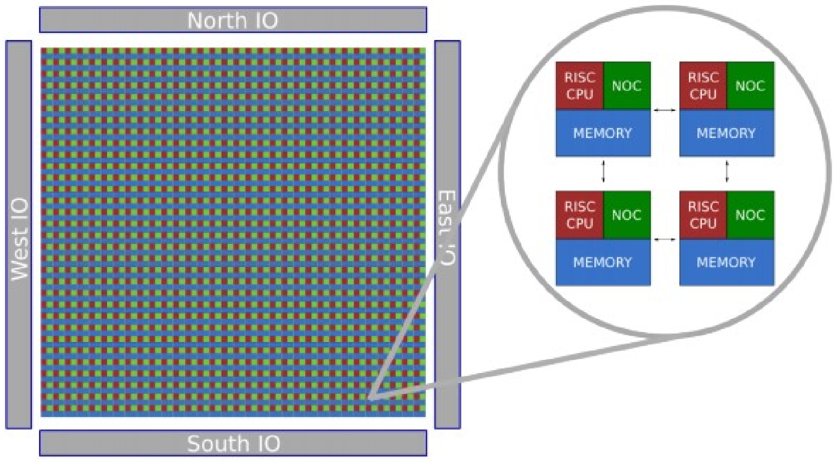


Figure 1: Epiphany-V Overview

This is the architecture of a single chip. We can see the cores placed in a grid, with IO for connecting multiple chips together. Note the NOC (Network on chip) that takes around ¼ of the space

III.B Memory Architecture

The Epiphany 64-bit memory map is split into 1 Billion 1MB memory regions, with 30 bits dedicated to x,y,z addressing. The complete Epiphany memory map is flat, distributed, and shared by all processors in the system. Each individual memory region can be used by a single processor or aggregated as part of a shared memory pool. The Epiphany architecture uses multi-banked software-managed scratch-pad memory at each processor node. On every clock cycle, a processor node can:

- Fetch 8 bytes of instructions
- Load/store 8 bytes of data
- Receive 8 bytes from another processor in the system
- Send 8 bytes to another processor in the system

III.C Network-On-Chip

The Epiphany-V mesh Network-on-Chip ("emesh") consists of three independent 136-bit wide mesh networks. Each one of the three NOCs serve different purposes:

- rmesh: Read request packets
- cmesh: On-chip write packets
- xmesh: Off-chip write packets

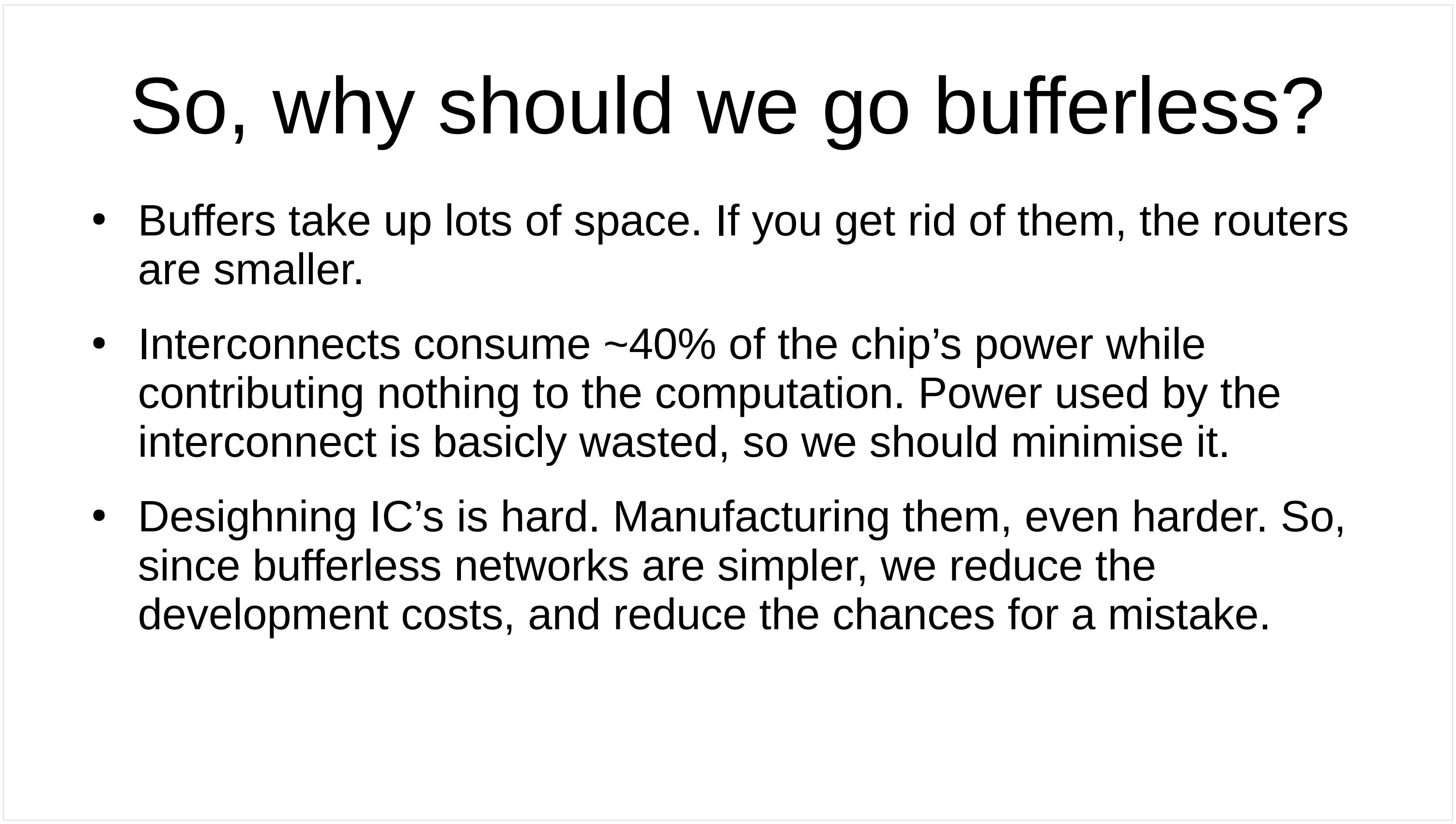
Epiphany NOC packets are 136 bits wide and transferred between neighboring nodes in one and a half clock cycles. Packets consist of 64 bits of data, 64 bits of address, and 8 bits of control. Read requests puts a second 64-bit address in place of the data to indicate destination address for the returned read data.

Network-On-Chip routing follows a few simple, static rules. At every hop, the router compares its own coordinate address with the packet's destination address. If the column addresses are not equal, the packet gets immediately routed to the south or north; otherwise, if the row addresses are not equal, the packet gets routed to the east or west; otherwise the packet gets routed into the hub node.

Each routing node consists of a round robin five direction arbiter and a single stage FIFO. Single cycle transaction push-back enables network stalling without packet loss.

How much do interconnects cost?

Actually, quite a lot. If we go with the epiphany V figures, interconnects eat up 10% of the chip. That is quite a lot. That’s why we will delve into the topic of bufferless routers and see how they work, and how they can improve the design of a chip.



So, why should we go bufferless?

- Buffers take up lots of space. If you get rid of them, the routers are smaller.
- Interconnects consume ~40% of the chip’s power while contributing nothing to the computation. Power used by the interconnect is basicly wasted, so we should minimise it.
- Desighning IC’s is hard. Manufacturing them, even harder. So, since bufferless networks are simpler, we reduce the development costs, and reduce the chances for a mistake.

So, what are the arguments for swithcing to bufferless designs?

Since IC design and fabrication is pretty much the most expensive thing you can realistically do in electronics, the design should be compact (to minimise the footprint of the chip) and simple (to minimise chances for faliures in design, that render a batch of processors unusable).

Power dissapation is also a big factor, as powering buffers (so basically, big chunks of memory) takes lot’s of power, hindering the thermal performance and efficiency of the chip

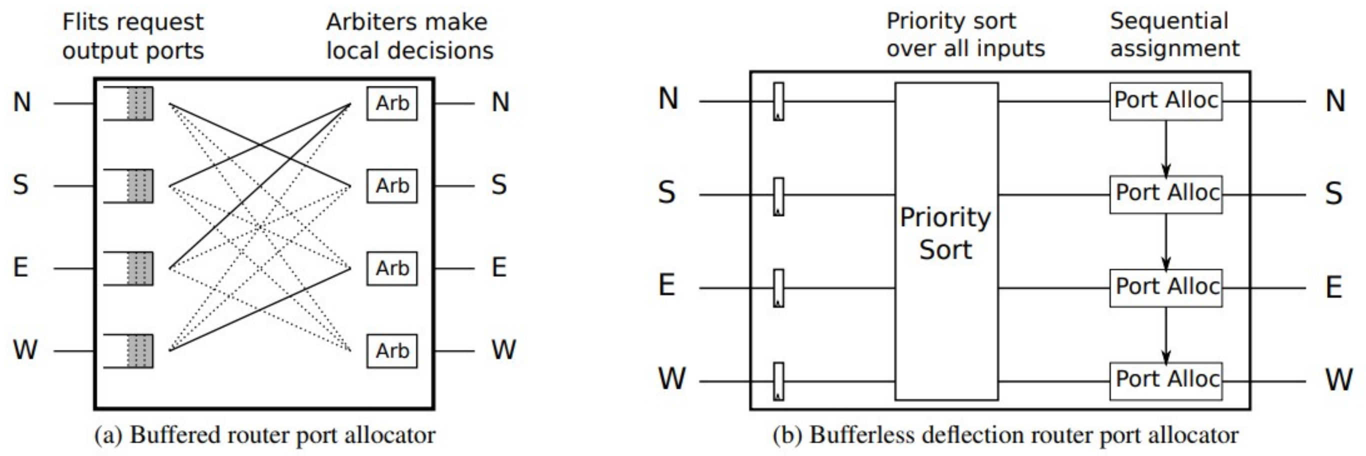

Flits request output ports
Arbiters make local decisions
N
S
E
W
Arb
Arb
Arb
Arb
N
S
E
W
(a) Buffered router port allocator
Priority sort over all inputs
Sequential assignment
N
S
E
W
Priority Sort
Port Alloc
Port Alloc
Port Alloc
Port Alloc
N
S
E
W
(b) Bufferless deflection router port allocator

Figure 1: Port allocator structures: deflection routing requires more complex logic with a longer critical path.

Let's look at the previous attempts of creaing a bufferless network. On the left, we have a typical buffered approach, where incoming data is stored in the memory if it can't exit. On the right, we can see a BLESS router. In bless, we sort the packets by priority (for example, by their age) and choose to route packets with higher priority to places that they want to go. If a packet cannot fit, it will be pushed to a different port.

This assures, that the network won't clog, however, it comes at a cost of slow comparator and sorting circuit, and big overhead for the packet, as it needs to store it's age.

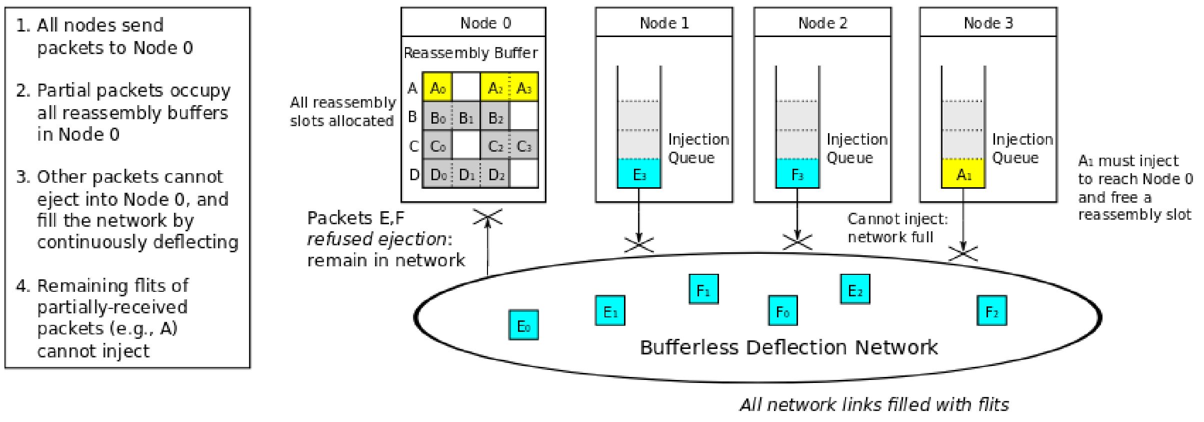
Injection and ejection

Since each router has N inputs and N outputs data injection to the network is non-trivial

So, let’s talk about actually sending some data via the network. Since each router has the same number of inputs as outputs, the node has to wait before injecting the data, to assure that there will be at least one free output for the injected data. Injection buffers are not needed, but should be implemented, as the core would stop working untill there is a free slot in the router.

Ejection is handled by the router itself (so we assume that the node can fit all the data that we want to feed it). This however is a bad assumption, as packets can be receaved out of order, and we need to take care of reordering them.

Deadlock prevention


1. All nodes send packets to Node 0
2. Partial packets occupy all reassembly buffers in Node 0
3. Other packets cannot eject into Node 0, and fill the network by continuously deflecting
4. Remaining flits of partially-received packets (e.g., A) cannot inject

Node 0
Reassembly Buffer
All reassembly slots allocated
A
B
C
D
A_0
A_2
A_3
B_0
B_1
B_2
C_0
C_2
C_3
D_0
D_1
D_2
Node 1
Node 2
Node 3
Injection Queue
E_3
F_3
A_1
A_1 must inject to reach Node 0 and free a reassembly slot
Packets E,F *refused ejection*: remain in network
Cannot inject: network full
F_1
E_2
E_0
E_1
F_0
F_2
Bufferless Deflection Network
All network links filled with flits

Figure 2: Deadlock due to reassembly-buffer overflow.

Since we have no local feedback, deadlocks in the network can occur. Thus, we have to assume pesimistic circumstances. That would mean a need to implement a reassembly buffer capable of fitting data from all the network. And since the whole idea of this paper is to minimise the buffer size and numbers, this is not the way we would like to solve the deadlock problem

CHIPPER router architecture

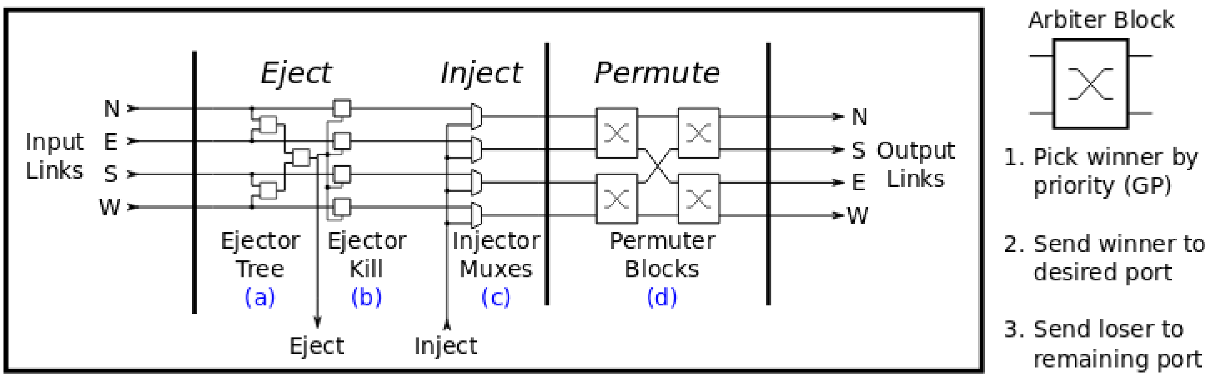
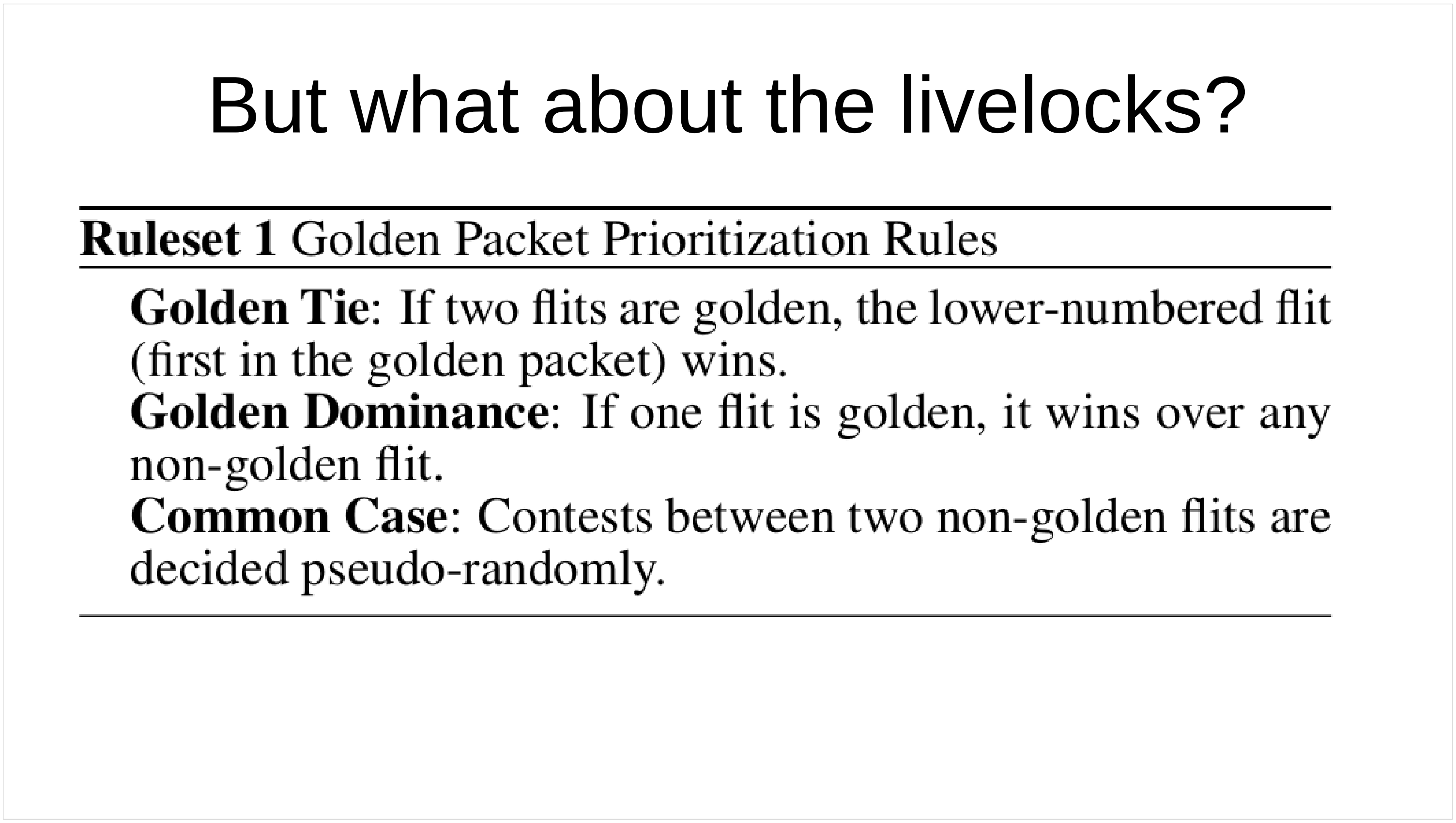


Figure 3: CHIPPER architecture: a permutation network replaces the traditional arbitration logic and crossbar.

This is the router architecture for the CHIPPER. Let's analise it.

We can see, that the proposed architecture is much simpler, and works in a more parallel manner. This is acheaved because we have relaxed the requirements, we will only care about the most important packet, the rest can go anywhere.

We can also see ejectors and injectors for pushing/pulling data from the network. You might wonder why Eject/inject isn't done with another I/O link. That's because we wouldn't have a router with a number of ports being a power of two (design simplicity). Also, you can see that the outputs are premutated. That's because we want to continue going E→E and S→S (those are more popular choices for a packet) when permuter blocks are inactive



But what about the livelocks?

Ruleset 1 Golden Packet Prioritization Rules

Golden Tie: If two flits are golden, the lower-numbered flit (first in the golden packet) wins.
Golden Dominance: If one flit is golden, it wins over any non-golden flit.
Common Case: Contests between two non-golden flits are decided pseudo-randomly.

Livelocks are still possible in this scheme. That is why we will be adding a concept such as golden packet. We will be randomly selecting a single packet, and calling it „golden" giving it priority over any other packet. This soluton ensures that we don't get livelocks (every packet will be golden at some point) and choosing the priviliged packet inside the router is very easy to implement. The status of being a golden packet might be held as a single bit inside the message. We will also need to have comparators for the Golden Tie case, but as those are used the least ofter, we will not care about their power consumption.

Choosing a packet to be golden is a different story. We are sure, that all routers must agree on it, so implementing a global system that tells all routers the golden packet is possible. But since we want to keep the routers as separate as possible, we will implement an algorithm inside all of them, that cycles through all possible packets, and calls them

Couple more words about the reassembly buffers

This one is actually really clever. Since at each node, we have L1 and L2 caches, and if we want to receave data we need to have space alocated for it, we will be using L1 or L2 directly as a reassembly buffer.

We can also use these as injection buffers.

Solutions for deadlocks

So, we have seen that golden packets are solutions for livelocks. But deadlocks can still occur. To mitigate this, we have two obvious solutions.

The first one, is to send a allocation request to the recepient. This is easily implemented, but now, every transfer is proceeded by an allocation request, that slows everything down considerably.

The second one, is to simply drop the incoming packet when the recepient has no space left. This forces us to implement some recovery system, such as the recepient sending a retransmit request. This causes further problems (such as senders keeping the data stored for prolonged amounts of time), so the solution that the autors went with was

Retransmit once

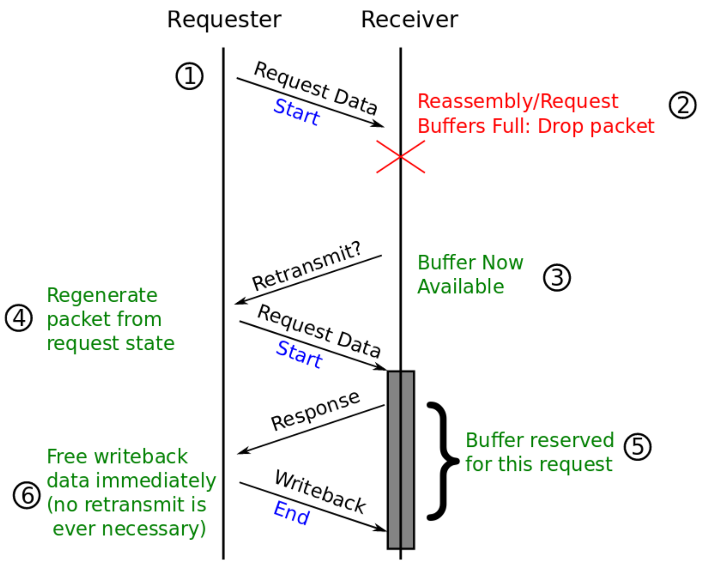


Figure 4: Retransmit-Once flow control scheme.

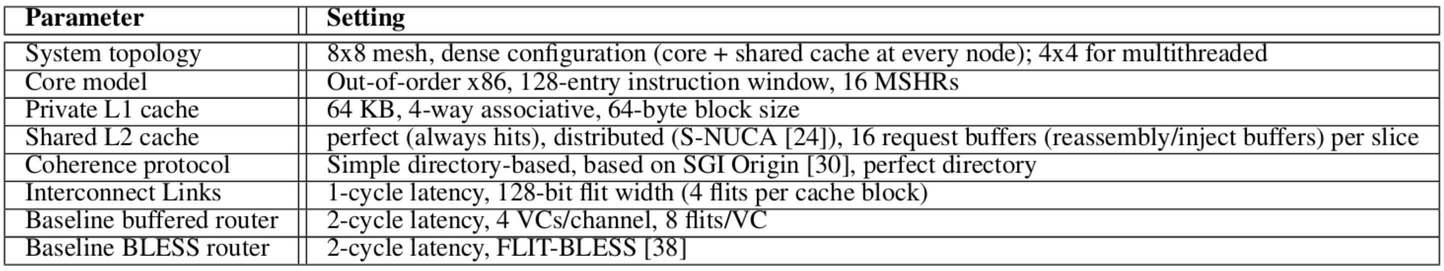
The key idea, is that we can send a allocation request and hope it gets through. We don’t need to store it, as we can generate it easily, since we have the data that we actually want to send.

If the recepient has receaved the allocation request, it will say that the buffer is free, and we can start the data transmission.

If the recepient has dropped the allocation request, it will send a retransmit message to get the allocation request once more. Since we are sending the retransmit message only when we have space, the allocation request will be only sent twice, in the worst case scenario.

This retransmit once protocol works well with the golden packet.

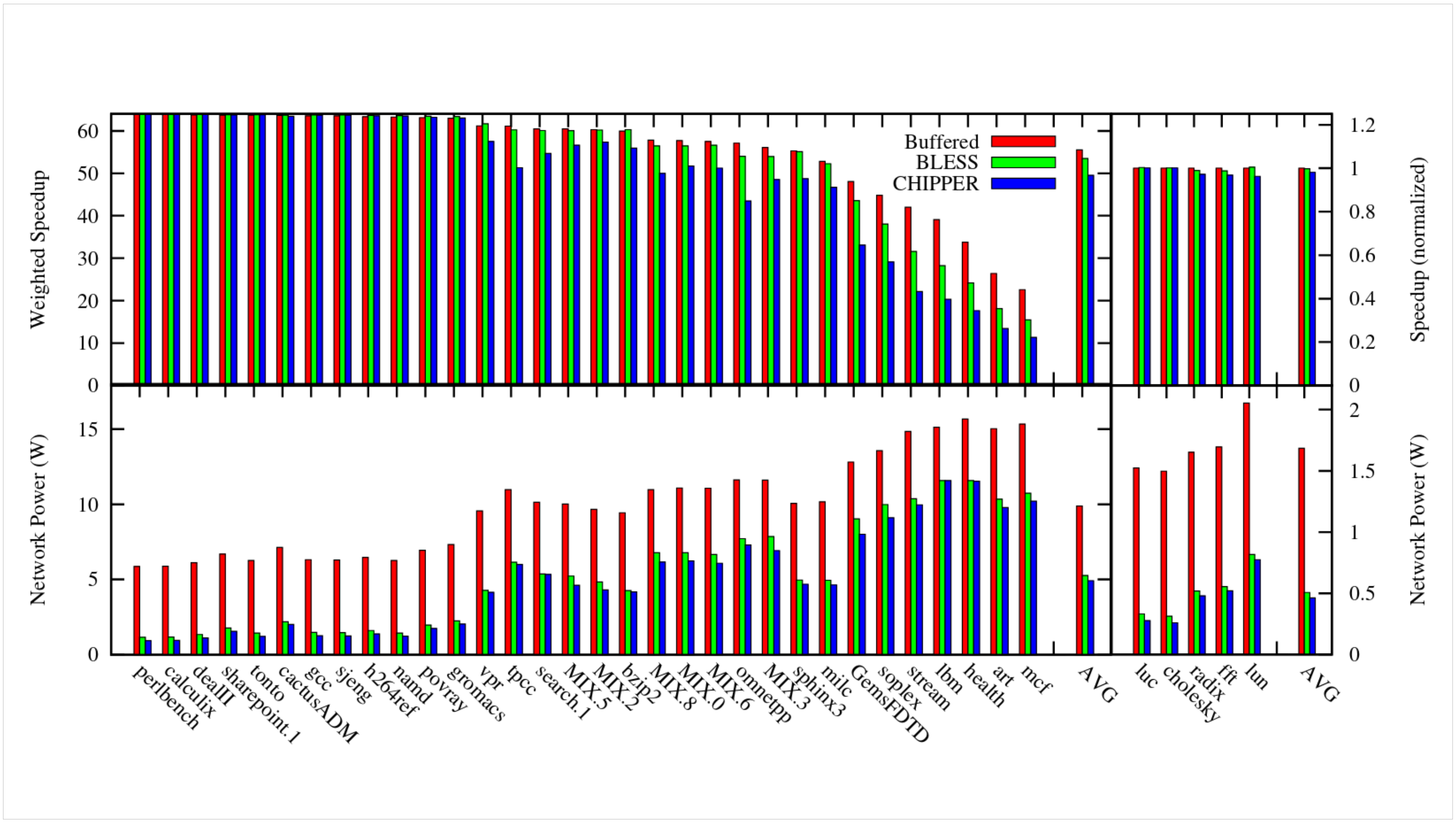
Evaluation



Parameter	Setting
System topology	8x8 mesh, dense configuration (core + shared cache at every node); 4x4 for multithreaded
Core model	Out-of-order x86, 128-entry instruction window, 16 MSHRs
Private L1 cache	64 KB, 4-way associative, 64-byte block size
Shared L2 cache	perfect (always hits), distributed (S-NUCA [24]), 16 request buffers (reassembly/inject buffers) per slice
Coherence protocol	Simple directory-based, based on SGI Origin [30], perfect directory
Interconnect Links	1-cycle latency, 128-bit flit width (4 flits per cache block)
Baseline buffered router	2-cycle latency, 4 VCs/channel, 8 flits/VC
Baseline BLESS router	2-cycle latency, FLIT-BLESS [38]

Table 1: System parameters used in our evaluation.

The proposed system has been simulated on a clock accurate simulator, across different workloads. Above you can see the parameters of the simulated machine

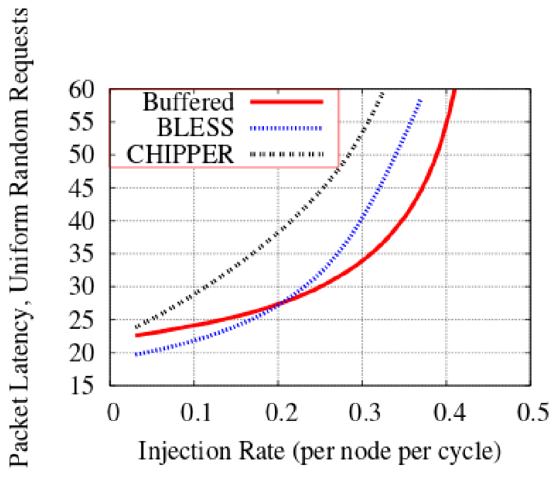


Here we can see relative performance and power usage of a system with different interocnnects. We can see that CHIPPER has the best power efficiency, while acheaving results that are not really worse from BLESS.

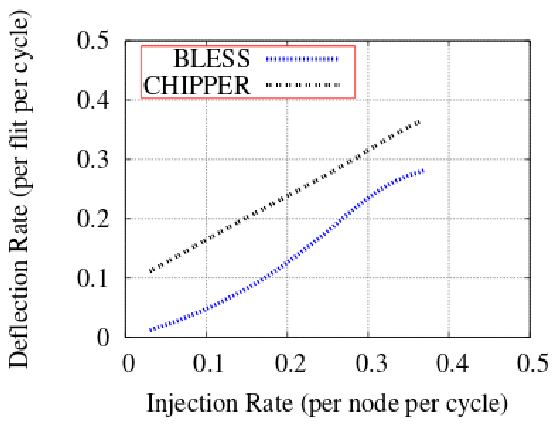
The tests have also shown, that the best size of the buffers is 16 packets big. This results in 0.0016% retransmit rate. Since 16 packet big buffers are a realistic choice, the authors conclude that the retransmit overhead is negligible.

Locality aware data mapping

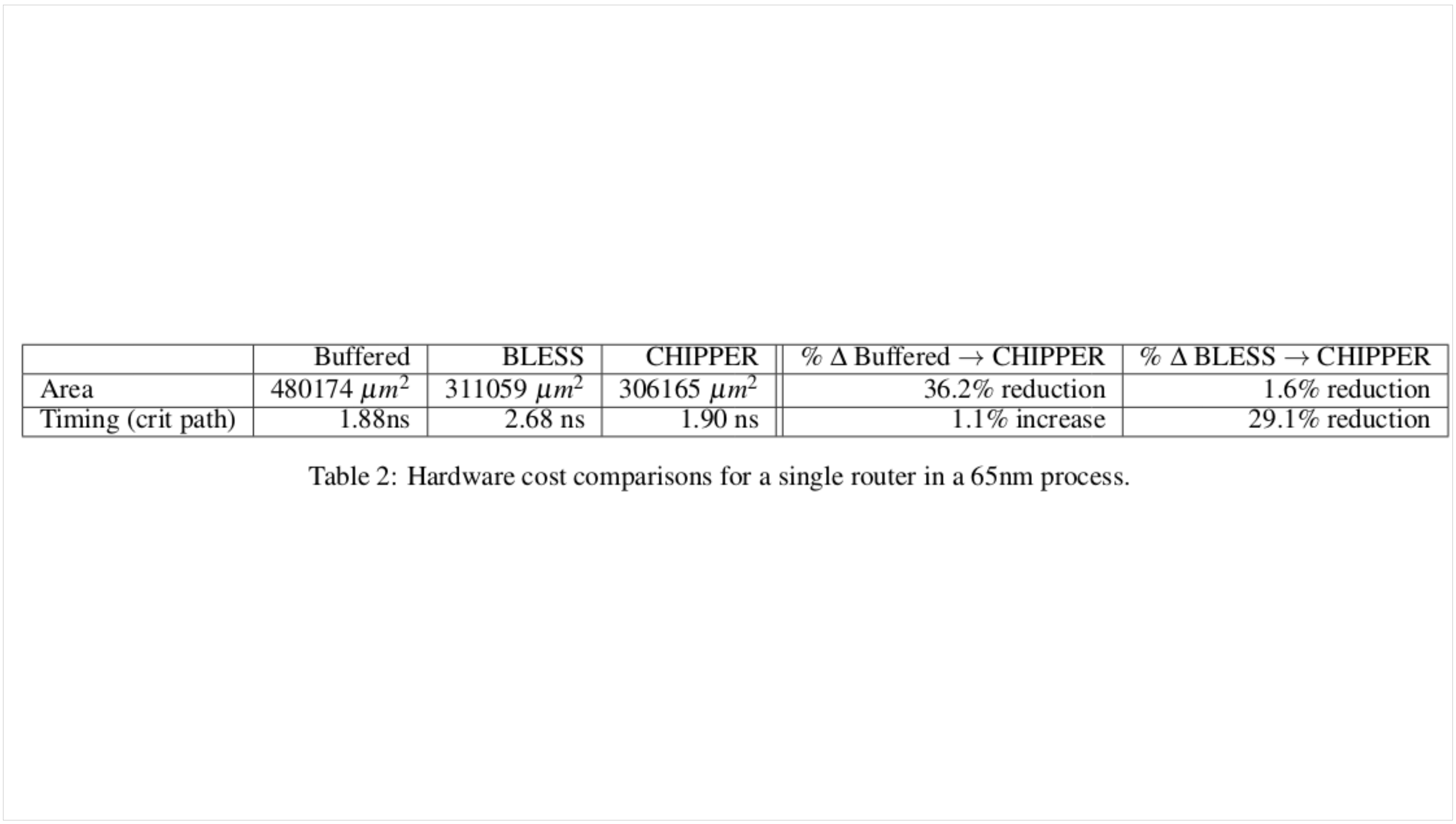
While running CHIPPER on a 8x8 mesh network of processors, we get a 11.7% loss of performance.
If we group processors in 4x4 chunks, the performance degradation drops to 6.8%.
If the chunks are even smaller, the performance drop starts getting neglegible, down to 1.1%.
There results show that when reducing overall network load the performance of the chip increases



(a) Total latency



(b) Deflection rate



	Buffered	BLESS	CHIPPER	% Δ Buffered → CHIPPER	% Δ BLESS → CHIPPER
Area	480174 μm^2	311059 μm^2	306165 μm^2	36.2% reduction	1.6% reduction
Timing (crit path)	1.88ns	2.68 ns	1.90 ns	1.1% increase	29.1% reduction

Table 2: Hardware cost comparisons for a single router in a 65nm process.

Now, CHIPPER might sound like a bad option. It’s slower, saturates way faster than buffered or BLESS networks. So now the question is „why should CHIPPER exist” or even „why whas this paper created”. The main reason is on this slide. 35% reduction in size compared to buffered switches is a huge deal.

Further reading

en.wikichip.org ← lots of informations on processors, architectures and beautyfull pictures of dies