Understanding CPU Microarchitecture

🏎For Maximum Performance

Overview

- What happens inside a CPU?
- Where do CPU intensive programs get delayed?
- What tools are there to help measure performance bottlenecks?
- How can we make programs run faster?

©2020 Alex Blewitt

Performance Pyramid

inst
memory
cpu
hardware
algorithm
system architecture
distributed architecture

This talk

Other QCon talks

https://www.infoq.com/qconlondon2020/

©2020 Alex Blewitt

Platform Topologies

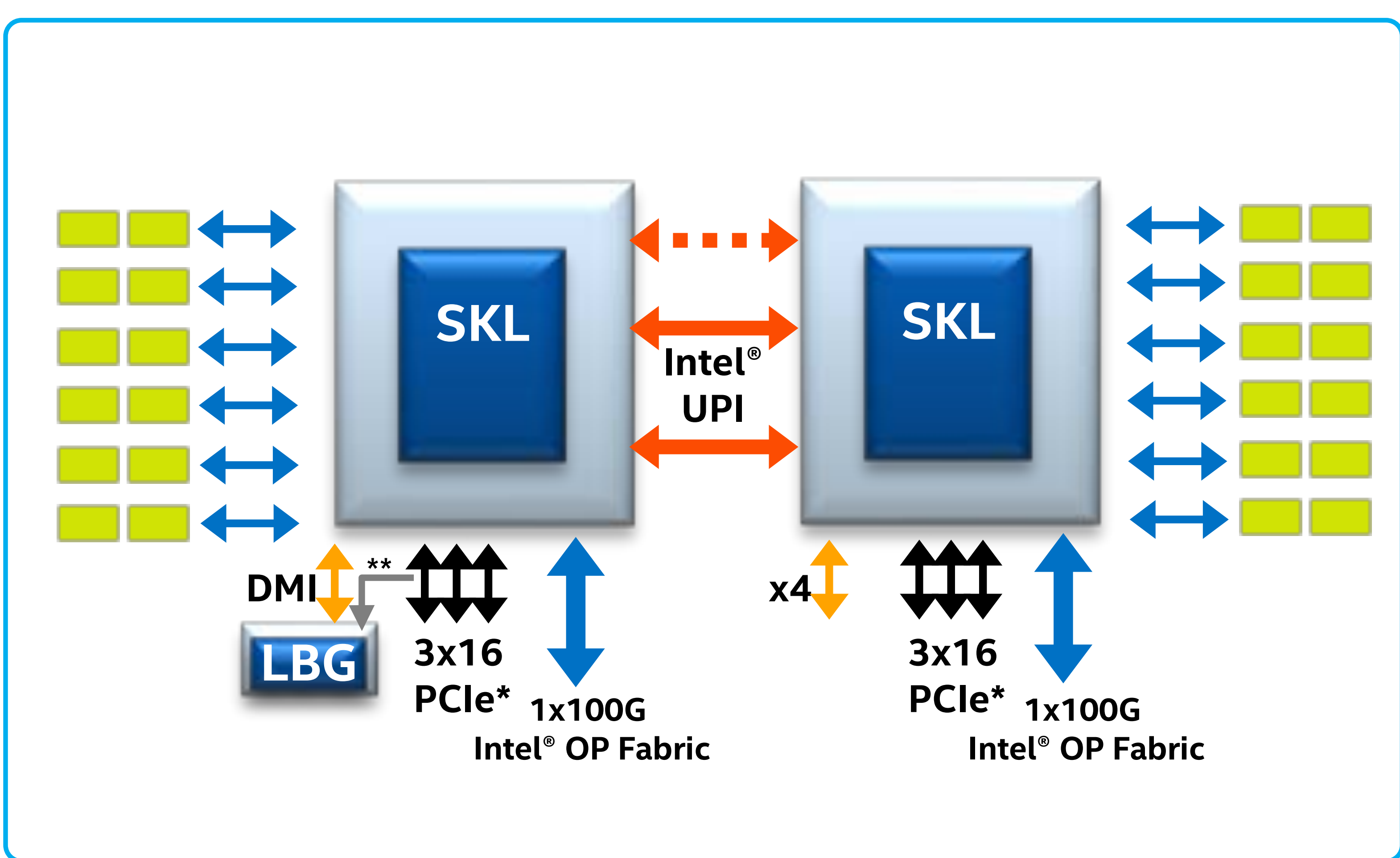
https://simplecore-ger.intel.com/swdevcon-uk/wp-content/uploads/sites/5/2017/10/UK-Dev-Con_Toby-Smith-Track-A_1000.pdf

©2020 Alex Blewitt

Non Uniform Memory Architecture (NUMA)

Platform Topologies

2S Configurations



(2S-2UPI & 2S-3UPI shown)

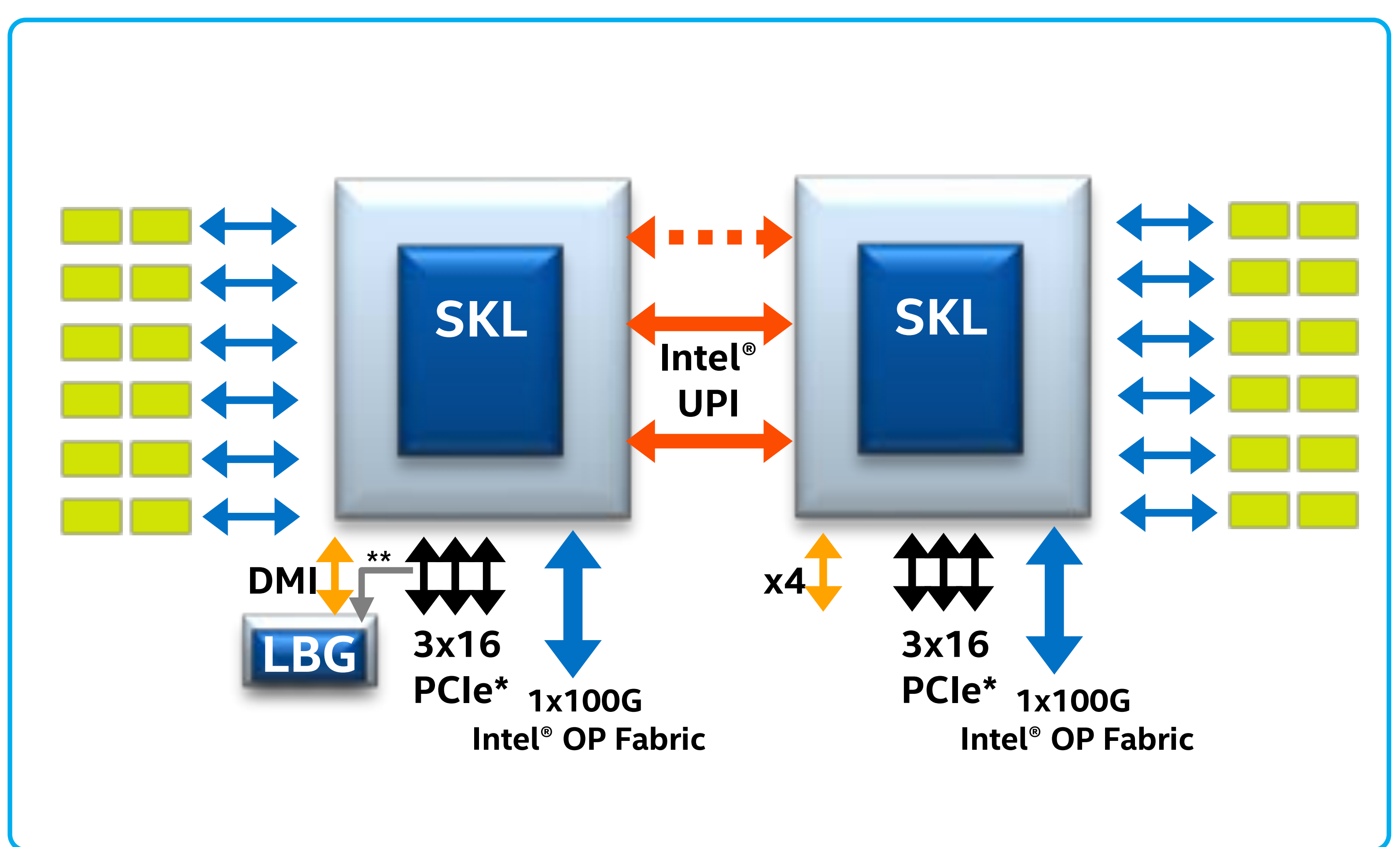
https://simplecore-ger.intel.com/swdevcon-uk/wp-content/uploads/sites/5/2017/10/UK-Dev-Con_Toby-Smith-Track-A_1000.pdf

©2020 Alex Blewitt

Non Uniform Memory Architecture (NUMA)

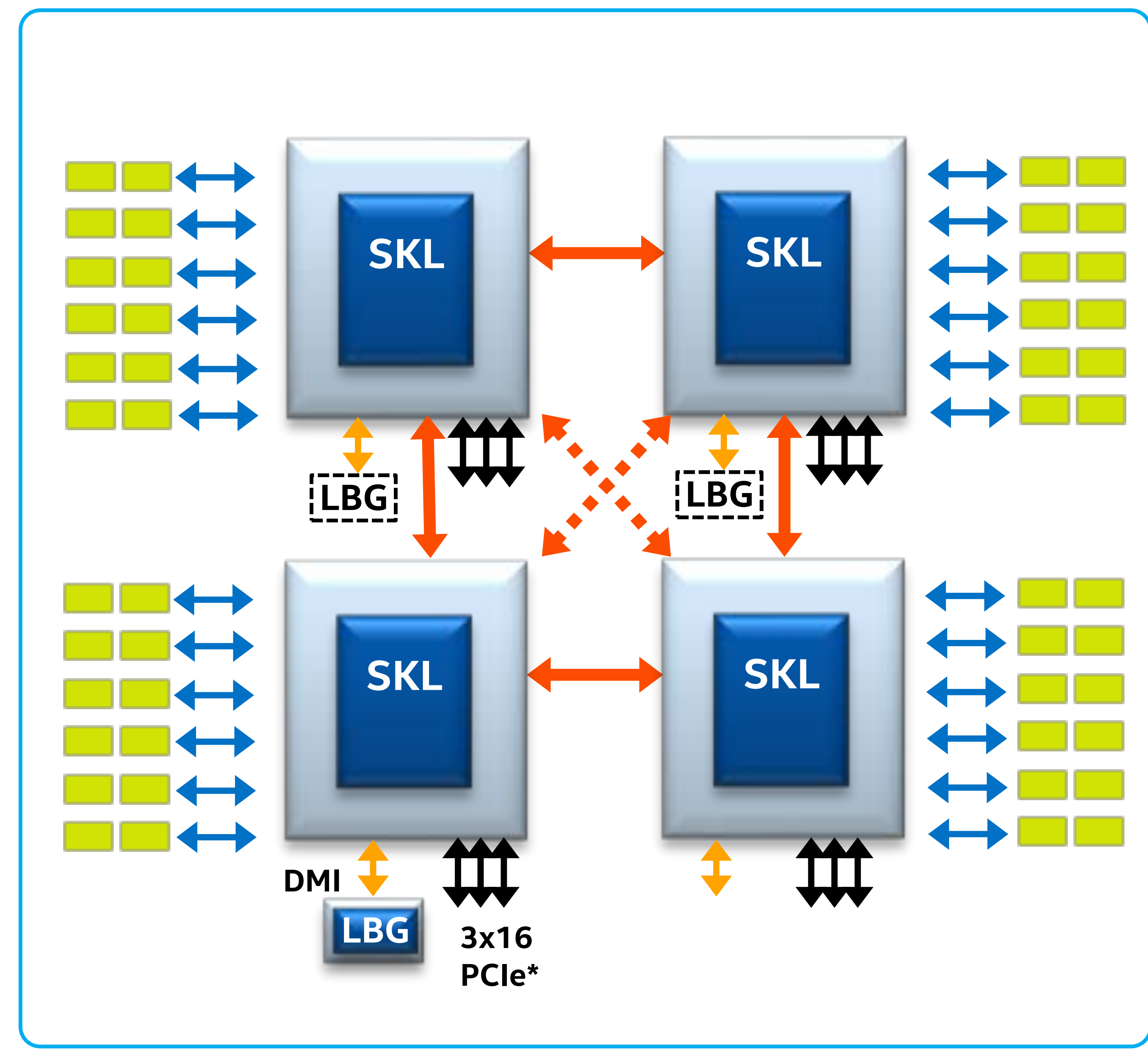
Platform Topologies

2S Configurations



(2S-2UPI & 2S-3UPI shown)

4S Configurations



(4S-2UPI & 4S-3UPI shown)

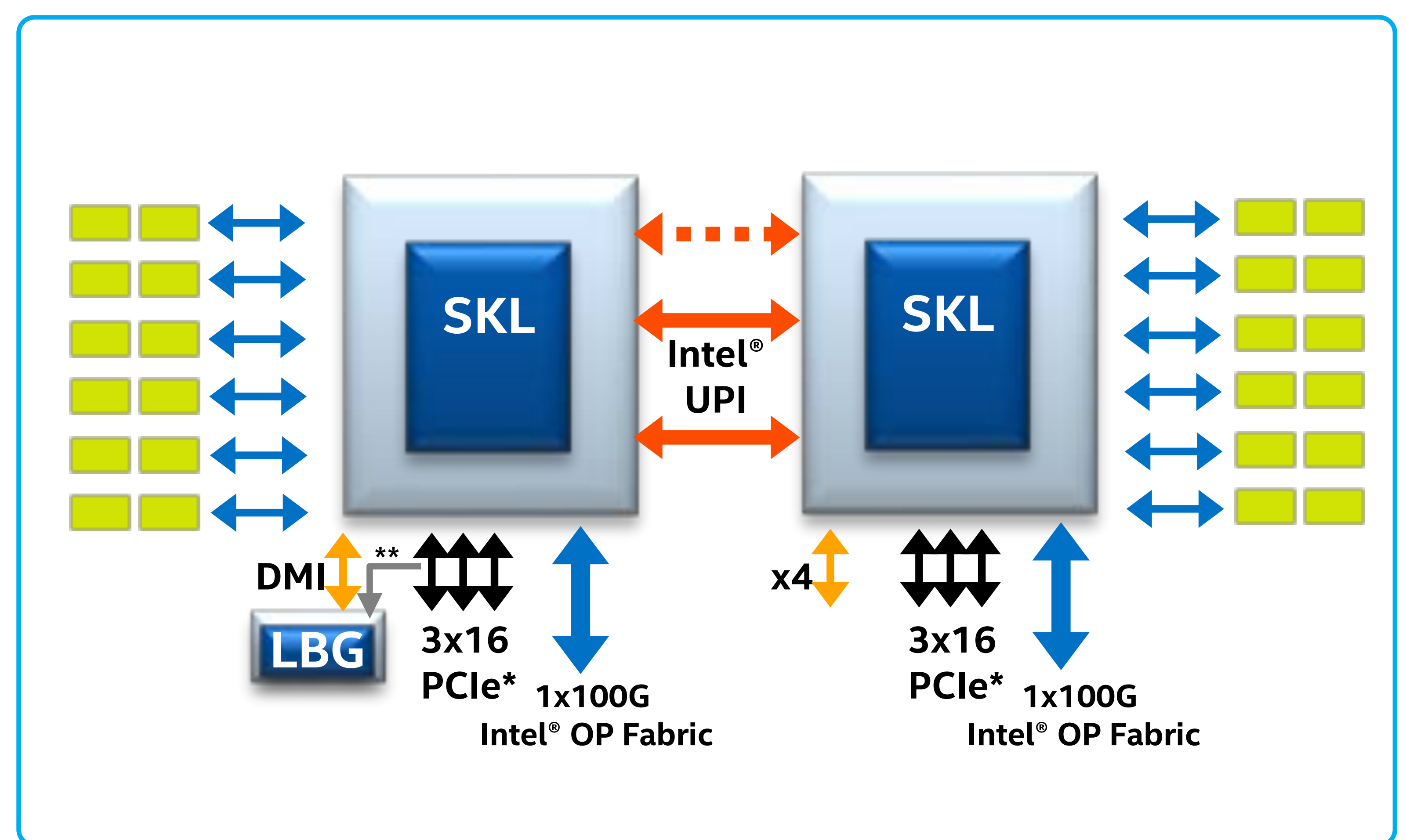
https://simplecore-ger.intel.com/swdevcon-uk/wp-content/uploads/sites/5/2017/10/UK-Dev-Con_Toby-Smith-Track-A_1000.pdf

©2020 Alex Blewitt

Non Uniform Memory Architecture (NUMA)

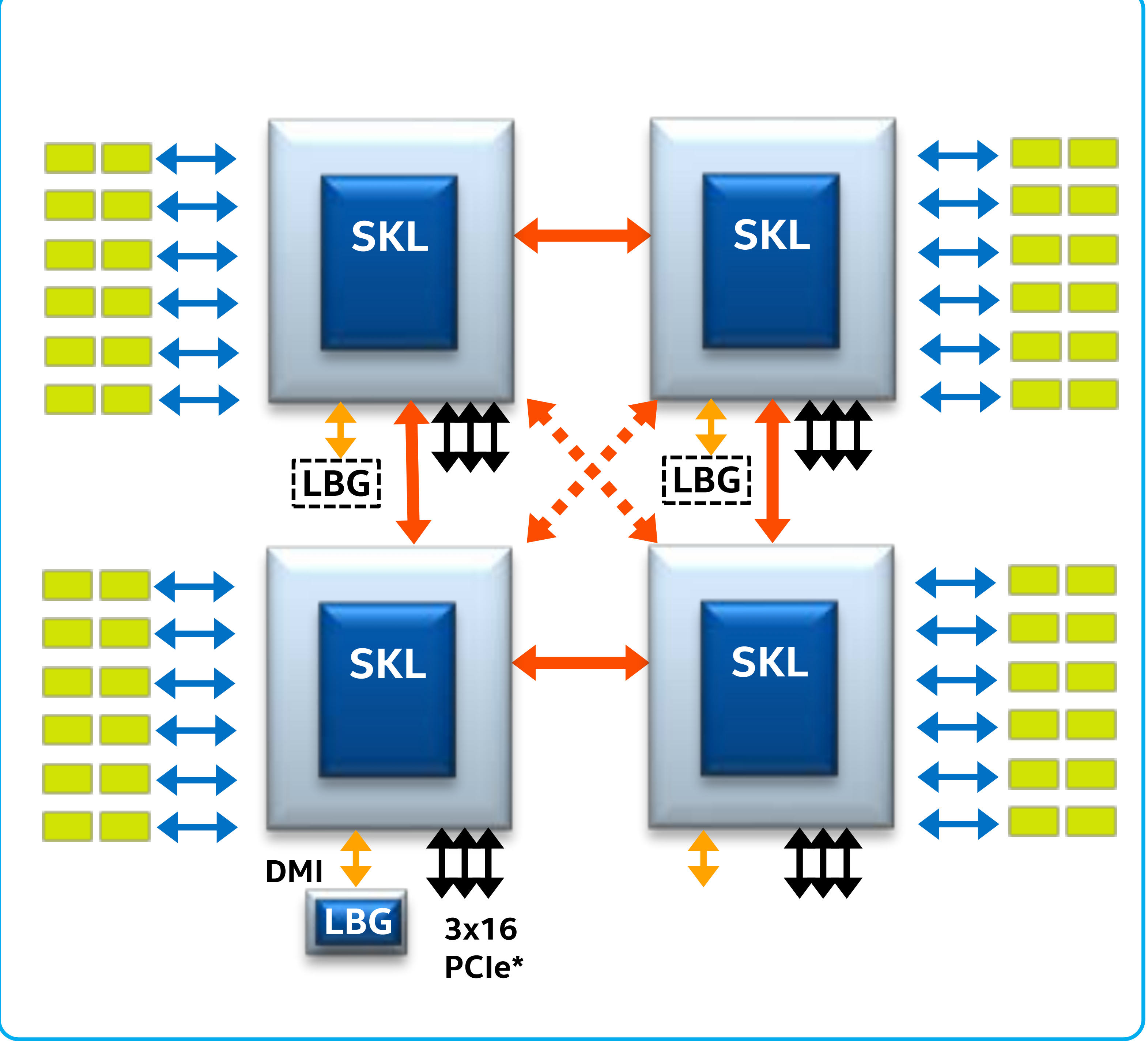
Platform Topologies

2S Configurations



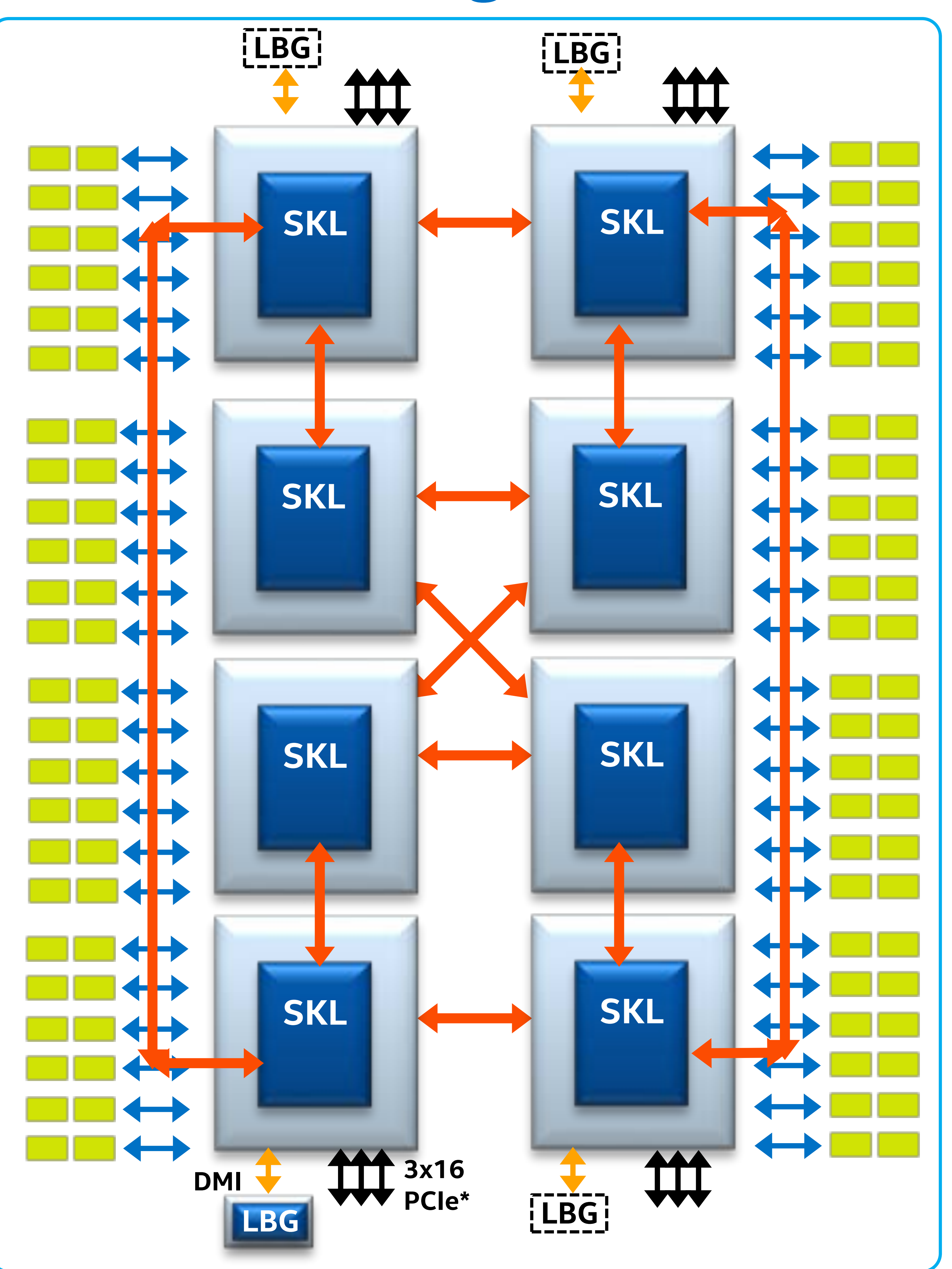
(2S-2UPI & 2S-3UPI shown)

4S Configurations



(4S-2UPI & 4S-3UPI shown)

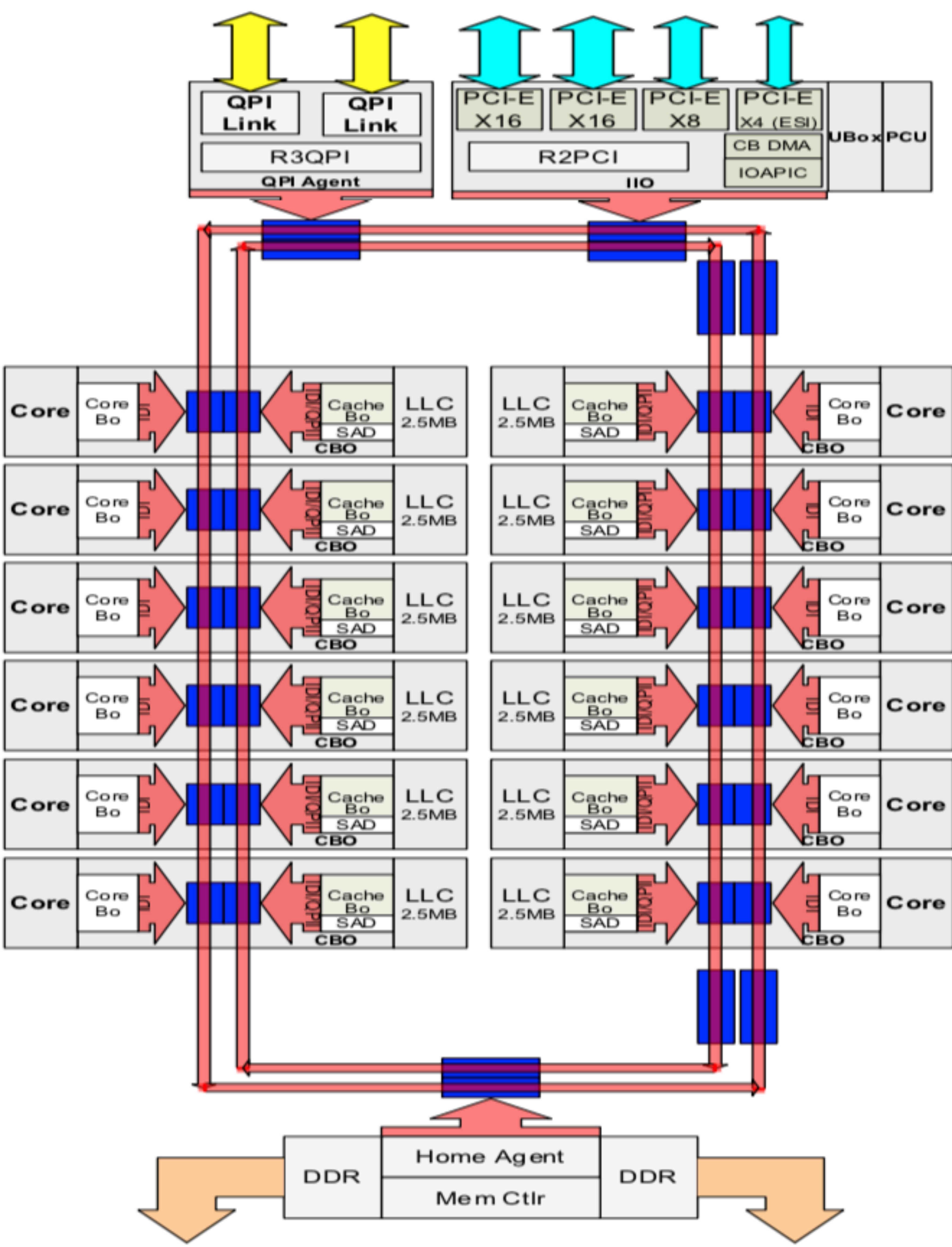
8S Configuration



https://simplecore-ger.intel.com/swdevcon-uk/wp-content/uploads/sites/5/2017/10/UK-Dev-Con_Toby-Smith-Track-A_1000.pdf

©2020 Alex Blewitt

Broadwell EX 12-core die



https://simplecore-ger.intel.com/swdevcon-uk/wp-content/uploads/sites/5/2017/10/UK-Dev-Con_Toby-Smith-Track-A_1000.pdf

©2020 Alex Blewitt

Broadwell EX 18-core die

QPI Link
QPI Link
R3QPI
QPI Agent
PCI-E X16
PCI-E X16
PCI-E X8
PCI-E X4 (ESI)
R2PCI
CB DMA
IOAPIC
IIO
UBox
PCU
Core
Core Bo
Cache Bo
SAD
CBO
LLC 2.5MB
Home Agent
Mem Ctlr
DDR

https://simplecore-ger.intel.com/swdevcon-uk/wp-content/uploads/sites/5/2017/10/UK-Dev-Con_Toby-Smith-Track-A_1000.pdf

©2020 Alex Blewitt

Broadwell EX 24-core die

QPI Link
QPI Link
R3QPI
QPI Agent
PCI-E X16
PCI-E X16
PCI-E X8
PCI-E X4 (ESI)
R2PCI
CB DMA
IOAPIC
IIO
UBox
PCU

Core | Core Bo | LLC 2.5MB | Cache Bo SAD | CBO

Home Agent
Mem Ctlr
DDR
DDR

https://simplecore-ger.intel.com/swdevcon-uk/wp-content/uploads/sites/5/2017/10/UK-Dev-Con_Toby-Smith-Track-A_1000.pdf

©2020 Alex Blewitt

Cascade/Skylake 10-core die

CHA – Caching and Home Agent ; SF – Snoop Filter; LLC – Last Level Cache;
SKX Core – Skylake Server Core; UPI – Intel® UltraPath Interconnect

https://simplecore-ger.intel.com/swdevcon-uk/wp-content/uploads/sites/5/2017/10/UK-Dev-Con_Toby-Smith-Track-A_1000.pdf

©2020 Alex Blewitt

Cascade/Skylake 18-core die

2x UPI x 20
PCIe* x16
PCIe x16
DMI x 4
CBDMA
PCIe x16

CHA/SF/LLC
SKX Core
CHA/SF/LLC
SKX Core
CHA/SF/LLC
SKX Core
CHA/SF/LLC
SKX Core

DDR4
DDR4
DDR4
MC
CHA/SF/LLC
SKX Core
CHA/SF/LLC
SKX Core
MC
DDR4
DDR4
DDR4

CHA/SF/LLC
SKX Core
CHA/SF/LLC
SKX Core
CHA/SF/LLC
SKX Core
CHA/SF/LLC
SKX Core

CHA/SF/LLC
SKX Core
CHA/SF/LLC
SKX Core
CHA/SF/LLC
SKX Core
CHA/SF/LLC
SKX Core

CHA/SF/LLC
SKX Core
CHA/SF/LLC
SKX Core
CHA/SF/LLC
SKX Core
CHA/SF/LLC
SKX Core

CHA – Caching and Home Agent ; SF – Snoop Filter; LLC – Last Level Cache;
SKX Core – Skylake Server Core; UPI – Intel® UltraPath Interconnect

https://simplecore-ger.intel.com/swdevcon-uk/wp-content/uploads/sites/5/2017/10/UK-Dev-Con_Toby-Smith-Track-A_1000.pdf

©2020 Alex Blewitt

Cascade/Skylake 28-core die

https://simplecore-ger.intel.com/swdevcon-uk/wp-content/uploads/sites/5/2017/10/UK-Dev-Con_Toby-Smith-Track-A_1000.pdf

Cascade 56 core ‘die’

2x UPI x 20	PCIe* x16	PCIe x 16 DMI x 4 CBDMA	On Pkg PCIe x16	1x UPI x 20	PCIe x 16
CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core
DDR4 DDR4 DDR4 MC	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	MC DDR4 DDR4 DDR4
CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core
CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core
CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core

https://simplecore-ger.intel.com/swdevcon-uk/wp-content/uploads/sites/5/2017/10/UK-Dev-Con_Toby-Smith-Track-A_1000.pdf

©2020 Alex Blewitt

Cascade 56 core package

Package

Die

2x UPI x 20	PCIe* x16	PCIe x16 DMI x 4 CBDMA	On Pkg PCIe x16	1x UPI x 20	PCIe x16
CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core
DDR4 DDR4 DDR4 MC	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	MC DDR4 DDR4 DDR4
CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core
CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core
CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core

Die

2x UPI x 20	PCIe* x16	PCIe x16 DMI x 4 CBDMA	On Pkg PCIe x16	1x UPI x 20	PCIe x16
CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core
DDR4 DDR4 DDR4 MC	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	MC DDR4 DDR4 DDR4
CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core
CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core
CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core	CHA/SF/LLC SKX Core

©2020 Alex Blewitt

QCon
LONDON by InfoQ
Clock Cycles
Memory and Cache ($)
Information for Cascade/Skylake systems
1
Register file
180 Integer
168 Floating Point
4
4
L1 Data (L1D$)
32 KiB 8-way
L1 Instruction (L1I$)
32 KiB 8-way
12
L2$
1 MiB 16-way
Inclusive
150
40
L3$ (LLC)
1.375 MiB 11-way
Non-inclusive
RAM
RAM
Non-inclusive
RAM
50
Non-inclusive
RAM
300
Non-inclusive
©2020 Alex Blewitt
@alblue

lstopo --no-io

Single socket system

Shared memory between CPU and GPU

Cache levels

Four core processor

HyperThreads

Machine (16GB)

Package P#0

L4 (128MB)

L3 (6144KB)

L2 (256KB)	L2 (256KB)	L2 (256KB)	L2 (256KB)
L1d (32KB)	L1d (32KB)	L1d (32KB)	L1d (32KB)
L1i (32KB)	L1i (32KB)	L1i (32KB)	L1i (32KB)
Core P#0	Core P#1	Core P#2	Core P#3
PU P#0	PU P#1	PU P#2	PU P#3
PU P#4	PU P#5	PU P#6	PU P#7

Virtual	Physical	PCID
00008000(1234)	5e38450c(1234)	10
~~00008000(1234)~~	~~48656c6f(1234)~~	20
ffffffffffb(8080)	2345ffffffb(8080)	0

Memory Pages

Two layer page table structure shown
x86_64 has 4 level paging (48 bits, 256TiB virtual, 64TiB real)
Ice Lake processors support 5 level paging (57 bits, 128Pb virtual, 4PiB real)

©2020 Alex Blewitt

Huge Pages

Pages can be 4K, 2M or 1G

`grep /proc/cpuinfo`
`pse:` 2M support
`pdpe1g:` **1G** support

0000

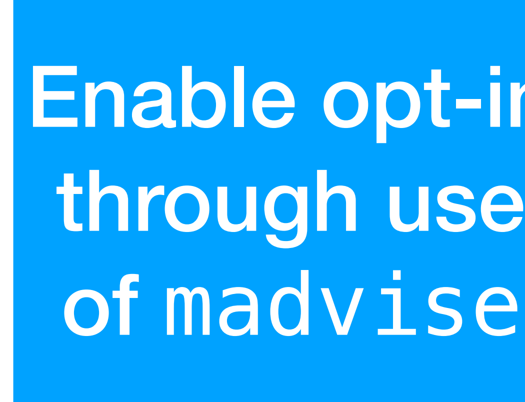
- 👍 Better use of TLB
- 👍 Fewer memory cache misses
- 👎 More complex to set up
- 👎 May waste memory
- 👎 Hugetblfs needs to be configured

©2020 Alex Blewitt

- Requires kernel configuration to reserve memory ahead of time
  - Boot parameter `hugepages=N` puts aside memory for huge page use
  - Boot parameter `hugepagesz={2M,1G}` specifies huge page size
- Requires a `hugetblfs` mount to be provided
- Requires root (or suitably permissioned app) to use hugepages

Transparent Huge Pages

- Does not require boot time configuration or special permissions
  - khugepaged assembles contiguous physical memory for large pages
- Default page size is still 4k, but processes can `madvise()` use of large pages
  - Allows specific apps to opt-in on demand
  - Benefits of smaller TLB with less wasted memory



```
# echo madvise > /sys/kernel/mm/transparent_hugepages/enabled
# echo defer >   /sys/kernel/mm/transparent_hugepage/defrag
```

Defer instead of blocking large page request

Cache lines, loads and stores

- Unit of granularity of a cache entry is 64 bytes (512 bits)
  - Even if you only read/write 1 byte you're writing 64 bytes
- Cache lines can generally be in different states:
  - ➡ M – exclusively owned by that core, and modified (dirty)
  - ➡ E – exclusively owned by that core, but not modified
  - ➡ S – shared read-only with other cores
  - ➡ I – invalid, cache line not used

©2020 Alex Blewitt

Memory prefetching (CPU)

CPU issues automatic prefetch for streamed data

Also notices striding by certain amounts as well

Can also use `__builtin_prefetch` to explicitly suggest prefetching memory elsewhere but needs to be a measured improvement

False sharing

- Two cores trying to write to bytes in the same cache-line will thrash
  - First thread will try to acquire exclusive ownership of cache line
  - Second thread (on different core) will try to do the same
  - ~~Updates may be lost if writes are not atomic *~~
  - Performance will suffer when cache line repeatedly moved
- Avoid by padding to at least cacheline size * 2 (128 bytes) for writes

* **UPDATE**: This confused more than it helped. Synchronisation and false sharing are orthogonal.

Memory performance strategies

- Ensure data fits in L1/L2/L3 cache where possible

- Stream or stride through data in a single pass if possible
- Consider pivoting data (array-of-structs or structs-of-arrays)
- Add padding for multi-threaded contended writes
    - Prefer thread-local or cpu-local accumulators with final merge step
- Compress data where practical (compressed pointers)

©2020 Alex Blewitt

Pinning memory/threads

- Pinning memory or threads to a particular core can improve performance
  - Reduces intra-core memory ownership traffic
  - Less likely to have cache invalidations
- `isolcpu` allows reservation of CPUs for non-kernel use with `cpusets`
- `taskset` allows binding of a process to specific cores
- `numactl` allows cores/memory to be clamped for a process
- `libnuma` has additional affinity settings for programmatic use

©2020 Alex Blewitt

Core

x86_64

L1 Instruction
32 KiB 8-way

Frontend

μop

L1 Data
32 KiB 8-way

Backend

©2020 Alex Blewitt

Core

55 48 89 e5 fe 04 25 d2
04 00 00 41 6c 42 6c 75
Core
x86_64
L1 Instruction
32 KiB 8-way
Pre-decode
Instructions
µop decoders
branch
prediction
µop cache
loop decode
µop
L1 Data
32 KiB 8-way
Backend

Core

x86_64

L1 Instruction
32 KiB 8-way

```
55|48 89 e5|fe 04 25 d2
04 00 00|41 6c 42 6c 75
```

Instructions

µop decoders

branch prediction

µop cache

loop decode

µop

Backend

L1 Data
32 KiB 8-way

©2020 Alex Blewitt

Core

Backend

©2020 Alex Blewitt

Core

x86_64

L1 Instruction
32 KiB 8-way

Pre-decode

Instructions

µop decoders

incb 0x4d2

branch
prediction

µop cache

loop decode

µop

L1 Data
32 KiB 8-way

Backend

©2020 Alex Blewitt

Core

x86_64

L1 Instruction
32 KiB 8-way

Pre-decode

Instructions

TMP = [0x4d2]

INC TMP

[0x4d2] = TMP

branch
prediction

µop cache

loop decode

µop

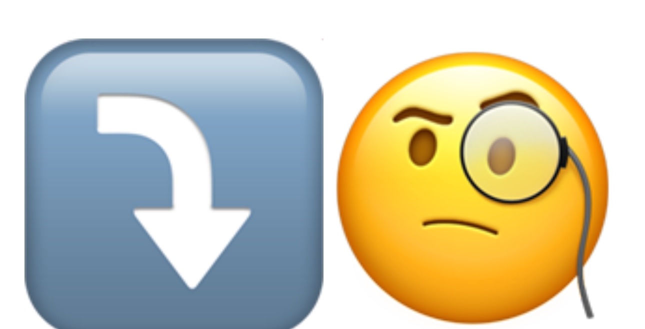
L1 Data
32 KiB 8-way

Backend

©2020 Alex Blewitt

Core

©2020 Alex Blewitt

Branch Prediction 

- Correct 95% of the time
  - Queues up instructions assuming the branch has been taken
- Learns patterns in code based on existing behaviour
  - Iterating through predictable (sorted) data may be more efficient
- Throws away inaccurate work if incorrect
  - May cause observable side channel behaviour e.g. cache invalidation 👻

```
cmp eax,42; jne
```

©2020 Alex Blewitt

Branch Target Predictor 🎯

- Predicts where the target is going if taken
- Hard coded addresses/offsets always predictable
  - Jump to location of register may be more difficult
  - Often seen when jumping through object oriented code
- Inlining is the master optimisation because it avoids unknowable branches

©2020 Alex Blewitt

Core

Frontend

Allocate
Rename
Retire

L1 Instruction
32 KiB 8-way

reorder buffer

register files

Port 0 and 1 can be fused for a 512 bit operation
Port 5 is a 512 bit wide operation
All others handle 256 bits

Port	Integer	Unit			Floating	Unit		
0	ALU	*LEA*	Shift	Branch	ALU	FMA	Shift	Divide
1	ALU	LEA	Multiply	Divide	ALU	FMA	Shift	*Shuffle*
5	ALU	LEA	*Multiply*		ALU	FMA		Shuffle
6	ALU	*LEA*	Shift	Branch				

L1 Data
32 KiB 8-way

load buffer

store buffer

Address generation

2
3
4
7
8
9

Scheduler

Port 8 and 9 added in Ice Lake

Execution units added in Ice Lake

©2020 Alex Blewitt

Core

Port 0 and 1 can be fused for a 512 bit operation
Port 5 is a 512 bit wide operation
All others handle 256 bits

Port	Integer Unit				Floating Point Unit			
0	ALU	*LEA*	Shift	Branch	ALU	FMA	Shift	Divide
1	ALU	LEA	Multiply	Divide	ALU	FMA	Shift	*Shuffle*
5	ALU	LEA	*Multiply*		ALU	FMA		Shuffle
6	ALU	*LEA*	Shift	Branch				

Execution units added in Ice Lake

©2020 Alex Blewitt

Core

Frontend

Allocate
Rename
Retire

R99 = [0x4d2]   INC R99   [0x4d2] = R99

…e …ration
Port 5 is a 512 bit wide operation
All others handle 256 bits

reorder buffer

register files

L1 Instruction
32 KiB 8-way

L1 Data
32 KiB 8-way

load buffer

store buffer

Address generation

Scheduler

Port	Integer	Unit			Floating	Unit		
0	ALU	*LEA*	Shift	Branch	ALU	FMA	Shift	Divide
1	ALU	LEA	Multiply	Divide	ALU	FMA	Shift	*Shuffle*
5	ALU	LEA	*Multiply*		ALU	FMA		Shuffle
6	ALU	*LEA*	Shift	Branch				

Scheduler ports 2, 3, 4, 7, 8, 9 connect to the load buffer and store buffer.

Execution units added in Ice Lake

Port 8 and 9 added in Ice Lake

©2020 Alex Blewitt

Core

Frontend

Allocate
Rename
Retire

reorder buffer

register files

Port 0 and 1 can be fused for a 512 bit operation
Port 5 is a 512 bit wide operation
All others handle 256 bits

Integer	Unit			Floating	Unit		
			Branch	ALU	FMA	Shift	Divide
			Divide	ALU	FMA	Shift	*Shuffle*
ALU	LEA	*Multiply*		ALU	FMA		Shuffle
ALU	*LEA*	Shift	Branch				

INC R99

load buffer

R99 = [0x4d2]

generation

L1 Instruction
32 KiB 8-way

L1 Data
32 KiB 8-way

store buffer

[0x4d2] = R99

Scheduler

2 3 4 7 8 9

0 1 5 6

added in Ice Lake

Execution units added in Ice Lake

©2020 Alex Blewitt

Core

Frontend

Allocate
Rename
Retire

L1 Instruction
32 KiB 8-way

reorder buffer

register files

Port 0 and 1 can be fused for a 512 bit operation
Port 5 is a 512 bit wide operation
All others handle 256 bits

L1 Data
32 KiB 8-way

load buffer

store buffer

R99 = 2A

generation

[0x4d2] = R99

Port 8 and 9 added in Ice Lake

Scheduler

INC R99

Port	Integer	Unit			Floating	Unit		
0				Branch	ALU	FMA	Shift	Divide
1	ALU	LEA	Multiply	Divide	ALU	FMA	Shift	*Shuffle*
5	ALU	LEA	*Multiply*		ALU	FMA		Shuffle
6	ALU	*LEA*	Shift	Branch				

Execution units added in Ice Lake

©2020 Alex Blewitt

Core

Frontend

L1 Instruction
32 KiB 8-way

Allocate
Rename
Retire

reorder buffer

R99 = 2A

Port 0 and 1 can be fused for a 512 bit operation
Port 5 is a 512 bit wide operation
All others handle 256 bits

Integer	Unit			Floating	Unit		
			Branch	ALU	FMA	Shift	Divide
			Divide	ALU	FMA	Shift	*Shuffle*
ALU	LEA	*Multiply*		ALU	FMA		Shuffle
ALU	*LEA*	Shift	Branch				

INC R99

L1 Data
32 KiB 8-way

load buffer

Address generation

store buffer

Scheduler

2 3 4 7 8 9

0 1 5 6

[0x4d2] = R99

added in Ice Lake

Execution units added in Ice Lake

©2020 Alex Blewitt

Core

Frontend

L1 Instruction
32 KiB 8-way

Allocate
Rename
Retire

reorder buffer

register files

R99 = 2B

Port 0 and 1 can be fused for a 512 bit operation
Port 5 is a 512 bit wide operation
All others handle 256 bits

Port	Integer	Unit			Floating	Unit		
0	ALU	*LEA*	Shift	Branch	ALU	FMA	Shift	Divide
1	ALU	LEA	Multiply	Divide	ALU	FMA	Shift	*Shuffle*
5	ALU	LEA	*Multiply*		ALU	FMA		Shuffle
6	ALU	*LEA*	Shift	Branch				

Execution units added in Ice Lake

L1 Data
32 KiB 8-way

load buffer

Address generation

store buffer

[0x4d2] = R99

Scheduler

2 3 4 7 8 9

Port 8 and 9 added in Ice Lake

©2020 Alex Blewitt

Core

Frontend

Allocate
Rename
Retire

L1 Instruction
32 KiB 8-way

reorder buffer

register files

Port 0 and 1 can be fused for a 512 bit operation
Port 5 is a 512 bit wide operation
All others handle 256 bits

Port	Integer	Unit			Floating	Unit		
0	ALU	*LEA*	Shift	Branch	ALU	FMA	Shift	Divide
1	ALU	LEA	Multiply	Divide	ALU	FMA	Shift	*Shuffle*
5	ALU	LEA	*Multiply*		ALU	FMA		Shuffle
6	ALU	*LEA*	Shift	Branch				

L1 Data
32 KiB 8-way

load buffer

Address generation

store buffer

Scheduler

2 3 4 7 8 9

`[0x4d2] = 2B`

Port 8 and 9 added in Ice Lake

Execution units added in Ice Lake

©2020 Alex Blewitt

Core

Frontend

L1 Instruction
32 KiB 8-way

Allocate
Rename
Retire

reorder buffer

register files

Port 0 and 1 can be fused for a 512 bit operation
Port 5 is a 512 bit wide operation
All others handle 256 bits

Port	Integer	Unit			Floating	Unit		
0	ALU	*LEA*	Shift	Branch	ALU	FMA	Shift	Divide
1	ALU	LEA	Multiply	Divide	ALU	FMA	Shift	*Shuffle*
5	ALU	LEA	*Multiply*		ALU	FMA		Shuffle
6	ALU	*LEA*	Shift	Branch				

L1 Data
32 KiB 8-way

load buffer

Address generation

store buffer

Scheduler

Ports: 2, 3, 4, 7, 8, 9

[0x4d2] = 2B

Port 8 and 9 added in Ice Lake

Execution units added in Ice Lake

©2020 Alex Blewitt

perf

- Linux perf (compiled from linux/tools/perf, or from linux-tools/linux-perf)
  - Running in Docker requires compilation from source
- Commands available
  - `record` – record execution performance for process/pid
  - `report` – generate a report from prior recording
  - `annotate` – annotate a report from a prior recording
  - `stat` – record performance counters for process/pid

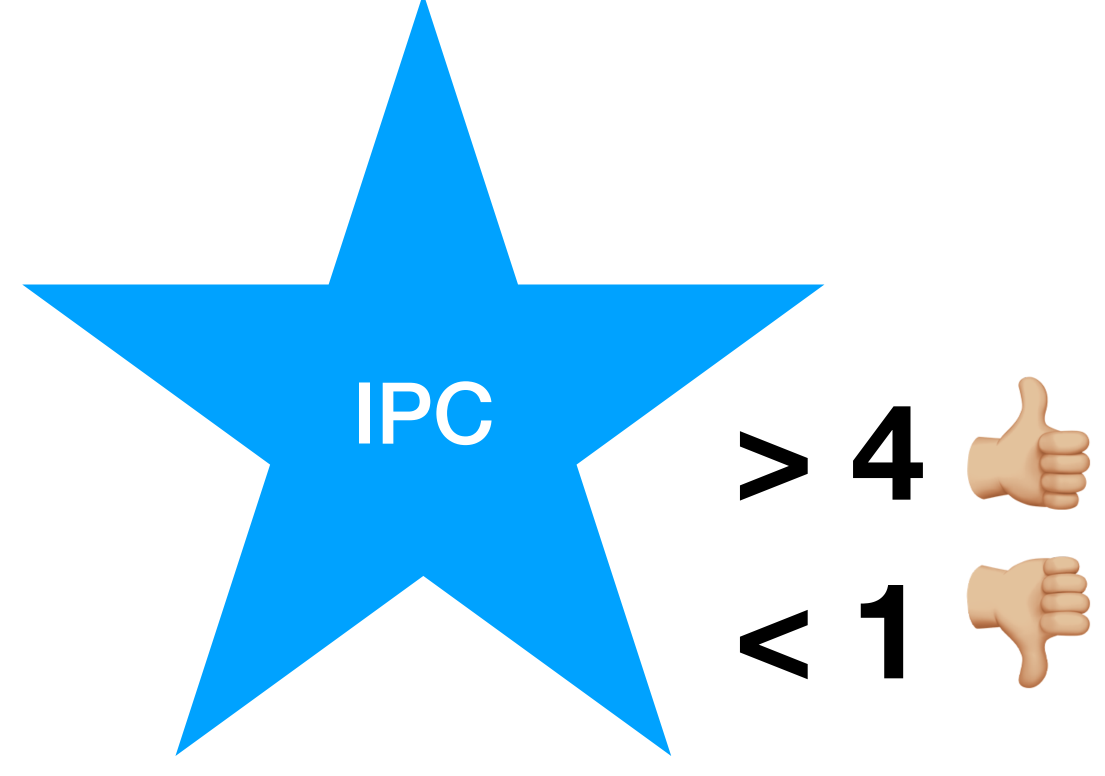
https://perf.wiki.kernel.org
https://github.com/alblue/scripts/blob/master/perf-Dockerfile

perf record

- Perf record will sample the process(es) and generate stack traces
- Events may be skewed from their location
  - Improve accuracy with `:p`, `:pp` or `:ppp` suffix to event
- Can capture branches, last branch records or use processor tracing
  - `perf record -b program`
  - `perf record --call-graph lbr -j any_call,any_ret program`
  - `perf record -e intel_pt//u program`

https://lwn.net/Articles/680985/
https://lwn.net/Articles/680996/

perf stat




```
$ perf stat base64 <(echo hello)
d29ybGQK

 Performance counter stats for 'base64 /dev/fd/63':

          0.341382      task-clock (msec)         #    0.649 CPUs utilized
                 0      context-switches          #    0.000 K/sec
                 0      cpu-migrations            #    0.000 K/sec
                65      page-faults               #    0.190 M/sec
         1,218,176      cycles                    #    3.568 GHz
           811,468      stalled-cycles-frontend   #   66.61% frontend cycles idle
           855,999      instructions              #    0.70  insn per cycle
                                                  #    0.95  stalled cycles per insn
           169,032      branches                  #  495.140 M/sec
             8,883      branch-misses             #    5.26% of all branches

       0.000526160 seconds time elapsed
```

Performance counters

- Intel cores have a few dedicated and programmable counters
  - Instruction cycles, branches, branch misses …
  - Counters can be multiplexed (read X for 1µs, read Y for 1µs)
- Programmable counters can be set to specific measurements
  - iTLB-load-misses, LLC-load-misses, uops_dispatched_port.port_5 ...
- Undocumented performance counters can be specified with events
  - cpu/event=0x3c,umask=0x0,any=1/

Top-down Microarchitecture Analysis

Locating Issues

CPU Bound ⇒ Analyze

Frontend Bound
Bad Speculation
Retiring
Backend Bound

Frontend Latency
Band-width
Branch Mispred
Machine Clears
BASE
Microcode Sequencer
Core Bound
Memory Bound

iTLB Miss
iCache Miss
Branch Rest.
DSB switch.
MS Switches
LCP
MITE
DSB
LSD
FP-arith
Other
Divider
Ports Utilization
Stores Bound
L1 Bound
L2 Bound
L3 Bound
Ext. Memory Bound

STLB hit
STLB miss
L2 hit
L2 miss
X87
Scalar
Vector
2+ ports
1 port
0 ports
Store Miss
False Sharing
dTLB Store
Store fwd blk
dTLB Load
Contested Access
Data Sharing
L3 Latency
MEM Bandwidth
MEM Latency

Have Precise events for sampling

Precise events added in Skylake

https://www.researchgate.net/publication/269302126_A_Top-Down_method_for_performance_analysis_and_counters_architecture

Ahmed Yasin

©2020 Alex Blewitt

Top-down Analysis Method

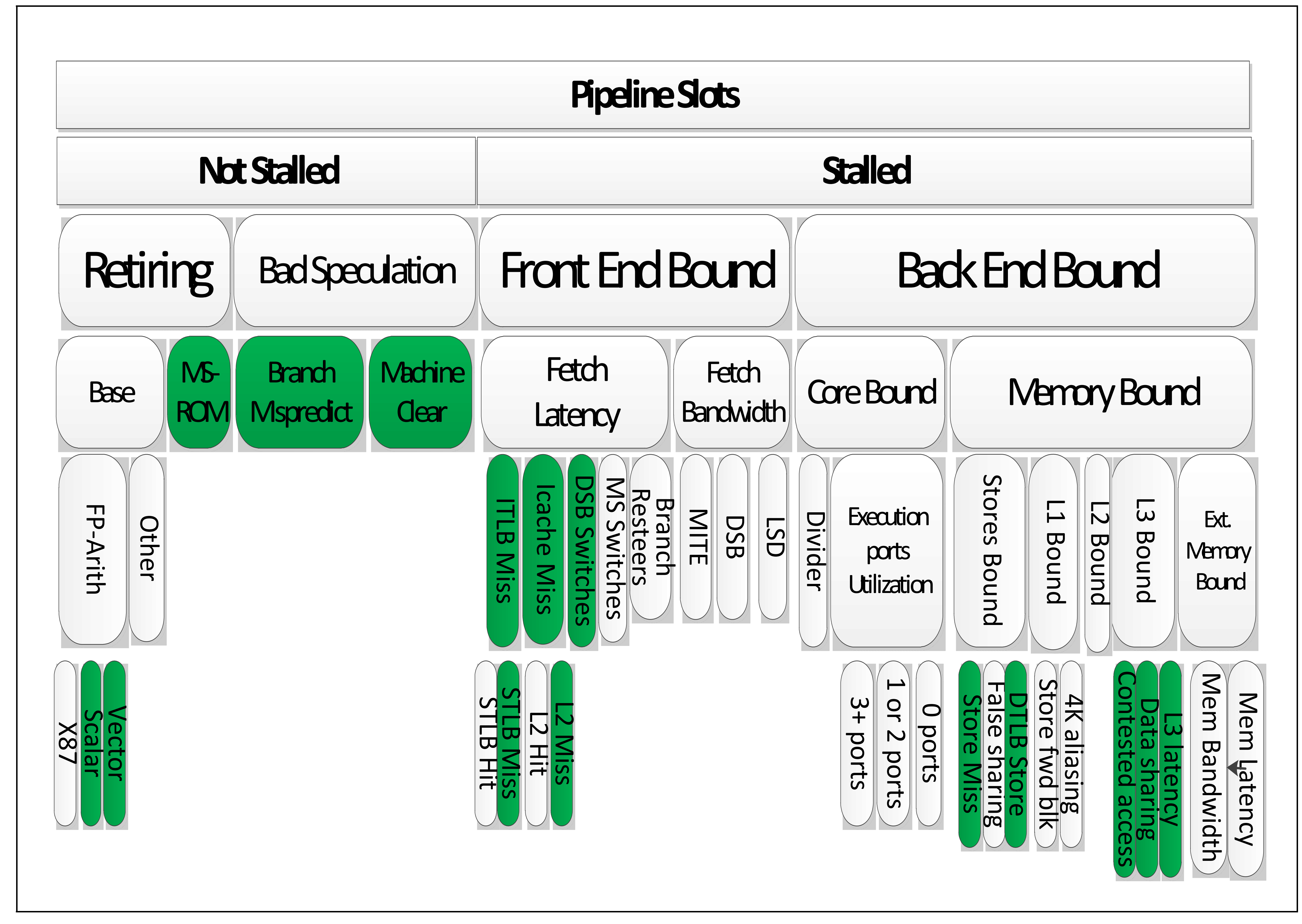


Figure B-3. TMAM Hierarchy Supported by Skylake Microarchitecture

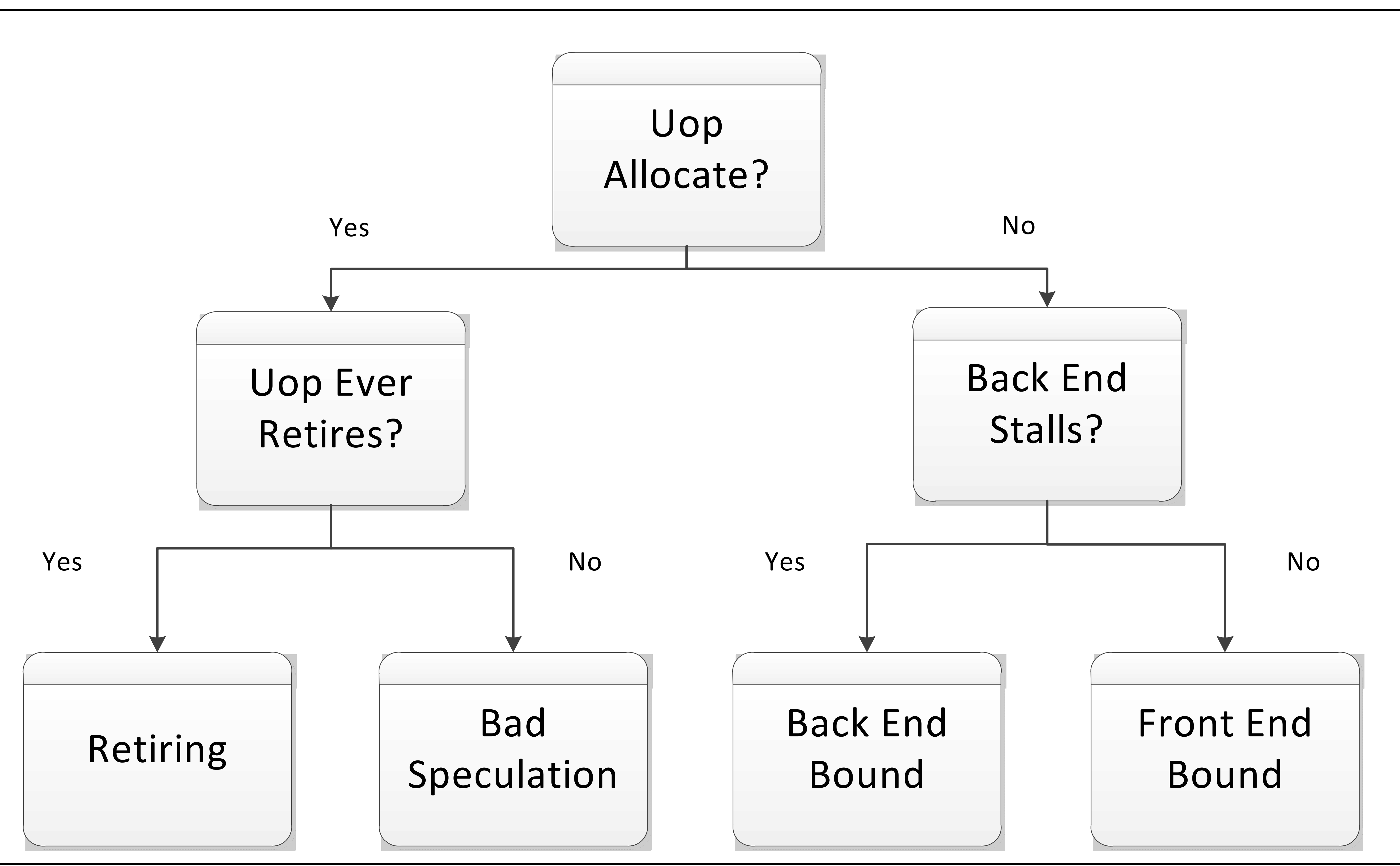


Figure B-2. TMAM's Top Level Drill Down Flowchart

https://software.intel.com/en-us/download/intel-64-and-ia-32-architectures-optimization-reference-manual

Ahmed Yasin

perf stat --topdown

```
$ perf stat -a --topdown sleep 1
nmi_watchdog enabled with topdown. May give wrong results.
Disable with echo 0 > /proc/sys/kernel/nmi_watchdog

 Performance counter stats for 'system wide':

                 retiring            bad speculation     frontend bound    backend
bound
S0-C0           2      15.3%               2.8%               32.1%             49.9%
S0-C1           2      23.3%               4.0%               27.3%             45.4%
S0-C2           2      15.2%               2.9%               29.8%             52.1%
S0-C3           2      16.7%               0.0%               31.8%             51.5%
S0-C4           2      35.7%              10.7%               26.2%             27.4%
S0-C5           2      14.9%               2.5%               34.1%             48.5%

       1.000889285 seconds time elapsed
```

©2020 Alex Blewitt

Toplev PMU tools

- Andi Kleen has written toplev.py which allows top-down analysis
  - Initial download caches processor information from download.01.org
- Uses perf to record stats, but with custom event filters
- If workload is repeatable, can use `--no-multiplex` to repeat results
- Run with `-l1`, see if issues are present, run with `-l2` ...

toplev.py --single-thread

```
$ dd if=/dev/urandom of=/tmp/rand bs=4096 count=4096

$ ./toplev.py --single-thread --no-multiplex -l1 -- base64 /tmp/rand > /dev/null
# 3.6-full on Intel(R) Xeon(R) CPU E5-1650 v2 @ 3.50GHz
BE              Backend_Bound                            % Slots       24.07  <==

$ ./toplev.py --single-thread --no-multiplex -l2 -- base64 /tmp/rand > /dev/null
BE              Backend_Bound                            % Slots       23.82
BE/Core         Backend_Bound.Core_Bound                 % Slots       16.08  <==

$ ./toplev.py --single-thread --no-multiplex -l3 -- base64 /tmp/rand > /dev/null
BE              Backend_Bound                            % Slots       23.96
BE/Core         Backend_Bound.Core_Bound                 % Slots       16.35
BE/Core         Backend_Bound.Core_Bound.Ports_Utilization  % Clocks   24.51  <==
```

©2020 Alex Blewitt

Instruction layout

`__builtin_expect(error,1)`

`__builtin_expect(error,0)`

Loop stream detector

32 bit
aliognment

Bad
Loop

Cache line

Good
Loop

Cache line

Align with

```
-mllvm -align-all-nofallthru-blocks=5
-mllvm -align-all-functions=5
```

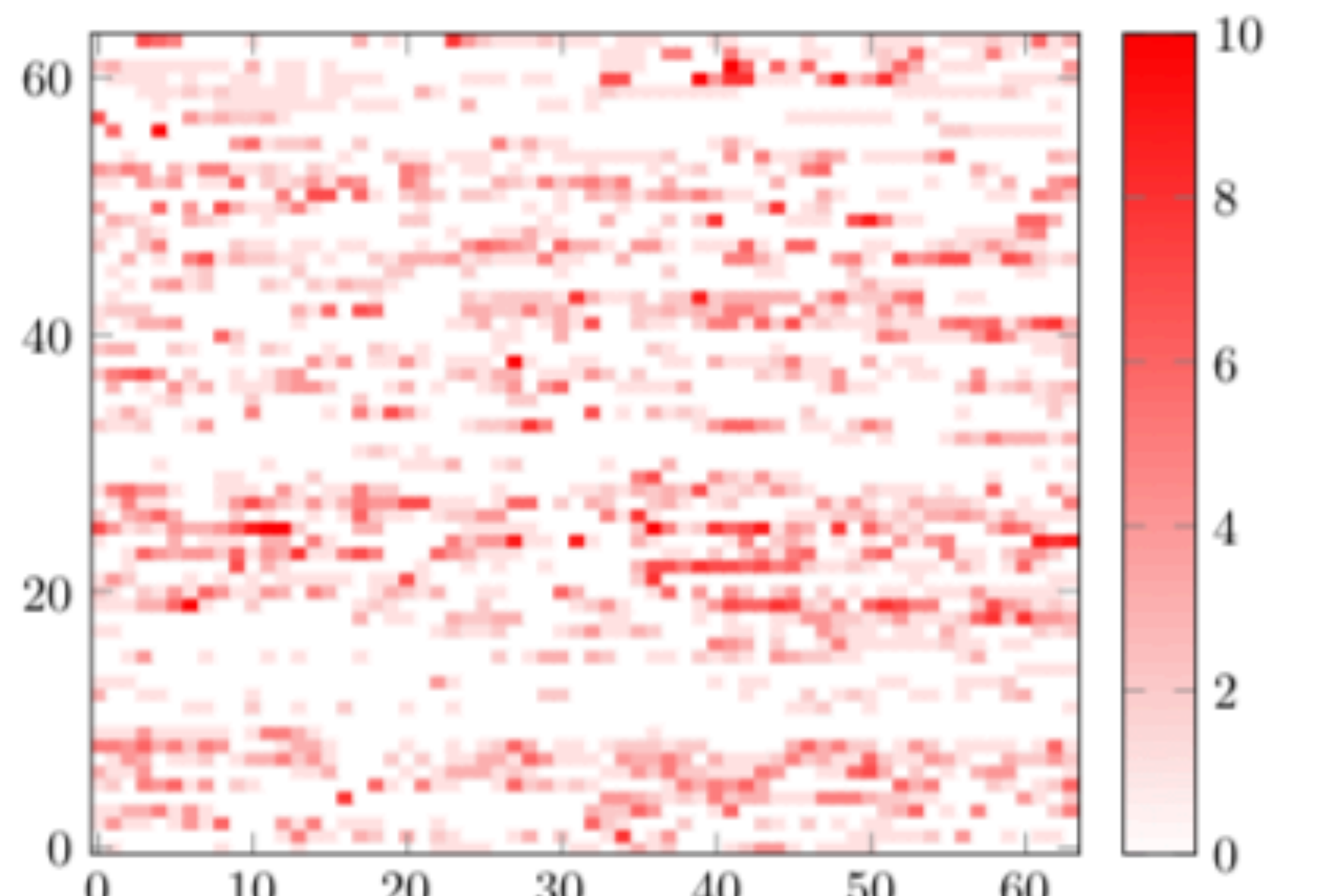
©2020 Alex Blewitt

Facebook BOLT

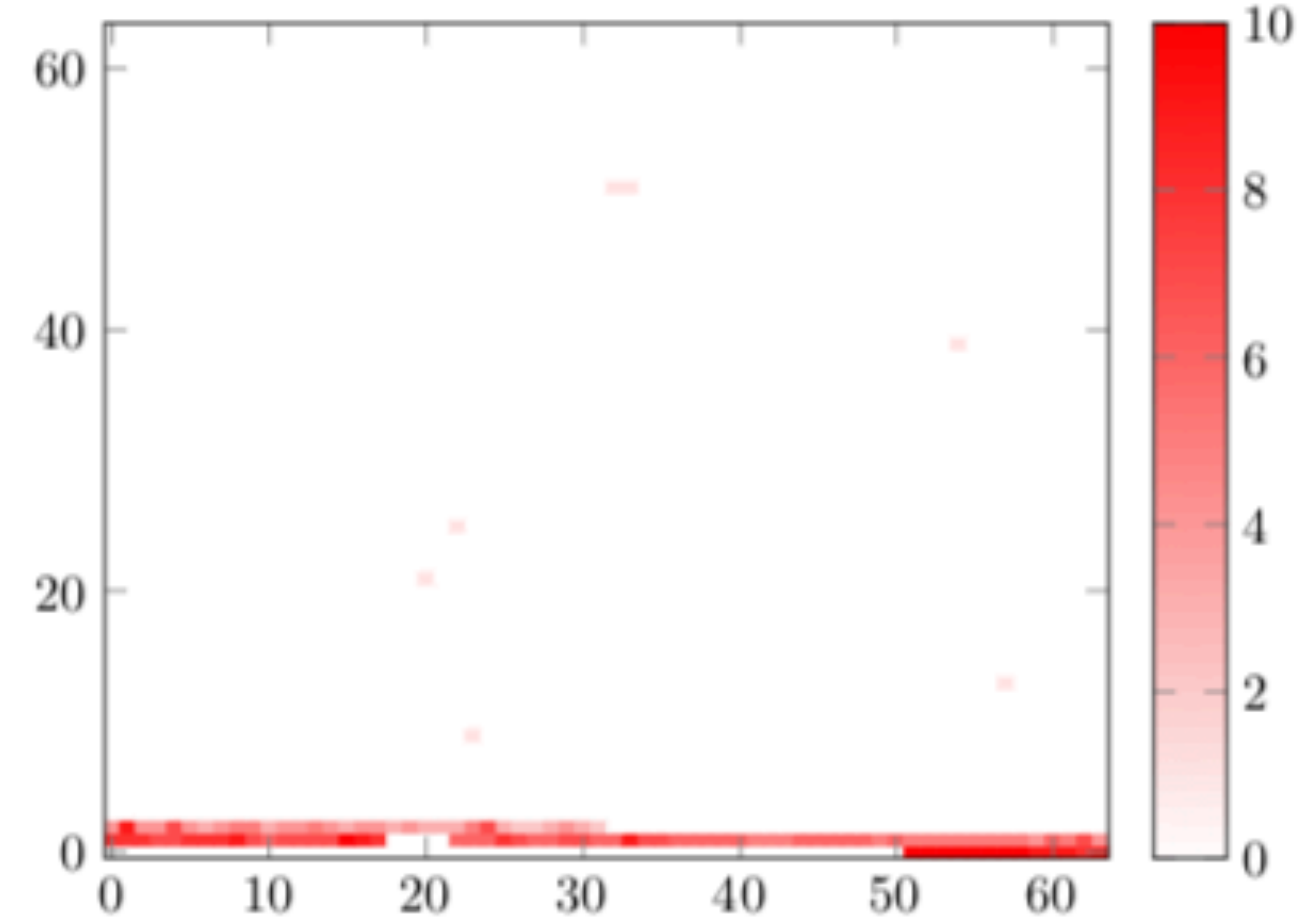
Executed instructions are distributed across icache space

After sorting basic blocks guided by profiling data, the icache space is defragmented

Figure 9: Heat maps for instruction memory accesses of the HHVM binary, without and with BOLT. Heat is a log scale.



(a) without BOLT



(b) with BOLT

https://arxiv.org/abs/1807.06735
https://github.com/facebookincubator/BOLT

©2020 Alex Blewitt

SIMD JSON parser

simdjson
RapidJSON
sajson

parsing speed (GB/s)

3.0
2.5
2.0
1.5
1.0
0.5
0.0

apache_builds.json
canada.json
citm_catalog.json
github_events.json
gsoc-2018.json
instruments.json
marine_ik.json
mesh.json
mesh.pretty.json
numbers.json
random.json
twitter.json
twitterescaped.json
update-center.json

https://arxiv.org/abs/1902.08318
https://github.com/lemire/simdjson

Summary: Memory

- Use cacheline-aligned or cacheline-aware data structures
- Compress data in memory and decompress on the fly
- Avoid random memory access when possible
- Configure huge pages and use madvise & defer
- Partition memory with libnuma for data locality

Summary: CPU

- Each CPU is its own networked mesh cluster
- Branch speculation and memory/TLB misses are costly
  - Use branch free and lock free algorithms when possible
- Analyse perf counters with top down architectural analysis
- Use (auto)vectorisation and use XMM/YMM/ZMM when sensible

References

https://alblue.bandlem.com/

https://arxiv.org/abs/1807.06735 → https://github.com/facebookincubator/BOLT

https://arxiv.org/abs/1902.08318 → https://github.com/lemire/simdjson/

https://github.com/andikleen/pmu-tools/wiki/toplev-manual

https://lwn.net/Articles/680985/ && https://lwn.net/Articles/680996/

https://perf.wiki.kernel.org

https://simplecore-ger.intel.com/swdevcon-uk/wp-content/uploads/sites/5/2017/10/UK-Dev-Con_Toby-Smith-Track-A_1000.pdf

https://software.intel.com/en-us/download/intel-64-and-ia-32-architectures-optimization-reference-manual

https://www.researchgate.net/publication/269302126_A_Top-Down_method_for_performance_analysis_and_counters_architecture

Links

https://easyperf.net/notes/

https://epickrram.blogspot.com/

https://groups.google.com/forum/#!forum/mechanical-sympathy/

https://lemire.me/en/

https://psy-lob-saw.blogspot.com/

https://richardstartin.github.io/

https://travisdowns.github.io/

https://www.agner.org/optimize/

https://www.real-logic.co.uk/

©2020 Alex Blewitt

Thank you

- 🗒️ https://alblue.bandlem.com
- 🦉 https://twitter.com/alblue
- 🐙 https://github.com/alblue
- 📺 https://vimeo.com/alblue
- 📠 https://speakerdeck.com/alblue