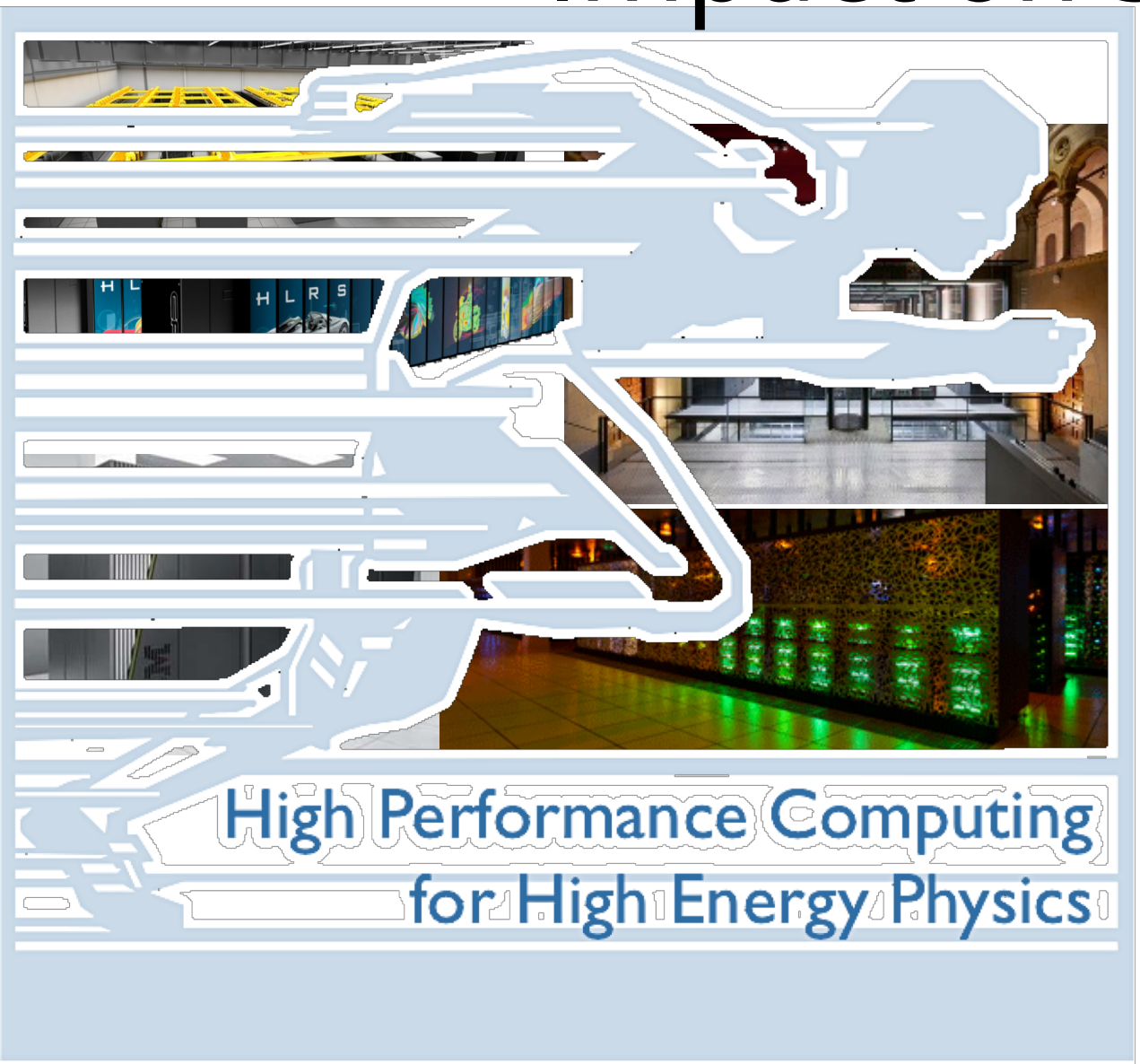
Computing Architecture and its Impact on Scientific Applications

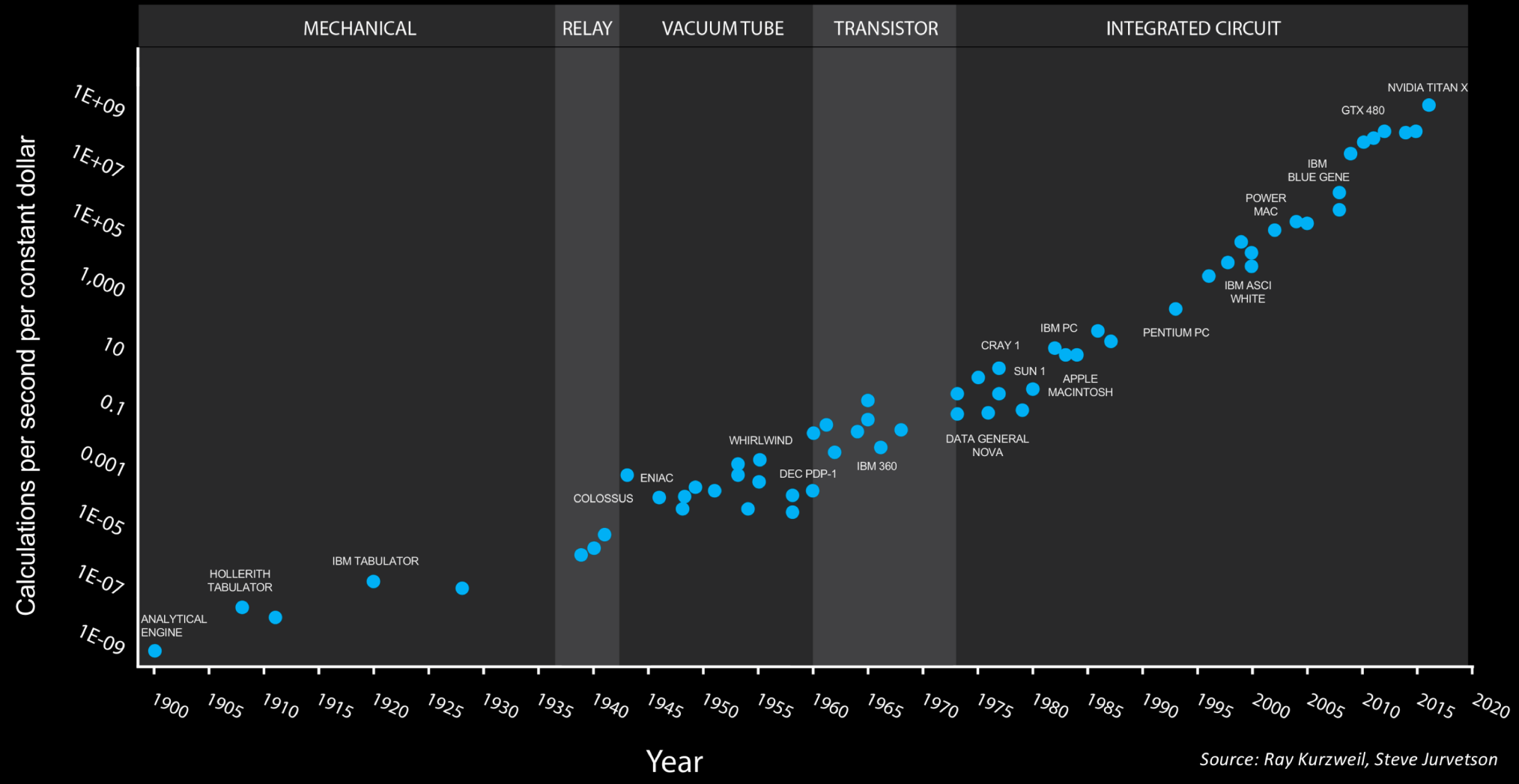


Vincenzo Innocente
CERN
CMS Experiment

ESC, Bertinoro, October 2018

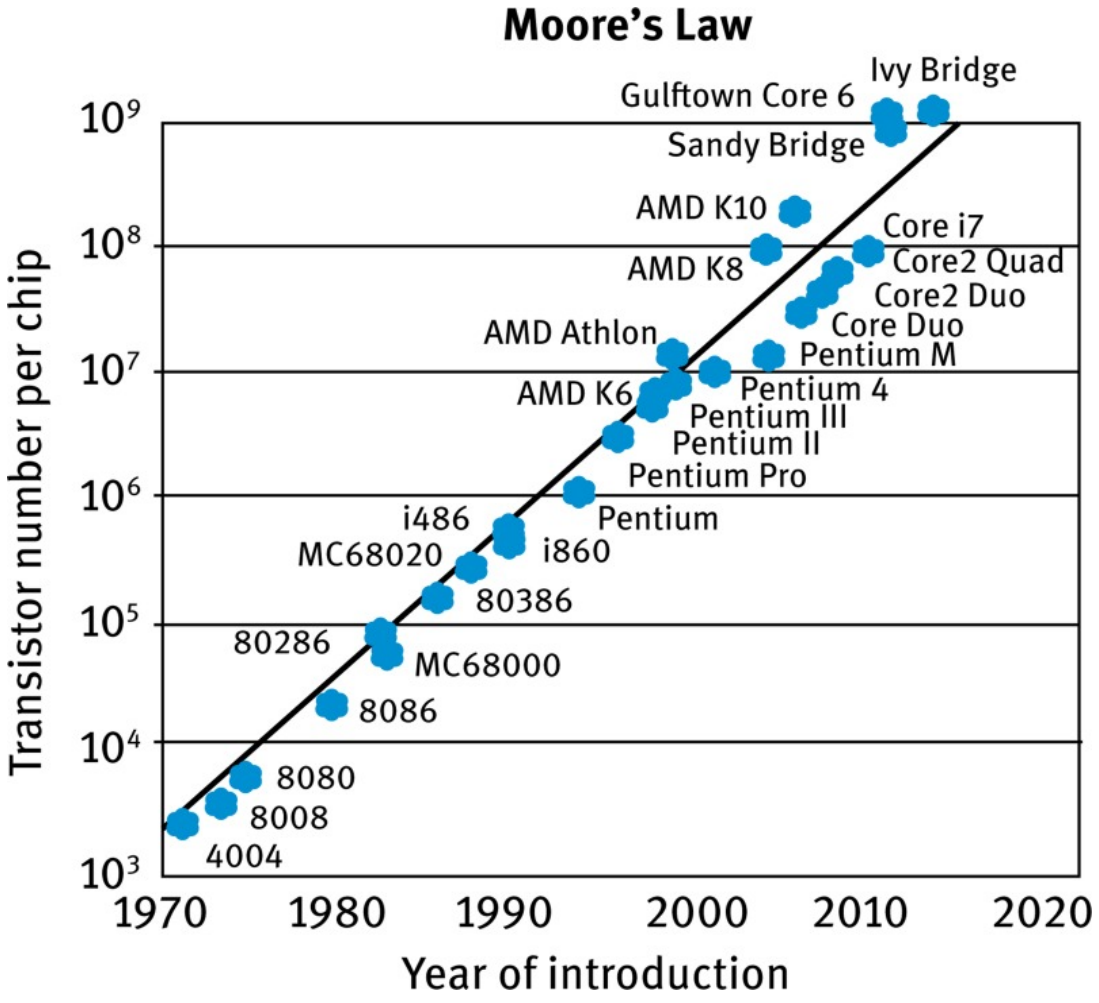
Why are we Here?

FUTURE VENTURES — 120 Years of Moore's Law



Source: Ray Kurzweil, Steve Jurvetson

Moore's Law



- A marching order established ~50 years ago
  - *"Let's continue to double the number of transistors every other year!"*
- First published as:
  - Moore, G.E.: *Cramming more components onto integrated circuits.* Electronics, 38(8), April 1965.
- Accepted by all partners:
  - Semiconductor manufacturers
  - Hardware integrators
  - Software companies
  - Us, the consumers

Dennard scaling law (downscaling)

new VLSI gen.

old VLSI gen.

$L' = L / 2$

$V' = V / 2$

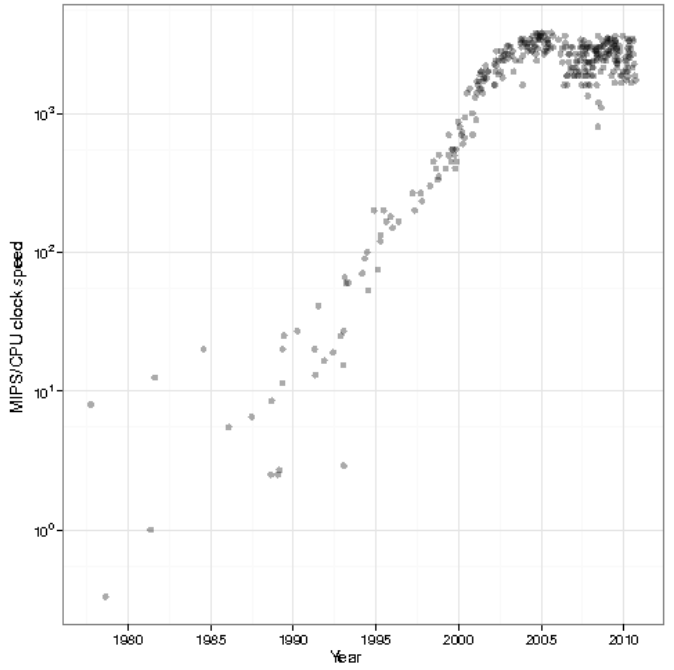
$F' = F * 2$

$D' = 1 / L^2 = 4D$

$P' = P$

do not hold anymore!

The core frequency and performance do not grow following the Moore's law any longer



$L' = L / 2$

$V' = \sim V$

$F' = \sim F * 2$

$D' = 1 / L^2 = 4 * D$

$P' = 4 * P$

The power crisis!

Increase the number of cores to maintain the architectures evolution on the Moore's law

Programming crisis!

- Now, power and/or heat generation are the limiting factors of the down-scaling

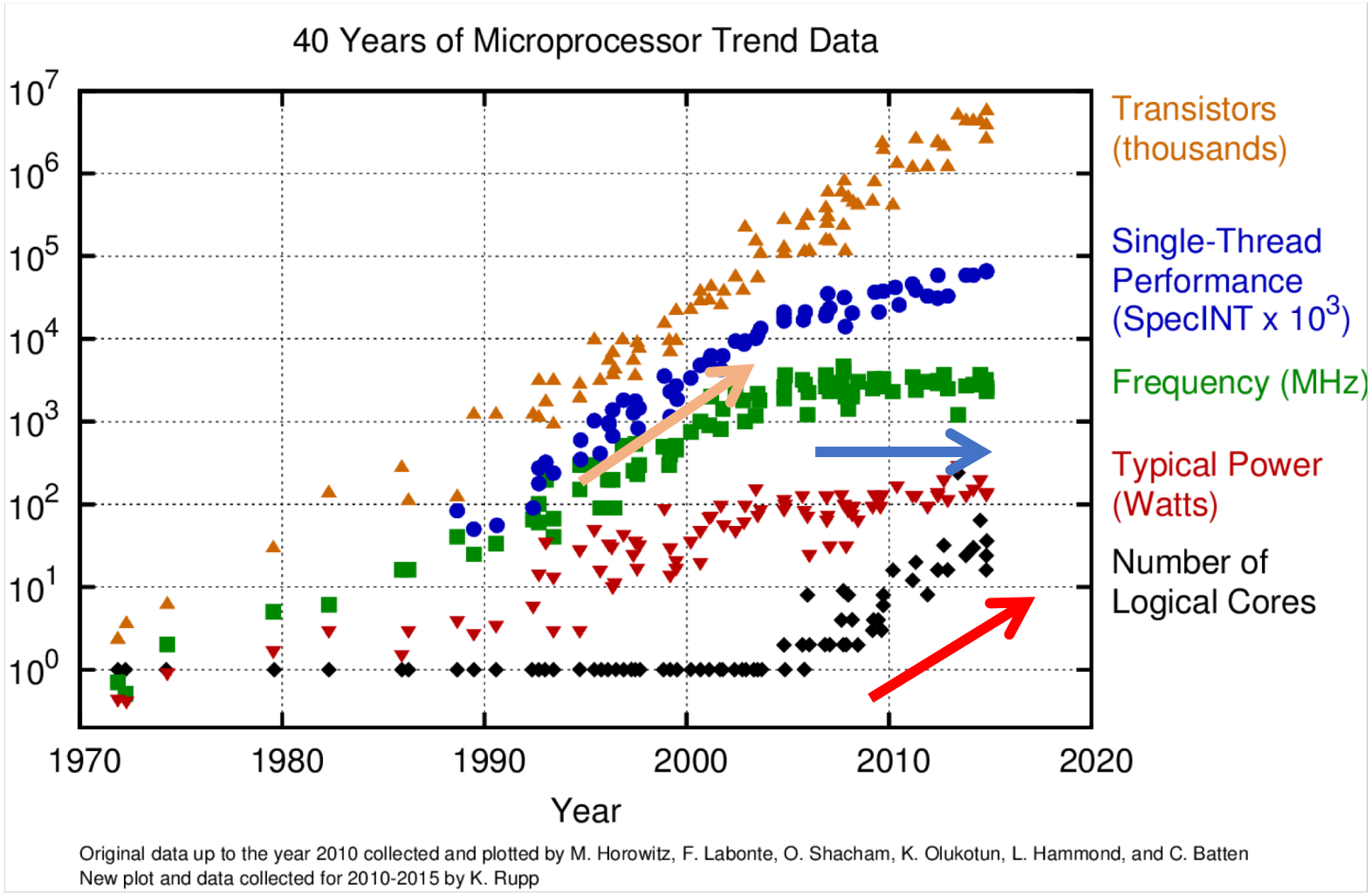
- Supply voltage reduction is becoming difficult, because Vth cannot be decreased any more, as described later.

- Growth rate in clock frequency and chip area becomes smaller.

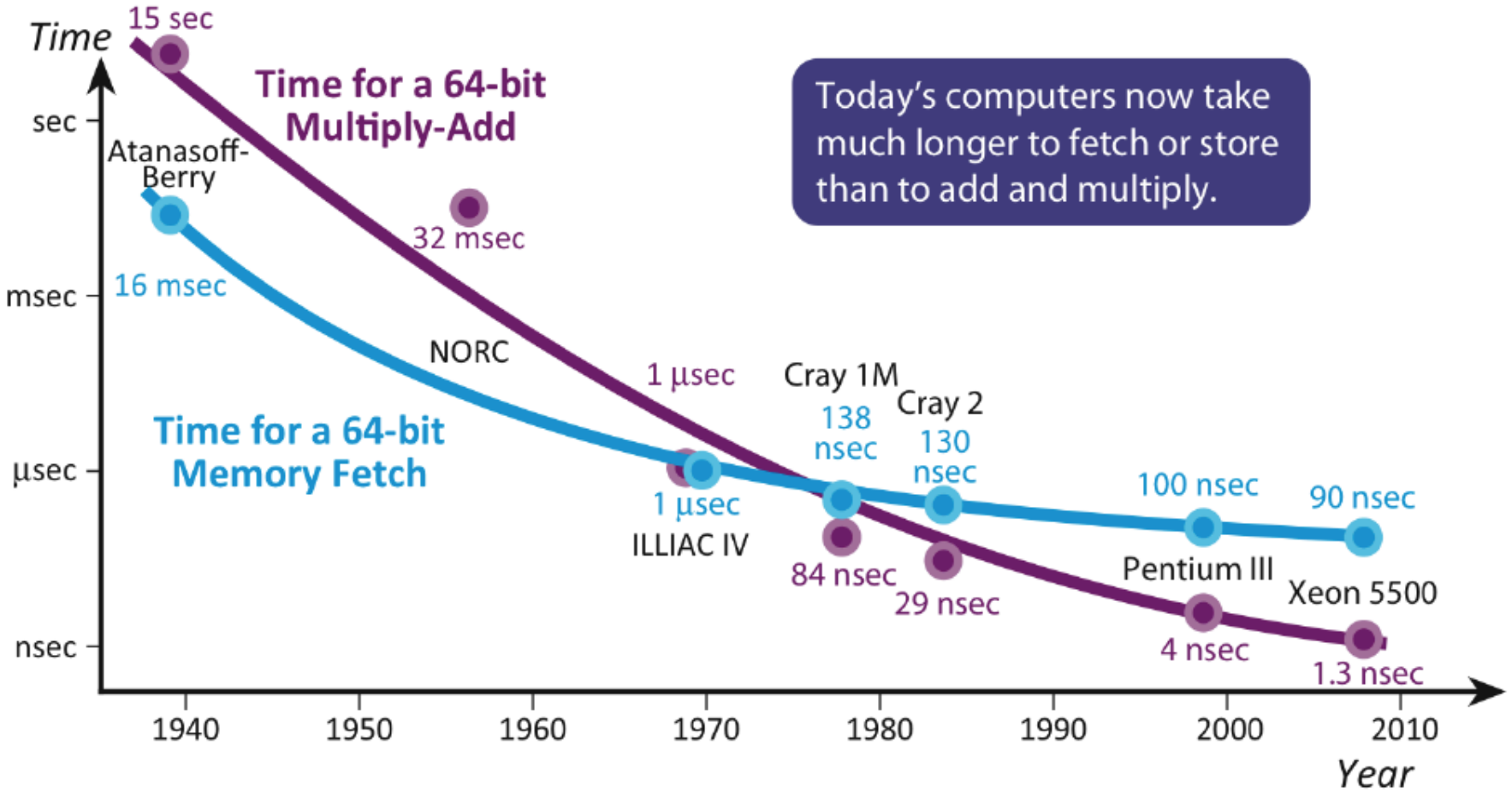
Credit: Carlo Cavazzoni, CINECA

Consequences

- The 7 "fat" years of frequency scaling:
  - The Pentium Pro in 1996: 150 MHz (12W)
  - The Pentium 4 in 2003: 3.8 GHz (~25X) (115W)
- Since then
  - Core 2 systems:
    - ~3 GHz
    - Multi-core
- Recent CERN purchase:
  - Intel Xeon E5-2630 v3
    - "only" 2.40 GHz (85W)
    - 8 core

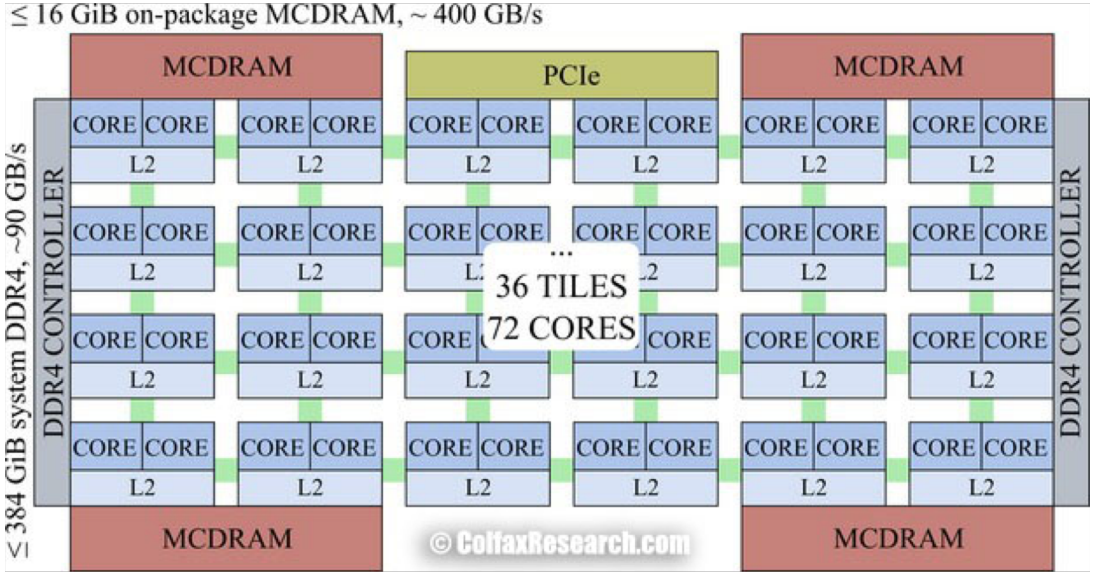


Memory Latency

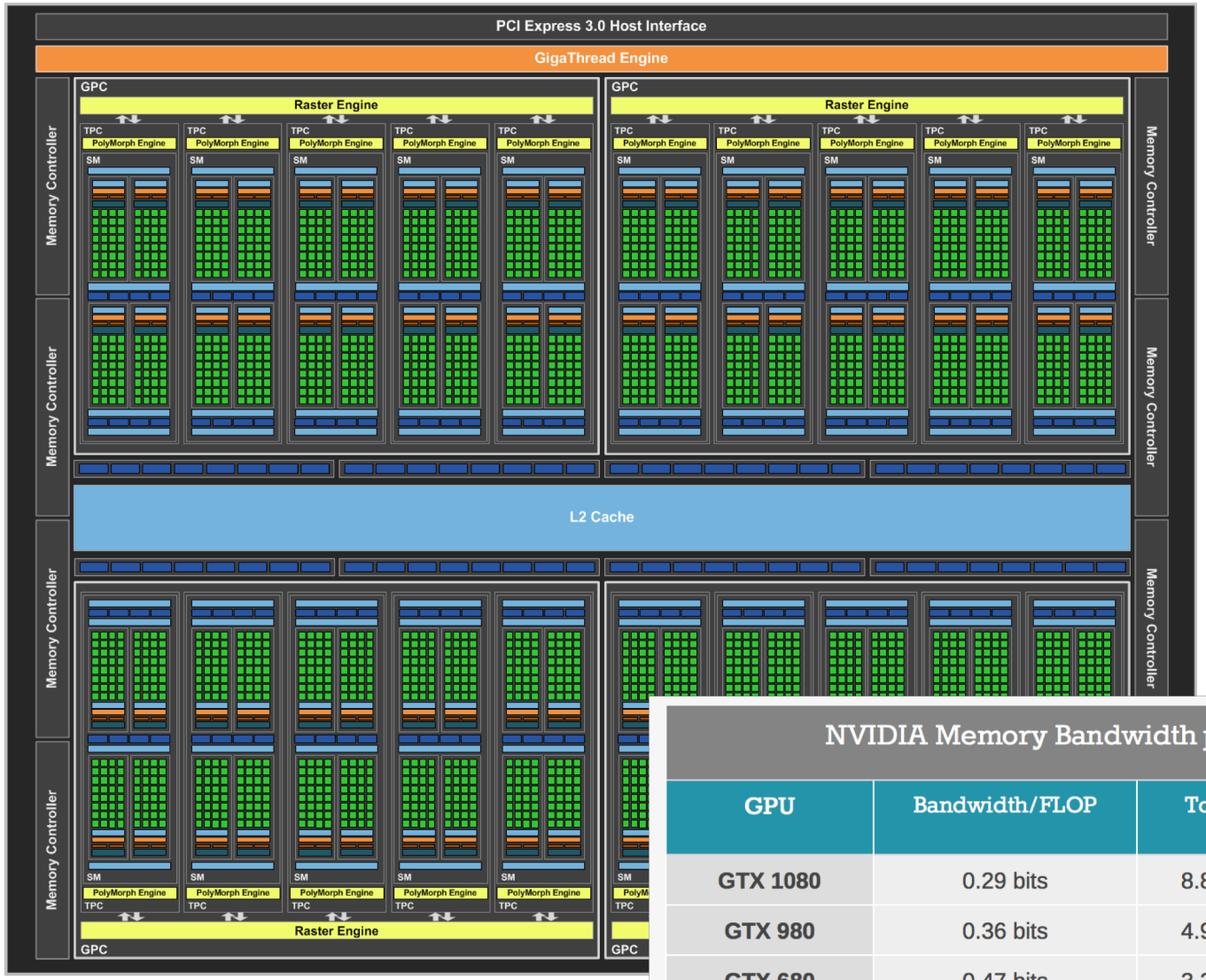


Simple, but illustrative example

- Intel KNL has ~64 cores @1.30GHz, 2FMA port (VPU) each, 4-way hardware threading, hardware vectors of size 8 (Double Precision), 16GB of fast memory:
- 3TFLOPS DP for 400GB/s = 0.5bit/flop-sp
  - 60 fp-ops = 1 fp-load



Streaming Multiprocessor Architecture



NVIDIA Pascal
32 CUDA core
x4 x5 x4 = 2560
Floating Point Units
@1.7GHz
8GB fast memory

Require 110 fp-ops
to hide
one memory access!

NVIDIA Memory Bandwidth per FLOP (In Bits)

GPU	Bandwidth/FLOP	Total FLOPs	Total Bandwidth
GTX 1080	0.29 bits	8.87 TFLOPs	320GB/sec
GTX 980	0.36 bits	4.98 TFLOPs	224GB/sec
GTX 680	0.47 bits	3.25 TFLOPs	192GB/sec
GTX 580	0.97 bits	1.58 TFLOPs	192GB/sec

credit AnandTech

Do More with Less

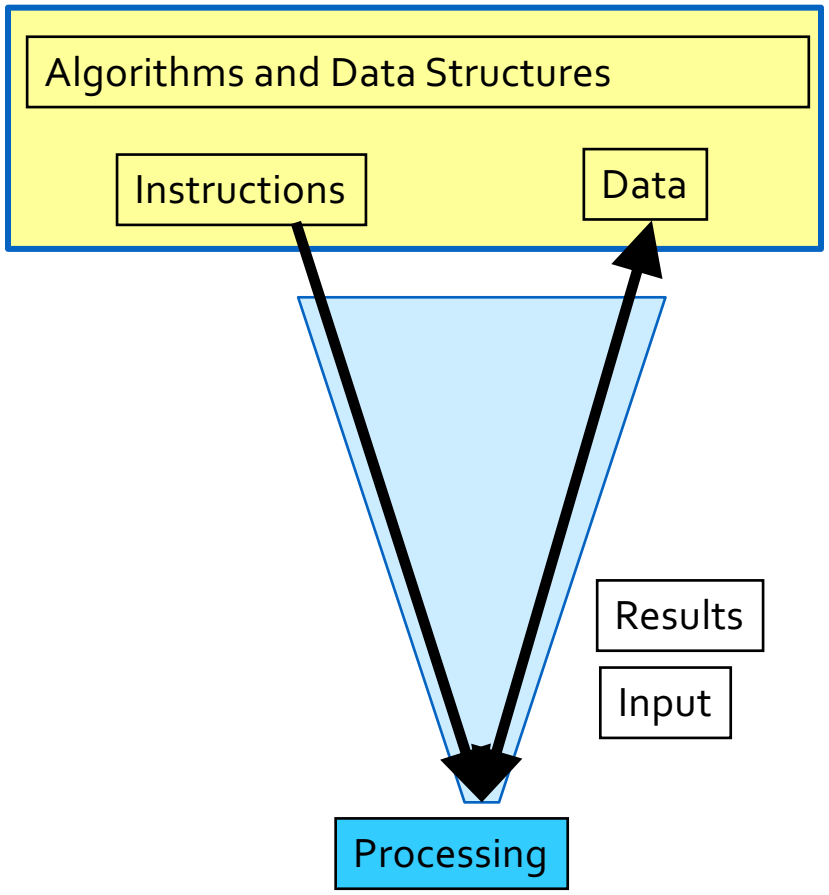
- Improving throughput and/or latency requires exploiting optimal massive parallelization at all levels
- Speeding up algorithms will not pay up if memory access is not reduced

Computing Architecture

Von Neumann architecture

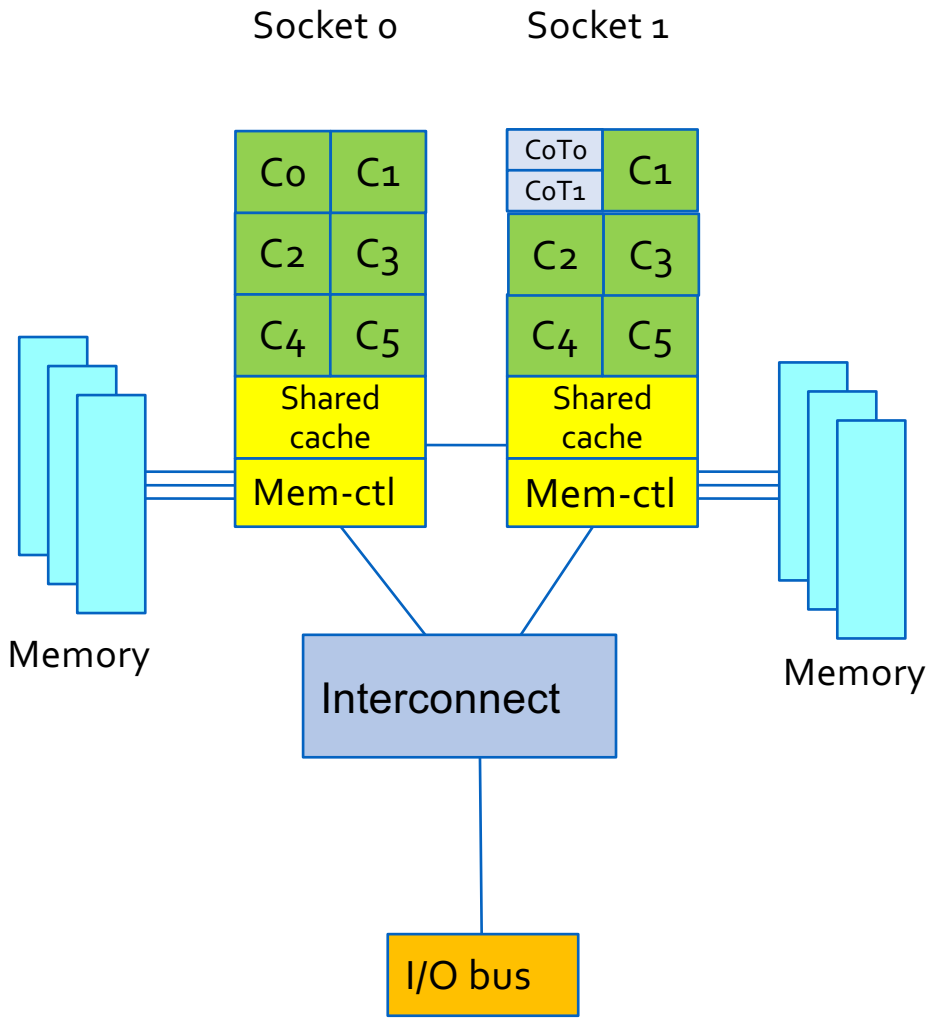
- From Wikipedia:
  - The von Neumann architecture is a computer design model that uses a processing unit and a single separate storage structure to hold both instructions and data.

- It can be viewed as an entity into which one streams instructions and data in order to produce results



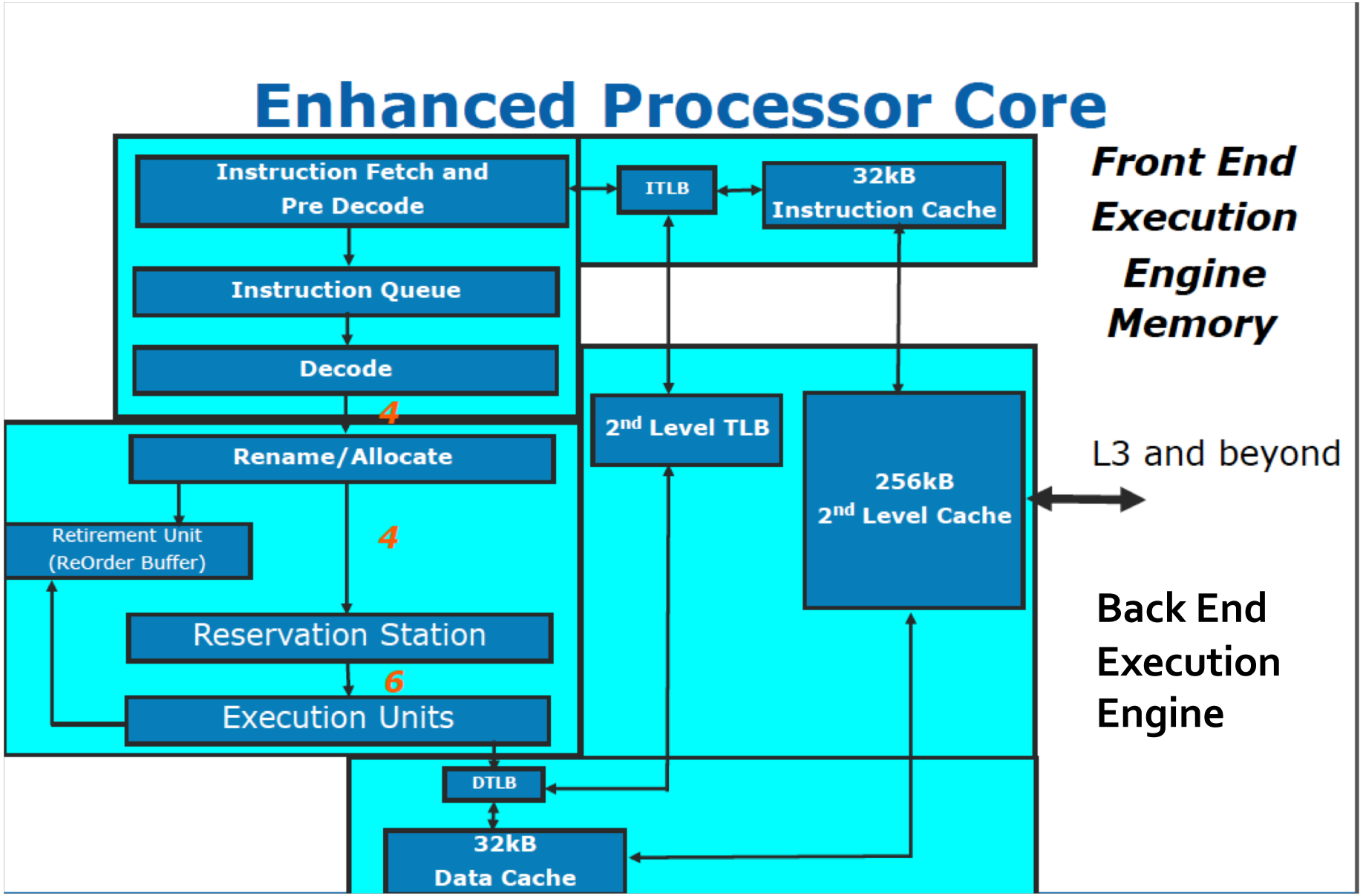
Simple server diagram

- Multiple components which interact during the execution of a program:
  - Processors/cores
    - w/private caches
      - I-cache, D-cache
  - Shared caches
    - Instructions and Data
  - Memory controllers
  - Memory (non-uniform)
  - I/O subsystem
    - Network attachment
    - Disk subsystem



Intel Nehalem

Single Core Architecture


Enhanced Processor Core
Instruction Fetch and Pre Decode
ITLB
32kB Instruction Cache
Instruction Queue
Decode
4
Rename/Allocate
2nd Level TLB
256kB 2nd Level Cache
Retirement Unit (ReOrder Buffer)
4
Reservation Station
6
Execution Units
DTLB
32kB Data Cache
Front End
Execution Engine
Memory
L3 and beyond
Back End Execution Engine

Architecture: front end

Feeds "decoded" instructions to the scheduler

Affected by instruction non-locality (iCache-miss, iTLB misses) and misspredicted branches

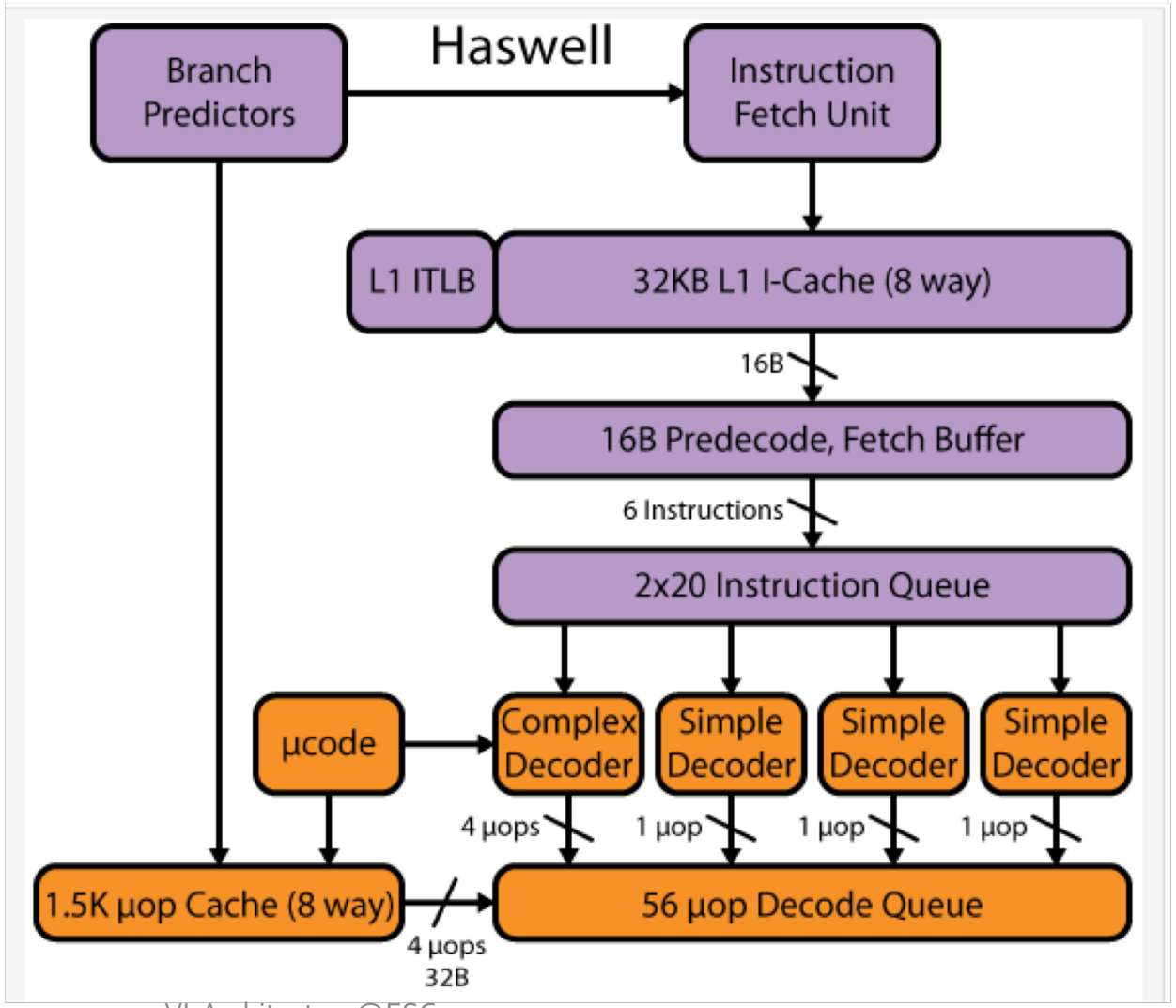
Main metrics:

L1-icache-load-misses (icache.ifdata_stall)

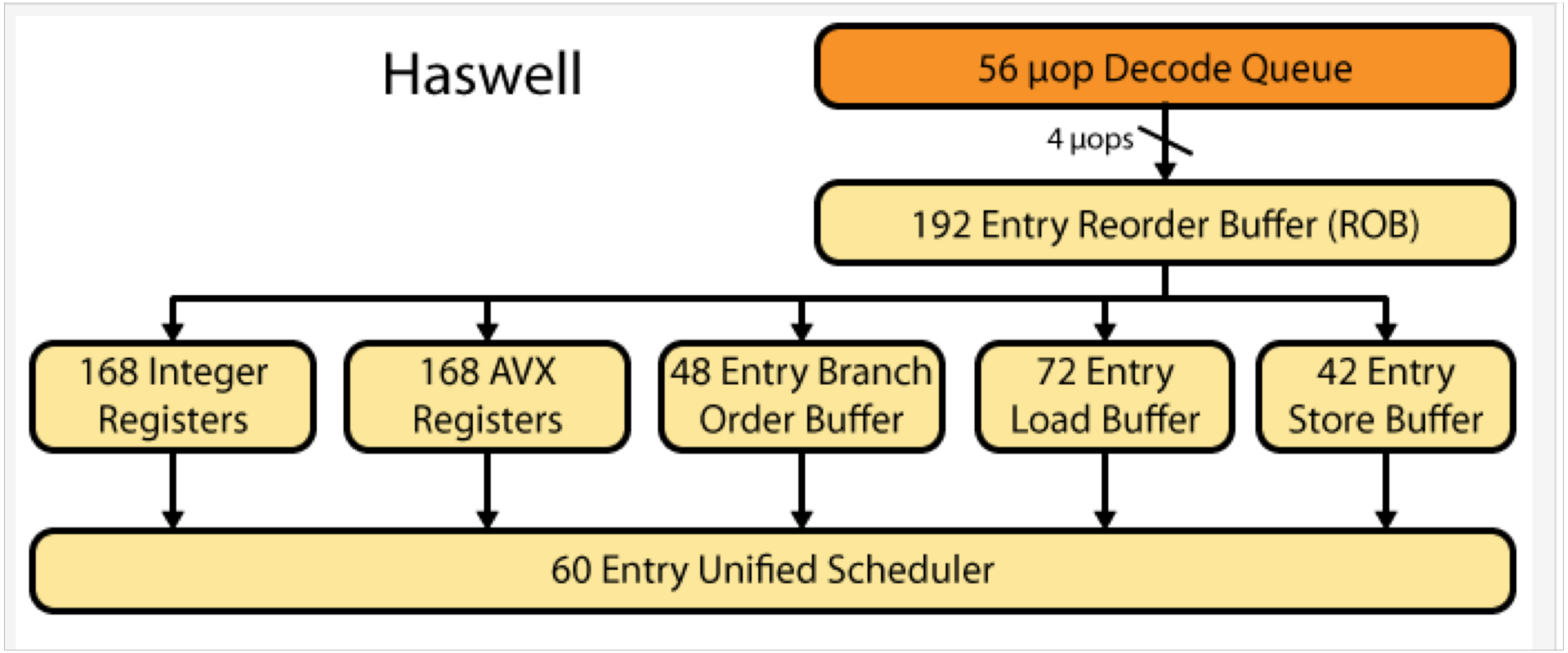
Cycles where a code fetch is stalled due to L1 instruction cache miss.

branch-misses (br_misp_retired.all_branches)

This event counts all mispredicted branch instructions retired.



Architecture: Out of order scheduler



Main metric:
rs_events.empty_cycles
This event counts cycles during which the reservation station (RS) is empty
RS == Unified scheduler

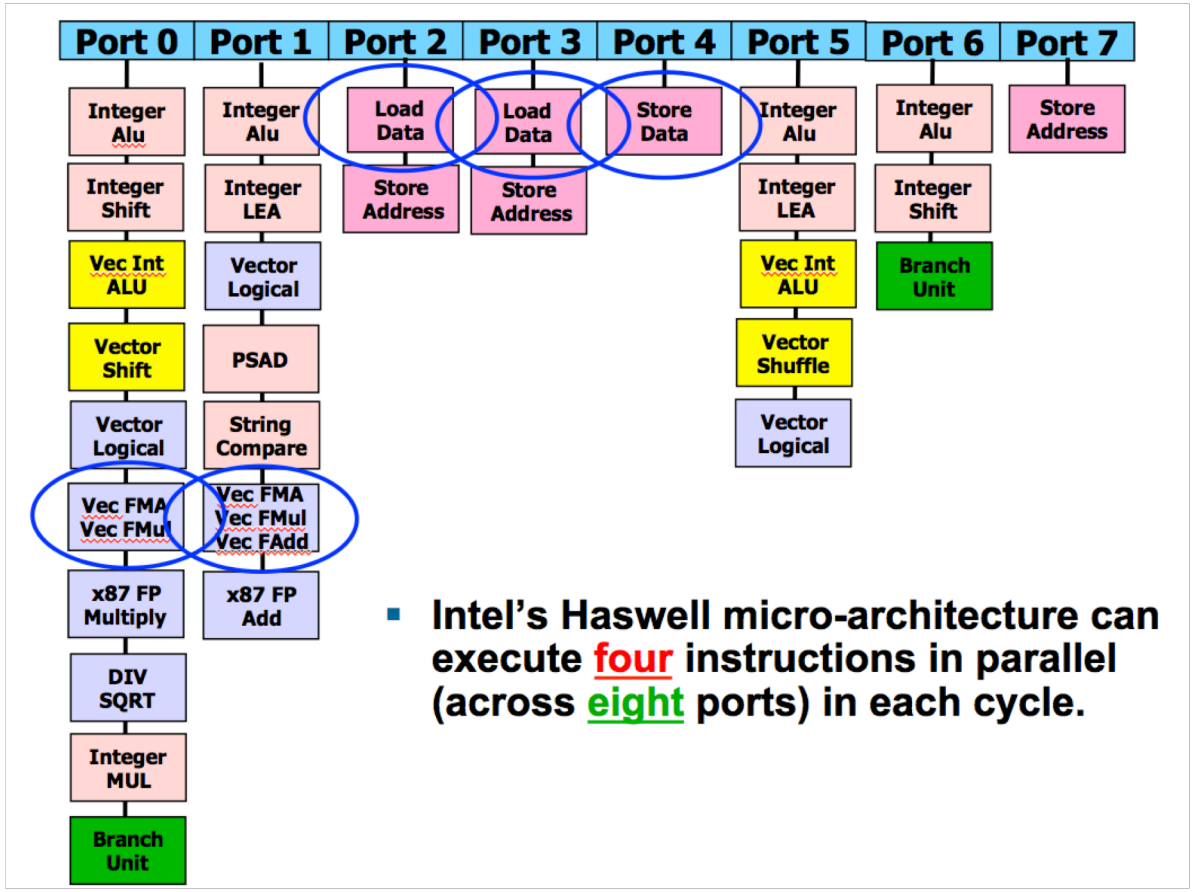
Out-of-order (OOO) scheduling

- Most modern processors use OOO scheduling
  - This means that they will speculatively execute instructions ahead of time (Xeon: inside a "window" of ~150 instructions)
  - In certain cases the results of such executed instructions must be discarded
- At the end, there is a difference between "executed instructions" and "retired instructions"
  - One typical reason for this is mispredicted branches
- Potential problem with OOO:
  - A lot of extra energy is needed!
- Interestingly: ARM has two designs:
  - A53 (low power, in-order), A57 (high power, OOO)

Architecture: Backend

Computational engine
Affected by

- instruction dependency
  - instruction parallelism
  - pipelining
- Memory access
- Latency of "heavy instructions"
  - div sqrt
- Vectorization



Main Metrics:

uops_executed.stall_cycles

This event counts cycles during which no uops were dispatched from the Reservation Station (RS)

uops_executed.thread

Number of uops to be executed each cycle.

cycle_activity.stalls_mem_any

Execution stalls while memory subsystem has an outstanding load.

arith.divider_active

Cycles when divide unit is busy executing divide or square root operations. Accounts for integer and floating-point operations.

Real-life latencies

- Most integer/logic instructions have a one-cycle execution latency:
  - For example (on an Intel Xeon processor)
    - ADD, AND, SHL (shift left), ROR (rotate right)
  - Amongst the exceptions:
    - IMUL (integer multiply): 3
    - IDIV (integer divide): 13 – 23
- Floating-point latencies are typically multi-cycle
  - FADD (3), FMUL (5)
    - Same for both x87 and SIMD double-precision variants
  - Exception: FABS (absolute value): 1
  - Many-cycle, no pipepine : FDIV (20), FSQRT (27)
  - Other math functions: even more

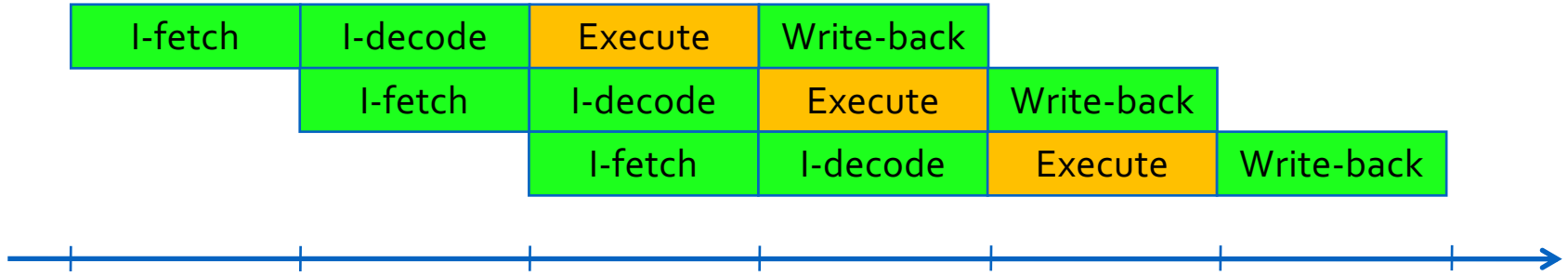
As of Haswell:
FMA (5 cycles)
As of Skylake:
SIMD ADD, MUL,FMA: 4 cycles

Latencies in the Core micro-architecture (Intel Manual No. 248966-026 or later).
AMD processor latencies are similar.

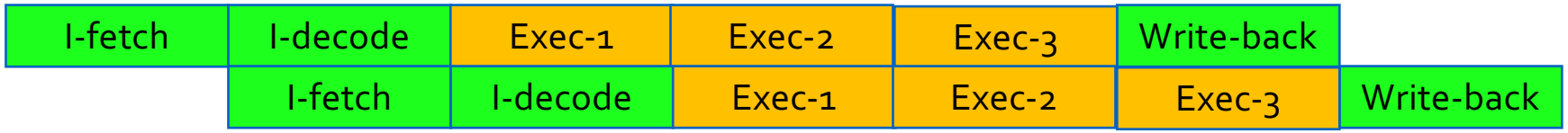
http://www.agner.org/optimize/instruction_tables.pdf

Instruction pipelining

- Instructions are broken up into stages.
  - With a one-cycle execution latency (simplified):



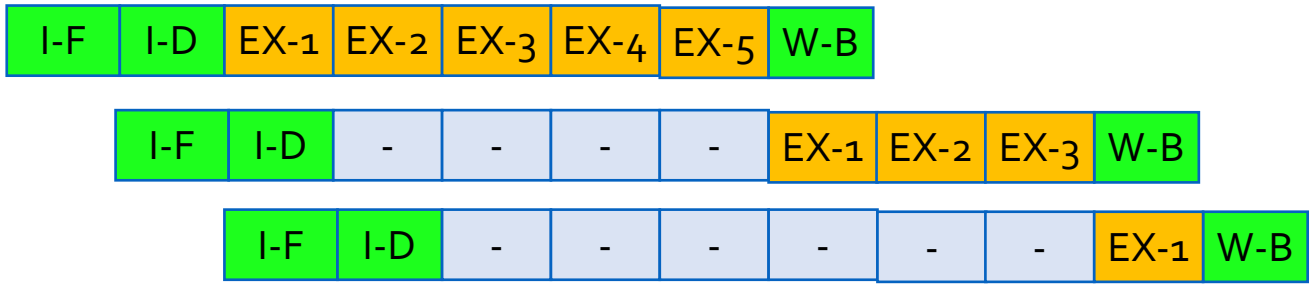
  - With a three-cycle execution latency:



Latencies and serial code (1)

- In serial programs, we typically pay the penalty of a multi-cycle latency during execution:
  - In this example:
    - Statement 2 cannot be started before statement 1 has finished
    - Statement 3 cannot be started before statement 2 has finished

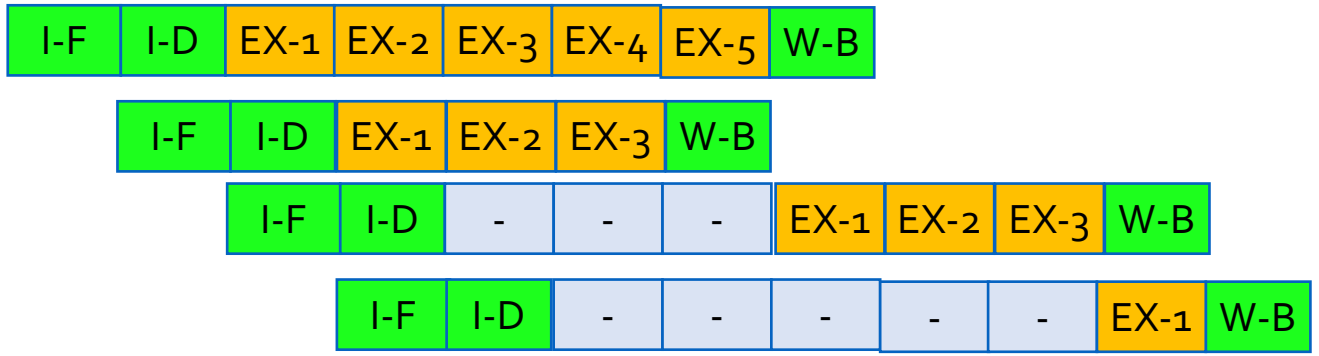
```
double a, b, c, d, e, f;

b = 2.0; c = 3.0; e = 4.0;

a = b * c;  // Statement 1

d = a + e;  // Statement 2

f = fabs(d);   // Statement 3
```



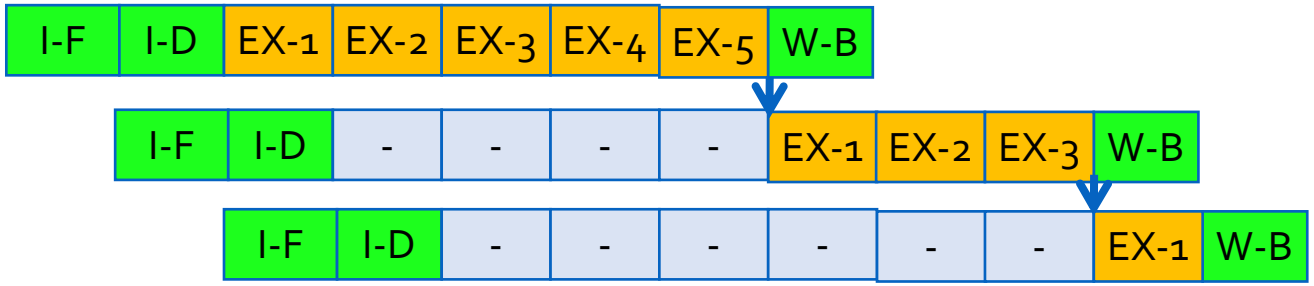
Latencies and serial code (1)

- In serial programs, we typically pay the penalty of a multi-cycle latency during execution:
  - In this example:
    - Statement 2 cannot be started before statement 1 has finished
    - Statement 3 cannot be started before statement 2 has finished

```
double a, b, c, d, e, f;

b = 2.0; c = 3.0; e = 4.0;

a = b * c;  // Statement 1

 f = b + e;
d = a + f;  // Statement 2


f = fabs(d);   // Statement 3
```



Latencies and serial code (2)



- Observations:
  - Even if the processor can fetch and decode a new instruction every cycle, it must wait for the previous result to be made available
    - Fortunately, the result takes a 'bypass', so that the write-back stage does not cause even further delays
  - The result: CPI is equal to 3
    - 9 execution cycles are needed for 3 instructions!
- A good way to hide latency is to [get the compiler to] unroll (vector) loops !

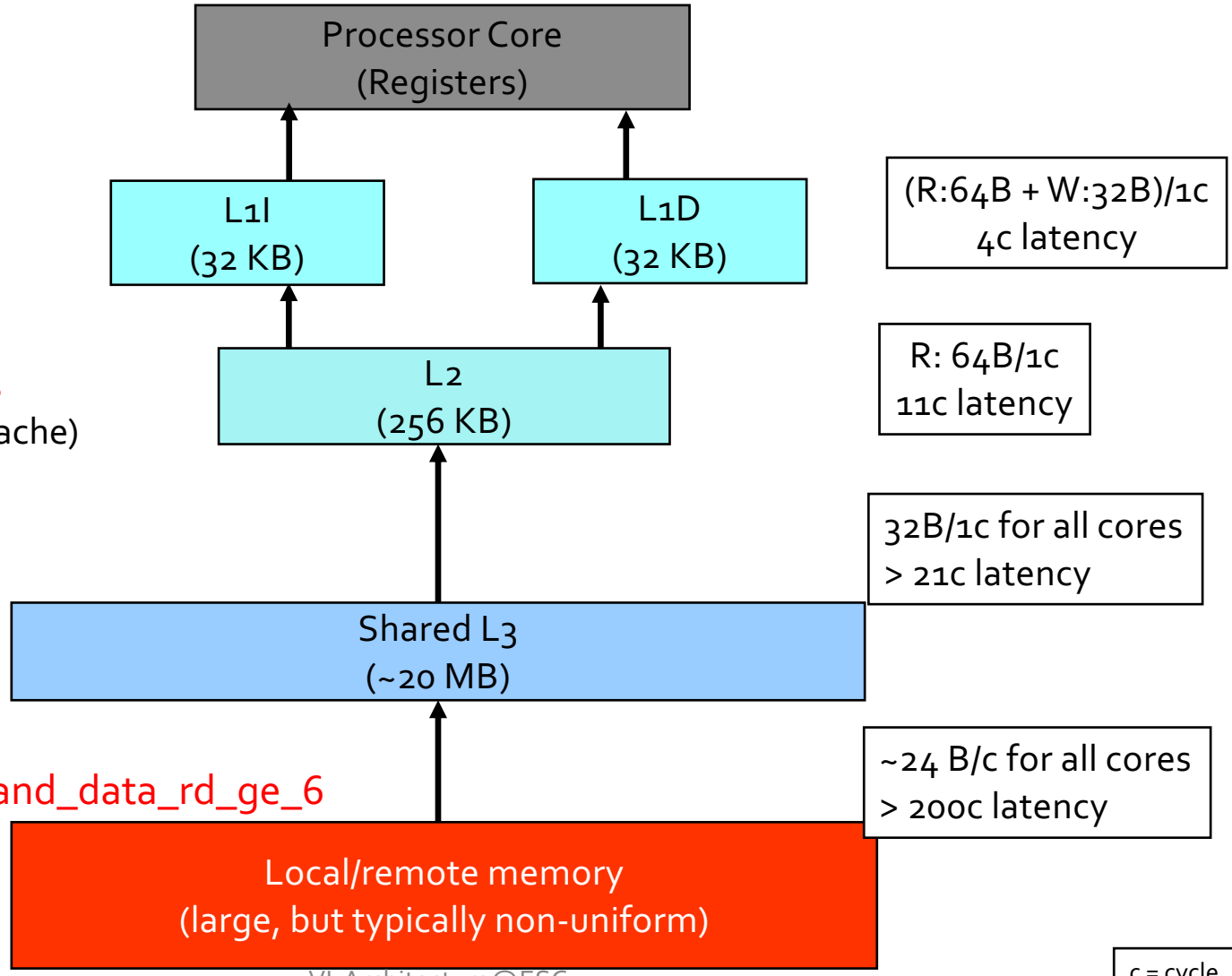
Memory architecture

Cache/Memory Hierarchy

- From CPU to main memory on a recent Haswell processor
  - With multicore, memory bandwidth is shared between cores in the same processor (socket)

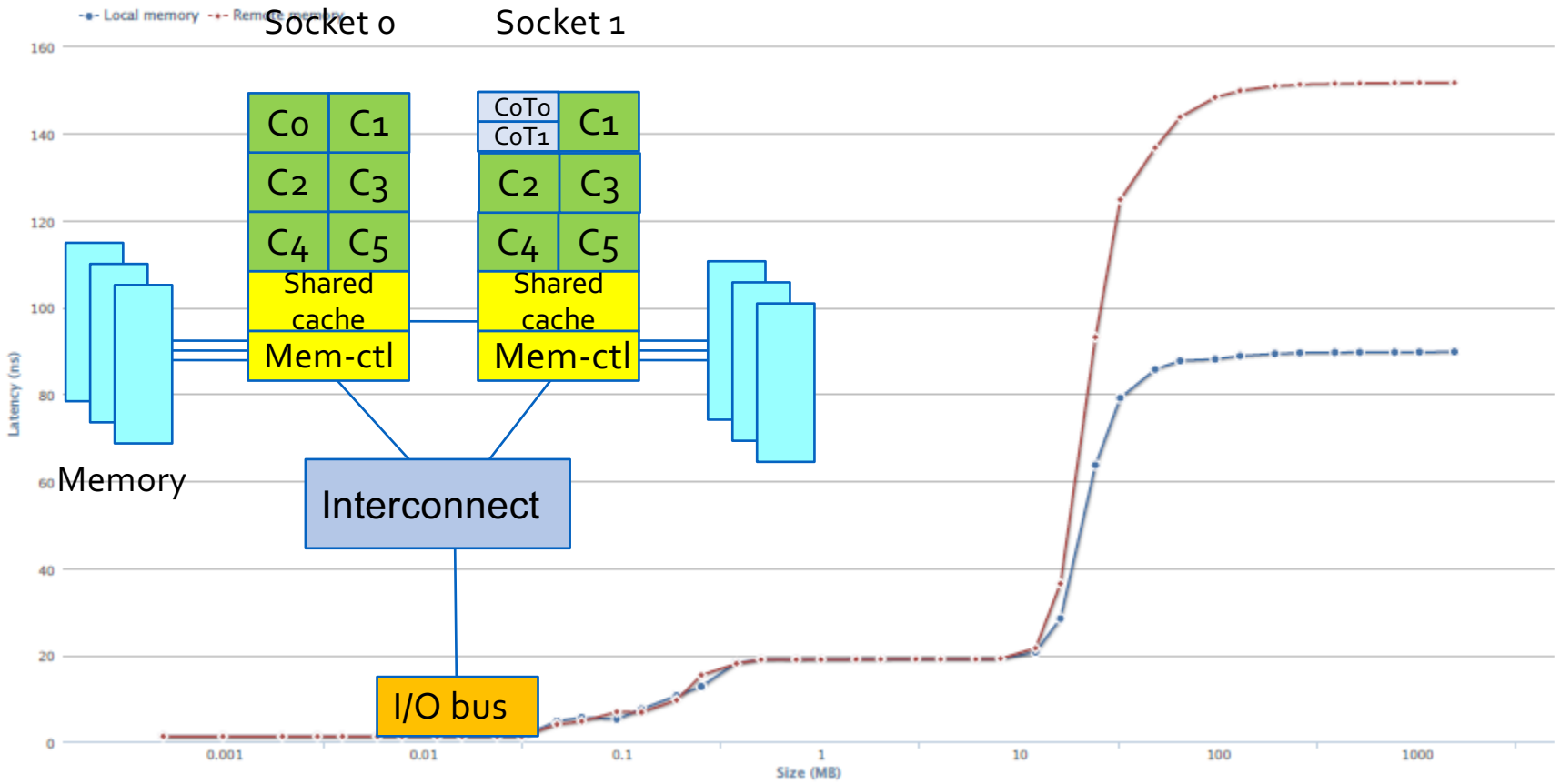
Main metrics:
L1-dcache-loads, L1-dcache-load-misses
LLC-loads, LLC-load-misses (LastLevelCache)

mem_load_retired.l1_hit
mem_load_retired.l2_hit
mem_load_retired.l3_hit
mem_load_retired.l3_miss
offcore_requests.all_requests
offcore_requests_outstanding.demand_data_rd_ge_6
cycle_activity.stalls_mem_any



Latency Measurements (example)

- Memory Latency on Sandy Bridge-EP 2690 (dual socket)
  - 90 ns (local) versus 150 ns (remote)



Recent architectures

The numbers we looked at were "Random load latency stride=16 Bytes" (LMBench).

Mem Hierarchy	IBM POWER8	Intel Broadwell Xeon E5-2640v4 DDR4-2133	Intel Broadwell Xeon E5-2699v4 DDR4-2400
L1 Cache (cycles)	3	4	4
L2 Cache (cycles)	13	12-15	12-15
L3 Cache 4-8 MB(cycles)	27-28 (8 ns)	49-50	50
16 MB (ns)	55 ns	26 ns	21 ns
32-64 MB (ns)	55-57 ns	75-92 ns	80-96 ns
Memory 96-128 MB (ns)	67-74 ns	90-91 ns	96 ns
Memory 384-512 MB (ns)	89-91 ns	91-93 ns	95 ns

Source AnandTech

Cache lines (1)

- When a data element or an instruction is requested by the processor, a cache line is ALWAYS moved (as the minimum quantity), usually to Level-1

- A cache line is a contiguous section of memory, typically 64B in size (8 * double) and 64B aligned
  - A 32KB Level-1 cache can hold 512 lines
- When cache lines have to be moved come from memory
  - Latency is long (>200 cycles)
    - It is even longer if the memory is remote
  - Memory controller stays busy (~8 cycles)

Cache lines (2)

- Good utilisation is vital
  - When only one element (4B or 8B) element is used inside the cache line:
    - A lot of bandwidth is wasted!

- Multidimensional C arrays should be accessed with the last index changing fastest:

```
for (auto & a : v)
    a->x += increment;
```

- Pointer chasing (in linked lists) can easily lead to "cache thrashing" (too much memory traffic)

Cache lines (3)

- Prefetching:
  - Fetch a cache line before it is requested
    - Hiding latency
  - Normally done by the hardware
    - Especially if processor executes Out-of-order
  - Also done by software instructions
    - Especially when In-order (IA-64, Xeon Phi, etc.)
- Locality is vital:
  - Spatial locality – Use all elements in the line
  - Temporal locality – Complete the execution whilst the elements are certain to be in the cache

Programming the memory hierarchy is an art in itself.

Further reading:

- "Designing and Building Parallel Programs", I. Foster, Addison-Wesley, 1995
- "Foundations of Multithreaded, Parallel and Distributed Programming", G.R. Andrews, Addison-Wesley, 1999
- "Computer Architecture: A Quantitative Approach", J. Hennessy and D. Patterson, 3rd ed., Morgan Kaufmann, 2002
- "Patterns for Parallel Programming", T.G. Mattson, Addison Wesley, 2004
- "Principles of Concurrent and Distributed Programming", M. Ben-Ari, 2nd edition, Addison Wesley, 2006
- "The Software Vectorization Handbook", A.J.C. Bik, Intel Press, 2006
- "The Software Optimization Cookbook", R. Gerber, A.J.C. Bik, K.B. Smith and X. Tian; Intel Press, 2nd edition, 2006
- "Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism", J. Reinders, O'Reilly, 1st edition, 2007
- "Inside the Machine", J. Stokes, Ars Technica Library, 2007