Opportunities and choice in a new vector era

How parallel computing technologies influence physics software frameworks and their design

ACAT 2013, Beijing, China

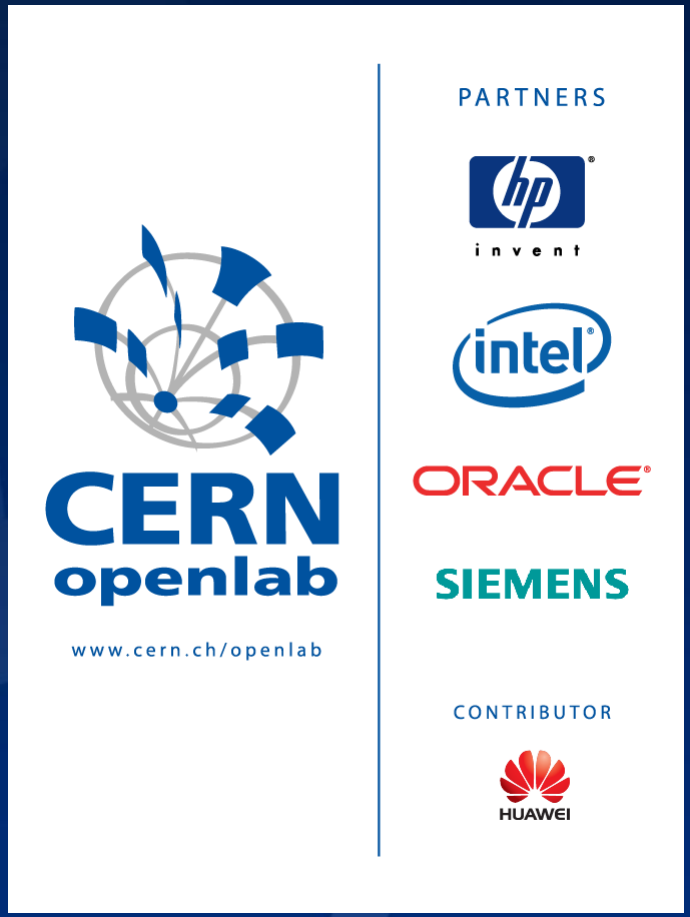
May 18th 2013

Andrzej Nowak, CERN openlab

CERN openlab

CERN openlab

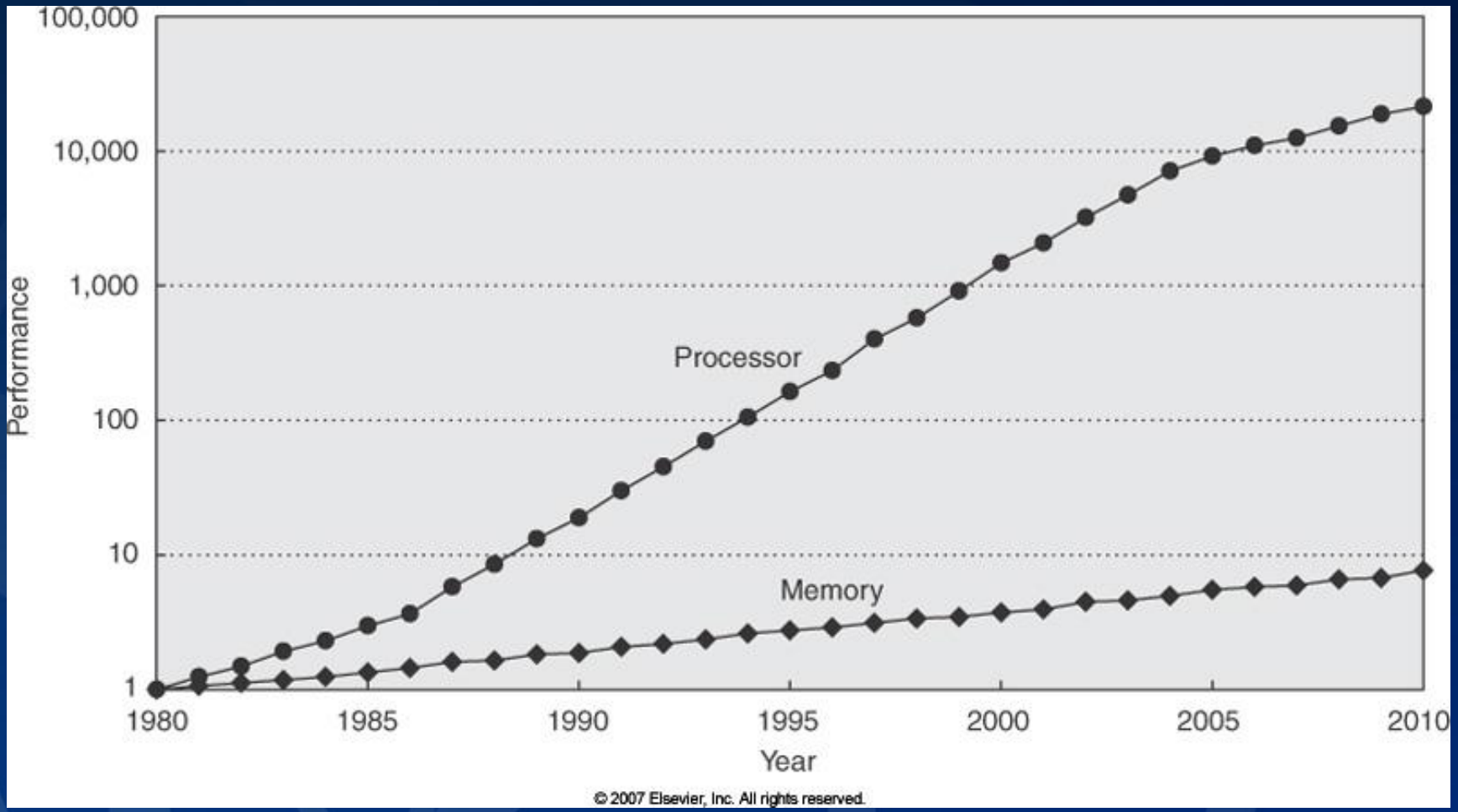
- CERN openlab is a framework for evaluating and integrating cutting-edge IT technologies or services in partnership with industry
- The Platform Competence Center (PCC) has worked closely with Intel for the past decade and focuses on:
  - many-core scalability
  - performance tuning and optimization
  - benchmarking and thermal optimization
  - teaching



The Platform Competence Center

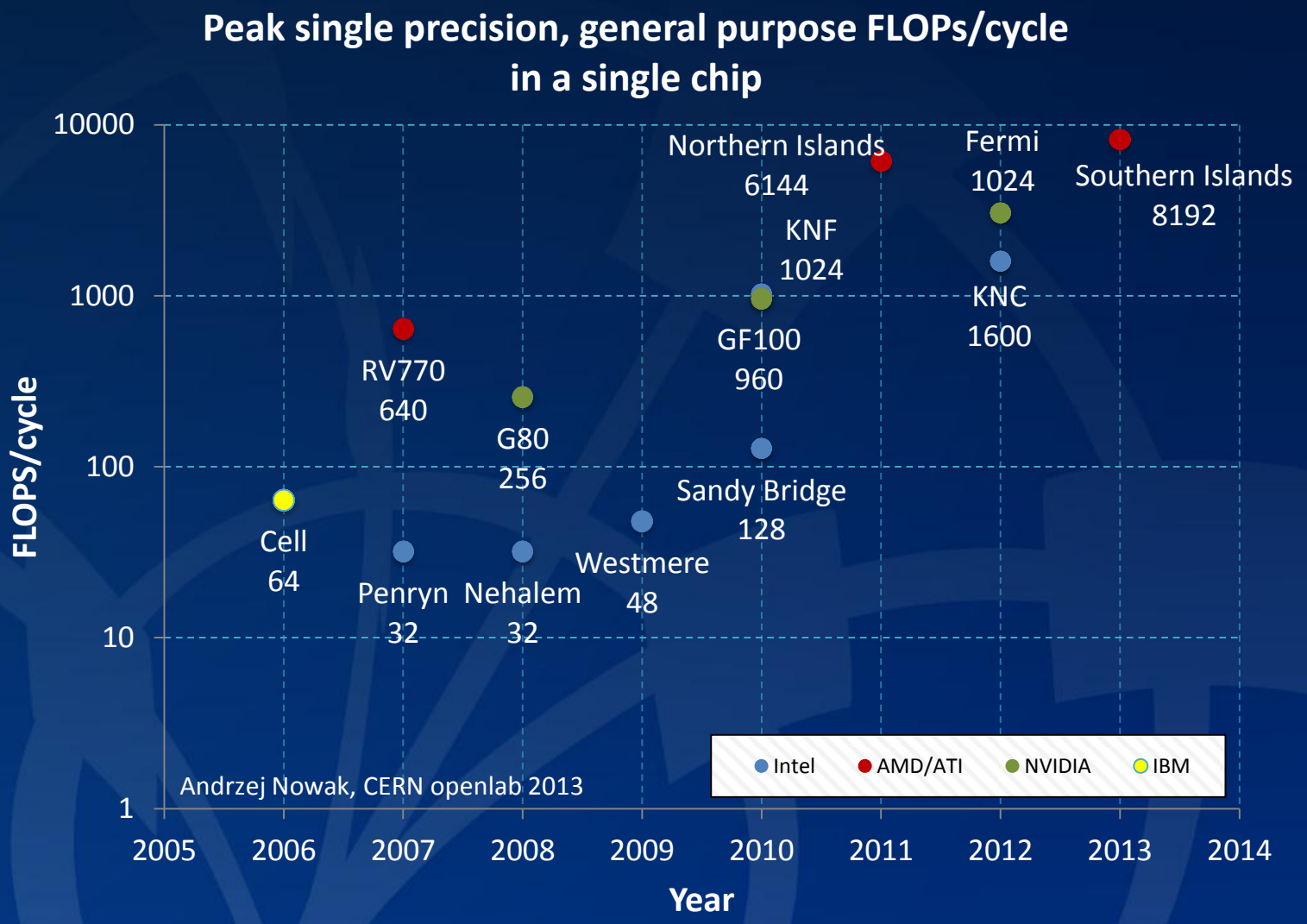
Focus on efficient computing

Motivation (the past)



Modelled after ”Computer Architecture”: Hennessy, John L.; Patterson, David A.

Motivation (the present)



Motivation (the future?)

Operation	Energy cost
L1 access	2 pJ
L2 access	150 pJ
RAM	2 000 pJ
FLOP	?

Where the (computing) space is headed

- There is a perception of a slowing client trend
  - Less piggyback opportunities for HPC/HTC
- New instruction sets make it a worse idea to keep using intrinsics
- Growing number of cores
- 4-socket systems gaining popularity
- Bias towards accelerators and vectors
- Warmer welcome for new architectures

Corollary

Raw platform performance is expanding in multiple dimensions simultaneously

Foundations of cost effective computing

- **The PC server with bells and whistles**
  - Variants with low power consumption
  - Variants with modern vectors (256 bits)
  - Most have 2 CPUS, some have 1 or 4
  - Fixed amount of single threaded jobs per core
  - Fixed amount of memory per job
- **Quite a homogeneous environment**

- **Does this model scale and hold in the future?**

Where we still are
(with large frameworks)

	SIMD	ILP	HW THREADS	CORES	SOCKETS
THEORY	4	4	1.35	8	4
OPTIMIZED	2.5	1.43	1.25	8	2
HEP	1	0.80	1	6	2

	SIMD	ILP	HW THREADS	CORES	SOCKETS
THEORY	4	16	21.6	172.8	691.2
OPTIMIZED	2.5	3.57	4.46	35.71	71.43
HEP	1	0.80	0.80	4.80	9.60

Not all HEP code is here – but a lot is

Using a low single digit percentage of raw machine power available today

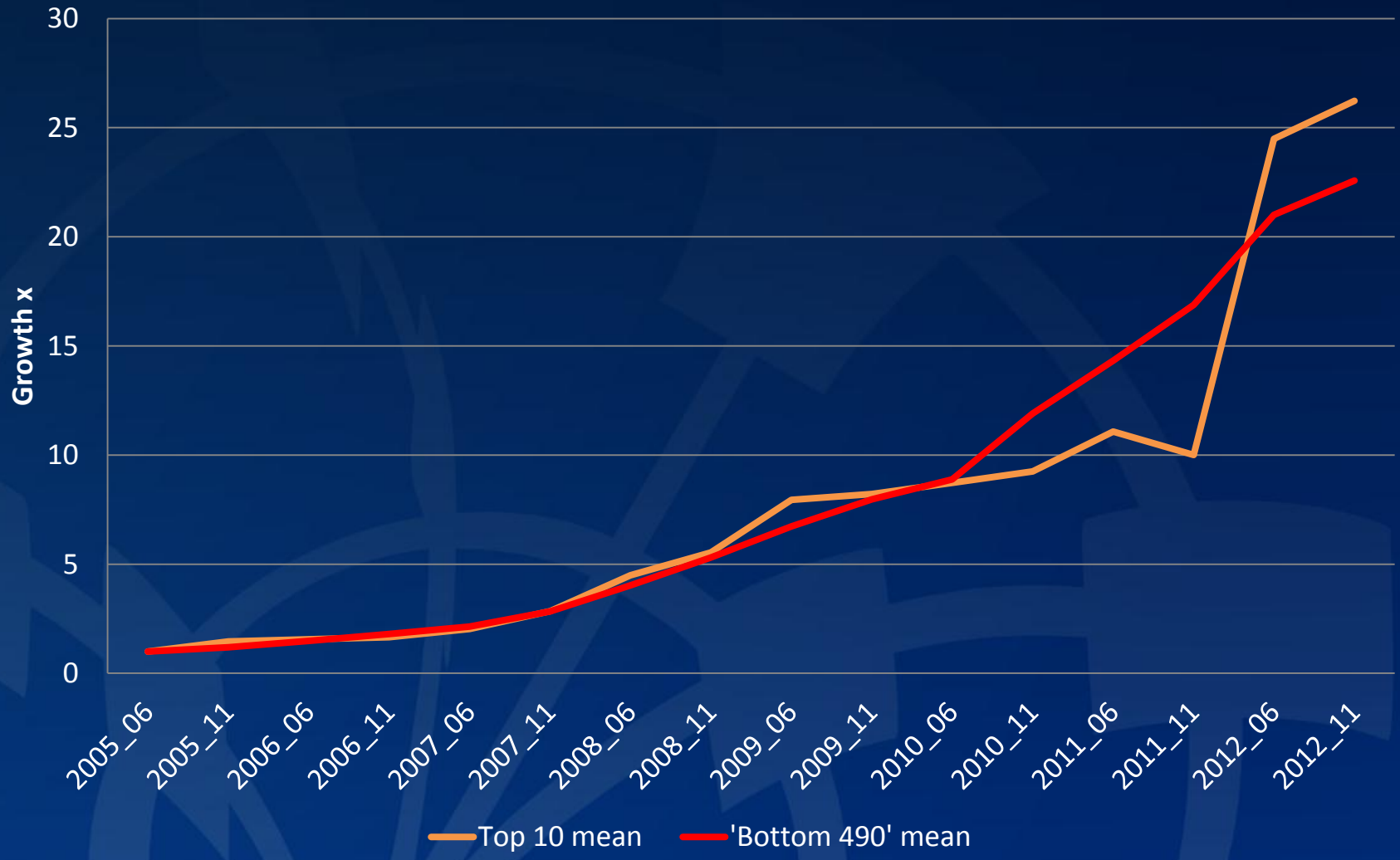
_%

Write your percentage here

Corollary

Need to program for tomorrow's hardware today

Top500 CPU core count growth



Modeled after Lehto, Manninen, von Alfthan

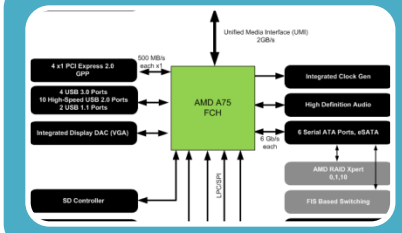
Heterogeneity - scale and scenarios

Cluster level

- Non-homogeneous nodes
- Large scale, expensive interconnect

Node level

- Non-homogeneous components of a node
- Standard platform interconnect



Chip level

- Non-homogeneous components in a package/chip
- On-chip interconnect or standard bus

Heterogeneity - scale and scenarios

Native mode

workload runs entirely on a co-processor system (e.g. networked via PCIe)

Offload

Co-processor as an accelerator where host gets weak

Balanced

Co-processor and host work together

Cluster

application distributed across multiple cards (possibly including host)

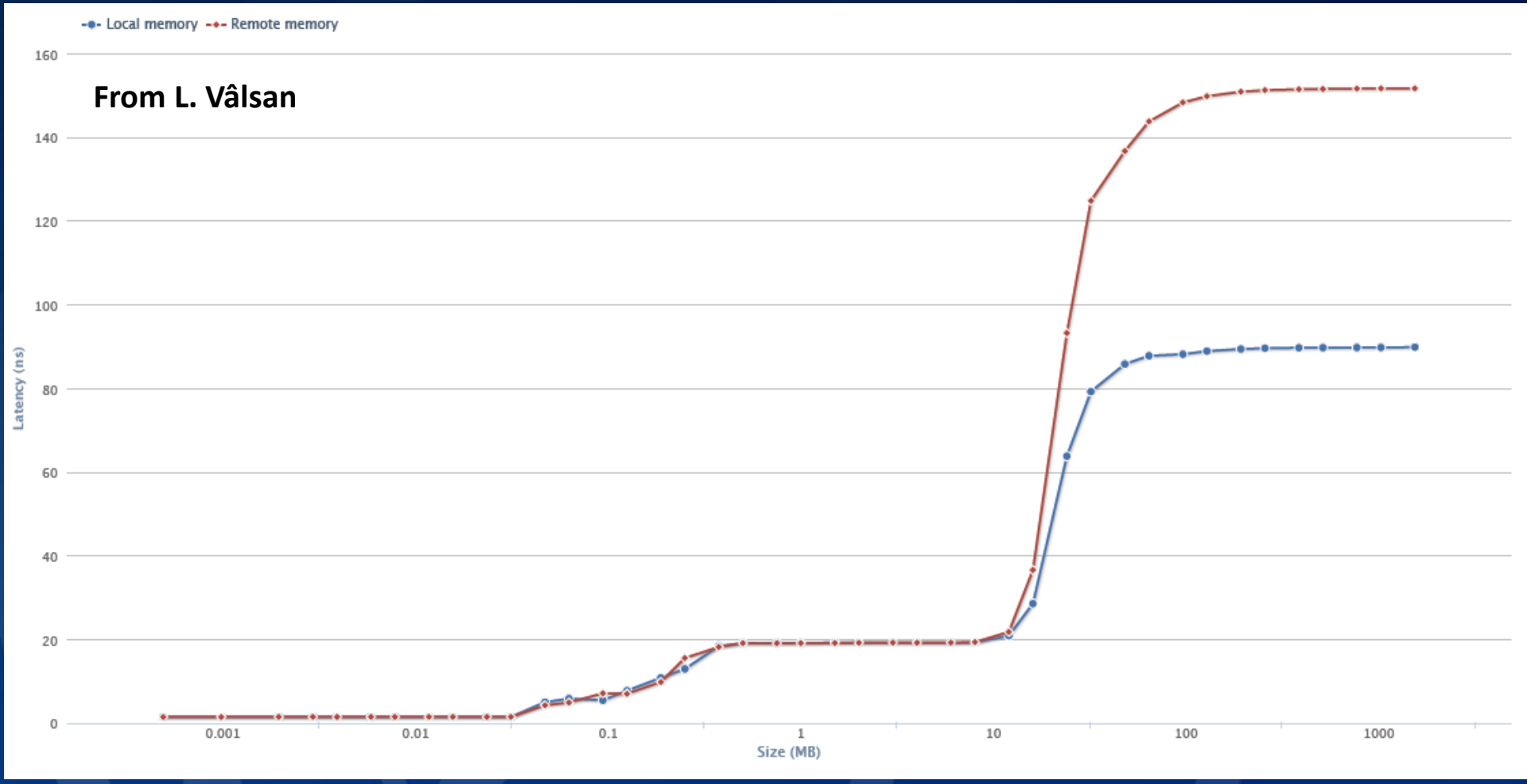
Heterogeneity

- A whole new set of problems
- How is heterogeneity expressed in hardware?
- How far from one node to another?
- Will the floating point results match?
- How to express heterogeneity in code?
- What coding standards to use? Will code compile anywhere? Will it perform well?
- How to split up the workload?

Heterogeneity – hardware aspects

- **Differing architectures talking to each other**
  - Incompatible instructions, registers, binaries
- **Limited memory**
  - Host sharing might be possible
- **Limited communication opportunities**
  - Larger node distance
  - Memory transfer costly but flops are cheap
  - Need standard interconnect/communication
- **Previously synchronous behavior might become asynchronous**

Non-uniform Memory Access



Heterogeneity – software aspects

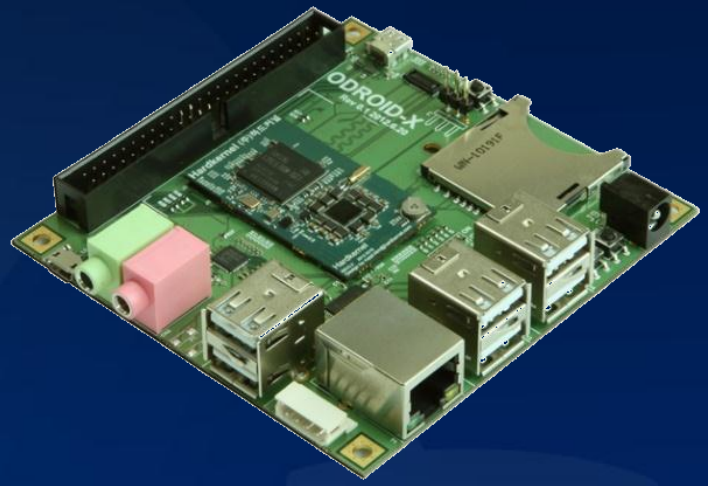
- **Software stack**
  - Disjoint compilers and toolsets
  - Interface kernel driver
    - Will it kick in during compute?
    - Will kick in during I/O
- **Source code**
  - Is it possible to maintain a single source?
    - Single source compiled to different targets
    - Single source running with different backends
  - How to express targeted code?
    - #pragma, #ifdef
    - OpenACC
    - Dedicated functions/kernels

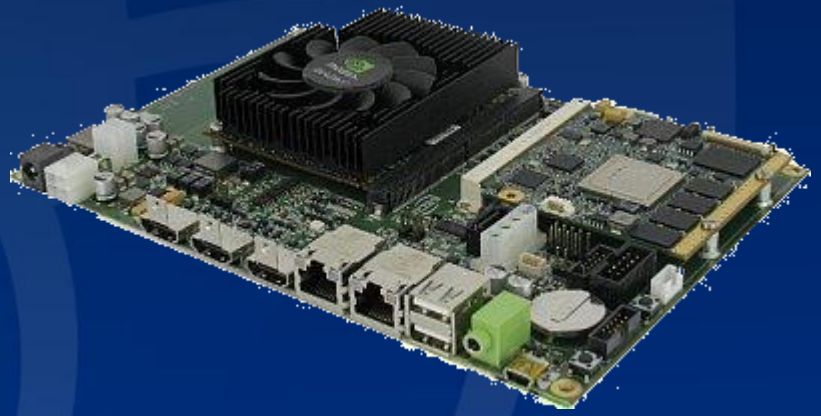
Heterogeneity – numerical aspects

- **Floating point results will differ**
  - Does the algorithm support that?
  - Is it possible to live with different results?
  - Reproducibility
- **Math functions will vary in precision across platforms**
  - "Fast" functions might be used as defaults!
- **Results can depend on data and operation ordering**
  - E.g. "if two items are equal, choose the first one" (risky)
  - Solution: sort or identify data uniquely
- **Associativity may vary across platforms**
  - Results differ as a result
- **In the end, it's a good thing**
  - Rethink floating point, algorithms
  - Optimize precision to that really needed ("meta libm")

ARM 64

- Plenty of servers coming
- “Sea of Cores”
- NEON extensions
- Well received, competitive with Intel Atom
- Interesting combos
  - Carma (used by LHCb)
  - Tidbit: AMD will have an ARM Opteron
- Software still needs work



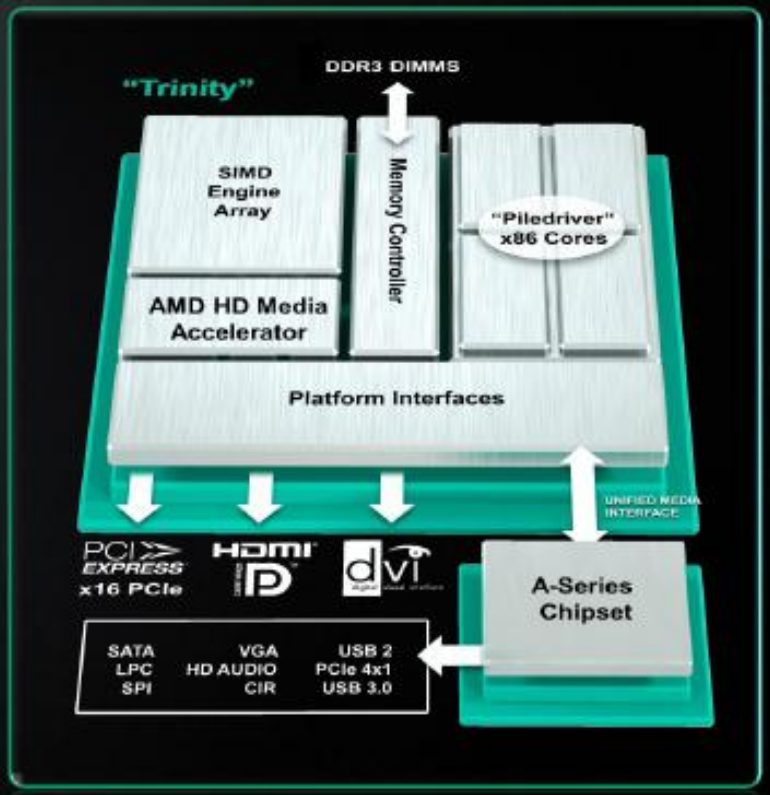


APUs

AMD APU "TRINITY" WITH AMD DISCRETE CLASS GRAPHICS

ALL NEW ARCHITECTURE FOR UP TO 50% GPU[1] AND UP TO 25% BETTER X86 PERFORMANCE[2]

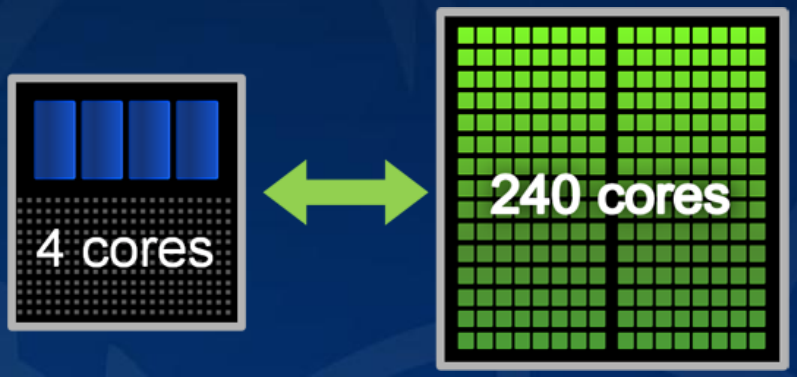
- "Piledriver" Cores
  - 2nd-Gen "Bulldozer" core ("Piledriver")
  - 3rd-Gen Turbo Core technology
- Multiple Configurations
  - Memory support up to DDR3-1866 (1600 for notebook)
  - Low power DDR3 (1.25V) support
  - Quad CPU Core with total of 4MB L2
- 2nd-Gen AMD Radeon™ with DirectX® 11 support
  - 384 Radeon™ Cores 2.0
- HD Media Accelerator
  - Accelerates and improves HD playback
  - Accelerates media conversion
  - Improves streaming media
  - Allows for smooth wireless video
- Enhanced Display Support
  - AMD Eyefinity Technology[3]
  - 3 Simultaneous DisplayPort 1.2 or HDMI/DVI links
  - Up to 4 display heads with display multi-streaming



AMD

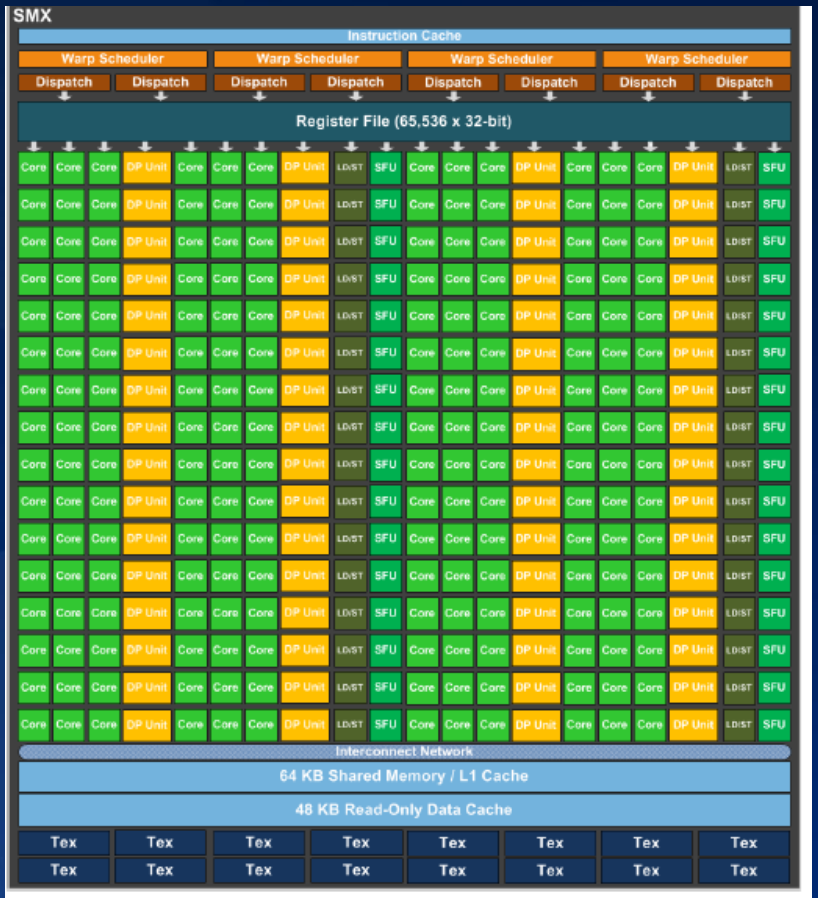
2 | AMD "Trinity" HotChips 2012 | Pending AMD Legal Release Approval | August 2012

GPUs



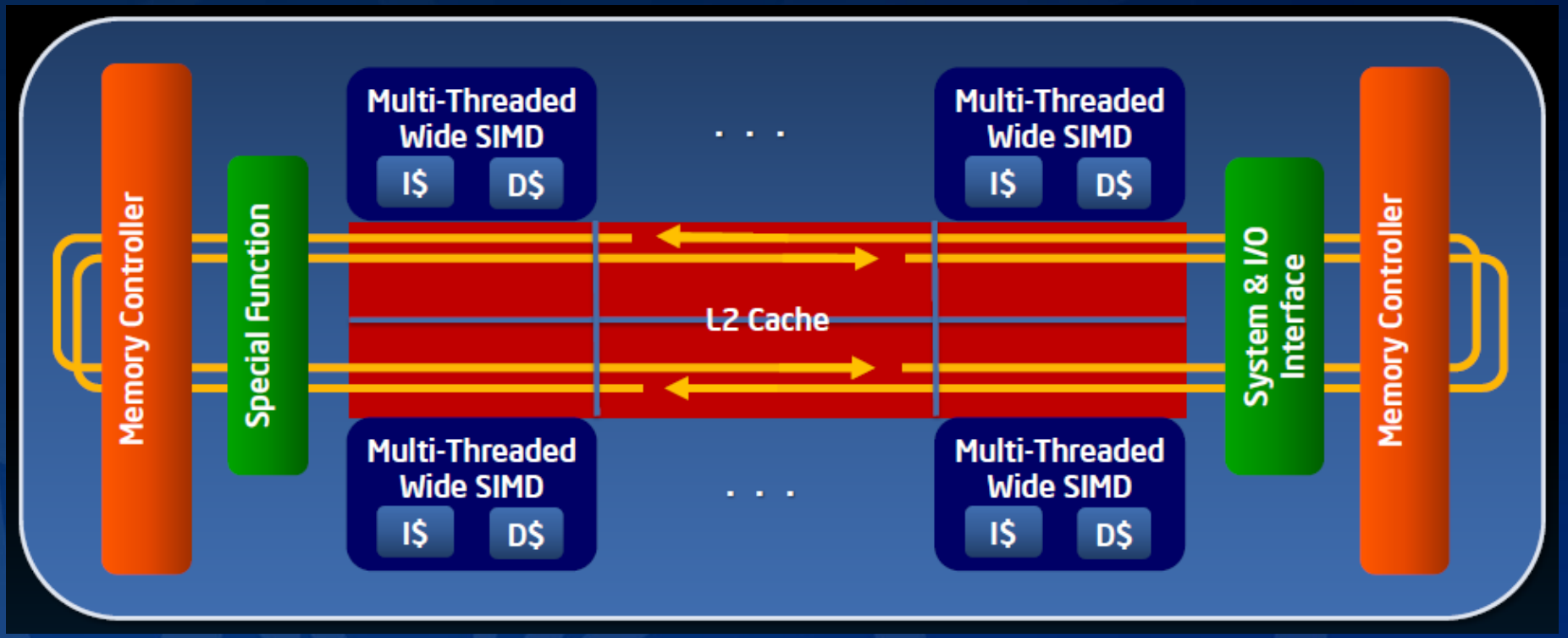
Not the same "cores" at all

(1 GPU core = 1 CPU SIMD element)

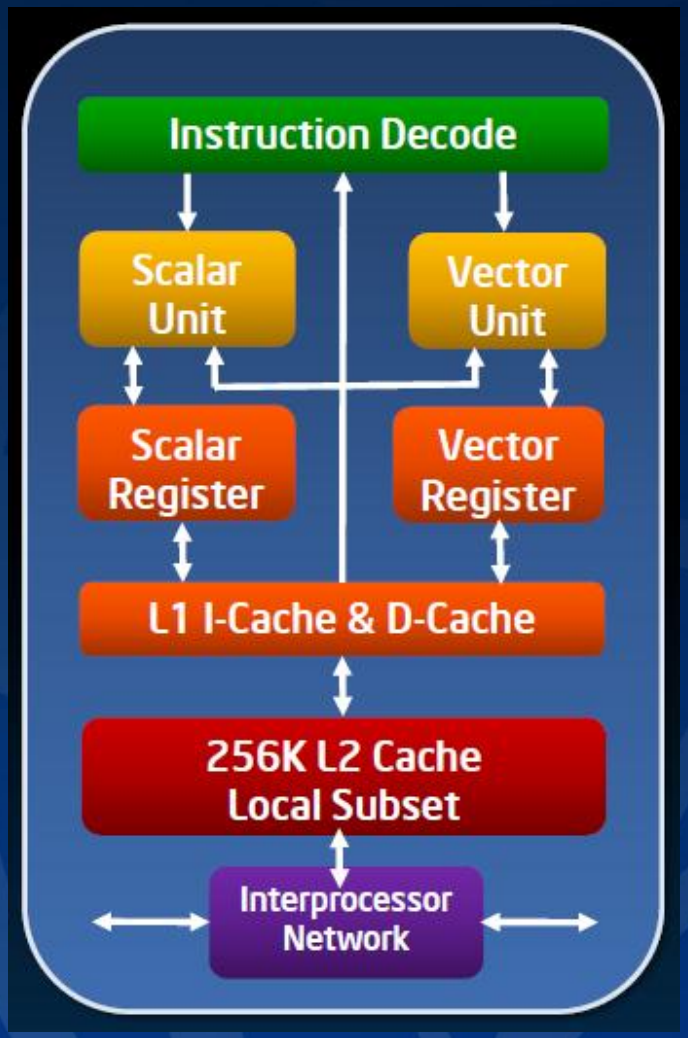


SMX: 192 single-precision CUDA cores, 64 double-precision units, 32 special function units (SFU), and 32 load/store units (LD/ST).

Intel MIC / Xeon Phi / KNC

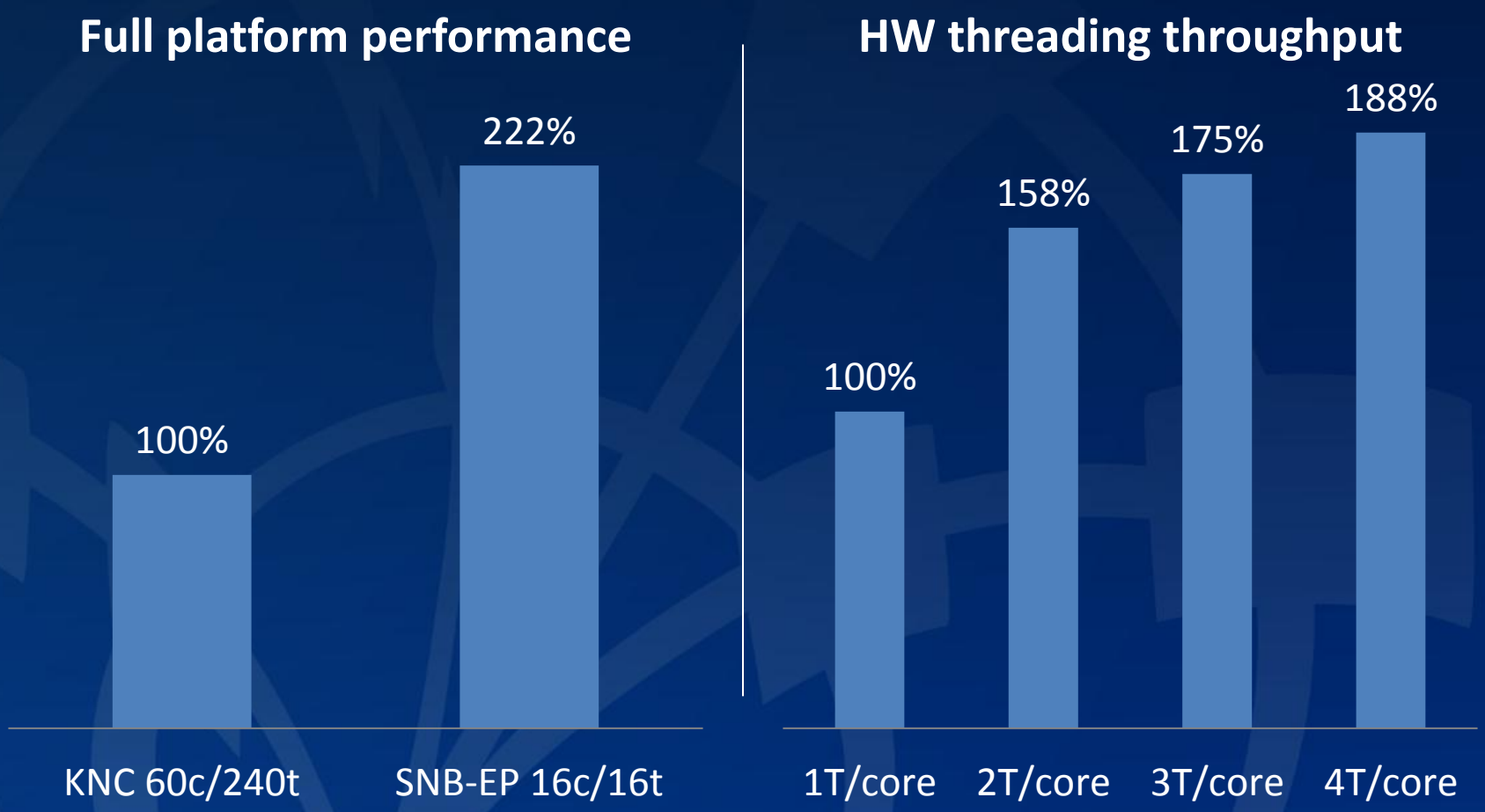


A Knights core



- Superscalar
- 32k L1 I/D cache
- 256k Coherent L2 caches
- 2x512-bit ring bus inter-CPU network
- 4-wide SIMD with separate register sets

MTG4 performance on MIC
(higher is better, no vectorization)



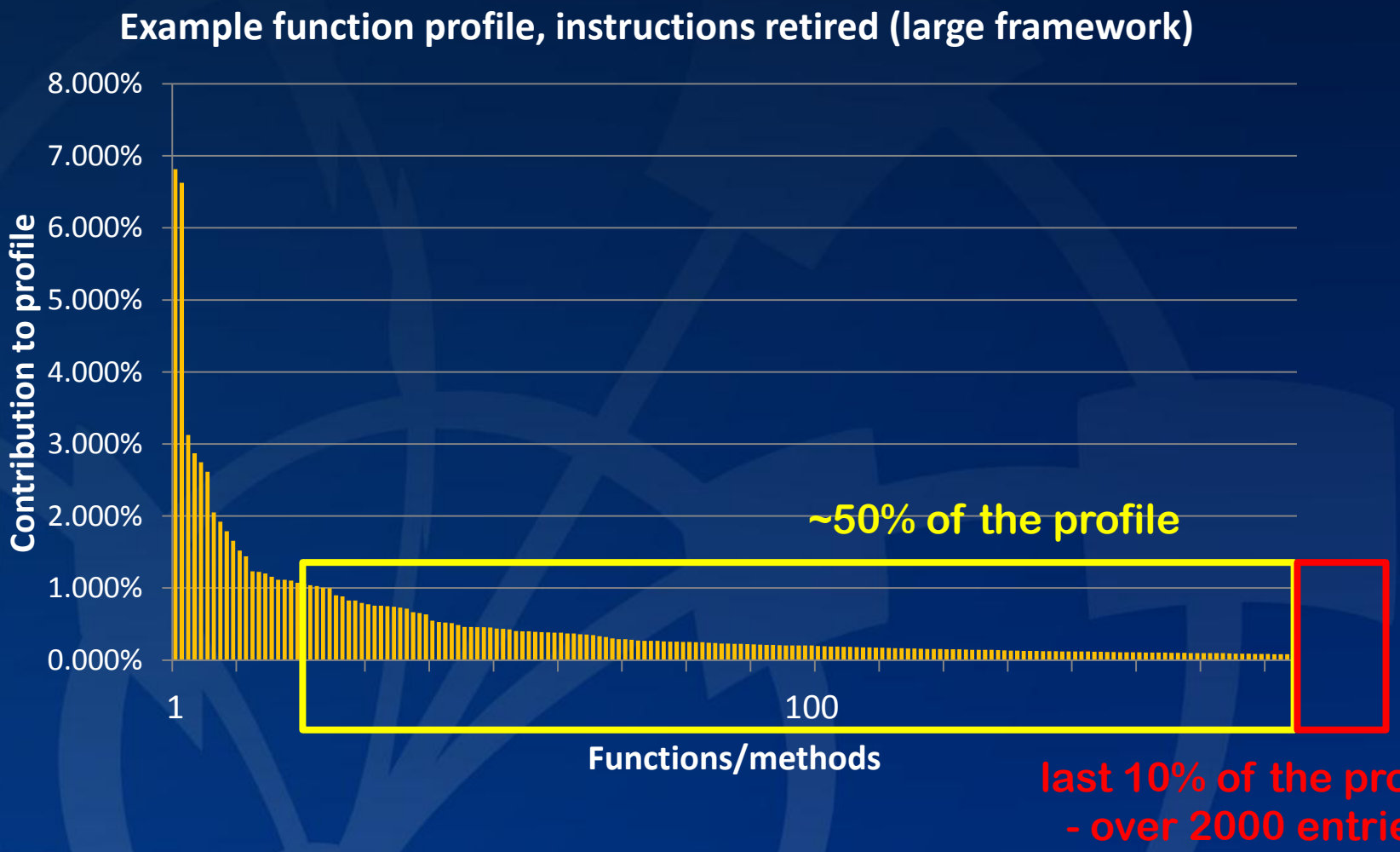
Porting – how much work?

	LOC	1st port time	New ports	Tuning
TF	< 1’000	days	N/A	2 weeks
MLFit	3’000	< 1 day	< 1 day	weeks
MTG	2’000’000	1 month	< 1 day	< 1 week

Intel MIC - conclusions

- No-go without vectorization
- Optimized applications surpass dual-socket Xeon performance
- Non-optimized performance reaches approximately a single Xeon socket
- Math function usage and performance are key vis a vis Xeon
- Compiler maturity still an open question

Large jobs – profile fragmentation



Mainstream production software today

- **Independent events (collisions of particles)**
  - trivial (read: pleasant) parallel processing
- **Large C++ frameworks with millions of lines of code**
  - Thousands of shared libraries in a distribution, gigabytes of binaries
  - Low number of key players but high number of brief contributors
- **Large regions of memory read only or accessed infrequently**
- **Characteristics:**
  - Significant portion of double precision floating point (10%+)
  - Loads/stores up to 60% of instructions
  - Unfavorable for the x86 microarchitecture (even worse for others)
    - Low number of instructions between jumps (<10)
    - Low number of instructions between calls (several dozen)
- **For the most part, code not fit for accelerators in its current shape**
- **Conservative compiler options constrain optimization**

The need for Data Oriented Design

- Design around data flow rather than control flow
- Make data contiguous and cache-friendly
- Make data requests cache-friendly and predictable
- Use good memory management libraries
- Avoid frequent or regular calls to virtual functions
- Pointers can be avoided in lieu of fixed references

Software support for parallelism

- **Not all technologies mix**
  - For us, autovectorization/Cilk+ + OpenMP/TBB + MPI worked very well
  - See A. Lazzaro's report (openlab webpages)
- **On our radar:**
  - TBB
  - OpenMP
  - Cilk+
  - Autovectorization
  - Message passing (multiprocess)
  - several others

Autovectorization

- **Heavily compiler and code dependent**
  - Although the principle is the same, GCC and ICC differ
- **Numerous benefits, numerous pitfalls**
  - Speedups of 2x are not uncommon
  - Delicate: for example, one data type change in your loop variable can derail all compiler efforts to vectorize the loop
- **Pros:**
  - Speedups can often be achieved with virtually no effort on the programmer's part (even on large code bases!)
  - Compiler reports make it easier to work with this technique
  - Architecture independent on the source level
- **Cons:**
  - Difficult to control, many pitfalls
  - Heavy dependencies
  - Gains not as significant as with direct techniques – only as good as the compiler

Cilk+

- **"C extended array notation"**
  - A way to express vector parallelism
  - Arrays explicitly denoted in code (syntax extension)
- **Example syntax:**
  - `A[index:num_elements] = B[index:num_elements]`
  - `A[i:n] = B[i:n]`
  - `A[i:n] = 2*B[i:n]`
  - `A[i:n][j:m] += 5`
- **Implementation since ICC 12.0 compiler, trickling down to GCC**
  - No switches or magic needed, just use the extensions

Smart intrinsics wrapping

- Vector Classes (VC – M. Kretz) – a portable library for explicit vectorization

```
__m128 c = _mm_add_ps(a, b);
```

vs.

```
float_v c = a + b;
```

- The concept (zero overhead):

```
v_type operator +(const v_type &a, const v_type &b)
      { return __mm256_add_pd(a, b); }
```

- Supports multiple formats (including scalar)

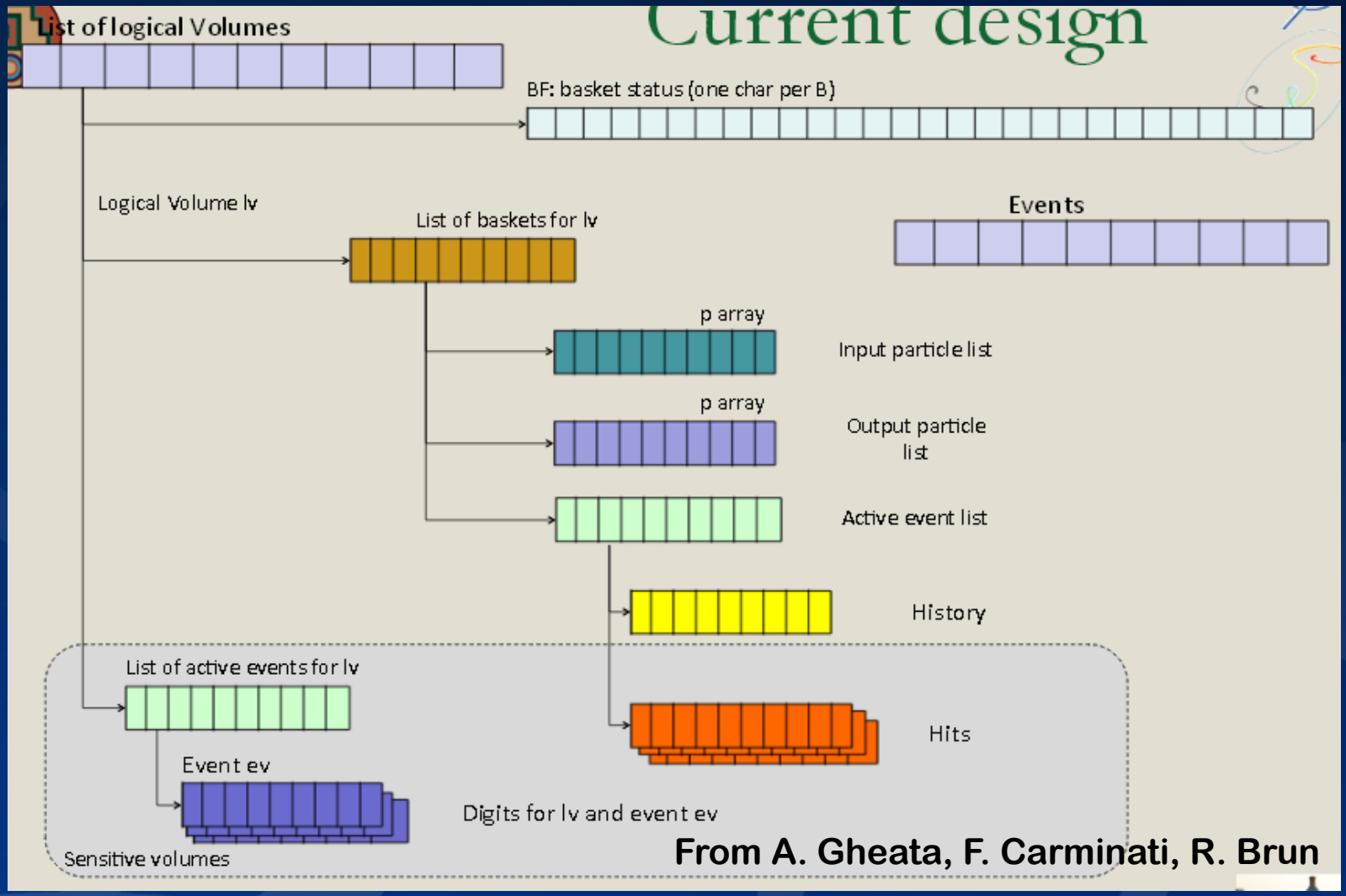
OpenCL / CUDA

- Kernel based languages
- Neither has grown much past C
- In terms of performance, neither is truly device independent
- Taking into account:
  - Data transfer time
  - Very limited caches
  - Limited number of control threads
  - Limited on-board memory

Proofs of Concept

- **CERN Concurrency Forum**
  - Numerous small technology evaluations
- **openlab evaluations**
- **Int'l Tracking Workshops**
- **Wide spectrum of technologies – how to pick and mix?**

Next-gen physics simulation



Example single source approach (assuming incompatible standards)

common.cpp:

```
__DECL FitTrack(int n) {
....
}
```

cpu_wrapper.cpp:

```
#define __DECL void
#include ``common.cpp``

void FitTracks() {
  for (int i = 0;i < nTr;i++) {
    FitTrack(n);
  }
}
```

gpu_wrapper.cpp:

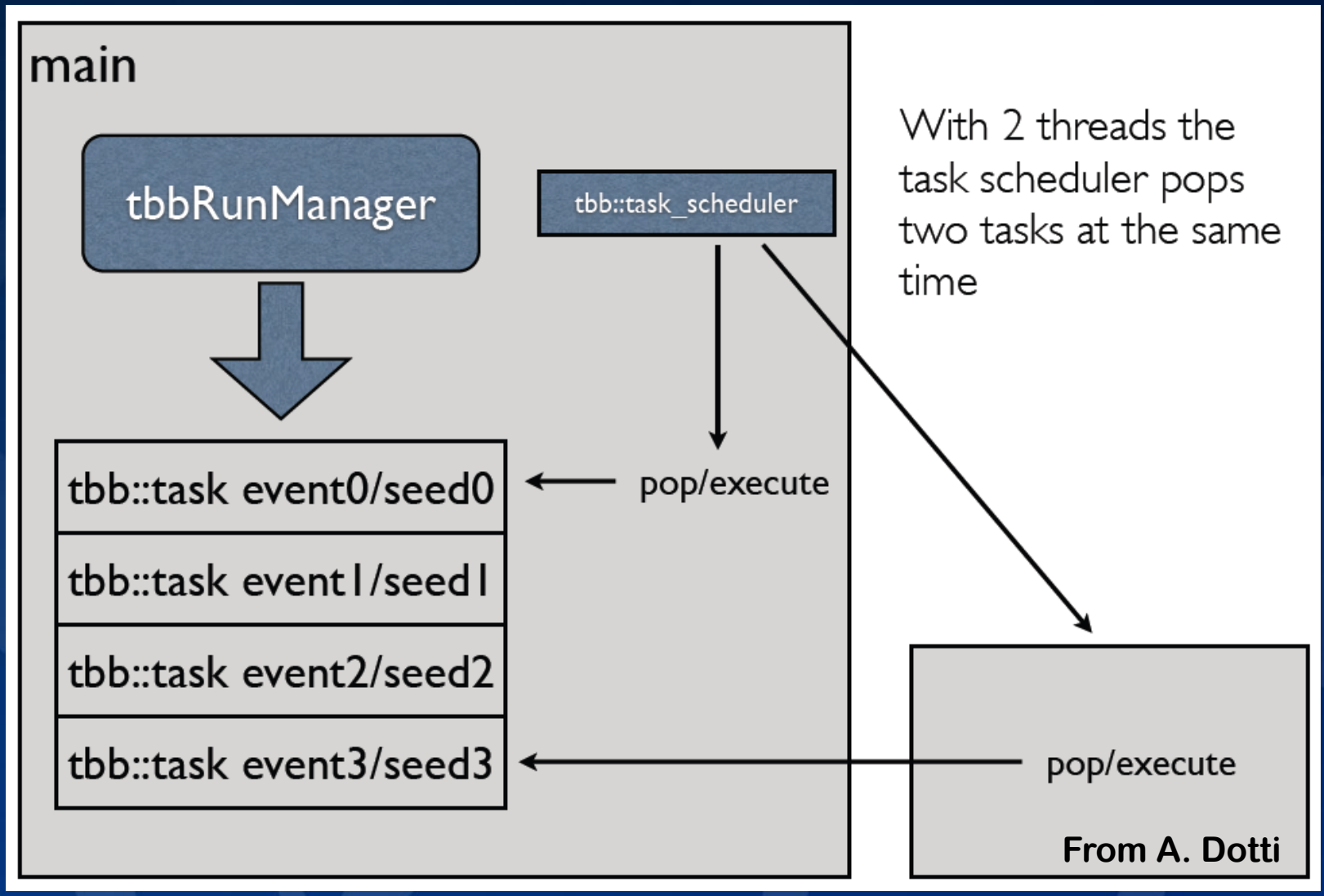
```
#define __DECL __device void
#include ``common.cpp``

__kernel void FitTracksGPU() {
  FitTrack(threadIdx.x);
}

void FitTracks() {
  FitTracksGPU<<<nTr>>>();
}
```

From David Rohr

Geant4MT + TBB



Low power computing

Type	Cores	Power	Events/ min/core	Events/ min/Watt
Exynos4412 Prime @ 1.704GHz	4	4W?	1.14	1.14
Xeon L5520 @ 2.27GHz	2x4	120W?	3.50	0.23
Xeon E5-2630L @ 2.0GHz	2x6	190W?	3.33	0.21

From P. Elmer et al. (initial results, work in progress)

Low power computing

Brunel running on ARM

- Multiple Brunel test runs show –
  - almost 100% identical results to x86_64
  - negligible residual differences thought to come from 32-bit architecture (check pending)

"Brem Match"	sum		mean/eff^*		rms/err^*	
	ARMv7	x86_64	ARMv7	x86_64	ARMv7	x86_64
#calos	50085	50085	60.489	60.489	30.140	30.140
#chi2	2.737109e+09	2.737105e+09	5009.1	5009.1	2866.4	2866.4
#links	546430	546415	659.94	659.92	611.38	611.33
#overflow	4038434	4038430	4877.3	4877.2	5074.2	5074.0
#tracks	58610	58609	70.785	70.784	48.513	48.511

LHCb Reconstruction on ARM - Vijay Kartik 11

From V. Kartik, B. Couturier, M. Clemencic, N. Neufeld
Work in progress

“Yes, we can”

	Stage	Description	Time/track	Speedup
		Initial scalar version	12 ms	–
Intel	1	Approximation of the magnetic field	240 μs	50
Intel	2	Optimization of the algorithm	7.2 μs	35
Intel	3	Vectorization	1.6 μs	4.5
Cell	4	Porting to SPE	1.1 μs	1.5
Cell	5	Parallelization on 16 SPEs	0.1 μs	10
		Final simdized version	0.1 μs	120000

10000x faster on any PC

Comp. Phys. Comm. 178 (2008) 374-383

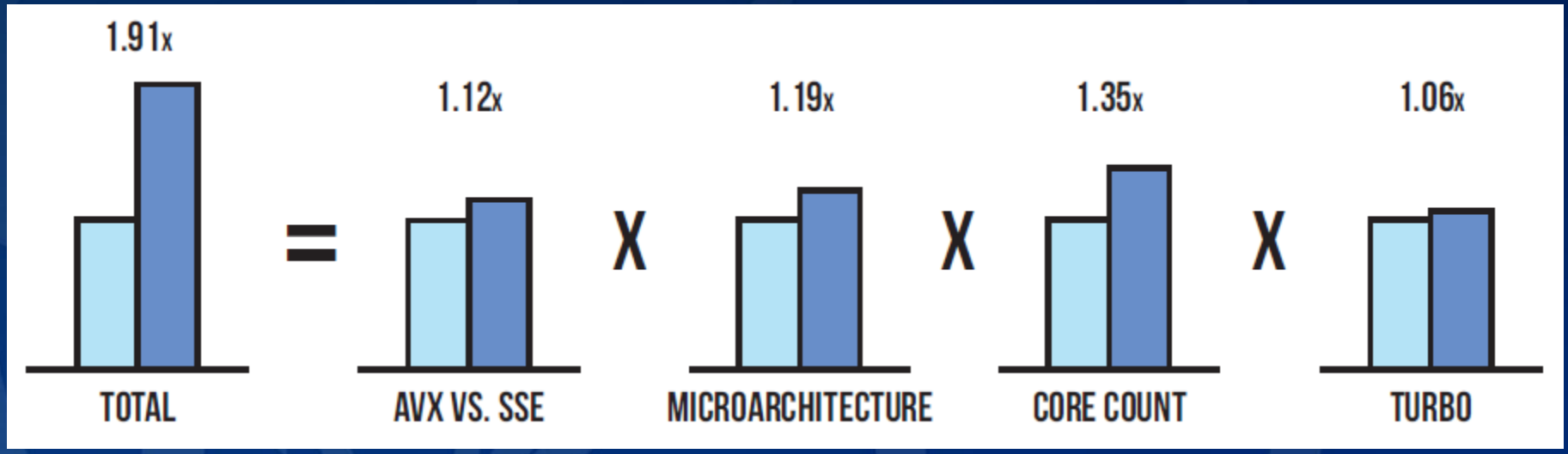
From I. Kisel et al.

Andrzej Nowak (OpenLab, CERN) by Hans von der Schmitt (ATLAS) at GPU Workshop, DESY, 15-16 April 2013

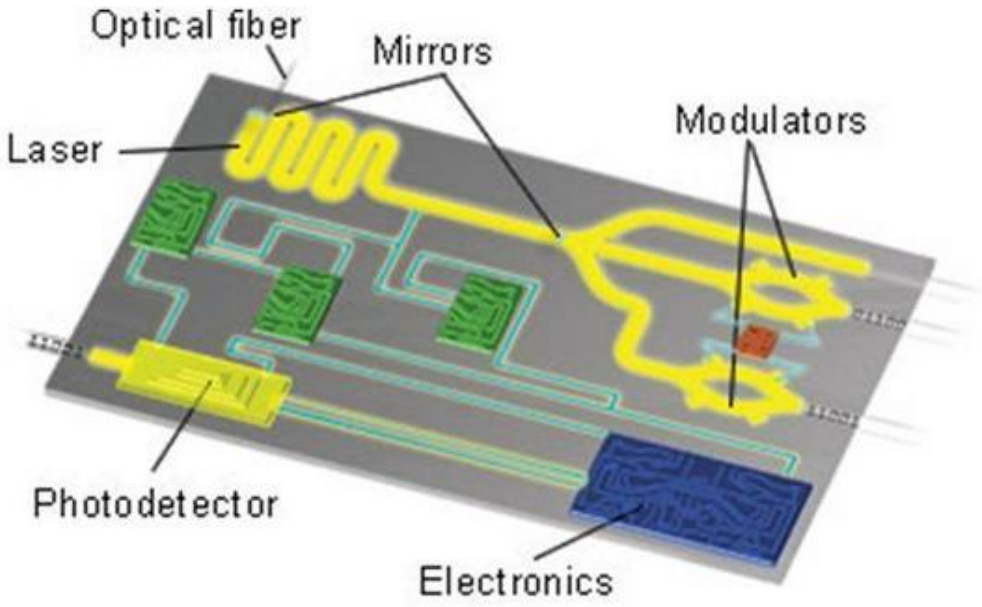
	SIMD	Instr. Level Parallelism	HW Threads	Cores	Sockets	Factor	Efficiency
MAX	4	4	1.35	8	4	691.2	100.0%
Typical	2.5	1.43	1.25	8	2	71.5	10.3%
HEP	1	0.80	1	6	2	9.6	1.4%
CBM@FAIR	4	3	1.3	8	4	499.2	72.2%

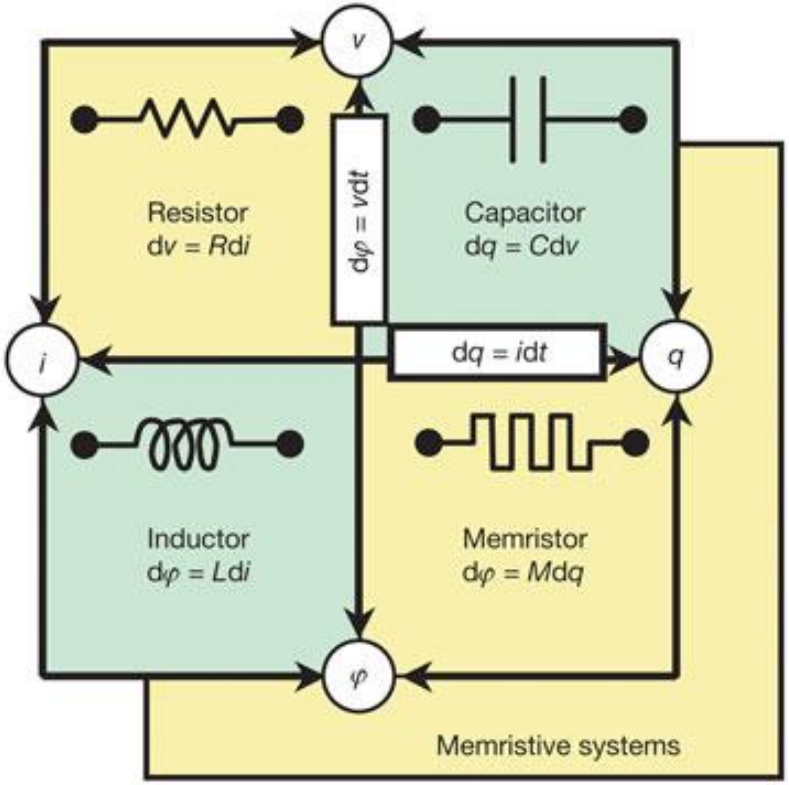
“Yes, we can”

- A. Lazzaro’s RooFit upgrade from Westmere-EP to Sandy Bridge-EP



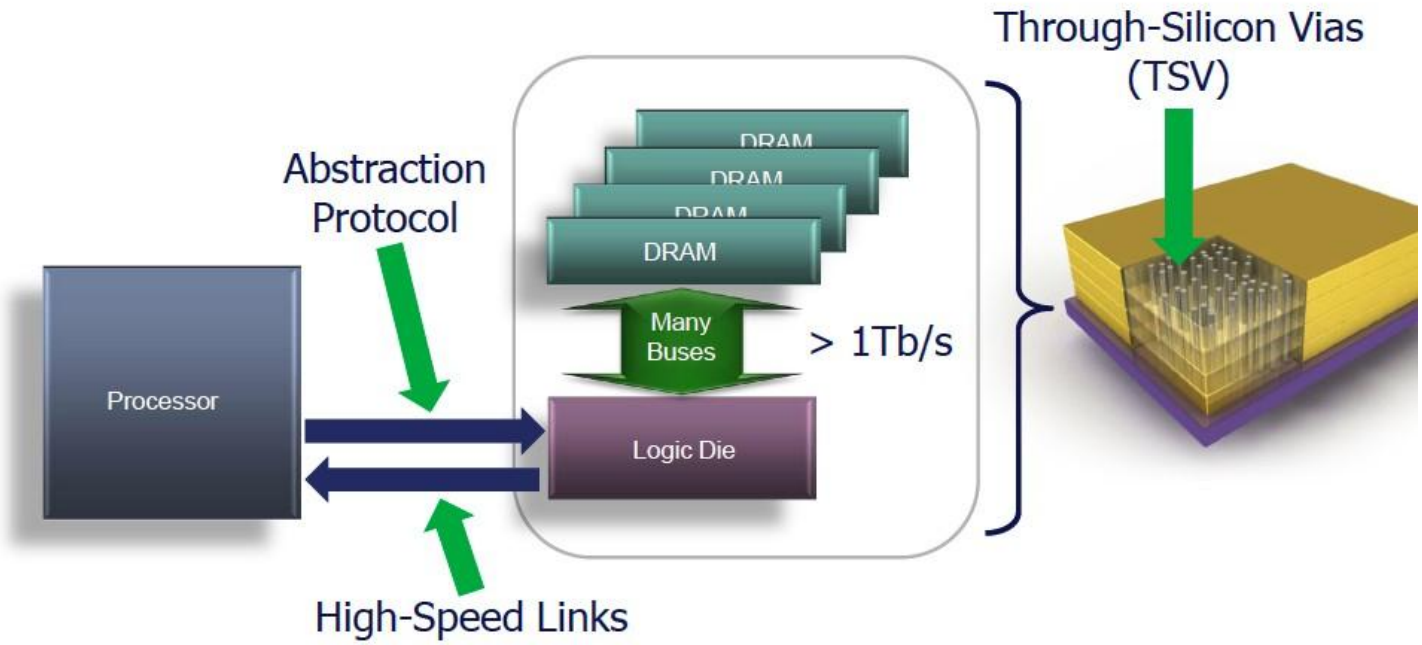
“Crazy” stuff


Optical fiber
Mirrors
Laser
Modulators
Photodetector
Electronics


v
Resistor
dv = Rdi
dφ = vdt
Capacitor
dq = Cdv
i
dq = idt
q
Inductor
dφ = Ldi
Memristor
dφ = Mdq
φ
Memristive systems

“Crazy” stuff

Hybrid Memory Cube (HMC)



	Bandwidth	Energy
DDR3 (Today)	> 10GB/s	> 50 pJ / bit
HMC (Prototype)	> 100 GB/s	< 10 pJ / bit

Slide: Micron Tech. / S. Borkar, Intel

What should next-generation code look like?

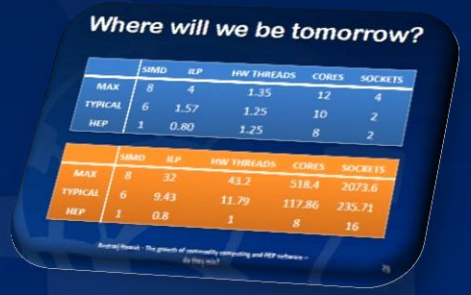
Balance of programmability and performance	• Can't entirely sacrifice one for the other • Can't ignore either
Using universal standards, languages and compilers	• Sustainability, interoperability
Scaling in multiple dimensions	• Parallelization, vectorization, multi-process domain, message passing readiness, "asynchronicity", lack of coherency • Good domain decomposition a prerequisite
Power efficient	• Optimal balance of data movement and compute

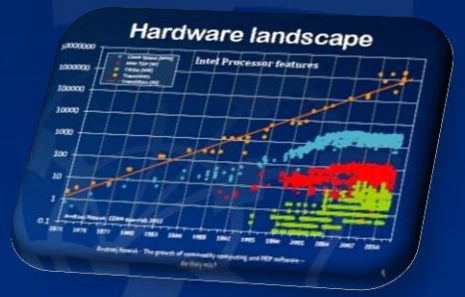
Tuning – reality check

Level	Potential gains	Estimate
Algorithm	Major	~10x-1000x
Source code	Medium	~1x-10x
Compiler level	Medium-Low	~10%-20% (more possible with autovec or parallelization)
Operating system	Low	~5-20%
Hardware	Medium	~10%-30%

Summary

- Embrace technology, adapt to constant change
  - A lot of it in known directions
- Domain experts working alongside technical experts
- No longer a “one team effort”
- Weaknesses can become strengths
- Use good tools
- Now is the time, the gap is growing





THANK YOU

Q & A

Questions? Andrzej.Nowak@cern.ch