

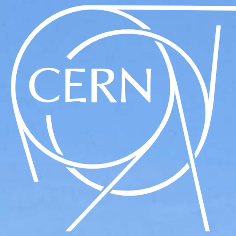
# Software optimization in the many-core era – the CERN case

Studencki Festiwal Informatyczny, Krakow  
March 16<sup>th</sup> 2013

Andrzej Nowak, CERN openlab

[Andrzej.Nowak@cern.ch](mailto:Andrzej.Nowak@cern.ch)

# PART 1: BACKGROUND



# The European Particle Physics Laboratory based in Geneva, Switzerland

Founded in 1954 by 12 countries for fundamental physics research in a post-war Europe

In 2013, it's a global effort of 20 member countries and scientists from 110 nationalities, working on the world's most ambitious physics experiments

~2'500 personnel, > 15'000 users  
~1 bln CHF yearly budget







Mont Blanc (4,808m)

Geneva (pop. 190'000)

Lake Geneva (310m deep)

SUISSE  
FRANCE

CMS

LHCb

ATLAS

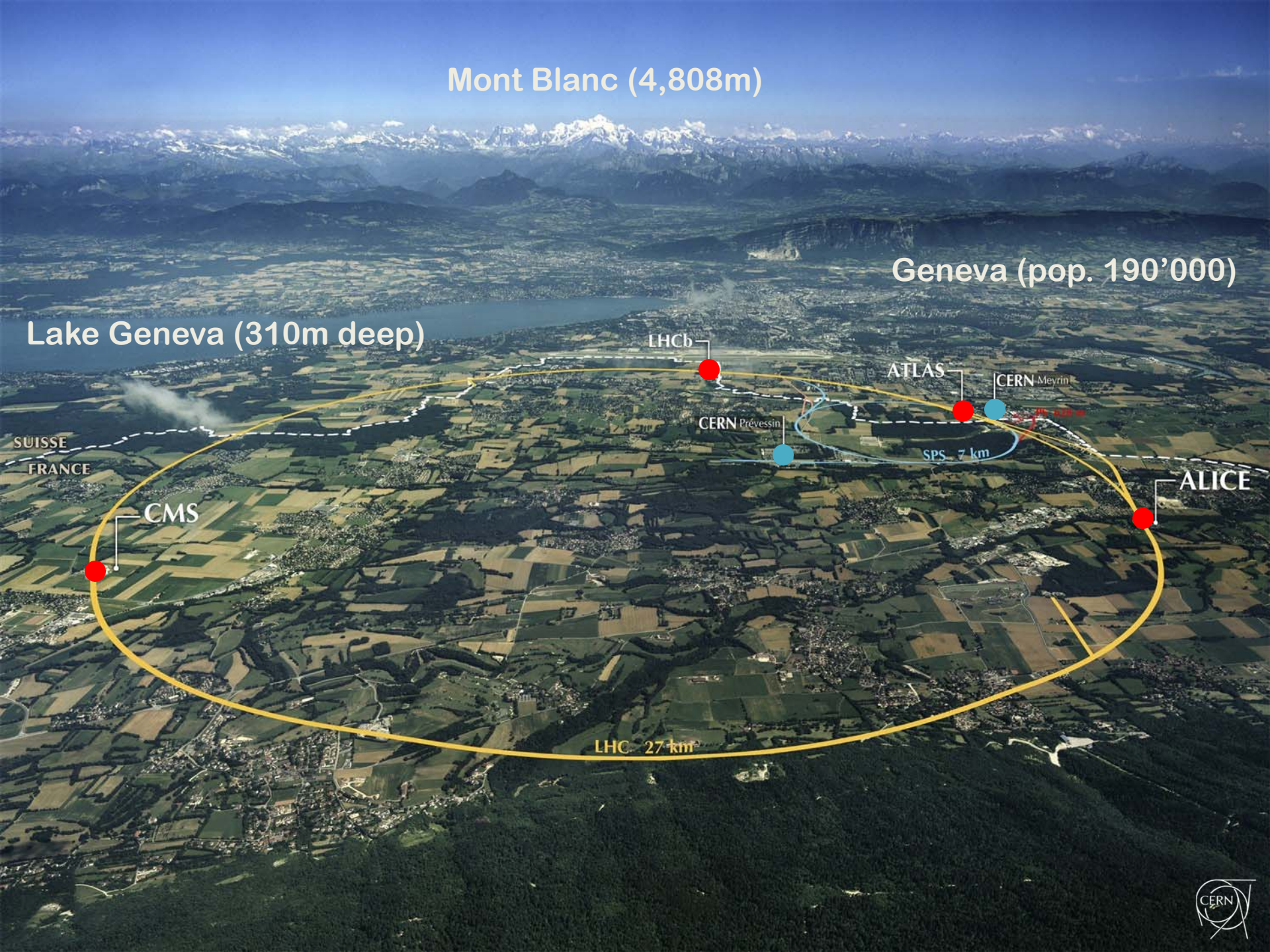
CERN Meyrin

CERN Prévessin

SPS 7 km

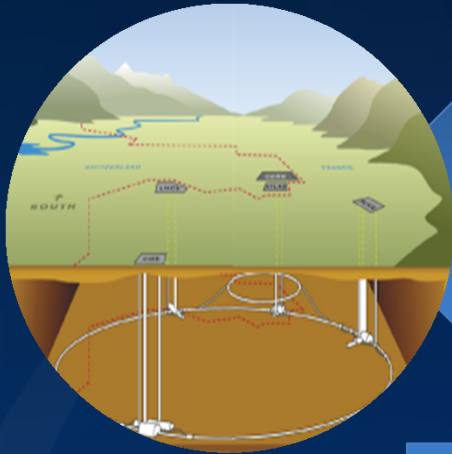
ALICE

LHC 27 km



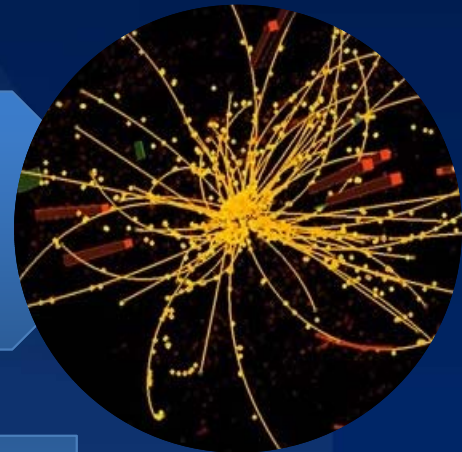


# The Large Hadron Collider



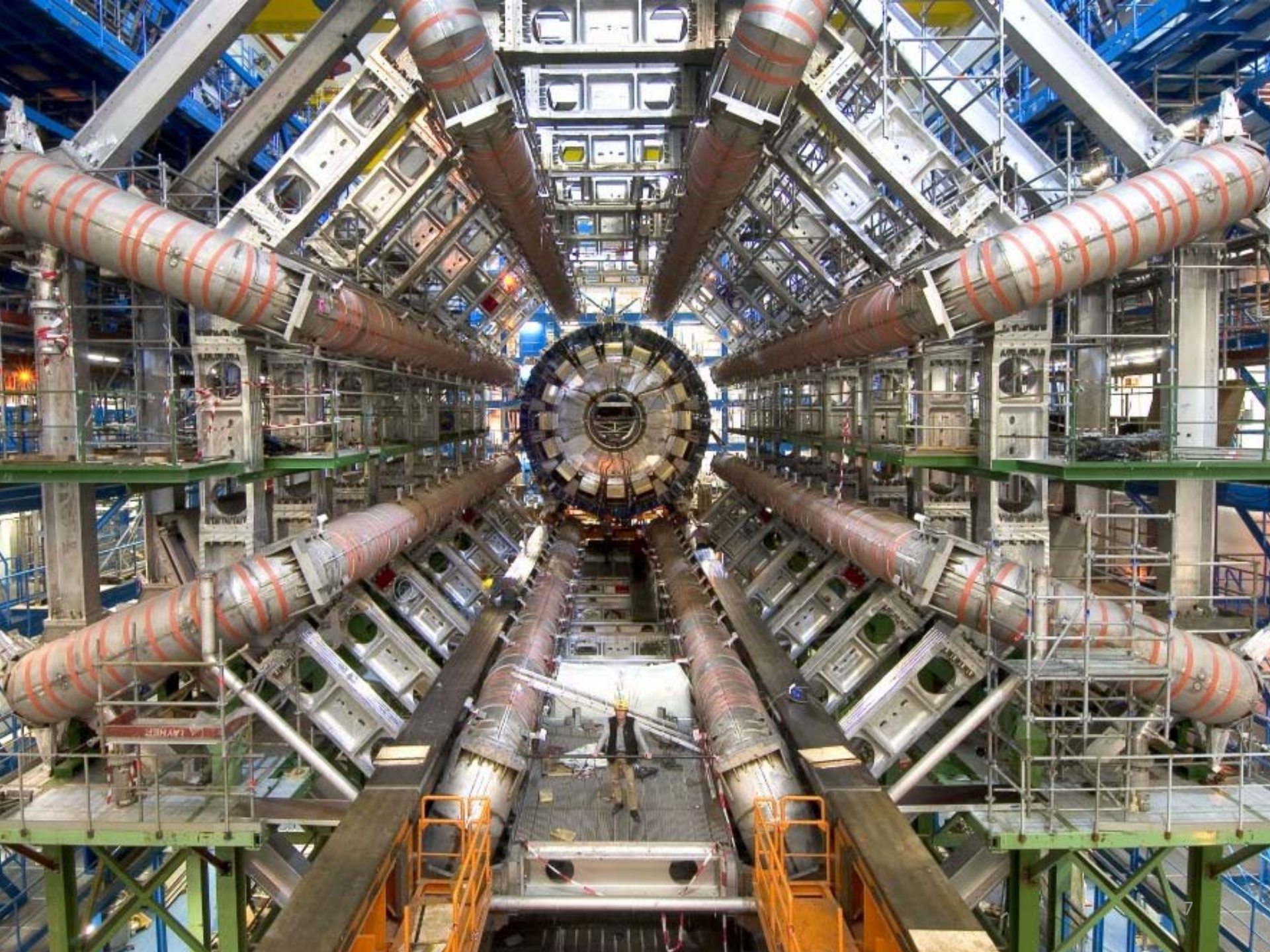
27 km underground  
superconducting ring – possibly the  
largest machine ever built by man

40 million collisions per second



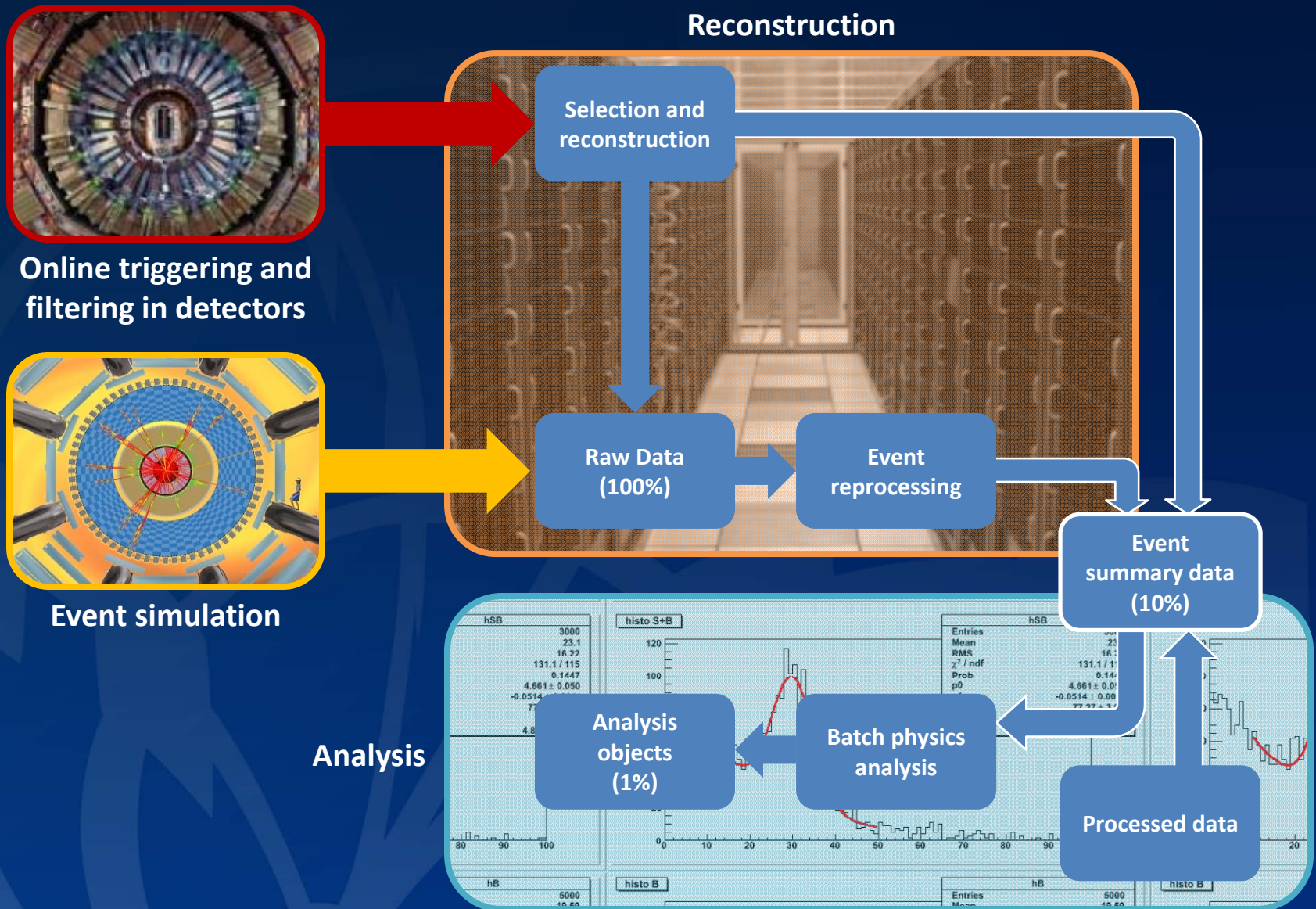
150-200 MW power consumption



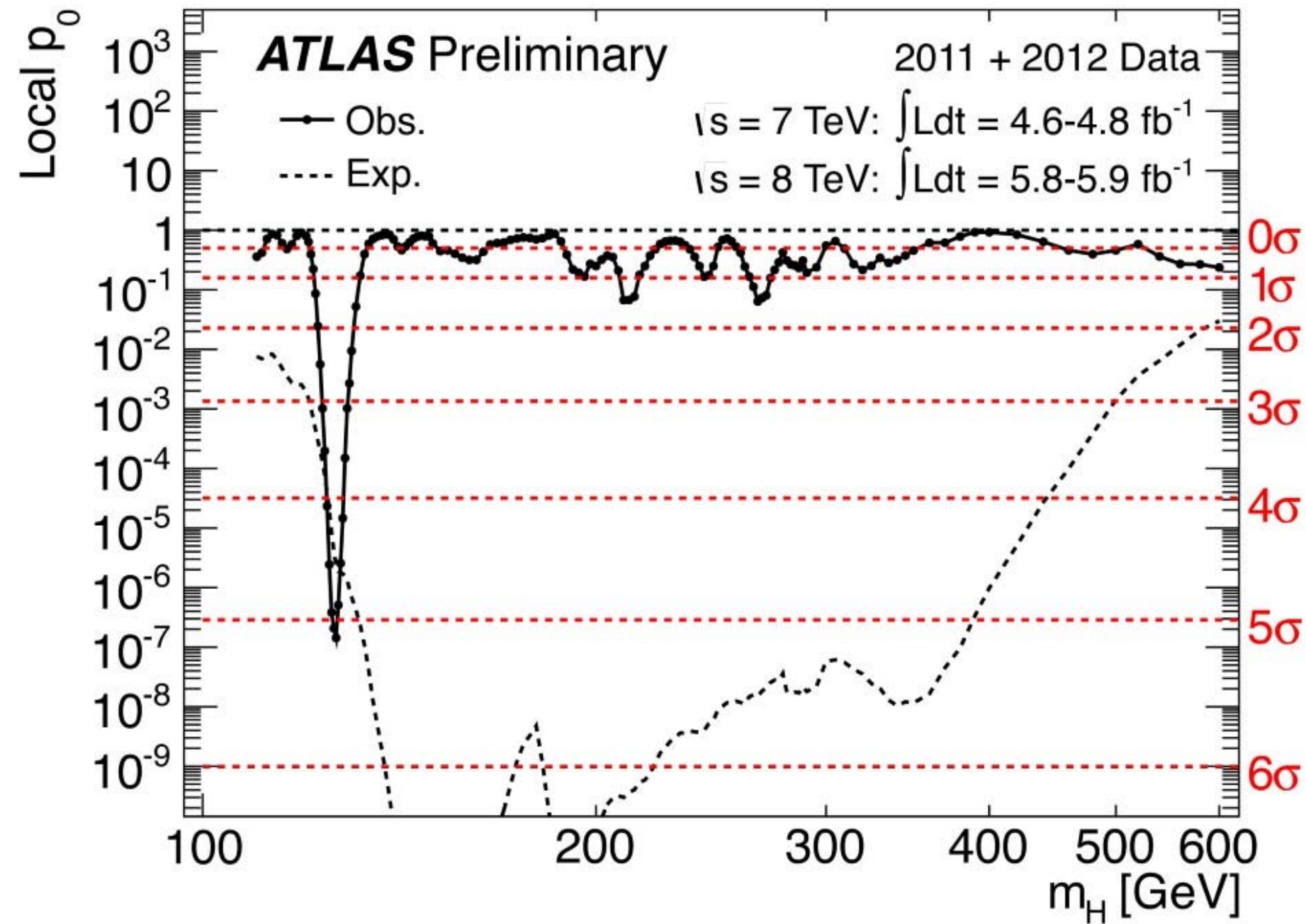




# Data flow from the LHC detectors

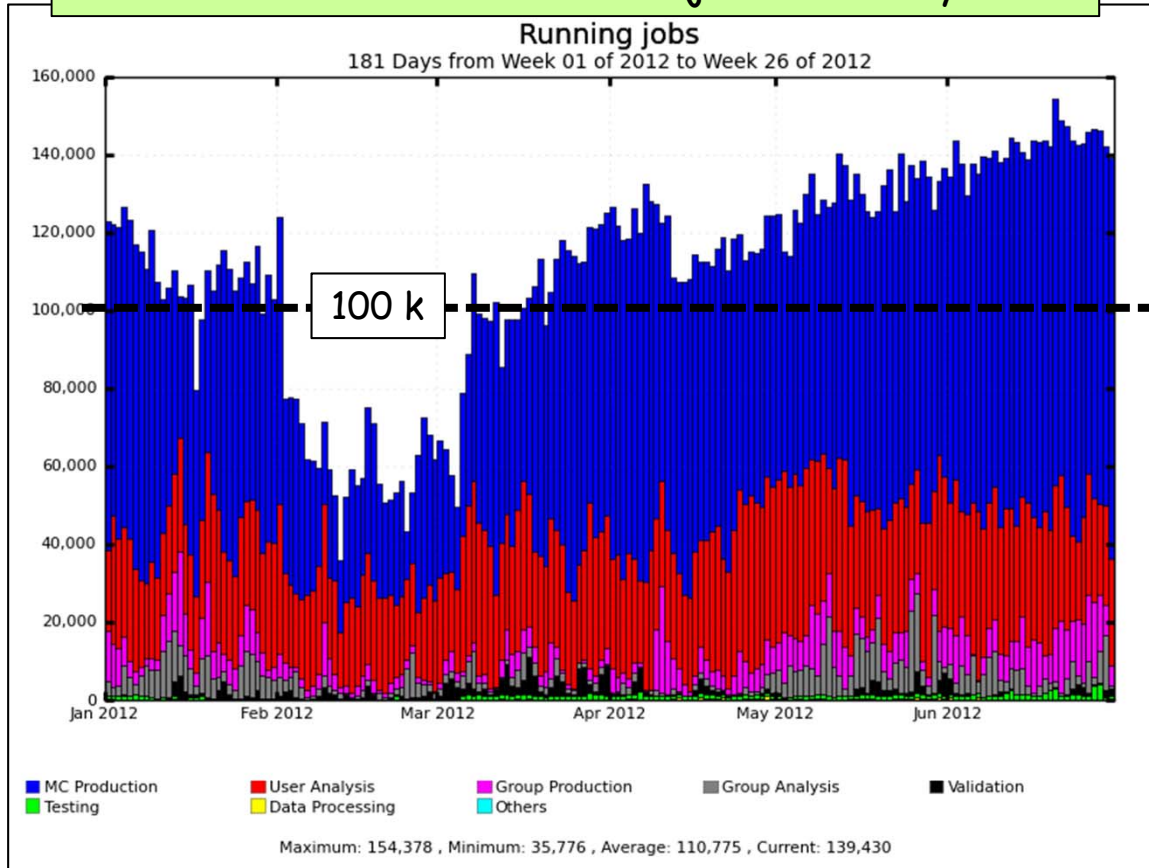






It would have been impossible to release physics results so quickly without the outstanding performance of the Grid (including the CERN Tier-0)

### Number of concurrent ATLAS jobs Jan-July 2012



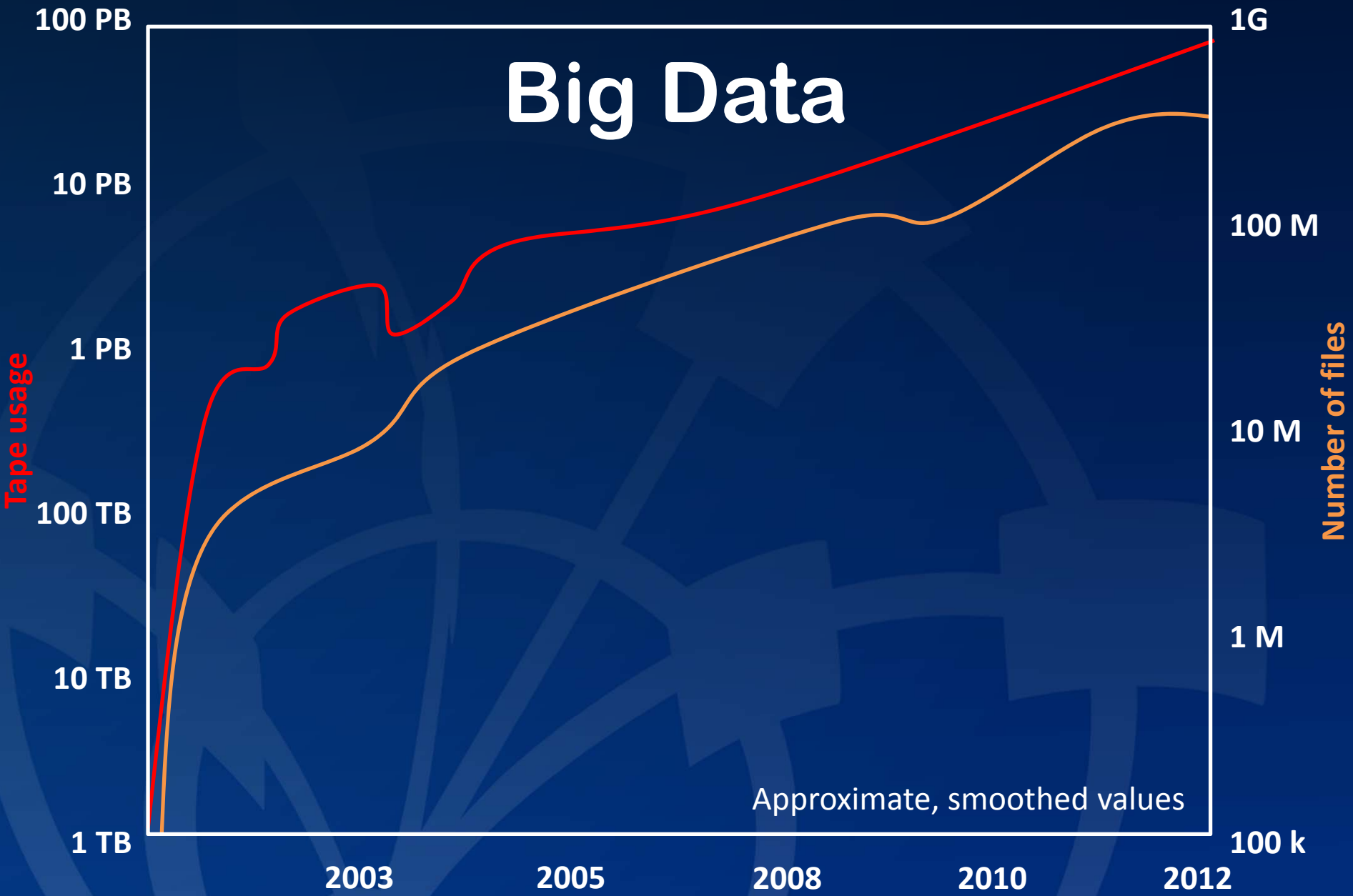
Includes MC production, user and group analysis at CERN, 10 Tier1-s, ~ 70 Tier-2 federations → > 80 sites

> 1500 distinct ATLAS users do analysis on the GRID

- ❑ Available resources fully used/stressed (beyond pledges in some cases)
- ❑ Massive production of 8 TeV Monte Carlo samples
- ❑ Very effective and flexible Computing Model and Operation team → accommodate high trigger rates and pile-up, intense MC simulation, analysis demands from worldwide users (through e.g. dynamic data placement)



# Big Data



Approximate, smoothed values

# Innovation in computing

1989: First high bandwidth transatlantic links

1999: The Grid vision materializes

2003: Several Internet2 land speed records

2012: LHC delivering intense data challenges

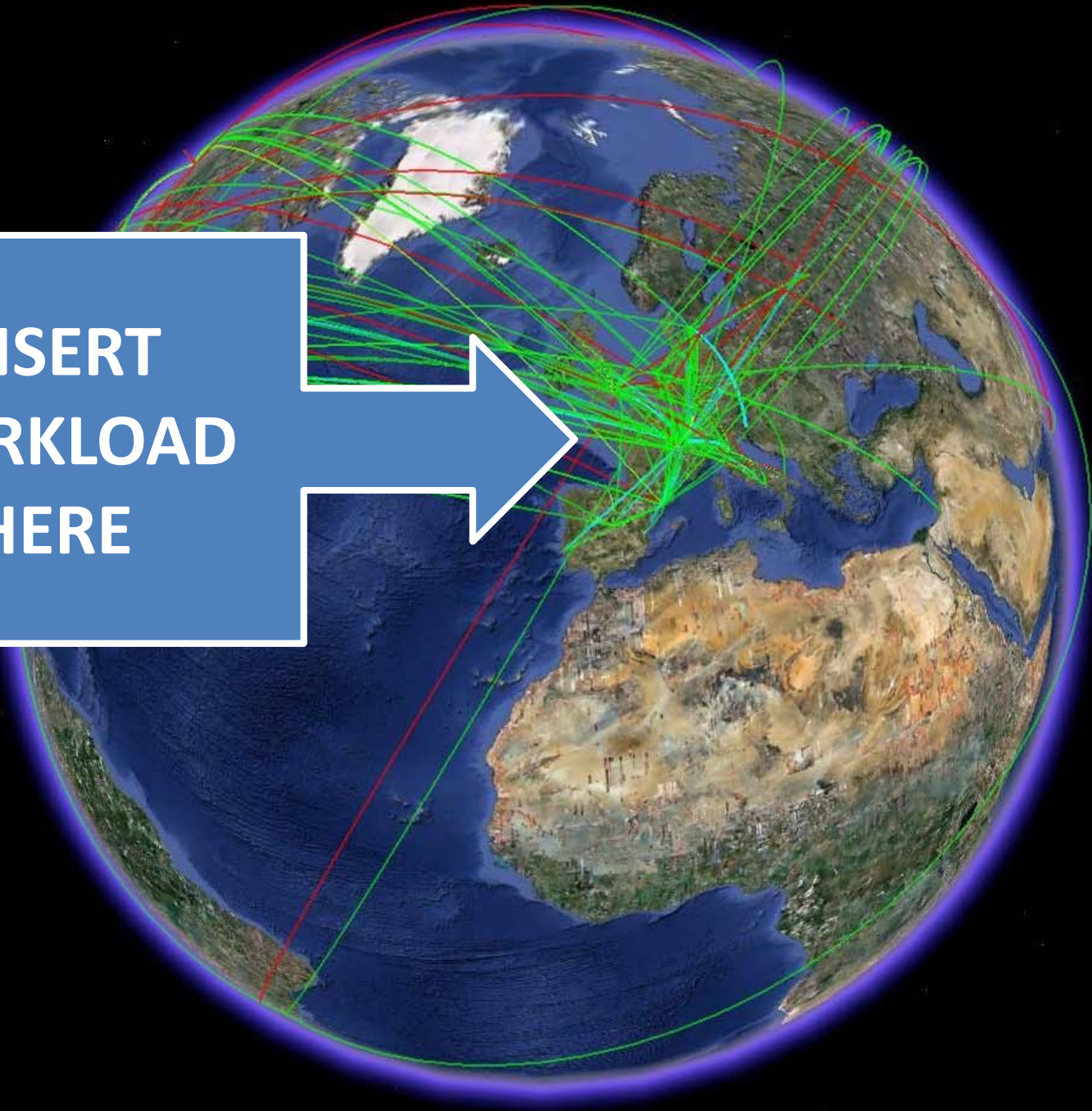
1991: The World Wide Web is born at CERN

2001: CERN wins Computerworld's 21<sup>st</sup> Century Achievement Award for SHIFT

2008: The WLCG is the world's largest grid



**INSERT  
WORKLOAD  
HERE**



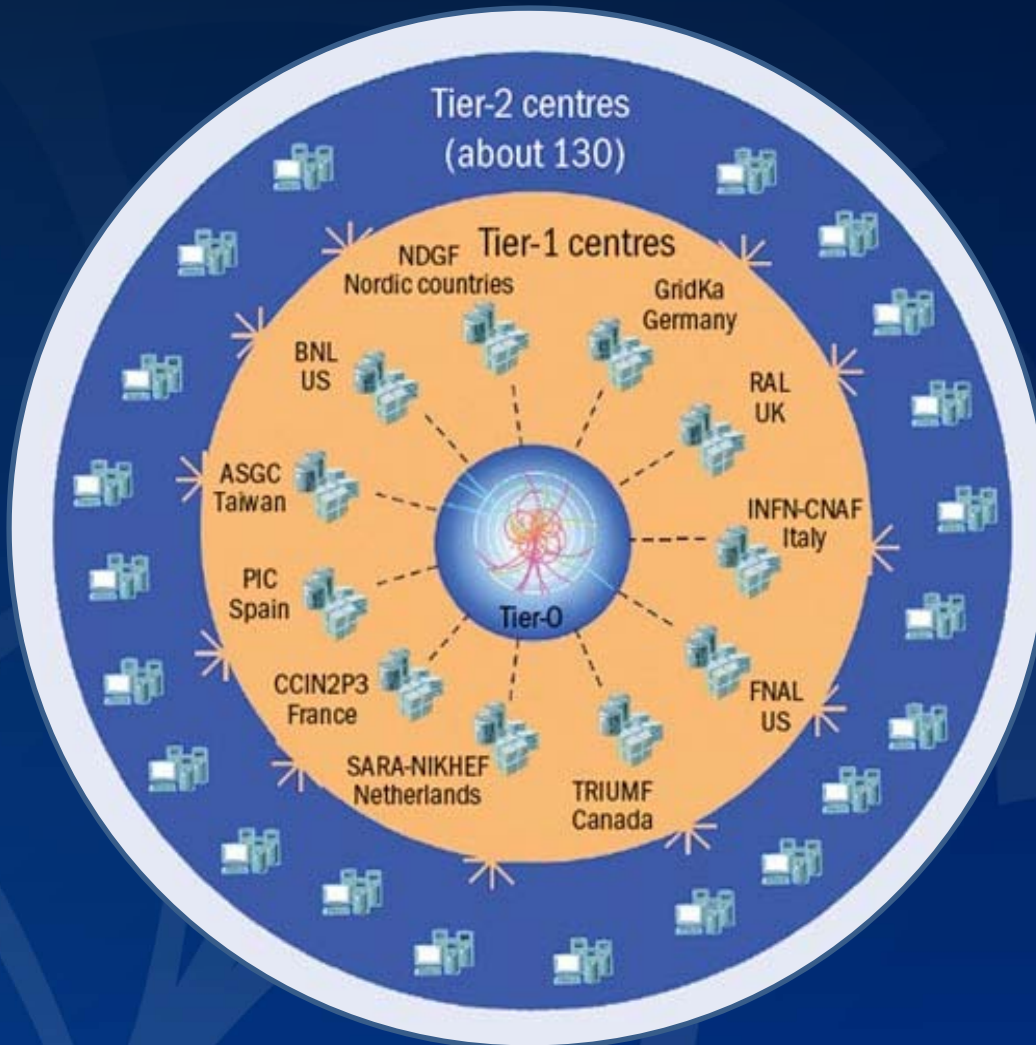
# Collaboration on big data and computing

## The Worldwide LHC Computing Grid

**Tier-0 (CERN):** data recording, reconstruction and distribution

**Tier-1:** permanent storage, re-processing, analysis

**Tier-2:** Simulation, end-user analysis



nearly 160 sites

~250'000 cores

173 PB of storage

> 2 million jobs/day



# Challenges in computing

## Big(ger) Data

- LHC upgrades
- New paradigms, science

## Exascale

- Computing evolution
- Next-gen interconnect

## Society

- Scientific leadership
- Sustainable computing

# Big(ger) data

Data rates at the LHC to increase by ~100x



“Sustainable computing”



# PART 2: COMPUTING LANDSCAPE

# Co-design



Design software  
independently  
of the hardware

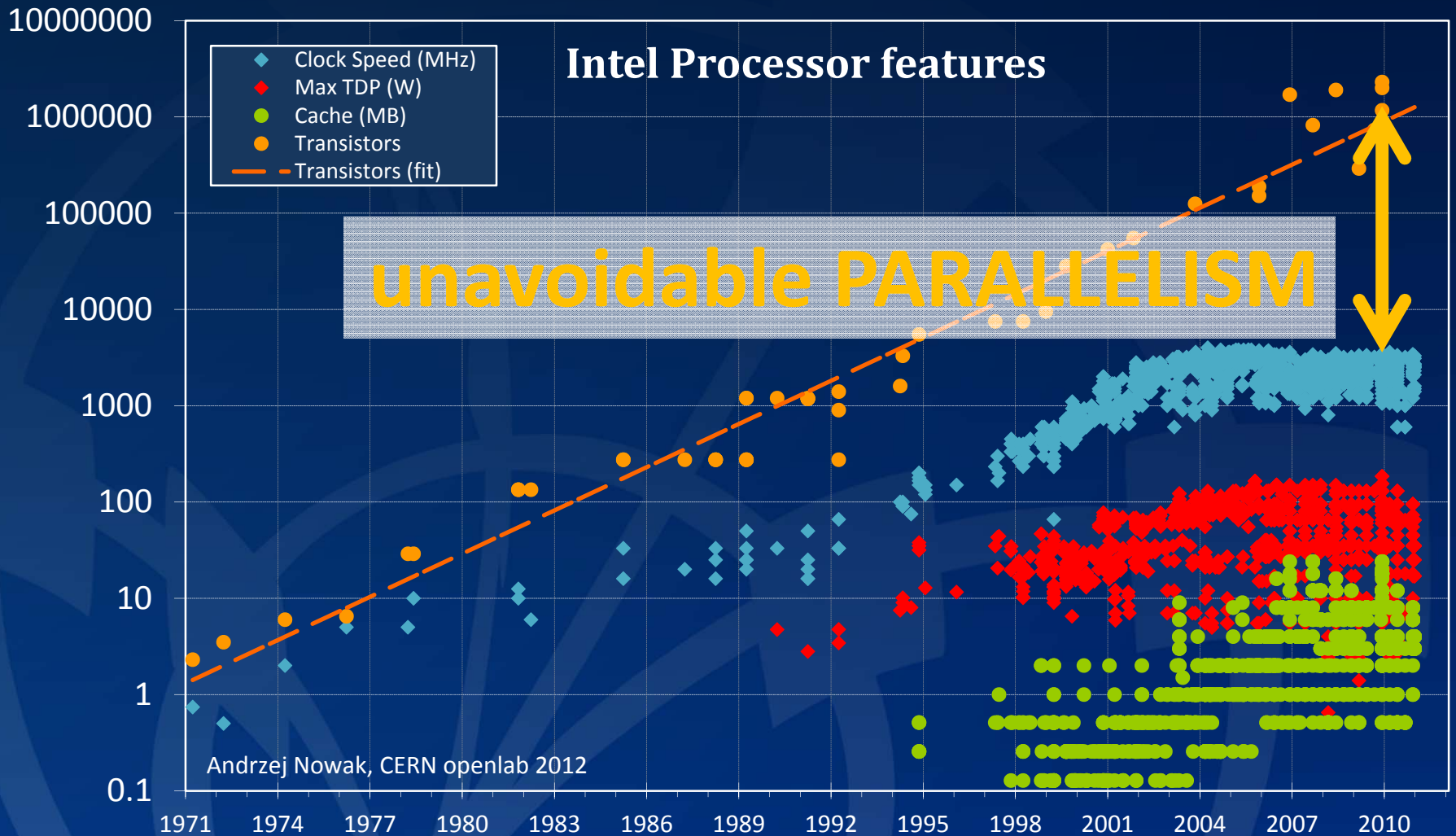


Optimize  
software for the  
hardware

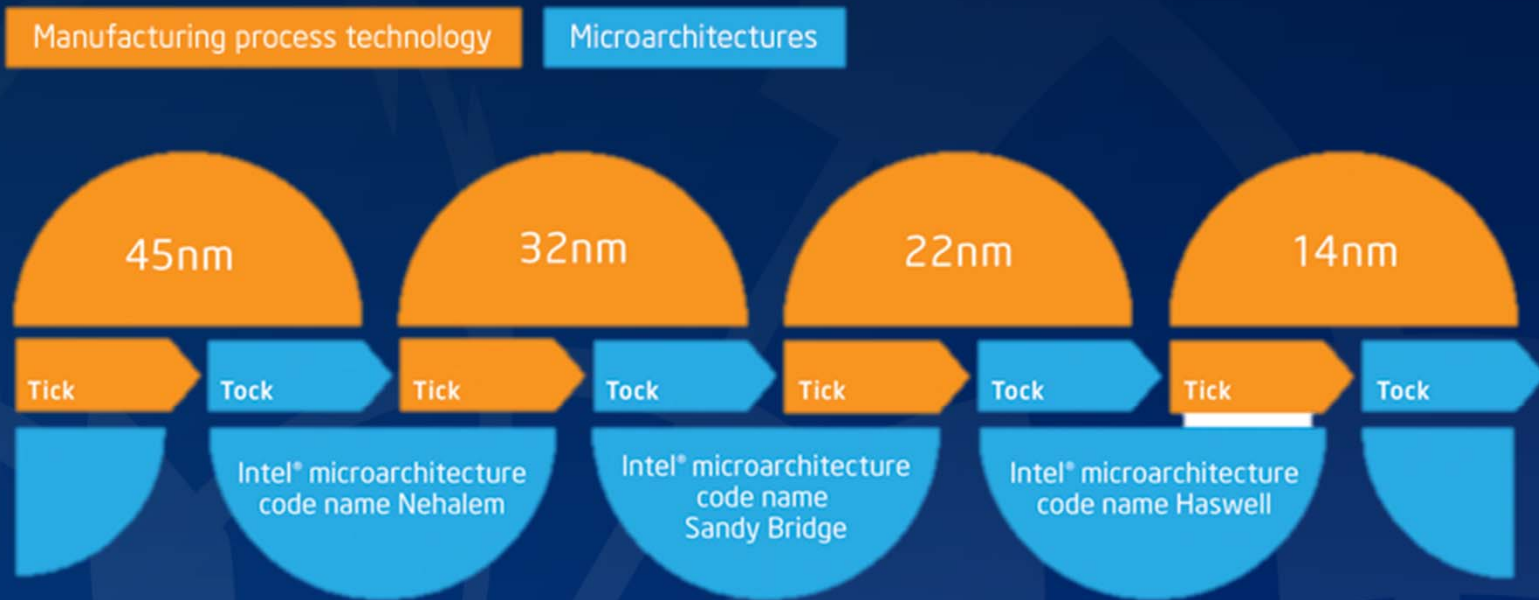




# Hardware landscape



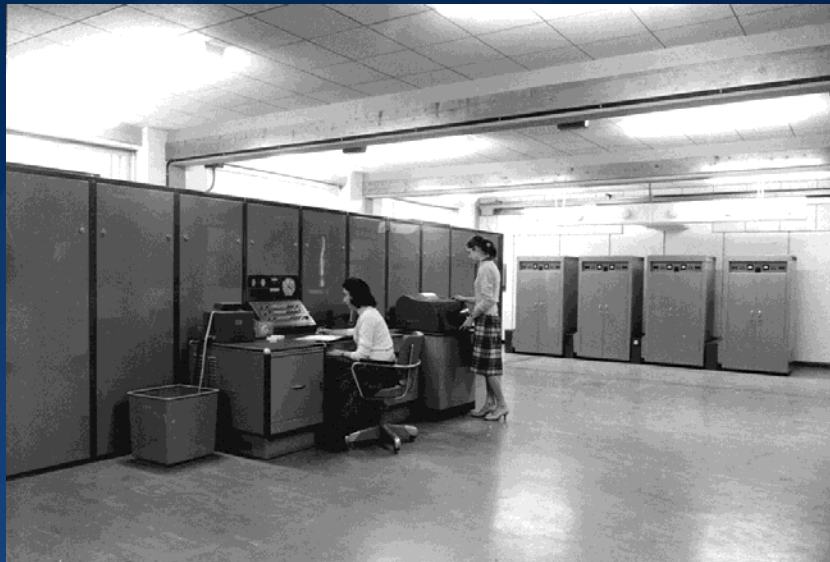
# The Intel tick-tock model



Source: Intel

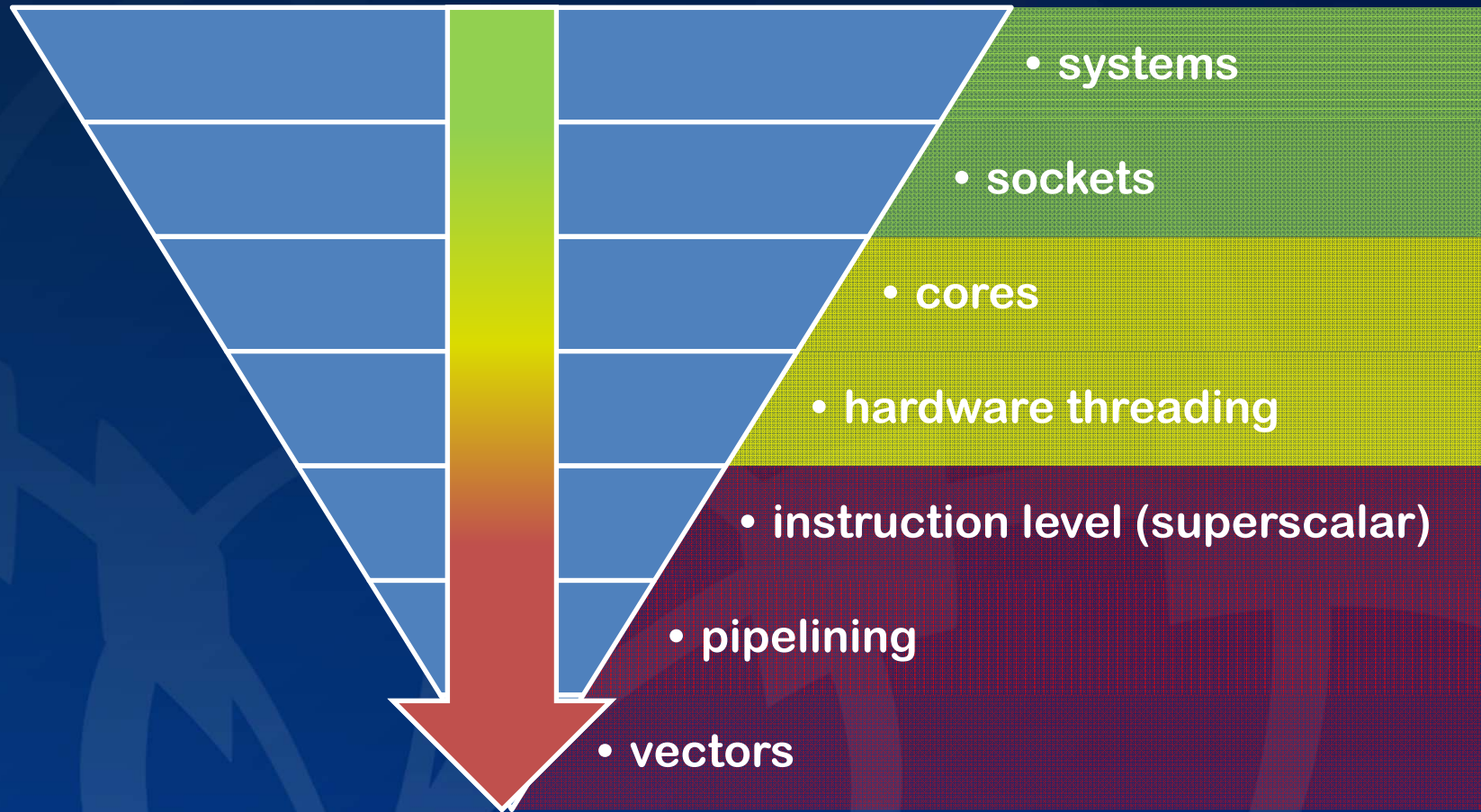


# A “modern” CPU that really is archaic

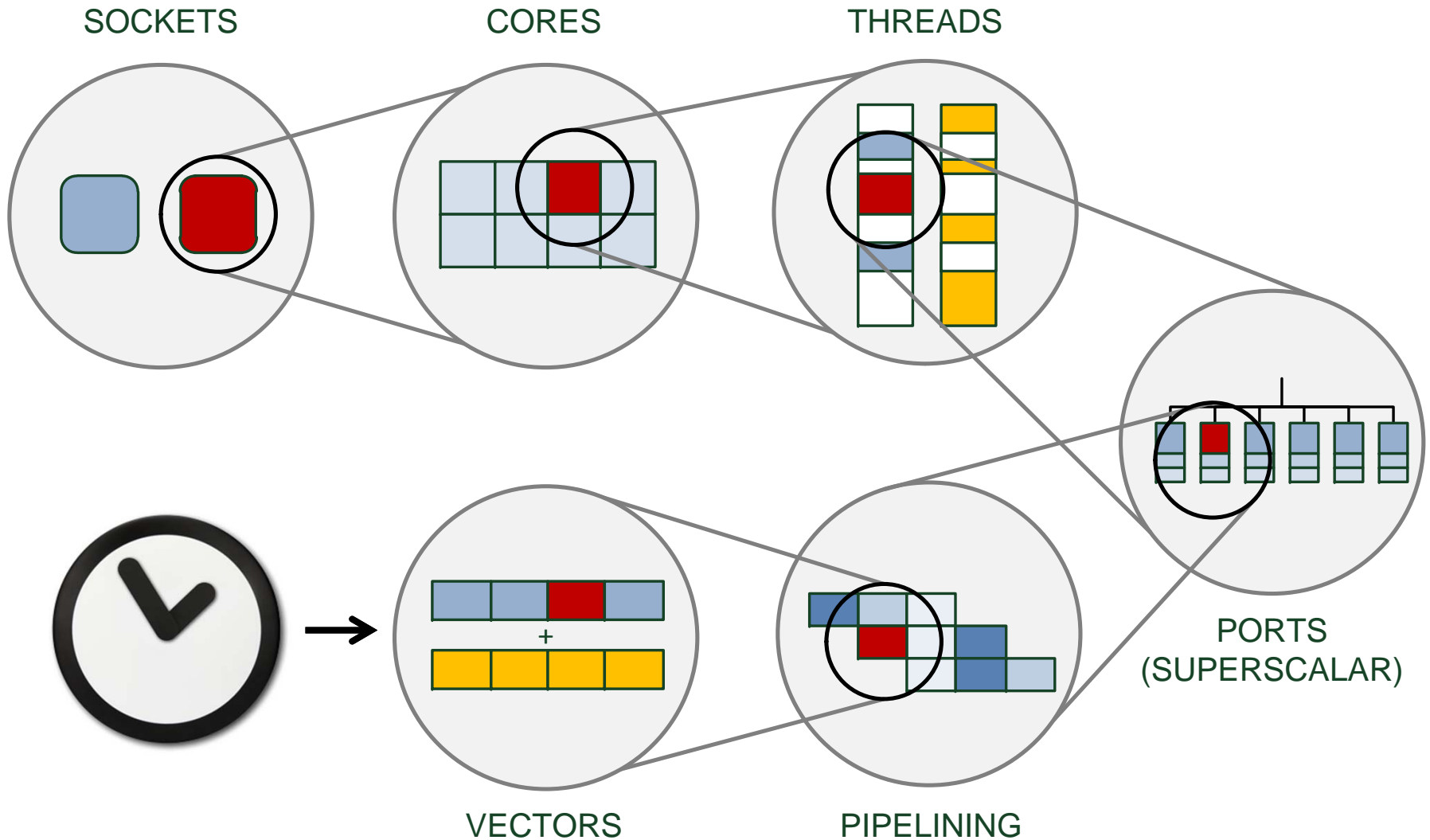


- As “stupid” as 50 years ago
- Still based on the Von Neumann architecture
- Primitive “machine language”
- **Ferranti Mercury:**
  - Floating-point calculations:  
Add: 3 cycles; Multiply: 5 cycles
- **Today:**
  - Programming for performance is the same headache as in the past

# Omnipresent multiplicative parallelism



# Inside a modern PC platform

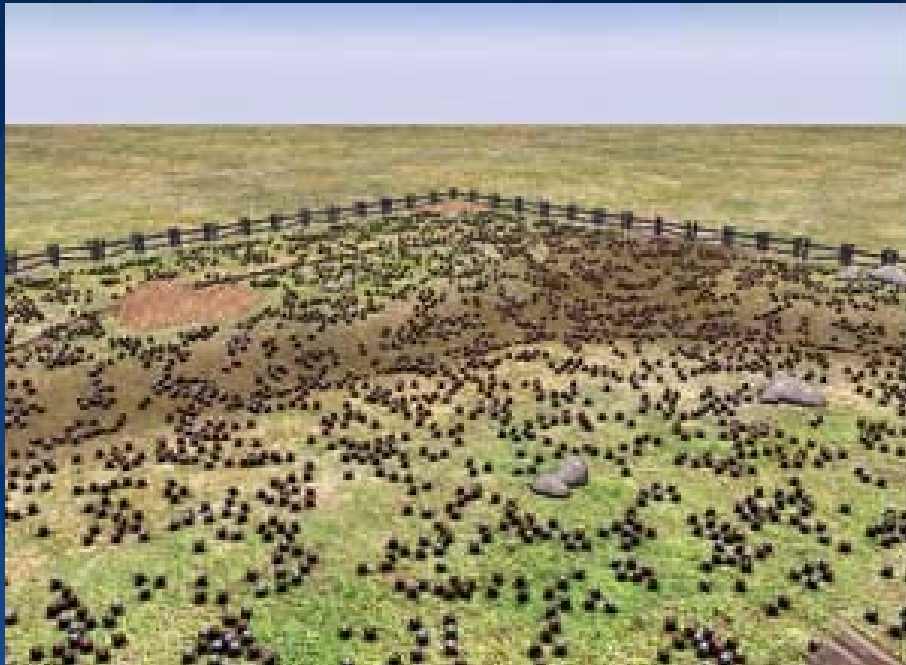




# Medium-term hardware trends

- Pricing follows market pressure, not technology
- IO, disk and memory do not progress at the same rate as compute power
  - bytes/FLOP decreasing
  - pJ/FLOP decreasing
- Bulk of improvements in x86 comes from Moore's Law still being in effect
- Heterogeneous architectures – cross platform, cross socket, hybrid CPUs, accelerators, split into throughput and classic computing

# Vector computing comes back with a vengeance



# Vectors

- **Comeback with a vengeance but lessons learned 20 years ago: growing substantially**
  - 128-bit SSE → 256-bit AVX (designed for more)
  - AVX: new execution units
  - LRBni (Intel MIC): 512 bits, new vector instructions, FMA, 3-4op
- **Good news:**
  - can now hold 4 doubles
  - only one architecture to worry about
  - plenty of technologies to choose from
  - compilers getting increasingly better at autovectorization (can get 2x)
- **Bad news:**
  - increasingly a key element in the performance equation
  - not everything will vectorize
  - iterative and auto-vectorization are promoted, but are not the “magic bullet” solution for many legacy workloads
  - good vectorization requires a data centric design (sacrifices have to be made)
  - bad past experience (before PCs)

The Intel AVX logo is displayed in a blue rectangular box. It features the text "Intel® AVX" in white, with a stylized white arrow pointing to the right.



# Intel64 & ILP considerations

- **x86 microarchitecture**
  - steady, but limited improvements (<10% per “tock”)
  - increasingly advanced features – can large code benefit?
- **Frequency – very modest changes, if any**
  - Rise of the Turbo boost
- **CPI for large code is often too high, literally wasting CPU power**
  - CPI figures for the major experiments hover around 0.9-1.5
- **C++/OOD abuse will produce significant side-effects**
  - Very frequent jumps and calls + more
  - Dynamic code has penalties – x86 is already quite good at executing it but there are limits
- **Pipelining not discussed explicitly as it folds into ILP**

# Hardware threading

- **Sharing of some CPU resources**
  - Little on-die overhead
  - The OS makes few distinctions between threads and cores
- **A part of the solution to bad ILP**
- **Free money**
  - Usually between +20% and +30%
  - Recommendation: use whenever possible, good for simulation
- **Does not help in all cases – depends on the bottleneck**

# Multi-core vs. many-core

- **A typical modern workhorse machine has 12-16 cores**
  - Some have 48 or more (AMD)
  - Major datacenters can be 1 or 2 generations behind
- **# of cores “at home” grows arithmetically**
  - various reasons, most linked to the way people use their computers
- **# of cores in the enterprise space still grows geometrically (per platform)**
- **The number of cores in the datacenter grows between the two, will slow down in the long run**
  - The trend is important, not the end amount
  - Is the trend sustainable? What about all these transistors?



# Multiple sockets and systems

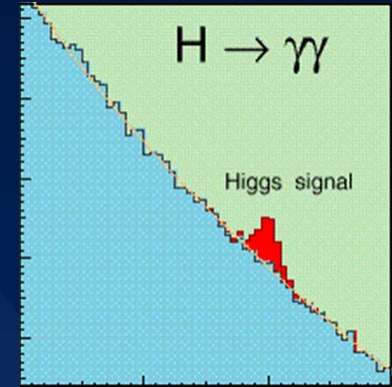
- **Sockets** – slight growth with a limit, ultimately impacts core count per platform
- **Multi-machine**
  - mostly HPC
  - HTC: independent machines and processes
- **Many-core is not multi-core**
  - Memory hierarchy issues pop up
    - Cache coherency
    - NUMA
    - Memory bandwidth or IO paths may be constraining
  - Strong scalability tanks (need weakly scalable workloads)
- **Multiprocess is a convenient model that can do the job, but it is not sustainable nor scalable**

# Corollary

**Raw platform performance is expanding  
in multiple dimensions simultaneously**

# The CERN case: physics jobs

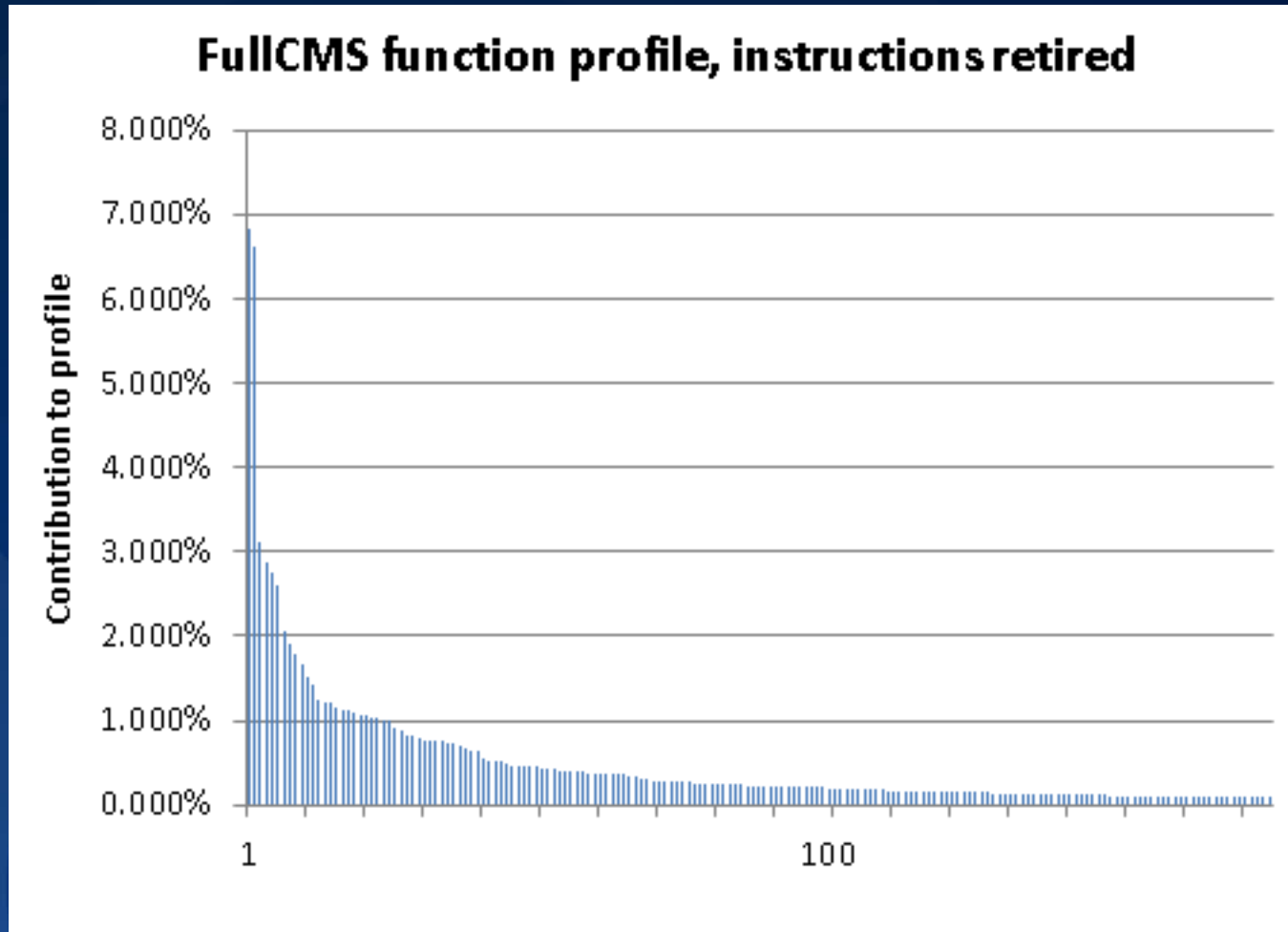
- **Independent events (collisions of particles)**
  - trivial (read: pleasant) parallel processing
- **Bulk of the data is read-only**
- **Very large aggregate requirements:**
  - computation, data, input/output
- **Chaotic workload**
  - research environment - physics extracted by iterative analysis:  
Unpredictable, Unlimited demand
- **Compute power scales with combination of SPECint and SPECfp**
  - Good double-precision floating-point (10%-20% of total) is important!
  - Good transcendental math libraries needed
- **Key foundation: Linux together with GNU C++ compiler**



From S. Jarp



# Large jobs – profile fragmentation



# Where is High Energy Physics code now?

- **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 won't fit accelerators in its current shape**
- **Intensive upgrade efforts underway**

# How does High Energy Physics use hardware now?

- **Very limited vectorization**
  - Bad conditions to vectorize
- **Sub-optimal instruction level parallelism (CPI at >1)**
- **Hardware threading introduced**
  - Memory constraints
- **Cores used well through multiprocessing – bar the stiff memory requirements**
  - However, systems put in production with tender related delays
- **Sockets used well**
- **Multiple systems used well**
- **Relying on in-core improvements and # cores for scaling**



# Where are we now?

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

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

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

\_%

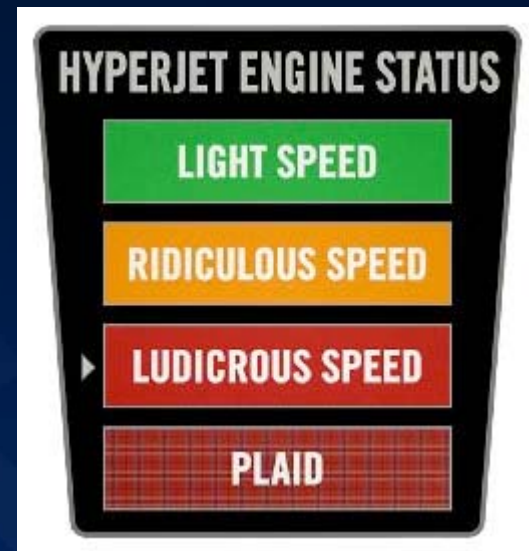
Write your  
percentage here



# Corollary

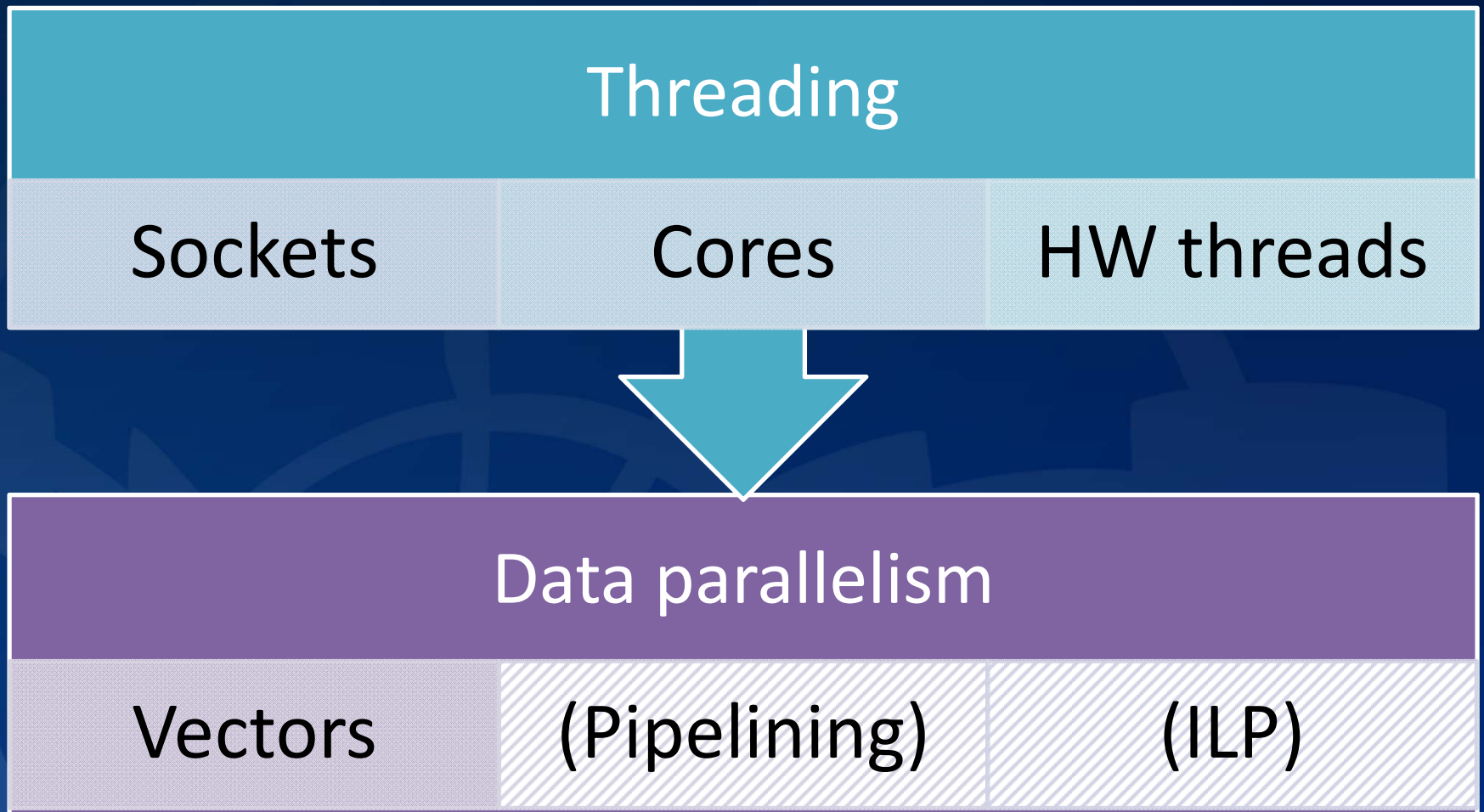
**Need to program for tomorrow's  
hardware today**



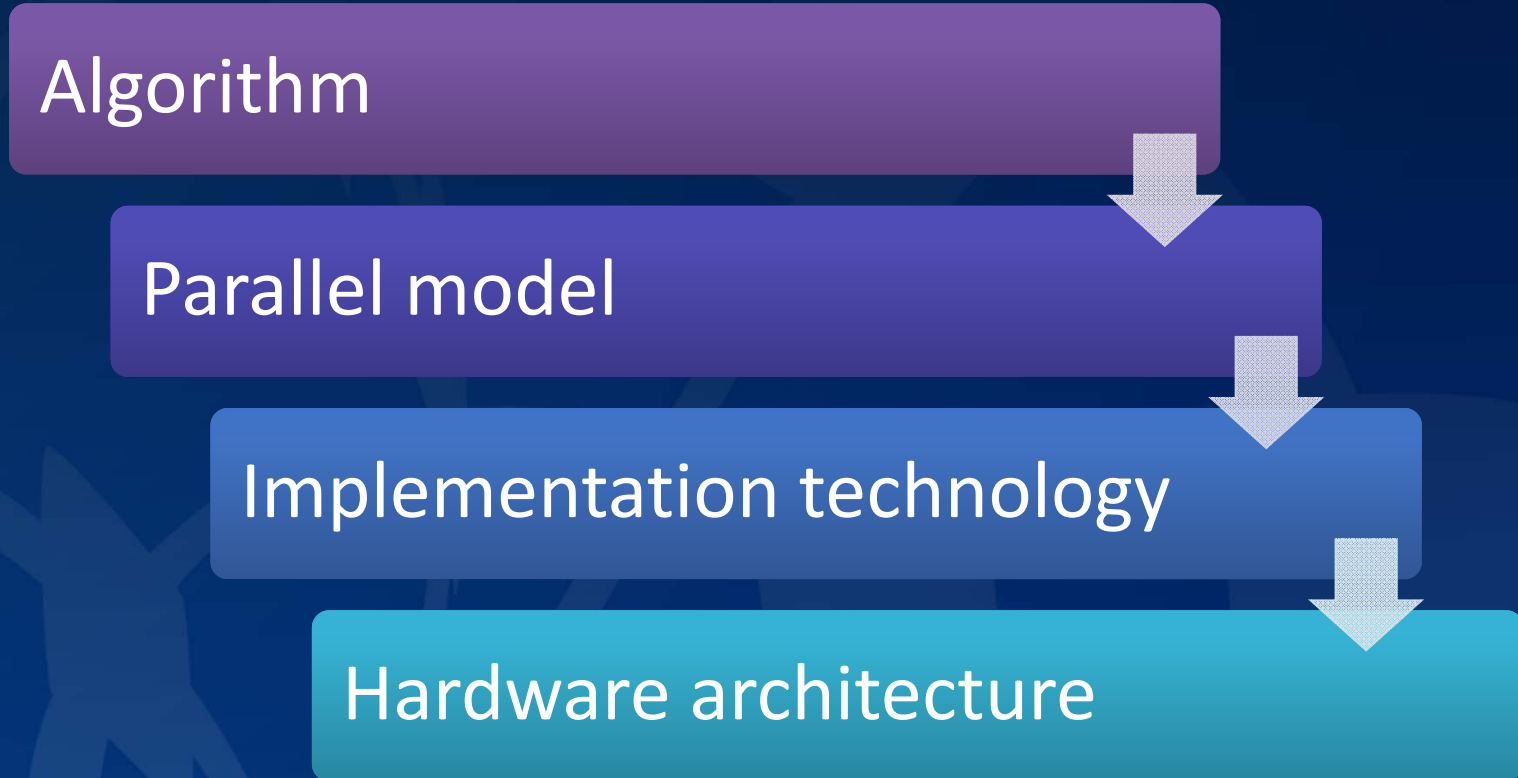


# PART 3: EFFICIENT CODE

# SW performance dimensions

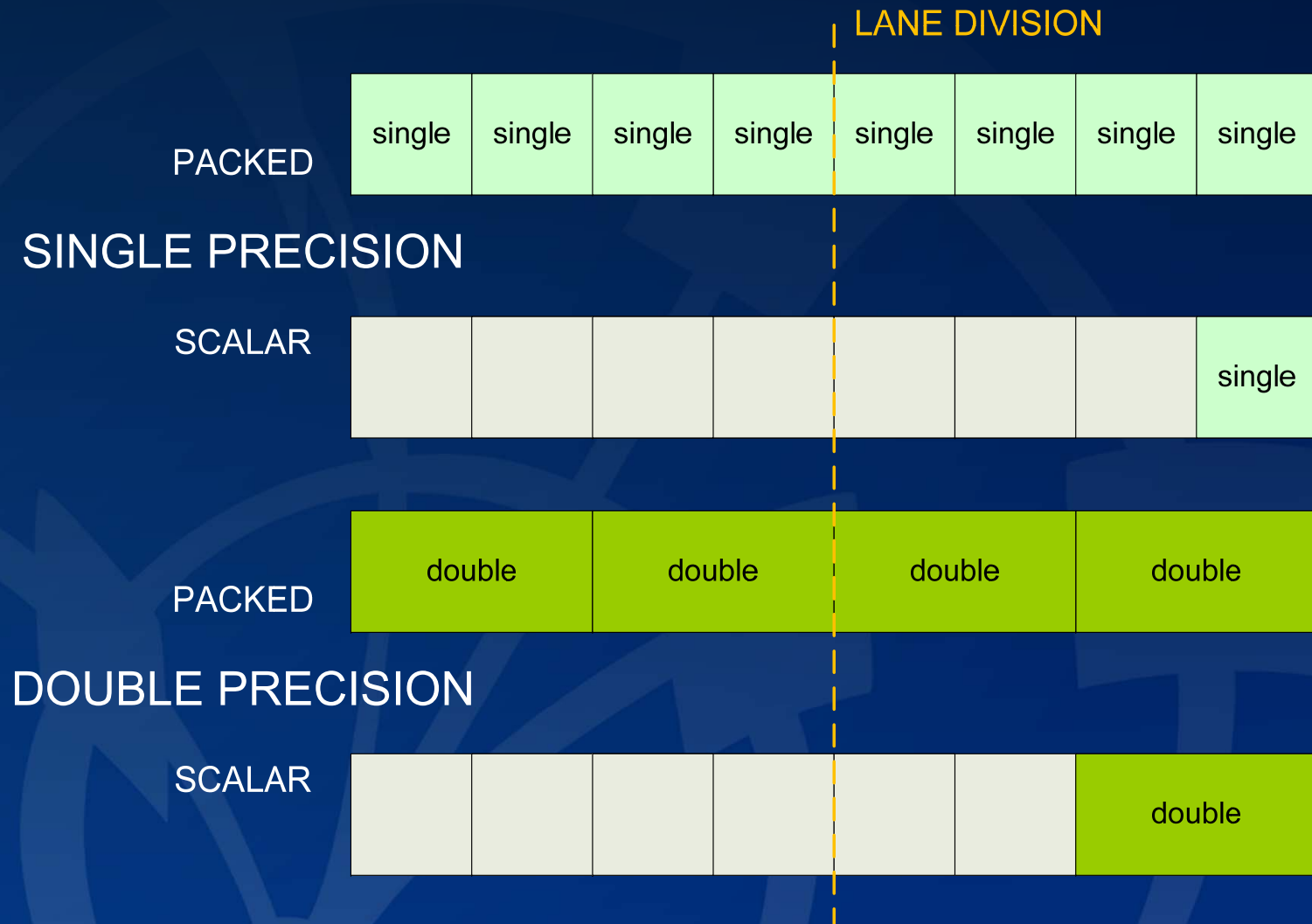


# The parallel technology stack





# AVX data layout



# Programming with vectors

- **Classical tradeoff: code manageability vs. speed**
- **AoS vs. SoA**
  - SoA favored
- **Available levels of abstraction**
  - Assembly
  - Intrinsic (C/C++)
  - Auto-vectorization (C/C++)
  - High level and interpreted code
- **Data-centric design is key**

# Auto-vectorization and vector notations

- Compilers are becoming increasingly effective in auto-vectorization
  - ICC leading the effort, GCC is behind but pushing forward
  - Numerous caveats and dependencies: need to align data, use pragmas, restrict keywords etc.
- Array notations gaining popularity (CEAN etc)

## Example: FIR Scalar Code

```
for (i=0; i<M-K; i++){  
    s = 0  
    for (j=0; j<K; j++){  
        s+= x[i+j] * c[j];  
    }  
    y[i] = s;  
}
```

## Example: FIR Inner Loop Vector

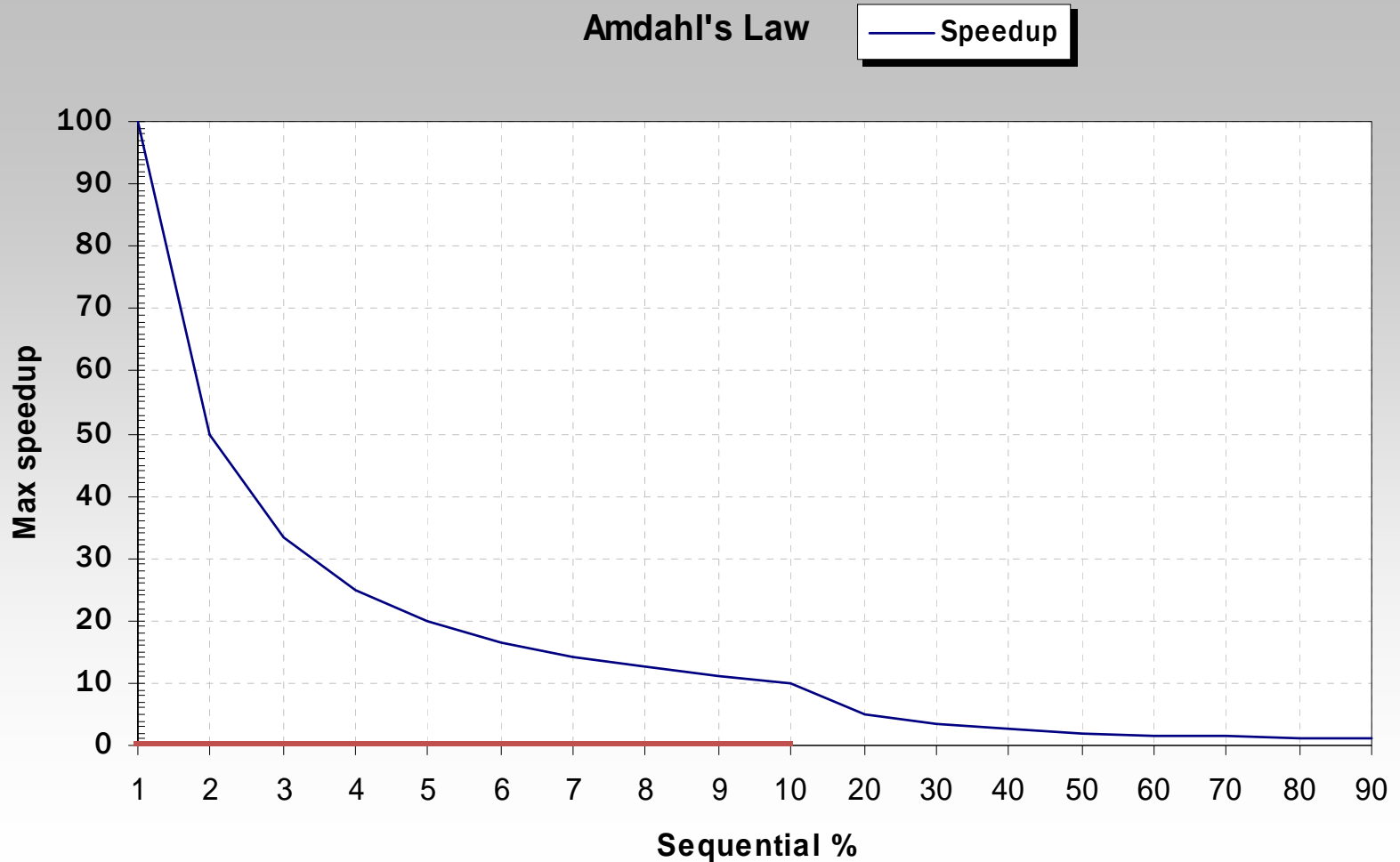
```
for (i=0; i<M-K; i++){  
    y[i] = __sec_reduce_add(x[i:K] * c[0:K]);  
}
```

## Example: FIR Outer Loop Vector

```
y[0:M-K] = 0;  
for (j=0; j<K; j++){  
    y[0:M-K] += x[j:M-K] * c[j];  
}
```



# Amdahl's Law



# Gustafson's Law

- Often when the problem size increases, the sequential portion remains constant
- Therefore, as the problem size increases, so do the opportunities for parallelization
- Let  $a(n)$  be the sequential portion function of the program, diminishing as  $n$  approaches infinity

$$\text{Speedup} = a(n) + N(1 - a(n))$$

- As  $n$  approaches infinity, the speedup approaches the number of processors  $N$

# A plethora of options for parallelism (subset)

- OpenMP
- CUDA
- pthreads
- MPI
- Cilk
- Ct/RapidMind/ArBB
- TBB
- OpenCL
- Boost threads
- Concurrent Collections
- Less mainstream:
  - Axum
  - Co-array Fortran
  - UPC
  - Go
  - Chapel
  - Fortress
  - X10
  - Erlang
  - Linda
  - Haskell

# The Hype Cycle

Peak of Inflated Expectations

Plateau of Productivity

Slope of Enlightenment

Trough of Disillusionment

Technology Trigger

Modeled after Gartner Inc.



# Tradeoffs in software development (with focus on hardware)

- **Flexibility and programmability vs. performance**
  - Impacts the choice of the programming language, technologies etc
- **Revamp vs. iterative improvement**
- **Homogeneous vs. heterogeneous processing model**
- **Single/multi process vs. multi-threaded**
- **Data-centric software design or not?**
- **Kernels vs. heavy code**
- **Program for specific architectures or not?**

# Key aspects in any choice



# C++ specific issues

- **Key performance related aspects of the C++ language:**
  - the compiler is particularly important
  - memory allocation
  - virtual mechanisms
  - small and temporary objects
- **Large projects in C++ produce extreme optimization challenges**
  - Highly fragmented profiles, few instructions/jump, few instructions/call
  - Difficult without collaboration from domain experts

# STL Container Efficiency

With material from A. Lazzaro

Container	Stores	Overhead	[ ]	Iterators	Insert	Erase	Find	Sort
list	T	8	n/a	Bidirect'l	C	C	N	N log N
deque	T	12	C	Random	C at begin or end; else N/2	C at begin or end; else N	N	N log N
vector	T	0	C	Random	C at end; else N	C at end; else N	N	N log N
set	T, Key	12	n/a	Bidirect'l	log N	log N	log N	C
multiset	T, Key	12	n/a	Bidirect'l	log N	d log (N+d)	log N	C
map	Pair, Key	16	log N	Bidirect'l	log N	log N	log N	C
multimap	Pair, Key	16	n/a	Bidirect'l	log N	d log (N+d)	log N	C
stack	T	n/a	n/a	n/a	C	C	n/a	n/a
queue	T	n/a	n/a	n/a	C	C	n/a	n/a
priority_queue	T	n/a	n/a	n/a	log N	log N	n/a	n/a
slist	T	4	n/a	Forward	C	C	N	N log N



# Other C++ tips (1)

- **Avoid virtual functions and classes – extra memory, indirections, compiler optimizations prevented**
  - Use C++ templates where possible
- **dynamic\_cast can be expensive**
- **Conditions are evaluated from left to right**
- **Switch statements prevent branch prediction**
  - Avoid unless absolutely necessary
- **Use STL with caution**

With material from A. Lazzaro

# Other C/C++ tips (2)

- **Dynamic memory allocation will suffer from fragmentation**
  - Use pools
  - Pass arguments by reference
  - Reuse objects instead of creating new ones
  - Avoid temporary variables
- **Data locality matters**
- **Code locality matters in large code**
- **Align data for vectorization**
- **Pay attention to floating point math**
- **Always help the compiler**

With material from A. Lazzaro

# The complexity of a large software project

- Strategy and hardware-related requirements are a must when the hardware is a variable
- Hard to plan for unknowns, but easier to plan for changes
- Requirements management
- Software middle-men threaten scalability
- Long time to produce and stabilize
  - Consequently: faraway targets should be considered, not current

# PART 4: OPTIMIZATION



# Performance optimization is:

Fair  
benchmarking

- Workload characterization

Problem  
identification

- Performance monitoring

Bottleneck  
elimination

- Performance tuning

# Benchmarking tips

- **Be objective**
- **Be in control**
  - Especially: PIN YOUR JOBS (CPU and memory)
- **Choose representative, stable, correct benchmarks**
- **Choose good metrics**
  - Throughput, latency, scalability, etc...
- **Repeatability**
- **Keep a log**

# Benchmarking control

- **Benchmarking a modern PC has become a complex issue:**
  - CPU frequency
  - Number of cores
  - Number and configuration of sockets
  - Vector and floating point usage
  - Cache
  - Memory size and layout
  - BIOS and firmware version
  - Hardware threading on or off
  - Turbo mode on or off
  - Power consumption
  - Virtualization
  - Operating system version, kernel, libraries
  - Compiler version and flags
  - Pinning to cores and to NUMA memory

# 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%

# Performance monitoring

Program  
execution



Analysis



Actionable  
information



Collection





# Measuring performance

- **The most common performance measurement unit is time**
  - Wall clock time – “how long do I have to wait for it to be done?”
  - CPU time – “for how long is the computer busy?”
  - Latency – “how long do I have to wait to get an answer?”
  - Throughput – “how much of X in a period of time?”
- **Minimizing time/latency is not the same as maximizing throughput**
  - and vice versa – i.e. see Amdahl’s and Gustafson’s laws
- **1 second: 9,192,631,770 periods of specific radiation of  $^{133}\text{Cs}$** 
  - 1 second is just 1 event, composed of individual events (oscillations)
- **What if there could be a different “event” to define performance? Or a whole set of them?**

# An OS abuses your trust

- 100% CPU utilization in “top”? Awesome. Aren't we done here?
- Maybe not:
  - Spinlocks
  - Pointless iterations
  - Floating point side effects
  - Memory traffic
  - etc etc
- Use performance counters to obtain an accurate image

# Performance monitoring in hardware

- Most modern CPUs are able to provide real-time statistics concerning executed instructions via a Performance Monitoring Unit (PMU)
- The PMU is spying in real time on your application (and everything else that goes through the CPU)
- Limited number of “sentries” (counters) available, but they are versatile
- Counters monitor events as they happen
- Recorded occurrences are called samples or counts
- Typically on modern Intel CPUs:
  - 2-4 universal counters per HW thread: #0, #1 (#2, #3)
  - 3 specialized counters: #16, #17, #18
  - Additional 8 “uncore” counters

# Performance events

- **Many events in the CPU can be monitored**
  - A comprehensive list is dependent on the CPU and can be extracted from the manufacturer's manuals or from relevant tools
  - Examples: cache misses, instructions, cycles, loads, vector ops
- **On some CPUs (e.g. Intel Core), some events have bit-masks which limit their range, called “unit masks” or “umasks”**
  - Example: instructions retired: “ALL” or “only LOAD” or “only STORE”
- **Extensive information: Intel Manual 248966-023a**
  - Intel Manual 248966-023a “Intel 64 and IA-32 Architectures Optimization Reference Manual”
- **AMD CPU-specific manuals**
  - i.e. “BIOS and Kernel Developer's Guide for AMD Family 10h Processors” or “Software Optimization Guide For AMD Family 10h and 12h Processors”

# The CPI figure and its meaning

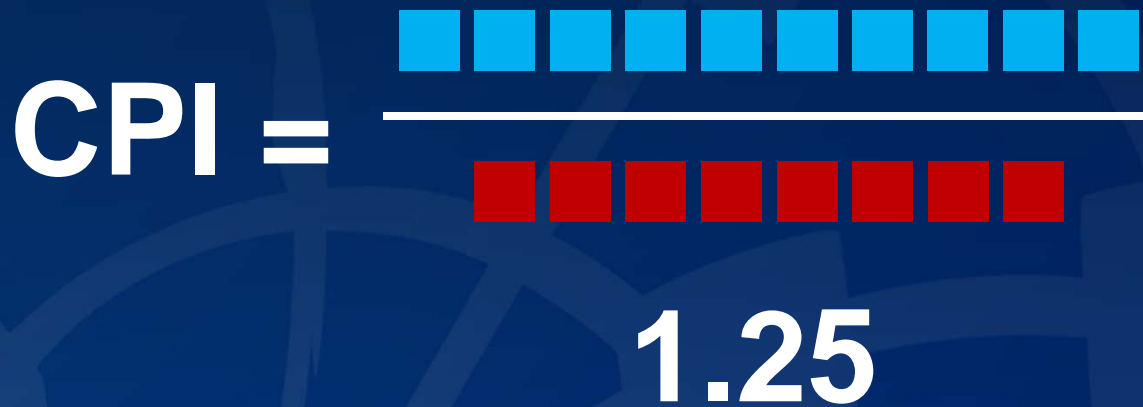
- **CPI – cycles per instruction**
  - Thanks to multiple execution ports (superscalar architecture), more than one instruction can be executed per cycle
  - In modern Intel CPUs, CPI can go as low as 0.25 = 4 instructions per cycle
  - CPI above 1.0 is generally not impressive
- **The ratio of the number of CPU cycles spent on a program to the number of program instructions retired by the CPU**
  - CYCLES / INSTRUCTIONS
- **Lower CPI often means better efficiency**
  - This figure illustrates the CPU usage efficiency in an indirect way, and, like all ratios, can be tricky to interpret



# Simple CPI demo

$$\text{CPI} = \frac{\text{cycles}}{\text{instructions}}$$

1.25

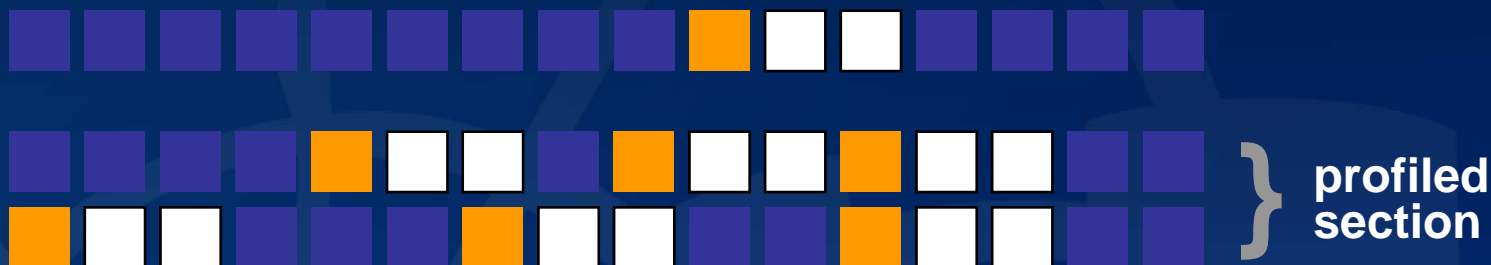
A diagram illustrating the calculation of CPI. It shows a horizontal line with 10 blue squares above it and 8 red squares below it. The text 'CPI =' is to the left of the line, and '1.25' is centered below the red squares.

 cycles

 instructions

# Simple cache miss demo

- 50 cycles of work (incl. L1 consultations/misses)
- 50 cycles of work with one L2 cache miss
- 50 cycles of no work



Assuming 20% of the instructions are loads and 3% of L2 misses...

**~35% cycles wasted, program runs ~60% slower!**

# False Sharing







File Help

Am | [Icons] | [Icons]

r000hs

# Hotspots - Hotspots

Intel VTune Amplifier XE 2011

Analysis Target | Analysis Type | Collection Log | Summary | **Bottom-up** | Top-down Tree

/Task Type /Function /Call Stack	CPU Time	Module	Function (Full)
Event loop	95.099s		
Initialization	4.960s		
[Outside any task]	0.220s		
Selected 1 row(s):		95.099s	

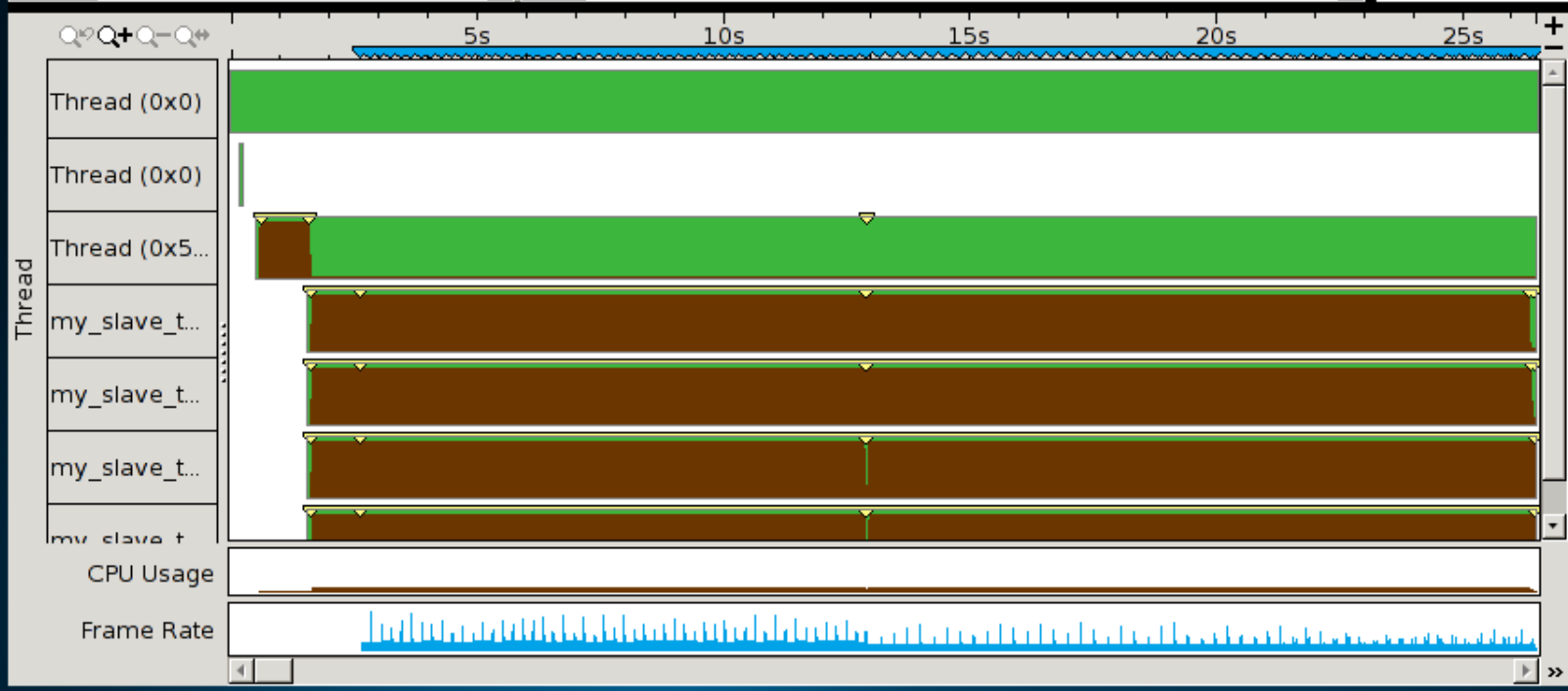
Task stack

1 stack(s) selected. Viewing 1 of 1

Current stack is 100.0% of selection

100.0% (95.099s of 95.099s)

test40!ParRunManager::DoEventLo..



### Ruler Area

- Frames
- Thread
  - Running
  - CPU Time
  - User Tasks
- CPU Usage
  - CPU Time
- Frame Rate
  - Frame Rate

No filters are applied. Module: [All] Thread: [All] Call Stack Mode: Only user functions



# Google GOODA

GOoDA Visualizer

GOoDA Visualizer

file:///Users/vitillo/Dropbox/gooda-visualizer/index.html#report=atlas\_reco\_cg

### Generic Optimization Data Analyzer GUI

atlas\_reco\_cg Hotspots

Cycles Samples

process path	module path	unhalted_core_cycles	br_inst_ret...	ear_return	function_sources	function_targets
		196450 (100%)	467913	7483705	7483705	
[-] athena.py		191816 (100%)	461322	7378950	7378950	
	libCaloEvent.so	14867 (100%)	71535	1382740	1132088	
	libCLHEP-Matrix-1.9.4.7.so	4637 (100%)	40059	589095	610775	
	ld-2.12.so	5218 (100%)	37767	625809	322026	
	libtcmalloc_minimal.so	12180 (100%)	5767	408070	389066	

Cycles Samples

function name	offset	length	
[-] ElementLink, ForwardIndex...	0x4f110	0x107	libCaloE...
[-] CLHEP::HepSymMatrix::num_...	0x1cf50	0x4	libCLHEP...
[-] operator new(unsigned lon...	0x134b0	0x3da	libtcmal...
[-] CLHEP::operator*(CLHEP::H...	0x19070	0x23d	libCLHEP...
[-] operator delete(void*)	0x12c10	0x2da	libtcmal...
[-] std::_Rb_tree_increment(s...	0x69c00	0x5a	libstdc+
[-] CaloEnergyCluster::getCel...	0x59360	0x9b	libCaloE...
[-] CaloCellContainer::findIn...	0x6dec0	0x99	libCaloE...

Help

# Optimization tips (1)

- **Unimpressive CPI / stalls**
  - Doing too many operations?
  - Large latency instructions in the code?
  - Using vector instructions?
  - Do a stall analysis to see where and why you're stalling
- **Cache misses, false sharing**
  - Memory access characteristics
  - Data structures and their layout
  - Does your program fit in the cache?
  - Help the hardware prefetcher!
  - Do a cache analysis to see which data (and where) is not serviced properly

# Optimization tips (2)

- **Many mispredicted branches**
  - Is there a way to restructure the code?
  - Is there a way to make the “ifs” more predictable?
  - Rearranging conditions and loops
  - Too many jumps / function calls?
  - Sample with branch events (e.g. in perfmon2 or PTU) to locate offending pieces of code
- **Excessive floating point operations**
  - Does everything need to be calculated?
  - Are you running in loops?
  - Could some results be reused?
  - Do you really need that much precision?

# Additional tips

- **Pay special attention to memory access patterns**
  - How does your programming language, compiler and allocator lay out your structures and variables? Is it better to leak or to “free()”?
  - What is the temporal and spatial locality of your data?
    - Temporal locality: accesses to the same data in a short time frame
    - Spatial locality: accesses to nearby data in a short time frame
  - Consider your memory usage model
    - Data set
    - Data organization
    - Data access patterns
  - This area will only grow in complexity in the upcoming years
- **The Pareto (80/20) principle**
  - Sometimes adapted to “90/10”
  - Might suggest that improving 20% of code will give you 80% of your results
  - Might also suggest that 90% of the time is spent in 10% of the code – Is it true for large applications with their C++ fragmentation and large codebases?

# Relating to design

- **Choose best available algorithms**
  - Re-implementing an algorithm is usually the last thing you will want to do when “it” finally works
- **Choose the appropriate programming language**
  - Managed and VM languages are not performance friendly; optimizing properly is often impossible
  - Common tradeoff: an object oriented language values form over performance, and C or Fortran the other way around
- **Make informed decisions. The optimal method of doing something isn't always the fastest one, but you need to know why**



# Common misconceptions

- **On hardware**
  - Myth: “Performance is a hardware issue”
  - Reality: See the slides on Moore’s Law
- **On responsibility**
  - Myth: “Performance should be handled by the compiler and libraries”
  - Reality: no compiler and libraries will fix poorly written code (remember the elephant)
- **On premature optimization**
  - Myth: “Premature optimization is the root of all evil”
  - Reality: an often misunderstood quote, referring to not optimizing bottlenecks that are not yet apparent – not to “any optimization” as a whole
- **On “GOTO programming”**
  - Myth: GOTO programming is entirely bad
  - Reality: It’s a tradeoff – see the assembly “jmp” instruction

# True pitfalls

- **The rule of diminishing returns**
  - The amount of benefits that you can get with a relatively small effort is limited; each additional investment will yield less results
- **Not knowing your limits and not knowing when to stop**
- **Over-optimization can do damage**
  - Especially if you optimize for a certain processor family: consider placing highly optimized routines in *CPU-specific libraries*
- **Double-check your results**
  - Flukes aren't common – they're frequent! Consider this talk for an idea of the hardware complexity you have to deal with
- **Wrong or overlapping optimizations can do damage**
  - Optimize when the time is right, but design with optimization in mind
  - Premature micro-optimizations (of unquantifiable benefit) can reduce the readability / comprehension / maintainability of the code and threaten correctness
  - DO NOT wait until the end of the project to optimize
- **Under-optimization is not worth the time**
  - Do it right – and remember that the penalty for abandoning the benefits of new platforms or techniques can be very high!

# Tuning summary

**Get the right tools for the job**

**Master the 7 performance dimensions**

**Scalable designs and high performance  
are friends**

# PART 5: FUTURE CHALLENGES

# The CERN openlab

## A unique research partnership of CERN and the industry

Objective: The advancement of cutting-edge computing solutions to be used by the worldwide LHC community

- Partners support manpower and equipment in dedicated competence centers
- openlab delivers published research and evaluations based on partners' solutions – in a very challenging setting
- Created robust hands-on training program in various computing topics, including international computing schools; Summer Student program
- Past involvement: Enterasys Networks, IBM, Voltaire, F-secure, Stonesoft, EDS; Future involvement: Huawei
- Now in phase IV: 2012-2014

<http://cern.ch/openlab>



PARTNERS



ORACLE

SIEMENS



# The Platform Competence Center

## Focus on efficient computing



## Close collaboration with the Physics department at CERN

# PCC - particular interests

- **Compute optimization**
  - Absolute, per CHF, per Watt
  - Optimization tools
- **Compilers**
- **Parallelization**
  - x86 compatible technologies spanning the whole range from OpenCL to MPI
- **Accelerators**
  - Intel MIC, limited interest in GPUs, combos
- **Storage**
- **Next phase V in 2015-2018 (Exascale era)**

# Intel MIC at openlab

## Early access

- Work since MIC alpha (under RS-NDA)
- ISA reviews in 2008

## Results

- 3 benchmarks ported from Xeon and delivering results: ROOT, Geant4, ALICE HLT trackfitter

## Expertise

- Understood and compared with Xeon
- **Post-launch dissemination**

# MTG4 on MIC – example profile

Function / Call Stack	CLK %	INST %
sqrt	14.35%	22.16%
exp	6.47%	9.47%
atan2	4.22%	6.31%
CLHEP::RanluxEngine::flat	3.24%	5.60%
G4ElasticHadrNucleusHE::HadronNucleusQ2_2	3.01%	2.41%
G4PhysicsVector::Value	2.76%	0.95%
log	2.22%	2.85%
G4VoxelNavigation::LevelLocate	2.05%	0.66%
G4VoxelNavigation::ComputeStep	1.64%	1.10%
G4ClassicalRK4::DumbStepper	1.59%	2.96%
G4SteppingManager::DefinePhysicalStepLength	1.54%	1.39%
G4Navigator::ComputeStep	1.40%	1.01%

# Teaching

- International computing schools
- Workshops
  - 10 workshops in 2012
  - >350 participants





# ICE-DIP

- EU Framework Program 7 project looking for (amongst other things) efficient methods of accelerator/co-processor use
- Focus on data taking past 2016
- Of particular interest
  - Getting data into the platform
  - Getting data into the accelerator/co-processor
  - Efficient processing
  - Efficient distribution of results
- What role for software?
- **Are you interested?**
  - We will employ 5 PhD students to work on Si Photonics, FPGAs, networks, many-core



# Possible future directions

- **Software replacing hardware**
  - Programmability replaces rigid structures
- **Intensive compute**
  - Local farms must have much higher processing capacity
- **Accelerators**
  - Experiments with Intel MIC and GPUs
- **Silicon photonics**



**Where will we be tomorrow?**

	SIMD	ILP	HW THREADS	CORES	SOCKETS
MAX	8	4	1.35	12	4
TYPICAL	6	1.57	1.25	10	2
HEP	1	0.80	1.25	8	2

	SIMD	ILP	HW THREADS	CORES	SOCKETS
MAX	8	32	43.2	518.4	2073.6
TYPICAL	6	9.43	11.79	117.86	235.71
HEP	1	0.8	1	8	16



# THANK YOU

## Q & A



Questions? [Andrzej.Nowak@cern.ch](mailto:Andrzej.Nowak@cern.ch)