Shanghai Many-Core Workshop

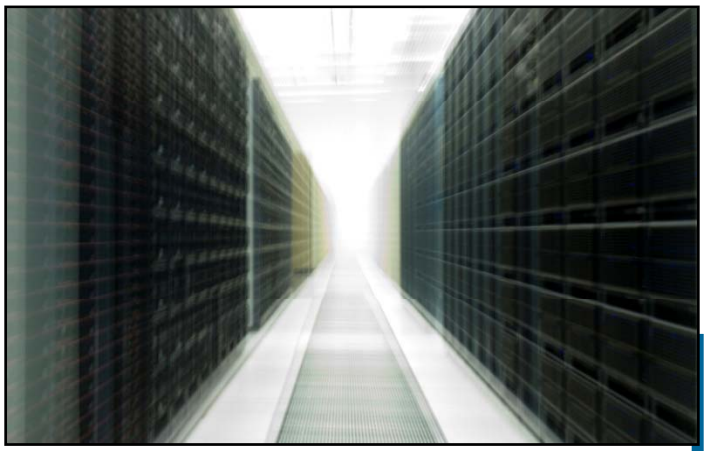
Parallel Processing Models and Research at CERN

“Capacity computing in seven dimensions”

Sverre Jarp

CERN openlab CTO

CERN



Gelato Presentation – 28 March 2008

Contents

- **Introduction**
  - CERN's new Collider is ready
- **The computing issues**
  - and, there are several
- **Possible remedies**
  - but, few are painless
- **Conclusions**

CERN openlab

Introduction

What is CERN?

- **CERN is the world's largest particle physics centre**
  - Particle physics is about:
    - Elementary particles: The constituents all matter in the Universe is made of
    - Fundamental forces which hold matter together

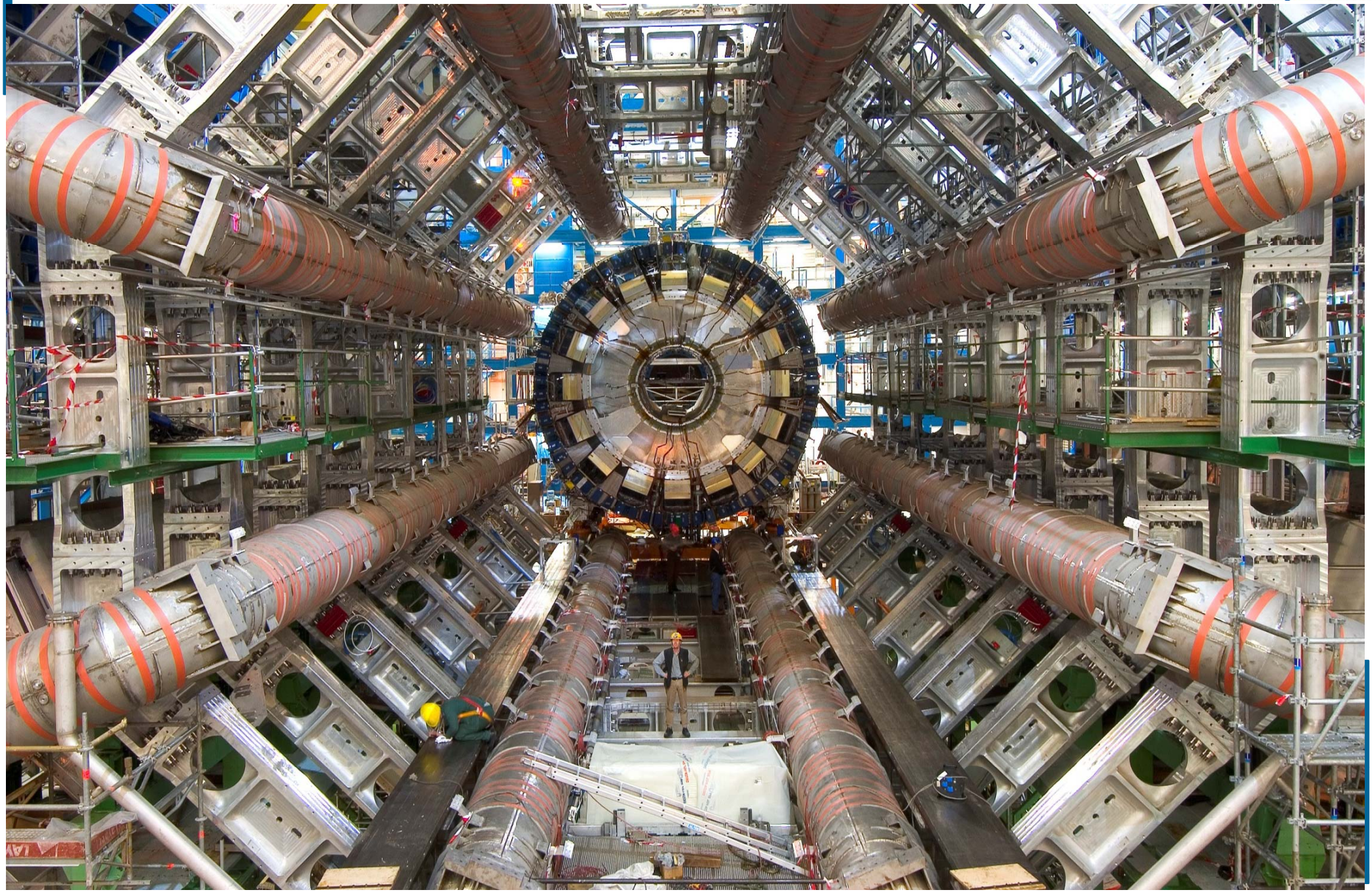
- **Particles physics requires:**
  - special tools to create and study new particles
    1) Accelerators
    2) Particle Detectors
    3) Powerful computers

The Large Hadron Collider (LHC)

- **The Large Hadron Collider will collide beams of protons at an energy of 14 TeV (in the late summer of 2008)**
- **Using the latest super-conducting technologies, it will operate at about – 271°C, just above the temperature of absolute zero.**
- **With its 27 km circumference, the accelerator will be the largest superconducting installation in the world.**

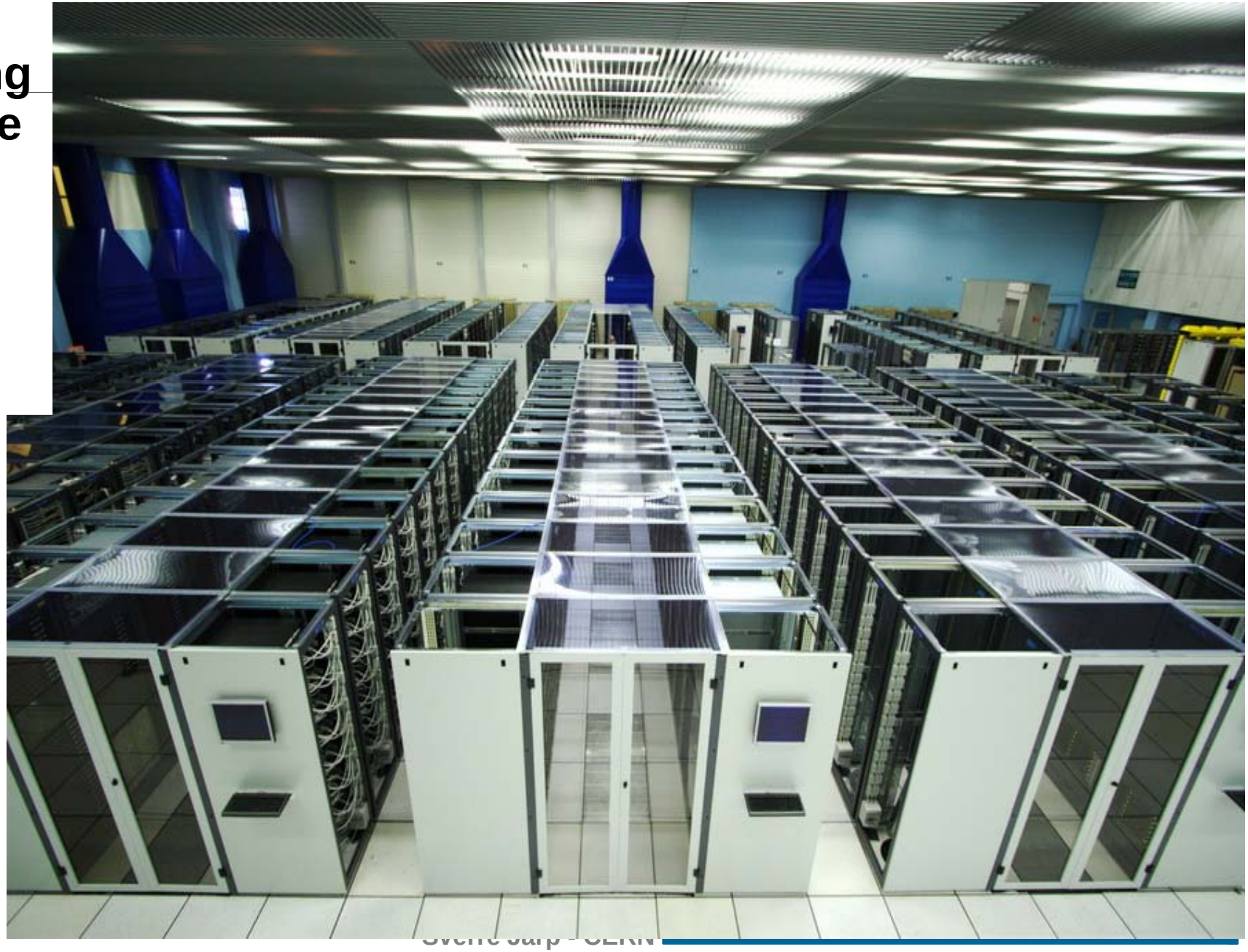
The ATLAS detector

CERN openlab



The Computer Centre and the Grid

- **In the Computing Centre, we are also ready!**



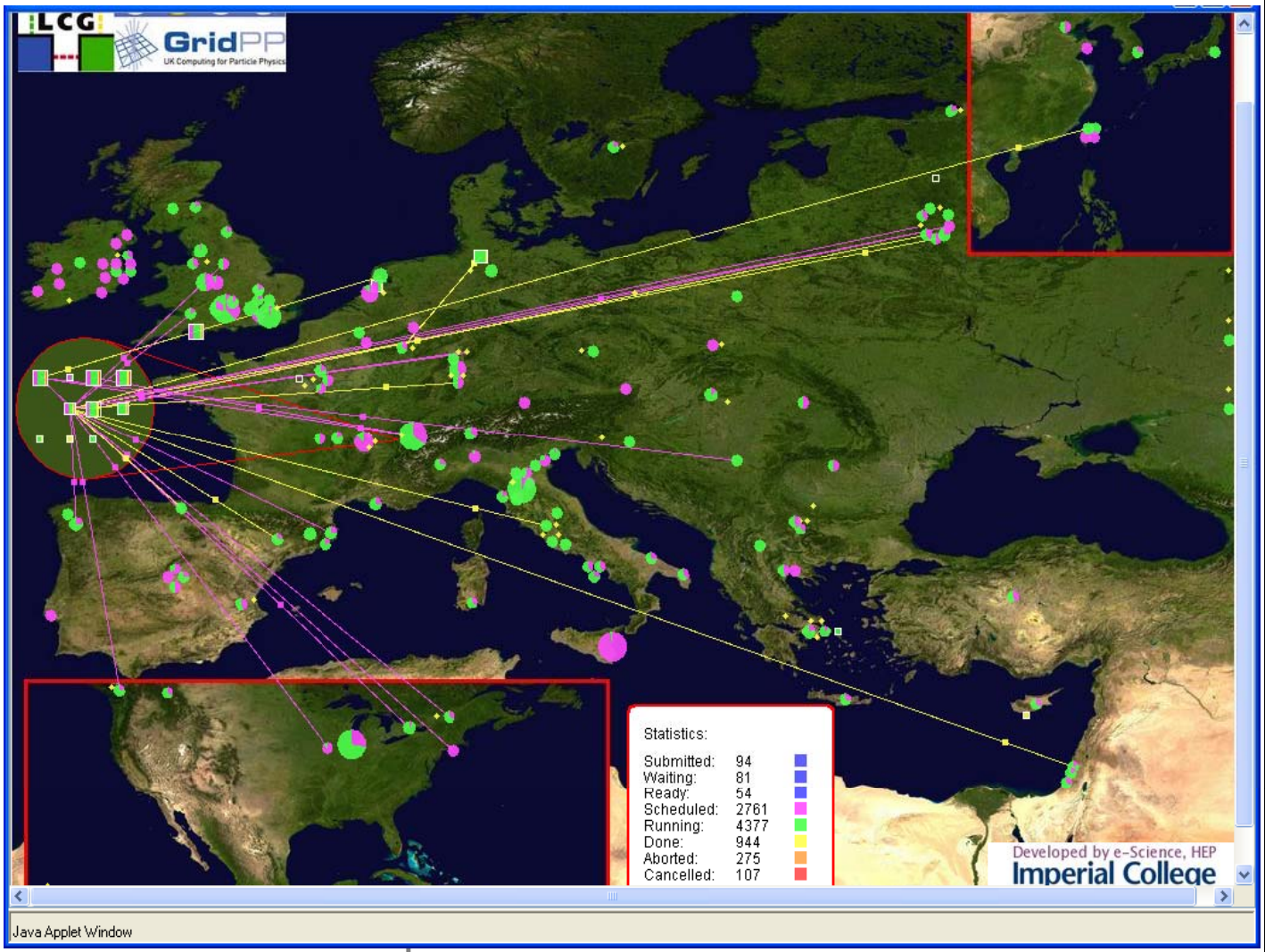
LHC Computing Grid

- **Largest Grid service in the world !**

• Almost 200 sites in 40 countries

• Tens of thousands of servers (w/Linux)

• Tens of petabytes of storage

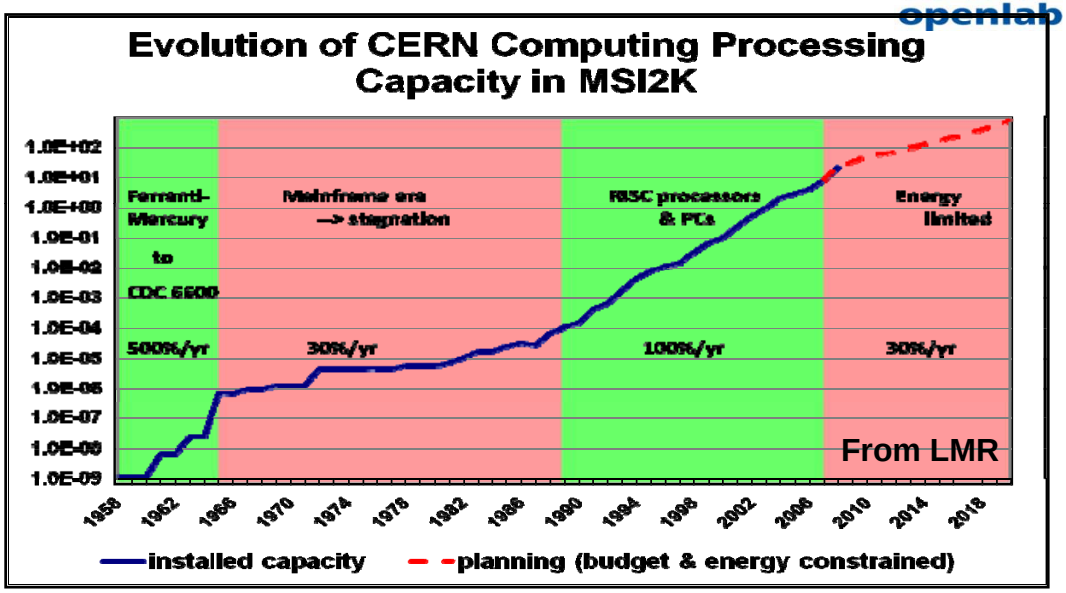


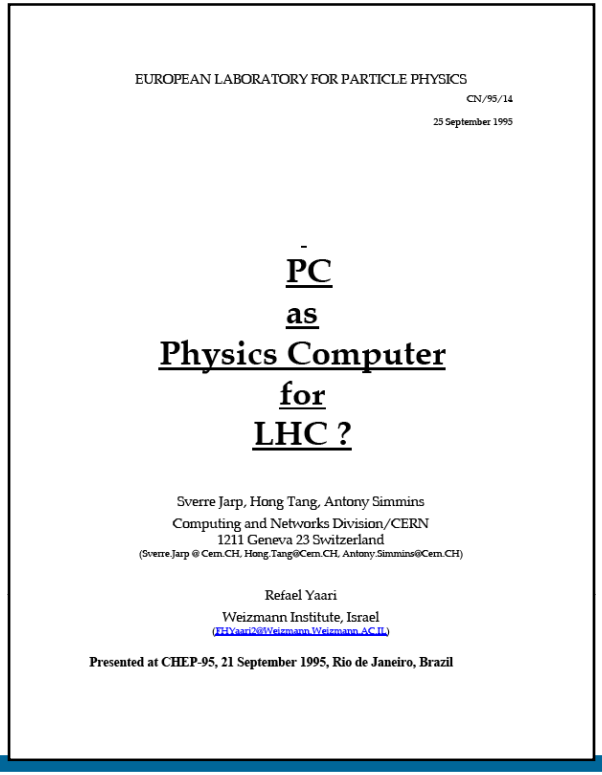
CERN openlab

The issues

Evolution of CERN's computing capacity

- **During LEP era (1989 – 2000):**
  - Doubling of compute power every year
  - Initiated with the move from mainframes to RISC systems
- **At an internal conference in 1995:**
  - With my colleagues, I made the first recommendation to move to PCs





EUROPEAN LABORATORY FOR PARTICLE PHYSICS

CN/95/14

25 September 1995

PC as Physics Computer for LHC ?

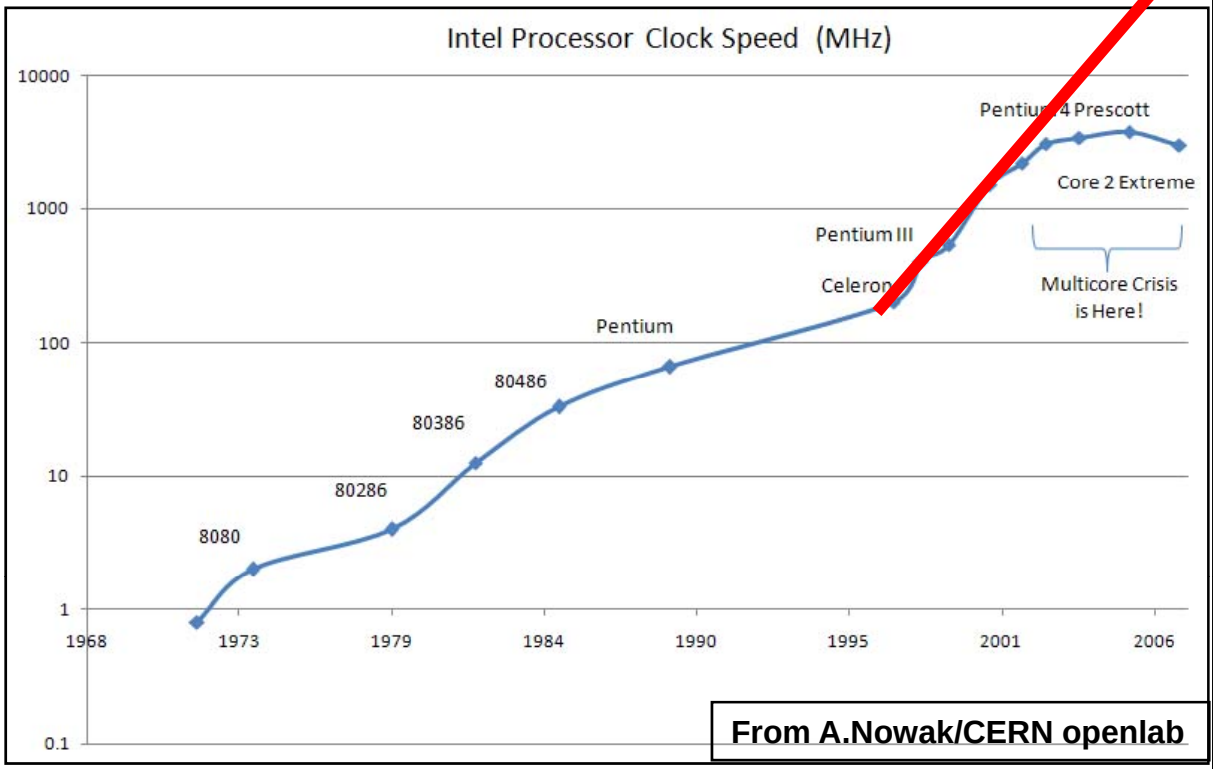
Sverre Jarp, Hong Tang, Antony Simmins
Computing and Networks Division/CERN
1211 Geneva 23 Switzerland
(Sverre.Jarp @ Cern.CH, Hong.Tang@Cern.CH, Antony.Simmins@Cern.CH)

Refael Yaari
Weizmann Institute, Israel
(FHYaari2@Weizmann.Weizmann.AC.IL)

Presented at CHEP-95, 21 September 1995, Rio de Janeiro, Brazil

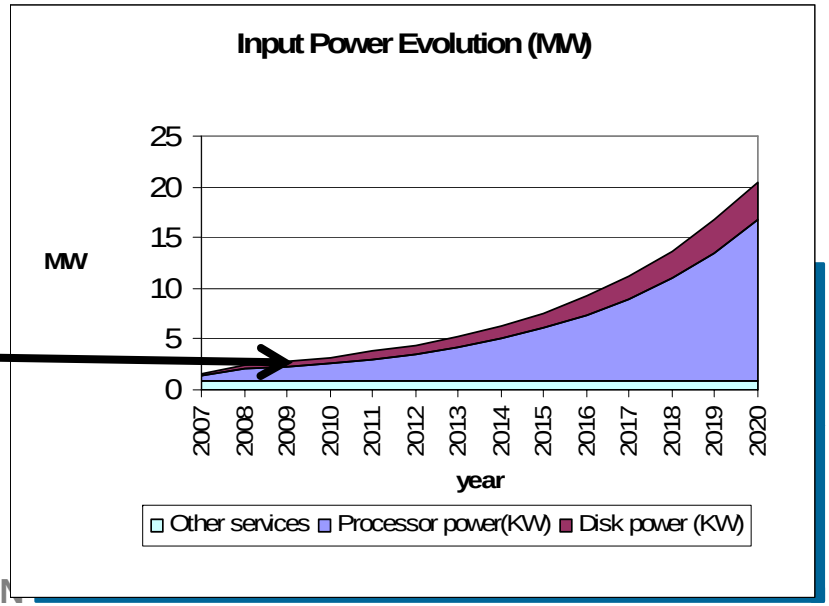
1) Frequency scaling is over!

- **The 7 “fat” years of frequency scaling in HEP**
  - Pentium Pro (1996): 150 MHz
  - Pentium 4 (2003): 3.8 GHz (~25x)
- **Since then**
  - Core 2 systems:
    - ~3 GHz
    - Quad-core



2) The Power Wall

- **The CERN Computer Centre can "only" supply 2.5MW of electric power**
  - Plus 2MW to remove the corresponding heat!
- **Spread over a complex infrastructure:**
  - CPU servers; Disk servers
  - Tape servers + robotic equipment
  - Database servers
  - Other infrastructure servers
    - AFS, LSF, Windows, Build, etc.
  - Network switches and routers
- **This limit will be reached in 2009!**

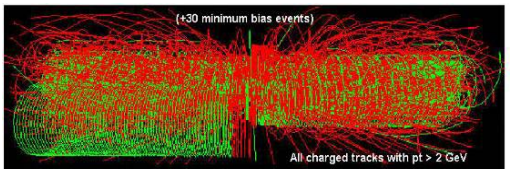


The move to many-core systems

- **Examples of process slots in servers**
- **Sockets * Cores * HW-Threads**
  - Today:
    - Dual-socket Intel quad-core (Harpertown):
      - 2 * 4 * 1 = 8
    - Dual-socket Sun Niagara (T2) processors w/8 cores and 8 threads
      - 2 * 8 * 8 = 128
  - Tomorrow:
    - Quad-socket Intel Nehalem “octocore” with dual threading
      - 4 * 8 * 2 = 64
    - Single-socket Larrabee
      - 1 * 24 * 4 = 96
- **In the near future: Hundreds of process slots!**

3) Our programming paradigm

- **Event-level parallelism has been used for decades**
  - Process event-by-event in a single process

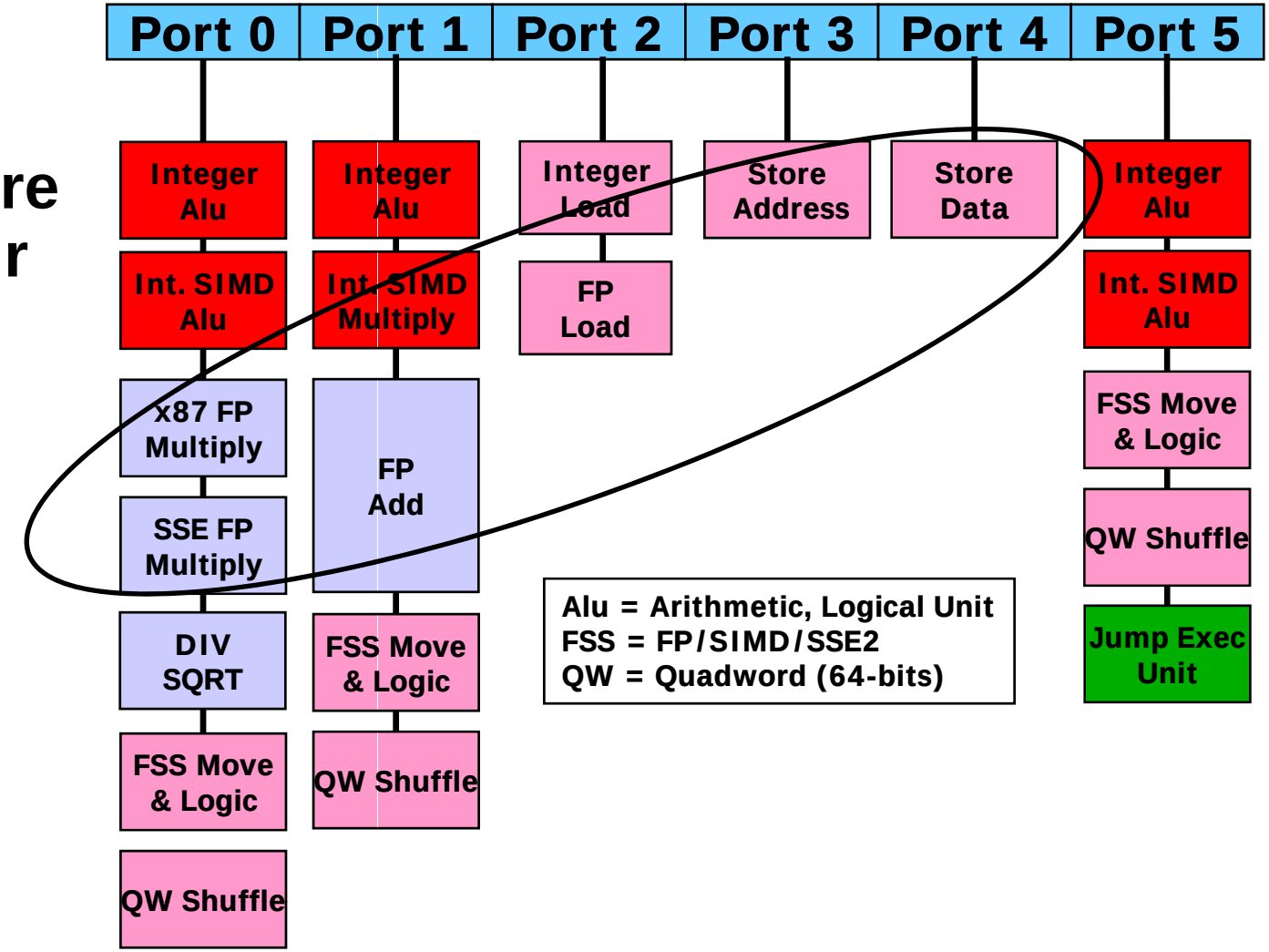


- **Advantage**
  - Large jobs can be split into N efficient processes, each responsible for processing M events
    - Built-in scalability
    - Great for “capacity computing” (high-throughput batch computing)

- **Disadvantage**
  - Memory must be made available to each process
    - With 2 – 4 GB per process
    - A dual-socket server with Quad-core processors
      - Needs 16 – 32 GB (or more) – we currently buy only 16!

Core 2 execution ports

- **Intel's new microarchitecture can execute four instructions in parallel:**



Issue ports in the Core 2 micro-architecture (from Intel Manual No. 248966-016)

15

4) HEP code density

- **Averages about 1 instruction per cycle.**
  - This “extreme” example shows even less:

High level C++ code →

```
if (abs(point[0] - origin[0]) > xhalfsz) return FALSE;
```

Assembler instructions →

```
movsd 16(%rsi), %xmm0
subsd 48(%rdi), %xmm0   // load & subtract
andpd _2il0floatpacket.1(%rip), %xmm0 // and with a mask
comisd 24(%rdi), %xmm0 // load and compare
jbe ..B5.3      # Prob 43%  // jump if FALSE
```

Cycle	Port 0	Port 1	Port 2	Port 3	Port 4	Port 5
1			load point[0]			
2			load origin[0]			
3						
4						
5						
6		subsd	load float-packet			
7						
8			load xhalfsz			
9						
10	andpd					
11						
12	comisd					
13						jbe

Same instructions laid out according to latencies on the Core 2 processor →

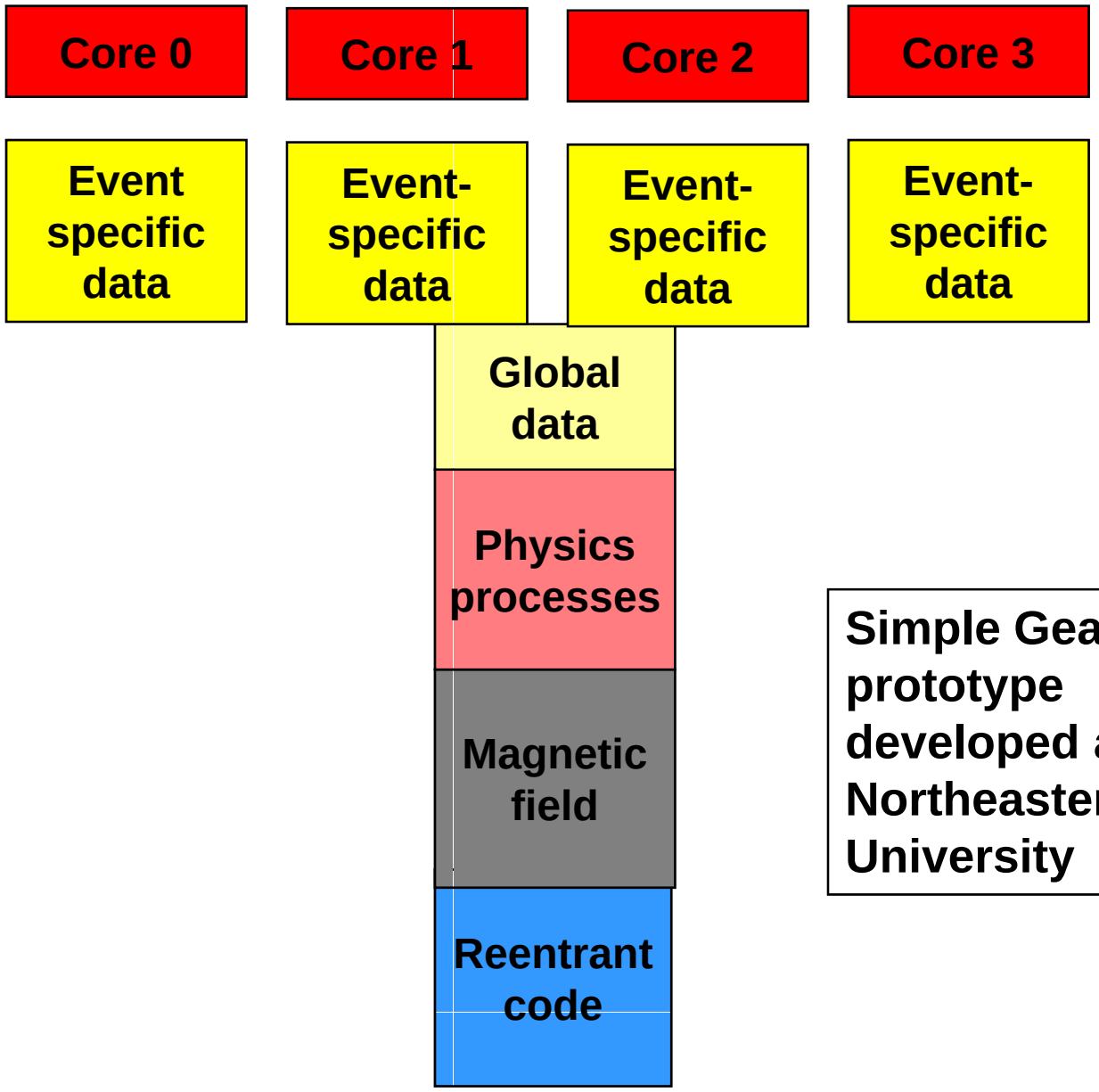
NB: Out-of-order scheduling not taken into account.

CERN openlab

Possible remedies

1) More efficient memory footprint

- **As follows:**



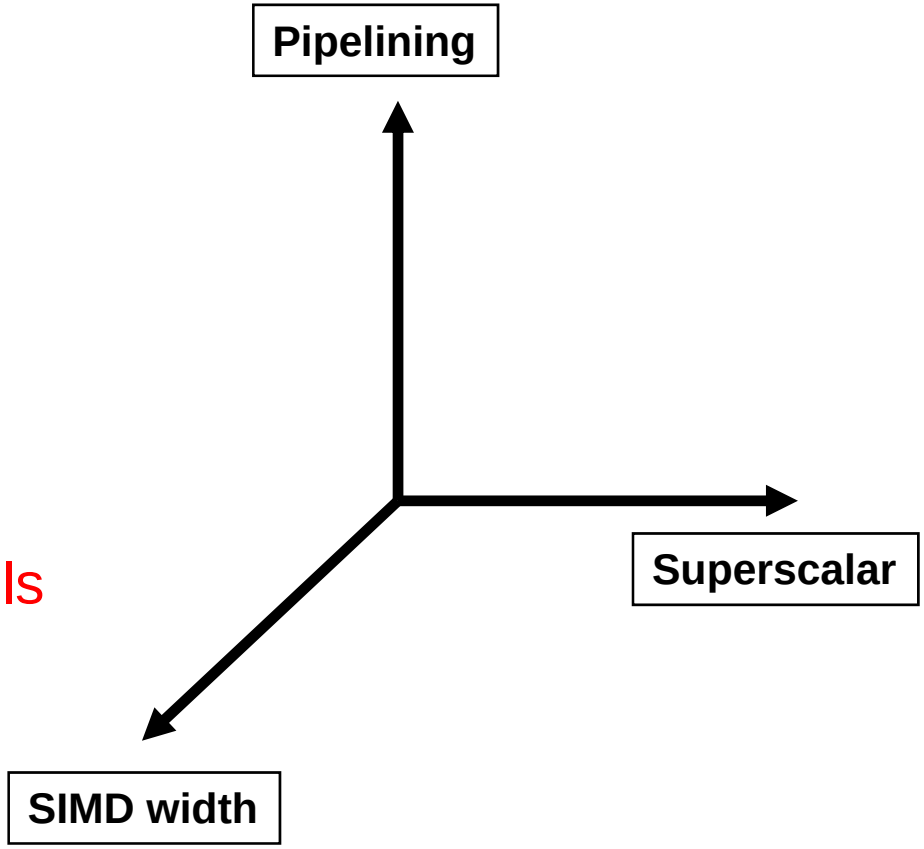
Simple Geant4 prototype developed at Northeastern University

2) Teach parallelism

- **Evangelize/teach parallel programming**
- **Two workshops arranged together w/Intel in 2007**
- **Each event:**
  - 1 day lectures, 1 day exercises
  - 5 lecturers (2 Intel + 3 CERN), 45 participants, 20 people oversubscribed
  - Survey: 100% said expectations met
  - Next workshop: Late Spring 2008
- **Licenses for the Intel Threading Tools (and other SW products) available**
  - to all CERN users

Part 1: Opportunities for scaling performance inside a core

- **First three dimensions:**
  - Superscalar: Fill the ports
    - Measure instructions per cycle
  - Pipelined: Fill the stages
    - Measure bubbles/resource stalls
  - SIMD: Fill the register width
    - Measure SSE usage



SIMD = Single Instruction Multiple Data
SSE = Streaming SIMD Extensions

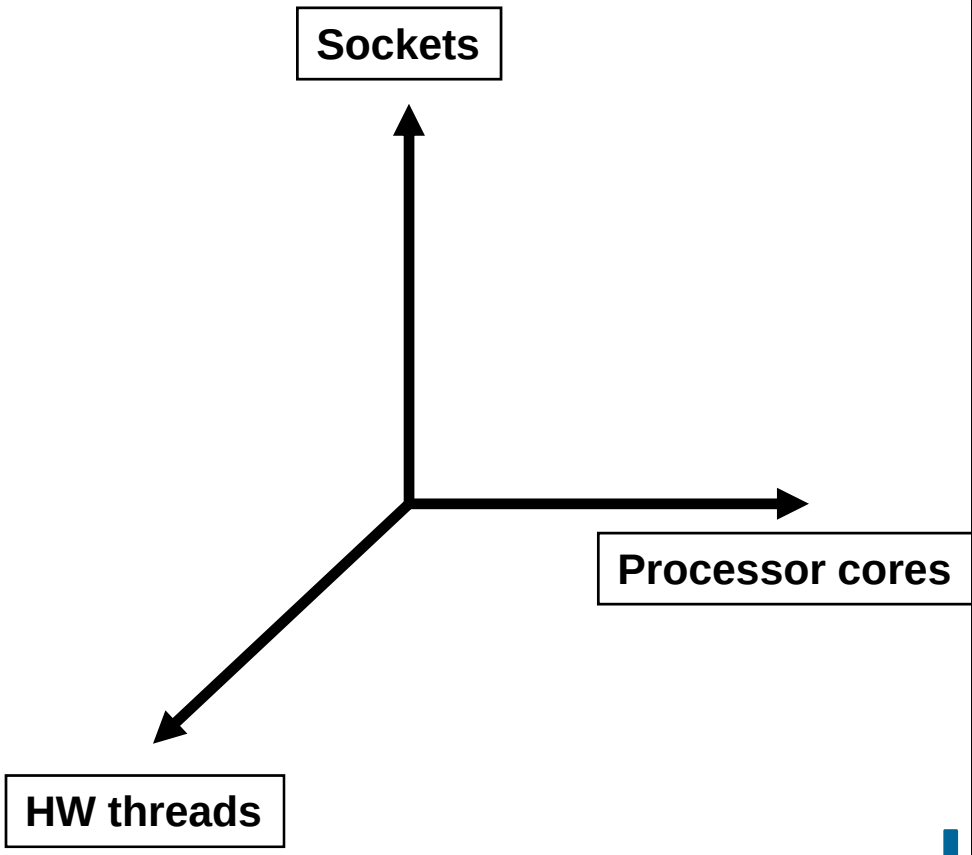
3) Rely on Symmetric Multithreading

- **Provided the memory issue is solved**
  - We could easily tolerate 4x SMT !

Cycle	Port 0	Port 1	Port 2	Port 3	Port 4	Port 5
1			load point[0]			
2			load origin[0]			
3						
4						
5						
6		subsd	load float-packet			
7						
8			load xhalfsz			
9						
10	andpd					
11						
12	comisd					
13						jbe

Part 2: Parallel execution across multithreaded cores

- **After having tried to maximize capacity within a core**
  - First three dimensions
- **We move to the next level**
  - Three additional dimensions inside a node:
    - HW threads
    - Processor cores
    - Sockets



Seventh dimension represented by multiple nodes.

Rethink concurrency in HEP

- **We are "blessed" with lots of it:**
  - Events
  - Particles, tracks and vertices
  - Physics processes
  - I/O streams (Trees, branches)
  - Buffer handling (also compaction, etc.)
  - Fitting variables
  - Partial sums, partial histograms
  - (Your favorite comes here)

C++ multithreading support

- **Beyond auto-vectorization/auto-parallelization,**
- **Large selection of low-level tools:**
  - OpenMP
  - MPI
  - pthreads/Windows threads
  - Threading Building Blocks (TBB)
  - TOP-C (from NE University)
  - RapidMind
  - Ct (in preparation)
  - etc.

Complementary tools available at CERN:
Intel Thread Checker, Thread Profiler
Linux perfmon2 (Stéphane Eranian)

Intel TBB 2.0 overview

- **Key features:**
  - Open source extension to C++ (GPL)
  - Task patterns instead of threads
    - Focus on the work, not the workers
  - Designed for scalable performance
    - Automatic scaling to use available resources
  - Components
    - Generic parallel algorithms: parallel_for, parallel_reduce, etc.
    - Low-level synchronisation primitives: atomic, mutex, etc.
    - Concurrent containers: concurrent_vector, concurrent_hash_map, etc.
    - Task scheduler
    - Memory allocation: cache_aligned_allocator
    - Timing

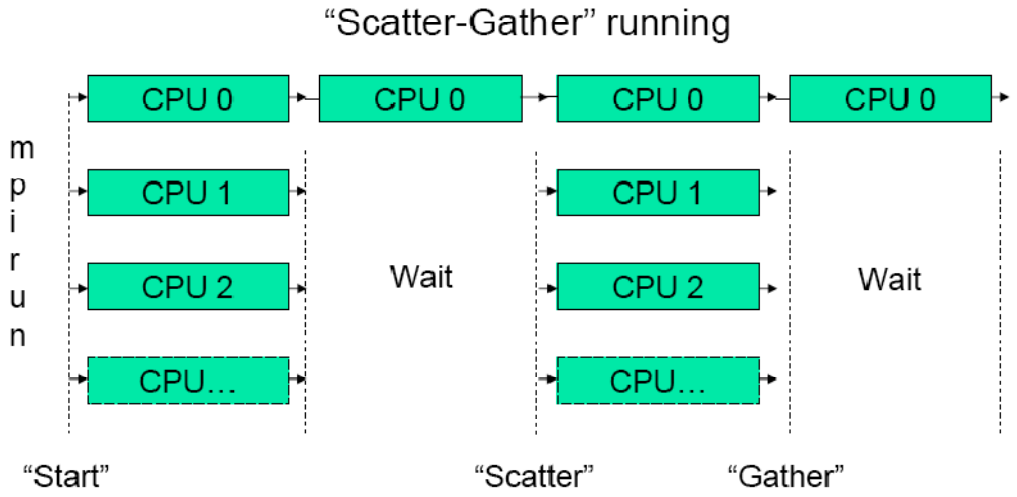
```
#include "tbb/task_scheduler_init.h"
#include "tbb/parallel_for.h"
#include "tbb/blocked_range.h"
using namespace tbb;
//
task_scheduler_init init;
  tasks = atoi( argv[1] );
//
 parallel_for(blocked_range<int>(0,
NTracksV, NTracksV / tasks),
ApplyFit(TracksV, vStations, NStations));
```

More features in preparation

Examples of parallelism: RooFit (1)

- **Example of Data Analysis (Fitting) in BaBar (SLAC)**
  - Uses MPI to run scatter/gather
    - Based on the Negative-Log Likelihood function which requires the calculation of separate values for each free parameter in each minimization step

$$NLL = \ln\left(\sum_{j=1}^{s} n_j\right) - \sum_{i=1}^{N}\left(\ln \sum_{j=1}^{s} n_j \mathcal{P}_j^i\right)$$



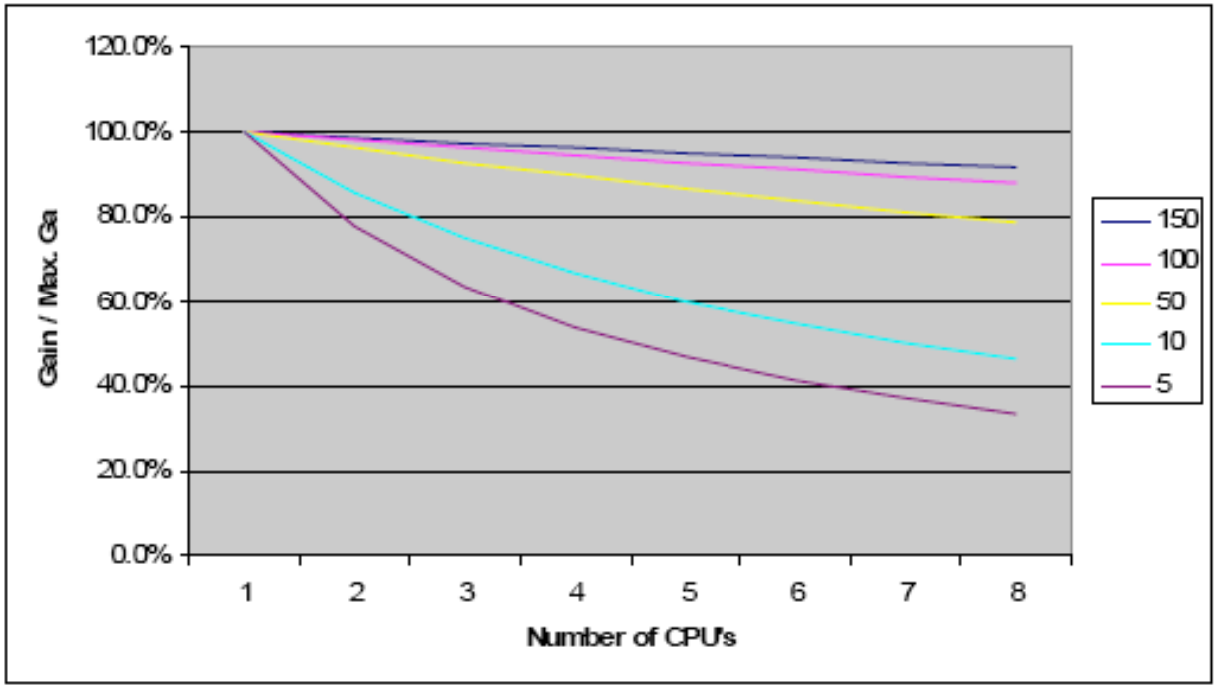
From B.Meadows's talk at RooFit Mini Workshop @ SLAC (December 2007): http://www.slac.stanford.edu/BFROOT/www/doc/Workshops/2007/BaBar_RooFit/Agenda.html

RooFit (2)

- **It works well in case of large number of parameters**

Gain ~ NCPU*(NPAR + 2) / (NPAR + 2*NCPU)

Max. Gain = NCPU



Programming strategies

- **Advice given to programming community:**
  - Get memory usage (per process) under control
    - To allow higher multiprogramming level per server
  - Draw maximum benefit from hardware threading
  - Introduce gross-grained software multithreading
    - To allow further scaling with large core counts
  - Revisit vector constructs at the very base
    - Gain performance inside each core
  - Use appropriate tools (perfmon2/Thread Profiler, etc.)
    - To monitor detailed program behaviour

Conclusions

- **CERN and its community are in front of several "new" computing issues!**
- **Some solutions are easier than others:**
  - Switch on SMT in the BIOS (whenever relevant)
  - Reduce memory foot-print
  - Gradually introduce parallelism (across cores)
  - Build an additional (5 MW?) computer centre
  - Revisit vectors
- **But, above all, maintain software scalability and portability**
- **In any case, we must teach programmers to be masters of the 7 hardware dimensions!**