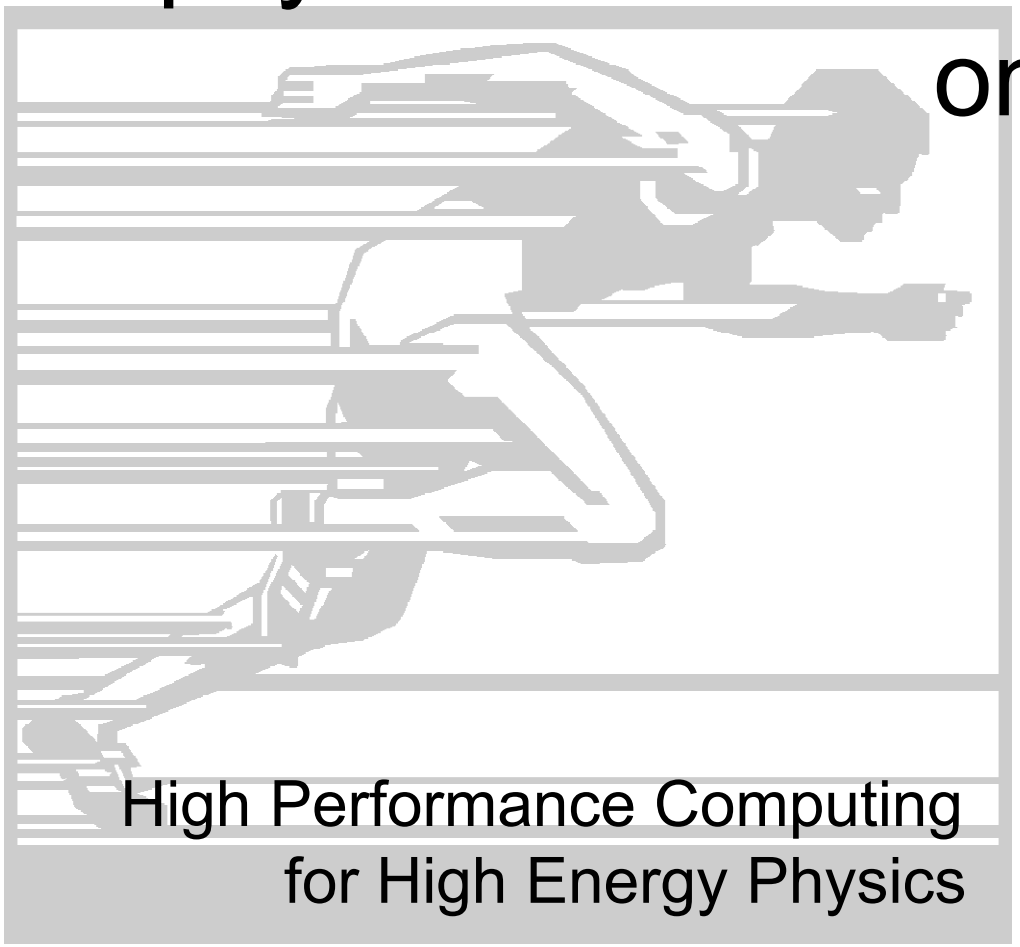
The challenge of adapting HEP physics software applications to run on many-core cpus

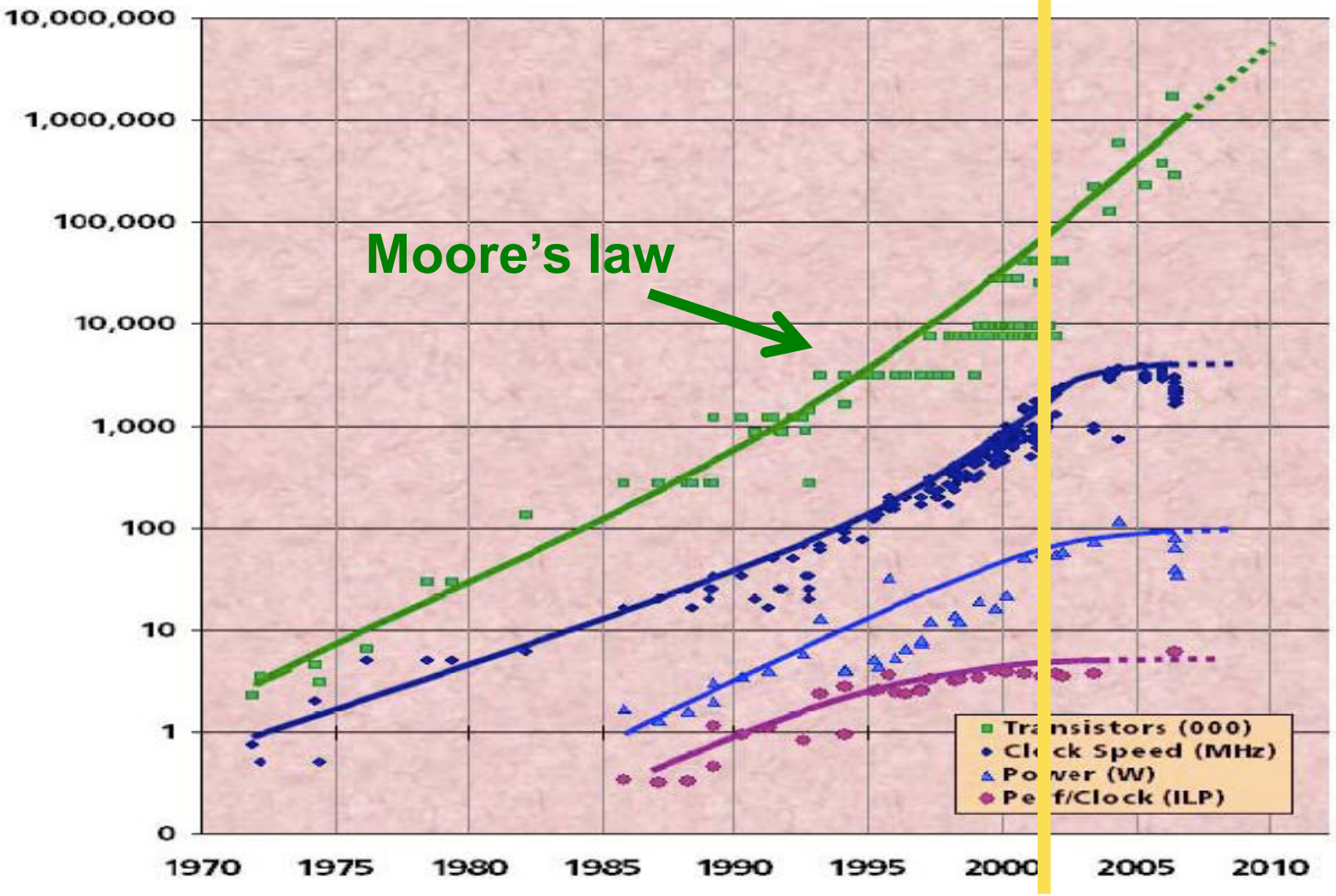


CERN, June ’09

Vincenzo Innocente
CERN

Computing in the years Zero

Transistors used to increase *raw-pow* increase *global powe*



The ‘three walls’

While hardware continued to follow Moore’s law, the perceived exponential grow of the “effective” computing power faded away in hitting three “walls”:

1. The memory wall
   - » Memory slower than cpu
2. The power wall
   - » Computing center cannot install/remove more power
3. The instruction level parallelism (micro-architecture) wall
   - » Applications unable to exploit large instruction pipelines

More details in next talk

Go Parallel: many-cores!

– A turning point was reached and a new technology emerged: **multicore**
  » Keep low frequency and consumption
  » Transistors used for multiple cores on a single chip: 2, 4, 6, 8 cores on a single chip
– Multiple hardware-threads on a single core
  » simultaneous Multi-Threading (Intel Core i7 2 threads per core (4 cores), Sun UltraSPARC T2 8 threads per core (8 cores))
– Dedicated architectures:
  » GPGPU: up to 240 threads (NVIDIA, ATI-AMD, Intel Larrabee)
  » CELL
  » FPGA (Reconfigurable computing)

Where are WE?

- HEP code does not exploit the power of current processors
  - One instruction per cycle at best
  - Little or no use of vector units (SIMD)
  - Poor code locality
  - Abuse of the heap
- Running N jobs on N=8 cores still efficient but:
  - Memory (and to less extent cpu cycles) wasted in non sharing
    - “static” condition and geometry data
    - I/O buffers
    - Network and disk resources
  - Caches (memory on CPU chip) wasted and trashed
    - L1 cache local per core, L2 and L3 shared
    - Not locality of code and data (thread/core affinity)
- This situation is already bad today, will become only worse in future many-cores architectures

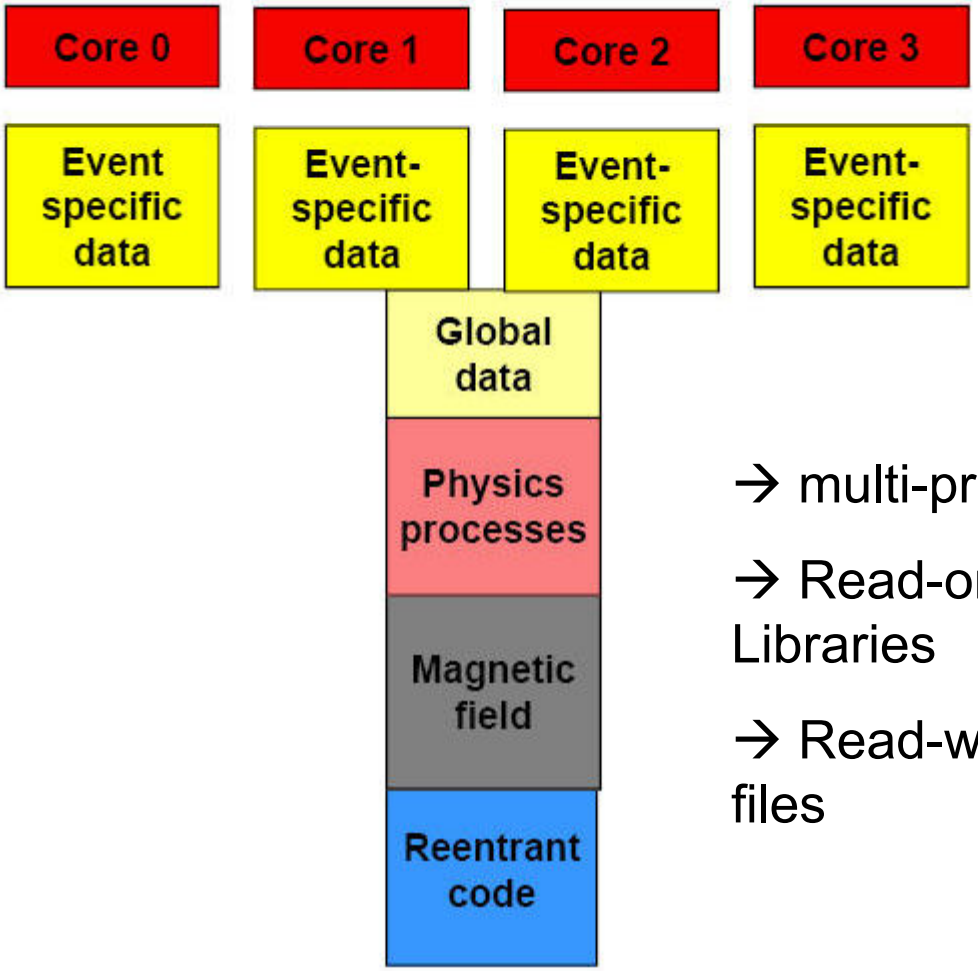
HEP software on multicore: a R&D effort

- Collaboration among experiments, IT-departments, projects such as OpenLab, Geant4, ROOT, Grid
- Target multi-core (8-24/box) in the short term, many-core (96+/box) in near future
- Optimize use of CPU/Memory architecture
- Exploit modern OS and compiler features
  - » Copy-on-Write
  - » MPI, OpenMP
  - » SSE/AltiVec, Intel Ct, OpenCL

Event parallelism

Opportunity: Reconstruction Memory-Footprint shows large condition data

How to share common data between different processes?



→ multi-process vs multi-threaded

→ Read-only: Copy-on-write, Shared Libraries

→ Read-write: Shared Memory, sockets, files

Experience and requirements

- Complex and dispersed “legacy” software
  - » Difficult to manage/share/tune resources (memory, I/O): better to rely in the support from OS and compiler
  - » Coding and maintaining thread-safe software at user-level is hard
  - » Need automatic tools to identify code to be made thread-aware
    - Geant4: 10K lines modified! (**thread-parallel** Geant4)
    - Not enough, many hidden (optimization) details
- “Simple” multi-process seems more promising
  - » ATLAS: fork() (exploit copy-on-write), shmem (needs library support)
  - » LHCb: python
  - » PROOF-lite
- Other limitations are at the door (I/O, communication, memory)
  - » Proof: client-server communication overhead in a single box
  - » Proof-lite: I/O bound >2 processes per disk
  - » Online (Atlas, CMS) limit in in/out-bound connections to one box

Exploit Copy on Write (COW)

See Sebastien Binet's talk @ CHEP09

- Modern OS share read-only pages among processes dynamically
  - » A memory page is copied and made private to a process only when modified
- Prototype in Atlas and LHCb
  - » Encouraging results as memory sharing is concerned (50% shared)

Memory (ATLAS)
One process: *700MB VMem and 420MB RSS*
COW:
(before) evt 0: private: 004 MB | shared: 310 MB
(before) evt 1: private: 235 MB | shared: 265 MB
. . .
(before) evt50: private: 250 MB | shared: 263 MB

Exploit “Kernel Shared Memory”

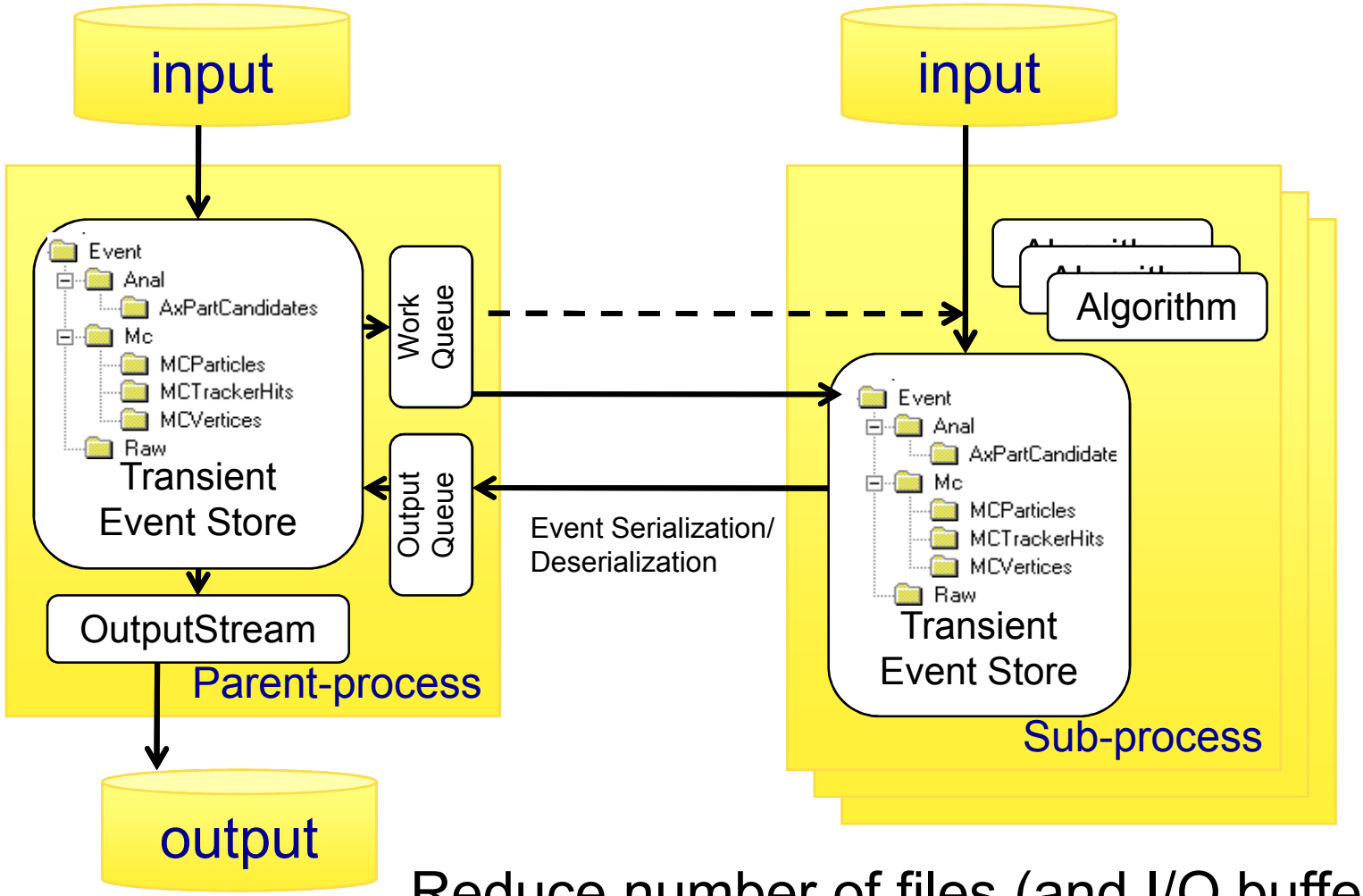
- KSM is a linux driver that allows dynamically sharing identical memory pages between one or more processes.
  - » It has been developed as a backend of KVM to help memory sharing between virtual machines running on the same host.
  - » KSM scans just memory that was registered with it. Essentially this means that each memory allocation, sensible to be shared, need to be followed by a call to a registry function.
- Test performed “retrofitting” TCMalloc with KSM
  - » Just one single line of code added!
- CMS reconstruction of real data (Cosmics with full detector)
  - » No code change
  - » 400MB private data; 250MB shared data; 130MB shared code
- ATLAS
  - » No code change
  - » In a Reconstruction job of 1.6GB VM, up to 1GB can be shared with KSM

A prototype for a **thread-parallel** Geant4

Gene Cooperman and Xin Dong (NEU Boston)

- » Master/Worker paradigm
- » Event-level parallelism: separate events on different threads
  - only 1 RAM : increase sharing of memory between threads
- » Phase I : code sharing, but no data sharing *Done*
- » Phase II : sharing of geometry, materials, particles, production cuts *Done, undergoing validation*
- » Phase III : sharing of data for EM physics processes *In Progress*
  - ❑ Physics tables are read-only, but small caches and different API
- » Phase IV : other physics processes *Todo*
- » Phase V : General Software Schema: new releases of sequential Geant4 drive corresponding multi-threaded releases *In Progress*
  - Patch **parser.c** of **gcc** to identify static and globals declarations in G4
  - Currently 10,000 lines of G4 modified automatically + 100 lines by hand

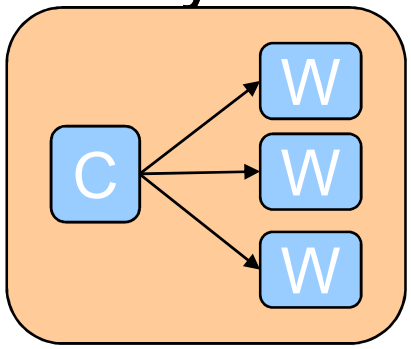
Handling Event Input/Output



Reduce number of files (and I/O buffers) by 1-2 orders of magnitude

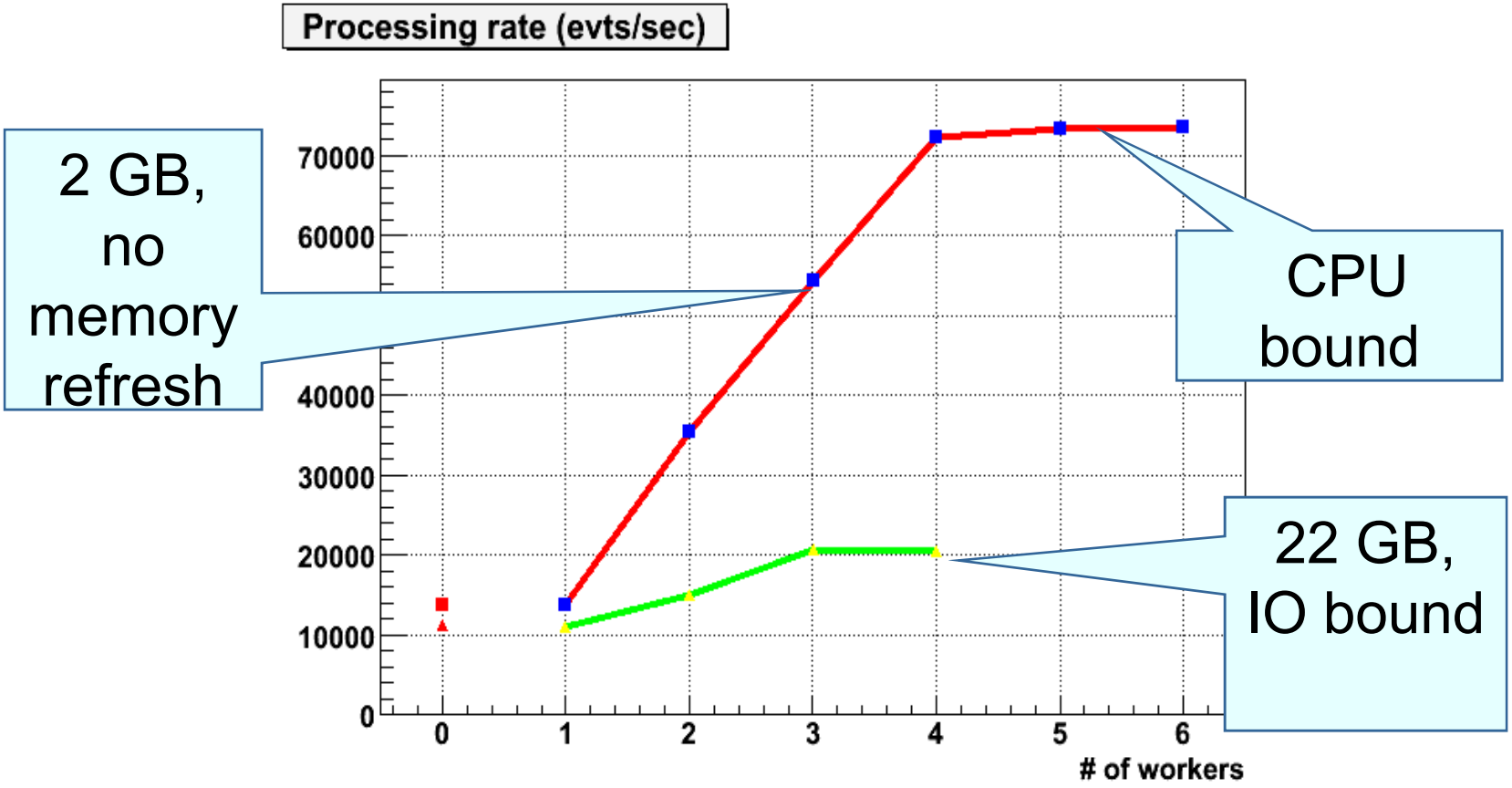
PROOF Lite

- PROOF Lite is a realization of PROOF in 2 tiers
  - The client starts and controls directly the workers
  - Communication goes via UNIX sockets
- No need of daemons:
  - workers are started via a call to ‘system’ and call back the client to establish the connection
- Starts N_{CPU} workers by default



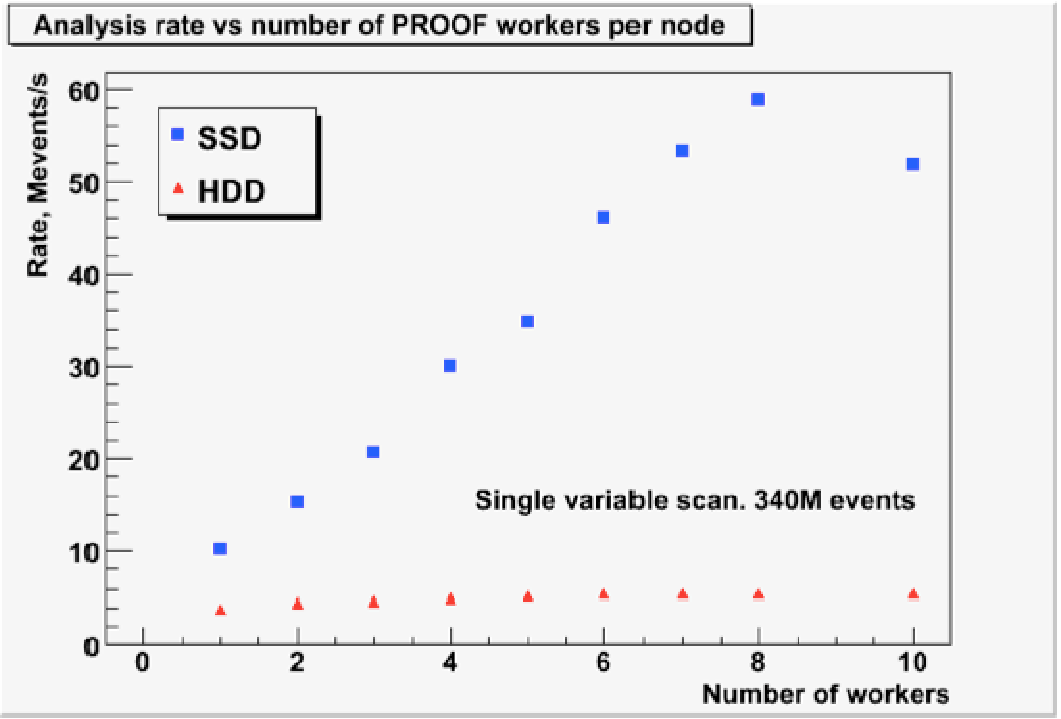
Scaling processing a tree, example (4core box)

- Datasets: 2 GB (fits in memory), 22 GB



SSD vs HDD on 8 Node Cluster

See Sergey Panitkin's talk



Solid State Disk:
120GB for 400Euro

- Aggregate (8 node farm) analysis rate as a function of number of workers per node
- Almost linear scaling with number of nodes

Algorithm Parallelization

- Ultimate performance gain will come from parallelizing **algorithms** used in current LHC physics application software
  - » Prototypes using posix-thread, OpenMP and parallel gcclib
  - » Effort to provide basic thread-safe/multi-thread library components
    - Random number generators
    - Parallel minimization/fitting algorithms
    - Parallel/Vector linear algebra
- Positive and interesting experience with MINUIT
  - » Parallelization of parameter-fitting opens the opportunity to enlarge the region of multidimensional space used in physics analysis to essentially the whole data sample.

Parallel MINUIT *A. L. and Lorenzo Moneta*

– Minimization of Maximum Likelihood or χ^2 requires iterative computation of the gradient of the NLL function

$$\left.\frac{\partial NLL}{\partial \hat{\theta}}\right|_{\hat{\theta}_0} \approx \frac{NLL(\hat{\theta}_0 + \hat{\mathrm{d}}) - NLL(\hat{\theta}_0 - \hat{\mathrm{d}})}{2\hat{\mathrm{d}}}$$

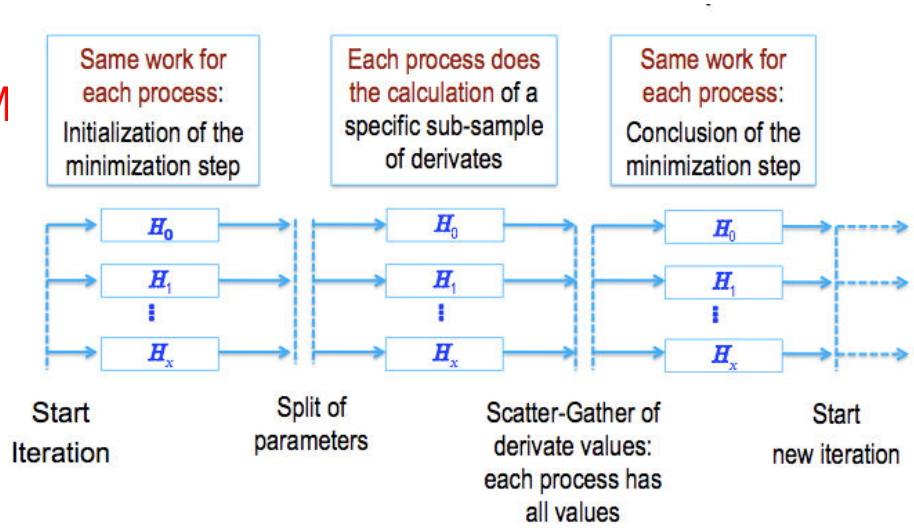
$$NLL = \ln\left(\sum_{j=1}^{s} n_j\right) - \sum_{i=1}^{N}\left(\ln \sum_{j=1}^{s} n_j \mathcal{P}_j^i\right)$$

j species (signals, backgrounds)
n_j number of events for specie j
$\mathcal{P}_j$ probability density functions (PDFs)
N number total of events to fit

– Execution time scales with number θ free parameters and the number N of input events in the fit

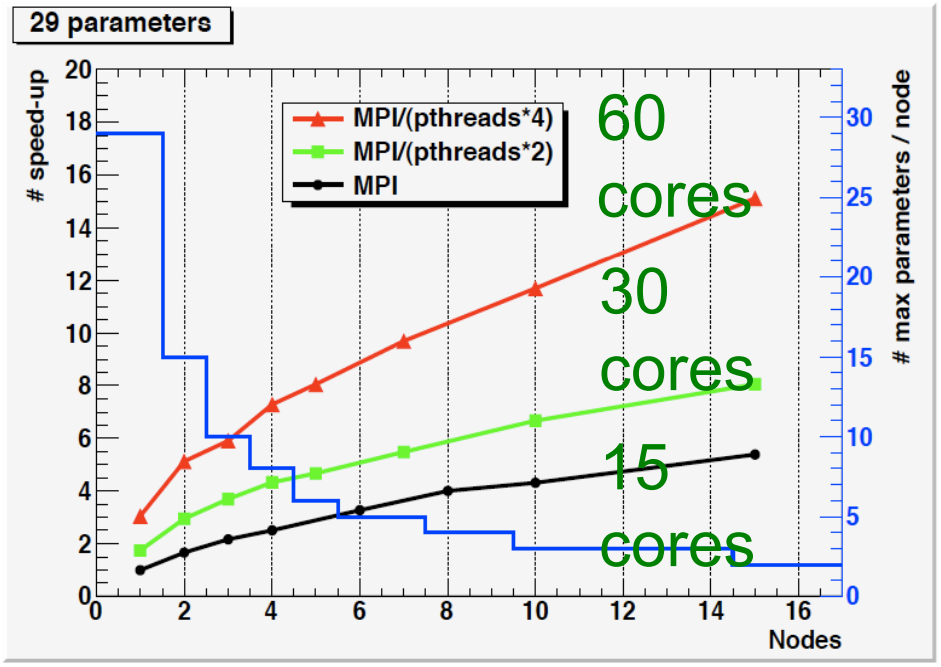
– Two strategies for the parallelization of the gradient and NLL calculation:

1. Gradient **or** NLL calculation on the same multi-cores node (OpenM
1. Distribute Gradient on different nodes (MPI) **and** parallelize NLL calculation on each multi-cores node (pthreads): **hybrid solution**



Minuit Parallelization – Example

– Waiting time for fit to converge down from several days to a night (Babar examples)
  » iteration on results back to a human timeframe!



– Improved version of the code (MPI parallelization of gradient AND NLL) currently under test at CNAF (thanks to A. Fella for the support)

Outlook

- The work described above demonstrated how we can exploit effectively multi-core architectures essentially running the very same application on all cores of the same node.
- The successful parallelization of Minuit opens the way to exploit MPI-like models in the area of analysis and for specialized applications such as alignment.
- I/O is often a bottleneck
  - » parallel application may produce many small files that will need to be merged
  - » fast inter-process communication may open novel ways to distribute "work" and collect products at event or even at sub-event level.
- In most of these cases modification/addition at kernel and system level are required
- Current models of job scheduling and resource allocation in use in HEP/Grid do not easily make room for this