Adding Concurrency to LHC Software

Many-core architectures for LHCb, 25 April 2012

Pere Mato/CERN

28/3/2012

Outline

- Why we need to do something
- What we think we need to do
- How and what tools and technologies to use
- When can/should we do it

Why?

- We need to adapt current data processing applications to the new many-core architectures (~100 cores)
  - No major change expected in the overall throughput with respect trivial one-job-per-core parallelism
- Reducing the required resources per core
  - I/O bandwidth
  - Memory requirements and locality
  - Connections to DB, open files, etc.
- Reduce latency for single jobs (e.g. trigger, user analysis)
  - Run a given job in less time making use of all available cores
- Window of opportunity with the long LHC shutdown

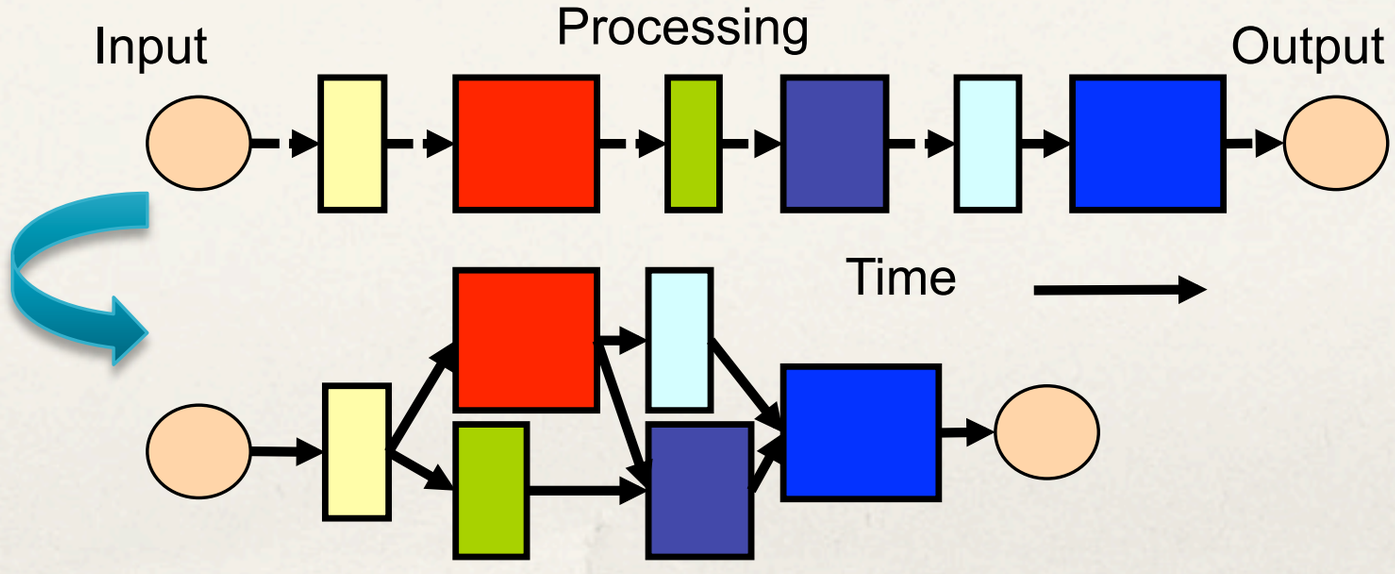
Concurrency at what level?

- Concrete algorithms can be parallelized with some effort
  - Making use of Threads, OpenMP, MPI, GPUs, etc.
  - But difficult to integrate them in a complete application
  - Performance-wise only makes sense to parallelize the complete application and not only some of the parts
- Developing and validating parallel code is very difficult
  - 'Physicists' should be saved from this
  - Concurrency will limit what can and can not be done in the algorithmic code (strong policies required)
- At the **Framework level** you have the full overview and control of the application

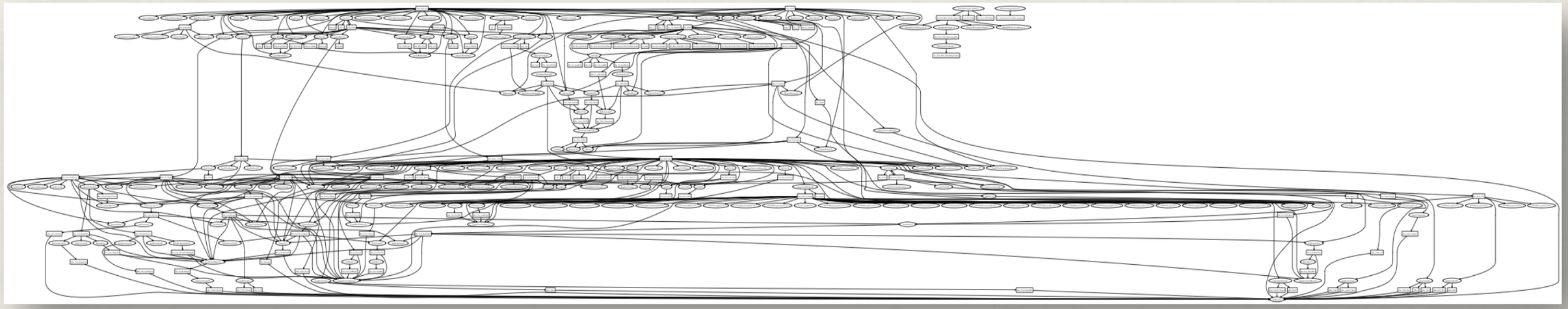
Concurrent 'Algorithm' processing

- Ability to schedule modules/algorithms concurrently
  - Full data dependency analysis would be required (no global data or hidden dependencies)
  - Need to resolve the DAGs (Direct Acyclic Graphs) statically and/or dynamically

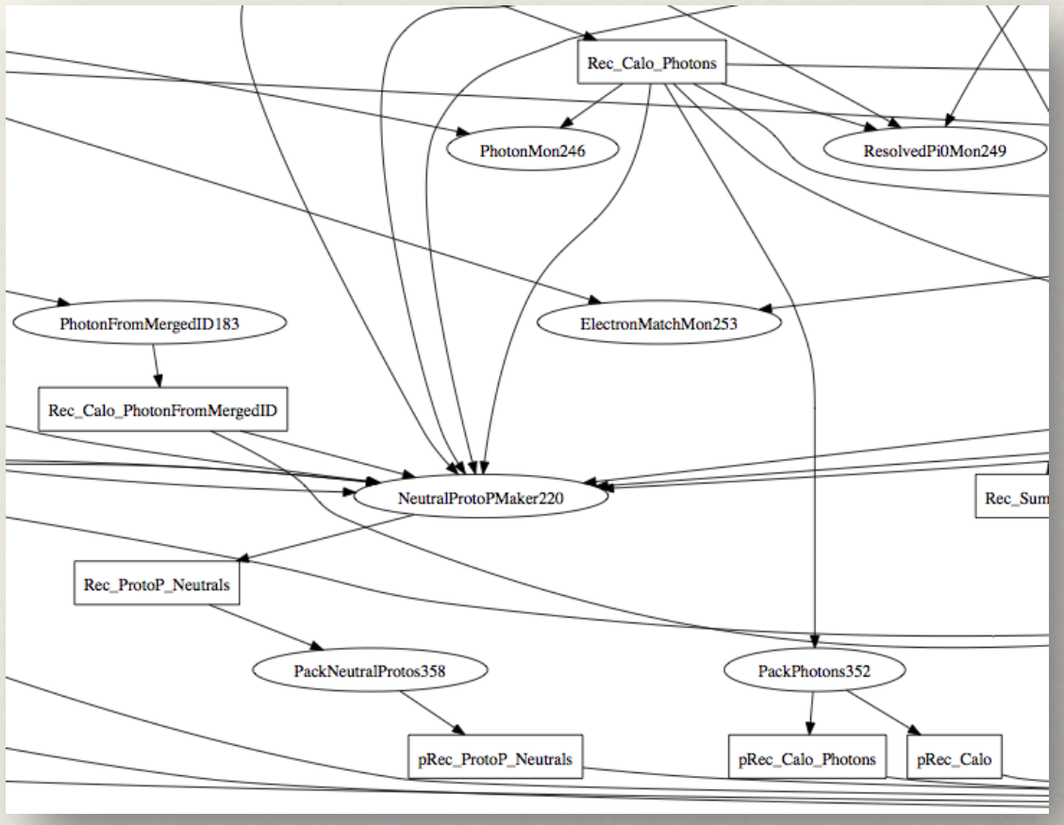
- Not much to gain with today's existing 'Algorithms'
  - Potential 'concurrency' factor rather low



Example: LHCb Reconstruction

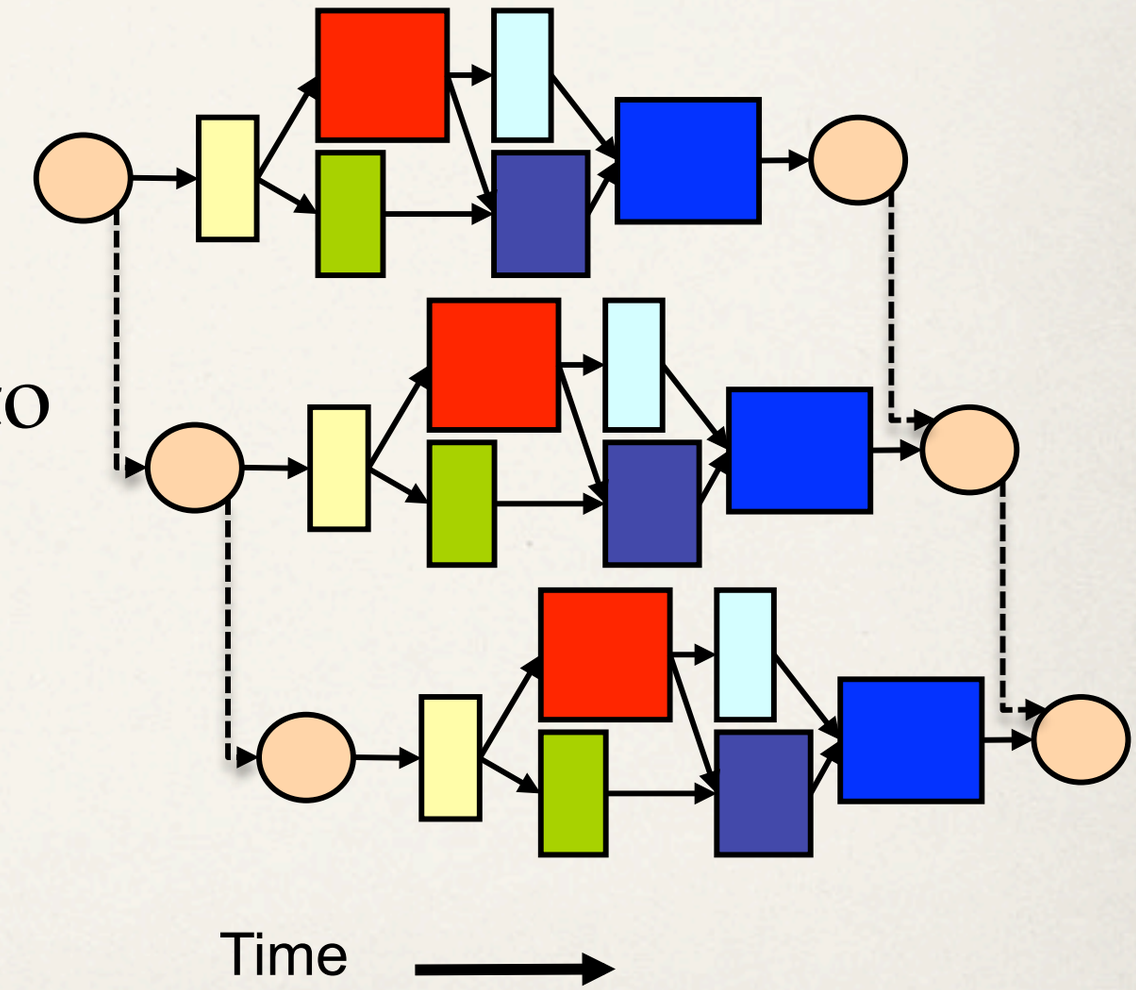


- DAG of Brunel
  - Obtained from the existing code instrumented with ‘Auditors’
  - Probably still missing ‘hidden or indirect’ dependencies (e.g. Tools)
- Can give an idea of potential for ‘concurrency’
  - Assuming no changes in current reconstruction algorithms

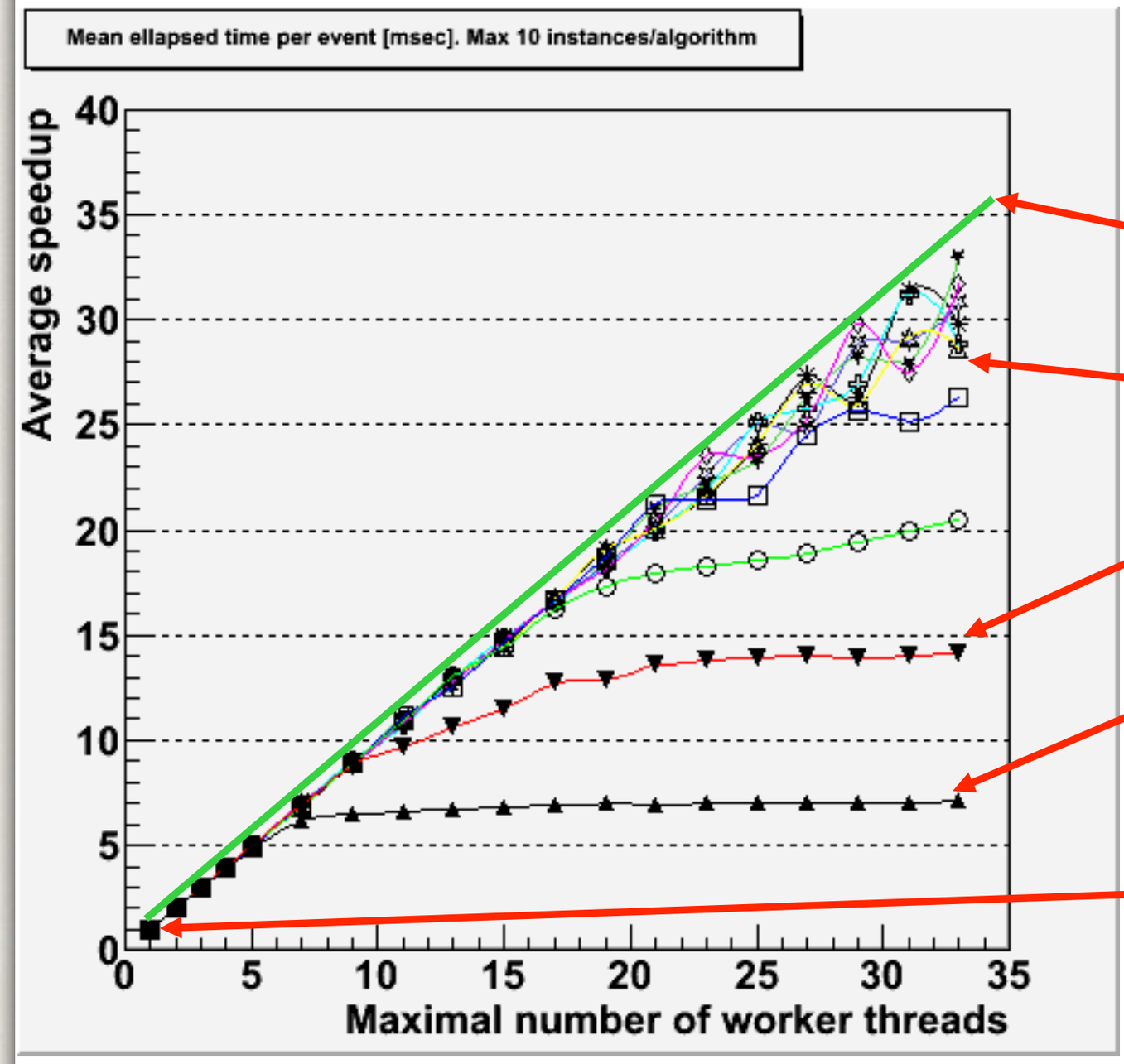


Many 'Concurrent' Events

- Need to deal with the tails of sequential processing
- Introducing Pipeline processing
  - Exclusive access to resources or non-reentrant algorithms can be pipelined e.g. file writing
- Current frameworks will need to be adapted
  - Design or use a powerful and flexible scheduler
  - Need to define the concept of an "event context"
- Nice results from Markus Frank



Model Result Top 4:
Max. 10 instances of top 4 algorithms



- Max 10 events in parallel
- TOP 4 algorithms reentrant with max 10 instances
- Cut 25 msec [4.3 %]

Theoretical limit

Max evts > 3
Speedup up to ~30

Max 2 events
1 event * 2

Max 1 event
Algorithmic parallel limit
Speedup: ~7

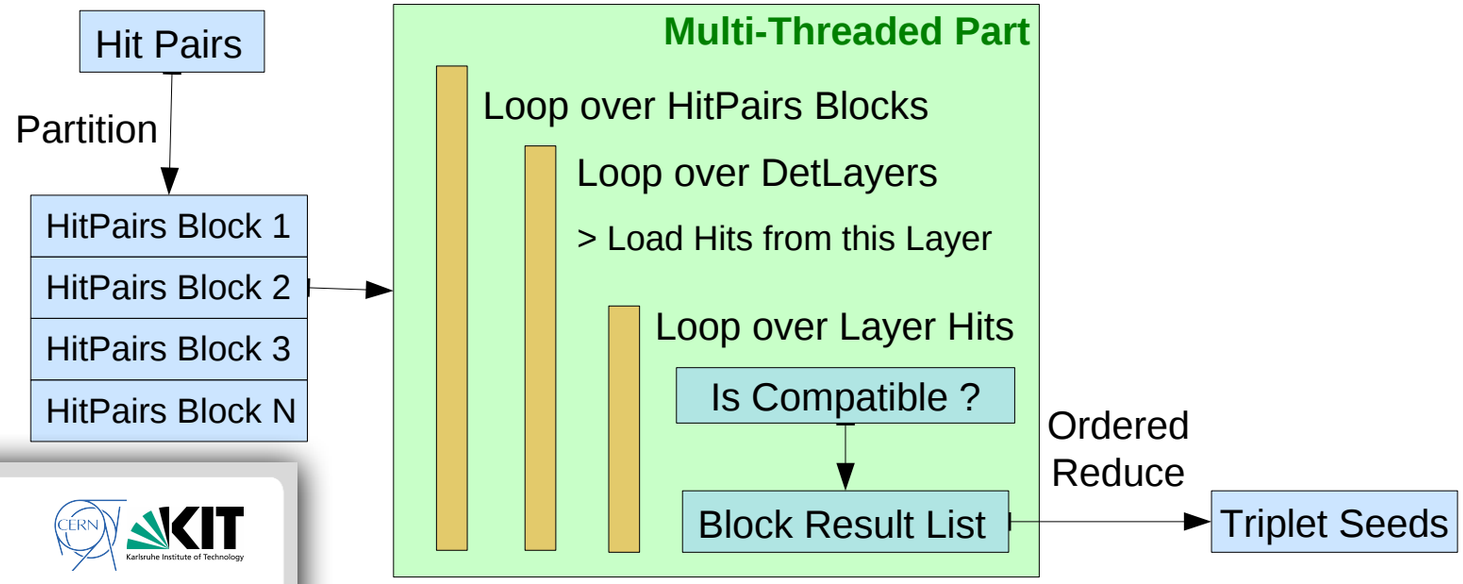
One thread
= classic processing (t_1)

Algorithm Parallelization

- In current applications there is always an ‘Algorithm’ that takes a large fraction of the total processing time
  - Limiting the application scaling for many-cores
- Several options:
  - Make the algorithm fully ‘re-entrant’ ==> probably very difficult
  - Instantiate several copies of the ‘Algorithm’, each one working on a different event concurrently
  - Parallelize the Algorithm, for example parallelizing the main loop over data items (e.g. ‘TrackFit’ Algorithm is looping over ‘Tracks’)
- Nice results from Thomas Hauth and Danilo Piparo

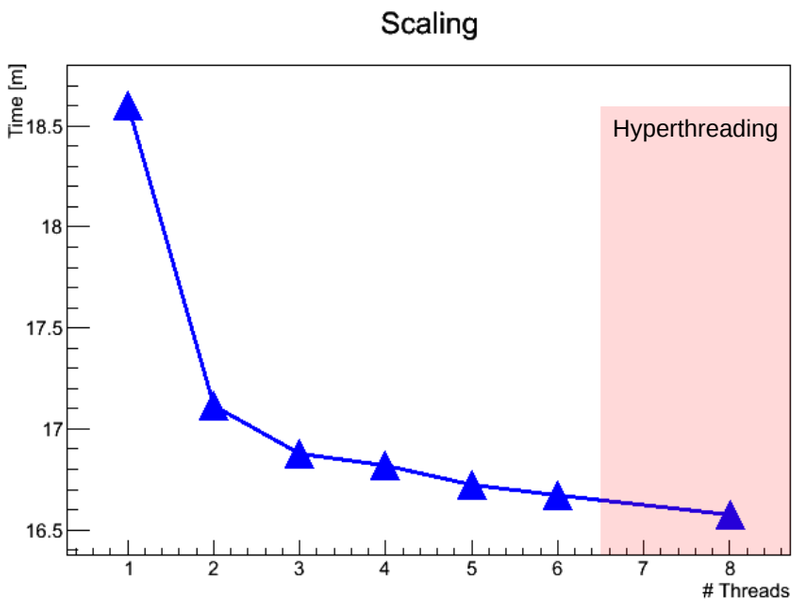
Triplet Seeding in CMS: Parallel Execution

- Before running the multi-threaded code part, the Hit-Pair list is partitioned into N equally sized work chunks
- The available threads process the HitPair Blocks via TBB's **`parallel_for`** method and stores the resulting TripletSeeds in a Block-local Result List
- TripletSeeds resulting from a HitPairs Block are merged into the output collection respective to their order in the input collection
- This guarantees the order of the output is not depending of the amount of threads



Scaling behavior of the Implementation

- Higher-than-expected scaling from 1 to 2 cores, probably due to the positive effects of using the L1/L2 caches of two cores simultaneously



The triplet seeding takes about 10% of the runtime in the serial version

What?

- We need to add ‘concurrency’ to our data processing applications at 3 levels at the same time
  - Event-level parallelism
  - Algorithm-level parallelism
  - Sub-algorithm-level parallelism (including I/O)
- Essential to ‘standardize’ on a **single concurrency programming model** to ensure efficient and balanced use of resources
- LHC experiments would like to see a **set of functional components** from where to pick and choose what to incorporate into their frameworks
  - Experiments have a huge investment in ‘algorithmic’ code and configuration based on a specific framework

How?

- We do not know [yet]
- So, we organized a workshop...
  - Workshop on Concurrency in the many-cores era, 21-22 November 2011, Fermilab
    - Understand the requirements and constraints of current and future experiments
    - Assess the interest for a joint development
    - Determine the degree of overlap between the needs and medium to long term plans of the Fermilab and CERN experiments
    - Identify potential technologies and ensure that all the relevant solutions are being explored

Outcome of the FNAL workshop

- Interest for common effort to make rapid progress on **exploratory R&D activities during 2012**
- Aiming to **share a common concurrency model**
- Identified a number of '**demonstrators**' to exercise different capabilities in a small scale with clear deliverables, metrics, and work in short cycles of a few months long
- Setup regular bi-weekly meetings (Wed at 17:00 CET)
  - Forum on Concurrent Programming Models and Frameworks
  - Understanding common requirements and constraints
  - Share knowledge and learn from each other
  - As a community, identify what tool/library/model works and what does not for our applications

Demonstrators

- Created a web site concurrency.web.cern.ch
- Current set of proposed 'demostrators'
  - details can be found in slides from presentations

Demostrator	Summary	Contact Person(s)
'Whiteboard' service	Design and implementation of a 'service' able to schedule algorithms/modules based on their data dependencies	B. Hegner, P. Mato
ng-go-gaudi	Investigate use of Go as a programming language to expose and harness concurrency in HEP	S. Binet
CET multicore framework	Use of MPI and OpenMP parallelism on a DAQ project based on Darkside-50	M. Paterno, J. Kowalkowski
libdispatch investigation	Further investigate use of libdispatch (GCD)	C. Jones
Evaluation of framworks	Define standard set of modules timings and configurations, data samples, and set of module dependencies. Define torture tests that will help determine performance limits.	C. Jones
Multithreaded I/O	An attempt to integrate ROOT I/O subsystem and libdispatch by scheduling the different operations (disk I/O, compression, object serialization) concurrently	C. Jones, Ph. Canal
OpenCL for Physics Applications	Find a programming model to exploit current and future hardware for compute intensive tasks	T. Hauth, V. Innocente
Virtualization	Can virtualization help in improving performance in many core environment	P. Buncic
TBB investigation for SuperB	Implement a small system with the Intel Threading Building Blocks (TBB), which is a library offering a rich approach to expressing parallelism in a C++ program	F. Giacomini
Data Locality in G4	Try different data flow organizations within G4 and measure the actual effect on the use and leverage of CPU caches	Ph.Canal
High performance simulation	Develop a simulation aimed at achieving maximum performance on parallel architectures integrating fast and full simulation in the same framework. Study parallelism at different granularity levels.	F. Carminati
Object sharing	Demonstrate sharing of C++ objects among processes	P. Calafiura
Likelihood parallelization	Parallelization of likelihood functions for data analysis	A. Lazzaro

Some of the “Demonstrators” that my group is involved directly

Virtualization

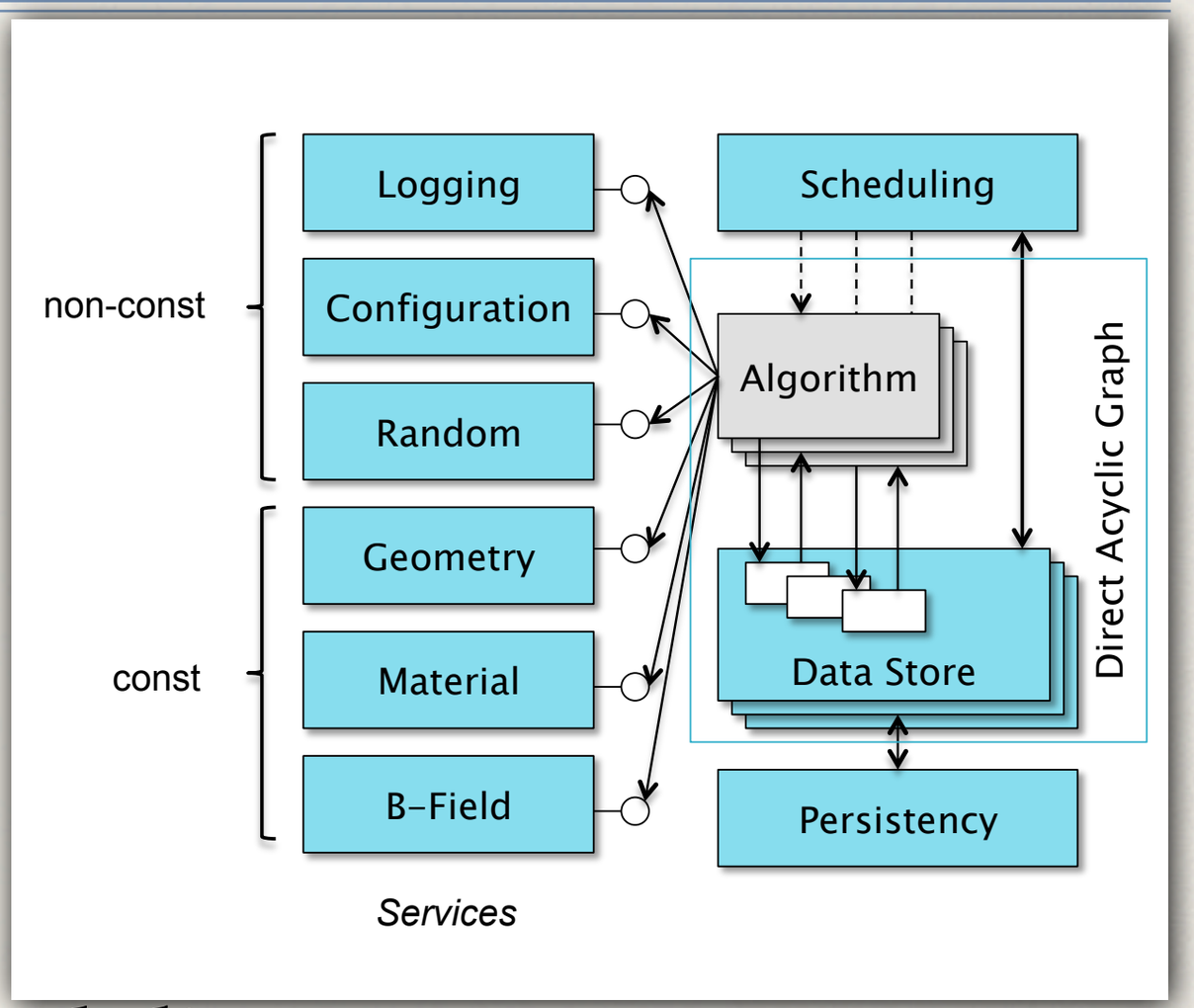
- Unmodified applications can run on many VM instances exploiting the many cores of new CPUs
- Virtualization can also help in reducing memory needs
  - KVM/KSM provides a mechanism to share the same memory pages between guests

- Near term
  - Develop tools to benchmark application performance and resource consumption while running in virtualized environment
    - Basic tools/scripts exist, need to be properly packaged (2 weeks)
  - Integrate LHC application(s) benchmarks into standard CernVM test suite for regression testing
    - Testing framework developed last year as GSOC project
    - Here we need some input and guidance form the experiments
    - Investigate potential for memory sharing across virtual machines
    - Possible task for a summer student (6 months)
- Mid to long term
  - Depending on technology evolution and availability of the hardware
  - Investigate possible use of GPGPU/MIC from VM environment
  - End of the year

CernVM Team
(Predrag Buncic)

Whiteboard Demonstrator

- Knowing the dependencies of algorithms (i.e. data transformations) determines when things can be scheduled without conflicts
  - Defining a single memory model essential for a parallel application



- Plan on testing algorithm scheduling based on TBB (Threading Building Blocks), libdispatch (GCD)
  - Update in today's Concurrency Forum Meeting

Benedikt Hegner
Riccardo Bianchi

Parallelization Opportunities In Math Libraries

- Parallelization in tools for data analysis and concentrated on likelihood evaluation (fitting)
  - most time consuming tasks and immediate benefit for end-users
  - other analysis tools (e.g. multi-variate tools) would benefit as well from same code optimization
  - very useful findings from prototype developed by Openlab
  - opportunity to work on optimize and parallelize algorithms at the same time
- Whenever possible, a parallelized version of an algorithm should be provided
  - Example of Minuit. Parallel version can be used without changing user code
- Need to improve also thread safety of existing code
- Started investigation of parallelization in vector and matrix operations (reconstruction or simulation applications)
  - vectorization looks promising
- Random number generators for parallel applications
- Other parallelization opportunities exist but less relevant in HEP
  - e.g. parallelization of large linear algebra systems

ROOT Team
(Lorenzo Moneta)

Geant4 Demonstrators

G4MT overview in 1 slide

- G4MT today parallelizes at the event level
  - Static scheduling of events, per-event RNG seed
- Goal: 100% reproducibility – or as good as G4
- Threads share RO data: geometry and EM XS
  - A small set of classes is involved
  - Different instances of most classes on each thread
- Cost of extra worker (thread) is small fraction of the total memory footprint

Geant4 Team
(John Apostolakis)

Summary: short term goals

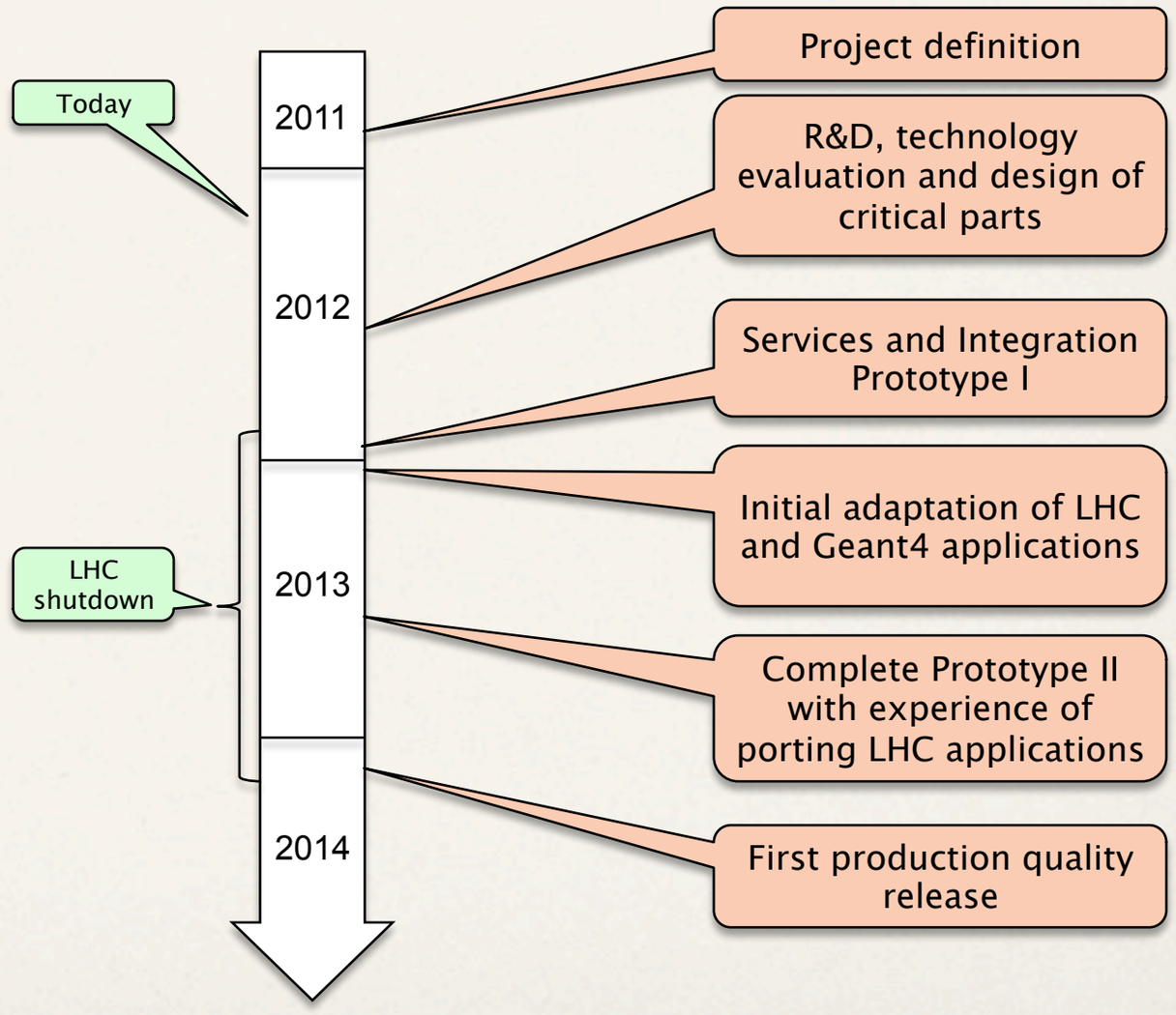
- Adapt to external dispatcher parallelism
  - Identify what changes would be required
  - Proceed in collaboration with G4 experts
- Investigate potential for improved use of caches by 'bunching' particles by type
  - Use sequential Geant4 as test-bed
  - Check effect of reordering tracks – is cache use changed ? Does CPU time profile change?

GPUs also…

- Explore the possibility to use GPUs to improve the performances of specific software algorithms used in Trigger
  - Use of GPU in NA62 Trigger (Felice's talk in this meeting)
- Prototype implementation of a radiation transport algorithm with the Geant4 toolkit using GPUs
  - Google Summer of Code student granted to work on this

When?

- Window of opportunity with the long LHC shutdown
- Use 2012 for R&D and partial prototype developments "demonstrators"
- End of 2013 have components ready for Experiments to integrate



Summary

- I do not need to convince you why we need to do it
- I tried to convince you that we need to add ‘concurrency’ at 3 levels (event, algorithm, sub-algorithm)
  - Adding it at one level or two is not sufficient for typical ‘offline’ data processing applications
  - Common concurrency model is essential
- R&D activities organized as a set of “demonstrators” executed by different teams in the LHC experiments/Labs
  - Forum on Concurrency Programming Models and Frameworks
  - Coordinating the interested people to cover all aspects
  - Coming with conclusions (yes/no) within few months

http://concurrency.web.cern.ch/