Introducing Concurrency in the Gaudi Data Processing Framework

M. Clemencic, B. Hegner, D. Piparo, P. Mato
(CERN)

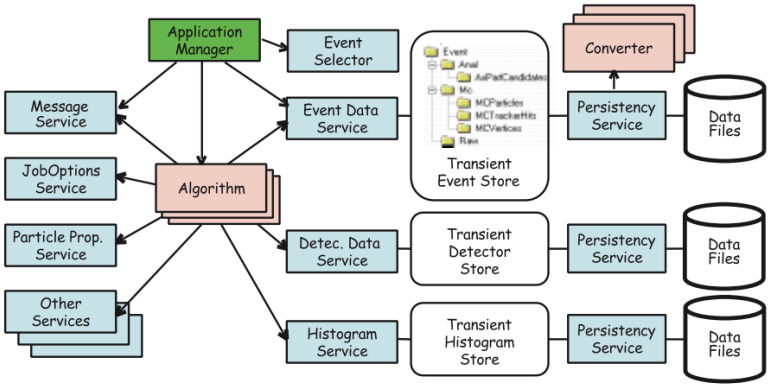
CHEP 2013, Amsterdam

Concurrency at What Level?

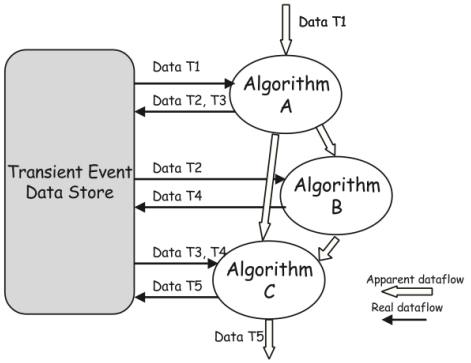
- It is obvious that we have to “go parallel” rather sooner than later
- How do we do that in concrete?
- Multiple jobs
  - Huge memory consumption
  - Job and output file management a problem
  - Huge number of resources needed (open files, DB connections, …)
- Multiple processes
  - Helps on memory consumption
  - File merging a problem
  - Number of required resources not addressed
- Concurrent framework
  - Helps greatly on memory consumption
  - Reduces number of required resources
  - Allows concurrent handling of multiple events
    - **Pre-requisite for offloading to heterogenous resources**
  - More challenging software wise !

The Gaudi Framework

- Gaudi is an experiment framework
- Among others used by ATLAS and LHCb



- Based on the idea of splitting up the work in separate objects (algorithms)



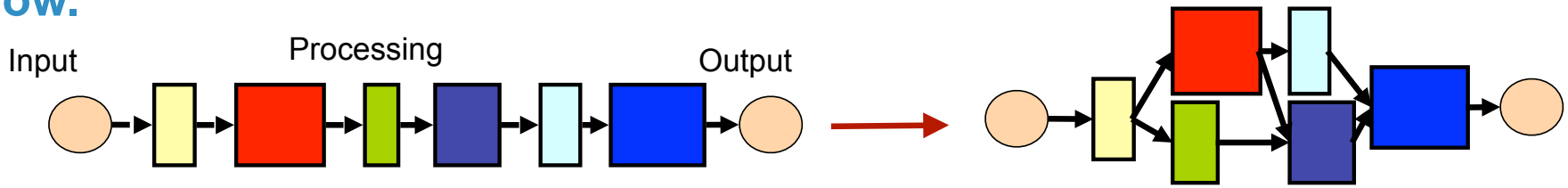
Basic Principle for Introducing Concurrency

- In current experiment frameworks the work is already split into smaller tasks (a.k.a. algorithms)
- Concurrent task execution is in theory constrained by two concepts

- **Data Flow**
  - Algorithms depend on data products other algorithms can produce
  - E.g. electron reconstruction requires ecal clusters
- **Control Flow**
  - Conditional execution of algorithms or sequences thereof
  - Trigger as prime example

Resolve these dependencies automatically.

Run everything in parallel that isn't constrained by control flow or data flow.



Prototyping Project

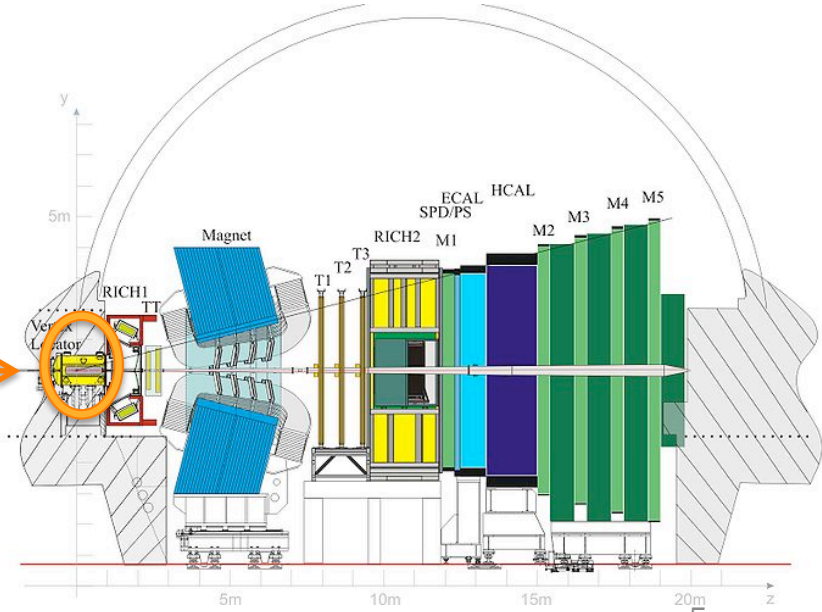
Provide refurbished Gaudi framework which

- Allows **concurrent** execution of **algorithms**
- Supports simultaneous processing of **multiple events**
- Requires minimal change of user code

Pragmatic approach

- Minimize everything that has an impact on current users of Gaudi
- Development centered around a real use case
  - 14 algorithms and associated tools
  - Raw decoding and Velo tracking

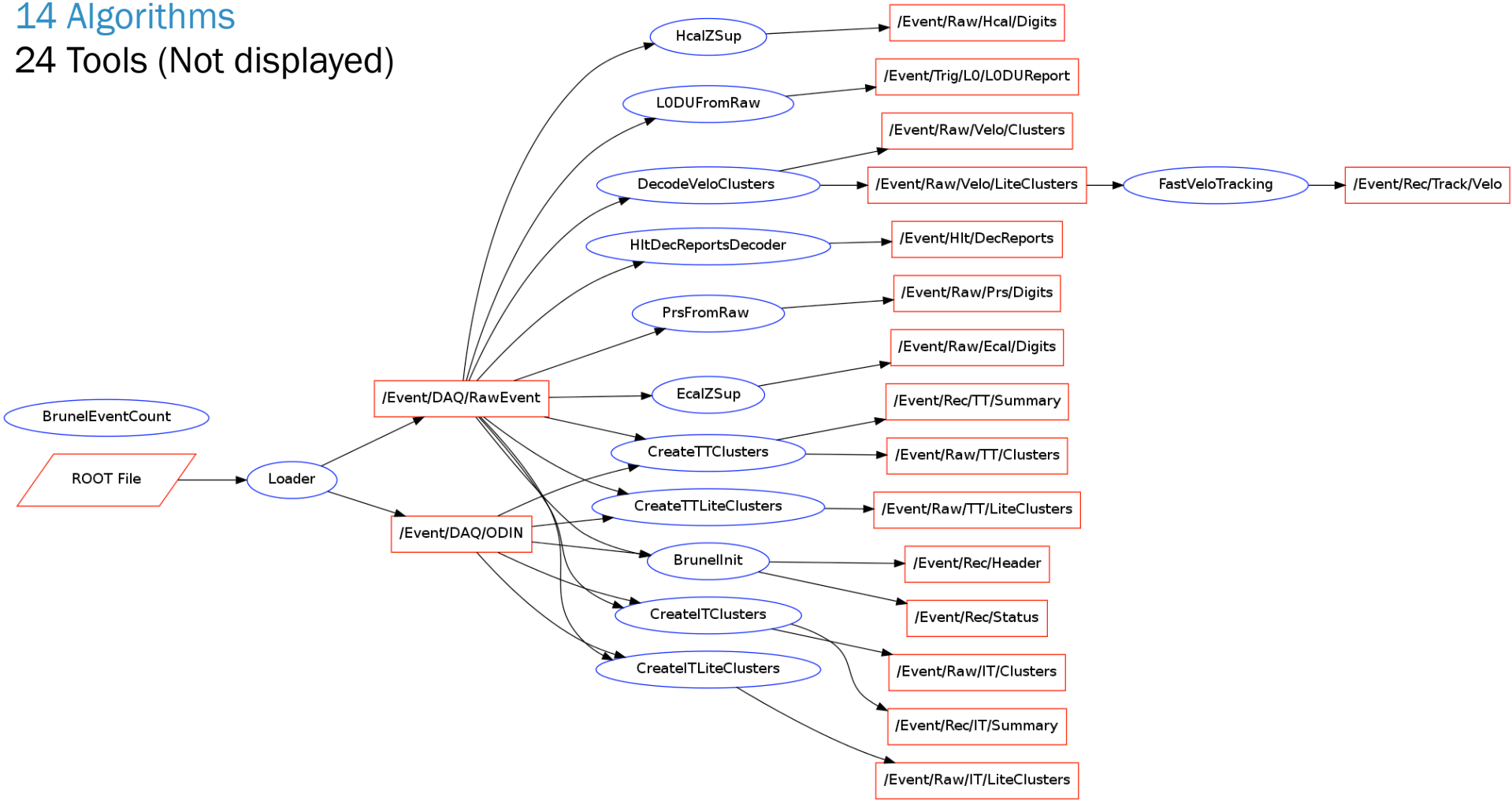
“MiniBrunel”
span within the
LHCb detector



MiniBrunel: Data Dependencies

14 Algorithms

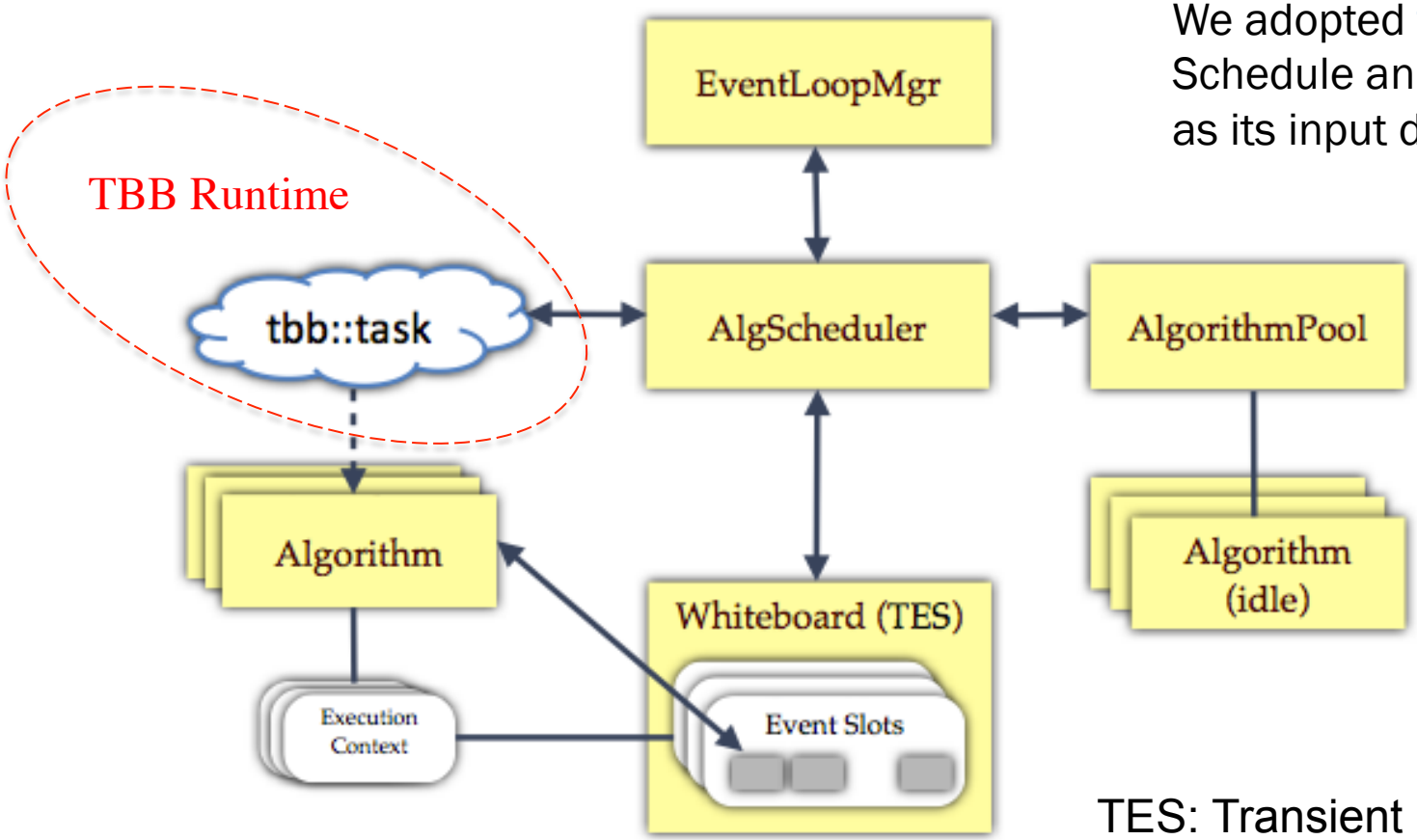
24 Tools (Not displayed)



Control flow dependencies not displayed

Components Overview

- New components added to Gaudi to support concurrency
  - E.g. Scheduler, Whiteboard, AlgorithmPool
- Existing components upgraded
  - E.g. ToolSvc, EventLoopMgr



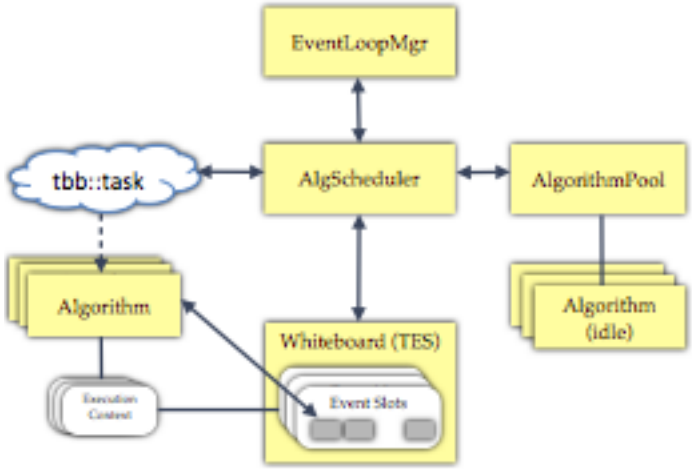
We adopted forward scheduling: Schedule an algorithm as soon as its input data are available

TES: Transient Event Store

Concurrent Gaudi Basic Ideas

New components' design centred around

- Announced algorithm dependencies
- "Whiteboard" for algorithm-communication
- tbb::task for actual algorithm execution

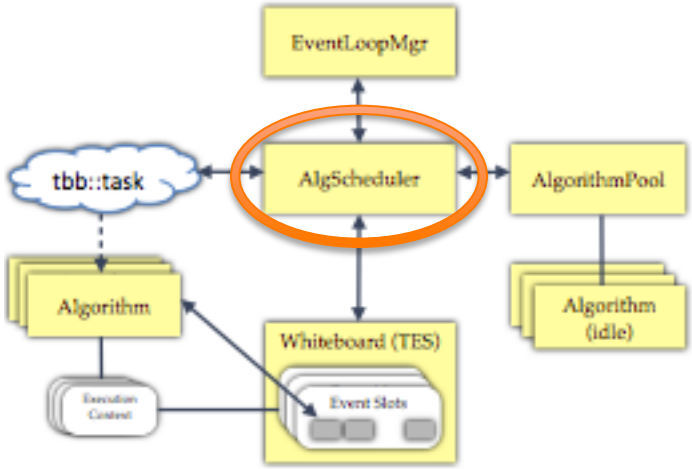


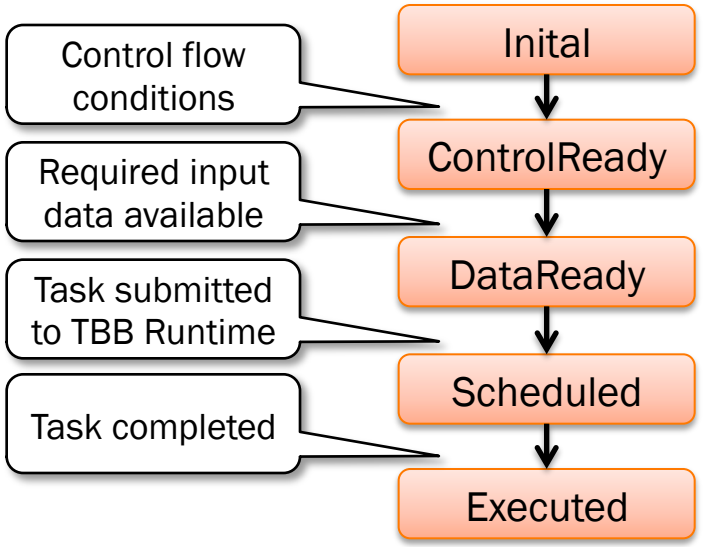
- More details on
  http://concurrency.web.cern.ch/GaudiHive

The Forward Scheduler

Keeps the state of each algorithm for each event

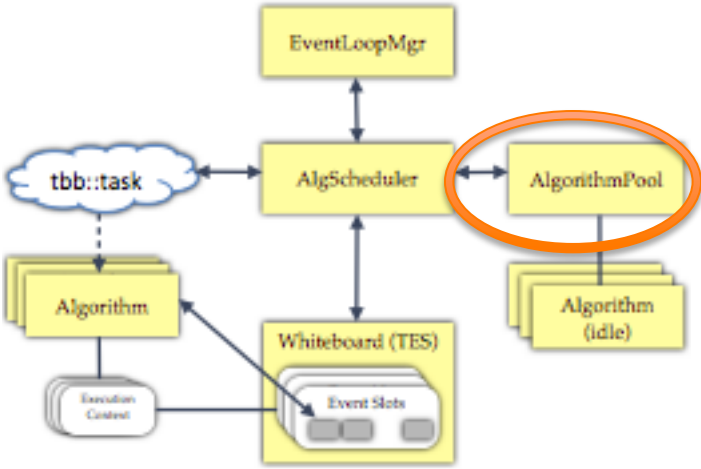
- Simple finite state machine
- Receive new events from loop manager
- Interrogate Whiteboard for new DataObjects
- Pull algorithms from AlgorithmPool if they are available
- Prepare a tbb::task for execution
- Update once tbb:task finished





Algorithm Pool

The **AlgorithmPool** handles (non-) thread-safety of algorithms



- Gives away instances to run, retrieves finished algorithms
- Clones algorithms
  - Number depends on code re-entrancy:
    non re-entrant (1 copy only)
    non re-entrant (use n copies)
    fully re-entrant (re-use same instance n times)
- Allow for exclusive resource checking
  e.g. if 2 algorithms use a non re-entrant external library, only one at the time can run.

- Algorithms' and resources' thread-safety can be tackled one by one

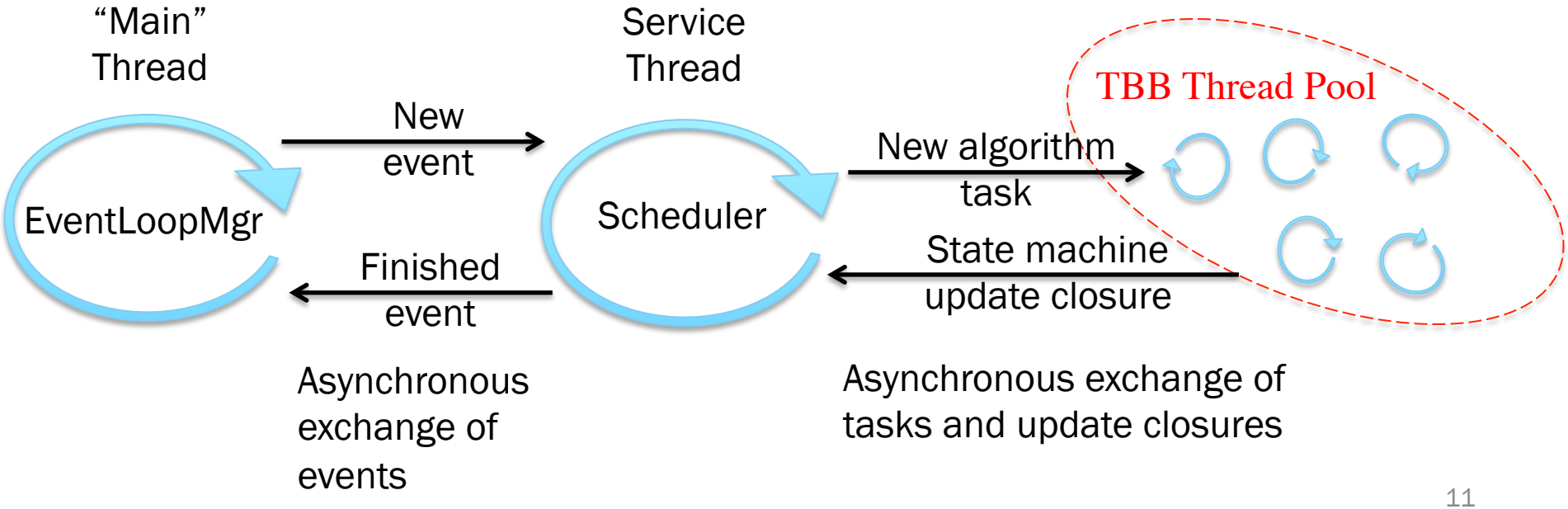
Service Threads

An additional "service" thread (outside the TBB pool) is spawned:

- Host the scheduler method to update the state machine when an algorithm has run. If no work is available, it sleeps.

The "main" thread manages the event loop ("little more than an event factory"). While the scheduler processes the events, it sleeps.

Other service threads existed and continue to exist (e.g. conditions watchdogs)



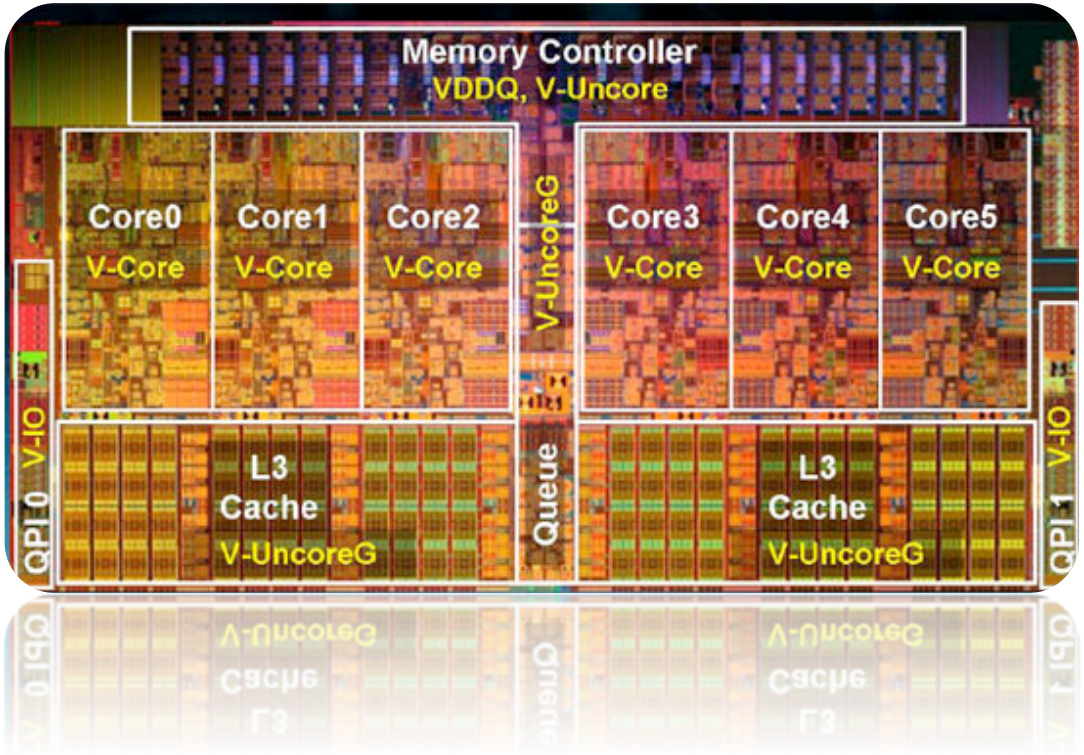
Code Changes: Executive Summary

- **Algorithm dependencies**
  - Data dependencies: announced by the algorithms themselves
- **Global data structures**
  - A few objects served as back-door communication channels bypassing the official (event data) channel
- **Fix assumptions of only one event at a time**
  - Meaning of many global incidents radically changed (e.g. BeginEvent)
  - Raw Data Conversion Caches and their cleanup

For more details see
"Preparing HEP Software for Concurrency"
on Thursday, 11:22 in Track 5 (Effectenbeurszaal)

Measuring Performance

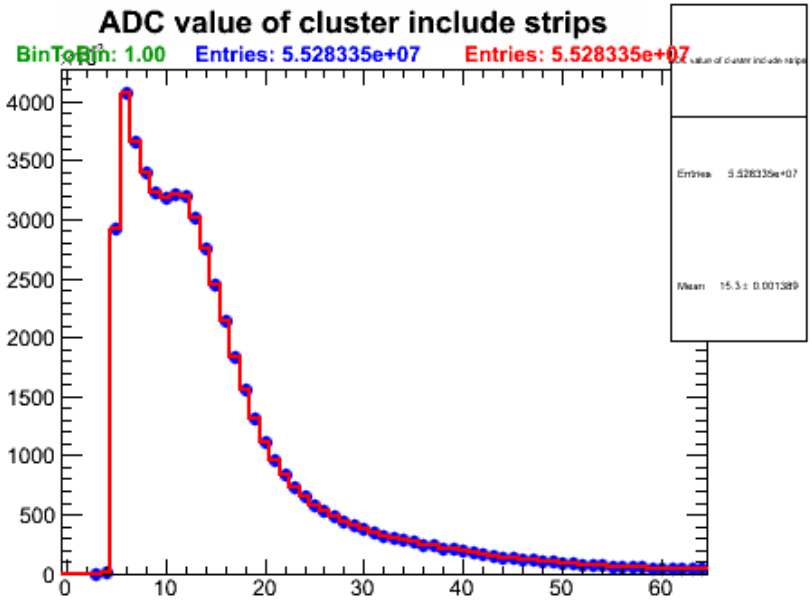
- The Testbed
  - 10k events (60k for the physics performance estimation)
  - SLC6, gcc46
  - TCMalloc
  - Xeon L5640 @2.27 GHz
  - 2 sockets 6+6 HT Cores each (Westmere)



Output Validation

- Only successfully tested software is working software
- Our test case: LHCb standard set of data quality monitoring histograms
- Necessary but not sufficient to guarantee production quality results
- Check histograms for serial and concurrent version (high number of simultaneous events and algorithms)

Example of data monitoring histogram: ADC counts.



All standard histograms identical bin by bin

Runtime Overhead

There is an overhead when using new components designed for concurrency when limiting to one worker thread only (as ~expected)

Timing for the event loop only (no initialisation/finalisation):

Serial Gaudi (no new components) ….. 72.9 s

Concurrent Gaudi 1 evt in flight …….. 97.7 s

Concurrent Gaudi 2 evts in flight ……. 73.9 s

Concurrent Gaudi 10 evts in flight …... 72.3 s

1 worker thread = 1 algorithm at a time

Frequency of task queue updates is too small to keep worker thread busy with only one event in flight

2 events in flight: enough to get rid of 'starvation'

Does it help with memory consumption?

Running mode:

- 1 clone per event in flight of 3 longest running algorithms
- Full TBB thread pool (24 threads)
- Limit algorithms in flight to 6

Resident Set Size at the end of the event loop (no finalisation):

Serial Gaudi (no new components) ….. 478 MB

Concurrent Gaudi 1 evt in flight …….. 480 MB

Concurrent Gaudi 2 evts in flight ……. 485 MB

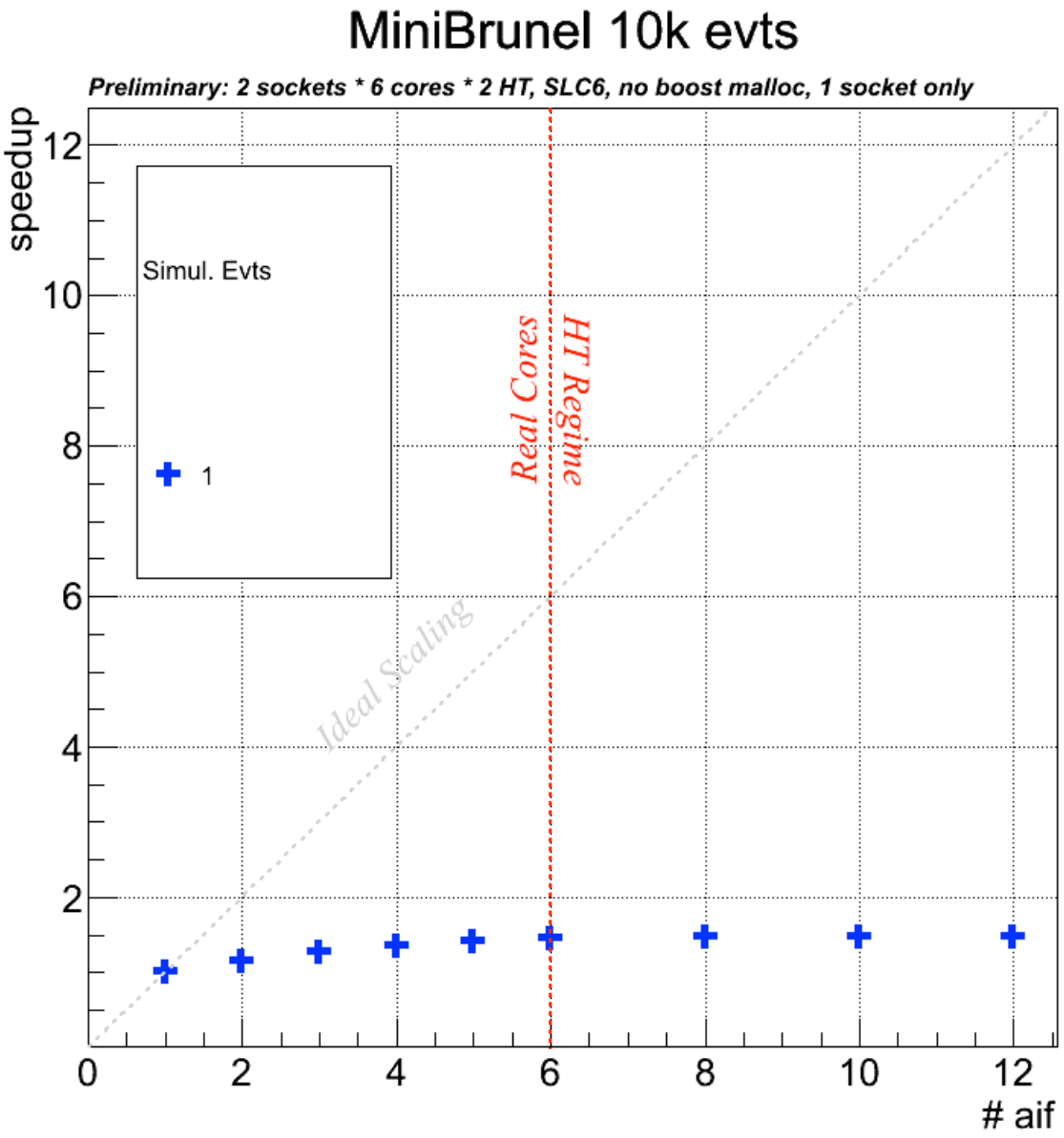
Concurrent Gaudi 10 evts in flight …... 514 MB

Note: Not full LHCb events but MiniBrunel events.

6 algorithms running simultaneously

Memory: multithreaded solution is cheap!

Scaling on One Processor

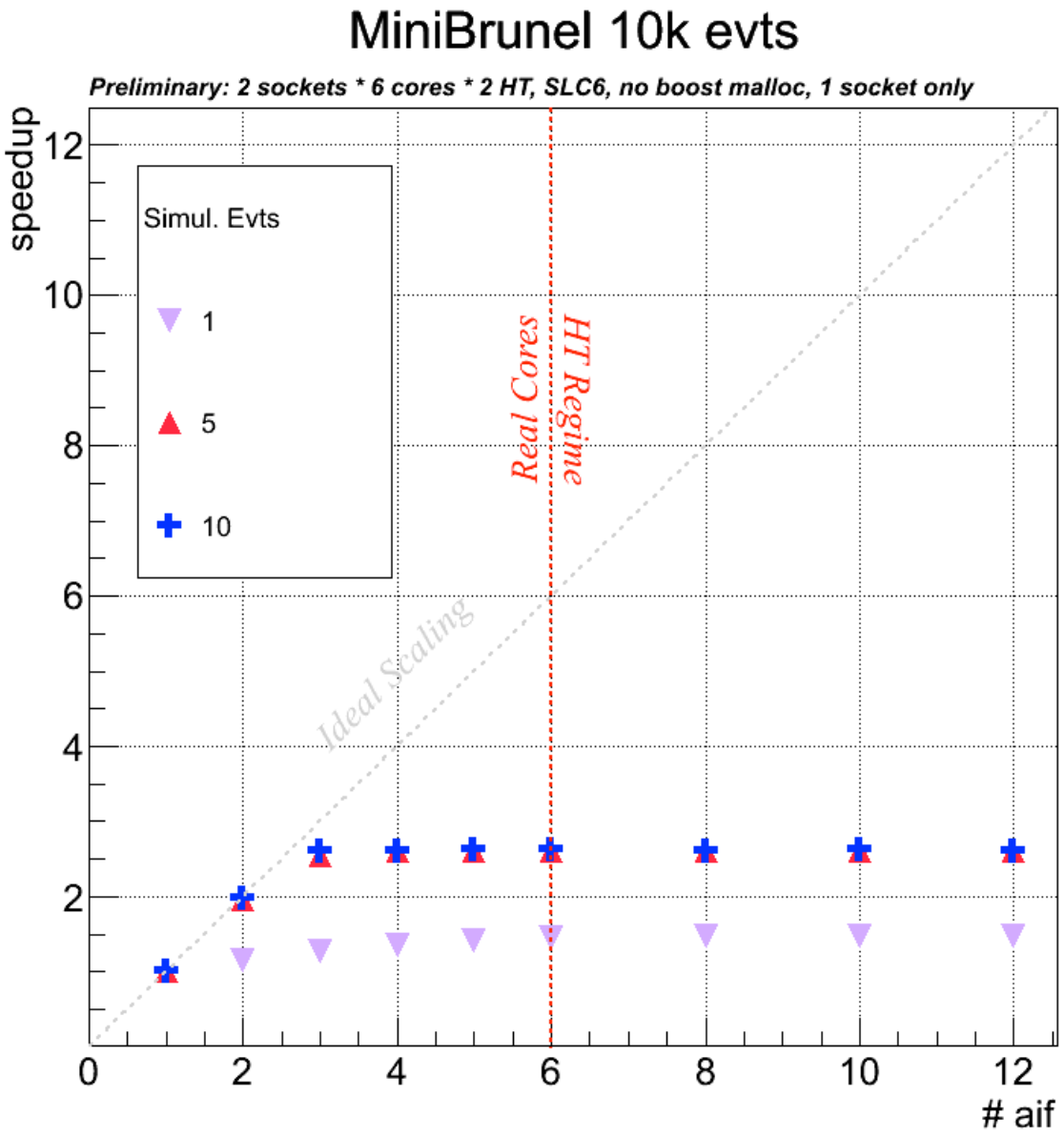


N algorithms simultaneously

Maximum Speedup: ~30%

Limited by **critical path** in algorithm dependencies (Amdahl's Law)

Scaling on One Processor

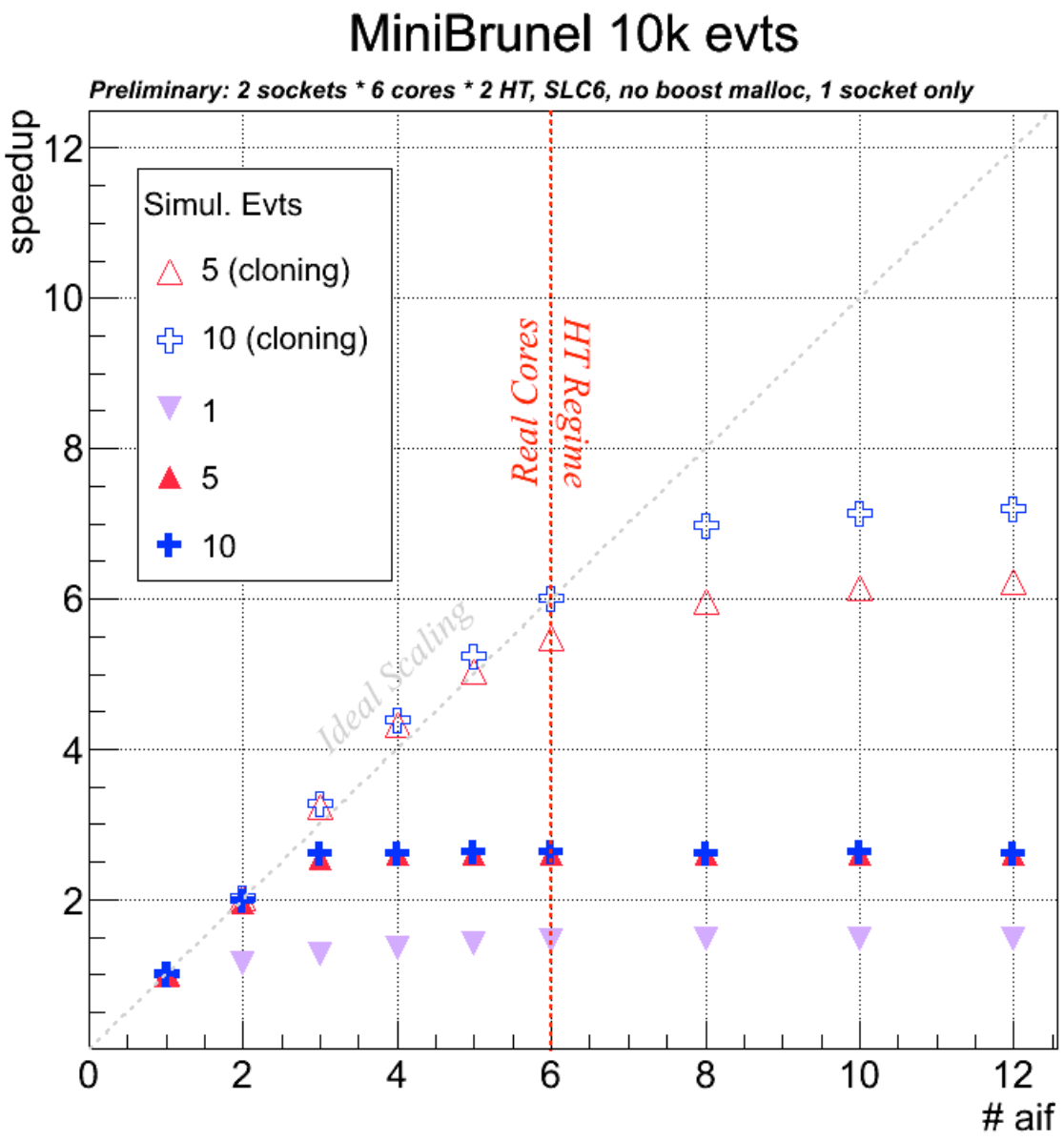


Multiple events in flight
N algorithms simultaneously

Maximum Speedup: 2.5x

Limited by algorithms being usable only for one event at a time

Scaling on One Processor



Multiple events in flight
Clone 3 most time consuming algs (1 copy per event in flight)

Linear scaling of speedup up to number of physical cores

10 events in flight already enough for peak performance (thanks to HT)

Conclusions

A concurrent framework design was presented

- Supporting concurrency at all levels

Finished all developments necessary for the test case

- Framework: components for MT execution (Scheduler, EventLoopManager) and integration with TBB runtime
- “User” code: input declaration, thread-safety fixes, compatibility with >1 event simultaneously processed

Outcome very successful

- Serial and concurrent Minibrunel yield identical physics output
- Concurrent MiniBrunel scales linearly on a single die
- Negligible increase of memory consumption

Future activities

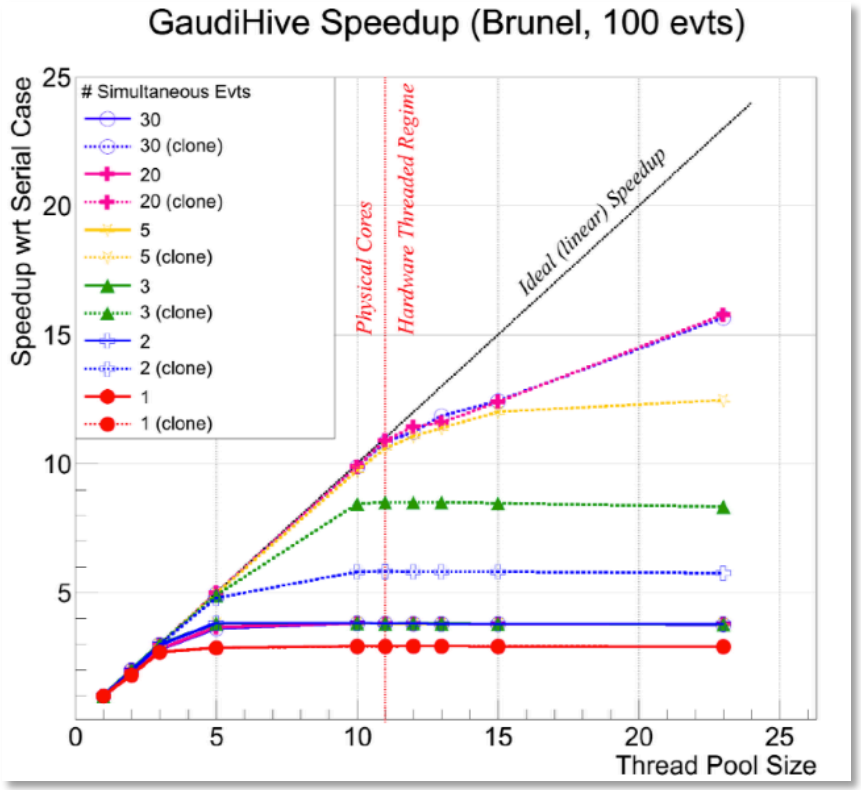
- Extend the test scenario to a bigger LHCb example
- Complete the set of thread-safe classes

Thanks to the LHCb core software team for the support!

Additional Material

The Past (2012): CPUCrunchers Demonstrator

- Emulate an LHCb full reconstruction workflow with CPUCrunching algorithms (no real work done, just keep cpus busy)
- Explore expected behaviour
- Demonstrate potential of the multithreaded approach



In Nov 2012

Evolving LHC Data Processing Frameworks for Efficient Exploitation of New CPU Architectures
B. Hegner at al, IEEE-NSS 2012

Useful Material

Project Page on the Concurrency Forum Site:

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

Main Twikipage:

https://twiki.cern.ch/twiki/bin/view/C4Hep

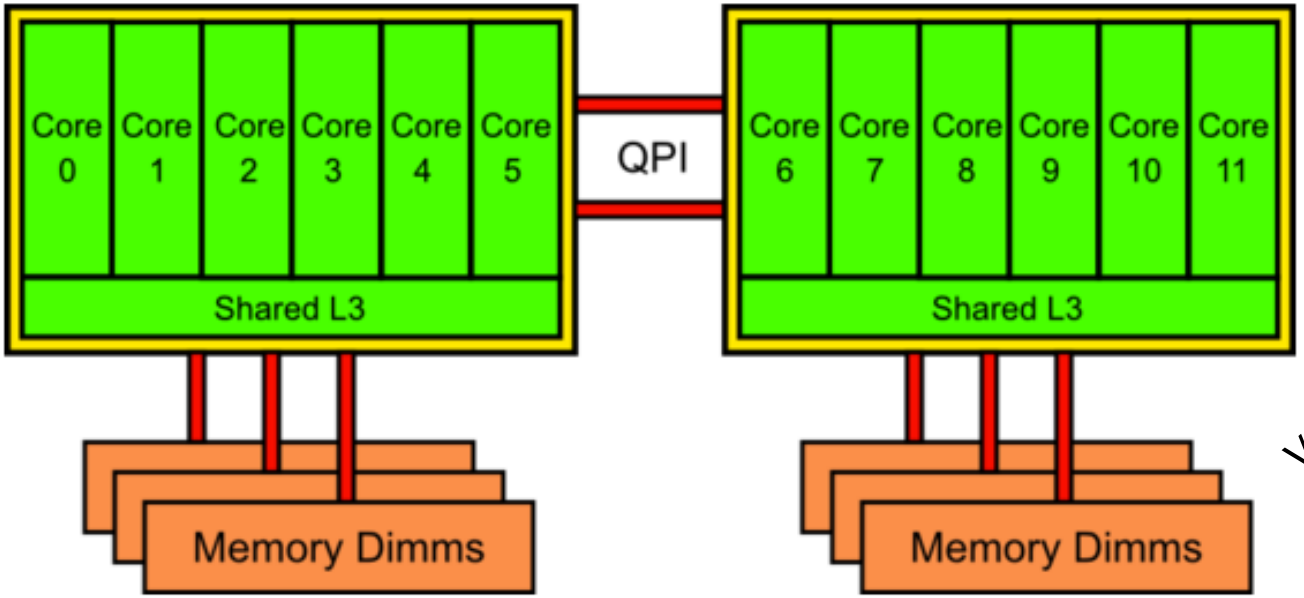
Git Repository Web Interface:
http://lcgapp.cern.ch/git/GaudiMT/

Jira:

https://sft.its.cern.ch/jira/browse/CFHEP

Scaling on Two Sockets

- Behaviour of the application on a full NUMA* node is not trivial
  - E.g.: remote DRAM access, cross-socket caches synchronisation…



* NUMA = Non-Uniform Memory Access

Numa Effects

- Run with 10 events in flight and 6 threads
- Use the “taskset” command to assign cpus to a process
- Start with 6 cpus on one socket, move them one by one to the other
- Measure event loop time and use perf to count cache misses

Thanks to Vincenzo I. for the fruitful discussions!!