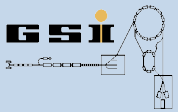


Status of Reconstruction in CBM

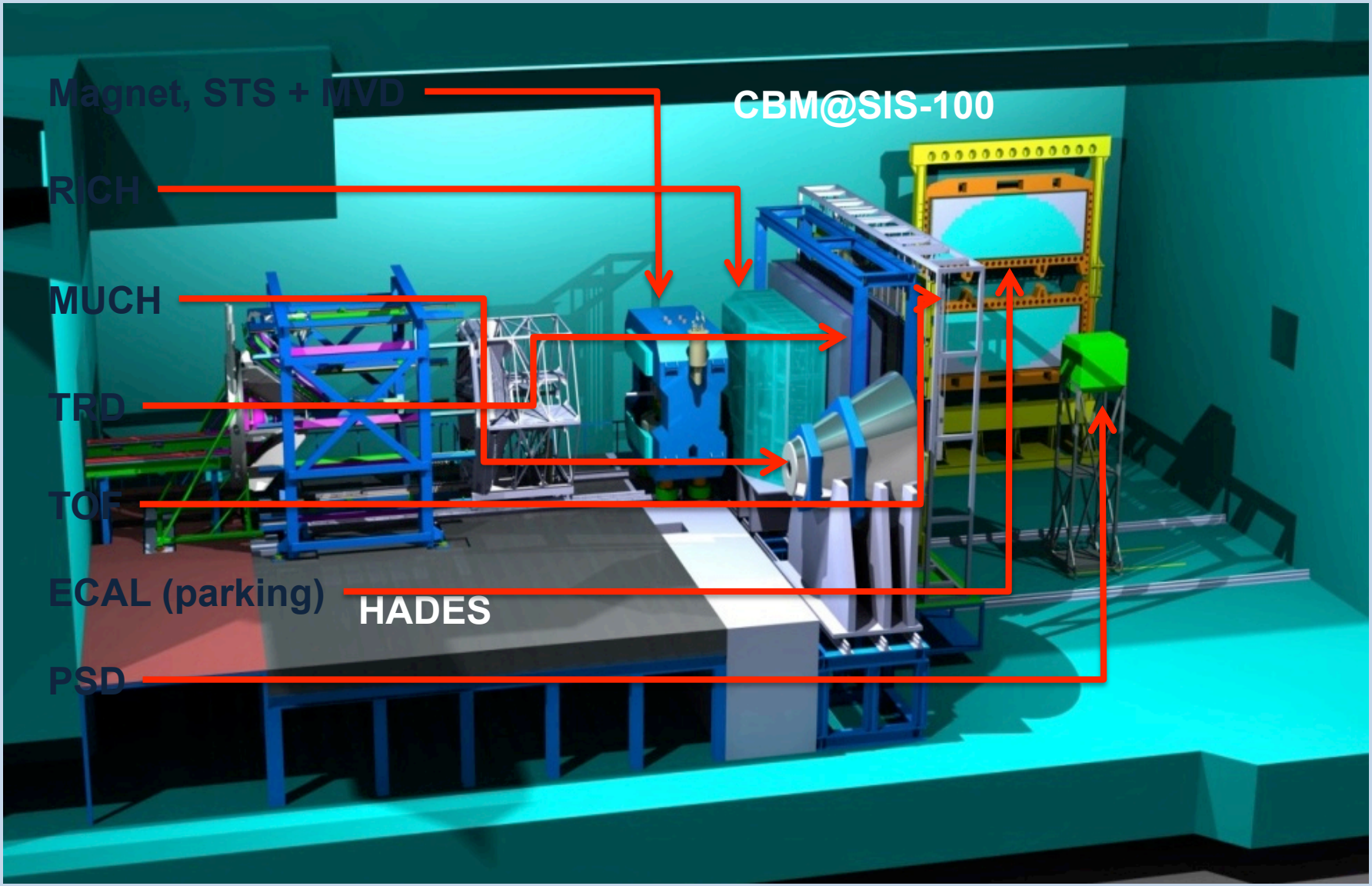
Volker Friese
GSI Darmstadt

5th International Workshop for Future Challenges in Tracking and Trigger Concepts
12 May 2014, Frankfurt, Germany

CBM in a Nutshell

- Compressed Baryonic Matter: a heavy-ion experiment at the future facility FAIR in Darmstadt
- Investigation of strongly interacting matter at extreme net-baryon densities
- Fixed-target operation on extracted beams, 2 – 45 GeV/nucleon
- Large-acceptance spectrometer (dipole + Si main tracker)
- Hadron, lepton and photon ID: RICH, TRD, TOF, ECAL
- Up to 600 charged tracks per collisions in the acceptance
- Observables: yields, spectra, flow, correlations, fluctuations of bulk hadrons, multi-strange hyperons, open charm and charmonium; low-mass di-leptons
- Operation from 2018 on

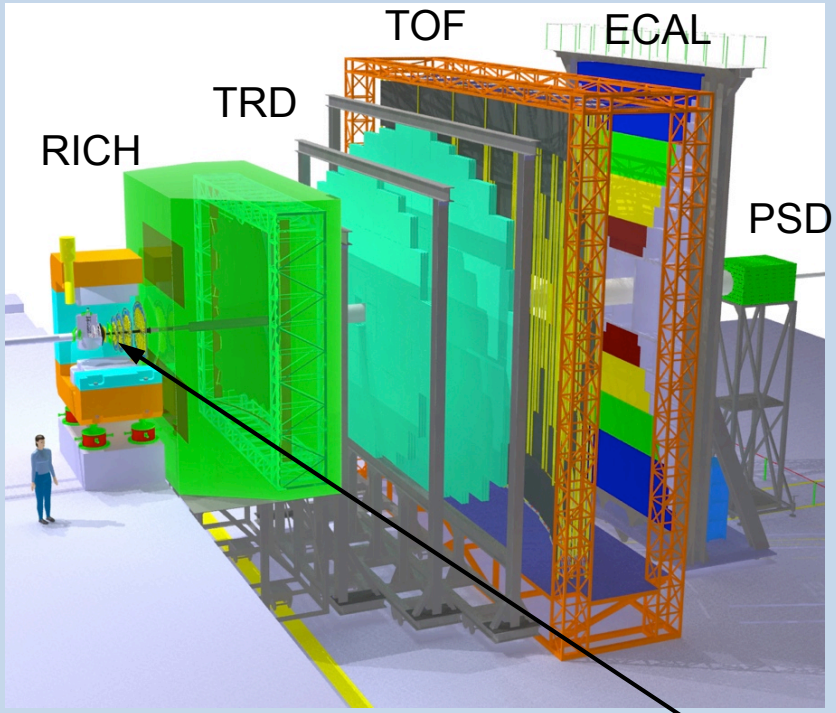
CBM: Experimental Setup



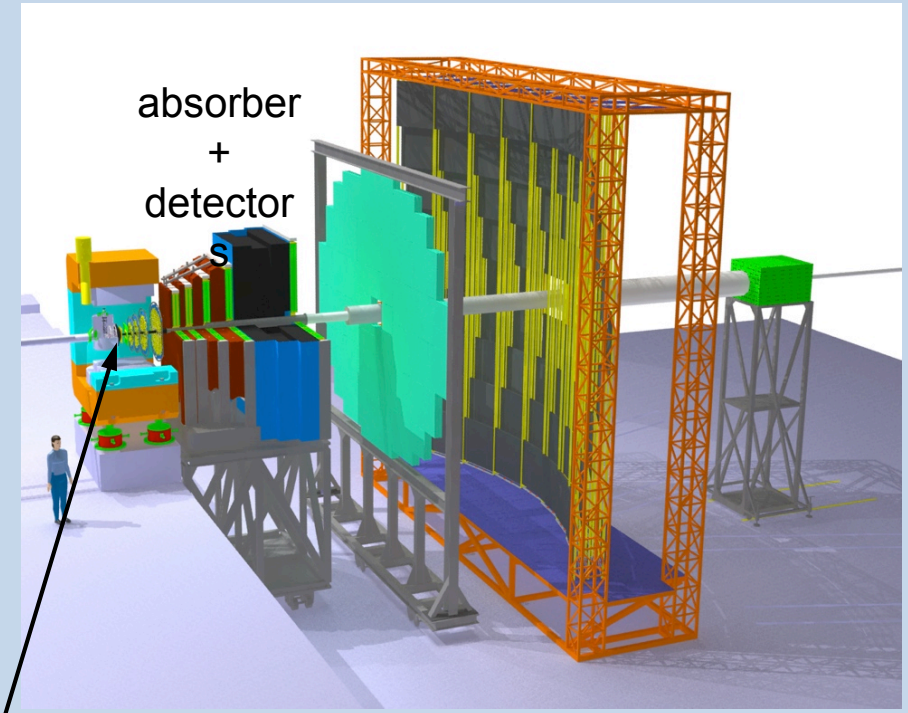
CBM: setup

Versatile setup: foresees moving and exchanging of detector systems

Electron + Hadron setup



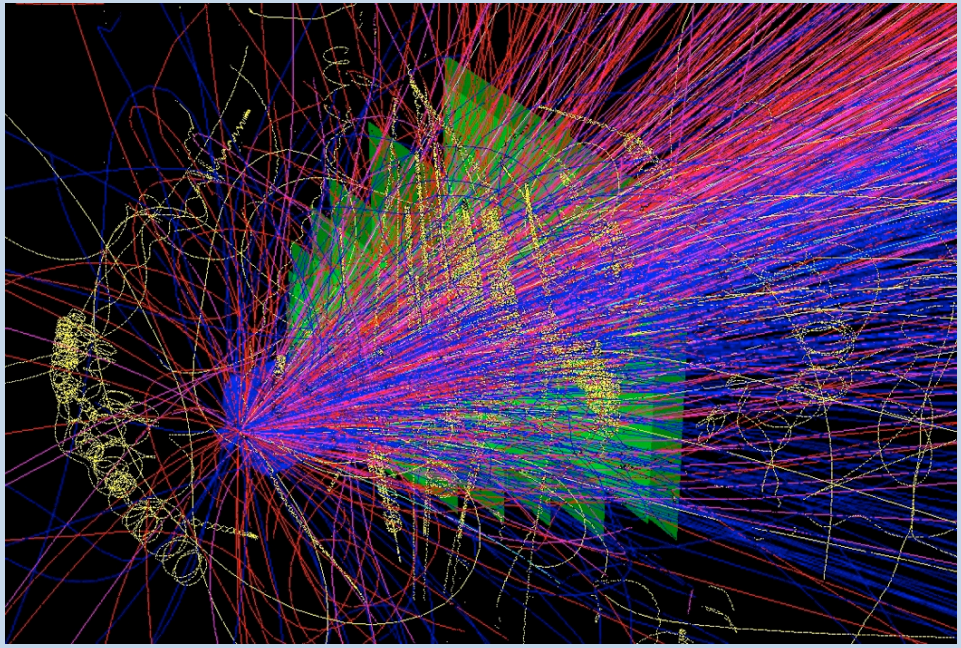
Muon setup

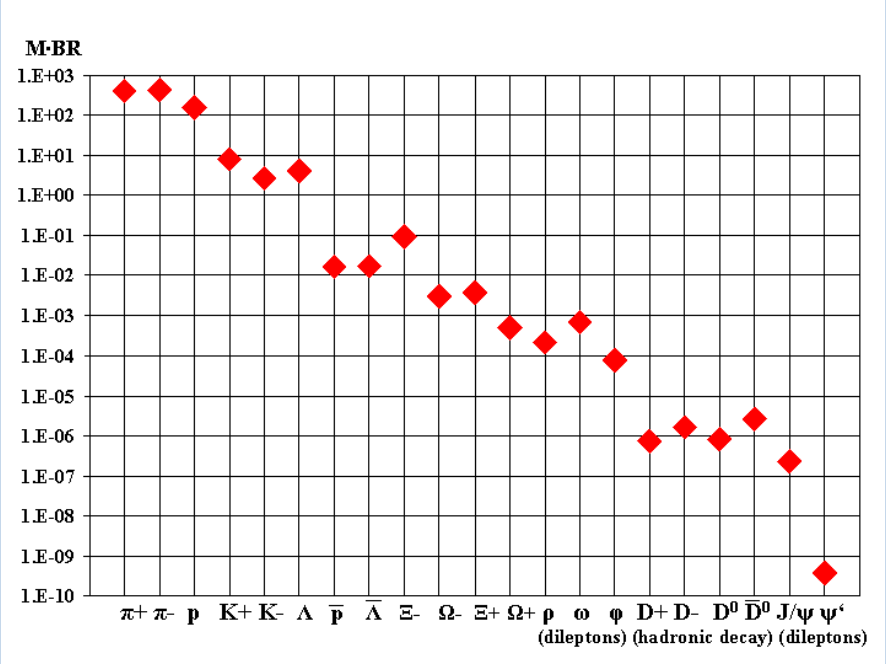


STS+MVD

Rare Probes, High Rates

Simulation of UrQMD Au+Au, central 25 GeV/u



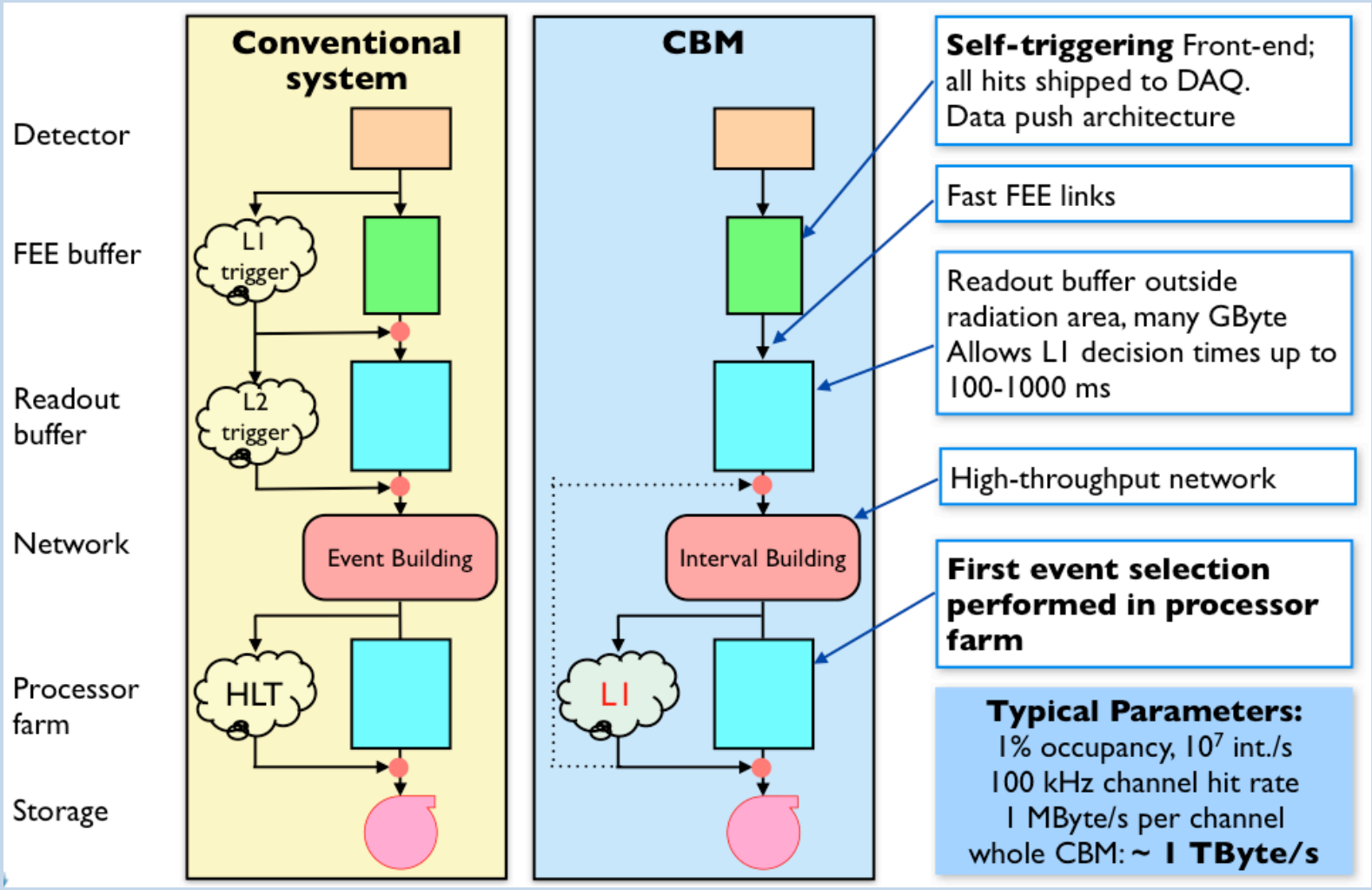


Many of the (most interesting) observables are extremely rare
E.g. J/ψ: M x BR < 10^{-6}
CBM punchline: extreme interaction rates (up to 10 MHz)
requires data suppression by a factor of several hundreds

Trigger Considerations

- Signatures vary qualitatively:
  - local and simple: $J/\psi \rightarrow \mu^+\mu^-$
  - non-local and simple: $J/\psi \rightarrow e^+e^-$
  - non-local and complex: D,Ω->charged hadrons
- For maximal interaction rate, reconstruction in STS is always required (momentum information), but not necessarily of all tracks in STS.
- Trigger architecture must enable
  - variety of trigger patterns (J/ψ: 1% of data, D mesons: 50% of data)
  - multiple triggers at a time
  - multiple trigger steps with subsequent data reduction
- Complex signatures involve secondary decay vertices; difficult to implement in hardware.
- Extreme event rates set strong limits to trigger latency.

CBM Readout Concept



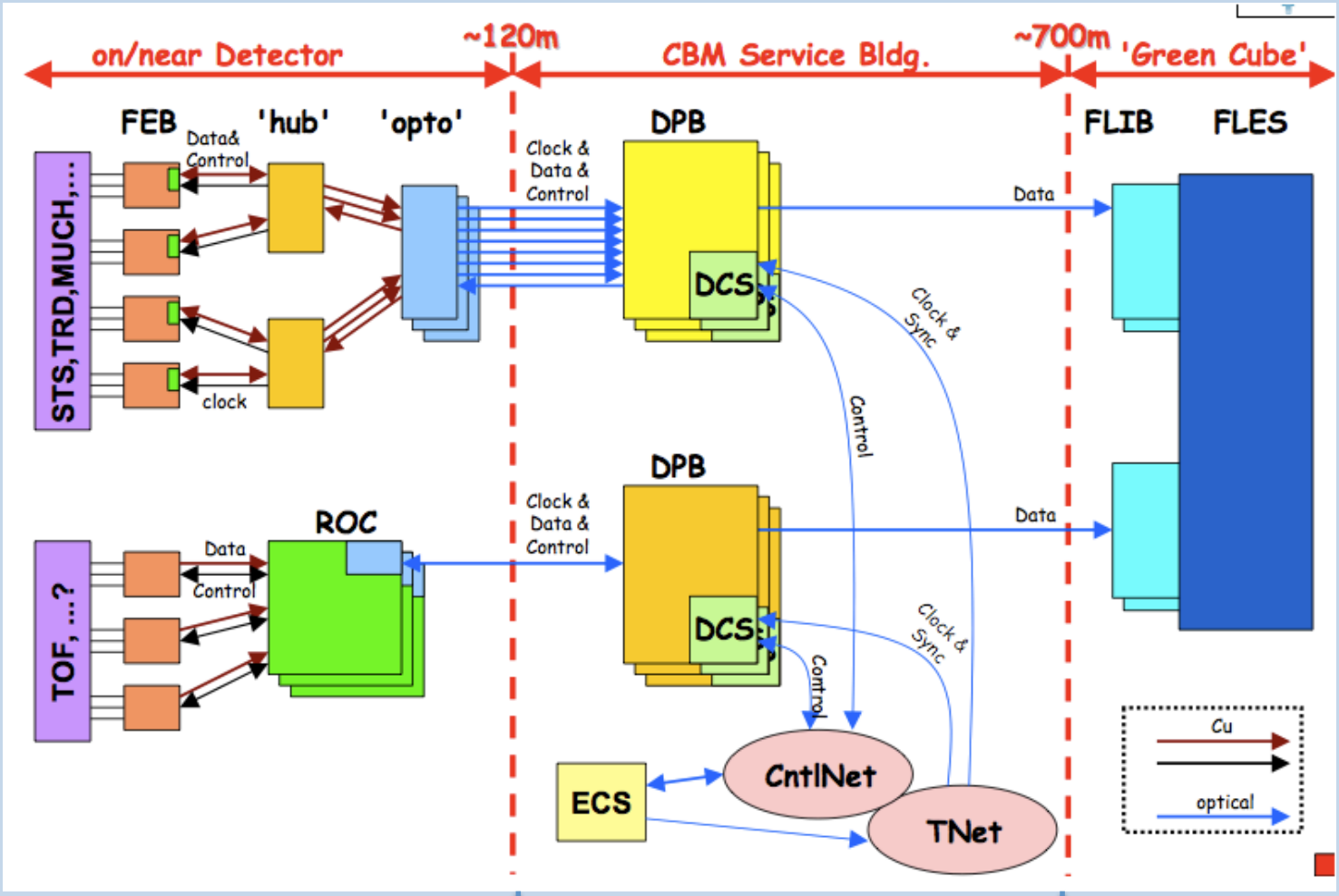
Finite-size FEE buffer: latency limited

throughput limited

Consequences

- The system is limited only by the throughput capacity and by the rejection power of the online computing farm.
- There is no a-priori event definition: data from all detectors come asynchroneously; events may overlap in time.
- The classical DAQ task of „event building“ is now rather a „time-slice building“. Physical events are defined later in software.
- Data reduction is shifted entirely to software: maximum flexibility w.r.t. physics

CBM Readout Architecture

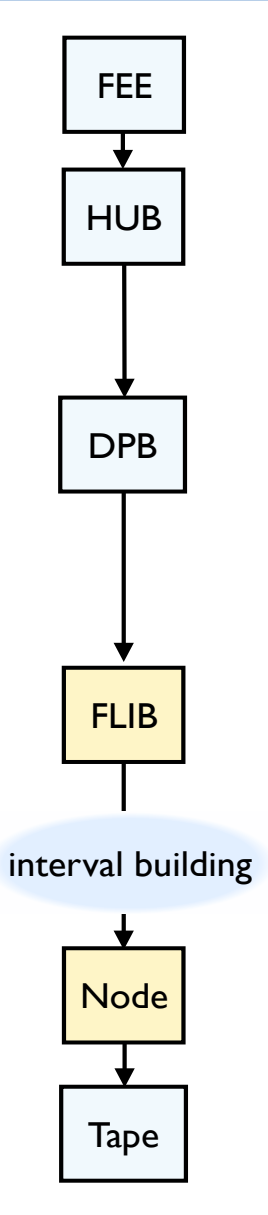


DAQ: data aggregration
time-slice building
(pre-processing?)

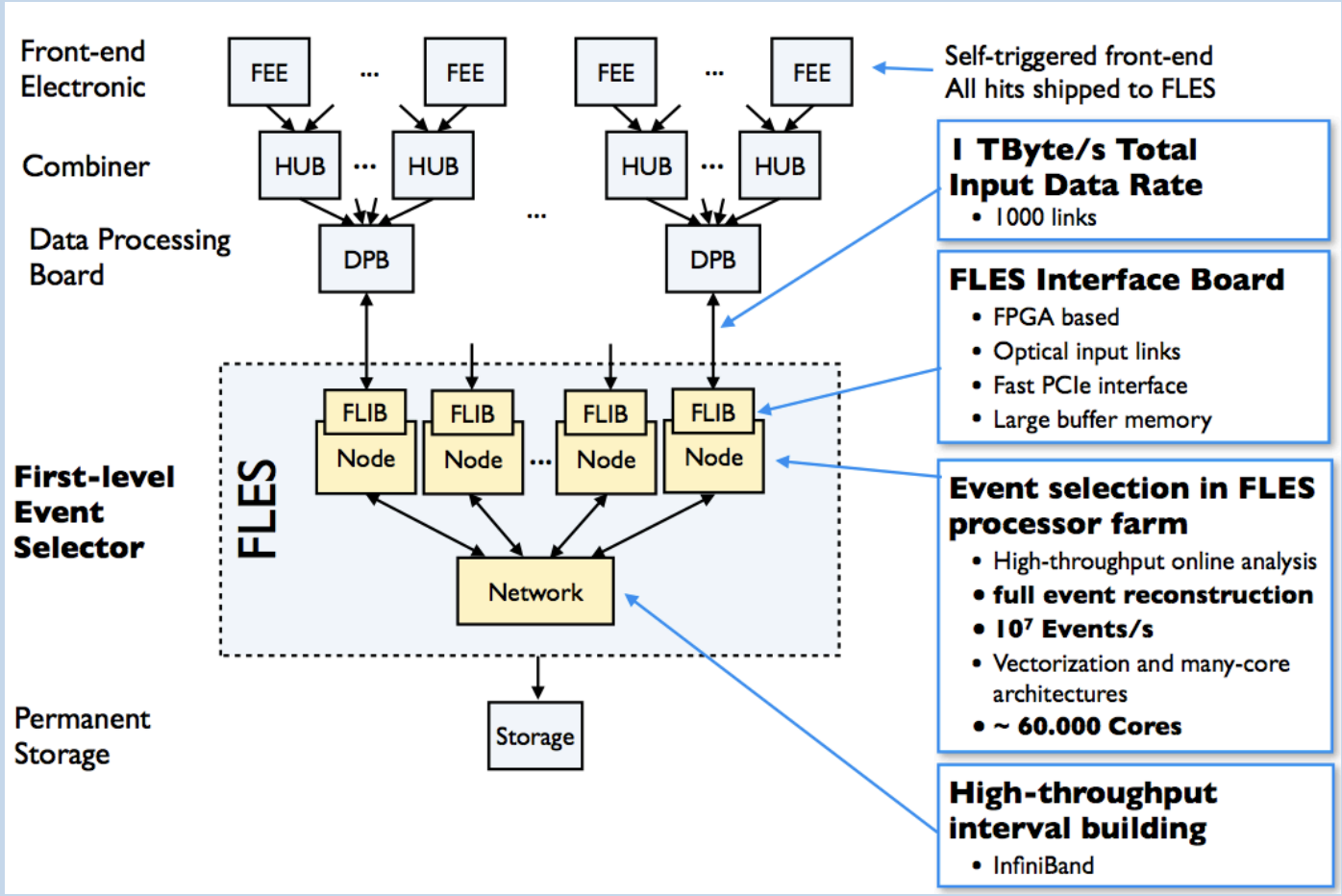
FLES: event
reconstruction
and selection

Components of the read-out chain

- Detector Front-Ends
  - each channel performs autonomous hit detection and zero suppression
  - associate absolute time stamp with hit, aggregrate data
  - data push architecture
- Data Processing Board (DPB)
  - perform channel and segment local data processing
    - feature extraction, time sorting, data reformatting , merging input streams
  - time conversion and creation of microslice containers
- FLES Interface Board (FLIB)
  - time indexing and buffering of microslice containers
  - data sent to FLES is concise: no need for additional processing before interval building
- FLES Computing Nodes
  - calibration and global feature extraction
  - full event reconstruction (4-d)
  - event selection



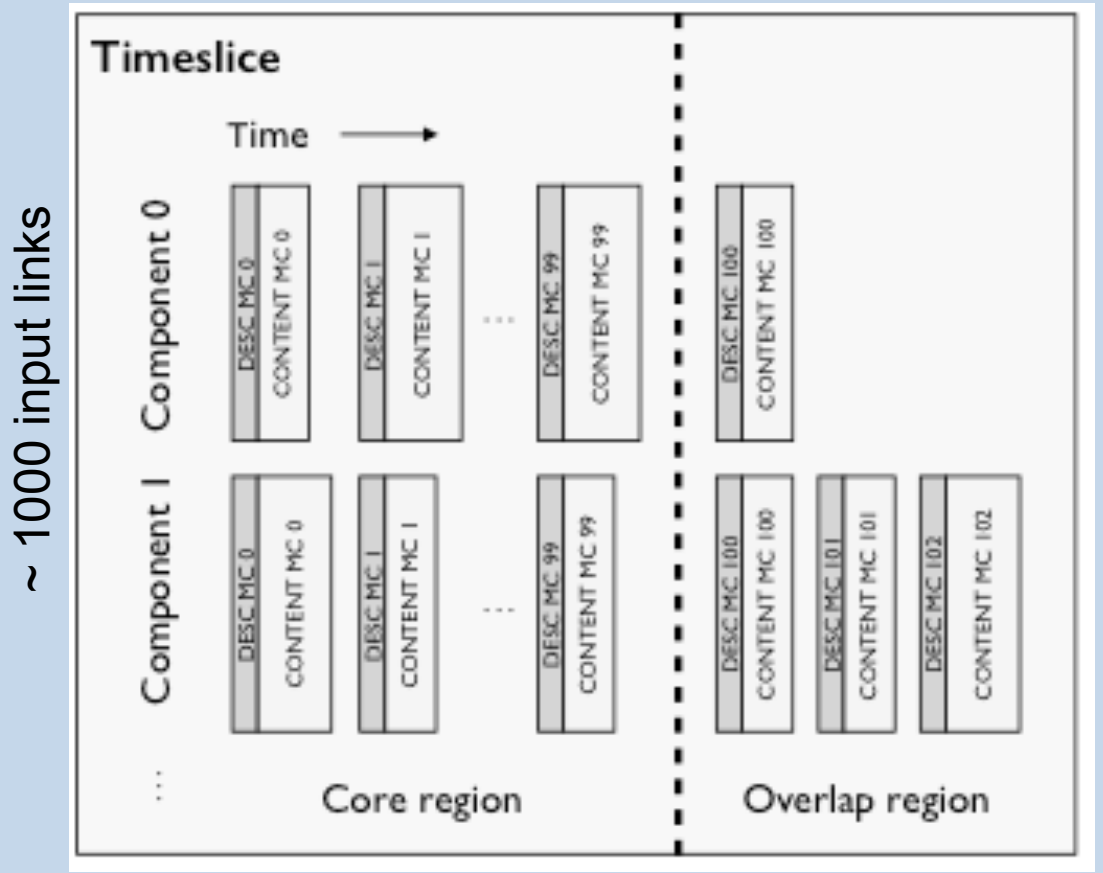
CBM online data flow



The data chunk given to each computing node is a time slice (all experiment raw data in a given time interval). Size will be adjusted to hardware specifications.

Raw Data API

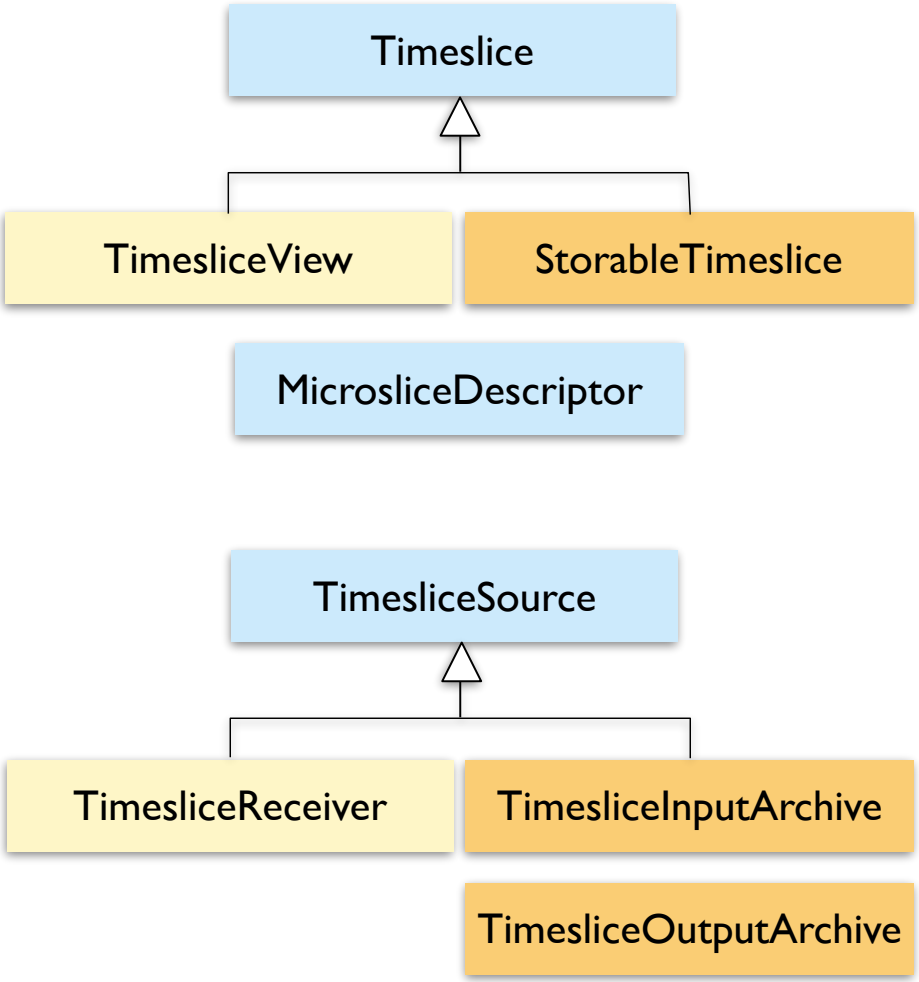
- Timeslice API defined: 2-D array of microslices in time and components
- contains all detector raw data within given time interval
- this will be the input for reconstruction on the compute nodes of the FLES



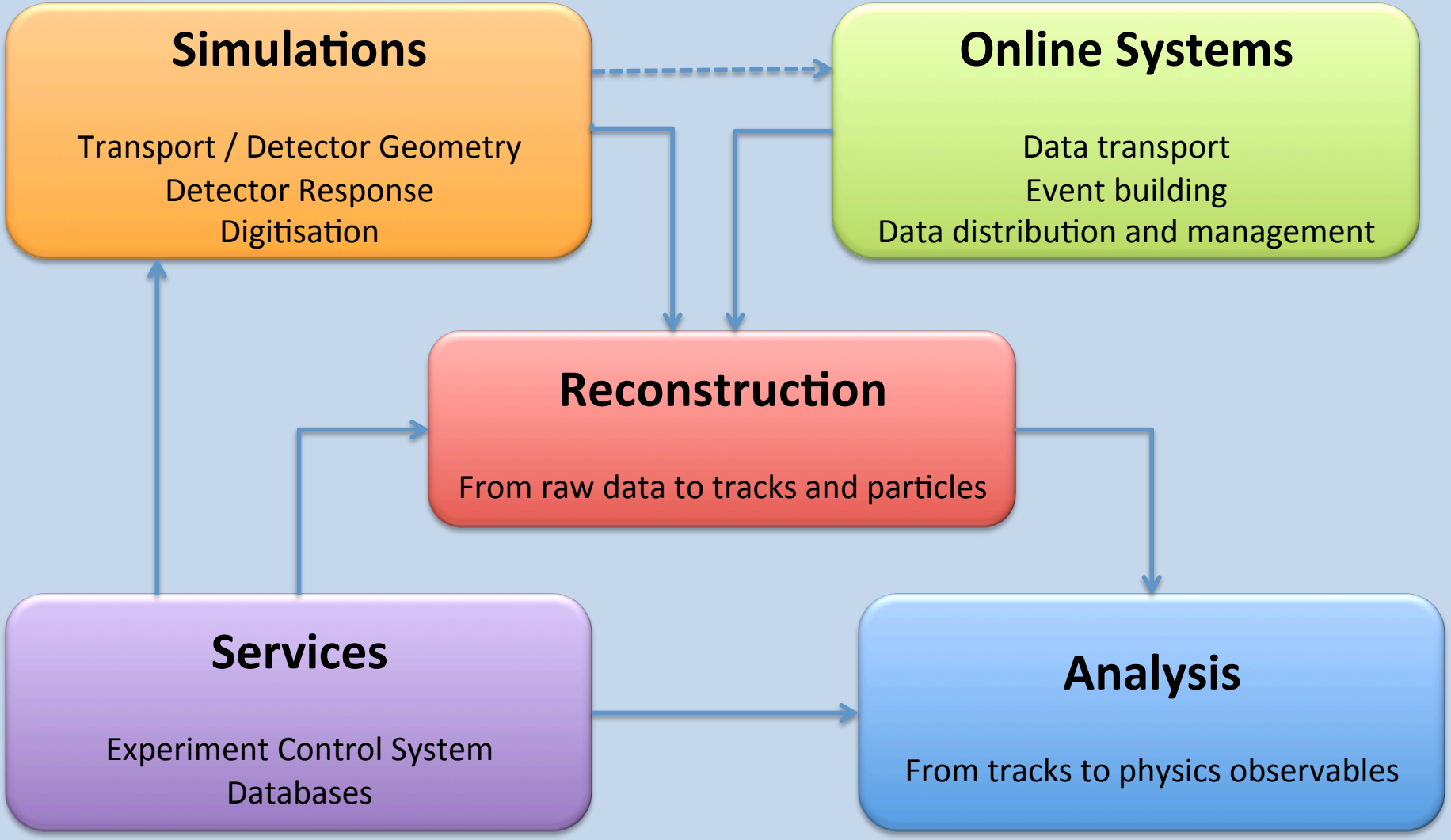
Timeslice API

- Efficient online access on FLES compute node
  - Directly uses RDMA receive buffers via shared memory
- Supports serialization
- Identical access to online and stored timeslices
- StorableTimeslice can be filled with simulated data
- Timeslice consumer: separate framework (e.g., based on CBMRoot)

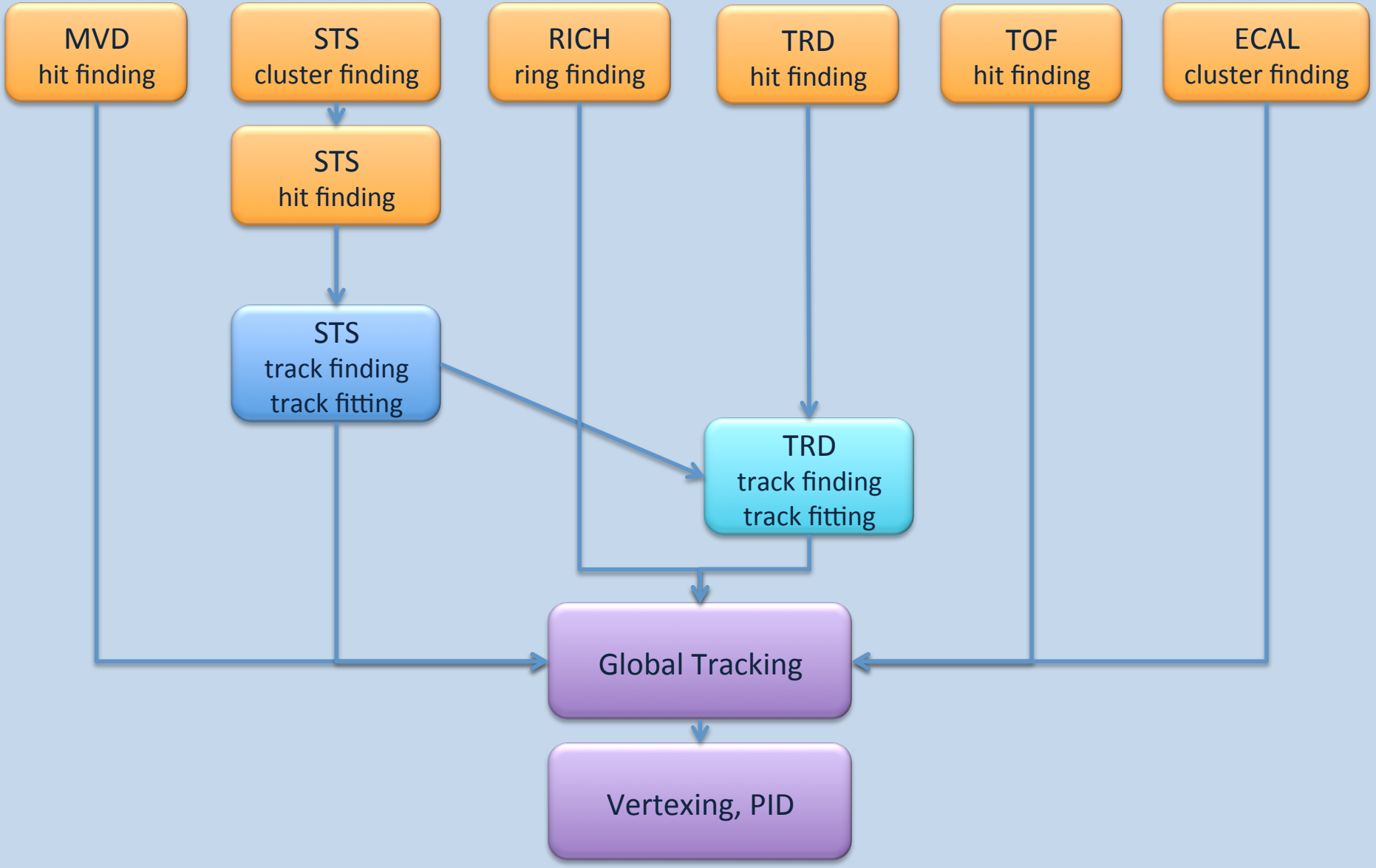
Main API Classes



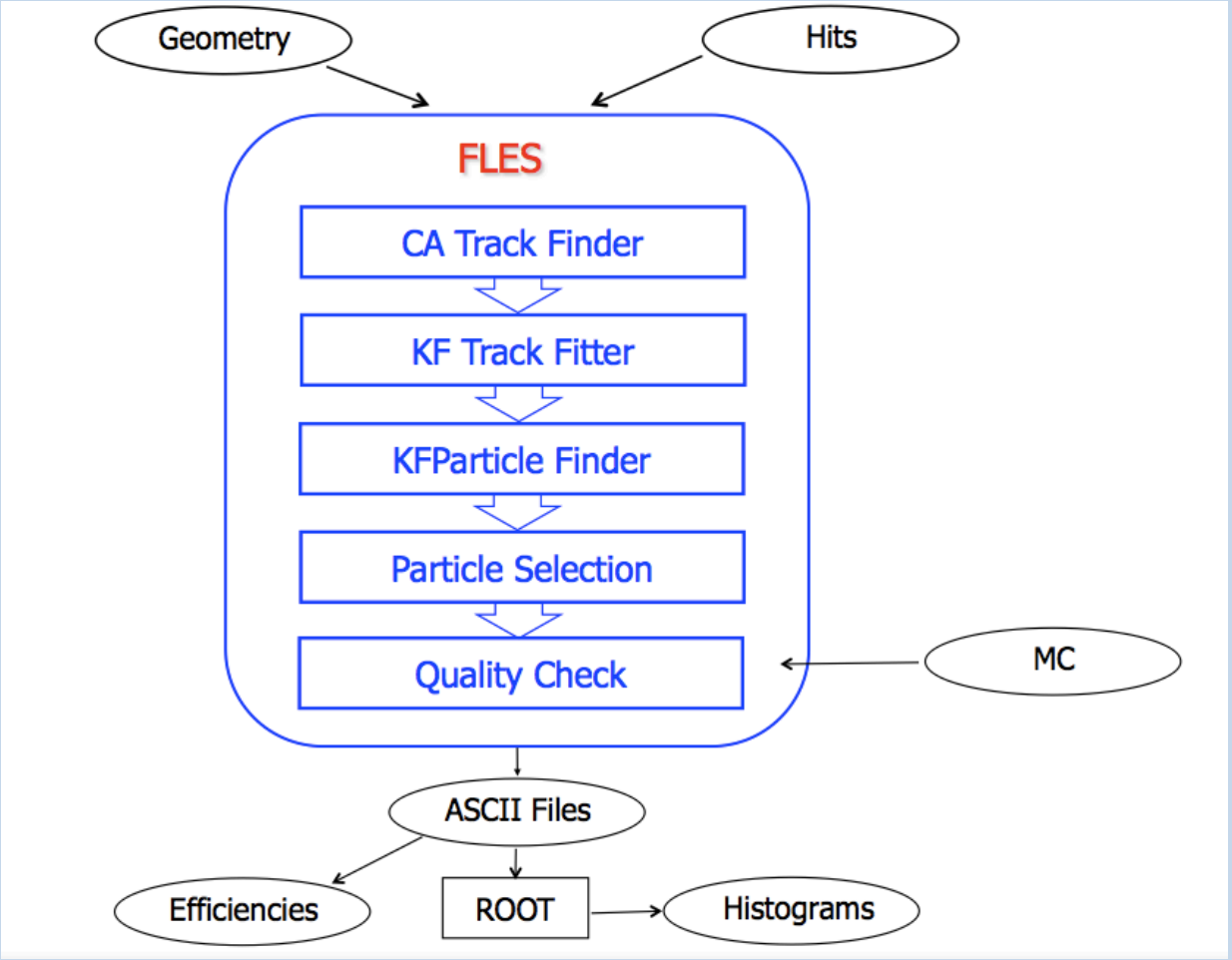
Computing Tasks



Reconstruction Chain



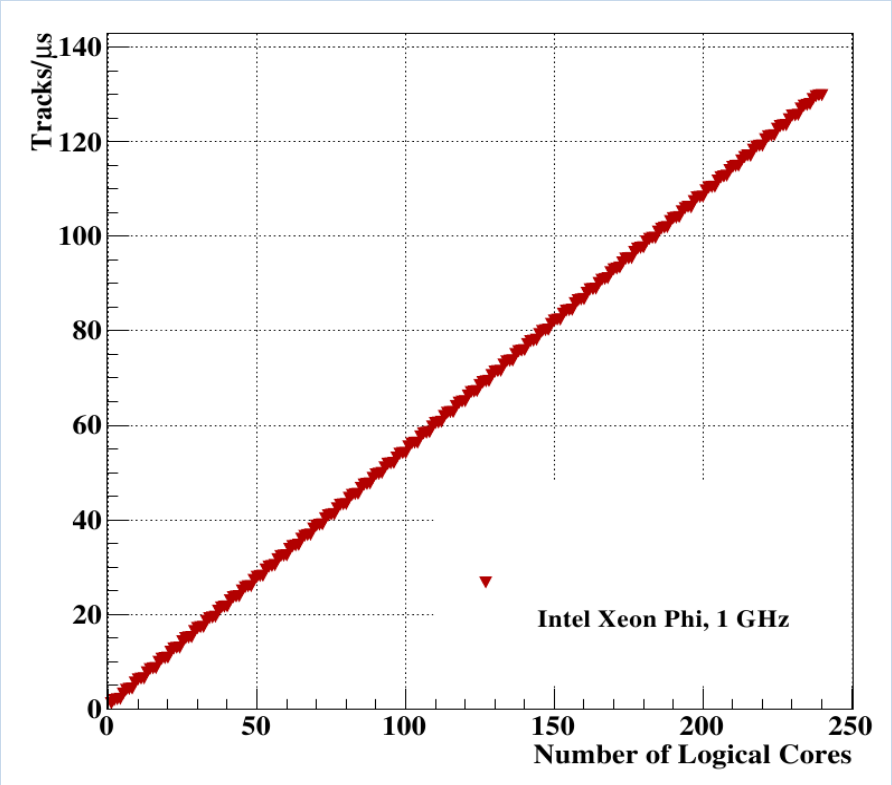
Development of FLES software



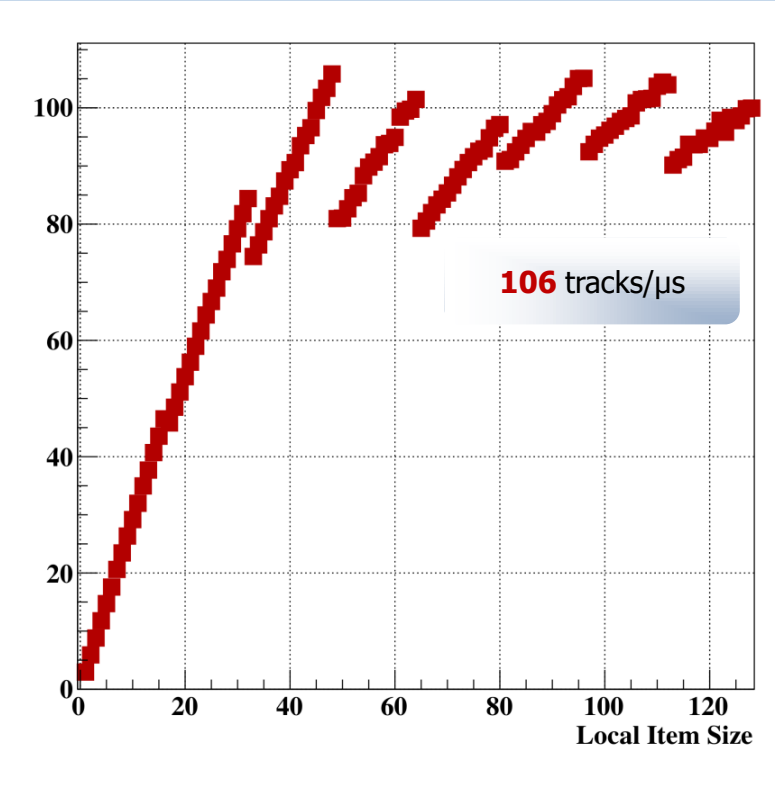
Highly performant algorithms for parts of the CBM reconstruction were developed „standalone“, i.e. outside of the cbmroot framework.

Architectural Studies

CPU + Intel Phi



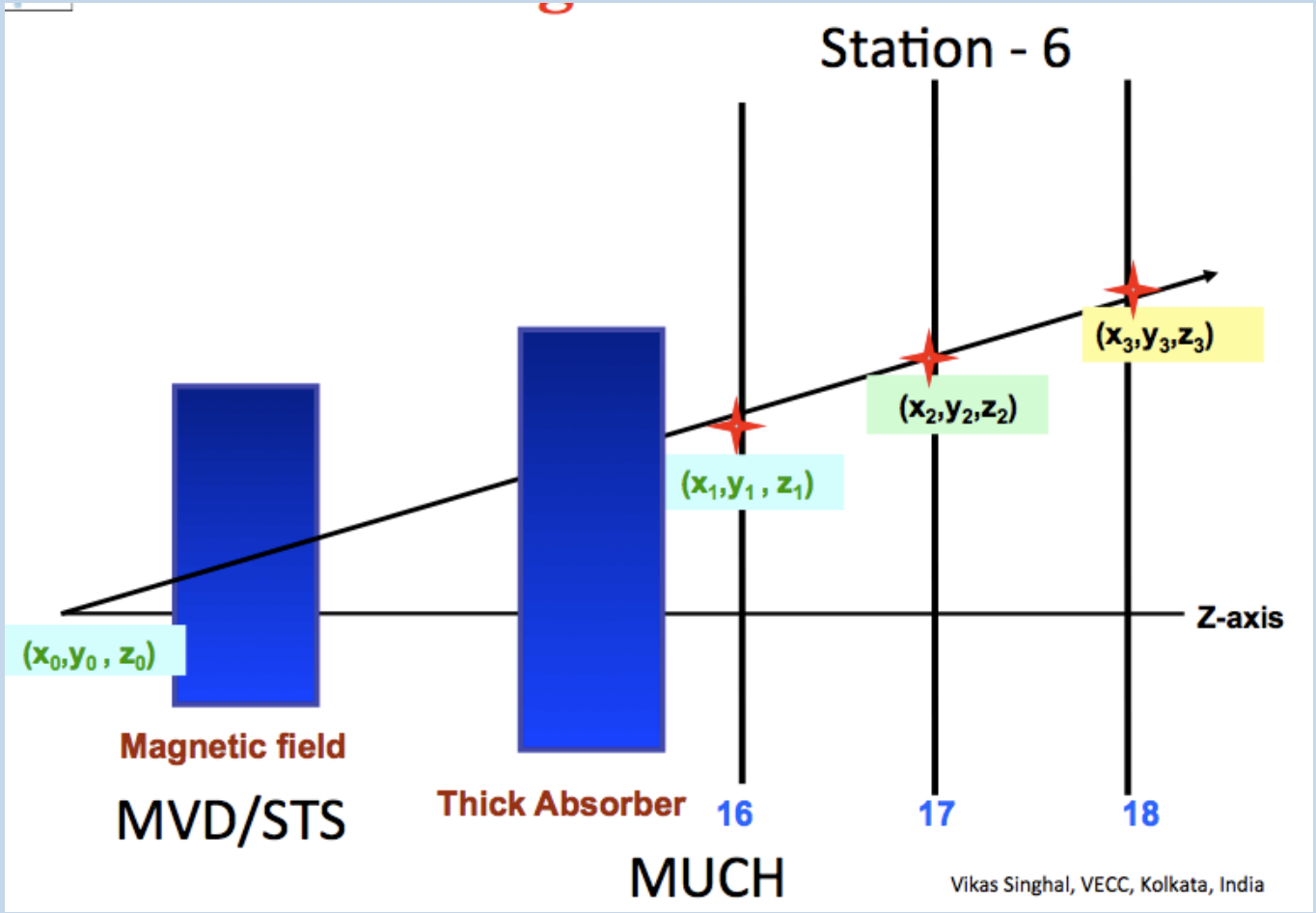
GPU only (2 x Radeon)



Performance on both servers quite similar (about 150 tracks / mus)

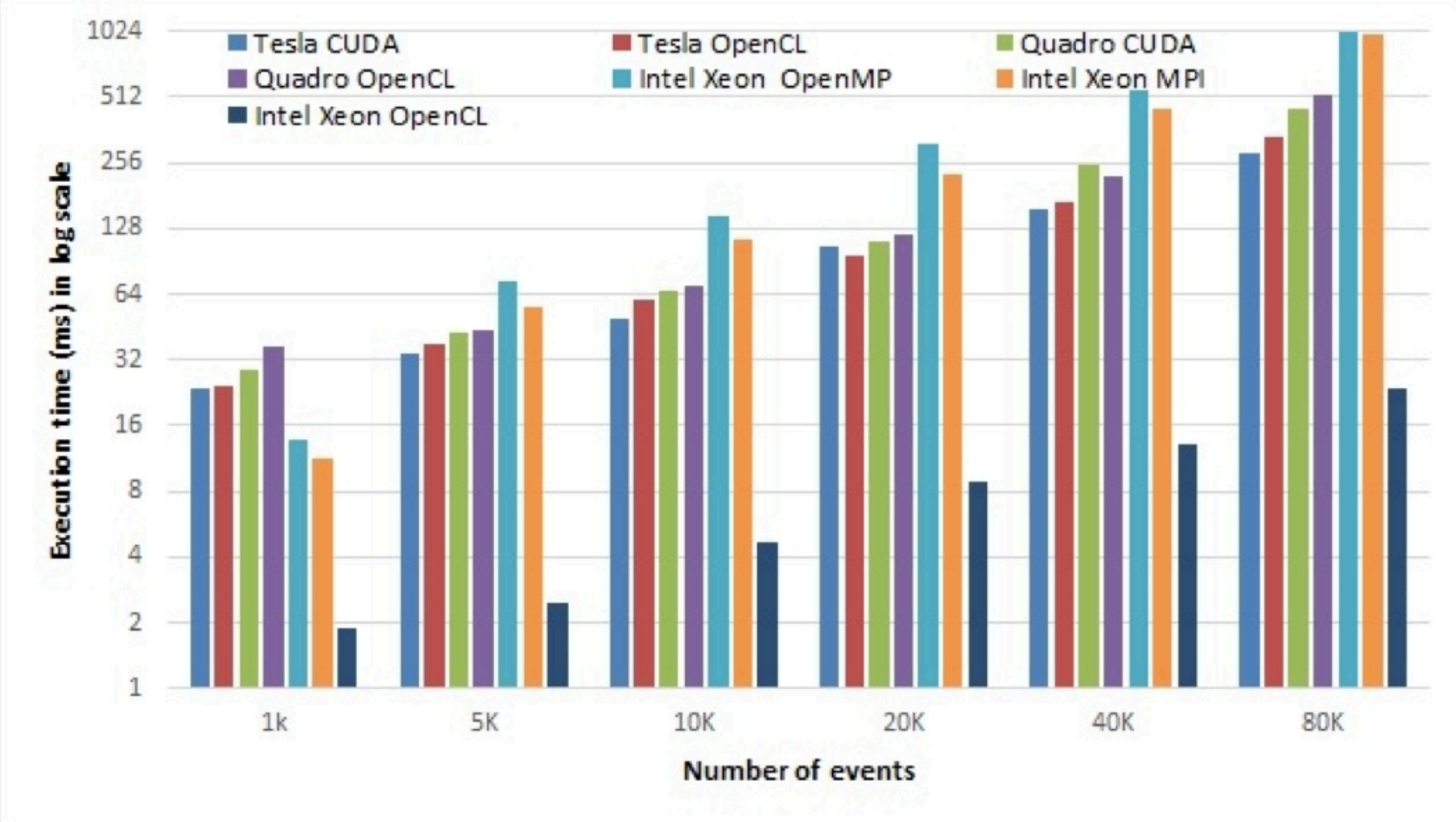
Only track fitter (KF)!
Plan: Port CA track finder also

Muon trigger studies



Relatively simple algorithm: fit three measurements after absorber with straight line, extrapolate to target.

Muon trigger studies

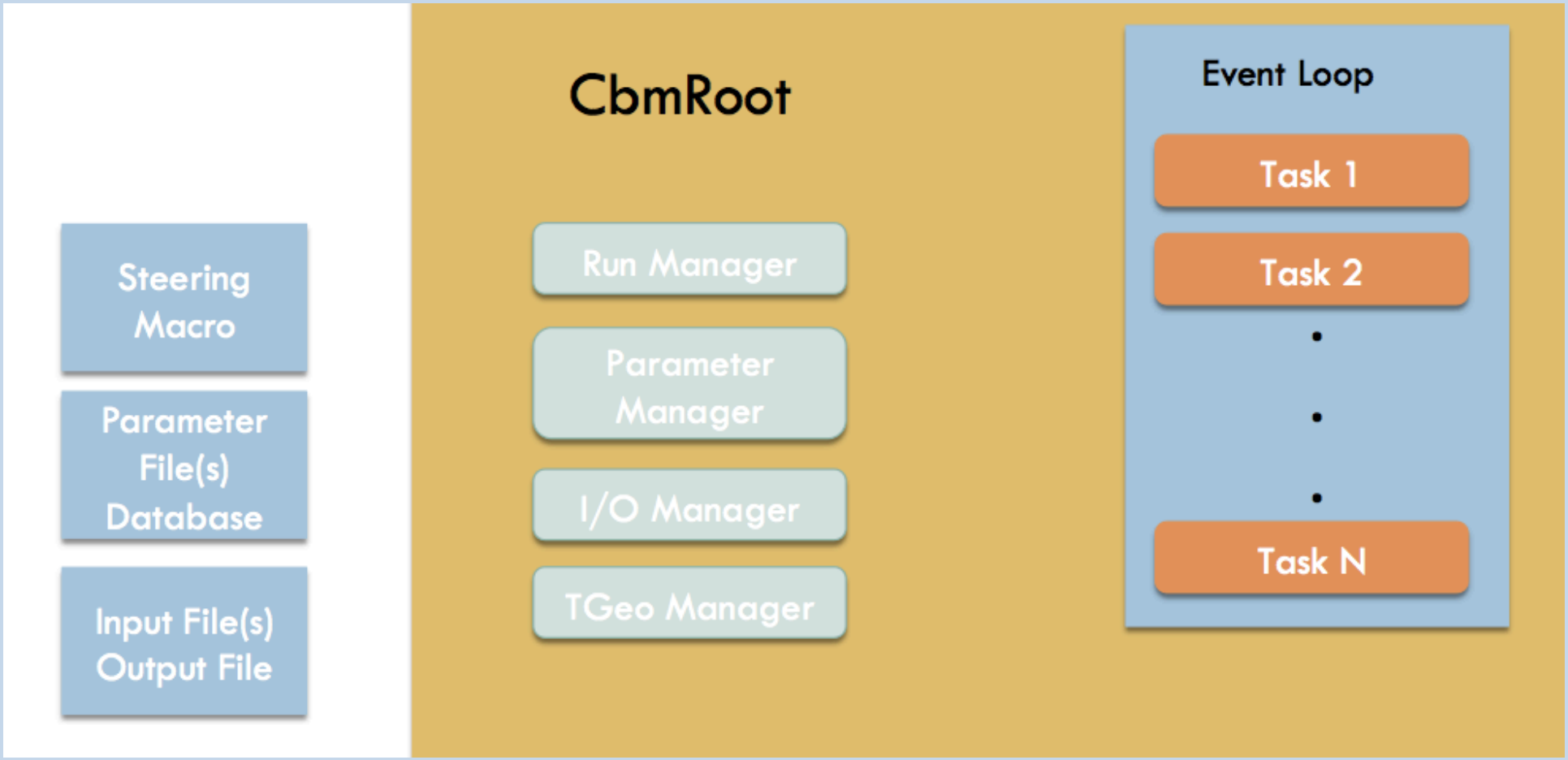


Investigation on several CPU / GPU architectures. Strong differences between different computing paradigms on same architecture.

Problems

- Only part of full reconstruction yet covered.
  - no data pre-processing before tracking
  - upstream detectors not connected
- Simple, almost parameter-less environment
  - how to configure / control?
- Still event-based reconstruction. Need to switch to time-based.
- Integration into software framework unclear.

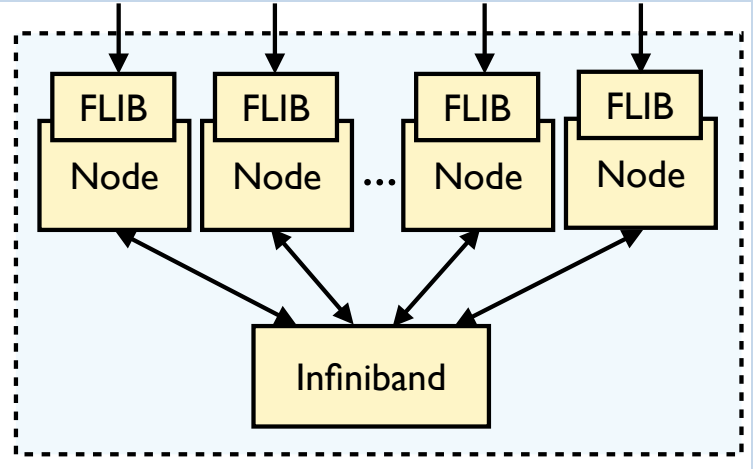
Current framework data flow



Fully serial execution

Where to parallelize?

- FLES is designed as HPC cluster
  - commodity hardware
  - GPGPU accelerators
- Chunks of data („time slice“) distributed to independently operating computing nodes.
- Obvious data parallelism on event / time-slice level
- But: each computing node will have a large number of cores
- Need in addition parallelism within event / time slice



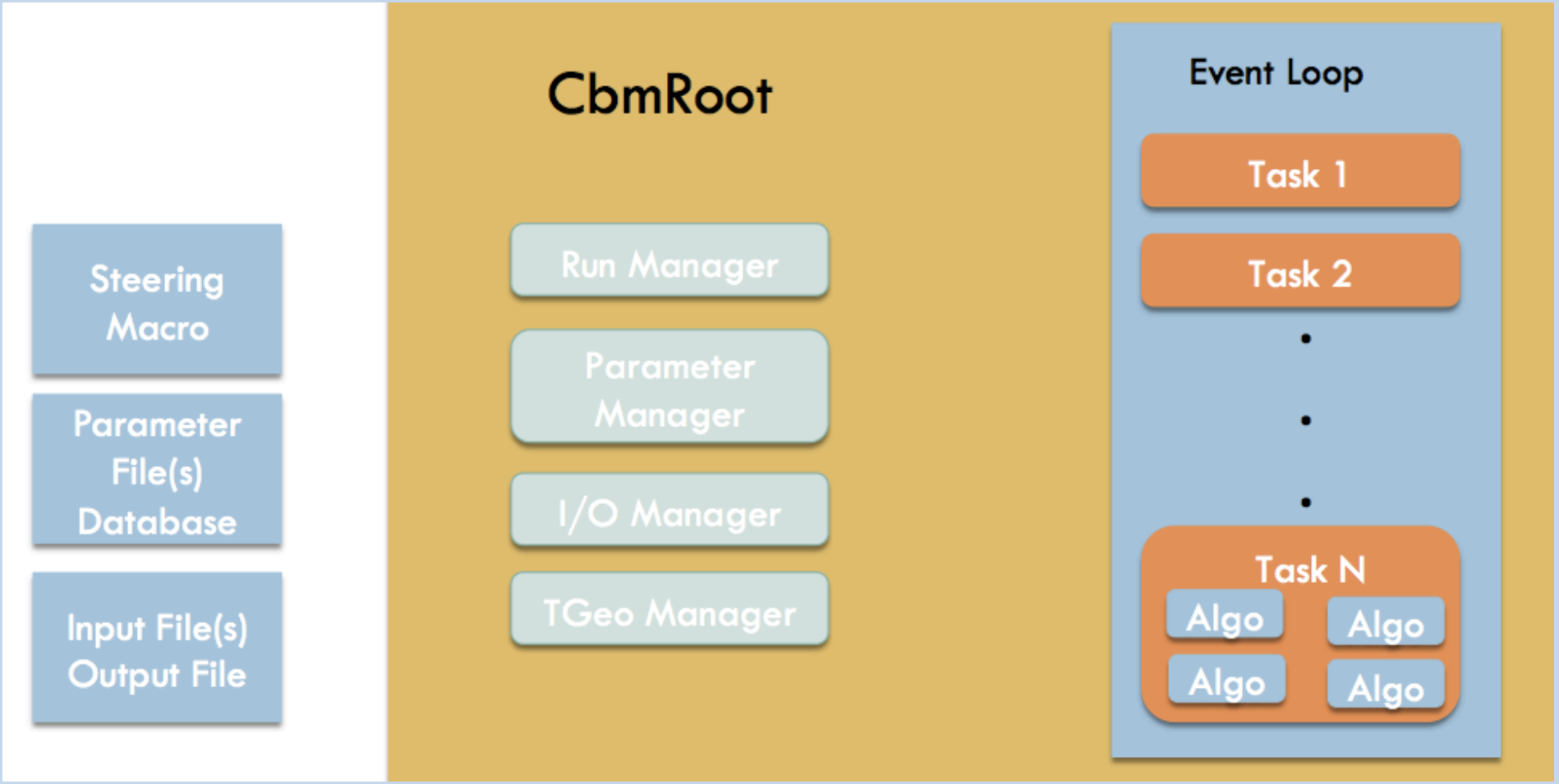
Parallel Problems

- The way to accelerate reconstruction code is surely parallelisation. Any sequential code exploits only a small fraction of the available computing power on nowadays commodity hardware.
- But: parallel programming requires a high level of specialised skill. There is commonly no usable abstraction layer / language that enables the common programmer (experienced in C++) to efficiently program parallelly.
- Parallel code is nowadays hardware dependent – e.g. Intel TBB, NVIDIA CUDA. But a choice of hardware for a computing farm to be built in five years is not wise. There is no manpower to develop code on different archtitecturs.
- Parallel code is by factors larger than sequential one, ugly and hard to read, thus hard to maintain.
- Our platform ROOT is not (yet) trivially parallisable.

Parallelisation in the framework

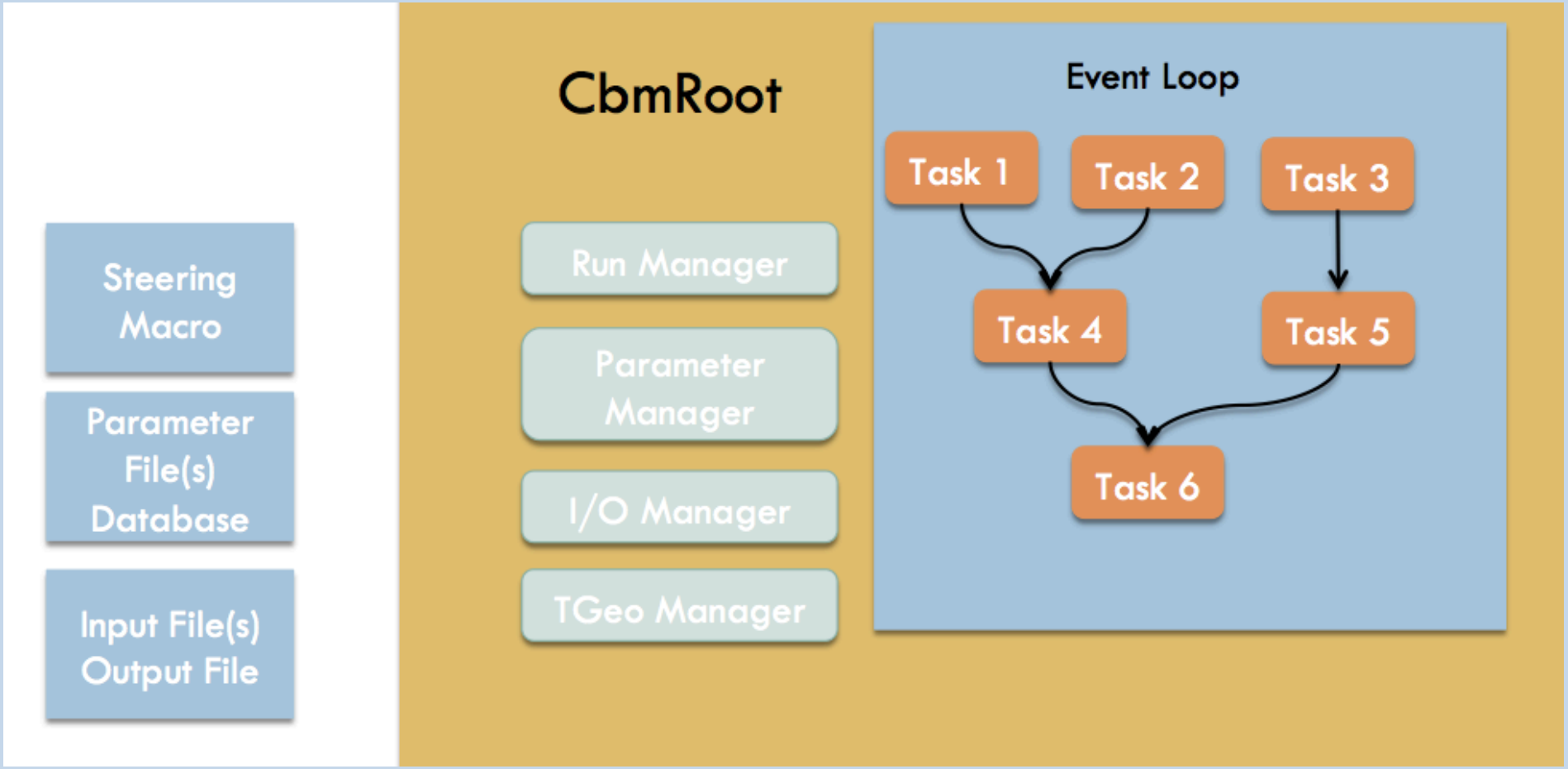
- No support for concurrency from C/C++
  - nor from ROOT
- Workload is highly data-parallel
  - average event size is only 100 kB
- Start as many CbmRoot processes as cores are available?
  - memory expensive (context)
  - can this work online for the FLES?

Parallelisation within tasks



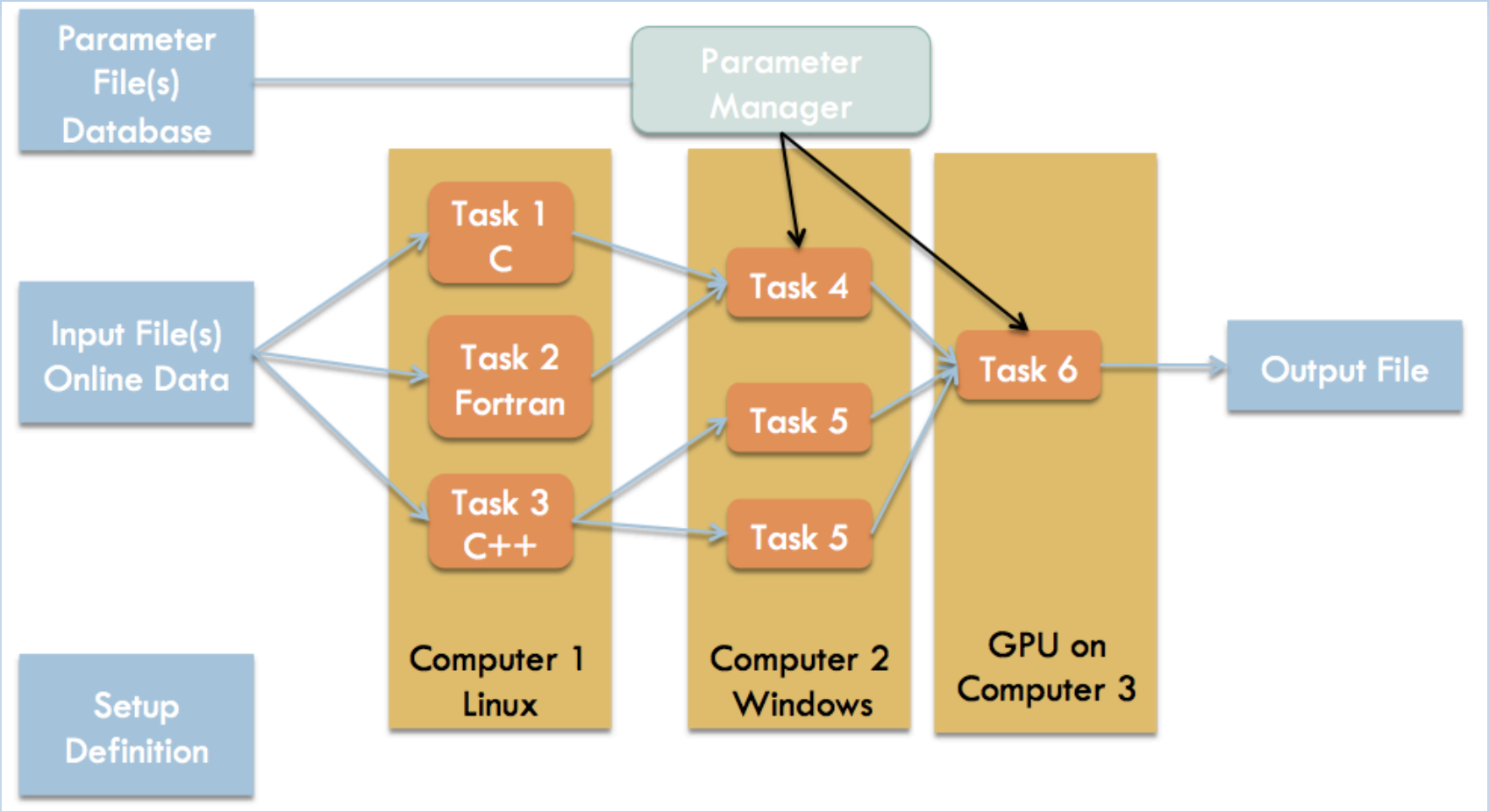
- Call parallel kernels from within a task in user-defined code
- No need to change the framework
- Still monolithic: event has to finish before next one can be started

Parallelisation on task level



- Start tasks in multiple threads
- Major changes to framework required
- Scaling? Concurrent data access? Locks / Race conditions?

Multi-Processing



- Each task is an independent process: no context switch, no locks / syncs
- No event loop
- Use 0MC for asynchronous I/O (no copy if inproc / on the same node)