Report from

S. Longo – 4° SuperB Collaboration Meeting – La Biodola

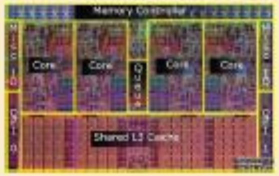
Event Processing Track Summary

Adam Lyon (Fermilab)
Axel Naumann (CERN)
Rolf Seuster (CERN)

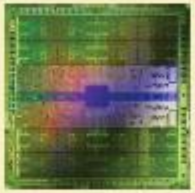
CHEP 2012 @ NYC

https://indico.cern.ch/contributionDisplay.py?sessionId=0&contribId=598&confId=149557

Outline

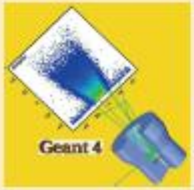


Processing on multicores



Processing on GPUs

Common frameworks



Simulations

Reconstruction algorithms

Everything else

Framework Talks

Study of a Fine Grained Threaded Framework Design

Christopher Jones

FNAL

On behalf of the CMS Offline Organization

https://indico.cern.ch/contributionDisplay.py?sessionId=3&contribId=194&confId=149557

Framework Pieces

Events can be processed in parallel

An Event is filtered by Paths

Paths run in parallel

Paths hold a list of Filters

Filter runs only if previous Filter passes

EndPaths hold OutputModules

EndPaths run in parallel after Paths finish

Producers make data

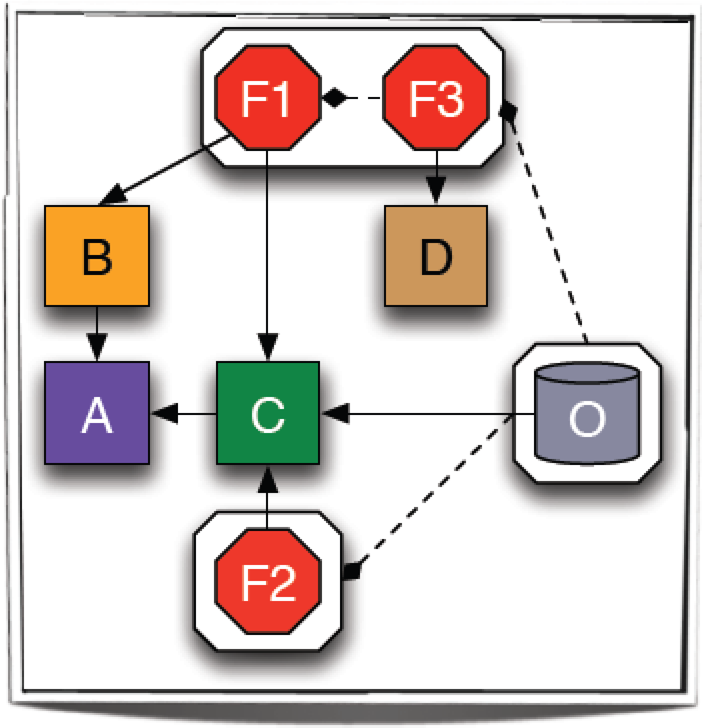
Run first time their data is requested

Producers run in parallel

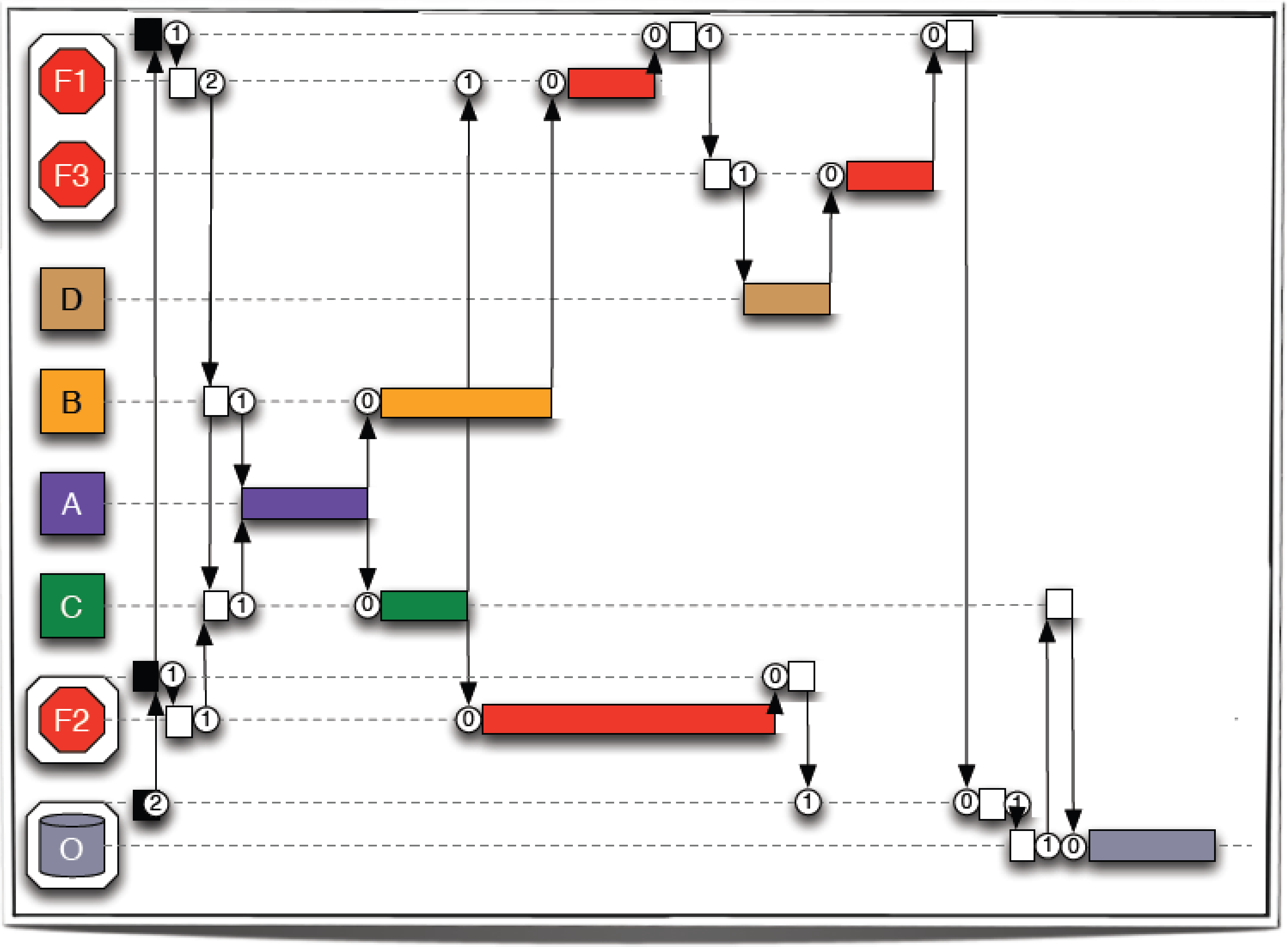
Filters, Producers & OutputModules

All are referred to as Modules

Run only after their input data is available



Parallelization



libdispatch

Developed by Apple Inc

- Port is available for Linux and Windows

Task Queue based system

- Task is a C/C++ function plus context
  - Context can be any data you want
- Tasks are placed in a light weight queue
  - Can easily support millions of queues in one process
- Tasks are pulled from queues and then run
- System guarantees that cores are not oversubscribed

Implementation

Events

Run N Event instances simultaneously using global concurrent queue

- N is configurable

Paths

Path starts a task for the first Filter on the Path

When Filter finishes it launches a task to run the next Filter on the Path

Modules

Modules have a list of data they will request from Event

- Used to do parallel prefetching

Modules are shared between all Event instances

- Keeps memory overhead as low as possible

ModuleWrappers

One per Module per Event

Has serial queue used to guarantee module is run only once per event

Has a task group used to notify when data prefetches have completed

ProducerWrappers

One per Producer per Event

Remembers if Producer has already run for that Event

Has a task group used to notify others when Producer has made its data

Measurement Strategy

Approximate reconstruction behavior

489 Producers

2 OutputModules

278 Producers have their data requested directly from OutputModule

Module Dependencies

What data each module uses

Such information is recorded by CMS framework already

Module Timing

Get per event module timing for 2011 high pileup data

~30 interactions per crossing

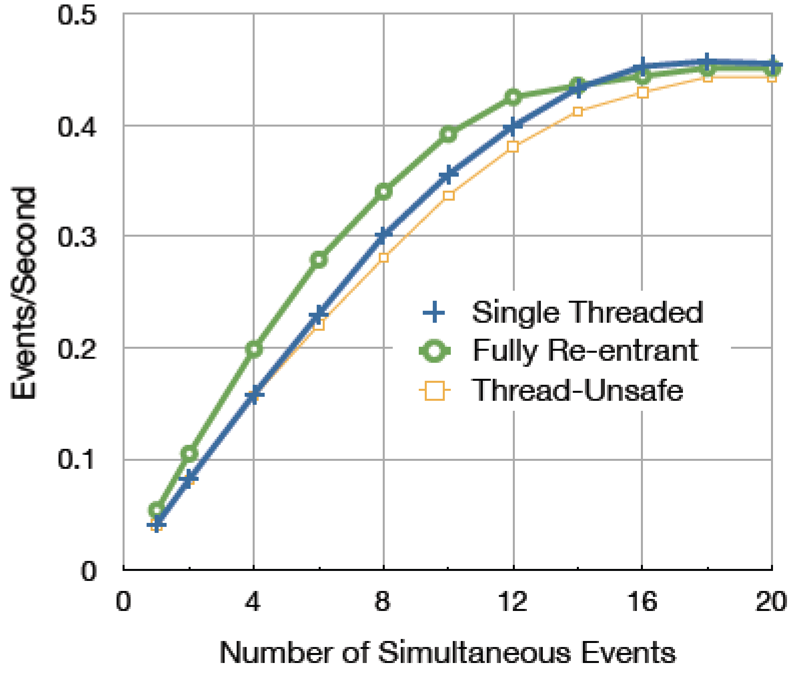
Feed dependencies and timing to demo framework

Compare timing to a simple single threaded demo framework

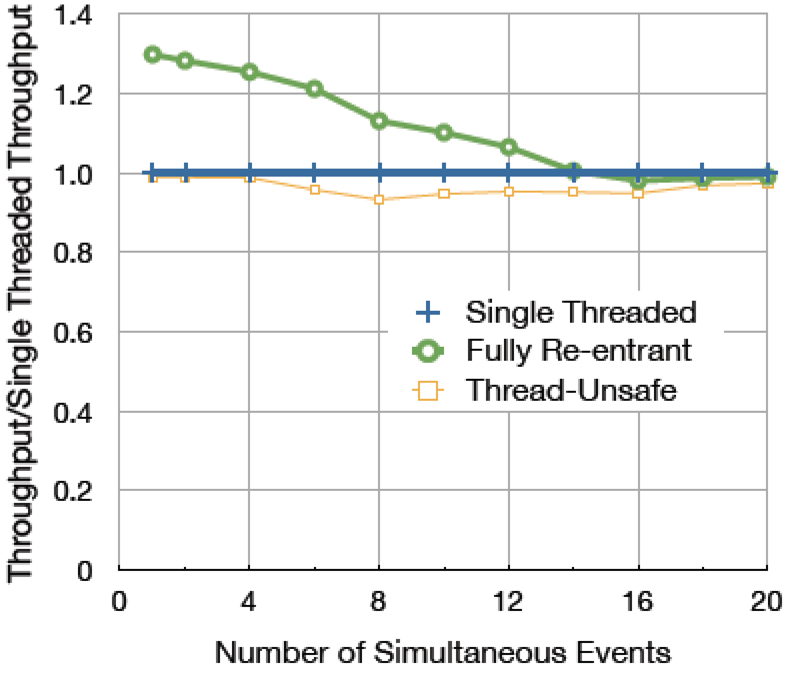
Allows estimate of overhead from libdispatch

Scaling: 16 Cores

Throughput



Relative to Single Threaded



Producers fully use a core by doing a numeric integration

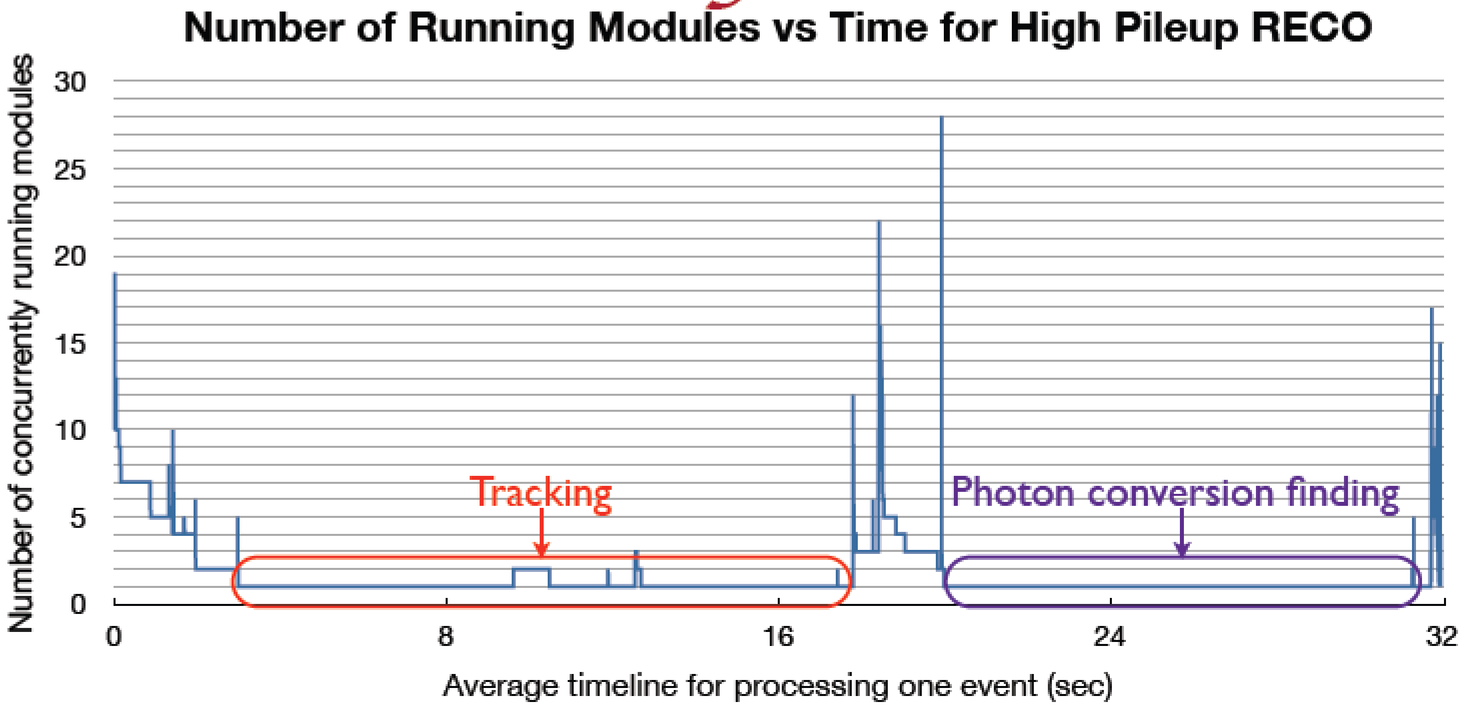
calibrated how many seconds per integration step

Thread-unsafe module case scales to 95+% of single threaded

Both are running N processes rather than N events in one process

Fully re-entrant peaks at 30% faster

Concurrency Limit



Short periods of high module level parallelism

Long periods with only 1 or 2 modules

First period is tracking

Second period is photon conversion finding

Parallelizing within those module would be beneficial

Conclusion

Task queue based systems can be used for HEP frameworks

Technology scales well

Can transition code to be thread safe one module at a time

Don't expose thread primitives to physicists

Can use task queues internal to their own modules which are simpler than locks

Concurrency limited by dependencies between modules

Parallelizing tasks within long running modules would be beneficial

Development and Evaluation of Vectorised and Multi-Core Event Reconstruction Algorithms within the CMS Software Framework Software by Thomas Hauth

Session: Engineering, Data Stores and Databases Thursday 5PM

Presently testing additional threading technologies

OpenMP

Intel's Threading Building Blocks

CMS will choose a threading technology this year

Start transitioning CMS's framework to use threads in 2013

*The **art** Framework*

Chris Green
Fermilab Scientific Software
Infrastructure Group
CHEP 2012
21 May, 2012

Fermilab

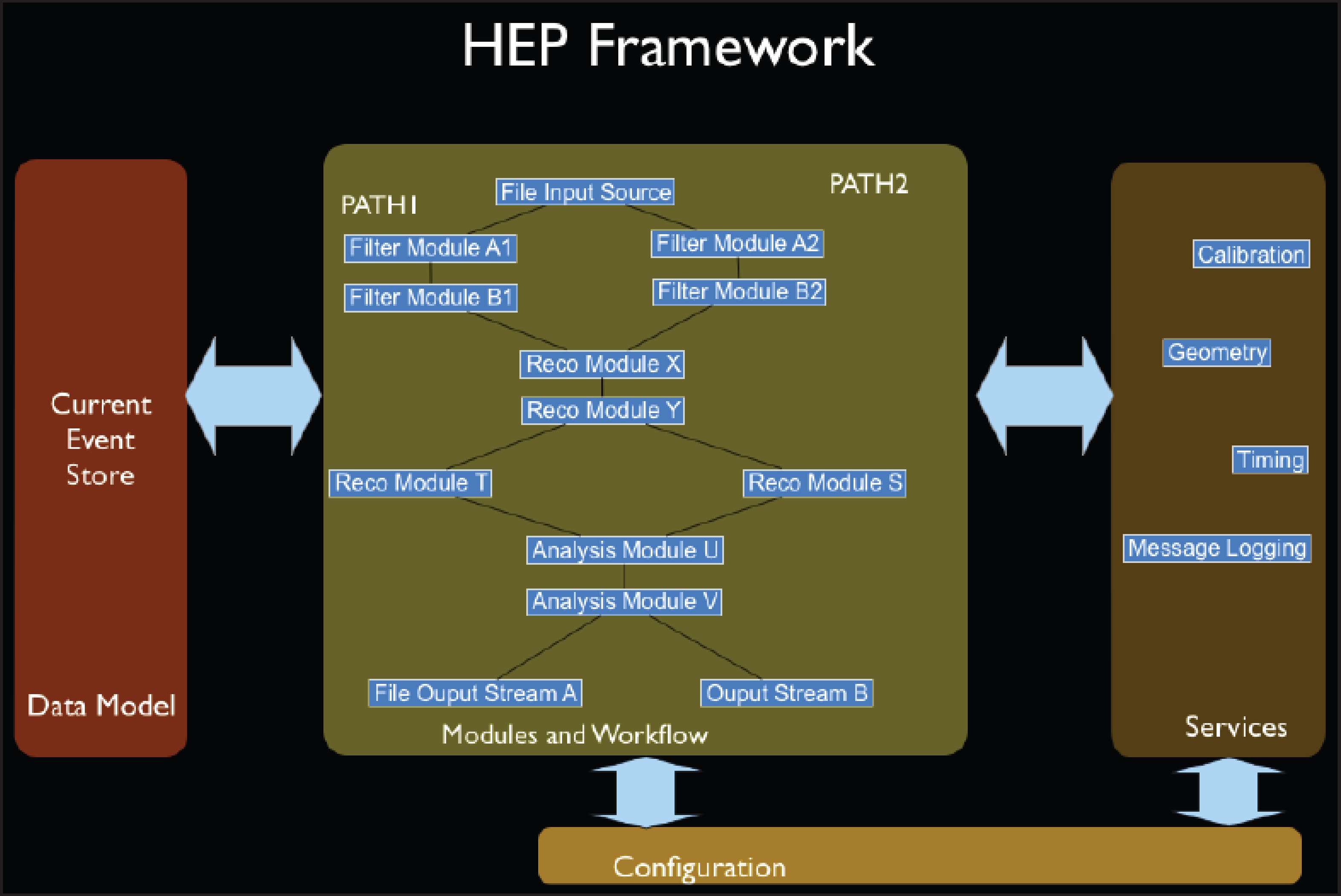
Fermi National Accelerator Laboratory
Office of Science / U.S. Department of Energy
Managed by Fermi Research Alliance, LLC

https://indico.cern.ch/contributionDisplay.py?sessionId=3&contribId=354&confId=149557

What and why is ***art****?*

- What is **art**?
  art is a generic C++-based modular analysis framework, for use from generator-level or DAQ event building through simulation, production and user analysis. **art** grew out of the CMS framework and was developed to satisfy the common requirements of intensity frontier experiments (initially **Mu2e**, **NOνA** and **LArSoft**).
- Why is **art**?
  Most HEP experiments use a framework; **art** is a framework that is being used by multiple experiments, which has relieved them of the need to produce and maintain their own.

Architecture



Architecture

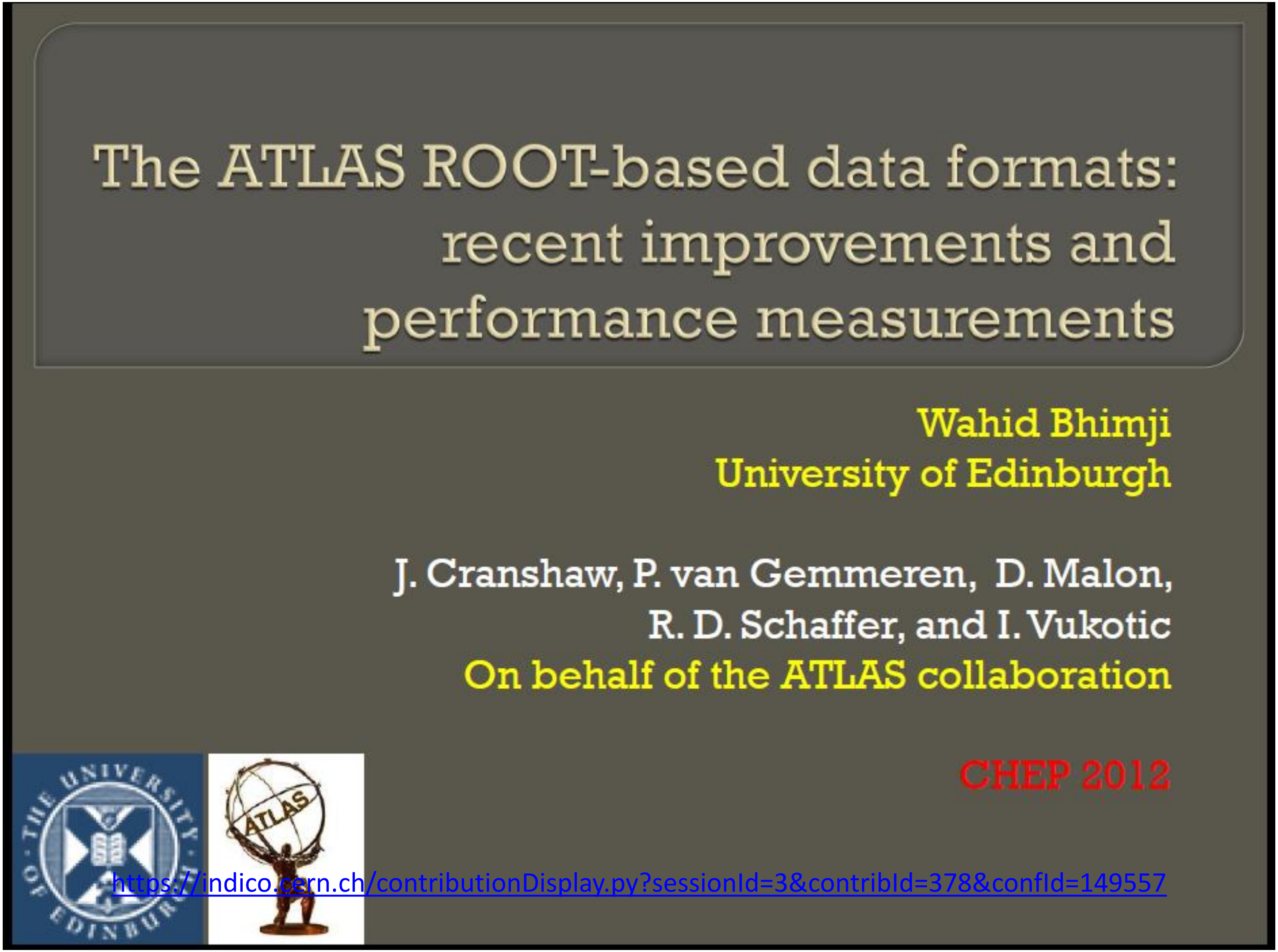
- Experiments use **art** as an external package – their build system is not tied to that used to develop **art**.
- I/O and work schedule are handled by a state machine.
- Modules are generally provided by users, and are divided into inputs (**`sources`**), **`producers`**, **`filters`**, **`analyzers`** and **`outputs`**.
- Inter-module communication is handled principally by means of persistent data structures (**`products`**) passed via entities with known lifetimes: **`event`**, **`subrun`**, **`run`**.
- **`products`** are distinguished from algorithms ⟹ modules don't need to address persistency mechanics.
- **`products`** retrieved from the data store are non-modifiable: derived or edited data are saved as a new product.
- Configurable exception handling: categorization of a failure is distinct from its handling action.

Key features

- Facility for products to refer to other products in collections already saved (`Ptr`).
- `product` mixing ("pile-up"): users need to know how to combine the data from multiple instances of a particular `product`, but not the mechanics of obtaining those data and writing out the merged `product`.
- Metadata may be stored in a relational SQLite database in memory and / or embedded in a ROOT data file.
- Simple configuration language with partitioned module configuration information.
- Bi-directional associations (`Assns`) between `products` already in the data store.
- An input source class template for more straightforward user implementation of "raw" data input.

Future enhancements

- Expand use of **SQLite** DB to all existing metadata.
- Unify the concepts of **event**, **subrun** and **run**.
- Revamp processing intervals.
- Remove internal use of Reflex to be ready for **ROOT/ Cling**.
- Move to **ISO C++ 2011** (already used in development, **artdaq**).
  - Allow user-defined metadata in **SQLite** DB.
  - Event display toolkit (graphical toolkit agnostic): better-defined / -suited interface to framework for operators, algorithm developers.
  - Generalize and expand **CMake**-based build / package delivery system for use by experiments as an alternative to supporting their own build system.
- "Multi-schedule **art**": process multiple events simultaneously in the same executable; in addition, allowing for algorithm parallelization within modules.
- Currently prototyping DAQ event-building and triggering using **art** (**artdaq**) in conjunction with **MPI**[3] for **DS50**, **Mu2e**, **μBooNE**, **NOνA** experiments.
- Multi-thread and multi-process parallel I/O.



The ATLAS ROOT-based data formats: recent improvements and performance measurements

Wahid Bhimji
University of Edinburgh

J. Cranshaw, P. van Gemmeren, D. Malon,
R. D. Schaffer, and I. Vukotic
On behalf of the ATLAS collaboration

CHEP 2012

https://indico.cern.ch/contributionDisplay.py?sessionId=3&contribId=378&confId=149557

ROOT I/O features we use

See e.g. P. Canal's talk at CHEP10

Writing files:

- Split level: object data members placed in separate branches
  - Initial 2011 running: AOD /ESD fully-split (99) into primitive data
- From 2011 use ROOT "autoflush" and "optimize baskets"
  - Baskets (buffers) resized so they have similar number of entries
  - Flushed to disk automatically once a certain amount of data is buffered to create a cluster that can be read back in a single read
  - Initial 2011 running we used the default 30 MB
- Also use member-wise streaming
  - Data members of object stored together on disk

Reading files:

- There is a memory buffer TTreeCache (TTC) which learns used branches and then pre-fetches them
- Used by default for AOD->D3PD in Athena
- For user code its up to them

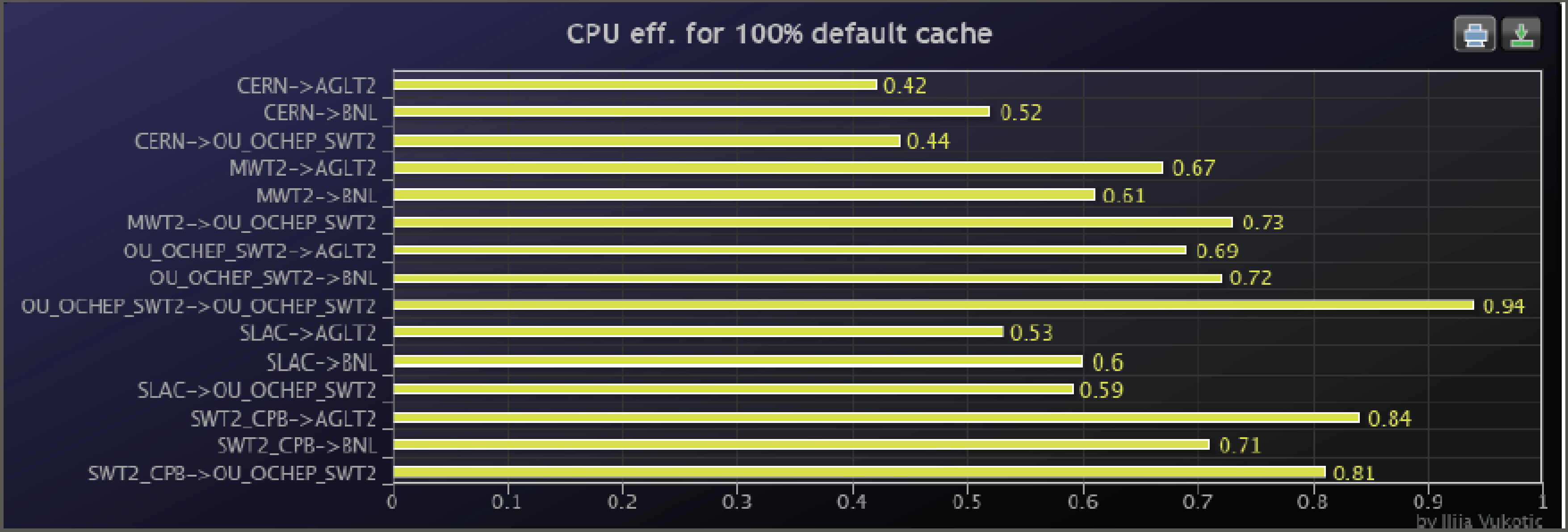
Performance Results

Repeated local disk read; Controlled environment; Cache cleaned

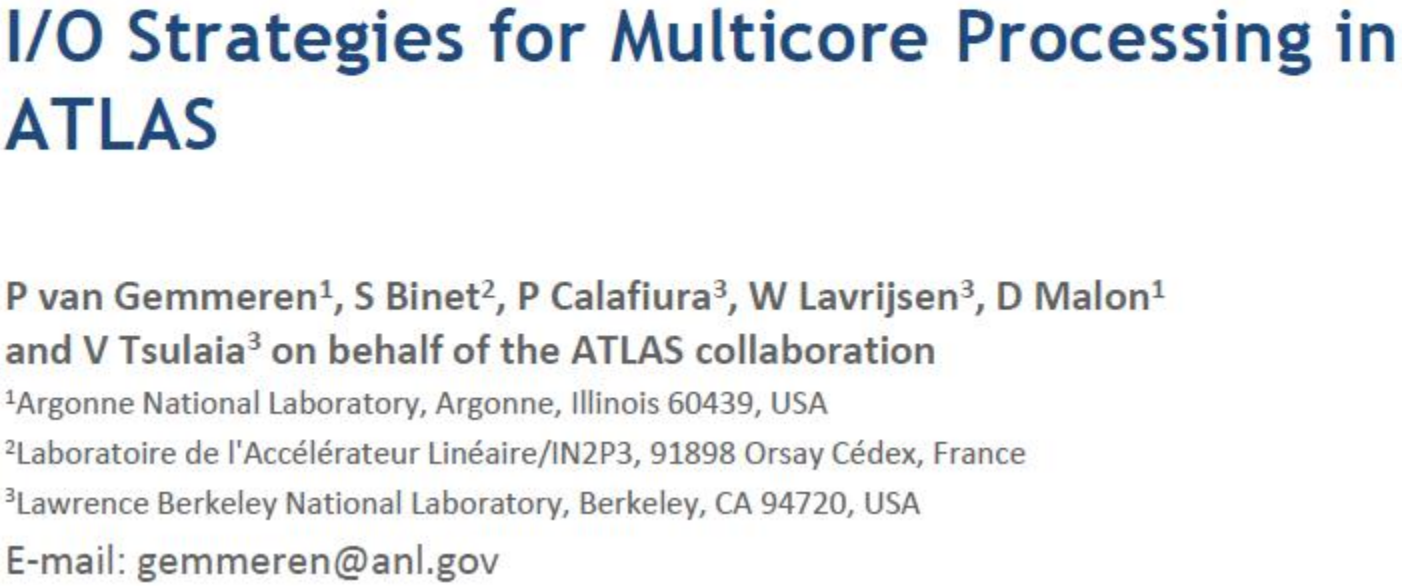
AOD Layout	Reading all events	Selective 1% read
OLD: Fully split, 30 MB Auto-flush	55 (±3) ms/ev.	270 ms /ev.
CURRENT: No split, 10 event Auto-flush	35 (±2) ms /ev.	60 ms/ev.

- Reading all events is ~30% faster
- Selective reading (1%) using TAGs: 4-5 times faster

CPU Efficiencies over WAN



- First measurements …. not bad rates
  - 94% local read eff. drops to 60%-80% for other US sites
  - and around 45% for reading from CERN
- Offers promise for this kind of running if needed
  - Plan to use such measurements for scheduling decisions



I/O Strategies for Multicore Processing in ATLAS

P van Gemmeren[1], S Binet[2], P Calafiura[3], W Lavrijsen[3], D Malon[1] and V Tsulaia[3] on behalf of the ATLAS collaboration

[1]Argonne National Laboratory, Argonne, Illinois 60439, USA

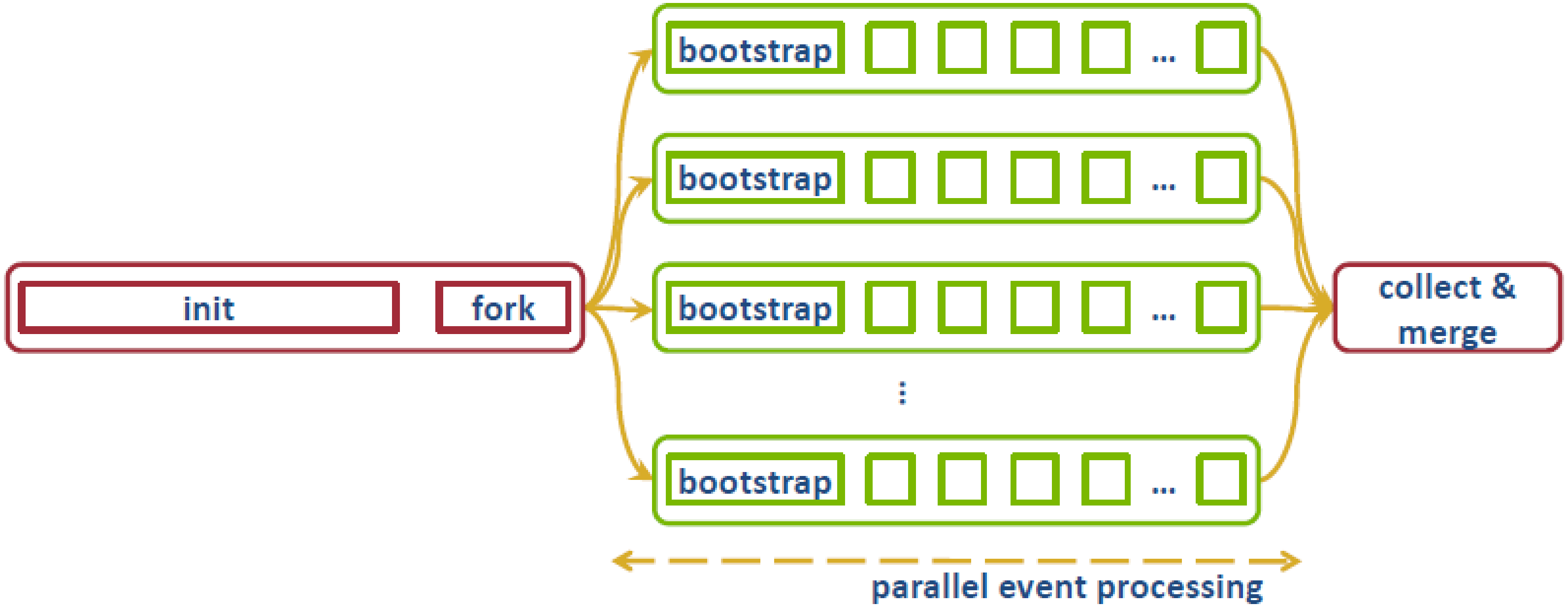
[2]Laboratoire de l'Accélérateur Linéaire/IN2P3, 91898 Orsay Cédex, France

[3]Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

E-mail: gemmeren@anl.gov

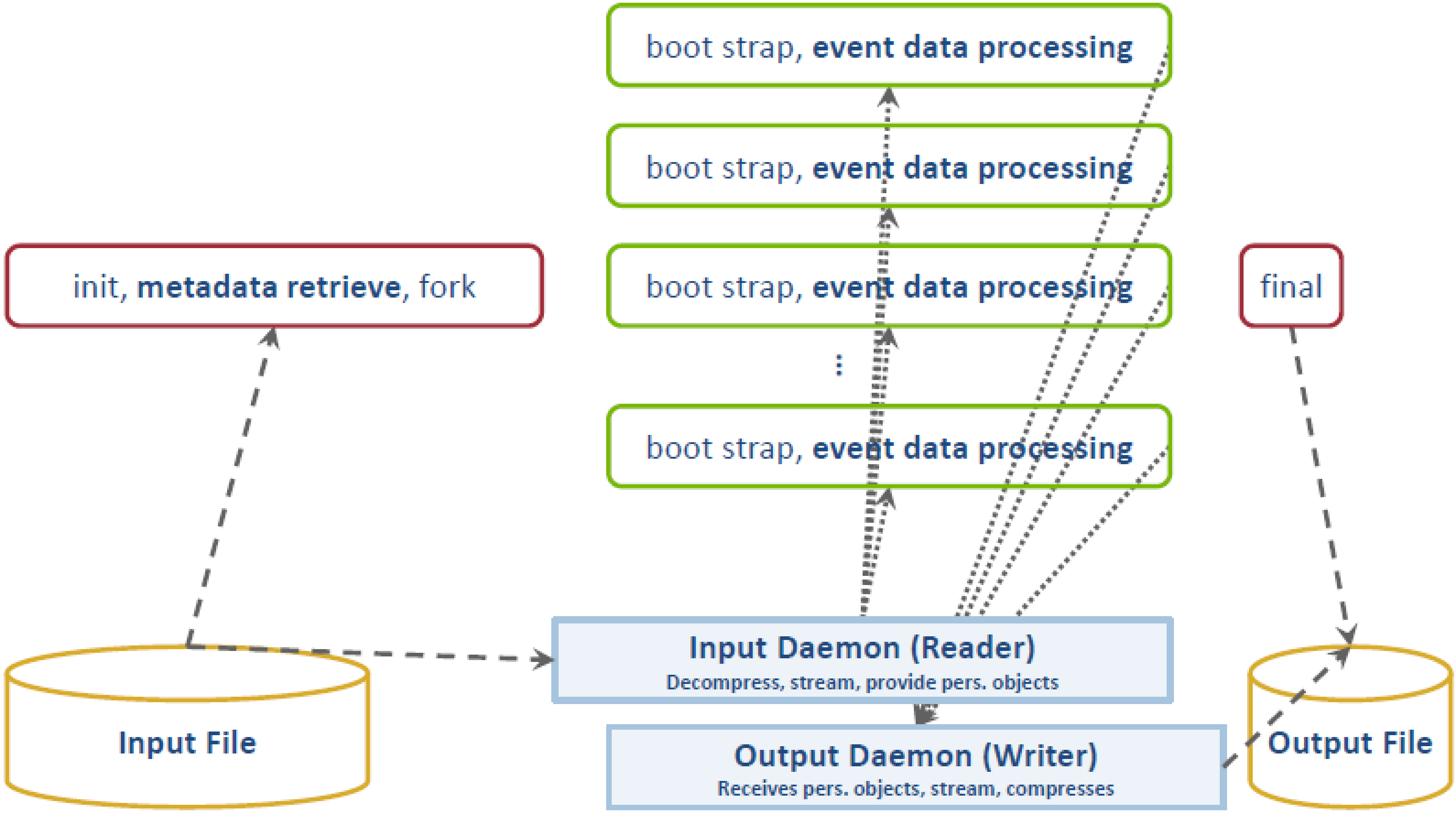
https://indico.cern.ch/contributionDisplay.py?sessionId=3&contribId=377&confId=149557

Current AthenaMP I/O infrastructure



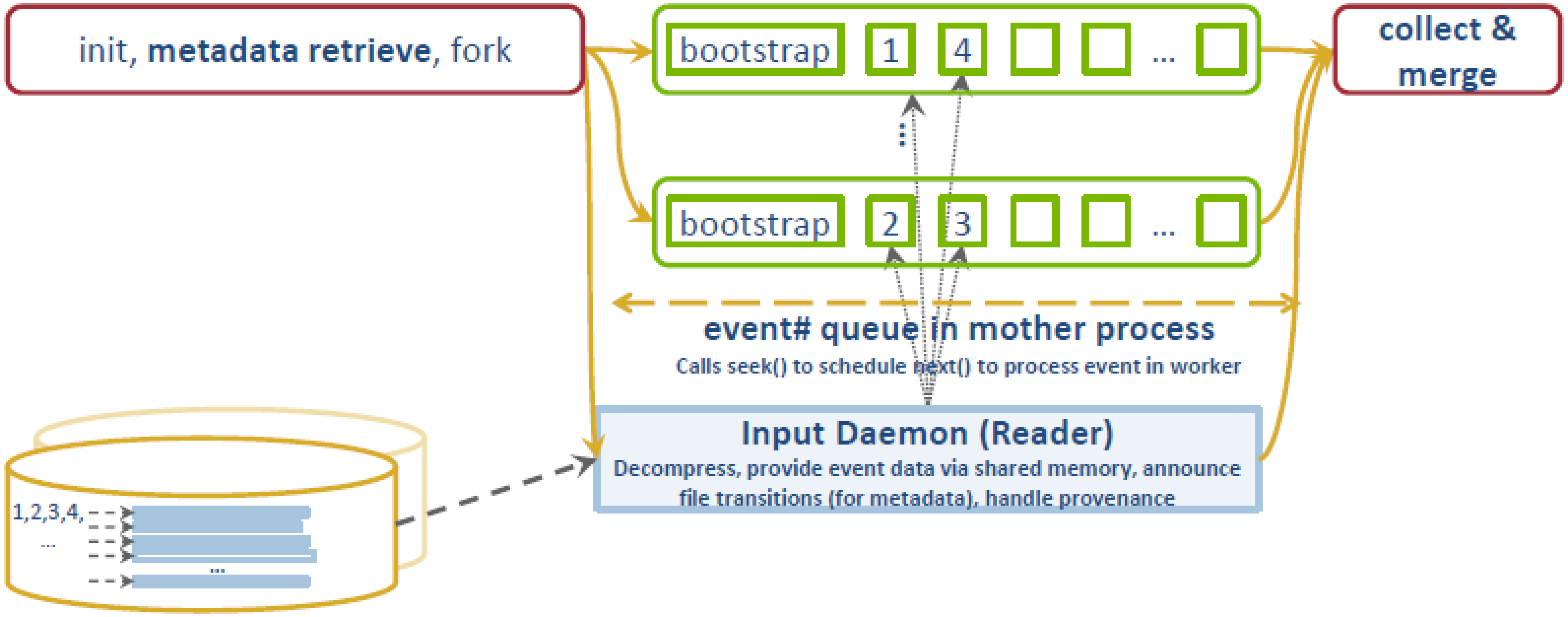
- Multiple workers handle **I/O independently** using **distinct instances of a serial I/O framework**
  - Each worker process produces its own output file, which need to be merged after all workers are done.

Scatter / Gather architecture for multicore I/O


boot strap, **event data processing**
boot strap, **event data processing**
init, **metadata retrieve**, fork
boot strap, **event data processing**
final
⋮
boot strap, **event data processing**
Input Daemon (Reader)
Decompress, stream, provide pers. objects
Input File
Output Daemon (Writer)
Receives pers. objects, stream, compresses
Output File

Shared reader strategies: ByteStream data

- For Raw data, providing the next event is rather straightforward as all event data are contiguously stored in a single block which is read entirely.
  - At the same time, benefit of a single reader for ByteStream is small as Raw events can be decompressed individually.



init, metadata retrieve, fork
bootstrap
1
4
...
bootstrap
2
3
...
collect & merge
event# queue in mother process
Calls seek() to schedule next() to process event in worker
Input Daemon (Reader)
Decompress, provide event data via shared memory, announce file transitions (for metadata), handle provenance
1,2,3,4,
...
...

Summary and Outlook

- A multi-process **control framework** (like AthenaMP)) to enable HEP event processing on multicore computing architectures is an important step, but it is only one of many steps that need to be accomplished.
- Optimizing **event data storage**, so that it can be efficiently retrieved in the granularity needed by the multiple worker processes is key to avoiding performance penalties during reading.
  - The 2011 change to member-wise streaming with a small number of entries per basket will help ATLAS to tackle inefficiencies in multi-process reading of ROOT data.
  - The data layout also must ensure that data produced by multiple processes can be:
    - efficiently combined, as merging is typically done serially,
    - the resulting output is as efficient to read and store
- ATLAS is in the process of developing an **I/O architecture and components** that can efficiently support even higher numbers of parallel worker processes.
- First prototypes are being tested, but much work remains.

Other really interesting Talks

Track finding and fitting with GPUs
First steps toward a software trigger

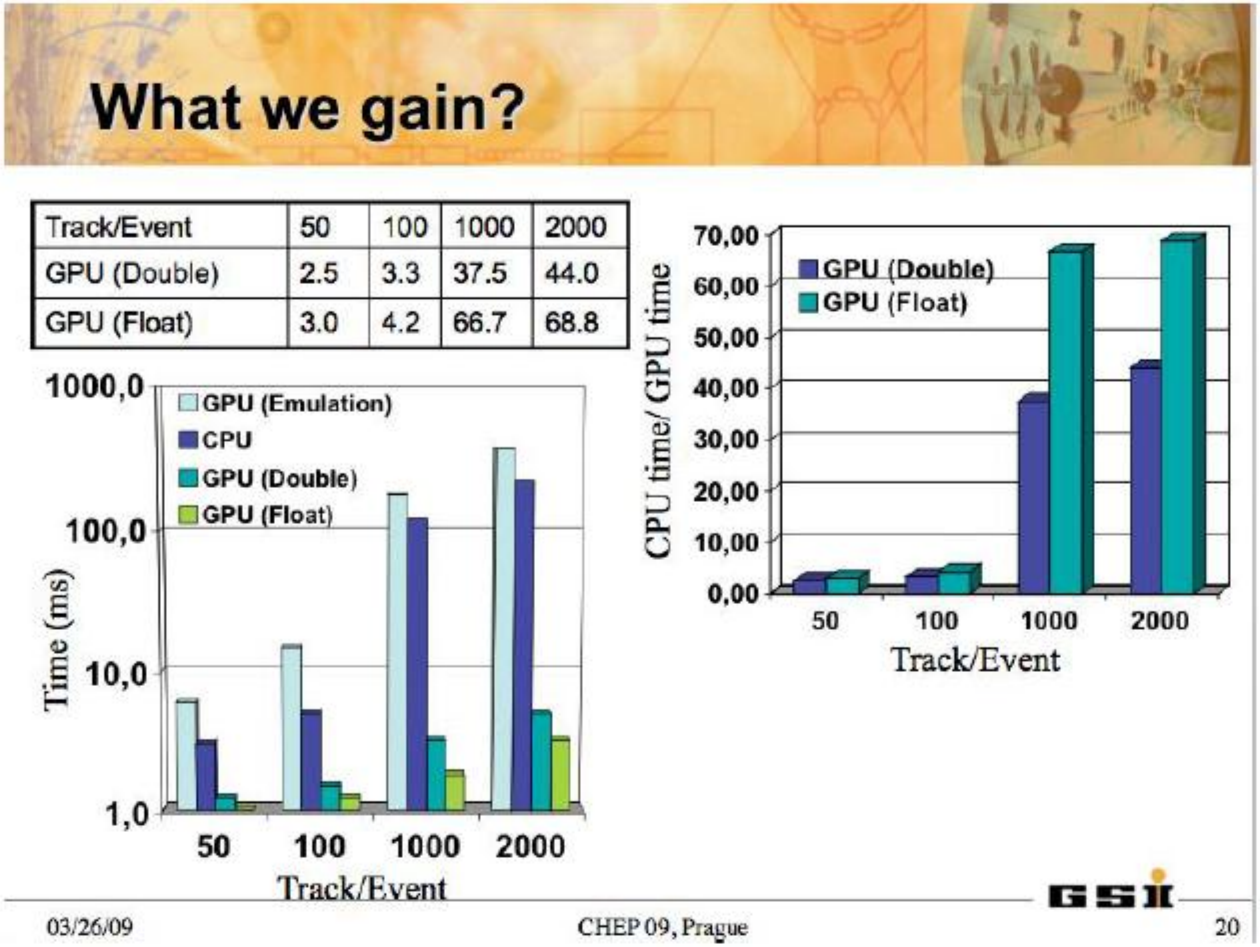
Mohammad Al-Turany
(GSI-Scientific Computing)

https://indico.cern.ch/contributionDisplay.py?sessionId=3&contribId=353&confId=149557

10^7 Events/s full reconstruction in real time!

- 1 Event/s full reconstruction = 1 -10 ms → 10 000 – 100 000 CPU cores are needed for the online trigger
- The Challenge:
  - On which data level we have to parallelize
    - Event
    - Tracks
    - Clusters
  - Which hardware to use:
    - CPUs
    - GPUs
    - FPGA
    - …

We started in 2008 to explore the GPUs for track fitting



What we gain?

Track/Event	50	100	1000	2000
GPU (Double)	2.5	3.3	37.5	44.0
GPU (Float)	3.0	4.2	66.7	68.8

03/26/09 CHEP 09, Prague 20

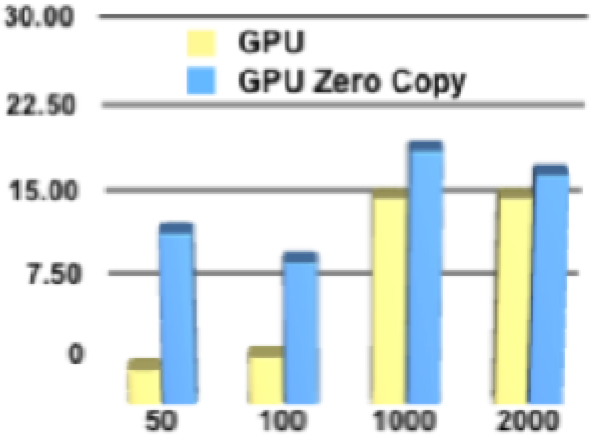
https://indico.cern.ch/contributionDisplay.py?sessionId=60&contribId=109&confId=35523

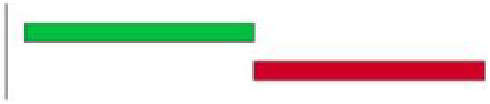
Just following the development in CUDA we improved on the same hardware

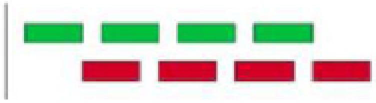
The same program code, same hardware, just using Pinned Memory instead of Mem-Copy!

CPU time/GPU time

Track/Event	50	100	1000	2000
GPU	3.0	4.2	18	18
GPU (Zero Copy)	15	13	22	20

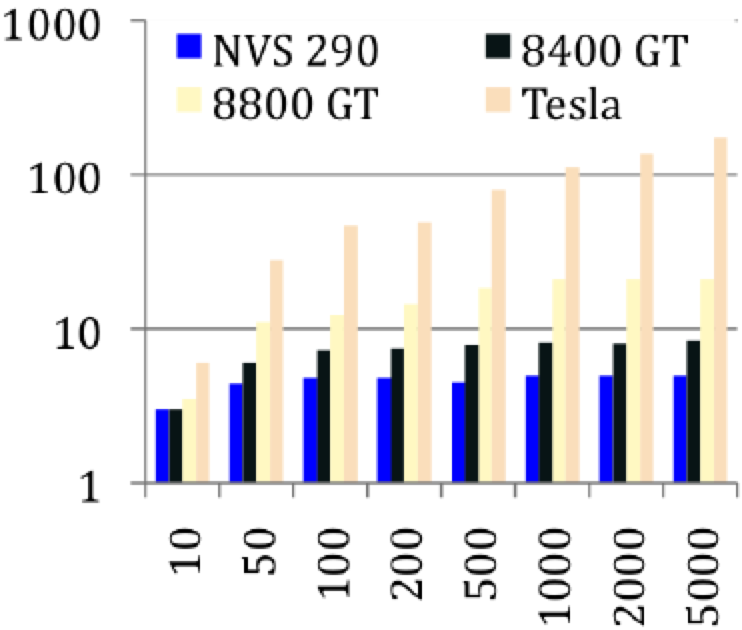



Copy data
Execute


Copy data
Execute

We start using more GPU specific features: Texture memory for field maps

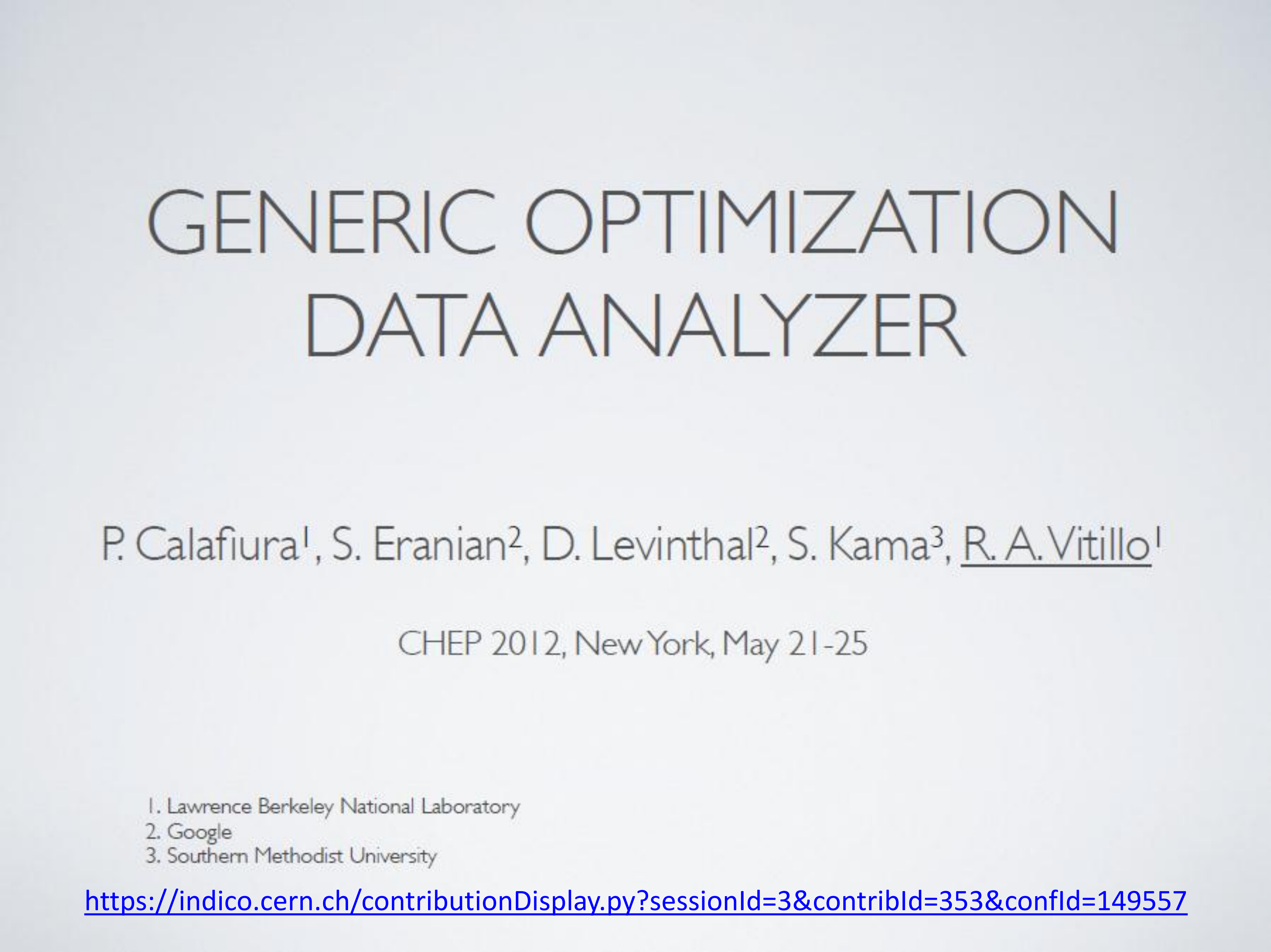
Track propagation (RK4) using PANDA Field



Speedup : up to factor 175

Trk/ Event	NVS 290	8400 GT	8800 GT	Tesla
10	3	3	3.5	6
50	4.4	6	11	28
100	4.8	7.3	12.3	47
200	4.8	7.5	14.5	49
500	4.5	7.9	18.5	80
1000	5	8.1	21	111
2000	5	8	21	137
5000	5	8.4	21	175

ACAT 2010: Applying CUDA Computing Model To Event Reconstruction Software
https://indico.cern.ch/contributionDisplay.py?contribId=147&confId=59397



GENERIC OPTIMIZATION DATA ANALYZER

P. Calafiura[1], S. Eranian[2], D. Levinthal[2], S. Kama[3], R. A. Vitillo[1]

CHEP 2012, New York, May 21-25

1. Lawrence Berkeley National Laboratory
2. Google
3. Southern Methodist University

https://indico.cern.ch/contributionDisplay.py?sessionId=3&contribId=353&confId=149557

WHAT IS GOODA?

- **Low overhead** open source Performance Monitoring Unit (PMU) event data analysis package
  - A CPU profiler
- Developed in collaboration between Google and LBNL
- Logically composed of four main components:
  - A kernel subsystem that provides an interface to the PMU
  - An event data collection tool
  - An analyzer creates call graphs, control flow graphs and spreadsheets for a variety of granularities (process, module, function, source etc.)
  - A web based GUI displays the data

CONCLUSION

- Low overhead profiler
- Implements a novel cycle accounting methodology
- Visualization of reports require only a browser
- Open Source Tool (contributions welcome!)

ACRL

Advanced Computing Research Laboratory

CHEP 2012
May 20 – 25, 2012
Lennart Johnsson

Computing Technology Future

Lennart Johnsson,
University of Houston, Houston, TX

https://indico.cern.ch/contributionDisplay.py?sessionId=0&contribId=594&confId=149557

«Backup»…
If we have more time

https://indico.cern.ch/contributionDisplay.py?sessionId=3&contribId=431&confId=149557

THE SIMULATION- AND ANALYSIS-FRAMEWORK FAIRROOT

CHEP 2012, New York

FairRoot

- Framework for simulation, reconstruction and data analysis
- Very flexible
  - No executable
    - Use plug-in mechanism from Root to load libraries only when needed
    - Use Root macros to define the experimental setup or the tasks for reconstruction/analysis
    - Use Root macros to set the configuration (Geant3, Geant4, …)
  - No fixed simulation model
    - Use different simulation models (Geant3, Geant4, …) with the same user code (VMC)

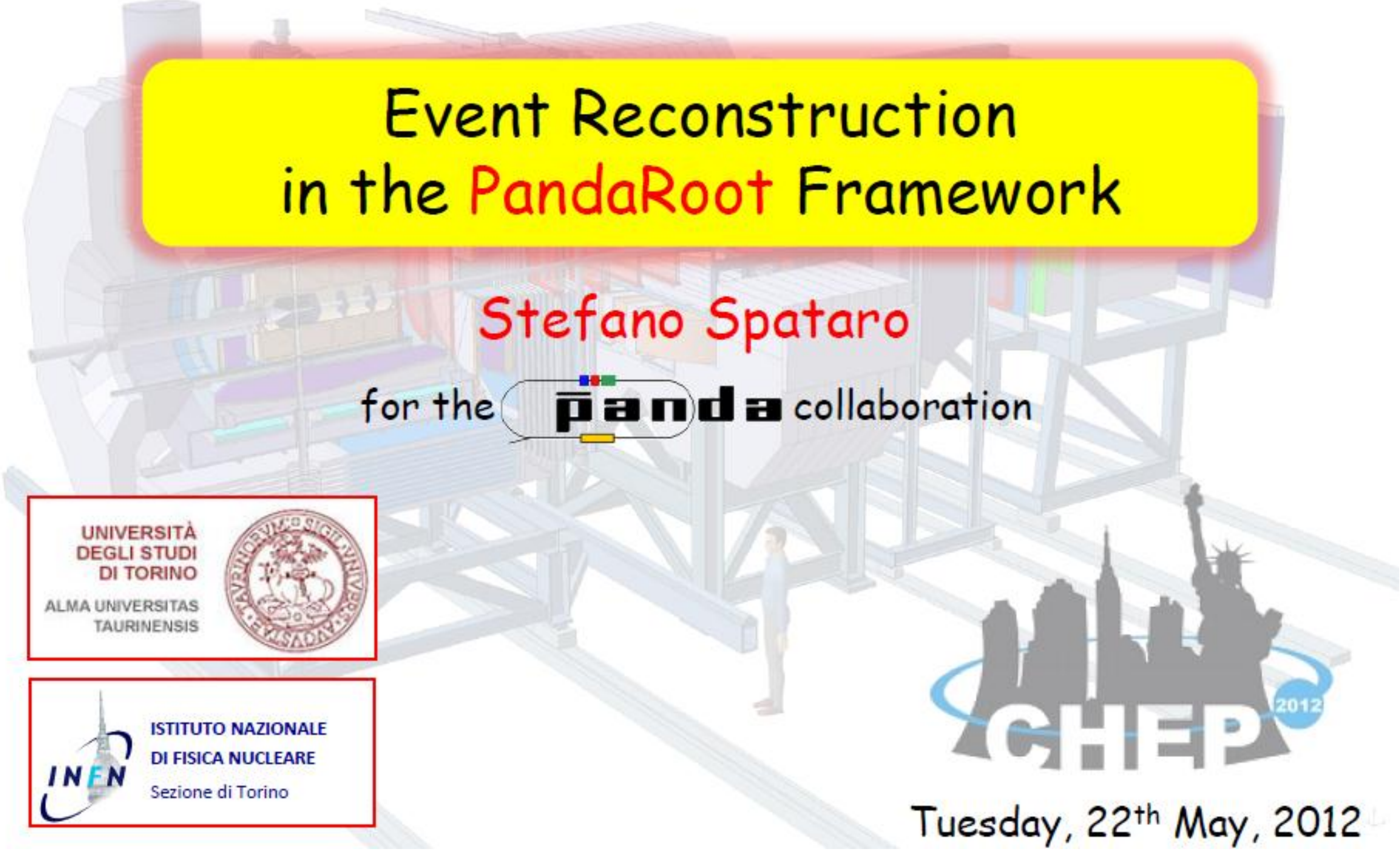
Summary and Outlook

- Hope I could show you that FairRoot
  - is flexible
  - is easy to use
  - is easy to extend
- Special tools to do dose studies
- Tools for time based simulation are implemented
  - Calculation of event time
  - Mixing of events by automatic buffering and write out when needed
  - Fast sorting of data
  - Several event builder functions

Summary and Outlook

- Many more topics only touched or not showed at all
  - Proof integration
  - Database connectivity
  - GPU usage inside of FairRoot
  - Build and test system
  - …
- Resources
  - Webpage: http://fairroot.gsi.de
  - Forum: http://forum.gsi.de
  - Test Dashboard: http://cdash.gsi.de/CDash

https://indico.cern.ch/contributionDisplay.py?sessionId=3&contribId=394&confId=149557



Event Reconstruction in the PandaRoot Framework

Stefano Spataro

for the panda collaboration

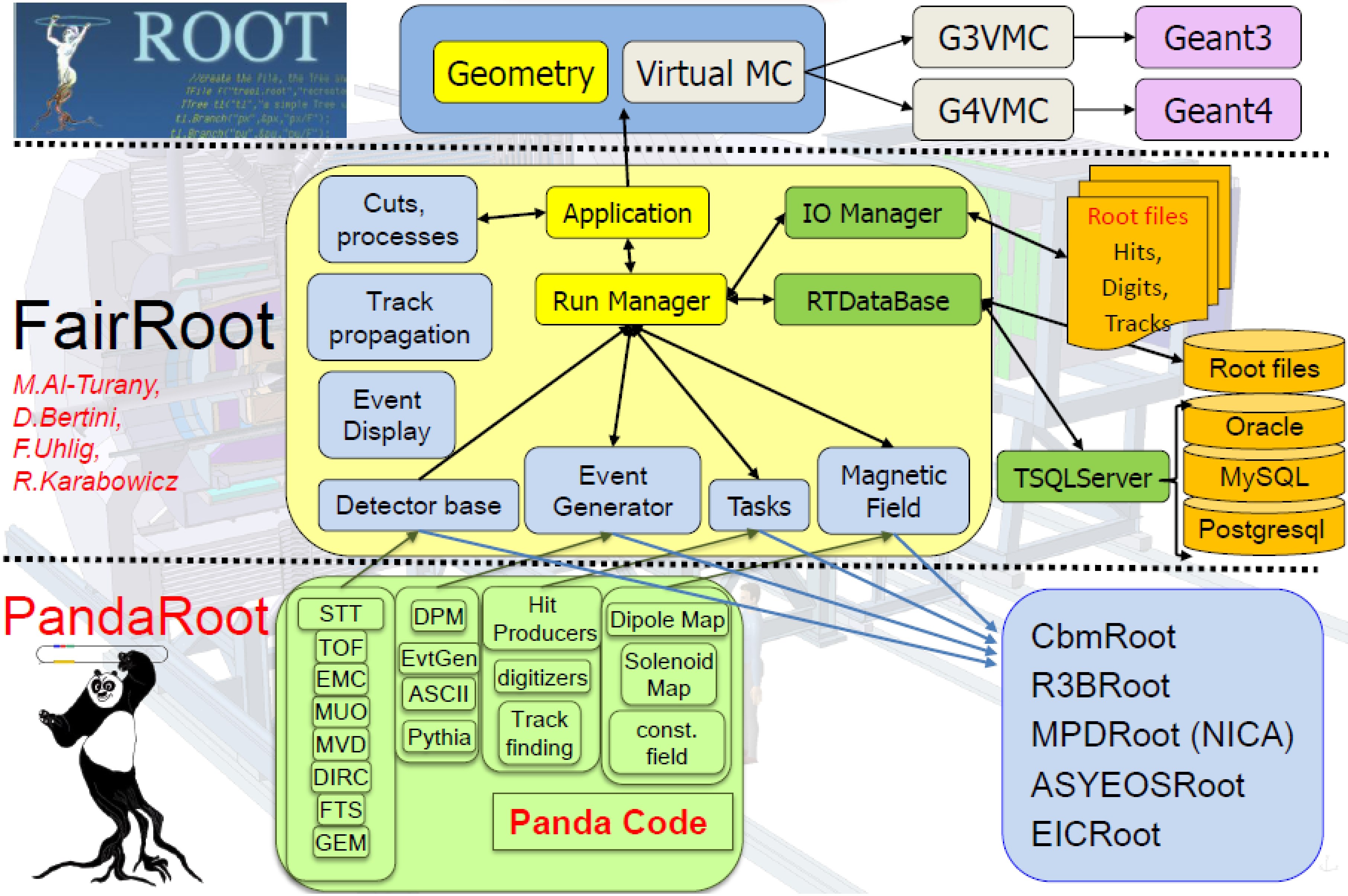
UNIVERSITÀ DEGLI STUDI DI TORINO
ALMA UNIVERSITAS TAURINENSIS

ISTITUTO NAZIONALE DI FISICA NUCLEARE
Sezione di Torino

CHEP 2012

Tuesday, 22th May, 2012

Code Design



ROOT

Geometry

Virtual MC

G3VMC

G4VMC

Geant3

Geant4

FairRoot

M.Al-Turany,
D.Bertini,
F.Uhlig,
R.Karabowicz

Cuts, processes

Application

IO Manager

Track propagation

Run Manager

RTDataBase

Event Display

Detector base

Event Generator

Tasks

Magnetic Field

Root files
Hits,
Digits,
Tracks

Root files

Oracle

MySQL

Postgresql

TSQLServer

PandaRoot

STT
TOF
EMC
MUO
MVD
DIRC
FTS
GEM

DPM
EvtGen
ASCII
Pythia

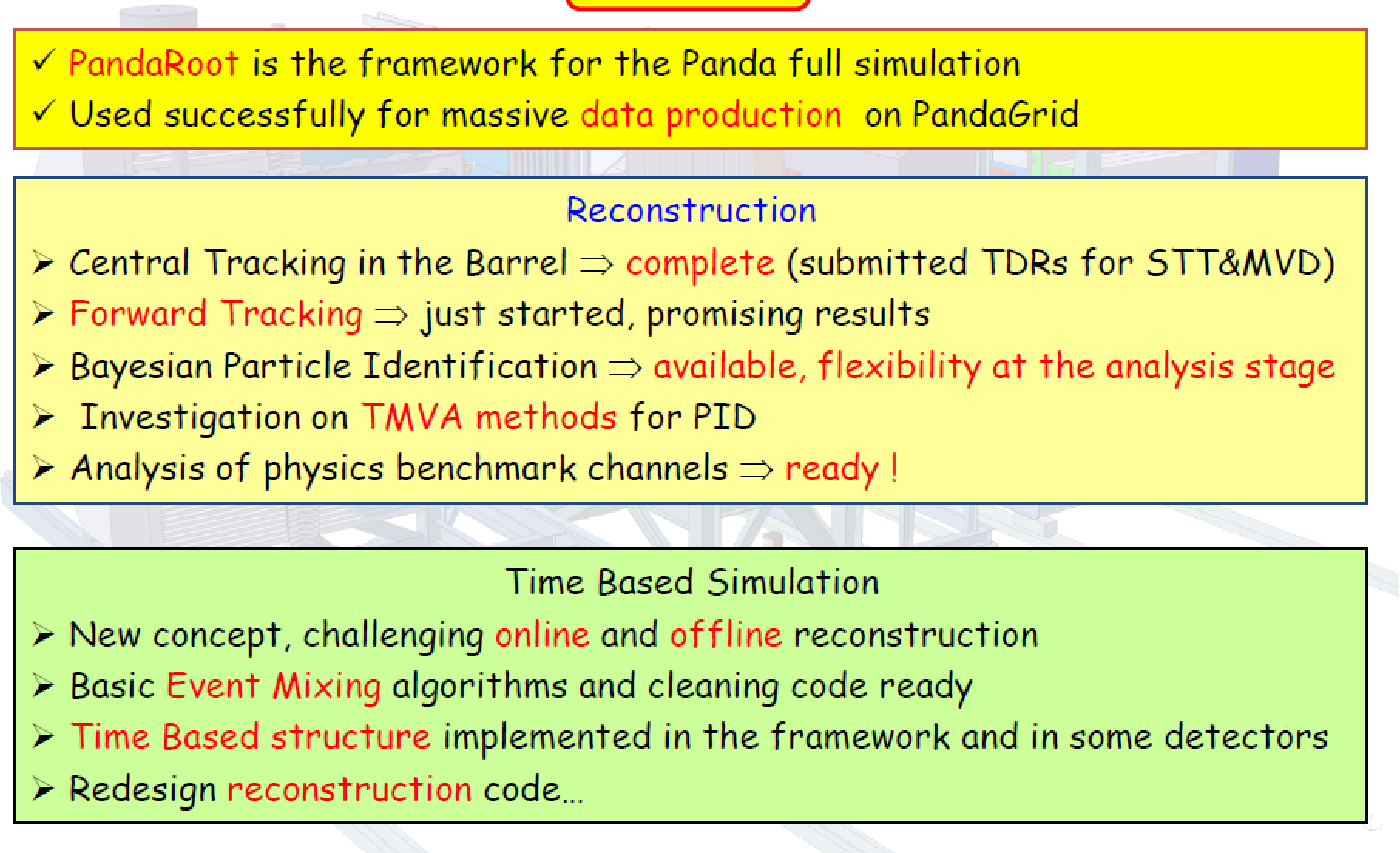
Hit Producers
digitizers
Track finding

Dipole Map
Solenoid Map
const. field

Panda Code

CbmRoot
R3BRoot
MPDRoot (NICA)
ASYEOSRoot
EICRoot

Summary



- ✓ PandaRoot is the framework for the Panda full simulation
- ✓ Used successfully for massive data production on PandaGrid

Reconstruction

- ➢ Central Tracking in the Barrel ⇒ complete (submitted TDRs for STT&MVD)
- ➢ Forward Tracking ⇒ just started, promising results
- ➢ Bayesian Particle Identification ⇒ available, flexibility at the analysis stage
- ➢ Investigation on TMVA methods for PID
- ➢ Analysis of physics benchmark channels ⇒ ready !

Time Based Simulation

- ➢ New concept, challenging online and offline reconstruction
- ➢ Basic Event Mixing algorithms and cleaning code ready
- ➢ Time Based structure implemented in the framework and in some detectors
- ➢ Redesign reconstruction code…