



# EDM and Framework Project Overview & Status



# Outline

---

- Introduction to concepts
- Highlights of the road map
- Project Plan
  - Phase 0 – Bare Bones
  - Phase 1 – First Implementation
  - Phase 2 – All requirements for magnet test
  - Phase 3 – Adjustments based on user feedback
- Status of Implementation



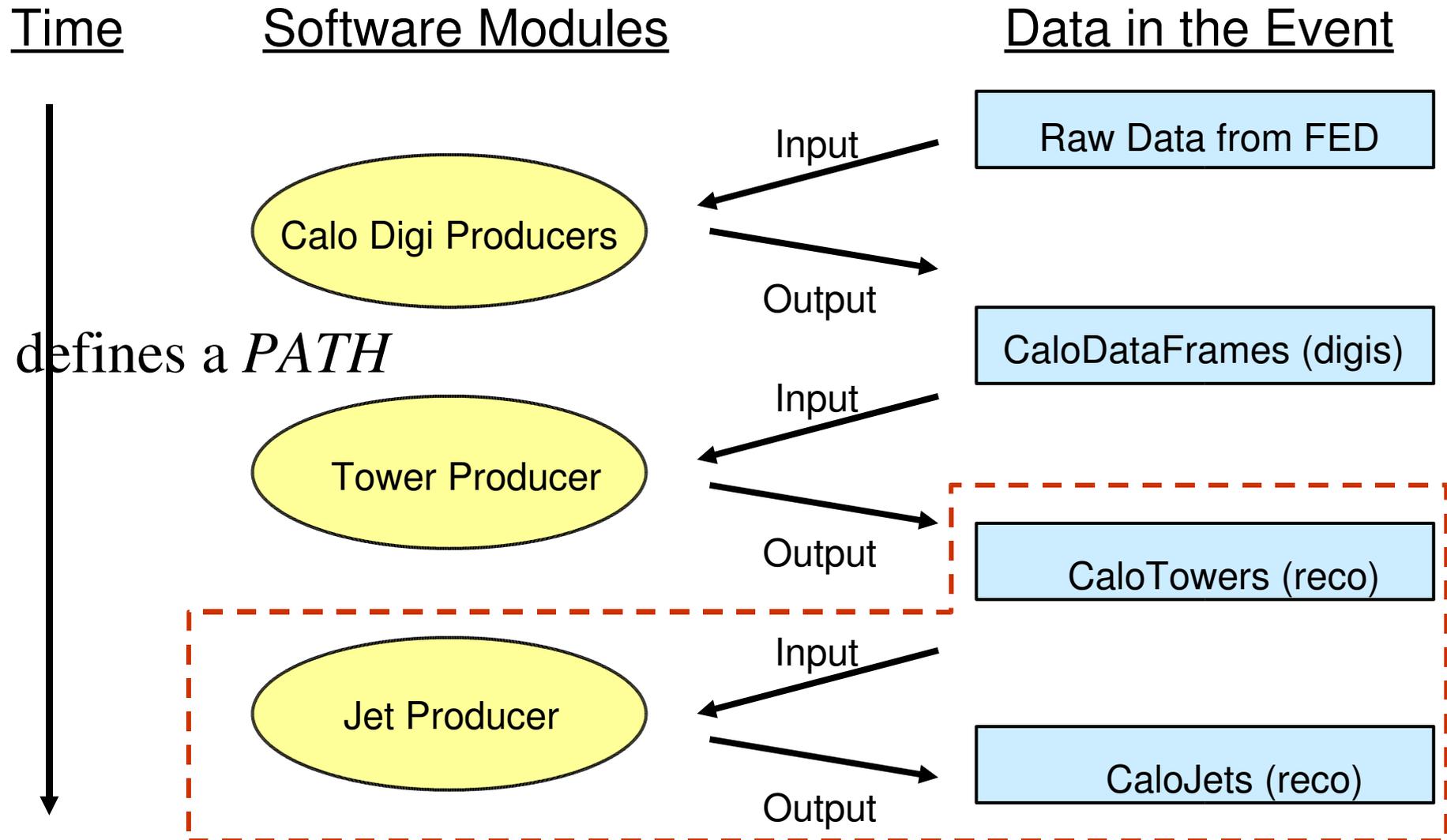
# Introduction to Concepts

---

- A Software Bus Model
  - Start from the raw-raw data
  - Producers are scheduled to operate on the event data and produce their output which is written into the event
  - At any point in the processing chain, the execution can be halted and the contents of the event can be examined outside of the context of the process that made it
  - The schedule can be checked for correctness since the modules and declare their inputs (and outputs if they are EDProducers)



# Example from Calorimetry





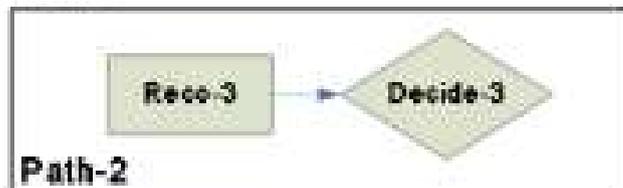
# “Where do I plug in ?”

---

- As you can see from the previous slide each box must be defined by a reconstruction writer.
- First step define what are the input and output objects needed for your algorithmic task
- These need to be defined as C++ classes that follow the EDM rules
- Decide what services your algorithm needs (calibrations, geometry description)
- Write the EDProducer that receives all of the information and passes it on to the algorithms



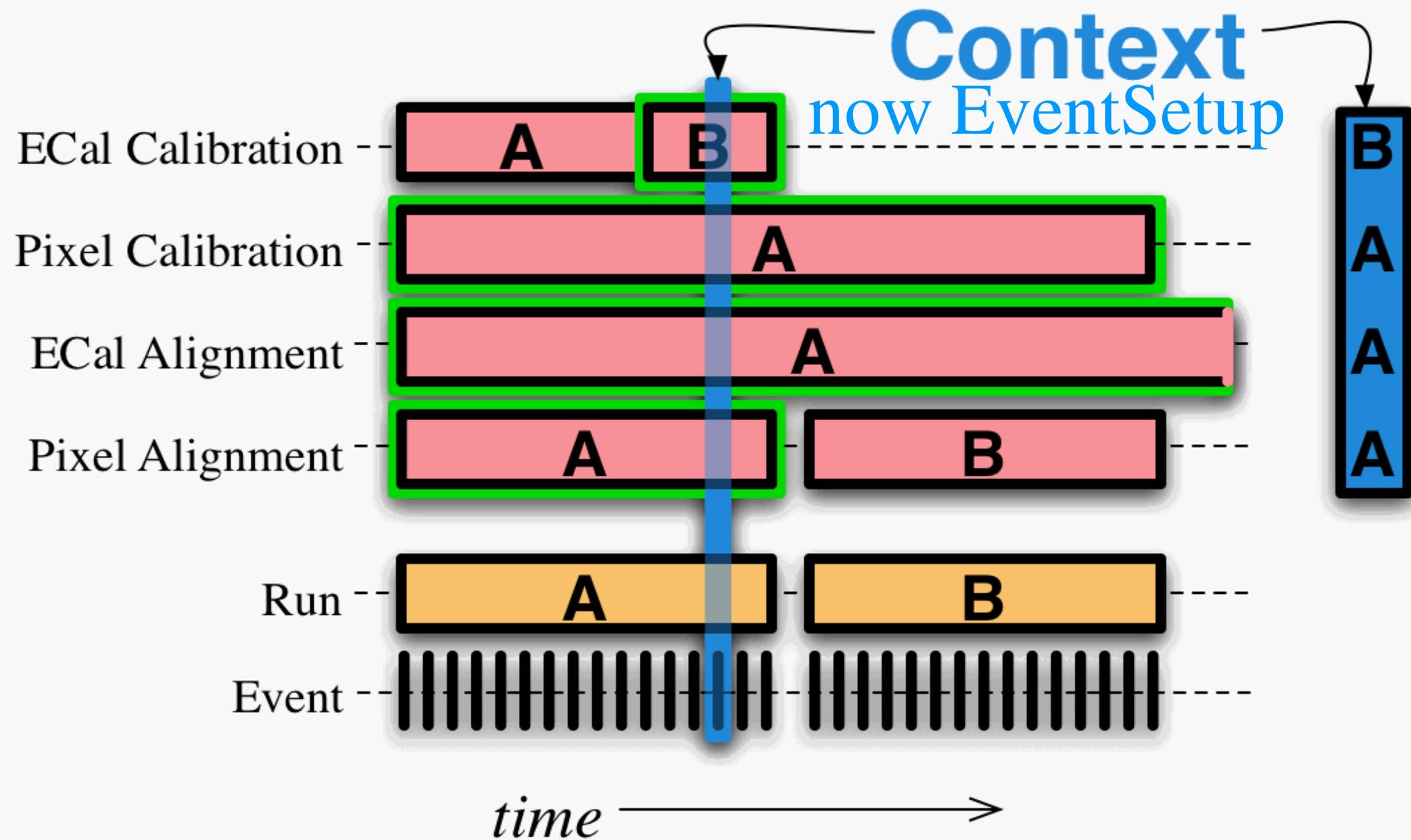
# Explicit Scheduling



- Paths are independent triggers or analysis
- It must be possible to do a complete coverage analysis in the case of the HLT where there are  $O(100)$  paths



# EventSetup





# Highlights from the RoadMap

---

- An evolving document whose main purpose is to aid in the development process.
- Use as a reference, and starting point for future project documentation
- Architectural Overview (sec.4):
- Analysis – describes the reasoning for choices (sec.5)
- Design (sec. 6)
  - *EventProcessor* created by main, handles configuration and Seal loadable factories for framework components
  -



# Highlights from the RoadMap (cont.)<sup>9.</sup>

---

- Design (cont.)
  - *return EventProcessor.run()* can be last line of main
  - *Event* – interface that the user sees, get and put defined here
  - *EventPrinciple* – responsible for exception safe transaction model as well as provenance tracking. It is the real container
  - *EDProducts* – event data objects (a concept?)
  - *Provenance* – describes how associated product was made
  - *Selectors* – lookup based on Provenance



# Highlights from the RoadMap (cont.)

---

- Design (cont.)
  - *ScheduleBuilder* – collects path configuration into a schedule
  - *ScheduleExecuter* – loops over the above for an event
  - *Modules (Eproducers, EDAnalyzer, EDFilter, OutputModule)* – components invoked by *ScheduleExecuter* for each event
  - *InputService* – source of primary event
  - *ModuleRegistry* – owner of *Modules* and their wrappers
  - *ParameterSet* – part of the configuration system also used by the *ScheduleBuilder* to define the application



# Highlights from the RoadMap (cont.)

---

- Design (cont.)
  - *EventSetup* – Source of all non-event data, Conditions, Calibrations, Geometry, ect. Manages IOV and data is retrieved on request
  - *ESSource* – reads the data from disk/DB, like an *InputService*
  - *ESProducer* – optional creator of hybrid objects. For example a TrackerGeometry may depend on the perfect geometry as well as an Alignment DB record both are used by the *ESProducer* to make TrakerGeometry available from *EventSetup*



# Highlights from the RoadMap (cont.)

---

- Design (cont.)
  - Note that even though there are many similarities to the event processing system in *EventSetup*, versioning and tagging of this data is expected to be kept elsewhere. Therefore the requirements on the provenance tracking system are less. Also for most applications a single instance of any particular type (like *TrackerGeometry*) is all that is required.



# Other Services

---

- Generator Service – provides CMS interface to externally supplied generators like Pythia, provides persistence for MC truth event information
- Magnetic Field Service – The design calls for this being delivered through *EventSetup* since it will have an IOV
- Pile Up Handling – The framework must provide a mechanism for handling multiple events in order to support this application



# Project Plan

- Phase 0 – complete by 1-Jun *toward* **CPT-DS-201, DONE**  
**tagged as the pre3 release of CMSSW**
  - **Module registry** – interface for iterating thorough worker instances and getting workers using parameter set and job config done. Manages lifetime of workers successfully
  - **Schedule Builder** – Very rudimentary does what it's told with one path. Uses parsed path expression to create a schedule in the form of a list of lists
  - **Schedule Executer** – Gets event from the event source. Propagates event through list of lists produced above. Filters terminate a path.

The above work is being done by Stefano Argiro in consultation with Jim Kowalkowski and Mark Paterno



# Project Plan

- **Phase 0 – CPT-DS-201**

- **Event Processor** – implements module factory creation with correct configuration, starts the event loop and invokes the schedule executor appropriately. **by Mark and Jim**
- **Event Setup Factory** – creates ESModules and ESProducers according to configuration file. **by Chris Jones**
- **EventSetup** – Can successfully deliver composit data with the right IOV **as above**
- **Input Service** – PoolInputService successfully delivers events to the EventProcessor
- **Output Service** – PoolOutputService successfully persists EDProducts **by Bill Tanenbaum and Luca Lista**
- **Parameter Set System** – Configuration file defined, parsing adequate **by Walter Brown, Mark, Jim, Bill and Chris**



# Project Plan (cont.)

---

- Phase 1 – complete by 1-Aug *toward* **CPT-DS-203**
  - Schedule Builder – redundancy removal, sequence handling, schedule verification  
**Stefano**
  - Schedule Executer – add error and exception handling **Stefano**
  - Event Processor – add BeginRun and others, seal utilities added, improved command line handling **Marc and Jim**
  - Module requirements interface defined **Stefano**
  - Input Service – pileup handling **Ursula Berthon et. al**
  - Output Service – event streaming capability added **Luca or Bill**



# Other Services

---

## Phase 1 (cont.)

**Parameter Set System** — agreement on user interface, ID to name lookup

implemented locally **Marc and Bill**

**EventSetup** — additional information interface during IOV changes, optionally force the

running of ESProducers at the beginning of event **Chris**

**Start collaboration with the data management group to define interfaces and first implementation. This includes definition of command line options, dataset selection, and output handling** **Liz, Greg+, Peter+**

**Start collaboration with EvF group for event streaming/logging and run handling capability** **Liz, Harry, Vincenzo...**



# Project Plan (cont.)

---

- Phase 2 – complete by 1-Oct *toward* **CPT-DS-208**
  - Schedule Builder – verification of schedule, **Stefano**
  - ParameterSet persistence, design of global scope database worked out and agreed upon by DM, **Marc and DM group**
  - path decisions made persistent in the event **Bill**
  - Revisit and revise issues in initial implementation **All**
  - Make any adjustments needed to support the physics TDR work **All**



# Project Plan (cont.)

---

- Phase 3 – complete by or before CSA-2006
  - Continue work to bring the project to production quality. This includes addressing issues of scalability and performance.
  - Allow for on the fly module reconfiguration
  - Implement unscheduled executor
  - Revisit any design problems found in the prototype



# Conclusions

---

- A bare bones framework is now available
- So far phase 1 is on track; however it is very ambitious
- The planning for the project is near completion