Design example
HEP Offline SW

Benedikt Hegner
(CERN)

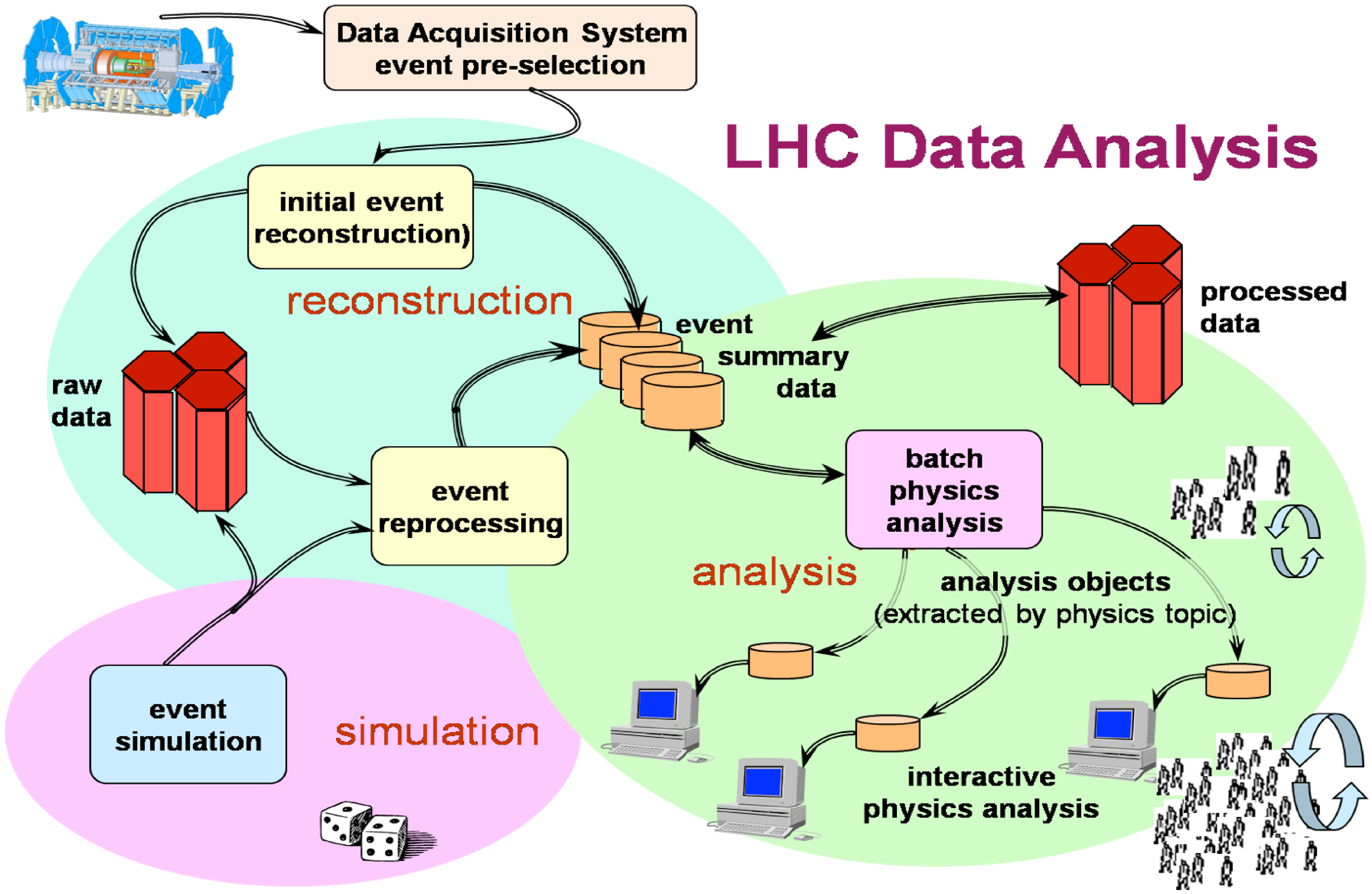
LHC Computing Characteristics

- Large number of physicists and engineers participating actively in the data analysis and for extended period of time

- Widely distributed computing environment

- Huge quantity of data that has to be distributed and shared by all members of each experiment

Data Flow Processing and Stages



LHC Software Requirements

- Design should take into account the >15 years lifetime of the LHC

- Resilient designs, technology choices will evolve over time

- The standard language for physics applications software in all four LHC experiments is C++

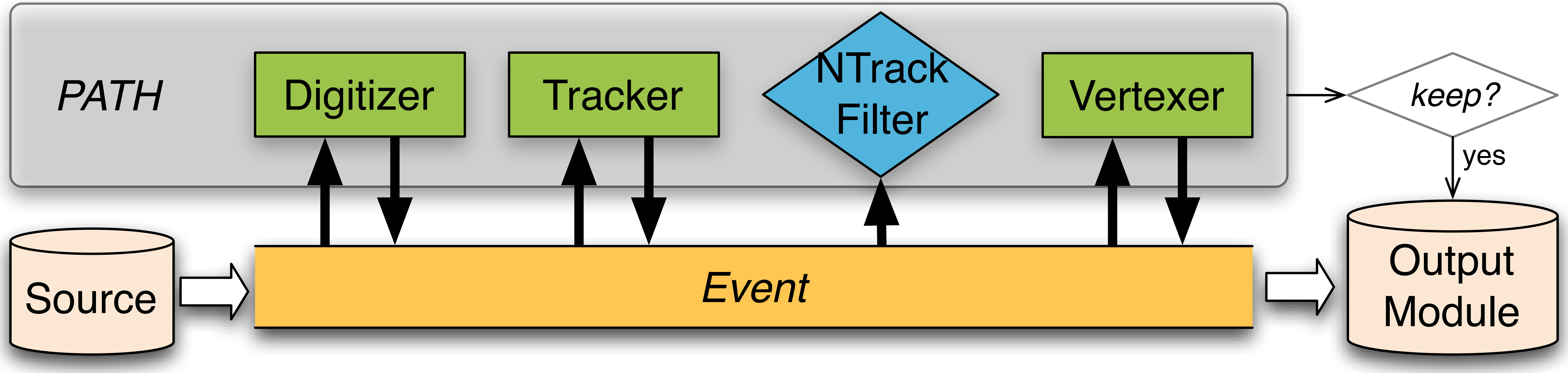
- Language choice may change in the future or multi-language could be introduced

- Operate seamlessly in a distributed environment and also be functional in ‘disconnected’ local environments

- Modularity of components with well-defined interfaces and interchangeability of implementations

One of the Principal Design Choices

- Clear separation between data and algorithms
- Data store-centered ("blackboard") architectural style
- Well defined component "interfaces" with plug-in capabilities
- Objects serialized with ROOT to disk

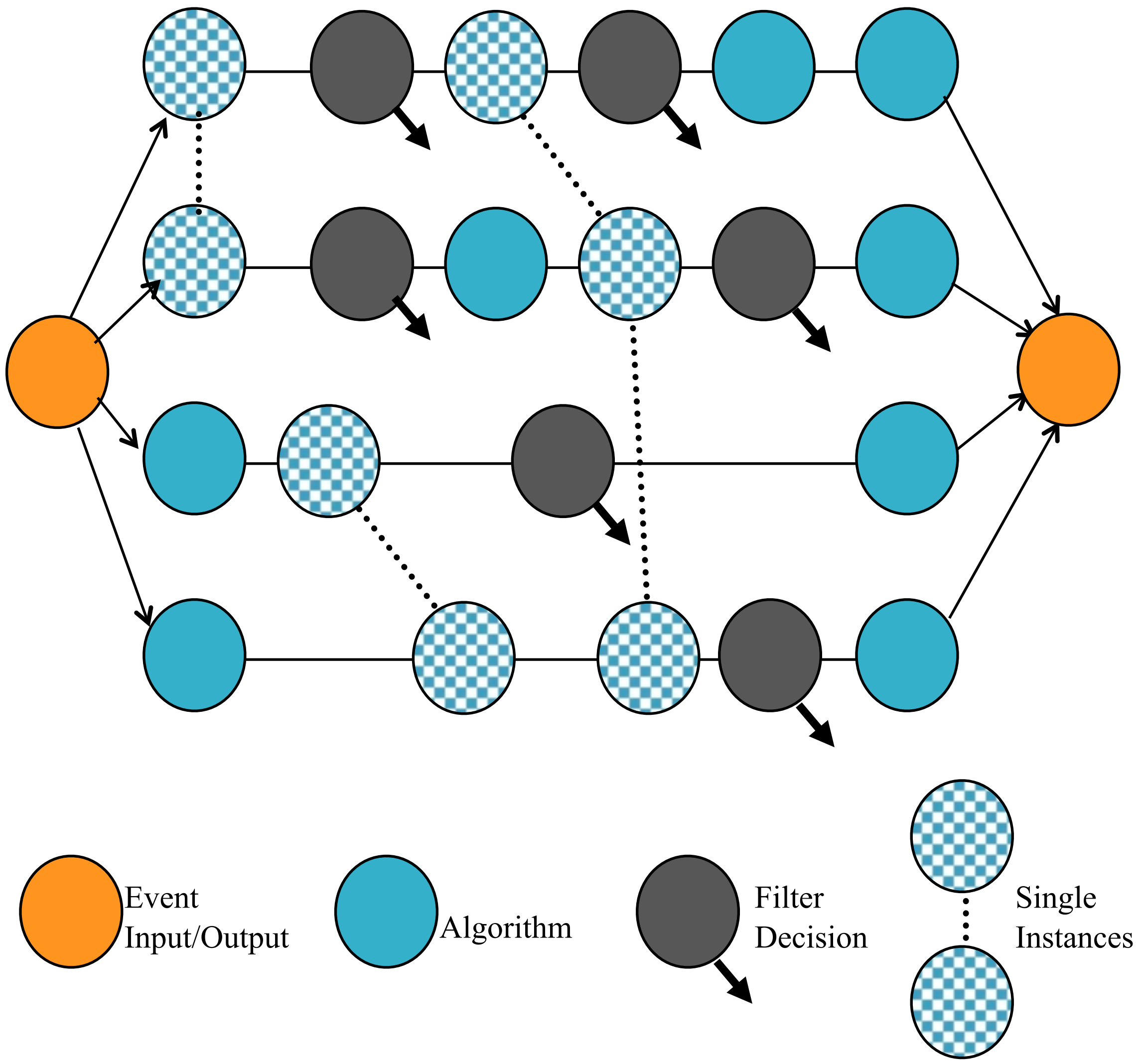


Complex Control Sequences

- Concept of sequences to group various interdependent modules together
  - Avoid recalling same module on same data
  - Different instances of same module possible

- Event filtering
  - Avoid passing all events through all the processing chain

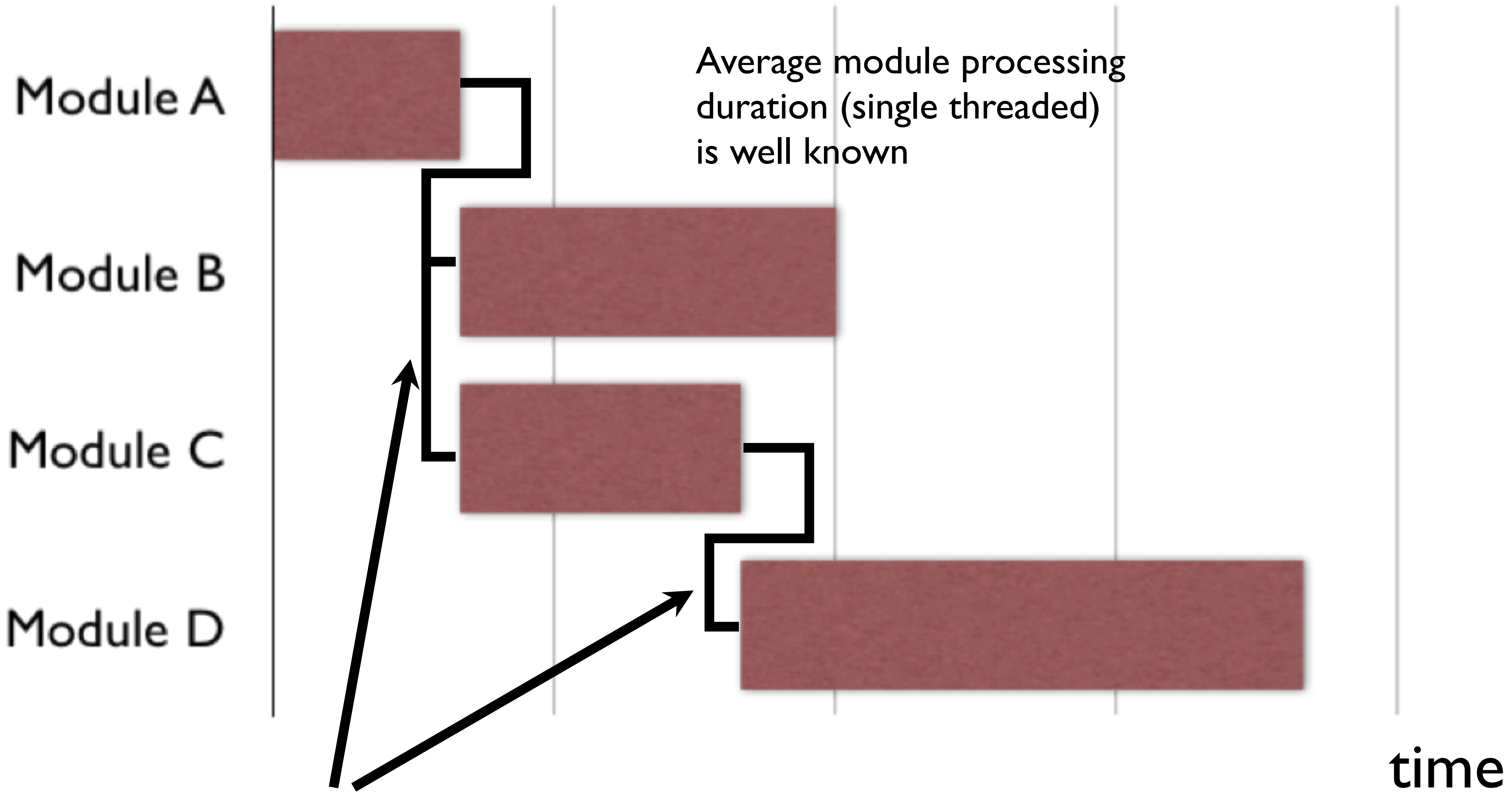
- Module dependencies are a directed acyclic graph



Complex Control Sequences (2)

```
import FWCore.ParameterSet.Config as cms

process = cms.Process("EXAMPLE")
process.source = cms.Source("EmptySource")
process.maxEvents = cms.untracked.PSet( input = cms.untracked.int32(100) )

process.thingy = cms.EDProducer("ThingProducer")

process.test = cms.EDAnalyzer("ThingConsumer",
                               input = cms.untracked.InputTag("thingy")
                              )

process.p = cms.Path( process.thingy * process.test)
```

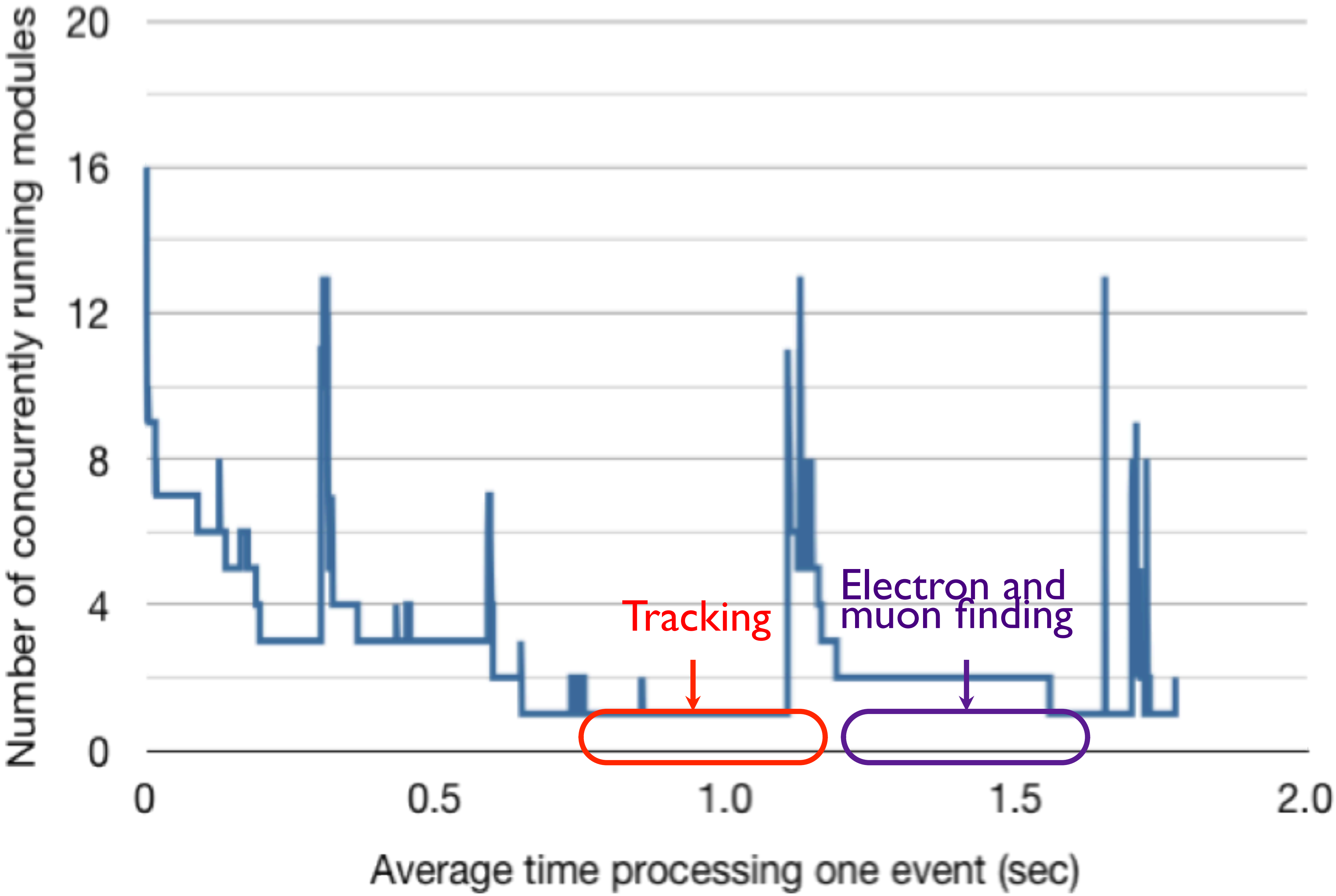
So can’t we run our modules in parallel?



Average module processing duration (single threaded) is well known

Module dependencies are known

Unfortunately it doesn’t work too well with our current SW



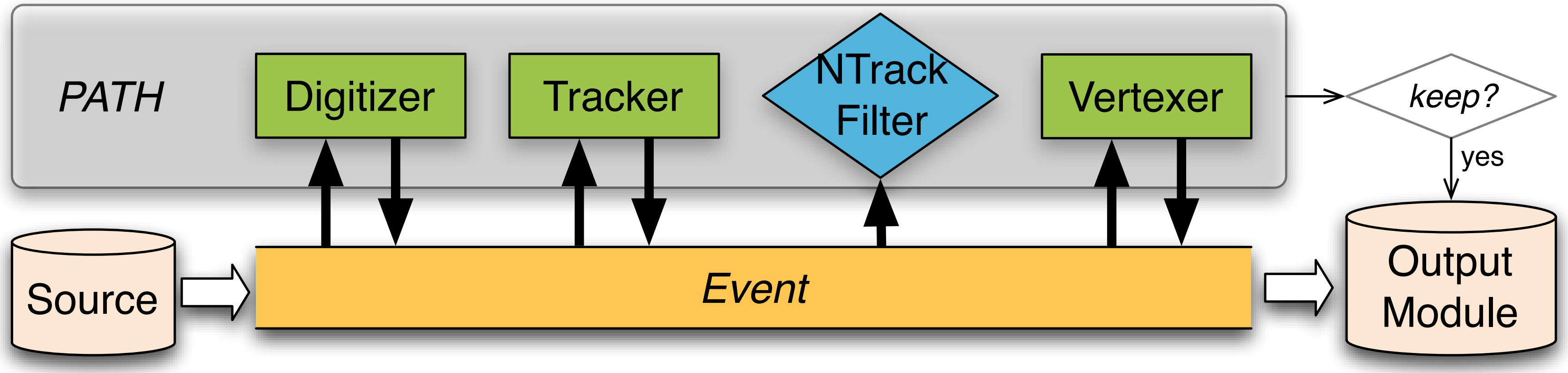
So we need to parallelize around and inside it

What about more efficient data structures?

- OOP as dreamed of in the books:
  - It combines data and algorithms into a single entity
  - It ensures that the developer does not need to code up the control flow explicitly.

- We already violate this with the ‘blackboard pattern’
  - The stored objects are mainly only data
  - We define the control flow explicitly
  - Data transformations happen in modules

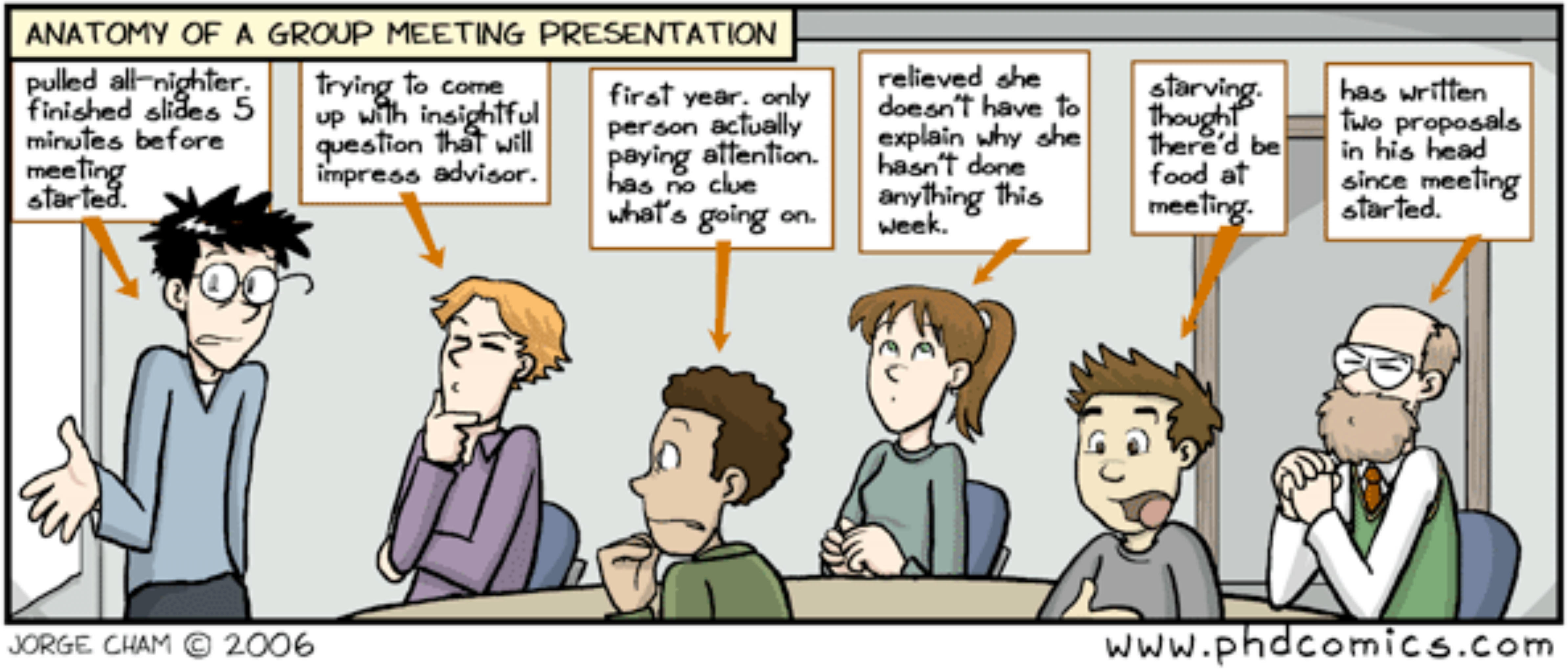
- Leaves room to switch to data oriented design



What’s ahead of us?

- We have to choose with more thought when to follow which programming paradigm
    - Many identical data chunks & high throughput => data oriented
    - Small number of objects & heterogenous data => object oriented
- For a lot of the use cases we have to redesign our data formats to become much dumber
    - expert operation
- Analysis and other cases much more heterogenous
    - “data-to-smart object” translation layer to ease the use?
- Parallelizing frameworks and algorithms

That's it :-)


ANATOMY OF A GROUP MEETING PRESENTATION
pulled all-nighter. finished slides 5 minutes before meeting started.
trying to come up with insightful question that will impress advisor.
first year. only person actually paying attention. has no clue what's going on.
relieved she doesn't have to explain why she hasn't done anything this week.
starving. thought there'd be food at meeting.
has written two proposals in his head since meeting started.
JORGE CHAM © 2006
www.phdcomics.com