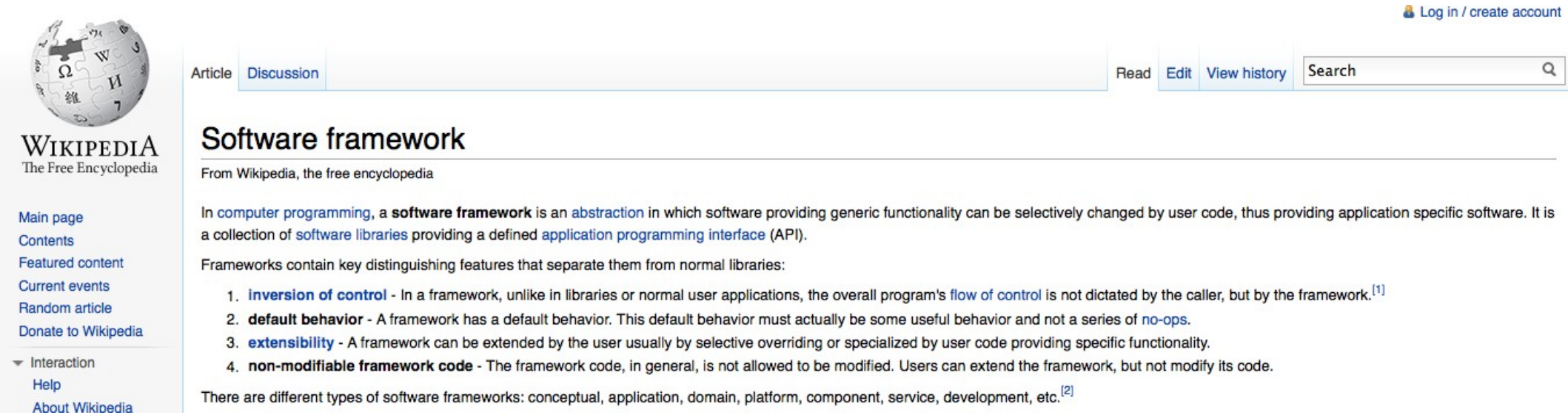
Event Processing Frameworks

Peter Elmer
Princeton University
SuperB Computing R&D Workshop
05 July, 2011

Software Frameworks



Log in / create account

Article | Discussion | Read | Edit | View history | Search

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Interaction
Help
About Wikipedia

Software framework

From Wikipedia, the free encyclopedia

In computer programming, a **software framework** is an abstraction in which software providing generic functionality can be selectively changed by user code, thus providing application specific software. It is a collection of software libraries providing a defined application programming interface (API).

Frameworks contain key distinguishing features that separate them from normal libraries:

1. **inversion of control** - In a framework, unlike in libraries or normal user applications, the overall program's flow of control is not dictated by the caller, but by the framework.[1]
2. **default behavior** - A framework has a default behavior. This default behavior must actually be some useful behavior and not a series of no-ops.
3. **extensibility** - A framework can be extended by the user usually by selective overriding or specialized by user code providing specific functionality.
4. **non-modifiable framework code** - The framework code, in general, is not allowed to be modified. Users can extend the framework, but not modify its code.

There are different types of software frameworks: conceptual, application, domain, platform, component, service, development, etc.[2]

In this talk I'm going to focus on the “Event Processing Framework” and a few of the important services/associated-libraries

See Pere's talk last year for a more general description of frameworks within the typical HEP software stack

Event Processing Framework

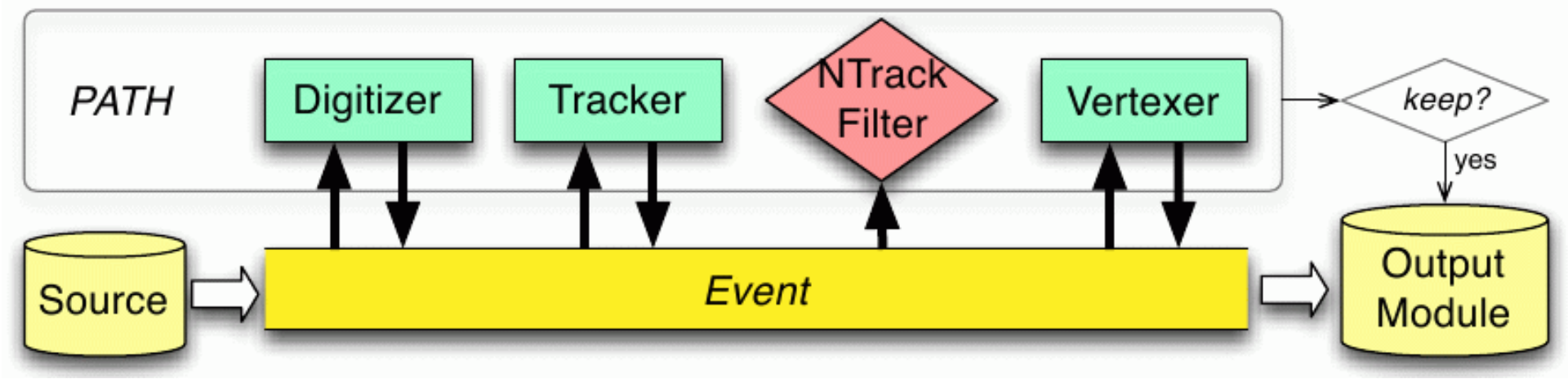
- Basic idea: we don't have one “application”, but in fact many applications.
- The generic terms “Simulation”, “Reconstruction”, “Skimming” and “Analysis” cover a plurality of possibilities
- Individual collaborators may want to run code with different configurations or mix and match code from others with their own

Very Basic Picture

Basic architectural ideas:

1) Separate the problem conceptually into "data" and algorithms operating on that data ("Event bus model", "Whiteboard Model")
2) Those using the application write "modules" ("algorithms") which can use individual data products to produce new event data products, make decisions (produce histos, etc.)
   The event data model is also extensible with new event data types

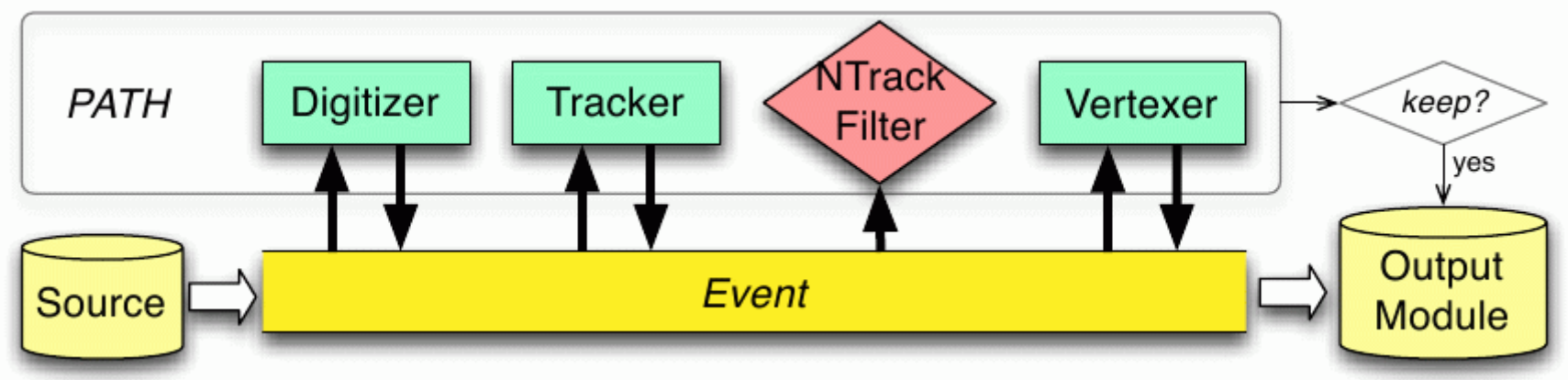
The (event-processing) "Framework" is responsible for getting data into and out of the "event bus" and for running a series of "modules" in an event loop:



Advanced Features

In practice event processing frameworks provide a number of other features:

1) Dynamic configuration via a control script language (e.g. TCL, Python)
2) Application building not only by "modules", but also by groups of modules ("sequences", "paths")
3) Multiple output modules (and at times multiple inputs), filtering choices to stop processing or writing to an output. Algorithmic code is independent of the specifics of how data is read or written.
4) Encapsulation, interface standardization and eased interoperability with "other frameworks" (e.g. generators, Geant4, etc.)
5) Various internal services (see later slide)



Basic interface to override

```
namespace edm {
  class EDProducer : public ProducerBase {
  public:
    // ....
    EDProducer ();
    virtual ~EDProducer();

  private:
    virtual void produce(Event&, EventSetup const&) = 0;
    virtual void beginJob() {}
    virtual void endJob(){}
    virtual void beginRun(Run&, EventSetup const&){}
    virtual void endRun(Run&, EventSetup const&){}

    virtual void beginLuminosityBlock(LuminosityBlock&, EventSetup const&){}
    virtual void endLuminosityBlock(LuminosityBlock&, EventSetup const&){}
  // ...
  }
}
```

(CMS Example)

CMS "Analyzer" modules

Counter: PERF_TICKS

Rank	% total	Counts: to / from this	Counts: Total	Paths: Including child / parent	Paths: Total	Symbol name
	10.82	200.97	200.98	3	3	edm::WorkerT<edm::EDAnalyzer>::implDoBegin(edm::EventPrincipal&, edm::EventSetup const&, edm::CurrentProcessingContext const*)
[21]	10.82	0.01	200.96	3	3	**edm::EDAnalyzer::doEvent(edm::EventPrincipal const&, edm::EventSetup const&, edm::CurrentProcessingContext const*)**
	2.32	43.04	43.04	3	3	HLTMuonOfflineAnalyzer::analyze(edm::Event const&, edm::EventSetup const&)
	1.63	30.27	30.27	3	3	SiStripMonitorTrack::analyze(edm::Event const&, edm::EventSetup const&)
	1.08	20.11	20.11	3	3	SiPixelTrackResidualSource::analyze(edm::Event const&, edm::EventSetup const&)
	0.94	17.43	17.43	3	3	JetMETAnalyzer::analyze(edm::Event const&, edm::EventSetup const&)
	0.81	15.02	15.02	3	3	FourVectorHLTOffline::analyze(edm::Event const&, edm::EventSetup const&)
	0.47	8.66	8.66	3	3	SiStripMonitorCluster::analyze(edm::Event const&, edm::EventSetup const&)
	0.41	7.60	7.60	3	3	SiStripFEDMonitorPlugin::analyze(edm::Event const&, edm::EventSetup const&)
	0.36	6.61	6.61	3	3	TrackingMonitor::analyze(edm::Event const&, edm::EventSetup const&)
	0.32	5.93	5.93	3	3	MonitorTrackResiduals::analyze(edm::Event const&, edm::EventSetup const&)
	0.25	4.55	4.55	3	3	SiStripMonitorDigi::analyze(edm::Event const&, edm::EventSetup const&)
	0.19	3.47	3.47	3	3	EgHLTOfflineSource::analyze(edm::Event const&, edm::EventSetup const&)
	0.17	3.18	3.18	3	3	JetMETHLTOfflineSource::analyze(edm::Event const&, edm::EventSetup const&)
	0.16	2.92	2.92	3	3	SiStripFEDCheckPlugin::analyze(edm::Event const&, edm::EventSetup const&)
	0.15	2.87	2.87	2	2	RPCMonitorDigi::analyze(edm::Event const&, edm::EventSetup const&)
	0.12	2.26	2.26	3	3	CSCMonitorModule::analyze(edm::Event const&, edm::EventSetup const&)
	0.12	2.22	2.22	3	3	EwkMuLumiMonitorDQM::analyze(edm::Event const&, edm::EventSetup const&)
	0.12	2.18	2.18	3	3	PiZeroAnalyzer::analyze(edm::Event const&, edm::EventSetup const&)
	0.09	1.60	1.60	2	2	SiPixelHitEfficiencySource::analyze(edm::Event const&, edm::EventSetup const&)
	0.08	1.56	1.56	3	3	SiPixelDigiSource::analyze(edm::Event const&, edm::EventSetup const&)
	0.08	1.49	1.49	3	3	EEOccupancyTask::analyze(edm::Event const&, edm::EventSetup const&)
	0.07	1.29	1.29	3	3	EBOccupancyTask::analyze(edm::Event const&, edm::EventSetup const&)
	0.07	1.28	1.28	3	3	TkAlCaRecoMonitor::analyze(edm::Event const&, edm::EventSetup const&)
	0.07	1.25	1.25	3	3	EETriggerTowerTask::analyze(edm::Event const&, edm::EventSetup const&)
	0.05	1.00	1.00	2	2	SiPixelHLTSource::analyze(edm::Event const&, edm::EventSetup const&)
	0.05	0.91	0.91	2	2	QcdUeDQM::analyze(edm::Event const&, edm::EventSetup const&)
	0.05	0.89	0.89	2	2	BTagPerformanceAnalyzerOnData::analyze(edm::Event const&, edm::EventSetup const&)
	0.04	0.77	0.77	2	2	HcalRawDataMonitor::analyze(edm::Event const&, edm::EventSetup const&)
	0.04	0.73	0.73	2	2	AlcaBeamMonitor::analyze(edm::Event const&, edm::EventSetup const&)
	0.03	0.55	0.55	3	3	HcalRecHitMonitor::analyze(edm::Event const&, edm::EventSetup const&)
	0.03	0.52	0.52	2	2	SiPixelClusterSource::analyze(edm::Event const&, edm::EventSetup const&)
	0.03	0.52	0.52	2	2	SiPixelRecHitSource::analyze(edm::Event const&, edm::EventSetup const&)
	0.03	0.47	0.47	2	2	HcalNoiseMonitor::analyze(edm::Event const&, edm::EventSetup const&)
	0.03	0.47	0.47	2	2	HcalDigiMonitor::analyze(edm::Event const&, edm::EventSetup const&)
	0.02	0.44	0.44	3	3	PhotonAnalyzer::analyze(edm::Event const&, edm::EventSetup const&)
	0.02	0.43	0.43	3	3	EBTriggerTowerTask::analyze(edm::Event const&, edm::EventSetup const&)

CMS "Producer" modules

Counter: PERF_TICKS

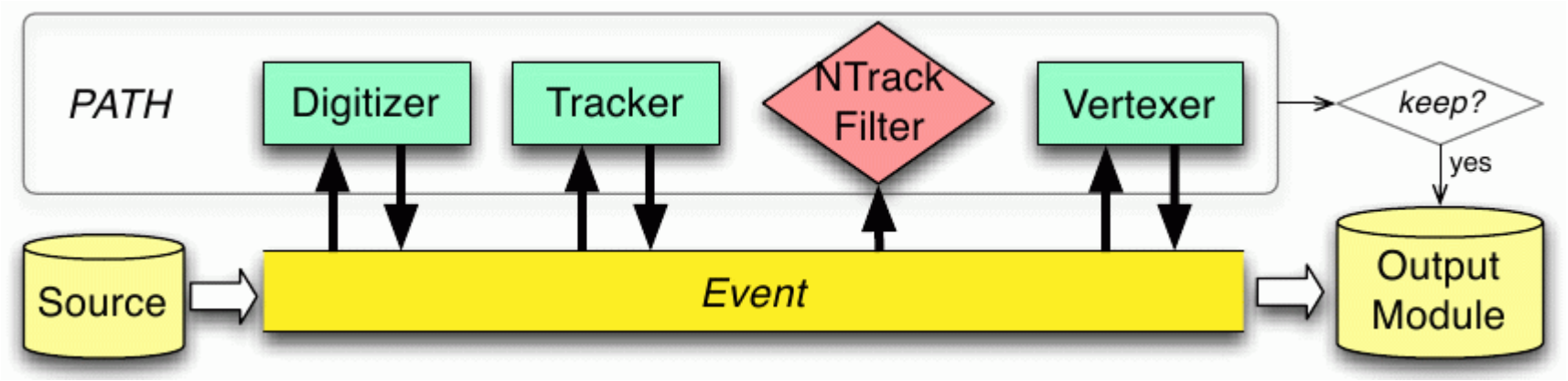
Rank	% total	Counts: to / from this	Counts: Total	Paths: Including child / parent	Paths: Total	Symbol name
	74.13	1,376.68	1,376.73	4	4	edm::WorkerT<edm::EDProducer>::implDoBegin(edm::EventPrincipal&, edm::EventSetup const&, edm::CurrentProcessingContext const*)
[16]	74.13	0.04	1,376.64	4	4	**edm::EDProducer::doEvent(edm::EventPrincipal&, edm::EventSetup const&, edm::CurrentProcessingContext const*)**
	14.97	278.01	278.01	3	3	cms::CkfTrackCandidateMakerBase::produceBase(edm::Event&, edm::EventSetup const&)
	8.52	158.18	158.18	3	3	ConversionProducer::produce(edm::Event&, edm::EventSetup const&)
	7.74	143.75	143.75	3	3	ConversionTrackCandidateProducer::produce(edm::Event&, edm::EventSetup const&)
	4.62	85.76	85.76	3	3	TrackProducer::produce(edm::Event&, edm::EventSetup const&)
	4.12	76.59	76.59	3	3	VirtualJetProducer::produce(edm::Event&, edm::EventSetup const&)
	4.03	74.86	74.86	3	3	GsfTrackProducer::produce(edm::Event&, edm::EventSetup const&)
	3.14	58.29	58.29	3	3	MuonIdProducer::produce(edm::Event&, edm::EventSetup const&)
	2.34	43.50	43.50	3	3	TauDiscriminationProducerBase<reco::PFTau, reco::PFTauDiscriminator>::produce(edm::Event&, edm::EventSetup const&)
	1.93	35.92	35.92	3	3	SeedGeneratorFromRegionHitsEDProducer::produce(edm::Event&, edm::EventSetup const&)
	1.26	23.48	23.48	3	3	RecoTauProducer::produce(edm::Event&, edm::EventSetup const&)
	1.19	22.03	22.03	3	3	SiStripRecHitConverter::produce(edm::Event&, edm::EventSetup const&)
	1.18	21.96	21.96	2	2	CosmicMuonProducer::produce(edm::Event&, edm::EventSetup const&)
	1.12	20.81	20.81	3	3	RecoTauCleanerImpl<std::vector<reco::PFTau, std::allocator<reco::PFTau> > >::produce(edm::Event&, edm::EventSetup const&)
	1.10	20.46	20.46	3	3	PrimaryVertexProducer::produce(edm::Event&, edm::EventSetup const&)
	1.01	18.73	18.73	2	2	AlCaIsoTracksProducer::produce(edm::Event&, edm::EventSetup const&)
	0.92	17.12	17.12	3	3	GoodSeedProducer::produce(edm::Event&, edm::EventSetup const&)
	0.91	16.97	16.97	3	3	EcalUncalibRecHitProducer::produce(edm::Event&, edm::EventSetup const&)
	0.55	10.27	10.27	3	3	PFBlockProducer::produce(edm::Event&, edm::EventSetup const&)
	0.55	10.17	10.17	3	3	cms::SimpleTrackListMerger::produce(edm::Event&, edm::EventSetup const&)
	0.52	9.62	9.62	2	2	SETMuonSeedProducer::produce(edm::Event&, edm::EventSetup const&)
	0.52	9.57	9.57	3	3	cms::METProducer::produce(edm::Event&, edm::EventSetup const&)
	0.45	8.39	8.39	3	3	SecondaryVertexProducer::produce(edm::Event&, edm::EventSetup const&)
	0.45	8.36	8.36	3	3	CaloTowersCreator::produce(edm::Event&, edm::EventSetup const&)
	0.40	7.46	7.46	3	3	SiStripClusterizer::produce(edm::Event&, edm::EventSetup const&)
	0.40	7.38	7.38	3	3	HcalHitReconstructor::produce(edm::Event&, edm::EventSetup const&)
	0.39	7.20	7.20	3	3	PhotonProducer::produce(edm::Event&, edm::EventSetup const&)
	0.36	6.71	6.71	3	3	CaloRecoTauProducer::produce(edm::Event&, edm::EventSetup const&)
	0.36	6.60	6.60	3	3	TrackExtrapolator::produce(edm::Event&, edm::EventSetup const&)
	0.35	6.41	6.41	3	3	PFDisplacedVertexProducer::produce(edm::Event&, edm::EventSetup const&)
	0.33	6.22	6.22	3	3	TCRecoTauProducer::produce(edm::Event&, edm::EventSetup const&)
	0.31	5.80	5.80	3	3	CSCRecHitDProducer::produce(edm::Event&, edm::EventSetup const&)
	0.30	5.62	5.62	3	3	PFRecHitProducer::produce(edm::Event&, edm::EventSetup const&)
	0.30	5.53	5.53	3	3	JetPlusTrackProducer::produce(edm::Event&, edm::EventSetup const&)
	0.29	5.42	5.42	3	3	PFElecTkProducer::produce(edm::Event&, edm::EventSetup const&)

Services

- In addition to the "data/algorithm/event-loop" aspects, the experiment frameworks usually include a number of services:
  - Access to calibrations (varying with event being processed)
  - Access to geometry descriptions (roughly static, except perhaps across long shutdowns), magnetic field
  - Storing of histograms
  - Random number service
  - etc.

Resource Management

This type of "Event Processing Framework" implements implicitly a CPU "resource management" strategy, namely "keep a single core busy 100% by serially running modules, one event after another.



Overall resources are kept occupied by starting one instance of such an application per core and exploiting event-level parallelism. (Sometimes more than one, to the extent that the applications fail to keep the cores occupied.) This part of the resource management is managed by site/grid "workload/job/queue management" systems.

Given that this is how the frameworks work, this is the resource utilization strategy that has been used in the first part of the multicore era

Future resource management

- It is likely that optimum use of future processors can't be easily accomplished by this very simple strategy.
- The natural place to put in more sophisticated (and perhaps fine-grained) resource management is within the experiments event processing frameworks.
- Support for this must come from the underlying frameworks and/or libraries.
- This of course implies additional expectations on any individual “algorithms/modules”
- In addition data structures will need to be organized to support efficient calculation (i.e. no more freewheeling OO ad-hoc data classes in computationally intensive code)

Types of parallelism

- Event level parallelism
- “module level” parallelism
- Algorithmic fine grained (openmp-like) parallelism
- Data (opencl-like) parallelism

- Parallelism at all levels will need to be made explicit and exposed at some level to the Framework for resource management
- Synchronization rather than inter-communication is also a problem