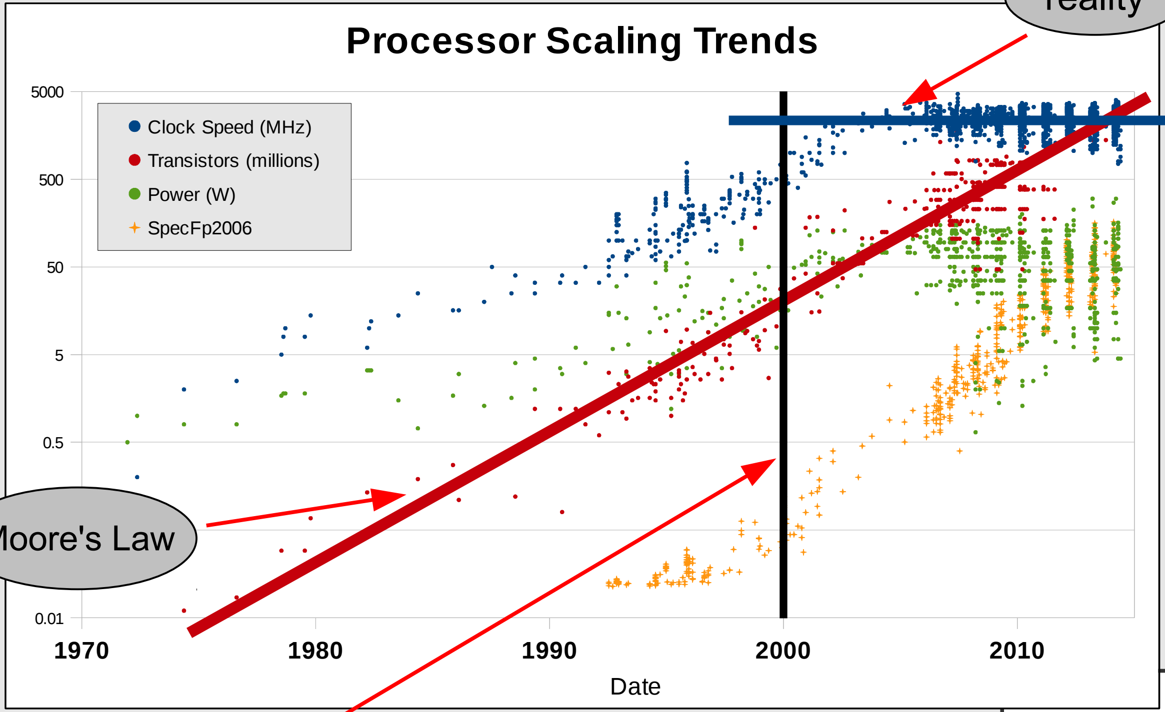
AthenaMT: Upgrading the ATLAS Software Framework for the Many-Core World with Multi-Threading

John Baines, Tomasz Bold, Paolo Calafiura, Steven Farrell, **Charles Leggett**, David Malon, Elmar Ritsch, Graeme Stewart, Scott Snyder, Vakhtang Tsulaia, Benjamin Wynne, Peter Van Gemmeren

for the ATLAS Collaboration

CHEP 2016

Future Computing Challenges



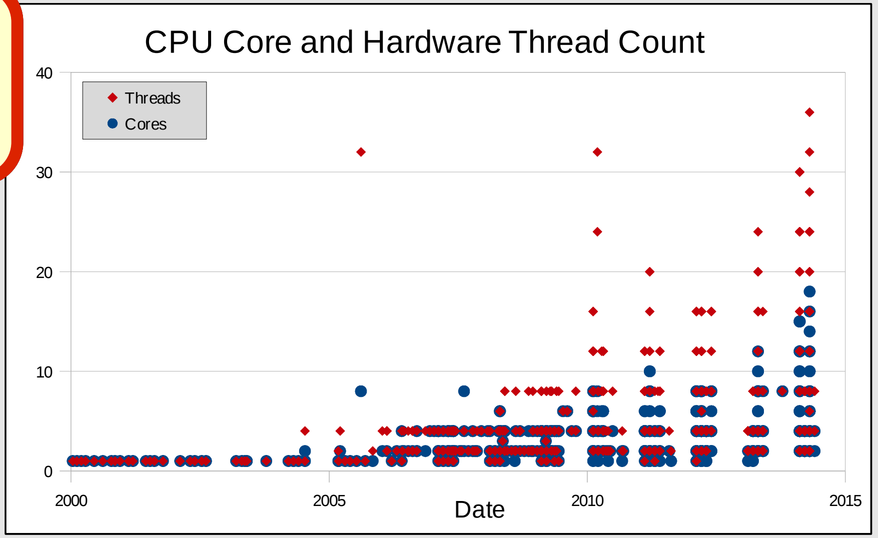
reality

Moore's Law

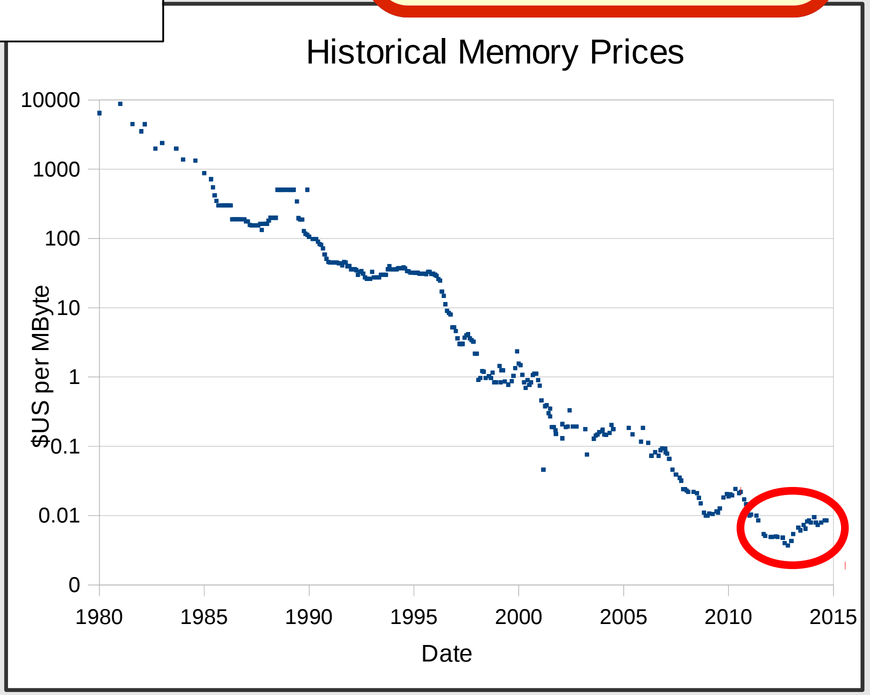
original Gaudi design

CPUs are not getting faster

CPUs are getting wider



memory prices have plateaued



- ATLAS reconstruction uses upwards of 3 GB of memory, more with high luminosity runs
- Cost to equip all grid compute nodes with full memory requirements is more than US$ 6 Million
- Increasing adoption of many-core systems, *eg* Cori, Aurora

Long Term Solution:

Invest in multi-threading

Migration of ATLAS Software to AthenaMT

Date	Framework	Algorithms
2015	Event Store access via Data Handles; Event View design completed; Updated Configuration design; Re-integration of Hive features into Gaudi trunk	Few Algorithms as concurrent prototypes, concentrate on high inherent parallelism; general clean-up of code
2016 Q2	Event Views implemented; IO Layer redesigned; Core Gaudi service migration starts	Wider set, prototype CPU expensive Algs with internal parallelism
2016 Q4	Parallel Algorithm support; Detector/Condition Store re-implementation; Schedulable Incidents; Main Athena development branch moved to Gaudi trunk	First trigger chains running with Event Views; limited reconstruction
2017	All Athena and Gaudi Services made thread safe; Support for re-entrant Algorithms	Serious migration with select groups; Core of useful Algorithms to allow for framework optimization
2018	Framework optimization, and tuning for different hardware	Bulk of Algorithm migration
2019		Integration and Readiness for Run 3

- Aggressive schedule
  - many migrations steps are not parallelizable
- On track for most milestones
  - but not all!

- Will focus on what we've accomplished in 2016

Enabling Concurrency for Core Services

- Majority of hard work in migrating ATLAS code to AthenaMT is in making shared Services thread safe or able to handle multiple concurrent events.

- Some Services can be made concurrent / thread safe with simple mutexes or thread safe data structures

- Some need more modifications to handle state information of multiple concurrent events
  - `MagFieldSvc`: carry event specific cache along with each request
  - `THistSvc`: users can choose whether to share or clone histograms
    - thread safe access to shared histograms via smart locking handles

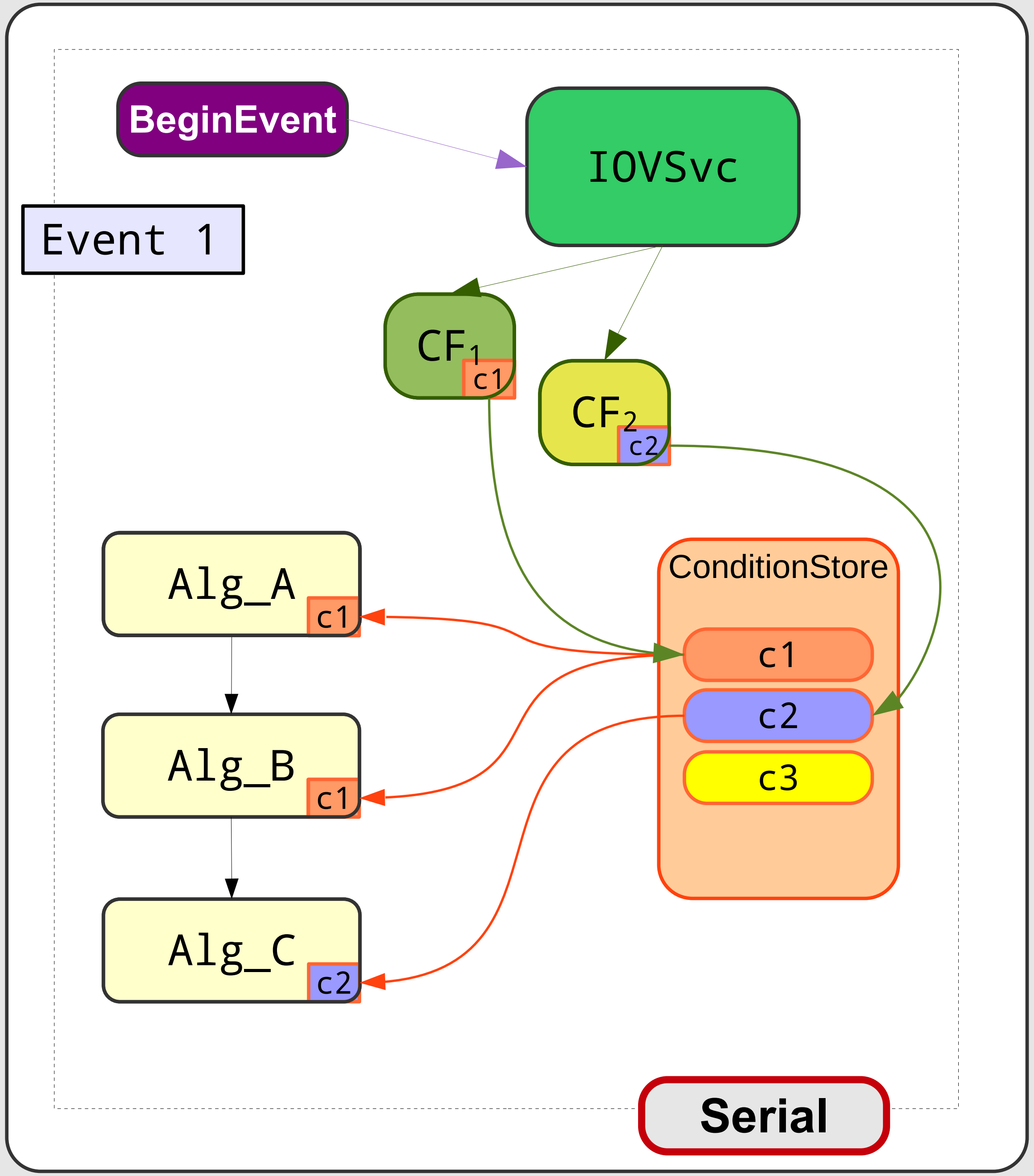
- Some need complete redesign
  - Conditions / `IOVSvc` (Intervals of Validity) / `GeoModelSvc` (Detector Alignment)
  - `IncidentSvc`

Concurrent Processing of Asynchronous Data

- **Conditions**
  - *eg* high voltages, calibrations, *etc*
- **Detector Geometry and Alignments**
  - *eg* position changes

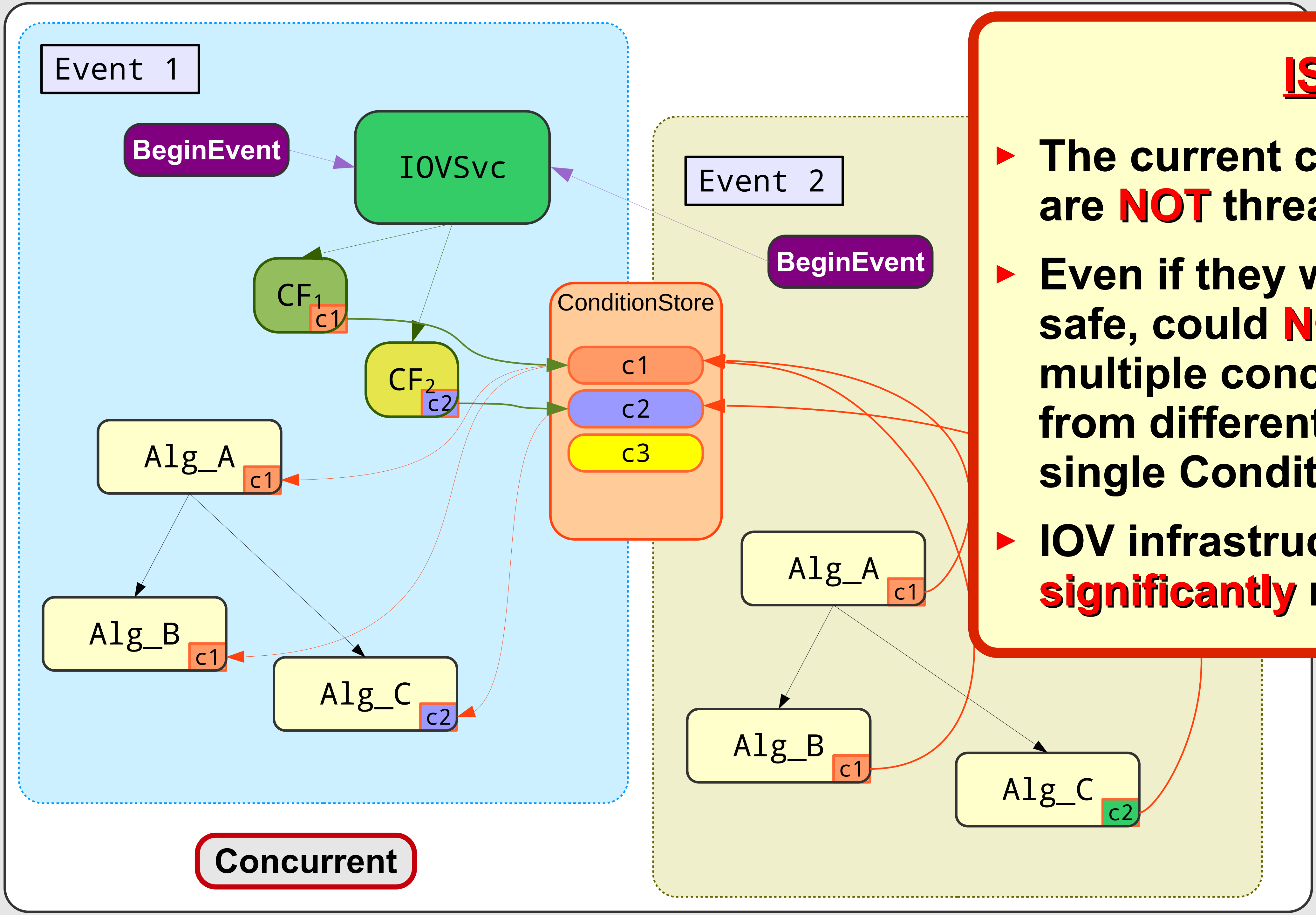
- **Requirement**: Minimize changes to client code
  - there's lots and lots of it!
  - avoid forcing Users to implement fully thread-safe code by handling most thread-safety issues at the framework / Services level (without compromising concurrency)
- **Requirement**: All access to Event data is *via* smart **DataHandles**, which also declare data dependency relationship to the framework
  - we can use this by implementing **ConditionHandles**
    - do all the heavy lifting inside the Handle

Serial Processing with Conditions



- All framework elements process data from the same **I**nterval **O**f **V**alidity
- Algorithms are blind to the IOV, and retrieve data from the **ConditionStore**
- At the start of every Event, **IOVSvc** checks IOVs, and triggers any necessary updates
  - handled by the Callback Functions
  - Callback Functions are **shared** instances
- Only one copy of any Conditions object is maintained in the Store

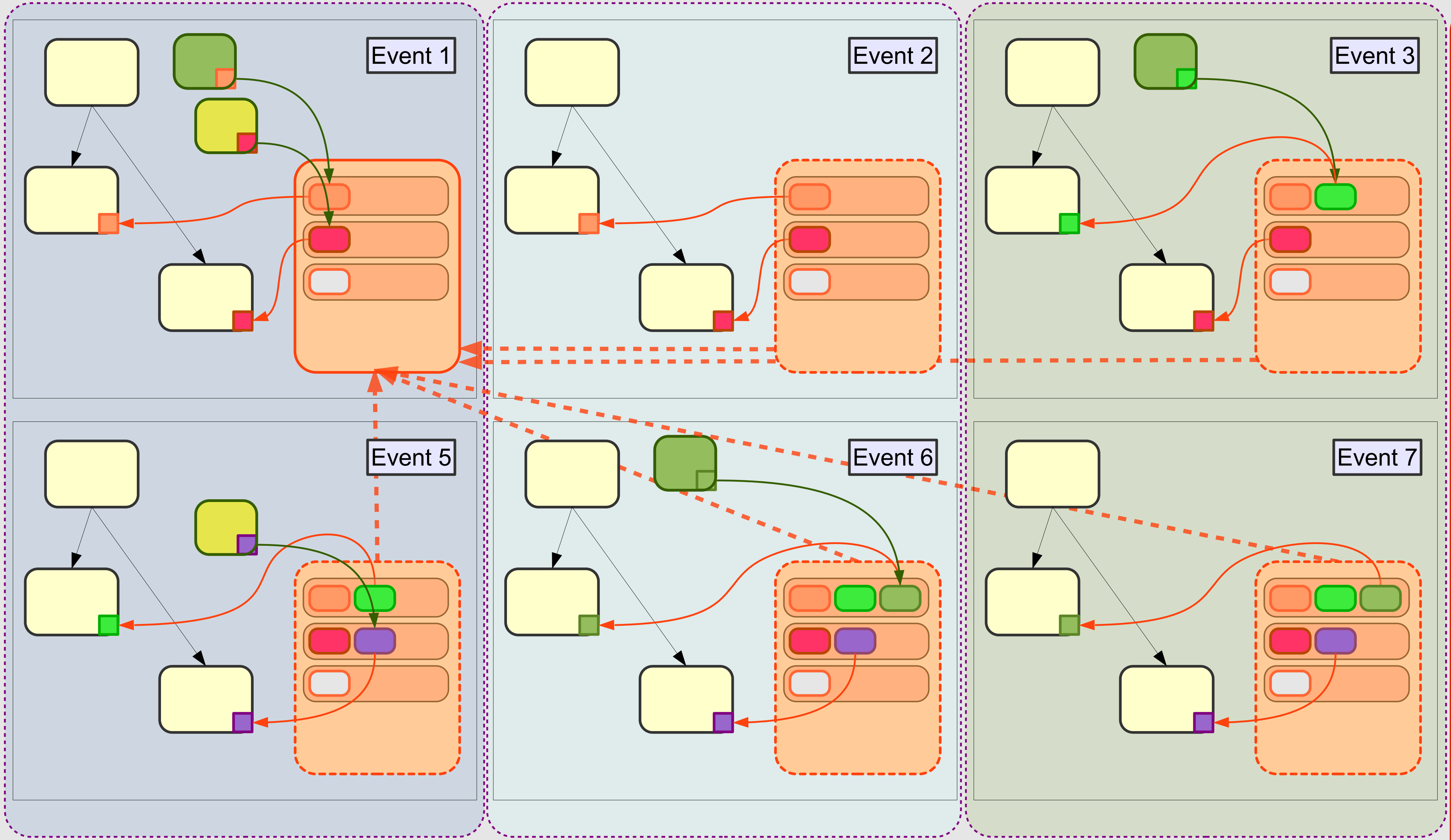
Concurrent Processing with Conditions



ISSUES

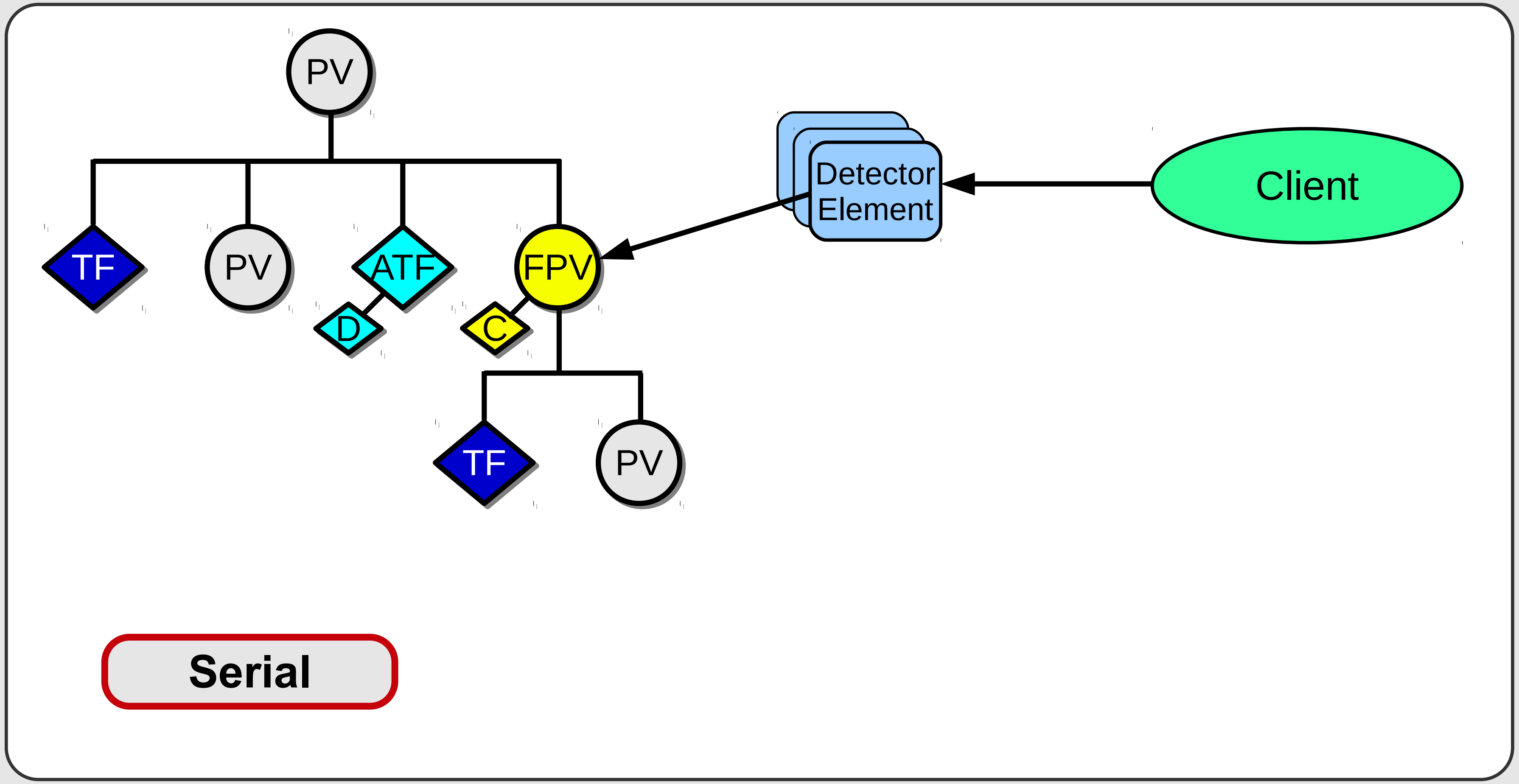
- The current callback functions are **NOT** thread-safe
- Even if they were made thread-safe, could **NOT** run with multiple concurrent Events from different IOVs due to the single ConditionStore
- IOV infrastructure needs to be **significantly** modified for MT

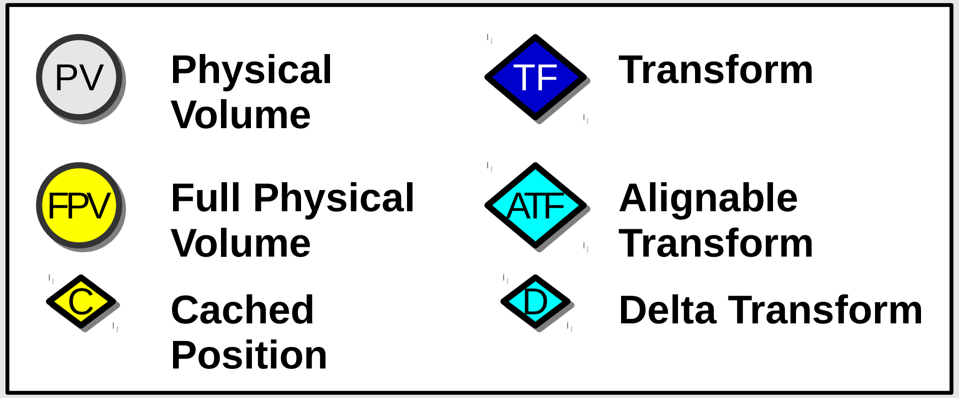
Concurrent: Single Multi-Cache Condition Store



- **One** ConditionStore, shared by all Events.
  - no wasted memory
  - no duplicate calls
  - Store elements are **ConditionContainers**, with one entry per IOV
- Data access via **ConditionHandles** that point to appropriate entry
- Callback Functions become **Algorithms**, scheduled by framework

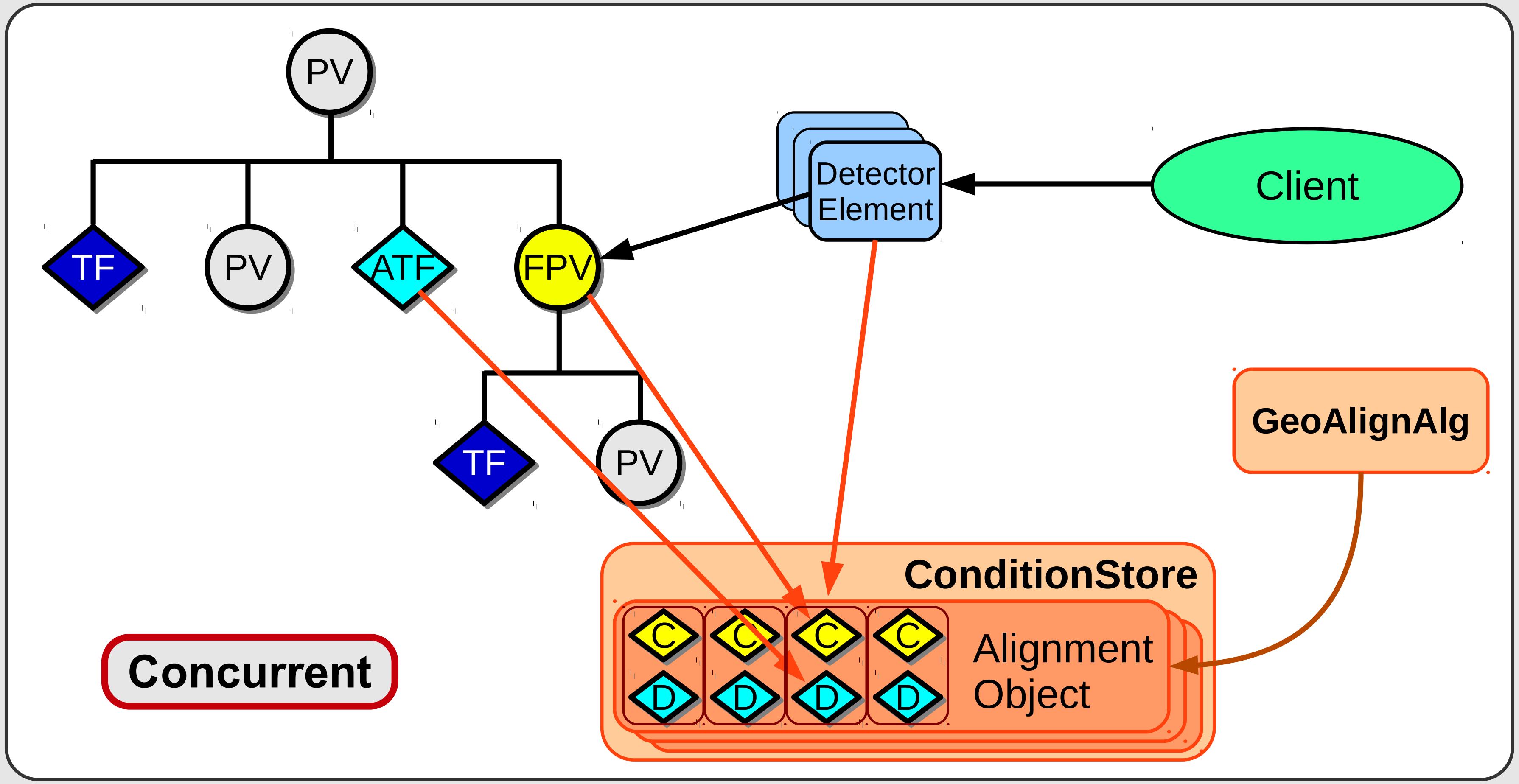
Geometry Alignments in AthenaMT

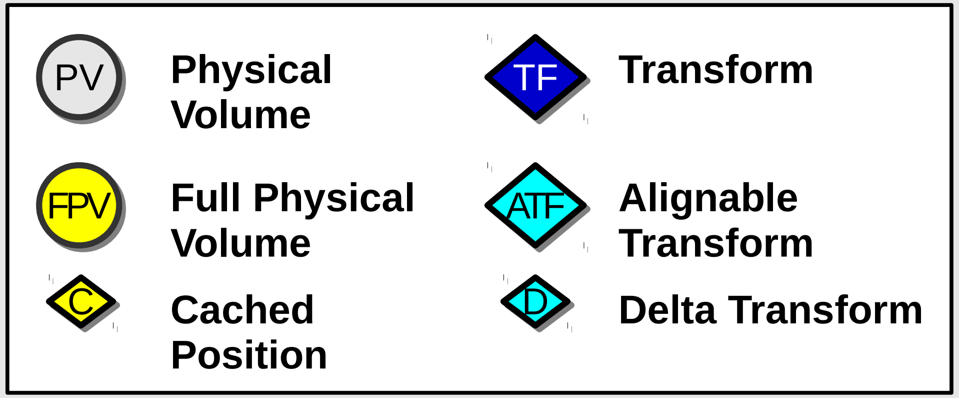




- Detector Element position cached in **Full Physical Volume**
  - built from a **Physical Volume** description, a **Transform**, and a time dependent **Alignable Transform** that reads a **Delta** from a database
- Not functional with concurrent events that have different Deltas and associated caches

Geometry Alignments in AthenaMT

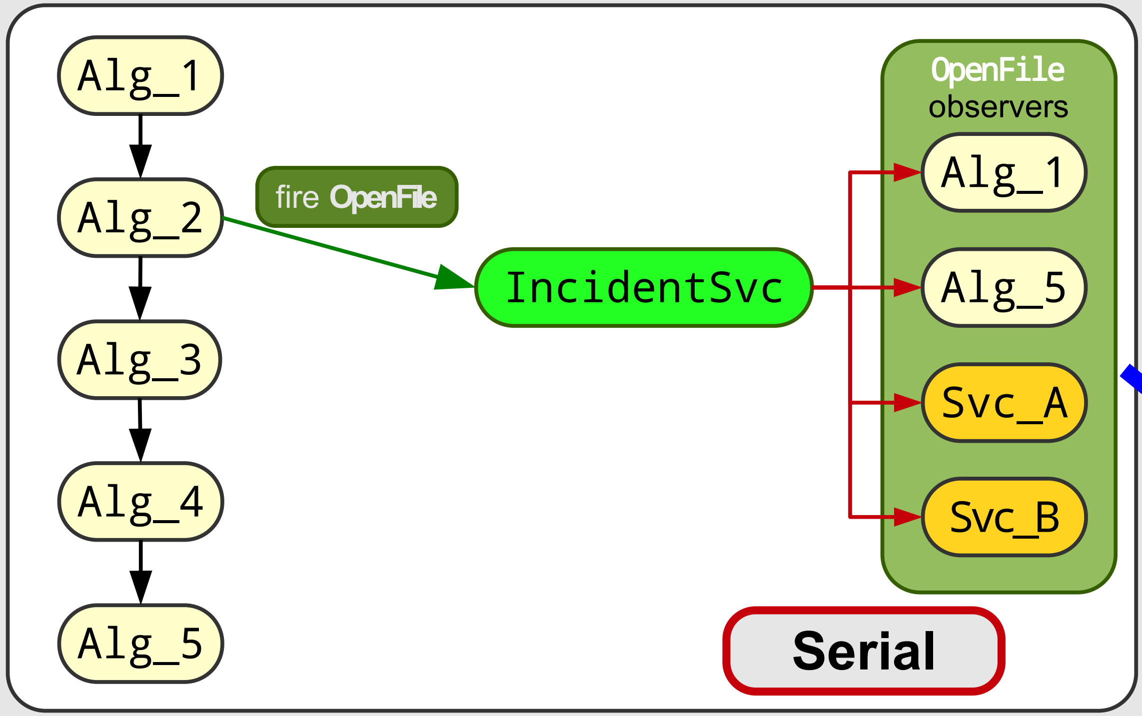




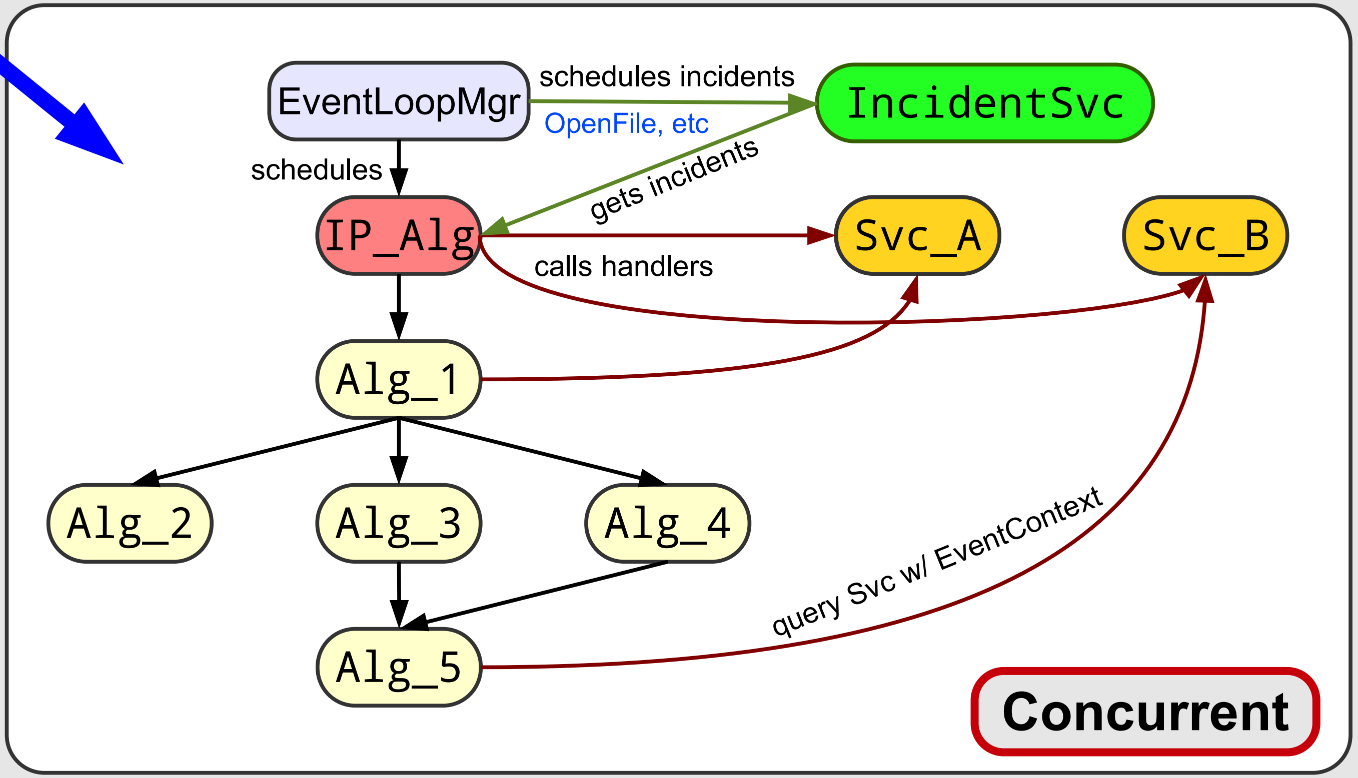
- Encapsulate alignment deltas and cached positions in AlignmentObjects that reside in the ConditionStore
  - accessed via ConditionHandles
  - updated on demand via a scheduled GeoAlginAlg
- Clients of DetectorElements are completely unaware of migration

IncidentSvc in AthenaMT

- **IncidentSvc**: manages asynchronous callbacks for clients using an Observer pattern



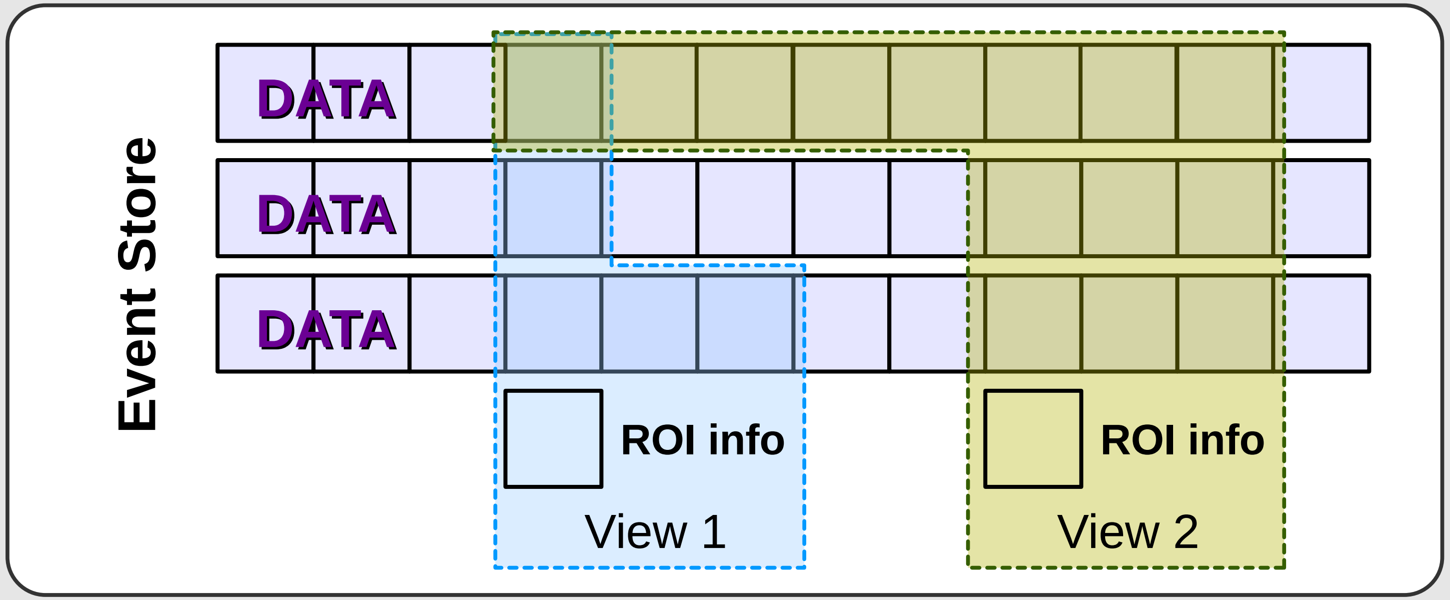
- **Study**: design is far more flexible than actual usage
  - mostly fired **outside** of the Algorithm processing loop



- **Solution**: **limit scope**: Incidents can be re-classified as discrete state changes
  - Incidents become **schedulable**, managed by framework
  - Incident handlers / observers become discrete Algorithms, that interact with EventContext aware Services

Event Views and DataHandles

- For performance reasons, the High Level Trigger operates on geometrical Regions of Interest (ROI)
- Since all Algorithms access Event data via smart DataHandles, they can be run unmodified in a ROI simply by having the framework modify the DataHandle itself
- Implemented an `EventView` class that can be used interchangeably with the whole event store. Each View is populated with data corresponding to a single ROI
  - Each EventView has the same interface as the whole event store
  - Contain Data Objects that describe the corresponding ROI
  - Allow for potential alternative use-cases



see presentation by Ben Wynne on Tuesday at 2PM for further details

Re-Entrant Algorithms

- Cloning of Algorithms in GaudiHive allows us to avoid most thread safety issues
  - clones can run concurrently with different Event Contexts without interference
  - have to avoid "thread hostile" behaviour
    - global statics
    - back channel communications
  - some Algorithms can't be cloned – scheduling bottleneck

- Downside of many clones is increased memory use
  - can limit number of clones, at the expense of limiting possible concurrency

- Added support for re-entrant Algorithms
  - only one instance
  - can be executed simultaneously in multiple threads in different Events
  - **MUST** be thread safe
    - enforced with new base class and `Algorithm::execute_r() const` signature
  - envision limited usage for special purpose tasks, written by experts

Summary

- ATLAS has begun the migration of core framework elements that require the most significant design changes beyond mere thread safety
  - sometimes by completely re-evaluating functionality and limiting design to actual use cases
- We have made design choices that minimized alterations to client code
  - leverage existing features of framework, *eg* DataHandles and the Scheduler
- Changes to Algorithmic client code that use these elements are also underway
  - relatively straight forward recipe for the most part (but a lot of grunt work)
- Anticipate on-schedule finalization of design, and implementation of essential core Services by end of 2016, with full support of MT concurrency by end of 2017
  - we already have production level Atlas G4 simulation running in MT on KNL.

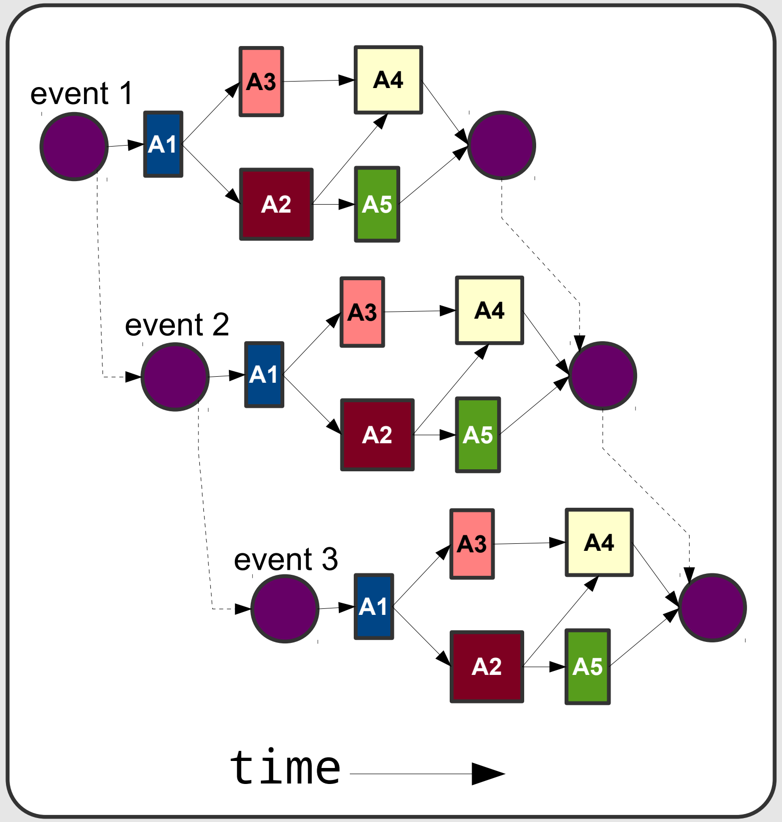
see Steve Farrell's talk Thursday, track 2, 2PM

- Broad migration of Algorithm code to use these features will begin in 2017

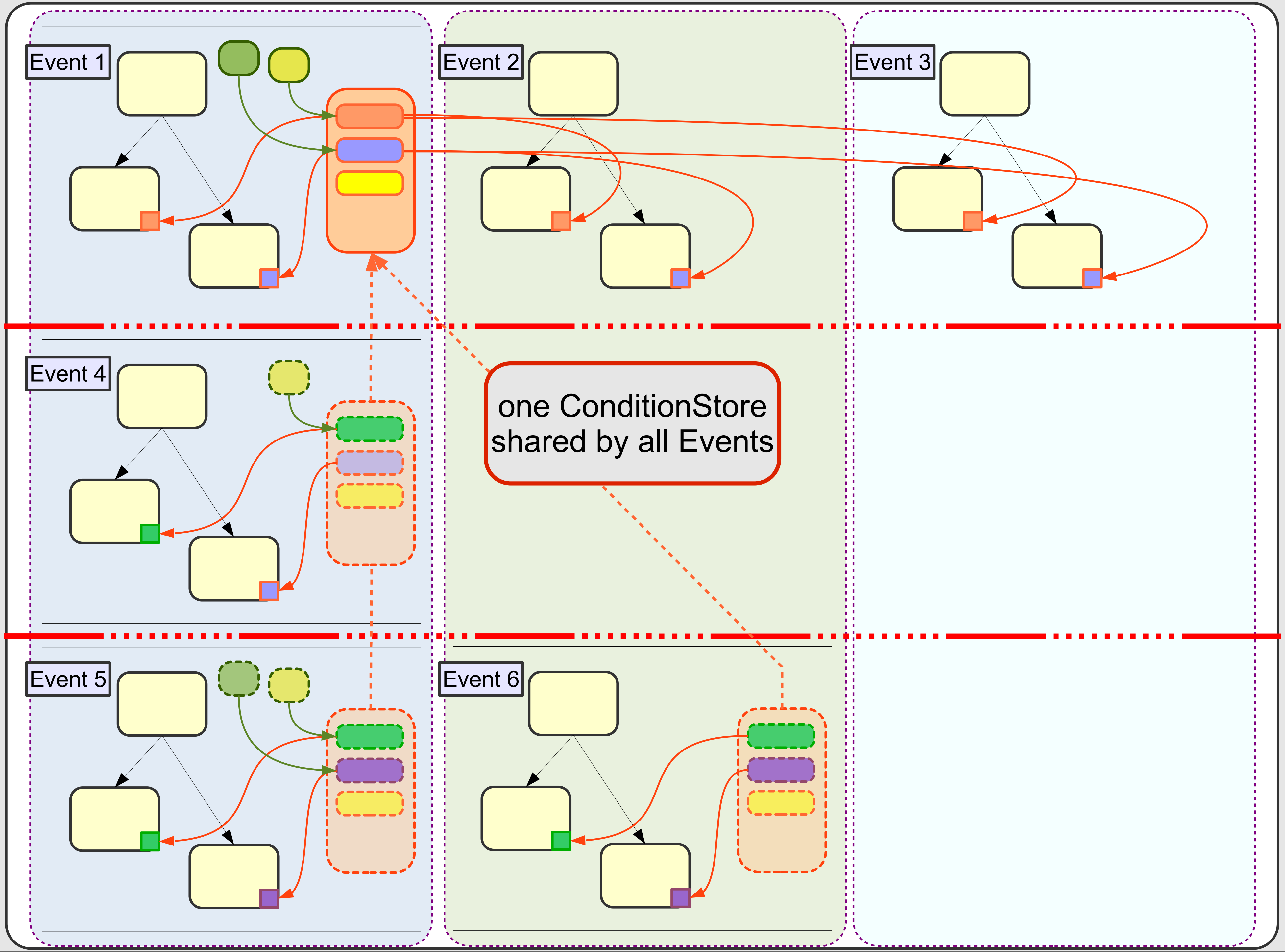
Extras

AthenaMT / Gaudi Hive

- AthenaMT: based on Gaudi Hive: multi-threaded, concurrent **extension** to Gaudi
- Data Flow driven
  - Algorithms declare their data dependencies
  - Scheduler automatically executes Algorithms as data becomes available.
    - optimal traversal of graph possible if avg. Algorithm runtimes known
- Multi-threaded
  - Algorithms process events in their own thread, from a shared Thread Pool.
- Pipelining: multiple algorithms and events can be executed concurrently
  - some Algorithms are long, and produce data that many others need (eg track fitting). instead of waiting for it to finish, and idling processor, start a new event.
- Algorithm Cloning
  - multiple instances of the same Algorithm *may* exist, and be executed concurrently, each with different Event Context.
    - **legacy** : one instance, non-concurrent
    - **cloneable** : one or more instances, in its own thread
    - **re-entrant** : once instance, executed concurrently by multiple threads
- Thread Safety
  - Only shared Services and re-entrant Algorithms need to be thread safe
  - Algorithms must avoid thread-hostile behaviour
    - global statics, etc



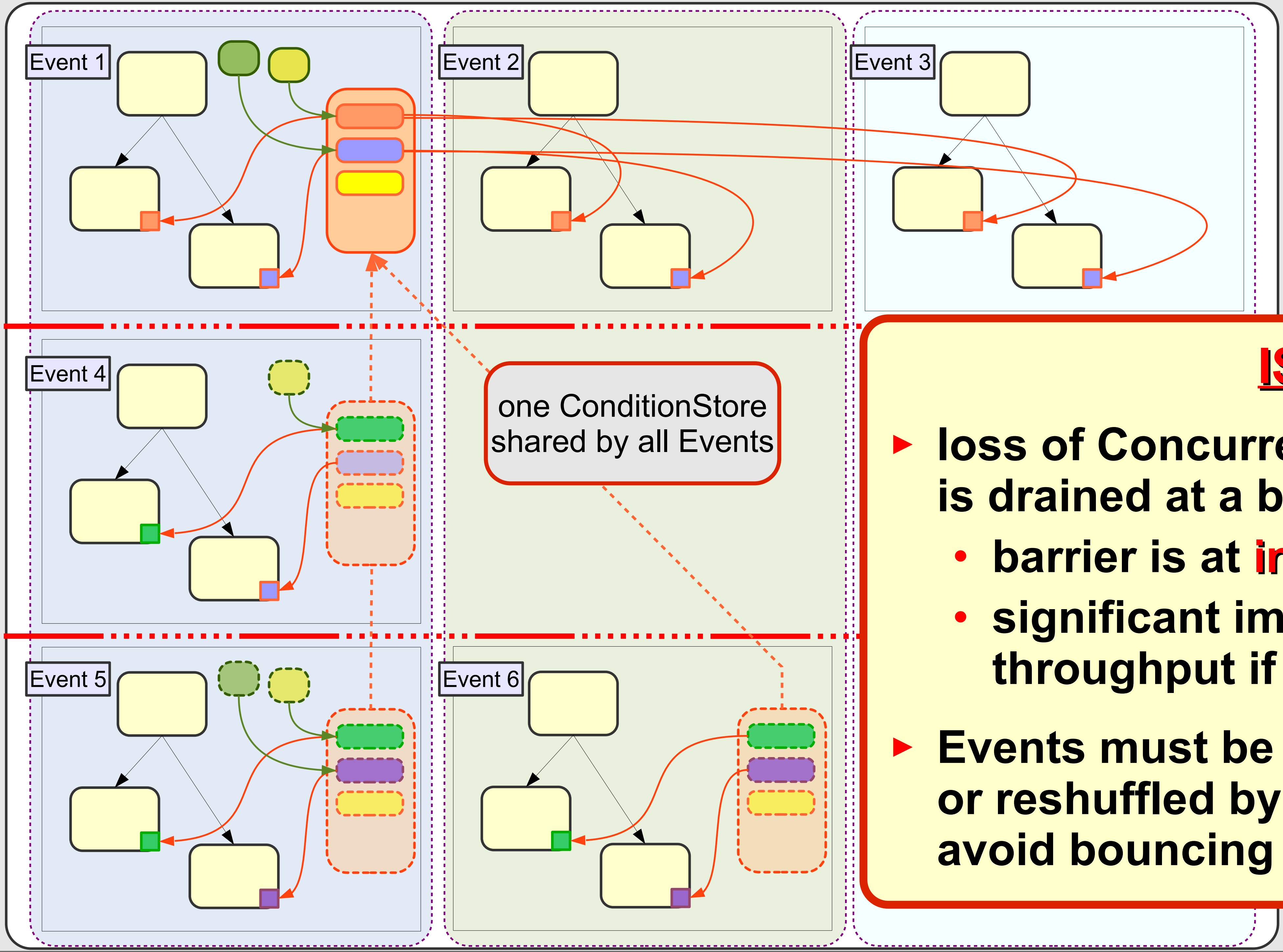
Concurrent: Scheduling Barrier



Scheduler can only **concurrently** process events which have **all** Conditions in the **same** IOV

NO changes required in User code and minimal changes in IOV code

Concurrent: Scheduling Barrier



Scheduler can only **concurrently** process events which have **all** Conditions in the **same** IOV

ISSUES

- **loss of Concurrency when Scheduler is drained at a barrier**
  - **barrier is at intersection of all IOVs**
  - **significant impact on Event throughput if IOVs change often**
- **Events must be processed in order, or reshuffled by the Scheduler to avoid bouncing back and forth**

ConditionHandles