# Implementation Highlights

Mike Miller

Yale University

# Outline

- Large Scale Project, expanding scope
  - 106 header files, 96 source files
  - 25k lines of code
  - more than 110 classes
- Focus! (4 topics)
  - Object creation, ownership
  - Object communication
  - Hit/Material pattern
  - Pattern Recognition

# Creation/Ownership: Why the Fuss?

**Speed, Speed, Speed!**

- Memory allocation takes time
- O(1000) tracks/event
- O(100,000) hits/event
- But, can use transient objects

Allocate objects once, reuses

Flexibility

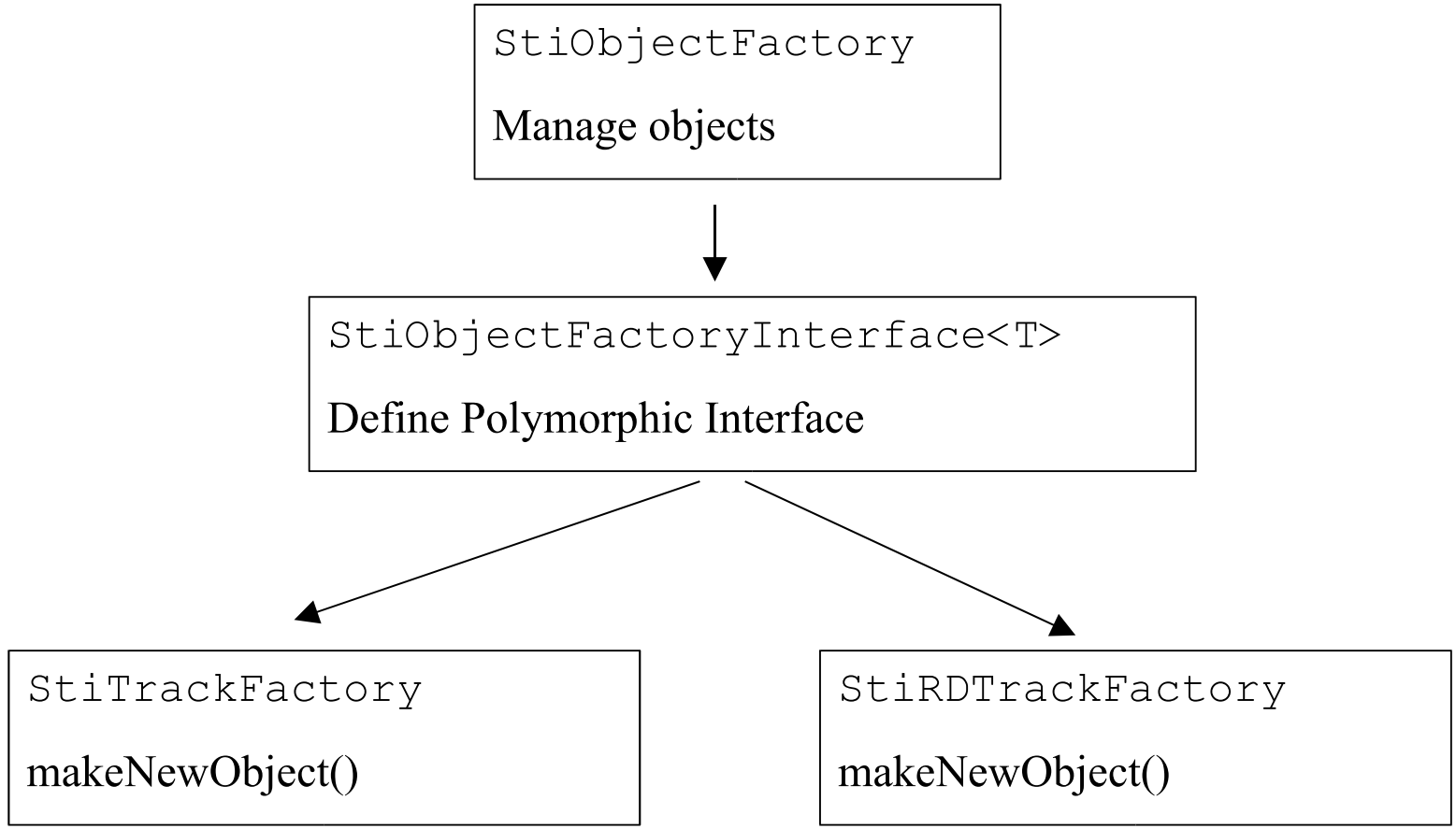
- Dynamic control type of objects to make
- But, code must be maintainable, readable (minimize conditionals, e.g. bfc.C)

Use factory pattern to make, serve objects

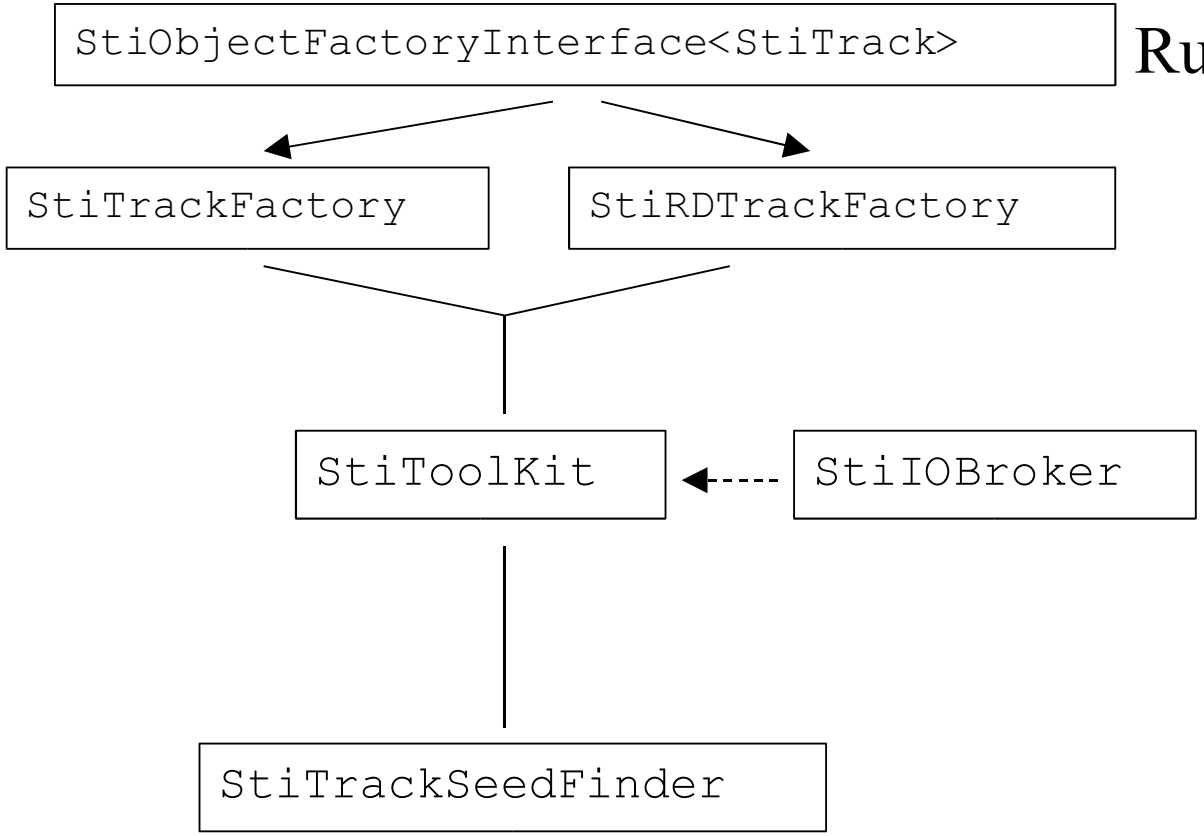
# Creation/Ownership: Two Design Patterns in One

- Memory Pool
  - Allocate memory in large chunks (pool)
  - Third party is responsible for rationing pool
- ITTF actually re-uses objects each event
  - No calls to new or delete!
- Factory
  - Instead of calling new/delete for objects, request objects from factory
  - Factory decides what objects to make
  - User code (e.g., tracker) is independent of actual object type

# Creation/Ownership: Class Design



# Creation/Ownership: An Example




Run in GUI?

- StiToolKit makes the right factory
- Seed Finder gets a pointer to the factory interface
- Seed Finder requests new tracks from the factory

**Leak free, type safe, fast!**

# Communication: StiIOBroker

Goal: centralize all dynamic parameters in one place

`class StiIOBroker`

Abstract class to define get/set methods

Singleton

`class StiRootIOBroker : public StiIOBroker`

Available at the root prompt

Currently used to pass info at the macro level

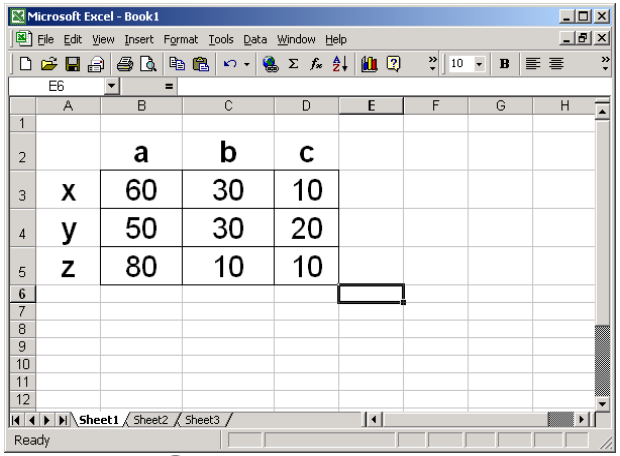
`class StiMySqlIOBroker : public StiIOBroker`

Not yet implemented

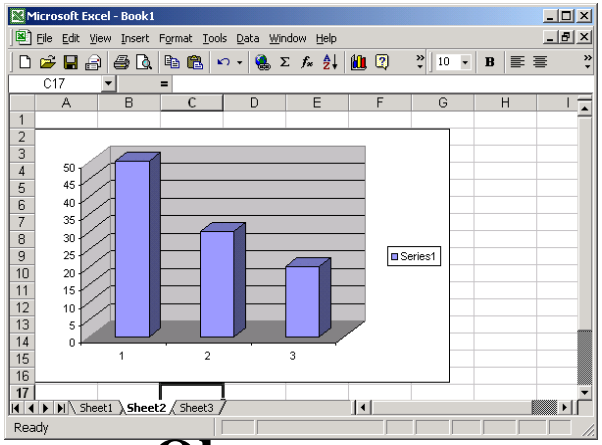
# Communication: Subject/Observer Pattern (I)

Define a one to many relationship

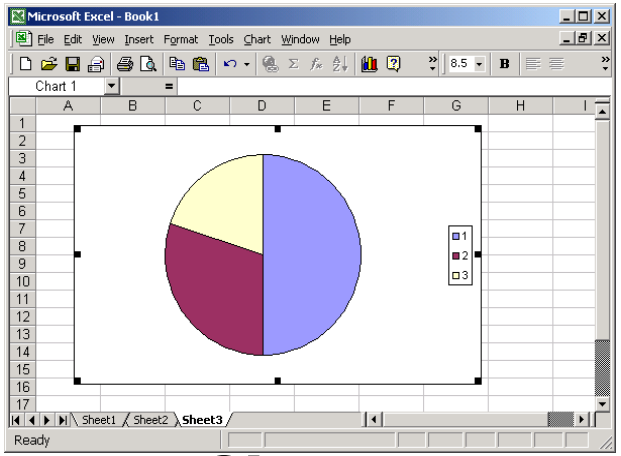
Guarantee dynamic propagation of information



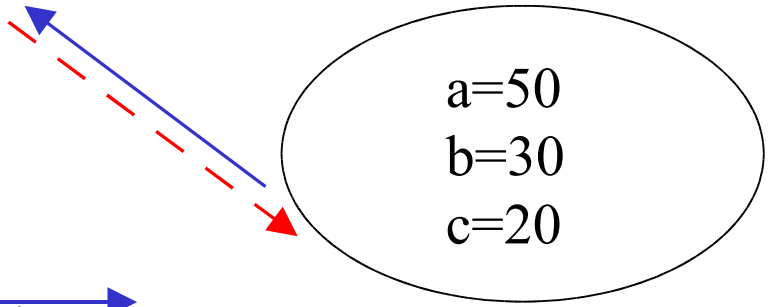

Observer




Observer




Observer




Subject

notification

requests

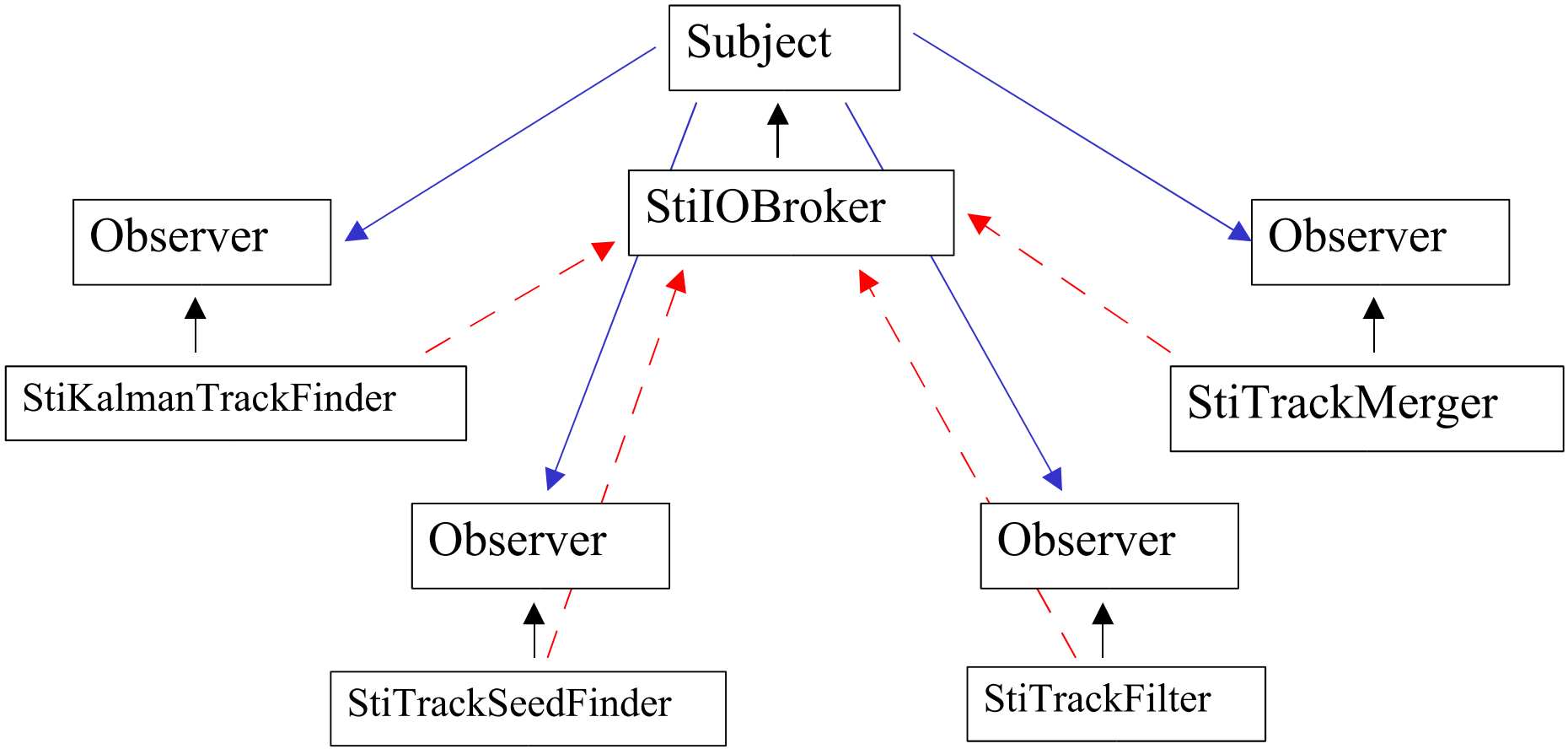
# Communication: Subject/Observer Pattern (II)

```
class Subscriber {
public:
   add(Observer*);
   detach(Observer*);
   notify();
private:
   vector<Observer*> mVec;
};
```

```
class Observer {
public:
   update(Subject*)=0;
private:
   Subject* mSubject;
};
```

Simply derive from these classes and dynamic updates are guaranteed!

# Communication: Example



# Hits/Material: Why Separate?

- Hits come from detectors, dont they?
- Charge: Plug and Play
  - Decouple patter recognition algorithms (track finding) from track fitting algorithms (kalman)
  - But, be quick about it!
- Design:
  - Decouple: StiHitContainer, StiDetectorContainer
  - Preserve simple, fast, natural mapping

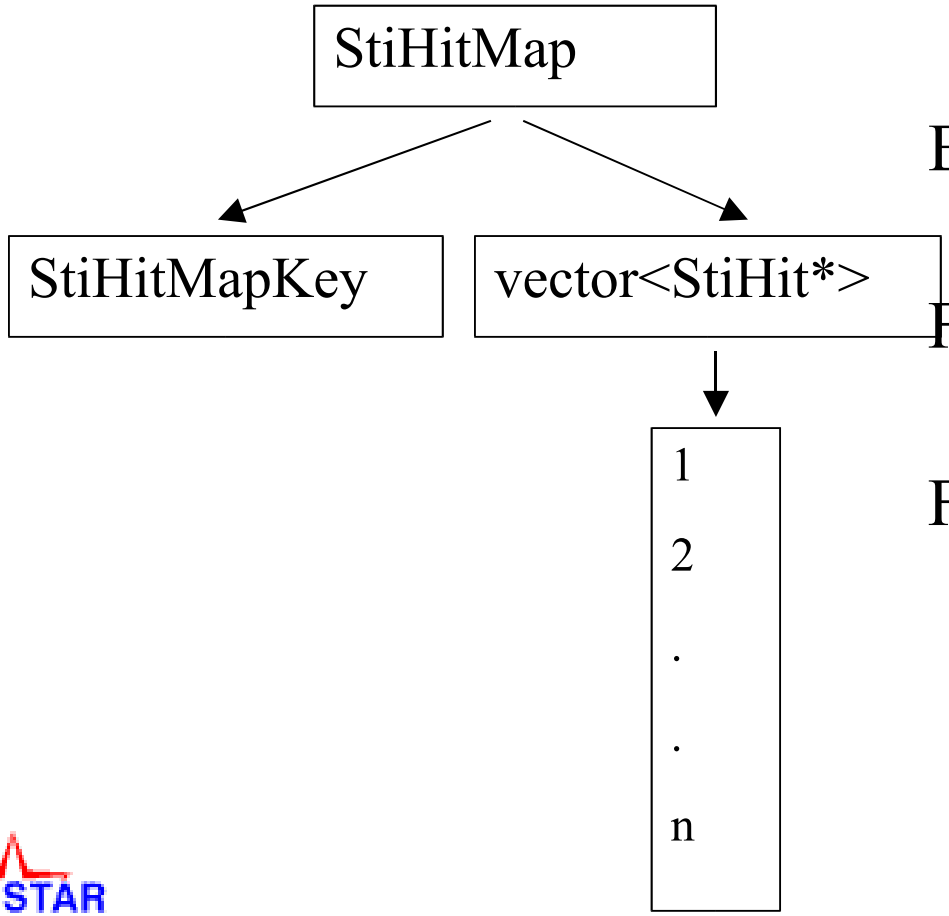
# StiHitContainer: Requirements

- Treat hits from all subsystems equally
  - Identify minimal information for common representation
- Extremely fast
  - fill, sort, retrieve O(500k) hits/event in <5cpu sec
  - Return hits in some window about reference.
- Applicable to all pattern recognition algorithms
  - Must function in local **and** global representations

# StiHitContainer: Intro to STL

- Standard Template Library
  - Sequential, associative containers
  - Many classes of iterators
  - Algorithms!
- Consequences
  - No c arrays (no out of bounds run errors!)
  - Decouple algorithms from containers
  - Never right another algorithm:
    - Sort, find, search, count, transform, etc
- Many numerical algorithms -> STL usage

# StiHitContainer: Storage

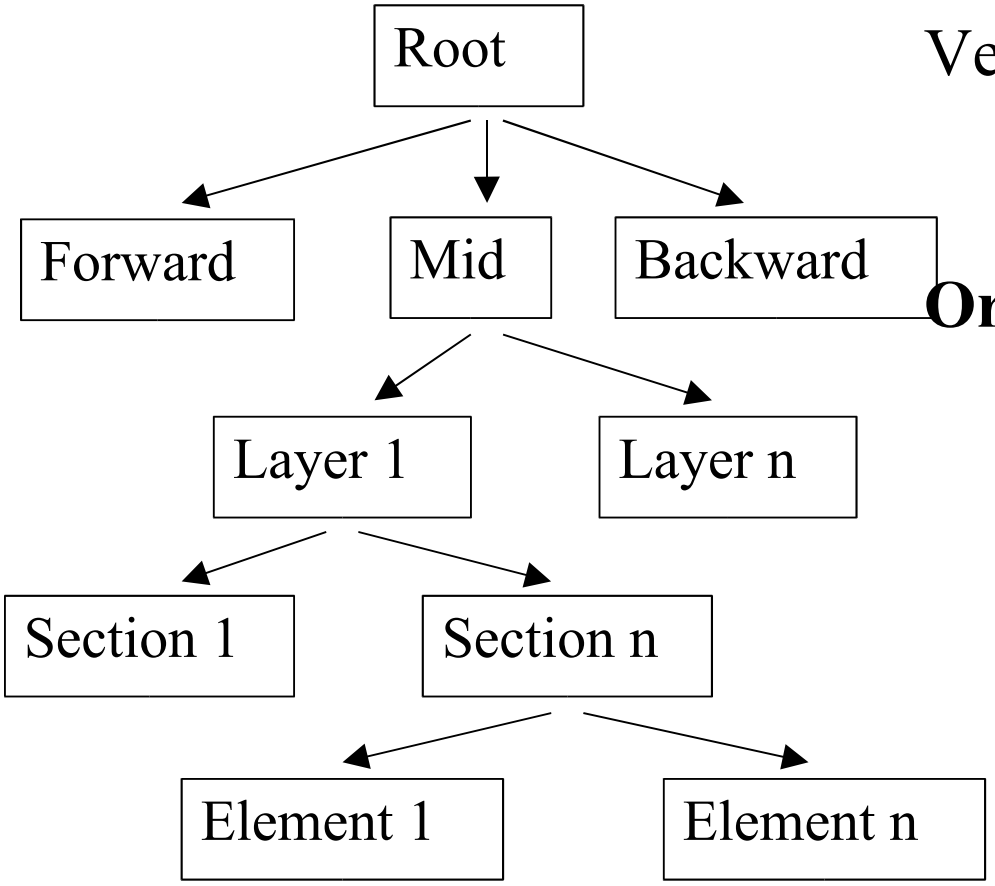



- Use STL map/vector to organize hits
- Establish one to many relationships
- Fast retrieval of layer via key
- Fast retrieval of subset of hits in layer

# StiHitContainer: Retrieval

- Set reference point
  - Predicted intersection of track + plane
  - Time: O(log(M))
    - M~number of detectors (~45*12)
    - Key into hit map
- Binary search in global z-window
  - Hits sorted by z in detector plane
  - Time: O(log(N))
    - N~number of hits in plane (~100)
- Linear search in distance along (pad) plane
  - Time: O(P)
    - P~number of hits in z window (<10)

# StiDetectorContainer: Design




Version 5.x

- tried multimap, lattice, polygons, volumes, tree

**Ordered** composite tree structure

- Fast
- Flexible
- StiCompositeTreeNode<T>

# StiDetectorContainer: Navigation

Set to element

- Defined by region, position, angle

moveIn(), moveOut(), movePlusPhi(), moveMinusPhi()

Example: moveIn()

- Remember angle
- If next layer preserves symmetry, constant time
- If next layer has different symmetry, O(log(N))

**10k swims from padrow 45-vertex in 0.17 cpu sec**

# StiSeedFinder: Requirements

Responsible for pattern recognition

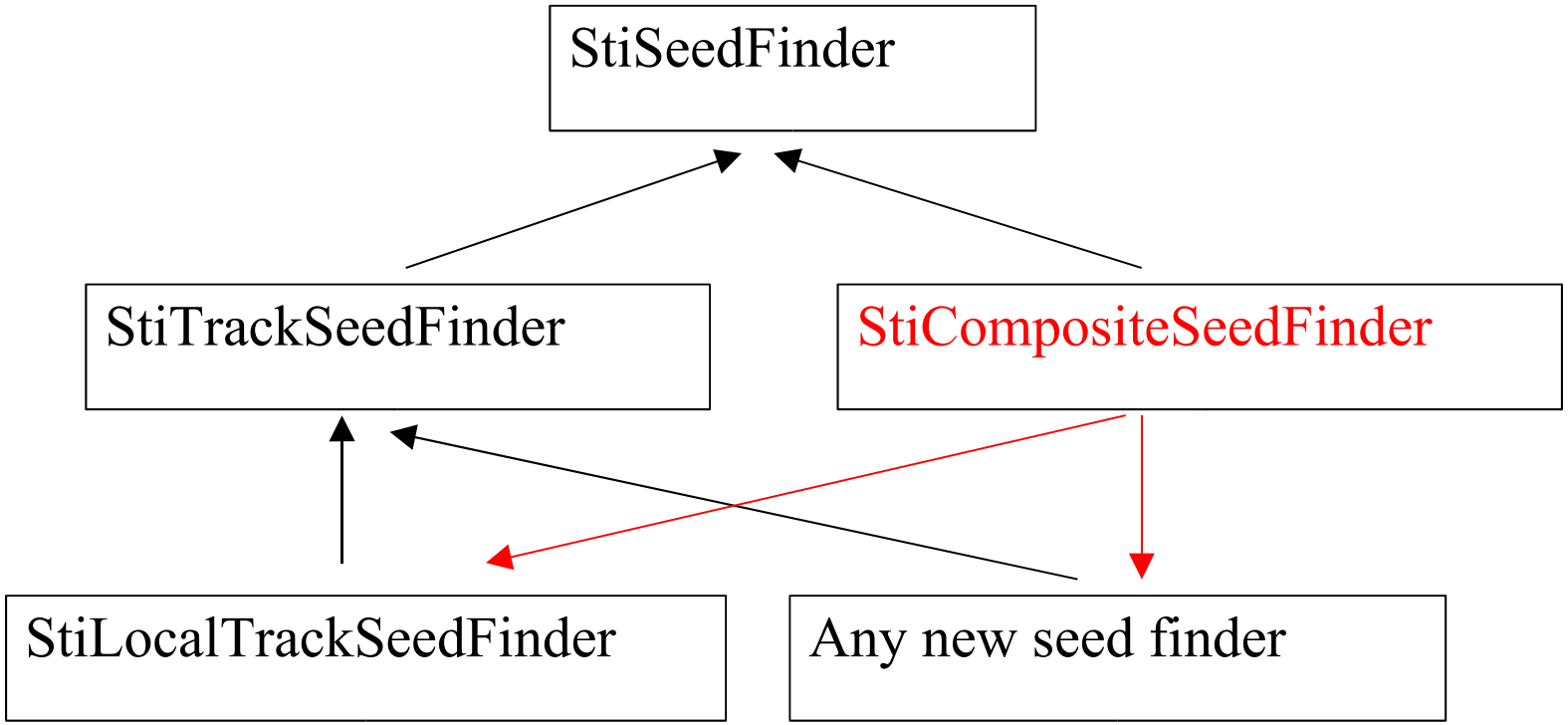
Plug and Play

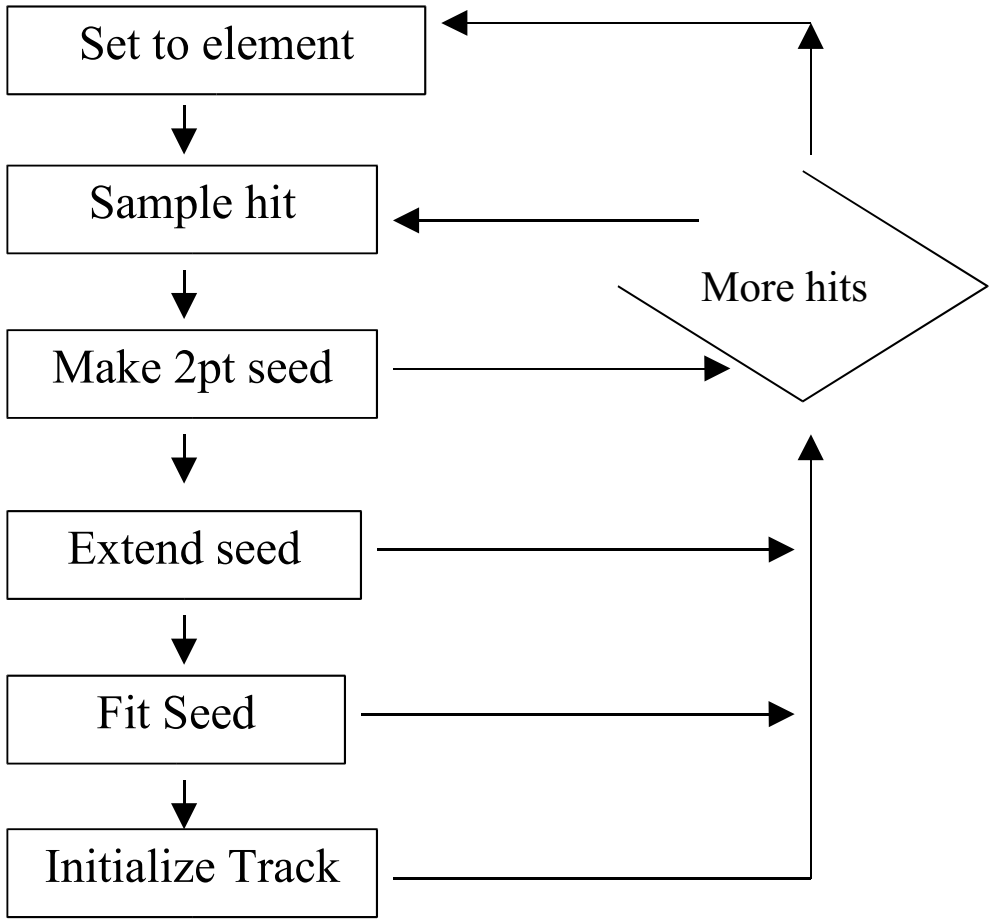
- easily implement new pattern recognition algorithms
- Multiple algorithms in series and/or parallel

Fast

- Less than 10 cpu sec per central event

# StiSeedFinder: Hierarchy



# StiLocalTrackSeedFinder: Algorithm

- Implement road-finder
- Begin tracking at outside of tpc
- Spiral inwards
- Extend tracks into innermost layer
- T.B.D.
  - Second pass (e.g., merge spirals)