Event Collections

- Issue: Event Data is spread across several files
  - Digi, Recon, MC, Merit, Svac, Cal Tuple, GCR Tuple
  - Some have simple NTuples, others have ROOT objects
- Problems with this.
  - Synchronicity: need to make sure that files stay in sync
    - Requires ad-hoc solutions certain cases
      - Writing empty events into some files
    - Makes sparse collections impractical
      - Need to deep-copy all the data you want
  - File management:
    - Different files made by different tasks
      - Might not be stored in the same place
    - Pain for users, need to keep track of several files

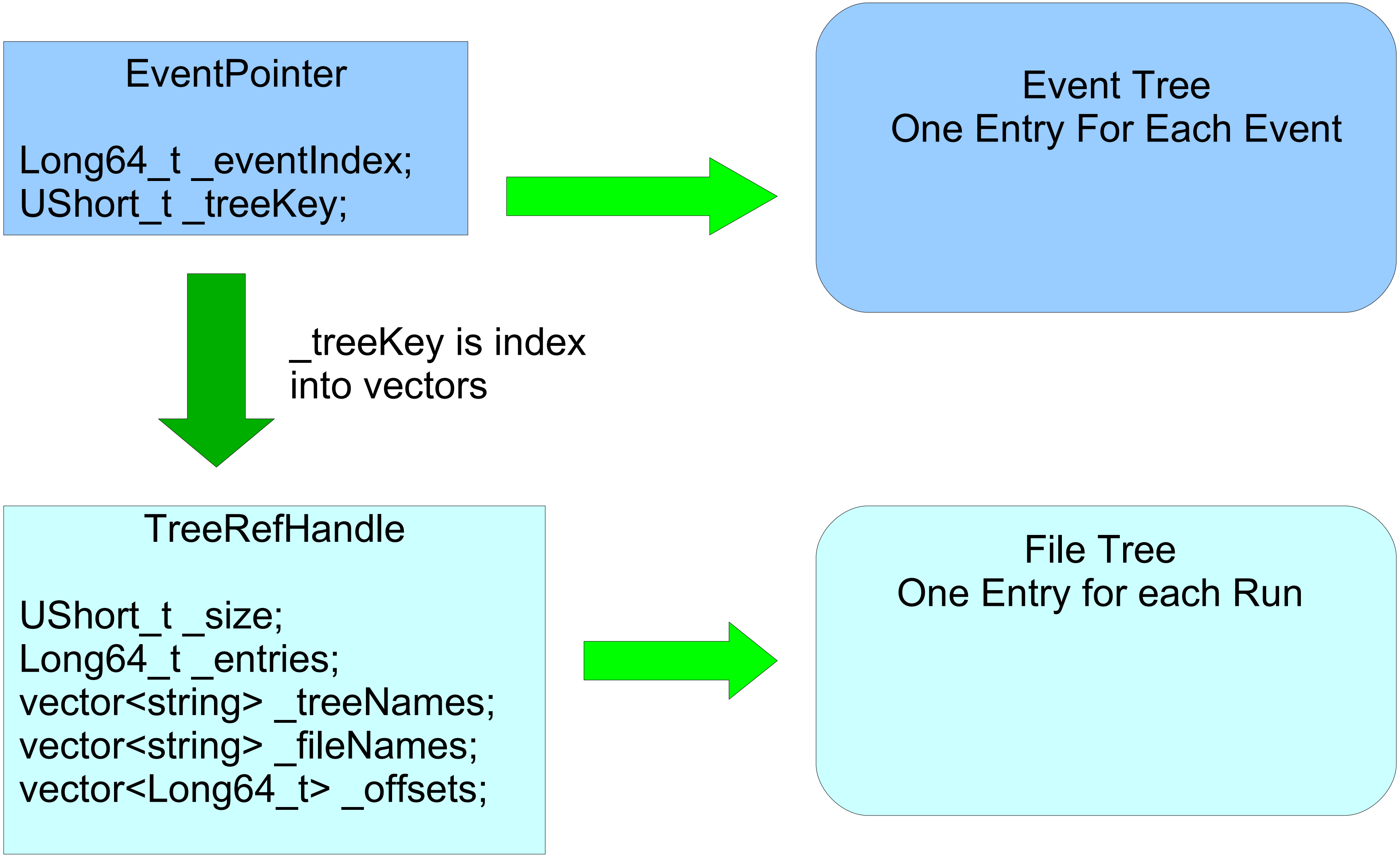
Panacea

- Build system to collect parts of events into groups
- Basically "pointer" or "meta-data" collection
  - Store a TTree with just enough information to find the various parts of the event
- Define event components
  - A component is one entry in a TTree that contains data for a given event
  - Build a event by having a bunch of pointers to components

Event Component Pointer

- Minimum data to find an event component
  - Which file
  - Which tree in file
  - Which entry in tree
- How to specify that information
  - In ROOT Entry is just Long64_t {64 bit signed integer)
  - File/ Tree is more complicated
    - Easiest is strings for FileName, TreeName
  - Many events from same file/tree
    - Keep file/tree names in separate TTree with only a few entries

Design For One Component



EventPointer

```
Long64_t _eventIndex;
UShort_t _treeKey;
```

_treeKey is index
into vectors

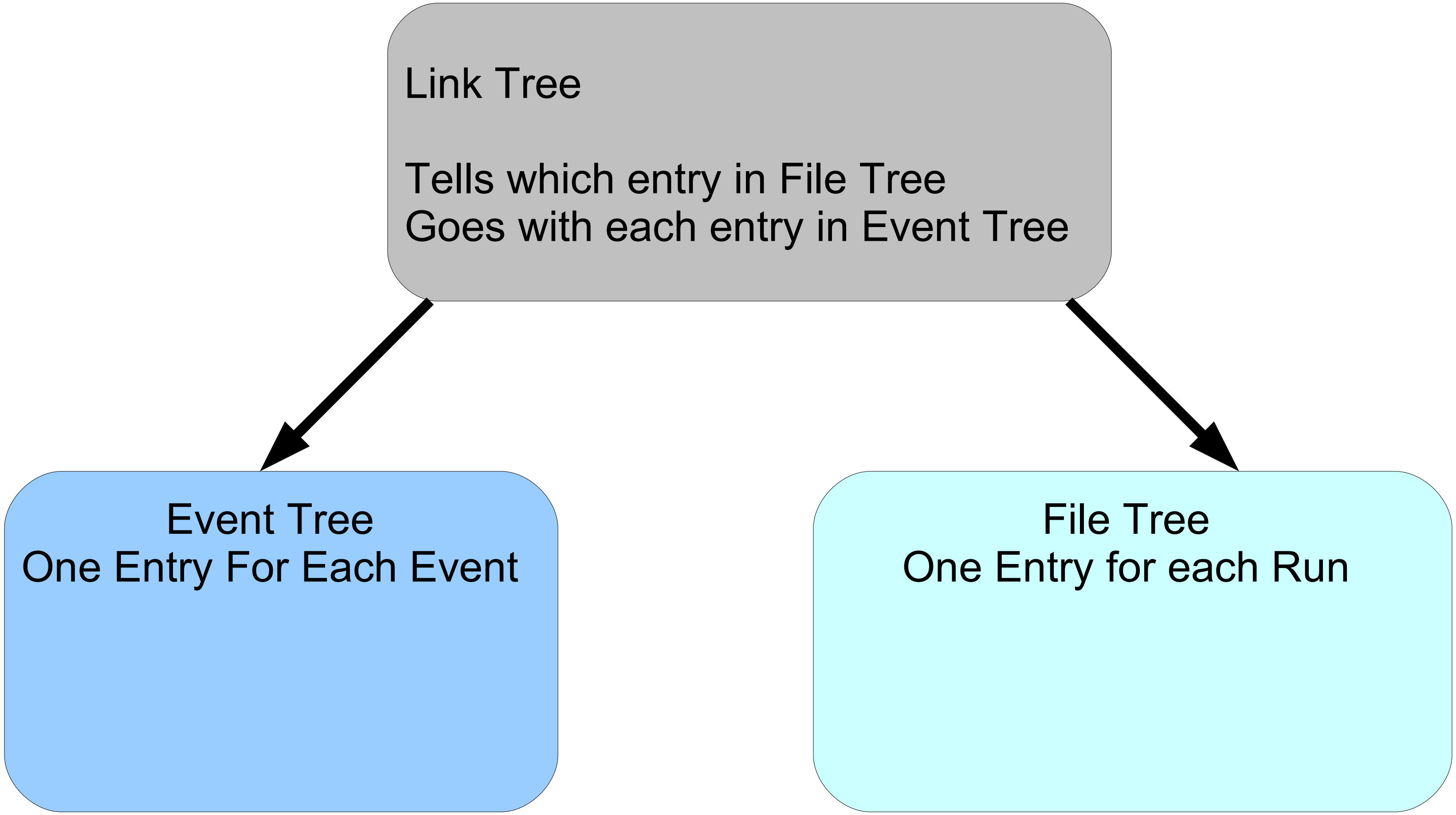
Event Tree
One Entry For Each Event

TreeRefHandle

```
UShort_t _size;
Long64_t _entries;
vector<string> _treeNames;
vector<string> _fileNames;
vector<Long64_t> _offsets;
```

File Tree
One Entry for each Run

Overall Design



Link Tree

Tells which entry in File Tree
Goes with each entry in Event Tree

Event Tree
One Entry For Each Event

File Tree
One Entry for each Run

When merging collection, Event Tree and File Tree stand alone and can be copied with uncompressing.
Only Link Tree needs to be modified.

Some Bonuses

- Sparse Collections
  - Can make collections containing only events that pass certain cuts.
    - Useful for calibrations
    - Avoids data duplication, actual data off in XROOTD, only have pointers to the event components
      - Can make ‘deep-copies’ as needed when you want to transfer data to outside sources
- Replaceable components
  - When re-running part of processing just generate new index files that point to new version of processed data
    - Point users at the new index files
      - Less headache for users

Working Example

- This has all been implemented for GLAST
- Functionality discussed here is in metaRooData
- Main Classes is PointerSkim
  - Long64_t PointerSkim::fillEvent(vector<TTree*>& trees)
    - Stores one event, gets file names and tree entries numbers
    - Returns number of bytes written, on negative # for error
  - Long64_t PointerSkim::fillMeta()
    - Called at end of run, fills entry in file tree
    - Returns number of bytes written, on negative # for error
  - TChain* PointerSkim::buildChain(TObjArray* chainList)
    - Builds and return a TChain with all the events
    - Also builds TChains for each component
    - Uses PointerIndex (sub-class of TvirtualIndex) to point to events in compontent Chains

•File Handling

- All file handling broken out into FileUtil.h (cxx)
  - This allows us to get fancy with file names
    - Logical File names
    - Relative File names
    - Sticking stuff behind XROOTD
  - TFile* FileUtil::openFile(const char* fileName);
    - Opens a file given LFN
  - TFile* getFile(TTree& tree)
    - Get LFN give a TTree