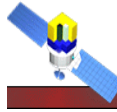
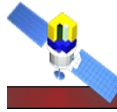
Event Collections



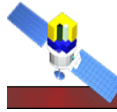
- Issue: Event Data is spread across several files
  - Digi, Recon, MC, Merit, Svac, Cal Tuple, GCR Tuple
  - Some have simple NTuples, others have ROOT objects
- Problems with this.
  - Synchronicity: need to make sure that files stay in sync
    - Requires ad-hoc solutions certain cases
      - Writing 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



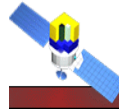
- Build system to collect parts of events into groups
- Basically "pointer" or "meta-data" collection
  - Store a TTree with just enough information to find the various parts of the event
- Define event components
  - A component is one entry in a TTree that contains data for a given event
  - Build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

Event Component Pointer



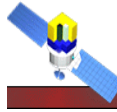
- Minimum data to find an event component
  - Which file
  - Which tree in file
  - Which entry in tree
- How to specify that information
  - In ROOT Entry is just Long64_t {64 bit signed integer)
  - File/ Tree is more complicated
    - Easiest is strings for FileName, TreeName
    - Several Improvements on this possible

Specifying File/ Tree



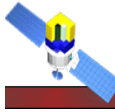
- Improvement: use external table for File/Tree names
  - Entry in n-tuple column is just index into table
    - Maybe Int_t -> 32 bit signed integer or even smaller
  - Entries in table are more complicated and flexible
    - Physical File Name/ Tree Name
    - Logical File Name (in XROOTD) / Tree Name
    - Relative File Name
      - Assumes that all files in same area, allows them to be moved as a group

Reading Collections



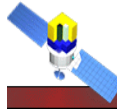
- Easiest solution: provide a class to open/ access the component trees
  - reader->getEvent(Long64_t iEvt, vector<TTree*>& comps);
    - Reads the correct events from the component trees, puts them onto the vector "comps"
  - Can add some control of what does/ doesn't get read
    - reader->readBranch(const char* tree, const char* branch)
- Fancier solution: make sub-classes of ROOT stuff to handle our trees
  - GTree : public TTree
  - GTreeReader : public TTreeReader
    - Might only need one, not both
    - GTree->LoadTree(Long64_t) -> also loads component trees
    - Use FriendTree stuff in ROOT to allow plotting data from different components against each other

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 BaBar
  - Actually, BaBar model much more complicated
- Functionality discussed here is in:
  - KanEvent/KanHeaderTree
    - Inherits from TTree
    - Overrides TTree::Fill() and TTree::LoadTree()
    - Basic representation of Event component is
      - pair<TTree*,TFile*> FileAndTree
      - Int_t index