CMS PFlow Present Design

Lindsey Gray
13 May, 2014

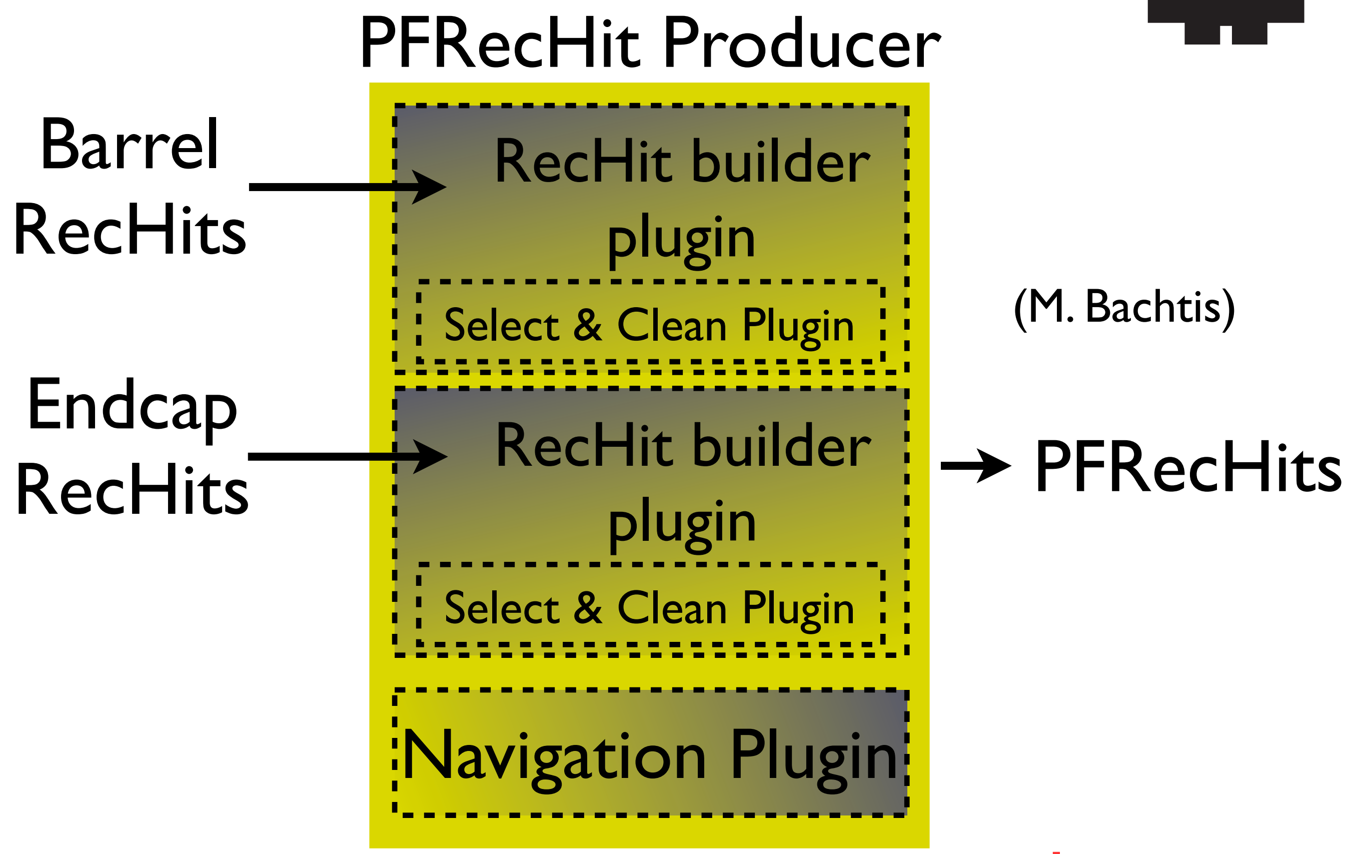
on behalf of the CMS Cross-POG Forum

Quick CMS PFlow Overview

- Developed in-house over a number of years
  - Started out highly specialized for CMS and very ‘concretely’ implemented for such a task
  - Optimized for working with low to ‘high-ish’ granularity calorimeters
    - Simple re-clustering based on calorimeter vs. track resolution
    - Originally designed around only 2D calorimeters
  - Specialized around working with a dense tracker
    - Specialized electron and photon reconstructions to deal with lossy tracks and high conversion rate
    - Nuclear interaction finding, track refitting, a number of other specializations
- Huge amount of customization and experience from data taking and high in-time PU available (and immediate interface to CMS Geometry)
  - Needed to be made more flexible to deal with upgrade possibilities and improved clustering techniques or more information
  - For detector-specific clustering and topological linking we have done this
    - Reclustering and particle ID parts presently under re-design
  - Very curious to figure out how to correctly exploit Pandora’s algorithms in CMS given this redesign. Likewise lessons from CMS should be more conducive to porting or use in some other framework.

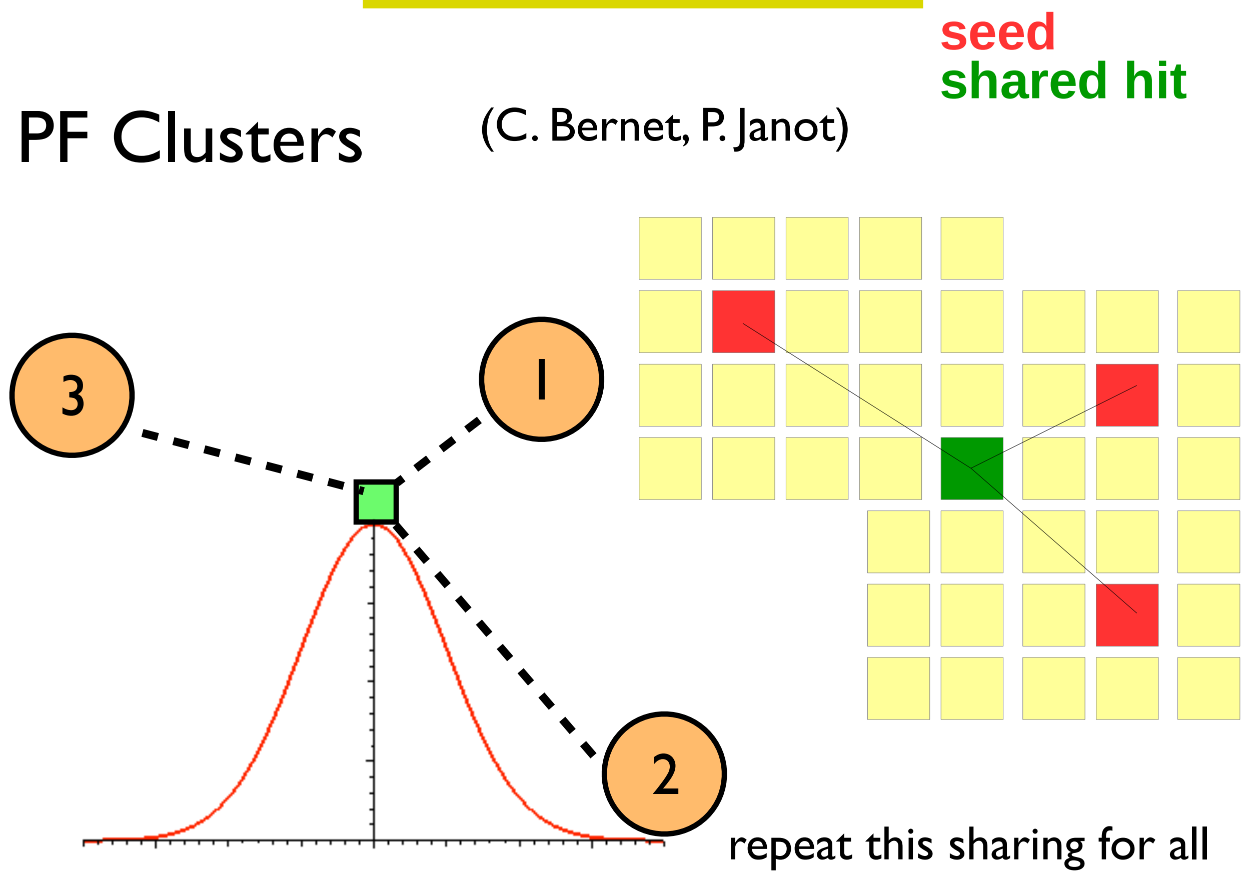
PFRechits & Clusters

- PFRecHits are basic rechits (energy, time,flags) with embedded geometry and topology
  - Allows PFRecHit to be independent of underlying detector
  - Topological associations only need to know nearest neighbors at each active calorimeter cell
  - With this in hand clustering becomes an abstract problem
- PFClusters are built by a generic algorithm for 2D calorimeters that recovers shower substructure through energy sharing
  - Expose shower substructure by allowing any local maximum in energy to be a cluster seed
  - Use a gaussian profile to then share the rechits between the seeds
  - Recalculate position with addition energy from shared rechits, iterate until cluster positions are stable



(M. Bachtis)

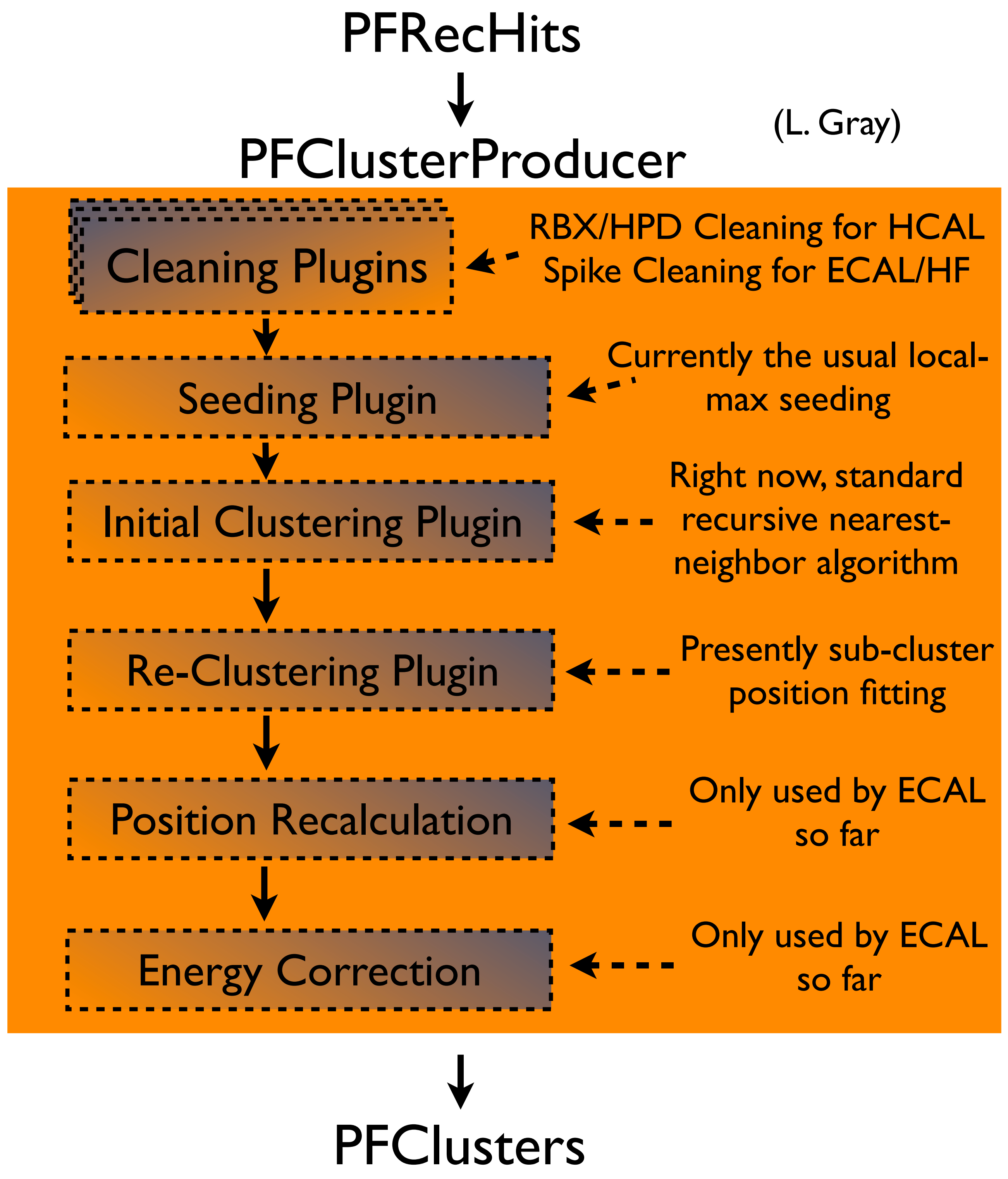
PF Clusters (C. Bernet, P. Janot)



repeat this sharing for all cells in topo cluster (yellow)

PFClusters Design

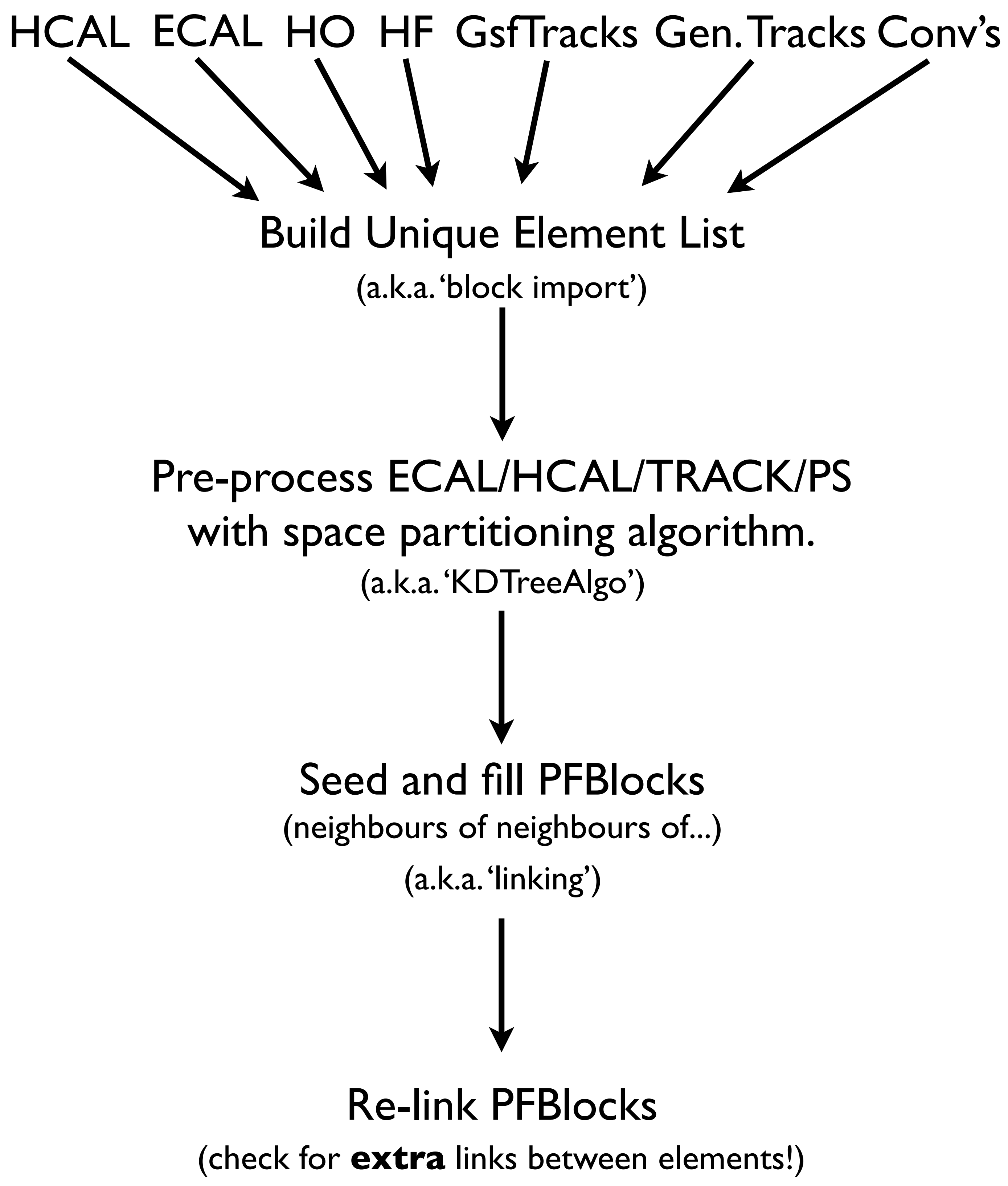
- For Run One, PFRechitProducer and PFClusterProducer were very monolithic
  - Only one detector to cluster
  - Now we have three detectors to deal with!
    - and radically different calorimeters to cluster in some scenarios
- In order to be able to adapt to the future it was necessary to reorganize these valuable tools
  - Create abstractions to remove underlying detector configuration dependencies
  - Clustering methodologies, rechit cleaning, practically everything except for the gross anatomy of the algorithms are now modularized and easily replaceable with minimal overhead
  - New calorimeters are easy to add in this reworked framework for particle flow
  - Allows developers to focus on bigger picture while still maintaining fine grained control!



PFRecHits
PFClusterProducer
(L. Gray)
Cleaning Plugins
RBX/HPD Cleaning for HCAL
Spike Cleaning for ECAL/HF
Seeding Plugin
Currently the usual local-max seeding
Initial Clustering Plugin
Right now, standard recursive nearest-neighbor algorithm
Re-Clustering Plugin
Presently sub-cluster position fitting
Position Recalculation
Only used by ECAL so far
Energy Correction
Only used by ECAL so far
PFClusters

PFBlockAlgo

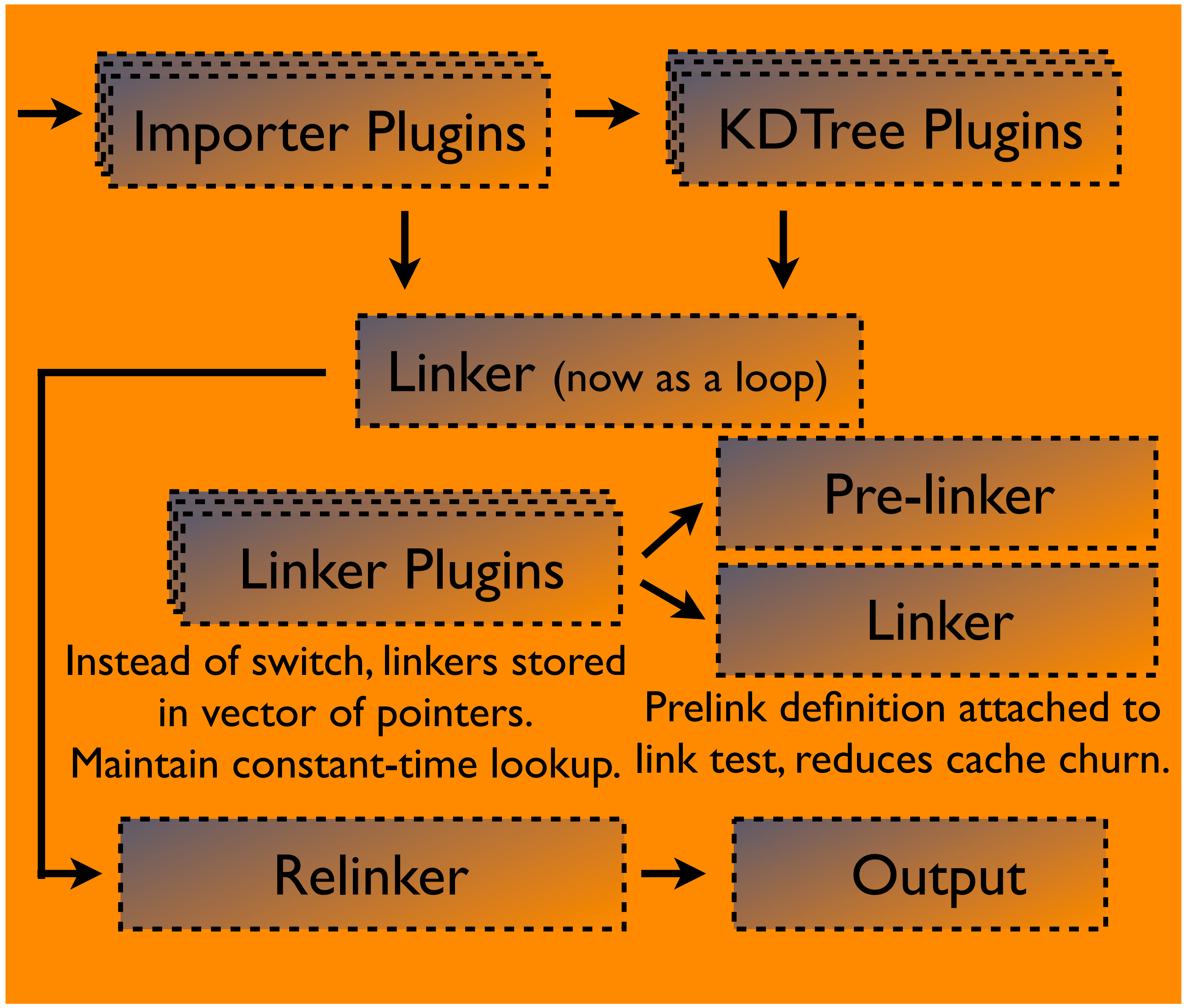
- The PFBlock algorithm makes Particle Flow possible
  - It organizes tracks and clusters into sets that are <u>topologically connected</u>
  - Significantly reduces time taken to find what objects are <u>closest</u> together
- This goal achieved by defining detector objects to 'import' and 'links' that define what objects can be neighbors and distance between them
  - Create a unique element list otherwise linking is ambiguous
  - Form the block by finding all elements related by neighbors to the current front of the element list
  - Check for extra neighbors and then start the process again for a new block
- This is the very first determination of the gross energy flow in the event
  - Elements in different blocks cannot be linked to each other


HCAL ECAL HO HF GsfTracks Gen. Tracks Conv's
Build Unique Element List
(a.k.a. 'block import')
Pre-process ECAL/HCAL/TRACK/PS with space partitioning algorithm.
(a.k.a. 'KDTreeAlgo')
Seed and fill PFBlocks
(neighbours of neighbours of...)
(a.k.a. 'linking')
Re-link PFBlocks
(check for **extra** links between elements!)

PFBlockAlgo Design

- In a similar fashion and for similar reasons we modularized the PFBlock producer
  - New block producer behavior defined entirely by python
  - Importers are clearly exposed to the user
    - New ones can be easily written and integrated to PFlow
  - Similarly various link types are now displayed as an action matrix
    - Each linker says exactly what kind of elements it connects
  - Intuitive way to configure the behavior of the first layer of particle flow
- New linking to account for new detector types is easy to write and introduce
  - Again minimal overhead and allows a contributor keep their mind close to the problem at hand

(L. Gray)


Importer Plugins
KDTree Plugins
Linker (now as a loop)
Pre-linker
Linker Plugins
Linker
Instead of switch, linkers stored in vector of pointers. Maintain constant-time lookup.
Prelink definition attached to link test, reduces cache churn.
Relinker
Output