Modularized PFBlockProducer

Lindsey Gray

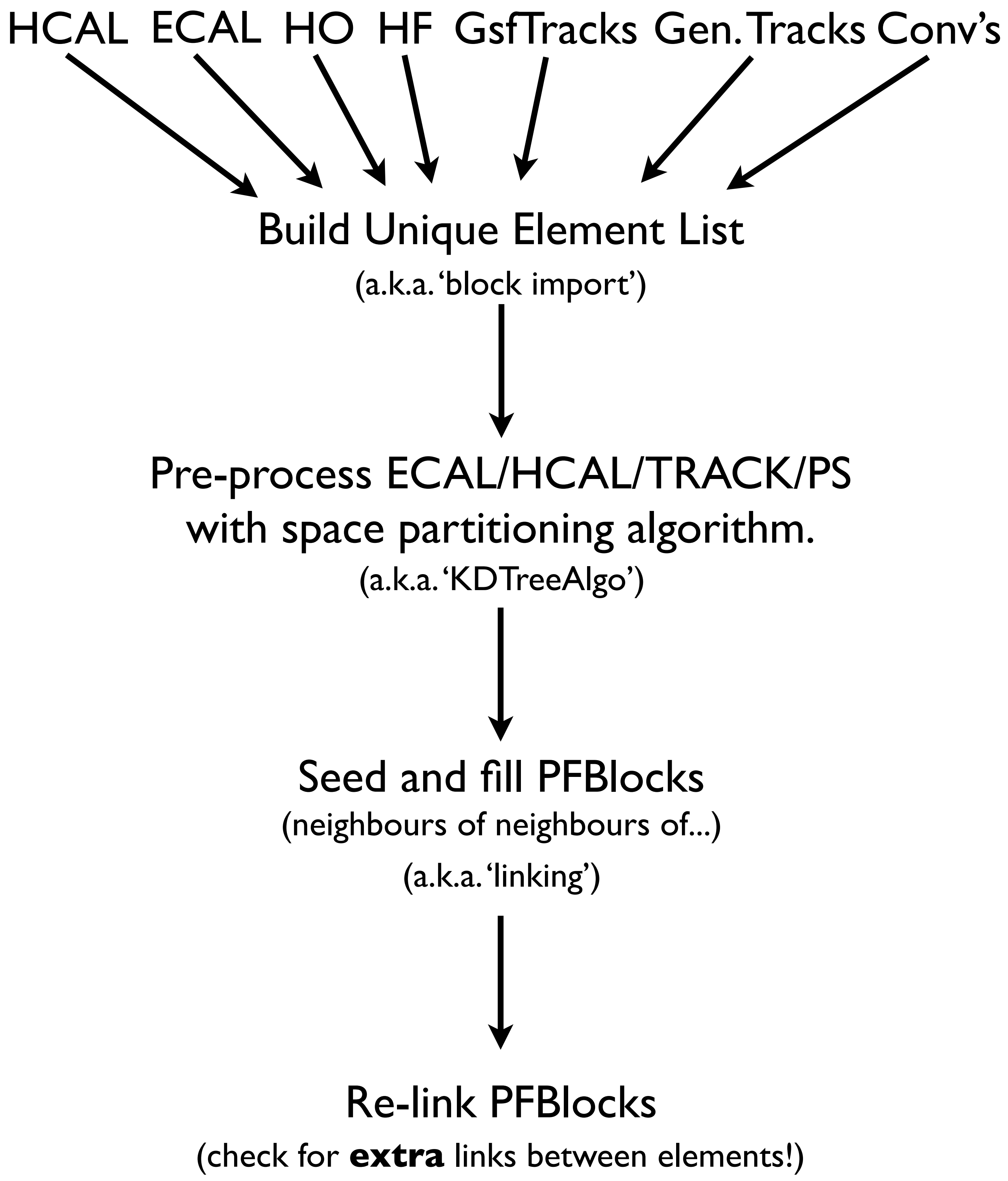
8 April, 2014

What does PFBlockProducer do?

- It is the first step that considers a true global event description in the PFlow reconstruction
  - Tracks and clusters are first compared to each other and topological associations are made
  - Exhaustive sets of associations are 'PFBlocks'
- PFBlocks are the most coarsely determined level of 'energy flow' in an event
  - Tracks are eventually linked to their **closest** cluster
  - Clusters/Tracks that are not in the same PFBlock cannot be linked to each other
  - This sets the largest scale at which we can start to mitigate the effects of double counting and really make particles
    - Subdivision into blocks makes further energy-flow determination computationally feasible by reducing combinatorics
  - Downstream reconstructions are not prevented from working across blocks (jets, taus, MET...)

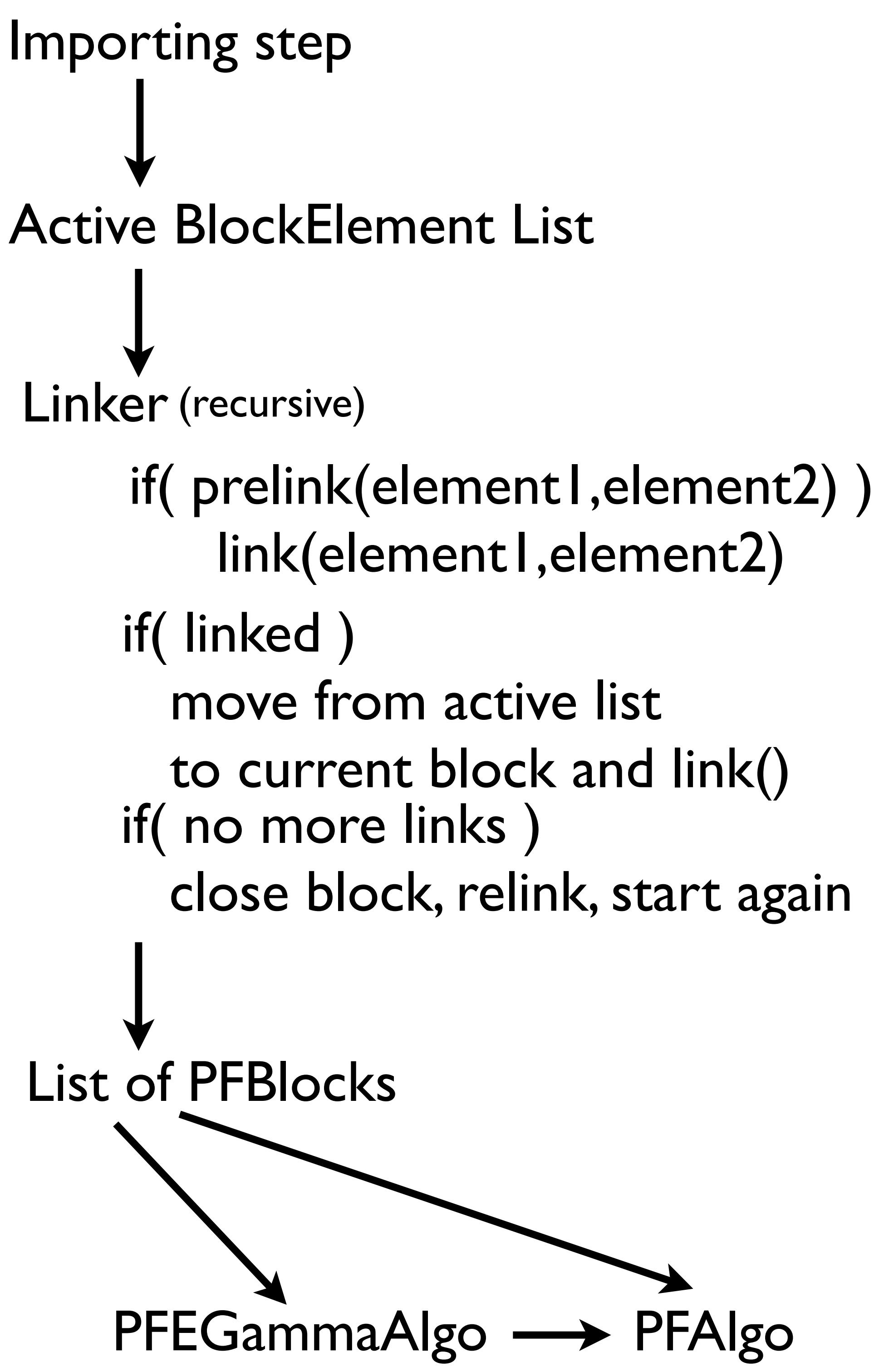
PFBlockProducer Workflow

- List of input objects different for HLT and Offline
  - Reduced list for HLT timing budget
- Most combinatorially expensive pairs are pre-processed
  - KDTreeAlgo gives quick access to closest neighbour
- Links found through iterative-neighbours approach
  - The same as topological clusters!
- Only one link tested during the first step
  - Need to check for additional links in block (can change final EFlow!)


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!)

Current PFBlockProd’r Implementation

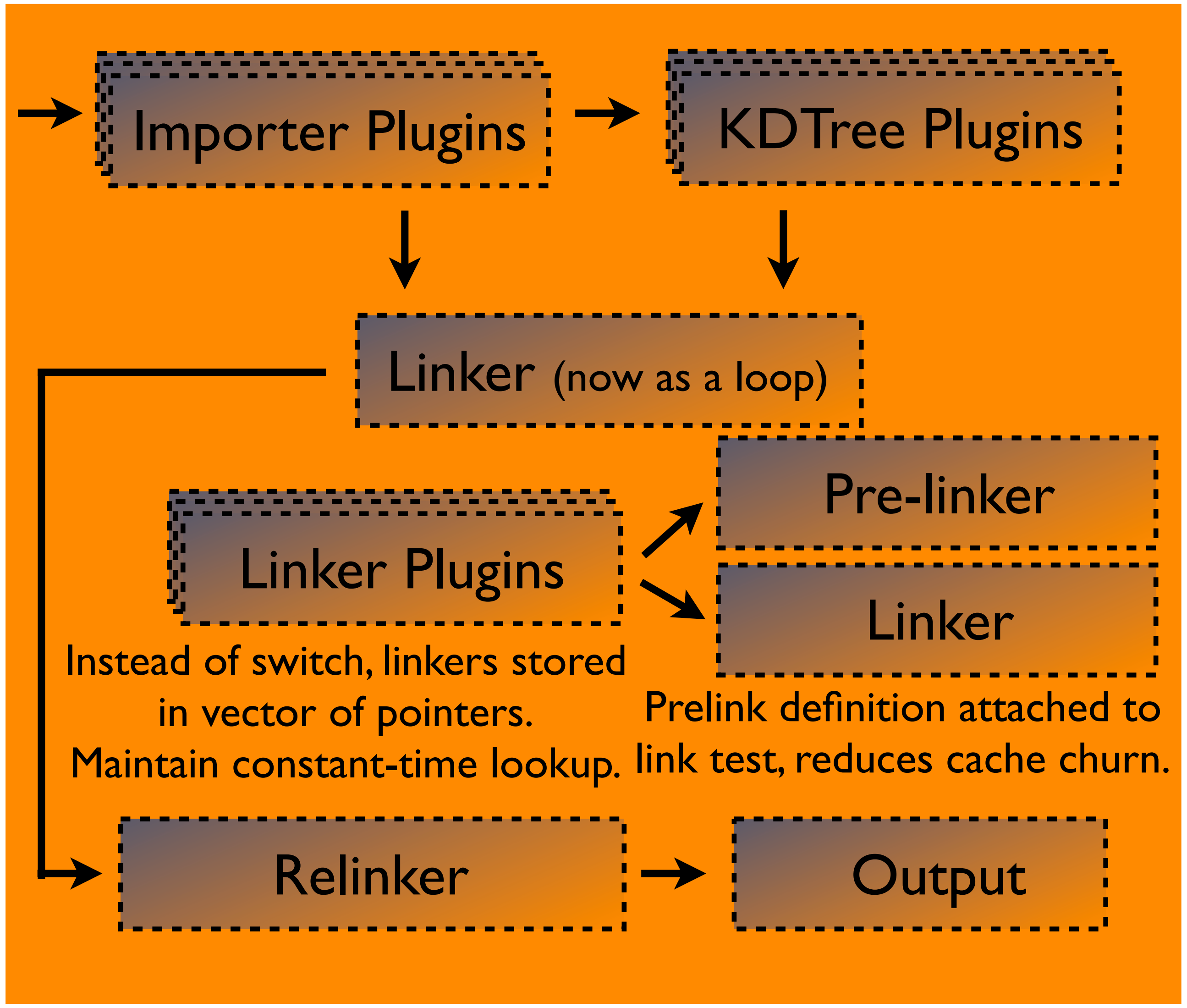
- Monolithic algorithm
    - Importing, (p/re)linking all hardcoded
- HLT mode is defined by a function template that calls another function template
- Available link types hardcoded
    - link types not obviously/pleasantly advertised
    - 1200 line switch statement (of doom)
    - Prelinker is not actually a predictor for what linking happens
        - Just that linking **can** happen
        - This makes the processor unhappy :-(
- Adding new linking somewhat painful due to prelinker being disconnected



```
if( prelink(element1,element2) )
    link(element1,element2)
if( linked )
  move from active list
  to current block and link()
if( no more links )
  close block, relink, start again
```

Proposal for New PFBlockProd'r

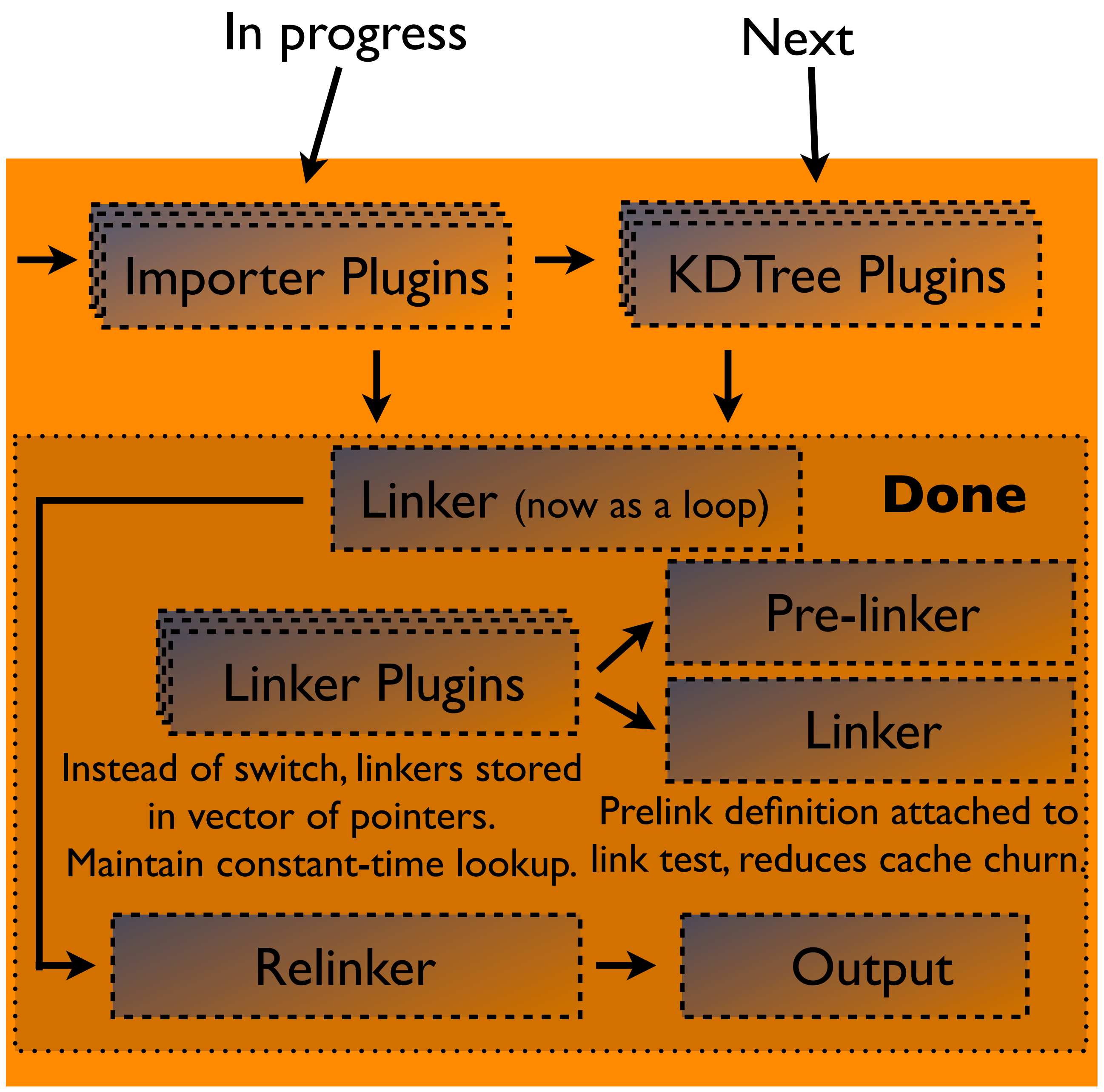
- Remove all instances of hardcoding
  - Configuration is driven entirely by python cfi's
  - Importer plugins each import one variety of object
    - Still able to protect against double-import!
  - KDTree plugins for link preprocessing
    - Possibility to add in new fast-linking in whatever variables you want
  - Linker plugins each define their pre-linking condition and link test
- Linker no longer recursive
  - Yields minor performance gain for deep function calls (+ easier to read)



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

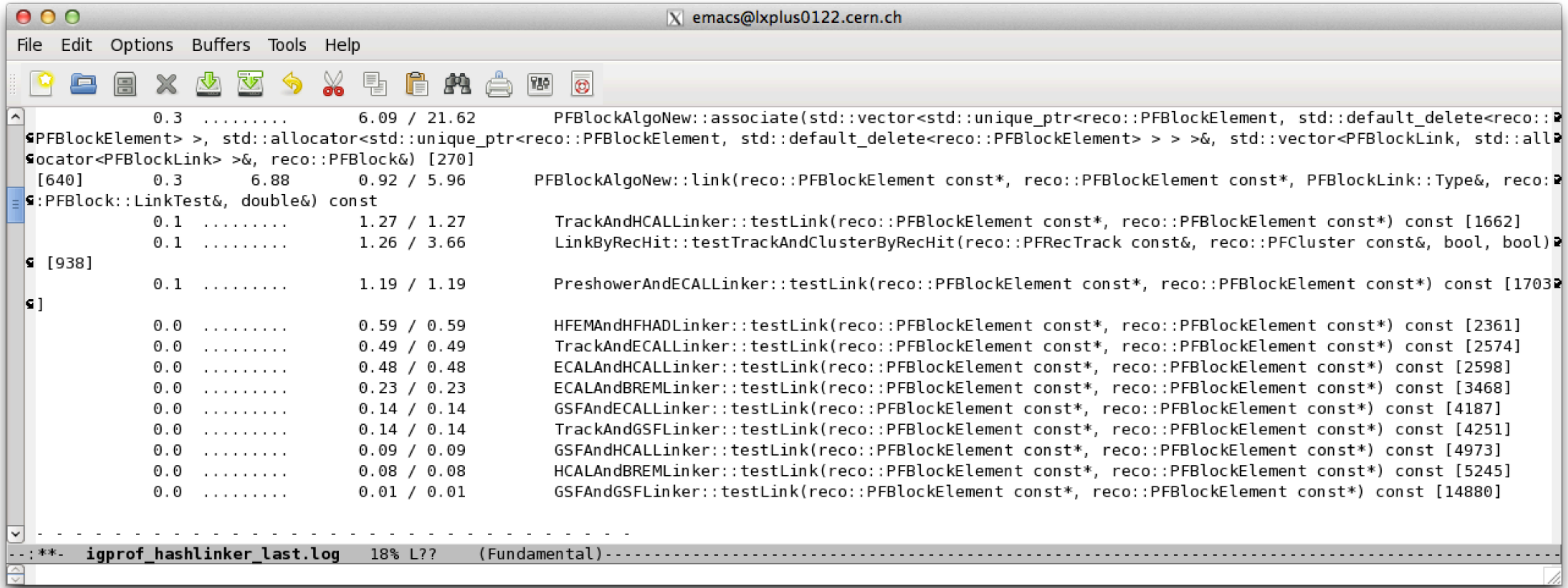
Present Status of Development

- Linking loop and linker classes fully implemented
  - Produces same blocks as before
  - 30% performance gain from binding of pre-linkers and linker
    - ttbar 25ns 20PU
    - Improves cache locality by tons in busy events
    - No evaluation of control flow that doesn't matter
  - Further improvements possible by checking for leaf elements?
- Importers going smoothly
- KDTree linker last
  - Want to revisit this a little with Maxime, improve how links are stored


In progress
Next
Importer Plugins
KDTree Plugins
Linker (now as a loop)
Done
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

Faster is better

- New design allows us to accurately account where we are spending our time!
- Even with the KDTree, the Preshower-ECAL linking (even the prelinking) takes a large-ish chunk of time... it probably shouldn't....
    - Track and HCAL biggest time user, but can be more reasonably justified



```
            0.3  .........        6.09 / 21.62        PFBlockAlgoNew::associate(std::vector<std::unique_ptr<reco::PFBlockElement, std::default_delete<reco::
PFBlockElement> >, std::allocator<std::unique_ptr<reco::PFBlockElement, std::default_delete<reco::PFBlockElement> > > >&, std::vector<PFBlockLink, std::all
ocator<PFBlockLink> >&, reco::PFBlock&) [270]
 [640]      0.3       6.88        0.92 / 5.96       PFBlockAlgoNew::link(reco::PFBlockElement const*, reco::PFBlockElement const*, PFBlockLink::Type&, reco:
:PFBlock::LinkTest&, double&) const
            0.1  .........        1.27 / 1.27         TrackAndHCALLinker::testLink(reco::PFBlockElement const*, reco::PFBlockElement const*) const [1662]
            0.1  .........        1.26 / 3.66         LinkByRecHit::testTrackAndClusterByRecHit(reco::PFRecTrack const&, reco::PFCluster const&, bool, bool)
 [938]
            0.1  .........        1.19 / 1.19         PreshowerAndECALLinker::testLink(reco::PFBlockElement const*, reco::PFBlockElement const*) const [1703
]
            0.0  .........        0.59 / 0.59         HFEMAndHFHADLinker::testLink(reco::PFBlockElement const*, reco::PFBlockElement const*) const [2361]
            0.0  .........        0.49 / 0.49         TrackAndECALLinker::testLink(reco::PFBlockElement const*, reco::PFBlockElement const*) const [2574]
            0.0  .........        0.48 / 0.48         ECALAndHCALLinker::testLink(reco::PFBlockElement const*, reco::PFBlockElement const*) const [2598]
            0.0  .........        0.23 / 0.23         ECALAndBREMLinker::testLink(reco::PFBlockElement const*, reco::PFBlockElement const*) const [3468]
            0.0  .........        0.14 / 0.14         GSFAndECALLinker::testLink(reco::PFBlockElement const*, reco::PFBlockElement const*) const [4187]
            0.0  .........        0.14 / 0.14         TrackAndGSFLinker::testLink(reco::PFBlockElement const*, reco::PFBlockElement const*) const [4251]
            0.0  .........        0.09 / 0.09         GSFAndHCALLinker::testLink(reco::PFBlockElement const*, reco::PFBlockElement const*) const [4973]
            0.0  .........        0.08 / 0.08         HCALAndBREMLinker::testLink(reco::PFBlockElement const*, reco::PFBlockElement const*) const [5245]
            0.0  .........        0.01 / 0.01         GSFAndGSFLinker::testLink(reco::PFBlockElement const*, reco::PFBlockElement const*) const [14880]

 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
--:**-  igprof_hashlinker_last.log   18% L??    (Fundamental)--------------------------------------------------------------------------------
```

Conclusions / ToDo

- Modularized implementation of PFBlockProducer/Algo proposed
  - Improves immediate understanding of what's being done
  - Modular design promotes future-proofness of CMS PF code
    - Adaptable with minimal pain to changes in underlying inputs
    - First test will be switching to direct SuperCluster input
  - New ideas for linking can be testing with minimal overhead, learning-curve, and turn around time
- Modularized linker implemented
  - Gives same results as old linker
    - No numerical oddities like with the clusters
  - Inherently faster than original
  - Still improvements/optimizations to be made
    - Still, a cheap lunch is nice some times
- Development in time for 710
  - Plan to backport immediately to 620_SLHC, along with clustering