HLT in athena MT

T. Bold AGH UST Krakow
for HLT s/w upgrade group

Why it is good to follow this path?

- Maintenance of trigger framework requires significant effort
    - Would be even more significant for concurrent events processing
- High rejection power available in offline codes
    - We already do that at EF for exact the same reason (egamma PIDs, b-jet taggers)
- Synergy between fast/early stage HLT algorithms and offline would be valuable (offline needs to get faster as well)

Offline framework offers (rightly) only few components:
Service, Algorithm, Tool
- goal is to cast HLT task into this components

HLT steering = offline algorithms

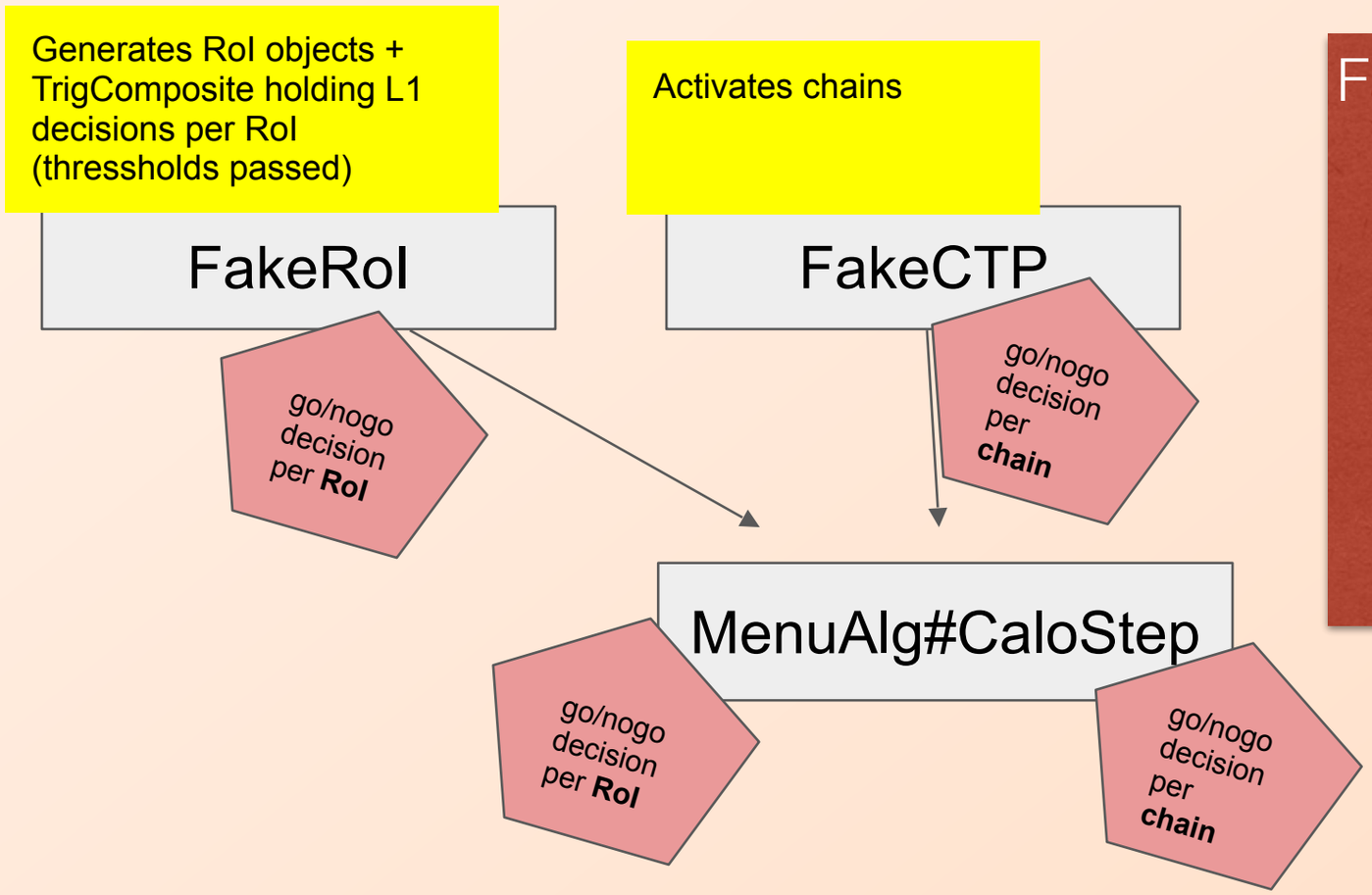
Menu as a set of plain algorithms

- Tricky part: interplay of the reco. and the menu.
- We retain concept of the chain - but it is not a functional object - just a concept.
- Step of the chain: needs an input (decisions) and produces an output (reduced set of positive decisions per chain/per RoI).
  - The algorithmic logic in the menu is fairly simple
    1) counting, 2) counting unique RoIs, 3) prescaling == generalised implementation
    - Menu algorithm == Step of set of similar chains
      → alg. implements one step e15_tight_calo, e15_loose_calo, 2e15_loose_calo counting
- Menu: assembly of consistently configured menu algorithms
  - +With them: testing will be simplified → FEX alg(s), Hypo alg(s), menu alg. == completely defined job

Menu as an algorithm - trial implementation

Algorithm has **two inputs**, and generates **two outputs**.

Decisions: per-chain and per RoI



Fake bootstrap generators
- fake RoI maker - to place desired number & types of RoIs in each event
- fake CTP make - activates desired "chains" for each event

Both configured from text files and require no input data - test jobs takes about 30s.

MenuAlg::exdecute (pseudo code)

```
for ( chain: chains_handled_by_this_algo )
   if ( not chainSurvivedPreviousStage(chain) ) continue; // keep rejected
   if ( requirement_satisfied(chain) )
      chainDecision->setPassed();
   else: chainDecision->setFailed();
for ( roi: rois )
  if ( contributedToPassedChain(roi) )
     roiDecision->setPassed();
```

We would be free to deliver other implementations
overlap checking
…

Rejection

Prescaling

- Decides if chain is interesting
- PrescaleDecision (see ViewAlgs/src for the code)
    - demo config:

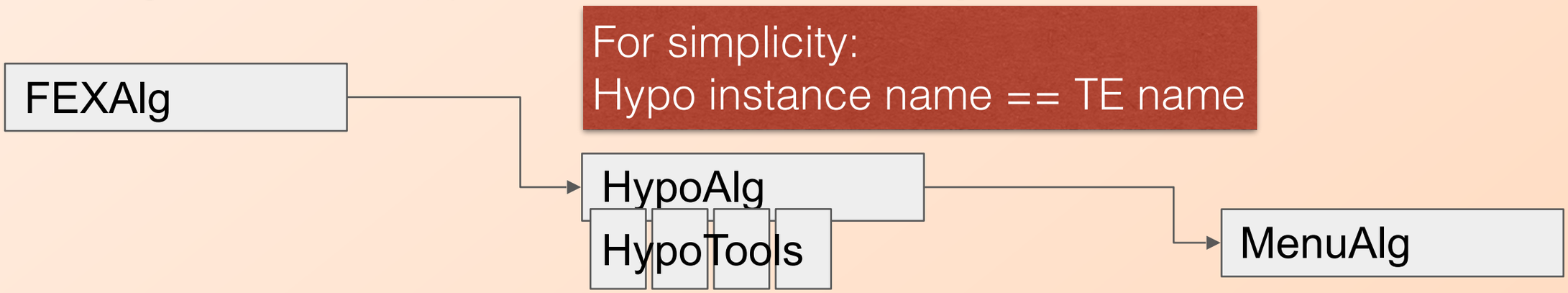
Chains in input

Reduced set of chains as output

```
from ViewAlgs.ViewAlgsConf import PrescaleDecision
ps  = PrescaleDecision("HLTChainsPrescaling")
ps.Prescales=["HLT_e3 20.99", "HLT_e7 2.5"]
ps.InputChainDecisions = "CTPDecisions"
ps.OutputChainDecisions = "ChainsPassingAfterPS"
ps.OutputChainDecisionsAux = "ChainsPassingAfterPSAux."
topSequence += ps
```

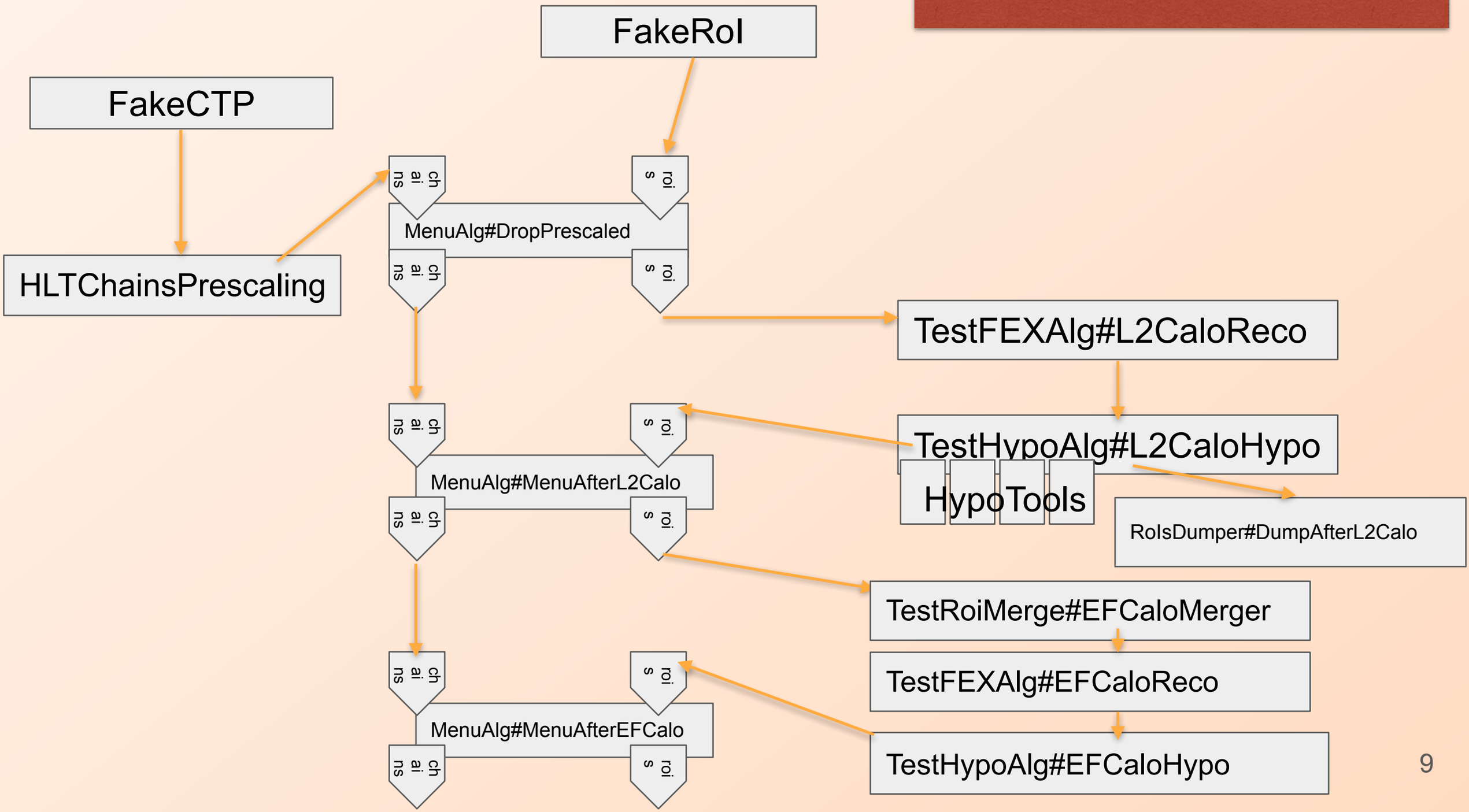
Testing: TestFEXAlg TestHypoAlg

- FEX algo (looking like producer of CaloClusters).
- Hypo algo works on clusters and decides:
  is the threshold passed?
- One algorithm serves many hypotheses verification
  - cuts outsourced to TestHypoTool(s) — we think this is a migration path.
  - One algorithm == no need to caching
  - hypo tools executed explicitly by the HypoAlg.
    - no sequence needed for that
- Wiring FEX and Hypo **via R/W handle dependencies**

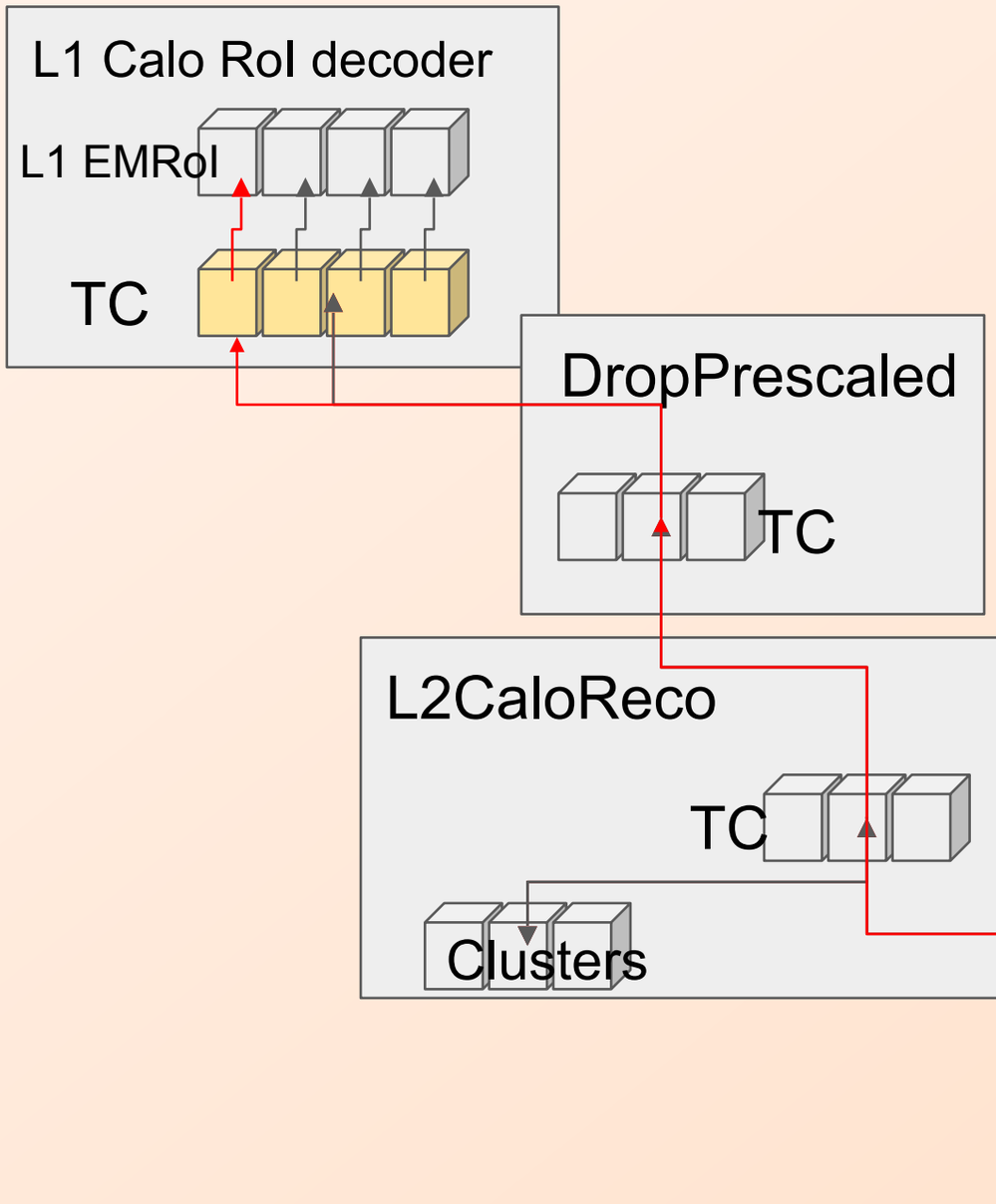


Assembled menu

New package ViewAlgsTest
hosts test algs & test itself



Wiring EDM object - incarnation of HLT navigation



TC == TrigComposite - yellow ones in addition to the links contain decisions of Hypo algorithms

Unique way of accessing objects through the ELs in TC (see the red path)

Advanced aspects RoIs merging/splitting

It may be/is beneficial to merge many RoIs to do one reco. step: jets, b-jets PV, overlapping reco.

Prototype algorithm in place (see TestViewAlgs):

- merges RoIs according to configured rules
- produces new RoIs (super RoI obj)
- objects are linked back to the source
- splitting explores this links back to decompose results to original RoIs

Studies ongoing

Merging and Splitting

Francesca Pastore

L1Decoder

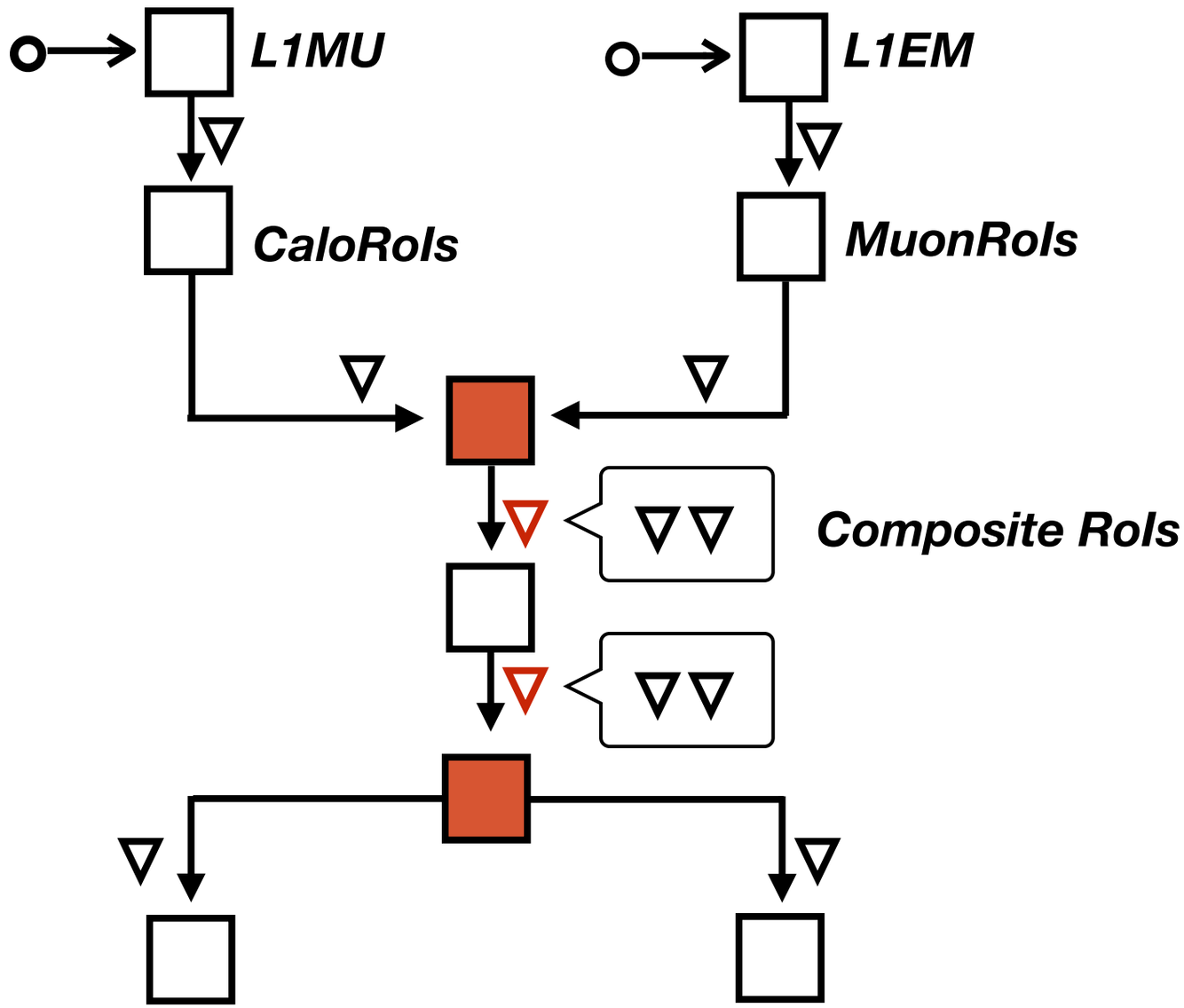
L2Alg

Merging

Tracking Alg

Splitting

EFTrack Match



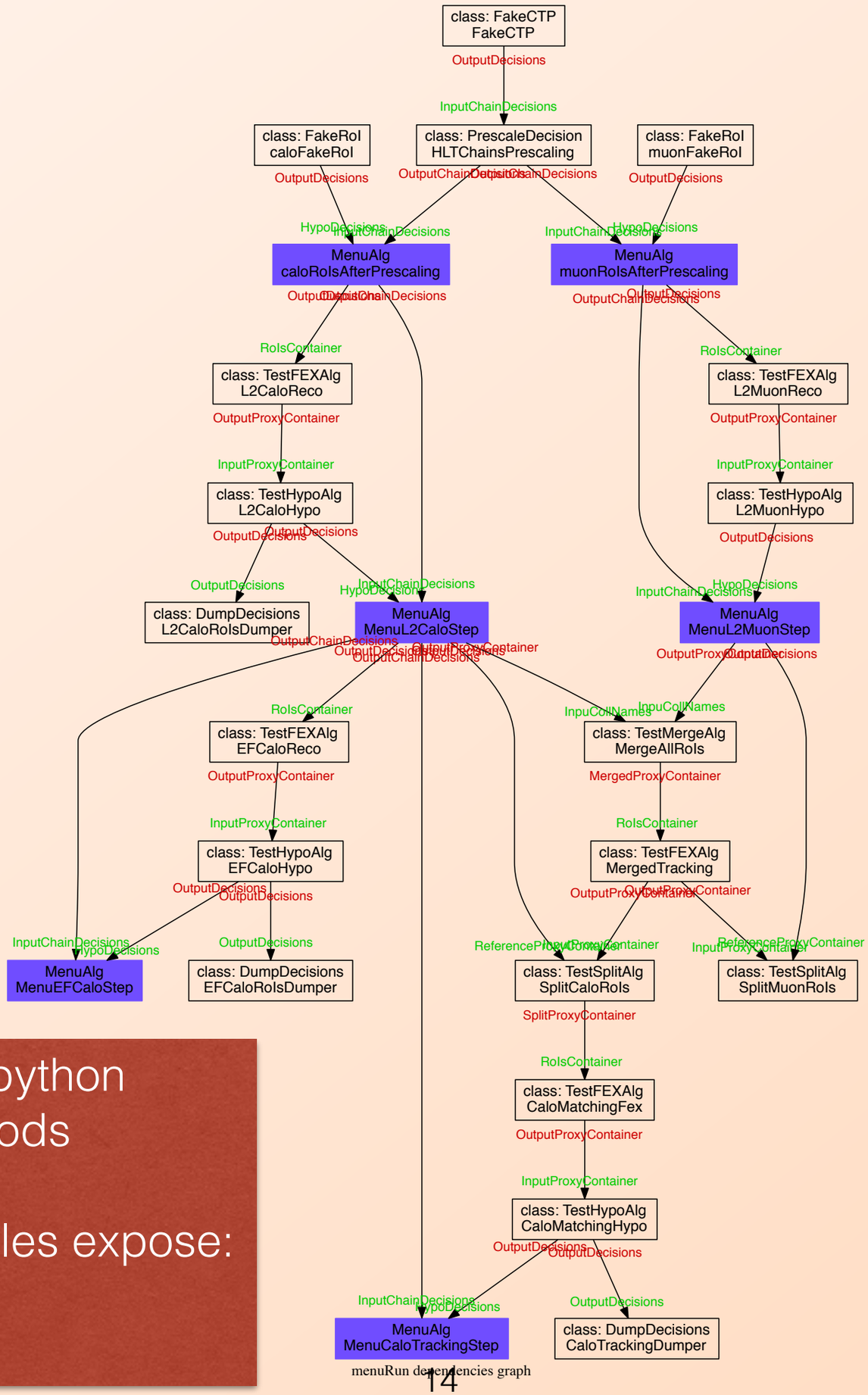
A more complex menu = relation to HLT chains

- Splitting chains from algs.: attempted in python:

```
chains = ChainsList()
chains + Chain("HLT_e3", "L1_EM3", 20.99, {"DropPrescaled": "EM3 x 1",
                                           "MenuL2CaloStep": "L2_3GeVCaloEtCut x 1",
                                           "MenuEFCaloStep": "EF_3GeVCaloEtCut x 1"})
```

- This information is then feed back to MenuAlgorithms

- Translation from HLT XML config would require:
  - re-arrangement of the signatures info into Menu Alg. properties
  - cast of chain properties like. PS to a appropriate alg. properties
  - mapping of **chain step** —> **Menu Alg.** should not be needed i.e. only one Menu Alg. is responsible for L2_3GeVCaloEtCut and this can be discovered in python



This is extracted from python via convenience methods connectAlgorithmsIO

Would be great if handles expose:

- r_or_w
- handled coll. type

menuRun dependencies graph

Summary & outlook

- Prototyping helped understanding if HLT can be cast to offline
    - with caveats: (no RoIs, no summary algo. yet, hacked ReadHandleArray) it seems possible to perform basic HLT tasks in purely offline framework
- Next steps for prototype:
    - Incorporate views
    - Expand set of menu utility algos to be able to arrive at final YES/NO + streams
    - Address coherence with existing HLT (adiabatic migrations of hypos, reuse of menu)
    - Trigger analysis tools - keep TDT interface