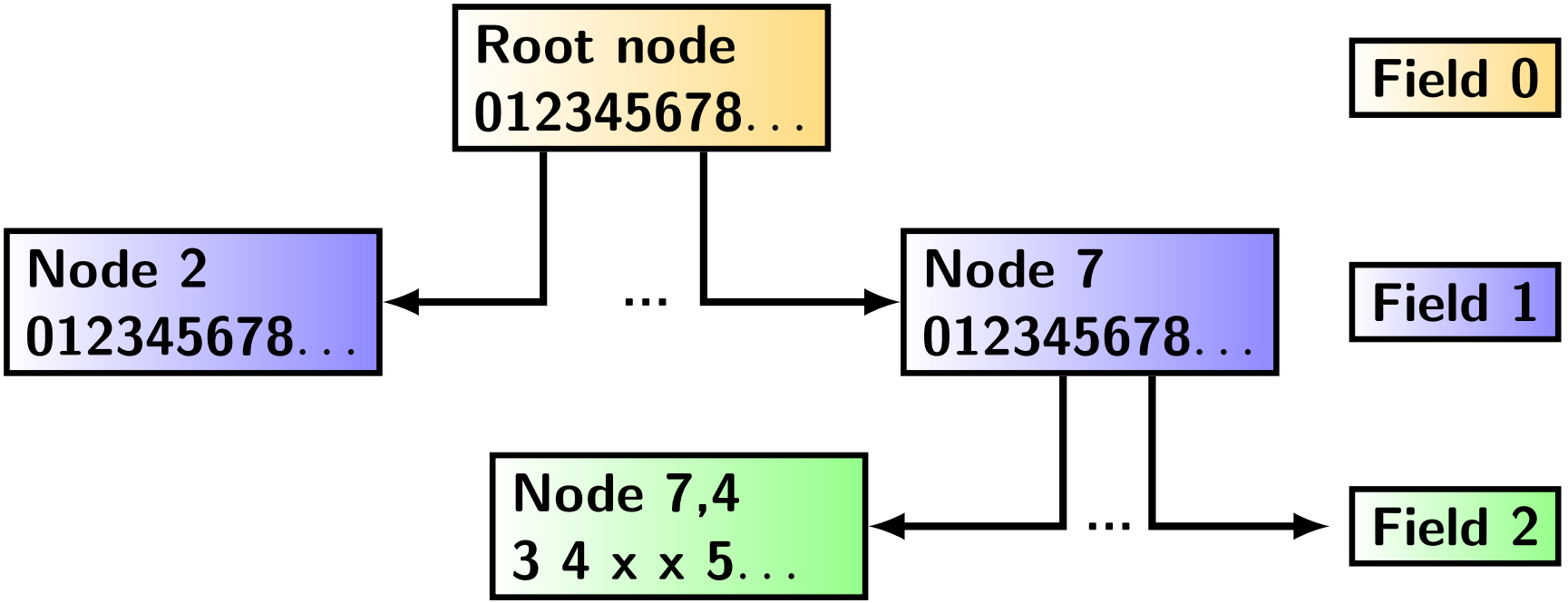
Calorimeter reconstruction performance

- ALLEGRO calorimeter reconstruction was unusably slow when noise was enabled (and all $\sim$ 2M cells are filled).
- Main culprit: lookup of cells by `CellID`.
- `CellID` is a 64-integer divided into bit fields: system, type, theta, phi, etc. $\rightarrow$ a *sparse* numbering scheme.
- Calibration constants, noise, etc. stored in `map`/`unordered_map` of `CellID`.
- Not cache-friendly, significant memory overhead.
- Would be much better to assign each cell an index from 0 to $N_{\text{cells}} - 1 \rightarrow$ a *dense* numbering, and store everything in `vectors`.
- Similiar to the `IdentifierHash` concept from ATLAS.
- ATLAS forms `IdentifierHash` with subsystem-specific code.
- Want something a little more generic at this point.

IDMap

- A trie-like map data structure, using a subset of `CellID` fields.



- For example, for EMB, use fields layer, theta, module.
  - (Order matters — small to large for best performance.)

	sequential	random	memory
`unordered_map`	3.30	9.94	
`IDMap`	0.25	4.49	15M
array	0.23	4.04	50M

Implementation 1

- Introduce `ICaloIndexer`:

```
virtual index_t index(CellID id) const = 0;
virtual std::span<const CellID> cellIDs() const = 0;
virtual std::span<const int> detIDs() const = 0;
virtual size_t detIDBits() const = 0;
```

  with implementations `IDMapIndexer` for a single subdetector and `MultiIndexer` to concatenate several subdetectors.

- Extend `ICalorimeterTool` to return cells and an indexer:

```
virtual const std::vector<CellID>& cellIDs() const = 0;
virtual std::unique_ptr<k4::recCalo::ICaloIndexer> indexer() cons
```

  Implementations in place for EM barrel and HCal; EM endcap awaiting merge after final validation.

Implementation 2

- Introduce `CaloCellIndexerSvc`:

```
virtual const ICaloIndexer*
  indexer(int detID, bool quiet = false) const;
virtual const ICaloIndexer*
  indexer(std::span<const int> detIDs, bool quiet = false);
```

  to return an indexer object for a set of detectors. Must be configured with the `ICalorimeterTools` for the detectors to be handled.
- Calibration, etc. tools have been storing their data in the tool instances.
  - Can lead to substantial wasted memory if tools are duplicated.
  - Introduce `CaloCellConstantsSvc` to store a single copy of the constants. (Poor man's detector store?)

```
template <class T>
const T* getObj(const std::string& key) const;
template <class T>
bool putObj(const std::string& key, T&& obj);
```

Pending merges

- Update calibration, noise, and crosstalk tools to use the new services.
- Update `CreateCaloCells`:
  - Use the indexing tool instead of `std::map`.
  - Avoid using `mutable` members to store event data.
  - Properly handle positioning. Eventually want to get rid of `CreatePositionedCaloCells`.
  - Update `CreateTopoClusterFCCee` to use indexing rather than maps; other speedups..
- In the end, gives a 15–20$\times$ speedup and a $\sim$ 25% memory reduction.

Calorimeter segmentations

- Cell structure is defined by `Segmentation` classes.
- `Segmentation` has a method to return a position for a given `CellID`.
- DD4hep interprets this in the coordinate system of the G4 volume.
- The calorimeter segmentation tools were not doing this, though; this resulted in hits being assigned incorrect positions.
- In reconstruction, positions are reset using a detector-specific positioning tool which knows the unique convention used by the segmentation.
- Updated EM Barrel and HCal segmentations to use the proper position convention. Now hit positions are correct, and can use a generic positioning tool (and for cells from hits, position can be taken directly from hits).
- EM endcap pending (Erich may be working on this?).

Job configuration

- Prototype ComponentAccumulator-style utility present in FCC-config.

```
from FCC_config.ALLEGRO.CreateCaloCells import defineCaloCellFlags,
    CreateECalBarrelCellsCfg
from FCC_config.ComponentAccumulator import ComponentAccumulator
flags = Flags()
...
defineCaloCellFlags(flags)
flags.ECal.Barrel.addCrosstalk = addCrosstalk
...
caldigi_cfg = ComponentAccumulator()
caldigi_cfg.merge(CreateECalBarrelCellsCfg(flags))
...
TopAlg += caldigi_cfg.algs()
ExtSvc += caldigi_cfg.svcs()
```

- Factor out configuration code into individual python modules.
- Can be shared between configurations (eg. o1_v03, o2_v01).
- Resolving conflicts to enable merging cell/cluster reco code.