Instrument 2.0 Update

Lamar Moore

Overview

1. Motivation
2. Prototyping
3. Implementation
4. Rollout and Challenges
5. Future

Motivation

Collaboration with the ESS and the decision to use the Mantid Framework as the tool for live data reduction presented a few major concerns about the current implementation of the virtual instrument in Mantid.

- **Complexity** – can the virtual instrument accurately represent experiment complexities for proposed instruments?
- **Performance Demands** – is Mantid performant enough to meet the live reduction challenge presented by the ESS?
- **Instrument Definition** – is the current scheme and syntax rich enough to support new instruments? (e.g. voxels)

Science & Technology Facilities Council ISIS

Motivation

Simon Heybrock (ESS) undertook a Performance Analysis survey of Mantid which was written in 2015.

[Performance Analysis Report 2015 - Simon Heybrock]

Motivation

Survey Findings (Section 7):

- Major performance bottleneck around geometry access for data reduction. SANS2D used as case study.
- `getDetector()` – instrument copies, map lookups.
- `getPos()` – requires traversal of instrument tree every time and `ParameterMap` lookup.
- `ParameterMap` access and COW mechanism suboptimal.
- Detectors identified by `detid_t` which is `typedef int32_t`
- Caching mechanism hindered by map lookups.
- Hugely inefficient for read access.

Science & Technology Facilities Council ISIS

Motivation

Survey Findings (Section 7):

- No easy way to include moving instruments.
- Instrument is duplicated for every workspace which would have performance implications in a distributed scenario.

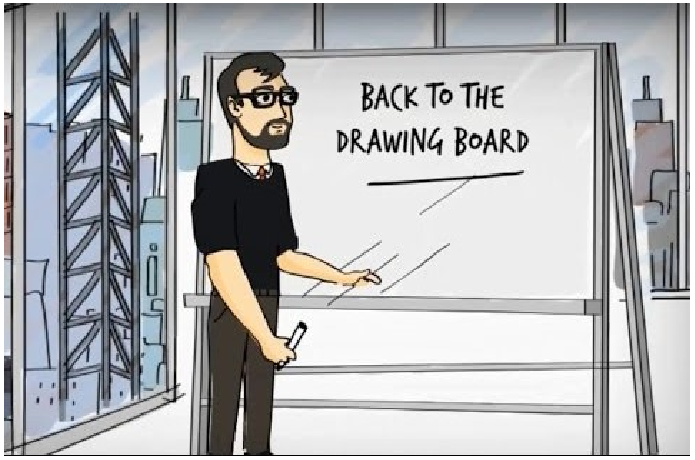
Motivation

Requirements gathering exercise 2016 which involved all of the facilities.

- Performance – should be at least 10x faster for reads. Highlighted the possibility of using SoA approach as opposed to the current AoS.
- Scanning (Moving Instruments) – Step, Scan, Triple Axis
- Threading - should be threadsafe for reading, does not need to be for writing.
- Visualization – (Instrument View)

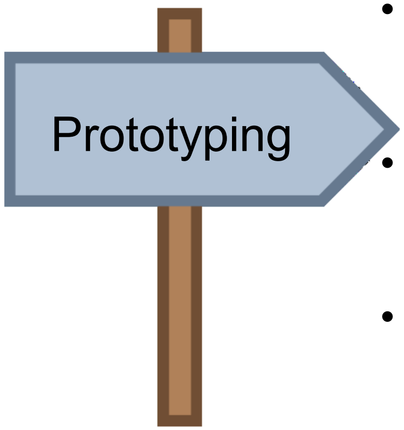
Motivation

Evaluation of the current state of the virtual instrument in Mantid against the requirements gathered lead to the decision for a rewrite of the virtual instrument in Mantid which was labelled Instrument 2.0.



Final approval of the requirements document kicked off a set of milestones for the project which included prototyping and scope setting.

Prototyping

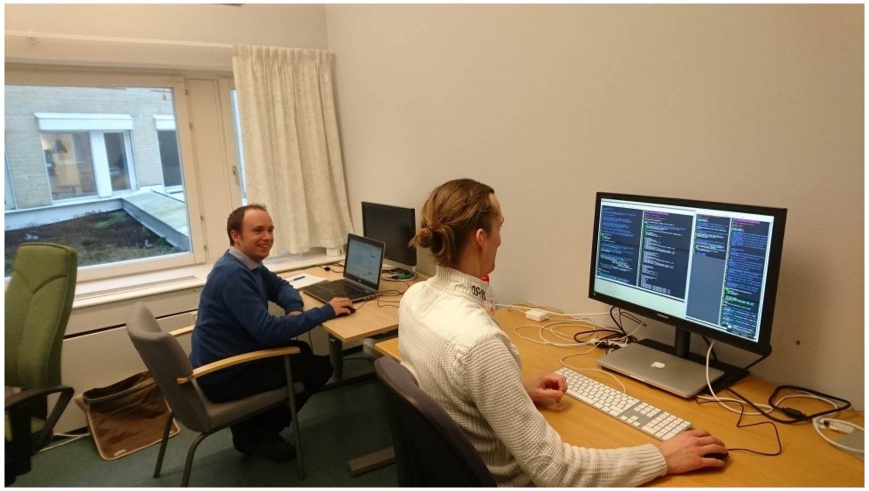


- Set out in Milestone 1.0. Investigation into the current state of the Instrument 1.0 functionally.
- Owen Arnold steered the prototype against the activity outputs high-risk and high priority items developed first.
- Large scale prototyping effort (2 rounds). Iterative, several prototypes were discarded.
- Isolated from Mantid Framework.
- Extensive Benchmarking.
- No performance compromises.

Prototyping

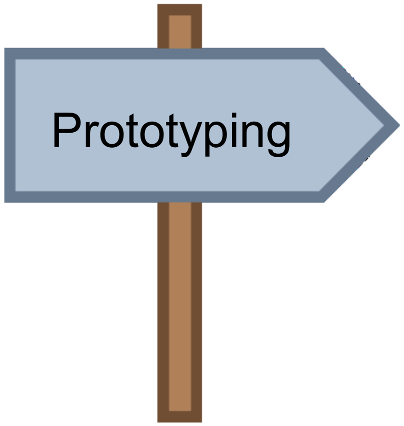
Prototyping

- Investment from ILL for scanning due to a need for scanning support.
- This came at the end of the prototyping period.
- ESS provided bulk effort design and infrastructure changes.
- ILL provided testing against real instruments and high-level functionality.


Ian Bush (left) and Simon Heybrock at the Scanning Workshop in Lund February 2017

Science & Technology Facilities Council ISIS

Prototyping



- Outputs `Beamline::SpectrumInfo`, `Beamline::ComponentInfo` and `Beamline::DetectorInfo` which represent Instrument 2.0. There is no actual Instrument 2.0 entity.

- Two orders of magnitude improvement in speed for reads.

- An order of magnitude improvement in speed for writes.

- Beamline provides native support for asynchronous and synchronous scans of detectors and synchronous scans of component assemblies. Flat tree data structure of new layers made this possible.

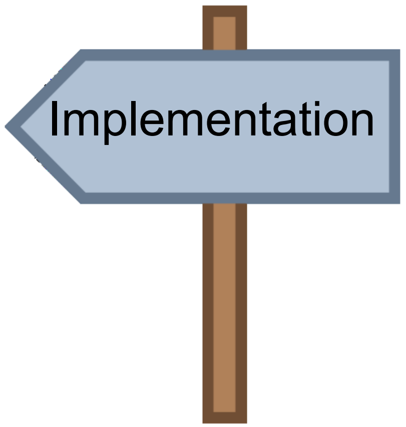
Science & Technology Facilities Council
ISIS

Implementation

- `Mantid::Beamline` has no dependence on `Mantid::Kernel`. The Eigen library was used in the prototypes and preserved for use in Instrument 2.0.

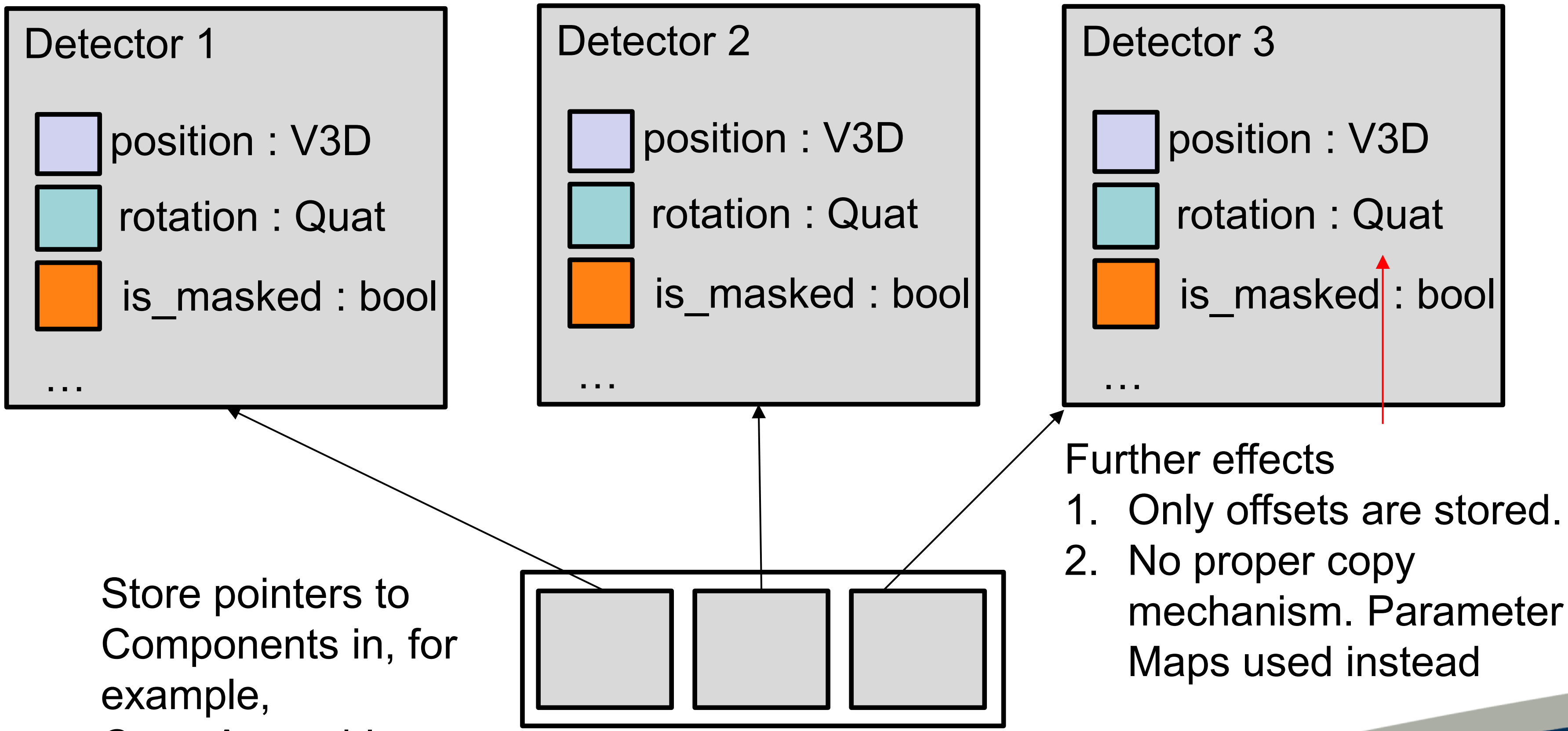
- There is no inheritance structure as with Component. Some Type information is however still preserved e.g. `Beamline::ComponentType`. Allows leveraging some faster customised approaches to some calculations e.g bouding box calculations.

- **Structure of Arrays** approach used to provide layers of access.



Implementation

Instrument 1.0 Storage



Further effects

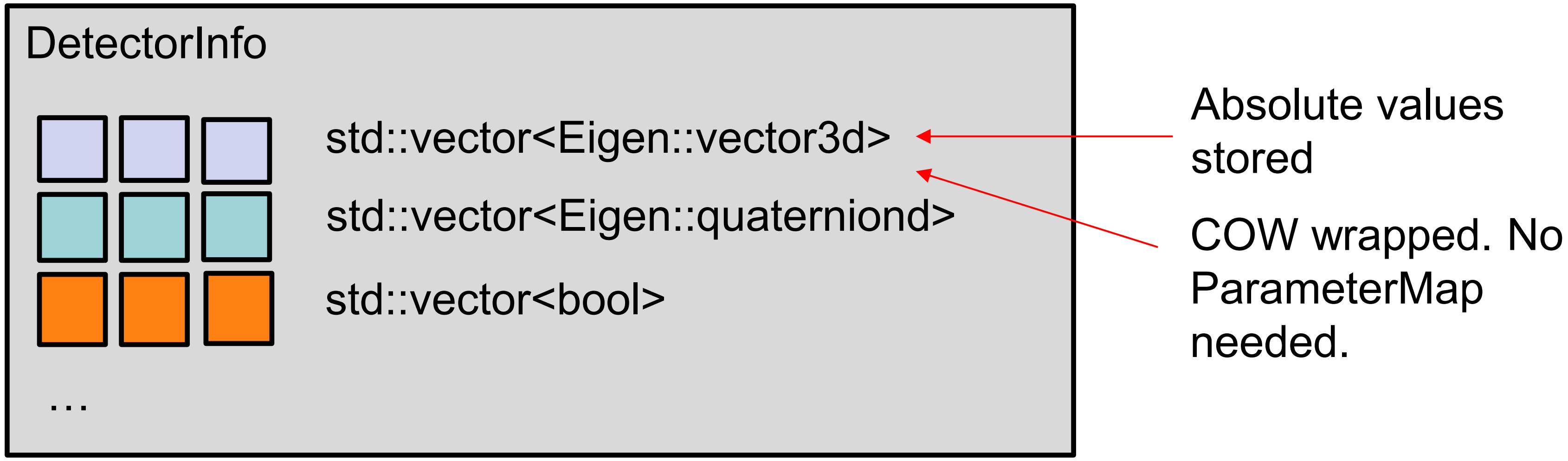
1. Only offsets are stored.
2. No proper copy mechanism. Parameter Maps used instead

Store pointers to Components in, for example, CompAssembly as std::vector

Array of Structures Approach

Implementation

Instrument 2.0 Storage



DetectorInfo
std::vector<Eigen::vector3d>
std::vector<Eigen::quaterniond>
std::vector<bool>
…
Absolute values stored
COW wrapped. No ParameterMap needed.

Structure of Arrays approach

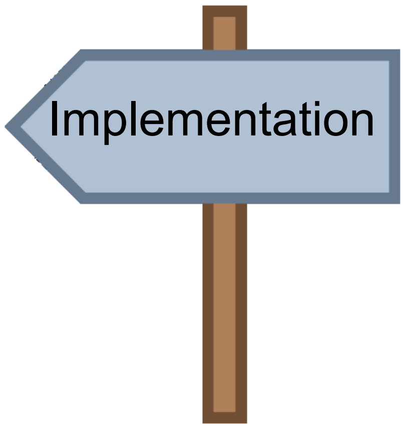
More data in the cache and fewer misses for most scenarios. Overall much, much faster access.

ComponentInfo is similar and also stores "Flat Tree" for navigation

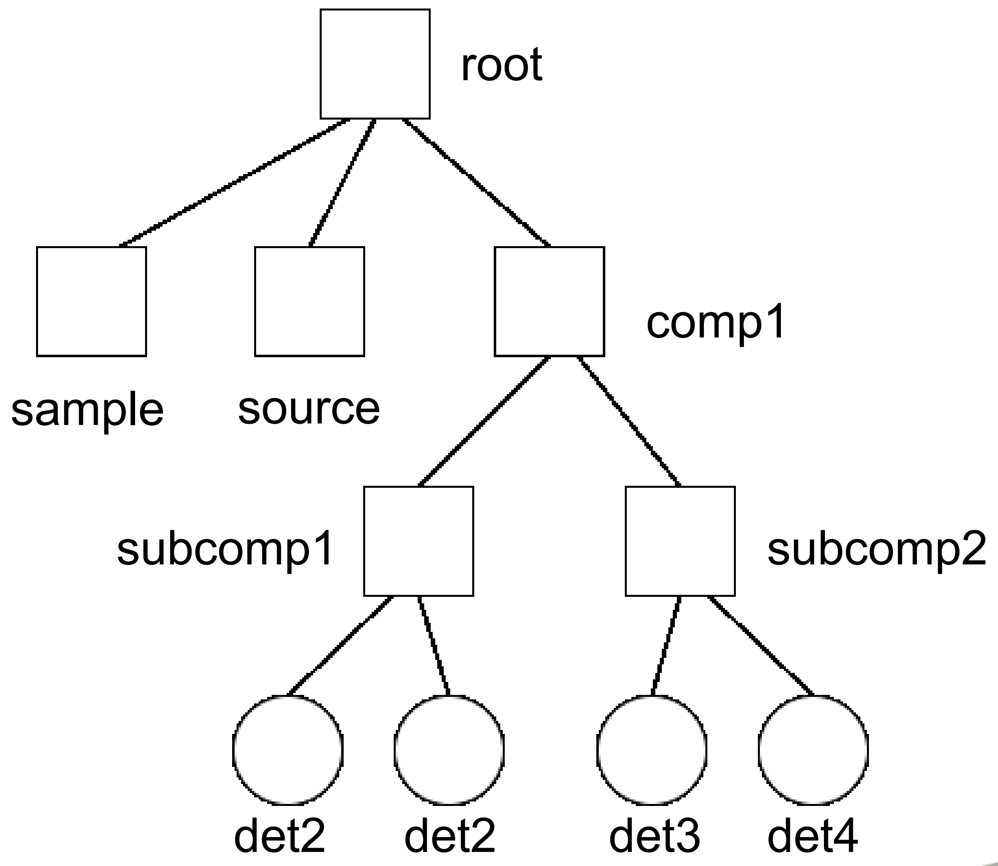
Implementation

Instrument Access Layers

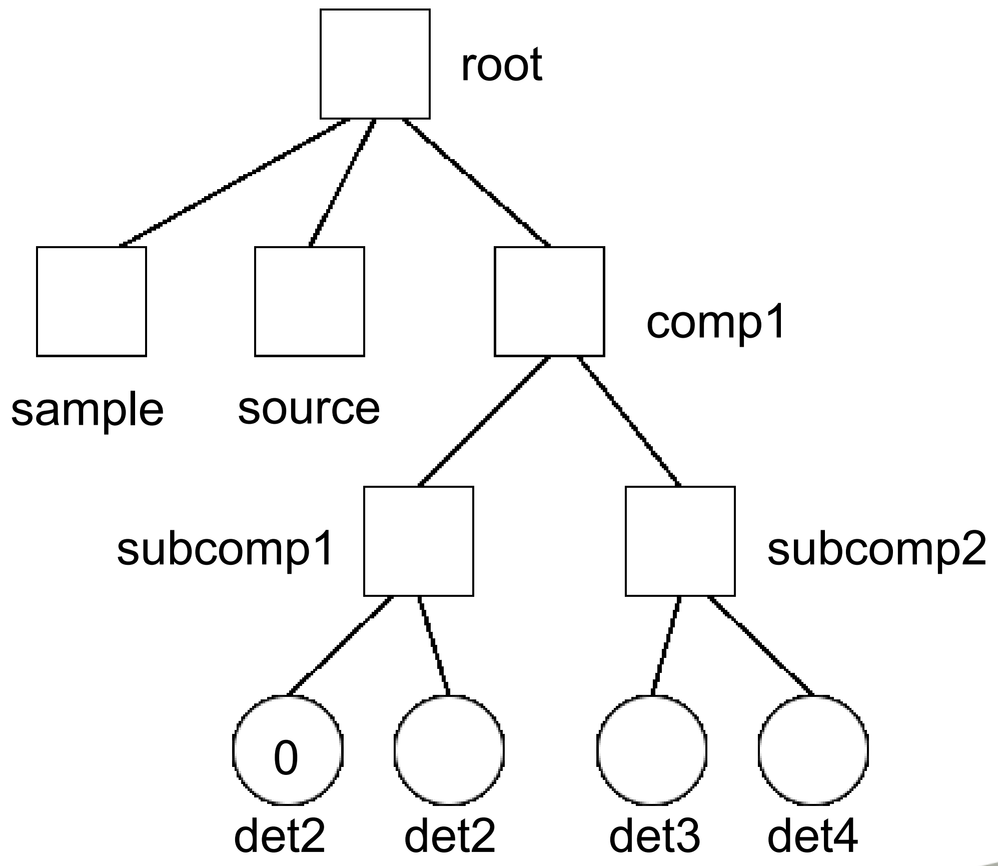
- Introduces a new paradigm for instrument access via indexing. No more map lookups for things like position and rotation.
  `ComponentInfo::position(index)`
- SpectrumInfo is a view of the instrument from a spe perspective e.g.
  `SpectrumInfo::isMonitor(index)`
  `SpectrumInfo::spectrumDefinition(index)`
- DetectorInfo contains an interface for dealing specif detectors.
  `DetectorInfo::isMasked(index)`
  `DetectorInfo::setPosition(index, position)`
  `DetectorInfo::l2(index)`
- ComponentInfo contains an interface for dealing with generic components (detectors included).
  `ComponentInfo::root()`
  `ComponentInfo::children(index)`
  `ComponentInfo::detectorsInSubtree(index)`



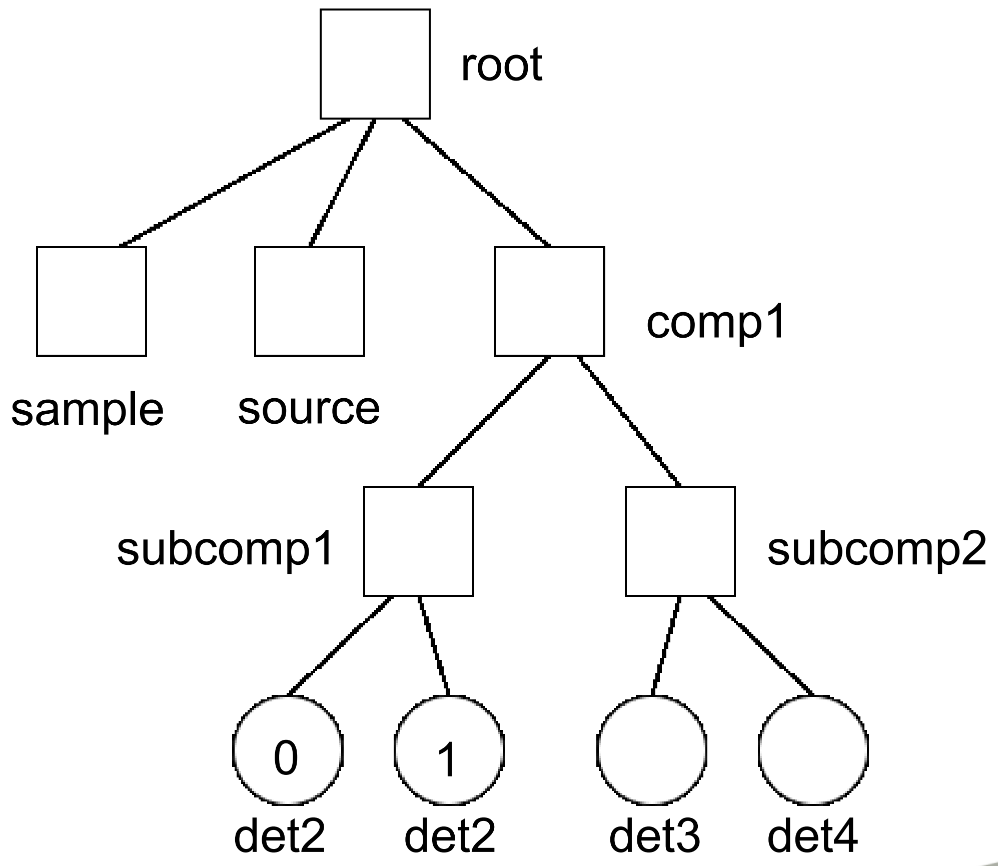
Implementation



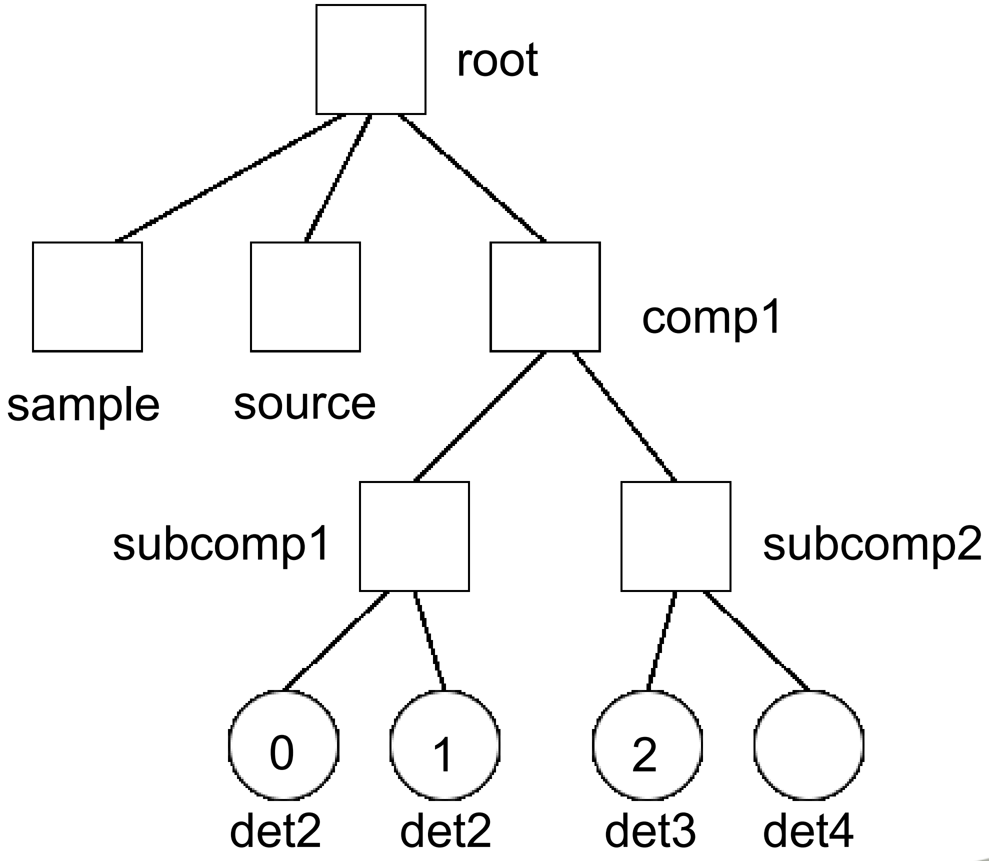
Implementation



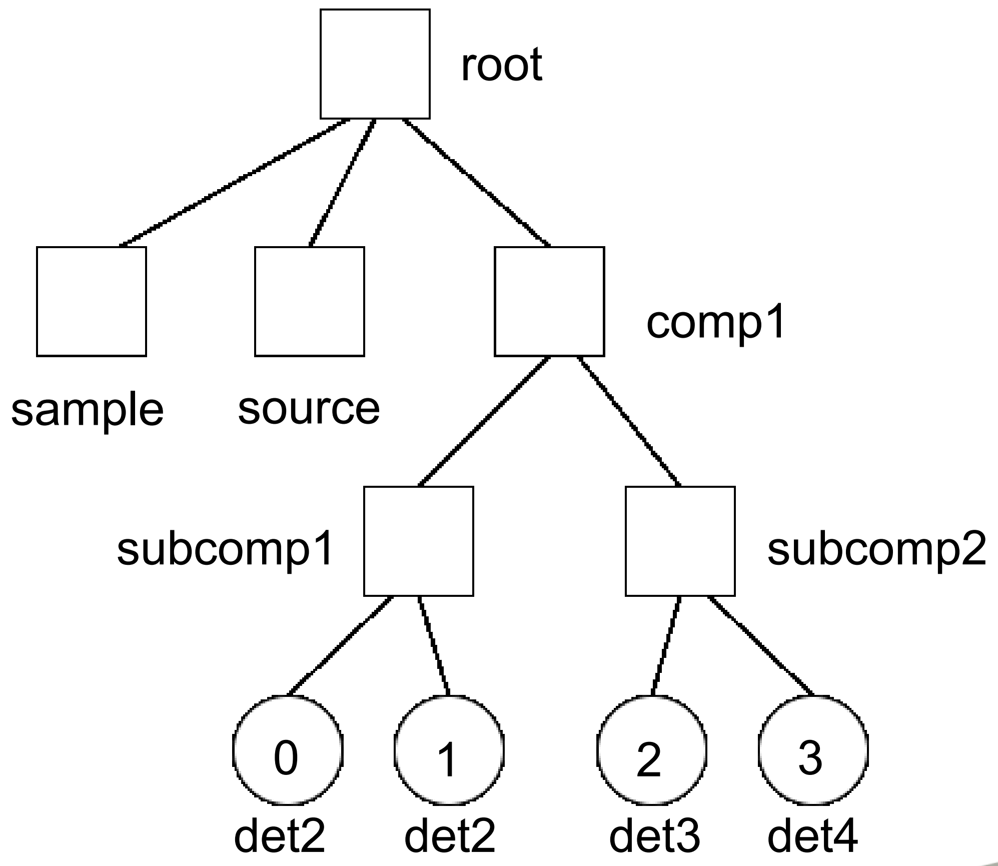
Implementation



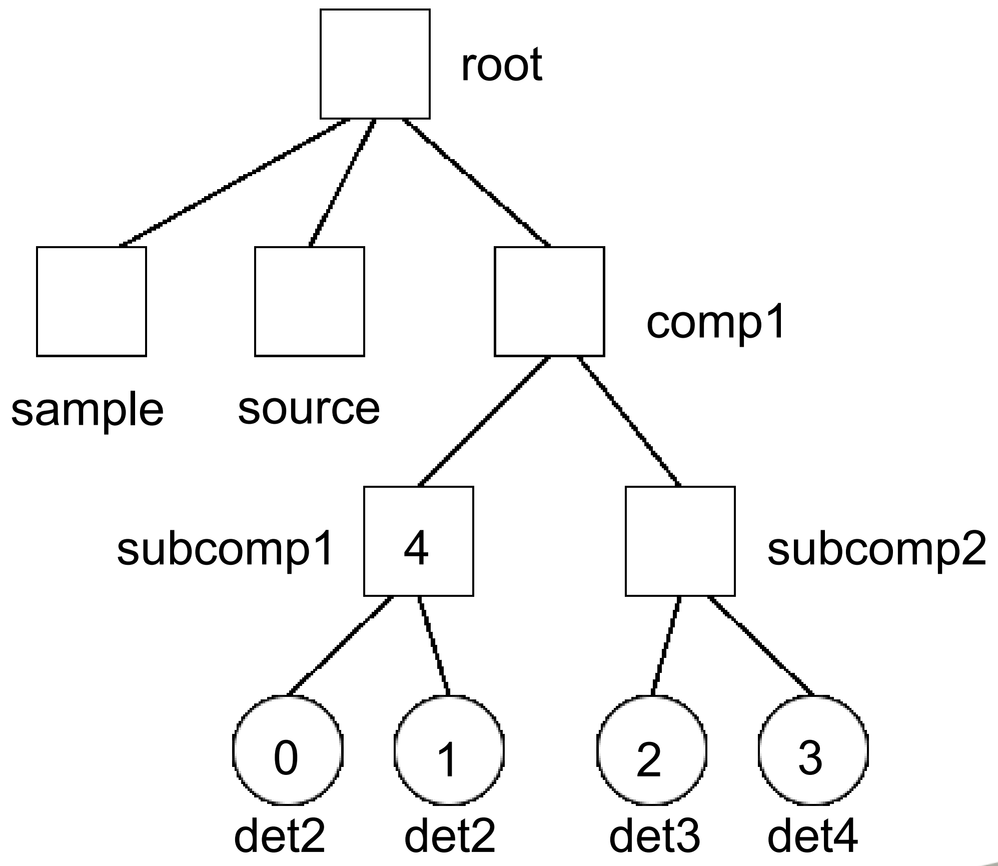
Implementation



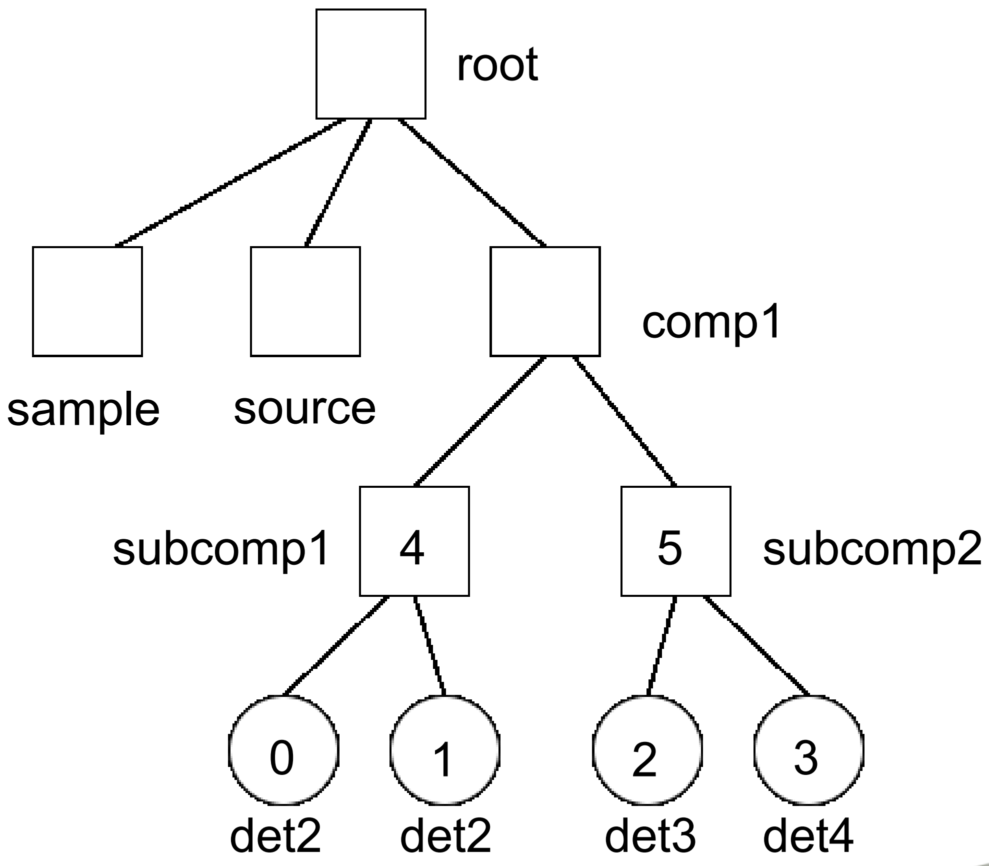
Implementation



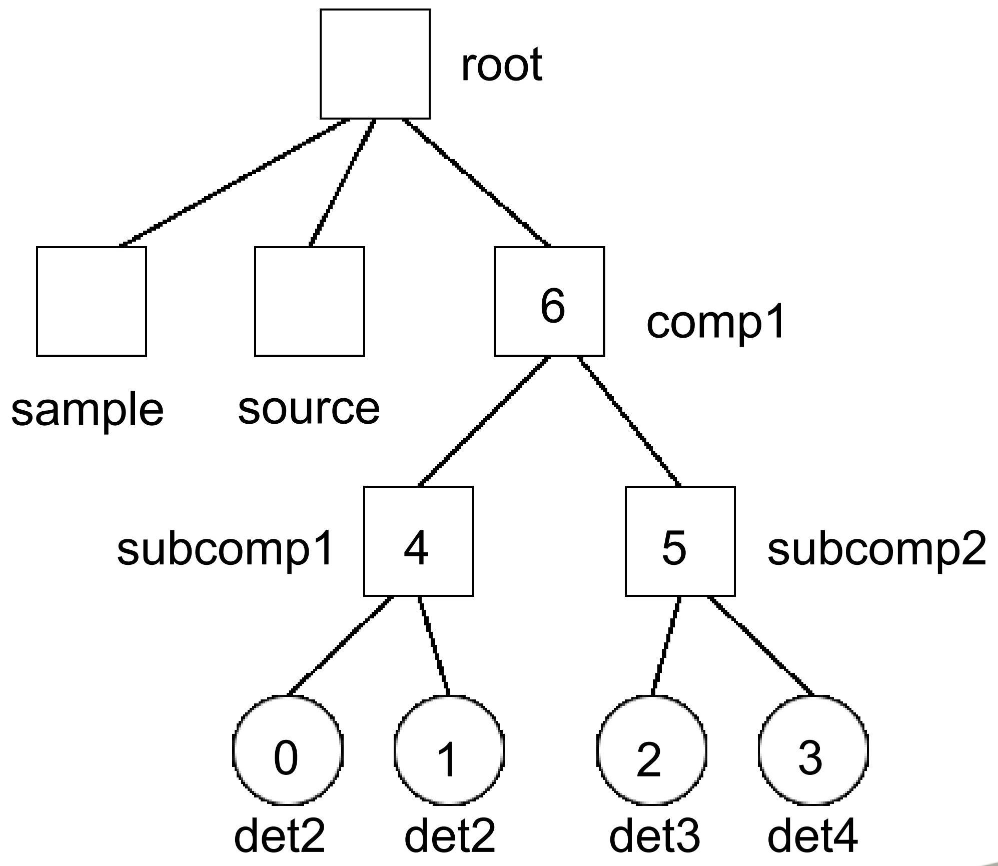
Implementation



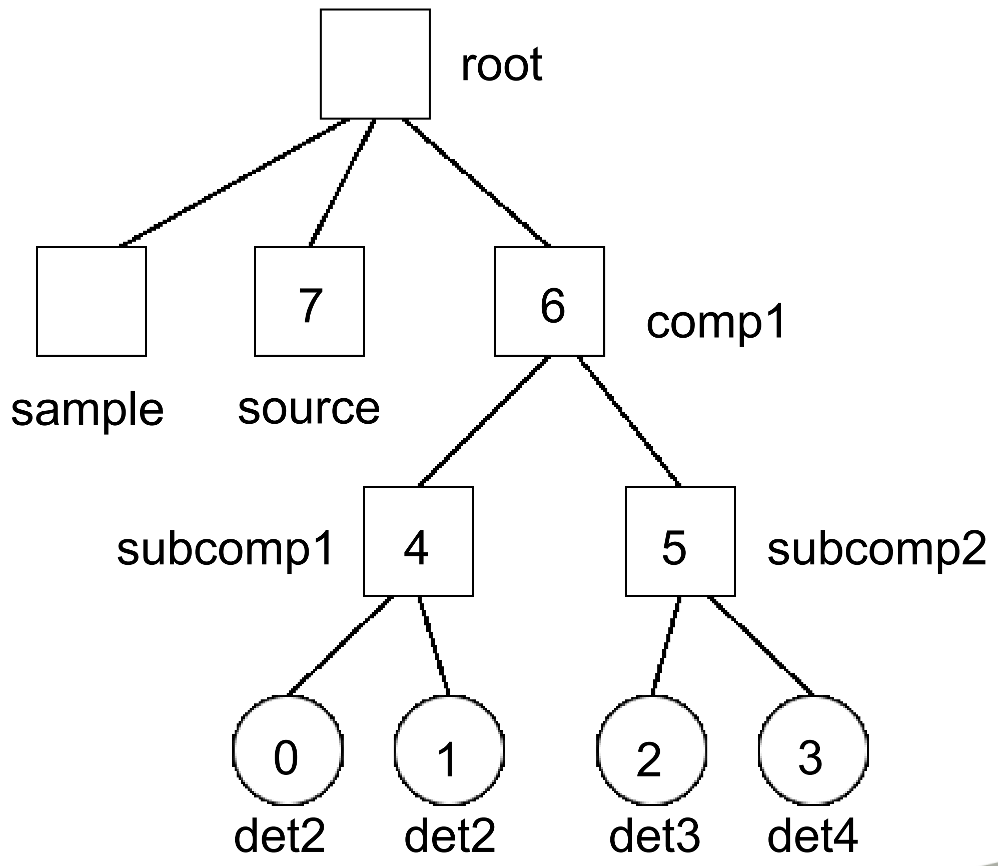
Implementation



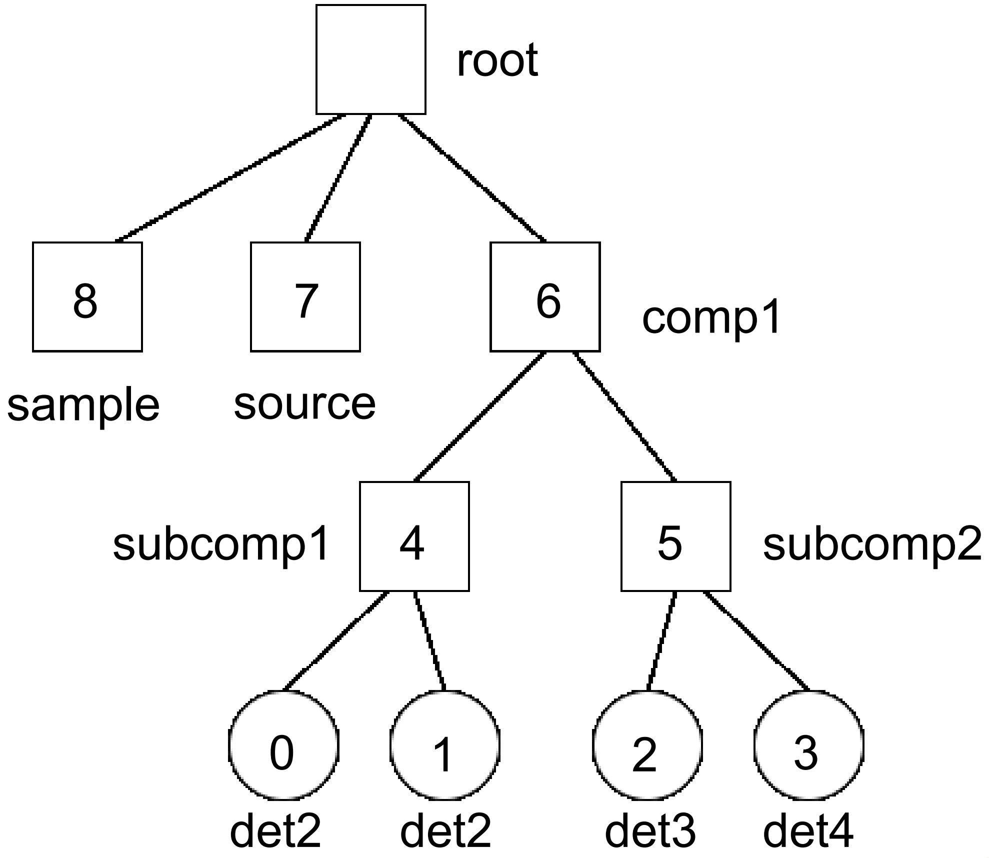
Implementation



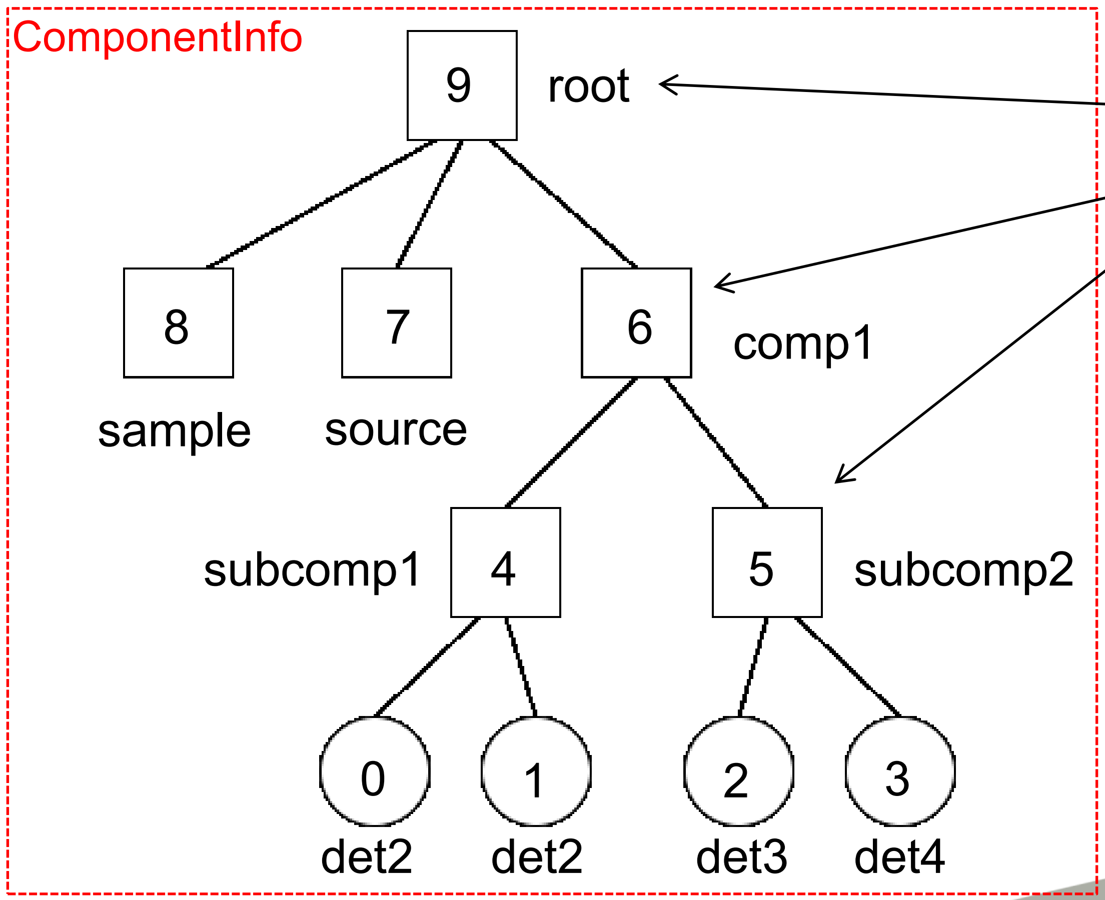
Implementation



Implementation



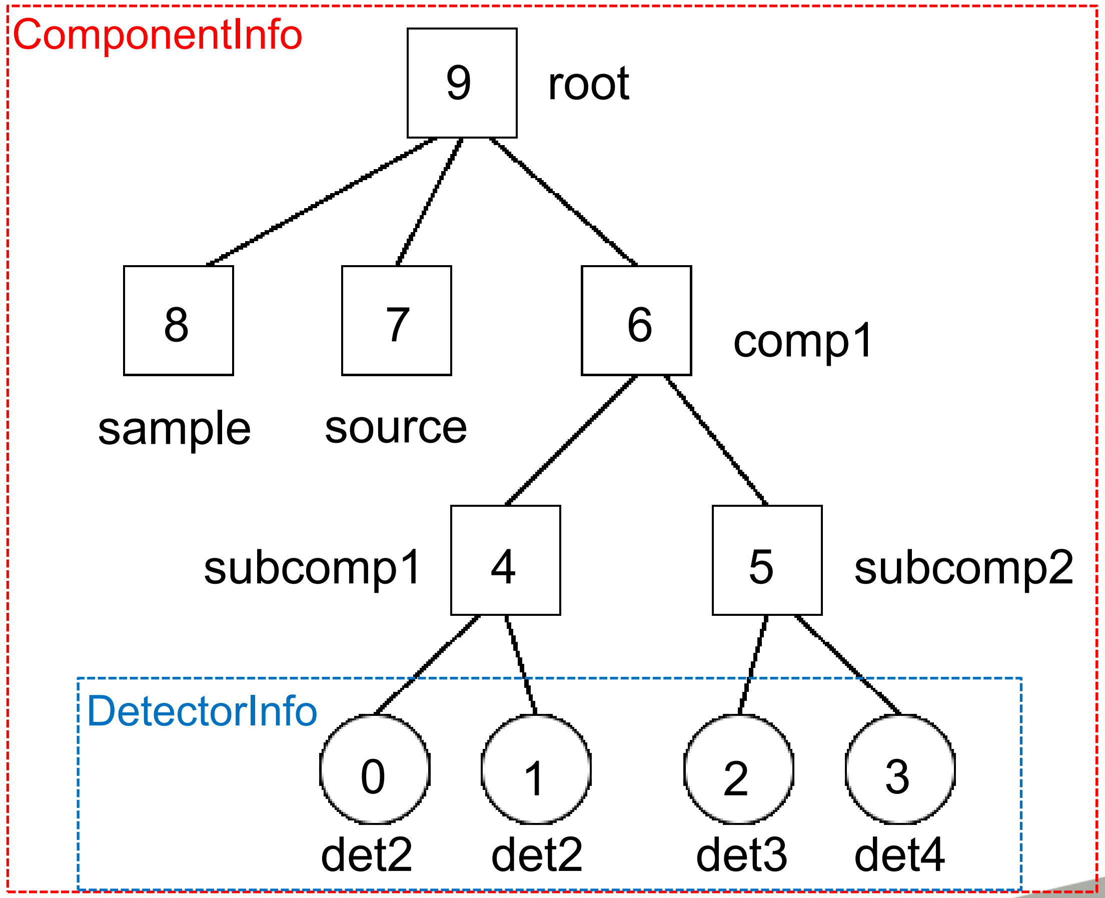
Implementation



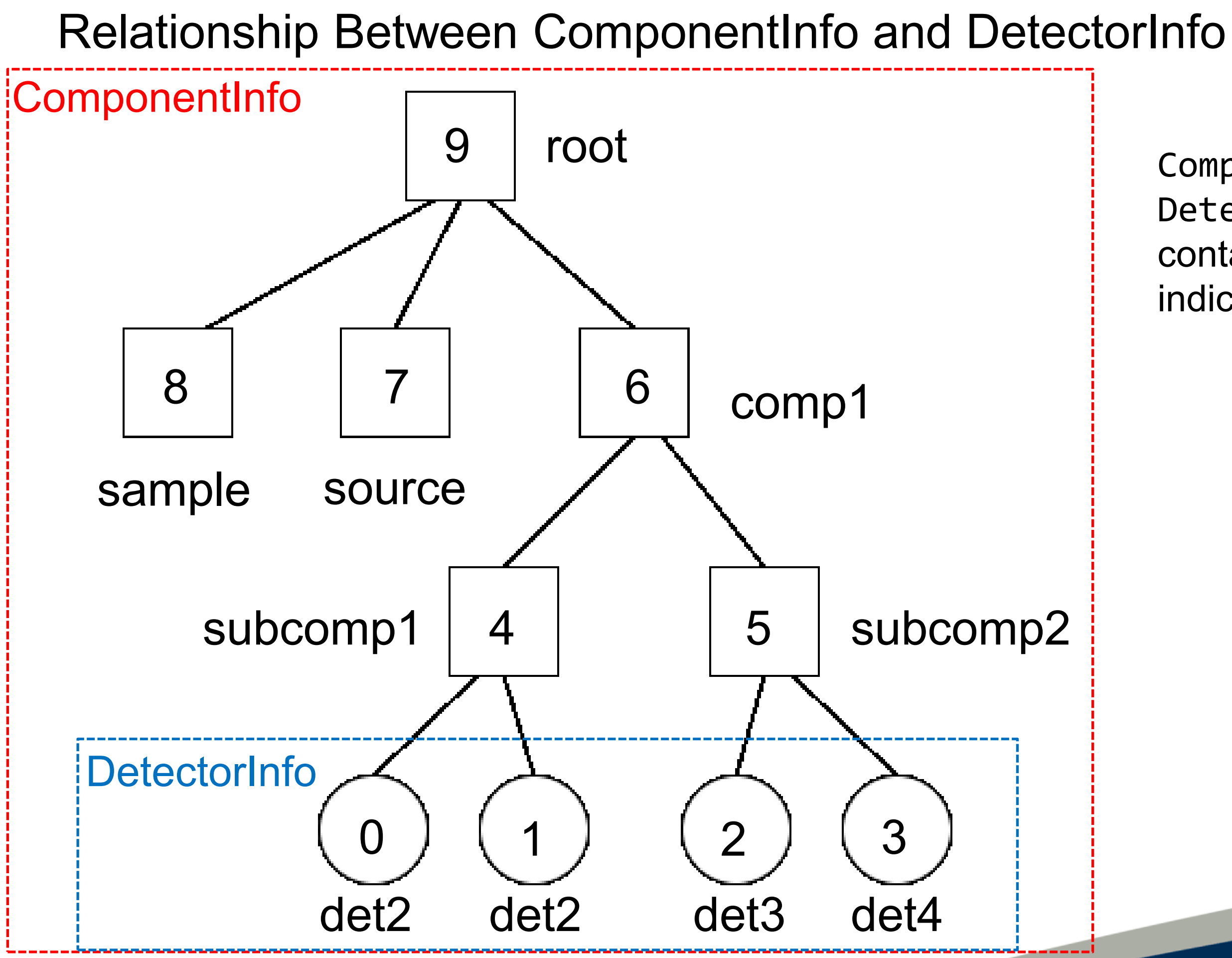
Higher level components always have higher indices than children.

Implementation

Relationship Between ComponentInfo and DetectorInfo



Implementation



`ComponentInfo` and `DetectorInfo` contain the same indices for detectors.

Implementation

- Within the codebase there currently exist 2 layes:

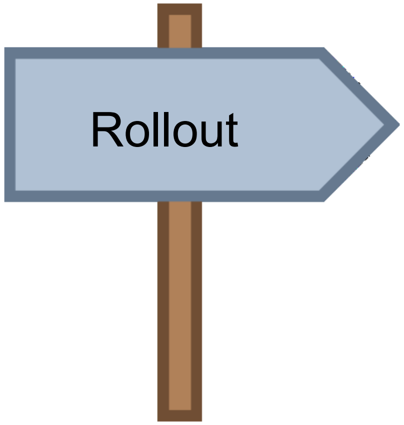
  ```
  Beamline::ComponentInfo
  Beamline::DetectorInfo
  Beamline::SpectrumInfo
  ```

  and

  ```
  Geometry::ComponentInfo
  Geometry::DetectorInfo
  Geometry::SpectrumInfo
  ```

- Currently all `*info` objects are in compatibility mode and are built by parsing Instrument 1.0 in the `InstrumentVisitor`

Rollout



- Due to extensive effort from Simon Heybrock, DetectorInfo rolled out first. Required work with parts of Instrument 1.0 in complex ways. Whole dev team helped upgrade Mantid codebase to use DetectorInfo.

- ComponentInfo rolled out second alongside better mechanisms for Instrument 1.0 -> Instrument 2.0 conversions. Second phase allowed a clean-up for the first.

- Performance test monitoring was performed weekly to ensure changes did not an adverse impact on Mantid performance.

- Minimized issues by delaying merging until the start of dev cycles.

Science & Technology Facilities Council ISIS

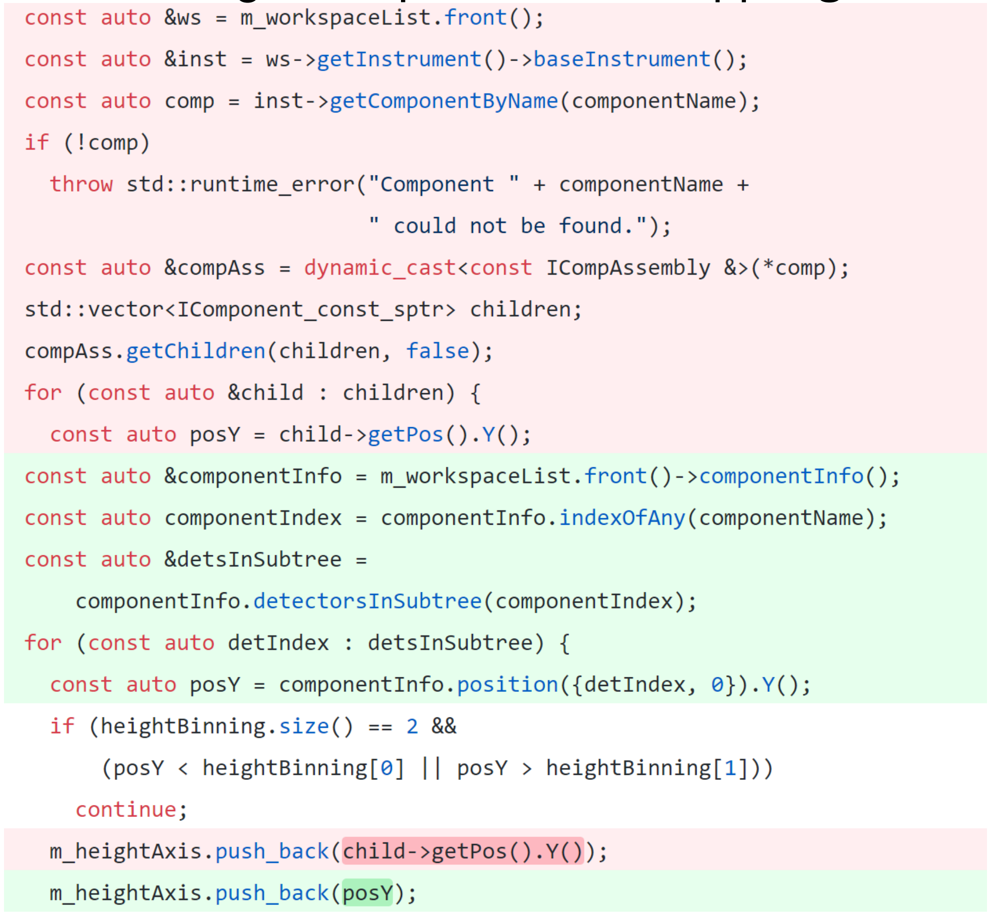
Rollout

- Some performance issues needed to be resolved urgently for users who were dependent on nightly builds.

- Performance related bounding box calculation were solved thanks to efforts by Owen Arnold which required many new optimizations in the face of design challenges.

Rollout

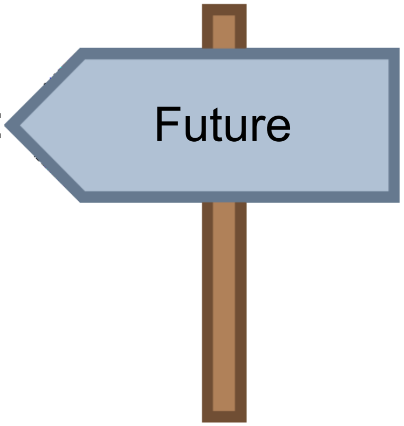
Refactoring Example `SumOverlappingTubes`



```
const auto &ws = m_workspaceList.front();
const auto &inst = ws->getInstrument()->baseInstrument();
const auto comp = inst->getComponentByName(componentName);
if (!comp)
  throw std::runtime_error("Component " + componentName +
                           " could not be found.");
const auto &compAss = dynamic_cast<const ICompAssembly &>(*comp);
std::vector<IComponent_const_sptr> children;
compAss.getChildren(children, false);
for (const auto &child : children) {
  const auto posY = child->getPos().Y();
const auto &componentInfo = m_workspaceList.front()->componentInfo();
const auto componentIndex = componentInfo.indexOfAny(componentName);
const auto &detsInSubtree =
    componentInfo.detectorsInSubtree(componentIndex);
for (const auto detIndex : detsInSubtree) {
  const auto posY = componentInfo.position({detIndex, 0}).Y();
  if (heightBinning.size() == 2 &&
      (posY < heightBinning[0] || posY > heightBinning[1]))
    continue;
  m_heightAxis.push_back(child->getPos().Y());
  m_heightAxis.push_back(posY);
```

Future

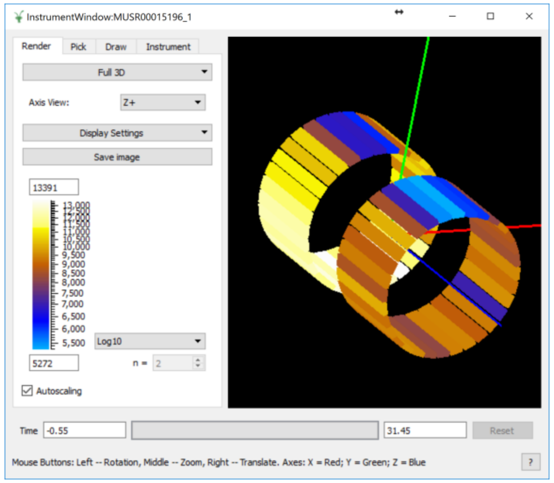
- At this time Instrument 2.0 work has stopped. Some features are now out of scope due to lack of resources:
  - No definition of beam paths.
  - New file formats for representing geometry.
  - Extended IDF aspects.
  - Still need to purge and write parameter map, no new file formats or mechanisms there.
- Compatibility mode must be removed Geometry::*info.



Future

Instrument View

- ESS Requirement for visualisation meant we needed to look into the instrument view performance.
- Survey carried out in 2017 on performance bottlenecks in the Instrument View. Found 30% of loading time spent on `dynamic_casts`. > 50% of time spent in MatrixWorkspace::getInstrument.
- Major refactoring work has started on the instrument view to make use of the new instrument access layers.
- The actor mechanism `ComponentActor`, `ComponentAssemblyActor` etc. have been entirely stripped out.
- Up to 10x loading speed-up so far for large instruments like WISH.


InstrumentWindow:MUSR00015196_1
Render
Pick
Draw
Instrument
Full 3D
Axis View:
Z+
Display Settings
Save image
13391
5272
Log10
n =
2
Autoscaling
Time
-0.55
31.45
Reset
Mouse Buttons: Left -- Rotation, Middle -- Zoom, Right -- Translate. Axes: X = Red; Y = Green; Z = Blue

Science & Technology Facilities Council
ISIS

The End

Science & Technology Facilities Council

ISIS

Questions?