Static **A**nalysis **S**uite

Plans for the static analysis tool

SFT Group Meeting

October 31st, 2016

Joschka Lingemann
Benedikt Hegner, Danilo Piparo

EP-SFT - CERN

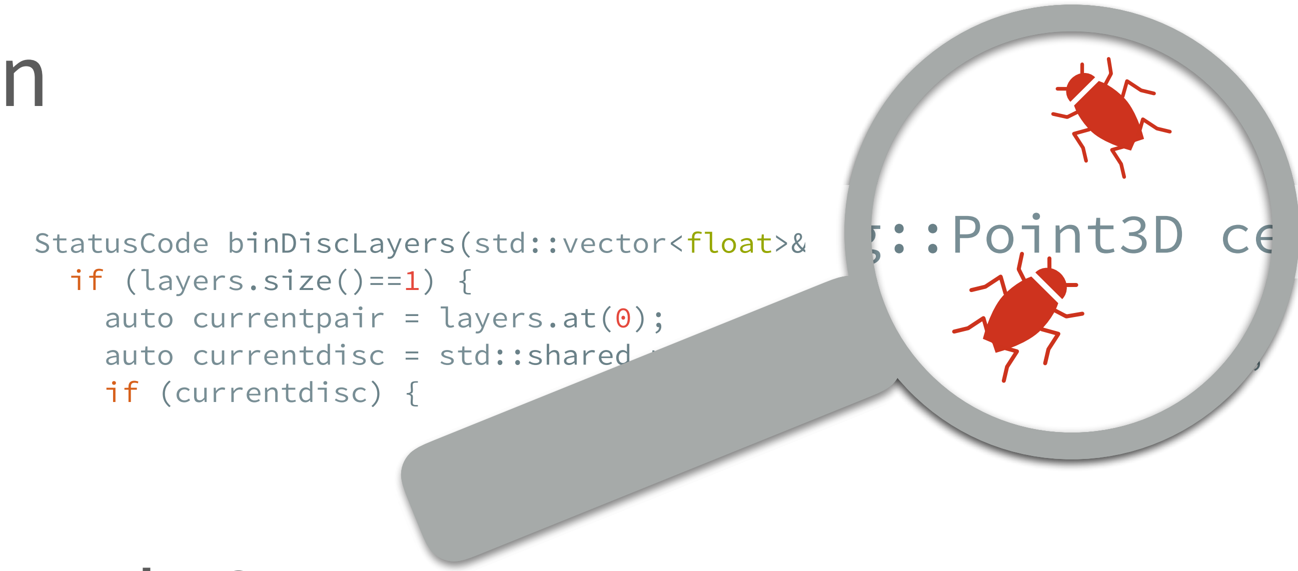
What is SAS?

What do we solve with SAS?

Static code analysis: Checking code without executing it

- i.e. before / during compilation

What does that mean?



```
StatusCode binDiscLayers(std::vector<float>&
  if (layers.size()==1) {
    auto currentpair = layers.at(0);
    auto currentdisc = std::shared
    if (currentdisc) {
```

- Is your code thread-safe?
- Are there performance bottle necks?
- Is new code conform with your coding conventions?
- Are you exploiting the new C++11/14/17 features?

SAS helps to answer these questions!

What is SAS?
A few words on history

Many checkers originate from the CMS software

- First static analysis efforts in CMS 2012

  (T. Hauth et. al.)
- Extracting checkers into standalone tool 2013

  (F. Bartek, D. Piparo)
- Major redesign in 2015 and contributions by GSoC Student
- Adaptation for big software projects (Gaudi / FCCSW, ROOT) 2016

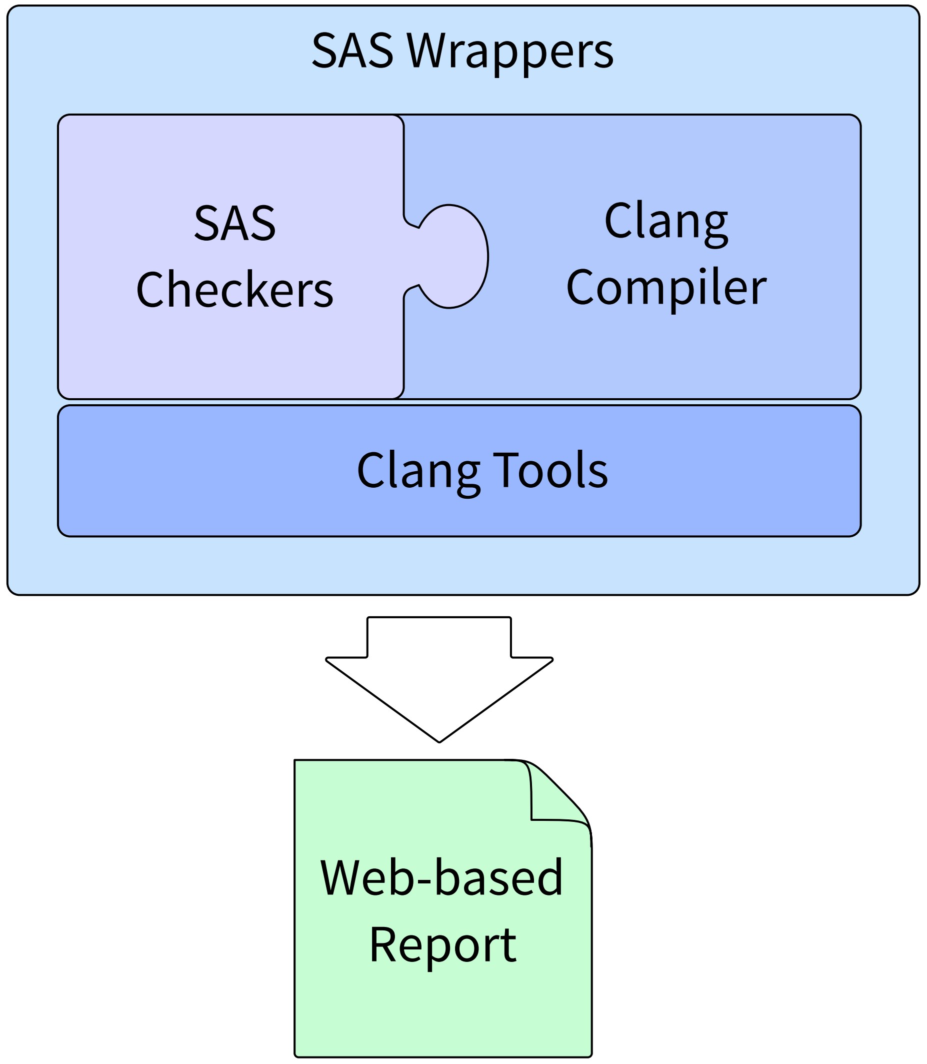
  (D. Ho)

What is SAS?
How we use the clang ecosystem in SAS

Wraps clang **compiler** and clang **tools**

- **compiler** builds abstract syntax tree:
  - full information about your code
  - examine tree in “checkers”
- **tools** for modernisation & formatting
  - use checkers and tools on equal footing

Generate report with all information



Why do you want to use SAS?

Where does SAS improve on the clang static analyser:

- **Convenience:**
  - Easier to write new checkers
  - When using CMake: Use SAS with one command
  - Clang tools + static analysis: Only “compile” once
- **Reporting** mechanism:
  - Projects with dependencies — scan-build is a nightmare
  - Show the **information that is relevant**

Why do you need SAS?

Why choose SAS over commercial solutions?

- Price point:
  - SAS + clang are free open source software
- **Modularity + Open Source:**
  - **Flexibility:** Add project specific checkers
  - Community driven effort to develop more checks
    - Framework to accommodate specific and generic checks
  - Projects can pick and choose what is suitable

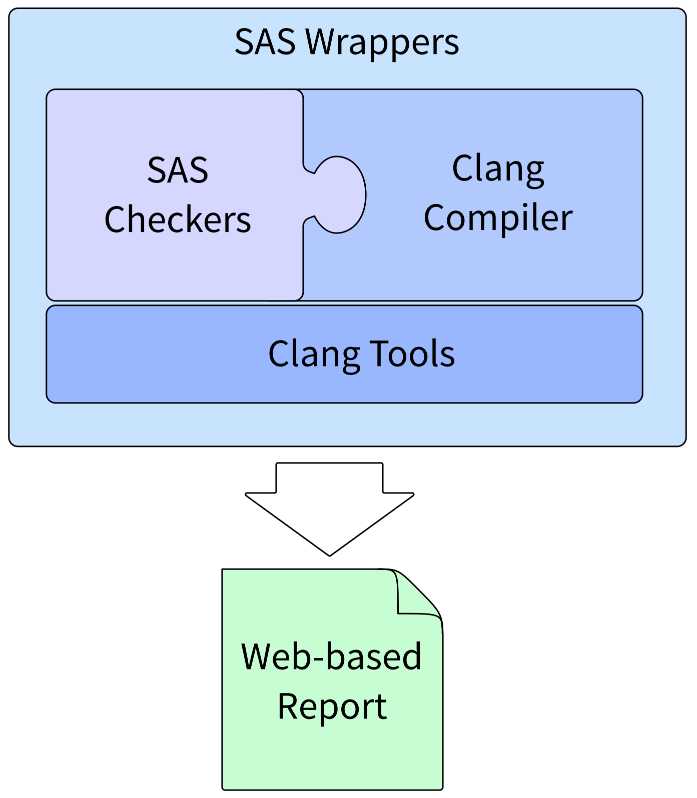
What are the SAS features?
Existing Functionality

Static Analysis

Thread Safety, Performance

False Positives:

- Black / white listing \w comments
- Black / white listing files
- Black listing folders



Coding Conventions

clang-format: formatting

clang-modernize: C++11/14

(clang-tidy soon)

Naming conventions

- test against them
- generate from reg-expr

Running SAS:

- Run on single files using wrappers:
  - Need all compiler arguments
  - Should use compilation database

Scaling:

Demonstrator on ROOT code base,

Integration into Gaudi fully tested

CMake integration

If your project’s build system is CMake based:

- Entry point is a single CMake-function:

```
find_package(sas)
enable_sas([options])
```

Creates new targets:

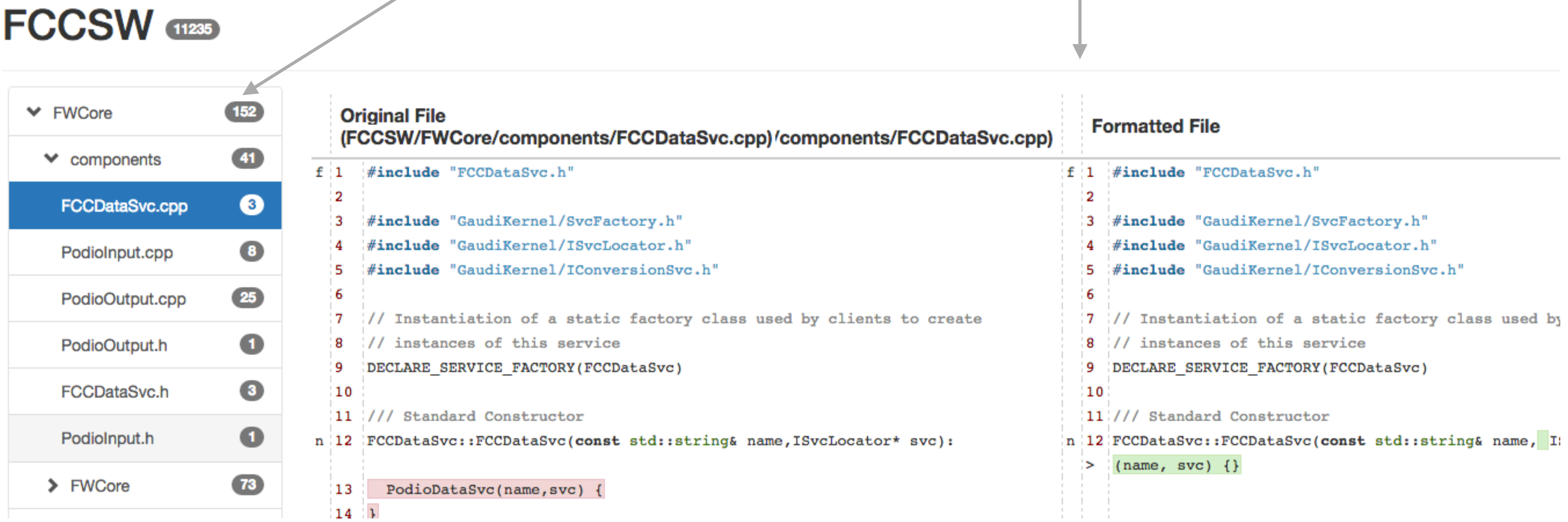
- Apply suggestions by clang-format & clang-modernize
- Generate the report (after compilation)

The SAS Report — an example

project summary

warnings per package (includes formatting)

original vs. suggestion diff



FCCSW 11235

FWCore	152
components	41
FCCDataSvc.cpp	3
PodioInput.cpp	8
PodioOutput.cpp	25
PodioOutput.h	1
FCCDataSvc.h	3
PodioInput.h	1
FWCore	73

Original File
(FCCSW/FWCore/components/FCCDataSvc.cpp)/components/FCCDataSvc.cpp)

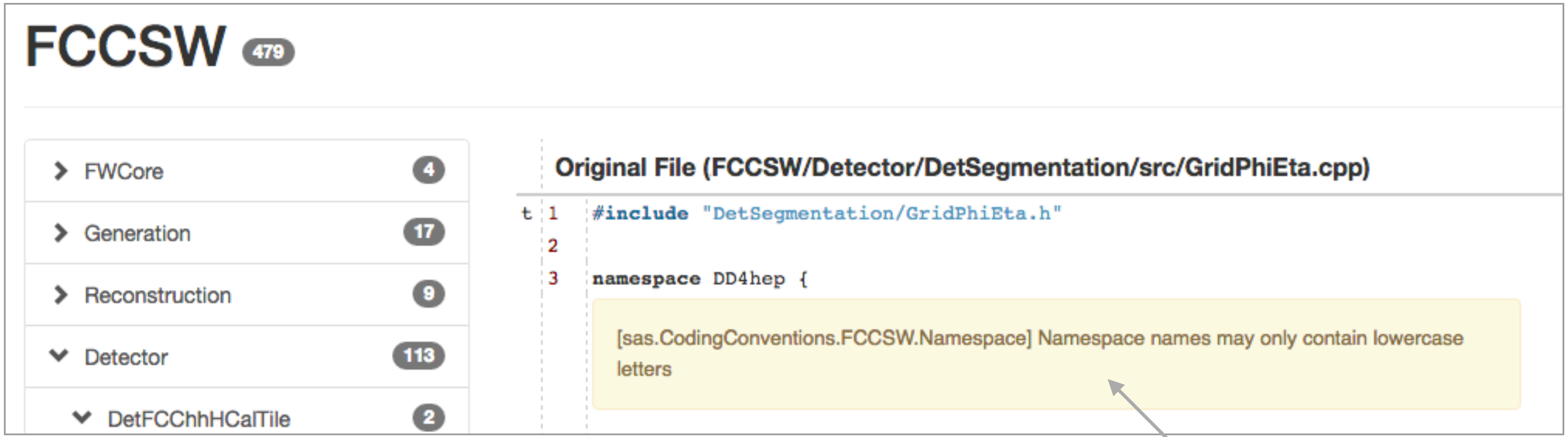
```
f  #include "FCCDataSvc.h"

   #include "GaudiKernel/SvcFactory.h"
   #include "GaudiKernel/ISvcLocator.h"
   #include "GaudiKernel/IConversionSvc.h"

   // Instantiation of a static factory class used by clients to create
   // instances of this service
   DECLARE_SERVICE_FACTORY(FCCDataSvc)

   /// Standard Constructor
n  FCCDataSvc::FCCDataSvc(const std::string& name,ISvcLocator* svc):

     PodioDataSvc(name,svc) {
   }
```

Formatted File

```
f  #include "FCCDataSvc.h"

   #include "GaudiKernel/SvcFactory.h"
   #include "GaudiKernel/ISvcLocator.h"
   #include "GaudiKernel/IConversionSvc.h"

   // Instantiation of a static factory class used by
   // instances of this service
   DECLARE_SERVICE_FACTORY(FCCDataSvc)

   /// Standard Constructor
n  FCCDataSvc::FCCDataSvc(const std::string& name, I
>  (name, svc) {}
```

Let's apply clang-format & -modernize: `make apply`

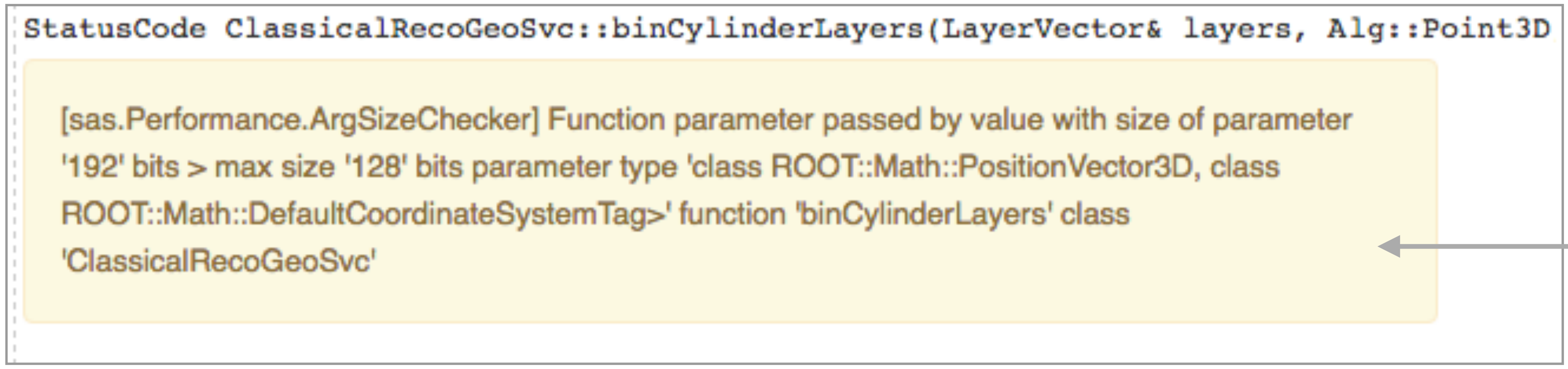
The SAS report — after SAS-apply


FCCSW 479

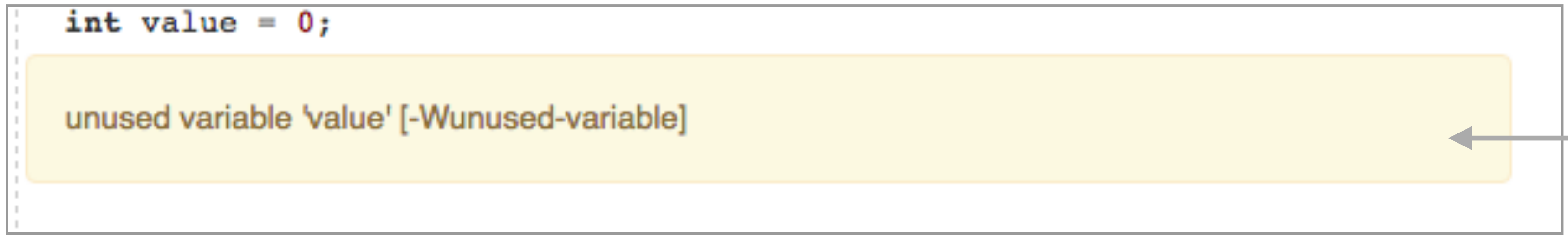
Original File (FCCSW/Detector/DetSegmentation/src/GridPhiEta.cpp)

```
#include "DetSegmentation/GridPhiEta.h"

namespace DD4hep {
```

[sas.CodingConventions.FCCSW.Namespace] Namespace names may only contain lowercase letters


```
StatusCode ClassicalRecoGeoSvc::binCylinderLayers(LayerVector& layers, Alg::Point3D
```

[sas.Performance.ArgSizeChecker] Function parameter passed by value with size of parameter '192' bits > max size '128' bits parameter type 'class ROOT::Math::PositionVector3D, class ROOT::Math::DefaultCoordinateSystemTag>' function 'binCylinderLayers' class 'ClassicalRecoGeoSvc'


```
int value = 0;
```

unused variable 'value' [-Wunused-variable]

conventions

performance checks

compiler warnings

Things we'd like to do

SAS improves existing functionality of clang

- Lots of ideas to improve usability
  - More interactivity in the web-report
  - More flexibility in configuring SAS for your project
  - More ways to run SAS (e.g. use compilation database)
- Upcoming clang API changes and tool deprecations need to be accounted for

We could try and re-integrate this into clang

- Doing that requires more polish
  - Collect "internal" feed-back - find more HEP / CERN users first

However development so far as a side project

Ideas on how to achieve that

Need somebody who works on the project full-time

- At least for a couple of months

Looking for students via:

- Knowledge transfer: “promote innovative work […] that have potential applications outside CERN”
- Portuguese trainee program: Recently graduated students from technical fields of study

Why knowledge transfer & Portuguese trainee programs?

Static code analysis is also **useful outside CERN**:

- Many commercial solutions:
  - Expensive licenses - Not viable for small companies
- A few open source solutions
  - Most have limited functionality

    (e.g. treat code as plain text and use regular expressions)

We think SAS is filling a gap:

- Extend clang analyser to be more useful (could be re-integrated)
- Especially with customisability - Interesting for smaller projects

Feedback greatly appreciated:

dpiparo/SAS

fccsw.web.cern.ch

Additional Material

What are the SAS features?
Existing Checker Functionality

Thread Safety

- Avoid casting away constness
- Discourage usage of non-const static variables
- Flag mutable members for further investigation

Code Performance

- Check size of arguments passed by value
- Encourage usage of fast math

Common Sense

- no “using namespace std”, please
- Discourage usage of std::cout