Static Analysis on HEP Software

Axel Naumann, CERN

ACAT 2011, Brunel

Overview

- Static Analysis
- Typical coding issues
- Diagnosis
- Conclusions

Static Analysis

Static versus Dynamic

- Dynamic analysis looks at runtime behavior, e.g. valgrind, sees what *does* happen
- Static analysis looks at source code: sees what *could* happen

Features

- Cross-function / module / developer
- Not test-driven

```
void receive() {
  if (!m_Server->receive())
    recover(m_Server);
  logger(m_Server->Name(), “receive”);
}
```

```
void recover(Server* S) {
  cout << “ERROR!\n”;
  delete S;
}
```

Features

- Independent of developers' assumptions

```
void f(int* p) {
  g(p);
  if (p) *p = 12;
}
```

```
void g(int* p) {
  *p = 0;
}
```

Features

- Builds condition matrix, tracking depth beyond human capabilities

```
void p(int flag) {
  if (flag > 2) {
    ...
    return;
  }
  int flag1 = flag * 2;
  if (flag1 < 10) {
    ...
  } else {
    // Algorithm that a physicist
    // worked on for two years
  }
}
```

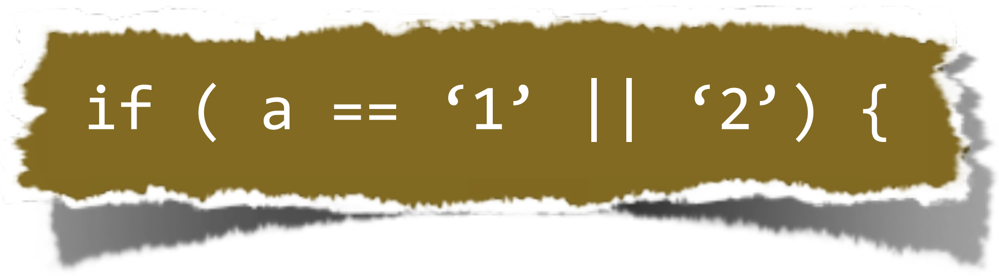
Features

- >4000 issues found in ROOT
- Some systematic, motivate systematic remedies

```
char buf[1024];
strcpy(buf, getenv(“PATH”));
```

Features

- A humans brain always interprets what it reads without seeing what is really written, and that holds even when reading some some snippet of regular text

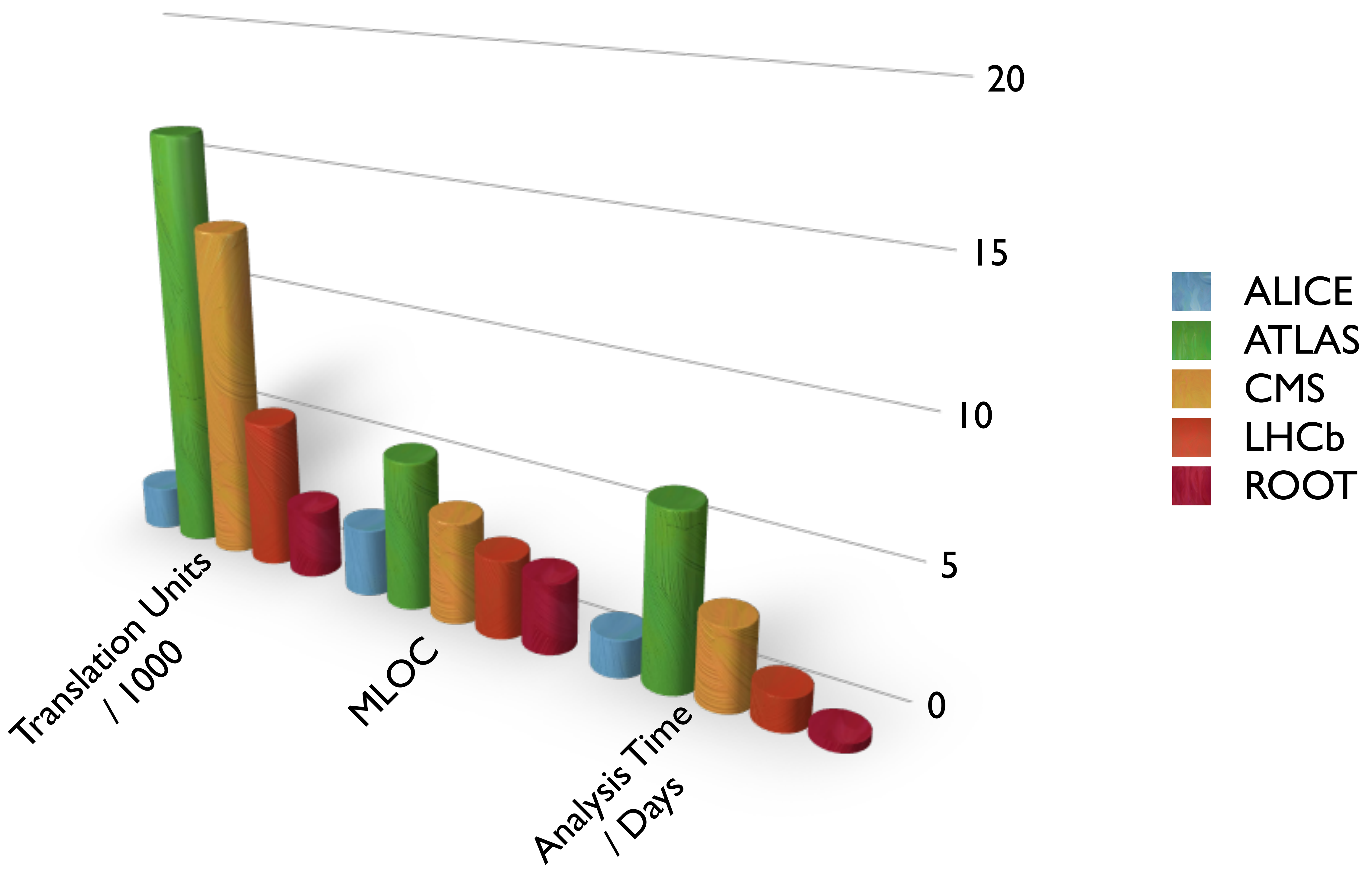


```
if ( a == ‘1’ || ‘2’) {
```

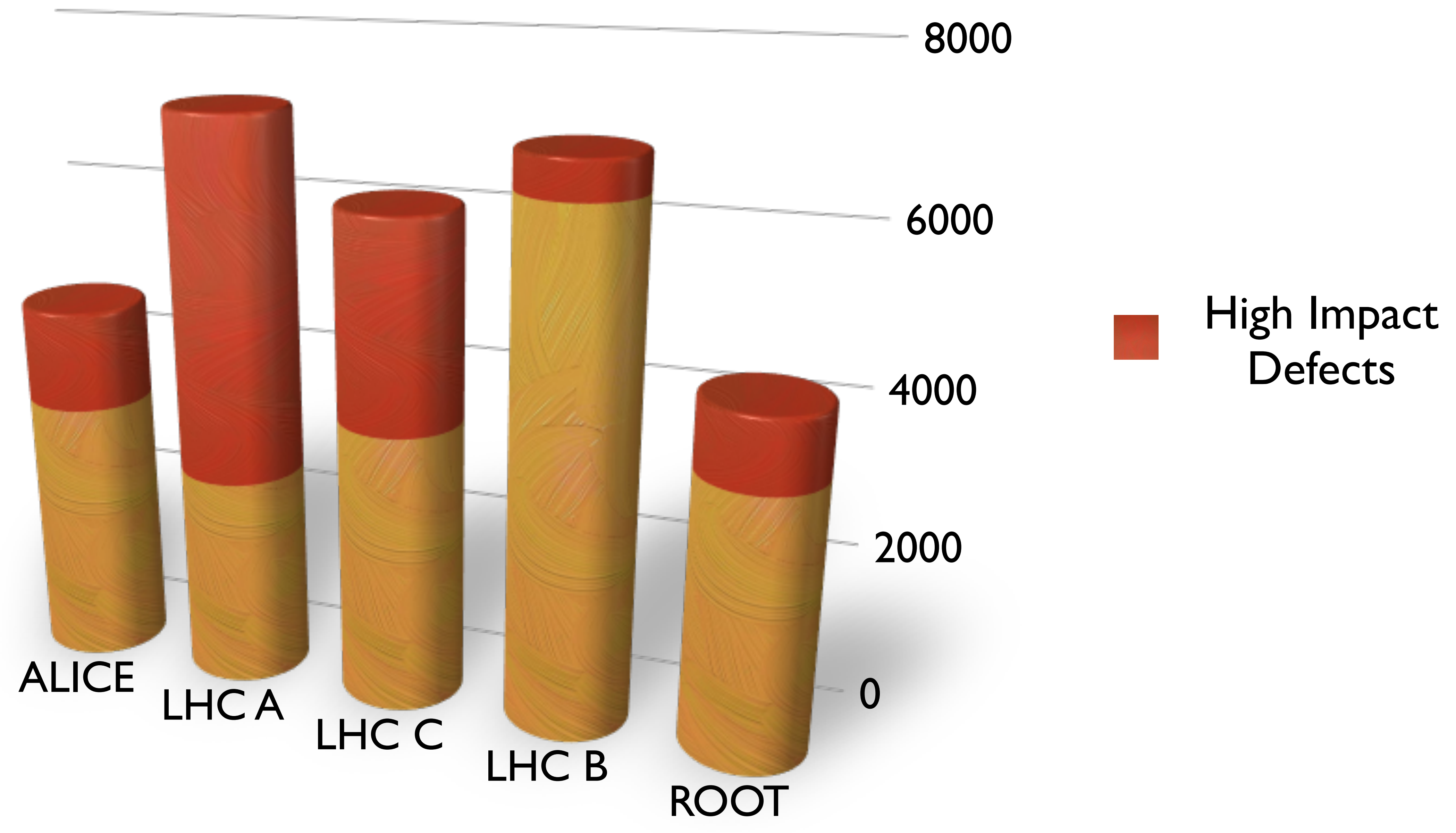
Tools

- CERN uses proprietary, donated Coverity
- Free alternatives exist, e.g.:
  - clang
  - checkcpp
  - cpplint

Analysis Time



First Reports



Typical Issues

Memory

- Possible buffer overflows
- Use after delete
- Uninitialized values
- Null pointer checks

Flow

- Missing break
- Undocumented intentionally missing break
- Logic flaws: if (the impossible)
- Misspelled conditions: if (a & b == c)
- Code path issues: for / if / break giving e.g. invalid array index

API

- Called function deletes, caller surprised
- Called function allocates, caller surprised
- Called function expects pointer != 0
- Function requires check of return value

Diagnosis

C++

- C++ is too complex for us

Medication

- Simple, clear, documented API helps
- No pointers
- At least no bare pointers: owning_pointer<TH1>
- Expect the unexpected: uneducated callers, context out of your control (“reuse”)

Long-Term Treatment

- In the end: nothing helps
- static analysis is an integral, irreplaceable part of Q/A tool set

Conclusions

- C++ is too difficult (but python too slow)
- Coding is too difficult
- Need big brother watching your code:
  - systematic testing
  - commit-centric feedback (“your change”)
  - automatic analysis, static *and* dynamic