Ajouter une autre dimension a la qualité des logiciels

Exemple de mise en oeuvre d'un outil d'analyse statique

Rene Brun, CERN

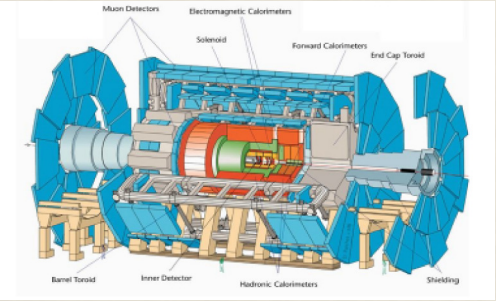
CERN

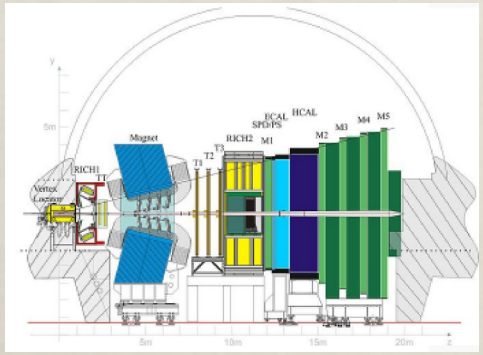
- Organisation Européenne pour la Recherche Nucléaire (Physique des particules): Frontière Franco-Suisse près de Genève
- Recherche fondamentale, mais aussi avec des retombées pratiques (électronique, vide, cryogénie, scanners et le WEB
- Comprendre le big-bang et l'évolution de l’univers

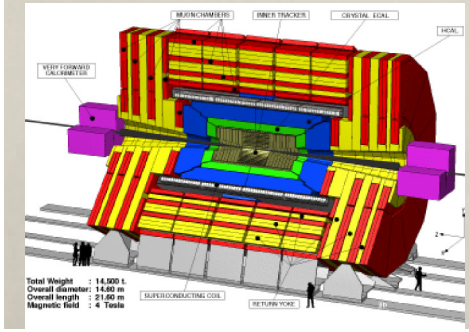
Le LHC
(Large Hadron Collider)

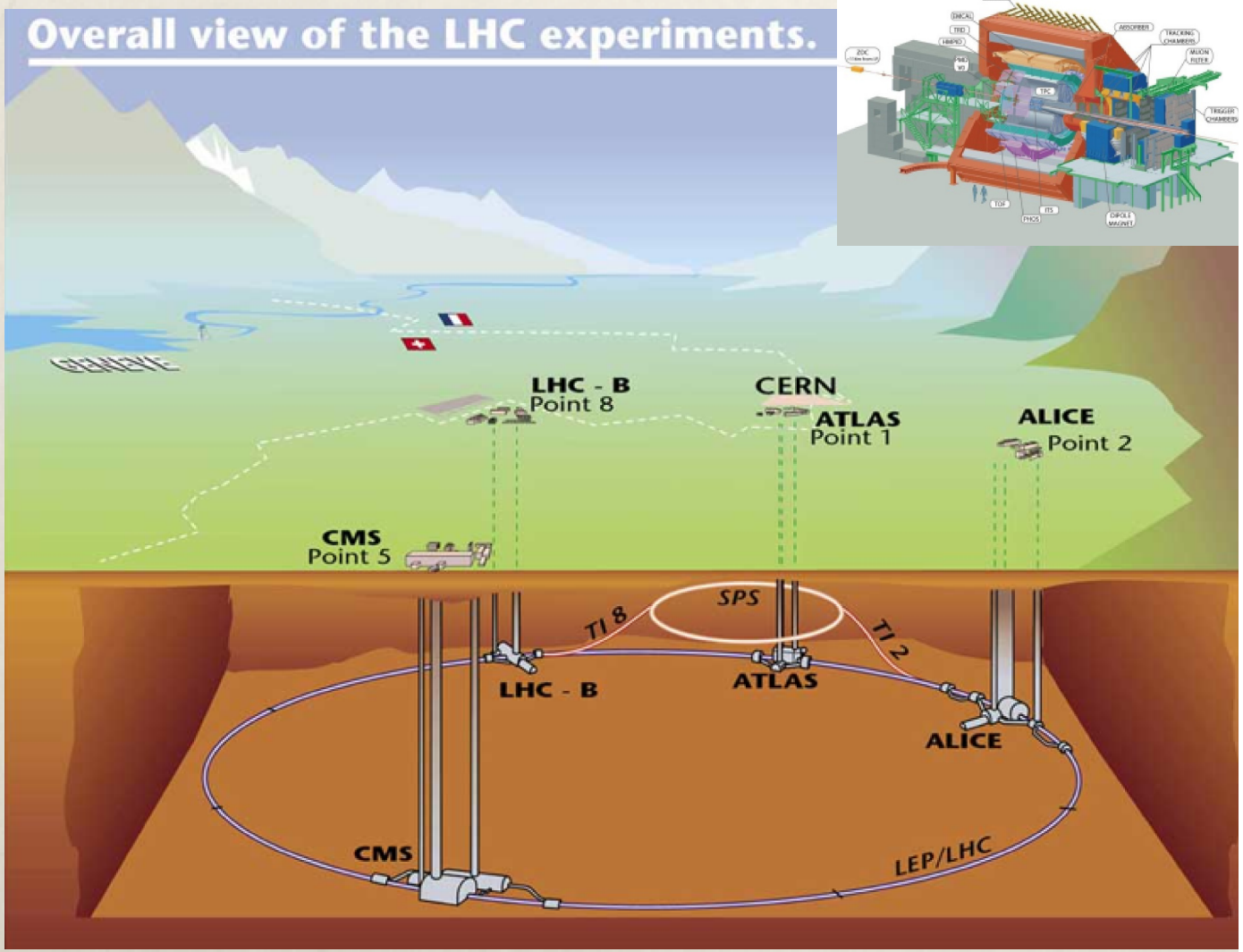
- Un tunnel de 28km a 100m sous terre a la frontiere FR/CH
- Collisioneur de protons ou d'ions lourds
- 10000 physiciens

LHC


Muon Detectors
Electromagnetic Calorimeters
Solenoid
Forward Calorimeters
End Cap Toroid
Barrel Toroid
Inner Detector
Hadronic Calorimeters
Shielding


RICH1
Magnet
RICH2
ECAL
HCAL
M1
M2
M3
M4
M5


MUON CHAMBERS
INNER TRACKER
CRYSTAL ECAL
HCAL
SUPERCONDUCTING COIL
RETURN YOKE
Total Weight : 14,500 t.
Overall diameter : 14.60 m
Overall length : 21.60 m
Magnetic field : 4 Tesla


Overall view of the LHC experiments.
GENEVE
LHC - B
Point 8
CERN
ATLAS
Point 1
ALICE
Point 2
CMS
Point 5
SPS
TI 8
TI 2
LHC - B
ATLAS
ALICE
CMS
LEP/LHC

CERN en chiffres

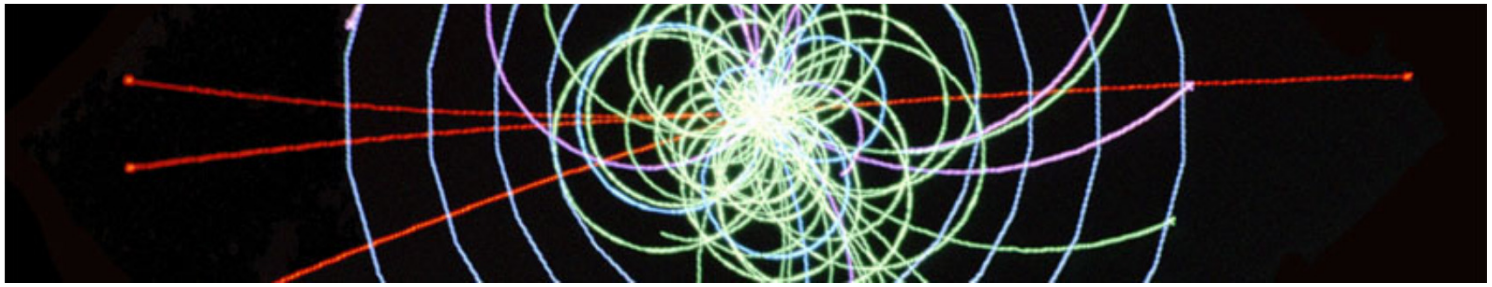
- Utilisé par 10000 physiciens de 608 universités dans 113 pays.
- Plus de 1000 physiciens par expérience
- Une expérience peut durer 40 ans!
- The LHC génère 15,000,000 Gigabytes de données par an.
- Budget: 1 milliard de CHF/an

Informatique au CERN

- Approx 50 MLOC C++ par les physiciens
- Beaucoup de données: code doit être rapide
- The code doit être correct: résultats *scientifiques*;
- Doit être stable: 1 Higgs / 10,000,000,000,000 collisions;

Logiciels au CERN

- Chaque expérience développe et maintient ses propres logiciels (2/3 du total environ).
- The CERN développe et maintient des outils généralistes communs a toutes les expériences: ROOT: math / statistiques, stockage et management des données, simulation de détecteurs, graphiques, GUI, calculs en parallèle.



Logiciels des expériences

- Plusieurs centaines de developpeurs/expérience: physiciens, quelques experts logiciels; actifs pour quelques années seulement (beaucoup de PostDoc).
- Développement très souvent anarchique.
- Beaucoup de dépendances sur des logiciels externes.

ROOT

- http://root.cern.ch
- 7 Developpeurs @ 2.6 MLOC
- Début en 1995, 90% du code avant 2005. Doit être maintenu sur un grand nombre de systèmes durant au moins 30 ans.
- Deux releases / an
- “Core project”, utilise par toutes les expériences/ physiciens y compris en dehors du domaine du CERN: 95000 adresses IP distinctes/an.

Software Risks

- Maintenance
- Implosion of huge software stack
- Integrity of years of LHC experiments' data
- Correctness of scientific results
- Failing searches for new physics
- *Note*: driven by reality, not ISO xyz

Challenges For QA

- C++
- anarchy
- few experts
- multi-platform
- huge programs
- part-time coding
- input data volume
- software as a tool
- interpreter binding
- lack of reproducibility

QA Requirements

- Results must be easy to understand: novices, spare time coding, complex code
- Results must be relevant: motivate, don't flood, developers
- QA must handle rare code paths: don't hide the Higgs
- QA as part of nightly / continuous testing

QA Toolbox

- Patch review (ROOT)
- Continuous integration (ROOT)
- Multi-Platform: compilers, versions, architectures
- Unit + functionality test suites 100k LOC for ROOT
- Regression tests on performance, results

QA Toolbox (2)

- Nightly builds + tests + automatic reports
- Memory: valgrind, massif, ROOT, igprof...
- Performance: callgrind, shark, gprof...
- Test coverage: gcov
- Coding rule checkers
- Automation + strong software management

Memory QA

- C++ inherent: memory errors
  - use after delete
  - uninitialized pointers
  - buffer overruns
  - memory leaks
- Amplified by novice coders, non-defined ownership rules,...

Performance QA

- Huge amounts of data, limited computing, thus performance sensitive
- Difficult to predict: caching, templates, inlining, call of external libraries
- Monitoring difficult to present, interpret
- Using advanced tools + techniques to handle >10 MLOC in a novice-friendly way

QA Toolbox: Full?

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

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

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

```
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
}
```

```
if ( a == '1' || '2') {
```

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

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

Open QA Issues

- Those *almost* impossible code paths!
- Problems that cannot be reproduced
- Too many code paths for humans
- Condition matrices
- “You know what I mean” coding

Static Code Checkers

- Check what *could be* run, not what *is* run
- CERN did a study of open source options

- /media/thomas/userprovideddata/mato/root/build/rmkdepend/include.c:295: **[4]** (buffer) *strcpy: Does not check for buffer overflows when copying to destination. Consider using strncpy or strlcpy (warning, strncpy is easily misused).*
- /media/thomas/userprovideddata/mato/root/build/rmkdepend/include.c:316: **[4]** (buffer) *sprintf: Does not check for buffer overflows. Use snprintf or vsnprintf.*
- /media/thomas/userprovideddata/mato/root/build/rmkdepend/main.c:553: **[4]** (buffer) *strcpy: Does not check for buffer overflows when copying to destination. Consider using strncpy or strlcpy (warning, strncpy is easily misused).*
- /media/thomas/userprovideddata/mato/root/build/rmkdepend/main.c:704: **[4]** (buffer) *sprintf: Does not check for buffer overflows. Use snprintf or vsnprintf.*
- /media/thomas/userprovideddata/mato/root/build/rmkdepend/main.c:752: **[4]** (format) *vfprintf: If format strings can be influenced by an attacker, they can be exploited. Use a*

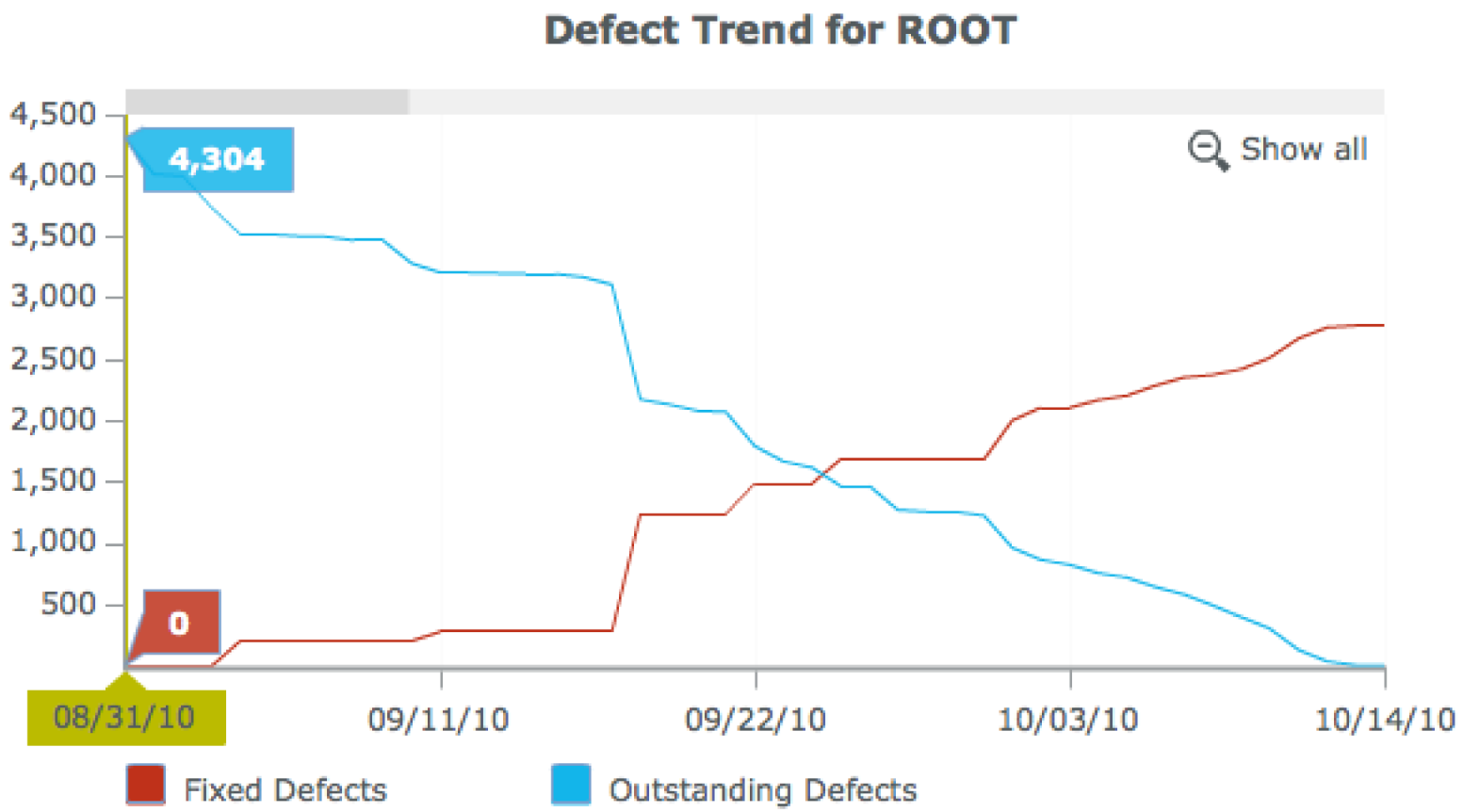
Coverity @ CERN

- ROOT traditionally (informally) spearheads new technologies at CERN: what works for ROOT (and scales!) works for CERN
- Realized lack of static analysis at CERN
- Discovered Coverity through famous Scan: Linux, Python, Apache, glibc, openSSL,...
- ROOT as CERN's guinea pig project

ROOT Meets Coverity

- Plan:
  - Product discovered by a team member
  - Convince ROOT developers
  - Convince CERN
- Set up as “idle time project” in 4 weeks: build integration, automation, reporting
- 2.6 MLOC: 4 processes * 8 hours

First Coverity Results

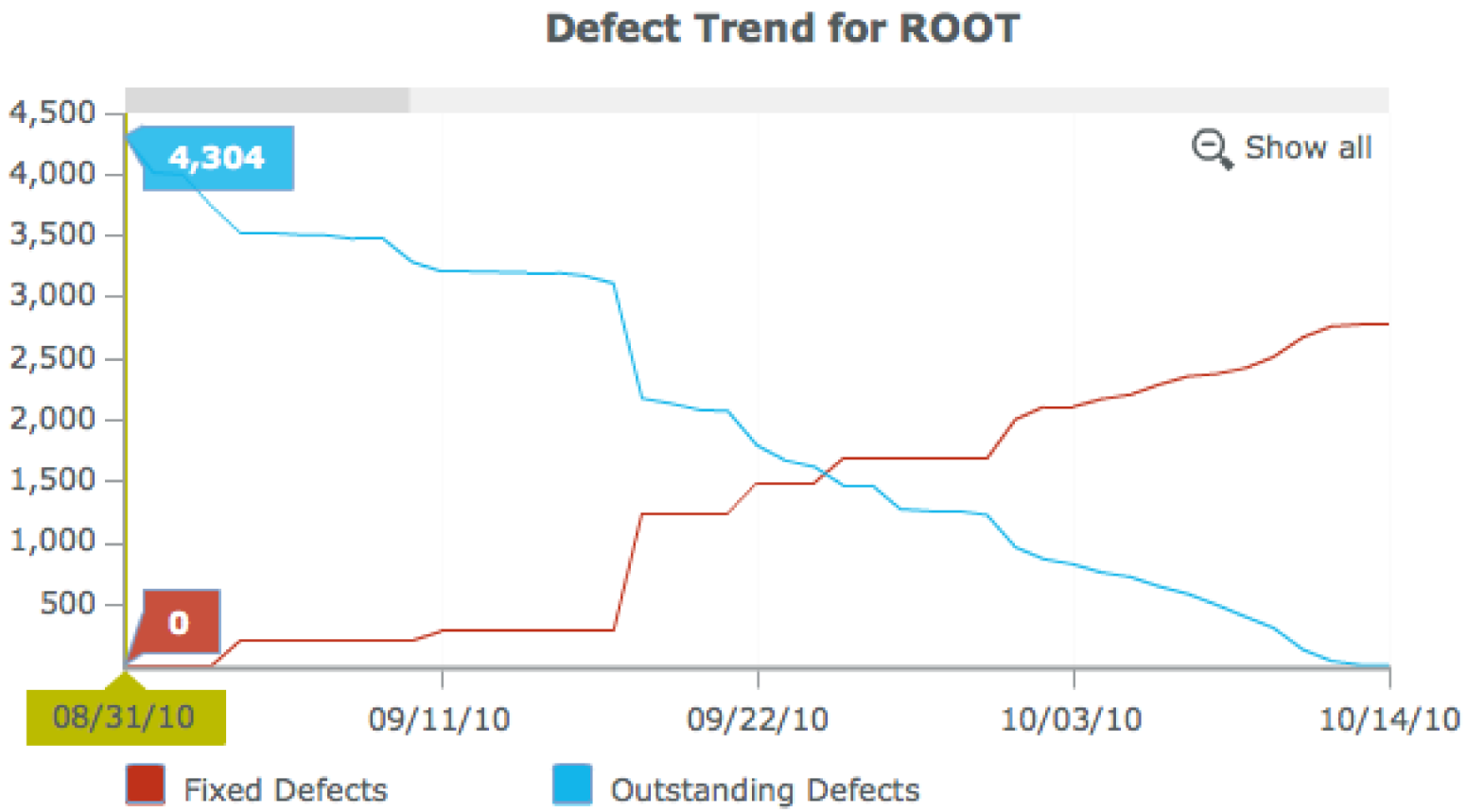


Report Quality

- Very readable reports embedded in source
- Mostly easy to understand, though complex contexts make complex reports
- Amazing Relevance: 12% false positives, 23% intentional thus 2/3 reports causing source changes - despite an already well-filled QA toolbox

Consequence

- Relevant reports create motivated developers!



Results With ROOT

- 4300 reports by 7 developers in 6 weeks
- Noticeably better release
  - Instability due blind changes, e.g. “missing break”
- Found several long-hunted bugs, e.g. hard to reproduce or track down
- Systematic deficiencies triggered systematic fixes: buffer overflows / strlcpy

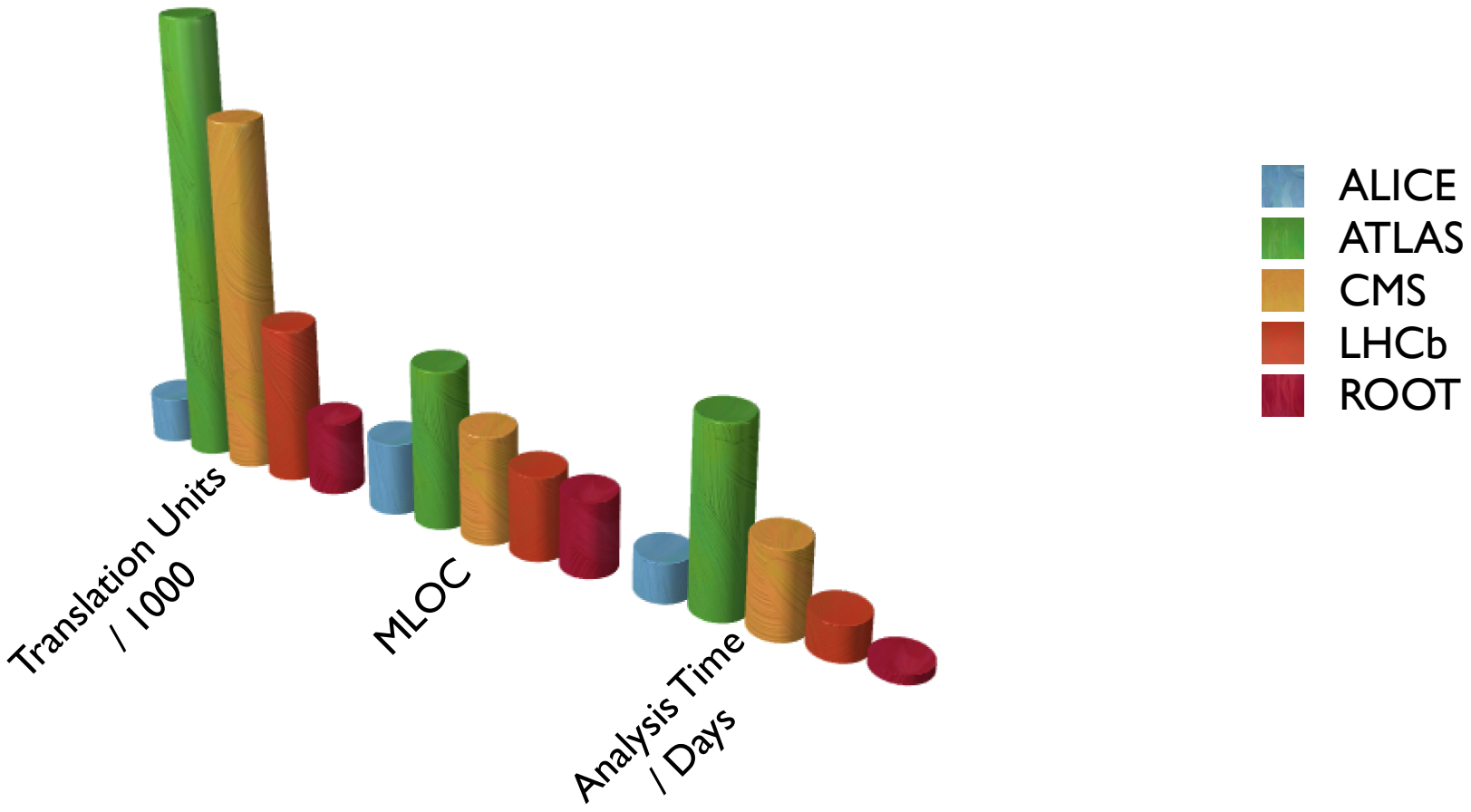
Coverity On 50MLOC

- Excellent show-case for experiments
- Immediate interest:
  - Allows for precise, relevant, accessible, large-scale, automated, quick turn-around reports for software management
  - Classification in high / low impact helps fixing critical reports first
  - Increased awareness of software quality issues

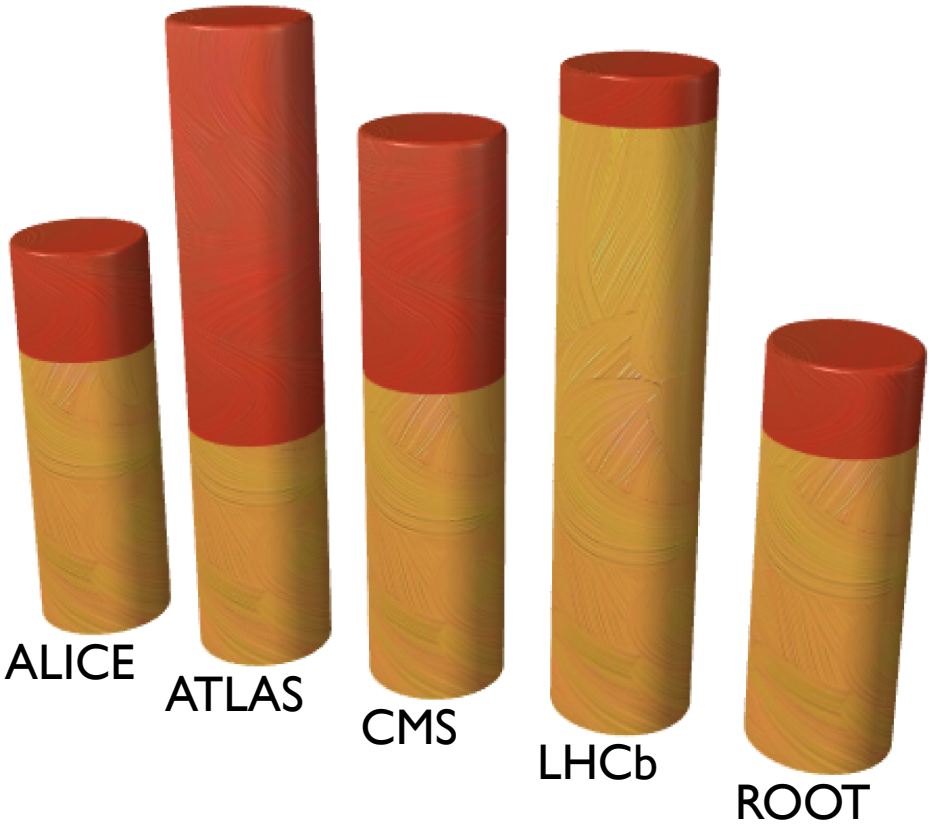
Setup

- Separate build + analysis server for each experiment due to processing times: 4 processes * 8 hours to 4 days - partially caused by build systems
- Dedicated database / tomcat server for web front-end, for each experiment
- Keeps databases reasonably small, user (web) interface responsive

Initial Numbers: Time



Initial Numbers: Issues



High Impact Defects

Has It Arrived?

- ROOT, particle beam, data transfer, virtual machine etc: nightly / regular use
- All LHC experiments use it, one even embedded in automatic QA
- Coverity is known in all experiments as “the tool that doesn’t forgive lazy coding”

Summary

- Software quality a key factor for CERN: influences quality of scientific results
- Blind spot despite of large QA toolset
- Coverity unique in abilities and quality
  - Improved software, happier users
  - Quick adoption at CERN (rare!): recognized to improve software quality dramatically