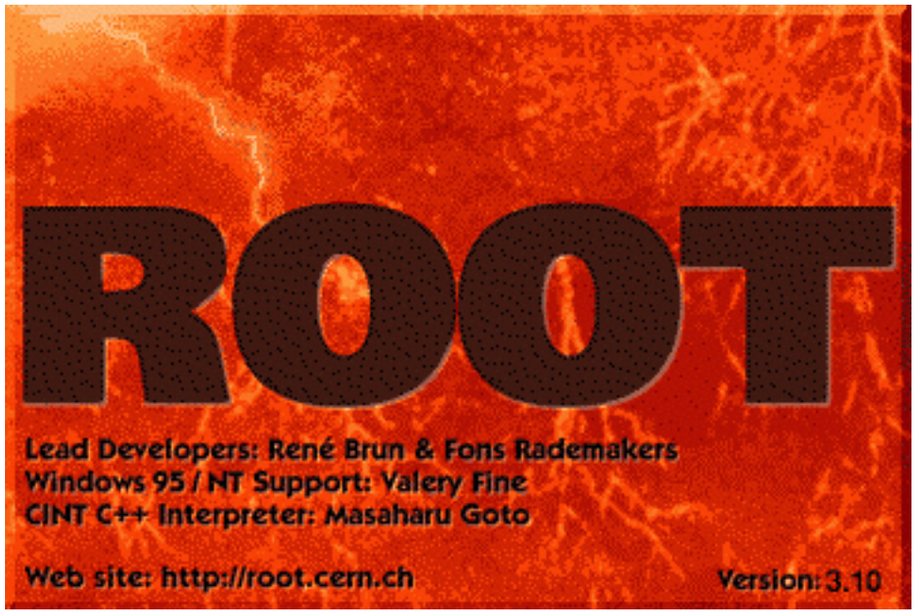
New Interfaces for ROOT 7

Axel Naumann

ROOT Users' Workshop, 2015

Prelude

ROOT has Evolved



Backward Compatibility

- For 20 years now, ROOT macros “just” work across ROOT versions:

```
TFile* f = new TFile(“hist.root”);
hpx->Draw( );
```

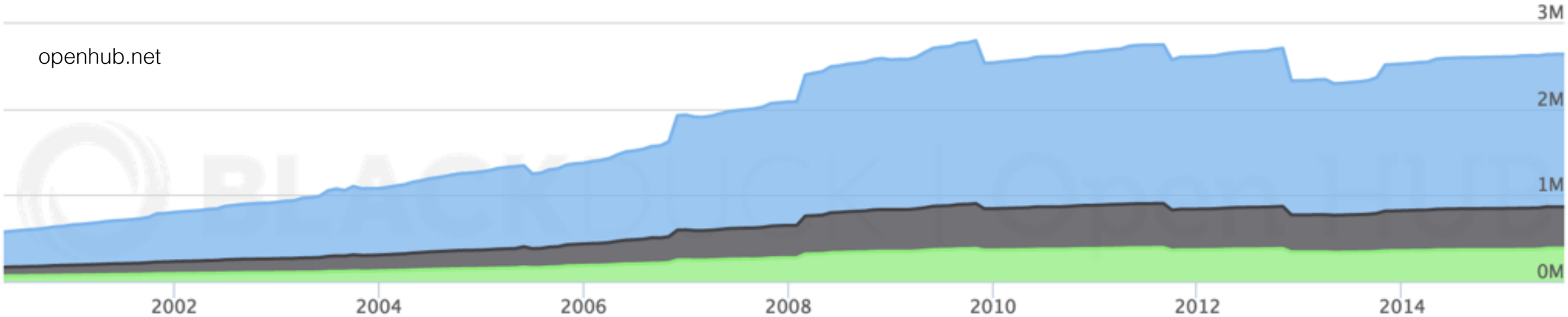
That’s Good, Right?

- Dated interface personality

- Functionality *changes* impossible

No Change of Behavior

- We can add. And we did. Plenty.



- But *changing* behavior is deadly for backward compatibility, see e.g. TAxis::SetRange()

v6 Interface Personality

- ROOT's interfaces homogenous and consistent
- Convey meaning besides the interface itself
- Example:

```
func(TObject* o);
```

  - takes something that inherits from TObject; check the documentation of what's allowed
  - it's likely not an optional argument (no "= 0")

v6 Interface Personality

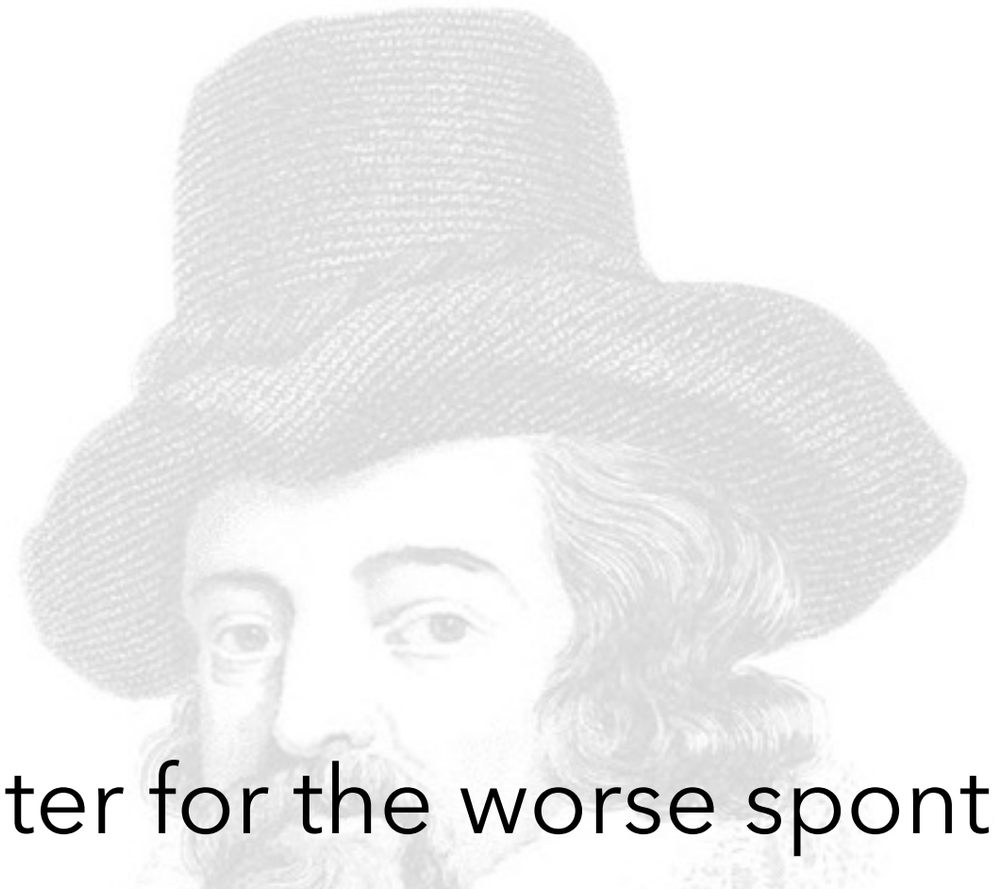
- Interface design is from the 1990s: object oriented, virtual interfaces; feature set available in C++98 and CINT

- Pointers-to-base

- PAW-style convenience: named objects, partial and implicit lifetime management

ROOT Bashing Is Fun!

- ROOT's design allowed us to do large scale, scientific computing for decades!
- Serialization! Math! Interaction with giant frameworks! All those plots! On (nearly) whatever platform!
- We all speak and know ROOT, and there is a reason
- ROOT is an excellent product, especially taking into account that it's ROOTed in the 90s!



"Things alter for the worse spontaneously, if they be not altered for the better designedly."

-Francis Bacon

This Talk

- Motivation
- Destination
- Progress
- Plan

Motivation

Attack Dated C++

- ROOT is not using the C++ you learn, nor that of 5 years from now

- Hinders interoperability with current code (user or frameworks): iterators, std library, virtual interfaces

New Interfaces

- Use language features to simplify physicists' life: more robust, more speed, more clarity
- Break backward compatibility **once**, to fix design issues without duck tape
  - not about TH1, but about RecursiveRemove(), const char* returns and int = -1 arguments, context (gPad, gDirectory) and alike

Goal: Interoperability

- Interoperability with current C++
  - standard types - std::string, std::vector
  - standard concepts - iterators, templates
  - side effect: less code in ROOT!
- Interoperability with other languages
  - explicit interfaces: no void*, no implicit ownership

Goal: Simplicity

- Very subjective - but we can likely agree on:
  - classes with 100 virtual functions are not simple
    - sweet spot of “simple” has changed with time: compiler support; multithreading; code size
- Split functionality, keep things focused
- Less TClass / interpreter, more regular C++

Goal: Task-Parallel

- Clear objective: reduce statics, caches
  - no context state: gDirectory, gPad
  - less object registration, no PAW-style memory management
- Plus “const means thread-safe” / documentation

Goal: Robustness

- Type-safety: no cast surprises
- Ownership defined on type-level (smart pointers); self-documenting
- func(const X&) not X* if func requires an X!
- Move from string flags to compiler-checkable identifiers: TFile::Open(…,"RECREATE") versus TFile::Recreate(…)

Goal: etc

- Backward compatibility until 2035
  - design for change: abstraction, separation of public and internal parts
- Speed
  - necessary, but not sufficient as an argument

Why Now?

- ROOT 6 is missing only PCMs and Windows
- We collected issues and solutions; hardware + software demand changes
- We have the tools now: modern C++ and its knowledge out there, deployment with experiments, cling
- Become ready with CERN experiments for Run 3

Destination

ROOT Is Our Language

```
canv->Draw(hist);
file->Write(“hpx”, hist);
```

…Adapt Where Needed

```
TFilePtr f = TFilePtr::Read(“hist.root”);
auto hist = f->Get<TH1F>(“hpx”);
```

Ingredients: C++ >= 14

- Rationale: target compilers in 3 years from now (but can actually be used already now)
- Makes code much simpler
  - example std::array_view (start and size), ranges

Ingredients: Ownership

- unique_ptr, shared_ptr
- TCoopPtr
  - pointer-handle: everyone can have one, once anyone calls TCoopPtr::Delete(), everyone will see a nullptr
  - replaces RecursiveRemove()

Ingredients: Focus

- Member function that only accesses public interfaces should not be a member: Fit()
- If adding state or optional complexity: split! THistBufferedFill, THistView
  - bad: more classes, good: clearer job description, reduced dependencies, only pay for what you use
- Documentation! Move internals into namespace

Ingredients != Goals

- Ingredients are tools to get the goals done

- Don’t misinterpret C++14 as the goal!

tutorials/v7/simple.cxx

```
#include "ROOT/THist.h"
#include "ROOT/TFit.h"
#include "ROOT/TFile.h"

void simple() {

  // Create a 2D histogram with an X axis with equidistant bins, and a y axis
  // with irregular binning.
  ROOT::TAxisConfig xAxis(100, 0., 1.);
  ROOT::TAxisConfig yAxis({0., 1., 2., 3.,10.});
  ROOT::TH2D histFromVars(xAxis, yAxis);

  // Or the short in-place version:
  // Create a 2D histogram with an X axis with equidistant bins, and a y axis
  // with irregular binning.
  ROOT::TH2D hist({100, 0., 1.}, {{0., 1., 2., 3.,10.}});

  // Fill weight 1. at the coordinate 0.01, 1.02.
  hist.Fill({0.01, 1.02});

  // Fit the histogram.
  ROOT::TFunction<2> func([](const std::array<double,2>& x,
                             const std::array_view<double>& par)
                          { return par[0]*x[0]*x[0] + (par[1]-x[1])*x[1]; });

  ROOT::TFitResult fitResult = ROOT::FitTo(hist, func, {{0., 1.}});

  ROOT::TFilePtr file = ROOT::TFile::Recreate("hist.root");
  file->Write("TheHist", &hist);
}
```

#includes

```
#include "ROOT/THist.h"
#include "ROOT/TFit.h"
#include "ROOT/TFile.h"
```

- Scoped headers

Explicit Concepts

```
ROOT::TAxisConfig xAxis(100, 0., 1.);
ROOT::TAxisConfig yAxis({0., 1., 2., 3.,10.});
ROOT::TH2D histFromVars(xAxis, yAxis);
```

- ROOT 6: “just” a set of arguments.

- Added structure, yet easily usable due to C++11:

```
ROOT::TH2D hist({100, 0., 1.},
                {{0., 1., 2., 3.,10.}});
```

Contemporary C++ == Safety

```
hist.Fill({0.01, 1.02});
```

- We might not know, but young people and compilers read: “pass collection with two elements”
- Map Double_t* to “coordinate” concept:
  - let compiler check the size! (and yes, make it available!)

Use Contemporary Wording

```
ROOT::TFitResult fitResult
  = ROOT::FitTo(hist, func, {{0., 1.}});
```

- Separate properties from operations on objects

More…

```
file->Write("TheHist", hist);
```

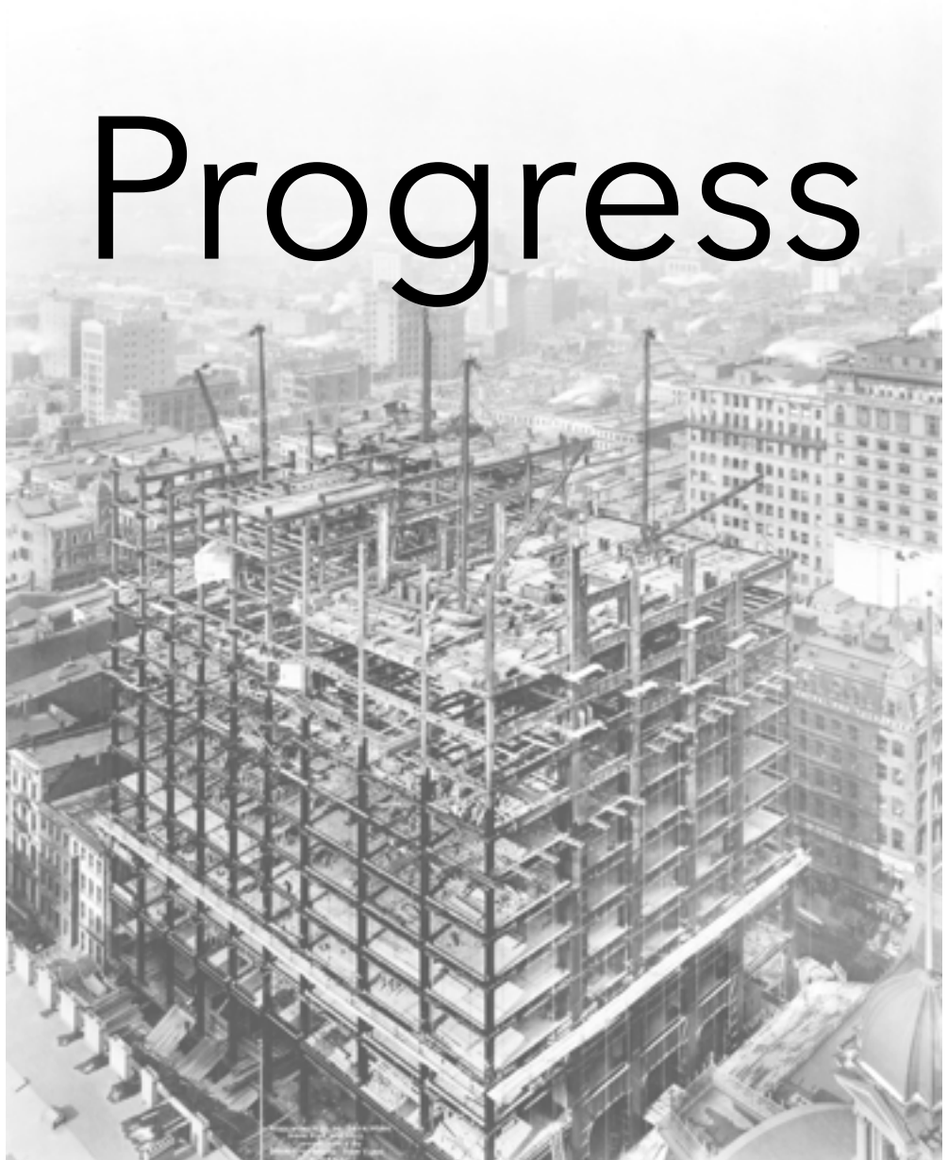
- No names, unless as a key

- Lifetime management is good - but now explicit

- Reduce TClass + interpreter use, replace by templates, abstract interfaces

- Pass reference if function expects valid argument

Progress



Proof-Of-Concept Prototype

```
cmake -Droot7=On -Dcxx14=On
```

- In v6.05.02! See tutorials/v7/

- Show-cases interfaces
  - started with histograms and their interaction with TFile, drawing
  - THistView, bin iteration, compile-time properties…

- Criticism very welcome!

Major Missing Features

- No graphing, no serialization yet
- (Almost) no histogram operations yet

- **Warning** - this is a non-functional prototype!
  For now…

THist<2, float>

- User-level interface
  - provides range (all or view) operations, for instance Add(), Project()
  - access by coordinates
- Points to…:

THistImplBase<2, float>

- Implementation (but accessible for users)
  - provides bin index operations, bin iterator
  - derived types templated on axis type (equidistant, irregular,…) and statistics type

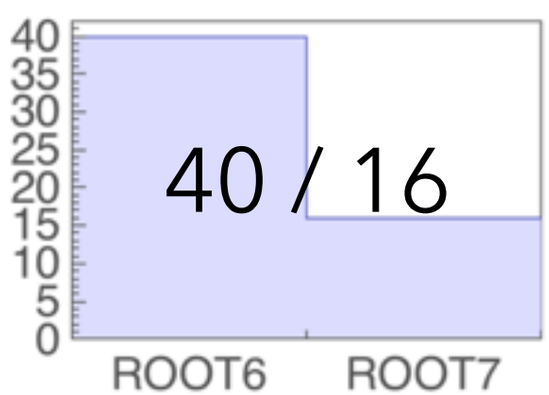
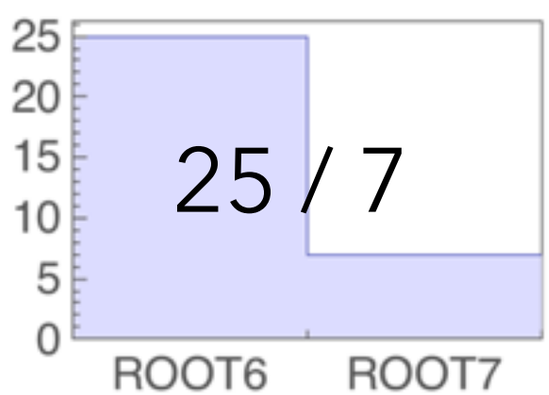
Separation THist/Impl

- Less runtime if-s: axis, statistics are compile-time
  - much faster Fill()
- Reduced interface complexity
- Can move vtable evaluation out of bin-loop
  - apply lambda to all bins: add, project etc

Performance

TH2D on a Xeon E5-2650 2.6GHz, no Sumw2()

Fixed numbers!

	Irregular binning ROOT 6 / ROOT 7	Equidistant binning ROOT 6 / ROOT 7
seconds / 10M Constructions		
seconds / 1G Fills		

TPad::Draw(WHAT, OPT)

- Calls

```
std::unique_ptr<TDrawable>
GetDrawable(…_ptr<THist<D,T>>,
                  HistOpts<D>)
```

- Free function creates TPad primitive, here for THist + drawing options; THist stays math object

- Loose coupling of painting through TDrawable interface: invokes THistPainter from libHistPainter

  - extendable drawing without TObject

The Plan

Gradual Transition

- New interfaces arrive one by one
  - cannot devote full team on this
  - design with care, take time: these interfaces should survive for the next 20 years!
- Think ROOT in the 90s
  - start small!

Avoiding Collisions

- Before release of ROOT 7:
  - new interfaces arrive in ROOT::
- After release of ROOT 7:
  - replaced, old interfaces move to ROOT::v6::

ROOT 7

- “ROOT 7” once relevant fraction is available, then deprecate old interfaces

- Goal: in time for Run 3 shutdown

- we know experiments are questioning their software fundamentals

- make ROOT the obvious building block again! We have the experience and expertise

Feasible?

- Allow reading old data into new ROOT
- Glue new interfaces to rest of ROOT 6
- Later: ROOT 6 interfaces use ROOT 7 ones
- Interact with experiments, early and continuously
  - take what worked well for ROOT 6

Conclusion

The Goal

- The world has changed, ROOT needs to adapt
- Successful maintenance, yet need for evolution
- Can only convince through features, robustness, simplicity: usability

The Path

- Small steps enable organic growth: enable feed-back loop

- Early involvement and adoption has proven a key ingredient to success of ROOT 6 (and v1, v2,…), much more for ROOT 7

- In time for Run 3!

ROOT 7

- Like ROOT 6: a revolution for ROOT
- New interfaces are an integral part, enabling ROOT to implement the lessons learnt over two decades
- While being backward incompatible once, ROOT will continue to be built by experts, based on experience, for HENP production tasks