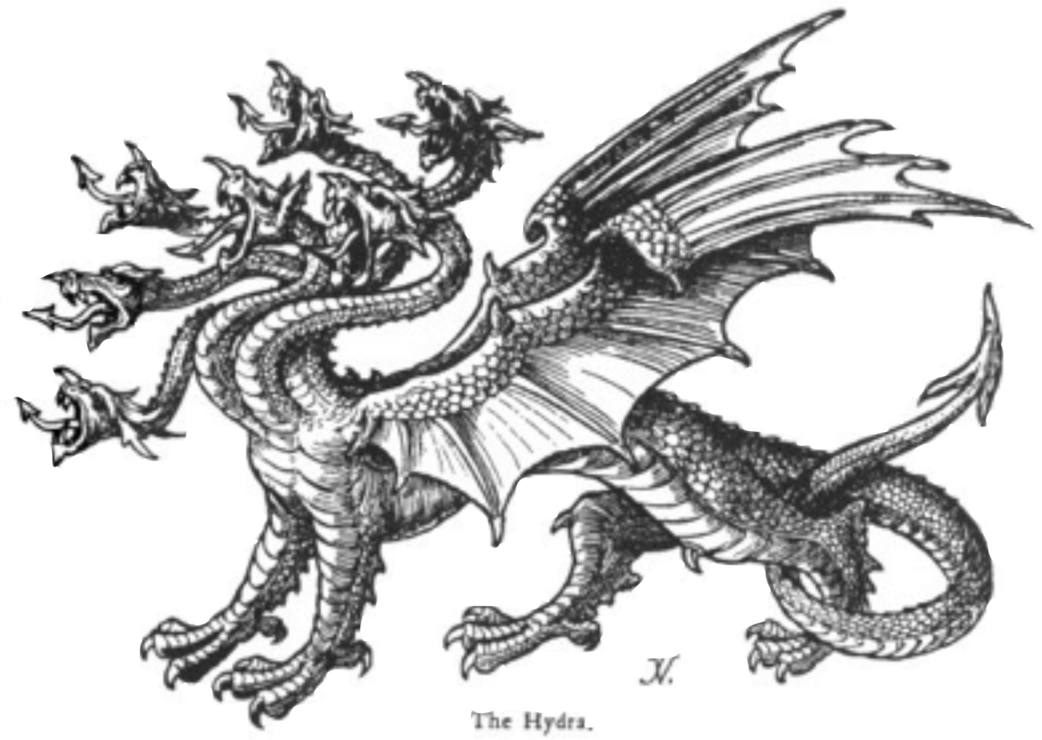
Your ROOT for Your Run3

Axel Naumann
ATLAS Software-TIM, Glasgow, 2016-06-06

Your Custom Order

- ROOT in a multi-threaded framework
- ROOT’s new interfaces

ROOT in MT framework



?

- Distinguish multi-threaded operations *in* ROOT
- But here, topic is “you have a MT framework and use ROOT”
  - but there is some interaction, so:

Existing MT Features

- MT usage *in* ROOT, shows areas of thread-safety
  - spawning TBB tasks, “implicit multi-threading"
- E.g. TTree::GetEntry() reads branches in parallel
  - no external task granularity: Gaudi cannot read “first branches first”
- More to come: math,…

Current Situation

- Some code paths are tested to be MT safe
  - CMS / Philippe implemented much of this. Thank you, CMS!
- MT safety almost completely due to selective locking plus a few atomics plus *thread* locals
  - thread != task

MT Parts / Paths {

Thread-Safe I/O

- One thread, one TFile
  - reading
  - and even writing
- “Proven” though CMSSW in production since years

Thread-Safe Histogramming

- NOT filling the same histogram from multiple threads!
- BUT one histogram per thread
    - plus ROOT::TThreadedObject

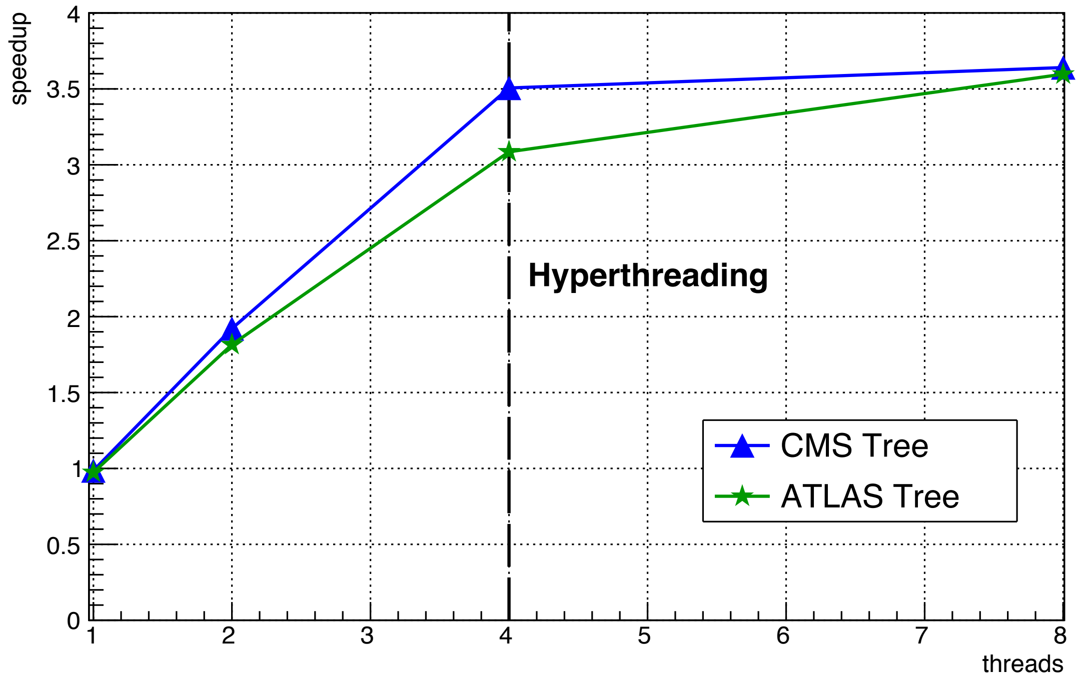
Thread-Safe Fitting

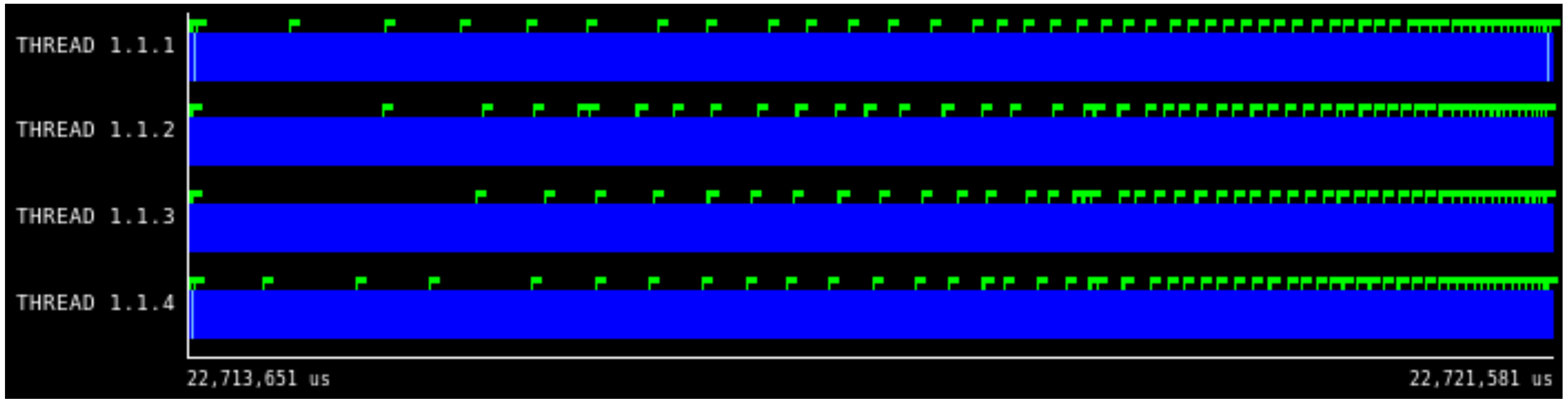
- CMSSW verified thread safety:
    - fit different histograms in different threads

- Started making it part of "implicit threading"

|| Branch Reads

- Part of "implicit threading"
- read-a-branch == TBB task
- Reading 200 top-level ATLAS branches in 4 threads





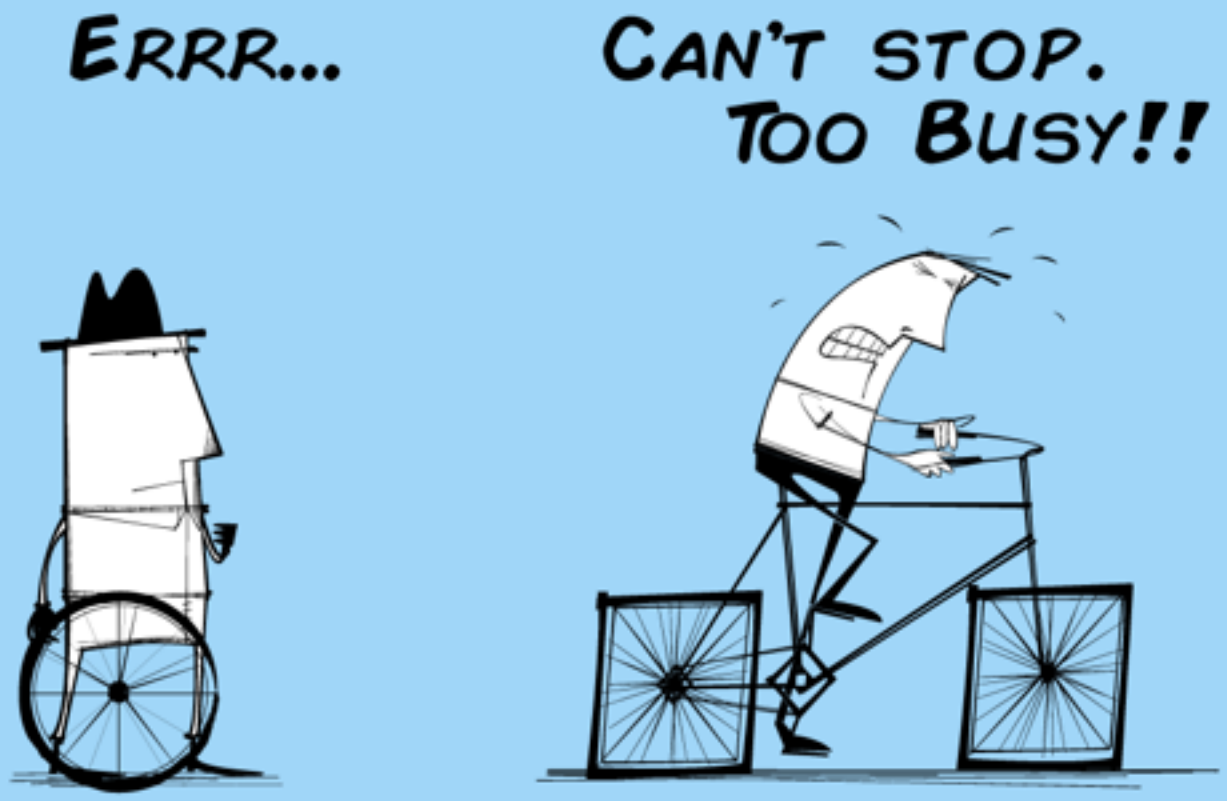
}

Retrofitting Thread-Safety

- Provide thread safety for required code paths
- Change implementations to be thread-safe
  - no more gDirectory, gPad, caches
  - get rid of global “list of cleanups” (memory management) - impossible :-(

Challenges of Retrofitting

- Pragmatic but intensive
  - requires a myriad of small-scale changes
- Sometimes not enabling parallelism but safeguarding (against) it
- Takes effort, usually requires interface changes, limited reach, trying since years…



audiencestack.com

New! Interfaces!

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

New! Interfaces!

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

```
// ROOT v6:
hist->Draw();
hist->SetName(“hpx”);
hist->Write();
```

ROOT's New Interfaces

- See https://indico.cern.ch/event/395887/contributions/947390/ from ATLAS Software Technical Meeting, Sept 2015, LBNL
- Short recap:

Motivation

- *Robustness:* type safety, modern ownership management, no pointers, no string options but identifiers
- *Simplicity:* standard types, be explicit about side-effects & context & ownership
- *Interoperability:* with current C++ code, other languages

Motivation (Less)

- *Speed:* less vtables, more inlines, more vectorization
- *Cooperativeness:* ROOT::, ROOT/, const == thread safe
- *Less magic:* Reduce impact of interpreter / TClass
- *Clean up:* 20 years of collected "incremental improvements"

Why now?

- C++ enables it (and makes TList a relic)
- cling
- Run 3 (and H*-LHC)
- Reaction to change of environment after 20 years

Truth *And* Consequence

- Virtually first backward incompatible change since 20 years!
- Will be backward compatible once moved from ROOT::Experimental:: to ROOT::
- Goal: intentionally similar in user code but very different in implementation / interface style
  - will likely provide rewrite-tool to update code

Gradual Deployment

- Can’t release all in one go
  - not enough resources, neither for experiments nor for ROOT
- Instead push fresh from keyboard
  - early feedback from physicists and experiments
  - co-existence of old and (more and more) new

Miracle-Gro
all purpose
continuous release
plant food
GROWS PLANTS TWICE AS BIG!*
UNIQUE MIX OF 10 NUTRIENTS
FOR HEALTHY, BEAUTIFUL PLANTS

Status

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

void simple() {
  TH1F hist(100, 0., 1.);
  auto file = TFile::Recreate("hist.root");
  file->Write("hpx", hist);
}
```

Surely you're joking!

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

void simple() {
  TH1F hist(100, 0., 1.);
  auto file = TFile::Recreate("hist.root");
  file->Write("hpx", hist);
}
```

Well no. Status:

- Histograms: creation, fill, addition (partially)
- I/O: writing (using v6 ROOT file in the back)
- Object registration by name / old-style memory management

Sketched:

- Drawing
- Fitting
- Concurrent fill through buffers
- Plugins: lib(ROOT)HistPainter
- Logging

Why do you care?

- ROOT: 20 years of experience, production use
  - there are pet peeves that matter (RecursiveRemove) and those that don't matter that much (TObject)
- We can integrate changes into physicists' production environments
  - power + responsibility

Concrete Examples, Please.

Convince Thru Code

- “Trust us, we know what we are doing!” :-)
- Otherwise, here’s the code:

  https://root.cern.ch/doc/master/namespaceROOT_1_1Experimental.html

- And here:

“Legacy” typedefs

```
using TH3F = THist<3, float,
  THistStatContent, THistStatUncertainty>;
using TH3C = THist<3, char,
  THistStatContent>;
```

- TH3F looks like current ROOT
- Mostly obvious: dimension, precision
  - specify which statistics to collect

Construction

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

- Axis is an entity
    - useful for declaring multiple histograms
- C++ allows unanimous agreement on array size

Fill

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

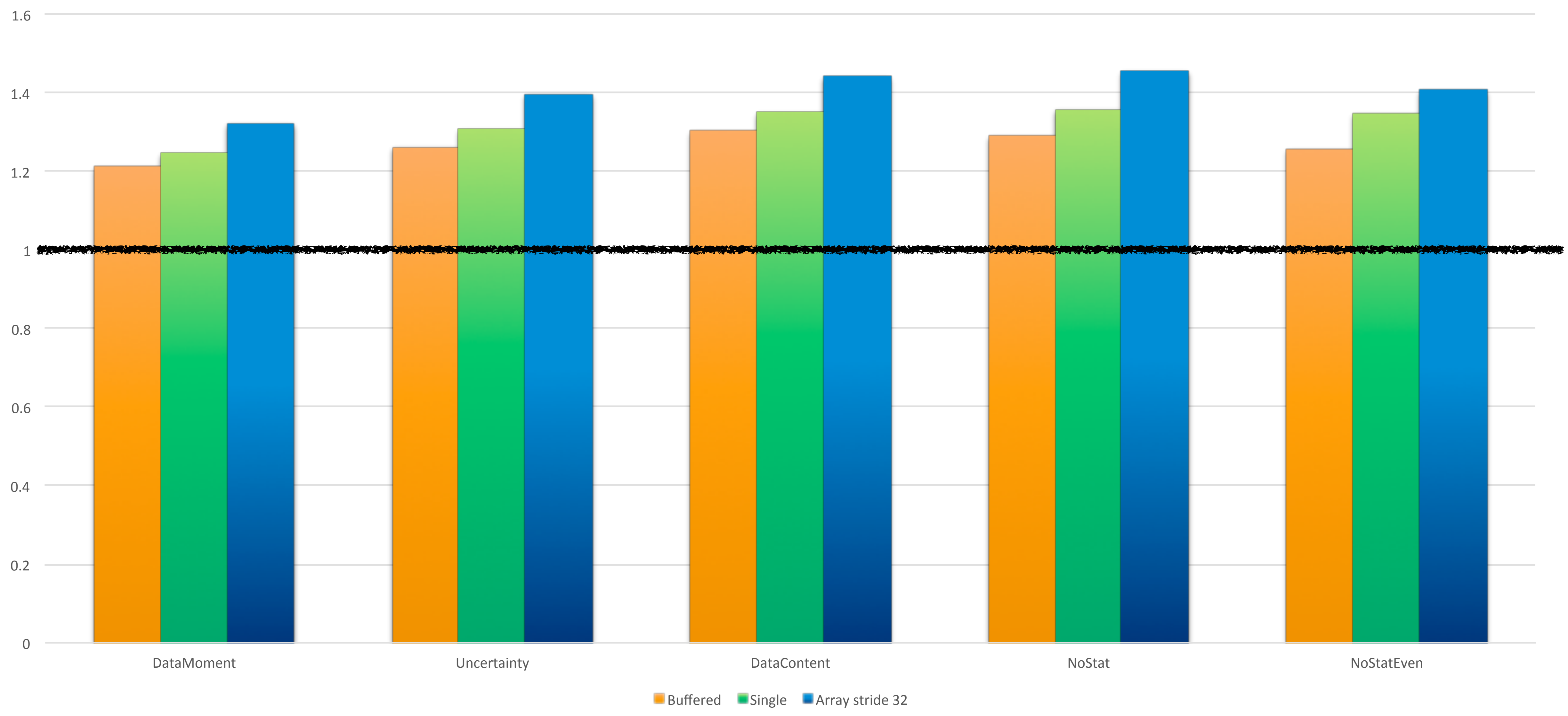
- Typical: same “language” but sturdy interface
- Here, too: well-defined array size of coordinate

Speed

- Necessary, but not sufficient.
- Philippe Canal measured new / old interfaces

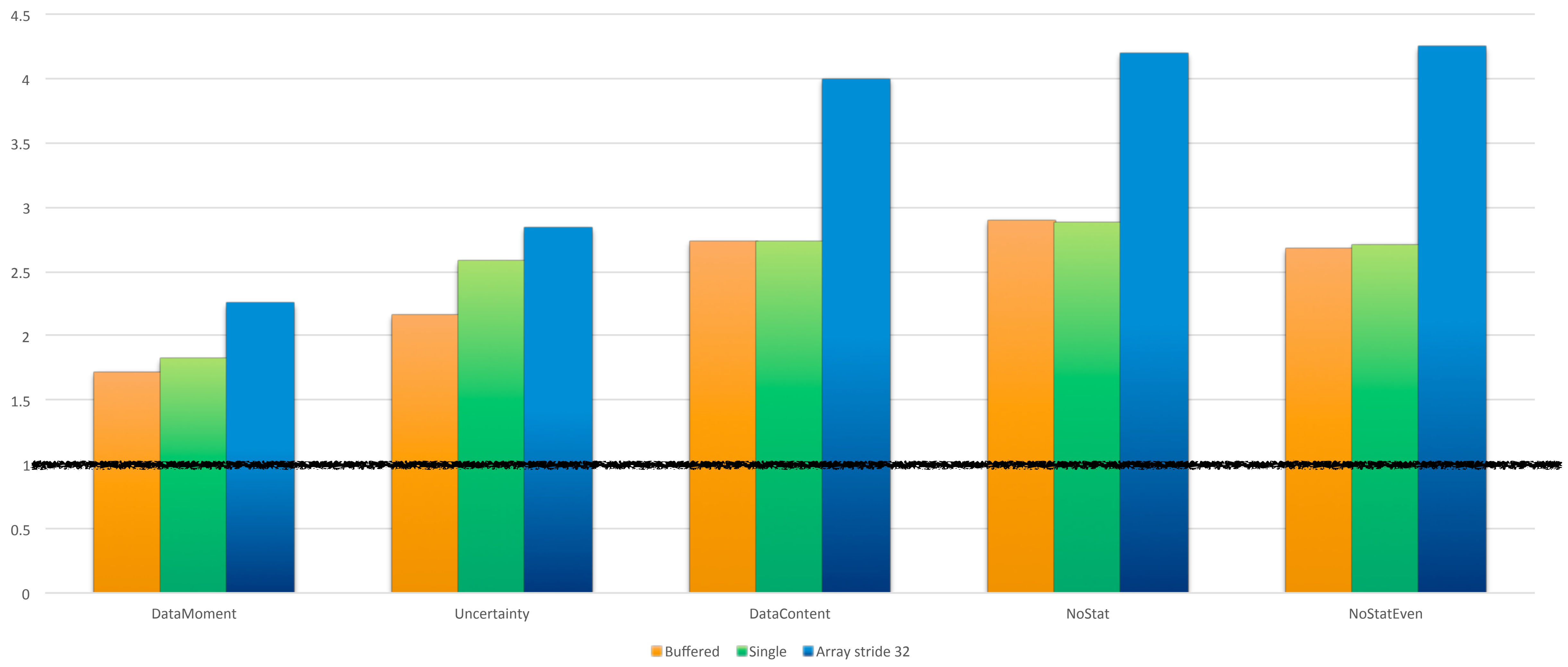
TH2D Irregular Bins

- Orange: Buffered, Green: Fill(), Blue: FillN(32)



TH2D Equidistant Bins

- Moment, SumW2, SumW, NoStats in new, NoStats



Speed of `bin += weight`?!

- Internally,

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

  is a pimpl to

```
THistImpl<Detail::THistData<2, double,
    Detail::THistDataDefaultStorage,
    THistStatContent, THistStatUncertainty>,
  TAxisEquidistant, TAxisIrregular>
```

- Detailed but efficient. And hidden.

THistImpl

```
THistImpl<Detail::THistData<2, double,
    Detail::THistDataDefaultStorage,
    THistStatContent, THistStatUncertainty>,
  TAxisEquidistant, TAxisIrregular>
```

- The axis kind
- No more “if this is variable bin, if it can grow, if etc etc etc”

THistImpl

```
THistImpl<Detail::THistData<2, double,
    Detail::THistDataDefaultStorage,
    THistStatContent, THistStatUncertainty>,
  TAxisEquidistant, TAxisIrregular>
```

- Which statistics to collect and to store
- No more “collect second moment just because”
- No more “half the hist was without Sumw2()”

THistImpl

```
THistImpl<Detail::THistData<2, double,
    Detail::THistDataDefaultStorage,
    THistStatContent, THistStatUncertainty>,
  TAxisEquidistant, TAxisIrregular>
```

- How to store per-bin data
  - super-expert customization
- E.g. allocator support

Simplicity

- Showing all of THist
  - except for noexcept, constexpr, = default noise
  - not showing “std::”

Simplicity

```
template<…>
class THist {
public:
   static int GetNDim();
   THist(array<TAxisConfig, DIMENSIONS> axes);
   THist(string_view histTitle,
         array<TAxisConfig, DIMENSIONS> axes);
… // + Overloads for 1-3 dimensions.

   ImplBase_t *GetImpl() const;
```

Simplicity

```
void Fill(const CoordArray_t &x,
          Weight_t weight = (Weight_t) 1);
void FillN(const array_view <CoordArray_t> xN,
           const array_view <Weight_t> weightN);
void FillN(const array_view <CoordArray_t> xN);

int64_t GetEntries() const;
Weight_t GetBinContent(const CoordArray_t &x) const;
double GetBinUncertainty(const CoordArray_t &x) const;
```

Simplicity

```
const_iterator begin() const;
const_iterator end() const;

void swap(THist<…> &other);
```

- Concise, standard, common

Free Functions

```
/// Add two histograms with no matching axes.
template<…>
void Add(THist<…_TO> &to, THist<…_FROM> &from) {
…
  auto add = [fillFuncTo, toImpl]
    (const FromCoord_t& x, FromWeight_t c)
  {
    (toImpl->*fillFuncTo)(x, c);
    // TODO: handle uncertainty
  };
  from.GetImpl()->ApplyXC(add);
}
```

Free Functions

```
/// Interface to graphics taking a
/// shared_ptr<THist>.
template<…>
unique_ptr <Internal::TDrawable>
GetDrawable(shared_ptr<THist<…> hist,
            THistDrawOptions<DIMENSIONS> opts={})
```

- Allow additional modularity
- Keep interfaces compact

Lessons Learned

- Had to re-learn C++ - and hell that was worth it!
- Our tooling infrastructure is inadequate
  - cannot express “dictionary for this template instance needs dictionary for those template instances”
  - sorry it took us a while… but we’re on the same page now!

Goal

- Provide basic implementation of new histograms in time for Run 3 software upgrades, i.e. by the end of the year
  - enough for frameworks to start adapting
- Several other developments going on in parallel, loosely to tightly coupled to new interfaces
  - e.g. new GUI, new TTree analysis approach

Next Steps

- Fix bug in rootcling for storing THist [June]
- Test coverage! [June]
- Implement dictionary selection mechanism [July]
- THist::Draw, using current TCanvas in the background [August]
- Fitting, using current fitting interfaces [August]

Your Run 3

- What are your requests for us?
  - performance!
  - parallelism!
  - analysis features!
  - I/O!
- Are we missing anything?