Faster Collections in RooFit

S. Hageboeck (CERN, EP-SFT) for the ROOT team

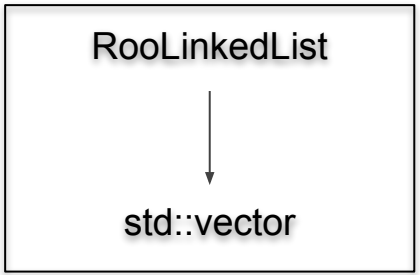
ROOT

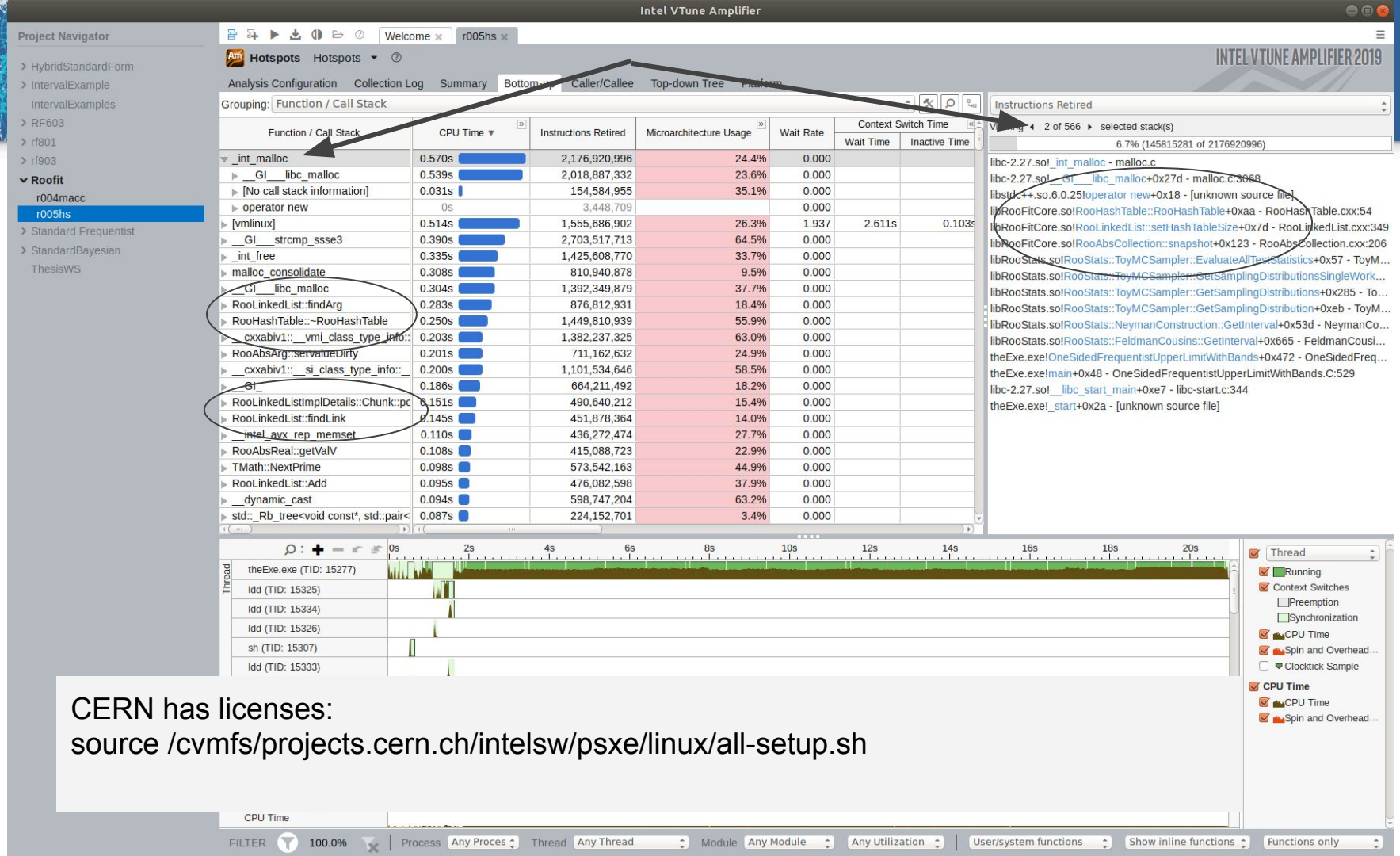
Data Analysis Framework

https://root.cern

RooFit has three kinds of collections:

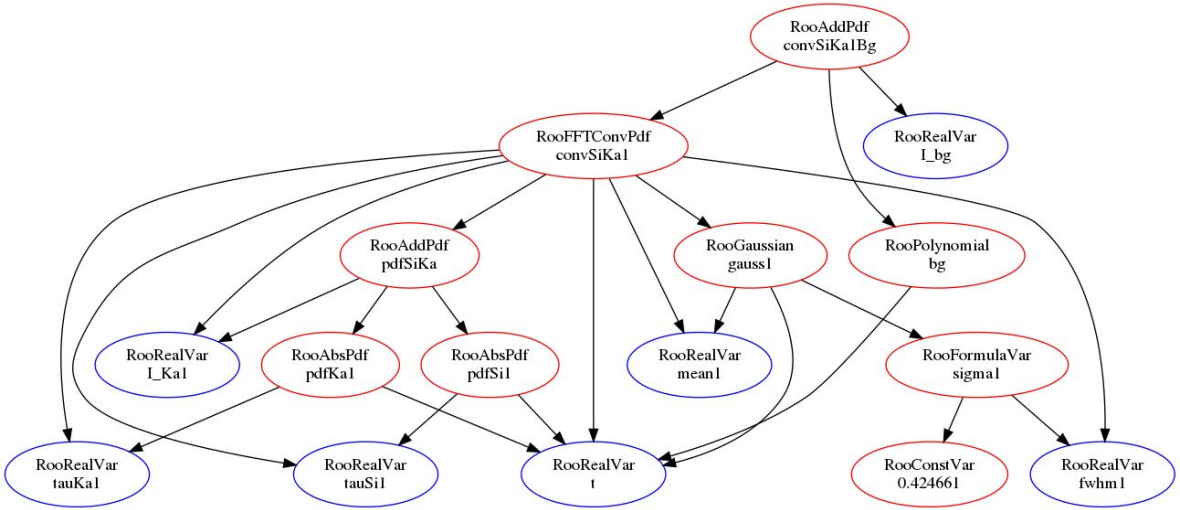
- RooLinkedList:
  - Linked list of most general class in RooFit (RooAbsArg)
  - Memory pool for faster allocations & more data locality
  - Can enable a hash list to faster find elements by name
- RooArgSet:
  - Set of RooAbsArgs
  - RooLinkedList at backend
- RooArgList:
  - List of RooAbsArgs. Mostly used like a std::vector.
  - RooLinkedList at backend
- Most common operations (for all three):
  - **Forward iteration!!**
  - Search element by name or address (= forward iteration)
  - Add element (to back!)





CERN has licenses:
source /cvmfs/projects.cern.ch/intelsw/psxe/linux/all-setup.sh

RooAbsCollection



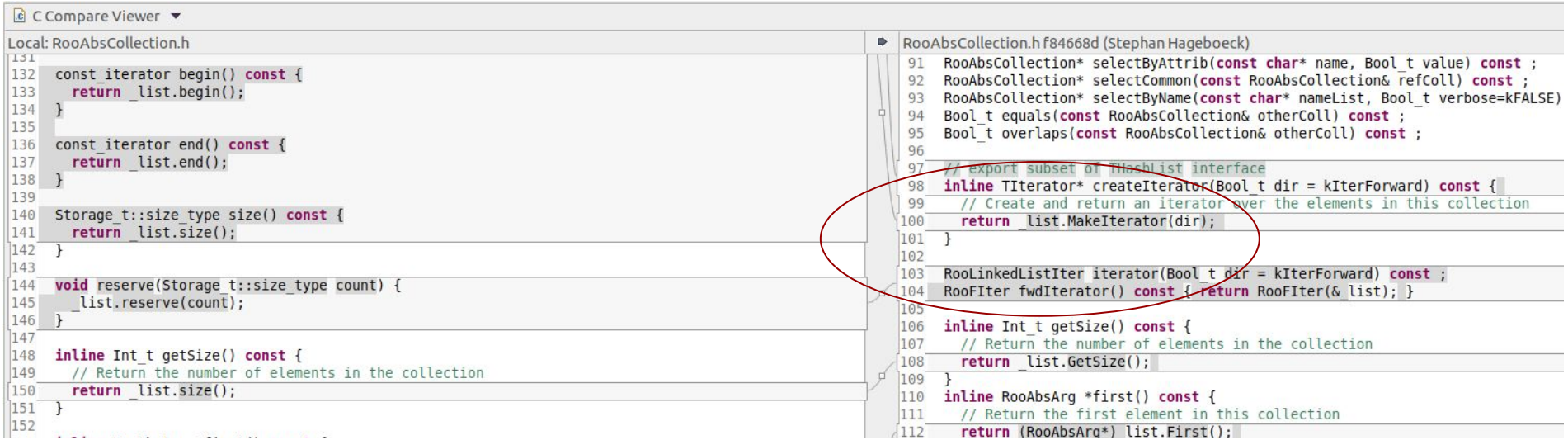
Collections:

- Expression tree (+ almost everything in RooFit) stored as RooLinkedList<RooAbsArg*>
  - Often small search & iterates
- Toy MonteCarlo generation:
  - ~50% of L3 cache misses due to linked list + hash table operations

- **The plan**:
  - Replace LinkedList by std::vector
  - Provide STL-like interface

The Challenge

- Axel: “How much user code are you going to break?”
  → All …
- The old collections directly expose the underlying storage implementation through the iterators



Solution

- Three kinds of old iterators need to be supported (all in use)
- RooLinkedList not completely gone anyway
- Write wrapper that delegates to RooLinkedList or STL as needed
- Downside: slower
  - Extra layer with virtual dispatch
  - Need to create&destroy iterators (and hand into userland) for polymorphy

```
/// Interface for RooFIter-compatible iterators
class GenericRooFIter
{
  public:
  virtual RooAbsArg * next() = 0;
  virtual ~GenericRooFIter() {}
};

////////////////////////////////////////////////////////////////////////////////////////////////
/// A one-time forward iterator working on RooLinkedList or RooAbsCollection.
/// This wrapper separates the interface visible to the outside from the actual
/// implementation of the iterator.
class RooFIter final
{
  public:
  RooFIter(std::unique_ptr<GenericRooFIter> && itImpl) : fIterImpl{std::move(itImpl)} {}
  RooFIter(const RooFIter &) = delete;
  RooFIter(RooFIter &&) = default;
  RooFIter & operator=(const RooFIter &) = delete;
  RooFIter & operator=(RooFIter &&) = default;

  RooAbsArg *next() {
    return fIterImpl->next();
  }

  private:
  std::unique_ptr<GenericRooFIter> fIterImpl;
};

////////////////////////////////////////////////////////////////////////////////////////////////
/// Implementation of a GenericRooFIter for the RooLinkedList
class RooFIterForLinkedList final : public GenericRooFIter
{
  public:
  RooFIterForLinkedList() {}
  RooFIterForLinkedList(const RooLinkedList* list) : fPtr (list->_first) {}

  /// Return next element in collection
  RooAbsArg *next() override {
    if (!fPtr) return nullptr ;
    TObject* arg = fPtr->_arg ;
    fPtr = fPtr->_next;
    return (RooAbsArg*) arg ;
  }

 private:
    const RooLinkedListElem * fPtr{nullptr};  //! Next link element
};
```

The Challenge II

- RooLinkedList:
  - Remove/add/replace before and after current iterator
  - No reallocations → iterator valid
- Solution: Legacy-to-STL adapters count
  - Can remove/add after iterator
  - Can replace everywhere
  - Safe also if reallocating
  - **But: Will break** when removing/adding **before** iterator

```
#ifdef NDEBUG
  RooAbsArg * next() override {
    if (atEnd())
      return nullptr;
    return fSTLContainer[fIndex++];
  }
#else
  RooAbsArg * next() override {
    if (atEnd())
      return nullptr;
    return nextChecked();
  }
#endif
```

The Challenge II

- Detect broken loops:
  - Unless NDEBUG set, legacy iterators check if they point to the same element before incrementing
  - Fail noisily if not
- Does this happen?
  - RooFit: two in hundreds of loops
  - User code: Users usually just forward-iterate

```
#ifdef NDEBUG
  RooAbsArg * next() override {
    if (atEnd())
      return nullptr;
    return fSTLContainer[fIndex++];
  }
#else
  RooAbsArg * next() override {
    if (atEnd())
      return nullptr;
    return nextChecked();
  }
#endif
```

```
#ifndef NDEBUG
template<class STLContainer>
RooAbsArg * TIteratorToSTLInterface<STLContainer>::nextChecked() {

  RooAbsArg * ret = fSTLContainer.at(fIndex);
  if (fCurrentElem != nullptr && ret != fCurrentElem) {
    throw std::logic_error("A RooCollection should not be modified while iterating. "
        "Only inserting at end is acceptable.");
  }
  fCurrentElem = ++fIndex < fSTLContainer.size() ? fSTLContainer[fIndex] : nullptr;

  return ret;
}
#endif
```

Is this the right way to notify users?

The Legacy Iterators now

- All legacy iterators work
- Slower than before
- Deprecated in Doxygen

- Flagged with `R__SUGGEST_FUNCTION`*:
  - Requested during user's workshop
  - Flags functions/classes whose use is discouraged, but won't be fully deprecated
  - https://github.com/root-project/root/pull/3100

C Compare Viewer

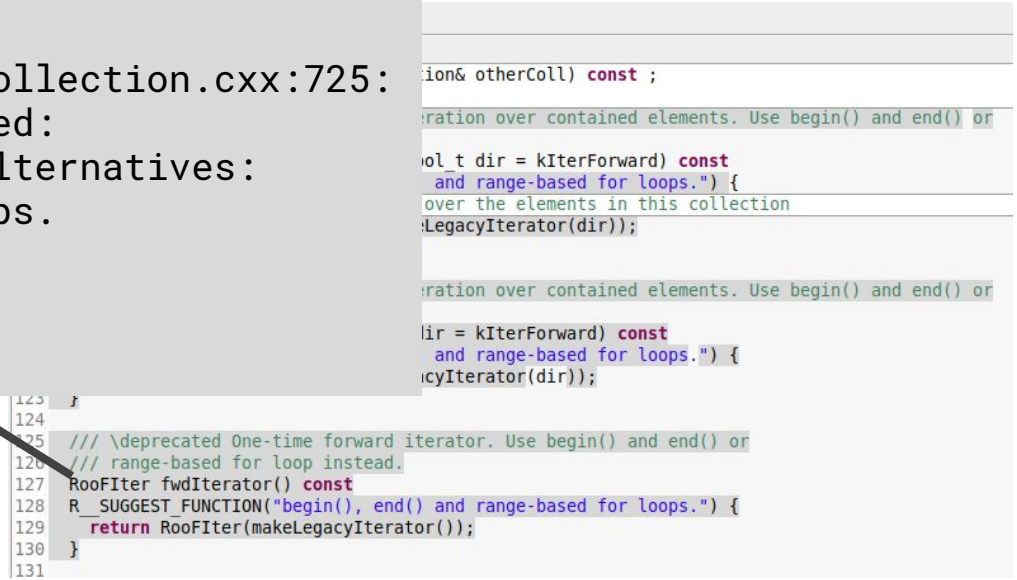
Local: RooAbsCollection.h

```
108  Bool_t overlaps(const RooAbsCollection& otherColl) const ;
109
110  /// \deprecated TIterator-style iteration over contained elements. Use begin() and end() or
111  /// range-based for loop instead.
112  inline TIterator* createIterator(Bool_t dir = kIterForward) const
113  R__SUGGEST_FUNCTION("begin(), end() and range-based for loops.") {
114    // Create and return an iterator over the elements in this collection
115    return new RooLinkedListIter(makeLegacyIterator(dir));
116  }
117
118  /// \deprecated TIterator-style iteration over contained elements. Use begin() and end() or
119  /// range-based for loop instead.
120  RooLinkedListIter iterator(Bool_t dir = kIterForward) const
121  R__SUGGEST_FUNCTION("begin(), end() and range-based for loops.") {
122    return RooLinkedListIter(makeLegacyIterator(dir));
123  }
124
125  /// \deprecated One-time forward iterator. Use begin() and end() or
126  /// range-based for loop instead.
127  RooFIter fwdIterator() const
128  R__SUGGEST_FUNCTION("begin(), end() and range-based for loops.") {
129    return RooFIter(makeLegacyIterator());
130  }
131
```

* Names not finalised. Opinions?

The Legacy Iterators now

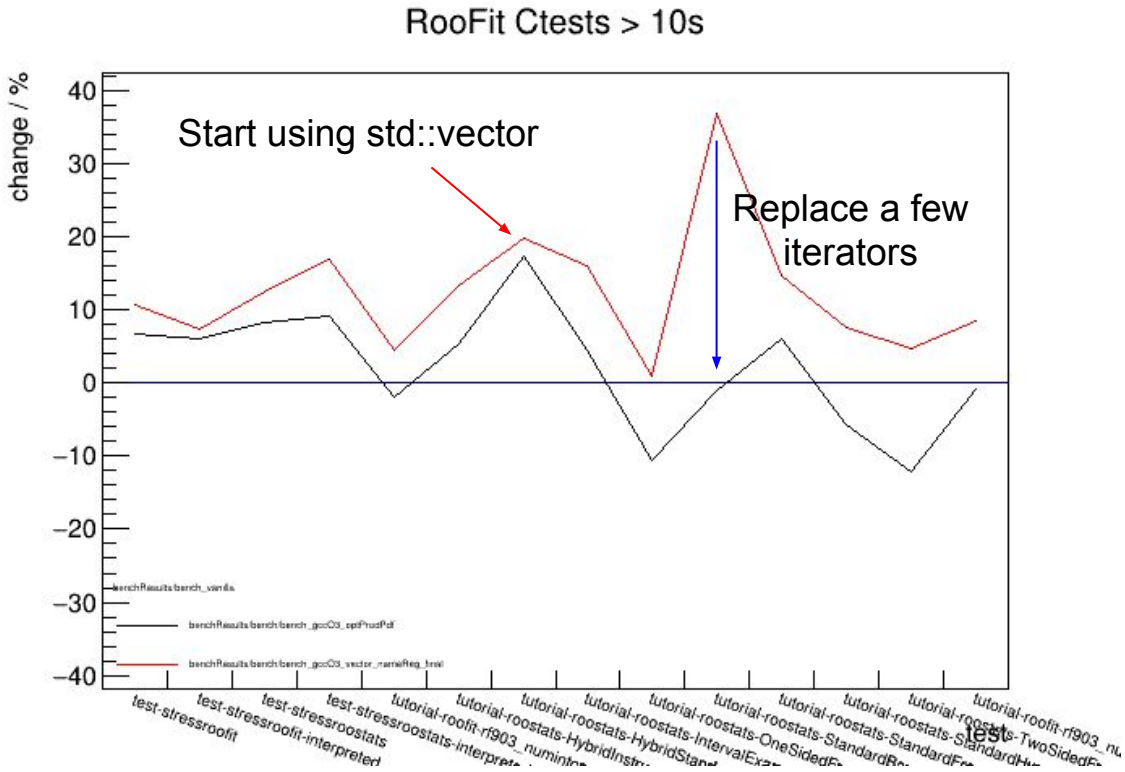
```
#define R__SUGGEST_FASTER_FUNCTIONS*


root-src/roofit/roofitcore/src/RooAbsCollection.cxx:725:
21: warning: 'fwdIterator' is deprecated:
This function has faster/more secure alternatives:
begin(), end() and range-based for loops.
[-Wdeprecated-declarations]
    RooFIter iter = fwdIterator() ;
                    ^
```



- workshop
- Flags functions/classes whose use is discouraged, but won't be fully deprecated
- https://github.com/root-project/root/pull/3100

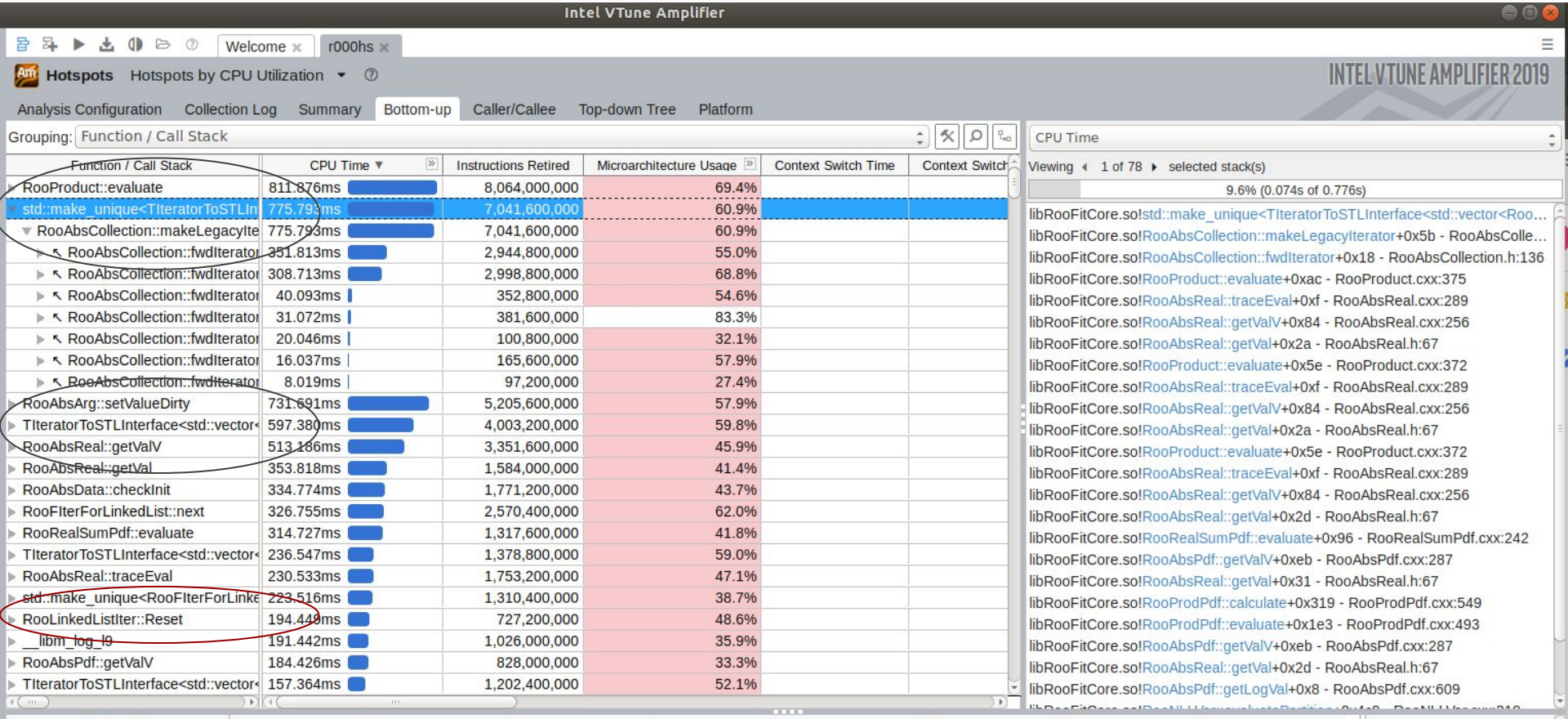
* Names not finalised. Opinions?

First Speed Tests

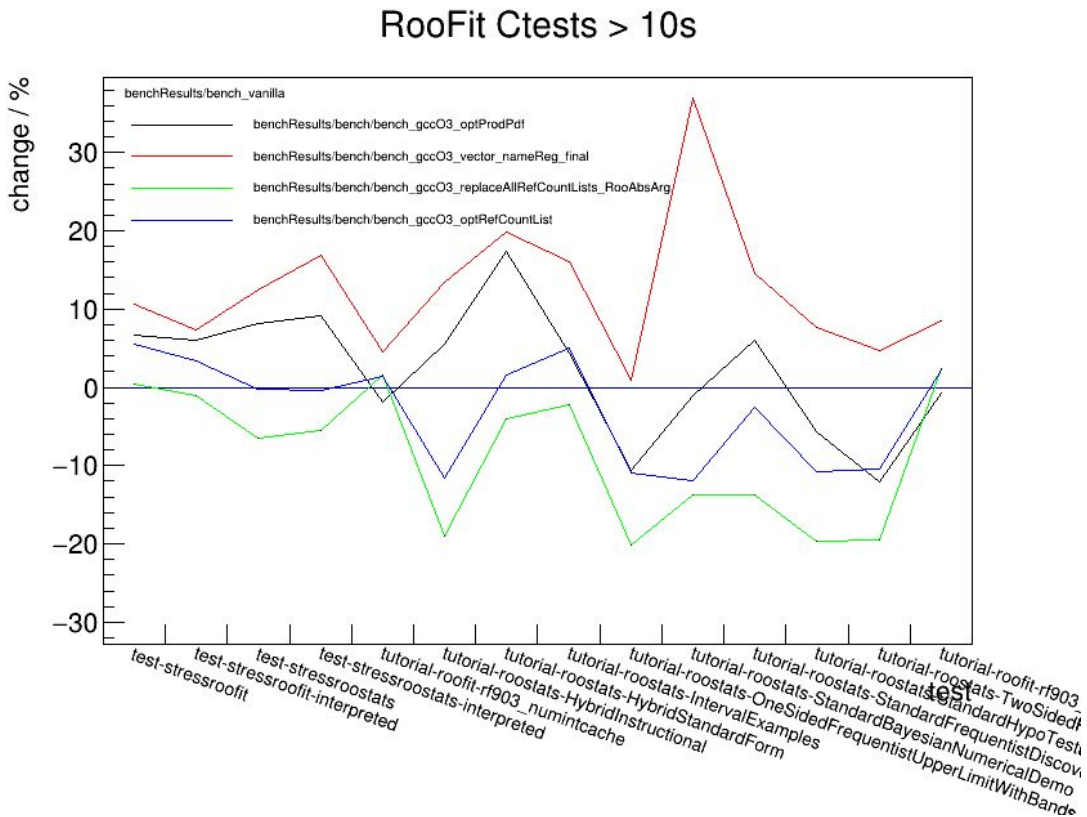


- No standardised benchmarks for RooFit
- My solution:
  - Run all ctests 5x
  - Calculate truncated mean of best 4 runs
- **Result**:
  When legacy iterators are used heavily, RooFit is ~20% slower
- Then: Repeat following cycle
  - VTunes
  - Check what's slow
  - Replace LegacyIterator by `for (auto elm : collection)`

Legacy Iterators are Easy to Find



Speed Tests II



- Couple of iterations of:
  - VTunes
  - Check what's slow
  - Replace LegacyIterator by `for (auto elm : collection)`
- RooFit faster than before
- **To do**:
  - Replace more iterators
  - Old collections only contain RooAbsArg*:

```
for (auto elm : list) {
  auto realVar = static_cast<RooRealVar*>(elm)
```

- New collection is templated!
  - No time to replace properly …

Addendum: RooRefCountList

- While profiling, RooLinkedList still showed up
- Turns out to be a reference-counting list
  - Saves an element and a number
  - When iterating, ref count is not used
  - Fully represented by std::vector<Element_t> std::vector<size_t>

```
class RooRefCountList : public RooLinkedList {
public:
  RooRefCountList() ;
  virtual ~RooRefCountList() {} ;

  virtual void Add(TObject* arg) { Add(arg,1) ; }
  virtual void Add(TObject* obj, Int_t count) ;
  virtual Bool_t Remove(TObject* obj) ;
  virtual Bool_t RemoveAll(TObject* obj) ;
  Int_t refCount(TObject* obj) const;
```

RooRefCountList → RooSTLRefCountList

- RooRefCountList reimplemented as header-only RooSTLRefCountList
- + conversion functions for I/O
- Seamlessly replaces RooRefCountList
  - Should change iterators, though:
    → faster
- Internal to RooFit
  → Users shouldn't notice (just faster)

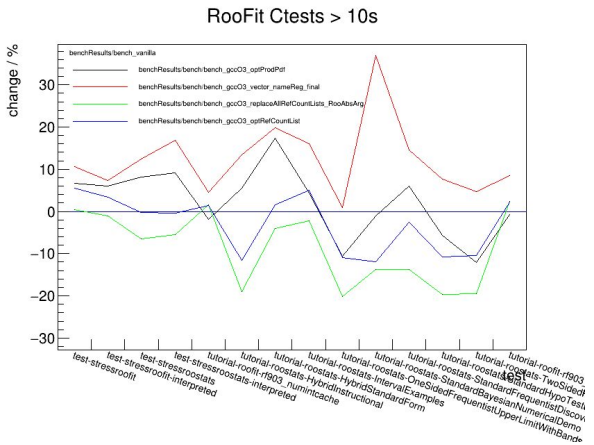
- **To do**:
  - Only slow & heavily-used instances have been replaced

```
template <class T>
class RooSTLRefCountList {
  public:
    using Container_t = std::vector<T*>;
  {...}

  ///Find an item by comparing its adress.
  template<typename Obj_t>
  typename Container_t::const_iterator findByPointer(const Obj_t * item) {
    auto byPointer = [item](const T * listItem) {
      return listItem == item;
    };

    return std::find_if(_storage.begin(), _storage.end(), byPointer);
  }

  {...}

  private:
    Container_t _storage;
    std::vector<std::size_t> _refCount;

    ClassDef(RooSTLRefCountList<T>,1);
};
```

@Axel: Eclipse spell-checks. Already fixed.

Summary I

- RooFit now iterates 20 - 30% faster
  - Real-world example:
    My H → bb thesis measurement
    - 11:30 min → 9:20 min
    - 586 Mb → 579 Mb
    - **Identical result!**
- Old iterators still work
  - Only ~25% replaced in RooFit
  - Replacing makes iteration faster
  - Slowest places presumably taken care of
  - **Caveats:**
    - Modifying collection in front of iterator not possible, any more
    - Can be detected if NDEBUG not set



Summary II

- New iterators look and feel like STL
- Two branches (almost) ready:
  - https://github.com/hageboeck/root/tree/ImproveRooAbsCollection
  - https://github.com/hageboeck/root/tree/ReplaceRooRefCountList
- PRs after cleanup + a bit of doxygen here and there

```
663 void RooAbsCollection::setAttribAll(const Text_t* nam
664 {
665   RooFIter iter= fwdIterator() ;
666   RooAbsArg* arg ;
667   while ((arg=iter.next())) {
668     arg->setAttribute(name,value) ;
669   }
670 }
```

```
643 void RooAbsCollection::setAttribAll(const Text_t* name,
644 {
645   for (auto arg : _list) {
646     arg->setAttribute(name, value);
647   }
648 }
649
650
```

Backup

Speed Tests for Short RooFit Examples