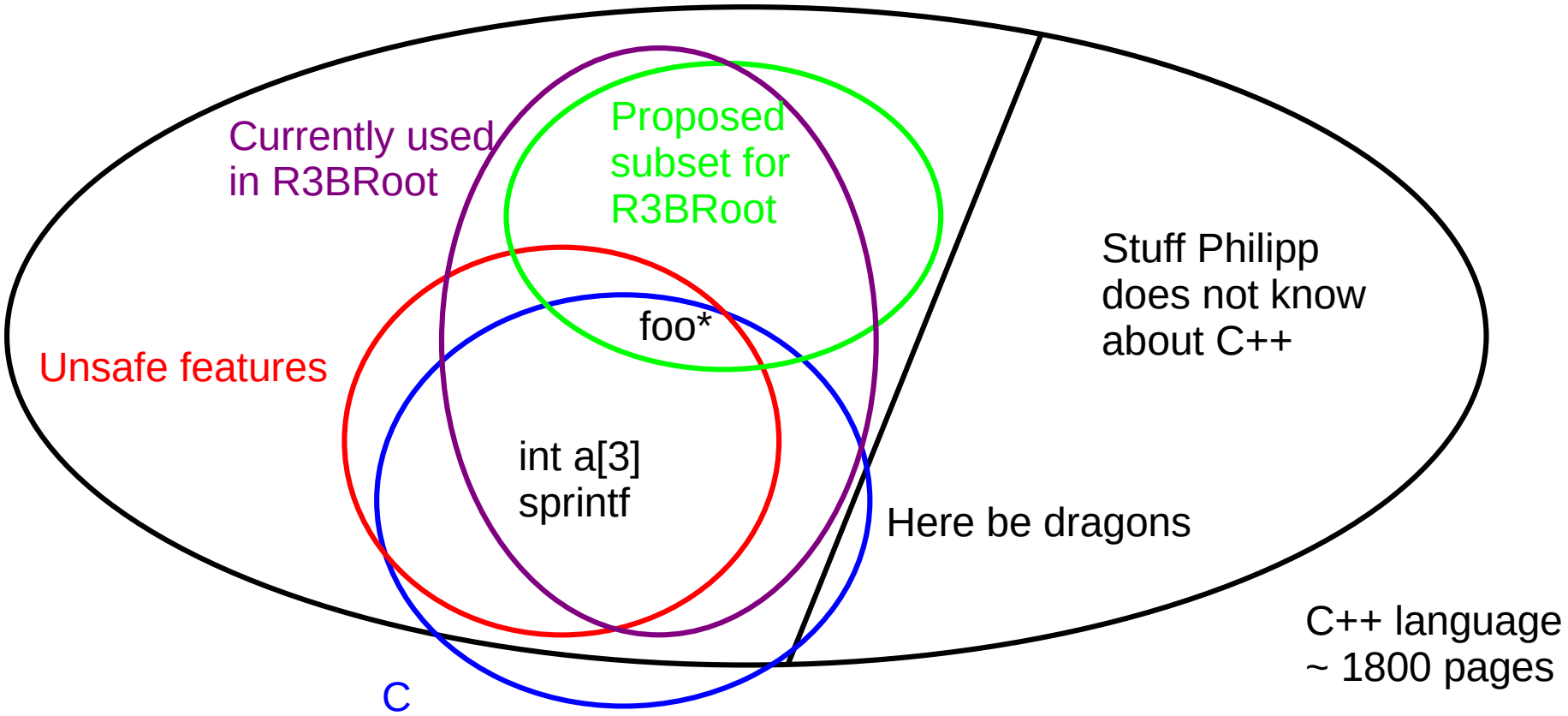
From “ROOT macro” to C++

Method	Advantages	Disadvantages	Conclusion
root -l -q -x macro.C	No headers req. No libs req.	No debug info Terrible error output	PyROOT is strictly better, IMHO
root -l -q -x macro.C+		Needs headers Compilation opaque No debug info?	
g++ $(root-config ...) (optionally with make)	Full control over compilation	Needs headers, libs PITA with R3B classes	Ok if you just need ROOT
CMake with R3BRoot	Full control, can transplant to github version	Separate branch or repo, involves writing CMake files	The proper way
compile_macro.py	No headers, no libs required	python script may have bugs	The quick & dirty way to compile

The sooner we get rid of R3BRoot macros, the sooner we can get rid of most rootcint / LinkDef

How to write safe C++

C++ is a complex language



Undefined runtime behavior

- During runtime, stack and heap are just bytes

• Out of bounds writes • Invalid type casts • Uninitialized pointer deref • Use after free	=>	• Silent data corruption • Segfault • Arbitrary code execution
• Uninitialized data	=>	• Invalid data
• nullptr dereference	=>	• Segfault

Avoiding undefined behavior

Unsafe practice	Can lead to	Safe replacement
1-dimensional C arrays int a[5];	Out of bounds writes	std::vector std::array (with -D_GLIBCXX_ASSERTIONS)
multi-dimensional C arrays double b[5][7][2];	Out of bounds writes	boost::multi_array
C strings char buf[255]; sprintf etc	Out of bounds writes	std::string (or TString)
C style casts auto hit=(FooData*)TCA.At(i);	Invalid type casts	dynamic_cast TFile::Get<> STL containers

Replacing 1-dim C arrays

- dynamic size: std::vector

```
std::vector<double> a; // empty vector

auto b=std::vector<double>(100, NAN);
// 100 elements, all NAN
std::vector<R3BFooData*> c;
```

- static size: std::array

```
std::array<double, 8> c; // 8 elements, zero initialized
```

- use a.data() if you have to get the underlying C array (e.g. for ROOT)

Replacing multi-dimensional C arrays

- Option 1: use std::vector<std::vector<double>>
  - Will get unreadable rather quickly
- Option 2: use boost::multi_array;

Replacing char* strings

- Many options: std::string, TString
- sprintf -> std::ostringstream
- If you need precise formatting (e.g. “%02d”):
  - boost::format (beware .str().c_str())
  - ROOT’s Form (unclear memory ownership)
- Convert to char* by calling mystdstring.c_str()

Replacing C style casts

- auto foo=(R3BFooData*)(obj);
- auto foo=dynamic_cast<R3BFooData*>(obj);
  assert(foo && “cast failed!”);
- template TDirectory::Get may be shorter

Your turn

- git clone git@github.com:klenze/cxxsamples.git
- cat c_stuff/readme
- Note that these slides are contained in ./slides/ for reference
- You are allowed to use google, cppreference, boost reference etc
- Happy hacking