LLVM Data-structures overview

LLVM Data-structures

Marcello Maggioni

[1]Codeplay Software Ltd.

EuroLLVM 2014

Outline

1. Motivation
   - Why having specific data-structures
   - LLVM Resources
2. Data-structures
   - Vectors
   - Maps
   - Sets

Outline

1. Motivation
   - Why having specific data-structures
   - LLVM Resources
2. Data-structures
   - Vectors
   - Maps
   - Sets

Why not using standard structures?

- C++ Standard data-structures have performance that is platform dependent
- C++ Standard might not have a specific kind of data structures (like HashMaps. With C++11 this problem was solved)
- Specialized data-structures can be made faster than the Standard generic ones

Outline

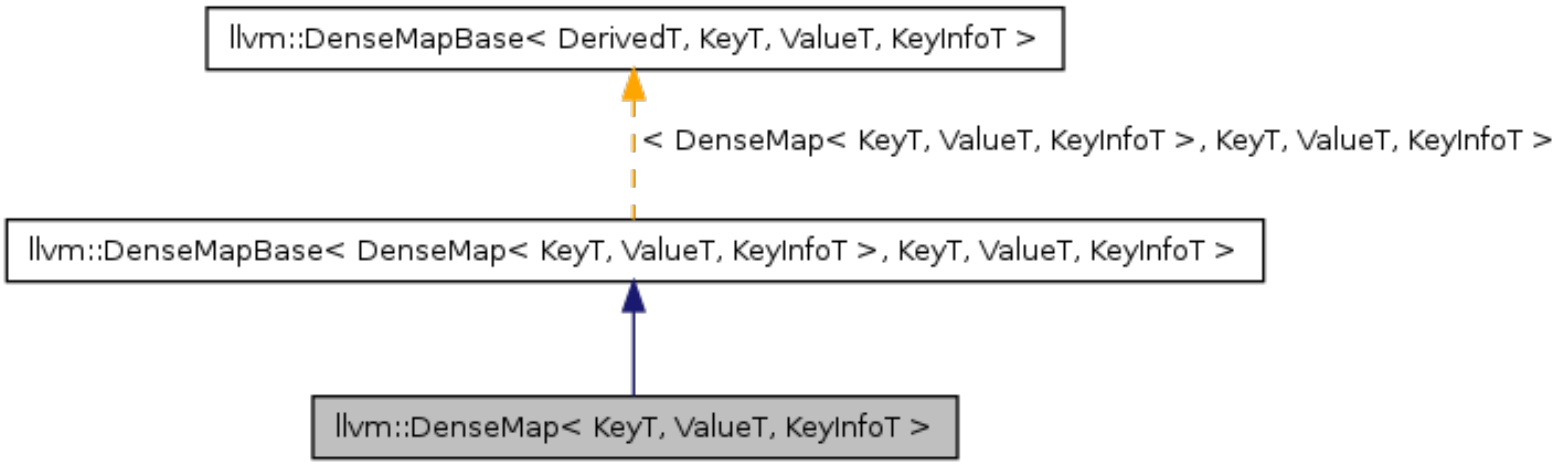
1. Motivation
   - Why having specific data-structures
   - LLVM Resources
2. Data-structures
   - Vectors
   - Maps
   - Sets

Resources on LLVM Data-Structures

- http://llvm.org/docs/ProgrammersManual.html
- http://llvm.org/docs/doxygen/html/

Looking at Doxygen info

- Look for methods in every subclass
- The most exposed interface does not expose all methods in documentation usually



Outline

1. Motivation
   - Why having specific data-structures
   - LLVM Resources
2. Data-structures
   - Vectors
   - Maps
   - Sets

Possible Choices

- LLVM SmallVector
- std::vector

SmallVector

- Vector-like data structure
- Is optimized to contain a fixed amount of elements
- It is flexible if more elements are added
- Interface similar to std::vector

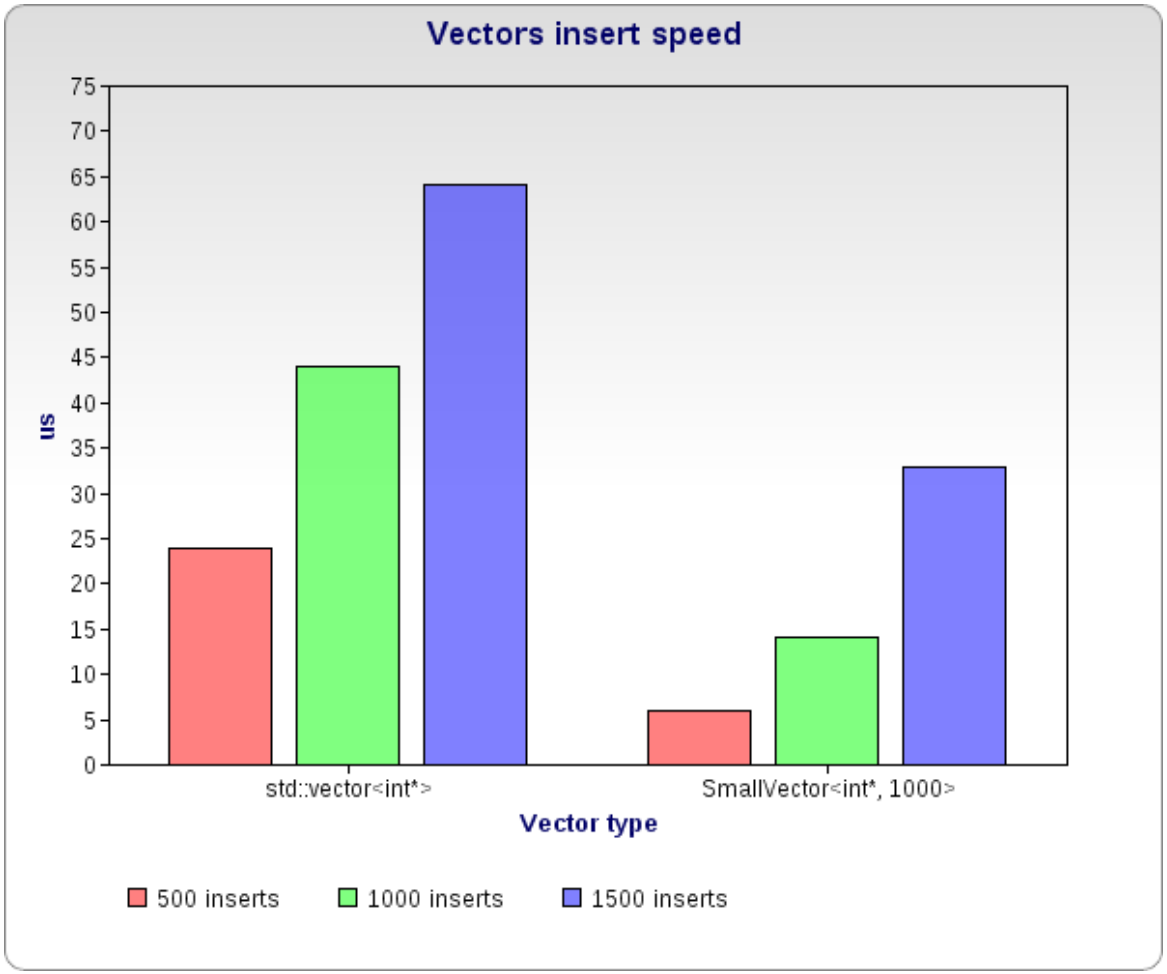
SmallVector

```
#include "llvm/ADT/SmallVector.h"

SmallVector<type, N> V;

void foo(SmallVectorImpl<type> &V) {
}
```

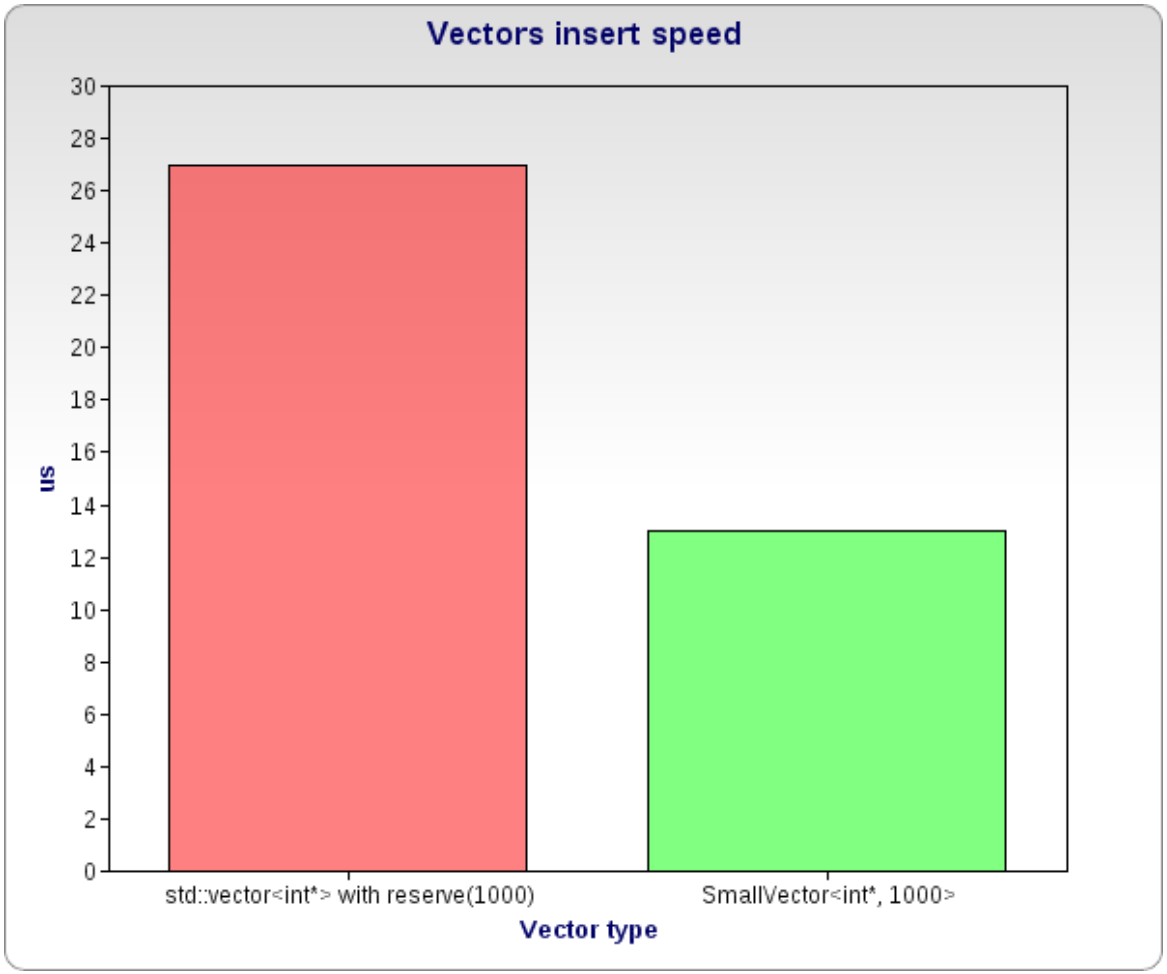
SmallVector

```
SmallVector<Instruction*, 10> WorkList;

for (...) {
  Instruction *I = ...;
  WorkList.push_back(I);
}
...
while (WorkList.empty()) {
  Instruction *I = WorkList.pop_back_val();
  ...

}
```

SmallVector vs World



SmallVector vs World



Outline

1. Motivation
   - Why having specific data-structures
   - LLVM Resources
2. Data-structures
   - Vectors
   - Maps
   - Sets

Possible Choices

- LLVM DenseMap
- LLVM StringMap (only for strings)
- std::map
- std::unordered_map

DenseMap

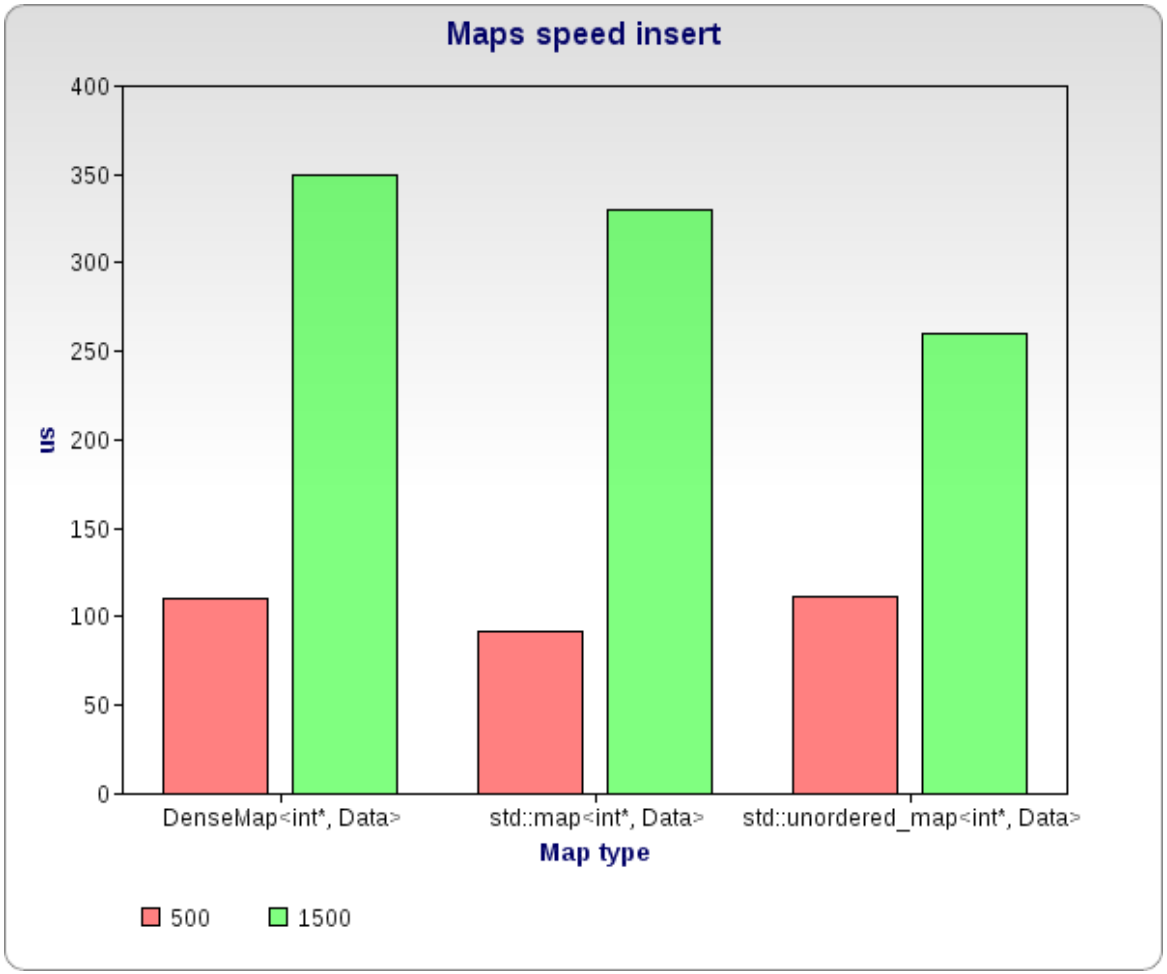
- DenseMap is a quadratically probed HashMap.
- Keeps everything in a single memory allocation
- Iterators potentially invalidated after insertion
- Matches pretty closely std::map interface
  - insert(std::pair<>)
  - find(Key&)
  - count(Key&)
  - begin(), end() (unordered)

DenseMap Keys

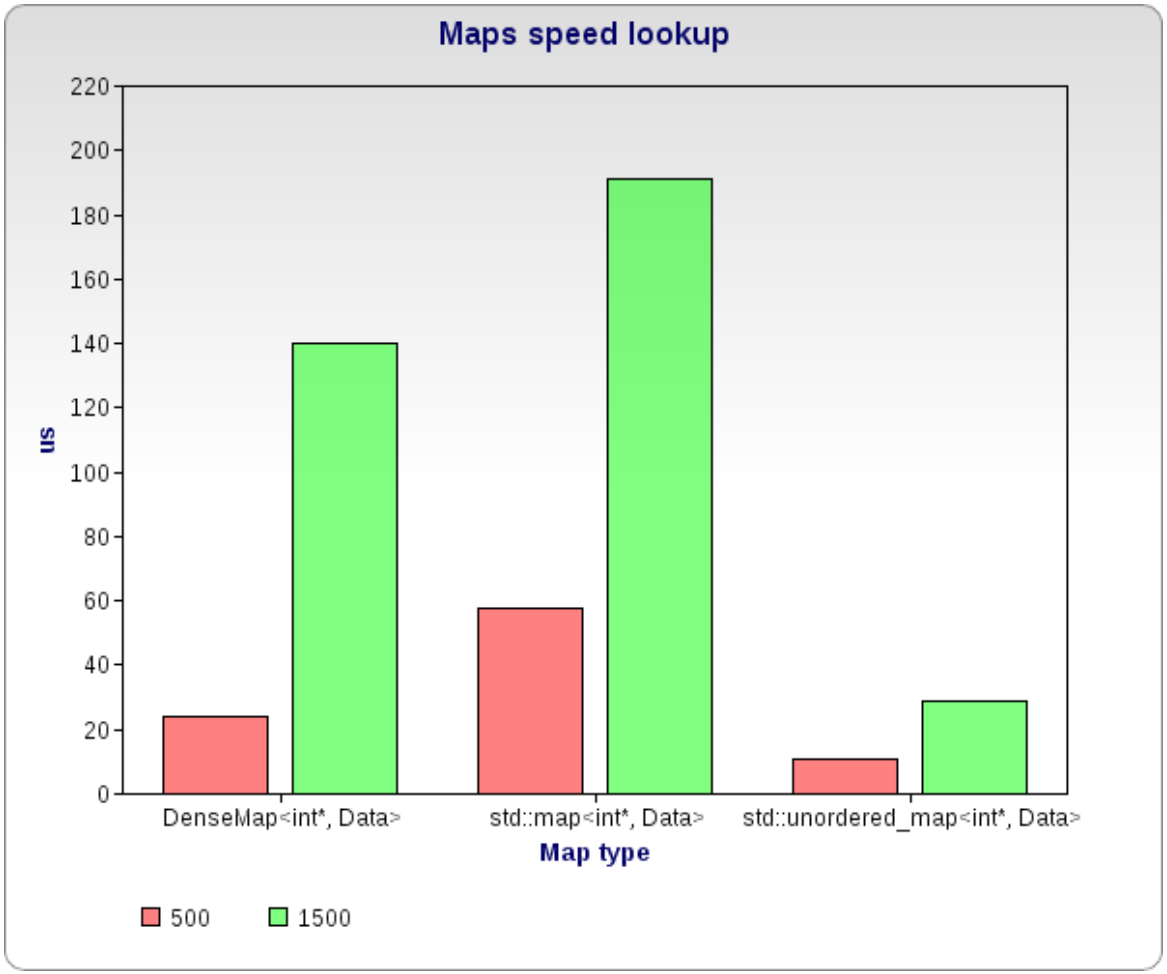
- Supports all pointers and integer types as keys
- Additional Key types can be specified defining a custom DenseMapInfo<> class

```
struct KeyInfo {
  static inline T getEmptyKey() {...}
  static inline T getTombstoneKey() {...}
  static unsigned getHashValue(const T &Val) {...}
  static bool isEqual(const T &LHS, const T &RHS)
      {...}
};

DenseMap<Key, Value, KeyInfo> M;
```

DenseMap vs World



DenseMap vs World



StringMap

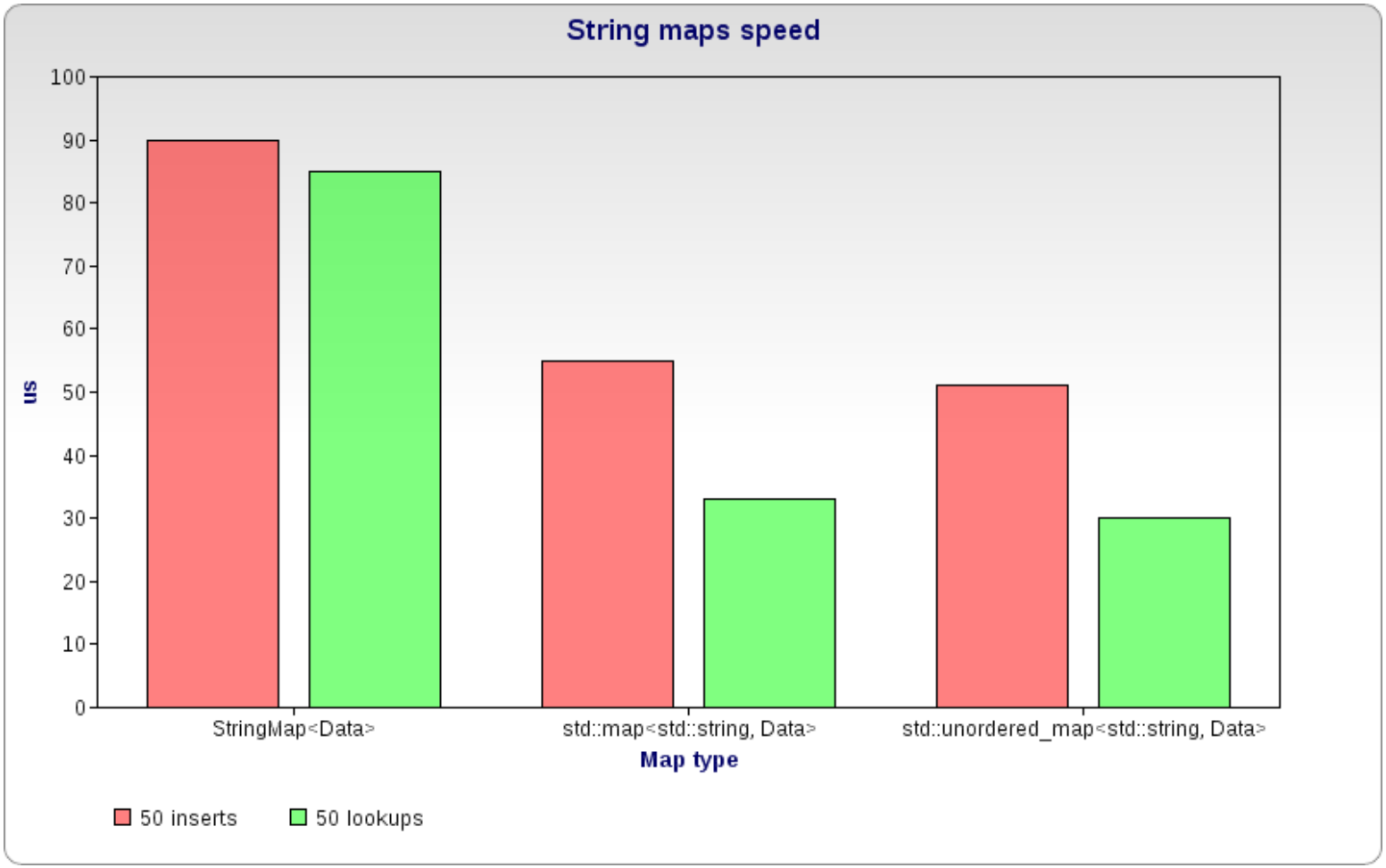
- Specific implementation of an HashMap only for having strings as keys
- Strings are copied into the map. They don't store the pointer to the map as a key.
- Similar interface to DenseMap
- Insert is different though ... it is actually called GetOrCreateValue()

StringMap

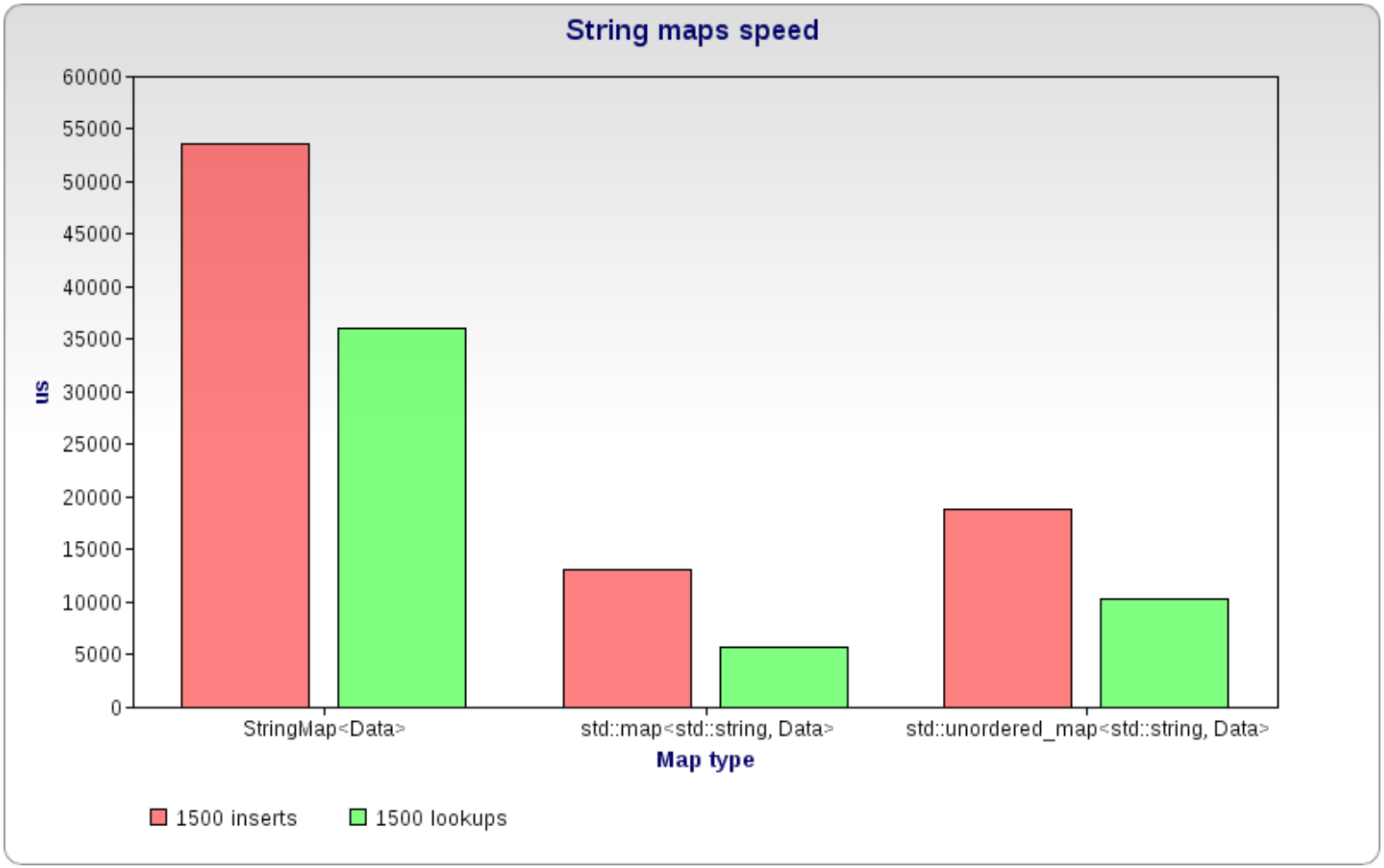
```
const char *str = "__some_symbol";

StringMap<Data> Map;
Data D = { 10, 5 };

Map.GetOrCreateValue(str, D);
Map[str] = D;
Map.find(str);
Map.count(str);
```

StringMap vs World

- Storing a 16 character wide random string



StringMap vs World



Outline

1. Motivation
   - Why having specific data-structures
   - LLVM Resources
2. Data-structures
   - Vectors
   - Maps
   - Sets

Possible Choices

- Sorted vectors
- LLVM SmallSet
- std::set
- std::unordered_set

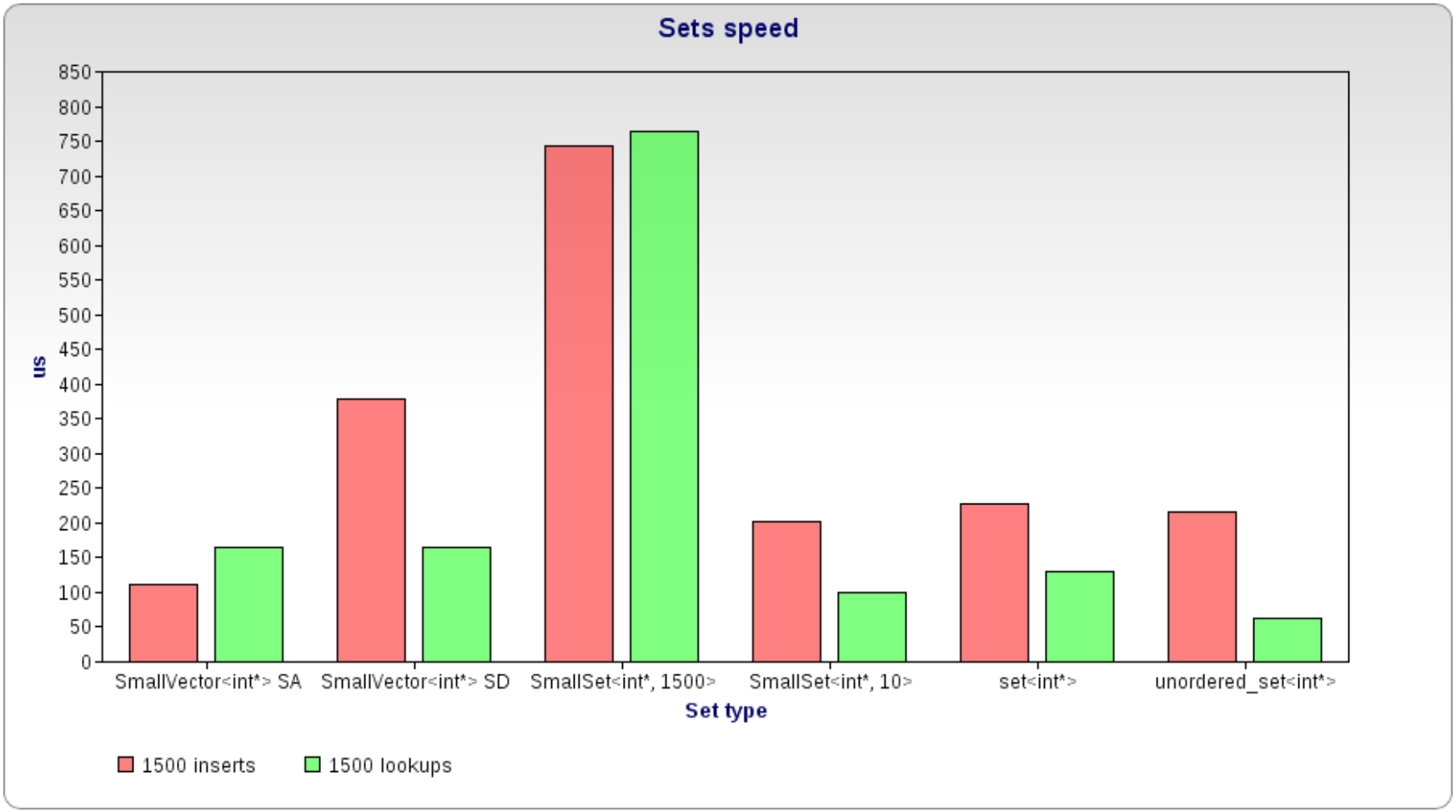
SmallSet

- Replacement for set in LLVM
- It is implemented as a small vector of fixed size that is not sorted
- Searches are linear in time
- When exceding the specified size switches to a quadratically probed set for some keys and std::set for others
- Cannot be iterated, only for querying

SmallSet

```
SmallSet<int*, 10> S
int a;
S.insert(&a);
S.count(&a);
```

SmallSet vs World



Summary

- Covered basic data structures and their performance comparisons
- Other data structures are available for specific needs (BitVectors, SparseSet, ValueMap ...)
- Using LLVM data-structures can give performance portabilty
- LVM data-structures are not always faster and may require parameter tuning

Contacts

Marcello Maggioni

- marcello.maggioni@gmail.com
- marcello@codeplay.com