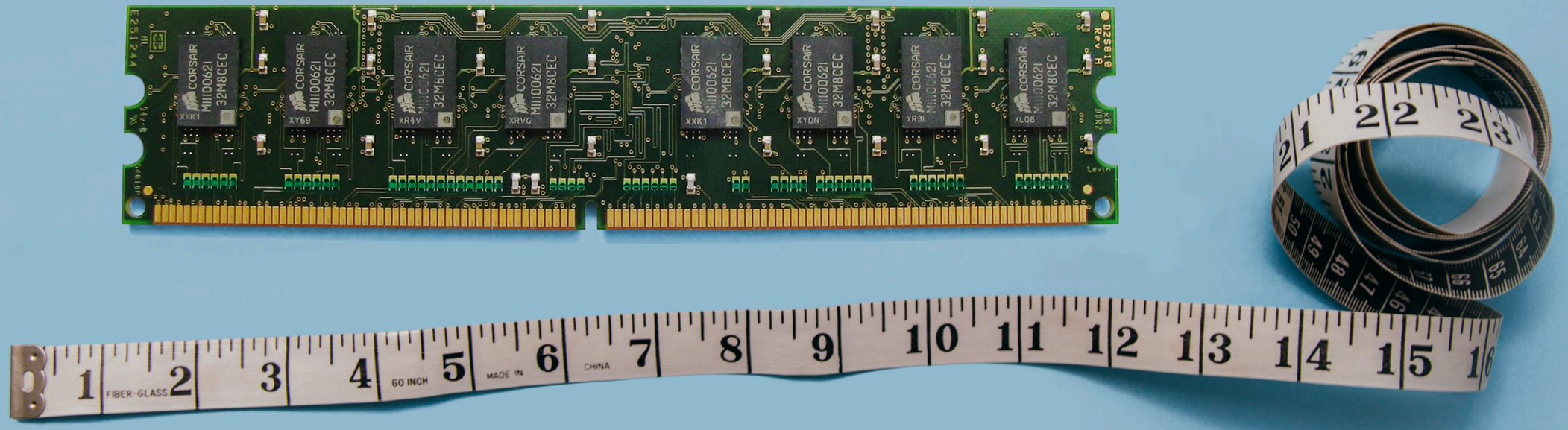
Memoro

Scaling an LLVM-Based Heap Profiler



Thierry Treyer
Performance & Capacity Intern

Mark Santaniello
Performance & Capacity Engineer

James Larus
EPFL IC School Dean
EPFL

```
vector<BigT> getValues(
    map<Id, BigT>& largeMap,
    vector<Id>& keys) {

  vector<BigT> values;
  values.reserve(largeMap.size());

  for (const auto& key: keys)
    values.emplace_back(largeMap[key]);

  return values;
}
```

40 GiB

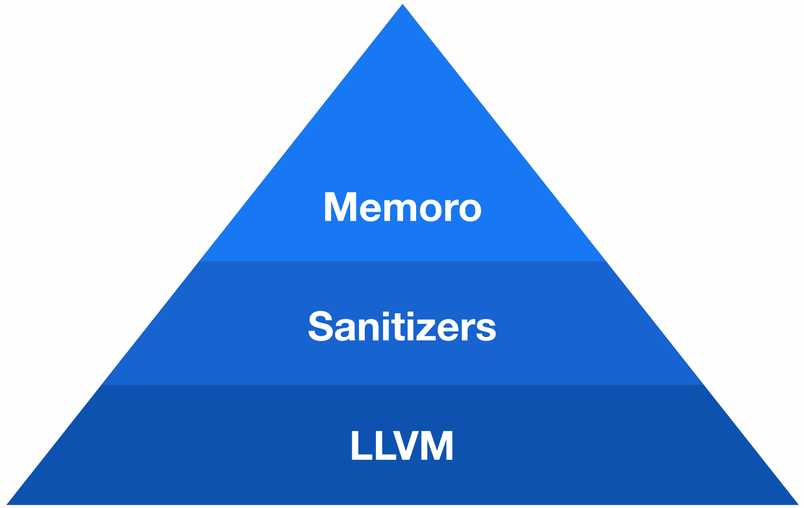
of DRAM wasted per server

```
vector<BigT> getValues(
    map<Id, BigT>& largeMap,
    vector<Id>& keys) {

  vector<BigT> values;
  values.reserve(largeMap.size());

  for (const auto& key: keys)
    values.emplace_back(largeMap[key]);

  return values;
}
```

```
vector<BigT> getValues(
    map<Id, BigT>& largeMap,
    vector<Id>& keys) {

  vector<BigT> values;
  values.reserve(largeMap.size());

  for (const auto& key: keys)
    values.emplace_back(largeMap[key]);

  return values;
}
```

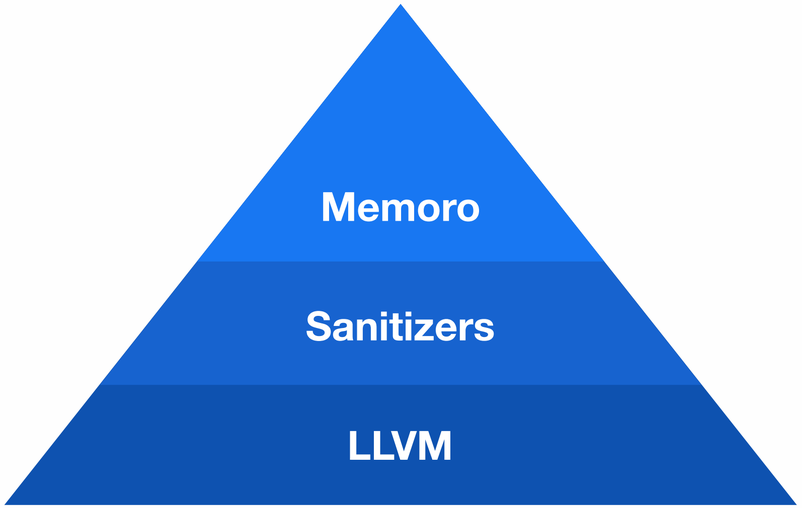
```
vector<BigT> getValues(
    map<Id, BigT>& largeMap,
    vector<Id>& keys) {

  vector<BigT> values;
  values.reserve(keys.size());

  for (const auto& key: keys)
    values.emplace_back(largeMap[key]);

  return values;
}
```

LLVM-Based Profiler



LLVM-Based Profiler

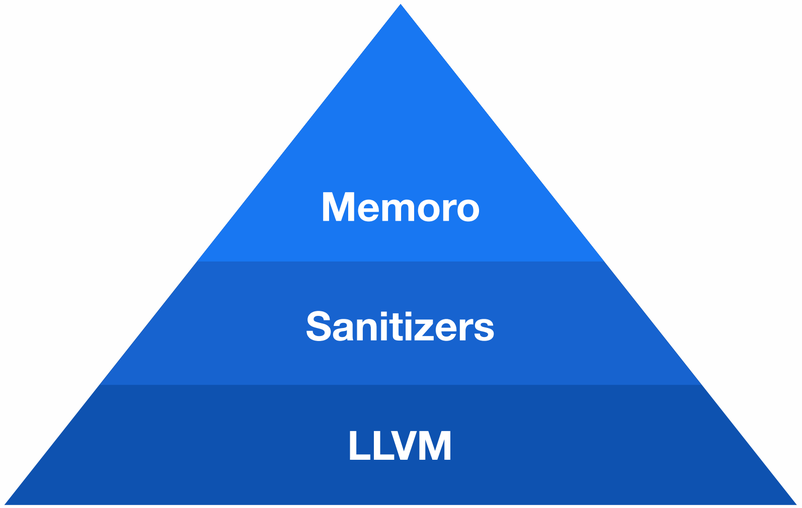
Manipulate the IR



LLVM-Based Profiler

Infrastructure

Manipulate the IR

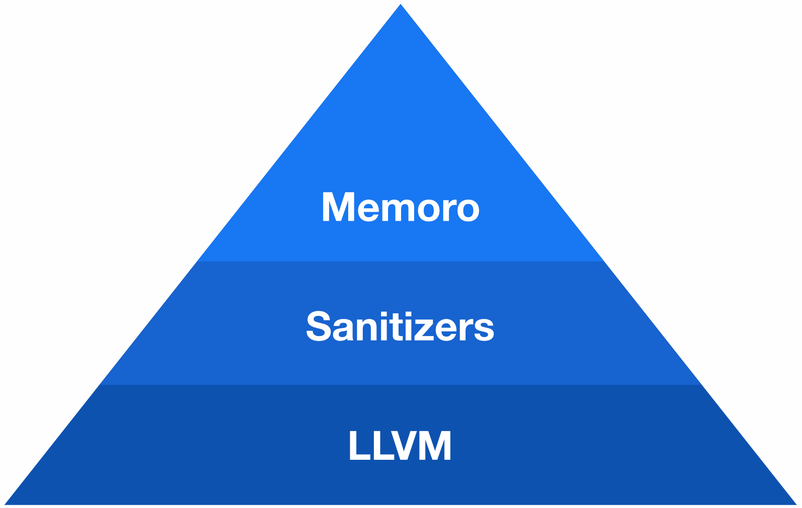


LLVM-Based Profiler

Collecting and Displaying data

Infrastructure

Manipulate the IR



Memoro +

Run-Time Overhead

Visualizer

Open Challenges

Overview

Overview

- No modification

Overview

INSTRUMENTATION PASS
(LLVM)

Source Code

- No modification

Compile

- Instrument loads/stores
- Instrument intrinsics
- Collect types

Run

Analyze

Overview

INSTRUMENTATION PASS (LLVM)

RUN-TIME (COMPILER-RT)

Source Code

Compile

Run

Analyze

- No modification

- Instrument loads/stores
- Instrument intrinsics
- Collect types

- Intercept alloc/free
- Intercept loads/stores
- Intercept syscalls
- Collect stats

Overview

Source Code	Compile	Run	Analyze
INSTRUMENTATION PASS (LLVM)		RUN-TIME (COMPILER-RT)	VISUALIZER (ELECTRON)
○ No modification	○ Instrument loads/stores ○ Instrument intrinsics ○ Collect types	○ Intercept alloc/free ○ Intercept loads/stores ○ Intercept syscalls ○ Collect stats	○ Score AP ○ Guide exploration

Overview

INSTRUMENTATION PASS (LLVM)

RUN-TIME (COMPILER-RT)

VISUALIZER (ELECTRON)

Source Code

- No modification

Compile

- Instrument loads/stores
- Instrument intrinsics
- Collect types

Run

- Intercept alloc/free
- Intercept loads/stores
- Intercept syscalls
- Collect stats

Analyze

- Score AP
- Guide exploration

Memoro +

Run-Time Overhead

Visualizer

Open Challenges

Memoro +

Run-Time Overhead

Visualizer

Open Challenges

1,000x

slowdown due to Memoro's run-time

Run-Time Sampling

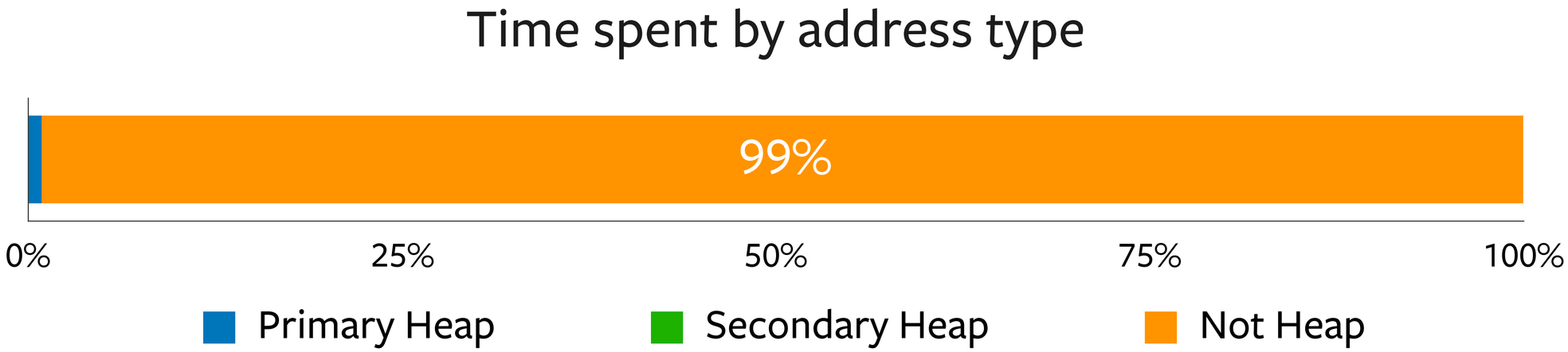
```
int sample_count = 0;

void interceptLoadStore(…) {
  // Sample accesses
  if (sample_count++ % access_sampling_rate != 0)
    return;

  /* Process access... */
}
```

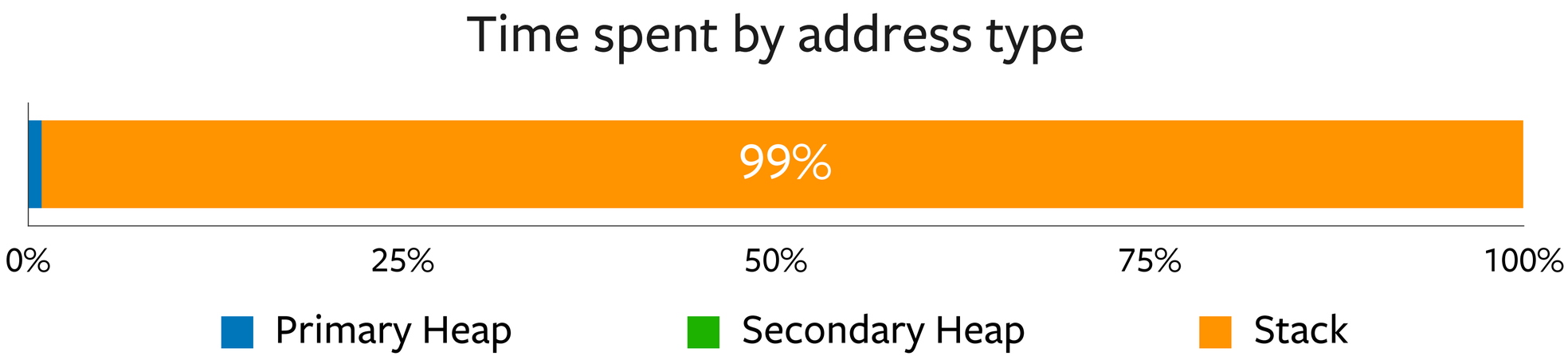
Run-Time Sampling

```
THREADLOCAL int sample_count = 0;

void interceptLoadStore(…) {
  // Sample accesses
  if (sample_count++ % access_sampling_rate != 0)
    return;

  /* Process access... */
}
```

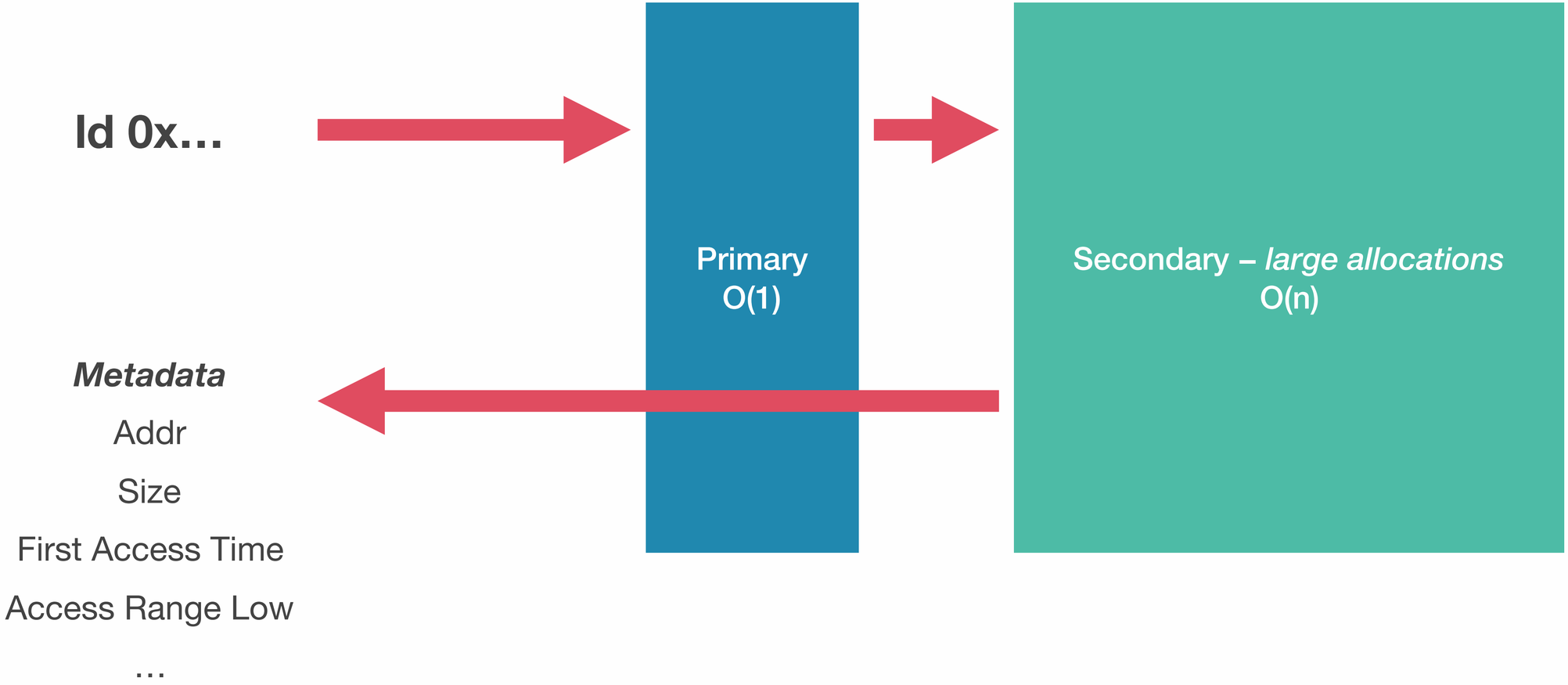
Power to the user!

```
MEMORO_OPTIONS="…" ./myapp
– access_sampling_rate
– ...

// Public API: memoro_interface.h
#include <memoro_interface.h>
void foo(…) {
  MemoroFlags *mflags = memoro::getFlags();
  mflags->access_sampling_rate = 50;
  /* ... */
}
```

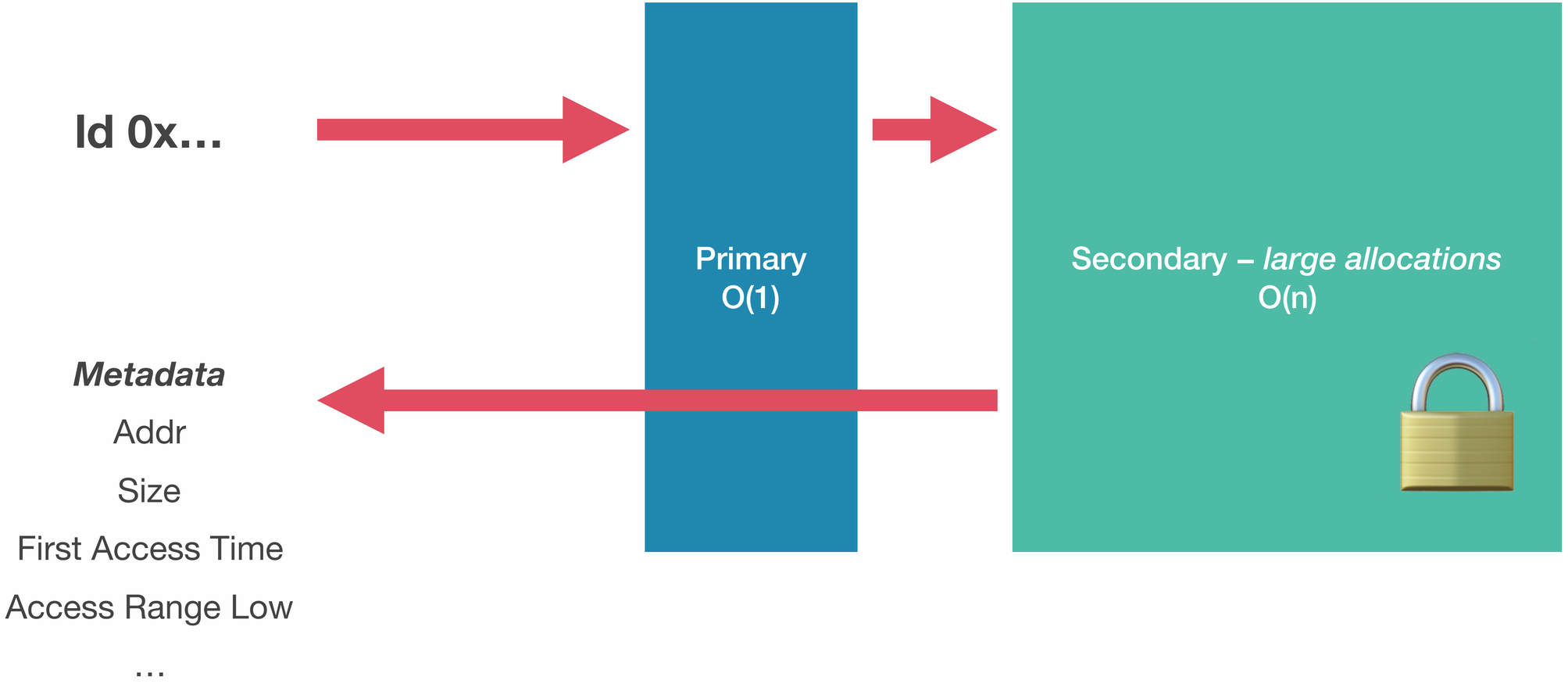

Time spent by address type
99%
0%
25%
50%
75%
100%
Primary Heap
Secondary Heap
Not Heap


Time spent by address type
99%
0%
25%
50%
75%
100%
Primary Heap
Secondary Heap
Stack

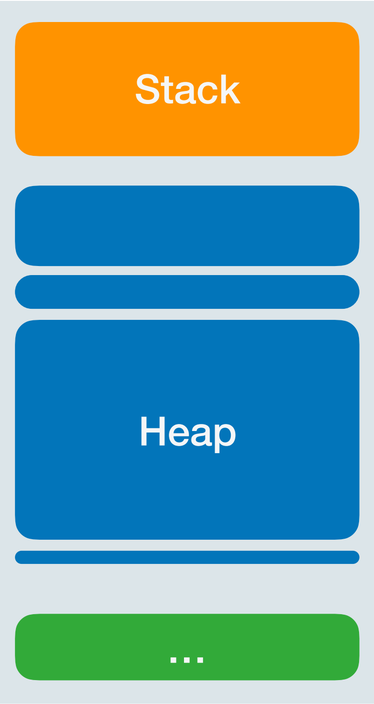
The Allocators



The Allocators

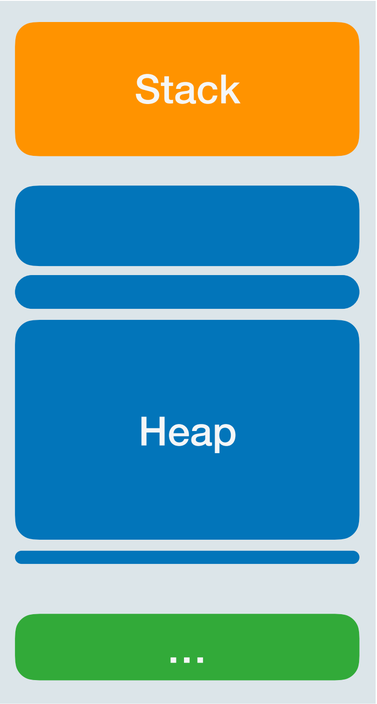


Issue with non-heap addresses



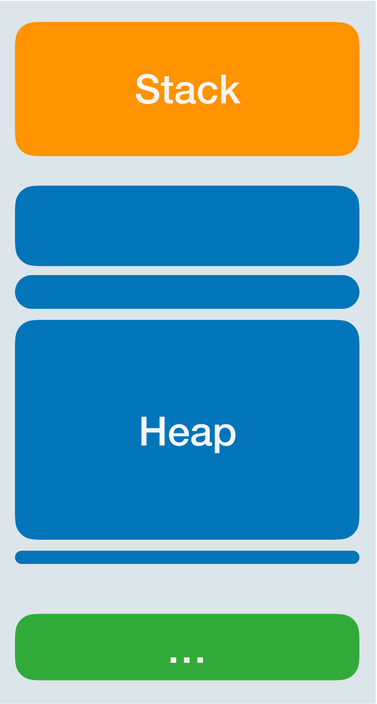
Issue with non-heap addresses

1. Allocators only know about heap



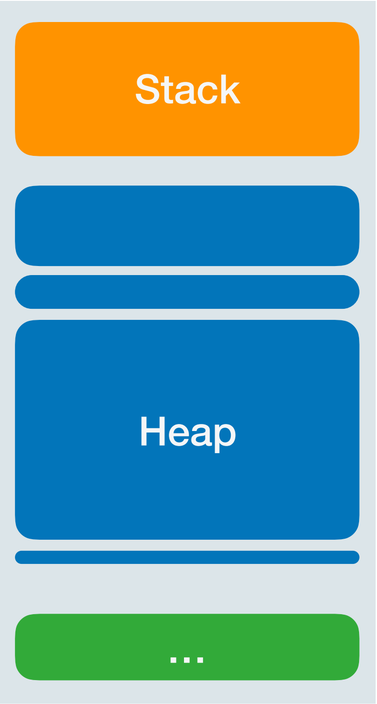
Issue with non-heap addresses

1. Allocators only know about heap
2. Traverse all allocations to discard them



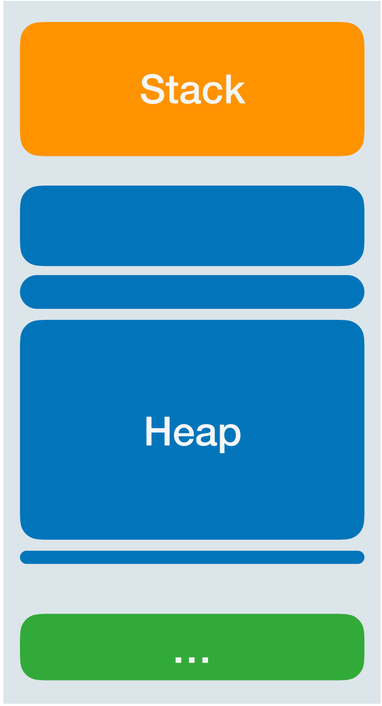
Issue with non-heap addresses

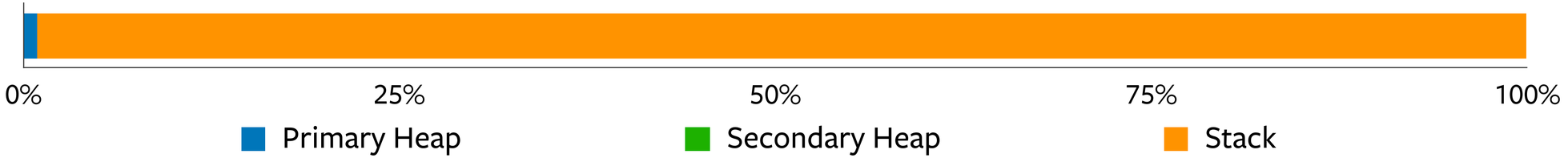
1. Allocators only know about heap
2. Traverse all allocations to discard them
3. Takes a global lock



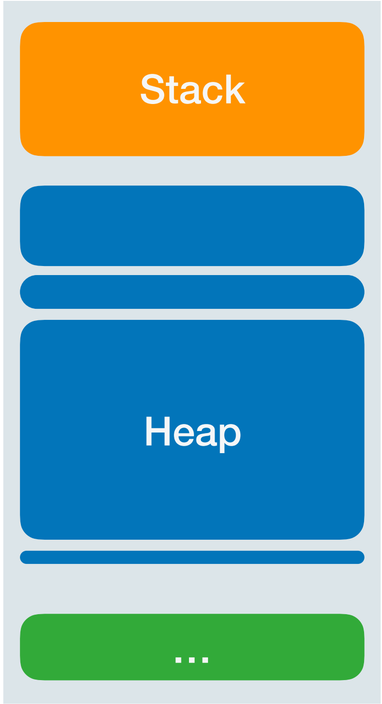
Issue with non-heap addresses

1. Allocators only know about heap
2. Traverse all allocations to discard them
3. Takes a global lock



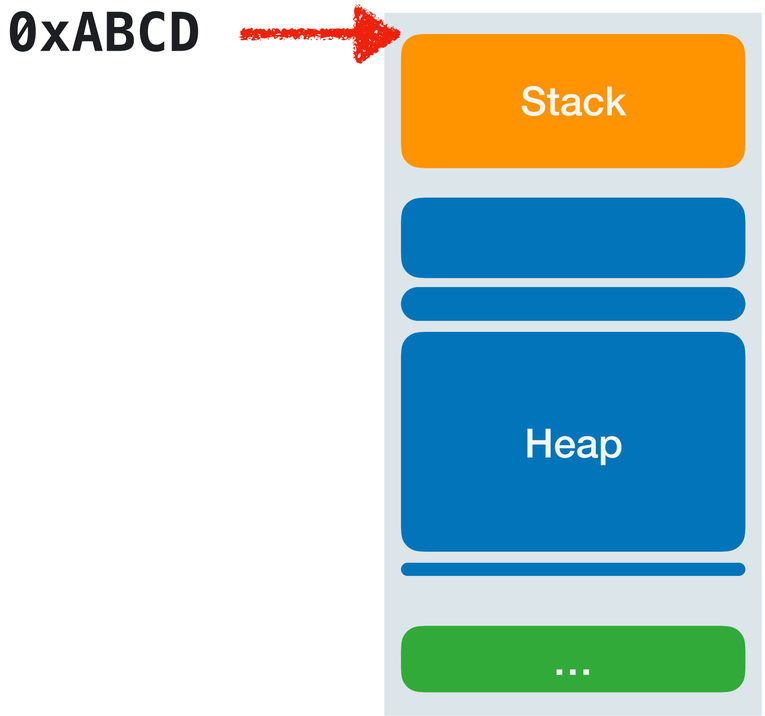


Run-Time Filter



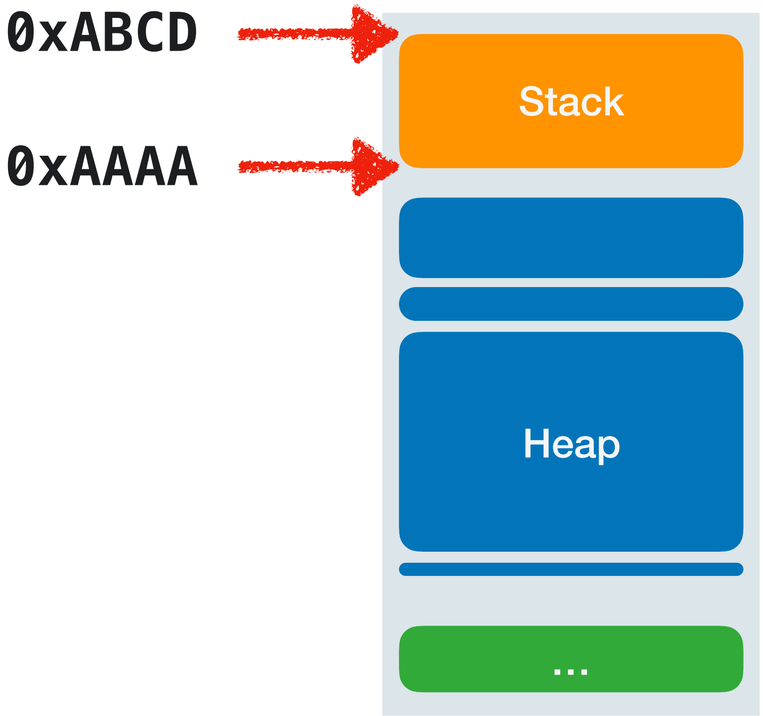
Run-Time Filter

1. *Thread start*: store stack top



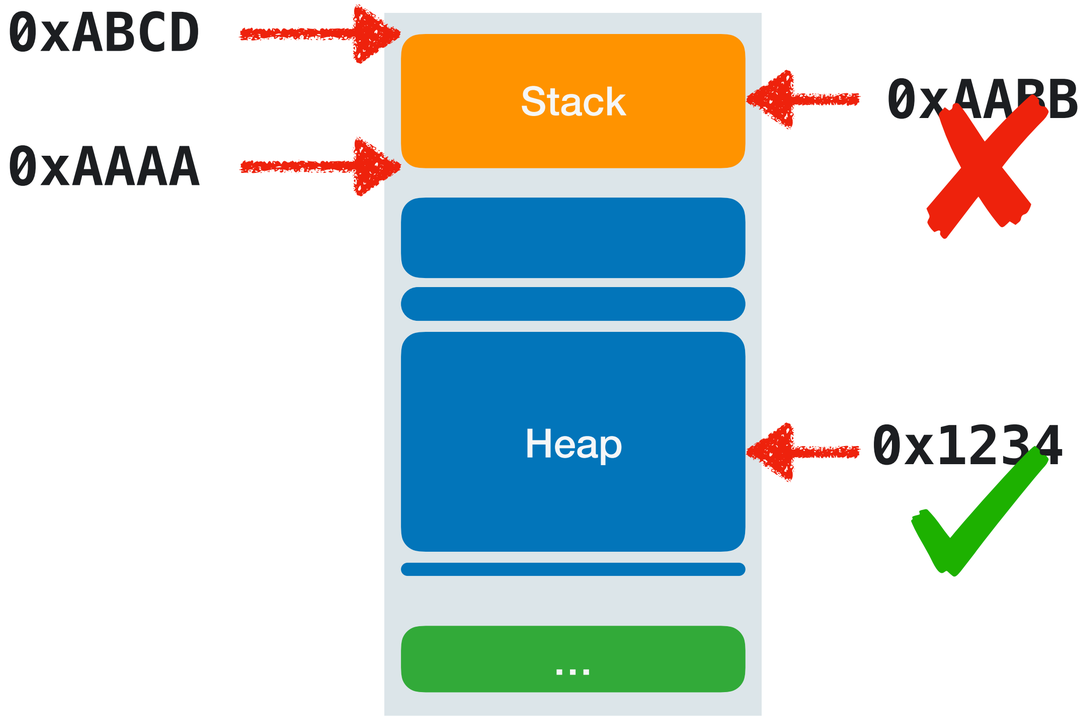
Run-Time Filter

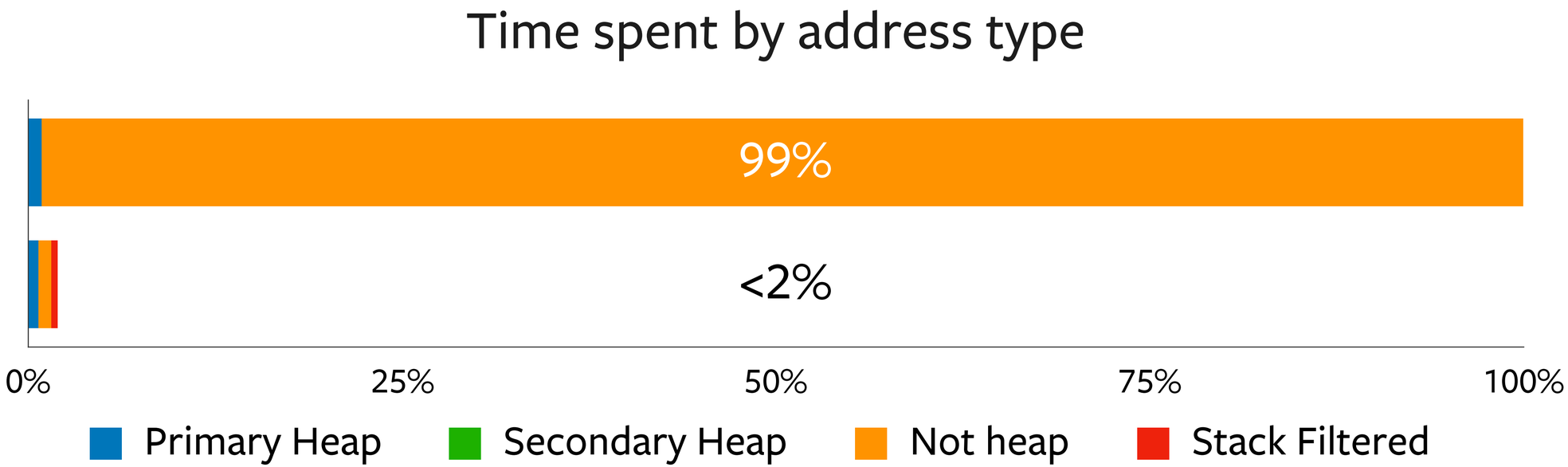
1. *Thread start*: store stack top
2. Get current stack bottom



Run-Time Filter

1. *Thread start*: store stack top
2. Get current stack bottom
3. Discard if Addr. in this range




Time spent by address type
99%
<2%
0%
25%
50%
75%
100%
Primary Heap
Secondary Heap
Not heap
Stack Filtered

1,000x

slowdown due to Memoro's run-time

5x

slowdown due to Memoro's run-time

Memoro +

Run-Time Overhead

Visualizer

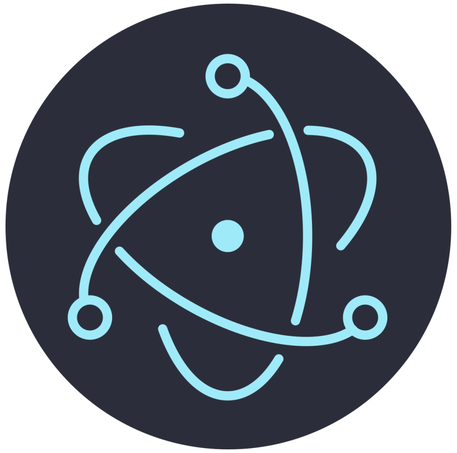
Open Challenges

Memoro +

Run-Time Overhead

Visualizer

Open Challenges

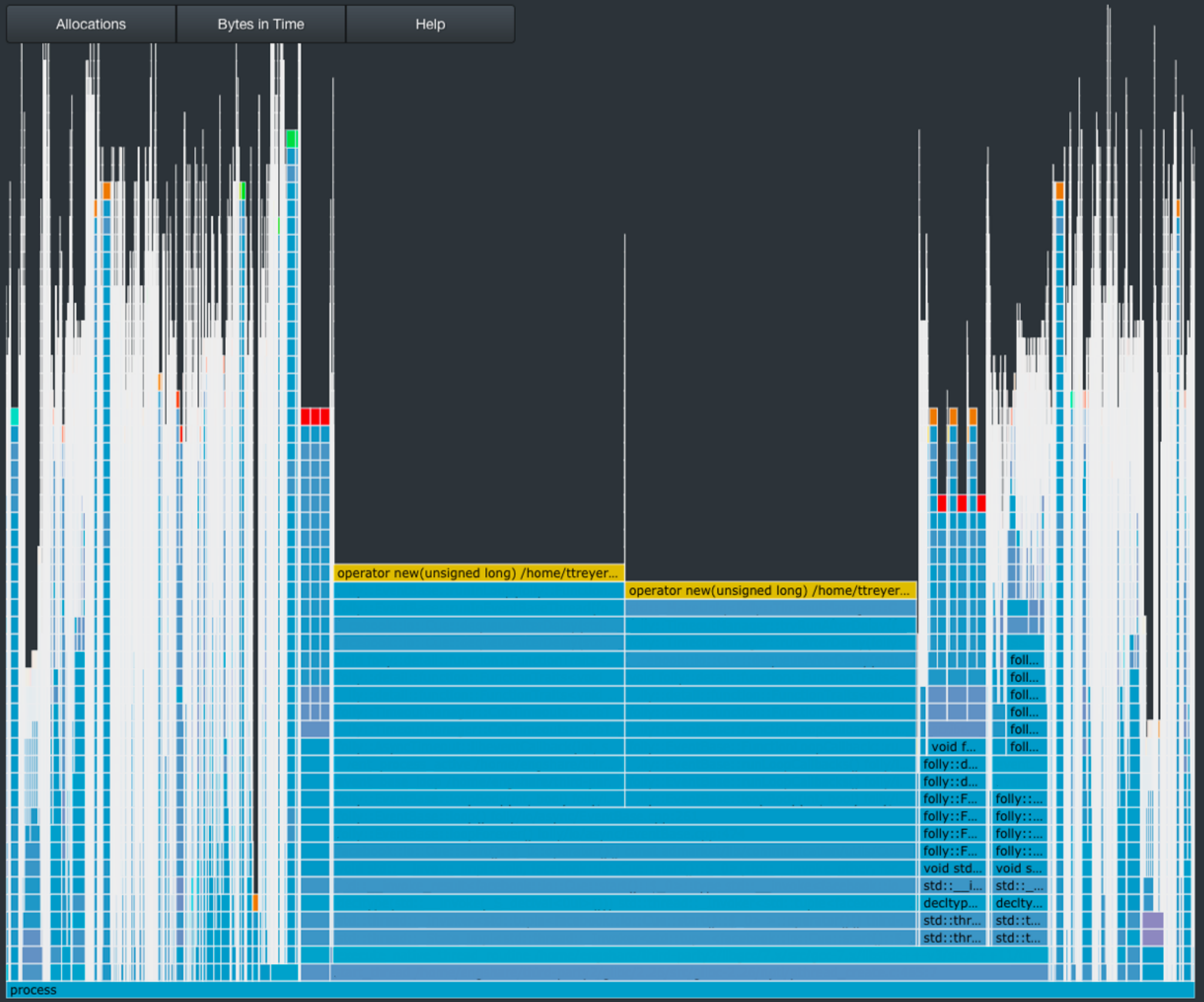


+ **100,000**

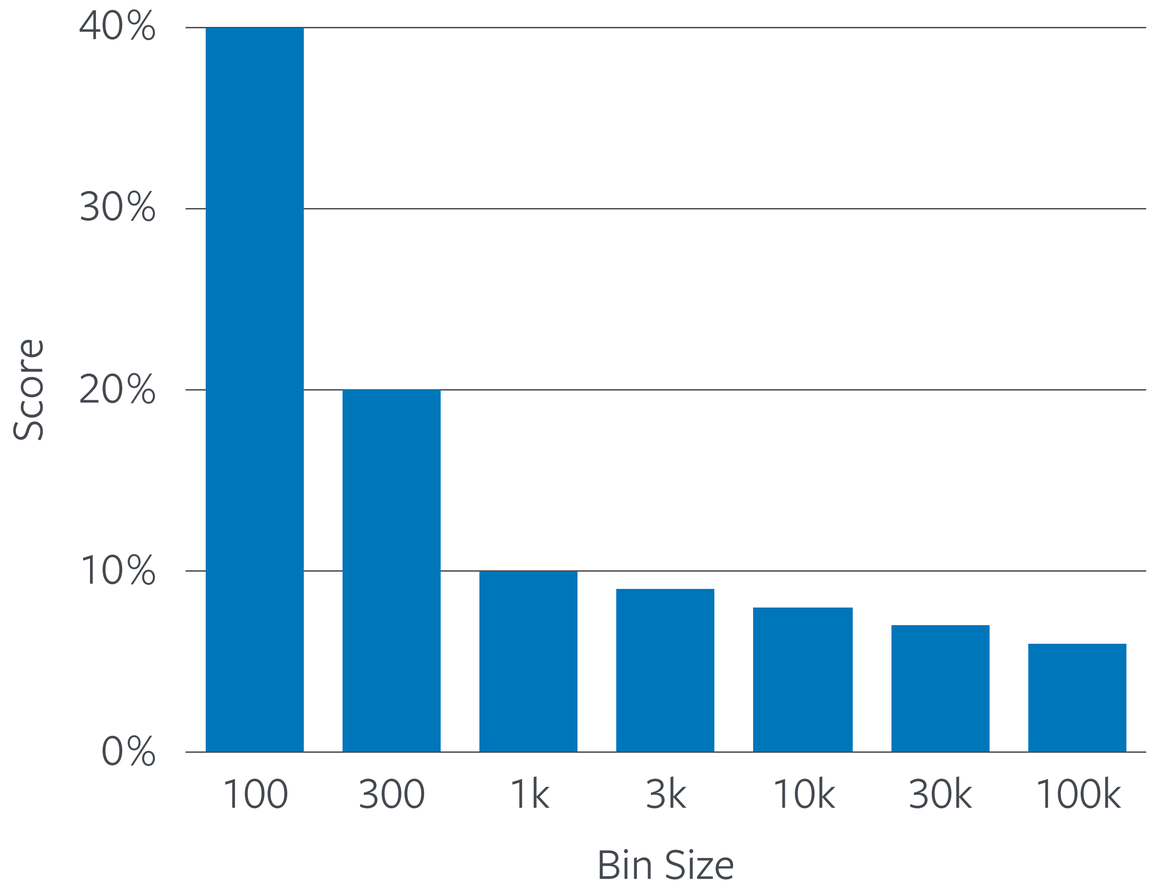
Stack Traces

+ **1B**

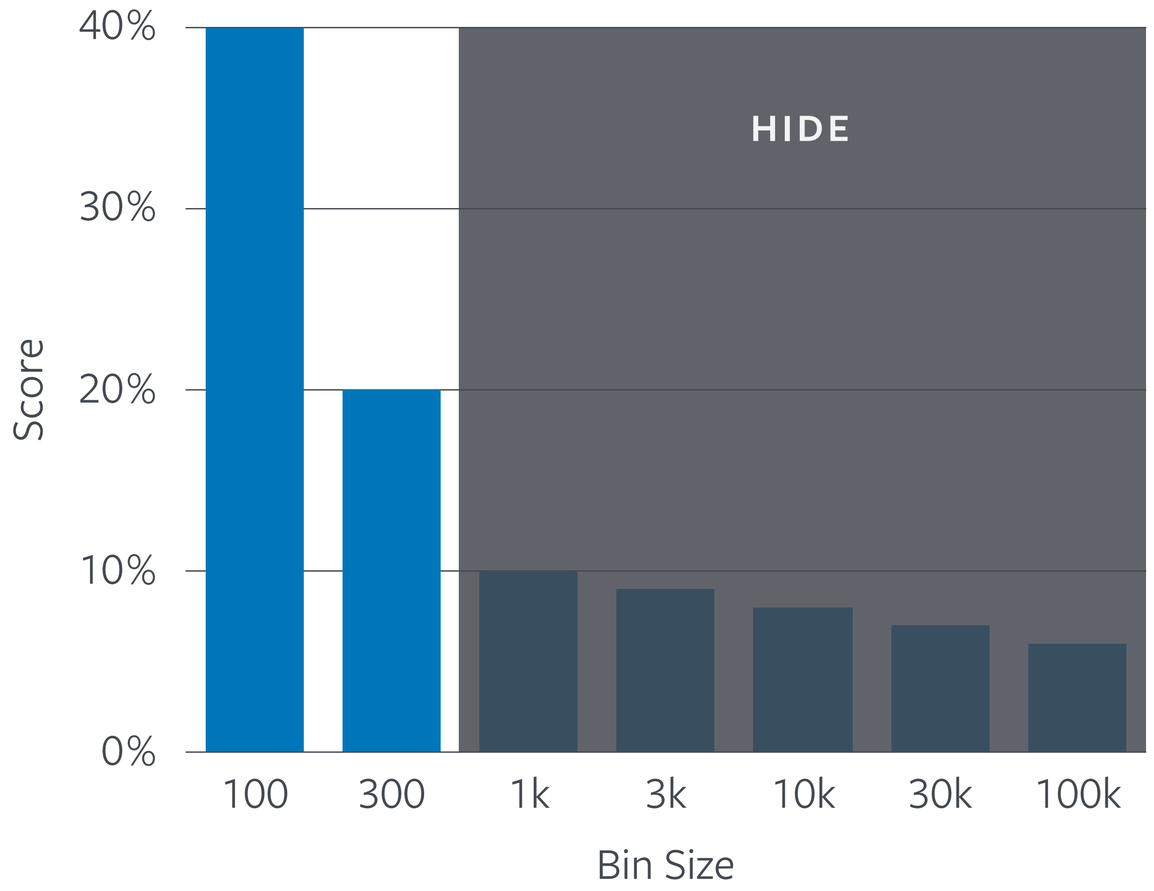
Allocations

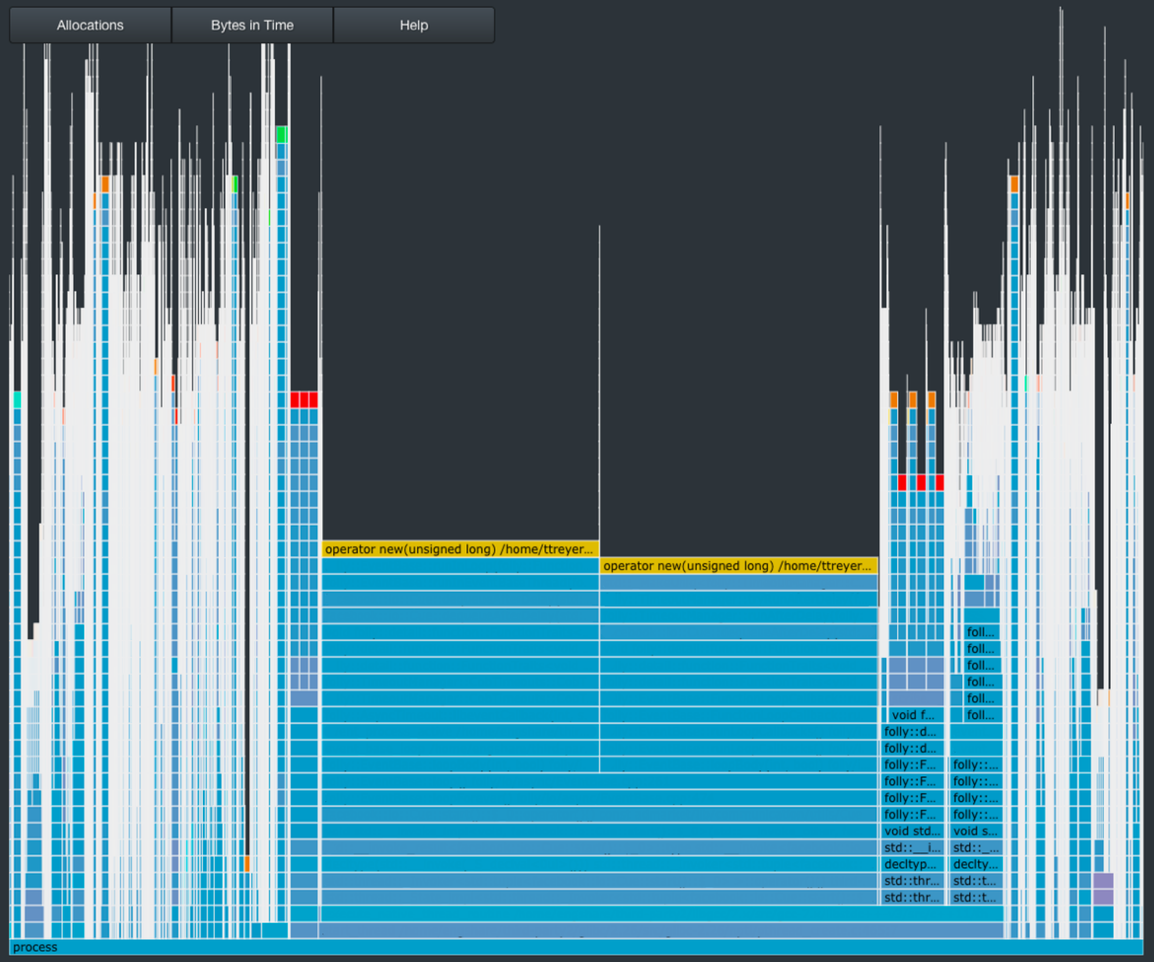

Allocations
Bytes in Time
Help
operator new(unsigned long) /home/ttreyer...
operator new(unsigned long) /home/ttreyer...
foll...
foll...
foll...
foll...
foll...
void f...
foll...
folly::d...
folly::d...
folly::F...
folly::...
folly::F...
folly::...
folly::F...
folly::...
folly::F...
folly::...
void std...
void s...
std::__i...
std::_...
decltyp...
declty...
std::thr...
std::t...
std::thr...
std::t...
process

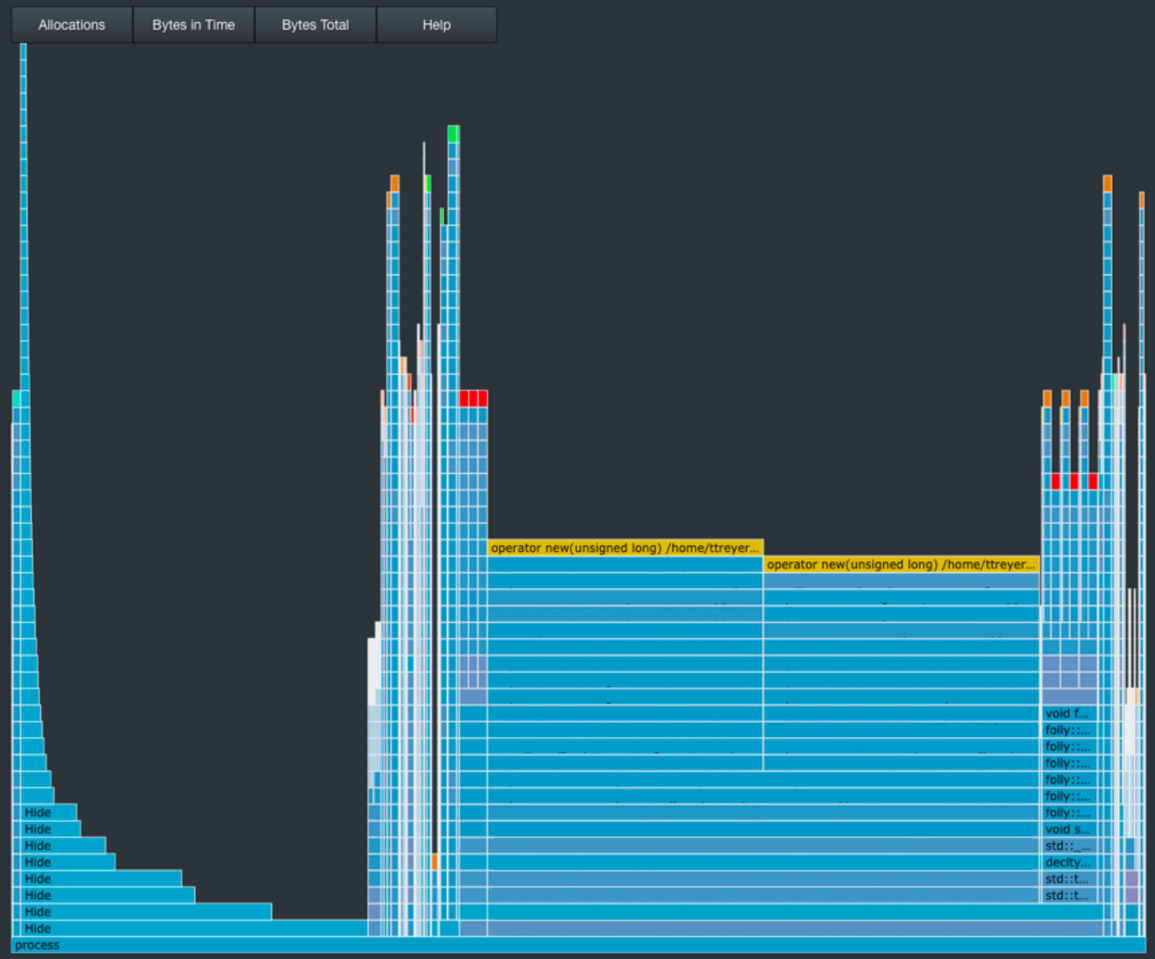
Truncate

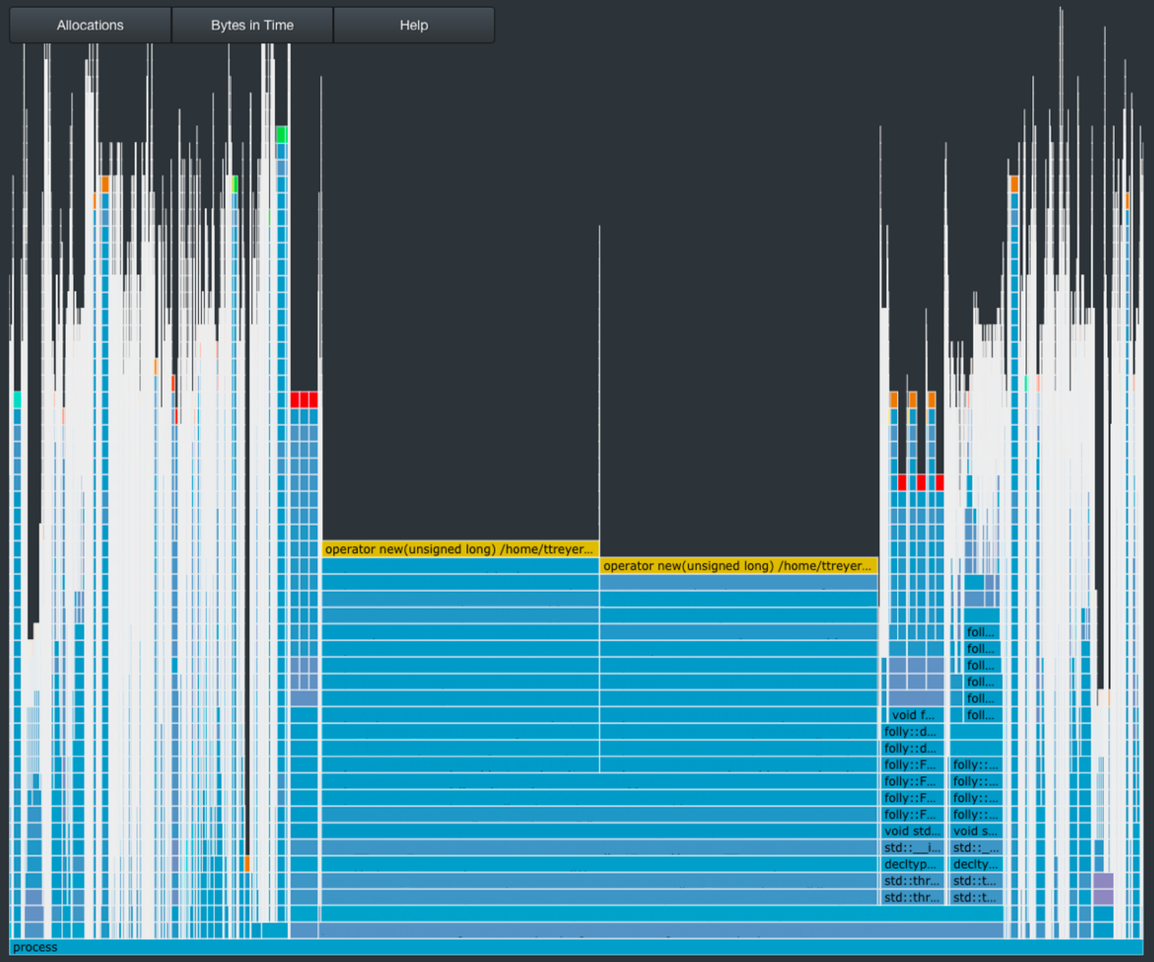


Truncate

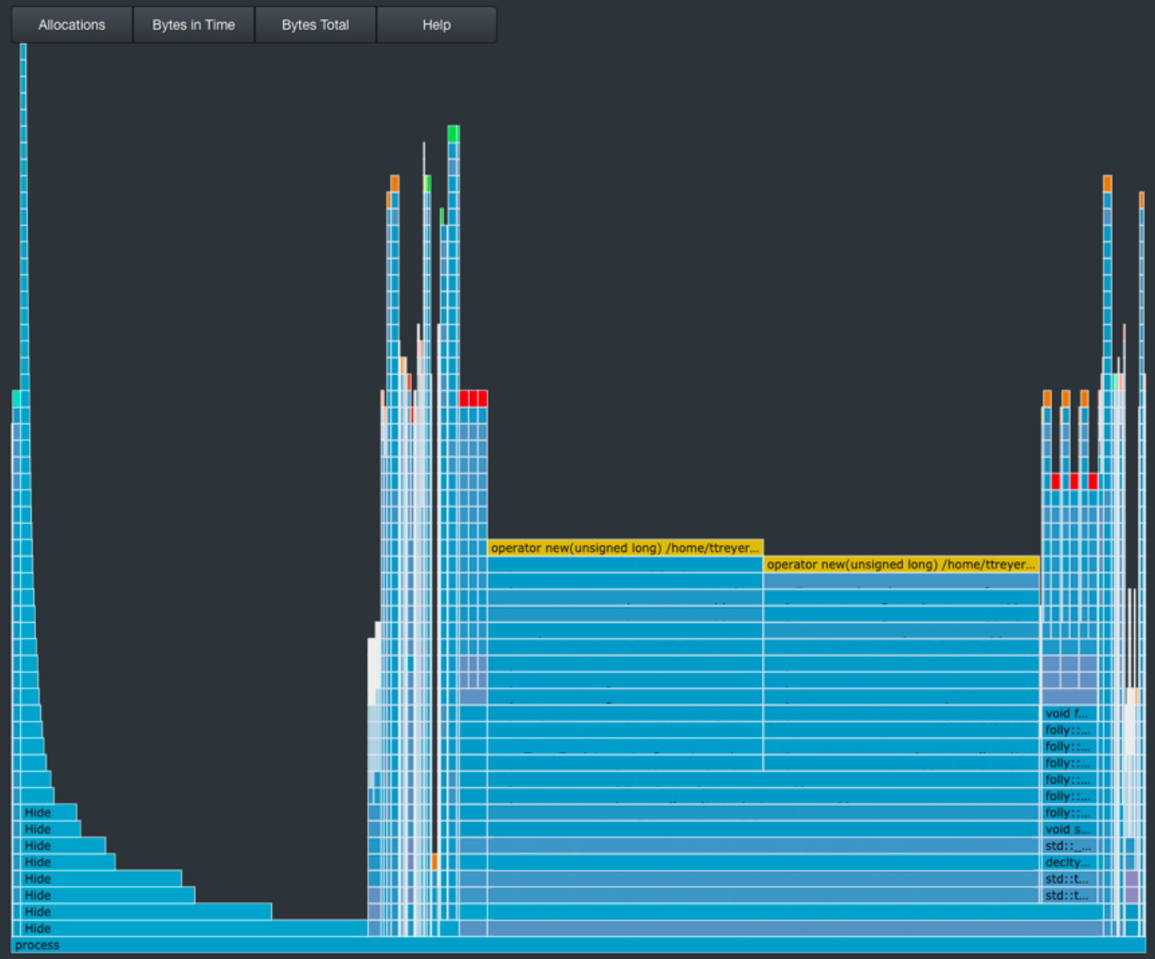



Allocations
Bytes in Time
Help
operator new(unsigned long) /home/ttreyer...
operator new(unsigned long) /home/ttreyer...
foll...
void f...
folly::d...
folly::F...
void std...
std::__i...
decltyp...
std::thr...
process


Allocations
Bytes in Time
Bytes Total
Help
operator new(unsigned long) /home/ttreyer...
operator new(unsigned long) /home/ttreyer...
Hide
void f...
folly::...
void s...
std::_...
decltyp...
std::t...
process

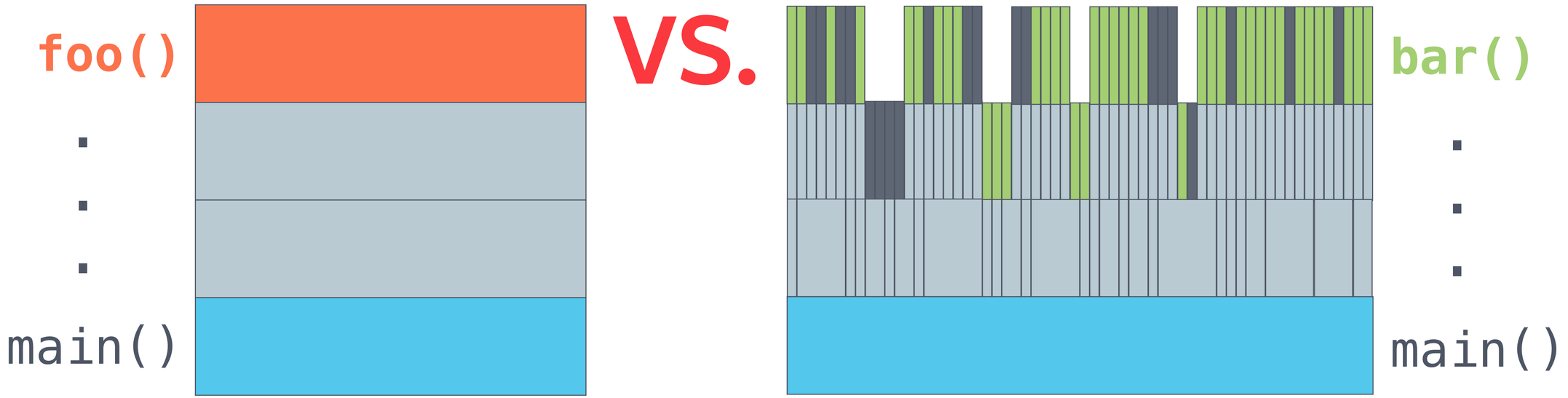


BEFORE

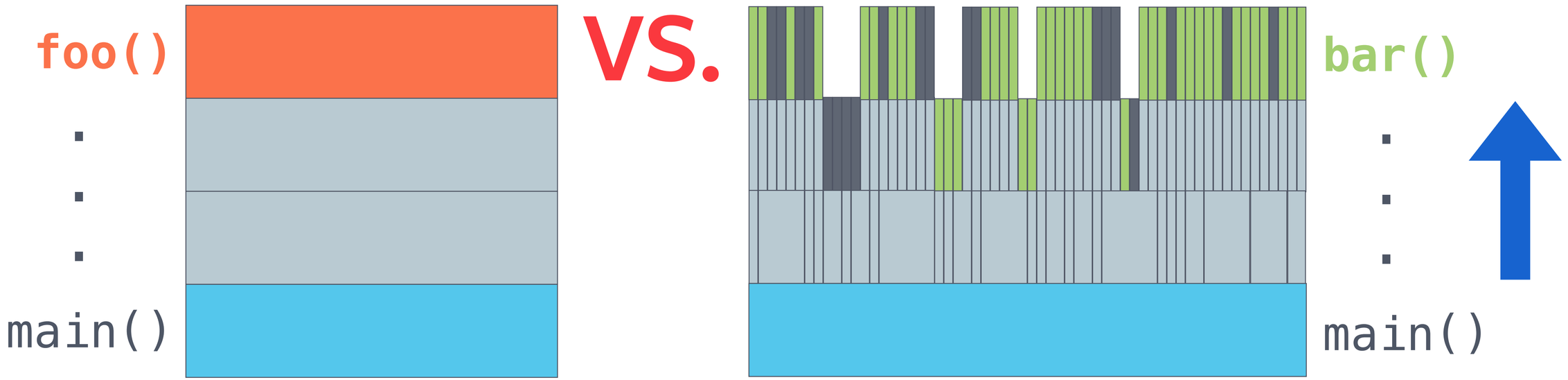


AFTER

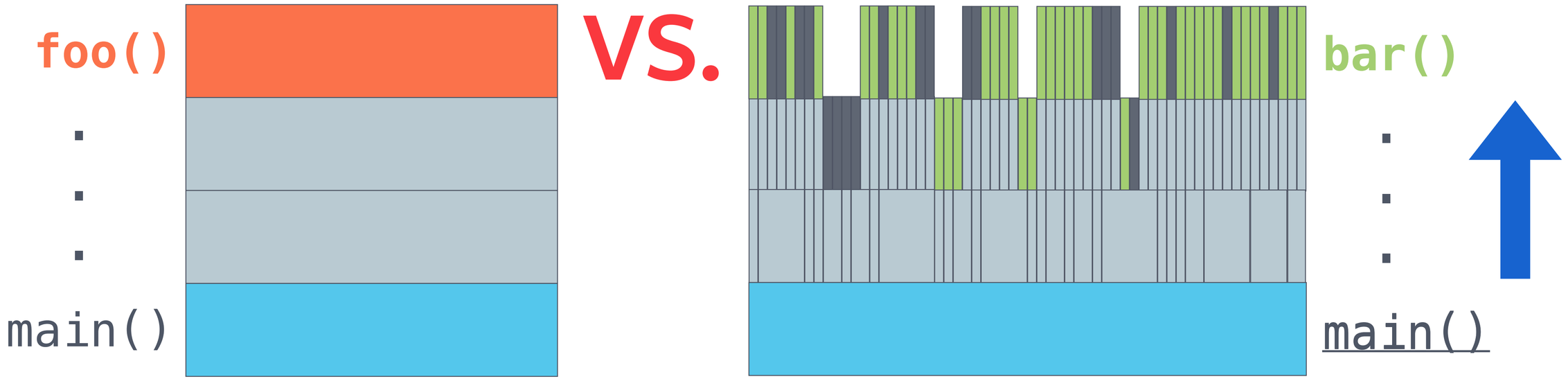
Death by a thousand cuts



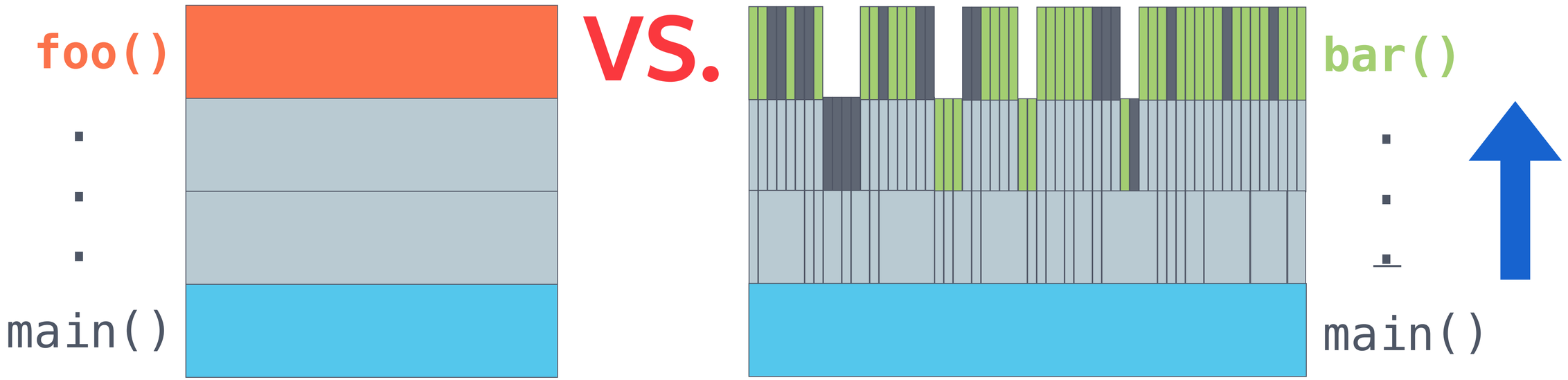
Death by a thousand cuts



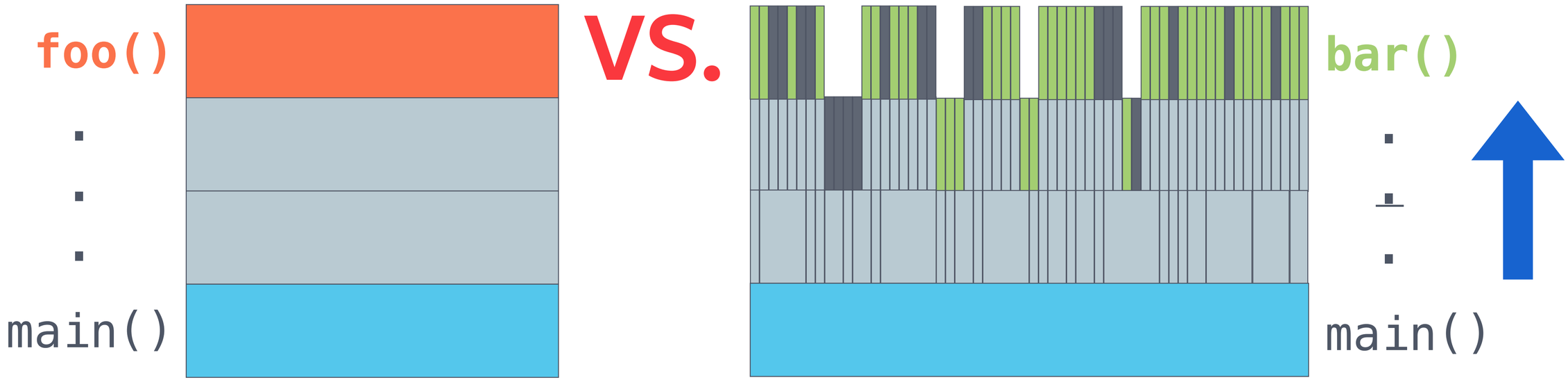
Death by a thousand cuts



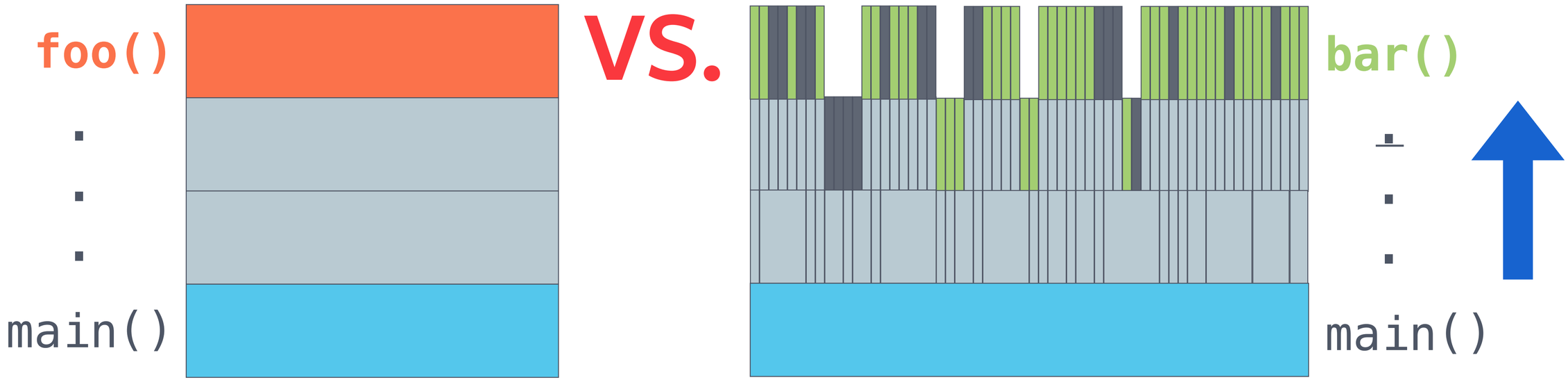
Death by a thousand cuts



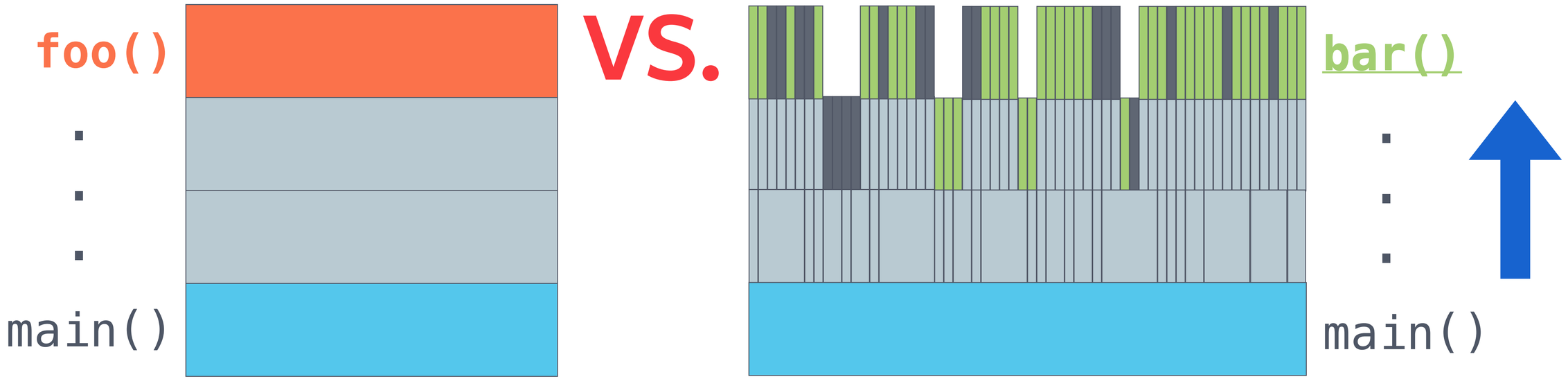
Death by a thousand cuts



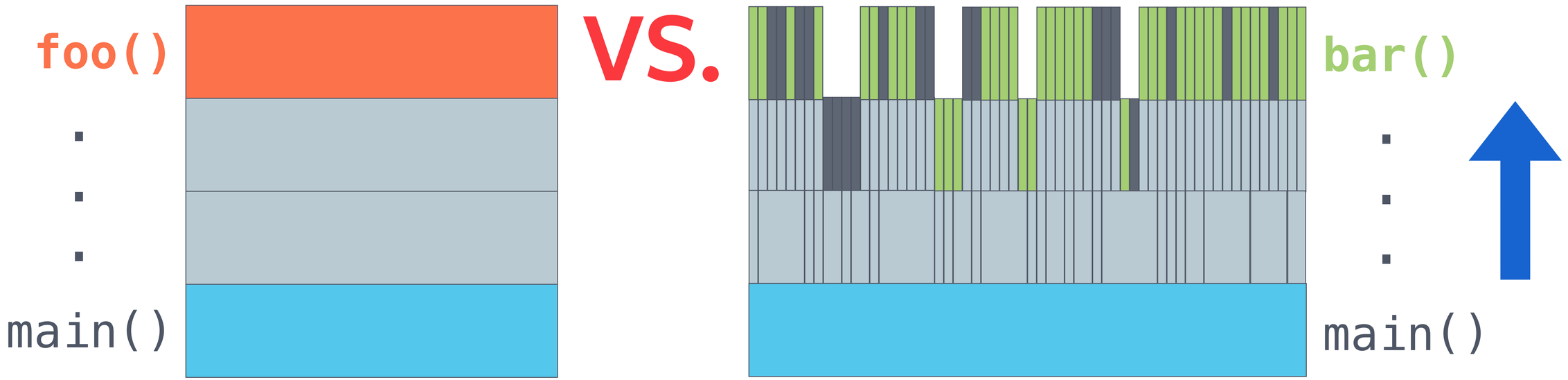
Death by a thousand cuts



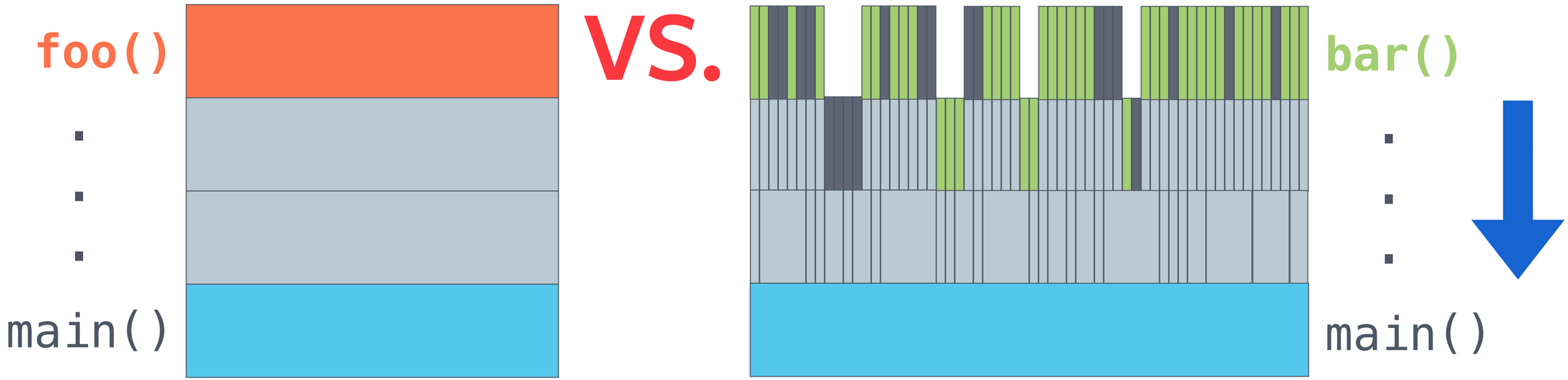
Death by a thousand cuts



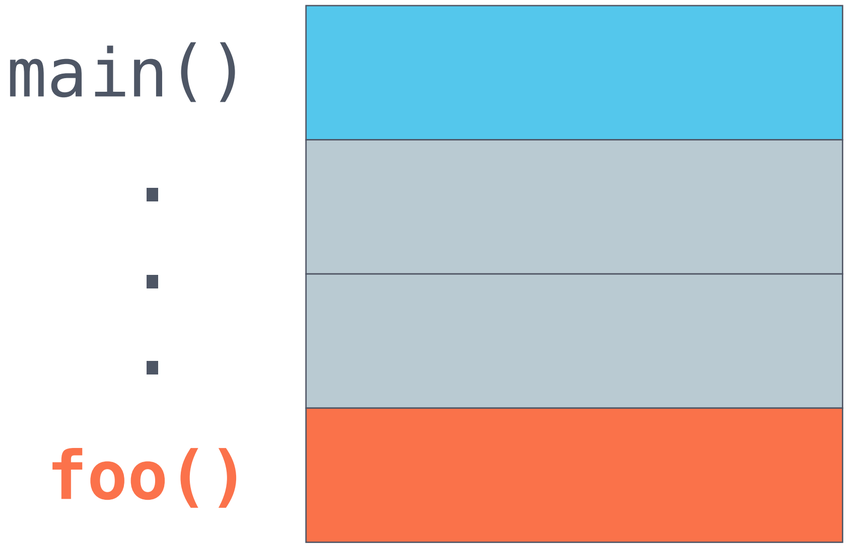
Death by a thousand cuts

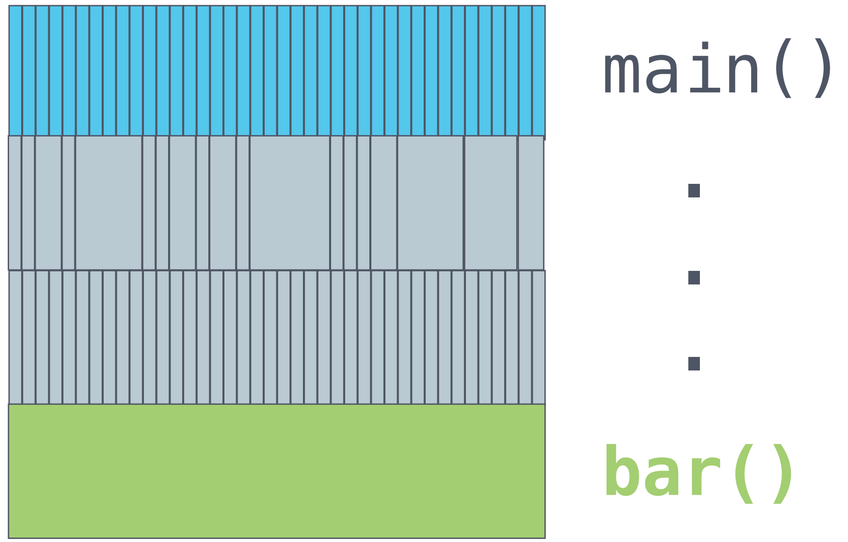


Death by a thousand cuts



Death by a thousand cuts





Memoro +

```
vector<BigT> getValues(
    map<Id, BigT>& largeMap,
    vector<Id>& keys) {

  vector<BigT> values;
  values.reserve(largeMap.size());

  for (const auto& key: keys)
    values.emplace_back(largeMap[key]);

  return values;
}
```

Demo

Memoro +

Run-Time Overhead

Visualizer

Open Challenges

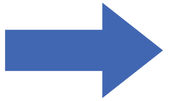
Dumping Profile

Your regular service

Dumping Profile

Your regular service

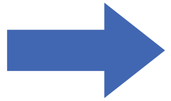
AtExit()

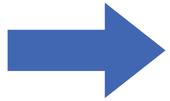


Dumping Profile

Your regular service

`AtExit()`





Dumping Profile

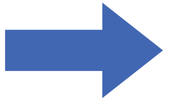
Facebook service

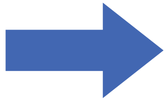
AtExit()

Dumping Profile

Facebook service

AtExit()





Dumping Profile

Facebook service

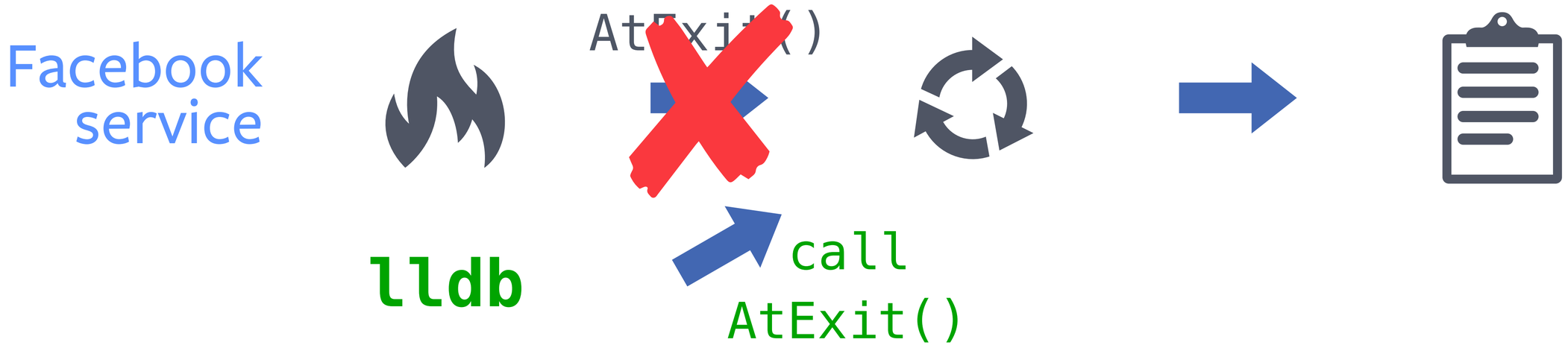
Dumping Profile

Facebook service

Dumping Profile

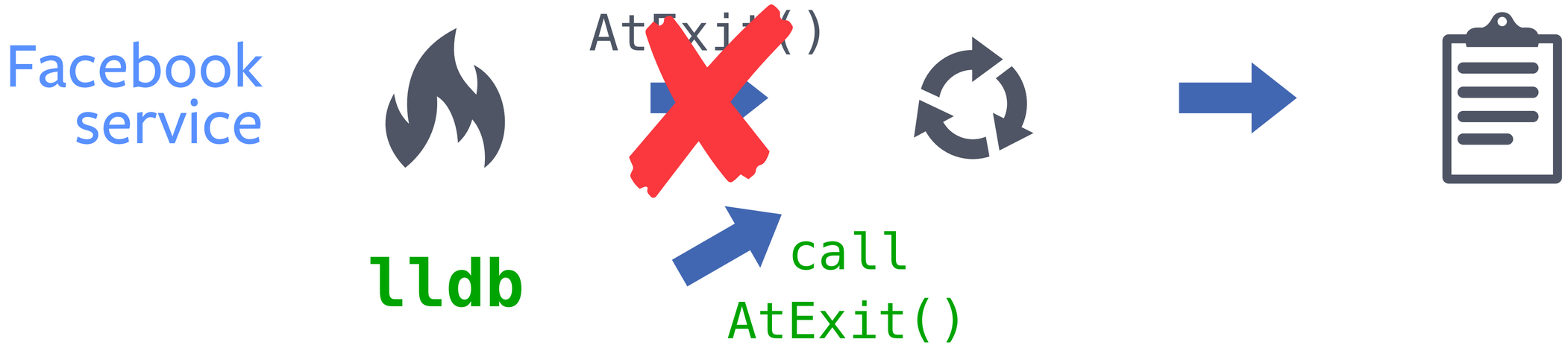
Facebook service

Dumping Profile



a. Signal to dump (`SIGPROF`)

Dumping Profile



a. Signal to dump (`SIGPROF`)

b. Ring buffer + Periodic write

Compile-Time Stack Analysis

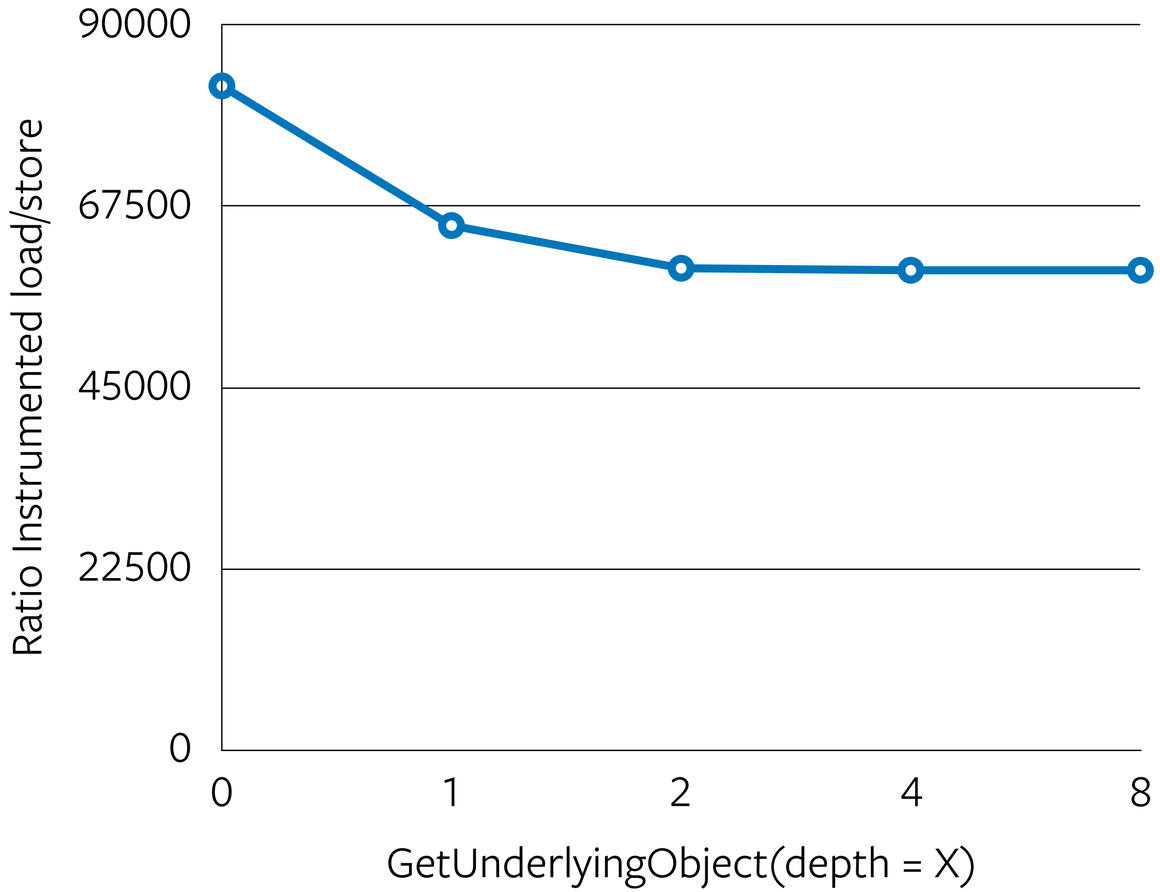
Compile-Time Stack Analysis

Compile-Time Stack Analysis

ld/st

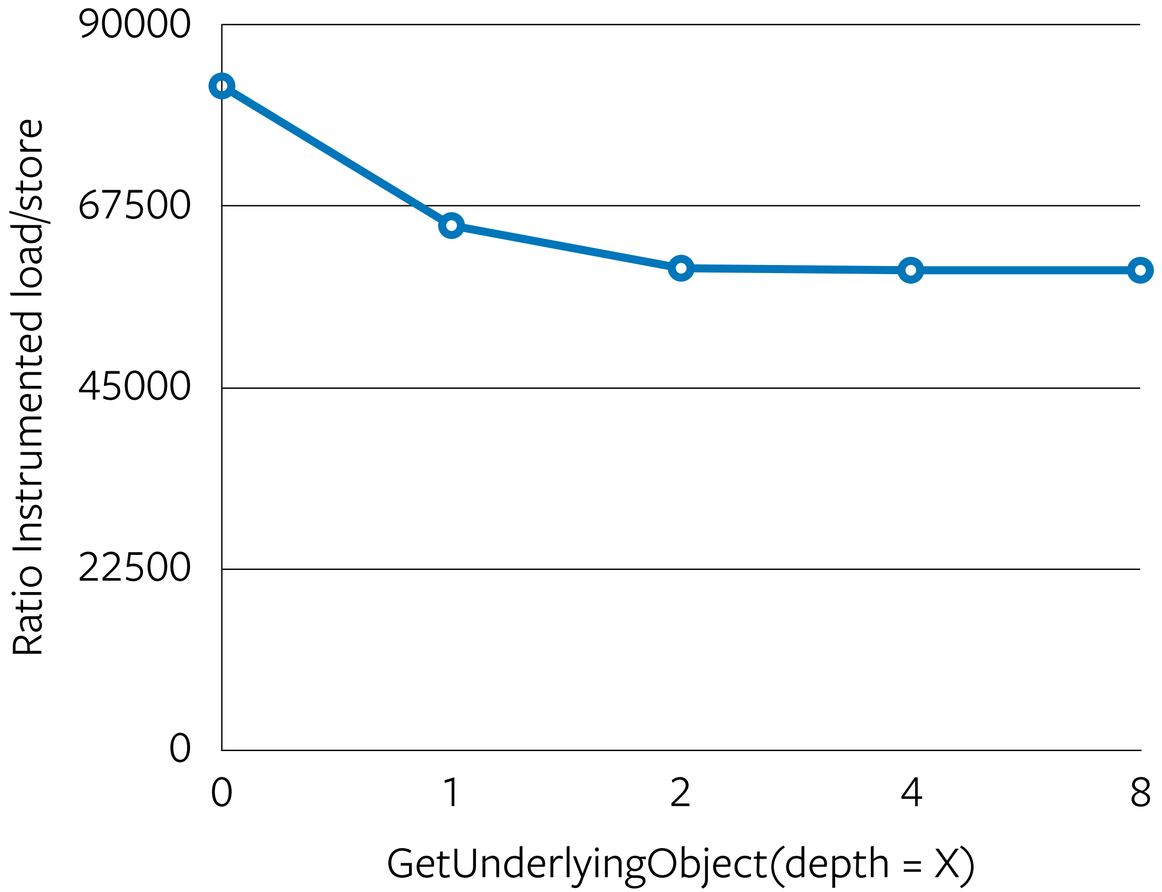
`llvm::GetUnderlyingObject()`

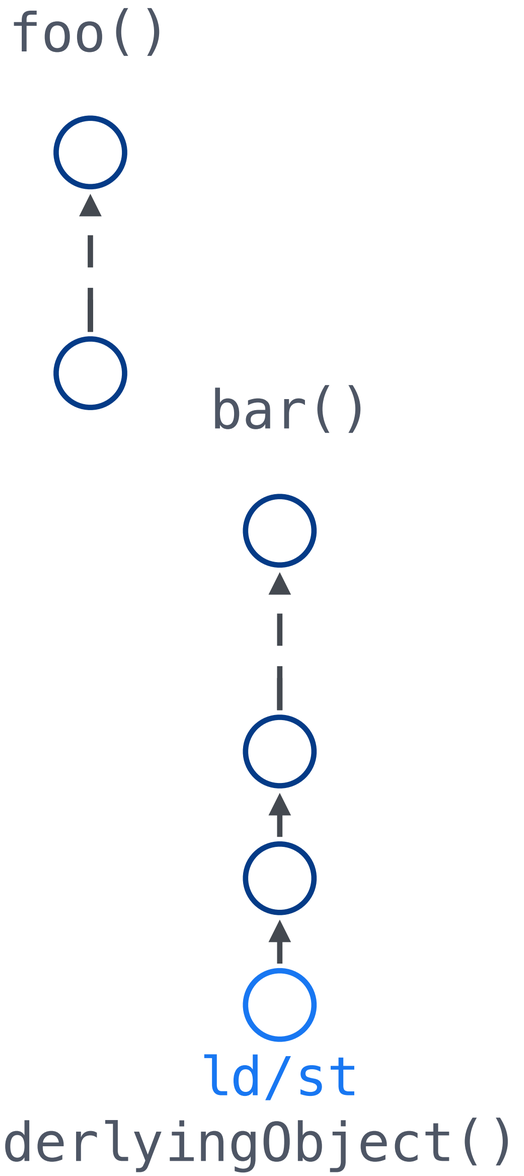
Compile-Time Stack Analysis



`llvm::GetUnderlyingObject()`

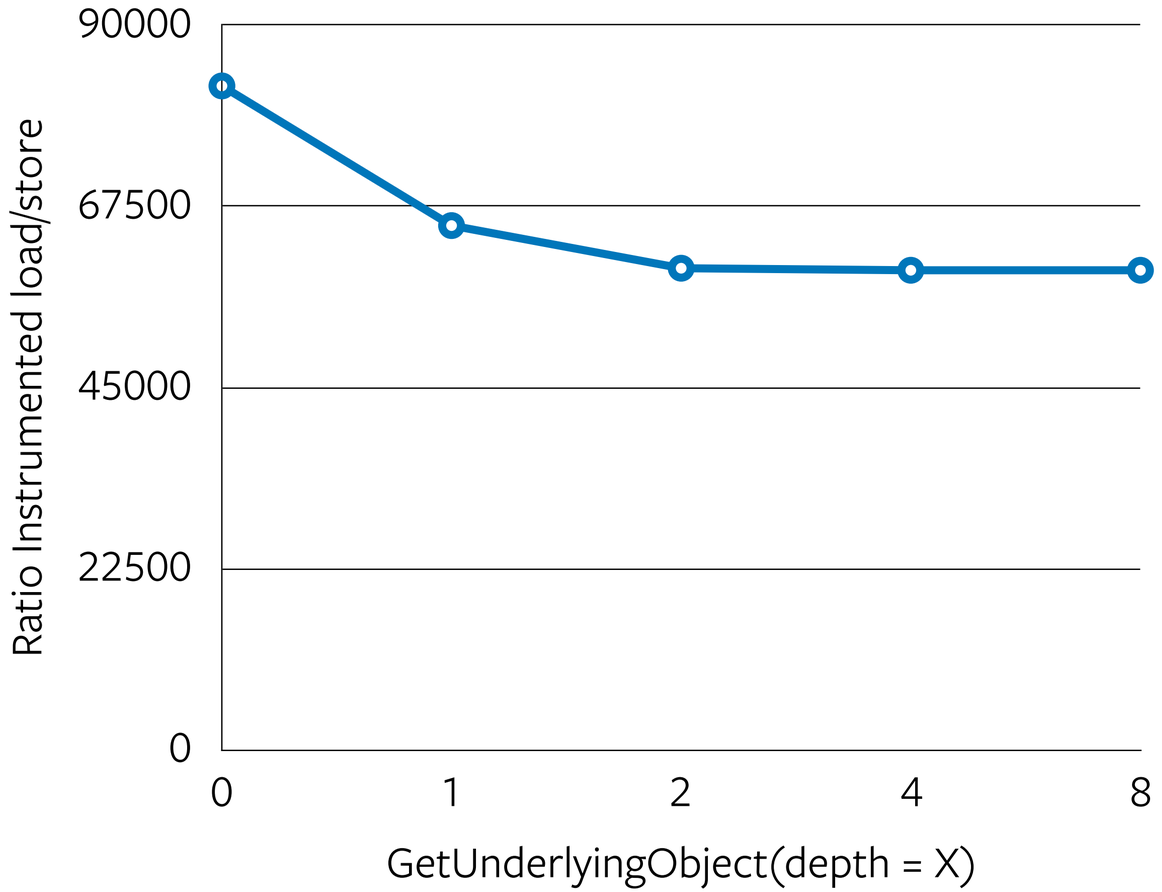
Compile-Time Stack Analysis

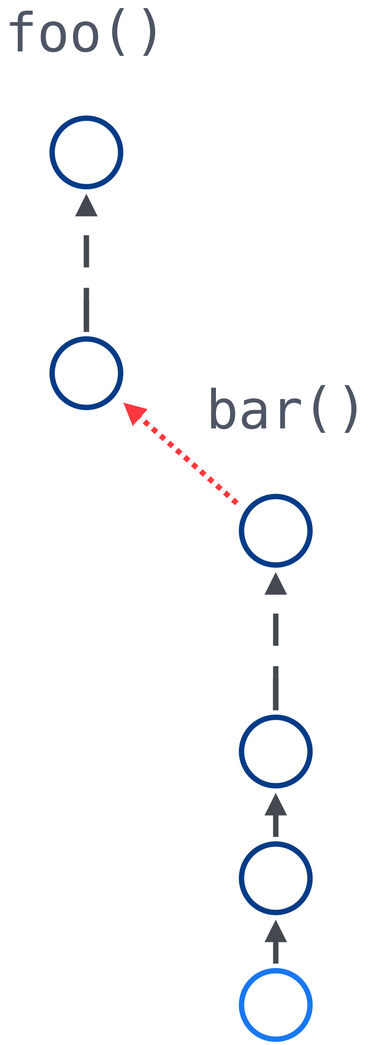




`llvm::GetUnderlyingObject()`

Compile-Time Stack Analysis

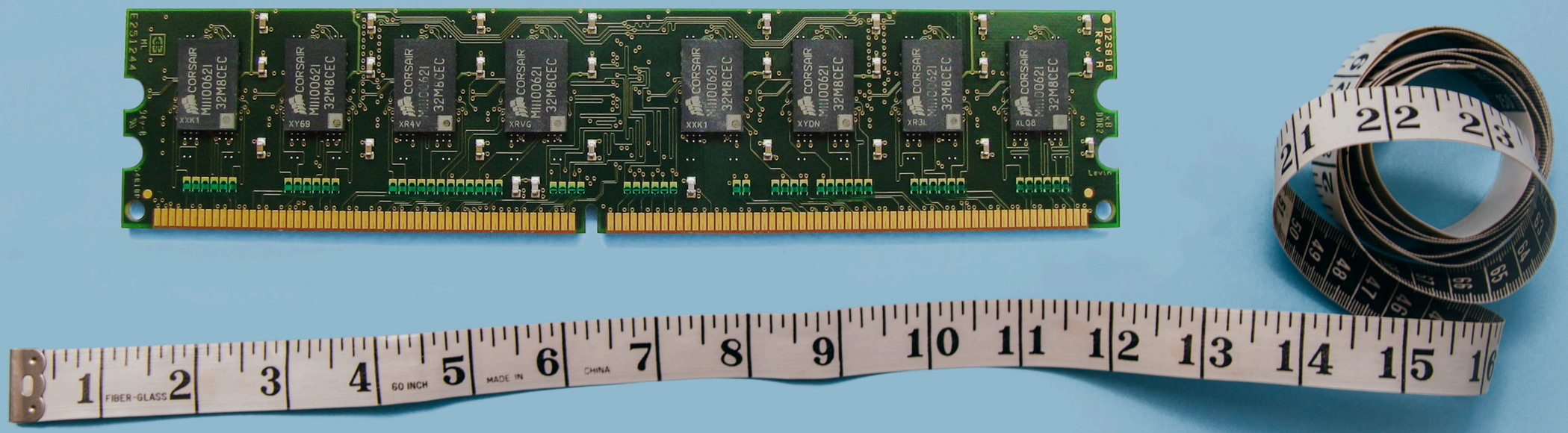




`llvm::GetUnderlyingObject()`

Thank you!

github.com/epfl-vlsc/memoro



Thierry Treyer
Performance & Capacity Intern

Mark Santaniello
Performance & Capacity Engineer

James Larus
EPFL IC School Dean
EPFL