Object Introspection:

A Revolutionary Memory Profiler for C++ Objects

JONATHAN HASLAM
& ADITYA SARWADE

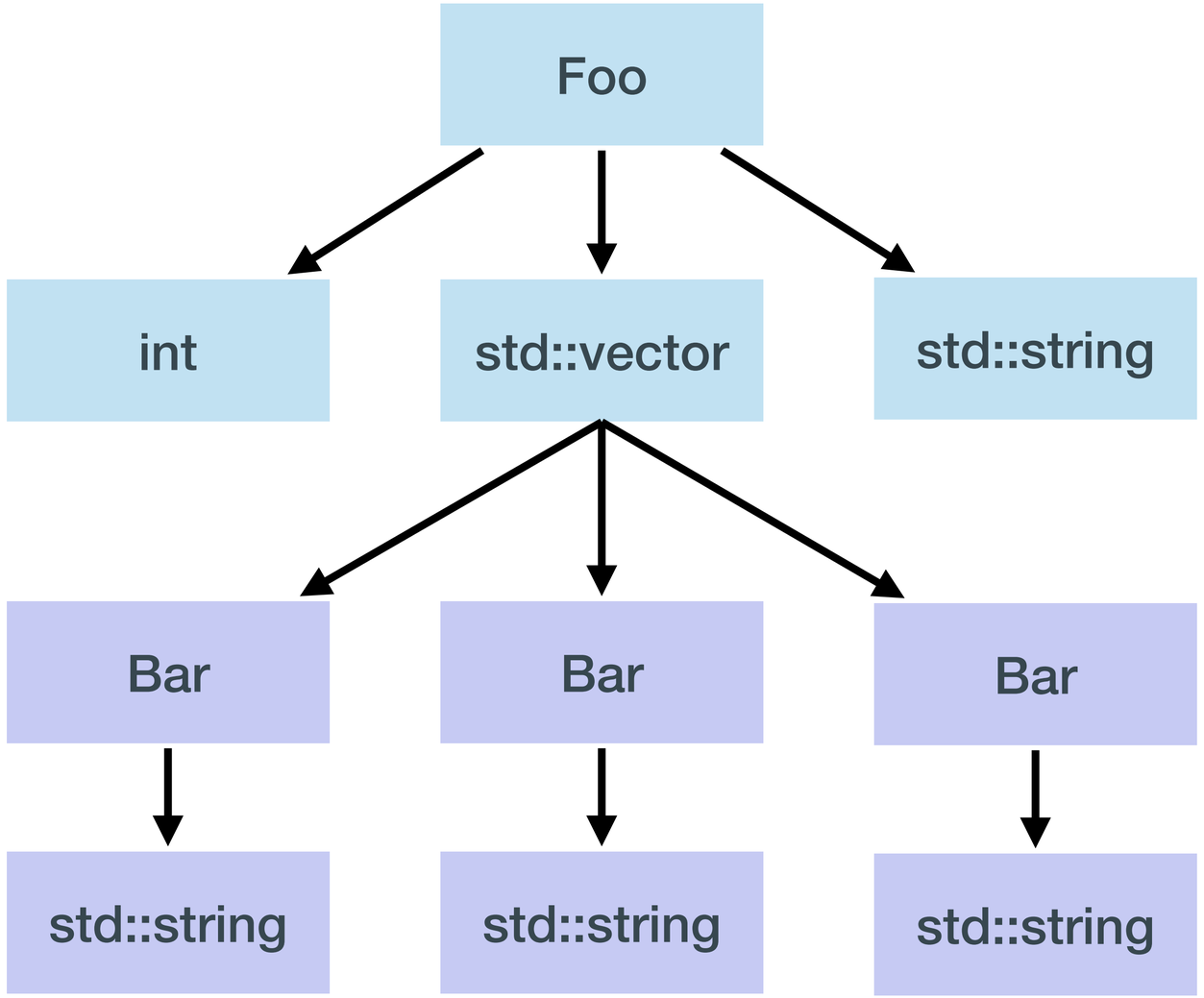
Scene Setting

> “… functions then provide the basic vocabulary of computation, just as types (built-in and user-defined) provide the basic vocabulary of data.”
>
> Bjarne Stroustrup, A Tour of C++, Second Edition.

```
struct TimeStamps {
   i64  insertionTime;
   i64  processingTime;
};
```

Scene Setting

```
struct Bar {
  std:string str;
};

struct Foo {
  int                n;
  std::vector<Bar> bar_vec;
  std::string        foo_str;
};
```

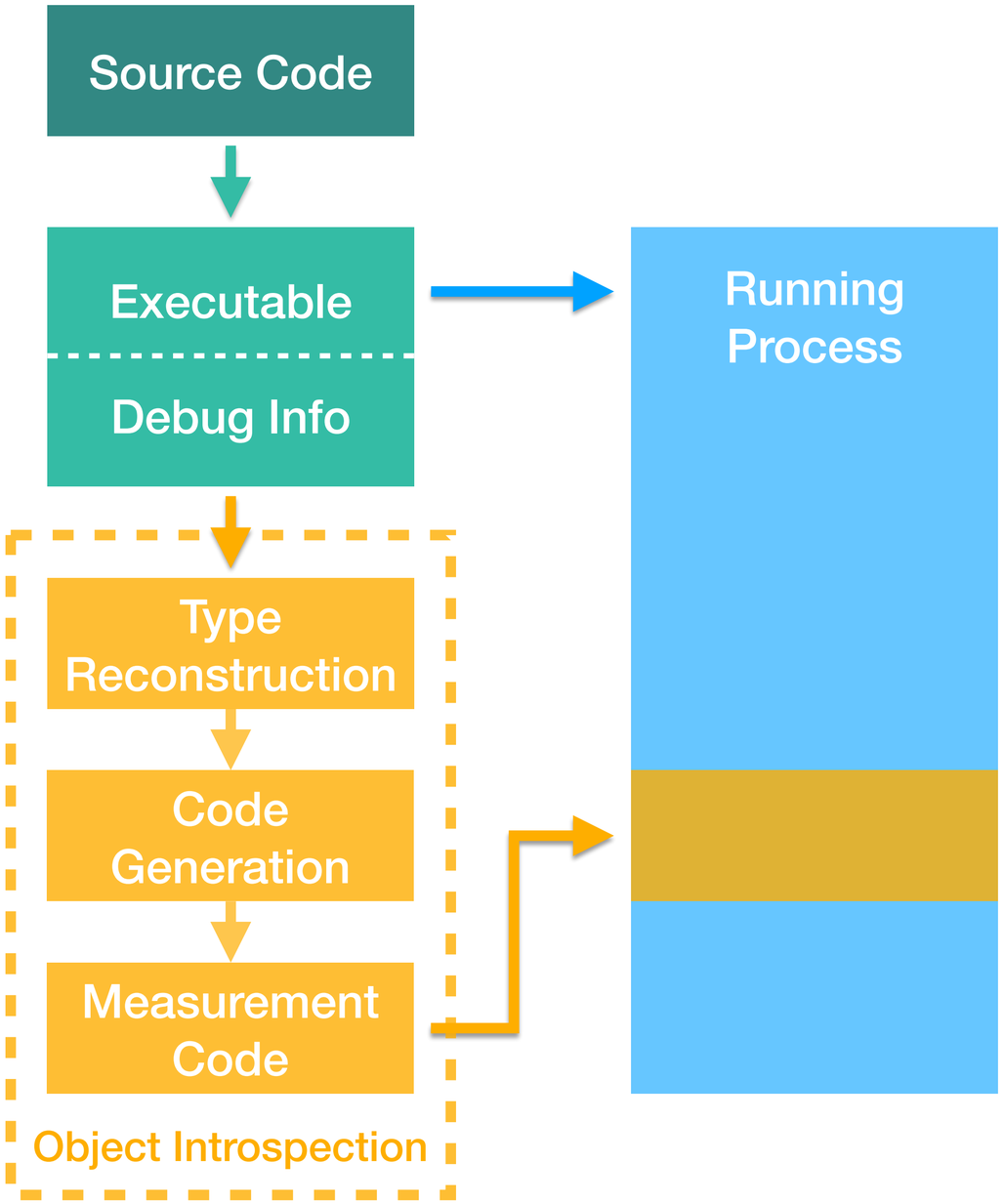


Object Introspection (OI) Goals

- Byte level memory footprint analysis for objects
  - Complete object type hierarchies
  - Dynamic allocations and containers
- Optimised, production applications in production environments
- Dynamic Profiler
  - Can be used with no recompilation or code modification
- APIs

Outline

- Debug Info Analysis
  - Type/Layout Reconstruction
  - Code Generation
- Object Introspection as a Library
- Object Introspection as a Profiler
- Object Introspection Applied



Type Reconstruction

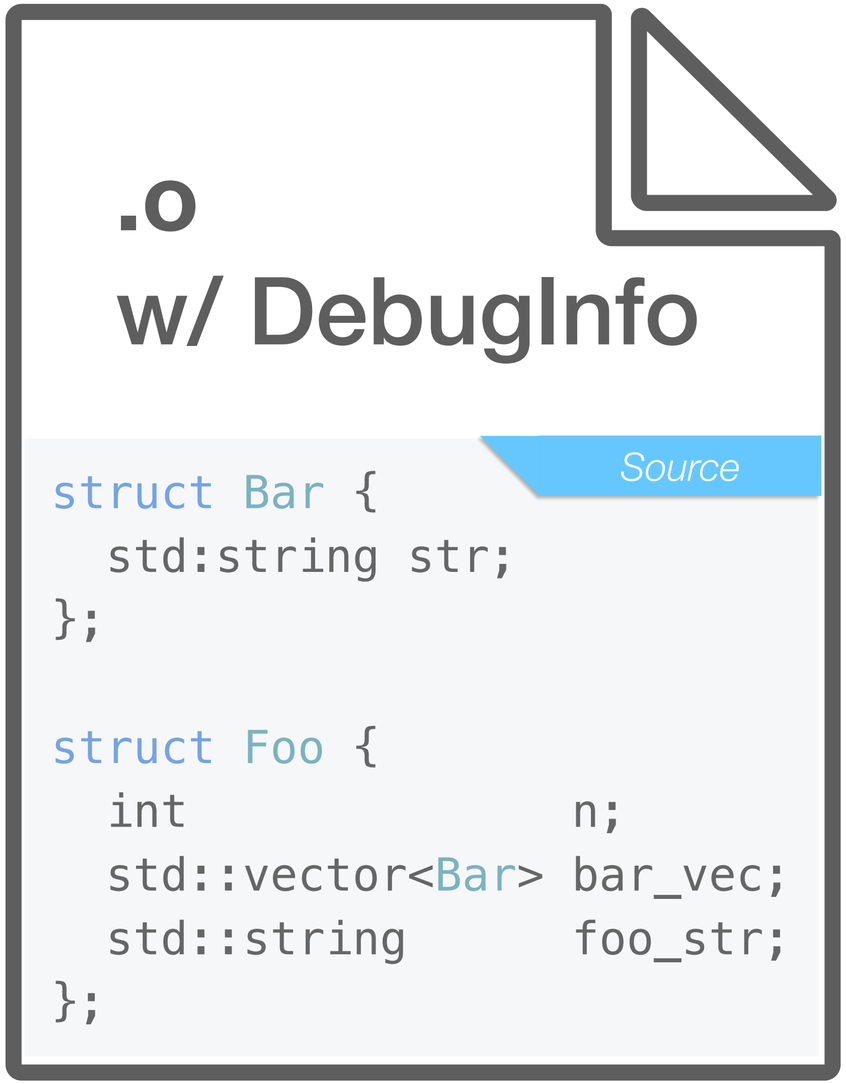
- (Re)construct an entire object hierarchy from a given root type
  - Understand the layout in memory of the entire hierarchy
  - Understand how to interpret data at memory offsets
  - Understand containers
- Compiler generated debug information required
  - DWARF v4 (DebugFission supported)
  - drgn (https://github.com/osandov/drgn) and lldb

Type Graph

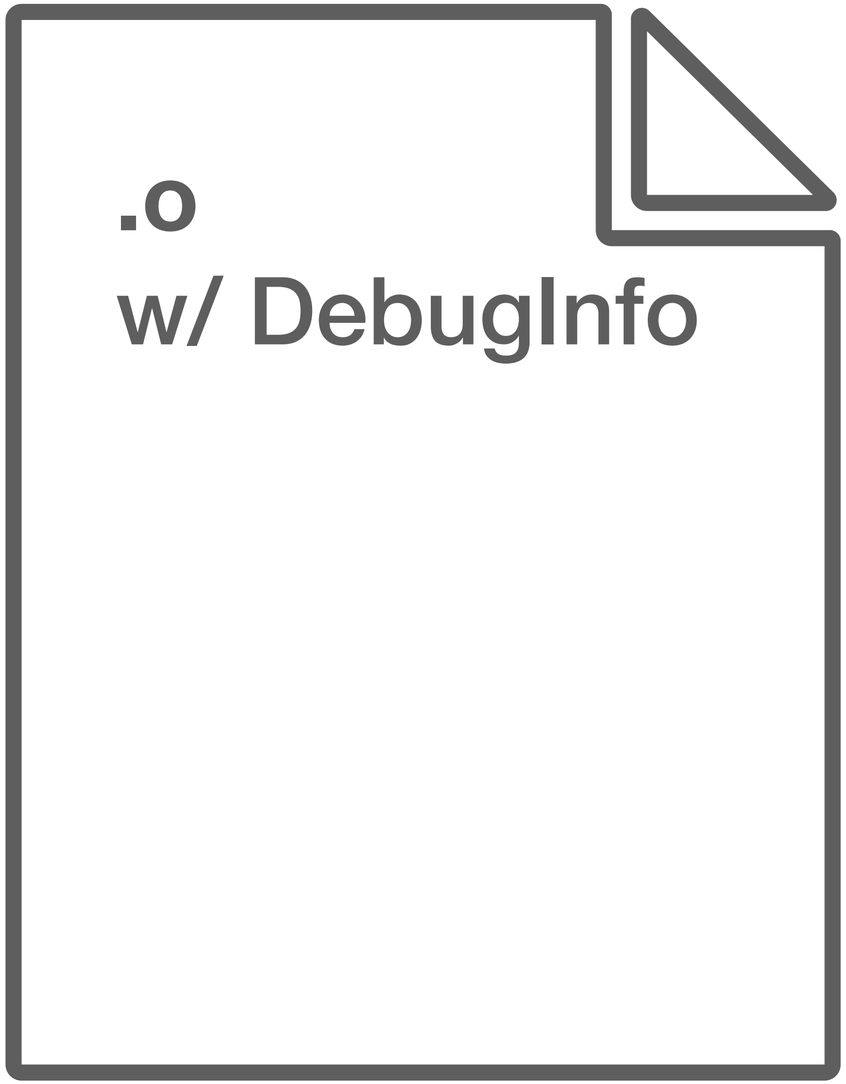
Source

```
struct Bar {
  std:string str;
};

struct Foo {
  int                n;
  std::vector<Bar> bar_vec;
  std::string        foo_str;
};
```

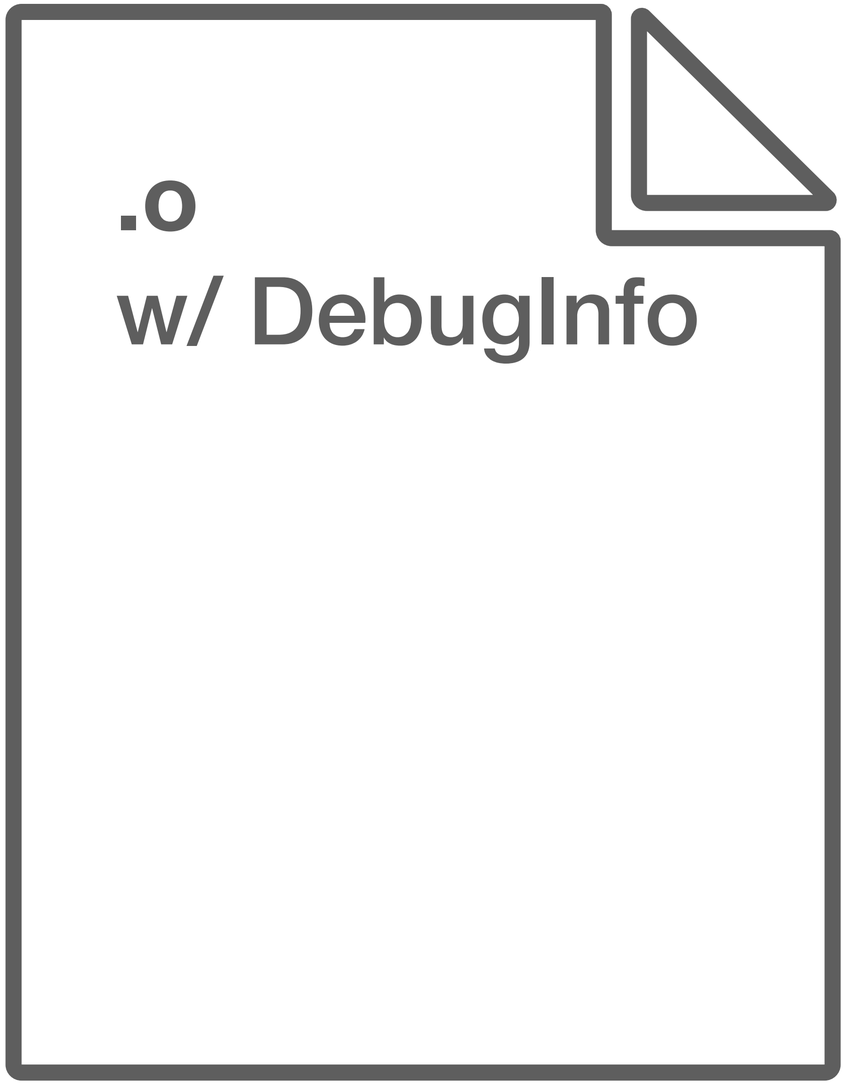
Type Graph

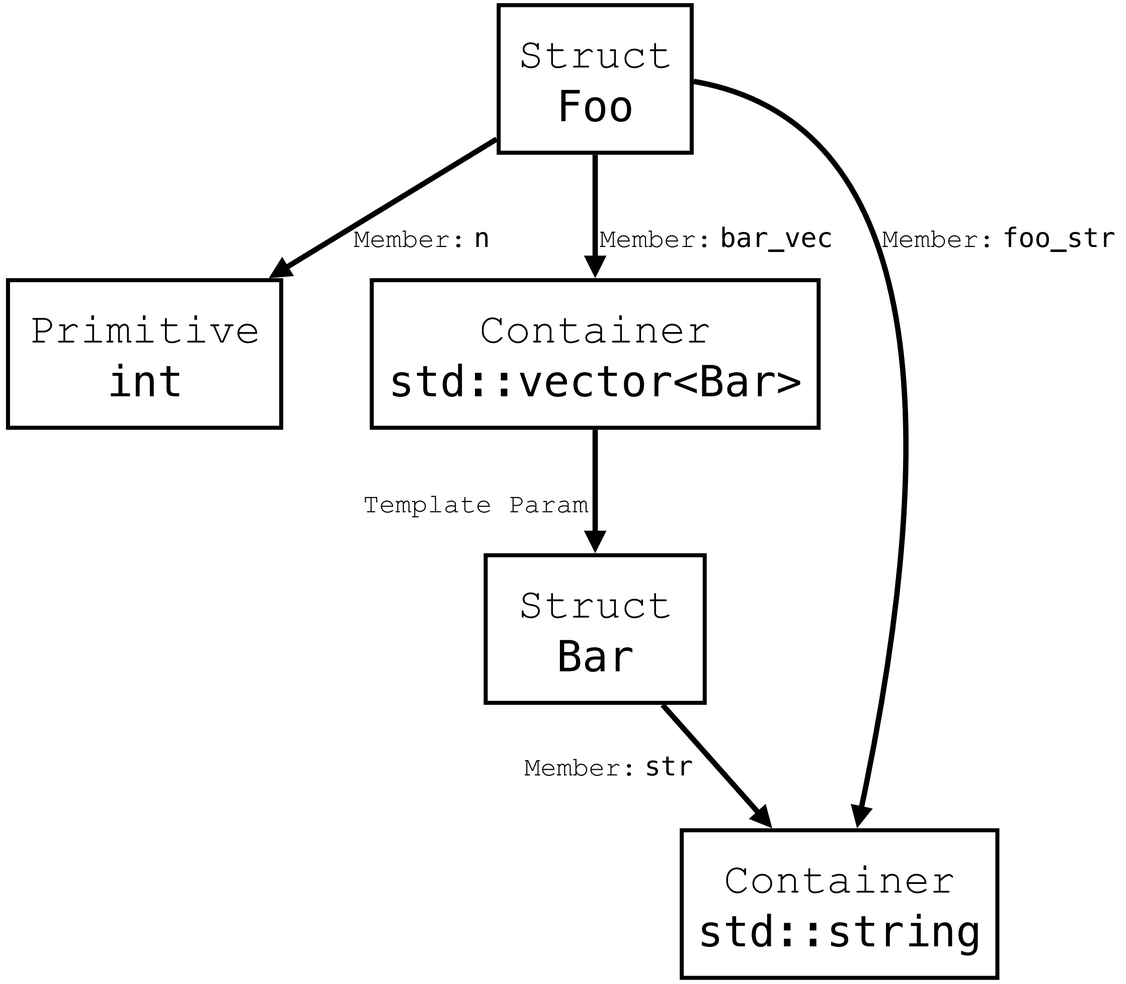

.o
w/ DebugInfo
Source
struct Bar {
std:string str;
};
struct Foo {
int n;
std::vector<Bar> bar_vec;
std::string foo_str;
};

Type Graph

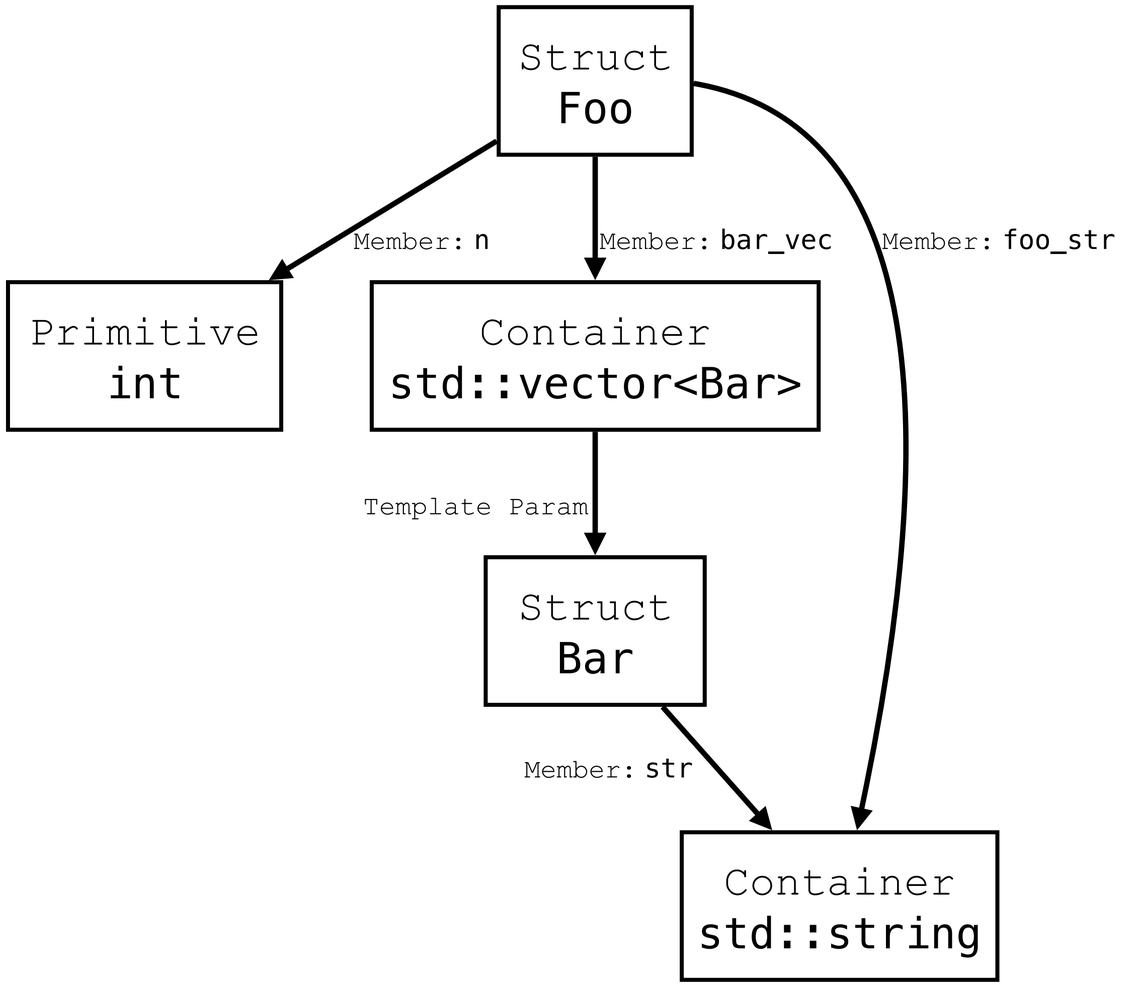


Type Graph


.o
w/ DebugInfo


Struct
Foo
Member: n
Member: bar_vec
Member: foo_str
Primitive
int
Container
std::vector<Bar>
Template Param
Struct
Bar
Member: str
Container
std::string

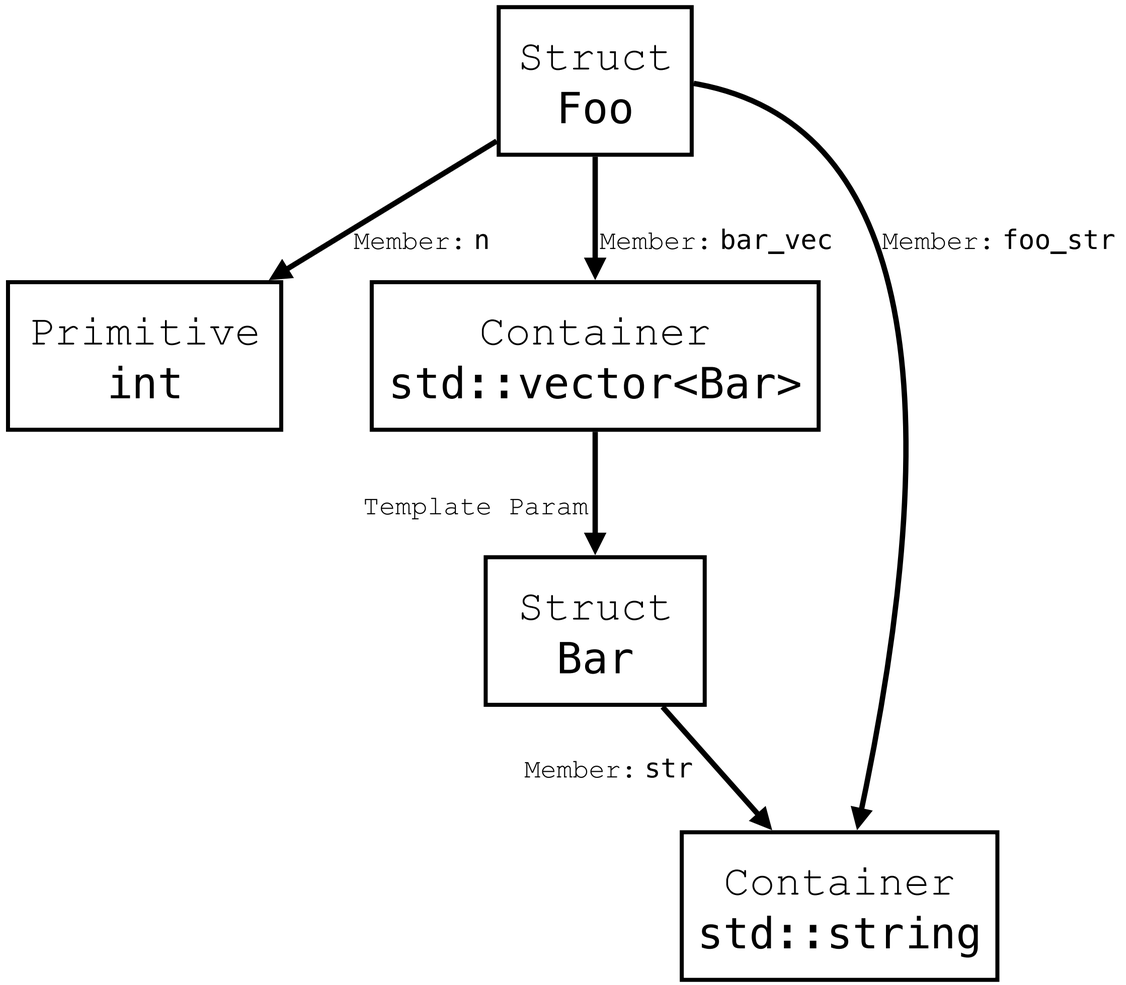
Type Graph


Struct
Foo
Member: n
Member: bar_vec
Member: foo_str
Primitive
int
Container
std::vector<Bar>
Template Param
Struct
Bar
Member: str
Container
std::string

Type Graph

Reconstructed

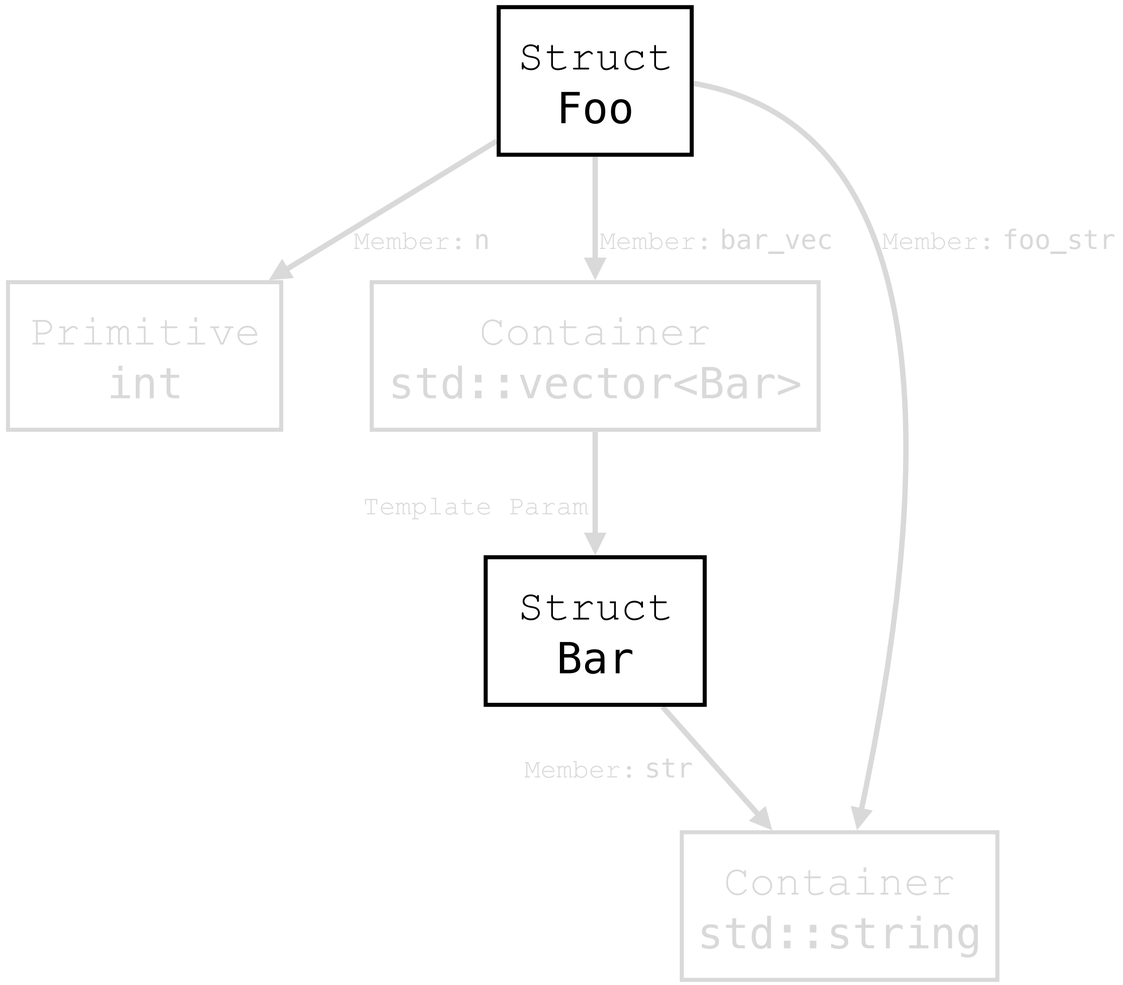
```
struct Bar {
  std:string str;
};

struct Foo {
  int                n;
  std::vector<Bar> bar_vec;
  std::string        foo_str;
};
```



Type Graph

Reconstructed

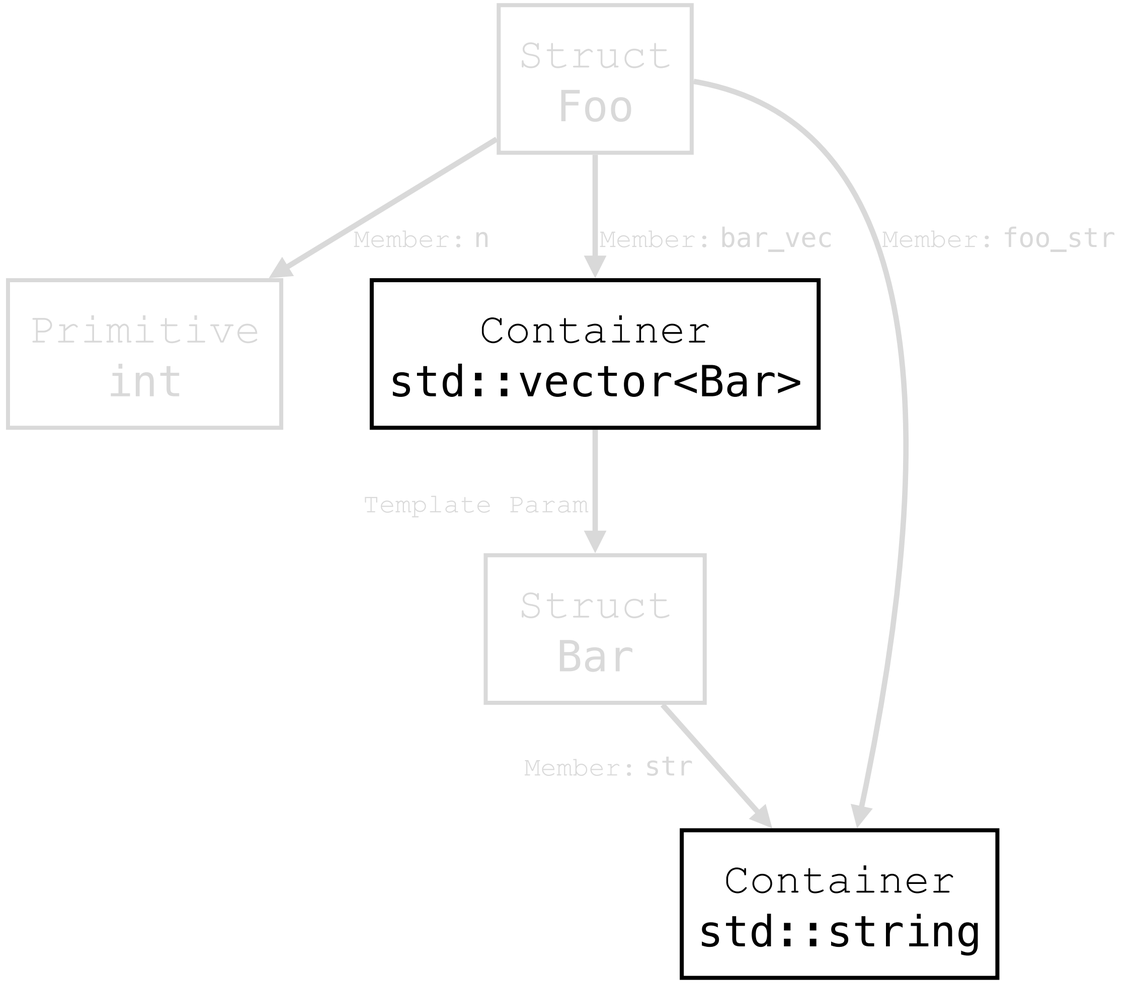
```
struct Bar {
  std:string str;
};

struct Foo {
  int                n;
  std::vector<Bar> bar_vec;
  std::string        foo_str;
};
```



Type Graph

Reconstructed

```
struct Bar {
  std:string str;
};

struct Foo {
  int                n;
  std::vector<Bar> bar_vec;
  std::string        foo_str;
};
```



Type Graph Transformation: Inline Inheritance

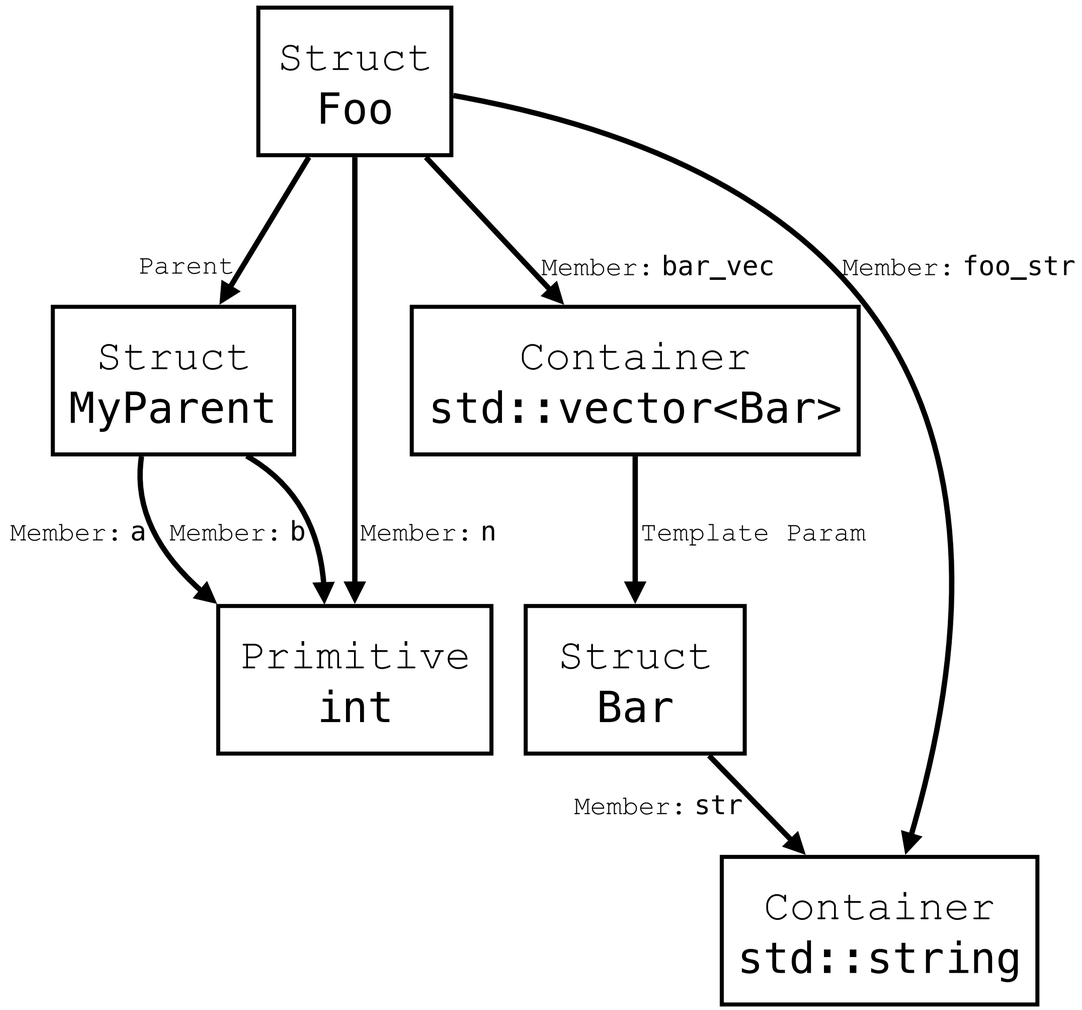
Source

```
struct Bar {
  std:string str;
};

struct MyParent {
  int a;
  int b;
};

struct Foo : public MyParent {
  int                n;
  std::vector<Bar> bar_vec;
  std::string        foo_str;
};
```

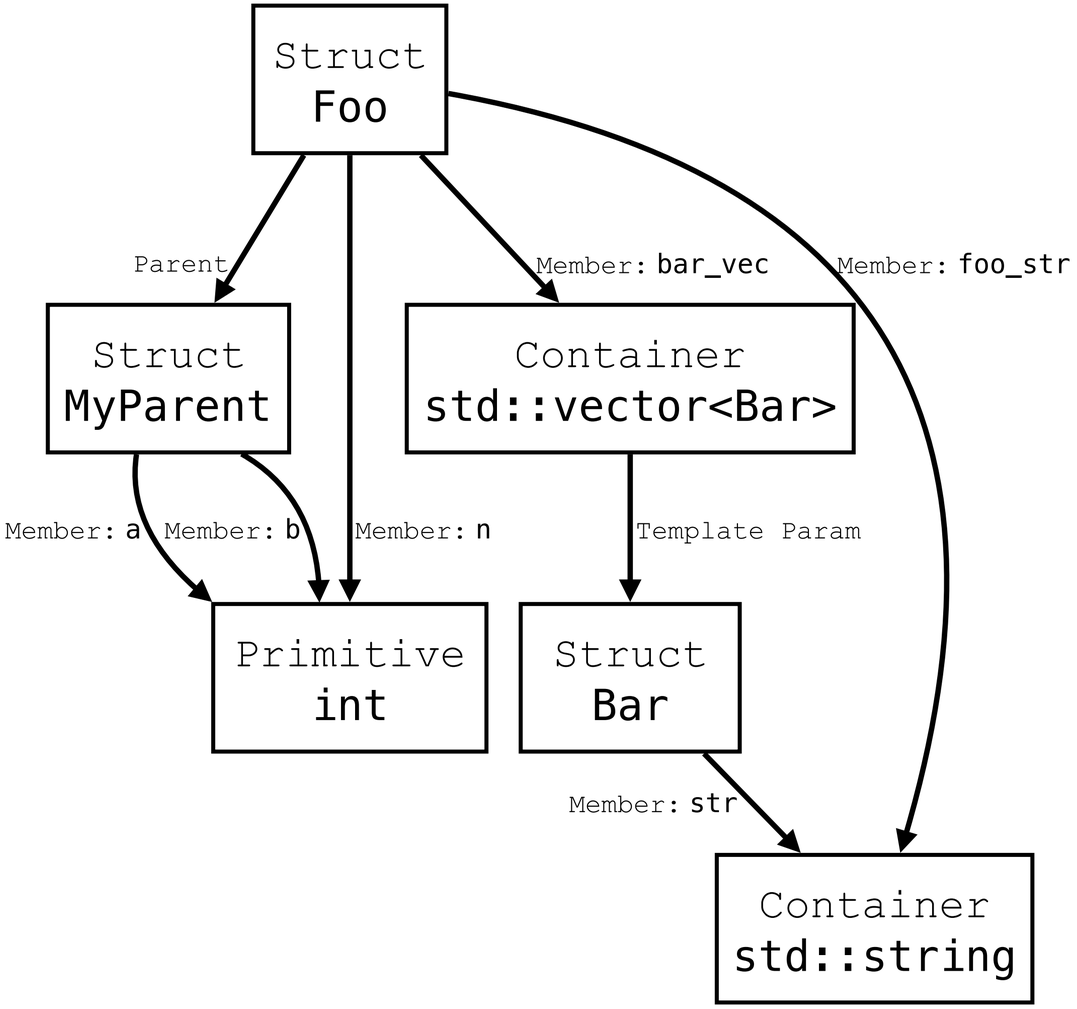
Type Graph Transformation: Inline Inheritance

Source

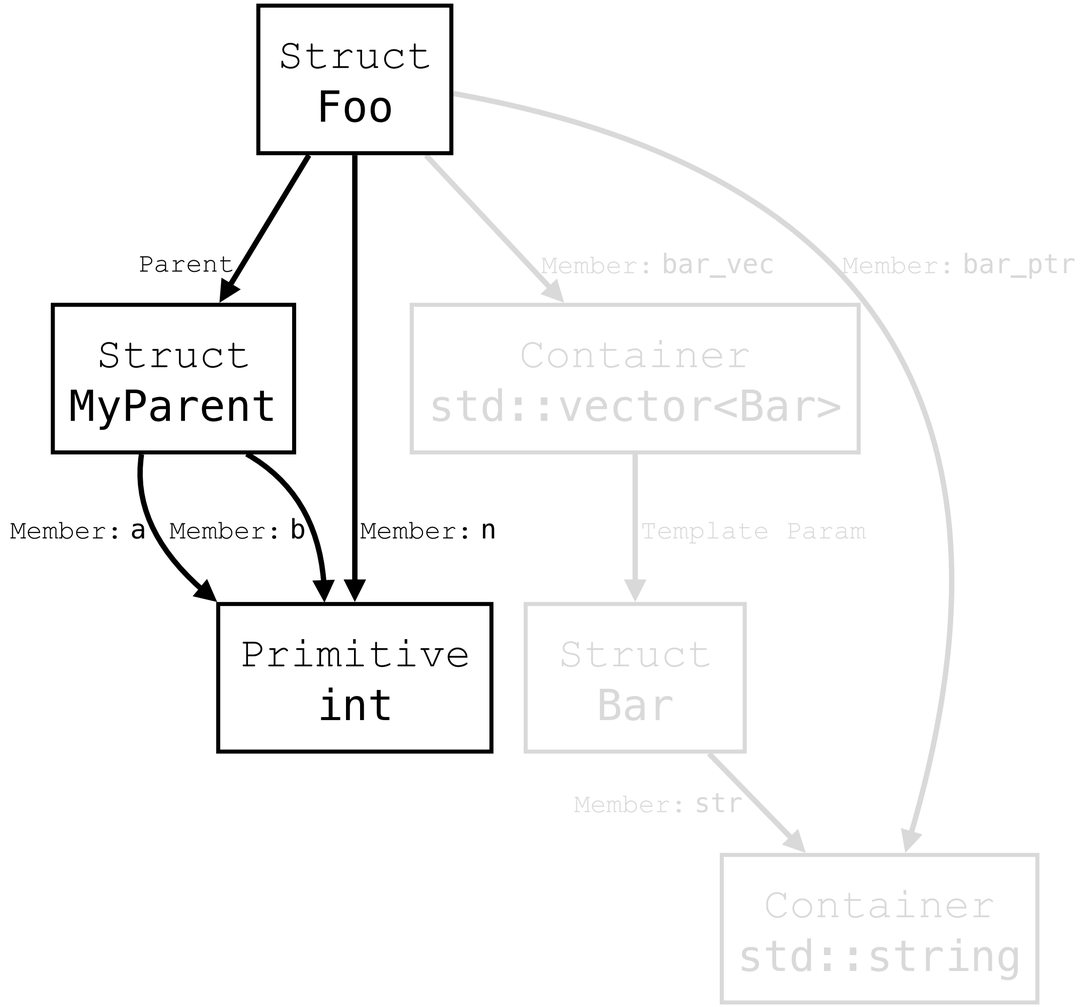
```
struct Bar {
  std:string str;
};

struct MyParent {
  int a;
  int b;
};

struct Foo : public MyParent {
  int                n;
  std::vector<Bar> bar_vec;
  std::string      foo_str;
};
```



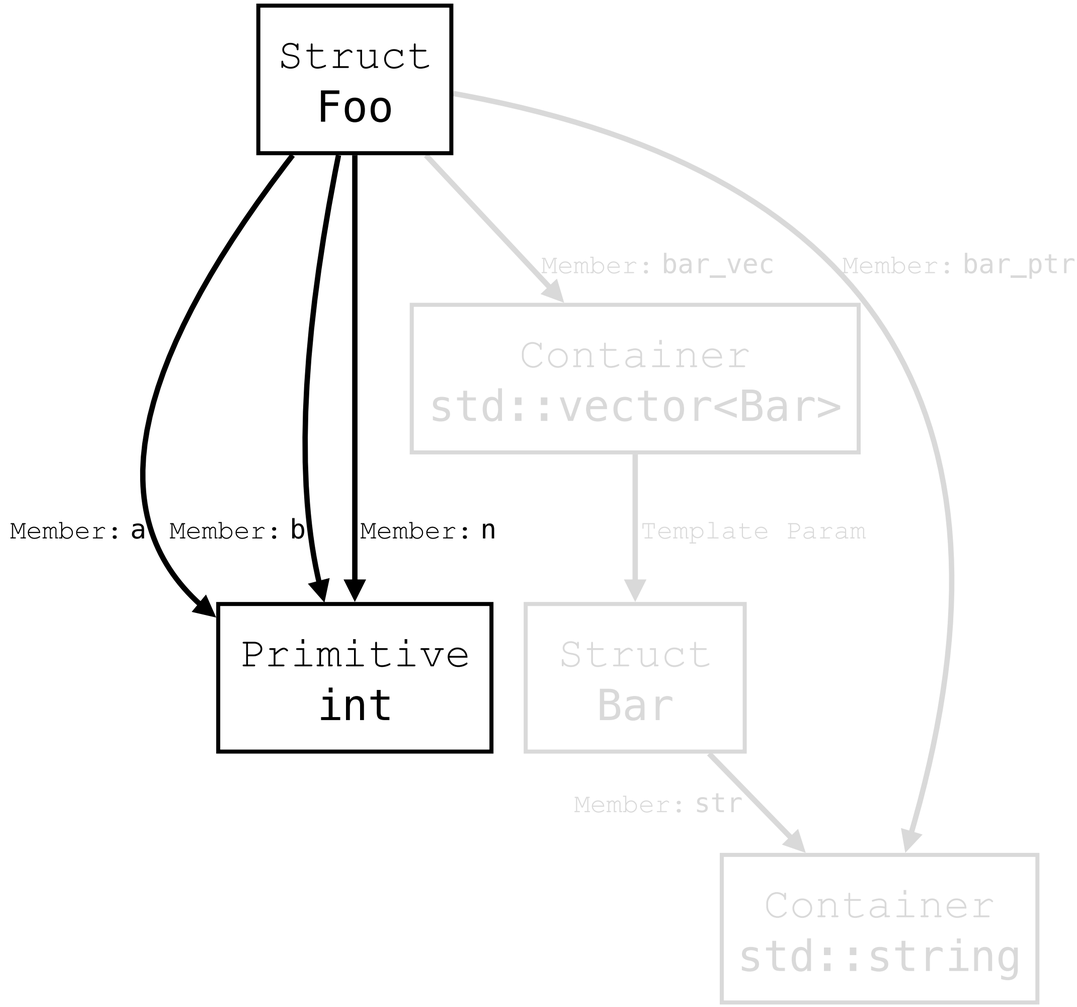
Type Graph Transformation: Inline Inheritance



Type Graph Transformation: Inline Inheritance



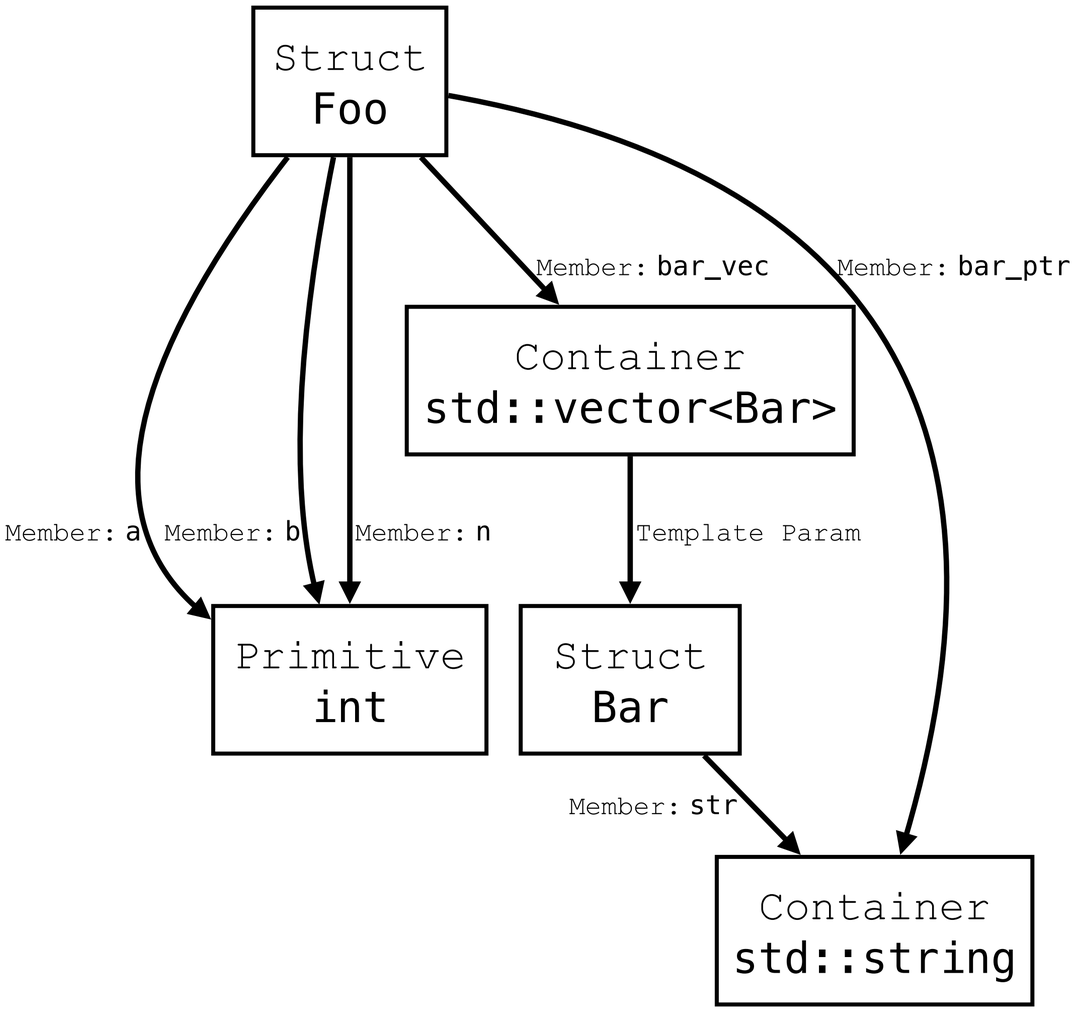
Type Graph Transformation: Inline Inheritance



Type Graph Transformation: Inline Inheritance

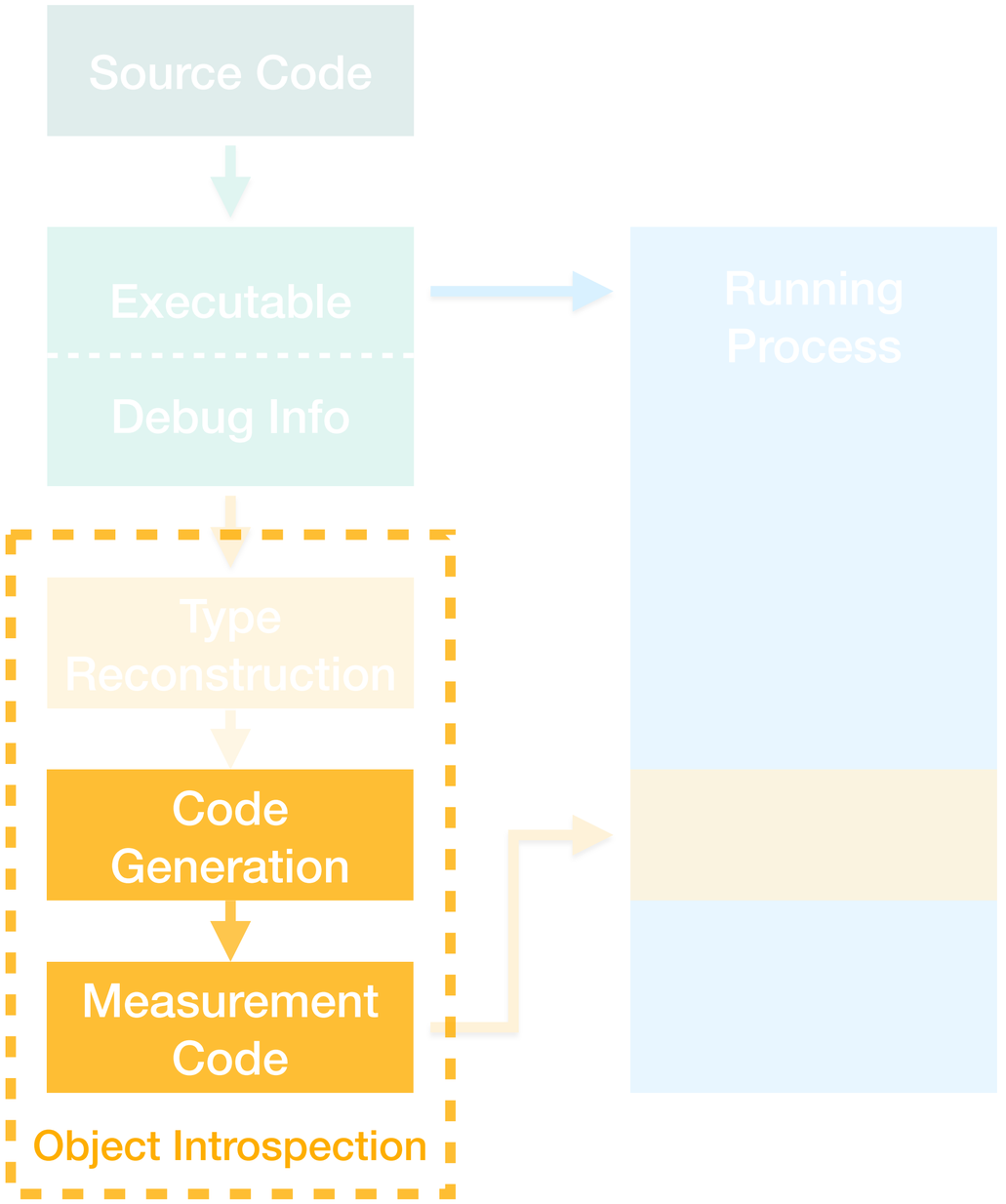
Reconstructed

```
struct Bar {
  std:string str;
};

/* struct MyParent {}; */

// Inline MyParent into Foo
struct Foo                          {
  int a; // from MyParent::a
  int b; // from MyParent::b
  int                 n;
  std::vector<Bar> bar_vec;
  std::string      foo_str;
};
```



Outline

- Debug Info Analysis
  - Type/Layout Reconstruction
  - Code Generation
- Object Introspection as a Library
- Object Introspection as a Profiler
- Object Introspection Applied



CodeGen: `class` & `struct`

```
struct Bar {
  std:string str;
};

struct Foo {
  int                 a;
  int                 b;
  int                 n;
  std::vector<Bar> bar_vec;
  std::string      foo_str;
};
```

```
/* Generated from Debug Info */
static types::st::Unit<DB> getSizeType(
    const Foo& t,
    typename TypeHandler<DB, Foo>::type returnArg) {
 return returnArg
    .delegate([&t](auto ret) { return getSizeType<DB>(t.a, ret); })
    .delegate([&t](auto ret) { return getSizeType<DB>(t.b, ret); })
    .delegate([&t](auto ret) { return getSizeType<DB>(t.n, ret); })
    .delegate([&t](auto ret) { return getSizeType<DB>(t.bar_vec, ret); })
    .consume ([&t](auto ret) { return getSizeType<DB>(t.foo_str, ret); });
}
```

CodeGen: containers

```
template <class T0, class T1>
class std::vector {...};
```

```
template <class T0, class T1>
static types::st::Unit<DB> getSizeType(
    const std::vector<T0, T1>& container,
    typename TypeHandler<DB, std::vector<T0, T1>>::type returnArg) {
  auto tail = returnArg
    .write(container.capacity())
    .write(container.size());
  for (const auto &it : container) {
    tail = tail.delegate([&it](auto ret) {
      return getSizeType<DB>(it, ret);
    });
  }
  return tail;
}
```

CodeGen: `class` & `struct`

```
struct Bar {
  std:string str;
};

struct Foo {
  int                 a;
  int                 b;
  int                 n;
  std::vector<Bar> bar_vec;
  std::string      foo_str;
};
```

```
/* Generated from Debug Info */
static types::st::Unit<DB> getSizeType(
    const Bar& t,
    typename TypeHandler<DB, Bar>::type returnArg) {
 return returnArg
    .consume ([&t](auto ret) { return getSizeType<DB>(t.str, ret); });
}
```

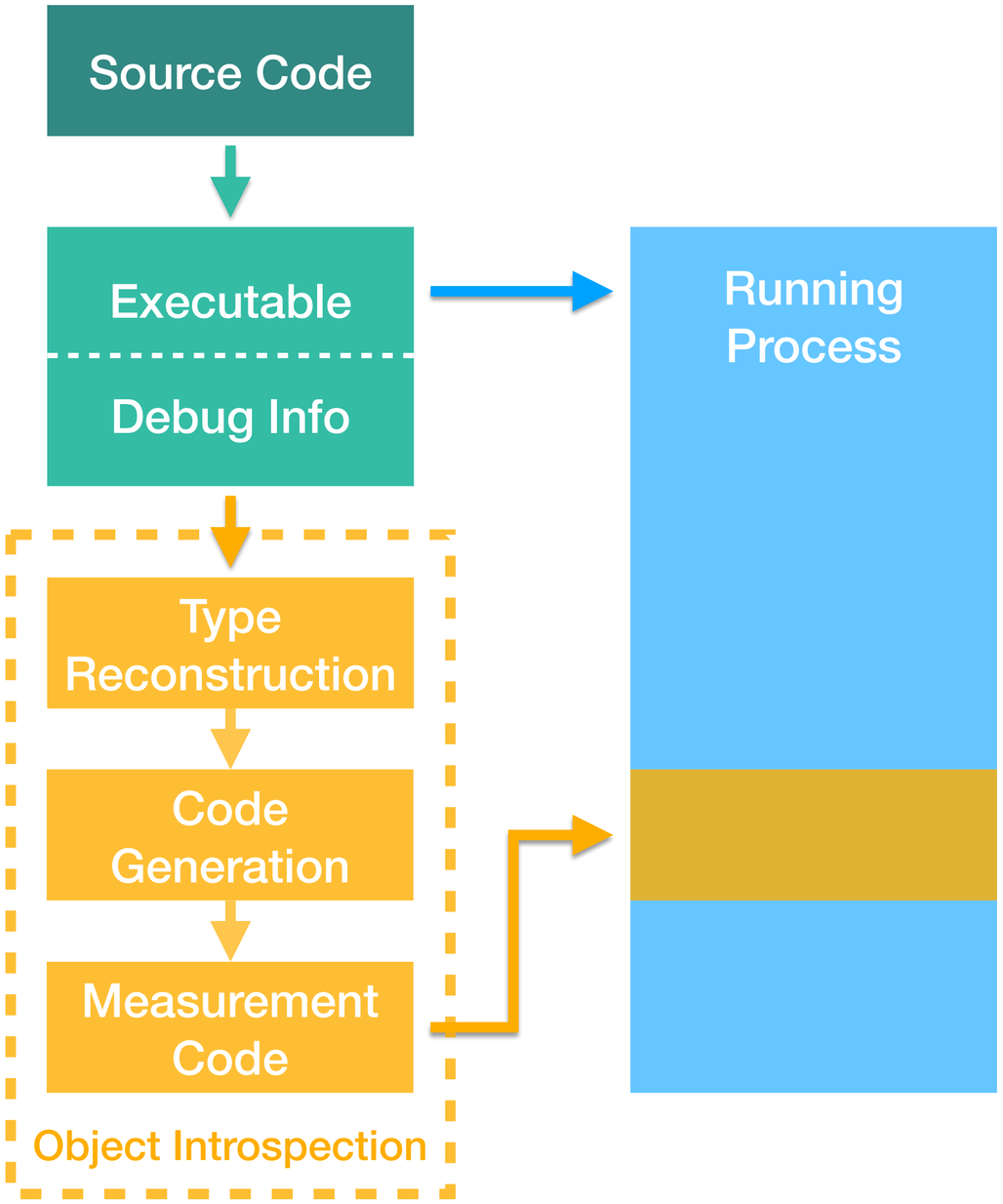
CodeGen: containers

```
template <class T0, class T1, class T2>
class std::basic_string {...};
```

```
template <class T0, class T1, class T2>
static types::st::Unit<DB> getSizeType(
    const std::basic_string<T0, T1, T2>& container,
    typename TypeHandler<DB, std::basic_string<T0, T1, T2>>::type returnArg) {
  bool sso = container.data() >= &container
          && container.data() < (&container + sizeof(container));
  return returnArg
    .write(container.capacity())
    .write(sso)
    .write(container.size());
}
```

Outline

- Debug Info Analysis
  - Type/Layout Reconstruction
  - Code Generation
- Object Introspection as a Library
- Object Introspection as a Profiler
- Object Introspection Applied



Object Introspection as a Library

```
Bar example(const Foo &f) {
  auto result = oi::setupAndIntrospect(f, opts);
  if (result) {
    oi::exporters::Json json(std::cout);
    json.setPretty(true);
    json.print(*result);
  }
}
```

JSON Output

```
struct Bar {
  std:string str;
};

struct Foo {
  int                a;
  int                b;
  int                n;
  std::vector<Bar> bar_vec;
  std::string      foo_str;
};
```

```
{ "name": "foo",
  "typePath": ["foo"],
  "typeNames": ["Foo"],
  "staticSize": 80,
  "exclusiveSize": 4,
  "members": [
    { "name": "a",
      "typePath": ["foo", "a"],
      "typeNames": ["int32_t"],
      "staticSize": 4,
      "exclusiveSize": 4,
    }, {
      "name": "b",
      "typePath": ["foo", "b"],
      [...]
```

JSON Output

```
[...],
{ "name": "bar_vec",
  "typePath": ["foo", "bar_vec"],
  "typeNames": ["std::vector<Bar, std::allocator<Bar>>"],
  "length": 155,
  "capacity": 256,
  "staticSize": 24,
  "exclusiveSize": 3256,
  "members": [
    { "name": "[]",
      "typePath": ["foo", "bar_vec", "[]"],
      "typeNames": ["Bar"],
      "staticSize": 32,
      "exclusiveSize": 0,
      "members": [
        { "name": "str",
```

```
struct Bar {
  std:string str;
};

struct Foo {
  int                 a;
  int                 b;
  int                 n;
  std::vector<Bar> bar_vec;
  std::string      foo_str;
};
```

JSON Output

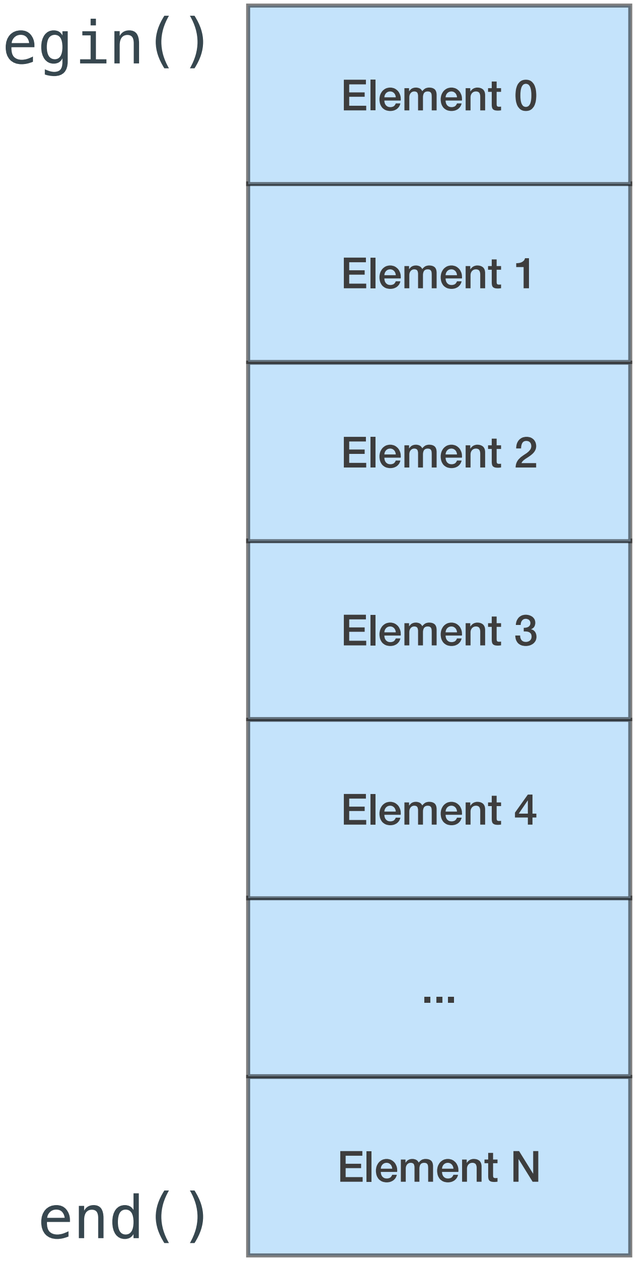
```
struct Bar {
  std:string str;
};

struct Foo {
  int                a;
  int                b;
  int                n;
  std::vector<Bar> bar_vec;
  std::string      foo_str;
};
```

```
      { "name": "[]",
        "typePath": ["foo", "bar_vec", "[]"],
        "typeNames": ["Bar"],
        "staticSize": 32,
        "exclusiveSize": 0,
        "members": [
          { "name": "str",
            "typePath": ["foo", "bar_vec", "[]", "str"],
            "typeNames": ["string", "std::basic_string<int8_t,
std::char_traits<int8_t>, std::allocator<int8_t>>"],
            "length": 12,
            "capacity": 15,
            "staticSize": 32,
            "exclusiveSize": 32
          }
        ]
```

Result Iterator

```
IntrospectionResult result =
  oi::setupAndIntrospect(f, opts);
```

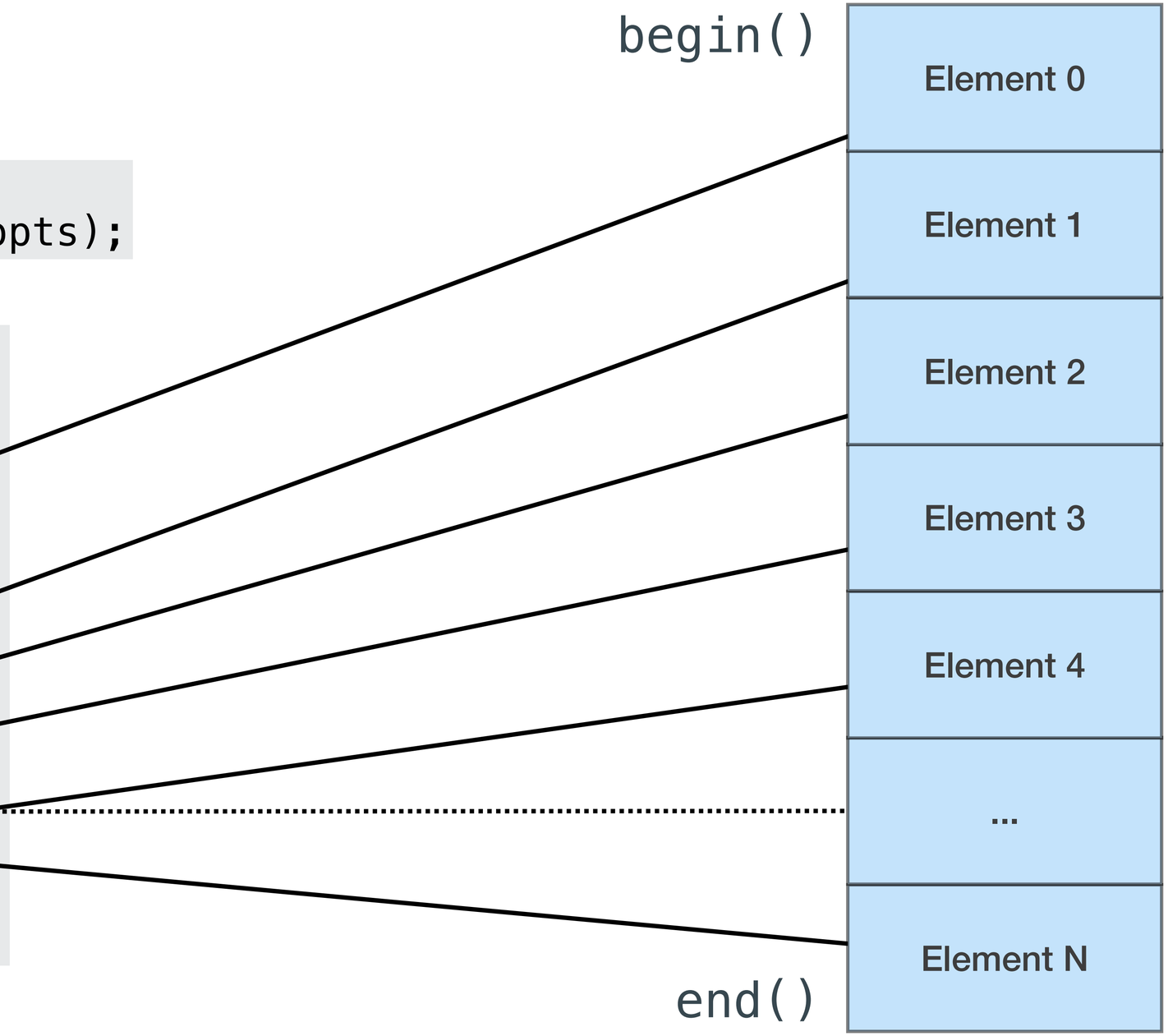
```
struct Bar {
  std:string        str;
};

struct Foo {
  int               a;
  int               b;
  int               n;
  std::vector<Bar> bar_vec;
  std::string       foo_str;
};
```



Result Iterator

```
IntrospectionResult result =
  oi::setupAndIntrospect(f, opts);
```

```
struct Bar {
  std:string        str;
};

struct Foo {
  int               a;
  int               b;
  int               n;
  std::vector<Bar> bar_vec;
  std::string       foo_str;
};
```



OIL: String Hunting

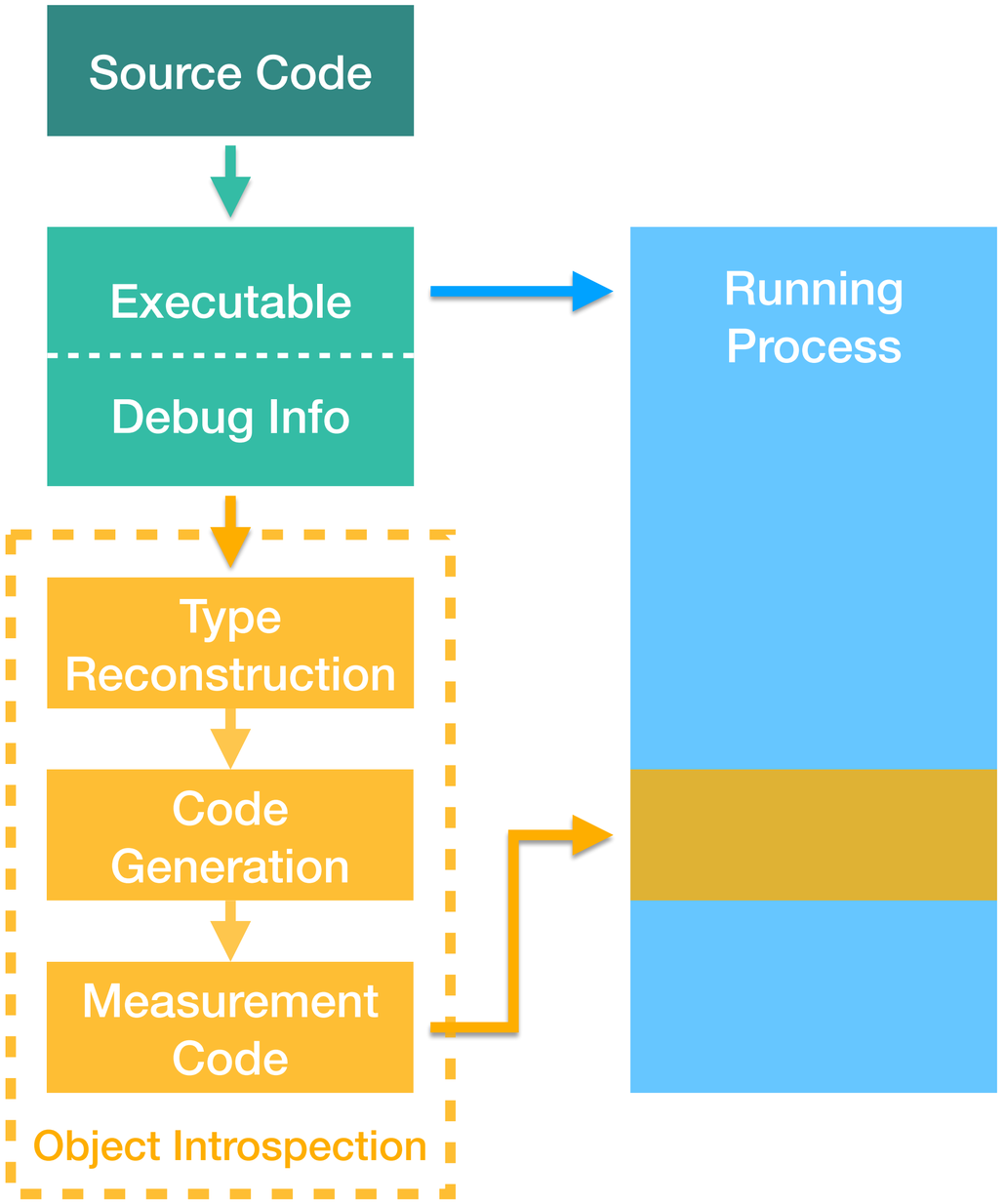
```
std::cout << "Field Name  Length Capacity\n";
std::cout << "==========  ====== ========\n";
for (const auto &el: results) {
  if (std::range::find(el.type_names, "std::string") != el.type_names.end()) {
    std::cout << std::format("{:<10}  {:>6} {:>8}\n",
      el.name, el.container_stats->length, el.container_stats->capacity);
  }
}
```

```
struct Bar {
  std:string str;
};

struct Foo {
  int              a;
  int              b;
  int              n;
  std::vector<Bar> bar_vec;
  std::string      foo_str;
};
```

```
Field Name  Length Capacity
==========  ====== ========
str             12       15
str             85       85
str             20       20
str             57       57
str             46       46
str             69       69
foo_str          0       15
```

Outline

- Debug Info Analysis
  - Type/Layout Reconstruction
  - Code Generation
- Object Introspection as a Library
- Object Introspection as a Profiler
- Object Introspection Applied



Object Introspection as a Profiler

- Classic ‘tracer-tracee’ style application
  - Type reconstruction / Code Generation
  - Code injection
  - Process/thread management
  - Result processing
- Non-interactive, configuration driven

Running the Profiler

```
Bar example(Foo &f);
```

Function arguments at entry and return

```
entry:_Z7exampleR3Foo:arg0
return:_Z7exampleR3Foo:arg0
return:_Z7exampleR3Foo:retval
```

```
$ oid --json --pid `pidof target` -S "entry:_Z7exampleR3Foo:arg0"
```

Probe examples

Multiple arguments

```
return:_Z8doitR3FooS0_:arg0,arg1 // void doit(Foo&, Foo&)
```

‘this’ pointer

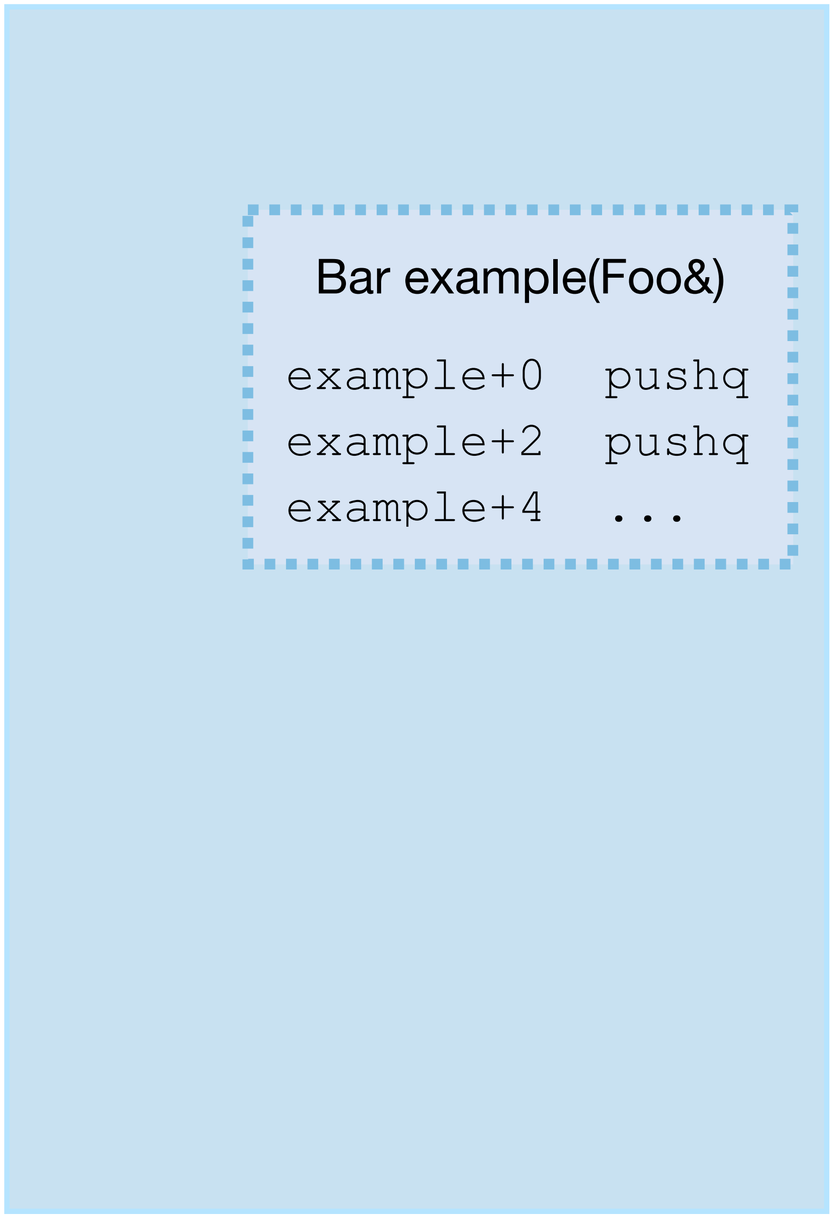
```
entry:_ZN3FooD1Ev:this          // Foo::~Foo()
```

Global data

```
global:myGlobalState
```

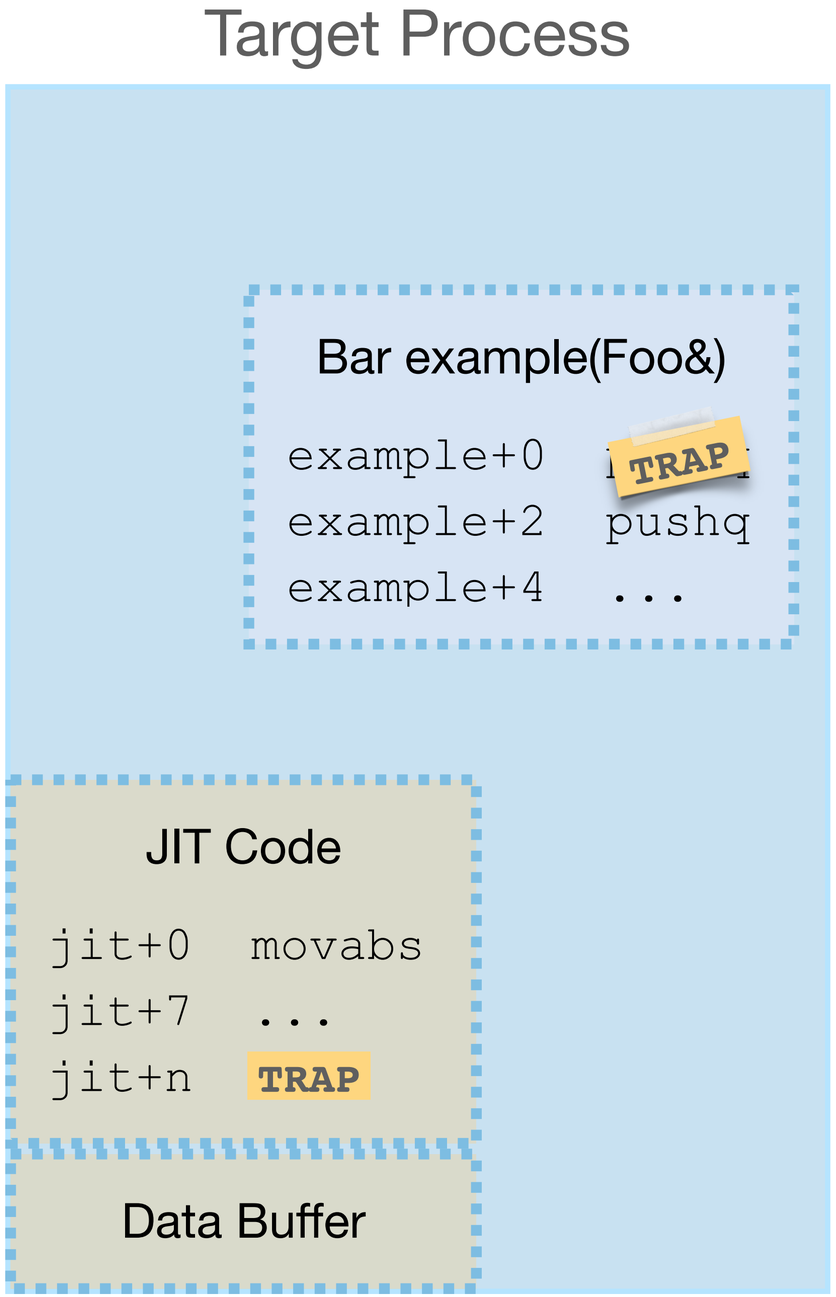
OI as a Profiler

1. Inject OI into the target process
2. Wait for a thread to hit our trap...
3. Redirect the thread to run our code
4. Put the redirected thread back on its tracks
5. Process the collected data



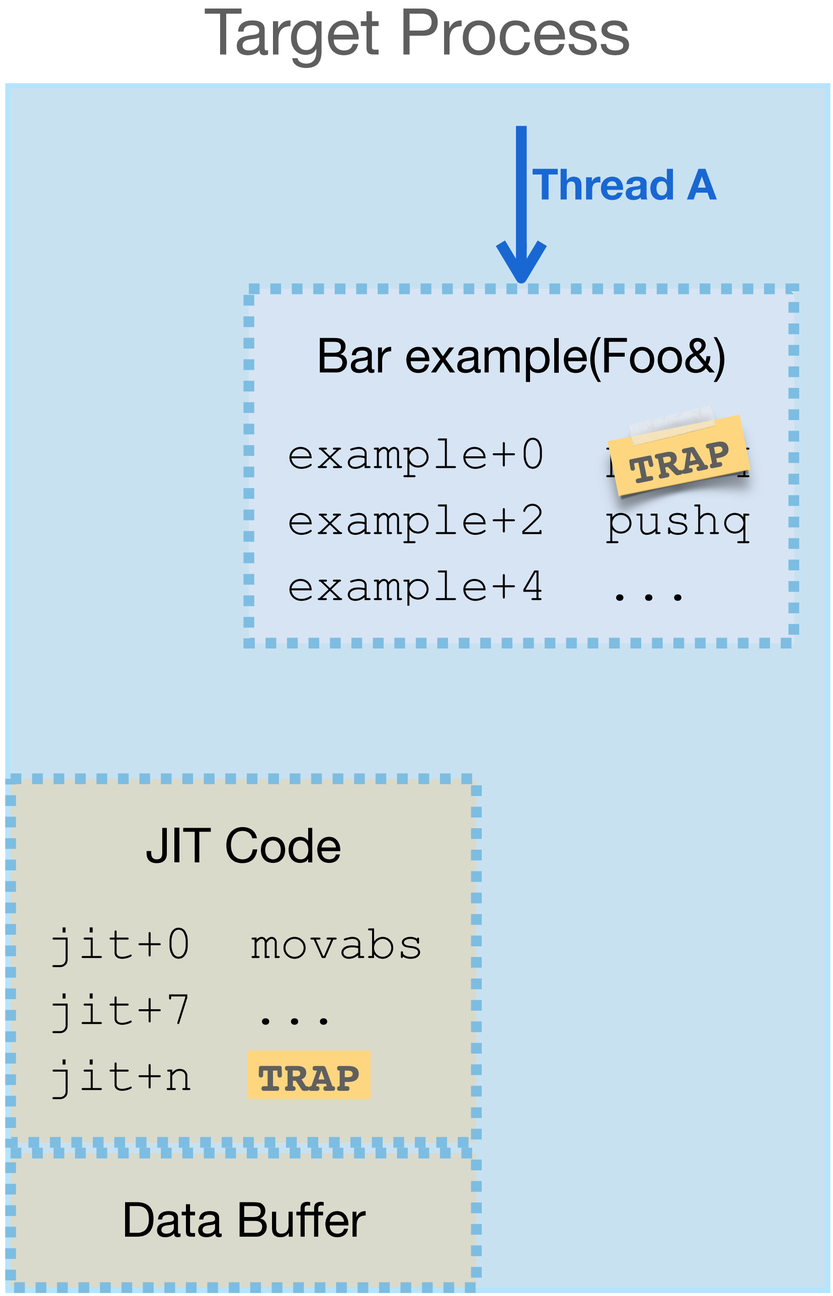
OI as a Profiler

1. Inject OI into the target process
2. Wait for a thread to hit our trap...
3. Redirect the thread to run our code
4. Put the redirected thread back on its tracks
5. Process the collected data



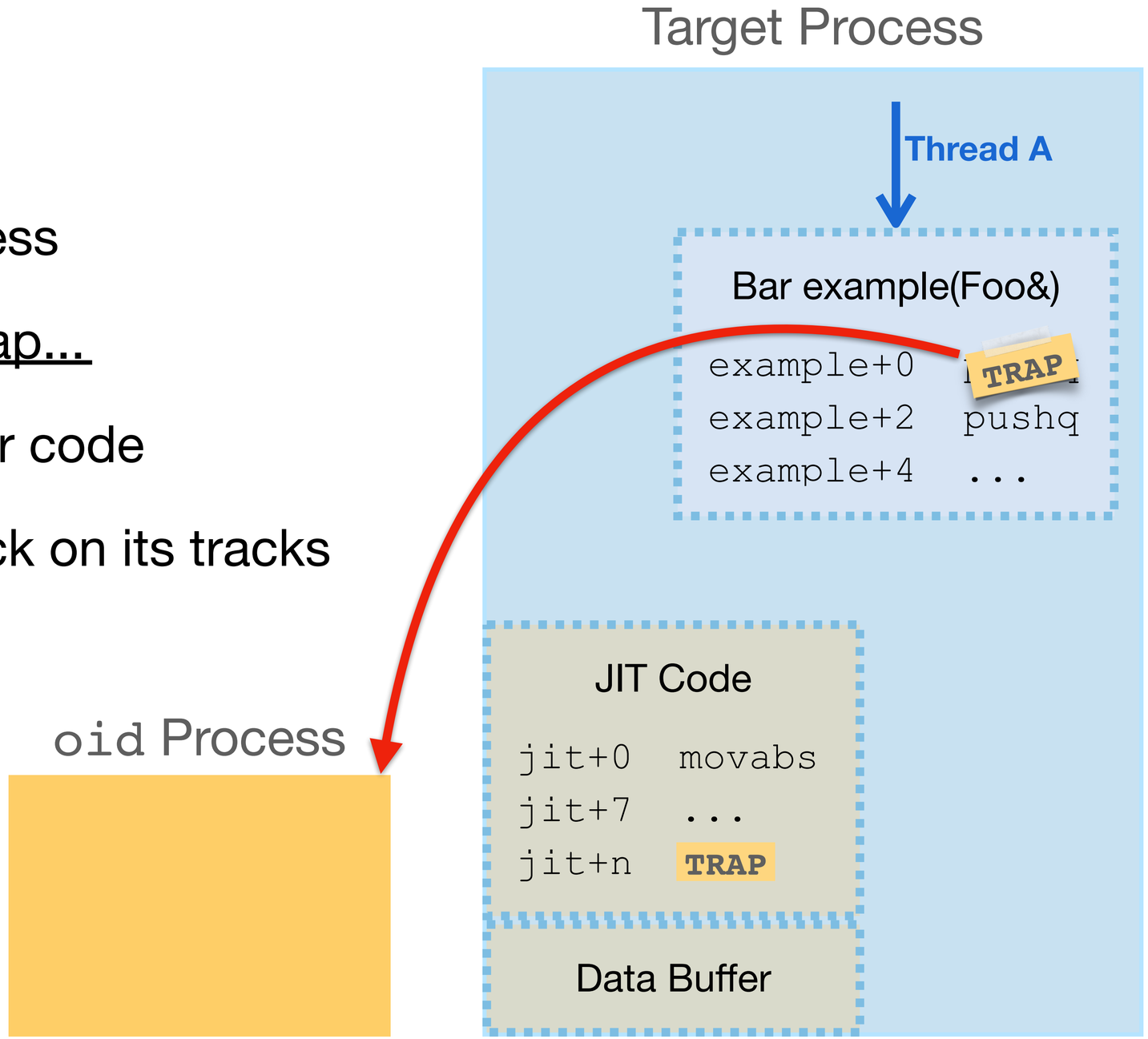
OI as a Profiler

1. Inject OI into the target process
2. Wait for a thread to hit our trap...
3. Redirect the thread to run our code
4. Put the redirected thread back on its tracks
5. Process the collected data



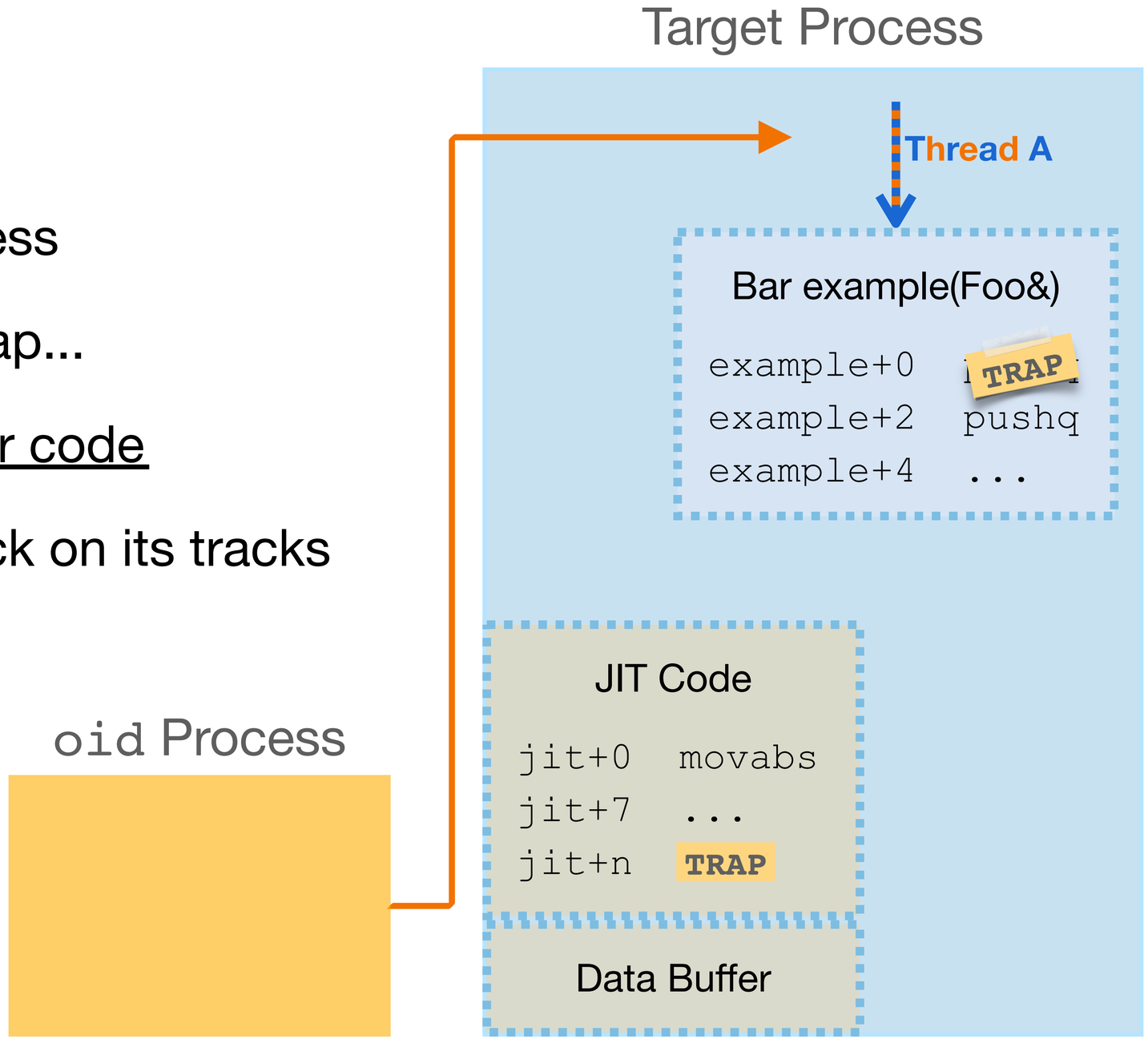
OI as a Profiler

1. Inject OI into the target process
2. Wait for a thread to hit our trap...
3. Redirect the thread to run our code
4. Put the redirected thread back on its tracks
5. Process the collected data



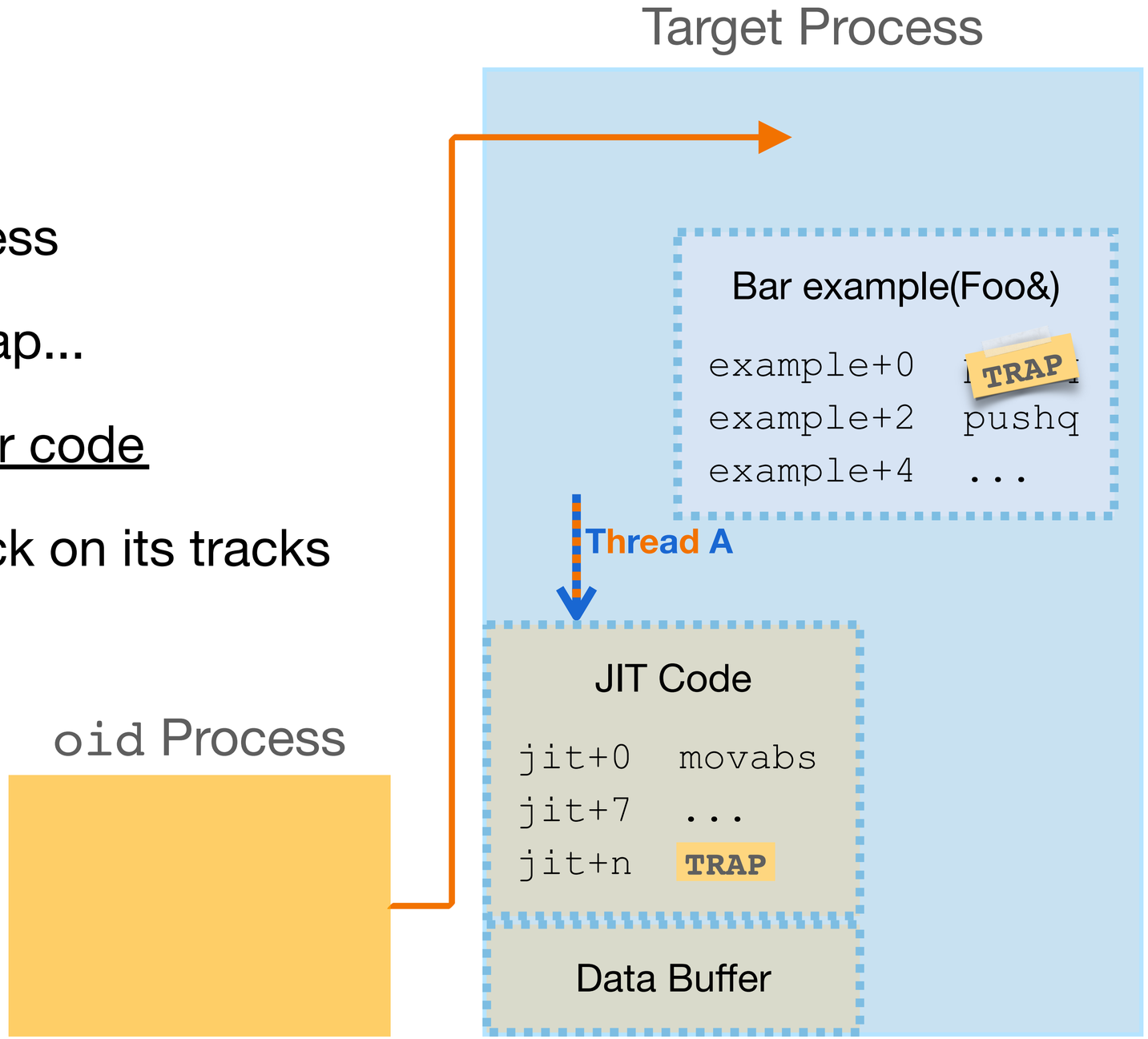
OI as a Profiler

1. Inject OI into the target process
2. Wait for a thread to hit our trap...
3. Redirect the thread to run our code
4. Put the redirected thread back on its tracks
5. Process the collected data



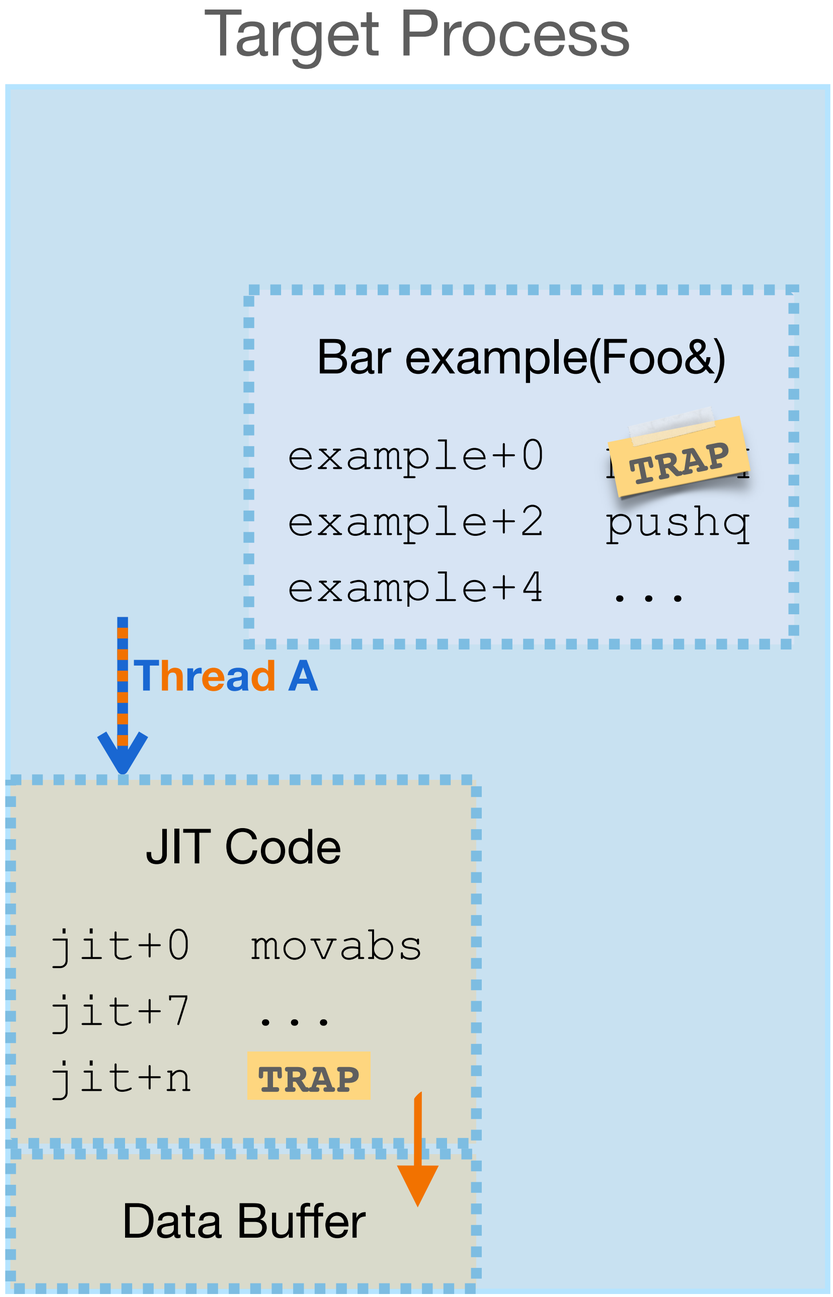
OI as a Profiler

1. Inject OI into the target process
2. Wait for a thread to hit our trap...
3. Redirect the thread to run our code
4. Put the redirected thread back on its tracks
5. Process the collected data



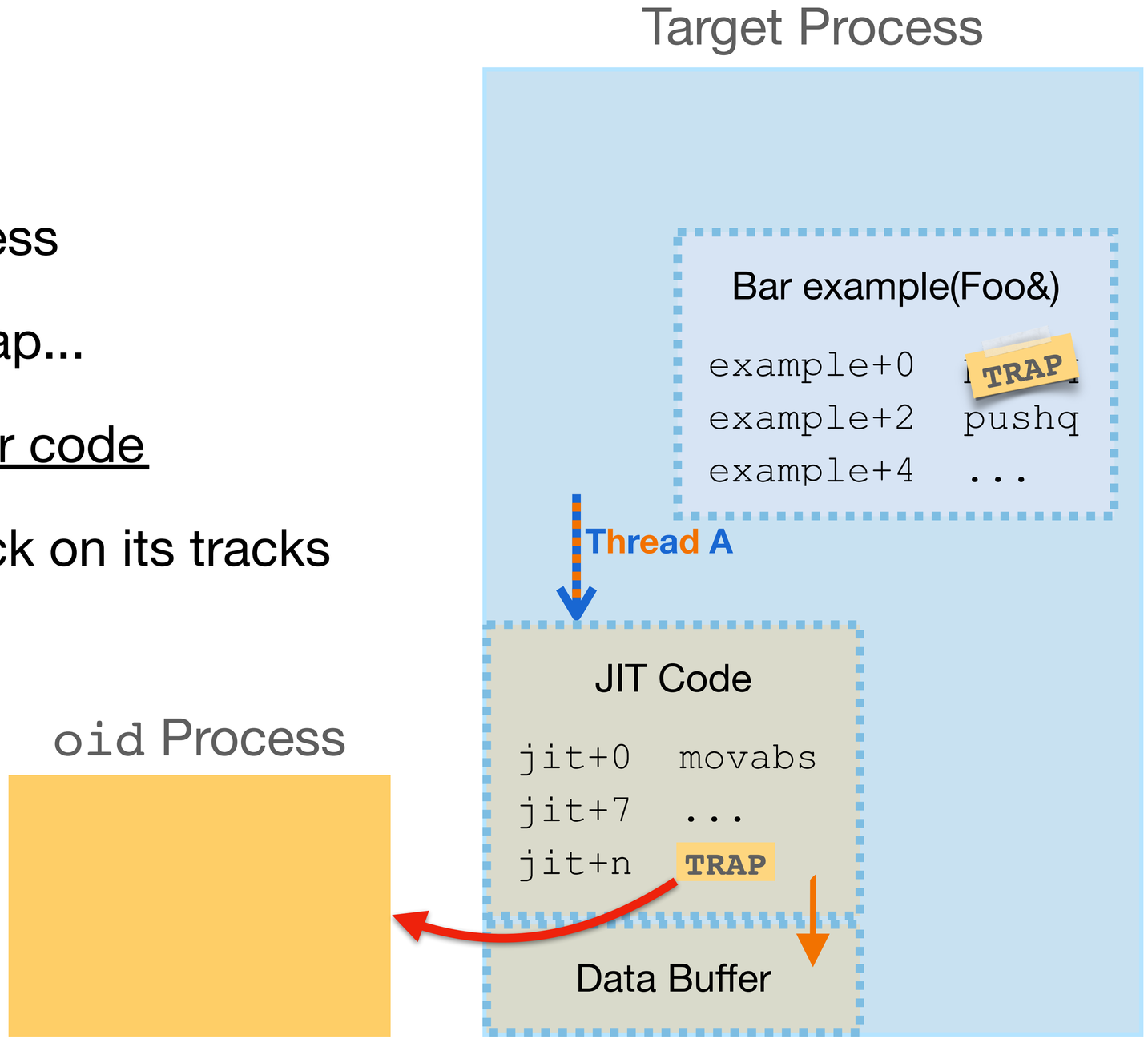
OI as a Profiler

1. Inject OI into the target process
2. Wait for a thread to hit our trap...
3. Redirect the thread to run our code
4. Put the redirected thread back on its tracks
5. Process the collected data



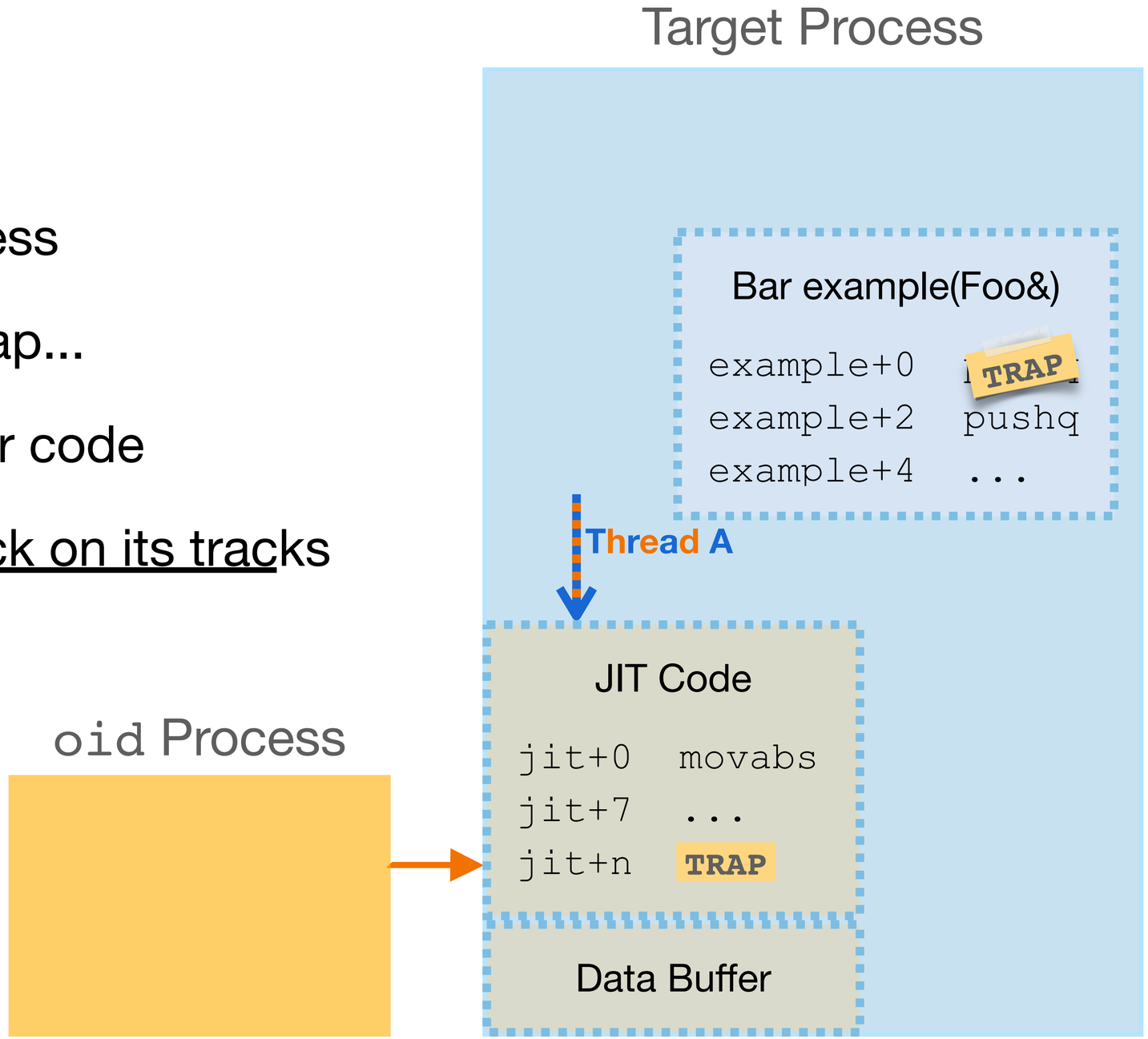
OI as a Profiler

1. Inject OI into the target process
2. Wait for a thread to hit our trap...
3. Redirect the thread to run our code
4. Put the redirected thread back on its tracks
5. Process the collected data



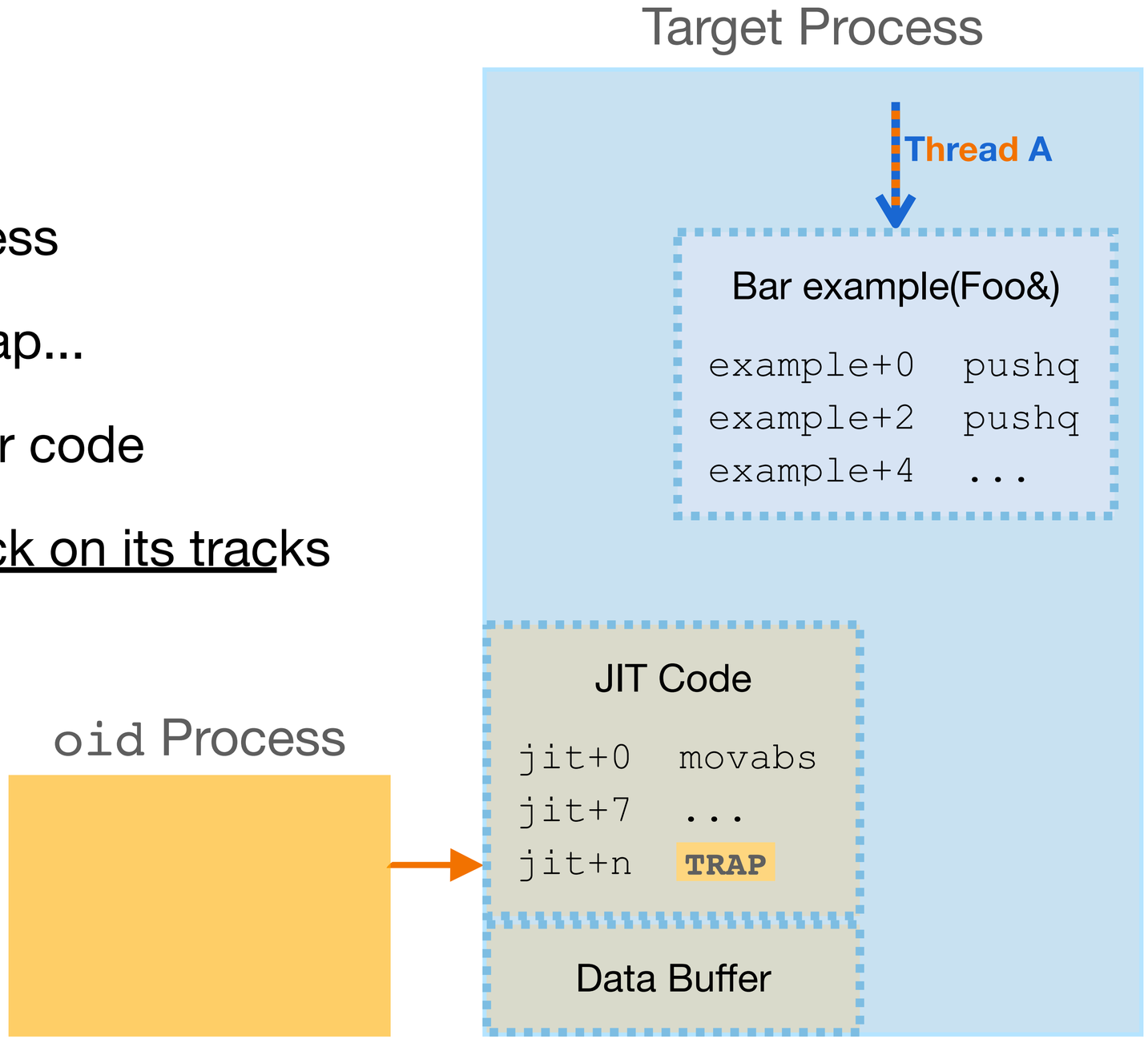
OI as a Profiler

1. Inject OI into the target process
2. Wait for a thread to hit our trap...
3. Redirect the thread to run our code
4. Put the redirected thread back on its tracks
5. Process the collected data



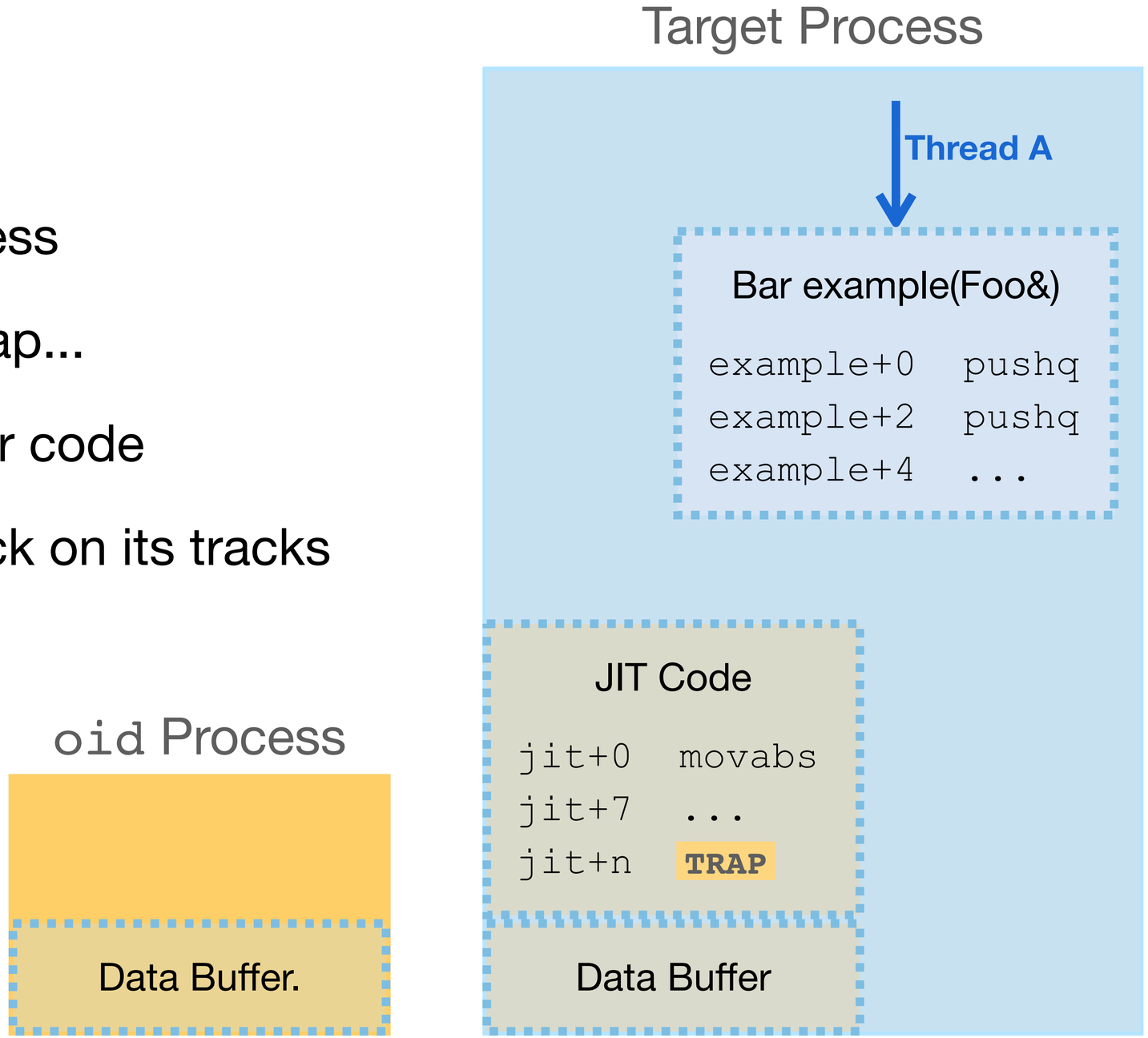
OI as a Profiler

1. Inject OI into the target process
2. Wait for a thread to hit our trap...
3. Redirect the thread to run our code
4. Put the redirected thread back on its tracks
5. Process the collected data



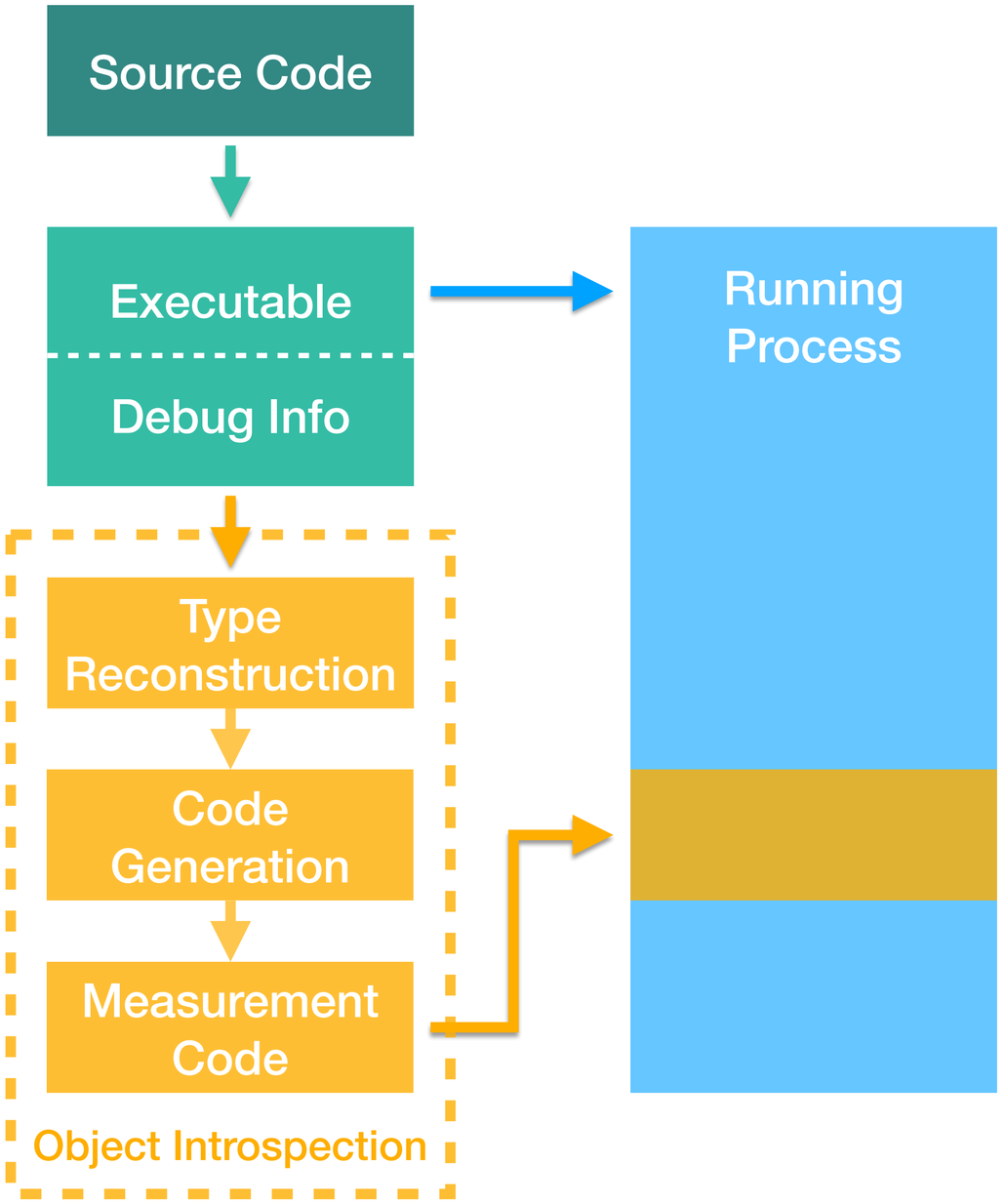
OI as a Profiler

1. Inject OI into the target process
2. Wait for a thread to hit our trap...
3. Redirect the thread to run our code
4. Put the redirected thread back on its tracks
5. Process the collected data



Outline

- Debug Info Analysis
  - Type/Layout Reconstruction
  - Code Generation
- Object Introspection as a Library
- Object Introspection as a Profiler
- Object Introspection Applied



Applied Example 1

- Unused container memory:

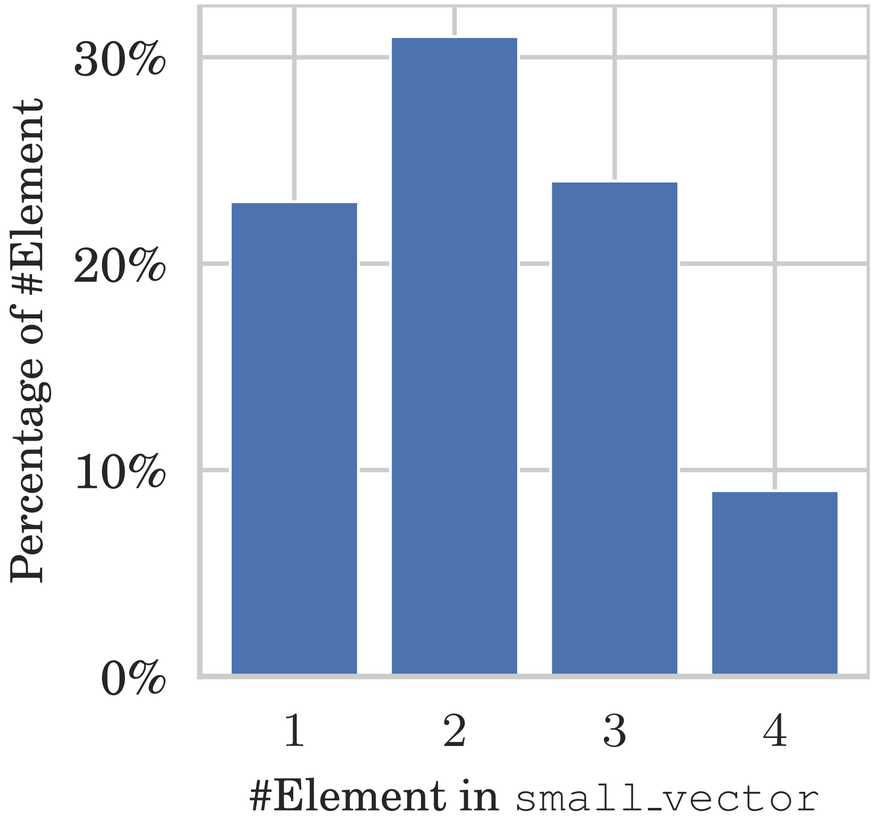
  Unused Sz = (C.capacity() - C.size()) * sizeof(element)

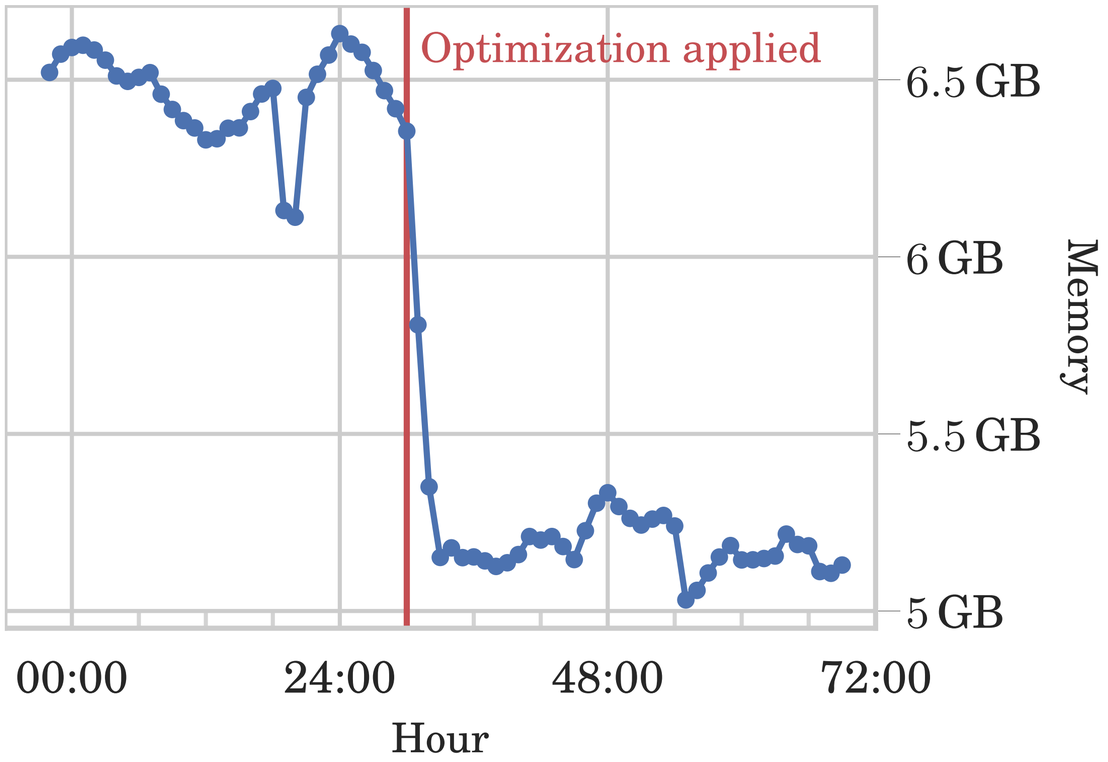
- 2800 nested objects of type ‘FooBar’ were analysed. Top of the list:

Name	TypeName	Number	ElemStatSz	Length	Capacity	DynSz	TotalSz	UnusedSz (Cumulative)
Feature	folly::small_vector<X, 10>	414132	8	2.359	10	0	88	25315061

- UnusedSz (cumulative) = (10 - 2.359) * 8 * 414132

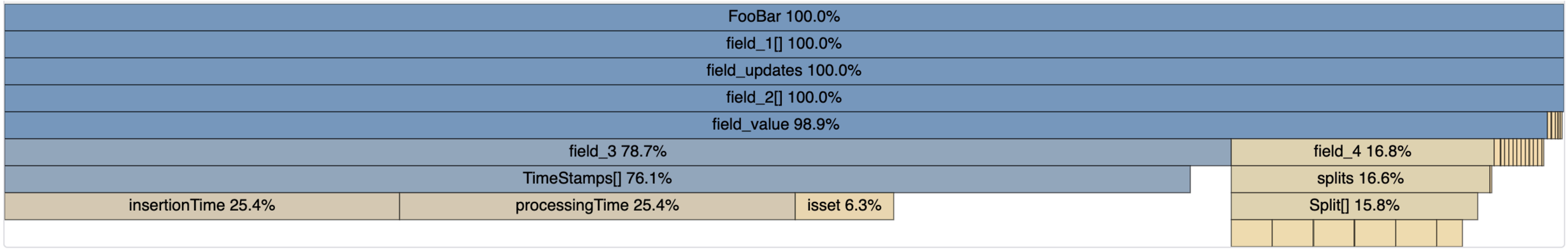
Applied Example 1: Results





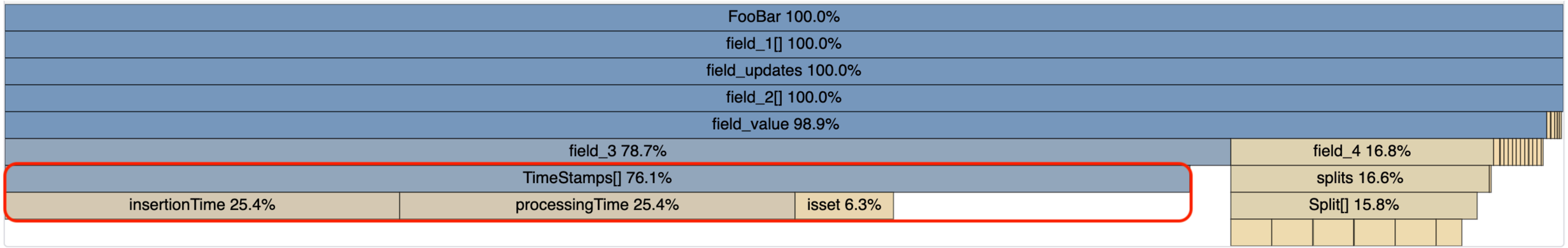
Applied Example 2

- 1000 objects of type ‘FooBar’ were sampled and introspected



Applied Example 2

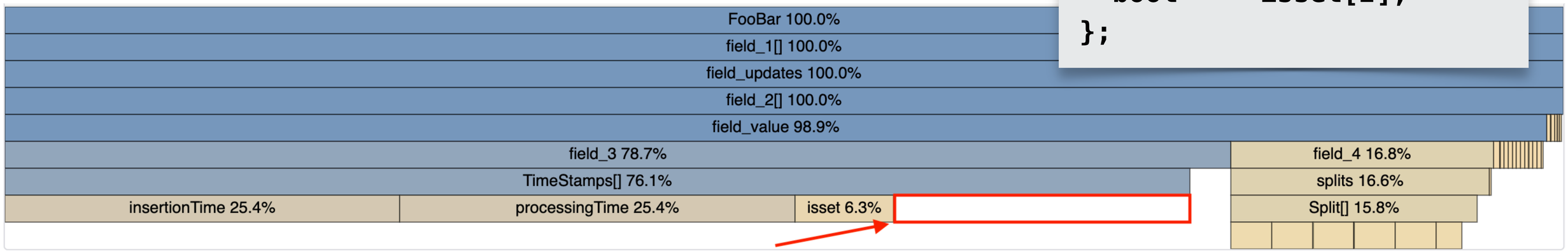
- 1000 objects of type ‘FooBar’ were sampled and introspected



Applied Example 2

- 1000 objects of type ‘FooBar’ were sampled and introspected

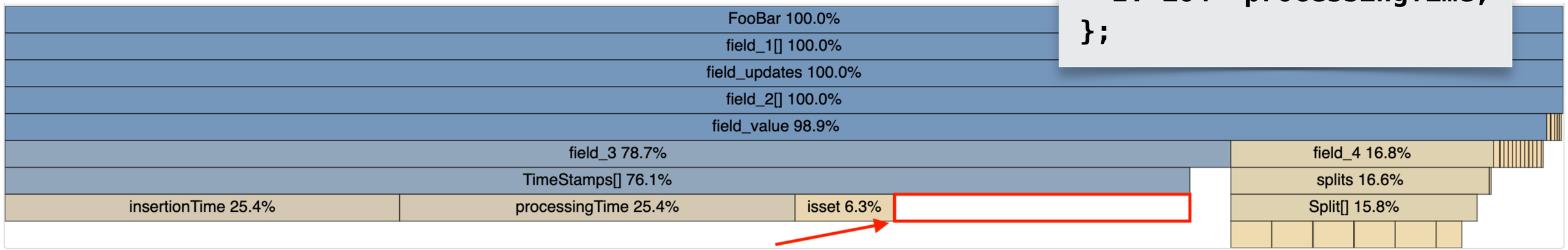
```
struct TimeStamps {
  int64_t  insertionTime;
  int64_t  processingTime;
  bool     isset[2];
};
```



Applied Example 2

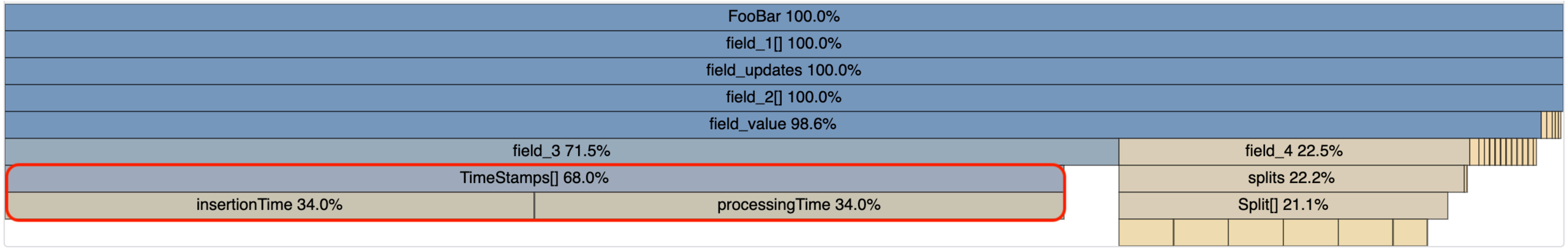
- 1000 objects of type ‘FooBar’ were sampled and introspected

```
// Thrift source
struct TimeStamps {
  1: i64  insertionTime;
  2: i64  processingTime;
};
```



Applied Example 2

- A thrift annotation, gets rid of isset field to avoid padding
- Profile collected after the change was applied
- Saved ~10% of p95 memory for that service



Gotchas / limitations

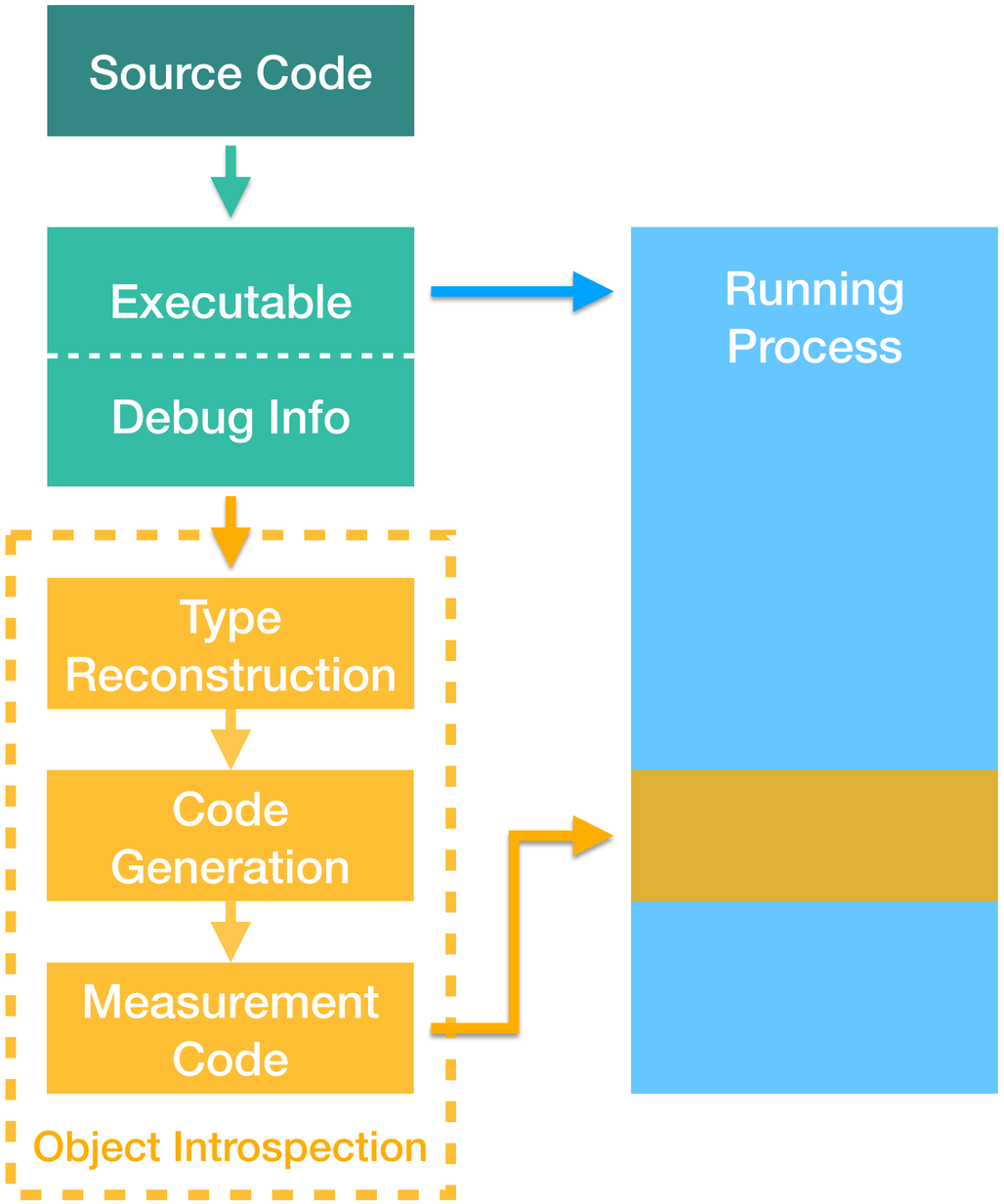
- Currently Linux, DWARF + x86_64 only.
- Dynamic linking
- Debug binaries
- (oid) Inlining!
- Data races

Gotchas / limitations

- User defined containers
- Template specialisation of containers
- Virtual Inheritance
- C style unions
- Pointers & `void*`

Outline

- Debug Info Analysis
  - Type/Layout Reconstruction
  - Code Generation
- Object Introspection as a Library
- Object Introspection as a Profiler
- Object Introspection Applied




object
introspection.org

Object Introspection:

A Revolutionary Memory Profiler for C++ Objects

JONATHAN HASLAM
& ADITYA SARWADE