NATIVE WINDOWS JITING IN LLVM

JITLINK

by Sunho Kim

OUTLINE

OUTLINE

How does JIT work in LLVM

OUTLINE

How does JIT work in LLVM

Motivation

OUTLINE

How does JIT work in LLVM

Motivation

Windows COFF JITLink example

OUTLINE

How does JIT work in LLVM

Motivation

Windows COFF JITLink example

Windows COFF JITLink plugin example

OUTLINE

How does JIT work in LLVM

Motivation

Windows COFF JITLink example

Windows COFF JITLink plugin example

Tips on using JITLink in COFF

OUTLINE

How does JIT work in LLVM

Motivation

Windows COFF JITLink example

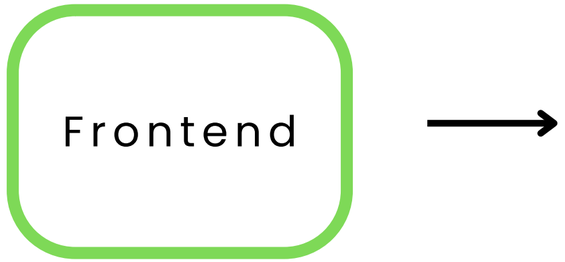
Windows COFF JITLink plugin example

Tips on using JITLink in COFF

clang-repl demo

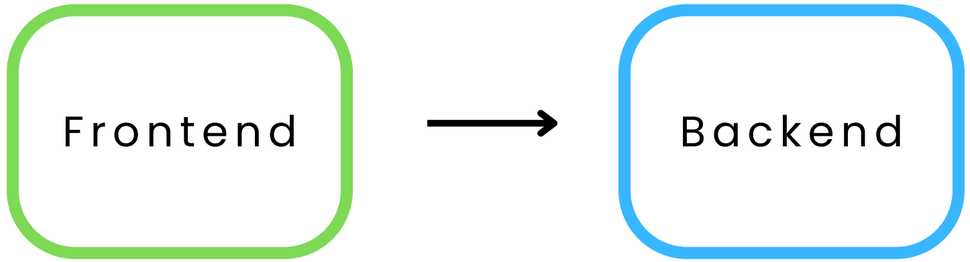
HOW DOES JIT WORK IN LLVM

Usual executable generation pipeline in LLVM



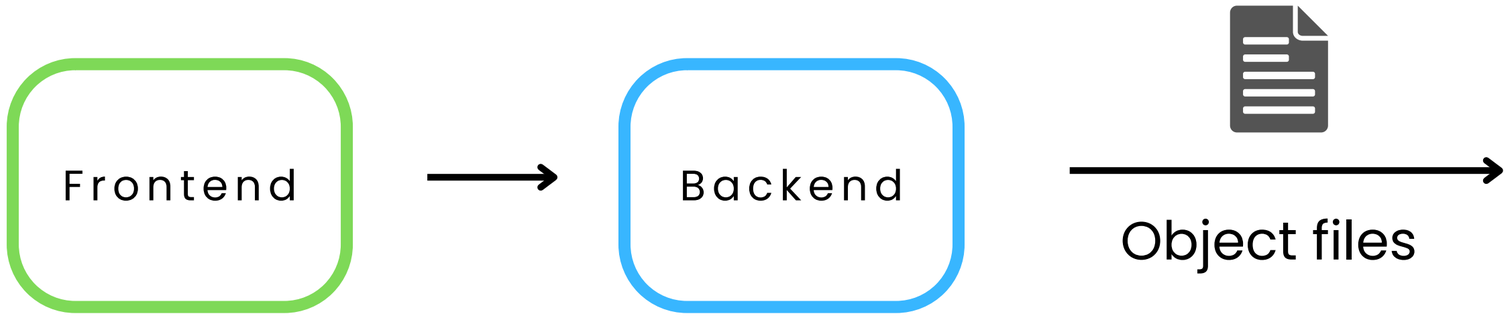
HOW DOES JIT WORK IN LLVM

Usual executable generation pipeline in LLVM



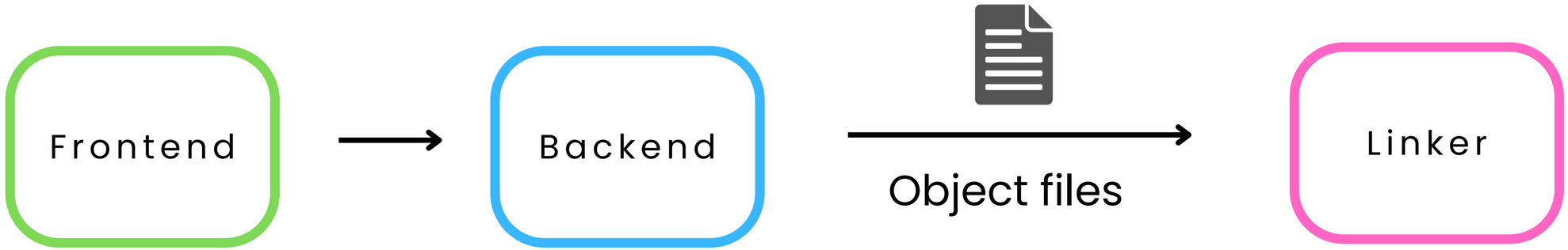
HOW DOES JIT WORK IN LLVM

Usual executable generation pipeline in LLVM



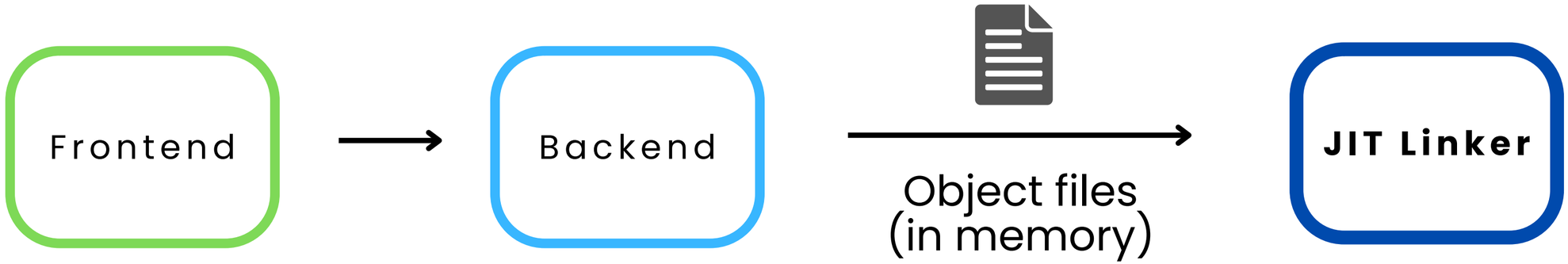
HOW DOES JIT WORK IN LLVM

Usual executable generation pipeline in LLVM



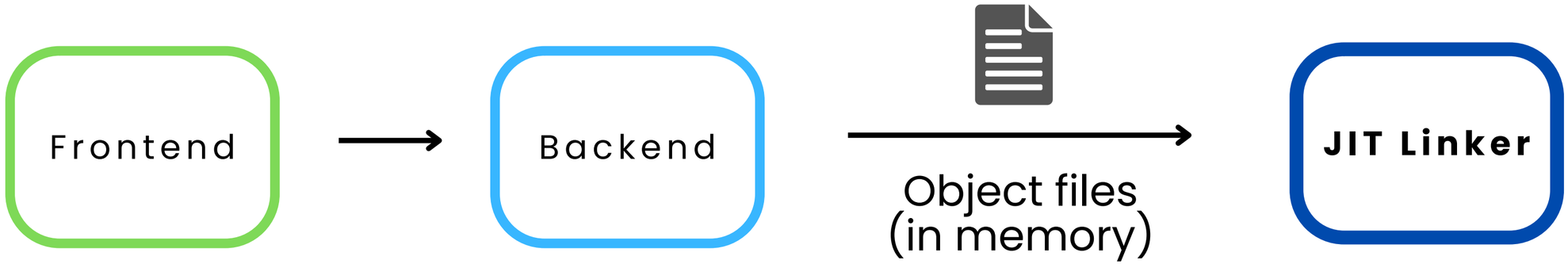
HOW DOES JIT WORK IN LLVM

JIT execution pipeline in LLVM



HOW DOES JIT WORK IN LLVM

JIT execution pipeline in LLVM



- Share a huge portion of pipeline with AOT
- Fewer breakage by LLVM internal code changes

MOTIVATION FOR JITLINK

Old JIT linker: RuntimeDyld

MOTIVATION FOR JITLINK

Old JIT linker: RuntimeDyld

- Small code model unsupported

MOTIVATION FOR JITLINK

Old JIT linker: RuntimeDyld

- Small code model unsupported
- Static initializers or thread local storage (TLS) supported in limited ways

MOTIVATION FOR JITLINK

Old JIT linker: RuntimeDyld

- Small code model unsupported
- Static initializers or thread local storage (TLS) supported in limited ways
- Developed in ad-hoc fashion

MOTIVATION FOR JITLINK

Old JIT linker: RuntimeDyld

- Small code model unsupported
- Static initializers or thread local storage (TLS) supported in limited ways
- Developed in ad-hoc fashion
- COFF support existed but very unstable

MOTIVATION FOR JITLINK

Old JIT linker: RuntimeDyld

- Small code model unsupported
- Static initializers or thread local storage (TLS) supported in limited ways
- Developed in ad-hoc fashion
- COFF support existed but very unstable
    - People used ELF on Windows

MOTIVATION FOR JITLINK

Old JIT linker: RuntimeDyld

- Small code model unsupported
- Static initializers or thread local storage (TLS) supported in limited ways
- Developed in ad-hoc fashion
- COFF support existed but very unstable
  - People used ELF on Windows

New JIT linker: JITLink

MOTIVATION FOR JITLINK

Old JIT linker: RuntimeDyld

- Small code model unsupported
- Static initializers or thread local storage (TLS) supported in limited ways
- Developed in ad-hoc fashion
- COFF support existed but very unstable
  - People used ELF on Windows

New JIT linker: JITLink

- Small code model aware memory allocator

MOTIVATION FOR JITLINK

Old JIT linker: RuntimeDyld

- Small code model unsupported
- Static initializers or thread local storage (TLS) supported in limited ways
- Developed in ad-hoc fashion
- COFF support existed but very unstable
  - People used ELF on Windows

New JIT linker: JITLink

- Small code model aware memory allocator
- Runtime features fully supported including static initializers and thread local storage

MOTIVATION FOR JITLINK

Old JIT linker: RuntimeDyld

- Small code model unsupported
- Static initializers or thread local storage (TLS) supported in limited ways
- Developed in ad-hoc fashion
- COFF support existed but very unstable
  - People used ELF on Windows

New JIT linker: JITLink

- Small code model aware memory allocator
- Runtime features fully supported including static initializers and thread local storage
- Generic linker object abstraction LinkGraph

MOTIVATION FOR JITLINK

Old JIT linker: RuntimeDyld

- Small code model unsupported
- Static initializers or thread local storage (TLS) supported in limited ways
- Developed in ad-hoc fashion
- COFF support existed but very unstable
    - People used ELF on Windows

New JIT linker: JITLink

- Small code model aware memory allocator
- Runtime features fully supported including static initializers and thread local storage
- Generic linker object abstraction LinkGraph
- Easy to fully implement native object file features

COFF SUPPORT IN JITLINK

COFF SUPPORT IN JITLINK

- Capable of linking object files **generated by MSVC**

COFF SUPPORT IN JITLINK

- Capable of linking object files **generated by MSVC**
- **COMDATs, WeakExternal, linker directive, dllimport stub, or CRT initializer** properly implemented

COFF SUPPORT IN JITLINK

- Capable of linking object files **generated by MSVC**
- **COMDATs, WeakExternal, linker directive, dllimport stub, or CRT initializer** properly implemented
- Able to jit-link the **VC runtime library**
  - Loading up msvcrt.lib ucrt.lib into JIT session
  - Static version of VC runtime works too

COFF SUPPORT IN JITLINK

- Capable of linking object files **generated by MSVC**
- **COMDATs, WeakExternal, linker directive, dllimport stub, or CRT initializer** properly implemented
- Able to jit-link the **VC runtime library**
  - Loading up msvcrt.lib ucrt.lib into JIT session
  - Static version of VC runtime works too
- Linking **Microsoft STL library** work out of shelf

COFF SUPPORT IN JITLINK

- Capable of linking object files **generated by MSVC**
- **COMDATs, WeakExternal, linker directive, dllimport stub, or CRT initializer** properly implemented
- Able to jit-link the **VC runtime library**
  - Loading up msvcrt.lib ucrt.lib into JIT session
  - Static version of VC runtime works too
- Linking **Microsoft STL library** work out of shelf
- **Incremental linking** works by default

WINDOWS COFF JITLINK EXAMPLE

LLVM IR executor

We're going to build a simple JIT application

WINDOWS COFF JITLINK EXAMPLE

LLVM IR executor

We're going to build a simple JIT application

- Executes the **LLVM IRs** written inside main.ll using JIT

WINDOWS COFF JITLINK EXAMPLE

LLVM IR executor

We're going to build a simple JIT application

- Executes the **LLVM IRs** written inside main.ll using JIT
- main.ll can be generated from any frontend such as clang or flang

WINDOWS COFF JITLINK EXAMPLE

LLVM IR executor

We're going to build a simple JIT application

- Executes the **LLVM IRs** written inside main.ll using JIT
- main.ll can be generated from any frontend such as clang or flang

Start by implementing LLVM IR executor and add advanced JIT usages on top of it

WINDOWS COFF JITLINK EXAMPLE

LLVM IR executor

https://gist.github.com/sunho/12f14f61309323bfd88832f94056e68d

Setup code template

WINDOWS COFF JITLINK EXAMPLE

LLVM IR executor

LLJIT::loadOrcRuntime function can be used to load orc runtime into JIT session.

orc_rt-x86_64.lib file is inside compiler-rt build

WINDOWS COFF JITLINK EXAMPLE

LLVM IR executor

After ORC runtime is loaded, many features just work including:

WINDOWS COFF JITLINK EXAMPLE

LLVM IR executor

After ORC runtime is loaded, many features just work including:

- sin, cos functions from vc runtime library

WINDOWS COFF JITLINK EXAMPLE

LLVM IR executor

After ORC runtime is loaded, many features just work including:

- sin, cos functions from vc runtime library
- std::map, std::mutex, std::cout, and more

WINDOWS COFF JITLINK EXAMPLE

LLVM IR executor

After ORC runtime is loaded, many features just work including:

- sin, cos functions from vc runtime library
- std::map, std::mutex, std::cout, and more
- c++ exception support

WINDOWS COFF JITLINK EXAMPLE

LLVM IR executor

After ORC runtime is loaded, many features just work including:

- sin, cos functions from vc runtime library
- std::map, std::mutex, std::cout, and more
- c++ exception support
- Structured Exception Handling (SEH) support

WINDOWS COFF JITLINK EXAMPLE

LLVM IR executor

Loading static library built by MSVC into JIT session

WINDOWS COFF JITLINK EXAMPLE

LLVM IR executor

Loading static library built by MSVC into JIT session

```
1     auto G = ExitOnErr(StaticLibraryDefinitionGenerator::Load(
2       J->getObjLinkingLayer(), "StaticLib1.lib")
3     );
4     J->getMainJITDylib().addGenerator(std::move(G));
```

WINDOWS COFF JITLINK EXAMPLE

LLVM IR executor

Loading static library built by MSVC into JIT session

```
1     auto G = ExitOnErr(StaticLibraryDefinitionGenerator::Load(
2       J->getObjLinkingLayer(), "StaticLib1.lib")
3     );
4     J->getMainJITDylib().addGenerator(std::move(G));
```

Notice

- object files generated by native compiler successfully linked

WINDOWS COFF JITLINK EXAMPLE

LLVM IR executor

Loading static library built by MSVC into JIT session

```
1     auto G = ExitOnErr(StaticLibraryDefinitionGenerator::Load(
2       J->getObjLinkingLayer(), "StaticLib1.lib")
3     );
4     J->getMainJITDylib().addGenerator(std::move(G));
```

Notice

- object files generated by native compiler successfully linked
- native static initializers inside static library worked out of shelf

WINDOWS COFF JITLINK PLUGIN EXAMPLE

Background

Overview of JITLink

WINDOWS COFF JITLINK PLUGIN EXAMPLE

Background

Overview of JITLink

- Different formats of object files: ELF, MachO, COFF

WINDOWS COFF JITLINK PLUGIN EXAMPLE

Background

Overview of JITLink

- Different formats of object files: ELF, MachO, COFF
- Different architecture of binary code: x86_64, aarch64, risc-v, ppc

WINDOWS COFF JITLINK PLUGIN EXAMPLE

Background

Overview of JITLink

- Different formats of object files: ELF, MachO, COFF
- Different architecture of binary code: x86_64, aarch64, risc-v, ppc
- JITLink **converts** object file into generic linker object representation **LinkGraph**
  - ELFLinkGraphBuilder, COFFLinkGraphBuilder, MachOLinkGraphBuilder

WINDOWS COFF JITLINK PLUGIN EXAMPLE

Background

Overview of JITLink

- Different formats of object files: ELF, MachO, COFF
- Different architecture of binary code: x86_64, aarch64, risc-v, ppc
- JITLink **converts** object file into generic linker object representation **LinkGraph**
  - ELFLinkGraphBuilder, COFFLinkGraphBuilder, MachOLinkGraphBuilder
- Then, it performs generic **memory allocation, symbol resolution** as described in **LinkGraph** and perform architecture-specific **relocations** as needed

WINDOWS COFF JITLINK PLUGIN EXAMPLE

Overview of LinkGraph

Block (Code)

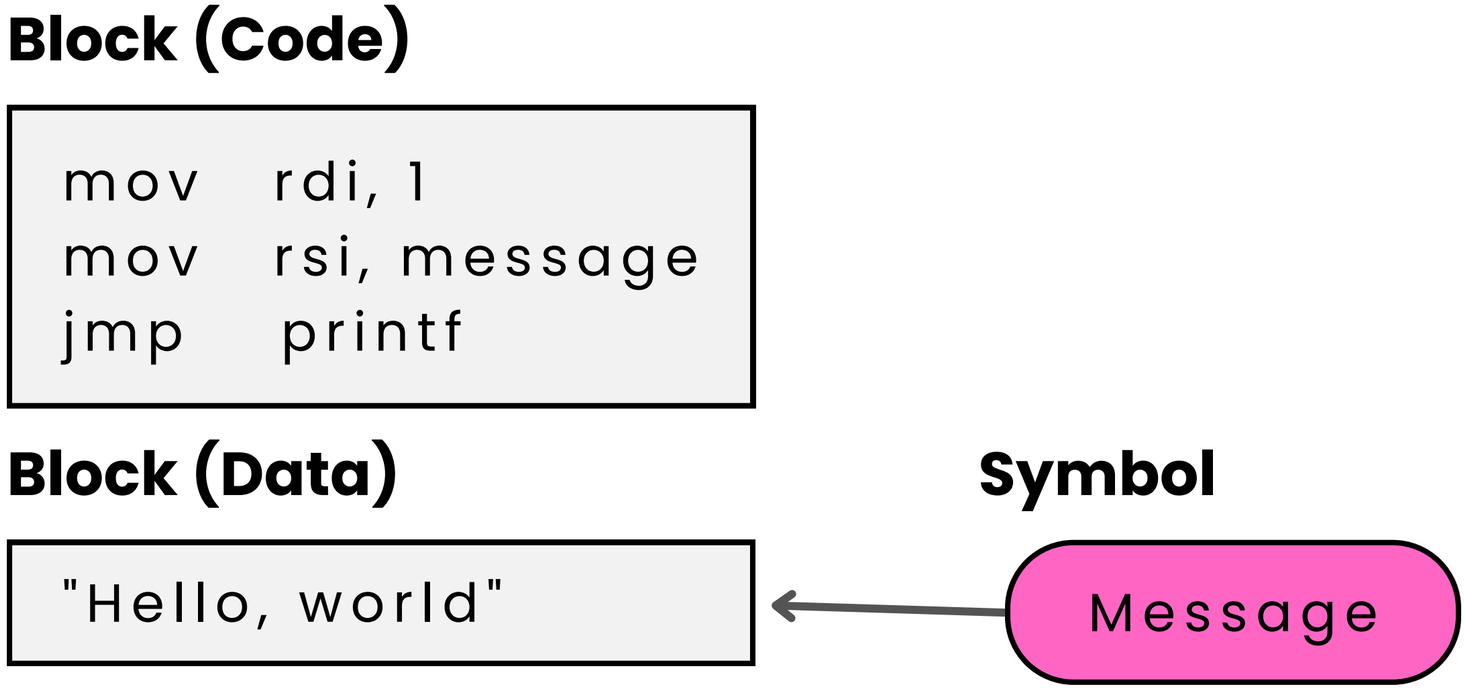
```
mov   rdi, 1
mov   rsi, message
jmp   printf
```

Block (Data)

```
"Hello, world"
```

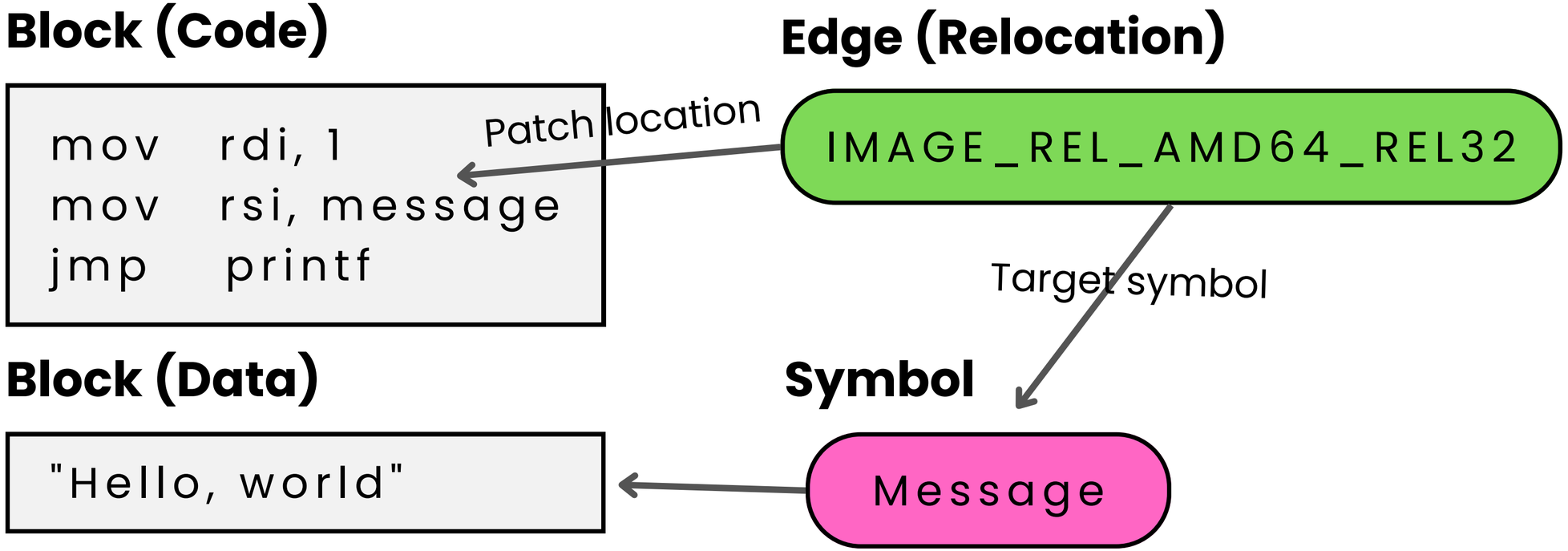
WINDOWS COFF JITLINK PLUGIN EXAMPLE

Overview of LinkGraph



WINDOWS COFF JITLINK PLUGIN EXAMPLE

Overview of LinkGraph



WINDOWS COFF JITLINK PLUGIN EXAMPLE

Basic plugin

```
1   class ExamplePlugin : public ObjectLinkingLayer::Plugin {
2   public:
3     void modifyPassConfig(MaterializationResponsibility &MR,
4                           jitlink::LinkGraph &G,
5                           jitlink::PassConfiguration &Config) override {
6       Config.PrePrunePasses.push_back([&](jitlink::LinkGraph &G) {
7         G.dump(llvm::outs());
8         return Error::success();
9       });
10    }
```

WINDOWS COFF JITLINK PLUGIN EXAMPLE

Basic plugin

```
1   class ExamplePlugin : public ObjectLinkingLayer::Plugin {
2   public:
3     void modifyPassConfig(MaterializationResponsibility &MR,
4                           jitlink::LinkGraph &G,
5                           jitlink::PassConfiguration &Config) override {
6       Config.PrePrunePasses.push_back([&](jitlink::LinkGraph &G) {
7         G.dump(llvm::outs());
8         return Error::success();
9       });
10    }
```

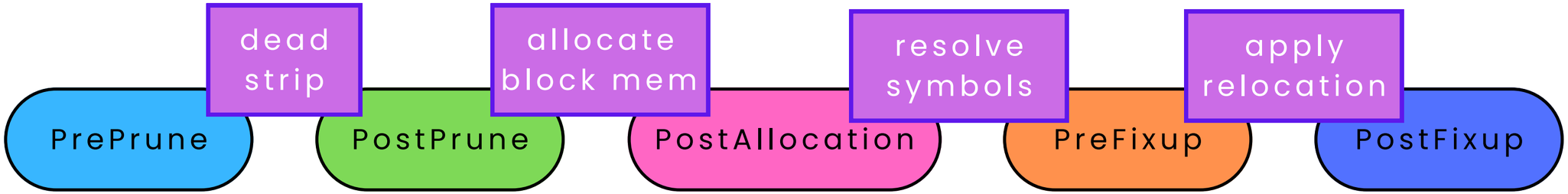
Callback gets called in specific linking phase

WINDOWS COFF JITLINK PLUGIN EXAMPLE

Basic plugin

```
1   class ExamplePlugin : public ObjectLinkingLayer::Plugin {
2   public:
3     void modifyPassConfig(MaterializationResponsibility &MR,
4                           jitlink::LinkGraph &G,
5                           jitlink::PassConfiguration &Config) override {
6       Config.PrePrunePasses.push_back([&](jitlink::LinkGraph &G) {
7         G.dump(llvm::outs());
8         return Error::success();
9       });
10    }
```

Callback gets called in specific linking phase



WINDOWS COFF JITLINK PLUGIN EXAMPLE

Unwind frame visualizer plugin

Unwind frame visualizer plugin

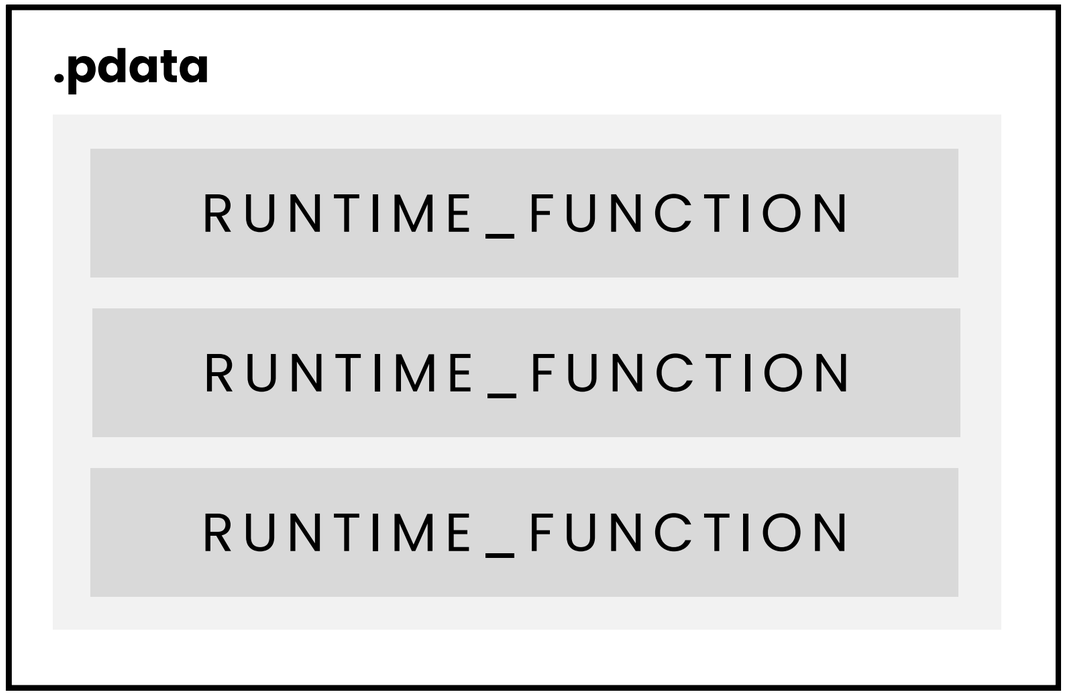
- Print the summary of unwind frame of each function contained inside object file

WINDOWS COFF JITLINK PLUGIN EXAMPLE

Unwind frame visualizer plugin

Windows unwind frame

example.obj

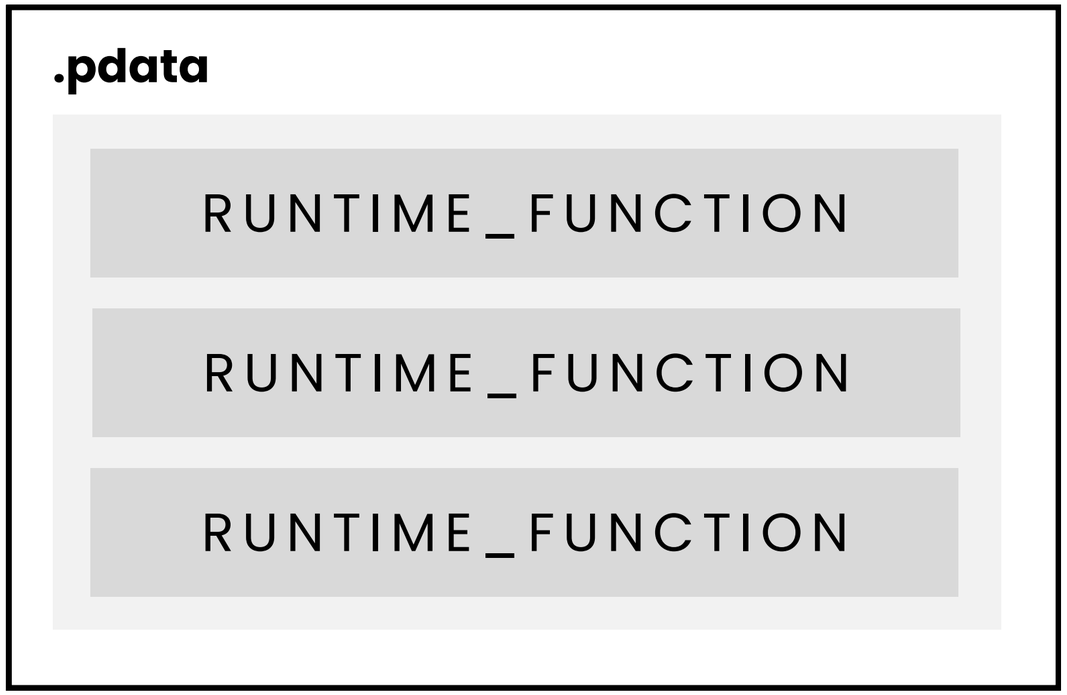


WINDOWS COFF JITLINK PLUGIN EXAMPLE

Unwind frame visualizer plugin

Windows unwind frame

example.obj



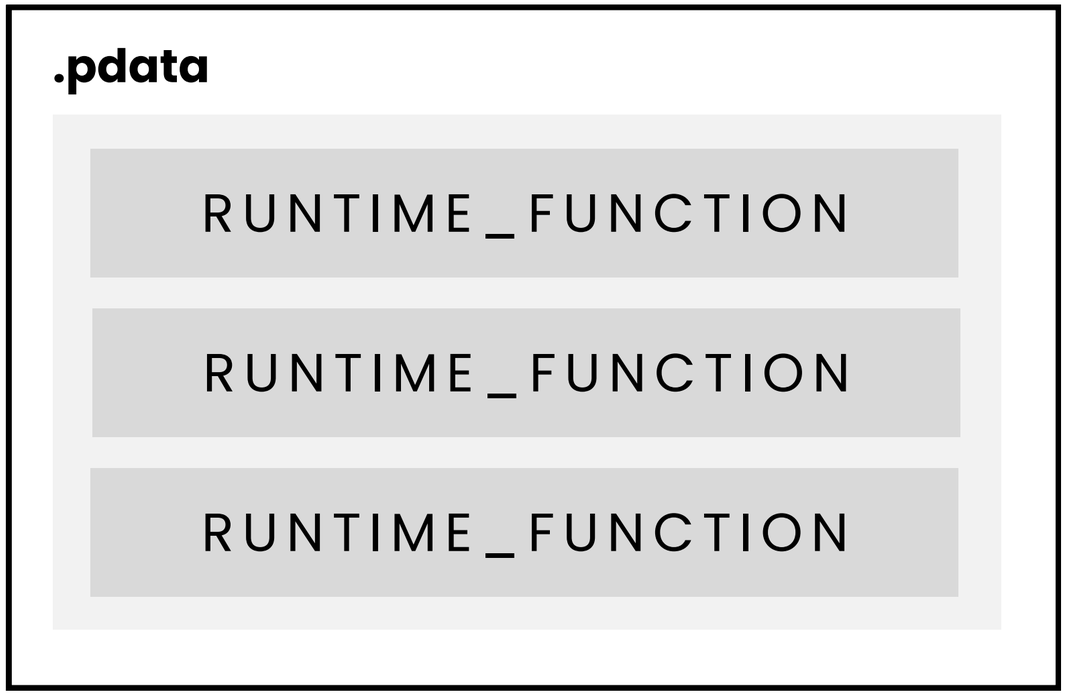
- Each RUNTIME_FUNCTION has code range of function and unwind info

WINDOWS COFF JITLINK PLUGIN EXAMPLE

Unwind frame visualizer plugin

Windows unwind frame

example.obj



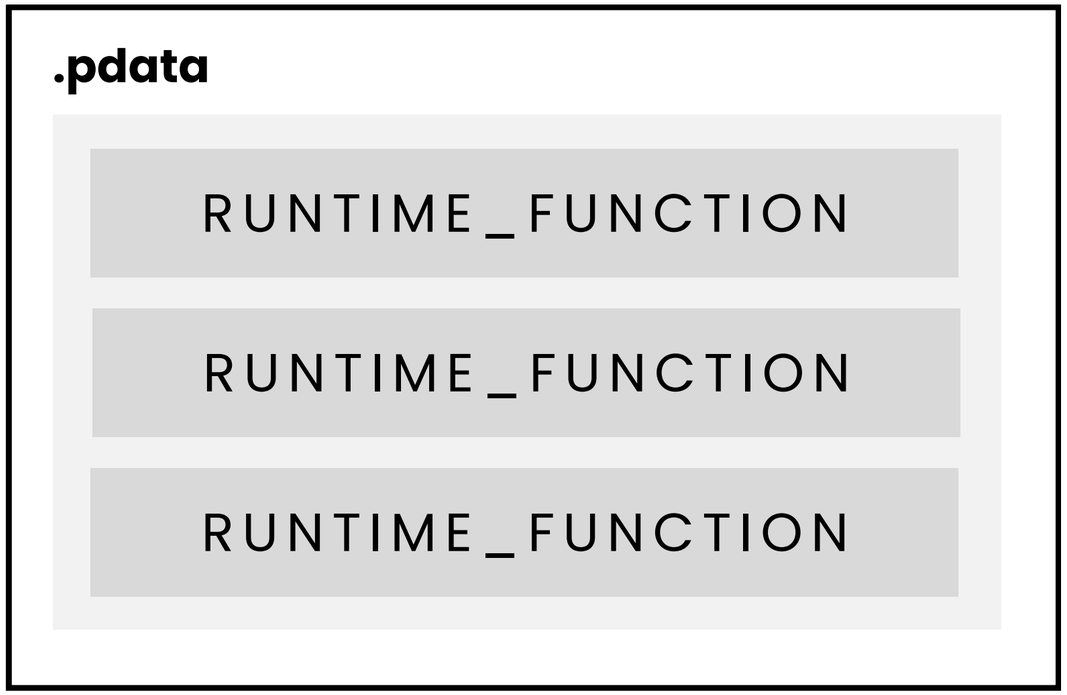
- Each RUNTIME_FUNCTION has code range of function and unwind info
- RUNTIME_FUNCTIONs emitted to .pdata section

WINDOWS COFF JITLINK PLUGIN EXAMPLE

Unwind frame visualizer plugin

Windows unwind frame

example.obj

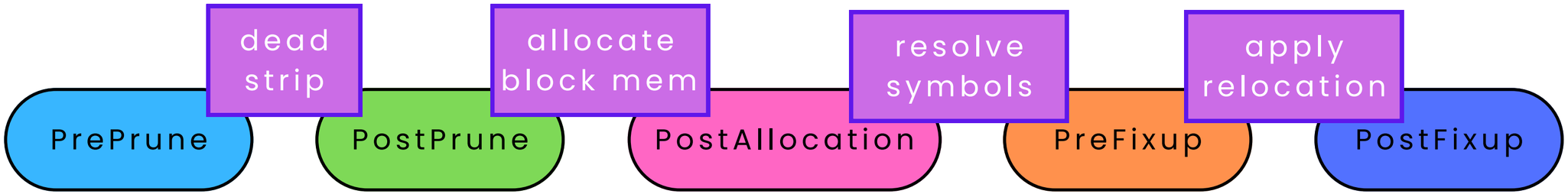


- Each RUNTIME_FUNCTION has code range of function and unwind info
- RUNTIME_FUNCTIONs emitted to .pdata section
- JITLink plugin system allows access to this .pdata section

WINDOWS COFF JITLINK PLUGIN EXAMPLE

Unwind frame visualizer plugin

Coding time



WINDOWS COFF JITLINK PLUGIN EXAMPLE

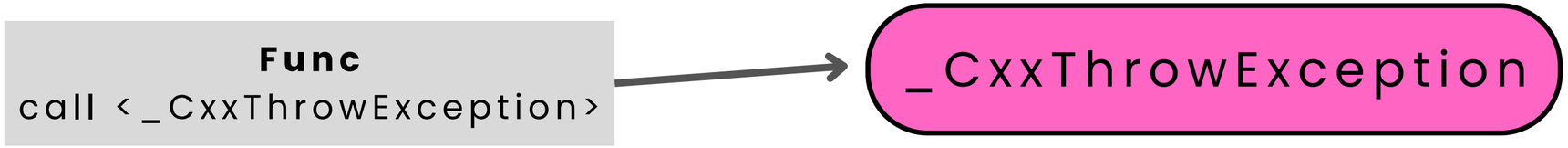
Exception instrumentation plugin

Exception instrumentation plugin

- Show disassembly of function that just raised exception

WINDOWS COFF JITLINK PLUGIN EXAMPLE

Exception instrumentation plugin

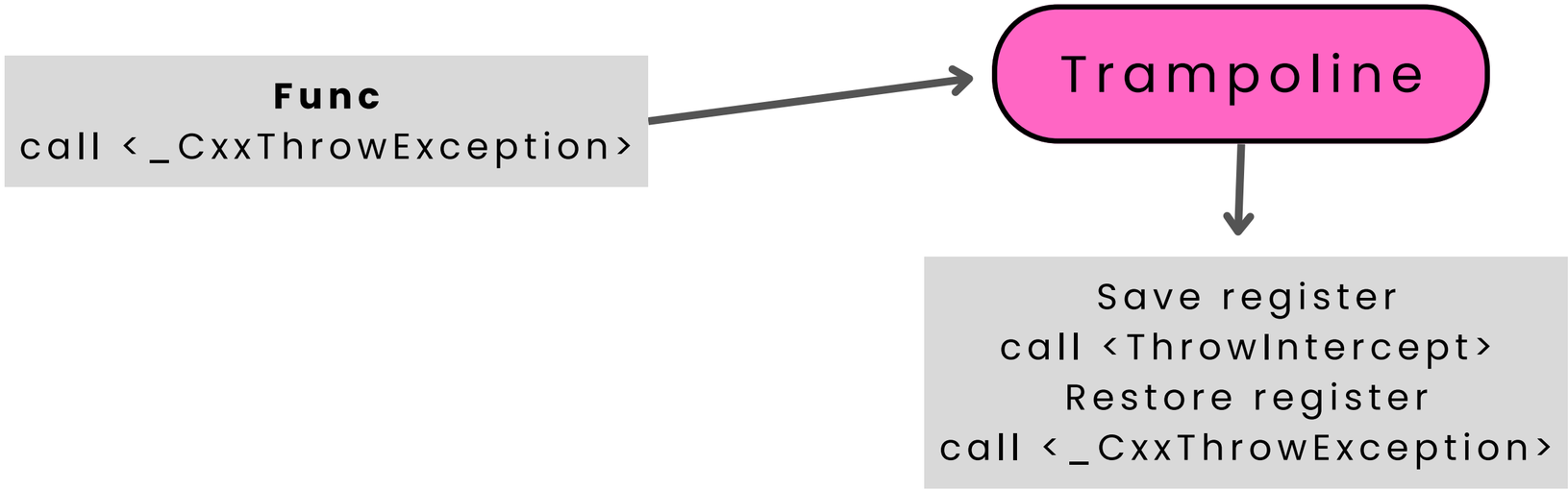


WINDOWS COFF JITLINK PLUGIN EXAMPLE

Exception instrumentation plugin

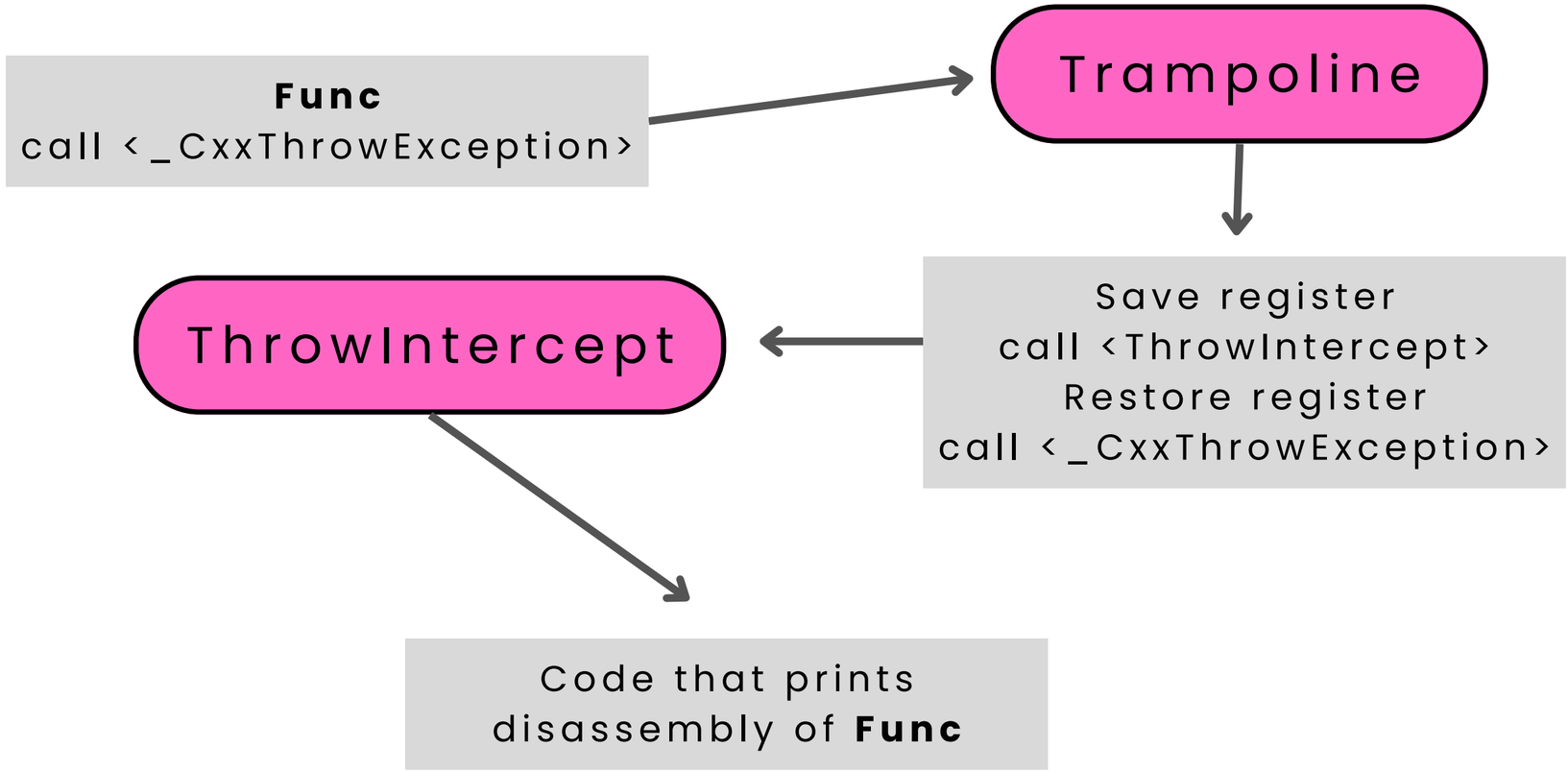
WINDOWS COFF JITLINK PLUGIN EXAMPLE

Exception instrumentation plugin



WINDOWS COFF JITLINK PLUGIN EXAMPLE

Exception instrumentation plugin



WINDOWS COFF JITLINK PLUGIN EXAMPLE

Exception instrumentation plugin

```
1   std::vector<jitlink::Edge> Edges;
2   std::vector<char> CodeBuf;
3
4   // Write x86 assembly code to CodeBuf
5   WriteSaveRegsCode(CodeBuf);
6   WriteCallFuncCode(CodeBuf);
7
```

WINDOWS COFF JITLINK PLUGIN EXAMPLE

Exception instrumentation plugin

```
1  std::vector<jitlink::Edge> Edges;
2  std::vector<char> CodeBuf;
3
4  // Write x86 assembly code to CodeBuf
5  WriteSaveRegsCode(CodeBuf);
6  WriteCallFuncCode(CodeBuf);
7
```

CodeBuf (content bytes of block)

```
1  0: pushq   %rbp
2  1: movq    %rsp, %rbp
3  4: subq    $512, %rsp
4  b: movq    %rcx, -16(%rbp)
5  f: movq    %rdx, -24(%rbp)
6  13: movq   %rsi, -32(%rbp)
7  17: movq   %rdi, -40(%rbp)
8  ...
9  6f: e8 00 00 00 00 callq <ThrowIntercept>
```

WINDOWS COFF JITLINK PLUGIN EXAMPLE

Exception instrumentation plugin

```
1  std::vector<jitlink::Edge> Edges;
2  std::vector<char> CodeBuf;
3
4  // Write x86 assembly code to CodeBuf
5  WriteSaveRegsCode(CodeBuf);
6  WriteCallFuncCode(CodeBuf);
7
```

CodeBuf (content bytes of block)

```
1  0: pushq   %rbp
2  1: movq    %rsp, %rbp
3  4: subq    $512, %rsp
4  b: movq    %rcx, -16(%rbp)
5  f: movq    %rdx, -24(%rbp)
6  13: movq   %rsi, -32(%rbp)
7  17: movq   %rdi, -40(%rbp)
8  ...
9  6f: e8 00 00 00 00 callq <ThrowIntercept>
```

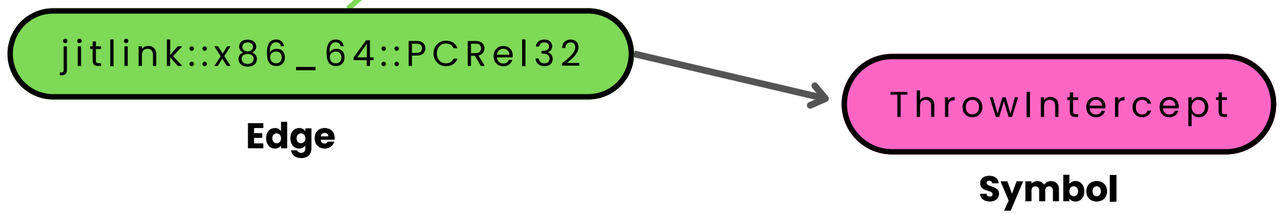
WINDOWS COFF JITLINK PLUGIN EXAMPLE

Exception instrumentation plugin

```
1   std::vector<jitlink::Edge> Edges;
2   std::vector<char> CodeBuf;
3
4   // Write x86 assembly code to CodeBuf
5   WriteSaveRegsCode(CodeBuf);
6   WriteCallFuncCode(CodeBuf);
7
8   // Add relocation edge to ThrowIntercept
9   auto ThrowInterceptSymbol =
10      &G.addExternalSymbol("ThrowIntercept", 0, jitlink::Linkage::Strong);
11  Edges.push_back(jitlink::Edge(jitlink::x86_64::PCRel32, CodeBuf.size() - 4,
12                                *ThrowInterceptSymbol, 0));
```

CodeBuf (content bytes of block)

```
1   0: pushq   %rbp
2   1: movq    %rsp, %rbp
3   4: subq    $512, %rsp
4   b: movq    %rcx, -16(%rbp)
5   f: movq    %rdx, -24(%rbp)
6   13: movq   %rsi, -32(%rbp)
7   17: movq   %rdi, -40(%rbp)
8   ...
9   6f: e8 00 00 00 00 callq <ThrowIntercept>
```



WINDOWS COFF JITLINK PLUGIN EXAMPLE

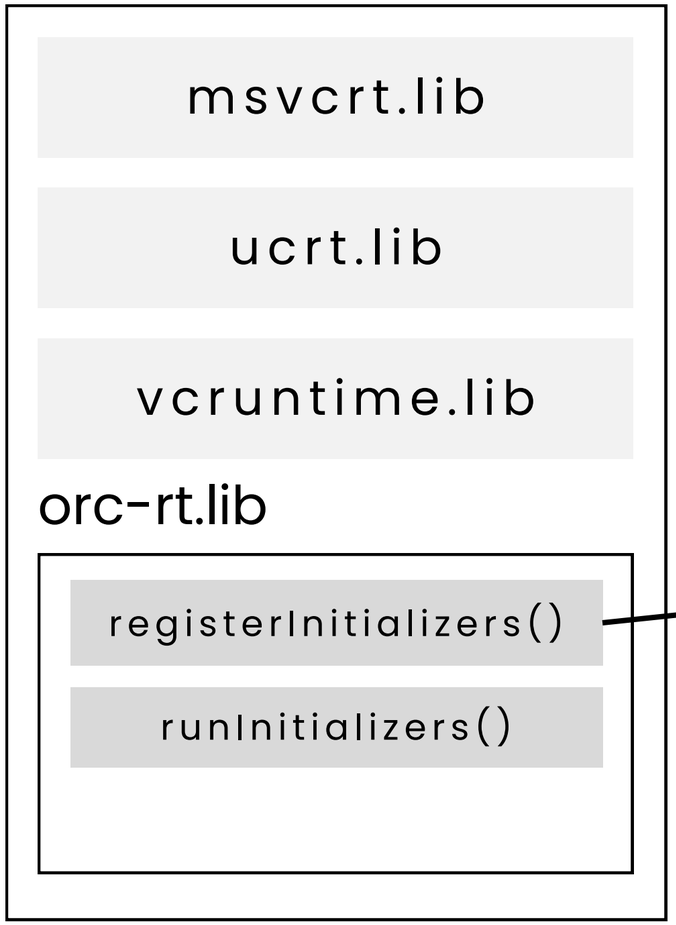
Exception instrumentation plugin

Coding time

TIPS ON USING JITLINK IN COFF

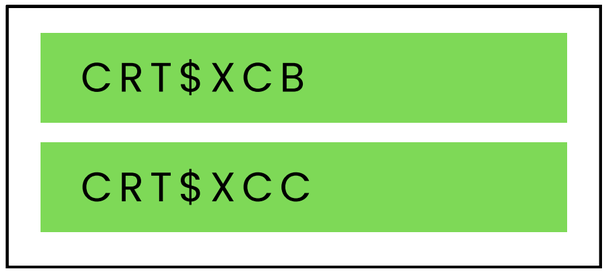
ORC Runtime at startup

MainJD



- COFFPlatform loads up vcruntime lib files and ORC runtime lib file.
- Uses JIT-linked orc runtime function to register and run static initializers
- ORC runtime itself uses JIT-linked STL library

New object file



TIPS ON USING JITLINK IN COFF

ORC Runtime at startup

Tips

- Care is needed to make sure ORC and vc runtime library files are available
  - by default, vc runtime libraries automatically detected from VC toolchain directories (can fail)
- Customizing vc runtime loading can be done by COFFVCRuntimeBootstrapper class
- It is still possible to use in-process vc runtime symbols, but need to export required symbols manually by using linker directive
  - #pragma comment(linker, "/export:??_7type_info@@6B@")

TIPS ON USING JITLINK IN COFF

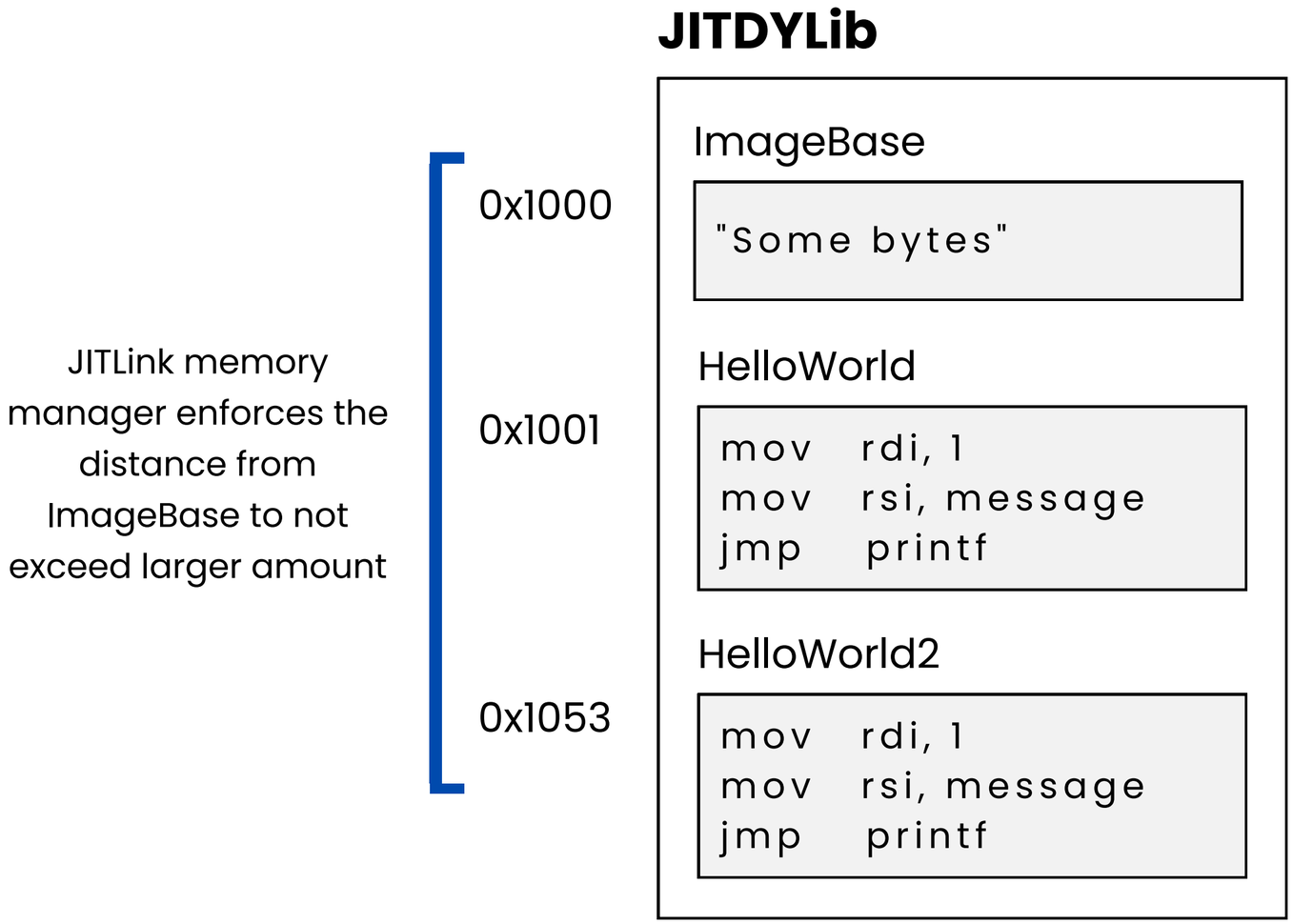
JITDYLIB: Emulated DYLIB inside JIT session

Challenges with COFF small code model

- Compilers assume that all symbols within the same executable or dylib are allocated close together
- It is not possible to "patch" instructions to use GOT pointer on demand when the required displacement exceeds 2Gb
- COFF x86 relocation points to the middle of instruction bytes
  - x86 encoding is not possible to be read backwards to know the start of instruction (for instructions of interest because of presence of RAX prefix)
  - -> can't patch this part

TIPS ON USING JITLINK IN COFF

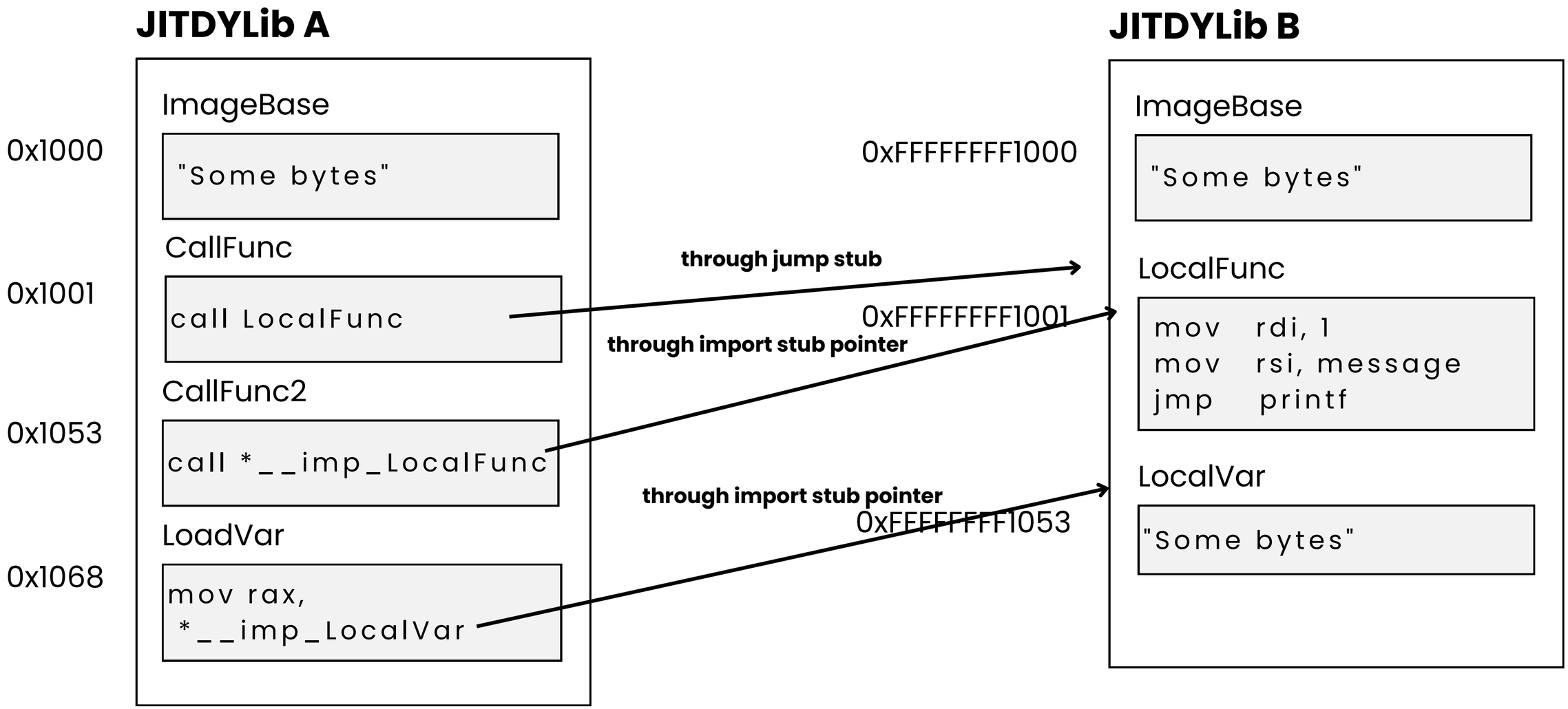
JITDYLIB: Emulated DYLIB inside JIT session



- Emulated dylib inside JIT session
- dlopen and dlclose JITDYLib inside JITted code

TIPS ON USING JITLINK IN COFF

JITDYLIB: Emulated DYLIB inside JIT session



TIPS ON USING JITLINK IN COFF

JITDYLIB: Emulated DYLIB inside JIT session

Tips

- Call function of another JITDYLib through usual call or dllimport attribute (__imp_)
- Access data of another JITDYLib only through dllimport attirbute (__imp_)
- Same practices are required in AOT world too but less clear in JIT world

CLANG-REPL DEMO

- Everything is in-tree in LLVM including clang-repl executable

THANKS