Behind the Scenes of the Projucer C++ Live-Build Engine

ADC'16 London, November 2016

Stefan Gränitz, Consultant Software Development JUCE

Outline

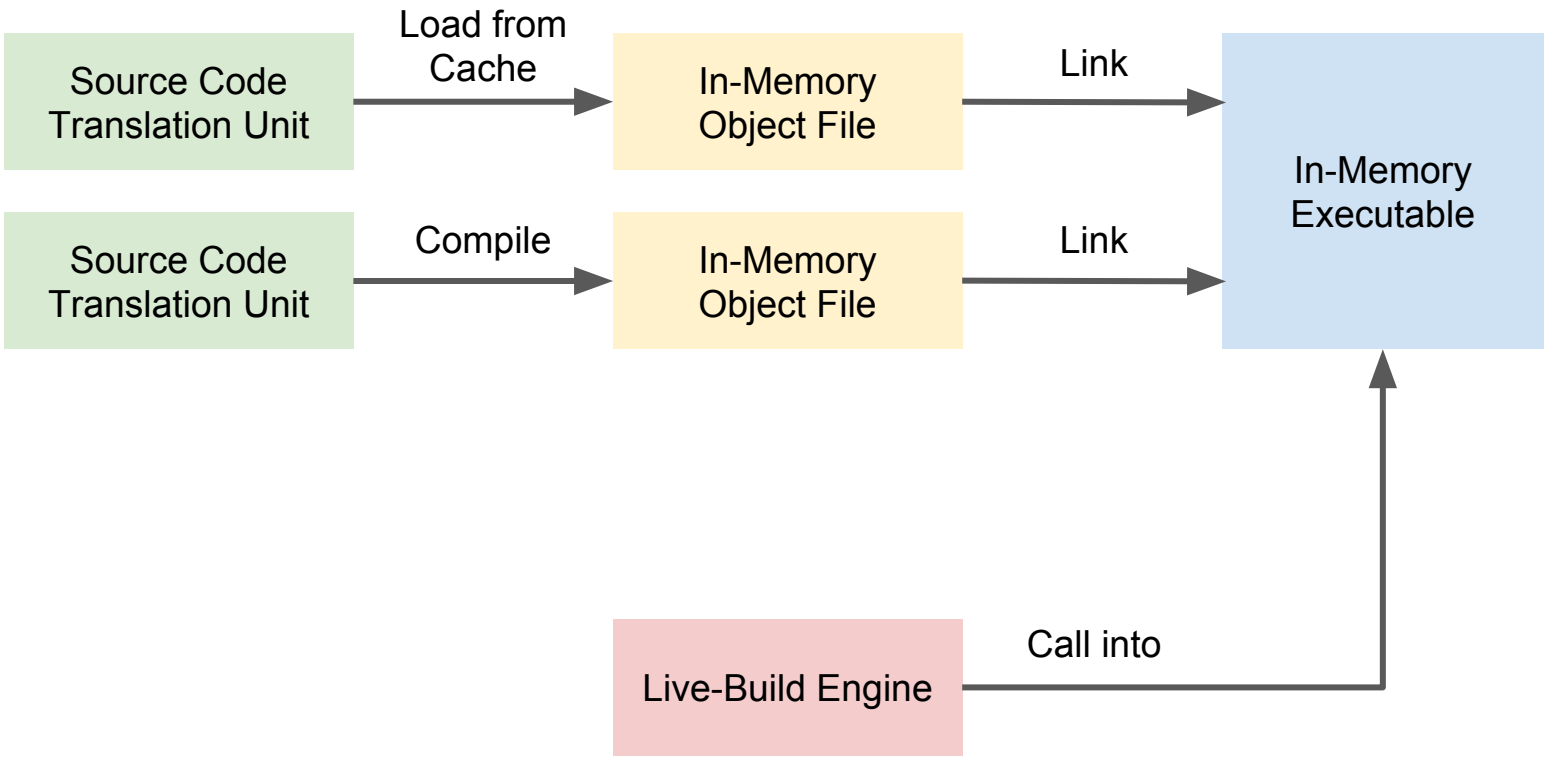
Part 1: Conceptual

- Demo
- How it works
- C++ for Live Coding?

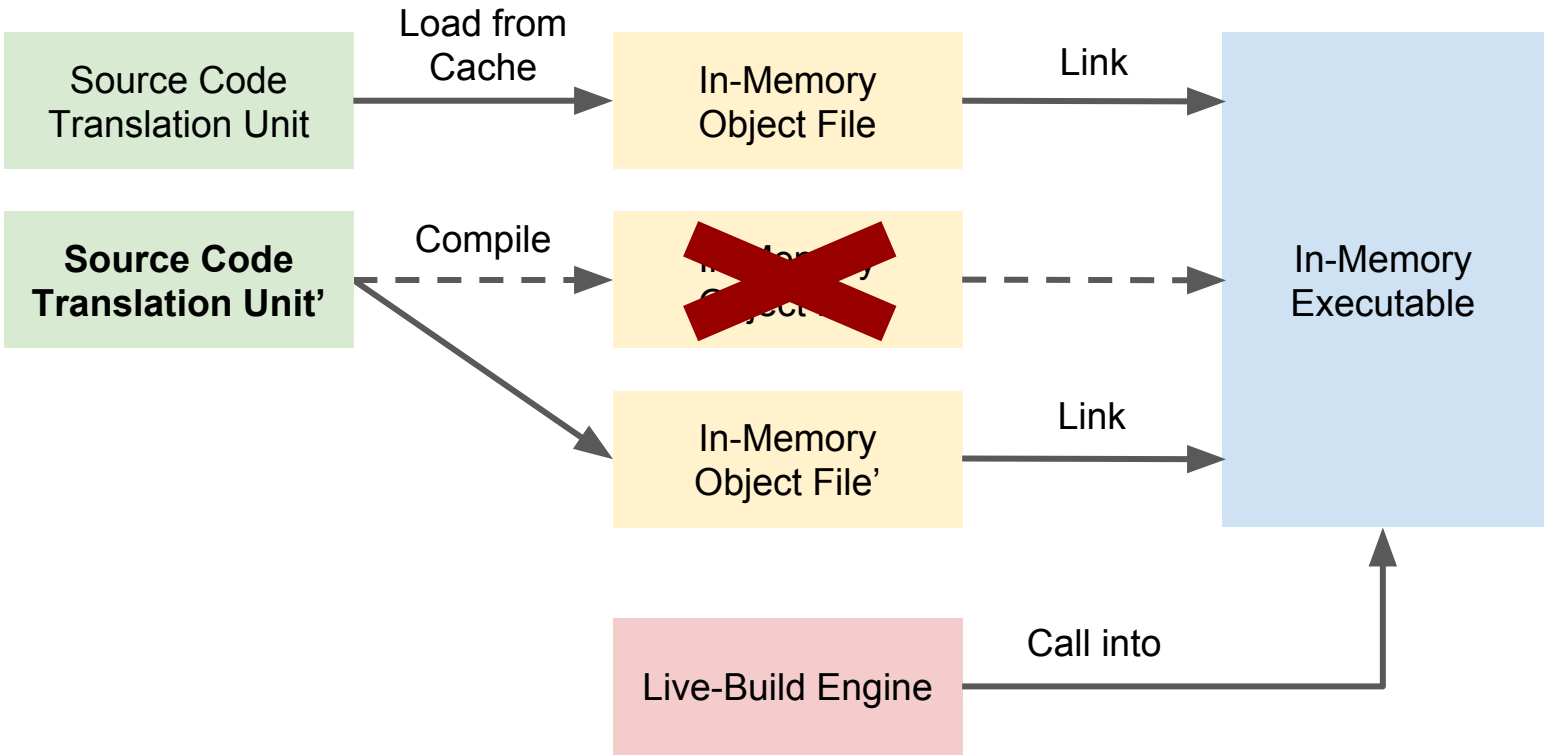
Part 2: Technical

- Linking: Static vs. Live

Live builds – High-level overview



Live builds – High-level overview

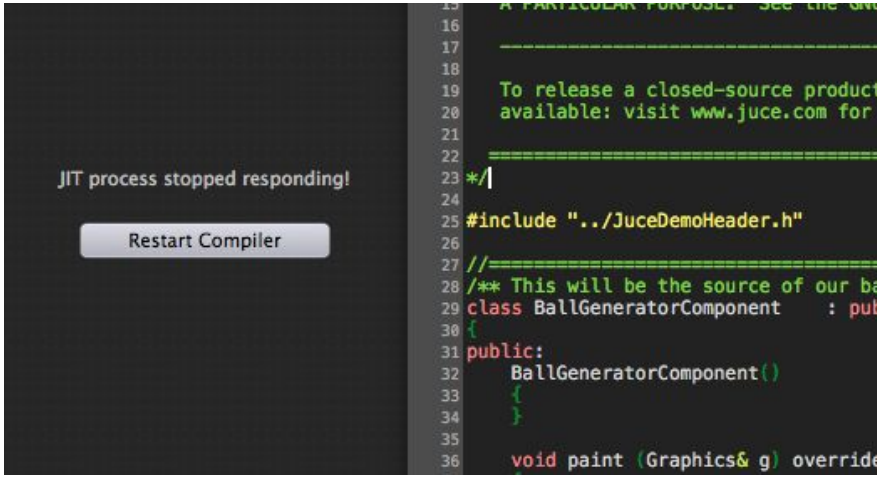


C++ vs. a Language Constructed for Live Coding

Ideal guaranteed crash-freeness → declarative or functional

C++ imperative → inherently **not** crash-safe

Projucer: live code runs in a separate process



C++ vs. a Language Constructed for Live Coding

Ideal unified runtime environment → avoid arbitrary external dependencies

C++ "There should be no language beneath C++"
→ handle all quirks of supported platforms manually

- Object File formats: PE/COFF on Windows, Mach Object on MacOSX x64, ELF on Linux
- No standardized ABI: name mangling

```
void foo(int bar)
```

→ unix-like style: `__Z3fooi`

→ msvc style: `?foo@@YAPEAHH@Z`

C++ vs. a Language Constructed for Live Coding

Ideal unified runtime environment → avoid arbitrary external dependencies

C++ "There should be no language beneath C++"
→ handle all quirks of supported platforms manually

- Object File formats: PE/COFF on Windows, Mach Object on MacOSX x64, ELF on Linux
- No standardized ABI: name mangling

```
// We should never ever see a FunctionNoProtoType at this point.
// We don't even know how to mangle their types anyway :).
```

MicrosoftMangle.cpp

C++ vs. a Language Constructed for Live Coding

Ideal unified runtime environment → avoid arbitrary external dependencies

C++ "There should be no language beneath C++"
→ handle all quirks of supported platforms manually

- Object File formats: PE/COFF on Windows, Mach Object on MacOSX x64, ELF on Linux
- No standardized ABI: name mangling, exception handling (Unix signals, Windows SEH/VEH/C++ exceptions), calling conventions (Windows `dllimport`), RTTI
- Intrinsics for extended instruction sets (SSE, HLSL, etc.)
  Projucer: ships intrinsics headers for its specific version of Clang

C++ vs. a Language Constructed for Live Coding

Ideal optimize for simplicity, uniformity, portability

C++ optimize for efficiency: “Only pay for what you use”

→ standard encourages compiler vendors to implement system-specific optimizations for performance reasons

C++ Standard Library:

- standardized syntax & semantics
- implementations and **headers** vary between compilers & platforms

Projucer: Xcode must be installed on Mac

Visual Studio 2015 Update 3 on Windows

C++ vs. a Language Constructed for Live Coding

Ideal strong static typing → allow runtime state restauration

C++ “No implicit violations of the type system, but allow explicit violations”

Projucer: no runtime state restauration

C++ vs. a Language Constructed for Live Coding

Ideal	will (most likely) never be adapted
C++	used a lot by real people today and for a long time to come

All these quirks make it hard!

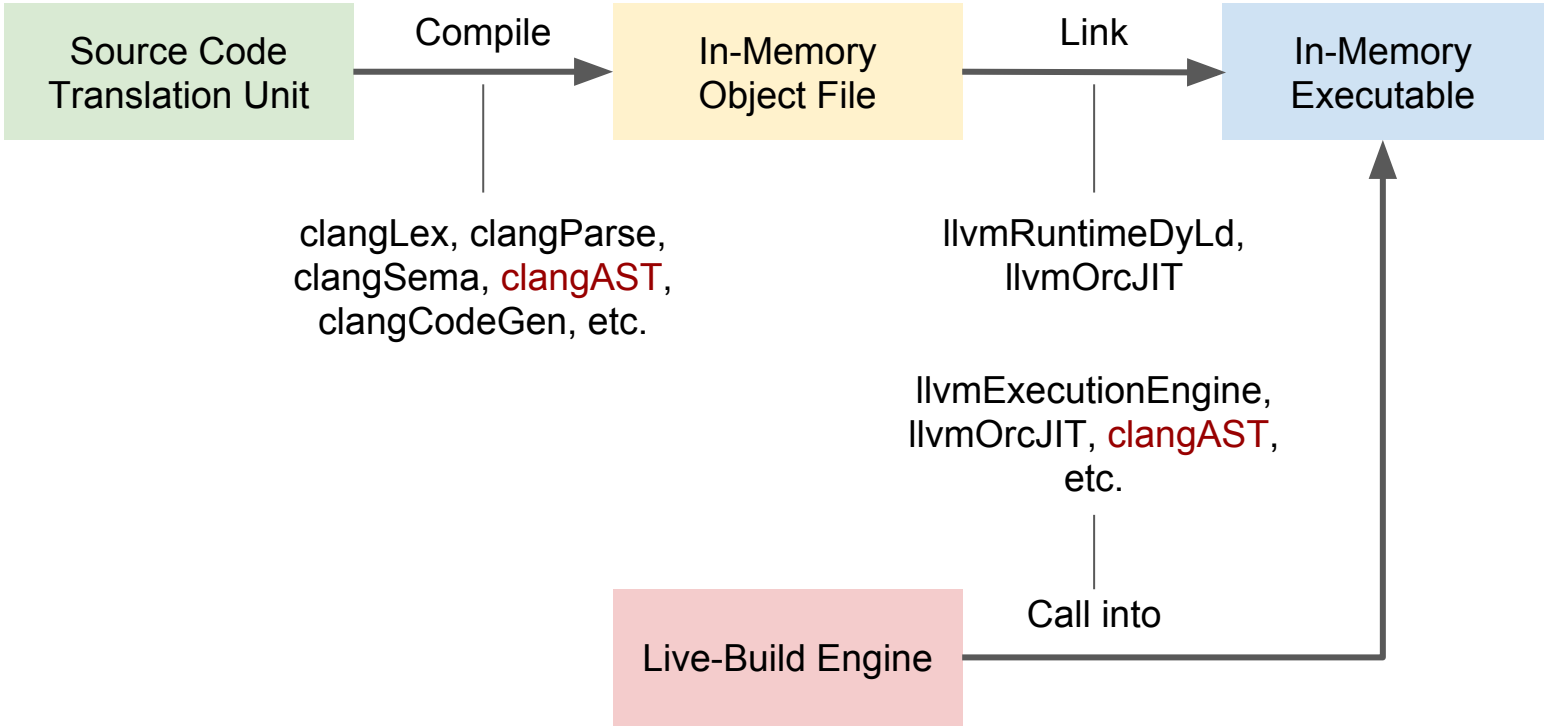
hard ≠ impossible

It needs some really good tools.

LLVM and Clang come to rescue!

- LLVM: compiler infrastructure
  - Intermediate Representation (LLVM IR)
  - set of tools and libraries to work with it
- Clang: C-language family frontend for LLVM

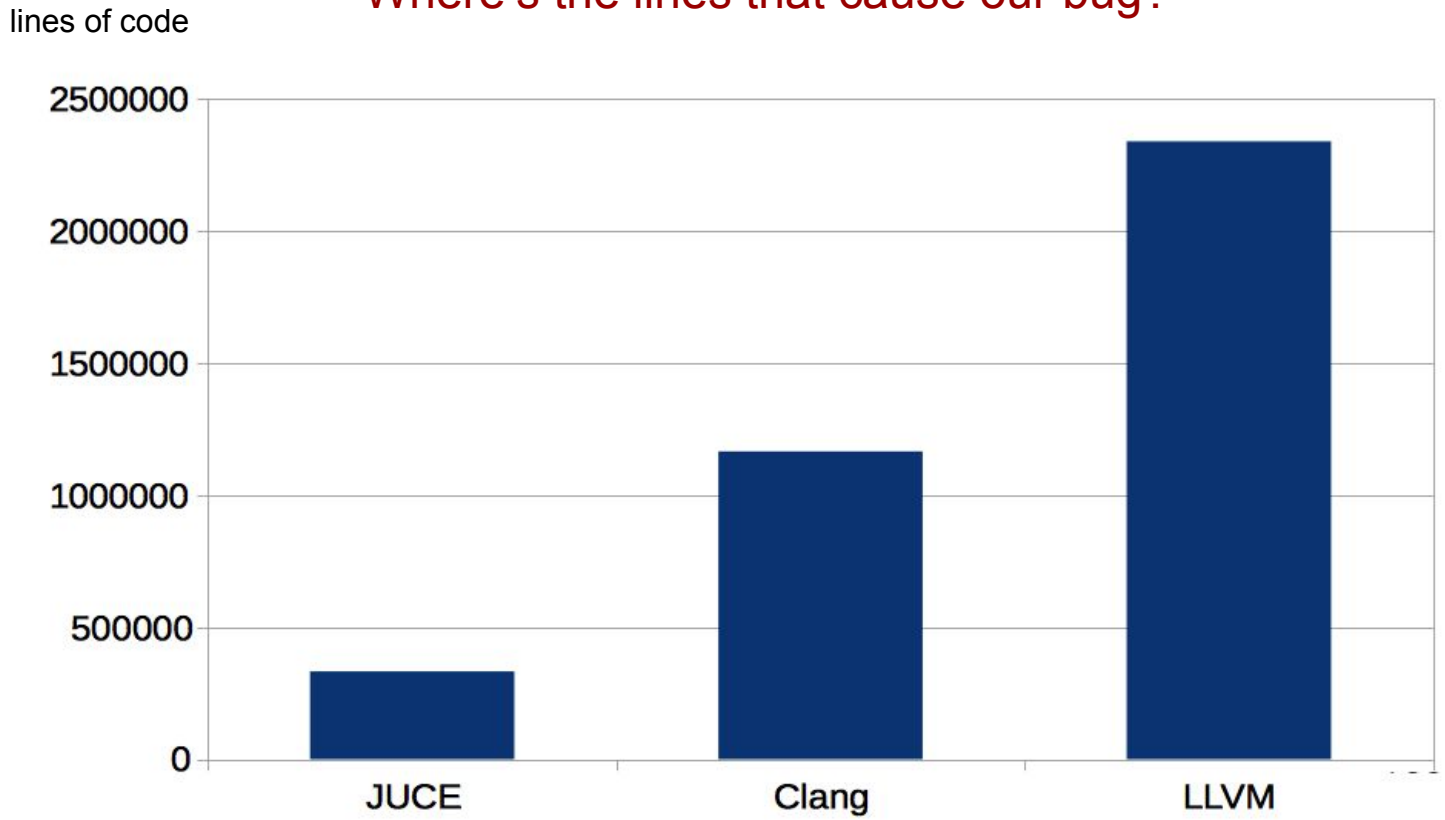
Live builds – High-level overview



Still: There be dragons

- LLVM & Clang are moving targets
- Upstream LLVM in numbers:
  - 30 commits per day on average
  - contributions from ~100 developers each month
  - Google, Intel, Microsoft, Qualcomm, AMD, …
- Maintenance cost for customizations is enormous
- 2015 DevMeeting: Living Downstream Without Drowning

Where's the lines that cause our bug?

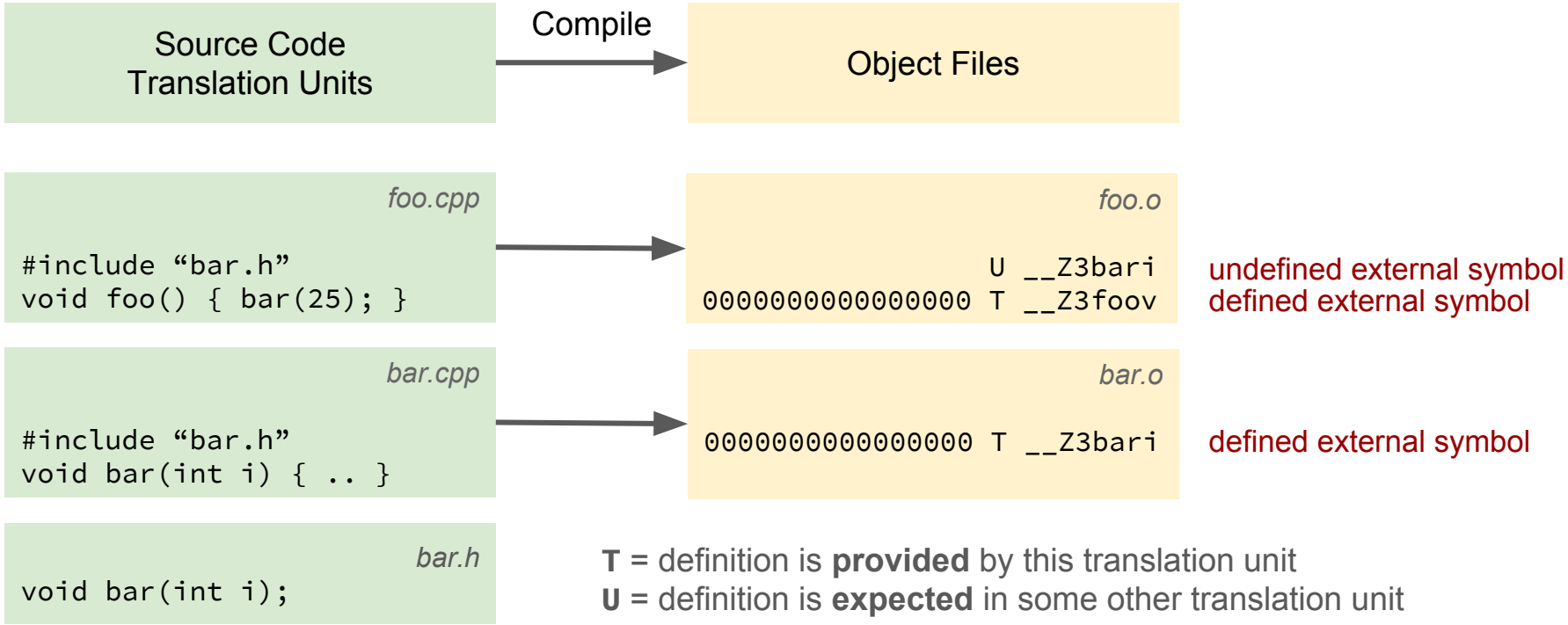


Blocker bug in the live build engine on Windows x64:

- Symptom: immediate crash when launching any live component preview

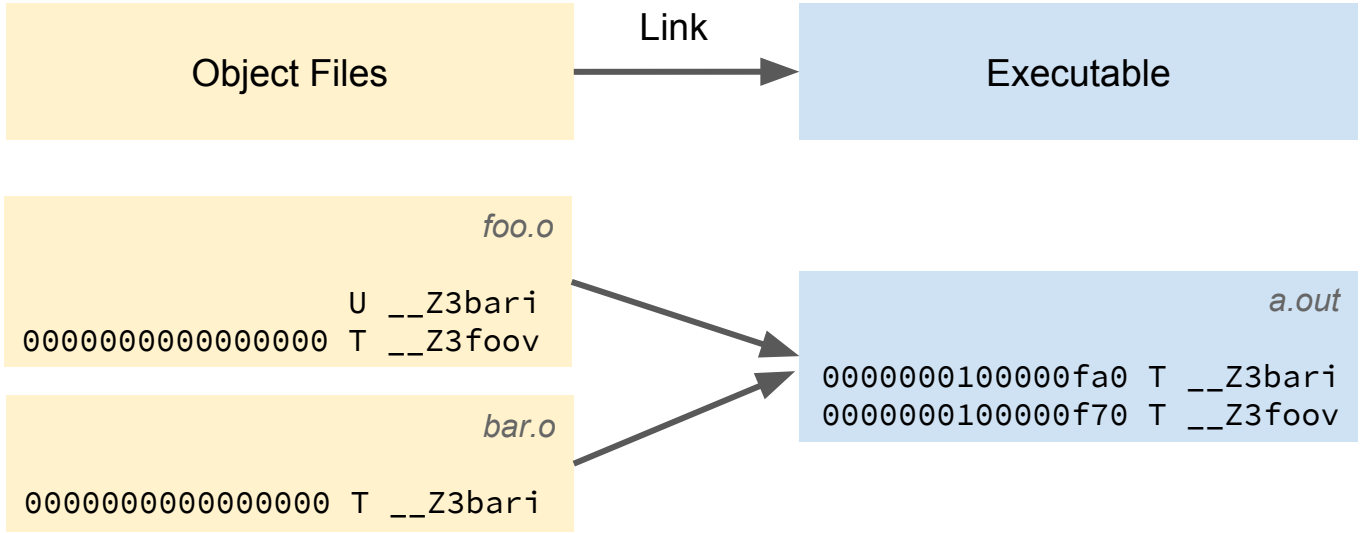
- ```
  Assertion failed while linking:
  ((int64_t)Result <= INT32_MAX) && "Relocation overflow"
  file: RuntimeDyldCOFFX86_64.h, line 81
  ```
  ```

A brief discourse on linking



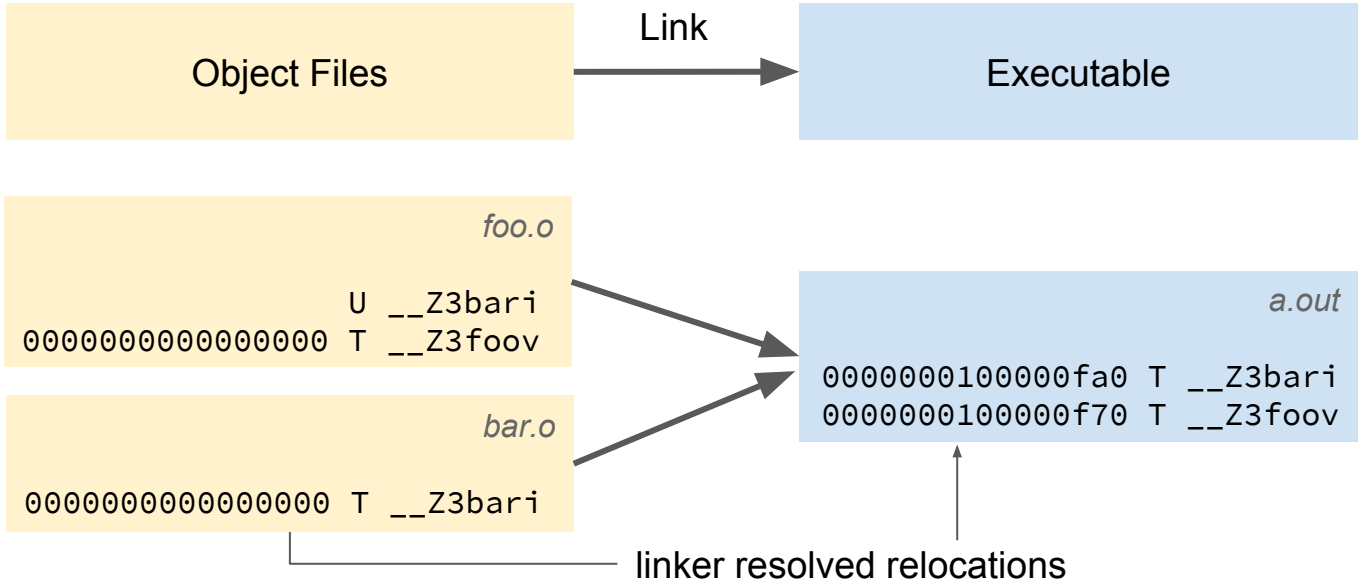
T = definition is **provided** by this translation unit
U = definition is **expected** in some other translation unit

A brief discourse on linking



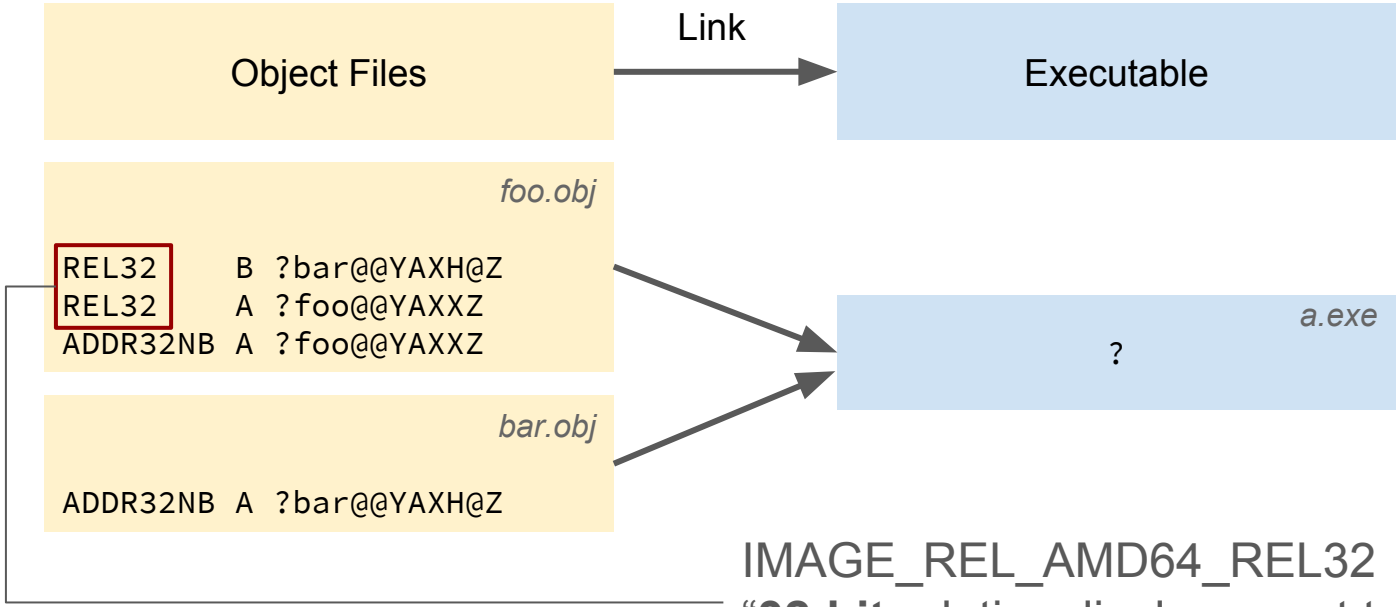
T = definition is **provided** by this translation unit
U = definition is **expected** in some other translation unit

A brief discourse on linking



T = definition is **provided** by this translation unit
U = definition is **expected** in some other translation unit

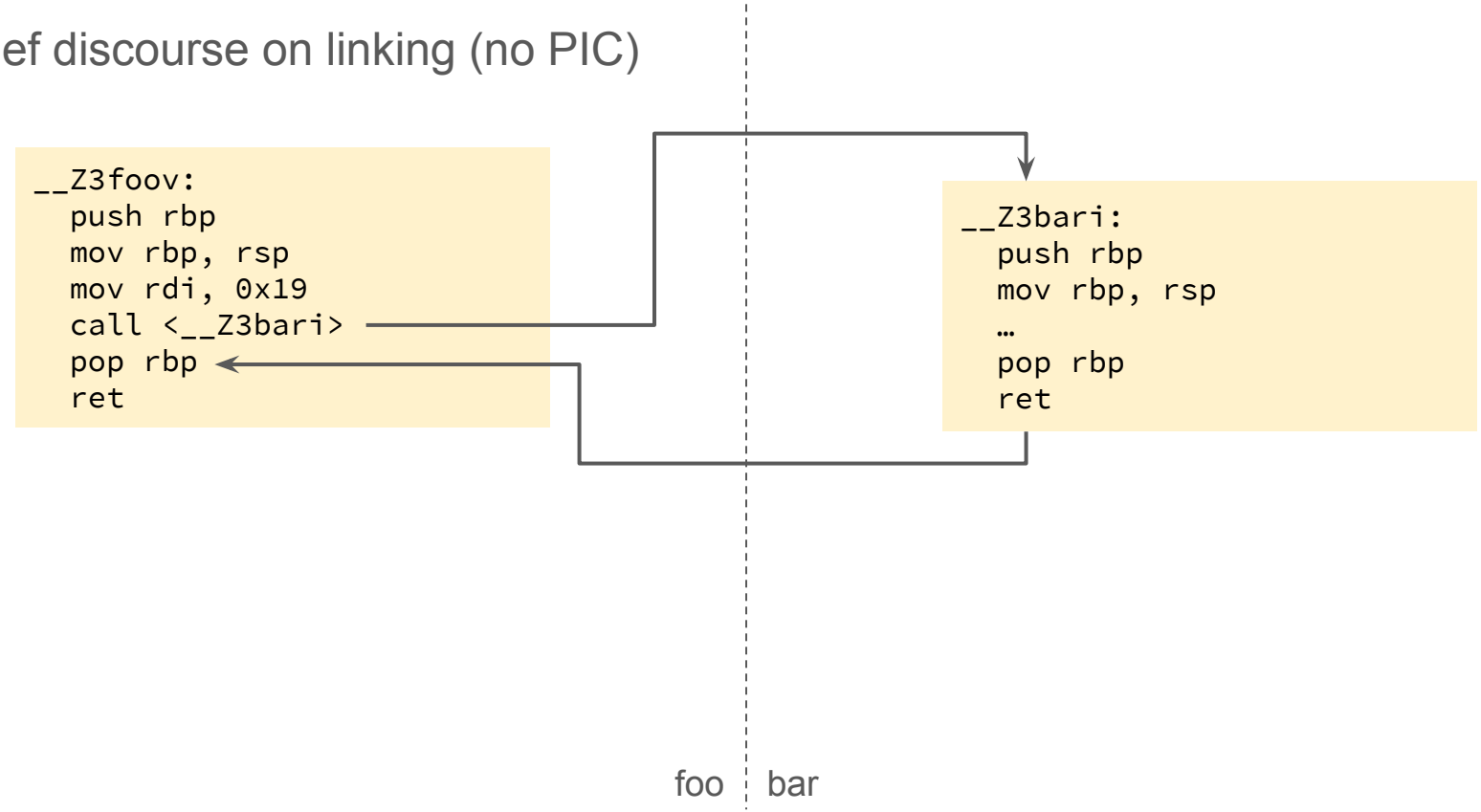
A brief discourse on linking – Clang COFF Objects Windows x64



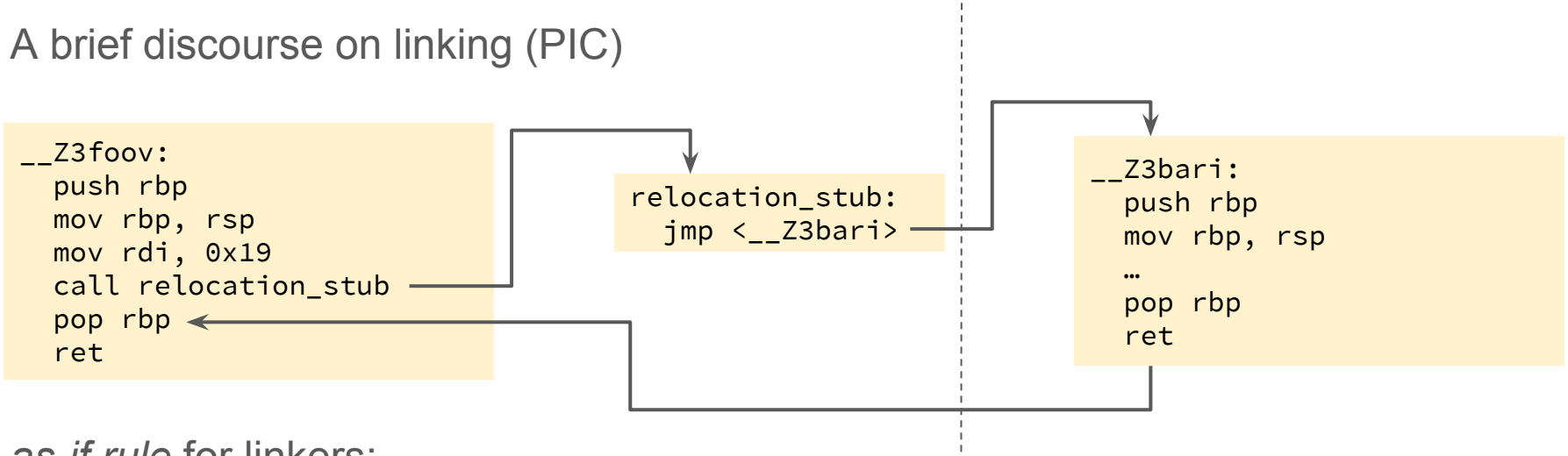
IMAGE_REL_AMD64_REL32
"**32-bit** relative displacement to the target"

Microsoft PE/COFF Specification

A brief discourse on linking (no PIC)



A brief discourse on linking (PIC)

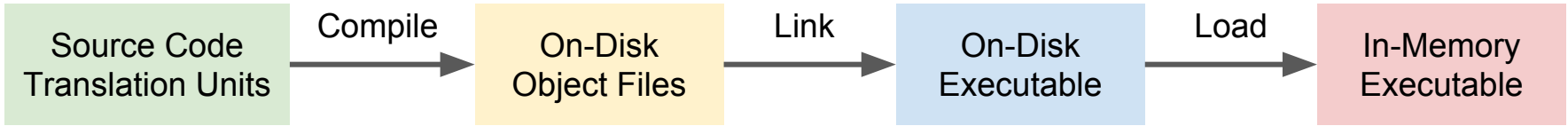


as-if rule for linkers:

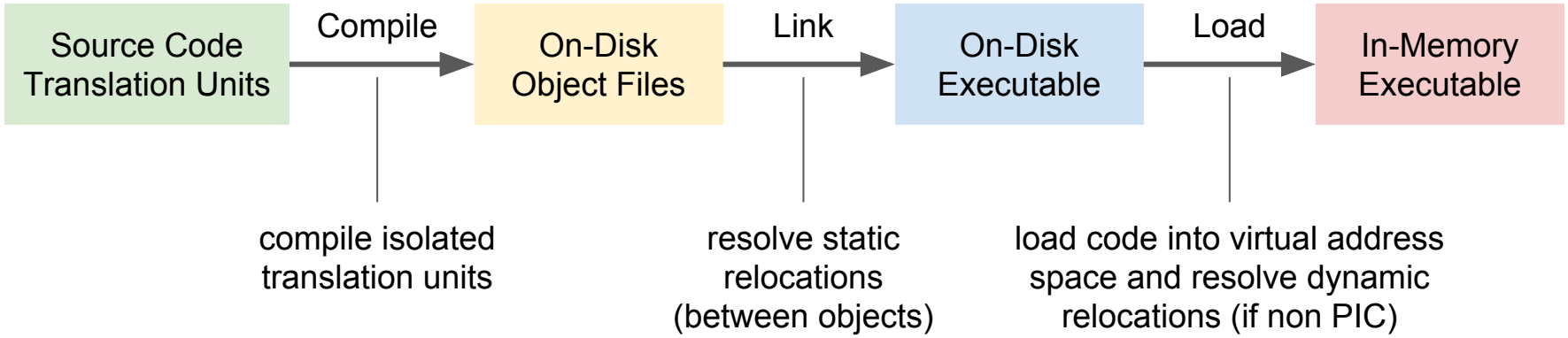
"The linker is responsible for creating [...] stub functions and lazy pointers [...] for calls to another linkage unit. Since **the linker** must create these entries, it **can also choose not to create them when it sees the opportunity.**"

The *x86-64 Code Model* in *Mach-O Programming Topics*

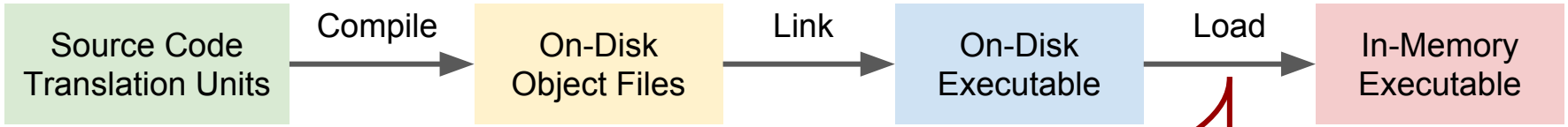
Static Builds – Phases



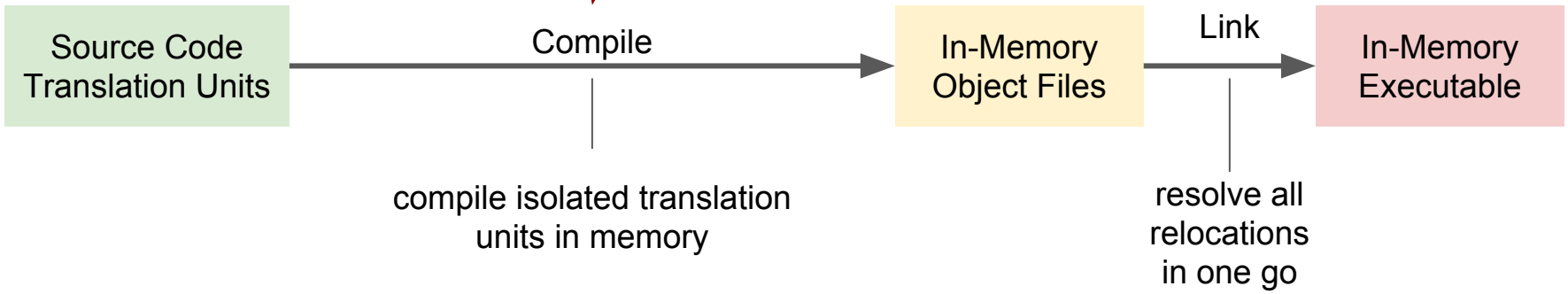
Static Builds – Phases



Static Builds – Phases



Live Builds – Phases



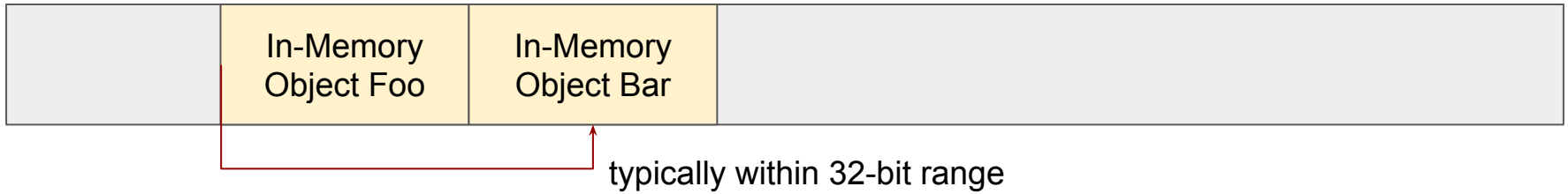
Static Builds: Link-Time Virtual Memory x64

Live Builds: Link-Time Virtual Memory x64

Static Builds: Link-Time Virtual Memory x64

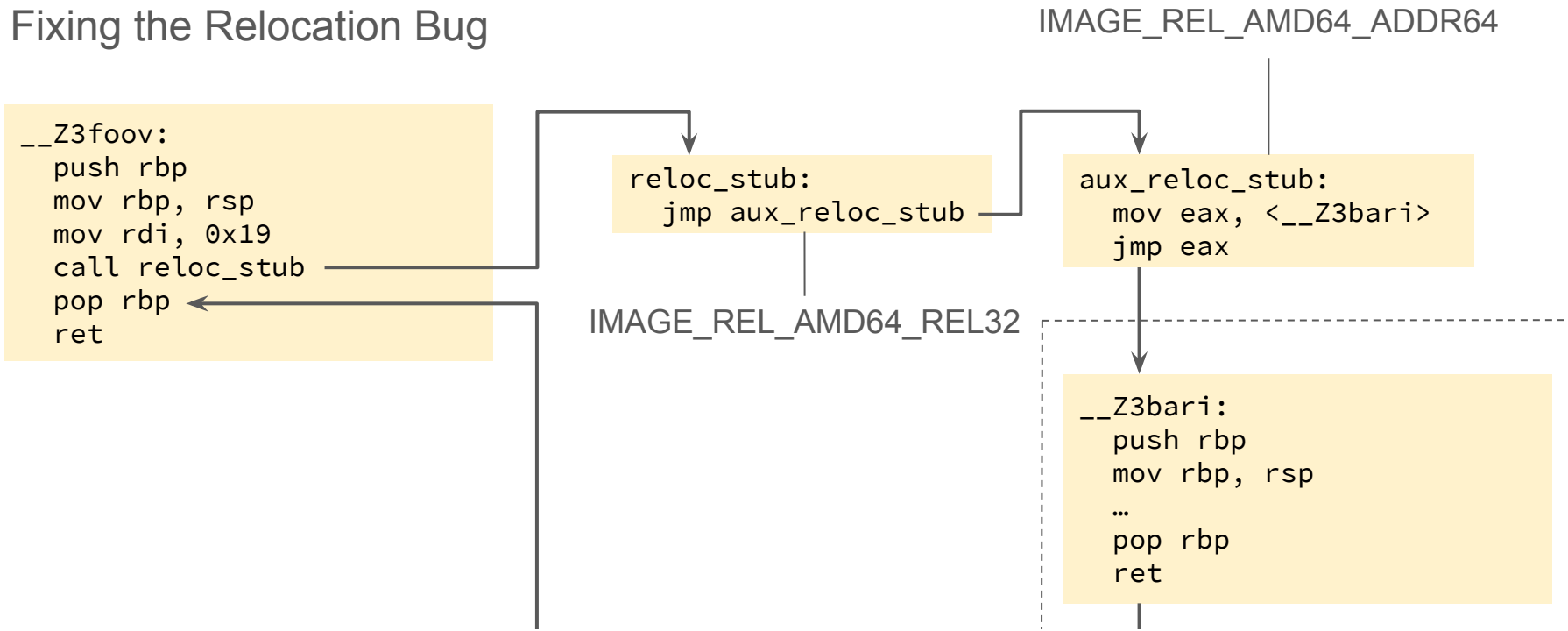
Live Builds: Link-Time Virtual Memory x64

Static Builds: Link-Time Virtual Memory x64



Live Builds: Link-Time Virtual Memory x64

Fixing the Relocation Bug



Patch Proposal

https://github.com/weliveindetail/pj-llvm/commit/f9f26dc8bf511dde02142dc2cf361f67b4964985

Literature and Further Reading

Eli Bendersky about ELF:

- Position Independent Code (PIC) in shared libraries
  http://eli.thegreenplace.net/2011/11/03/position-independent-code-pic-in-shared-libraries/
- Load-time relocation of shared libraries
  http://eli.thegreenplace.net/2011/08/25/load-time-relocation-of-shared-libraries/

Mach-O Programming Topics:

- Position-Independent Code
  https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/MachOTopics/1-Articles/dynamic_code.html
- x86-64 Code Model
  https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/MachOTopics/1-Articles/x86_64_code.html

Microsoft PE/COFF:

- A Tour of the Win32 Portable Executable File Format (1994)
  https://msdn.microsoft.com/en-us/library/ms809762.aspx
- Microsoft PE/COFF Specification (2015)
  http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/pecoff_v83.docx

Thanks for your attention.

Questions?