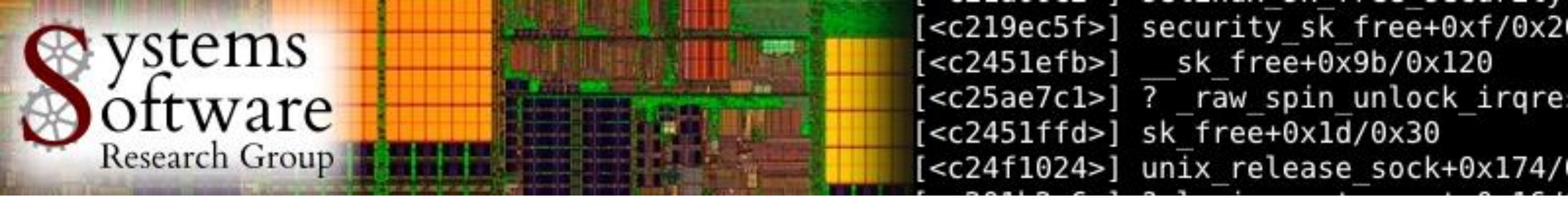


Popcorn Compiler Internals 101: The Gory Details

Goals

- Understand the design & flow of the Popcorn compiler at a high level
- Peel back the covers on LLVM internals & Popcorn-specific modifications
  - Architecture-agnostic middle-end (LLVM intermediate representation)
  - Architecture-specific back-end (target-specific machine code)
- Dig into runtime stack transformation
  - How compiler-generated metadata is stored & used at runtime
  - How migration is invoked on the *source* node
  - How execution resumes on the *destination* node

What we **won't** cover

- Traditional compiler topics
  - The language frontend or language parsing
  - Instruction selection or scheduling
  - Target-agnostic/specific optimizations
  - **Will** cover topics necessary for understanding the Popcorn compiler
- The internals of Popcorn Linux's kernel
- Performance of the system
- Benefits of migration (the hard research stuff)
  - When to migrate (performance, energy, security)
  - Where to migrate

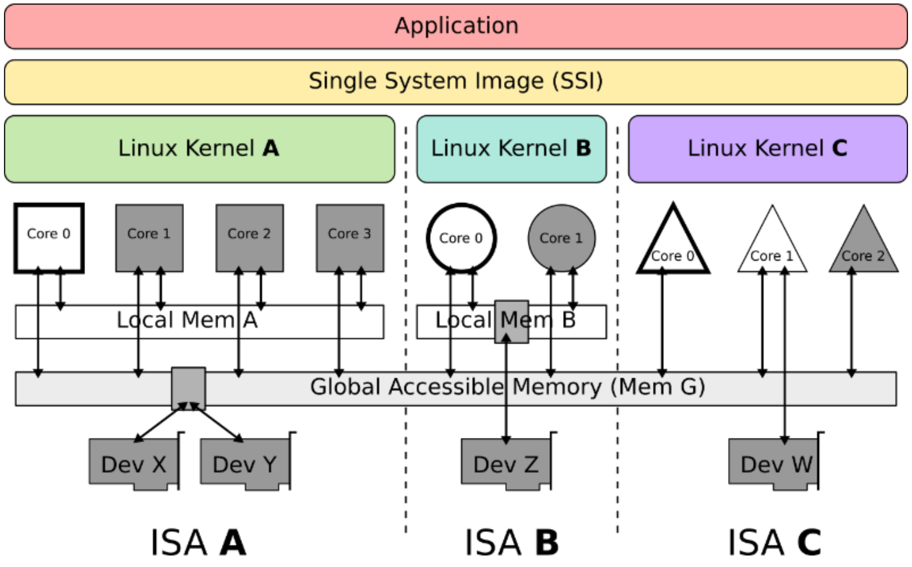
50,000 foot view of Popcorn Linux & its compiler

Popcorn Linux

- Goal – transparently execute compiled, shared-memory C/C++ applications across physically distinct, heterogeneous-ISA systems
- Developers take advantage of scalability & heterogeneity with no code modifications!

Popcorn Linux

- Multiple kernels provides ***single system image*** (SSI) allowing threads to migrate freely between nodes*
  - Thread migration – stop & resume thread on new node
  - Data migration – transfer data pages between nodes **on-demand**



*Antonio Barbalace, Robert Lyerly, Christopher Jelesnianski, Anthony Carno, Ho-Ren Chuang, Vincent Legout, and Binoy Ravindran. 2017. Breaking the Boundaries in Heterogeneous-ISA Datacenters. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS '17).

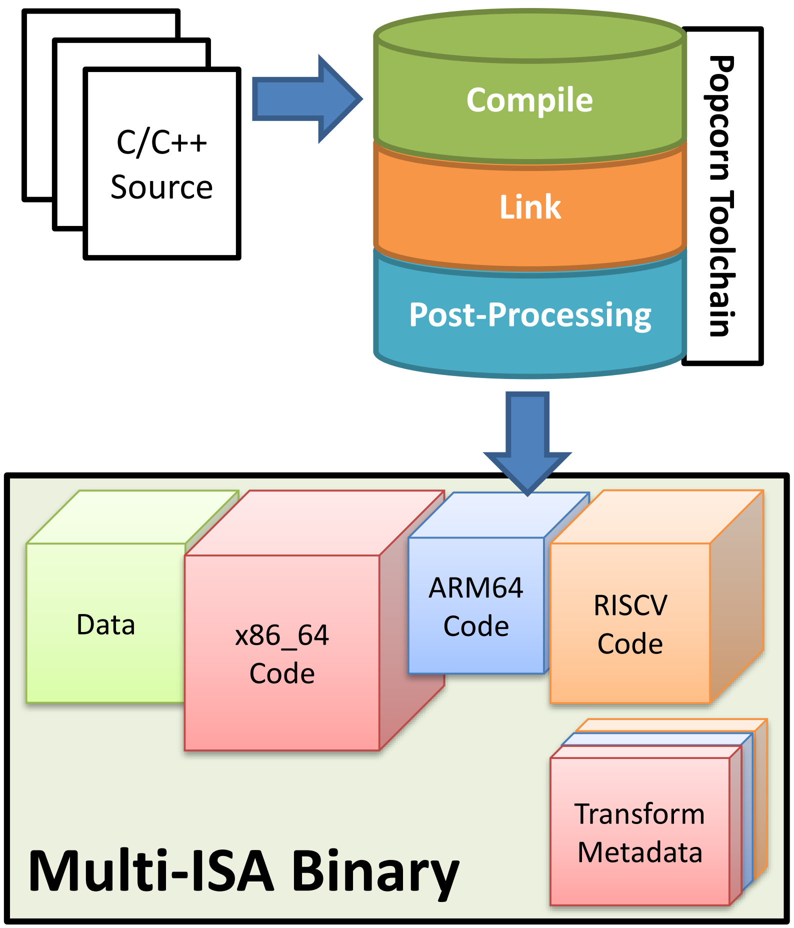
Popcorn Linux

- Threads invoke the kernel's migration service through userland syscalls
  - Can't migrate arbitrarily, only at ***migration points*** (details later)
- Kernels migrate data on-demand through the page-fault mechanism
  - Compiler/runtime need to ensure memory accesses observed by kernel are **semantically equivalent** across all architectures, i.e., pointers reference the same thing and are accessed the same way on all architectures

99% of the compiler implementation is aimed at satisfying this requirement!

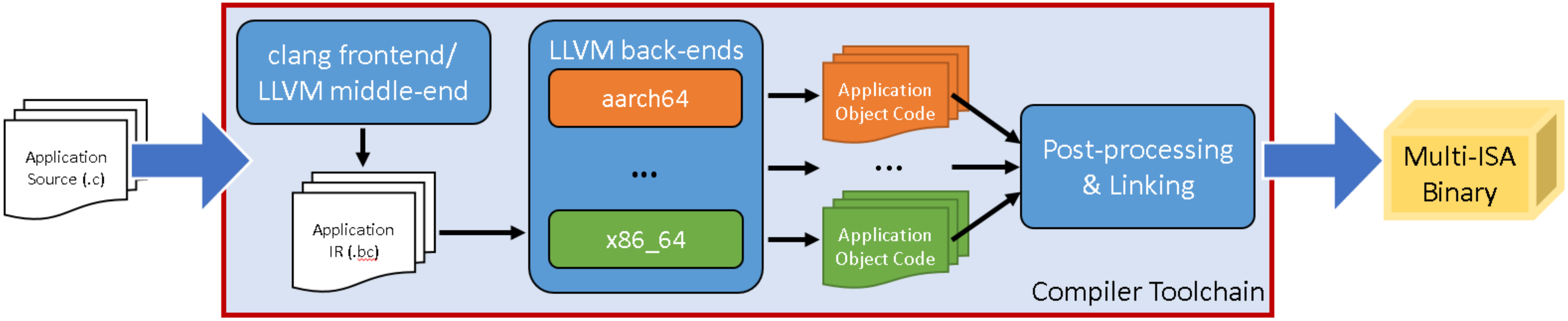
Popcorn Compiler Toolchain

- Compiler toolchain builds **multi-ISA binaries**
  - Create mostly-common virtual address space (data, code, heap)
    - Pointers are valid across all ISAs
  - Dynamically transform thread execution state (stack, registers) between ISA-specific formats at migration time
    - Transform pointers to be valid
  - Instrument generated code with migration points



Popcorn Compiler Toolchain

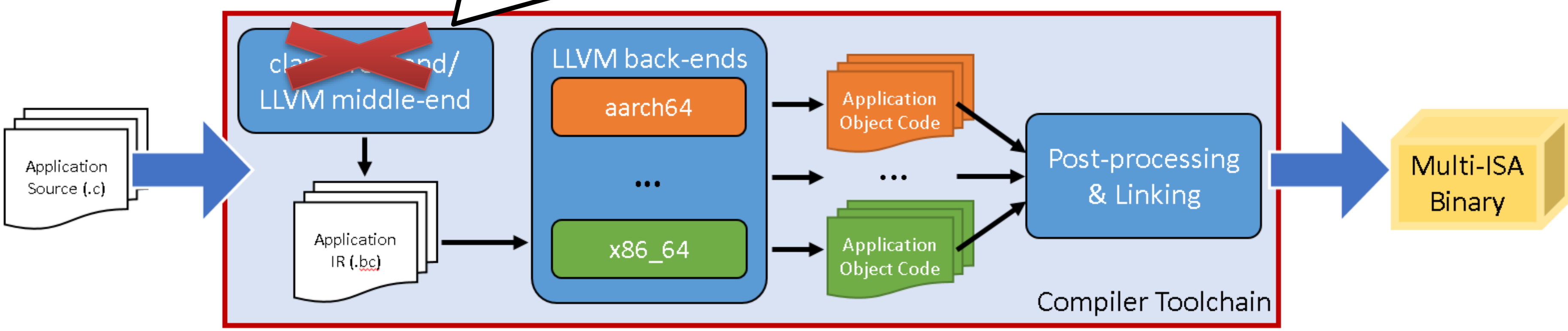
- Built on top of clang/LLVM
  - clang/LLVM 3.7.1, GNU gold 2.27, musl-libc 1.1.18
  - Custom address space alignment, post-processing tools
  - State transformation/migration libraries



Popcorn Compiler Toolchain

- Built on top of clang/LLVM
  - clang
  - Custo
  - State

- We don’t need to mess with the frontend – we only need to modify the middle- & back-end
- Compiler *should* extend to any language that can be lowered to LLVM bitcode, e.g., Rust, Javascript, etc. (YMMV)



Assumptions

- The ISAs have the following characteristics
  - 64-bit address space, meaning pointers are 64 bits
  - Little-endian data format
    - Some RISC architectures can switch between endianness formats, e.g., ARM64 & POWER8
    - Could potentially be relaxed for code, e.g., SPARCv9 allows little-endian data but requires big-endian code
  - Primitive data types have the same sizes & alignments
    - Characters – 8 bits, shorts – 16 bits, integers/longs – 32 bits
    - Long longs/pointers – 64 bits
    - Single precision floating point – 32 bits, double – 64 bits (IEEE format)
- Applications are statically linked, no dynamic libraries

Assumptions

- The ISAs have the following characteristics
  - 64-bit address space, meaning pointers are 64 bits
  - Little-endian data format
    - Some RISC architect … e.g., ARM64 & POWER8
    - Could potentially be … dian data but requires big-endia
  - Primitive data types have the same sizes & alignments
    - Characters – 8 bits, shorts – 16 bits, integers/longs – 32 bits
    - Long longs/pointers – 64 bits
    - Single precision floating point – 32 bits, double – 64 bits (IEEE format)
- Applications are statically linked, no dynamic libraries

Forces compiler to lay out all data, i.e., primitives, arrays, structs/objects, in the same format across all architectures

Limitations

- No inline assembly – opaque to liveness analysis in middle-/back-end
- No architecture-specific extensions
  - E.g., crypto instructions, ISA-specific SIMD, etc.

- Note: by limitations, we mean you **cannot have state created by these limitations live during migration – stack transformation can't handle it!**
  - E.g., you cannot have live data in SIMD registers during migration

Part 1:
A day in the life of an LLVM compilation

Part 1:
A day in the life of an LLVM compilation

All source & log files are available in the “vanilla-compile” folder

Generating LLVM IR

```
$ clang -O2 -c hello.c
$ ls
hello.c  hello.o
```

Generating LLVM IR

What's actually happening on the inside? clang/LLVM are designed to be very modular – let's break it down…

```
$ clang -O2 -c hello.c
$ ls
hello.c  hello.o
```

Generating LLVM IR

hello.c

```c
#include <stdio.h>

int main(int argc, char** argv)
{
  printf("Hello, world!\n");
  return 0;
}
```

Generating LLVM IR

```
clang -O2 -emit-llvm -S hello.c
```

hello.c

```c
#include <stdio.h>

int main(int argc, char** argv)
{
  printf("Hello, world!\n");
  return 0;
}
```

Generating LLVM IR

```
clang -O2 -emit-llvm -S hello.c
```

- `-emit-llvm`: lower C code to LLVM IR
- `-S`: emit in human-readable format

hello.c

```c
#include <stdio.h>

int main(int argc, char** argv)
{
  printf("Hello, world!\n");
  return 0;
}
```

Generating LLVM IR

```
clang -O2 -emit-llvm -S hello.c
```

hello.c

```c
#include <stdio.h>

int main(int argc, char** argv)
{
  printf("Hello, world!\n");
  return 0;
}
```

hello.ll

```llvm
; ModuleID = 'hello.c'
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

@str = private unnamed_addr constant [14 x i8] c"Hello, world!\00"

; Function Attrs: nounwind uwtable
define i32 @main(i32 %argc,
                 i8** nocapture readnone %argv) #0 {
entry:
  %puts = tail call i32 @puts(i8* getelementptr inbounds
            ([14 x i8], [14 x i8]* @str, i64 0, i64 0))
  ret i32 0
}

; Function Attrs: nounwind
declare i32 @puts(i8* nocapture readonly) #1
```

Systems Software Research Group

VIRGINIA TECH

Generating LLVM IR

hello.ll

```
; ModuleID = 'hello.c'
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-
S128"
target triple = "x86_64-unknown-linux-gnu"

@str = private unnamed_addr constant [14 x i8] c"Hello,
world!\00"

; Function Attrs: nounwind uwtable
define i32 @main(i32 %argc,
                 i8** nocapture readnone %argv) #0 {
entry:
  %puts = tail call i32 @puts(i8* getelementptr inbounds
             ([14 x i8], [14 x i8]* @str, i64 0, i64 0))
  ret i32 0
}

; Function Attrs: nounwind
declare i32 @puts(i8* nocapture readonly) #1
```

- Bitcode is ISA-agnostic, but looks like low-level assembly
  - Conceptually, assembly for some virtual machine or virtual ISA (the “VM” in LLVM)
  - Program objects (functions, global data, etc.) have not been placed in the virtual address space
  - No stack frame particulars (e.g., return address location)
  - No registers or stack slots for program variables (“values” in LLVM)
- See the language reference manual for a detailed explanation of LLVM’s IR

Generating LLVM IR

hello.ll

```
; ModuleID = 'hello.c'
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-
S128"
target triple = "x86_64-unknown-linux-gnu"

@str = private unnamed_addr constant [14 x i8] c"Hello,
world!\00"

; Function Attrs: nounwind uwtable
define i32 @main(i32 %argc,
                 i8** nocapture readnone %argv) #0 {
entry:
  %puts = tail call i32 @puts(i8* getelementptr inbounds
            ([14 x i8], [14 x i8]* @str, i64 0, i64 0))
  ret i32 0
}

; Function Attrs: nounwind
declare i32 @puts(i8* nocapture readonly) #1
```

Layouts, sizes & alignments of primitive types, e.g., integers, floating-point

Generating LLVM IR

hello.ll

```
; ModuleID = 'hello.c'
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

@str = private unnamed_addr constant [14 x i8] c"Hello, world!\00"

; Function Attrs: nounwind uwtable
define i32 @main(i32 %argc,
                 i8** nocapture readnone %argv) #0 {
entry:
  %puts = tail call i32 @puts(i8* getelementptr inbounds
          ([14 x i8], [14 x i8]* @str, i64 0, i64 0))
  ret i32 0
}

; Function Attrs: nounwind
declare i32 @puts(i8* nocapture readonly) #1
```

Target architecture/ABI

Generating LLVM IR

hello.ll

```
; ModuleID = 'hello.c'
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

@str = private unnamed_addr constant [14 x i8] c"Hello, world!\00"

; Function Attrs: nounwind uwtable
define i32 @main(i32 %argc,
                 i8** nocapture readnone %argv) #0 {
entry:
  %puts = tail call i32 @puts(i8* getelementptr inbounds
            ([14 x i8], [14 x i8]* @str, i64 0, i64 0))
  ret i32 0
}

; Function Attrs: nounwind
declare i32 @puts(i8* nocapture readonly) #1
```

Global variable definition

Generating LLVM IR

hello.ll

```
; ModuleID = 'hello.c'
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

@str = private unnamed_addr constant [14 x i8] c"Hello, world!\00"

; Function Attrs: nounwind uwtable
define i32 @main(i32 %argc,
                 i8** nocapture readnone %argv) #0 {
entry:
  %puts = tail call i32 @puts(i8* getelementptr inbounds
          ([14 x i8], [14 x i8]* @str, i64 0, i64 0))
  ret i32 0
}

; Function Attrs: nounwind
declare i32 @puts(i8* nocapture readonly) #1
```

Function definition

Generating LLVM IR

hello.ll

```
; ModuleID = 'hello.c'
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-
S128"
target triple = "x86_64-unknown-linux-gnu"

@str = private unnamed_addr constant [14 x i8] c"Hello,
world!\00"

; Function Attrs: nounwind uwtable
define i32 @main(i32 %argc,
                 i8** nocapture readnone %argv) #0 {
entry:
  %puts = tail call i32 @puts(i8* getelementptr inbounds
              ([14 x i8], [14 x i8]* @str, i64 0, i64 0))
  ret i32 0
}

; Function Attrs: nounwind
declare i32 @puts(i8* nocapture readonly) #1
```

Function body in single-static assignment form (SSA), organized as a series of basic blocks containing operations

- **Basic block**: a sequence of non-control flow instructions terminated by control flow, e.g., branch or jump

Systems Software Research Group

VIRGINIA TECH™

Generating LLVM IR

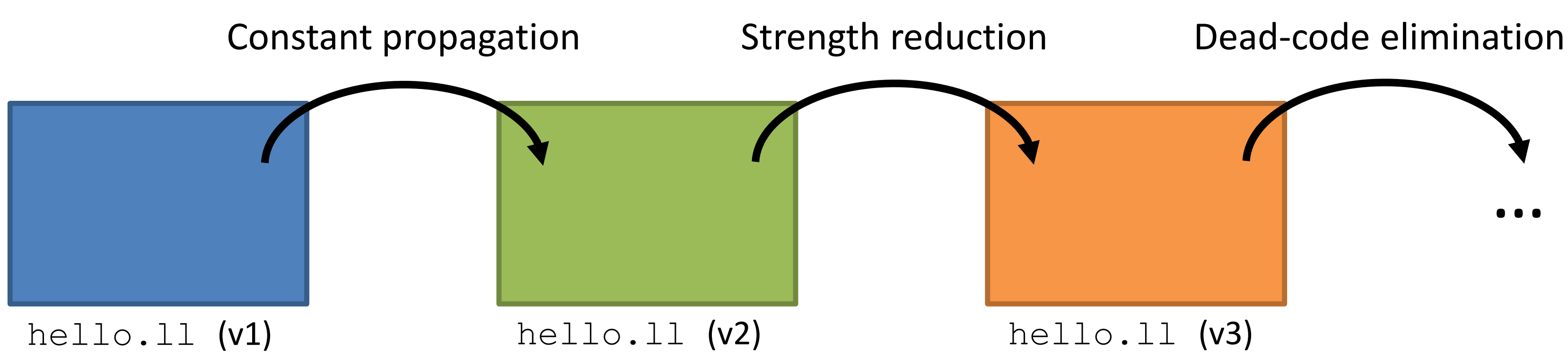
hello.ll

```
; ModuleID = 'hello.c'
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-unknown-linux-gnu"

@str = private unnamed_addr constant [14 x i8] c"Hello, world!\00"

; Function Attrs: nounwind uwtable
define i32 @main(i32 %argc,
                 i8** nocapture readnone %argv) #0 {
entry:
  %puts = tail call i32 @puts(i8* getelementptr inbounds
        ([14 x i8], [14 x i8]* @str, i64 0, i64 0))
  ret i32 0
}

; Function Attrs: nounwind
declare i32 @puts(i8* nocapture readonly) #1
```

Function declaration – definition lives in another source file or “compilation module” in LLVM terminology

Optimizing LLVM IR

- The optimizer (`opt`) is LLVM's "middle-end"
  - Don't worry, `clang` includes these optimizations!
- LLVM applies a large number of target-agnostic optimizations to the IR
  - Optimizations structured as a series of ***passes*** run over the IR, which consume IR & produce (hopefully) optimized IR



Optimizing LLVM IR

- What is single static assignment?
  - Program variables are lowered to **values** which have the following characteristics:
    - Each value is assigned **exactly once**
    - Each value **must be defined before its use**
    - Once assigned, a value is **immutable**
  - Program variables assigned multiple times are lowered to distinct values (**versioned**) by the compiler
  - LLVM bitcode has no notion of a variable inside of functions – only values!

Optimizing LLVM IR

- Why single static assignment?
  - Provides lots of useful information by construction
    - Explicit use-def chains of a value in a function
    - Liveness ranges for determining when a value is live inside a function
  - Enables many useful analyses & compiler optimizations
    - Instruction scheduling
    - Liveness analysis/register allocation
    - Tons of optimizations – see Wikipedia for examples

Lowering IR to machine code

- The system compiler (`llc`) is LLVM's back-end
  - Again, `clang` includes this too!
  - Implements the semantics of the IR using operations defined by the target's ISA
- LLVM implements code generation through its target-independent code generator
  - Another series of passes analyze and transform bitcode to assembly
  - Targets are "plugins" which describe opcodes, registers, ABIs, etc.
  - Most target-specific code lowering is implemented in a target-independent manner!

Lowering IR to machine code

- LLVM has a complex pattern-matching/graph-based framework for instruction selection & scheduling
  - The subject of several Ph.D. theses, not this tutorial
- The backend lowers bitcode into another type of IR, called machine code IR
  - Also in SSA, but very close to the target architecture

Lowering IR to machine code

hello.ll

```
; ModuleID = 'hello.c'
target datalayout = "…"
target triple = "…"

@str = … [14 x i8] c"Hello, world!\00"

; Function Attrs: nounwind uwtable
define i32 @main(i32 %argc,
                 i8** %argv) #0 {
entry:
  %puts = tail call i32 @puts(i8* …)
  ret i32 0
}

; Function Attrs: nounwind
declare i32 @puts(i8*) #1
```

Systems Software Research Group

VIRGINIA TECH™

Lowering IR to machine code

`llc hello.ll`

hello.ll

```
; ModuleID = 'hello.c'
target datalayout = "…"
target triple = "…"

@str = … [14 x i8] c"Hello, world!\00"

; Function Attrs: nounwind uwtable
define i32 @main(i32 %argc,
                 i8** %argv) #0 {
entry:
  %puts = tail call i32 @puts(i8* …)
  ret i32 0
}

; Function Attrs: nounwind
declare i32 @puts(i8*) #1
```

Lowering IR to machine code

`llc hello.ll`

hello.ll

```
; ModuleID = 'hello.c'
target datalayout = "…"
target triple = "…"

@str = … [14 x i8] c"Hello, world!\00"

; Function Attrs: nounwind uwtable
define i32 @main(i32 %argc,
                 i8** %argv) #0 {
entry:
  %puts = tail call i32 @puts(i8* …)
  ret i32 0
}

; Function Attrs: nounwind
declare i32 @puts(i8*) #1
```

hello.s

```
             .text
             .file          "hello.ll"
             .globl         main
             .align         16, 0x90
             .type          main,@function
main:                                       # @main
             .cfi_startproc
# BB#0:                                     # %entry
             pushq          %rax
.Ltmp0:
             .cfi_def_cfa_offset 16
             movl           $.Lstr, %edi
             callq          puts
             xorl           %eax, %eax
             popq           %rdx
             retq
.Lfunc_end0:
             .size          main, .Lfunc_end0-main
             .cfi_endproc

             .type          .Lstr,@object    # @str
             .section
             .rodata.str1.1,"aMS",@progbits,1
.Lstr:
             .asciz         "Hello, world!"
             .size          .Lstr, 14
```

Lowering IR to machine code

`llc hello.ll`

hello.ll

```
; ModuleID = 'hello.c'
target datalayout = "…"
target triple = "…"

@str = … [14 x i8] c"Hello, world!\00"

; Function Attrs: nounwind uwtable
define i32 @main(i32 %argc,
                 i8** %argv) #0 {
entry:
  %puts = tail call i32 @puts(i8* …)
  ret i32 0
}

; Function Attrs: nounwind
declare i32 @puts(i8*) #1
```

But in the middle…

hello.s

```
             .text
             .file        "hello.ll"
             .globl       main
             .align       16, 0x90
             .type        main,@function
main:                                     # @main
             .cfi_startproc
# BB#0:                                   # %entry
             pushq        %rax
.Ltmp0:
             .cfi_def_cfa_offset 16
             movl         $.Lstr, %edi
             callq        puts
             xorl         %eax, %eax
             popq         %rdx
             retq
.Lfunc_end0:
             .size        main, .Lfunc_end0-main
             .cfi_endproc

             .type        .Lstr,@object   # @str
             .section
             .rodata.str1.1,"aMS",@progbits,1
.Lstr:
             .asciz       "Hello, world!"
             .size        .Lstr, 14
```

Lowering IR to machine code

`llc` **`-debug-only=regalloc`** `hello.ll`

```
********** MACHINEINSTRS **********
# Machine code for function main: Post SSA

0B          BB#0: derived from LLVM BB %entry
16B                 ADJCALLSTACKDOWN64 0, 0, %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
32B                 %vreg2<def> = MOV32ri64 <ga:@str>; GR32:%vreg2
48B                 %vreg3<def> = SUBREG_TO_REG 0, %vreg2, 4; GR64:%vreg3 GR32:%vreg2
64B                 %RDI<def> = COPY %vreg3; GR64:%vreg3
80B                 CALL64pcrel32 <ga:@puts>, <regmask>, %RSP<imp-use>, %RDI<imp-use,kill>,
                              %RSP<imp-def>, %EAX<imp-def,dead>
96B                 ADJCALLSTACKUP64 0, 0, %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
112B                %vreg5<def> = MOV32r0 %EFLAGS<imp-def,dead>; GR32:%vreg5
128B                %EAX<def> = COPY %vreg5; GR32:%vreg5
144B                RETQ %EAX<kill>
```

Lowering IR to machine code

All values in middle-end which could *potentially* be held in a register are lowered to **virtual registers** (vregs)

- LLVM starts by assuming a virtual register set with unlimited registers
- Deciding which values are actually placed in registers and which are spilled to the stack is the purpose of the ***register allocator***

```
*********
# Machine

0B          BB#0: derive
16B                   ADJC      STACKDOWN64 0, 0, %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
32B                   %vreg2<def> = MOV32ri64 <ga:@str>; GR32:%vreg2
48B                   %vreg3<def> = SUBREG_TO_REG 0, %vreg2, 4; GR64:%vreg3 GR32:%vreg2
64B                   %RDI<def> = COPY %vreg3; GR64:%vreg3
80B                   CALL64pcrel32 <ga:@puts>, <regmask>, %RSP<imp-use>, %RDI<imp-use,kill>,
                                %RSP<imp-def>, %EAX<imp-def,dead>
96B                   ADJCALLSTACKUP64 0, 0, %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
112B                  %vreg5<def> = MOV32r0 %EFLAGS<imp-def,dead>; GR32:%vreg5
128B                  %EAX<def> = COPY %vreg5; GR32:%vreg5
144B                  RETQ %EAX<kill>
```

Lowering IR to machine code

```
llc -debug-only=regalloc hello.ll
```

LLVM bitcode operations are lowered to target-specific opcodes

```
********** MACHINEINSTRS **********
# Machine code

0B        BB#0: derived from
16B                ADJCALLSTACKDOW      0, 0, %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
32B                %vreg2<def> = MOV32ri64 <ga:@str>; GR32:%vreg2
48B                %vreg3<def> = SUBREG_TO_REG 0, %vreg2, 4; GR64:%vreg3 GR32:%vreg2
64B                %RDI<def> = COPY %vreg3; GR64:%vreg3
80B                CALL64pcrel32 <ga:@puts>, <regmask>, %RSP<imp-use>, %RDI<imp-use,kill>,
                             %RSP<imp-def>, %EAX<imp-def,dead>
96B                ADJCALLSTACKUP64 0, 0, %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
112B               %vreg5<def> = MOV32r0 %EFLAGS<imp-def,dead>; GR32:%vreg5
128B               %EAX<def> = COPY %vreg5; GR32:%vreg5
144B               RETQ %EAX<kill>
```

Lowering IR to machine code

```
llc -debug-only=regalloc hello.ll
```

Operands are encoded as part of the instruction

- Registers, constants, references to other program objects (here, reference to the string literal “Hello, world!”)

```
********** MACHINEINSTRS
# Machine code for funct

0B        BB#0: derived
16B                 ADJCALLSTACKDOWN64 0, 0,        <imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
32B                 %vreg2<def> = MOV32ri64 <ga:@str>; GR32:%vreg2
48B                 %vreg3<def> = SUBREG_TO_REG 0, %vreg2, 4; GR64:%vreg3 GR32:%vreg2
64B                 %RDI<def> = COPY %vreg3; GR64:%vreg3
80B                 CALL64pcrel32 <ga:@puts>, <regmask>, %RSP<imp-use>, %RDI<imp-use,kill>,
                              %RSP<imp-def>, %EAX<imp-def,dead>
96B                 ADJCALLSTACKUP64 0, 0, %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
112B                %vreg5<def> = MOV32r0 %EFLAGS<imp-def,dead>; GR32:%vreg5
128B                %EAX<def> = COPY %vreg5; GR32:%vreg5
144B                RETQ %EAX<kill>
```

Systems Software Research Group

VIRGINIA TECH

Lowering IR to machine code

`llc` **`-debug-only=regalloc`** `hello.ll`

```
********** MACHINEINSTRS **********
# Machine code for function main: Post SSA

0B
16B                                                                                                   use>
32B                            V32ri64 <ga:@str>; GR32:%vreg2
48B                  %vreg3<def> = SUBREG_TO_REG 0, %vreg2, 4; GR64:%vreg3 GR32:%vreg2
64B                  %RDI<def> = COPY %vreg3; GR64:%vreg3
80B                  CALL64pcrel32 <ga:@puts>, <regmask>, %RSP<imp-use>, %RDI<imp-use,kill>,
                               %RSP<imp-def>, %EAX<imp-def,dead>
96B                  ADJCALLSTACKUP64 0, 0, %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
112B                 %vreg5<def> = MOV32r0 %EFLAGS<imp-def,dead>; GR32:%vreg5
128B                 %EAX<def> = COPY %vreg5; GR32:%vreg5
144B                 RETQ %EAX<kill>
```

Physical registers appear prior to register allocation due to ABI calling conventions

Lowering IR to machine code

Post-register allocation

```
********** MACHINEINSTRS **********
# Machine code for function main: Post SSA

0B          BB#0: derived from LLVM BB %entry
16B                   ADJCALLSTACKDOWN64 0, 0, %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
64B                   %EDI<def,dead> = MOV32ri64 <ga:@str>, %RDI<imp-def>
80B                   CALL64pcrel32 <ga:@puts>, <regmask>, %RSP<imp-use>, %RDI<imp-use,kill>,
%RSP<imp-def>, %EAX<imp-def,dead>
96B                   ADJCALLSTACKUP64 0, 0, %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
128B                  %EAX<def> = MOV32r0 %EFLAGS<imp-def,dead>
144B                  RETQ %EAX<kill>
```

Lowering IR to machine code

Post-register allocation

```
********** MACHINEINSTRS **********
# Machine code for function main: Post SSA

0B        BB#0: derived from LLVM BB %entry
16B               ADJCALLSTACKDOWN64 0, 0, %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
64B               %EDI<def,dead> = MOV32ri64 <ga:@str>, %RDI<imp-def>
80B               CALL64pcrel32 <ga:@puts>, <regmask>, %RSP<imp-use>, %RDI<imp-use,kill>,
%RSP<imp-def>, %EAX<imp-def,dead>
96B               ADJCALLSTACKUP64 0, 0, %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
128B              %EAX<def> = MOV32r0 %EFLAGS<imp-def,dead>
144B              RETQ %EAX<kill>
```

- Vregs are assigned to physical registers or stack slots based on liveness analysis
  - In this case, LLVM was able to eliminate several vreg definitions by allocating them to the same physical register used in the calling conventions (`%edi`, `%eax`)

Compiler Goals Redux: In-Depth

Goals Redux

- Remember, we must ensure pointers are used in a **semantically equivalent** way across all architectures
- Approaches
  - Lay out program objects in a common format and at the same virtual address across all compilations
  - Transform program objects between ISA-specific formats (either statically or at migration time), update references accordingly
- The Popcorn compiler utilizes a mixture of both

Goals Redux

- Popcorn lays out applications in a ***mostly-common*** format
  - Program objects (global data, functions, the heap) placed at identical addresses for all architectures
  - Stack/register set highly optimized for each ISA – **no common format**
    - ISA defines number and types of registers
    - Compiler tailors stack frame to each ISA based on register allocation results, i.e., compiler spills values to stack that it can't put in registers
- Transform stack & registers between formats at migration time, everything else is aligned at link-time

Goals Redux

- Data objects are equivalent across all architectures
  - Same primitive type sizes & alignments, compiler is forced to lay out higher order types in an identical format
  - Can be placed at identical locations at link-time (details later)
- Code **cannot** be in identical format for different ISAs
  - Like register set, operations/operand format is defined by ISA
  - The manner in which a processor implements a given piece of code is dependent on the operations it supports
  - In other words, ***a single piece of code compiled for two different architectures, while semantically identical, executes in different ways***

Goals Redux

```c
void vec_add(const int* a, const int* b, int* c, size_t num) {
  size_t i;
  for(i = 0; i < num; i++)
    c[i] = a[i] + b[i];
}
```

```
vec_add_arm.o:     file format elf64-littleaarch64


Disassembly of section .text:

0000000000000000 <vec_add>:
   0: b40000e3   cbz x3, 1c <vec_add+0x1c>
   4: b8404408   ldr w8, [x0],#4
   8: b8404429   ldr w9, [x1],#4
   c: 0b080128   add w8, w9, w8
  10: b8004448   str w8, [x2],#4
  14: d1000463   sub x3, x3, #0x1
  18: b5ffff63   cbnz  x3, 4 <vec_add+0x4>
  1c: d65f03c0   ret
```

```
vec_add_x86.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <vec_add>:
   0: 48 85 c9                test   %rcx,%rcx
   3: 74 22                   je     27 <vec_add+0x27>
   5: 66 66 2e 0f 1f 84 00    data16 nopw %cs:0x0(%rax,%rax,1)
   c: 00 00 00 00
  10: 8b 06                   mov    (%rsi),%eax
  12: 03 07                   add    (%rdi),%eax
  14: 89 02                   mov    %eax,(%rdx)
  16: 48 83 c7 04             add    $0x4,%rdi
  1a: 48 83 c6 04             add    $0x4,%rsi
  1e: 48 83 c2 04             add    $0x4,%rdx
  22: 48 ff c9                dec    %rcx
  25: 75 e9                   jne    10 <vec_add+0x10>
  27: c3                      retq
```

Goals Redux

```
void vec_add(const int* a, const int* b, int* c, size_t num) {
  size_t i;
  for(i = 0; i < num; i++)
    c[i] = a[i] + b[i];
}
```

```
vec_add_arm.o:     file format elf64-littleaarch64


Disassembly of section .text:

0000000000000000 <vec_add>:
   0: b40000e3  cbz x3, 1c <vec_add+0x1c>
   4: b8404408  ldr w8, [x0],#4
   8: b8404429  ldr w9, [x1],#4
   c: 0b080128  add w8, w9, w8
  10: b8004448  str w8, [x2],#4
  14: d1000463  sub x3, x3, #0x1
  18: b5ffff63  cbnz  x3, 4 <vec_add+0x4>
  1c: d65f03c0  ret
```

Perform addition & update pointers

```
vec_add_x86.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <vec_add>:
   0: 48 85 c9                test   %rcx,%rcx
   3: 74 22                   je     27 <vec_add+0x27>
   5: 66 66 2e 0f 1f 84 00    data16 nopw %cs:0x0(%rax,%rax,1)
   c: 00 00 00 00
  10: 8b 06                   mov    (%rsi),%eax
  12: 03 07                   add    (%rdi),%eax
  14: 89 02                   mov    %eax,(%rdx)
  16: 48 83 c7 04             add    $0x4,%rdi
  1a: 48 83 c6 04             add    $0x4,%rsi
  1e: 48 83 c2 04             add    $0x4,%rdx
  22: 48 ff c9                dec    %rcx
  25: 75 e9                   jne    10 <vec_add+0x10>
  27: c3                      retq
```

Goals Redux

```c
void vec_add(const int* a, const int* b, int* c, size_t num) {
  size_t i;
  for(i = 0; i < num; i++)
    c[i] = a[i] + b[i];
}
```

```
vec_add_arm.o:     file format elf64-littleaarch64


Disassembly of section .text:

0000000000000000 <vec_add>:
   0: b40000e3  cbz x3, 1c <vec_add+0x1c>
   4: b8404408  ldr w8, [x0],#4
   8: b8404429  ldr w9, [x1],#4
   c: 0b080128  add w8, w9, w8
  10: b8004448  str w8, [x2],#4
  14: d1000463  sub x3, x3, #0x1
  18: b5ffff63  cbnz  x3, 4 <vec_add+0x4>
  1c: d65f03c0  ret
```

Perform addition & update pointers

```
vec_add_x86.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <vec_add>:
   0: 48 85 c9                 test   %rcx,%rcx
   3: 74 22                    je     27 <vec_add+0x27>
   5: 66 66 2e 0f 1f 84 00     data16 nopw %cs:0x0(%rax,%rax,1)
   c: 00 00 00 00
  10: 8b 06                    mov    (%rsi),%eax
  12: 03 07                    add    (%rdi),%eax
  14: 89 02                    mov    %eax,(%rdx)
  16: 48 83 c7 04              add    $0x4,%rdi
  1a: 48 83 c6 04              add    $0x4,%rsi
  1e: 48 83 c2 04              add    $0x4,%rdx
  22: 48 ff c9                 dec    %rcx
  25: 75 e9                    jne    10 <vec_add+0x10>
  27: c3                       retq
```

Perform addition

Update pointers

Goals Redux

```
void vec_add(const int* a, const int* b, int* c, size_t num) {
  size_t i;
  for(i = 0; i < num; i++)
    c[i] = a[i] + b[i];
}
```

```
vec_add_arm.o:     file format elf64-littleaarch64


Disassembly of section .text:

0000000000000000 <vec_add>:
   0: b40000e3  cbz x3, 1c <vec_add+0x1c>
   4: b8404408  ldr w8, [x0],#4
   8: b8404429  ldr w9, [x1],#4
   c: 0b080128  add w8, w9, w8
  10: b8004448  str w8, [x2],#4
  14: d1000463  sub x3, x3, #0x1
  18: b5ffff63  cbnz  x3, 4 <vec_add+0x4>
  1c: d65f03c0  ret
```

Perform addition & update pointers

```
vec_add_x86.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <vec_add>:
   0: 48 85 c9                test   %rcx,%rcx
   3: 74 22                   je     27 <vec_add+0x27>
   5: 66 66 2e 0f 1f 84 00    data16 nopw %cs:0x0(%rax,%rax,1)
   c: 00 00 00 00
  10: 8b 06                   mov    (%rsi),%eax
  12: 03 07                   add    (%rdi),%eax
  14: 89 02                   mov    %eax,(%rdx)
  16: 48 83 c7 04             add    $0x4,%rdi
  1a: 48 83 c6 04             add    $0x4,%rsi
  1e: 48 83 c2 04             add    $0x4,%rdx
  22: 48 ff c9                dec    %rcx
  25: 75 e9                   jne    10 <vec_add+0x10>
  27: c3                      retq
```

Perform addition

Update pointers

Goals Redux

```c
void vec_add(const int* a, const int* b, int* c, size_t num) {
  size_t i;
  for(i = 0; i < num; i++)
    c[i] = a[i] + b[i];
}
```

```
vec_add_arm.o:     file format elf64-littleaarch64


Disassembly of section .text:

0000000000000000 <vec_add>:
   0: b40000e3  cbz x3, 1c <vec_add+0x1c>
   4: b8404408  ldr w8, [x0],#4
   8: b8404429  ldr w9, [x1],#4
   c: 0b080128  add w8, w9, w8
  10: b8004448  str w8, [x2],#4
  14: d1000463  sub x3, x3, #0x1
  18: b5ffff63  cbnz  x3, 4 <vec_add+0x4>
  1c: d65f03c0  ret
```

Perform addition & update pointers

Perform addition

Update pointers

Safe for migration

```
vec_add_x86.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <vec_add>:
   0: 48 85 c9                test   %rcx,%rcx
   3: 74 22                   je     27 <vec_add+0x27>
   5: 66 66 2e 0f 1f 84 00    data16 nopw %cs:0x0(%rax,%rax,1)
   c: 00 00 00 00
  10: 8b 06                   mov    (%rsi),%eax
  12: 03 07                   add    (%rdi),%eax
  14: 89 02                   mov    %eax,(%rdx)
  16: 48 83 c7 04             add    $0x4,%rdi
  1a: 48 83 c6 04             add    $0x4,%rsi
  1e: 48 83 c2 04             add    $0x4,%rdx
  22: 48 ff c9                dec    %rcx
  25: 75 e9                   jne    10 <vec_add+0x10>
  27: c3                      retq
```

Goals Redux

- Questions in heterogeneous-ISA execution*
    1. Given a program address in machine code for one ISA, how do we find the equivalent address in code for another ISA (if there is one)?
    2. If we've established such a mapping, how do we generate a transformation of execution state (registers, stack frames) between ISA-specific formats? How do we ensure such a mapping is feasible?

*David G. von Bank, Charles M. Shub, and Robert W. Sebesta. 1994. A unified model of pointwise equivalence of procedural computations. *ACM Trans. Program. Lang. Syst.* 16, 6 (November 1994), 1842-1874.

Goals Redux

- Questions in heterogeneous-ISA execution*
  1. Given a program address in machine code for one ISA, how do we find the equivalent address in code for another ISA (if there is one)?
  2. If we've established such a mapping, how do we generate a transformation of execution state (registers, stack frames) between ISA-specific formats? How do we ensure such a mapping is feasible?

The compiler programmatically selects a set of all such ***equivalence points*** and inserts call-outs to a migration library. These inserted points are called ***migration points***.

*David G. von Bank, Charles M. Shub, and Robert W. Sebesta. 1994. A unified model of pointwise equivalence of procedural computations. *ACM Trans. Program. Lang. Syst.* 16, 6 (November 1994), 1842-1874.

Part 2:
A day in the life of a Popcorn compilation

Part 2:
A day in the life of a Popcorn compilation

All source & log files are available in the “het-compile” folder

Generating LLVM IR

fizzbuzz.c

```c
#include <stdio.h>

void fizzbuzz(unsigned max)
{
  unsigned i;
  for(i = 0; i < max; i++)
  {
    if((i % 5) == 0 && (i % 3) == 0)
      printf("fizzbuzz\n");
    else if((i % 5) == 0)
      printf("fizz\n");
    else if((i % 3) == 0)
      printf("buzz\n");
  }
}
```

Generating LLVM IR

```
clang -O2 -emit-llvm -S fizzbuzz.c
```

fizzbuzz.c

```c
#include <stdio.h>

void fizzbuzz(unsigned max)
{
  unsigned i;
  for(i = 0; i < max; i++)
  {
    if((i % 5) == 0 && (i % 3) == 0)
      printf("fizzbuzz\n");
    else if((i % 5) == 0)
      printf("fizz\n");
    else if((i % 3) == 0)
      printf("buzz\n");
  }
}
```

Generating LLVM IR

```
clang -O2 -emit-llvm -S fizzbuzz.c
```

fizzbuzz.c

```c
#include <stdio.h>

void fizzbuzz(unsigned max)
{
  unsigned i;
  for(i = 0; i < max; i++)
  {
    if((i % 5) == 0 && (i % 3) == 0)
      printf("fizzbuzz\n");
    else if((i % 5) == 0)
      printf("fizz\n");
    else if((i % 3) == 0)
      printf("buzz\n");
  }
}
```

fizzbuzz.ll

```llvm
define void @fizzbuzz(i32 %max) #0 {
entry:
  %cmp.23 = icmp eq i32 %max, 0
  br i1 %cmp.23, label %for.end, label %for.body.preheader

for.body.preheader:   ; preds = %entry
  br label %for.body

for.body:   ; preds = %for.body.preheader, %for.inc
  %i.024 = phi i32 [ %inc, %for.inc ], [ 0, %for.body.preheader ]
  %rem = urem i32 %i.024, 5
  %rem2 = urem i32 %i.024, 3
  %cmp3 = icmp eq i32 %rem2, 0
  …(if-else statement implementation)…

for.inc:   ; preds = %if.then, %if.else.8, %if.then.11, %if.then.6
  %inc = add nuw i32 %i.024, 1
  %exitcond = icmp eq i32 %inc, %max
  br i1 %exitcond, label %for.end.loopexit, label %for.body

for.end.loopexit:   ; preds = %for.inc
  br label %for.end

for.end:   ; preds = %for.end.loopexit, %entry
  ret void
}
```

Inserting Migration Points

fizzbuzz.ll

```
define void @fizzbuzz(i32 %max) #0 {
entry:
  %cmp.23 = icmp eq i32 %max, 0
  br i1 %cmp.23, label %for.end,
                 label %for.body.preheader

for.body.preheader:
  br label %for.body

for.body:
  %i.024 = phi i32 [ %inc, %for.inc ],
                   [ 0, %for.body.preheader ]
  …(if-else statement implementation)…

for.inc:
  …(increment loop induction variable)…
  br i1 %exitcond, label %for.end.loopexit,
                   label %for.body

for.end.loopexit:
  br label %for.end

for.end:
  ret void
}
```

Inserting Migration Points

```
opt -select-migration-points -migration-points -S -o fizzbuzz-migpoints.ll fizzbuzz.ll
```

fizzbuzz.ll

```
define void @fizzbuzz(i32 %max) #0 {
entry:
  %cmp.23 = icmp eq i32 %max, 0
  br i1 %cmp.23, label %for.end,
               label %for.body.preheader

for.body.preheader:
  br label %for.body

for.body:
  %i.024 = phi i32 [ %inc, %for.inc ],
                   [ 0, %for.body.preheader ]
  …(if-else statement implementation)…

for.inc:
  …(increment loop induction variable)…
  br i1 %exitcond, label %for.end.loopexit,
                   label %for.body

for.end.loopexit:
  br label %for.end

for.end:
  ret void
}
```

Inserting Migration Points

```
opt -select-migration-points -migration-points -S -o fizzbuzz-migpoints.ll fizzbuzz.ll
```

`fizzbuzz.ll`

```
define void @fizzbuzz(i32 %max) #0 {
entry:
  %cmp.23 = icmp eq i32 %max, 0
  br i1 %cmp.23, label %for.end,
                 label %for.body.preheader

for.body.preheader:
  br label %for.body

for.body:
  %i.024 = phi i32 [ %inc, %for.inc ],
                   [ 0, %for.body.preheader ]
  …(if-else statement implementation)…

for.inc:
  …(increment loop induction variable)…
  br i1 %exitcond, label %for.end.loopexit,
                   label %for.body

for.end.loopexit:
  br label %for.end

for.end:
  ret void
}
```

`opt` lets you select individual passes to be run over the IR

- Popcorn passes are patched into LLVM at compiler install time
- Explore Popcorn passes/LLVM mods in `<repo>/patches/llvm/src`

Inserting Migration Points

```
opt -select-migration-points -migration-points -S -o fizzbuzz-migpoints.ll fizzbuzz.ll
```

fizzbuzz.ll

```
define void @fizzbuzz(i32 %max) #0 {
entry:
  %cmp.23 = icmp eq i32 %max, 0
  br i1 %cmp.23, label %for.end,
                 label %for.body.preheader

for.body.preheader:
  br label %for.body

for.body:
  %i.024 = phi i32 [ %inc, %for.inc ],
                   [ 0, %for.body.preheader ]
  …(if-else statement implementation)…

for.inc:
  …(increment loop induction variable)…
  br i1 %exitcond, label %for.end.loopexit,
                   label %for.body

for.end.loopexit:
  br label %for.end

for.end:
  ret void
}
```

fizzbuzz-migpoints.ll

```
define void @fizzbuzz(i32 %max) #0 {
entry:
  call void @check_migrate(void (i8*)* null, i8* null)
  %cmp.23 = icmp eq i32 %max, 0, !popcorn !2
  br i1 %cmp.23, label %for.end,
                 label %for.body.preheader

for.body.preheader:
  br label %for.body

for.body:
  %i.024 = phi i32 [ %inc, %for.inc ],
                   [ 0, %for.body.preheader ]
  …(if-else statement implementation)…

for.inc:
  …(increment loop induction variable)…
  br i1 %exitcond, label %for.end.loopexit,
                   label %for.body

for.end.loopexit:
  br label %for.end

for.end:
  call void @check_migrate(void (i8*)* null, i8* null)
  ret void, !popcorn !2
}
```

Inserting Migration Points

```
opt -select-migration-points -migration-points -S -o fizzbuzz-migpoints.ll fizzbuzz.ll
```

fizzbuzz.ll

```
de
...
for.inc:
  …(increment loop induction variable)…
  br i1 %exitcond, label %for.end.loopexit,
                   label %for.body

for.end.loopexit:
  br label %for.end

for.end:
  ret void
}
```

Compiler inserts migration points at beginning & end of all functions by default

- `check_migrate` defined in libmigrate.a and linked in by the compiler
- See `SelectMigrationPoints.cpp` in compiler repo for tuning options

fizzbuzz-migpoints.ll

```
define void @fizzbuzz(i32 %max) #0 {
entry:
  call void @check_migrate(void (i8*)* null, i8* null)
  %cmp.23 = icmp eq i32 %max, 0, !popcorn !2
  br i1 %cmp.23, label %for.end,
                 label %for.body.preheader

for.body.preheader:
  br label %for.body

for.body:
  %i.024 = phi i32 [ %inc, %for.inc ],
                   [ 0, %for.body.preheader ]
  …(if-else statement implementation)…

for.inc:
  …(increment loop induction variable)…
  br i1 %exitcond, label %for.end.loopexit,
                   label %for.body

for.end.loopexit:
  br label %for.end

for.end:
  call void @check_migrate(void (i8*)* null, i8* null)
  ret void, !popcorn !2
}
```

Inserting Stackmaps

- Need to accomplish the following:
  - Tag program locations (**all call sites**) with a unique ID
  - Record where live values at that location are stored
- Can't do either at the bitcode level!
  - No code layout – what does a program location mean at the IR level?
  - No storage allocated for live values
- Insert ***stackmap*** intrinsic functions into bitcode
  - Record program and storage locations as back-end lowers bitcode to concrete representation
  - Modified for Popcorn Linux
  - See Stack maps and patch points in LLVM for more details

Inserting Stackmaps

- Need to accomplish the following:
  - Tag program locations (**all call sites**) with a unique ID
  - Record where live values at that location are stored

Pop quiz – why **all** call sites instead of just migration points?

- Can't do either at the bitcode level:
  - No code layout – what does a program location mean at the IR level?
  - No storage allocated for live values
- Insert ***stackmap*** intrinsic functions into bitcode
  - Record program and storage locations as back-end lowers bitcode to concrete representation
  - Modified for Popcorn Linux
  - See Stack maps and patch points in LLVM for more details

Inserting Stackmaps

fizzbuzz-migpoints.ll

```
define void @fizzbuzz(i32 %max) #0 {
entry:
  call void @check_migrate(void (i8*)* null, i8* null)
  %cmp.23 = icmp eq i32 %max, 0, !popcorn !2
  br i1 %cmp.23, label %for.end,
                 label %for.body.preheader

for.body.preheader:
  br label %for.body

for.body:
  %i.024 = phi i32 [ %inc, %for.inc ],
                   [ 0, %for.body.preheader ]
  …(if-else statement implementation)…

for.inc:
  …(increment loop induction variable)…
  br i1 %exitcond, label %for.end.loopexit,
                   label %for.body

for.end.loopexit:
  br label %for.end

for.end:
  call void @check_migrate(void (i8*)* null, i8* null)
  ret void, !popcorn !2
}
```

Systems Software Research Group

VIRGINIA TECH

Inserting Stackmaps

```
opt -insert-stackmaps -S -o fizzbuzz-stackmaps.ll fizzbuzz-migpoints.ll
```

fizzbuzz-migpoints.ll

```
define void @fizzbuzz(i32 %max) #0 {
entry:
  call void @check_migrate(void (i8*)* null, i8* null)
  %cmp.23 = icmp eq i32 %max, 0, !popcorn !2
  br i1 %cmp.23, label %for.end,
                 label %for.body.preheader

for.body.preheader:
  br label %for.body

for.body:
  %i.024 = phi i32 [ %inc, %for.inc ],
                   [ 0, %for.body.preheader ]
  …(if-else statement implementation)…

for.inc:
  …(increment loop induction variable)…
  br i1 %exitcond, label %for.end.loopexit,
                   label %for.body

for.end.loopexit:
  br label %for.end

for.end:
  call void @check_migrate(void (i8*)* null, i8* null)
  ret void, !popcorn !2
}
```

Inserting Stackmaps

```
opt -insert-stackmaps -S -o fizzbuzz-stackmaps.ll fizzbuzz-migpoints.ll
```

fizzbuzz-migpoints.ll

```
define void @fizzbuzz(i32 %max) #0 {
entry:
  call void @check_migrate(void (i8*)* null, i8* null)
  %cmp.23 = icmp eq i32 %max, 0, !popcorn !2
  br i1 %cmp.23, label %for.end,
                 label %for.body.preheader

for.body.preheader:
  br label %for.body

for.body:
  %i.024 = phi i32 [ %inc, %for.inc ],
                   [ 0, %for.body.preheader ]
  …(if-else statement implementation)…

for.inc:
  …(increment loop induction variable)…
  br i1 %exitcond, label %for.end.loopexit,
                   label %for.body

for.end.loopexit:
  br label %for.end

for.end:
  call void @check_migrate(void (i8*)* null, i8* null)
  ret void, !popcorn !2
}
```

fizzbuzz-stackmaps.ll

```
define void @fizzbuzz(i32 %max) #0 {
entry:
  call void @check_migrate(void (i8*)* null, i8* null)
  call void (i64, i32, ...) @llvm.experimental.stackmap(i64 0,
                            i32 0, i32 %max)
  %cmp.23 = icmp eq i32 %max, 0, !popcorn !2
  br i1 %cmp.23, label %for.end, label %for.body.preheader

…

if.then:
  %puts22 = tail call i32 @puts(i8* getelementptr inbounds
              ([9 x i8], [9 x i8]* @str.4, i64 0, i64 0))
  call void (i64, i32, ...) @llvm.experimental.stackmap(i64 1,
                            i32 0, i32 %i.024, i32 %max)
  br label %for.inc

…

for.end:                                        ; preds =
%for.end.loopexit, %entry
  call void @check_migrate(void (i8*)* null, i8* null)
  call void (i64, i32, ...) @llvm.experimental.stackmap(i64 4,
                            i32 0)
  ret void, !popcorn !2
}
```

Inserting Stackmaps

```
call void (i64, i32, ...) @llvm.experimental.stackmap(i64 0, i32 0, i32 %max)
```

Inserting Stackmaps

Unique call-site ID – made uniqued across all compiled files during post-processing (only have global view at link-time)

```
call void (i64, i32, ...) @llvm.experimental.stackmap(i64 0, i32 0, i32 %max)
```

Systems Software Research Group

Virginia Tech

Inserting Stackmaps

Shadow bytes – unused by Popcorn

```
call void (i64, i32, ...) @llvm.experimental.stackmap(i64 0, i32 0, i32 %max)
```

Inserting Stackmaps

List of live values at this program location

```
call void (i64, i32, ...) @llvm.experimental.stackmap(i64 0, i32 0, i32 %max)
```

Inserting Stackmaps

```
call void (i64, i32, ...) @llvm.experimental.stackmap(i64 0, i32 0, i32 %max)
```

We **create** equivalence points by inserting stackmaps into the bitcode

- Tags equivalent program locations across all compilations, as the **same IR** is lowered through multiple target-specific back-ends
- Lists **all live values** at equivalence point – back-ends are simply responsible for recording where live values are located
- Back-ends must **not** optimize across stackmaps as this violates these invariants (see slide notes)

Generating Stack Metadata

fizzbuzz-stackmaps.ll

```
define void @fizzbuzz(i32 %max) #0 {
entry:
  call void @check_migrate(void (i8*)* null, i8* null)
  call void (i64, i32, ...) @llvm.experimental.stackmap(i64 0,
                             i32 0, i32 %max)
  %cmp.23 = icmp eq i32 %max, 0, !popcorn !2
  br i1 %cmp.23, label %for.end, label %for.body.preheader

…

if.then:
  %puts22 = tail call i32 @puts(i8* getelementptr inbounds
             ([9 x i8], [9 x i8]* @str.4, i64 0, i64 0))
  call void (i64, i32, ...) @llvm.experimental.stackmap(i64 1,
                             i32 0, i32 %i.024, i32 %max)
  br label %for.inc

…

for.end:                                         ; preds =
%for.end.loopexit, %entry
  call void @check_migrate(void (i8*)* null, i8* null)
  call void (i64, i32, ...) @llvm.experimental.stackmap(i64 4,
                             i32 0)
  ret void, !popcorn !2
}
```

Generating Stack Metadata

```
clang -c -mllvm -optimize-regalloc -o fizzbuzz.o fizzbuzz-stackmaps.ll
```

fizzbuzz-stackmaps.ll

```
define void @fizzbuzz(i32 %max) #0 {
entry:
  call void @check_migrate(void (i8*)* null, i8* null)
  call void (i64, i32, ...) @llvm.experimental.stackmap(i64 0,
                          i32 0, i32 %max)
  %cmp.23 = icmp eq i32 %max, 0, !popcorn !2
  br i1 %cmp.23, label %for.end, label %for.body.preheader

…

if.then:
  %puts22 = tail call i32 @puts(i8* getelementptr inbounds
              ([9 x i8], [9 x i8]* @str.4, i64 0, i64 0))
  call void (i64, i32, ...) @llvm.experimental.stackmap(i64 1,
                          i32 0, i32 %i.024, i32 %max)
  br label %for.inc

…

for.end:                                          ; preds =
%for.end.loopexit, %entry
  call void @check_migrate(void (i8*)* null, i8* null)
  call void (i64, i32, ...) @llvm.experimental.stackmap(i64 4,
                          i32 0)
  ret void, !popcorn !2
}
```

Generating Stack Metadata

```
clang -c -mllvm -optimize-regalloc -o fizzbuzz.o fizzbuzz-stackmaps.ll
```

fizzbuzz-stackmaps

- `-mllvm`: Pass option directly to LLVM's middle-/back-end
- `-optimize-regalloc`: use an optimizing register allocator (versus LLVM's fast allocator), which calculates live value ranges in the backend; required by Popcorn's analyses

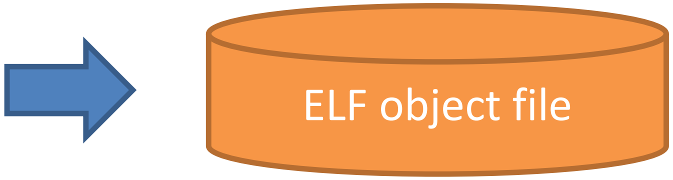
```
define void @fizzbuzz(
entry:
  call void @check_mig
  call void (i64, i32,

  %cmp.23 = icmp eq i3
  br i1 %cmp.23, label

…

if.then:
  %puts22 = tail call i32 @puts(i8* getelementptr inbounds
            ([9 x i8], [9 x i8]* @str.4, i64 0, i64 0))
  call void (i64, i32, ...) @llvm.experimental.stackmap(i64 1,
                            i32 0, i32 %i.024, i32 %max)
  br label %for.inc

…

for.end:                                          ; preds =
%for.end.loopexit, %entry
  call void @check_migrate(void (i8*)* null, i8* null)
  call void (i64, i32, ...) @llvm.experimental.stackmap(i64 4,
                            i32 0)
  ret void, !popcorn !2
}
```

Generating Stack Metadata

`clang -c -mllvm -optimize-regalloc -o fizzbuzz.o fizzbuzz-stackmaps.ll`

fizzbuzz-stackmaps.ll

```
define void @fizzbuzz(i32 %max) #0 {
entry:
  call void @check_migrate(void (i8*)* null, i8* null)
  call void (i64, i32, ...) @llvm.experimental.stackmap(i64 0,
                            i32 0, i32 %max)
  %cmp.23 = icmp eq i32 %max, 0, !popcorn !2
  br i1 %cmp.23, label %for.end, label %for.body.preheader

…

if.then:
  %puts22 = tail call i32 @puts(i8* getelementptr inbounds
              ([9 x i8], [9 x i8]* @str.4, i64 0, i64 0))
  call void (i64, i32, ...) @llvm.experimental.stackmap(i64 1,
                            i32 0, i32 %i.024, i32 %max)
  br label %for.inc

…

for.end:                                             ; preds =
%for.end.loopexit, %entry
  call void @check_migrate(void (i8*)* null, i8* null)
  call void (i64, i32, ...) @llvm.experimental.stackmap(i64 4,
                            i32 0)
  ret void, !popcorn !2
}
```

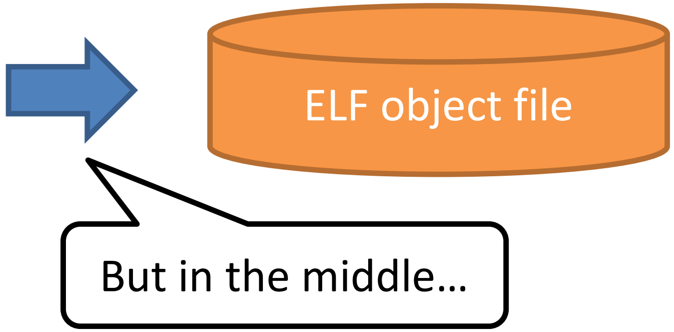


Generating Stack Metadata

```
clang -c -mllvm -optimize-regalloc -o fizzbuzz.o fizzbuzz-stackmaps.ll
```

fizzbuzz-stackmaps.ll

```
define void @fizzbuzz(i32 %max) #0 {
entry:
  call void @check_migrate(void (i8*)* null, i8* null)
  call void (i64, i32, ...) @llvm.experimental.stackmap(i64 0,
                            i32 0, i32 %max)
  %cmp.23 = icmp eq i32 %max, 0, !popcorn !2
  br i1 %cmp.23, label %for.end, label %for.body.preheader

…

if.then:
  %puts22 = tail call i32 @puts(i8* getelementptr inbounds
              ([9 x i8], [9 x i8]* @str.4, i64 0, i64 0))
  call void (i64, i32, ...) @llvm.experimental.stackmap(i64 1,
                            i32 0, i32 %i.024, i32 %max)
  br label %for.inc

…

for.end:                                          ; preds =
%for.end.loopexit, %entry
  call void @check_migrate(void (i8*)* null, i8* null)
  call void (i64, i32, ...) @llvm.experimental.stackmap(i64 4,
                            i32 0)
  ret void, !popcorn !2
}
```



Generating Stack Metadata

```
clang -c -mllvm -optimize-regalloc -o fizzbuzz.o fizzbuzz-stackmaps.ll -mllvm -debug-only=regalloc
```

```
********** MACHINEINSTRS **********
# Machine code for function fizzbuzz: Post SSA
Function Live Ins: %EDI in %vreg4

0B          BB#0: derived from LLVM BB %entry
                Live Ins: %EDI
16B                   %vreg5<def> = COPY %EDI; GR32:%vreg5
80B                   ADJCALLSTACKDOWN64 0, 0, %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
96B                   %EDI<def,dead> = MOV32r0 %EFLAGS<imp-def,dead>, %RDI<imp-def>
112B                  %ESI<def,dead> = MOV32r0 %EFLAGS<imp-def,dead>, %RSI<imp-def>
128B                  CALL64pcrel32 <ga:@check_migrate>, <regmask>, %RSP<imp-use>, %RDI<imp-use>,
                                                              %RSI<imp-use,kill>
144B                  ADJCALLSTACKUP64 0, 0, %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
160B                  ADJCALLSTACKDOWN64 0, %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
176B                  STACKMAP 0, 0, %vreg5, ...; GR32:%vreg5
192B                  ADJCALLSTACKUP64 0, 0, %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
208B                  CMP32ri8 %vreg5, 0, %EFLAGS<imp-def>; GR32:%vreg5
224B                  JE_1 <BB#10>, %EFLAGS<imp-use,kill>
              Successors according to CFG: BB#10 BB#1
```

Generating Stack Metadata

```
clang -c -mllvm -optimize-regalloc -o fizzbuzz.o fizzbuzz-stackmaps.ll -mllvm -debug-only=regalloc
```

```
********** MACHINEINSTRS **********
# Machine code for function fizzbuzz: Post SSA
Function Live Ins: %EDI in

0B          BB#0: derived
                Live Ins:
16B                 %vreg
80B                 ADJCA                                                      , %RSP<imp-use>
96B                 %EDI<
112B                %ESI<
128B                CALL6                                                      %RDI<imp-use>,
                                                                               kill>
144B                ADJCALLSTACK         , %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
160B                ADJCALLSTA      OWN64 0, %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
176B                STACKMAP 0, 0, %vreg5, ...; GR32:%vreg5
192B                ADJCALLSTACKUP64 0, 0, %RSP<imp-def>, %EFLAGS<imp-def,dead>, %RSP<imp-use>
208B                CMP32ri8 %vreg5, 0, %EFLAGS<imp-def>; GR32:%vreg5
224B                JE_1 <BB#10>, %EFLAGS<imp-use,kill>
                Successors according to CFG: BB#10 BB#1
```

Bitcode stackmaps are lowered to machine code IR stackmaps
- First 2 arguments unchanged (ID, shadow bytes)
- Bitcode values lowered to machine code operands depending on where they're allocated (virtual registers, constants, stack slots)

Generating Stack Metadata

```
clang -c -mllvm -optimize-regalloc -o fizzbuzz.o fizzbuzz-stackmaps.ll -mllvm -debug-only=stacktransform
```

```
********** STACK TRANSFORMATION METADATA **********
********** Function: fizzbuzz
********** REGISTER MAP **********
[%vreg5 -> %R14D] GR32
[%vreg11 -> %EDX] GR32
[%vreg12 -> %AL] GR8
...

*** Stack slot copies ***

Stackmap 0:
  STACKMAP 0, 0, %vreg5, ...; GR32:%vreg5

i32 %max: in register %R14D (vreg 5)

Duplicate operand locations:

...
*** Finding architecture-specific live values ***

  STACKMAP 0, 0, %vreg5, ...; GR32:%vreg5
  -> Call instruction SlotIndex 128B, searching vregs 0 -> 31 and stack slots 0 -> 0
```

Systems Software Research Group

VIRGINIA TECH

Generating Stack Metadata

```
clang -c -mllvm -optimize-regalloc -o fizzbuzz.o fizzbuzz-stackmaps.ll -mllvm -debug-only=stacktransform
```

```
********** STACK TRANSFORMATION METADATA **********
********** Function: fizzbuzz
********** REGISTER MAP **********
[%vreg5 -> %R14D] GR32
[%vreg11 -> %EDX] GR32
[%vreg12 -> %AL] GR8
...

*** Stack slot copies ***

Stackmap 0:
  STACKMAP 0, 0, %vreg5, ...; GR32:%vreg5

i32 %max: in register %R14D (vreg 5)

Duplicate operand locations:

...
*** Finding architecture-specific live values ***

  STACKMAP 0, 0, %vreg5, ...; GR32:%vreg5
  -> Call instruction SlotIndex 128B, searching vregs 0 -> 31 and stack slots 0 -> 0
```

- Metadata emitted by vanilla stackmaps is not complete
  - Intended to allow capturing enough live state to jump to optimized implementation (e.g., hot-patching optimized code in virtual machine)
  - Need to augment with complete frame information
  - See `StackTransformMetadata.cpp` for more details

Generating Stack Metadata

```
clang -c -mllvm -optimize-regalloc -o fizzbuzz.o fizzbuzz-stackmaps.ll -mllvm -debug-only=stacktransform
```

```
********** STACK TRANSFORMATION METADATA **********
********** Function: fizzbuzz
********** REGISTER MAP **********
[%vreg5 -> %R14D] GR32
[%vreg11 -> %EDX] GR32
[%vreg12 -> %AL] GR8
...

*** Stack slot copies ***

Stackmap 0:
  STACKMAP 0, 0, %vreg5, ...; GR32:%vreg5

i32 %max: in register %R14D (vreg 5)

Duplicate operand locations:

...
*** Finding architecture-specific live values ***

  STACKMAP 0, 0, %vreg5, ...; GR32:%vreg5
  -> Call instruction SlotIndex 128B, searching vregs 0 -> 31 and stack slots 0 -> 0
```

Register allocation results

Systems Software Research Group

VIRGINIA TECH

Generating Stack Metadata

```
clang -c -mllvm -optimize-regalloc -o fizzbuzz.o fizzbuzz-stackmaps.ll -mllvm -debug-only=stacktransform

********** STACK TRANSFORMATION METADATA **********
********** Function: fizzbuzz
********** REGISTER MAP **********
[%vreg5 -> %R14D] GR32
[%vreg11 -> %EDX] GR32
[%vreg12 -> %AL] GR8
...

*** Stack slot copies ***

Stackmap 0:
  STACKMAP 0, 0, %vreg5, ...; GR32:%vreg5

i32 %max: in register %R14D (vreg 5)

Duplicate operand locations:

...
*** Finding architecture-specific live values ***

  STACKMAP 0, 0, %vreg5, ...; GR32:%vreg5
  -> Call instruction SlotIndex 128B, searching vregs 0 -> 31 and stack slots 0 -> 0
```

Mapping of IR values -> machine-code values
- Value *may* be live in multiple locations

Generating Stack Metadata

```
clang -c -mllvm -optimize-regalloc -o fizzbuzz.o fizzbuzz-stackmaps.ll -mllvm -debug-only=stacktransform
```

```
********** STACK TRANSFORMATION METADATA **********
********** Function: fizzbuzz
********** REGISTER MAP **********
[%vreg5 -> %R14D] GR32
[%vreg11 -> %EDX] GR32
[%vreg12 -> %AL] GR8
...

*** Stack slot copies ***

Stackmap 0:
  STACKMAP 0, 0, %vreg5, ...; GR32:%vreg5

i32 %max: in register %R14D (vreg 5)

Duplicate operand locations:

...
*** Finding architecture-specific live values ***

  STACKMAP 0, 0, %vreg5, ...; GR32:%vreg5
  -> Call instruction SlotIndex 128B, searching vregs 0 -> 31 and stack slots 0 -> 0
```

Handle architecture-specific live values that may arise due to backend-optimizations or ABI

Generating Stack Metadata

```
clang -c -mllvm -optimize-regalloc -o fizzbuzz.o fizzbuzz-stackmaps.ll -mllvm -debug-only=stacktransform
```

```
********** STACK TRANSFORMATION METADATA **********
********** Function: fizzbuzz
********** REGISTER MAP **********
[%vreg5 -> %R14D] GR32
[%vreg11 -> %EDX] GR32
[%vreg12 -> %AL] GR8
...

*** Stack slot copies ***

Stackmap 0:
  STACKMAP 0, 0, %vreg5, ...; GR32:%vreg5

i32 %max: in register %R14D (vreg 5)

Duplicate operand locations:

...
*** Finding architecture-specific li

  STACKMAP 0, 0, %vreg5, ...; GR32:%vreg5
  -> Call instruction SlotIndex 128B, searching vregs 0 -> 31 and stack slots 0 -> 0
```

SlotIndex: instruction location in machine code IR

Generating Stack Metadata

```
clang -c -mllvm -optimize-regalloc -o fizzbuzz.o fizzbuzz-stackmaps.ll -mllvm -debug-only=stacktransform
```

```
********** STACK TRANSFORMATION METADATA **********
********** Function: fizzbuzz
********** REGISTER MAP **********
[%vreg5 -> %R14D] GR32
[%vreg11 -> %EDX] GR32
[%vreg12 -> %AL] GR8
...

*** Stack slot copies ***

Stackmap 0:
  STACKMAP 0, 0, %vreg5, ...; GR32:%vreg5

i32 %max: in regis

Duplicate operand

...
*** Finding archit

  STACKMAP 0, 0, %vreg5, ...; GR32:%vreg5
  -> Call instruction SlotIndex 128B, searching vregs 0 -> 31 and stack slots 0 -> 0
```

Looking for registers/stack slots live across stackmap but not contained in stackmap

- Back-end will issue warning if it finds live value it can't handle during transformation – **pay attention to these warnings**!

Generating Stack Metadata

readelf –SW fizzbuzz.o

```
There are 17 section headers, starting at offset 0x5e0:

Section Headers:
  [Nr] Name                Type            Address          Off    Size   ES Flg Lk Inf Al
  [ 0]                     NULL            0000000000000000 000000 000000 00      0   0  0
  [ 1] .strtab             STRTAB          0000000000000000 0004b0 00012f 00      0   0  1
  [ 2] .text               PROGBITS        0000000000000000 000040 0000b9 00  AX  0   0 16
  [ 3] .rela.text          RELA            0000000000000000 0003a8 0000c0 18     16   2  8
  [ 4] .data               PROGBITS        0000000000000000 0000fc 000000 00  WA  0   0  4
  [ 5] .bss                NOBITS          0000000000000000 0000fc 000000 00  WA  0   0  4
  [ 6] .rodata.str1.1      PROGBITS        0000000000000000 0000fc 000013 01 AMS  0   0  1
  [ 7] .comment            PROGBITS        0000000000000000 00010f 000035 01  MS  0   0  1
  [ 8] .note.GNU-stack     PROGBITS        0000000000000000 000144 000000 00      0   0  1
  [ 9] .stack_transform.unwind PROGBITS        0000000000000000 000144 000010 04      0   0  4
  [10] .stack_transform.unwind_arange PROGBITS        0000000000000000 000158 000010 10      0   0  8
  [11] .rela.stack_transform.unwind_arange RELA            0000000000000000 000468 000018 18     16  10  8
  [12] .llvm_stackmaps     PROGBITS        0000000000000000 000168 000118 00   A  0   0  8
  [13] .rela.llvm_stackmaps RELA            0000000000000000 000480 000018 18     16  12  8
  [14] .eh_frame           PROGBITS        0000000000000000 000280 000038 00   A  0   0  8
  [15] .rela.eh_frame      RELA            0000000000000000 000498 000018 18     16  14  8
  [16] .symtab             SYMTAB          0000000000000000 0002b8 0000f0 18      1   7  8
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), l (large)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

Systems Software Research Group

VIRGINIA TECH

Generating Stack Metadata

`readelf -SW fizzbuzz.o`

```
There are 17 section headers, starting at offset 0x5e0:

Section Headers:
  [Nr] Name              Type            Address          Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            0000000000000000 000000 000000 00      0   0  0
  [ 1] .strtab           STRTAB                                                    0  1
  [ 2] .text             PROGBITS                                                  0 16
  [ 3] .rela.text        RELA                                                      2  8
  [ 4] .data             PROGBITS                                                  0  4
  [ 5] .bss              NOBITS                                                    0  4
  [ 6] .rodata.str1.1    PROGBITS                                                  0  1
  [ 7] .comment          PROGBITS        0000             00010f 000035 01  MS  0   0  1
  [ 8] .note.GNU-stack   PROGBITS        000000000000000 000144 000000 00      0   0  1
  [ 9] .stack_transform.unwind PROGBITS  0000000000000000 000144 000010 04      0   0  4
  [10] .stack_transform.unwind_arange PROGBITS      0000000000000000 000158 000010 10      0   0  8
  [11] .rela.stack_transform.unwind_arange RELA         0000000000000000 000468 000018 18     16  10  8
  [12] .llvm_stackmaps   PROGBITS        0000000000000000 000168 000118 00   A  0   0  8
  [13] .rela.llvm_stackmaps RELA            0000000000000000 000480 000018 18     16  12  8
  [14] .eh_frame         PROGBITS        0000000000000000 000280 000038 00   A  0   0  8
  [15] .rela.eh_frame    RELA            0000000000000000 000498 000018 18     16  14  8
  [16] .symtab           SYMTAB          0000000000000000 0002b8 0000f0 18      1   7  8
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), l (large)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

Metadata describing callee-save information for functions

- Unwind frames from the stack

Generating Stack Metadata

readelf -SW fizzbuzz.o

```
There are 17 section headers, starting at offset 0x5e0:

Section Headers:
  [Nr] Name                  Type            Address          Off    Size   ES Flg Lk Inf Al
  [ 0]                       NULL            0000000000000000 000000 000000 00      0   0  0
  [ 1] .strtab               STRTAB          0000000000000000 0004b0 00012f 00      0   0  1
  [ 2] .text                 PROGBITS        0000000000000000 000040 0000b9 00  AX  0   0 16
  [ 3] .rela.text            RELA            0000000000000000 0003a8 0000c0 18     16   2  8
  [ 4] .data                 PROGBI                                                     0  4
  [ 5] .bss                  NOBITS                                                     0  4
  [ 6] .rodata.str1.1        PROGBI                                                     0  1
  [ 7] .comment              PROGBI                                                     0  1
  [ 8] .note.GNU-stack       PROGBI                                                     0  1
  [ 9] .stack_transform.unwind P                                                    0   0  4
  [10] .stack_transform.unwind_arange PROGBITS    0000000000000000 000158 000010 10      0   0  8
  [11] .rela.stack_transform.unwind_arange RELA        0000000000000000 000468 000018 18     16  10  8
  [12] .llvm_stackmaps       PROGBITS        0000000000000000 000168 000118 00   A  0   0  8
  [13] .rela.llvm_stackmaps RELA            0000000000000000 000480 000018 18     16  12  8
  [14] .eh_frame             PROGBITS        0000000000000000 000280 000038 00   A  0   0  8
  [15] .rela.eh_frame        RELA            0000000000000000 000498 000018 18     16  14  8
  [16] .symtab               SYMTAB          0000000000000000 0002b8 0000f0 18      1   7  8
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), l (large)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

Generating Stack Metadata

```
dump-llvm-stackmap -f fizzbuzz.o
```

```
Reading section .llvm_stackmaps: Found 1 stackmaps
Stackmap v1: 1 functions, 0 constants, 5 call sites
  Function 0: address=0, stack size=24, number of unwinding entries: 4, offset into unwinding
section: 0
  Call site 0: function 0, offset @ 19, 1 locations, 0 live-outs, 0 arch-specific locations
    Location: in register 14
  Call site 1: function 0, offset @ 92, 2 locations, 0 live-outs, 0 arch-specific locations
    Location: in register 3
    Location: in register 14
  Call site 2: function 0, offset @ 121, 2 locations, 0 live-outs, 0 arch-specific locations
    Location: in register 3
    Location: in register 14
  Call site 3: function 0, offset @ 154, 2 locations, 0 live-outs, 0 arch-specific locations
    Location: in register 3
    Location: in register 14
  Call site 4: function 0, offset @ 180, 0 locations, 0 live-outs, 0 arch-specific locations
```

Generating Stack Metadata

`dump-llvm-stackmap -f fizzbuzz.o`

```
Reading section .llvm_stackmaps: Found 1 stackmaps
Stackmap v1: 1 functions, 0 constants, 5 call sites
  Function 0: address=0, stack size=24, number of unwinding entries: 4, offset into unwinding
section: 0
  Call site 0: function 0                     locations, 0 live-outs, 0 arch-specific locations
    Location: in reg
  Call site 1: funct                                                     ecific locations
    Location: in reg
    Location: in register 14
  Call site 2: function 0, offset @ 121, 2 locations, 0 live-outs, 0 arch-specific locations
    Location: in register 3
    Location: in register 14
  Call site 3: function 0, offset @ 154, 2 locations, 0 live-outs, 0 arch-specific locations
    Location: in register 3
    Location: in register 14
  Call site 4: function 0, offset @ 180, 0 locations, 0 live-outs, 0 arch-specific locations
```

Number of functions for which we have metadata,
number of stackmap call sites across all functions

Generating Stack Metadata

`dump-llvm-stackmap –f fizzbuzz.o`

```
Reading section .llvm_stackmaps: Found 1 stackmaps
Stackmap v1: 1 functions, 0 constants, 5 call sites
  Function 0: address=0, stack size=24, number of unwinding entries: 4, offset into unwinding
section: 0
  Call site 0: function 0,     t @ 19, 1 locations, 0 live-outs, 0 arch-specific locations
    Location: in register
  Call site 1: fun                                                  specific locations
    Location: in r
    Location: in r
  Call site 2: function 0, offset @ 121, 2 locations, 0 live-outs, 0 arch-specific locations
    Location: in register 3
    Location: in register 14
  Call site 3: function 0, offset @ 154, 2 locations, 0 live-outs, 0 arch-specific locations
    Location: in register 3
    Location: in register 14
  Call site 4: function 0, offset @ 180, 0 locations, 0 live-outs, 0 arch-specific locations
```

Per-function metadata describing frame size and how to unwind the frame to the caller

Generating Stack Metadata

```
dump-llvm-stackmap -f fizzbuzz.o
```

```
Reading section .llvm_stackmaps: Found 1 stackmaps
Stackmap v1: 1 functions, 0 constants, 5 call sites
  Function 0: address=0, stack size=24, number of unwinding entries: 4, offset into unwinding
section: 0
  Call site 0: function 0, offset @ 19, 1 locations, 0 live-outs, 0 arch-specific locations
    Location: in register 14
  Call site 1: function 0, offset @ 92, 2 locations, 0 live-outs, 0 arch-specific locations
    Location: in register
    Location: in register 14
  Call site 2: fun                                                         ific locations
    Location: in r
    Location: in r
  Call site 3: function 0, offset @ 154, 2 locations, 0 live-outs, 0 arch-specific locations
    Location: in register 3
    Location: in register 14
  Call site 4: function 0, offset @ 180, 0 locations, 0 live-outs, 0 arch-specific locations
```

Per-call site information describing program location & live values (one per stackmap inserted into bitcode)

Systems Software Research Group

VIRGINIA TECH

Putting It All Together

```
$ clang -O2 -popcorn-migratable -c fizzbuzz.c
$ ls
fizzbuzz.c  fuzzbuzz.o  fizzbuzz_x86_64.o
$ file fizzbuzz.o
fizzbuzz.o: ELF 64-bit LSB relocatable, ARM aarch64, version 1 (GNU/Linux), not stripped
$ file fizzbuzz_x86_64.o
fizzbuzz_x86_64.o: ELF 64-bit LSB relocatable, x86-64, version 1 (GNU/Linux), not stripped
```

Putting It All Together

- Insert migration library call-outs, collect stack transformation metadata at equivalence points
- Generate object files for **all** supported architectures simultaneously (requires `-c`)

```
$ clang -O2 -popcorn-migratable -c fizzbuzz.c
$ ls
fizzbuzz.c  fuzzbuzz.o  fizzbuzz_x86_64.o
$ file fizzbuzz.o
fizzbuzz.o: ELF 64-bit LSB relocatable, ARM aarch64, version 1 (GNU/Linux), not stripped
$ file fizzbuzz_x86_64.o
fizzbuzz_x86_64.o: ELF 64-bit LSB relocatable, x86-64, version 1 (GNU/Linux), not stripped
```

Compilation Recap

- Inserts call-outs to migration library
- Constructs equivalence points by inserting stackmaps into LLVM bitcode
  - Tags program locations across compilations for all targets
  - Captures locations of all live values at equivalence points
- Generates single set of optimized LLVM bitcode and lowers it through multiple target-specific backends

Part 3:
Linking & post-processing a Popcorn compilation

Part 3:
Linking & post-processing a Popcorn compilation

All source & log files are available in the "het-link" folder

Link-Time Alignment

- The compiler has taken care of generating metadata required for stack/register transformation
- Still need to align global objects
  - Statically-allocated global data
  - Code, i.e., functions
  - We **do not** need to worry about dynamically-allocated data (heap)
    - Memory allocator (e.g., `malloc`) is responsible for arranging data in heap
    - Use semantically equivalent memory allocator (musl) on all architectures

Link-Time Alignment

- Linux uses the ***Executable and Linkable Format*** (ELF)
  - Data sections (statically allocated):
    - `.rodata` – read-only global data initialized within the program
    - `.data` – readable/writable global data initialized within the program
    - `.bss` – readable/writable global data **not** initialized within the program
      - Linux initializes to zero
  - Code section:
    - `.text` – ISA-specific machine code generated by the compiler
  - Miscellaneous – symbol/string tables, constructors/destructors , debugging information
  - Popcorn's compiler adds metadata sections required for stack transformation

Link-Time Alignment

- Use ***linker scripts*** to align symbols across all compilations
  - Program objects referenced by symbol – requires **all** program objects have a symbol attached, including string literals
    - See `NameStringLiterals.cpp` in repo for details
  - After generating "vanilla" (read: unaligned) version of binary for each architecture, generate linker script & re-link
  - See the documentation on linker scripts for more details

Link-Time Alignment

- Linker scripts can't control placement of symbols, only ELF sections
  - Luckily, ELF format permits arbitrary numbers of sections in object files
  - Solution: place each program object in its own section
    - Requires `-ffunction-sections` & `-fdata-sections` (included automatically with `-popcorn-migratable`)

Link-Time Alignment

fizzbuzz.c

```c
#include <stdio.h>

void fizzbuzz(unsigned max)
{
  unsigned i;
  for(i = 0; i < max; i++)
  {
    if((i % 5) == 0 && (i % 3) == 0)
      printf("fizzbuzz\n");
    else if((i % 5) == 0)
      printf("fizz\n");
    else if((i % 3) == 0)
      printf("buzz\n");
  }
}
```

Link-Time Alignment

```
clang -O2 -popcorn-migratable -c fizzbuzz.c
```

fizzbuzz.c

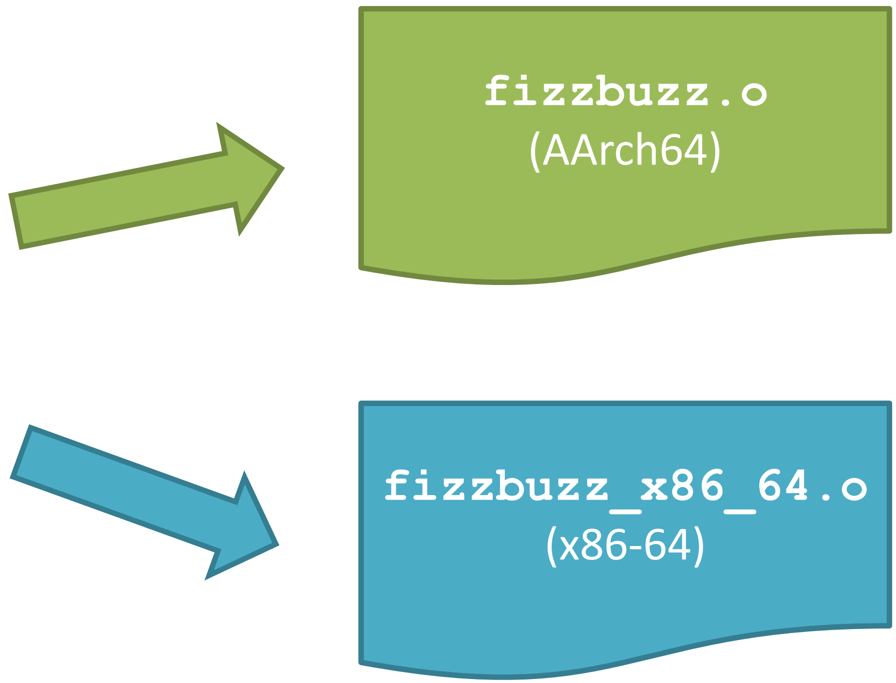
```
#include <stdio.h>

void fizzbuzz(unsigned max)
{
  unsigned i;
  for(i = 0; i < max; i++)
  {
    if((i % 5) == 0 && (i % 3) == 0)
      printf("fizzbuzz\n");
    else if((i % 5) == 0)
      printf("fizz\n");
    else if((i % 3) == 0)
      printf("buzz\n");
  }
}
```

Link-Time Alignment

```
clang -O2 -popcorn-migratable -c fizzbuzz.c
```

fizzbuzz.c

```c
#include <stdio.h>

void fizzbuzz(unsigned max)
{
  unsigned i;
  for(i = 0; i < max; i++)
  {
    if((i % 5) == 0 && (i % 3) == 0)
      printf("fizzbuzz\n");
    else if((i % 5) == 0)
      printf("fizz\n");
    else if((i % 3) == 0)
      printf("buzz\n");
  }
}
```



Link-Time Alignment

fizzbuzz.o
(AArch64)

readelf -SW fizzbuzz.o

```
There are 20 section headers, starting at offset 0x7f0:

Section Headers:
  [Nr] Name                 Type            Address          Off    Size   ES Flg Lk Inf Al
  [ 0]                      NULL            0000000000000000 000000 000000 00      0   0  0
  [ 1] .strtab              STRTAB          0000000000000000 000640 0001a9 00      0   0  1
  [ 2] .text                PROGBITS        0000000000000000 000040 000000 00  AX  0   0  4
  [ 3] .data                PROGBITS        0000000000000000 000040 000000 00  WA  0   0  4
  [ 4] .bss                 NOBITS          0000000000000000 000040 000000 00  WA  0   0  4
  [ 5] .text.fizzbuzz       PROGBITS        0000000000000000 000040 0000fc 00  AX  0   0  4
  [ 6] .rela.text.fizzbuzz RELA            0000000000000000 0004f0 000108 18     19   5  8
  [ 7] .rodata.fizzbuzz__str_fizzbuzz__ PROGBITS        0000000000000000 00013c 00000a 00   A  0   0  1
  [ 8] .rodata.fizzbuzz__str_1_fizz__ PROGBITS        0000000000000000 000146 000006 00   A  0   0  1
  [ 9] .rodata.fizzbuzz__str_2_buzz__ PROGBITS        0000000000000000 00014c 000006 00   A  0   0  1
  [10] .comment             PROGBITS        0000000000000000 000152 000035 01  MS  0   0  1
  [11] .note.GNU-stack      PROGBITS        0000000000000000 000187 000000 00      0   0  1
  [12] .stack_transform.unwind PROGBITS        0000000000000000 000188 000010 04      0   0  4
  [13] .stack_transform.unwind_arange PROGBITS        0000000000000000 000198 000010 10      0   0  8
  [14] .rela.stack_transform.unwind_arange RELA            0000000000000000 0005f8 000018 18     19  13  8
  [15] .llvm_stackmaps      PROGBITS        0000000000000000 0001a8 000118 00   A  0   0  8
  [16] .rela.llvm_stackmaps RELA            0000000000000000 000610 000018 18     19  15  8
  [17] .eh_frame            PROGBITS        0000000000000000 0002c0 000038 00   A  0   0  8
  [18] .rela.eh_frame       RELA            0000000000000000 000628 000018 18     19  17  8
  [19] .symtab              SYMTAB          0000000000000000 0002f8 0001f8 18      1  15  8
```

Link-Time Alignment

```
readelf -SW fizzbuzz.o
```



```
There are 20 section headers, starti

Section Headers:
  [Nr] Name              Type
  [ 0]                   NULL
  [ 1] .strtab           STRTAB
  [ 2] .text             PROGBITS
  [ 3] .data             PROGBITS
  [ 4] .bss              NOBITS
  [ 5] .text.fizzbuzz    PROGBITS        0000000000000000 000040 0000fc 00  AX  0   0  4
  [ 6] .rela.text.fizzbuzz RELA          0000000000000000 0004f0 000108 18      19   5  8
  [ 7] .rodata.fizzbuzz__str_fizzbuzz__ PROGBITS        0000000000000000 00013c 00000a 00   A  0   0  1
  [ 8] .rodata.fizzbuzz__str_1_fizz__ PROGBITS        0000000000000000 000146 000006 00   A  0   0  1
  [ 9] .rodata.fizzbuzz__str_2_buzz__ PROGBITS        0000000000000000 00014c 000006 00   A  0   0  1
  [10] .comment          PROGBITS        0000000000000000 000152 000035 01  MS  0   0  1
  [11] .note.GNU-stack   PROGBITS        0000000000000000 000187 000000 00      0   0  1
  [12] .stack_transform.unwind PROGBITS        0000000000000000 000188 000010 04      0   0  4
  [13] .stack_transform.unwind_arange PROGBITS        0000000000000000 000198 000010 10      0   0  8
  [14] .rela.stack_transform.unwind_arange RELA            0000000000000000 0005f8 000018 18      19  13  8
  [15] .llvm_stackmaps   PROGBITS        0000000000000000 0001a8 000118 00   A  0   0  8
  [16] .rela.llvm_stackmaps RELA            0000000000000000 000610 000018 18      19  15  8
  [17] .eh_frame         PROGBITS        0000000000000000 0002c0 000038 00   A  0   0  8
  [18] .rela.eh_frame    RELA            0000000000000000 000628 000018 18      19  17  8
  [19] .symtab           SYMTAB          0000000000000000 0002f8 0001f8 18       1  15  8
```

Each program object is placed into its own section & prepended with the section name corresponding to that object type (e.g., `.text` for code)

Link-Time Alignment

- We use a modified linker based on GNU's `gold` and a python tool `pyalign` for alignment
- First linking pass
  - Link vanilla (unaligned) version of binary for each target
  - Dump section names, including sizes & alignments, into a ***map file***
- Alignment
  - Parse binaries/map file and generate linker script for each target
- Second linking pass
  - Link heterogeneous (aligned) version of binary for each target using linker scripts generated by `pyalign`

Link-Time Alignment

```
$ ld.gold -L/usr/lib/gcc-cross/aarch64-linux-gnu/5 -Map map_aarch64.txt \
    main.o fizzbuzz.o –o fizzbuzz_aarch64
    <install>/aarch64/lib/crt1.o <install>/aarch64/lib/libc.a \
    <install>/aarch64/lib/libmigrate.a <install>/aarch64/lib/libstack-transform.a \
    <install>/aarch64/lib/libelf.a <install>/aarch64/lib/libc.a \
    --start-group -lgcc -lgcc_eh -end-goup
```

Link-Time Alignment

```
$ ld.gold -L/usr/lib/gcc-cross/aarch64-linux-gnu/5 -Map map_aarch64.txt \
    main.o fizzbuzz.o -o fizzbuzz_aarch64
    <install>/aarch64/lib/crt1.o <install>/aarch64/     ibc.a \
    <install>/aarch64/lib/libmigrate.a <insta        4/lib/libstack-transform.a \
    <insta                                                   \
    --sta
```

Generate map file named `map_aarch64.txt`

Link-Time Alignment

```
$ ld.gold -L/usr/lib/gcc-cross/aarch64-linux-gnu/5 -Map map_aarch64.txt \
    main.o fizzbuzz.o -o fizzbuzz_aarch64
    <install>/aarch64/lib/crt1.o <install>/aarch64/lib/libc.a \
    <install>/aarch64/lib/libmigrate.a <install>/aarch64/lib/libstack-transform.a \
    <install>/aarch64/lib/libelf.a <install>/aarch64/lib/libc.a \
    --start-group -lgcc -lgcc_eh -end-goup
```

map_aarch64.txt

```
Archive member included because of file (symbol)
…
Memory map

 ** file header
                0x0000000000400000       0x40
 ** segment headers
                0x0000000000400040       0xe0

.text           0x0000000000400120    0x21744
 .text.exit     0x0000000000400120       0x24 0x4 /home/rlyerly/Downloads/popcorn-compiler/test-
install/aarch64/lib/libc.a(exit.o)
                0x0000000000400120                exit
 .text          0x0000000000400144        0x0 0x4 main.o
 .text.main     0x0000000000400144       0x7c 0x4 main.o
                0x0000000000400144                main
…
```

Systems Software Research Group

VIRGINIA TECH

Link-Time Alignment

```
$ ld.gold -L/usr/lib/gcc-cross/aarch64-linux-gnu/5 -Map map_aarch64.txt \
    main.o fizzbuzz.o -o fizzbuzz_aarch64
    <install>/aarch64/lib/crt1.o <install>/aarch64/lib/libc.a \
    <install>/aarch64/lib/libmigrate.a <install>/aarch64/lib/libstack-transform.a \
    <install>/aarch64/lib/libelf.a <install>/aarch64/lib/libc.a \
    --start-group -lgcc -lgcc_eh -end-goup
```

map_aarch64.txt

```
Archive member included because of file (symbol)
…
Memory map

 ** file header
                 00          0x40
 *
                0x0000000000400040          0xe0

.text            0x0000000000400120    0x21744
 .text.exit      0x0000000000400120       0x24 0x4 /home/rlyerly/Downloads/popcorn-compiler/test-install/aarch64/lib/libc.a(exit.o)
                 0x0000000000400120                exit
 .text           0x0000000000400144        0x0 0x4 main.o
 .text.main      0x0000000000400144       0x7c 0x4 main.o
                 0x0000000000400144                main
…
```

Section name

Link-Time Alignment

```
$ ld.gold -L/usr/lib/gcc-cross/aarch64-linux-gnu/5 -Map map_aarch64.txt \
    main.o fizzbuzz.o -o fizzbuzz_aarch64
    <install>/aarch64/lib/crt1.o <install>/aarch64/lib/libc.a \
    <install>/aarch64/lib/libmigrate.a <install>/aarch64/lib/libstack-transform.a \
    <install>/aarch64/lib/libelf.a <install>/aarch64/lib/libc.a \
    --start-group -lgcc -lgcc_eh -end-goup
```

map_aarch64.txt

```
Archive member included because of file (symbol)
…
Memory map

 ** file header
                0x0
 ** segment headers
                0x000000        040         0xe0

.text           0x0000000000400120    0x21744
 .text.exit     0x0000000000400120       0x24 0x4 /home/rlyerly/Downloads/popcorn-compiler/test-
install/aarch64/lib/libc.a(exit.o)
                0x0000000000400120                exit
 .text          0x0000000000400144        0x0 0x4 main.o
 .text.main     0x0000000000400144       0x7c 0x4 main.o
                0x0000000000400144                main
…
```

Virtual memory address

Link-Time Alignment

```
$ ld.gold -L/usr/lib/gcc-cross/aarch64-linux-gnu/5 -Map map_aarch64.txt \
    main.o fizzbuzz.o -o fizzbuzz_aarch64
    <install>/aarch64/lib/crt1.o <install>/aarch64/lib/libc.a \
    <install>/aarch64/lib/libmigrate.a <install>/aarch64/lib/libstack-transform.a \
    <install>/aarch64/lib/libelf.a <install>/aarch64/lib/libc.a \
    --start-group -lgcc -lgcc_eh -end-goup
```

map_aarch64.txt

```
Archive member included because of file (symbol)
…
Memory map

 ** file header
                0x0000000000400000        0
 ** segment headers
                0x0000000000400040        0xe

.text           0x0000000000400120     0x21744
 .text.exit     0x0000000000400120        0x24  0x4 /home/rlyerly/Downloads/popcorn-compiler/test-
install/aarch64/lib/libc.a(exit.o)
                0x0000000000400120                  exit
 .text          0x0000000000400144         0x0  0x4 main.o
 .text.main     0x0000000000400144        0x7c  0x4 main.o
                0x0000000000400144                  main
…
```

Size

Systems Software Research Group

VIRGINIA TECH

Link-Time Alignment

```
$ ld.gold -L/usr/lib/gcc-cross/aarch64-linux-gnu/5 -Map map_aarch64.txt \
    main.o fizzbuzz.o -o fizzbuzz_aarch64
    <install>/aarch64/lib/crt1.o <install>/aarch64/lib/libc.a \
    <install>/aarch64/lib/libmigrate.a <install>/aarch64/lib/libstack-transform.a \
    <install>/aarch64/lib/libelf.a <install>/aarch64/lib/libc.a \
    --start-group -lgcc -lgcc_eh -end-goup
```

map_aarch64.txt

```
Archive member included because of file (symbol)
…
Memory map

 ** file header
                0x0000000000400000        0x40
 ** segment headers
                0x0000000000400040        0xe0

.text           0x0000000000400120     0x21744
 .text.exit     0x0000000000400120        0x24 0x4 /home/rlyerly/Downloads/popcorn-compiler/test-
install/aarch64/lib/libc.a(exit.o)
                0x0000000000400120                 exit
 .text          0x0000000000400144         0x0 0x4 main.o
 .text.main     0x0000000000400144        0x7c 0x4 main.o
                0x0000000000400144                 main
…
```

Alignment

Link-Time Alignment

```
$ pyalign --compiler-inst <install> \
    --arm-bin fizzbuzz_aarch64 --arm-map map_aarch64.txt \
    --x86-bin fizzbuzz_x86_64 --x86-map map_x86_64.txt
```

Link-Time Alignment

```
$ pyalign --compiler-inst <install> \
    --arm-bin fizzbuzz_aarch64 --arm-map map_aarch64.txt \
    --x86-bin fizzbuzz_x86_64 --x86-map map_x86_64.txt
```

linker_script_arm.x

```
/* Default linker script, for normal executables */
OUTPUT_FORMAT("elf64-littleaarch64", "elf64-bigaarch64",
                 "elf64-littleaarch64")
OUTPUT_ARCH(aarch64)
ENTRY(_start)
SECTIONS
{
…
. .text     : ALIGN(0x100000)
{
           . = . + 1;
           . = ALIGN(0x1000);
           …
           . = ALIGN(0x10); /* align for .text.main */
           "main.o"(.text.main); /* size 0x7c */
           . = ALIGN(0x10); /* align for .text.fizzbuzz */
           "fizzbuzz.o"(.text.fizzbuzz); /* size 0xfc */
           …
```

Systems Software Research Group

VIRGINIA TECH

Link-Time Alignment

```
$ pyalign --compiler-inst <install> \
    --arm-bin fizzbuzz_aarch64 --arm-map map_aarch64.txt \
    --x86-bin fizzbuzz_x86_64 --x86-map map_x86_64.txt
```

linker_script_arm.x

```
/* Default linker script, for normal executables */
OUTPUT_FORMAT("elf64-littleaarch64"
                 "elf64-littleaar
OUTPUT_ARCH(aarch64)
ENTRY(_start)
SECTIONS
{
…
.  .text     : ALIGN(0x100000)
{
           . = . + 1;
           . = ALIGN(0x1000);
           …
           . = ALIGN(0x10); /* align for .text.main */
           "main.o"(.text.main); /* size 0x7c */
           . = ALIGN(0x10); /* align for .text.fizzbuzz */
           "fizzbuzz.o"(.text.fizzbuzz); /* size 0xfc */
           …
```

Re-link each binary with generated linker script – they're now aligned!

Post-Processing

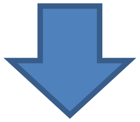
- Need to reorganize LLVM’s stackmap section
  - Not conducive for fast runtime acces
    - Variable sized components within a call site record
    - Stackmap sections from multiple files lumped together
  - Need to uniquify call site IDs across compilation units

Post-Processing

```
gen-stackinfo -f fizzbuzz_aarch64 && readelf -SW fizzbuzz_aarch64
```

Post-Processing

```
gen-stackinfo –f fizzbuzz_aarch64 && readelf –SW fizzbuzz_aarch64
```



```
There are 38 section headers, starting at offset 0x66a3cb:

Section Headers:
  [Nr] Name                  Type            Address          Off    Size   ES Flg Lk Inf Al
  [ 0]                       NULL            0000000000000000 000000 000000 00      0   0  0
  [ 1] .text                 PROGBITS        0000000000500000 100000 024ec4 00  AX  0   0 1048576
  [ 2] .rodata               PROGBITS        0000000000600000 200000 00438f 00   A  0   0 1048576
  …
  [31] .symtab               SYMTAB          0000000000000000 655618 00fe10 18     32 2280  8
  [32] .strtab               STRTAB          0000000000000000 665428 0039ef 00      0   0  1
  [33] .shstrtab             STRTAB          0000000000000000 668e17 000208 00      0   0  1
  [34] .stack_transform.id PROGBITS          0000000000000000 66901f 0007ec 34      0   0  0
  [35] .stack_transform.addr PROGBITS          0000000000000000 66980b 0007ec 34      0   0  0
  [36] .stack_transform.live PROGBITS          0000000000000000 669ff7 000348 0c      0   0  0
  [37] .stack_transform.arch_const PROGBITS          0000000000000000 66a33f 00008c 14      0   0  0
```

Post-Processing

```
gen-stackinfo -f fizzbuzz_aarch64 && readelf -SW fizzbuzz_aarch64
```

```
There are 38 section headers, starting at offset 0x66a3cb:

Section Headers:
  [Nr] Name              Type            Address          Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            0000000000000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        0000000000500000 100000 024ec4 00  AX  0   0 1048576
  [ 2] .rodata                                                                  0   0 1048576
  …
  [31] .symtab                                                                 32 2280  8
  [32] .strtab           STRT            0000000000000000 665428 0039ef 00      0   0  1
  [33] .shstrtab         STRTAB          0000000000000000 668e17 000208 00      0   0  1
  [34] .stack_transform.id PROGBITS        0000000000000000 66901f 0007ec 34      0   0  0
  [35] .stack_transform.addr PROGBITS        0000000000000000 66980b 0007ec 34      0   0  0
  [36] .stack_transform.live PROGBITS        0000000000000000 669ff7 000348 0c      0   0  0
  [37] .stack_transform.arch_const PROGBITS        0000000000000000 66a33f 00008c 14      0   0  0
```

Consumed by the stack transformation runtime

Post-Processing

- The binaries are now ready for runtime migration
  - One binary per target in the format `<name>_<target>`
- Useful utilities:
  - `check-align.py`: consume the symbol tables of two binaries to verify that all symbols begin at aligned virtual addresses
  - `check-stackmaps.py`: sanity check the stackmaps generated by the compiler (same numbers and types of functions, call sites, live values)
    - Must be run post-alignment – matches functions by address

Part 4:
Migration & Stack Transformation

Migrating Between Architectures

- Migration in Popcorn Linux
  - Somebody (either inside or outside the application) ***proposes*** that a given thread migrates to a given destination node
  - The `check_migrate` function queries whether a migration has been proposed for the current thread
  - If so, `check_migrate` invokes stack transformation
    - Take a snapshot of current register set
    - Rewrite stack to another location in memory (in userspace)
    - Return populated destination ISA register set
  - Pass rewritten register set to kernel's thread migration service
  - Resume execution in `check_migrate` on destination

Triggering Migration

- Proposing migration (`libmigrate:trigger.c`):

```
syscall(SYSCALL_PROPOSE_MIGRATION, pid, nid)
```

Triggering Migration

- Proposing migration (`libmigrate:trigger.c`):

Direct shortcut to the system call interface

```
syscall(SYSCALL_PROPOSE_MIGRATION, pid, nid)
```

Triggering Migration

- Proposing migration (`libmigrate:trigger.c`):

System call number – proposing a migration

```
syscall(SYSCALL_PROPOSE_MIGRATION, pid, nid)
```

Triggering Migration

- Proposing migration (`libmigrate:trigger.c`):

For which task are we proposing migration

```
syscall(SYSCALL_PROPOSE_MIGRATION, pid, nid)
```

Triggering Migration

- Proposing migration (`libmigrate:trigger.c`):

To which node the task should migrate

```
syscall(SYSCALL_PROPOSE_MIGRATION, pid, nid)
```

Triggering Migration

- Checking to see if a migration was proposed (`libmigrate:migrate.c`):

```
syscall(SYSCALL_MIGRATION_PROPOSED)
```

Triggering Migration

- Checking to see if a migration was proposed (`libmigrate:migrate.c`):

Return whether a migration was proposed for the current thread
- >= 0: proposed destination node
- < 0: no migration proposed

```
syscall(SYSCALL_MIGRATION_PROPOSED)
```

Triggering Migration

- Invoking thread migration service (`libmigrate:migrate.c`):

```
syscall(SYSCALL_SCHED_MIGRATE, nid, regs_dst, sp, bp)
```

Triggering Migration

- Invoking thread migration service (`libmigrate:migrate.c`):

Migrate the thing!

```
syscall(SYSCALL_SCHED_MIGRATE, nid, regs_dst, sp, bp)
```

Triggering Migration

- Invoking thread migration service (`libmigrate:migrate.c`):

Where to migrate

```
syscall(SYSCALL_SCHED_MIGRATE, nid, regs_dst, sp, bp)
```

Triggering Migration

- Invoking thread migration service (`libmigrate:migrate.c`):

Destination register set – thread will be restarted with these registers on destination

- Generated by stack transformation runtime

```
syscall(SYSCALL_SCHED_MIGRATE, nid, regs_dst, sp, bp)
```

Triggering Migration

- Invoking thread migration service (`libmigrate:migrate.c`):

New stack & frame pointer on destination

```
syscall(SYSCALL_SCHED_MIGRATE, nid, regs_dst, sp, bp)
```

Triggering Migration

- Invoking thread migration service (`libmigrate:migrate.c`):

These interfaces are subject to change – see `<repo>/lib/migrate` for up-to-date versions!

ation

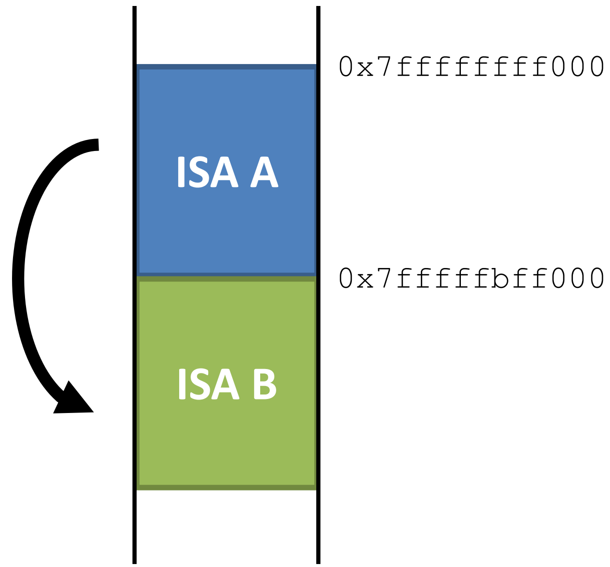
```
syscall(S                                        , sp, bp)
```

Stack Transformation

- Rewrite entire stack from outermost frame inwards
- Stack transformation runtime (the "runtime") opens metadata sections using `libelf`
  - `.stack_transform.unwind` – per-function callee-saved register locations on the stack for frame unwinding
  - `.stack_transform.unwind_arange` – address ranges for functions in the binary (used for bootstrapping outermost frame)
  - `.stack_transform.id` – call sites (stackmaps) sorted by ID
  - `.stack_transform.addr` – call sites sorted by program location
  - `.stack_transform.live` – live value location records
  - `.stack_transform.arch_const` – architecture-specific live value records

Stack Transformation

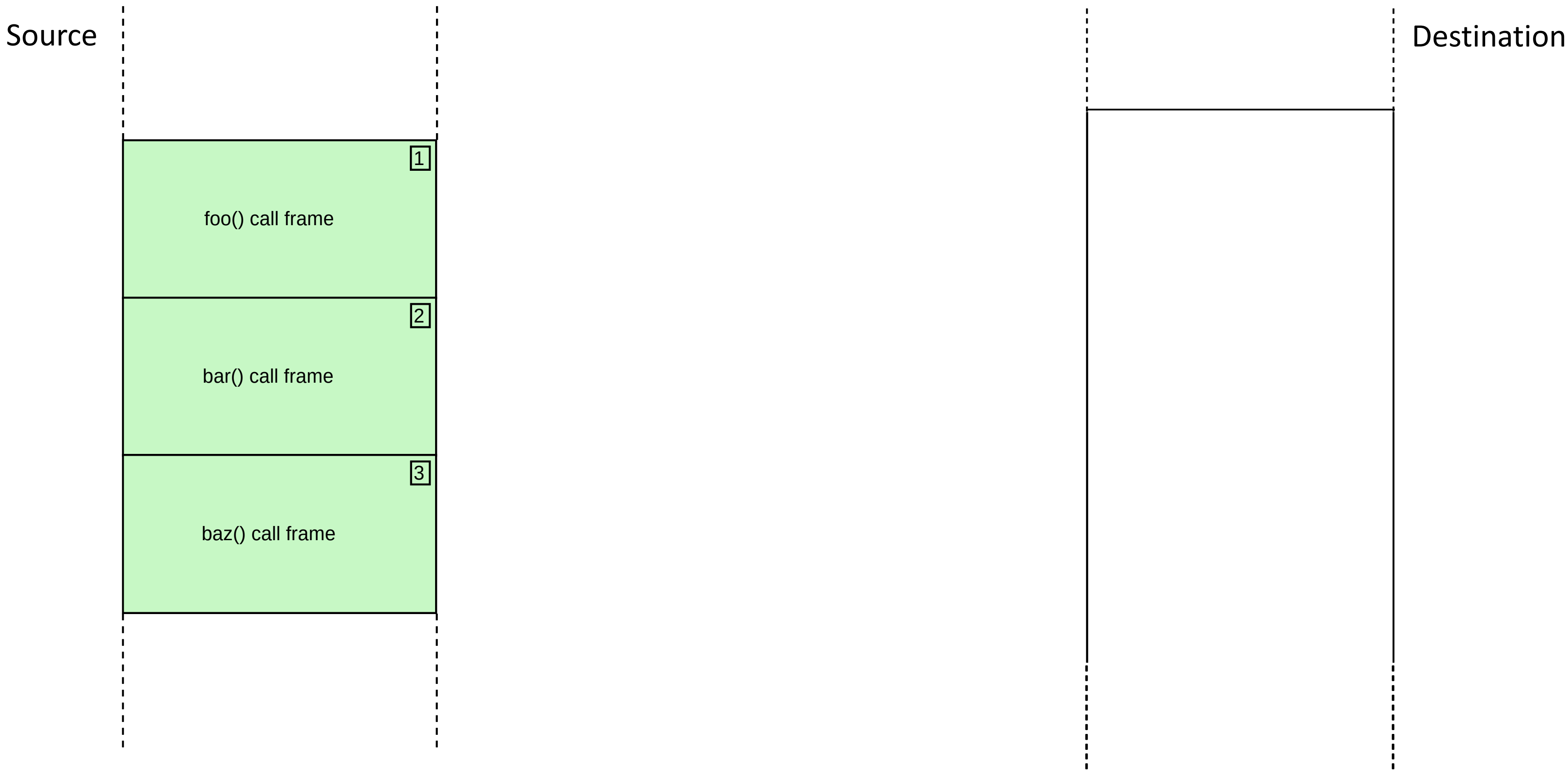
- All transformation is performed in userspace
  - Linux by default gives the main thread 8MB of stack space (musl is modified to do the same for spawned threads)
  - Divide stack into upper/lower half, rewrite from one to the other at migration time



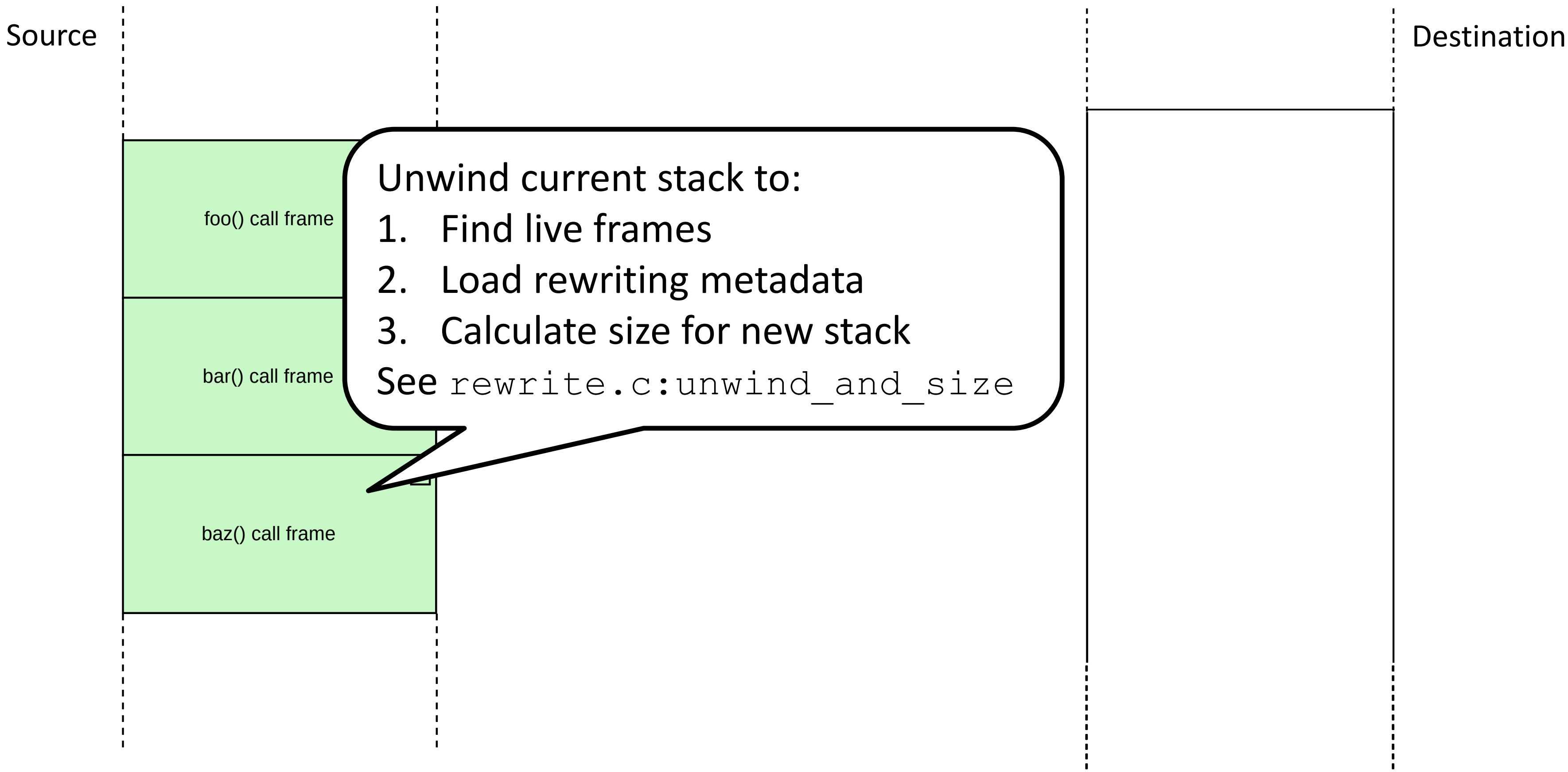
Stack Transformation

- Runtime initializes handles containing transformation metadata at program startup
  - Ways to tell runtime name of binary containing a target's metadata
    1. Set environment variable `ST_<target>_BIN`
    2. Define symbol `<target>_fn`
    3. If (1) & (2) not used, runtime will default to appending target to current executing binary's name

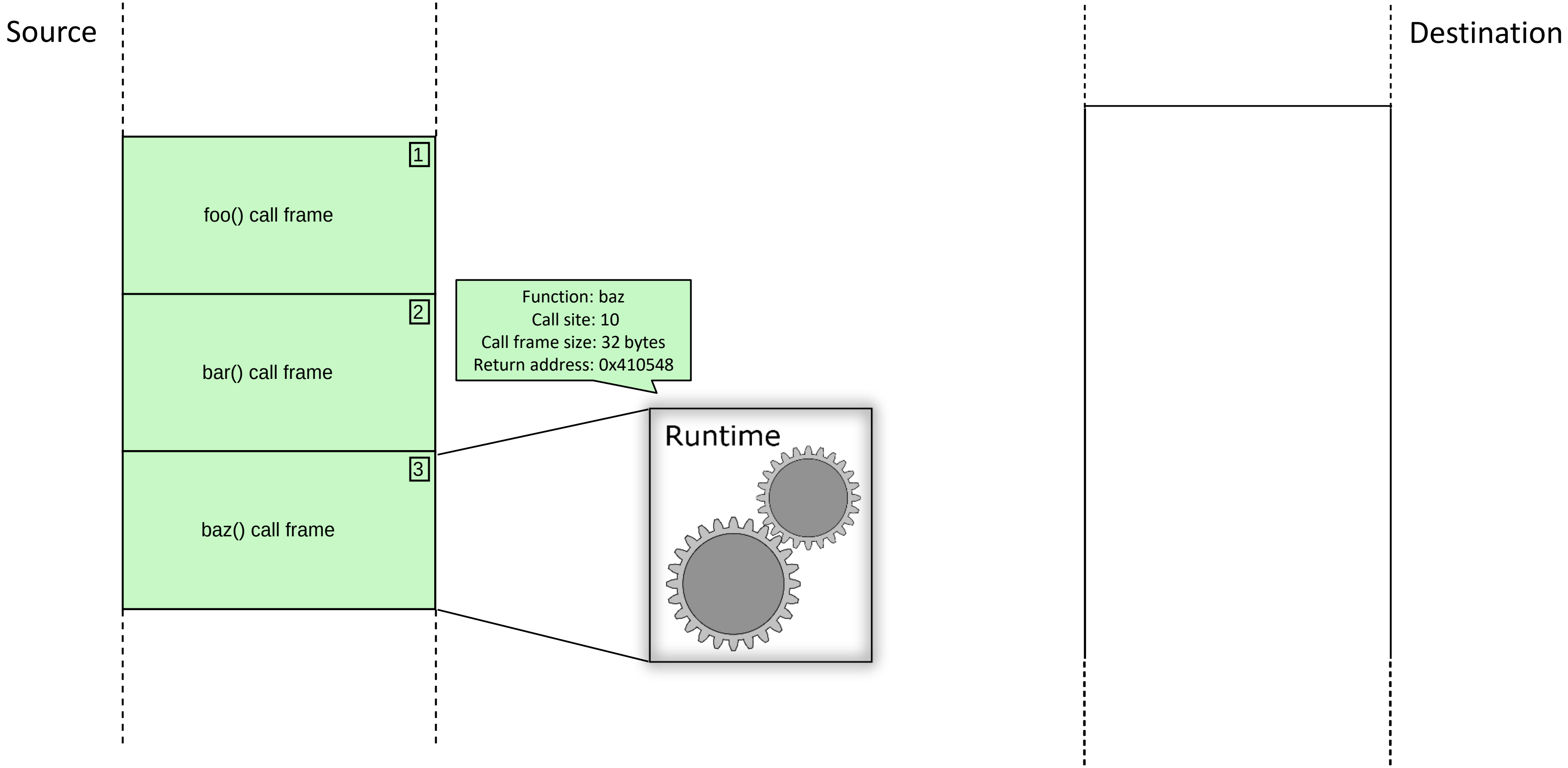
Stack Transformation



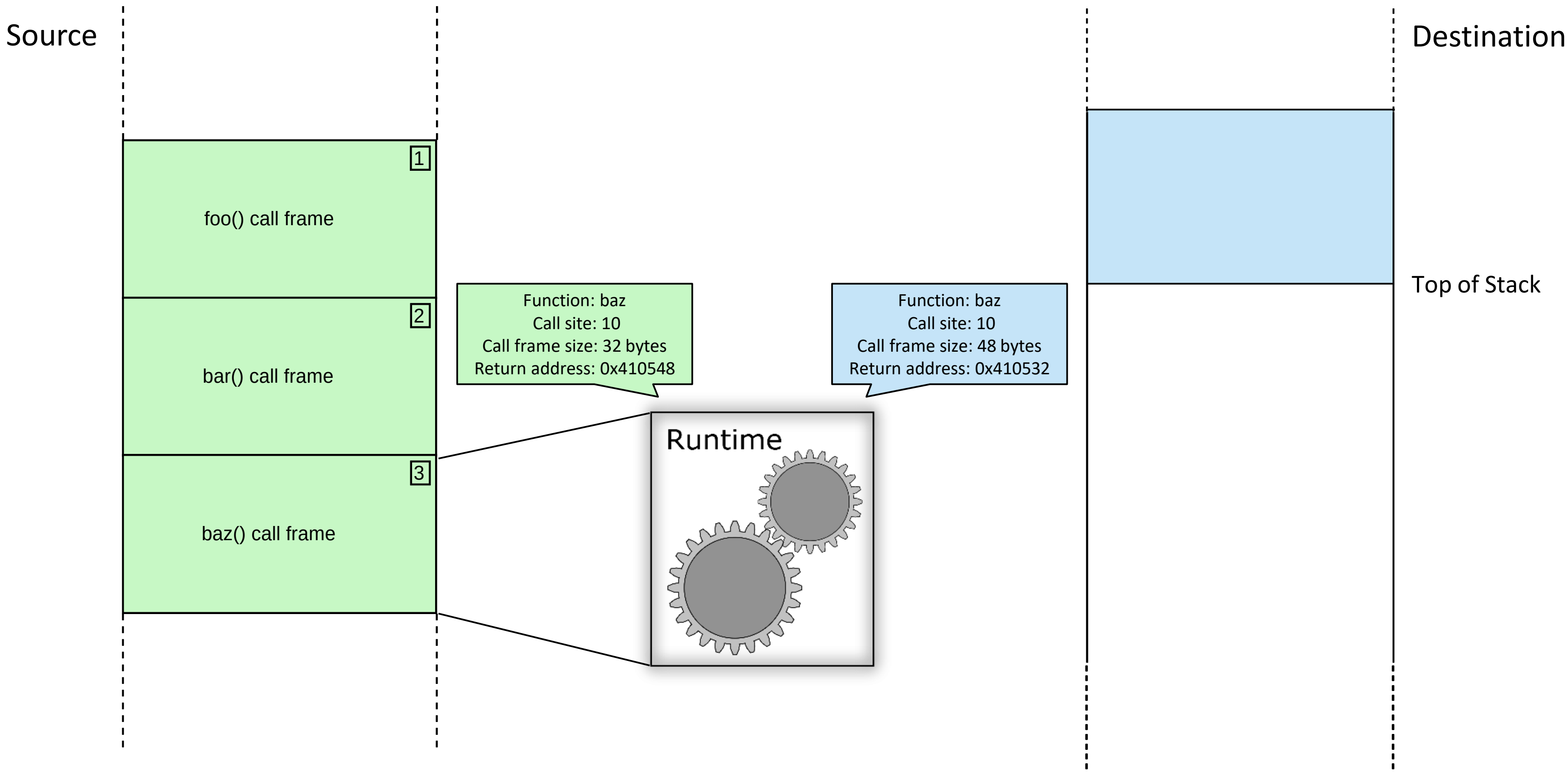
Stack Transformation



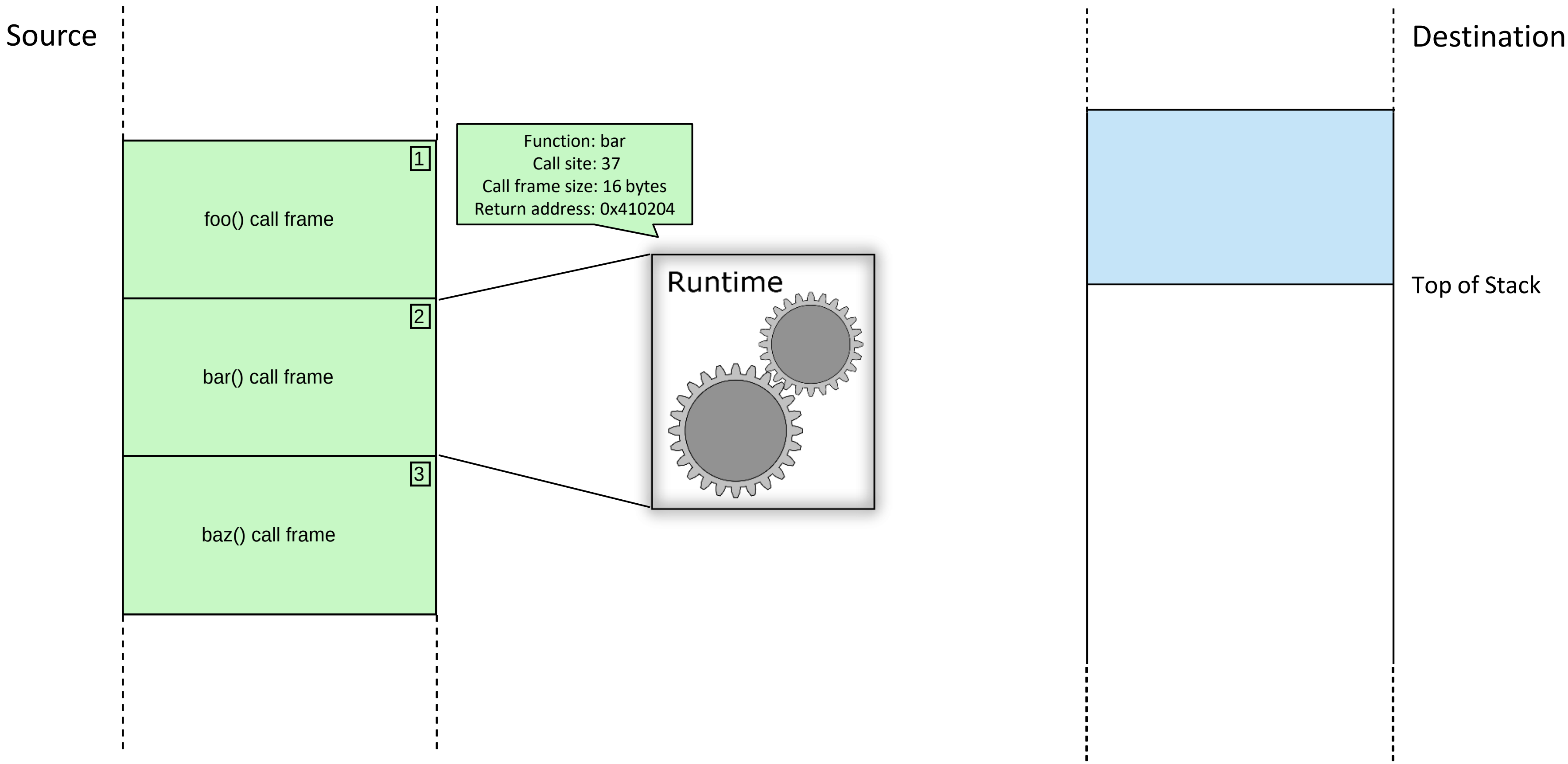
Stack Transformation



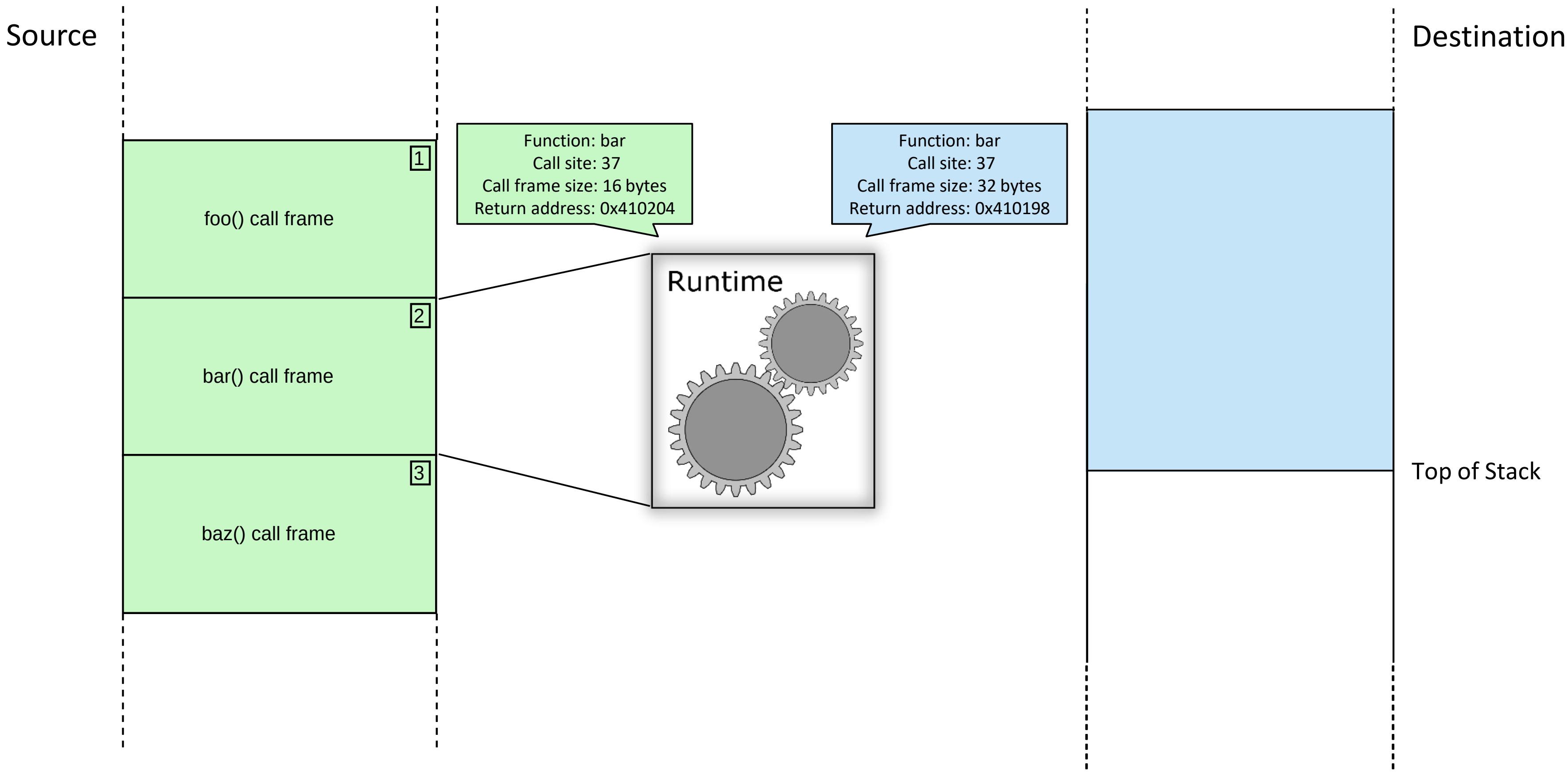
Stack Transformation



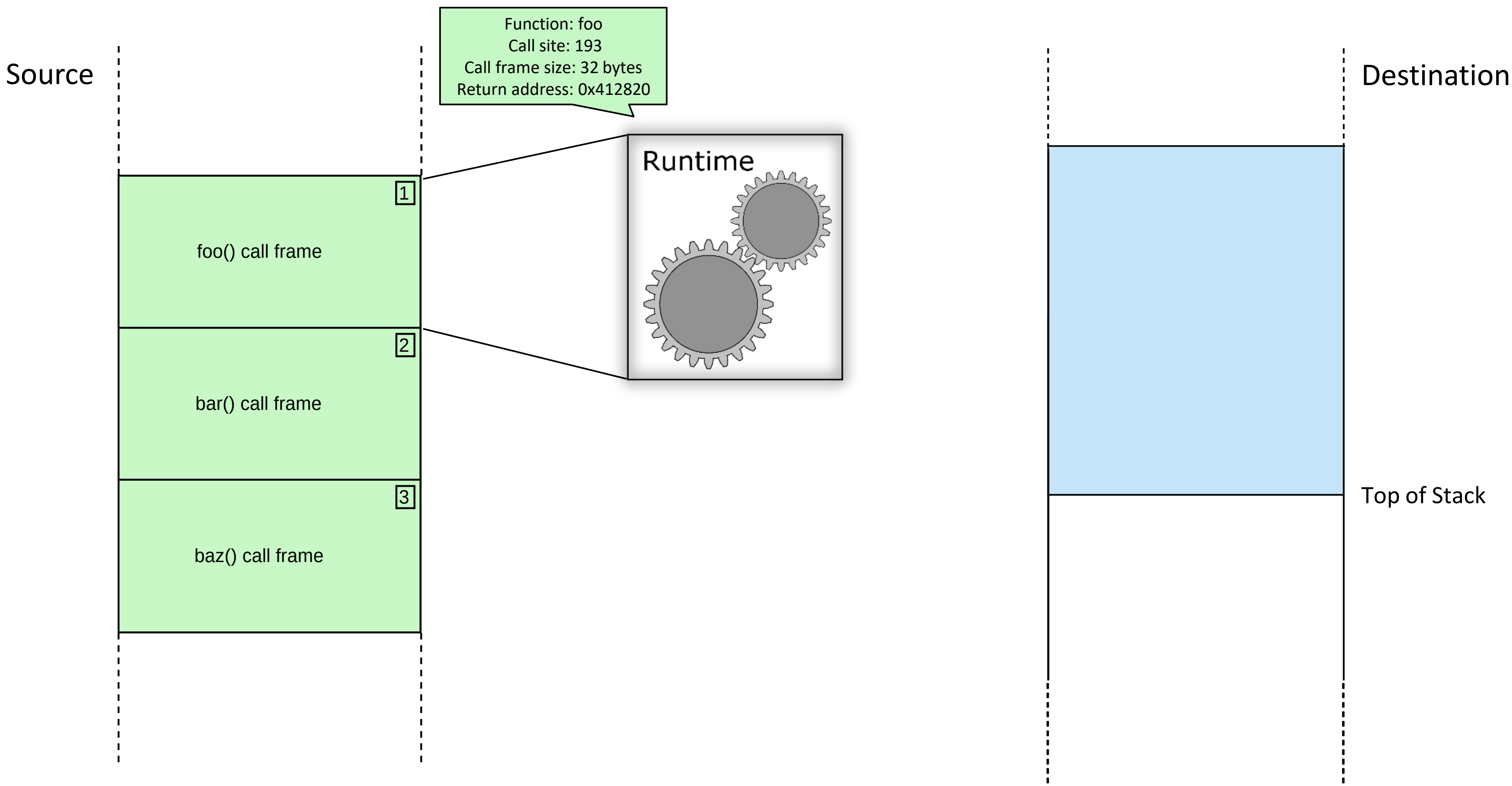
Stack Transformation



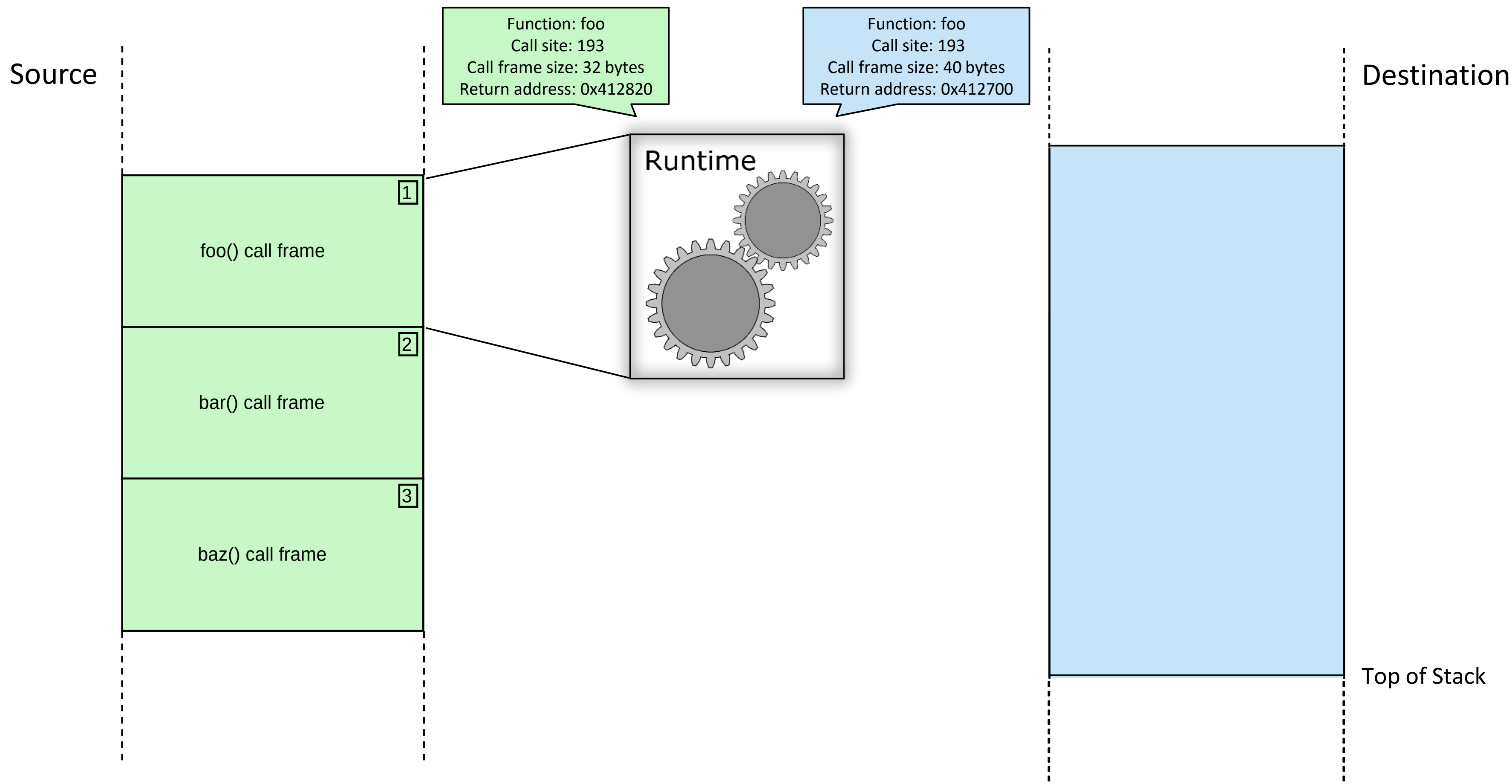
Stack Transformation



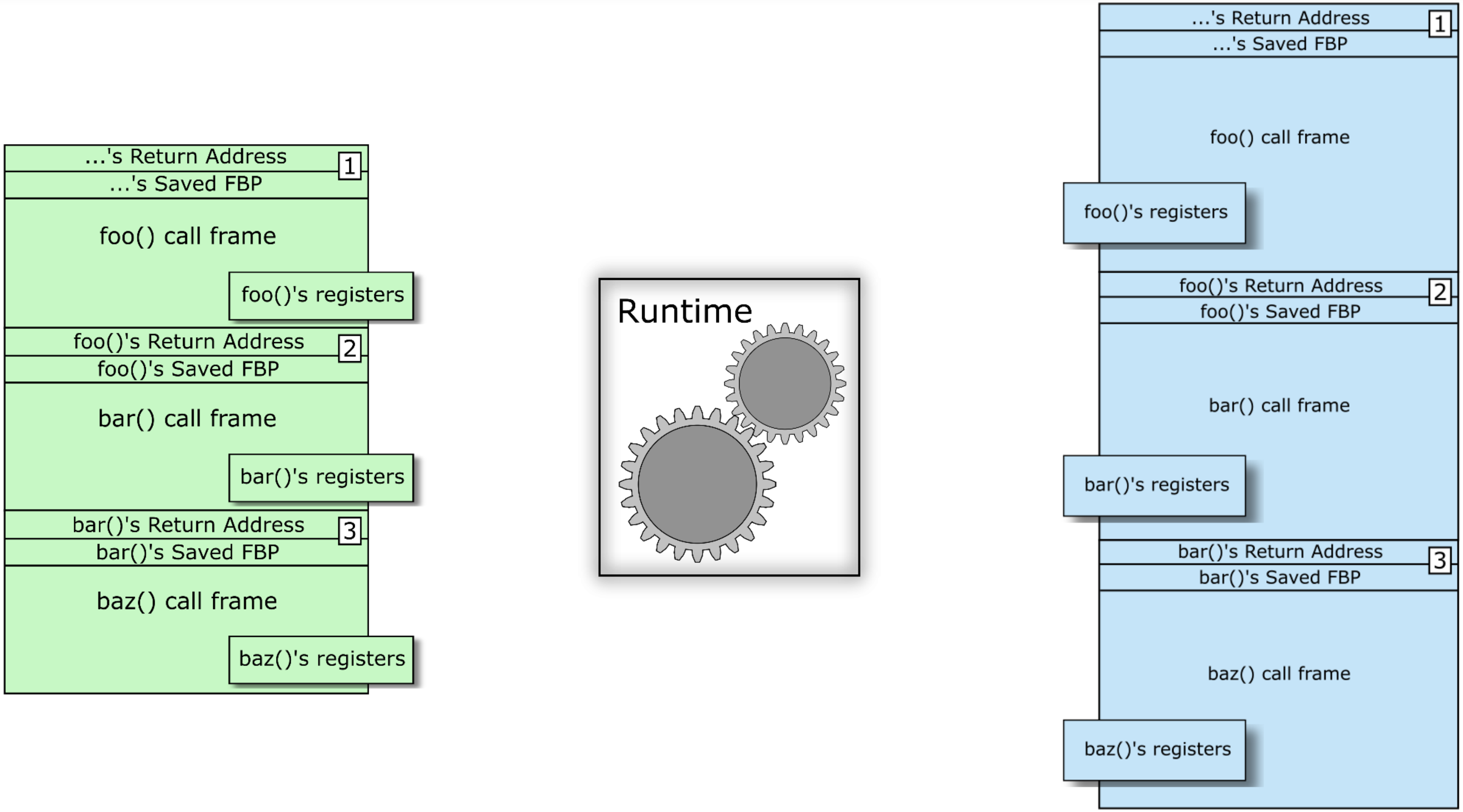
Stack Transformation



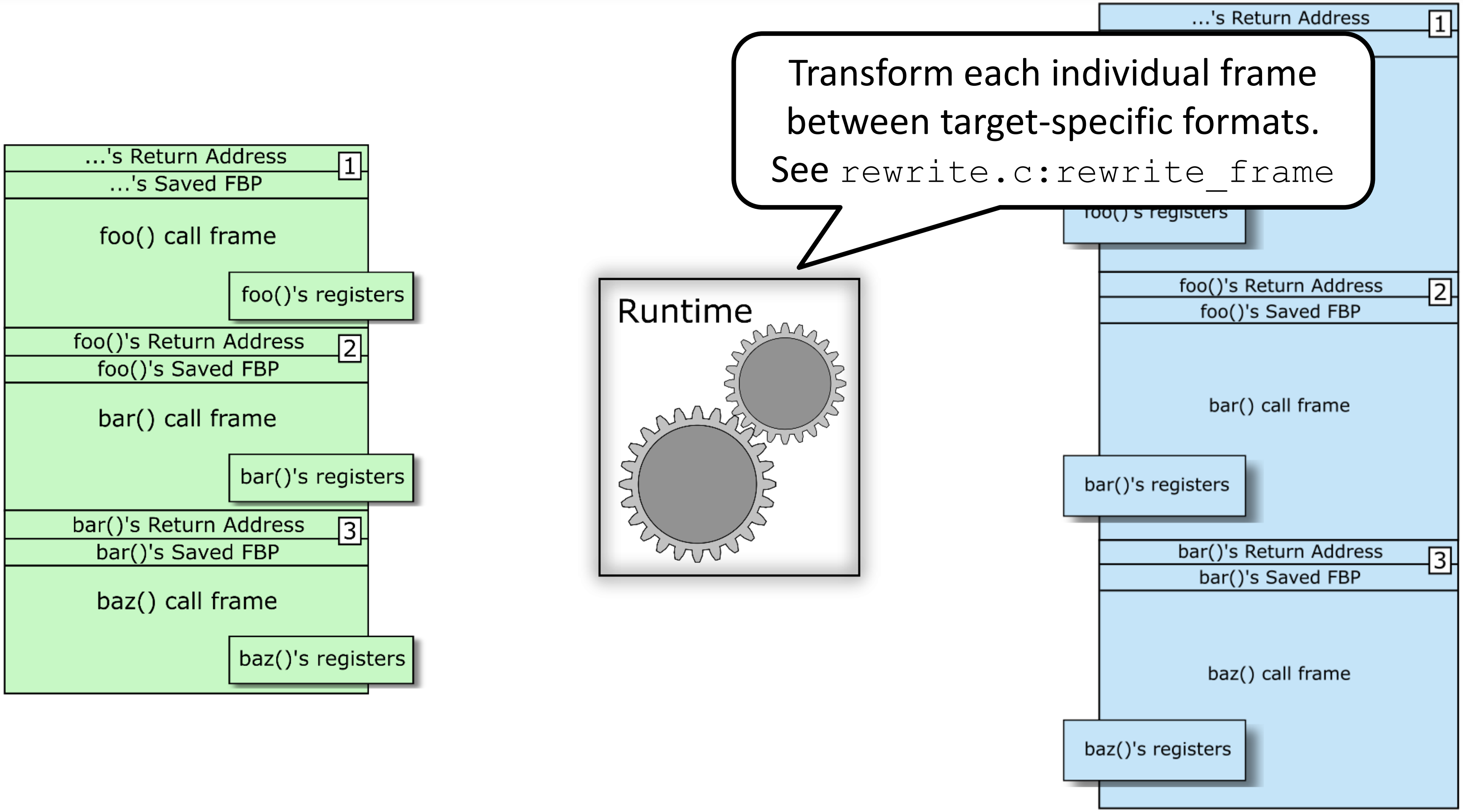
Stack Transformation



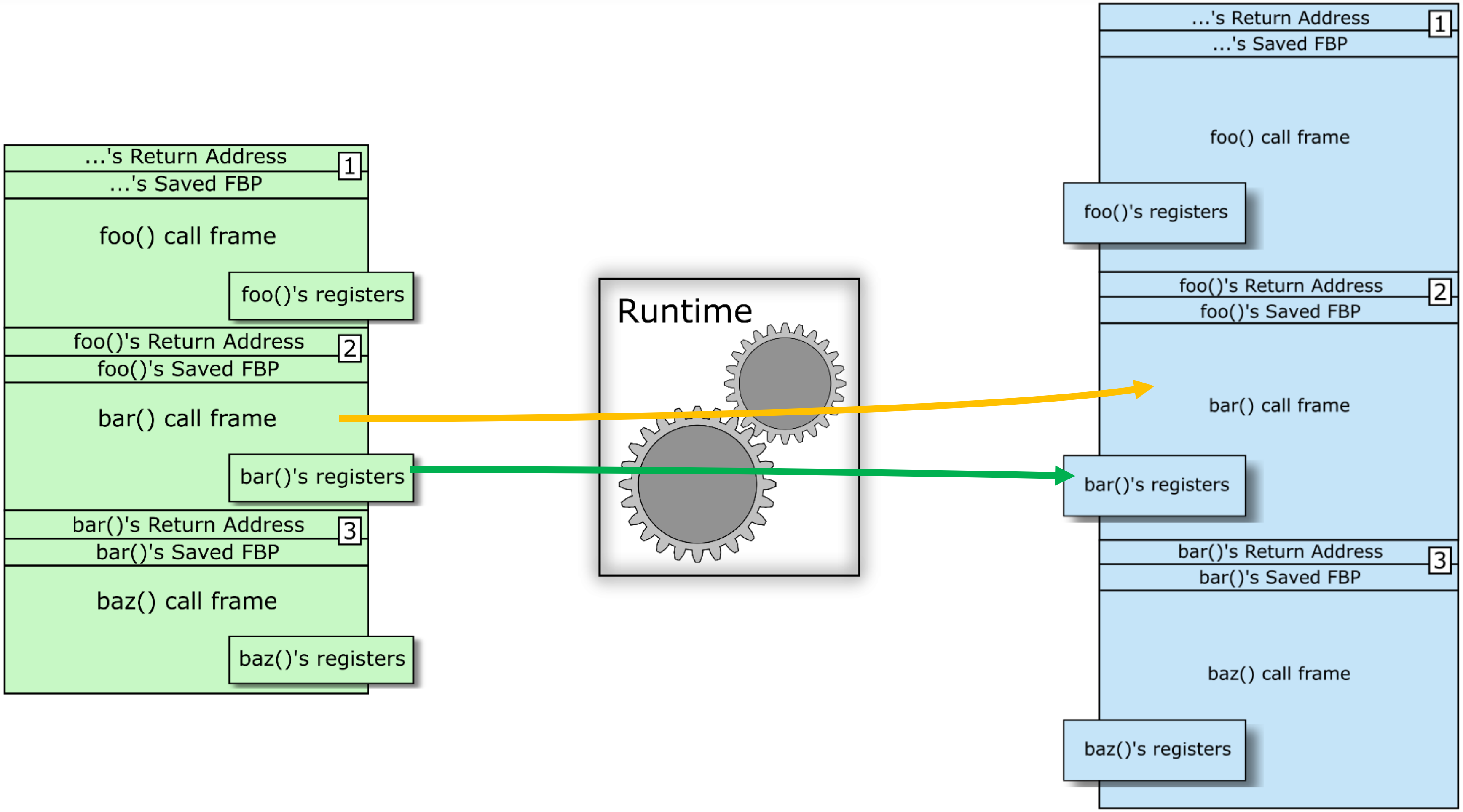
Stack Transformation



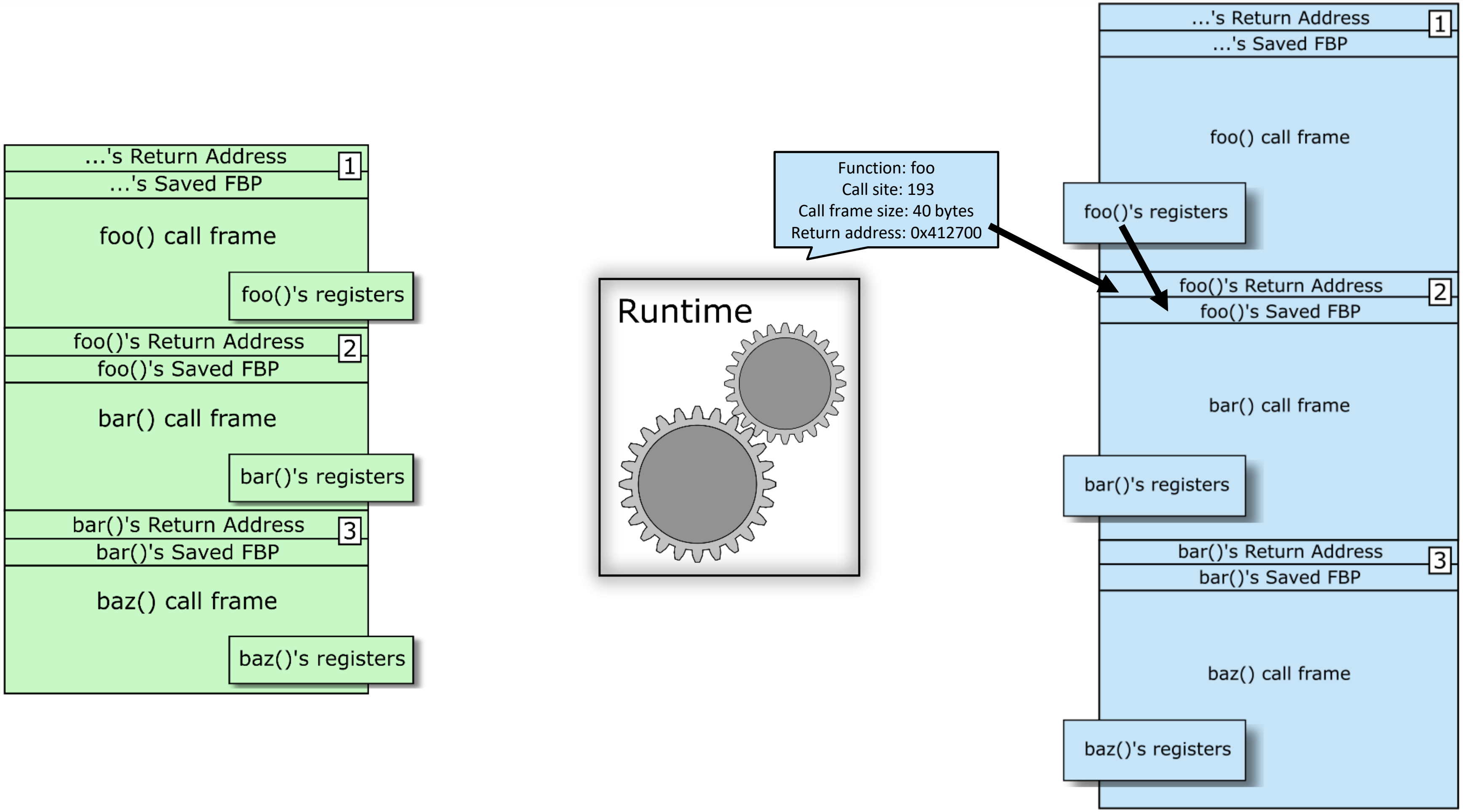
Stack Transformation



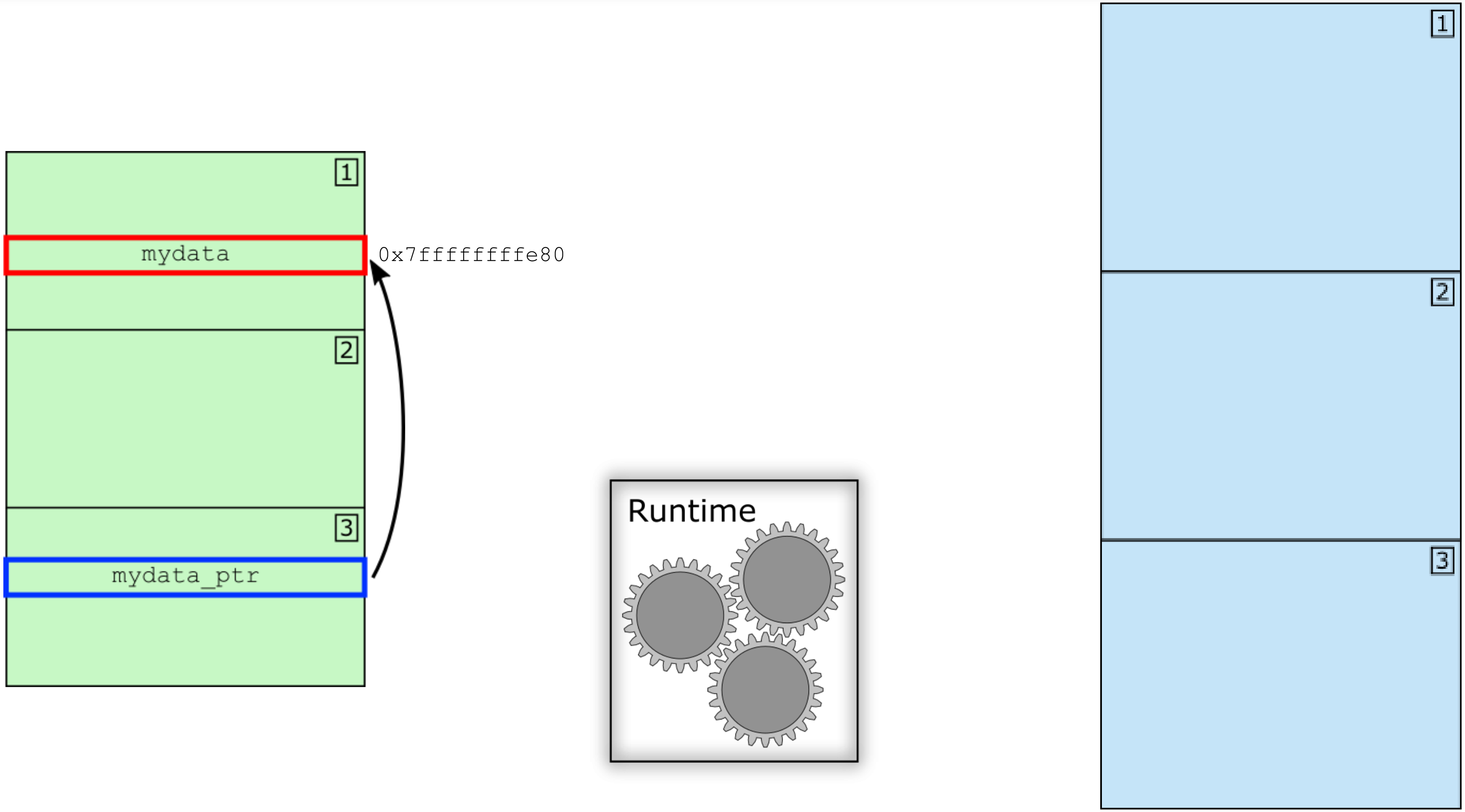
Stack Transformation



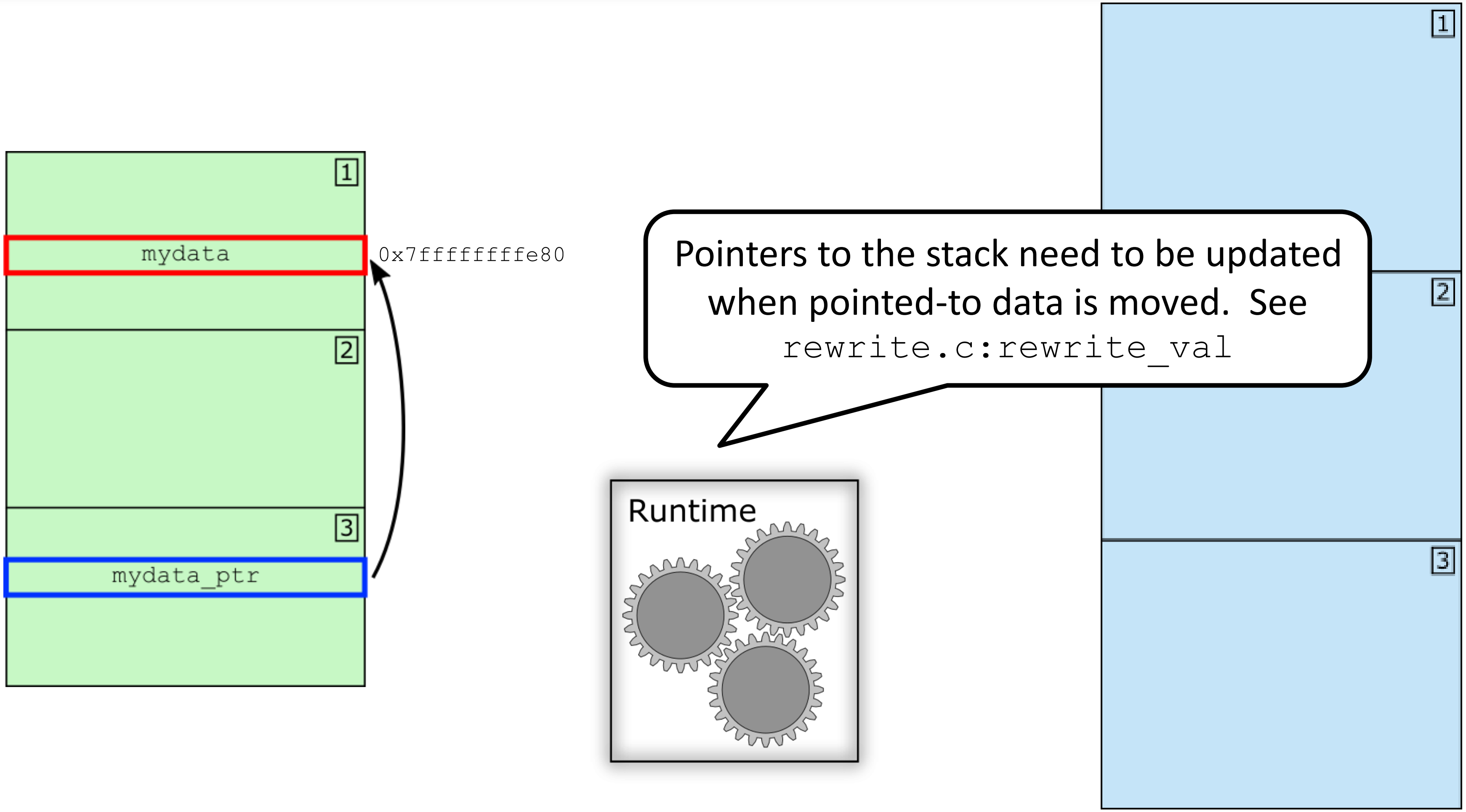
Stack Transformation



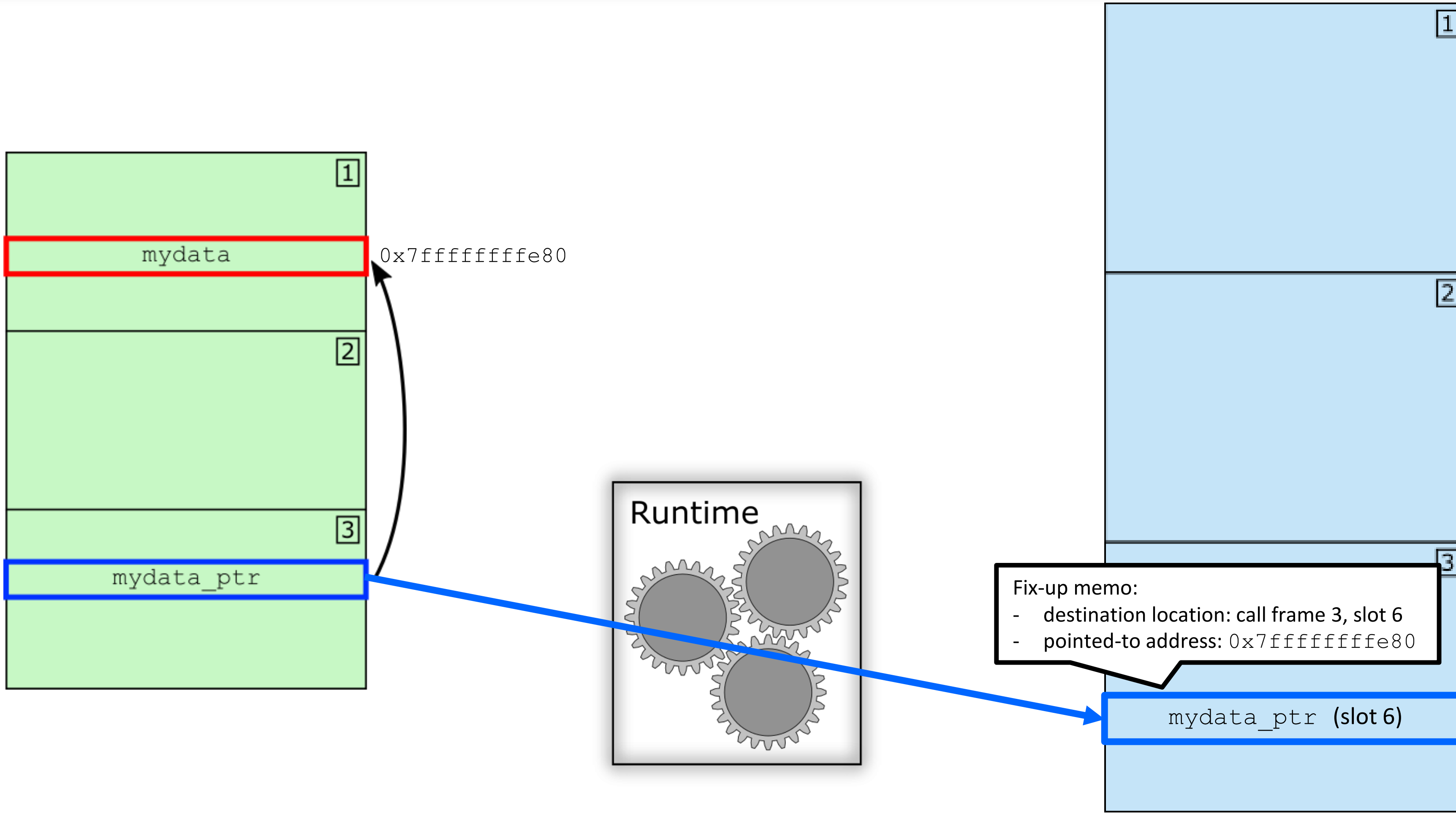
State Transformation



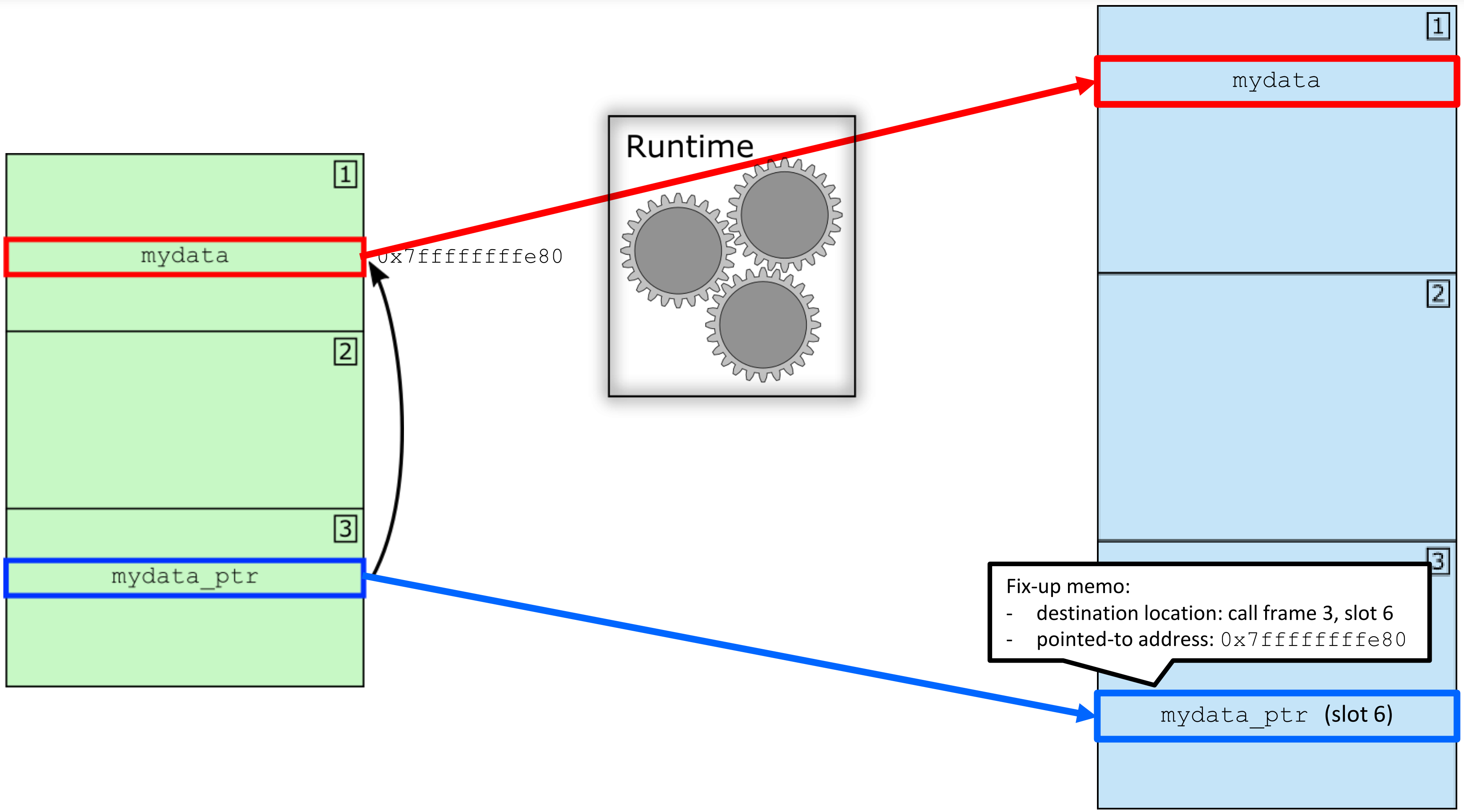
State Transformation



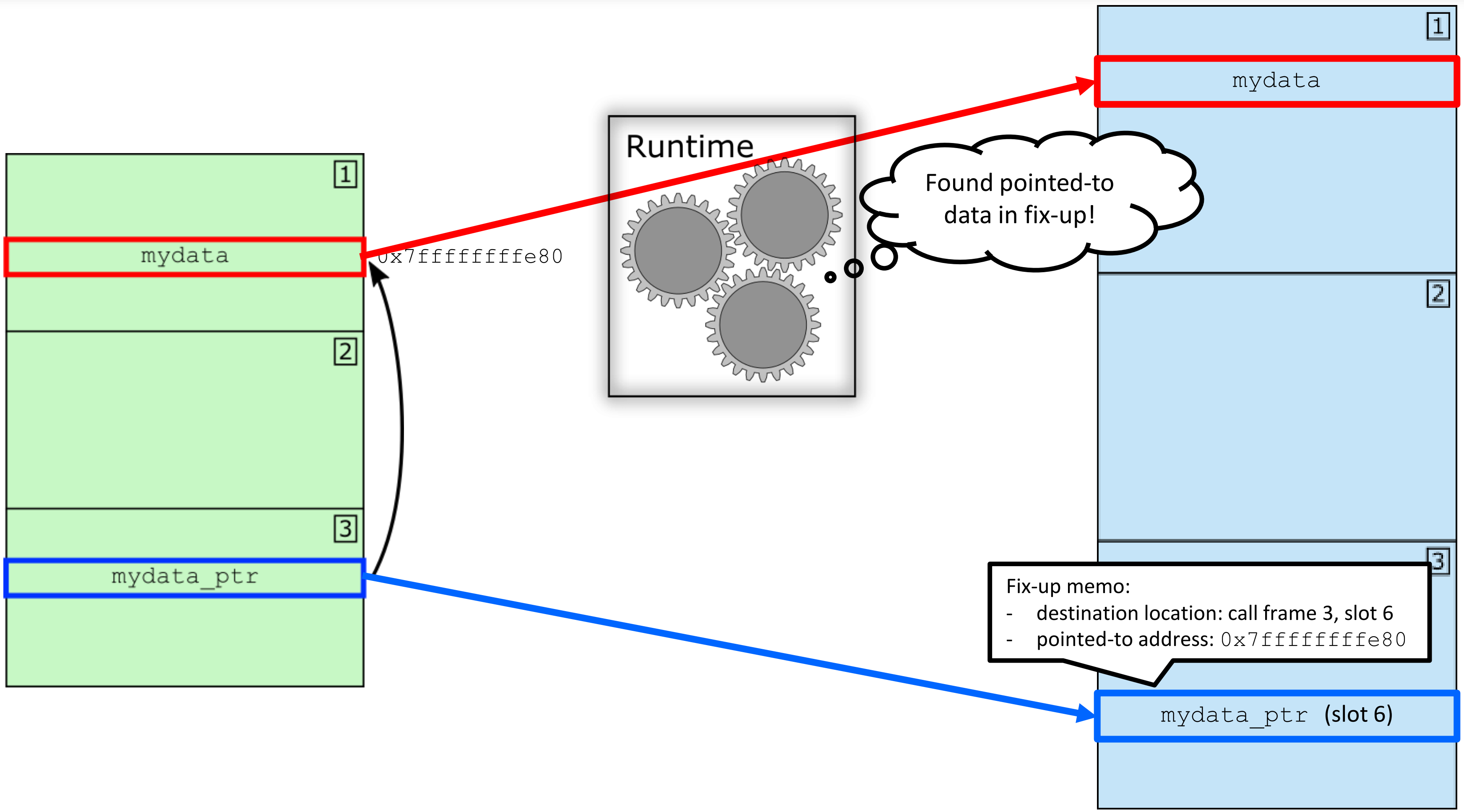
State Transformation



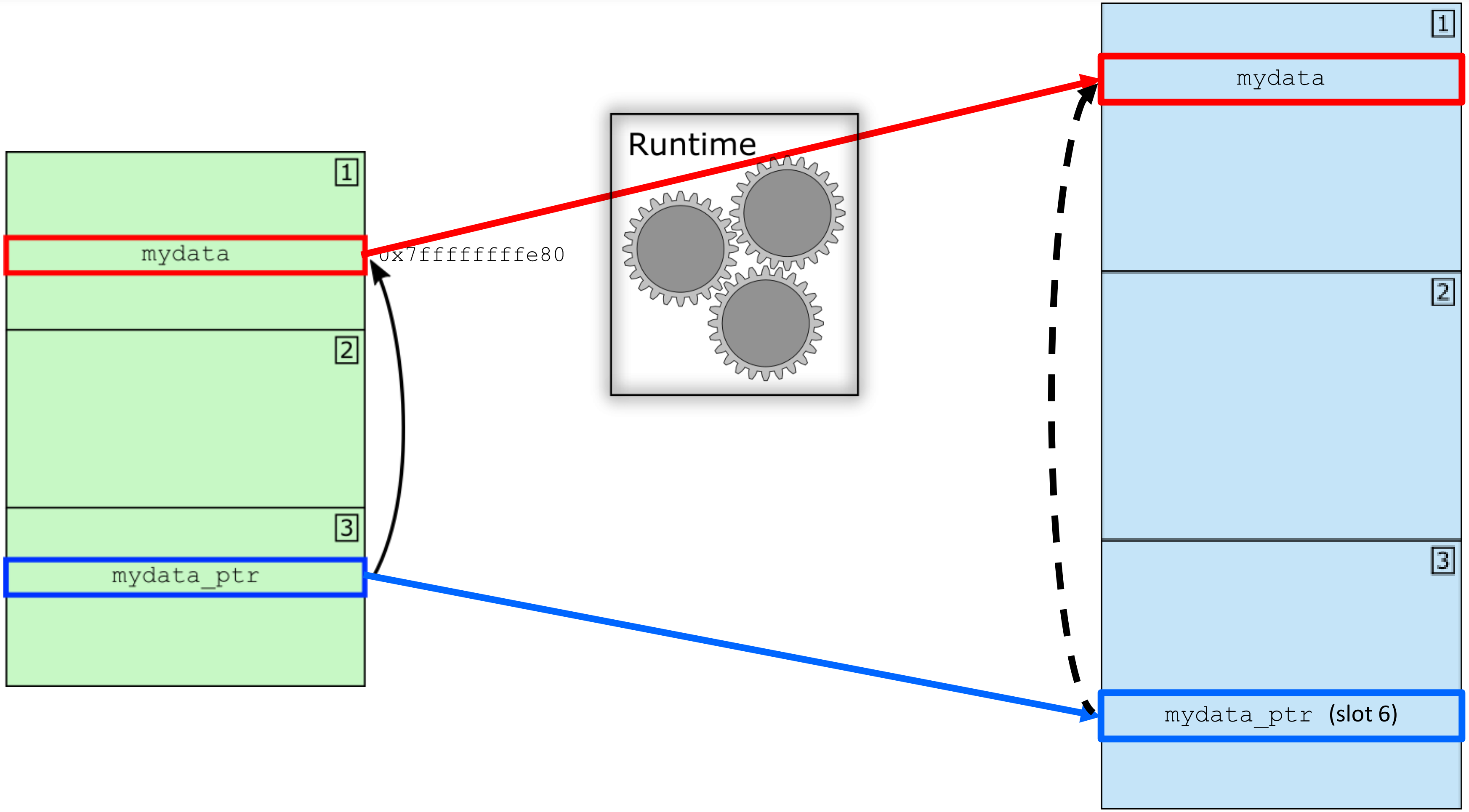
State Transformation



State Transformation



State Transformation



Runtime Summary

- Migrations are ***proposed*** by threads/applications
- Threads check to see if migration is proposed at migration points
- If so, invoke migration procedure
  - Transform thread's register set, stack between target-specific formats
  - Invoke kernel's thread migration service
  - Bootstrap on destination & return to normal execution