Proposing LLVM Extensions for Generating Native Code Fragments

Frej Drejhammar and Lars Rasmusson
Swedish Institute of Computer Science
{frej,lars.rasmusson}@sics.se

150413

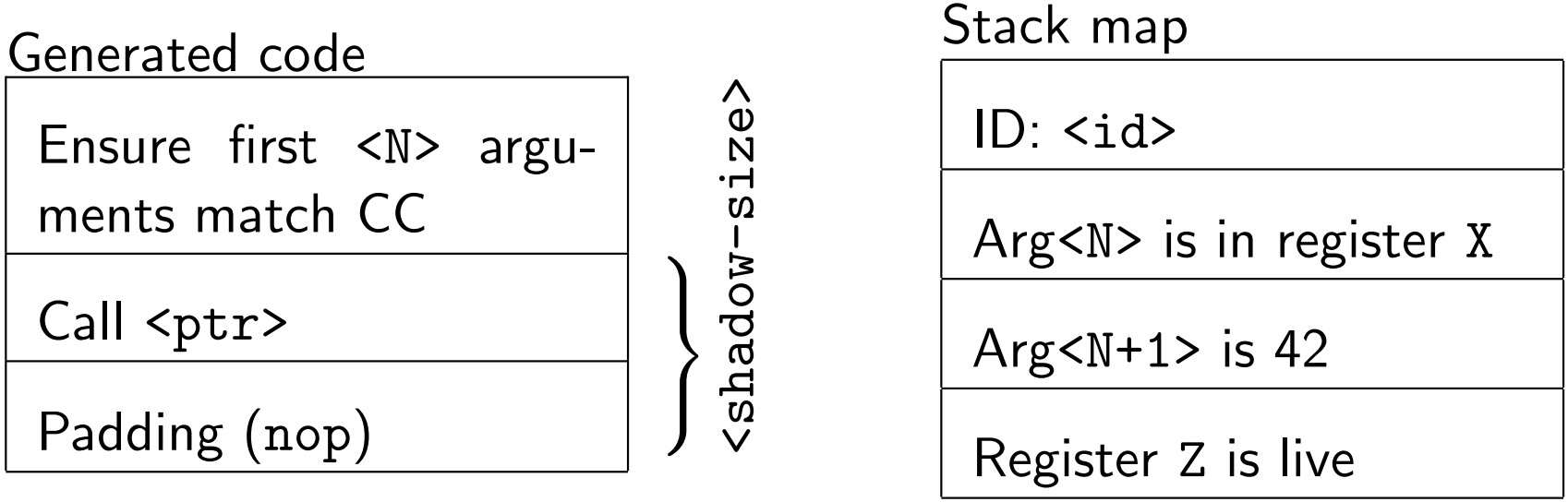
SWEDISH ICT SICS

Who am I?

Senior researcher at the Swedish Institute of Computer Science (SICS) working on programming languages, tools and distributed systems.
Currently working on an Ericsson funded JIT-compiler for Erlang.

Patchpoints

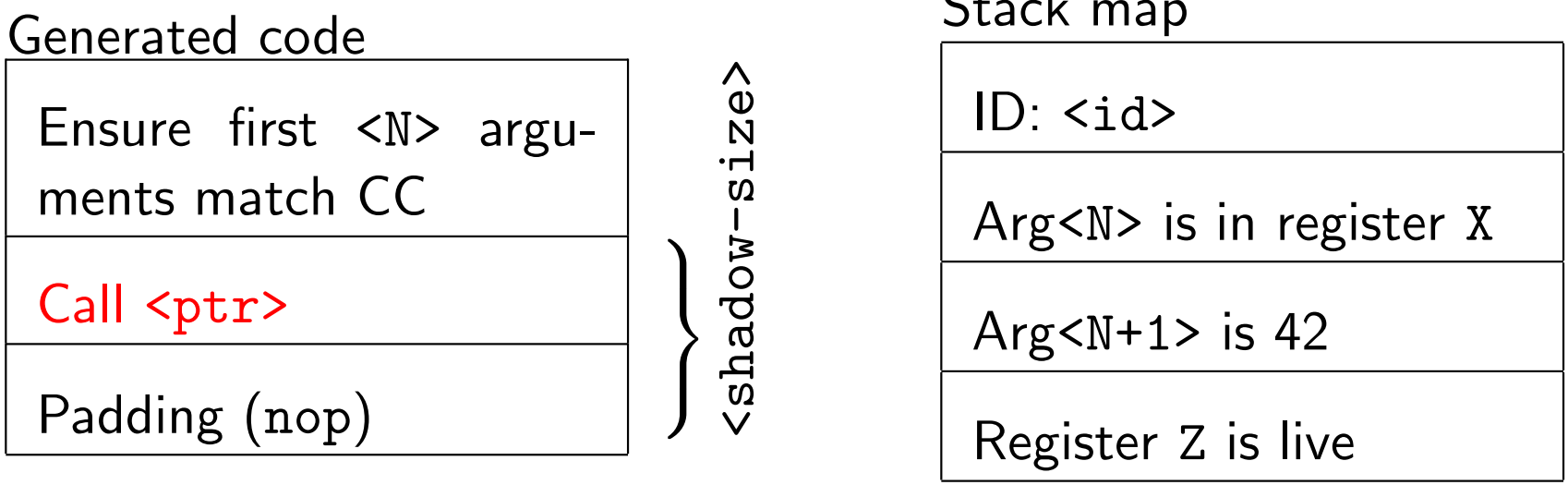
```
call void (i64, i32, i8*, i32, ...)*
      @llvm.experimental.patchpoint.void(
           i64 <id>, i32 <shadow-size>,i8* <ptr>,
           i32 <N>, arg0, arg1, ....)
```



But how to create the function pointed to by <ptr>?

Patchpoints

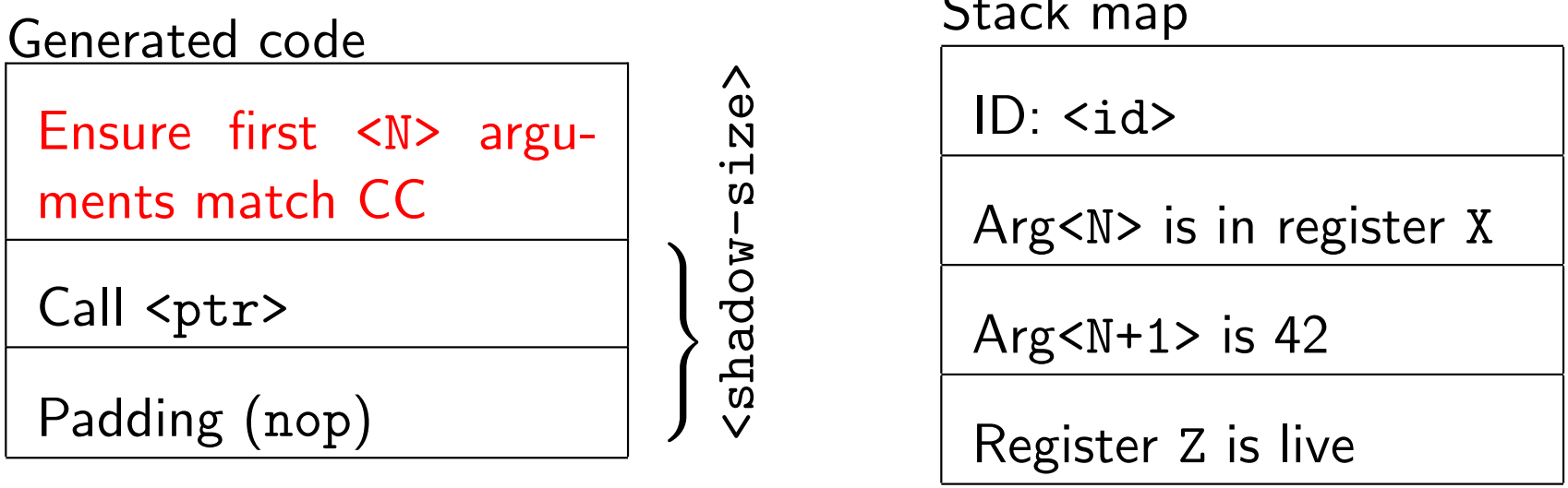
```
call void (i64, i32, i8*, i32, ...)*
      @llvm.experimental.patchpoint.void(
           i64 <id>, i32 <shadow-size>,i8* <ptr>,
           i32 <N>, arg0, arg1, ....)
```



But how to create the function pointed to by <ptr>?

Patchpoints

```
call void (i64, i32, i8*, i32, ...)*
      @llvm.experimental.patchpoint.void(
           i64 <id>, i32 <shadow-size>,i8* <ptr>,
           i32 <N>, arg0, arg1, ....)
```



But how to create the function pointed to by <ptr>?

Patchpoints

```
call void (i64, i32, i8*, i32, ...)*
      @llvm.experimental.patchpoint.void(
           i64 <id>, i32 <shadow-size>,i8* <ptr>,
           i32 <N>, arg0, arg1, ....)
```

Generated code

Ensure first <N> arguments match CC
Call <ptr>
Padding (nop)

<shadow-size>

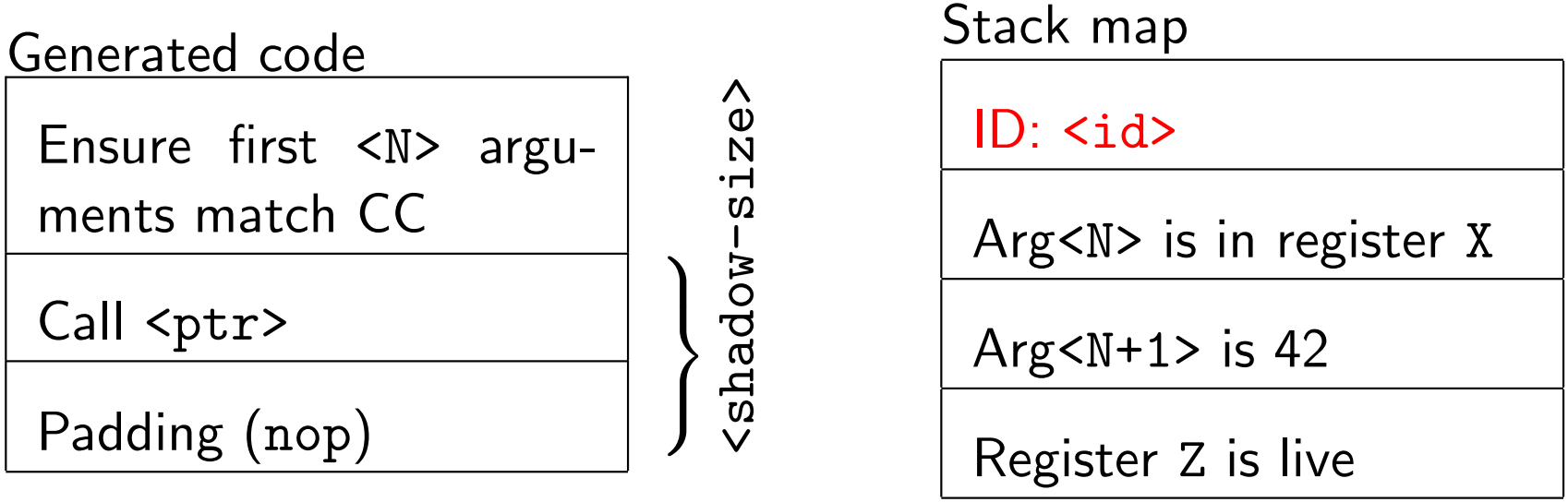
Stack map

ID: <id>
Arg<N> is in register X
Arg<N+1> is 42
Register Z is live

But how to create the function pointed to by <ptr>?

Patchpoints

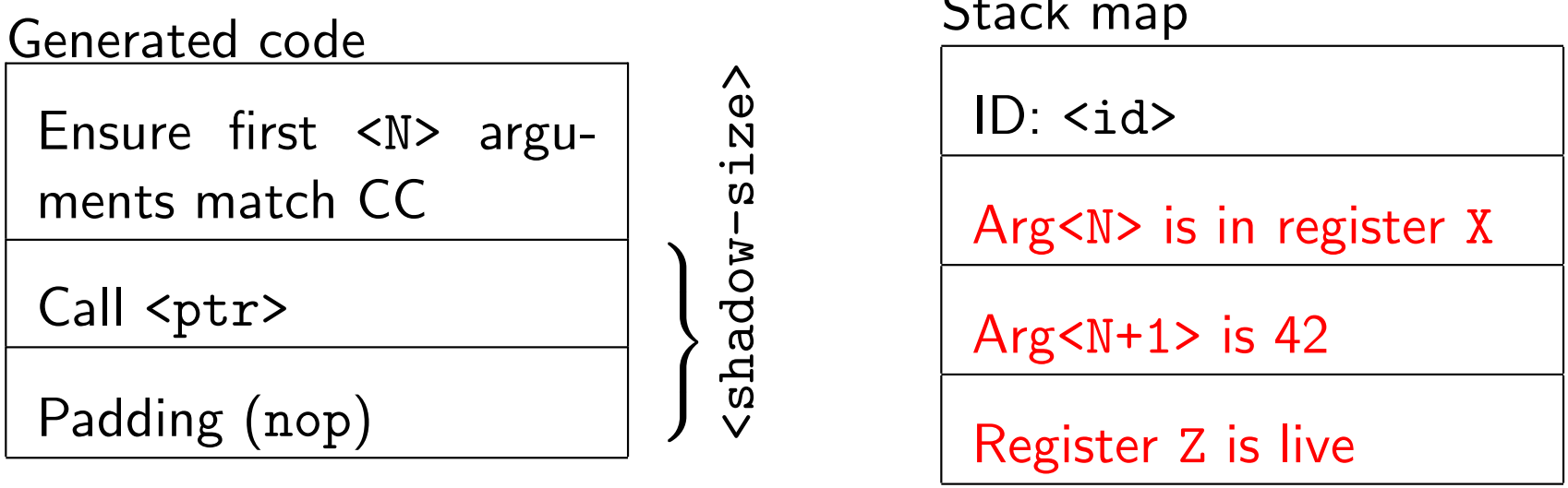
```
call void (i64, i32, i8*, i32, ...)*
      @llvm.experimental.patchpoint.void(
           i64 <id>, i32 <shadow-size>,i8* <ptr>,
           i32 <N>, arg0, arg1, ....)
```



But how to create the function pointed to by <ptr>?

Patchpoints

```
call void (i64, i32, i8*, i32, ...)*
      @llvm.experimental.patchpoint.void(
           i64 <id>, i32 <shadow-size>,i8* <ptr>,
           i32 <N>, arg0, arg1, ....)
```



But how to create the function pointed to by <ptr>?

The explicitcc Calling Convention

```
define explicitcc void @foo(
            i64 hwreg(35) %p0,   ; rax
            i64 hwreg(37) %p1)   ; rbx
            noclobber(38, 40) { ; rcx, rdx

    call void (...)* @llvm.experimental.retwr(
            i64 hwreg(37) %p0,
            i64 hwreg(35) %p1)
}
```

Register allocator is told where the arguments are and which registers to preserve.

`llvm.experimental.retwr` allows returning values in registers.

Use the same internal register ids as the `llvm.read_register` and `llvm.write_register` intrinsics.

The explicitcc Calling Convention

```
define explicitcc void @foo(
            i64 hwreg(35) %p0,   ; rax
            i64 hwreg(37) %p1)   ; rbx
            noclobber(38, 40) { ; rcx, rdx

    call void (...)* @llvm.experimental.retwr(
            i64 hwreg(37) %p0,
            i64 hwreg(35) %p1)
}
```

Register allocator is told where the arguments are and which registers to preserve.

`llvm.experimental.retwr` allows returning values in registers.

Use the same internal register ids as the `llvm.read_register` and `llvm.write_register` intrinsics.

The explicitcc Calling Convention

```
define explicitcc void @foo(
            i64 hwreg(35) %p0,   ; rax
            i64 hwreg(37) %p1)   ; rbx
            noclobber(38, 40) { ; rcx, rdx

    call void (...)* @llvm.experimental.retwr(
            i64 hwreg(37) %p0,
            i64 hwreg(35) %p1)
}
```

Register allocator is told where the arguments are and which registers to preserve.

`llvm.experimental.retwr` allows returning values in registers.

Use the same internal register ids as the `llvm.read_register` and `llvm.write_register` intrinsics.

The explicitcc Calling Convention

```
define explicitcc void @foo(
            i64 hwreg(35) %p0,   ; rax
            i64 hwreg(37) %p1)   ; rbx
            noclobber(38, 40) { ; rcx, rdx

    call void (...)* @llvm.experimental.retwr(
            i64 hwreg(37) %p0,
            i64 hwreg(35) %p1)
}
```

Register allocator is told where the arguments are and which registers to preserve.

`llvm.experimental.retwr` allows returning values in registers.

Use the same internal register ids as the `llvm.read_register` and `llvm.write_register` intrinsics.

The explicitcc Calling Convention

```
define explicitcc void @foo(
            i64 hwreg(35) %p0,   ; rax
            i64 hwreg(37) %p1)   ; rbx
            noclobber(38, 40) { ; rcx, rdx

    call void (...)* @llvm.experimental.retwr(
            i64 hwreg(37) %p0,
            i64 hwreg(35) %p1)
}
```

Register allocator is told where the arguments are and which registers to preserve.

`llvm.experimental.retwr` allows returning values in registers.

Use the same internal register ids as the `llvm.read_register` and `llvm.write_register` intrinsics.

Try it out

Complete patch series

http://reviews.llvm.org/D8953
http://reviews.llvm.org/D8954
http://reviews.llvm.org/D8955
http://reviews.llvm.org/D8956
http://reviews.llvm.org/D8957
http://reviews.llvm.org/D8959
http://reviews.llvm.org/D8960
http://reviews.llvm.org/D8961

Example

```
define explicitcc void @ex0(
  i64 hwreg(35) %p0, ; rax
  i64 hwreg(37) %p1) ; rbx
  noclobber() {
  call void (...)* @retwr(
    i64 hwreg(37) %p0,
    i64 hwreg(35) %p1)
  unreachable
}
```

Example

```
define explicitcc void @ex0(
  i64 hwreg(35) %p0, ; rax
  i64 hwreg(37) %p1) ; rbx
  noclobber() {
  call void (...)* @retwr(
    i64 hwreg(37) %p0,
    i64 hwreg(35) %p1)
  unreachable
}
```

```
ex0:
  movq    %rbx, %rcx
  movq    %rax, %rbx
  movq    %rcx, %rax
  retq
```

Example

```
define explicitcc void @ex0(
  i64 hwreg(35) %p0, ; rax
  i64 hwreg(37) %p1) ; rbx
  noclobber() {
  call void (...)* @retwr(
    i64 hwreg(37) %p0,
    i64 hwreg(35) %p1)
  unreachable
}
```

```
ex0:
  movq    %rbx, %rcx
  movq    %rax, %rbx
  movq    %rcx, %rax
  retq
```

```
define explicitcc void @ex1() {
  noclobber(38) {       ; rcx
  call void asm sideeffect
    "call foo", "~{rcx}"()
  ret
}
```

Example

```
define explicitcc void @ex0(
  i64 hwreg(35) %p0, ; rax
  i64 hwreg(37) %p1) ; rbx
  noclobber() {
  call void (...)* @retwr(
    i64 hwreg(37) %p0,
    i64 hwreg(35) %p1)
  unreachable
}
```

```
ex0:
  movq    %rbx, %rcx
  movq    %rax, %rbx
  movq    %rcx, %rax
  retq
```

```
define explicitcc void @ex1() {
  noclobber(38) {       ; rcx
  call void asm sideeffect
    "call foo", "~{rcx}"()
  ret
}
```

```
ex1:
  pushq   %rcx
  callq   foo
  popq    %rcx
  retq
```