Native Support for Explicit Stacks in LLVM

Kavon Farvardin
*University of Chicago**

Simon Peyton Jones
Microsoft Research

9 September 2017

**This work was started while at Microsoft Research*

GHC's LLVM Backend

- It often produces faster programs (~7%) than native codegen.
- The only option for ARM systems.
- Many of LLVM's optimizations are ineffective.

Problem

GHC is forced to emit unusual LLVM code,
which evades analysis.

Mismatch: LLVM's Implicit Stack

```
blockA:
    ...
    x = a + 2
    y = call f (b)
    z = x * y
    ...
```

x is live across this call

How will x be preserved?

Mismatch: Cmm has an explicit stack

```
blockA:
   ...
   x = a + 2
   Sp[8] = x
   Sp[0] = &returnPoint
   tailcall f (b, Sp)
```

x is saved to stack

a block's address

physical return reg

```
returnPoint:
   y = R1
   x = Sp[8]
   z = x * y
   ...
```

Mismatch Consequences

- GHC must break **every** return point into a new LLVM function.
  - Every allocating function is split into pieces.
  - Loops within a function are effectively destroyed.

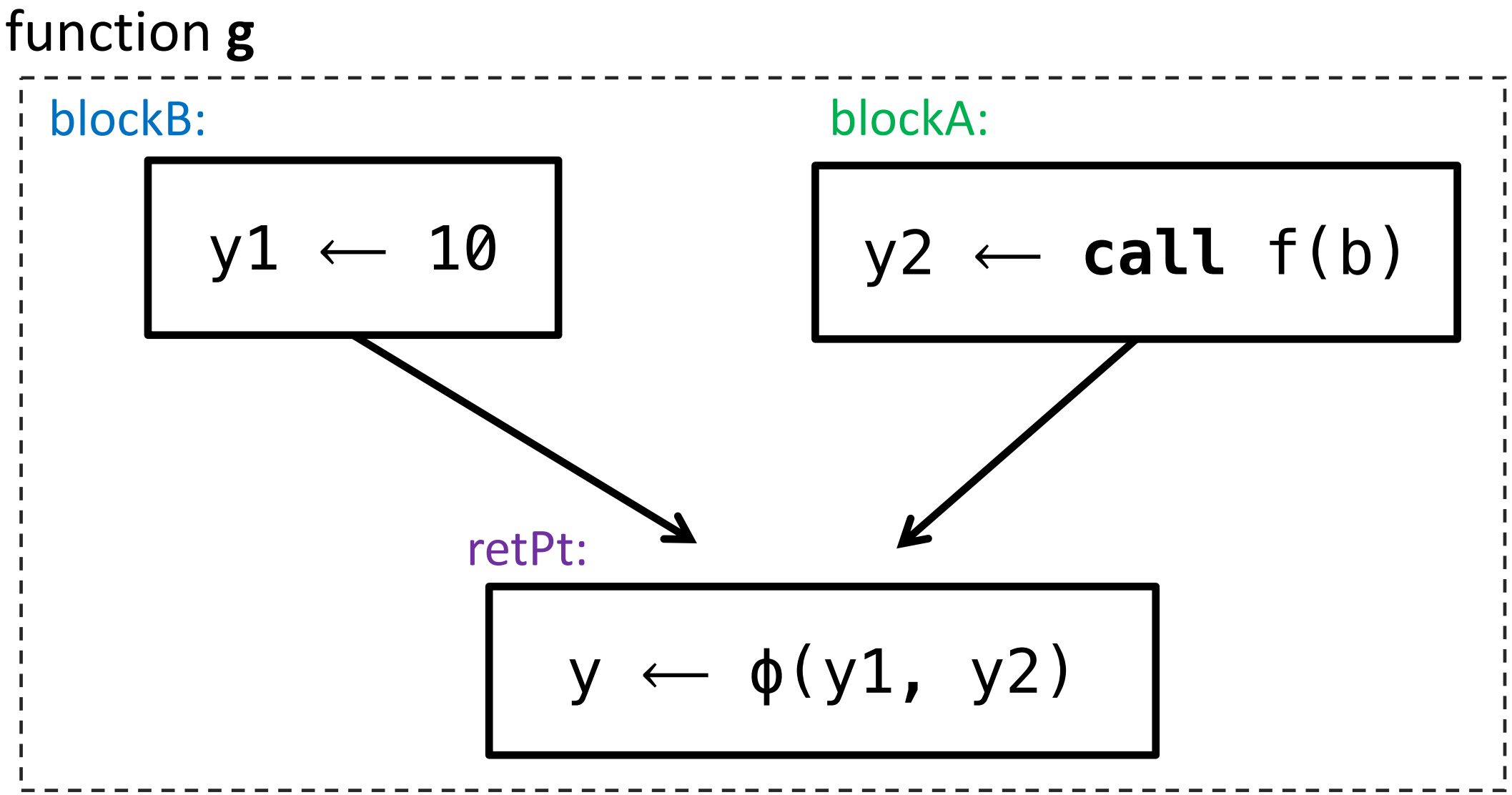
- LLVM does not handle it well.
  - Loop optimizations no longer apply.
  - Inliner is befuddled and tries to piece together the program.

Advantages of an Explicit Stack

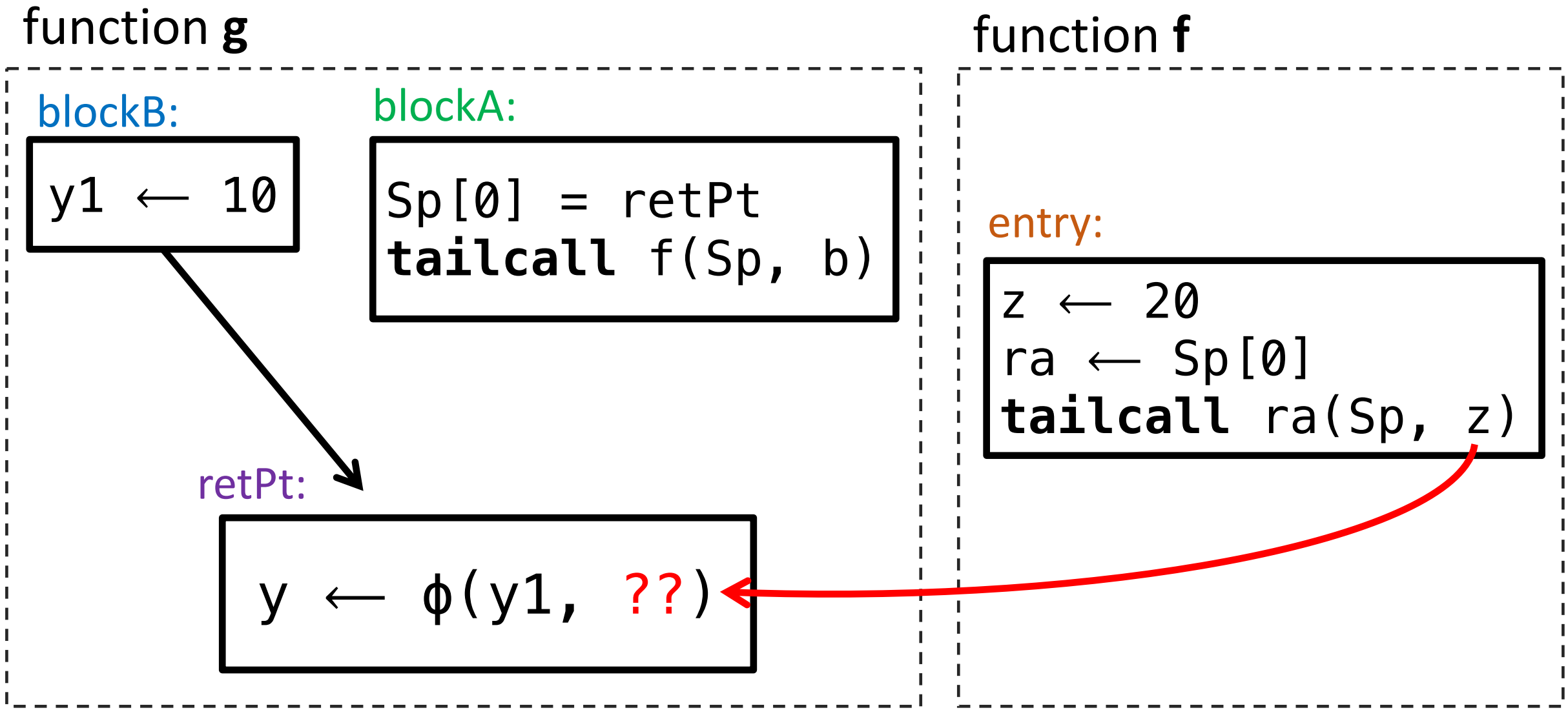
- Precise Garbage Collection
  - Stack overflow occurs regularly, must be recoverable.
- Lightweight Concurrency
  - Millions of heap-allocated stacks!
  - Express one-shot continuations for lightweight threading
- Other customized implementations of
  - Exception handling
  - Argument passing
  - …

Upgrading LLVM

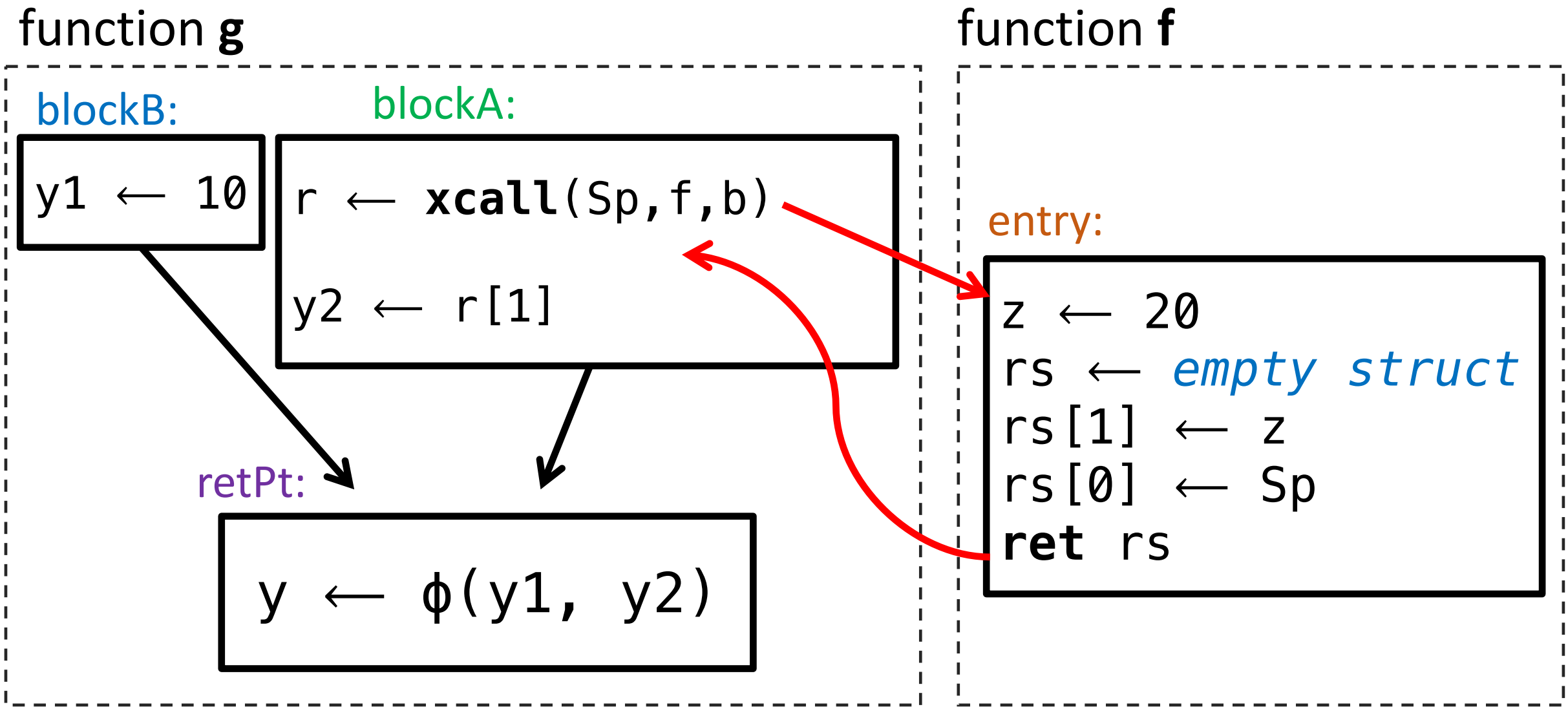
Typical LLVM Control-flow Join



Unworkable Explicit Stack Usage in LLVM



Proposal: a new call instruction "**xcall**"



Overview of the “xcall” instruction.

```
declare %someTy @llvm.xcall(
        i64*,
        {i64*,…} (i64*,…)*, ...)
```

Stack Pointer

Function to be called

Other arguments passed to function

Some details omitted for clarity.

Properties of the xcall instruction.

```
rv ← xcall (Sp, f, b)
...
```

- Callee must be an “xcalled” function.
- The word pointed to by **Sp** must be free to use.
- The LLVM stack will not change.
- Otherwise, an **xcall** behaves like a normal LLVM call.

Liveness Requirement

```
blockA:
   ...
   x = a + 2
   Sp[8] = x
   r = xcall (Sp,f,b)
   Sp' = getVal 0, r
   x' = Sp'[8]
   ...
```

An xcall passes all of the live values!

(also a property of CPS calls)

Thus, we disallow having values live across an xcall.

Lowering xcall to Assembly

```
blockA:
  ...
  x = a + 2
  Sp[8] = x
  r = xcall f (Sp, b)
  Sp' = getVal 0, r
  x' = Sp'[8]
  ...
```

```
store &retpt, Sp[off]
jump f
```

expand

split

This becomes a new block, 'retpt', with no predecessor.

Properties of an xcalled function.

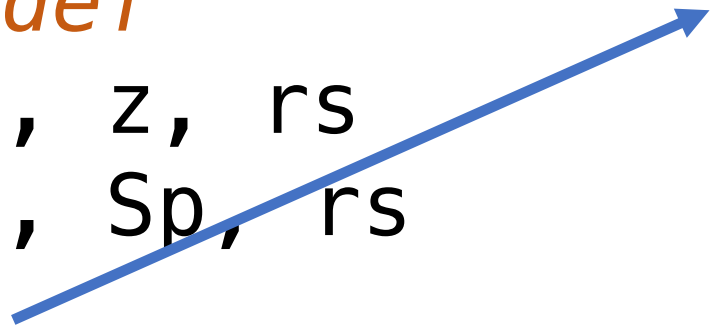
```
define cc _ @f(Sp, b) xcalled {
..
  ret {i64*,…} vals
```

- **Sp[0]** must not be overwritten.
- **Sp'** must point to the same value as **Sp**, where **Sp'** = vals[0].
- The calling convention must pass all values in register.
- Otherwise, the **ret** behaves like a normal return.

Lowering xret to Assembly

```
blockA:
  ...
  z ← 20
  rs ← undef
  setVal 1, z, rs
  setVal 0, Sp, rs
  xret rs
```

```
asm_blockA:
  ...
  mov 20, r1
  jump *(r0)
```



Sp is in r0 as defined
by the calling convention

Evaluation

LLVM Backend Comparison (GHC)

Program performance changes
when using xcall instead of “proc-point splitting”.

Program	Size	Allocs	Runtime	Elapsed	TotalMem
Min	−8.5%	−0.0%	−6.6%	−6.7%	−3.3%
Max	0.0%	+0.2%	+4.8%	+5.0%	0.0%
Geo Mean	−1.3%	+0.0%	−0.5%	−0.7%	−0.0%

WIP Notes:

- 5 of 107 programs do not work yet.
- xcall programs are hampered by temporary workarounds that disable some optimizations.

Future Work

Optimizing under the Liveness Requirement

```
x ← 8
Sp[8] ← x
r ← xcall(Sp,f,b)
Sp’ ← r[0]
x’ ← Sp’[8]
```

```
x ← 8
xstore(Sp,8,x)
r ← xcall(Sp,f,b)
Sp’ ← r[0]
x’ ← xload(Sp,8)
```

Remaining Work

1. Stabilize and write-up the xcall design and implementation.
2. Submit patch to upstream LLVM.
3. Once the patch has landed in an LLVM release, merge into GHC.

Branch on GHC git: wip/kavon-nosplit-llvm
Depends on: github.com/kavon/ghc-llvm

Contributions Welcome!