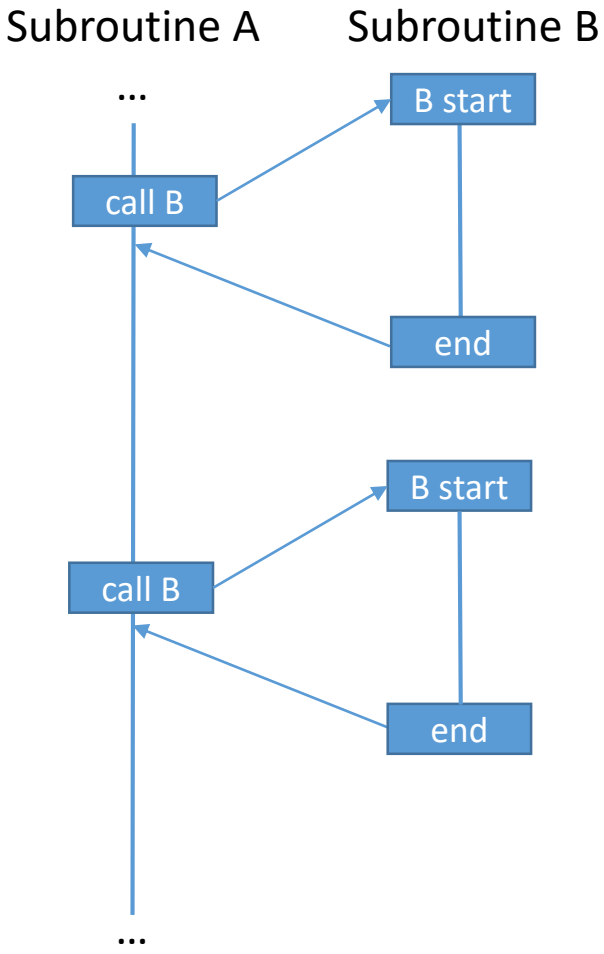
LLVM Coroutines

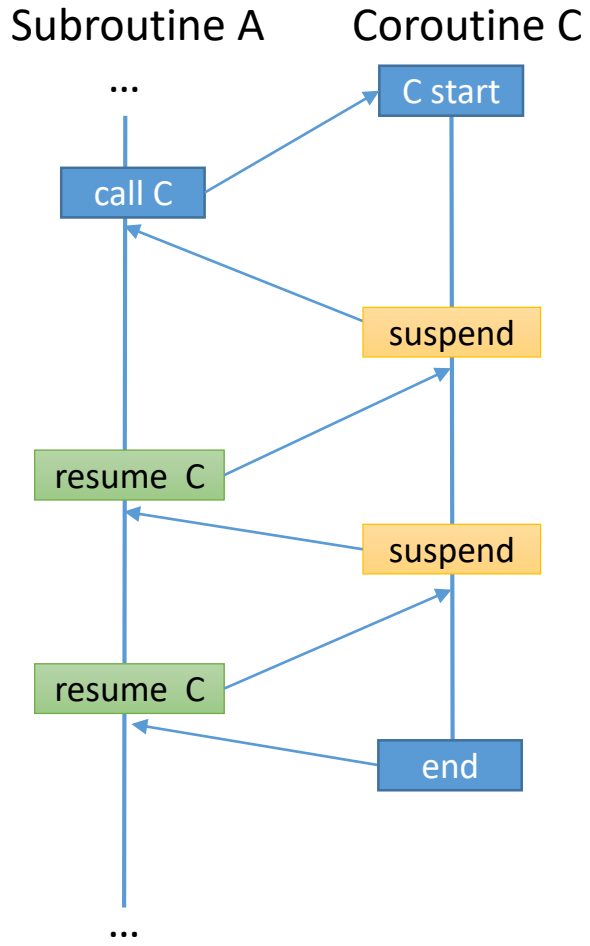
Bringing resumable functions to LLVM

Coroutines



- Introduced in 1958 by Melvin Conway
- Donald Knuth, 1968: “generalization of subroutine”

	subroutines	coroutines
call	Allocate frame, pass parameters	Allocate frame, pass parameters
return	Free frame, return result	Free frame, return eventual result
suspend	x	yes
resume	x	yes



Only with Coroutines. 100 cards per minute!

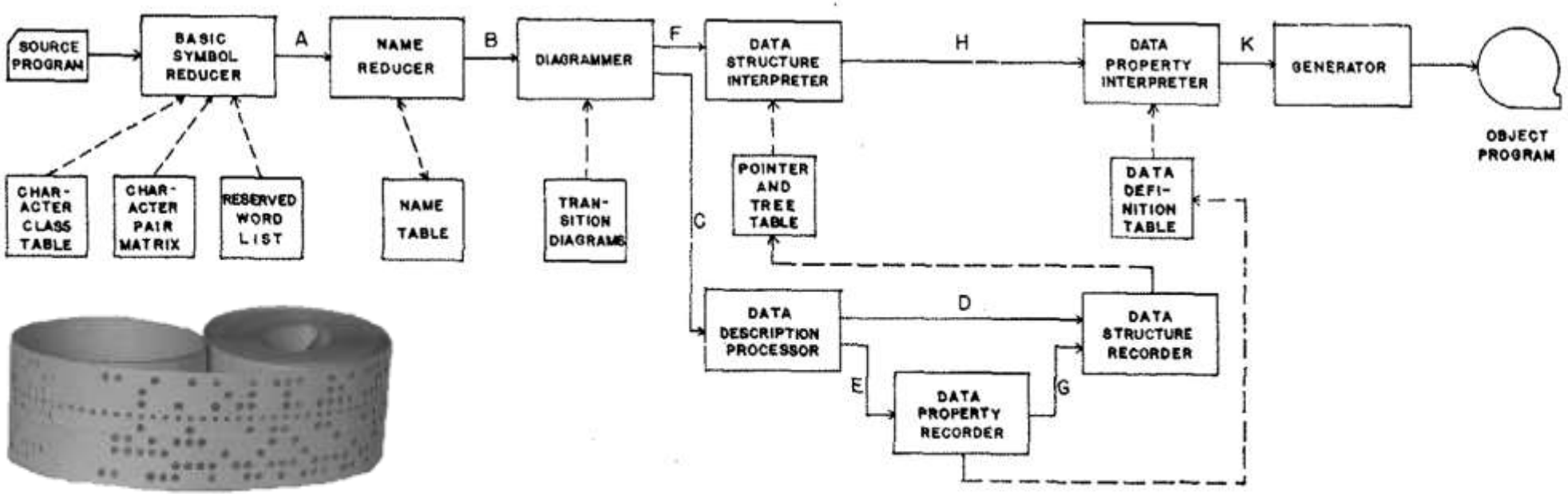
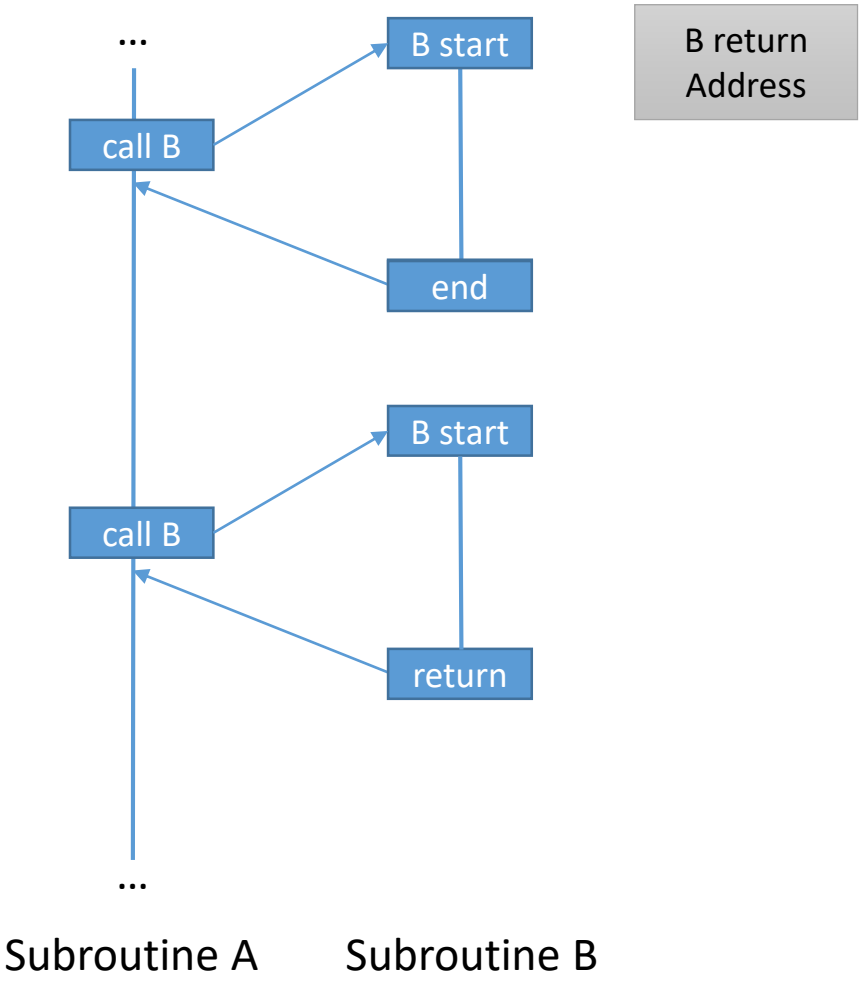


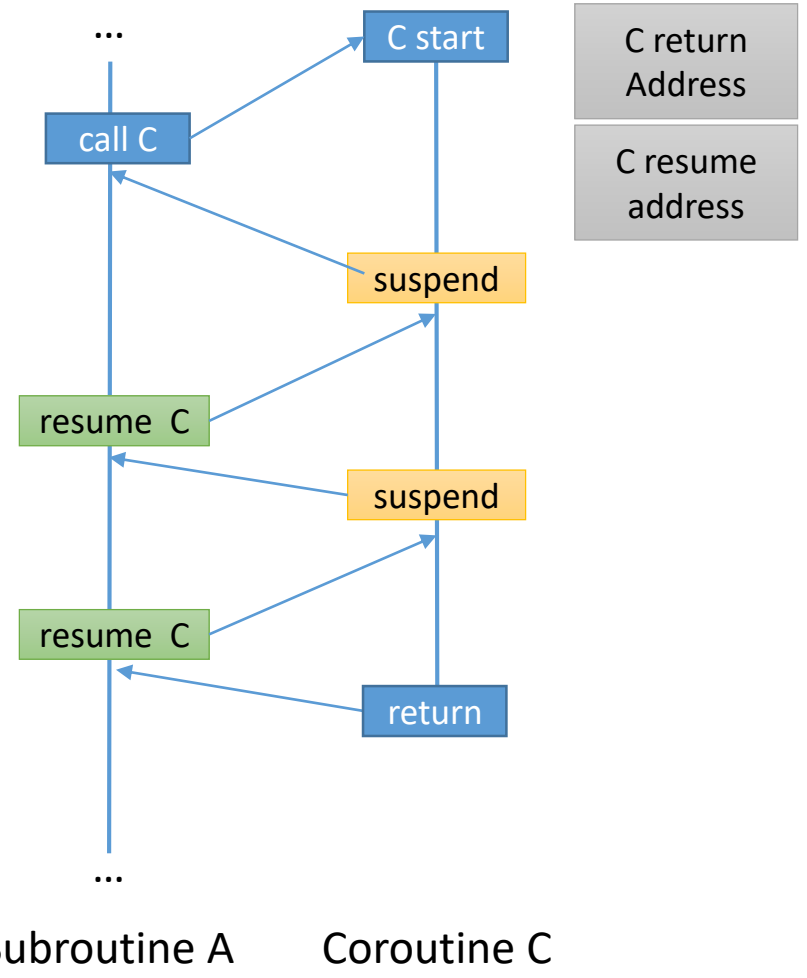
FIG. 4. COBOL Compiler Organization

Communications of the ACM

Volume 6 / Number 7 / July, 1963

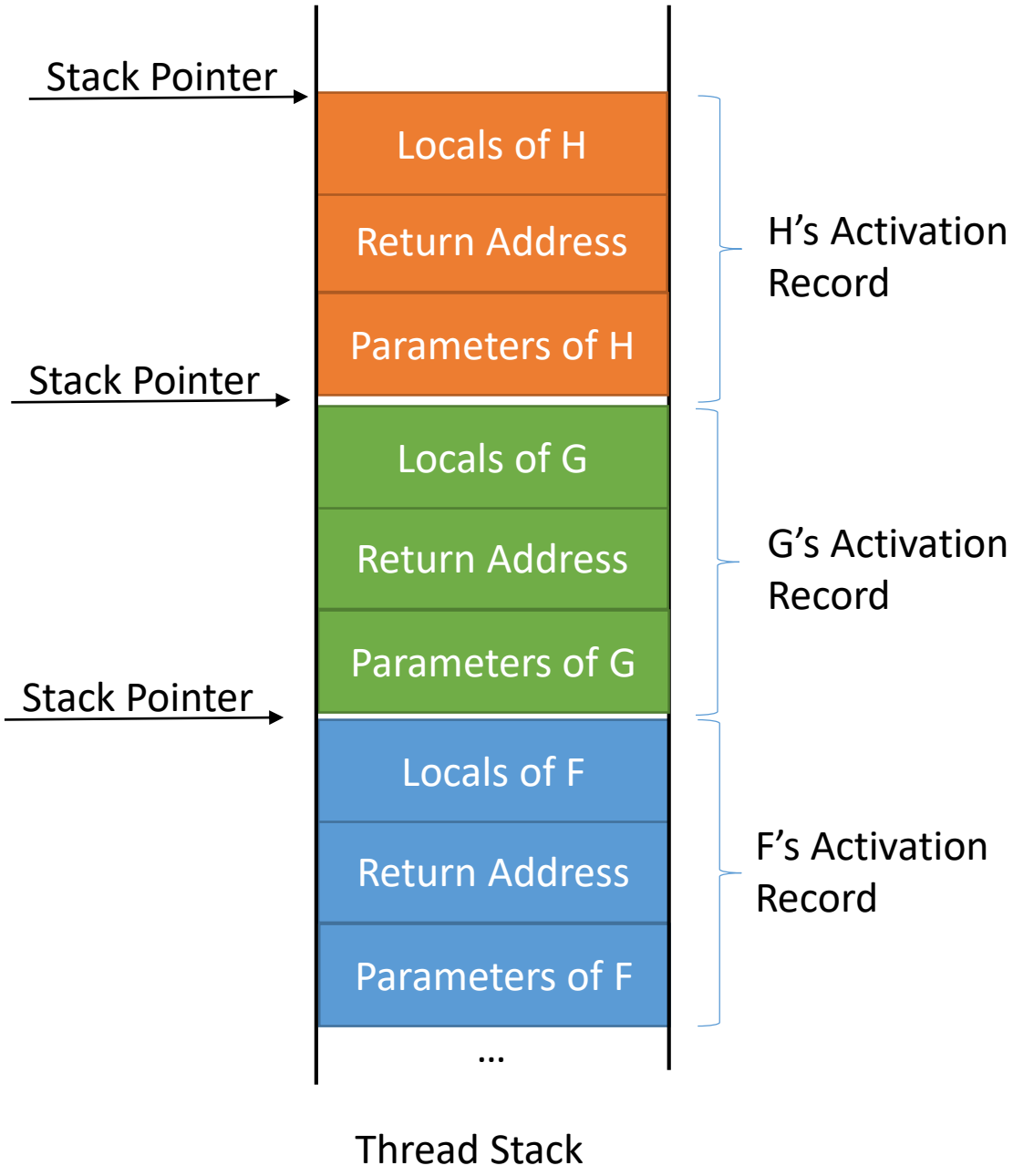
Subroutines vs Coroutines



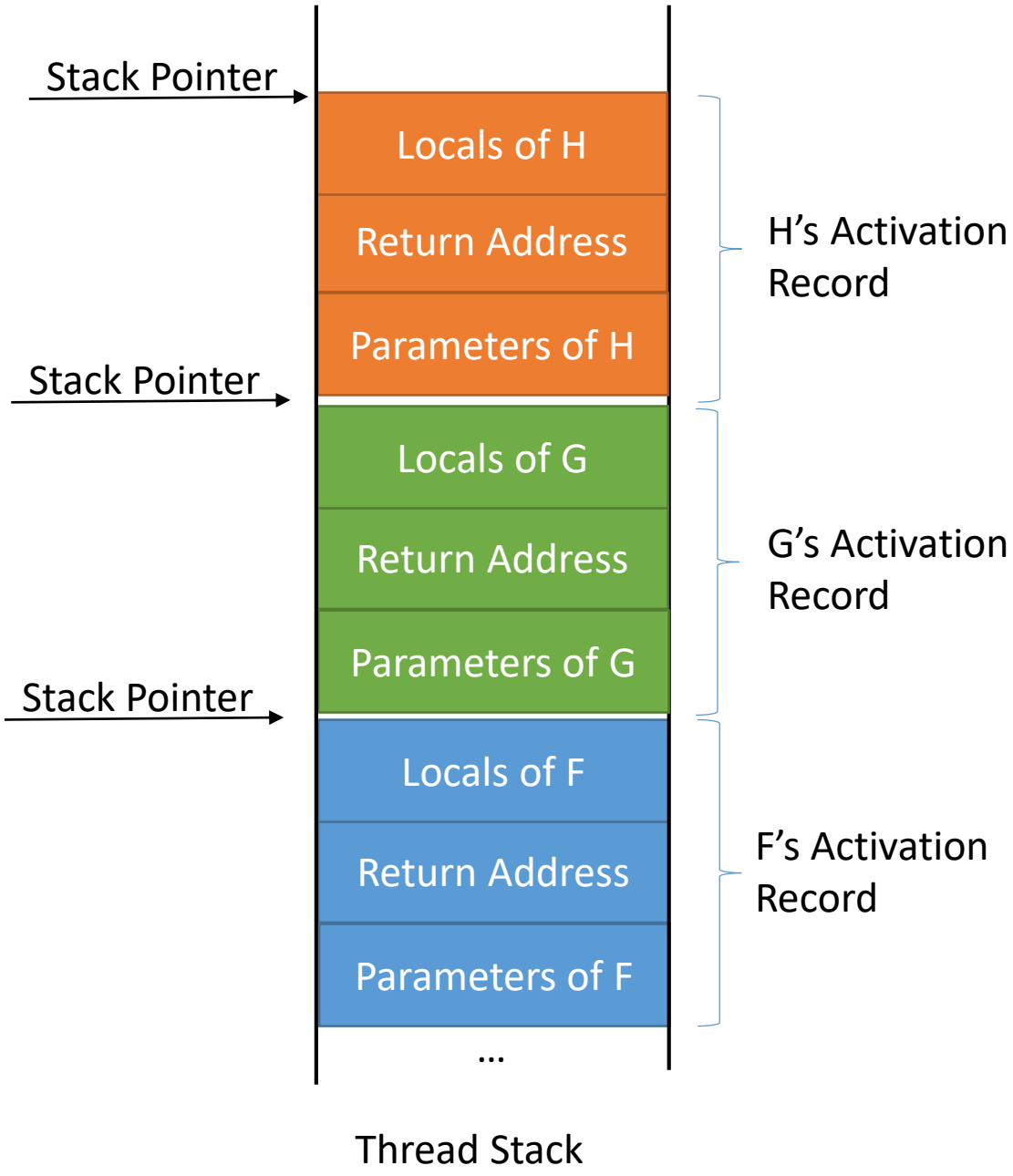


Algol-60

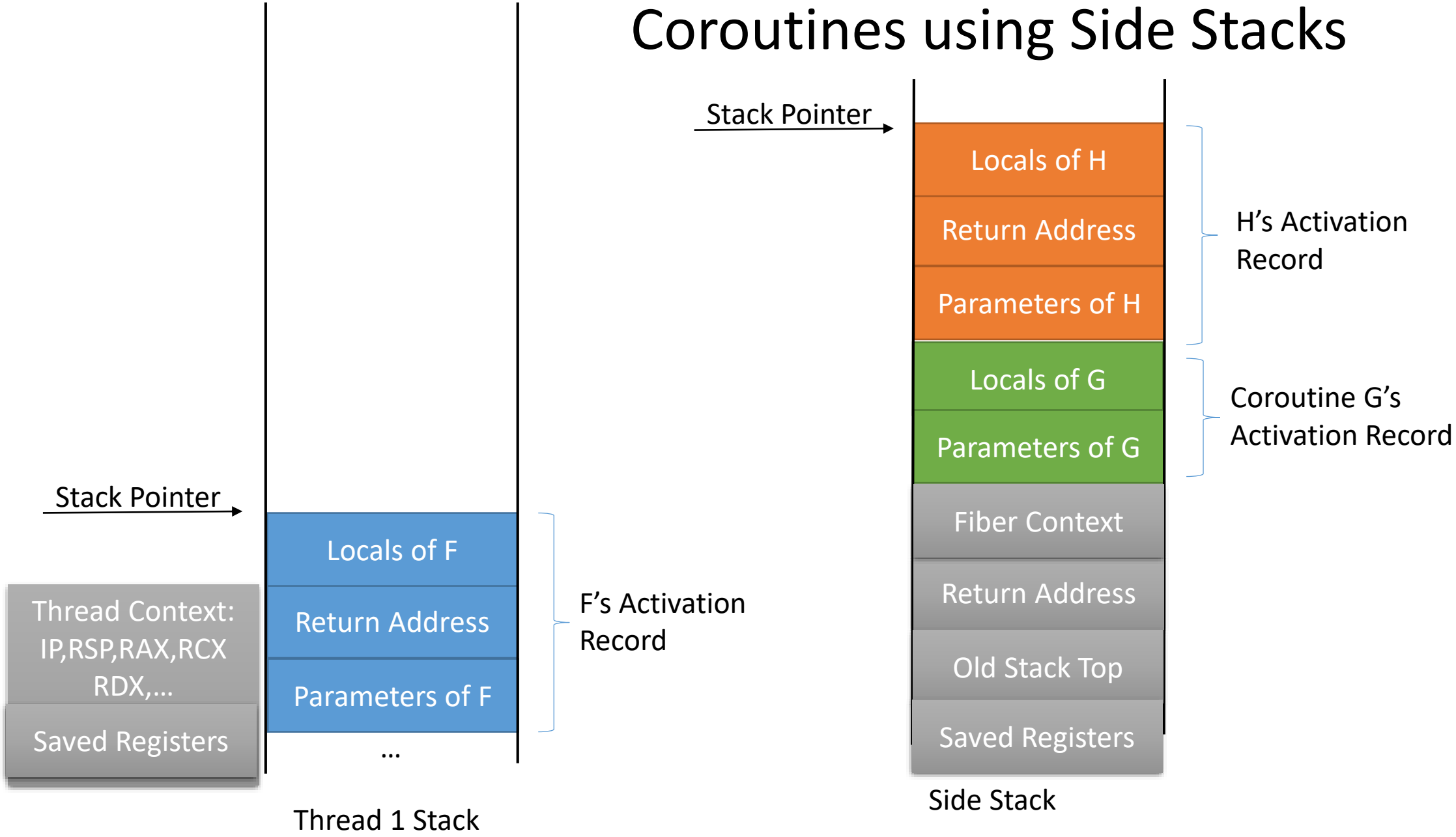
Normal Functions



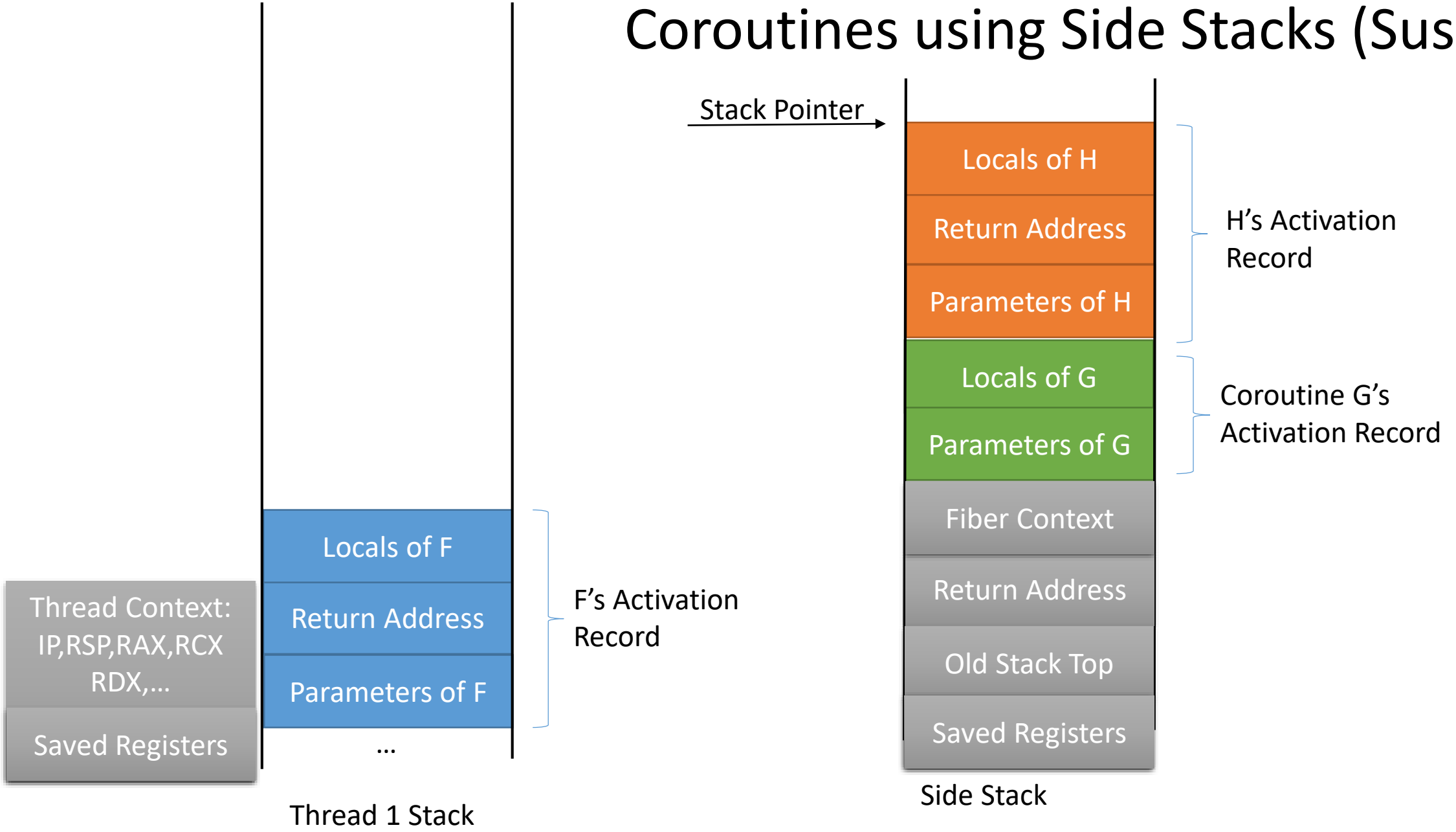
Normal Functions



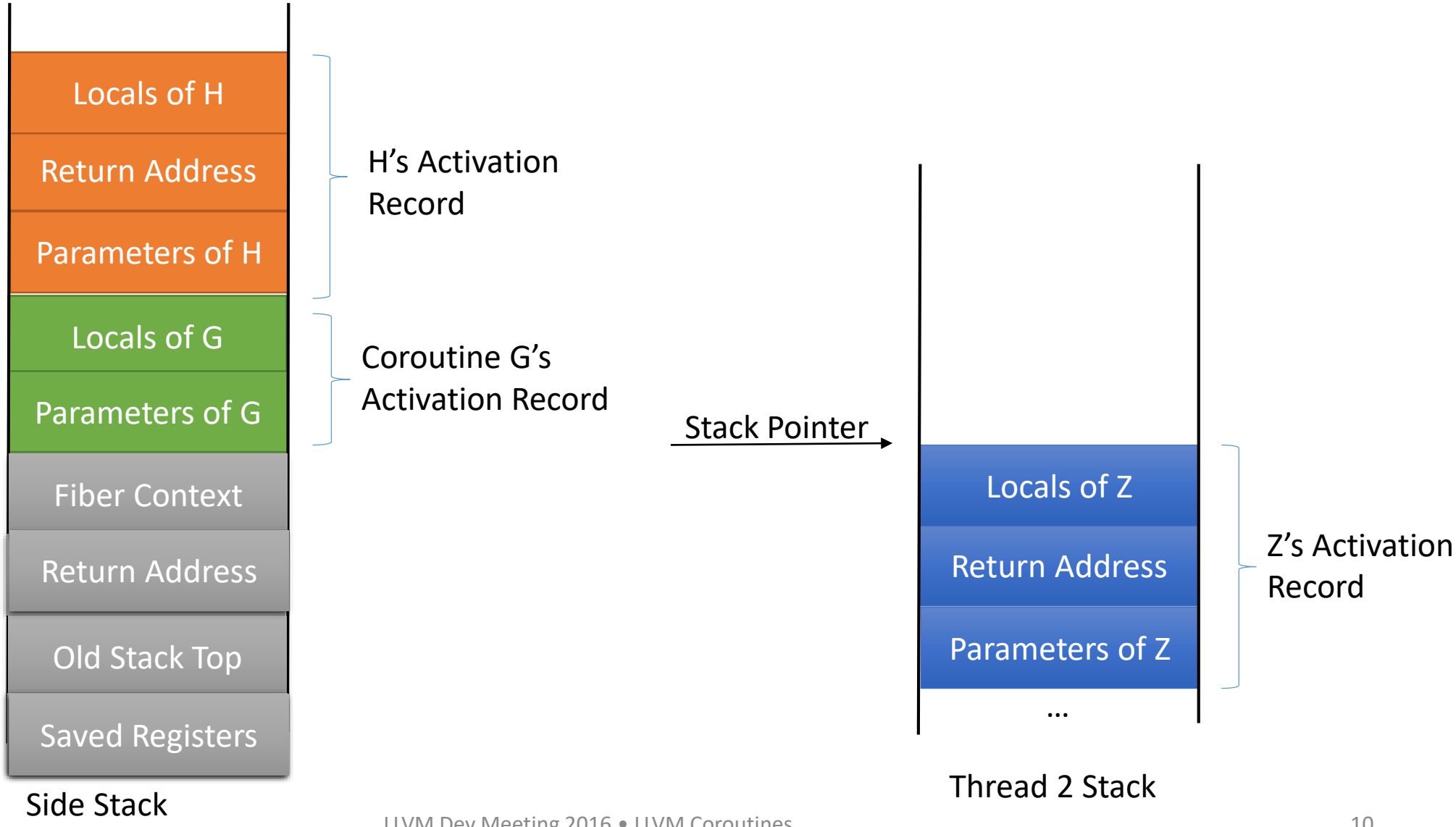
Coroutines using Side Stacks



Coroutines using Side Stacks (Suspend)



Coroutines using Side Stacks (Resume)



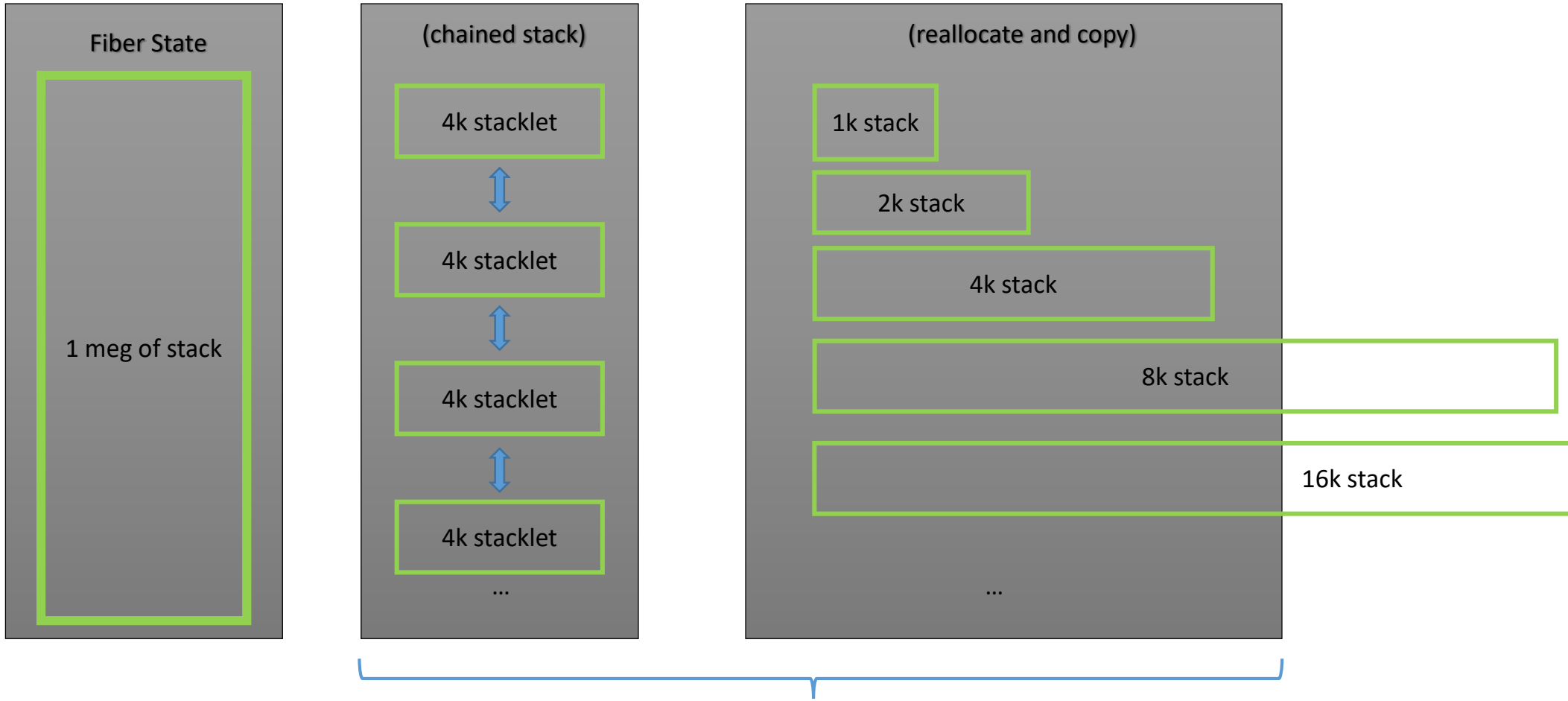
https://github.com/mirror/boost/blob/master/libs/context/src/asm/jump_x86_64_ms_pe_masm.asm (1/2)

```
jump_fcontext PROC EXPORT FRAME
    .endprolog

    mov     [rcx],      r12         ; save R12
    mov     [rcx+08h],  r13         ; save R13
    mov     [rcx+010h], r14         ; save R14
    mov     [rcx+018h], r15         ; save R15
    mov     [rcx+020h], rdi         ; save RDI
    mov     [rcx+028h], rsi         ; save RSI
    mov     [rcx+030h], rbx         ; save RBX
    mov     [rcx+038h], rbp         ; save RBP

    mov     r10,        gs:[030h]   ; load NT_TIB
    mov     rax,        [r10+08h]   ; load current stack base
    mov     [rcx+050h], rax         ; save current stack base
    mov     rax,        [r10+010h]  ; load current stack limit
    mov     [rcx+060h], rax         ; save current stack limit
    mov     rax,        [r10+01478h] ; load current deallocation stack
    mov     [rcx+0130h], rax        ; save current deallocation stack
    mov     rax,        [r10+018h]  ; load fiber local storage
    mov     [rcx+068h], rax         ; save fiber local storage

    test    r9,         r9
    je      nxt

    stmxcsr [rcx+070h]              ; save MMX control and
    fnstcw  [rcx+074h]              ; save x87 control word
    ; save XMM storage
    ; save start address of SSE register block in R10
    lea     r10,          [rcx+090h]
    ; shift address in R10 to lower 16 byte boundary
    ; == pointer to SEE register block
    and     r10,        -16

    movaps  [r10],      xmm6
    movaps  [r10+010h], xmm7
    movaps  [r10+020h], xmm8
    movaps  [r10+030h], xmm9
    movaps  [r10+040h], xmm10
    movaps  [r10+050h], xmm11
    movaps  [r10+060h], xmm12
    movaps  [r10+070h], xmm13
    movaps  [r10+080h], xmm14
    movaps  [r10+090h], xmm15

    ldmxcsr [rdx+070h]              ; restore MMX control a
    fldcw   [rdx+074h]              ; restore x87 control w
```

https://github.com/mirror/boost/blob/master/libs/context/src/asm/jump_x86_64_ms_pe_masm.asm (2/2)

```
    ; restore XMM storage
        ; save start address of SSE register block in R10
    lea     r10,            [rdx+090h]
    ; shift address in R10 to lower 16 byte boundary
    ; == pointer to SEE register block
    and     r10,            -16

    movaps  xmm6,           [r10]
    movaps  xmm7,           [r10+010h]
    movaps  xmm8,           [r10+020h]
    movaps  xmm9,           [r10+030h]
    movaps  xmm10,          [r10+040h]
    movaps  xmm11,          [r10+050h]
    movaps  xmm12,          [r10+060h]
    movaps  xmm13,          [r10+070h]
    movaps  xmm14,          [r10+080h]
    movaps  xmm15,          [r10+090h]

nxt:
    lea     rax,            [rsp+08h]   ; exclude the return address
    mov     [rcx+040h],     rax         ; save as stack pointer
    mov     rax,            [rsp]       ; load return address
    mov     [rcx+048h],     rax         ; save return address

    mov     r12,            [rdx]       ; restore R12
    mov     r13,            [rdx+08h]   ; restore R13
    mov     r14,            [rdx+010h]  ; restore R14
    mov     r15,            [rdx+018h]  ; restore R15
    mov     rdi,            [rdx+020h]  ; restore RDI
    mov     rsi,            [rdx+028h]  ; restore RSI
    mov     rbx,            [rdx+030h]  ; restore RBX
    mov     rbp,            [rdx+038h]  ; restore RBP

    mov     r10,            gs:[030h]   ; load NT_TIB
    mov     rax,            [rdx+050h]  ; load stack base
    mov     [r10+08h],      rax         ; restore stack base
    mov     rax,            [rdx+060h]  ; load stack limit
    mov     [r10+010h],     rax         ; restore stack limit
    mov     rax,            [rdx+0130h] ; load deallocation stack
    mov     [r10+01478h],   rax         ; restore deallocation stack
    mov     rax,            [rdx+068h]  ; load fiber local storage
    mov     [r10+018h],     rax         ; restore fiber local storage

    mov     rsp,            [rdx+040h]  ; restore RSP
    mov     r10,            [rdx+048h]  ; fetch the address to returned to

    mov     rax,            r8          ; use third arg as return value after jump
    mov     rcx,            r8          ; use third arg as first arg in context fur

    jmp     r10                         ; indirect jump to caller
jump_fcontext ENDP
```

Memory Footprint



Compiler based coroutines

```
generator<int> f() {
  for (int i = 0; i < 5; ++i) {
     co_yield i;
}
```

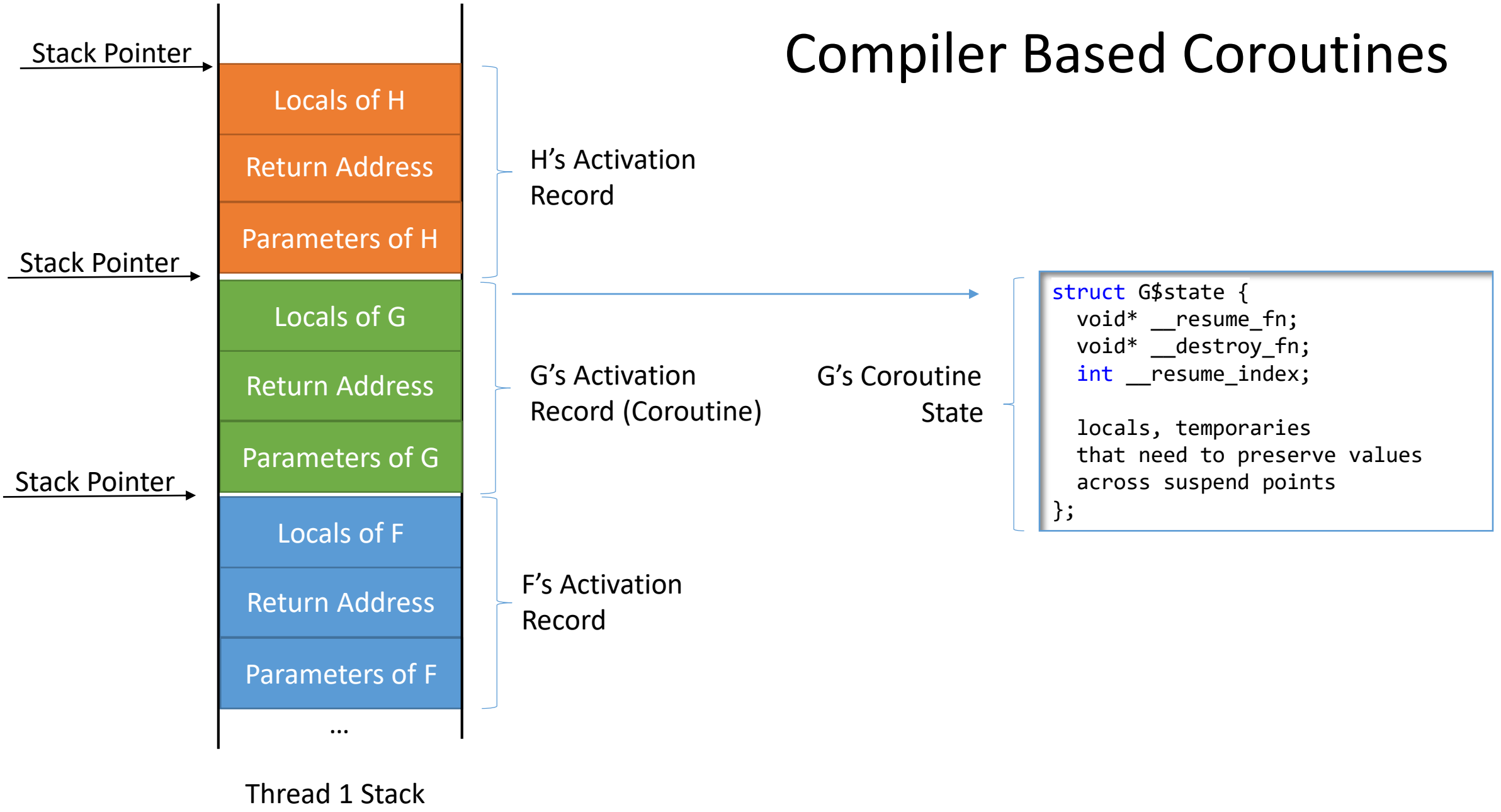
```
generator<int> f() {
  f.state *mem = new f$state;
  mem->__resume_fn = &f$resume;
  mem->__destroy_fn = &f$destroy;
  return {mem};
}
```

```
struct f$state {
  void *__resume_fn;
  void *__destroy_fn;
  int __resume_index = 0;
  int i, __current_value;
};
```

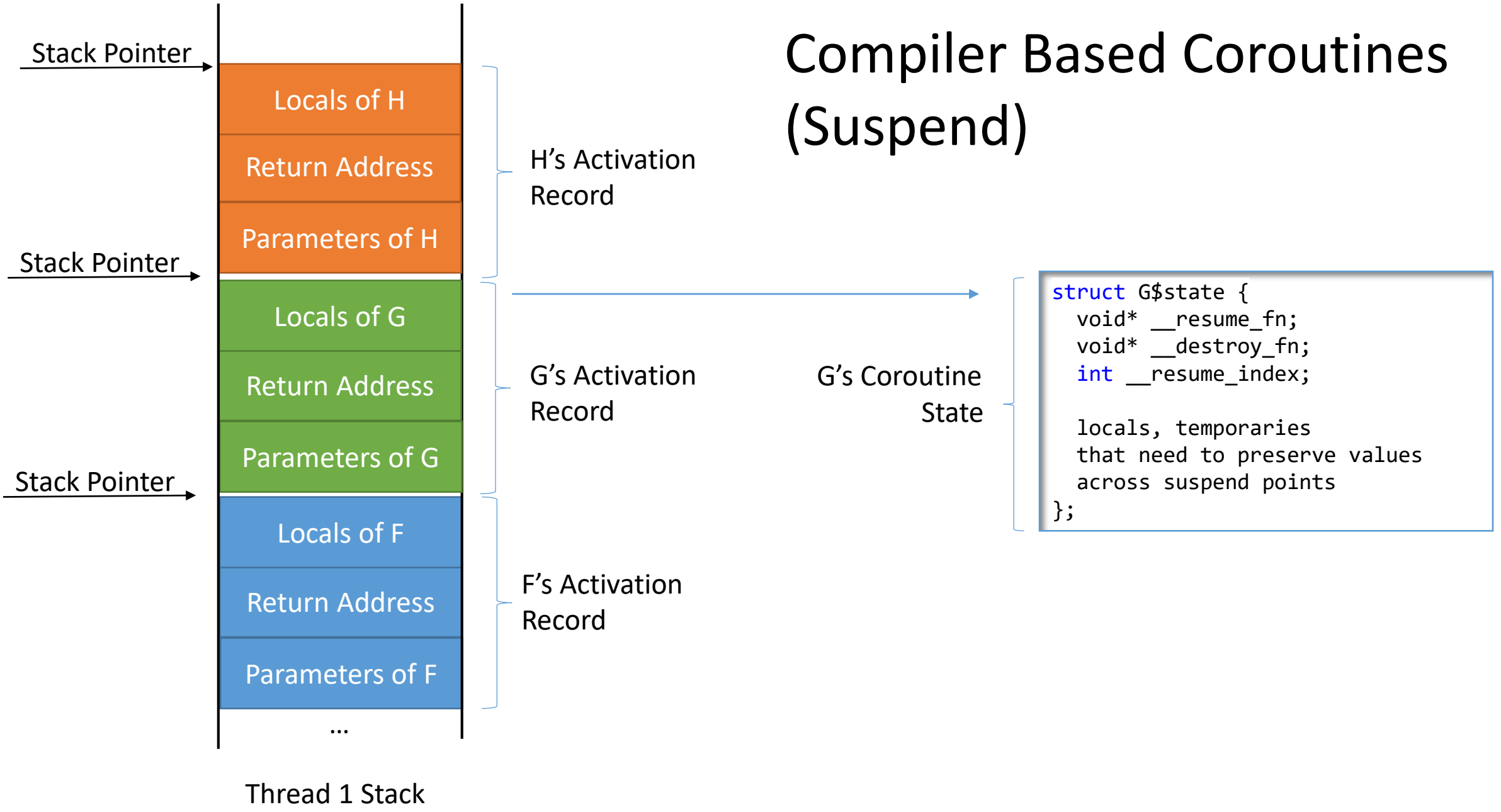
```
void f$resume(f$state *s) {
  switch (s->__resume_index) {
     case 0: s->i = 0; s->resume_index = 1; break;
     case 1: if( ++s->i == 5) { s->resume_index = 2; return; }
  }
  s->__current_value = s->i;
}
```

```
void f$destroy(f$state *s) {
  delete s;
}
```

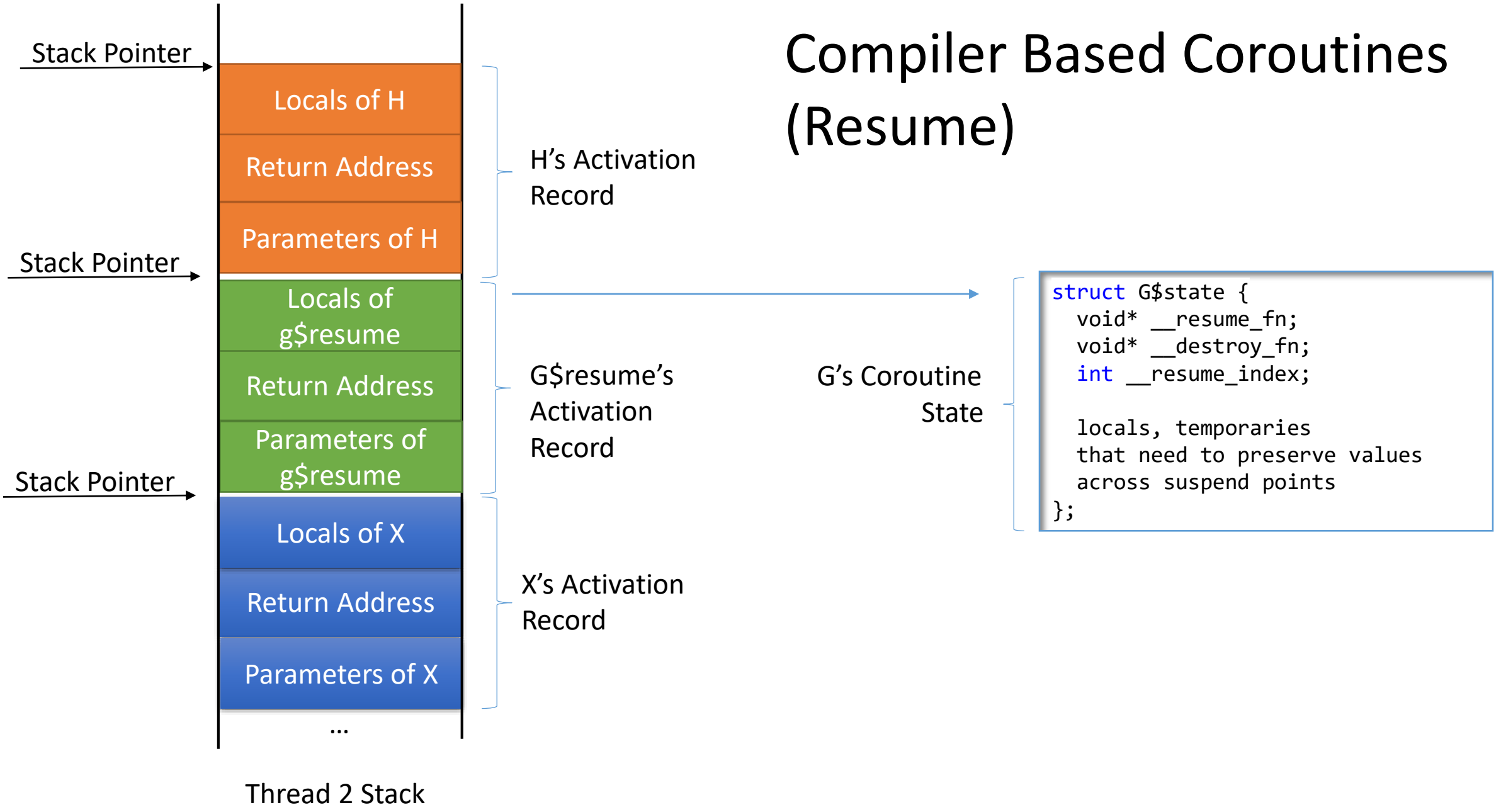
Compiler Based Coroutines



Compiler Based Coroutines (Suspend)



Compiler Based Coroutines (Resume)



Compiler based coroutines

```
generator<int> f() {
  for (int i = 0; i < 5; ++i) {
     co_yield i;
}
```

```
int main() {
  for (int v: f())
    printf("%d\n", v);
}
```

```
int main() {
   printf("%d\n", 0);
   printf("%d\n", 1);
   printf("%d\n", 2);
   printf("%d\n", 3);
   printf("%d\n", 4);
}
```

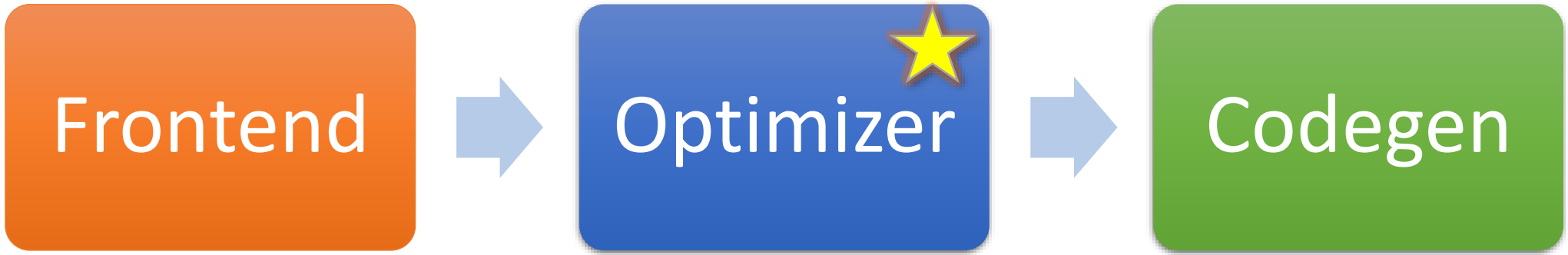
```
generator<int> f() {
  f.state *mem = new f$state;
  mem->__resume_fn = &f$resume;
  mem->__destroy_fn = &f$destroy;
  return {mem};
}
```

```
struct f$state {
  void *__resume_fn;
  void *__destroy_fn;
  int __resume_index = 0;
  int i, __current_value;
};
```

```
void f$resume(f$state *s) {
  switch (s->__resume_index) {
     case 0: s->i = 0; s->resume_index = 1; break;
     case 1: if( ++s->i == 5) { s->resume_index = 2; return; }
  }
  s->__current_value = s->i;
}
```

```
void f$destroy(f$state *s) {
  delete s;
}
```

Where would you split a coroutine?

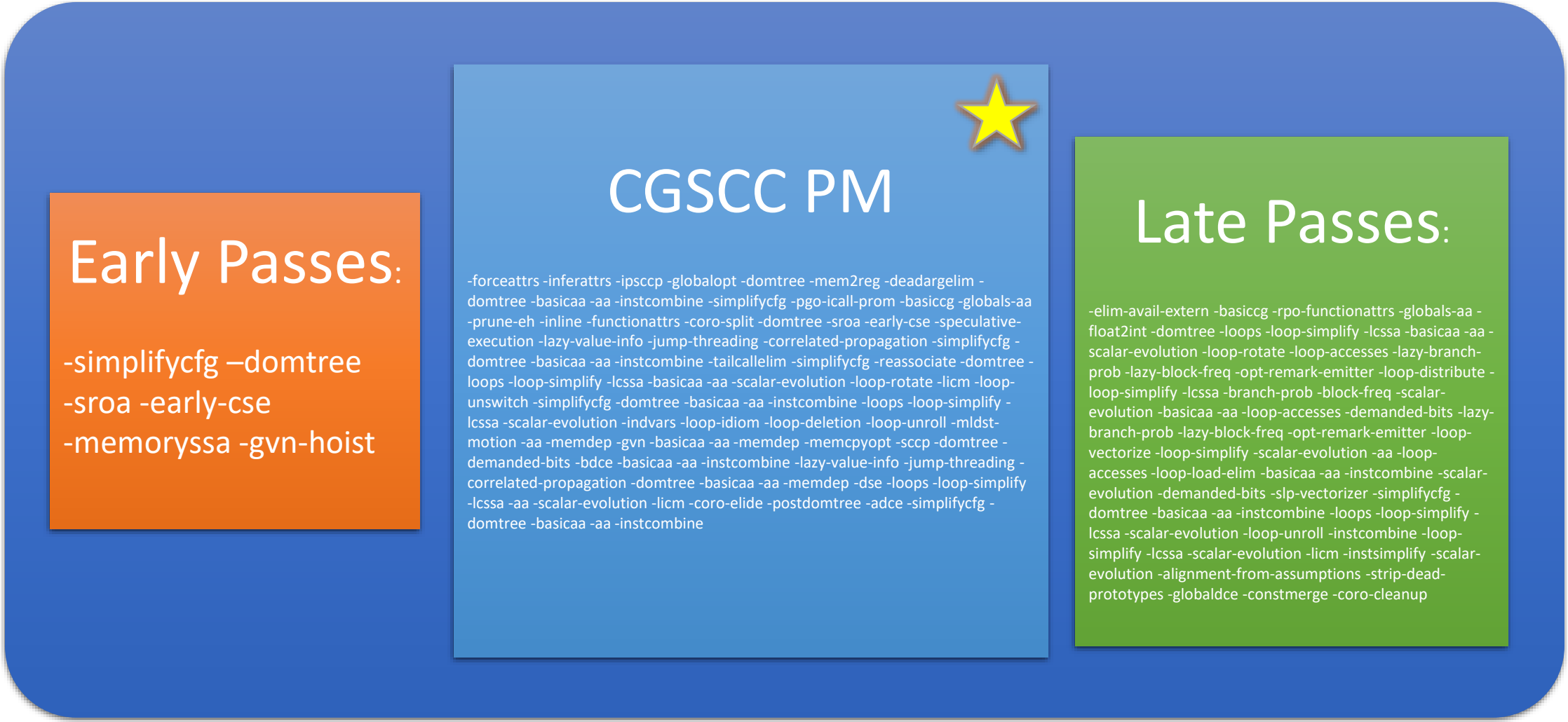



```
generator<int> seq(int start) {
  for (;;)
    co_yield start++;
}
```

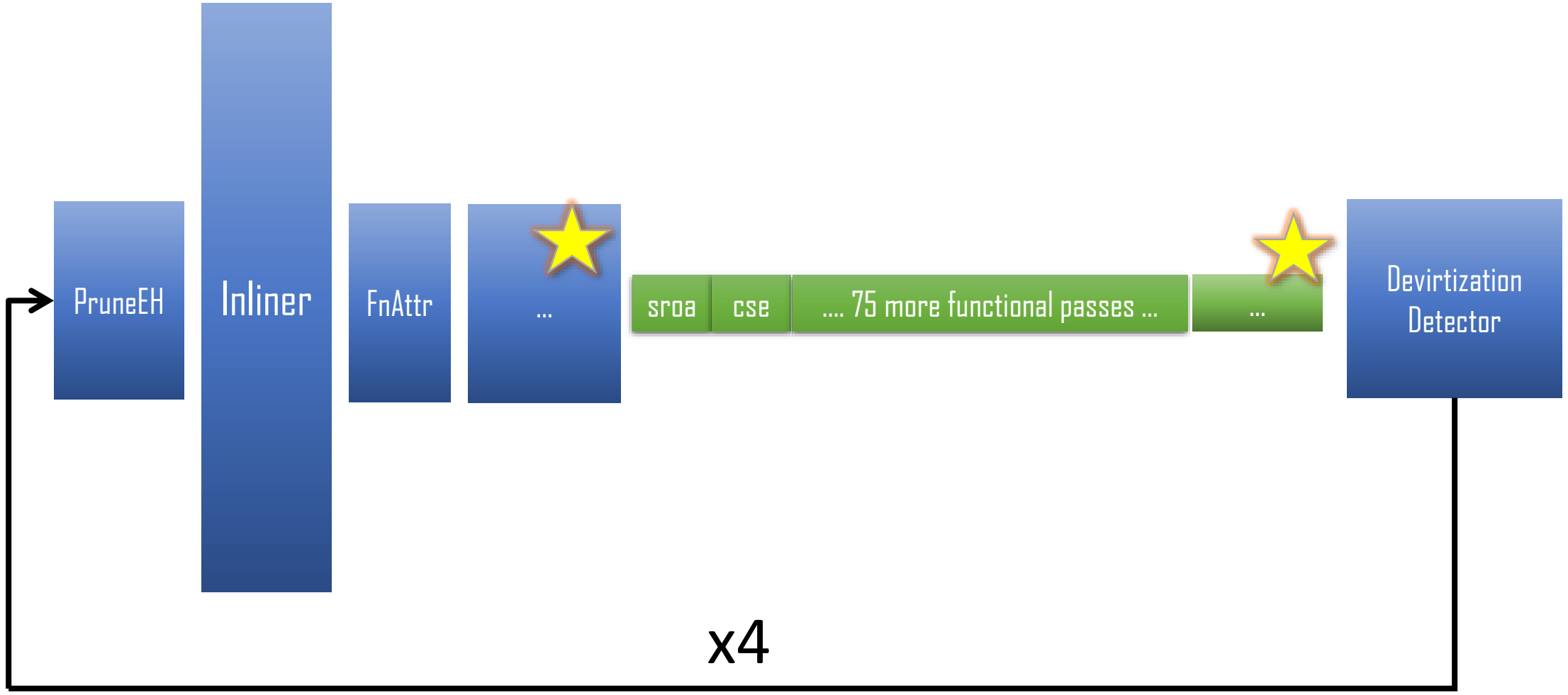
```
define void @seq(%struct.generator* noalias sret %agg.result) #0 {
entry:
  %coro.promise = alloca %"struct.generator<int>::promise_type", align 4
  %coro.gro = alloca %struct.generator, align 8
  %ref.tmp = alloca %"struct.std::suspend_always", align 1
  %undef.agg.tmp = alloca %"struct.std::suspend_always", align 1
  %agg.tmp = alloca %"struct.std::coroutine_handle.0", align 8
                    ...
```

```
seq:
        pushq   %rbx
        movq    %rdi, %rbx
        movl    $32, %edi
        callq   _Znwm@PLT
          ...
```

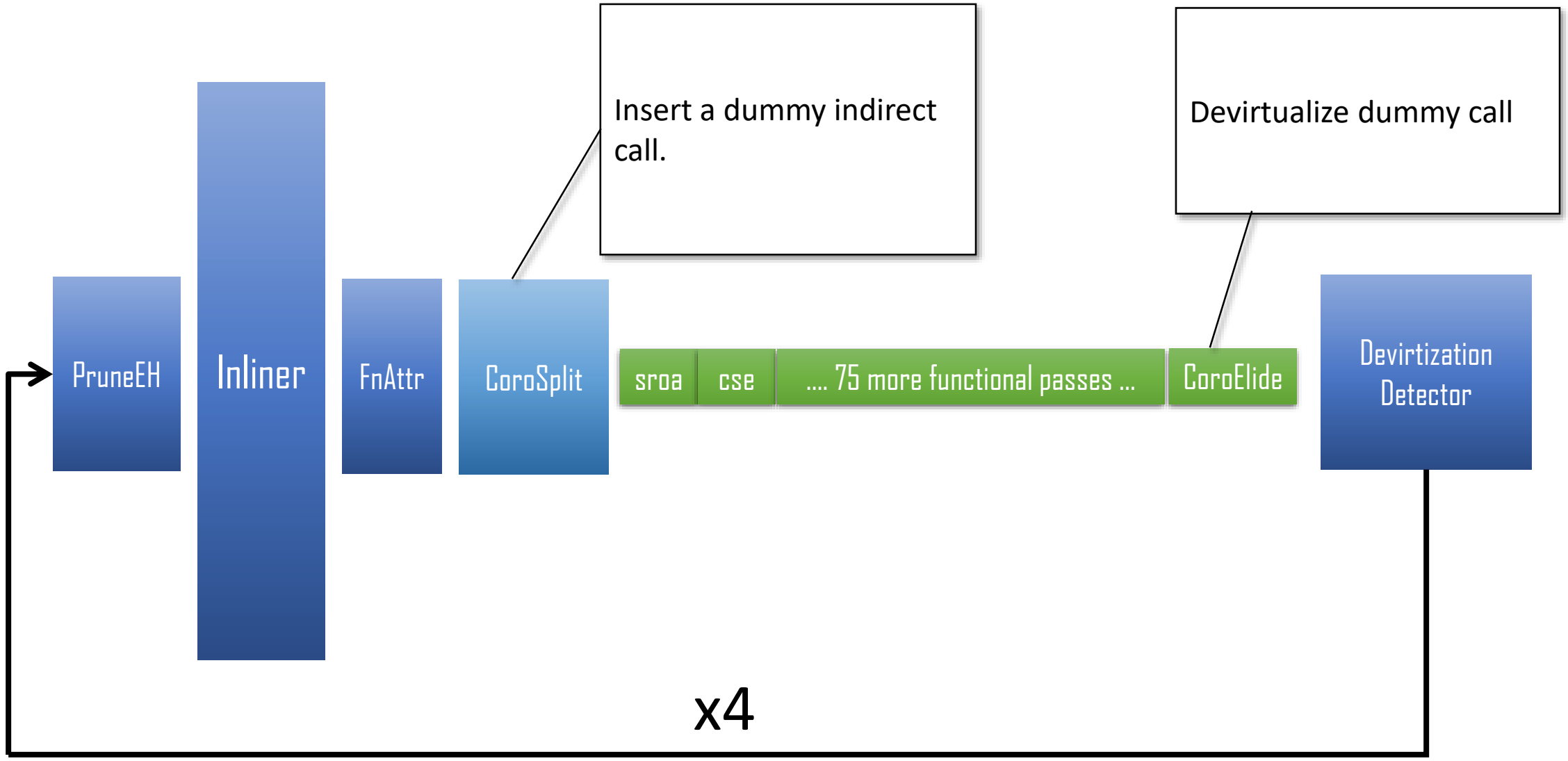
Where would you split a coroutine?



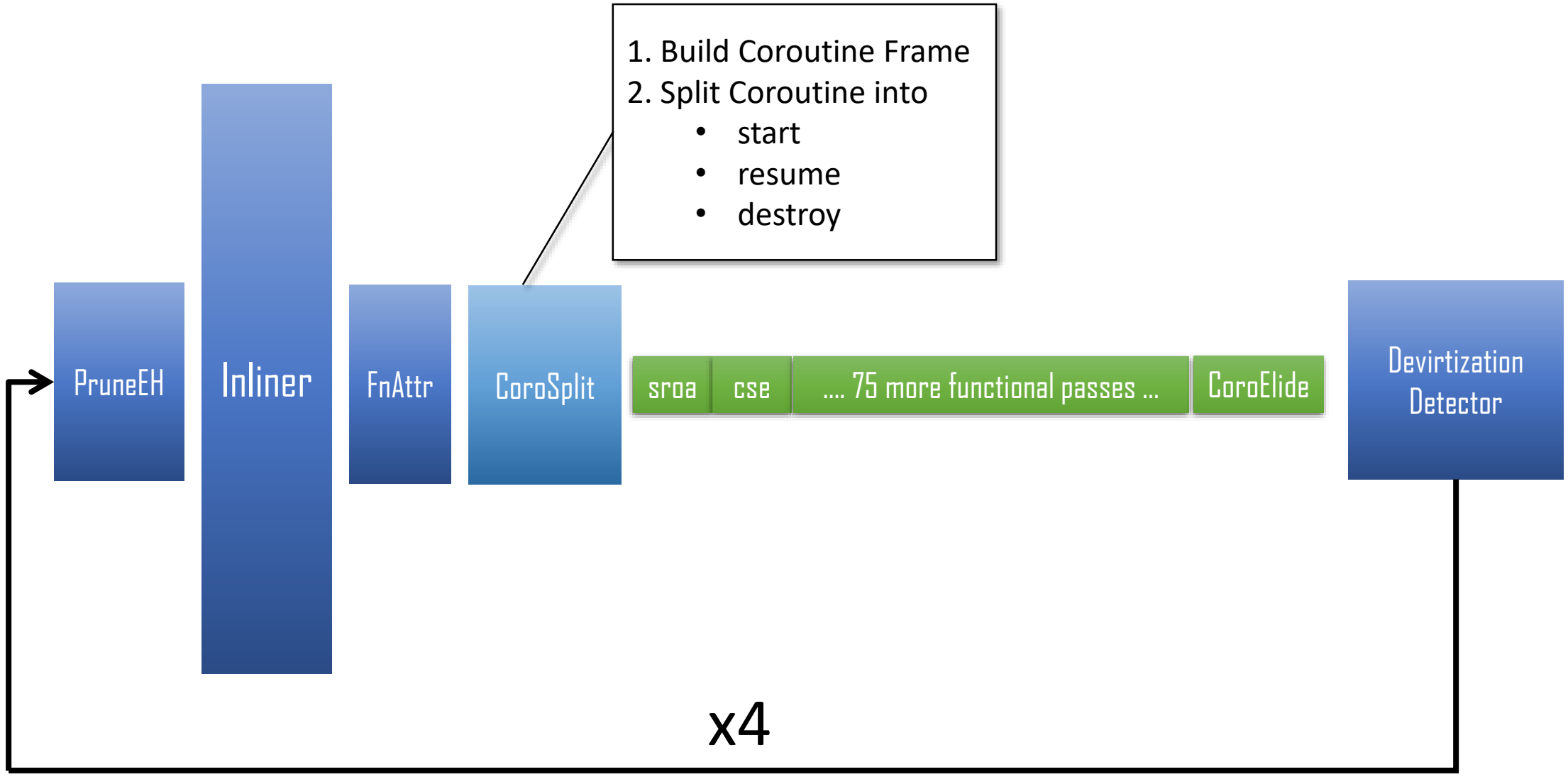
Where would you split a coroutine?



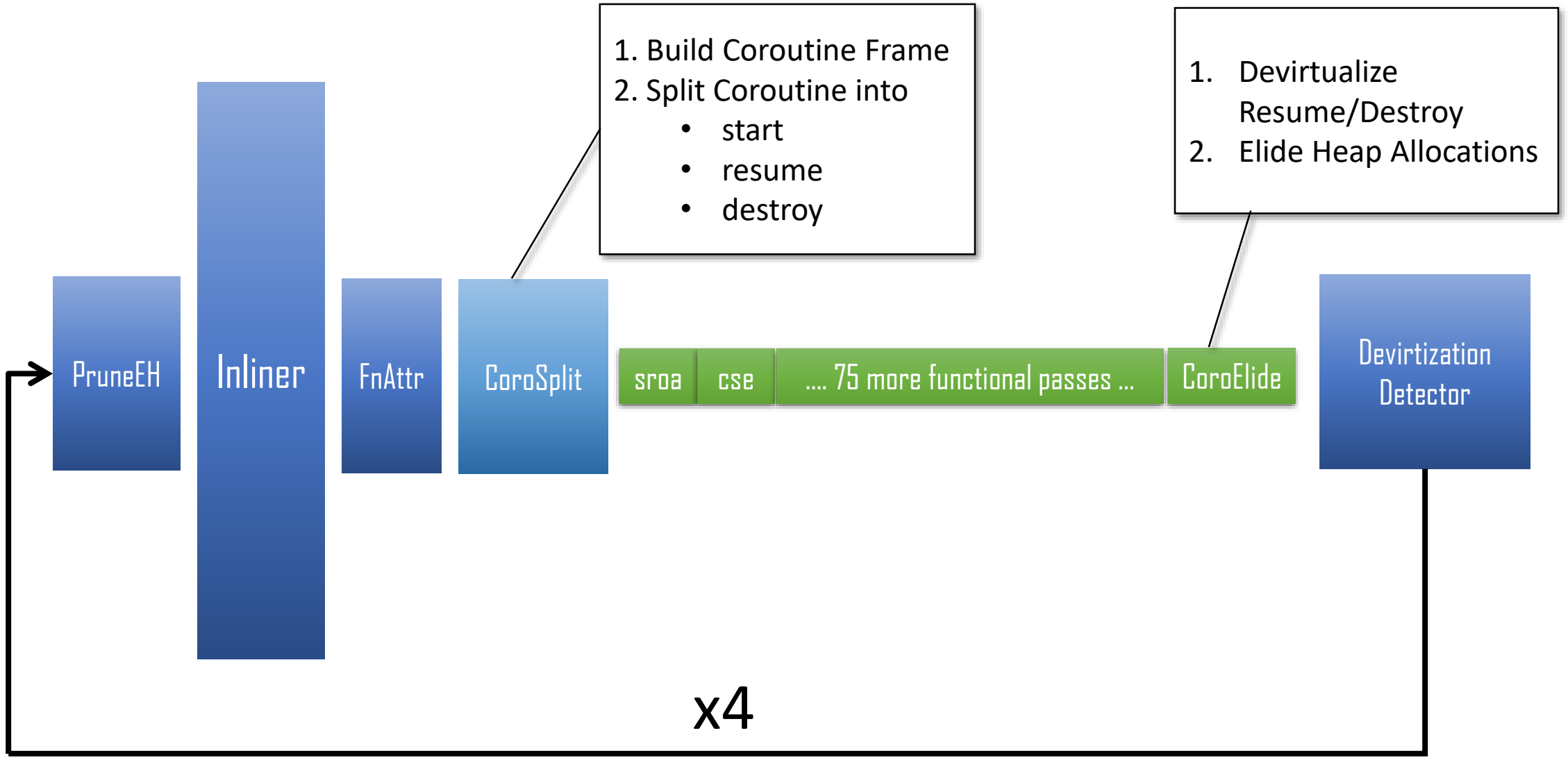
Where would you split a coroutine?



Where would you split a coroutine?



Where would you split a coroutine?



Coroutine intrinsics

```
define i32 @main() {
entry:
  %hdl = call i8* @gen(i32 9)
  call void @llvm.coro.resume(i8* %hdl)
  call void @llvm.coro.resume(i8* %hdl)
  call void @llvm.coro.destroy(i8* %hdl)
  ret i32 0
}
```

Let’s code up in LLVM IR this coroutine

```
void *gen(int n) {
    for(;;) {
      print(n++);
      <suspend> // returns a coroutine
                // handle on first suspend
    }
}
```

Same Coroutine in LLVM IR

```
define i8* @gen(i32 %n) {
entry:
  %id = call token @llvm.coro.id(i32 0, i8* null, i8* null, i8* null)
  %size = call i32 @llvm.coro.size.i32()
  %alloc = call i8* @malloc(i32 %size)
  %hdl = call noalias i8* @llvm.coro.begin(token %id, i8* %alloc)
  br label %loop
loop:
  %n.val = phi i32 [ %n, %entry ], [ %inc, %loop ]
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop
                                       i8 1, label %cleanup]
cleanup:
  %mem = call i8* @llvm.coro.free(token %id, i8* %hdl)
  call void @free(i8* %mem)
  br label %suspend_or_ret
suspend_or_ret:
  %unused = call i1 @llvm.coro.end(i8* %hdl, i1 false)
  ret i8* %hdl
}
```

Same Coroutine in LLVM IR

```
define i8* @gen(i32 %n) {
entry:
  %id = call token @llvm.coro.id(i32 0, i8* null, i8* null, i8* null)
  %size = call i32 @llvm.coro.size.i32()
  %alloc = call i8* @malloc(i32 %size)
  %hdl = call noalias i8* @llvm.coro.begin(token %id, i8* %alloc)
  br label %loop
loop:
  %n.val = phi i32 [ %n, %entry ], [ %inc, %loop ]
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop
                                       i8 1, label %cleanup]
cleanup:
  %mem = call i8* @llvm.coro.free(token %id, i8* %hdl)
  call void @free(i8* %mem)
  br label %suspend_or_ret
suspend_or_ret:
  call void @llvm.coro.end(i8* %hdl, i1 false)
  ret i8* %hdl
}
```

ALLOCATION PART

USER BODY

DEALLOCATION PART

SUSPEND/RETURN PART

Same Coroutine in LLVM IR

```
define i8* @gen(i32 %n) {
entry:
  %id = call token @llvm.coro.id(i32 0, i8* null, i8* null, i8* null)
  %size = call i32 @llvm.coro.size.i32()
  %alloc = call i8* @malloc(i32 %size)
  %hdl = call noalias i8* @llvm.coro.begin(token %id, i8* %alloc)
  br label %loop
loop:
  %n.val = phi i32 [ %n, %entry ], [ %inc, %loop ]
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop
                                       i8 1, label %cleanup]
cleanup:
  %mem = call i8* @llvm.coro.free(token %id, i8* %hdl)
  call void @free(i8* %mem)
  br label %suspend_or_ret
suspend_or_ret:
  call void @llvm.coro.end(i8* %hdl, i1 false)
  ret i8* %hdl
}
```

USER BODY

DEALLOCATION PART

SUSPEND/RETURN PART

Same Coroutine in LLVM IR

```
define i8* @gen(i32 %n) {
entry:
  %id = call token @llvm.coro.id(i32 0, i8* null, i8* null, i8* null)
  %size = call i32 @llvm.coro.size.i32()
  %alloc = call i8* @malloc(i32 %size)
  %hdl = call noalias i8* @llvm.coro.begin(token %id, i8* %alloc)
  br label %loop
loop:
  %n.val = phi i32 [ %n, %entry ], [ %inc, %loop ]
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop
                                       i8 1, label %cleanup]
cleanup:
  %mem = call i8* @llvm.coro.free(token %id, i8* %hdl)
  call void @free(i8* %mem)
  br label %suspend_or_ret
suspend_or_ret:
  call void @llvm.coro.end(i8* %hdl, i1 false)
  ret i8* %hdl
}
```

USER BODY

SUSPEND/RETURN PART

Same Coroutine in LLVM IR

```
define i8* @gen(i32 %n) {
entry:
  %id = call token @llvm.coro.id(i32 0, i8* null, i8* null, i8* null)
  %size = call i32 @llvm.coro.size.i32()
  %alloc = call i8* @malloc(i32 %size)
  %hdl = call noalias i8* @llvm.coro.begin(token %id, i8* %alloc)
  br label %loop
loop:
  %n.val = phi i32 [ %n, %entry ], [ %inc, %loop ]
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop
                                       i8 1, label %cleanup]
cleanup:
  %mem = call i8* @llvm.coro.free(token %id, i8* %hdl)
  call void @free(i8* %mem)
  br label %suspend_or_ret
suspend_or_ret:
  call void @llvm.coro.end(i8* %hdl, i1 false)
  ret i8* %hdl
}
```

ALLOCATION PART

USER BODY

DEALLOCATION PART

Same Coroutine in LLVM IR

```
define i8* @gen(i32 %n) {
entry:
  %id = call token @llvm.coro.id(i32 0, i8* null, i8* null, i8* null)
  %size = call i32 @llvm.coro.size.i32()
  %alloc = call i8* @malloc(i32 %size)
  %hdl = call noalias i8* @llvm.coro.begin(token %id, i8* %alloc)
  br label %loop
loop:
  %n.val = phi i32 [ %n, %entry ], [ %inc, %loop ]
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop
                                       i8 1, label %cleanup]
cleanup:
  %mem = call i8* @llvm.coro.free(token %id, i8* %hdl)
  call void @free(i8* %mem)
  br label %suspend_or_ret
suspend_or_ret:
  call void @llvm.coro.end(i8* %hdl, i1 false)
  ret i8* %hdl
}
```

ALLOCATION PART

suspend

DEALLOCATION PART

SUSPEND/RETURN PART

Build Coroutine Frame

```
define i8* @gen(i32 %n) {
entry:
  …
  br label %loop
loop:
  %n.val = phi i32 [ %n, %entry ], [ %inc, %loop ]
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop
                                       i8 1, label %cleanup]
cleanup:
  …
}
```

Build Coroutine Frame: Simplify PHI Nodes

```
define i8* @gen(i32 %n) {
 …
loop.from.entry:
  %n.val.from.entry = phi i32 [ %n, %entry ]
  br label %loop
loop:
  %n.val = phi i32 [%n.val.from.entry, %loop.from.entry ], [ %inc, %loop ]
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop
                                       i8 1, label %cleanup]
cleanup:
  …
}
```

Build Coroutine Frame: Simplify PHI Nodes

```
define i8* @gen(i32 %n) {
 …
loop.from.entry:
  %n.val.from.entry = phi i32 [ %n, %entry ]
  br label %loop
loop:
  %n.val = phi i32 [%n.val.from.entry, %loop.from.entry ], [ %inc.from.loop, %loop.from.loop
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop.from.loop
                                       i8 1, label %cleanup]
loop.from.loop:
  %inc.from.loop = phi i32 [ %inc, %loop ]
  br label %loop
  …
}
```

Build Coroutine Frame

```
define i8* @gen(i32 %n) {
 …
loop.from.entry:
  %n.val.from.entry = phi i32 [ %n, %entry ]
  br label %loop
loop:
  %n.val = phi i32 [%n.val.from.entry, %loop.from.entry ], [ %inc.from.loop, %loop.from.loop
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
-------------------------------------------------------------------------------
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop.from.loop
                                       i8 1, label %cleanup]
loop.from.loop:
  %inc.from.loop = phi i32 [ %inc, %loop ]
  br label %loop
  …
}
```

```
%f.frame = type { }
```

Build Coroutine Frame

```
define i8* @gen(i32 %n) {
 …
loop.from.entry:
  %n.val.from.entry = phi i32 [ %n, %entry ]
  br label %loop
loop:
  %n.val = phi i32 [%n.val.from.entry, %loop.from.entry ], [ %inc.from.loop, %loop.from.loop
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop.from.loop
                                       i8 1, label %cleanup]
loop.from.loop:
  %inc.from.loop = phi i32 [ %inc, %loop ]
  br label %loop
  …
}
```

```
%f.frame = type { }
```

Build Coroutine Frame

```
define i8* @gen(i32 %n) {
 …
loop.from.entry:
  %n.val.from.entry = phi i32 [ %n, %entry ]
  br label %loop
loop:
  %n.val = phi i32 [%n.val.from.entry, %loop.from.entry], [ %inc1, %loop.from.loop]
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop.from.loop
                                       i8 1, label %cleanup]
loop.from.loop:
  %inc1 = add nsw i32 %n.val, 1
  br label %loop
  …
}
```

```
%f.frame = type { }
```

Build Coroutine Frame

```
define i8* @gen(i32 %n) {
 …
loop.from.entry:
  %n.val.from.entry = phi i32 [ %n, %entry ]
  br label %loop
loop:
  %n.val = phi i32 [%n.val.from.entry, %loop.from.entry], [ %inc1, %loop.from.loop]
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop.from.loop
                                       i8 1, label %cleanup]
loop.from.loop:
  %inc1 = add nsw i32 %n.val, 1
  br label %loop
  …
}
```

```
%f.frame = type {}
```

Build Coroutine Frame

```
define i8* @gen(i32 %n) {
 …
loop.from.entry:
  %n.val.from.entry = phi i32 [ %n, %entry ]
  br label %loop
loop:
  %n.val = phi i32 [%n.val.from.entry, %loop.from.entry], [ %inc1, %loop.from.loop]
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop.from.loop
                                       i8 1, label %cleanup]
loop.from.loop:
  %inc1 = add nsw i32 %n.val, 1
  br label %loop
  …
}
```

```
%f.frame = type { i32 }
```

%n.val spill

Build Coroutine Frame

```
define i8* @gen(i32 %n) {
entry:
   …
  %hdl = call noalias i8* @llvm.coro.begin(token %id, i8* %alloc)
  %frame = bitcast i8* %hdl to %f.frame*
  br label %loop
loop:
  %n.val = phi i32 [%n, %entry ], [ %inc1, %loop.from.loop ]
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
   …
loop.from.loop:
  %inc1 = add nsw i32 %n.val, 1
  br label %loop
  …
}
```

```
%f.frame = type { i32 }
```

Build Coroutine Frame

```
define i8* @gen(i32 %n) {
entry:
   …
  %hdl = call noalias i8* @llvm.coro.begin(token %id, i8* %alloc)
  %frame = bitcast i8* %hdl to %f.frame*
  br label %loop
loop:
  %n.val = phi i32 [%n, %entry ], [ %inc.from.loop, %loop.from.loop ]
  %n.val.spill.addr = getelementpointer %f.frame, %frame, i32 0, i32 0
  store i32 %n.val, i32* %n.val.spill.addr
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
   …
loop.from.loop:
  %inc1 = add nsw i32 %n.val, 1
  br label %loop
  …
}
```

```
%f.frame = type { i32 }
```

Build Coroutine Frame

```
define i8* @gen(i32 %n) {
entry:
   …
  %hdl = call noalias i8* @llvm.coro.begin(token %id, i8* %alloc)
  %frame = bitcast i8* %hdl to %f.frame*
  br label %loop
loop:
  %n.val = phi i32 [%n, %entry ], [ %n.val.from.loop, %loop.from.loop ]
  %n.val.spill.addr = getelementpointer %f.frame, %frame, i32 0, i32 0
  store i32 %n.val, i32* %n.val.spill.addr
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
   …
loop.from.loop:
  %n.val.reload = load i32, i32* %n.val.spill.addr
  %inc1 = add nsw i32 %n.val.reload, 1
  br label %loop
  …
```

```
%f.frame = type { i32 }
```

Split the coroutine

Split Coroutine

```
define i8* @gen(i32 %n) {
entry:
  …
  %hdl = call noalias i8* @llvm.coro.begin(token %id, i8* %alloc)
  %frame = bitcast i8* %hdl to %f.frame*
  br label %loop
loop:
  %n.val = phi i32 [ %n, %entry ], [ %inc1, %loop.from.loop ]
  %n.val.spill.addr = getelementpointer %f.frame, %frame, i32 0, i32 0
  store i32 %n.val, i32* %n.val.spill.addr
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop.from.loop
                                       i8 1, label %cleanup]
  …
suspend_or_ret:
  call void @llvm.coro.end(i8* %hdl, i1 false)
  ret i8* %hdl
}
```

Split Coroutine

```
define fastcc void @gen.resume(%f.frame* %frame) {
entry:
  …
  %hdl = call noalias i8* @llvm.coro.begin(token %id, i8* %alloc)
  %frame = bitcast i8* %hdl to %f.frame*
  br label %loop
loop:
  %n.val = phi i32 [ %n, %entry ], [ %inc1, %loop.from.loop ]
  %n.val.spill.addr = getelementpointer %f.frame, %frame, i32 0, i32 0
  store i32 %n.val, i32* %n.val.spill.addr
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop.from.loop
                                       i8 1, label %cleanup]
  …
suspend_or_ret:
  call void @llvm.coro.end(i8* %hdl, i1 false)
  ret i8* %hdl
}
```

Split Coroutine

```
define fastcc void @gen.resume(%f.frame* %frame) {
entry:
  …
  %hdl = call noalias i8* @llvm.coro.begin(token %id, i8* %alloc)
  %frame = bitcast i8* %hdl to %f.frame*
  br label %loop
loop:
  %n.val = phi i32 [ %n, %entry ], [ %inc1, %loop.from.loop ]
  %n.val.spill.addr = getelementpointer %f.frame, %frame, i32 0, i32 0
  store i32 %n.val, i32* %n.val.spill.addr
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
  br label %resume1
resume1:
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop.from.loop
                                       i8 1, label %cleanup]
  …
suspend_or_ret:
  call void @llvm.coro.end(i8* %hdl, i1 false)
  ret i8* %hdl
}
```

Split Coroutine

```
define fastcc void @gen.resume(%f.frame* %frame) {
entry:



  br label %resume1 ; or a switch based on an index stored in the frame
loop:
  %n.val = phi i32 [ %n, %entry ], [ %inc1, %loop.from.loop ]
  %n.val.spill.addr = getelementpointer %f.frame, %frame, i32 0, i32 0
  store i32 %n.val, i32* %n.val.spill.addr
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
  br label %resume1
resume1:
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop.from.loop
                                       i8 1, label %cleanup]

  …
suspend_or_ret:
  call void @llvm.coro.end(i8* %hdl, i1 false)
  ret i8* %hdl
}
```

Split Coroutine

```
define fastcc void @gen.resume(%f.frame* %frame) {
entry:


  br label %resume1 ; or a switch based on an index stored in the frame
loop:
  %n.val = phi i32 [ %n, %entry ], [ %inc1, %loop.from.loop ]
  %n.val.spill.addr = getelementpointer %f.frame, %frame, i32 0, i32 0
  store i32 %n.val, i32* %n.val.spill.addr
  %inc = add nsw i32 %n.val, 1
  call void @print(i32 %n.val)
  br label %resume1
resume1:
  %0 = call i8 @llvm.coro.suspend(token none, i1 false)
  switch i8 %0, label %suspend_or_ret [i8 0, label %loop.from.loop
                                       i8 1, label %cleanup]
  …
suspend_or_ret:
  ret void
}
```

Finishing Touches

- Clone gen.resume twice and name the clones:
  gen.destroy and gen.cleanup

`llvm.coro.suspend`	
-1	In start function
0	In resume function
1	In destroy and cleanup functions

`llvm.coro.free(hdl)`	
0	In cleanup function
hdl	elsewhere

Split Coroutine

```
define fastcc void @gen.resume (%f.frame* %frame) {
  %n.val.spill.addr = getelementpointer %f.frame, %frame, i32 0, i32 0
  %n.val = load i32, i32* %n.val.spill.addr
  %inc1 = add nsw i32 %n.val, 1
  store i32 %inc1, i32* %n.val.spill.addr
  call void @print(i32 %n.val)
  ret void
}

define fastcc void @gen.destroy(%f.frame* %frame) {
  %mem = bitcast %f.frame* %frame to i8*
  call void @free(i8* %mem)
  ret void
}

define fastcc void @gen.cleanup(%f.frame* %frame) {
  ret void
}
```

Split Coroutine

```
define i8* @gen(i32 %n) {
entry:
  %id = call token @llvm.coro.id(i32 0, i8* null, i8* null, i8* null)
  %alloc = call i8* @malloc(i32 4)
  %hdl = call noalias i8* @llvm.coro.begin(token %id, i8* %alloc)
  %frame = bitcast i8* %hdl to %f.frame*

  %n.val.spill.addr = getelementpointer %f.frame, %frame, i32 0, i32 0
  store i32 %n, i32* %n.val.spill.addr
  call void @print(i32 %n.val)
  ret i8* %hdl
}
```

Devirtualization and Allocation Elision

Before Inlining

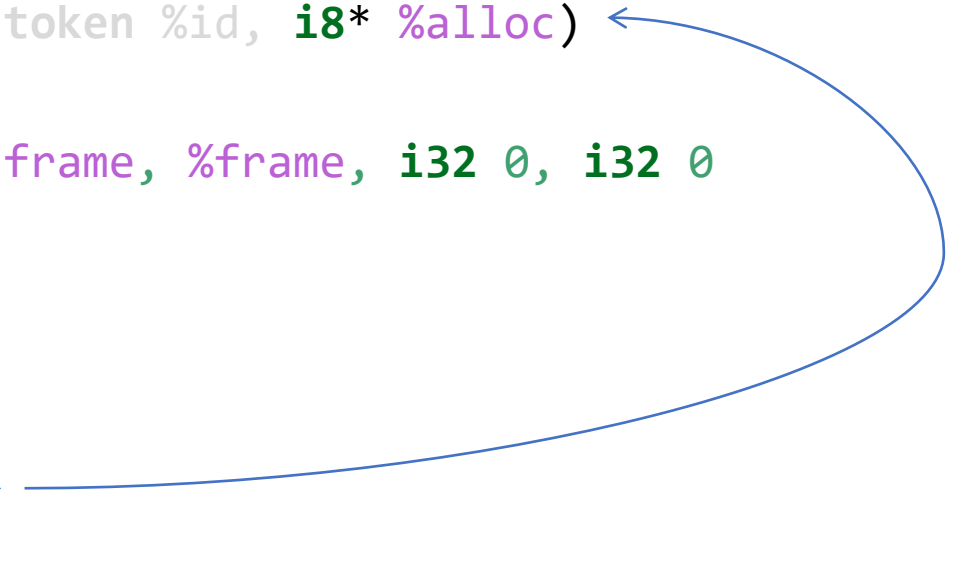
```
define i32 @main() {
entry:
  %hdl = call i8* @gen(i32 9)
  call void @llvm.coro.resume(i8* %hdl)
  call void @llvm.coro.resume(i8* %hdl)
  call void @llvm.coro.destroy(i8* %hdl)
  ret i32 0
}
```

After Inlining

```
define i32 @main() {
entry:
  %id = call token @llvm.coro.id(i32 0, i8* null, i8* null, @f.resumers)
  %alloc = call i8* @malloc(i32 4)
  %hdl = call noalias i8* @llvm.coro.begin(token %id, i8* %alloc)
  %frame = bitcast i8* %hdl to %f.frame*
  %n.val.spill.addr = getelementpointer %f.frame, %frame, i32 0, i32 0
  store i32 9, i32* %n.val.spill.addr
  call void @print(i32 9)

  call void @llvm.coro.resume(i8* %hdl)
  call void @llvm.coro.resume(i8* %hdl)
  call void @llvm.coro.destroy(i8* %hdl)
  ret i32 0
}
```

Devirtualization

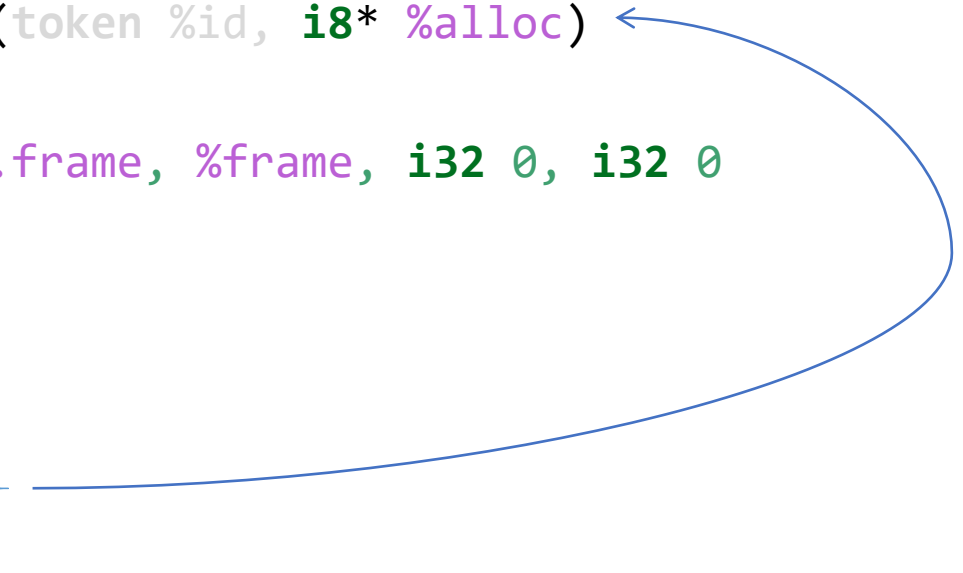
```
define i32 @main() {
entry:
  %id = call token @llvm.coro.id(i32 0, i8* null, i8* null, @gen.resumers)
  %alloc = call i8* @malloc(i32 4)
  %hdl = call noalias i8* @llvm.coro.begin(token %id, i8* %alloc)
  %frame = bitcast i8* %hdl to %f.frame*
  %n.val.spill.addr = getelementpointer %f.frame, %frame, i32 0, i32 0
  store i32 9, i32* %n.val.spill.addr
  call void @print(i32 9)

  call void @llvm.coro.resume(i8* %hdl)
  call void @llvm.coro.resume(i8* %hdl)
  call void @llvm.coro.destroy(i8* %hdl)
  ret i32 0
}
```



```
@gen.resumers = private constant [3 x void (%gen.frame*)*]
                                 [@gen.resume, @gen.destroy, @f.cleanup]
```

Devirtualization

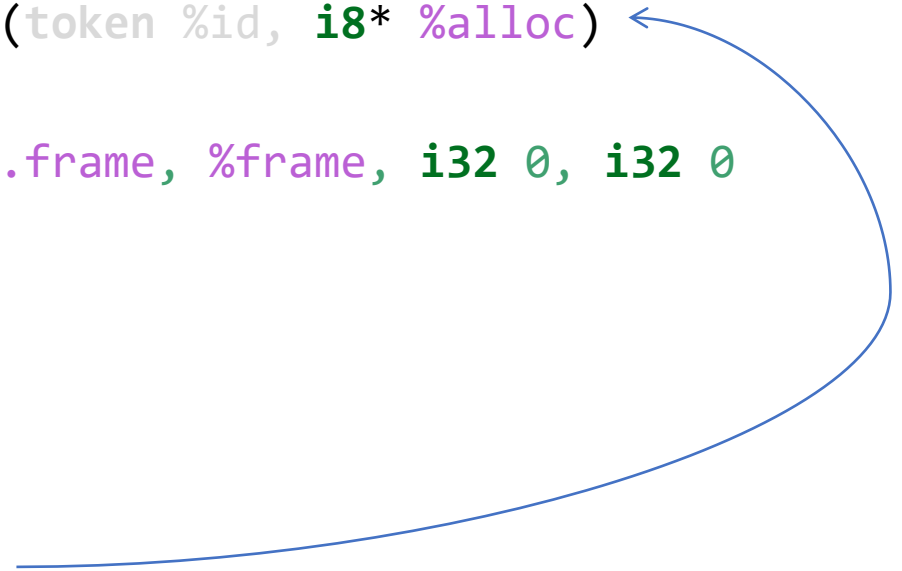
```
define i32 @main() {
entry:
  %id = call token @llvm.coro.id(i32 0, i8* null, i8* null, @gen.resumers)
  %alloc = call i8* @malloc(i32 4)
  %hdl = call noalias i8* @llvm.coro.begin(token %id, i8* %alloc)
  %frame = bitcast i8* %hdl to %f.frame*
  %n.val.spill.addr = getelementpointer %f.frame, %frame, i32 0, i32 0
  store i32 9, i32* %n.val.spill.addr
  call void @print(i32 9)

  call void @gen.resume(%f.frame* %frame)
  call void @gen.resume(%f.frame* %frame)
  call void @gen.destroy(%f.frame* %frame)
  ret i32 0
}
```



```
@gen.resumers = private constant [3 x void (%gen.frame*)*]
                                 [@gen.resume, @gen.destroy, @f.cleanup]
```

Heap Elision

```
define i32 @main() {
entry:
  %id = call token @llvm.coro.id(i32 0, i8* null, i8* null, @gen.resumers)
  %alloc = call i8* @malloc(i32 4)
  %hdl = call noalias i8* @llvm.coro.begin(token %id, i8* %alloc)
  %frame = bitcast i8* %hdl to %f.frame*
  %n.val.spill.addr = getelementpointer %f.frame, %frame, i32 0, i32 0
  store i32 9, i32* %n.val.spill.addr
  call void @print(i32 9)

  call void @gen.resume(%f.frame* %frame)
  call void @gen.resume(%f.frame* %frame)
  call void @gen.destroy(%f.frame* %frame)
  ret i32 0
}
```



Heap Elision

```
define i32 @main() {
entry:
  %id = call token @llvm.coro.id(i32 0, i8* null, i8* null, @gen.resumers)


  %frame = alloca %f.frame
  %n.val.spill.addr = getelementpointer %f.frame, %frame, i32 0, i32 0
  store i32 9, i32* %n.val.spill.addr
  call void @print(i32 9)

  call void @gen.resume(%f.frame* %frame)
  call void @gen.resume(%f.frame* %frame)
  call void @gen.cleanup(%f.frame* %frame)
  ret i32 0
}
```

At the end of –O2

```
define i32 @main() {
entry:
  call void @print(i32 9)
  call void @print(i32 10)
  call void @print(i32 11)
  ret i32 0
}
```

C++ Coroutine Design Goals

- **Scalable** (to **b**illions of concurrent coroutines)
- **Efficient** (resume and suspend operations comparable in cost to a function call overhead)
- Seamless interaction with existing facilities **with no overhead**
- **Open ended** coroutine machinery allowing library designers to develop coroutine libraries exposing various high-level semantics, such as generators, goroutines, tasks and more.
- **Usable** in environments where **exceptions** are forbidden or **not available**

LLVM/Clang Coroutines
Great thanks to:

Alexey Bataev

Chandler Carruth

David Majnemer

Eli Friedman

Eric Fiselier

Hal Finkel

Jim Radigan

Lewis Baker

Mehdi Amini

Richard Smith

Sanjoy Das

Victor Tong

More Info & Status

- LLVM Coroutines:
  http://llvm.org/docs/Coroutines.html
  experimental implementation is in the trunk of LLVM

  opt flag **–enable-coroutines** to try them out

  Examples: https://github.com/llvm-mirror/llvm/tree/master/test/Transforms/Coroutines

- C++ Coroutines:
  - http://wg21.link/P0057
  - MSVC – now
  - Clang Coroutines, soon, Clang 4.0 - possible

Questions?

More Work in LLVM

- A coroutine frame is bigger than it could be. Adding stack packing and stack coloring like optimization on the coroutine frame will result in tighter coroutine frames.
- Take advantage of the lifetime intrinsics for the data that goes into the coroutine frame. Leave lifetime intrinsics as is for the data that stays in allocas.
- The CoroElide optimization pass relies on coroutine ramp function to be inlined. It would be beneficial to split the ramp function further to increase the chance that it will get inlined into its caller.
- Design a convention that would make it possible to apply coroutine heap elision optimization across ABI boundaries.
- Cannot handle coroutines with *inalloca* parameters (used in x86 on Windows).
- Alignment is ignored by coro.begin and coro.free intrinsics.
- Make required changes to make sure that coroutine optimizations work with LTO.

Backup

Why coroutines?

```
int copy(Stream streamR, Stream streamW)
{
    char buf[512];
    int cnt = 0;
    int total = 0;
    do
    {
        cnt = streamR.read(sizeof(buf), buf);
        if (cnt == 0) break;
        cnt = streamW.write(cnt, buf);
        total += count;
    }
    while (cnt > 0);
    return total;
}
```

Why coroutines?

```
future<int> copy(Stream streamR, Stream streamW)
{
    char buf[512];
    int cnt = 0;
    int total = 0;
    do
    {
        cnt = co_await streamR.read(sizeof(buf), buf);
        if (cnt == 0) break;
        cnt = co_await streamW.write(cnt, buf);
        total += count;
    }
    while (cnt > 0);
    co_return total;
}
```

Why coroutines?

```
future<void> copy(Stream r, Stream w) {
    struct State {
        Stream streamR, streamW;
        char buf[512];
        char total = 0;
        State(Stream& r, Stream& w)
           : streamR(move(r)), streamW(move(streamW)) {}
    };
    auto state = make_shared<State>(streamR, streamW);
    return do_while([state]() -> future<bool> {
       return state->streamR.read(512, state->buf)
          .then([state](int count)) {
              return (count == 0) ? make_ready_future(false)
                 : [state, count] {
                     return state->streamR.write(count, state->buf)
                        .then([state](int count) {
                            state->total += count;
                           return make_ready_future(count > 0);
                        })();
              })
       ;
       }).then([state](auto){ return make_ready_future(state->total)});
    ;
}
```

Coroutines in C++

8.4.4 Coroutines **[dcl.fct.def.coroutine]**

Add this section to 8.4.

A function is a *coroutine* if it contains a *coroutine-return-statement* (6.6.3.1), an *await-expression* (5.3.8), a *yield-expression* (5.21), or a range-based `for` (6.5.4) with `co_await`. The *parameter-declaration-clause* of the coroutine shall not terminate with an ellipsis that is not part of a *parameter-declaration*.

```
generator<char> hello() {
   for (auto ch: "Hello, world\n")
      co_yield ch;
}

int main() {
   for (auto ch : hello()) cout << ch;
}
```

```
future<void> sleepy() {
   cout << “Going to sleep…\n";
   co_await sleep_for(1ms);
   cout << “Woke up\n";
   co_return 42;
}

int main() {
   cout << sleepy.get();
}
```

Coroutines are popular!

DART 1.9

```
Future<int> getPage(t) async {
 var c = new http.Client();
 try {
  var r = await c.get('http://url/search?q=$t');
  print(r);
  return r.length();
 } finally {
  await c.close();
 }
}
```

Python: PEP 0492

```
async def abinary(n):
    if n <= 0:
        return 1
    l = await abinary(n - 1)
    r = await abinary(n - 1)
    return l + 1 + r
```

C#

```
async Task<string> WaitAsynchronouslyAsync()
{
   await   Task.Delay(10000);
   return "Finished";
}
```

HACK

```
async function gen1(): Awaitable<int> {
 $x = await Batcher::fetch(1);
 $y = await Batcher::fetch(2);
 return $x + $y;
}
```

C++20?

```
future<string> WaitAsynchronouslyAsync()
{
   co_await   sleep_for(10ms);
   co_return "Finished“s;
}
```