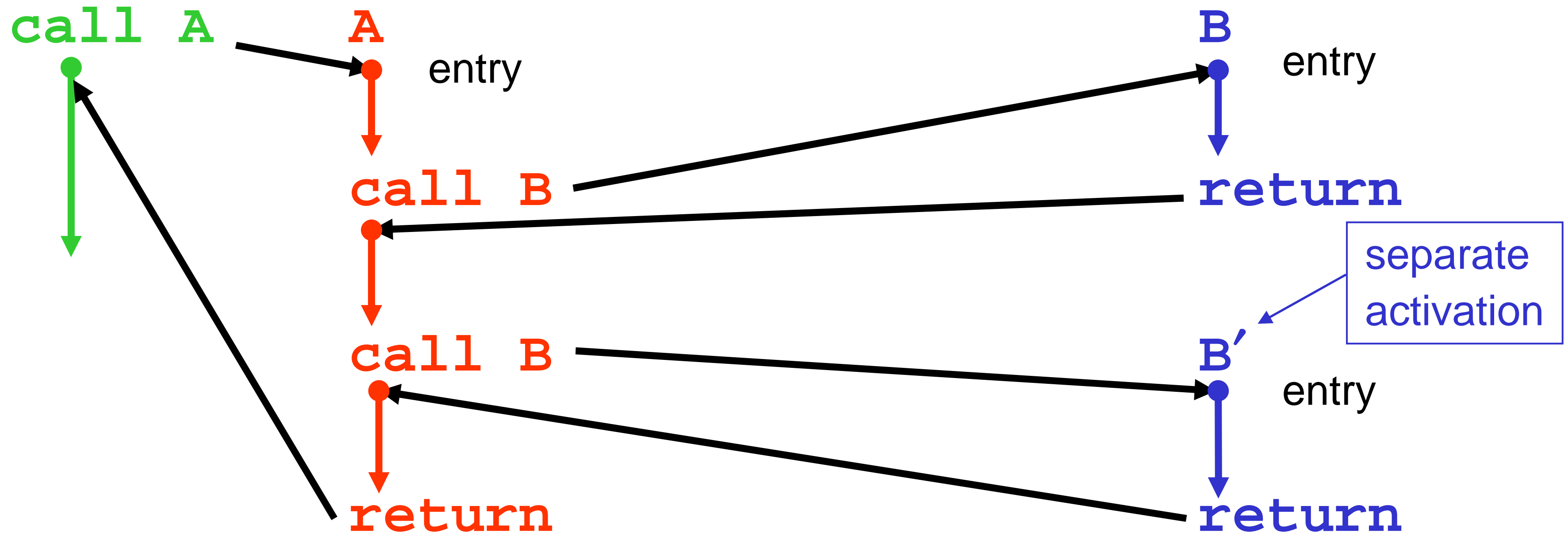
Principles of Programming Languages

Lecture 09

Coroutines

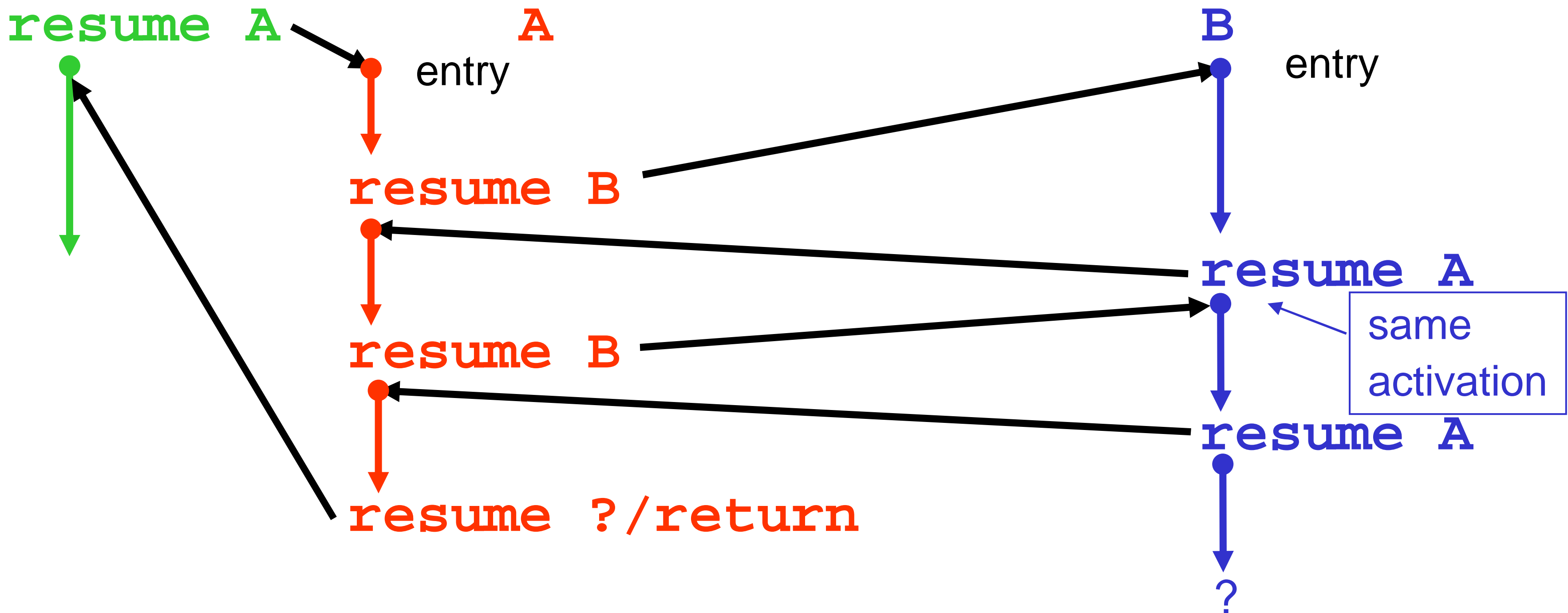
Subroutines vs Coroutines

- Subroutine call/return



Subroutines vs Coroutines (cont.)

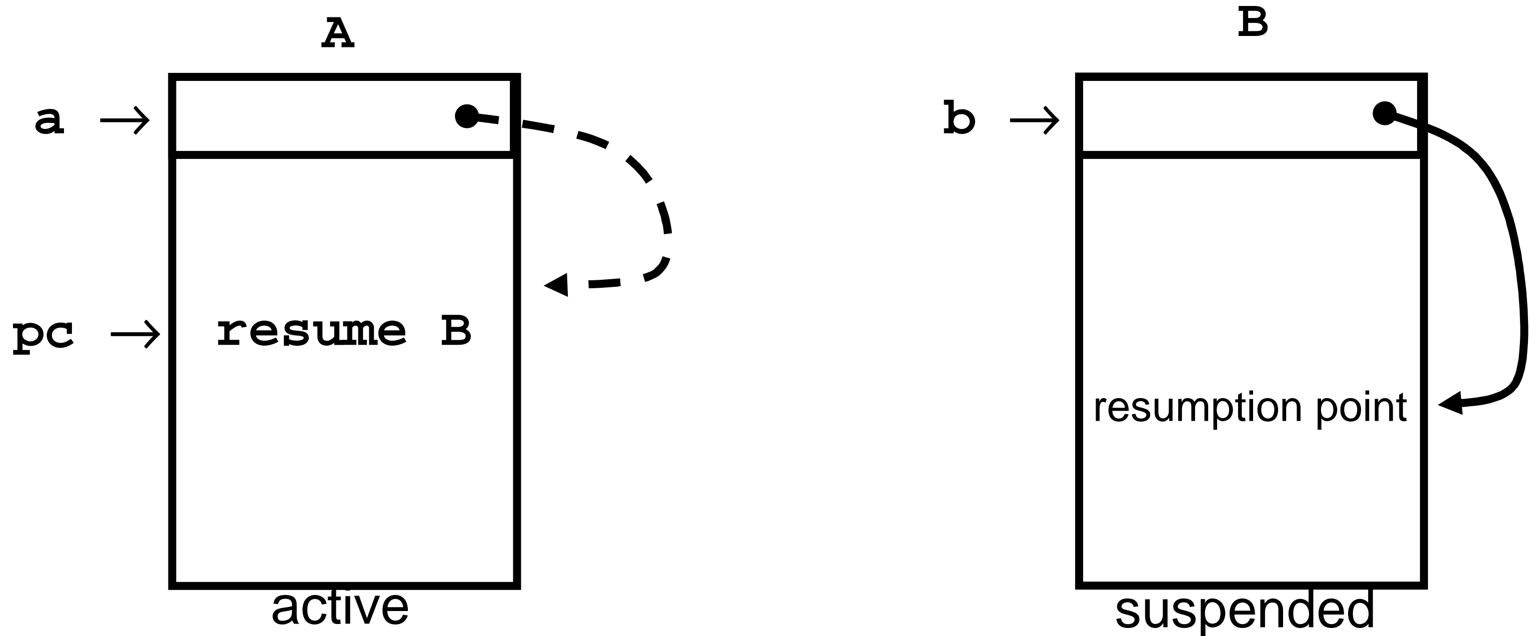
- Coroutine resume/resume



- Non-nested lifetimes $\Rightarrow$ abandon stack
- Activation lifetimes potentially unlimited if no “return”

Simple Coroutines

- No recursion
- Only one activation of each coroutine at any time

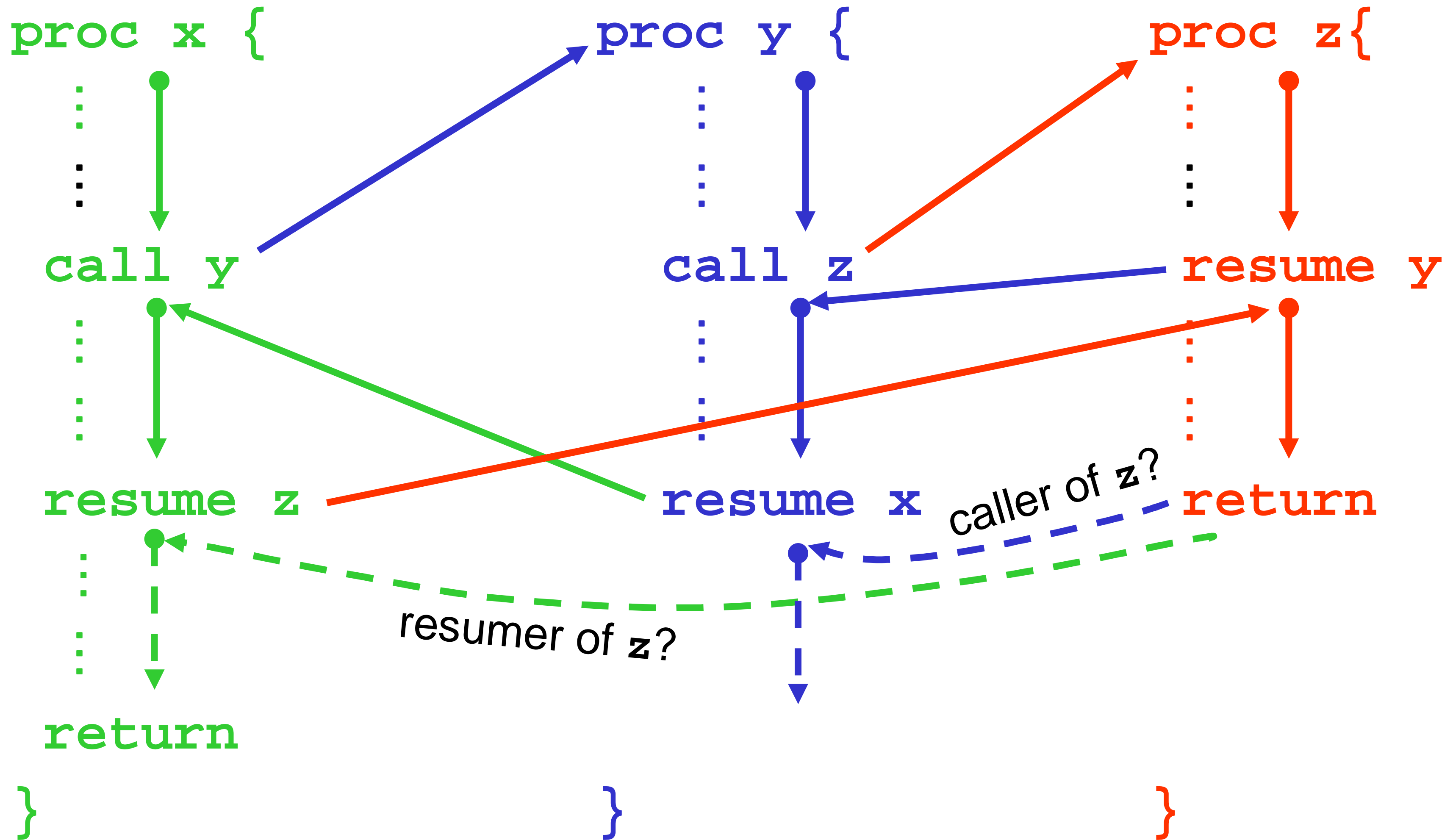



```
resume B:     *a = pc + 2
              jrst @b
              (control returns here)
```

Recursive Coroutines

- Initial resume (call) of `X:`
  - create activation for `X`
  - `resume execution at entry point`
- `resume Y :` suspend current activation
  - Resume which activation of `Y`?
- `resume ? ≡ return`
  - anonymous resume
  - "terminated" activation
- Call ≡ create & resume

Recursive Coroutines—the problem



Recursive Coroutines—solutions

- SIMULA 67
  - **`return (“detach”`** in Simula 67) in **`z`** resumes “caller of **`z`**”
- SL5
  - **`return`** resumes in latest activation that resumed **`z`**
  - With bindings at activation creation

SL5

- Activations (called "environments") are first class objects
- `p := procedure ( . . . ) . . . end`
  - Creates a procedure ("template") and assigns it to `p`
- `e := create p`
  - Uses template to create activation (including variable storage, continuation point &c.)
- `e := e with ( . . . )`
  - Transmits arguments (through "transmitters" for each argument)
- `resume e`
  - Suspends current activation and resume in `e`
  - Suspender becomes latest resumer of `e`
- `return [to e]`
  - Suspends current activation
  - Returns control to most recent resumer [to `e`]
- No deallocation of ARs—they are garbage collected

SL5 Primitives

- Let **c** = currently executing AR
- e **:= create** p

```
e = allocate(p)
e.cont = entrypoint(p)
e.creator = c
e.resumer = c
for each X nonlocal in p do {
    t = c
    while t != nil do
        if X public in t then
            e.X.lval = t.X.lval
        else t = t.creator
    if t == nil then error(X)
}
```

SL5 Primitives (cont.)

- e **:=** e **with** (a_1, a_2, …, a_n)

```
    e.par[1]= transmitter1(a1)
    . . .
```

- **resume** e

```
    c.cont = resumepoint
    // e.creator untouched
    e.resumer = c
    c = e
    goto c.cont
resumepoint:
```

SL5 Primitives (cont.)

- **return**

```
    c.cont = resumepoint
    c = c.resumer
    goto c.cont
resumepoint:
```

- **return to** e

```
    c.cont = resumepoint
    // no alteration of e.resumer
    c = e
    goto c.cont
resumepoint:
```

Procedure Call/Return—special case

```
f(a₁, a₂, …, aₙ) ⇒   resume  (create f
                          with (a₁, a₂, …, aₙ))
return           ⇒   return
```

- Binding is dynamic

e := create f

```
e= allocate(f)
e.cont = entrypoint(f)
e.creator = c  // "access link"
e.resumer = c  // dynamic link
// bind nonlocals using creator chain
```

with(a₁,…,an)

```
e.par[1]= transmitter₁(a₁)
. . .
```

Procedure Call/Return (cont.)

resume e

```
        e.cont = resumepoint
        e.resumer = c   // redundant
        c = e
        goto c.cont     // entrypoint(f)
resumepoint:
        . . .
```

return

```
        e.cont = resumepoint //never used
        c = c.resumer   // follow dl
        goto c.cont     // entrypoint(f)
resumepoint:
        . . .
```

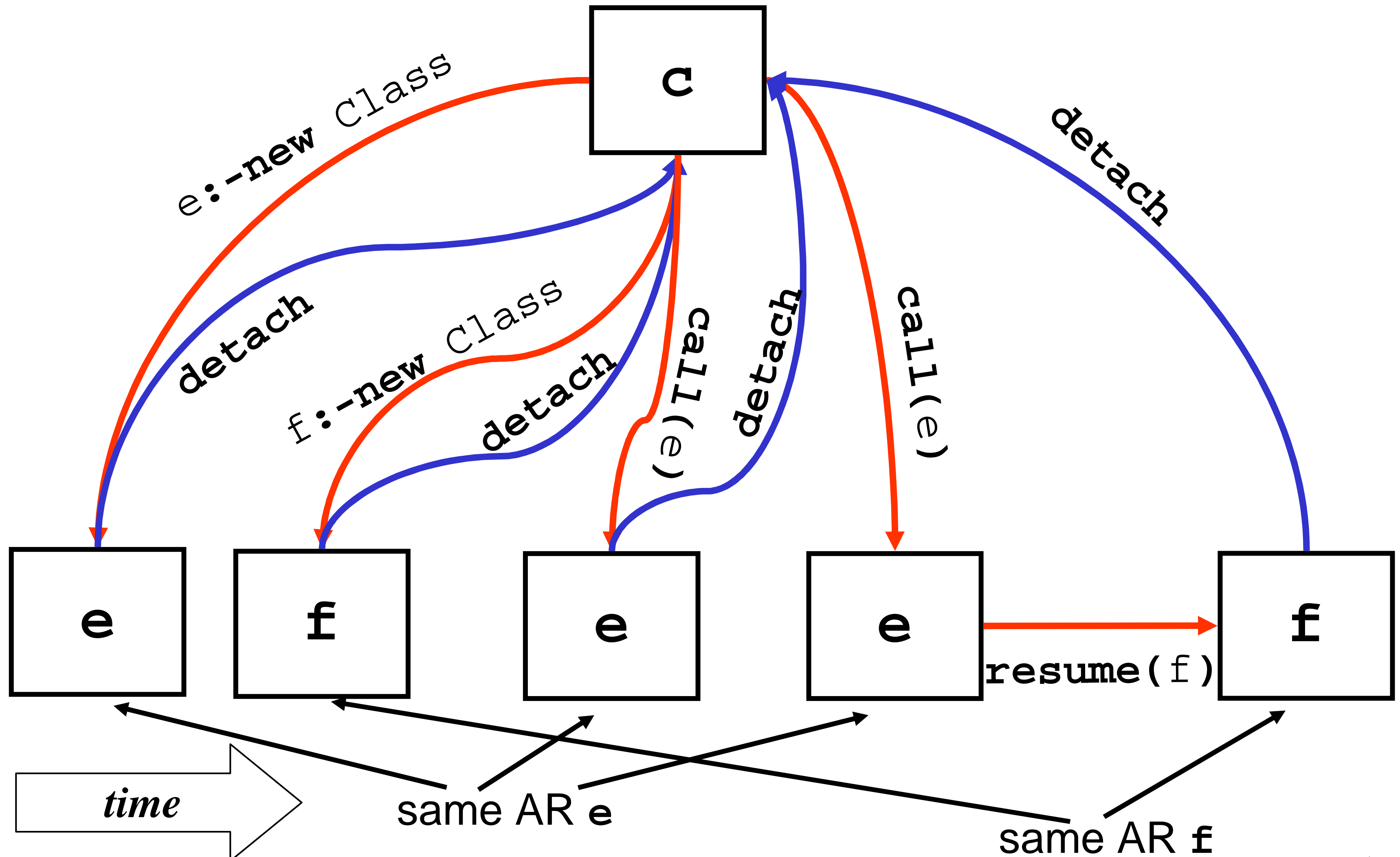
SIMULA 67

- Can create class instances (= objects) subordinate to block (AR) in which created
- All objects are "attached" to some AR during execution
- When suspended, AR is "detached"
- `class p(...);declarations;begin ... end p;`
  - Defines class template with formal parameters
- `e :- new p(...);`
  - Creates an object (AR) of class `p`: [`ref(p) e;`]
  - Transmits arguments
  - *Commences execution* in AR of `e`
  - AR `e` is "attached" to the suspended (creating) AR

SIMULA 67 (cont.)

- **`detach;`**
  - Suspend current activation
  - Resume in AR to which current is "attached"
  - Current AR marked "detached"
  - Approximately a "return"
  - **`end`** ⇒ **`detach`** (blocks detach when exited)
- **`call(`**`e`**`)`**
  - If `e` is detached, mark AR `e` as "attached" to caller (current AR)
  - Suspend caller (current AR)
  - Resume in AR `e`
- **`resume(`**`e`**`)`**
  - If `e` is detached, suspend current AR and resume execution in AR `e`
  - `e` is "attached" to AR to which current AR is "attached"—**`resume`** passes its attachment to `e`

SIMULA 67 (cont.)



SIMULA 67 Primitives

- Let **c** = currently executing AR
- `e :- new p( . . . );`

```
    e = allocate(p)
    { transmit parameters (CBN, CBV in Simula67)}
    e.cont = entrypoint(p)
    e.attached = c   // attacher of e is c
    { using c.sl and snl(p) and c's snl, calculate
     AR in which p was defined (created)
     & put ptr into t}
    c.sl = t
    c.cont = resumepoint
    c.attached = nil
    c = e
    goto c.cont
resumepoint:
```

SIMULA67 Primitives (cont.)

- **detach;**

```
    c.cont = resumepoint
    if c.attached == nil then error()
    else {
         t = c.attached
         c.attached = nil
         c = t      // back to attacher
         goto c.cont
         }
resumepoint:
```

SIMULA67 Primitives (cont.)

- **call**(e) —no parameters

```
      if e.attached != nil then error()
      e.attached = c              // e attached to caller
      c.cont = resumepoint
      c.attached = nil
      c = e
      goto c.cont
resumepoint:
```

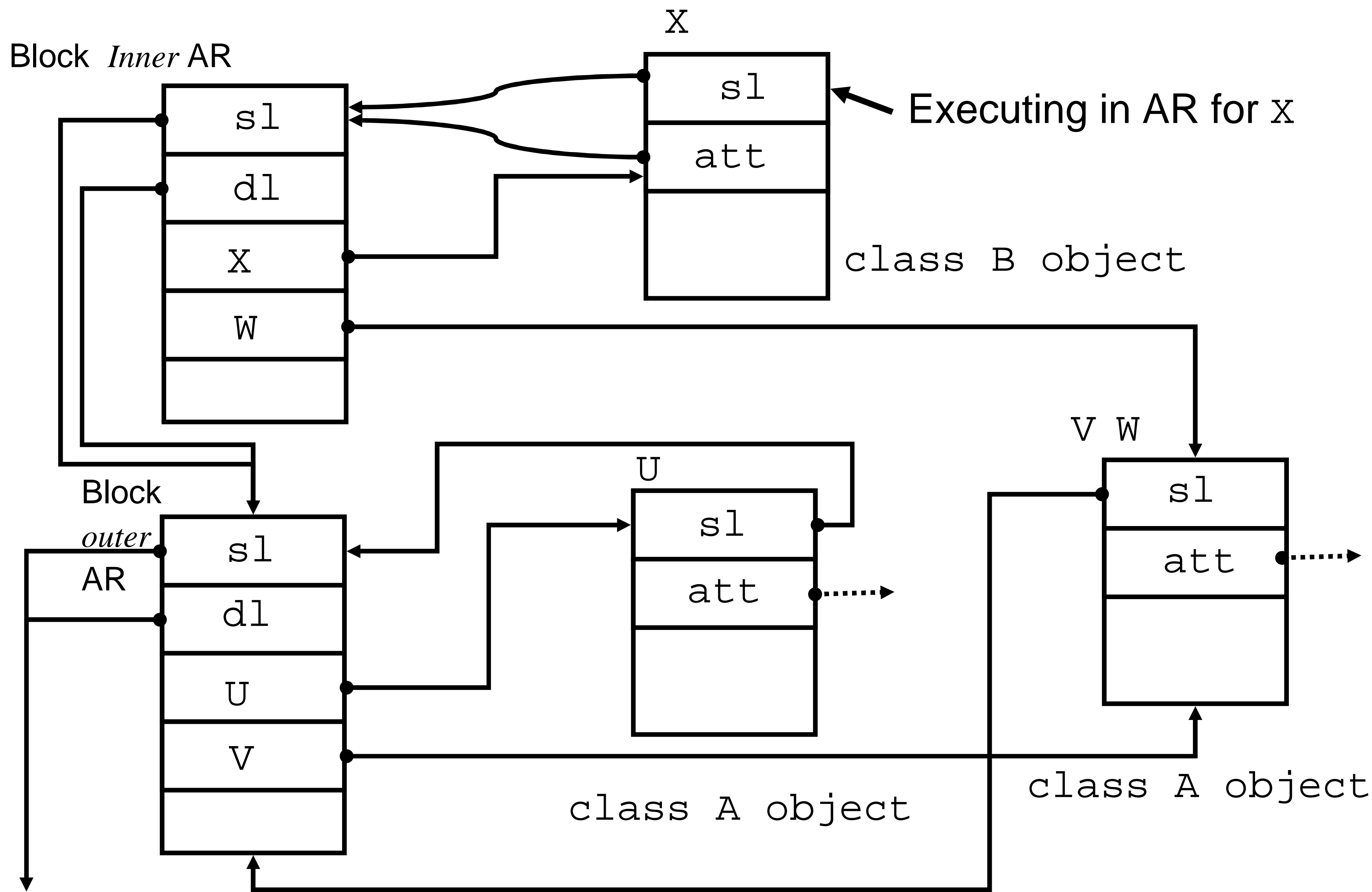
- **resume**(e)

```
      if e.attached != nil then error()
      e.attached = c.attached // e inherits attacher
      c.cont = resumepoint
      c.attached = nil
      c = e
      goto c.cont
resumepoint:
```

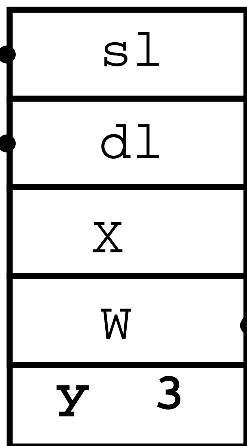
SIMULA67 Example

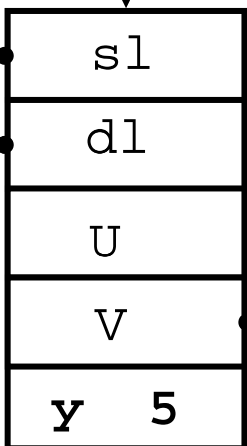
```
outer block: begin      class A; … detach; … ;
        ref(A) U,V;
        U :- new A;
        inner block: begin class B; … detach; … ;
                ref(B) X;
                ref(A) W;
                V :- W :- new A;
                X :- new B;
                . . .
         pc → L: call(X);
                . . .
        end inner block;
      . . .
      call(V);
      . . .
end outer block;
```

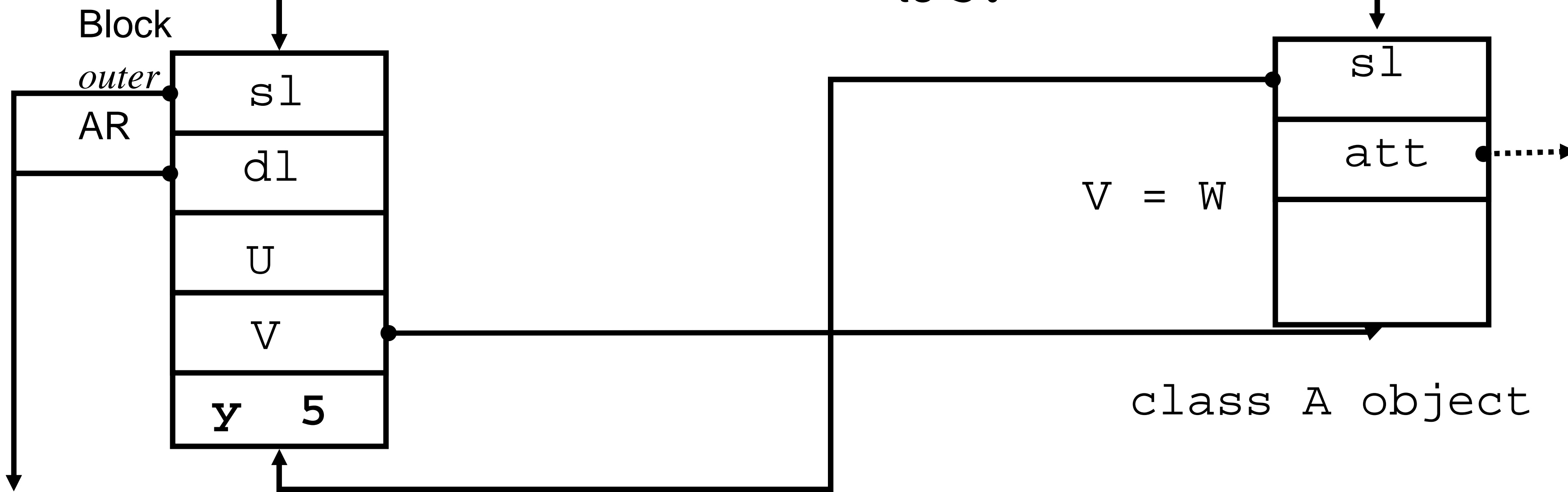
Example: picture at pc →



Example (cont.): Static Links

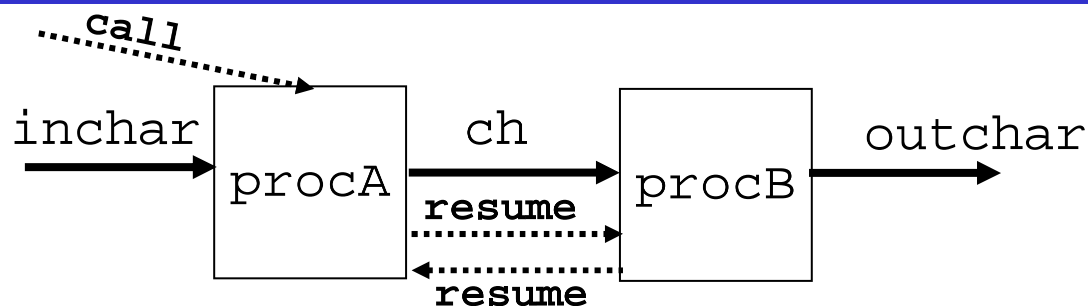






- Why is static link from `V = W` to block *outer* AR?
- `V :- W :- new(A)` done in block *inner*
- Static binding says static environment is where name is declared (space allocated)
- If static link to *inner*, **`y`** resolves to **`3!`**

Example: 2 coroutines




```
begin character ch;
  class aatob: ...below...end aatob;
  class bbtoc: ...below...end bbtoc;
  ref(aatob) procA ;
  ref(bbtoc) procB ;
  procA :- new aatob;
  procB :- new bbtoc;
  call(procA);
end
```

Example (cont.)

```
class aatob;
   begin
    detach;
    while true do begin
        ch := inchar;
        if ch = 'a' then
          begin ch := inchar;
                if ch = 'a' then
                     begin ch := 'b'; resume(procB) end
                else
                     begin character save;
                       save := ch;
                       ch := 'a'; resume(procB)
                       ch := save; resume(procB)
                     end
          end
         else resume(procB)
    end while
end aatob;
```

Example (cont.)

```
class bbtoc;
  begin
   detach;
   while true do begin
        if ch = 'b' then
           begin
             resume(procA)
             if ch = 'b' then
               outchar('c')
             else
               begin outchar('b'); outchar(ch) end
           end
        else outchar(ch);
        resume(procA)
   end while
end bbtoc;
```

Example (cont.)

- **`stdin:`**

```
bbbbb
aaaaa
ababab
aabbaabb
xaaaaxaabbxababxaaayyy
bbbbbb
aaaaaa
^D
```

- **`stdout:`**

```
ccb
ca
ababab
ccc
xcxcbxababxbayyy
ccc
cb
```