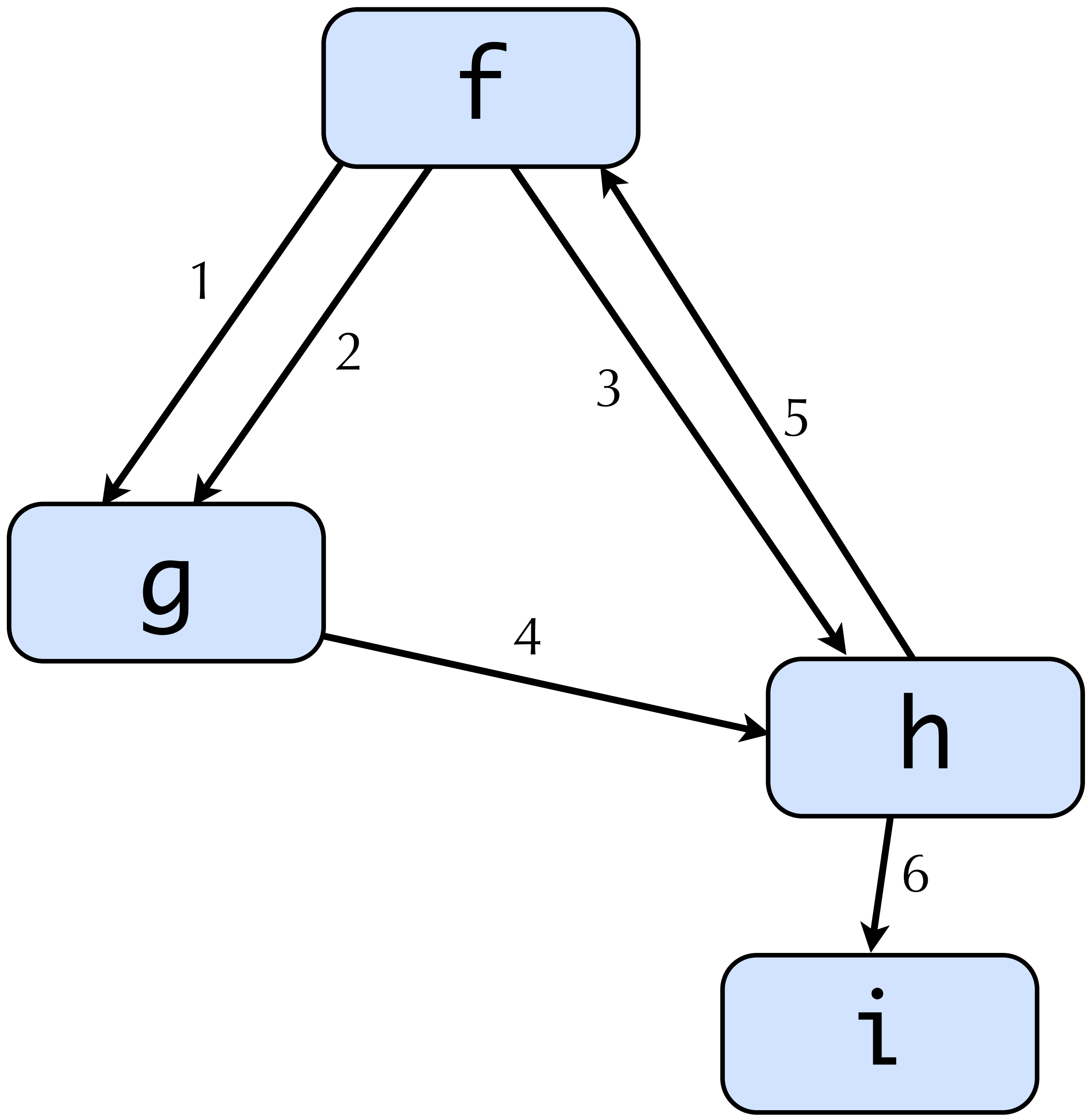
HARVARD
School of Engineering
and Applied Sciences

Interprocedural Analysis

CS252r Spring 2011

Call graph example

```
f() {
    1:  g();
    2:  g();
    3:  h();
}

g() {
    4: h();
}

h() {
    5: f();
    6: i();
}

i() { … }
```

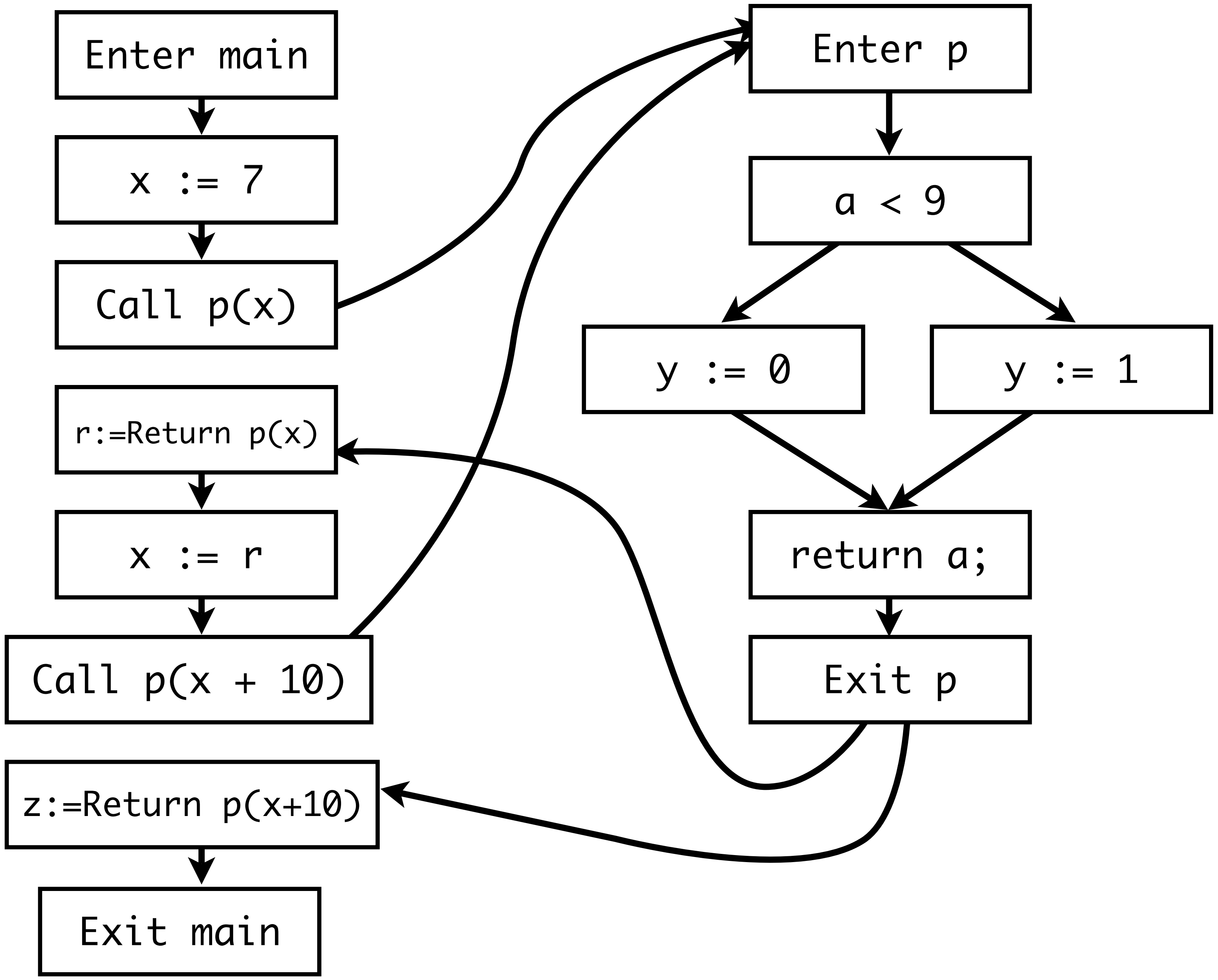


$$\begin{tikzcd}
 & f \arrow[dl, bend right, "1"'] \arrow[dl, "2"] \arrow[dr, "3"] & \\
g \arrow[rr, "4"] & & h \arrow[ul, bend right, "5"'] \arrow[d, "6"] \\
 & & i
\end{tikzcd}$$

Interprocedural dataflow analysis

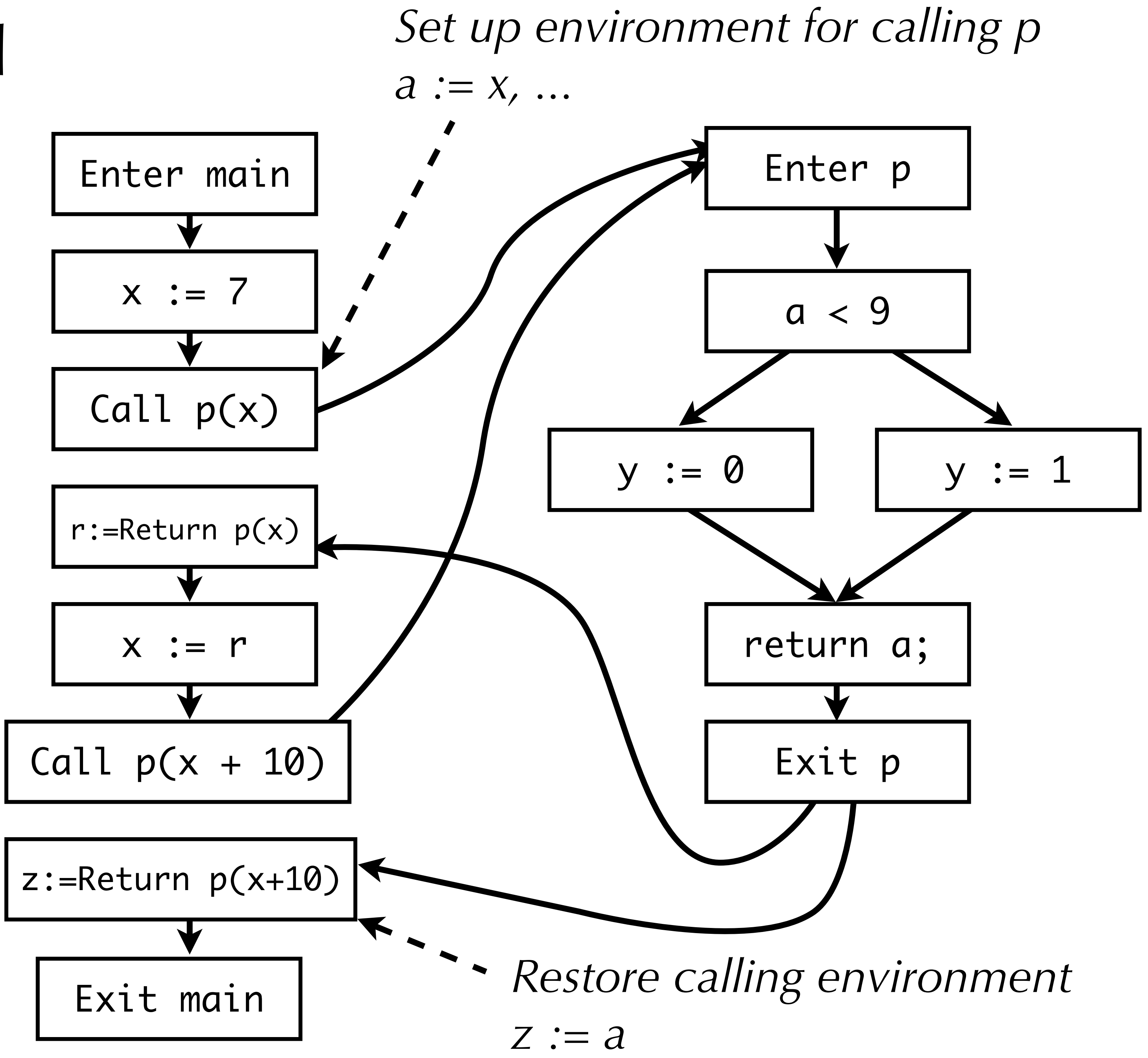
- How do we deal with procedure calls?
- Obvious idea: make one big CFG

```
main() {
   x := 7;
   r := p(x);
   x := r;
   z := p(x + 10);
}

p(int a) {
   if (a < 9)
      y := 0;
   else
      y := 1;
   return a;
}
```



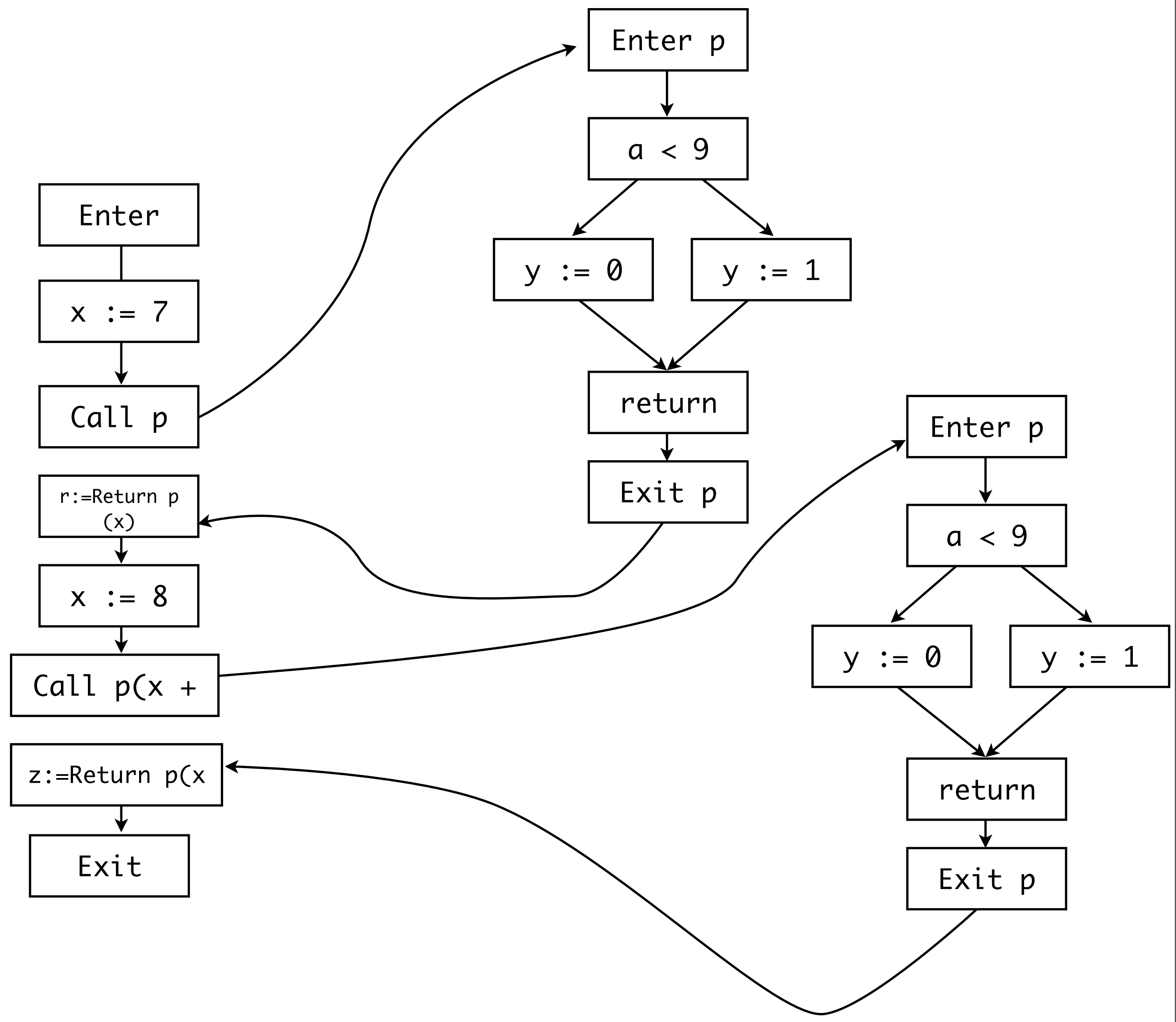
Interprocedural CFG

- CFG may have additional nodes to handle call and returns
  - Treat arguments, return values as assignments
- Note: a local program variable represents multiple locations



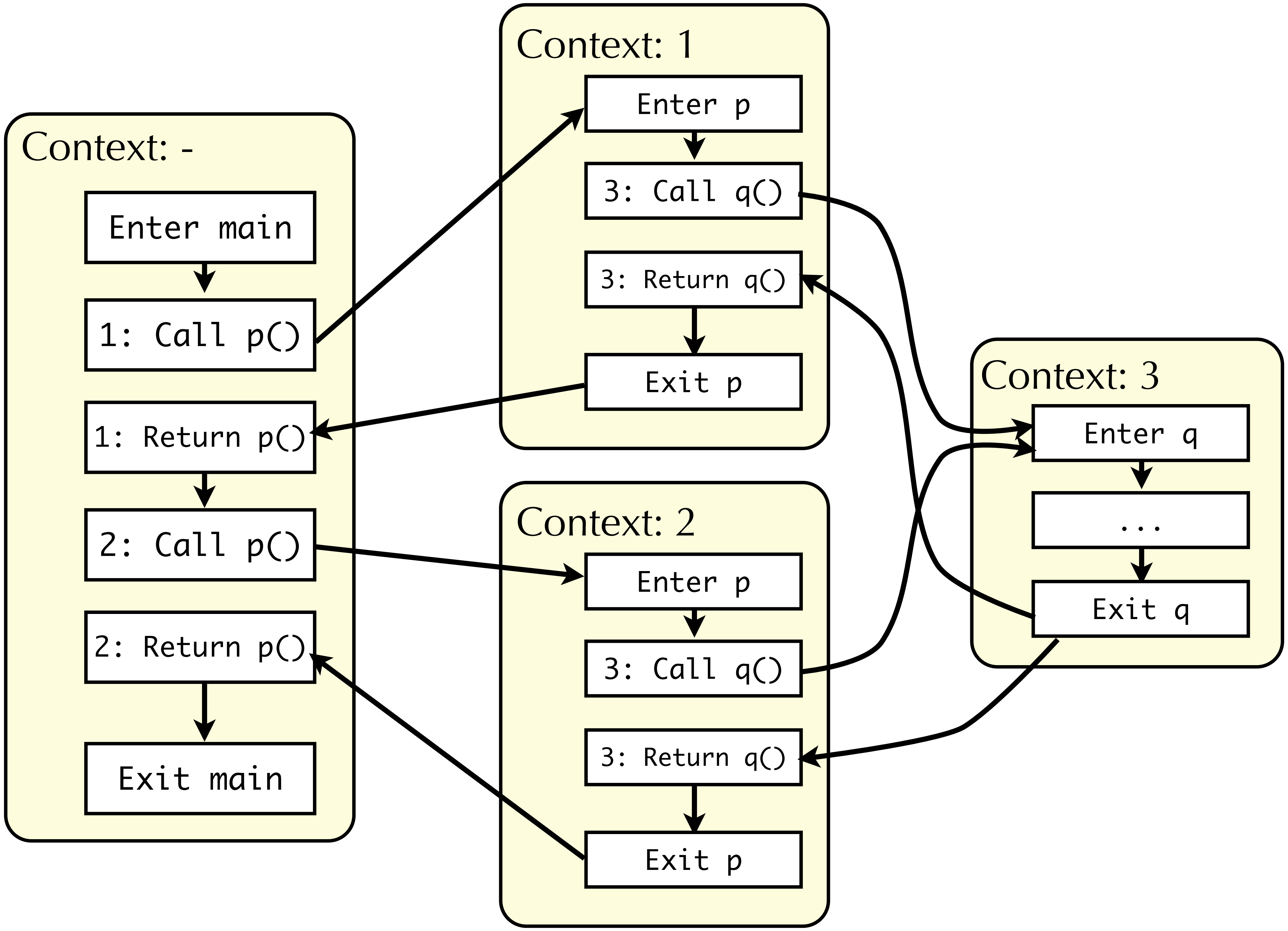
Inlining

- Inlining
  - Use a new copy of a procedure's CFG at each call site
- Problems? Concerns?
  - May be expensive! Exponential increase in size of CFG
    - p() { q(); q(); }   q() { r(); r() }   r() { … }
  - What about recursive procedures?
    - p(int n) { … p(n-1); … }
    - More generally, cycles in the call graph



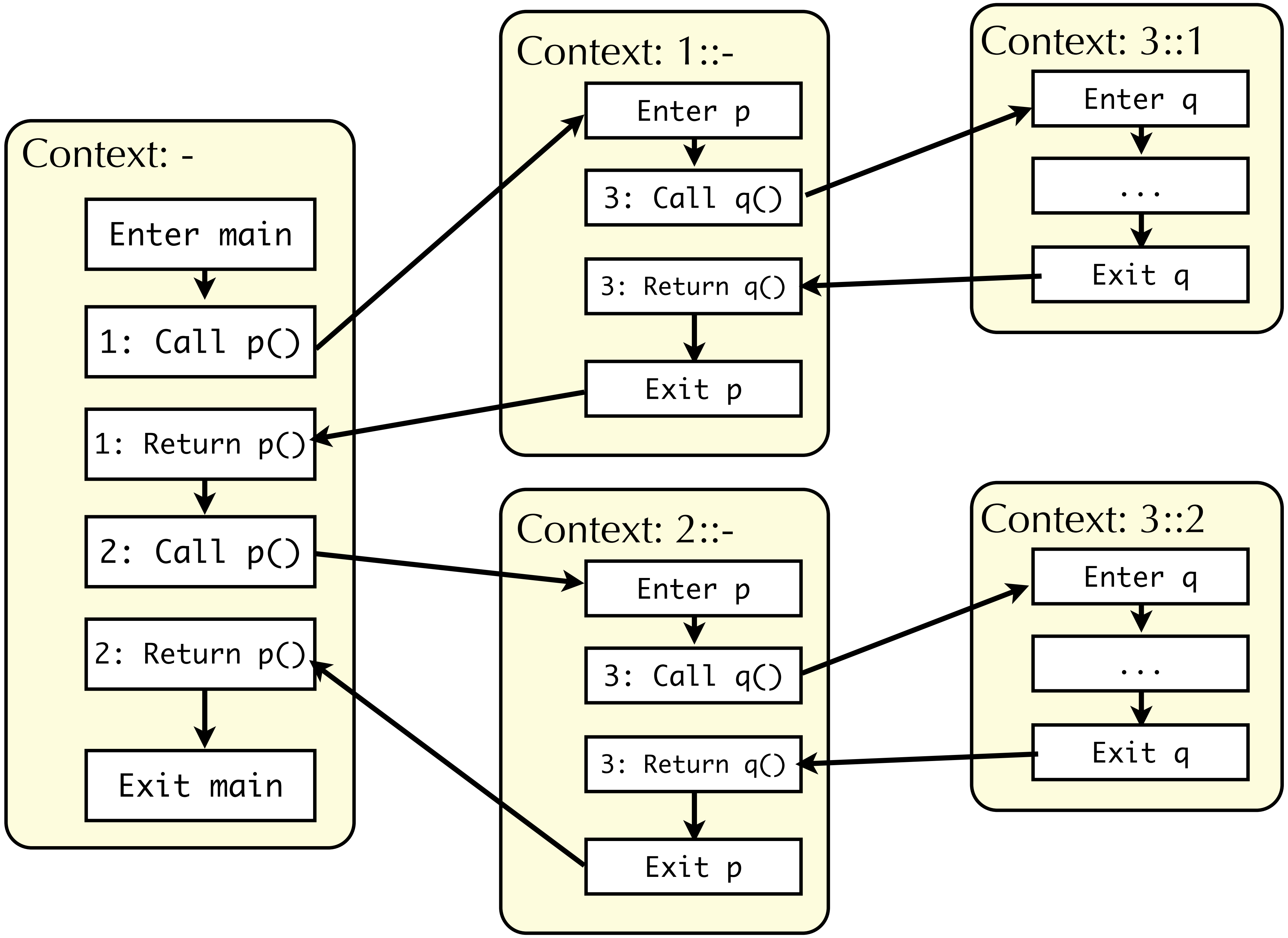
Context sensitivity example

```
main() {
  1: p();
  2: p();
}

p() {
  3: q();
}
q() {
  …
}
```



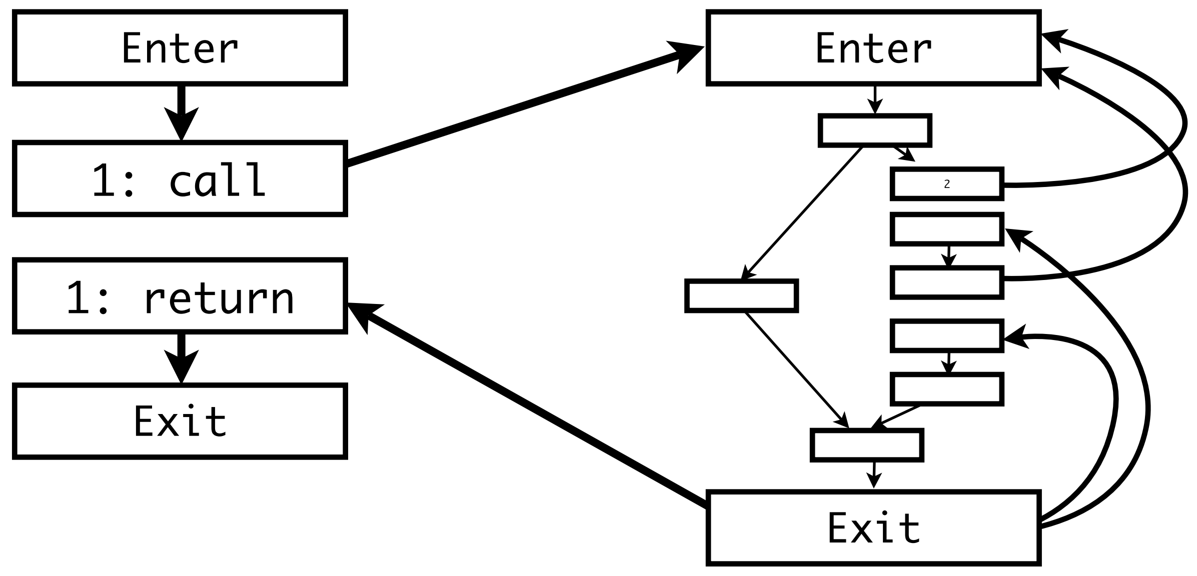
Context sensitivity example

```
main() {
  1: p();
  2: p();
}

p() {
  3: q();
}
q() {
  …
}
```



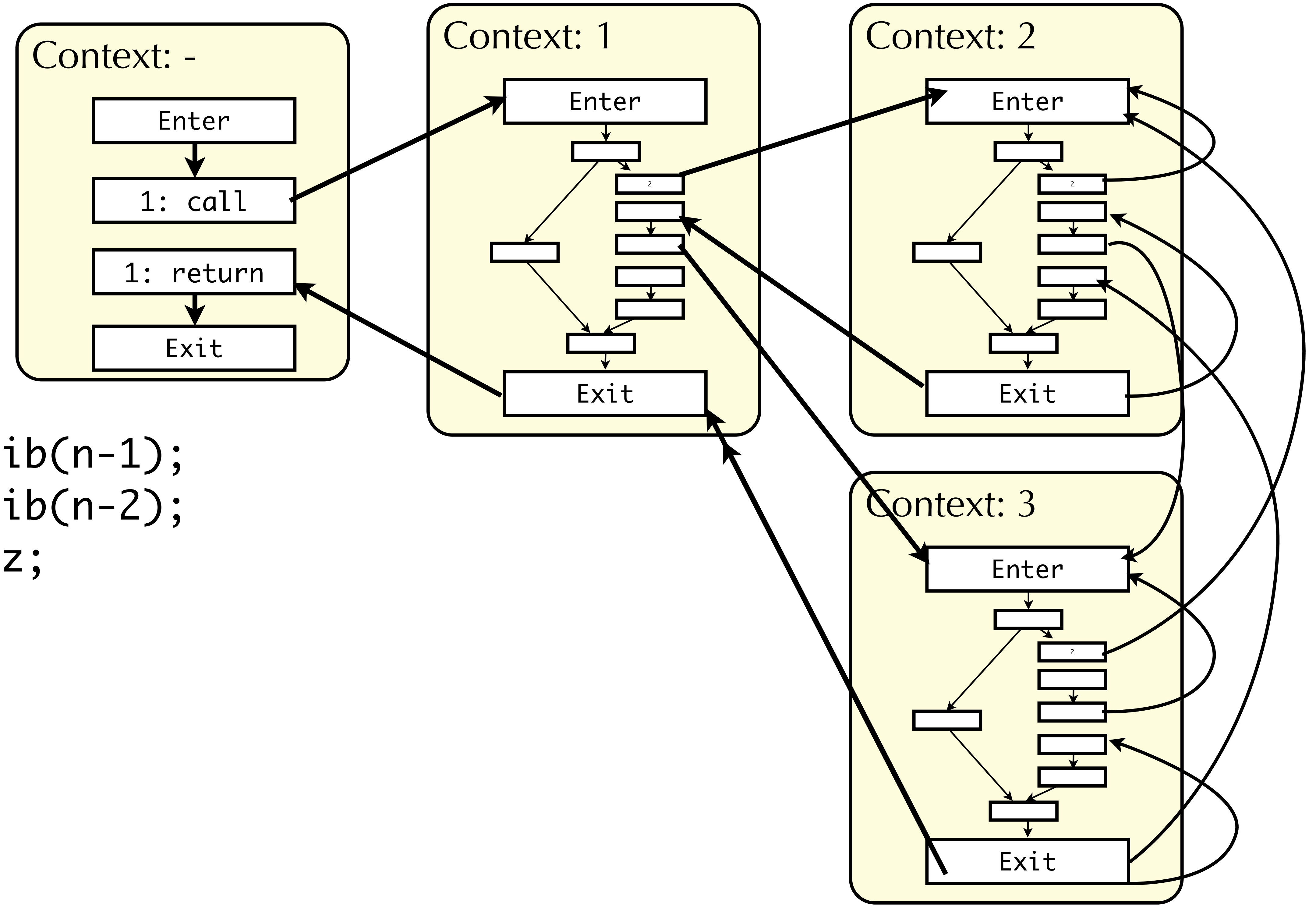
Fibonacci: context insensitive

```
main() {
  1: fib(7);
}

fib(int n) {
  if n <= 1
       x := 0
  else
    2: y := fib(n-1);
    3: z := fib(n-2);
       x:= y+z;
  return x;
}
```



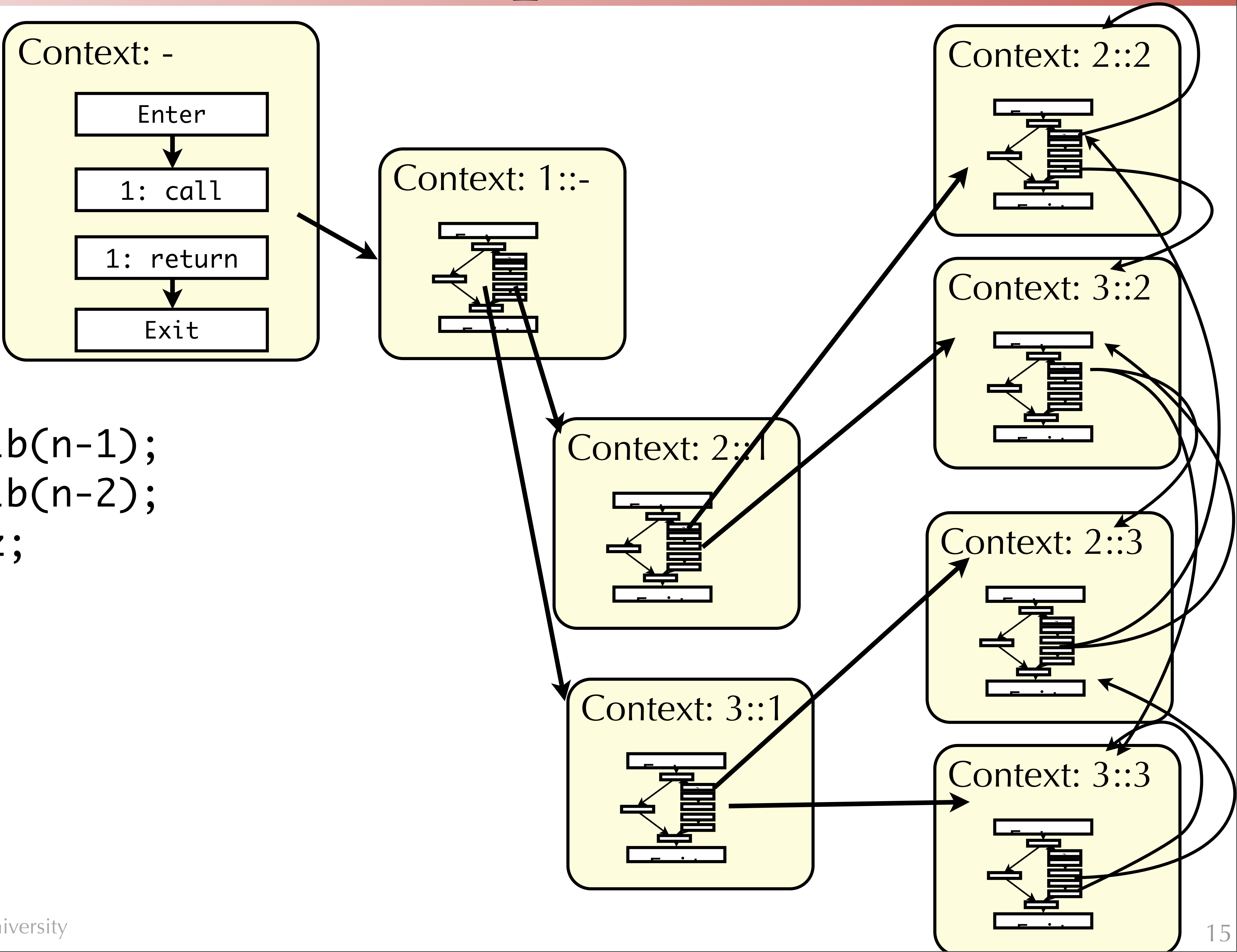
Fibonacci: context sensitive, stack depth 1

```
main() {
  1: fib(7);
}

fib(int n) {
  if n <= 1
       x := 0
  else
    2: y := fib(n-1);
    3: z := fib(n-2);
       x:= y+z;
  return x;
}
```



Fibonacci: context sensitive, stack depth 2

```
main() {
  1: fib(7);
}

fib(int n) {
  if n <= 1
       x := 0
  else
    2: y := fib(n-1);
    3: z := fib(n-2);
       x:= y+z;
  return x;
}
```


Context: -
Enter
1: call
1: return
Exit
Context: 1::-
Context: 2::1
Context: 3::1
Context: 2::2
Context: 3::2
Context: 2::3
Context: 3::3