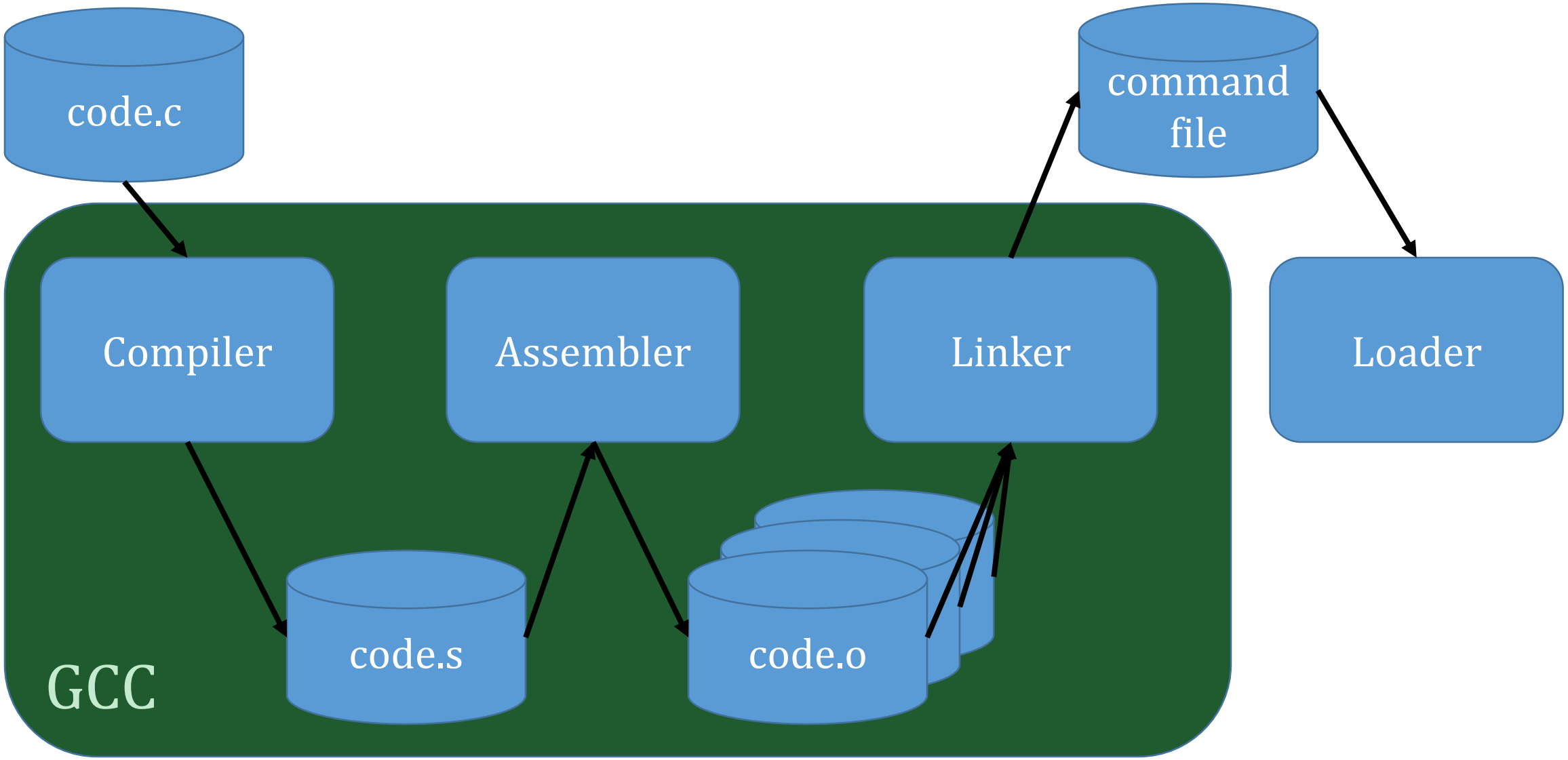
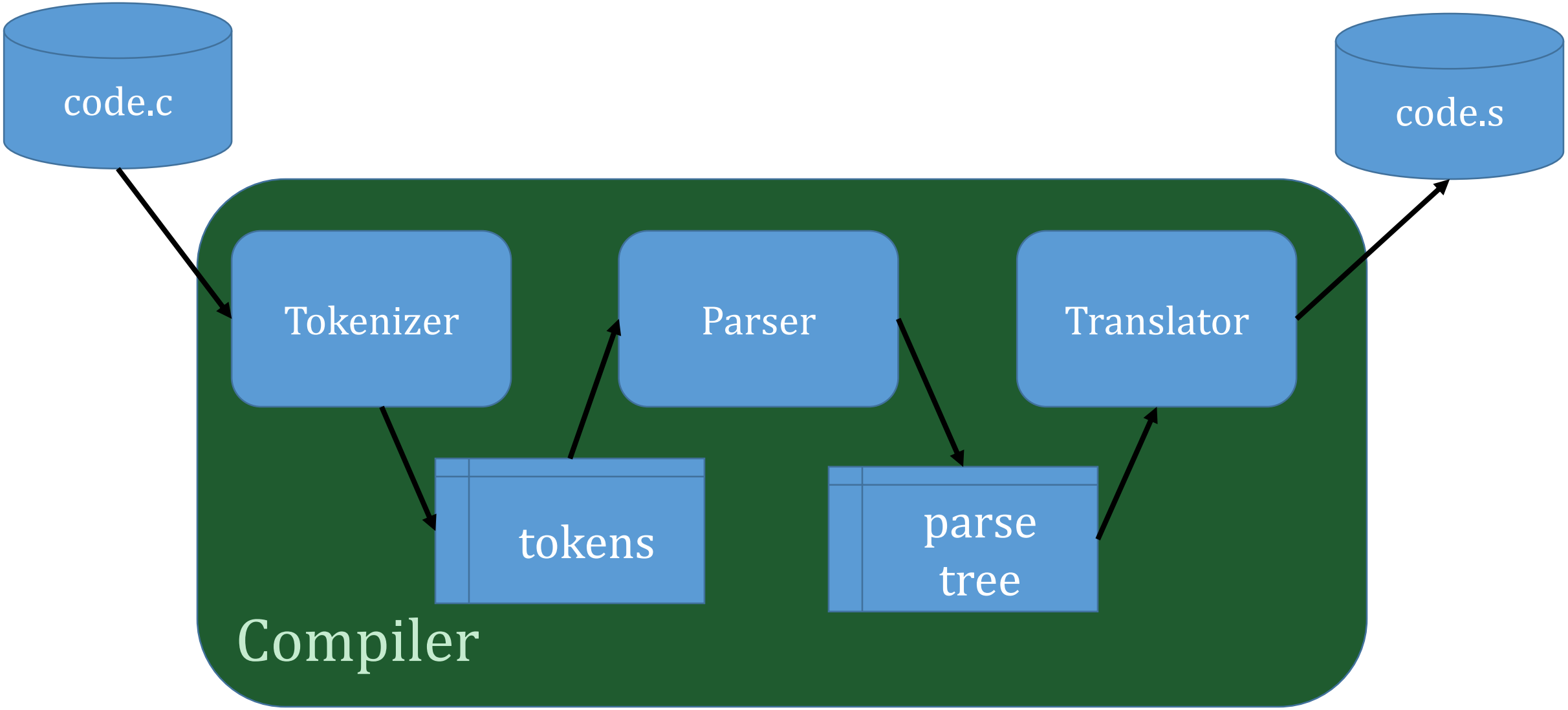
C Internals

Text: Introduction to Computer Systems : Chapters 11,14.3

From C to Execution



C Compiler Components



Example: Simple C Program

```
int bignuff(int x) { return (x>10)?1:0; }

int main(int argc,char **argv) {
        if (bignuff(12)) return 0;
        return 1;
}
```

Tokenizer

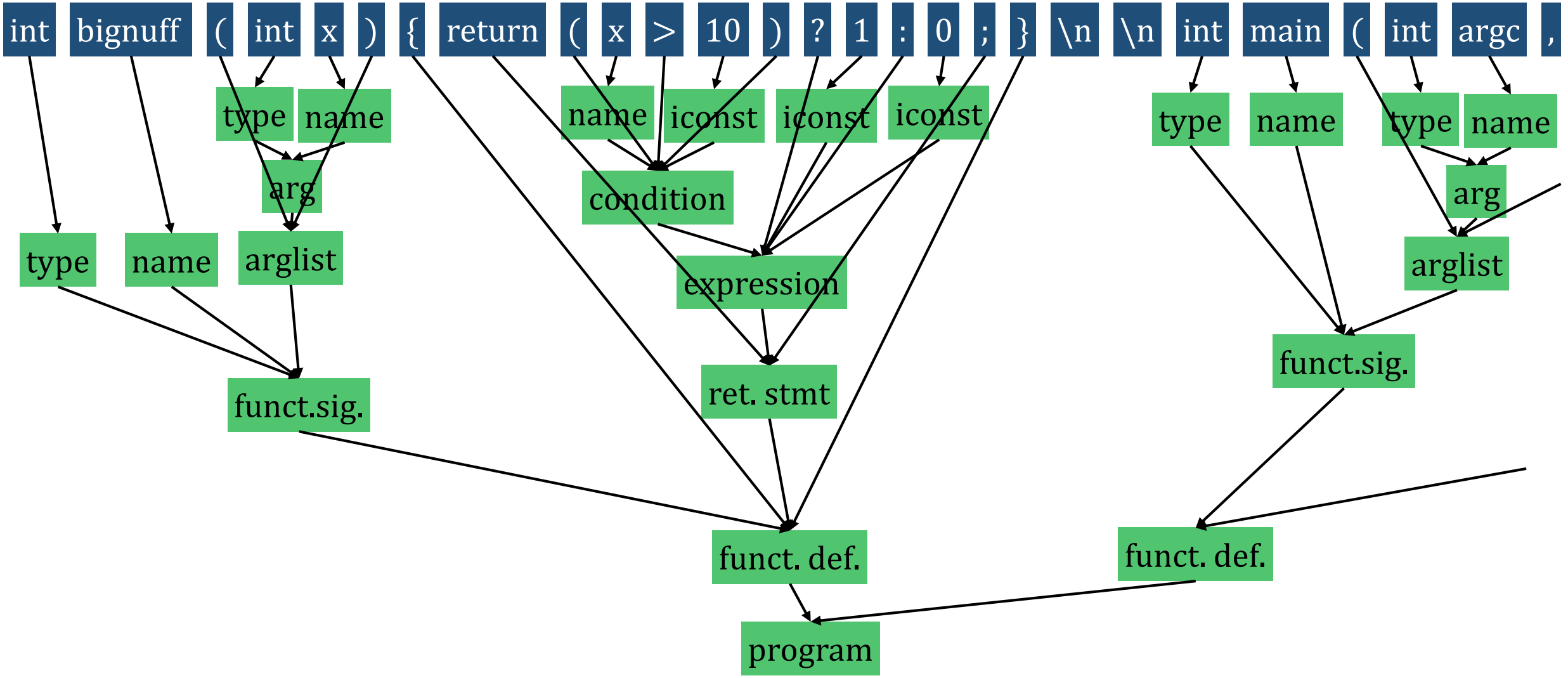
```
int bignuff(int x) { return (x>10)?1:0; }

int main(int argc,char **argv) {
        if (bignuff(12)) return 0;
        return 1;
}
```

int | bignuff | (| int | x |) | { | return | (| x | > | 10 |) | ? | 1 | : | 0 | ; | } | \n | \n | int | main | (| int | argc | ,

char | * | * | argv |) | { | \n | \t | if | (| bignuff | (| 12 |) | return | 0 | ; | \n | \t | return | 1 | ; | \n | }

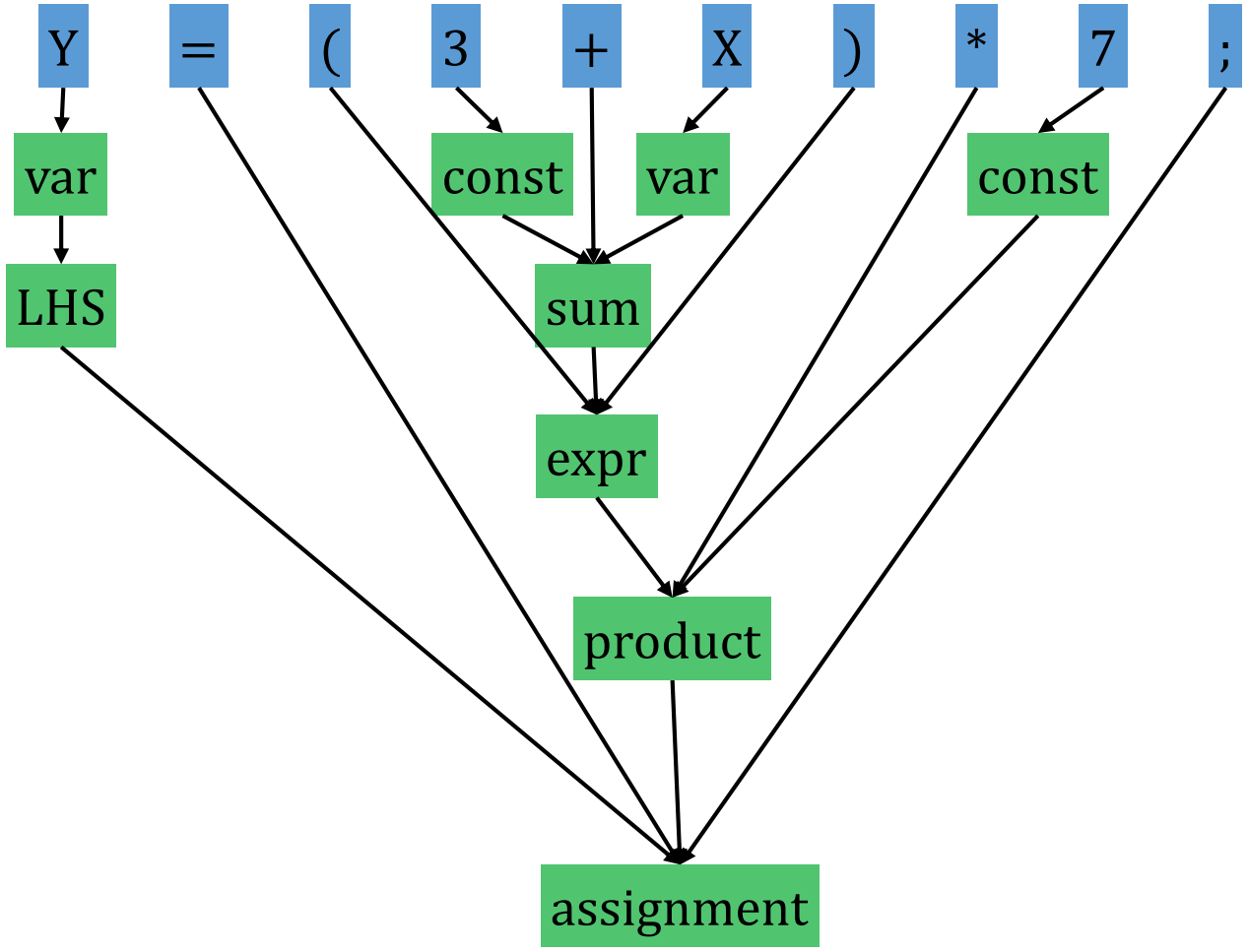
Parser


int
bignuff
(
int
x
)
{
return
(
x
>
10
)
?
1
:
0
;
}
\n
\n
int
main
(
int
argc
,
type
name
arg
arglist
name
iconst
iconst
iconst
condition
expression
ret. stmt
type
name
arglist
funct.sig.
type
name
arg
funct.sig.
funct. def.
funct. def.
program

Translator

- Reads tree starting at root program
- At each level
  - writes assembly code to start that level
  - invokes translator for each component
  - writes assembly code to finish that level

Translator Example




```
-> assignment
  -> product
    ->expr
      -> sum
          -> const:    and r0,r0,#0
          <- const:    add r0,r0,#3
          -> var:      ldr r1,r6,x1A
          <- var:
        <- sum:        add r0,r1,r0
      <- expr:
      -> const:        add r1,r1,#0
      <- const:        add r1,r1,#7
    <- product:        jsr multr0r1
<- assignment:         str r0,r6,#0C
```

Function Activation Frame

- "Frame" – list of data (not always same type)
- Activation Frame: Data associated with One Function Invocation
- Activation Frame Contents:
  - Return Address
  - Return Value
  - Argument Values
  - Local Variable Values
  - Saved Register Values
- Created on function invocation
- Deleted after function returns

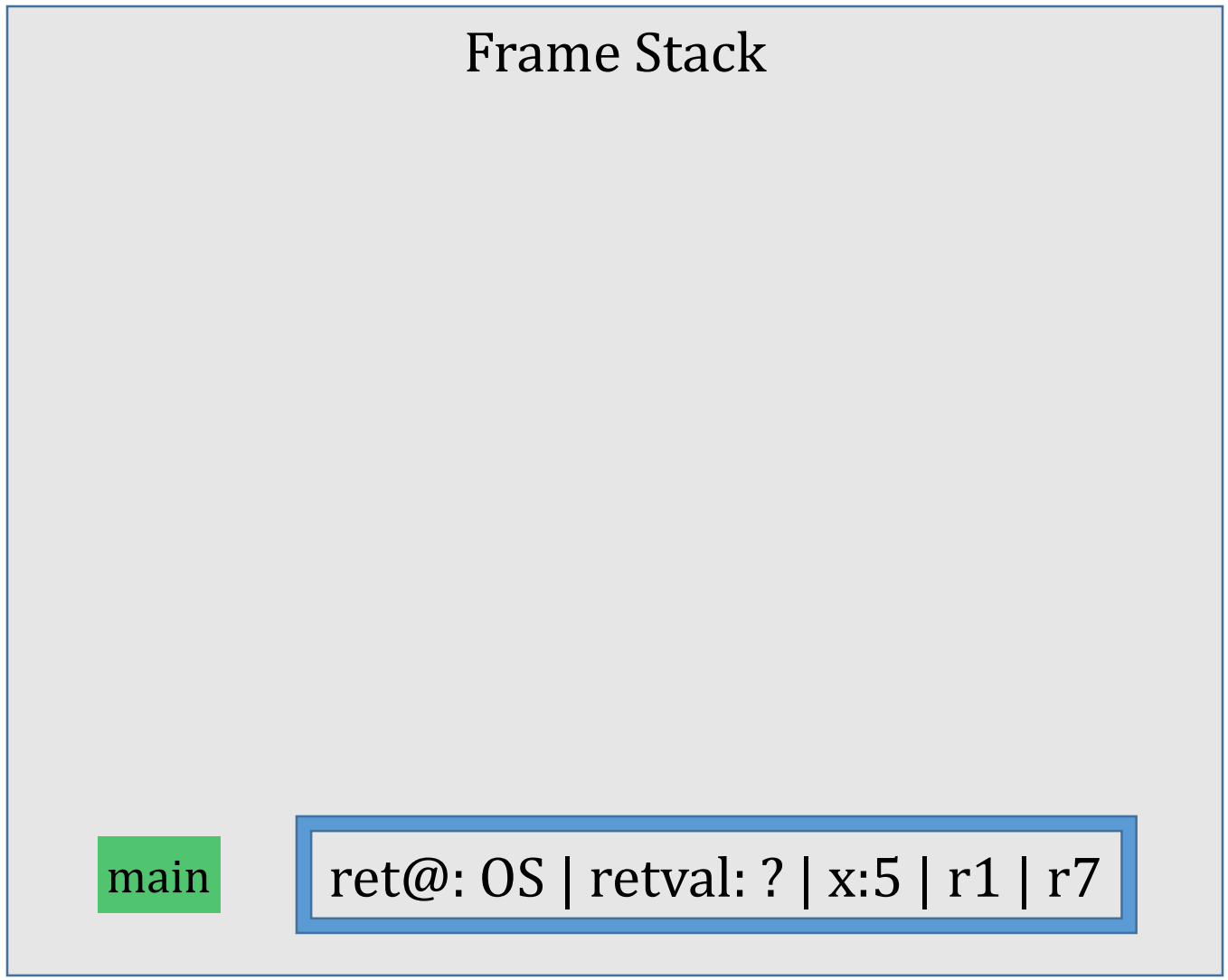
| ret@ | retval | arg1 | arg2 | x | y | r0 | r1 | r7 |

Frame Stack

- Stack of Activation Frames
- Function Invocation ⇔ Push Activation Frame on Frame stack
- Function return ⇔Pop Activation Frame off of Frame stack
- OS pushes “main” frame stack on stack frame when started
- Frame stack contains history of who’s calling whom

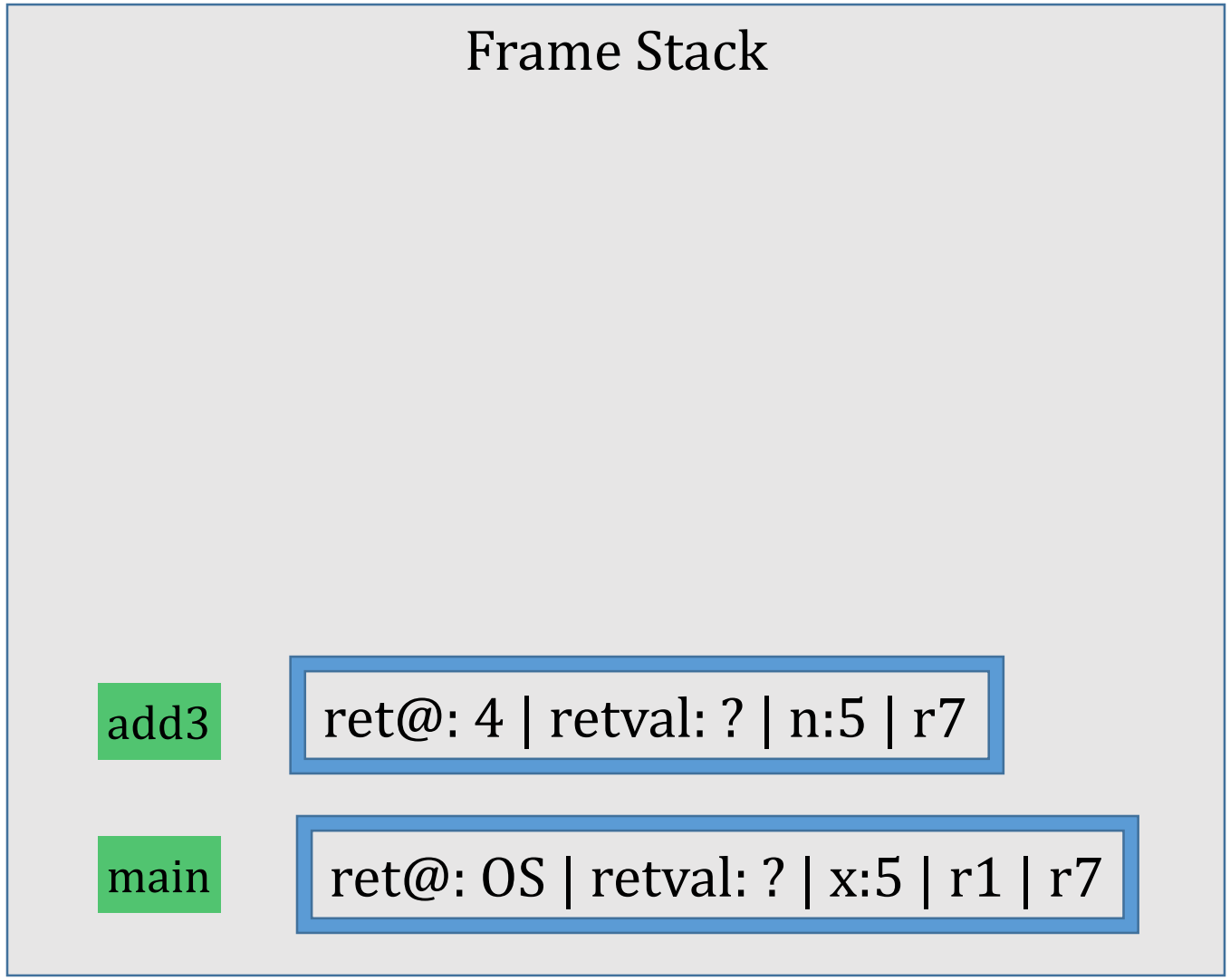
Frame Stack Example

```
1: int main() { int x=5;
2:    return (bignuff(
3:       add3(
4:          add3(x)))
5:             ?0:x; }
6: int bignuff(int n) {
7:    if (n>10) return 1;
8:    return 0; }
9: int add3(int n) {
10: n=n+3;
11: return n; }
```



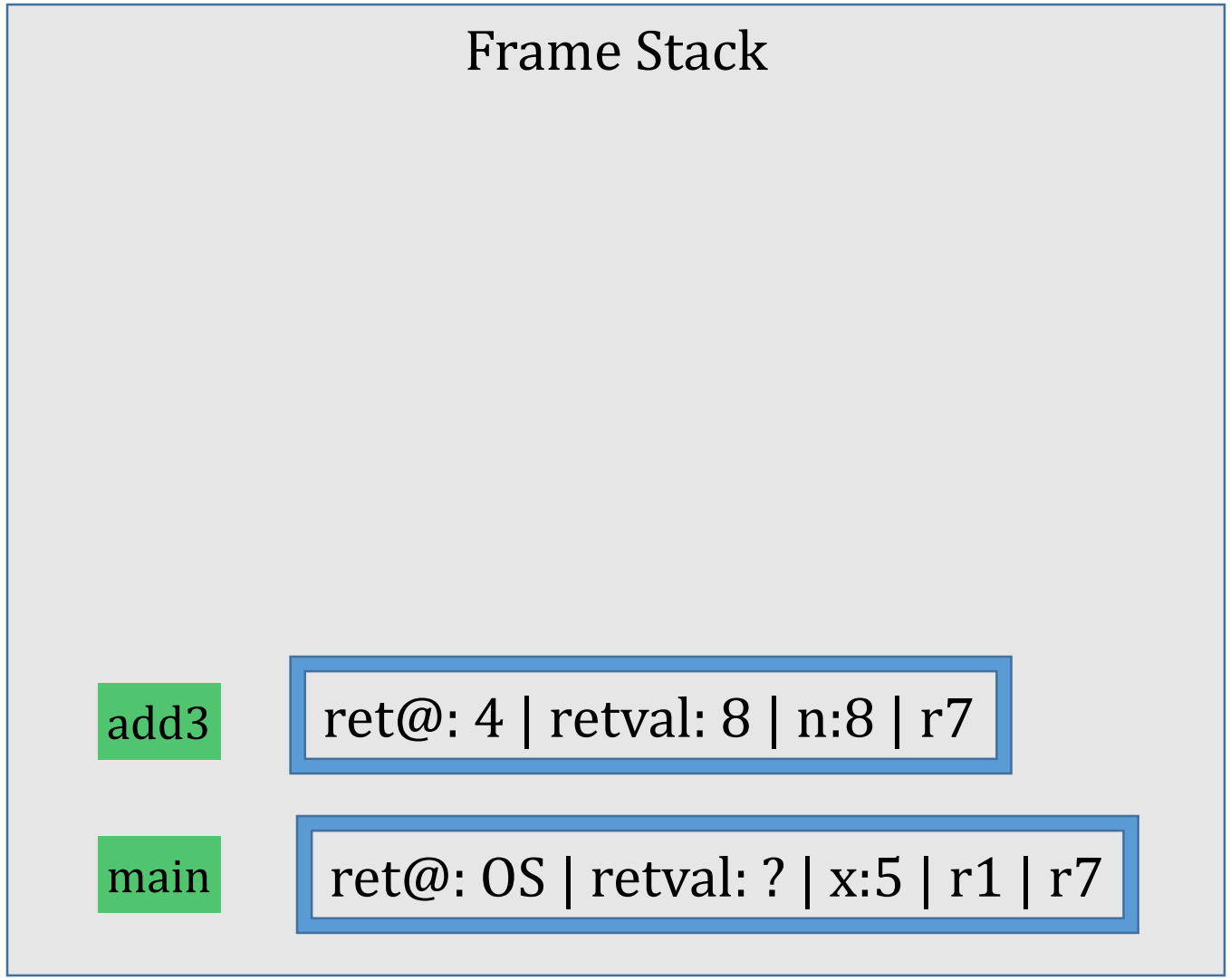
Frame Stack Example

```
1: int main() { int x=5;
2:    return (bignuff(
3:       add3(
4:          add3(x)))
5:             ?0:x; }
6: int bignuff(int n) {
7:    if (n>10) return 1;
8:    return 0; }
9: int add3(int n) {
10: n=n+3;
11: return n; }
```



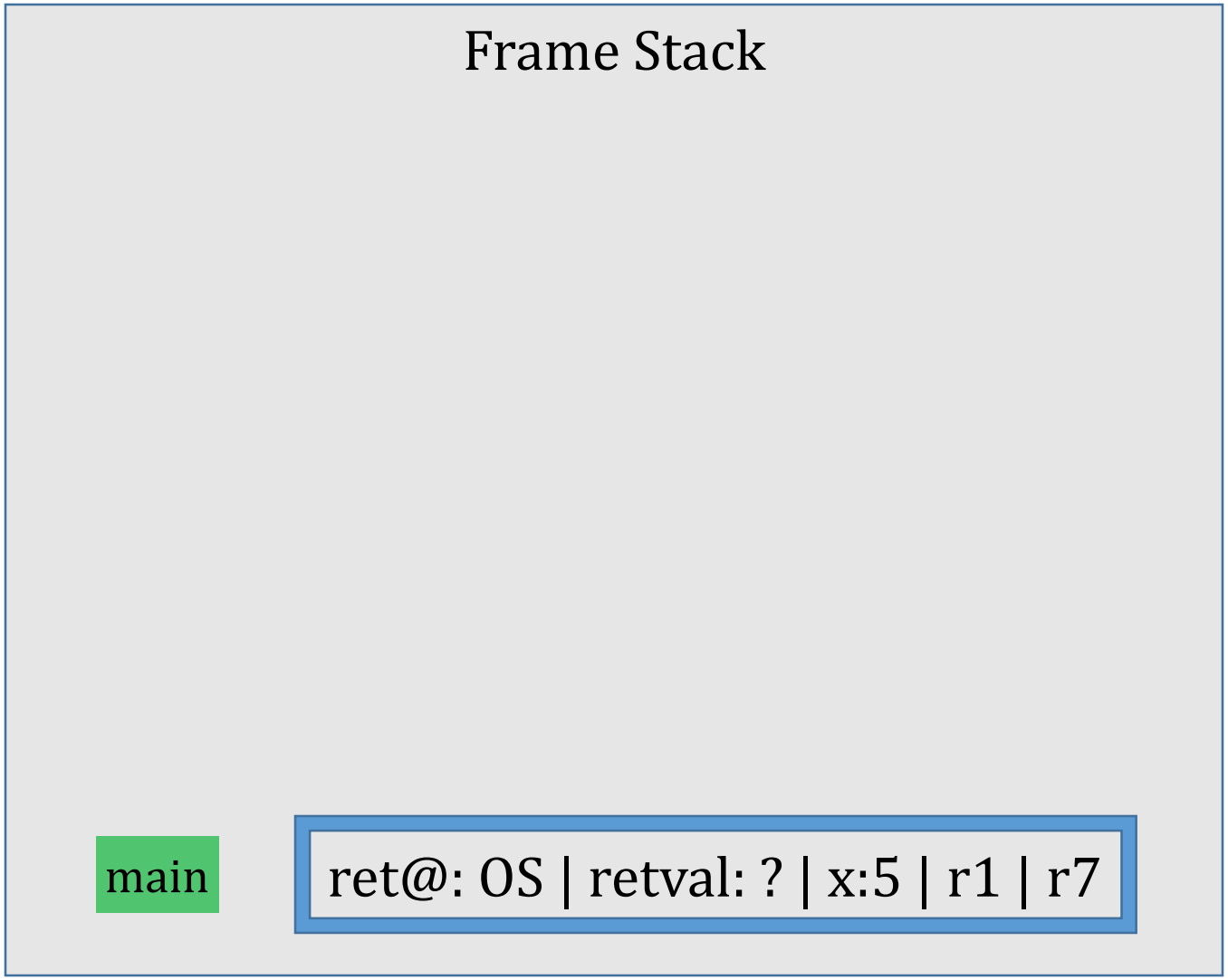
Frame Stack Example

```
1: int main() { int x=5;
2:    return (bignuff(
3:       add3(
4:          add3(x)))
5:             ?0:x; }
6: int bignuff(int n) {
7:    if (n>10) return 1;
8:    return 0; }
9: int add3(int n) {
10: n=n+3;
11: return n; }
```



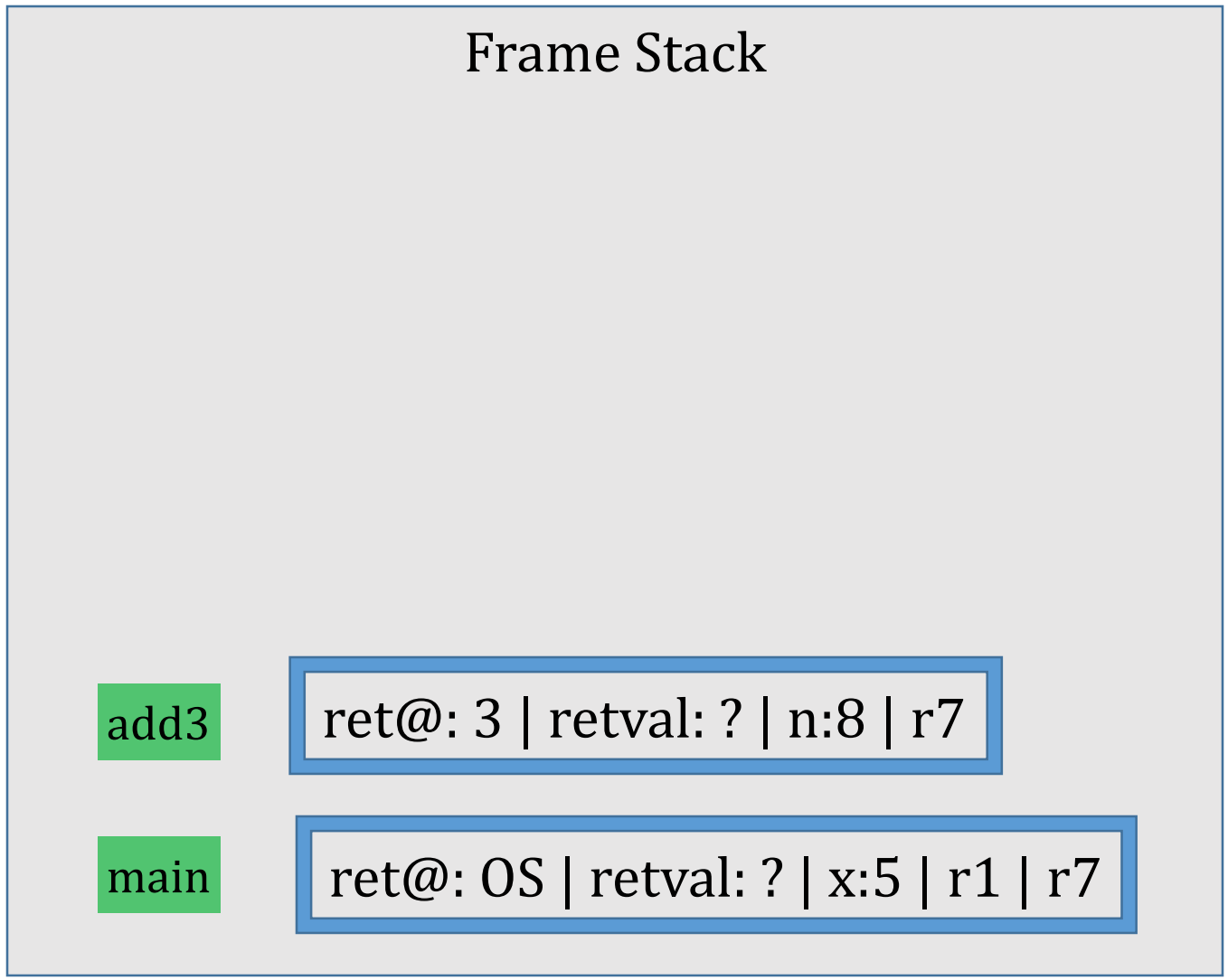
Frame Stack Example

```
1: int main() { int x=5;
2:    return (bignuff(
3:       add3(
4:          add3(x)))
5:             ?0:x; }
6: int bignuff(int n) {
7:    if (n>10) return 1;
8:    return 0; }
9: int add3(int n) {
10: n=n+3;
11: return n; }
```



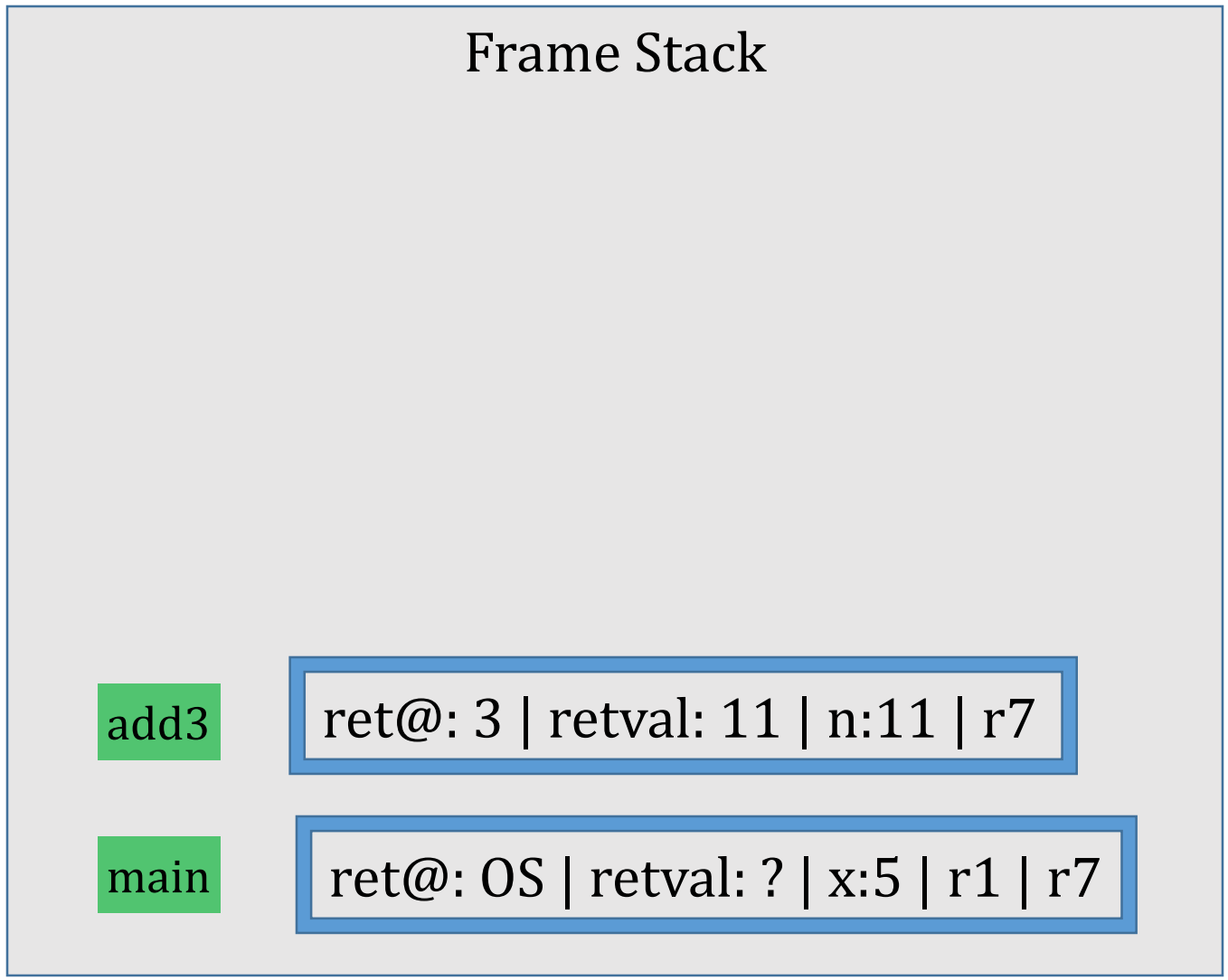
Frame Stack Example

```
1: int main() { int x=5;
2:    return (bignuff(
3:       add3(
4:          add3(x)))
5:             ?0:x; }
6: int bignuff(int n) {
7:    if (n>10) return 1;
8:    return 0; }
9: int add3(int n) {
10: n=n+3;
11: return n; }
```



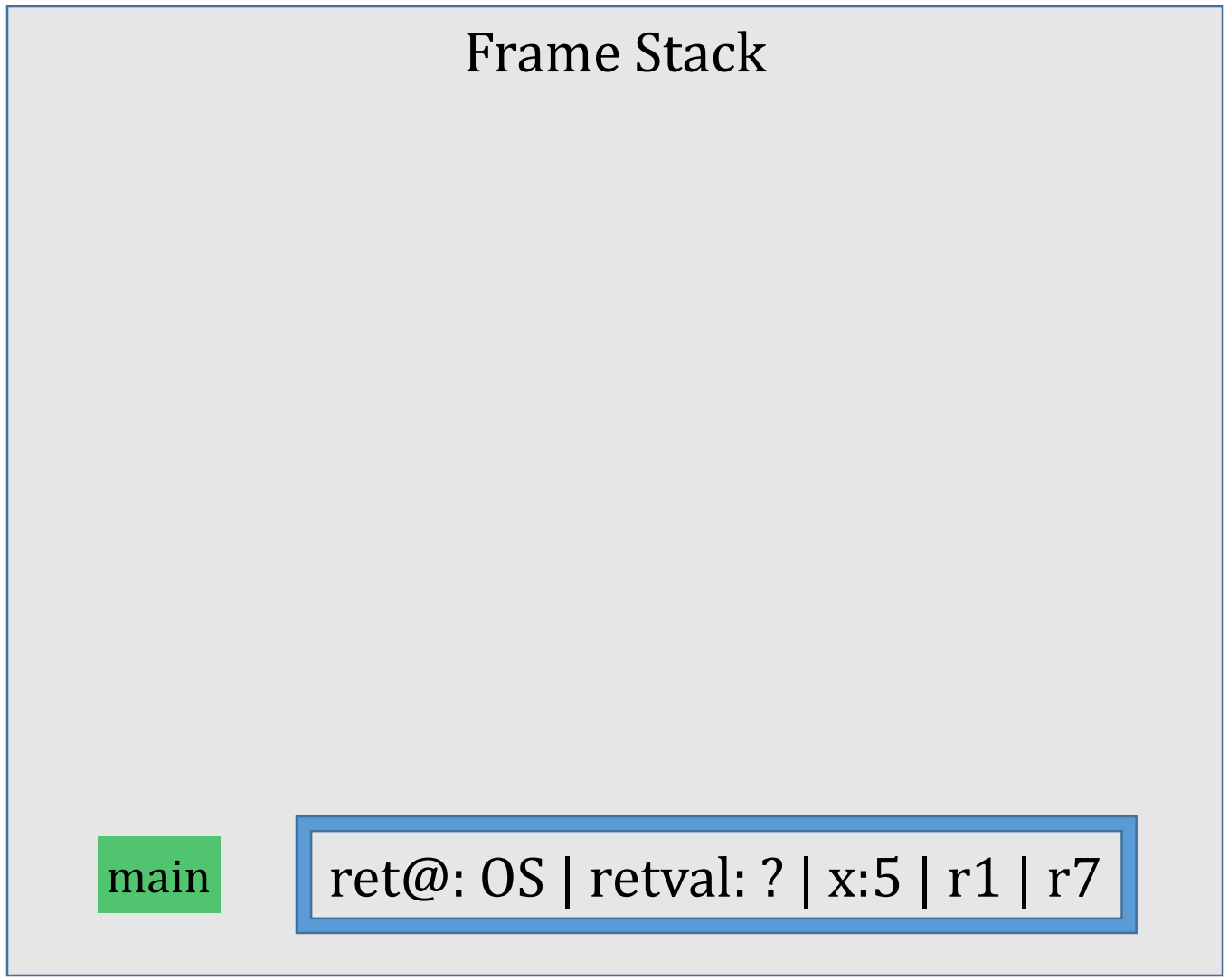
Frame Stack Example

```
1: int main() { int x=5;
2:    return (bignuff(
3:       add3(
4:          add3(x)))
5:             ?0:x; }
6: int bignuff(int n) {
7:    if (n>10) return 1;
8:    return 0; }
9: int add3(int n) {
10: n=n+3;
11: return n; }
```



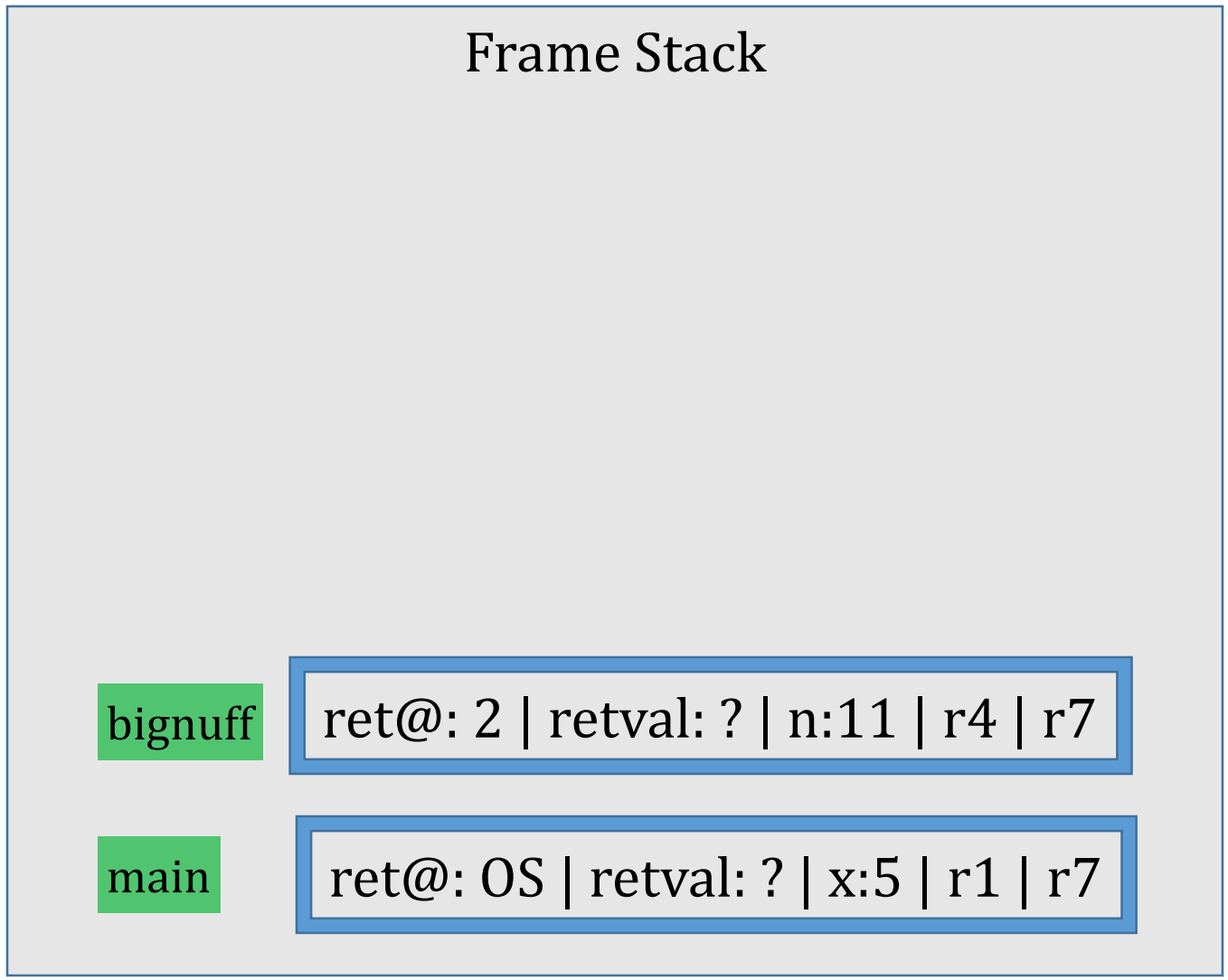
Frame Stack Example

```
1: int main() { int x=5;
2:    return (bignuff(
3:       add3(
4:          add3(x)))
5:             ?0:x; }
6: int bignuff(int n) {
7:    if (n>10) return 1;
8:    return 0; }
9: int add3(int n) {
10: n=n+3;
11: return n; }
```



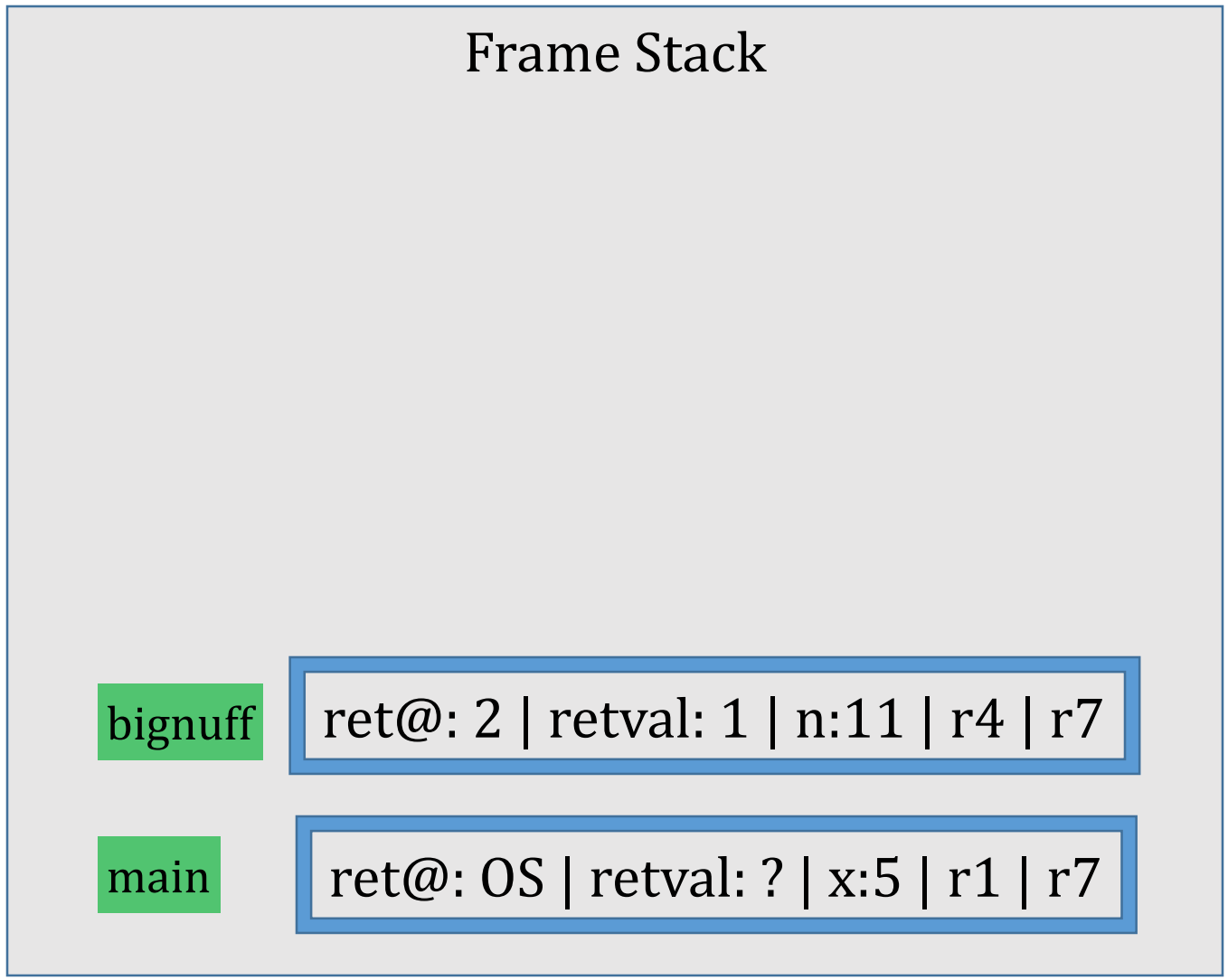
Frame Stack Example

```
1: int main() { int x=5;
2:    return (bignuff(
3:       add3(
4:          add3(x)))
5:             ?0:x; }
6: int bignuff(int n) {
7:    if (n>10) return 1;
8:    return 0; }
9: int add3(int n) {
10: n=n+3;
11: return n; }
```



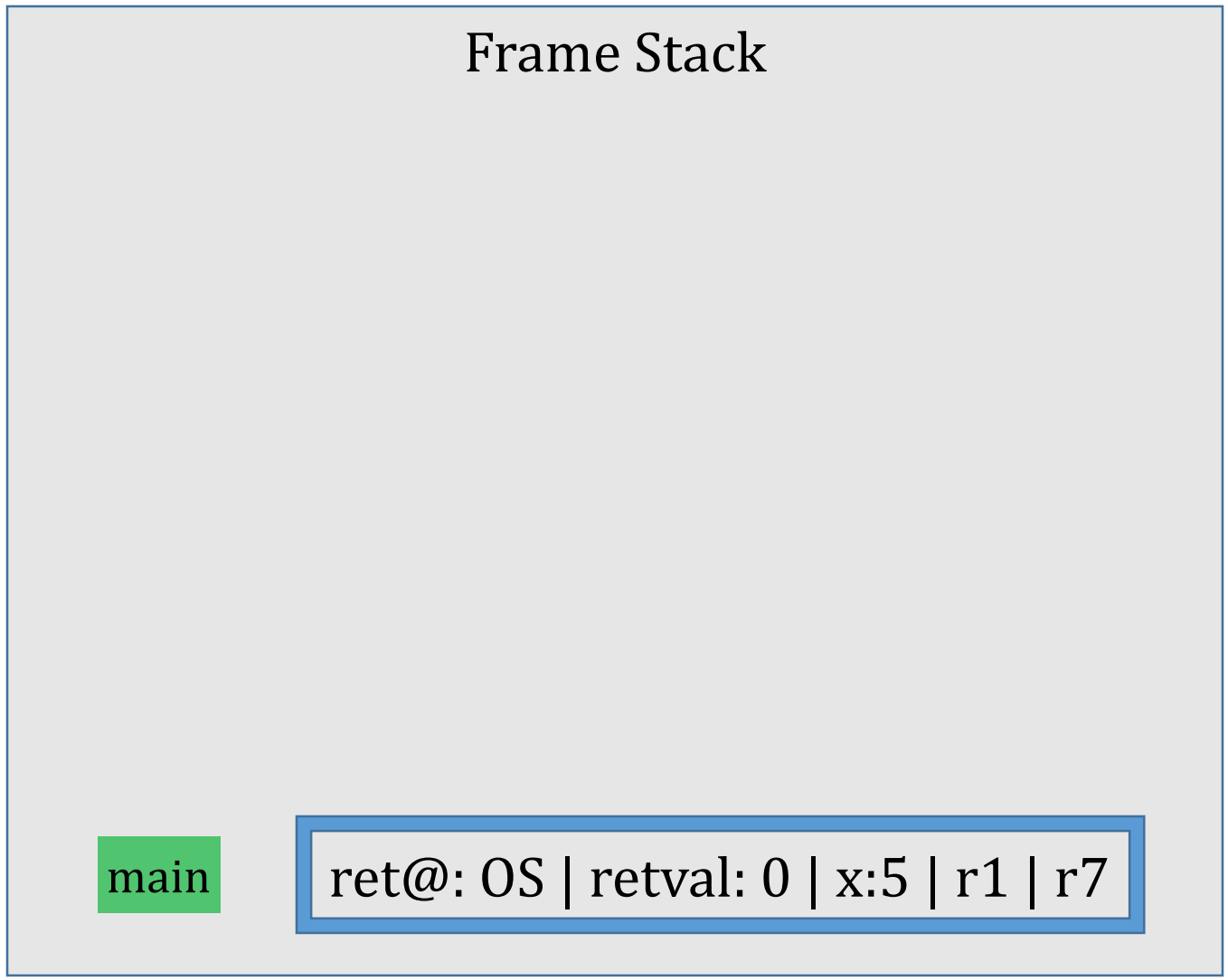
Frame Stack Example

```
1: int main() { int x=5;
2:    return (bignuff(
3:       add3(
4:          add3(x)))
5:             ?0:x; }
6: int bignuff(int n) {
7:    if (n>10) return 1;
8:    return 0; }
9: int add3(int n) {
10: n=n+3;
11: return n; }
```



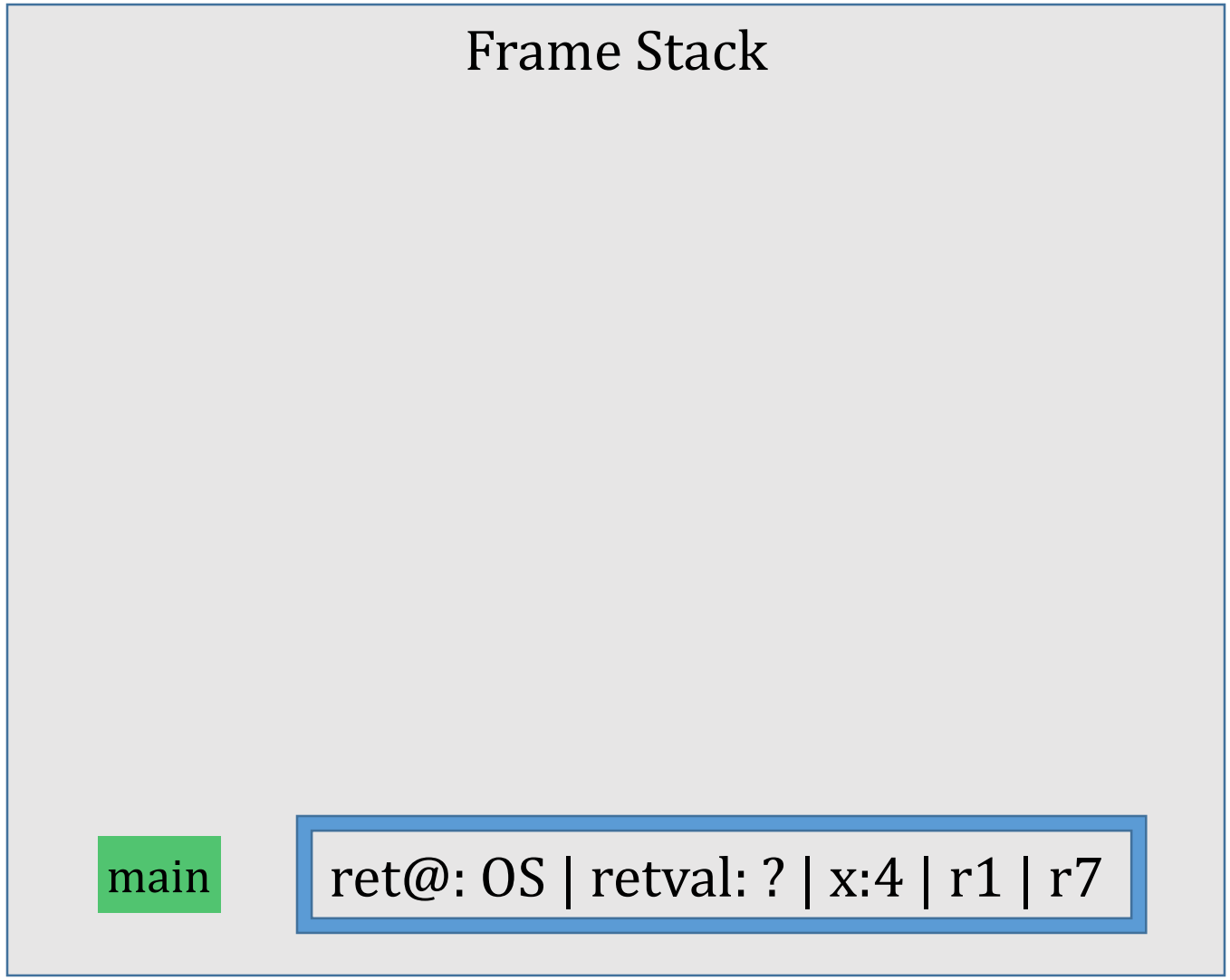
Frame Stack Example

```
1: int main() { int x=5;
2:    return (bignuff(
3:       add3(
4:          add3(x)))
5:             ?0:x; }
6: int bignuff(int n) {
7:    if (n>10) return 1;
8:    return 0; }
9: int add3(int n) {
10: n=n+3;
11: return n; }
```



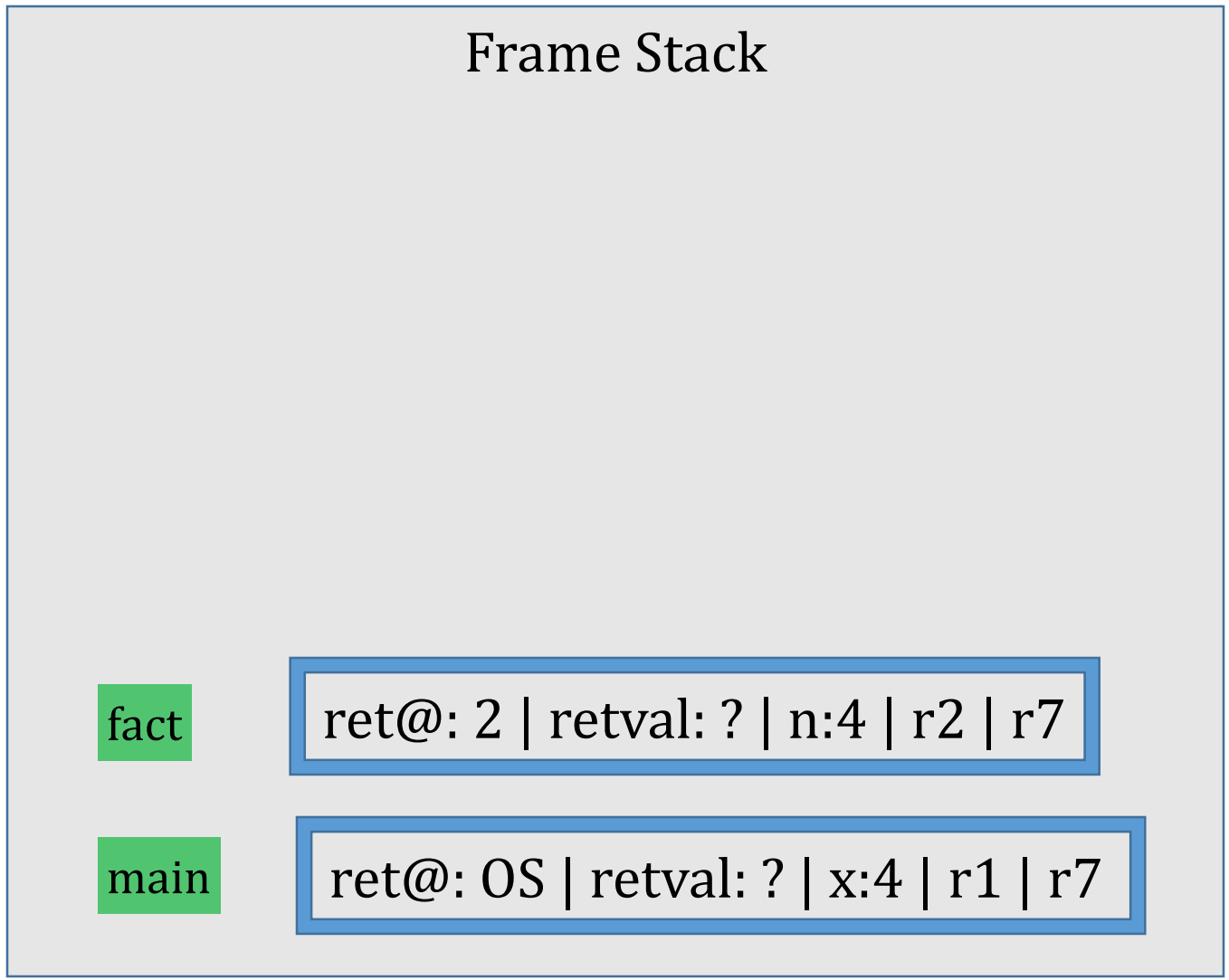
Frame Stack and Recursion

```
1: int main() { int x=4;
2:   return ( fact(x) );
3:
4: int fact(int n) {
5:       if (n<2) return 1;
6:       return  n *
7:             fact(n-1); }
```



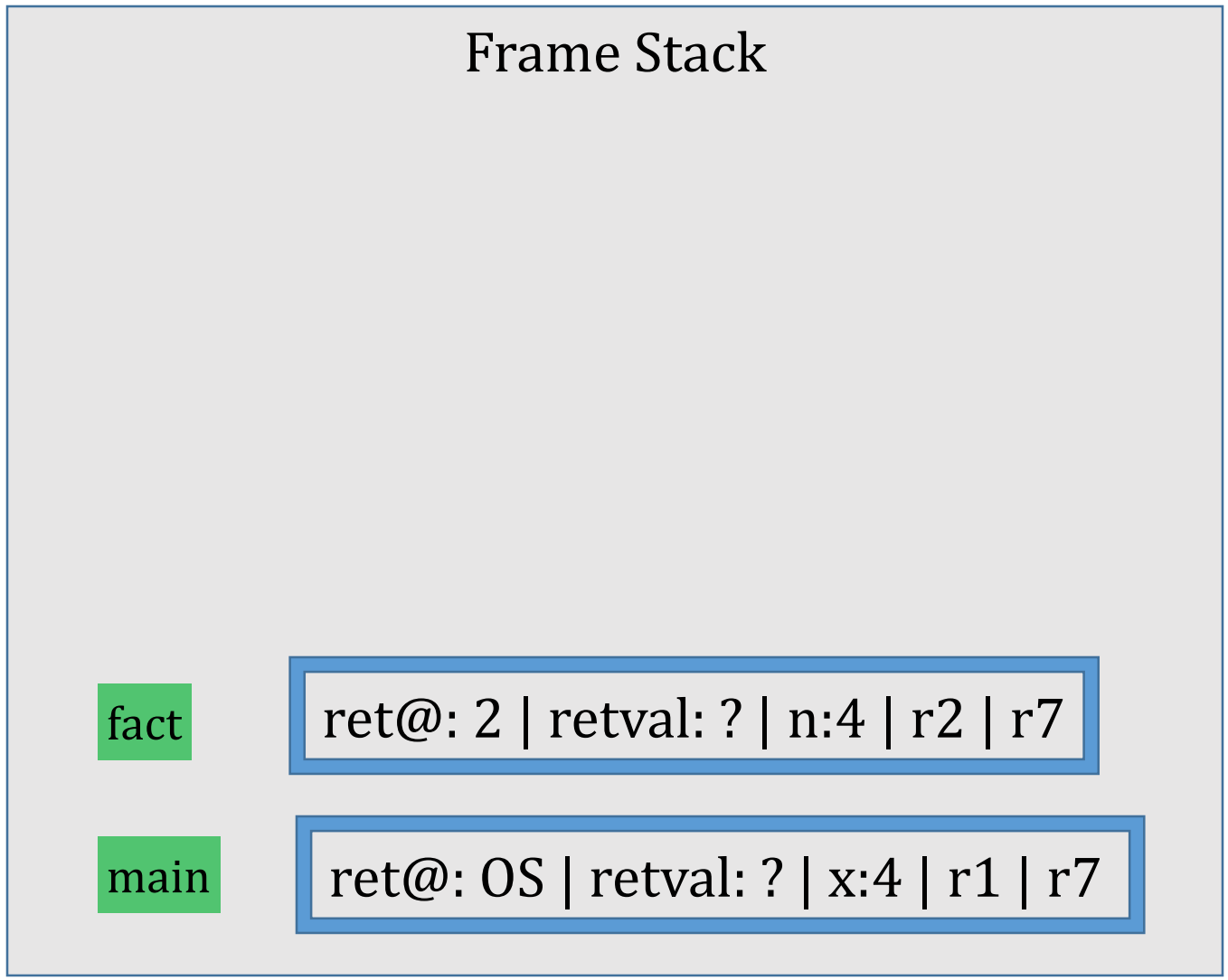
Frame Stack and Recursion

```
1: int main() { int x=4;
2:   return ( fact(x) );
3:
4: int fact(int n) {
5:      if (n<2) return 1;
6:      return  n *
7:           fact(n-1); }
```



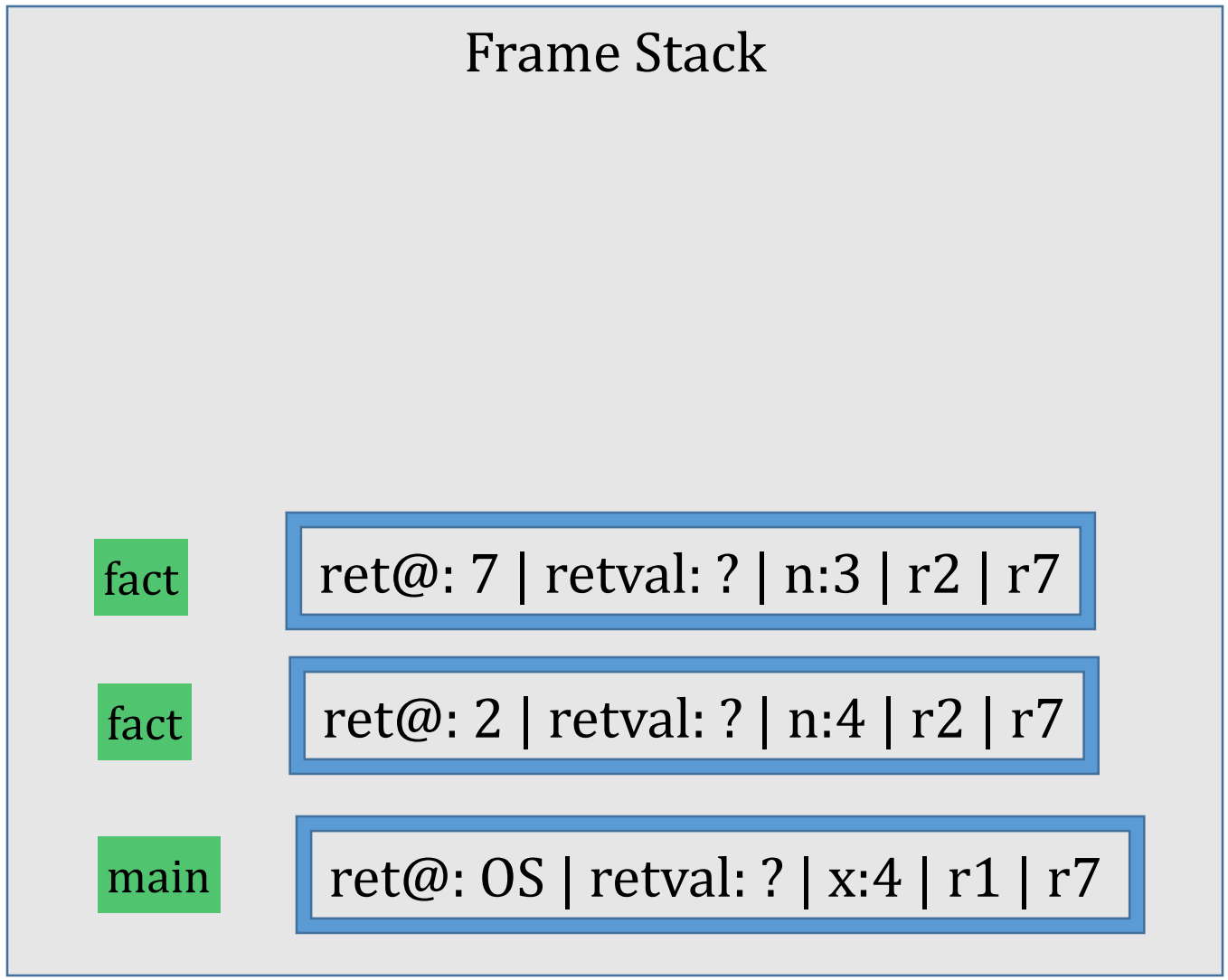
Frame Stack and Recursion

```
1: int main() { int x=4;
2:   return ( fact(x) );
3:
4: int fact(int n) {
5:      if (n<2) return 1;
6:      return  n *
7:           fact(n-1); }
```



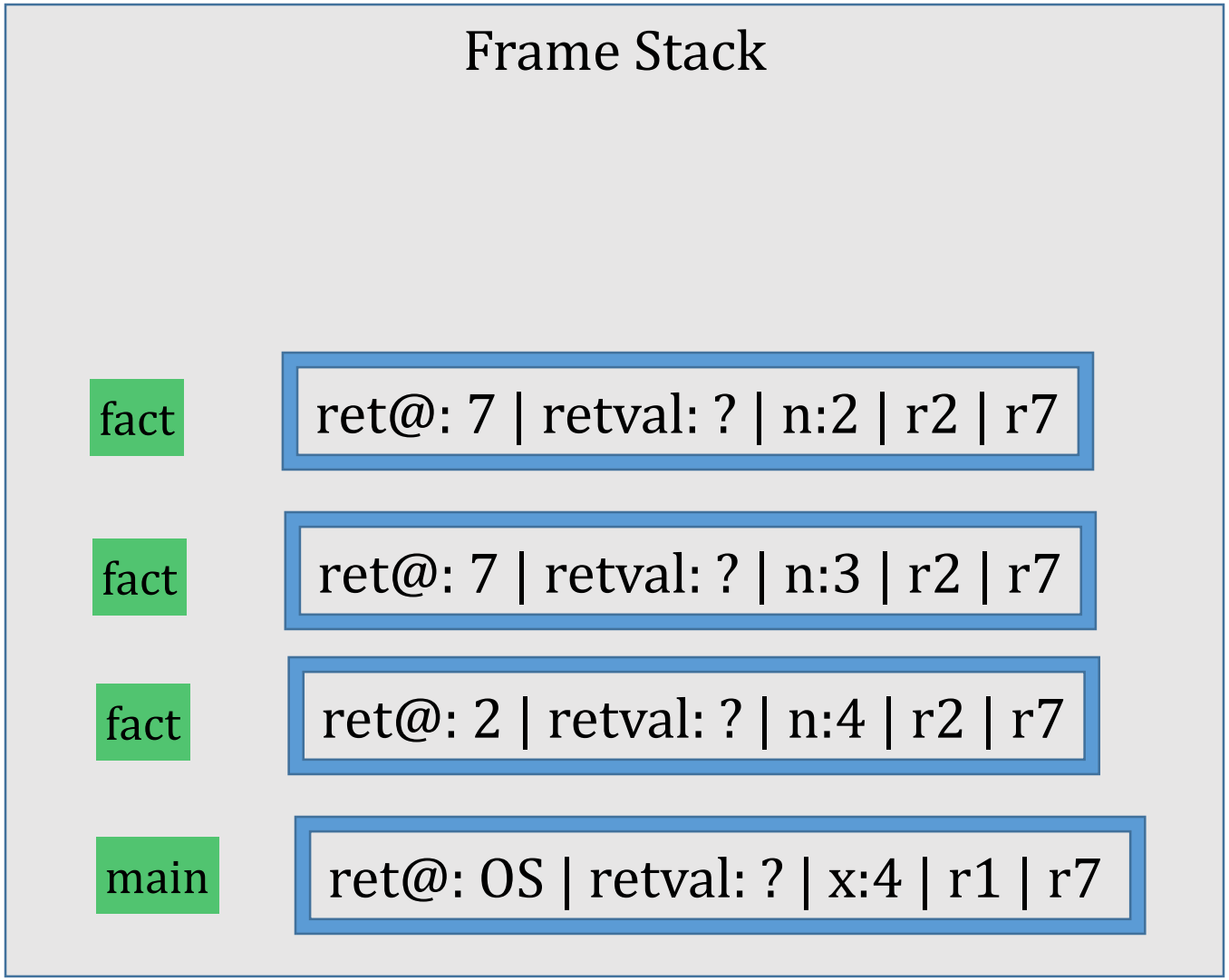
Frame Stack and Recursion

```
1: int main() { int x=4;
2:   return ( fact(x) );
3:
4: int fact(int n) {
5:     if (n<2) return 1;
6:     return  n *
7:           fact(n-1); }
```



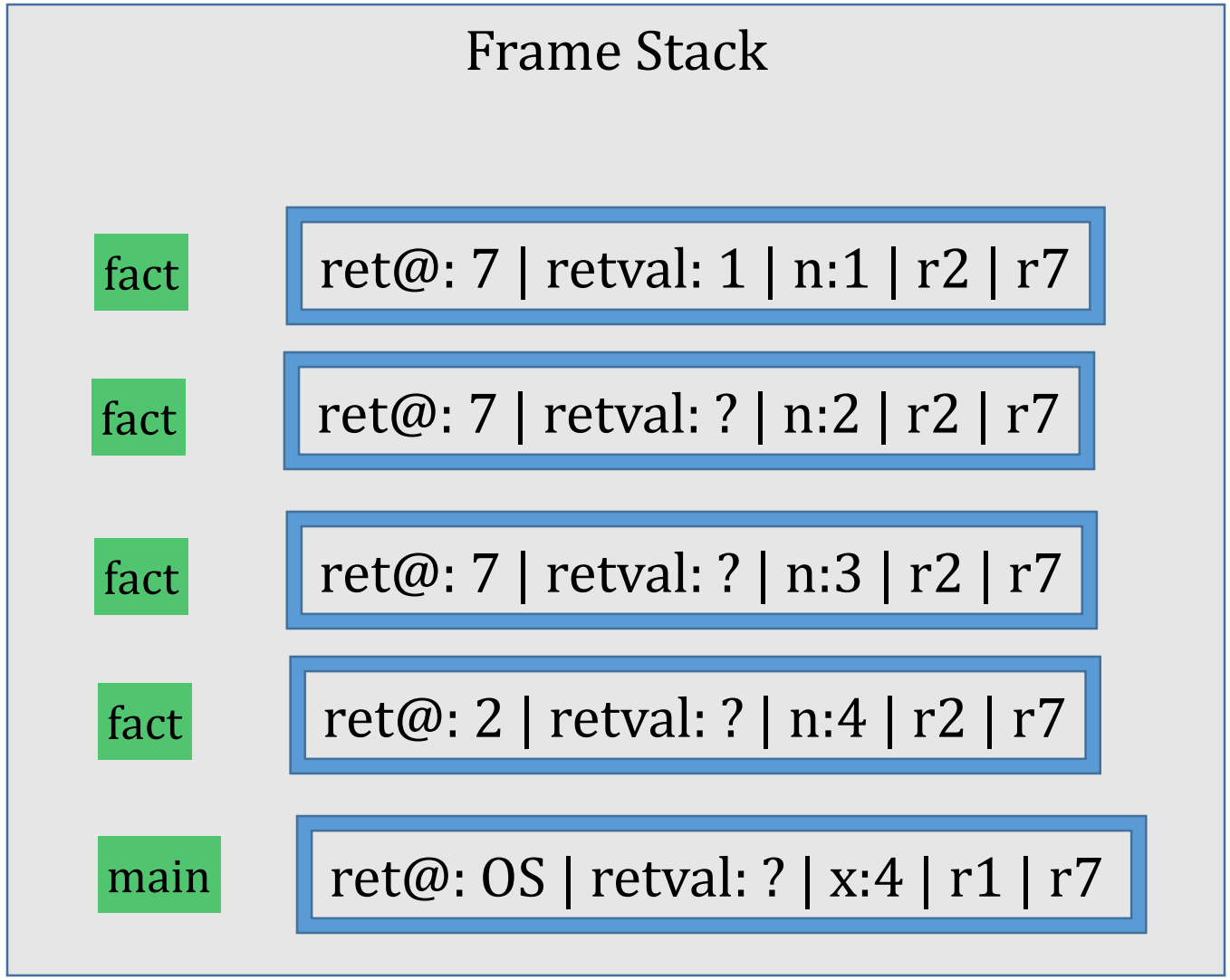
Frame Stack and Recursion

```
1: int main() { int x=4;
2:   return ( fact(x) );
3:
4: int fact(int n) {
5:     if (n<2) return 1;
6:     return  n *
7:           fact(n-1); }
```



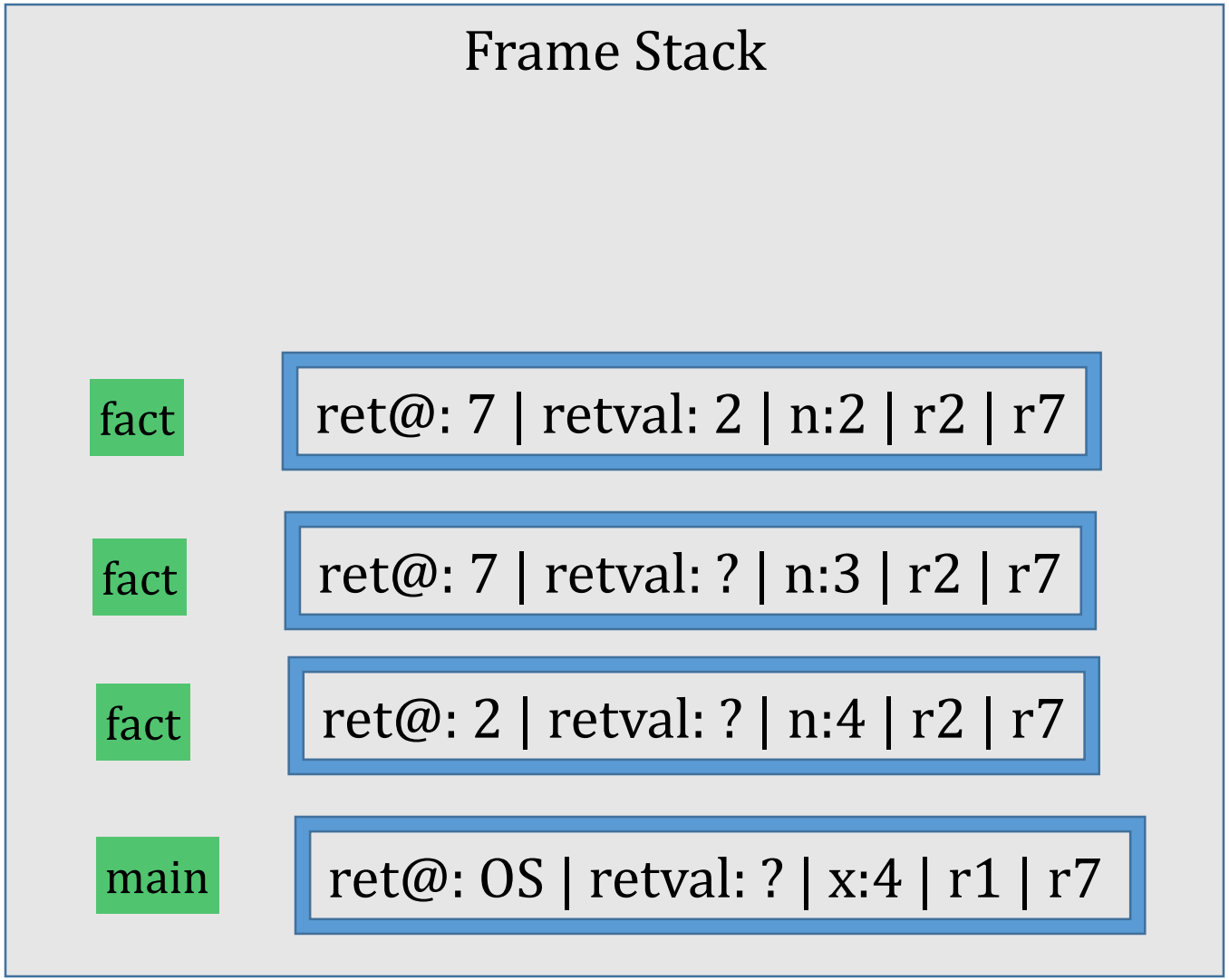
Frame Stack and Recursion

```
1: int main() { int x=4;
2:   return ( fact(x) );
3:
4: int fact(int n) {
5:     if (n<2) return 1;
6:     return  n *
7:           fact(n-1); }
```



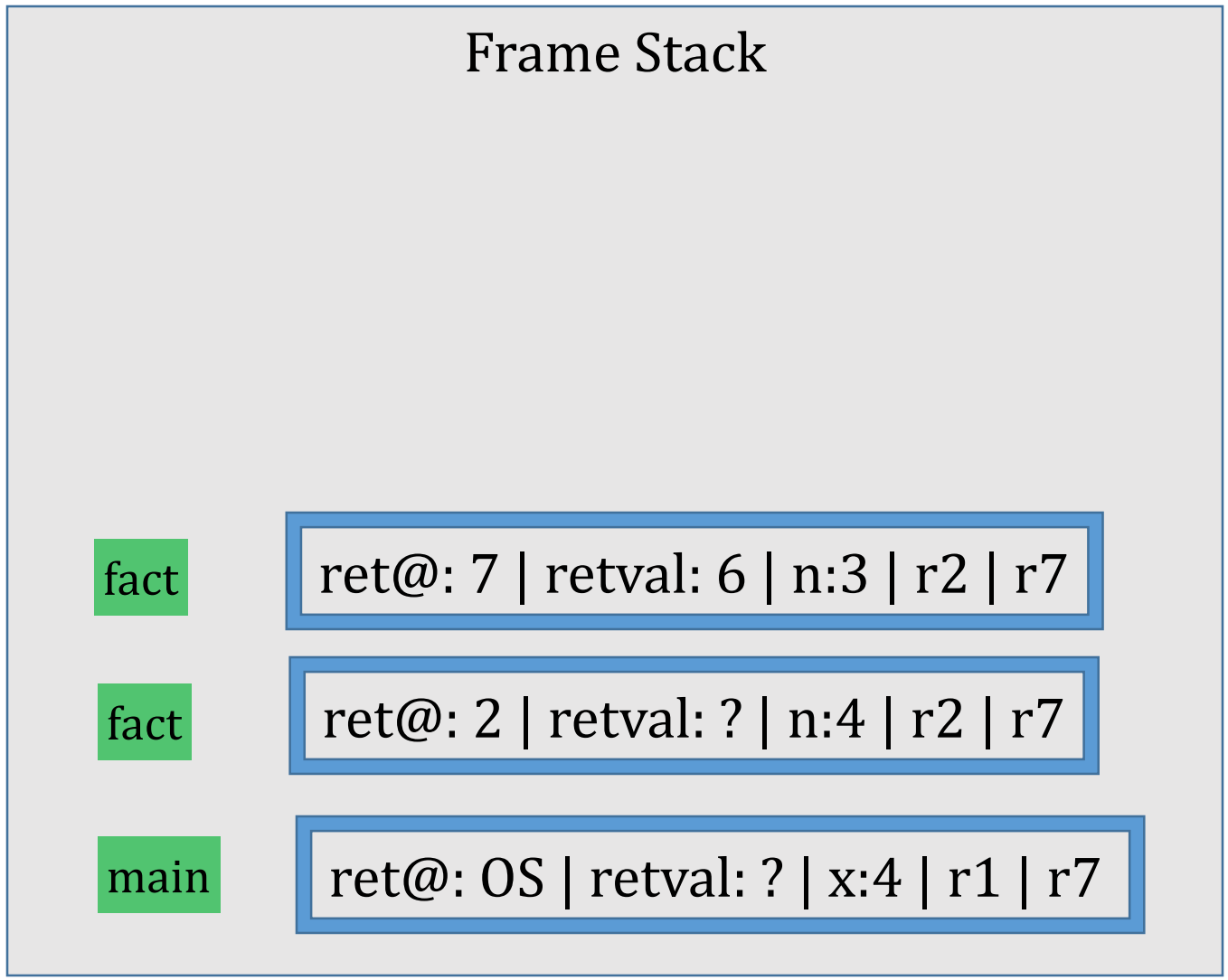
Frame Stack and Recursion

```
1: int main() { int x=4;
2:   return ( fact(x) );
3:
4: int fact(int n) {
5:      if (n<2) return 1;
6:      return  n *
7:           fact(n-1); }
```



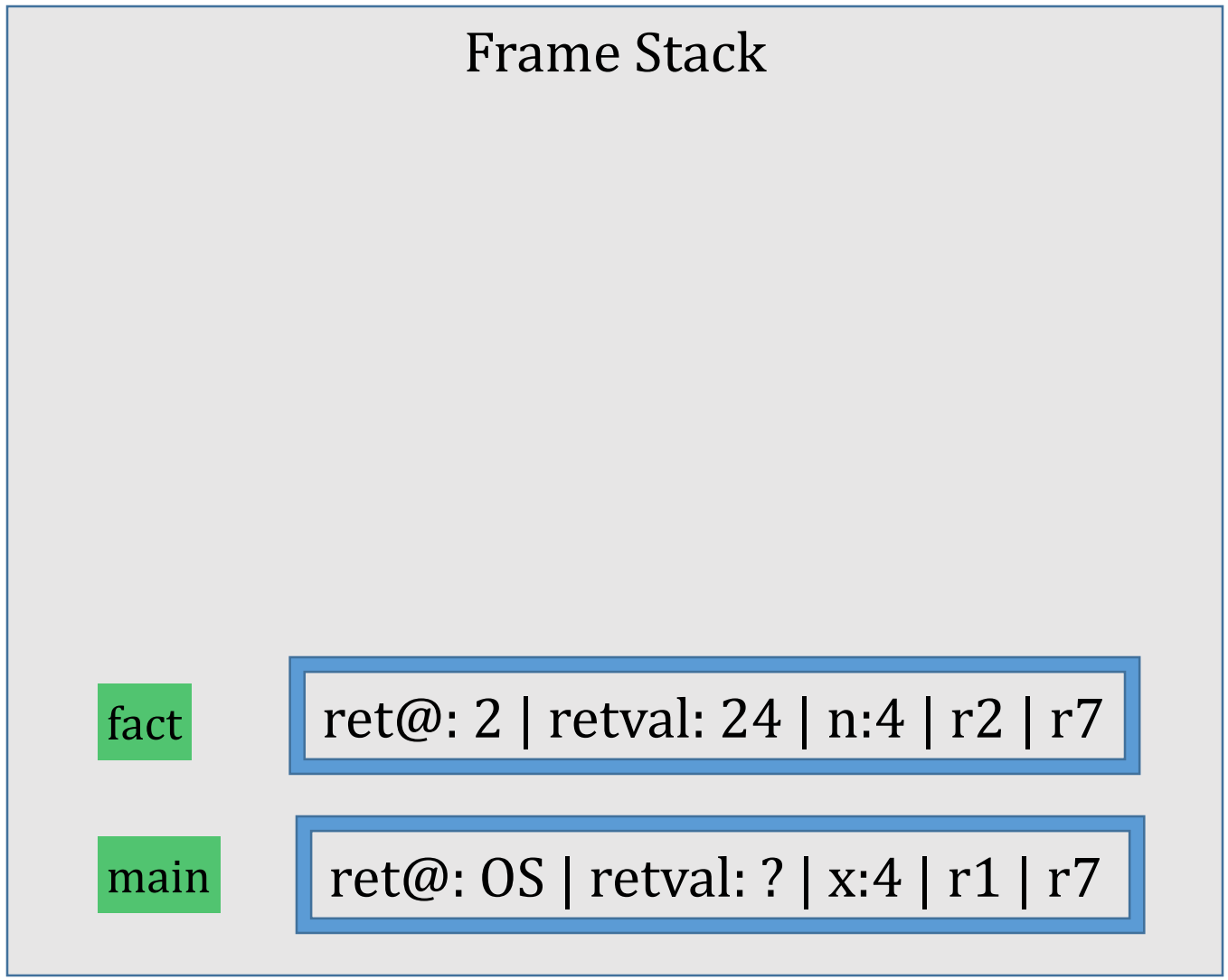
Frame Stack and Recursion

```
1: int main() { int x=4;
2:   return ( fact(x) );
3:
4: int fact(int n) {
5:      if (n<2) return 1;
6:      return  n *
7:           fact(n-1); }
```



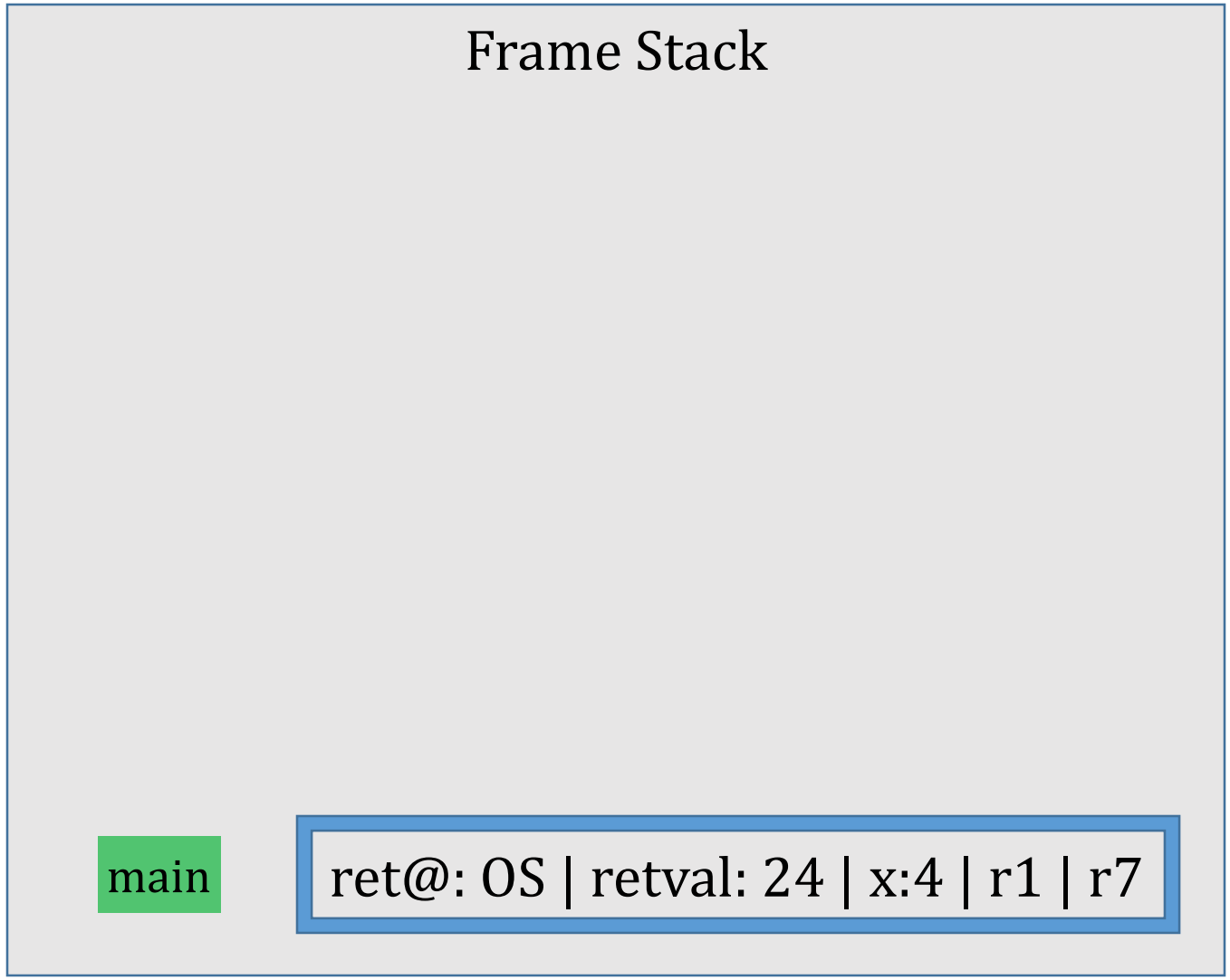
Frame Stack and Recursion

```
1: int main() { int x=4;
2:   return ( fact(x) );
3:
4: int fact(int n) {
5:      if (n<2) return 1;
6:      return  n *
7:           fact(n-1); }
```



Frame Stack and Recursion

```
1: int main() { int x=4;
2:   return ( fact(x) );
3:
4: int fact(int n) {
5:       if (n<2) return 1;
6:       return  n *
7:             fact(n-1); }
```



Variable Scope

- A variable is visible inside the block in which it is declared
  - Unless another variable with the same name is declared in a sub-block
- Variables declared outside any block are “global”
  - Visible everywhere
  - Unless another variable with the same name is declared inside a block
- Normally, the “block” is a function block

Example of variable scope

```
int vg=1; int v=2;
int main() {
    int vm=3; int v=4;
    for(v=0; v<vm; v++) {
        int vb=5; int v=6;
        printf("vg=%d, vm=%d, vb=%d, v=%d\n", vg, vm, vb, v);
    }
    printf("v=%d\n");
}
```