CS614: Advanced Compilers

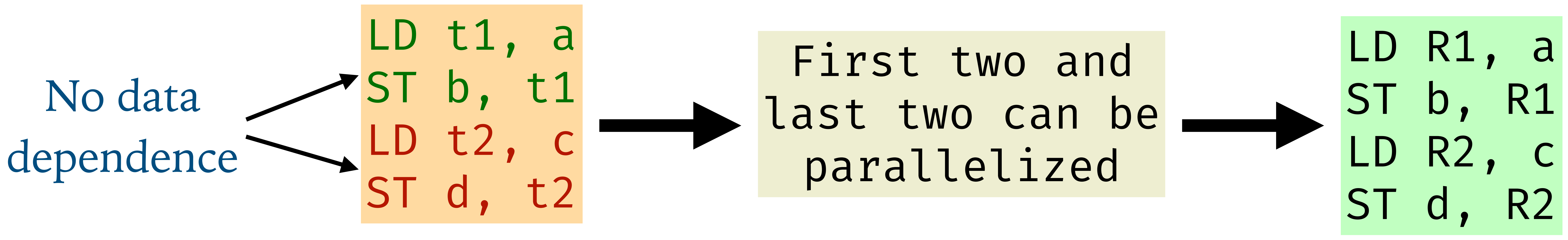
Instruction Scheduling (Cont.)

Manas Thakur
CSE, IIT Bombay

Spring 2025

Phase ordering between RA and ISc

- ISc then RA:

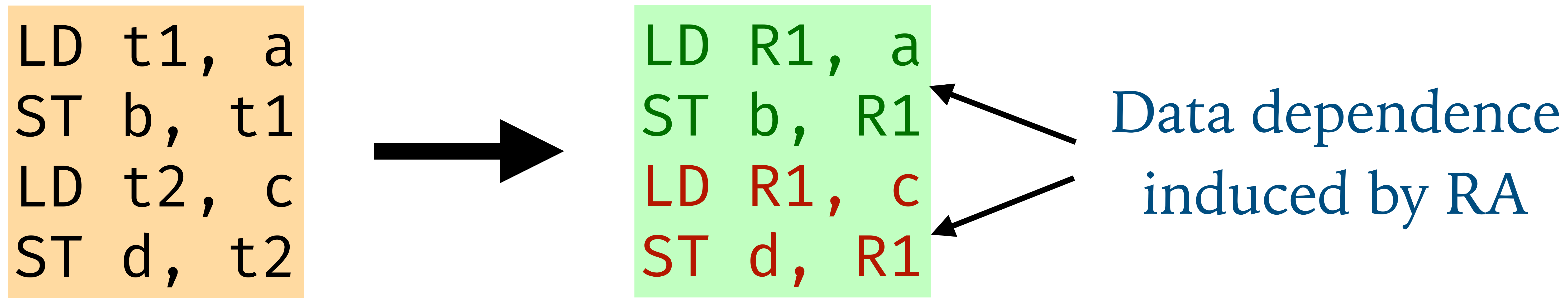


```
LD t1, a
ST b, t1
LD t2, c
ST d, t2
```

First two and last two can be parallelized

```
LD R1, a
ST b, R1
LD R2, c
ST d, R2
```

- RA then ISc:



```
LD t1, a
ST b, t1
LD t2, c
ST d, t2
```

```
LD R1, a
ST b, R1
LD R1, c
ST d, R1
```

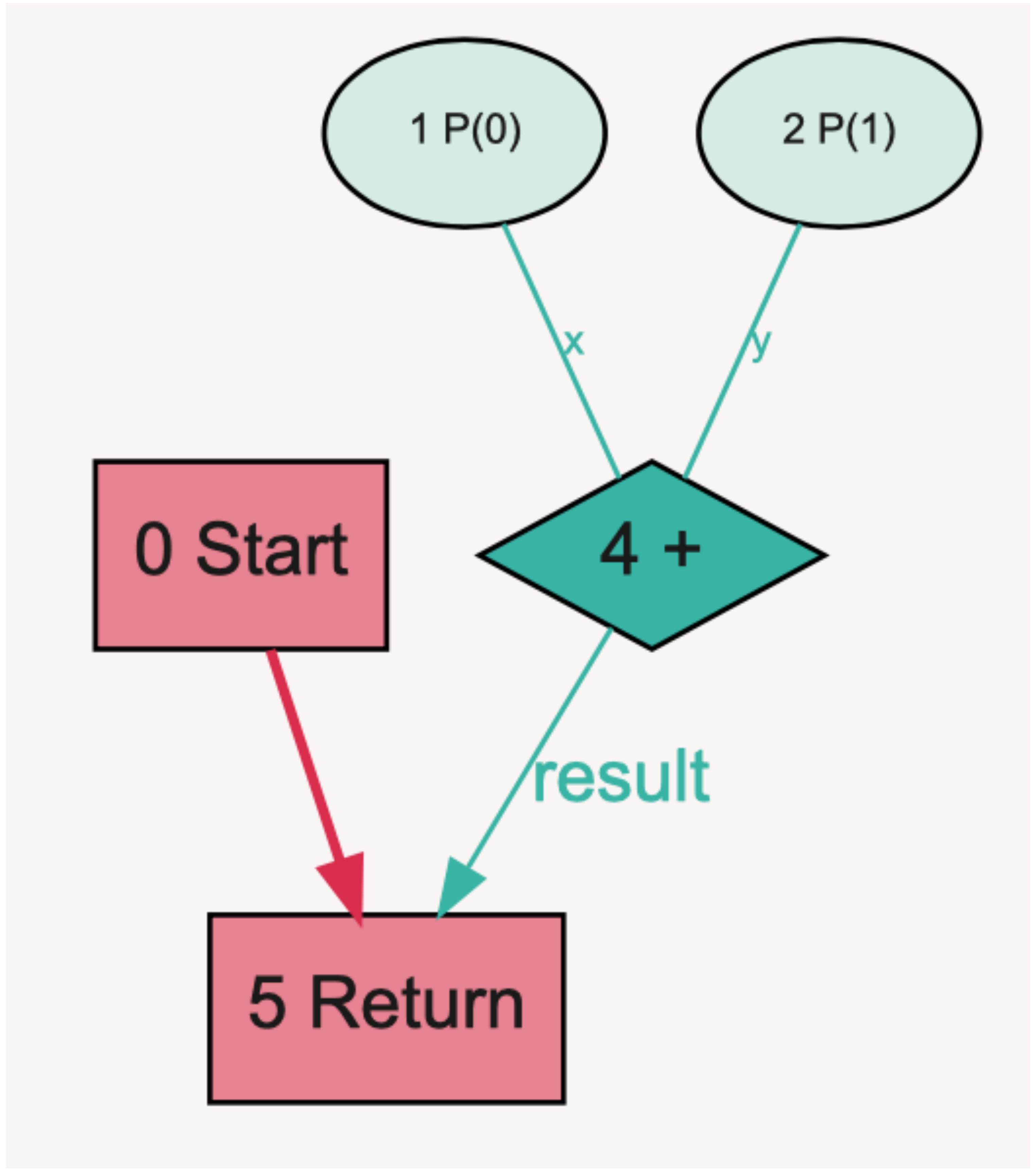
- Choose between register minimization and parallelization!

Sea of Nodes IR

- Many compilers (e.g. C2 and Graal) use a "sea of nodes" IR.
- Essentially a Program-Dependence Graph.
- Includes both:
  - Data-flow edges (data-dependence graphs, from the last class)
  - Control-flow edges (control-flow graphs, from the pre-midsem days)
- Value numbering performed to reduce redundant computations
- Instruction scheduling can be done directly
- Reordering based on dependences becomes trivial

Examples of Graal IR (1)

```
private static int exampleArithOperator(int x, int y) {
    return x + y;
}
```



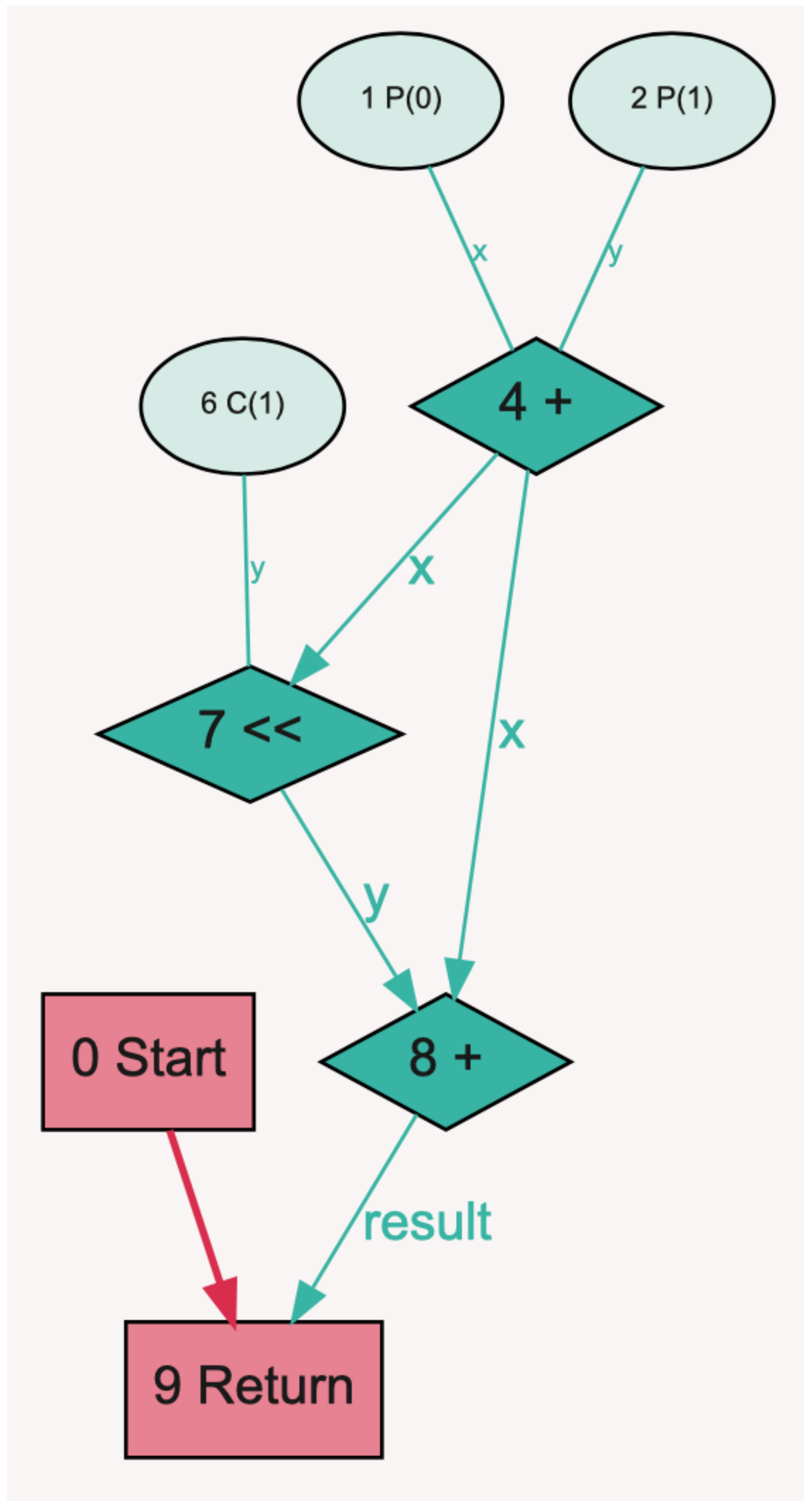
Red edges denote control flow;
green edges denote data flow

Examples of Graal IR (2)

```
private static int exampleLocalVariables(int x, int y) {
    int a = x + y;
    return a * 2 + a;
}
```

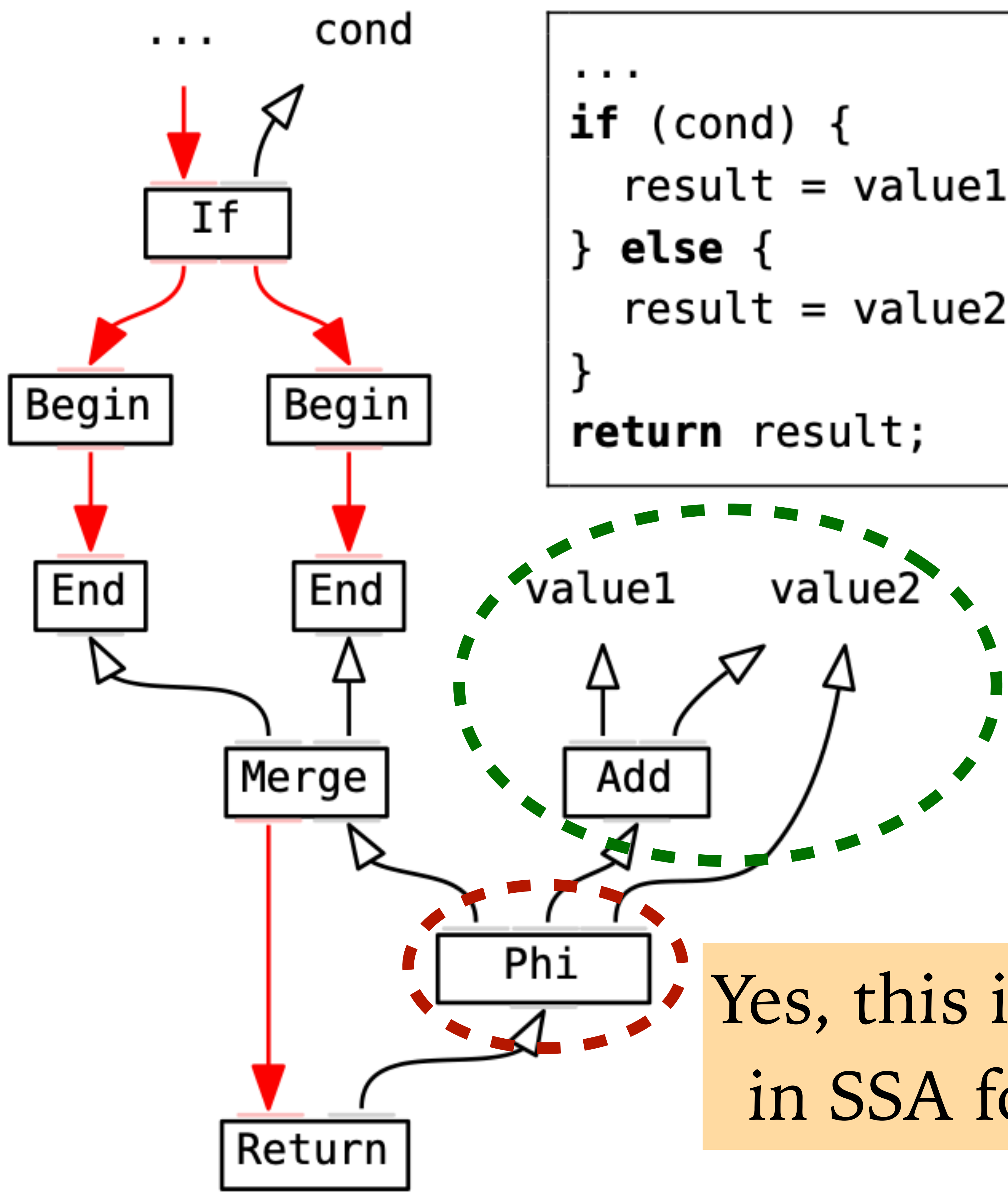
x and y are inputs to computation nodes; not program variables.

```
class AddNode extends Node {
    @Input Node x;
    @Input Node y;
}
```



Examples of Graal IR (3)

We are free to perform the addition before evaluating the condition!

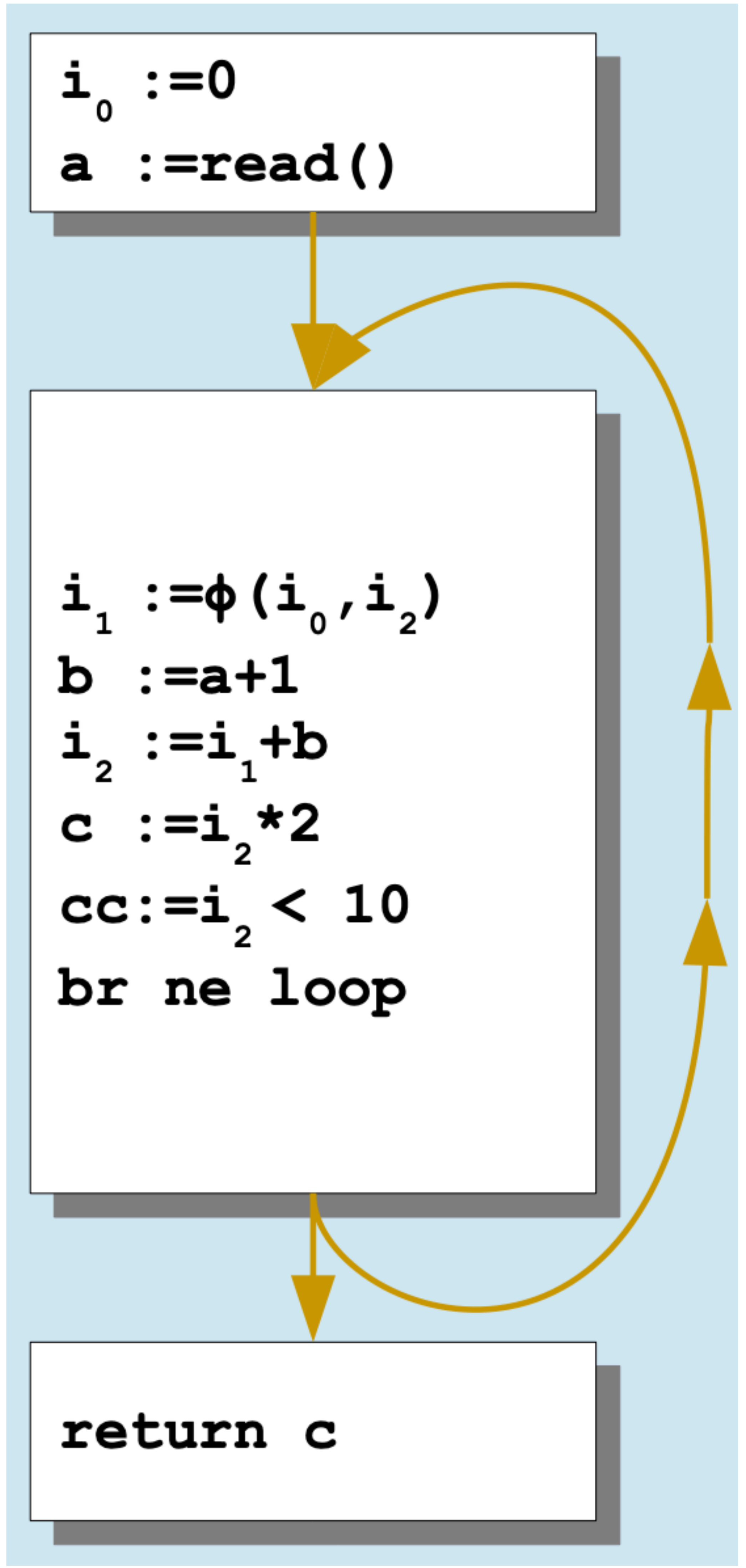


```
...
if (cond) {
  result = value1 + value2;
} else {
  result = value2;
}
return result;
```

No dependence on `cond`

Yes, this is also in SSA form!

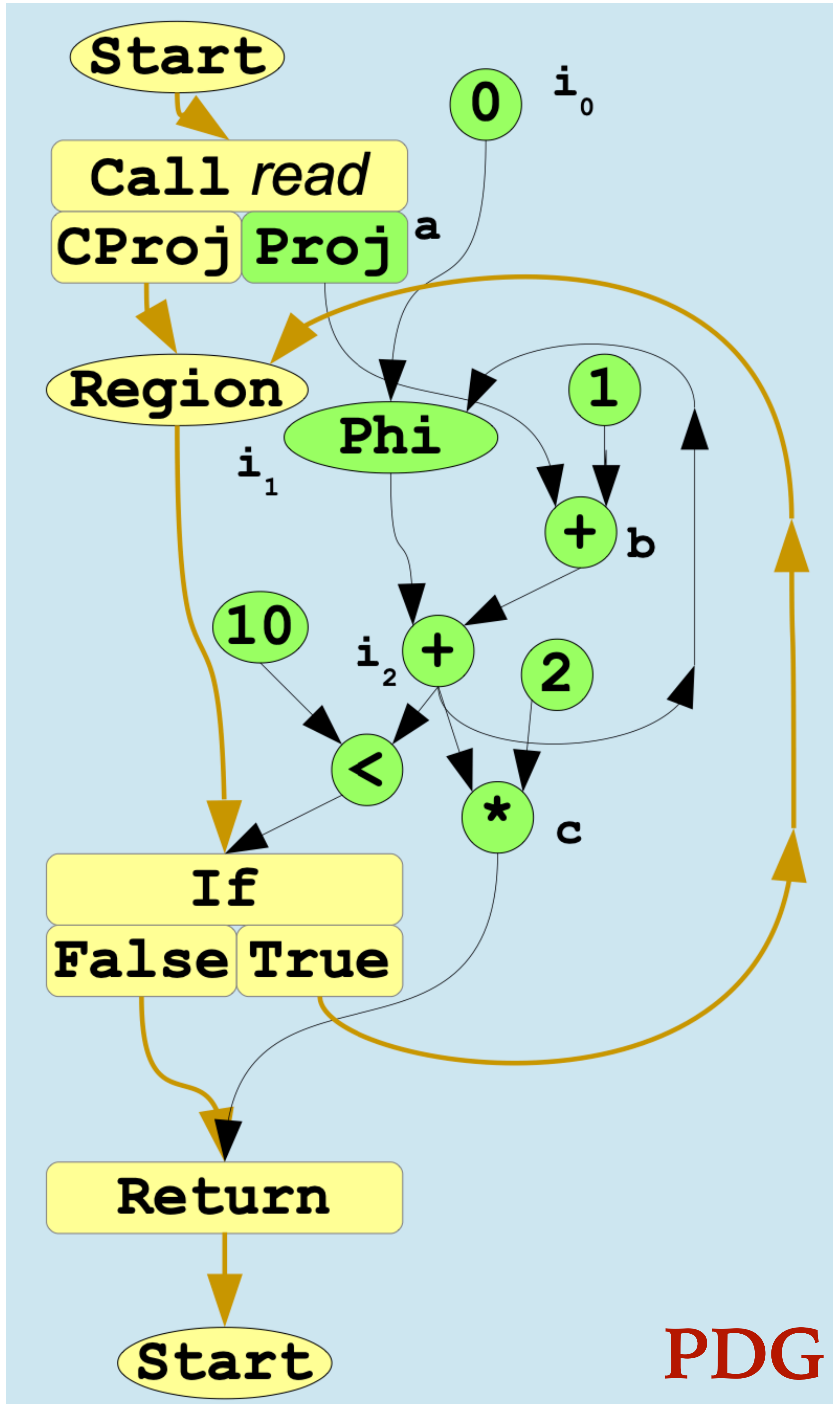
Example of C2 IR

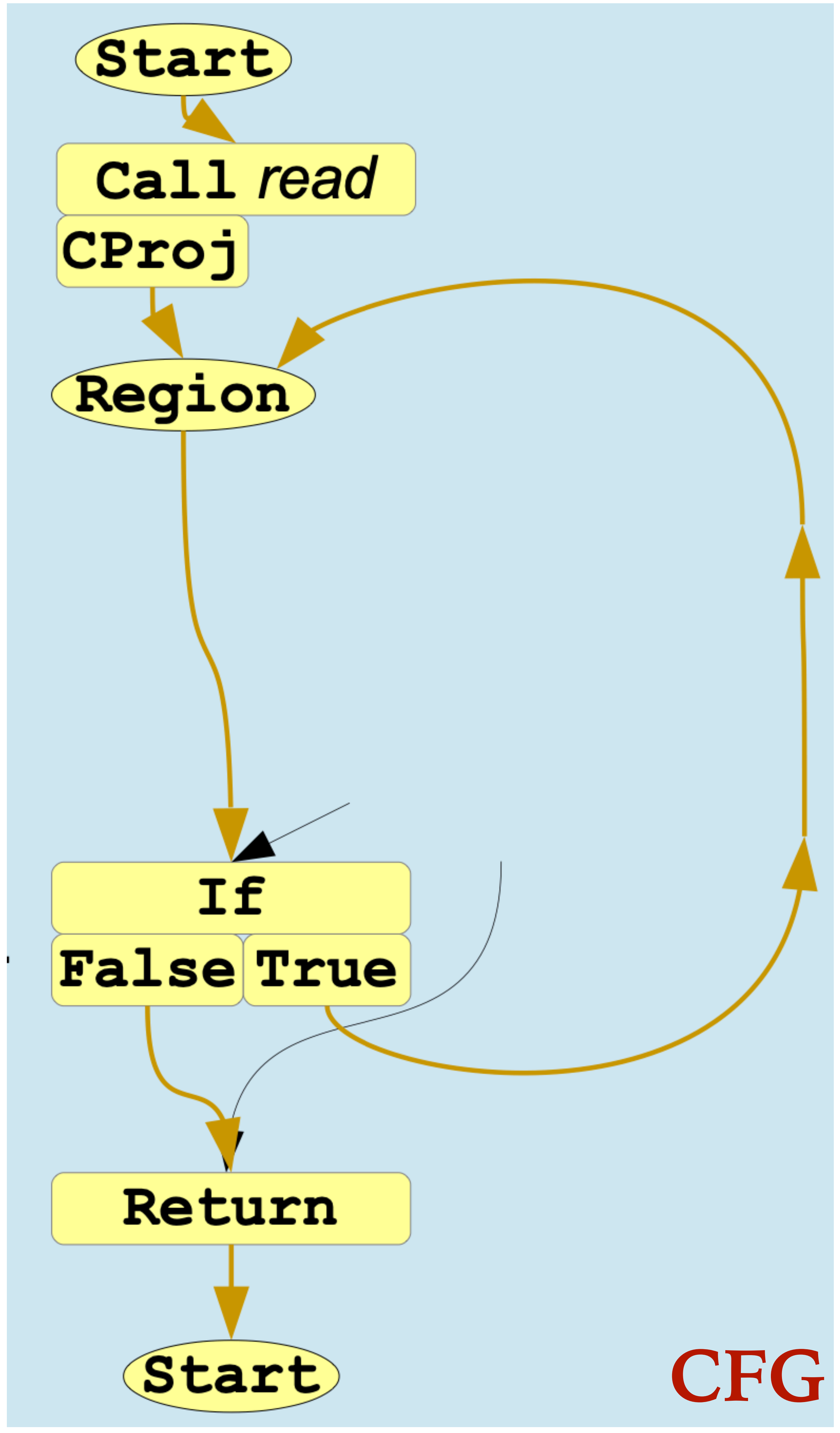


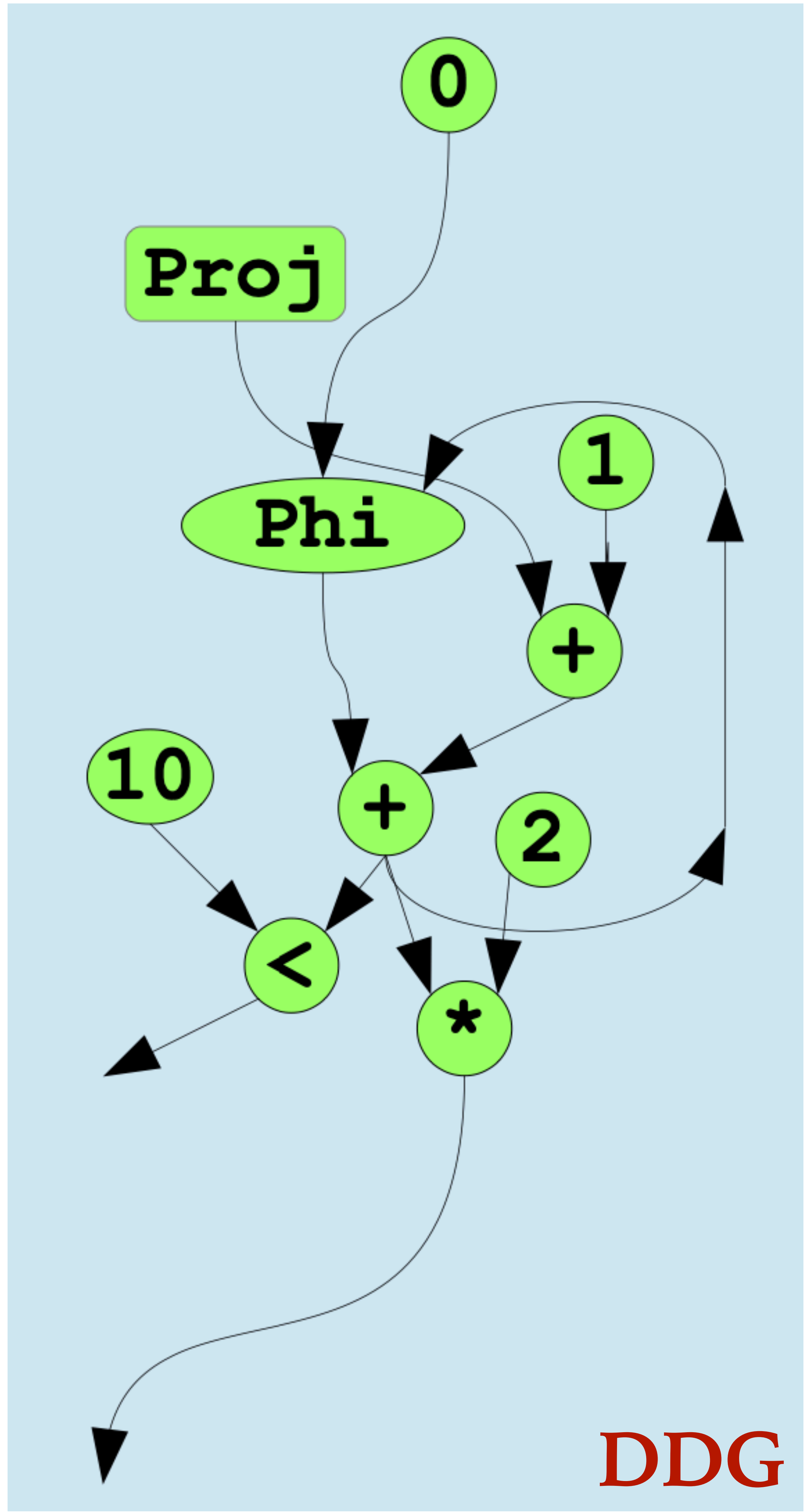
```
i₀ :=0
a  :=read()
```

```
i₁ :=ϕ(i₀,i₂)
b  :=a+1
i₂ :=i₁+b
c  :=i₂*2
cc:=i₂ < 10
br ne loop
```

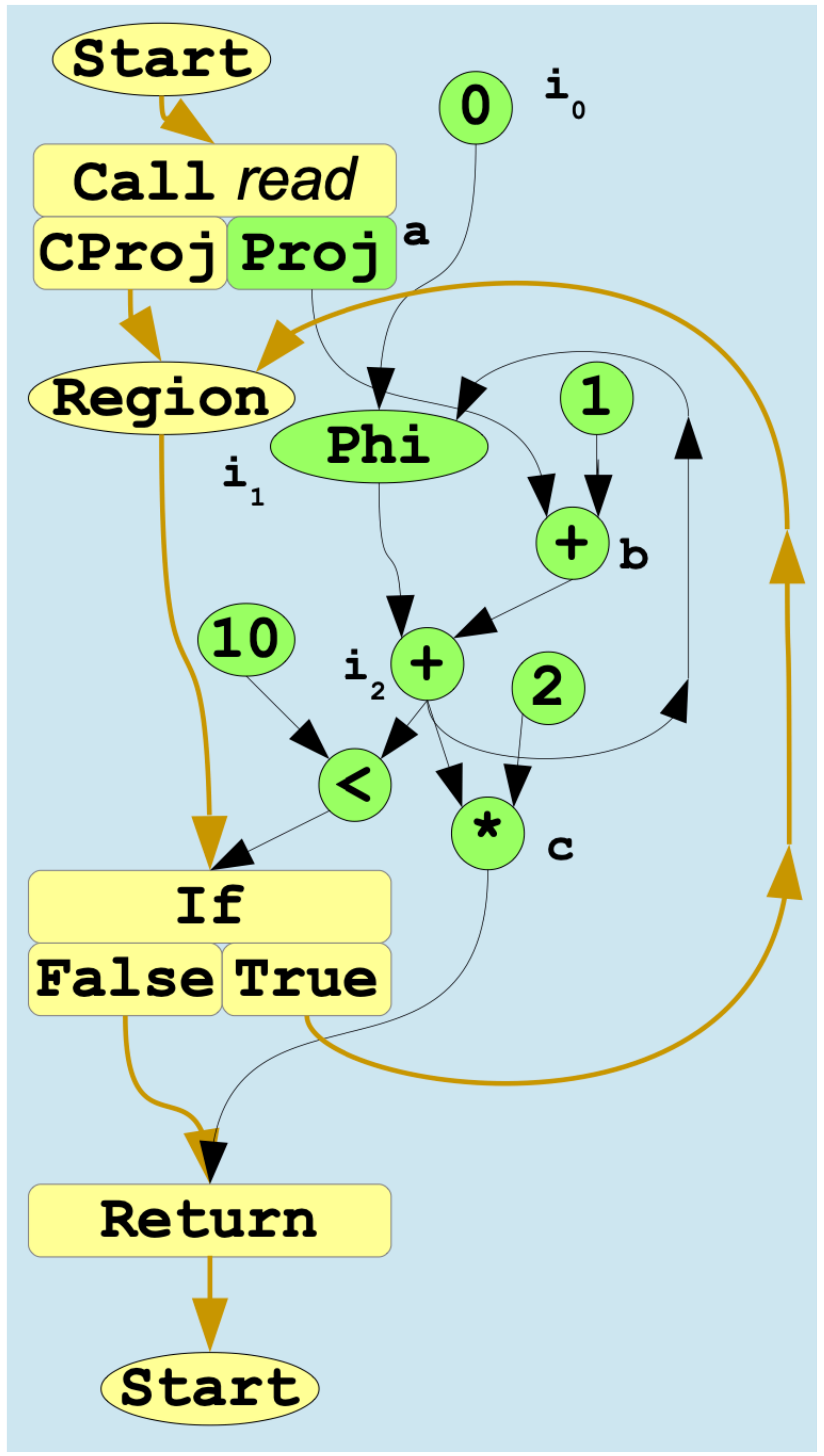
```
return c
```

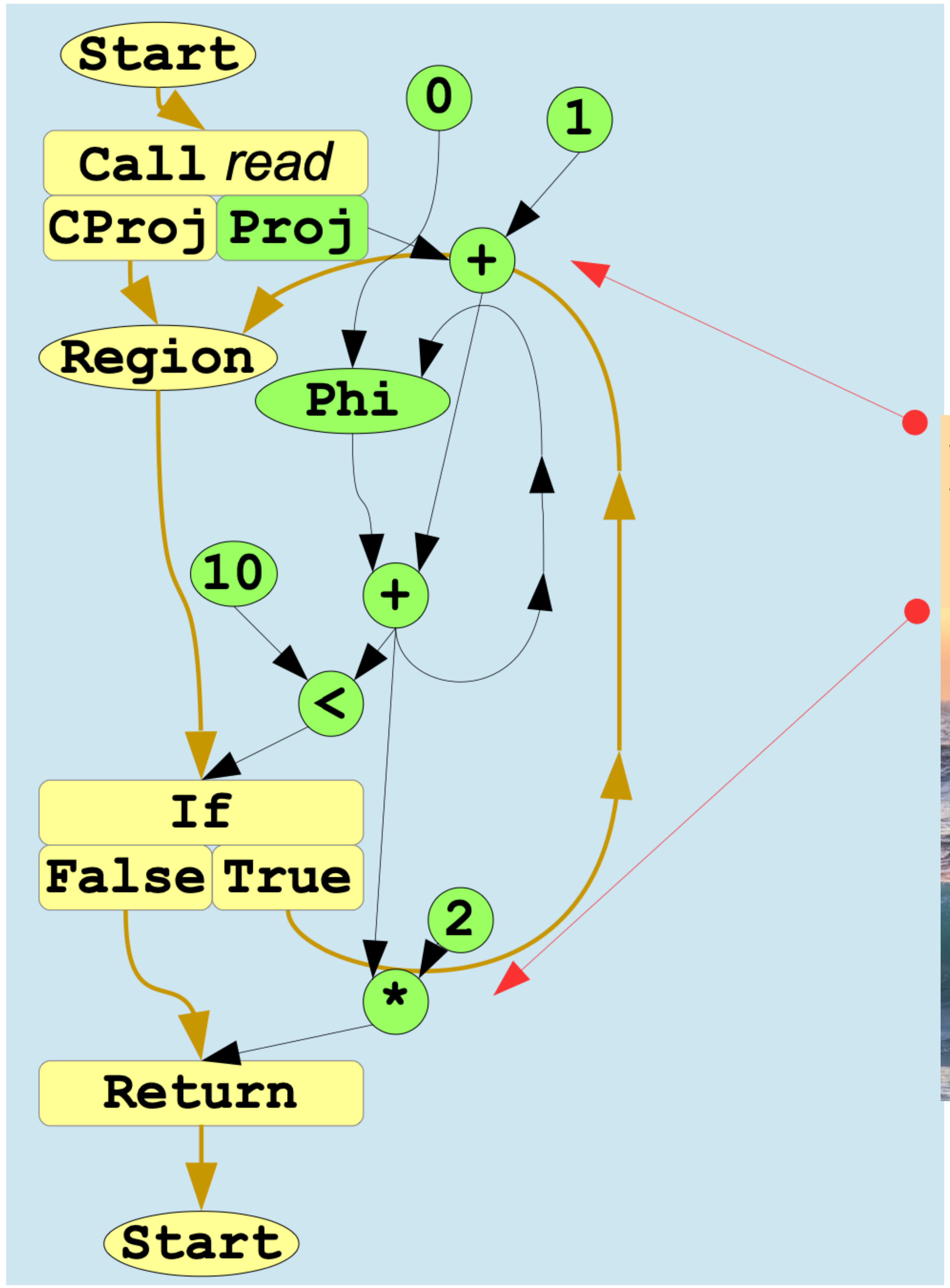






Example of C2 IR (Cont.)





Called "Ideal IR" :-)

Nodes have no real *place*; they float about.

Assignment 3

Deadline: **March 22**.

- Linear scan register allocation.
  - Requires liveness analysis <== We have already provided!
- Same grammar as A2.
- Max numbers of registers given.
- Access APIs to get liveness information.
- Form live intervals based on earliest and latest liveness points.
- Assign registers to variables and print code using register names (declared at the top).
- Spill variables that do not get registers (APIs provided).