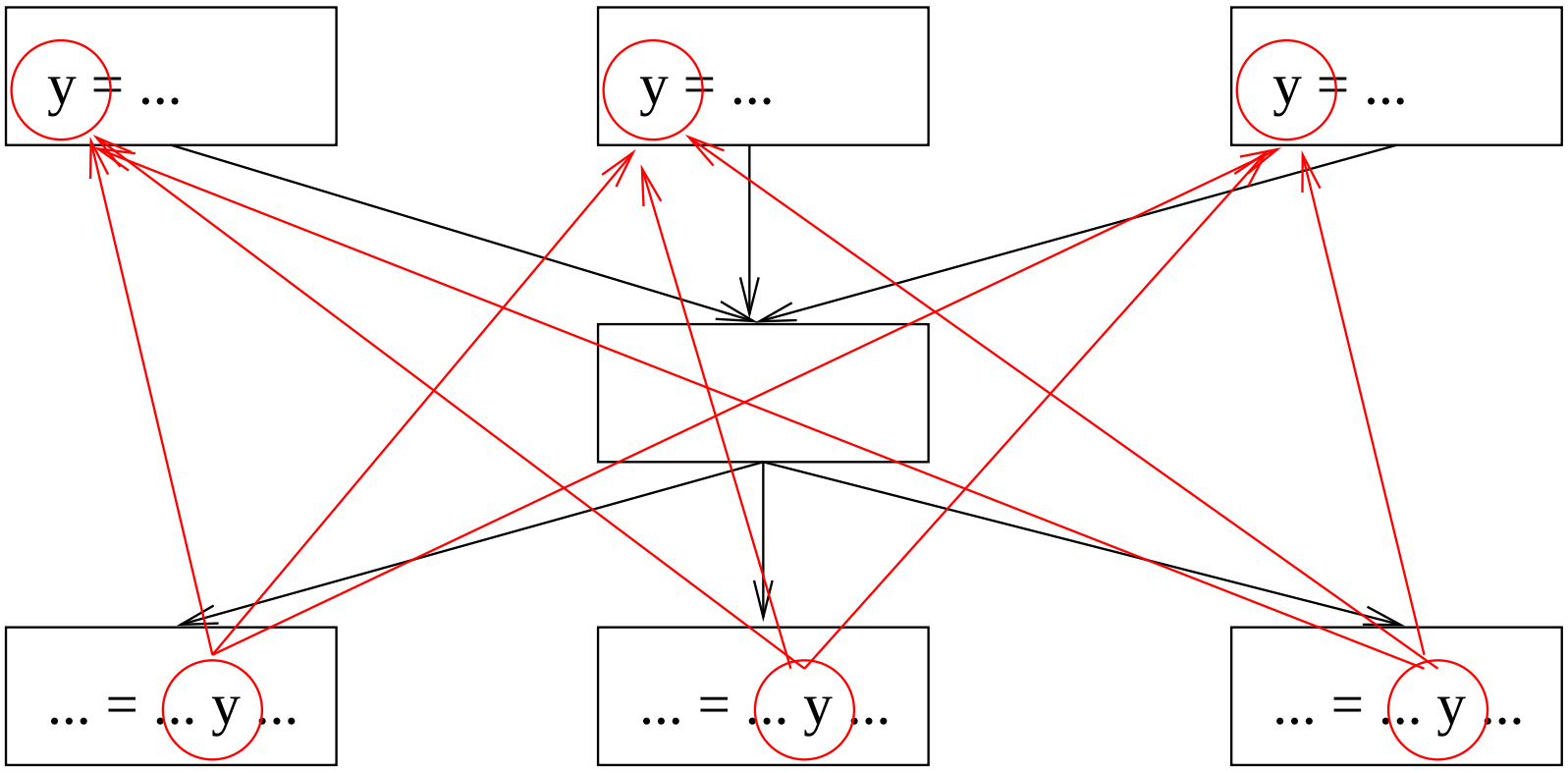
Static Single Assignment (SSA) Motivation



Static Single Assignment

Idea: Make each variable have only one definition.

```
a = 2;
b = a + 1;
a = 3;
b = a + 1;
```

```
a1 = 2;
b1 = a1 + 1;
a2 = 3;
b2 = a2 + 1;
```

Static Single Assignment

Idea: Make each variable have only one definition.

```
a = 2;
b = a + 1;
a = 3;
b = a + 1;
```

```
a1 = 2;
b1 = a1 + 1;
a2 = 3;
b2 = a2 + 1;
```

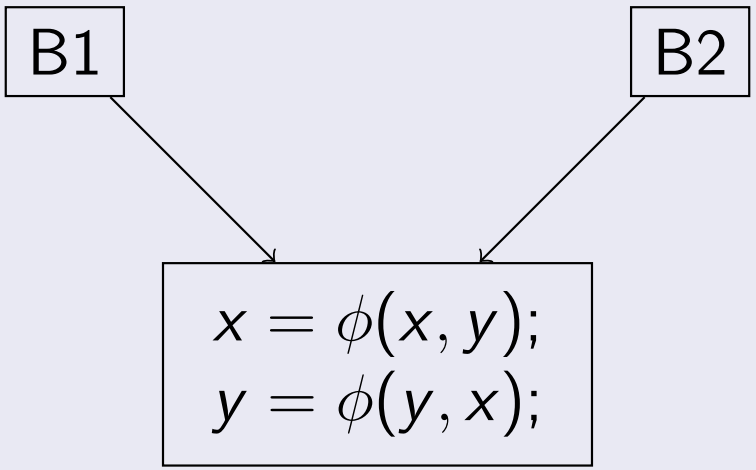
```
if() {
 a = 2;
} else {
 a = 3;
}
b = a + 1;
```

```
if() {
 a1 = 2;
} else {
 a2 = 3;
}
a3 = φ(a1, a2);
b1 = a3 + 1;
```

What does ϕ really mean?

- ϕ with n arguments only makes sense at the beginning of a basic block with n incoming edges. Incoming control-flow edges are implicit parameters of ϕ "function".
- All ϕ's in a BB are "executed" simultaneously, NOT sequentially.

Example



- If coming from B1, keep x and y unchanged.
- If coming from B2, swap x and y.

SSA Algorithm 1 ("maximal", "really crude")

1. At every basic block with multiple predecessors, insert ϕ node for every variable.
2. Do reaching definitions analysis. Every use will have one reaching def. (Why?)
3. Number each def, and assign the same number to the uses it reaches.

SSA Algorithm 2 [Aycock & Horspool]

1. Do SSA Algorithm 1.
2. Remove ϕ functions of the following forms:
   - $x_i = \phi(x_i, x_i, \ldots, x_i)$
   - $x_i = \phi(x_K, x_K, \ldots, x_K)$, where each K is either i or j; replace all uses of x_i with x_j.

   until there are no such ϕ functions left.

SSA Algorithm 3 (Dominance Frontier)

Dominance frontier criterion

Whenever basic block x defines variable a, every node z in the dominance frontier of x needs a ϕ function for a.

Algorithm $\mathrm{SSA3}()$:

1: **while** $\exists x, z, a$ satisfying the criterion and z has no ϕ for a **do**
2: add ϕ for a to z

Note: adding ϕ adds a definition of a

SSA Algorithm 4 ("pruned SSA")

Dominance frontier criterion with liveness

Whenever basic block x defines variable a, every node z in the dominance frontier of x such that a is live before z needs a ϕ function for a.

Algorithm $\mathrm{SSA4}()$:

1: perform live variables analysis
2: **while** $\exists x, z, a$ satisfying the criterion and z has no ϕ for a **do**
3: add ϕ for a to z

SSA Algorithm 5 (linear time)

DJ graph

Dominator tree (D edges) augmented with J edges – CFG edges that are not already D edges.

Details: Sreedhar, Gao. A Linear Time Algorithm for Placing ϕ-Nodes. POPL 1995.

Useful Properties of SSA Form

Property

Every variable has exactly one definition.

Useful Properties of SSA Form

Property

Every variable has exactly one definition.

Property

Every use of a variable is dominated by its (unique) definition.

Useful Properties of SSA Form

Property

Every variable has exactly one definition.

Property

Every use of a variable is dominated by its (unique) definition.
Corollaries:

- If v is live at statement s, then def of v dominates s.

Useful Properties of SSA Form

Property

Every variable has exactly one definition.

Property

Every use of a variable is dominated by its (unique) definition.
Corollaries:

- If v is live at statement s, then def of v dominates s.
- If v, w are live at the same statement, then def of v dominates def of w or vice versa.

Applications of SSA

Dead code elimination

`a = b + c` is dead code iff there are no uses of variable `a`.

Applications of SSA

Dead code elimination

`a = b + c` is dead code iff there are no uses of variable `a`.

Constant propagation

Whenever the single assignment to `x` is a constant c, all uses of `x` can be replaced with c.

Applications of SSA

Dead code elimination

`a = b + c` is dead code iff there are no uses of variable `a`.

Constant propagation

Whenever the single assignment to `x` is a constant c, all uses of `x` can be replaced with c.

Copy propagation

`x = y` can be deleted and uses of `x` replaced with `y`.

Converting out of SSA form

- Basic idea: insert assignments at end of predecessors.
- Difficulties:
  - ϕ nodes must be "executed" simultaneously.
  - Predecessor may have multiple successors.
    - insert nodes on edge from predecessor ("edge splitting")