8. Static Single Assignment Form

Marcus Denker

UNIVERSITÄT BERN

Roadmap

- Static Single Assignment Form (SSA)
- Converting to SSA Form
- Examples
- Transforming out of SSA

Static Single Assignment Form

- Goal: simplify procedure-global optimizations

- *Definition:*

  Program is in SSA form if every variable is only assigned once

Why *Static*?

- Why Static?
  - *We only look at the static program*
  - *One assignment per variable in the program*
- At runtime variables are assigned multiple times!

Example: Sequence

> Easy to do for sequential programs:

Original

```
a := b + c
b := c + 1
d := b + c
a := a + 1
e := a + b
```

SSA

$$
\begin{aligned}
a_1 &:= b_1 + c_1 \\
b_2 &:= c_1 + 1 \\
d_1 &:= b_2 + c_1 \\
a_2 &:= a_1 + 1 \\
e_1 &:= a_2 + b_2
\end{aligned}
$$

Example: Condition

> Conditions: what to do on control-flow merge?

Original

```
if B then
  a := b
else
  a := c
end
  … a …
```

SSA

```
if B then
  a1 := b
else
  a2 := c
End

  … a? …
```

Solution: Φ-Function

- Conditions: what to do on control-flow merge?

Original

```
if B then
   a := b
else
   a := c
end
   … a …
```

SSA

$$
\begin{array}{l}
\texttt{if B then} \\
\quad a_1 := b \\
\texttt{else} \\
\quad a_2 := c \\
\texttt{End} \\
a_3 := \Phi(a_1, a_2) \\
\quad \ldots\ a_3\ \ldots
\end{array}
$$

The Φ-Function

- Φ-functions are always at the beginning of a basic block
- Select between values depending on control-flow
- $a_1 := \Phi(a_1 \ldots a_k)$: the block has k preceding blocks

PHI-functions are all evaluated simultaneously.

SSA and CFG

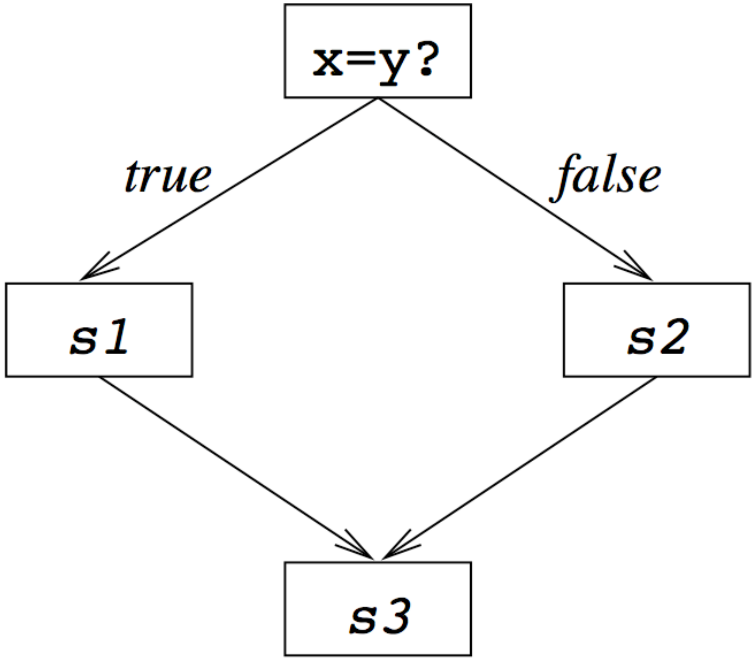
> SSA is normally done for control-flow graphs (CFG)

> Basic blocks are in 3-address form

Repeat: Control flow graph

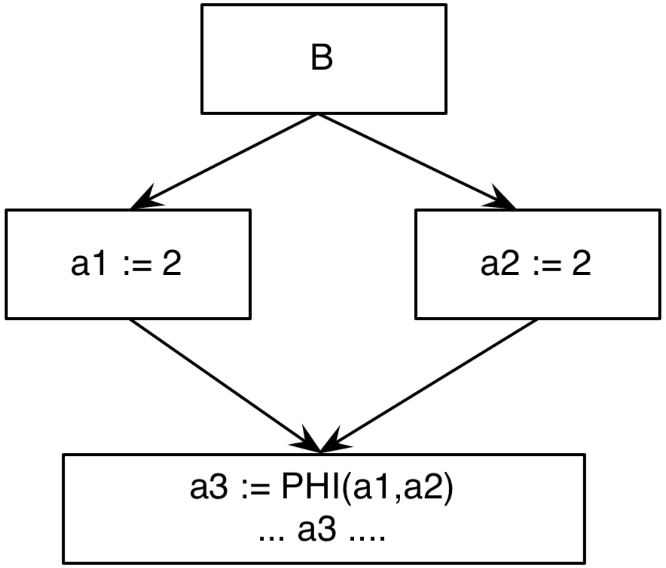
> A CFG models *transfer of control* in a program
  — nodes are *basic blocks* (straight-line blocks of code)
  — edges represent *control flow* (loops, if/else, goto …)

```
if x = y then
   S1
else
   S2
end
S3
```

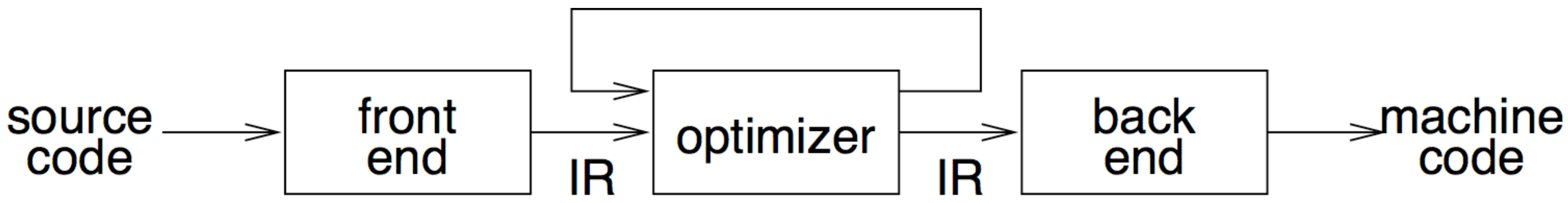


SSA: a Simple Example

```
if B then
   a1 := 1
else
   a2 := 2
End
a3 := PHI(a1,a2)
   … a3 …
```

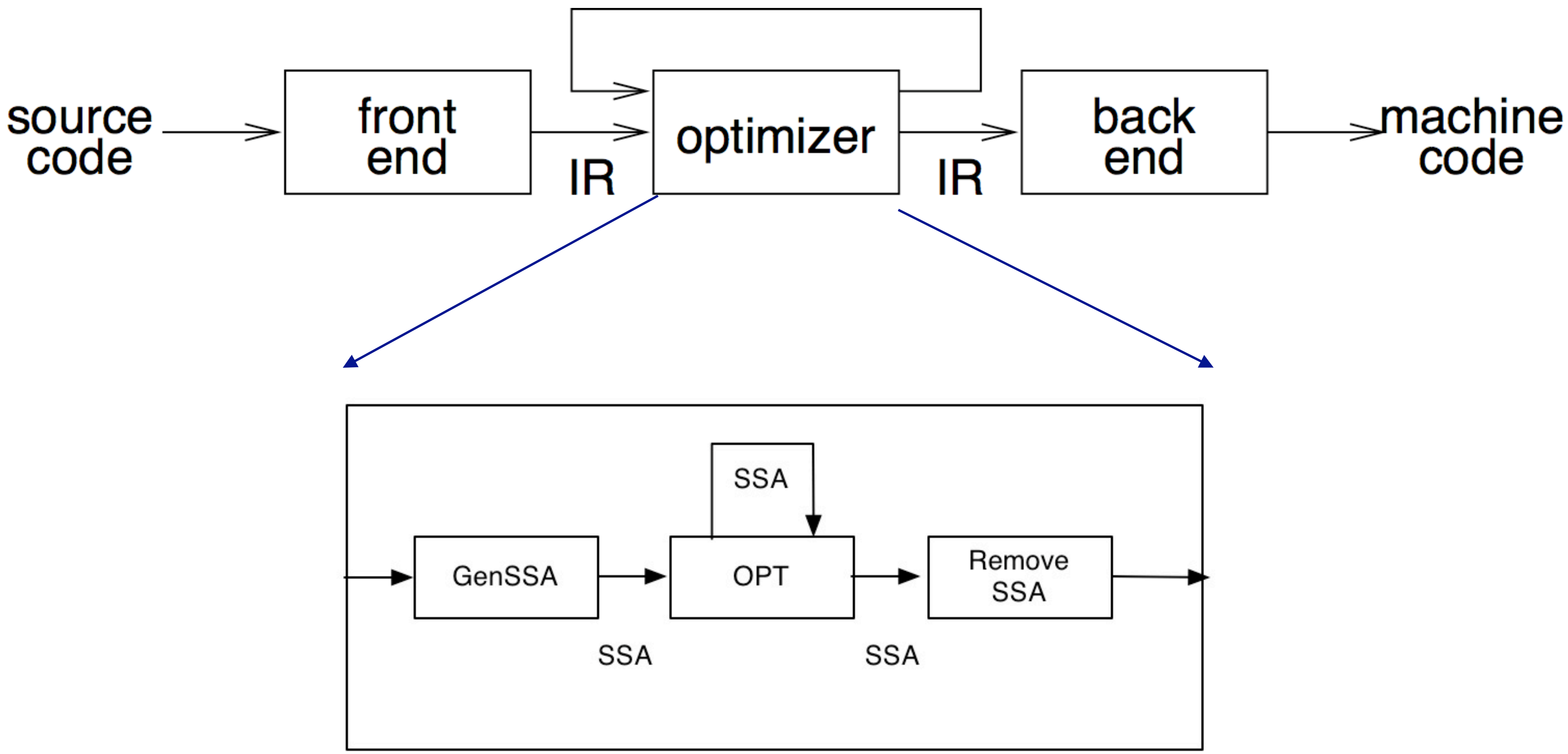


Repeat: IR



- front end produces IR
- optimizer transforms IR to more efficient program
- back end transform IR to target code

SSA as IR


source
code
front
end
IR
optimizer
IR
back
end
machine
code
SSA
GenSSA
OPT
Remove
SSA
SSA
SSA

Transforming to SSA

- Problem: Performance / Memory
  - *Minimize number of inserted* Φ*-functions*
  - *Do not spend to much time*
- Many relatively complex algorithms
  - *We do not go too much into details*
  - *See literature!*

Minimal SSA

- Two steps:
  - Place Φ-functions
  - Rename Variables

- Where to place Φ-functions?

- We want minimal amount of needed Φ
  - *Save memory*
  - *Algorithms will work faster*

Path Convergence Criterion

> There should be a Φ for a at node Z if:

1. *There is a block X containing a definition of a.*
2. *There is a block Y (Y != X) containing a definition of a.*
3. *There is a nonempty path P_{xz} of edges from X to Z.*
4. *There is a nonempty path P_{yz} of edges from Y to Z.*
5. *Path P_{xz} and P_{yz} do not have any nodes in common other than Z*
6. *The node Z does not appear within both P_{xz} and P_{yz} prior to the end (although it may appear in one or the other)*

Iterated Path-Convergence

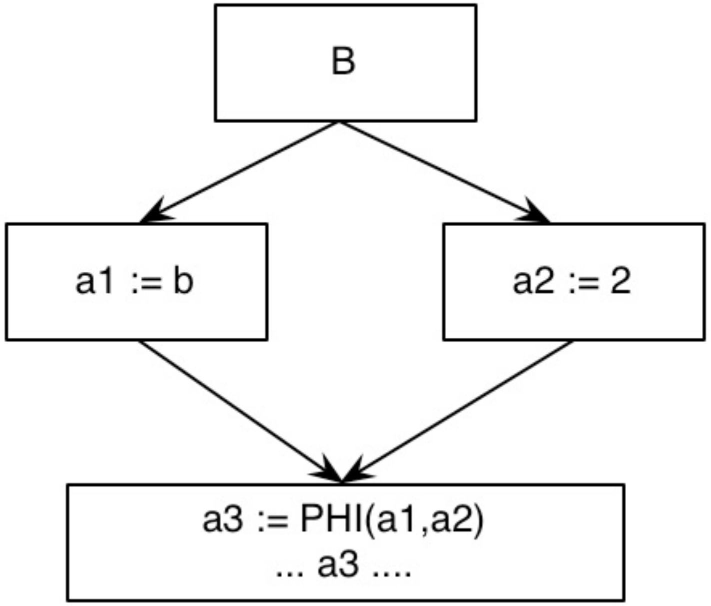
> Inserted Φ is itself a definition!

```
While there are nodes X,Y,Z satisfying conditions 1-5
   and Z does not contain a phi-function for a
 do
   insert PHI at node Z.
```

A bit slow, other algorithms used in practice

Example (Simple)

1. *block X containing a definition of a*
2. *block Y (Y != X) containing a definition of a.*
3. *path P_{xz} of edges from X to Z.*
4. *path P_{yz} of edges from Y to Z.*



5. *Path P_{xz} and P_{yz} do not have any nodes in common other than Z*
6. *The node Z does not appear within both P_{xz} and P_{yz} prior to the end*

Dominance Property of SSA

> Dominance: node *D* dominates node *N* if every path from the start node to *N* goes through *D*.

("strictly dominates": D!=N)

Dominance Property of SSA:

1. If x is used in a Phi-function in block N, then the definition of x dominates every predecessor of N.
2. If x is used in a non-Phi statement in N, then the definition of x dominates N

"Definition dominates use"

Dominance and SSA Creation

> Dominance can be used to efficiently build SSA

> Φ-Functions are placed in all basic blocks of the *Dominance Frontier*.

> **Dominance Frontier:** the set of all nodes N such that D dominates an immediate predecessor of N but does not strictly dominate N.

Dominance and SSA Creation

DF(D) = the set of all nodes N such that D dominates an immediate predecessor of N but does not strictly dominate N.

Intuition: Nodes at the border of a region of dominance

Dominance and SSA Creation

DF(D) = the set of all nodes N such that D dominates an immediate predecessor of N but does not strictly dominate N.

?

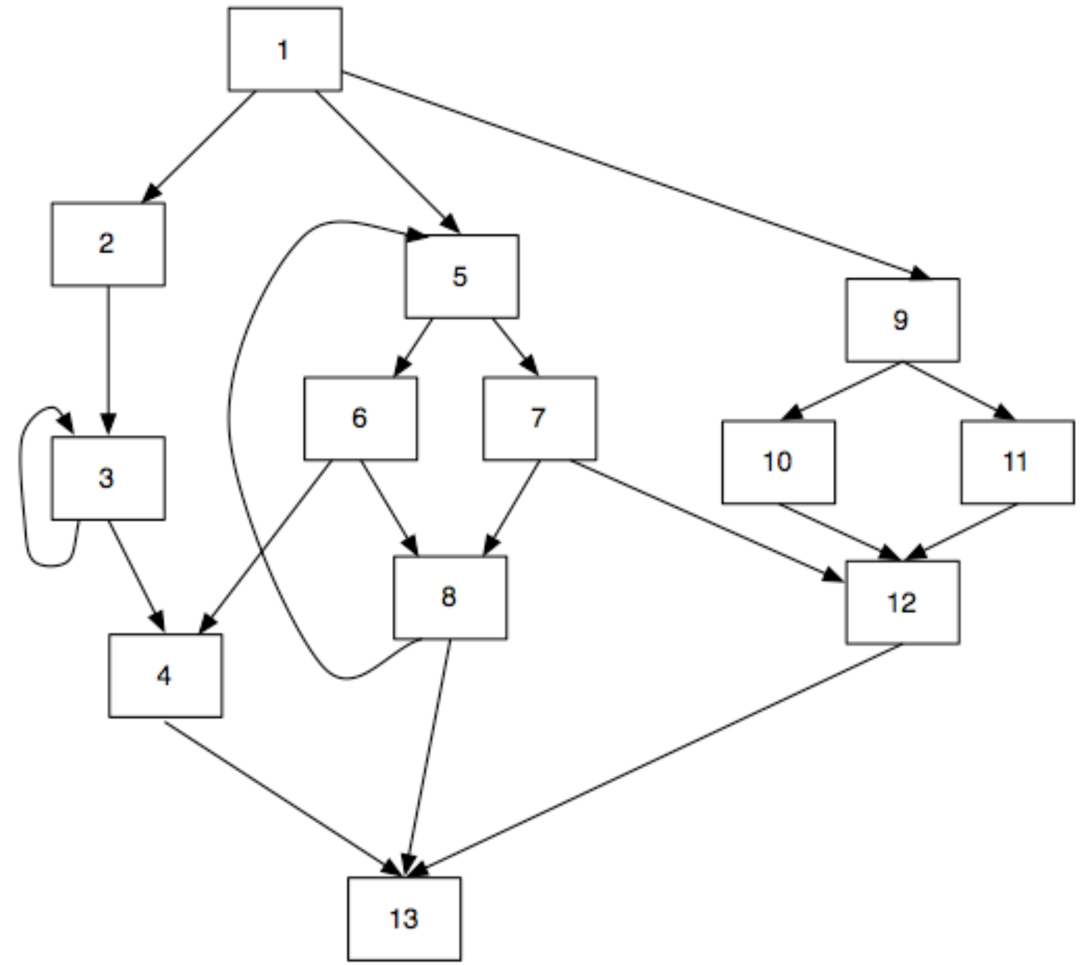
Dominance and SSA Creation

DF(D) = the set of all nodes N such that D dominates an immediate predecessor of N but does not strictly dominate N.

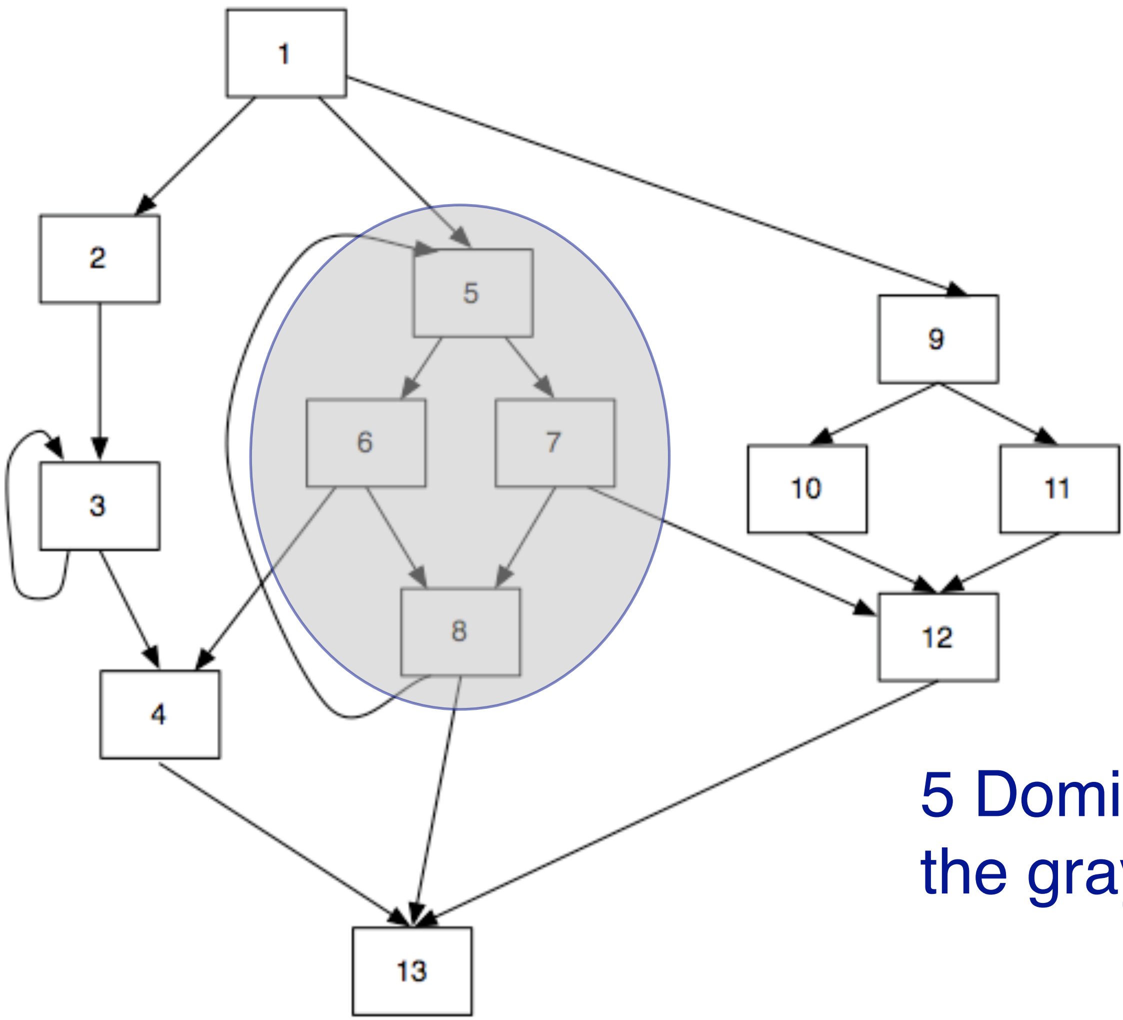
Intuition:

Nodes at the border of a region of dominance

Dominance and SSA Creation

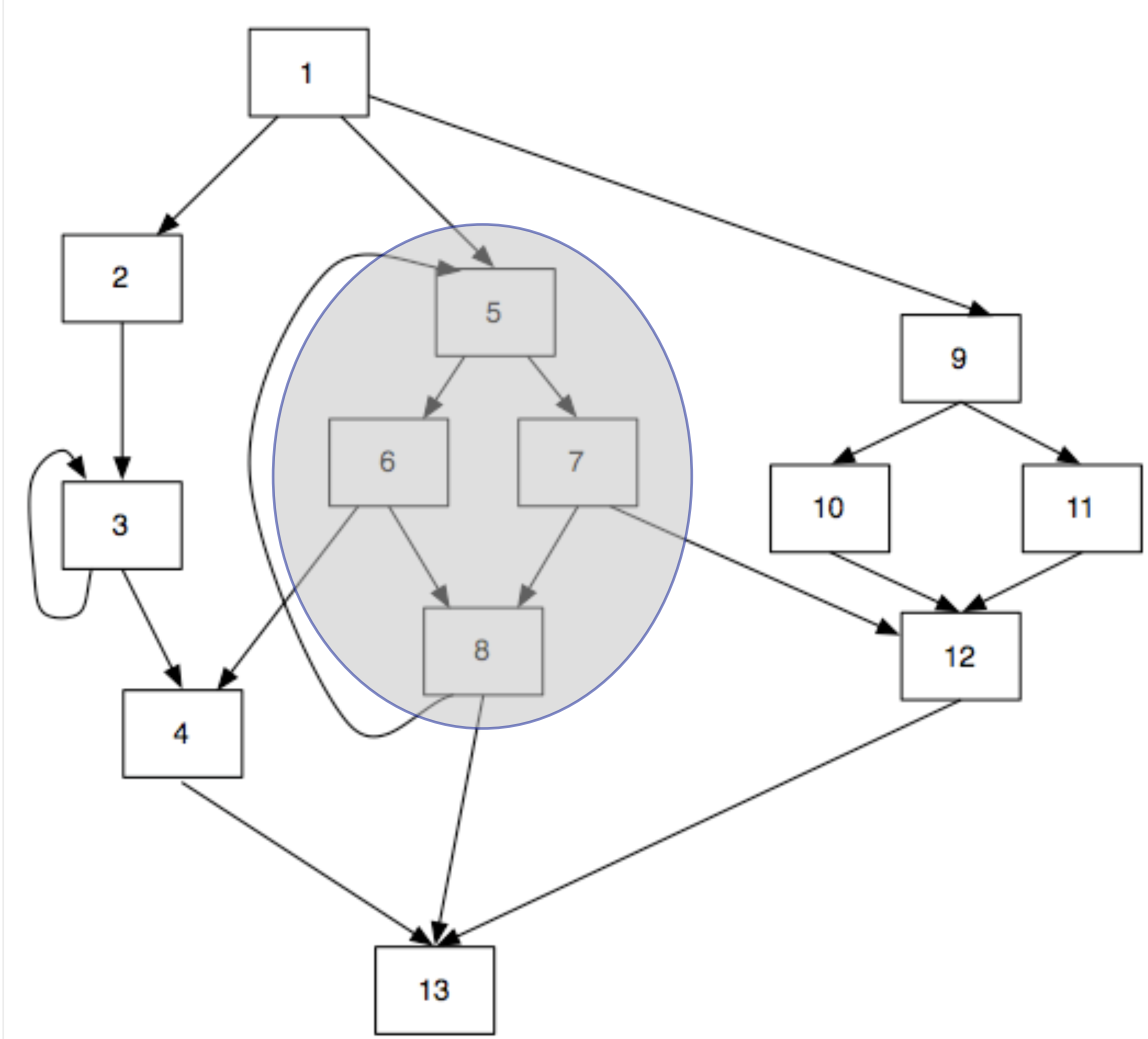


Dominance and SSA Creation



5 Dominates all nodes in the gray area

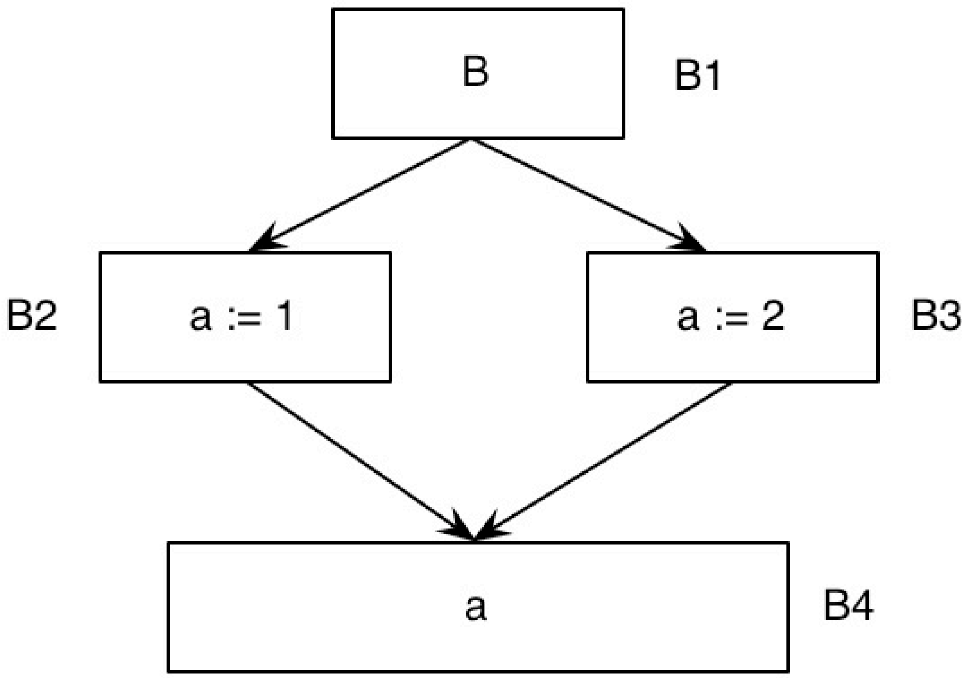
Dominance and SSA Creation



Targets of edges from the dominates by 5 to the region not *strictly dominated by 5.*

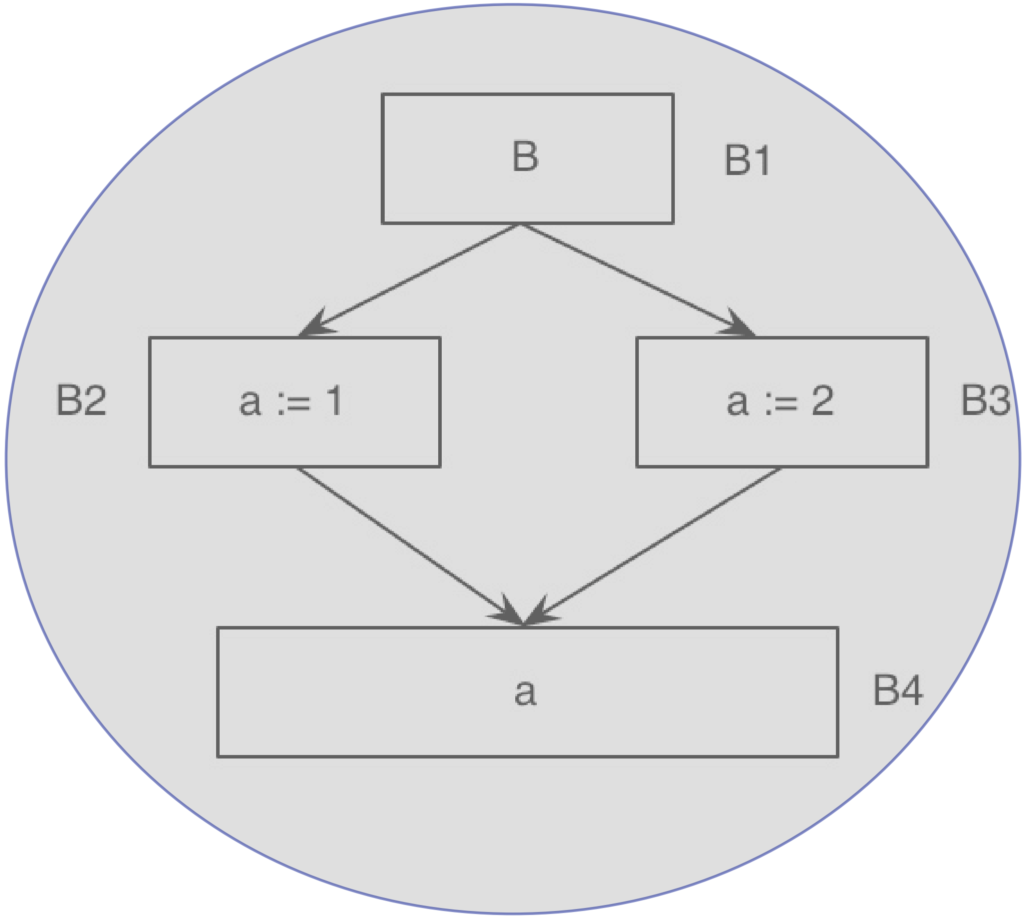
DF(5)= {4, 5, 12, 13}

Simple Example



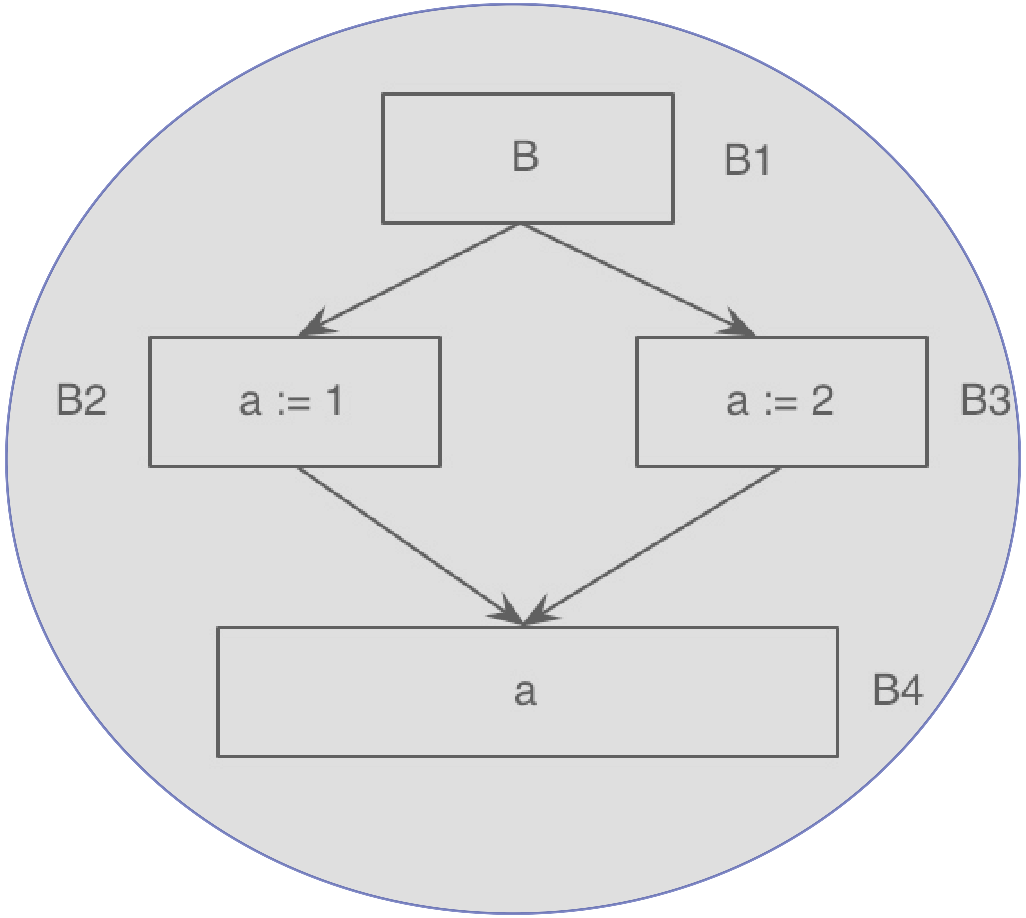
DF(B1)=
DF(B2)=
DF(B3)=
DF(B4)=

Simple Example



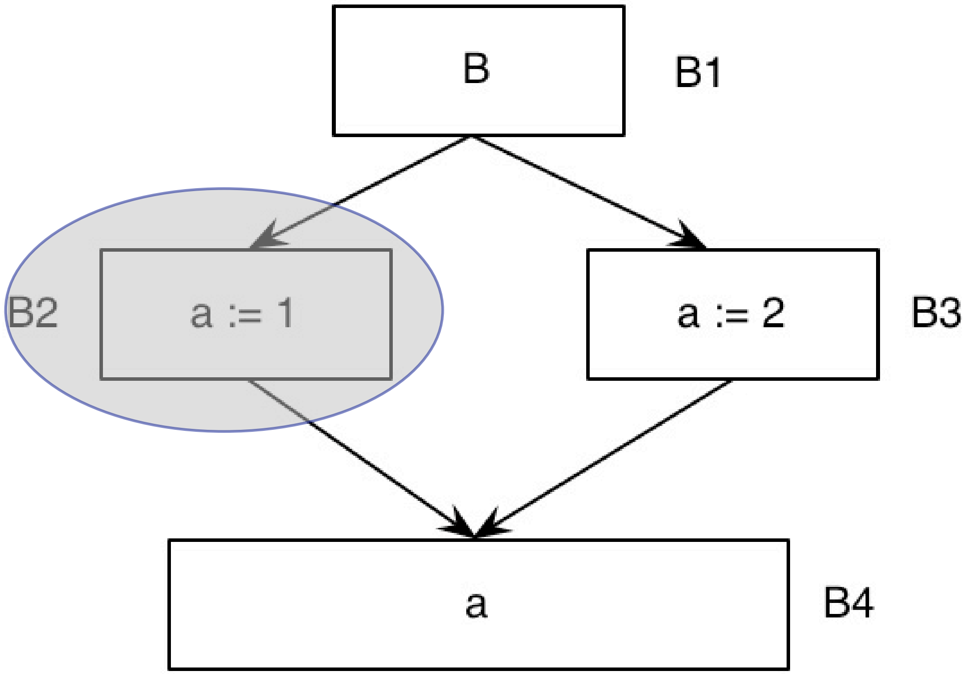
DF(B1)={?}
DF(B2)=
DF(B3)=
DF(B4)=

Simple Example



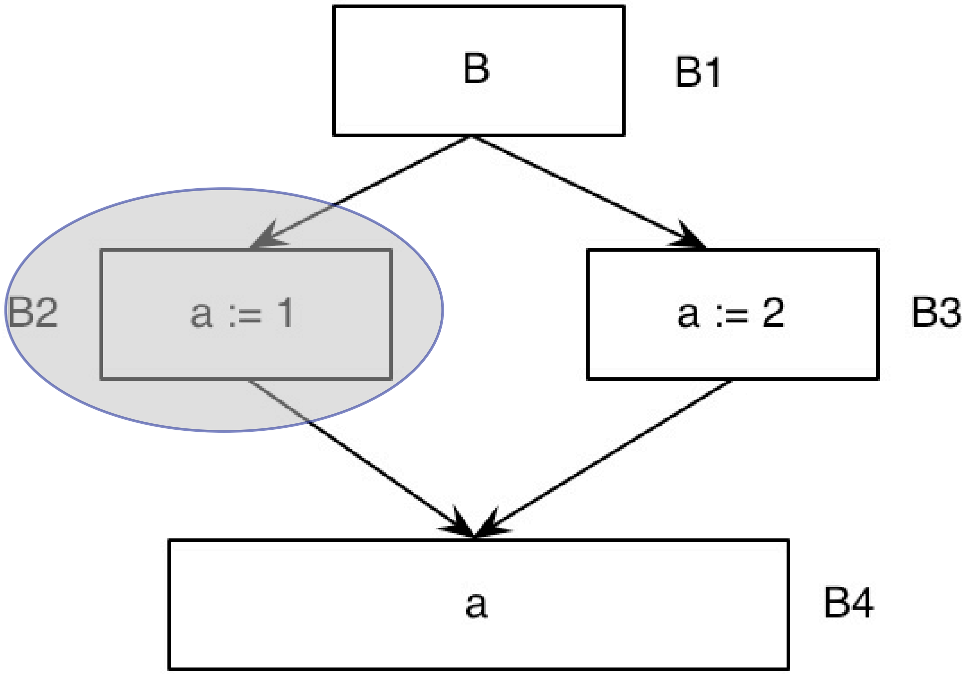
DF(B1)={}
DF(B2)=
DF(B3)=
DF(B4)=

Simple Example



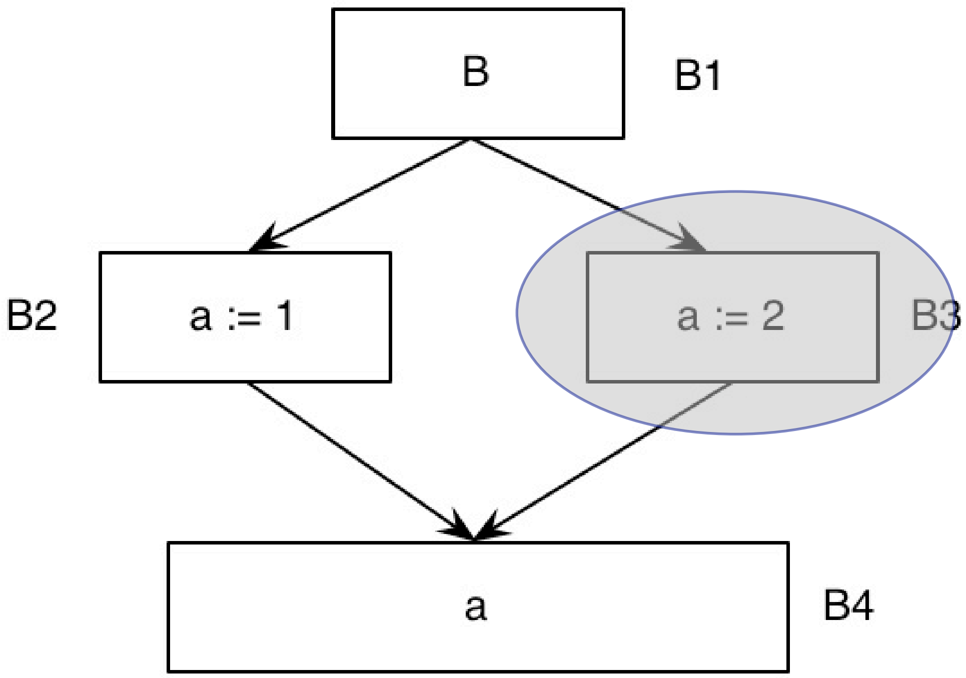
DF(B1)={}
DF(B2)={?}
DF(B3)=
DF(B4)=

Simple Example



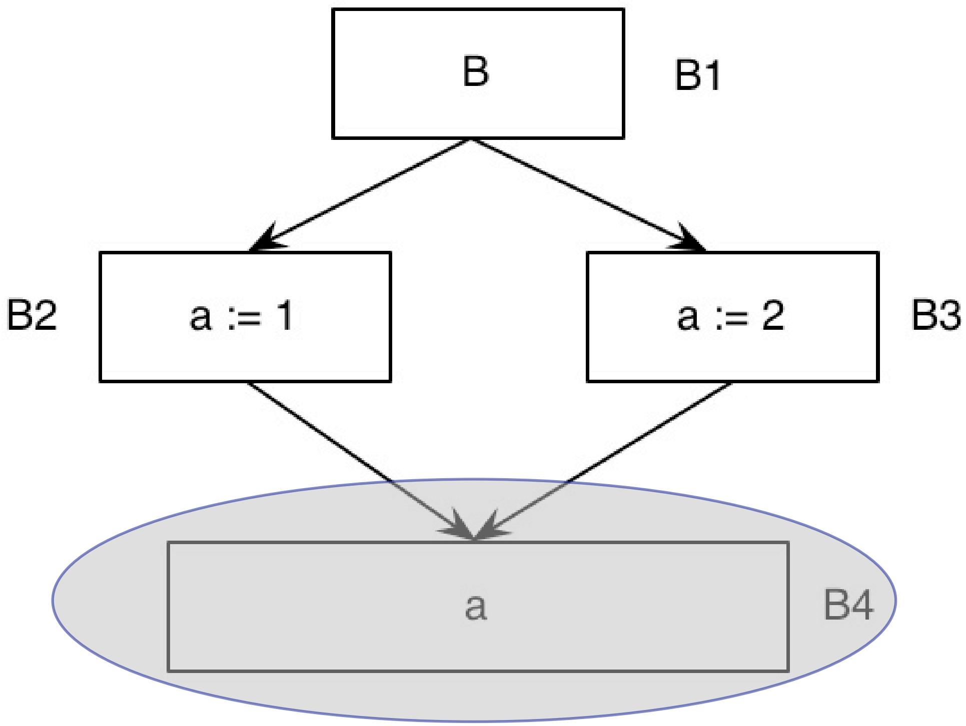
DF(B1)={}
DF(B2)={B4}
DF(B3)=
DF(B4)=

Simple Example



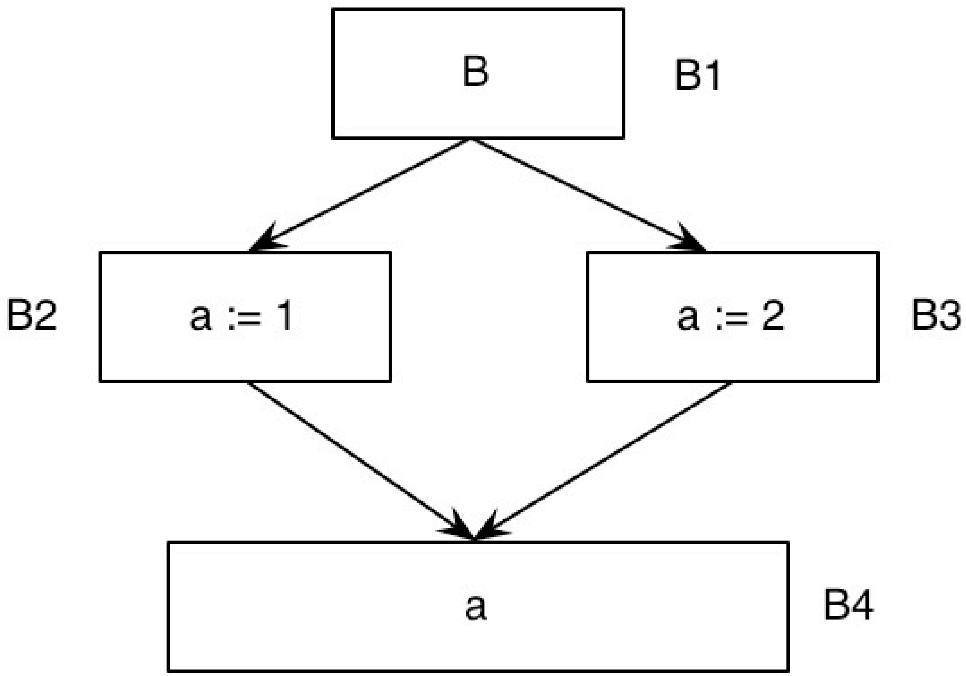
DF(B1)={}
DF(B2)={B4}
DF(B3)={B4}
DF(B4)=

Simple Example



DF(B1)={}
DF(B2)={B4}
DF(B3)={B4}
DF(B4)={}

Simple Example



DF(B1)={}
DF(B2)={B4}
DF(B3)={B4}
DF(B4)={}

PHI-Function needed in B4 (for a)

Properties of SSA

- Simplifies many optimizations
  - *Every variable has only one definition*
  - *Every use knows its definition, every definition knows its uses*
  - *Unrelated variables get different names*

- *Examples:*
  - *Constant propagation*
  - *Value numbering*
  - *Invariant code motion and removal*
  - *Strength reduction*
  - *Partial redundancy elimination*

Next Week!

SSA in the Real World

- Invented end of the 80s, a lot of research in the 90s

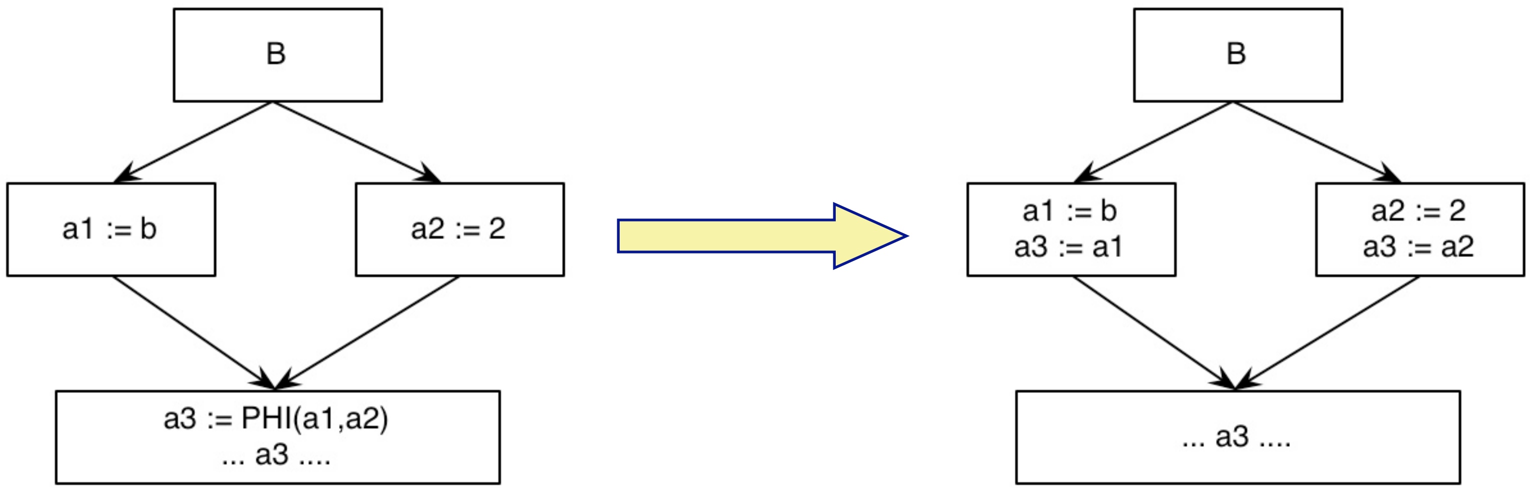
- Used in many modern compilers
  - *ETH Oberon 2*
  - *LLVM*
  - *GNU GCC 4*
  - *IBM Jikes Java VM*
  - *Java Hotspot VM*
  - *Mono*
  - *Many more…*

Transforming out-of SSA

> Processor cannot execute Φ-Function

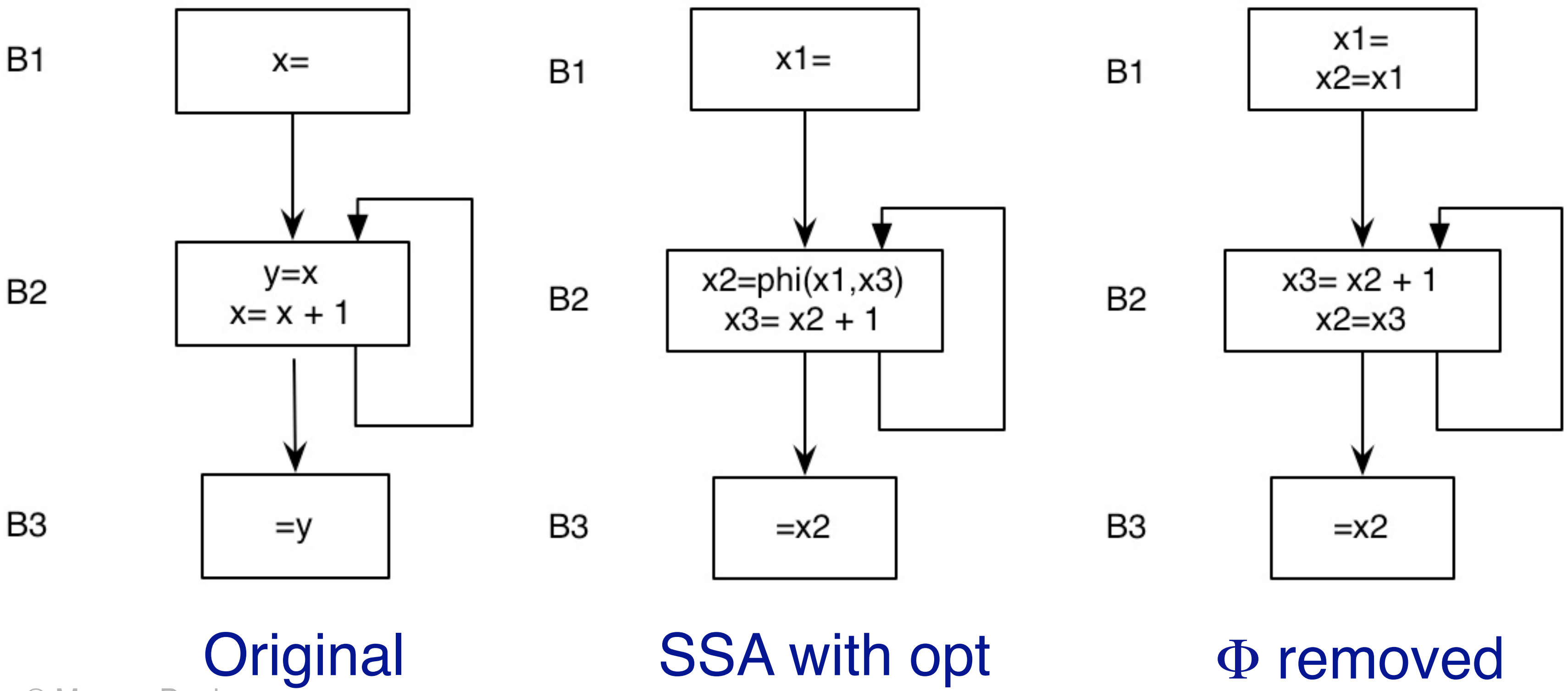
> How do we remove it?

Simple Copy Placement



Problems

> Problems:
- *Copies need to be removed*
- *Wrong in some cases after reordering of code*



Φ-Congruence

Idea: transform program so that all variables in Φ are the same:

a1 = Φ(a1, a1) ---> a1 = a1

- Insert Copies
- Rename Variables

Φ-Congruence: Definitions

Φ-connected(x):

a3 = Φ(**a1**, a2)
a5 = Φ(a3, **a4**)

--> a1, a4 are connected

Φ-congruence-class:

Transitive closure of Φ-connected(x).

Φ-Congruence Property

Φ-congruence property:

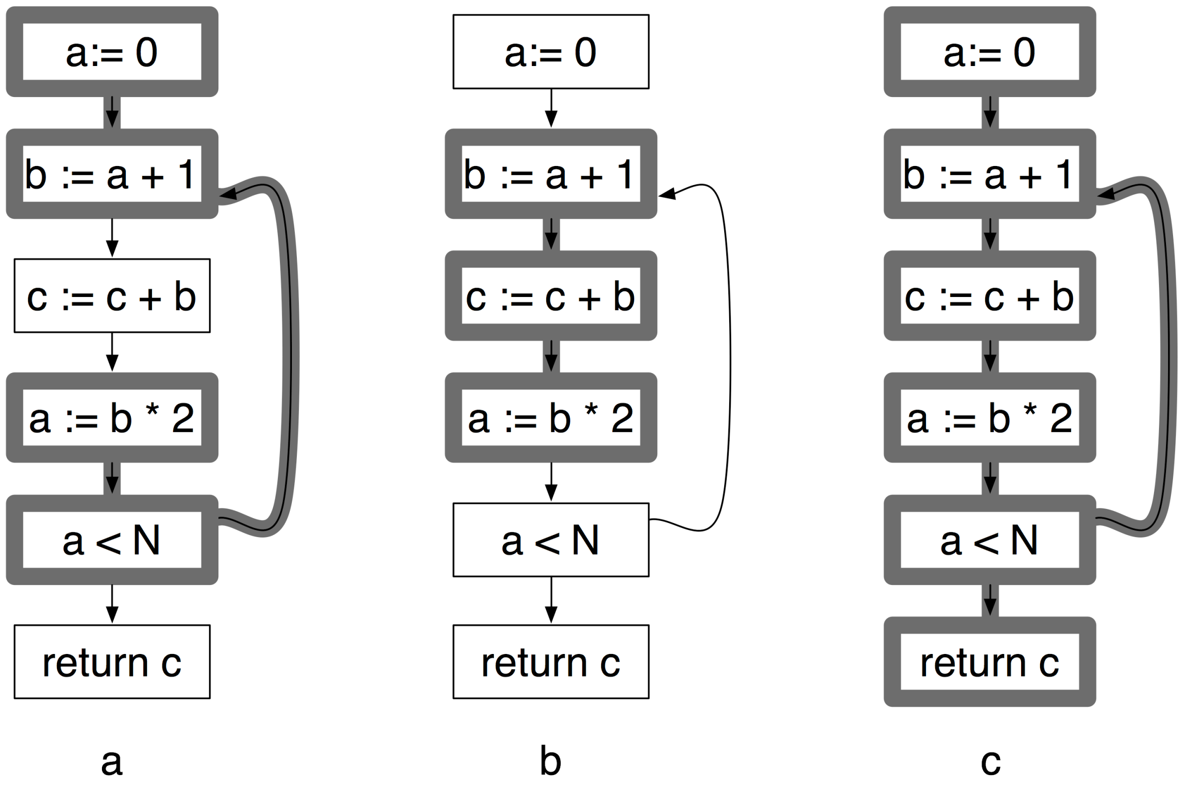
All variables of the same congruence class can be replaced by one representative variable without changing the semantics.

SSA without optimizations has Φ-congruence property

Variables of the congruence class never live at the same time (by construction)

Repeat: Liveness

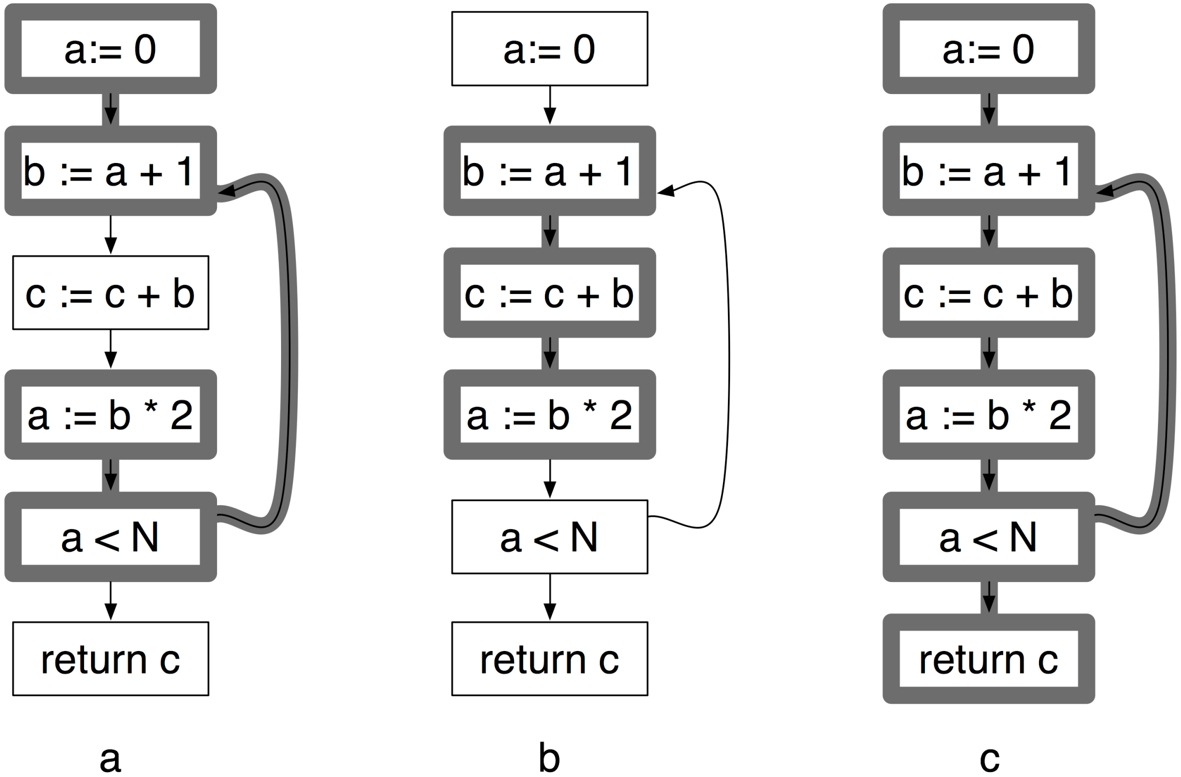
A variable *v* is *live* on edge *e* if there is a path from *e* to a use of *v* not passing through a definition of *v*



a and b are never live at the same time, so two registers suffice to hold a, b and c

Interference

A variable *v* is *live* on edge *e* if there is a path from *e* to a use of *v* not passing through a definition of *v*



a, c live at the same time: interference

Φ-Removal: Big picture

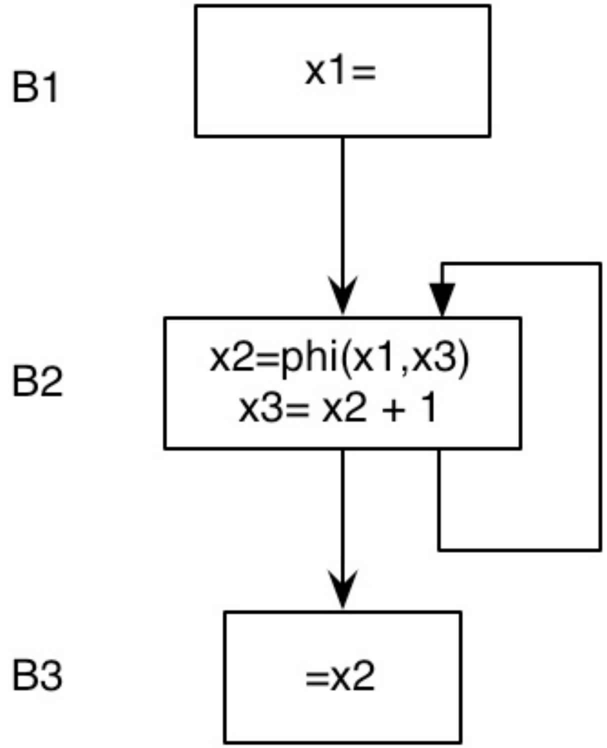
CSSA: SSA with Φ-congruence-property.
- *directly after SSA generation*
- *no interference*

TSSA: SSA without Φ-congruence-property.
- after optimizations
- interference

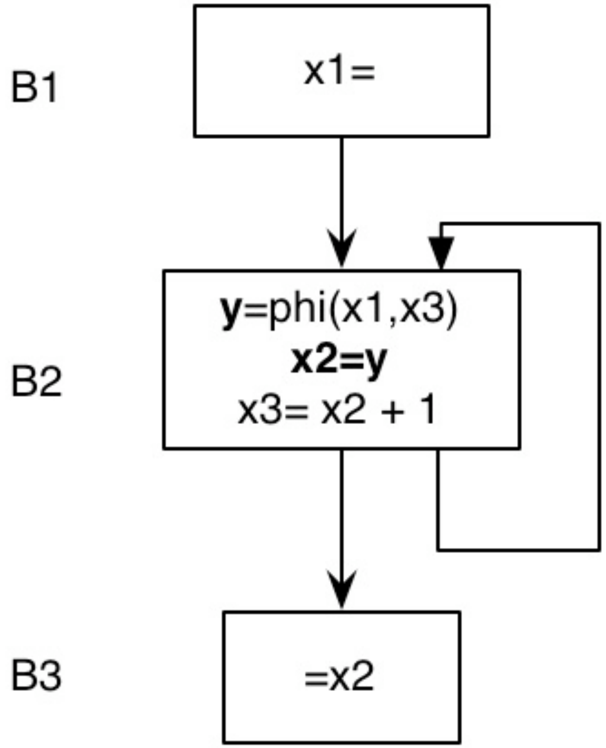
1. Transform TSSA into CSSA (fix interference)
2. Rename Φ-variables
3. Delete Φ

Example: Problematic case

X2 and X3 interfere



Solution: Break up



SSA and Register Allocation

- Idea: remove Φ as late as possible
- Variables in Φ-function never live at the same time!
  — *Can be stored in the same register*
- Do register allocation on SSA!

SSA: Literature

Books:

- SSA Chapter in Appel
  Modern Compiler Impl. In Java
- Chapter 8.11 Muchnik:
  Advanced Compiler Construction

SSA Creation:

Cytron et. al: *Efficiently computing Static Single Assignment Form and the Control Dependency Graph* (TOPLAS, Oct 1991)

PHI-Removal: Sreedhar et at. *Translating out of Static Single Assigment Form* (LNCS 1694)

Summary

> SSA, what it is and how to create it
  — Where to place Φ-functions?

> Transformation out of SSA
  — Placing copies
  — Remove Φ

Next Week: Optimizations

What you should know!

- ✎ *When a program has SSA form.*
- ✎ *What is a Φ-function.*
- ✎ *When do we place Φ-functions*
- ✎ *How to remove Φ-functions*

Can you answer these questions?

- ✎ *Why can we not directly generate executable code from SSA?*
- ✎ *Why do we use 3-adress code and CFG for SSA?*

License

> http://creativecommons.org/licenses/by-sa/2.5/