SSA

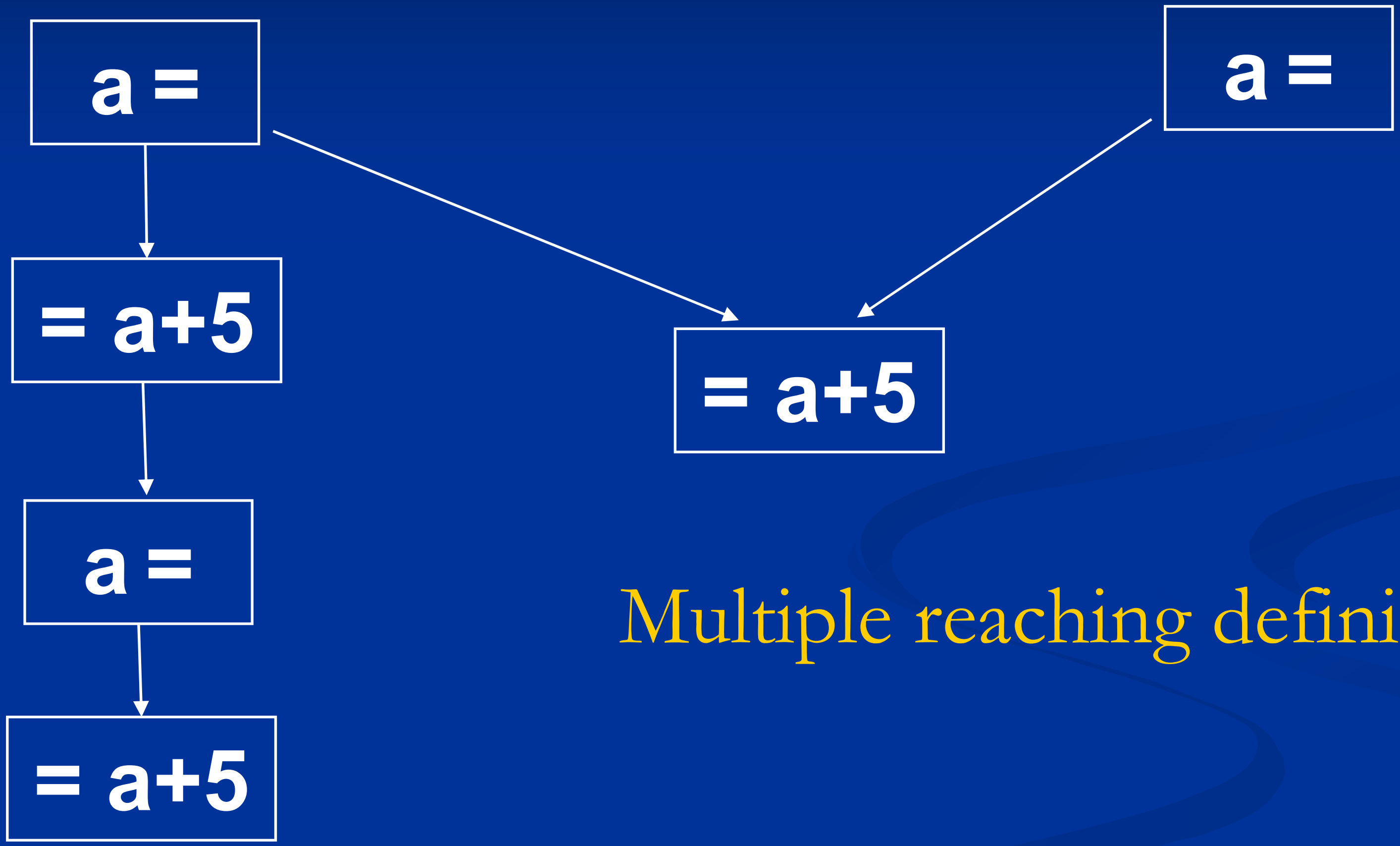
Overview

- *SSA Representation*
- SSA Construction
- Converting out of SSA

Static Single Assignment

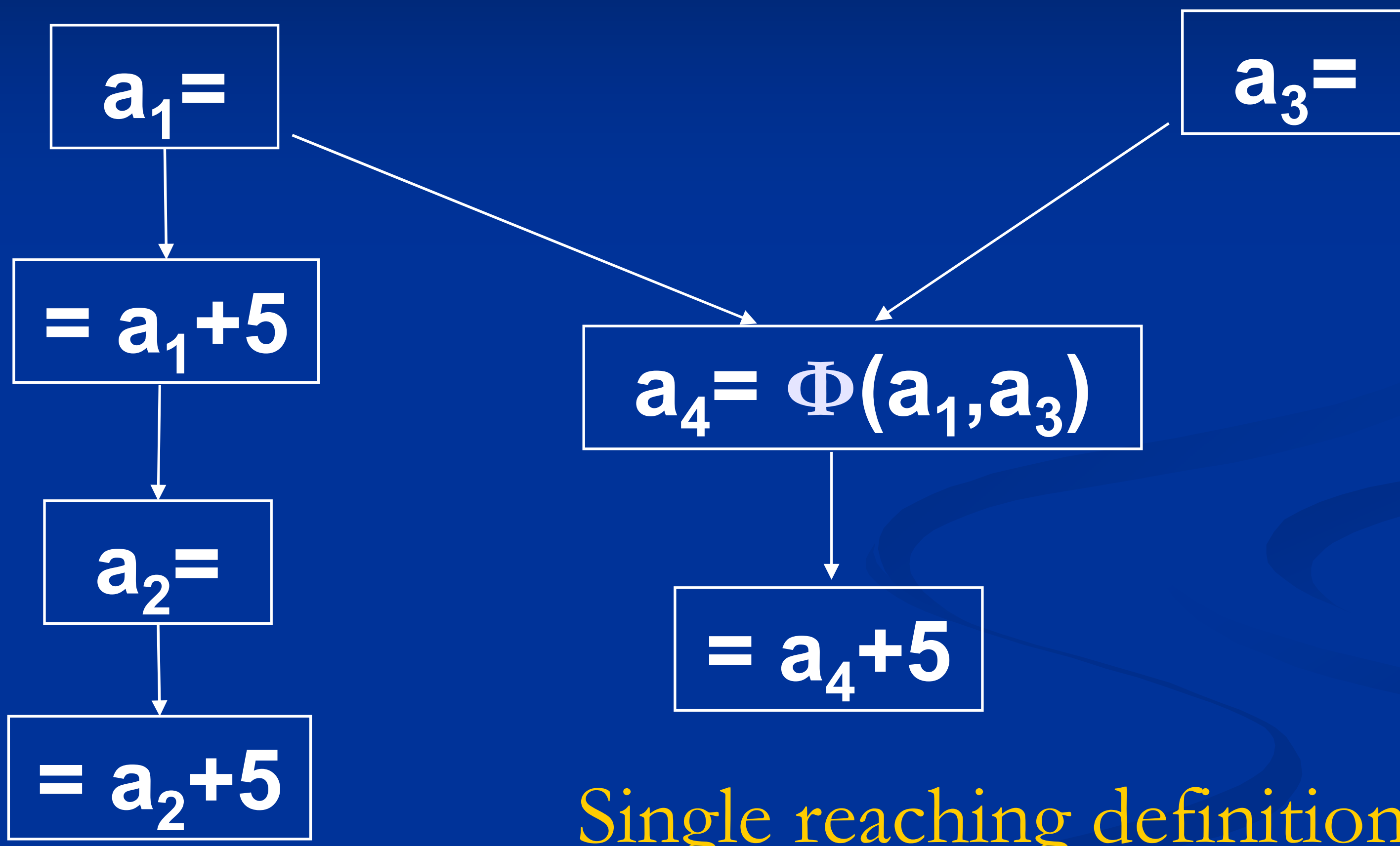
- Each variable has only one reaching definition.
- When two definitions merge, a Φ function is introduced to with a new definition of the variable.
- First consider SSA for alias free variables.

Example: CFG



Multiple reaching definitions

Example: SSA Form



Single reaching definition

Φ Functions

- A Φ operand represents the reaching definition from the corresponding predecessor.
- The ordering of Φ operands are important for knowing from which path the definition is coming from.

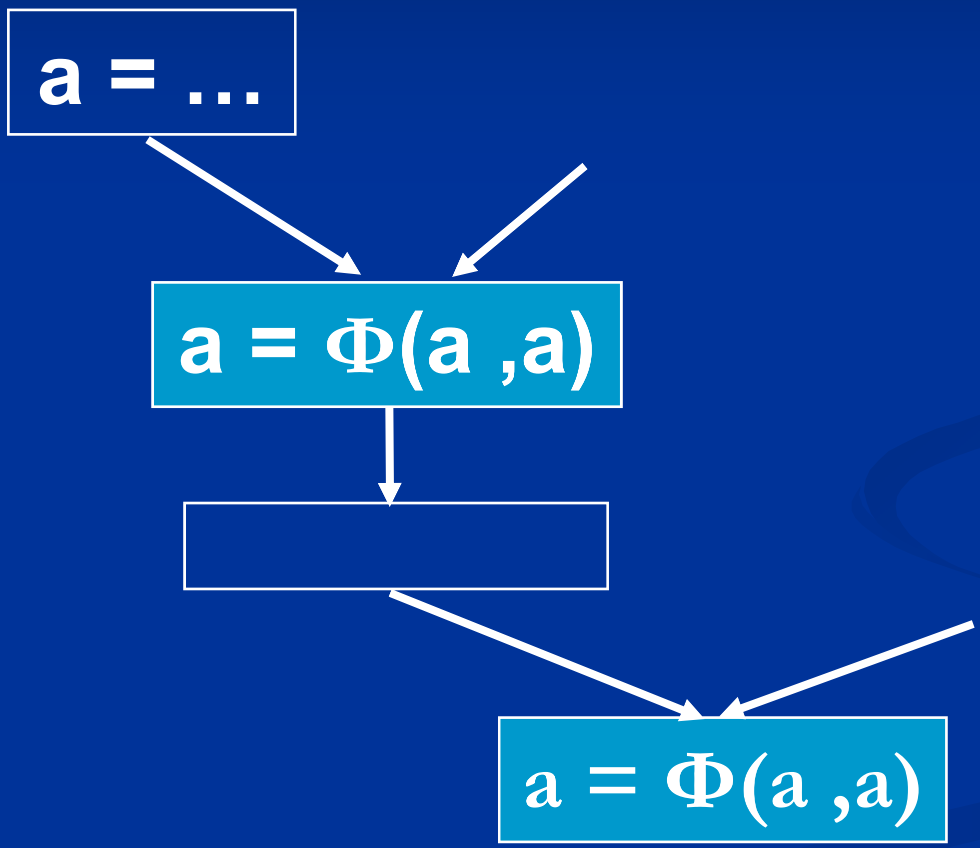
SSA Conditions

1. If two nonnull paths $X \rightarrow^{+} Z$ and $Y \rightarrow^{+} Z$ converge at node Z, and nodes X and Y contains (V =..), then $V = \Phi(V, .., V)$ has been inserted at Z.
2. Each mention of V has been replaced by a mention of V_i
3. V and the corresponding V_i have the same value.

Overview

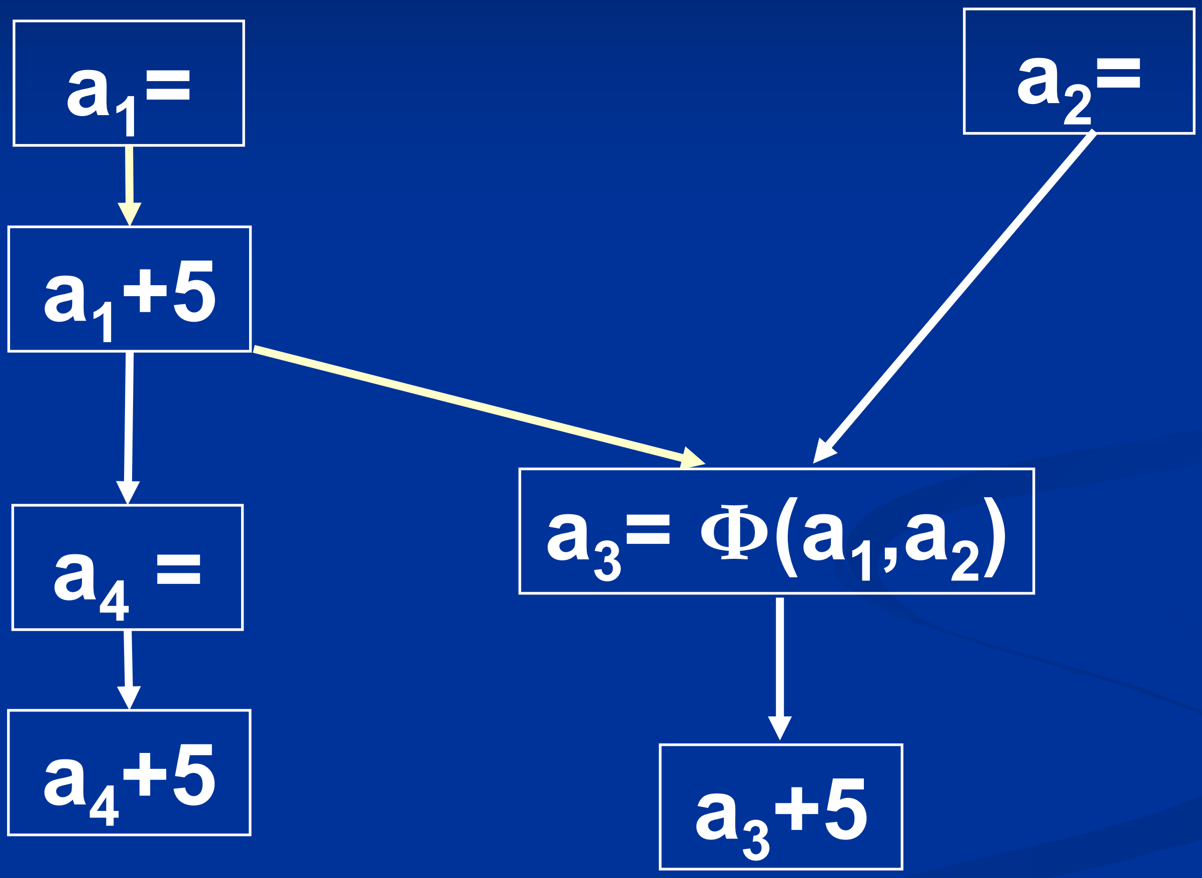
- SSA Representation
- *SSA Construction*
  - *Step 1: Place Φ statements*
  - *Step 2: Rename all variables*
- Converting out of SSA

Φ Placement



Place minimal number of Φ functions

Renaming



SSA Construction (I)

- Step 1: Place Φ statements by computing iterated dominance frontier

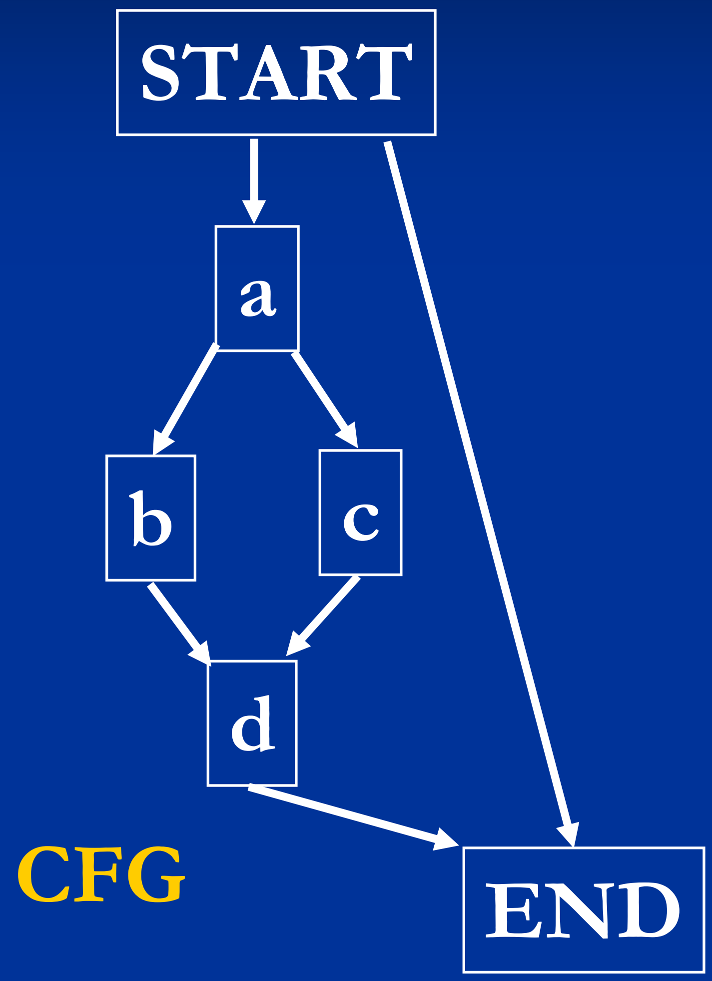
CFG

- A control flow graph $G = (V, E)$
- Set V contains distinguished nodes START and END
  - every node is reachable from START
  - END is reachable from every node in G.
  - START has no predecessors
  - END has no successors.
- Predecessor, successor, path

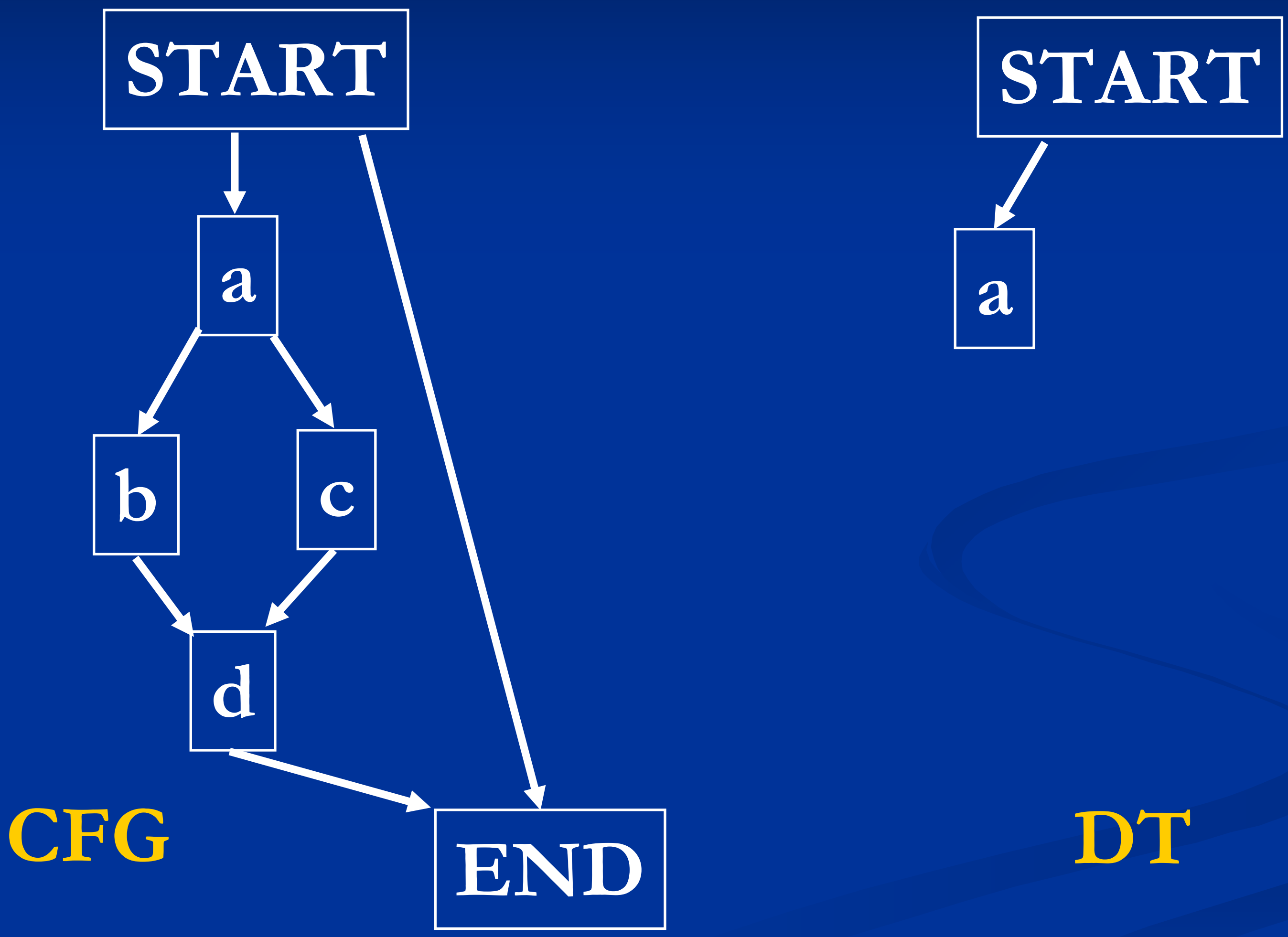
Dominator Relation

- If X appears on every path from START to Y, then X ***dominates*** Y.
- Domination is both reflexive and transitive.
- idom(Y): immediate dominator of Y
- Dominator Tree
  - START is the root
  - Any node Y other than START has idom(Y) as its parent
  - Parent, child, ancestor, descendant

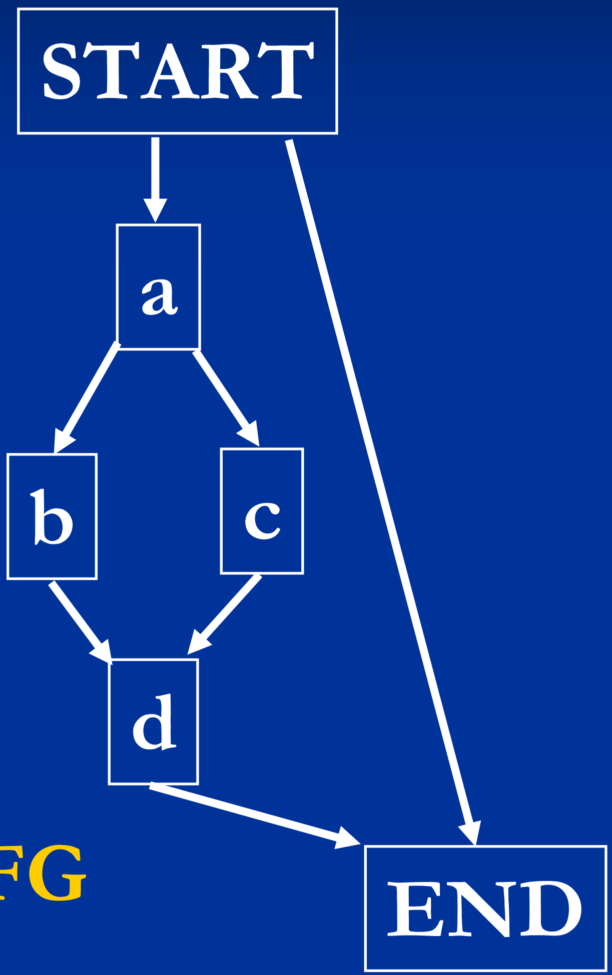
Dominator Tree Example



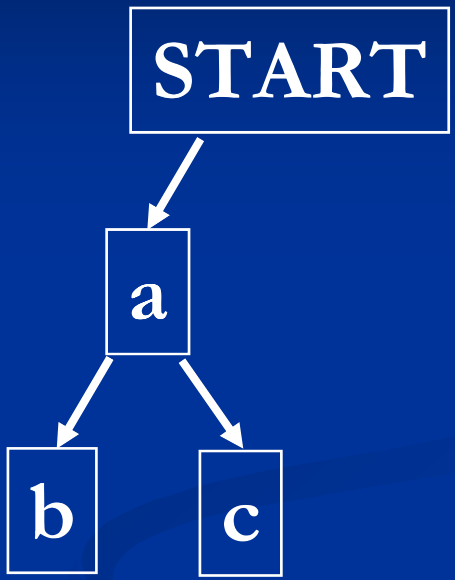
Dominator Tree Example



Dominator Tree Example

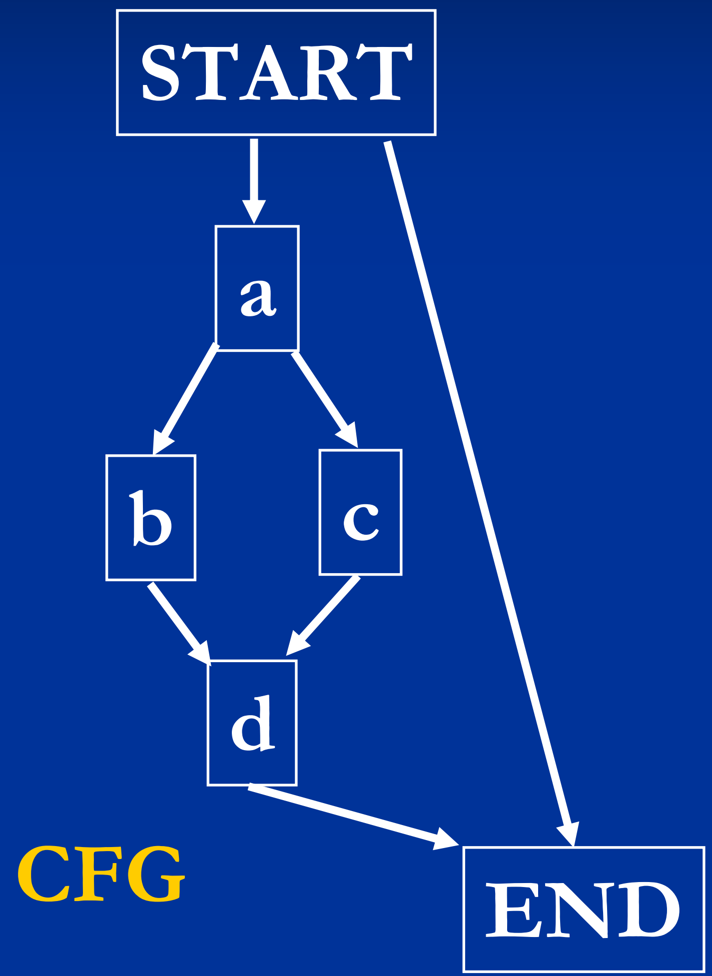


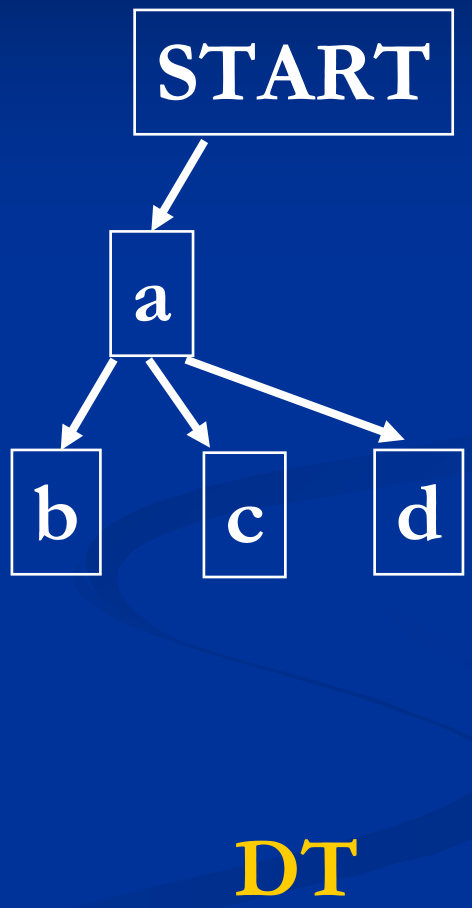
CFG



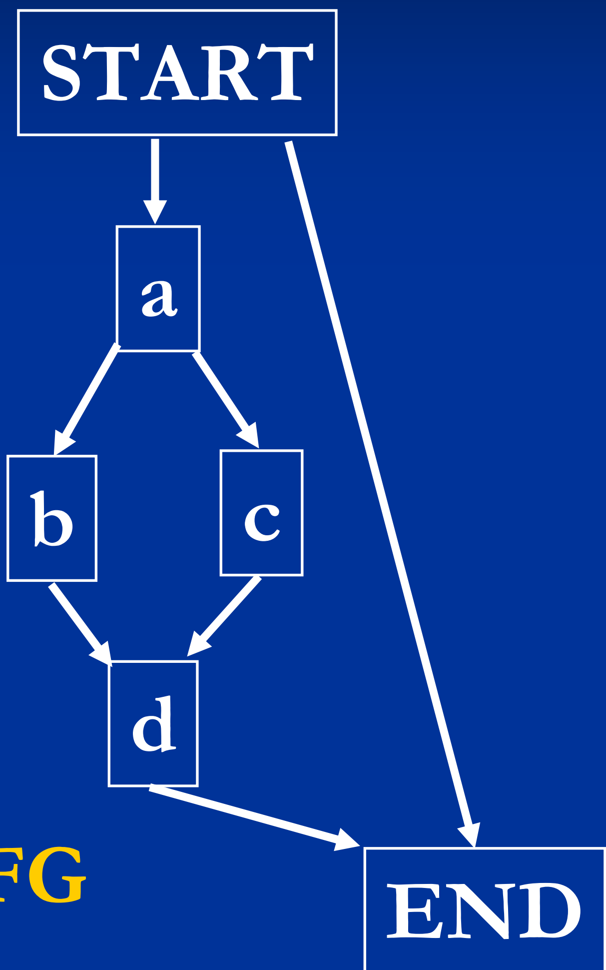
DT

Dominator Tree Example

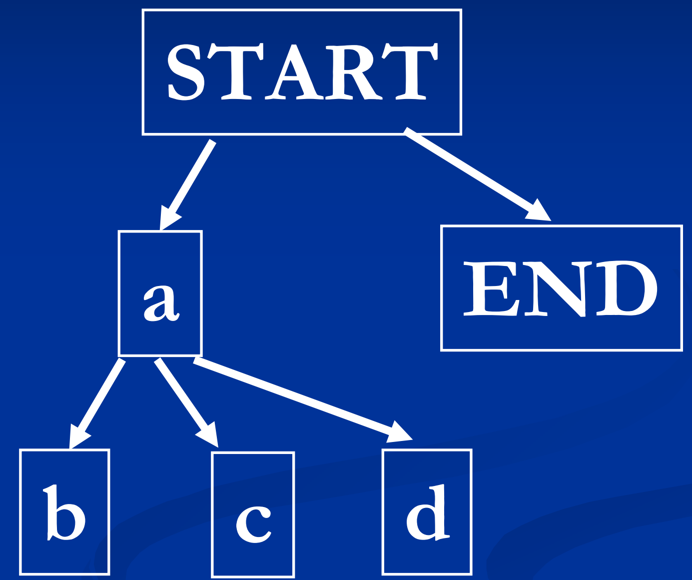




Dominator Tree Example



CFG

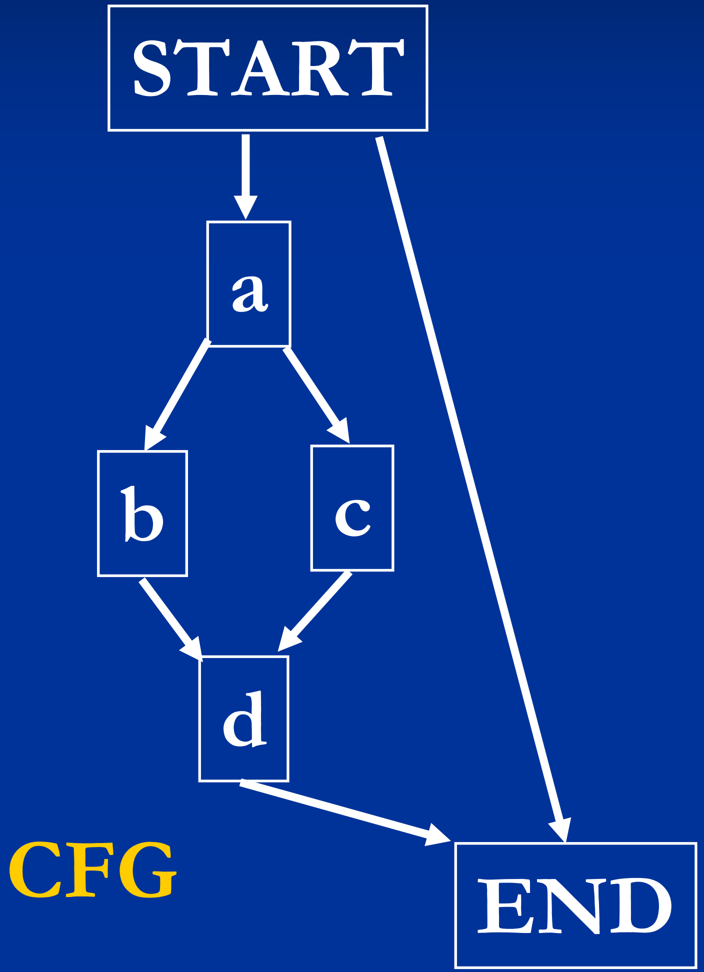


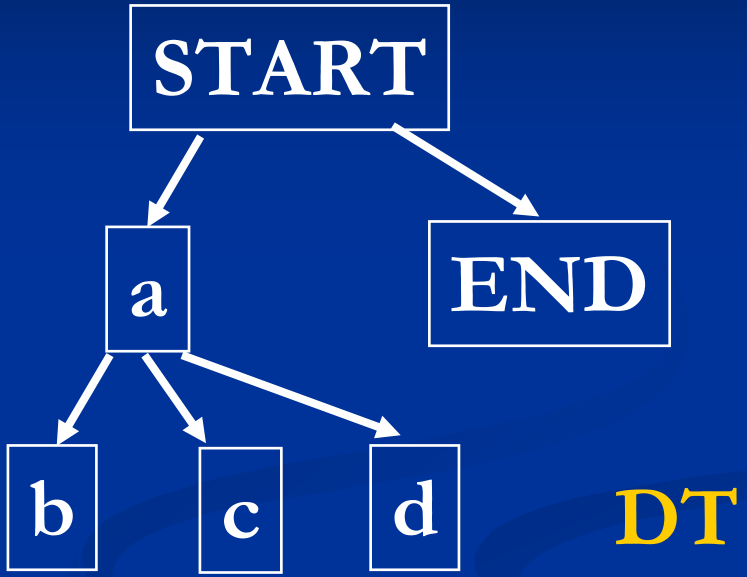
DT

Dominance Frontier

- Dominance Frontier DF(X) for node X
  - Set of nodes Y
  - X dominates a predecessor of Y
  - X does not strictly dominate Y

DF Example

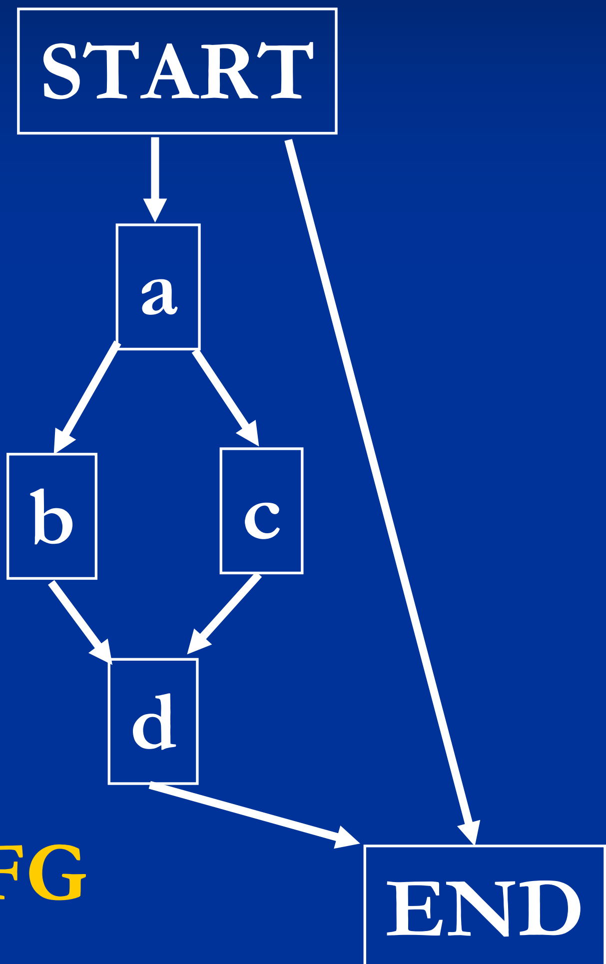




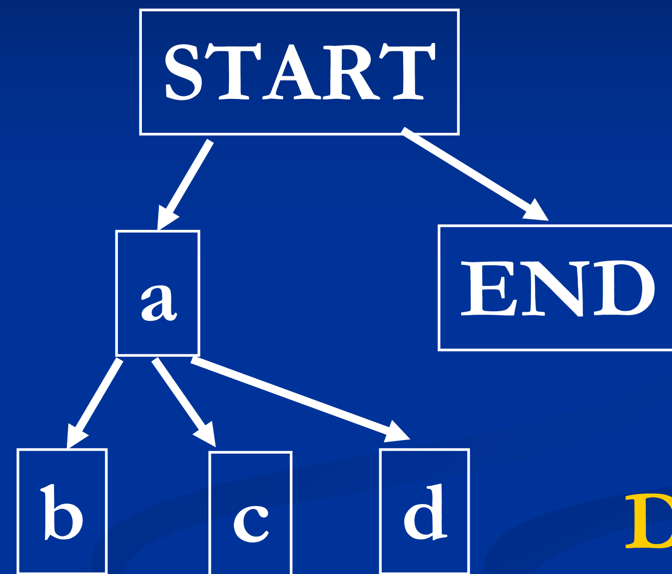
DF(c) = ?

DF(a) = ?

DF Example



CFG

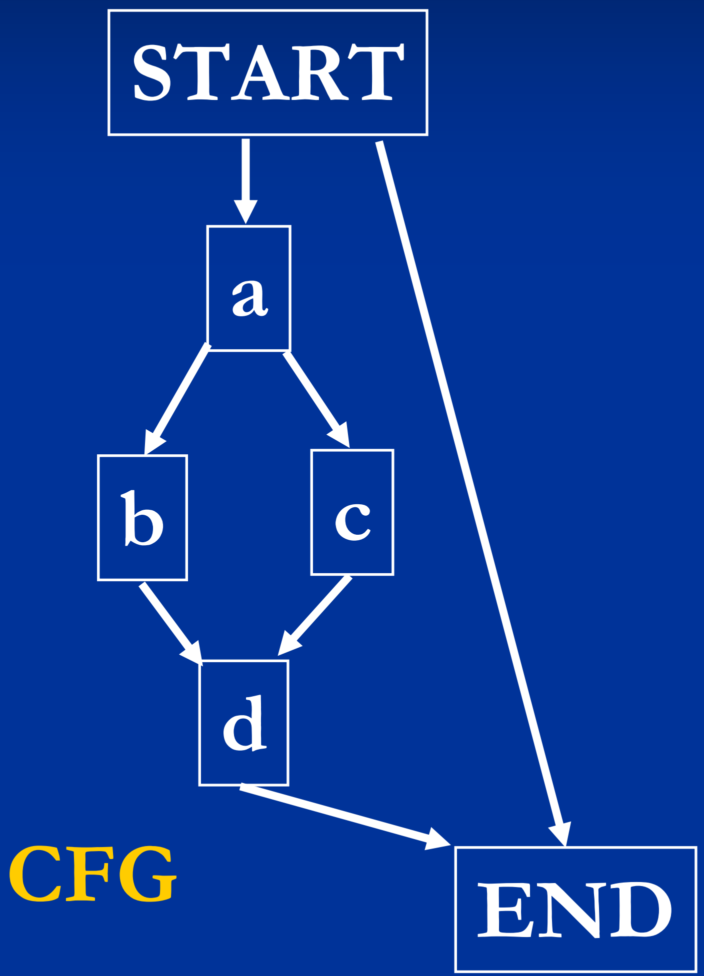


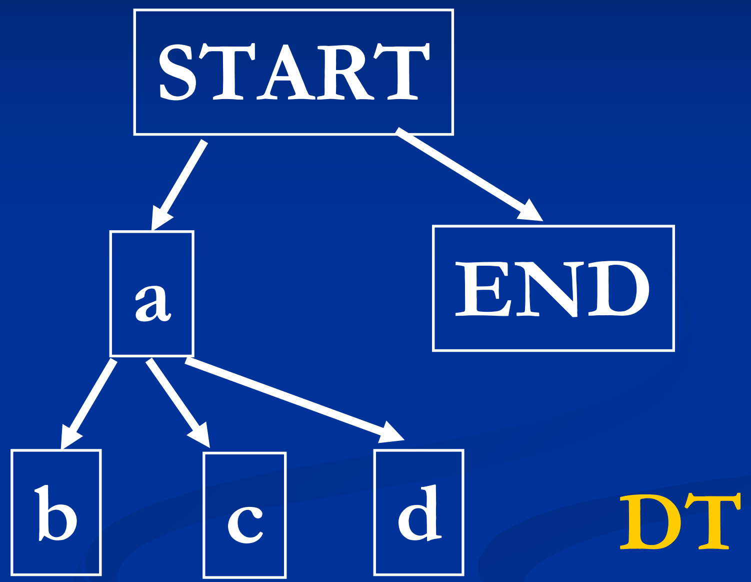
DT

DF(c) = {d}

DF(a) = ?

DF Example





DF(c) = {d}

DF(a) = {END}

Computing DF

- DF(X) is the union of the following sets
  - $DF_{local}(X)$, a set of successor nodes that X doesn't strictly dominate
    - E.g. $DF_{local}(c) = \{d\}$
  - $DF_{up}(Z)$ for all Z є Children(X)
    - $DF_{up}(Z) = \{Y \in DF(Z) \mid idom(Z)$ doesn't strictly dominate $Y\}$
    - E.g. X = a, Z = d, Y = END

Iterated Dominance Frontier

- DF(SET) is the union of DF(X), where X є SET.
- Iterated dominance frontier $DF^+(SET)$ is the limit of
  - $DF_1 = DF(SET)$ and $DF_{i+1} = DF(SET \cup DF_i)$

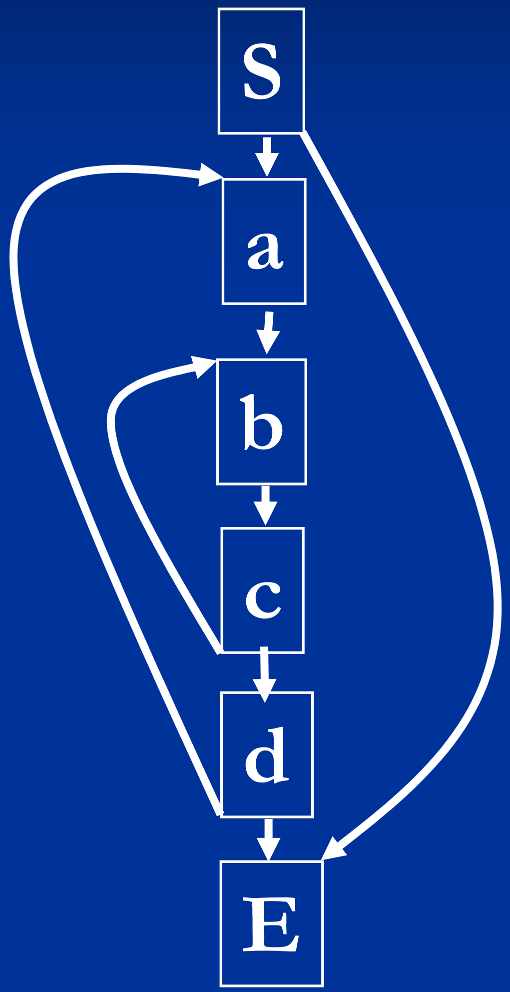
Computing Joins

- J(SET) of join nodes
  - Set of all nodes Z
  - There are two nonnull CFG paths that start at two distinct nodes in SET and converge at Z.
- Iterated join J^+(SET) is the limit of
  - J_1 = J(SET) and J_{i+1} = J(SET U J_i)
- J^+(SET) = DF^+(SET)

Placing Φ Functions

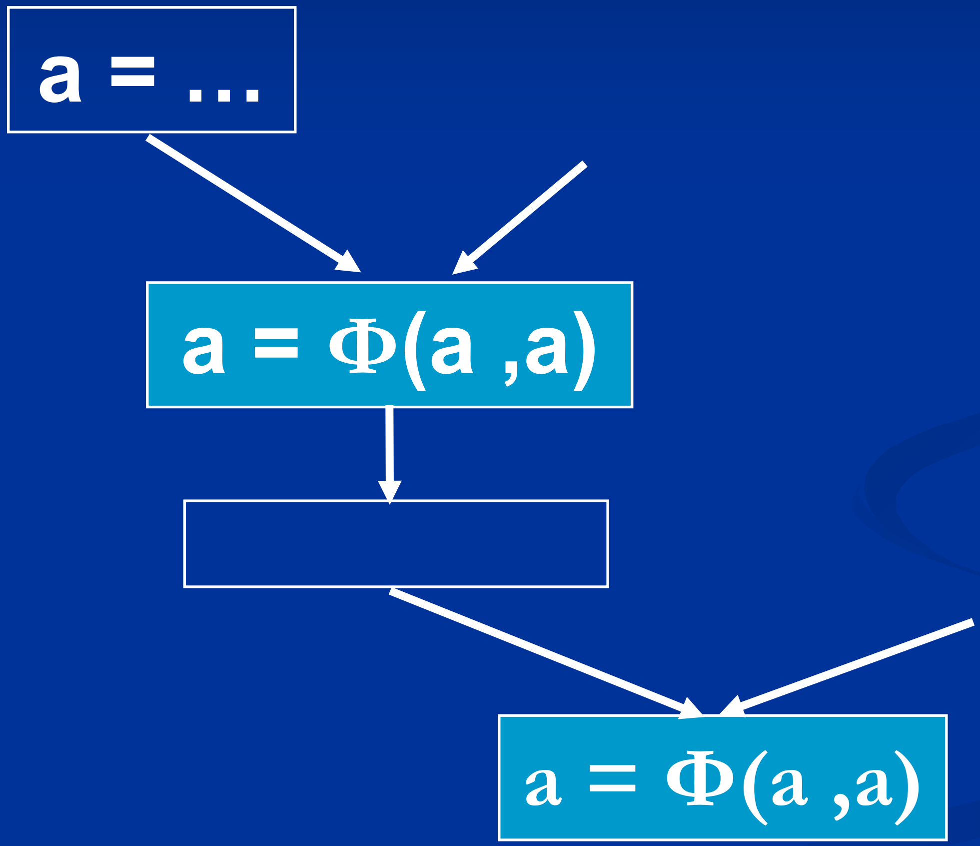
- For each variable V
  - Add all nodes with assignments to V to worklist W
  - While X in W do
    - For each Y in DF(X) do
      - If no Φ added in Y then
        - Place (V = Φ (V,…,V)) at Y
        - If Y has not been added before, add Y to W.

Computational Complexity



- Constructing SSA takes $O(A_{tot} * avrgDF)$, where
  - A_{tot}: total number of assignments
  - avrgDF: weighted average DF size
- The computational complexity is $O(n^2)$.
  - e.g. nested repeat-until loops

Φ Placement Example



Place Φ at Iterative Dominance Frontiers

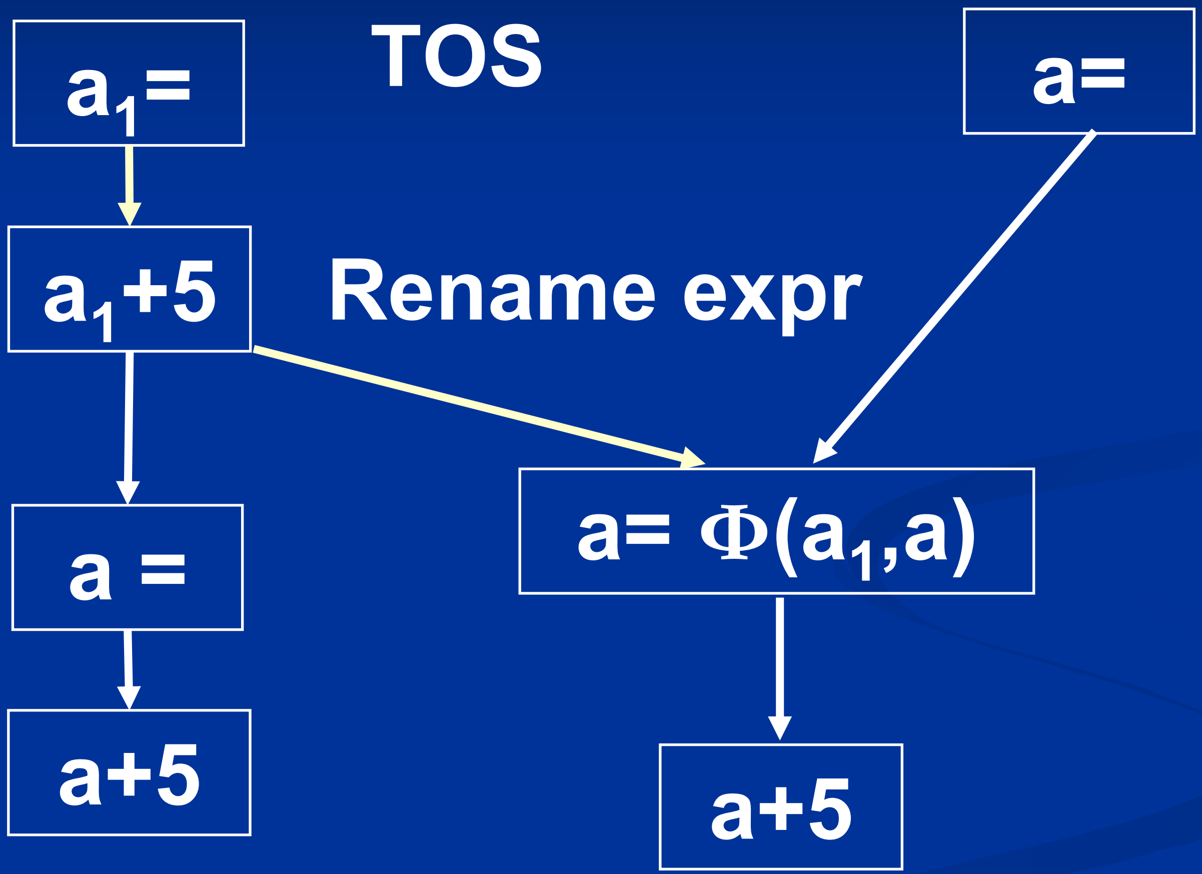
SSA Construction (II)

- Step 2: Rename all variables in original program and Φ functions, using dominator tree and rename stack to keep track of the current names.

Variable Renaming

- Rename from the START node recursively
- For node X
  - For each assignment (V = …) in X
    - Rename any use of V with the TOS of rename stack
    - Push the new name V_i on rename stack
    - $i = i + 1$
  - Rename all the Φ operands through successor edges
  - Recursively rename for all child nodes in the dominator tree
  - For each assignment (V = …) in X
    - Pop V_i in X from the rename stack

Renaming Example



Overview

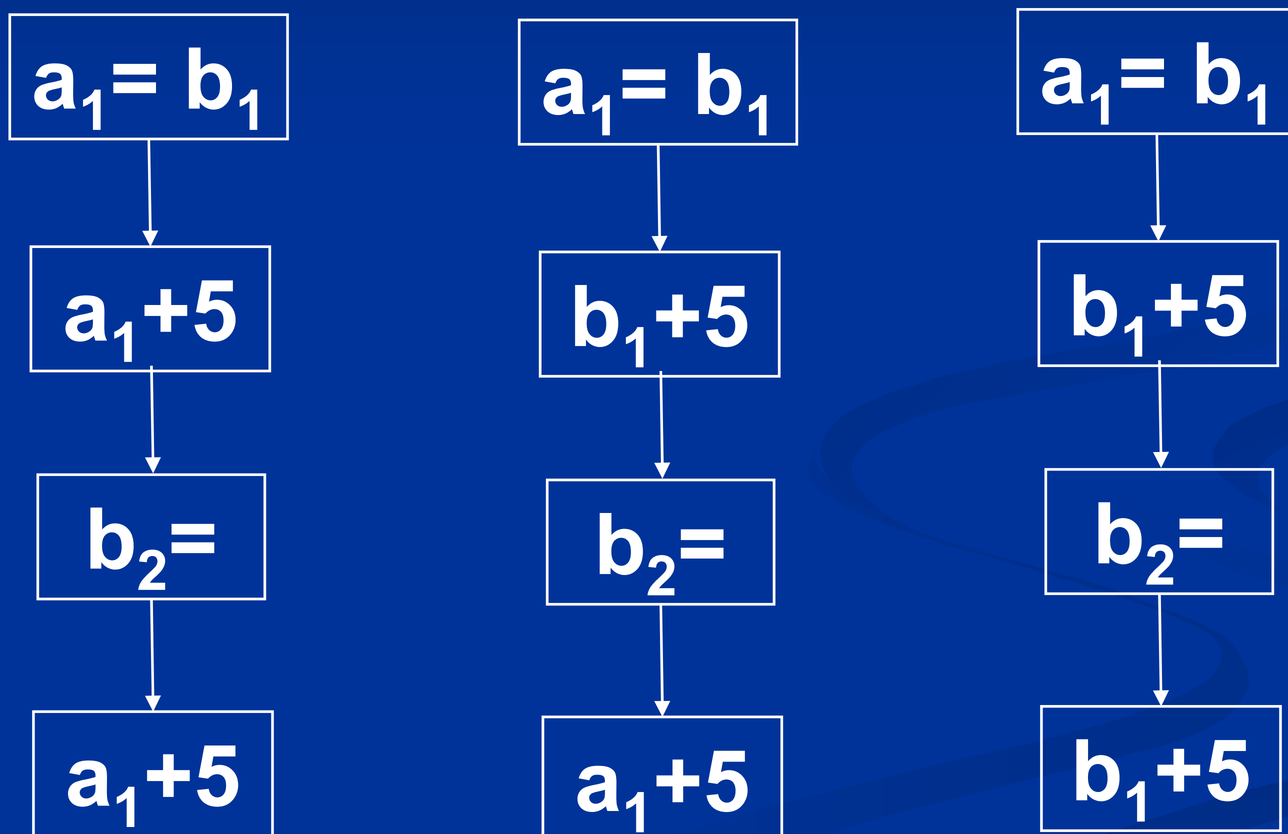
- SSA Representation
- SSA Construction
- *Converting out of SSA*

Converting Out of SSA

- Mapping all V_i to V?

Overlapping Live Ranges

- Simply mapping all V_i to V may not work



Converting Out of SSA

- Option 1: coloring
  - Compute live ranges, and assign a unique variable name for each live range
  - Similar techniques used in register allocation to be covered next week.
- Option 2: simply remove all Φ functions
  - Every optimization in SSA needs to guarantee not to generate overlapping live ranges

Reference

- "Efficient Computing Static Single Assignment Form and the Control Dependence Graph", R. Cytron, J. Ferrante, B. Rosen, M. Wegman, and F. K. Zadeck, Transactions on Programming Languages and Systems (TOPLAS), Oct 1991. http://citeseer.ist.psu.edu/cytron91efficiently.html

Backup

Handling Arrays

- Difficult to treat A[i] as a variable

$= A[i]$ $A[j] = V$ $= A[k]$	$= R(A,i)$ $A = W(A,j,V)$ $= R(A,k)$	$= R(A_8,i_7)$ $A_9 = W(A_8,j_6,V_5)$ $= R(A_9,k_4)$

- The entire array can be treated like a scalar.

Unnecessary Liveness

- W operator may introduce unnecessary liveness for A. Introduce HW (HiddenW).

repeat	repeat	repeat
	$i_2 = \Phi(i_1,i_3)$	$i_2 = \Phi(i_1,i_3)$
	$A_1 = \Phi(A_0,A_2)$	
$A[i] = i$	$A_2 = W(A_1,i_2,i_2)$	$A_2 = HW(i_2,i_2)$
$i = i + 1$	$i_3 = i_2 + 1$	$i_3 = i_2 + 1$
until $i>10$	until $i_3>10$	until $i_3>10$