COMP 512
Rice University
Spring 2015

Translation Out of SSA Form

Benoit Boissinot, Alain Darte, Benoit Dupont de Dinechin, Christophe Guillon, and Fabrice Rastello, "Revisiting Out-of-SSA Translation for Correctness, Code Quality, and Efficiency," CGO 2009.

Citation numbers refer to entries in the EaC2e bibliography.

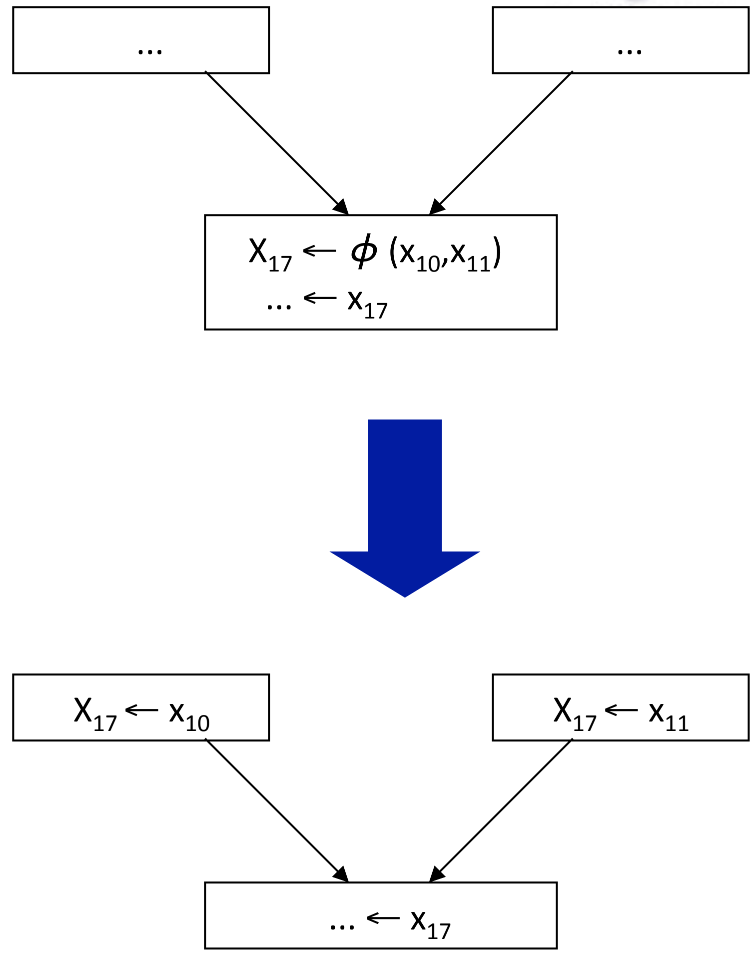
SSA Deconstruction

At some point, we need executable code

- Do machines implement ϕ-operations?
- Need to fix up the flow of values

Original idea [CFRWZ, 110]

- Replace ϕ-function with copies in predecessor blocks
- Simple algorithm
  - ♦ Works in most cases
- Adds lots of copies
  - ♦ Most of them coalesce away
  - ♦ Copy-coalescing is well understood
    - → *See, for example, Chaitin-Briggs GCRA* **[75,56]**



Translation Out of SSA Form

Two "classic" problems arise in SSA deconstruction

- Lost-copy problem
- Swap problem

In each case, simple copy insertion produces incorrect code

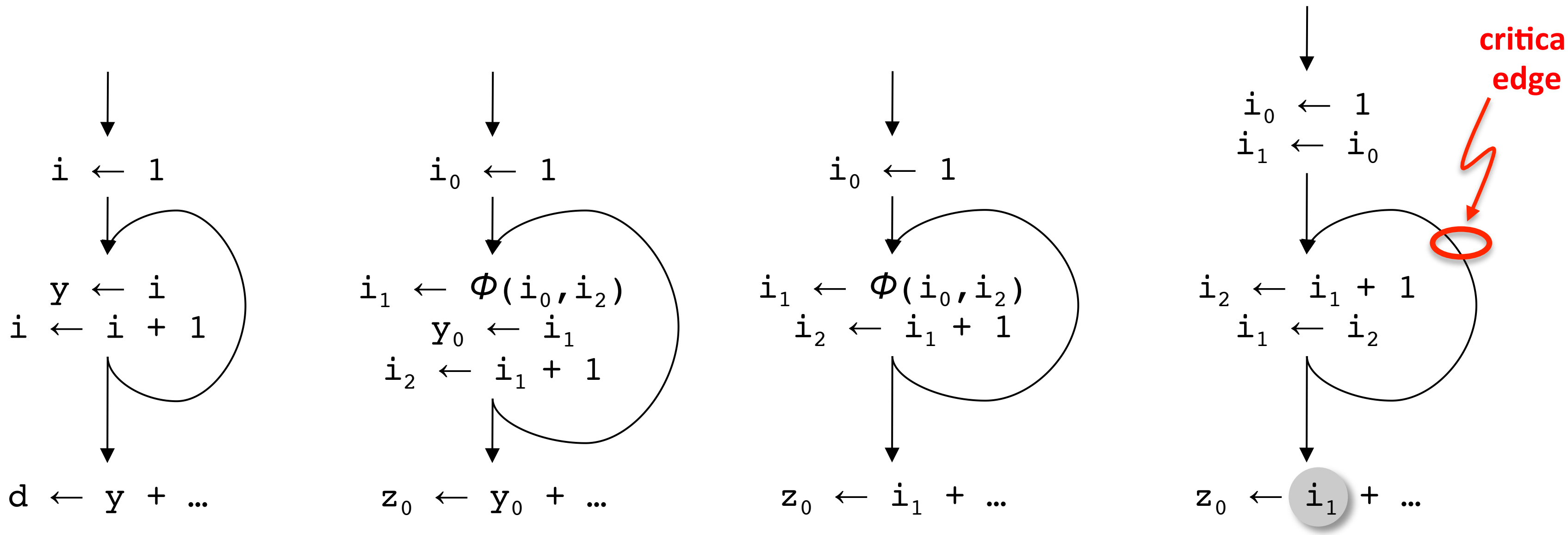
Critical Observation

- Both "problems" are caused by transformations that rewrite the code and move definitions and uses
- Some of the complication arises from the shift between the parallel semantics of the Φ–functions and the sequential semantics of copy

These problems were identified by Cliff Click and first published by Briggs et al. in 1997 [BCH&S, 50]. They presented ad-hoc ways of solving the problems.

This lecture is based on the work of Boissinot et al. and presents a more systematic approach to solving the problems inherent in translation out of SSA form. See "Revisiting Out-of-SSA Translation for Correctness, Code Quality, and Efficiency," by Benoit Boissinot, Alain Darte, Benoit Dupont de Dinechin, Christophe Guillon, and Fabrice Rastello in CGO 2009.

The Lost Copy Problem



Original code | **In SSA form** | **With copies folded** | **Copies naïvely inserted**

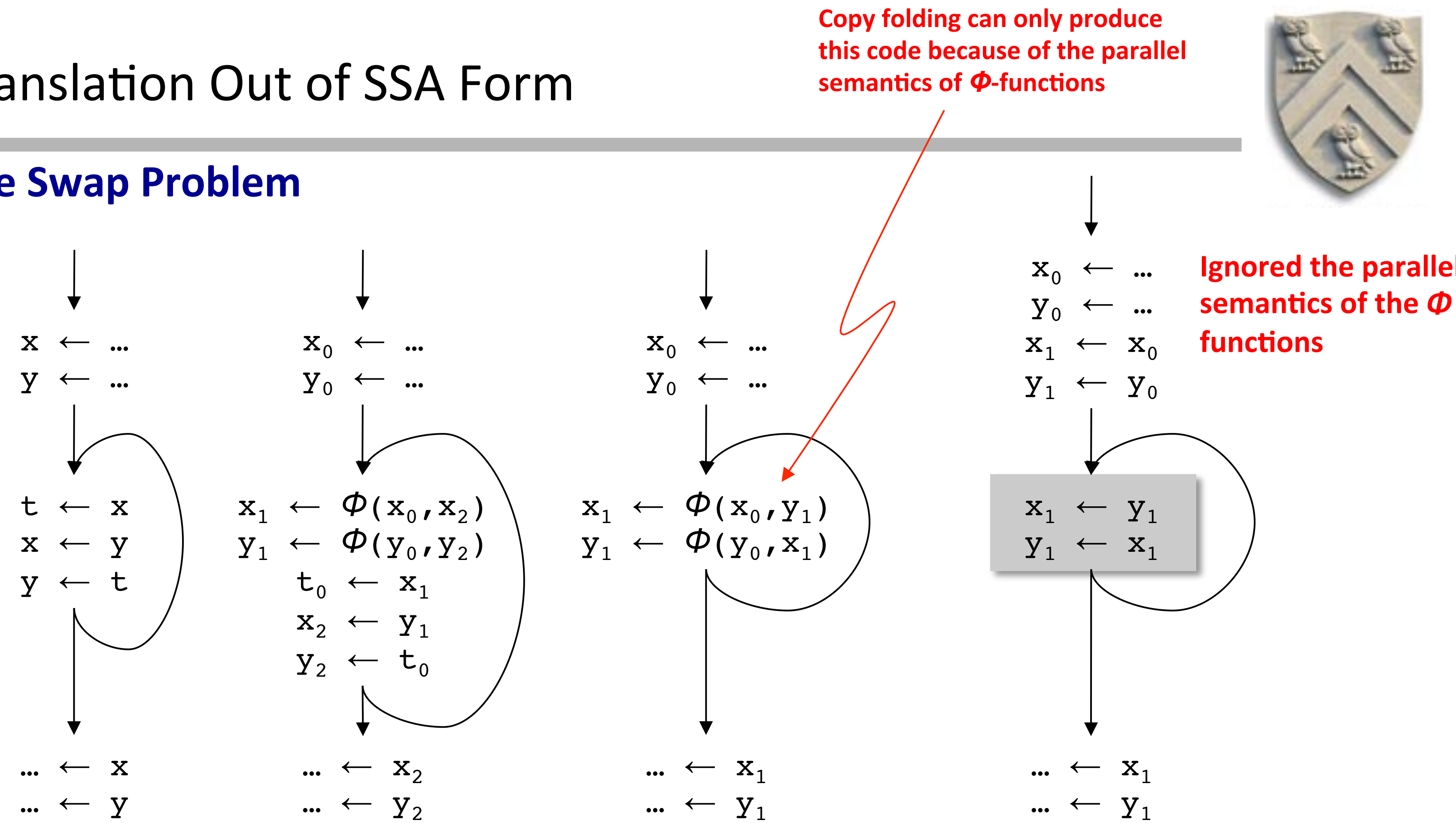
> Copy folding is a simple transformation that creates the problem. Other transformations can have the same effect.

> The assignment to z now receives the wrong value.

To fix this problem, the compiler needs to create a temporary name to hold the penultimate value of i.

Translation Out of SSA Form

The Swap Problem



Original code | **In SSA Form** | **Copies folded** | **Copies naïvely inserted**

Code is incorrect

This problem arises when a Φ-function argument is defined by a Φ-function in the same block. To generate correct code, the compiler will need to insert one or more additonal copy operations and temporary names. *

Translation Out of SSA Form

The Big Picture

- Swap problem & lost-copy problem arise from messing with *Φ*-function parameters
  - ♦ Renaming *Φ*-function parameters, moving them, …
  - ♦ Underlying issue is the parallel semantics of *Φ*-function evaluation
- One way to simplify out-of-**SSA** translation is to separate the parallelism from the *Φ*-functions and tackle it directly
  - ♦ Convert to a form of **SSA** where eliminating the subscripts produces correct code
  - ♦ We call that form "Conventional **SSA**" or **CSSA**

Translation Out of SSA Form

The Big Picture

- Insert parallel copies to convert **SSA** to conventional **SSA**
  - ♦ In **CSSA**, we can just drop the subscripts on **SSA** name
  - ♦ Introduces a new set of names
- Rename out of **CSSA** by replacing introduced names
- Eliminate *Φ* functions as in original paper
  - ♦ Insert copies at end of the predecessor blocks
- Sequentialize parallel copies (*may introduce new temporaries*)
- Aggressive copy coalescing to remove copies
  - ♦ Can coalesce copies before renaming or after we are done
  - ♦ Coalescing parallel copies requires some care
  - ♦ May be easier, and clearer, to coalesce after sequentialization

Moves the issues created by parallel *Φ* function semantics back into pred. blocks (in parallel copies).

e.g., Budimlic et. al. PLDI 2002, or unrestricted coalescing [75,56]

Sketch of ideas from "*Revisiting Out-of-SSA Translation for Correctness, Efficiency, and Speed*", Boissinot, Darte, de Dinechin, Guillon, and Rastello, *Proceedings of CGO 2009.*

The Individual Steps

Convert SSA to CSSA

For a ϕ-function $a_0 \leftarrow \phi(a_1, a_2, \ldots, a_n)$:

1. Insert a parallel copy $a_1' \leftarrow a_1$ at <u>the end</u> of the block corresponding to a_1
2. Replace $a_0 \leftarrow \phi(a_1, a_2, \ldots, a_n)$ with $a_0' \leftarrow \phi(a_1', a_2', \ldots, a_n')$
3. Insert a parallel copy $a_0 \leftarrow a_0'$ after the ϕ-function

Rename Out Of CSSA

$\forall$ primed name, i_j', drop the subscripts and replace each i' with a new name

Remove Φ Functions

- Replace Φ-functions with copies in predecessor blocks as in CFRWZ paper
  - Use parallel copies for good measure

Sequentialize The Parallel Copies

- Build a dependence graph and break cycles with a new name

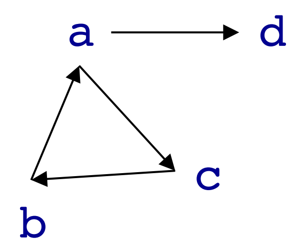
Copies inserted for different Φ-functions in the same block form a “parallel copy group”.

Sequentializing Parallel Copies

A parallel copy group forms a graph

$a \leftarrow_4 b$; $b \leftarrow_4 c$; $c \leftarrow_4 a$; $d \leftarrow_4 a$

Parallel copies



Corresponding graph

Graph is either a tree or a cycle

- Schedule a tree, bottom up from the leaves
- Must break each cycle with an extra copy
  - ♦ May require a new name
  - ♦ Other copies may avoid the extra name
    - → *If possible, break on the value preserved in a non-cycle copy*

$d \leftarrow a$
$a \leftarrow b$
$b \leftarrow c$
$c \leftarrow d$

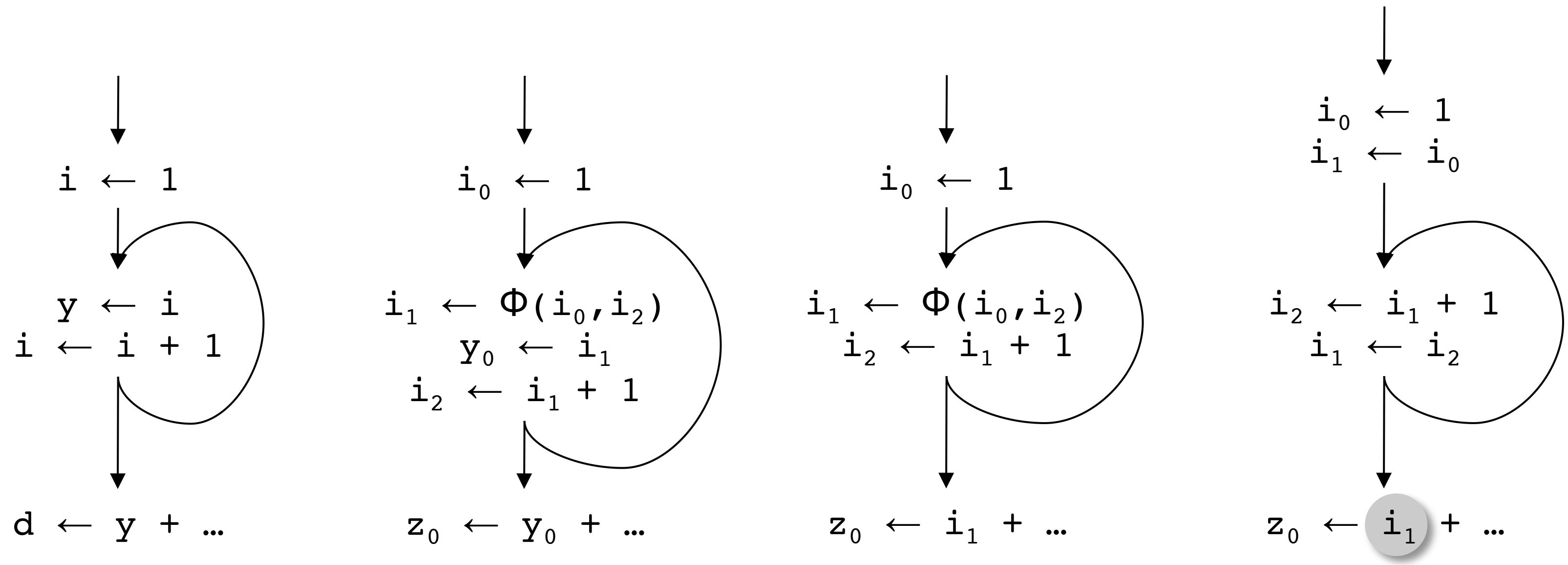
Serialized copies

Subscripts on the copies indicate parallel copy group membership.

Reminder from earlier slide

Translation Out of SSA Form

The Lost Copy Problem



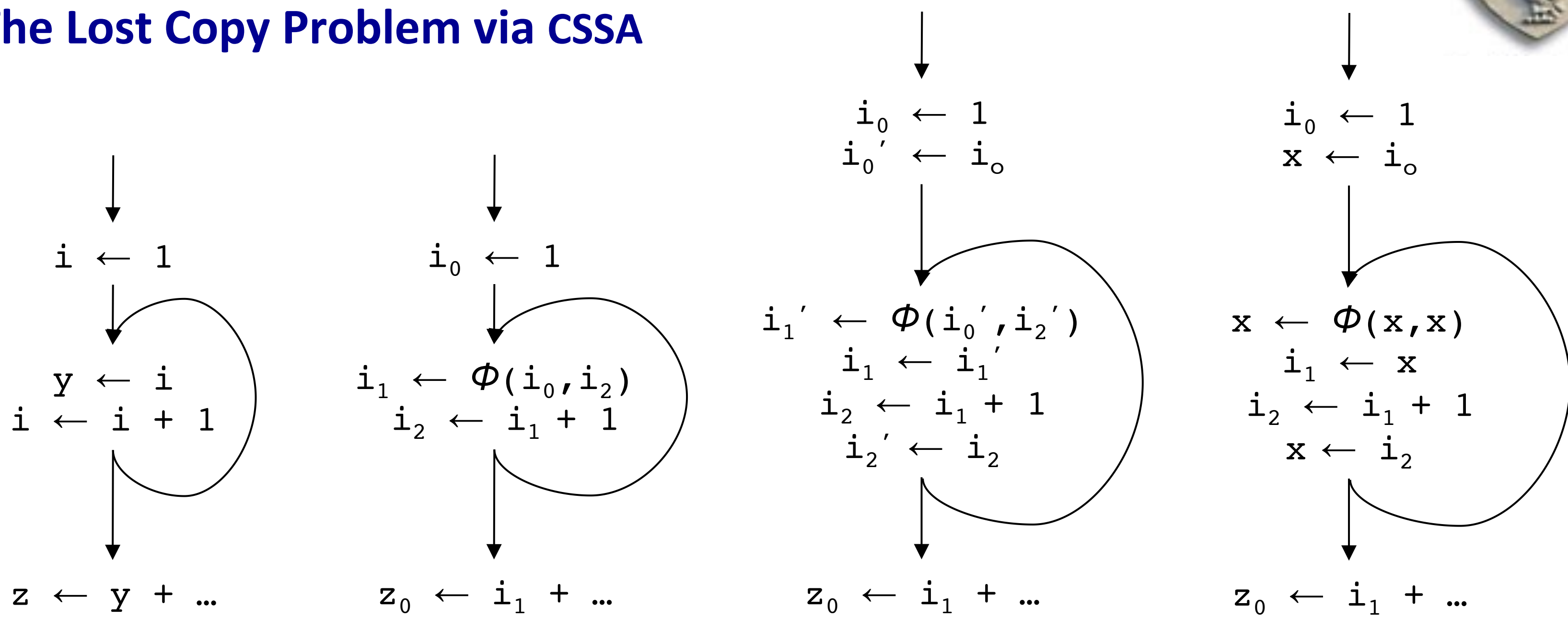
Original code | **In SSA form** | **With copies folded** | **Copies naïvely inserted**

Copy folding is a simple transformation that creates the problem. Other transformations can have the same effect.

The assignment to z now receives the wrong value.

To fix this problem, the compiler needs to create a temporary name to hold the penultimate value of i.

The Lost Copy Problem via CSSA

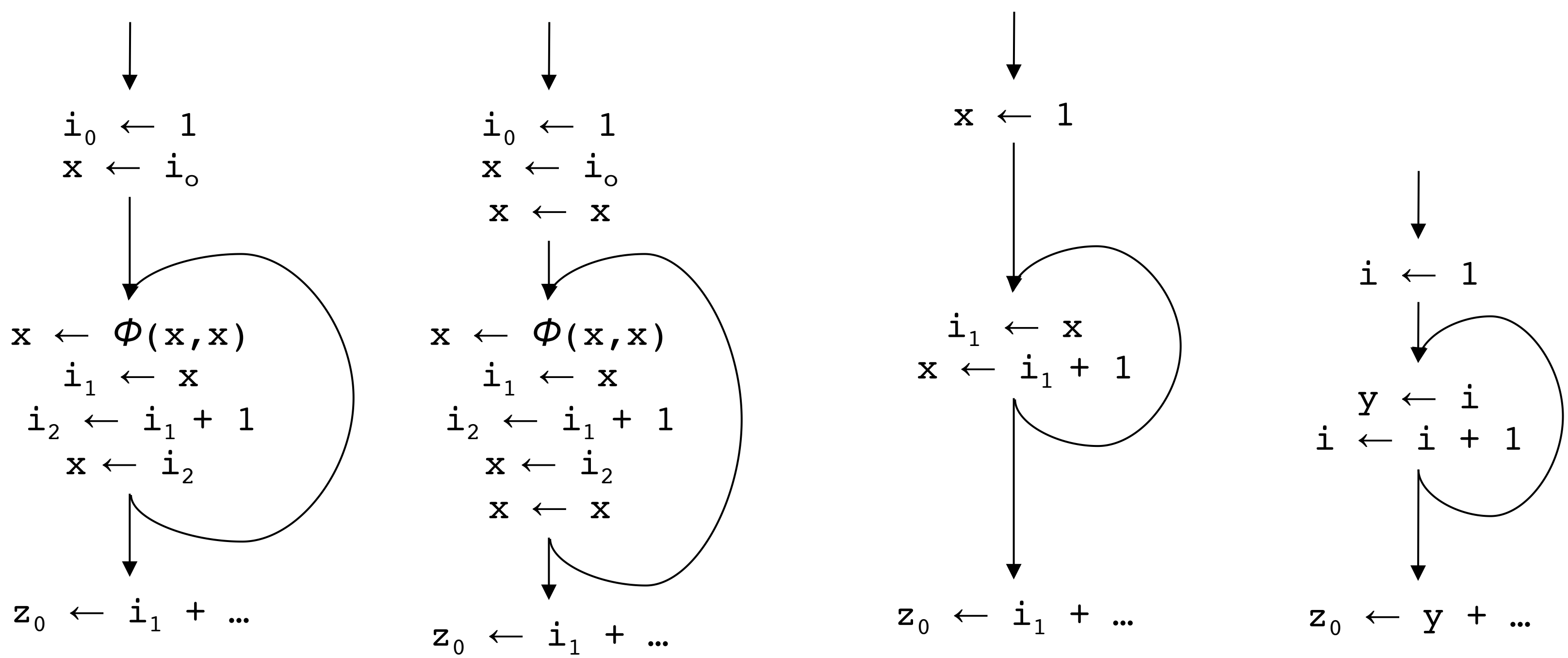


Original code | **In SSA with copies folded** | **Insert parallel copies to create CSSA** | **Rename out of CSSA**

With only one Φ, parallel is a singleton.

Each use of i' replaced with x. Use of i_1 to compute z_0 is correct.

The Lost Copy Problem via CSSA

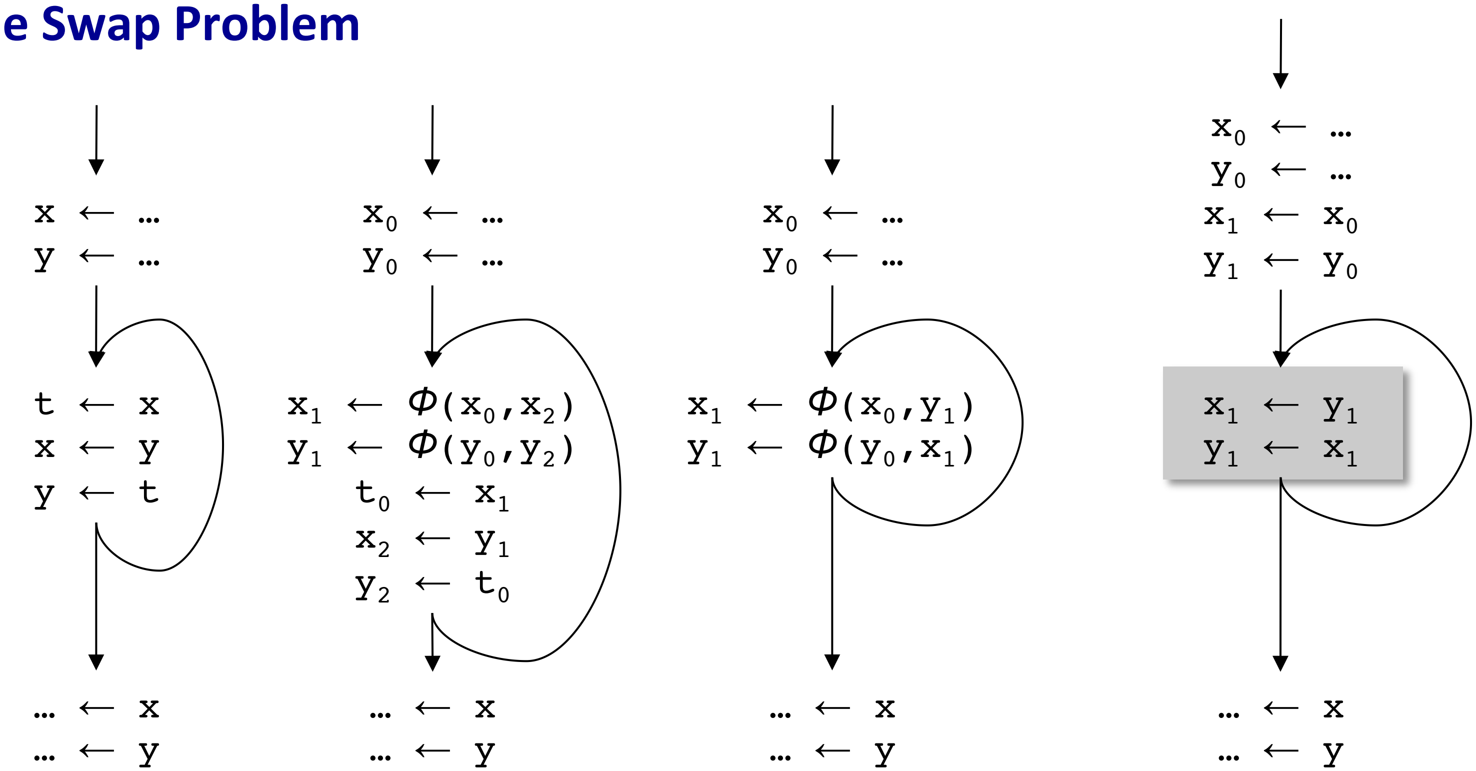


From previous slide | **Replace Φ's with copies** | **After coalescing** | **The original code**

Copies look rather stupid because it is a simple case, but it is *correct*.

Translation Out of SSA Form

The Swap Problem

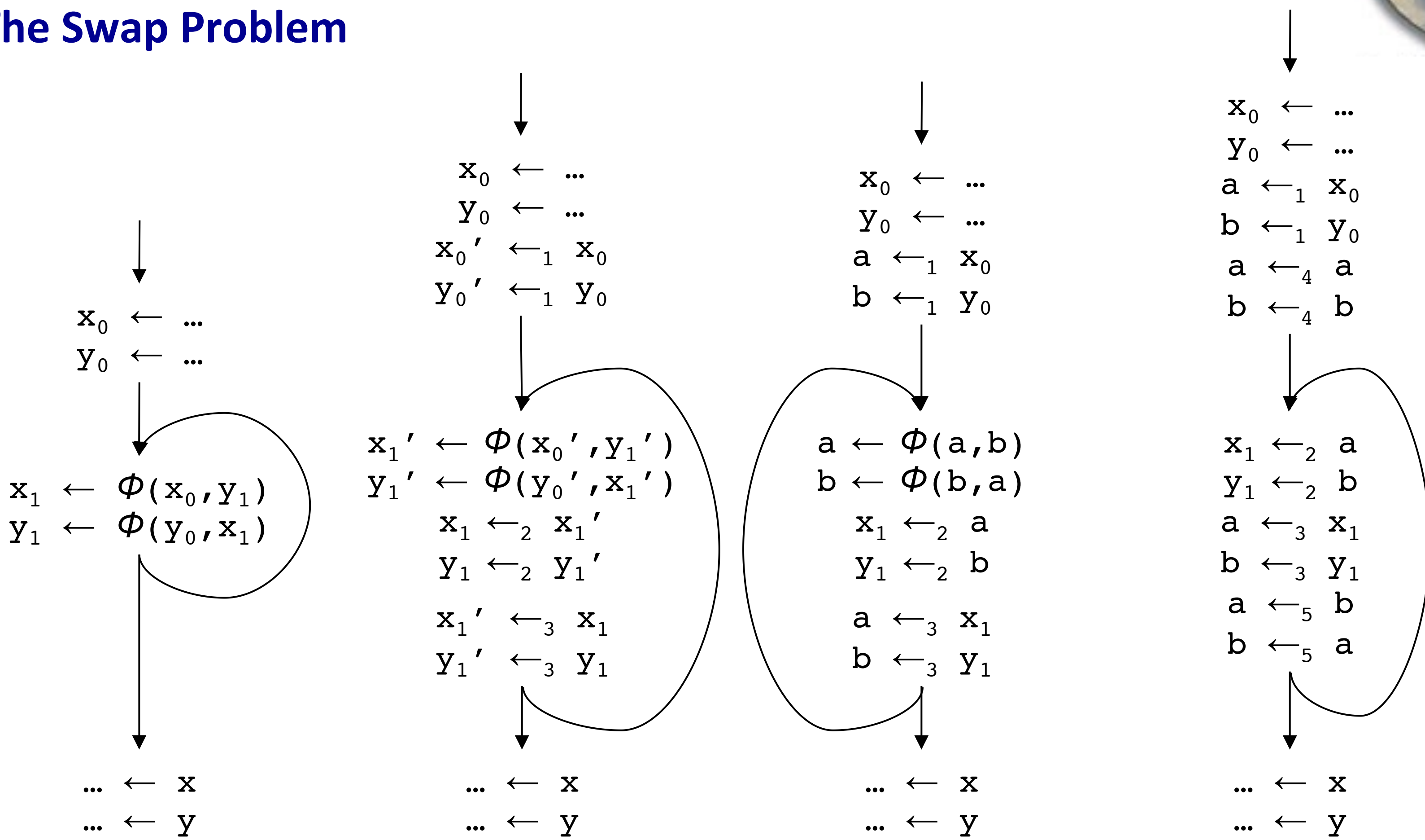


Original code | **In SSA Form** | **Copies folded** | **Copies naïvely inserted**

Code is incorrect

This problem arises when a Φ-function argument is defined by a Φ-function in the same block. To generate correct code, the compiler will need to insert one or more additonal copy operations and temporary names.

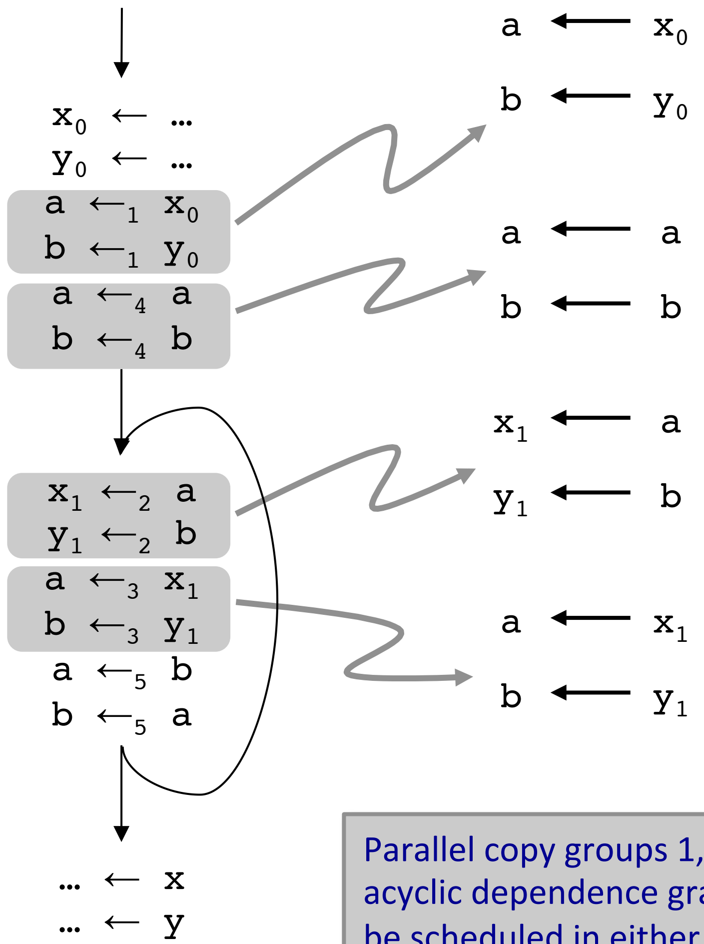
The Swap Problem



Starting Point (code in SSA form) | **Convert to CSSA** | **Rename primed variables** | **Eliminate Φ functions (used parallel copies)**

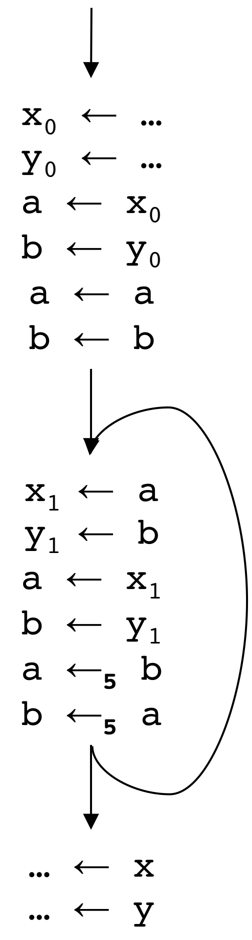
Subscript on ← indicates a parallel copy group

The Swap Problem



Parallel copy groups 1, 2, 3, & 4 have acyclic dependence graphs. They can be scheduled in either order.

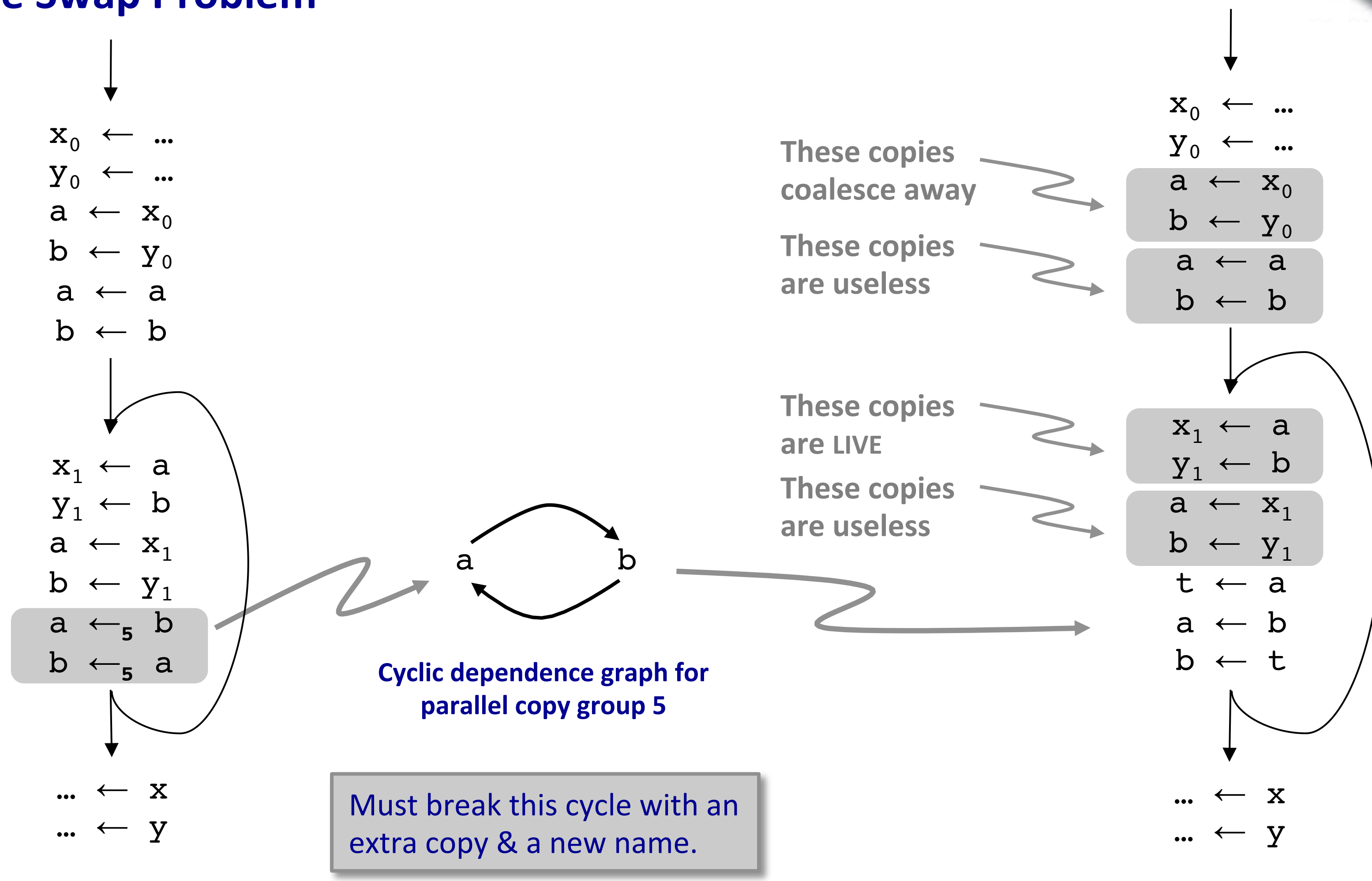
Result from previous slide



Groups 1, 2, 3, & 4 sequentialized

Translation Out of SSA Form

The Swap Problem



Groups 1, 2, 3, & 4 sequentialized

All copies are now sequential

Translation Out of SSA Form

To recap, the algorithm is

- Insert parallel copies to convert SSA to conventional SSA
  - ♦ In CSSA, we can just drop the subscripts on SSA name
  - ♦ Introduces a new set of names
- Rename out of CSSA by replacing introduced names
- Eliminate *Φ* functions by inserting parallel copies in prior blocks
  - ♦ May look redundant, but is necessary to handle the swap problem
- Sequentialize parallel copies (*may introduce new temporaries*)
- Aggressive copy coalescing to remove copies
  - ♦ Can coalesce copies before renaming or after we are done
  - ♦ Coalescing parallel copies requires some care
  - ♦ May be easier, and clearer, to coalesce after sequentialization

e.g., Chaitin-Briggs unrestricted coalescing or Budimlic et. al. PLDI 2002.

Using SSA Form in a Compiler

Need to translate source or IR into SSA form

- Algorithm from earlier lecture **[110,50]**
- **IR** needs to represent both Φ**–**functions and Φ**–**argument to **CFG** edge correspondance

Working with SSA Form

- The **SSA** name space makes some kinds of transformations harder
  - ♦ Code motion past Φ-functions is problematic
- **SSA** provides a useful sparse representation that can lead to efficiency
  - ♦ Wegman-Zadeck SCP and SCCP as examples **[347]**

Need to translate out of SSA form

- Algorithms from this lecture
- LLVM comes out of SSA in code generator (*selection, allocation, scheduling*)