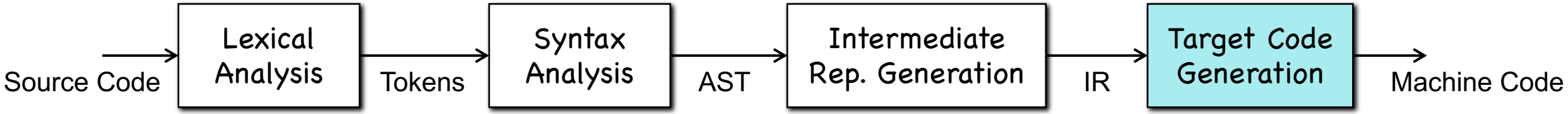
Made by Qingkai Shi
qingkaishi@nju.edu.cn

南京大学编译技术
及软件安全研究组

Supplementary Material
Translation Out of SSA Form

Target Code Generation



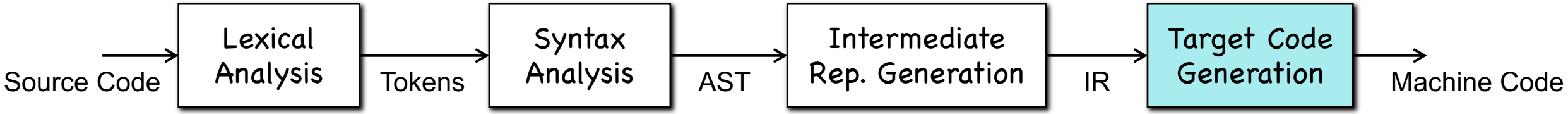

```
b = a + 5

c = b * 1

c = 2 + c

d = c
```

Three-Address Code

```
LD    R0,  a
ADD   R1,  R0,  5

ST    b,   R1
ST    c,   R1

ADD   R2,  R1,  2
ST    c,   R2

LD    R3,  c
ST    d,   R3
```

Machine Code

Target Code Generation



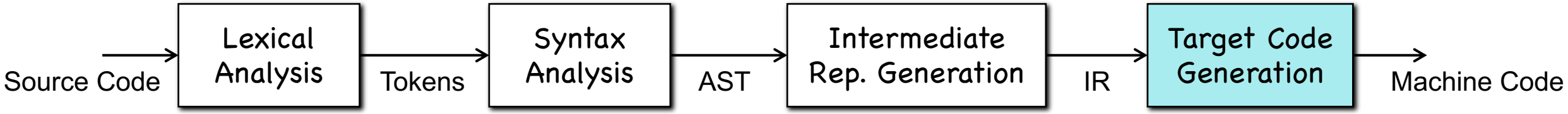

```
b = a + 5

c1 = b * 1

c2 = 2 + c1

d = c2
```

Three-Address Code (SSA)

```
LD    R0,  a
ADD   R1,  R0,  5

ST    b,   R1
ST    c1,  R1

ADD   R2,  R1,  2
ST    c2,  R2

LD    R3,  c2
ST    d,   R3
```

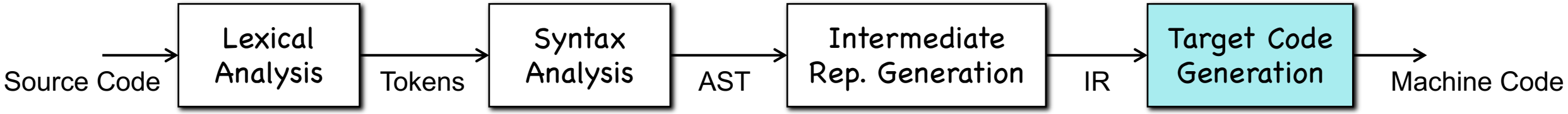
Machine Code

Recap: Why SSA?



Recap: Why SSA?

Recap: Why SSA?



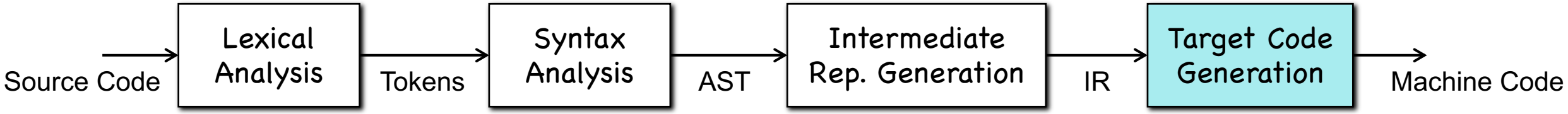
Recap: Why SSA?

```
1. a = b + c
2. b = a - d
3. c = b + c
4. d = a - d
```

⇨

```
1. a1 = b1 + c1
2. b2 = a1 - d1
3. c2 = b2 + c1
4. d1 = a1 - d1
```

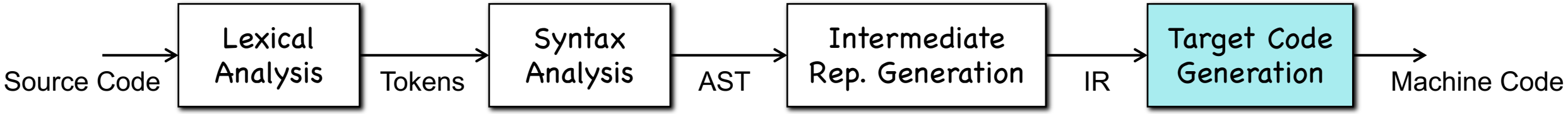
Recap: Translation to SSA



Recap: How can we generate IR in SSA form?

Hints: Two steps corresponding to two features of SSA.

Recap: Translation to SSA



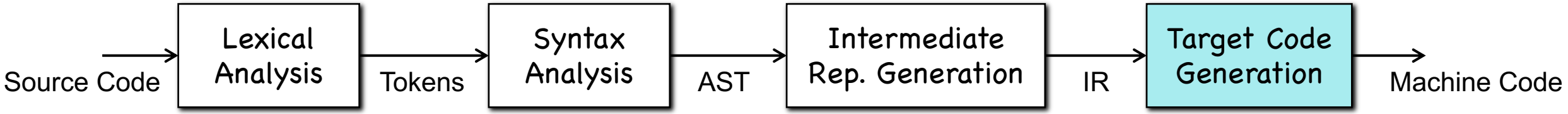
Recap: How can we generate IR in SSA form?

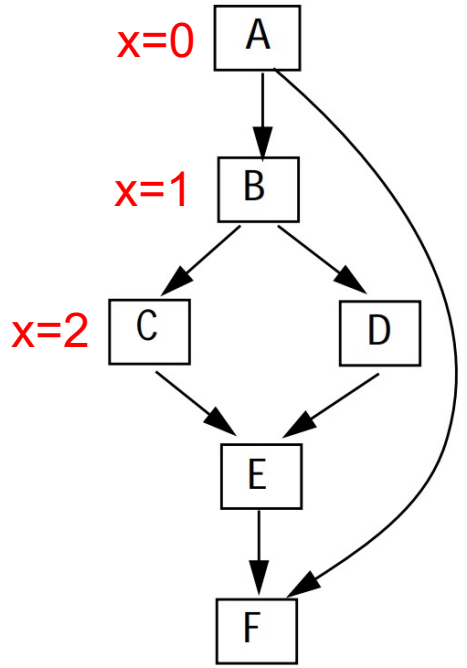
Hints: Two steps corresponding to two features of SSA.

Step 1: Inserting **ϕ** at the iterated dominance frontier to merge definitions.

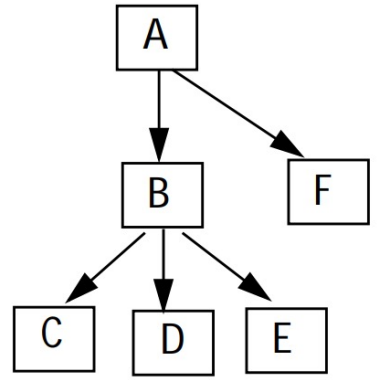
Step 2: Using subscripts to distinguish definitions of the same variable.

Recap: Translation to SSA



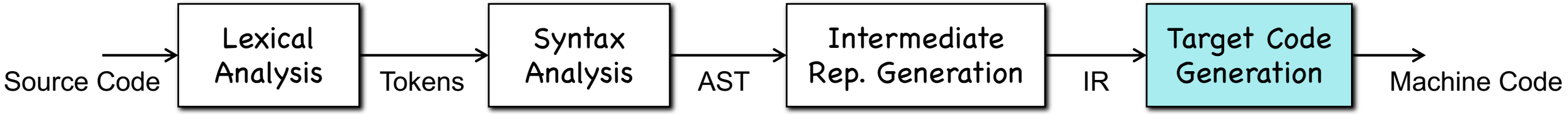


Control Flow Graph

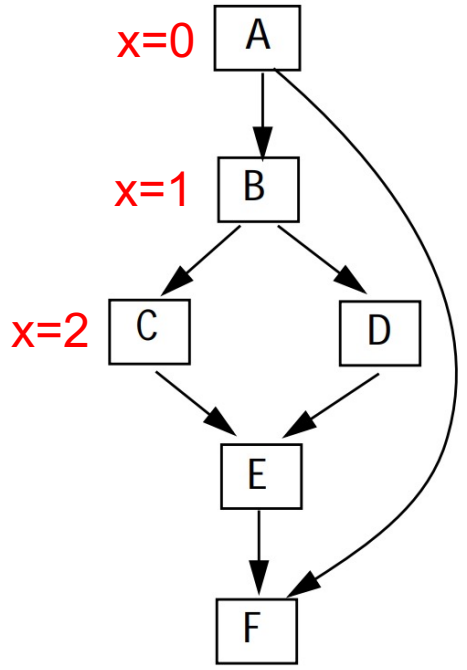


Dominator Tree

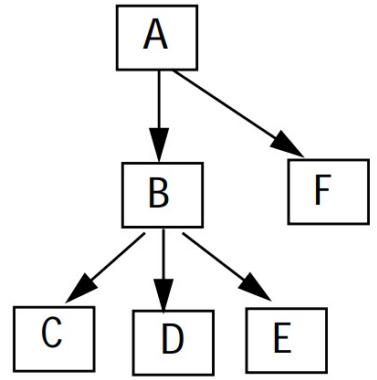
Recap: Translation to SSA



Block	A	B	C	D	E	F
Dominance Frontier	{ }	{F}	{E}	{E}	{F}	{ }

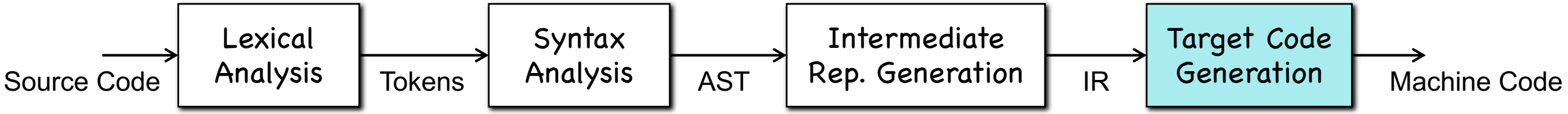


Control Flow Graph



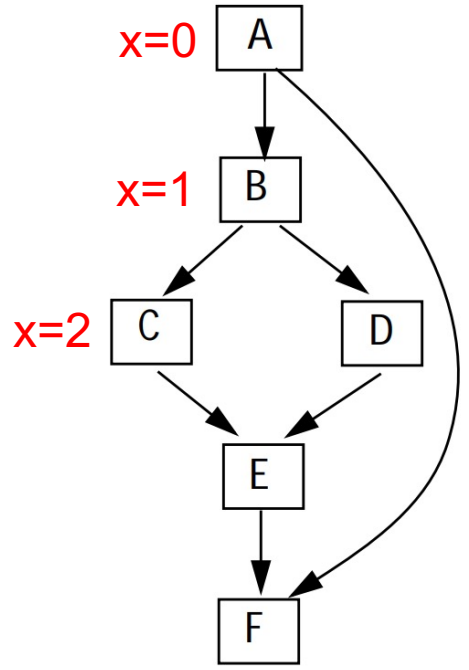
Dominator Tree

Recap: Translation to SSA

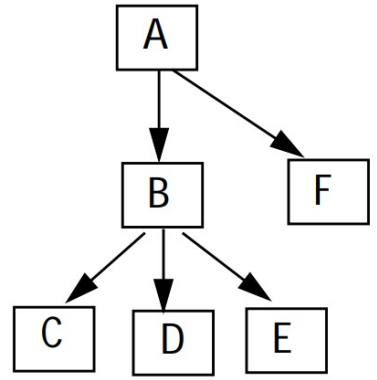


Block	A	B	C	D	E	F
Dominance Frontier	{ }	{F}	{E}	{E}	{F}	{ }

IDF({ A, B, C }) = { E, F }

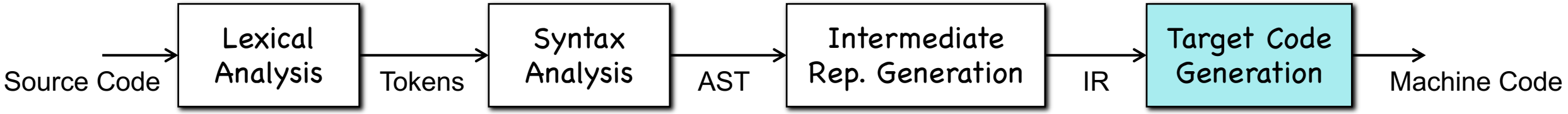


Control Flow Graph



Dominator Tree

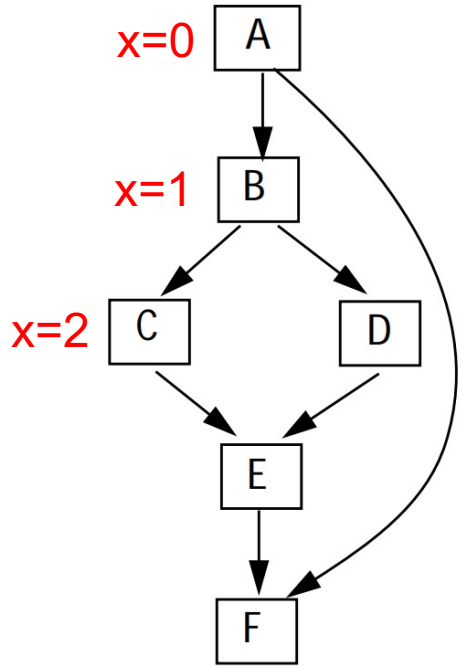
Recap: Translation to SSA



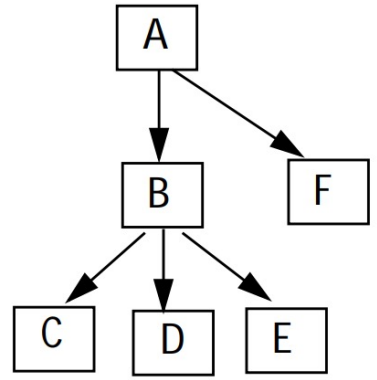
Block	A	B	C	D	E	F
Dominance Frontier	{ }	{F}	{E}	{E}	{F}	{ }

IDF({ A, B, C }) = { E, F }

Step 1: Insert **ϕ** at all blocks in IDF

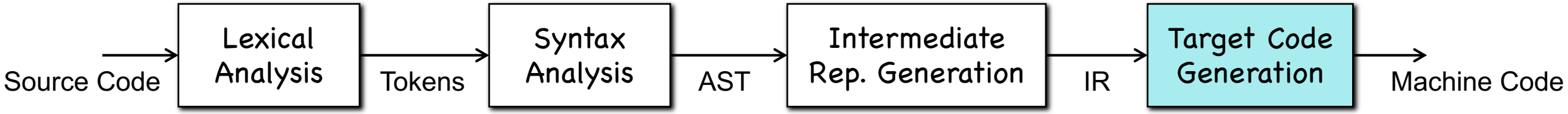


Control Flow Graph



Dominator Tree

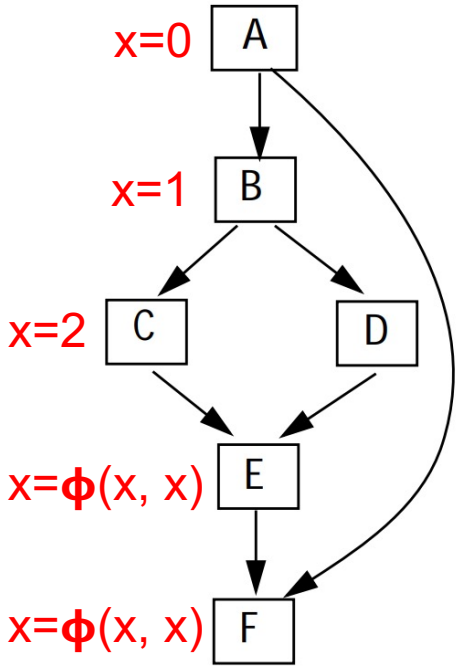
Recap: Translation to SSA



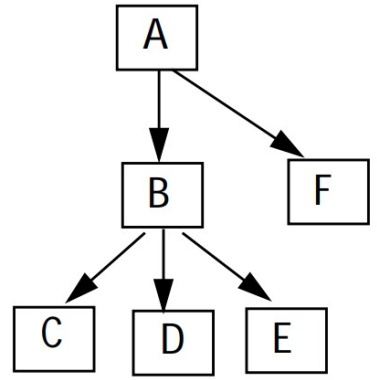
Block	A	B	C	D	E	F
Dominance Frontier	{ }	{F}	{E}	{E}	{F}	{ }

IDF({ A, B, C }) = { E, F }

Step 1: Insert **ϕ** at all blocks in IDF

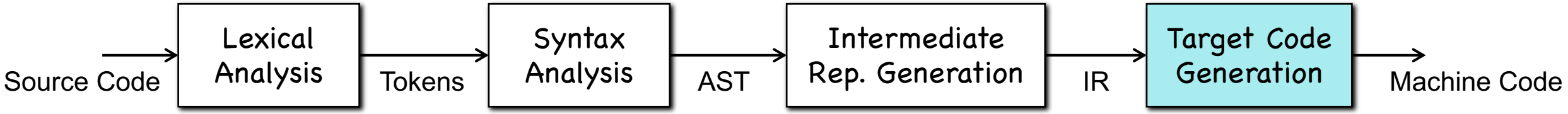


Control Flow Graph



Dominator Tree

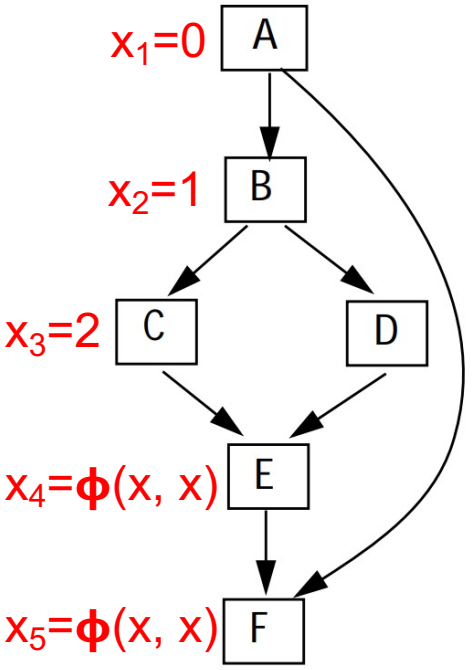
Recap: Translation to SSA



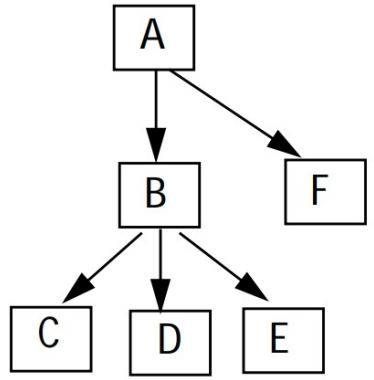
Block	A	B	C	D	E	F
Dominance Frontier	{ }	{F}	{E}	{E}	{F}	{ }

IDF({ A, B, C }) = { E, F }

Step 1: Insert **ϕ** at all blocks in IDF
Step 2: Rename definitions by adding subscripts

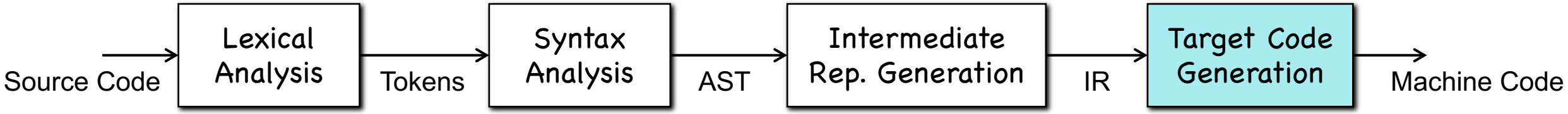


Control Flow Graph



Dominator Tree

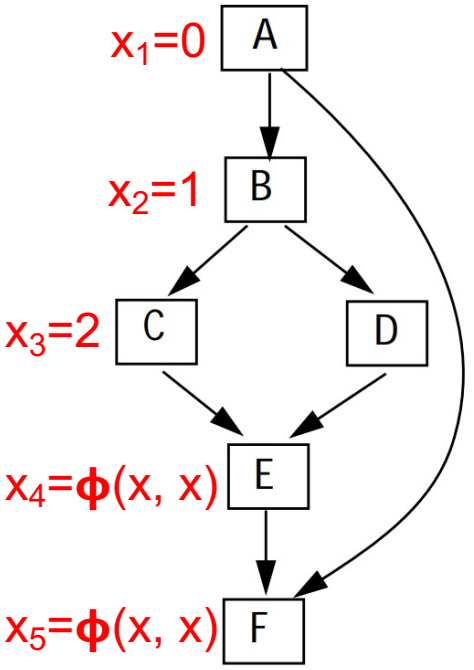
Recap: Translation to SSA



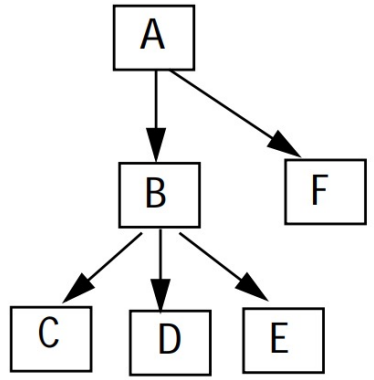
Block	A	B	C	D	E	F
Dominance Frontier	{ }	{F}	{E}	{E}	{F}	{ }

IDF({ A, B, C }) = { E, F }

Step 1: Insert **ϕ** at all blocks in IDF
Step 2: Rename definitions by adding subscripts
For each use of **x**, find proper def by climbing DT

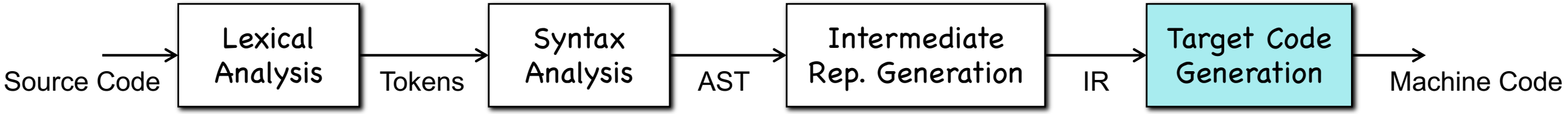


Control Flow Graph



Dominator Tree

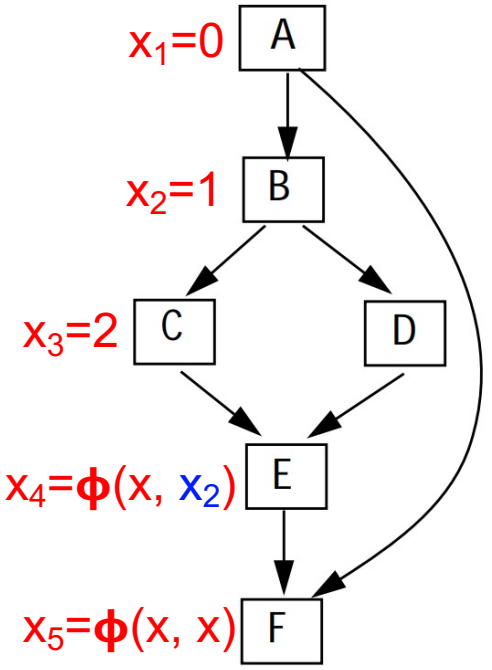
Recap: Translation to SSA



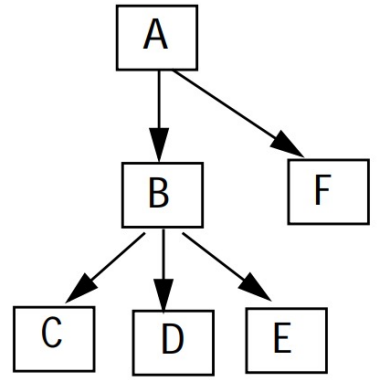
Block	A	B	C	D	E	F
Dominance Frontier	{ }	{F}	{E}	{E}	{F}	{ }

IDF({ A, B, C }) = { E, F }

Step 1: Insert **ϕ** at all blocks in IDF
Step 2: Rename definitions by adding subscripts
For each use of **x**, find proper def by climbing DT

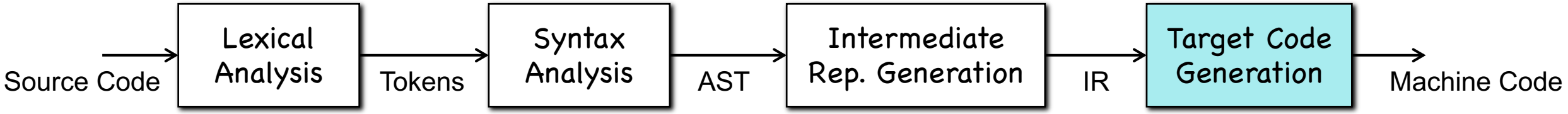


Control Flow Graph



Dominator Tree

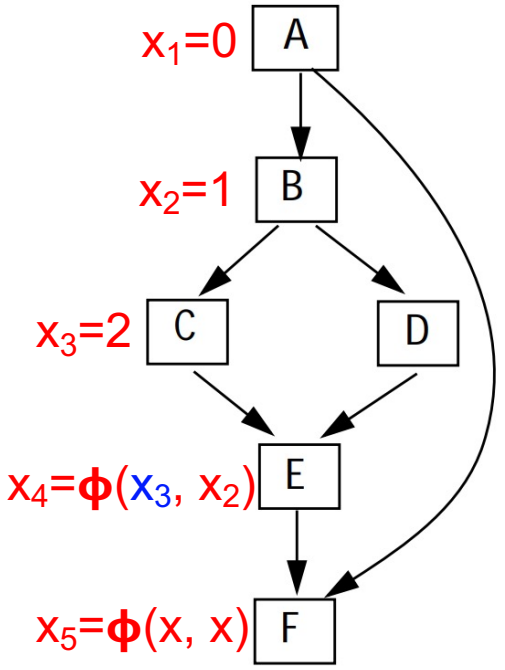
Recap: Translation to SSA



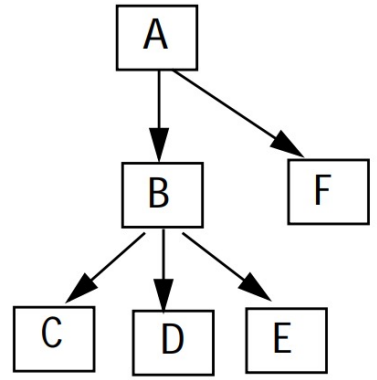
Block	A	B	C	D	E	F
Dominance Frontier	{ }	{F}	{E}	{E}	{F}	{ }

IDF({ A, B, C }) = { E, F }

Step 1: Insert **ϕ** at all blocks in IDF
Step 2: Rename definitions by adding subscripts
For each use of **x**, find proper def by climbing DT

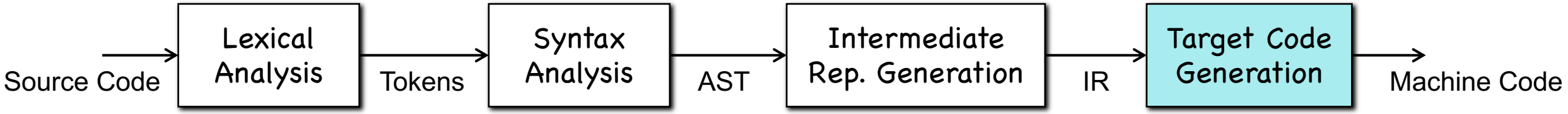


Control Flow Graph



Dominator Tree

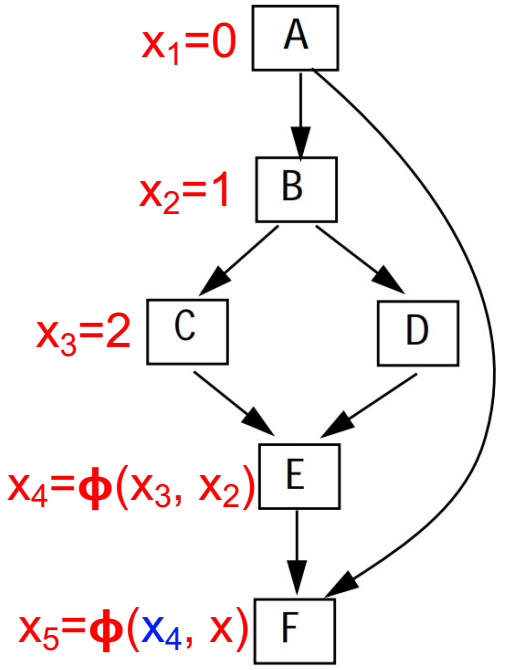
Recap: Translation to SSA



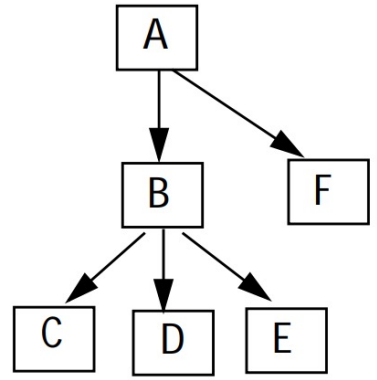
Block	A	B	C	D	E	F
Dominance Frontier	{ }	{F}	{E}	{E}	{F}	{ }

IDF({ A, B, C }) = { E, F }

Step 1: Insert **ϕ** at all blocks in IDF
Step 2: Rename definitions by adding subscripts
For each use of **x**, find proper def by climbing DT

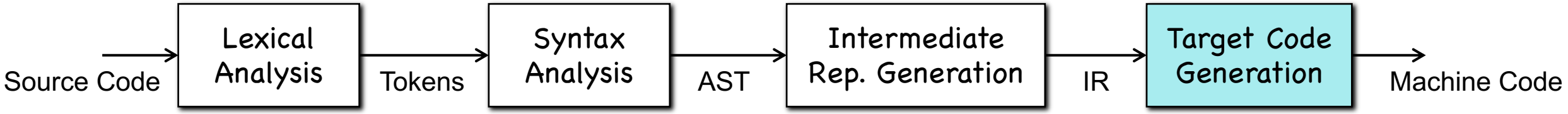


Control Flow Graph



Dominator Tree

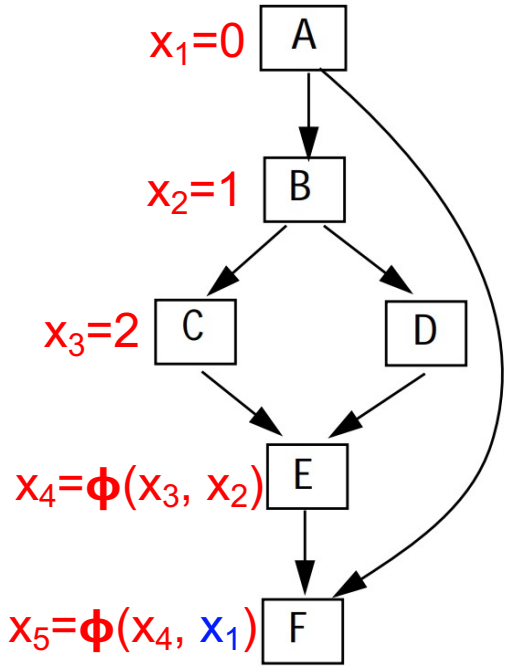
Recap: Translation to SSA



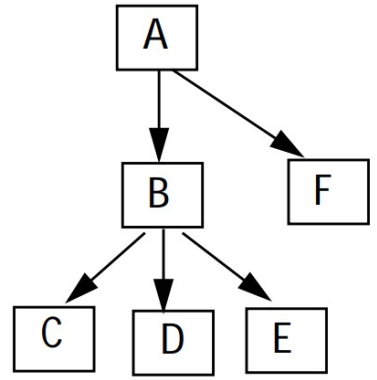
Block	A	B	C	D	E	F
Dominance Frontier	{ }	{F}	{E}	{E}	{F}	{ }

IDF({ A, B, C }) = { E, F }

Step 1: Insert **ϕ** at all blocks in IDF
Step 2: Rename definitions by adding subscripts
For each use of **x**, find proper def by climbing DT

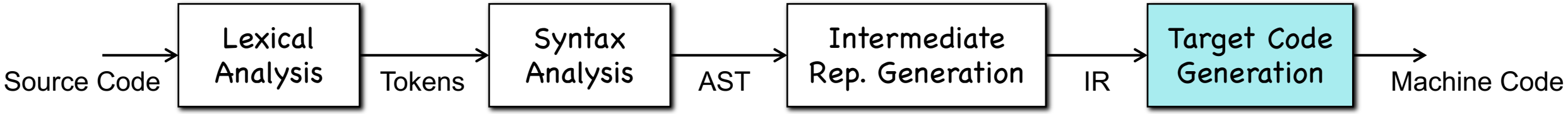


Control Flow Graph



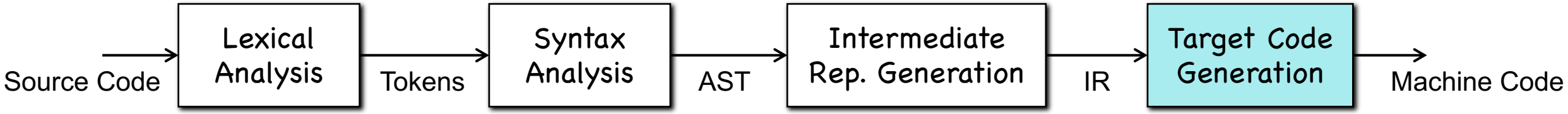
Dominator Tree

Target Code Generation



What challenges do we have to address to translate SSA IR (e.g., LLVM IR) to machine code?

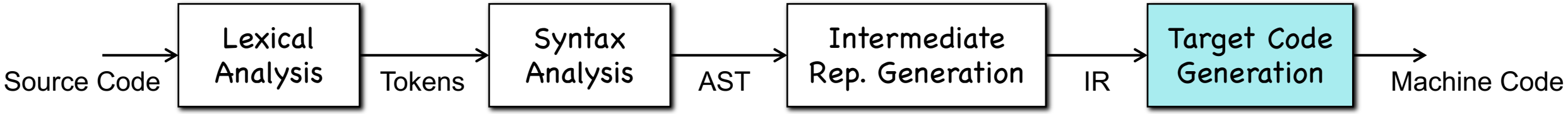
Target Code Generation



What challenges do we have to address to translate SSA IR (e.g., LLVM IR) to machine code?

The ϕ functions!

Target Code Generation

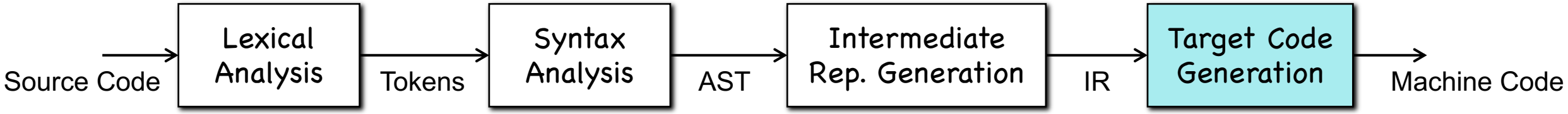


What challenges do we have to address to translate SSA IR (e.g., LLVM IR) to machine code?

The ϕ functions!

How can we translate (or eliminate) the ϕ functions?

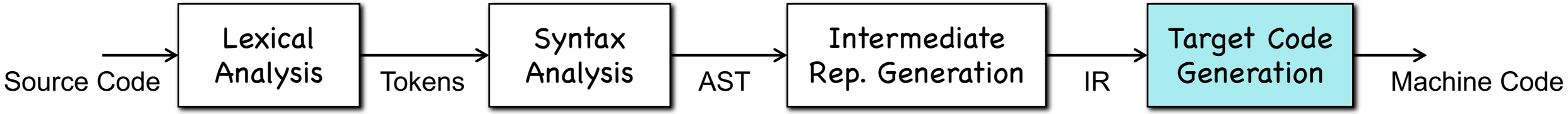
Target Code Generation




```
if ( flag ) x1 = -1; else x2 = 1;
x3 = ϕ (x1, x2);
y  = x3 *  a
```

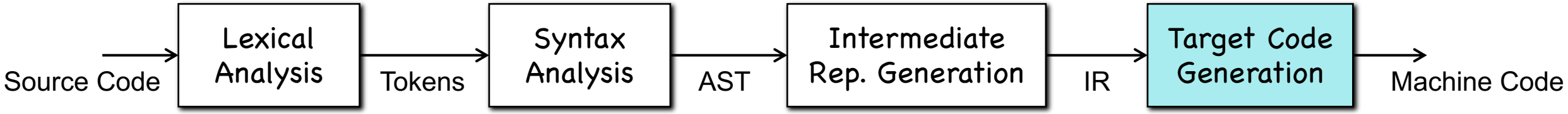
Try!

Target Code Generation



Solution 1: Since **ϕ** merge values of the same variable, let's replace variables merged by **ϕ** with the same variable.

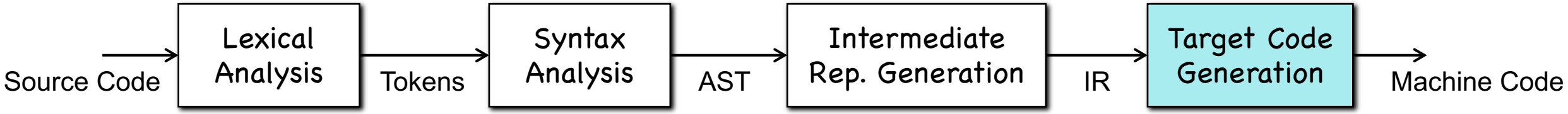
Target Code Generation



Solution 1: Since ϕ merge values of the same variable, let's replace variables merged by ϕ with the same variable.

```
if ( flag ) x = -1; else x = 1;
x3 = ϕ (x1, x2);
y = x * a
```

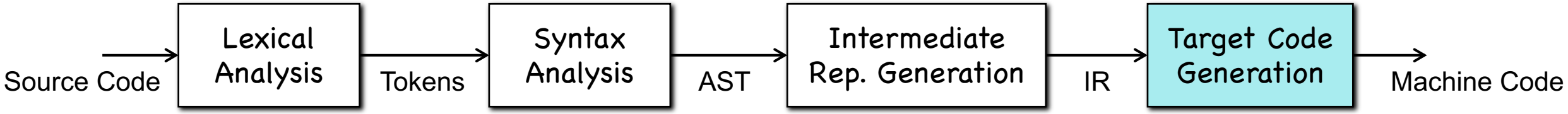
Target Code Generation



Solution 1: Since ϕ merge values of the same variable, let's replace variables merged by ϕ with the same variable.

```
if ( flag ) x = -1; else x = 1;
x3 = ϕ (x1, x2);
y = x * a
```

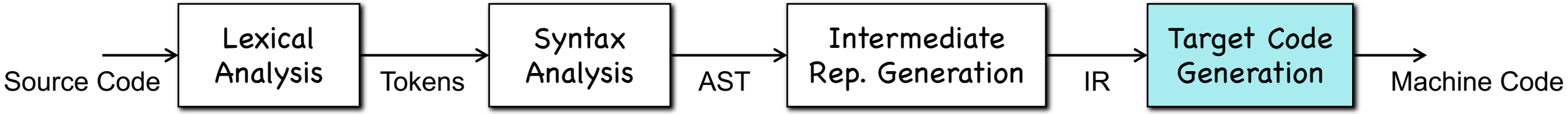
Target Code Generation



Solution 1: Since ϕ merge values of the same variable, let's replace variables merged by ϕ with the same variable.

```
if ( flag ) x = -1; else x = 1;
x3 = ϕ (x1, x2);
y = x * a
```

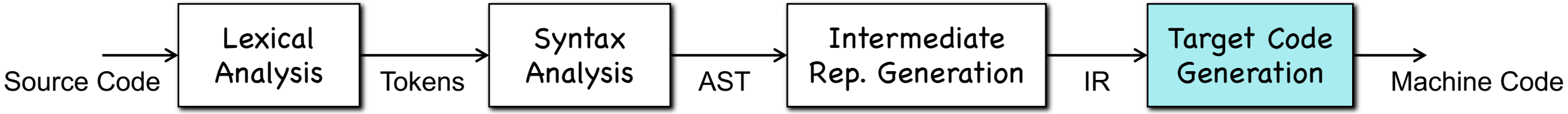
Target Code Generation



Solution 1: Since **ϕ** merge values of the same variable, let's replace variables merged by **ϕ** with the same variable.

```
x0 = 1;
do {
    x1 = ϕ(x0, x2);
    x2 = x1 + 1;
} while(…);
print(x1);
```

Target Code Generation



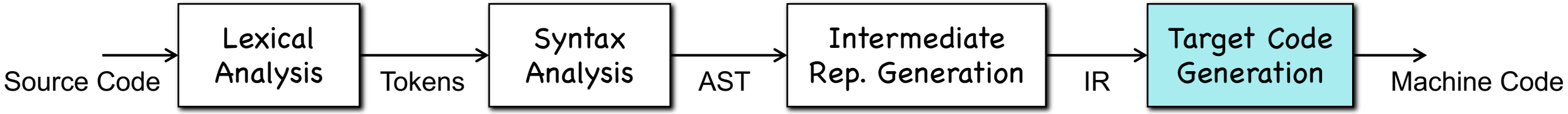
Solution 1: Since ϕ merge values of the same variable, let's replace variables merged by ϕ with the same variable.

```
x0 = 1;
do {
    x1 = ϕ(x0, x2);
    x2 = x1 + 1;
} while(…);
print(x1);
```

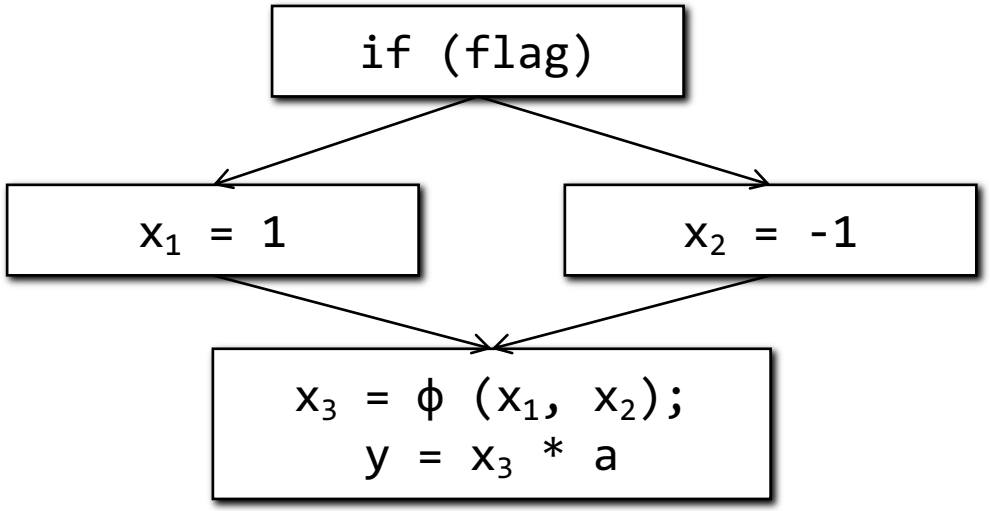
⇨

```
x = 1;
do {
    x1 = ϕ(x0, x2);
    x = x + 1;
} while(…);
print(x);
```

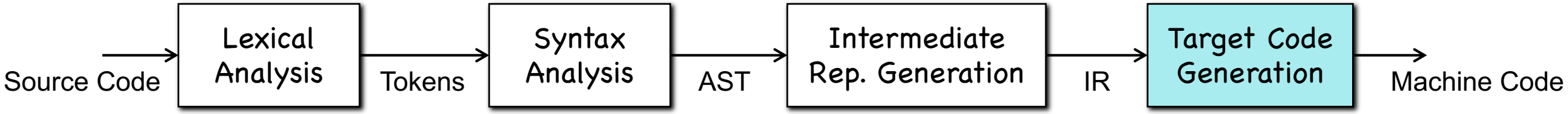
Target Code Generation



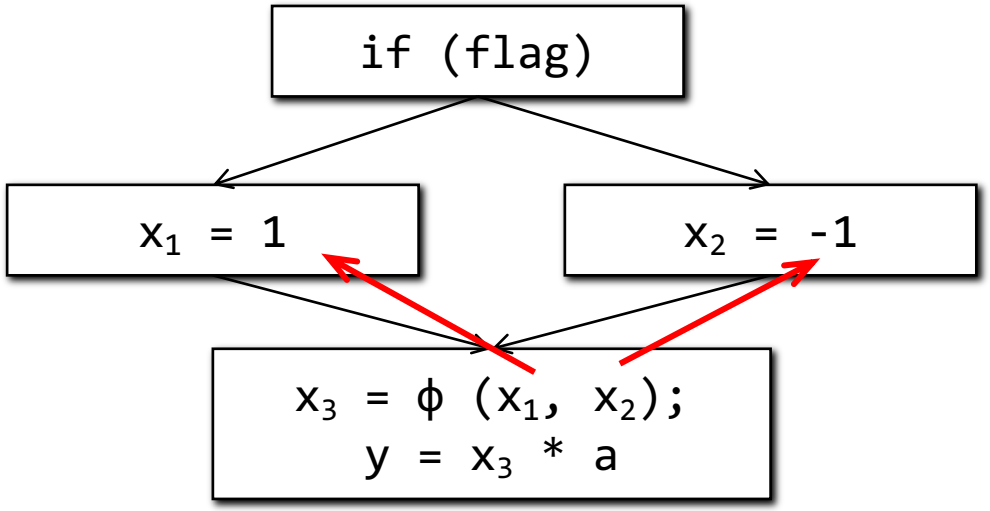
Solution 2: Replace ϕ function with copies in predecessor blocks.



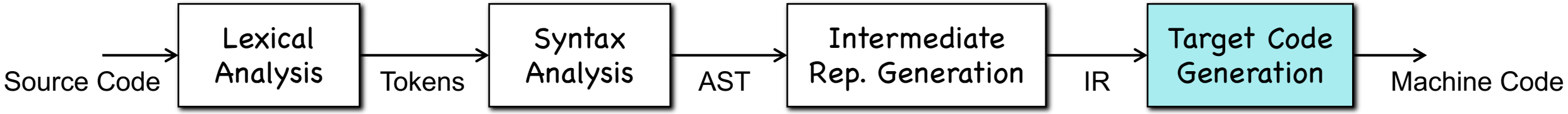
Target Code Generation



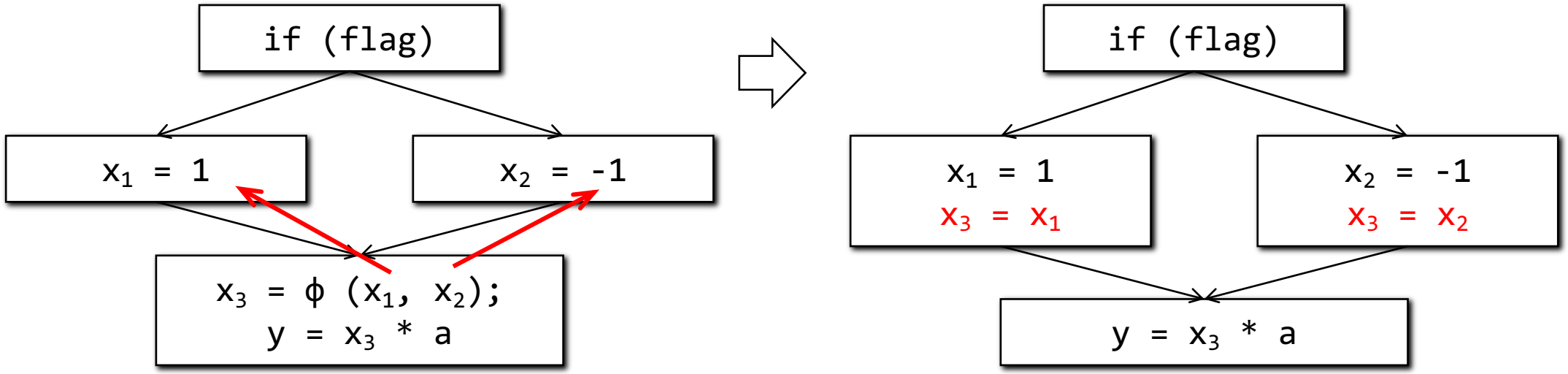
Solution 2: Replace ϕ function with copies in predecessor blocks.



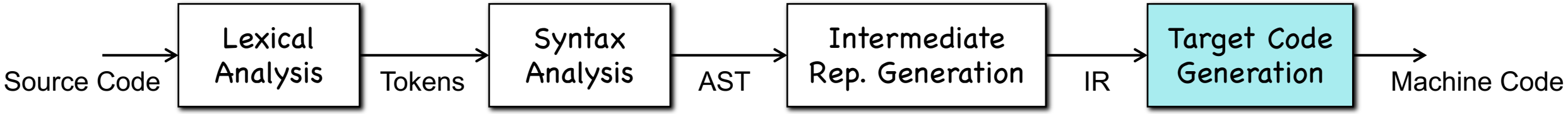
Target Code Generation



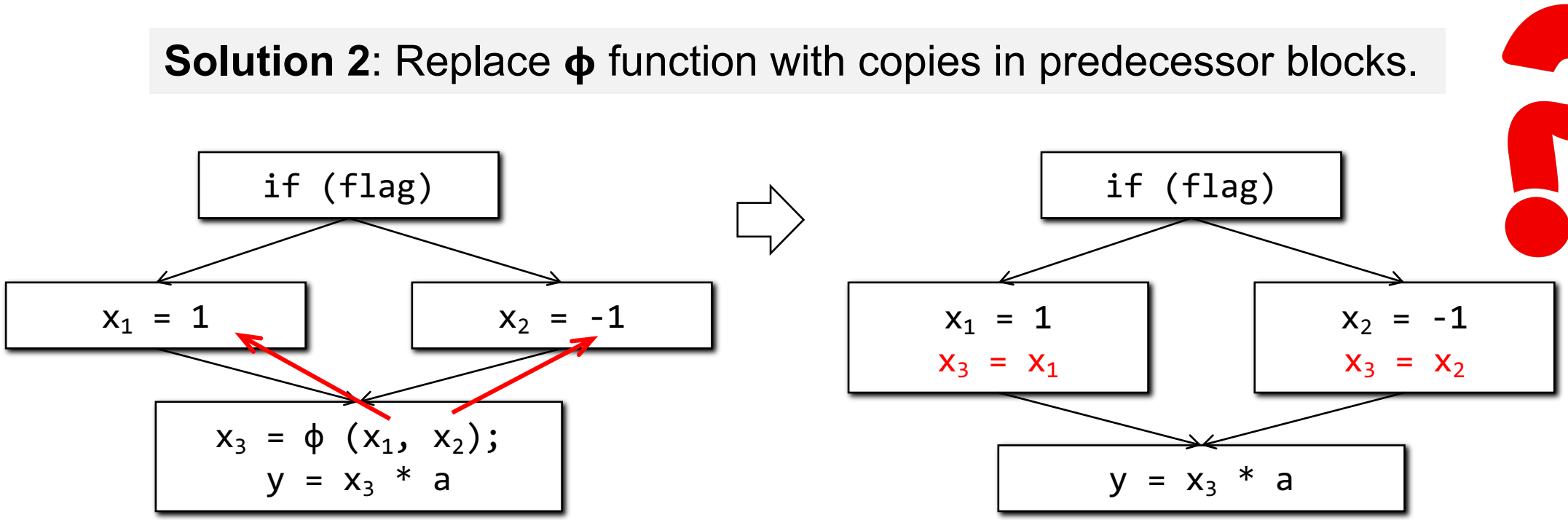
Solution 2: Replace ϕ function with copies in predecessor blocks.



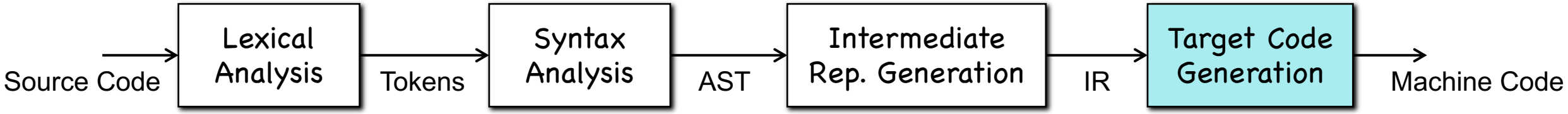
Target Code Generation



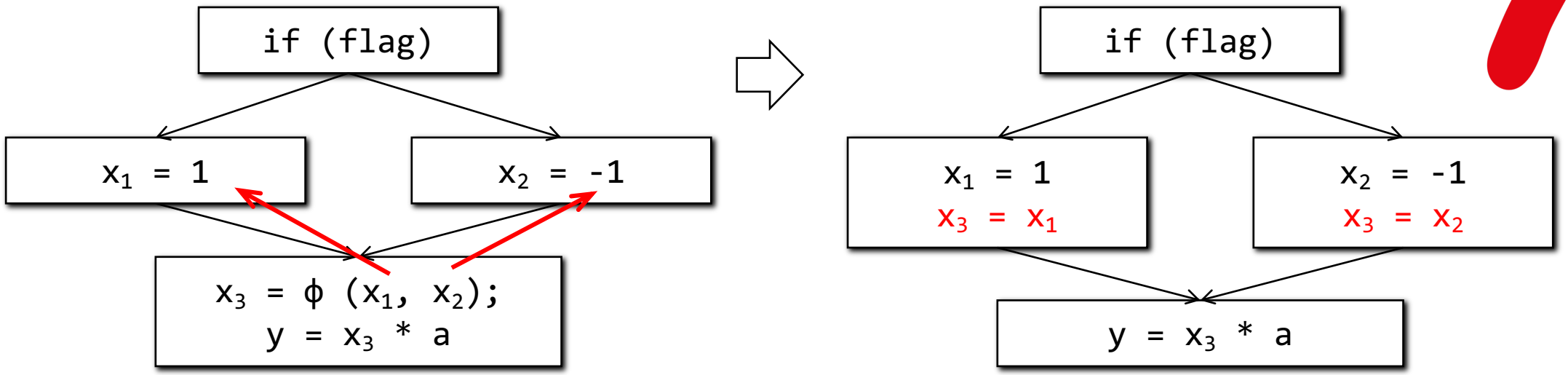
Solution 2: Replace ϕ function with copies in predecessor blocks.



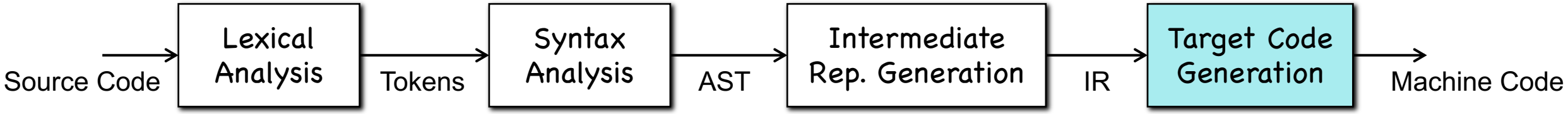
Target Code Generation



Solution 2: Replace ϕ function with copies in predecessor blocks.



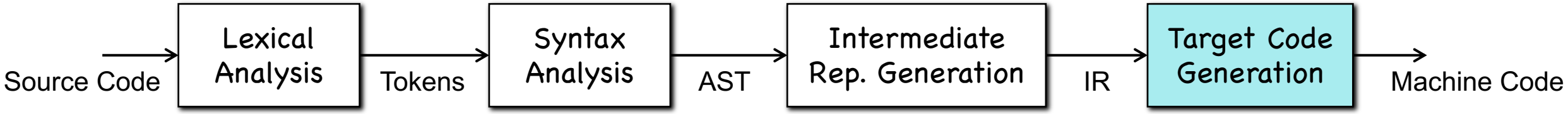
Target Code Generation



Solution 2: Replace **ϕ** function with copies in predecessor blocks.

```
x0 = 1;
do {
    x1 = ϕ(x0, x2);
    x2 = x1 + 1;

} while (…);
print(x1);
```

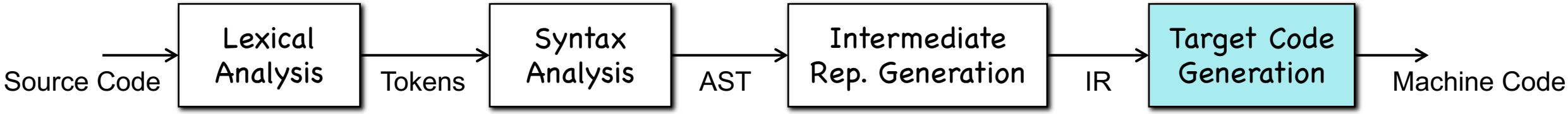
Target Code Generation



Solution 2: Replace ϕ function with copies in predecessor blocks.

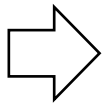
```
x0 = 1;
do {
    x1 = ϕ(x0, x2);
    x2 = x1 + 1;

} while (…);
print(x1);
```

Target Code Generation



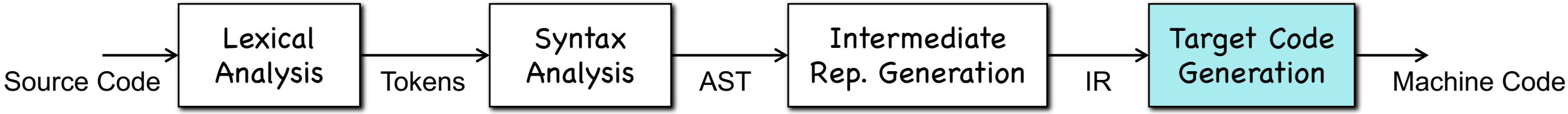
Solution 2: Replace **ϕ** function with copies in predecessor blocks.

```
x0 = 1;
do {
    x1 = ϕ(x0, x2);
    x2 = x1 + 1;

} while (…);
print(x1);
```

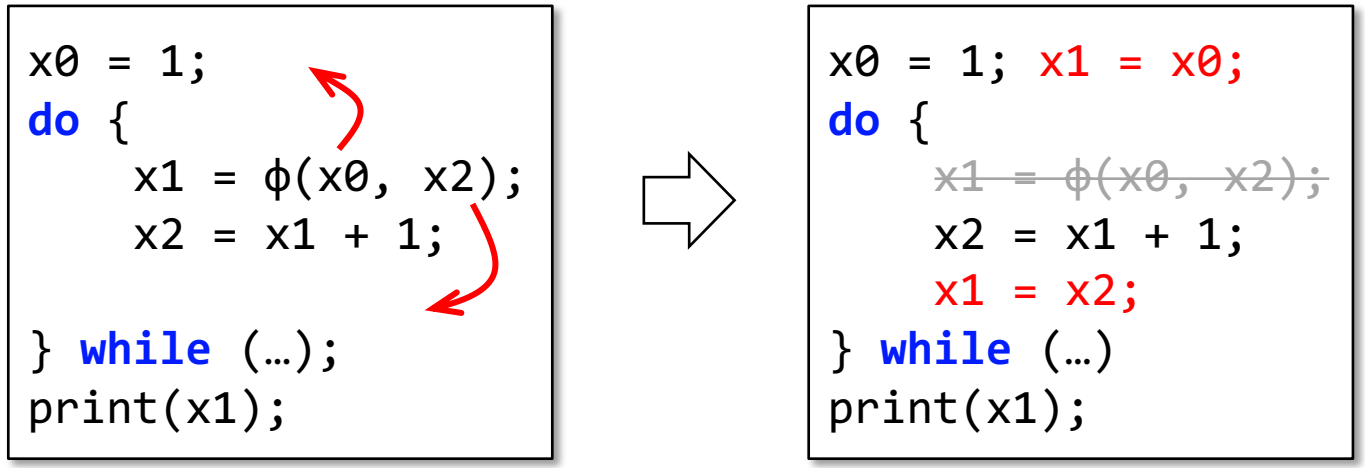


```
x0 = 1; x1 = x0;
do {
    x1 = ϕ(x0, x2);
    x2 = x1 + 1;
    x1 = x2;
} while (…)
print(x1);
```

Target Code Generation

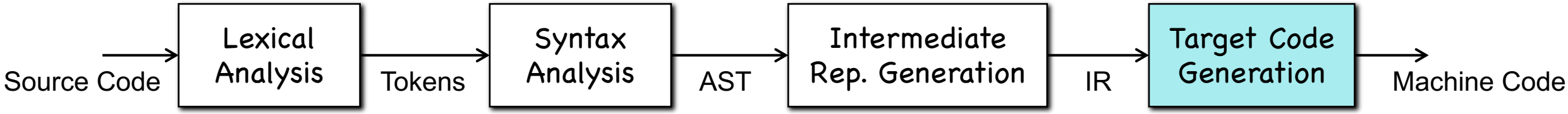


Solution 2: Replace ϕ function with copies in predecessor blocks.



The Lost Copy Problem!

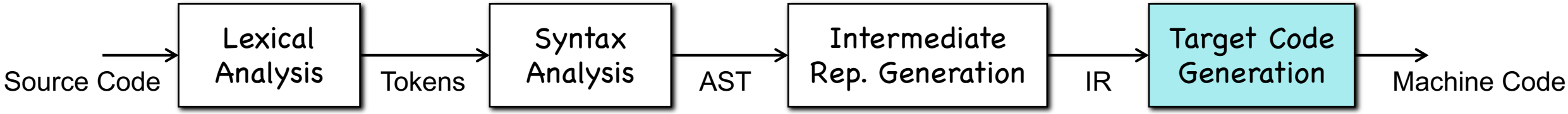
Target Code Generation



Solution 2: Replace **ϕ** function with copies in predecessor blocks.

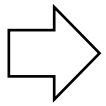
```
x0 = 1; y0 = 2;

do {
    x1 = ϕ(x0, y1);
    y1 = ϕ(y0, x1);
} while (…);
print(x1, y1);
```

Target Code Generation



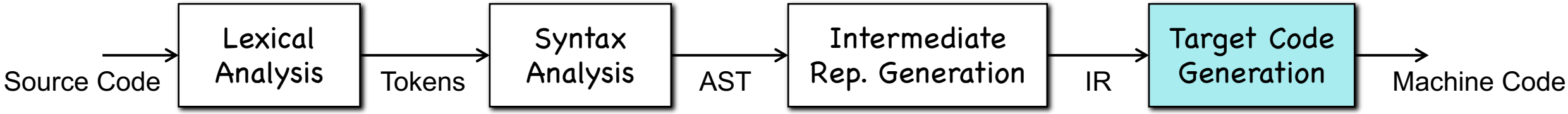
Solution 2: Replace **ϕ** function with copies in predecessor blocks.

```
x0 = 1; y0 = 2;

do {
    x1 = ϕ(x0, y1);
    y1 = ϕ(y0, x1);
} while (…);
print(x1, y1);
```



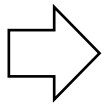
```
x0 = 1; y0 = 2;
x1 = x0; y1 = y0;
do {
    x1 = y1;
    y1 = x1;
} while (…);
print(x1, y1);
```

Target Code Generation



Solution 2: Replace **ϕ** function with copies in predecessor blocks.

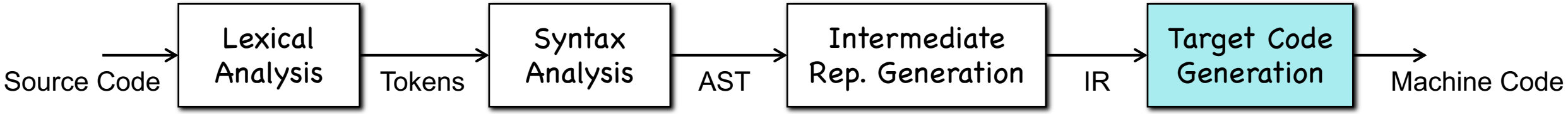
```
x0 = 1; y0 = 2;

do {
    x1 = ϕ(x0, y1);
    y1 = ϕ(y0, x1);
} while (…);
print(x1, y1);
```



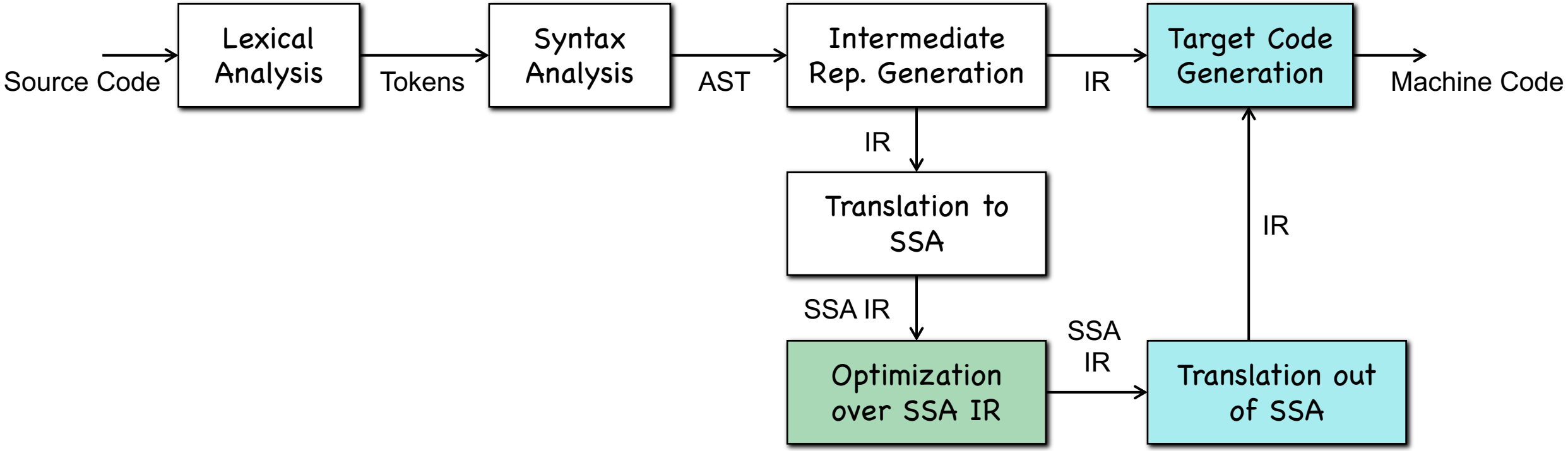
```
x0 = 1; y0 = 2;
x1 = x0; y1 = y0;
do {
    x1 = y1;
    y1 = x1;
} while (…);
print(x1, y1);
```

The Swap Problem!

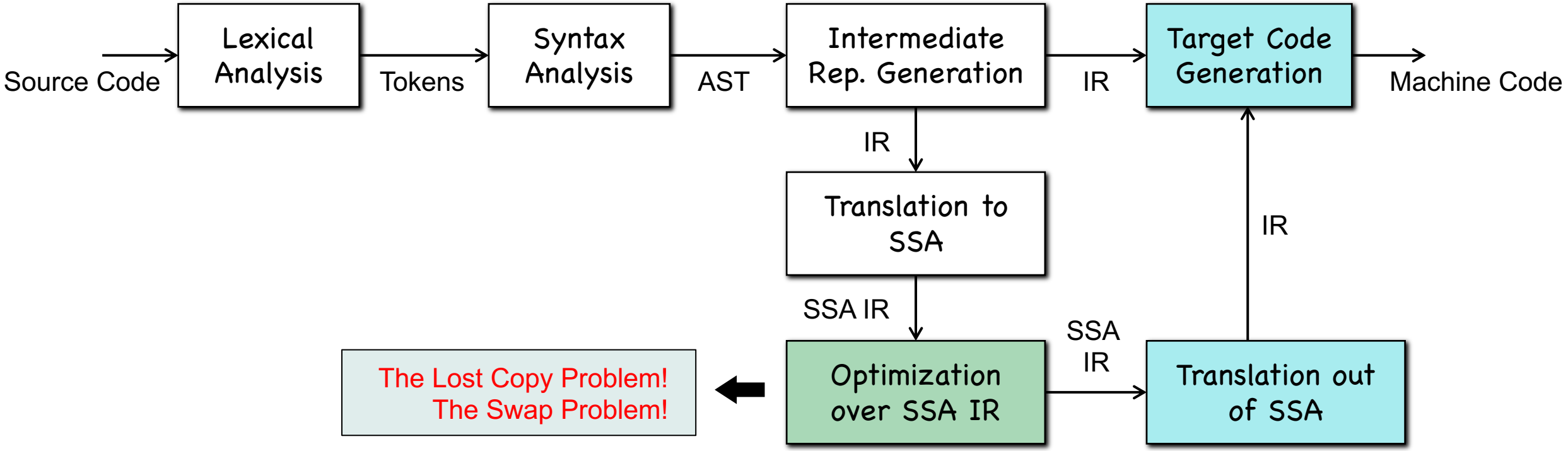
Where’re the Problems from?



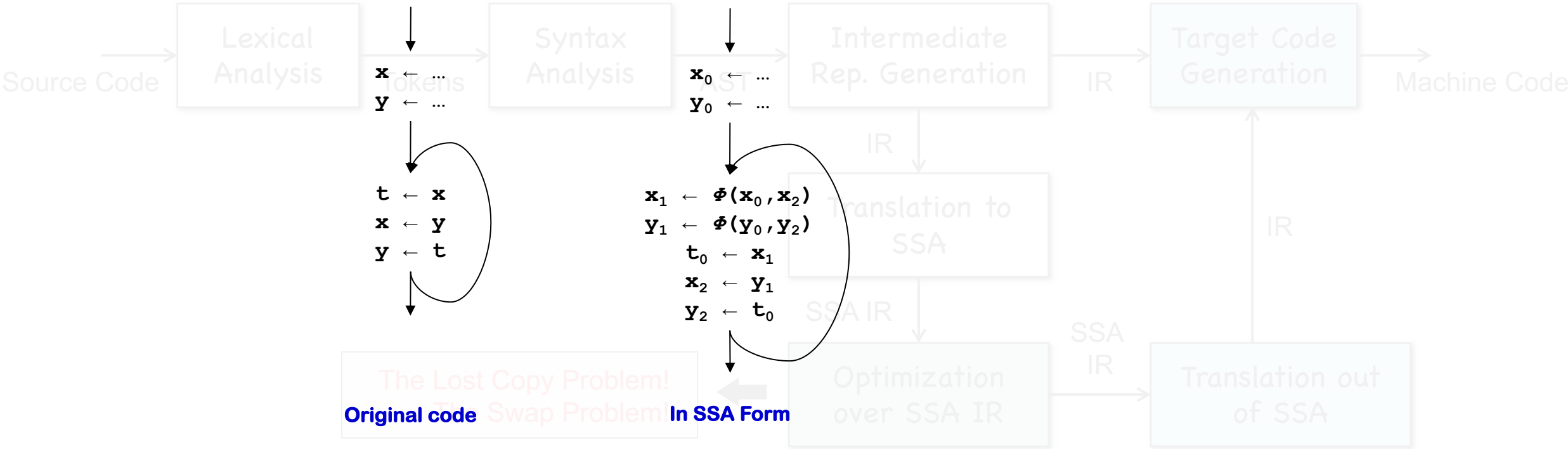
Where're the Problems from?



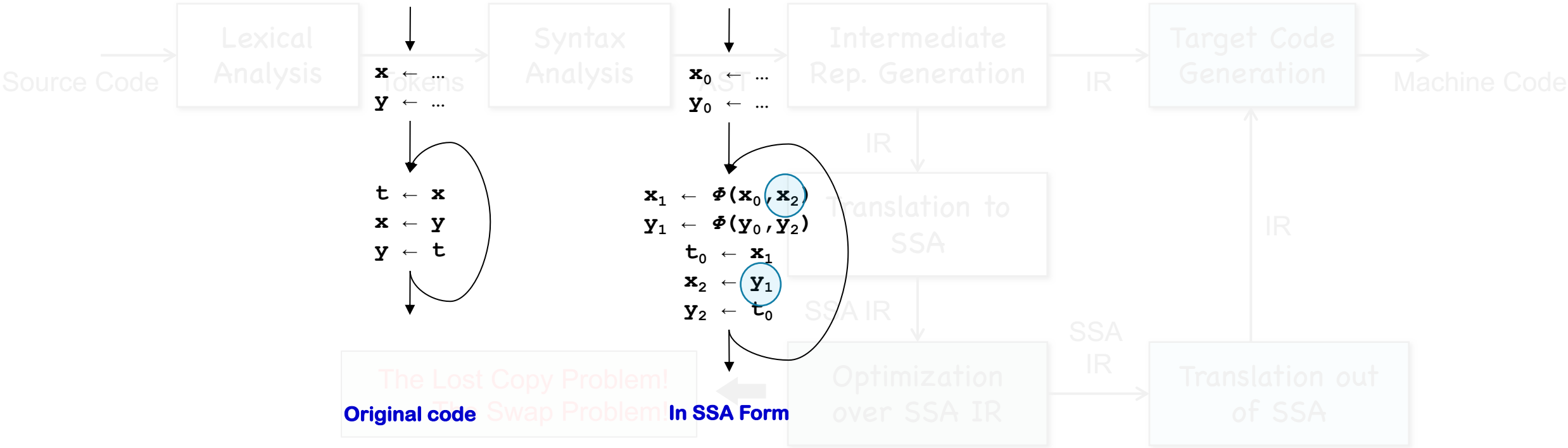
Where're the Problems from?



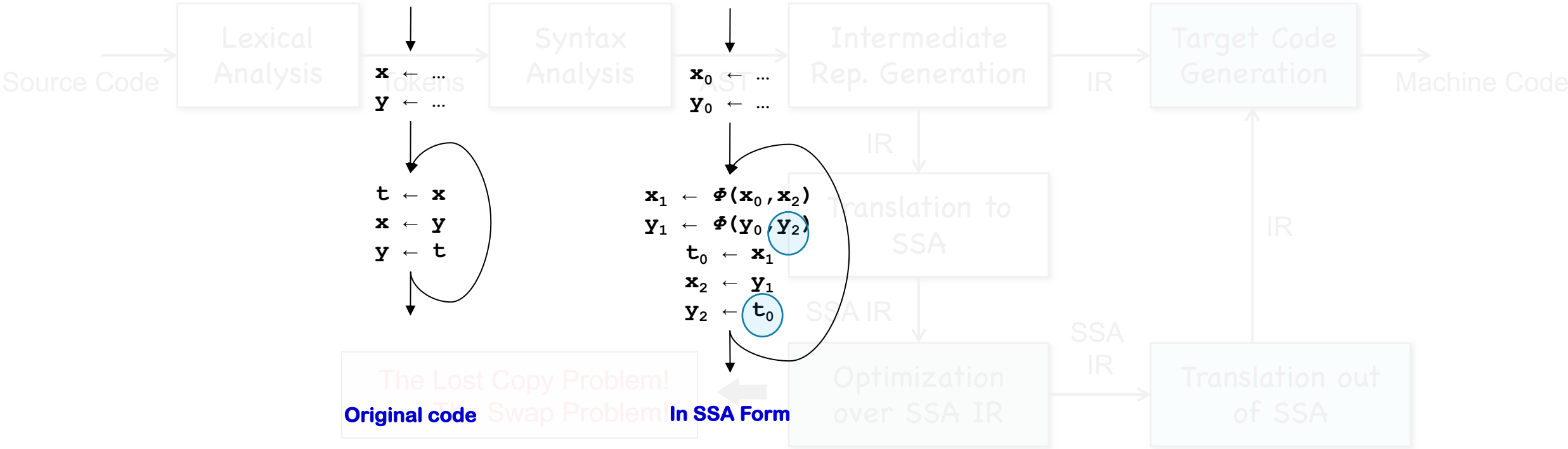
Where're the Problems from?



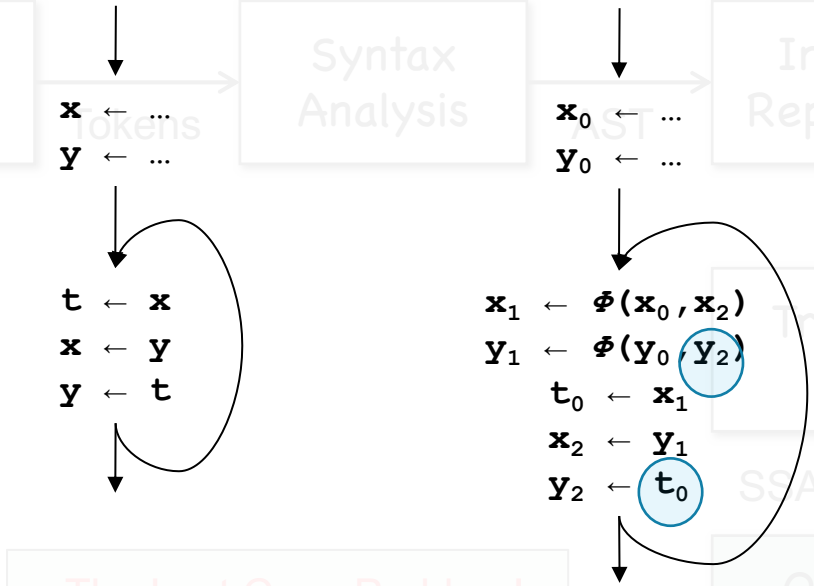
Where're the Problems from?



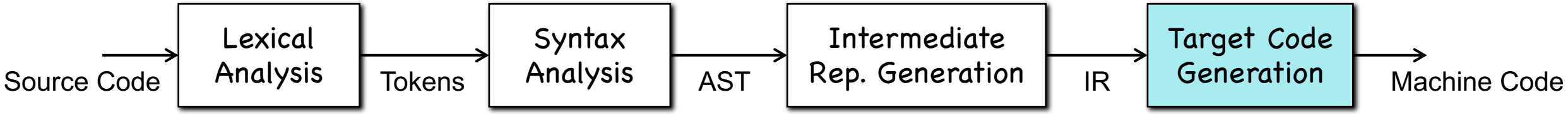
Where’re the Problems from?



Where're the Problems from?

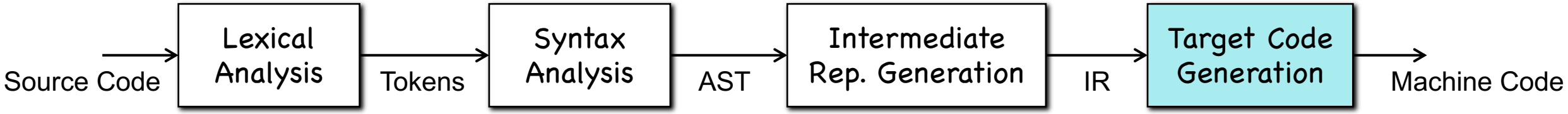


Target Code Generation



- These problems were identified by **Cliff Click** and were first published by **Briggs et. al.** in 1997. They address the issues by ad-hoc ways.

Target Code Generation

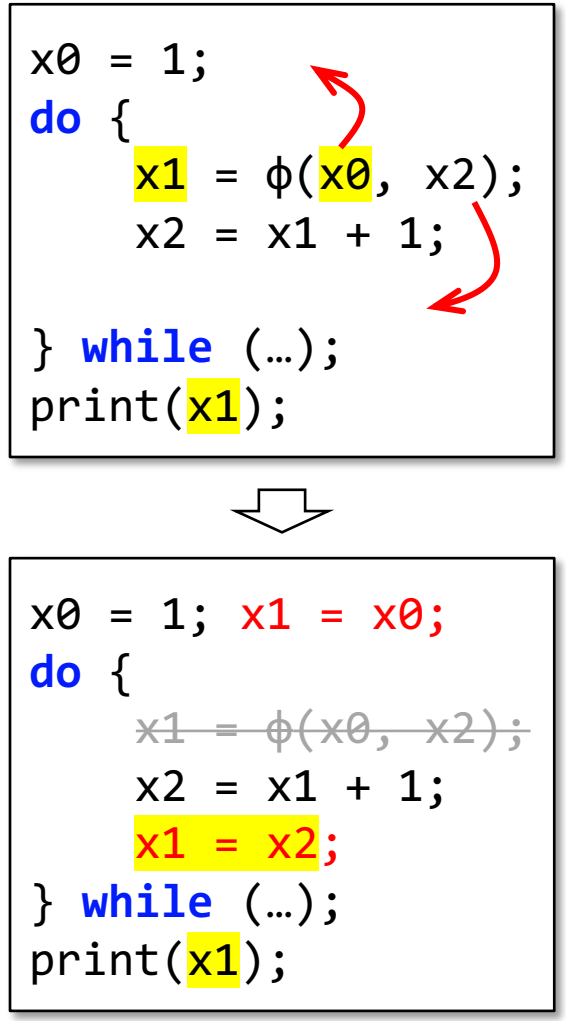


- These problems were identified by **Cliff Click** and were first published by **Briggs et. al.** in 1997. They address the issues by ad-hoc ways.

- Revisiting Out-of-SSA Translation for Correctness, Code Quality and Efficiency.
- Published in CGO 2009. By **Boissinot et. al.**

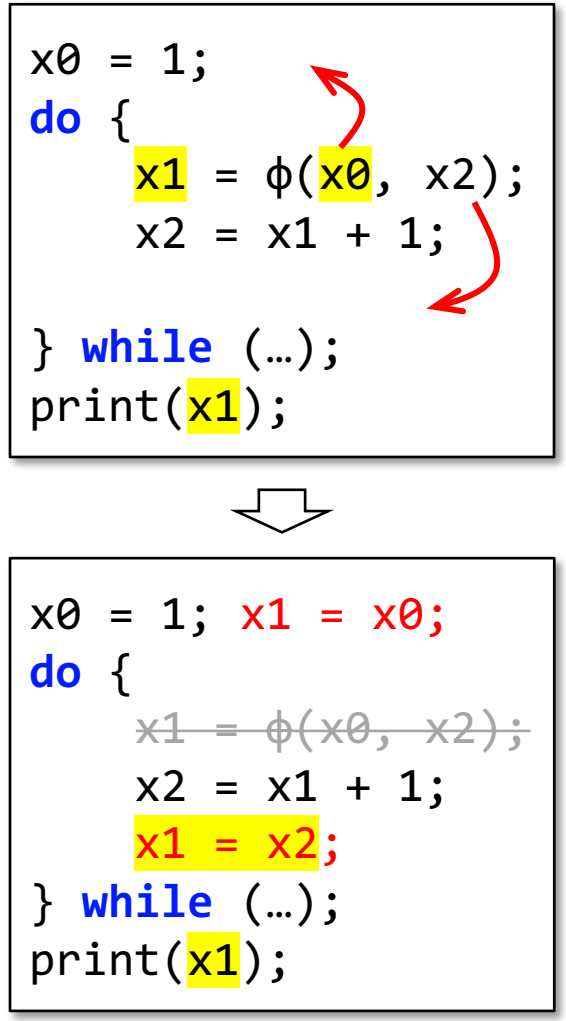
The Lost Copy Problem

- The copy X1 = X0 is lost!
- To address the issue, we need to create a temp variable to hold this value.



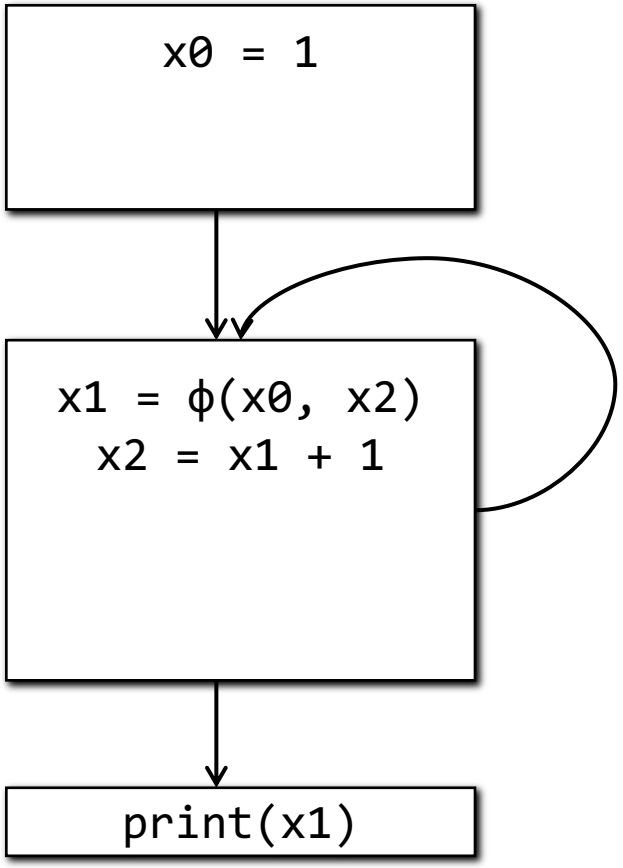
The Lost Copy Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Coalesce redundant copies



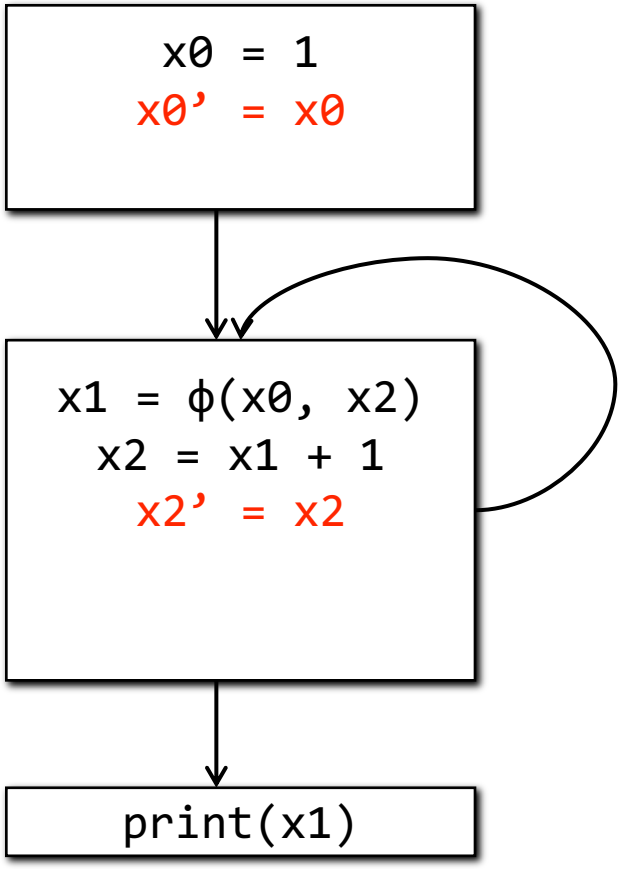
The Lost Copy Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Coalesce redundant copies



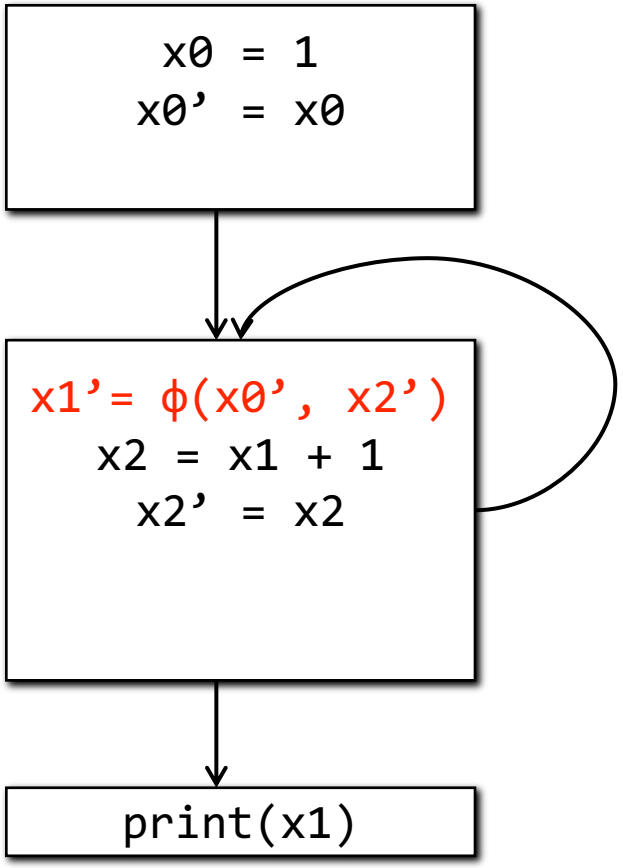
The Lost Copy Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Coalesce redundant copies



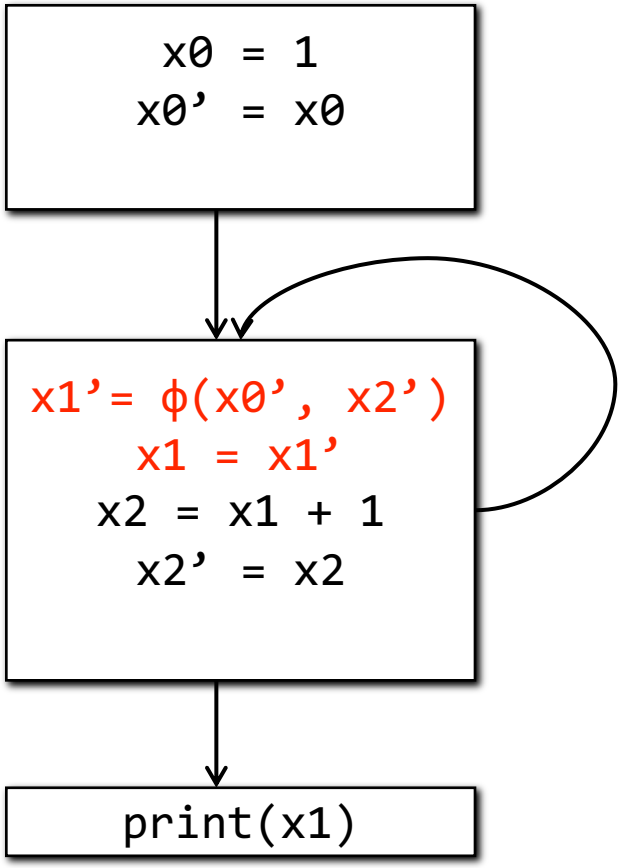
The Lost Copy Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Coalesce redundant copies



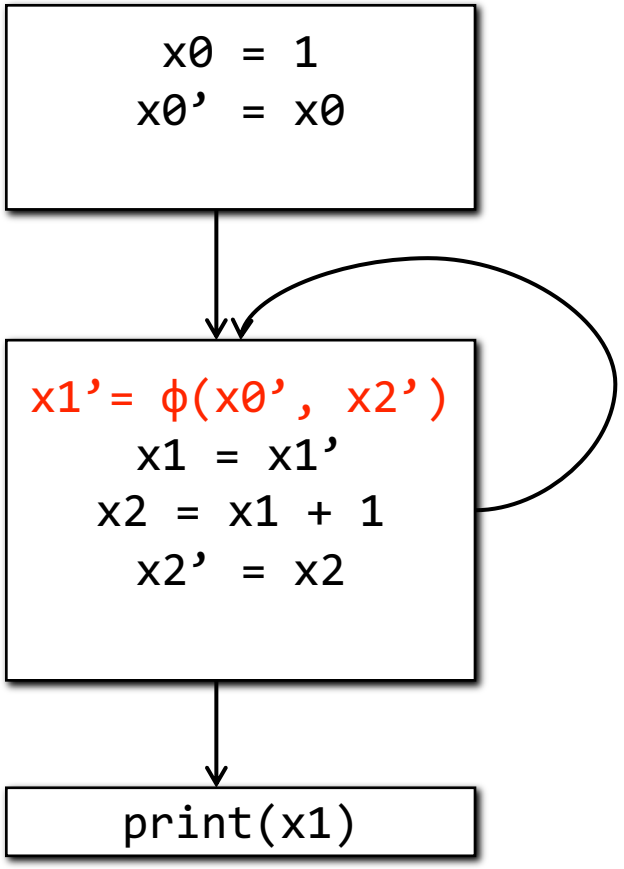
The Lost Copy Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Coalesce redundant copies



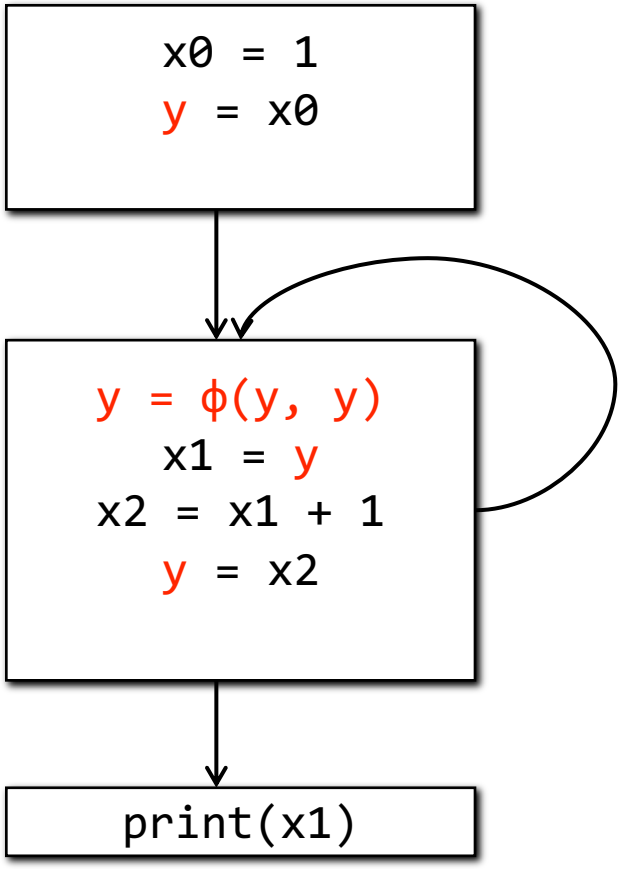
The Lost Copy Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Coalesce redundant copies



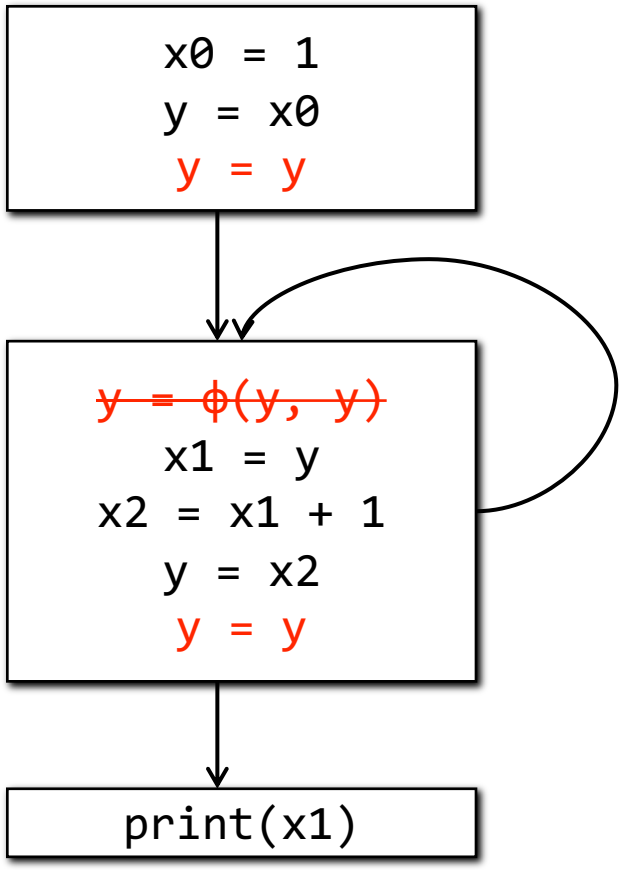
The Lost Copy Problem

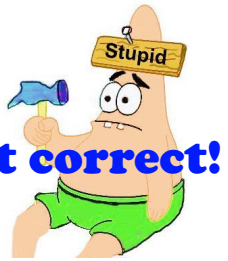
- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Coalesce redundant copies



The Lost Copy Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Coalesce redundant copies

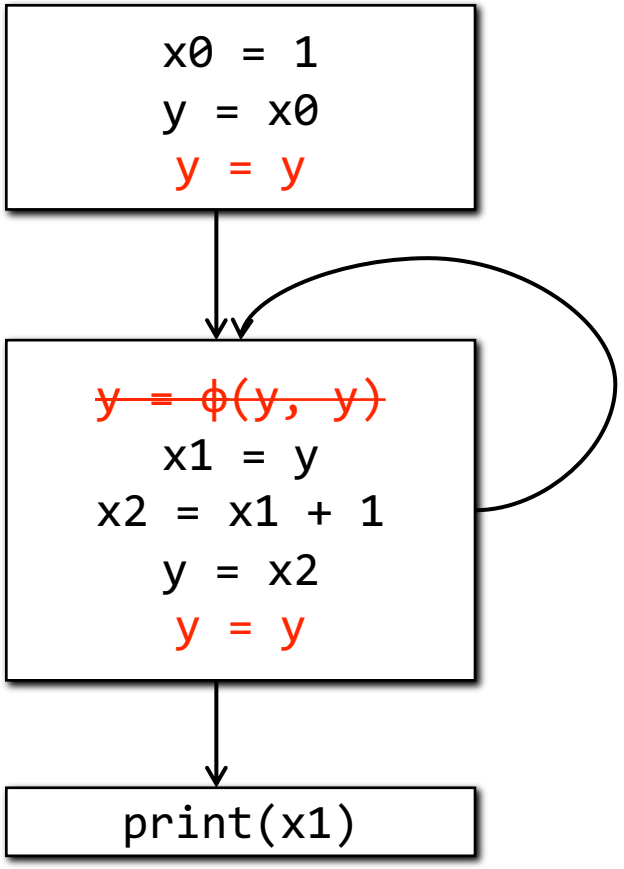




looks stupid but correct!

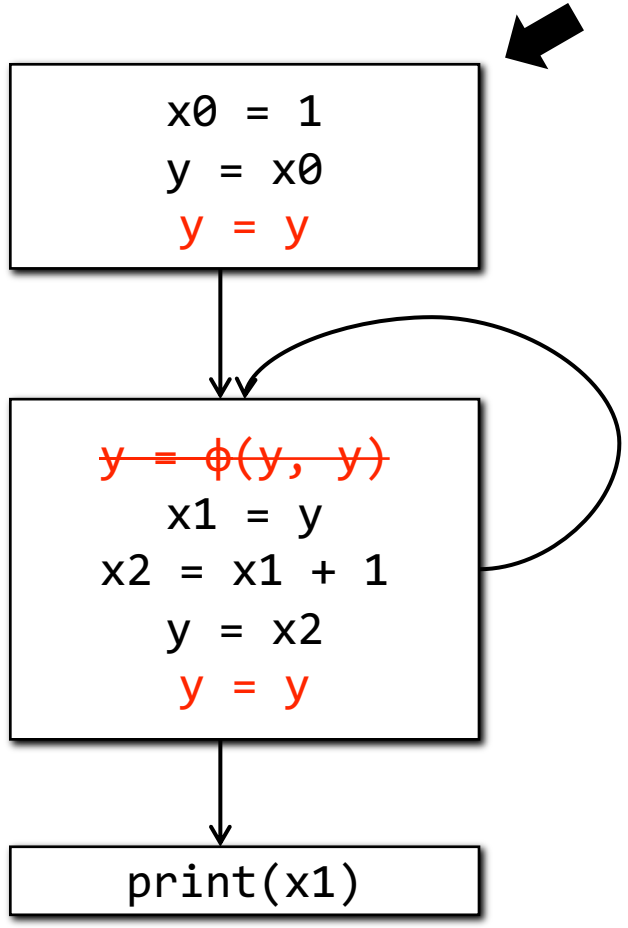
The Lost Copy Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Coalesce redundant copies



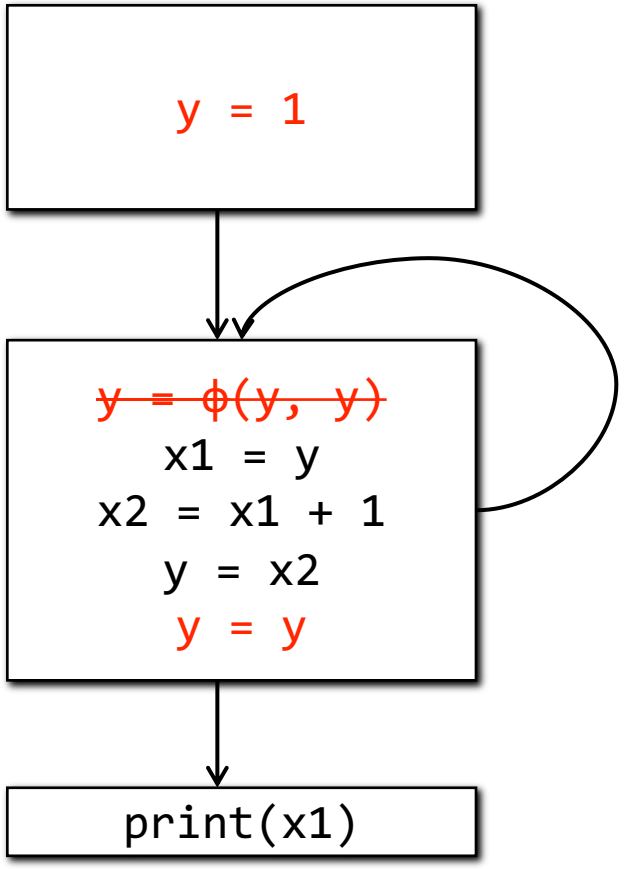
The Lost Copy Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Coalesce redundant copies



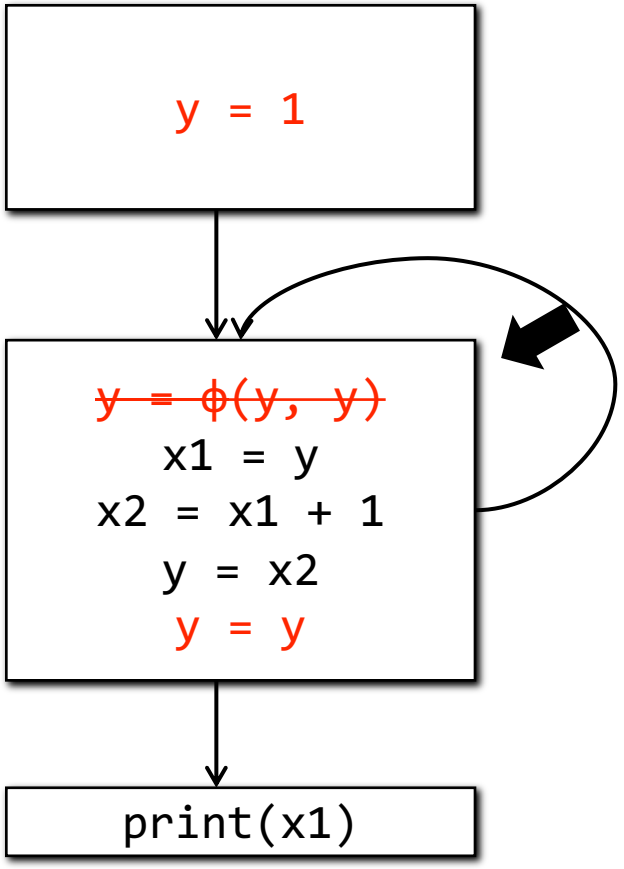
The Lost Copy Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Coalesce redundant copies



The Lost Copy Problem

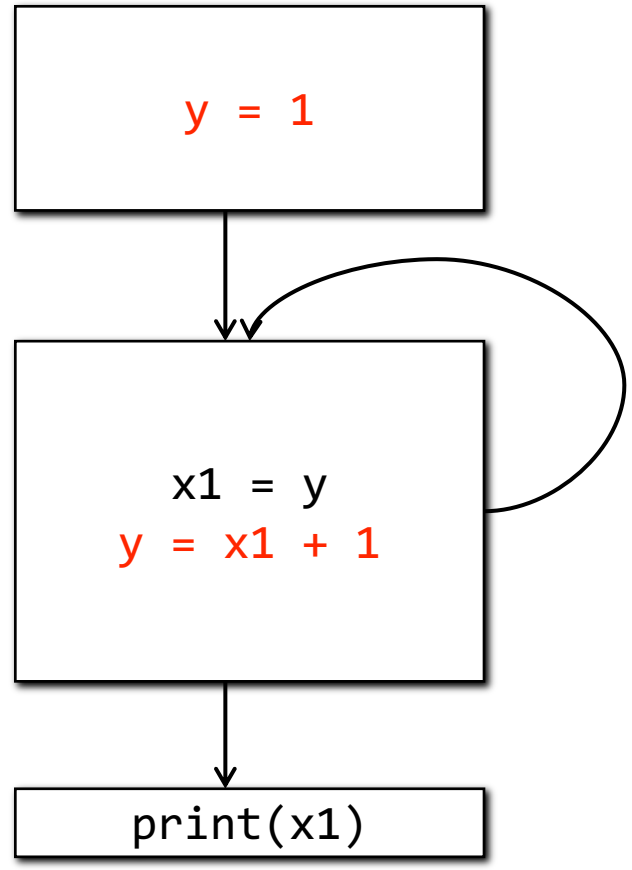
- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Coalesce redundant copies



The Lost Copy Problem

- For each a = ϕ(a1, a2, …)
  - Insert ai' = ai at the end of th
  - Replace the ϕ with a'= ϕ(a1',
  - Insert a copy of a=a' after ϕ
  - Replace all a' and ai' with a new name
  - Remove ϕ by inserting copies in predecessors
  - Coalesce redundant copies

```
x0 = 1;
do {
    x1 = ϕ(x0, x2);
    x2 = x1 + 1;

} while (…);
print(x1);
```



The Swap Problem

- Caused by multiple ϕ in the same block
- The parallel semantics of ϕ statements

```
x0 = 1; y0 = 2;

do {
    x1 = ϕ(x0, y1);
    y1 = ϕ(y0, x1);
} while (…);
print(x1, y1);
```

```
x0 = 1; y0 = 2;
x1 = x0; y1 = y0;
do {
    x1 = y1;
    y1 = x1;
} while (…);
print(x1, y1);
```

The Swap Problem

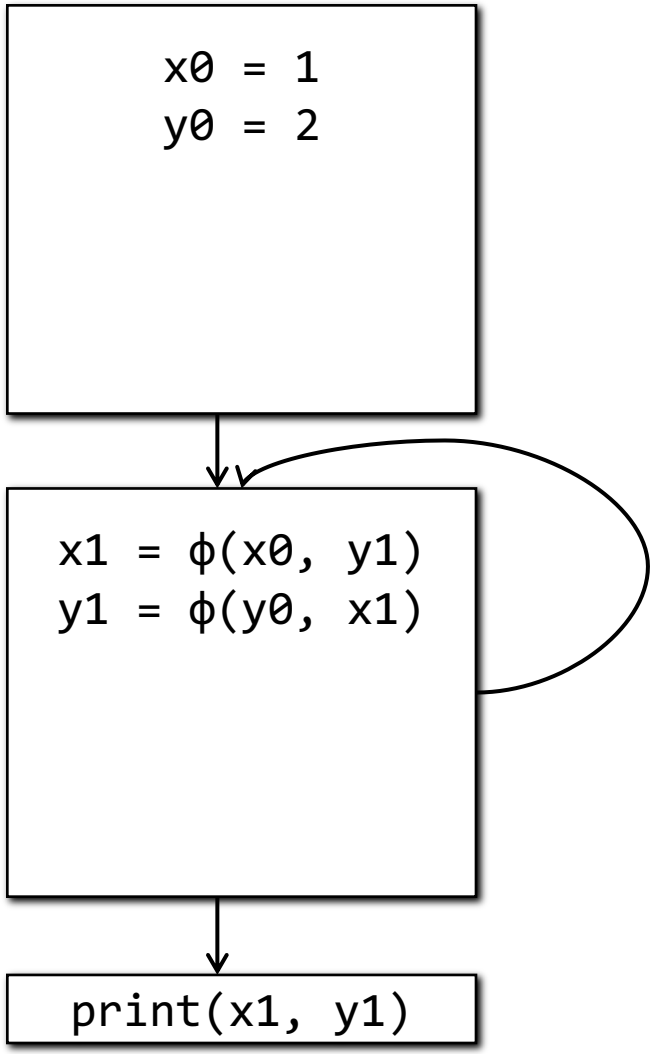
- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies

```
x0 = 1; y0 = 2;

do {
    x1 = ϕ(x0, y1);
    y1 = ϕ(y0, x1);
} while (…);
print(x1, y1);
```

```
x0 = 1; y0 = 2;
x1 = x0; y1 = y0;
do {
    x1 = y1;
    y1 = x1;
} while (…);
print(x1, y1);
```

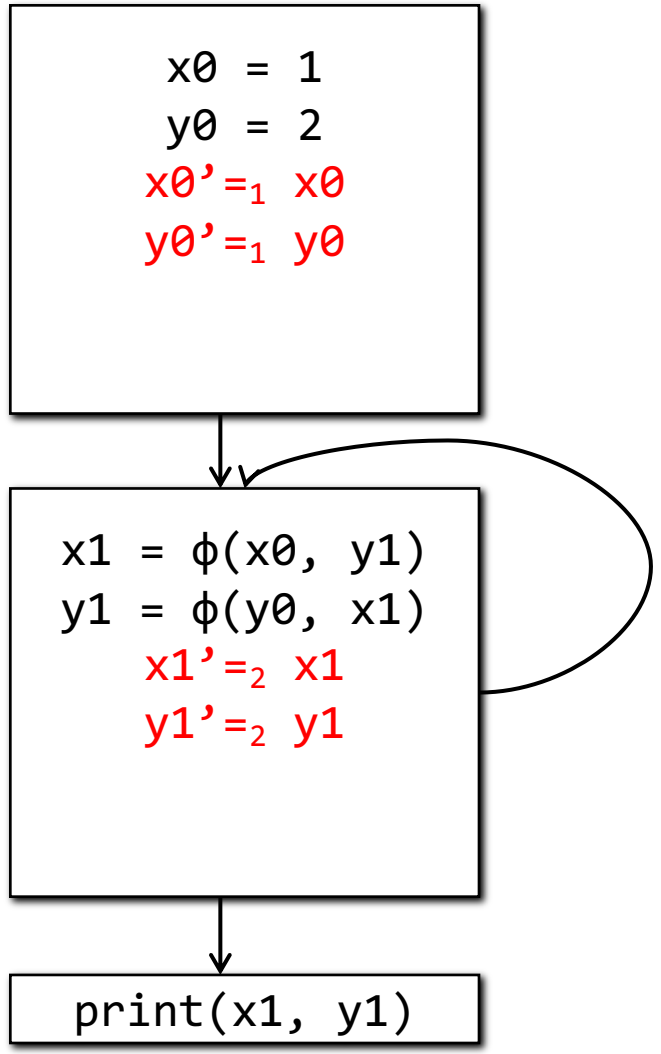
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



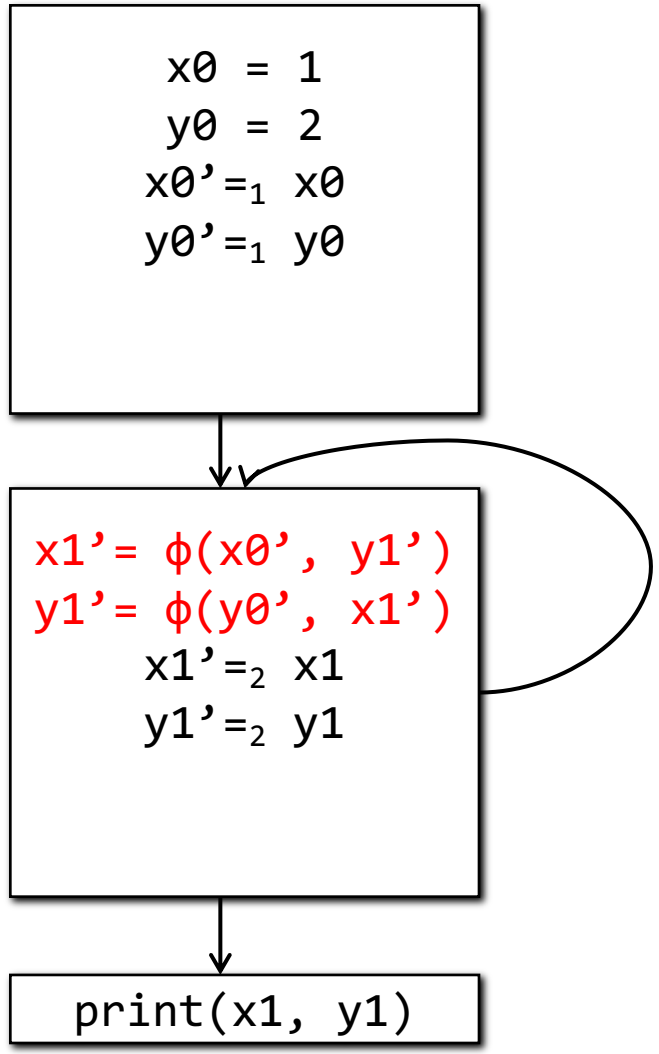
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



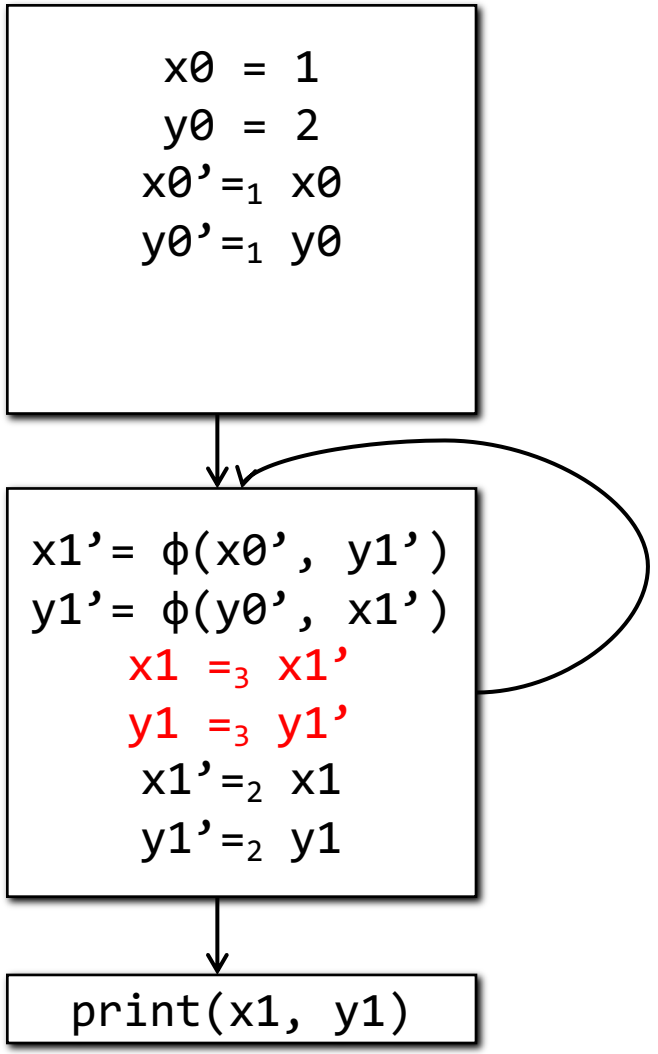
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



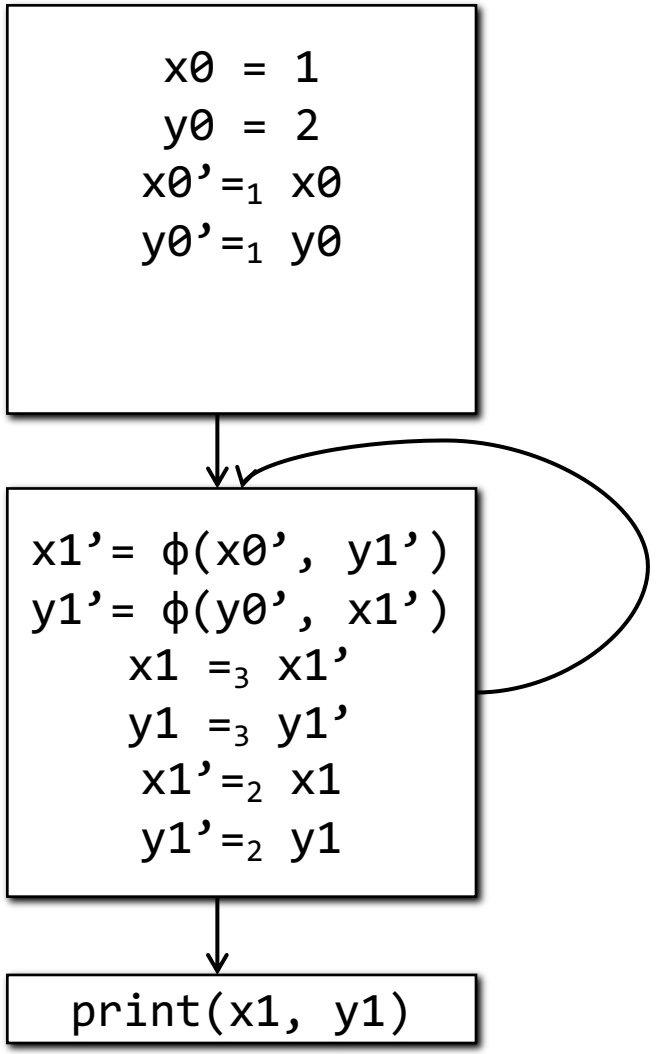
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



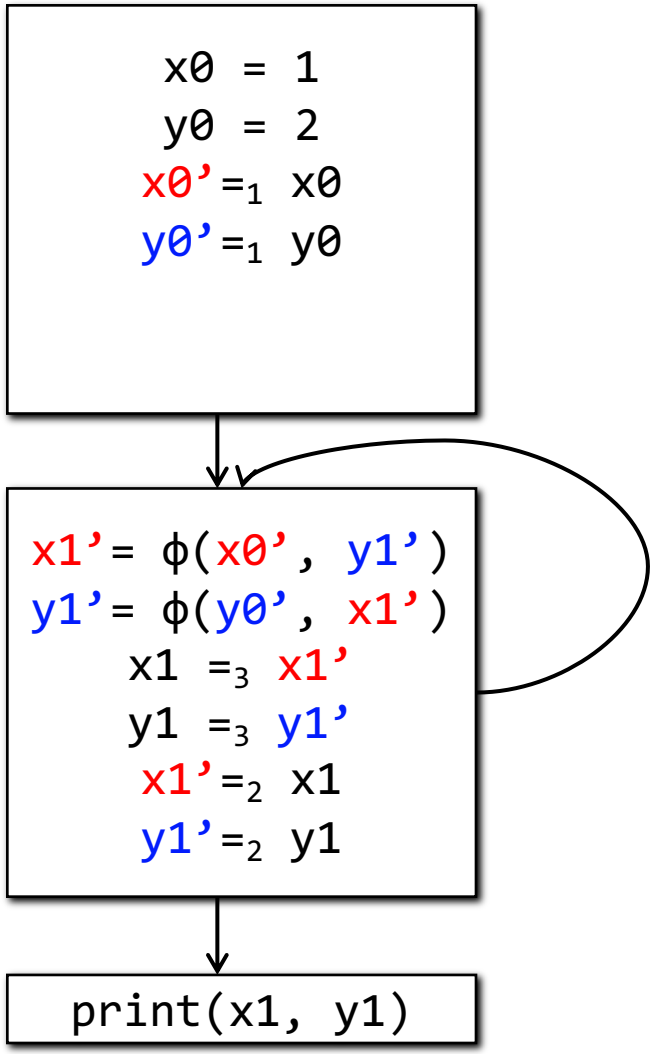
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



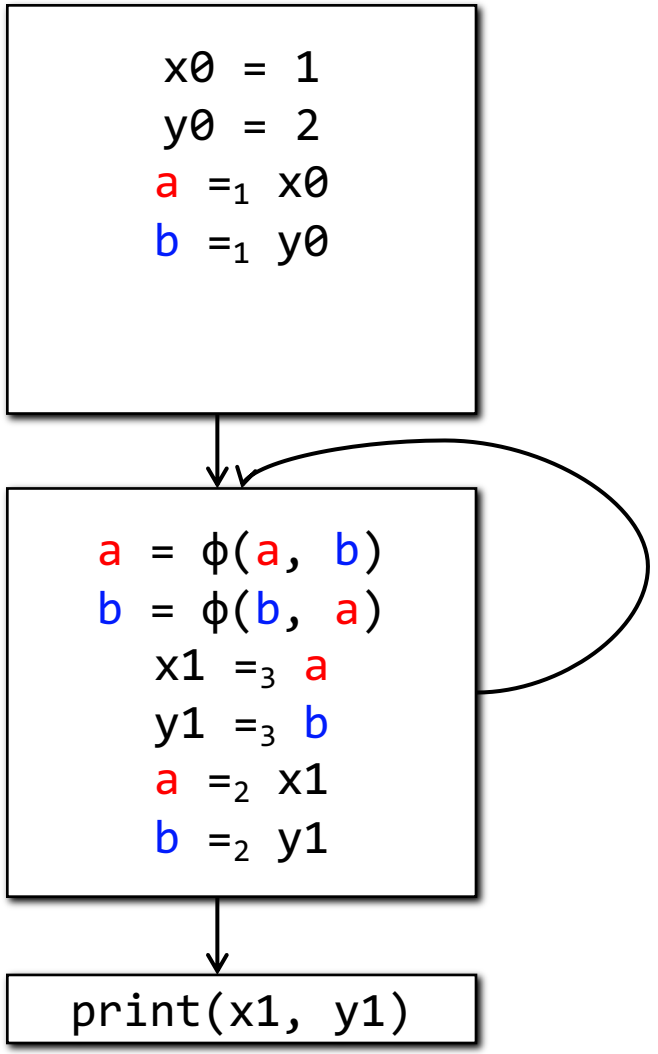
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



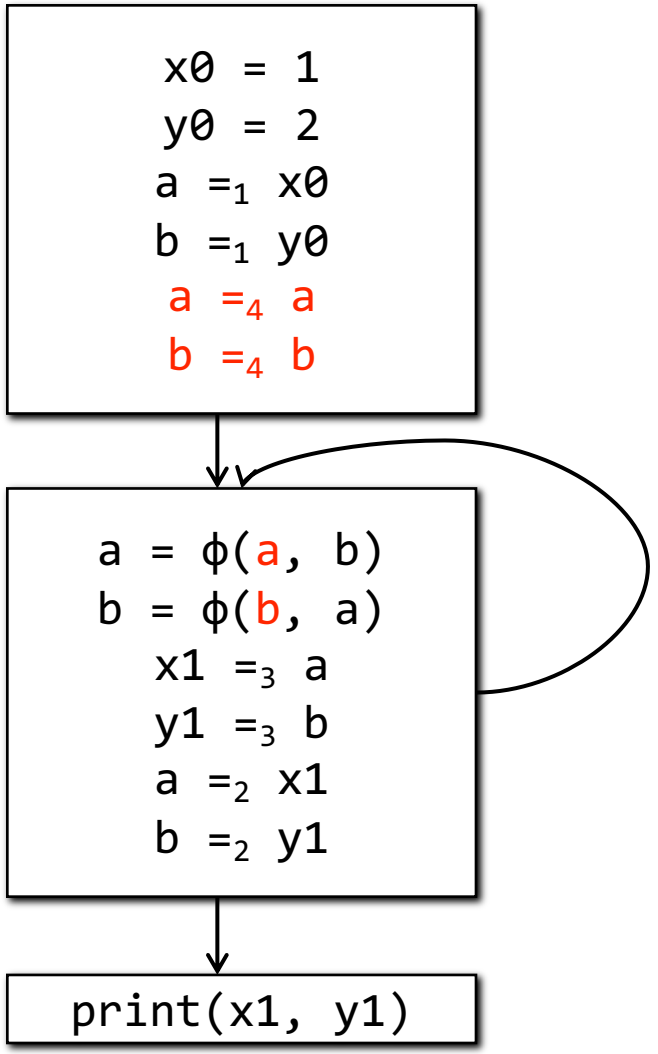
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



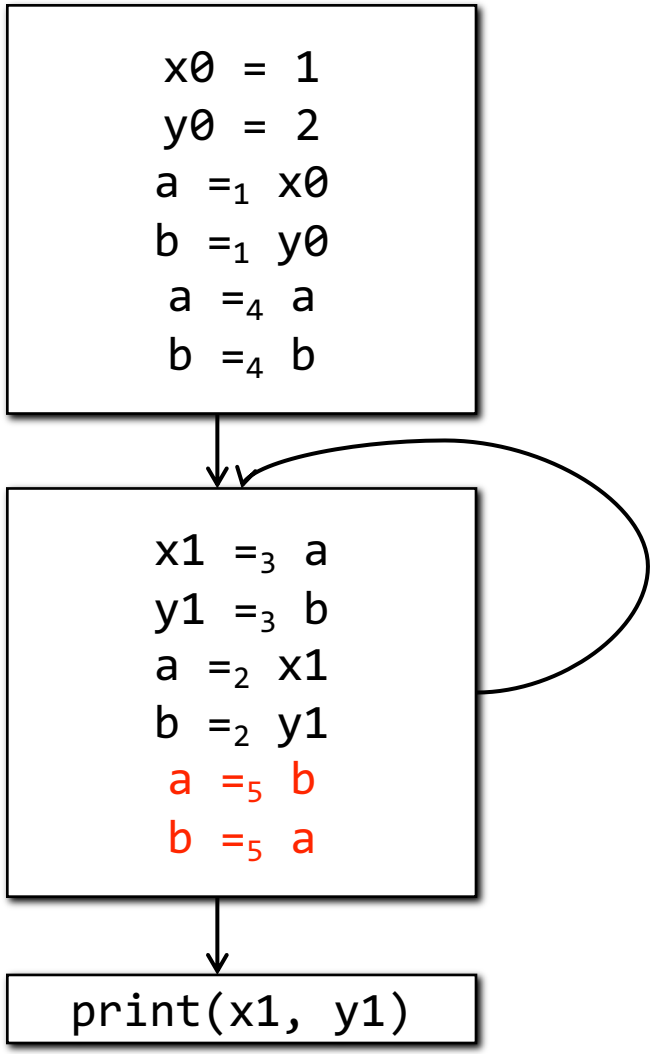
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



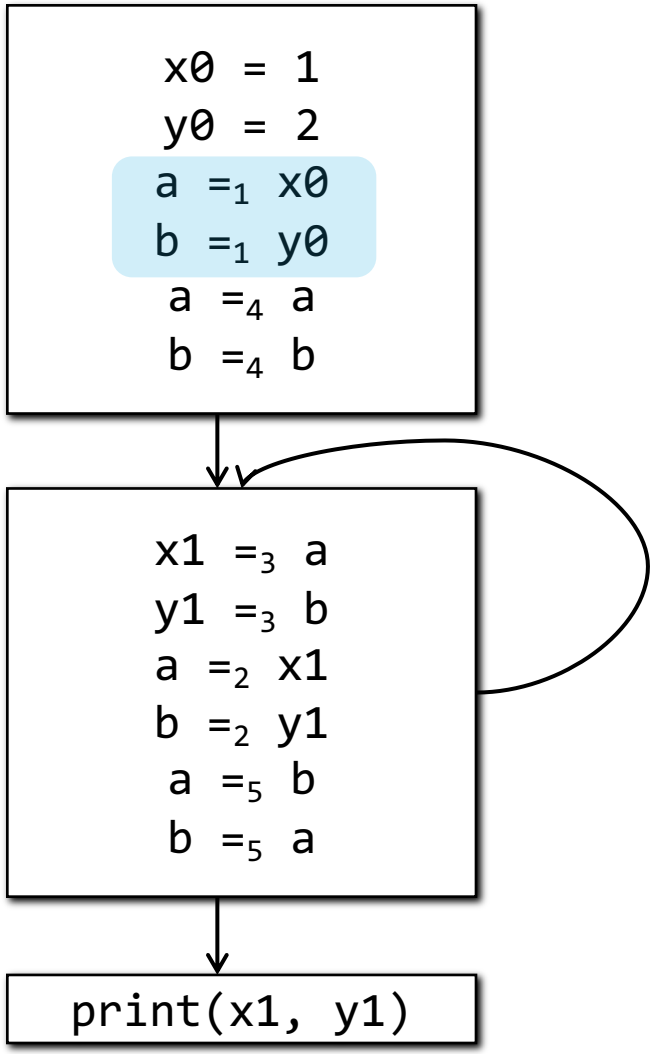
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



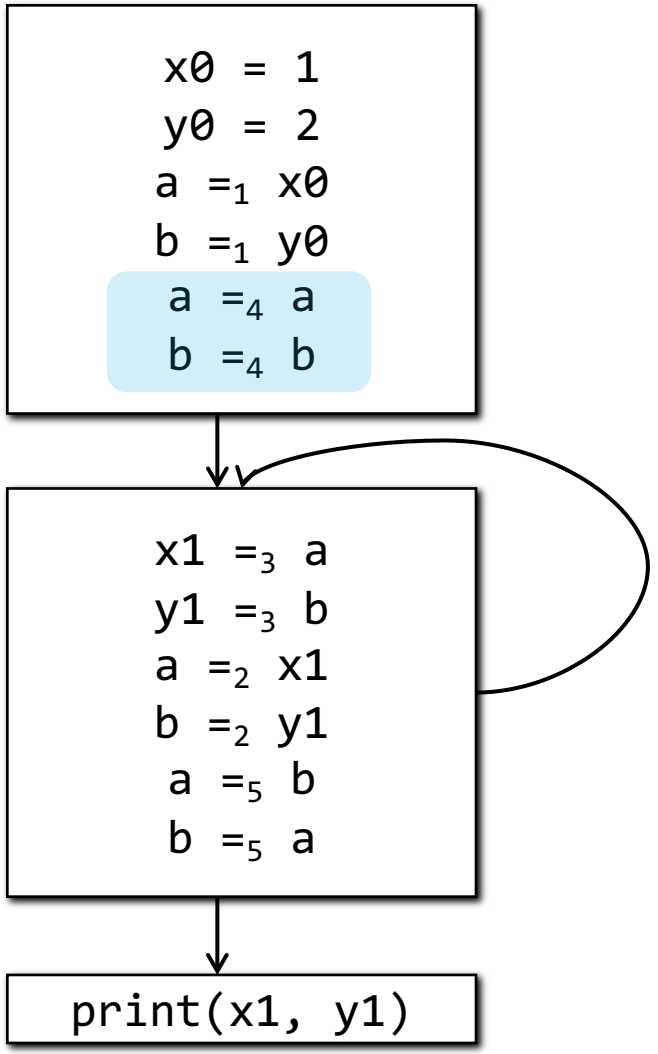
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



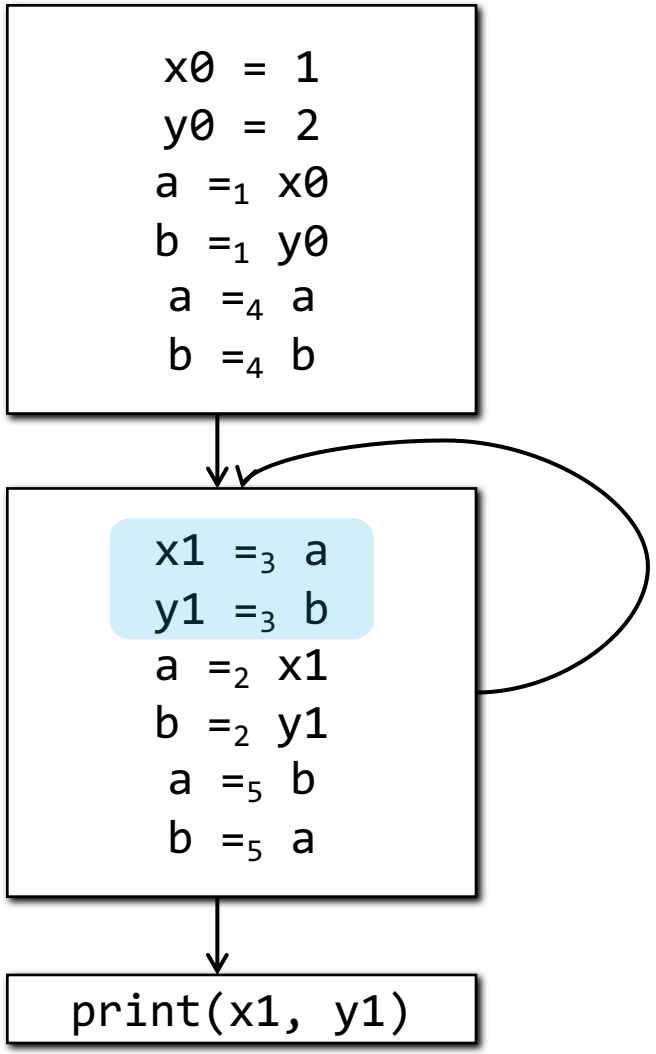
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



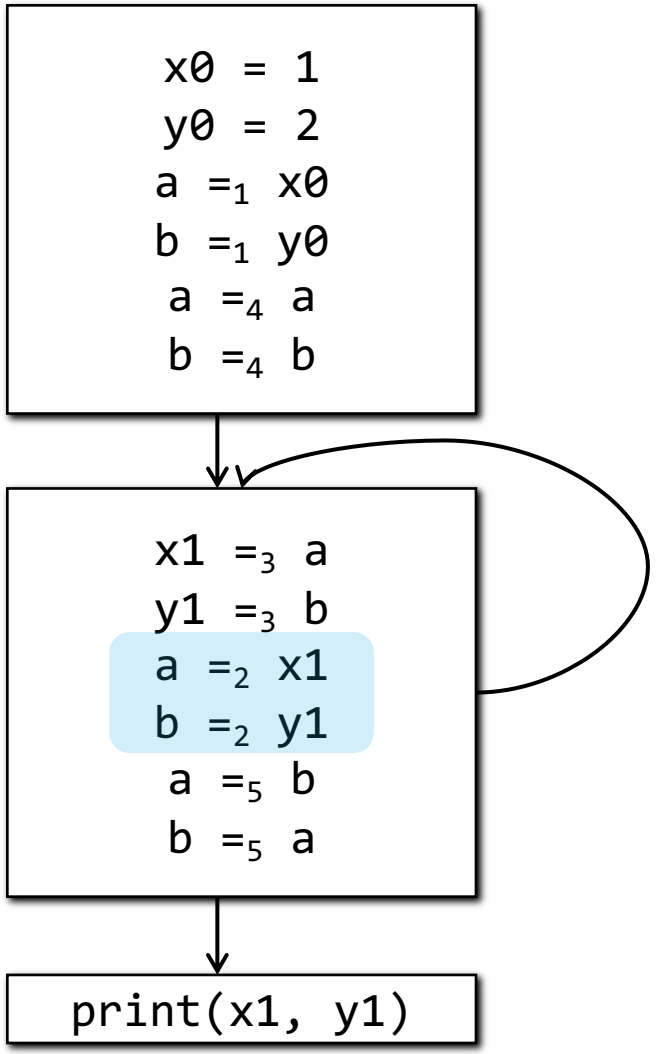
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



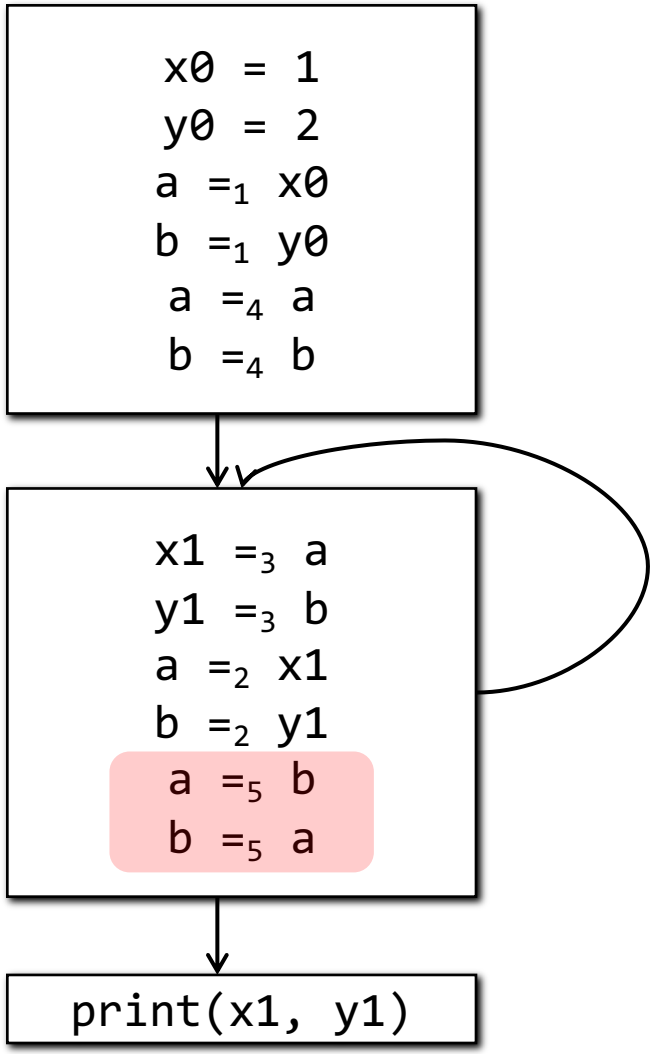
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



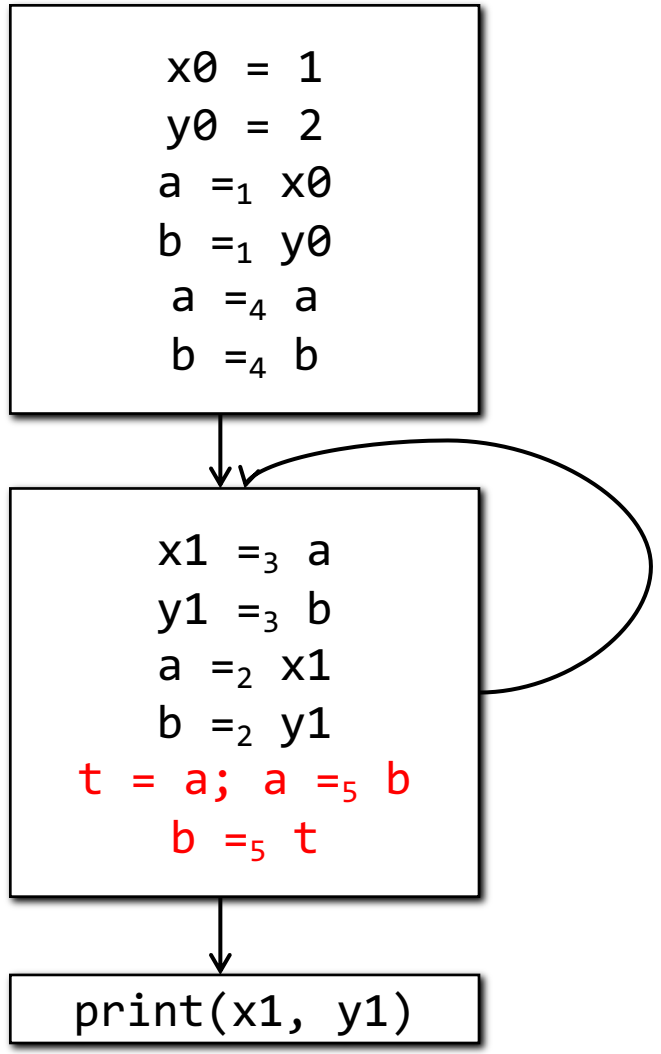
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



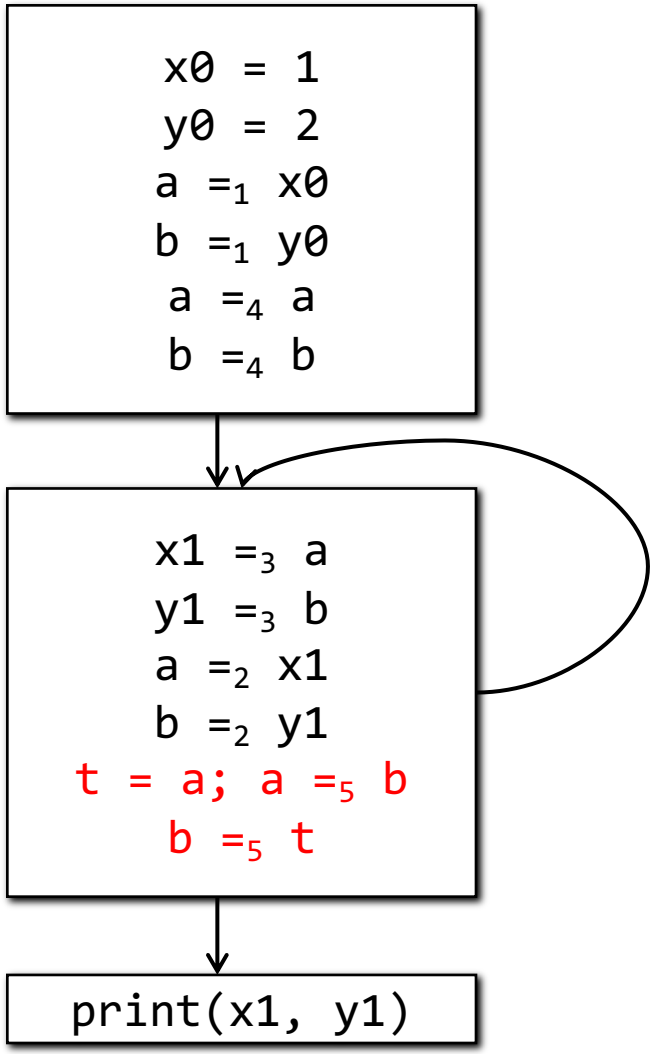
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



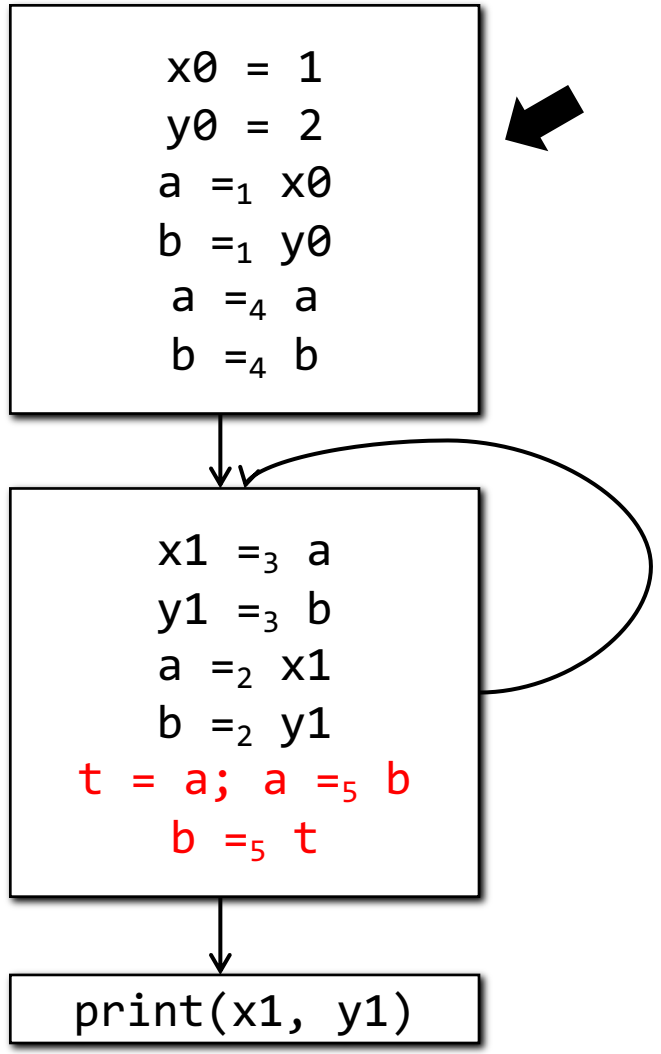
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



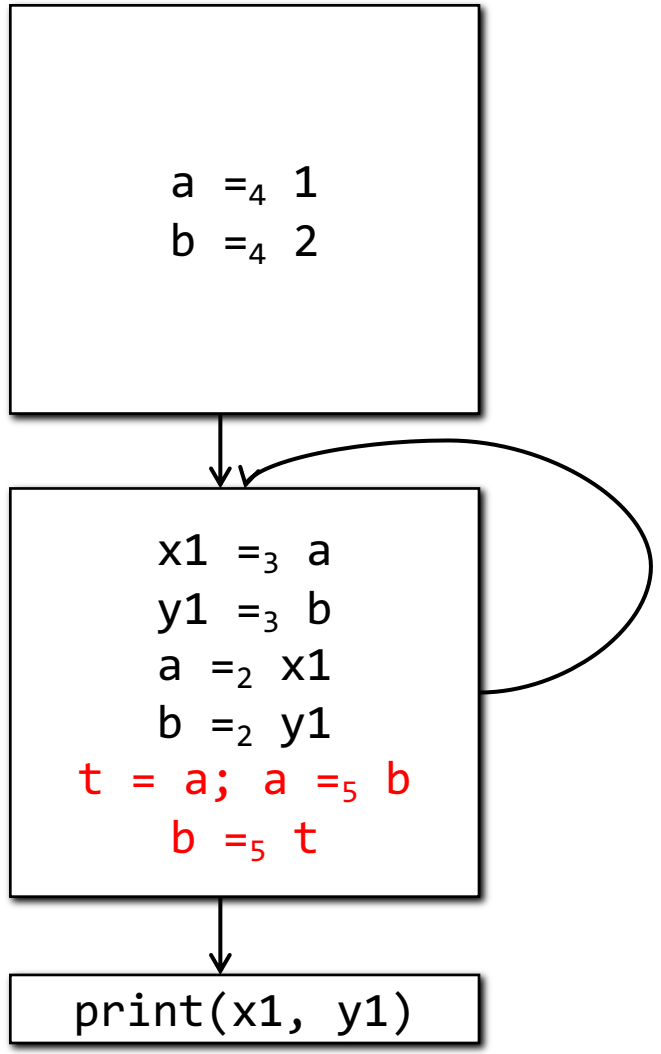
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



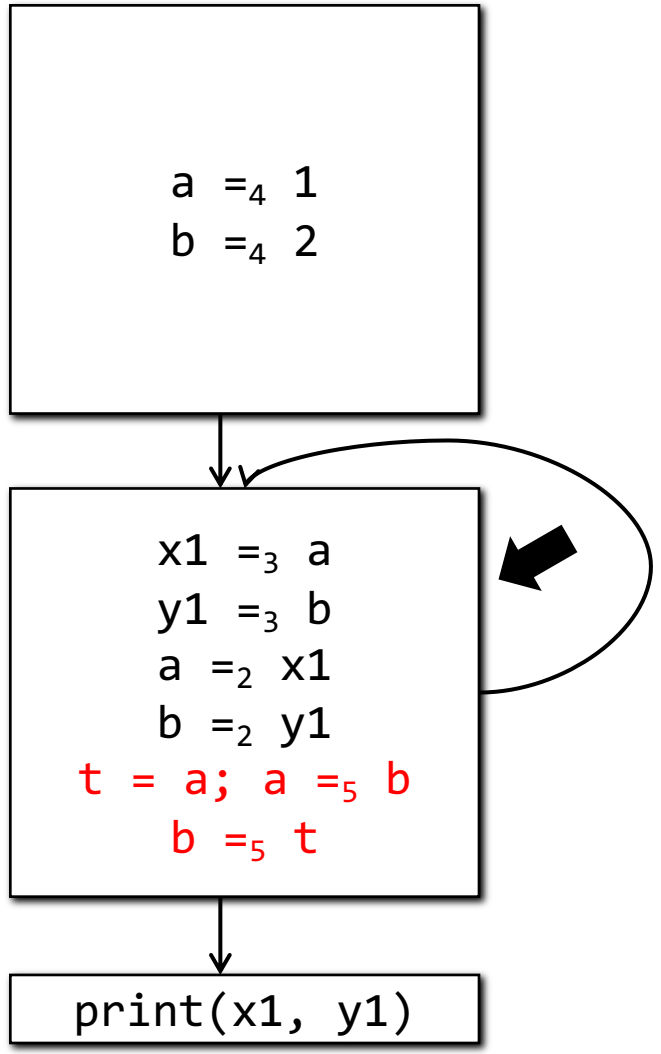
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



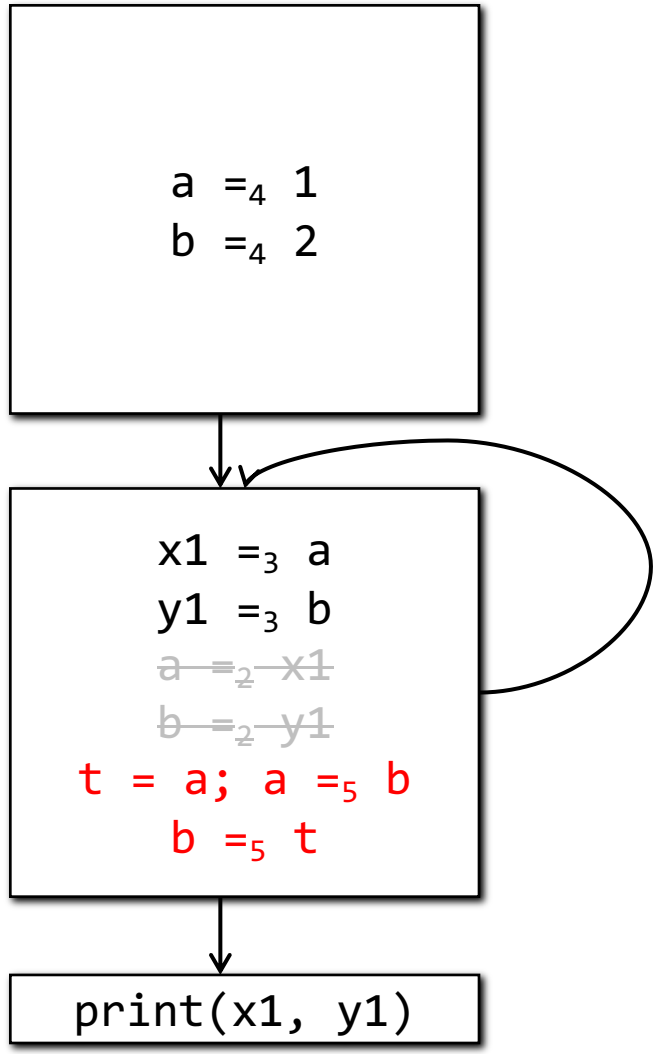
The Swap Problem

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



The Swap Problem

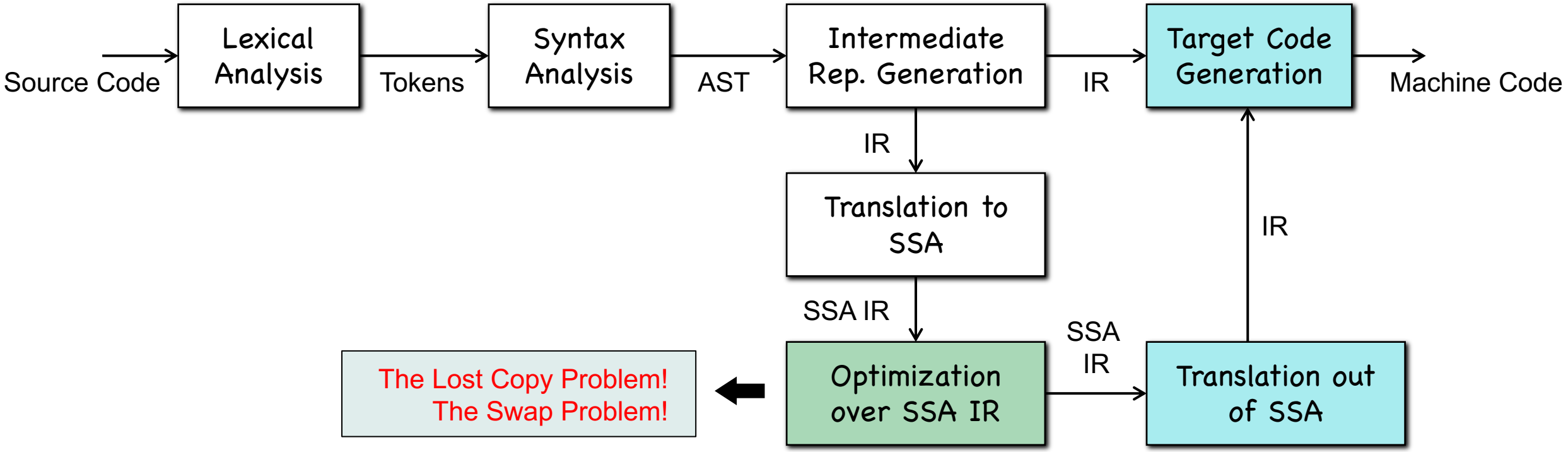
- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies
  - Coalesce redundant copies



Translation Out of SSA Form

- For each `a = ϕ(a1, a2, …)`
  - Insert `ai' = ai` at the end of the block of `ai`
  - Replace the ϕ with `a'= ϕ(a1', a2', …)`
  - Insert a copy of `a=a'` after ϕ
  - Replace all `a'` and `ai'` with a new name
  - Remove ϕ by inserting copies in predecessors
  - Sequentialize the parallel copies --- break cyclic dependency
  - Coalesce redundant copies

Summary


Source Code
Lexical Analysis
Tokens
Syntax Analysis
AST
Intermediate Rep. Generation
IR
Target Code Generation
Machine Code
IR
Translation to SSA
SSA IR
Optimization over SSA IR
SSA IR
Translation out of SSA
IR
The Lost Copy Problem!
The Swap Problem!

THANKS!