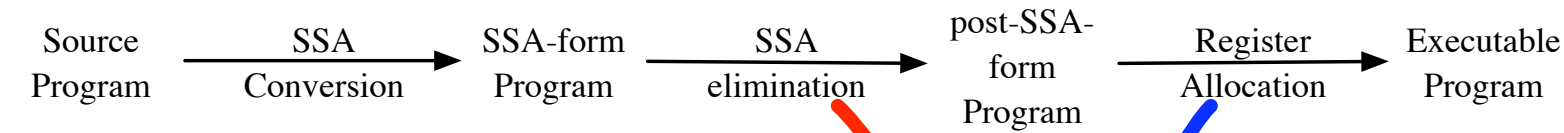
SSA-Elimination after Register Allocation

Fernando Pereira UFMG
Jens Palsberg UCLA

Outline

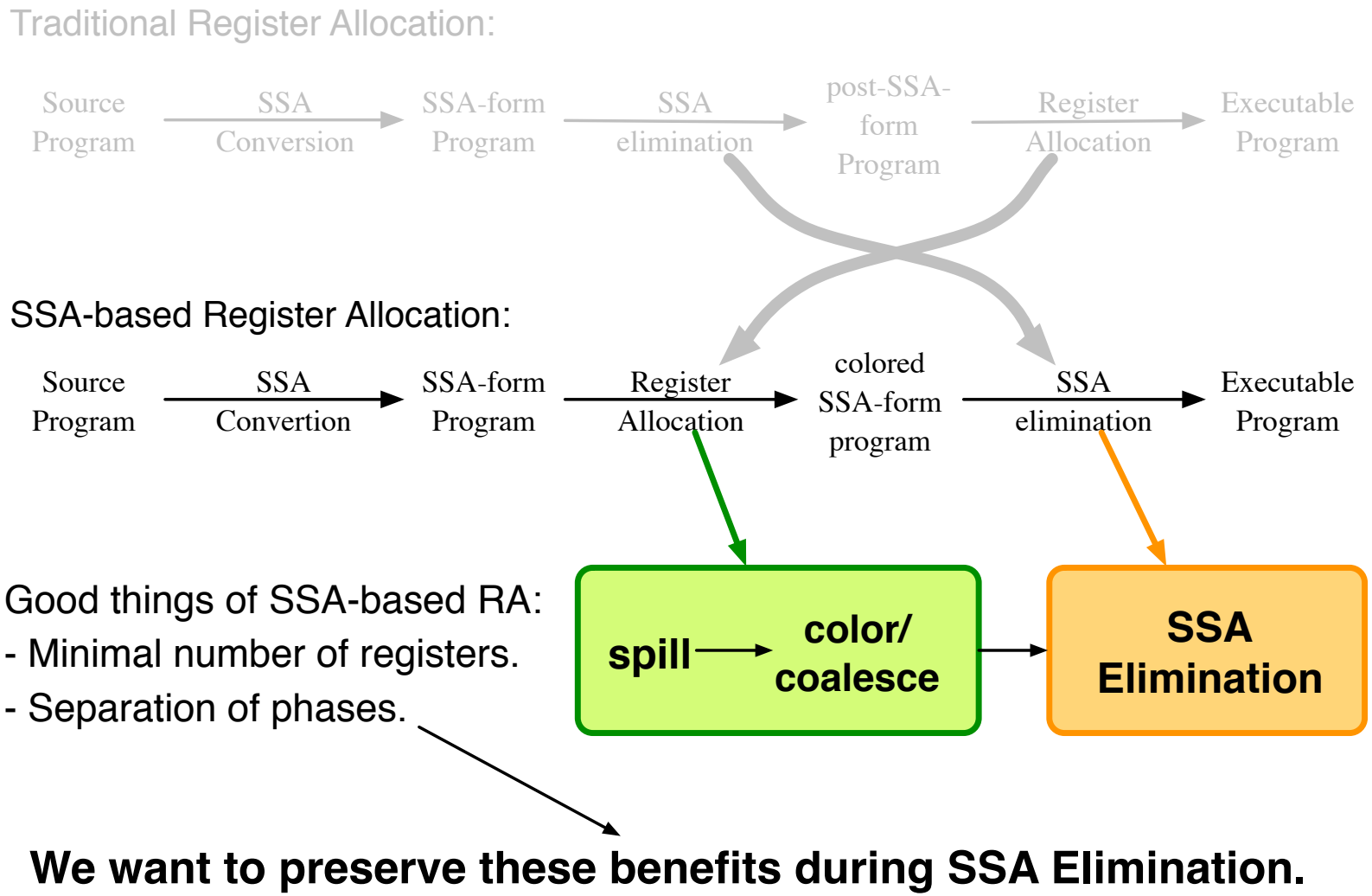
When to do SSA Elimination?

Traditional Register Allocation:



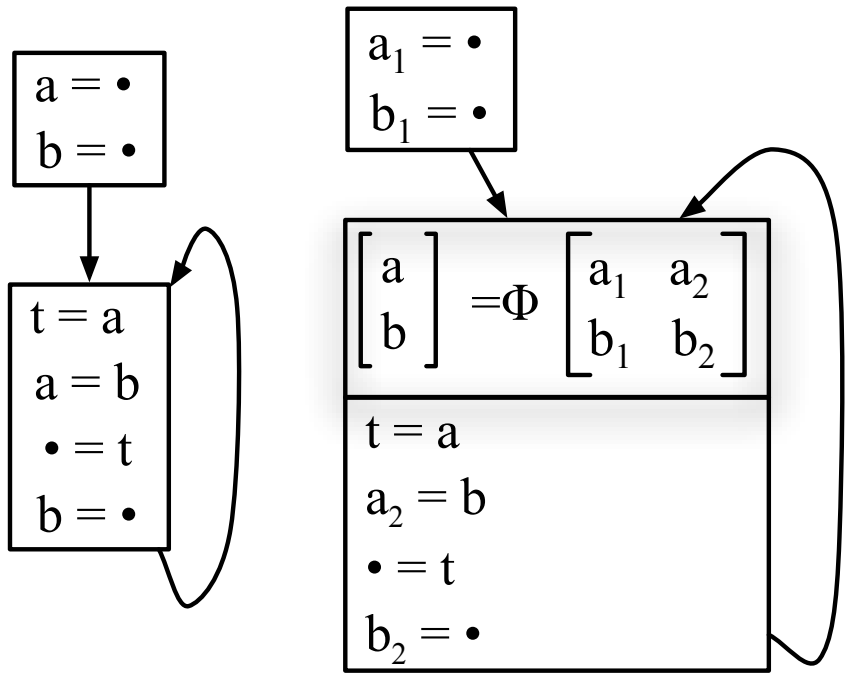
SSA-based Register Allocation:

When to do SSA Elimination?



Our running example

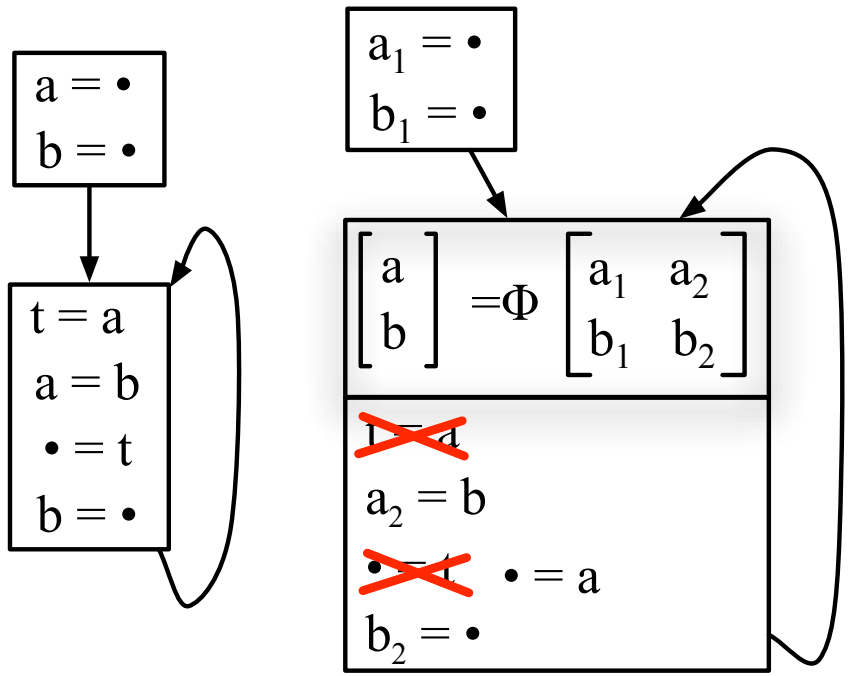
```
int a = 0;
int b = 0;
while(true) {
   t = a;
   a = b;
   print(t);
   b = read();
}
```



Variables that are related by ϕ-functions do not interfere. We shall see that this is a good property.

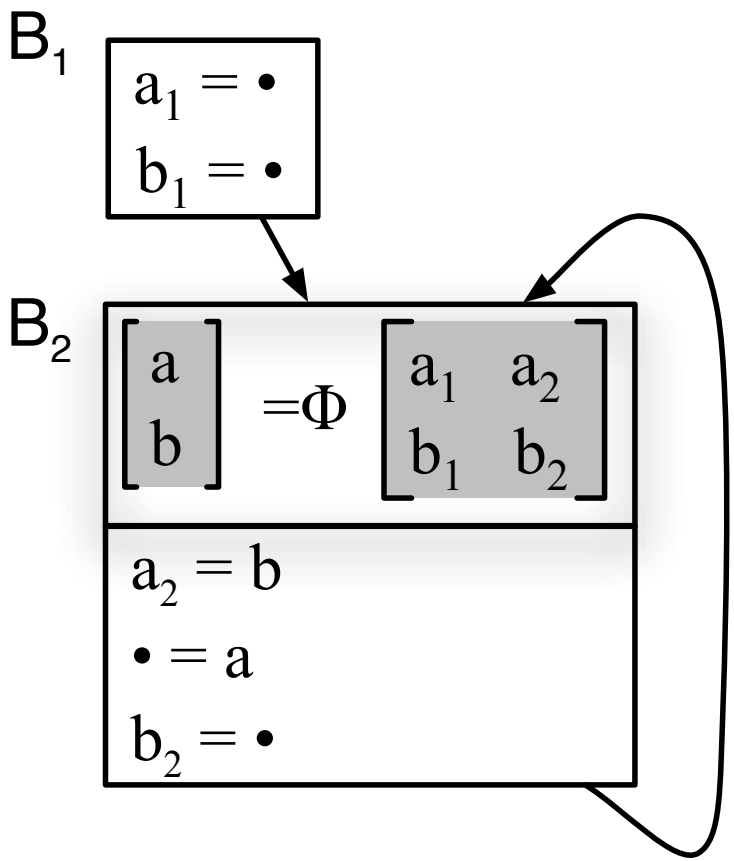
Some constant propagation...

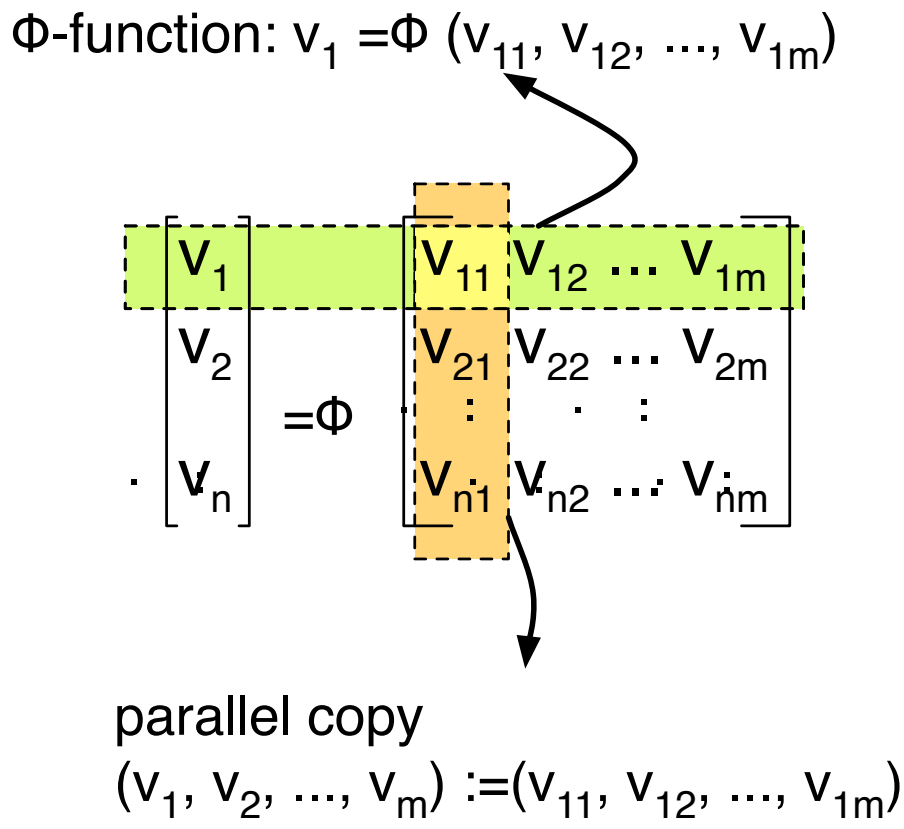
```
int a = 0;
int b = 0;
while(true) {
   t = a;
   a = b;
   print(t);
   b = read();
}
```



Now, variables that are related by ϕ-functions do interfere. We will see soon why this is a problem.

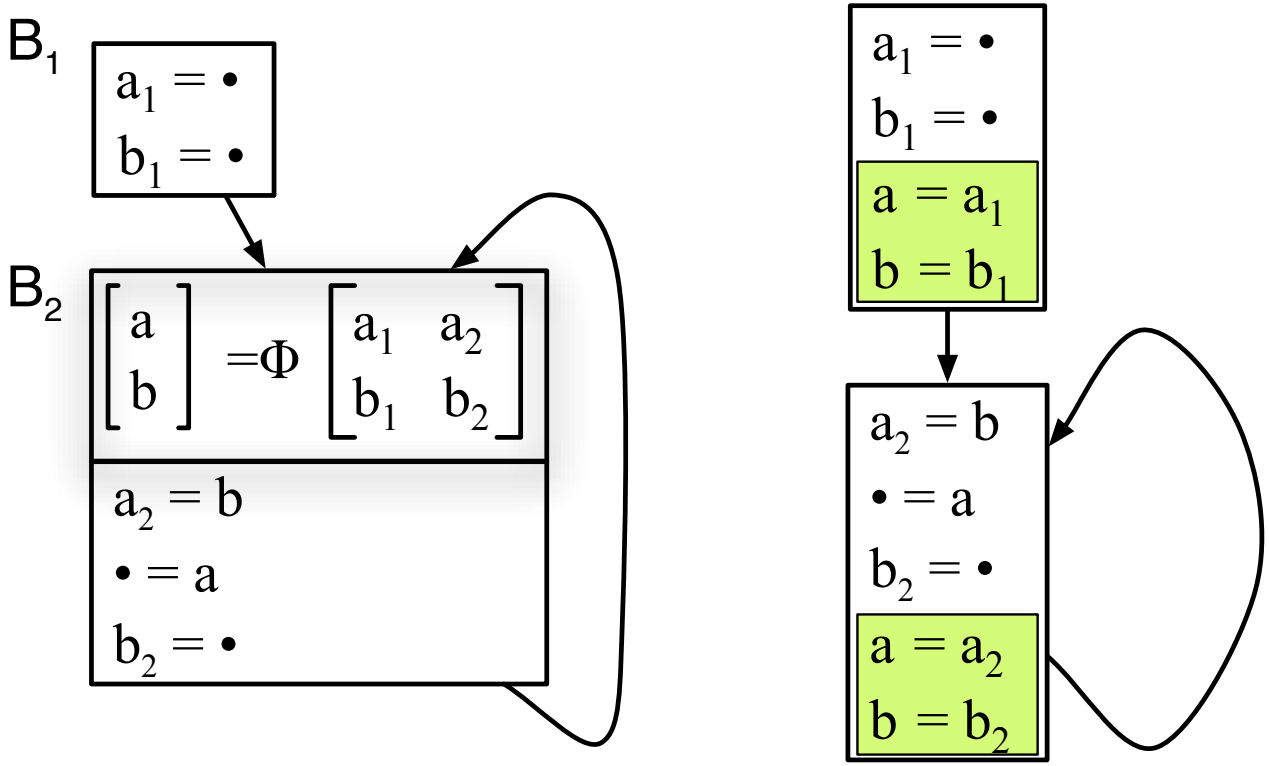
ϕ-functions and parallel copies





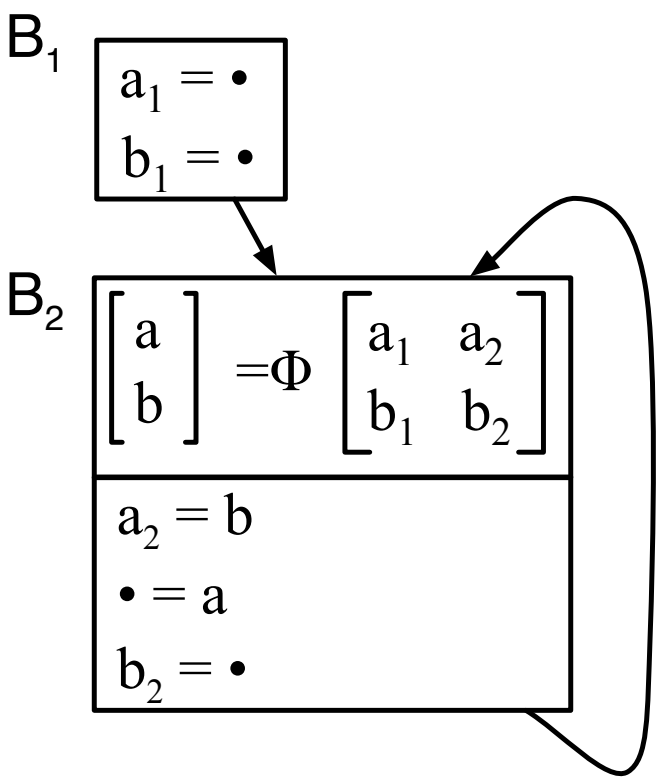
Classic SSA Elimination

- Classic algorithm due to Cytron *et al.* [3] replaces ϕ-functions with copy instructions.

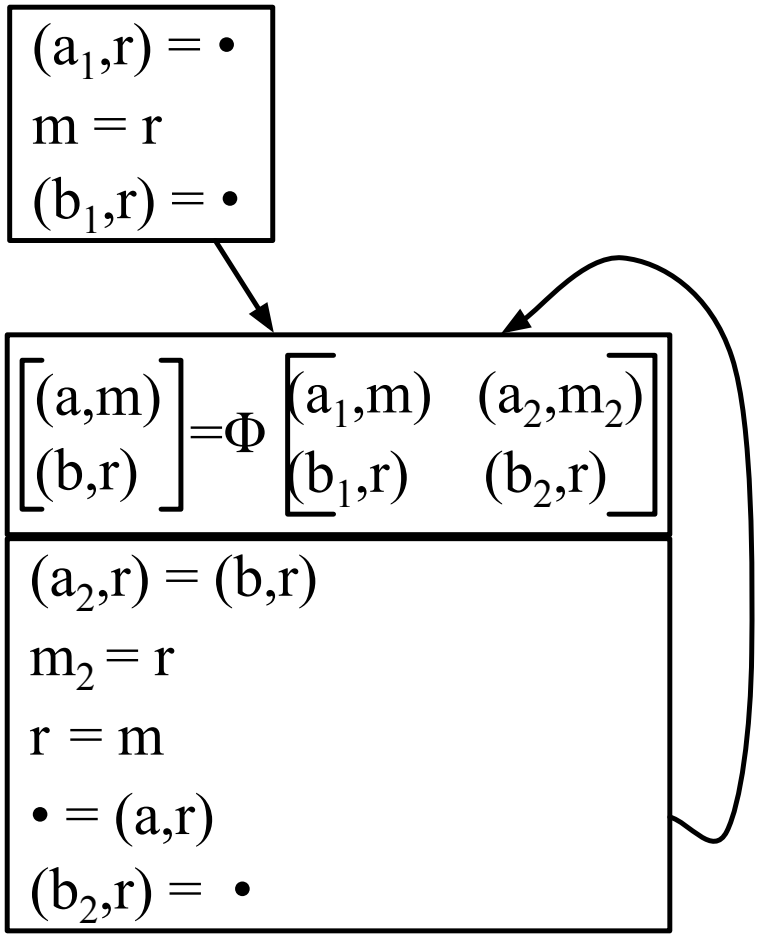


Outline

Colored SSA-Form

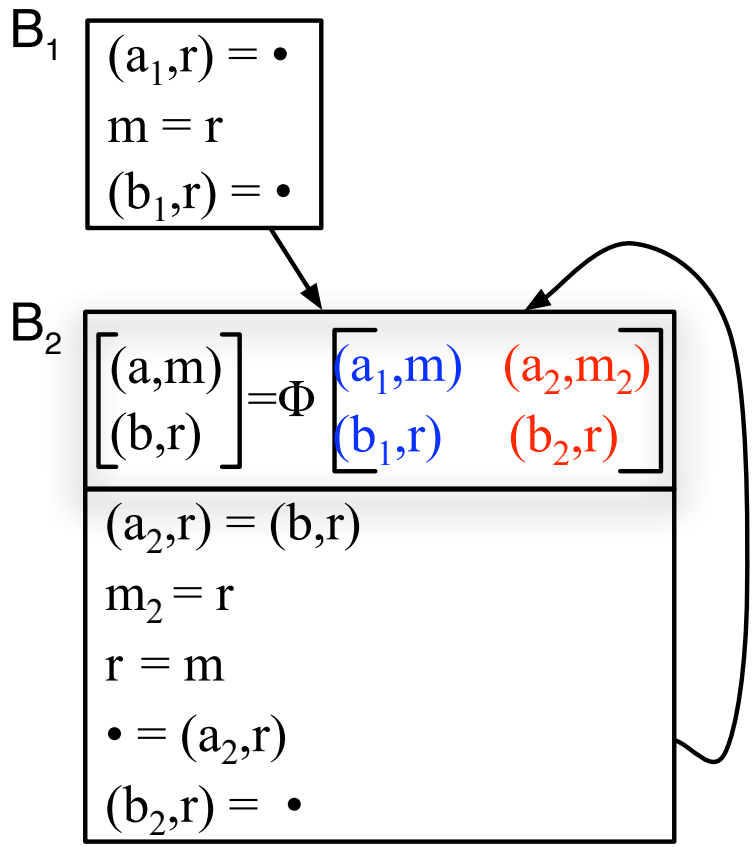


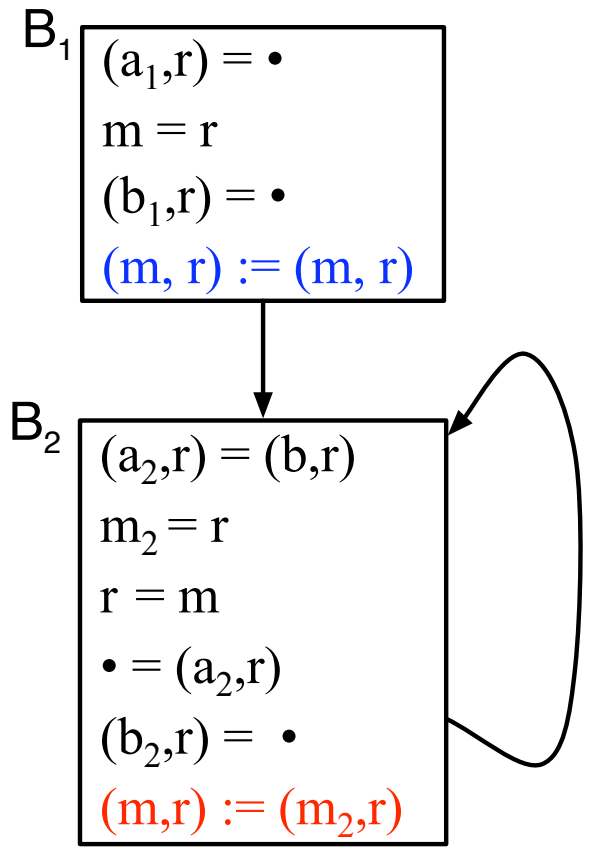
SSA-form program



Colored SSA-form program

SSA Elimination ≡ Implementing Parallel Copies





Location Transfer Graphs

- Parallel copy μ: $(l_1, \ldots, l_n) := (l_{1j}, \ldots, l_{nj})$.
- μ is **well defined** if locations in the left side are pairwise distinct.
- Locations $(l_1, l_2, \ldots)$ are either register $(r_1, \ldots, r_k)$ or memory $(m_1, m_2, \ldots)$.
- Parallel copies can be described as digraphs: we call them *Location Transfer Graphs*.
  - up to $2n$ vertices: $\{l_1, \ldots, l_n, l_{1j}, \ldots, l_{nj}\}$
  - n edges: (l_{ij}, l_i)

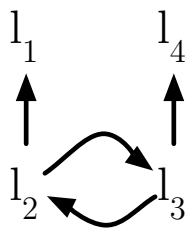
ϕ-functions and Windmills

- The in-degree of a vertex of a well defined LTG is at most one. This graph is called windmill.
- Each column of a ϕ-matrix determines a windmill.

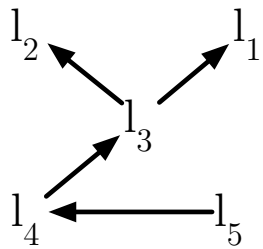
Φ-matrix

$$\begin{bmatrix} l_1 \\ l_2 \\ l_3 \\ l_4 \end{bmatrix} = \Phi \begin{bmatrix} l_2 & l_3 & l_4 \\ l_3 & l_3 & l_1 \\ l_2 & l_4 & l_2 \\ l_3 & l_5 & l_3 \end{bmatrix}$$

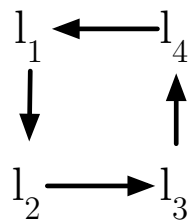
First Column



Second Column



Third Column



Problem Definition

- The language Seq has four types of instructions:
  - Copy: $r_1 := r_2$
  - Store: $m := r$
  - Load: $r := m$
  - Swap: $r_1 \oplus r_2$

SSA-elimination after Register Allocation

Instance: *a well defined parallel copy* μ.

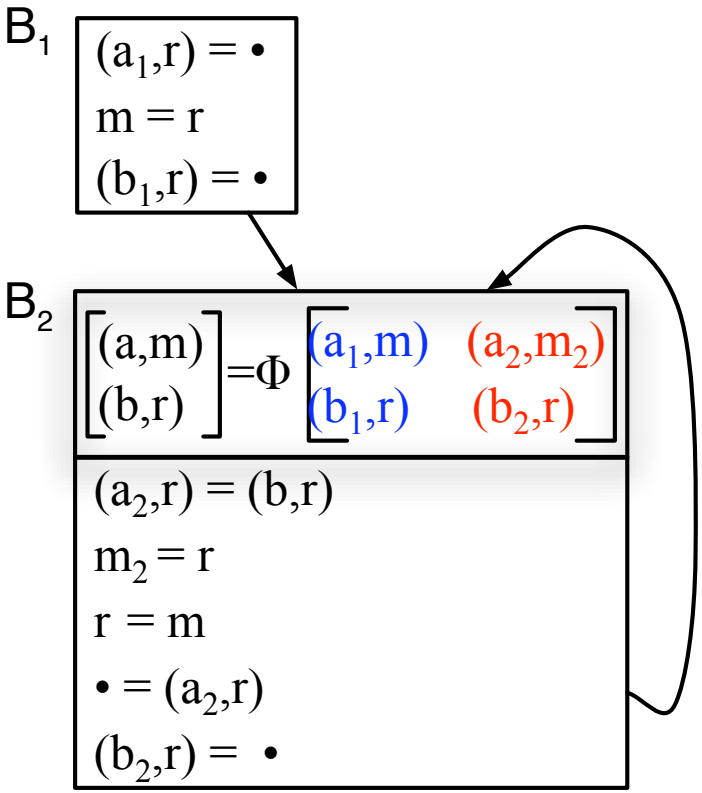
Problem: *find a* Seq *program* P *such that* P *is semantically equivalent to* μ.

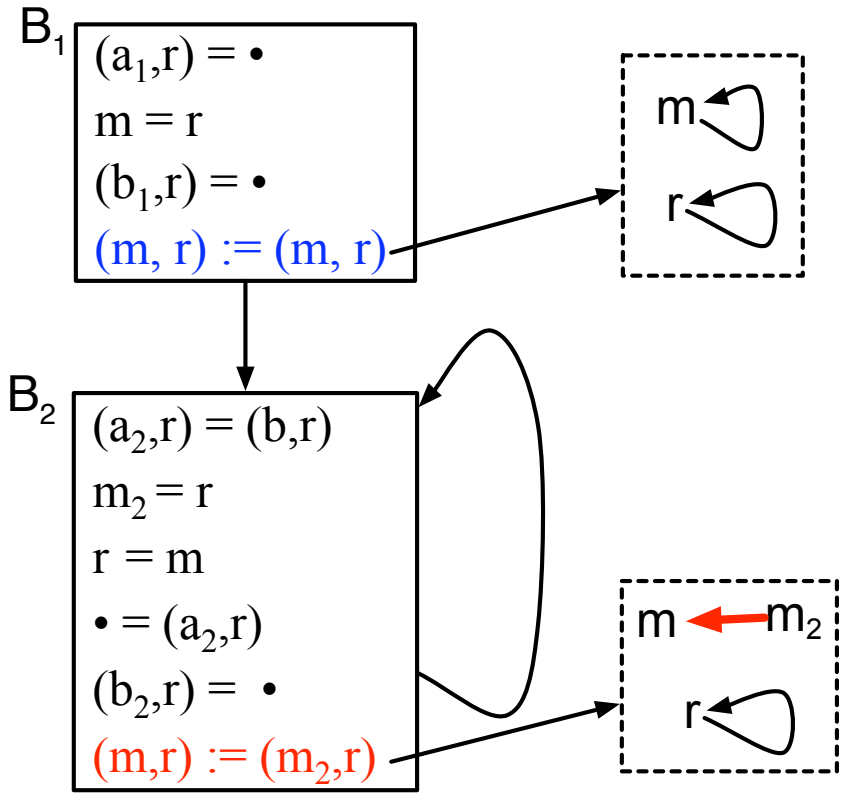
Memory Transfers

- The language Seq has only four types of instructions:
  - Copy: $r_1 := r_2$
  - Store: $m := r$
  - Load: $r := m$
  - Swap: $r_1 \oplus r_2$
- Seq has no memory copy, e.g $m_1 := m_2$ nor memory swap, e.g $m_1 \oplus m_2$
- To implement these instructions we need temporary registers:
  - $m_1 := m_2$ is equivalent to $r := m_2; m_1 := r$
  - $m_1 \oplus m_2$ is equivalent to $r_1 := m_1; r_2 := m_2; m_2 := r_1; m_1 := r_2$

Memory Transfers

Assuming an architecture with only one register r, how to implement the copy $m \leftarrow m_2$?

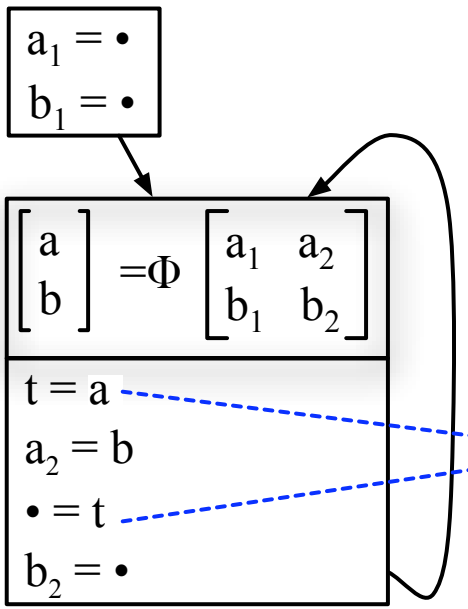




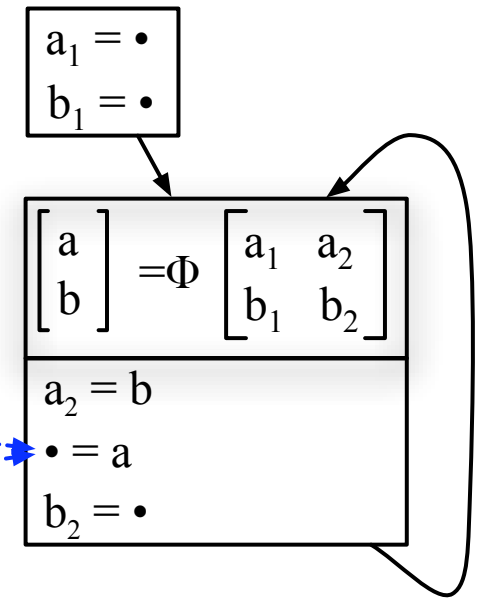
Our solution

- Let $\equiv$ be the smallest equivalence relation on variables in the entire procedure such that for each ϕ-function $v_i = \phi(v_{i1}, v_{i2}, \ldots, v_{im})$, we have that $v_i \equiv v_{i1} \equiv v_{i2} \equiv \ldots \equiv v_{im}$.
- If $v \equiv v'$, then we say that v and v' are ϕ-related.
- Compiler optimizations might cause ϕ-related variables to interfere.
- Conventional Static Single Assignment form [4] is a flavor of SSA-form where ϕ-related variables do not interfere;
- If the ϕ-related variables do not interfere, we can assign them to the same memory slot.
- There exist polynomial time algorithms to convert SSA-form programs into CSSA-form programs [1, 2, 4].

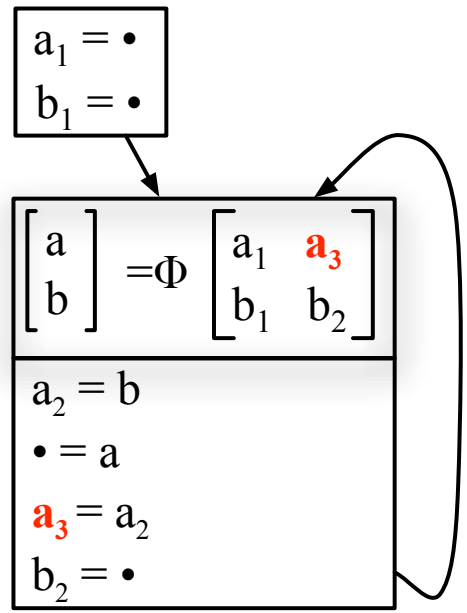
More on CSSA-form



The original algorithm of Cytron *et al.* produces programs in CSSA-form.



Compiler optimizations cause phi-related variables to interfere.



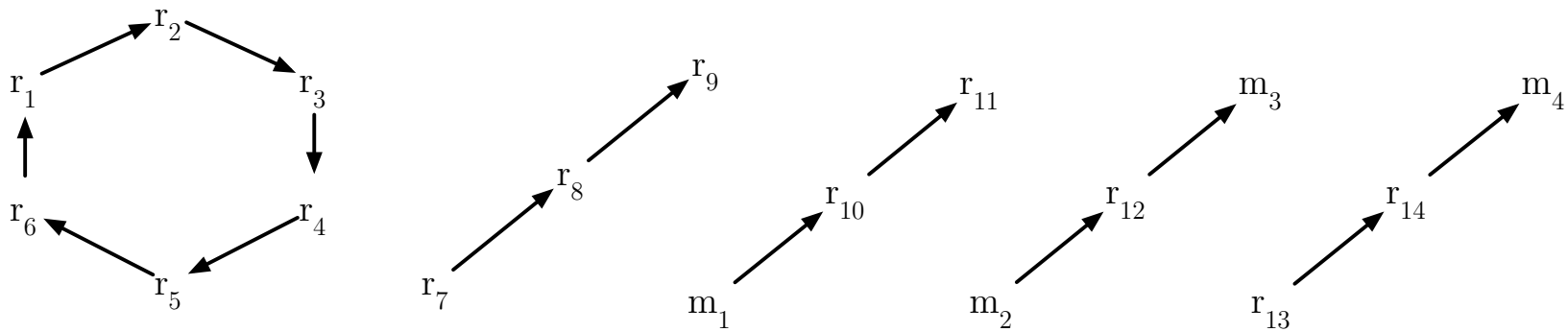
We can recover CSSA property via almost linear time algorithm.

Frugal Register Allocator

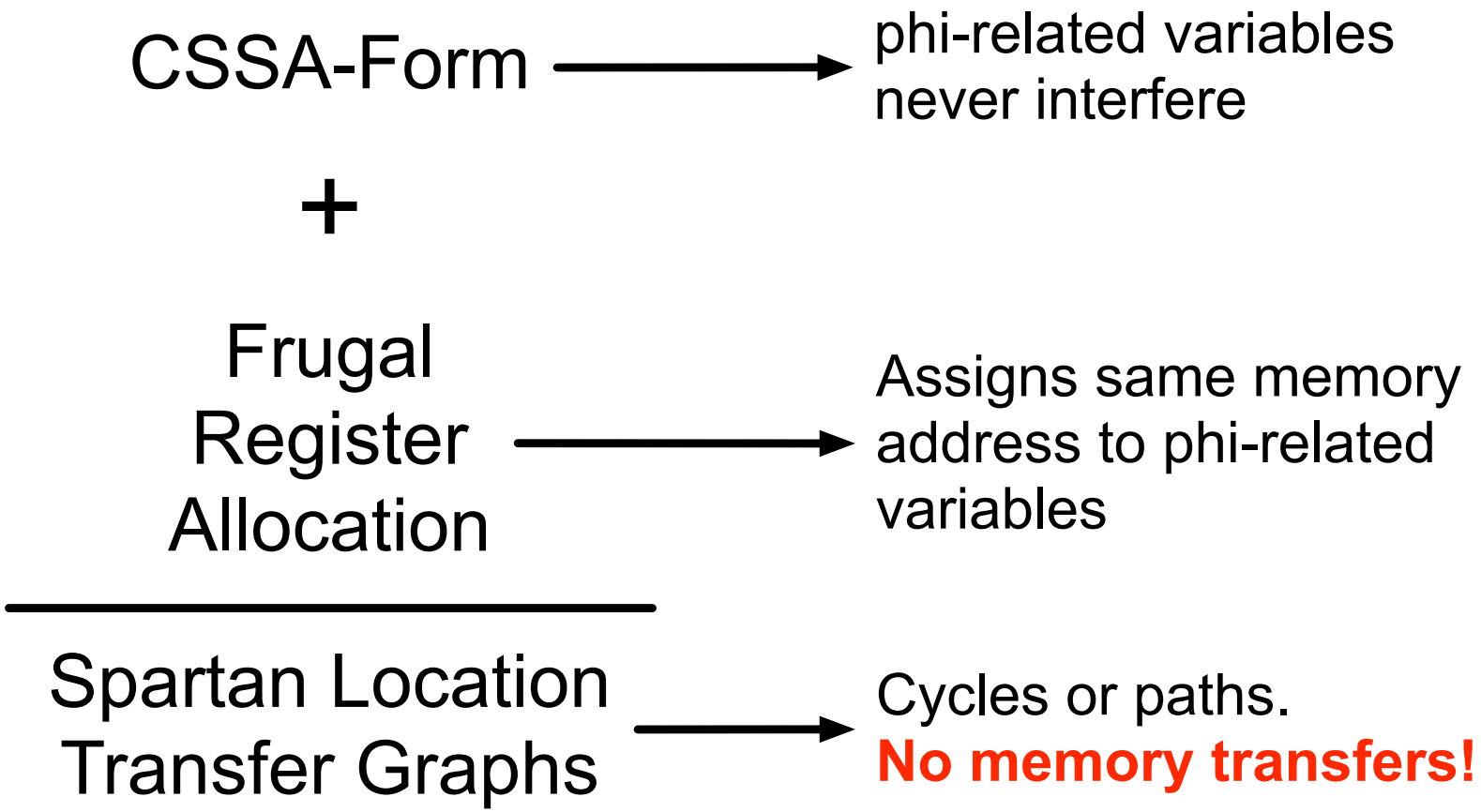
- A register allocator is frugal if and only if it assigns the same memory address to spilled variables that are ϕ-related.
- Frugal register allocation might produce incorrect code in general SSA-form programs.
  - Why? Because ϕ-related variables might interfere.
- However, it is always correct in CSSA-form programs.
  - Why? Because ϕ-related variables never interfere.

Spartan Location Transfer Graphs

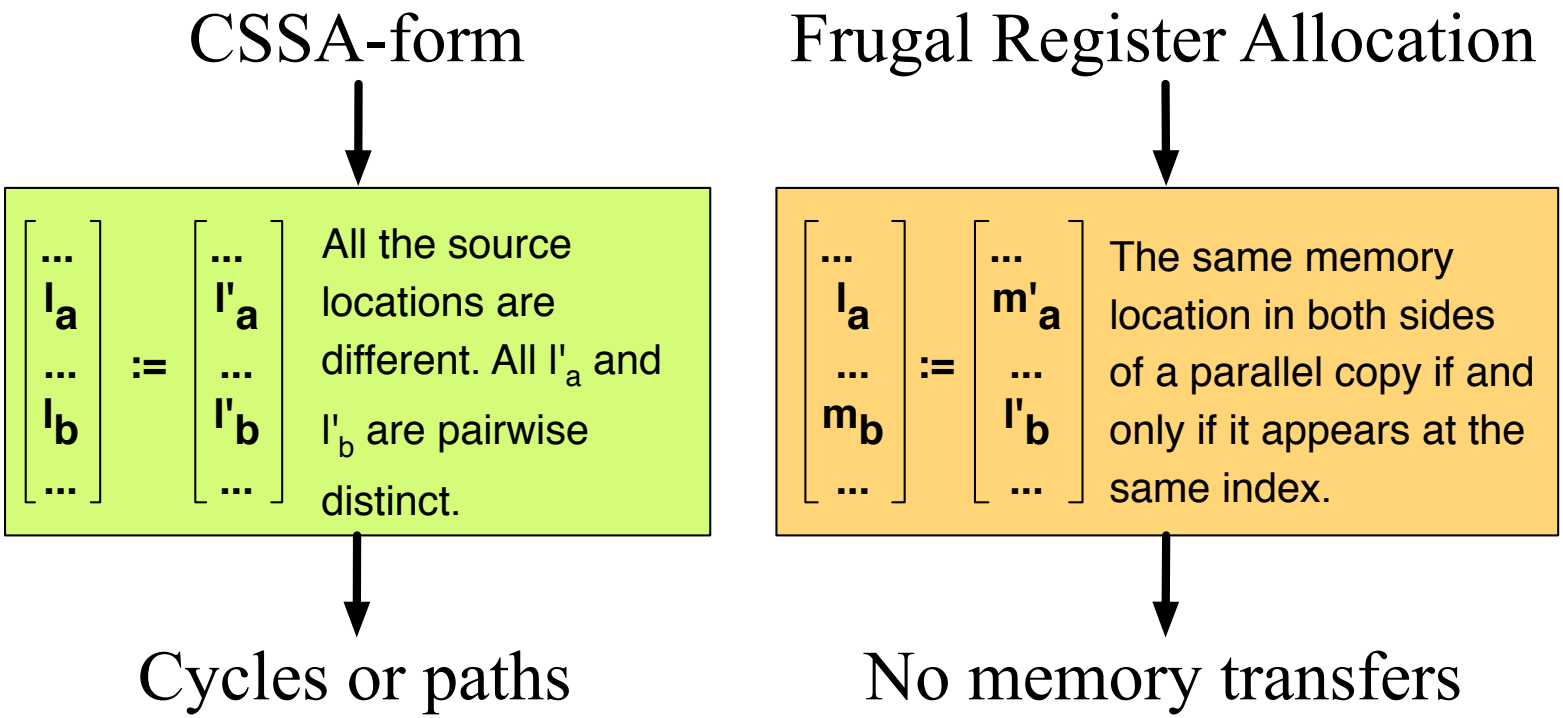
- Either a cycle or a path.
- Locations in cycles are always registers, and memory locations can only exist in the extremity of paths.
  - First copy may be a store.
  - Last copy may be a load.



The important result



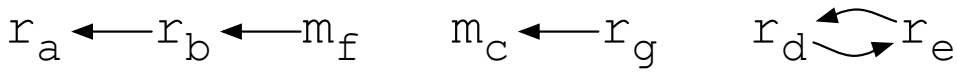
Frugal RA + CSSA = ...



SSA Elimination

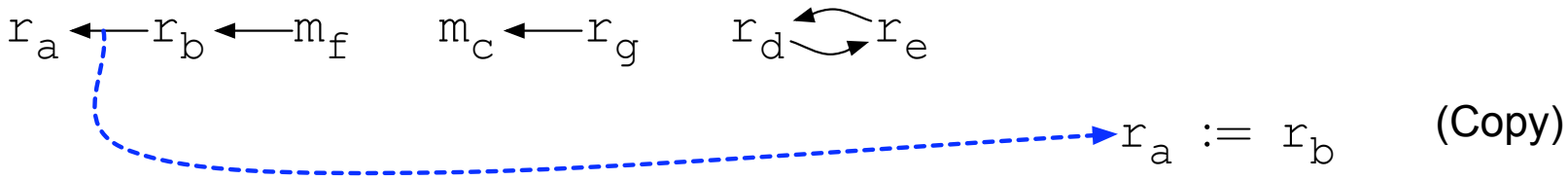
- Algorithm **ImplementComponent**
  - **Input**: Connected spartan location transfer graph G
  - **Output**: Seq program I
  - **If** G is a path $(l_1, r_2), \ldots, (r_{n-2}, r_{n-1}), (r_{n-1}, l_n)$ **then**
    - $I = (l_n := r_{n-1}; r_{n-1} := r_{n-2}; \ldots; r_2 := l_1)$
  - **else if** G is a cycle $(r_1, r_2), \ldots, (r_{n-1}, r_n), (r_n, r_1)$ **then**
    - $I = (l_n \oplus l_{n-1}; l_{n-1} \oplus l_{n-2}; \ldots; l_2 \oplus l_1)$
- Algorithm **ImplementSpartan**
  - **Input**: Connected spartan location transfer graph G
  - **Output**: Seq program I
  - **If** G has connected components $C_1, \ldots, C_m$ **then**
    - $I =$ **ImplementComponent**$(C_1); \ldots;$ **ImplementComponent**$(C_m);$

Example of SSA Elimination



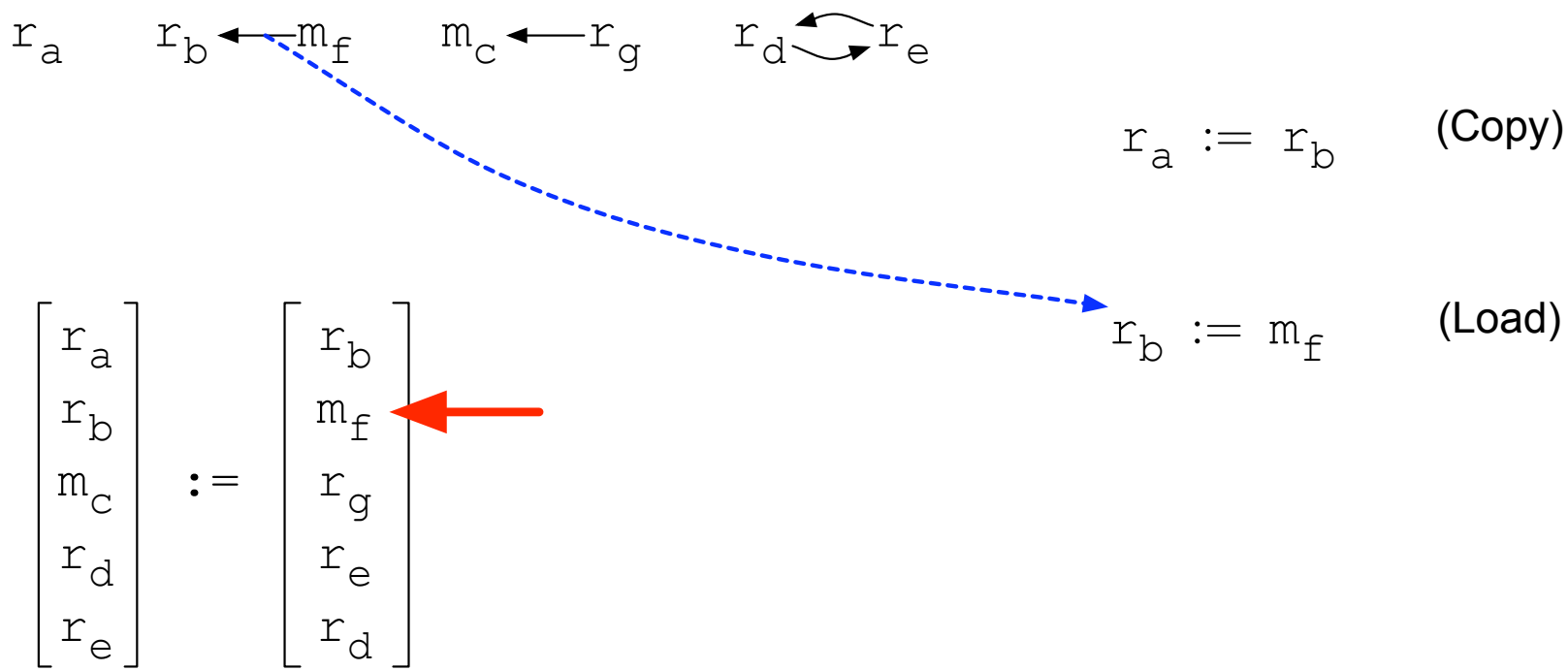
$$\begin{bmatrix} r_a \\ r_b \\ m_c \\ r_d \\ r_e \end{bmatrix} := \begin{bmatrix} r_b \\ m_f \\ r_g \\ r_e \\ r_d \end{bmatrix}$$

Example of SSA Elimination

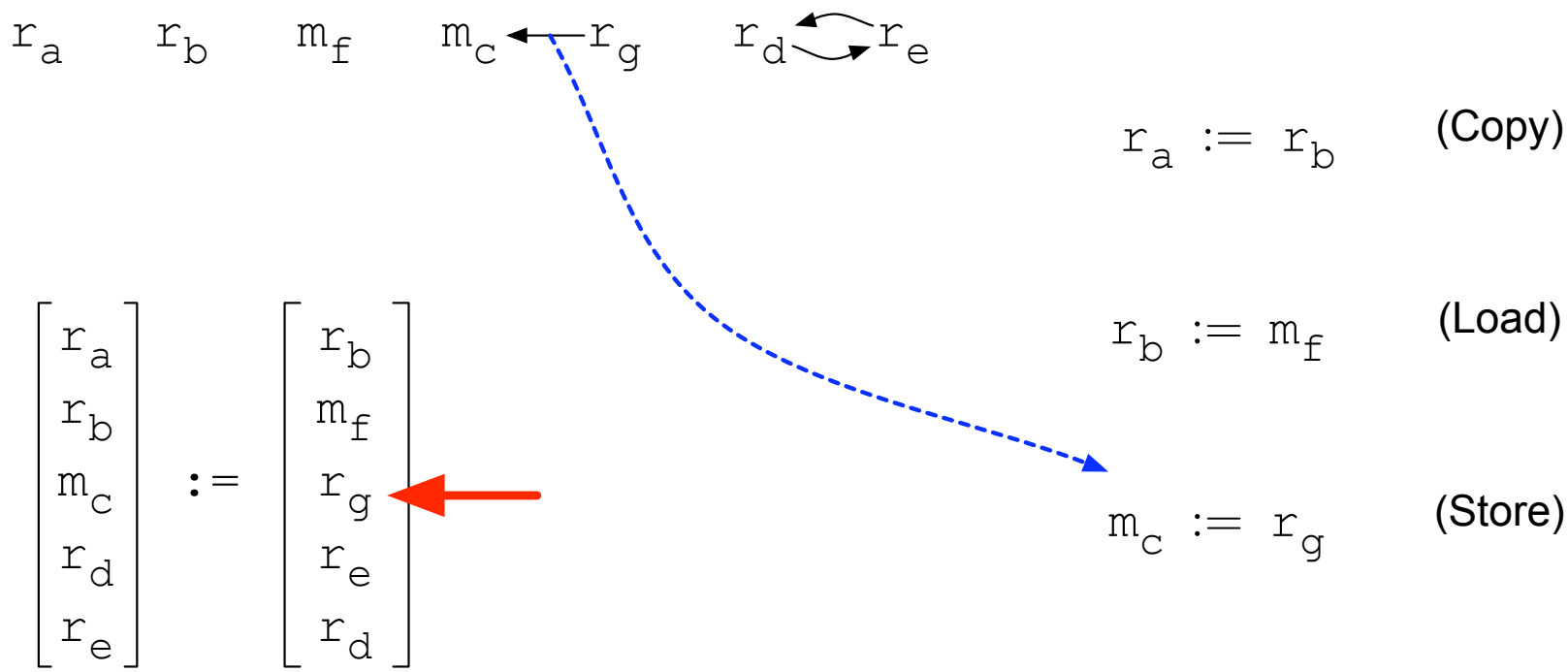


$$\begin{bmatrix} r_a \\ r_b \\ m_c \\ r_d \\ r_e \end{bmatrix} := \begin{bmatrix} r_b \\ m_f \\ r_g \\ r_e \\ r_d \end{bmatrix}$$

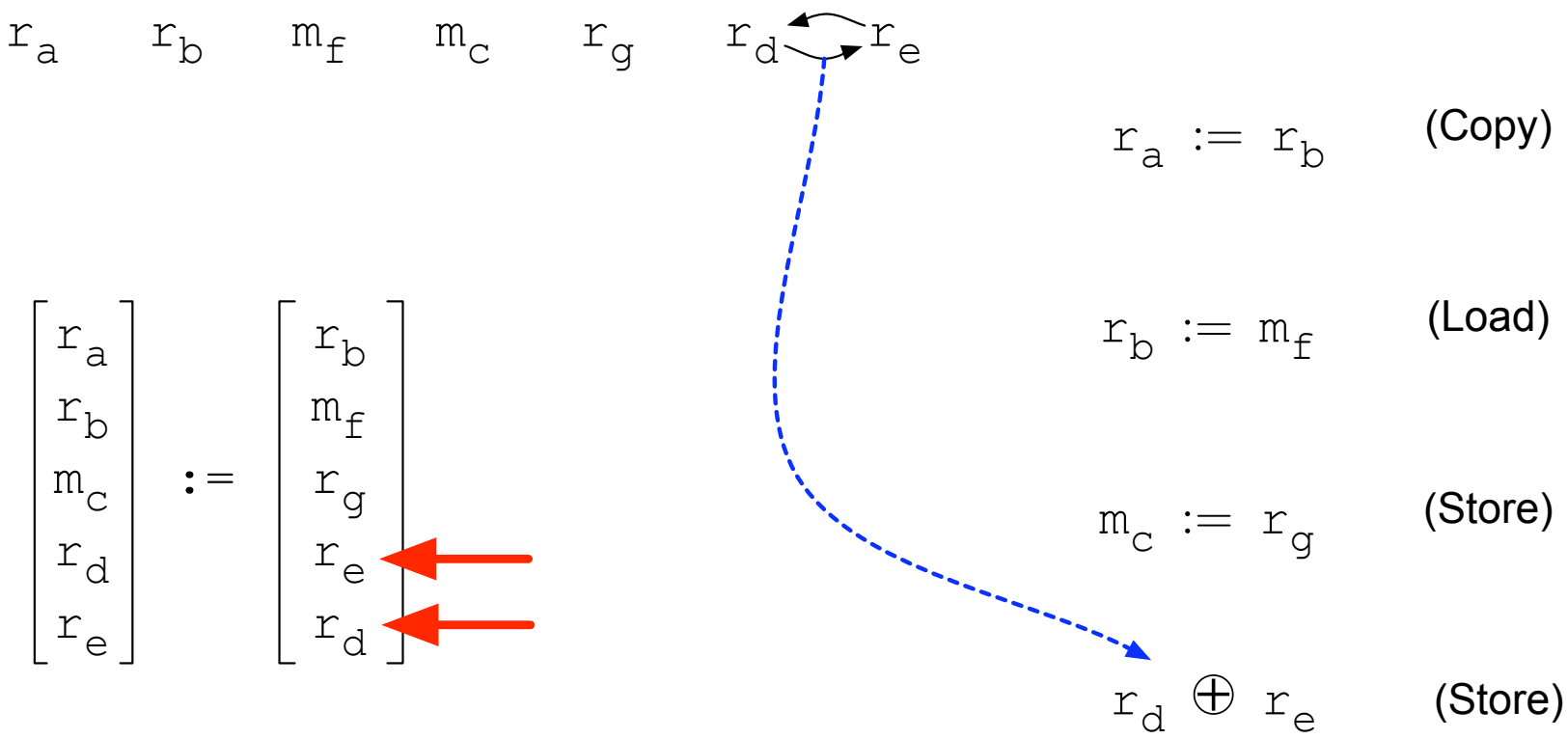
Example of SSA Elimination



Example of SSA Elimination

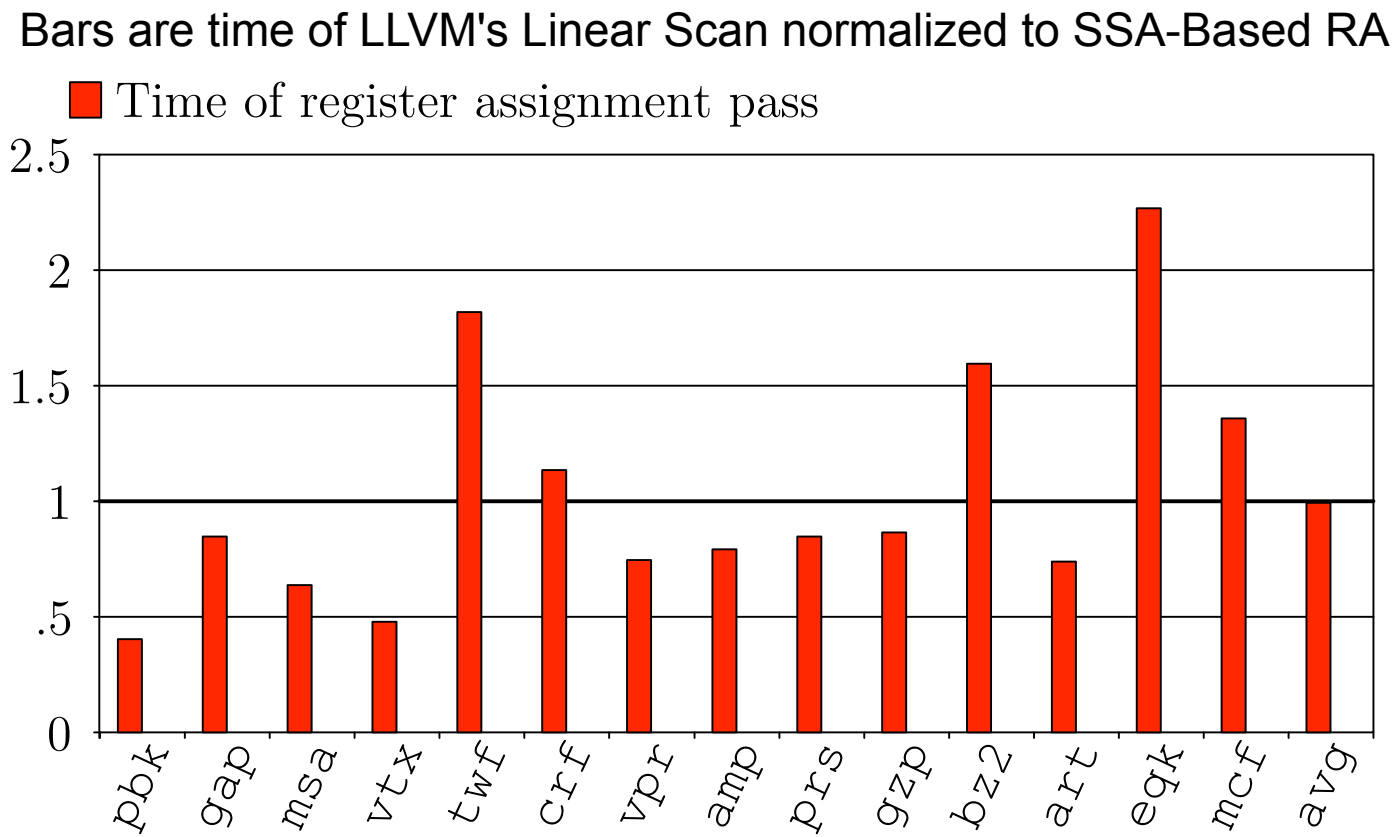


Example of SSA Elimination

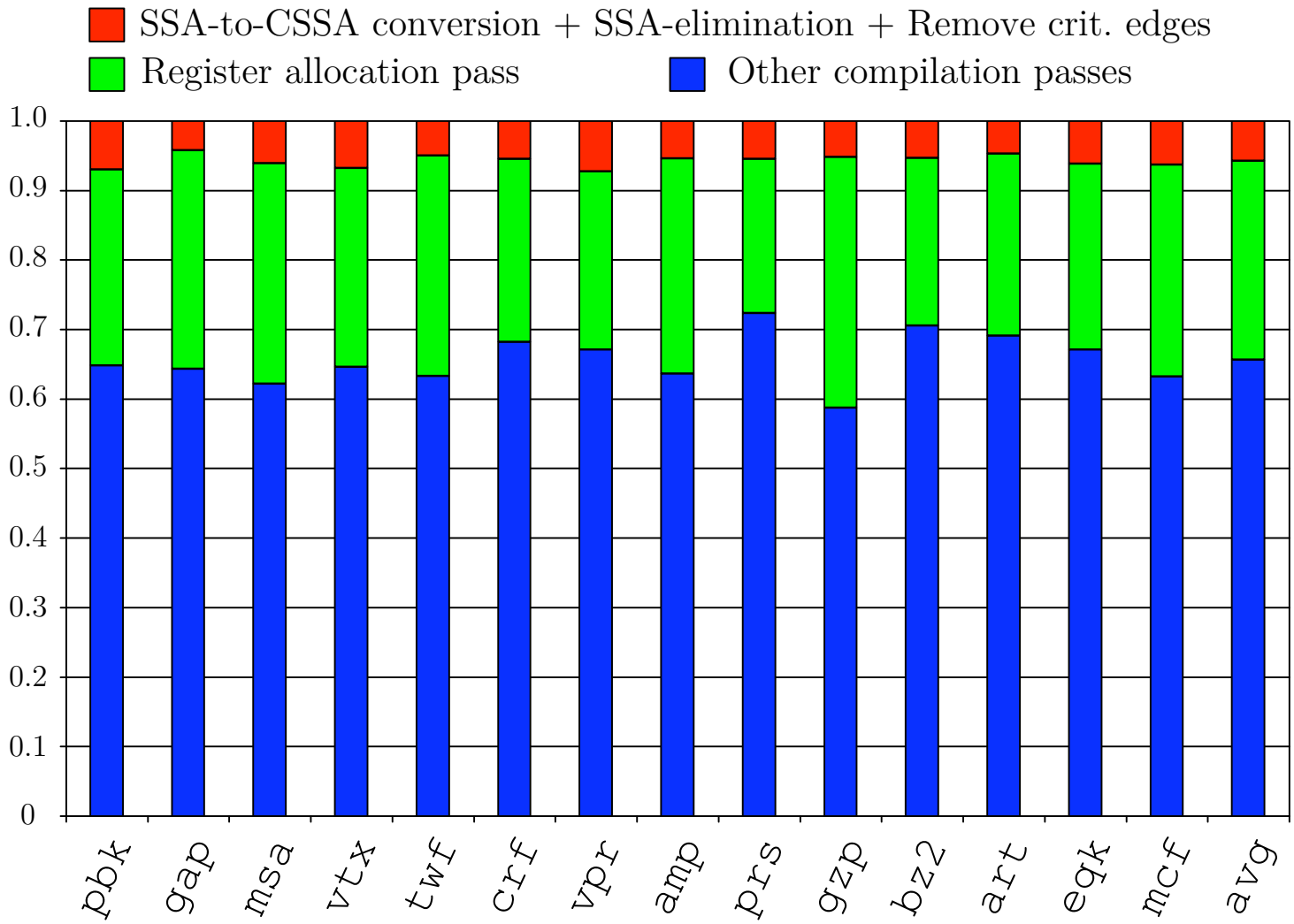


Outline

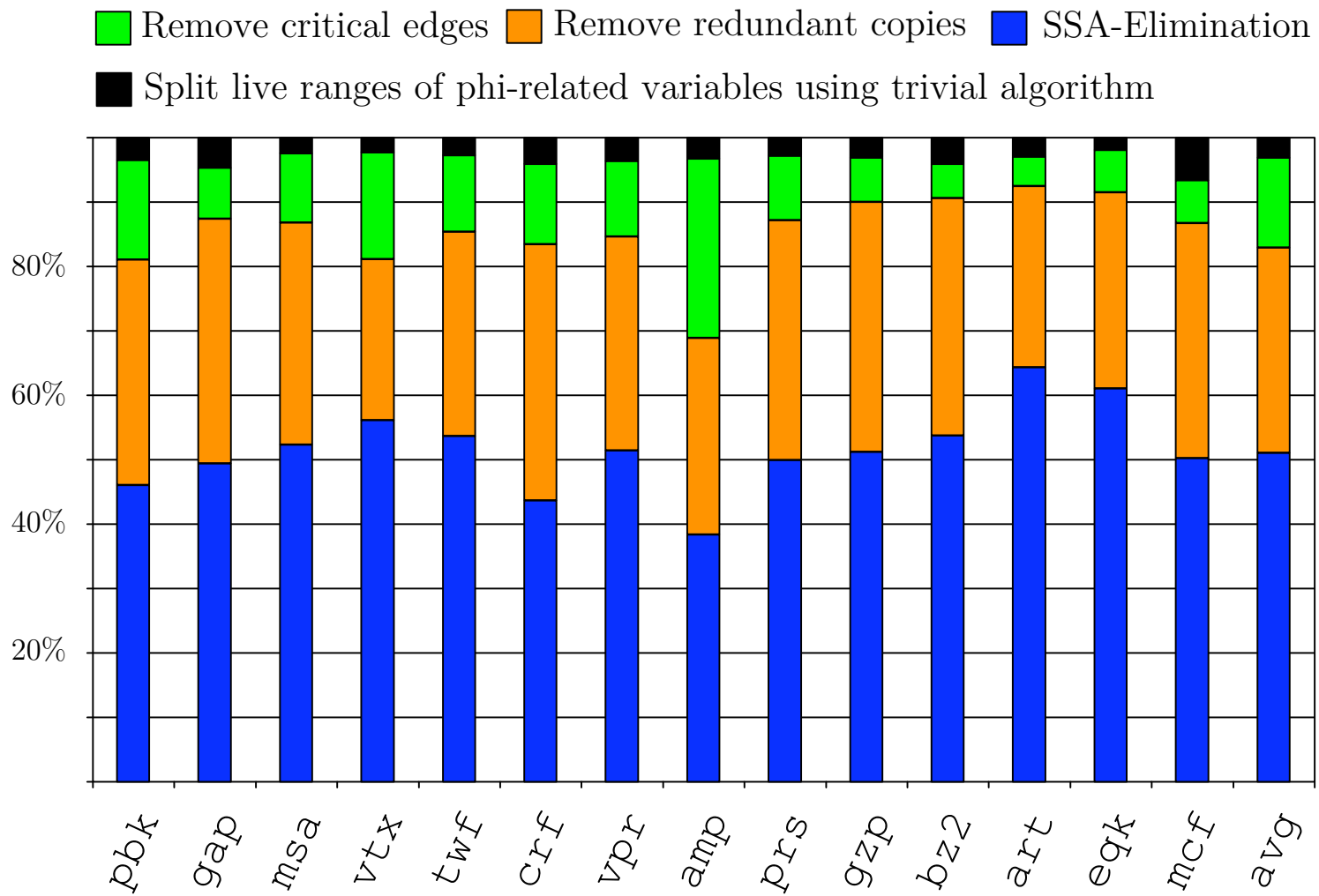
Compilation Time



Time Overhead of SSA-Elimination

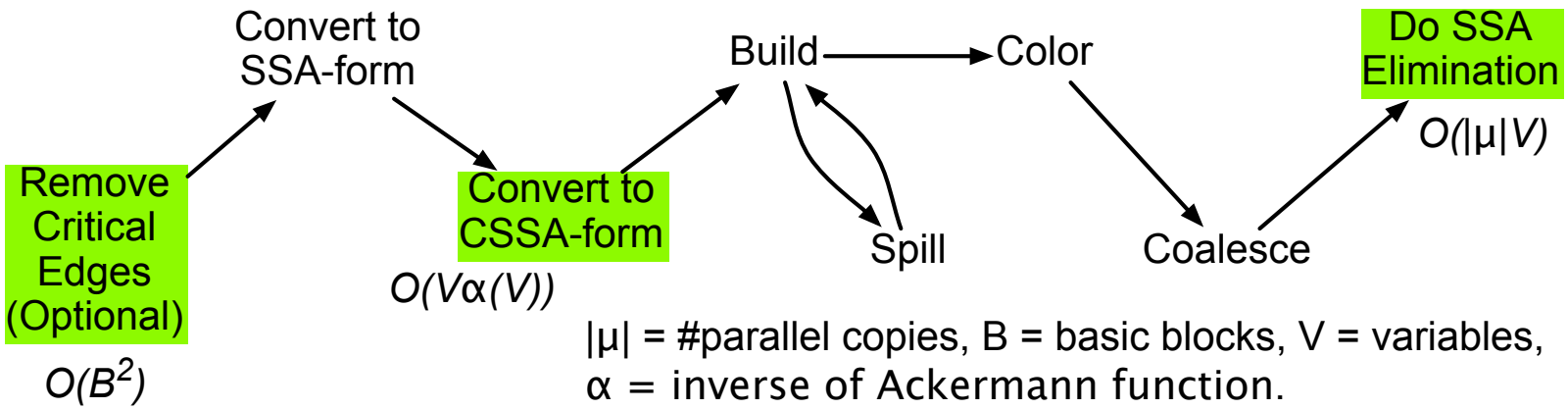


Time taken by Different Phases of SSA Elimation



Conclusion

- CSSA-form solves memory transfer problem.
- It is easy to convert a program from SSA into CSSA. See, for instance, *Revisiting Out-of-SSA Translation for Correctness, Code Quality, and Efficienty*, to be presented today, at 10:00 AM (in Seattle, US).
- The Big-Picture:




Benoit Boissinot, Alain Darte, Fabrice Rastello, Benoit Dupont de Dinechin, and Christophe Guillon.

Revisiting out-of-SSA translation for correctness, code quality, and efficienty.

In *CGO*, pages XX–XX, 2009.


Zoran Budimlic, Keith D. Cooper, Timothy J. Harvey, Ken Kennedy, Timothy S. Oberg, and Steven W. Reeves.

Fast copy coalescing and live-range identification.

In *PLDI*, pages 25–32. ACM Press, 2002.


Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck.

An efficient method of computing static single assignment form.

In *POPL*, pages 25–35, 1989.


Vugranam C. Sreedhar, Roy Dz ching Ju, David M. Gillies, and Vatsa Santhanam.

Translating out of static single assignment form.

In *SAS*, pages 194–210. Springer-Verlag, 1999.