Punctual Coalescing

Fernando Magno Quintão Pereira

Register Coalescing

- Register coalescing is an optimization on top of register allocation. The objective is to map both variables used in a copy instruction to the same register.

```
int a = 0;
int b = a;
print(b);
```

a ← R1
b ← R2

```
R1 = 0;
R2 = R1;
print(R2);
```

a ← R1
b ← R1

R1 = 0;
~~R1 = R1;~~
print(R1);

The Importance of Register Coalescing

Size reduction

Execution speedup

Energy saving

Punctual Coalescing

- *Register assignment* technique based on *puzzle solving*.
- It deals with *aliased registers*, e.g *x86*.
  - No worries: I will define these concepts ☺
- It is *O(K*I)*, where *K* = number of registers, and *I* is number of instructions.
- **Strength**: very fast!
- **Weakness:** it is a local solution.

Register Aliasing

Two registers alias when an assignment to one may change the value of the other.

Ex. X86's GP regs:

AX		***BX***		***CX***		***DX***	
AH	*AL*	*BH*	*BL*	*CH*	*CL*	*DH*	*DL*

Ex. PowerPC's doubles:

D0			***D15***	
F10	*F1*	…	*F30*	*F31*

And also ARM, UltraSparc, ST240, etc

Register Aliasing

- Register aliasing allows the register allocator to keep more variables in registers.
  - E.g.: two floats in the same register.
- But finding a register assignment that minimizes the number of registers taken is NP-complete.
  - Lee *et al*: Aliased register allocation for straight-line programs is NP-complete, TCS, 2008

Register allocation: fit the bars on the boxes.

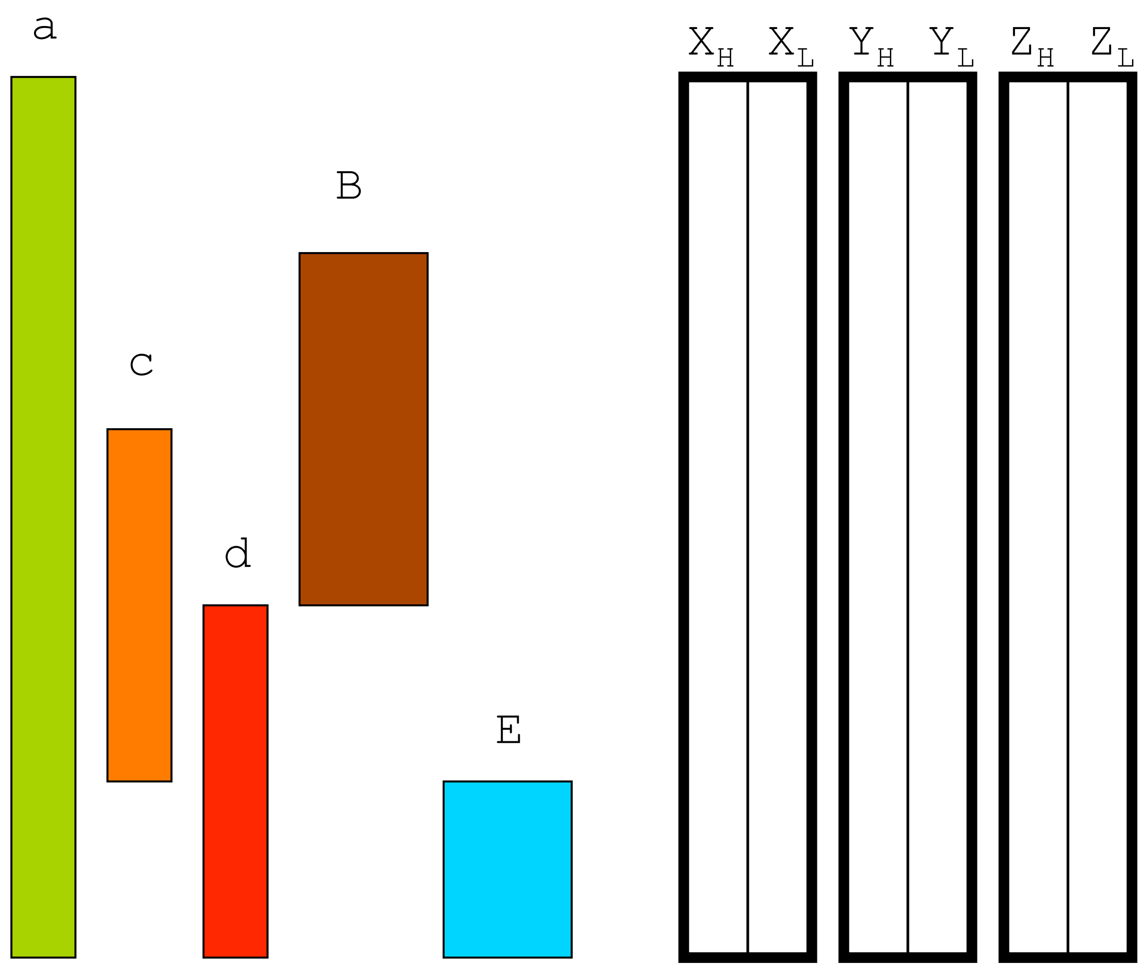
Aliased register allocation: Bars may have different widths.

Program

```
a = •

B = •

c = •

d = B

E = c

• = a, d, E
```

Live Ranges

Registers



Complexity result: optimal aliased register allocation is NP-complete.

Program

$X_H = \bullet$

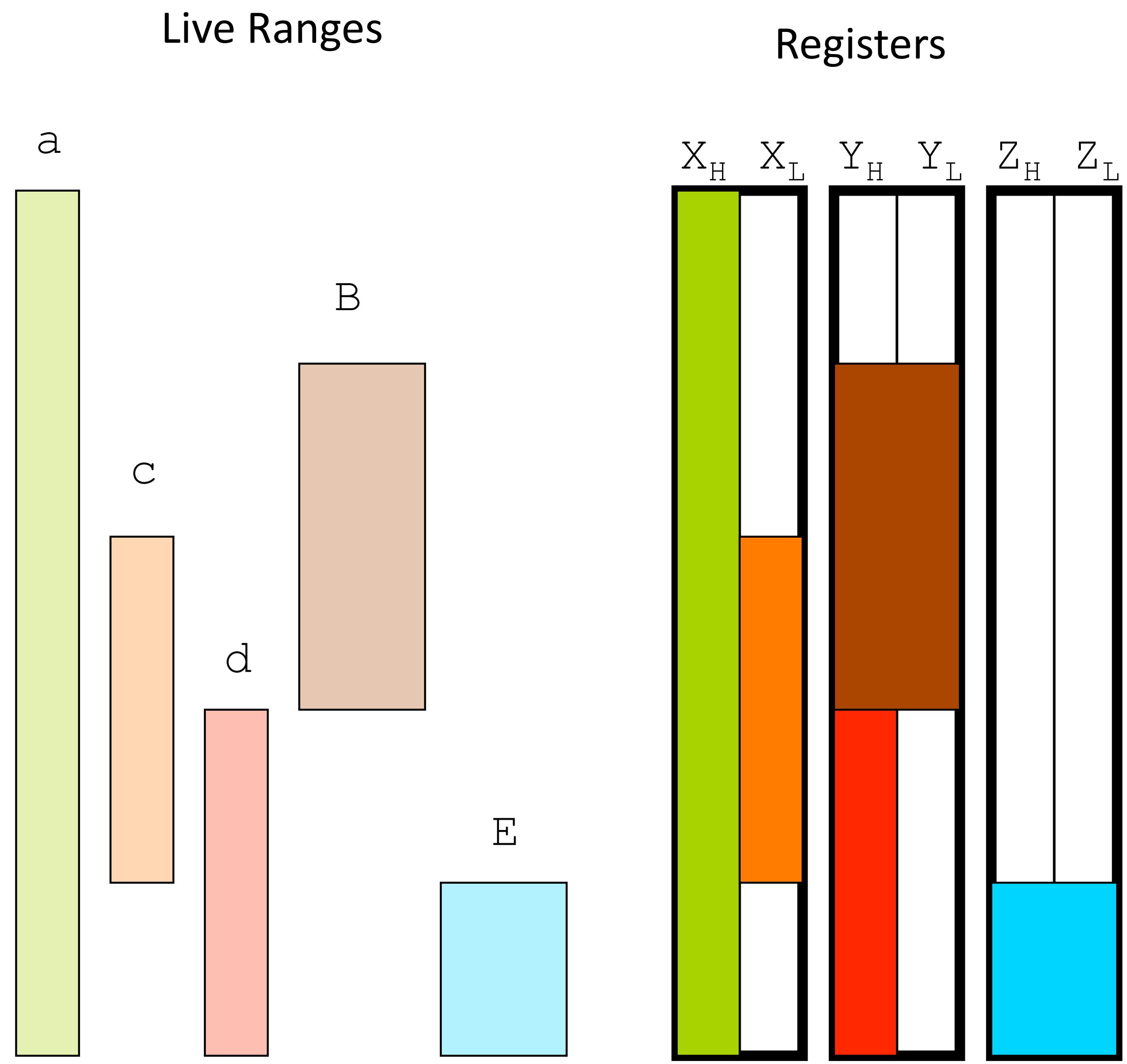
$Y = \bullet$

$X_L = \bullet$

$Y_H = Y$

$Z = X_L$

$\bullet = X_H, Y_H, Z$



Elementary form: split variables between each pair of consecutive instructions.

Elementary Program

$a_1 = \bullet$

$(a_2) = (a_1)$

$B_2 = \bullet$

$(a_3, B_3) = (a_2, B_2)$

$c_3 = \bullet$

$(a_4, B_4, c_4) = (a_3, B_3, c_3)$

$d_4 = B_4$

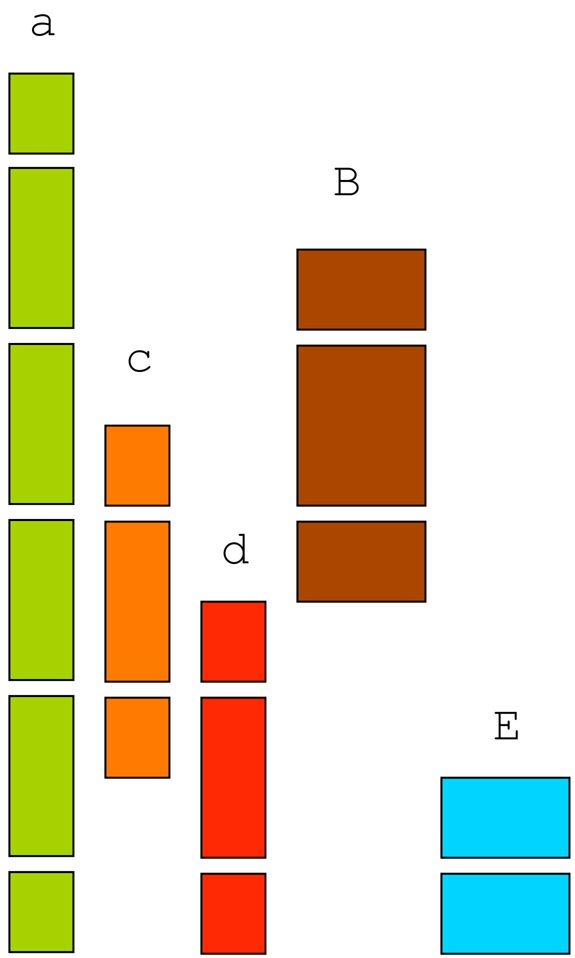
$(a_5, c_5, d_5) = (a_4, c_4, d_4)$

$E_5 = c_5$

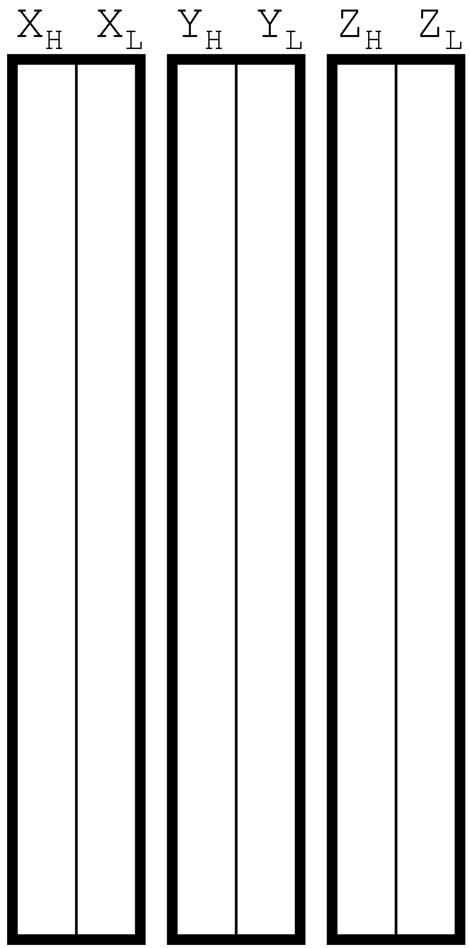
$(a_6, c_6, E_6) = (a_5, c_5, E_5)$

$\bullet = a_6, d_6, E_6$

Live Ranges



Registers



Complexity result: aliased register allocation has polynomial time solution for elementary form programs.

Elementary Program

$X_H = \bullet$

$Y = \bullet$

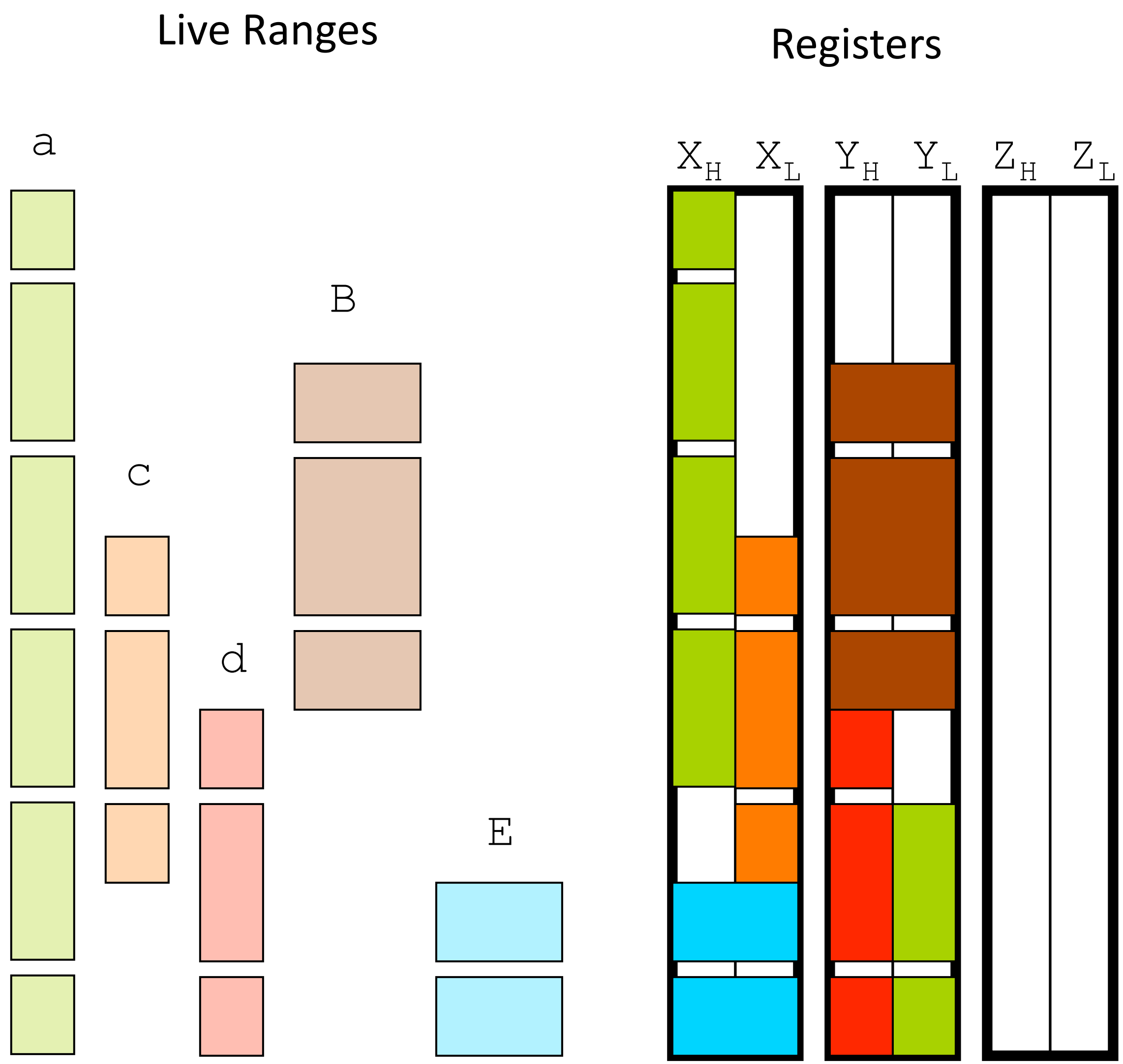
$X_L = \bullet$

$Y_H = Y$

$\boxed{\mathbf{Y_L = X_H}}$ **Copy**

$X = X_L$

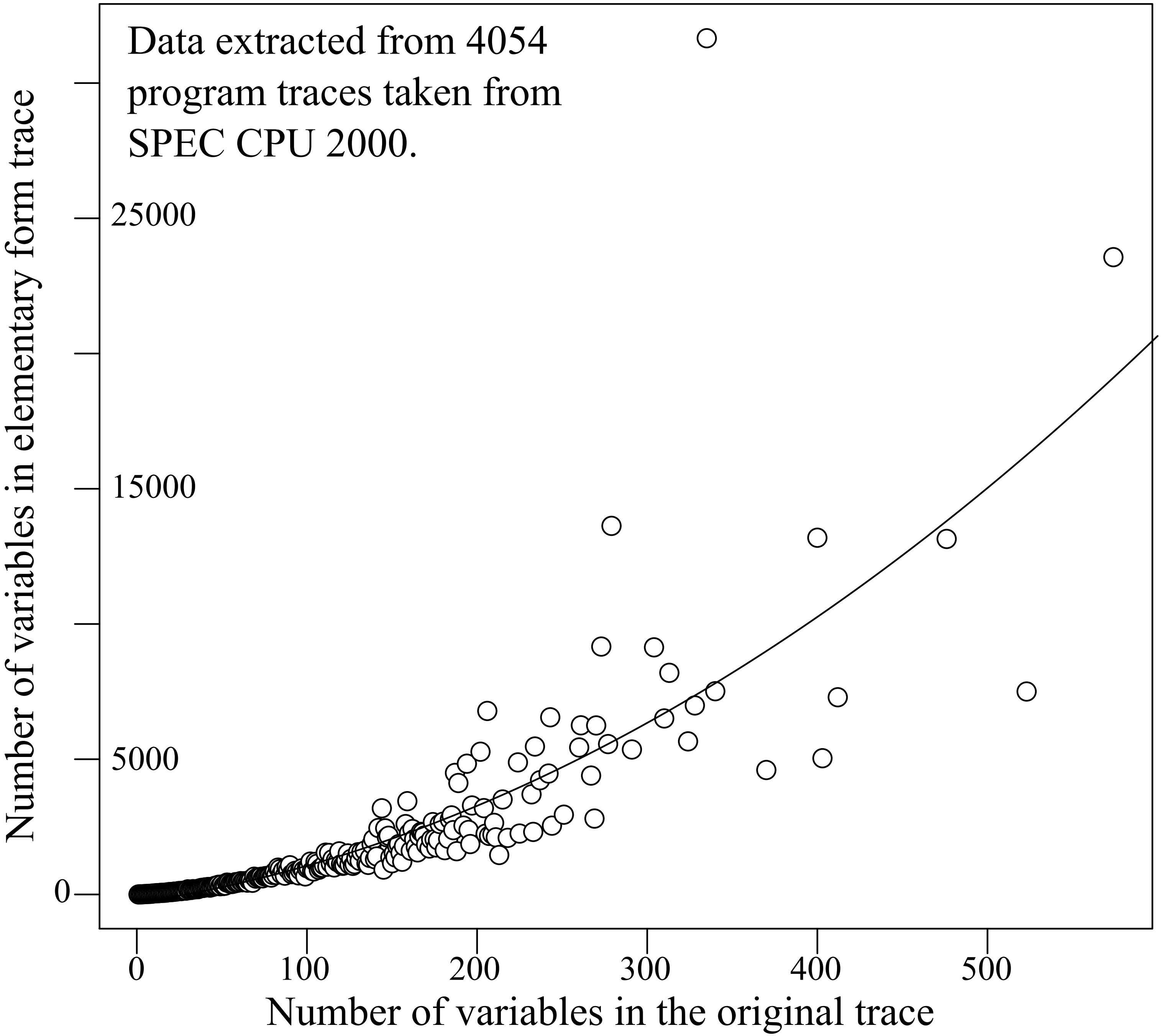
$\bullet = Y_L, Y_H, X$



Elementary Form = Slow RA?

- If a program has $O(V)$ variables and $O(I)$ instructions, then the elementary form program has $O(V*I)$ variables.
- If we assume that each instruction defines a variable, we have $O(V*I) = O(V^2)$.
- Many graph coloring based register allocators are $O(V^2)$.
- $O(V^4)$ Is just too slow....
- We have a $O(K*I)$ approach, where K is the number of registers!

Growth in number of variables



Our solution: traverse the program from top to bottom using the solution of previous puzzles to guide the solution of the next one to be solved.

$a_1 = \bullet$

$(a_2) = (a_1)$

$B_2 = \bullet$

$(a_3, B_3) = (a_2, B_2)$

$c_3 = \bullet$

Allocated!!!

$(a_4, B_4, c_4) = (a_3, B_3, c_3)$

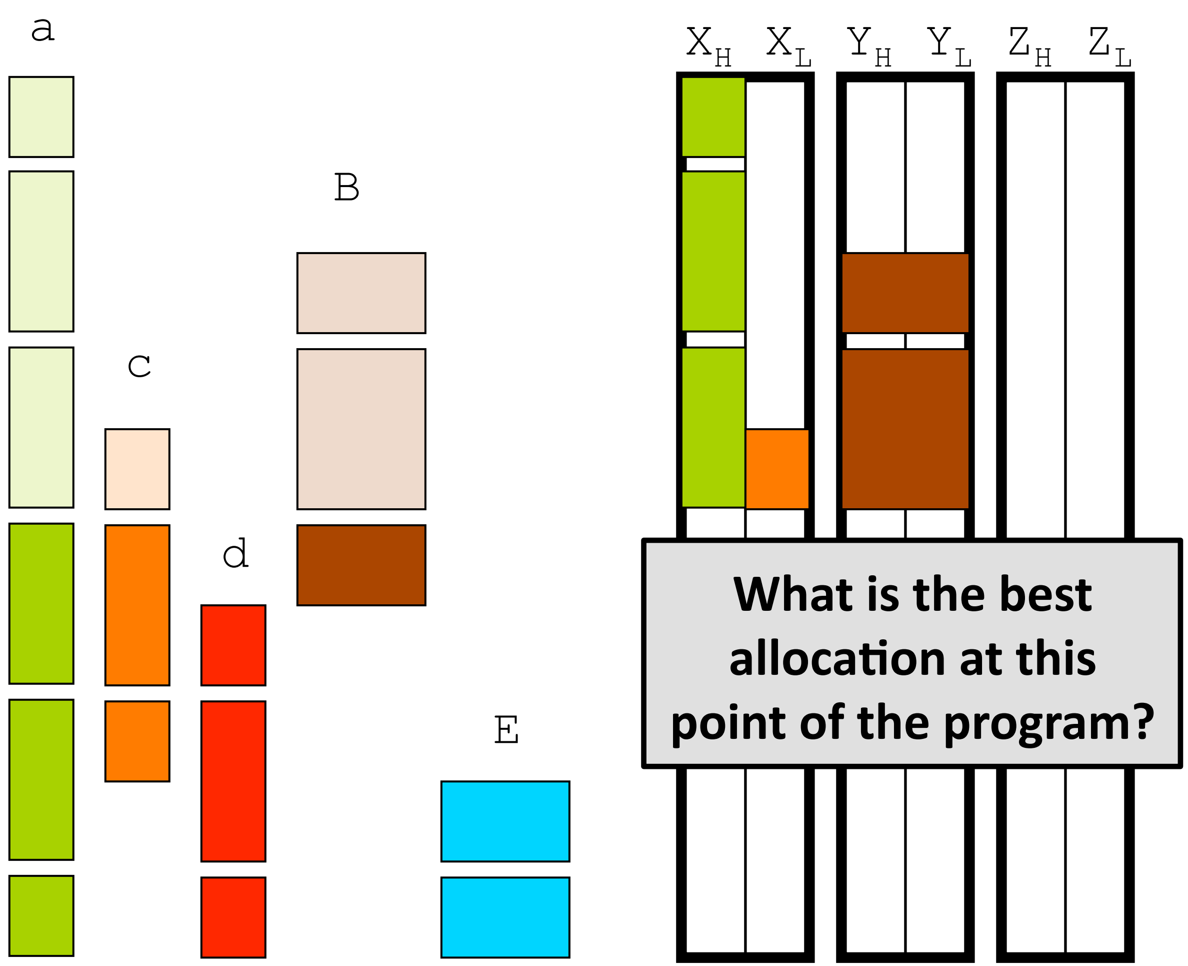
$d_4 = B_4$

$(a_5, c_5, d_5) = (a_4, c_4, d_4)$

$E_5 = c_5$

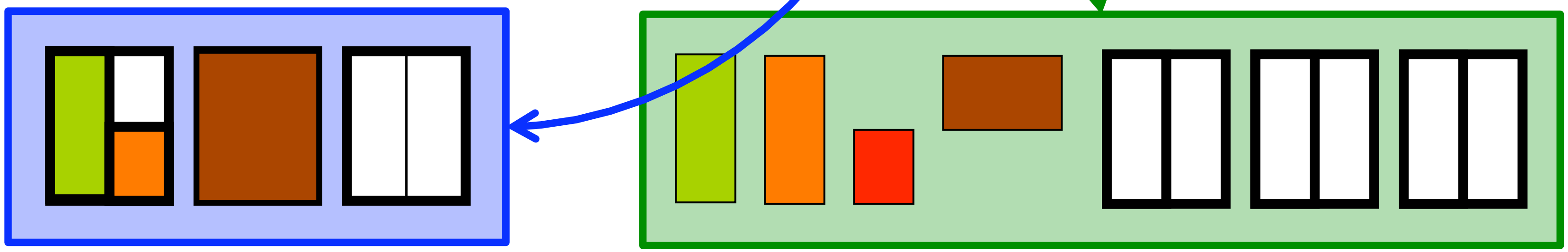
$(a_6, c_6, E_6) = (a_5, c_5, E_5)$

$\bullet = a_6, d_6, E_6$



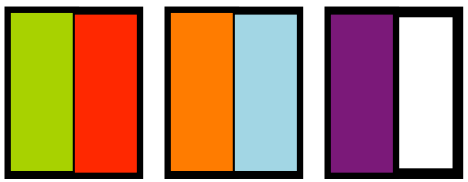
Punctual Coalescing

- **Instance**: two consecutive puzzles, p_1 and p_2, such that p_1 is already solved.
- **Problem**: find a solution for p_2 that minimizes the number of copies inserted between p_1 and p_1. Puzzle p_1 is called the *guider*, and puzzle p_2 is called the *follower*.

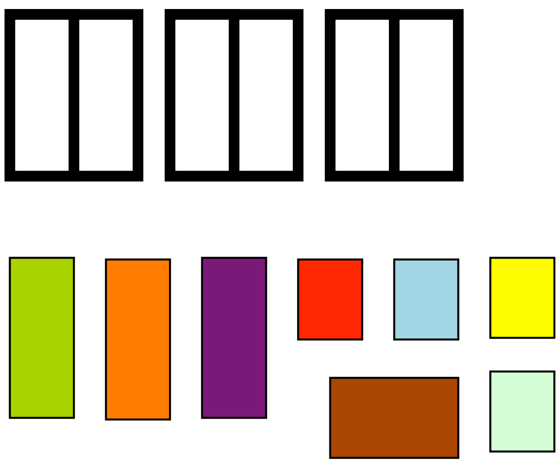


Example

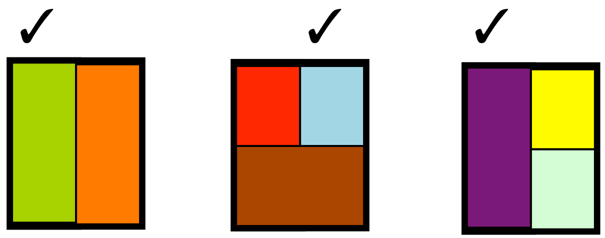
Guider



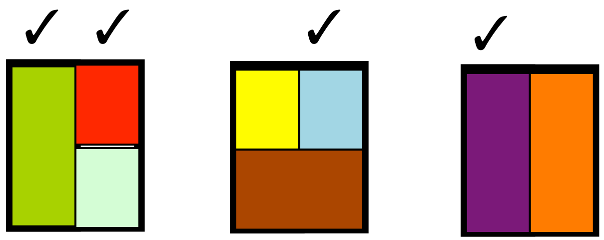
Follower



This solution has only three matches.



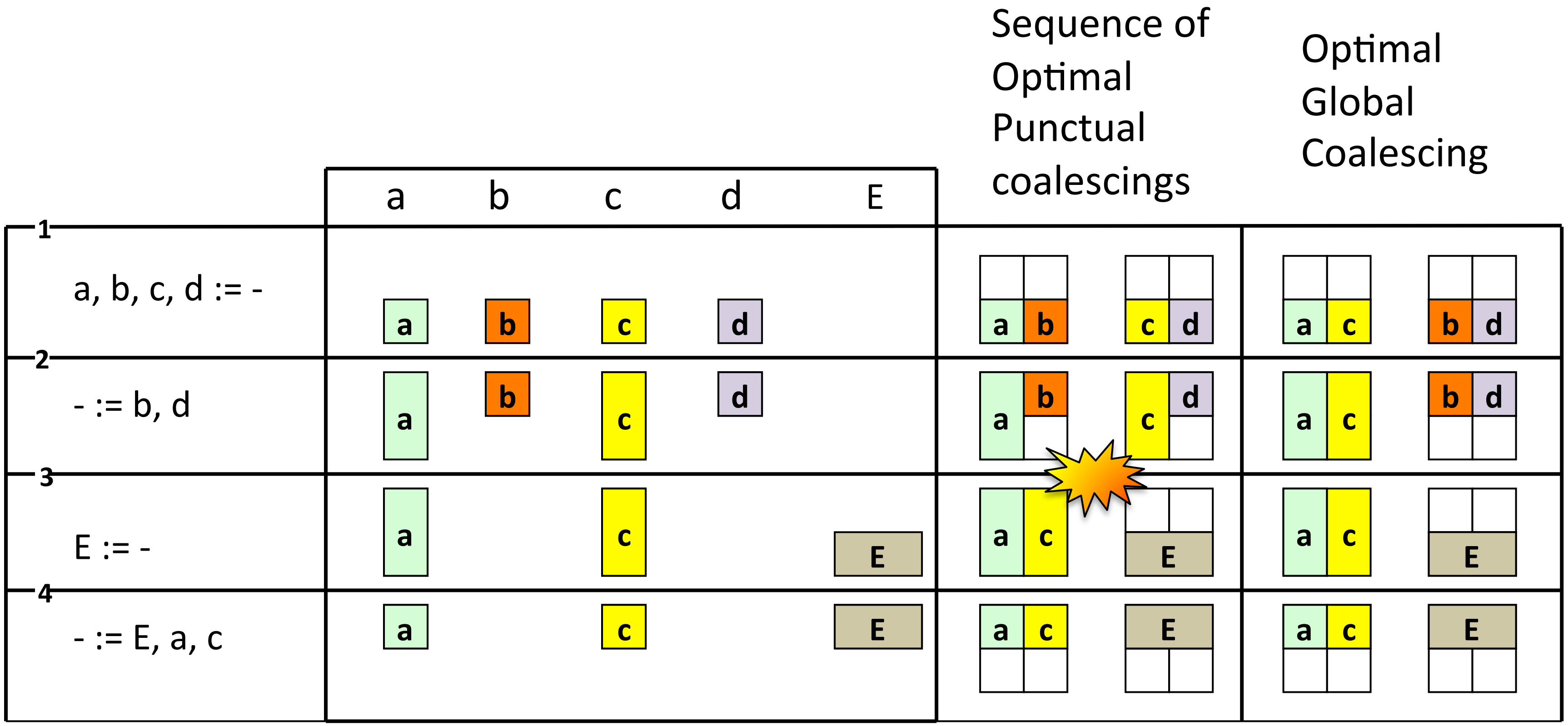
This solution is better: it has four matches.



Polynomial time Punctual Coalescing

- Punctual coalescing has a polynomial time solution, as long as the follower starts with an empty register board.
  - These are 89% of all the puzzles found in SPEC CPU 2000, when compiled to x86 using LLVM.
- Our solution is *O(K)*, where *K* is the number of registers. For the whole program, *O(I*K)*.
- For the solution, see our paper: *Punctual Coalescing*.

Punctual ≠ Global



Global Coalescing: find the register assignment that Minimizes the number of copies in the whole program. This problem is **NP-complete**.

Experiments

- We have compared four different coalescers:
  - A coalescing oblivious register allocator based on the coloring of *chordal graphs*.
  - The punctual coalescing heuristics used in register allocation *by puzzle solving*.
  - The optimal punctual coalescer.
  - An optimal solution for global coalescing, based on integer linear programming.
- Data: program traces from SPEC CPU 2000.

Number of copies inserted

	Gzip	Vpr	Gcc	Mcf	Crafty	Parser	Gap	Vortex	Bzip2	Twolf
Chordal (x 1000)	13,4	47,6	329,8	4,75	46,2	27,0	174,6	199,3	10,5	101,8
Heuristics	13	32	241	1	21	19	135	79	12	44
Punctual	0	10	17	0	1	5	33	1	0	0
ILP	0	2	4	0	0	0	3	0	0	0

Conclusion

- Punctual coalescing is a viable alternative for JIT compilers.
  - Has been implemented in LLVM with very competitive compilation times.
- Results are very good for program traces.
- We need a better algorithm for whole function optimization!
  - Slower compilation time, but better code quality.