Punctual Coalescing

Fernando Magno Quintão Pereira
– UFMG –
Jens Palsberg
– UCLA –

In the beginning there was a puzzle solver…

A puzzle solver is a register allocator

And it would perform some register coalescing

Which we never got the chance to explain

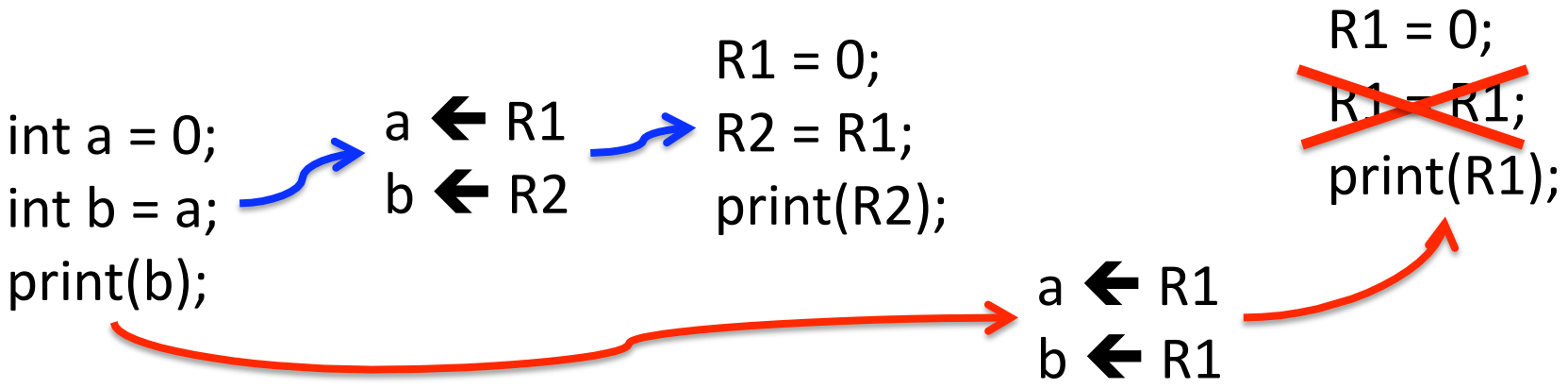
In CC'10 we fix this omission.

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(KI)*, where *K* = number of registers, and *I* is number of instructions.
- **Strength:** very fast!
- **Weakness:** local method.

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.



The Importance of Register Coalescing

Size reduction

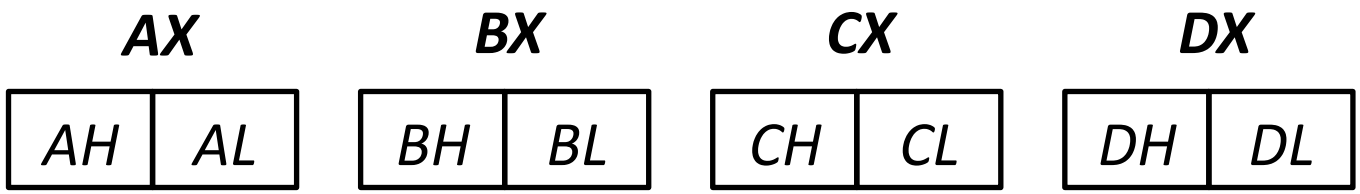
Execution speedup

Energy saving

Register Aliasing

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

Ex. X86's GP regs:



Ex. PowerPC's doubles:

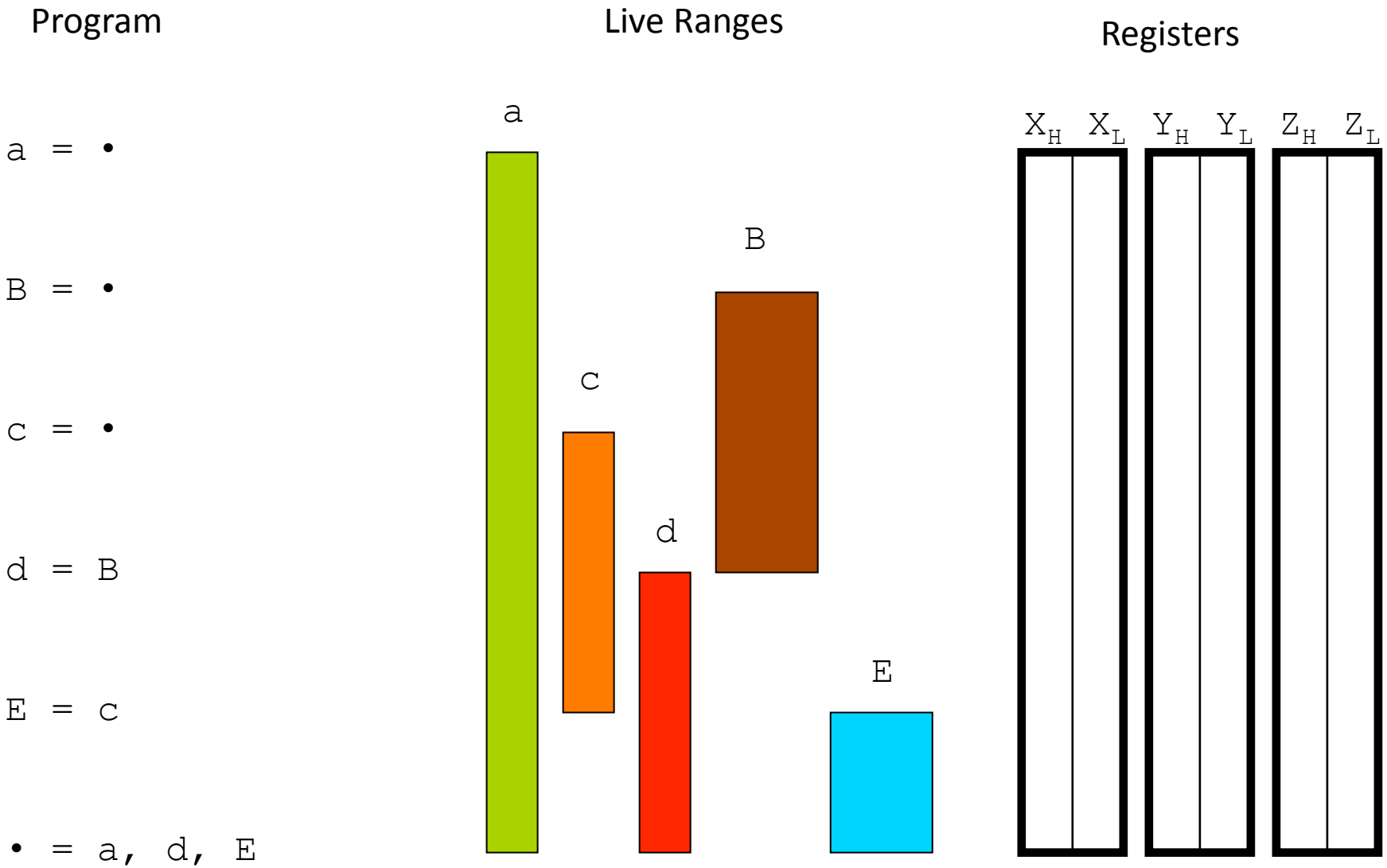
And also ARM, UltraSparc, ST240, etc

Register Aliasing

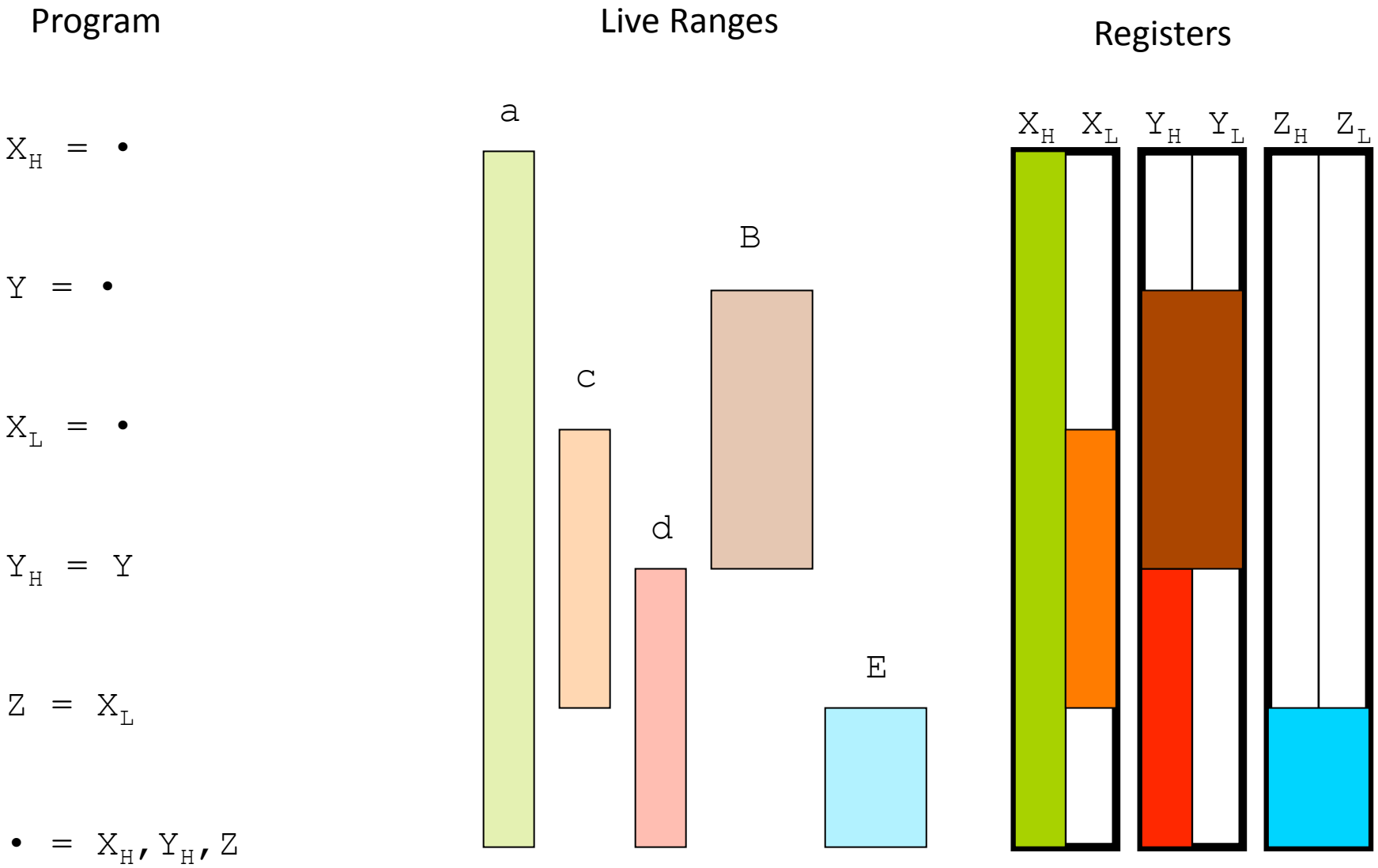
- Register aliasing allows the register allocator to keep more variables in registers.
  - E.g.: two floats in the same register.
- But finding a register assignment that minimizes the number of registers taken is NP-complete.
  - Lee *et al*: Aliased register allocation for straight-line programs is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

Register allocation: fit the bars on the boxes.

Aliased register allocation: Bars may have different widths.



Complexity result: optimal aliased register allocation is NP-complete for basic blocks in *SSA-form*.



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

Elementary Program

$a_1 = \bullet$

$(a_2) = (a_1)$

$B_2 = \bullet$

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

$c_3 = \bullet$

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

$d_4 = B_4$

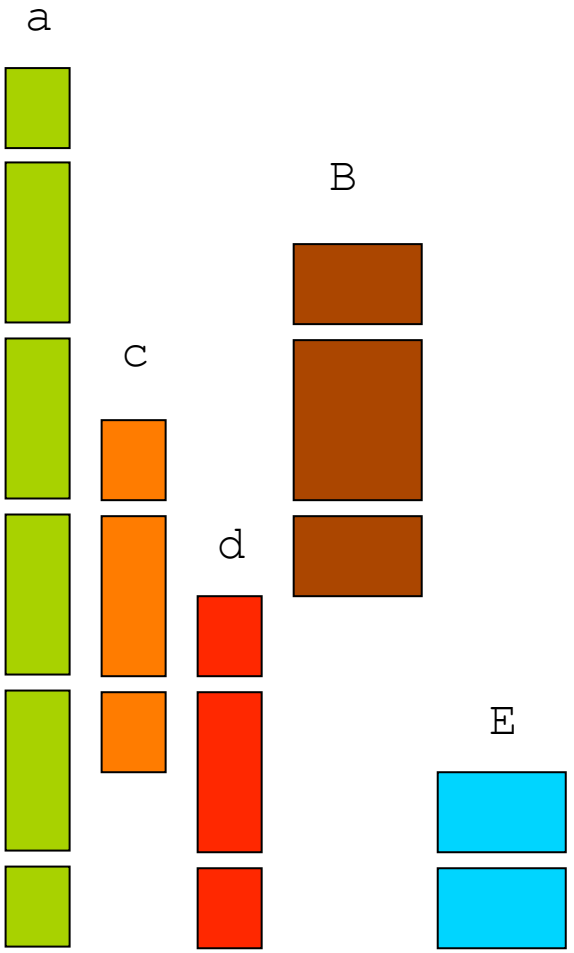
$(a_5, c_5, d_5) = (a_4, c_4, d_4)$

$E_5 = c_5$

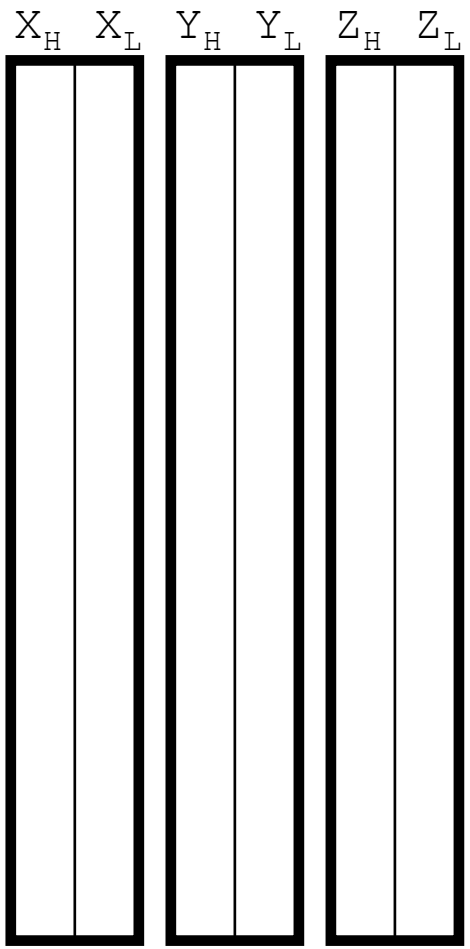
$(a_6, c_6, E_6) = (a_5, c_5, E_5)$

$\bullet = a_6, d_6, E_6$

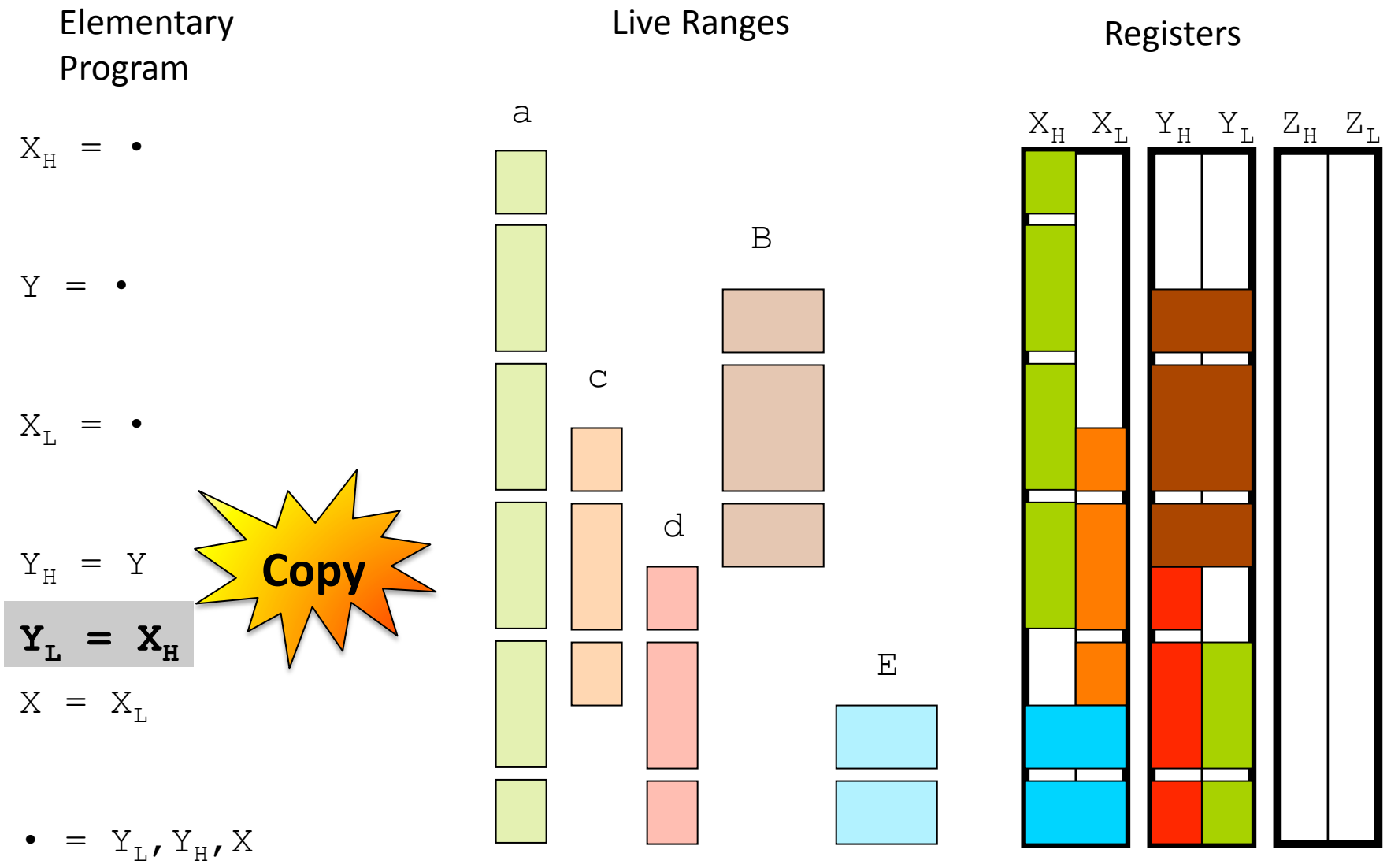
Live Ranges



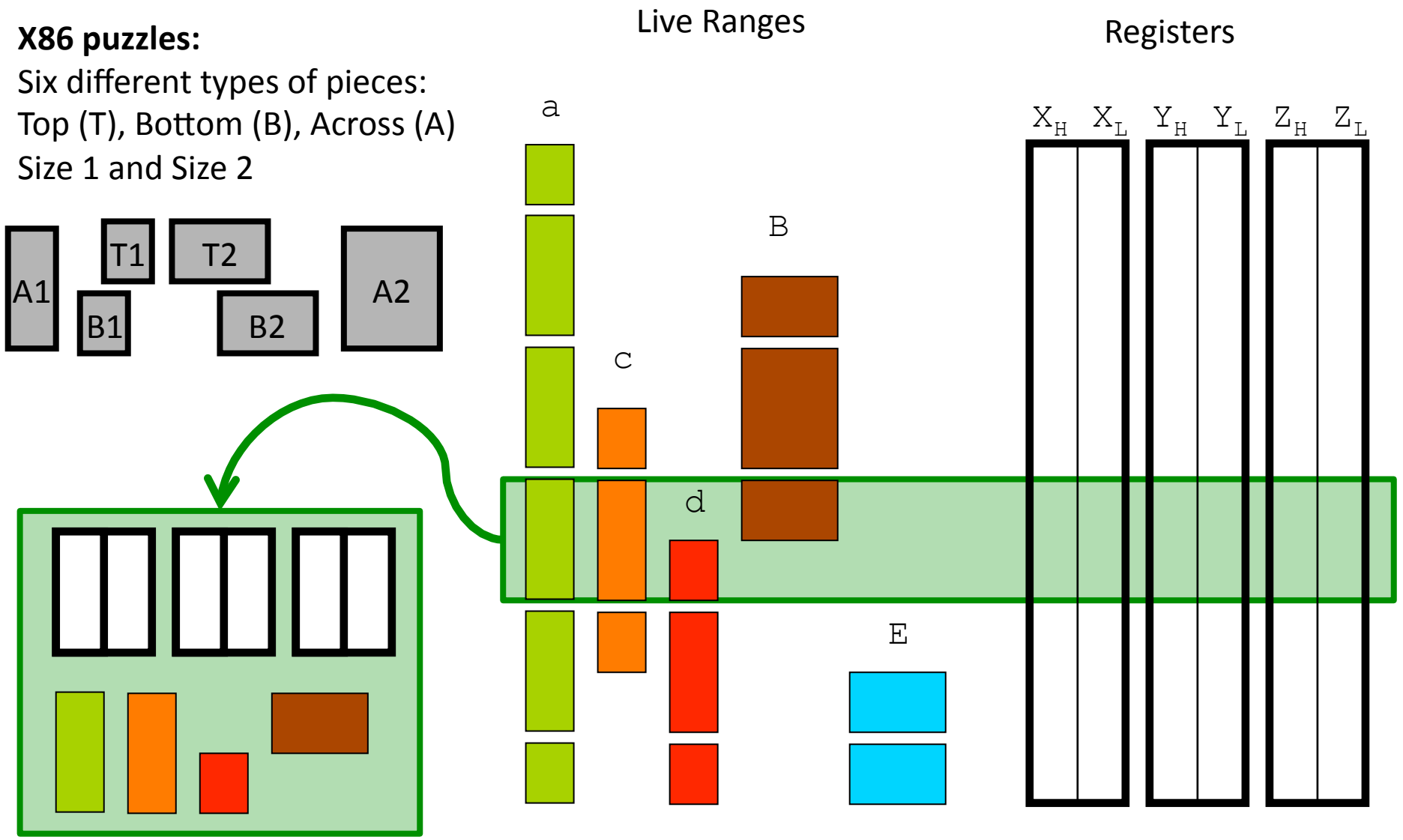
Registers



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



Register allocation puzzle: board formed by the register bank and pieces formed by the variables to be allocated.



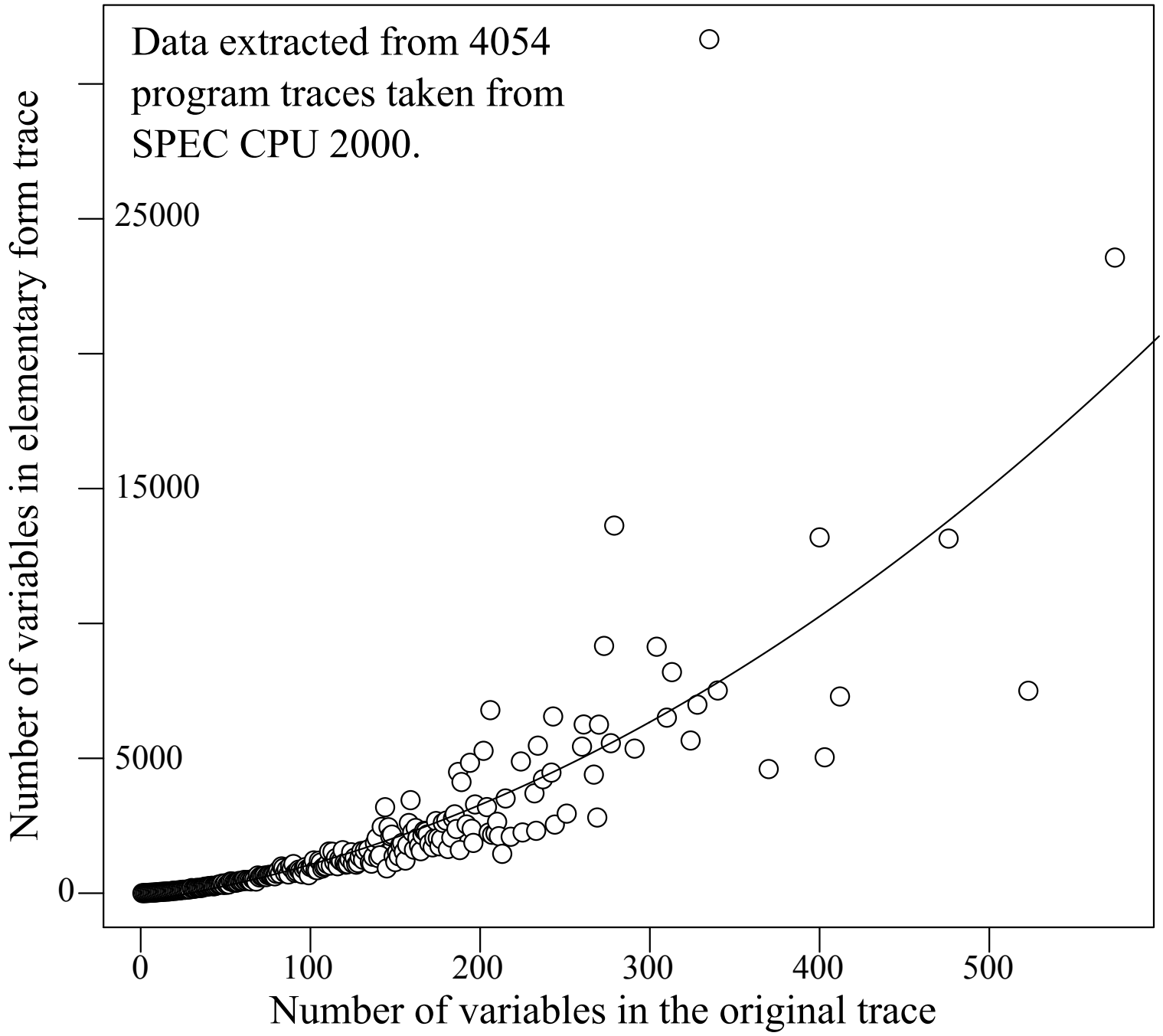
Dualism

- Live range splitting
  - More variables can fit in registers, so **less** spilling.
  - Adds **more** copies to assembly code.
  - Increases number of variables, so **slower** register allocation.
- Register coalescing
  - Might cause conflicts between variables, so **more** spilling.
  - **Removes** copies from assembly code.
  - Might **speed up** register allocation, for there will be less variables to handle.

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.

$a_1 = \bullet$

$(a_2) = (a_1)$

$B_2 = \bullet$

$(a_3, \_) = (a_2, B_2)$

$\__3 = \bullet$

Allocated!!!

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

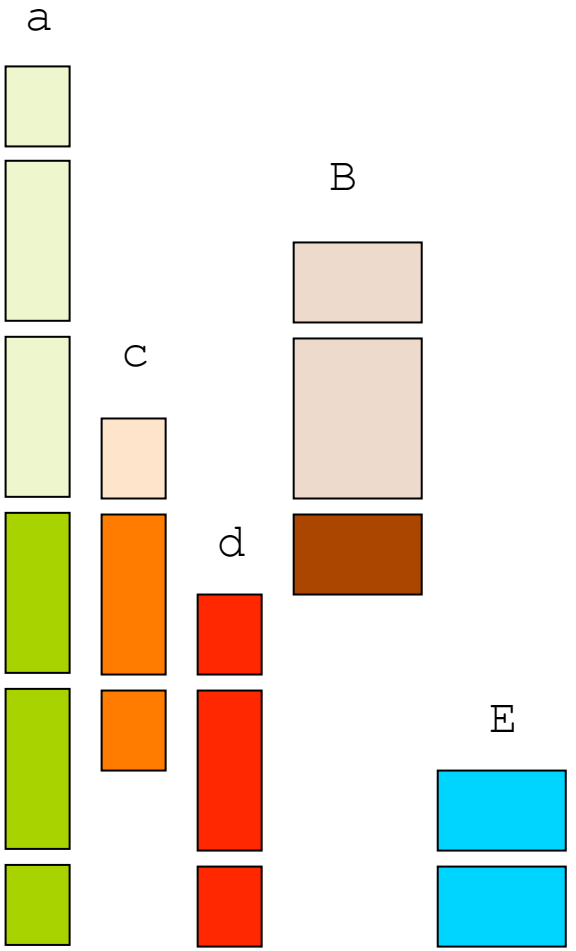
$d_4 = B_4$

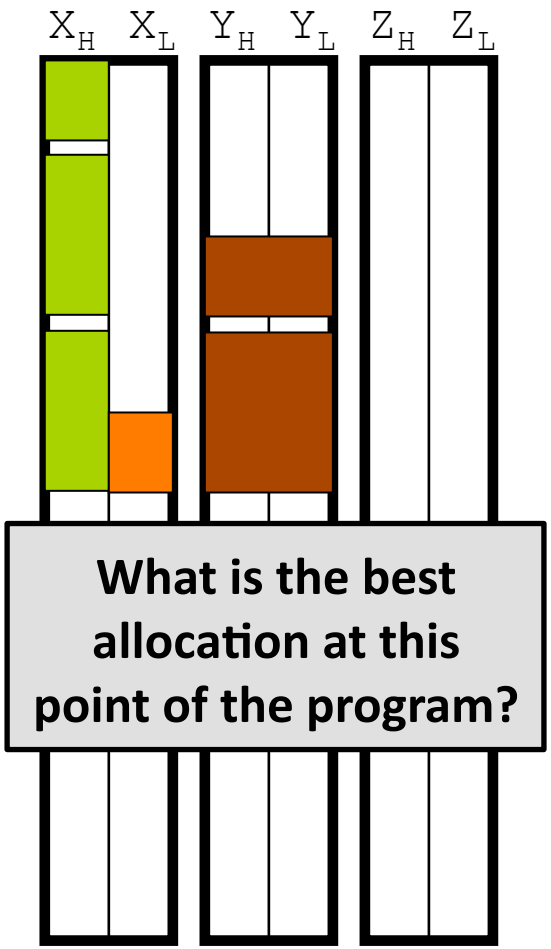
$(a_5, c_5, d_5) = (a_4, c_4, d_4)$

$E_5 = c_5$

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

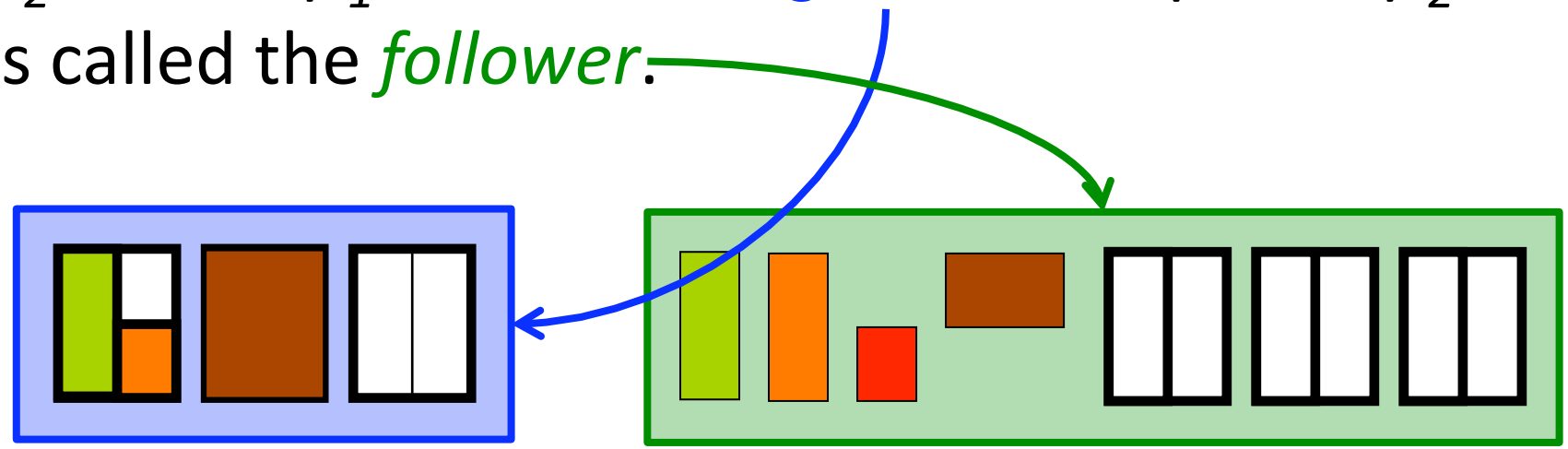
$\bullet = a_6, d_6, E_6$



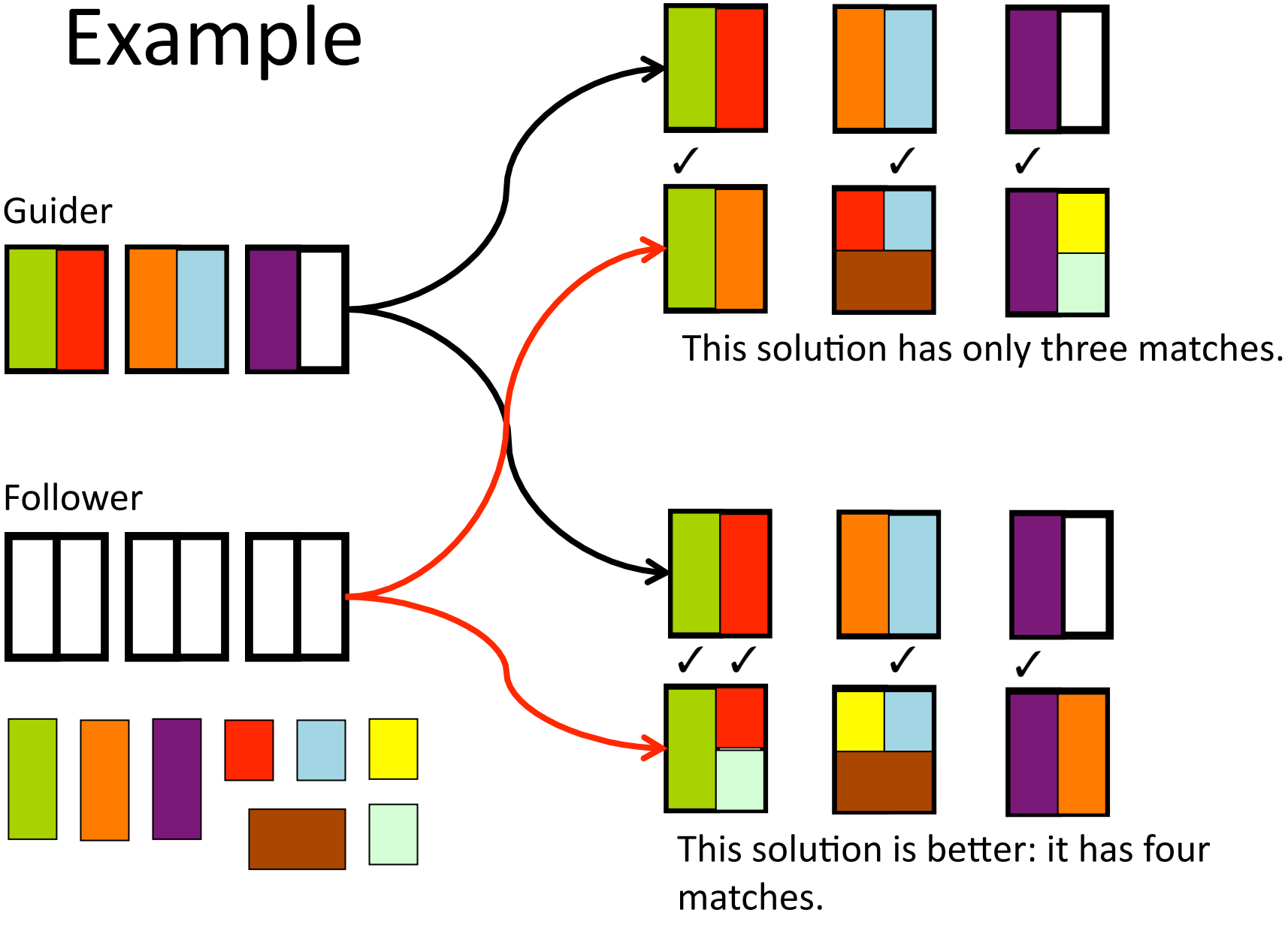


Punctual Coalescing

- **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_2. Puzzle p_1 is called the *guider*, and puzzle p_2 is called the *follower*.



Example

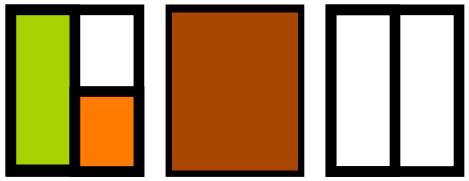

Guider
Follower
This solution has only three matches.
This solution is better: it has four matches.

Polynomial time Punctual Coalescing

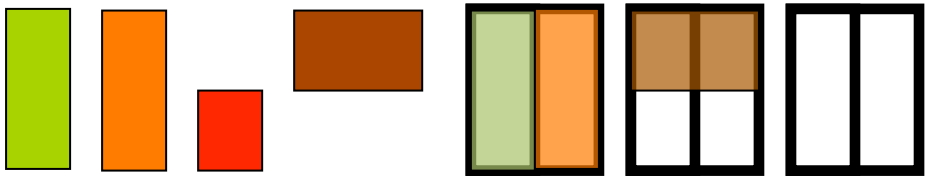
- Punctual coalescing has a polynomial time solution, as long as the follower starts with an empty register board.
  - These are 89% of all the puzzles found in SPEC CPU 2000, when compiled to x86 using LLVM.

Preferences of board areas

- Each board area has a preference.
- The preference of area *a* in the follower is defined by the placement of pieces onto *a* in the guider.
- Example:

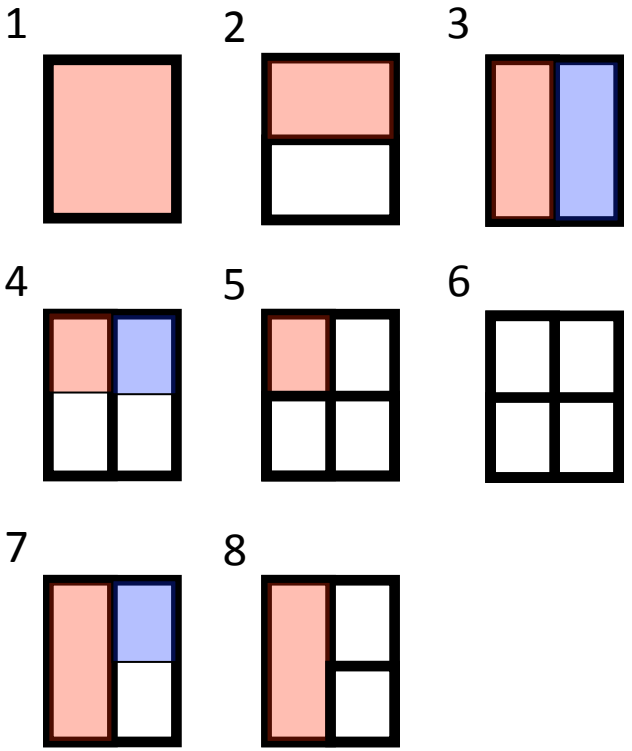


Guider

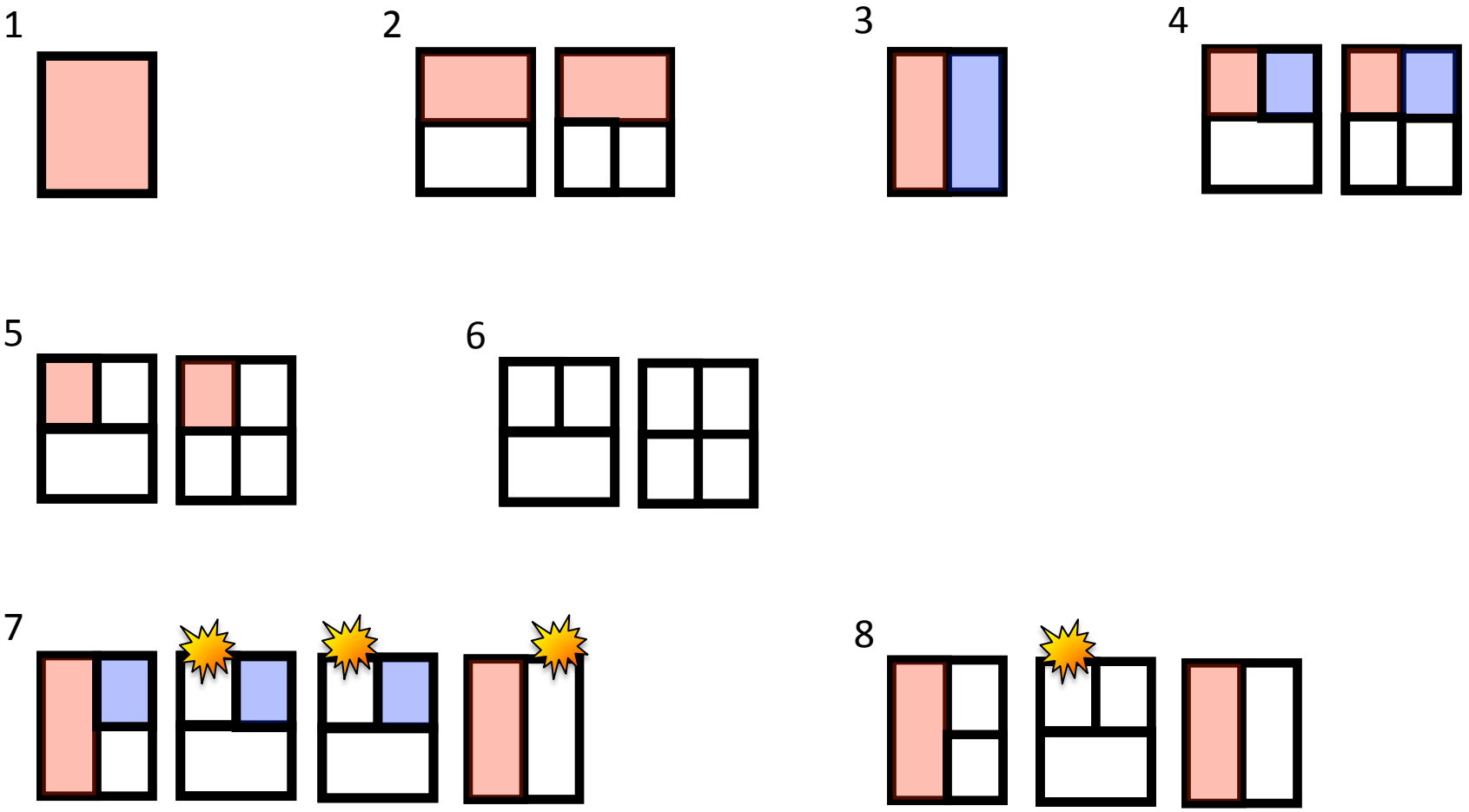


Follower

- There are eight possible configurations of preferences.



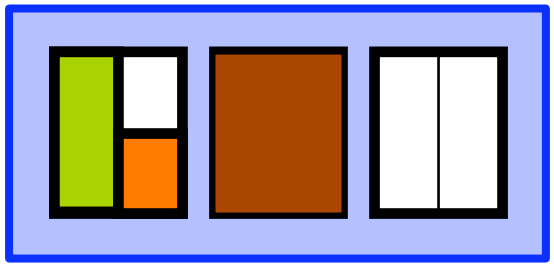
An algorithm for Punctual Coalescing

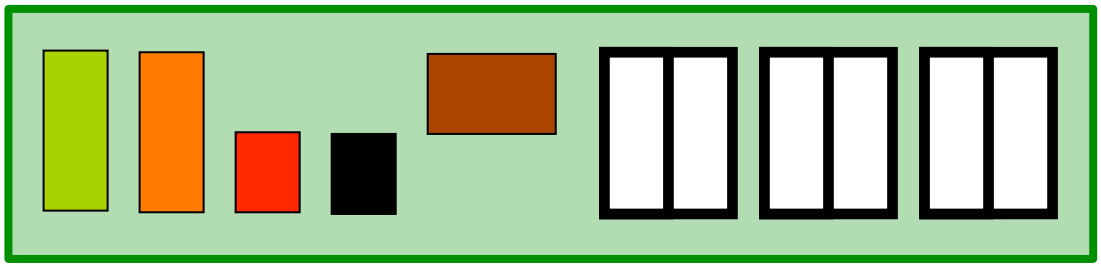


The Punctual Coalescer

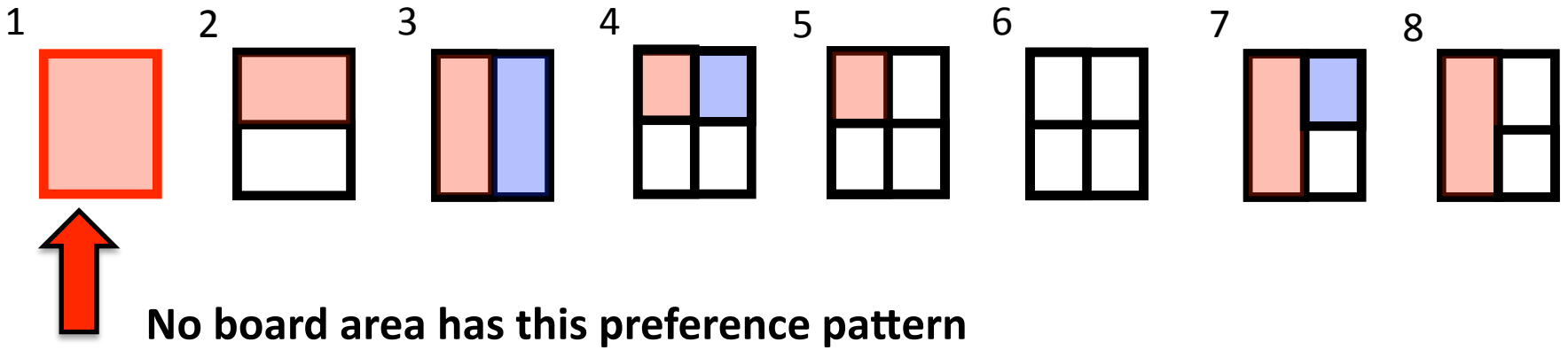
- For each statement s_i, from $1 < i < 8$:
  - For each empty area a such that the pattern of s_i matches a:
    - Apply s_i to a
    - If the application of s_i to a fails, then terminate the entire execution and report failure.

Example



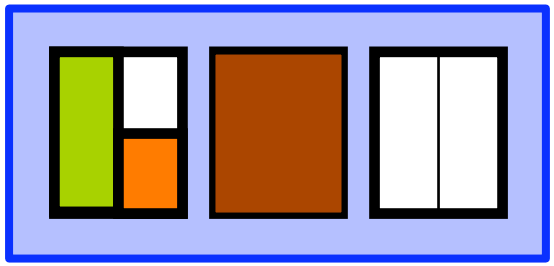


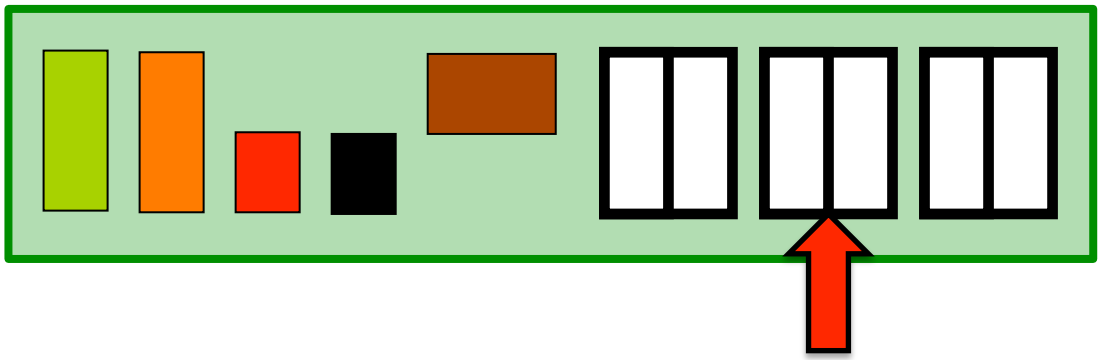
The preference patterns



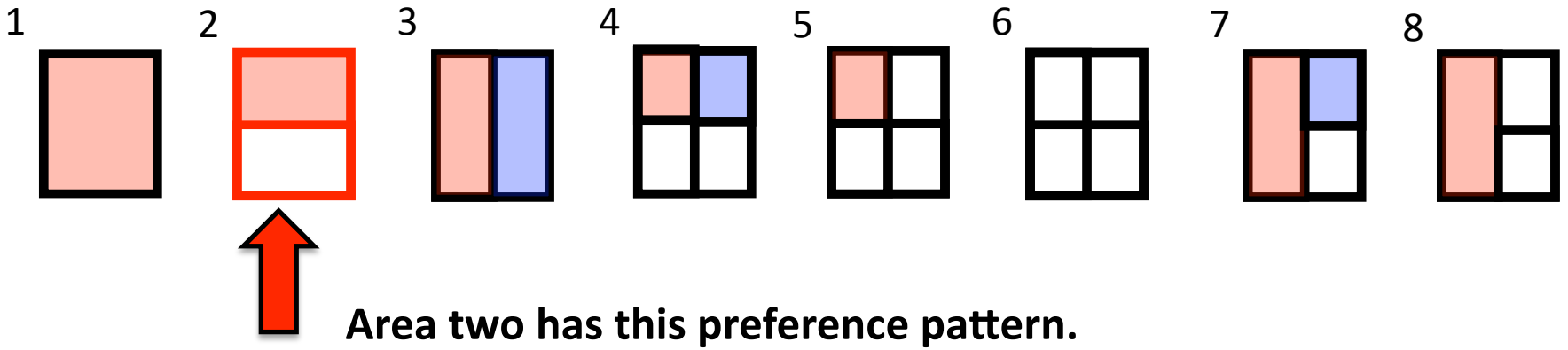
No board area has this preference pattern

Example



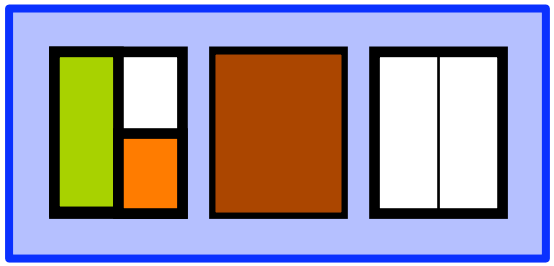


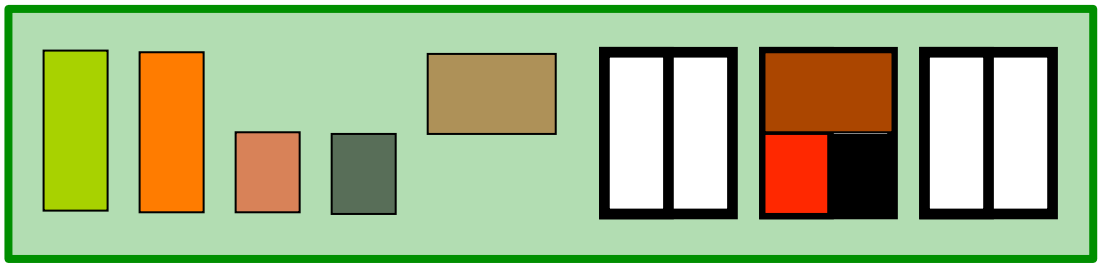
The preference patterns



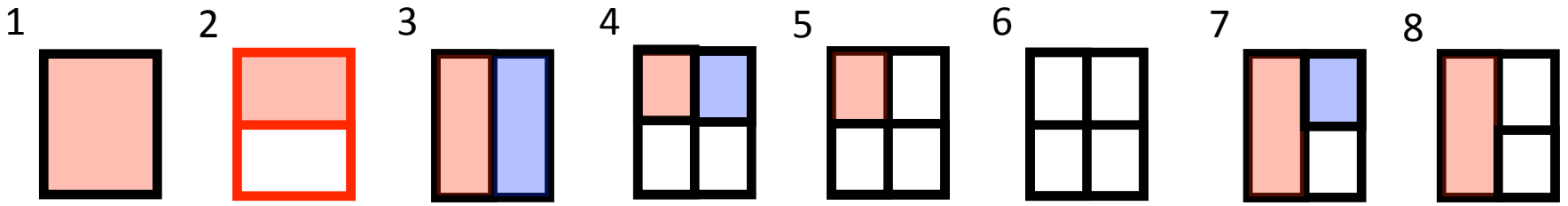
Area two has this preference pattern.

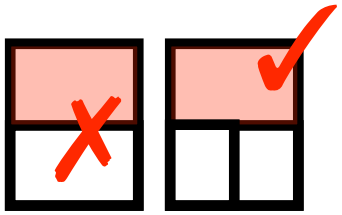
Example





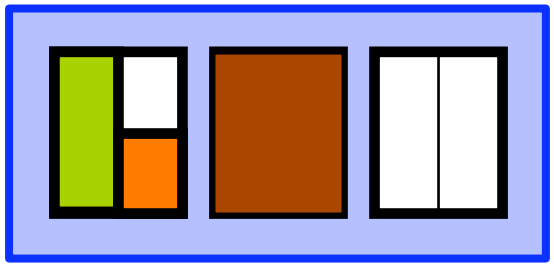
The preference patterns

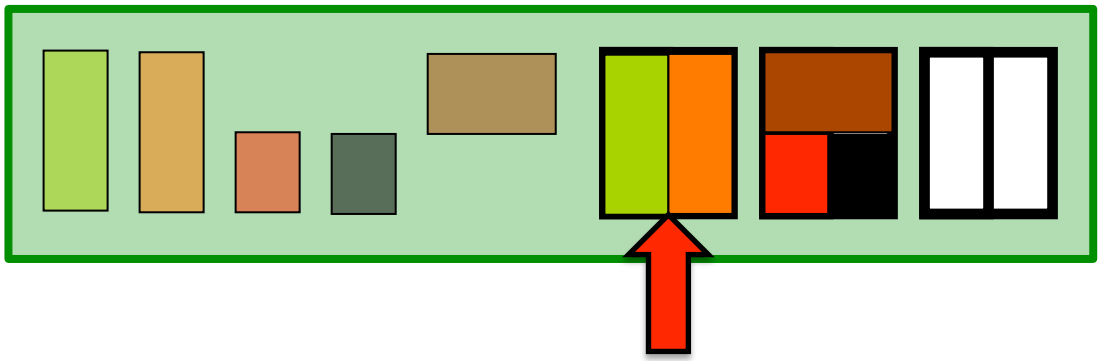




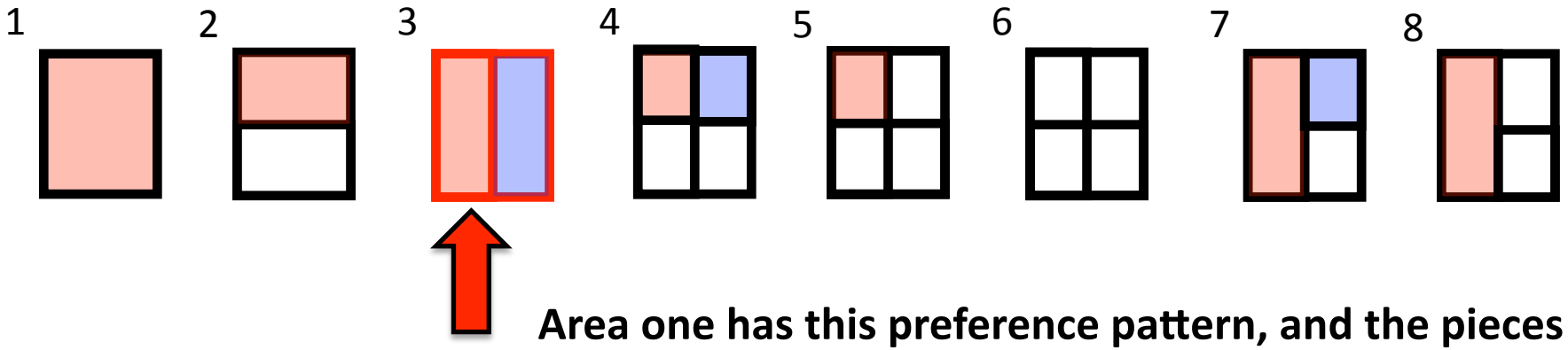
We do not have a size 2 bottom piece, but we have two size 1 bottom pieces.

Example





The preference patterns



Area one has this preference pattern, and the pieces are available.

Experiments

- We have compared four different coalescers:
  - A coalescing oblivious register allocator based on the coloring of *chordal graphs* (APLAS'05).
  - The punctual coalescing heuristics used in register allocation *by puzzle solving* (PLDI'08).
  - The optimal punctual coalescer (this paper).
  - An optimal solution for global coalescing, based on integer linear programming (this paper).
- Data: program traces from SPEC CPU 2000.

Running the Puzzle Solver

- Compared with LLVM 2.1 (early 2008). For SPEC CPU 2000:
  - 6-7% slower compilation time
  - The code produced runs 3-4% faster
  - Research artifact with room for improvements

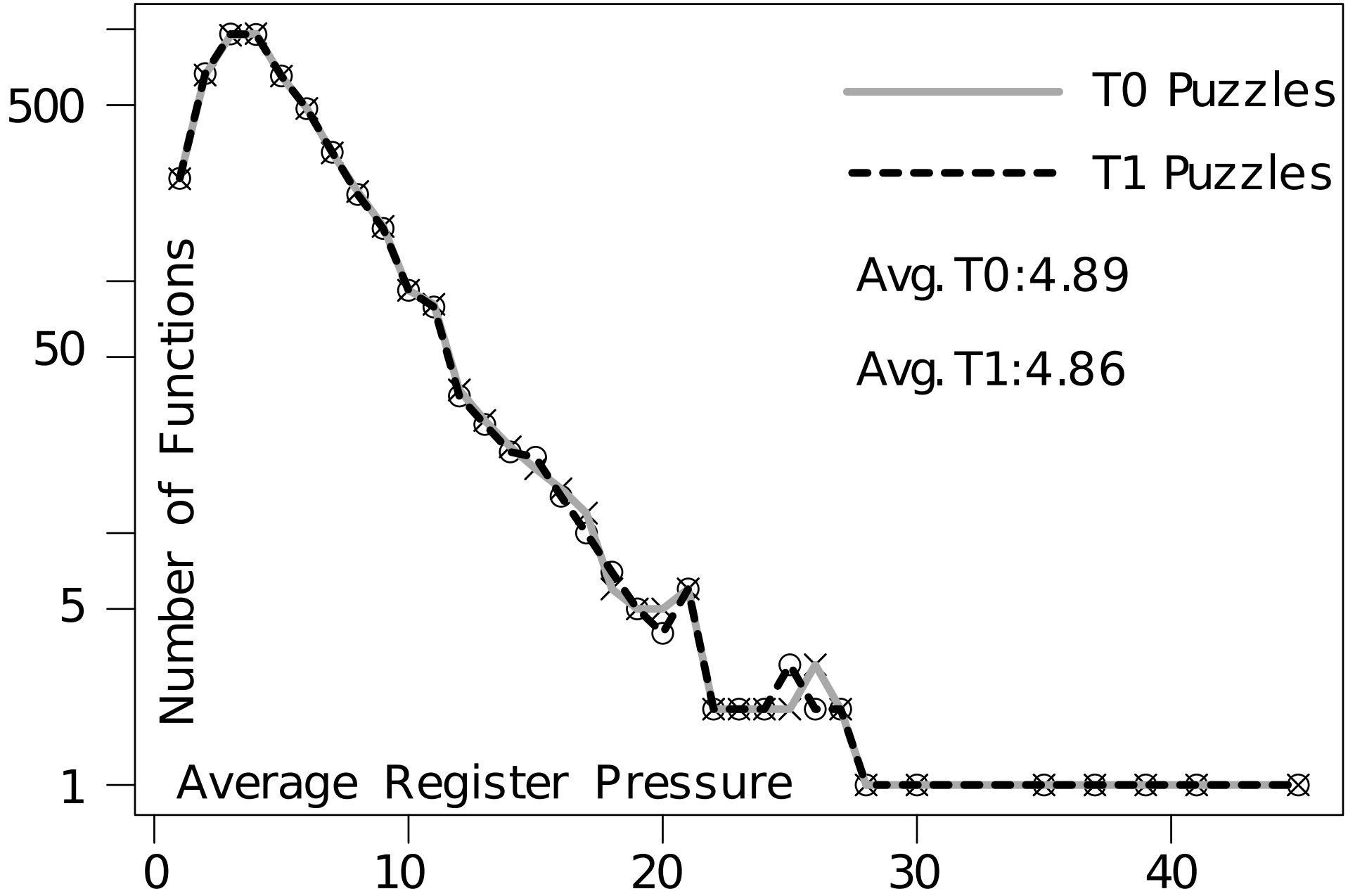
Number of copies inserted

	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

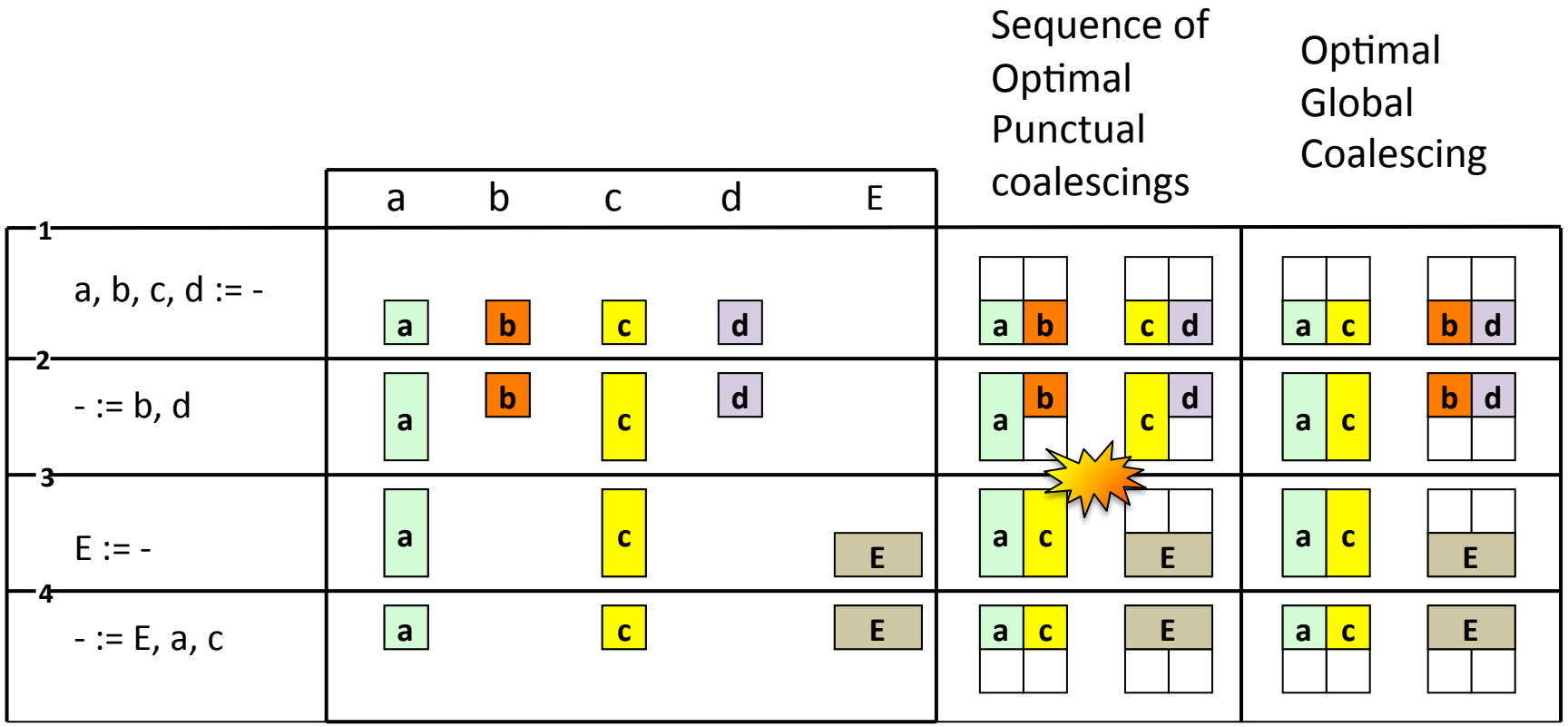
Aliasing is seldom used

- **T0 average register pressure**: if we must store each variable in a different register.

- **T1 average register pressure**: if we are allowed to store two small variables in the same register.

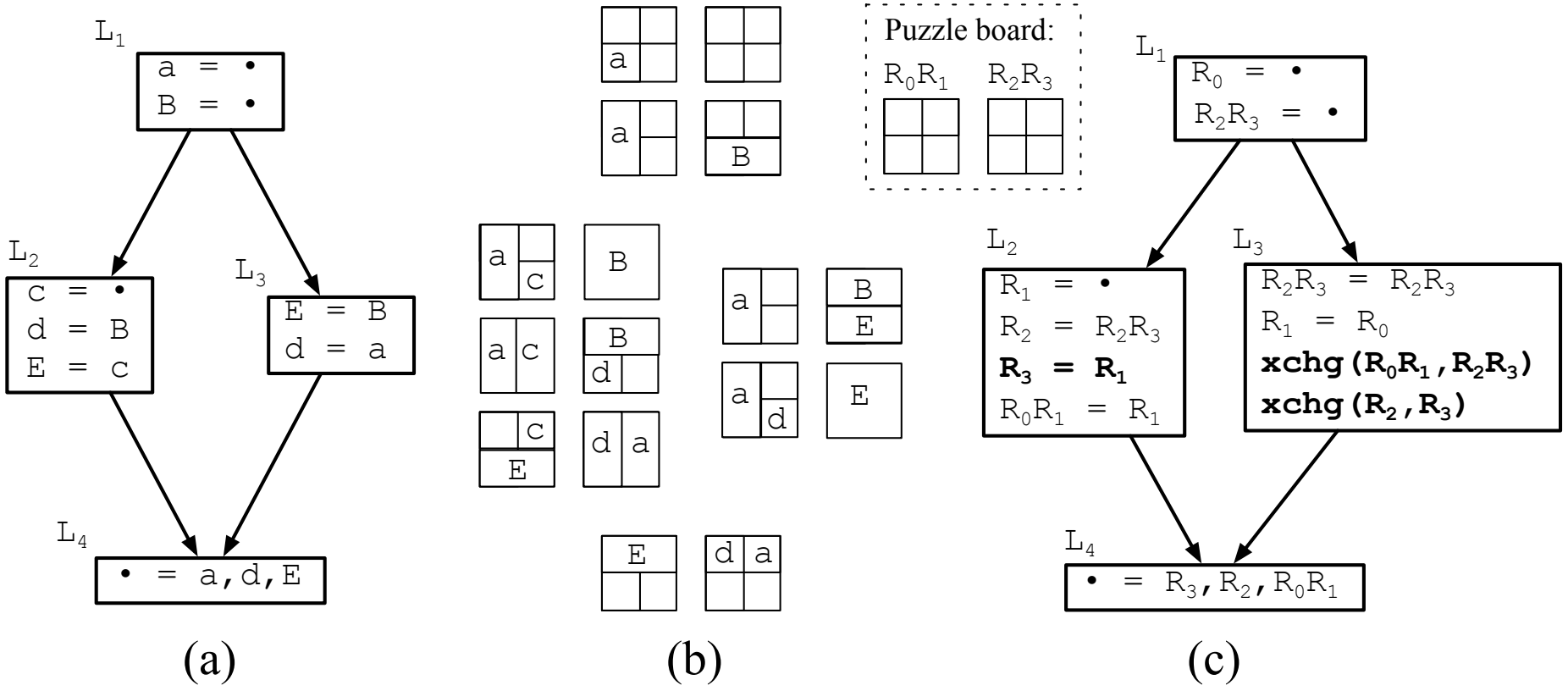

500
50
5
1
Number of Functions
0
10
20
30
40
Average Register Pressure
T0 Puzzles
T1 Puzzles
Avg. T0:4.89
Avg. T1:4.86

Limitations: Punctual ≠ Global



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

Limitations: Branches



Choosing a good order to scan the **dominator tree** is **essential** to get a good register assignment.

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.
  - Profiling guided compilation.
  - Static branch prediction.
  - Trace compilation: e.g, *traceMonkey*.