From Speculative Partial Redundancy Elimination to Speculative Partial Dead Code Elimination

Nigel Horspool, University of Victoria, Canada
David Pereira, University of Victoria, Canada

in collaboration with

Bernhard Scholz, University of Sydney, Australia
Jens Knoop, Vienna University of Technology, Austria

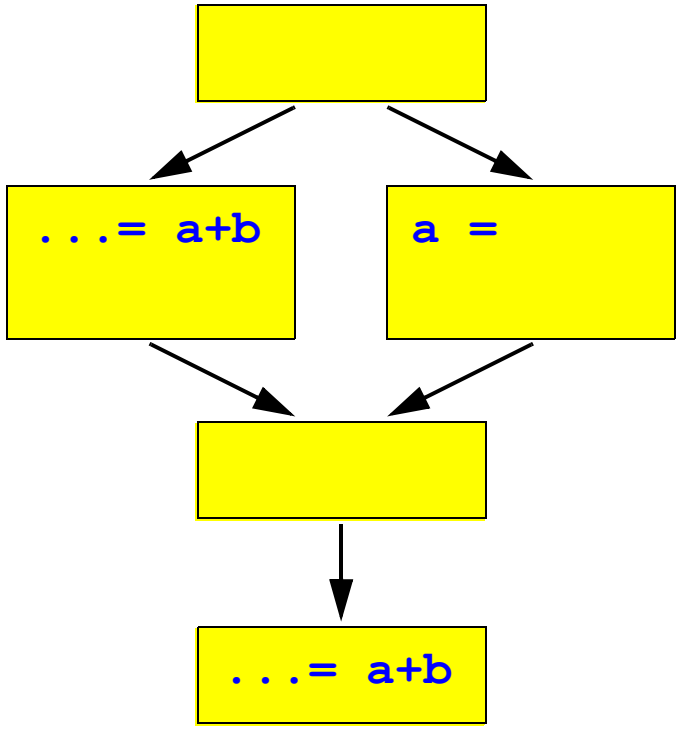
Synopsis

- This is work in progress.
- Some experimental results with SPRE were reported in proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES), June 2004.
- We are exploring ideas to generalize SPRE and to make it a practical approach for use in JIT compilers.

From PRE to SPRE

Partial Redundancy Elimination

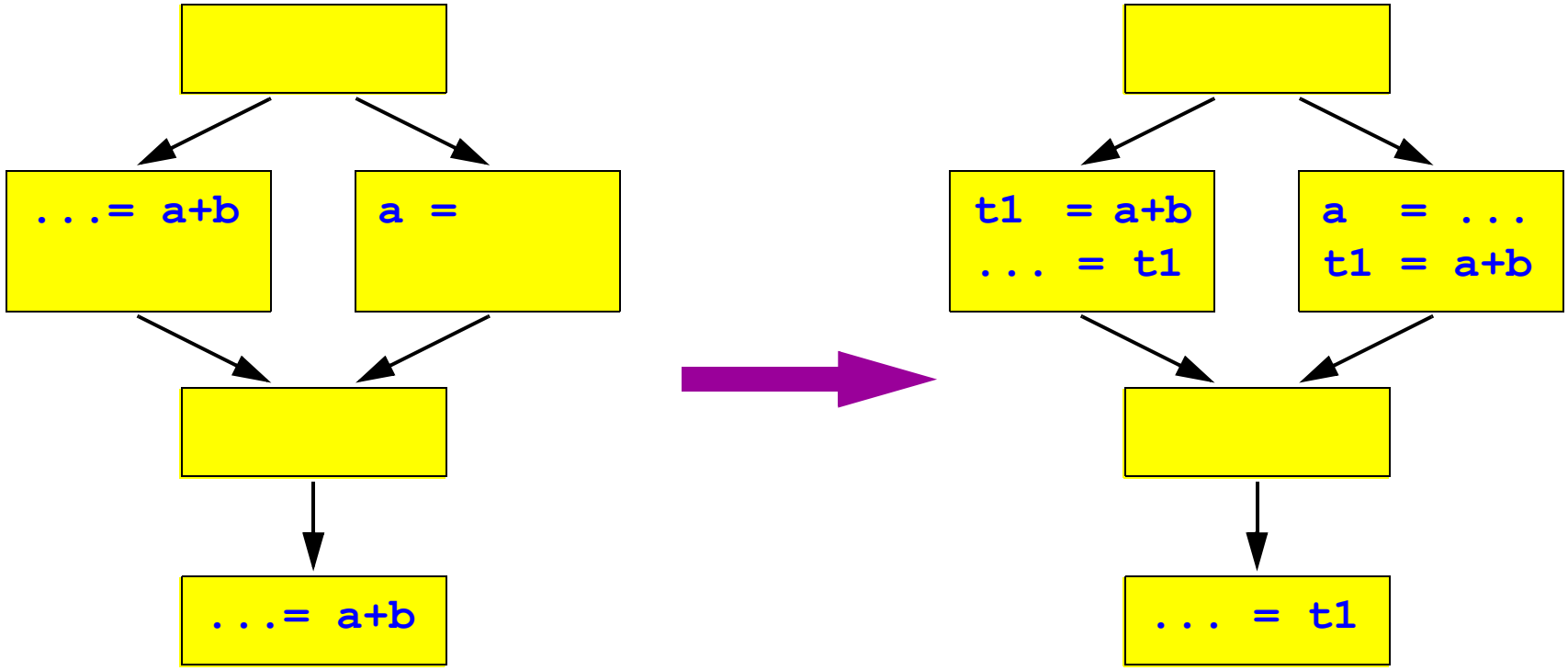
... is a generalization of code motion (Morel and Renvoise, 1979)



From PRE to SPRE

Partial Redundancy Elimination

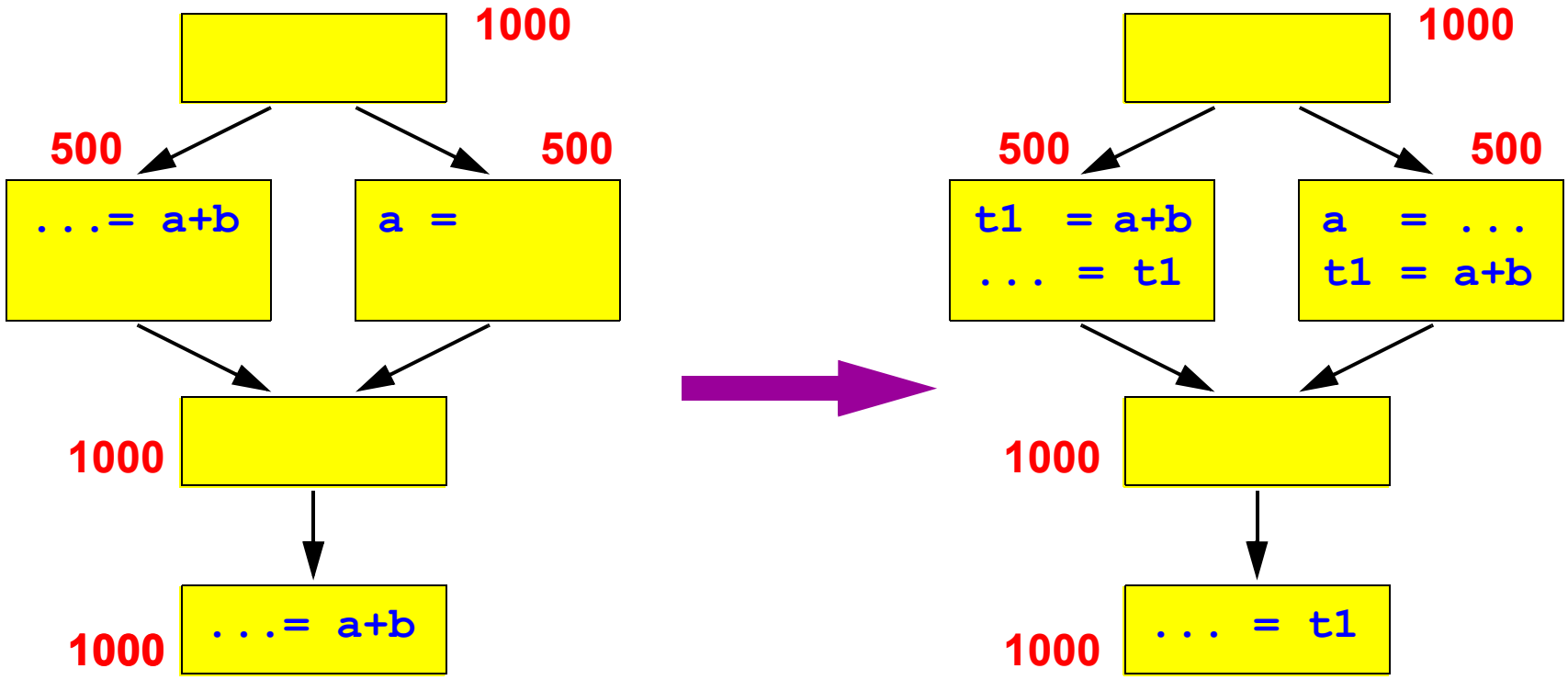
... is a generalization of code motion (Morel and Renvoise, 1979)



From PRE to SPRE

Partial Redundancy Elimination

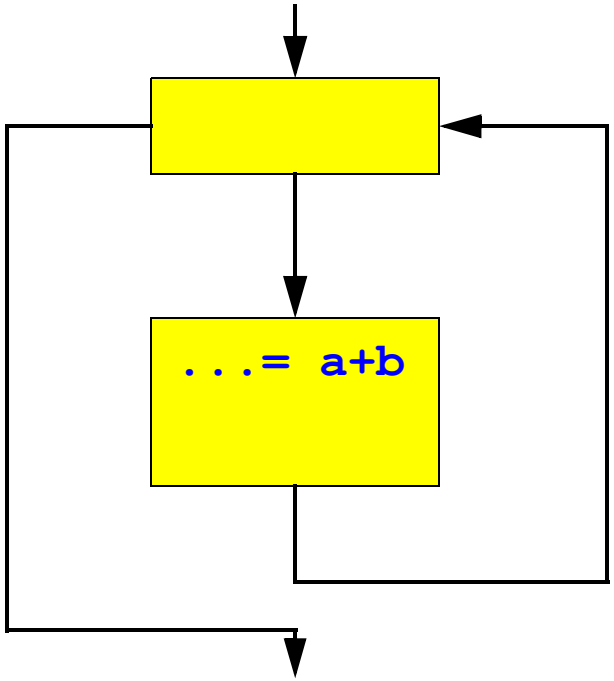
... is a generalization of code motion (Morel and Renvoise, 1979)



But... Partial Redundancy Elimination

... is also very conservative. An expression *e* can be inserted at a point P only if every path starting from P uses *e*. This restriction guarantees:

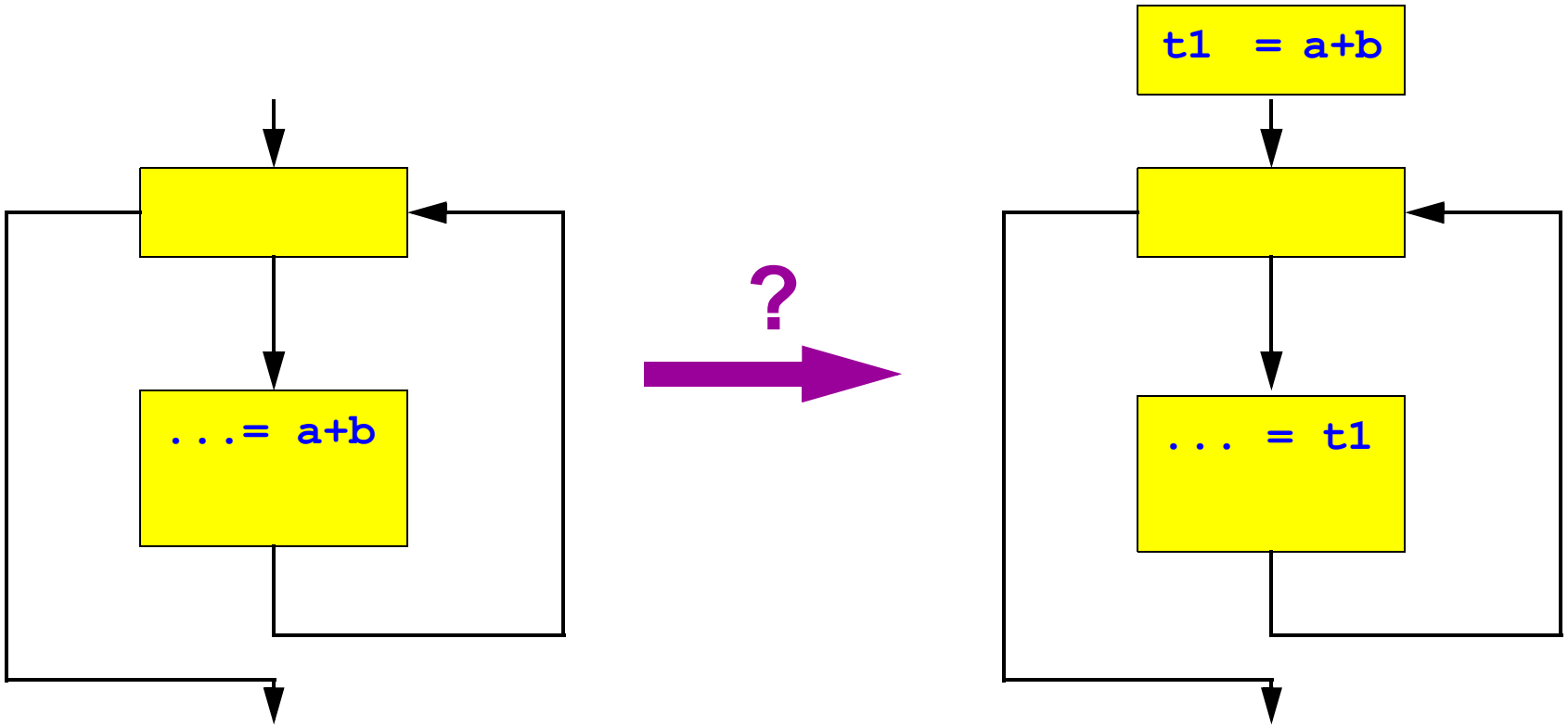
- safety, and
- optimality (no more evaluations of the expression than before).



But... Partial Redundancy Elimination

... is also very conservative. An expression *e* can be inserted at a point P only if every path starting from P uses *e*. This restriction guarantees:

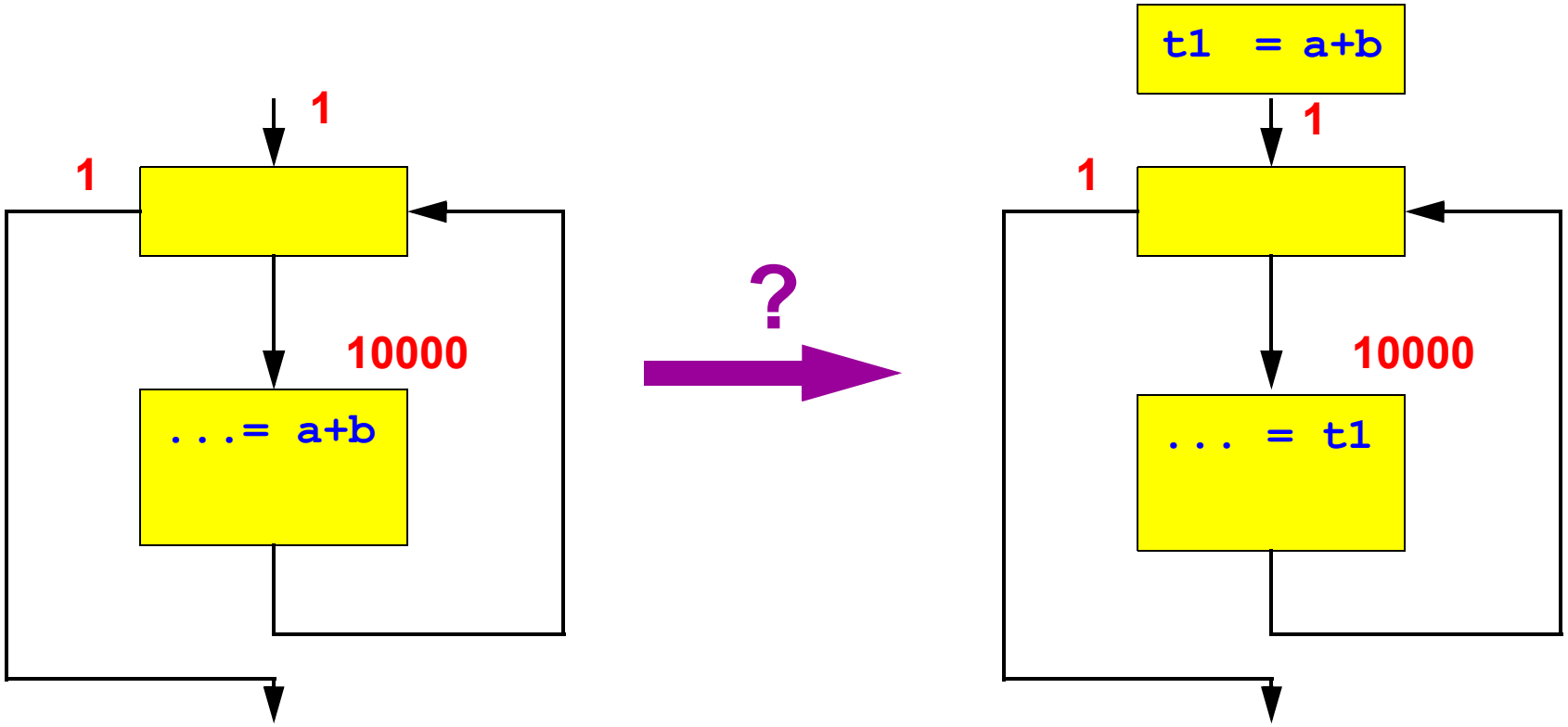
- safety, and
- optimality (no more evaluations of the expression than before).



But... Partial Redundancy Elimination

... is also very conservative. An expression *e* can be inserted at a point P only if every path starting from P uses *e*. This restriction guarantees:

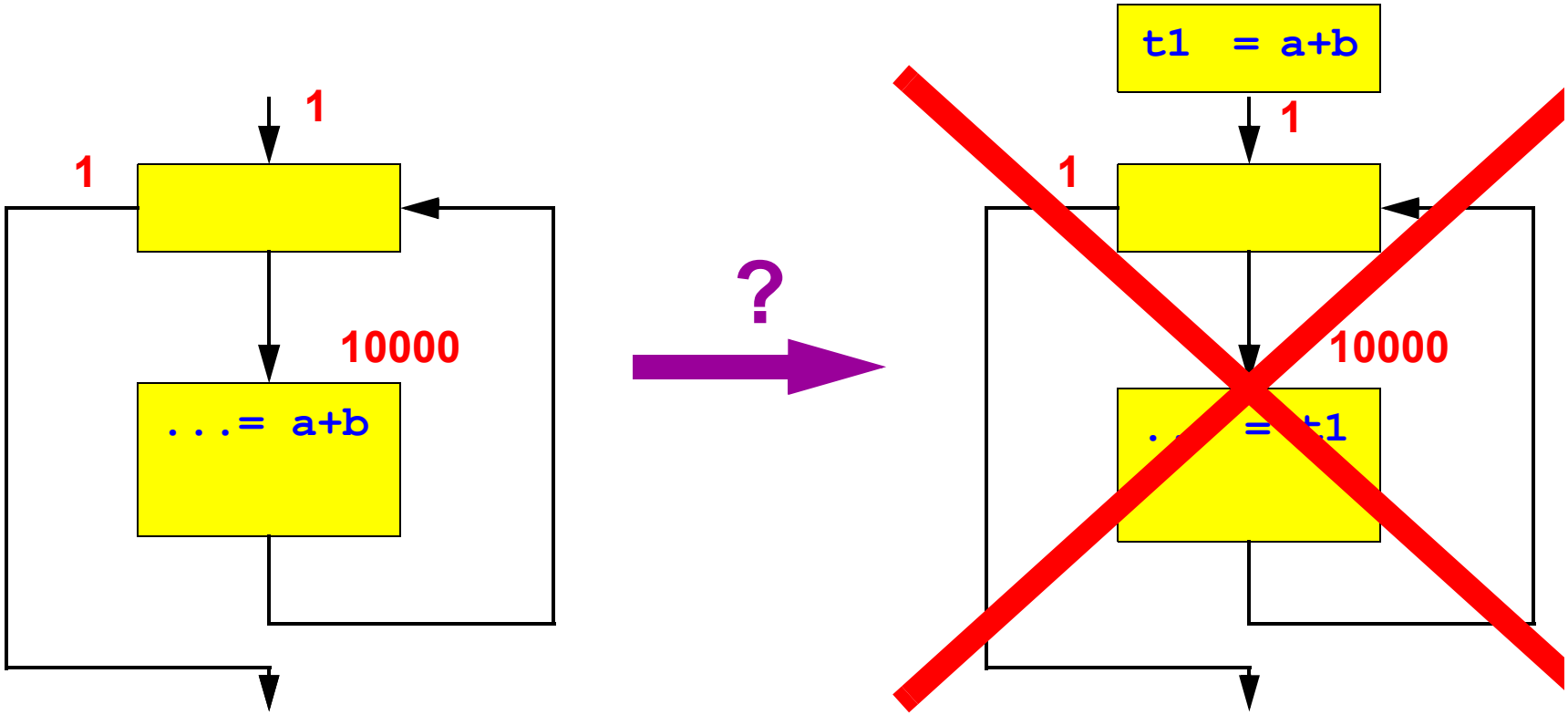
- safety, and
- optimality (no more evaluations of the expression than before).



But... Partial Redundancy Elimination

... is also very conservative. An expression *e* can be inserted at a point P only if every path starting from P uses *e*. This restriction guarantees:

- safety, and
- optimality (no more evaluations of the expression than before).



Speculative PRE

- An evaluation of *e* can be inserted anywhere as long as it is *safe* to do so,

Speculative PRE

- An evaluation of *e* can be inserted anywhere as long as it is *safe* to do so, and

- we speculatively compute *e* in the hope that the value will be useful later.

Speculative PRE

- An evaluation of *e* can be inserted anywhere as long as it is *safe* to do so, and

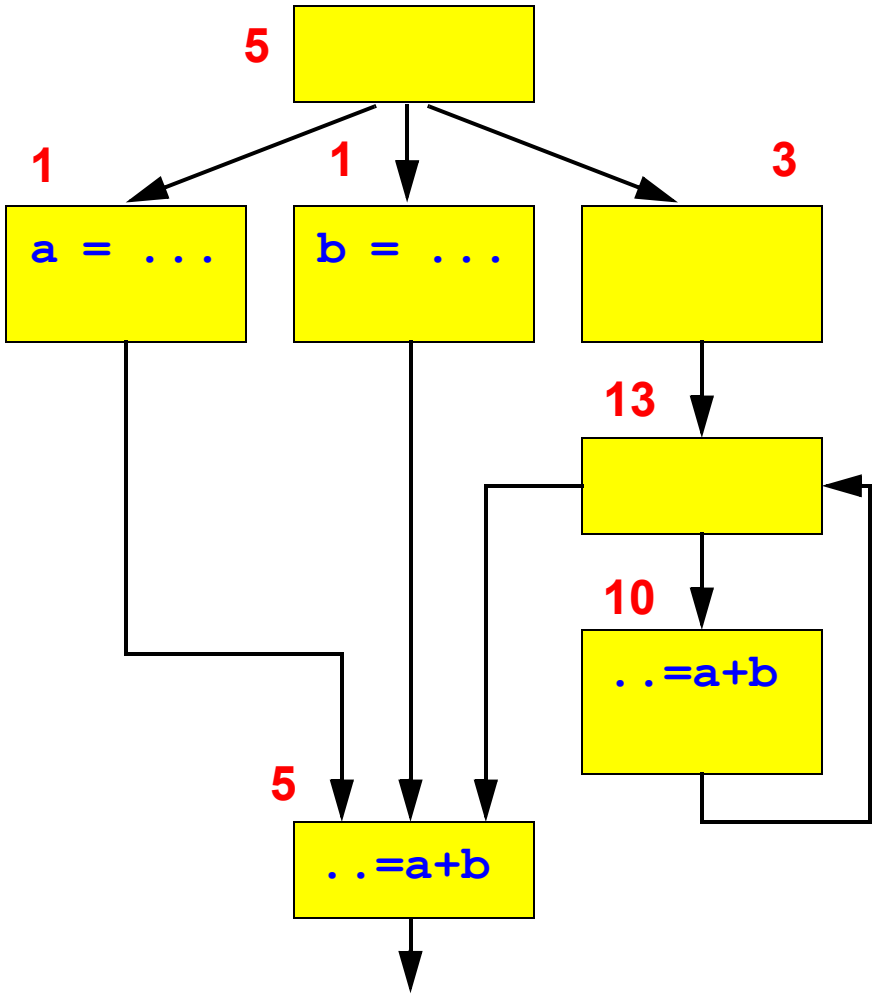
- we speculatively compute *e* in the hope that the value will be useful later.

- Using probabilistic information (from execution profiles or elsewhere), the optimality goal becomes minimization of the *expected* number of evaluations.

  Cai & Xue [CGO 2003] and Scholz, Horspool & Knoop [LCTES 2004] have presented SPRE algorithms which find optimal solutions.

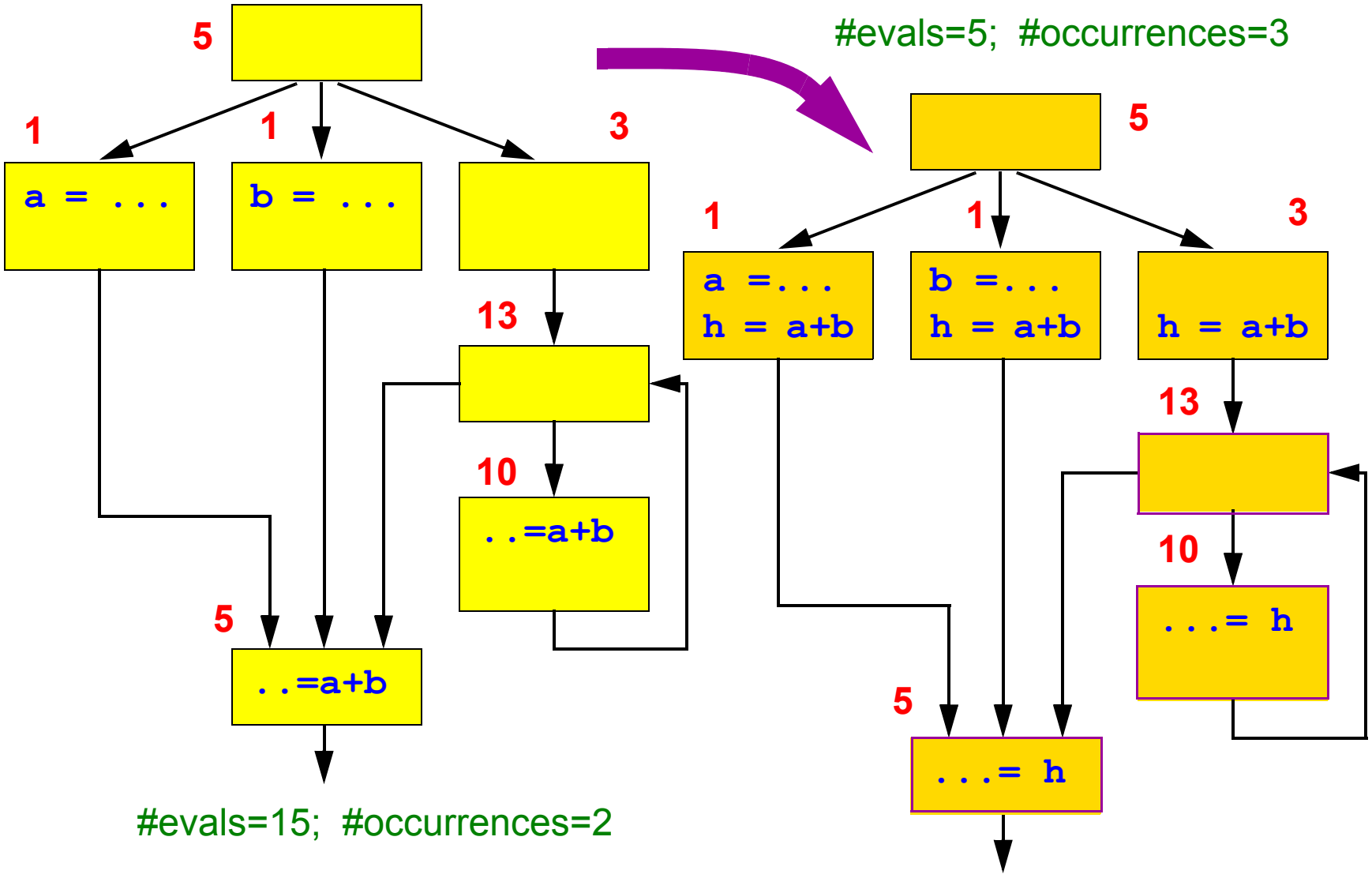
  We prefer the LCTES 2004 approach because the objective function can combine several different cost measures.

SPRE Example

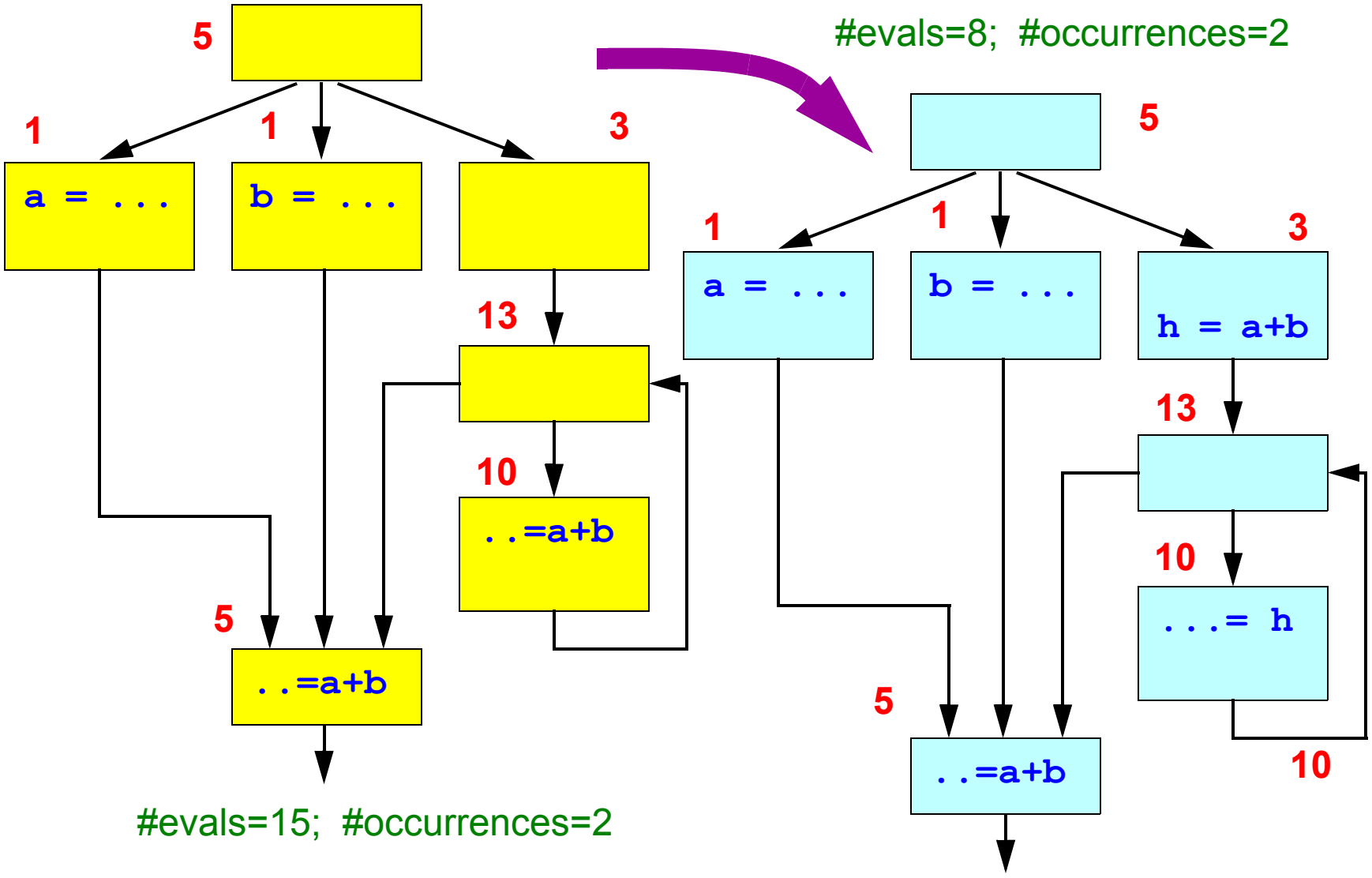


#evals=15; #occurrences=2

SPRE Example – optimized for time



SPRE Example – optimized for space+time



Optimal SPRE

- For each expression, our algorithm maps the optimal SPRE problem to a network flow problem.
- Efficient techniques exist to solve for maximum flow.
- The cost function is a linear combination of:
  - any static property (size is a useful measure),
  - execution frequency times any static property

Issues with SPRE and proposed solutions

1. Register pressure

Usually we have too many expressions and too few registers. How can we handle that?

Issues with SPRE and proposed solutions

1. Register pressure

Usually we have too many expressions and too few registers. How can we handle that?

Proposal:

Incorporate a measure of register pressure into the cost function.

E.g., lifetime of the value (distance from place where computed to the place where used) can be added into the objective function.

Cost of using a saved value =

$$a \times \textit{size of the expression} +$$
$$\text{frequency} \times (b \times \textit{execution time of original expression} +$$
$$c \times \textit{distance from top of block})$$

and there are similar costs elsewhere for keeping the value alive.

Issues with SPRE and proposed solutions

2. Computational cost

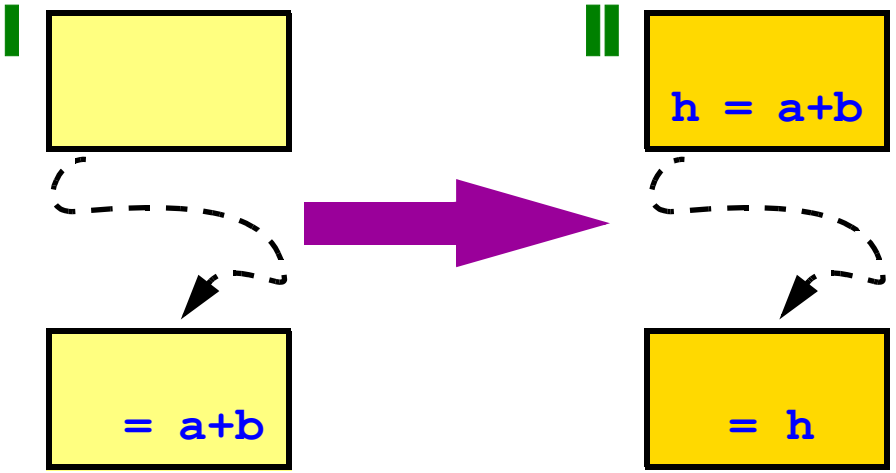
Only one expression at a time is analyzed. (It's not a bit-vector problem.)

Solutions to be tried

- Reduce the size of the constructed network by
  - applying some basic transformation techniques, and
  - discarding low frequency regions of the network.
- Incremental analysis (reusing a previous analysis).

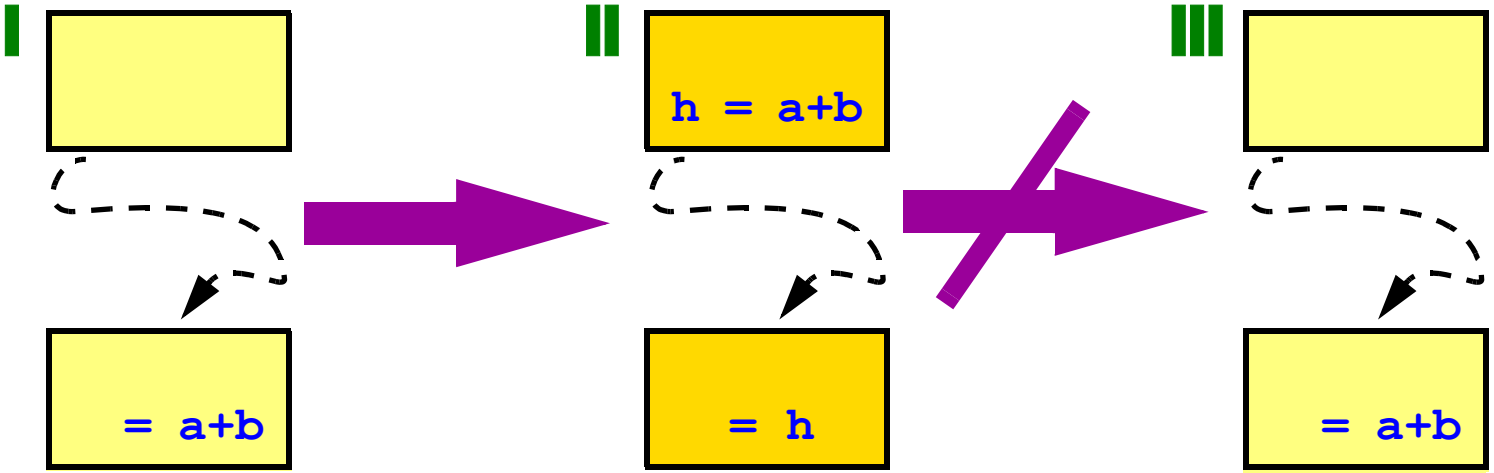
Issues with SPRE and proposed solutions

3. Non-reversibility of code transformation



Issues with SPRE and proposed solutions

3. Non-reversibility of code transformation

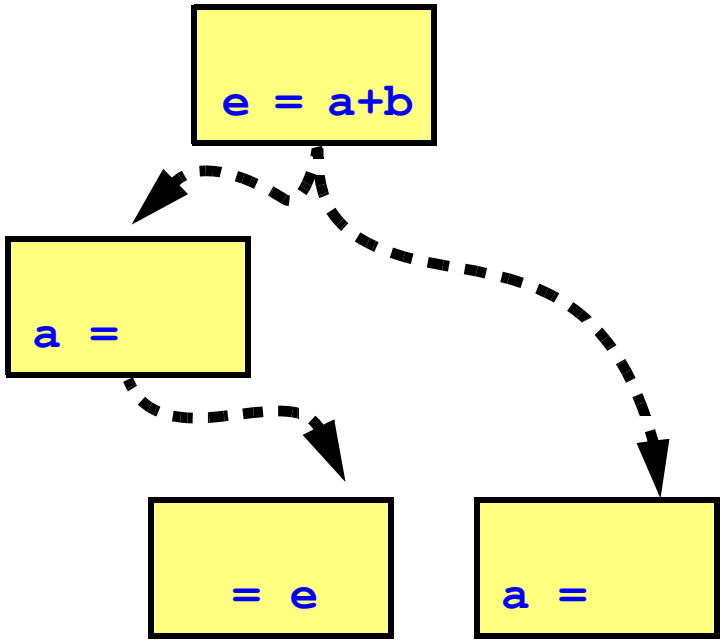


- Whether I or II is better depends on the objective function; but we cannot get from II to III using the standard PRE transformations.

  If the original program is like II, we may want to get to III (to reduce register pressure, to reduce size, ...)

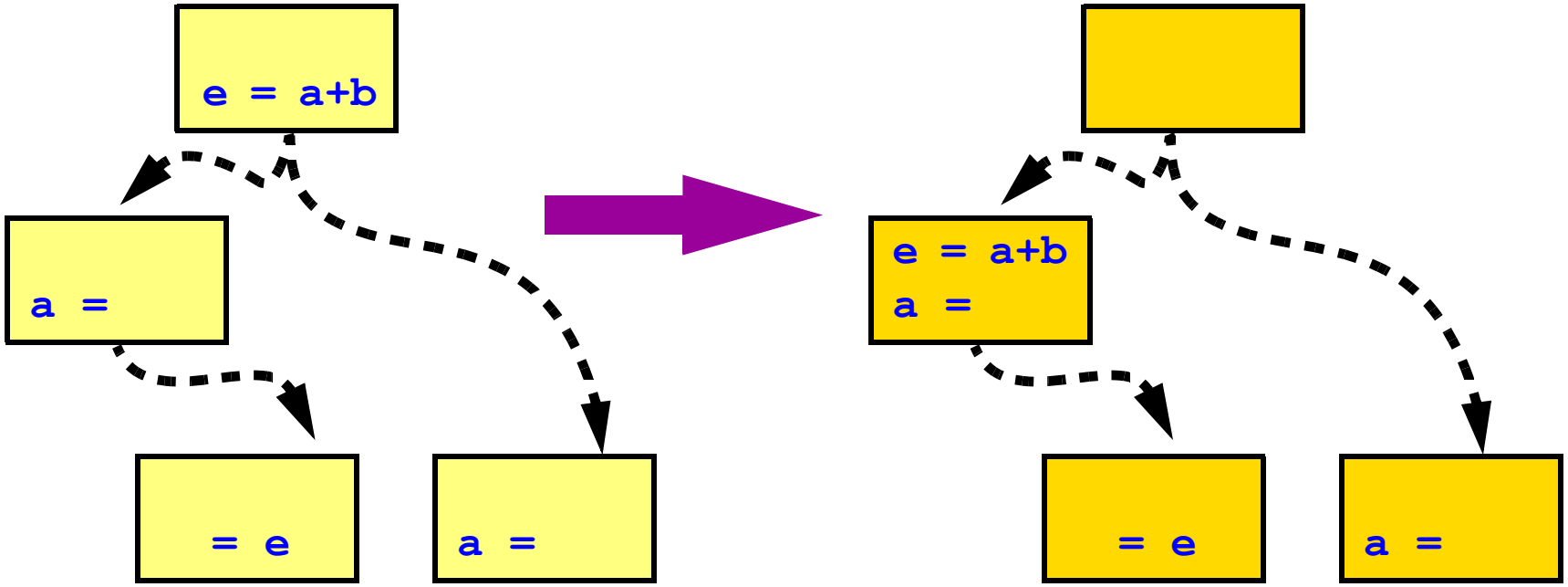
Partial Dead Code Elimination (PDE)

An expression *e* is not used on all subsequent execution paths ...



Partial Dead Code Elimination (PDE)

An expression *e* is not used on all subsequent execution paths ...



- we can push the evaluation later to point(s) where the expression is needed on all subsequent paths.

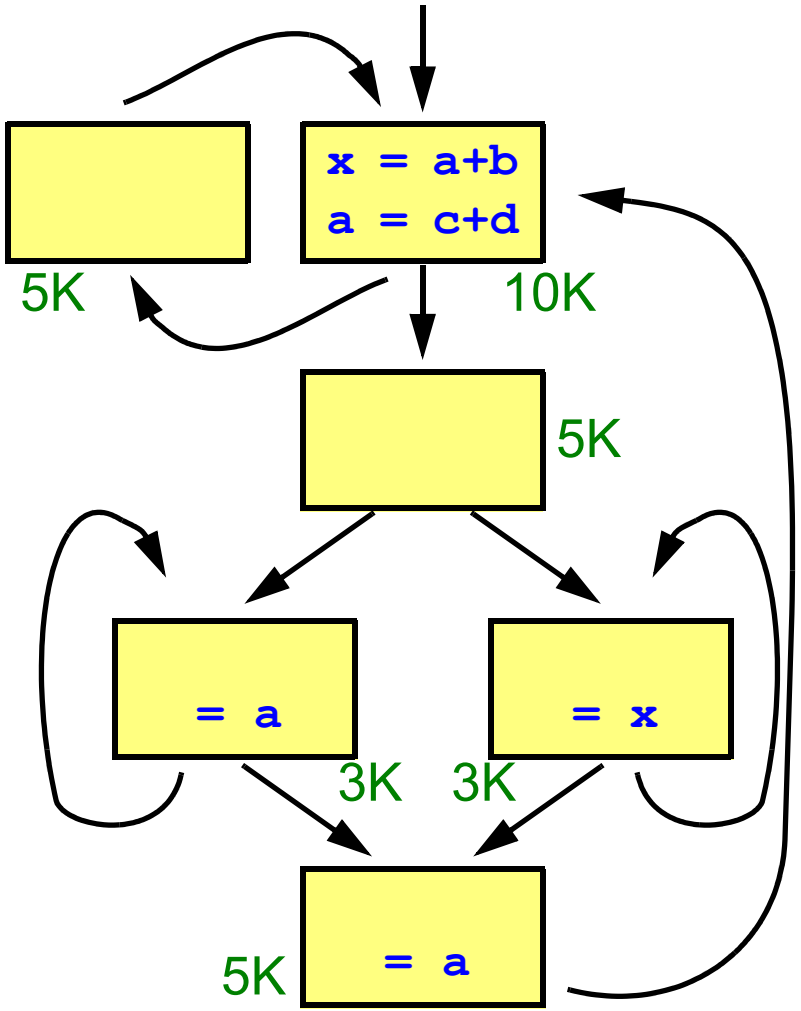
  (This is *assignment sinking*.)

From PDE to Speculative PDE

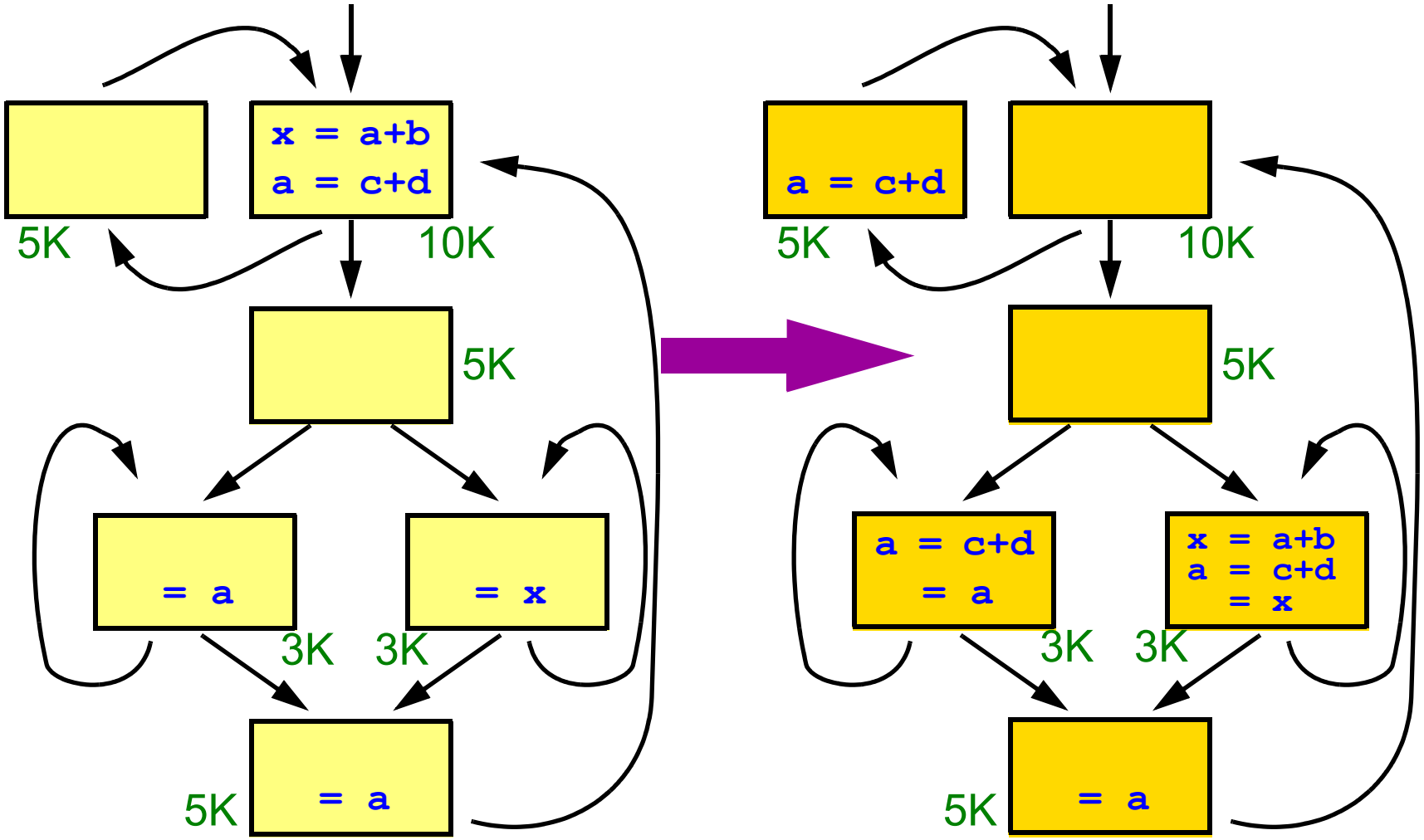
- A misnomer because there is no speculation involved.
- However, it makes sense to associate costs with expression evaluations which depend on the execution frequency.
- Assignment sinking reduces register pressure.
- Assignment sinking may increase program size.
- Note that optimal solutions will not be found efficiently because of second order effects – placement of one expression affects the placement of another.

  See example ...

2nd order effects in SPDE

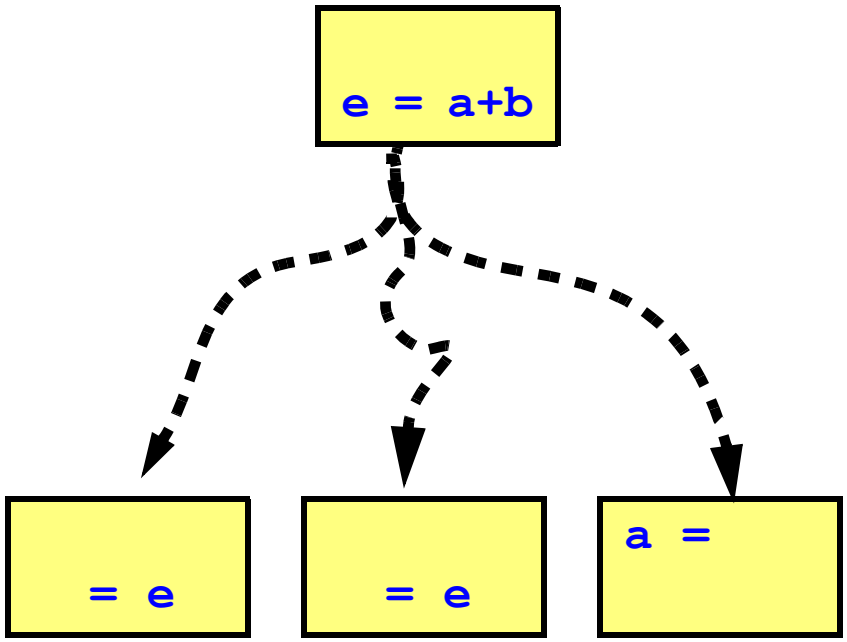


2nd order effects in SPDE

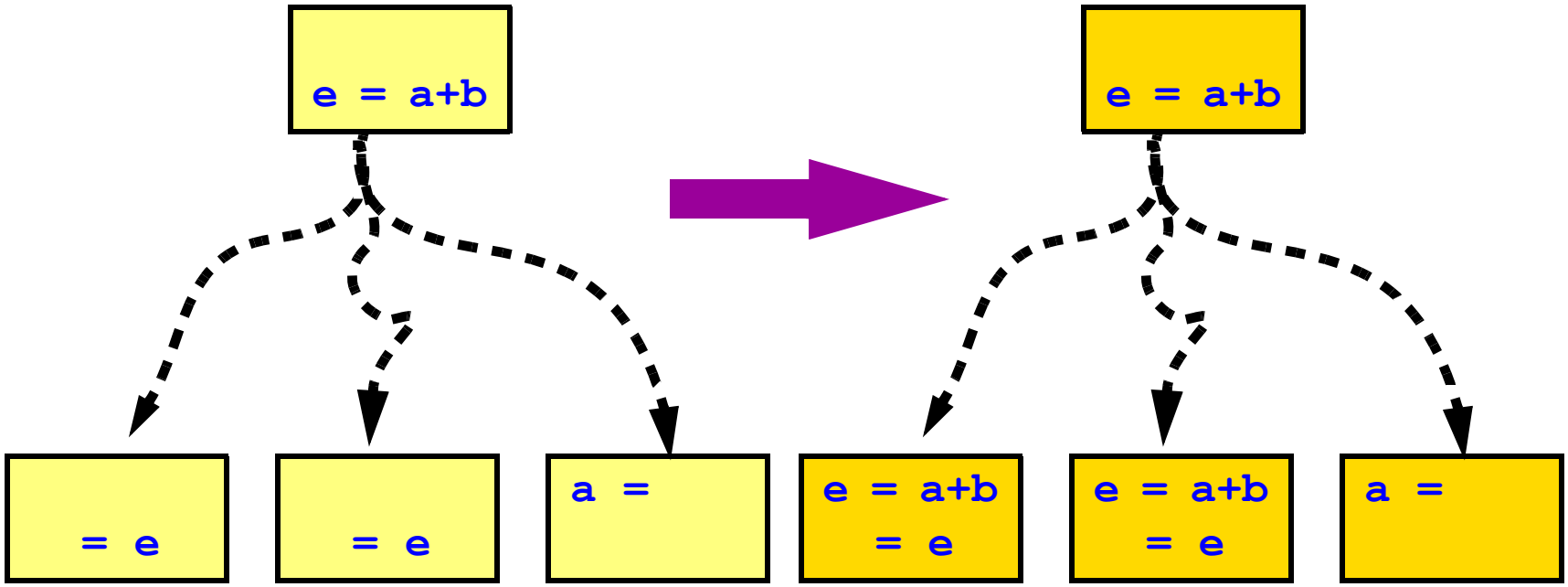


A suboptimal placement of `a=c+d` must be made before placing `x=a+b` and achieving the globally optimal solution.

Mapping SPDE to SPRE



Mapping SPDE to SPRE



- Insert the definition of a variable before each use; then run SPRE to remove redundant evaluations and move them to optimal points.
- If more than one definition reaches a use, we insert the definitions before the point where the control flow paths merge.
- Note: we are ignoring the second-order effects.

Summary

- Execution profiles permit much better results to be obtained from PRE and PDE optimizations.
- Our cost metric model permits an appropriate trade-off between (estimates of)
  - execution time,
  - space,
  - power consumption, and
  - register pressure.
- Only an implementation in a real compiler will show how well it will work in practice.

Thank you!

Any Questions?