ISPRE = Isothermal SPRE

Nigel Horspool, University of Victoria

with much help from
Bernhard Scholz, University of Sydney
David Pereira, University of Victoria
and the motivating idea came from ...
Allan Kielstra, IBM

SPRE

= Speculative PRE

= Speculative Partial Redundancy Elimination

= a very general form of code motion

The goal of code motion is to move computations from hot regions (heavily executed) to cold regions(rarely executed).

An Example of Code Motion

```
i = 0;
while(i<100) {
    a[i] = x*y;
    i++;
}
```

An Example of Code Motion

```
i = 0;
while(i<100) {
    a[i] = x*y;
    i++;
}
```

"Loop invariant code motion"

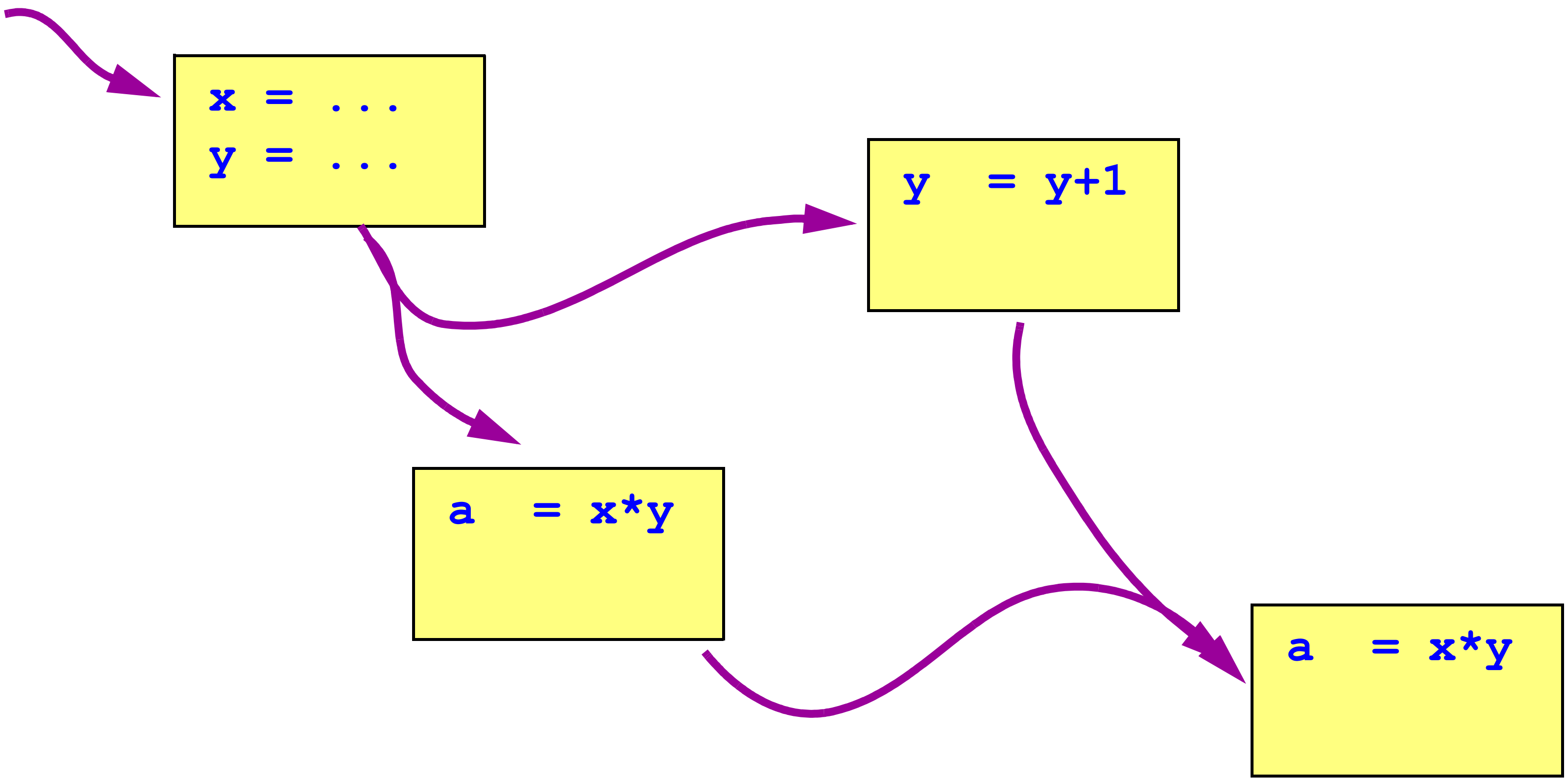
insertion

```
i = 0;
T1 = x*y;
while(i<100) {
    a[i] = T1;
    i++;
}
```

deletion

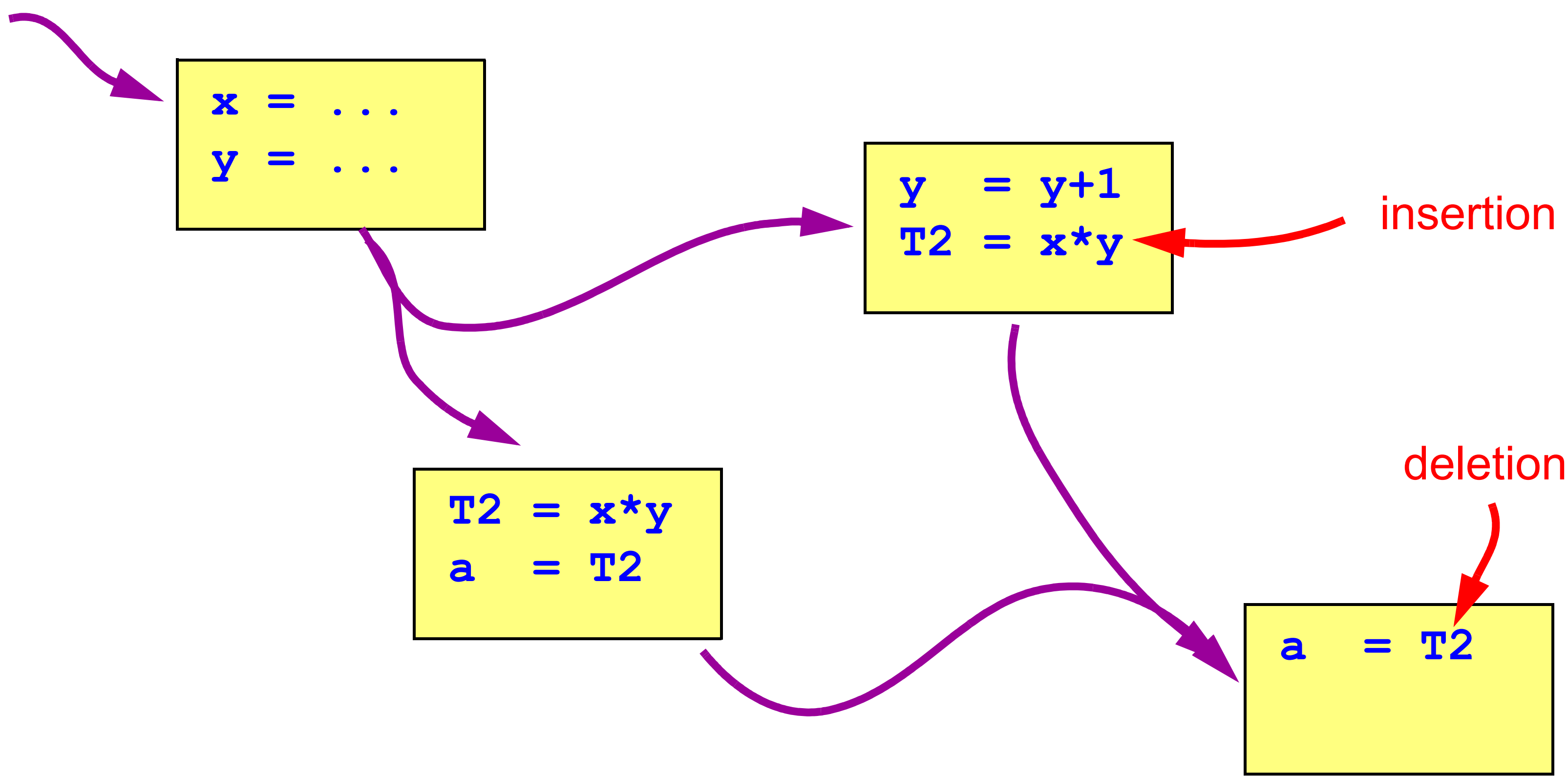
Partial Redundancy Elimination

Generalizes code motion transformations



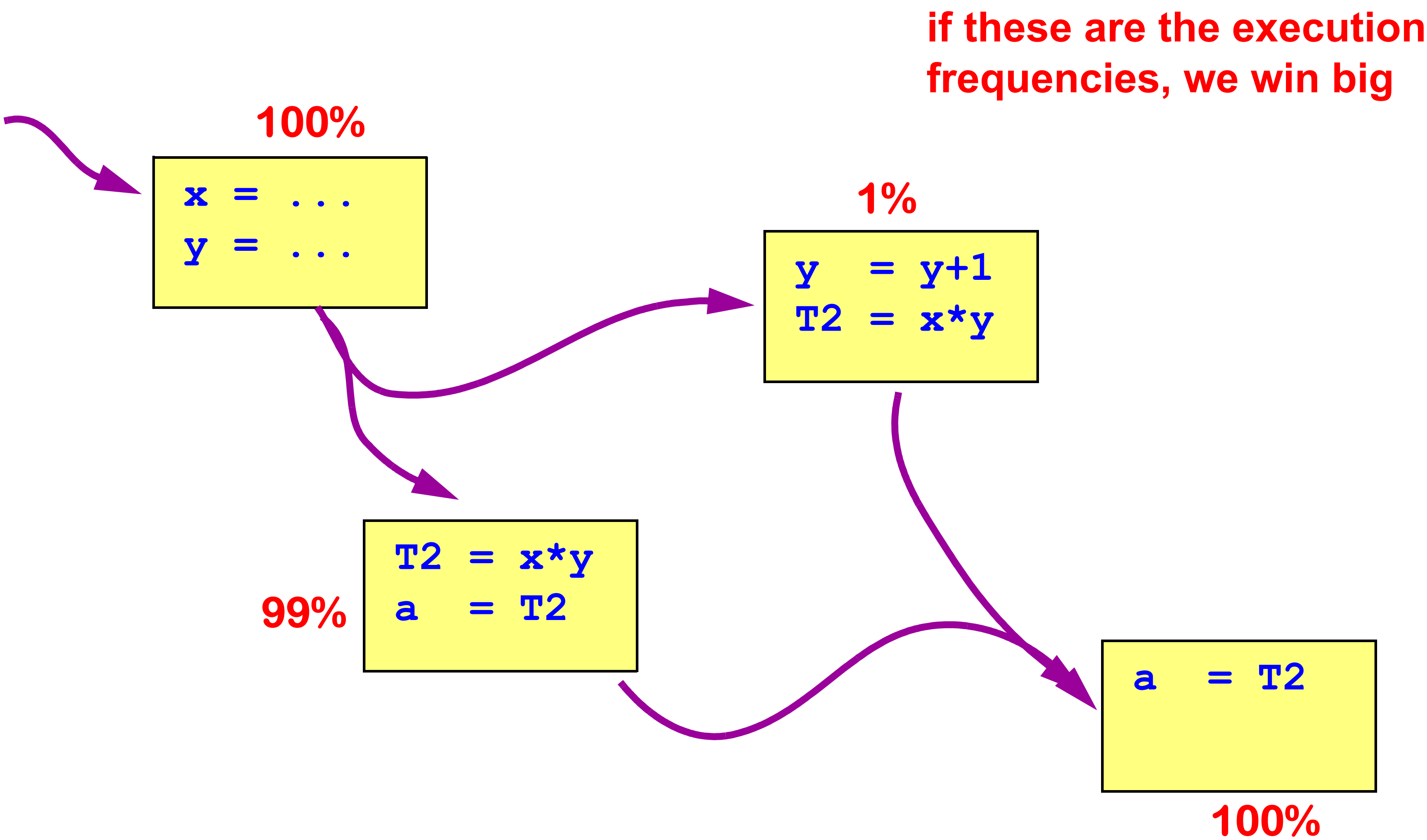
Partial Redundancy Elimination

Generalizes code motion transformations

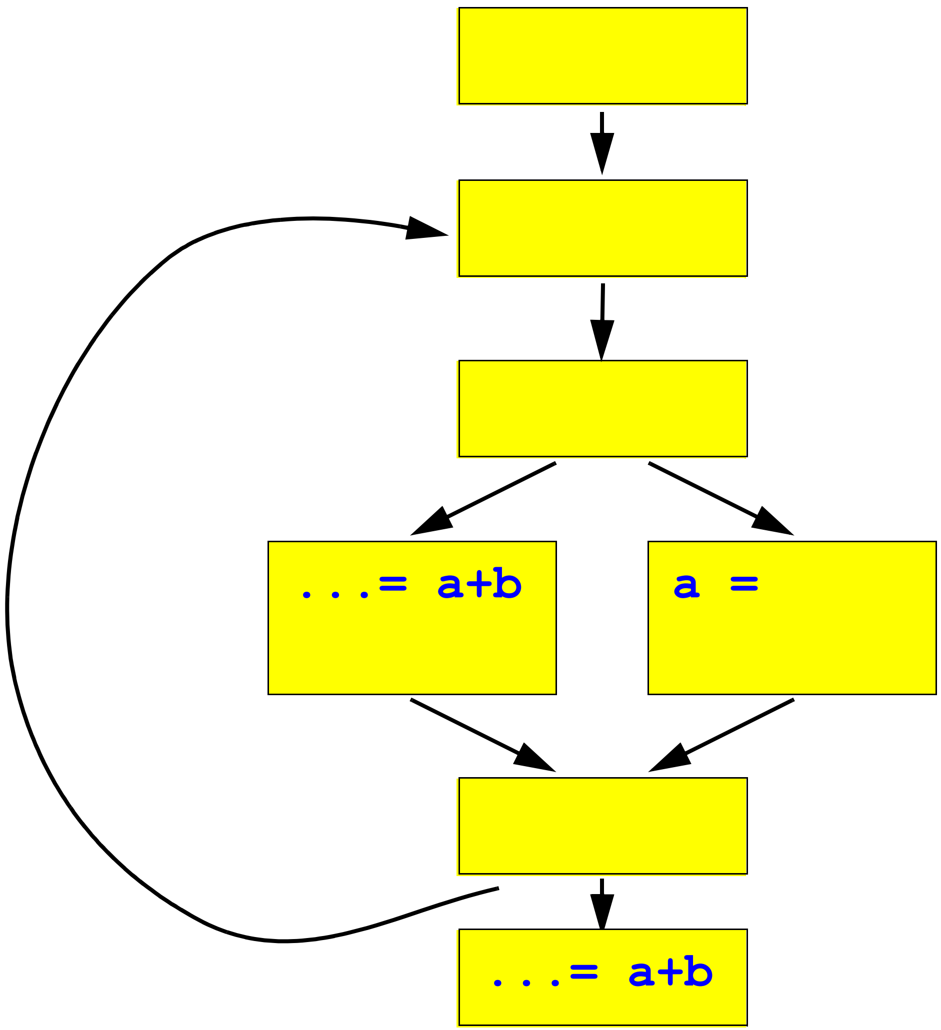


Partial Redundancy Elimination

Generalizes code motion transformations



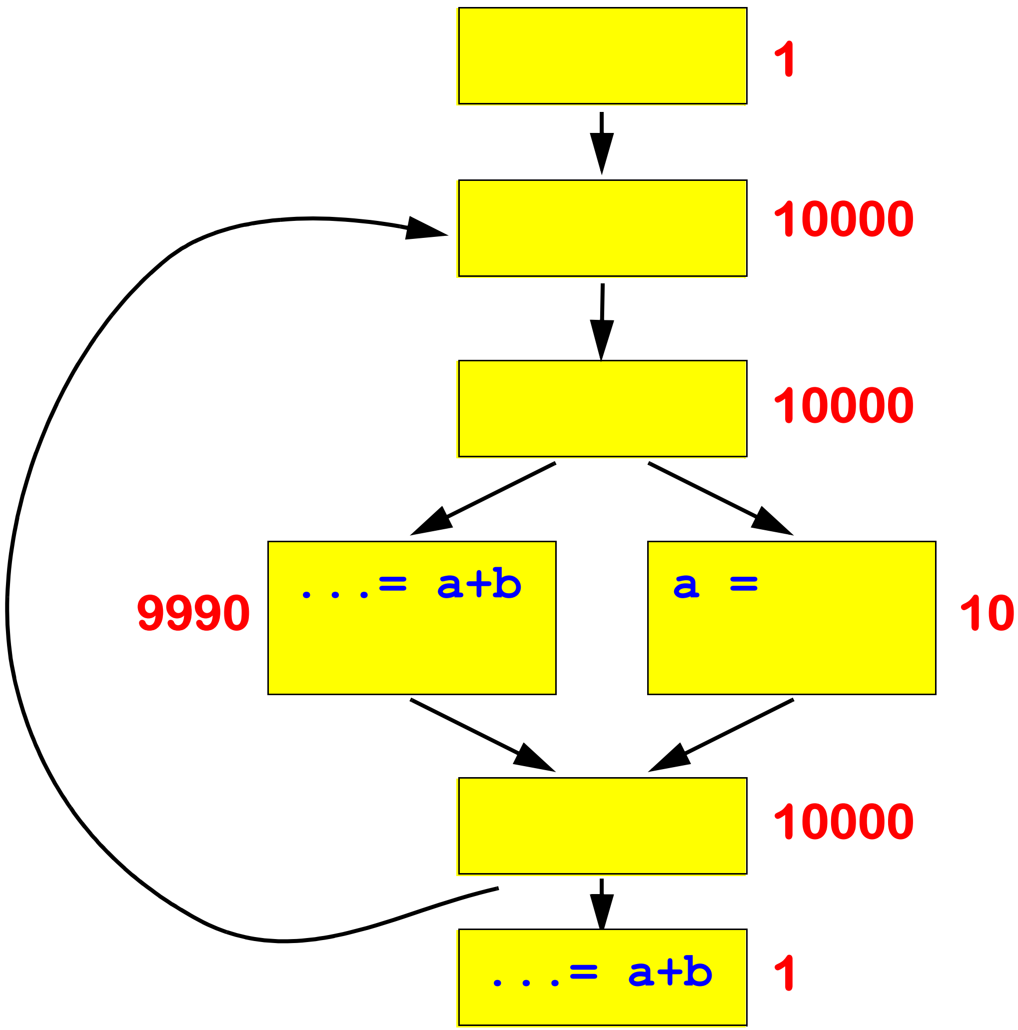
Speculative Partial Redundancy Elimination



This CFG will not be transformed by the classical PRE optimization

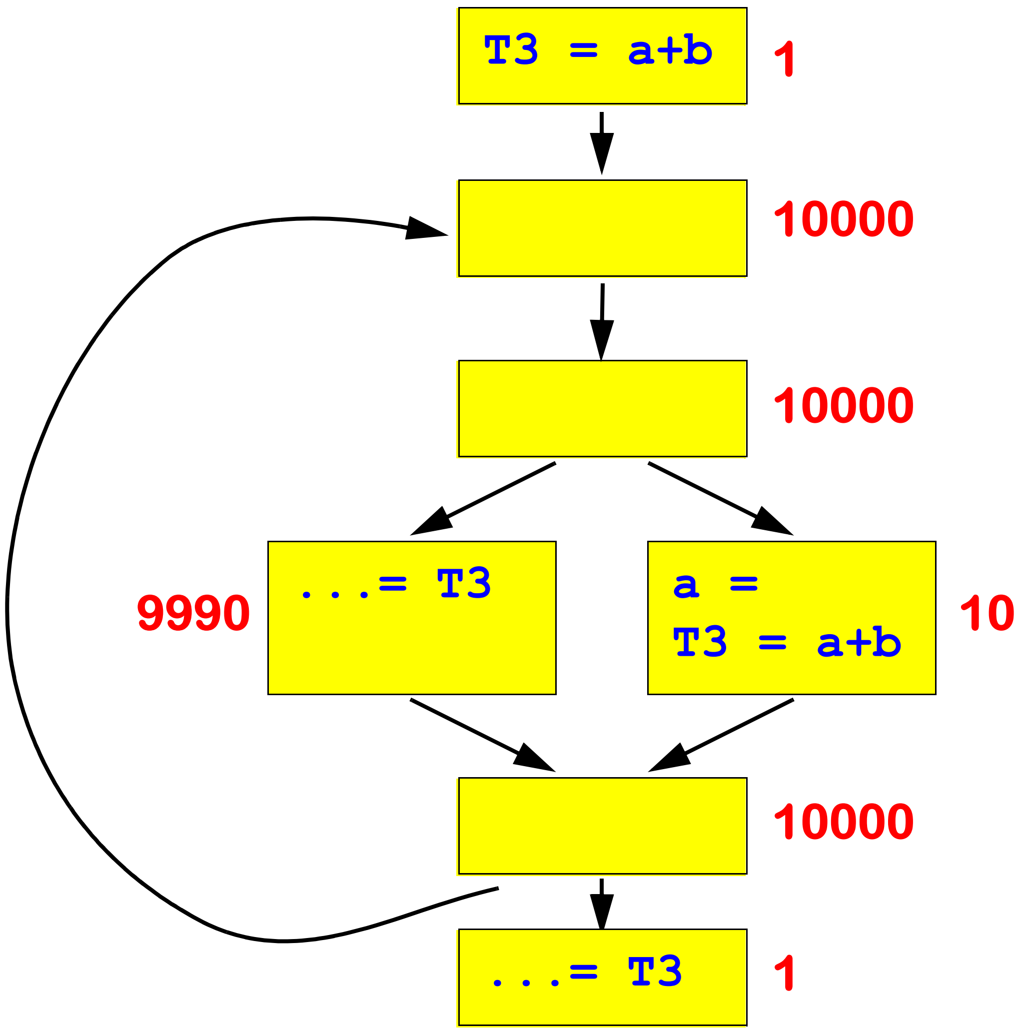
a+b is not anticipable on all execution paths

Speculative Partial Redundancy Elimination



Profile information may show that there is benefit from moving those expressions which are safe

Speculative Partial Redundancy Elimination



Profile information may show that there is benefit from moving those expressions which are safe

It is *speculative* because there is no guarantee that other runs will behave the same way

The Problem with SPRE

- It requires a computationally expensive analysis
  (a separate analysis for every expression)

- It typically performs a lot of code motion in cool program regions where the gains are small

- It greatly increases register pressure

- Needs to be modified to incorporate *laziness* (do not move computations any further than necessary)

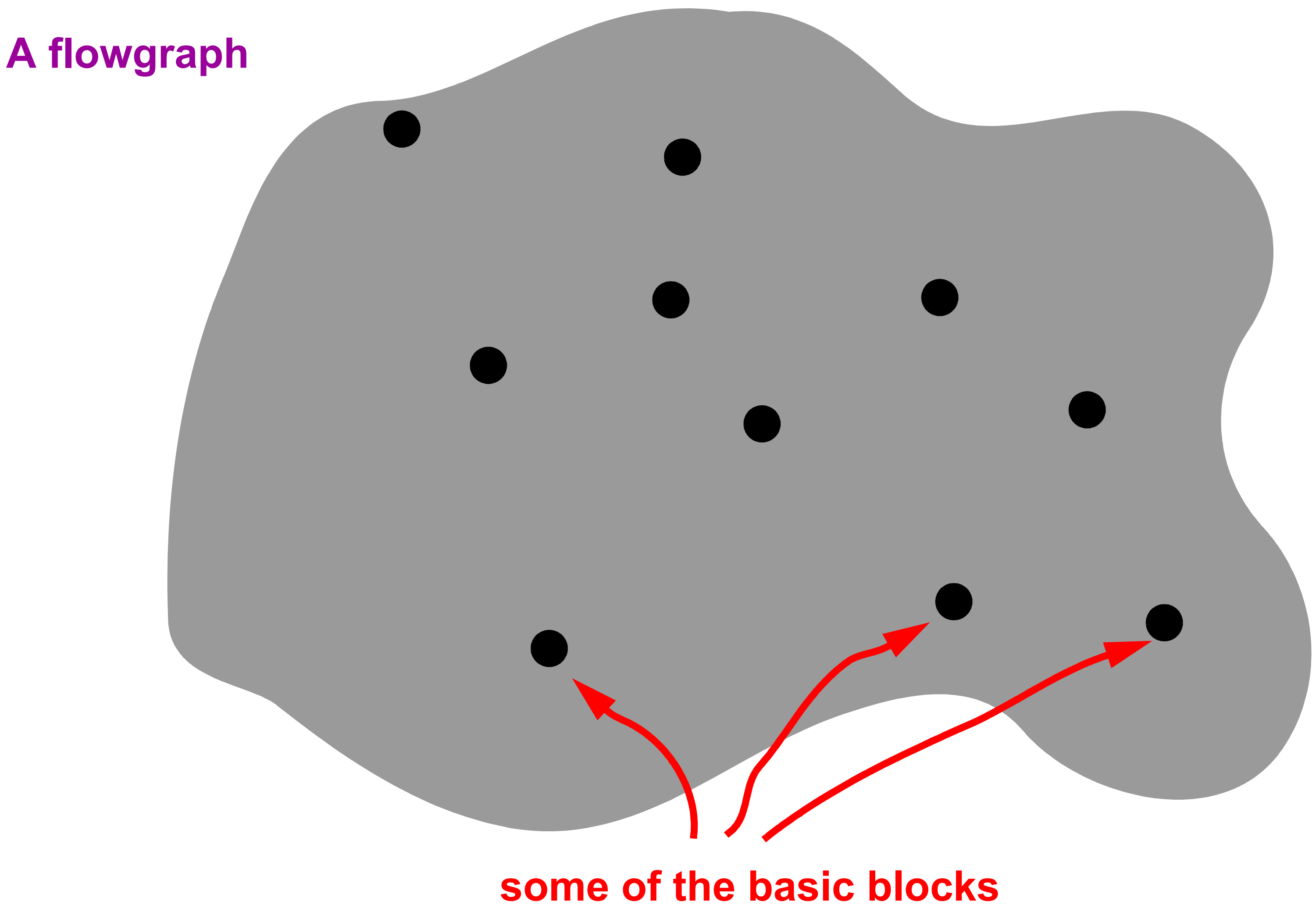
A New Approach

We should be willing to trade optimality for speed of analysis.

What if we just divide the program into hot and cold regions?

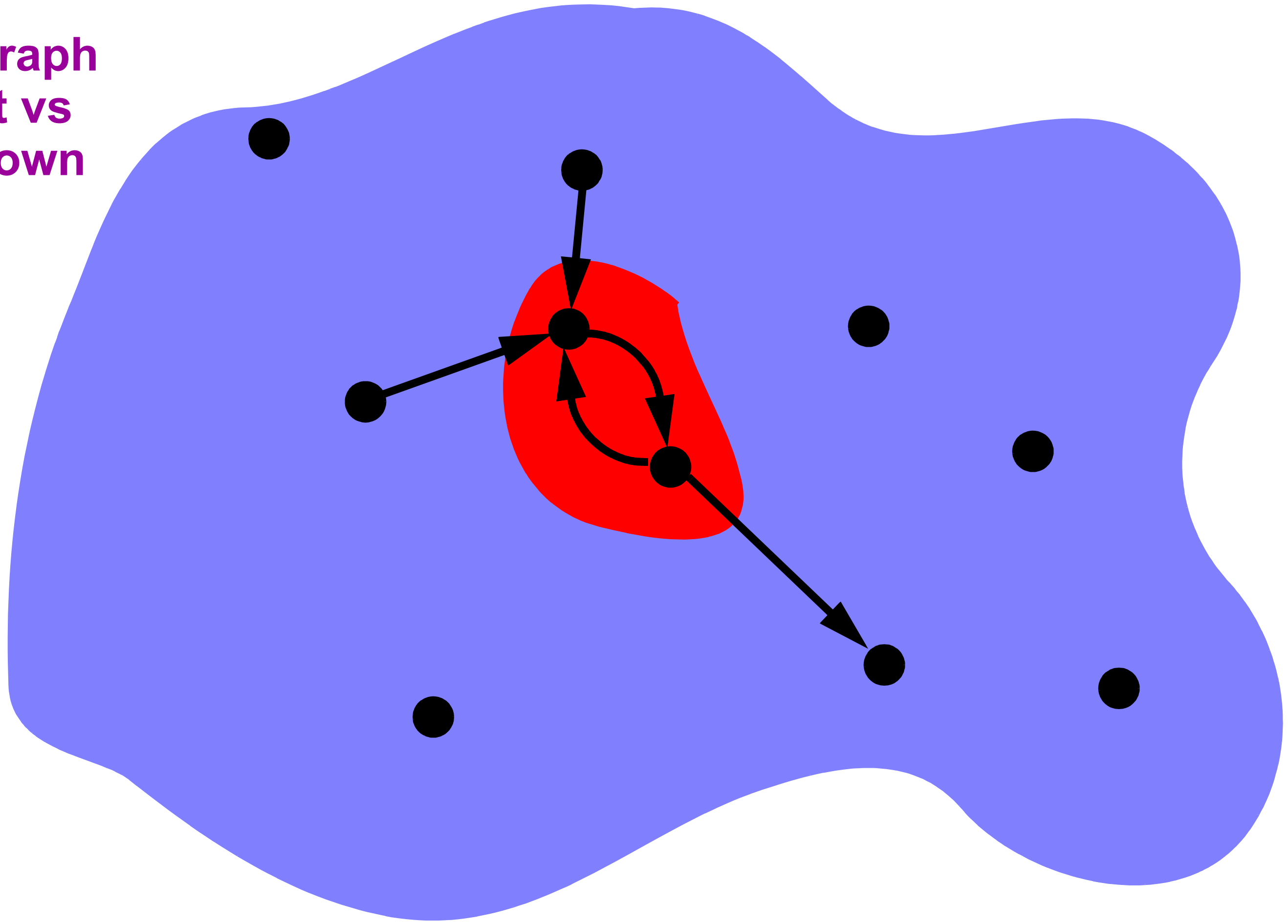
What can we do with that?

Hot and Cold Regions



Hot and Cold Regions

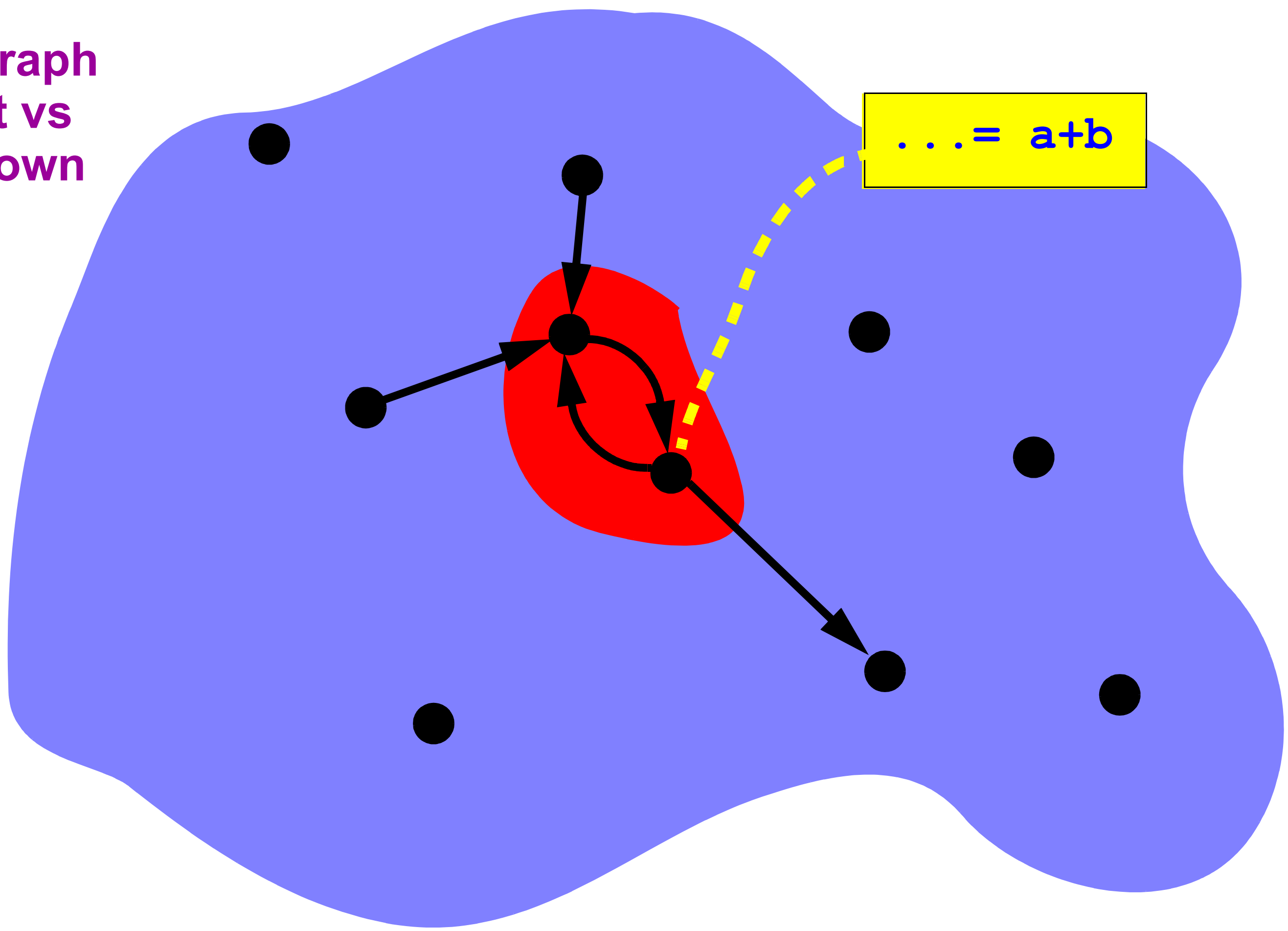
A flowgraph with hot vs cold shown



Let's focus on one expression in the hot region ... a+b

Hot and Cold Regions

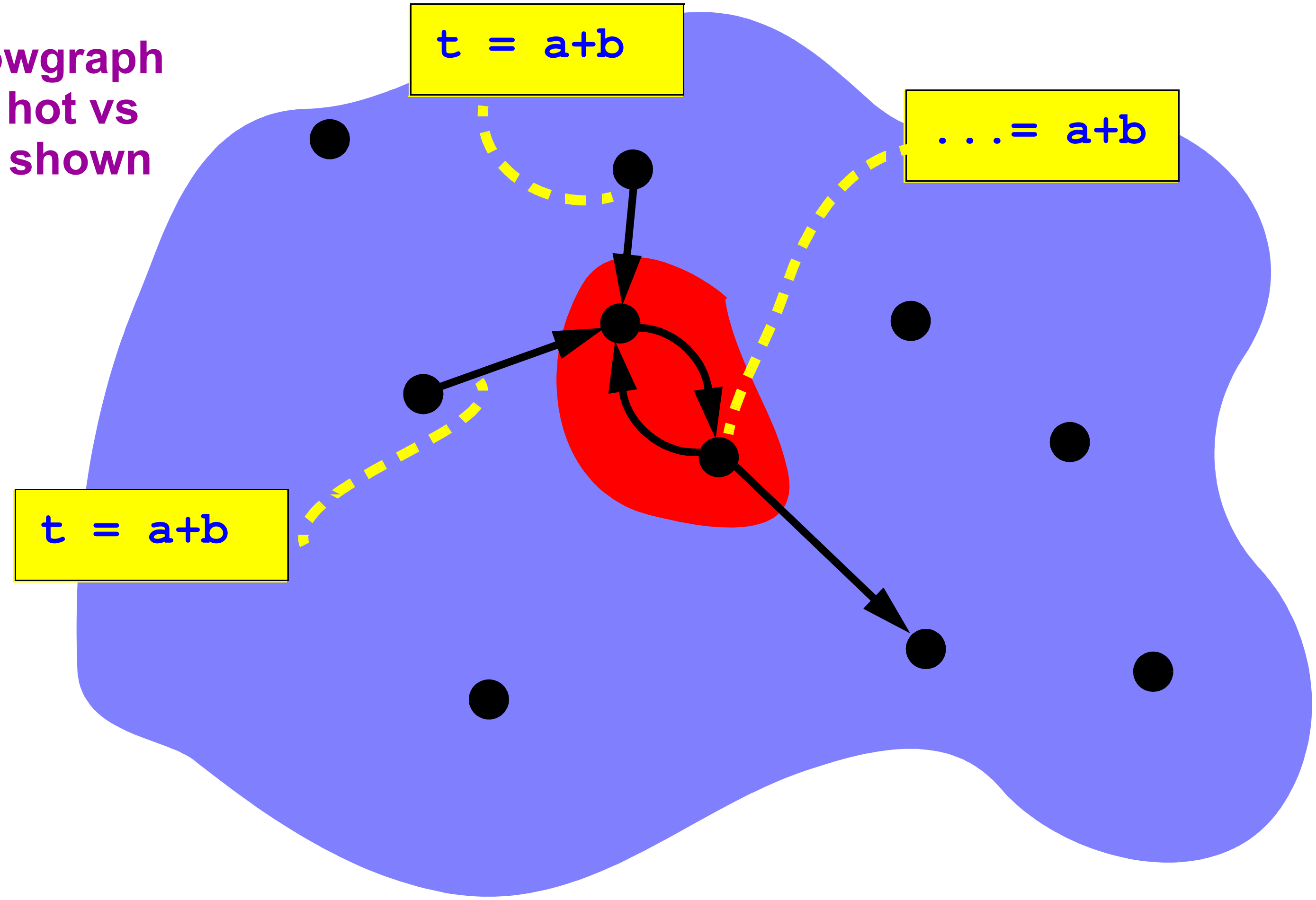
A flowgraph
with hot vs
cold shown



Now we insert a+b computations where we cross from cold to hot ...

Hot and Cold Regions

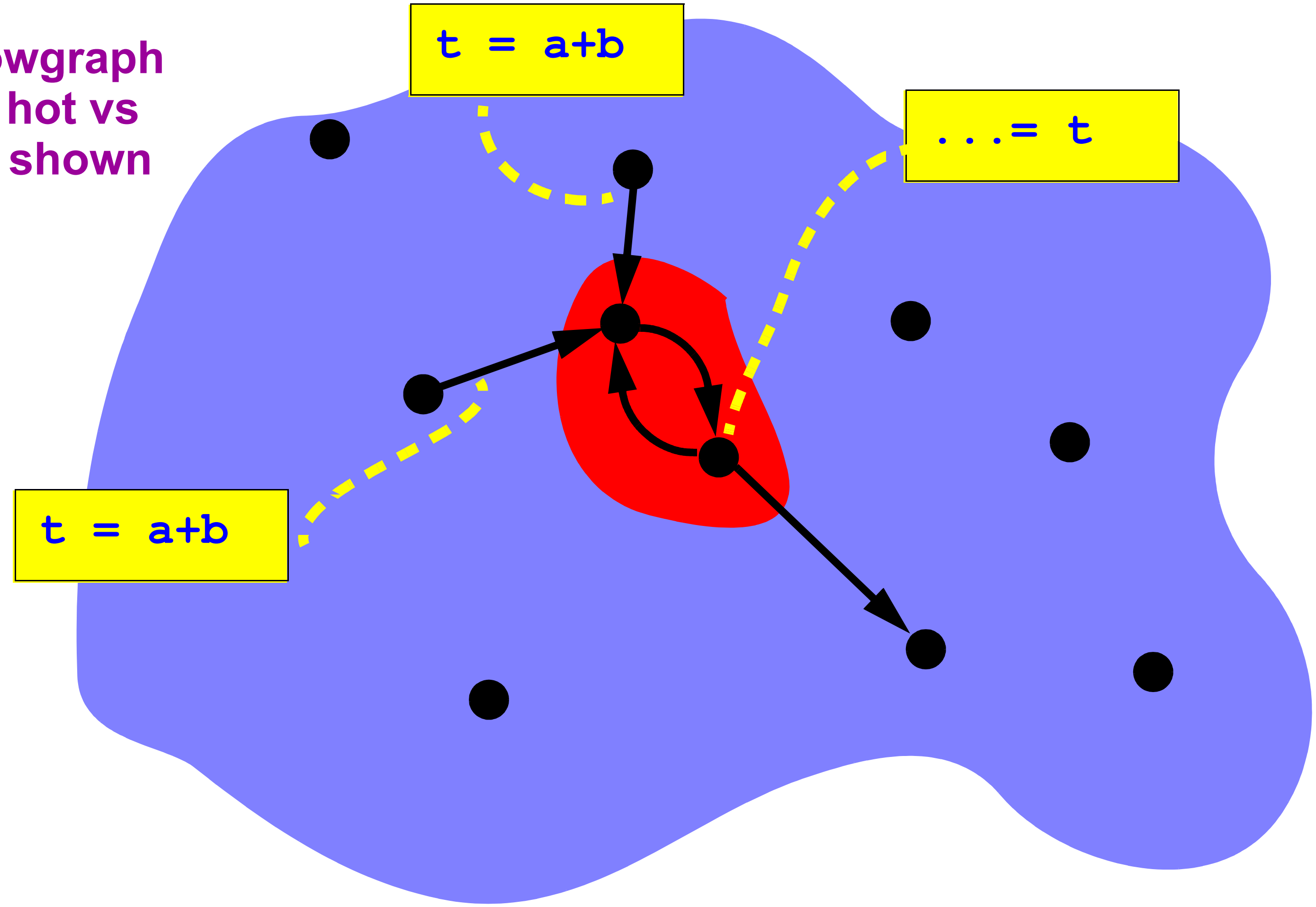
A flowgraph with hot vs cold shown



And we perform available expressions analysis ...

Hot and Cold Regions

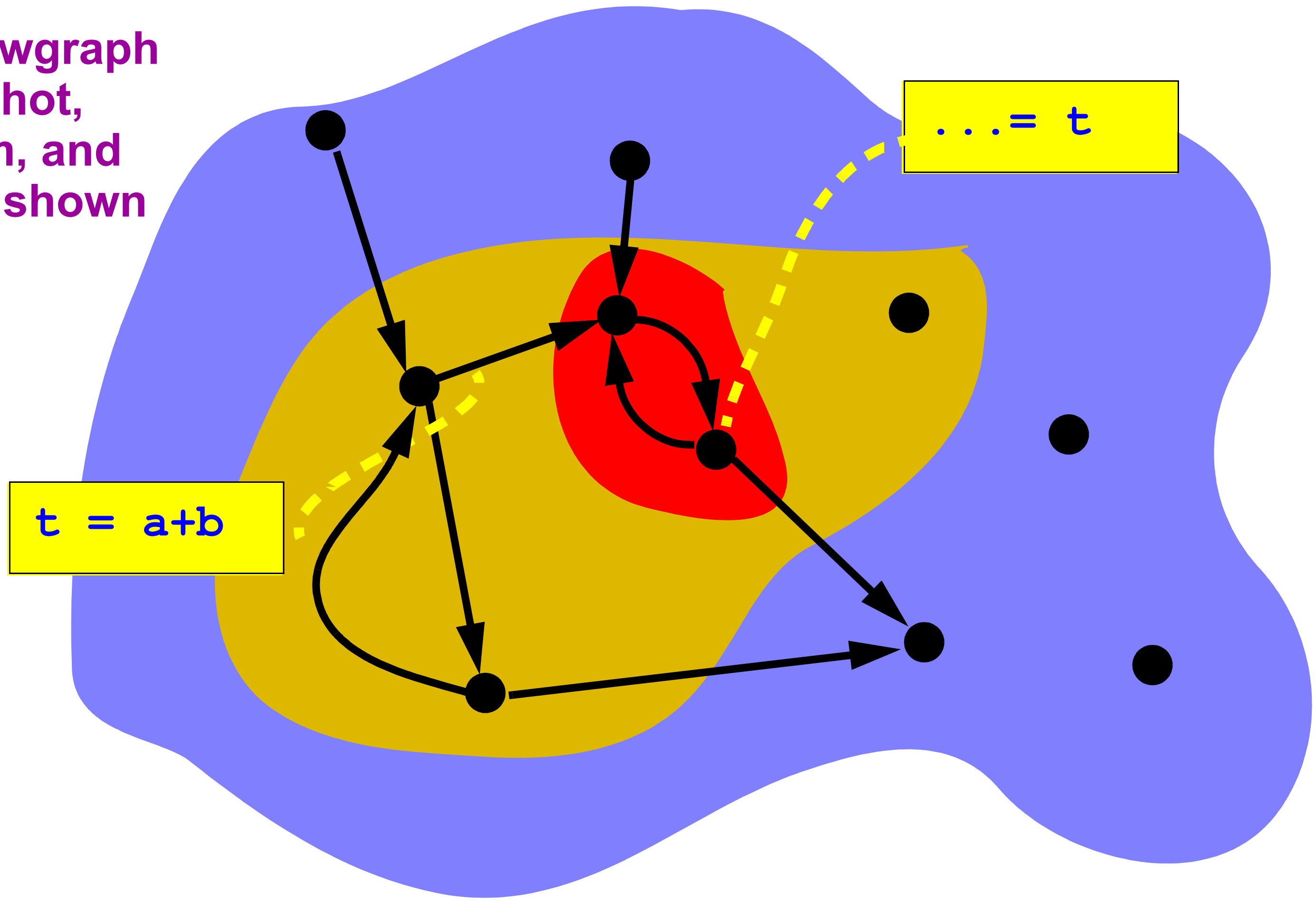
A flowgraph
with hot vs
cold shown



The a+b in the hot region was *fully* redundant and deleted!

Hot, Warm and Cold Regions

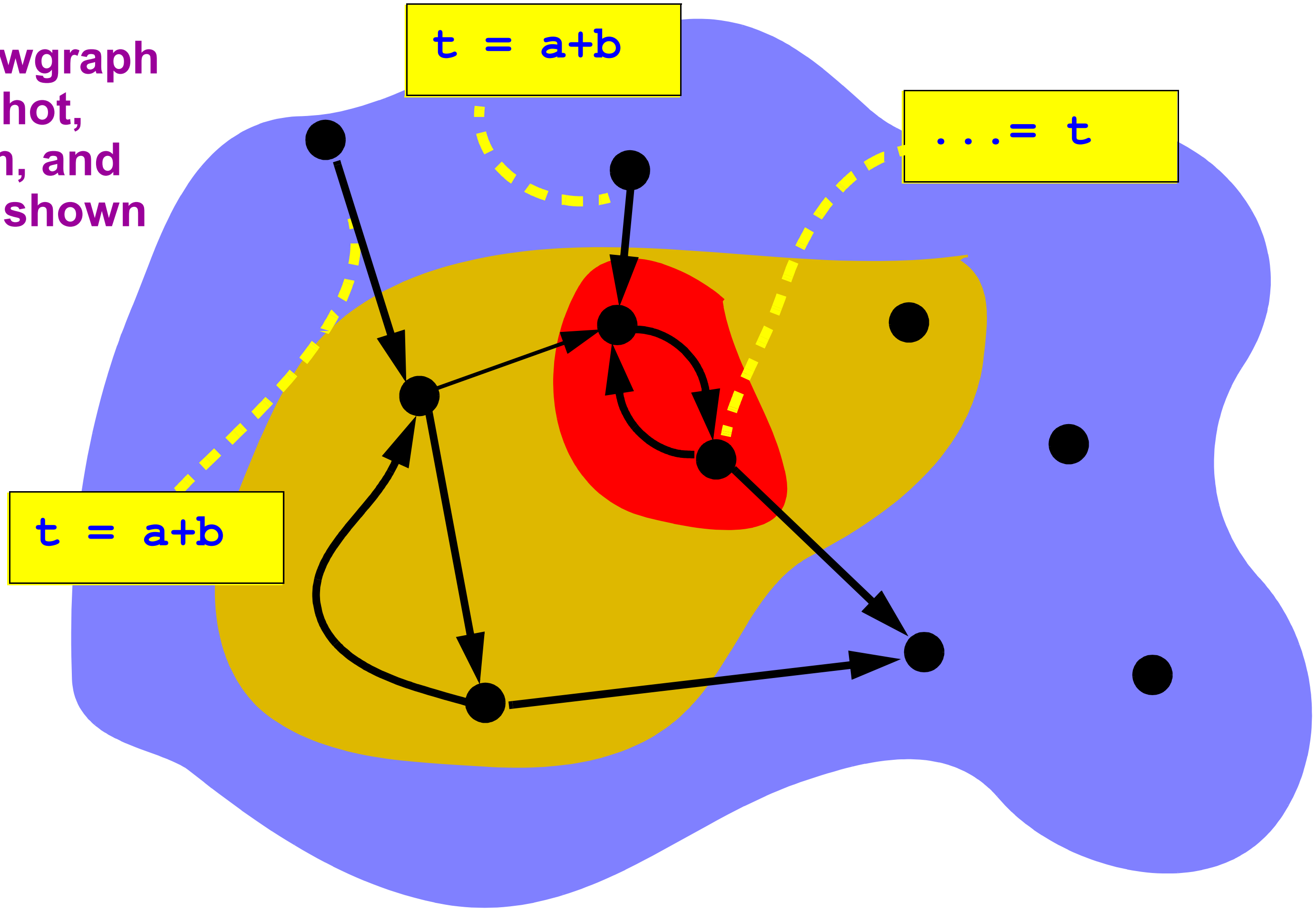
A flowgraph with hot, warm, and cold shown



We can repeat the process with a smaller temperature threshold ...

Hot, Warm and Cold Regions

A flowgraph with hot, warm, and cold shown



Hot, Warm and Cold Regions

and we can repeat with larger and larger cooler and cooler regions until the benefits become too small to pursue.

Note: If the analogy between temperature and execution frequency is continued, then the boundaries between the subgraphs at each level are isothermal lines, hence the name **Isothermal SPRE**.

Overview of the Analysis

First Pass – a forwards analysis

- Pretend to insert all expressions on all cold–hot edges
- Propagate information forward: performing available expressions analysis (on hot regions only) to determine which uses of expressions are now redundant.
  These expressions are deleted (replaced by uses of temporaries)

Second Pass – a backwards analysis

- From the redundant expressions, explore backward to find which of the inserted expressions are needed, and insert those.
- During this analysis, stores of expressions into temporaries are generated.

Both analyses are **bit-vector problems**; we can solve for all expressions in the flowgraph simultaneously.

Preliminary Implementation Results

- David Pereira has recently completed implementations of ISPRE in both GCC and Jikes. (A Testarossa implementation is coming.)
- Some comparisons between SPRE and ISPRE (just one pass) – showing the reductions in the numbers of dynamic computations:

Benchmark	SPRE	ISPRE	Ratio
swim	650677	648535	99.7%
wave5	3363114	2961535	88.1%
turb3d	1455389	1205252	82.8%
su2cor	1979773	1272940	64.3%
tomcatv	469805	467453	99.5%
apsi	2209155	1725965	78.1%
applu	1552663	1017997	65.6%
hydro2d	2974038	2770647	93.2%
mgrid	741757	694545	93.6%

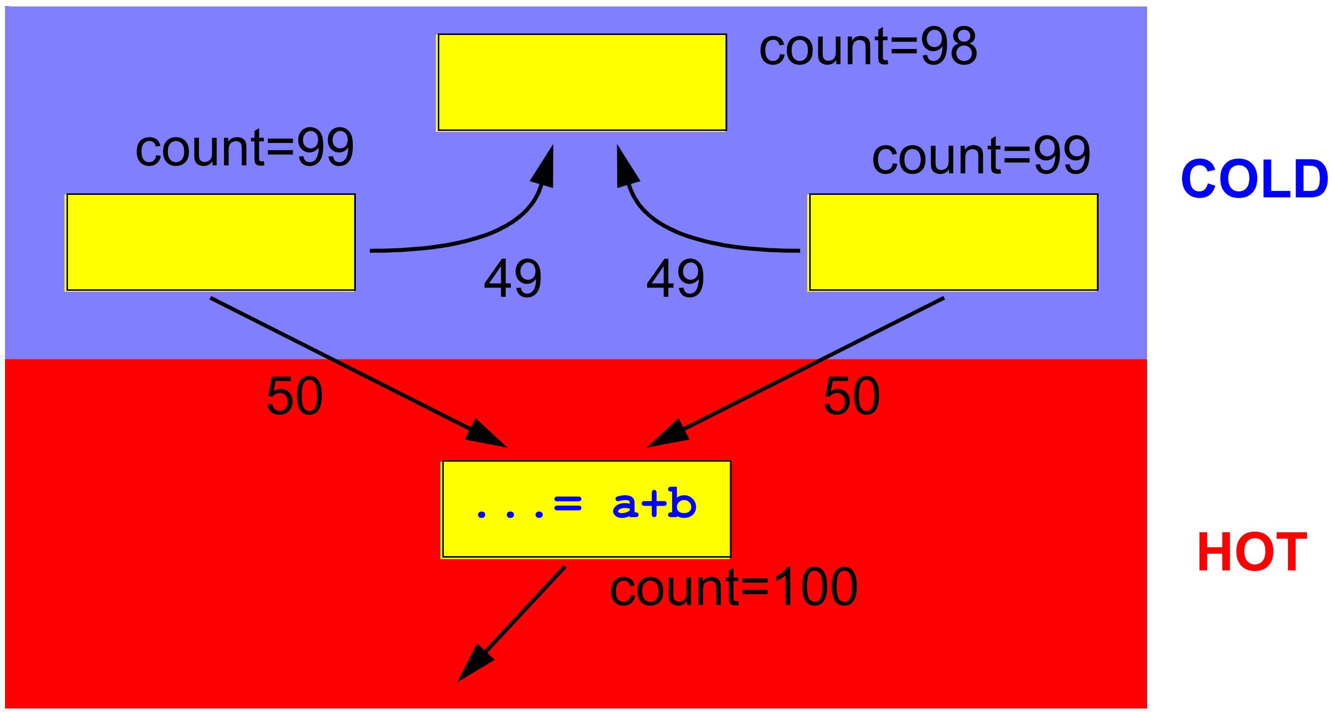
Comments/Discussion

- It's a fast analysis and much simpler than PRE (let alone SPRE).
- Integrating ISPRE into a JIT compiler is much more feasible than PRE or SPRE.
- We do give up all guarantees of optimality; indeed in unfortunate cases ISPRE can make the performance worse.
- ISPRE is inherently lazy – computations are moved to boundaries between hot and cold regions and no further.
- Perhaps we can use static predictions of branch frequencies to guess where the hot regions are? (David will work on this.)
- An interprocedural version is easy to create. (David will work on this!)
- Expressions which can raise exceptions (e.g. `A[i]`) are both a problem and an opportunity. (David will find a remarkable solution.)

Any Questions?

An Unfortunate Case

Assume threshold T = 100 ...



We can make examples as bad as we please.
(We have to hope that the unfortunate insertions are removed again in the next iteration.)