On Time-aware Instrumentation of Programs

Sebastian Fischmeister and Patrick Lam
sfischme@uwaterloo.ca, p.lam@ece.uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

UNIVERSITY OF Waterloo

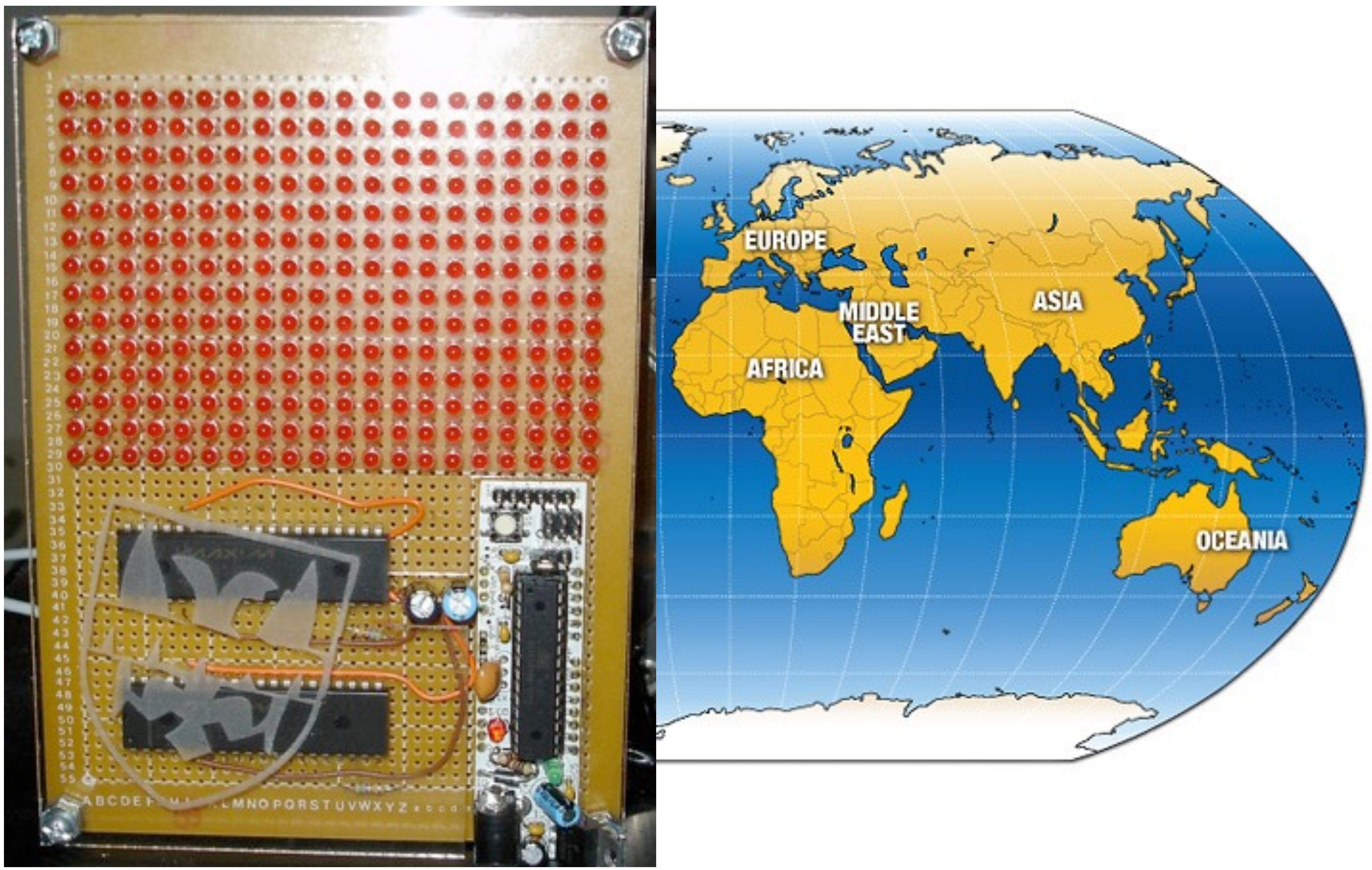
Motivation

- Hard real-time systems
- Background/foreground programs

- Real-time embedded systems are notoriously hard to debug
- 30-50% of development costs are testing and debugging

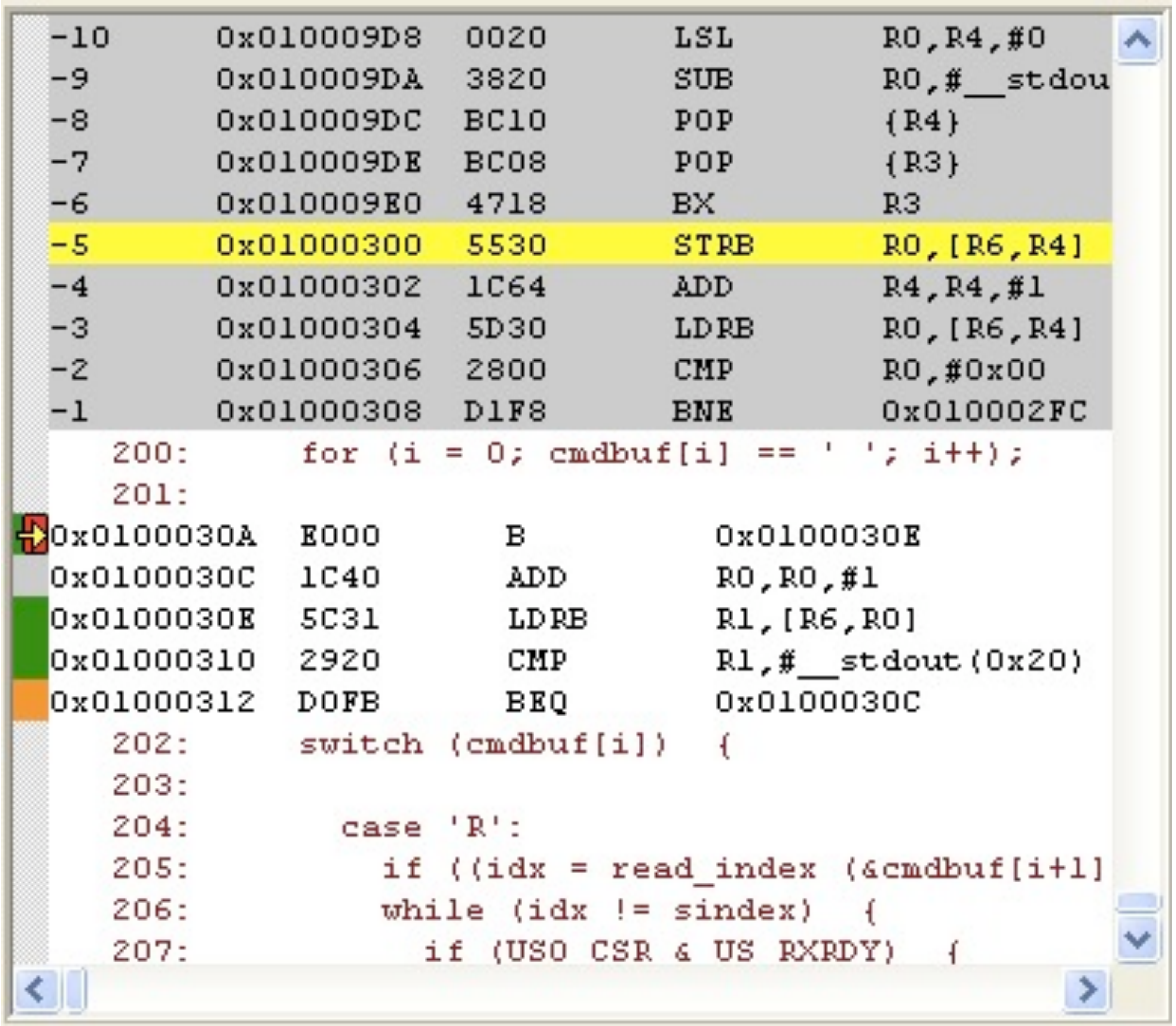
Do you Believe in LED-based Debugging?

Your planetary-scale debugging array:



Tracing

- Record online, replay offline

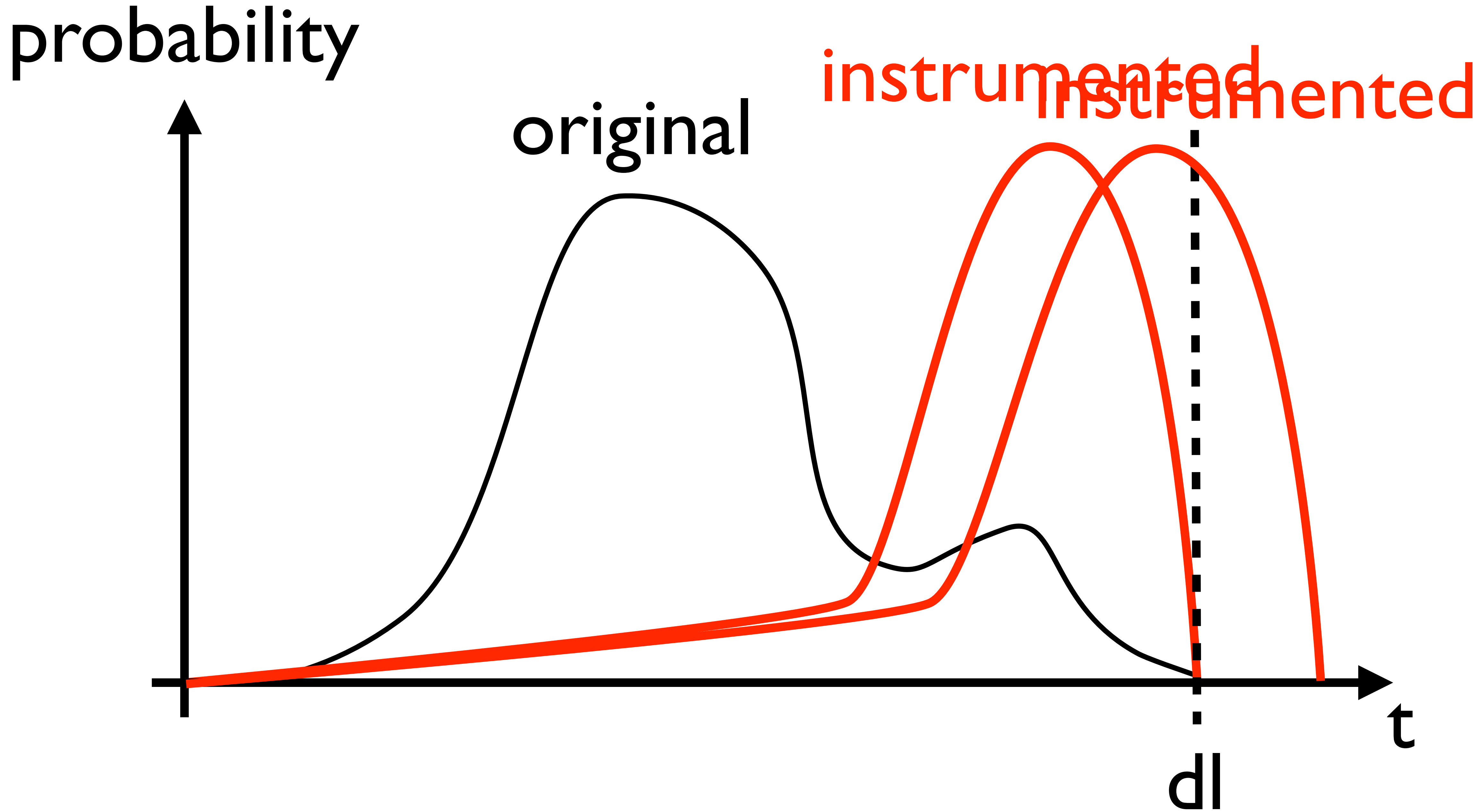


Problem Definition

- How can we automate capturing runtime behavior while minimizing timing interference?
  - Where in the code should you capture information?
  - What to do when you can't capture all?
  - What size do you need for your trace buffer?

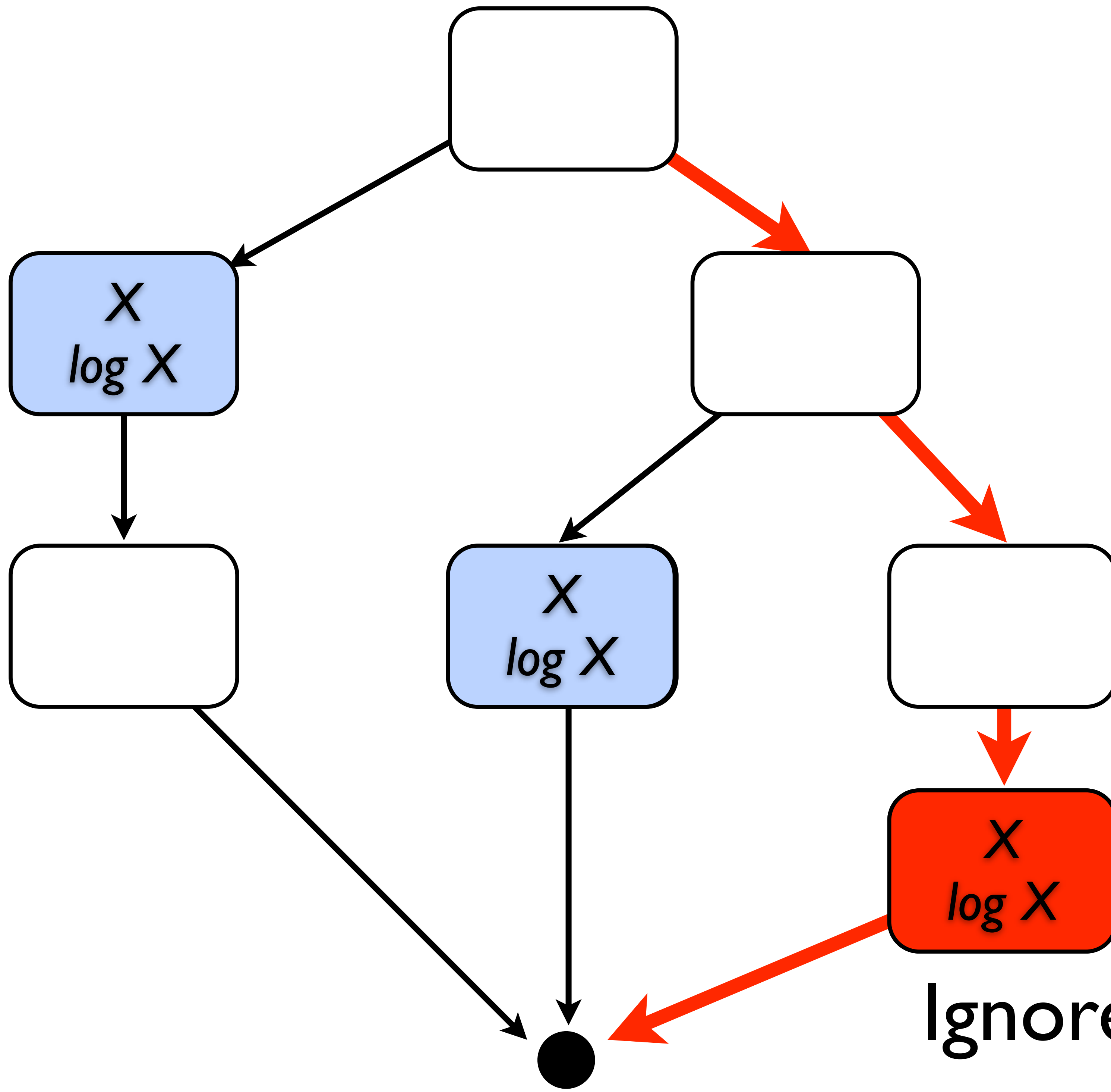
Key Ideas in a Nutshell

But what if it doesn't fit?



Tradeoff: *partial trace & trace reliability*

Example

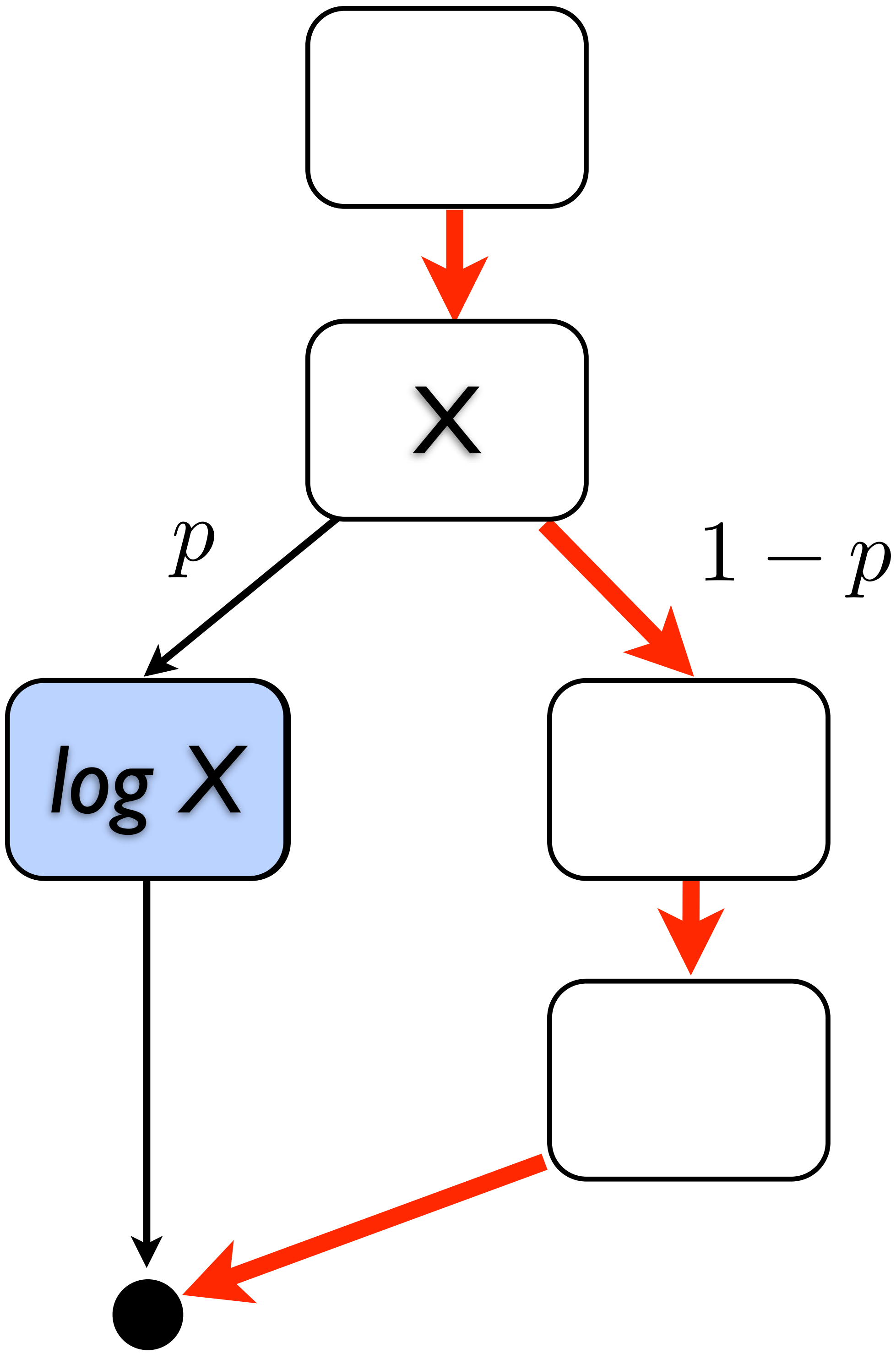


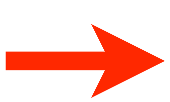
Ignore this one,
create partial trace

→ WCET path

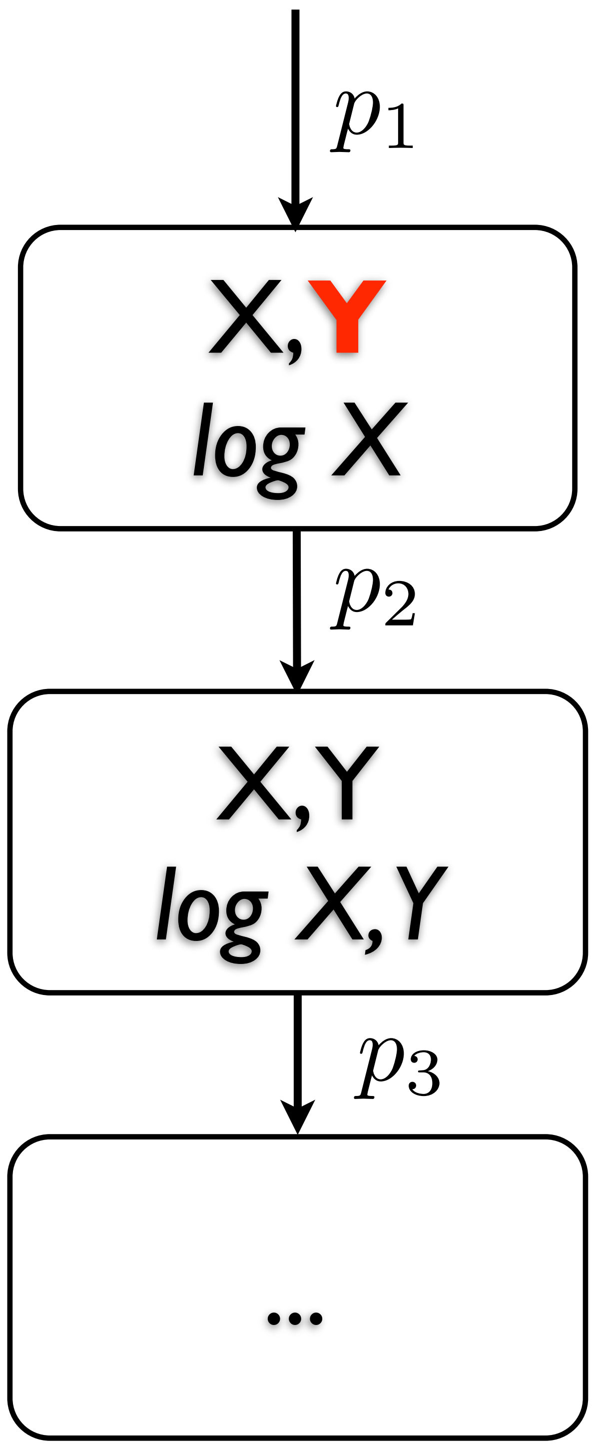
Reliability of Single Assign

Probability of capturing one assignment.



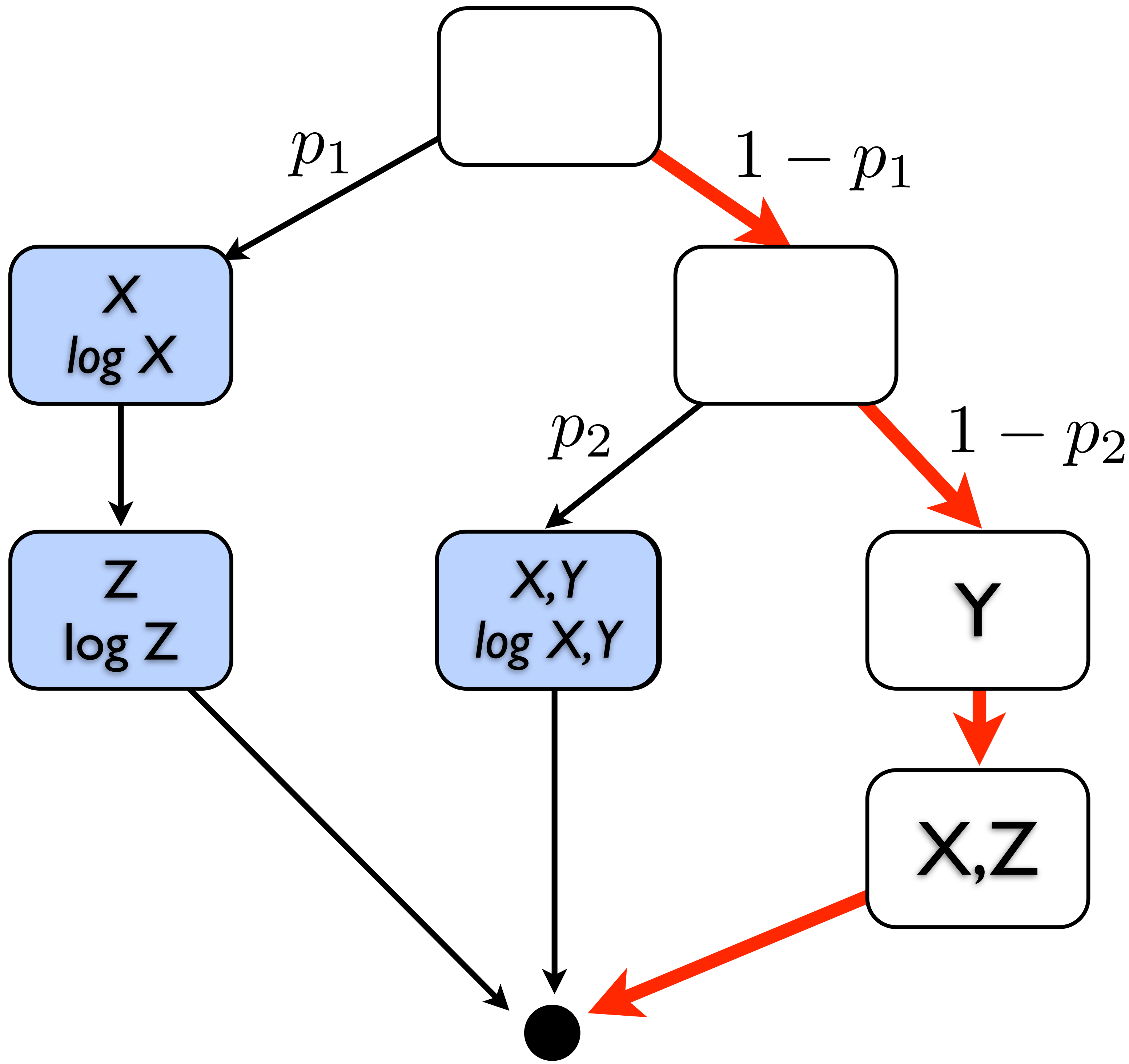


Reliability of A Path



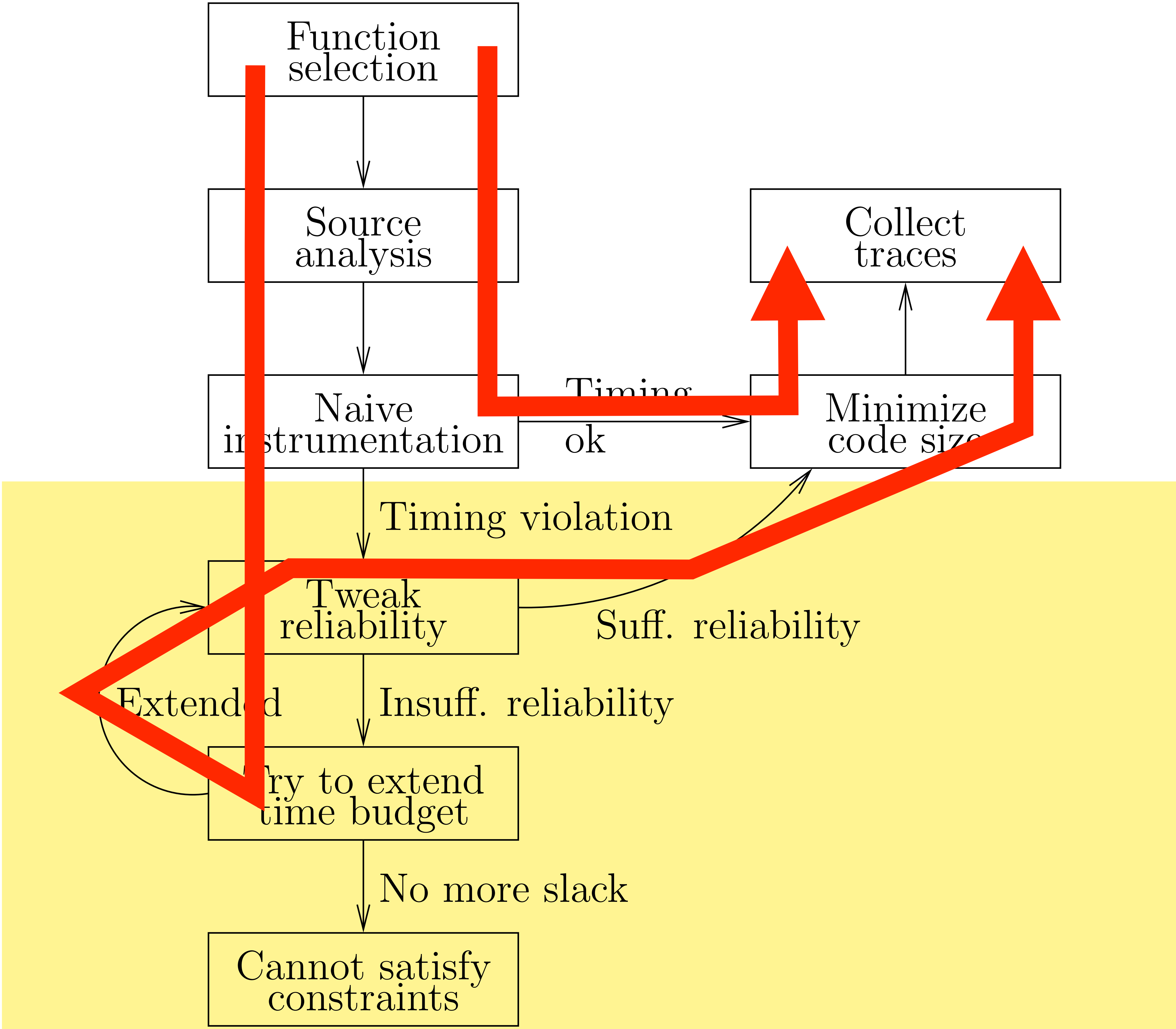
Completeness of the trace for a particular path

Reliability of an Instrumentation



WCET path

Tracing Method



Instrumentation Reliability

Instead of instrumenting every read/write, maximize captures within overhead budget

- Reliability at the insertion point
- Reliability of a path
- Reliability of an instrumentation

Maximize reliability of an instrumentation

Maximal Reliability

$$\max \sum_{p \in P} \sum_{v \in p_i} p(p_i, v_i) x_i \tag{4}$$

$$\sum_{v \in p_0} p(p_0, v_i) \cdot x_i \cdot c(x_i) \leq tb - \sum_{v \in p_0} c'(v_i) \tag{5}$$

$$\dots$$

$$\sum_{v \in p_n} p(p_n, v_i) \cdot x_i \cdot c(x_i) \leq tb - \sum_{v \in p_n} c'(v_i)$$

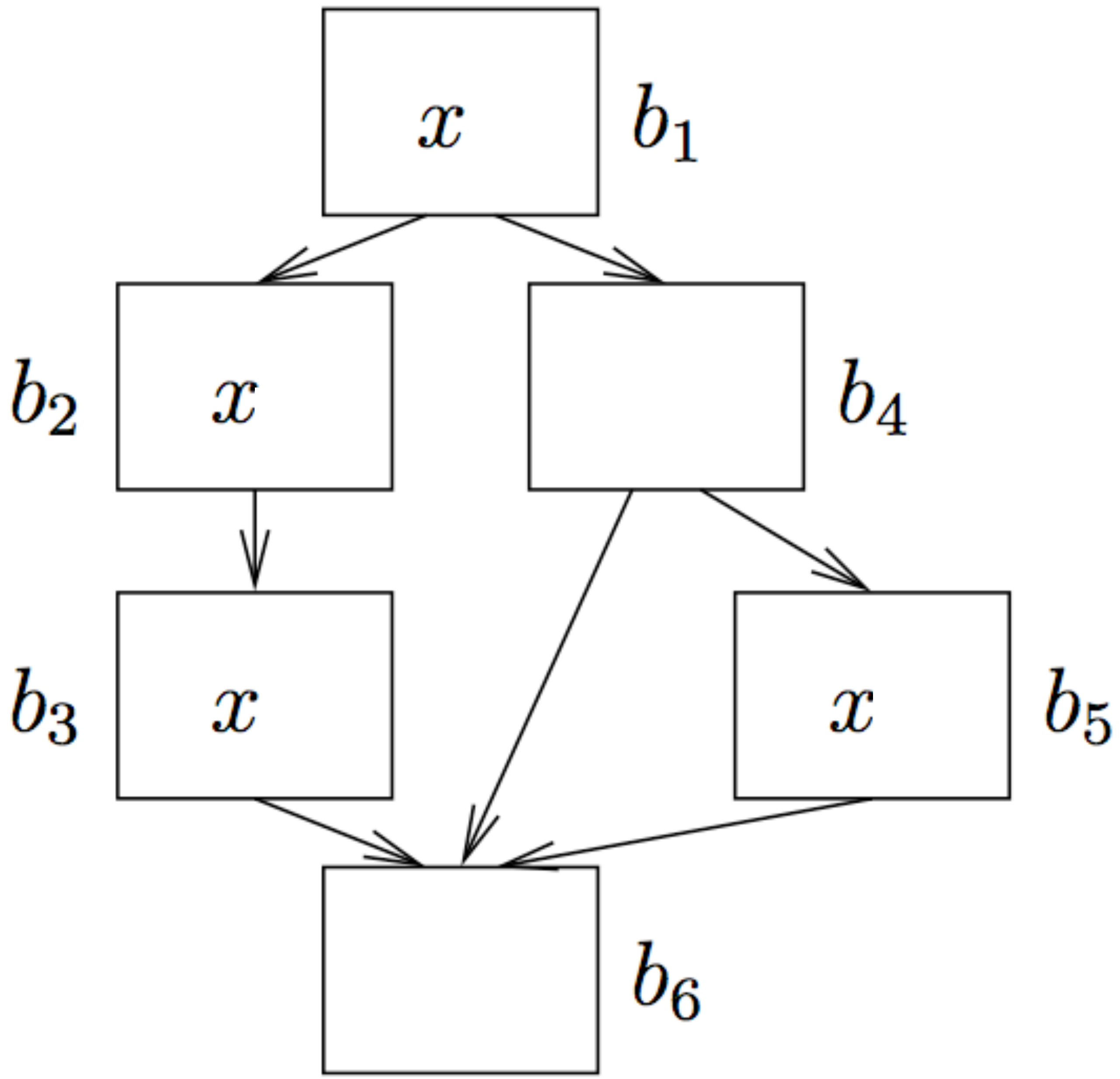
$$x_0 \leq |v_0.A| \tag{6}$$

$$\dots$$

$$x_n \leq |v_n.A|$$

Minimize Insertion Points

Defer captures until next write



Hitting set problem

Case Study: Objectives

- Test feasibility of our approach
- Test our hypothesis of shifting execution time
- Play around and look for surprising things
  - Attainable reliability with zero overhead
  - Increase in overhead vs reliability

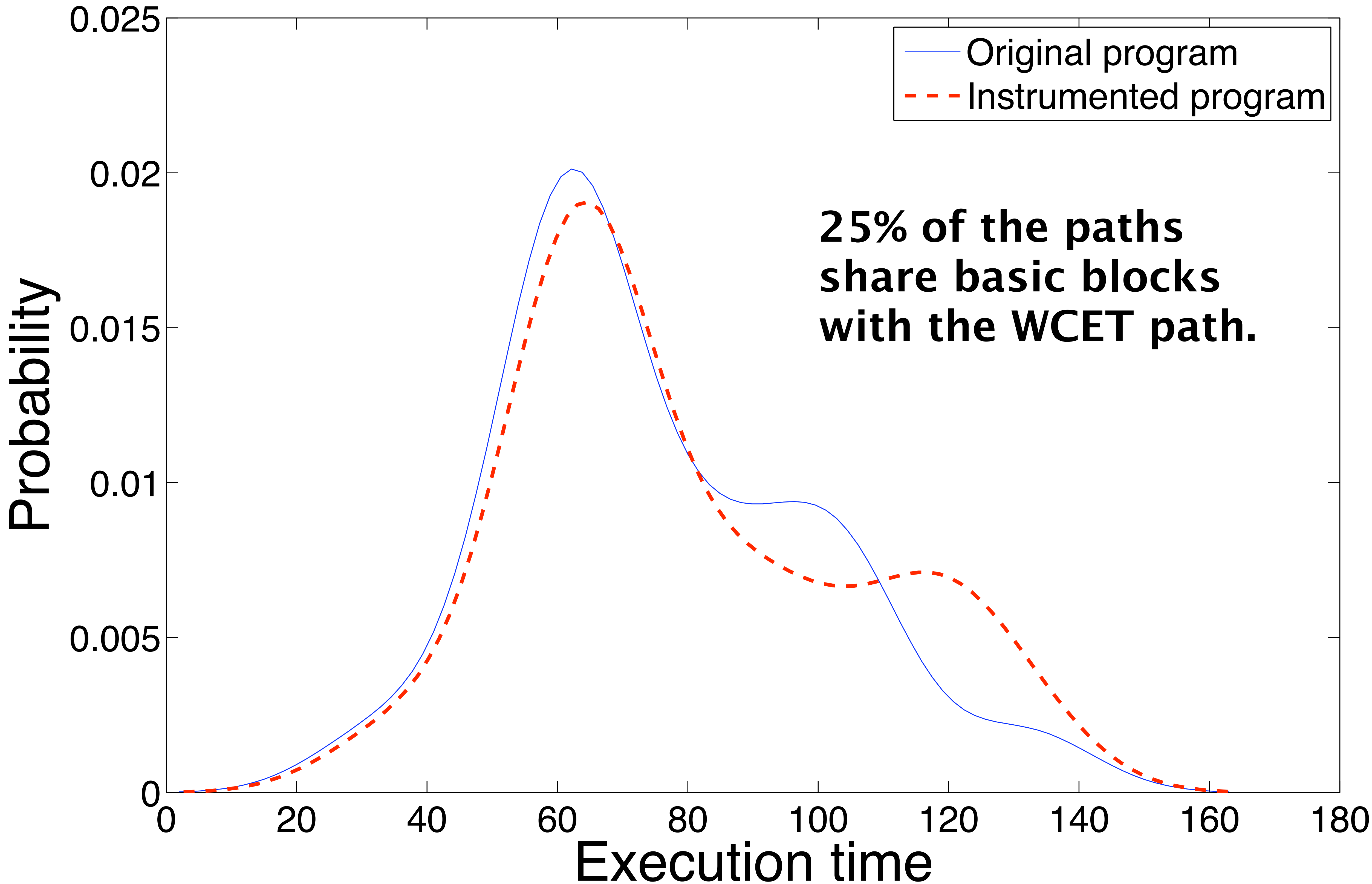
Case Study: OLPC

- Open source keyboard controller

Case Study

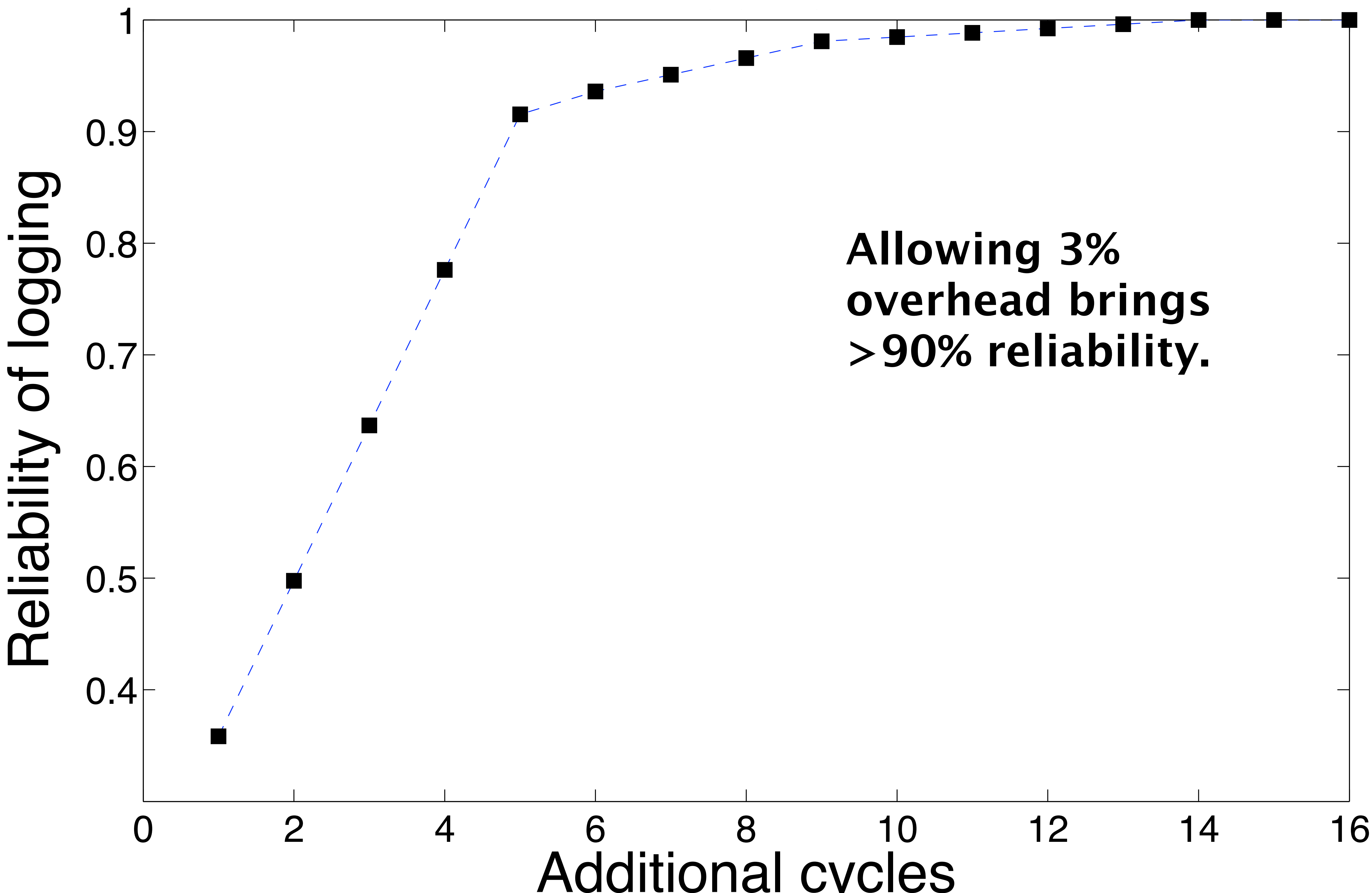
- Function `handle_power()`
  - 42 basic blocks
  - 20 different control flows
  - mean execution time is 75 cycles
  - worst-case execution time is 132 cycles
- Built source analysis tool in OCaml
- Use ILP library in Matlab

Execution Time

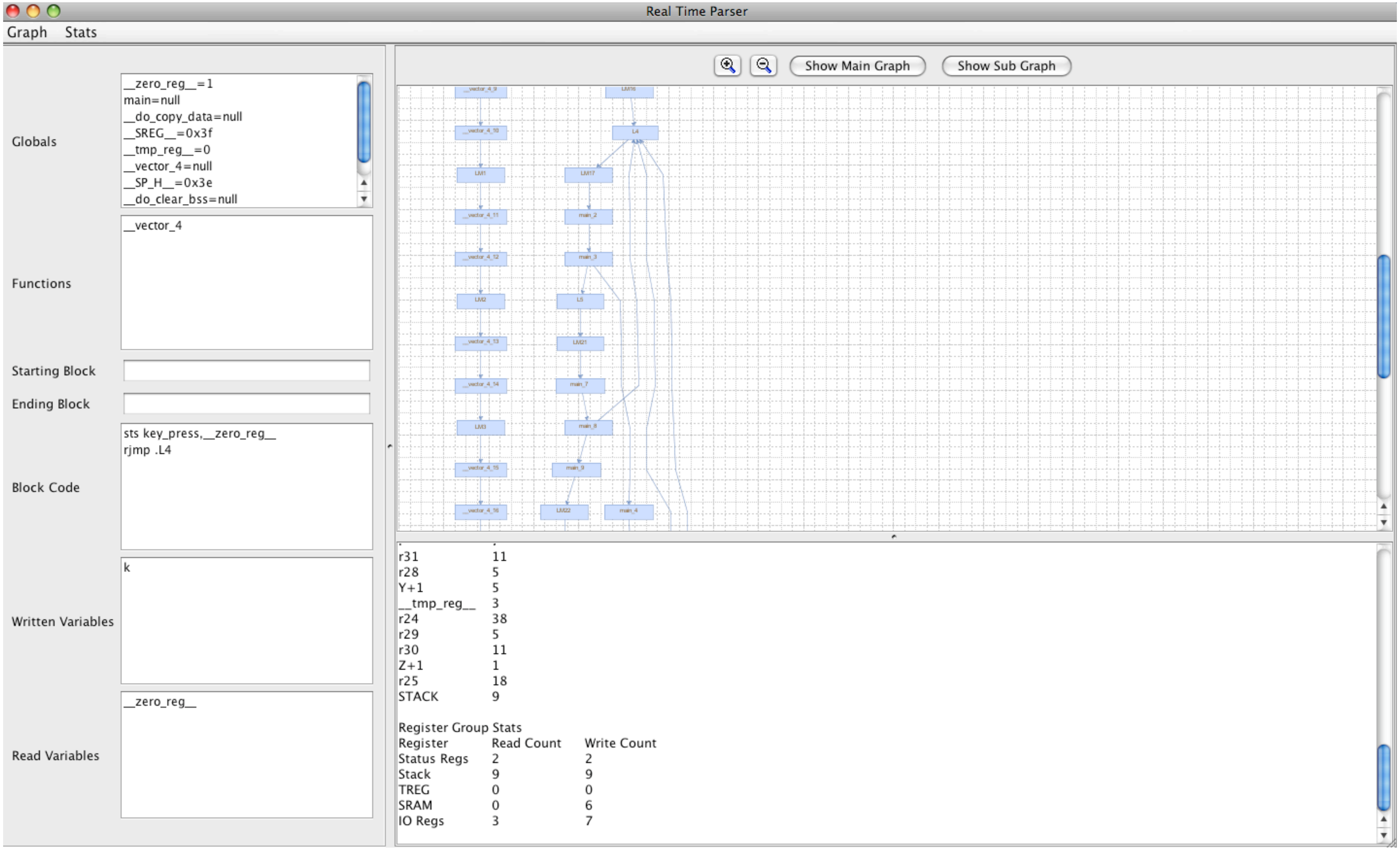


25% of the paths share basic blocks with the WCET path.

Increasing the Time Budget



Tool (gen. two)



gcc => assembly => analyze =>
instrument => compile => deploy

Future Work

- Extend to multiprogramming environments
- Refine insertion algorithm wrt interaction between minimization and reliability
- Open source Eclipse plugin

Conclusions

- Debugging is a real problem
- Tracing is a common, so far ad hoc, solution
- First steps towards automated trace generation for real-time programs
  - Optimization problem to maximize trace value.
  - Reduction for minimizing insertion points.
  - Equations for calculating the buffer size.

Acknowledgments

Thanks to:

- *Frieder Ferlemann* from the OLPC keyboard controller project
- *Aswinkumar Rajendiran & Ruchi Varshney* for the second generation software tool
- *Funding agencies* (NSERC, ORF) for providing the resources to make this research happen and picking up the cheques