Accuracy-Aware Program Transformations

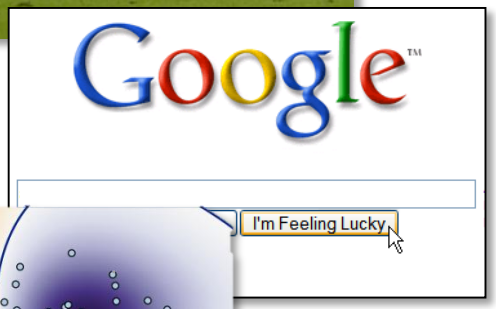
Sasa Misailovic

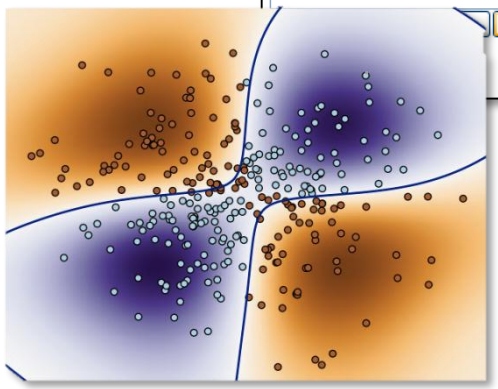
MIT CSAIL

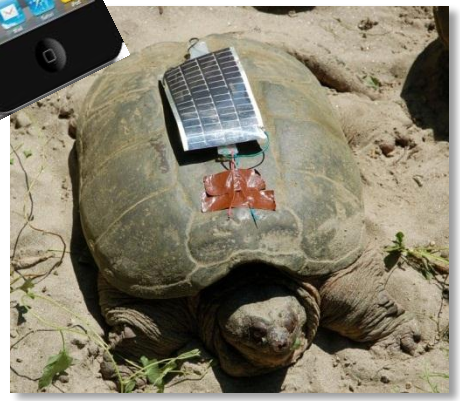
Collaborators

Martin Rinard, Michael Carbin,
Stelios Sidiroglou, Henry Hoffmann,
Deokhwan Kim, Daniel Roy,
Zeyuan Allen Zhu, Michael Kling,
Jonathan Kelner, Anant Agarwal

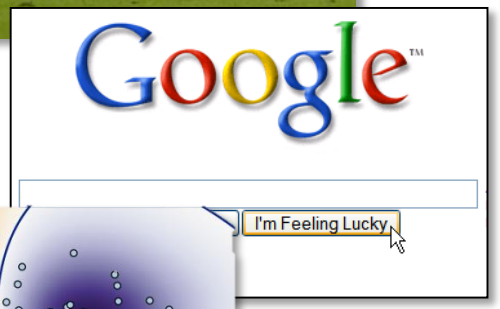
Emerging Software and Hardware

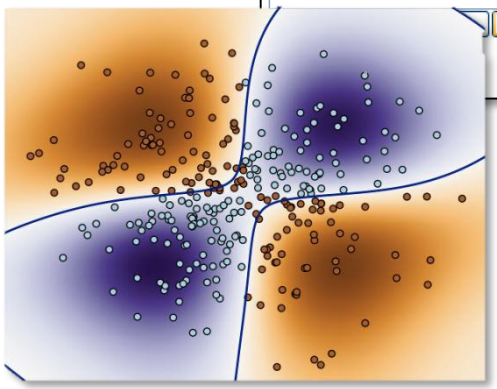


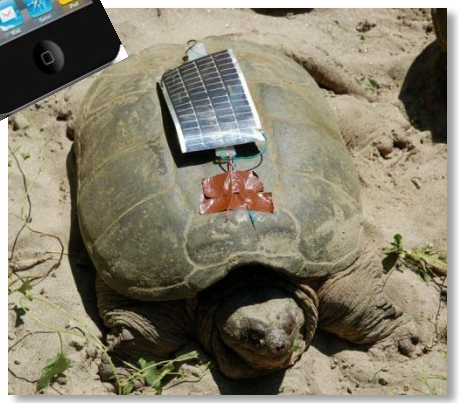




Emerging Software and Hardware

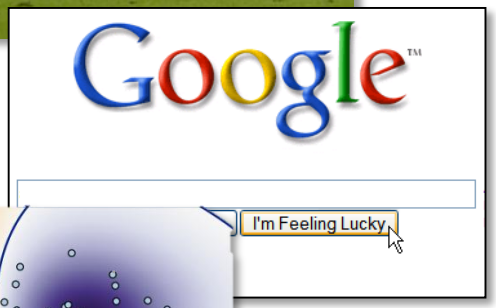


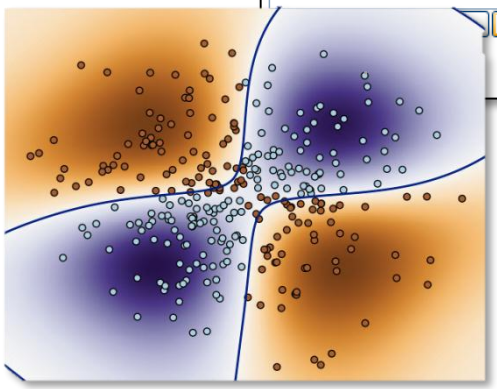




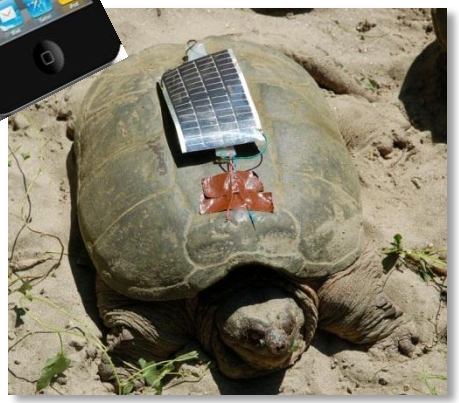
Big Data; Approximate

Emerging Software and Hardware



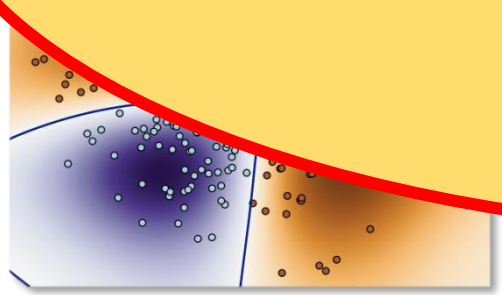


Big Data; Approximate



Energy Conscious

Emerging Software and Hardware

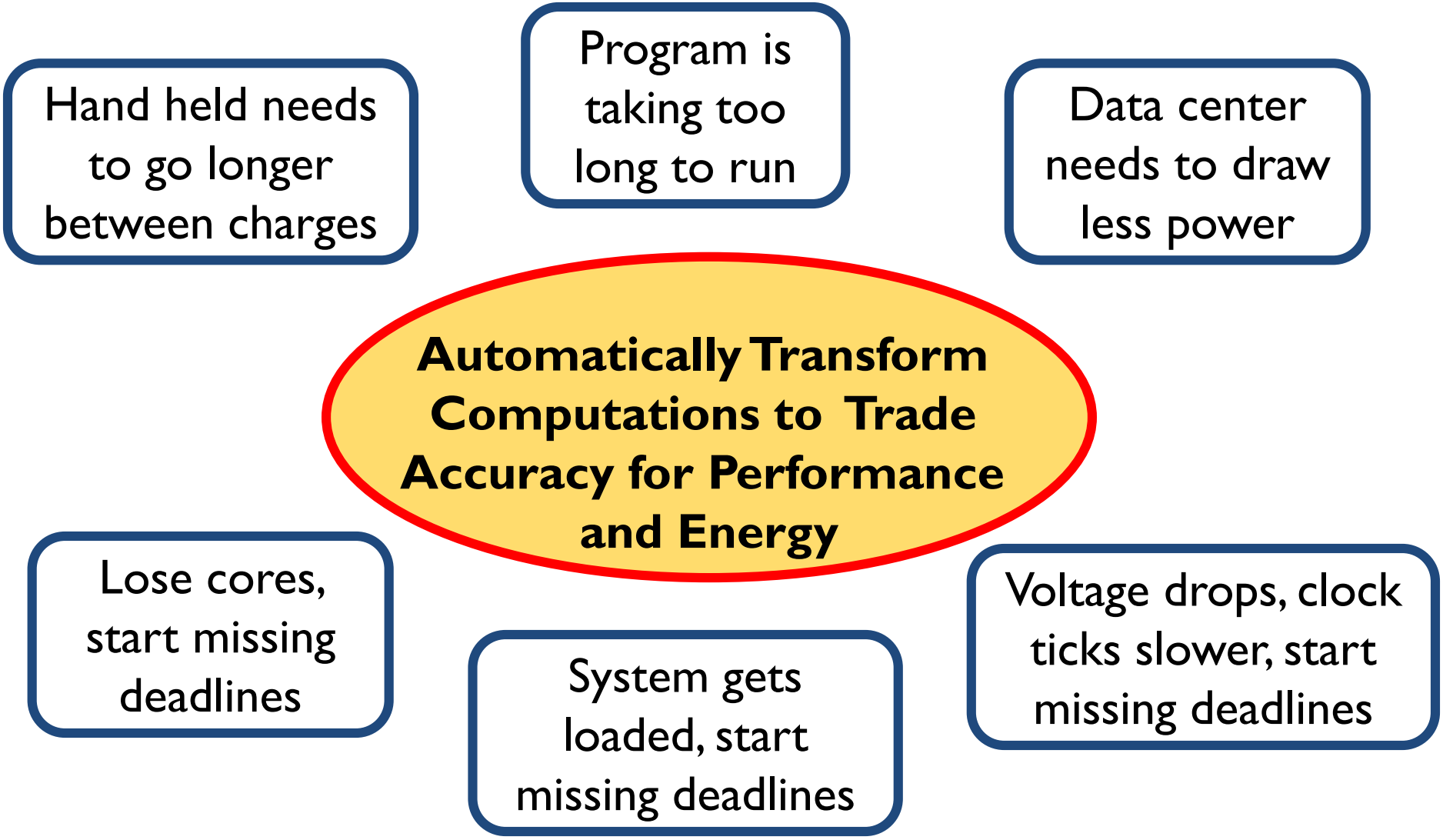


Big Data; Approximate

Energy Conscious

Automatically Transform Computations to Trade Accuracy for Performance and Energy

Solving Problems with Transformations



Consider This Transformation

```
for (i = 0; i < n; i++) { … }
```

```
for (i = 0; i < n; i += 2) { … }
```

Loop Perforation

```
for (i = 0; i < n; i++) { … }
```

```
for (i = 0; i < n; i += 2) { … }
```

Effects:

- ✓ Should improve performance
- ✓ Broadly applicable

Loop Perforation

```
for (i = 0; i < n; i++) { ... }
```

```
for (i = 0; i < n; i += 2) { ... }
```

Common Reaction:

But it changes the program semantics!
The result will be wrong ?!

Loop Perforation

```
for (i = 0; i < n; i++) { ... }
```

```
for (i = 0; i < n; i += 2) { ... }
```

Common Reaction:

But it changes the program semantics!

~~**The result will be wrong ?!**~~

The result can be less accurate!

Acceptability =

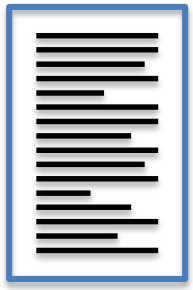
Accuracy + Integrity

Acceptability = Accuracy + Integrity

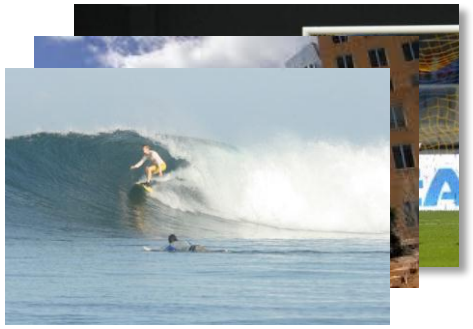
Optimization problem:
minimize execution time given constraints on accuracy and integrity of the computation

Optimization Inputs

Original Program



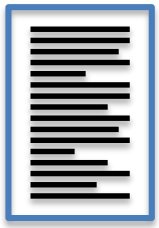
Input & Accuracy Specification

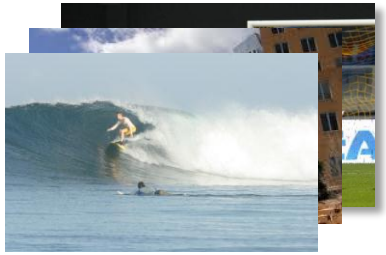


Program Transformation

```
for (i = 0; i < n; i++) { ... }
```

```
for (i = 0; i < n; i += 2) { ... }
```



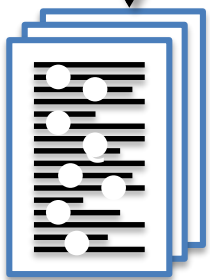


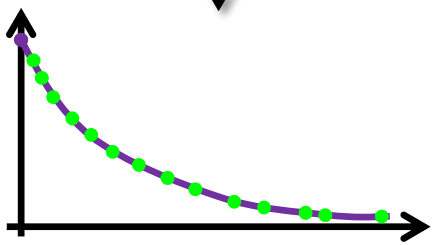
for (i = 0; i < n; i++) { … }

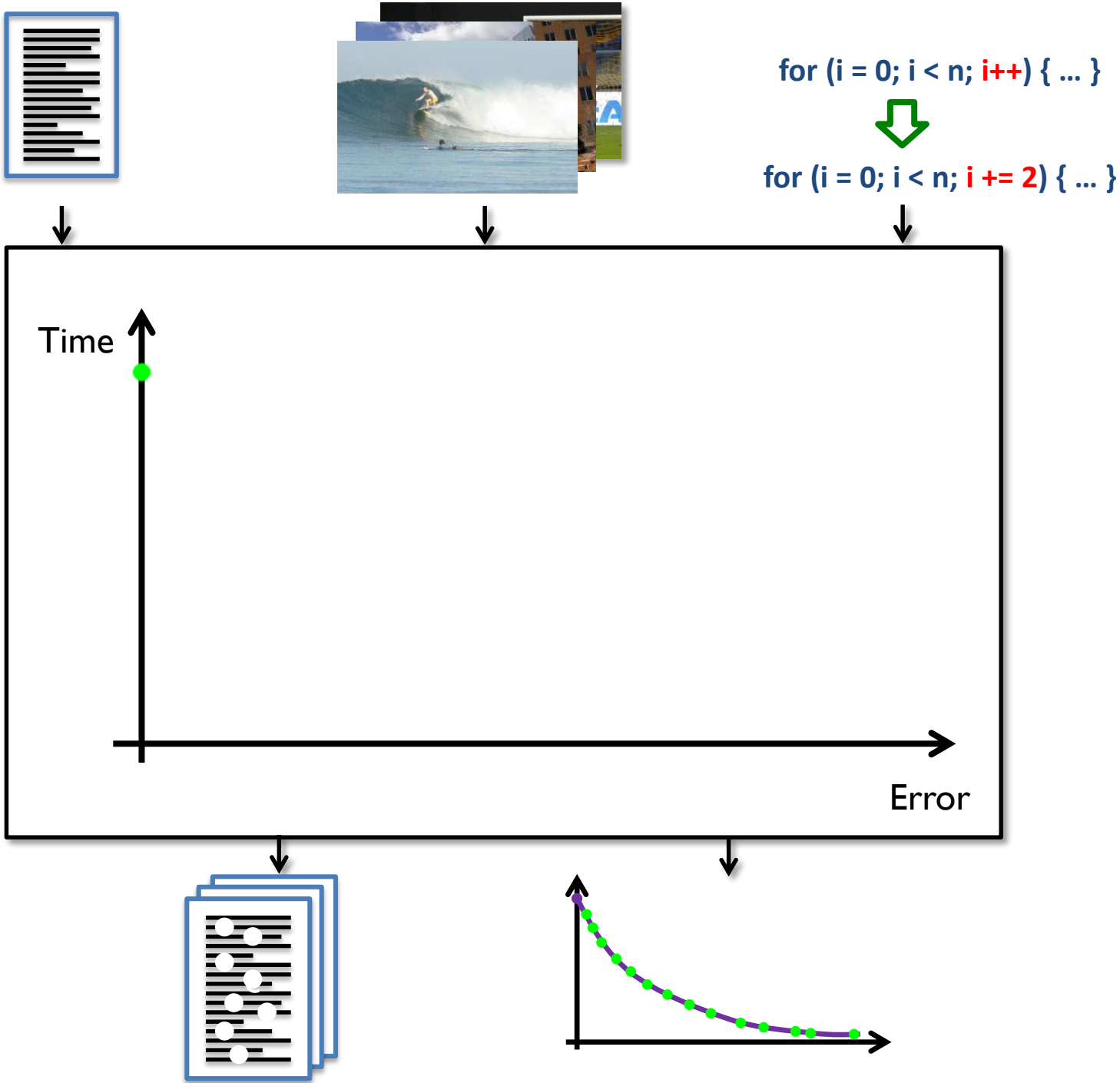
for (i = 0; i < n; i += 2) { … }

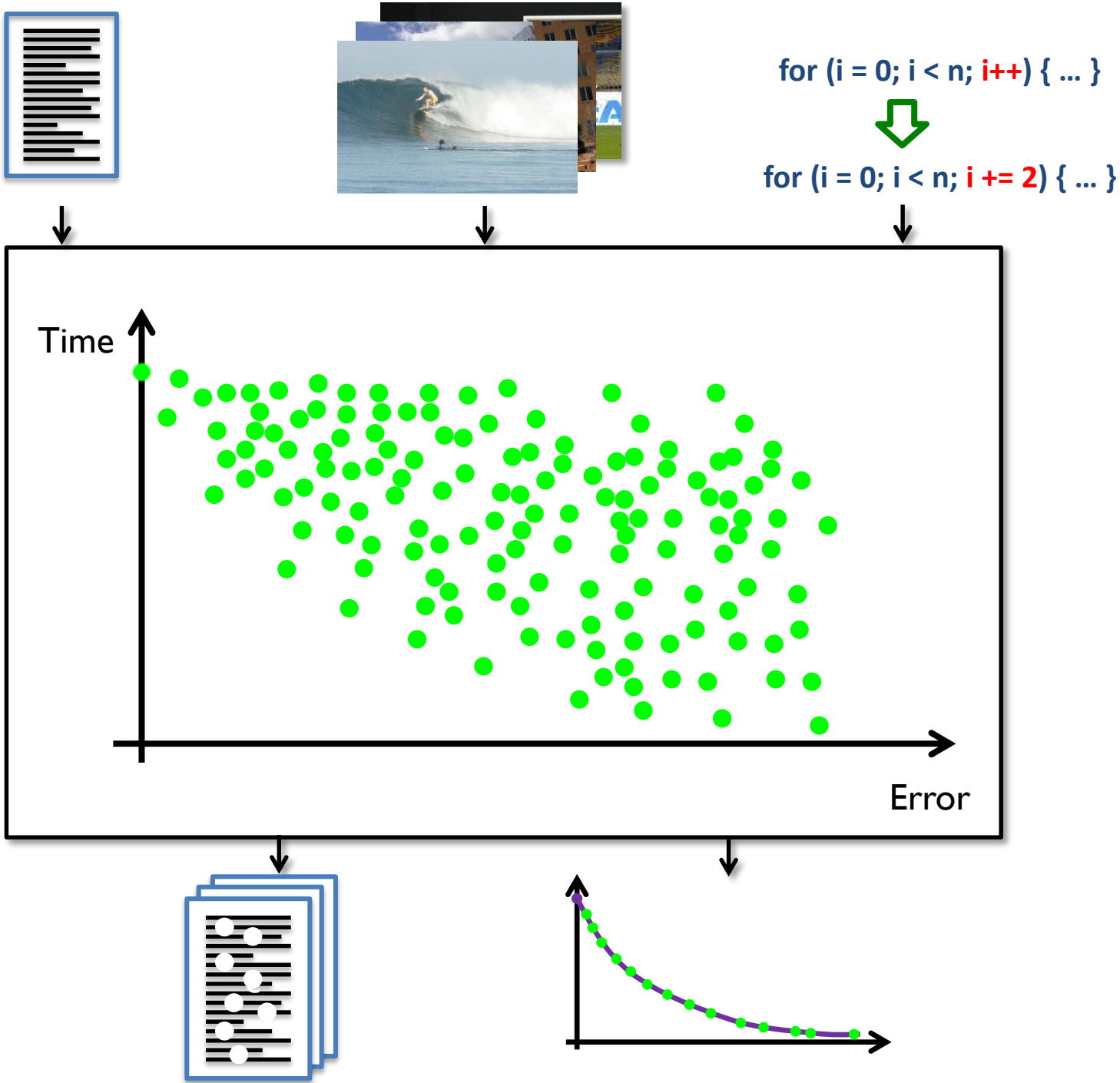
Optimization Framework

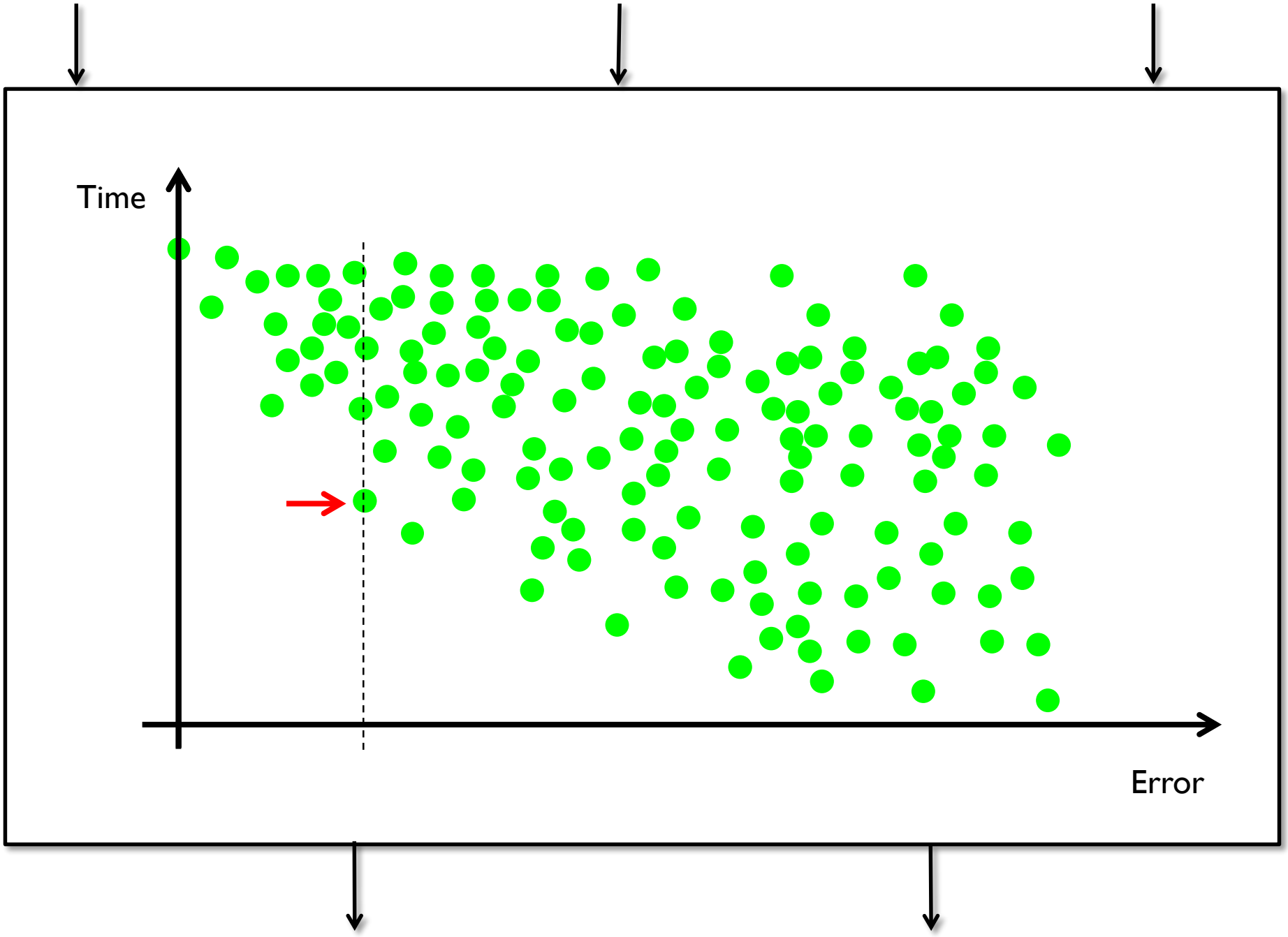
- Find Candidates for Transformation
- Analyze Effects of the Transformations
- Navigate Tradeoff Space

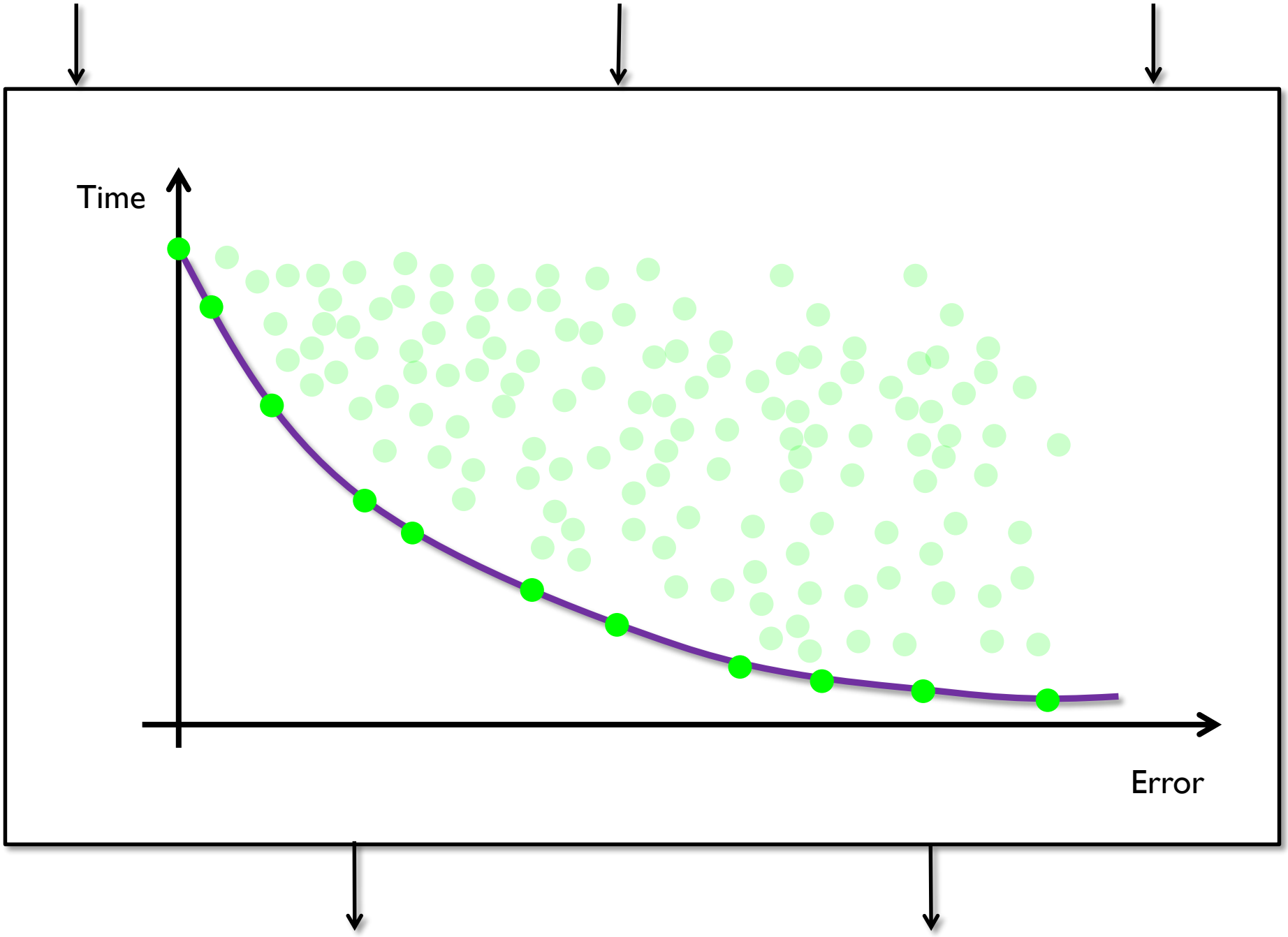


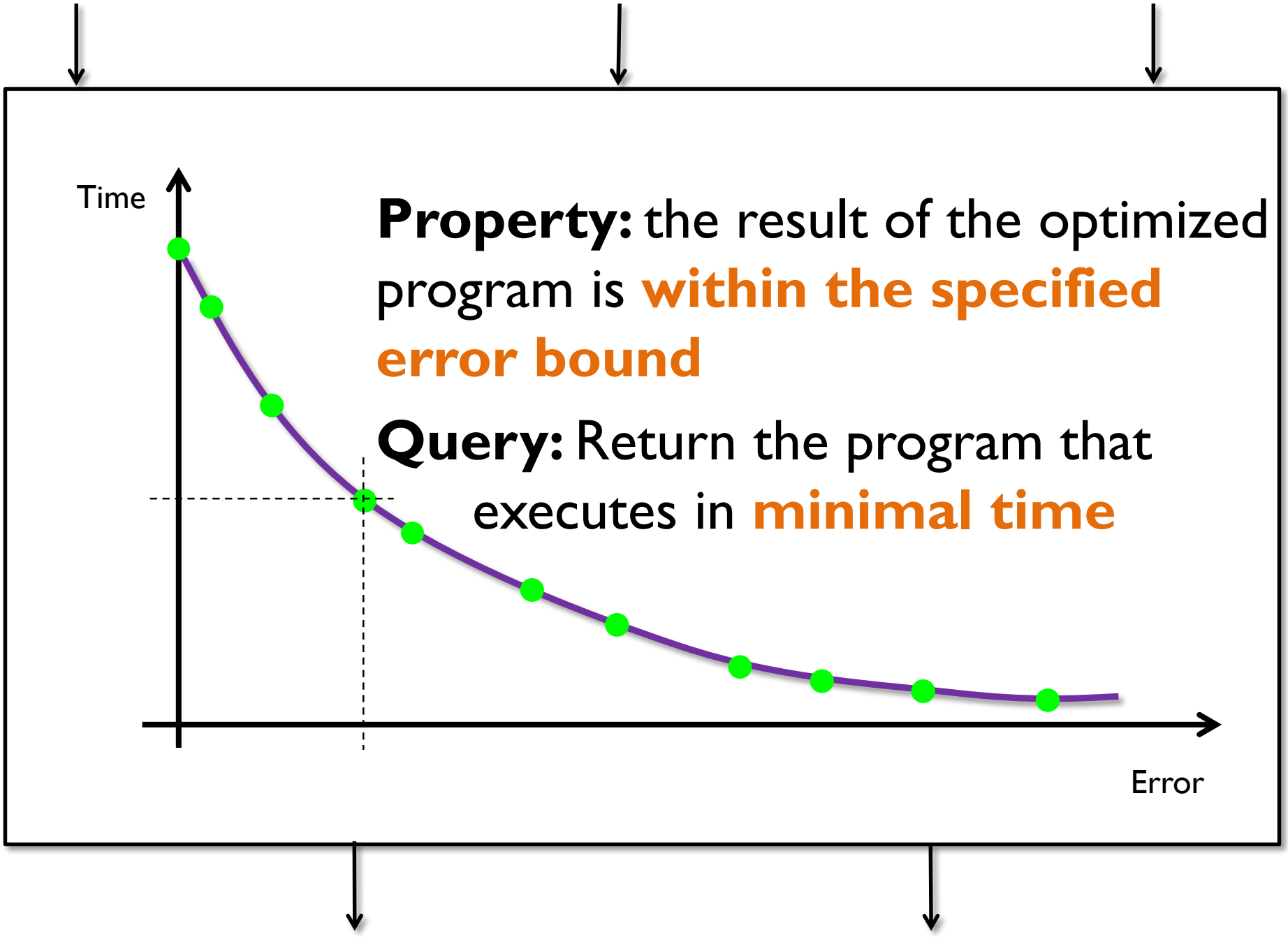



for (i = 0; i < n; i++) { … }
for (i = 0; i < n; i += 2) { … }
Time
Error


for (i = 0; i < n; i++) { … }
for (i = 0; i < n; i += 2) { … }
Time
Error


Time
Error


Time
Error


Time
Property: the result of the optimized program is within the specified error bound
Query: Return the program that executes in minimal time
Error

Explicit Search Algorithm for Perforation

Find Transformation Candidates:

- Profile program to find time-consuming for loops

Analyze the Effects of Perforation:

- Integrity: memory safety, well formed output
- Performance: Compare execution times
- Accuracy: Compare the quality of the results

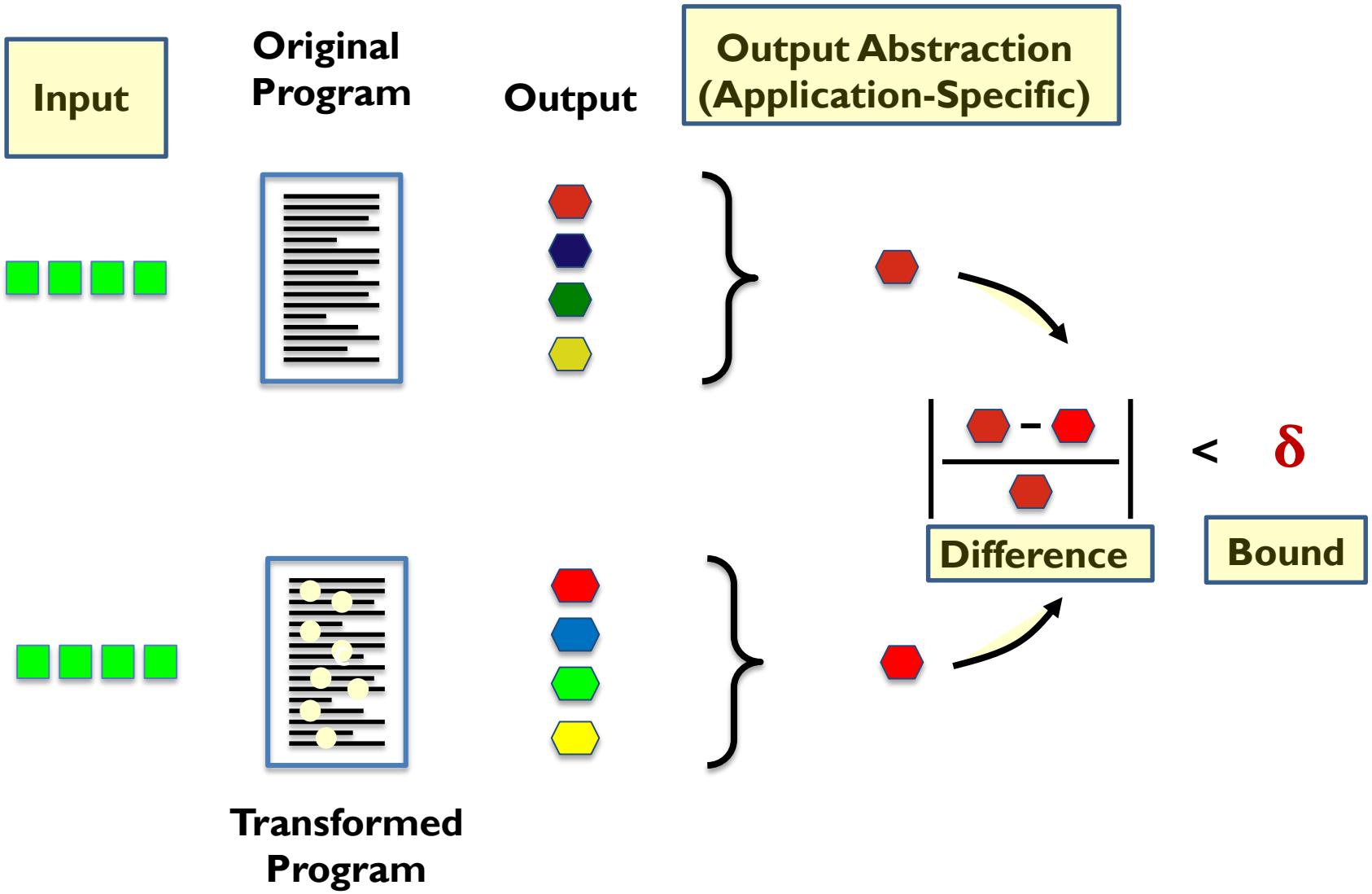
Navigate Tradeoff Space:

- Combine multiple perforatable loops

  Prioritize loops by their individual performance and accuracy

  Greedy or Exhaustive Search with Pruning

Accuracy Analysis of Computation

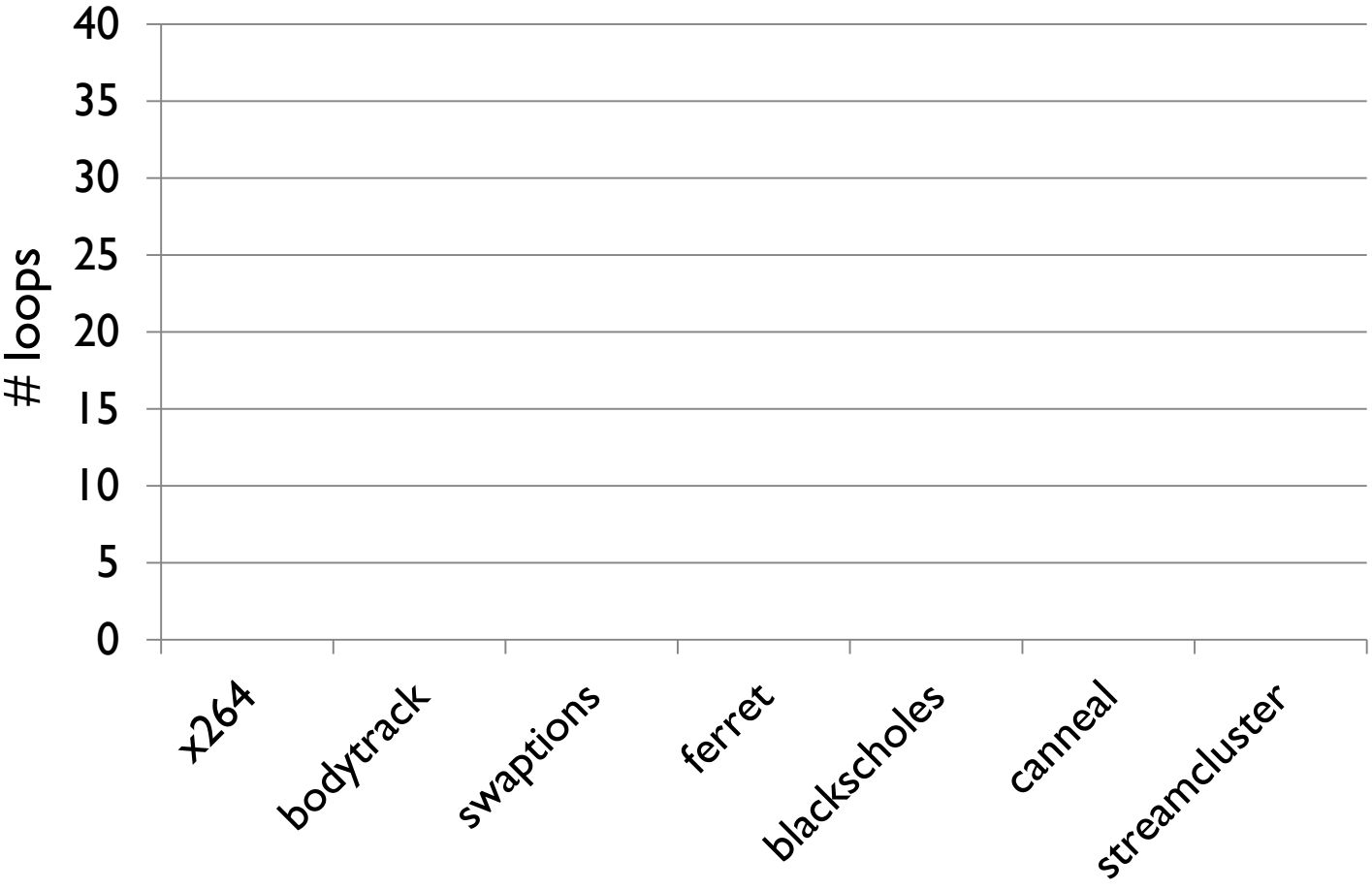


Analysis for Individual Loop Perforation

1. Perforate one time-consuming loop at a time
2. Execute perforated program
3. Filter out **critical loops**:
   a) Program crashes
   b) Accuracy loss > δ_{max}
   c) Execution slows down
   d) Latent memory errors (Valgrind)
4. Repeat 1-3 for all loops, inputs, perforation rates

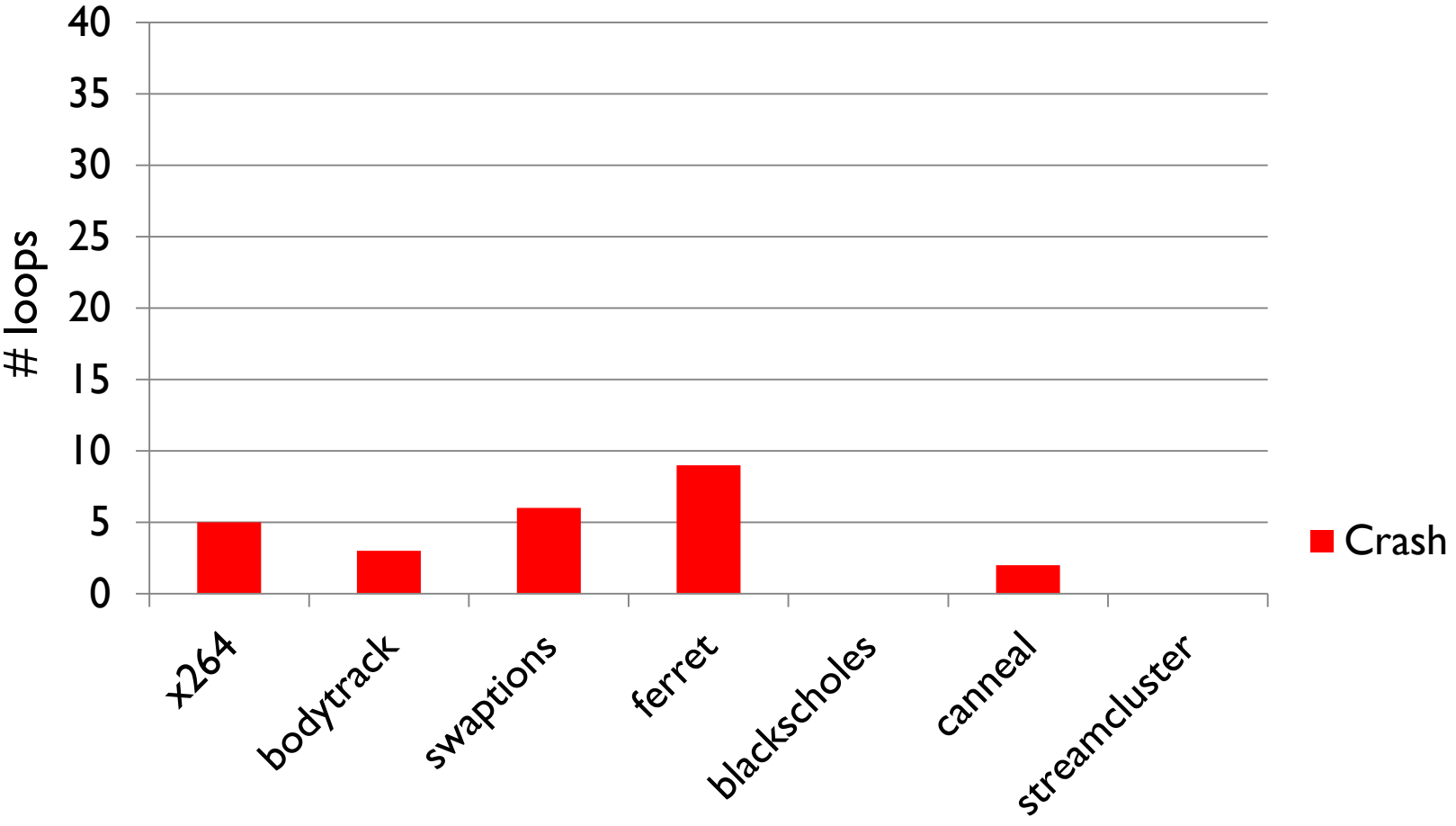
From [ICSE 2010]

Individual Loop Perforation Results



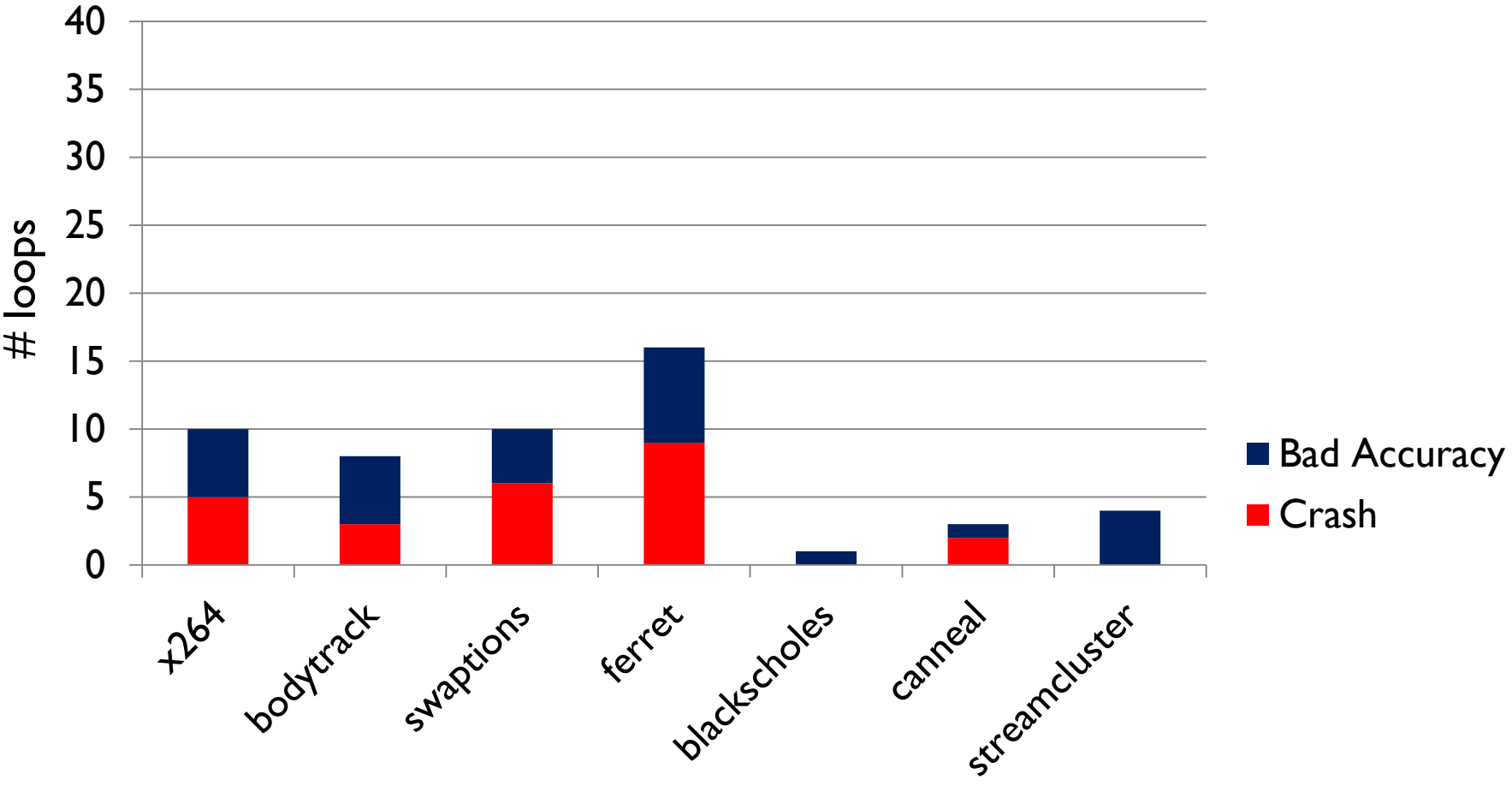
From [ICSE 2010]

Individual Loop Perforation Results



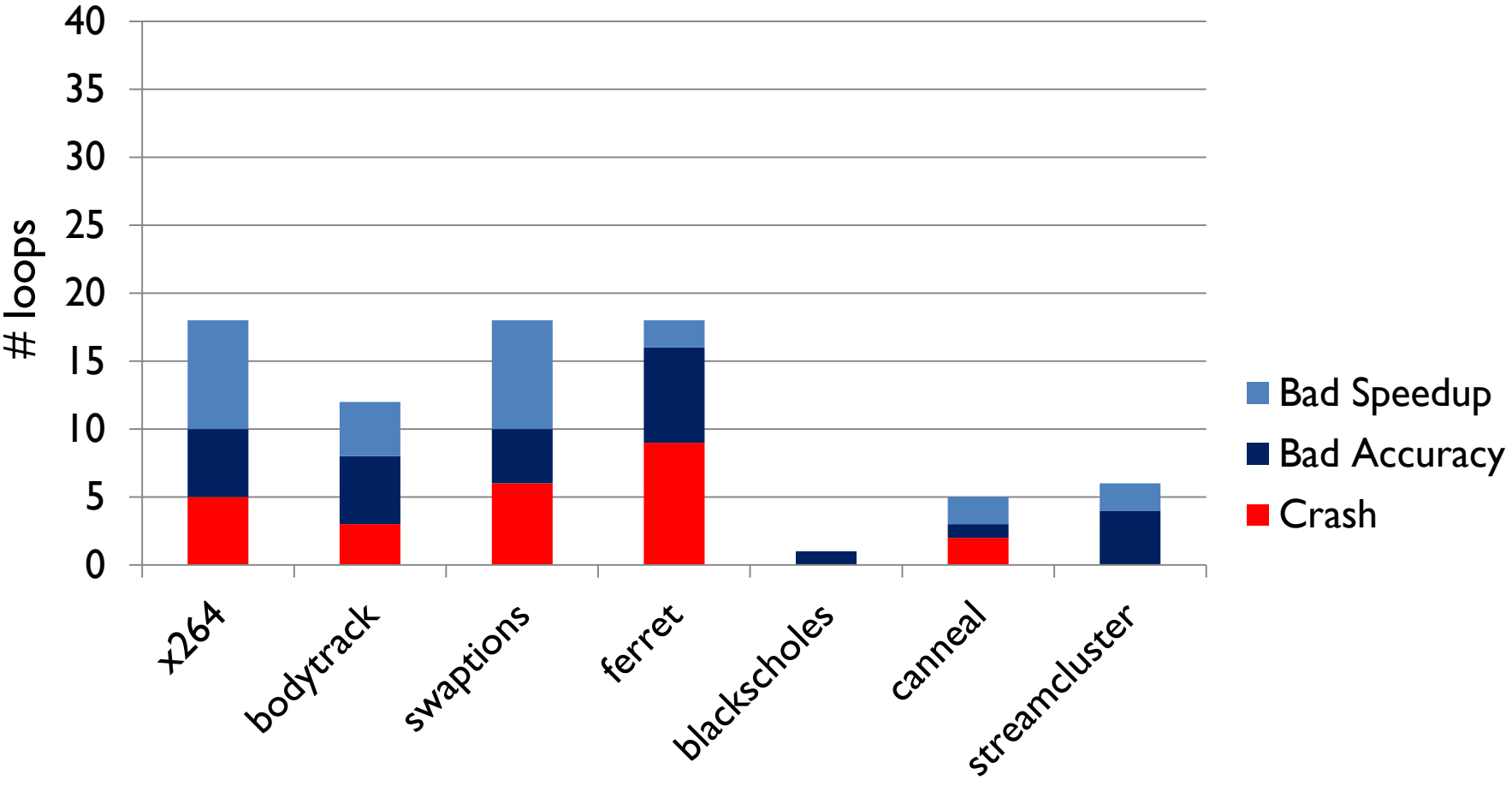
From [ICSE 2010]

Individual Loop Perforation Results



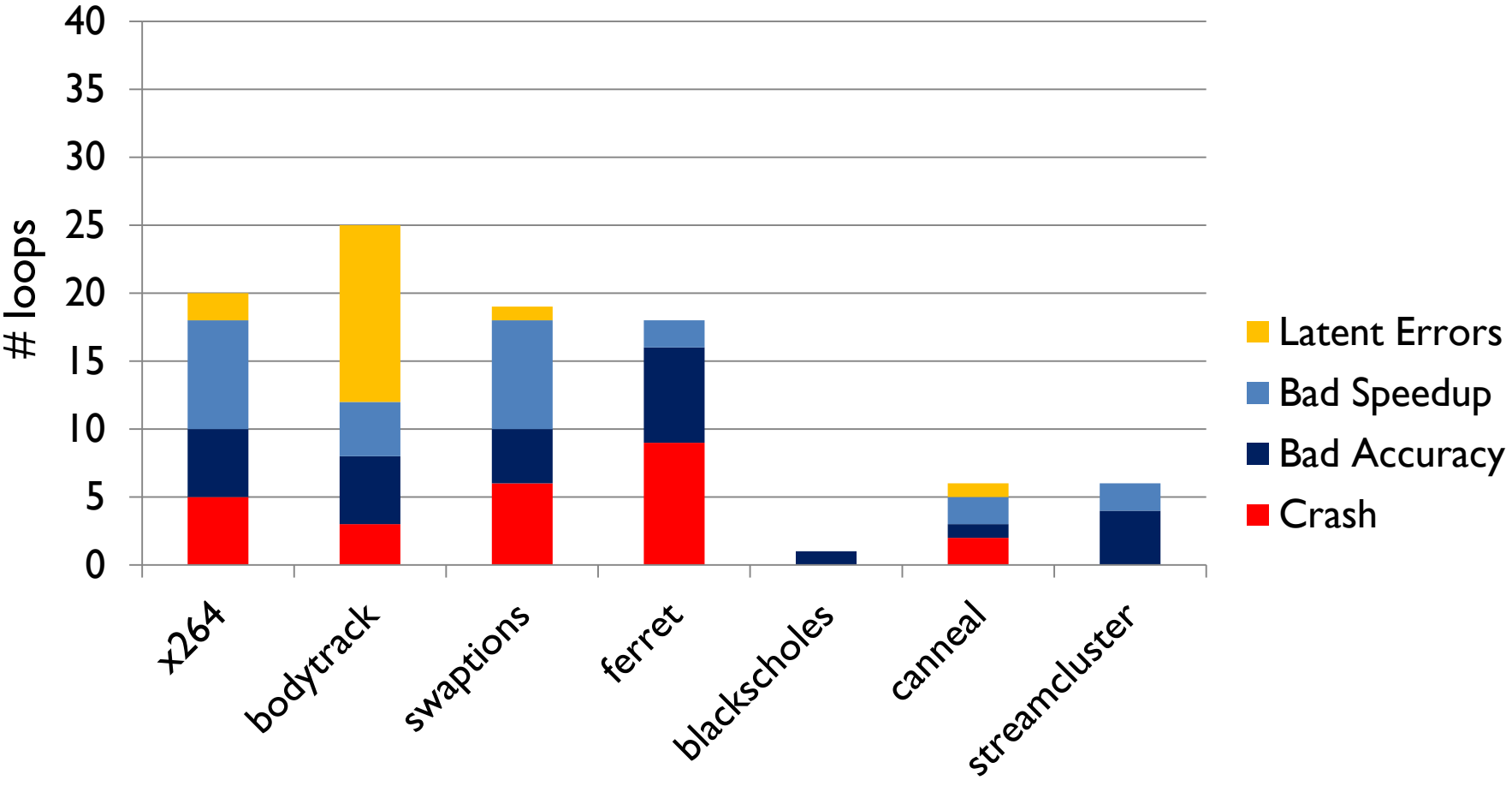
From [ICSE 2010]

Individual Loop Perforation Results



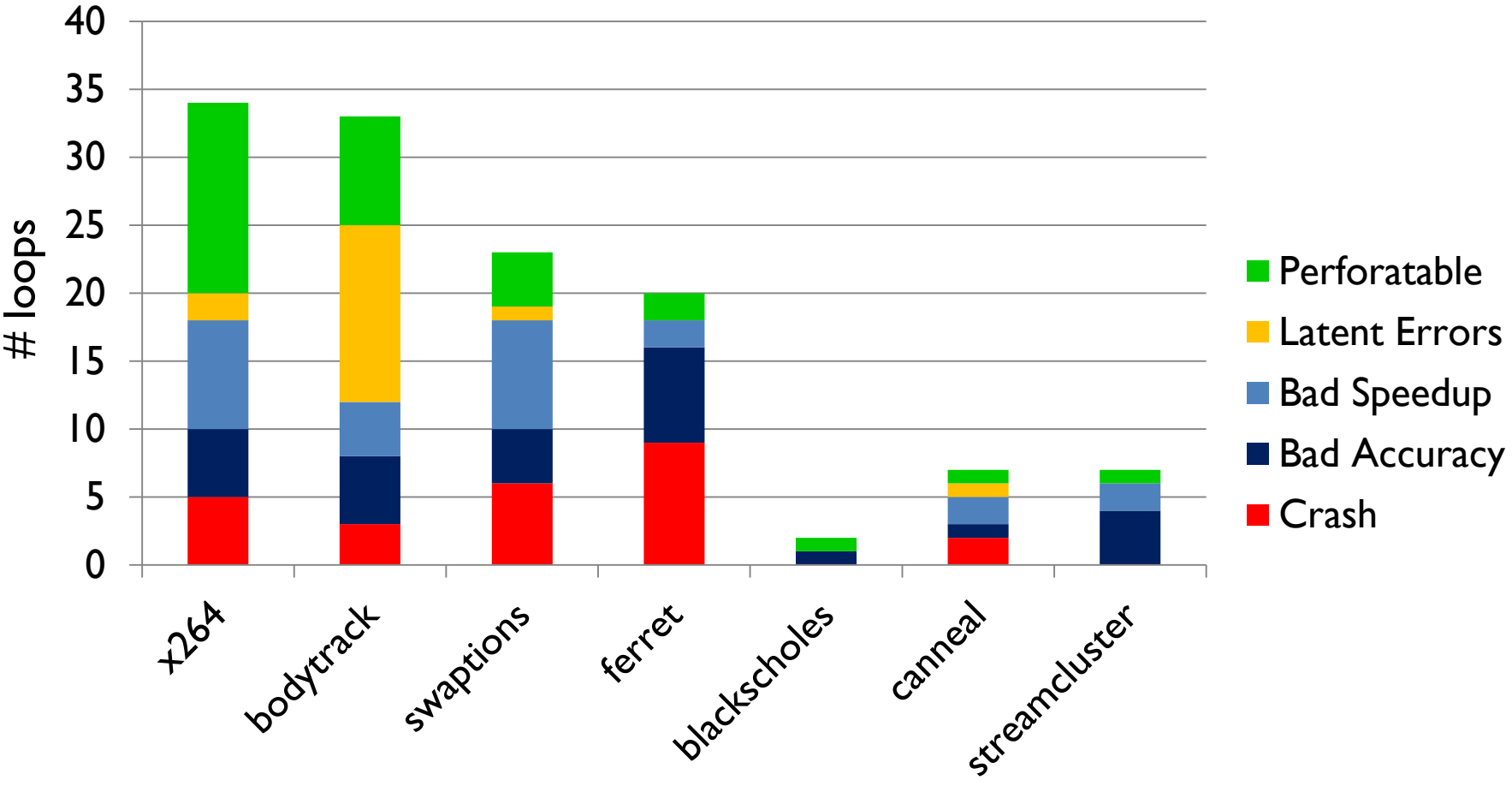
From [ICSE 2010]

Individual Loop Perforation Results

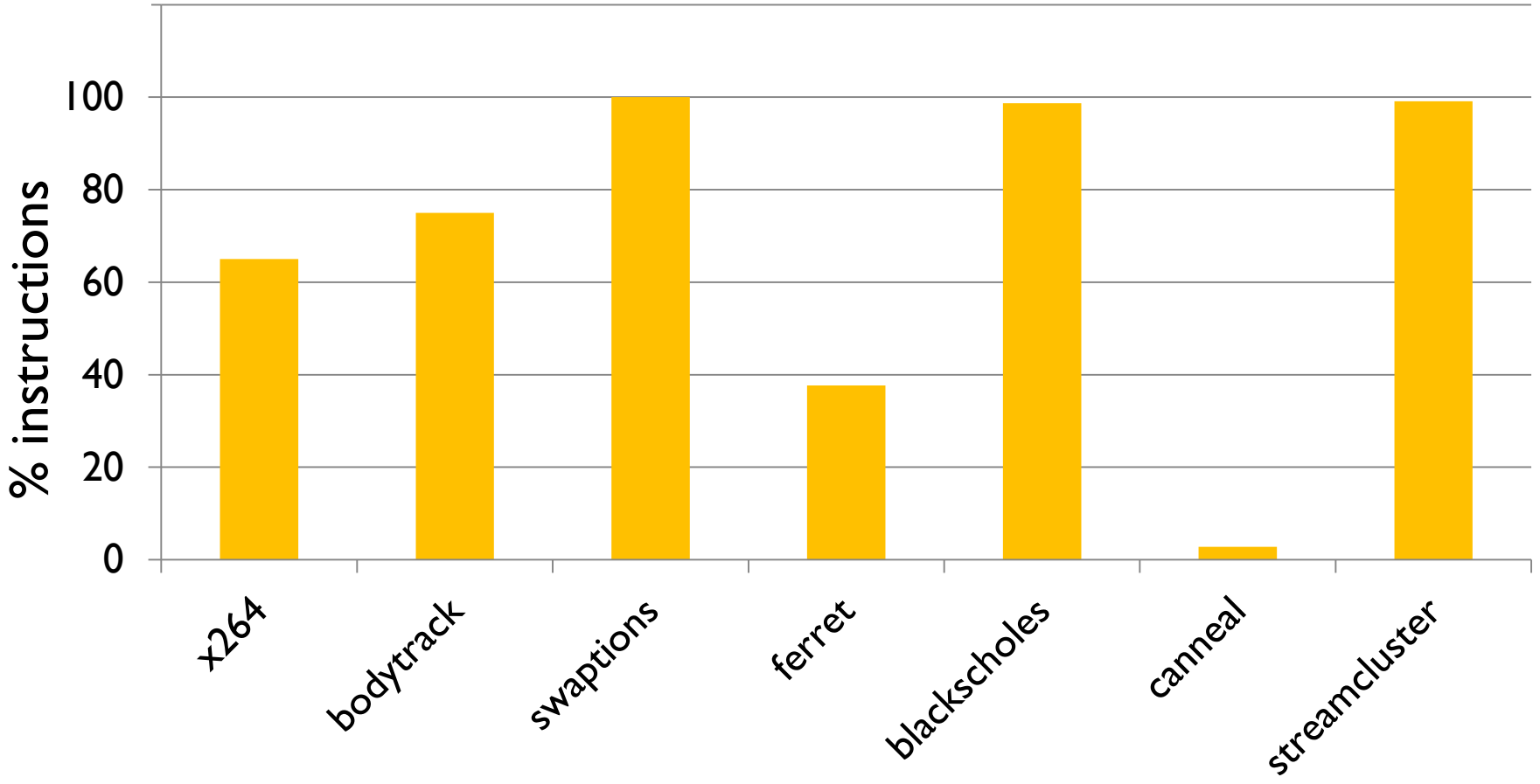


From [ICSE 2010]

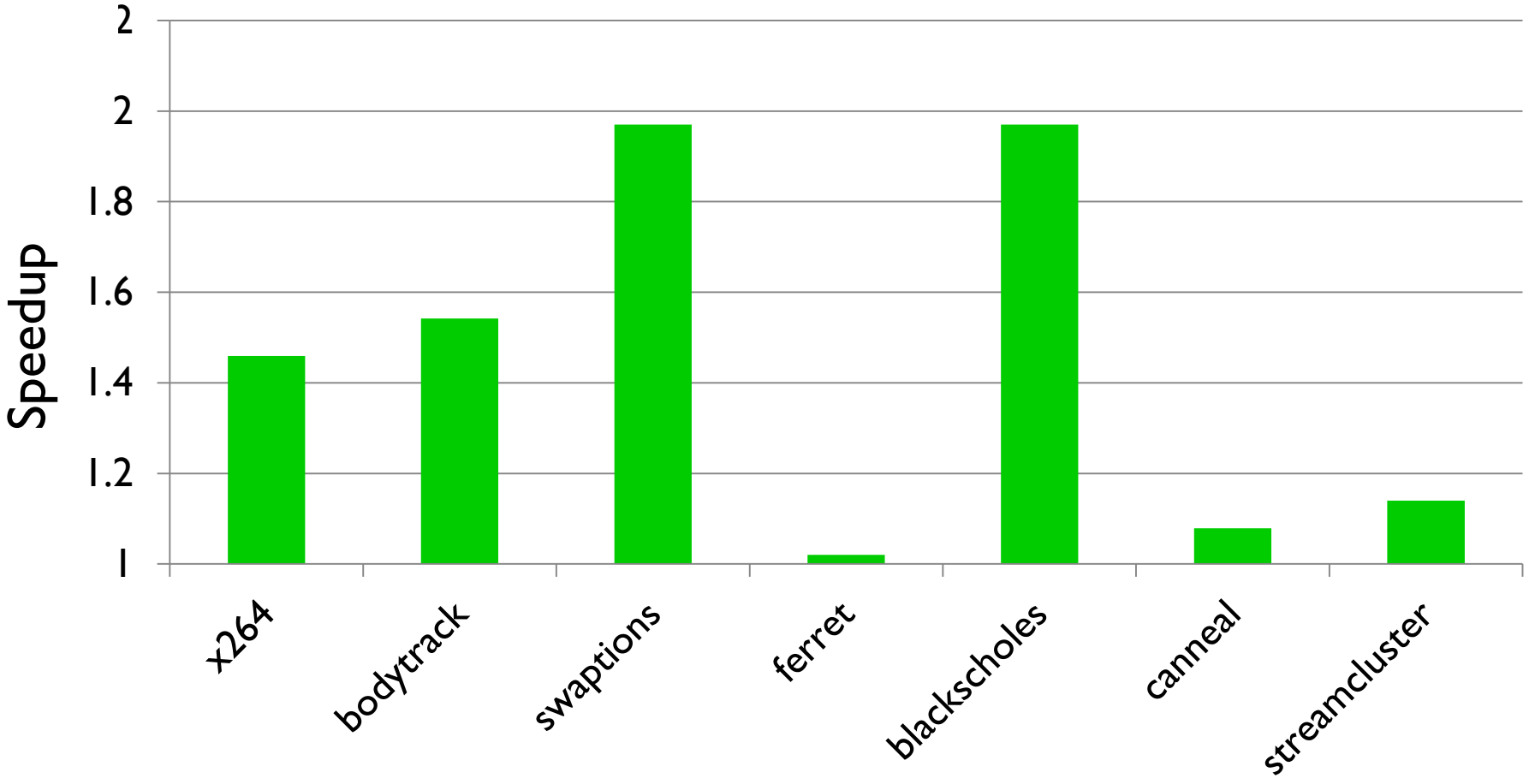
Individual Loop Perforation Results



Percentage of Work Done in Perforatable Loops



Performance Increase of the Top Perforatable Loop (Relative Error < 0.1)



Result Interpretation

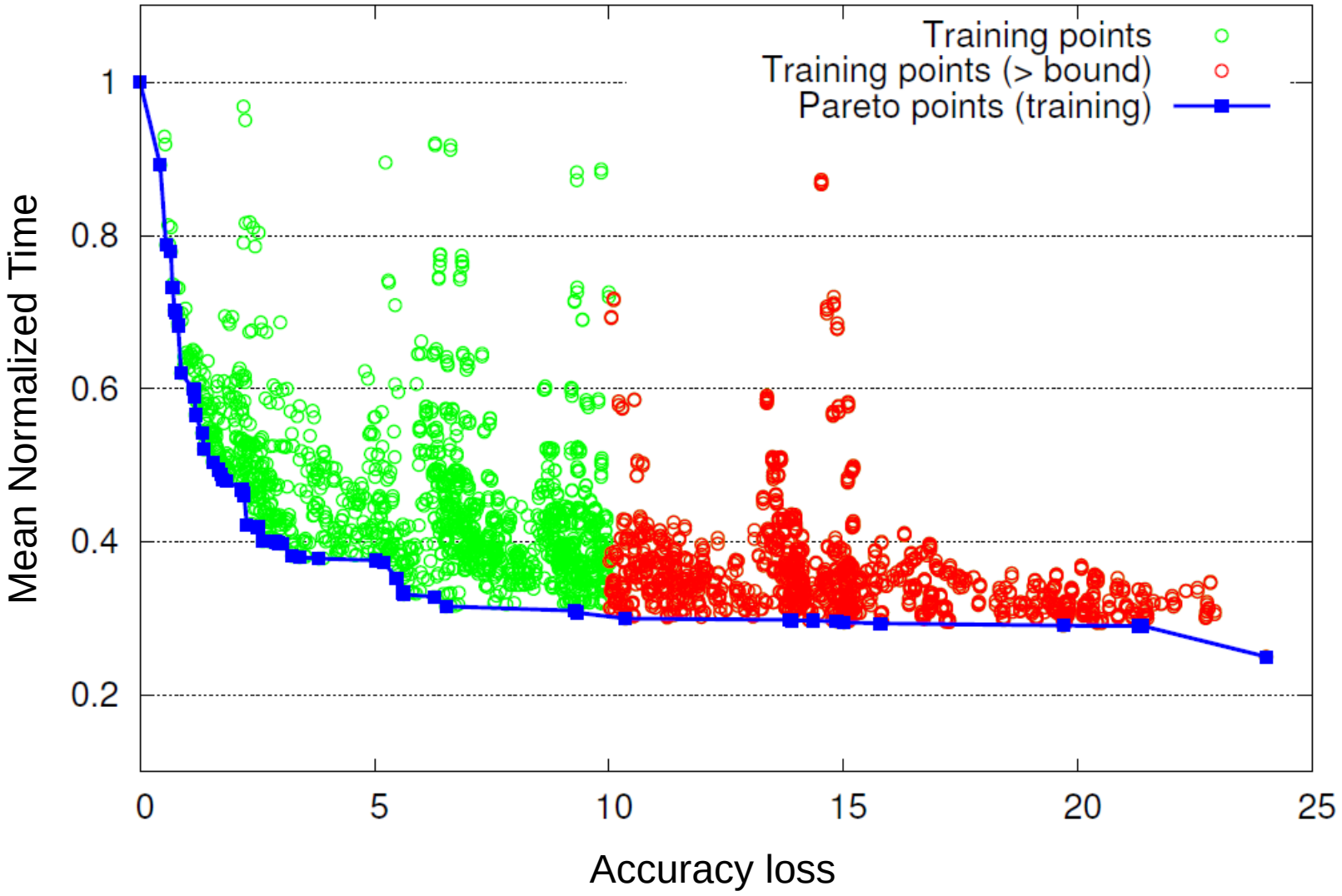
Manual inspection of perforatable computations:

- **x264:** motion estimation
- **bodytrack:** MCMC
- **swaptions:** Monte Carlo simulation
- **ferret:** similarity hashing
- **blackscholes:** redundant computation
- **canneal:** simulated annealing
- **streamcluster:** cluster center search

Common: Approximate/heuristic computations

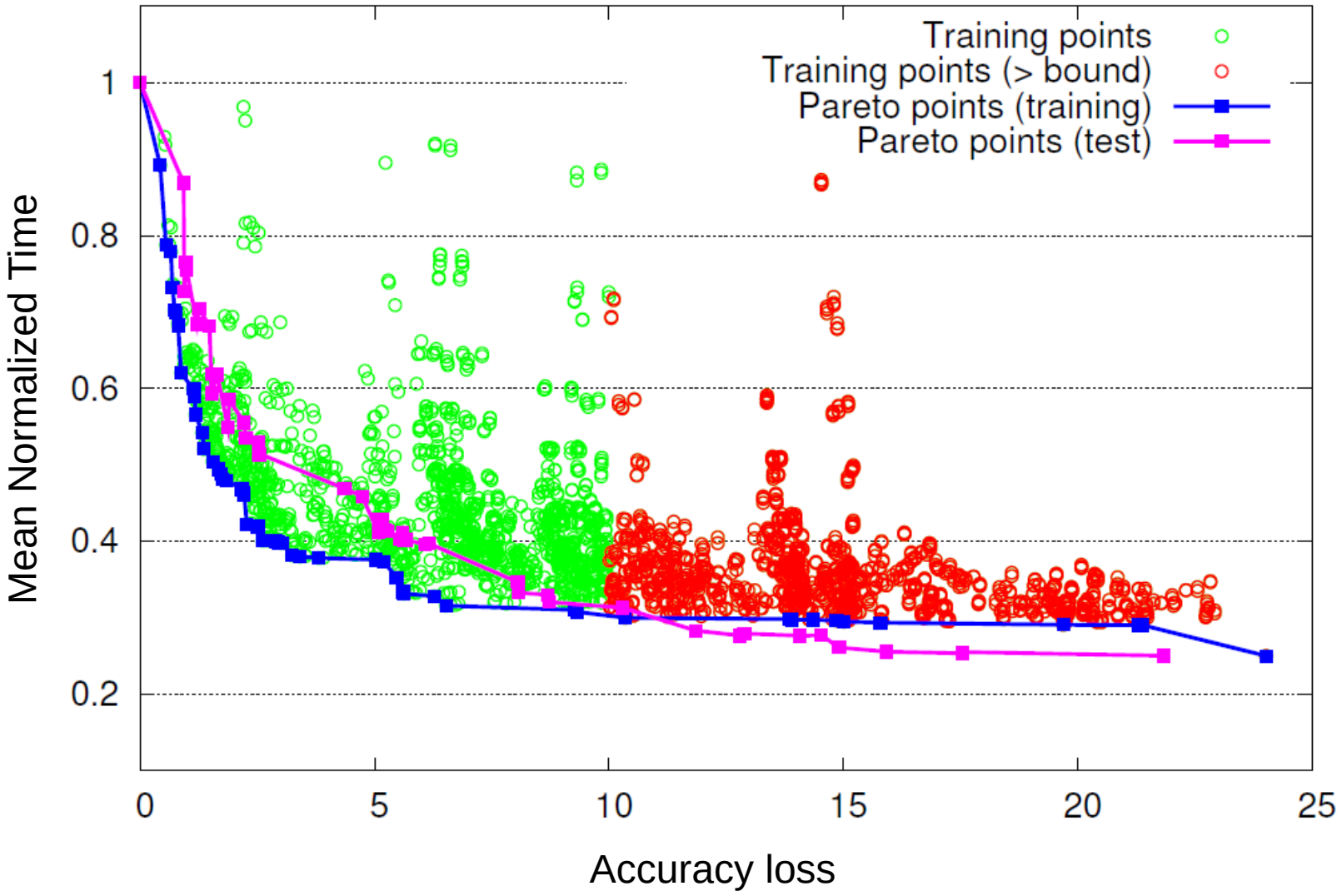
From [FSE 2011]

x264 Cumulative Loop Scores



From [FSE 2011]

x264 Cumulative Loop Scores



Status

Good:

Profitable accuracy/performance tradeoffs

Matches the approximate computations

But:

No guarantees on accuracy

No guarantees on safety

How to improve it?

How often large errors happen?

What safety guarantees can we provide?

Reasoning About Transformed Programs

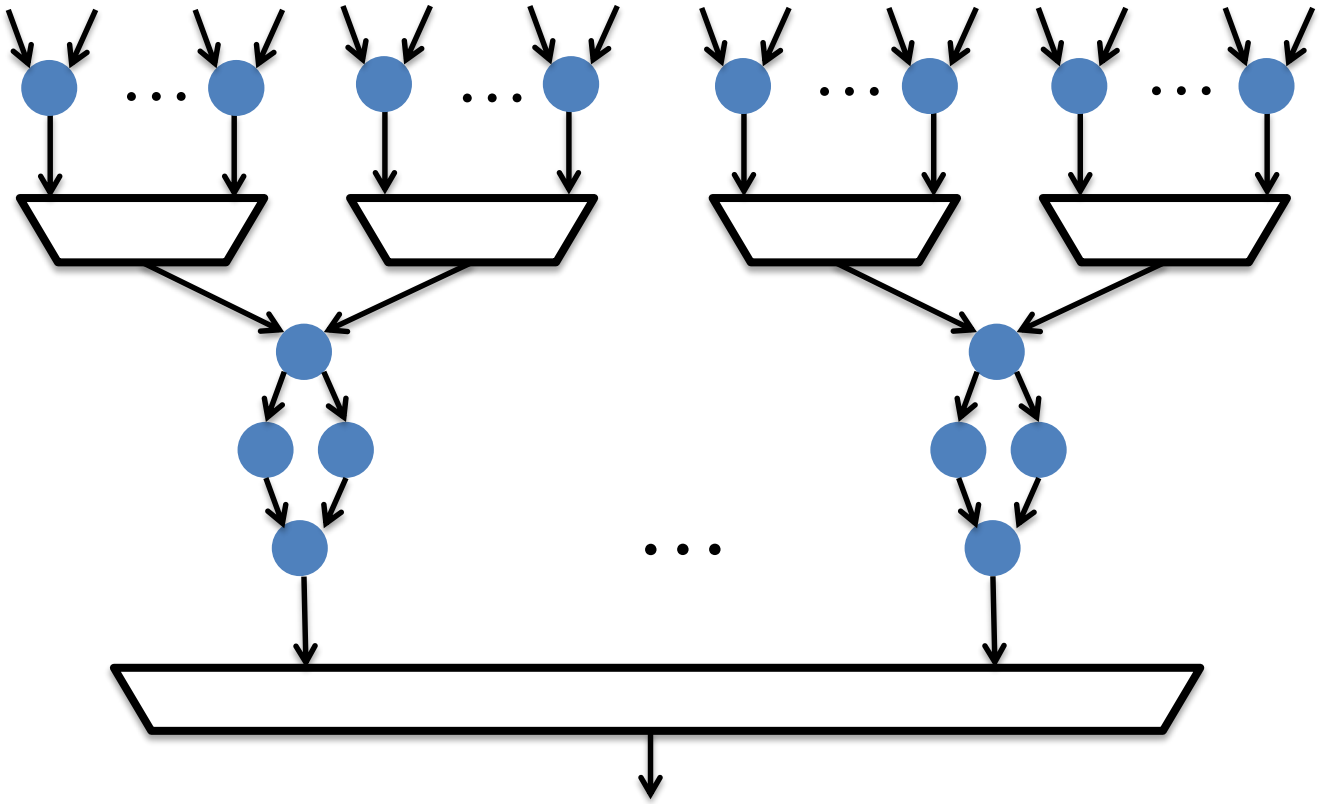
Accuracy

Probabilistic Reasoning
[SAS '11, POPL '12]
(with Z. Zhu, J. Kelner, D. Roy, M. Rinard)

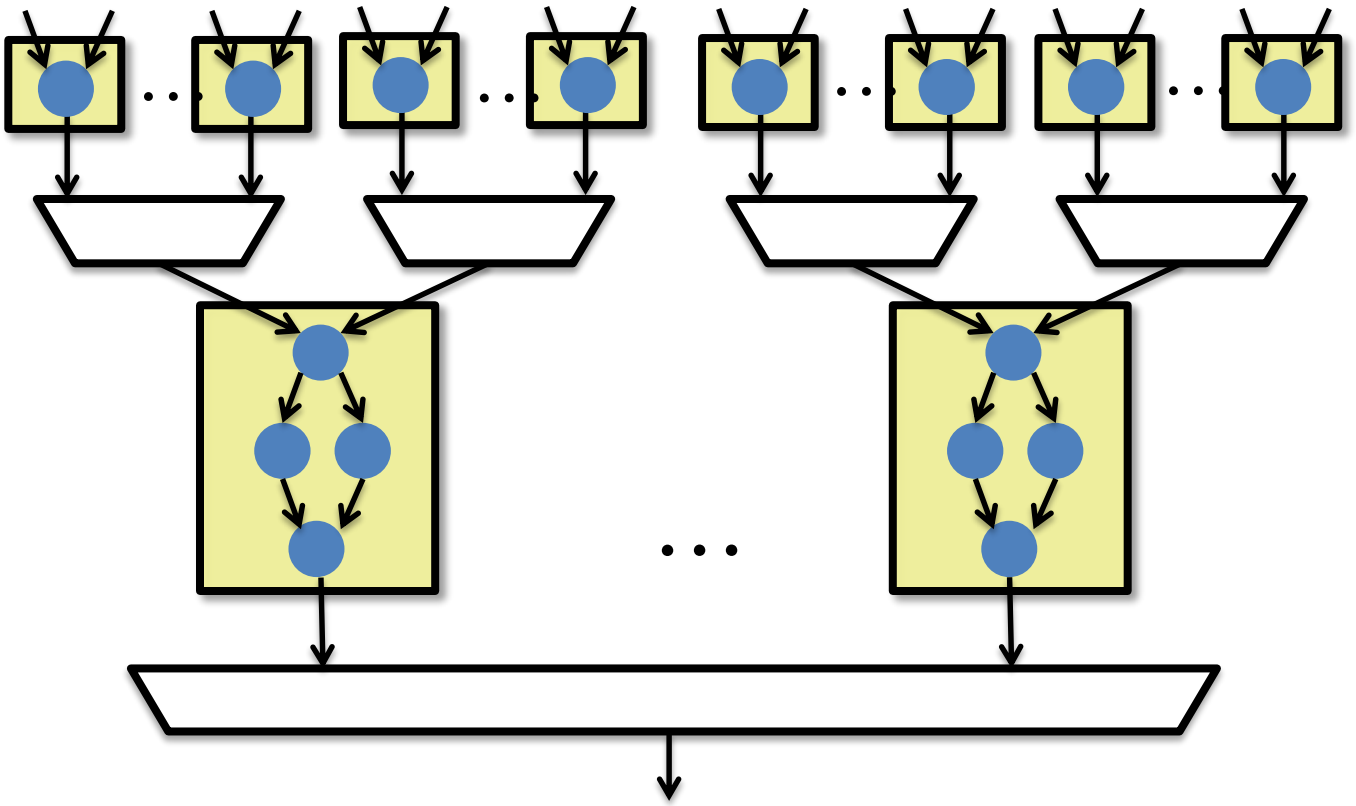
Integrity

Relational Logic Reasoning
[PLDI '12, PEPM '13]
(with M. Carbin, D. Kim, M. Rinard)

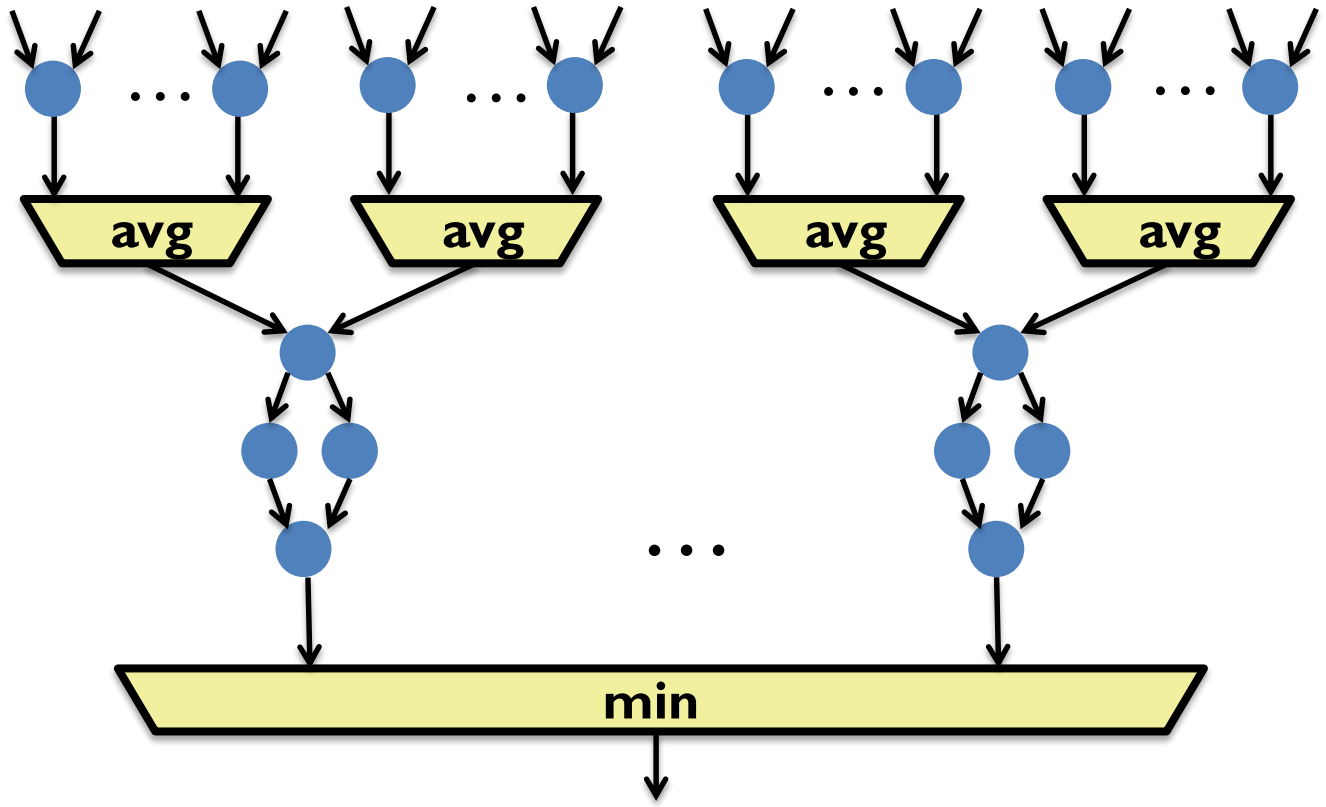
From [POPL '12]



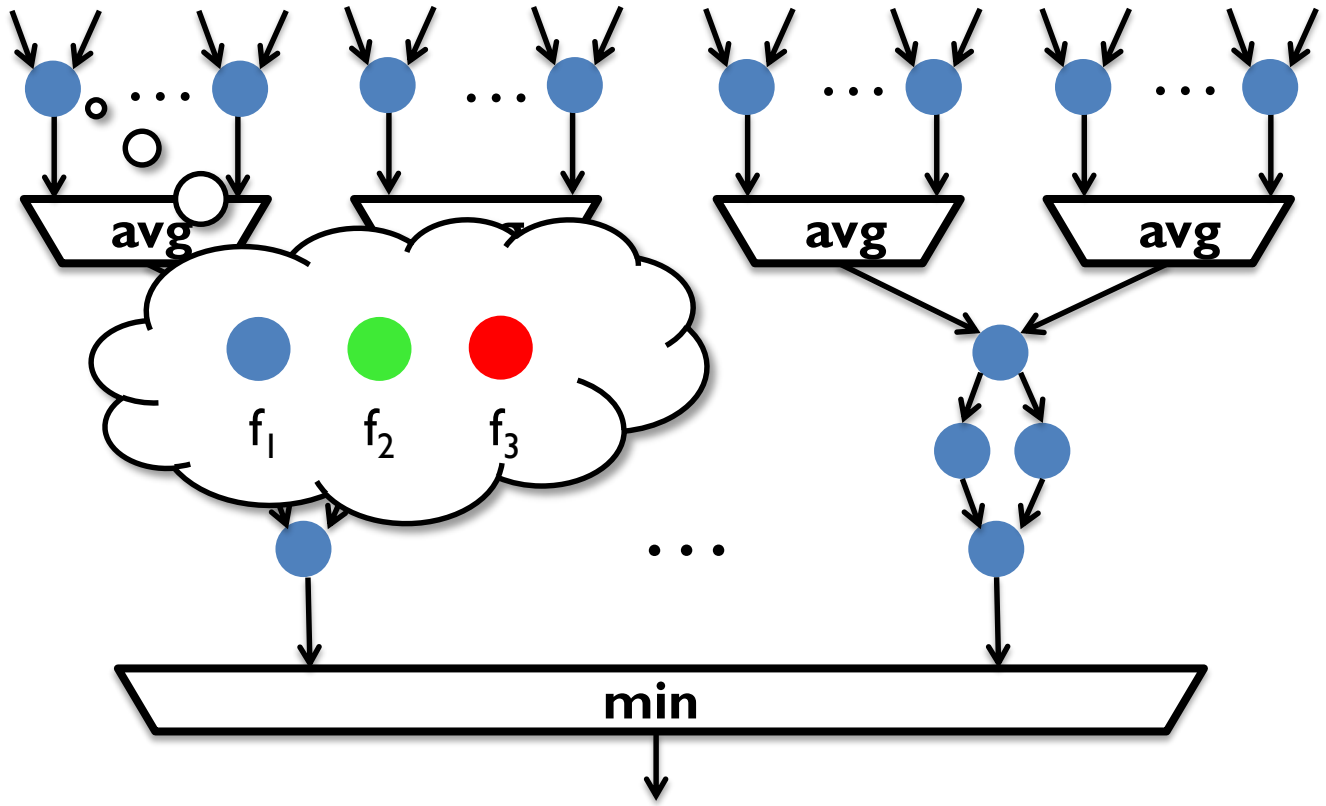
- Nodes represent computation
- Edges represent flow of data



- Functions – process individual data
- Reduction nodes – aggregate data

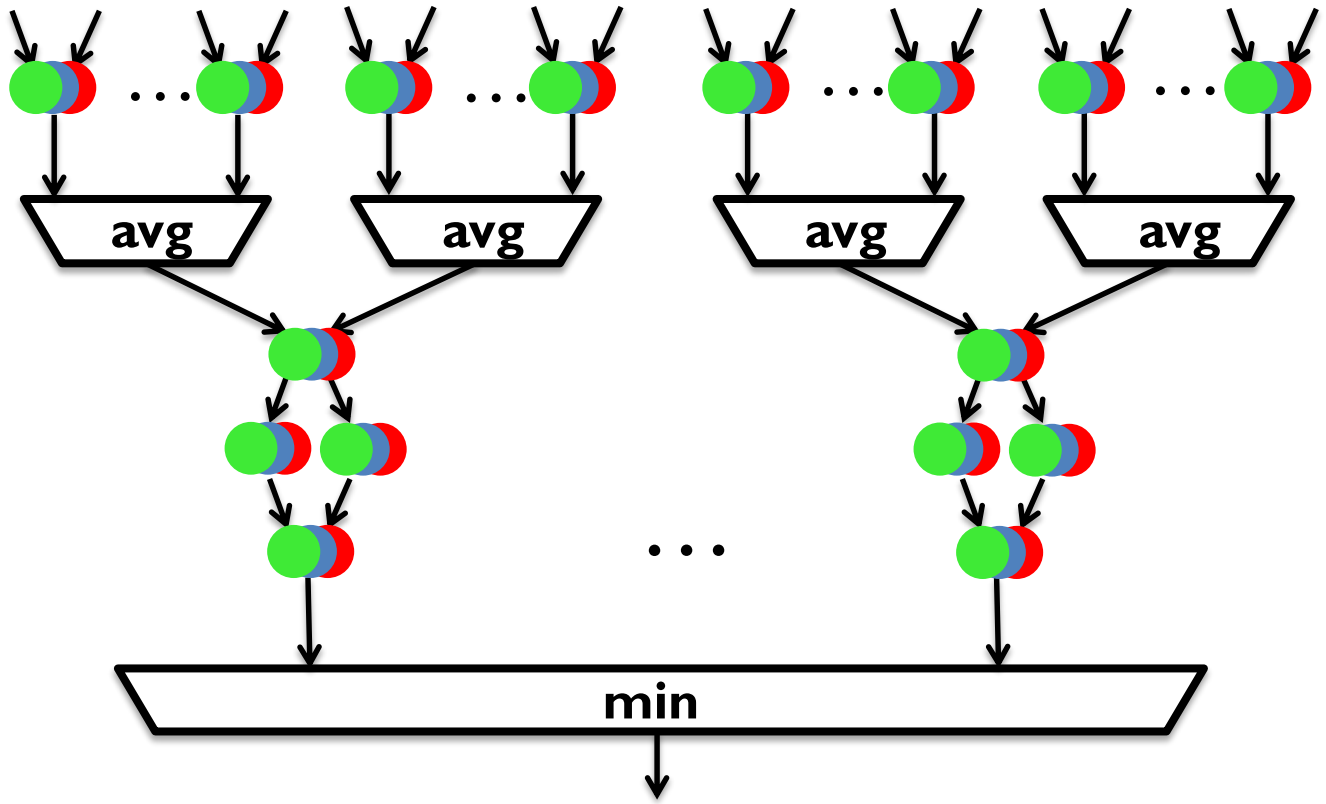


- Functions – process individual data
- Reduction nodes – aggregate data



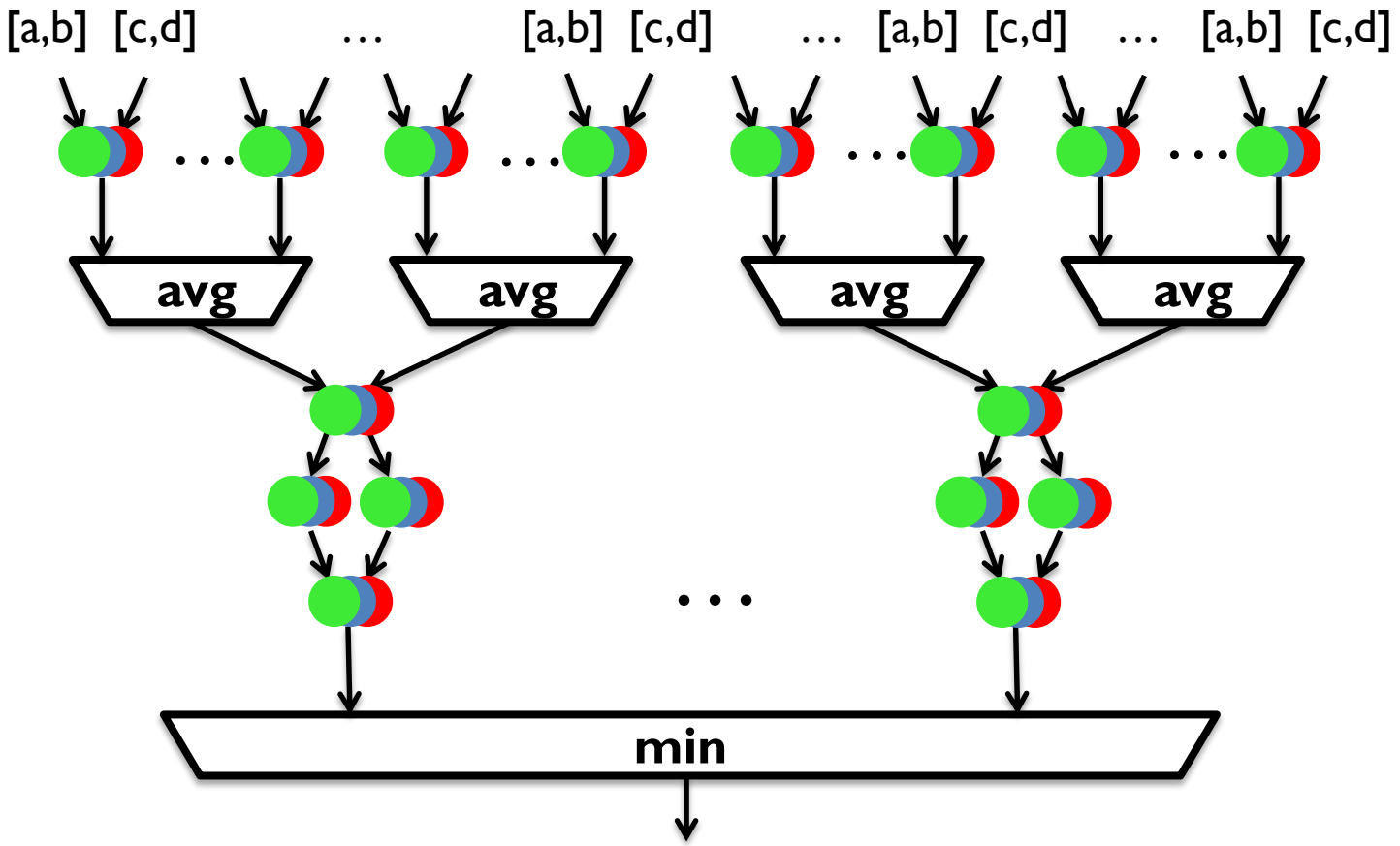
Function substitution

- Multiple implementations
- Each has expected error/time (E, T)



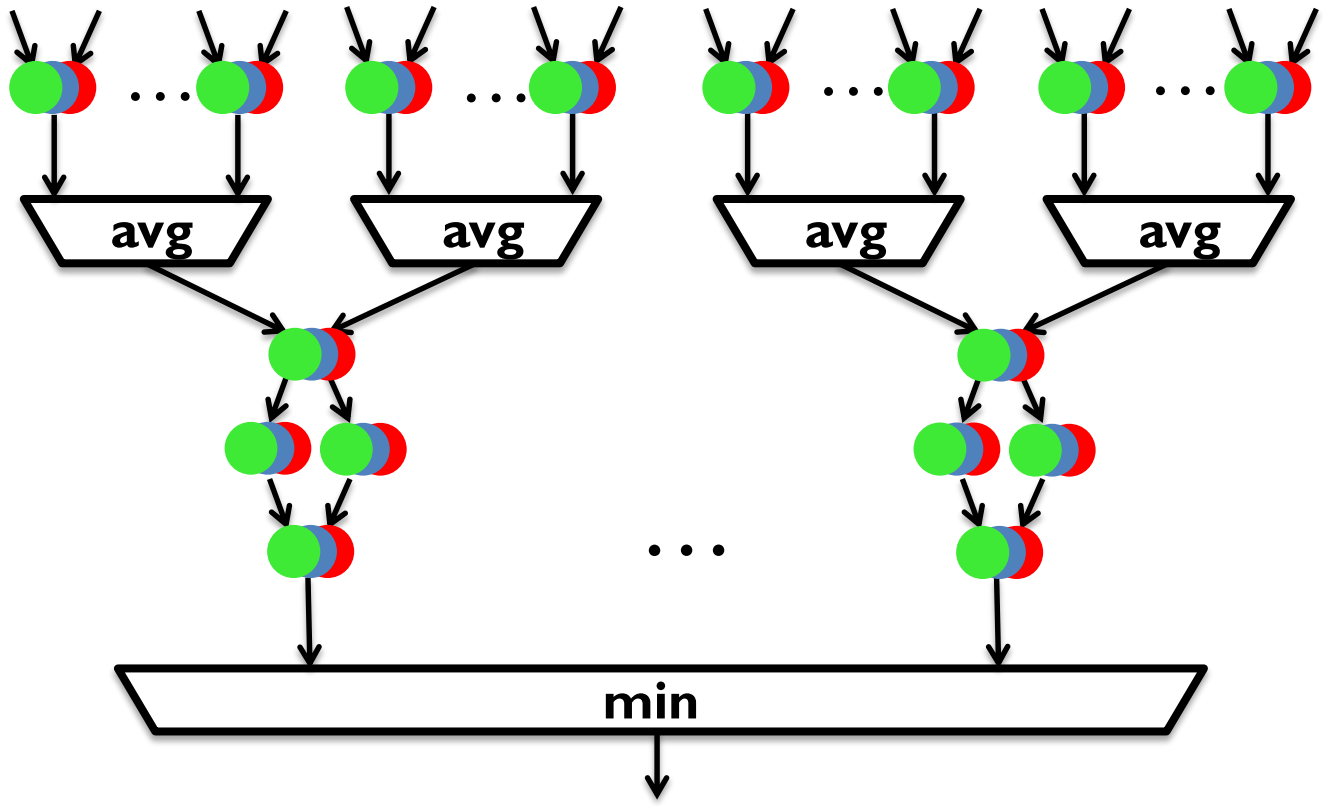
Function substitution

- Multiple implementations
- Each has expected error/time (E, T)



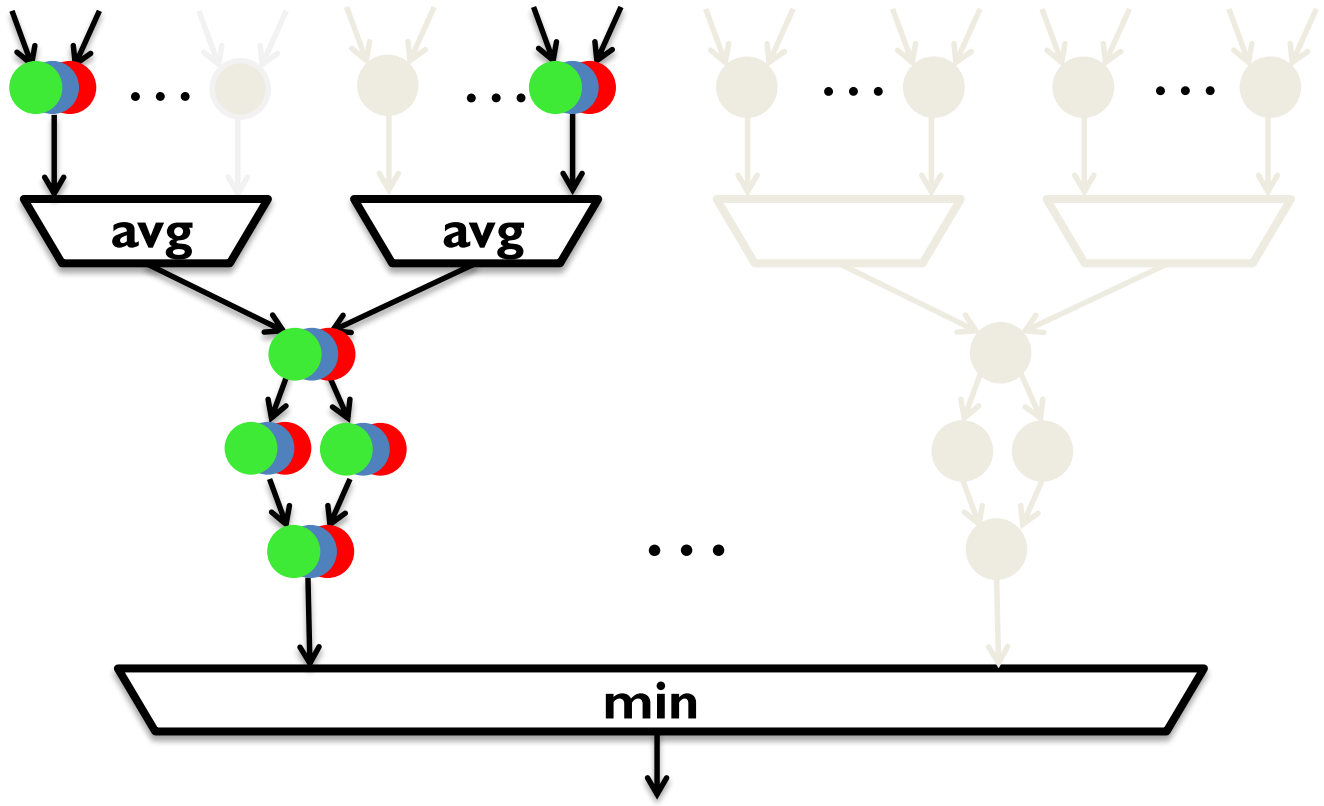
Function substitution

- Inputs of functions have specified ranges
- Each function has Lipschitz property



Sampling inputs of reduction nodes

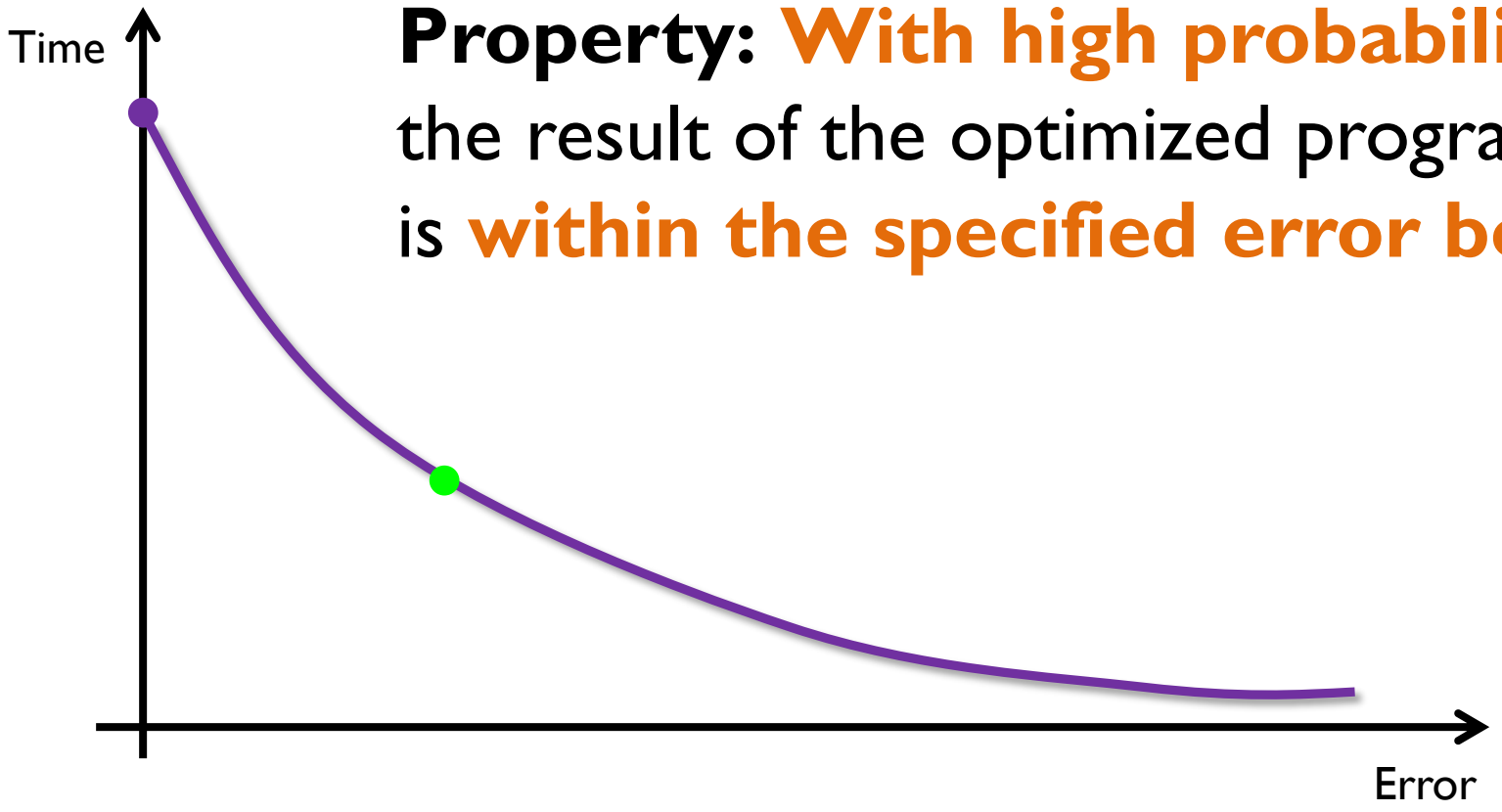
- Reductions consume fewer inputs



Sampling inputs of reduction nodes

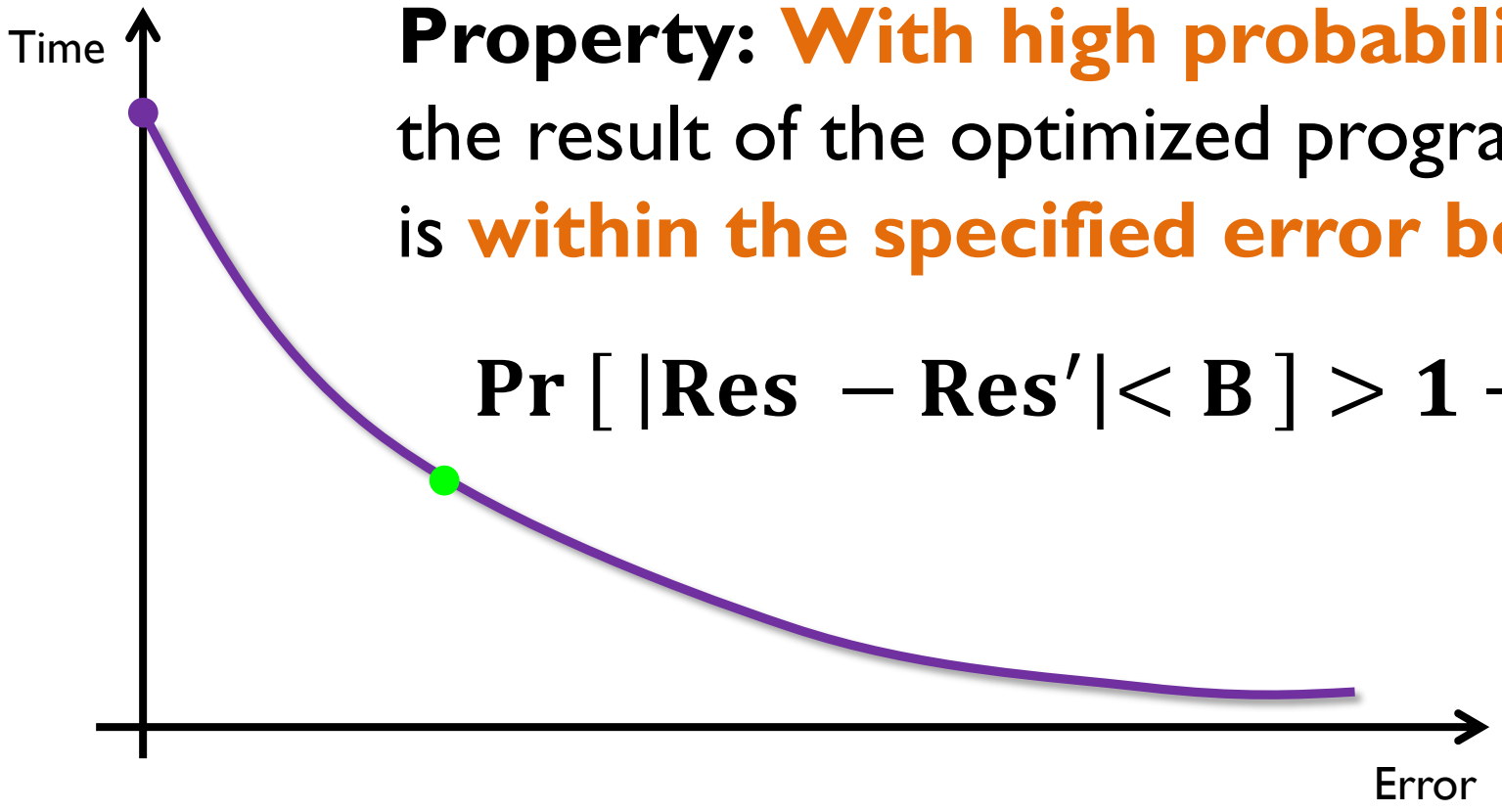
- Reductions consume fewer inputs

Search for Optimized Programs



Property: **With high probability**
the result of the optimized program
is **within the specified error bound**

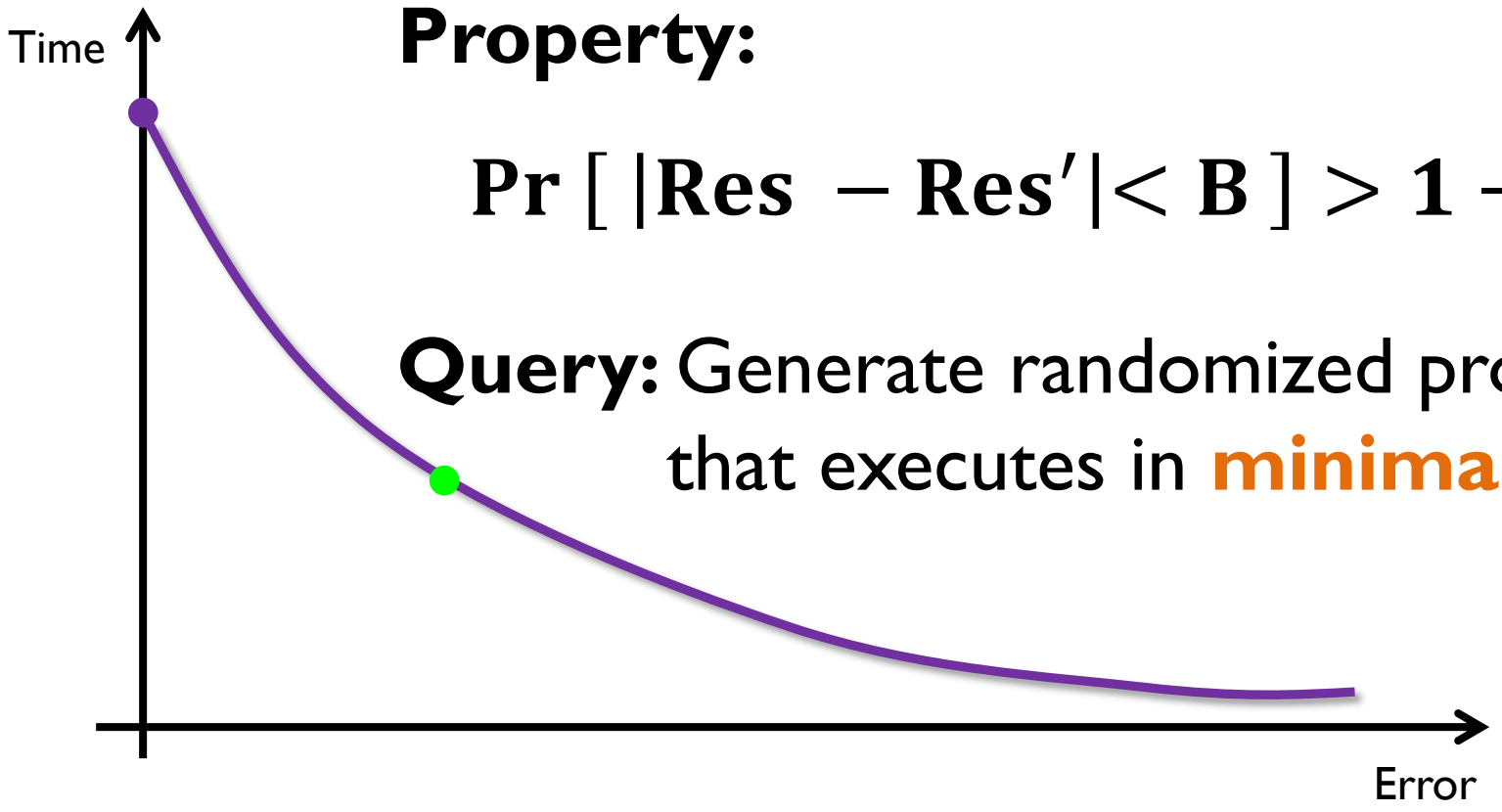
Search for Optimized Programs



Property: **With high probability** the result of the optimized program is **within the specified error bound**

$$\Pr\left[\,|\mathrm{Res} - \mathrm{Res}'| < B\,\right] > 1 - \delta$$

Search for Optimized Programs



Property:

$$\mathbf{Pr}\left[\,|\mathbf{Res} - \mathbf{Res}'| < \mathbf{B}\,\right] > \mathbf{1} - \boldsymbol{\delta}$$

Query: Generate randomized program that executes in **minimal time**

From [POPL '12]

Constraint Based Search Algorithm

Find Transformation Candidates:

- User provides function implementations and specs

Analyze Transformed Computations:

- Construct analytic expressions for (1) performance and (2) error emergence and propagation
- Variables: probabilities of executing alternate versions

Navigate Tradeoff Space:

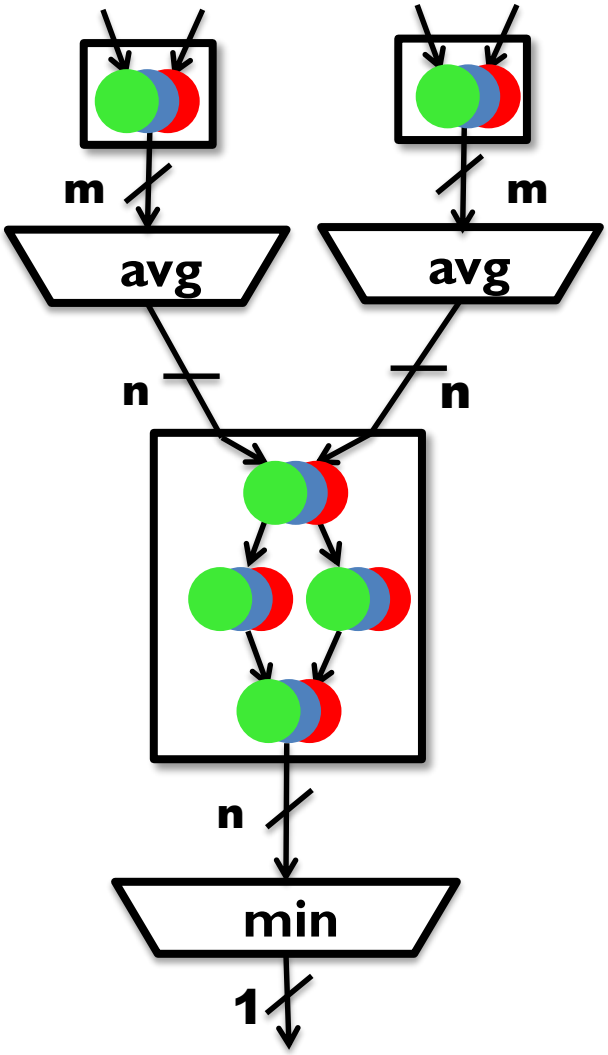
- Construct mathematical optimization problem:

  Using expressions for performance and error

- Non-linear Non-convex tradeoff space:

  $(1 + \varepsilon)$-approximation of globally optimal tradeoff curve

Tradeoff Curve Construction Algorithm



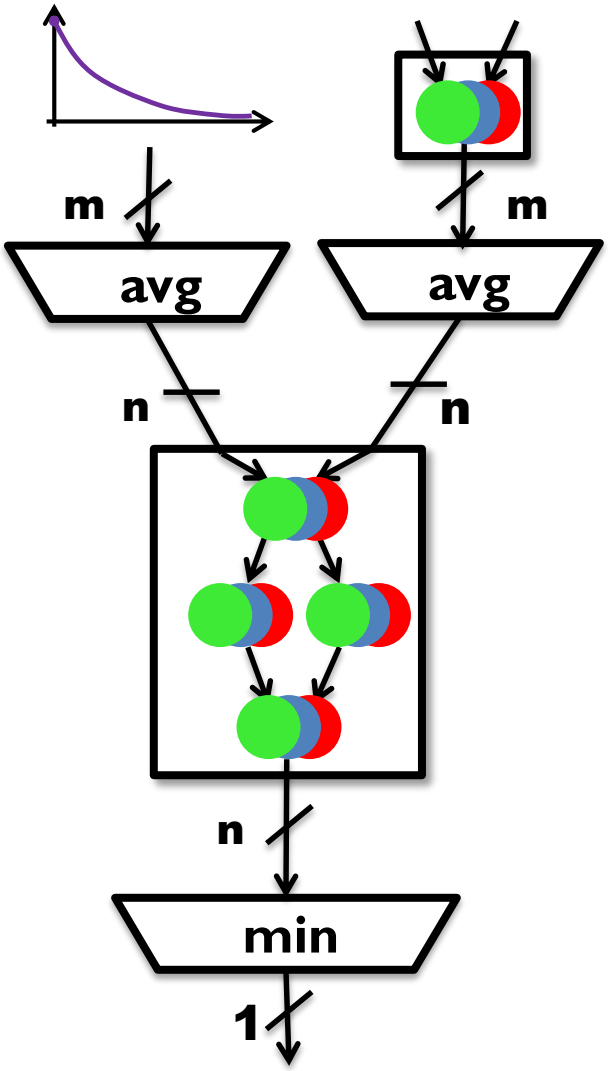
Divide and conquer

- For each subcomputation construct tradeoff curve
- Dynamic programming

Properties

- Polynomial time
- $(1+\varepsilon)$-approximation of true tradeoff curve

Tradeoff Curve Construction Algorithm



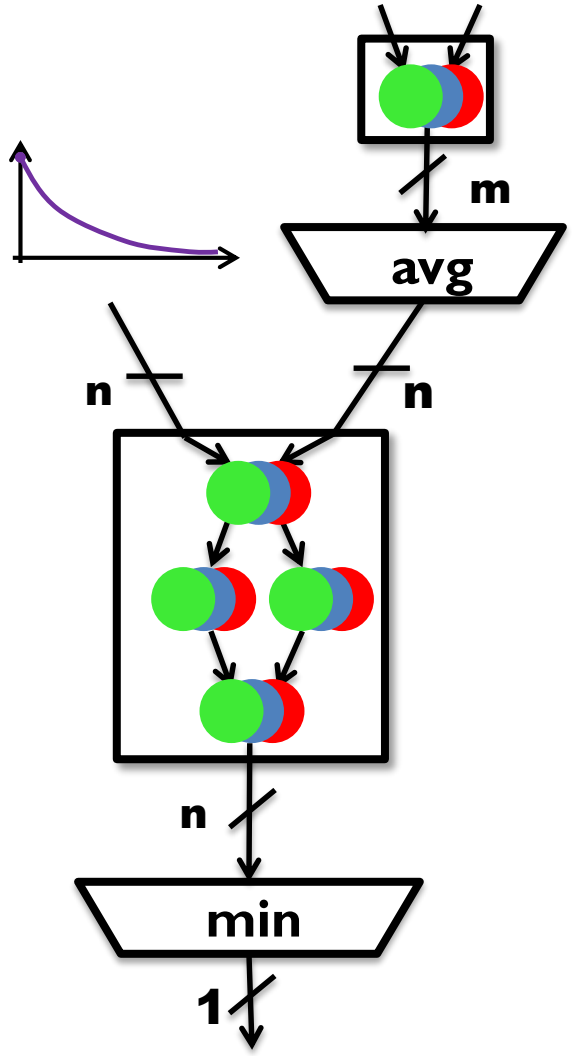
Divide and conquer

- For each subcomputation construct tradeoff curve
- Dynamic programming

Properties

- Polynomial time
- $(1+\varepsilon)$-approximation of true tradeoff curve

Tradeoff Curve Construction Algorithm



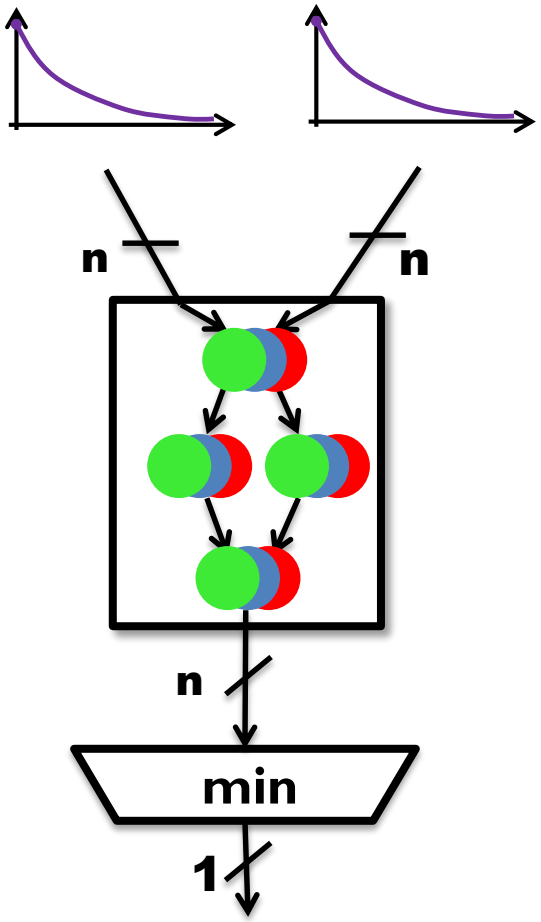
Divide and conquer

- For each subcomputation construct tradeoff curve
- Dynamic programming

Properties

- Polynomial time
- $(1+\varepsilon)$-approximation of true tradeoff curve

Tradeoff Curve Construction Algorithm



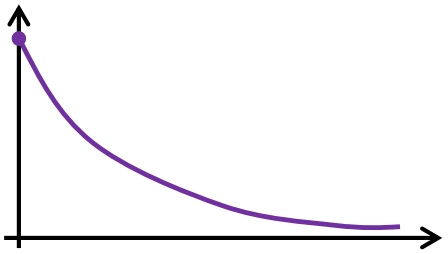
Divide and conquer

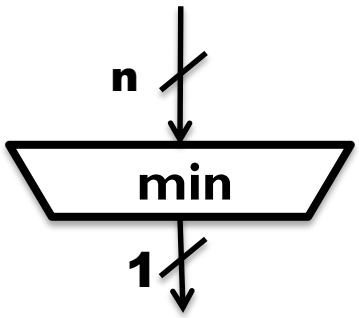
- For each subcomputation construct tradeoff curve
- Dynamic programming

Properties

- Polynomial time
- $(1+\varepsilon)$-approximation of true tradeoff curve

Tradeoff Curve Construction Algorithm





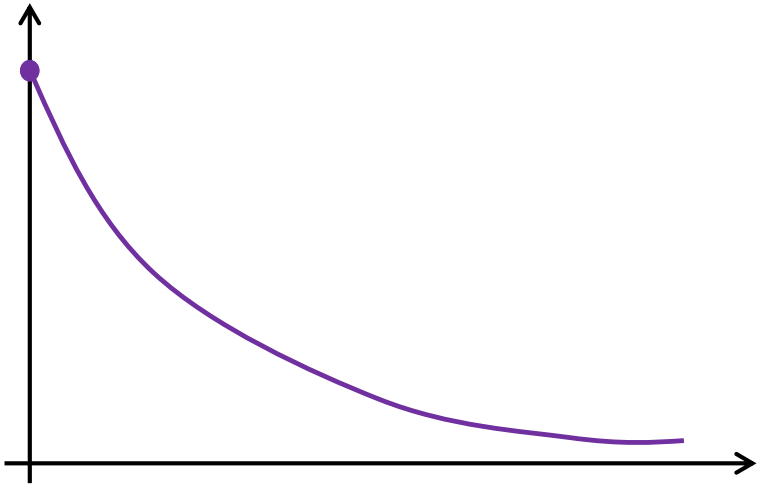
Divide and conquer

- For each subcomputation construct tradeoff curve
- Dynamic programming

Properties

- Polynomial time
- $(1+\varepsilon)$-approximation of true tradeoff curve

Tradeoff Curve Construction Algorithm



Divide and conquer

- For each subcomputation construct tradeoff curve
- Dynamic programming

Properties

- Polynomial time
- $(1+\varepsilon)$-approximation of true tradeoff curve

Comparison With Explicit Search

Finite vs Infinite Size Search Space

Input vs Declarative Specification Based

General vs Restricted Model of Computation

Related Work

Other Accuracy-aware Transformations We Explored:

- Task Skipping [Rinard ICS '06, Rinard OOPSLA '07]
- Loop Parallelization with Data Races [TECS PEC '12, RACES '12]
- Dynamic Knobs [ASPLOS '11]

Our group has also been working on transformations to prevent otherwise fatal errors (segmentation faults, infinite loops, buffer overflows, SQL injection attacks)

More Accuracy-aware Transformations Researchers Explored:

- Unreliable Data Stores [Liu et al ASPLOS '11, Sampson et al PLDI '11]
- Multiple Implementations [Ansel et al PLDI '09, Chilimbi et al PLDI '10]
- Approximate Memoization [Chaudhuri et al FSE '11]

Takeaway

Emerging trend of computations on **large data sets**

Accuracy-aware transformations are powerful tool

- Improve performance
- Reduce power
- Facilitate dynamic adaptation

Interaction of program analysis and search techniques to find **profitable, safe, and predictable** tradeoffs