CS 598sm

Probabilistic & Approximate Computing

http://misailo.web.engr.Illinois.edu/courses/cs598

Safari:

SOFTWARE TRANSFORMATIONS

Loop Perforation (2009)

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

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

Misailovic, Sidiroglou, Hoffmann, Rinard Quality of Service Profiling (ICSE 2010)
Sidiroglou, Misailovic , Hoffmann, Rinard Managing Performance vs. Accuracy Trade-offs With Loop Perforation (FSE 2011)

Loop Perforation

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

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

Loop Perforation

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

```
for (i = 0; i < n; i++) {
  if (rand(0.5)) continue;
  …
}
```

Reduction Sampling

```
for (i = 0; i < n; i++) {
    y = f( x[i] );
    s = s + y;
}
```

```
for (i = 0, z = 0; i < n; i++) {
    if (rand(0.75)) {z++; continue;}
    y = f( x[i] );
    s = s + y;
}
s = s * n/(n-z);
```

Zhu et al. Randomized Accuracy-Aware Program Transformations For Efficient Approximate Computations, POPL '12

Approximate Memoization

```
InType[] x; OutType[] y;
for (i = 0; i < n; i++) { y[i] = f(x[i]); }
```

```
var table = new Map<InType, OutType>;
for (i = 0; i < n; i++) {
  if ∃x',v . x'∈[x[i]-ε, x[i]+ε] && (x',v)∈table
      y[i] = v;
  else {
      y[i] = f(x[i]);
      table[x[i]] = y[i];
} }
```

Chaudhuri et al. Proving Programs Robust, FSE 2011

Approximate Tiling

```
InType[] x; OutType[] y;
for (i = 0; i < n; i++) { y[i] = f(x[i]); }
```

```
InType prev;
for (i = 0; i < n; i++) {
  if (i%2 == 1)
      y[i] = prev;
  else {
      y[i] = f(x[i]);
      prev = y[i];
} }
```

Chaudhuri et al. Proving Programs Robust, FSE '11

Samadi et al., Paraprox Pattern-Based Approximation for Data Parallel Applications, ASPLOS'14

Image Perforation: Automatically Accelerating Image Pipelines by Intelligently Skipping Samples, SIGGRAPH'16

Function Substitution

```
y = f(x);
```

```
y = f'(x);
```

Version	TimeSpec	ErrorSpec
f(x)	Time1	Err1
f'(x)	Time2	Err2

For instance, polynomial approximation of transcendental functions:

$$\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots \text{ for } x \text{ near } 0$$

$$R(x) \leq |x|^{n+1}/(n+1)!$$

Baek et al., PLDI 10;
Ansel et al., CGO '11

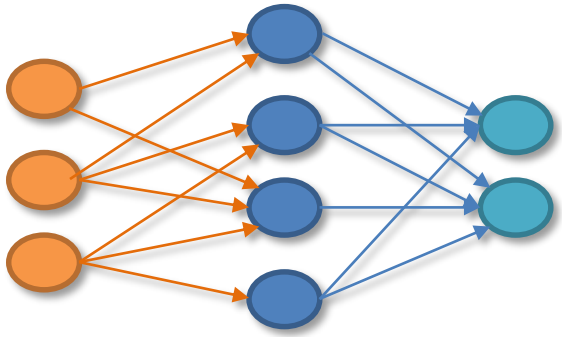
Function Substitution

```
y = f(x);
```

```
y = f'(x);
```

Version	TimeSpec	ErrorSpec
f(x)	Time1	Err1
f'(x)	Time2	Err2

Neural Network:



Esmaeilzadeh et al., Neural Acceleration for General-Purpose Approximate Programs, MICRO '12

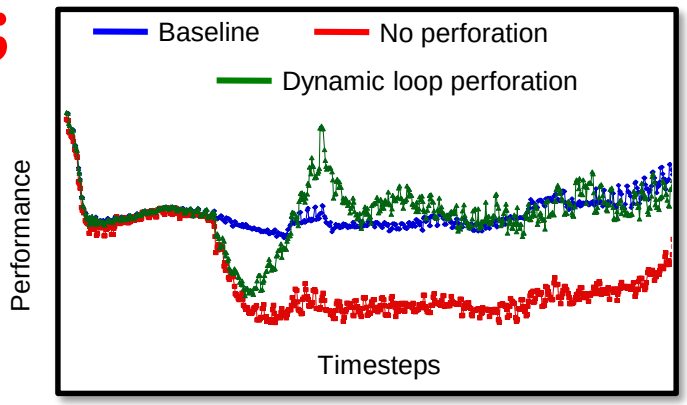
Dynamic Function Substitution

```
y = f(x);
```

Version	TimeSpec	ErrorSpec
f(x)	Time1	Err1
f'(x)	Time2	Err2

```
y = runtime.executeApprox()?
                f'(x): f(x);
```

- Baek et al., Green: A Framework for Supporting Energy-Conscious Programming using Controlled Approximation, PLDI 2010
- Hoffmann et al., Dynamic Knobs for Efficient Power Aware Computing, APSLOS 2011
- Mitra et al., Phase-aware Approximation in Approximate Computing CGO 2017



Floating Point Optimizations

```
double[] x, y
double z = f(x,y)
```

```
float[] x, y
float z = f(x,y)
```

Rubio-Gonzalez et al., Precimonious: Tuning Assistant for Floating-Point Precision, SC 2013

Skipping Tasks *(at Barrier Points)*

```
task {
 x = …
 y = …
}
```

```
task {
 x = …
 y = …
}
```

```
task {
 x = …
 y = …
}
```

```
task {
 x = …
 y = …
}
```

```
task {
 x = …
 y = …
}
```

```
task {
 x = …
 y = …
}
```

Continue execution after all tasks finish

```
task {
 x = …
 y = …
}
```

```
task {
 x = …
 y = …
}
```

```
task {
 x = …
 y = …
}
```

```
task {
 x = …
 y = …
}
```

```
task {
 x = …
 y = …
}
```

```
task {
 x = …
 y = …
}
```

Continue execution after all tasks finish before timeout, Otherwise kill delayed or non-responsive tasks

Rinard, Probabilistic accuracy bounds for fault-tolerant computations that discard tasks, ICS '06
Meng et al. Best-Effort Parallel Execution for Recognition and Mining Applications, IPDPS'09

Removing Synchronization

```
lock();
x = f(x,y);
y = g(x,y);
unlock();
```

```
lock();
x = f(x,y);
y = g(x,y);
unlock();
```

```
lock();
x = f(x,y);
y = g(x,y);
unlock();
```

```
lock();
x = f(x,y);
y = g(x,y);
unlock();
```

Renganarayana et al. Programming with Relaxed Synchronization, RACES '12
Misailovic et al. Dancing with Uncertainty, RACES '12

Transformations

Dimensions of impact:

- **Reducing computation**
  (perforation, memorization, tiling, function substitution)
- **Reducing data**
  (floating point optimizations)
- **Reducing communication/synchronization**
  (skipping tasks and lock elision)

Applying Transformations

Selecting where in code to approximate

- **Programmer-guided:** programmer writes annotations
- **Automatic:** system identifies the code and tunes the approximation
- **Combined:** programmer writes some annotations, system infers the rest
- **Interactive:** system identifies the code and presents the results to the developer who accepts/rejects

Applying Transformations

Choosing the level of approximation:

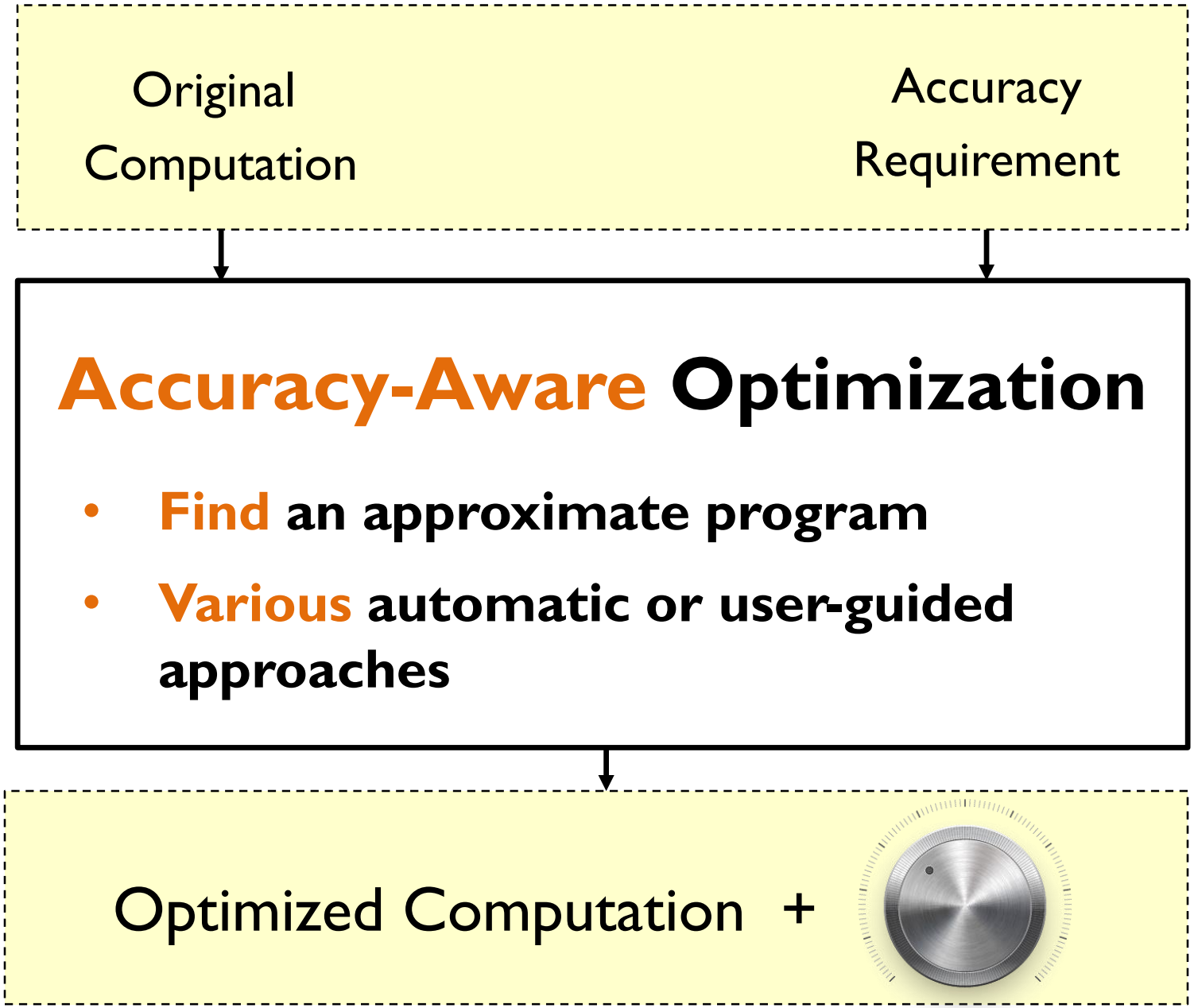
- Off-line: before execution starts
- On-line: during execution
- Combined: improve off-line models with on-line data

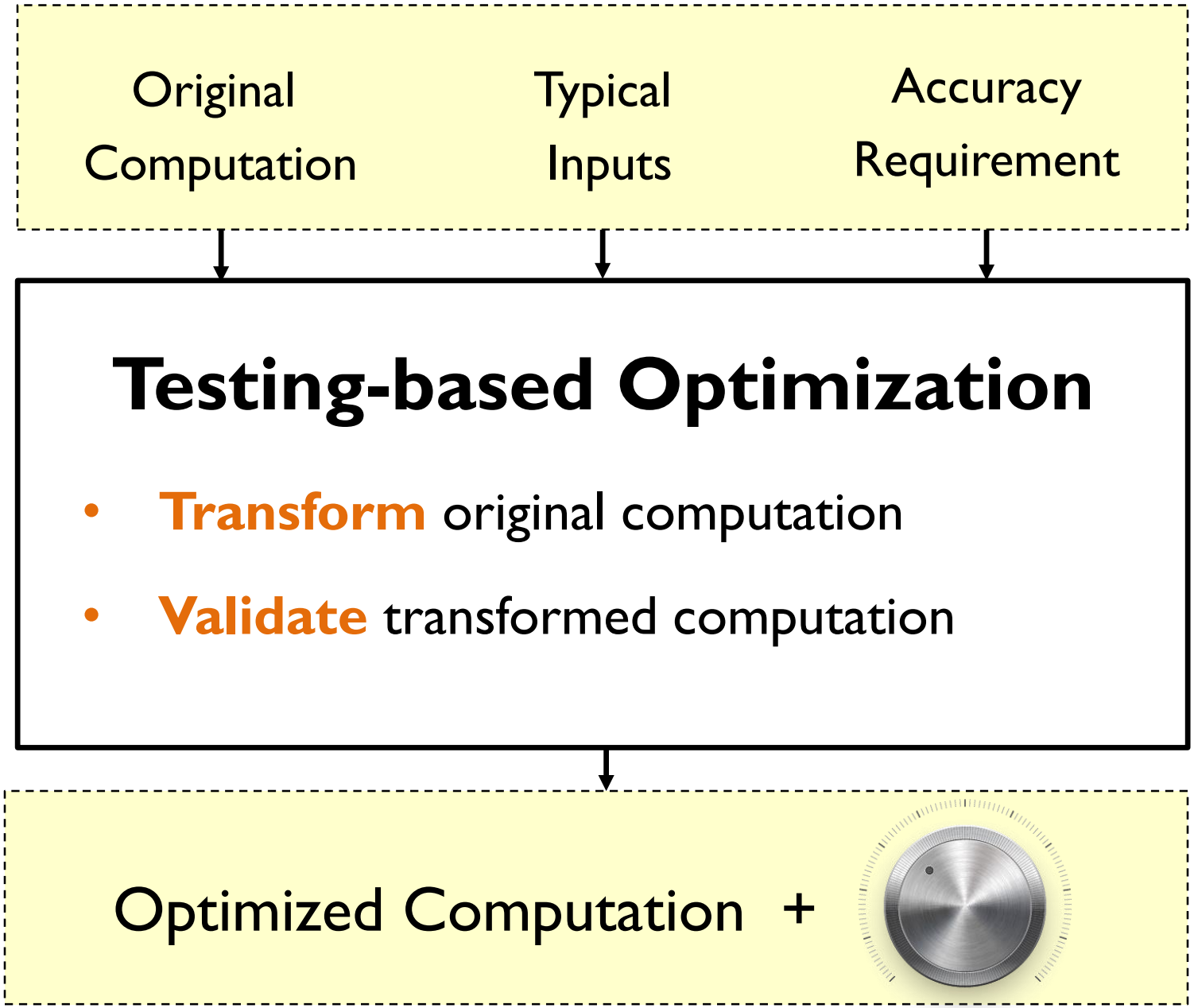
Some Key Characteristics:

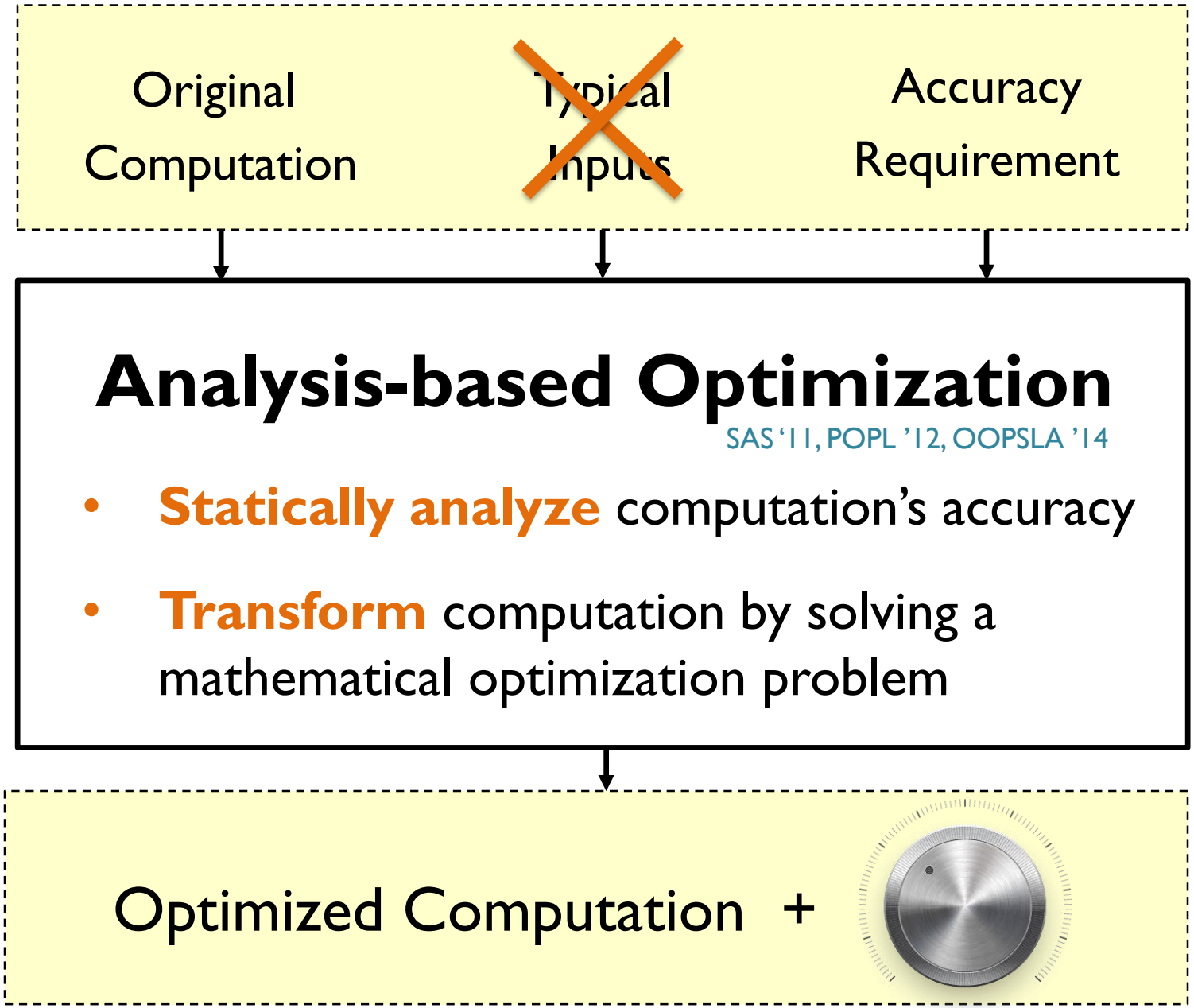
- **Approximate Kernel Computations**
  (have specific structure + functionality)

- **Accuracy vs Performance Knob**
  (tune how aggressively to approximate kernel)

- **Magnitude and Frequency of Errors**
  (kernels rarely exhibit large output deviations)


Original Computation
Accuracy Requirement
Accuracy-Aware Optimization
• Find an approximate program
• Various automatic or user-guided approaches
Optimized Computation +


Original Computation
Typical Inputs
Accuracy Requirement
Testing-based Optimization
• Transform original computation
• Validate transformed computation
Optimized Computation +


Original Computation
Typical Inputs
Accuracy Requirement
Analysis-based Optimization
SAS '11, POPL '12, OOPSLA '14
• Statically analyze computation's accuracy
• Transform computation by solving a mathematical optimization problem
Optimized Computation +

Background: Compiler Autotuning

Search for program with maximum performance by reordering instructions, compiler parameters, and program configurations

- There are so many ways to tile an array (e.g., fit different cache sizes)
- Which optimizations to try –O1, -O2, -O3, remove some, add some?

Empirical process: explores the complexity of the system stack:

- Try new configuration
- If better then previous, save; and
- Search for more profitable configuration

Interesting educational project: https://github.com/ctuning/ck/wiki/Compiler-autotuning

Compiler Autotuning

Try new configuration: select one combination out of the space of all possible combinations

- Often too large to try them all
- The results will depend on the inputs you used

If better: (traditionally) compare performance or energy

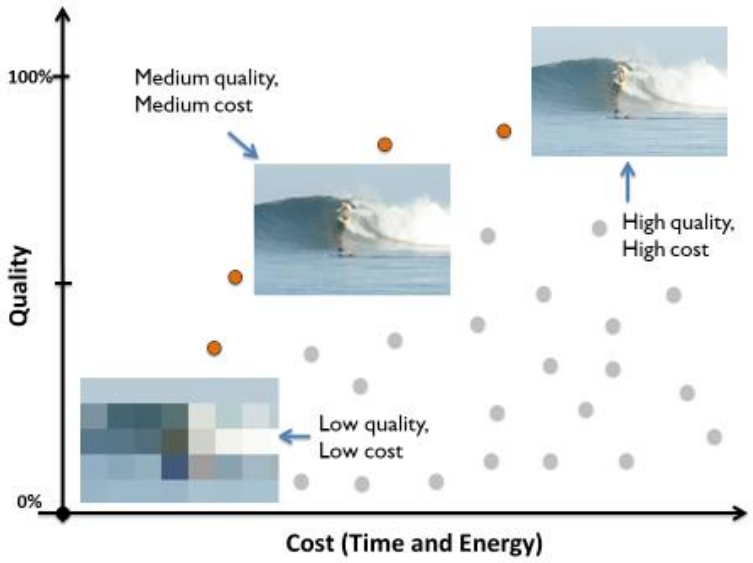
- Uses fitness function which orders the configurations

Search for more: various heuristic algorithms, these days mainly based on machine learning and heuristic search (e.g., genetic programming in OpenTuner)

A Survey on Compiler Autotuning using Machine Learning (CSUR 2019)

Compiler Autotuning

Accuracy opens up a new dimension for search



- Increases the number of options to try
- Includes (input-specific) accuracy metric in the fitness fun.
- Finds the configurations with best tradeoffs.

Petabricks

Language for algorithmic choice (expresses options to tune) and an autotuner (using genetic search)

Precusor to OpenTuner (popular autotuner)

Hand-coded algorithmic compositions are commonplace. A typical example of such a composition can be found in the C++ Standard Template Library (STL)[1] routine `std::sort`, which uses merge sort until the list is smaller than 15 elements and then switches to insertion sort. Our tests have shown that higher cutoffs (around 60-150) perform much better on current architectures. However, because the optimal cutoff is dependent on architecture, cost of the comparison routine, element size, and parallelism, no single hard-coded value will suffice.

Petabricks

Language for algorithmic choice (expresses options to tune) and an autotuner (using genetic search)

Precusor to OpenTuner (popular autotuner)

Classes of algorithms that can benefit from approximation:

- Polyalgorihtms
- NP-Complete Algorithms
- Iterative Algorithms
- Signal Processing

Language and Compiler Support for Auto-TuningVariable-Accuracy Algorithms (CGO 2011)

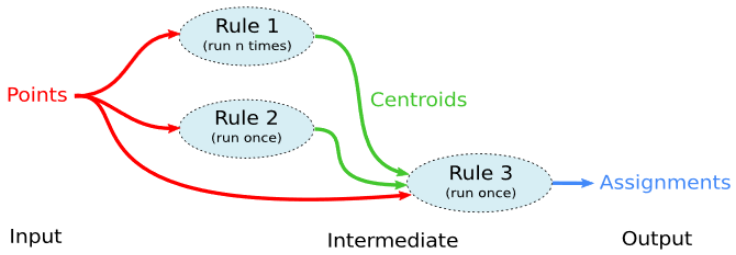
Petabricks Autotuner

```
transform kmeans
from Points[n,2] // Array of points (each column
                 // stores x and y coordinates)
through Centroids[sqrt(n),2]
to Assignments[n]
{
  // Rule 1:
  // One possible initial condition: Random
  // set of points
  to(Centroids.column(i) c) from(Points p) {
    c=p.column(rand(0,n))
  }

  // Rule 2:
  // Another initial condition: Centerplus initial
  // centers (kmeans++)
  to(Centroids c) from(Points p) {
    CenterPlus(c, p);
  }

  // Rule 3:
  // The kmeans iterative algorithm
  to(Assignments a) from(Points p, Centroids c) {
    while (true) {
      int change;
      AssignClusters(a, change, p, c, a);
      if (change==0) return; // Reached fixed point
      NewClusterLocations(c, p, a);
    }
  }
}
```

The rules contained in the body of the transform define the various pathways to construct the Assignments data from the initial Points data.



Language and Compiler Support for Auto-TuningVariable-Accuracy Algorithms (CGO 2011)

Petabricks Autotuner

```
transform kmeans
accuracy_metric kmeansaccuracy
accuracy_variable k
from Points[n,2] // Array of points (each column
                 // stores x and y coordinates)
through Centroids[k,2]
to Assignments[n]
{
```

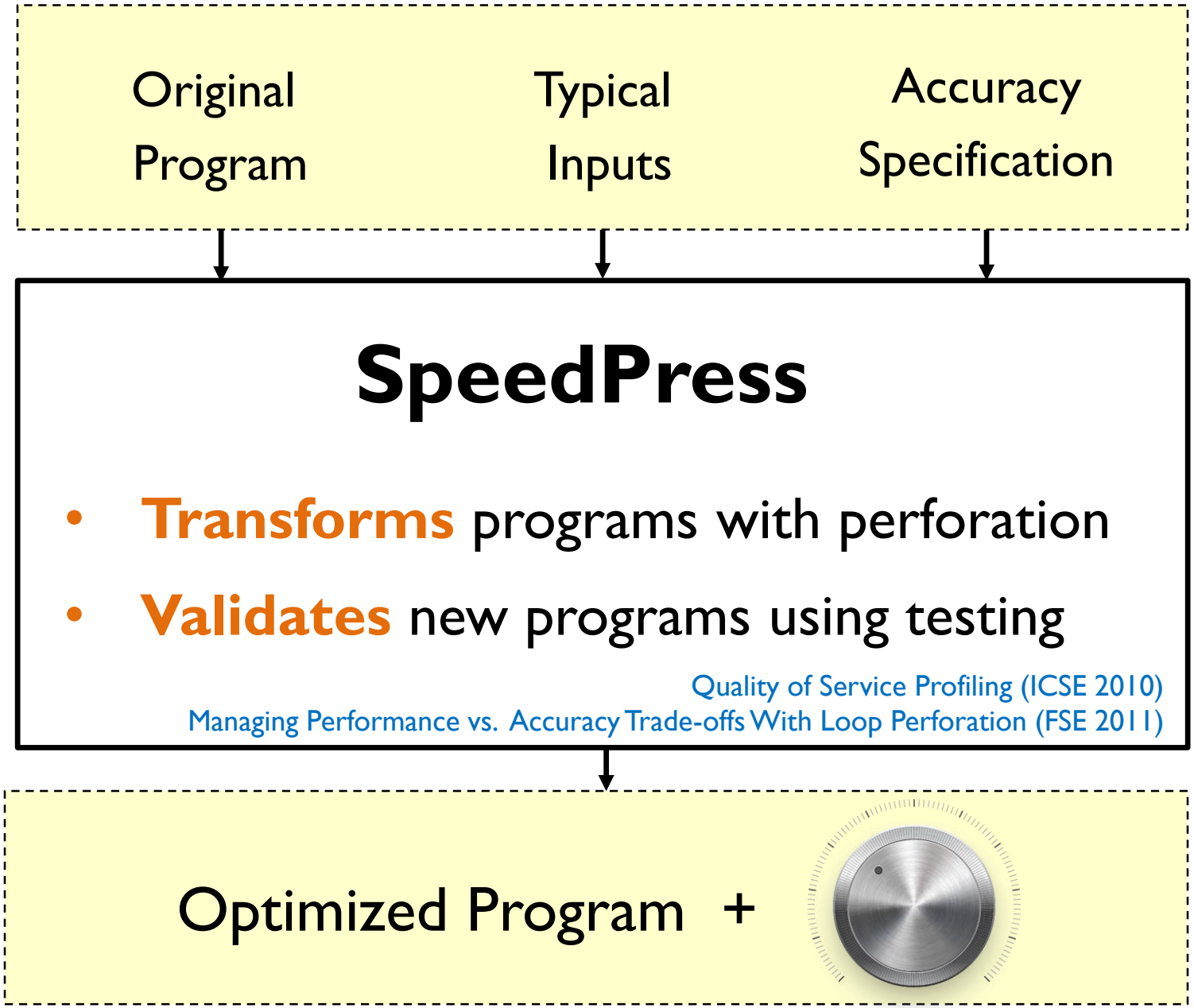
... (Rules 1 and 2 same as in Figure 1) ...

```
  // Rule 3:
  // The kmeans iterative algorithm
  to(Assignments a) from(Points p, Centroids c) {
    for_enough {
      int change;
      AssignClusters(a, change, p, c, a);
      if (change==0) return; // Reached fixed point
      NewClusterLocations(c, p, a);
    }
  }
}

transform kmeansaccuracy
from Assignments[n], Points[n,2]
to Accuracy
{
  Accuracy from(Assignments a, Points p){
    return sqrt(2*n/SumClusterDistanceSquared(a,p));
  }
}
```

Next Step

What if a language **does not** expose approximation choices?


Original Program
Typical Inputs
Accuracy Specification
SpeedPress
• Transforms programs with perforation
• Validates new programs using testing
Quality of Service Profiling (ICSE 2010)
Managing Performance vs. Accuracy Trade-offs With Loop Perforation (FSE 2011)
Optimized Program +

x264 Video Encoder Example

Typical Inputs

Accuracy Specification

- **Quality Metric:**
  e.g. PSNR and bit rate
- **Quality Loss:**
  e.g. relative difference **<10%**

Search for Perforatable Loops

- Run **performance** profiler
  Identify time consuming loops
- Perforate **one loop** at a time
  Filter out loops that do not satisfy accuracy requirement
- Perforate **multiple loops** together
  Find combinations of loops that maximize performance

Validate Perforated Loops

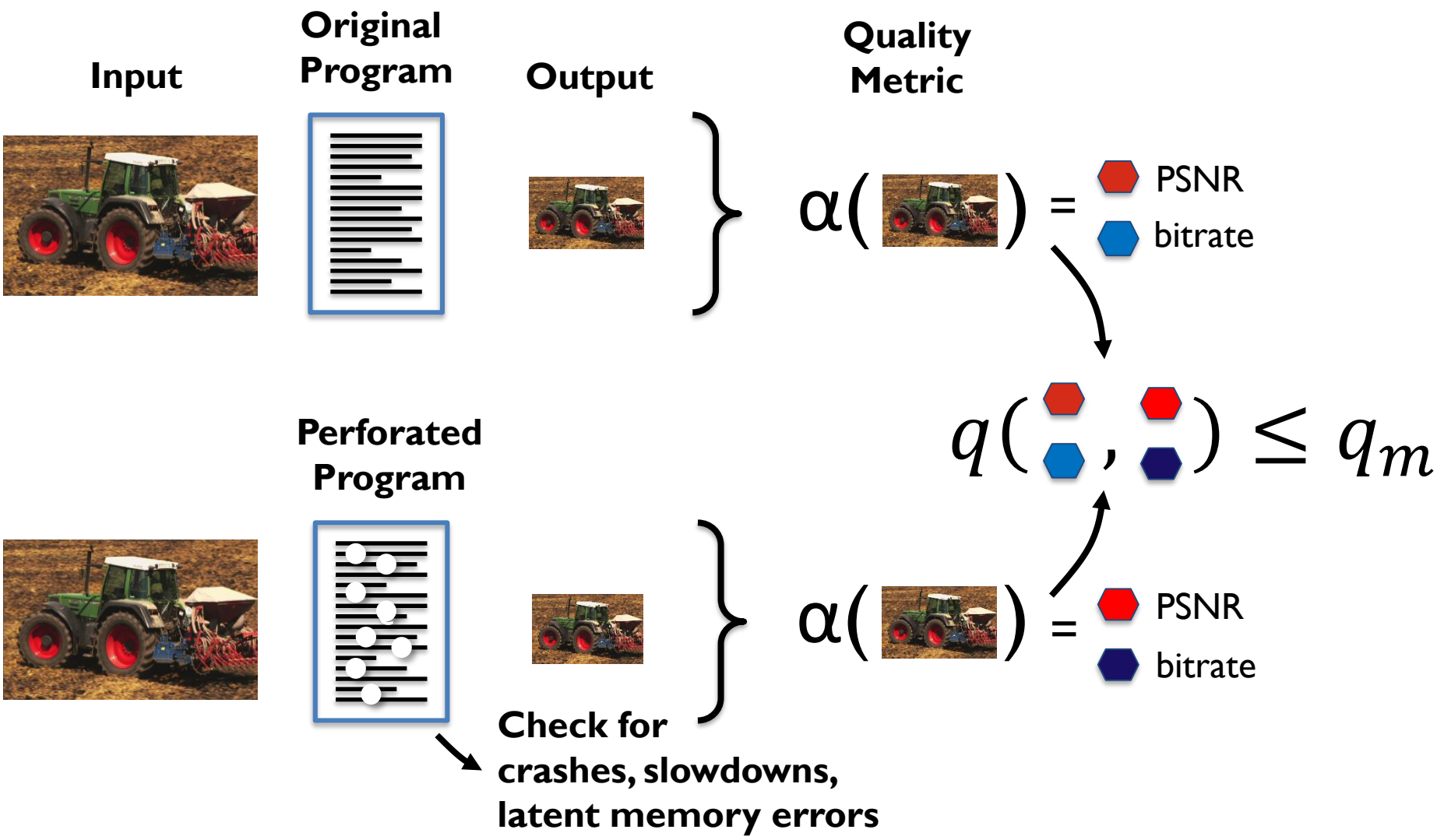
Filter out loops that do not satisfy requirement

Criticality Testing: Ensure that the program with perforated loop **does not:**

- Crash or return error
- Runs slower than original (or not terminates)
- Causes other errors identified by dynamic analysis (e.g., latent memory errors)
- Produces unacceptable result (e.g., NaN, inf…)
- Produces inaccurate result (according to accuracy metric)

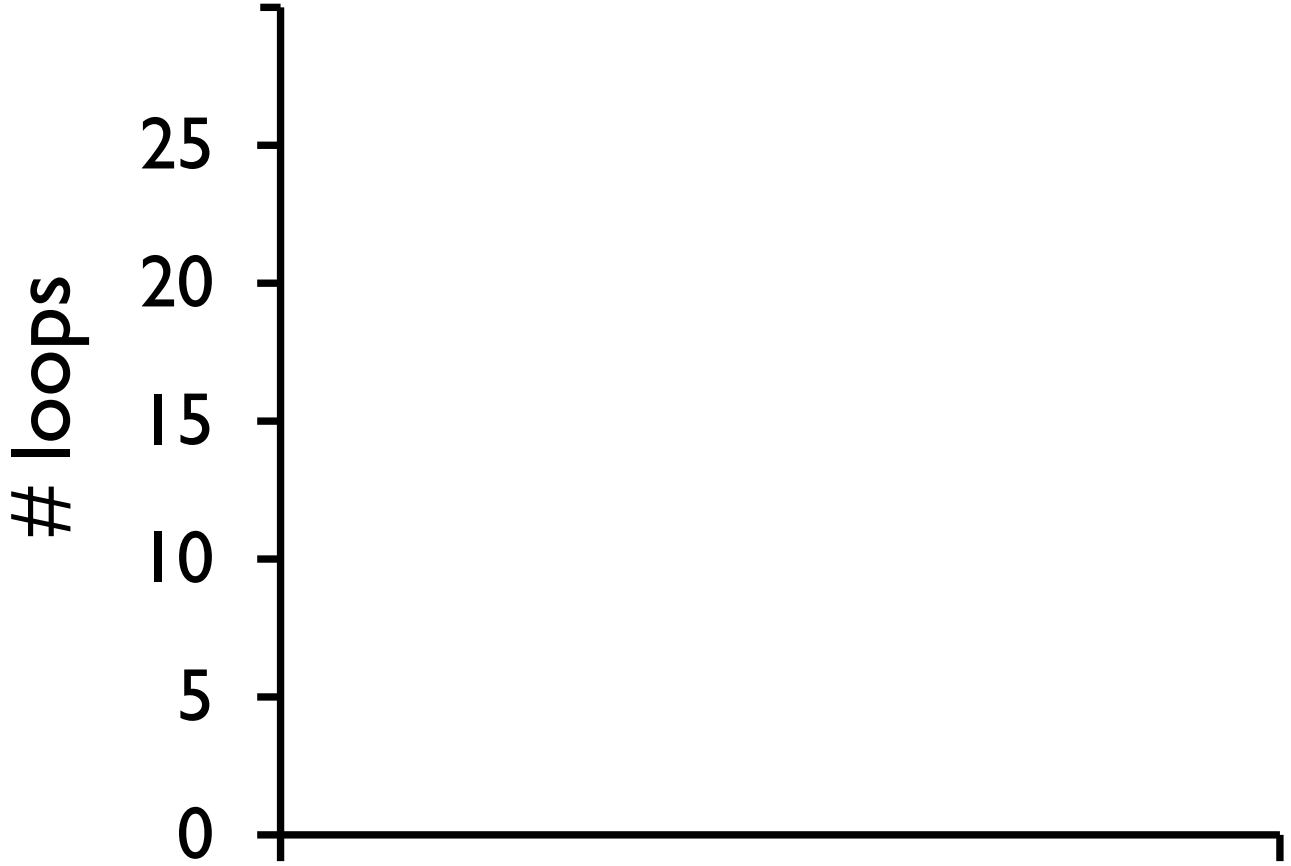
Validate Perforated Loops

Filter out loops that do not satisfy requirement



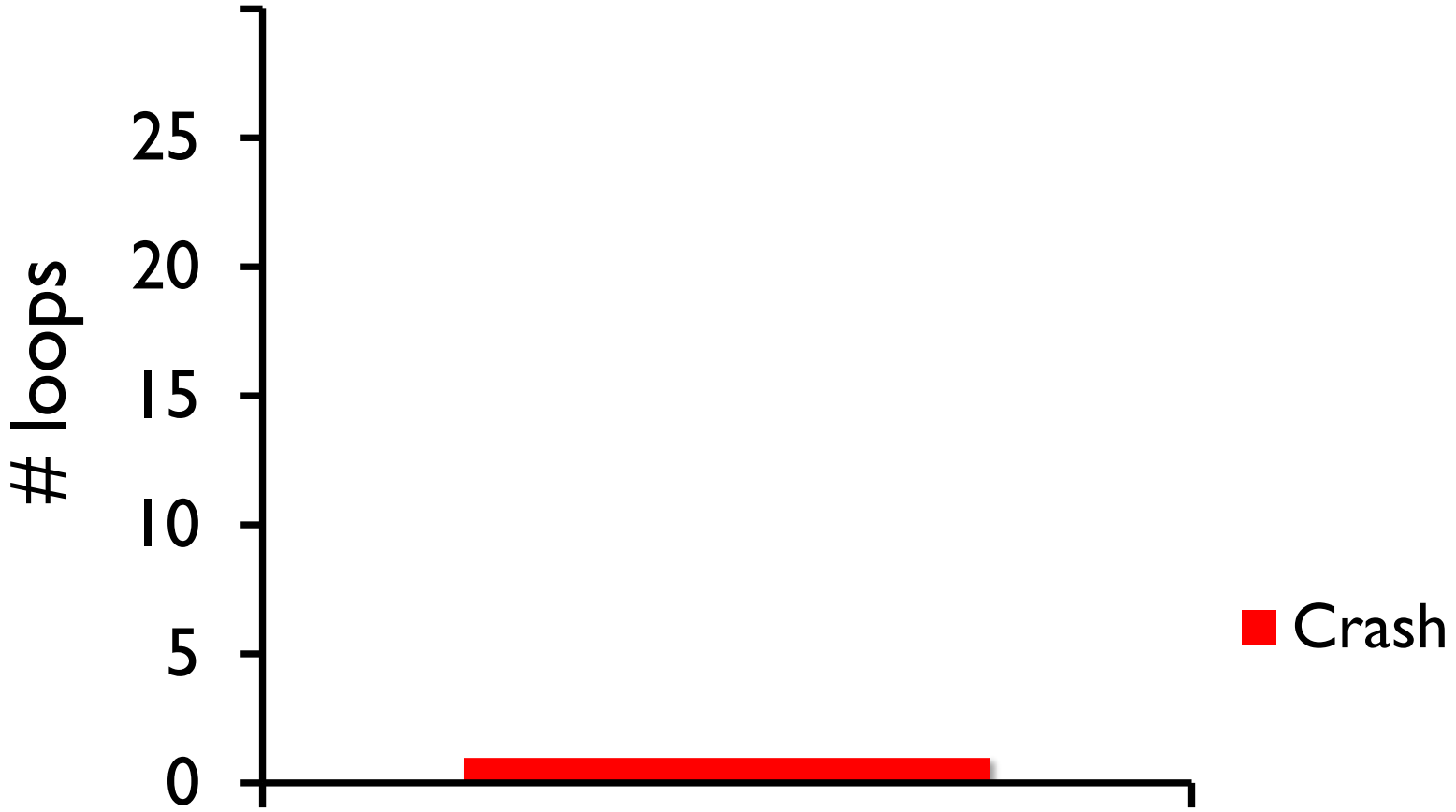
Perforating Individual Loops in x264

(Quality Loss < 0.1)



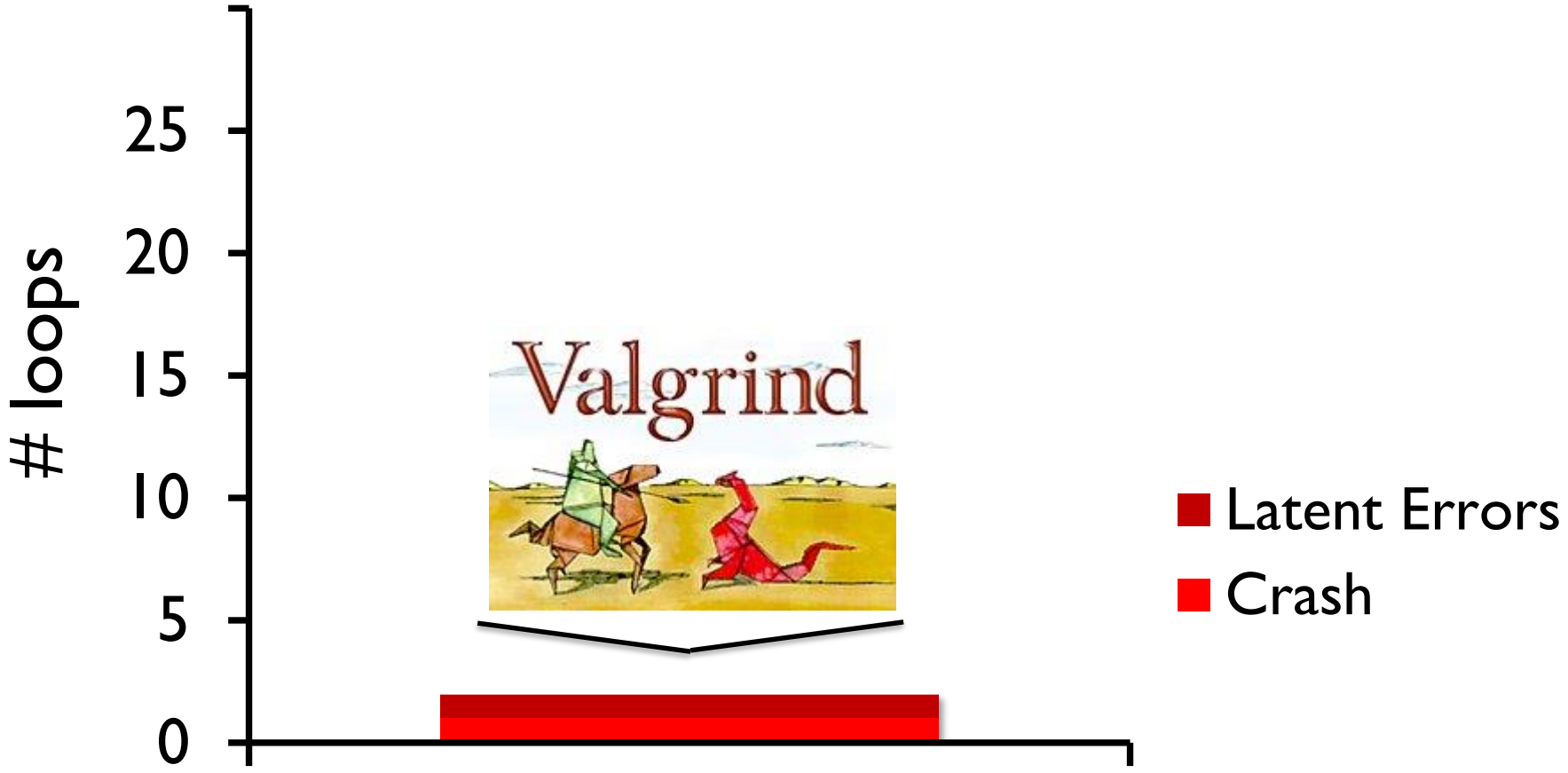
Perforating Individual Loops in x264

(Quality Loss < 0.1)



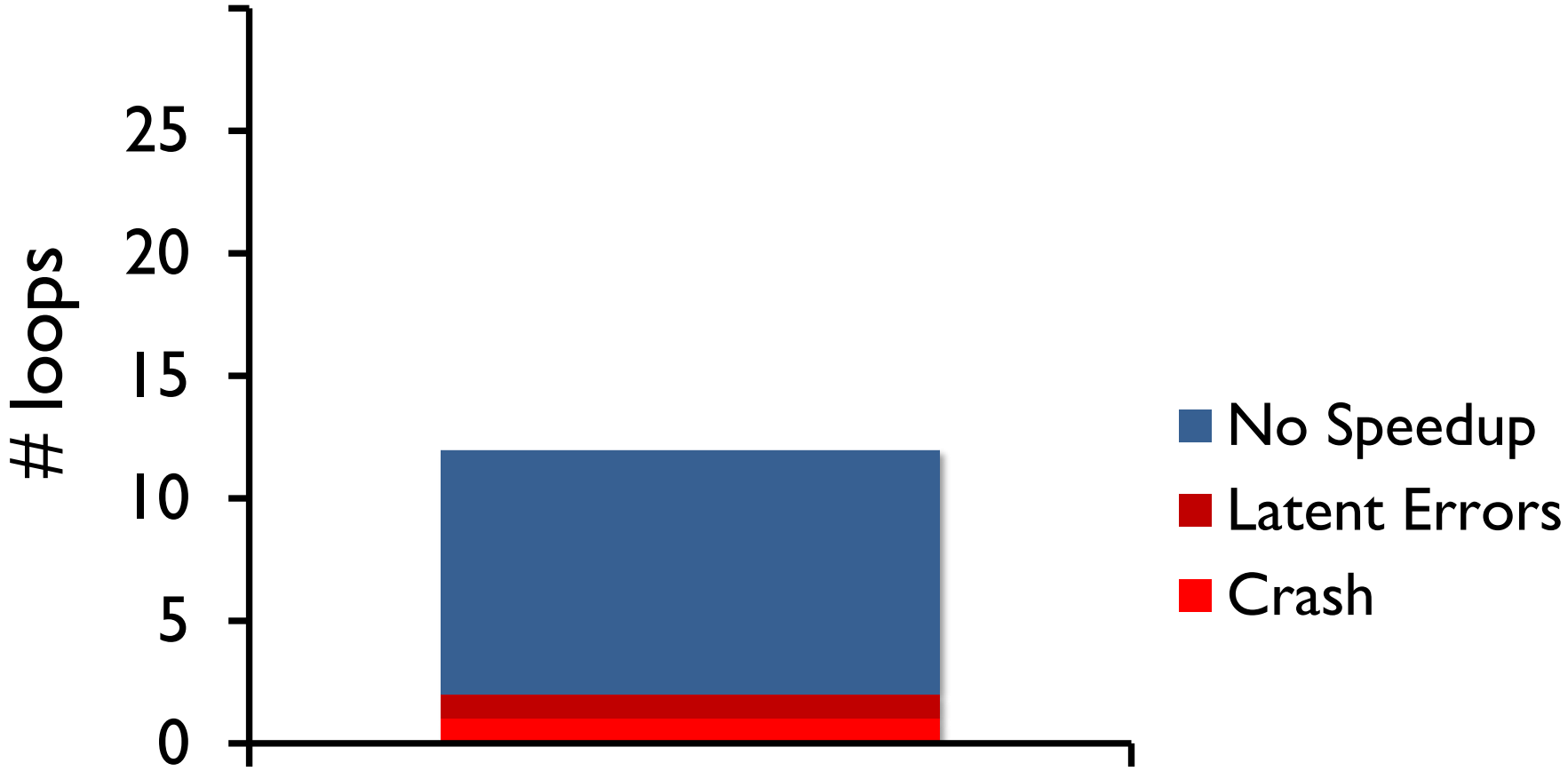
Perforating Individual Loops in x264

(Quality Loss < 0.1)



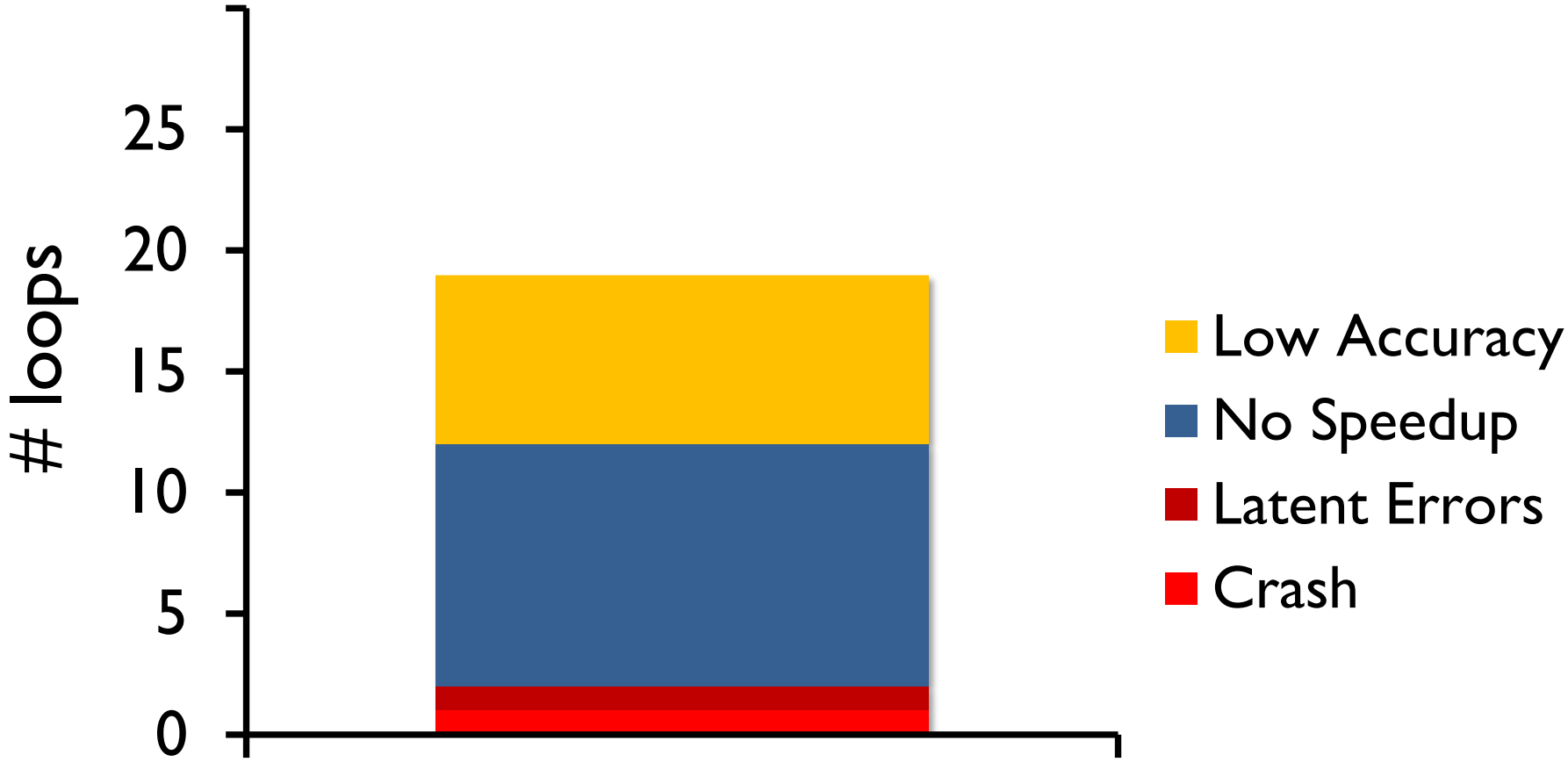
Perforating Individual Loops in x264

(Quality Loss < 0.1)



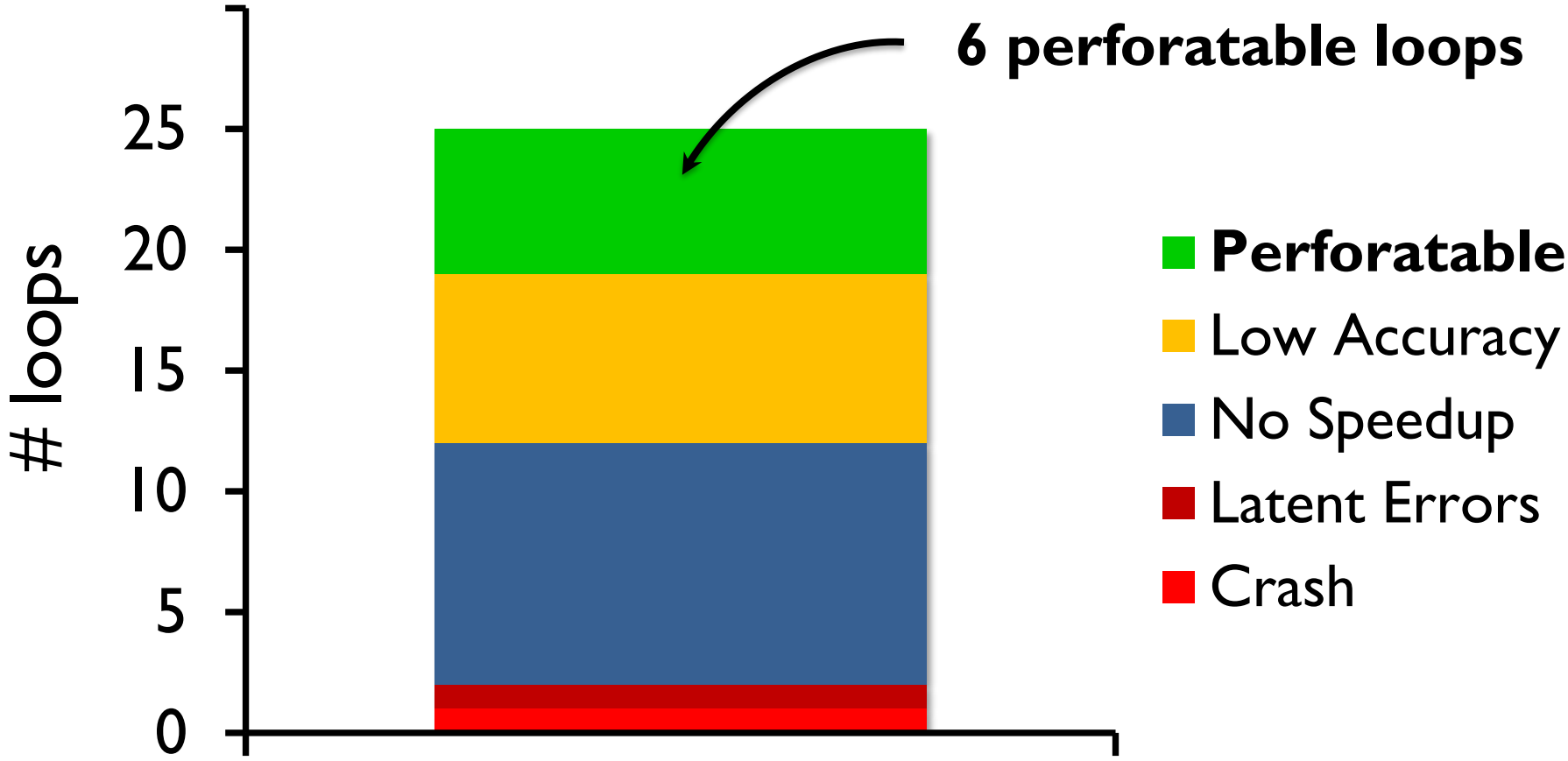
Perforating Individual Loops in x264

(Quality Loss < 0.1)

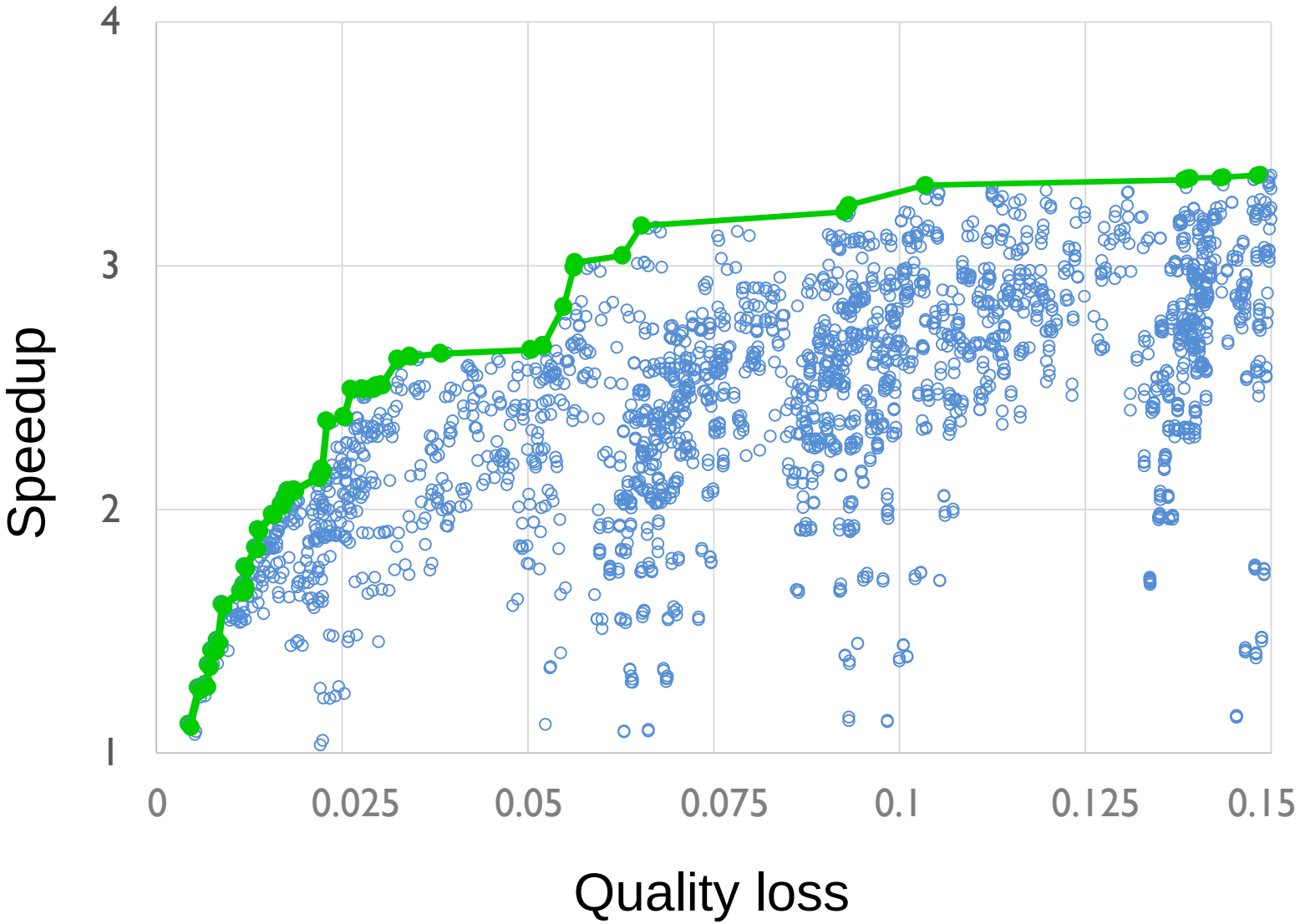


Perforating Individual Loops in x264

(Quality Loss < 0.1)



Navigate Tradeoff Space



Applications

From PARSEC Suite

x264	video encoder
bodytrack	human motion tracking
swaptions	financial analysis
ferret	image search
canneal	electronic circuit placement
streamcluster	point clustering
blackscholes	financial analysis

Inputs

Augmented or Replaced Existing Sets

x264	from Internet
bodytrack	augmented
swaptions	randomly generated
ferret	provided inputs
canneal	augmented (autogenerated)
streamcluster	from Internet
blackscholes	provided inputs

Metrics

Application Specific

x264	PSNR + Size
bodytrack	weighted relative difference
swaptions	relative difference
ferret	recall
canneal	relative difference
streamcluster	clustering metric
blackscholes	relative difference

Loop Perforation
(Quality Loss < 10%)

x264	**3.2x**
bodytrack	**6.9x**
swaptions	**5.0x**
ferret	**1.1x**
canneal	**1.2x**
streamcluster	**1.2x**

Loop Perforation
(Quality Loss < 10%)

x264	**3.2x**	motion estimation
bodytrack	**6.9x**	particle filtering
swaptions	**5.0x**	MC simulation
ferret	**1.1x**	image similarity
canneal	**1.2x**	simulated annealing
streamcluster	**1.2x**	cluster center search

Loop Perforation
(Quality Loss < 10%)

x264
bodytrack
swaptions
ferret
canneal
streamcluster

Tasks of most perforated loops:

- Distance metrics
- Search-space enumeration
- Iterative improvement
- Redundant executions

Main Observations

- **Approximate Kernel Computations**
  (have specific structure + functionality)

- **Accuracy vs Performance Knob**
  (tune how aggressively to approximate kernel)

- **Magnitude and Frequency of Errors**
  (kernels rarely exhibit large output deviations)

Approximate Program Analysis =

Accuracy + Safety

Accuracy and Guarantees

Logic-Based ***(worst-case)***

"for all inputs… "

Probabilistic ***(worst-case or average-case)***

"for all inputs, with probability at least p…"

"for inputs distributed as…"

Statistical ***(average-case)***

"for inputs distributed as… with confidence c"

"for tested inputs… with confidence c"

Empirical ***(typical-case)***

"for typical inputs…"

Green : Framework for Controlled Approximations (PLDI'10) *

End-to-end framework for controlled application on approximations

- Loop and function approximations

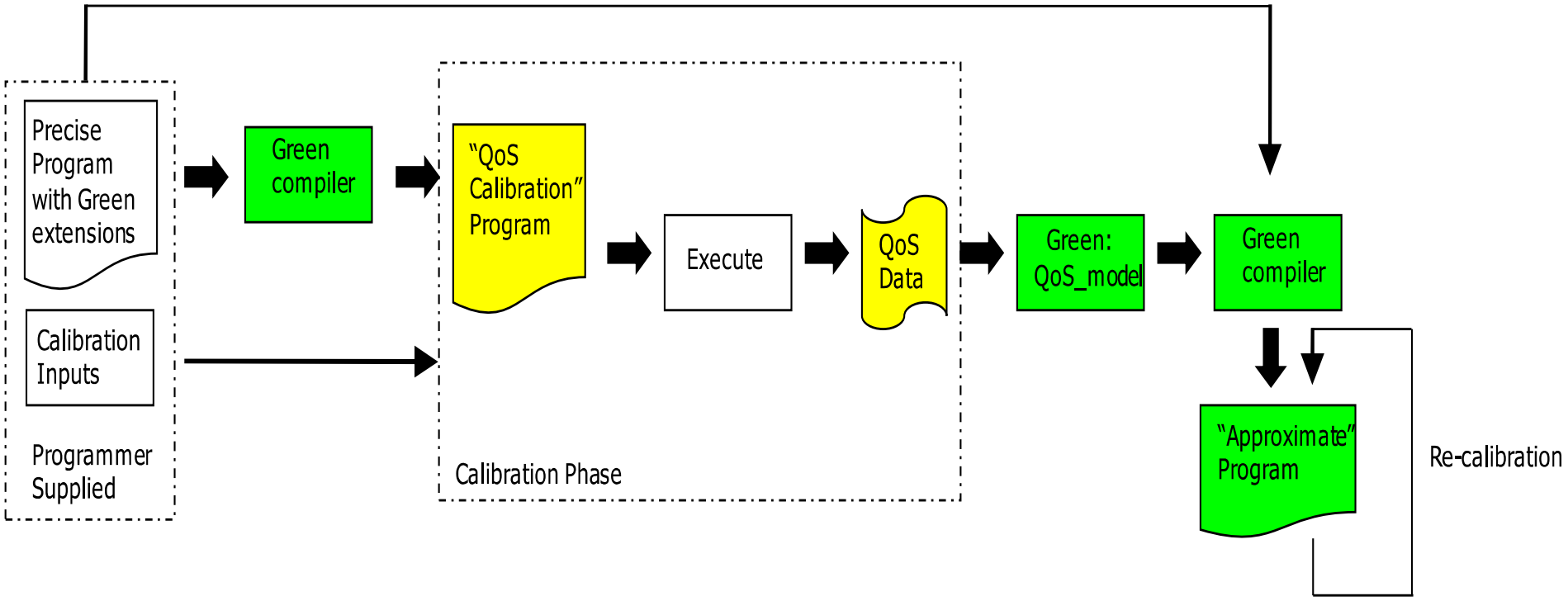
Relatively easy for programmers to use

Hooks for expert programmers and custom policies

Online mechanism to reactively adapt approximation policy to meet QoS

Adopted from slides by Radha Venkategiri

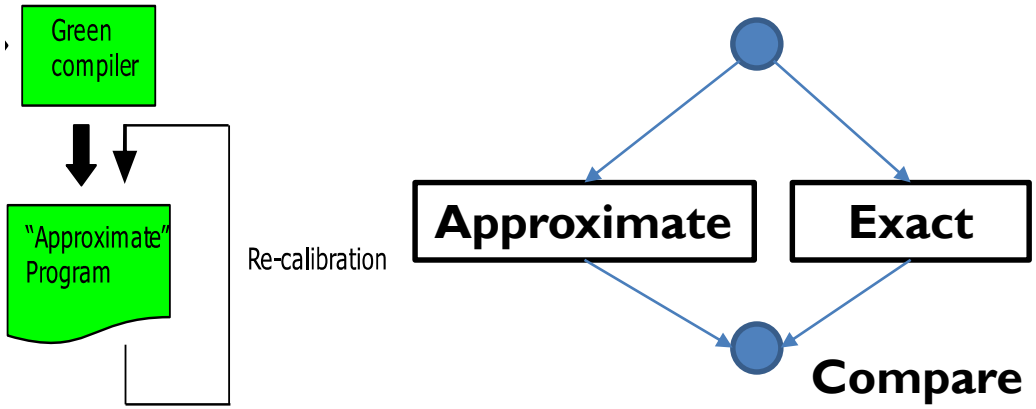
Green Framework

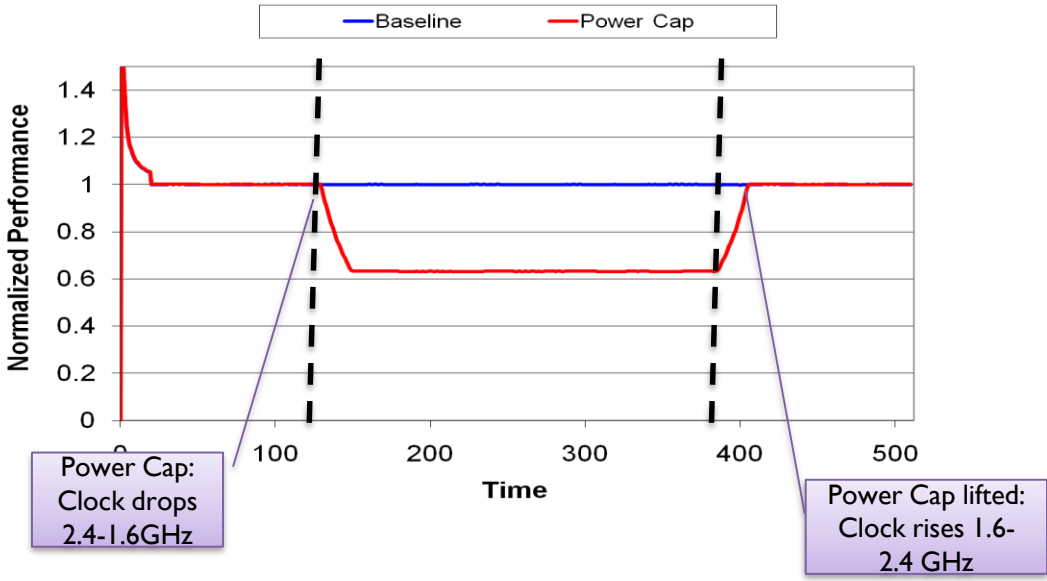


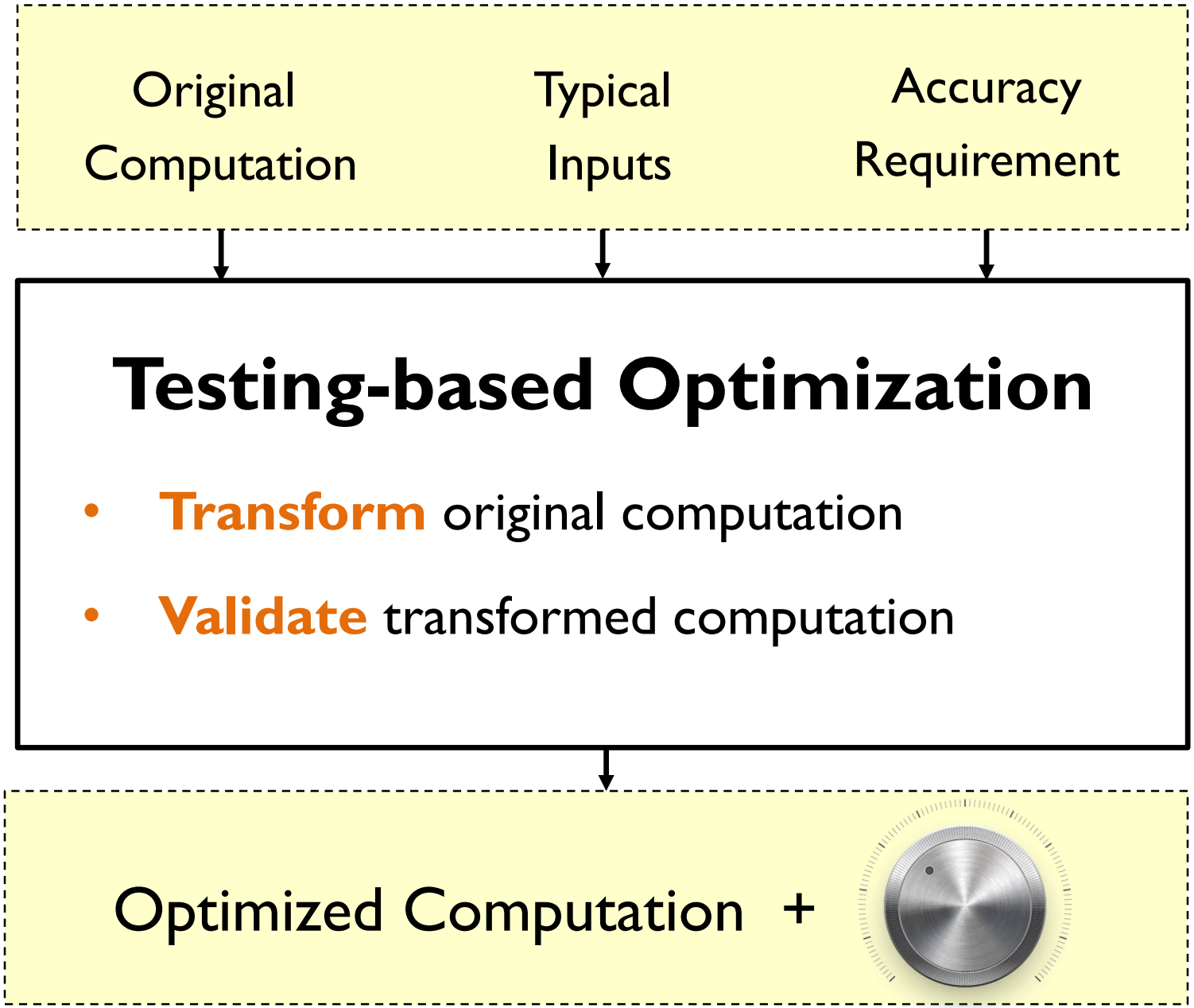
Goals of Runtime Adaptation

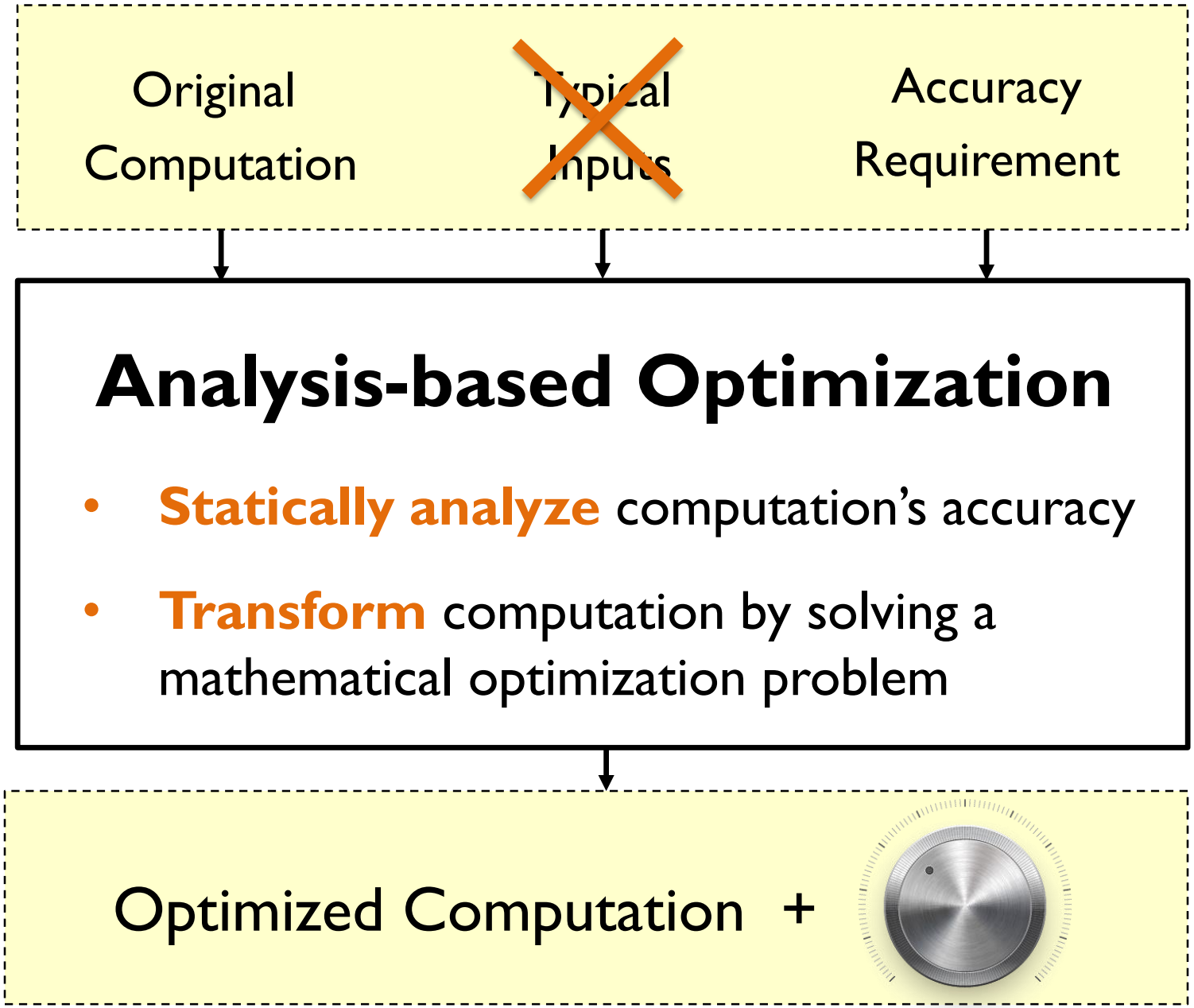
Accuracy (Green)

Time or Energy
(Loop perforation)






Original Computation
Typical Inputs
Accuracy Requirement
Testing-based Optimization
• Transform original computation
• Validate transformed computation
Optimized Computation +


Original Computation
Typical Inputs
Accuracy Requirement
Analysis-based Optimization
• Statically analyze computation's accuracy
• Transform computation by solving a mathematical optimization problem
Optimized Computation +

Approximate Program Safety:

Information-flow Type Systems

Relational Logic Reasoning

EnerJ Type System

Idea:

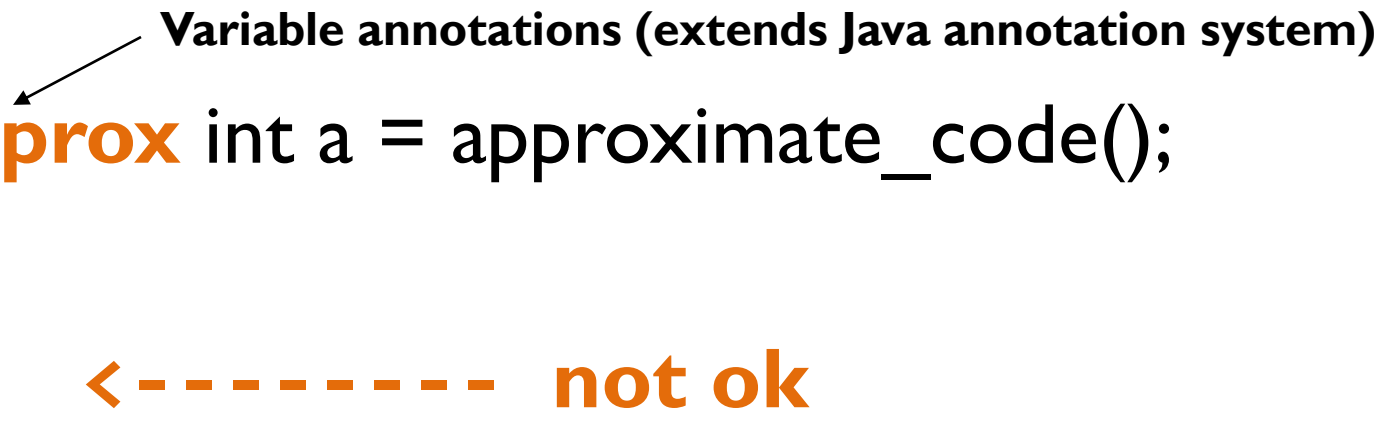
Isolate code and data that **must be precise** from those that **can be approximated**

Sampson, Dietl, Fortuna, Gnanapragasam, Ceze, Grossman
EnerJ: Approximate Data Types for Safe and General Low-Power Computation (PLDI 2011)

EnerJ Type System

Idea:

Isolate code and data that **must be precise** from those that **can be approximated**

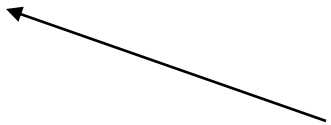


EnerJ Type System

Idea:

Isolate code and data that **must be precise** from those that **can be approximated**

```
@Approx int a = approximate_code();
int p;
if (a > 3) { p = 1; } else { p = 2; }
```



Control flow dependency (implicit flow)

EnerJ Type System

Idea:

Isolate code and data that **must be precise** from those that **can be approximated**

```
@Approx int a = approximate_code();
int p;
p = endorse(a);    <-------- ok
```

Like "(cast_type) a" in Java

EnerJ Type System

Consequence:

Then the approximate parts may be optimized automatically, but the developer needs to **ensure the endorsed values are valid**.

```
@Approx int a = approximate_code();
int p;
p = endorse(a);    <-------- ok
if ( isValid(p) ) { … } else { errorHandle(a) }
```

EnerJ Type System

Motivation:

Security information flow type systems –
prevent the program from leaking information about **private** variables into **public** variables.

Noninterference [Goguen and Meseguer 1982]:

"one group of users, using a certain set of commands is **noninterfering** with another group of users if the first group does with those commands can no effect on what the second group of users can see."

General Formal Reasoning About Relaxed Programs

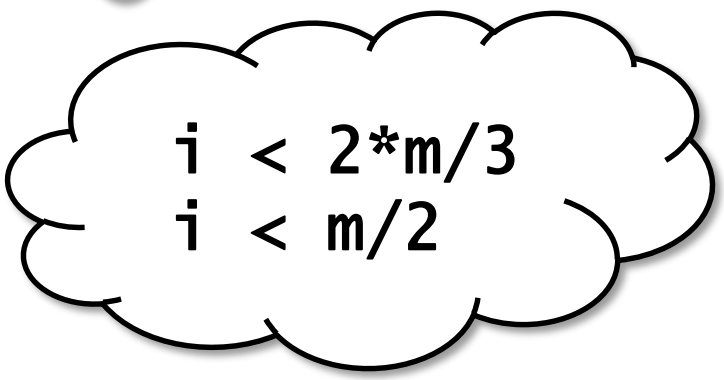
Carbin, Kim, Misailovic, Rinard
Proving acceptability properties of relaxed nondeterministic approximate programs (PLDI'12)

Carbin, Kim, Misailovic, Rinard
Verified integrity properties for safe approximate program transformations (PEPM'13)

Relational Safety Verification

```
for (i=0; i < m; i++) {
  sum = sum + x[i]
}

avg = sum / m
```



Carbin, Kim, Misailovic, Rinard
Proving acceptability properties of relaxed nondeterministic approximate programs (PLDI'12)
Carbin, Kim, Misailovic, Rinard
Verified integrity properties for safe approximate program transformations (PEPM'13)

Relational Safety Verification

```
relax (m) st (0 < m <= old(m))

for (i=0; i < m; i++) {
  sum = sum + x[i]
}

avg = sum / m
```

Relational Safety Verification

```
relax (m) st (0 < m <= old(m))

for (i=0; i < m; i++) {
  sum = sum + x[i]
}

avg = sum / m
```

Transformed execution accesses only (a subset of) memory locations that the original execution would have accessed

Relational Safety Verification

```
relax (m) st (0 < m <= old(m))

for (i=0; i < m; i++) {
  sum = sum + x[i]
}

avg = sum / m
```

The difference between the variable in the original and approximate runs is at most δ

$$|sum\langle o\rangle - sum\langle r\rangle| \leq \delta$$

Relative Safety

If the original program **satisfies all assertions**,
then the relaxed program satisfies all assertions

Relative Safety vs. Just Safety

Established **through any means**:
verification, testing, code review

If the original program **satisfies all assertions**,
then the relaxed program satisfies all assertions

Any inconsistent behavior must be in the original program!

Relative Safety vs. Just Safety

Established **through any means**:
verification, testing, code review

If the original program **satisfies all assertions**,
then the relaxed program satisfies all assertions

General Proofs: Mechanized in Coq [PLDI '12]

Pointer Safety: Automatic for loop perforation [PEPM '13]