CS 598sm

Probabilistic &
Approximate
Computing

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

Zoo:

SYSTEMS FOR ACCURACY-AWARE OPTIMIZATION

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 +

For **all** inputs

Original Computation

~~Typical Inputs~~

Accuracy Requirement

Analysis-driven Compiler

SAS '11, POPL '12, OOPSLA'13, OOPSLA '14, OOPSLA'19, CGO '20

- **Statically analyze** computation's <u>accuracy</u>
- **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

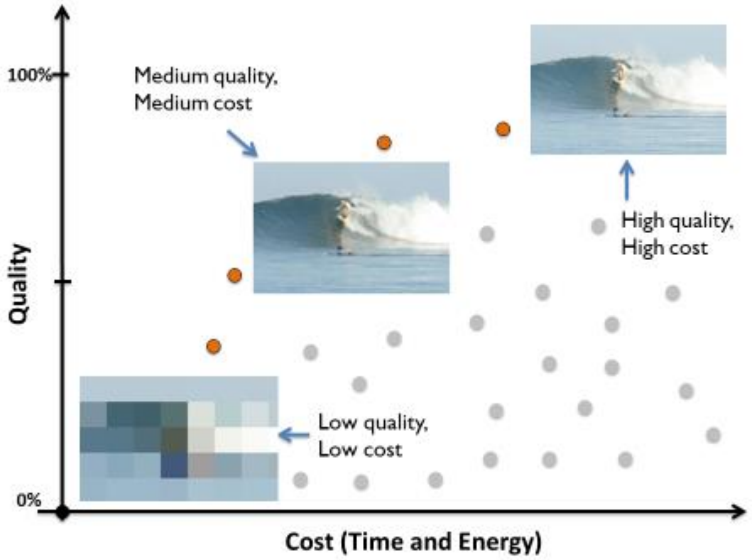
- Uses fitness function which orders the configurations

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.

Multiobjective Optimization (Reminder)

Functions to optimize are called ***objectives***

- Accuracy Loss – lower is better (or accuracy – higher is better)
- Speedup – higher is better (or normalized time – lower is better)
- Energy saving – higher is better (or consumption – lower is better)

They are the functions of program configuration – setting of knobs

Two candidate program configurations X and Y:

- X ***Pareto dominates*** Y if X is as good as Y in all objectives, and is better in at least one objective

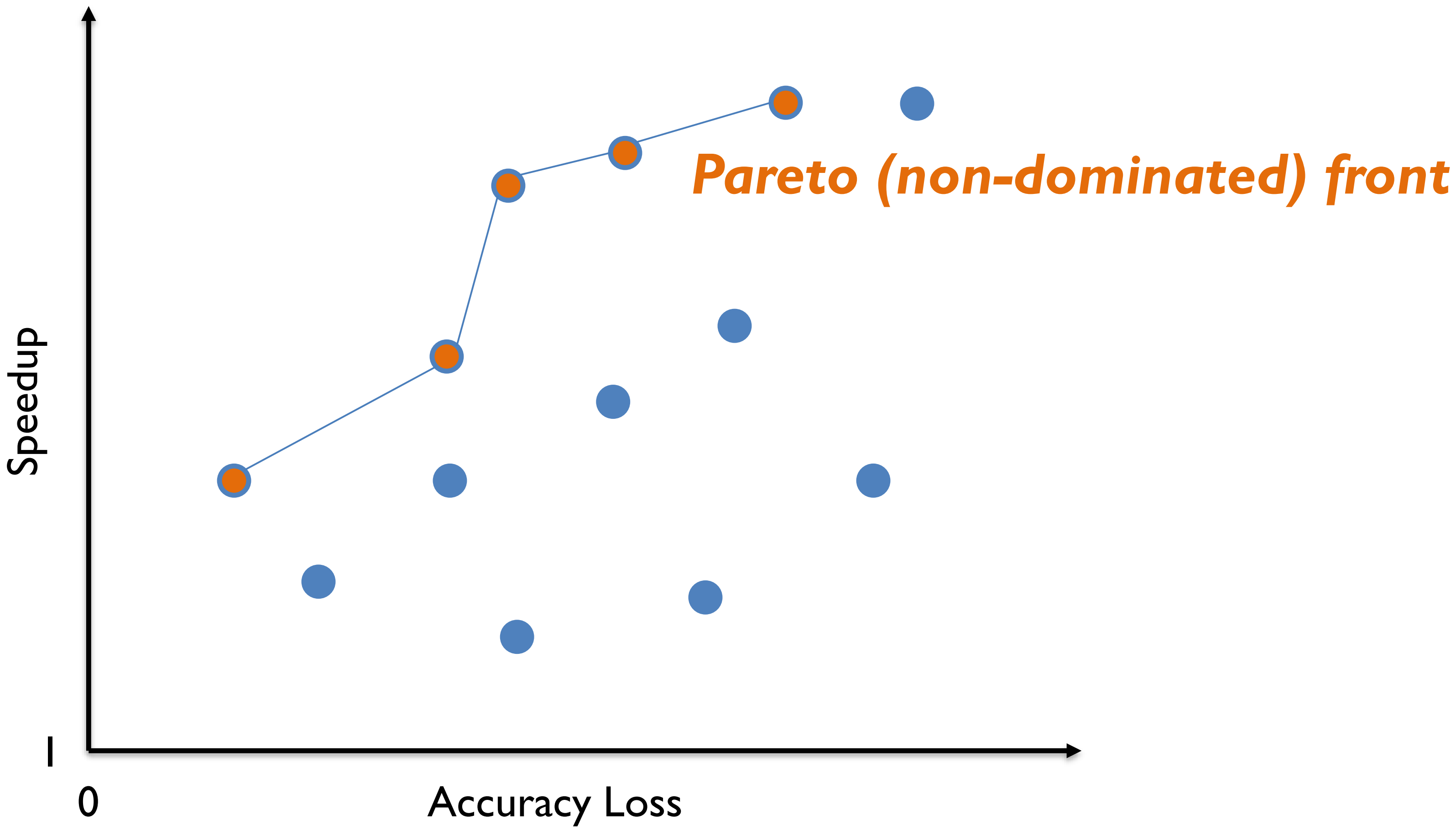
Pareto frontier: the set of points that are not dominated by other points

We will come back and formalize these notions later in the course!

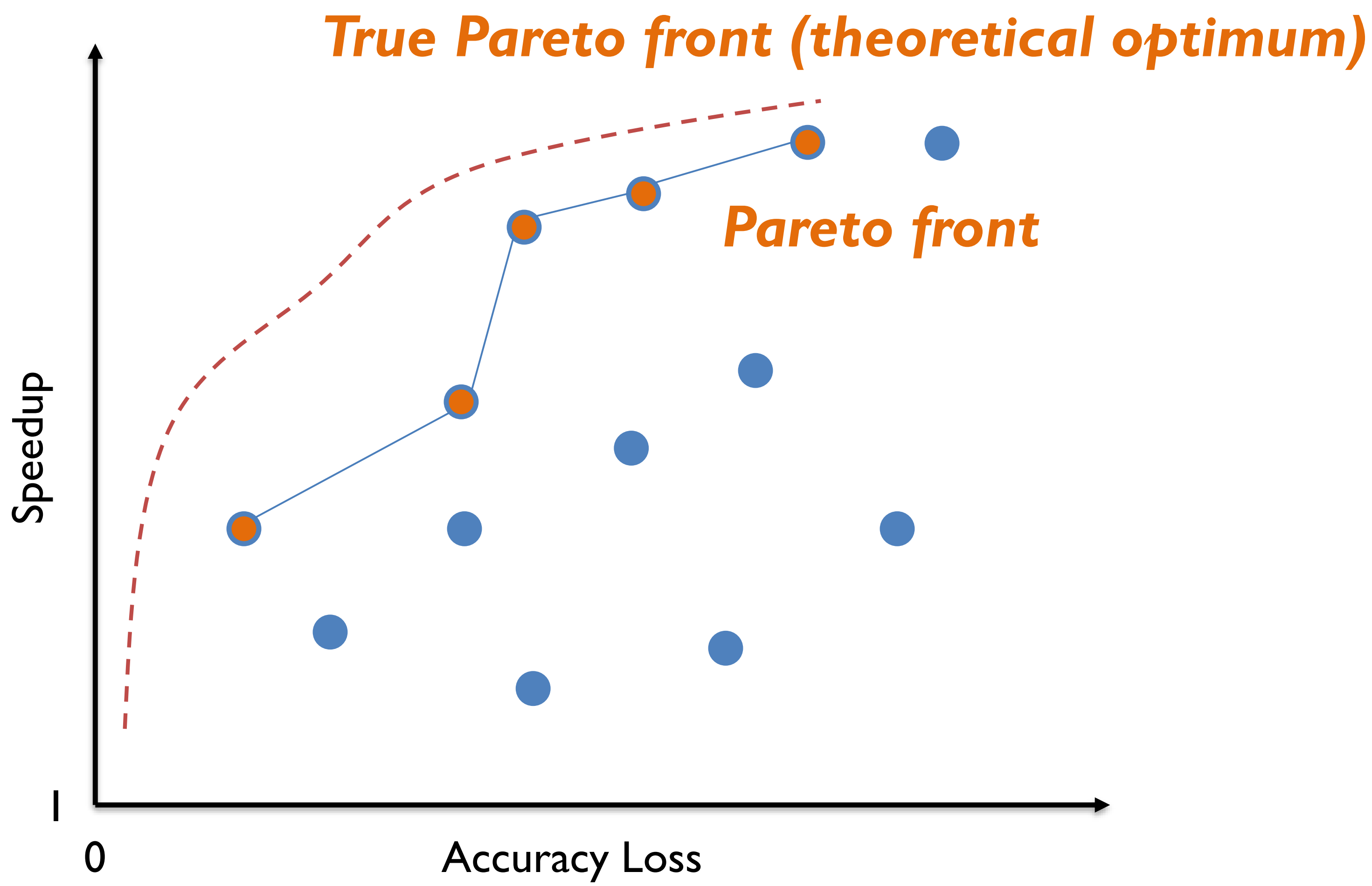
Example

Example

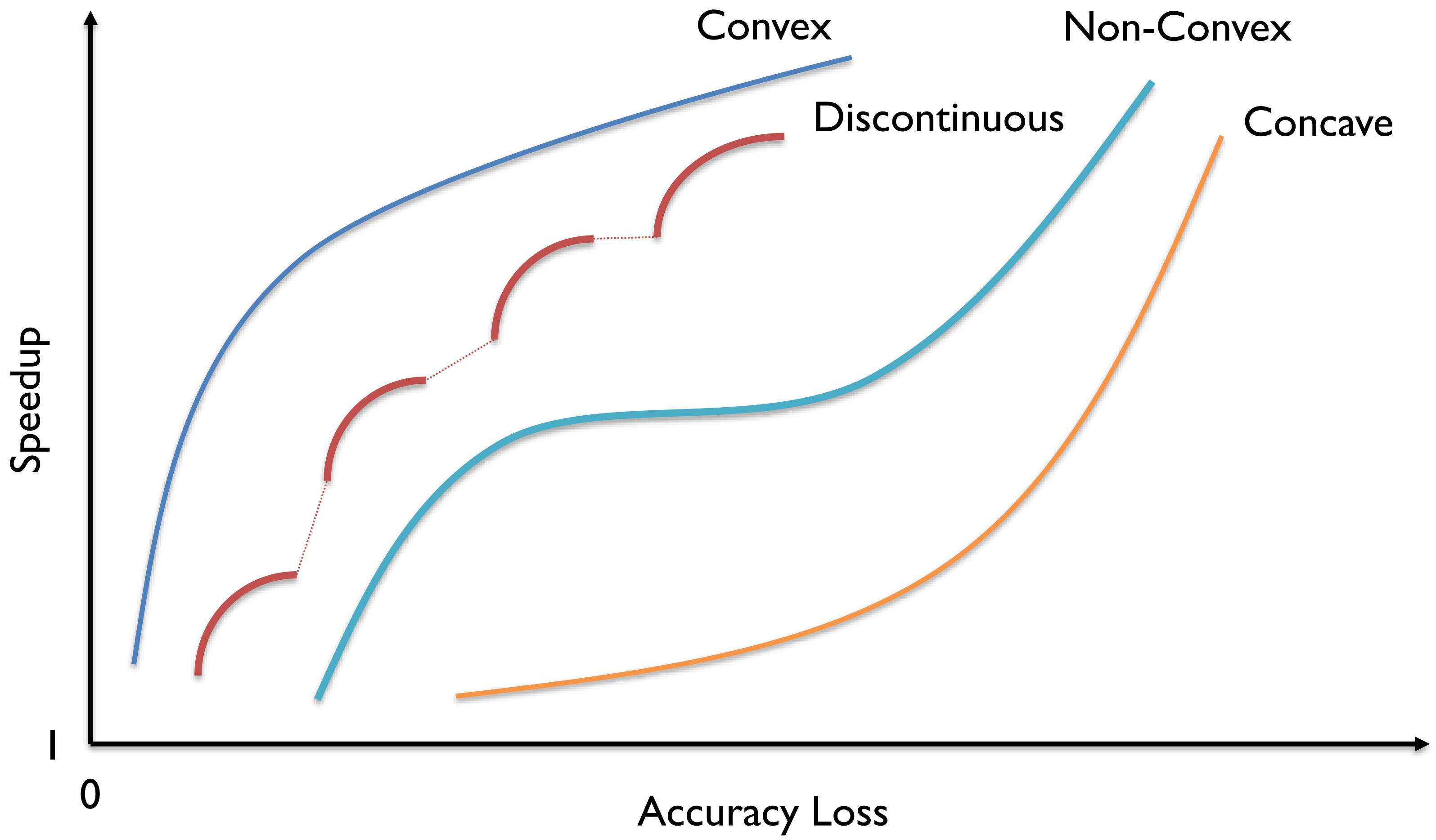
Example



Example


True Pareto front (theoretical optimum)
Pareto front
Speedup
1
0
Accuracy Loss

Pareto Fronts (aka Tradeoff curves)



A BIT OF FORMALISM

Based on Knowels, Thiele, Zitzler
A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers (2006)

Optimization Problem

Optimization Problem is a Quadruple $(X, Z, f, \preccurlyeq)$:

- X: decision space, and $x \in X$ is a ***decision vector***
- Z: objective space, and z $\in Z$ is a ***objective vector***
- $f: X \rightarrow Z$ is a function that assigns to each decision vector x an objective vector $z = \boldsymbol{f}(x)$
- We can think of it $z = (f_1, \dots, f_n) = \boldsymbol{f}(x_1, \dots, x_m)$ while assuming $Z = R^n$

- $\preccurlyeq$ is a binary relation over Z that defines a **partial order** of the objective space (it also induces a preorder on the decision space)

Weak Dominance

When n = 1 (single objective function):

- Optimization problem: $(X, \mathbb{R}, f, \preccurlyeq)$
- $\preccurlyeq$ is our good old $\leq$ on reals; there always exists a unique maximum

When n > 1 (multiple objective functions)

- Typically define $\preccurlyeq$ as $\mathrm{z}^{(a)} \preccurlyeq z^{(b)} \quad \equiv \quad \forall i \in \{1 \dots n\}\ z_i^{(a)} \leq z_i^{(b)}$
- Known as **weak Pareto dominance:** $\mathrm{z}^{(b)}$ weakly dominates $z^{(a)}$

Optimization Problem

Goal:
Find solution x^* that is mapped to a maximal element $z^* = \boldsymbol{f}(x^*)$ in the set $\boldsymbol{f}(X) = \{z \in Z \mid \exists x \in X : z = f(x)\}$

Think: x is ***program configuration***,
z is pair (accuracy, speedup), and
$\boldsymbol{f}$ computes (or records) accuracy and time of the execution.

- We can define the problem similar for searching minimal element (accuracy loss, run time)
- We can also make three dimensional tradeoff space accuracy, performance, energy, or even multidimensional

Our Optimization Problem

Select Program Configuration $X \in Configs$ to

$$\textbf{maximize}\ \big(Speedup(X,i), Accuracy(X,i)\big)$$
$$\textbf{forall}\ i \in InputSet$$

But these are most often competing objectives.

Consider turning into weighted single optimization problem ($w_{1,2}$ express preference):

$$\textbf{maximize}\ \ w_1 \times Speedup(X,i) + w_2 \times Accuracy(X,i)$$
$$\textbf{forall}\ i \in InputSet$$

To maintain accuracy guarantees rephrase: for every accuracy loss threshold δ

$$\textbf{maximize}\ Speedup(X,i)$$
$$\textbf{subject to}\ AccuracyLoss(X,i) \le \delta$$
$$\textbf{forall}\ i \in InputSet$$

Dominance

$z^{(a)} \preccurlyeq z^{(b)}$ for objective vectors of size n is defined as

$$\forall i \in \{1 \dots n\} \ . \ z_i^{(a)} \leq_{\mathbb{R}} z_i^{(b)}$$

It is also called weak Pareto dominance

A strong Pareto dominance $z^{(a)} \prec z^{(b)}$ is defined as above, but cannot have any element being equal.

Read:

- $z^{(a)} \preccurlyeq z^{(b)}$ we say that $z^{(b)}$ ***weakly dominates*** $z^{(a)}$
- $z^{(a)} \prec z^{(b)}$ we say that $z^{(b)}$ ***dominates*** $z^{(a)}$

We can similarly define this relation for the cases when we want to maximize one but minimize another objective.

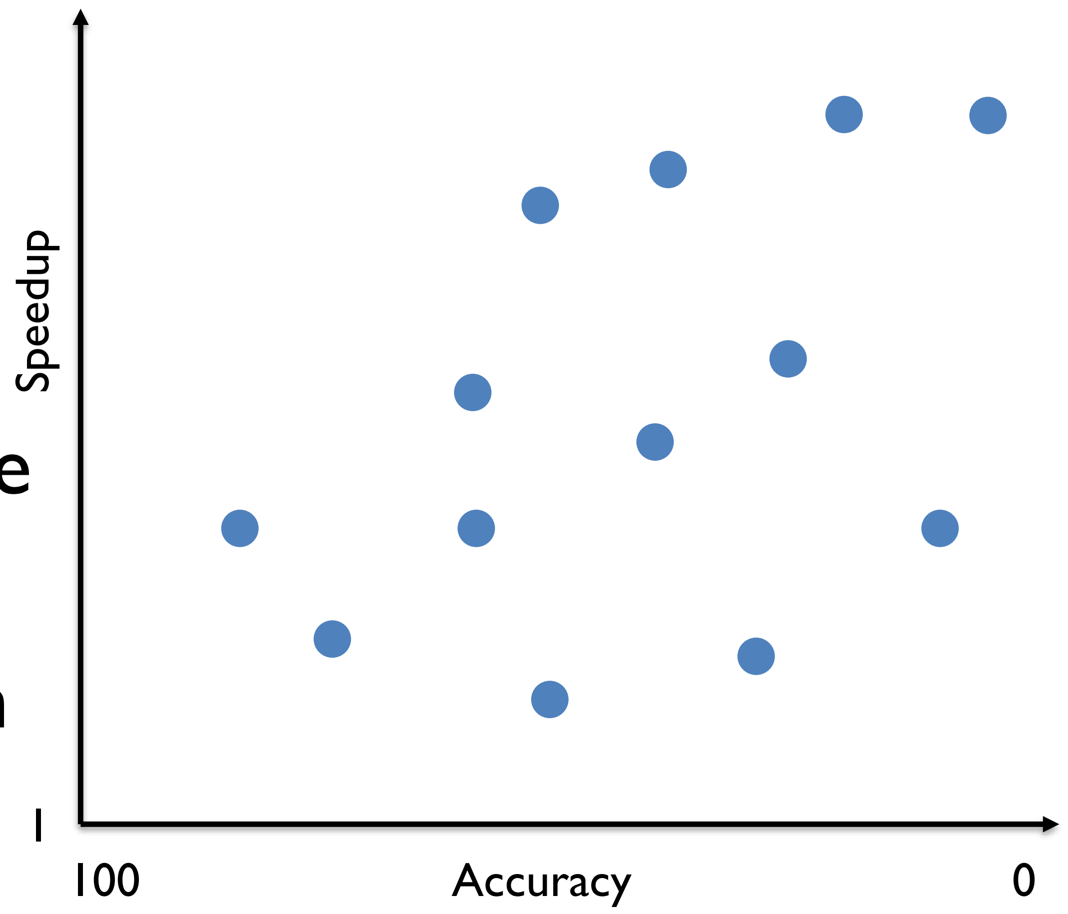
Dominance

We just learned about ***Pareto Dominance (and weak dominance)***

Incomparable points:
neither $z^{(a)} \preccurlyeq z^{(b)}$ nor $z^{(b)} \preccurlyeq z^{(a)}$

Indifferent: both points have the same value in all objectives

Strict domination: $z^{(a)}$ is better than $z^{(b)}$ in all objectives



Pareto Set Approximations

In optimization we are interested in the entire Pareto-optimal set, not just individual solutions

- The set comprises the non-dominated objectives and decisions:*
  $A = \{(z, x) \ \mid \exists x \in X \ \exists z \in Z \ s.t. \ z = f(x) \text{ and z in not dominated}\}$
- We want to find mutually incomparable solutions
- Each such solution is a Pareto set approximation

We can extend the optimization problem: we want the best set of Pareto points (over other sets)

- Think: we want the best tradeoff curve across all that can be computed

* With a small abuse of notation, $z_B \in B$ refers to $(z_{B,} x_B) \in B$ for some x_B but the decision vector x_B is not necessary in this context; Alternatively, one could write $(z_{B,}, \_) \in B$. We treat the case $x_B \in B$ the same way.

Comparing Pareto Sets Approximations

Let A and B the sets of Pareto-optimal points (e.g., produced by different search algorithms or multiple runs of a randomized algorithm)

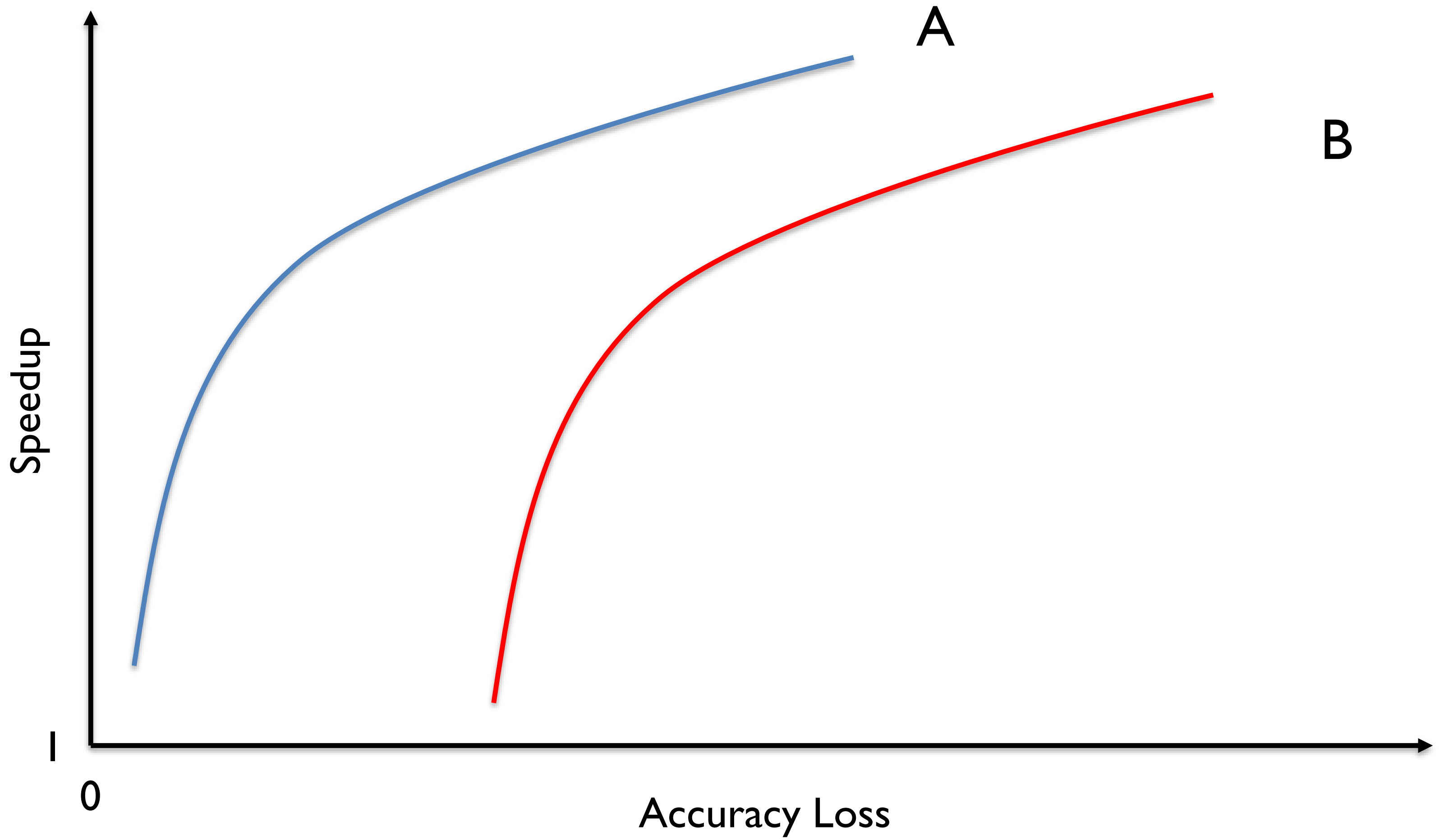
Is this enough? Typically no, we may need to define **quality indicators** to compare 'incomparable' sets

- There is no standard quality indicator, but needs to be selected based on context

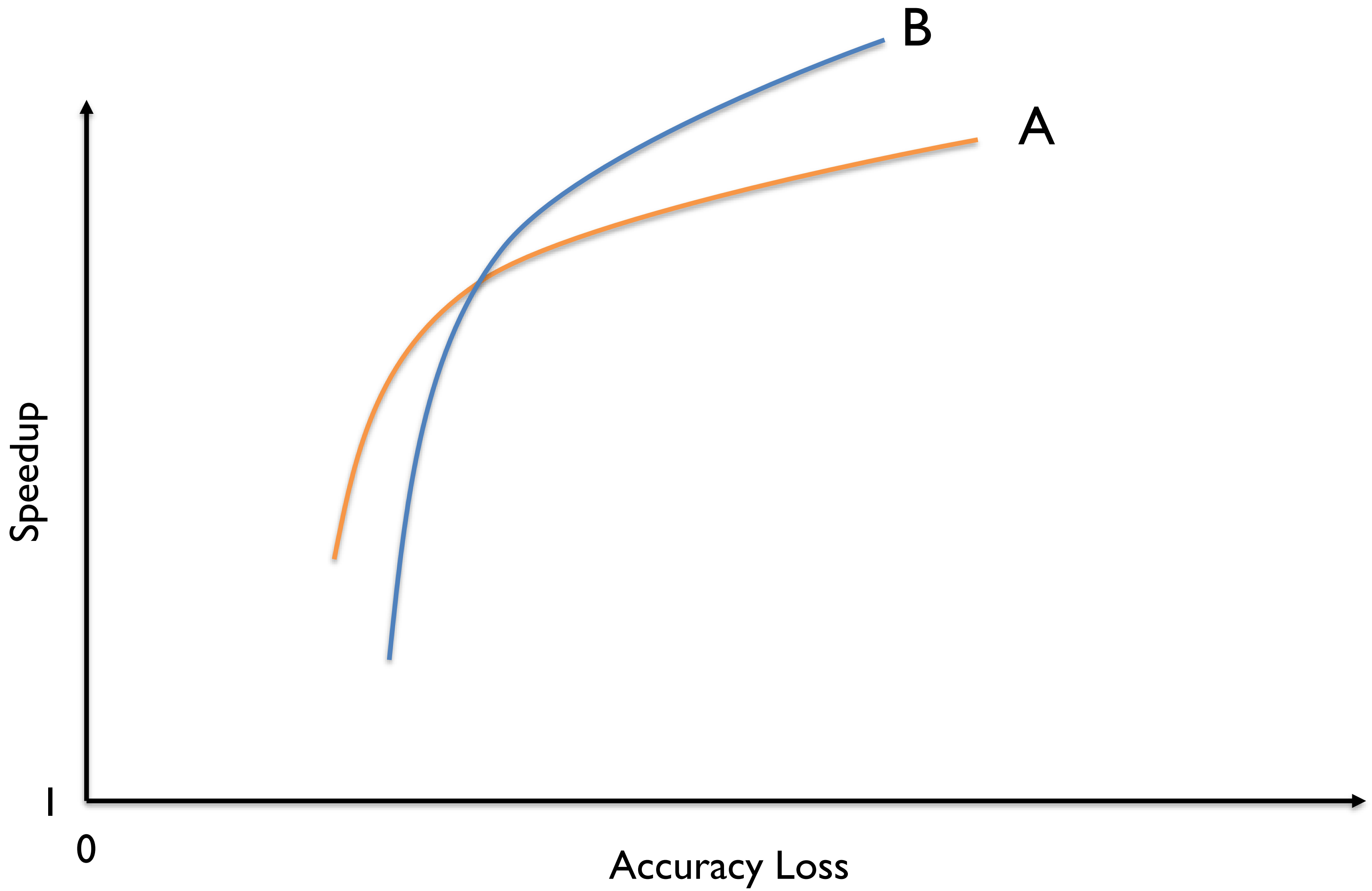
Hypervolume Indicator: intuitively, a volume (in our case area) of dominated solutions covered by the Pareto set.

- Need to select a reference point (or points). In our case, (speedup,accuracy) pairs (1.0, 100%) and (1.0, max-acceptable-accuracy) are intuitive choices
- Can order the Pareto sets $I(A) > I(B) \Rightarrow A \rhd B$ (i.e., A is better than B)
- For randomized search algorithms, can compute and compare expected indicators i.e., $\mathbb{E}\, I(A) > \mathbb{E}\, I(B) \Rightarrow A \rhd B$

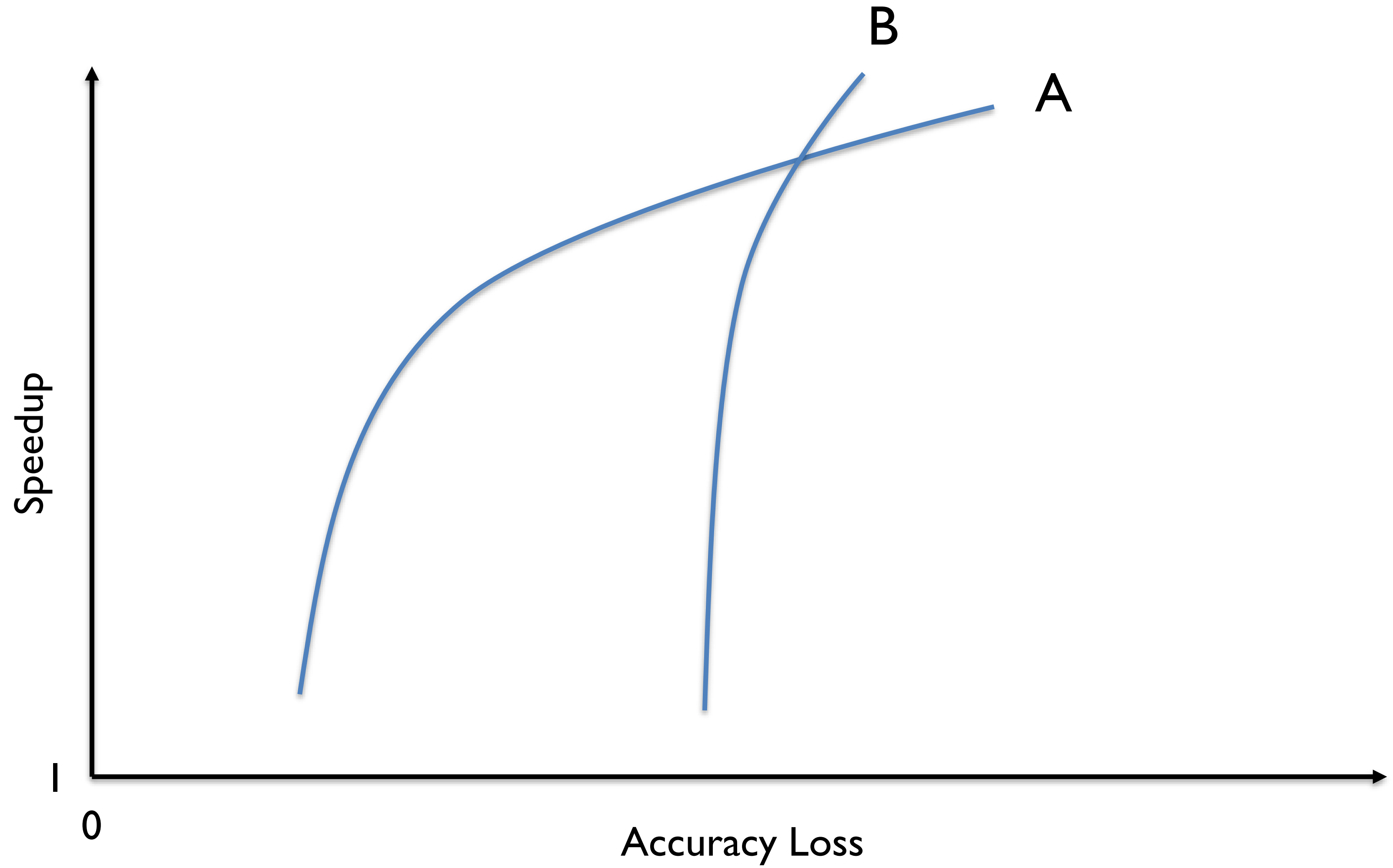
Pareto Fronts (aka Tradeoff curves)



Pareto Fronts (aka Tradeoff curves)



Pareto Fronts (aka Tradeoff curves)



Comparing Pareto Sets Approximations

Let A and B the sets of Pareto-optimal points (e.g., produced by different search algorithms or multiple runs of a randomized algorithm)

We can define the relations for the sets:

- **A dominates B** ($B \prec A$) iff every $z_B \in B$ is dominated by some $z_A \in A$
- **Weak domination** ($B \preccurlyeq A$) is defined similarly
- **A and B are indifferent:** A weakly dominates B and B weakly dominates A
- **A is better than B** ($B \lhd A$): every $z_B \in B$ is weakly dominated by at least one $z_A \in A$ and A and B are not indifferent
- **A and B are incomparable:** neither set weakly dominates the other
- Is this enough? Typically no, we may need to define quality indicators to compare 'incomparable' sets.

Note on Our Optimization

Since we execute the programs, the input distributions will impact the approximation sets

Alternatively, if we combine with static analysis, some of the tradeoffs will end up being conservatively set

The search algorithms (e.g., auto-tuners) will impact what solutions we find – especially if they are randomized

The distribution between the ‘training’ and ‘test’ inputs may change, impacting accuracy and performance

LET'S START WITH PRACTICE THEN

Petabricks

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

Precusor to OpenTuner (popular autotuner: http://opentuner.org)

> 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

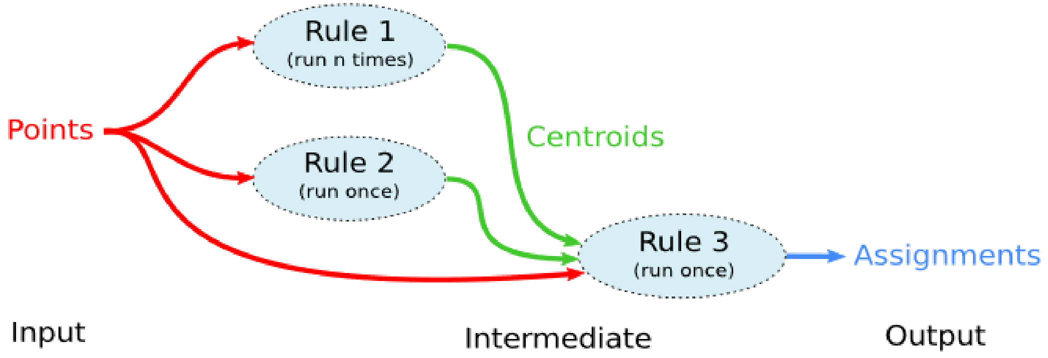
Petabricks Autotuner

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

```
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));
  }
}
```

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

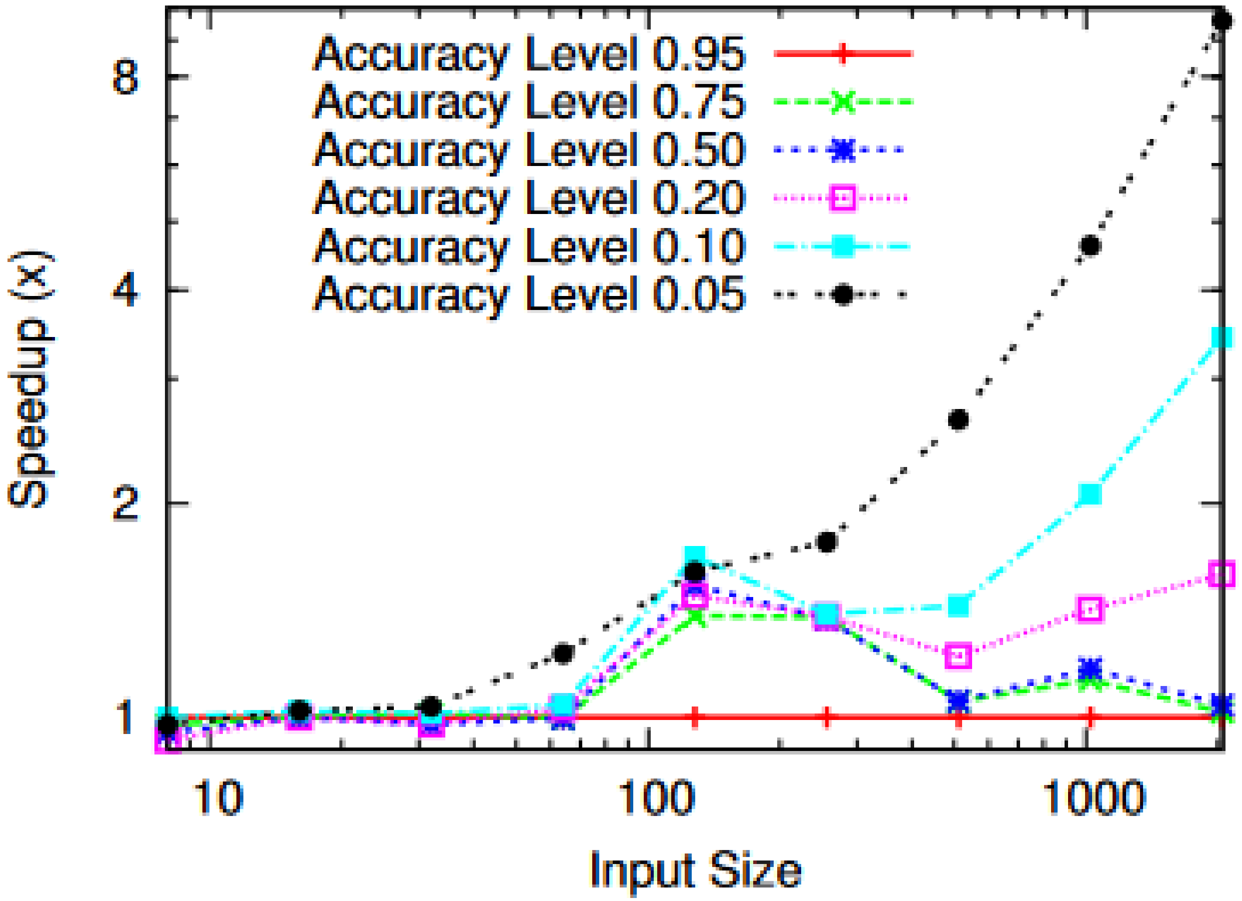
```
// Rule 1:
// One possible initial condition: Random
// set of points
to(Centroids.column(i) c) from(Points p) {
  c=p.column(rand(0,n))
}

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



Petabricks Autotuner

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

```
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?

Let a compiler find and expose some by modifying the program!

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%**

Phases of Approximate Compiler:

Find perforatable loops

- **Identify Opportunity:** Run **performance** profiler

  Identify time consuming loops

- **Sensitivity Testing:** Perforate **one loop** at a time

  Filter out loops that do not satisfy accuracy requirement

- **Search for Optimal Knobs:** Perforate **multiple loops**

  Find combinations of loops that maximize performance
  Return a tradeoff curve of best solutions found

Validate Perforated Loops

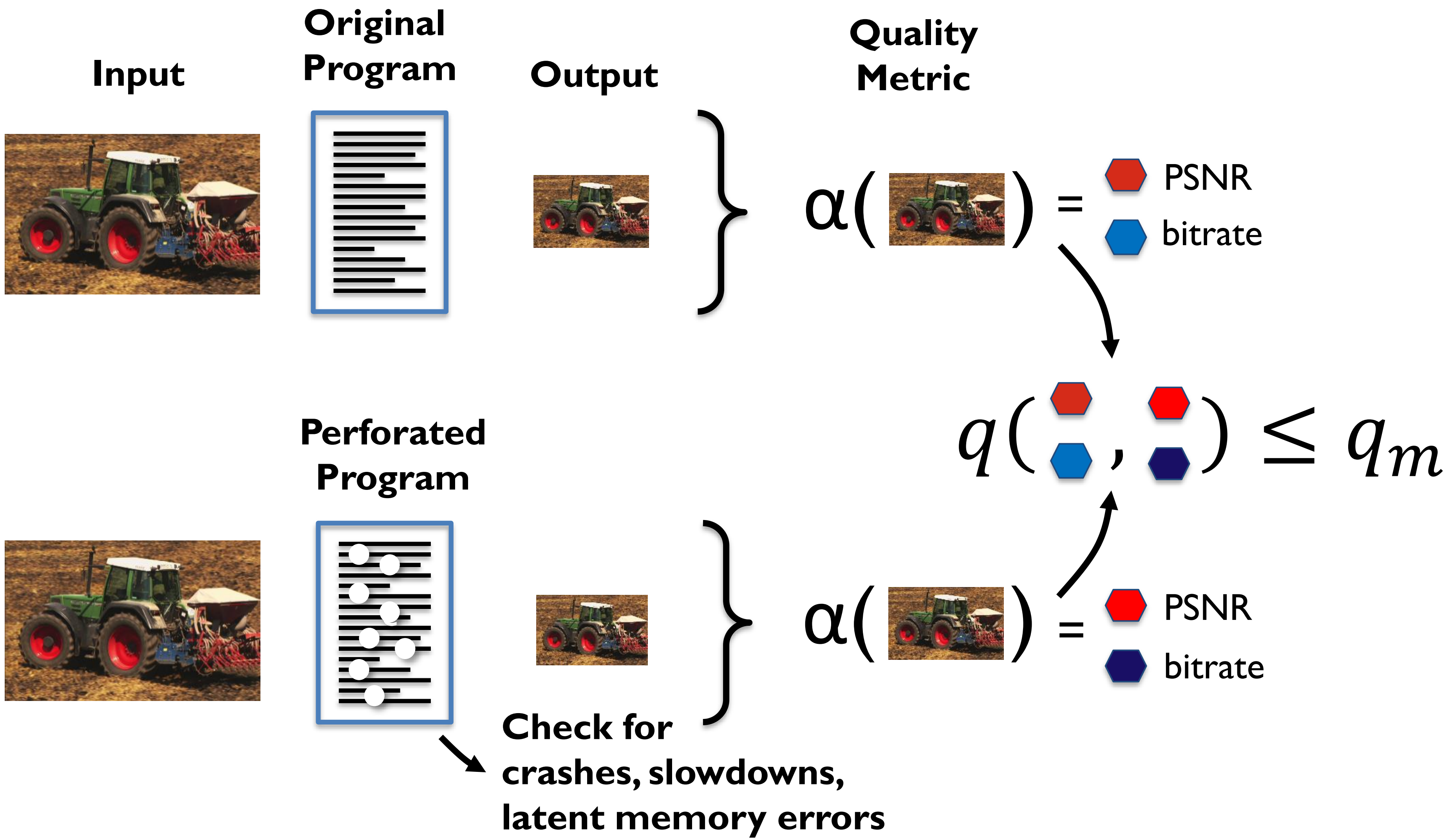
Filter out loops that do not satisfy requirement

Criticality (Sensitivity) 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)

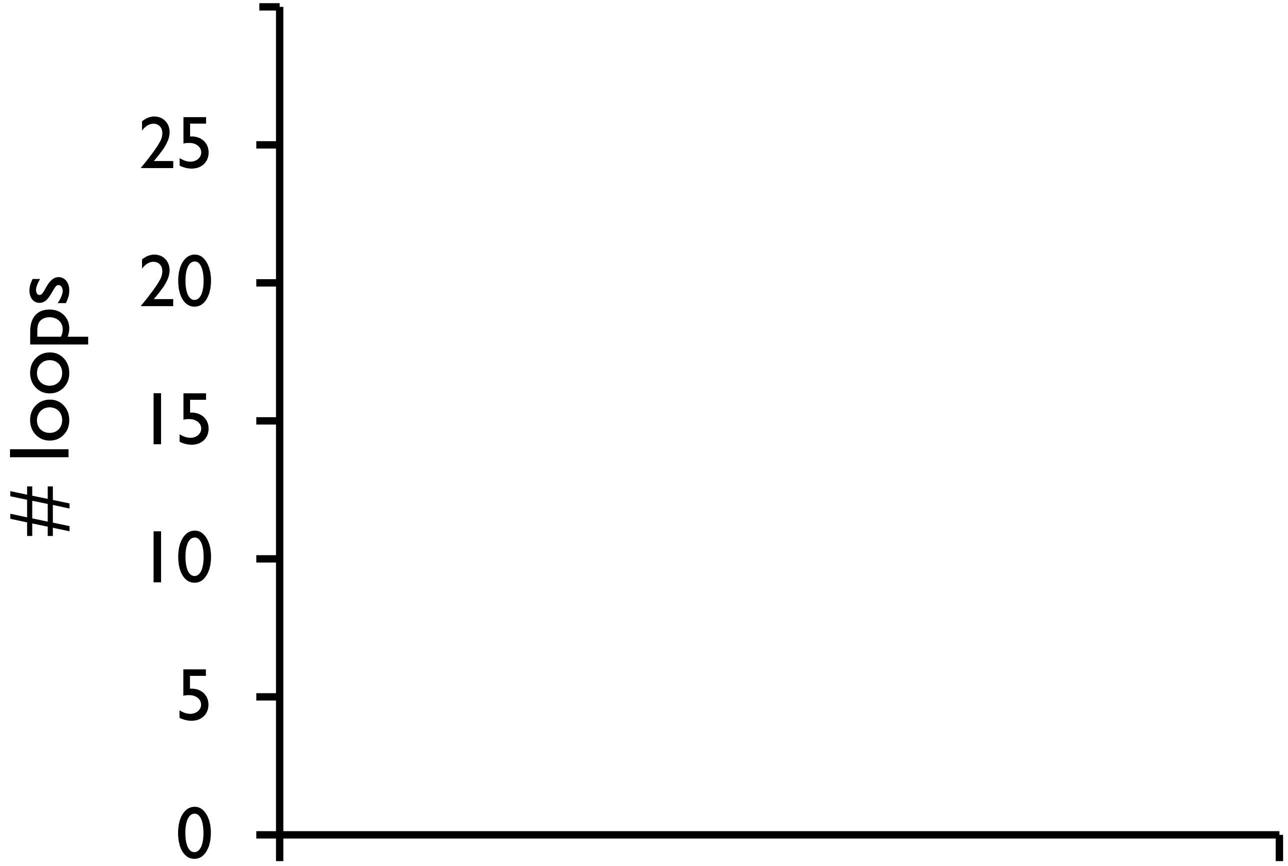
Criticality (Sensitivity) Testing:

Filter out loops that do not satisfy requirement



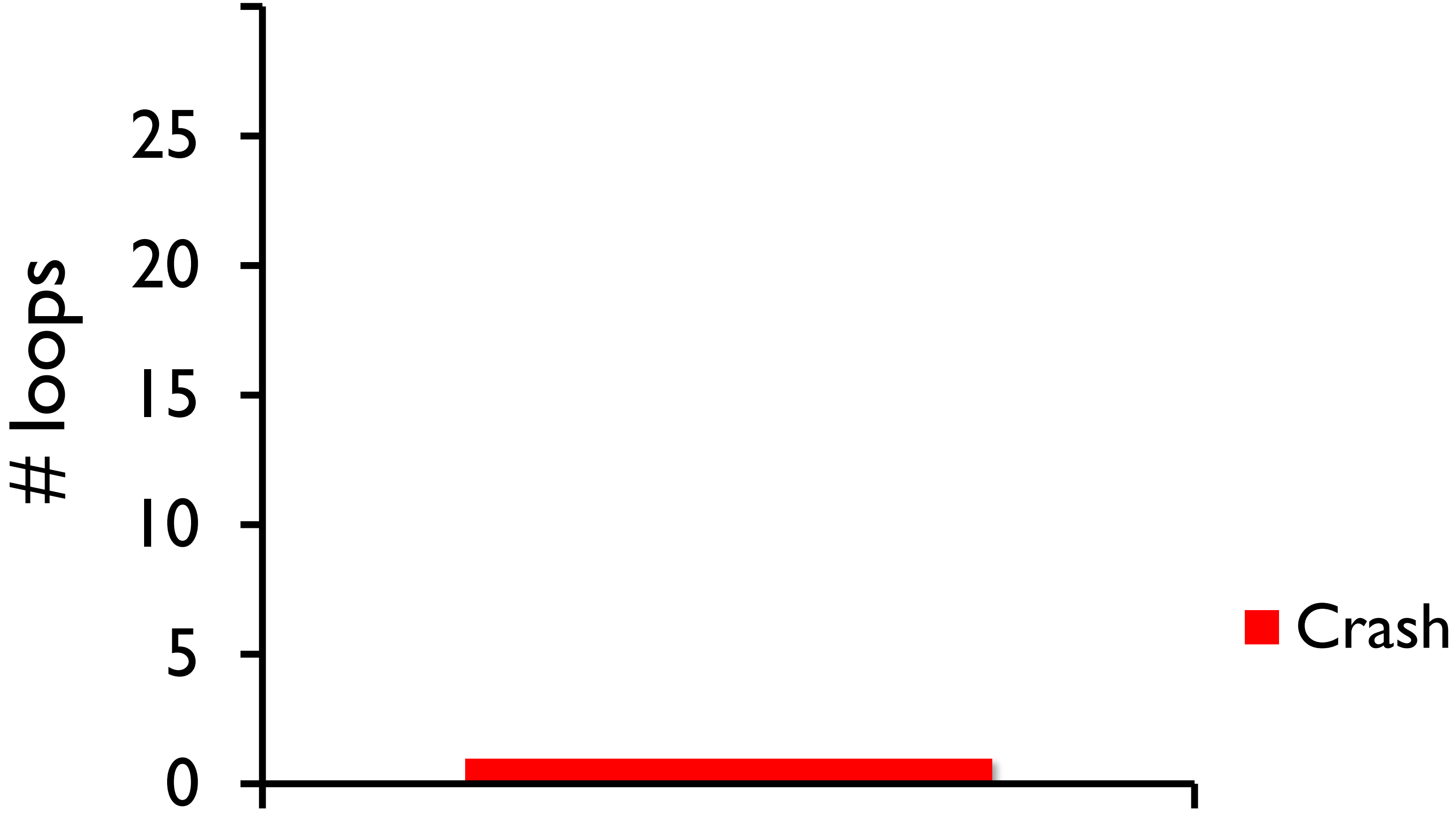
Perforating Individual Loops in x264

(Quality Loss < 0.1)



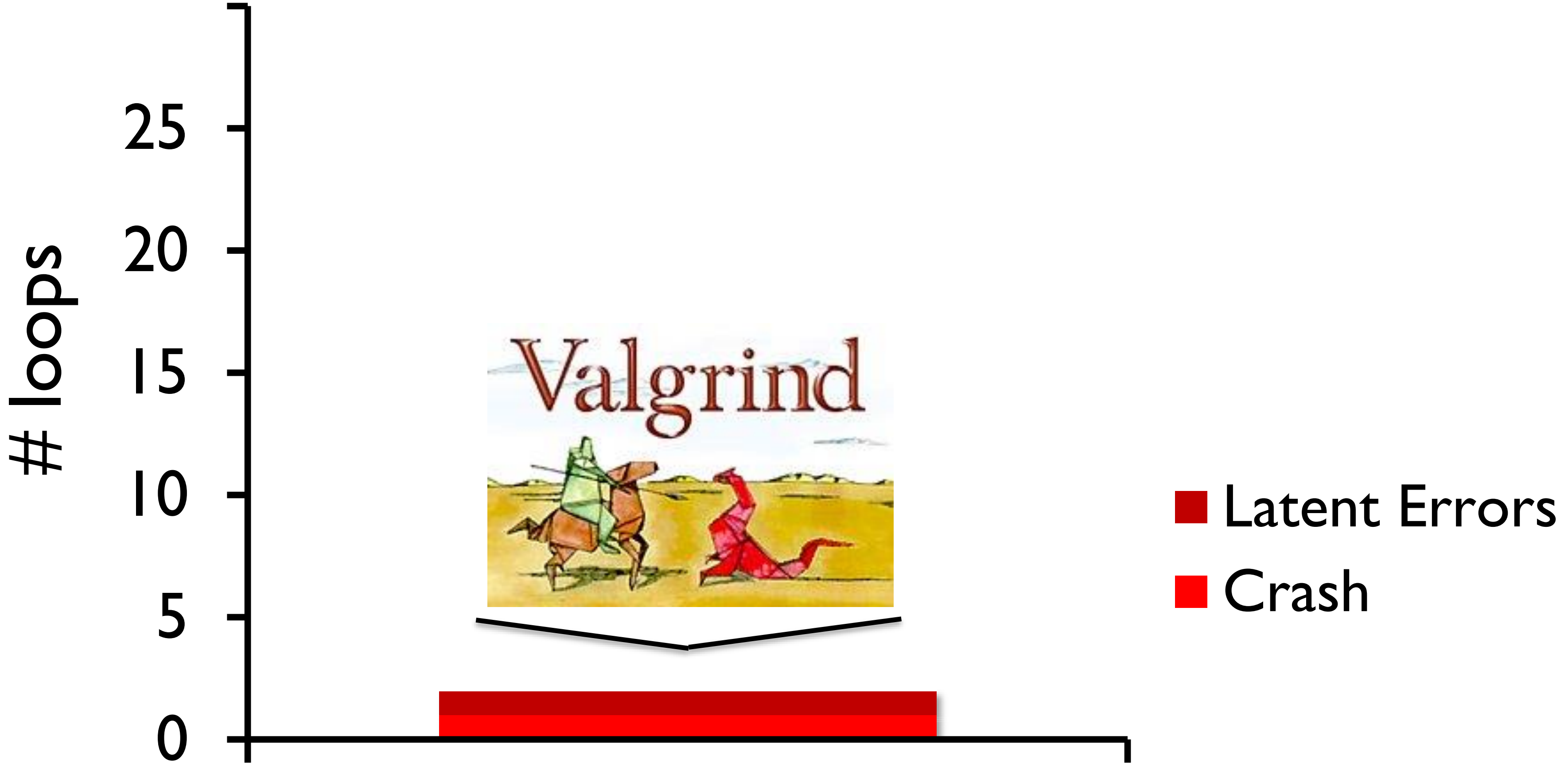
Perforating Individual Loops in **x264**

(Quality Loss < 0.1)



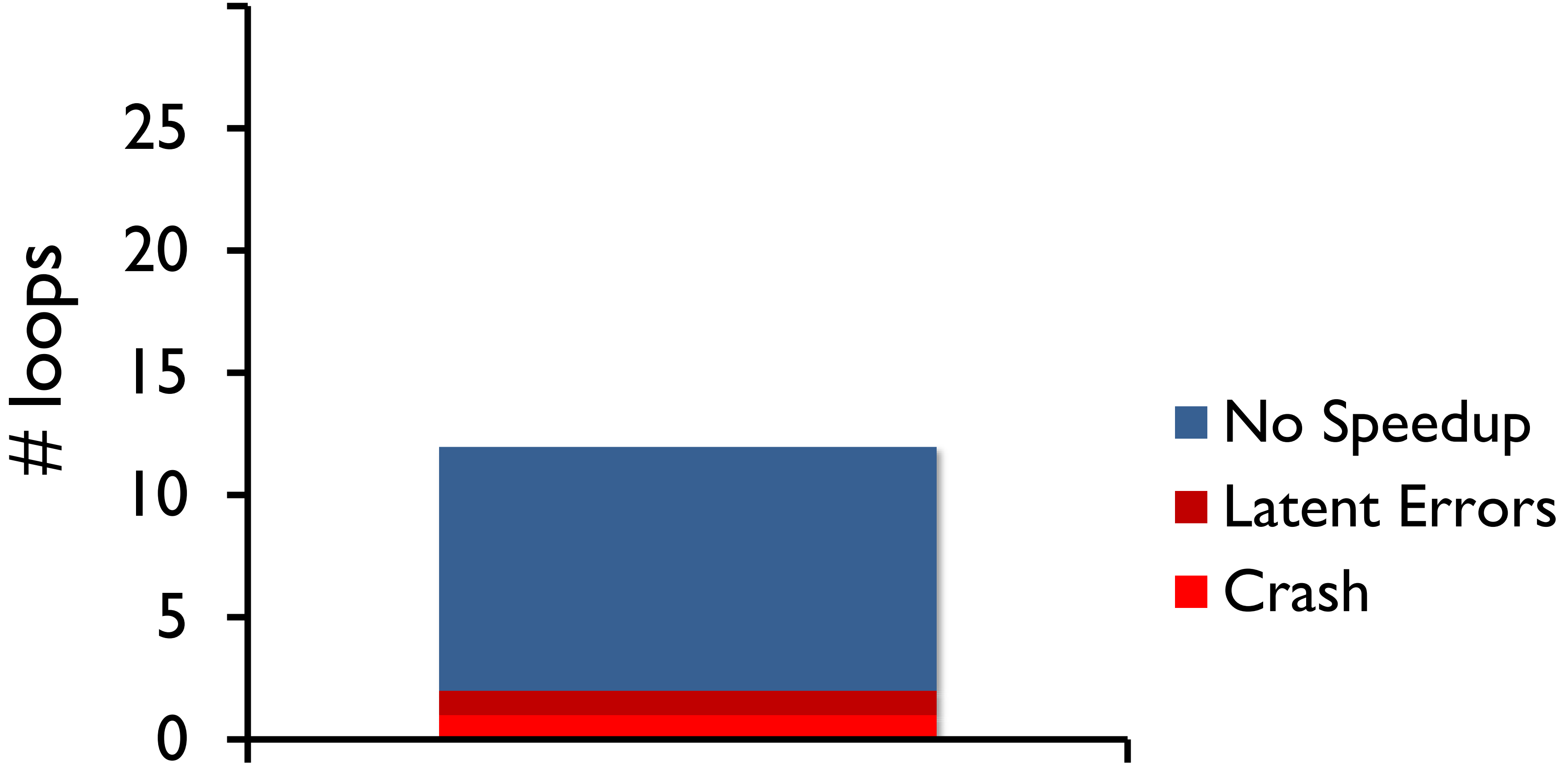
Perforating Individual Loops in x264

(Quality Loss < 0.1)



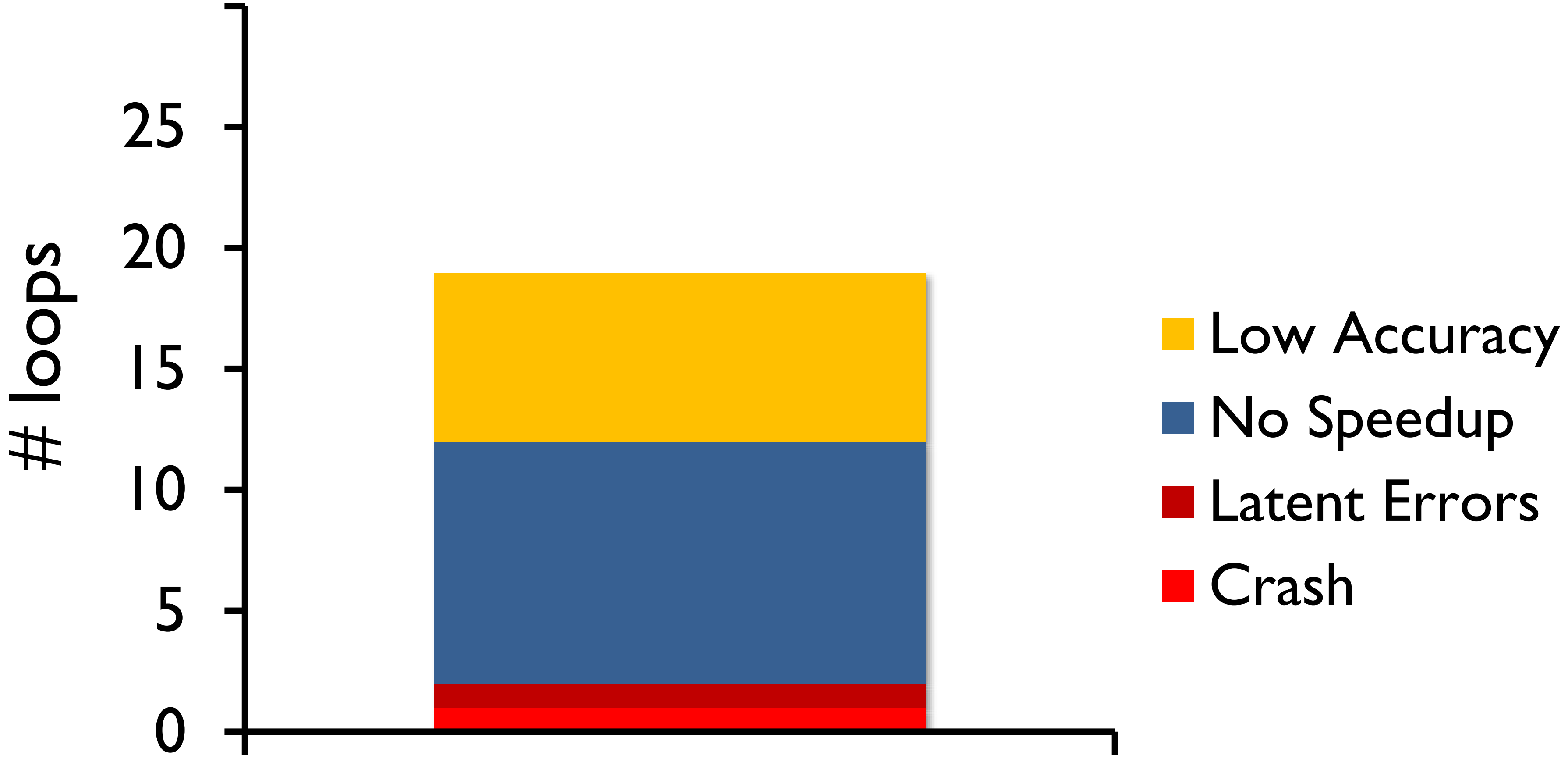
Perforating Individual Loops in x264

(Quality Loss < 0.1)



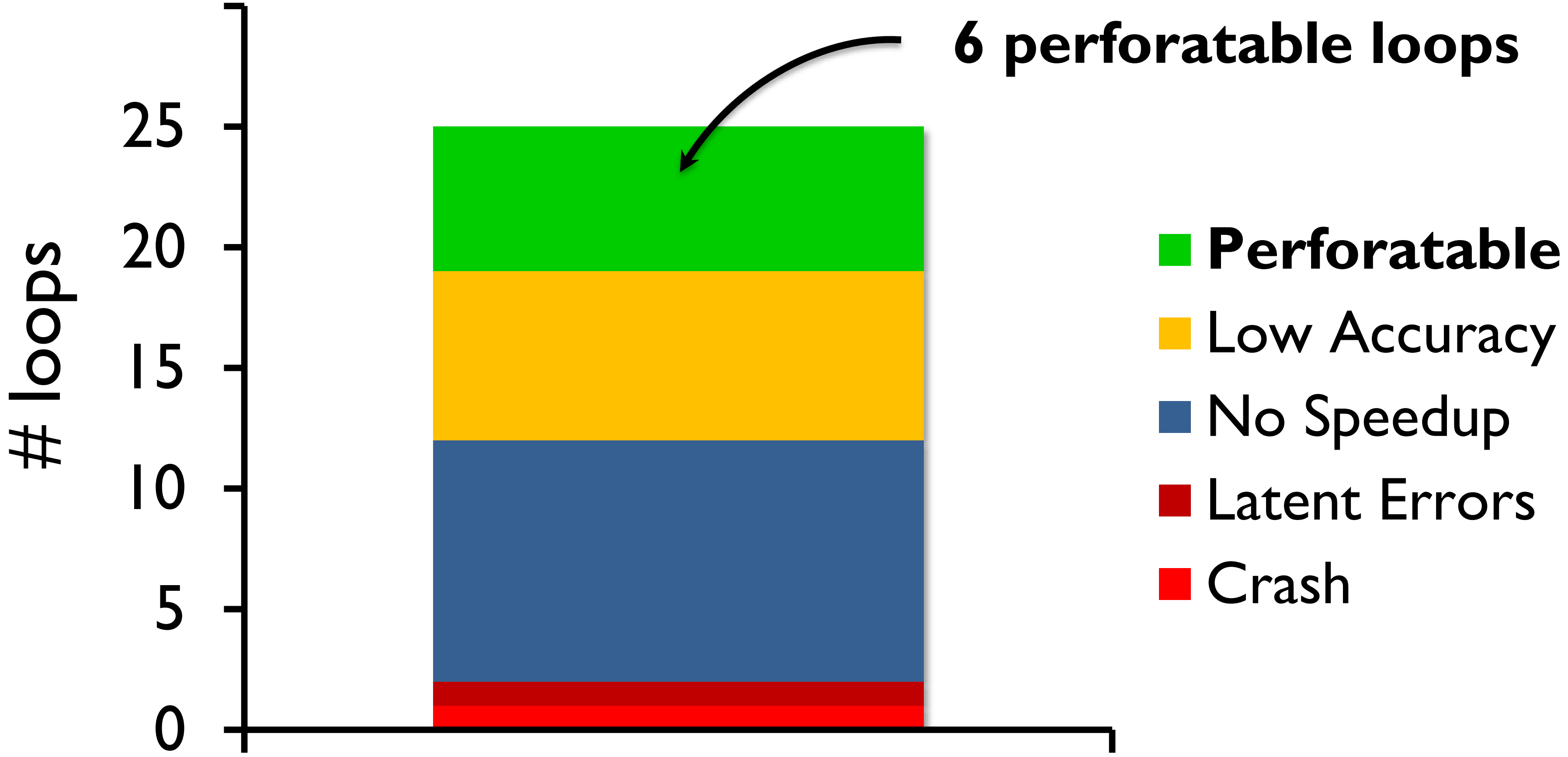
Perforating Individual Loops in x264

(Quality Loss < 0.1)



Perforating Individual Loops in x264

(Quality Loss < 0.1)



Status

We found approximate computations and exposed individual knobs

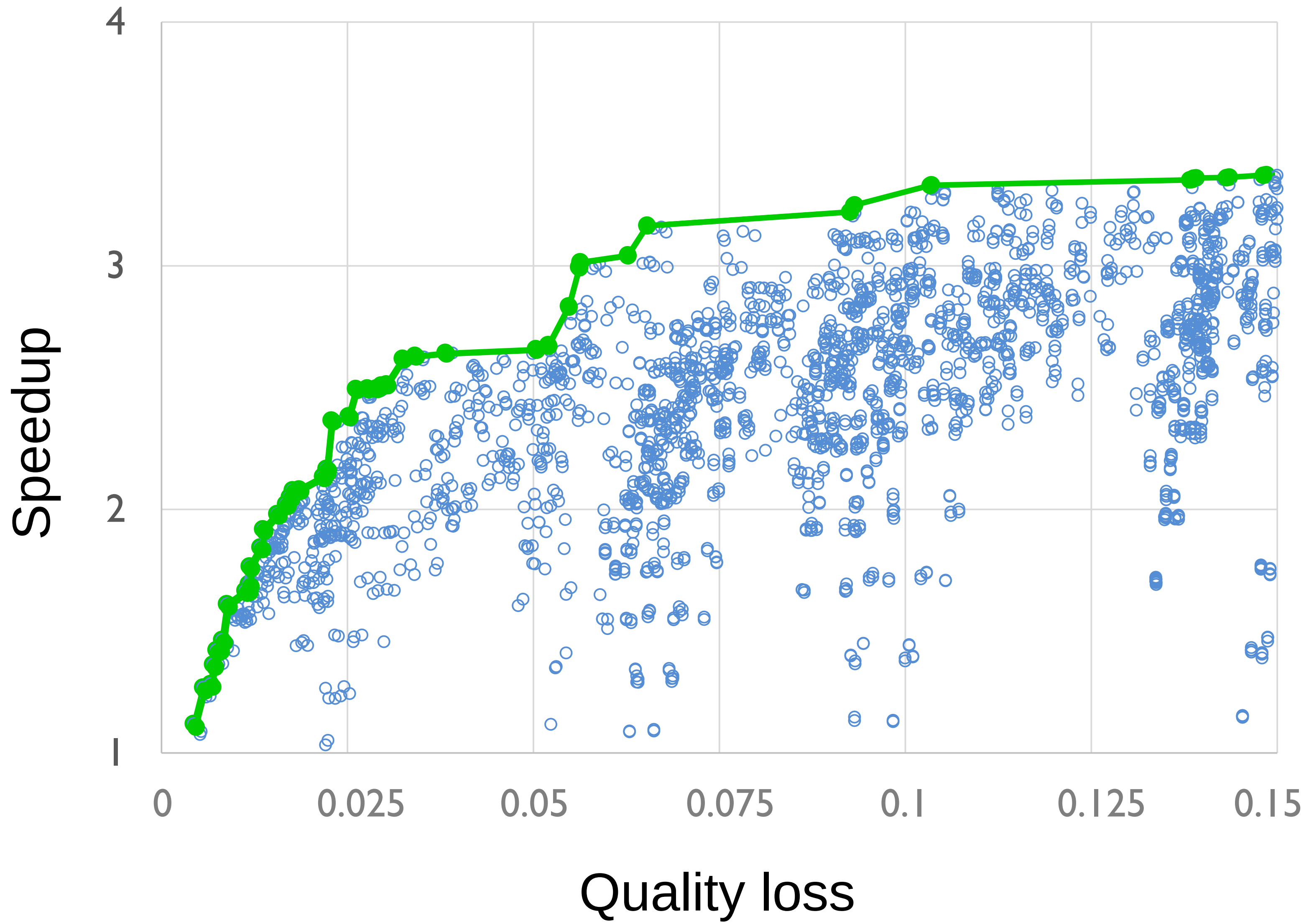
Next, let us combine the knob values to utilize the approximation "budget"

Search Strategies and Algorithms

- Greedy
- Exhaustive
- Combined
- Hill-climbing
- Simulated annealing
- Genetic algorithm
- Reinforcement learning
- …

- We had the comfort to do a bounded-exhaustive evaluation to explore the tradeoff space

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”

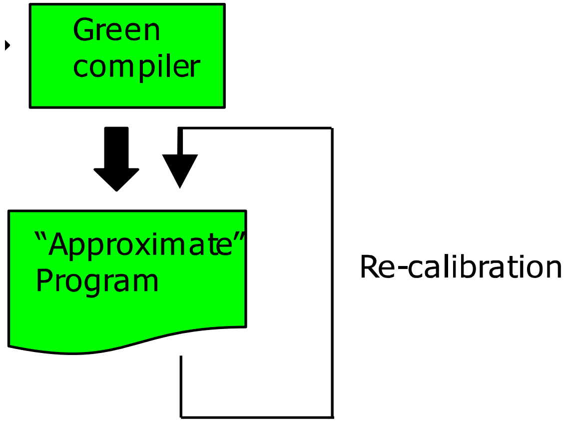
Empirical ***(typical-case)***

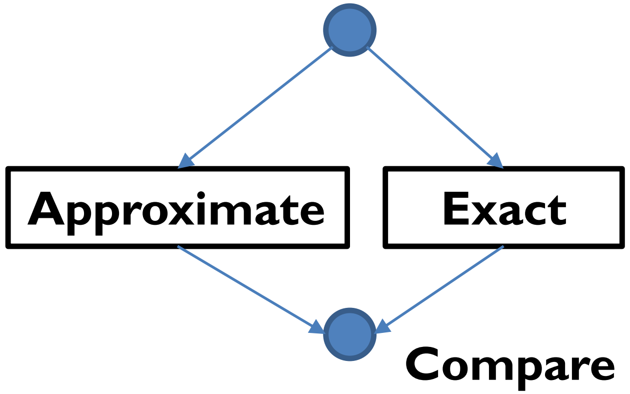
“for typical inputs…”

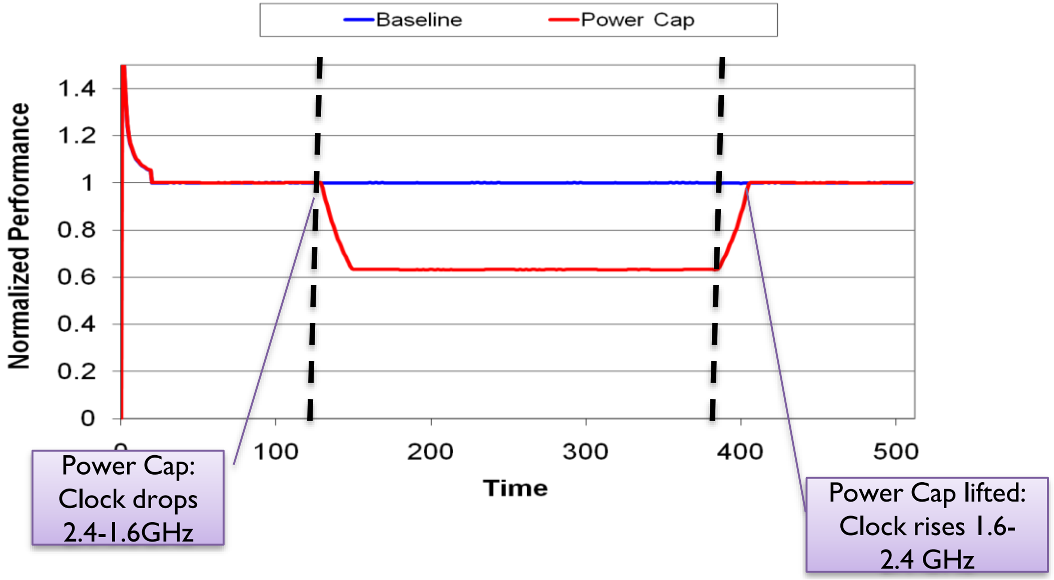
Goals of Runtime Adaptation

Accuracy (Green)

Time or Energy
(Loop perforation)







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

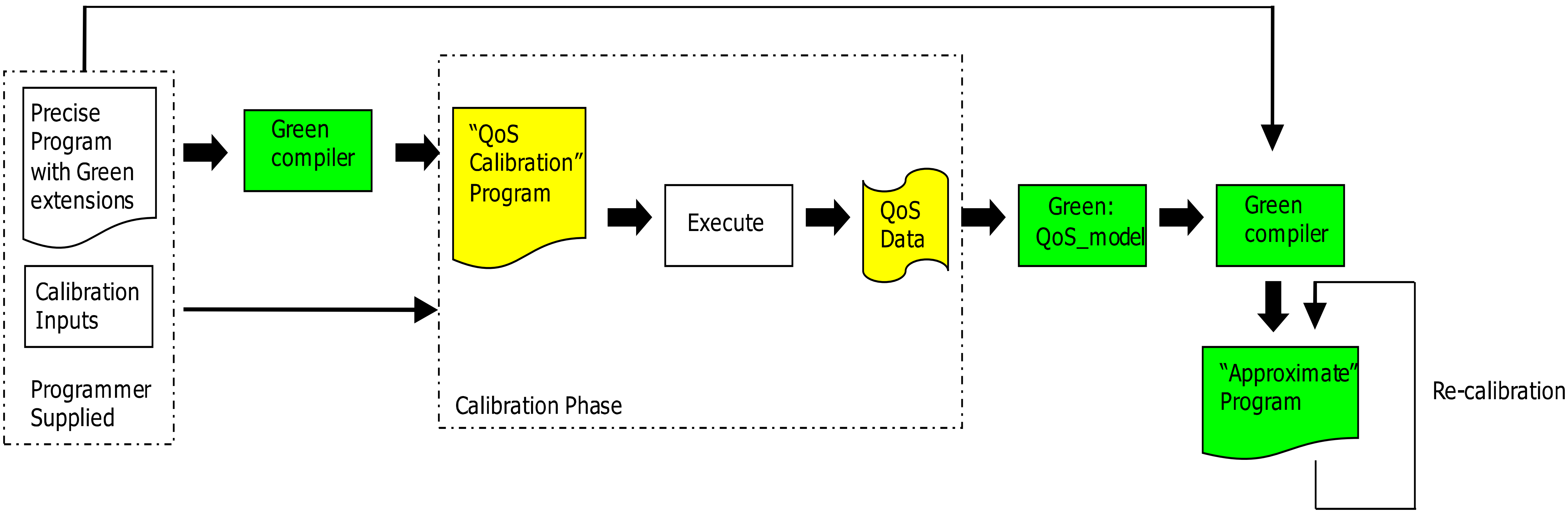
Hooks for expert programmers and custom policies

Online mechanism to reactively adapt approximation

policy to meet QoS

Adopted from slides by Radha Venkategiri

Green Framework



Recalibration

Concern: Overhead for running non-approximate

Address: Run infrequently, restructure the code

Original code:

```
#approx_loop (*QoS_Compute, Calibrate_QoS,
              QoS_SLA, Sample_QoS, static)
for(i=0; i<N; i++) {
  pi_est += factor/(2*i+1);
  factor /= -3.0;
}
```

Calibration code:

```
for(i=0; i<N; i++) {
   loop_body;
   if ((i%Calibrate_QoS)==0) {
     QoS_Compute(0, i, ...);
   }
}
QoS_loss = QoS_Compute(1, i, ...);
store(i, QoS_loss);
```

Approximation code:

```
count++;
recalib=false;
if (count%Sample_QoS==0) {
 recalib=true;
}

for (i=0; i<N; i++) {
 loop_body;
 if (QoS_Lp_Approx(i, QoS_SLA, true)) {
  if (!recalib) {
   // Terminate the loop early
   break;
  } else {
   // For recalibration, log the QoS value
   // and do not terminate the loop early
   if(!stored_approx_QoS) {
    QoS_Compute(0, i, …);
    stored_approx_QoS = 1;
   }}}}

if(recalib) {
 QoS_loss=QoS_Compute(1, i, …);
 QoS_ReCalibrate(QoS_loss, QoS_SLA);
}
```

Default QoS_Lp_Approx:

```
QoS_Lp_Approx(loop_count, QoS_SLA, static) {
 if (loop_count<M)
   return false;
 else {
  if (static)
   return true;
  else {
   // adaptive approximation
}}}
```

Default QoS_ReCalibrate:

```
QoS_ReCalibrate(QoS_loss, QoS_SLA) {
 if (QoS_loss>QoS_SLA) {
  // low QoS case
  increase_accuracy();
 } else if (QoS_loss<0.9*QoS_SLA) {
  // high QoS case
  decrease_accuracy();
 } else {
  ;// do nothing
 }
}
```

Figure 3. An end-to-end example of applying loop approximation to the Pi estimation program.

Recalibration

```
QoS_ReCalibrate(QoS_loss, QoS_SLA) {
 // n_m: number of monitored queries
 // n_l: number of low QoS queries in monitored queries
 if (n_m==0) {
  // Set Sample_QoS to 1 to trigger QoS_ReCalibrate
  // for the next 100 consecutive queries
  Saved_Sample_QoS=Sample_QoS;
  Sample_QoS=1;
 }
 n_m++;
 if (QoS_loss !=0)
  n_l++;
 if (n_m==100) {
  QoS_loss=n_l/n_m;
  if(QoS_loss>QoS_SLA) {
   // low QoS case
   increase_accuracy();
  } else if (QoS_loss < 0.9*QoS_SLA) {
   // high QoS case
   decrease_accuracy();
  } else {
   ; // no change
  }
  Sample_QoS=Saved_Sample_QoS;
 }
}
```

Figure 9. Customized QoS_ReCalibrate for Bing Search.

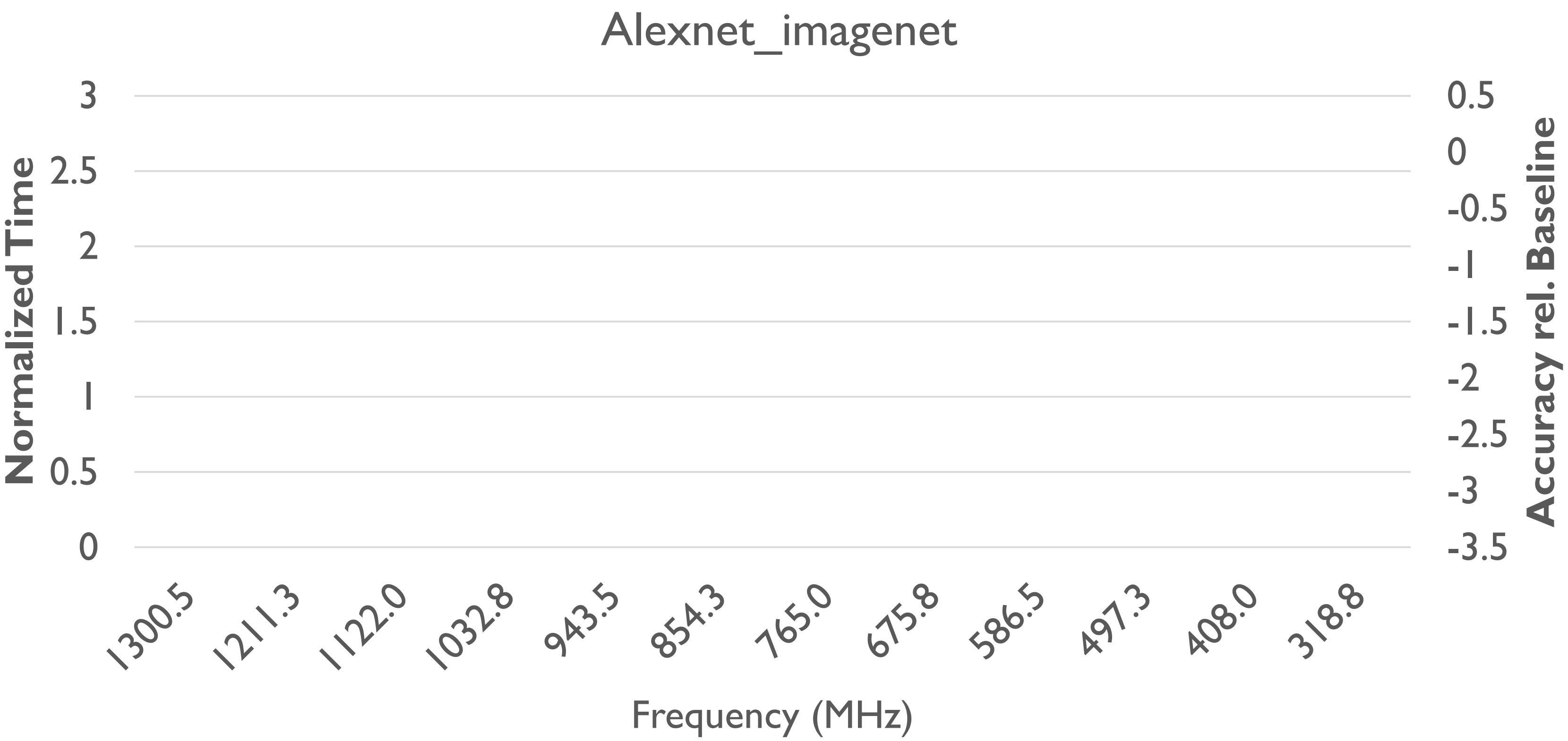
Runtime Adaptation for Accuracy

Key concerns:

- **Reexecuting infrequently to reduce the overhead**
  checking every result is expensive, rely on spatial and temporal locality

- **The computation needs to be amenable for re-execution:**
  think no side effects or crashes due to approximation

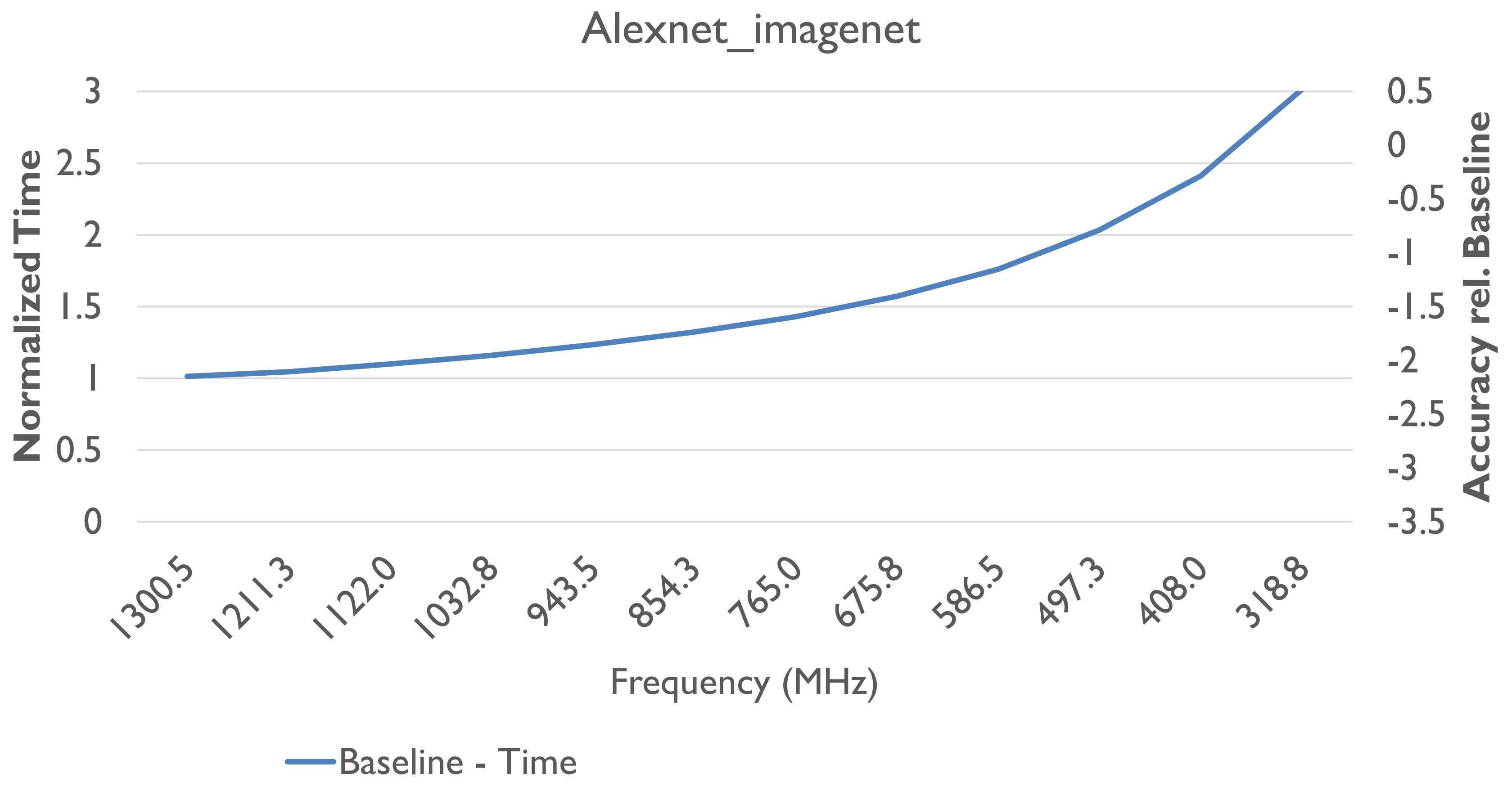
Runtime Approximation for Time/Energy



At regular time intervals we gradually reduce the frequency of the SoC

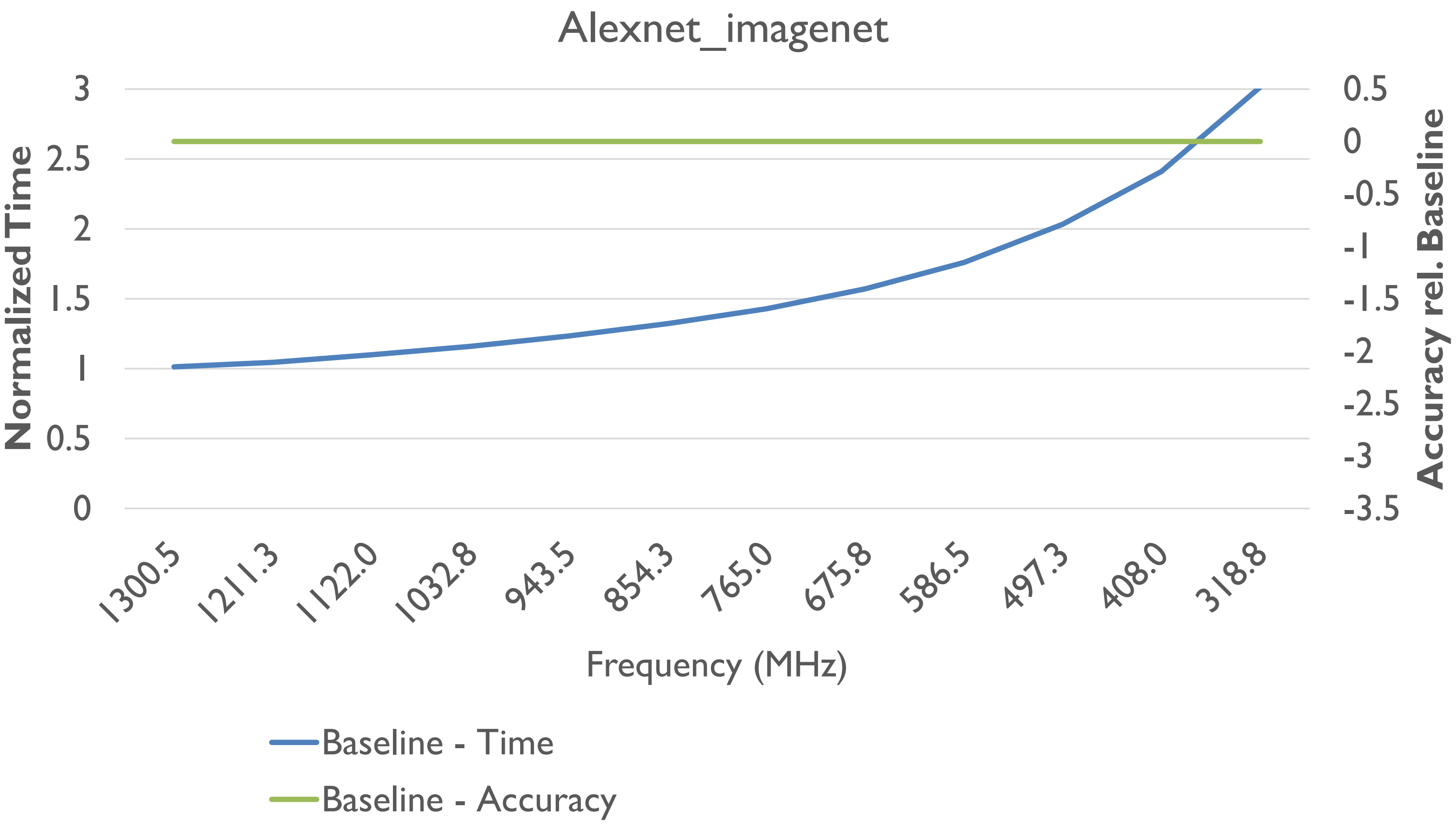
When you notice a disruption, read the value from the tradeoff curve that would negate the disruption

Runtime Approximation for Time/Energy



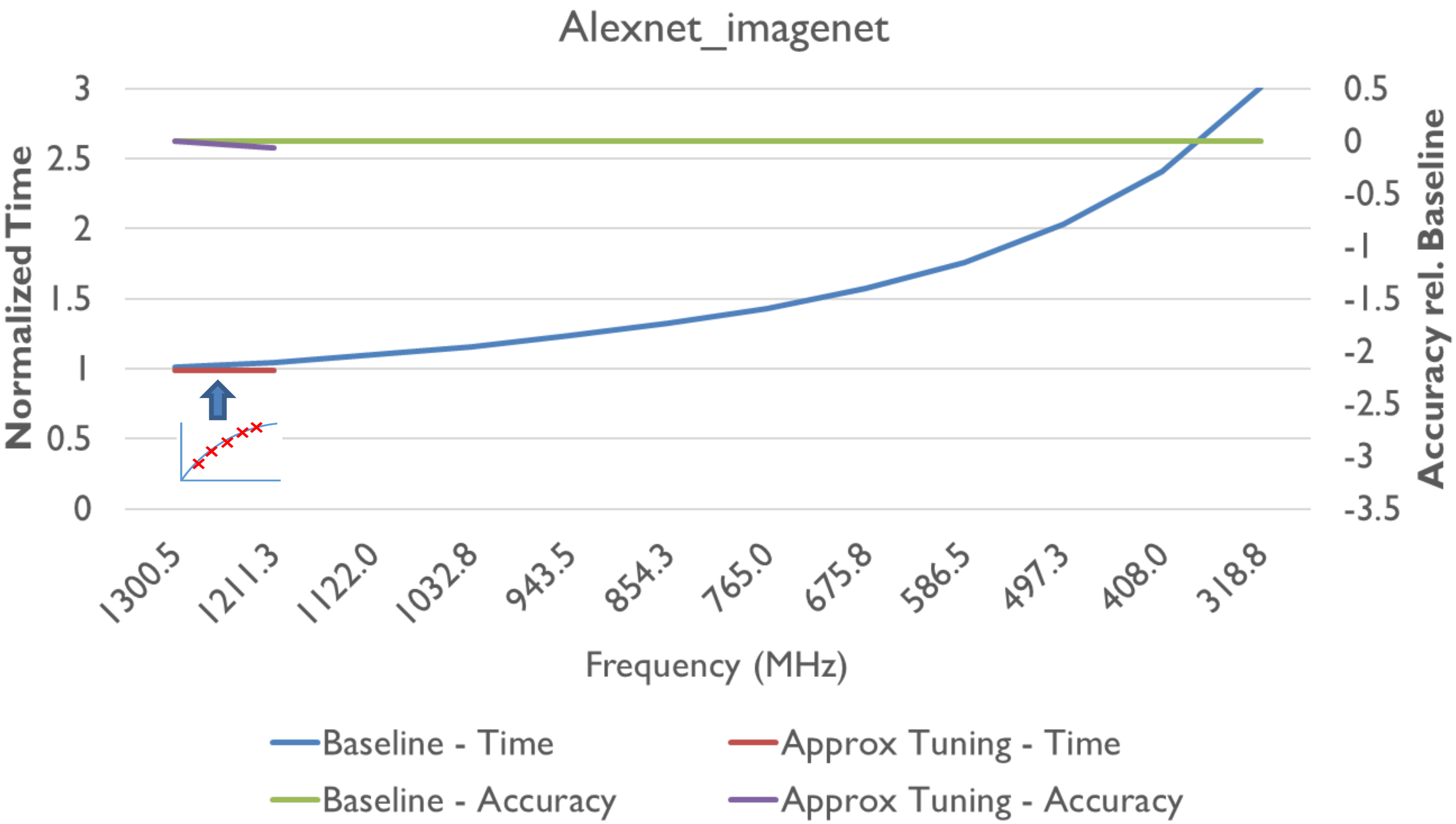
Runtime tuning helps maintain responsiveness in face of frequency changes

Runtime Approximation for Time/Energy



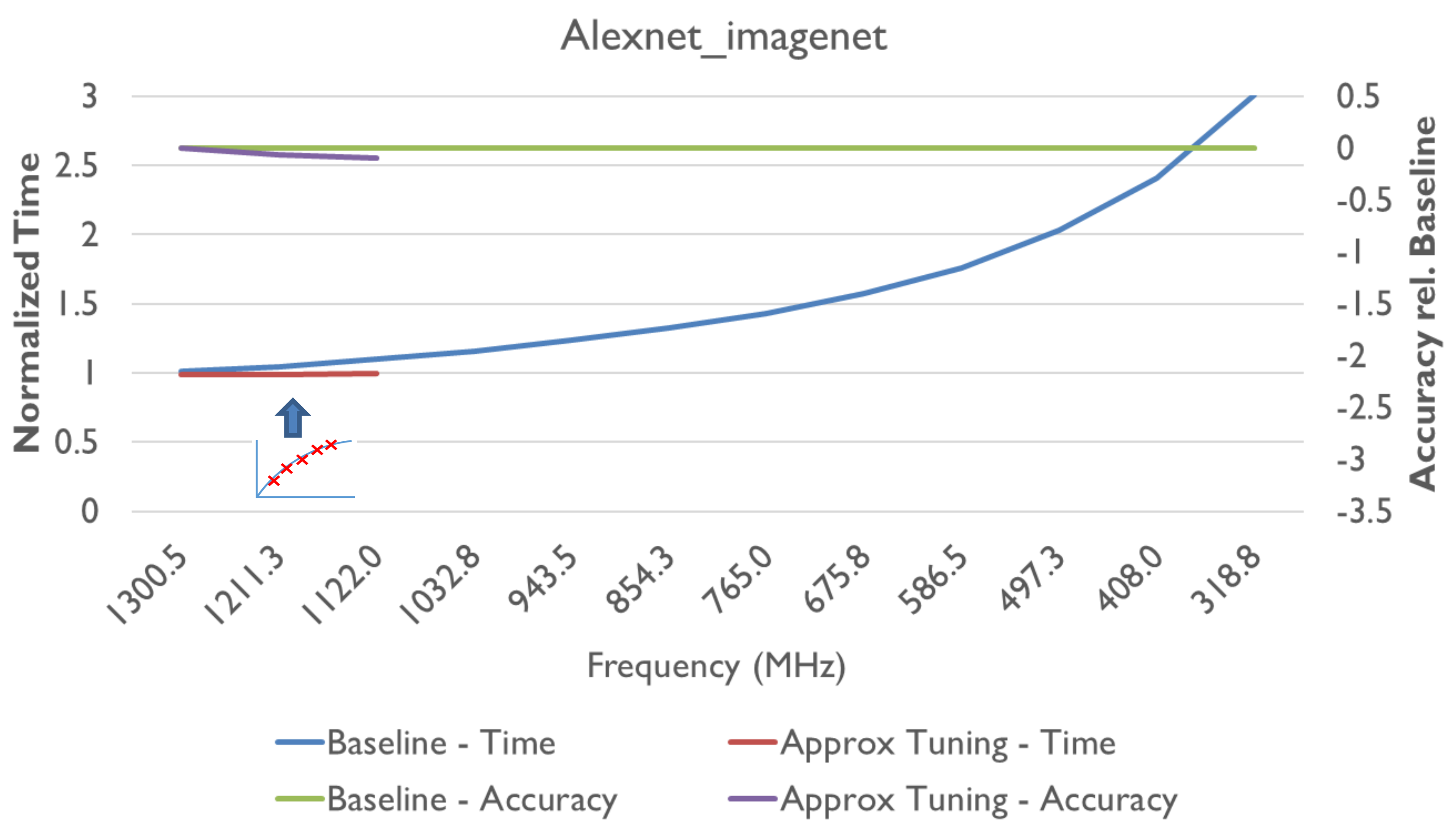
Runtime tuning helps maintain responsiveness in face of frequency changes

Runtime Approximation for Time/Energy



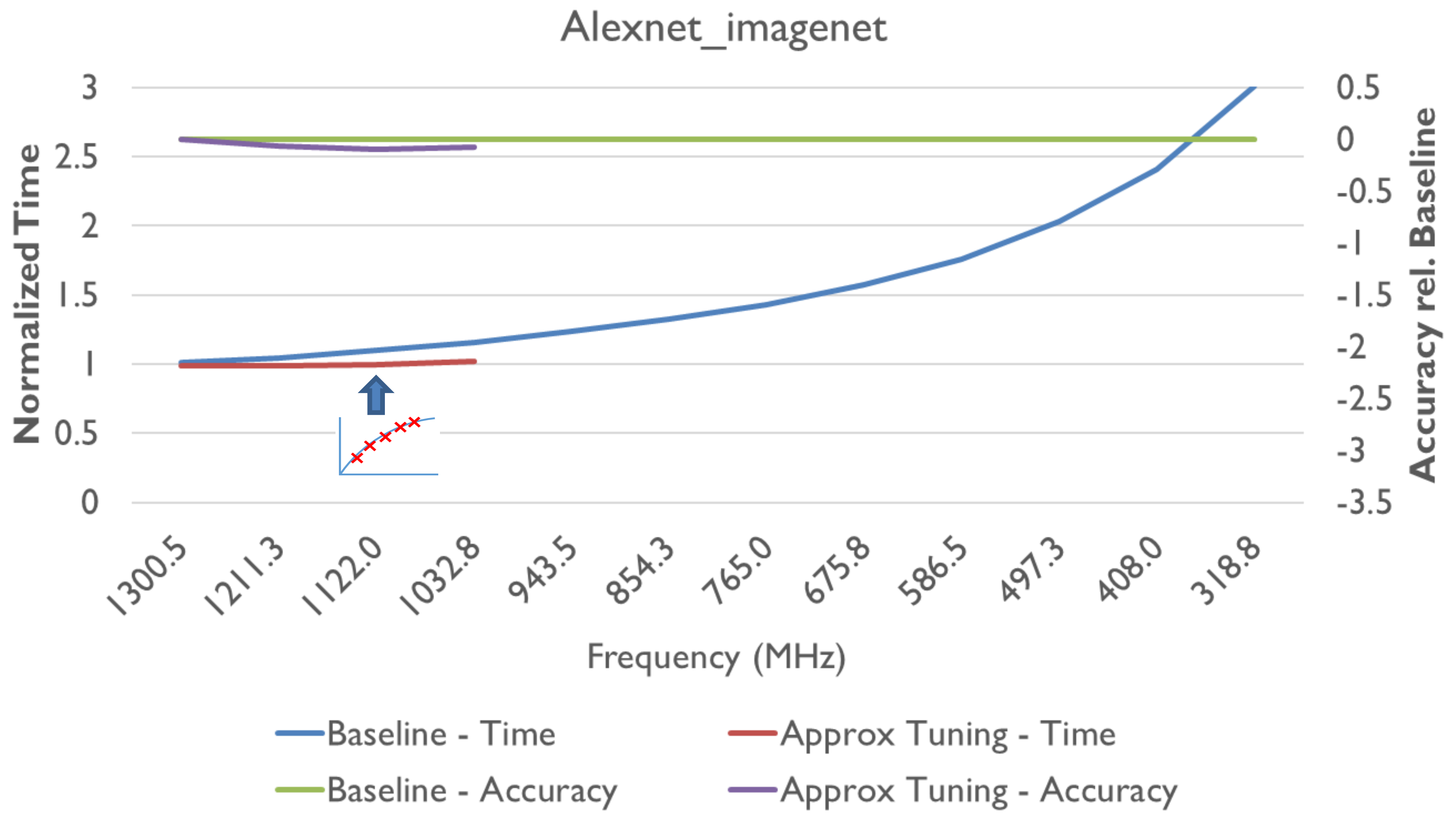
Runtime tuning helps maintain responsiveness in face of frequency changes

Runtime Approximation for Time/Energy



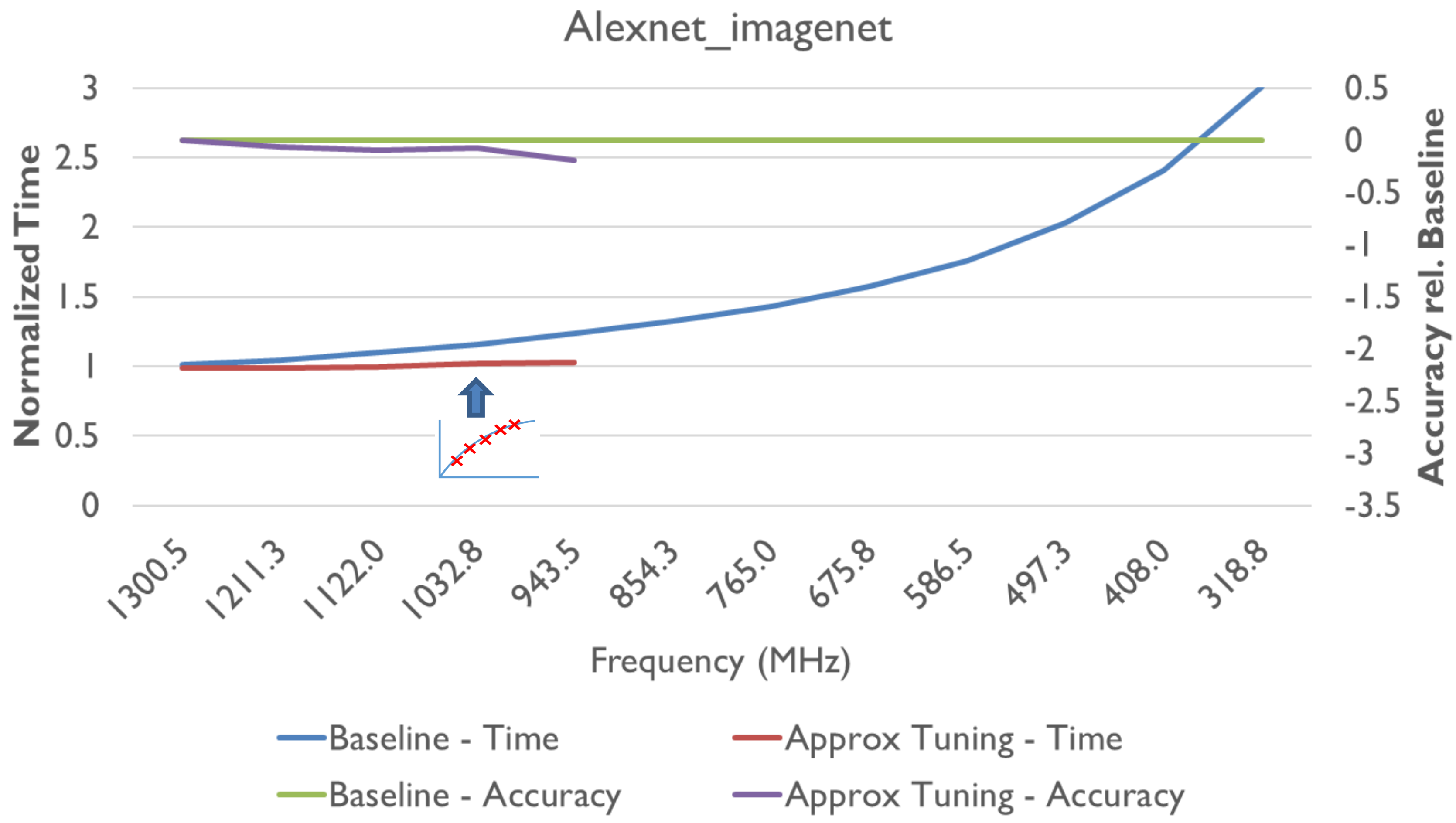
Runtime tuning helps maintain responsiveness in face of frequency changes

Runtime Approximation for Time/Energy



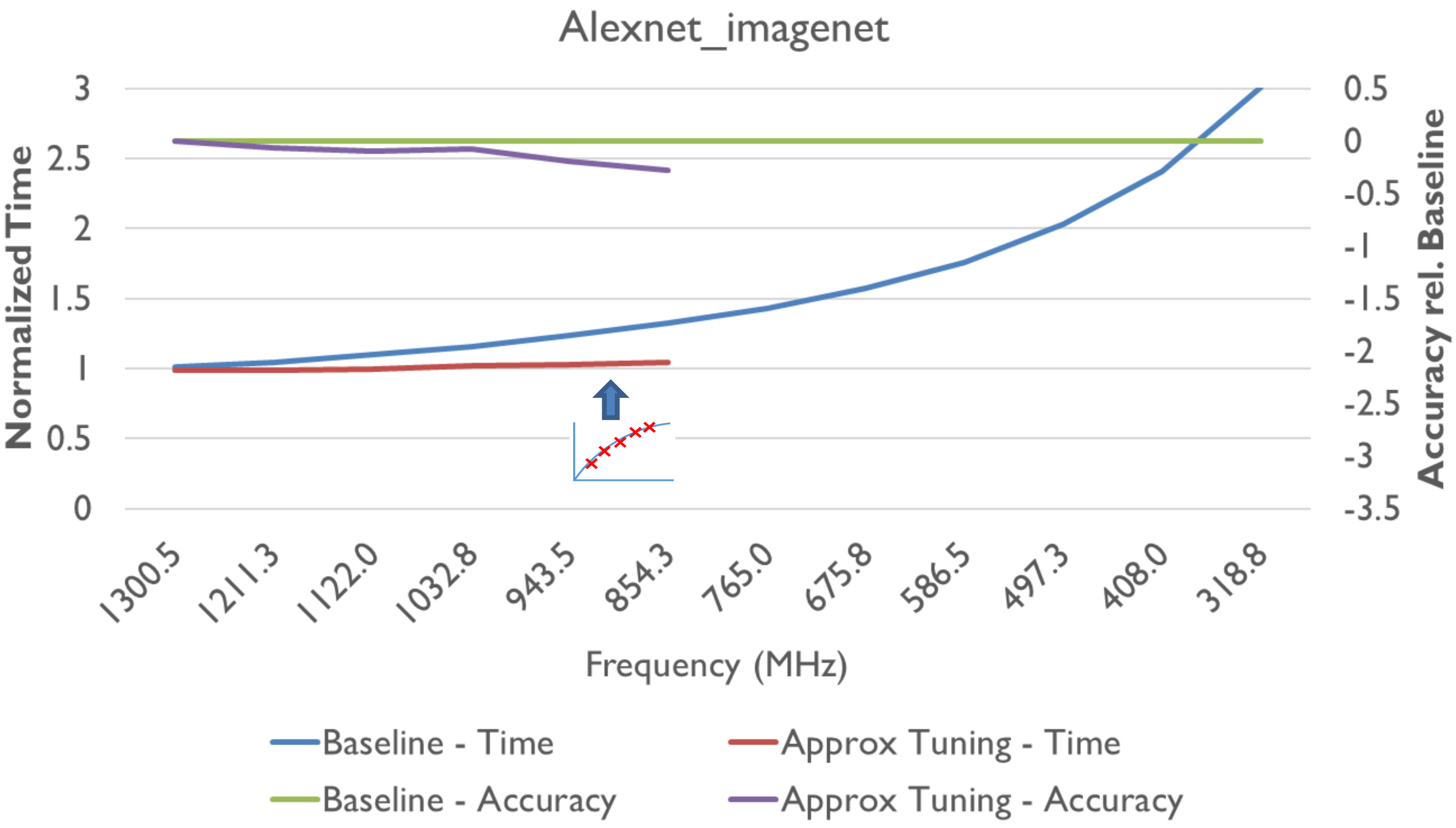
Runtime tuning helps maintain responsiveness in face of frequency changes

Runtime Approximation for Time/Energy



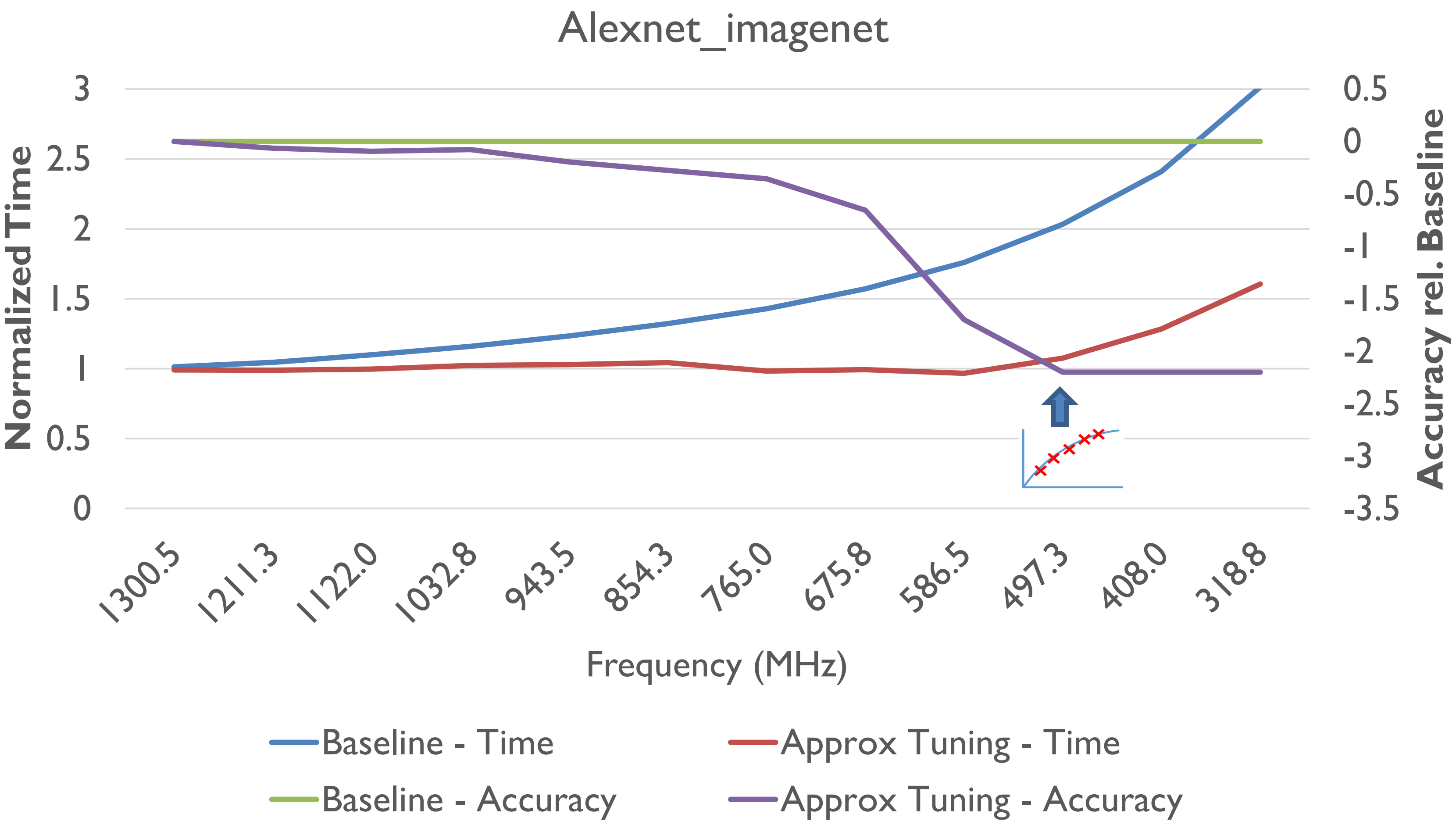
Runtime tuning helps maintain responsiveness in face of frequency changes

Runtime Approximation for Time/Energy



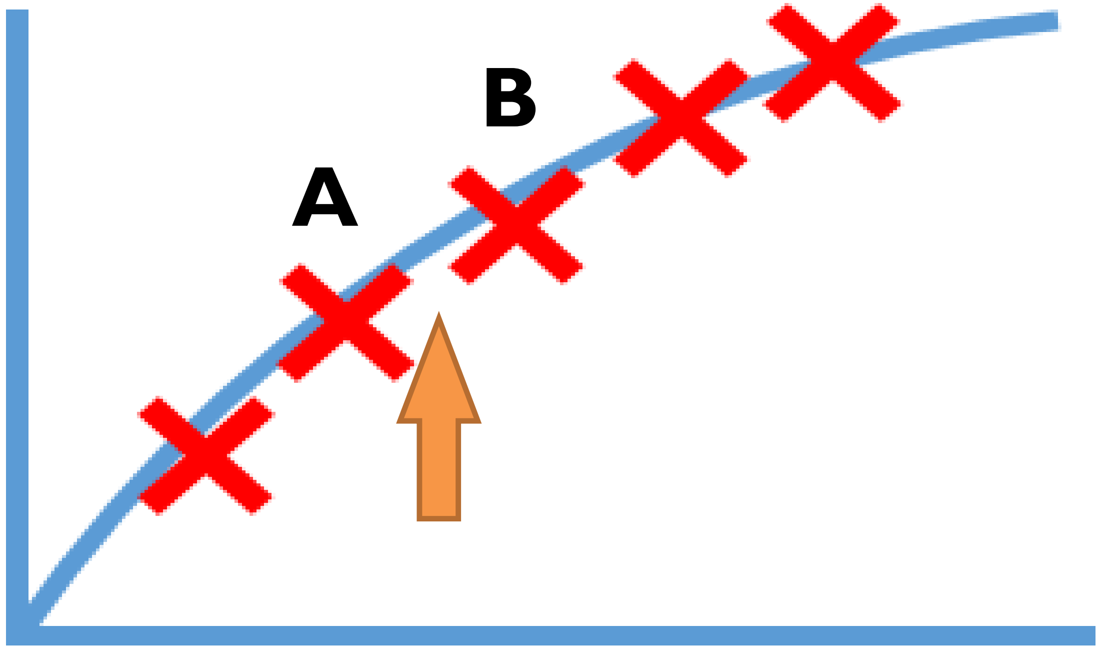
Runtime tuning helps maintain responsiveness in face of frequency changes

Runtime Approximation for Time/Energy



Runtime tuning helps maintain responsiveness in face of frequency changes

What if you don't have the exact point?



Solution 1: Select more conservative, suffer some performance drop (point A)

Solution 2: Select more aggressive, lose some more accuracy and make program even faster (point B)

Solution 3: We can use randomization

- Choose point A with probability p and
- Choose point B with probability 1-p

Why would this work over a long sequence of runs?

For **all** inputs

Original Computation

~~Typical Inputs~~

Accuracy Requirement

Analysis-driven Compiler

SAS '11, POPL '12, OOPSLA'13, OOPSLA '14, OOPSLA'19, CGO '20

- **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

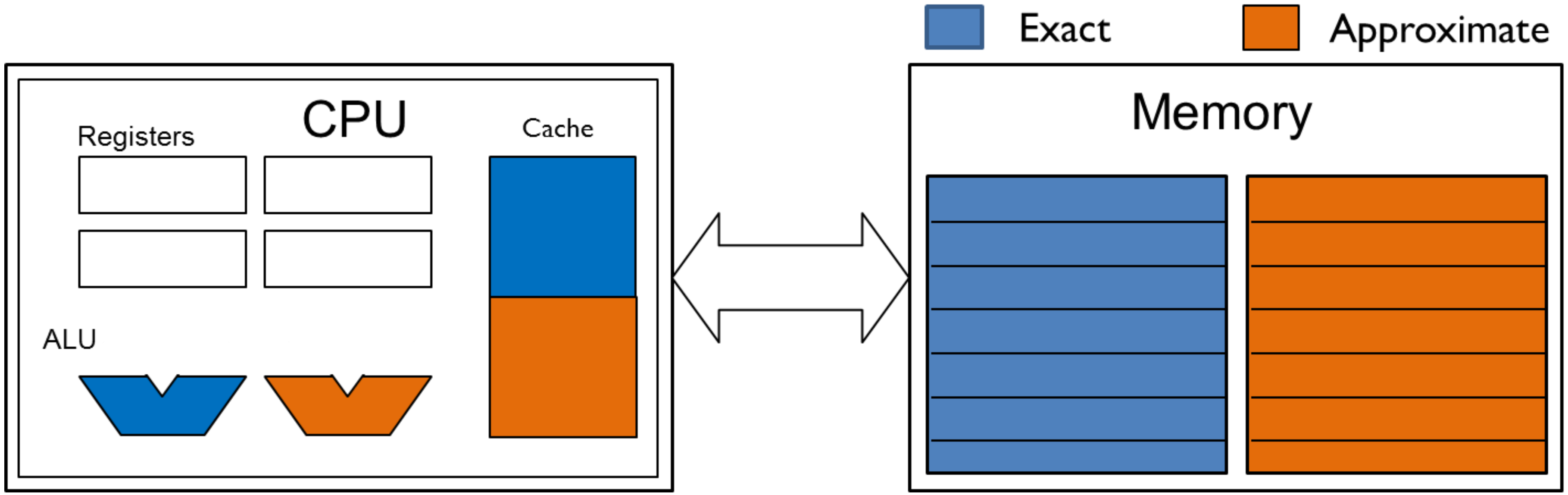
EnerJ Type System

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)

Approximate Hardware Model from EnerJ




```
reliability spec {
  operator (+.) = 1 - 10^-7;
  operator (-.) = 1 - 10^-7;
  operator (*.) = 1 - 10^-7;
  operator (<.) = 1 - 10^-7;
  memory rel {rd = 1, wr = 1};
  memory urel {rd = 1 - 10^-7, wr = 1};
}
```

Recall – hardware approximations:

- Soft errors
- Timing errors
- Voltage variations
- Aging, Refresh rates, …

All can be modeled as wrong bits (permanent or transient)

EnerJ Type System

Idea:

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

Variable annotations (extends Java annotation system)

@Approx int a = approximate_code();

int p;

p = a; **<---------- not ok**

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.”

Relational Safety Verification

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

avg = sum / m
```

```
i < 2*m/3
i < m/2
```

Carbin, Kim, Misailovic, Rinard, Proving acceptability properties of relaxed nond… …imate programs (PLDI'12)
Carbin, Kim, Misailovic, Rinard, Verified integrity properties for safe approximate program transformations (PEPM'13)
Boston, Gong, Carbin, Leto: Verifying Application-Specific Fault Tolerance through Parameterized Execution Models (OOPSLA'18)
He, Lahiri, Rakamaric, Verifying Relative Safety, Accuracy, and Termination for Program Approximations (JAR), 2018

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]

Analysis-Based Optimizations

Accuracy Specification

Reliability	Function computes result correctly with probability > ***0.99***
Absolute Error	Absolute error of function's result < ***2.0***
Reliability and Absolute Error	Absolute error of function's result < ***2.0*** with probability > ***0.99***

`int` $\{\Delta f \leq 2\,;\, 0.99 * \mathrm{R}(\Delta x = 0, \Delta y = 0)\}$ `f(int x, int y);`

Reliability Specification

`int` $< \overbrace{0.99}^{\text{Reliability degradation}} * \mathrm{R}(\Delta x = 0, \Delta y = 0, \Delta \mathrm{src} = 0) >$

```
interpolation(int x, int y, int src[][]);
```

The function computes result correctly with probability at least 0.99

Reliability Specification

$$\texttt{int} < \underbrace{0.99}_{\text{Reliability degradation}} * \underbrace{\mathbf{R}(\Delta x = 0, \Delta y = 0, \Delta \mathrm{src} = 0)}_{\text{Parameter Reliability}} >$$

```
interpolation(int x, int y, int src[][]);
```

Probability that the parameters have correct values before function starts executing

(facilitates function composition)

Reliability Specification

$$\texttt{int} < \overbrace{0.99}^{\text{Reliability degradation}} * \overbrace{\mathrm{R}(\Delta x = 0, \Delta y = 0, \Delta src = 0)}^{\text{Parameter Reliability}} >$$

```
interpolation(int x, int y, int src[][]);
```

- **Reliability factor:** $\mathrm{R}(\Delta v_1 \le d_1, \ldots, \Delta v_n \le d_n)$

$\boxed{\Delta v \equiv \left| v_{exact} - v_{approx} \right|}$

$\boxed{\text{Numerical bound}}$

Function Optimization Problem

Find Function Configuration q:

$$\begin{aligned}
\textbf{max}\quad & \text{EnergySavings}\,(q) \\
\textbf{s.t.}\quad & \text{Reliability}\,(q) \geq \text{ReliabilityBound} \\
& \text{AbsoluteError}\,(q) \leq \text{ErrorBound}
\end{aligned}$$

$\underbrace{\hphantom{\text{EnergySavings}\,(q)\quad}}_{\textbf{Analysis of the Function}}\qquad\underbrace{\hphantom{\text{ReliabilityBound}}}_{\textbf{Specifications}}$

Image Scaling

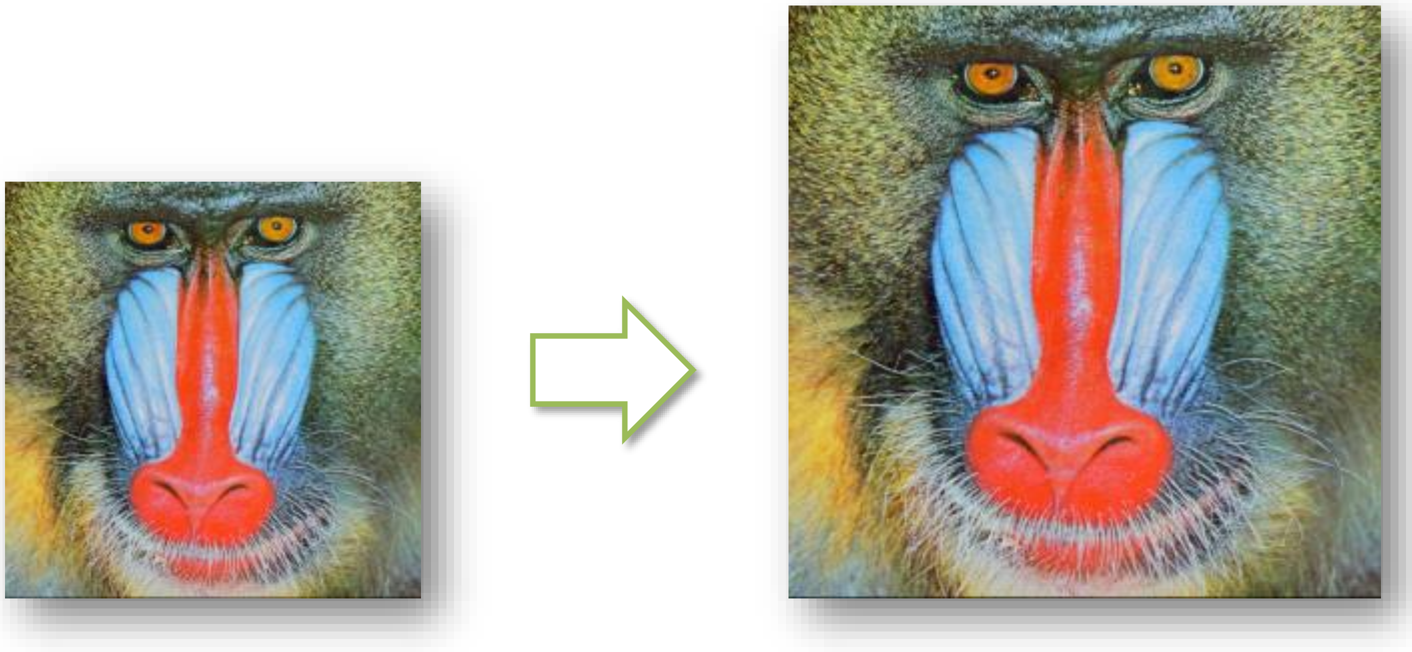
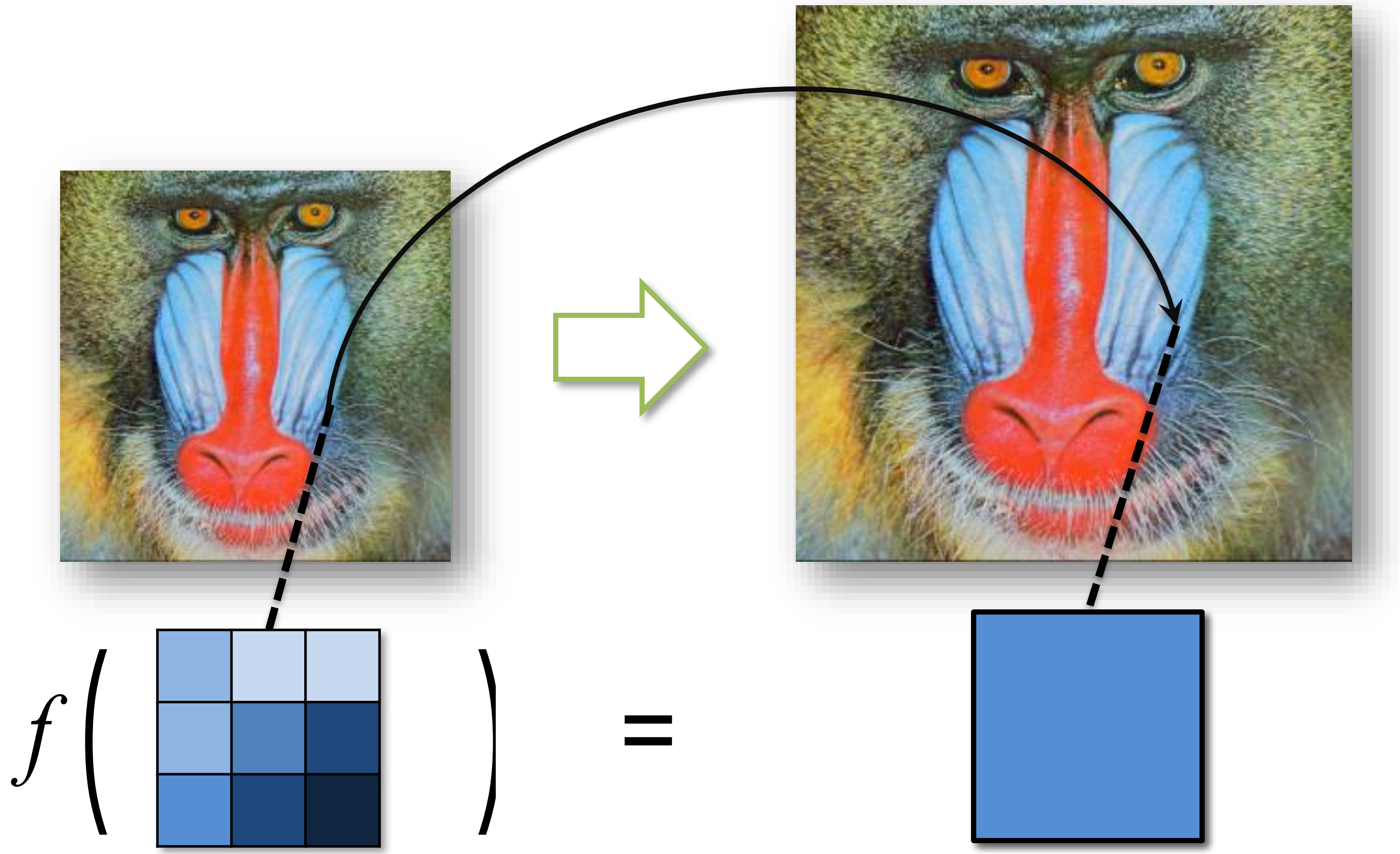


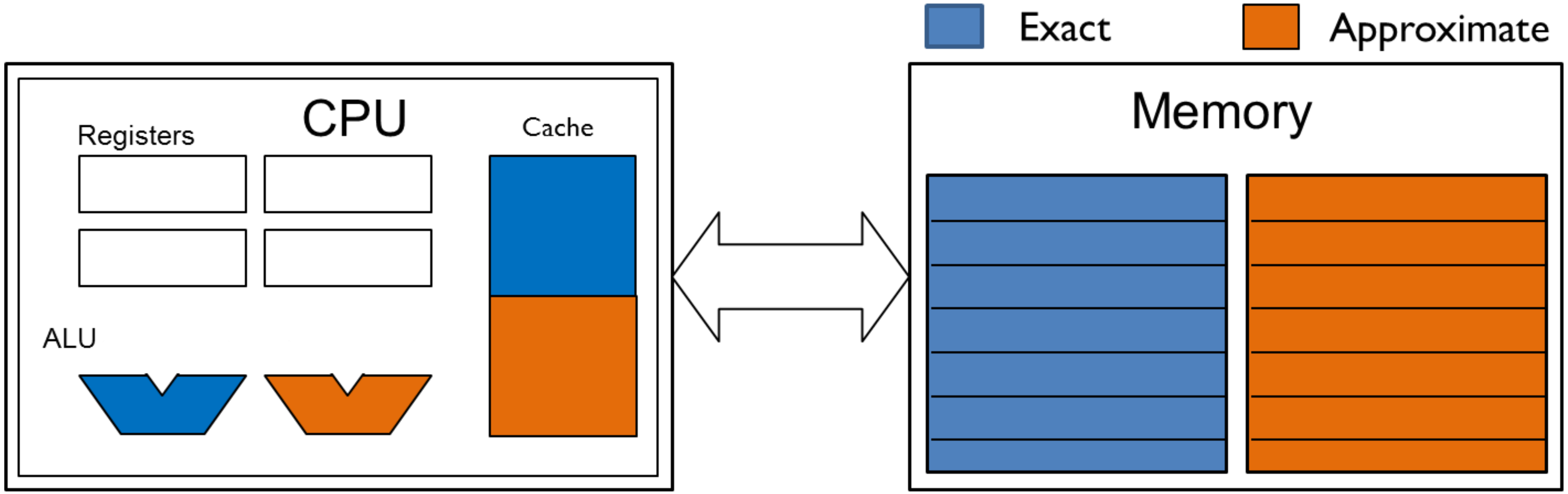
Image Scaling: Interpolation Function



Interpolation Function

```
int interpolation(int dst_x, int dst_y, int src[][])
{
    int x = src_location_x(dst_x, src),
        y = src_location_y(dst_y, src);

    int up    = src[y - 1][x],
        down  = src[y + 1][x],
        left  = src[y][x - 1],
        right = src[y][x + 1];

    int val = up + down + left + right;

    return 0.25 * val;
}
```

Approximate Hardware Model from EnerJ




```
reliability spec {
  operator (+.) = 1 - 10^-7;
  operator (-.) = 1 - 10^-7;
  operator (*.) = 1 - 10^-7;
  operator (<.) = 1 - 10^-7;
  memory rel {rd = 1, wr = 1};
  memory urel {rd = 1 - 10^-7, wr = 1};
}
```

Recall – hardware approximations:

- Soft errors
- Timing errors
- Voltage variations
- Aging, Refresh rates, …

All can be modeled as wrong bits (permanent or transient)

Run Function on Approximate Hardware

```
int interpolation(int dst_x, int dst_y, int src[][])
{
    int x = src_location_x(dst_x, src),
        y = src_location_y(dst_y, src);

    int up    = src[y -. 1][x],
        down  = src[y +. 1][x],
        left  = src[y][x -. 1],
        right = src[y][x +. 1];

    int val = up +. down +. left +. right;

    return 0.25 *. val;
}
```

Run Function on Approximate Hardware

```
int interpolation(int@ dst_x, int@ dst_y, int@ src[][])
{
    int@ x = src_location_x(dst_x, src),
         y = src_location_y(dst_y, src);

    int@ up    = src[y -. 1][x],
         down  = src[y +. 1][x],
         left  = src[y][x -. 1],
         right = src[y][x +. 1];

    int@ val = up +. down +. left +. right;

    return 0.25 *. val;
}
```

Function Configuration

Binary vector $\boldsymbol{q} = (q_1, q_2, \ldots, q_n)$

Variable Declarations:

- $\boldsymbol{q_i}$ - if 1, variable is stored in approximate memory
  if 0, variable is stored in exact memory

Arithmetic Operations:

- $\boldsymbol{q_i}$ - if 1, the operation is approximate,
  if 0, the operation is exact

Function Configuration

```
int interpolation(int dst_x, int dst_y, int src[][])
{
    int x = src_location_x(dst_x, src);
    int y = src_location_y(dst_y, src);

    int up    = src[y - 1][x];
    int down  = src[y + 1][x];
    int left  = src[y][x - 1];
    int right = src[y][x + 1];

    int val = up + down + left + right;

    return 0.25 * val;
}
```

Function Configuration

```
int interpolation(int_{q_dstx} dst_x, int_{q_dsty} dst_y, int_{q_src} src[][])
{
    int_{q_x} x = src_location_x(dst_x, src);
    int_{q_y} y = src_location_y(dst_y, src);

    int_{q_up}    up    = src[y - 1][x];
    int_{q_down}  down  = src[y + 1][x];
    int_{q_left}  left  = src[y][x - 1];
    int_{q_right} right = src[y][x + 1];

    int_{q_val} val = up + down + left + right;

    return 0.25 * val;
}
```

Each assignment of vector q denotes a different approximate function

```
int interpolation(int_{q_{dstx}} dst_x, int_{q_{dsty}} dst_y, int_{q_{src}} src[][])
{
    int_{q_x} x = src_location_x(dst_x, src);
    int_{q_y} y = src_location_y(dst_y, src);

    int_{q_{up}}    up    = src[y -_{q_7} 1][x];
    int_{q_{down}}  down  = src[y +_{q_6} 1][x];
    int_{q_{left}}  left  = src[y][x -_{q_5} 1];
    int_{q_{right}} right = src[y][x +_{q_4} 1];

    int_{q_{val}} val = up +_{q_1} down +_{q_2} left +_{q_3} right;

    return 0.25 *_{q_0} val;
}
```

Reliability Analysis

Motivation

- Efficiently represent reliability of **all approximate** function versions
- Construct constraints to separate **those** function versions **that satisfy specification**

Reliability Analysis

Approximate hardware specification:

- Reliability of arithmetic operations: $r_{op} \in (0, 1]$
- Reliability of memory reads and writes: $r_{rd}, r_{wr} \in (0, 1]$

```
operator (*) = 0.9999;
memory approx {rd = 0.99998, wr = 0.99999};
```

Analysis:

- Sound **static analysis**, operates backward
- Constructs **symbolic expressions** that characterize reliability of kernel's traces

Reliability Analysis

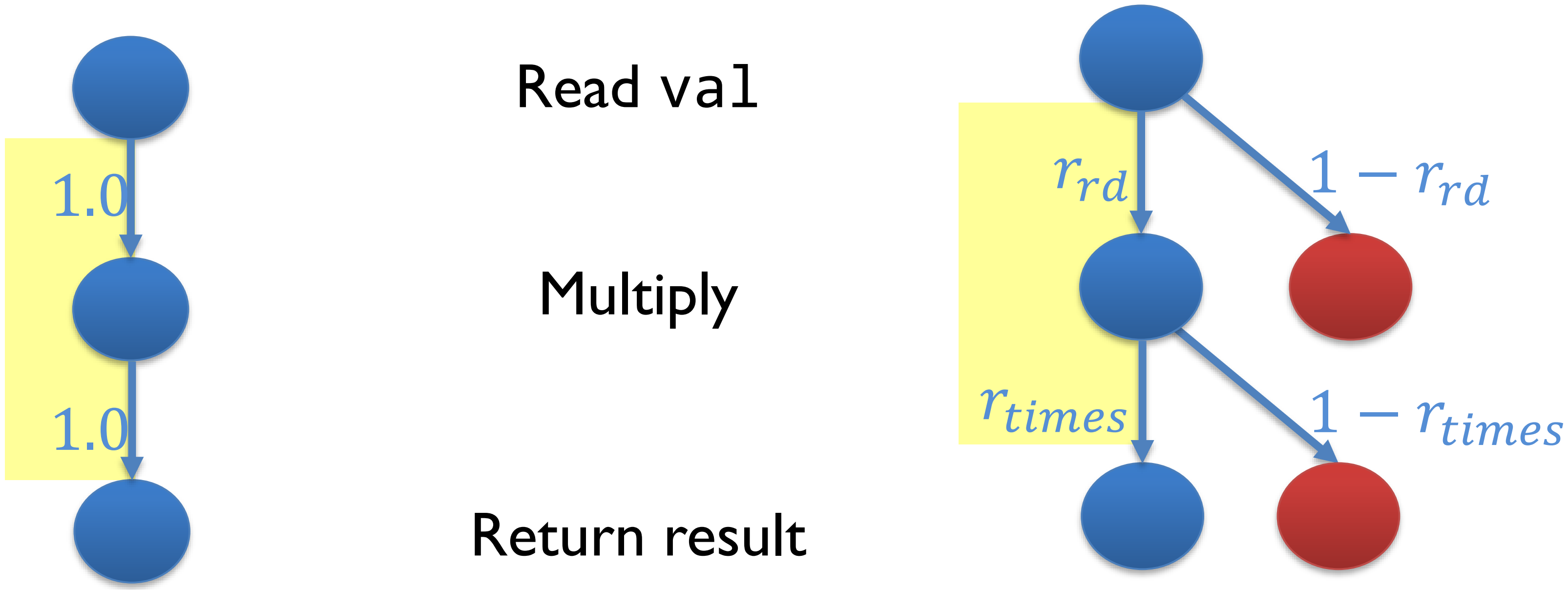
Statement `return val * 0.25;`

Exact Statement

*val and * exact*

Approximate Statement

*val and * approximate*

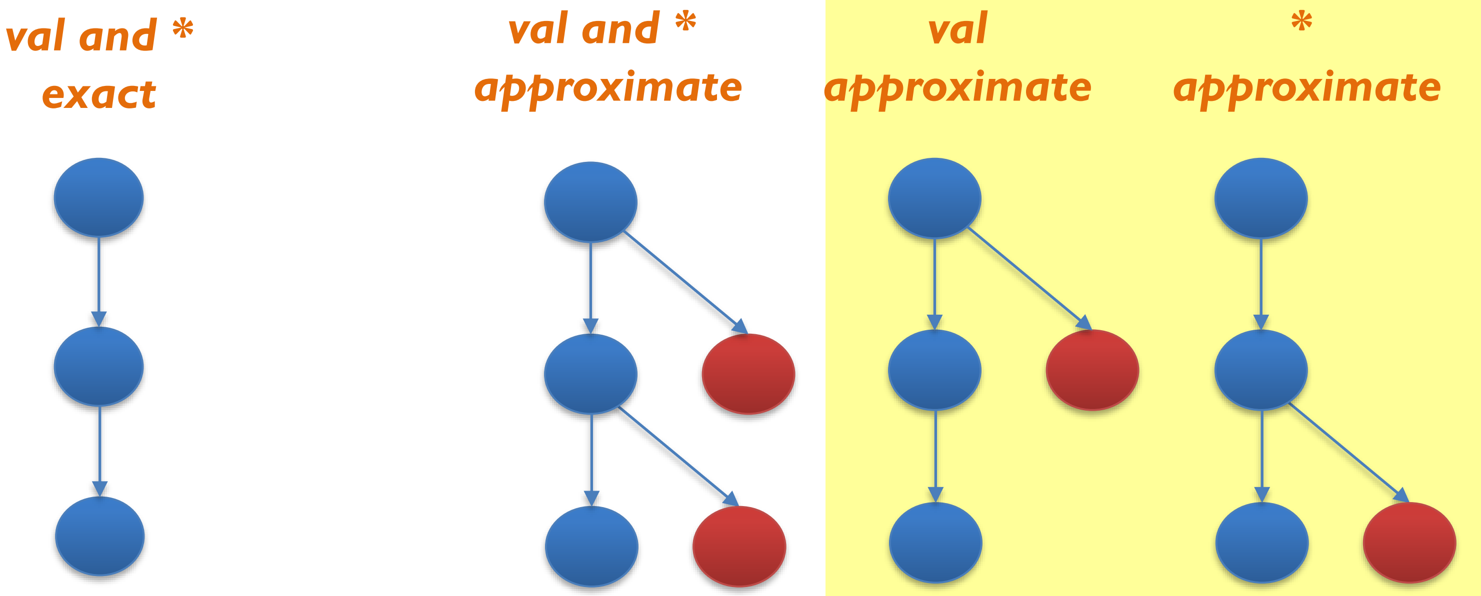


Reliability Analysis

Statement

```
return val * 0.25;
```

Exact Statement **Approximate Statements**

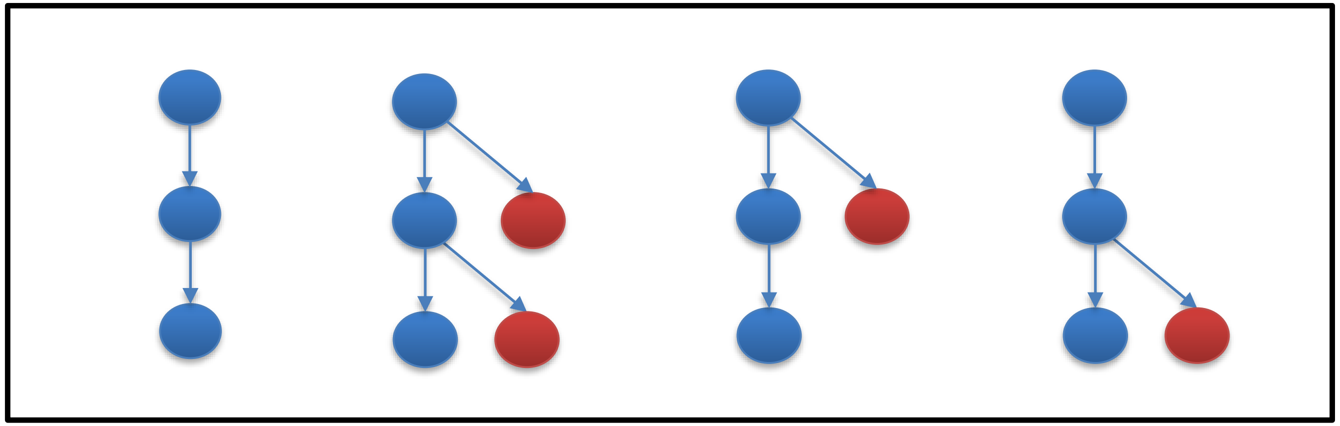


Statement reliability

Exact Statement	Approximate Statements		
1.0	$r_{rd} \cdot r_{times}$	r_{rd}	r_{times}

Reliability Analysis

Statement `return val * 0.25;`



Encode approximation choice:

- Variable declaration: `int` q_{val} `val;`
- Multiplication: `val *` q_{*} `0.25;`

Reliability Analysis

Statement

```
return val * 0.25;
```

Reliability Expression

$$\boxed{(r_{rd})^{q_{val}} \cdot (r_{times})^{q_*} \cdot \mathrm{R}(\Delta\mathrm{val} = 0)}$$

Encode approximation choice:

- Variable declaration: `int`$_{q_{val}}$ `val;`
- Multiplication: `val *`$_{q_*}$ `0.25;`

Reliability Analysis

Statement

```
return val * 0.25;
```

Reliability Expression

$$\boxed{\left(\boldsymbol{r_{rd}}\right)^{\boldsymbol{q_{val}}} \cdot \left(r_{times}\right)^{q_*} \cdot \mathrm{R}(\Delta\mathrm{val} = 0)}$$

Reliability of reading `val` from
<u>either</u> exact or approximate memory:

$$\boldsymbol{(r_{rd})^0 = 1.0} \qquad \boldsymbol{(r_{rd})^1 = r_{rd}}$$

Reliability Analysis

Statement

```
return val * 0.25;
```

Reliability Expression

$$(r_{rd})^{q_{val}} \cdot (\boldsymbol{r_{times}})^{\boldsymbol{q_{*}}} \cdot \mathrm{R}(\Delta\mathrm{val} = 0)$$

Reliability of <u>either</u> exact or approximate multiplication

Reliability Analysis

Statement

```
return val * 0.25;
```

Reliability Expression

$$\boxed{(r_{rd})^{q_{val}} \cdot (r_{times})^{q_{*}} \cdot \mathbf{R(\Delta val = 0)}}$$

Probability that previous statements computed `val` correctly

Interpolation Function

$$\texttt{int interpolation(int}_{q_{dstx}}\ \texttt{dst\_x, int}_{q_{dsty}}\ \texttt{dst\_y, int}_{q_{src}}\ \texttt{src[][])}$$

```
{
```

$$\boxed{\left(r_{rd}\right)^{q_{val}} \cdot \left(r_{times}\right)^{q_*} \cdot \mathbf{R(\Delta val = 0)}}$$

$$\texttt{return val *}_{q_*}\ \texttt{0.25;}$$

```
}
```

Interpolation Function

```
int interpolation(int_{q_dstx} dst_x, int_{q_dsty} dst_y, int_{q_src} src[][])
{
```

$$\left(r_{rd}\right)^{q_{val}} \cdot \left(r_{times}\right)^{q_*} \cdot \left(r_{plus}\right)^{q_1+q_2+q_3} \cdot \left(r_{rd}\right)^{q_{up}+q_{down}+q_{left}+q_{right}} \cdot \mathrm{R}(\Delta\mathrm{up} = 0, \Delta\mathrm{down} = 0, \Delta\mathrm{left} = 0, \Delta\mathrm{right} = 0)$$

```
    int_{q_val} val = up +_{q_1} down +_{q_2} left +_{q_3} right;
```

$$\left(r_{rd}\right)^{q_{val}} \cdot \left(r_{times}\right)^{q_*} \cdot \mathrm{R}(\Delta\mathrm{val} = 0)$$

```
    return val *_{q_*} 0.25;
}
```

Reliability Expression

Function's Reliability Expression:

$$\boxed{\begin{array}{c}\Delta dst\_x = 0, \\ \Delta dst\_y = 0, \\ \Delta src = 0\end{array}}$$

$$\underbrace{r_1^{q_1} \cdot r_2^{q_2} \cdot \ldots \cdot r_n^{q_n}}_{\substack{\textbf{Probability operations} \\ \textbf{executed reliably} \\ \textbf{(for all approximate versions} \\ \textbf{of the function)}}} \cdot \underbrace{\mathrm{R}\left(P_{param} \right)}_{\substack{\textbf{Probability parameters} \\ \textbf{have correct values} \\ \textbf{at function start}}}$$

Reliability Constraint

Relates developer's specification and analysis result:

$$\boxed{r_{spec} \cdot \mathrm{R}\left(P_{spec}\right) \leq r_1^{\boldsymbol{q_1}} \cdot r_2^{\boldsymbol{q_2}} \cdot \ldots \cdot r_n^{\boldsymbol{q_n}} \cdot \mathrm{R}\left(P_{param}\right)}$$

Reliability Constraint

$$r_{spec} \leq r_1^{\boldsymbol{q_1}} \cdot r_2^{\boldsymbol{q_2}} \cdot \ldots \cdot r_n^{\boldsymbol{q_n}}$$

and

$$\underbrace{\mathrm{R}\left(P_{spec}\right) \leq \mathrm{R}\left(P_{param}\right)}_{\textbf{Can Immediately Solve}}$$

Reliability Constraint

$$r_{spec} \le r_1^{q_1} \cdot r_2^{q_2} \cdot \ldots \cdot r_n^{q_n}$$

and

$$\mathrm{R}\left(P_{spec}\right) \le \mathrm{R}\left(P_{param}\right)$$

$$\boxed{\begin{array}{c}\Delta \mathrm{dst\_x} = 0,\\ \Delta \mathrm{dst\_y} = 0,\\ \Delta \mathrm{src} = 0\end{array}} \longrightarrow P_{spec} \Rightarrow P_{param} \longleftarrow \boxed{\begin{array}{c}\Delta \mathrm{dst\_x} = 0,\\ \Delta \mathrm{dst\_y} = 0,\\ \Delta \mathrm{src} = 0\end{array}}$$

Reliability Constraint

$$\boxed{r_{spec} \leq r_1^{q_1} \cdot r_2^{q_2} \cdot \ldots \cdot r_n^{q_n}}$$

Denotes approximate function versions that satisfy the developer's specification

Reliability Constraint

for the optimization problem

$$\boxed{\log(r_{spec}) \leq \boldsymbol{q_1} \cdot \log(r_1) + \boldsymbol{q_2} \cdot \log(r_2) + \cdots + \boldsymbol{q_n} \cdot \log(r_n)}$$

Denotes approximate function versions that satisfy the developer's specification

Reliability and Control Flow

Conditionals	Constraints for each program path
	Analysis removes redundant constraints
	(most constraints can be removed - OOPSLA ’13)
Bounded Loops	Statically known loop bound Analysis unrolls loop
Optimization Granularity	Optimize blocks of code instead of individual instructions

Function Optimization Problem

Find Function Configuration q: ✓

$$\textbf{max}\ \text{EnergySavings}\ (q)$$

$$\text{Reliability}\ (q) \geq \text{ReliabilityBound}$$ ✓

$$\boxed{\text{AbsoluteError}\ (q) \leq \text{ErrorBound}}$$

Absolute Error Analysis

Reduced-precision floating-point instructions:

- <u>Almost always</u> incorrect, but error is bounded
- Hardware specification: number of significant mantissa bits

Analysis:

- Bounds worst-case numerical deviation
- Embeds accuracy predicate in reliability factor:

$$r_1^{q_1} \cdot \ldots \cdot r_n^{q_n} \cdot \mathrm{R}(\Delta x = 0, \Delta y = 0)$$

Absolute Error Analysis

Reduced-precision floating-point instructions:

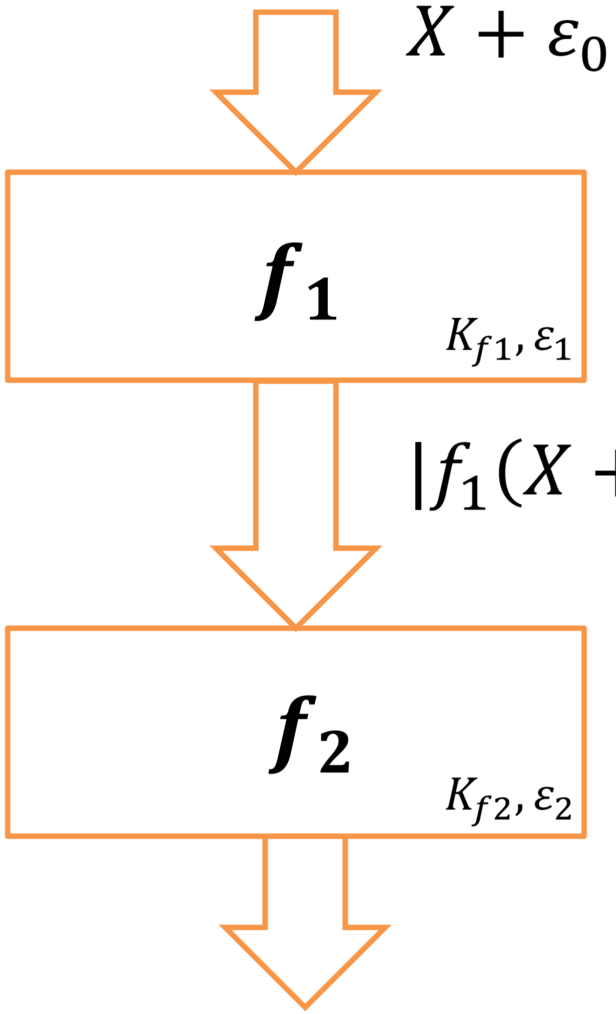
- Almost always incorrect, but error is bounded
- Hardware specification: number of significant mantissa bits

Analysis:

- Bounds worst-case numerical deviation
- Embeds accuracy predicate in reliability factor:

$$r_1^{\boldsymbol{q_1}} \cdot \ldots \cdot r_n^{\boldsymbol{q_n}} \cdot \boxed{\mathrm{R}\big(\underbrace{\Delta x + 2 \cdot \Delta y + \boldsymbol{q_1} \cdot \xi_{x,y}}_{\textbf{Linear function of } \boldsymbol{q_1}, \ldots, \boldsymbol{q_n}} < d\big)}$$

Recall Error Propagation



$$|f_1(X + \varepsilon_0) + \varepsilon_1 - f_1(X)| \leq K_{f1} \cdot \varepsilon_0 + \varepsilon_1$$

$$\boxed{K_f = \max_{x \in Inputs} \left| \frac{df}{dx} \right|}$$

$$|f_2(f_1(X + \varepsilon_0) + \varepsilon_1) + \varepsilon_2 - f_2(f_1(X))|$$

$$\leq K_{f2} \cdot K_{f1} \cdot \varepsilon_0 + K_{f2} \cdot \varepsilon_1 + \varepsilon_2$$

Error Propagation for Some Common Functions

$$K_f = \max_{x \in Inputs} \left| \frac{df}{dx} \right|$$

$$K_{fi} = \max_{x \in Inputs} \left| \frac{\partial f(x_1 \ldots x_n)}{\partial x_i} \right|$$

$f(x_1, x_2)$	Err
$x \cdot const$	$\Delta x \cdot const$
$x + y$	$\Delta x + \Delta y$
$x \cdot y$	$\Delta x \cdot \max(\lvert \mathrm{y} + \Delta y \rvert) + \Delta y \cdot \max(\lvert x + \Delta x \rvert)$

Interpolation Function

int interpolation(int$_{q_{dstx}}$ dst_x, int$_{q_{dsty}}$ dst_y, int$_{q_{src}}$ src[][])
{

$$\boxed{(r_{rd})^{q_{val}} \cdot (r_{times})^{q_*} \cdot \mathbf{R(0.25 \cdot \Delta val + q_* \cdot e_* \leq E)}}$$

    return val *$_{q_*}$ 0.25;
}

Interpolation Function

```
int interpolation(int_{q_dstx} dst_x, int_{q_dsty} dst_y, int_{q_src} src[][])
{
```

$$\boxed{(r_{rd})^{q_{val}} \cdot (r_{times})^{q_*} \cdot (r_{plus})^{q_1+q_2+q_3} \cdot (r_{rd})^{q_{up}+q_{down}+q_{left}+q_{right}} \cdot \mathrm{R}\left(\begin{array}{c} 0.25 \cdot (\Delta\mathrm{up} + \Delta\mathrm{down} + \Delta\mathrm{left} + \Delta\mathrm{right}) + \\ 0.25 \cdot (q_1 \cdot e_{+1} + q_2 \cdot e_{+2} + q_3 \cdot e_{+3}) + q_* \cdot e_* \leq E \end{array}\right)}$$

```
    int_{q_val} val = up +_{q_1} down +_{q_2} left +_{q_3} right;
```

$$\boxed{(r_{rd})^{q_{val}} \cdot (r_{times})^{q_*} \cdot \mathrm{R}(0.25 \cdot \Delta\mathrm{val} + q_* \cdot e_* \leq E)}$$

```
    return val *_{q_*} 0.25;
}
```

Function Optimization Problem

Find Function Configuration q: ✓

$$\textbf{max}\ \boxed{\text{EnergySavings}\ (q)}$$

$$\text{Reliability}\ (q) \geq \text{ReliabilityBound} ✓$$

$$\text{AbsoluteError}\ (q) \leq \text{ErrorBound} ✓$$

Energy Savings Analysis

Profile information:

- Collects traces from running representative inputs

Analysis:

- Estimates savings for instructions and variables from traces

instruction	**variable**
$\boldsymbol{q_\ell} \cdot Count_\ell \cdot Saving_{ALU}$	$\boldsymbol{q_m} \cdot Size_m \cdot Saving_{MEM}$

Energy Savings Analysis

Profile information:

- Collects traces from running representative inputs

Analysis:

- Estimates savings for instructions and variables from traces

$$\underbrace{c_{ALU} \sum_{\ell \in Instr} q_\ell \cdot Count_\ell \cdot Saving_{ALU}}_{\textbf{instruction}} + \underbrace{c_{MEM} \sum_{m \in Var} q_m \cdot Size_m \cdot Saving_{MEM}}_{\textbf{variable}}$$

Approximate hardware specification:

- Relative savings for operations and memories
- Percentage of system energy that ALU and memory consume

Function Optimization Problem

Find Function Configuration q: ✓

$$\textbf{max} \ \text{EnergySavings}\ (q)$$ ✓

$$\text{Reliability}\ (q) \geq \text{ReliabilityBound}$$ ✓

$$\text{AbsoluteError}\ (q) \leq \text{ErrorBound}$$ ✓

Reduces to Integer Programming

Find Function Configuration q: ✓

$$\mathbf{max}\ \text{EnergySavings}\ (q)$$ ✓

$$\text{Reliability}\ (q) \geq \text{ReliabilityBound}$$ ✓

$$\text{AbsoluteError}\ (q) \leq \text{ErrorBound}$$ ✓

Solve using off-the-shelf solvers (we use Gurobi)

Evaluation

Benchmarks With Approximated Functions:

Scale	image scaling
DCT	discrete cosine transform
IDCT	inverse discrete cosine transform
Blackscholes	financial option price calculation
SOR	successive over-relaxation kernel

Approximate Hardware Specifications:

- 5 specifications of ALU, caches, and memories from the literature [EnerJ – PLDI '11]

Complexity of Optimization Problem

Benchmark	Function LOC	Search Space Size	Reliability Constraints
Scale	88	2^{74}	4
DCT	62	2^{35}	1
IDCT	93	2^{53}	1
Blackscholes	143	2^{80}	2
SOR	23	2^{10}	1

Solver finds optimal solutions in less than a second

Energy/Accuracy Tradeoffs

Optimizer computes estimated system savings

Maximum estimated savings for hardware specifications:

Benchmark	Reliability Degradation	System-Level Energy Savings
Scale	0.995	19.4%
DCT	0.99992	8.7%
IDCT	0.992	13.4%
Blackscholes	0.999	9.8%
SOR	0.995	19.8%

Pros:

- Can explore the space induced by much finer grained transformations (e.g., numerical precision)
- The results are valid for all inputs within range
- New analyses were developed in the meantime

Cons:

- Static analysis is ***much*** more conservative than testing
- The set of supported programs is limited

Analysis: Middle Road

What if we know the distribution of the inputs?

CASE 1: Sum Computation

- Original sum computation

```
s = 0;
for (i = 0; i < n; i++)    s = s + f(i);
```

- Perforated, extrapolated sum computation

```
s = 0;
for (i = 0; i < n; i += 2)    s = s + f(i);
  s = s * 2;
```

Step 1: Represent Result Difference

- Original sum computation

```
s = 0;
for (i = 0; i < n; i++)    s = s + f(i);
```

- Perforated, extrapolated sum computation

```
s = 0;
for (i = 0; i < n; i += 2)    s = s + f(i);
  s = s * 2;
```

- Perforation noise: $D = s_{original} - s_{perforated}$

Step 2: Probabilistic Modeling

- Original sum computation

```
s = 0;
for (i = 0; i < n; i++)    s = s + f(i);
```

- Perforated, extrapolated sum computation

```
s = 0;
for (i = 0; i < n; i += 2)    s = s + f(i);
  s = s * 2;
```

- Perforation noise: $D = s_{original} - s_{perforated}$

Step 2: Probabilistic Modeling

- Original sum computation

  s = 0;

  for (i = 0; i < n; i++) s = s + x_i;

Specify assumptions

- Perforated, extrapolated sum computation

  s = 0;

  for (i = 0; i < n; i += 2) s = s + x_i;

  s = s * 2;

- Perforation noise: $D = s_{original} - s_{perforated}$

Analysis: Input/Output Relation

Perforation noise:

$$D = s_{\text{original}} - s_{\text{perforated}}$$

Analysis: Input/Output Relation

Perforation noise:

$$D = s_{\text{original}} - s_{\text{perforated}}$$

$$= \quad X_0 + X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 + \ldots$$

$$- 2 \cdot (X_0 + X_2 + X_4 + X_6 + \ldots)$$

Analysis: Input/Output Relation

Perforation noise*:

$$D = s_{\text{original}} - s_{\text{perforated}}$$

$$= \quad X_0 + X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 + \ldots$$

$$- \; X_0 - X_0 - X_2 - X_2 - X_4 - X_4 - X_6 - X_6 - \ldots$$

* Assuming for simplicity that the number of elements is even

Analysis: Input/Output Relation

Perforation noise*:

$$D = s_{\text{original}} - s_{\text{perforated}}$$

$$= \quad X_0 + X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 + \ldots$$

$$- \; X_0 - X_0 - X_2 - X_2 - X_4 - X_4 - X_6 - X_6 - \ldots$$

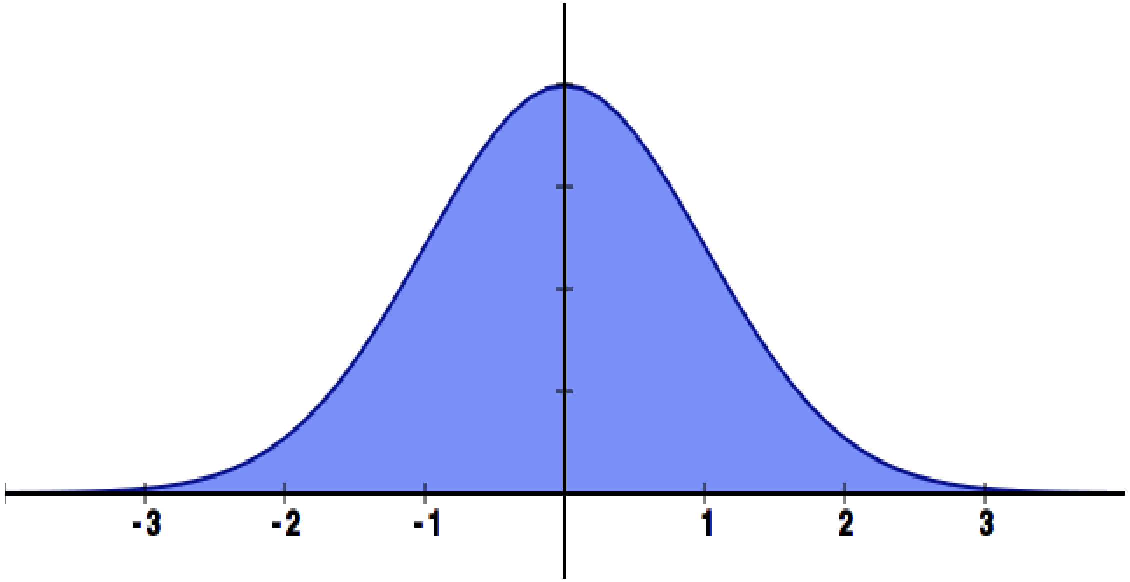
$$= \sum_{0 \le i < \frac{n}{2}} (X_{2i+1} - X_{2i})$$

* Assuming for simplicity that the number of elements is even

Analysis Results

Perforation noise:

$$D = \varphi(X_0, X_2, \ldots, X_{n-1})$$



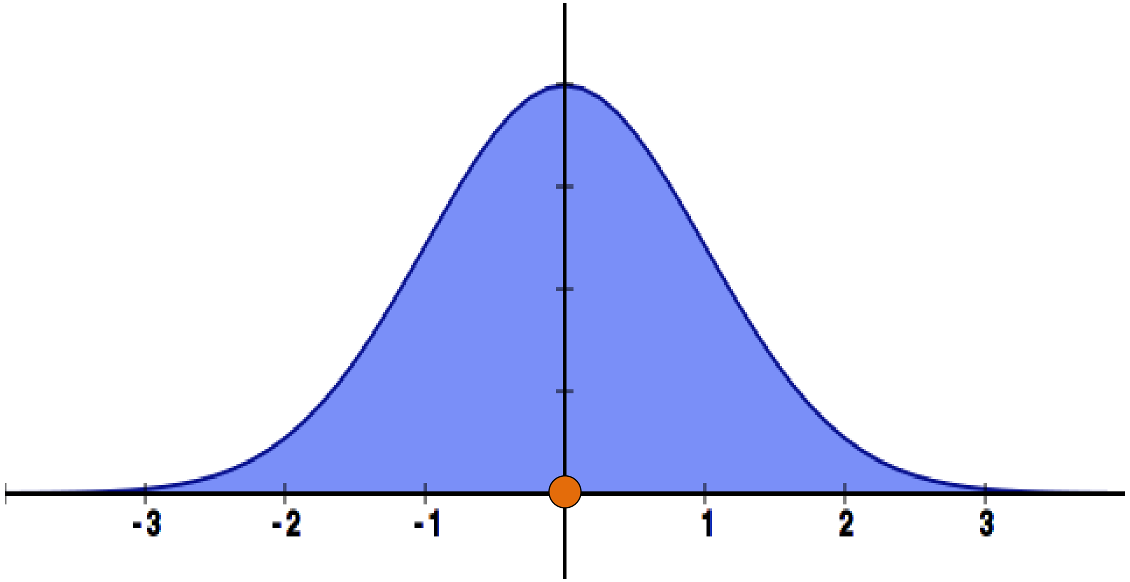
Analysis Results

Perforation noise:

$$D = \varphi(X_0, X_2, \ldots, X_{n-1})$$

Location: Mean

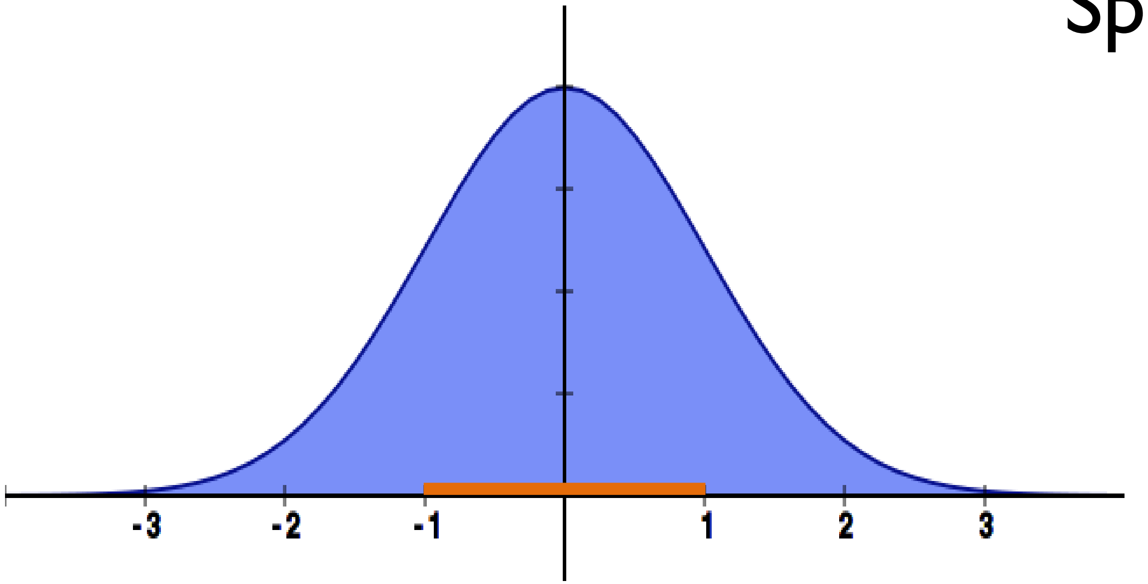
$$E(D) = \mu$$



Analysis Results

Perforation noise:

$$D = \varphi(X_0, X_2, \ldots, X_{n-1})$$



Location: Mean

$$E(D) = \mu$$

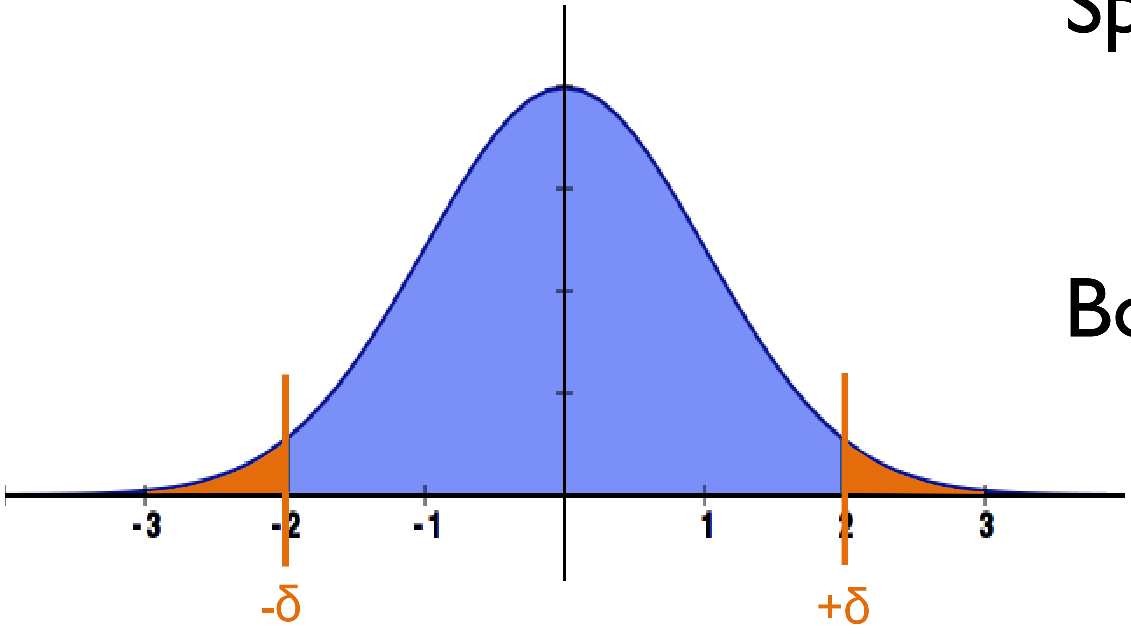
Spread: Variance

$$Var(D) = \sigma^2$$

Analysis Results

Perforation noise:

$$D = \varphi(X_0, X_2, \ldots, X_{n-1})$$



Location: Mean

$$E(D) = \mu$$

Spread: Variance

$$Var(D) = \sigma^2$$

Bound: Distribution tail

$$Pr[\,|D| > \delta\,] < \varepsilon$$

Next Time

Probabilistic programming:
Democratizing probabilistic inference