CS 598sm

Probabilistic &

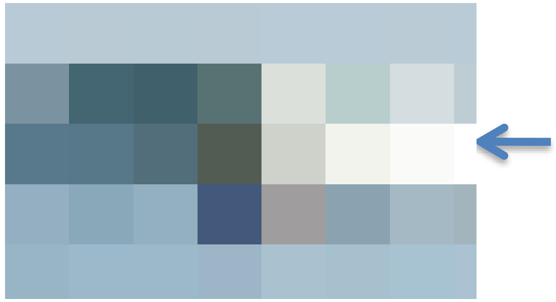
Approximate

Computing

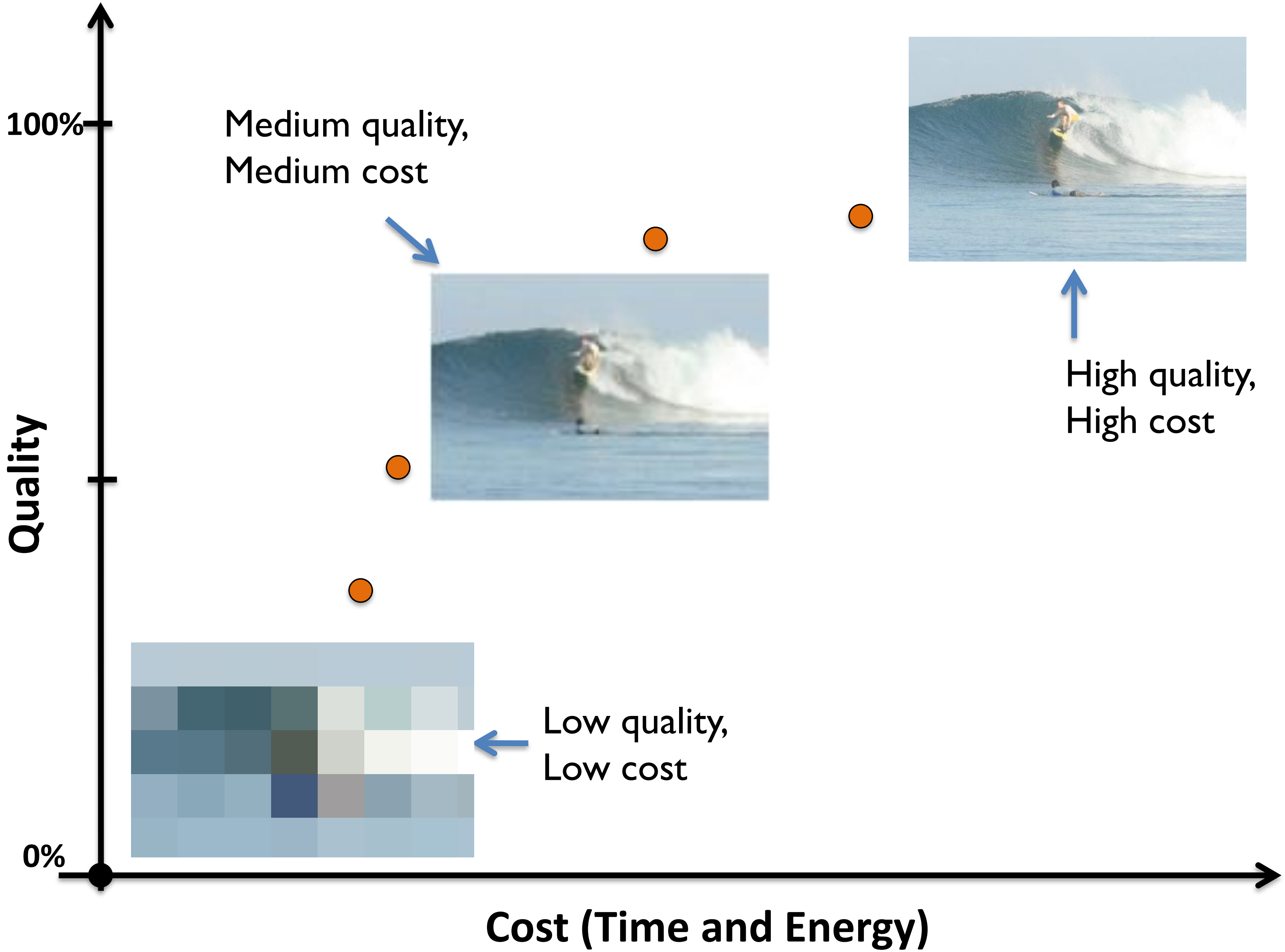
http://misailo.web.engr.Illinois.edu/courses/cs598

Medium quality,
Medium cost

High quality,
High cost



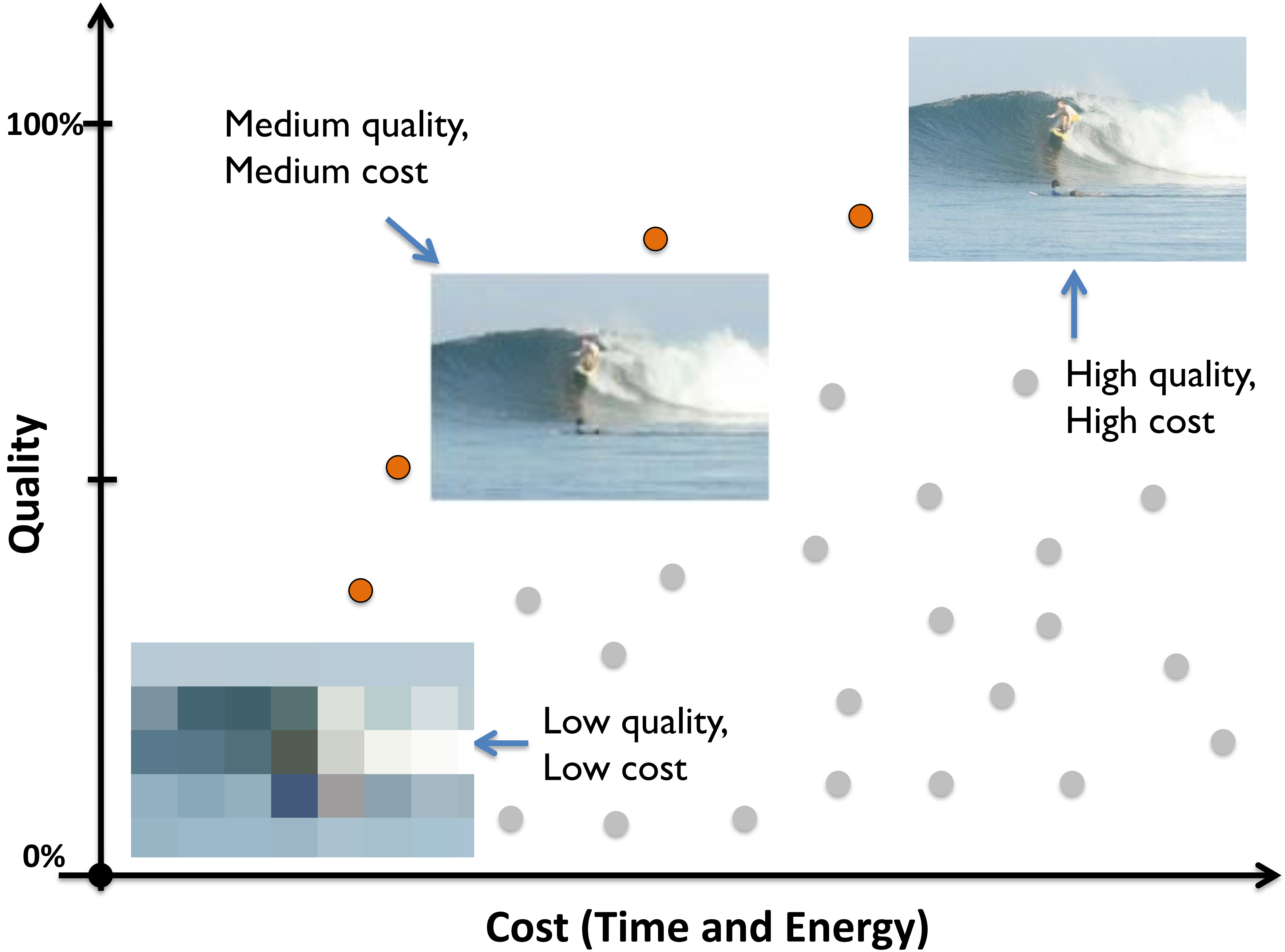
Low quality,
Low cost



Medium quality,
Medium cost

High quality,
High cost

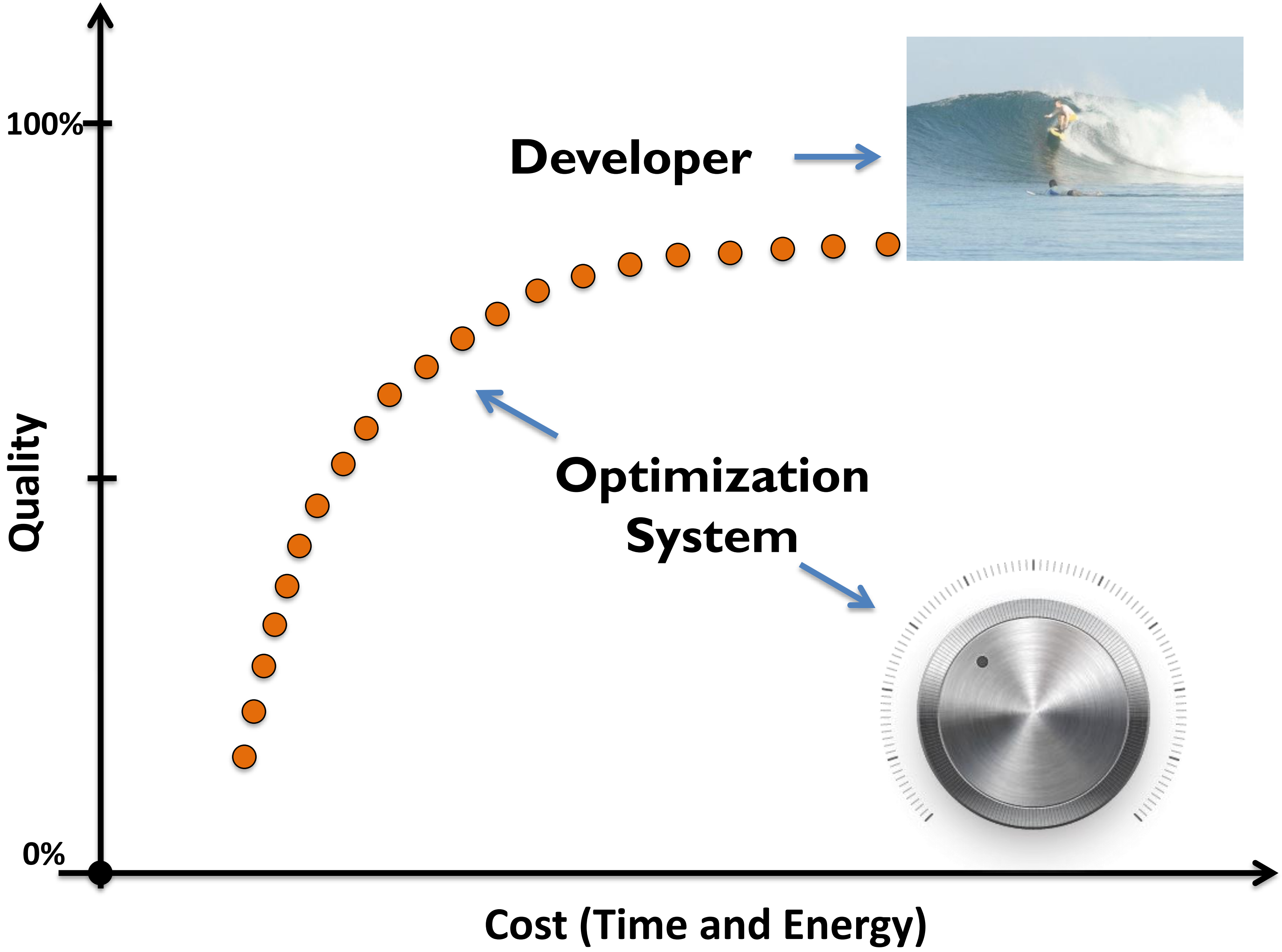
Low quality,
Low cost



Medium quality,
Medium cost

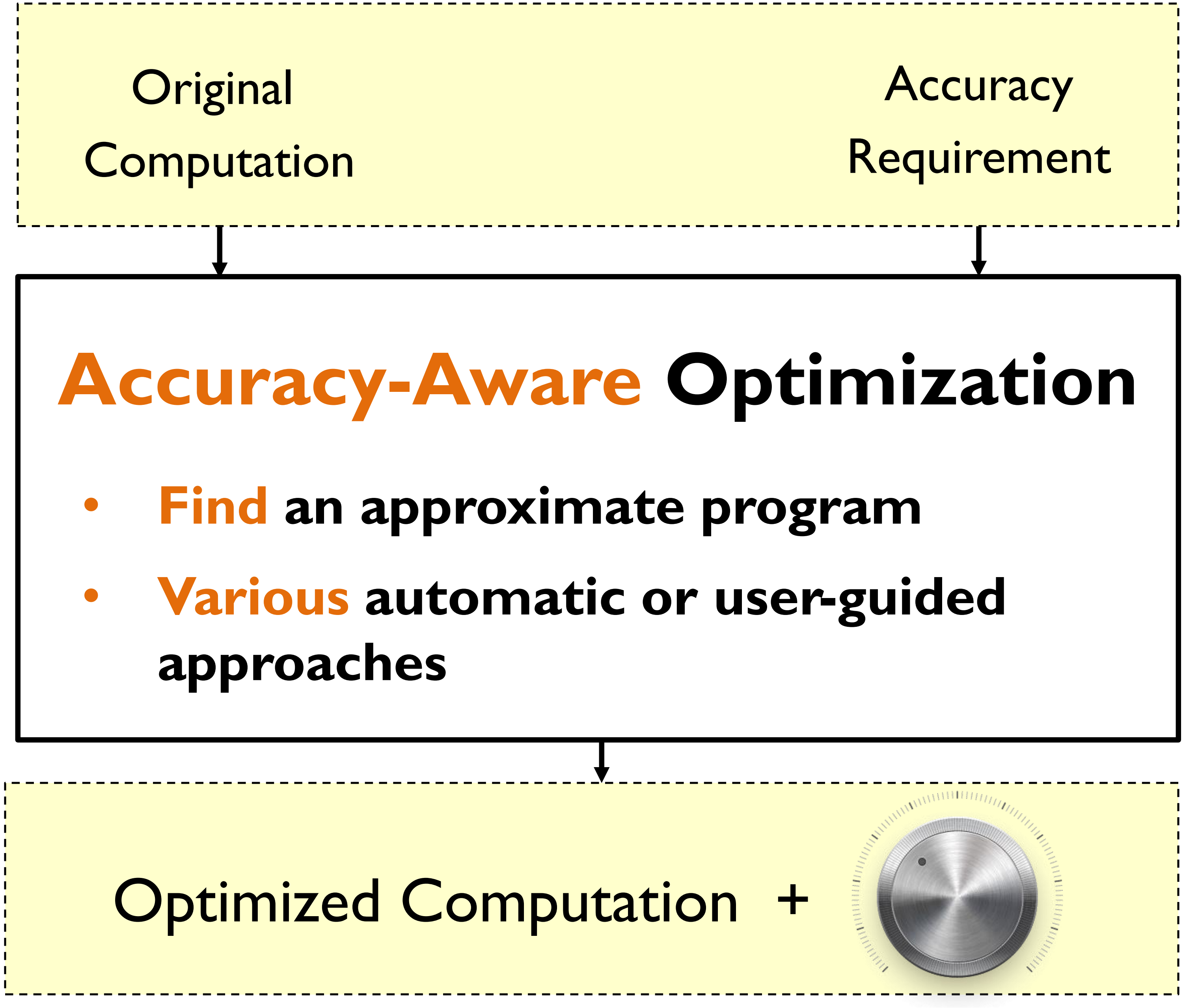
High quality,
High cost

Low quality,
Low cost



Developer

Optimization System



Original Computation

Accuracy Requirement

Accuracy-Aware Optimization

- **Find an approximate program**
- **Various automatic or user-guided approaches**

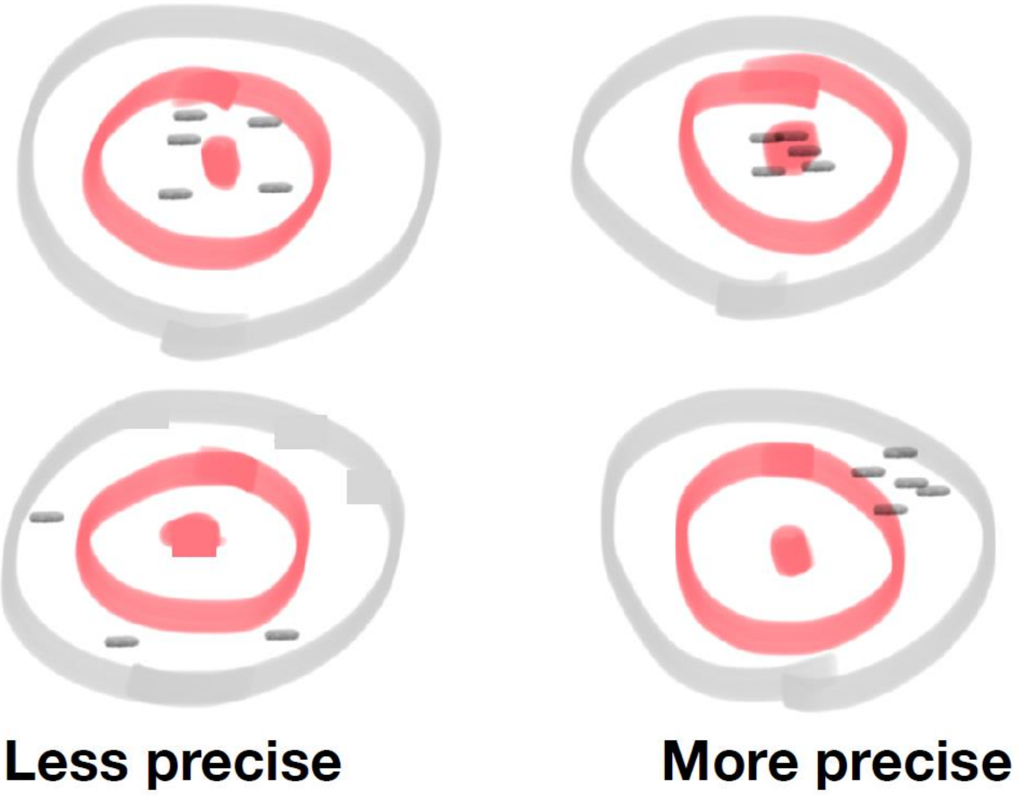
Optimized Computation +

Safari:

ACCURACY ~ CORRECTNESS

Precision

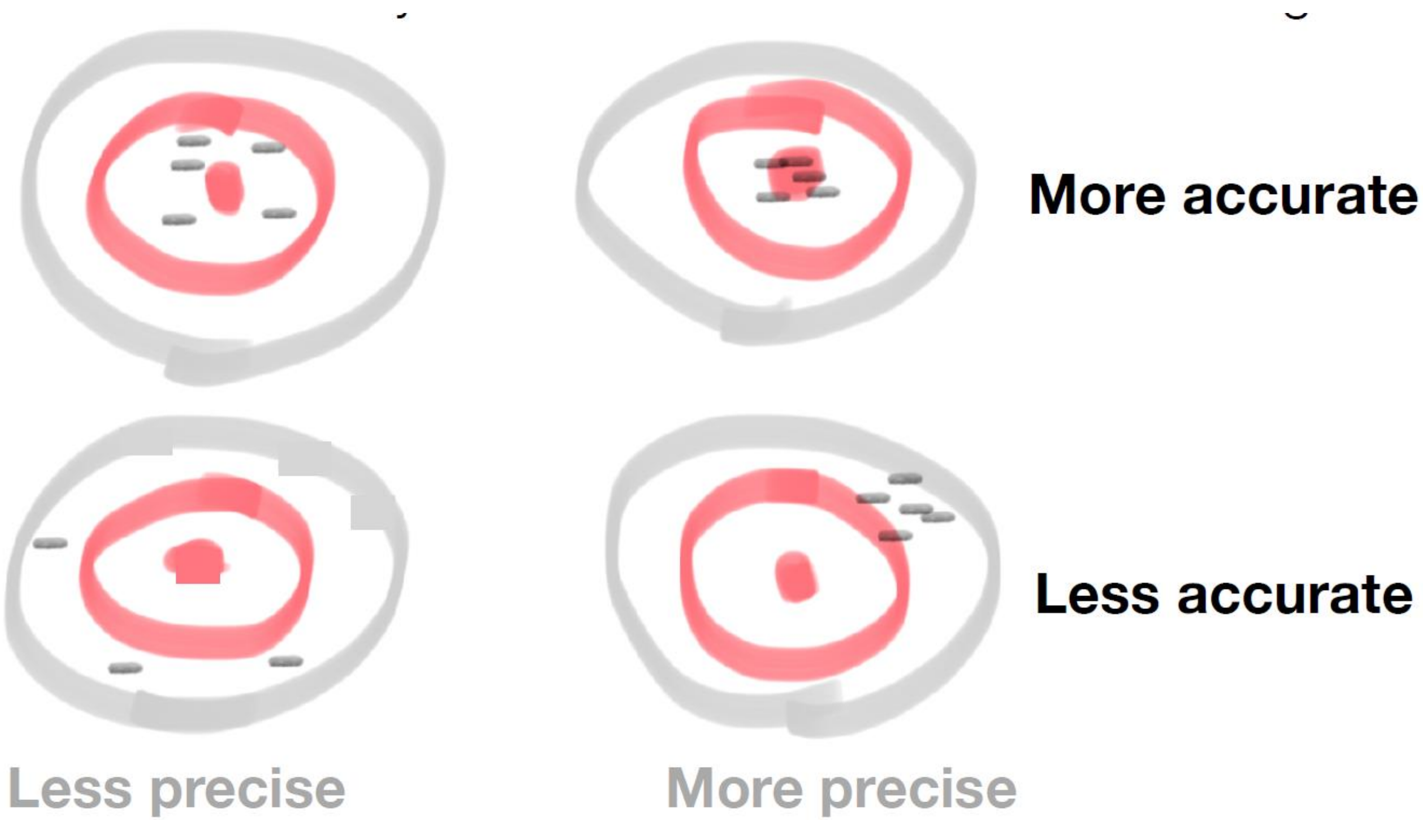
Repeatability or fineness of control



From Phillip Stanley-Marbell, Martin Rinard: Error-Efficient Computing Systems. (2017)

Accuracy

Difference from the correct value



From Phillip Stanley-Marbell, Martin Rinard: Error-Efficient Computing Systems. (2017)

Reliability

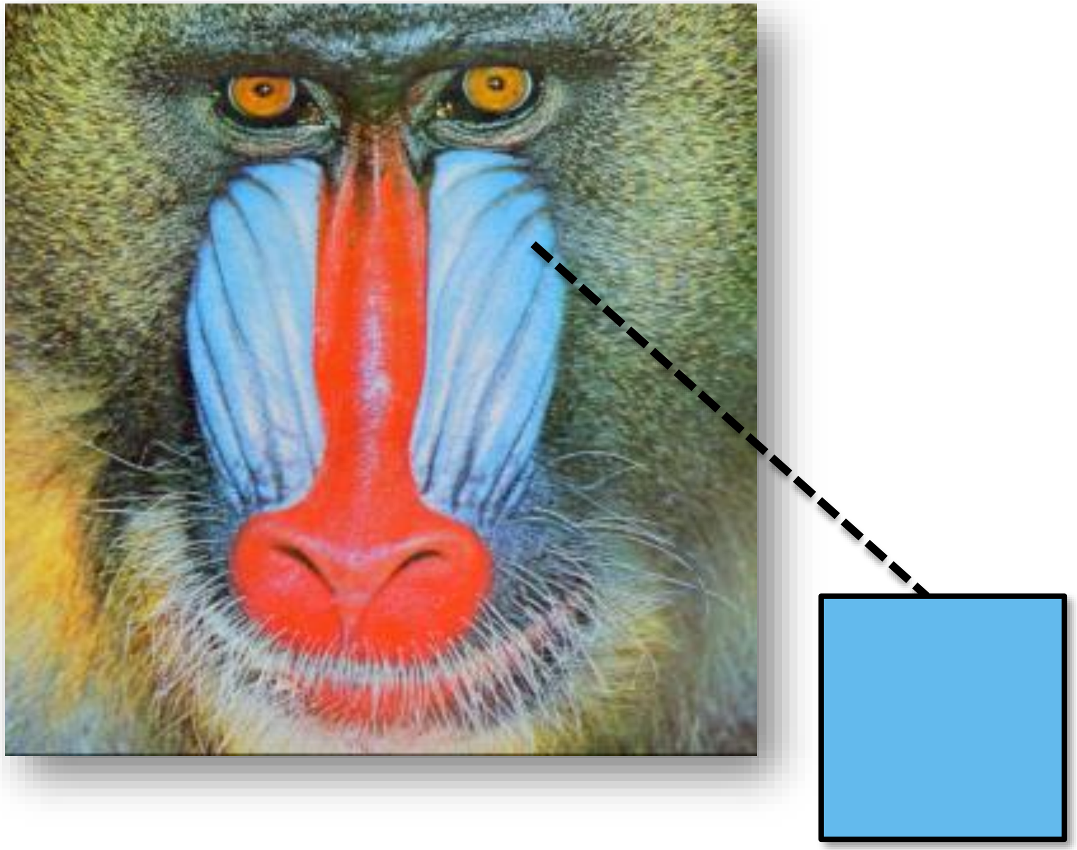
Probability that a system has been functioning correctly, continuously over the time interval [0, *t*]

Conventionally denoted by the function *R*(*t*)

Sometimes we implicitly use without t, meaning that reliability is over the period of operation

From Phillip Stanley-Marbell, Martin Rinard: Error-Efficient Computing Systems. (2017)

Another Thought Experiment

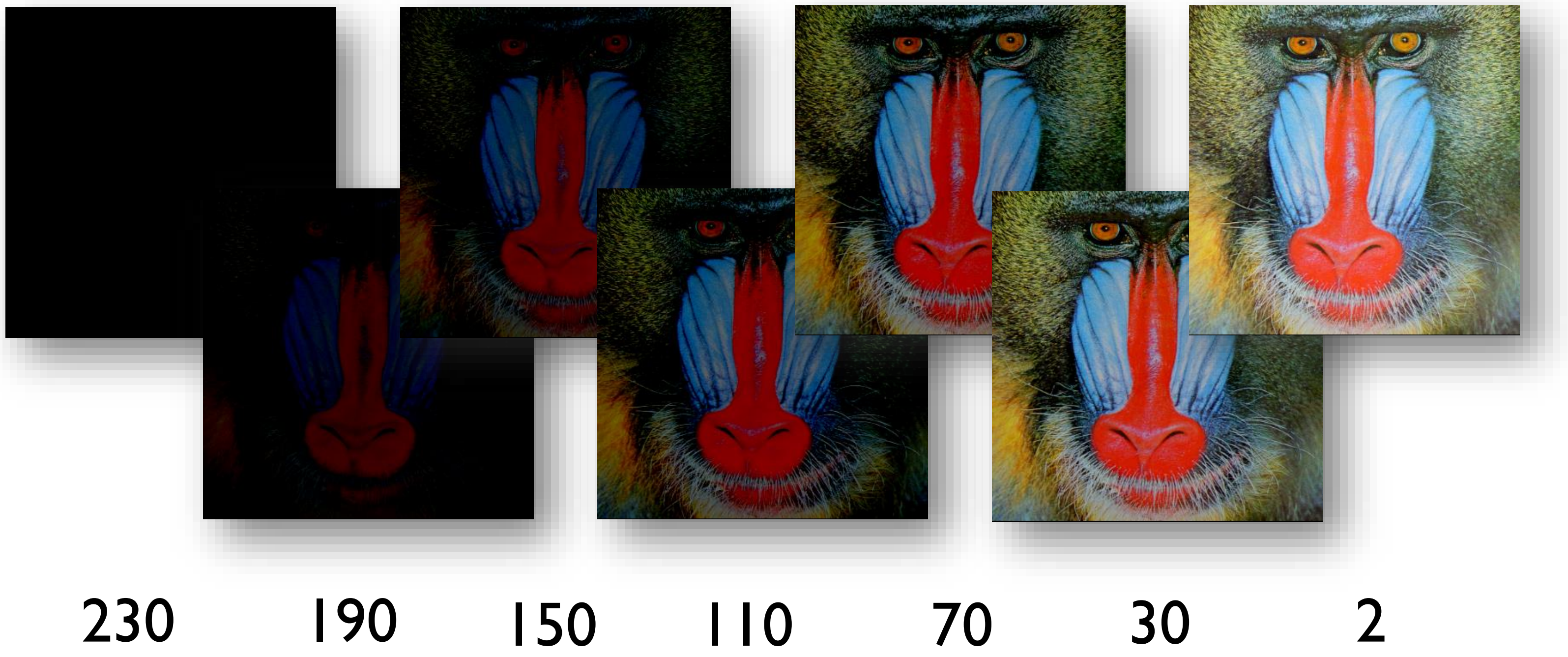


R	99
G	186
B	237

What if we change magnitude of the pixel?
What if we change frequency of the pixel (sometimes it's just black)?

Function’s and Program’s Accuracy

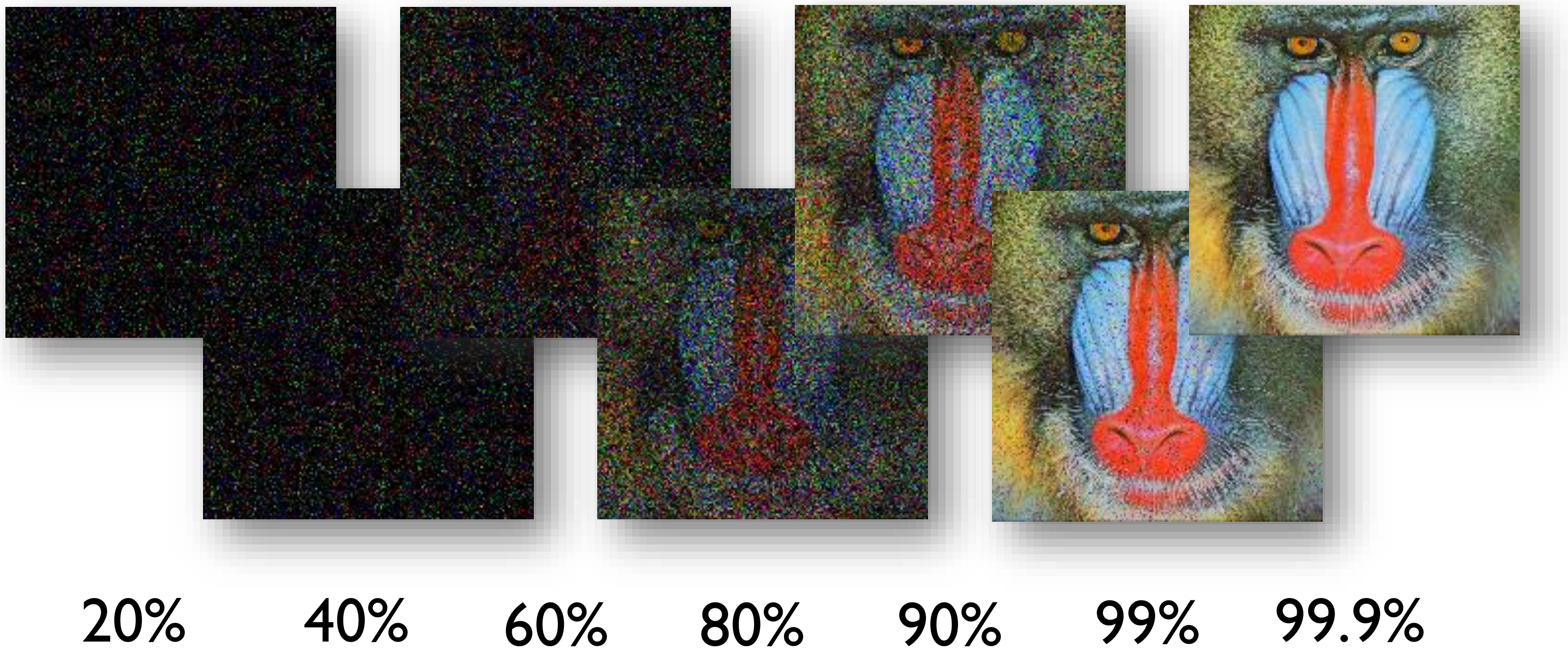
Magnitude of Noise



Difference *d* between the exact and approximate pixel values that interpolation kernel produces (for all color components)

Function’s and Program’s Accuracy

Frequency of Noise



Probability p with which interpolation kernel produces the correct pixel

We observe

Small Errors

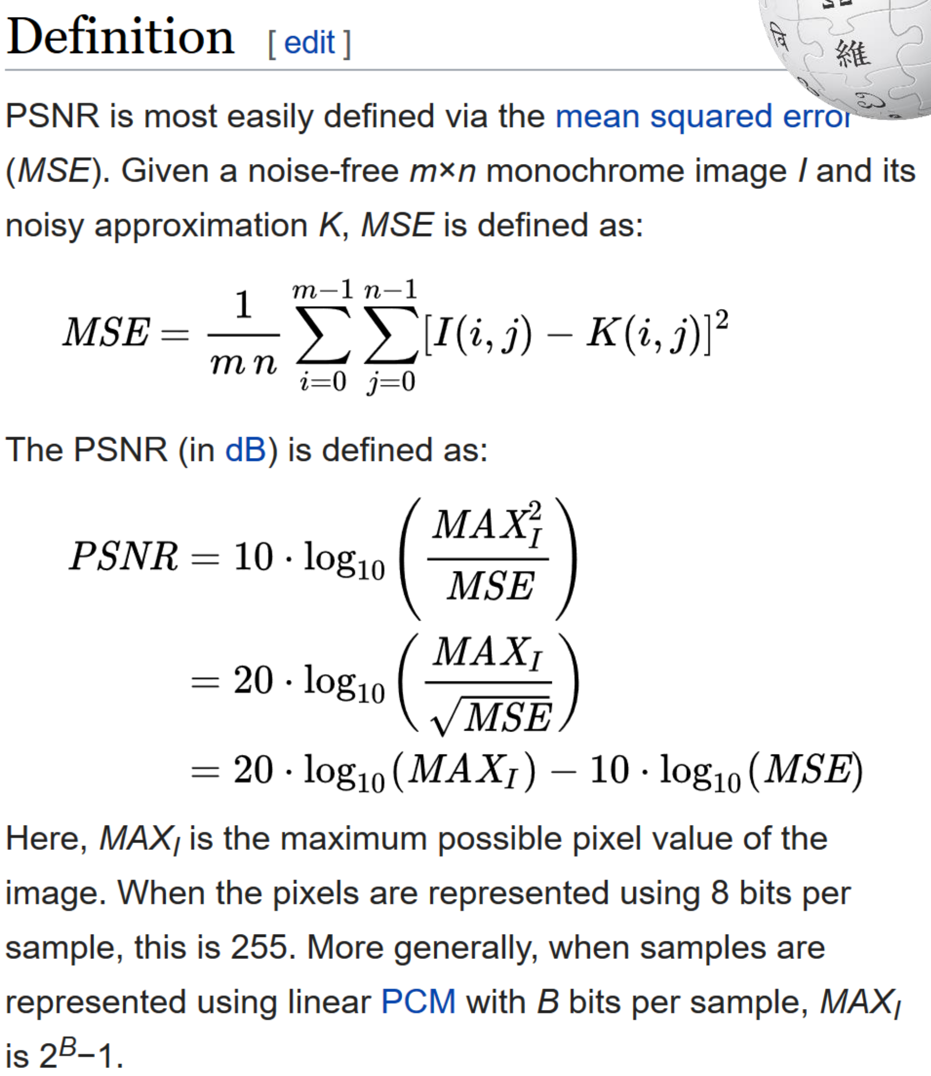
Most of the Time

Accuracy Requirement

Specify Metric and Threshold

- **Each application has its own**
- Requires ~~domain~~ problem expertise
- For visual data, historically PSNR has often been used (with all its imperfections)
- But one can think of other better perceptory metrics

More details on the roles of metrics:
Karpuzcu et al., On Quantification of Accuracy Loss in Approximate Computing, WDDD 2015.



Definition [edit]

PSNR is most easily defined via the mean squared error (*MSE*). Given a noise-free *m*×*n* monochrome image *I* and its noisy approximation *K*, *MSE* is defined as:

$$MSE = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

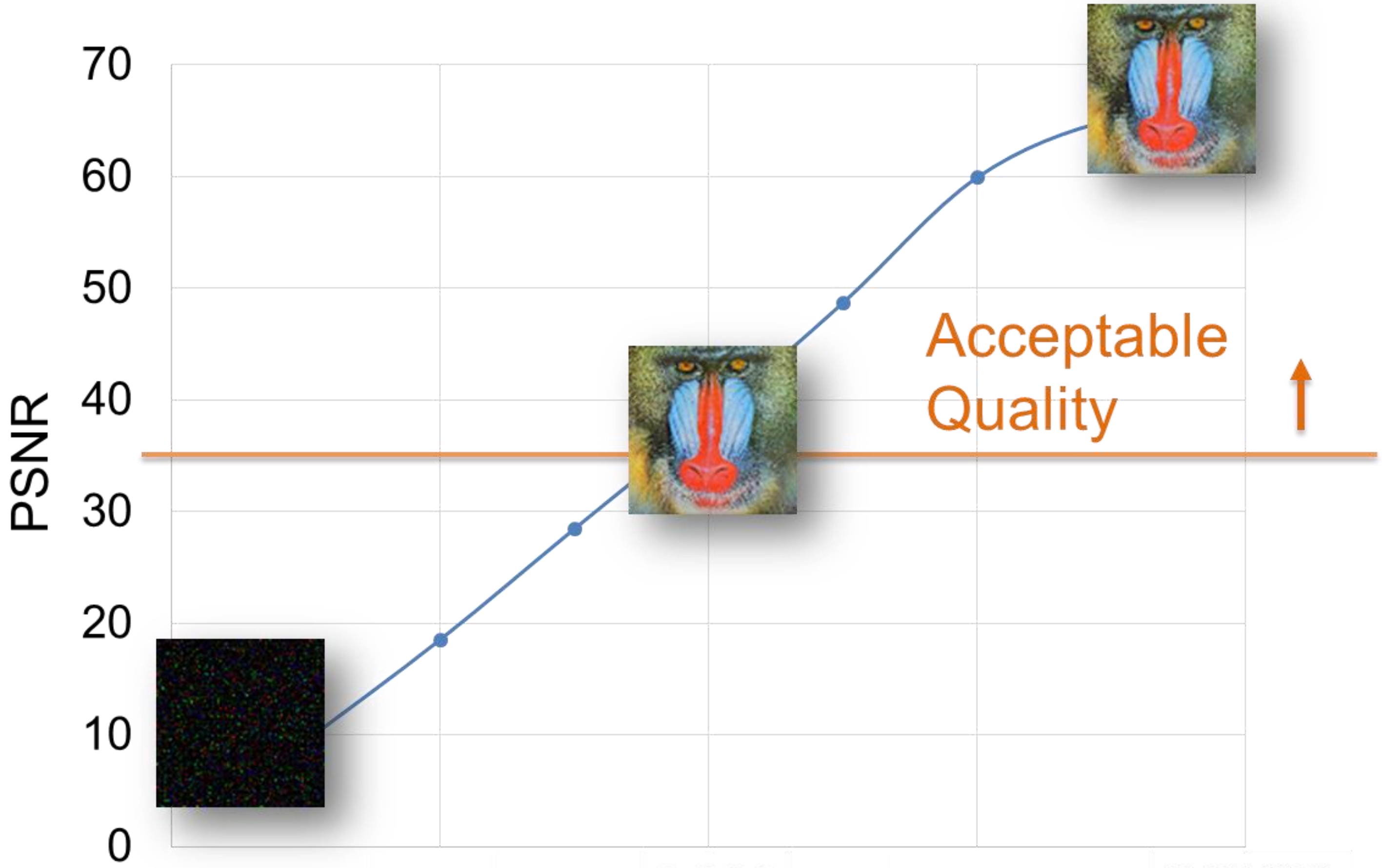
The PSNR (in dB) is defined as:

$$\begin{aligned} PSNR &= 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \\ &= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \\ &= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE) \end{aligned}$$

Here, MAX_I is the maximum possible pixel value of the image. When the pixels are represented using 8 bits per sample, this is 255. More generally, when samples are represented using linear PCM with *B* bits per sample, MAX_I is 2^B-1.

Accuracy Requirement

Specify Metric and Threshold



Accuracy Specifications

End-to-end: program output

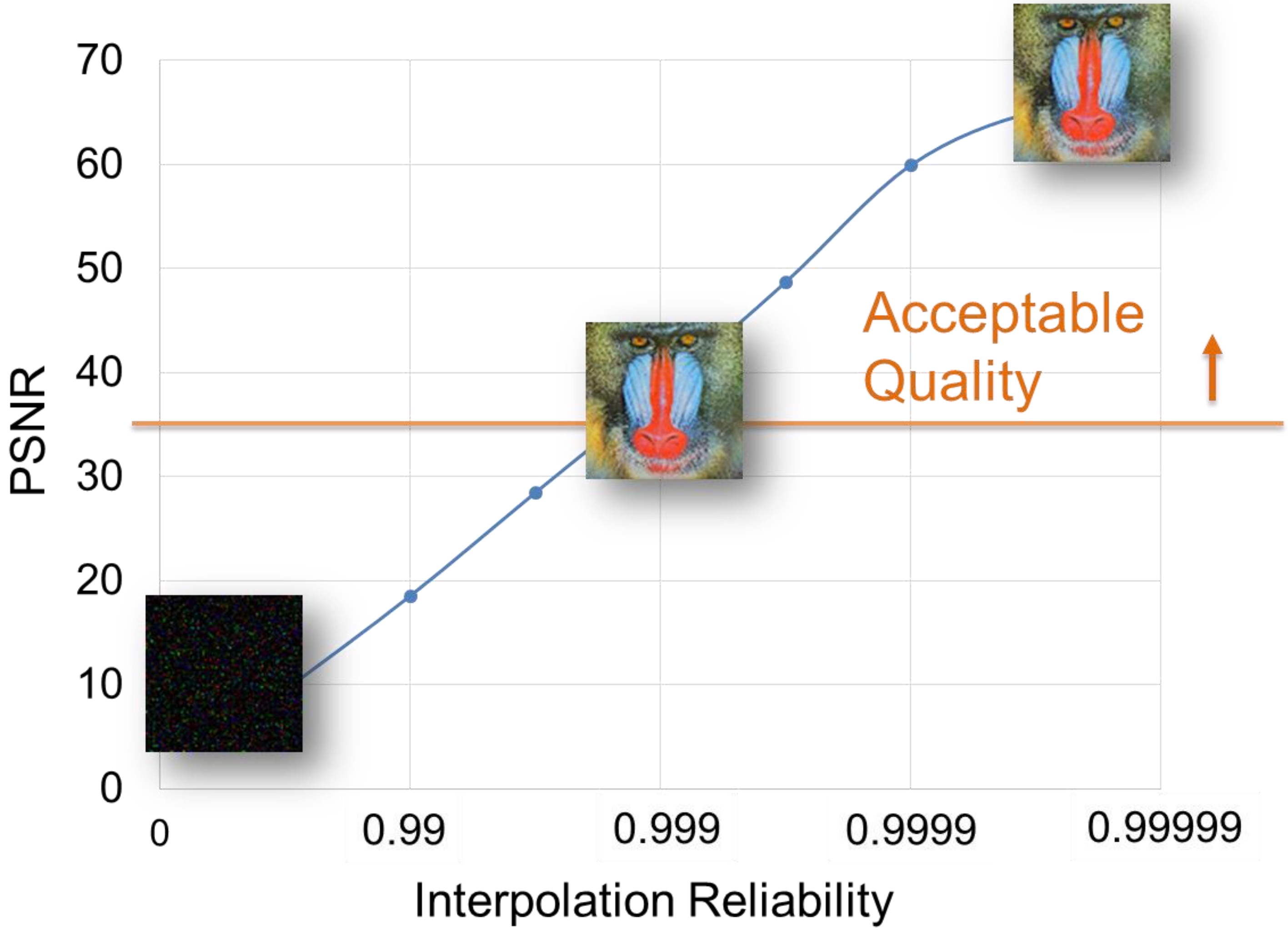
- You can compare outputs only at the end of the run
- Often better understood for representative domains

Kernel-level: each function has it specification

- Fine-grained control + checking of intermediate results
- Often ad-hoc or not intuitive
- While in general can lead to composition, hard to propagate all errors

Accuracy Requirement

Specify Metric and Threshold



Analytic Derivation

Use properties of the algorithm and implementation

Local Specification:	Kernel computes the pixel with reliability ***r***
Global Specification:	PSNR of the image
Computation Pattern:	Data parallel loop

$$PSNR(D, D') = 20 \cdot \log(255) - 10 \cdot \log\left(\frac{1}{h \cdot w} \sum_{i,j} (D_{ij} - D'_{ij})^2\right)$$

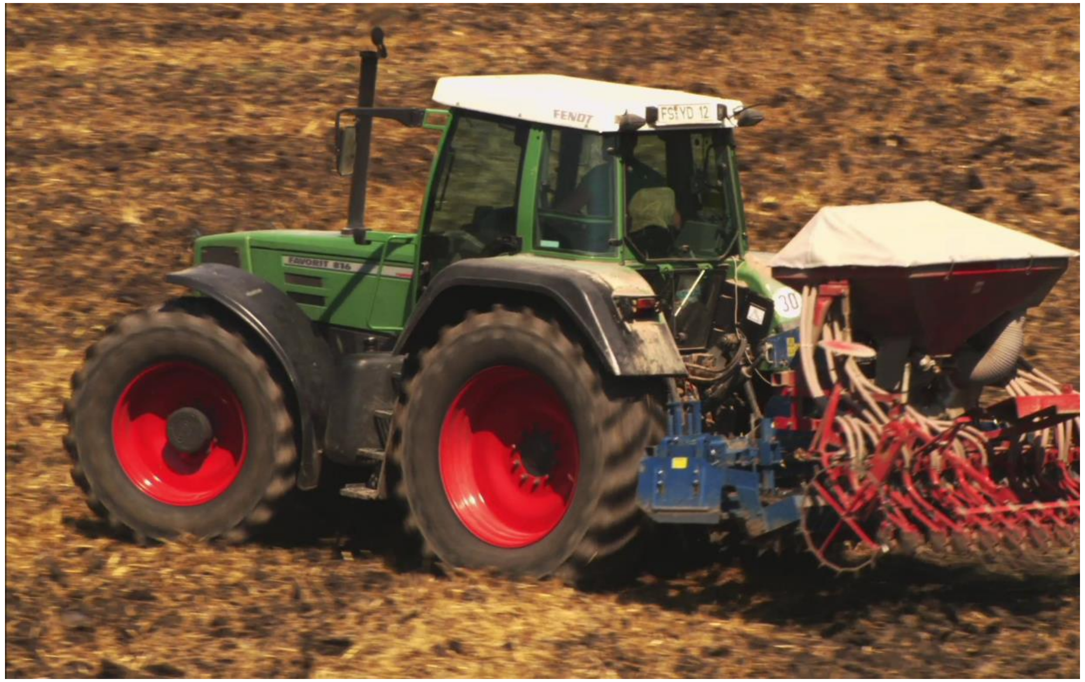
$$\boxed{\boldsymbol{r} \cdot 0 + (1 - \boldsymbol{r}) \cdot 255}$$

Analytic Derivation

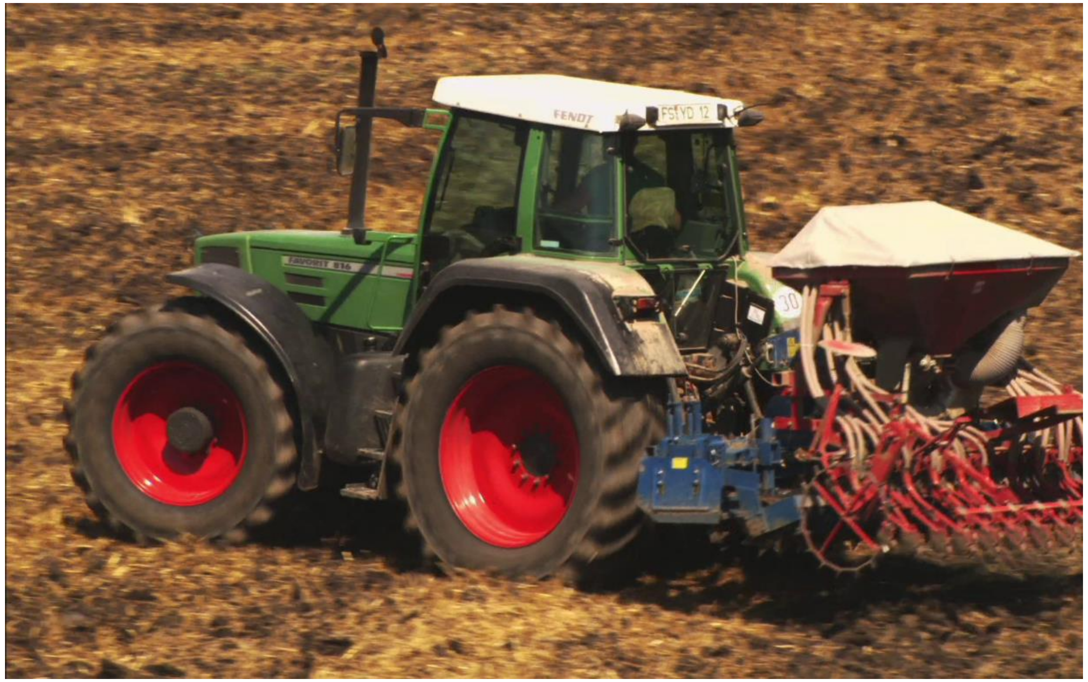
Use properties of the algorithm and implementation

Local Specification:	Pixel kernel reliability ***r***
Global Specification:	PSNR of the image
Computation Pattern:	Data parallel loop

$$\boxed{\mathbb{E}[PSNR(D, D')] \geq -10 \cdot \log(1 - \boldsymbol{r})}$$

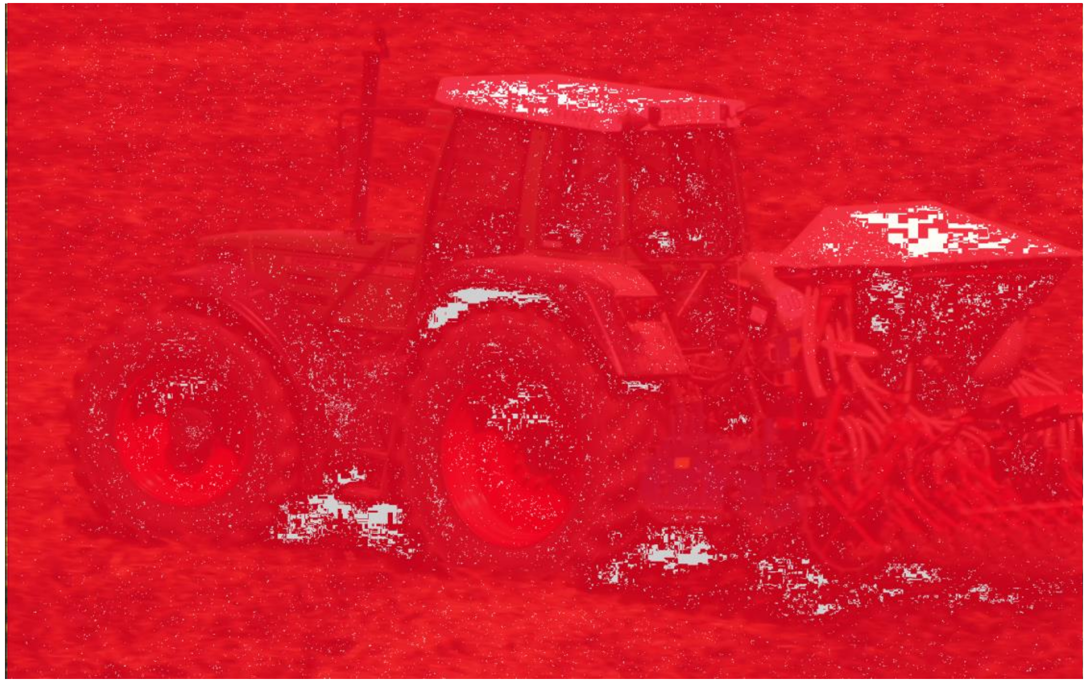


Original



Perforated

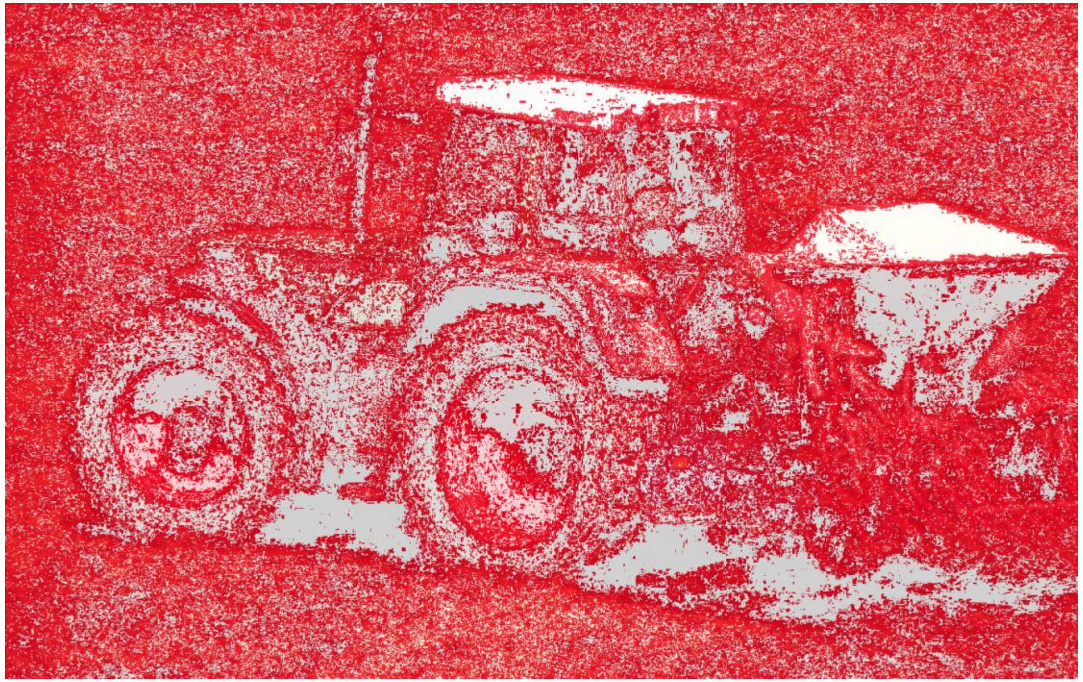
Original



Perforated

Any pixel difference

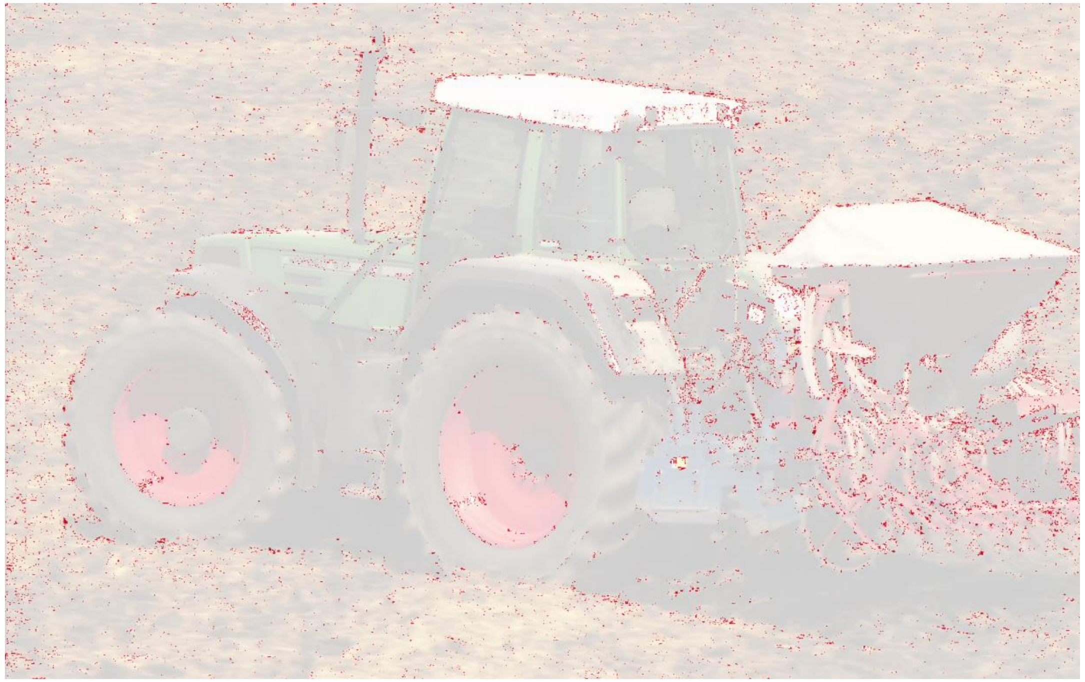
Original



Perforated

> 1% pixel difference

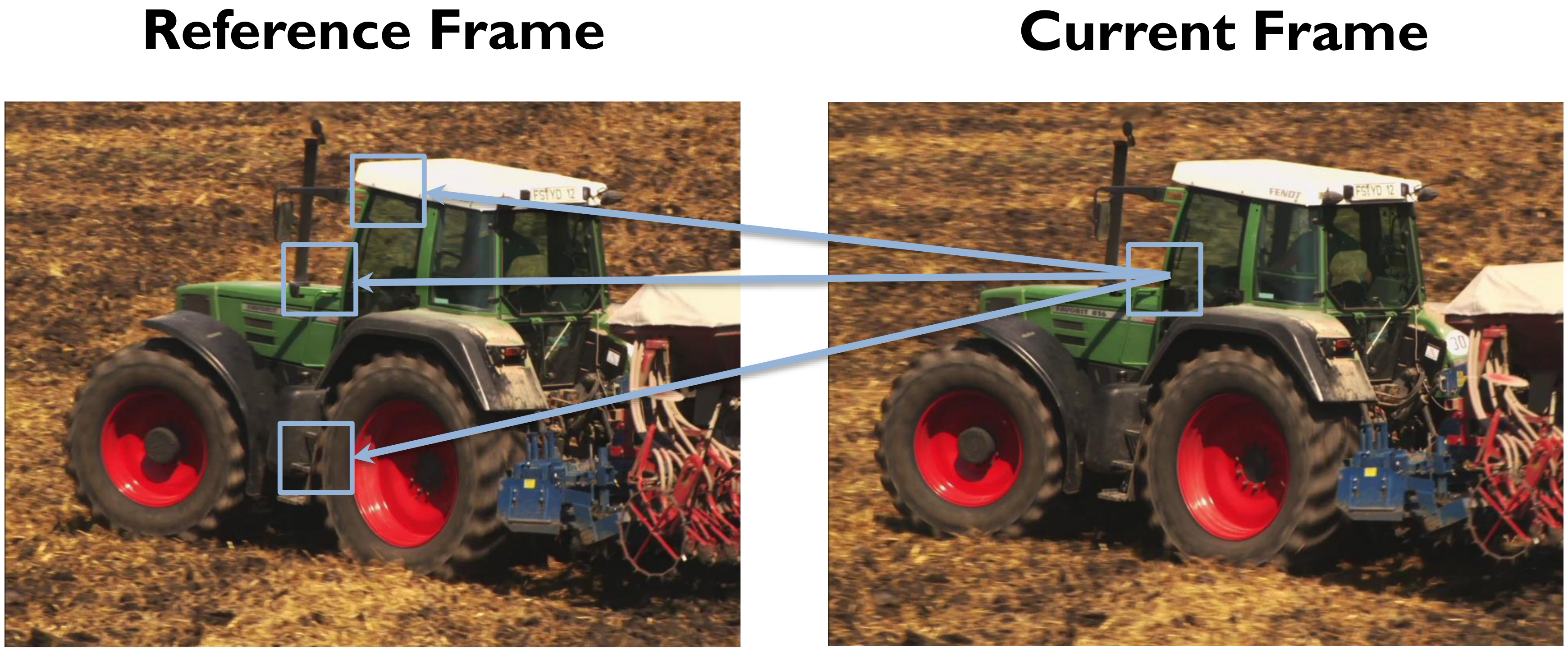
Original



Perforated

> 5% pixel difference

x264 Motion Estimation



Reference Frame

Current Frame

x264 Block Matching

```
score = 0;

for (i = 0; i < block_height; i++) {
   for (j = 0; j < block_width; j++) {

     idx1 = IDX(i, j, cur_start);
     idx2 = IDX(i, j, prev_start);
     diff = cur_frame[idx1] - prev_frame[idx2];
     adif = abs(diff);
     score = score + adif;

   }
}

return score;
```

x264 Block Matching

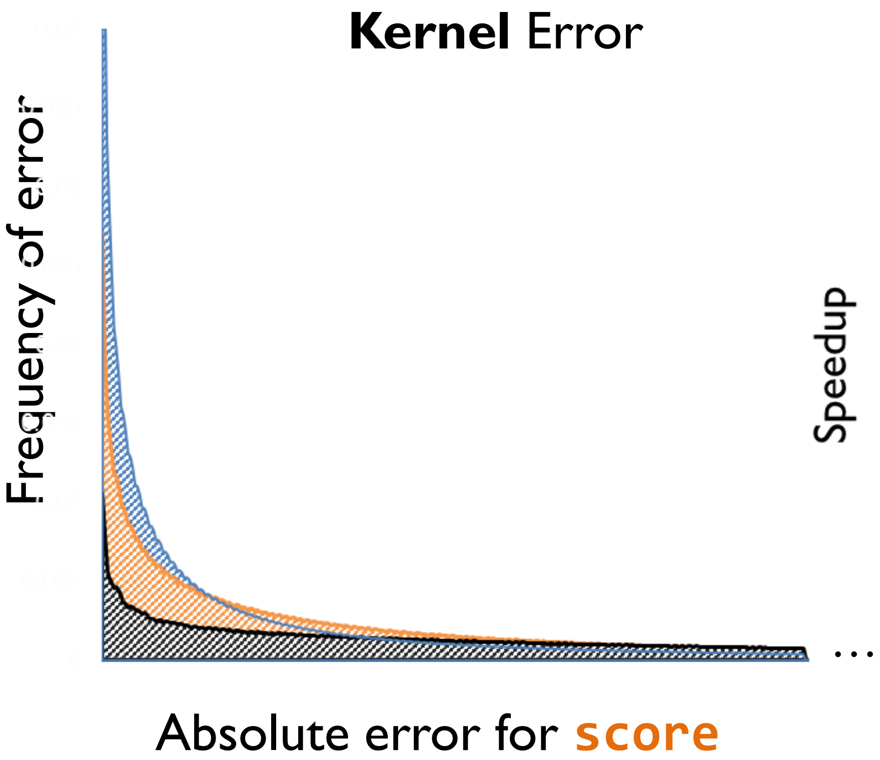
```
score = 0;

for (i = 0; i < block_height; i+=2) {
   for (j = 0; j < block_width; j+=2) {

     idx1 = IDX(i, j, cur_start);
     idx2 = IDX(i, j, prev_start);
     diff = cur_frame[idx1] - prev_frame[idx2];
     adif = abs(diff);
     score = score + adif;

   }
}

return score;
```

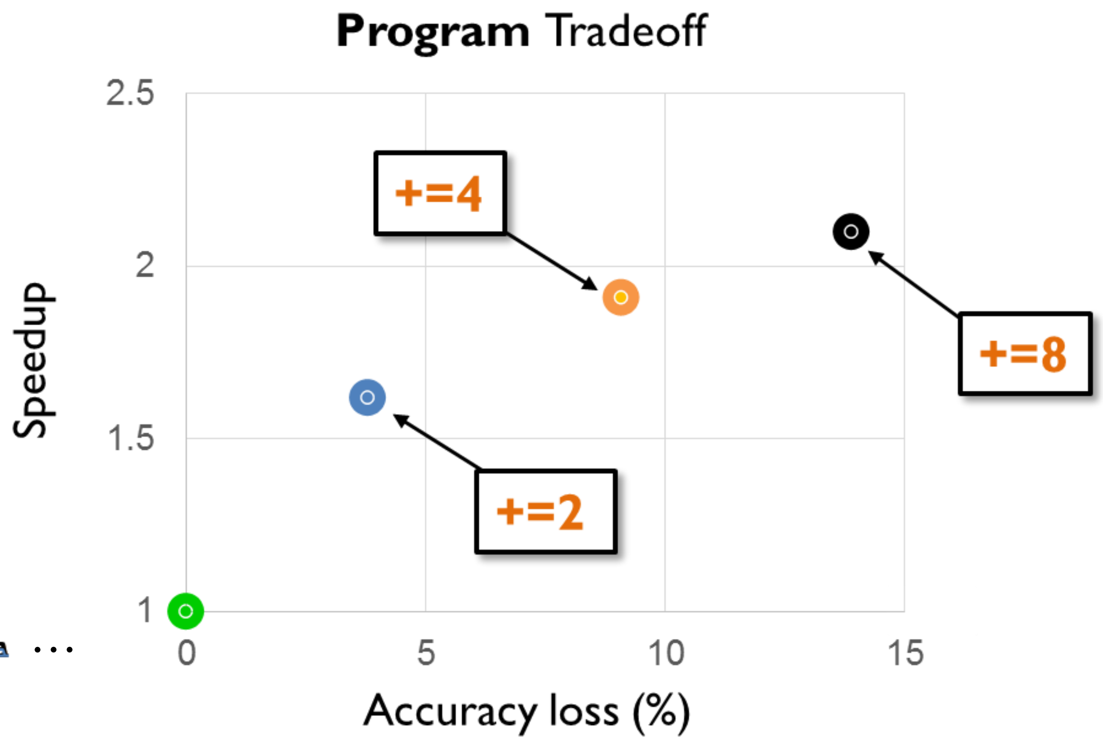
x264 Block Matching

```
score = 0;

for (i = 0; i < block_height; i+=2) {
   for (j = 0; j < block_width; j+=2) {

     idx1 = IDX(i, j, cur_start);
     idx2 = IDX(i, j, prev_start);
     diff = cur_frame[idx1] – prev_frame[idx2];
     adif = abs(diff);
     score = score + adif;

   }
}

return score * 4;
```

Absolute Error of Perforation

With Bias Compensation

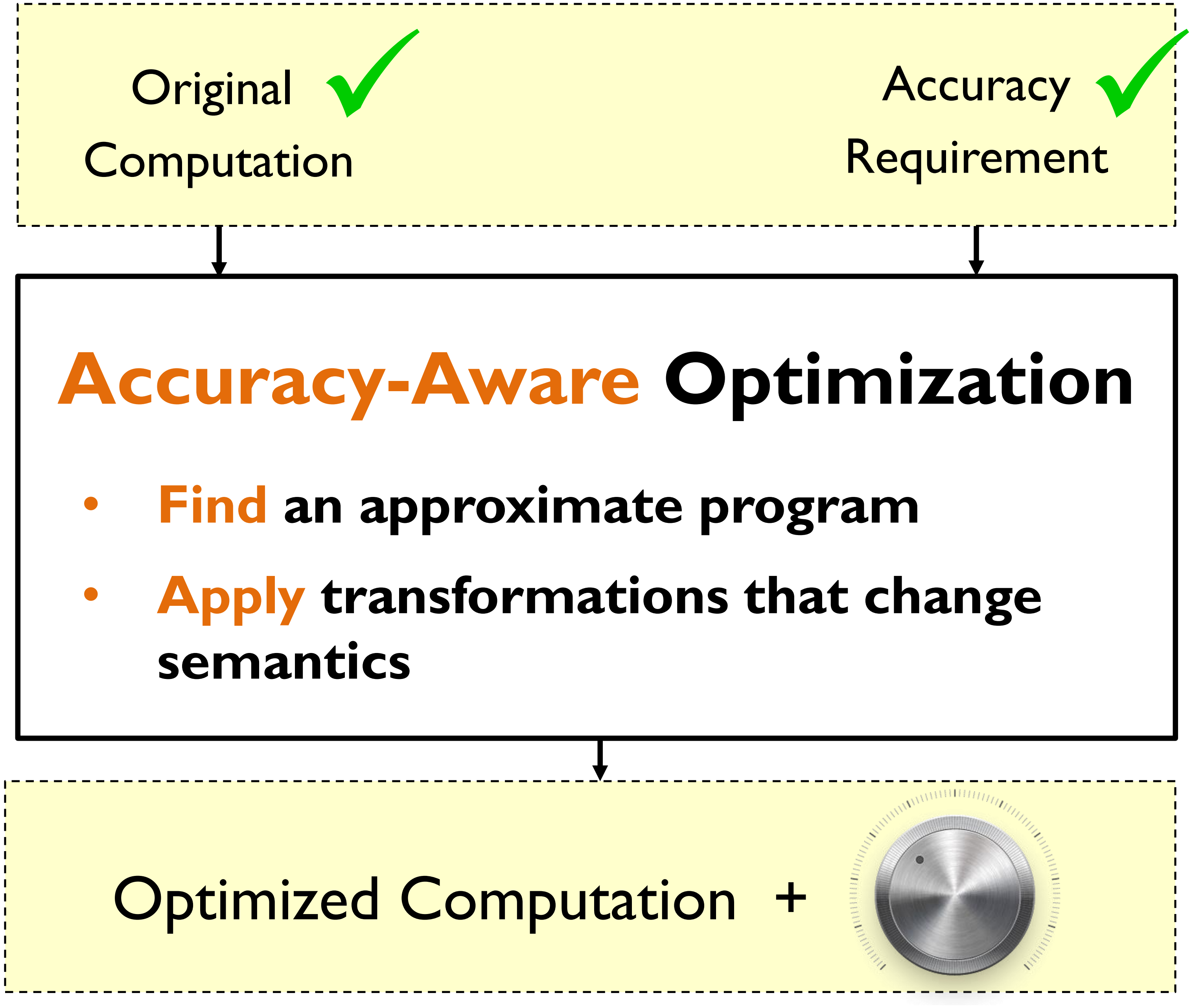




Most of the time errors of individual approximation computations are small!

Several Patterns Amenable to Approximation

- Map
- Reduce (sum, average, min, max, median)
- Stencil
- Scatter/Gather
- Iterative refinement loop
- …



Original Computation ✓

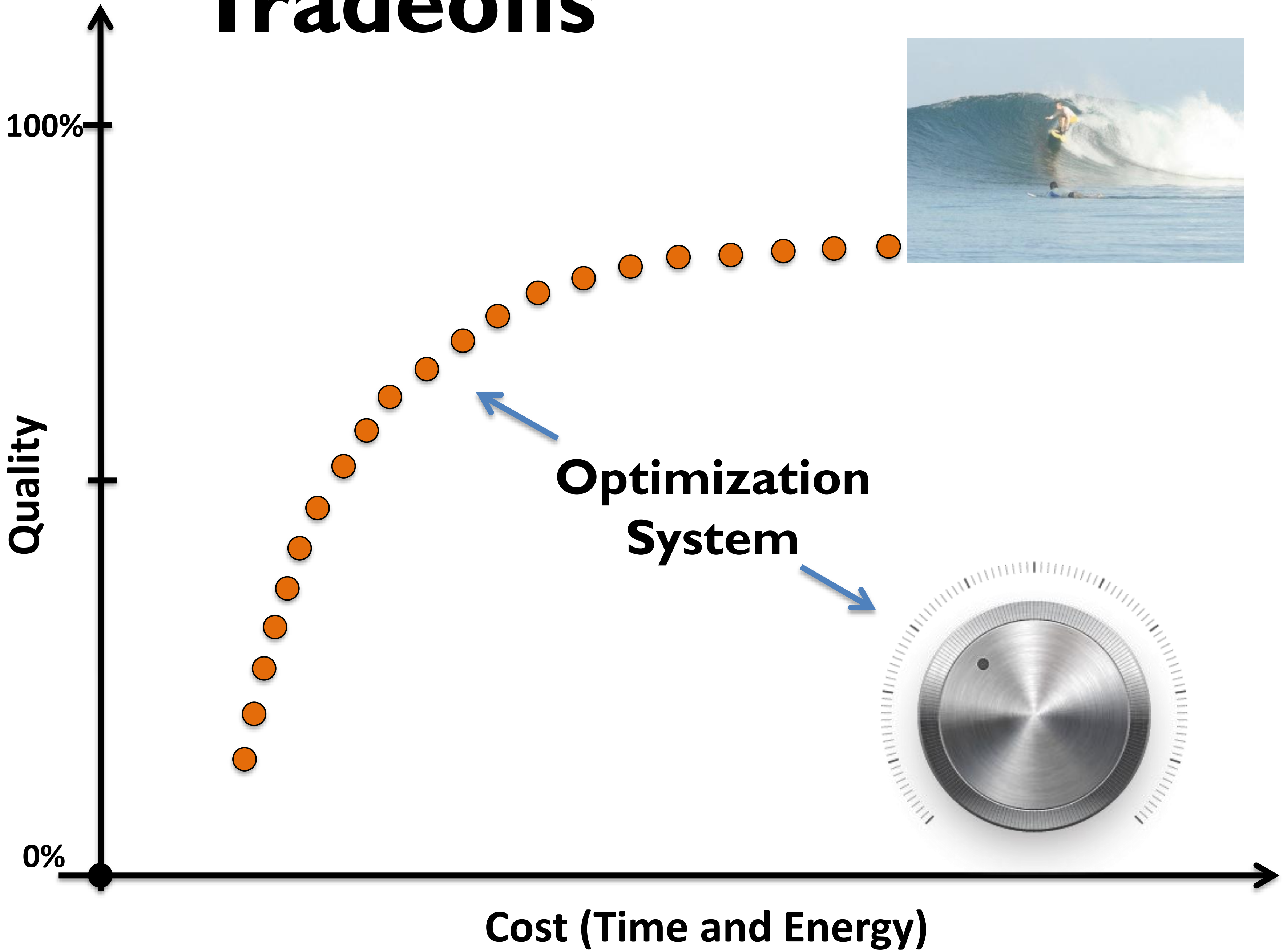
Accuracy Requirement ✓

Accuracy-Aware Optimization

- **Find an approximate program**
- **Apply transformations that change semantics**

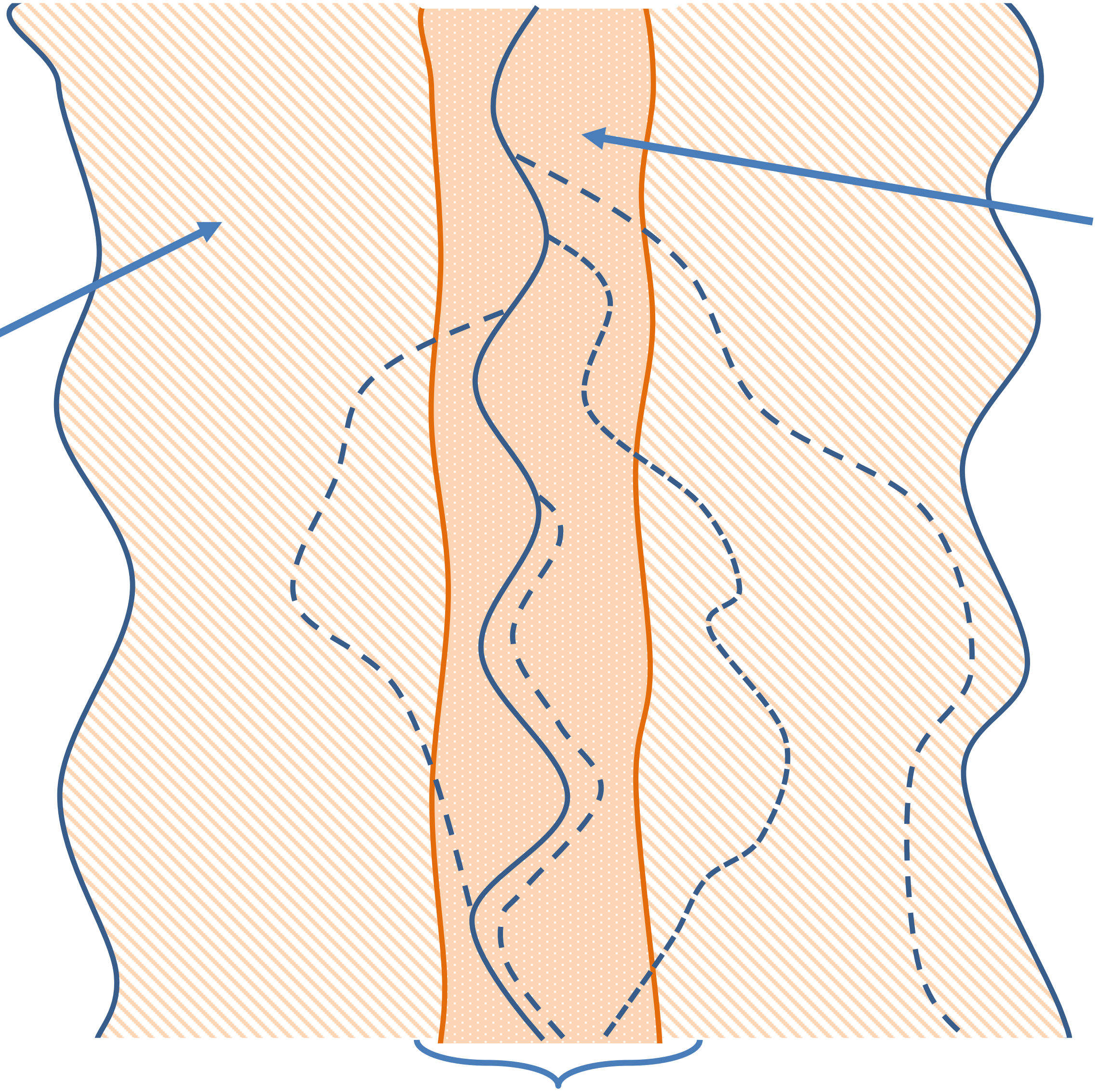
Optimized Computation +

Tradeoffs



Optimization System

Key Intuition



Original Program Execution

Transformations and induce a space of approximate executions

Many of these executions will be similar to the original execution

Sometimes, we can enforce that approximate programs must always execute near the original.

It can help the analysis, but is not necessary.

We want their final results to be similar (i.e., low accuracy loss)
Ideally, we want the execution that runs the fastest

General Optimization Problem

Select Program Configuration $X \in Configs$ to

$$\begin{aligned}&\textbf{maximize}\ \left(Speedup(X,i), Accuracy(X,i)\right)\\&\textbf{forall}\ i \in InputSet\end{aligned}$$

But these are most often competing objectives.

Rephrase: for every accuracy loss threshold δ

$$\begin{aligned}&\textbf{maximize}\ Speedup(X,i)\\&\textbf{subject to}\ AccuracyLoss(X,i) \leq \delta\\&\textbf{forall}\ i \in InputSet\end{aligned}$$

Multiobjective Optimization

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

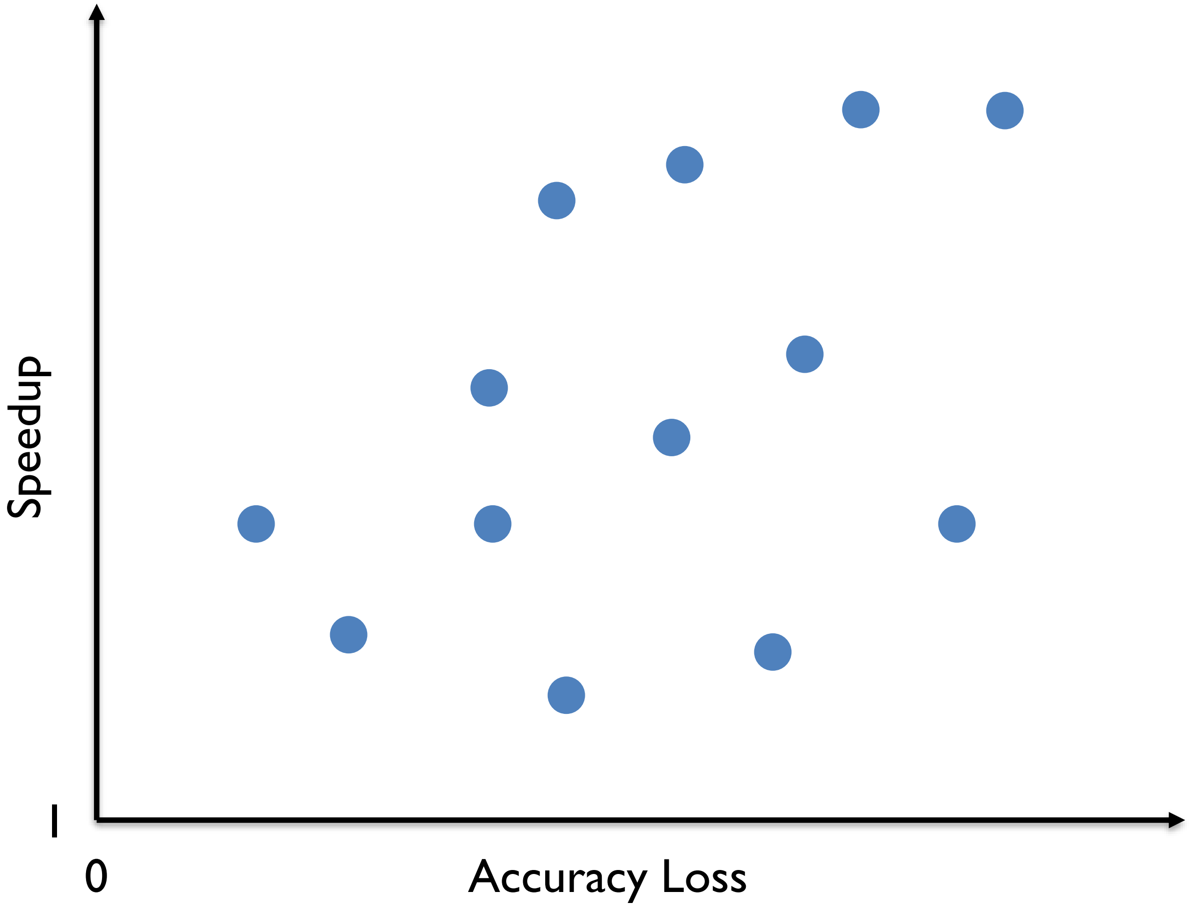
Two candidate program configurations X and Y:

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

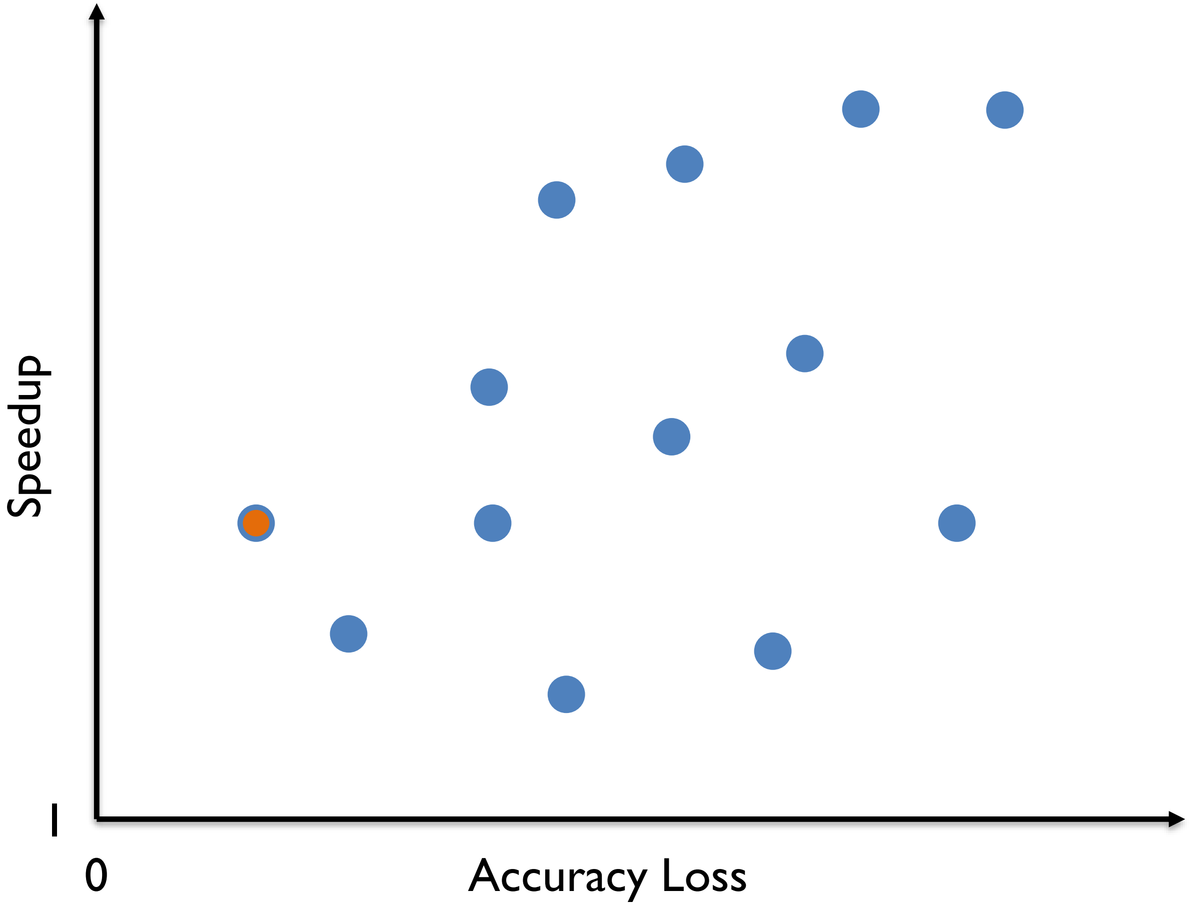
Pareto frontier: the set of points that are not dominated by other points

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

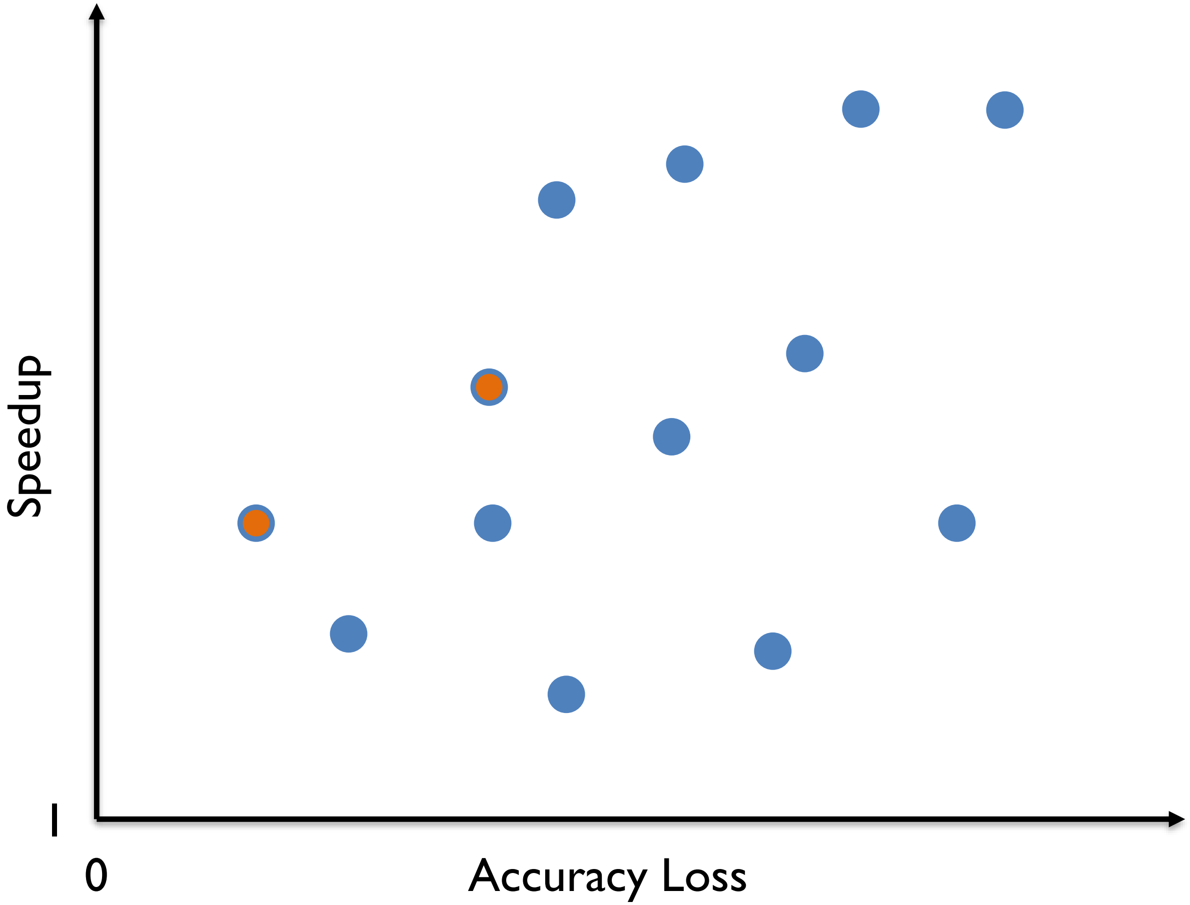
Example



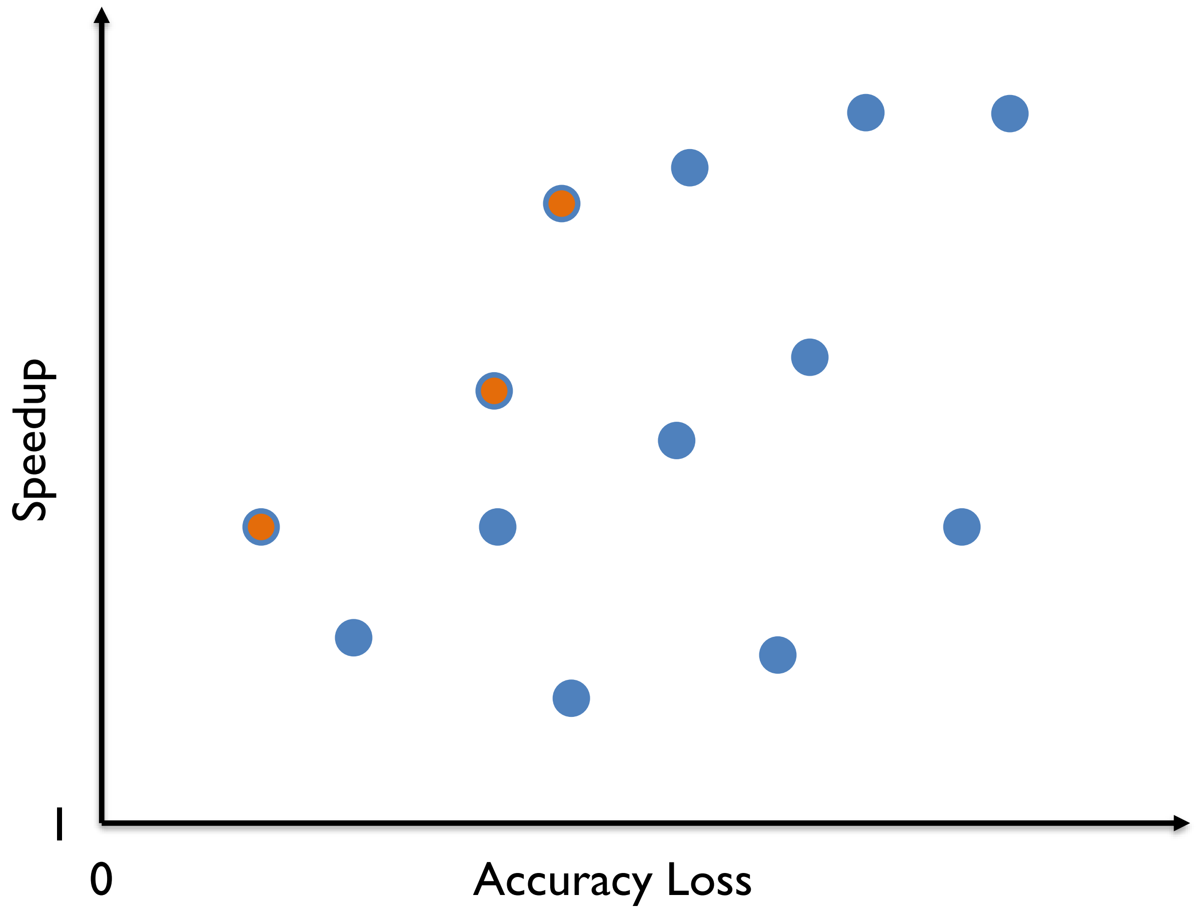
Example



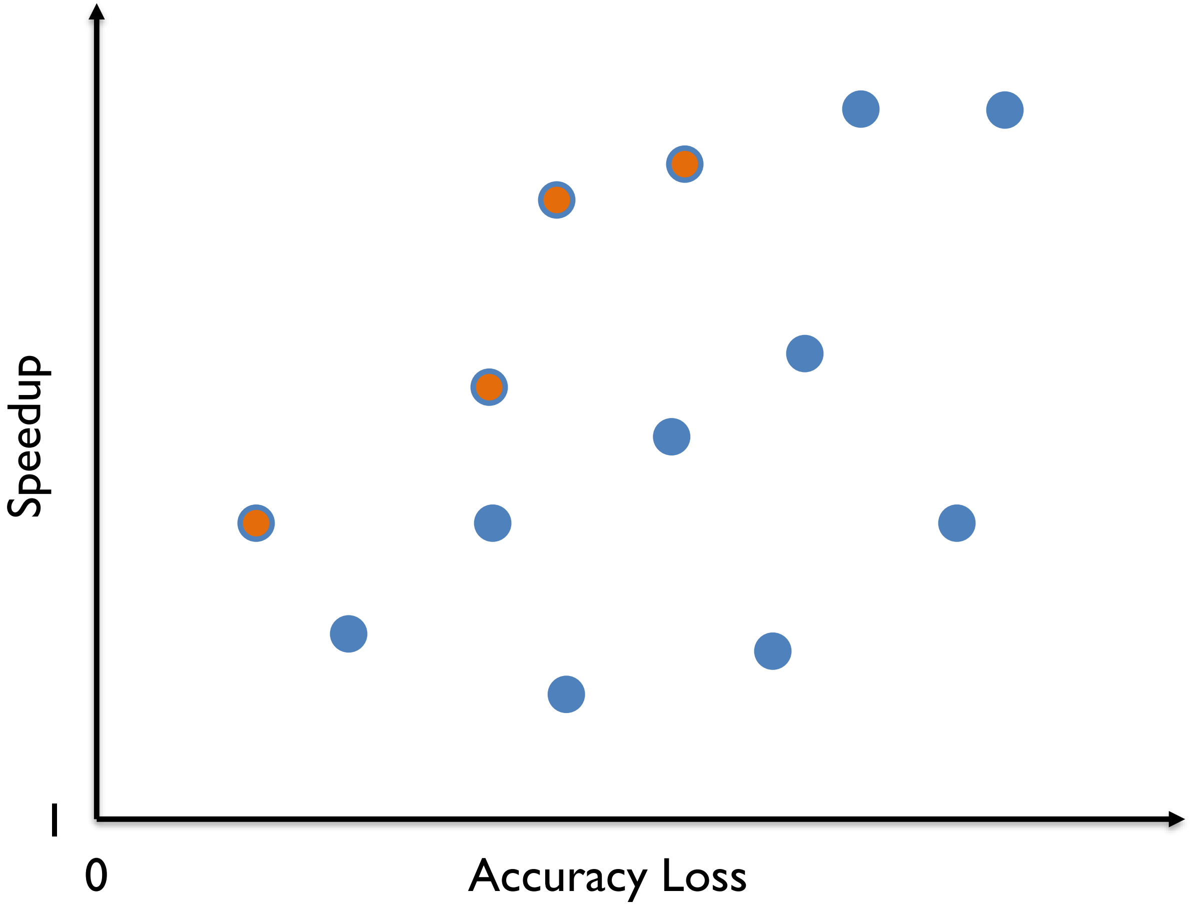
Example



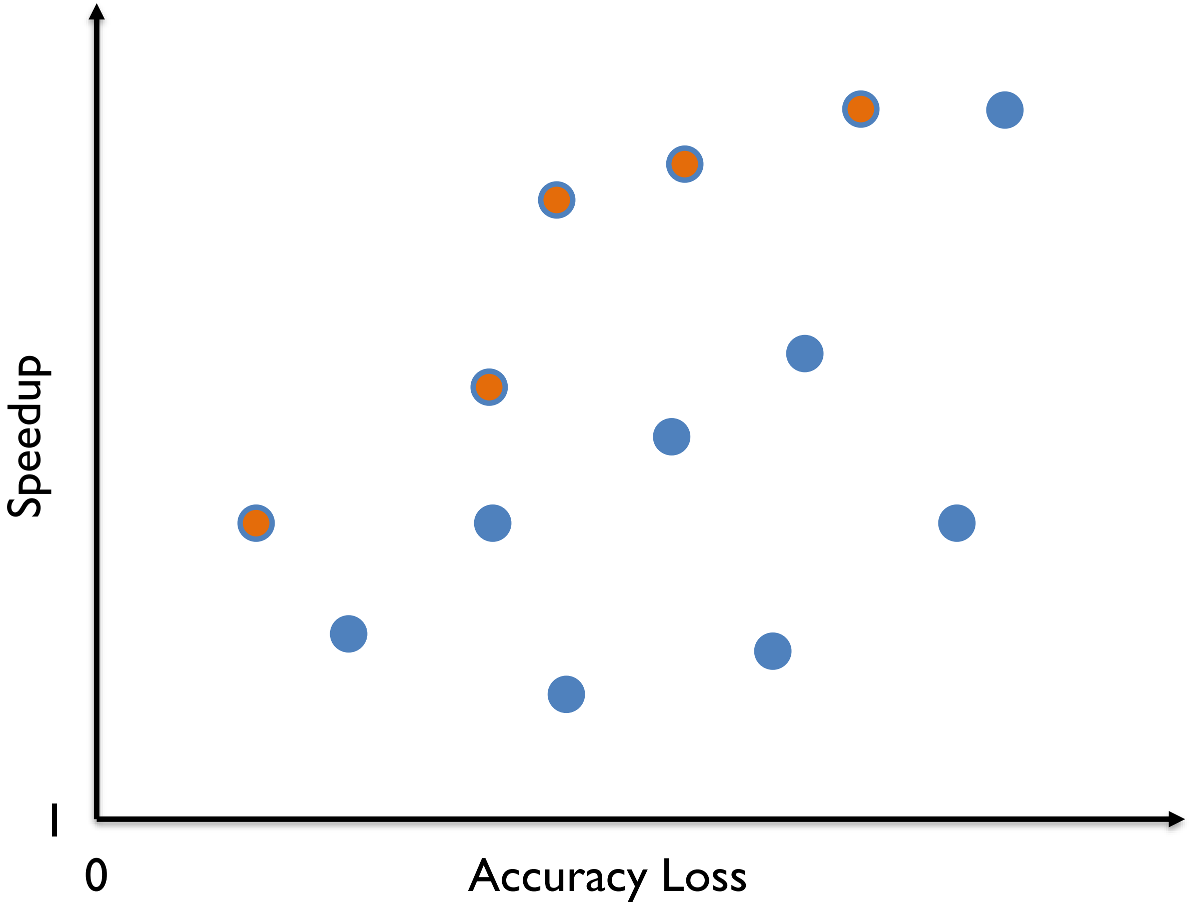
Example



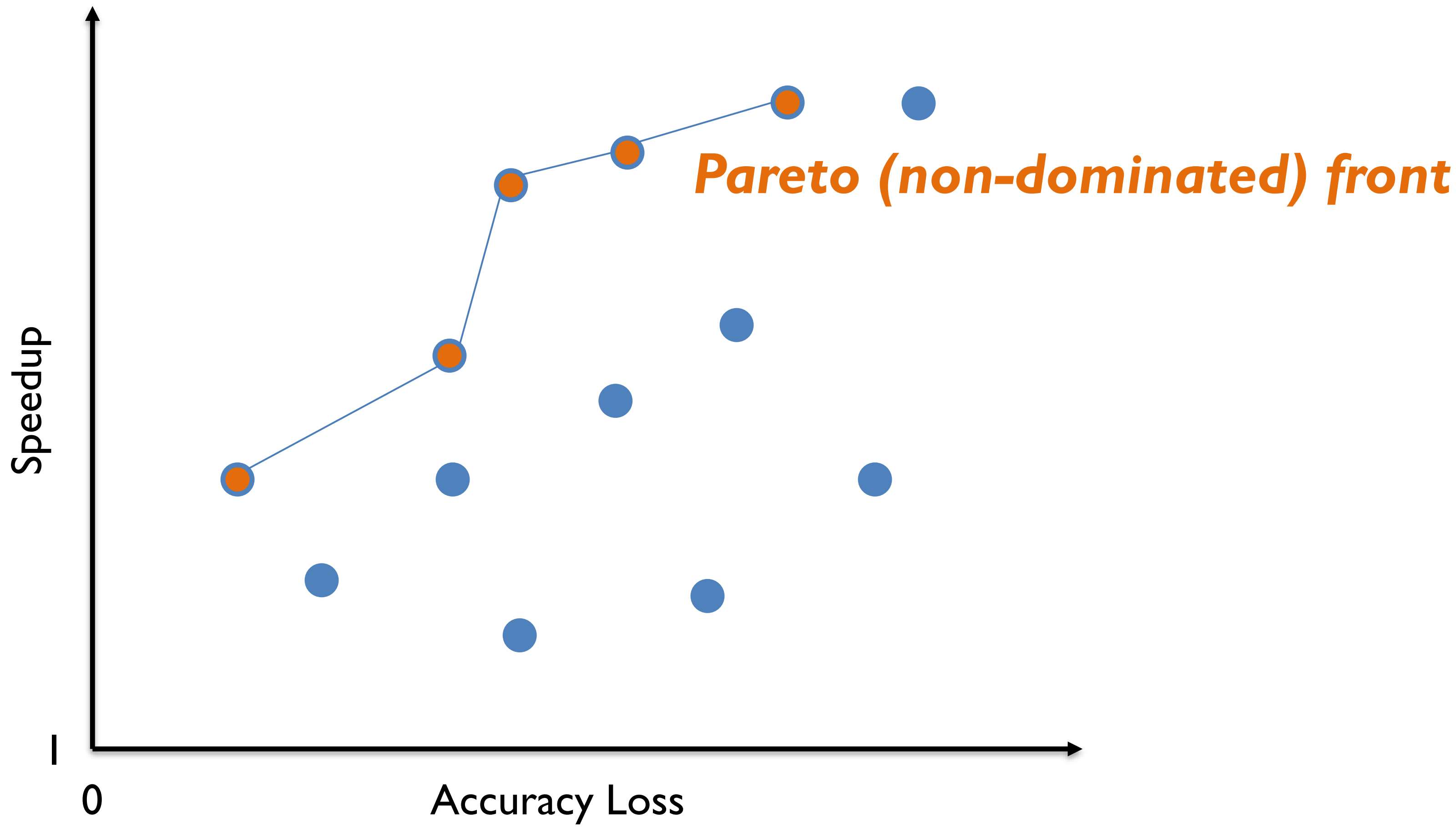
Example



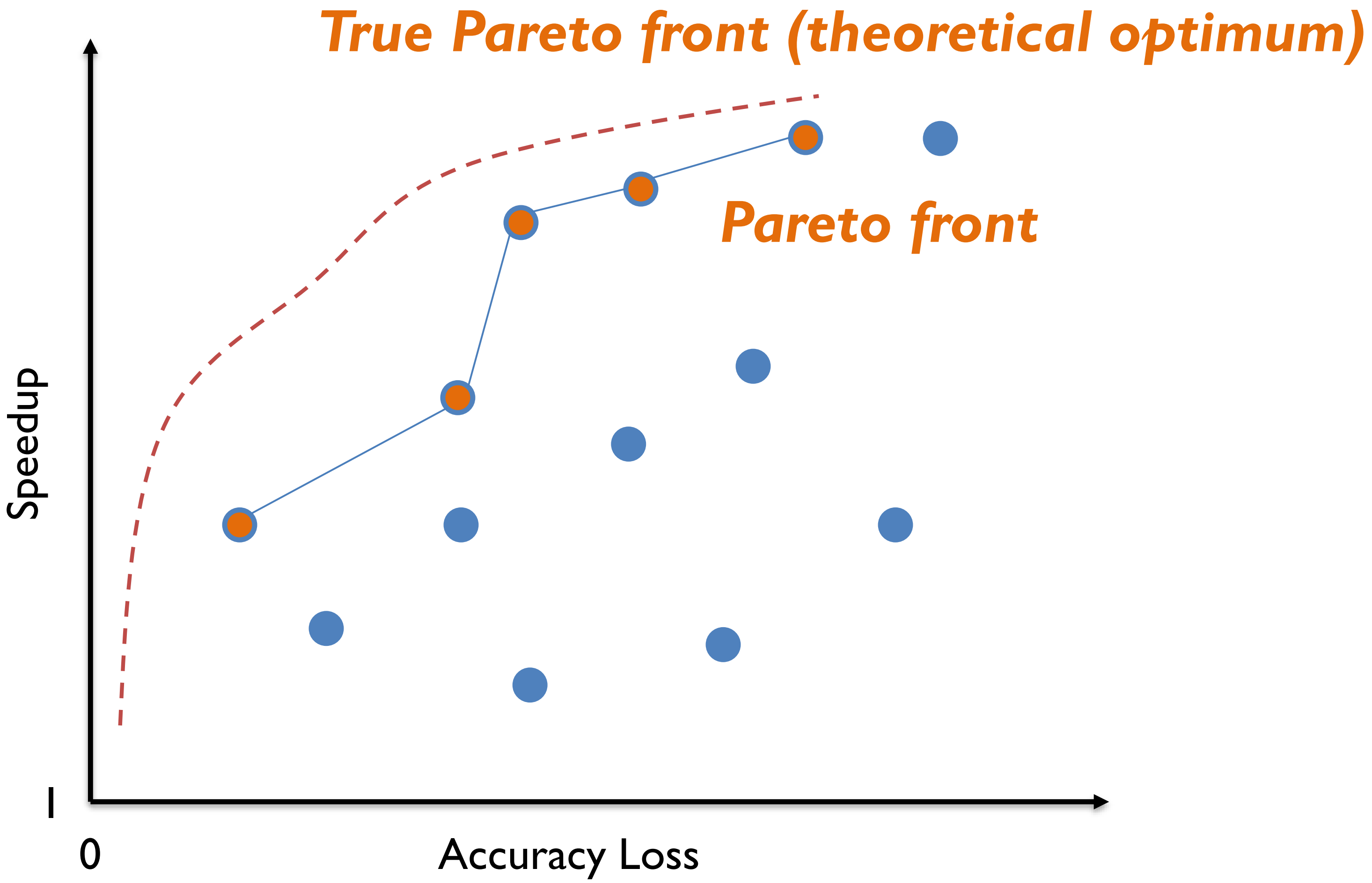
Example



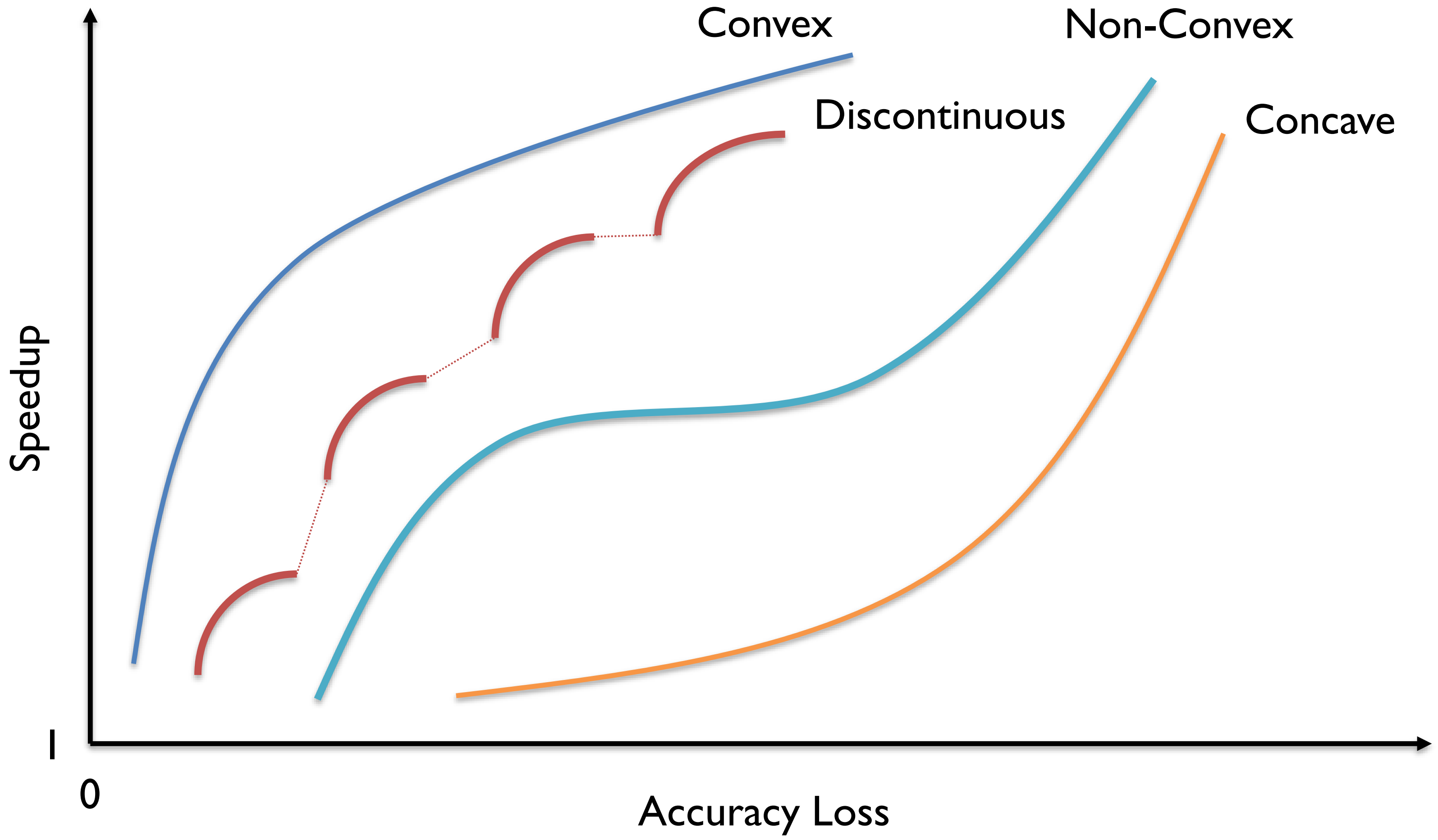
Example



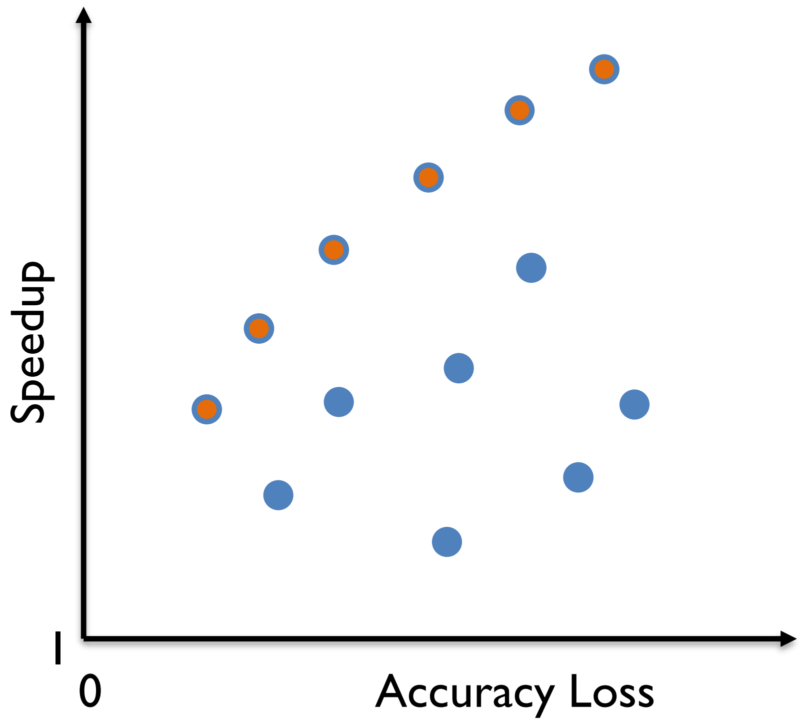
Example


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

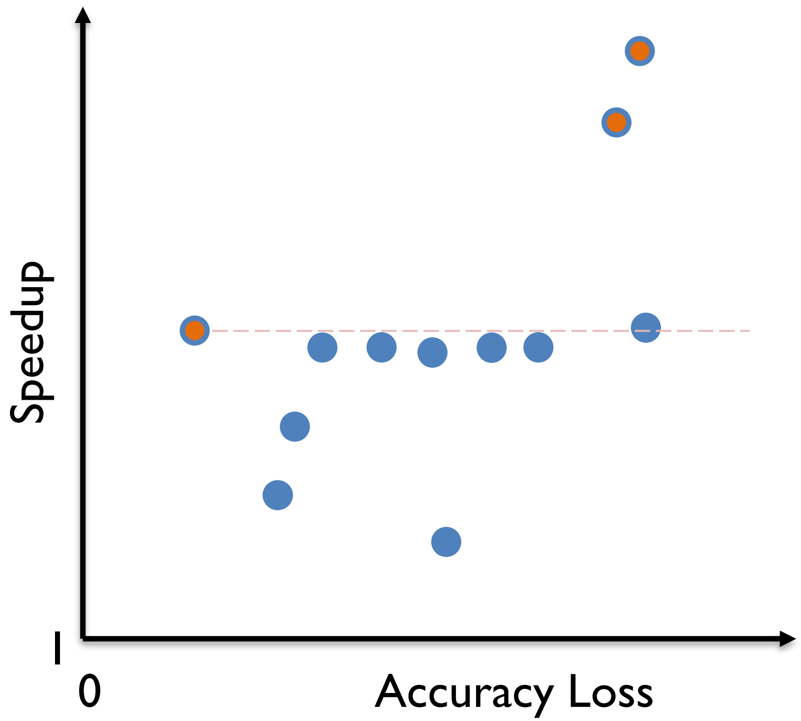
Pareto Fronts (aka Tradeoff curves)



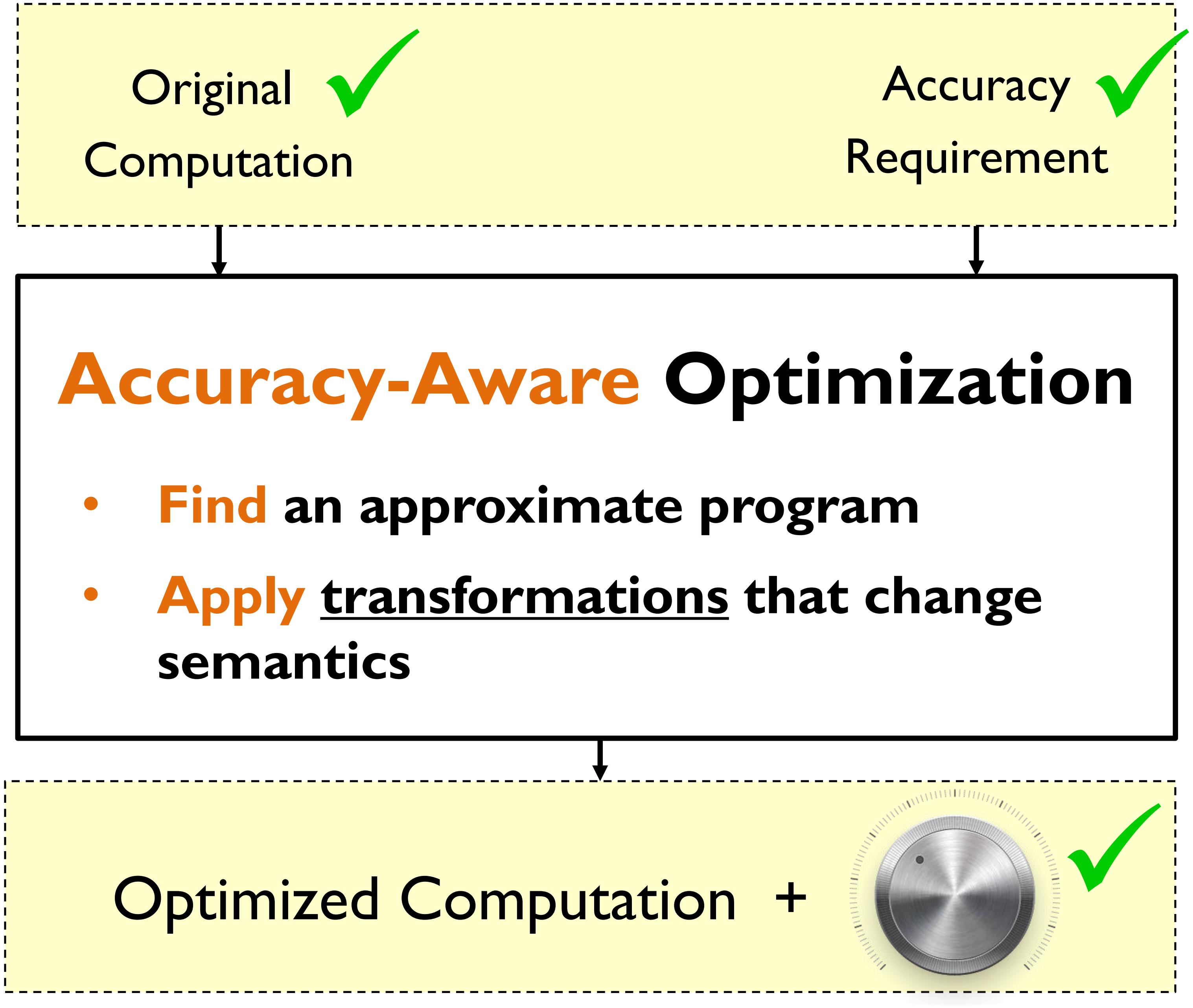
Spread of Solutions:



vs.



Often to have a useful set of points, a developer would like to have points spread across the entire space, not located only at the corners



Original Computation ✓

Accuracy Requirement ✓

Accuracy-Aware Optimization

- **Find an approximate program**
- **Apply transformations that change semantics**

Optimized Computation + ✓

Safari:

SOFTWARE TRANSFORMATIONS

Transformations

Dimensions of impact:

- **Reducing computation**
- **Reducing data**
- **Reducing communication/synchronization**

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

$$\text{Speedup} = \frac{\text{Original program time}}{\text{Approximate program time}}$$

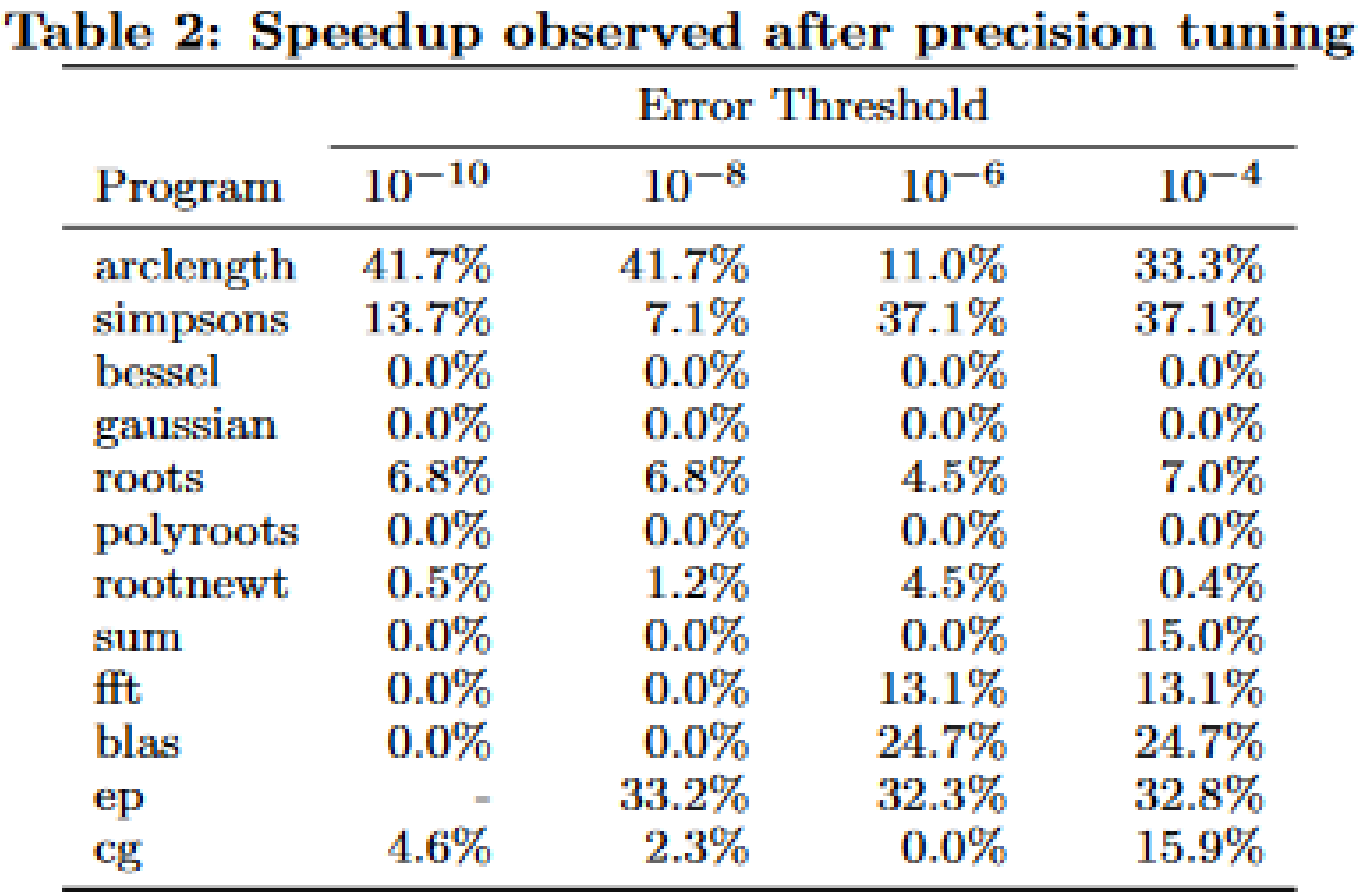


Table 2: Speedup observed after precision tuning

	Error Threshold			
Program	10^{-10}	10^{-8}	10^{-6}	10^{-4}
arclength	41.7%	41.7%	11.0%	33.3%
simpsons	13.7%	7.1%	37.1%	37.1%
bessel	0.0%	0.0%	0.0%	0.0%
gaussian	0.0%	0.0%	0.0%	0.0%
roots	6.8%	6.8%	4.5%	7.0%
polyroots	0.0%	0.0%	0.0%	0.0%
rootnewt	0.5%	1.2%	4.5%	0.4%
sum	0.0%	0.0%	0.0%	15.0%
fft	0.0%	0.0%	13.1%	13.1%
blas	0.0%	0.0%	24.7%	24.7%
ep	-	33.2%	32.3%	32.8%
cg	4.6%	2.3%	0.0%	15.9%

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

Loop Perforation

```
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++)  { ... }
```

⇩

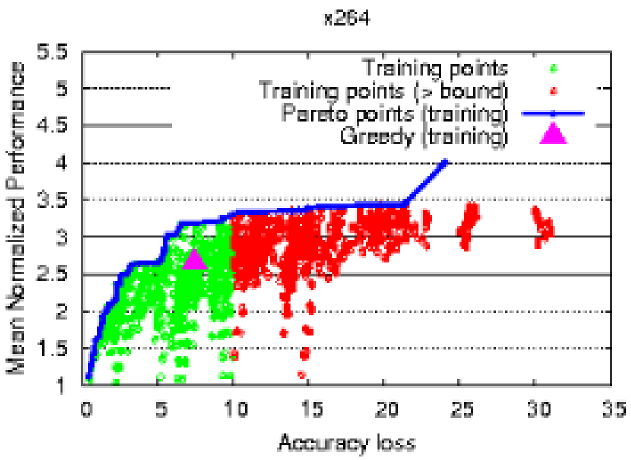
```
for (i = 0; i < n/2; i++) {... }
```

Loop Perforation

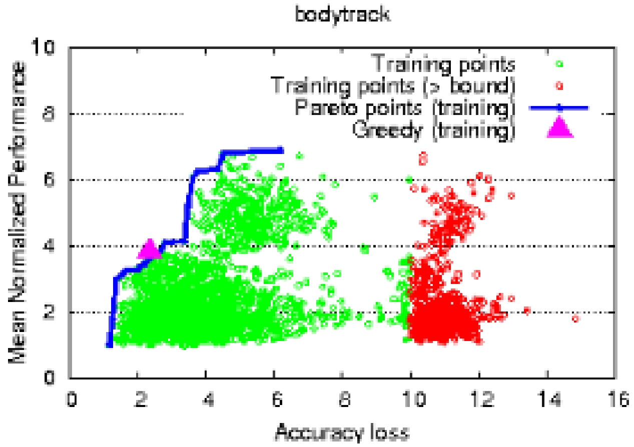
```
for (i = 0; i < n; i++)  { … }
```

⇩

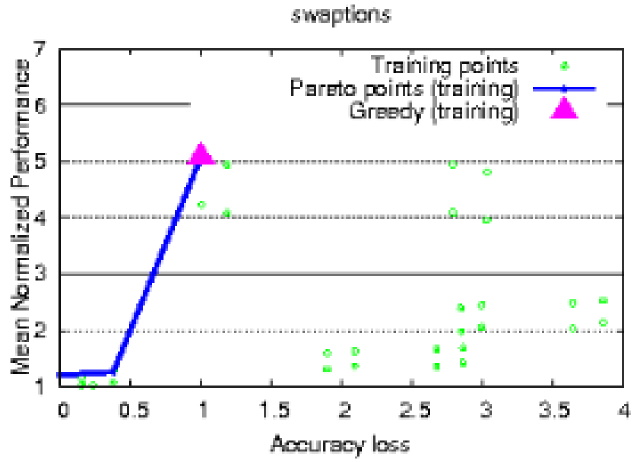
```
for (i = 0; i < n; i++) {
  if (rand(0.5)) continue;
  …
}
```



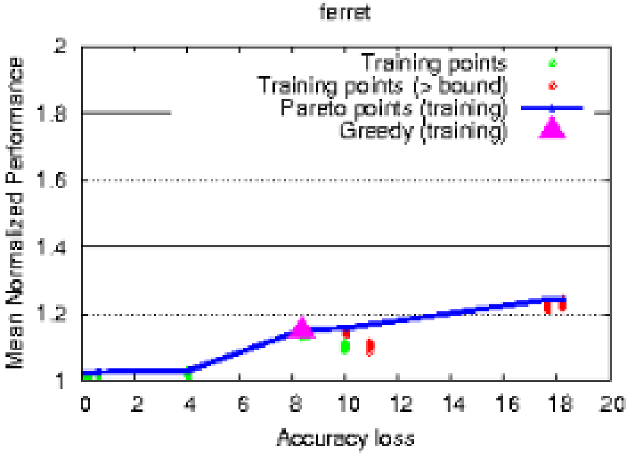
1: x264 (exhaustive)



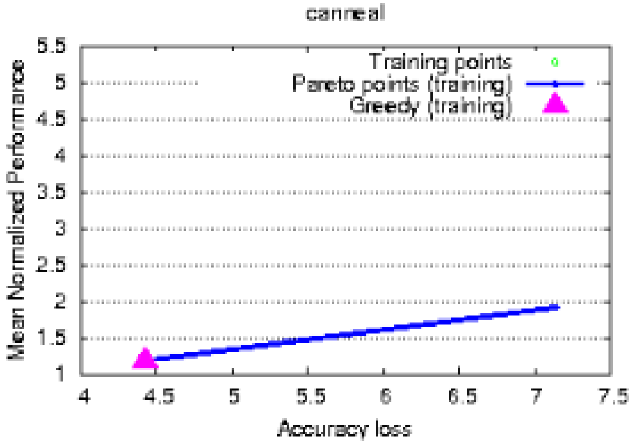
2: Bodytrack (exhaustive)



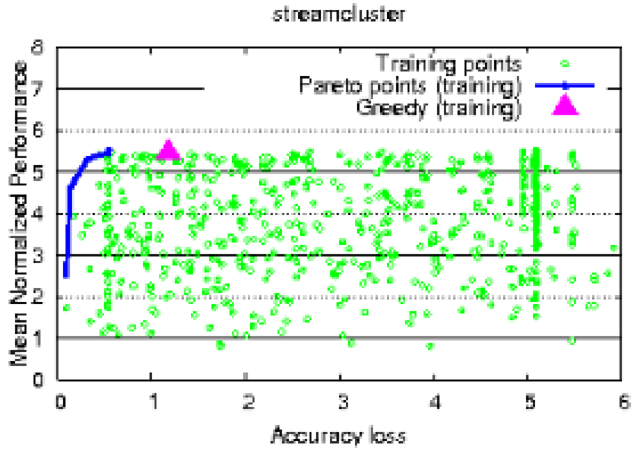
3: Swaptions (exhaustive)



4: Ferret (exhaustive)



5: Canneal (exhaustive)



6: Streamcluster (exhaustive)

Managing Performance vs. Accuracy Trade-offs With Loop Perforation FSE 2011

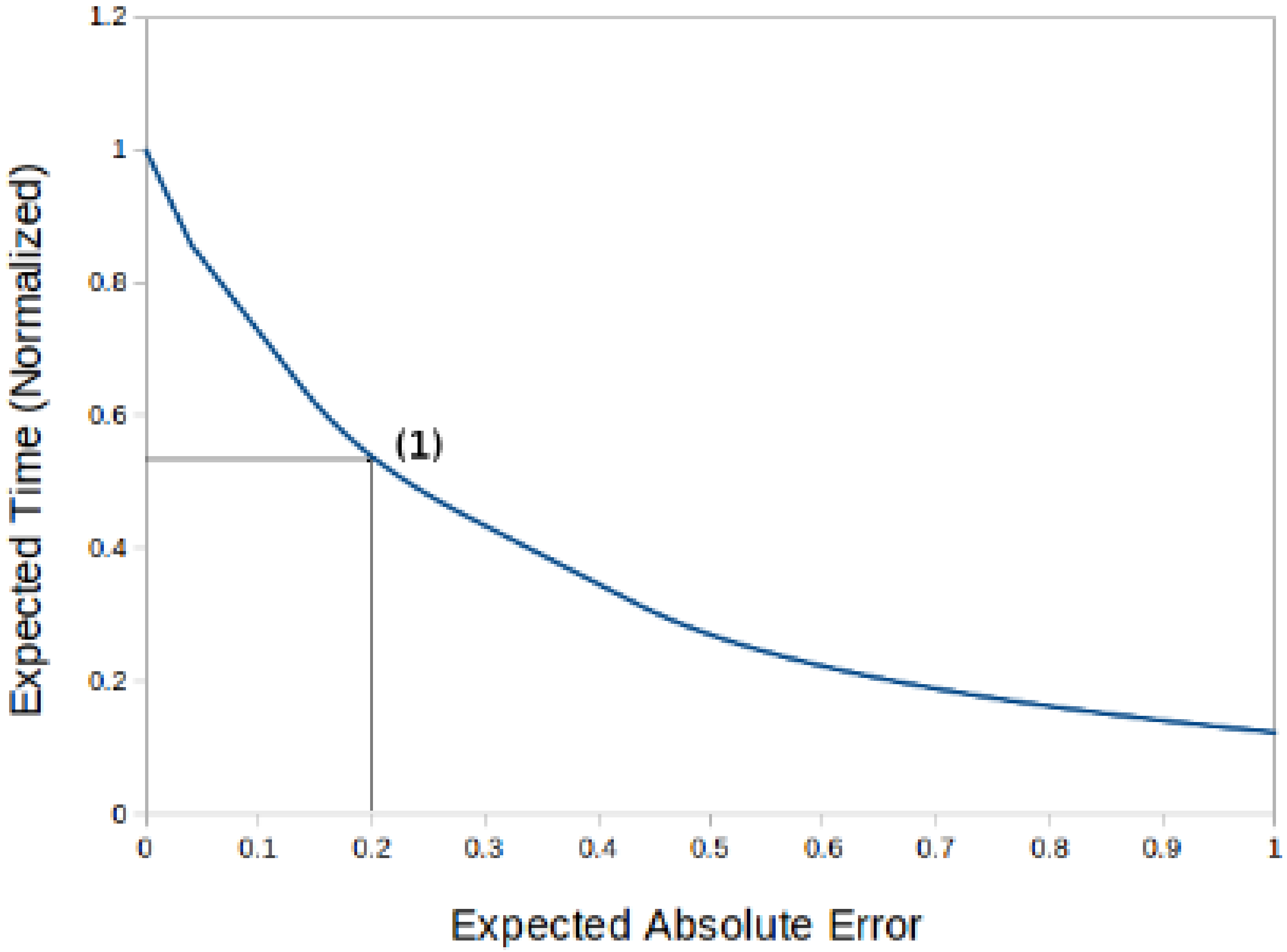
Reduction Sampling

```
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

Tradeoff curve for the main component of Bodytrack



Misailovic et al. Synthesis of Randomized Accuracy-Aware Map-Fold Programs (WACAS 2014)

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

Approximate Tiling

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

⇩

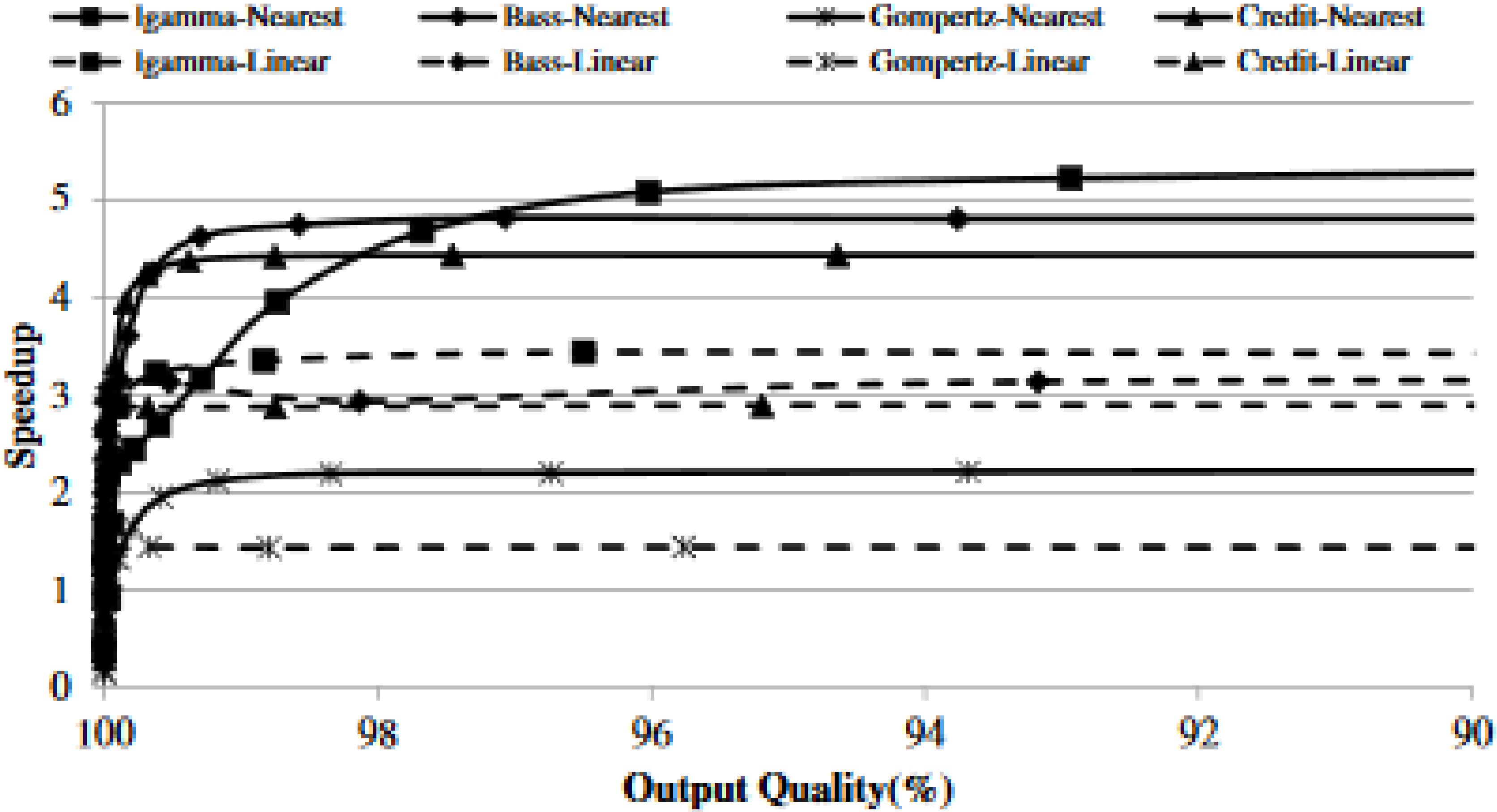
```
InType prev;
for (i = 0; i < n; i++) {
  if (i%2 == 1)
      y[i] = prev;
  else {
      y[i] = f(x[i]);
      prev = y[i];
} }
```

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

Figure 15: The impact of approximate memoization on four functions on a GPU. Two schemes are used to handle inputs that do not map to precomputed outputs: *nearest* and *linear*. *Nearest* chooses the nearest value in the lookup table to approximate the output. *Linear* uses linear approximation between the two nearest values in the table. For all four functions, *nearest* provides better speedups than *linear* at the cost of greater quality loss.



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

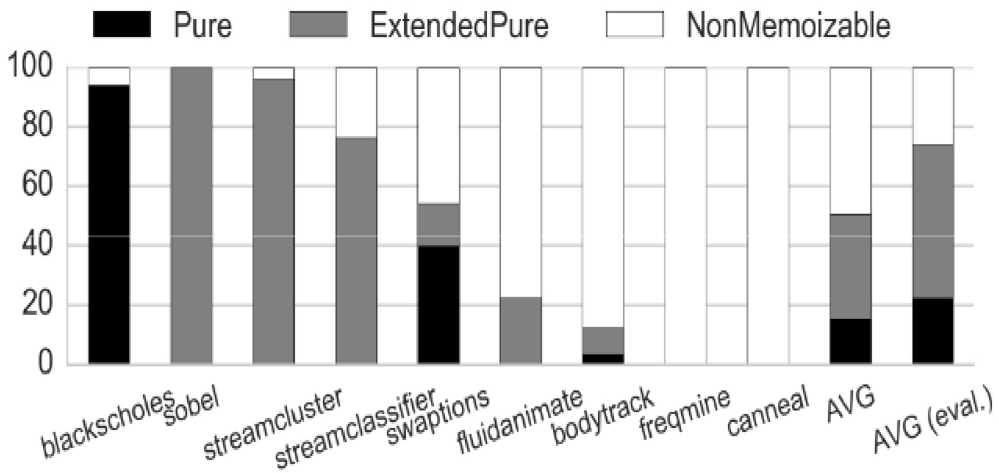


Figure 1. Execution time breakdown of all PARSEC 3.0 benchmarks that LLVM could compile. The AVG column presents the average breakdown across all benchmarks. The AVG (eval.) column presents the average breakdown across the benchmarks we consider in the remainder of this study (which exclude bodytrack, freqmine, and canneal, which have almost no pure or extended pure functions). Pure functions cover a small fraction of the total execution time, while extended pure functions achieve significantly higher coverage.

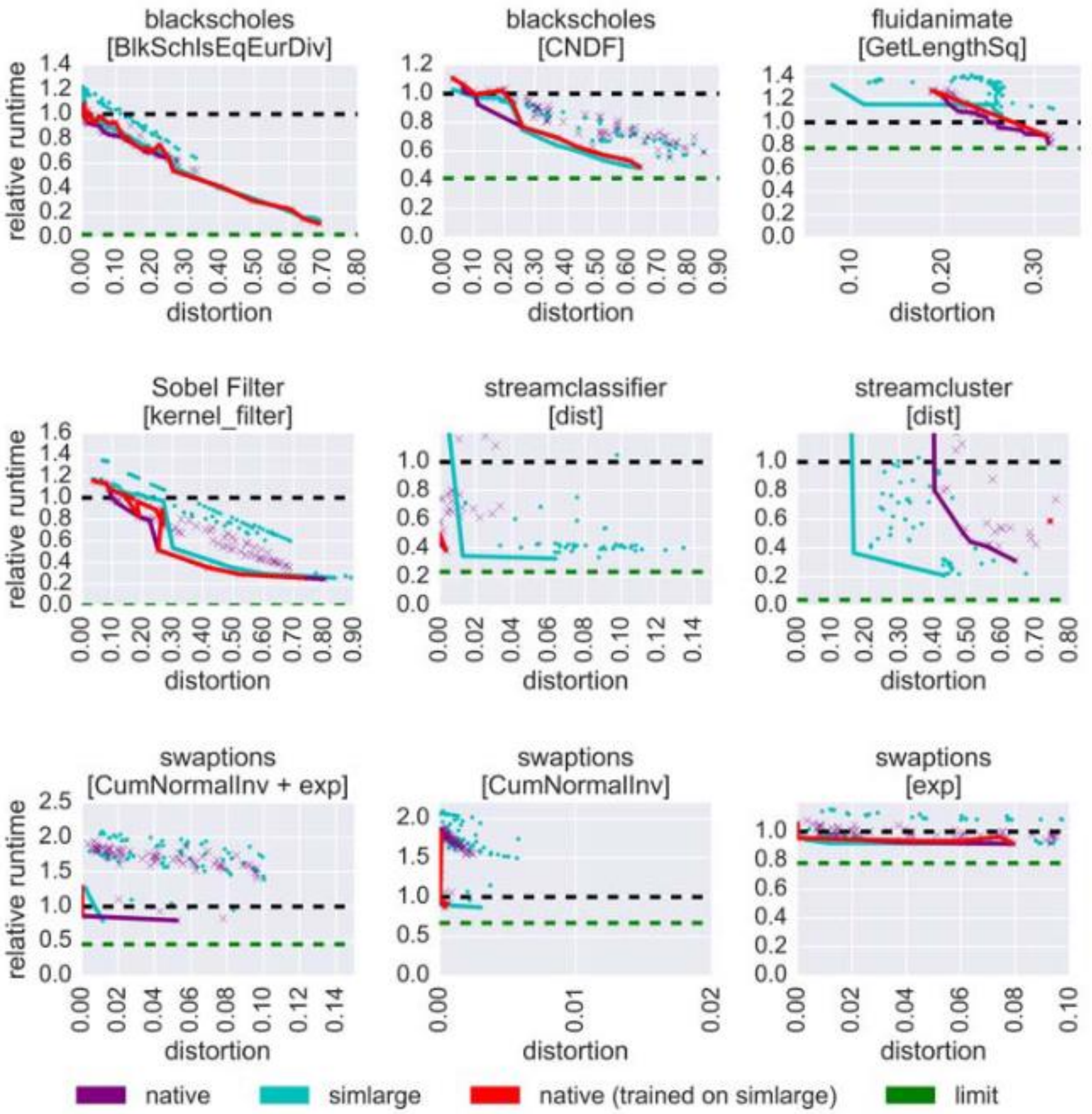
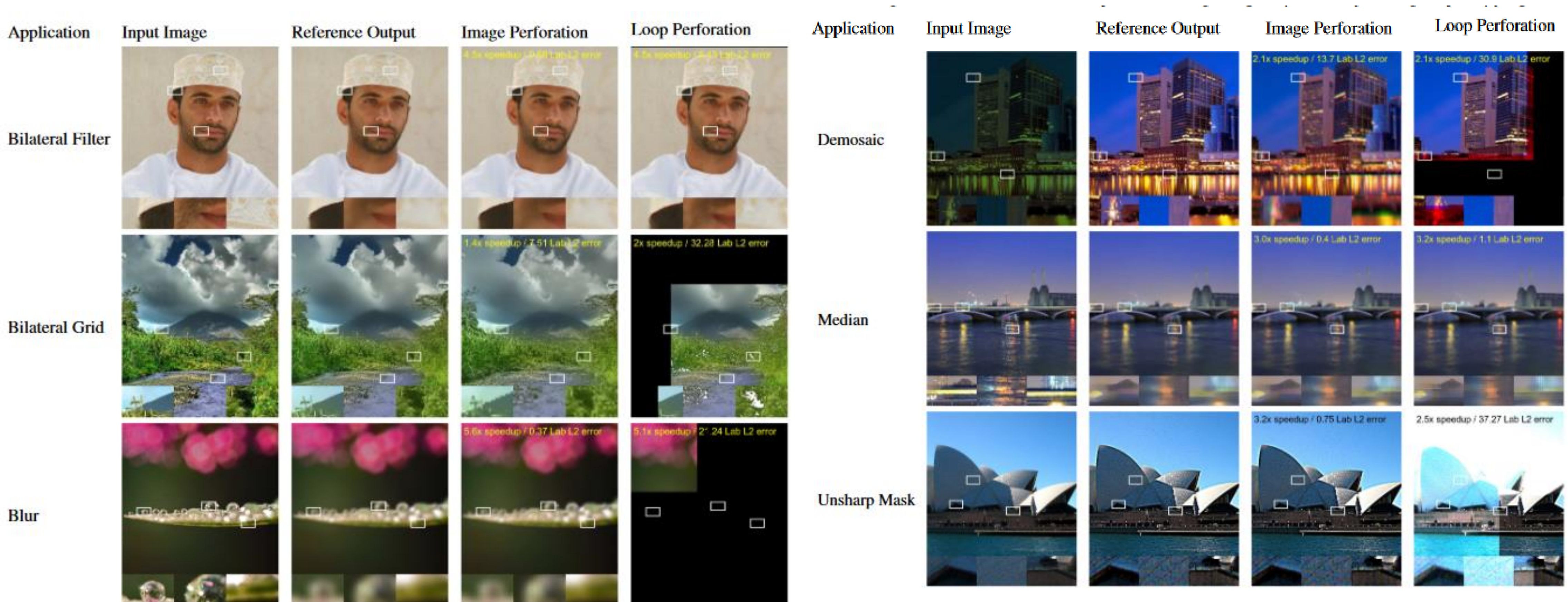


Figure 4: TAF-Memo distortion versus relative runtime. TAF-Memo achieves significant speedups with small distortion for most applications.

Tziantzioulis et al., Temporal Approximate Function Memoization (IEEE Micro Magazine 2017)

Fig. 8. Image perforation and loop perforation results for four image pipelines from top to bottom: **bilateral filter, bilateral grid, blur, demosaic, median** and **unsharp mask**. Each row compares optimized pipelines computed using each method for similar speedup factors. Please consult the supplemental document for extensive comparisons for each of these pipelines. Note that one can zoom in to see the Bayer mosaic pattern for the demosaic input. From top to bottom row, credits: © Charles Roffey; Trey Ratcliff; Neal Fowler; Eric Wehmeyer; Duncan Harris; Sandy Glass.


Application
Input Image
Reference Output
Image Perforation
Loop Perforation
Bilateral Filter
4.5x speedup / 0.68 Lab L2 error
4.5x speedup / 2.43 Lab L2 error
Bilateral Grid
1.4x speedup / 7.51 Lab L2 error
2x speedup / 32.28 Lab L2 error
Blur
5.6x speedup / 0.37 Lab L2 error
5.1x speedup / 2'.24 Lab L2 error
Application
Input Image
Reference Output
Image Perforation
Loop Perforation
Demosaic
2.1x speedup / 13.7 Lab L2 error
2.1x speedup / 30.9 Lab L2 error
Median
3.0x speedup / 0.4 Lab L2 error
3.2x speedup / 1.1 Lab L2 error
Unsharp Mask
3.2x speedup / 0.75 Lab L2 error
2.5x speedup / 37.27 Lab L2 error

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

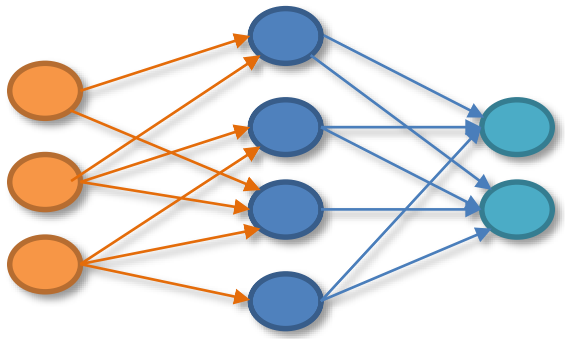
```
y = f(x);
```

⇩

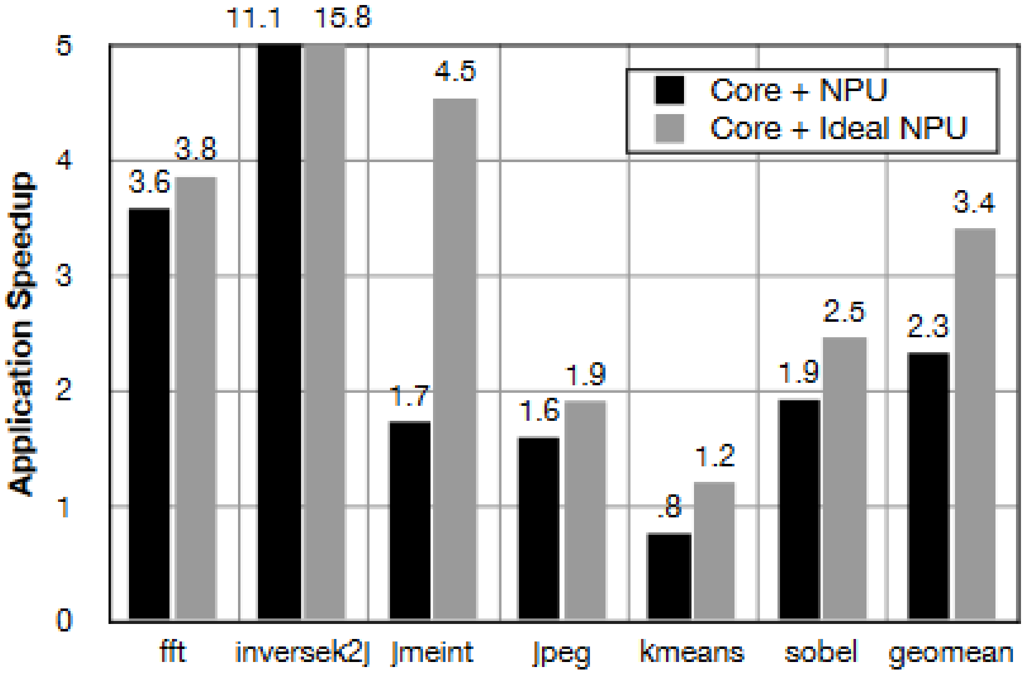
```
y = f'(x);
```

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



(a) Total application speedup with 8-PE NPU

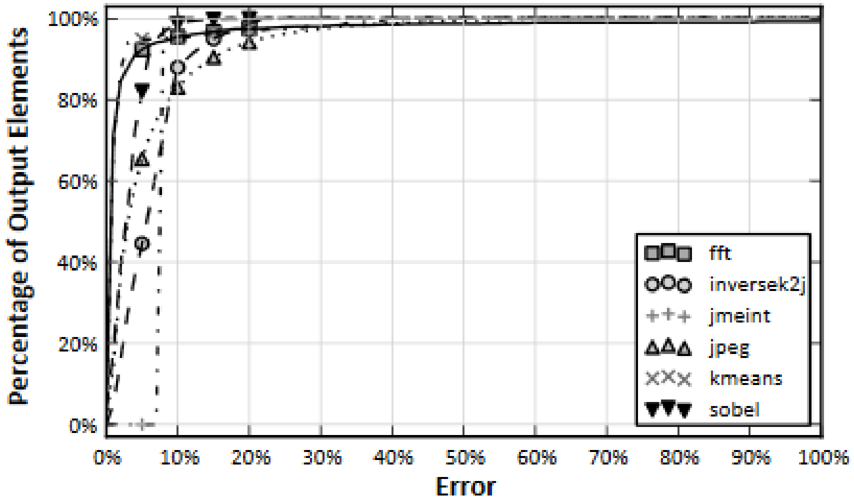


Figure 6: Cumulative distribution function (CDF) plot of the applications' output error. A point (x, y) indicates that y fraction of the output elements see error less than or equal to x.

The Parrot transformation degrades each application's average output quality by less than 10%, a rate commensurate with other approximate computing techniques.

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

Dynamic Function Substitution

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

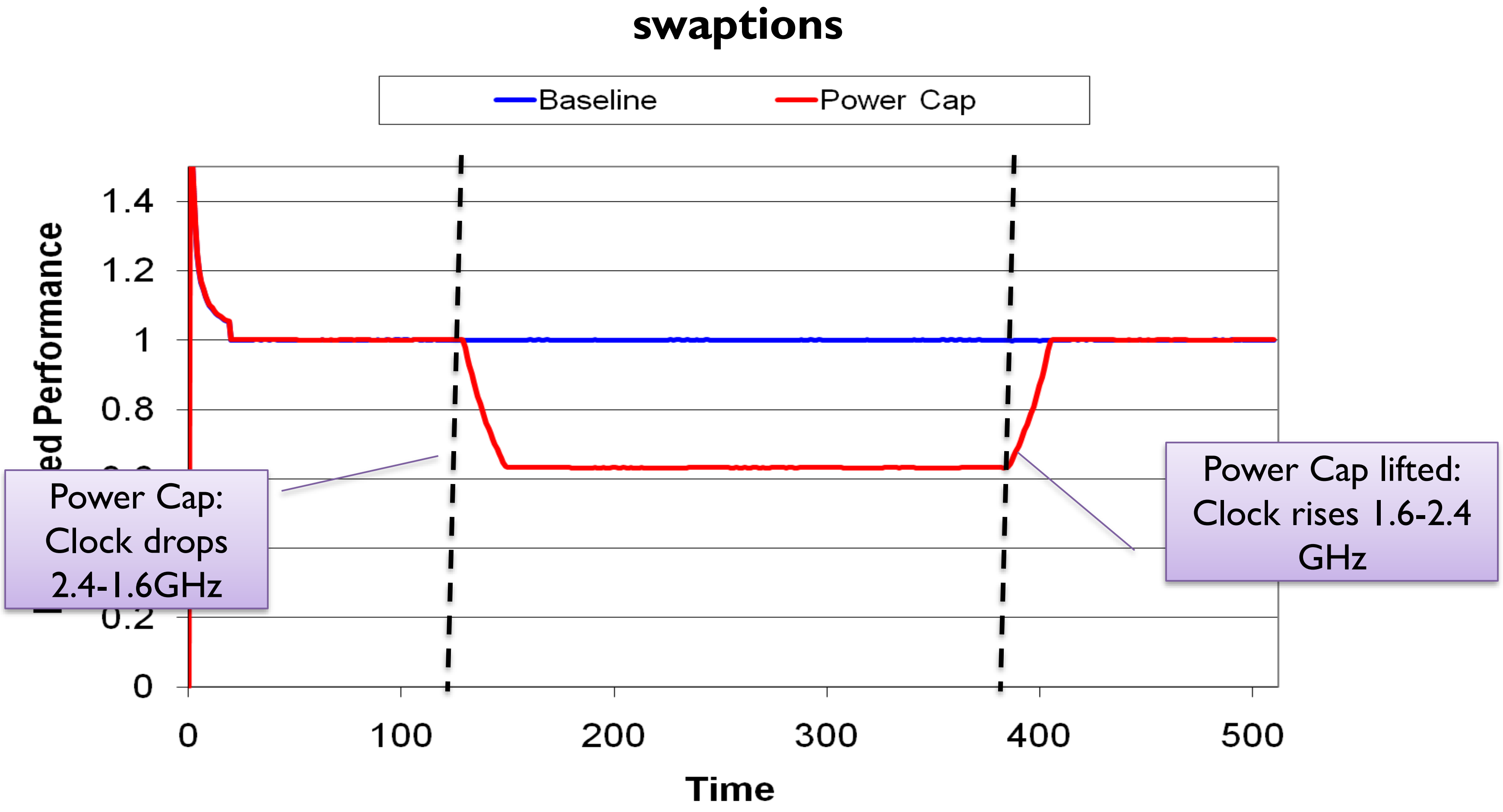
⇩

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

Dynamic Approximation

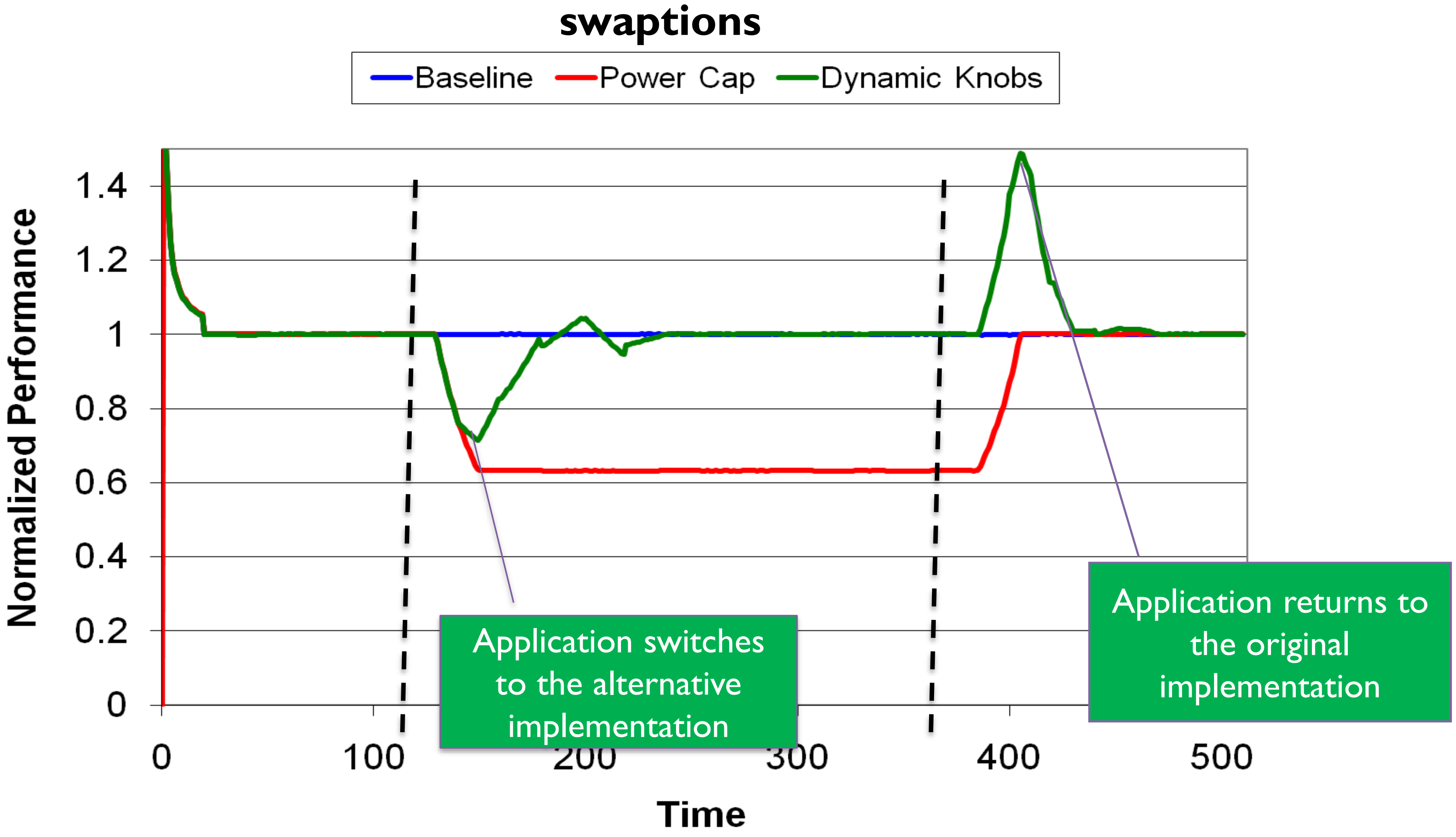


Power Cap: Clock drops 2.4-1.6GHz

Power Cap lifted: Clock rises 1.6-2.4 GHz

During the power cap, we either restart or suffer through poor performance.

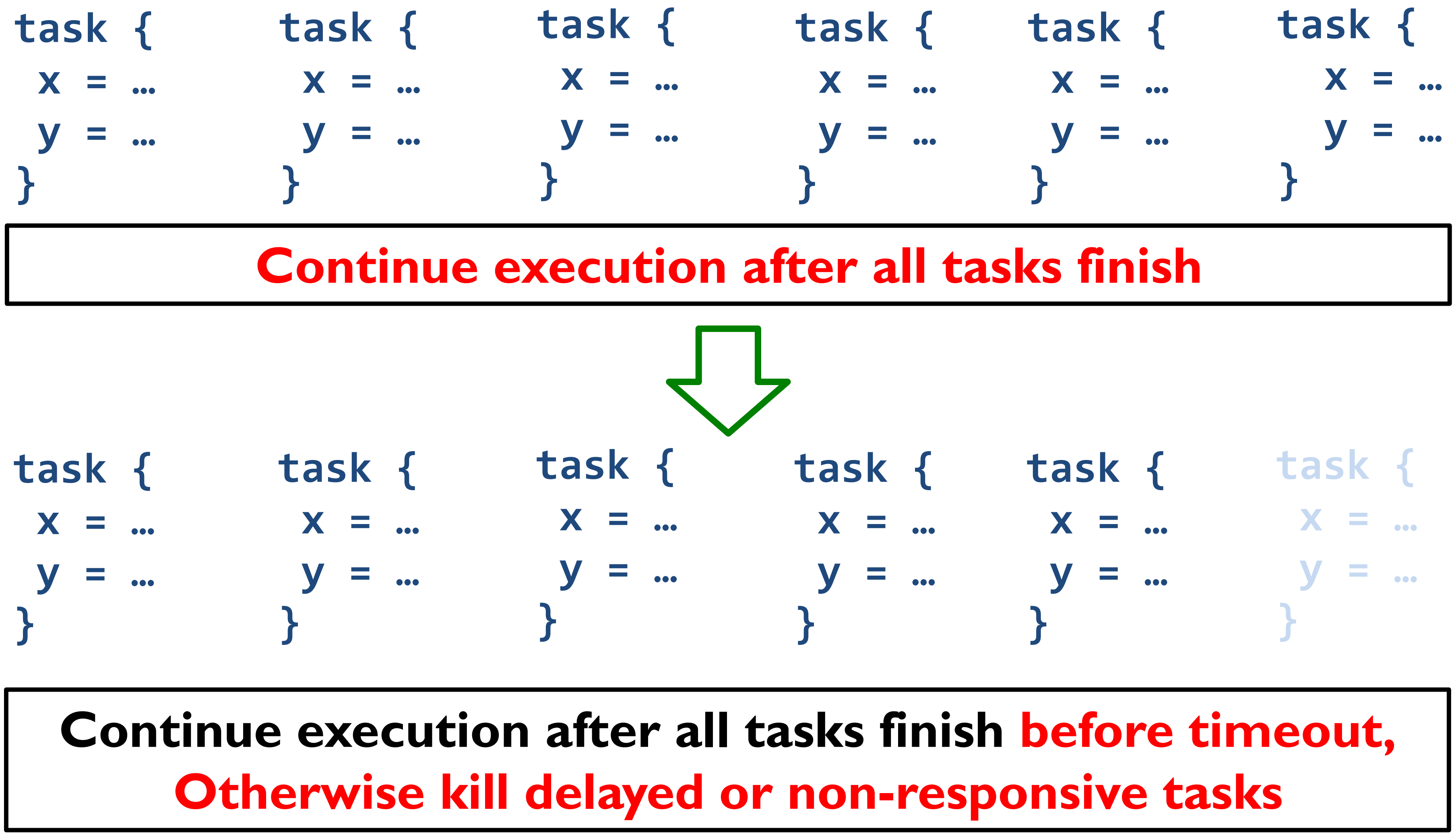
Dynamic Approximation



Application switches to the alternative implementation

Application returns to the original implementation

Skipping Tasks *(at Barrier Points)*



```
task {      task {      task {      task {      task {      task {
 x = …       x = …       x = …       x = …       x = …       x = …
 y = …       y = …       y = …       y = …       y = …       y = …
}           }           }           }           }           }
```

Continue execution after all tasks finish

```
task {      task {      task {      task {      task {      task {
 x = …       x = …       x = …       x = …       x = …       x = …
 y = …       y = …       y = …       y = …       y = …       y = …
}           }           }           }           }           }
```

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

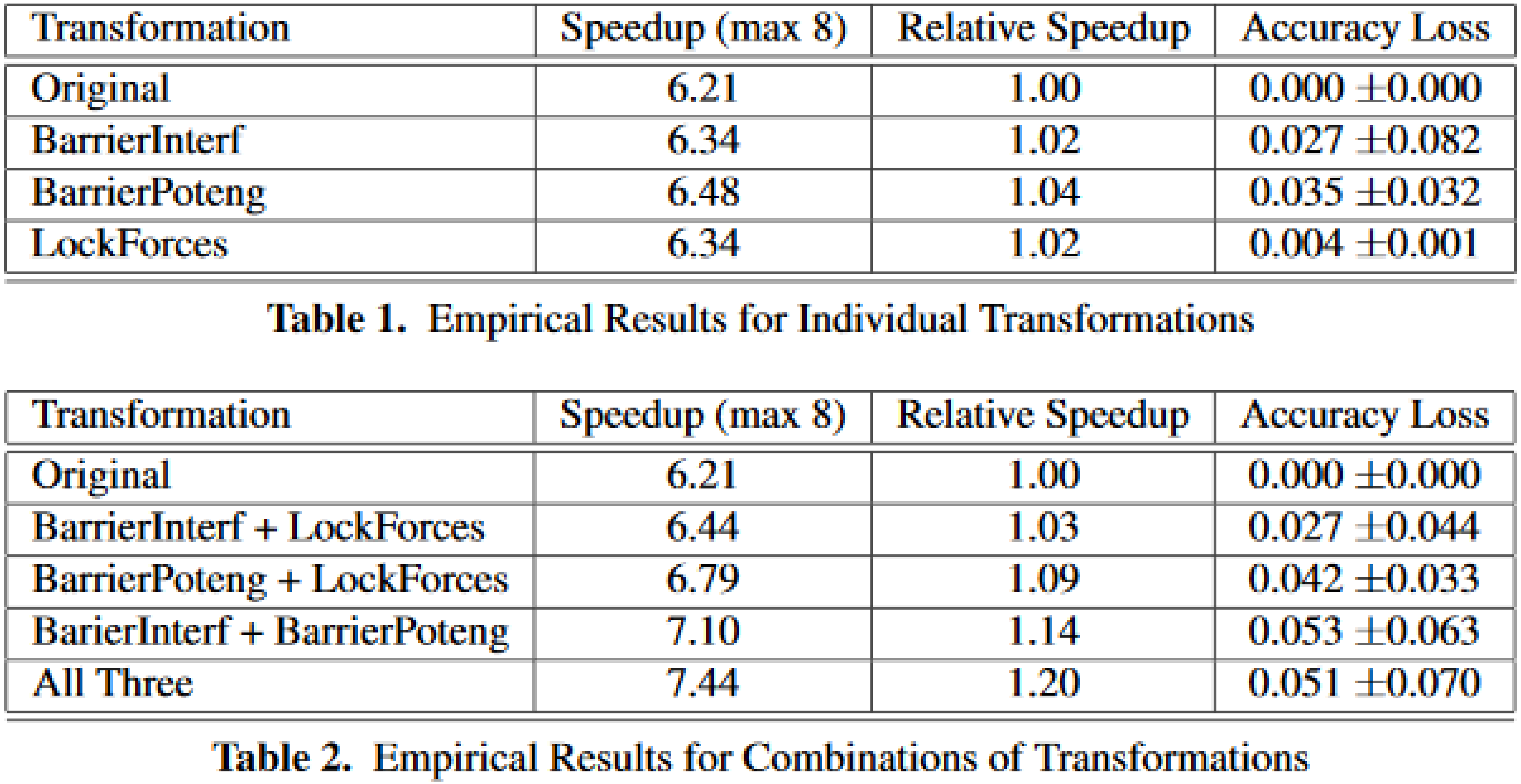
Rinard, Probabilistic accuracy bounds for fault-tolerant computations that discard tasks, ICS ’06
Meng et al. Best-Effort Parallel Execution for Recognition and Mining Applications, IPDPS’09

Removing Synchronization

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

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

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



Transformation	Speedup (max 8)	Relative Speedup	Accuracy Loss
Original	6.21	1.00	0.000 ±0.000
BarrierInterf	6.34	1.02	0.027 ±0.082
BarrierPoteng	6.48	1.04	0.035 ±0.032
LockForces	6.34	1.02	0.004 ±0.001

Table 1. Empirical Results for Individual Transformations

Transformation	Speedup (max 8)	Relative Speedup	Accuracy Loss
Original	6.21	1.00	0.000 ±0.000
BarrierInterf + LockForces	6.44	1.03	0.027 ±0.044
BarrierPoteng + LockForces	6.79	1.09	0.042 ±0.033
BarierInterf + BarrierPoteng	7.10	1.14	0.053 ±0.063
All Three	7.44	1.20	0.051 ±0.070

Table 2. Empirical Results for Combinations of Transformations

Misailovic et al. Dancing with Uncertainty, RACES '12

Transformations

Dimensions of impact:

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

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)

Applying Transformations

Selecting where in the 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 time to do the approximation:

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

We will discuss the algorithms and systems that help with approximating programs in detail!