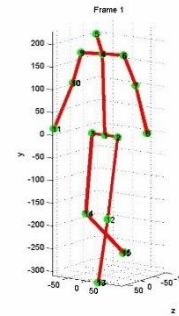
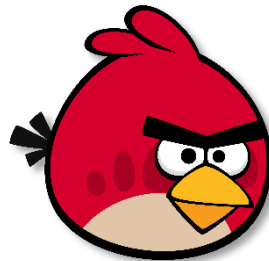
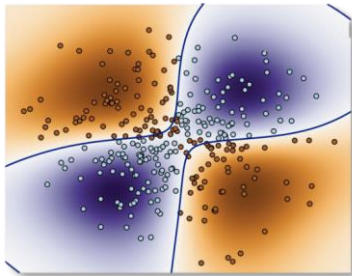
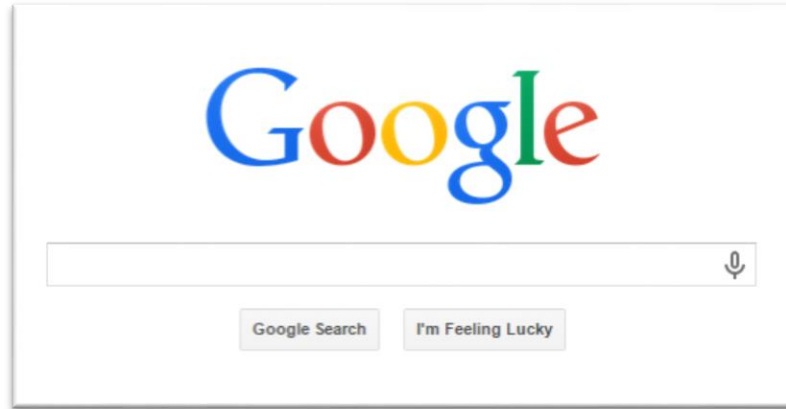
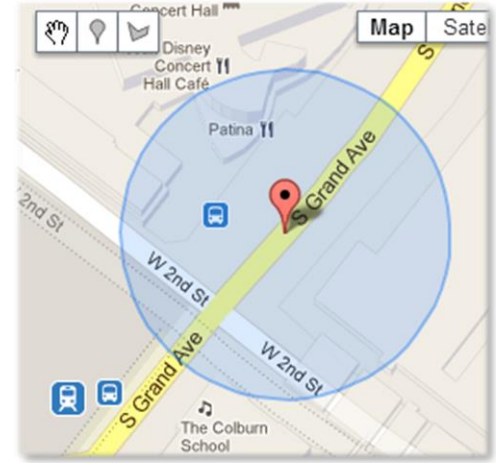
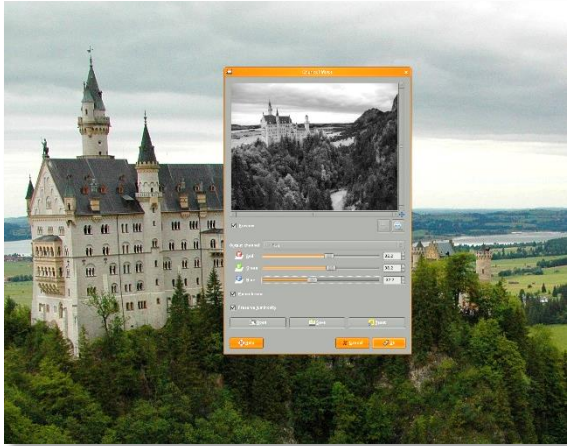


CS 598sm

Probabilistic &
Approximate
Computing

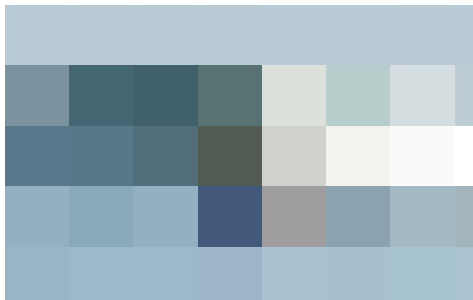
<http://misailo.web.engr.illinois.edu/courses/cs598>



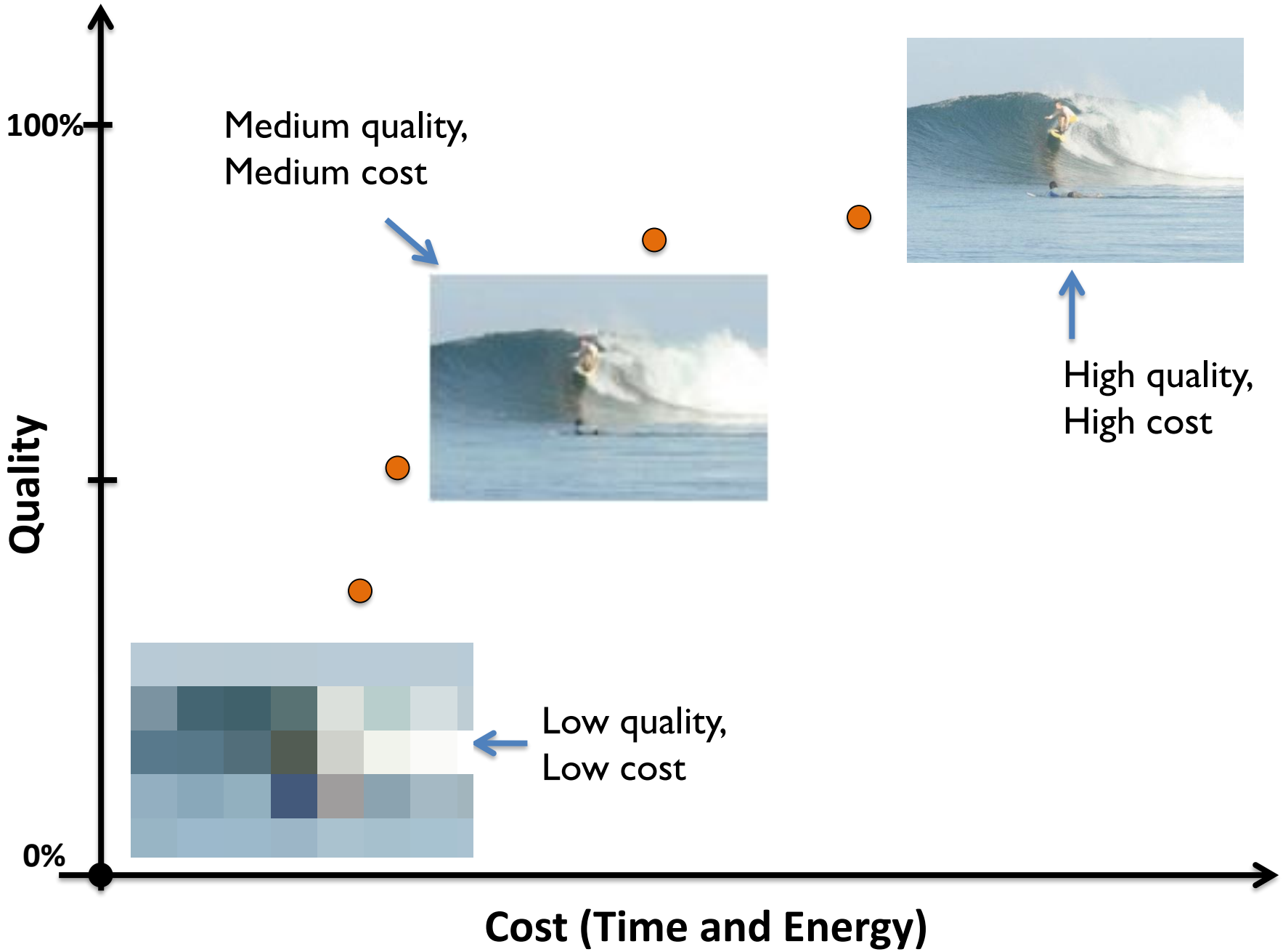
Medium quality,
Medium cost



High quality,
High cost



Low quality,
Low cost



100%

Medium quality,
Medium cost



High quality,
High cost

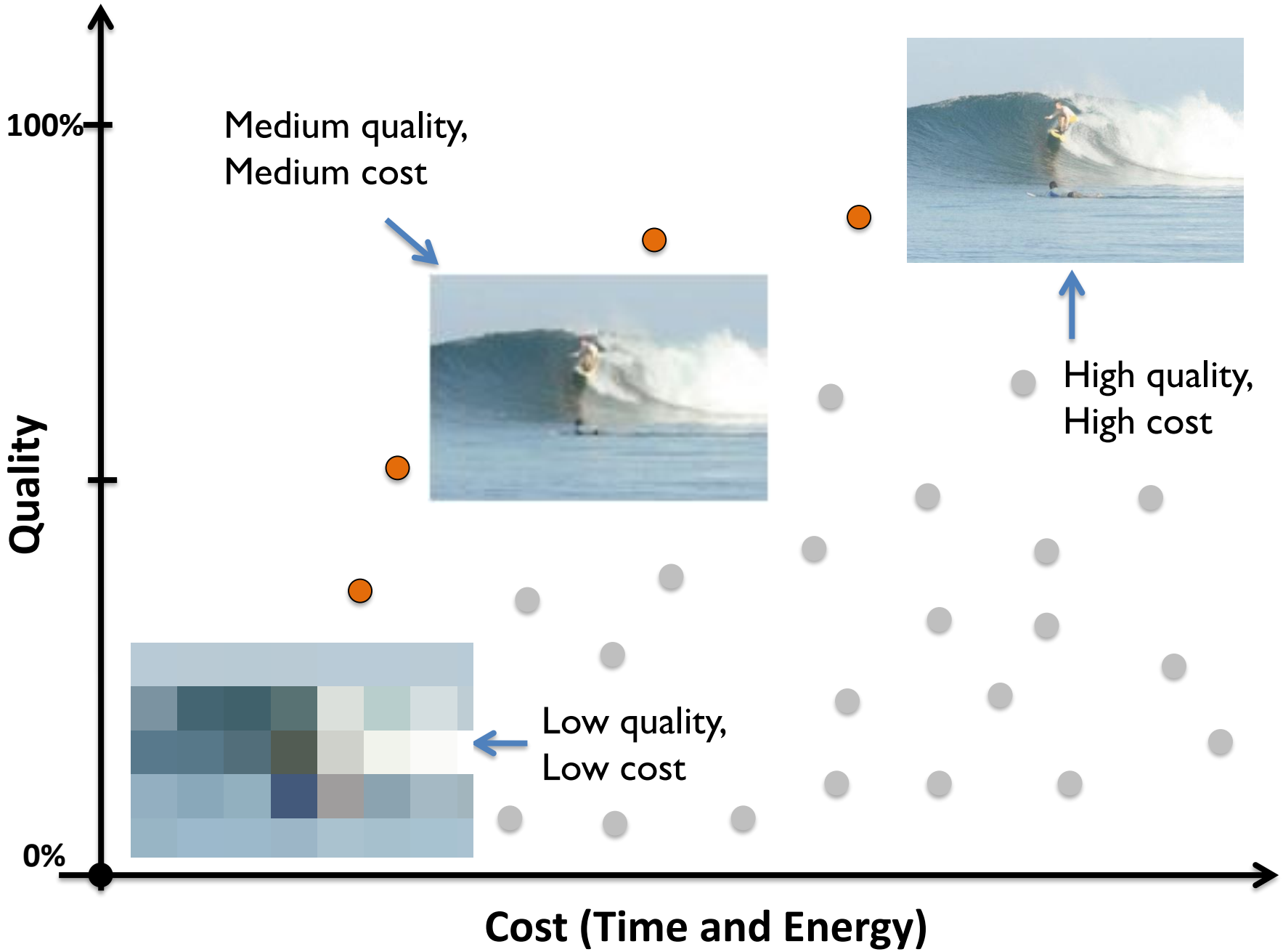


Quality

Low quality,
Low cost

0%

Cost (Time and Energy)



100%

Medium quality,
Medium cost



High quality,
High cost

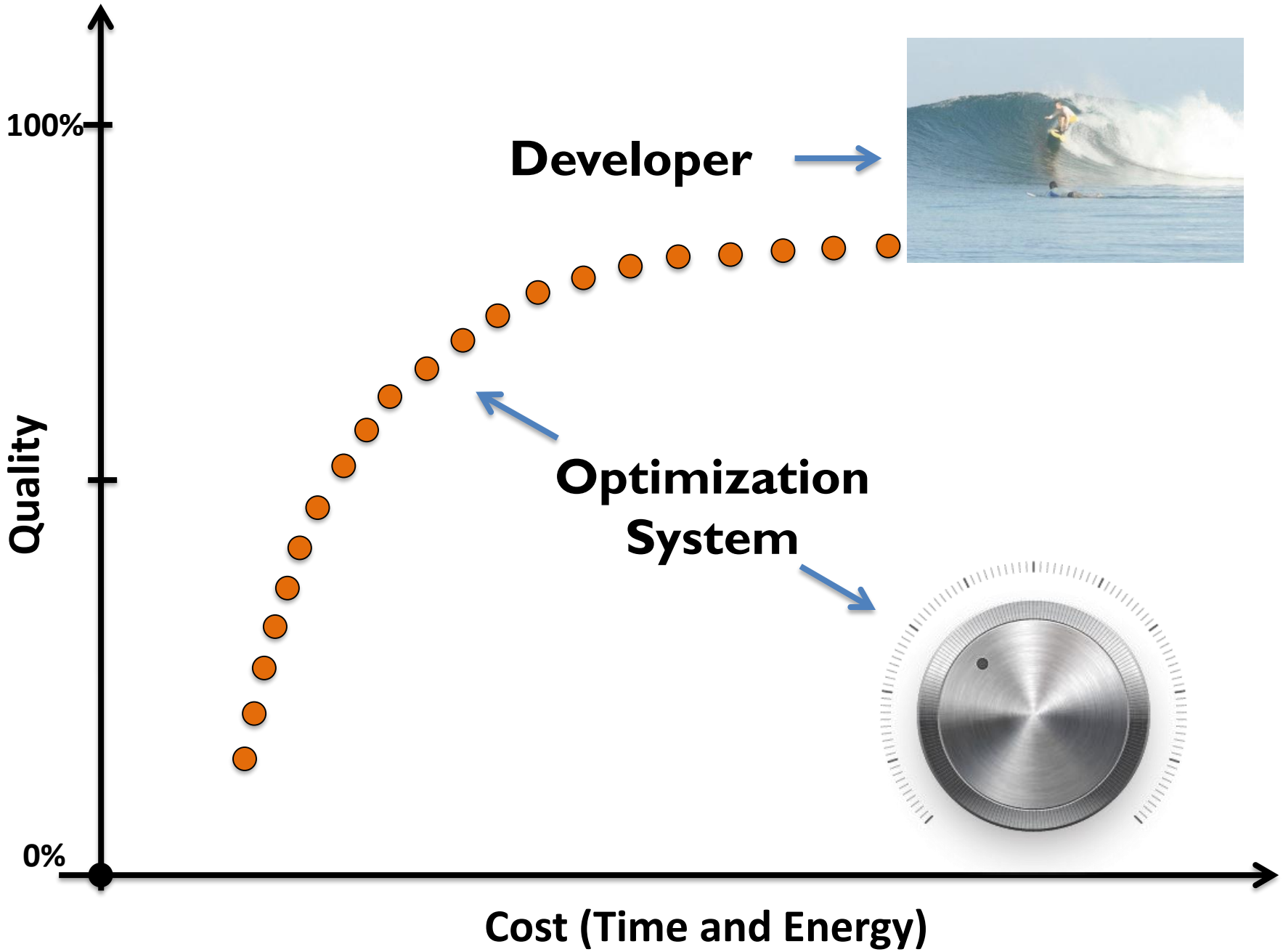
Quality



Low quality,
Low cost

0%

Cost (Time and Energy)



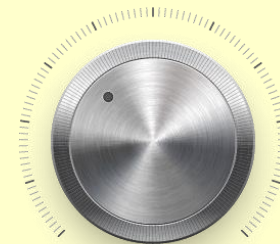
Original
Computation

Accuracy
Requirement

Accuracy-Aware Optimization

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

Optimized Computation +

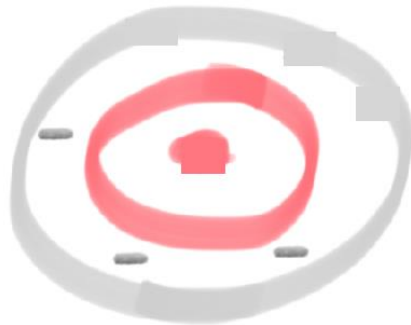
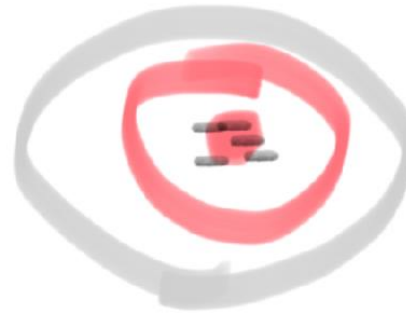
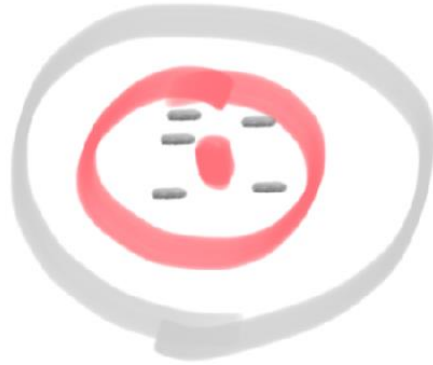


Safari:

**ACCURACY
MATTERS**

Precision

Repeatability or fineness of control

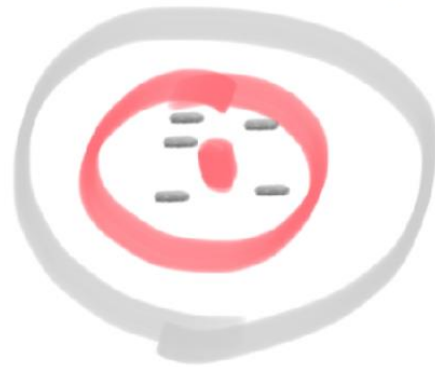


Less precise

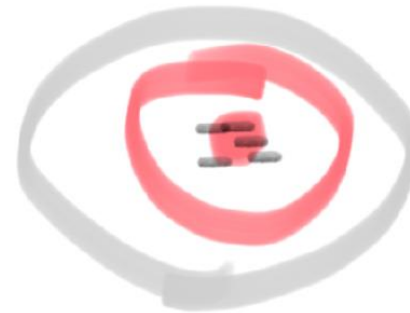
More precise

Accuracy

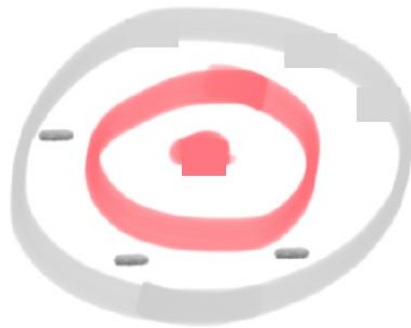
Difference from the correct value



Less precise



More accurate



Less accurate

More precise

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



Original



Perforated



Original



Perforated

**Any pixel
difference**



Original



Perforated

**> 1% pixel
difference**



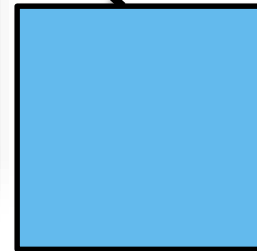
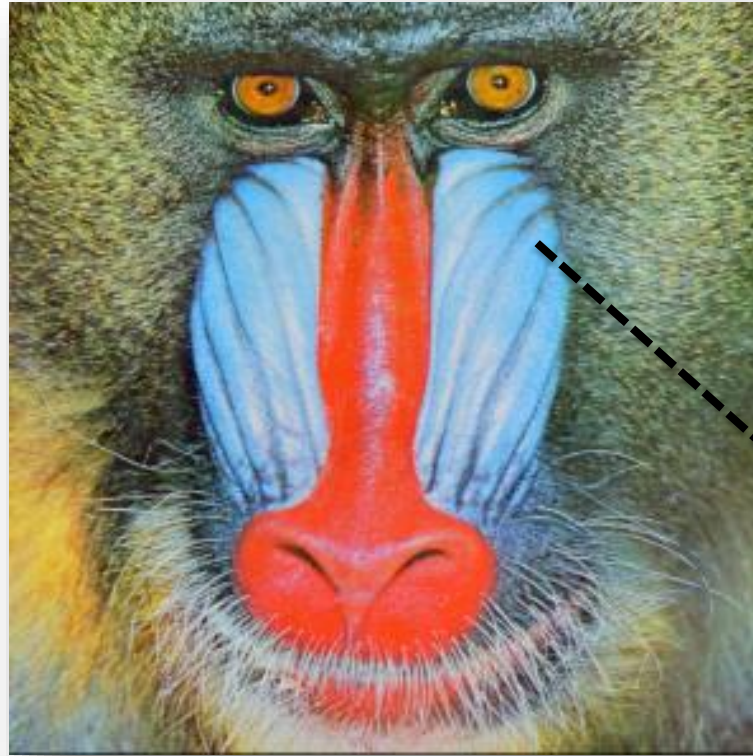
Original



Perforated

**> 5% pixel
difference**

Another Thought Experiment



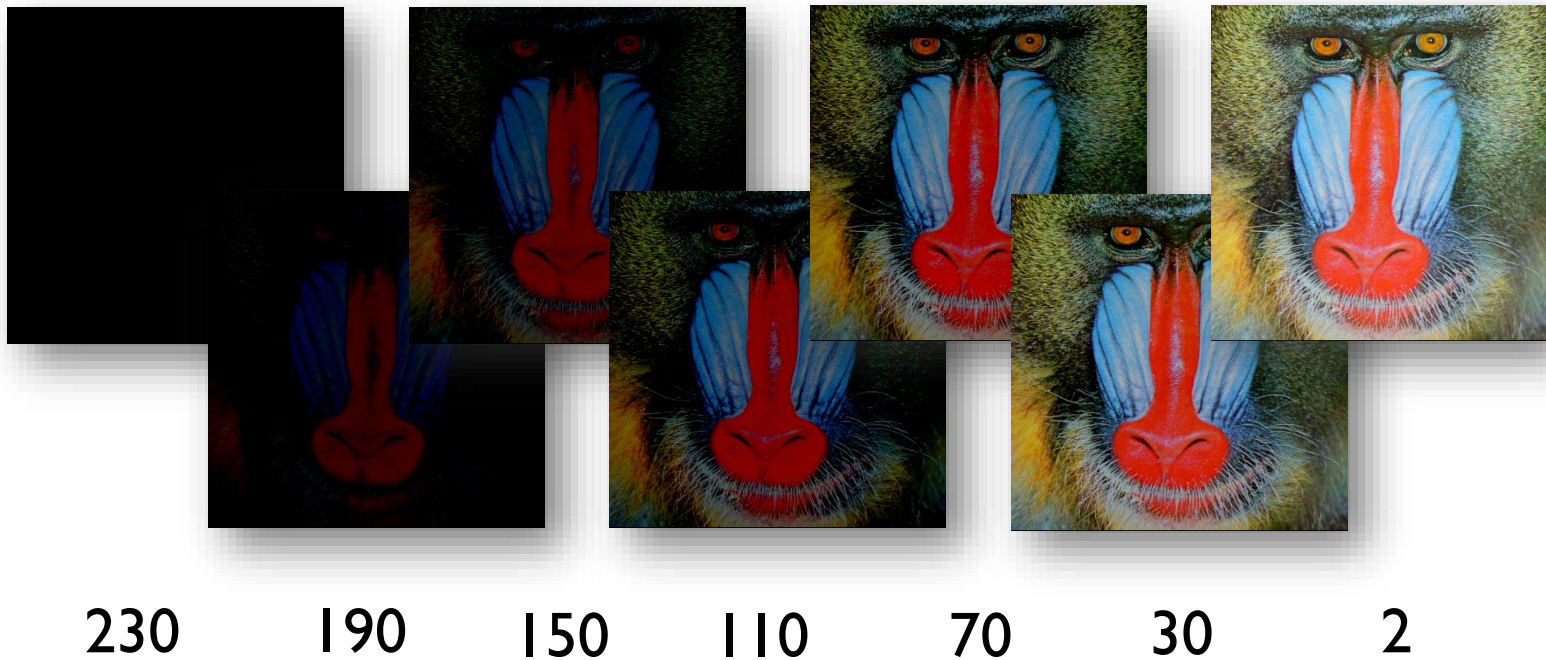
R 99
G 186
B 237

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

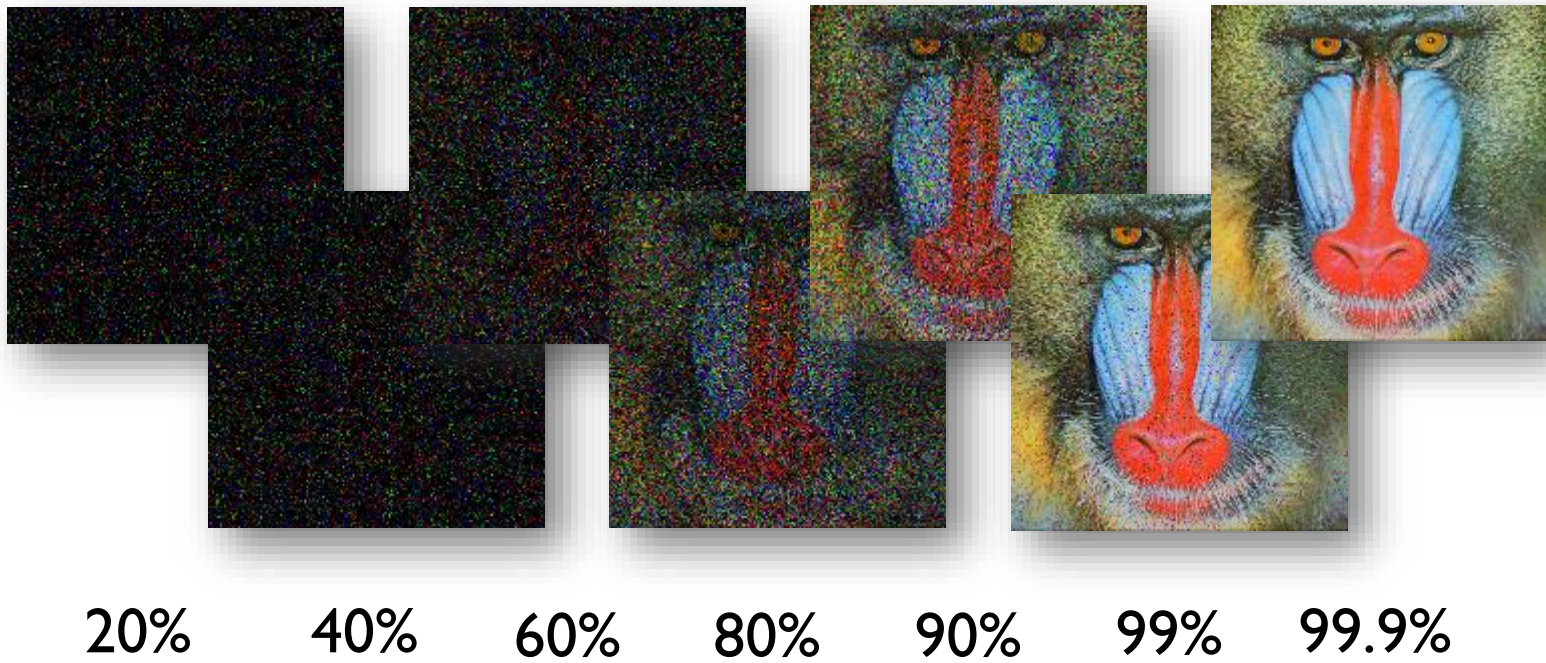
Magnitude of Noise



Difference d between the exact and approximate pixel values (for all color components)

Function's and Program's Accuracy

Frequency of Noise



Probability p with which interpolation kernel produces a correct pixel

Accuracy Requirement

Specify Metric and Threshold



- Each application has its own
- Require domain expertise (the only part!)
- For visual data, historically often PSNR is used
- 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 $m \times n$ monochrome image I and its noisy approximation K , *MSE* is defined as:

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

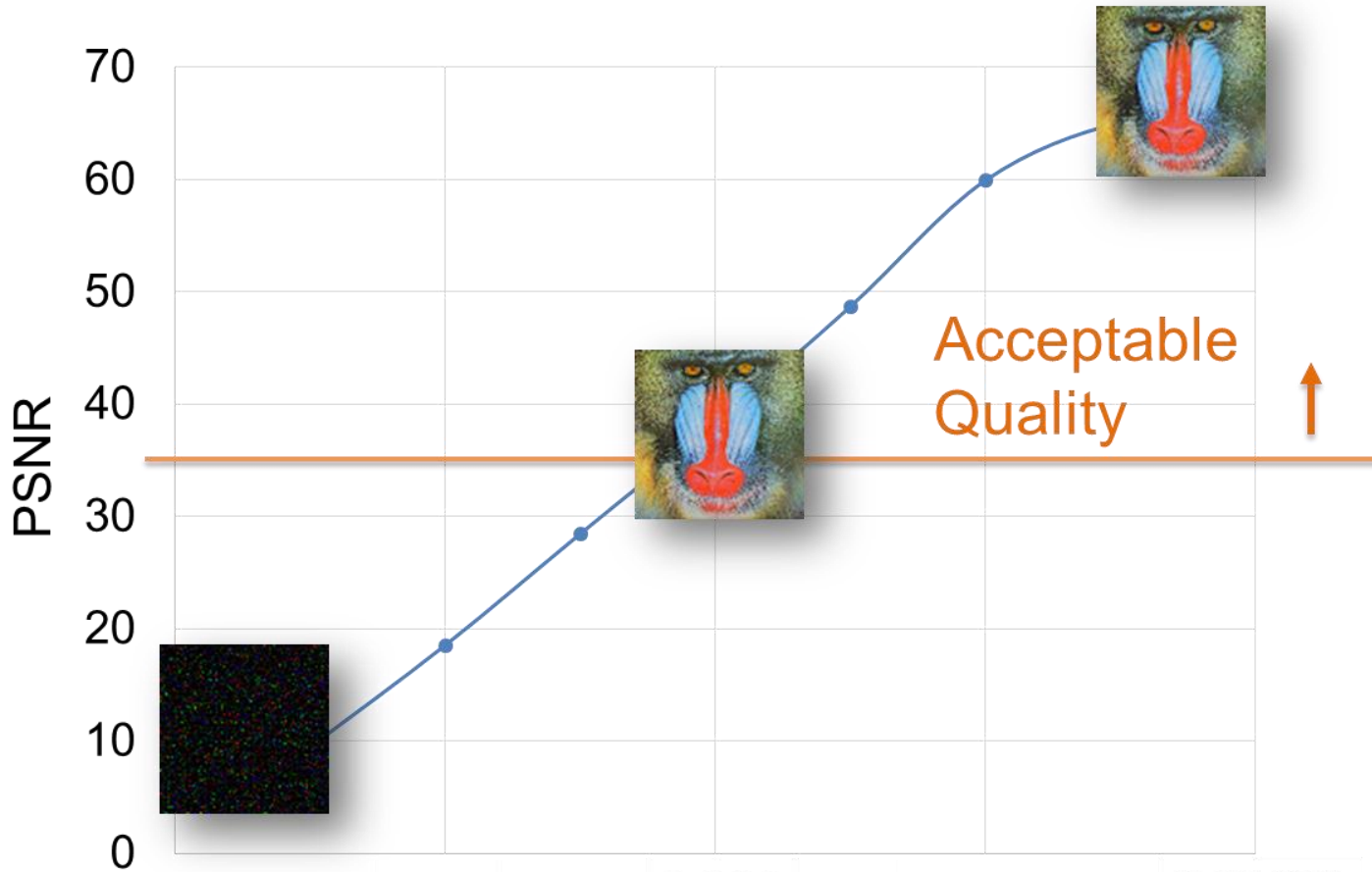
The PSNR (in [dB](#)) is defined as:

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

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

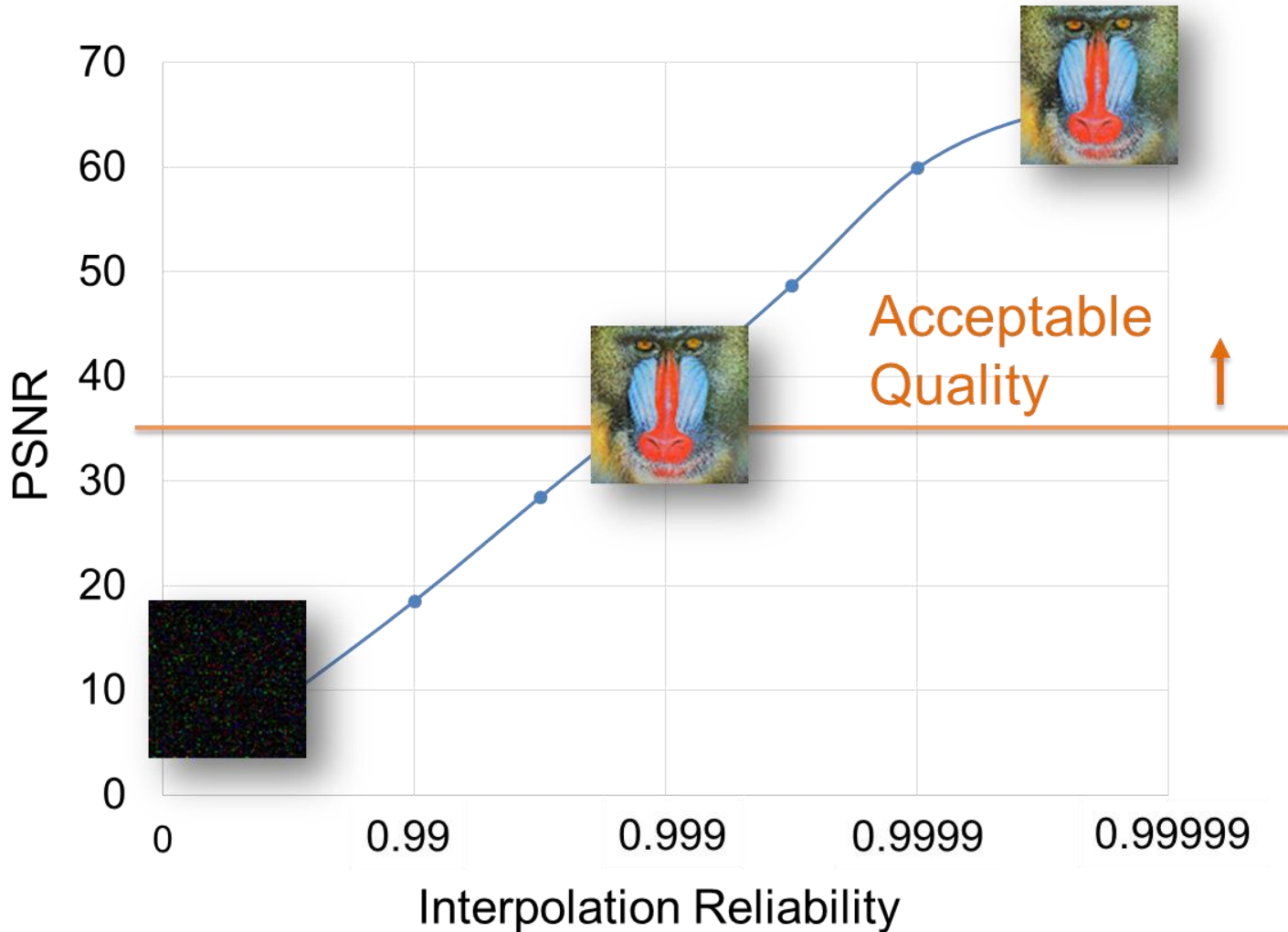
- You can compare outputs only at the end of the run
- Often better understood for representative domains

Kernel-level: each function has its 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

Pixel kernel reliability r

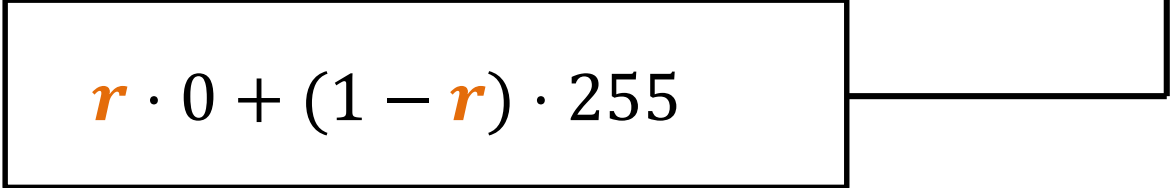
Global Specification

PSNR of the image

Computation Pattern

Data parallel loop

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

$$r \cdot 0 + (1 - r) \cdot 255$$


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

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

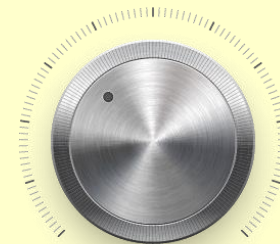
Original ✓
Computation

Accuracy ✓
Requirement

Accuracy-Aware Optimization

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

Optimized Computation +



Safari:

SOFTWARE TRANSFORMATIONS

Loop Perforation (2009)

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



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

Loop Perforation

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



```
for (i = 0; i < n/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);
```

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  $\exists x', v . x' \in [x[i] - \epsilon, x[i] + \epsilon] \ \&\& \ (x', v) \in \text{table}$   
        y[i] = v;  
    else {  
        y[i] = f(x[i]);  
        table[x[i]] = y[i];  
    }  
}
```

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:

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

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

Function Substitution

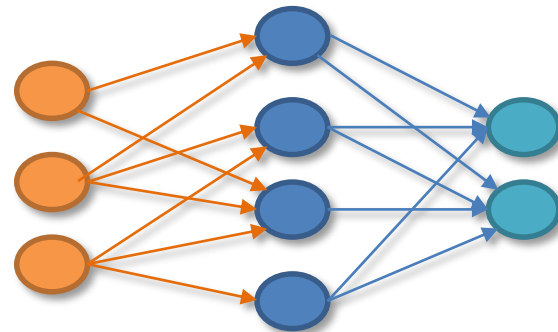
$$y = f(x);$$



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

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

Neural Network:



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

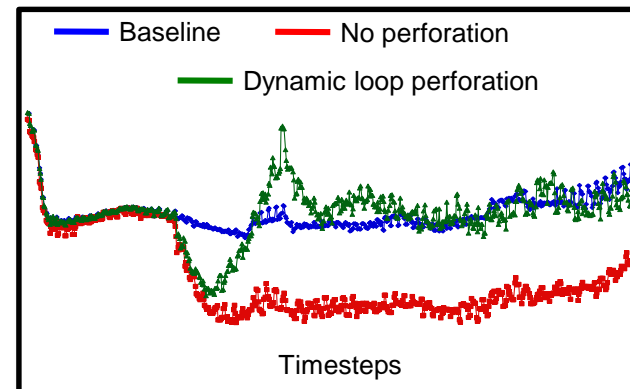
Dynamic Function Substitution

$y = f(x);$



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

$y = \text{runtime.executeApprox}();$
 $f'(x): f(x);$



- 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)
```

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**

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();
```

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)