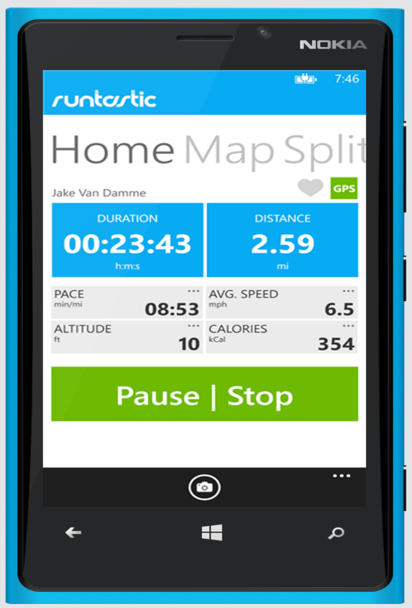
Uncertain<T>
Programming with Estimates

Kathryn S McKinley

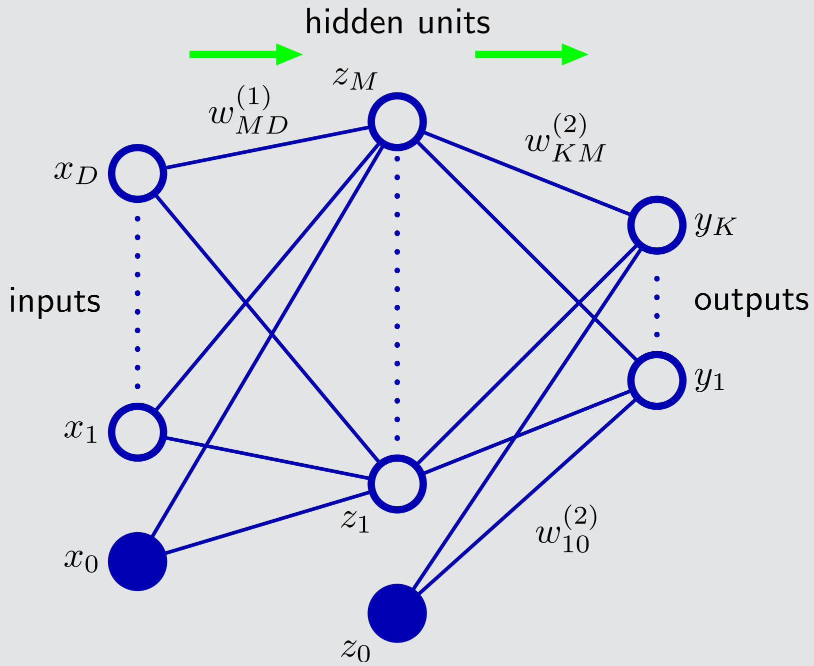
Microsoft Research

Uncertainty is everywhere

Sensors

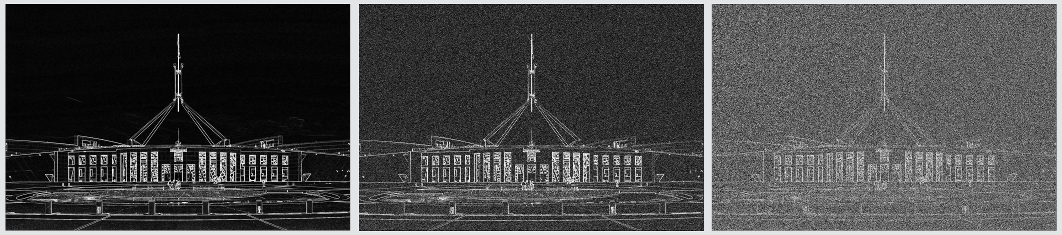


Machine learning

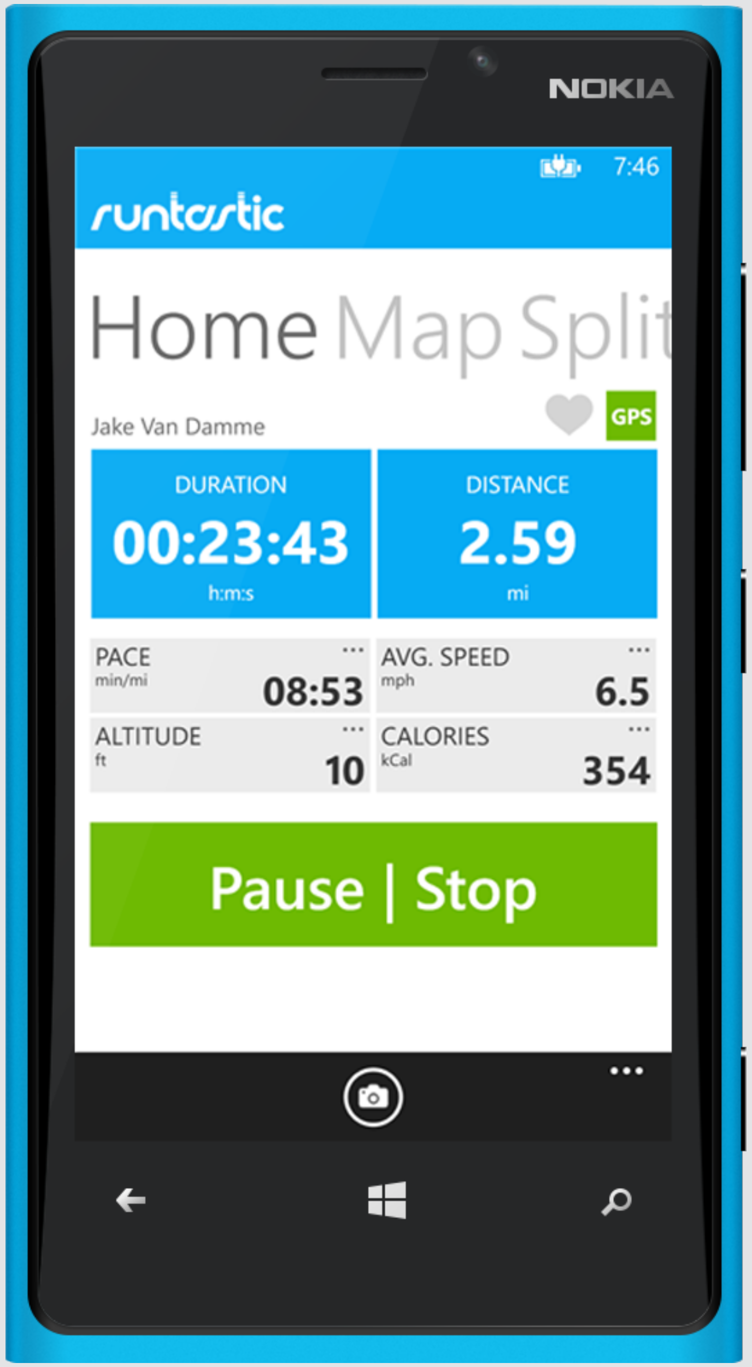


Approximate computing

approximate edge detection



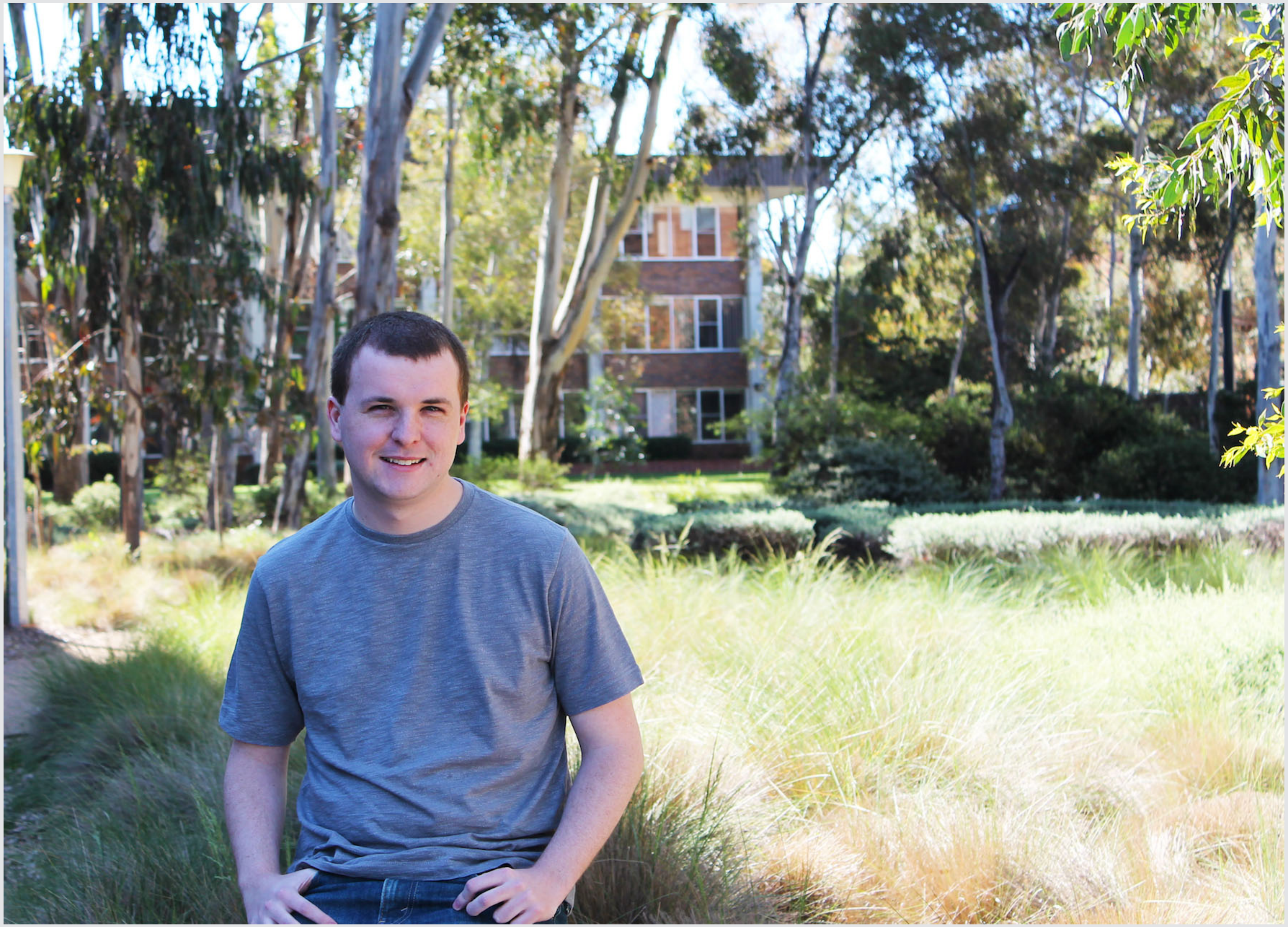
But we lack abstractions to help developers reason about uncertainty

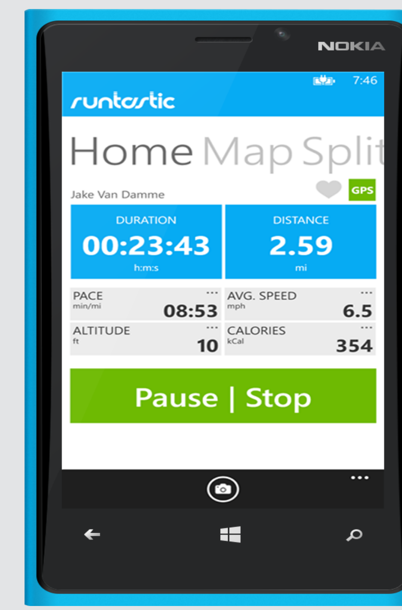

NOKIA
7:46
runtastic
Home Map Split
Jake Van Damme
GPS
DURATION
00:23:43
h:m:s
DISTANCE
2.59
mi
PACE
min/mi
08:53
AVG. SPEED
mph
6.5
ALTITUDE
ft
10
CALORIES
kCal
354
Pause | Stop

Usain Bolt is fast

24 mph

But James is faster…





59 mph

Programming with Estimates: Challenges

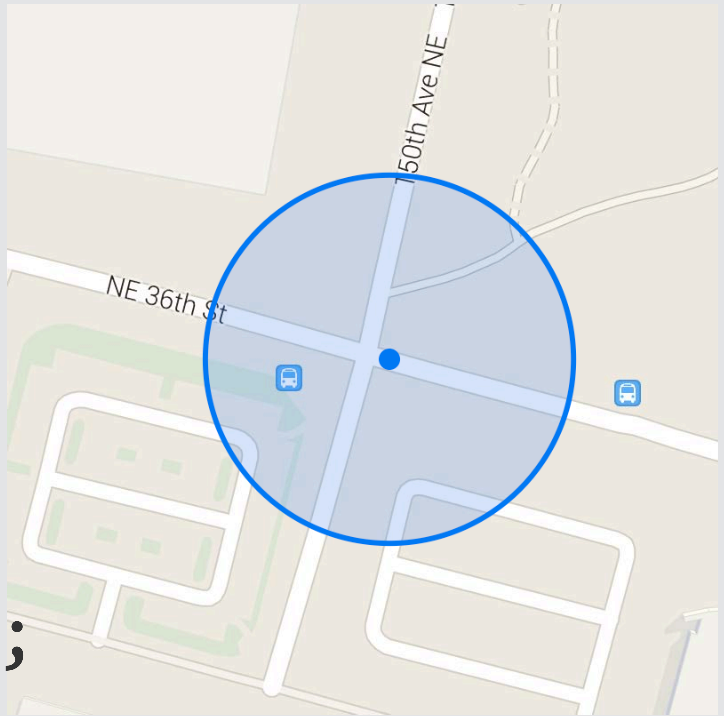
Estimates are noisy

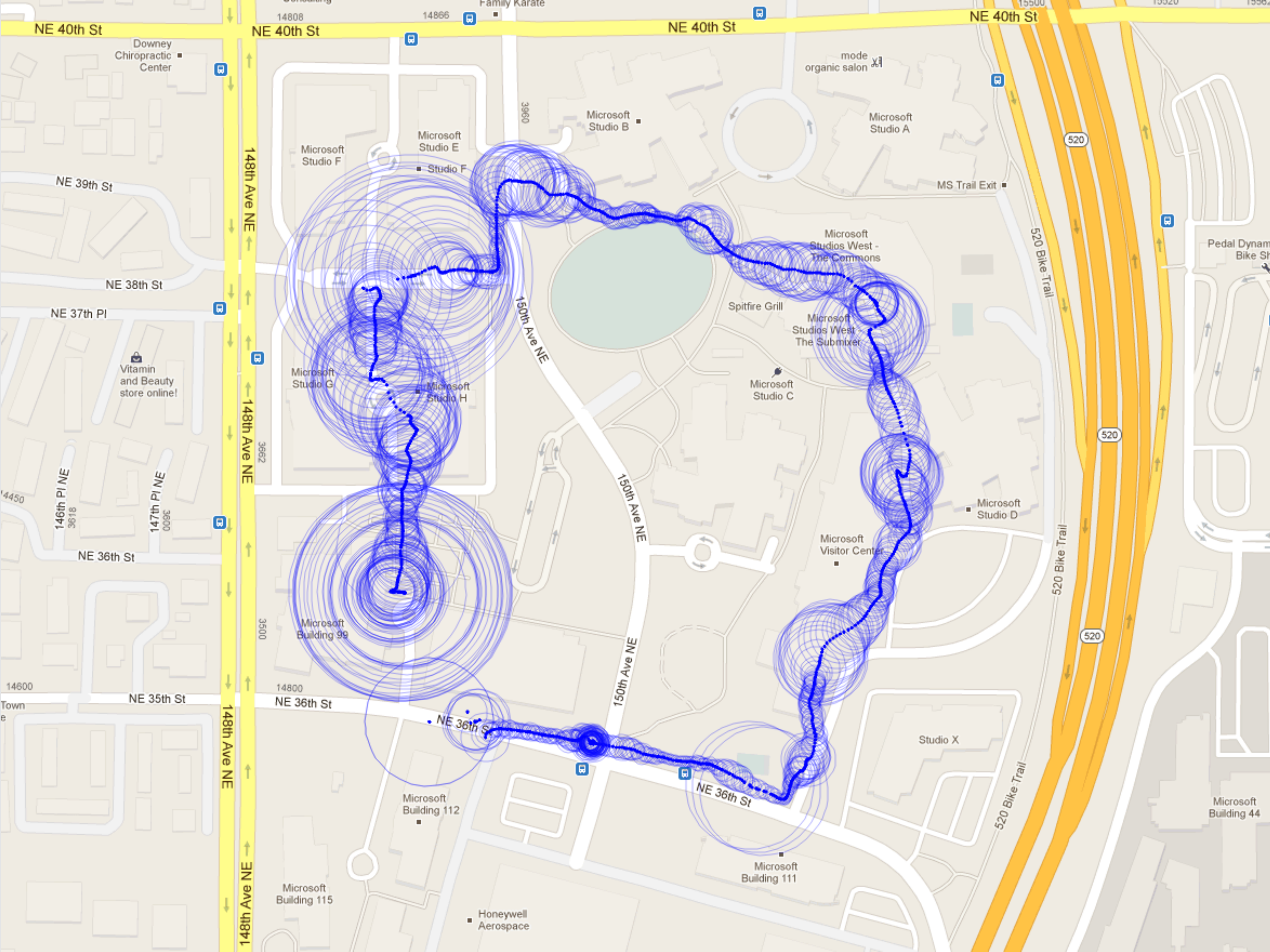
Improving estimates requires domain knowledge

What do these programs mean?

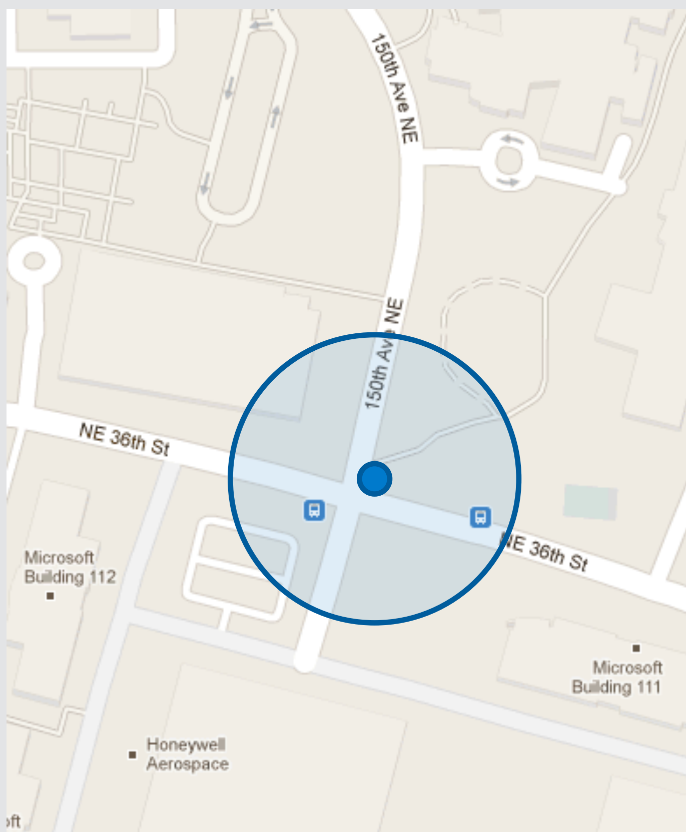
GPS locations

```
Loc lastloc = GPS.GetLocation();

double accuracy = GPS.GetAccuracy();

Map.DrawCircleWithCenter(lastloc, accuracy);
```




NE 40th St
Downey Chiropractic Center
Family Karate
mode organic salon
Microsoft Studio B
Microsoft Studio A
Microsoft Studio E
Microsoft Studio F
Studio F
NE 39th St
148th Ave NE
MS Trail Exit
NE 38th St
Microsoft Studios West - The Commons
NE 37th Pl
Spitfire Grill
Microsoft Studios West The Submixer
520 Bike Trail
Pedal Dynam Bike Sh
Vitamin and Beauty store online!
Microsoft Studio G
Microsoft Studio H
Microsoft Studio C
150th Ave NE
146th Pl NE
147th Pl NE
Microsoft Studio D
NE 36th St
Microsoft Visitor Center
Microsoft Building 99
NE 35th St
NE 36th St
Studio X
Microsoft Building 44
Microsoft Building 112
Microsoft Building 111
Microsoft Building 115
Honeywell Aerospace
520

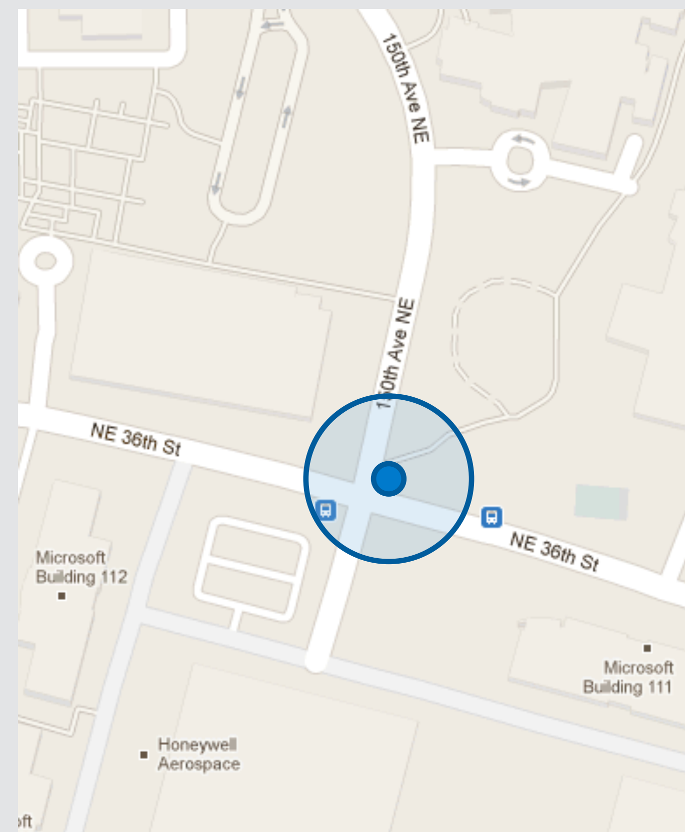
Which is more accurate?



Windows Phone

95% confidence interval

σ = 33 m

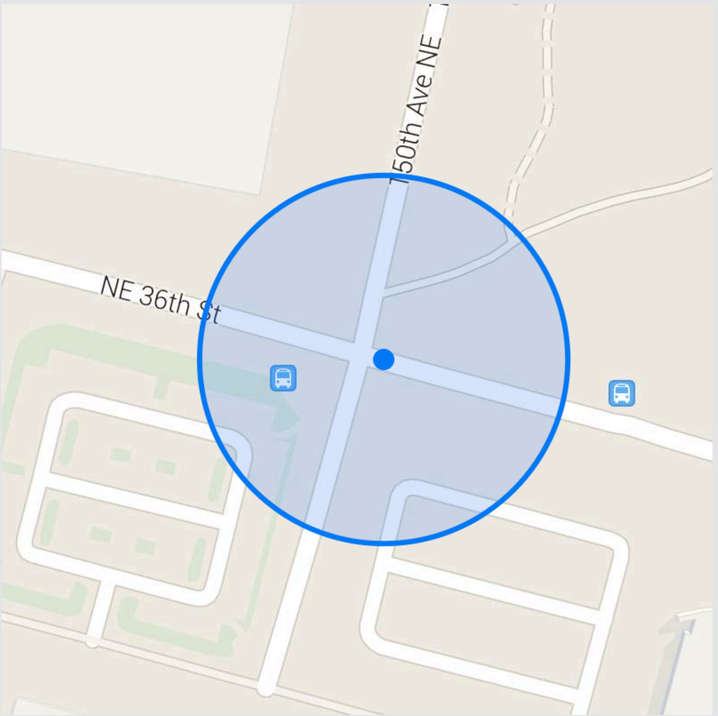


Android

68% confidence interval

σ = 39 m

Computing Speed from GPS




```
Loc lastloc = GPS.GetLocation();
Sleep(5);
Loc currloc = GPS.GetLocation();

double dist = GPS.Distance(currloc, lastloc);
double speed = dist / 5;

if (speed > 4) print("Great job!");

print(speed, accuracy);
```


GPS speed (95% CI)
Speed (mph)
60
50
40
30
20
10
0
Time
Great job!
Great job!
Great job!
Great job!
Great job!
Great job!
Great job!
Great job!

Problems

Using estimates as facts introduces errors

Computation compounds error

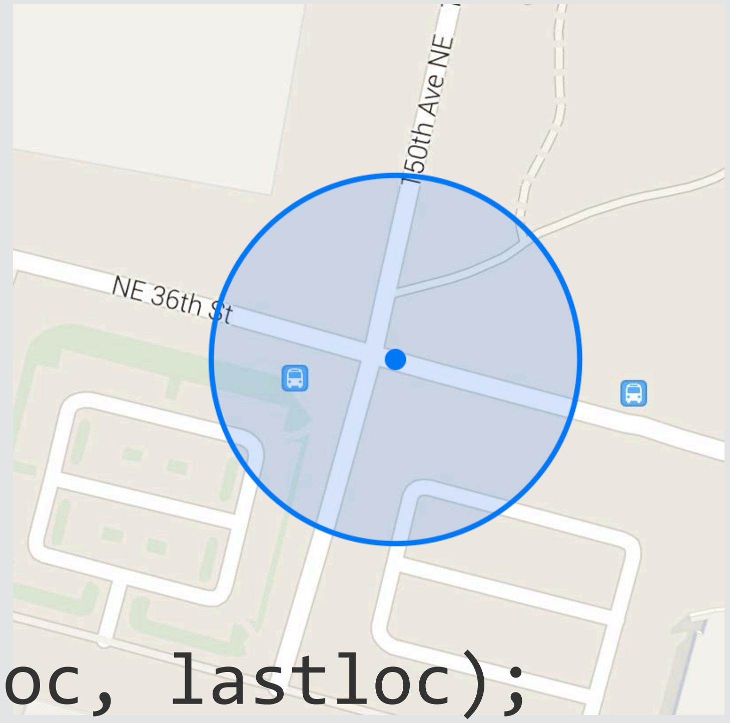
Boolean conditionals on probabilistic data introduce false positives and false negatives

Adding domain knowledge is adhoc and fragile

Uncertain<T>

Programming Model

Speed with Uncertain<T>

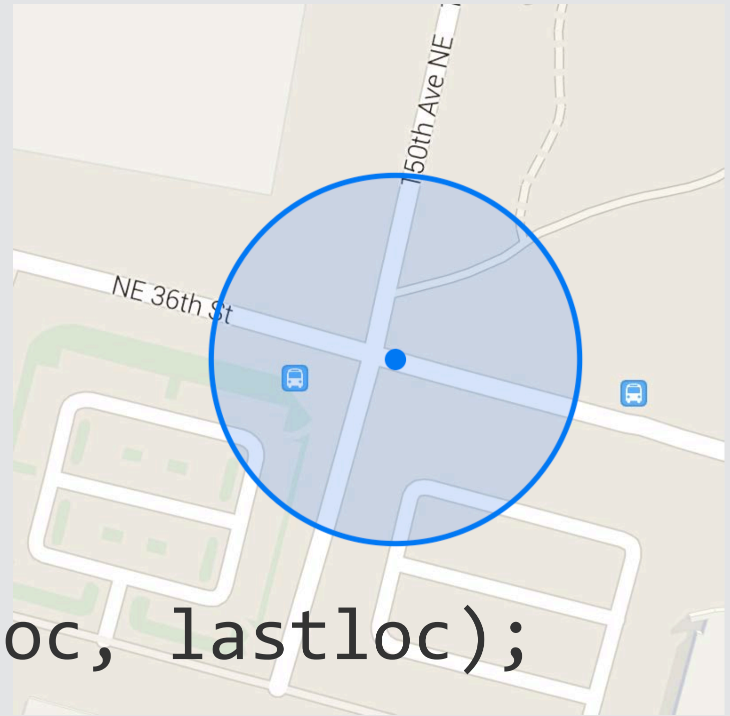



```
Uncertain<Loc> lastloc = GPS.GetLocation();
Sleep(5);
Uncertain<Loc> currloc = GPS.GetLocation();

Uncertain<double> dist = GPS.Distance(currloc, lastloc);
Uncertain<double> speed = dist / 5;
```

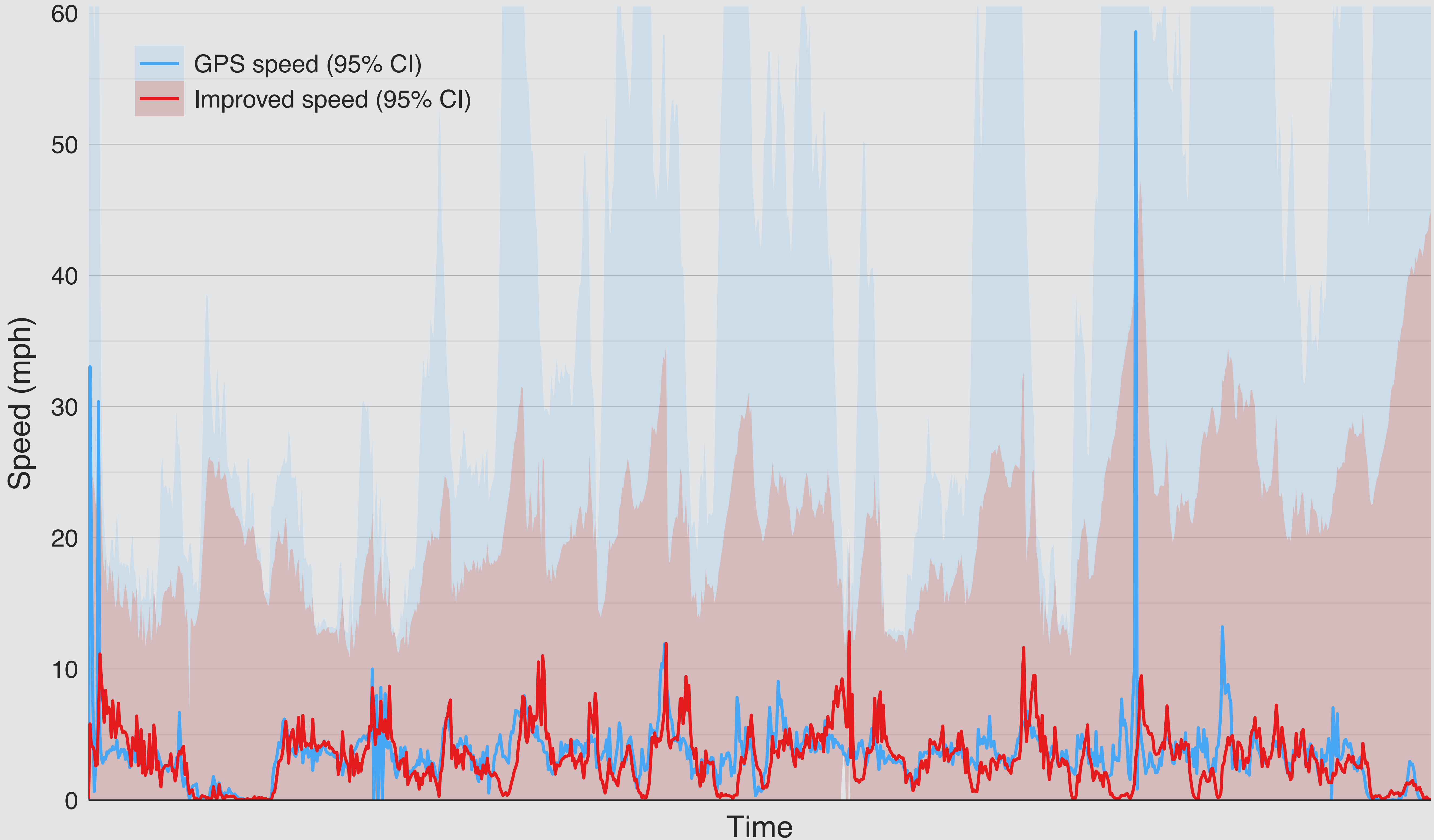
```
if ((newSpeed > 4).Pr(0.9)) print("Great job!");
```

```
print(speed, accuracy);
```

Speed with Uncertain<T>




```
Uncertain<Loc> lastloc = GPS.GetLocation();
Sleep(5);
Uncertain<Loc> currloc = GPS.GetLocation();

Uncertain<double> dist = GPS.Distance(currloc, lastloc);
Uncertain<double> speed = dist / 5;

Uncertain<double> walkPrior = new Uncertain<double> (()=>
   SamplePrior(0 mph, 10 mph, accuracy));

Uncertain<double> newSpeed = speed # walkPrior;

if (newSpeed > 4).Pr(0.9)) print("Great job!");

print(speed, accuracy);
```


GPS speed (95% CI)
Improved speed (95% CI)
Speed (mph)
Time
0
10
20
30
40
50
60

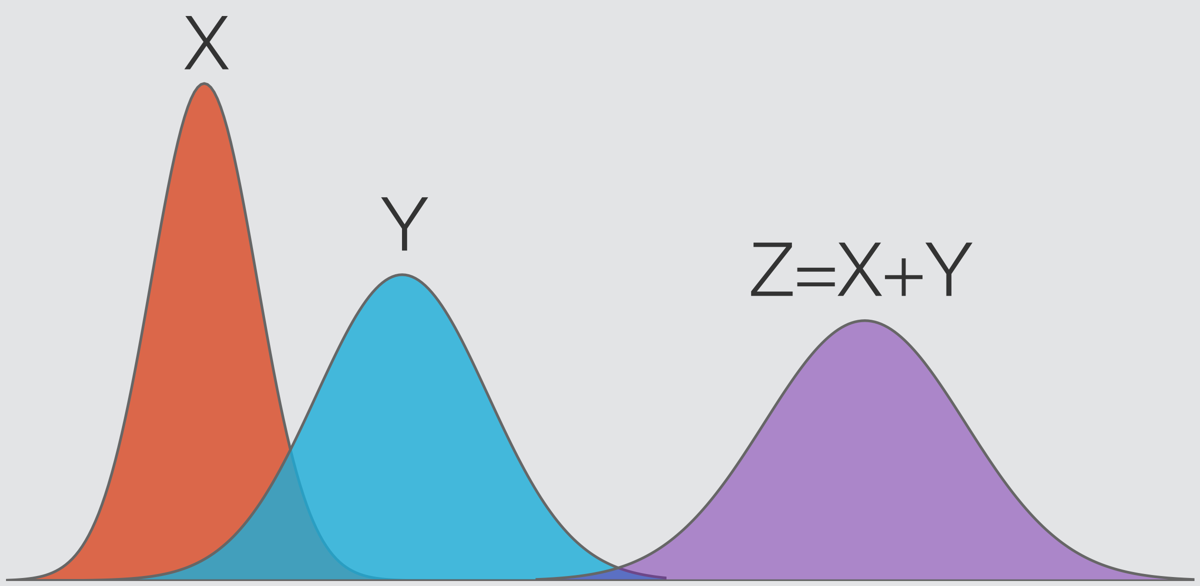
Uncertain<T>

Implementation

Semantics

```
Uncertain<double> Z = X + Y
```

Z is a random variable we represent as a distribution



If x	is a sample of X
and y	is a sample of Y
then $x+y$	is a sample of $X+Y$ *

* if X and Y are independent

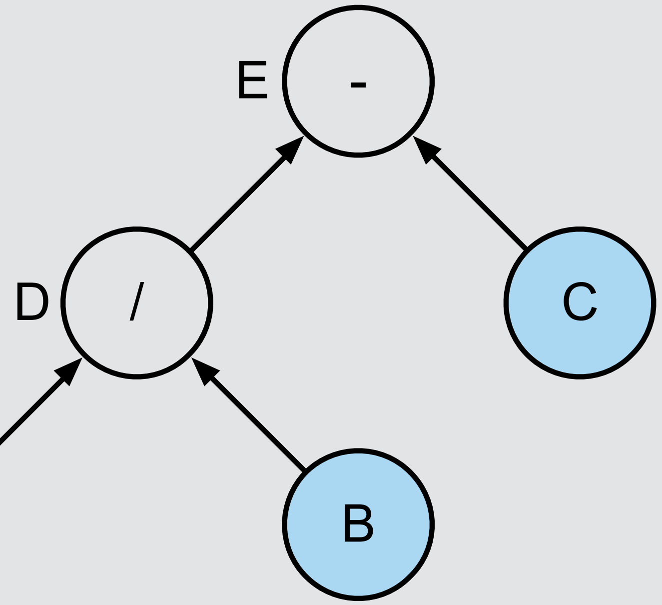
Sampling functions return random samples

- ✓ Simple computations
- ✓ Represent arbitrary distributions
- ✗ Sampling is approximate

Later: how Uncertain<*T*> learned to love approximation, and you can too

```
D = A / B
E = D - C
```

Bayesian network representation



Sampling function for **`E`** recursively samples children

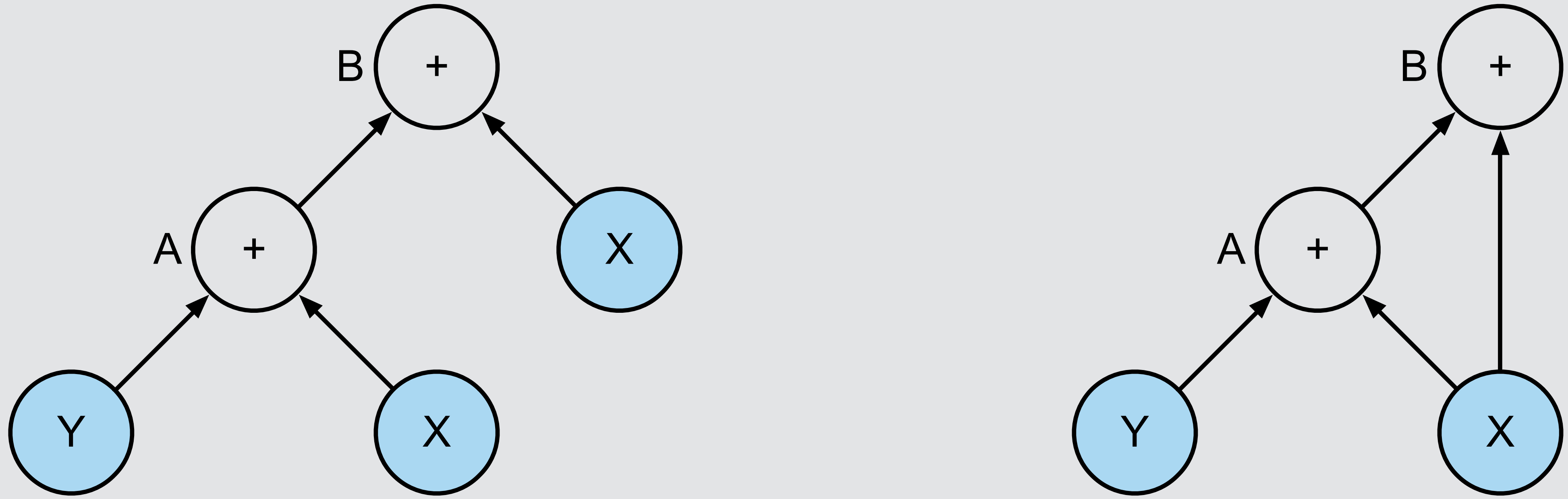
If x is a sample of X
and y is a sample of Y
then $x+y$ is a sample of $X+Y$ *

* Only if X and Y are independent.

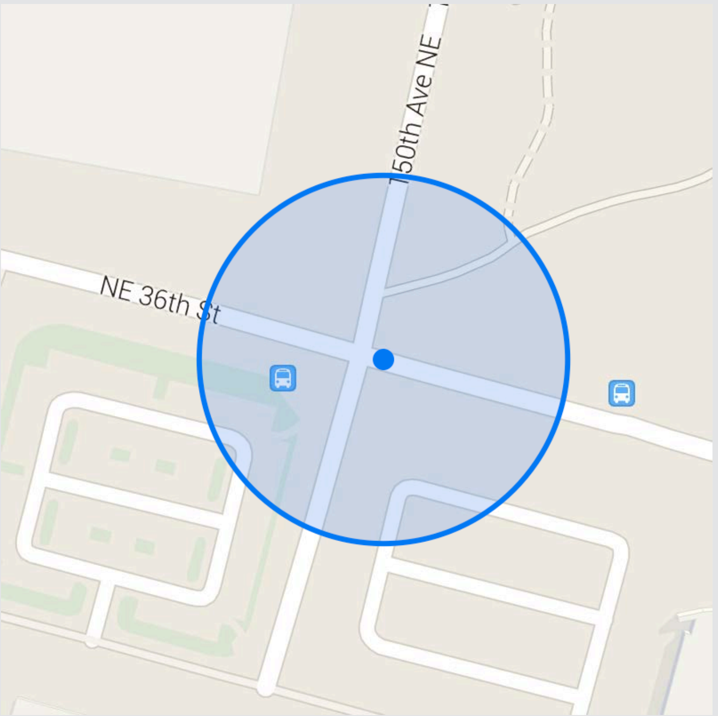
```
A = X + Y
B = A + X
```

(X,Y independent)

A and B depend on X – not independent!



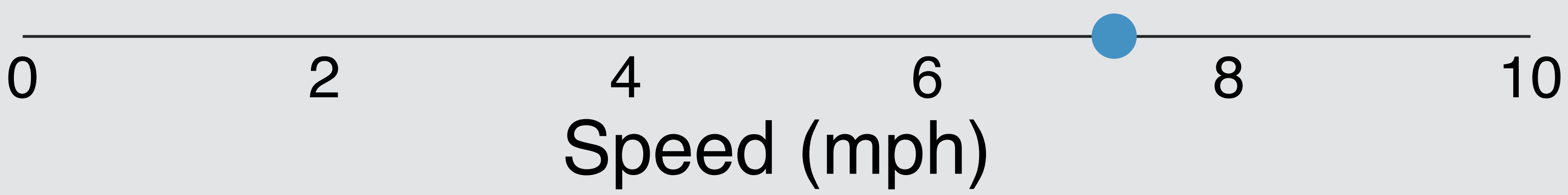
Speed with Uncertain<T>



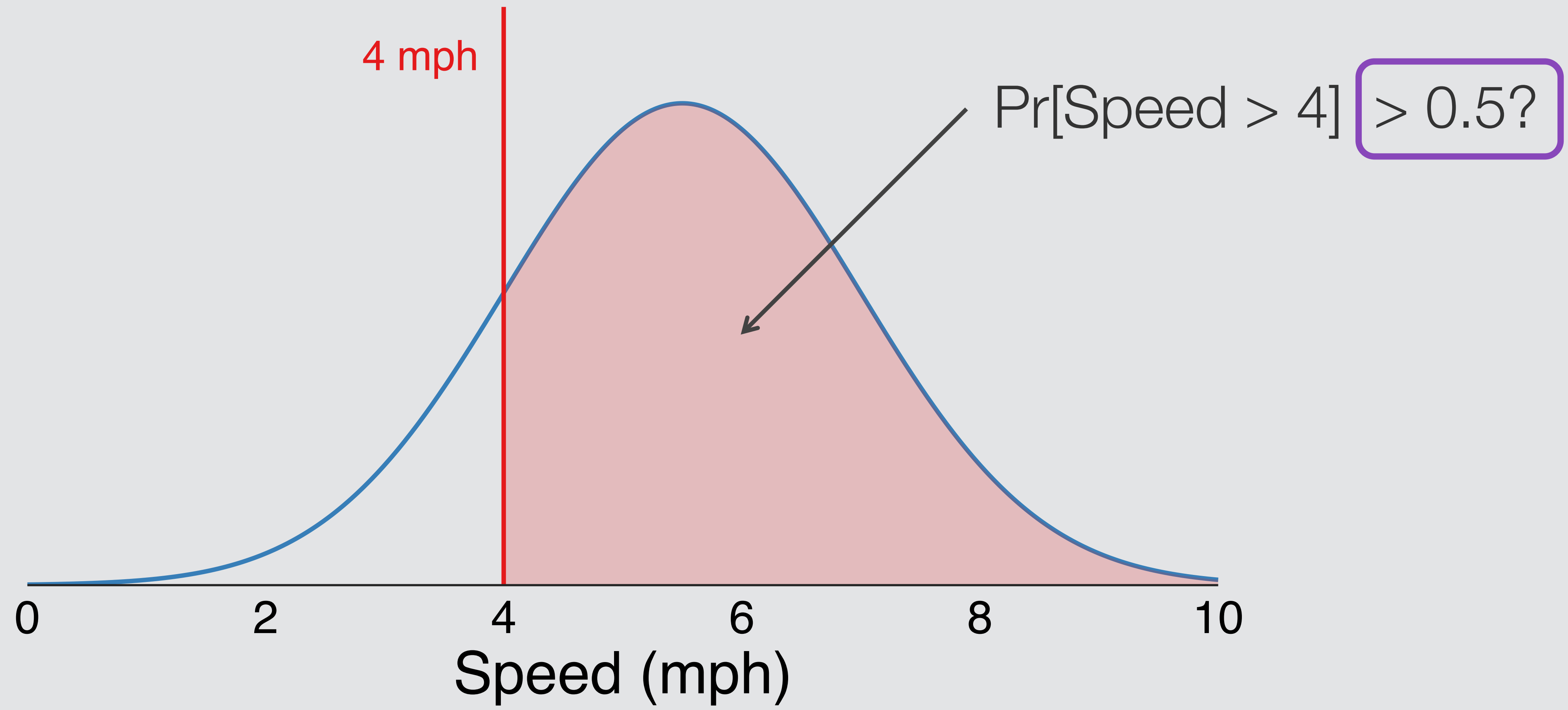

```
Uncertain<Loc> lastloc = GPS.GetLocation();
Sleep(5);
Uncertain<Loc> currloc = GPS.GetLocation();

Uncertain<double> dist = GPS.Distance(currloc, lastloc);
Uncertain<double> speed = dist / 5;

if (speed > 4) print("Great job!");
```

Hypothesis Test

```
if (speed > 4) print("Great job!”)
```

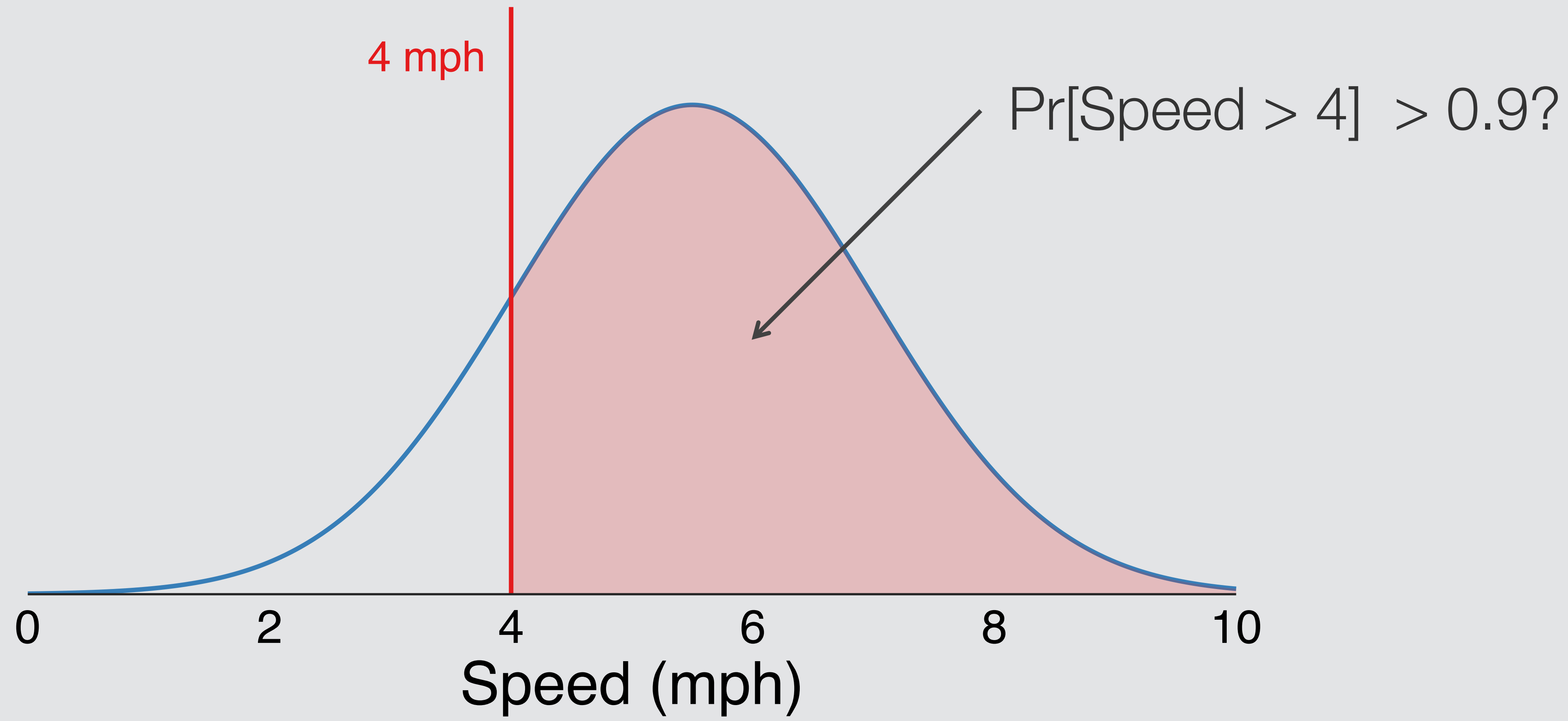


```
if (speed > 4) print("Great job!”)
```



More likely than not that Speed > 4?

```
if (speed > 4).Pr(0.9)
                        print("Great job!”)
```



At least 90% likely that Speed > 4?

```
if (speed > 4) print("Great job!”)
```

null hypothesis H_0: $\Pr[\text{Speed} > 4] \leq 0.5$

alternate hypothesis H_A: $\underbrace{\Pr[\text{Speed} > 4]}_{\text{approximate}} > 0.5$

How many samples? Too many = too slow

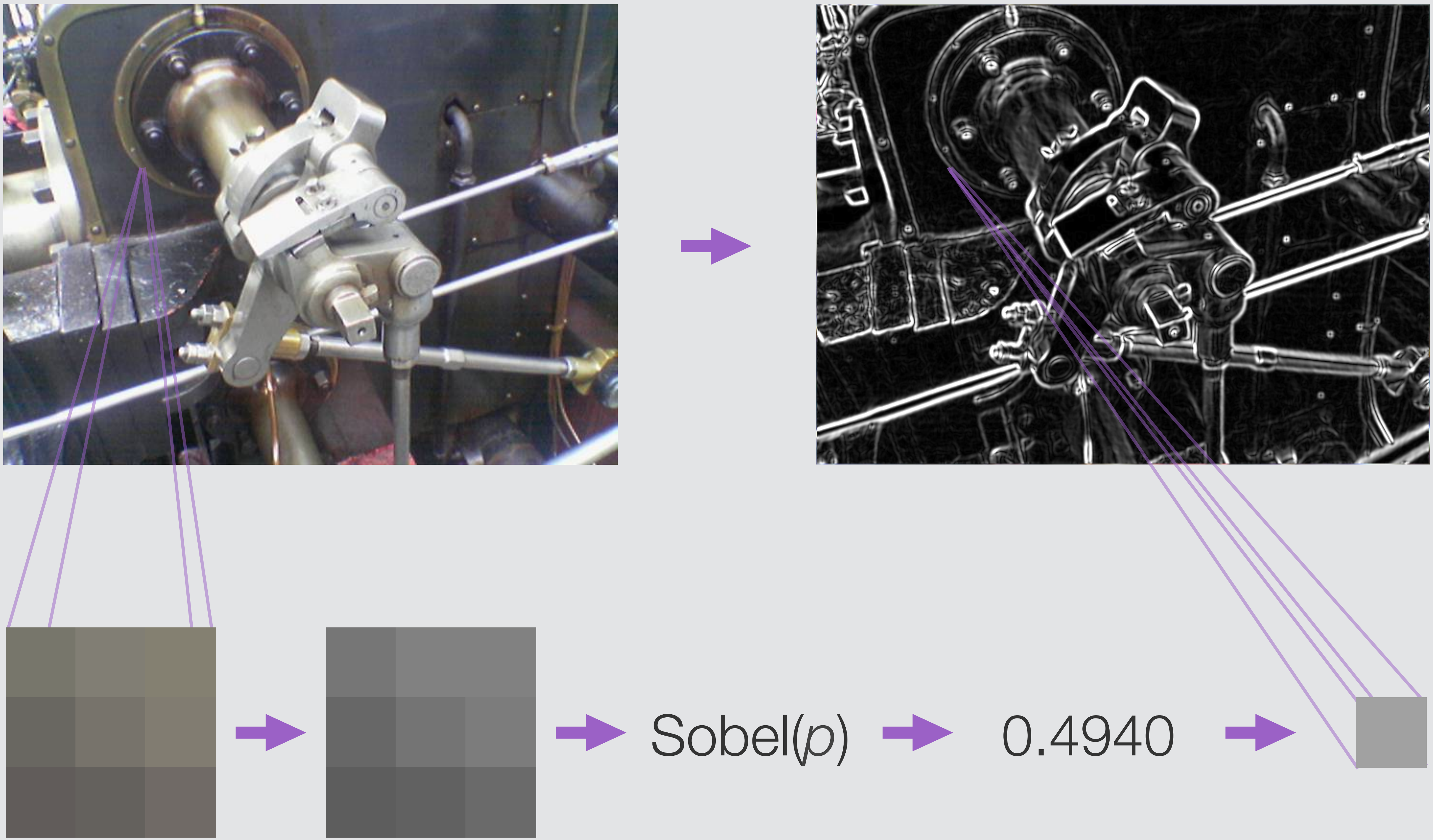
Too few = too noisy

Sequential sampling: sample size depends on progress

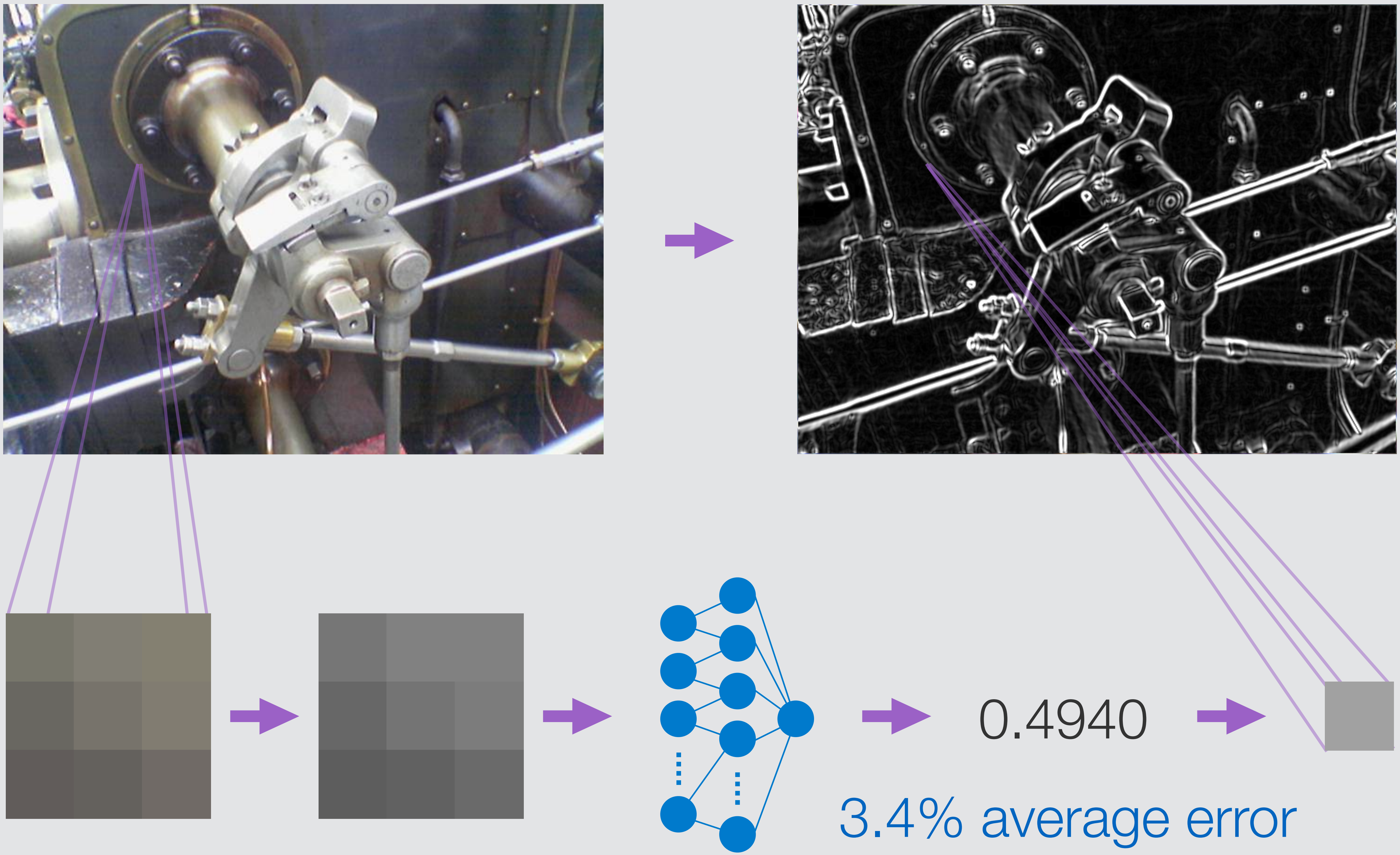
Uncertain<T>

Mitigates Bugs from Random Error

Edge detection



Approximate edge detection



Approximate edge detection

What is the gradient at pixel *p*?

```
Sobel(p)
```

3.4% average training error

Is there an edge at pixel *p*?

```
if (Sobel(p) > 0.1)
    EdgeFound();
```

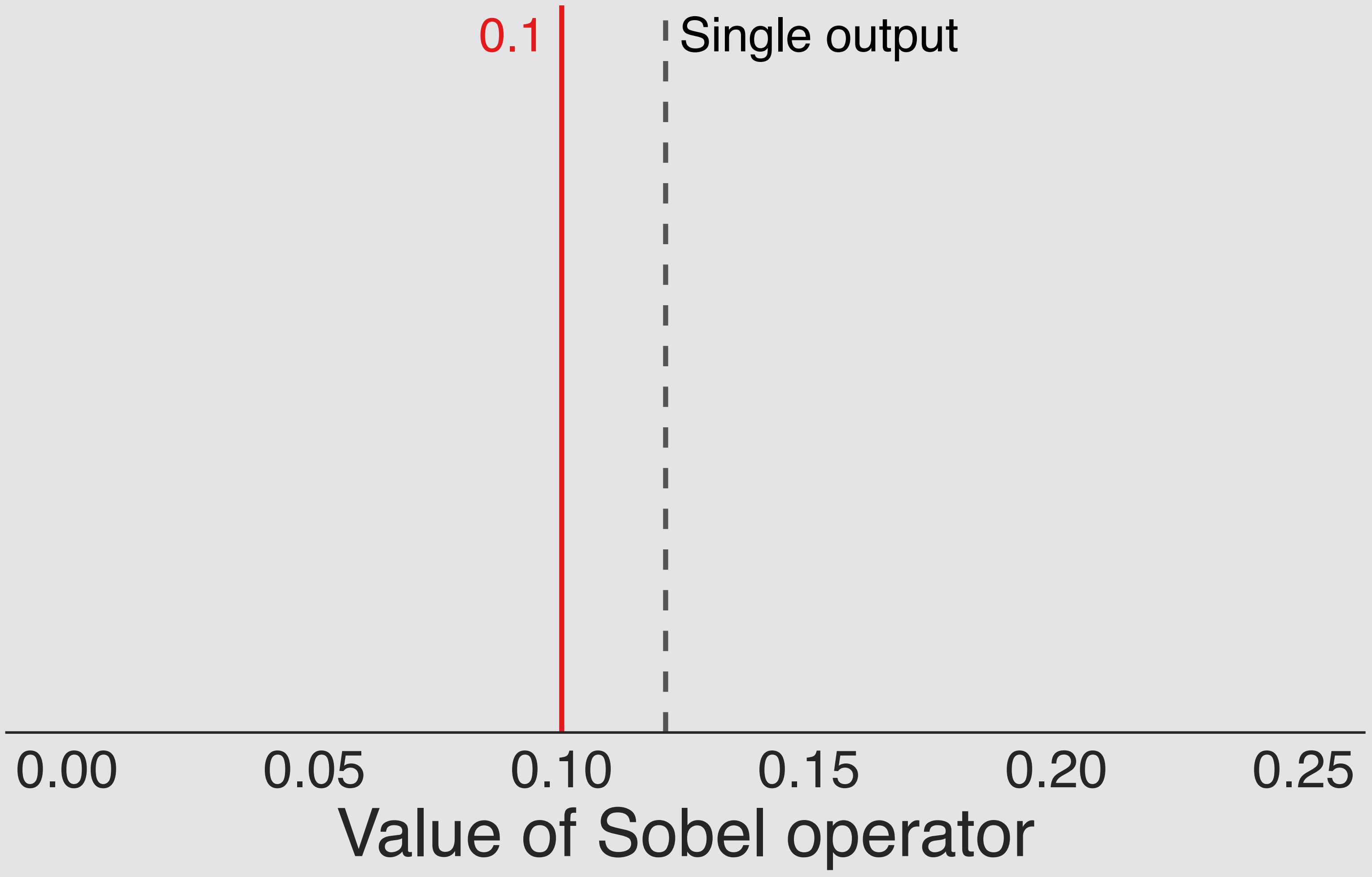
36% false positives on the same data!

Computation compounds uncertainty!

Is there an edge at pixel p?

```
if (Sobel(p) > 0.1)
    EdgeFound();
```

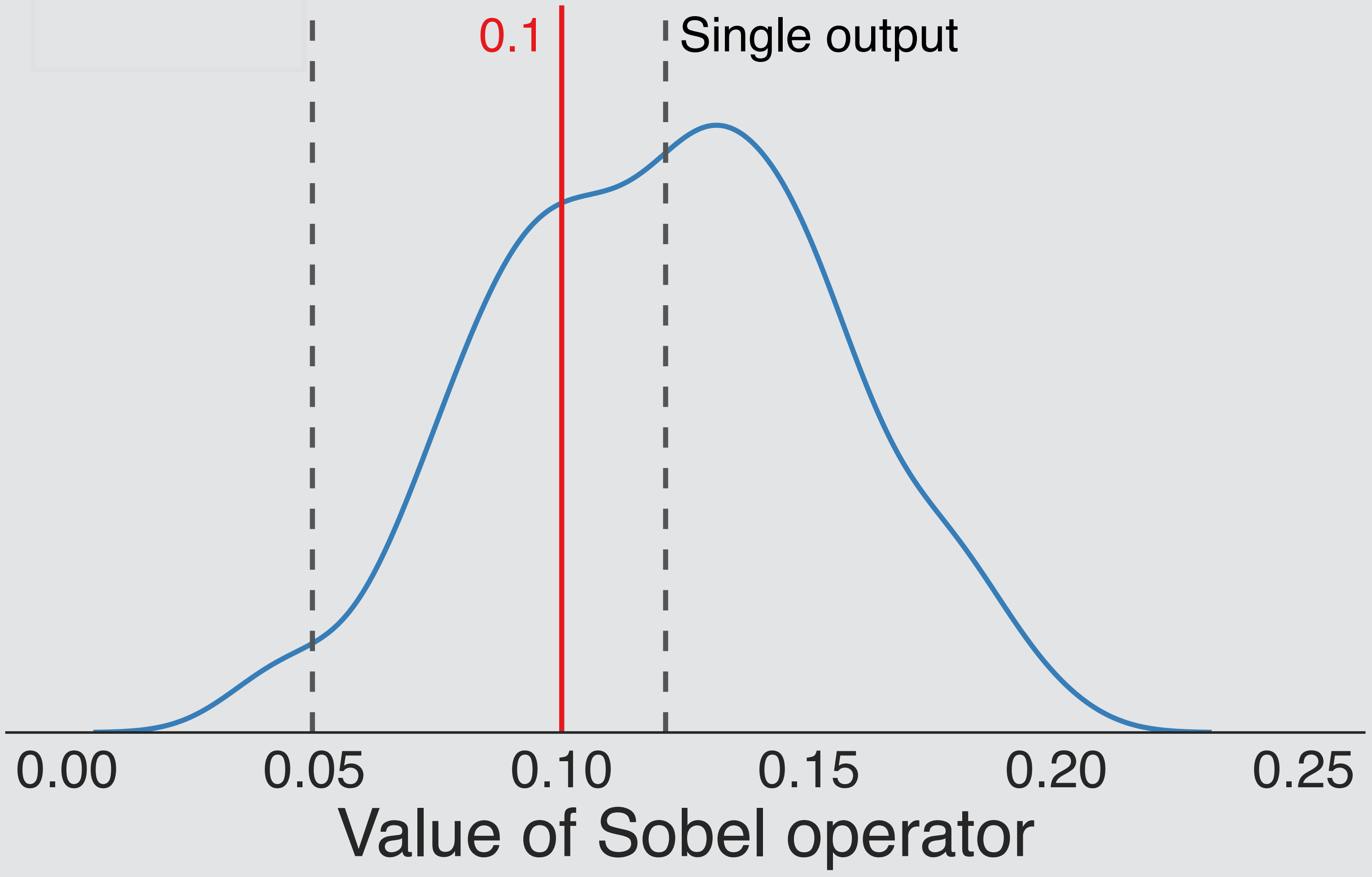
36% false positives!



Is there an edge at pixel p?

```
if (Sobel(p) > 0.1)
    EdgeFound();
```

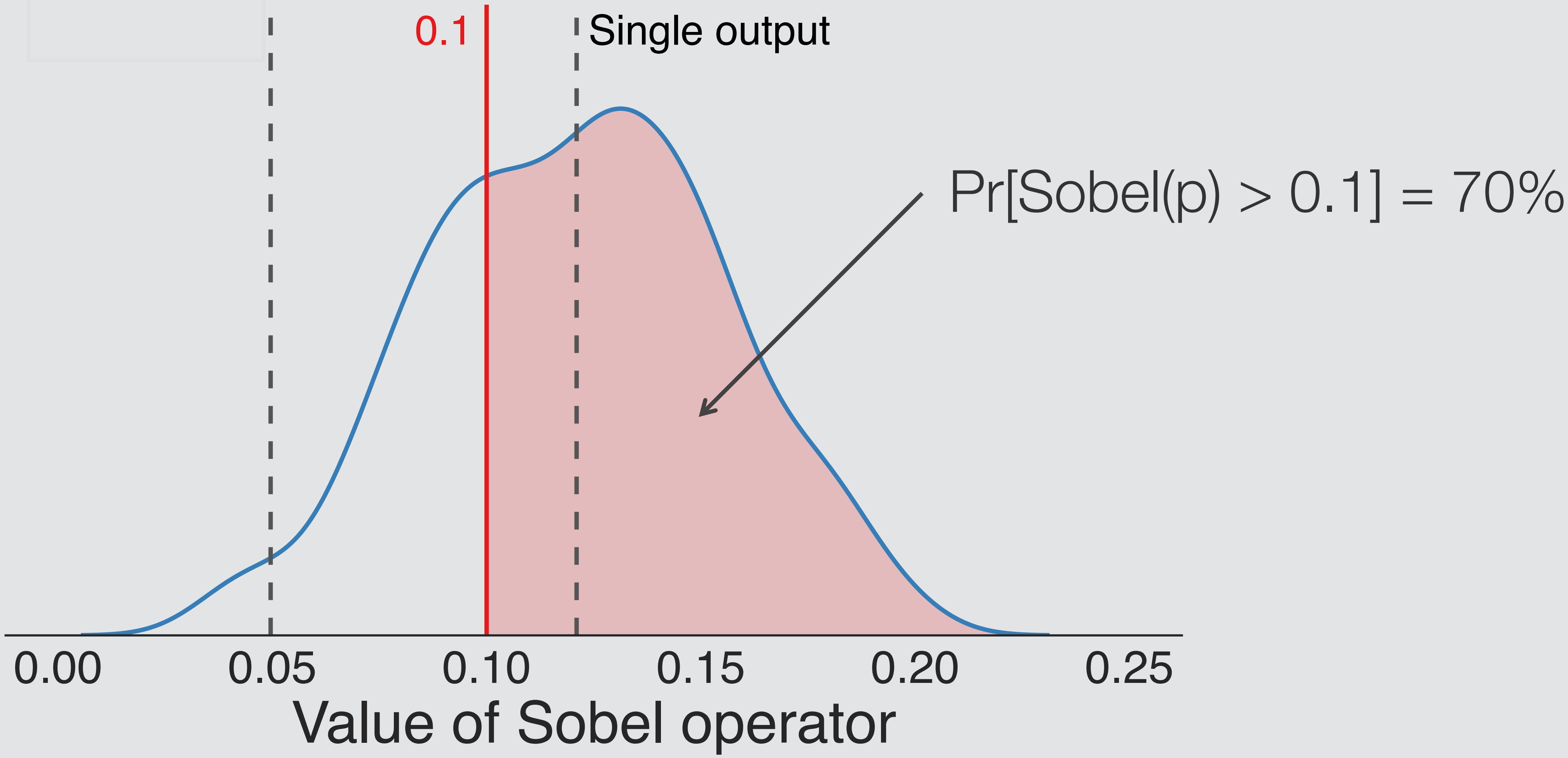
36% false positives!

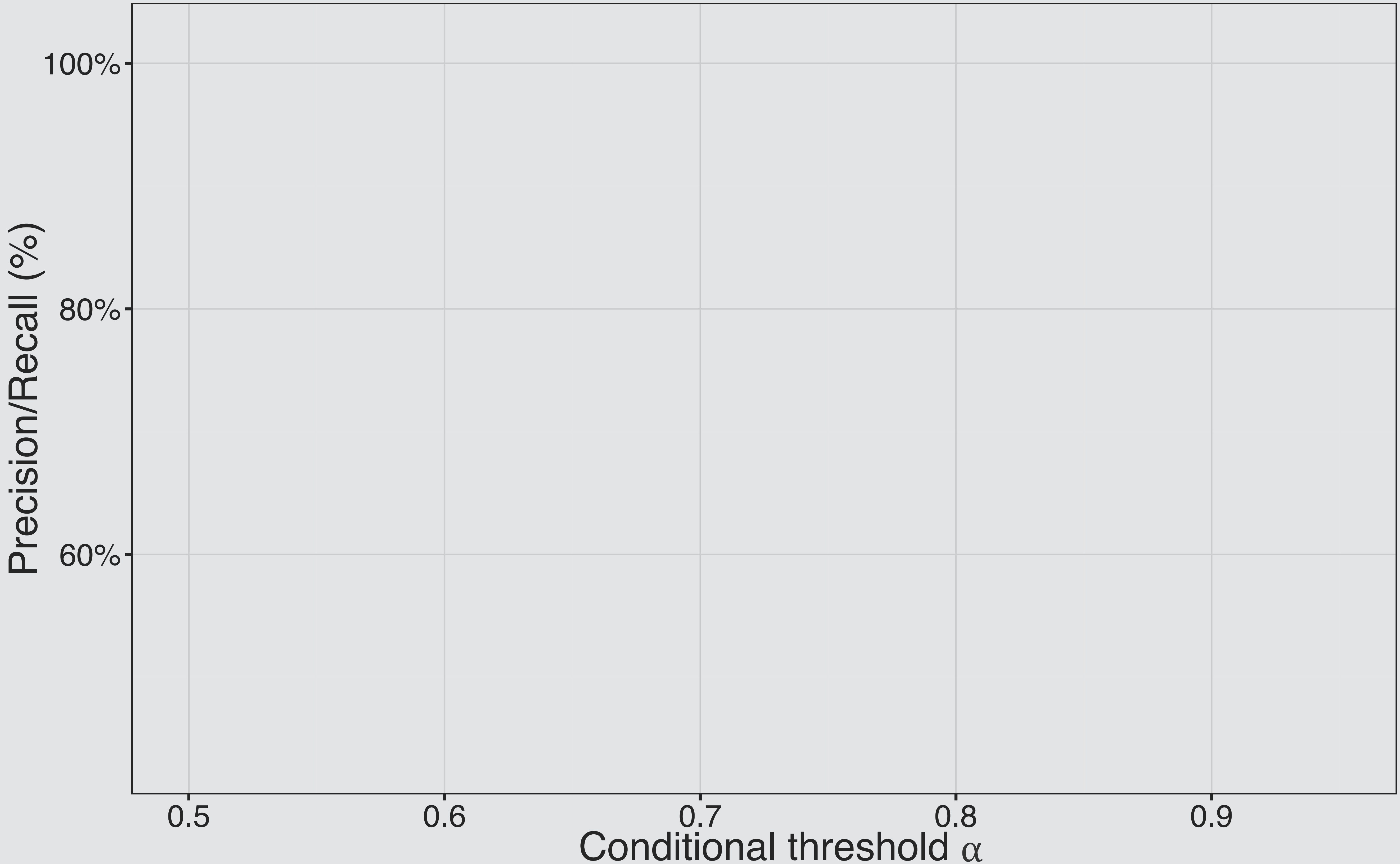


Is there an edge at pixel p?

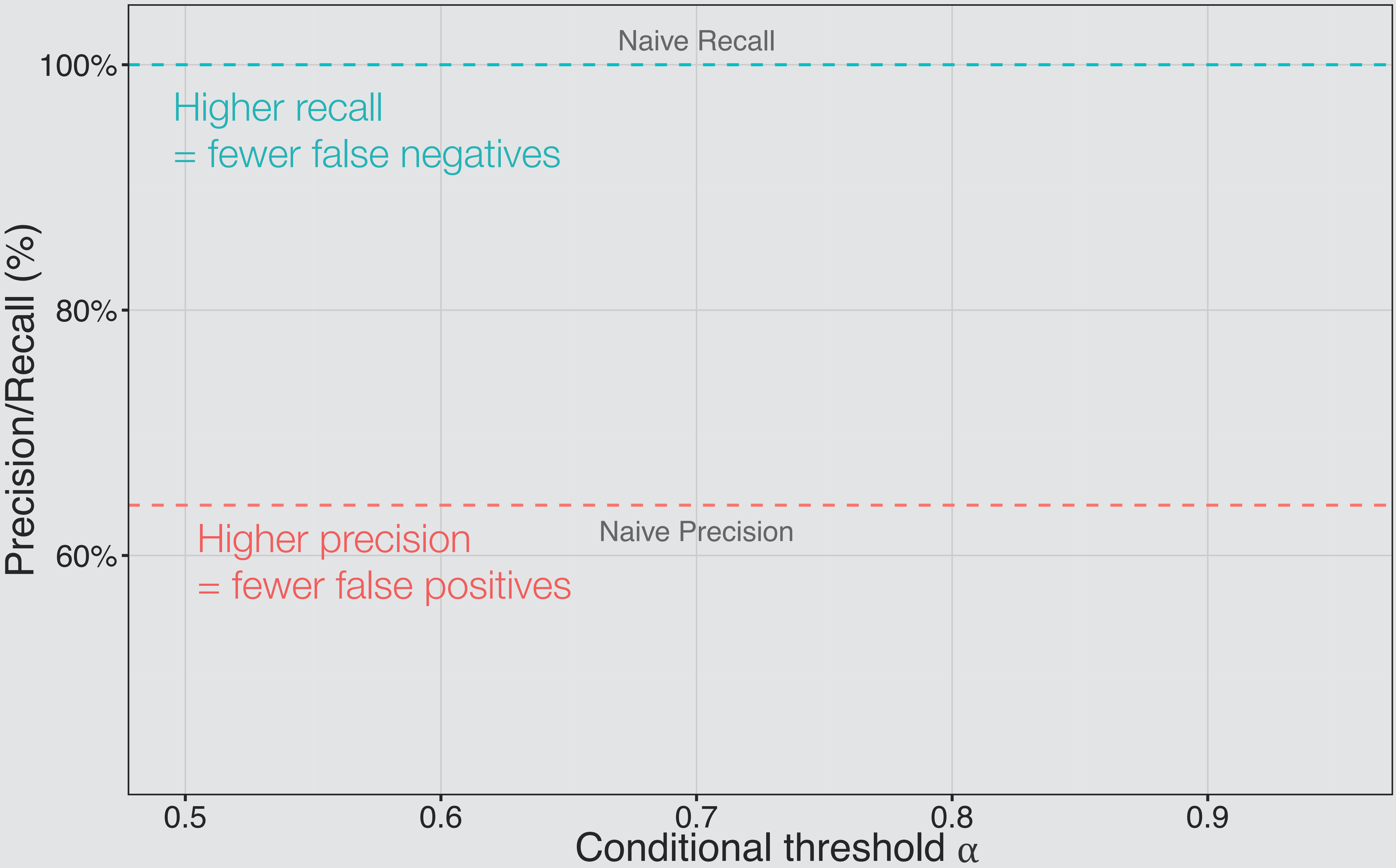
```
if (Sobel(p) > 0.1)
    EdgeFound();
```

36% false positives!

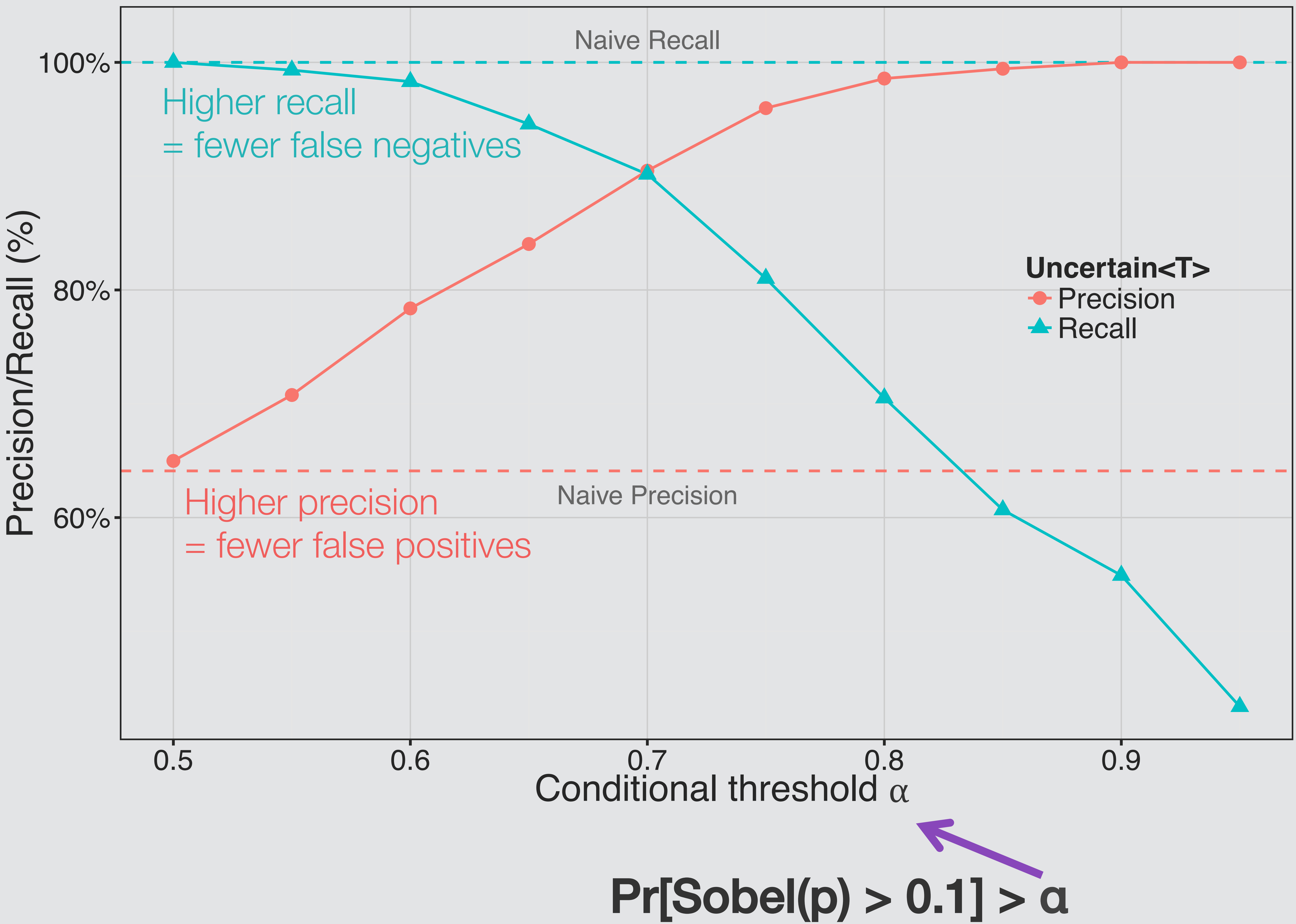




Pr[Sobel(p) > 0.1] > α



$$\Pr[\mathrm{Sobel}(p) > 0.1] > \alpha$$


Naive Recall
100%
80%
60%
Higher recall
= fewer false negatives
Higher precision
= fewer false positives
Naive Precision
Uncertain<T>
Precision
Recall
Precision/Recall (%)
0.5
0.6
0.7
0.8
0.9
Conditional threshold α
Pr[Sobel(p) > 0.1] > α

Uncertain<T>

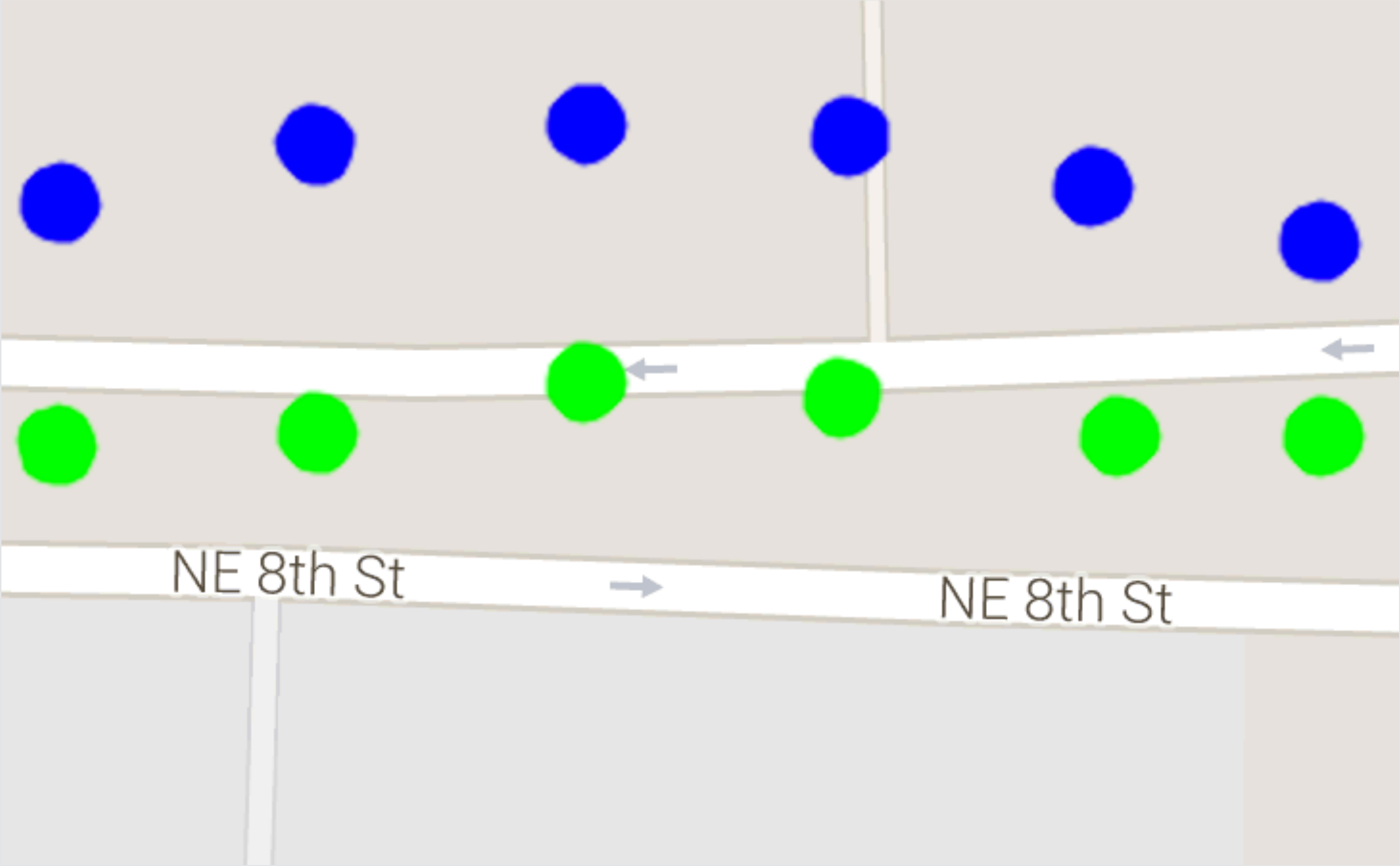
Mitigates Bugs from Random Error

Many Estimates are Inherently Noisy!
Add Domain Knowledge

GPS Navigation

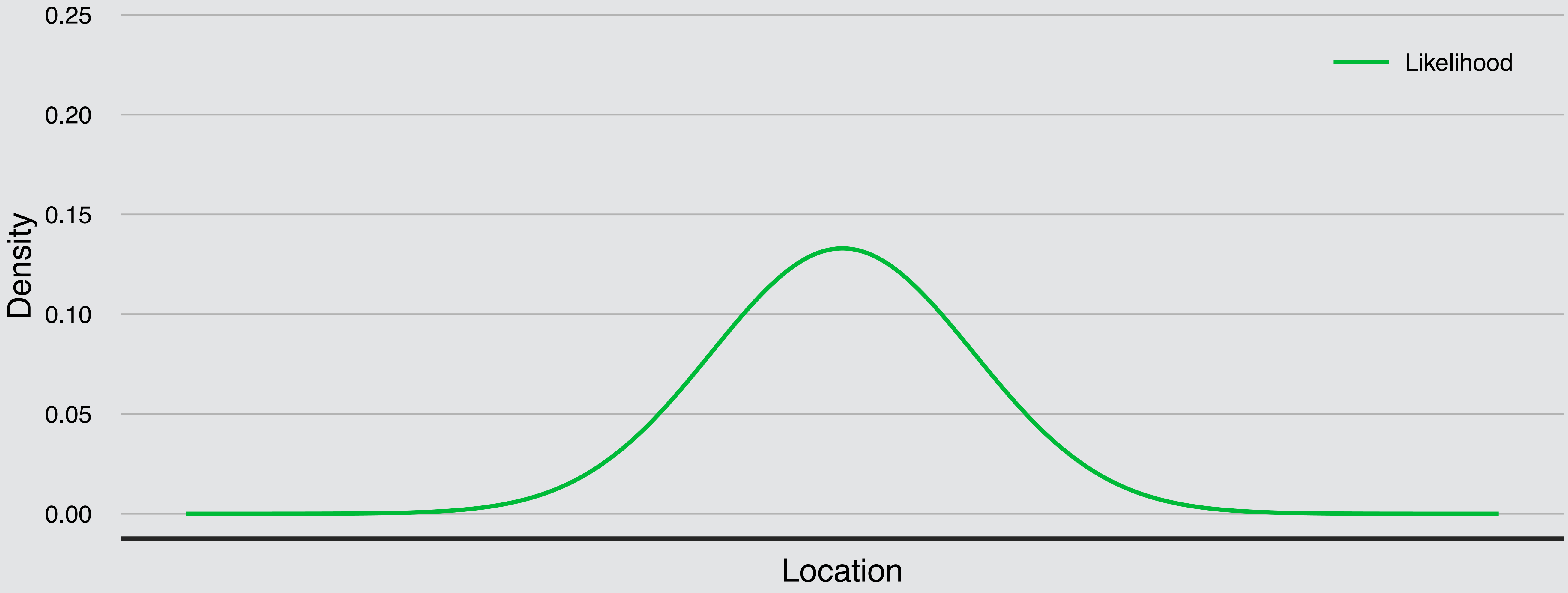
Driver is *likely* on a road!

Driving on a road (or not!)



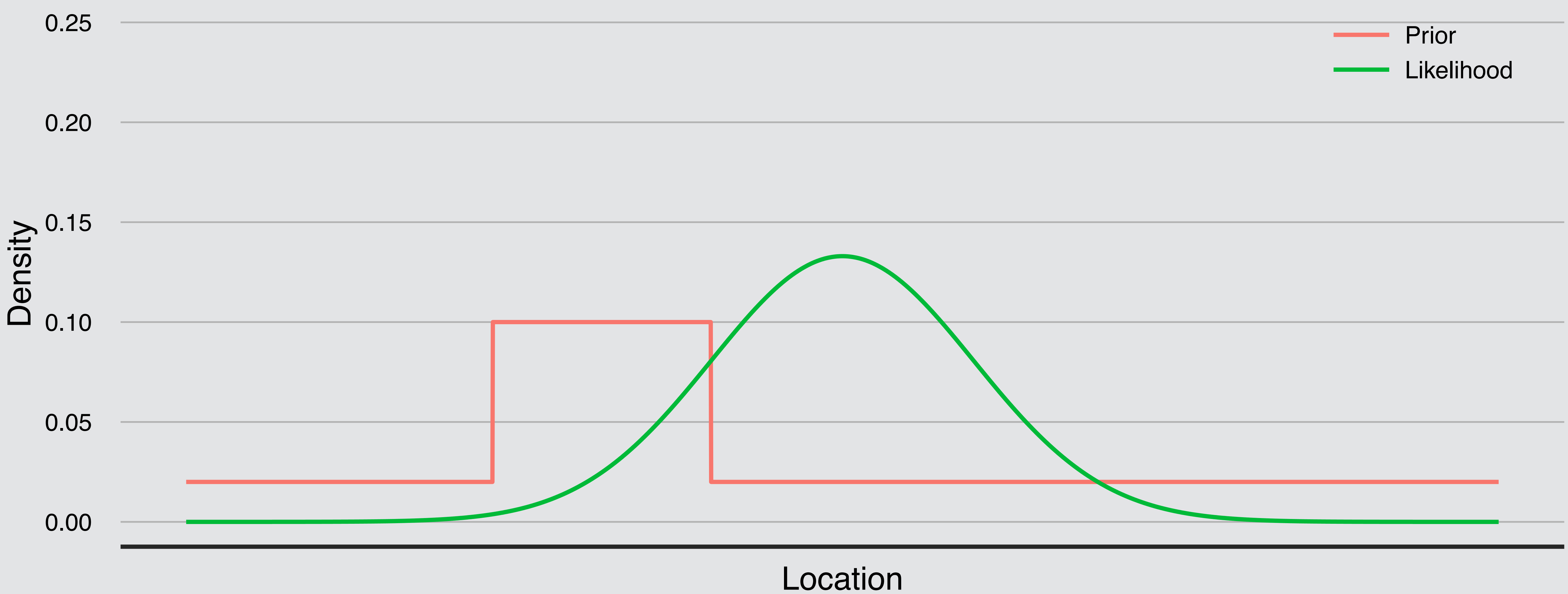
- GPS
- GPS + road snapping

Incorporate Domain Knowledge



Incorporate Domain Knowledge

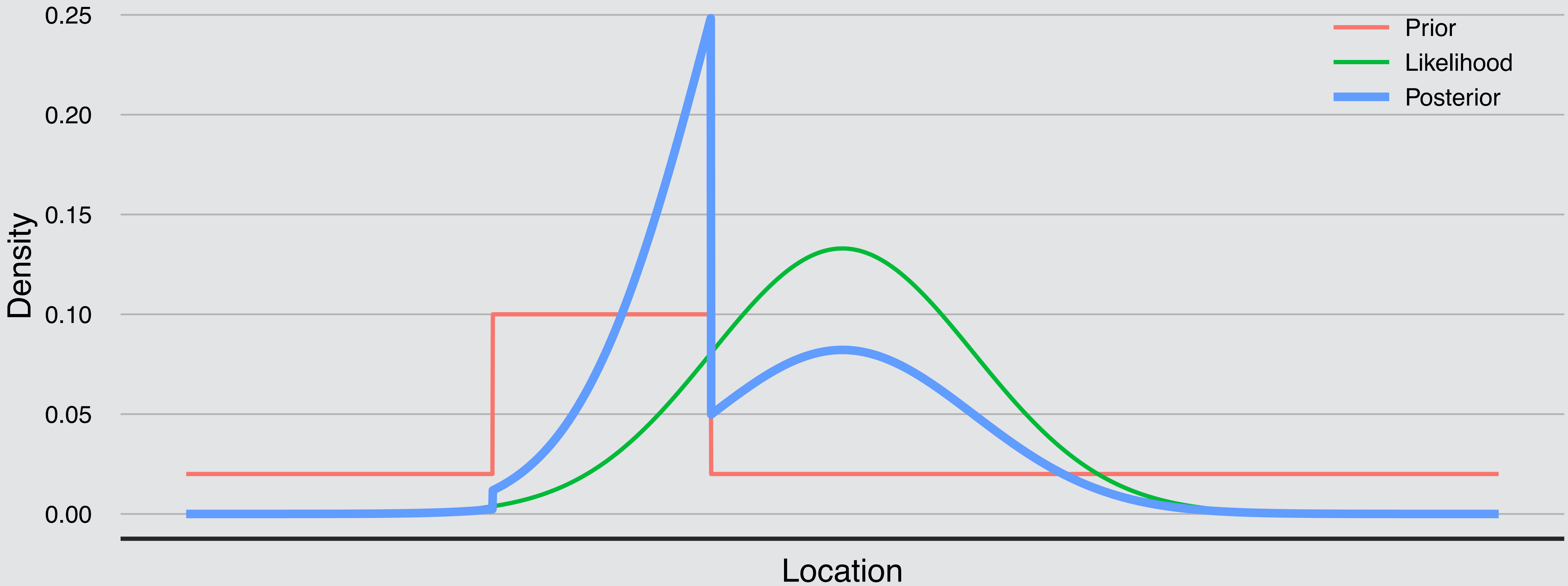
I am on a road



$$\underbrace{\Pr[H|E]}_{\text{posterior}} = \frac{\overbrace{\Pr[E|H]}^{\text{likelihood}}\,\overbrace{\Pr[H]}^{\text{prior}}}{\Pr[E]}$$

Incorporate Domain Knowledge

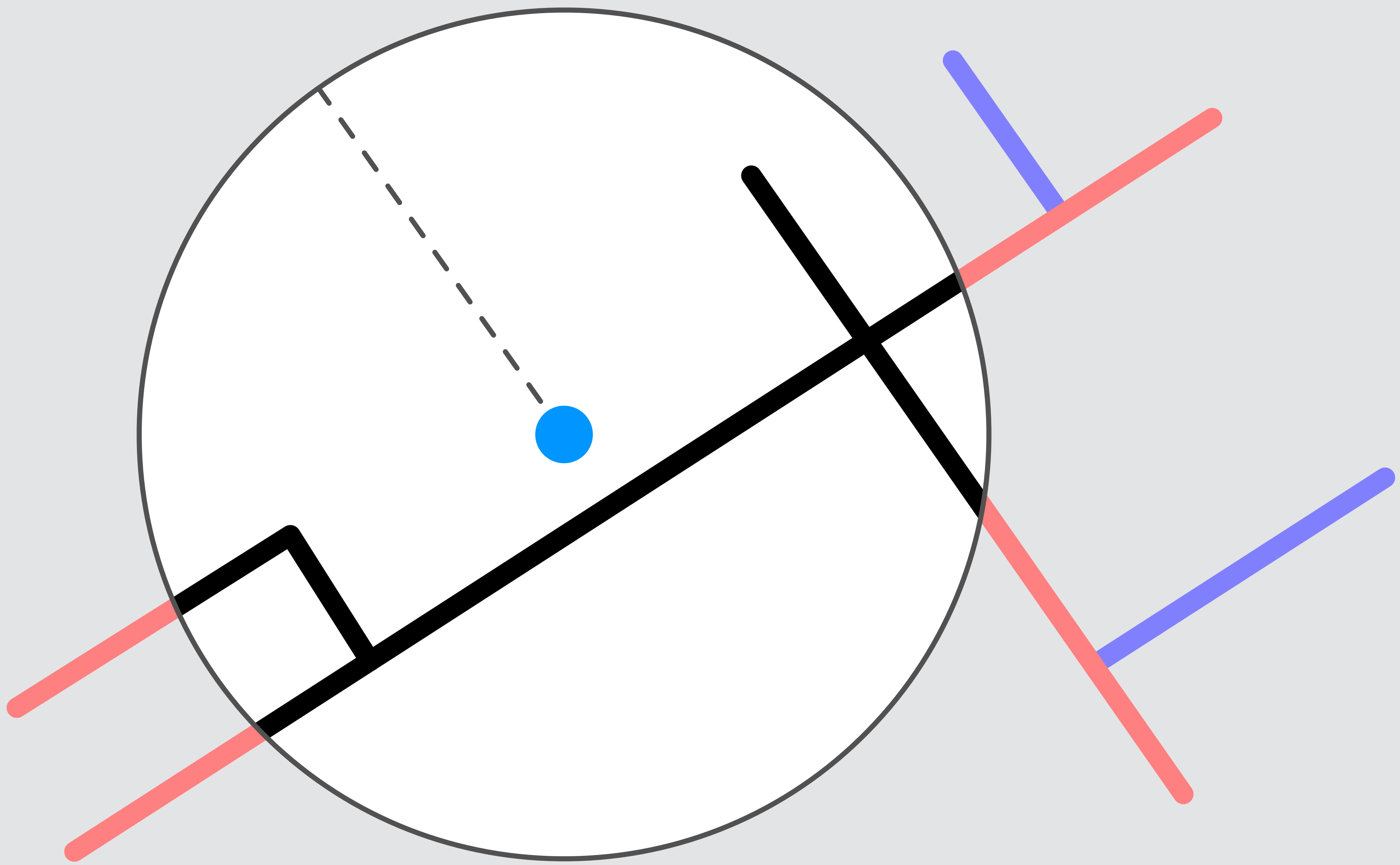
I am on a road



posterior, likelihood, prior

$$\Pr[H|E] = \frac{\Pr[E|H]\,\Pr[H]}{\Pr[E]}$$

Incorporate Domain Knowledge



Adding Context

New operators and semantics

>! Conditional distribution operator

Bayes operator

Implementation

Sequential likelihood reweighting (new)

Automatically picks sample size!

Forward inference for imperative programming languages!

Road Snapping Adding Context

```
// find relevant roads
Uncertain<Point> roadPrior = new uncertain<Point>(()=>
SamplePrior(location, accuracy, radiusFactor, weight))

// improve location estimate
Uncertain<Point> NewLocation = GPSLikelihood #
roadPrior
```

Road Snapping Sampling

```
Point SamplePrior (Point location, double accuracy,
                  double radiusFactor, double weight) {
   // find relevant roads
   Double radius = radiusFactor * accuracy;
   Road[] segments = NearbySegments(roads, location,
radius)
   // Generate random sample according to weight
   If (Random.NextDouble() < 1 – 1/(1+wieght))
      return SamplePoint(segments)
   else return SampleUniform(location, accuracy);
}
Point SamplePoint(Road{} segments) {
   Road segment = WeightedSample(segments, (s) =>
s.length)
   Return SampleUniform(segment);
```

How should programmers reason about probabilistic programs?

```
assert file != NULL
```

test

verify

check

`assert` *e*

e must hold on **every** execution

Approximate Computing

The approximate image is close to the precise version

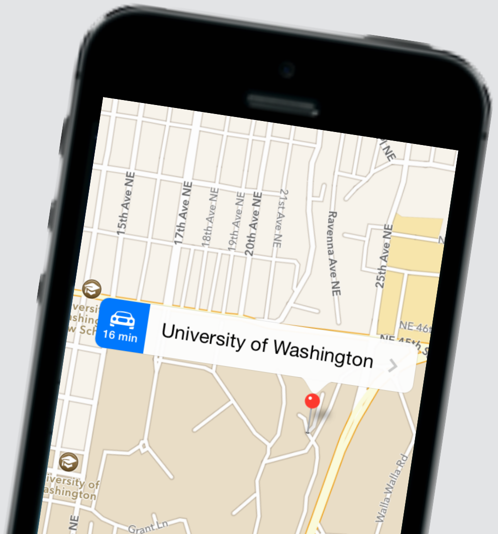
k-means clustering is likely to converge on unreliable hardware

```
assert e
```

e must hold on **every** execution

Obfuscation for Privacy

obfuscated data is still useful in aggregate



Mobile and Sensing

mostly on the road

Approximate Computing

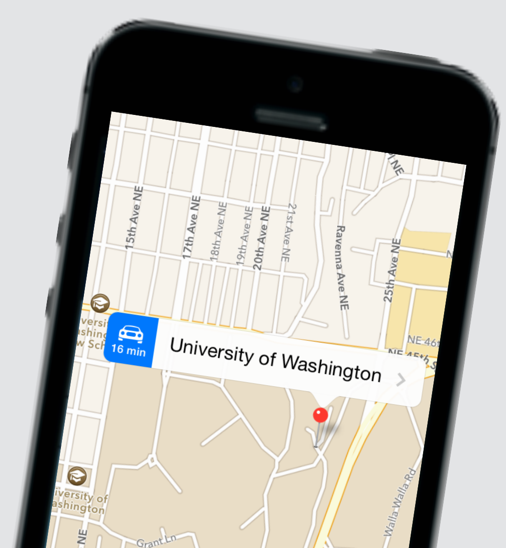
The approximate image is close to the precise version

k-means clustering is likely to converge on unreliable hardware

Traditional assertions are insufficient for programs with probabilistic behavior

Obfuscation for Privacy

obfuscated data is still useful in aggregate



Mobile and Sensing

mostly on the road

Assertions are insufficient for data obfuscation

```
true_avg = average(salaries)
private_avg =
  average(obfuscate(salaries))
assert true_avg - private_avg
       <= 10,000
```

Assertions are insufficient for data obfuscation

```
true_avg = average(salaries)
private_avg =
  average(obfuscate(salaries))
assert true_avg - private_avg
        <= 10,000
```

probability distribution

Assertions

`assert` *e*

Probabilistic assertion

`passert` *e*, *p*, *c*

Probabilistic assertion

`passert` e, p, c

e must hold with probability p
at confidence c

Probabilistic assertion

`passert` *e*, *p*, *c*

test?

verify?

check?

How to verify a probabilistic assertion

probabilistic program

```
float obfuscated(float n) {
  return n + gaussian(0.0, 1000.0);
}
float average_salary(float* salaries) {
  total = 0.0;
  for (int i = 0; i < COUNT; ++i)
    total += obfuscated(salaries[i]);
  avg = total / len(salaries);
  p_avg = ...;
  passert e, p, c
}
```

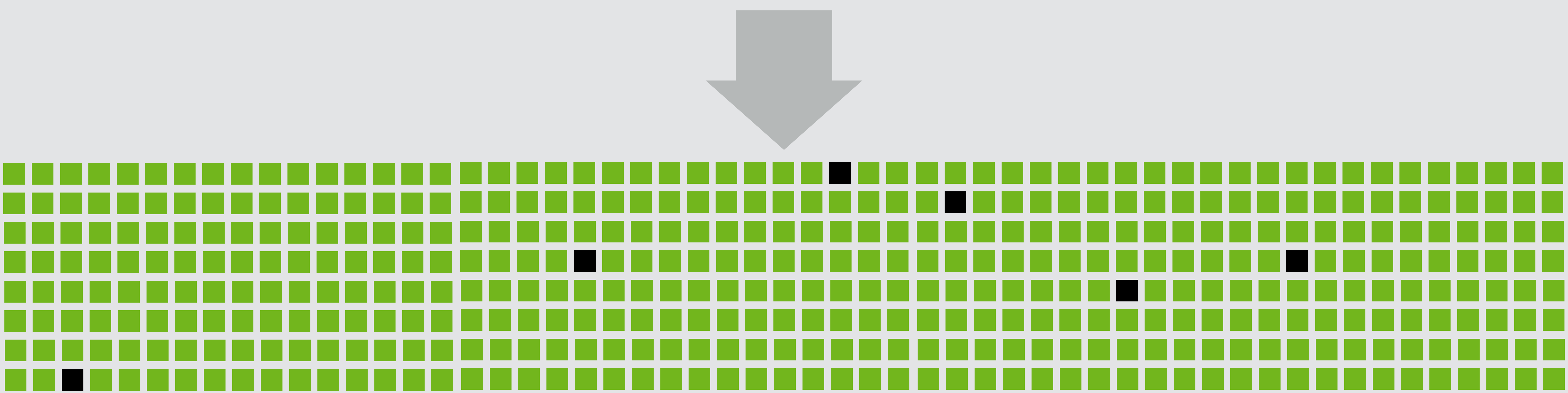
?

How to verify a probabilistic assertion naively

probabilistic program

```
float obfuscated(float n) {
  return n + gaussian(0.0, 1000.0);
}
float average_salary(float* salaries) {
  total = 0.0;
  for (int i = 0; i < COUNT; ++i)
    total += obfuscated(salaries[i]);
  avg = total / len(salaries);
  p_avg = ...;
passert e, p, c
}
```

?



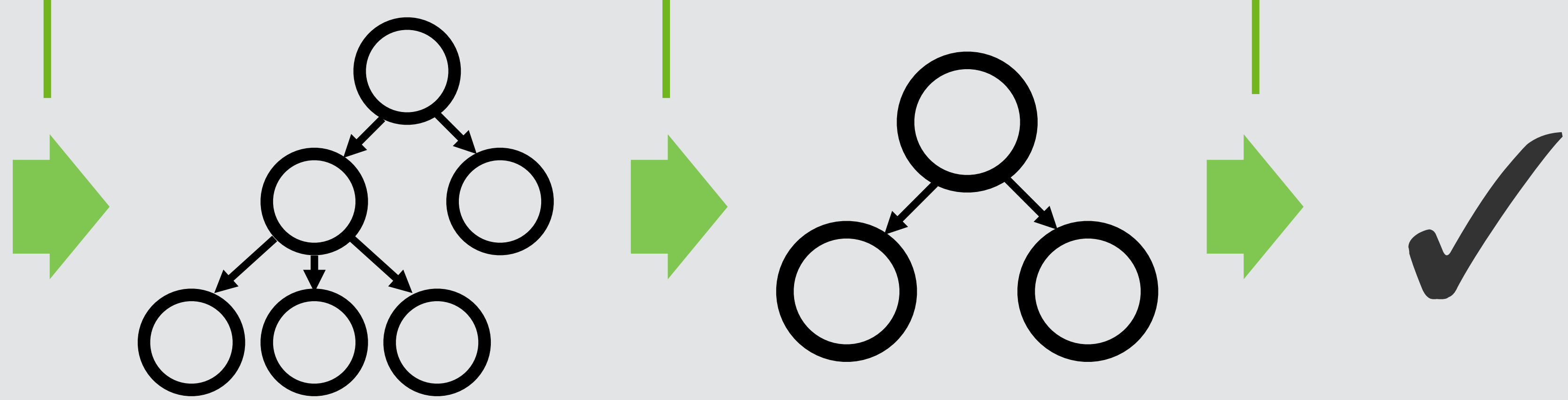
How to verify a probabilistic assertion efficiently

distribution extraction
via symbolic execution

statistical
optimizations

verification

```
float obfuscated(float n) {
  return n + gaussian(0.0, 1000.0);
}
float average_salary(float* salaries) {
  total = 0.0;
  for (int i = 0; i < COUNT; ++i)
    total += obfuscated(salaries[i]);
  avg = total / len(salaries);
  p_avg = ...;
passert e, p, c
}
```

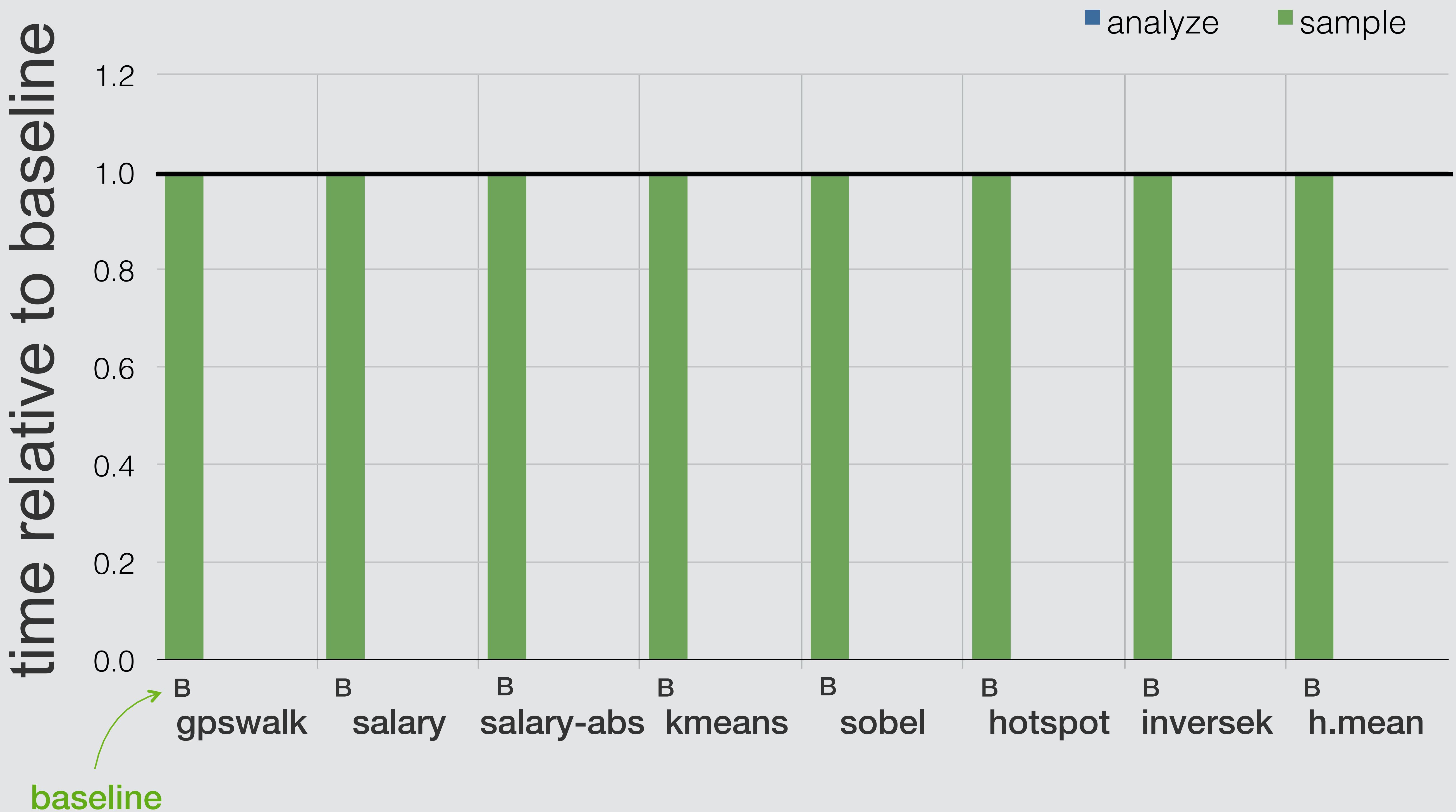


Bayesian network
IR

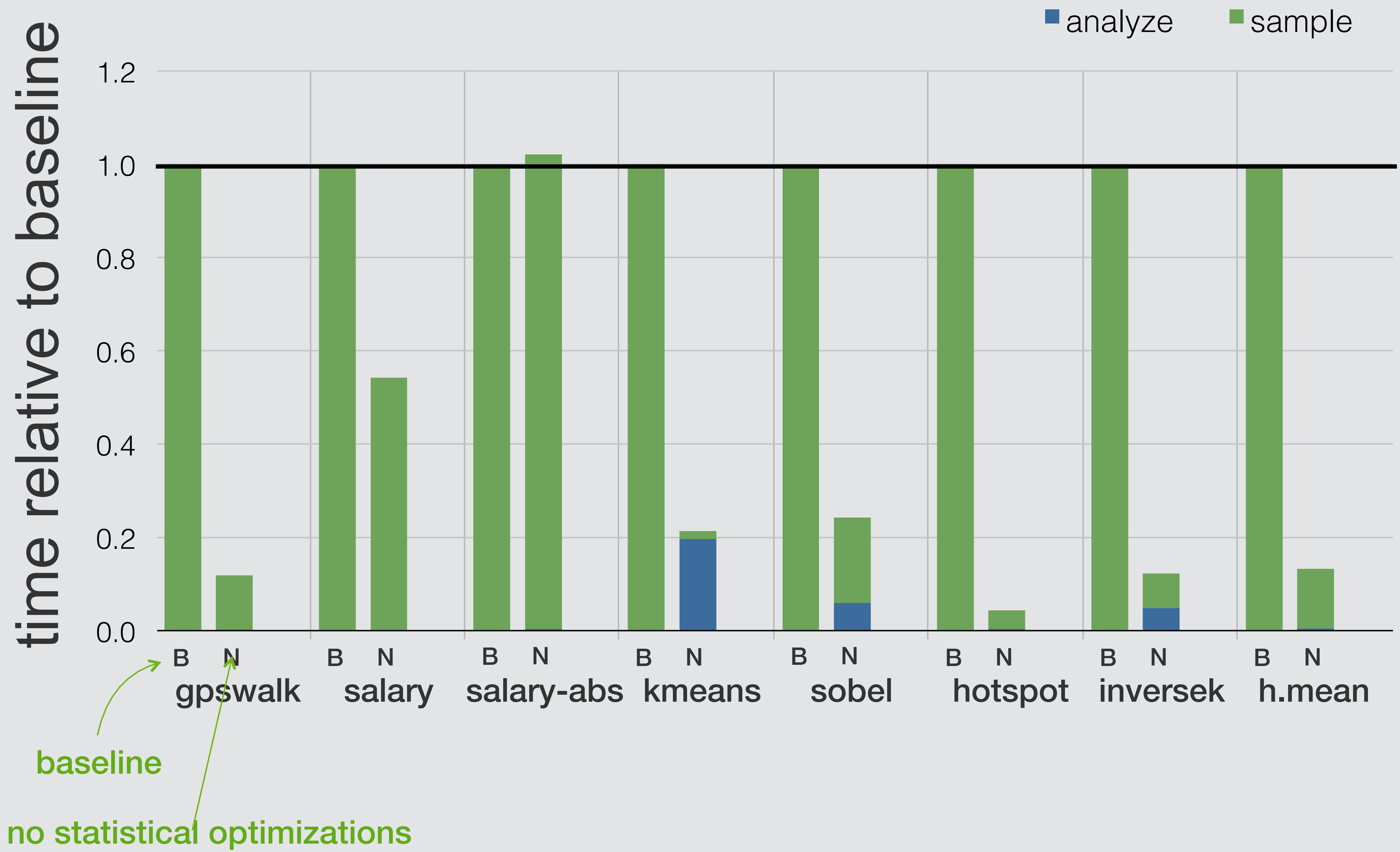
Probabilistic Assertion Verification Evaluation

sensing	**gpswalk**
privacy	**salary**
	salary-abs
approximate computing	**kmeans**
	sobel
	hotspot
	inversek2j

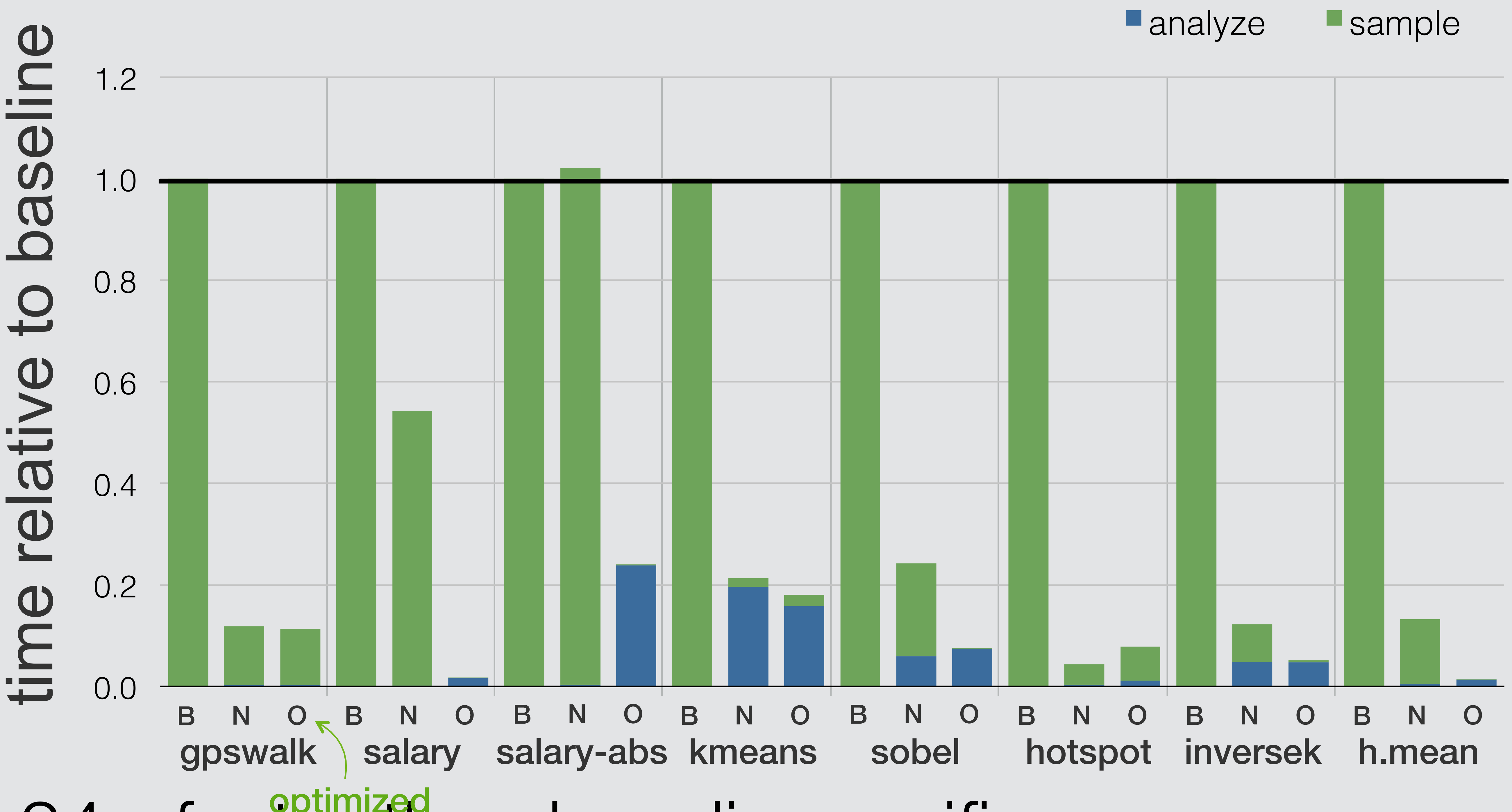
Time vs Stress Testing



Time vs Stress Testing



Time vs Stress Testing



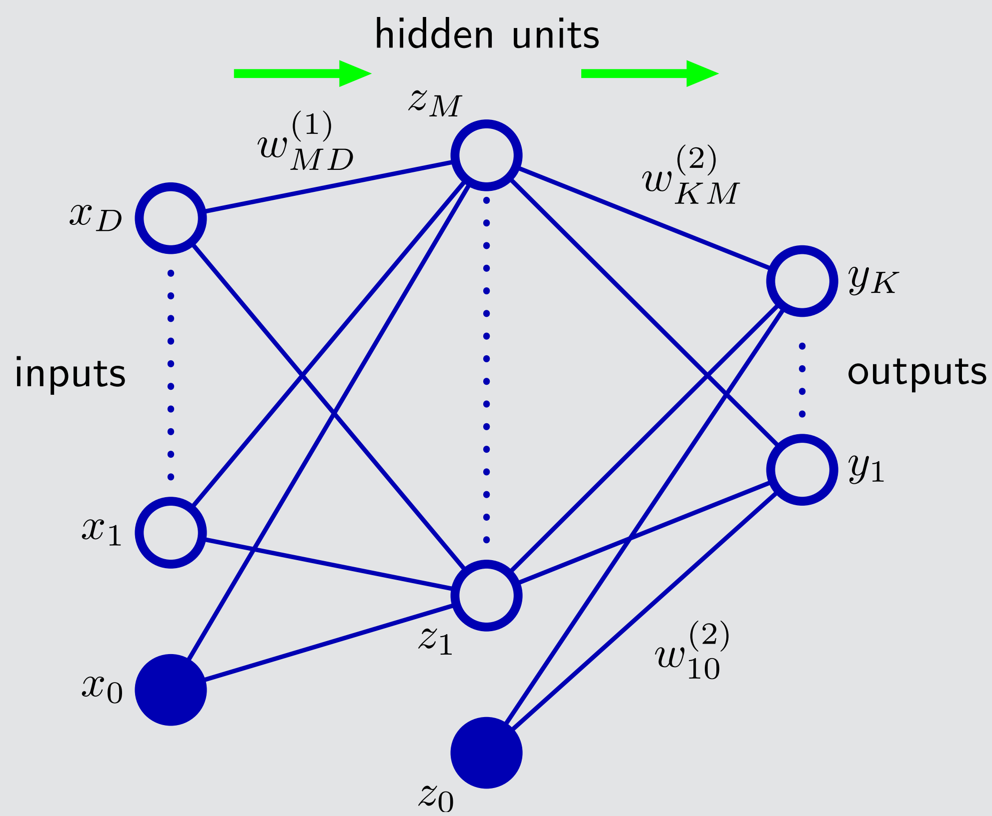
24× faster than baseline verifier on average

Mostly analysis time

Other Probabilistic Programming Languages

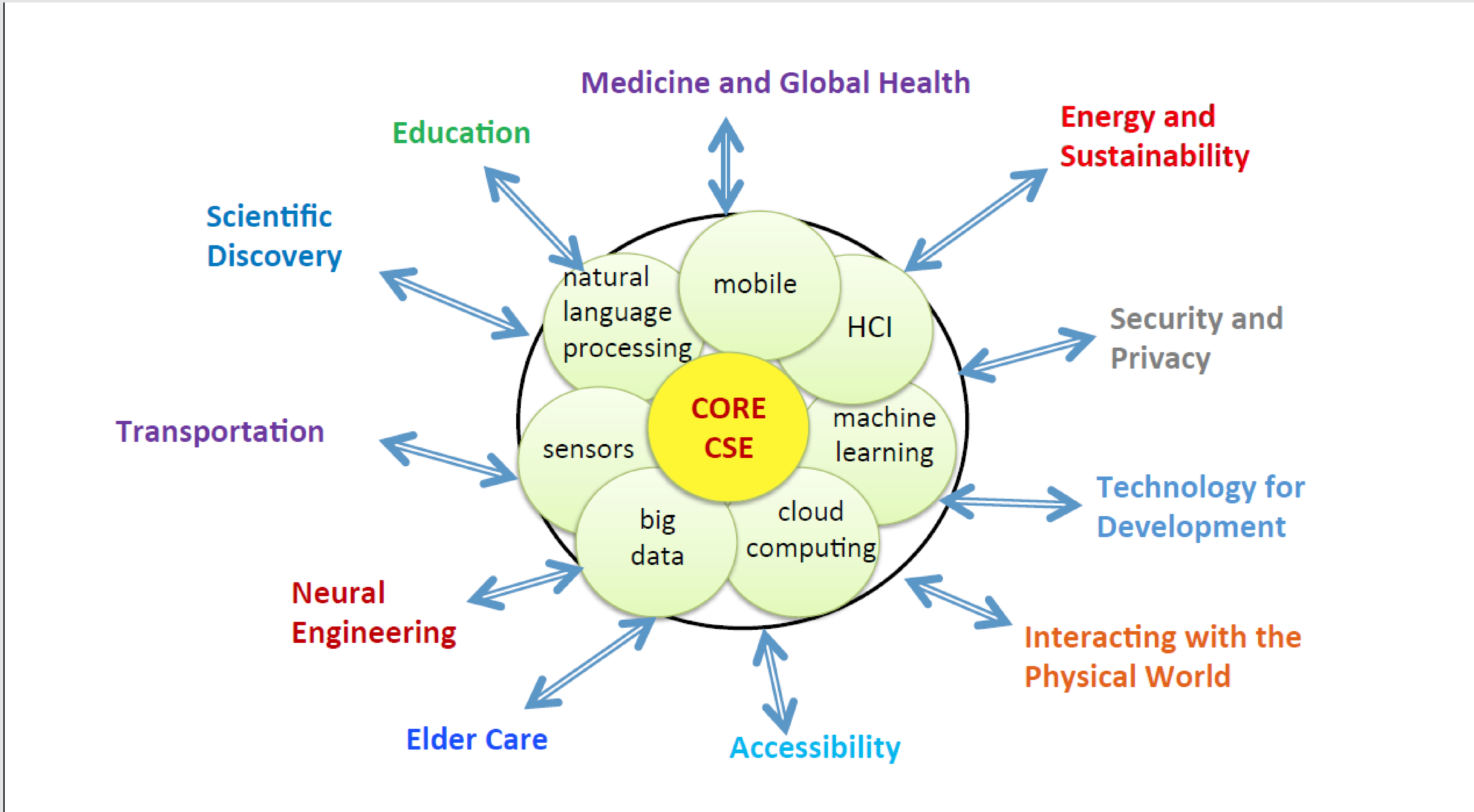
Probabilistic programming

BUGS, Church, Infer.NET, …



Uncertain<*T*> helps developers *without* statistics PhDs.

A Modern View of Computing



Ed Lazowska http://lazowska.cs.washington.edu/Wenk.pdf

Accuracy, Efficiency, & Programmer Productivity

The **Uncertain<T>** programming model, types, and operators help programmers reason about error in estimates and improve their accuracy.

Probabilistic Assertions express correctness properties of these programs. Our verifier accurately and efficiently checks them.

Collaborators

Todd Mytkowicz, Microsoft

James Bornholt, ANU & UW

Na Meng, The University of Texas at Austin

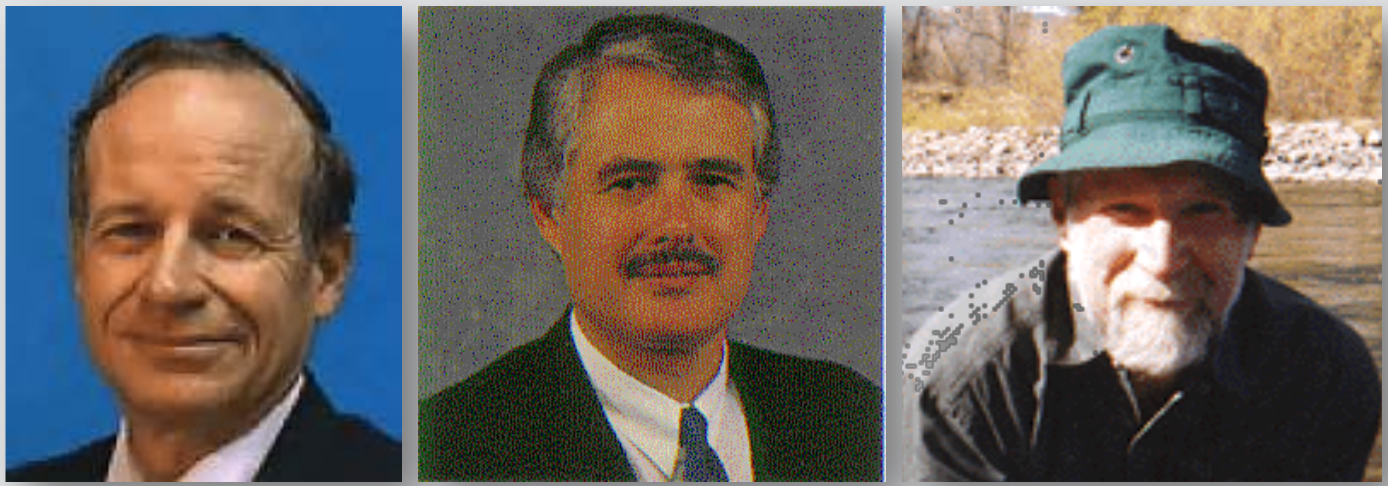
Adrian Samspon, The University of Washington, Seattle

Luis Ceze, The University of Washington, Seattle

Dan Grossman, The University of Washington, Seattle

A Byte of My Story

A Byte of My Story



Mentors

ACM Fellow

Family

Congressional Testimony

Success, Failure, and Learning

Rejected job applications

1984 (all), 1993 (8 of 11), 2011 (4 of 8)

Failed PhD qualifying exam

Rejected first three grant applications

Rejected 3 times my most cited paper

Rejected papers, grants, papers, …

learn & persist

Thank you!