# Expressing and Verifying **Probabilistic Assertions**


**Adrian Sampson**
Pavel Panchekha
| University of Washington

Todd Mytkowicz
Kathryn S. McKinley
| Microsoft Research

Dan Grossman
Luis Ceze
| University of Washington


PLDI 2014

**Probabilistic assertions** express correctness properties in modern software. Our verifier checks them **efficiently and accurately**.

**`assert`**` file != NULL`

test

verify

check

`assert` *e*

*e* must hold on **every** execution

≈

## Approximate Computing

*this approximate image is close to its precise version*

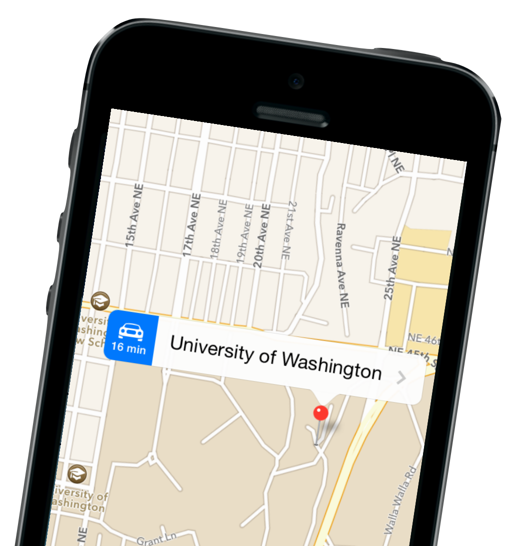
*k-means clustering is likely to converge even on unreliable hardware*

`assert` *e*

*e*

## Obfuscation for Data Privacy

*obfuscated data is still useful in aggregate*




## Mobile and Sensing

*sensor error does not render the app's conclusions useless*

## Approximate Computing

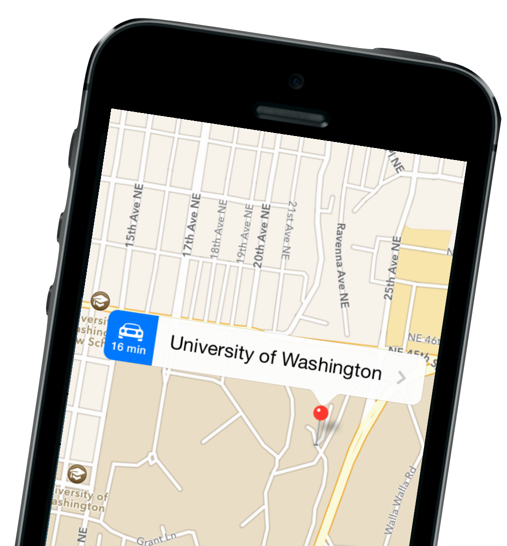
*this approximate image is close to its precise version*

*k-means clustering is likely to converge even on unreliable hardware*

Traditional assertions are **insufficient** for programs with **probabilistic behavior.**

## Obfuscation for Data Privacy

*obfuscated data is still useful in aggregate*




## Mobile and Sensing

*sensor error does not render the app's conclusions useless*

# Assertions are insufficient for private-data obfuscation

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

# Assertions are insufficient for private-data obfuscation

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

probability distribution

# Assertion

**`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
}
```

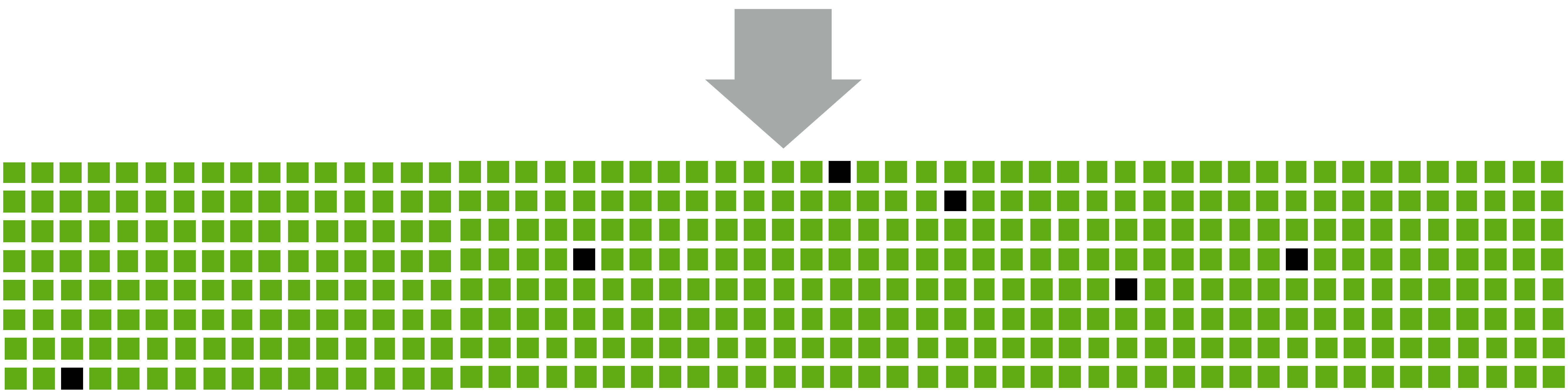
?

# 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 with statistical reasoning

queries & inference → `passert`

for statistical models → for probabilistic software

Church

Infer.NET

[Sankaranarayanan+ PLDI 2013]

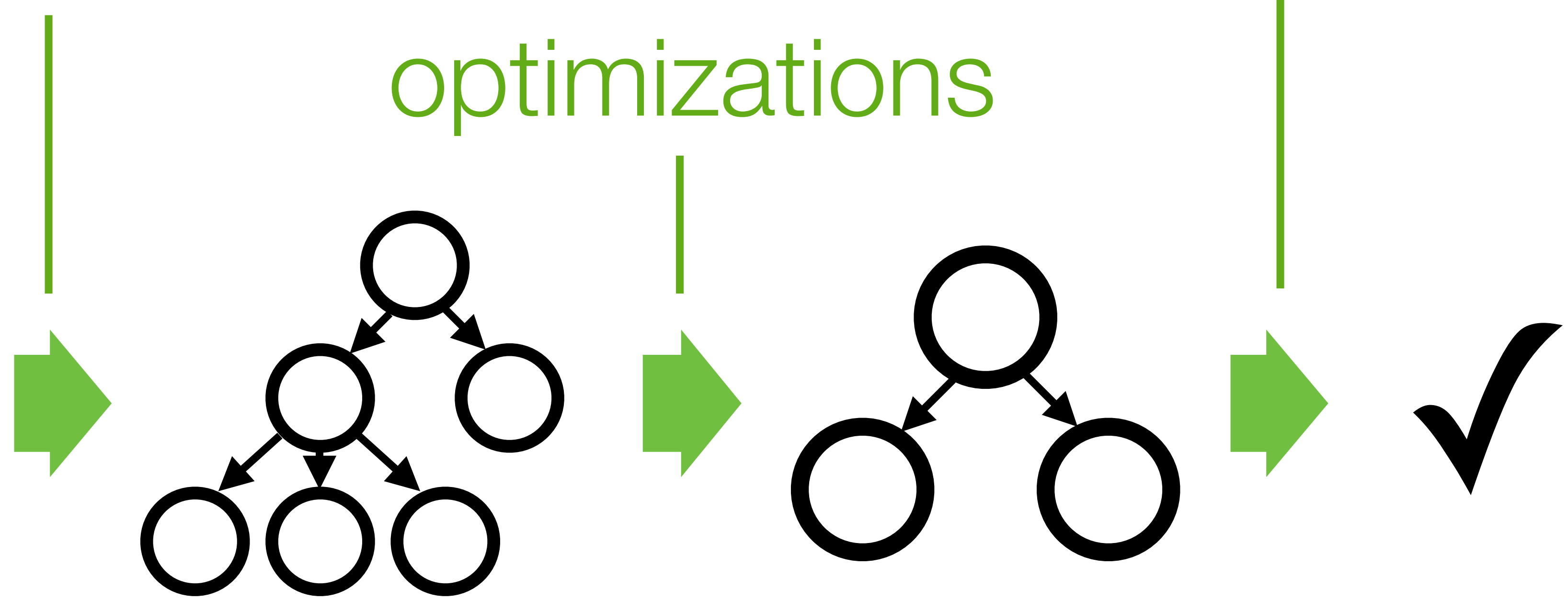
[Hur+ PLDI 2014]

⋮

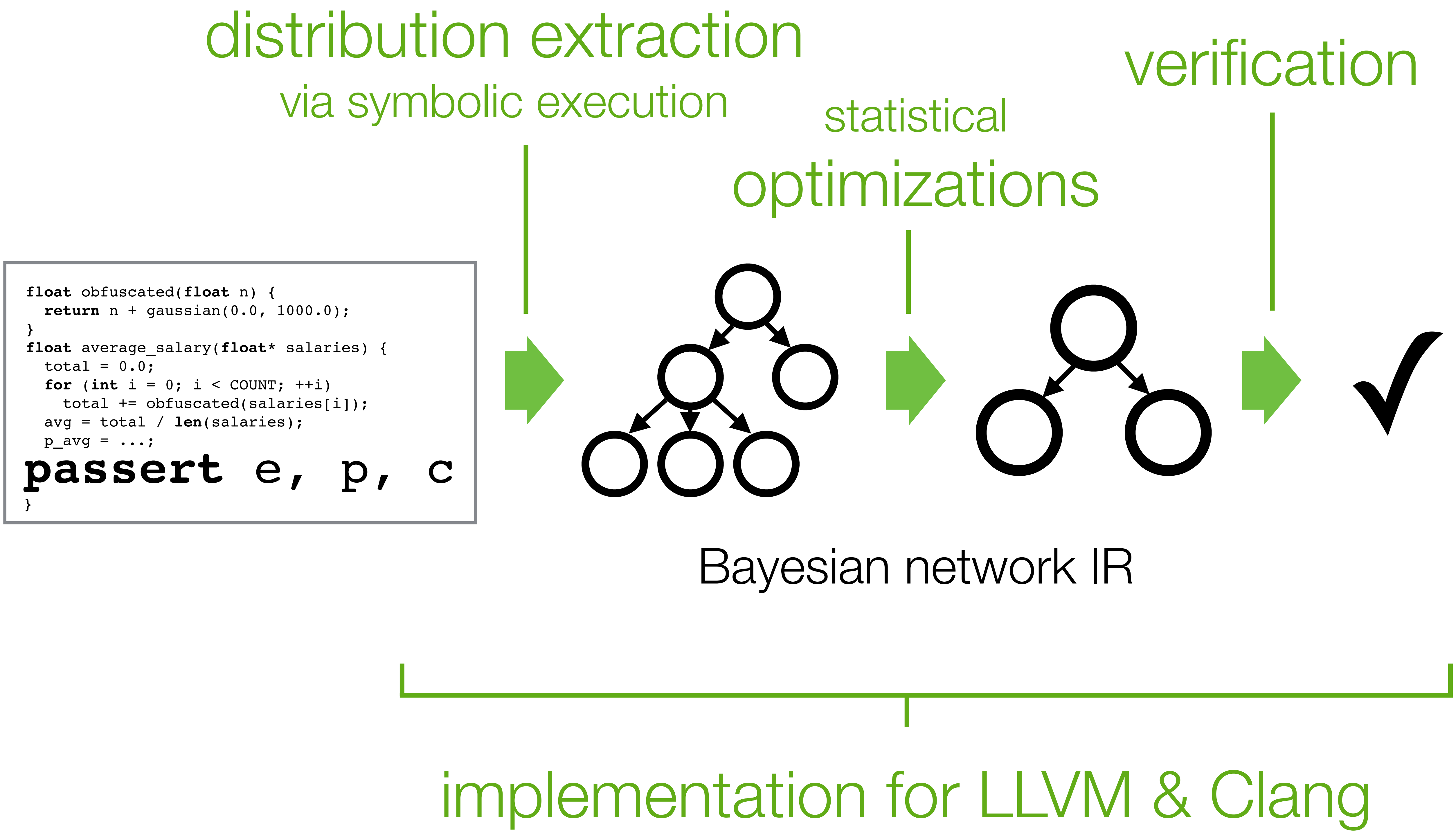
?

# How to verify a probabilistic assertion efficiently and accurately

```
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 and accurately

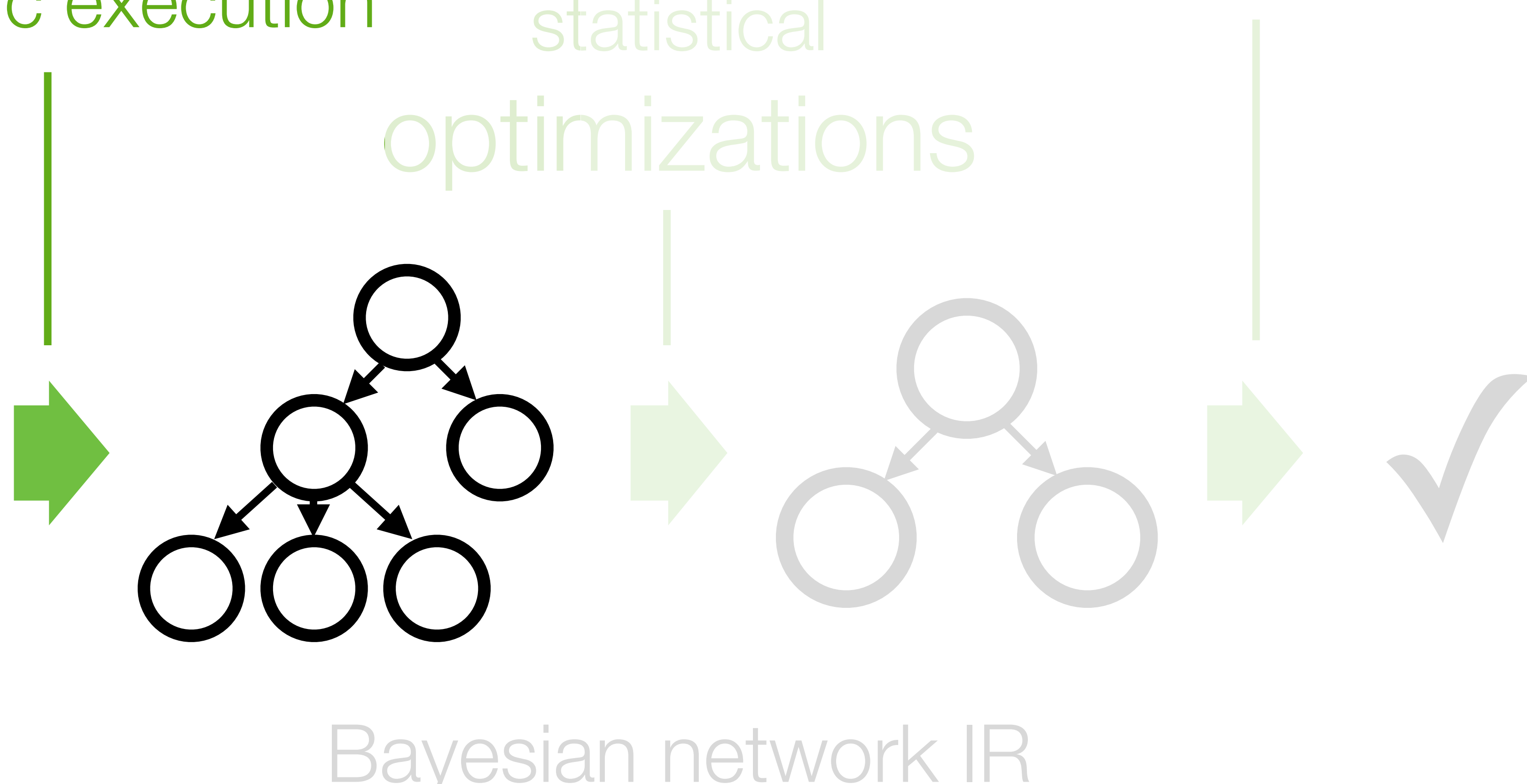


# How to verify a probabilistic assertion efficiently and accurately

distribution extraction

via symbolic execution

```
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
}
```




implementation for LLVM & Clang

# Distribution extraction: random draws are symbolic

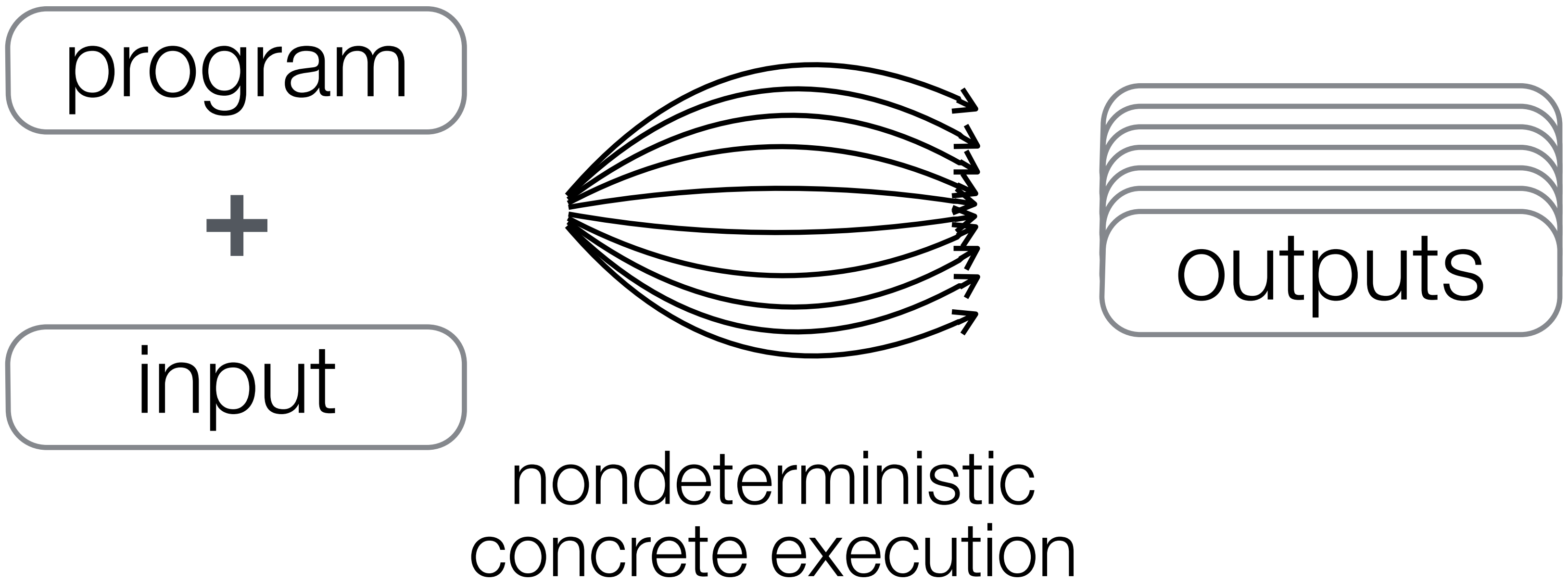
symbolic heap

| | |
|---|---|
| `a` | 4.2 |

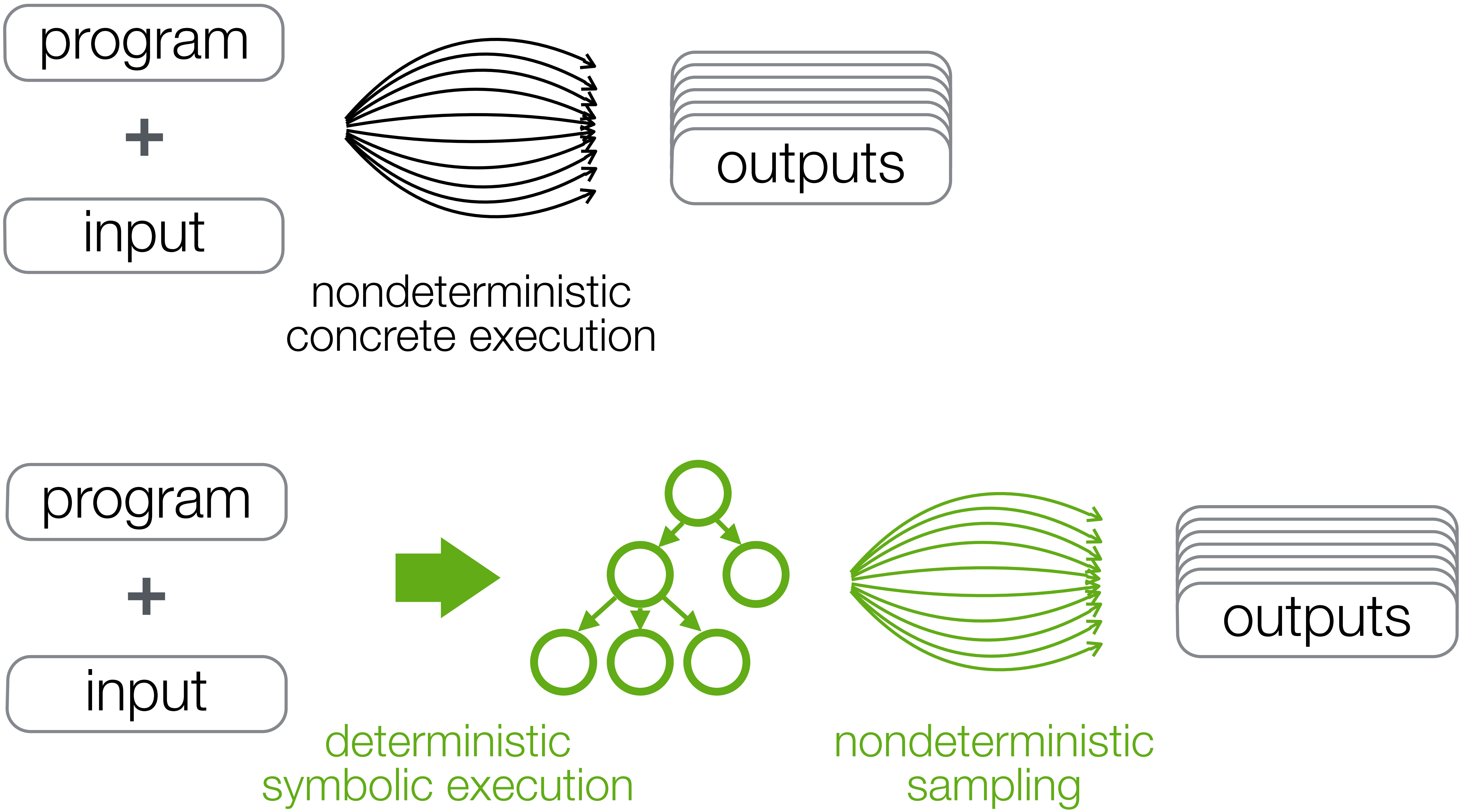
```
b = a + gaussian(0.0, 1.0)
```

| | |
|---|---|
| `a` | 4.2 |
| `b` | $4.2 + G_{0,1}$ |

# Concrete vs. symbolic semantics

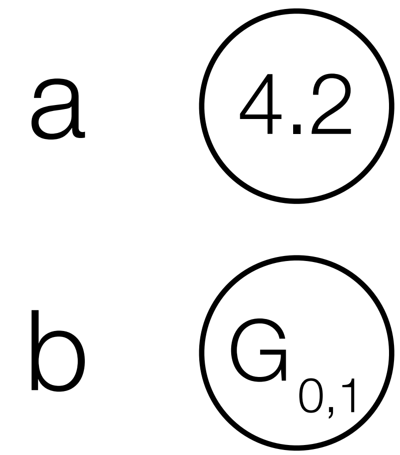


# Concrete vs. symbolic semantics



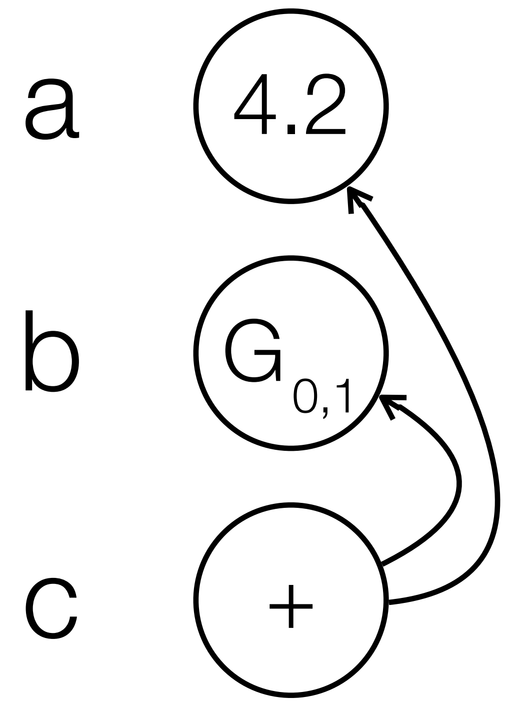
*input:* `a = 4.2`

`b = gaussian(0.0, 1.0)`



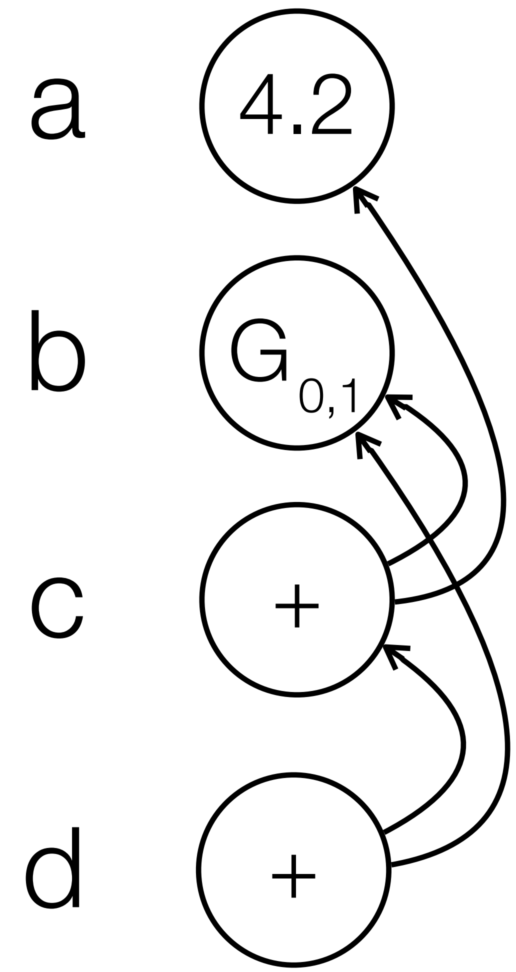
*input:* `a = 4.2`

```
b = gaussian(0.0, 1.0)
c = a + b
```

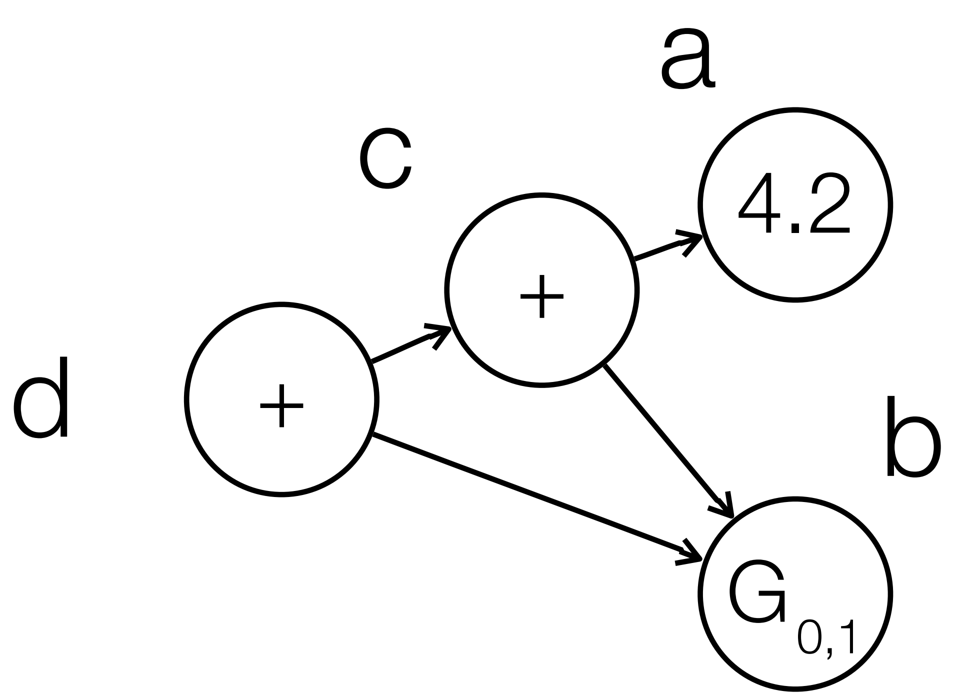


*input:* `a = 4.2`

```
b = gaussian(0.0, 1.0)
c = a + b
d = c + b
```

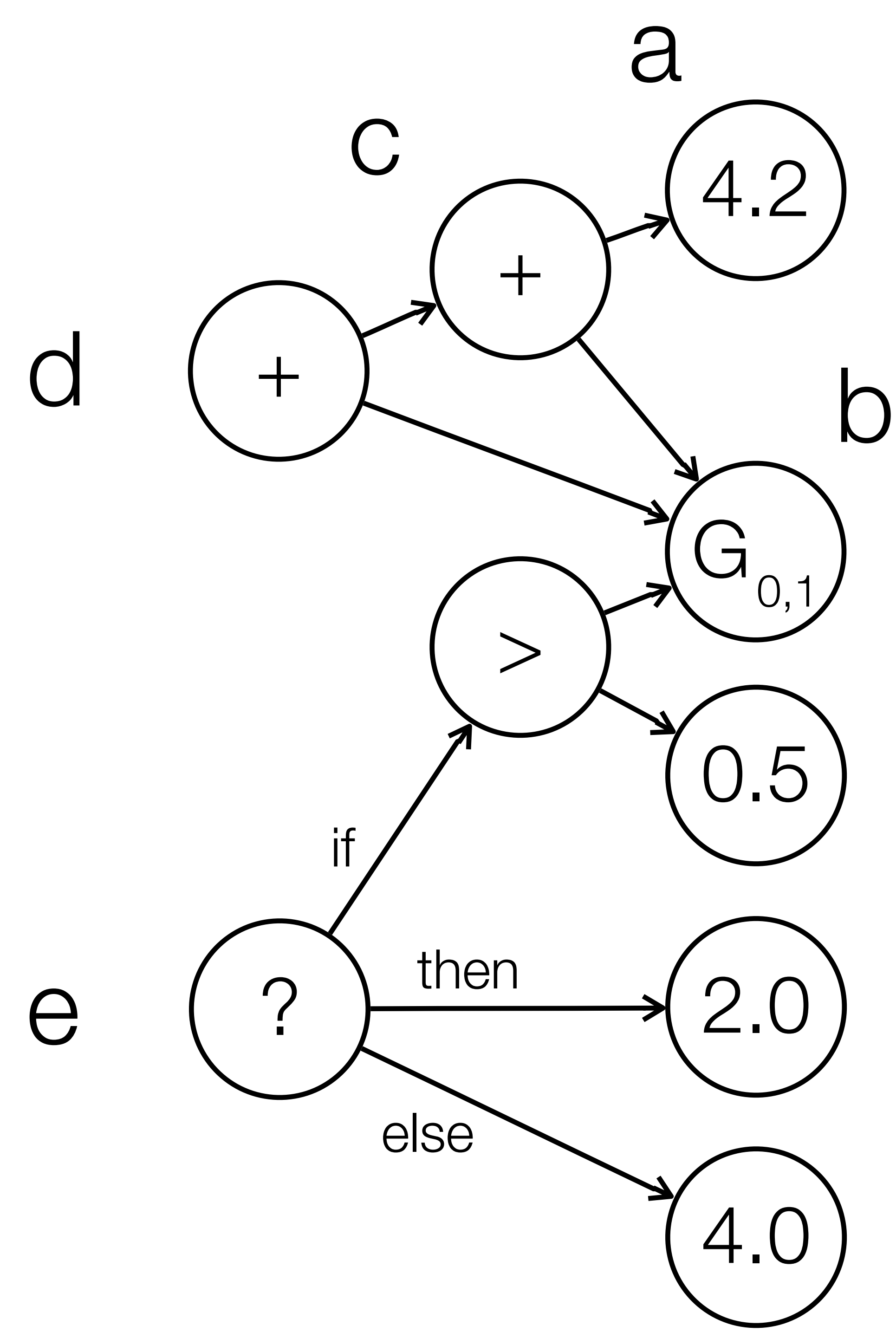


```
input: a = 4.2
b = gaussian(0.0, 1.0)
c = a + b
d = c + b
```

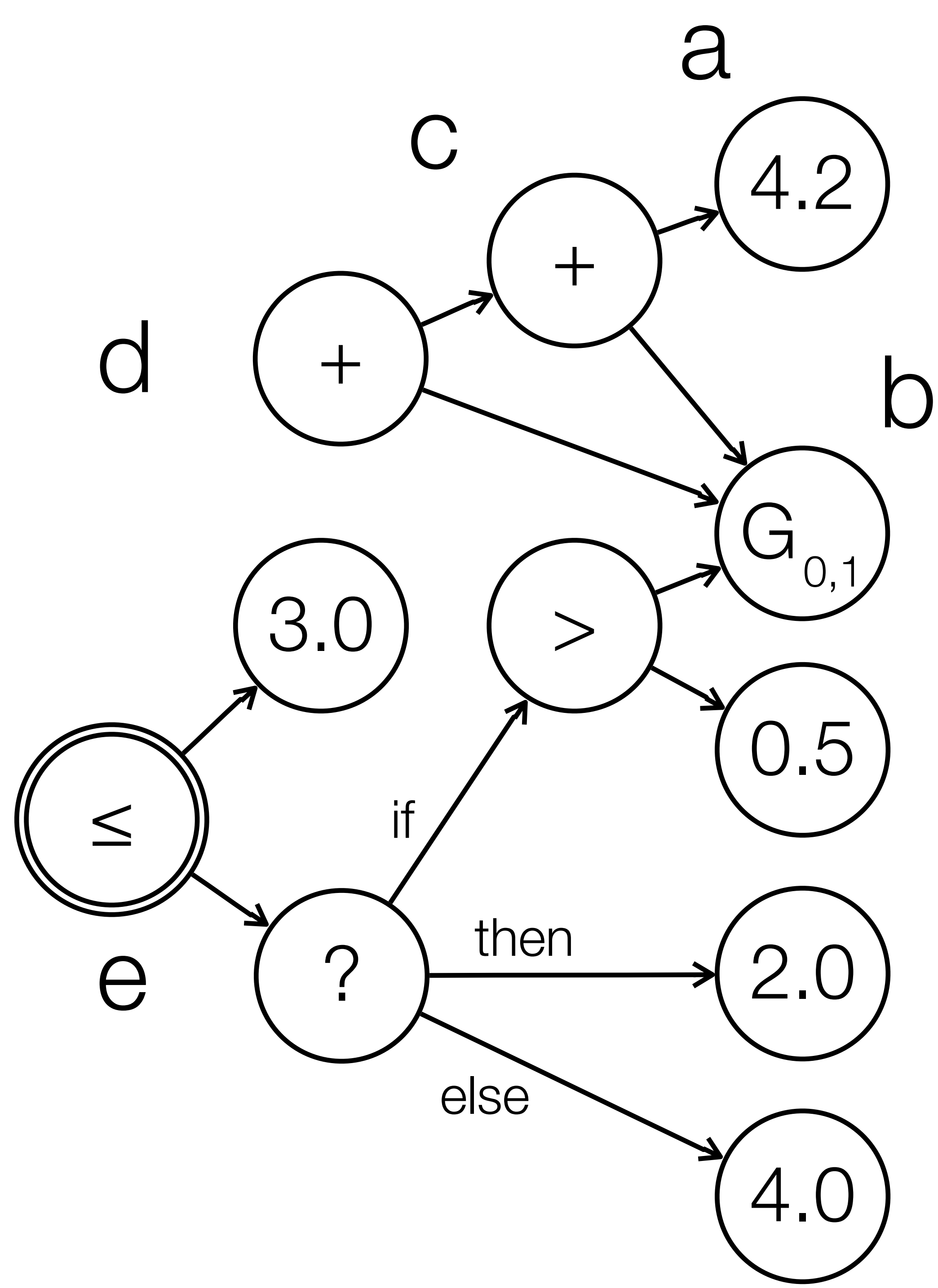


*input:* `a = 4.2`

```
b = gaussian(0.0, 1.0)
c = a + b
d = c + b
if b > 0.5
  e = 2.0
else
  e = 4.0
```

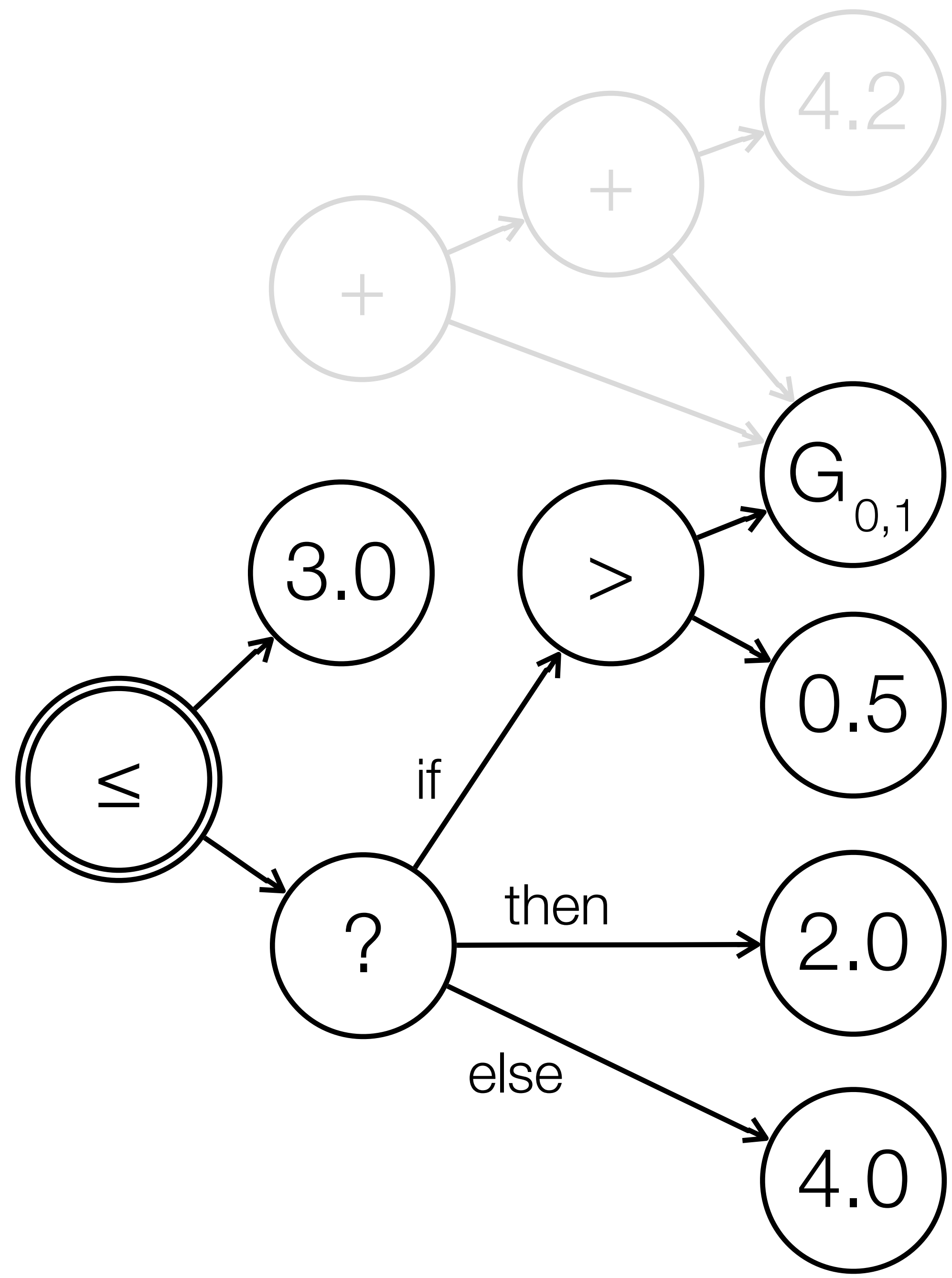


```
input: a = 4.2
b = gaussian(0.0, 1.0)
c = a + b
d = c + b
if b > 0.5
  e = 2.0
else
  e = 4.0
passert e <= 3.0,
          0.9, 0.9
```



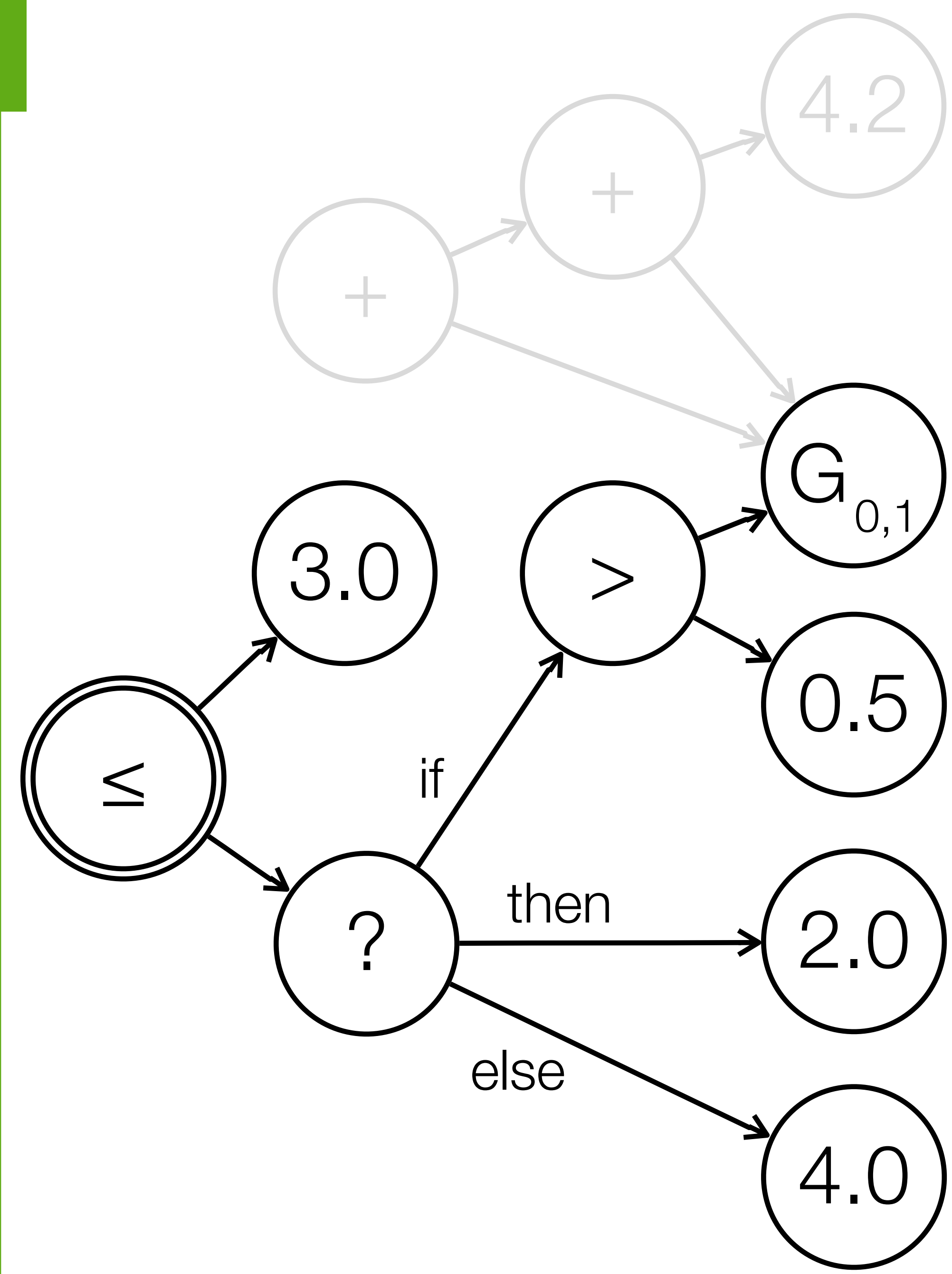
*input:* `a = 4.2`

```
b = gaussian(0.0, 1.0)
c = a + b
d = c + b
if b > 0.5
  e = 2.0
else
  e = 4.0
passert e <= 3.0,
        0.9, 0.9
```



*input:* `a = unif(2.0, 9.0)`

```
b = gaussian(0.0, 1.0)
c = a + b
d = c + b
if b > 0.5
  e = 2.0
else
  e = 4.0
passert e <= 3.0,
        0.9, 0.9
```



concrete input

`salary = $24,000`

**≈ testing**

input distribution

`salary = uniform(…)`

**≈ static analysis**

# More in the paper

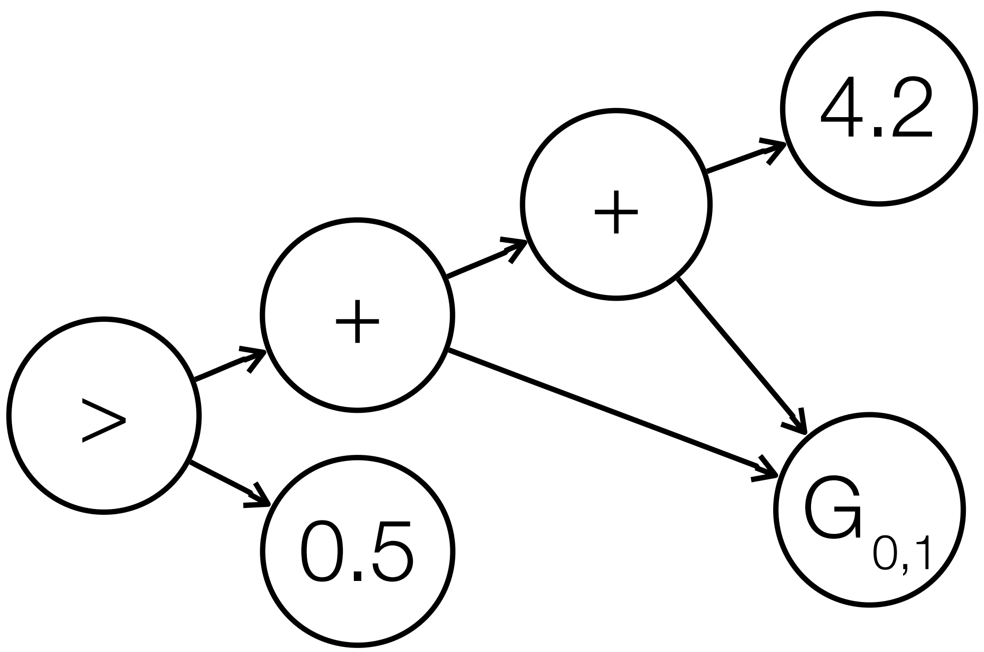
Arrays & pointers

Loops

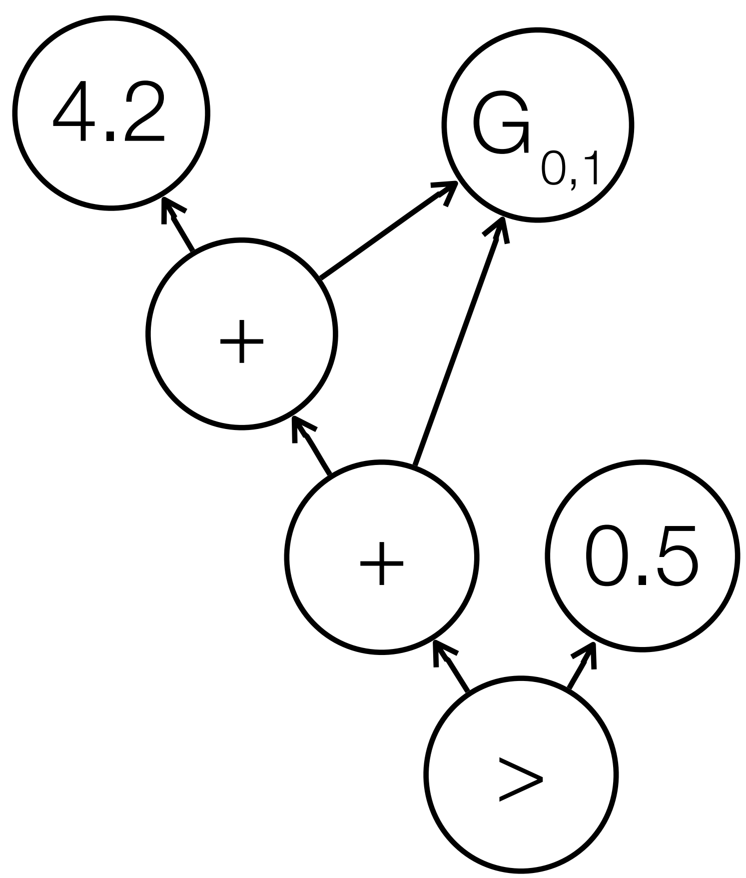
External code

Probabilistic path pruning

# Distribution extraction produces ~~an expression dag~~ Bayesian network



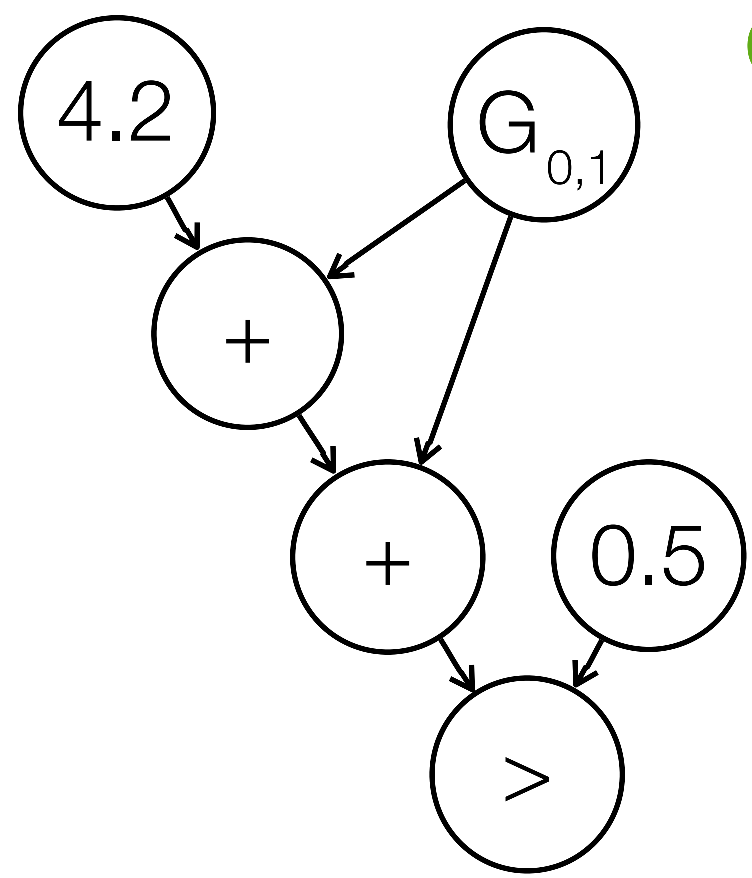
# Distribution extraction produces ~~an expression dag~~ Bayesian network



# Distribution extraction produces ~~an expression dag~~ Bayesian network

nodes: random variables

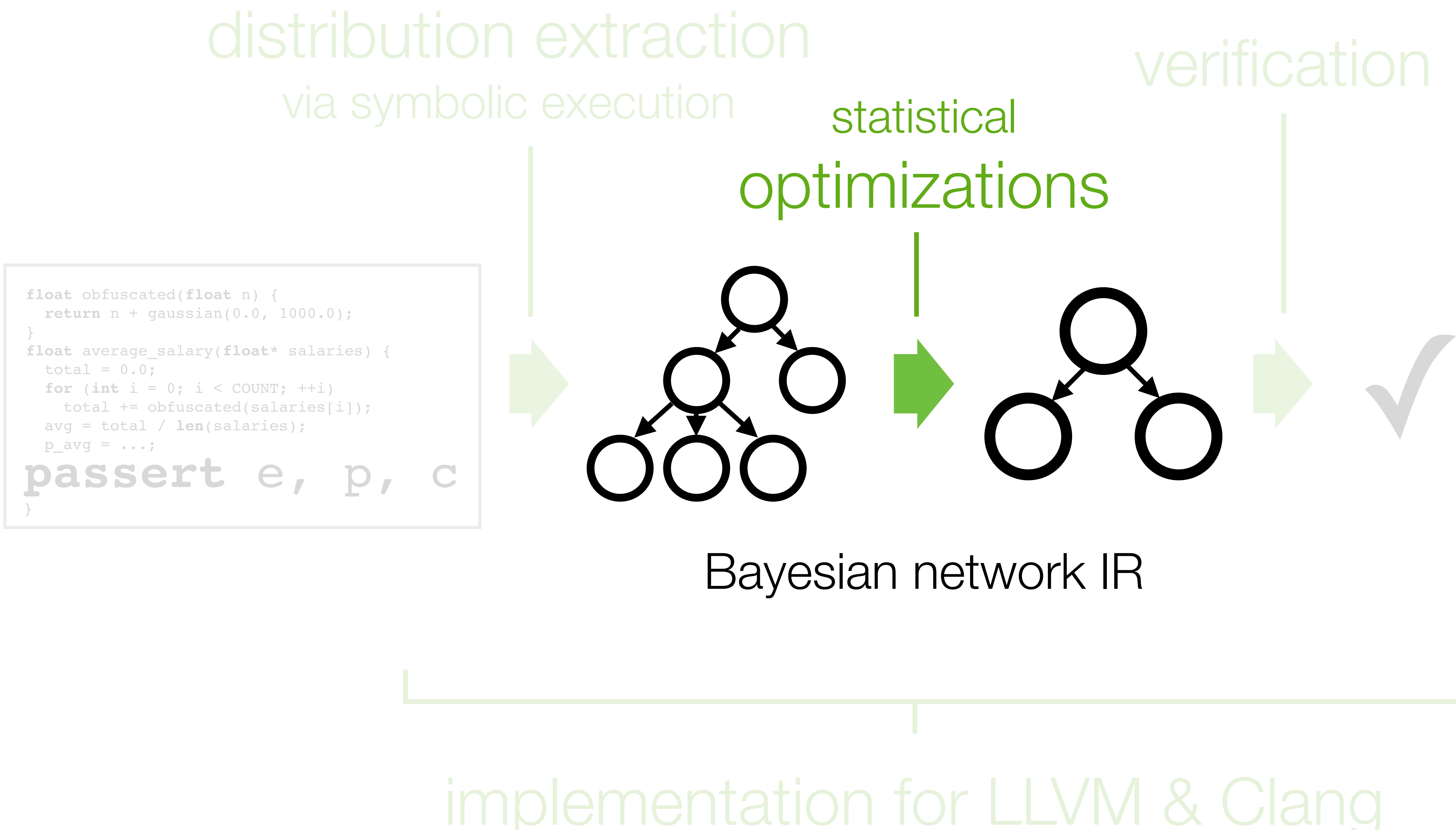
edges: dependence




directed & acyclic

random draws only at leaves

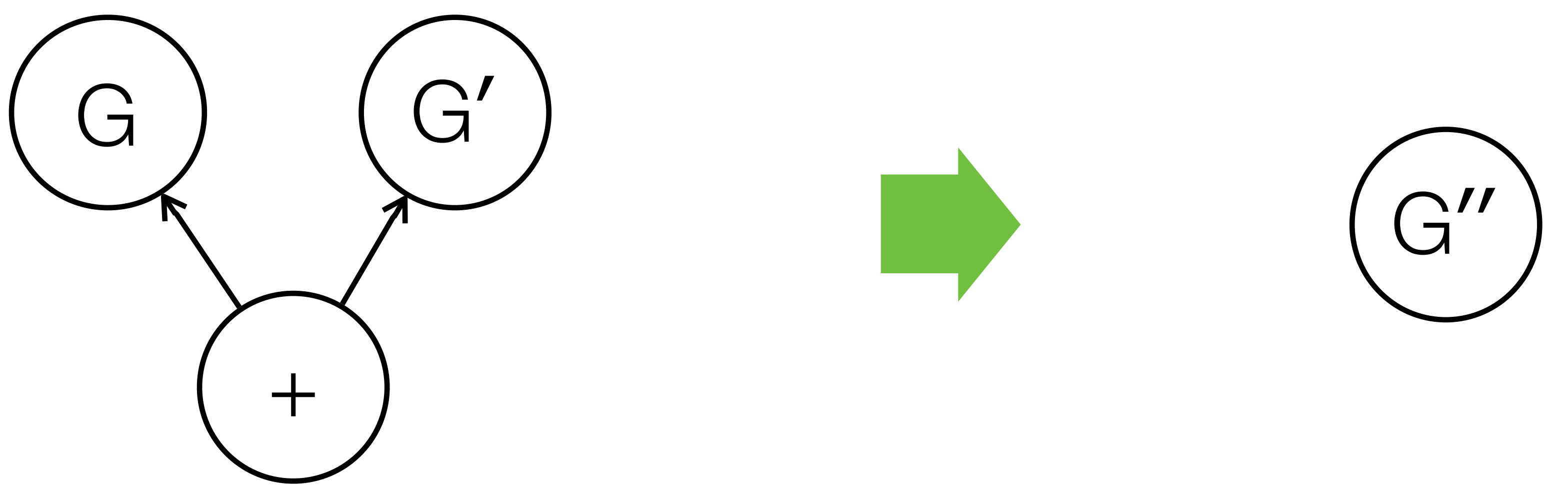
sample in a single pass


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
implementation for LLVM & Clang

statistical
property

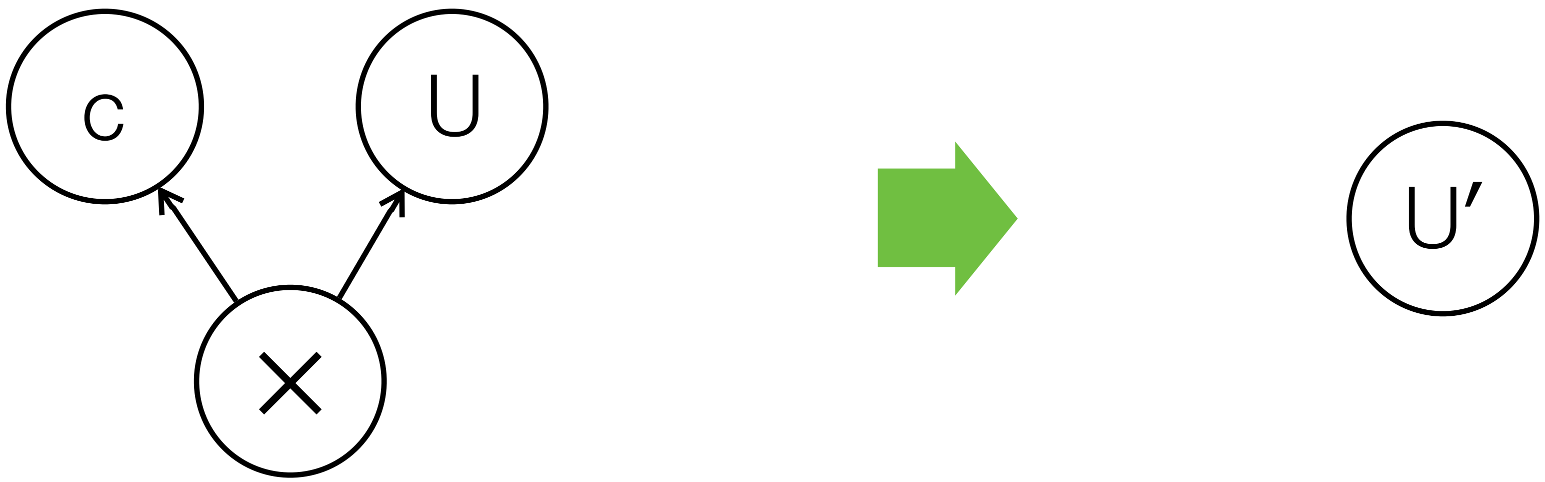
passert verifier
optimization

# Bayesian-network IR enables new optimizations



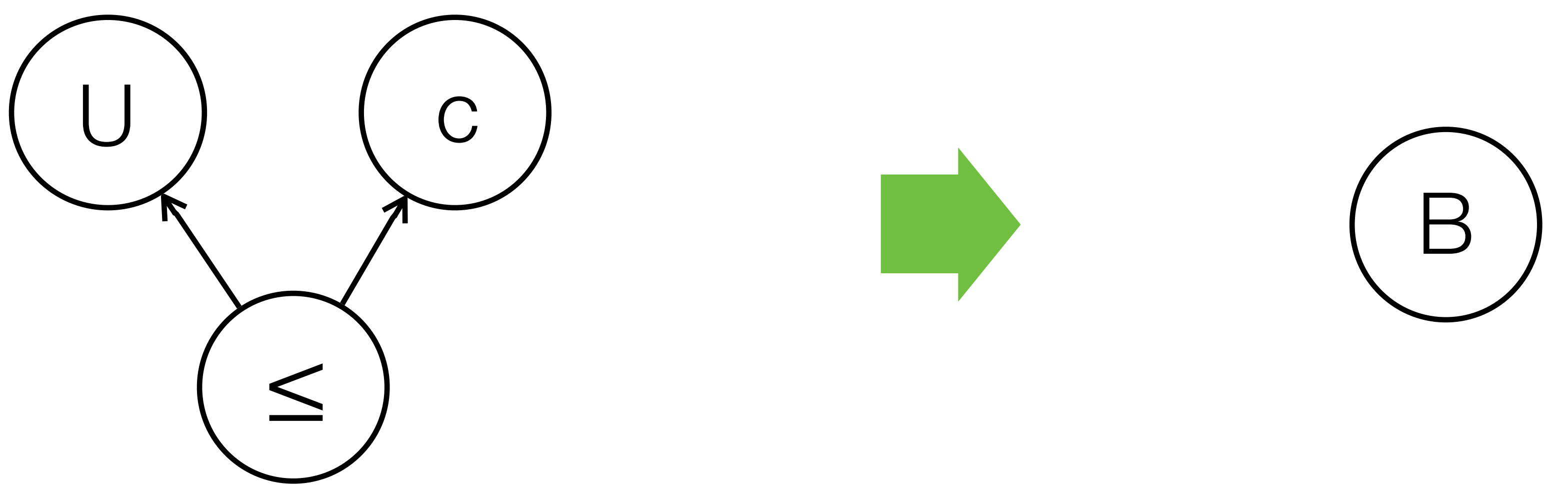

$$
\begin{aligned}
X &\sim G(\mu_X, \sigma_X^2) \\
Y &\sim G(\mu_Y, \sigma_Y^2) \\
Z &= X + Y \\
\Rightarrow Z &\sim G(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2)
\end{aligned}
$$

# Bayesian-network IR enables new optimizations




$$X \sim U(a, b)$$
$$Y = cX$$
$$\Rightarrow Y \sim U(ca, cb)$$

# Bayesian-network IR enables new optimizations



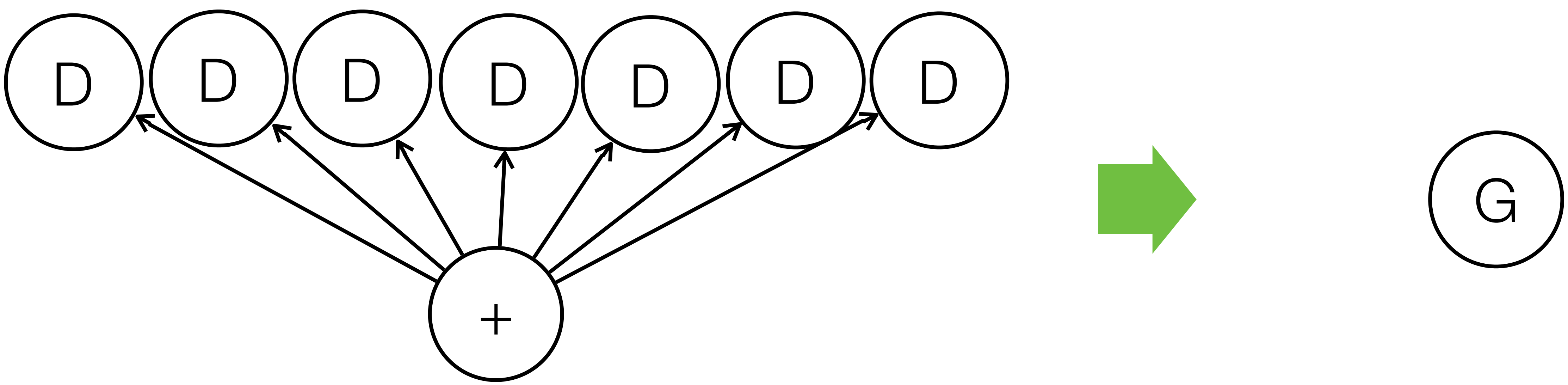

$$X \sim U(a, b)$$

$$Y \sim X \leq c$$

$$a \leq c \leq b$$

$$\Rightarrow Y \sim B\left(\frac{c-a}{b-a}\right)$$

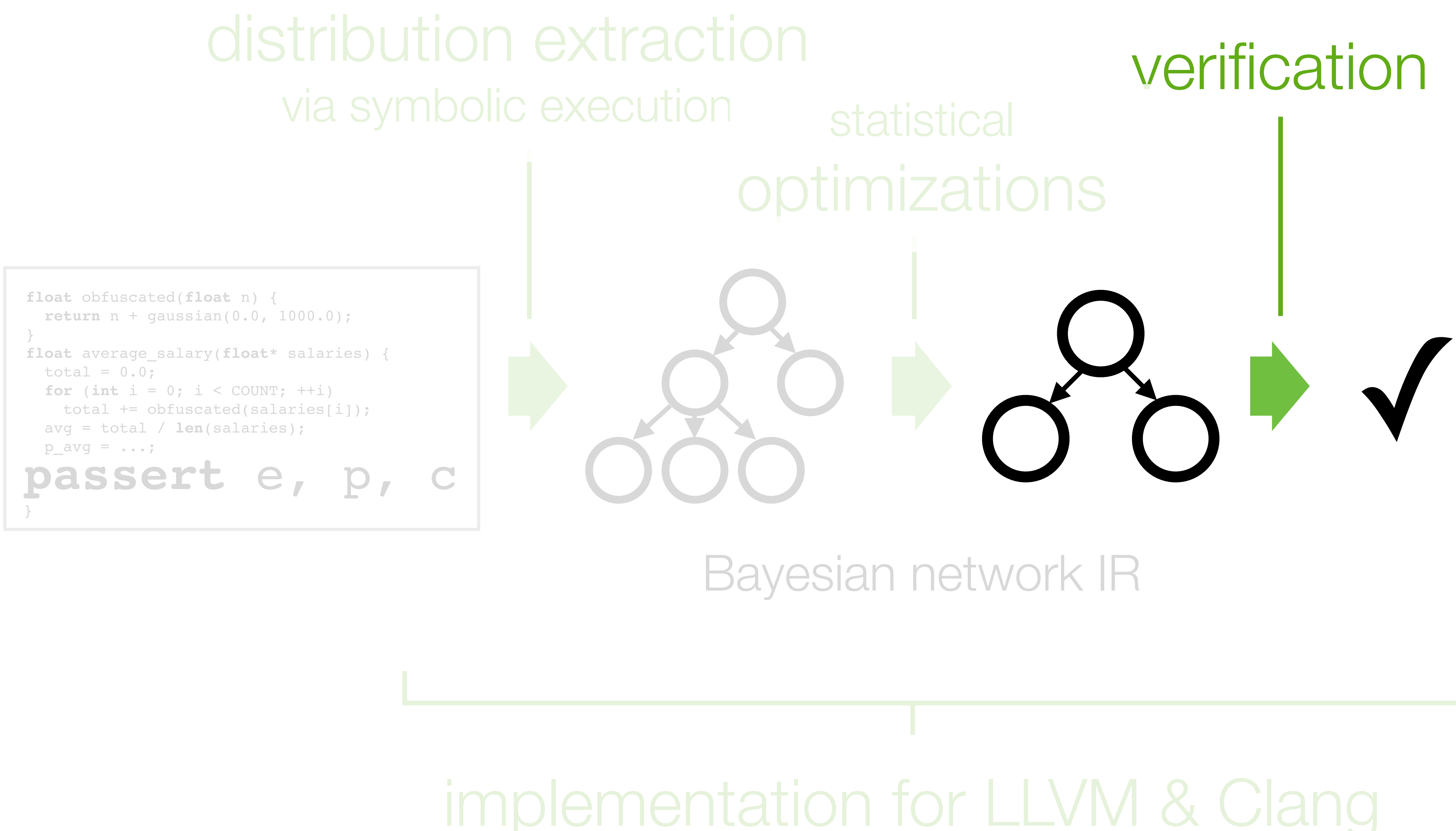
# Central Limit Theorem collapses large sums



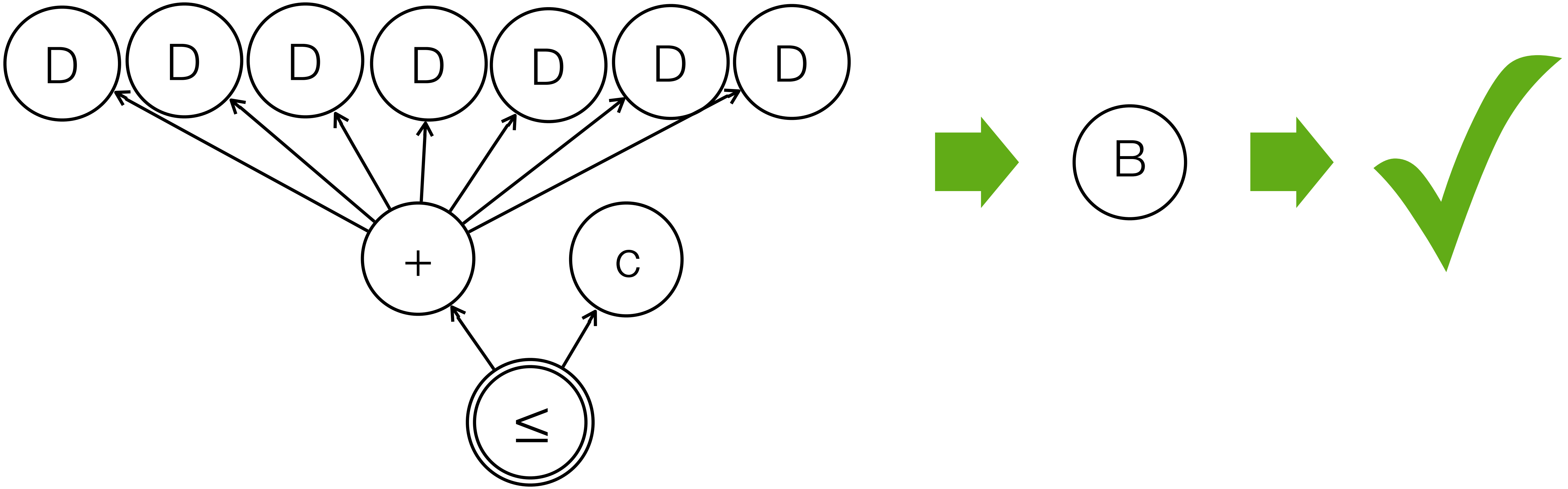

$$X_1, X_2, \ldots, X_n \sim D$$

$$Y = \sum_i X_i$$

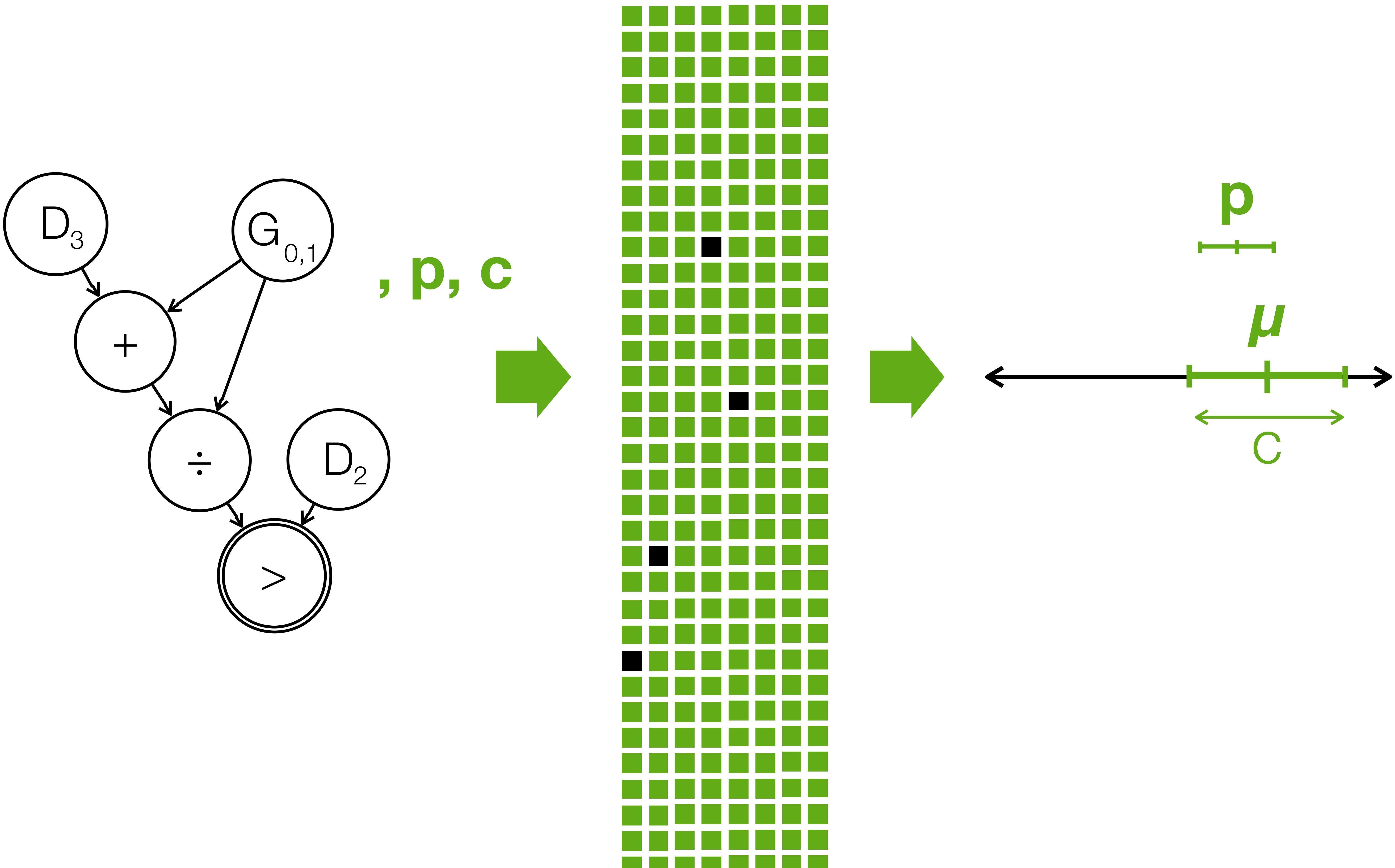
$$\Rightarrow Y \sim G(n\mu_D, n\sigma_D^2)$$


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
implementation for LLVM & Clang

# Verification via direct evaluation

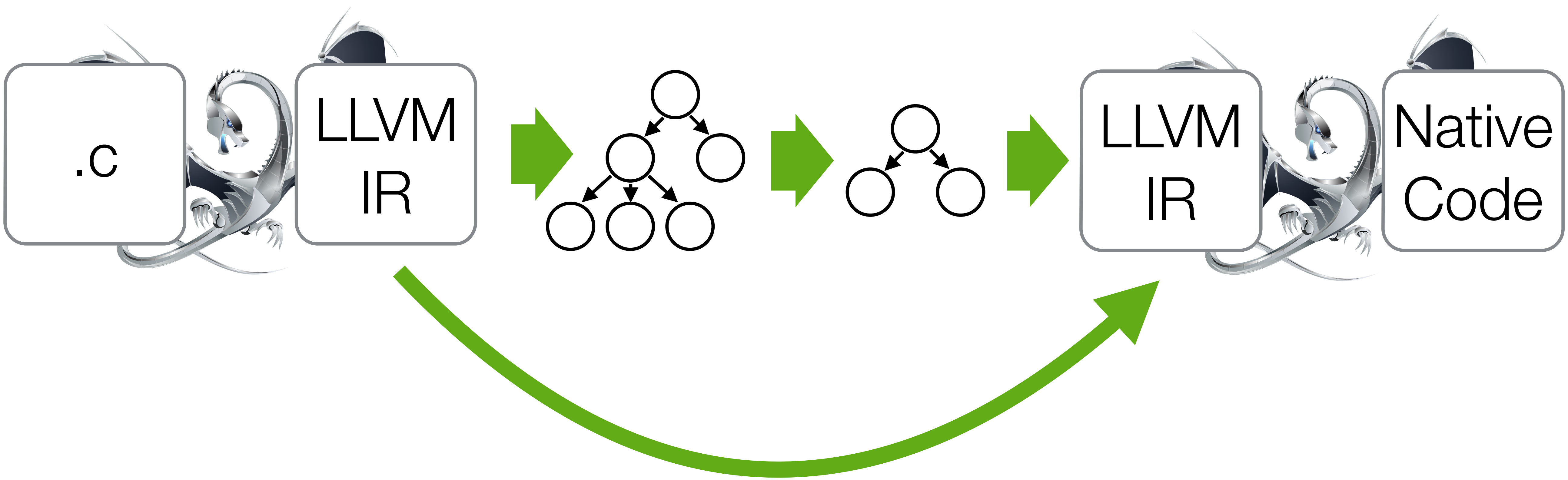


# Verification via hypothesis testing



implementation for LLVM & Clang

# Probabilistic assertions for C and C++

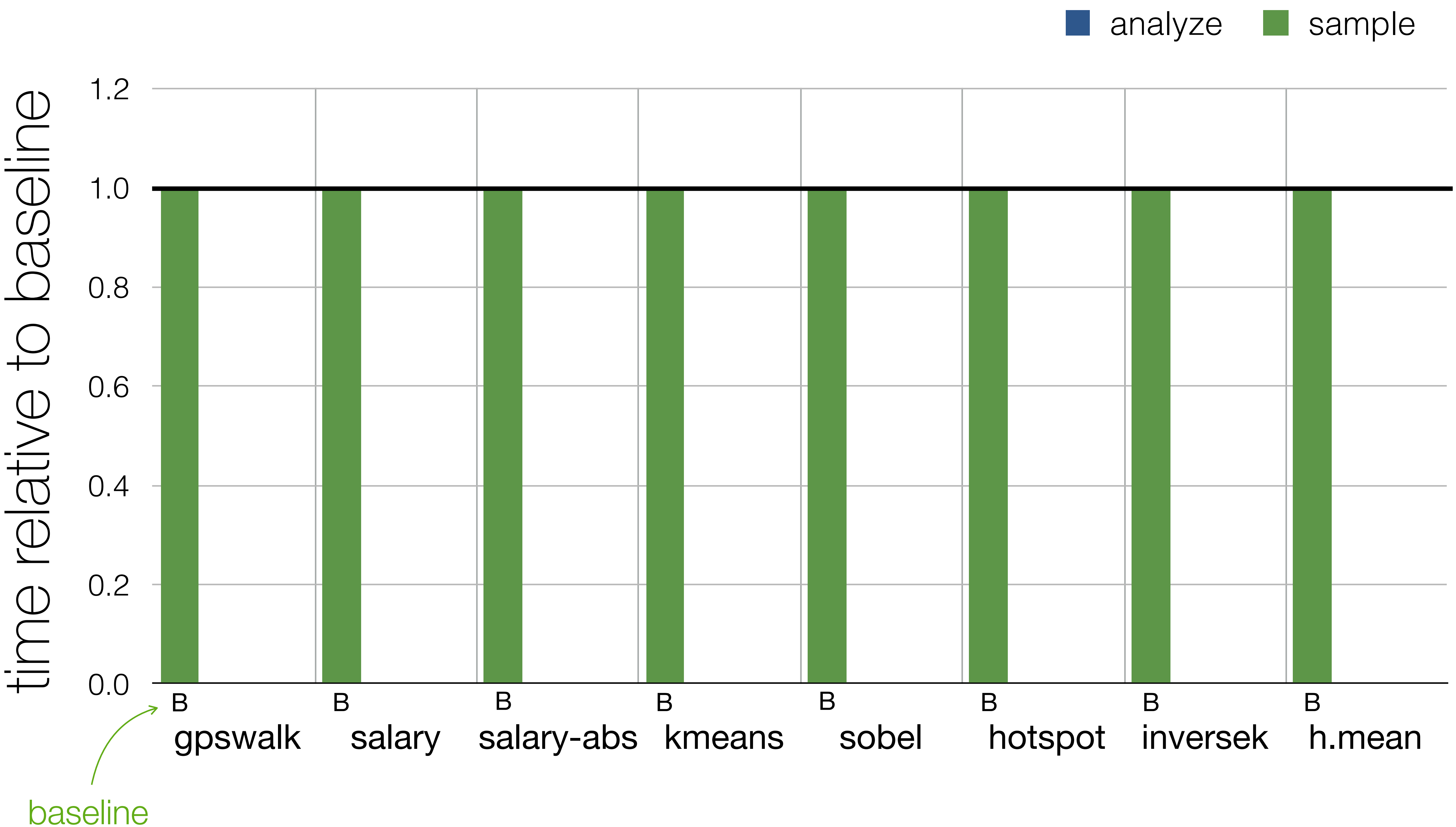



strawman stress-tester

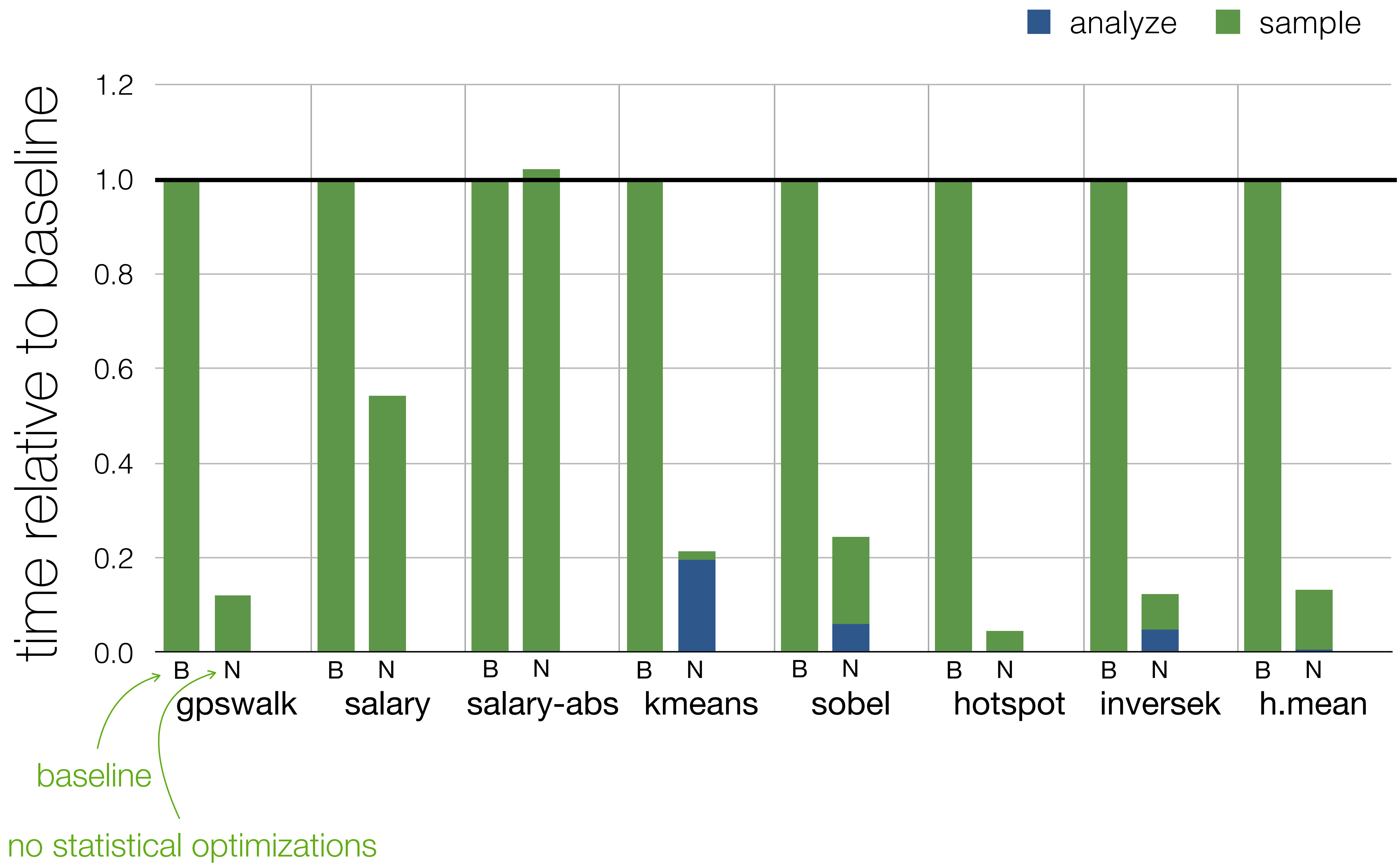
# Probabilistic programs used in the evaluation

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

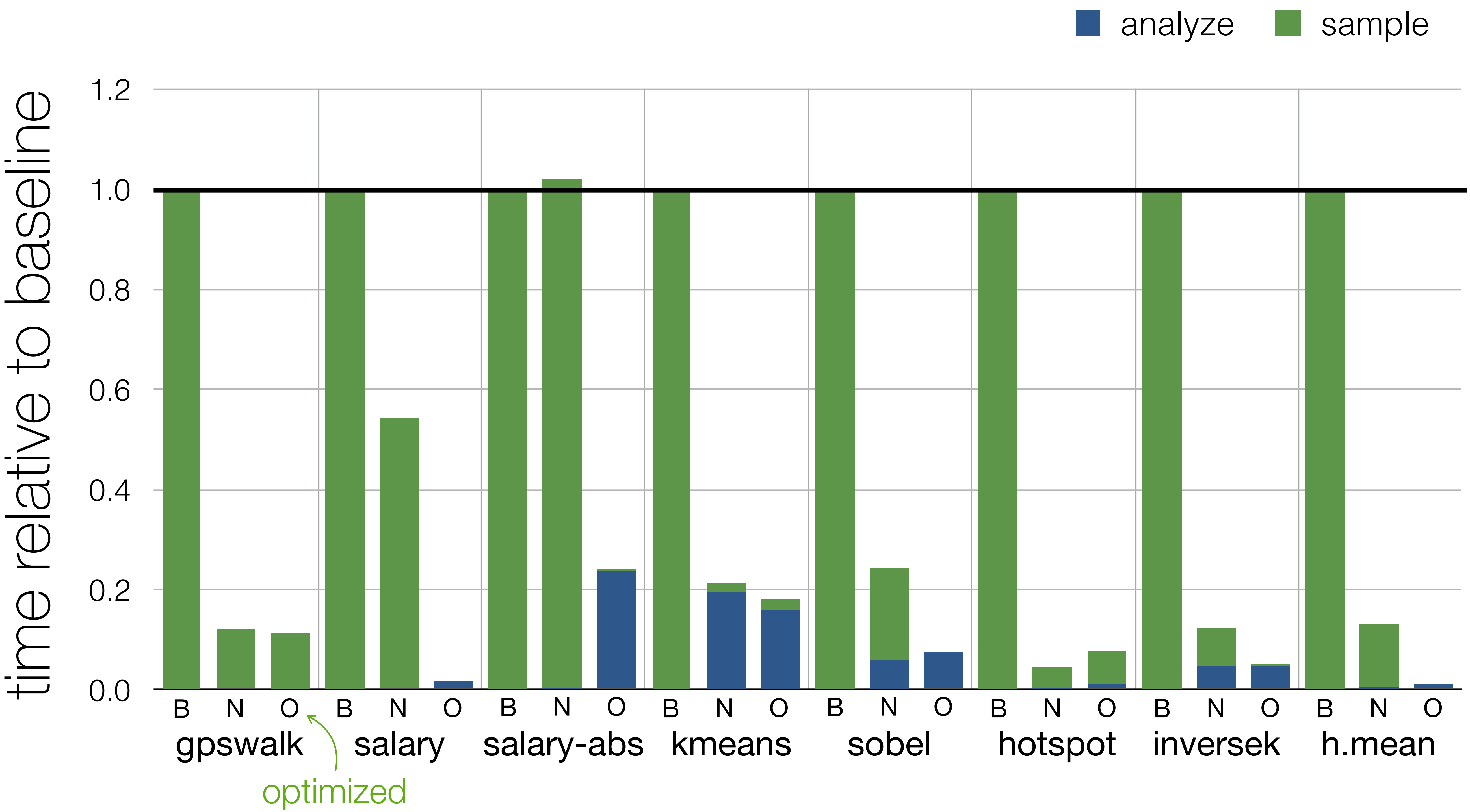
# Running time vs. stress testing



# Running time vs. stress testing



# Running time vs. stress testing




24× faster than baseline verifier on average

Mostly analysis time

**Probabilistic assertions** express correctness properties in modern software. Our verifier checks them **efficiently and accurately**.