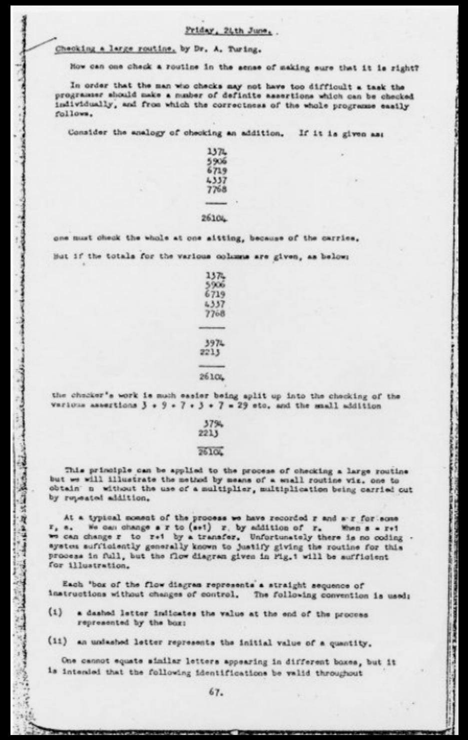
privacy and automated verification

aws albarghouthi university of wisconsin–madison

calvin smith

justin hsu

1949 1960 1970 1980 1990 2000



Friday, 24th June.

Checking a large routine, by Dr. A. Turing.

How can one check a routine in the sense of making sure that it is right?

In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.

Consider the analogy of checking an addition. If it is given as:

```
 1374
 5906
 6719
 4337
 7768
 ----
26104
```

one must check the whole at one sitting, because of the carries.

But if the totals for the various columns are given, as below:

```
 1374
 5906
 6719
 4337
 7768
 ----
 3974
2213
 ----
26104
```

the checker's work is much easier being split up into the checking of the various assertions 3 + 9 + 7 + 3 + 7 = 29 etc. and the small addition

```
 3794
2213
 ----
26104
```

This principle can be applied to the process of checking a large routine but we will illustrate the method by means of a small routine viz. one to obtain n without the use of a multiplier, multiplication being carried out by repeated addition.

At a typical moment of the process we have recorded r and s r for some r, s. We can change s r to (s+1) r by addition of r. When s = r+1 we can change r to r+1 by a transfer. Unfortunately there is no coding system sufficiently generally known to justify giving the routine for this process in full, but the flow diagram given in Fig.1 will be sufficient for illustration.

Each 'box of the flow diagram represents a straight sequence of instructions without changes of control. The following convention is used:

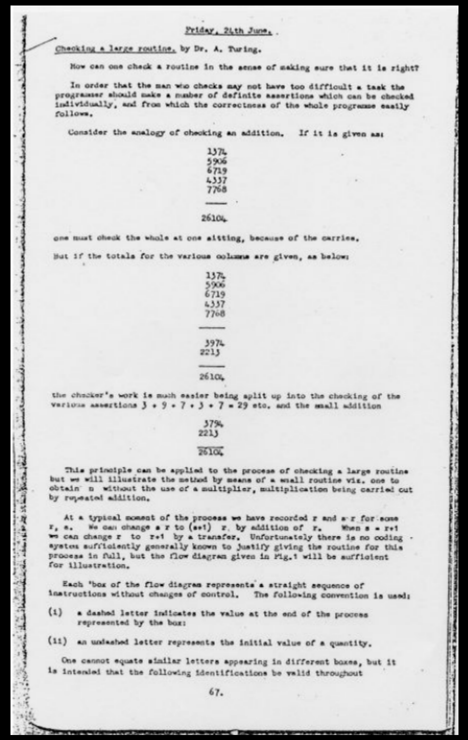
(i) a dashed letter indicates the value at the end of the process represented by the box:

(ii) an undashed letter represents the initial value of a quantity.

One cannot equate similar letters appearing in different boxes, but it is intended that the following identifications be valid throughout

67.

1949 1960 1970 1980 1990 2000



Friday, 24th June.

Checking a large routine, by Dr. A. Turing.

How can one check a routine in the sense of making sure that it is right?

In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.

Consider the analogy of checking an addition. If it is given as:

```
 1374
 5906
 6719
 4337
 7768
 ----
26104
```

one must check the whole at one sitting, because of the carries.

But if the totals for the various columns are given, as below:

```
 1374
 5906
 6719
 4337
 7768
 ----
 3794
2213
 ----
26104
```

the checker's work is much easier being split up into the checking of the various assertions 3 + 9 + 7 + 3 + 7 = 29 etc. and the small addition

```
 3794
2213
 ----
26104
```

This principle can be applied to the process of checking a large routine but we will illustrate the method by means of a small routine viz. one to obtain n without the use of a multiplier, multiplication being carried out by repeated addition.

At a typical moment of the process we have recorded r and s r for some r, s. We can change s r to (s+1) r by addition of r. When s = r+1 we can change r to r+1 by a transfer. Unfortunately there is no coding system sufficiently generally known to justify giving the routine for this process in full, but the flow diagram given in Fig.1 will be sufficient for illustration.

Each "box of the flow diagram represents a straight sequence of instructions without changes of control. The following convention is used:

(i) a dashed letter indicates the value at the end of the process represented by the box:

(ii) an undashed letter represents the initial value of a quantity.

One cannot equate similar letters appearing in different boxes, but it is intended that the following identifications be valid throughout

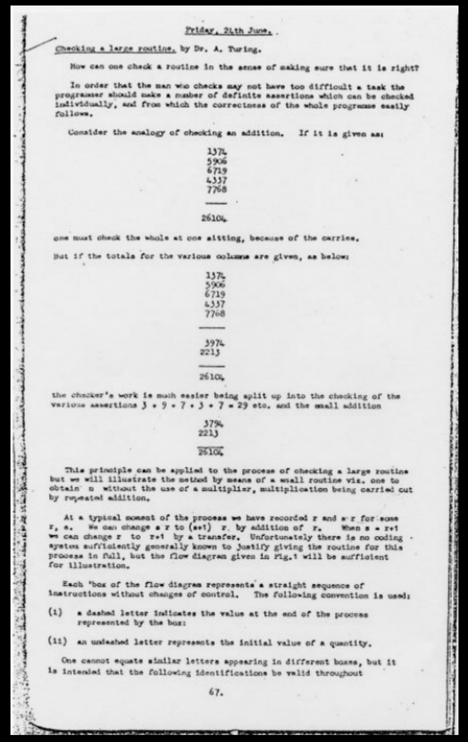
67.

program
logics

abstract interp
model checking

industrial
tools

1949 1960 1970 1980 1990 2000



Friday, 24th June.

Checking a large routine, by Dr. A. Turing.

How can one check a routine in the sense of making sure that it is right?

In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.

Consider the analogy of checking an addition. If it is given as:

```
 1374
 5906
 6719
 4337
 7768
 ----
26104
```

one must check the whole at one sitting, because of the carries.

But if the totals for the various columns are given, as below:

```
 1374
 5906
 6719
 4337
 7768
 ----
 3974
2213
 ----
26104
```

the checker's work is much easier being split up into the checking of the various assertions 3 + 9 + 7 + 3 + 7 = 29 etc. and the small addition

```
 3794
2213
 ----
26104
```

This principle can be applied to the process of checking a large routine but we will illustrate the method by means of a small routine viz. one to obtain n without the use of a multiplier, multiplication being carried out by repeated addition.

At a typical moment of the process we have recorded r and s r for some r, s. We can change s r to (s+1) r by addition of r. When s = r+1 we can change r to r+1 by a transfer. Unfortunately there is no coding system sufficiently generally known to justify giving the routine for this process in full, but the flow diagram given in Fig.1 will be sufficient for illustration.

Each 'box of the flow diagram represents a straight sequence of instructions without changes of control. The following convention is used:

(i) a dashed letter indicates the value at the end of the process represented by the box:

(ii) an undashed letter represents the initial value of a quantity.

One cannot equate similar letters appearing in different boxes, but it is intended that the following identifications be valid throughout

67.

program
logics

abstract interp
model checking

industrial
tools

1949 1960 1970 1980 1990 2000

```
assert(x ≠ null)
```

1 is my data private?

2 is the algorithm fair?

1 is my data private?

2 is the algorithm fair?

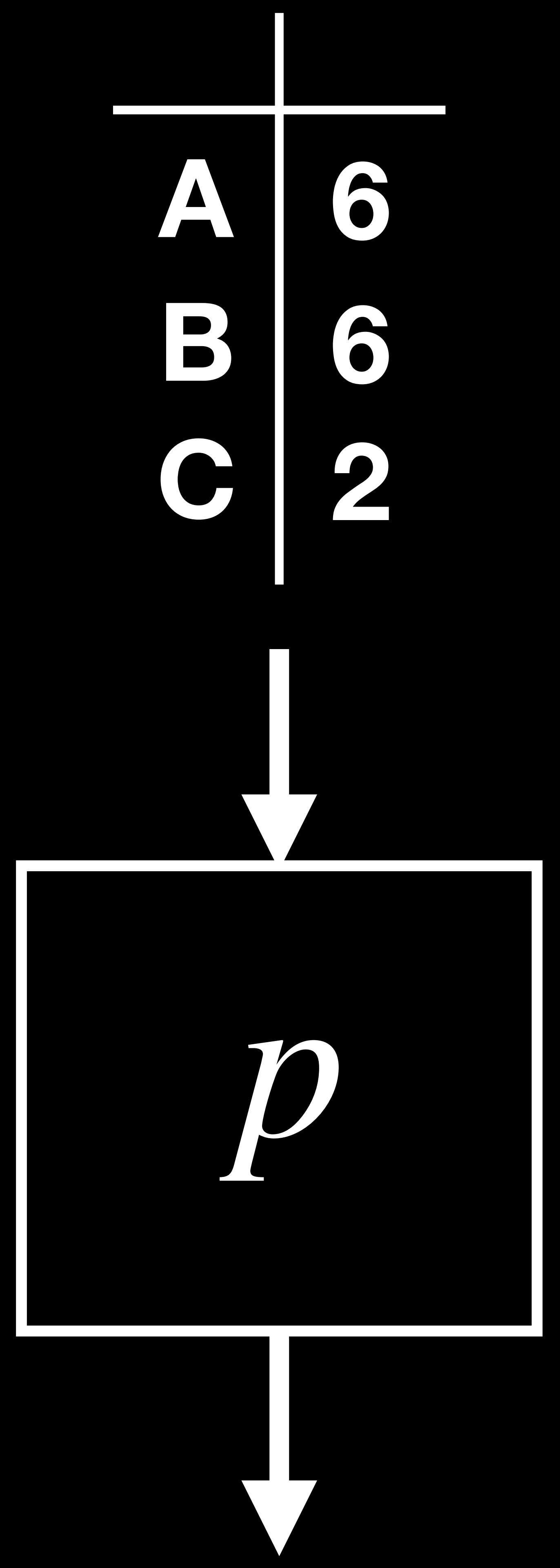
Robust De-anonymization of Large Sparse Datasets

Arvind Narayanan and Vitaly Shmatikov
The University of Texas at Austin

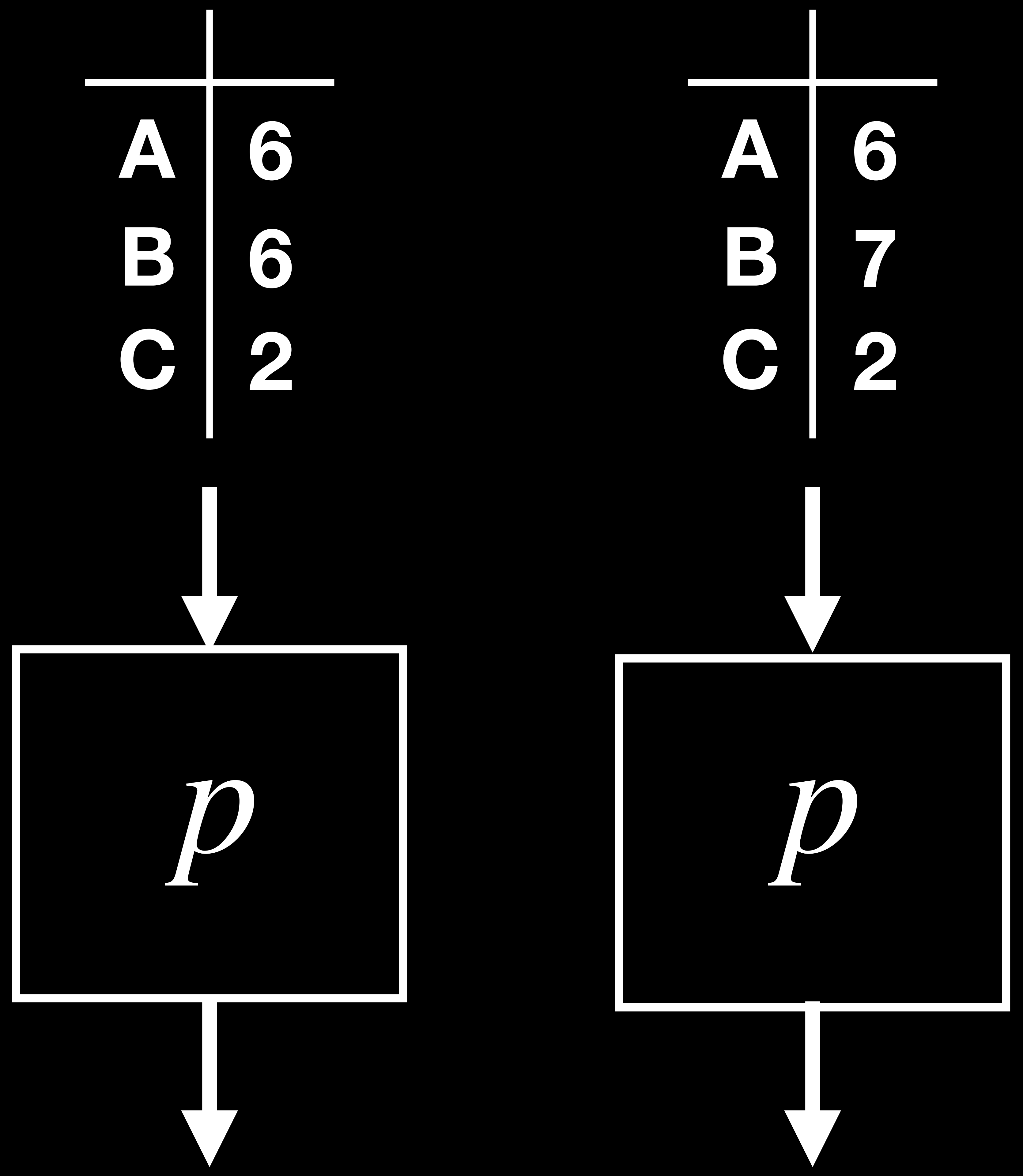
Identifying Participants in the Personal Genome Project by Name

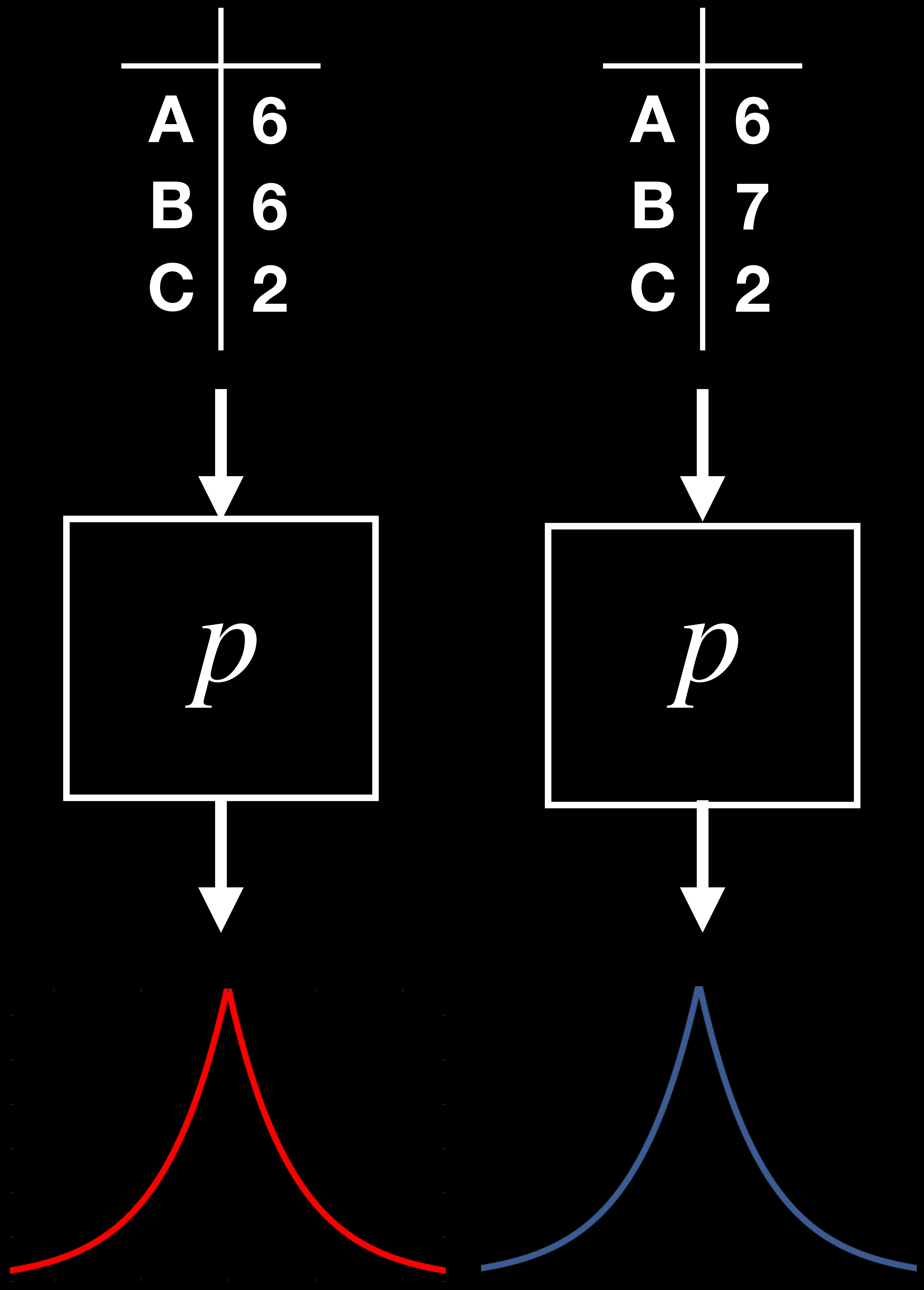
Latanya Sweeney, Akua Abu, Julia Winn

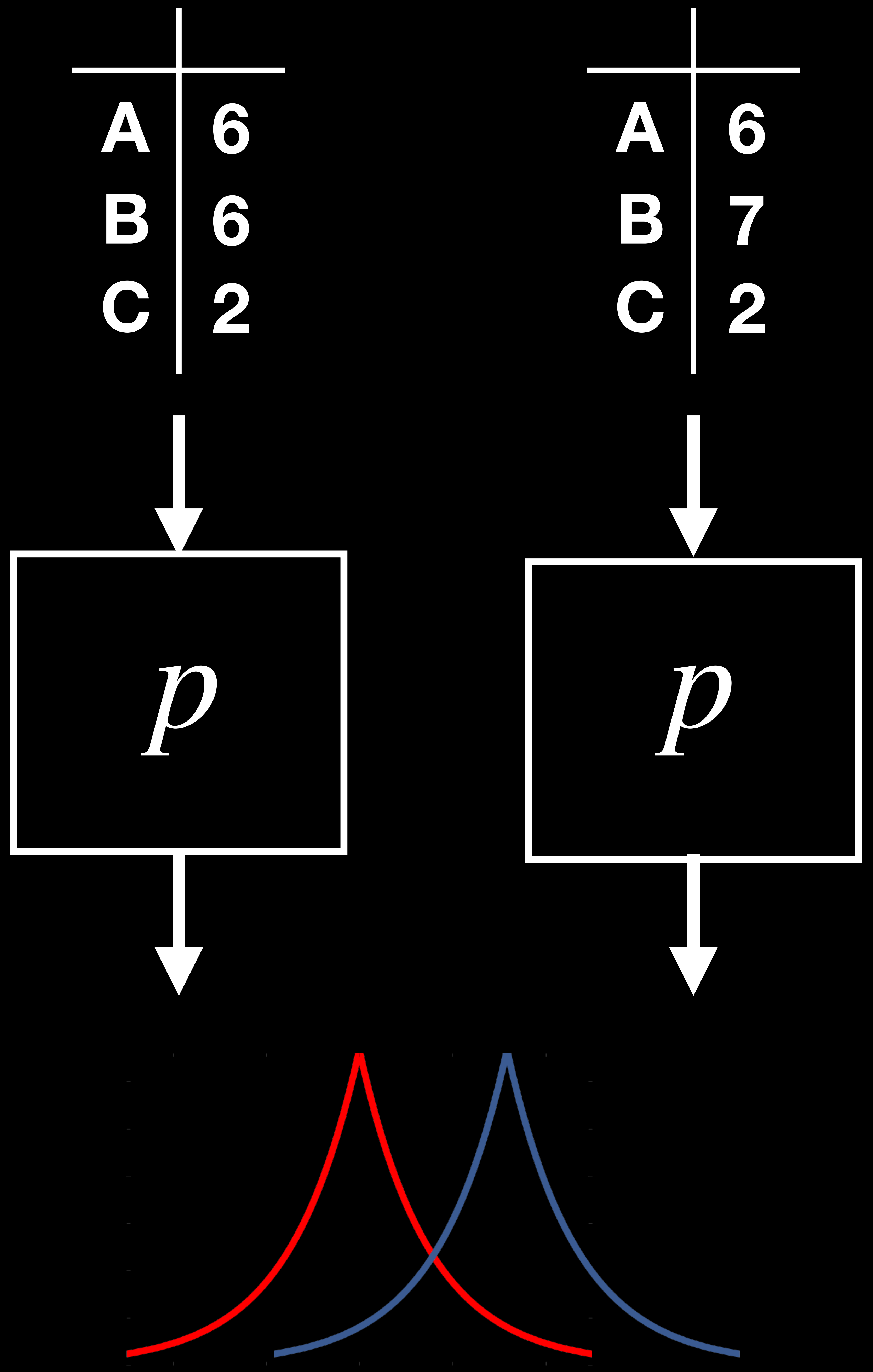
Harvard College

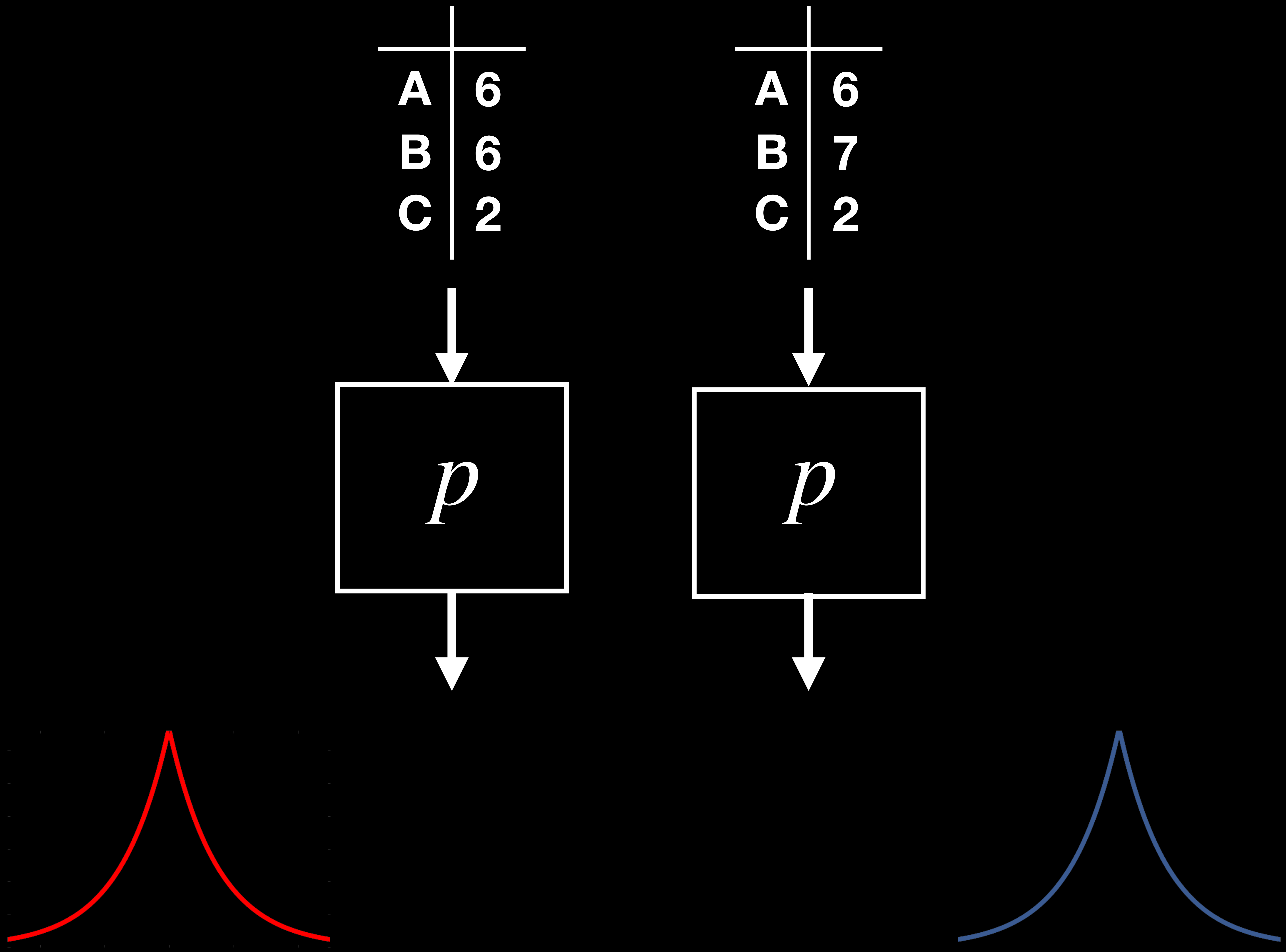


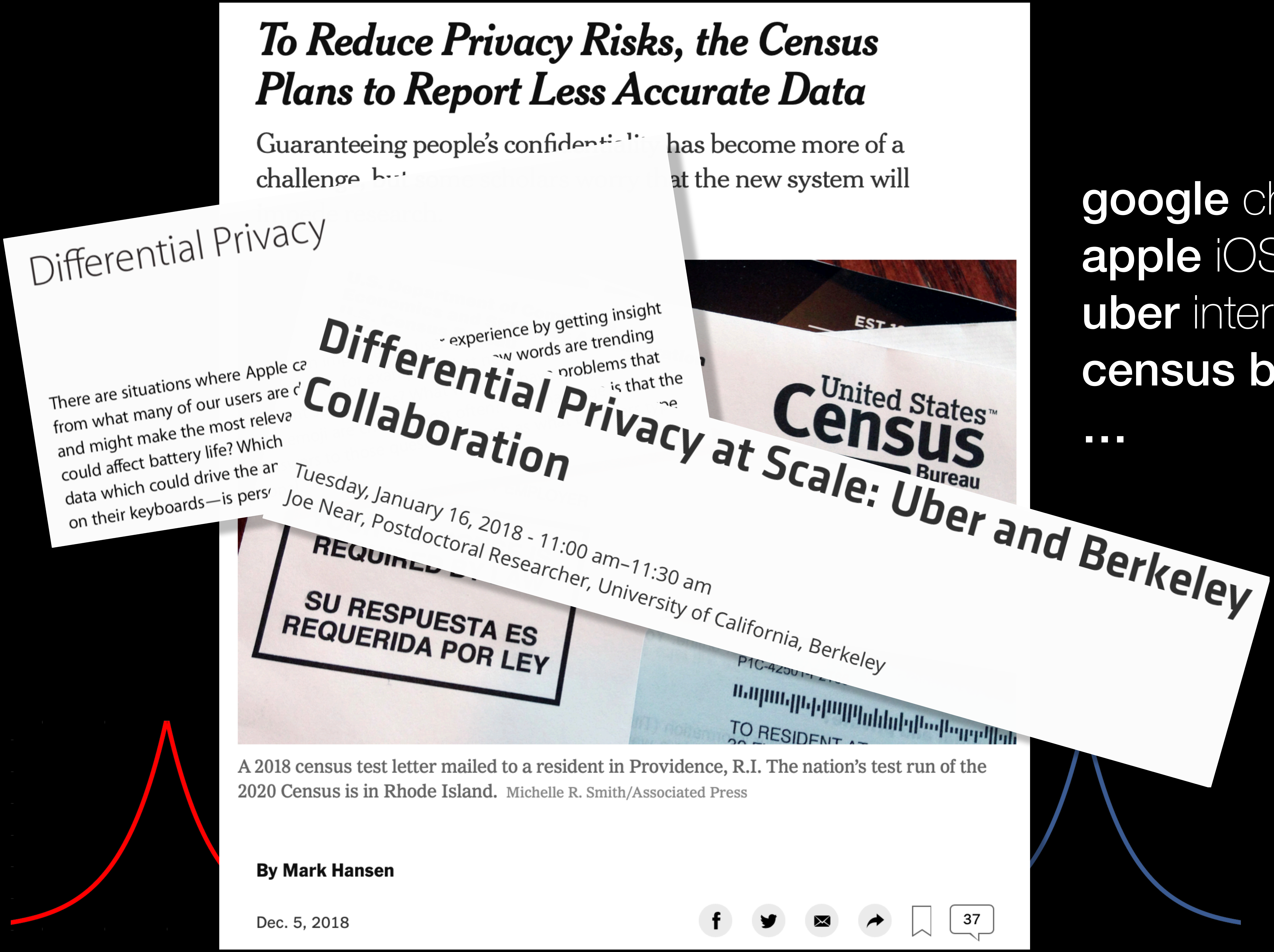
A 6
B 6
C 2
p


A 6
B 6
C 2
p
A 6
B 7
C 2
p

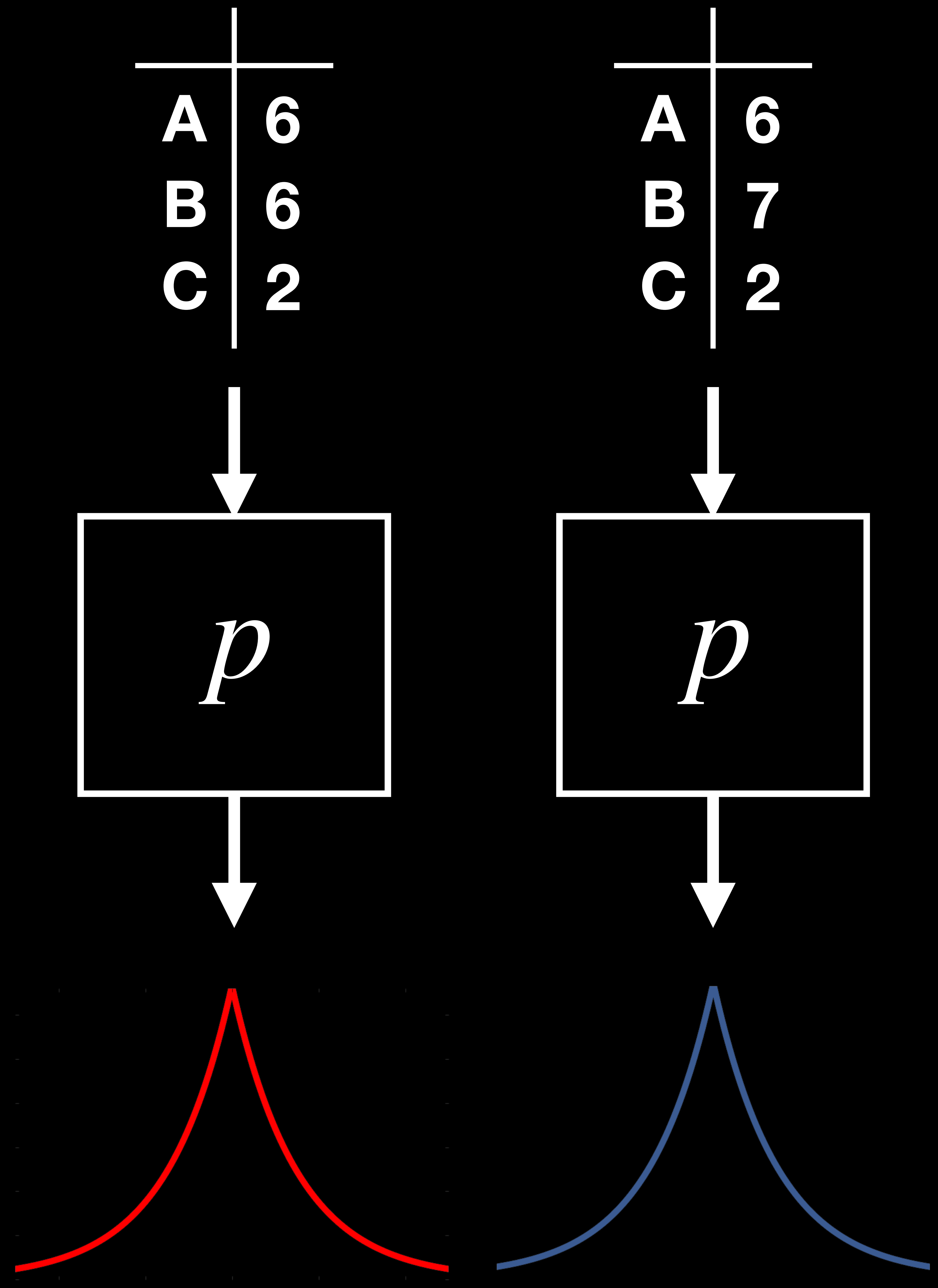

A 6
B 6
C 2
A 6
B 7
C 2
p
p


A 6
B 6
C 2
A 6
B 7
C 2
p
p

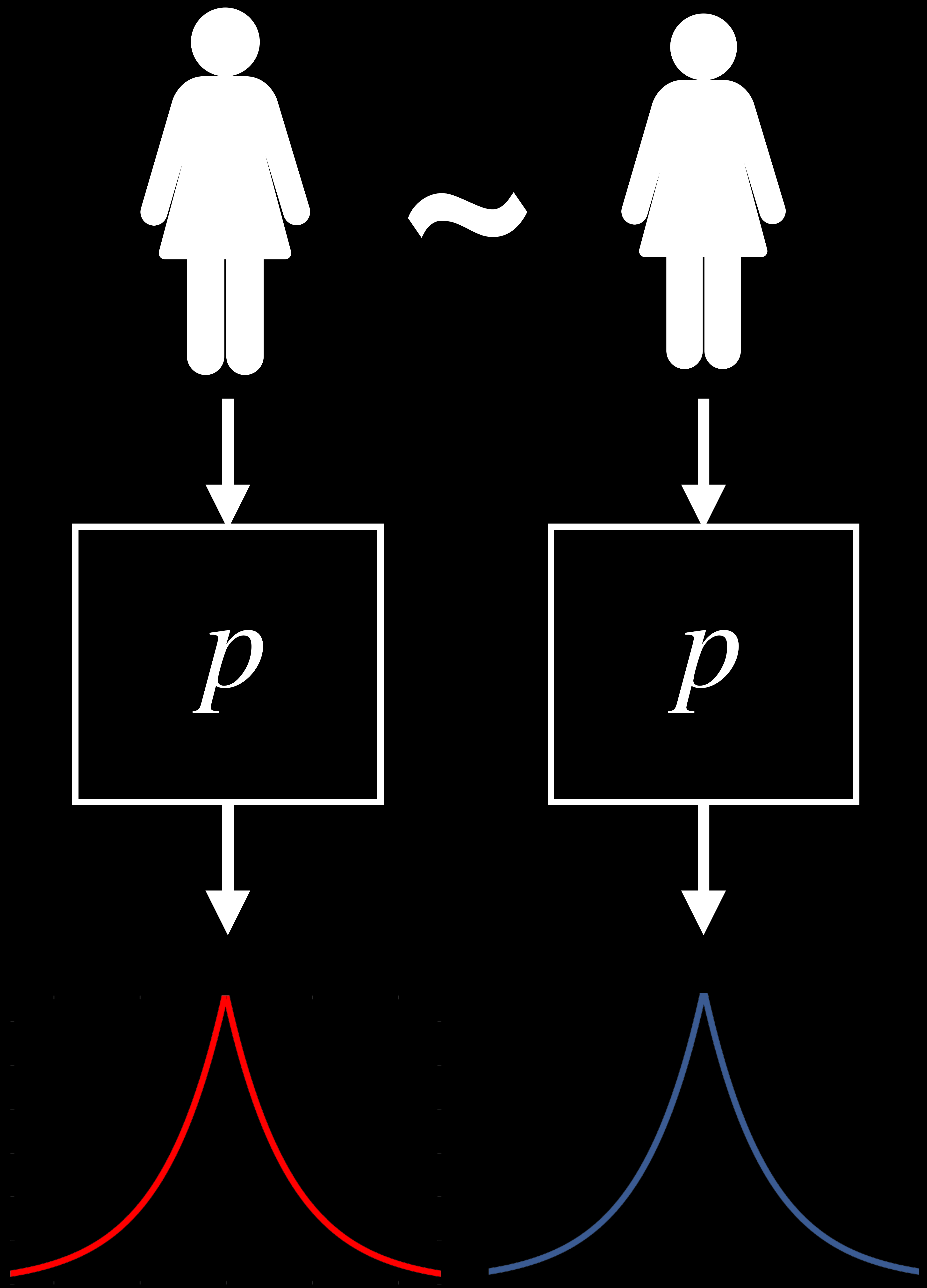

A 6
B 6
C 2
A 6
B 7
C 2
p
p

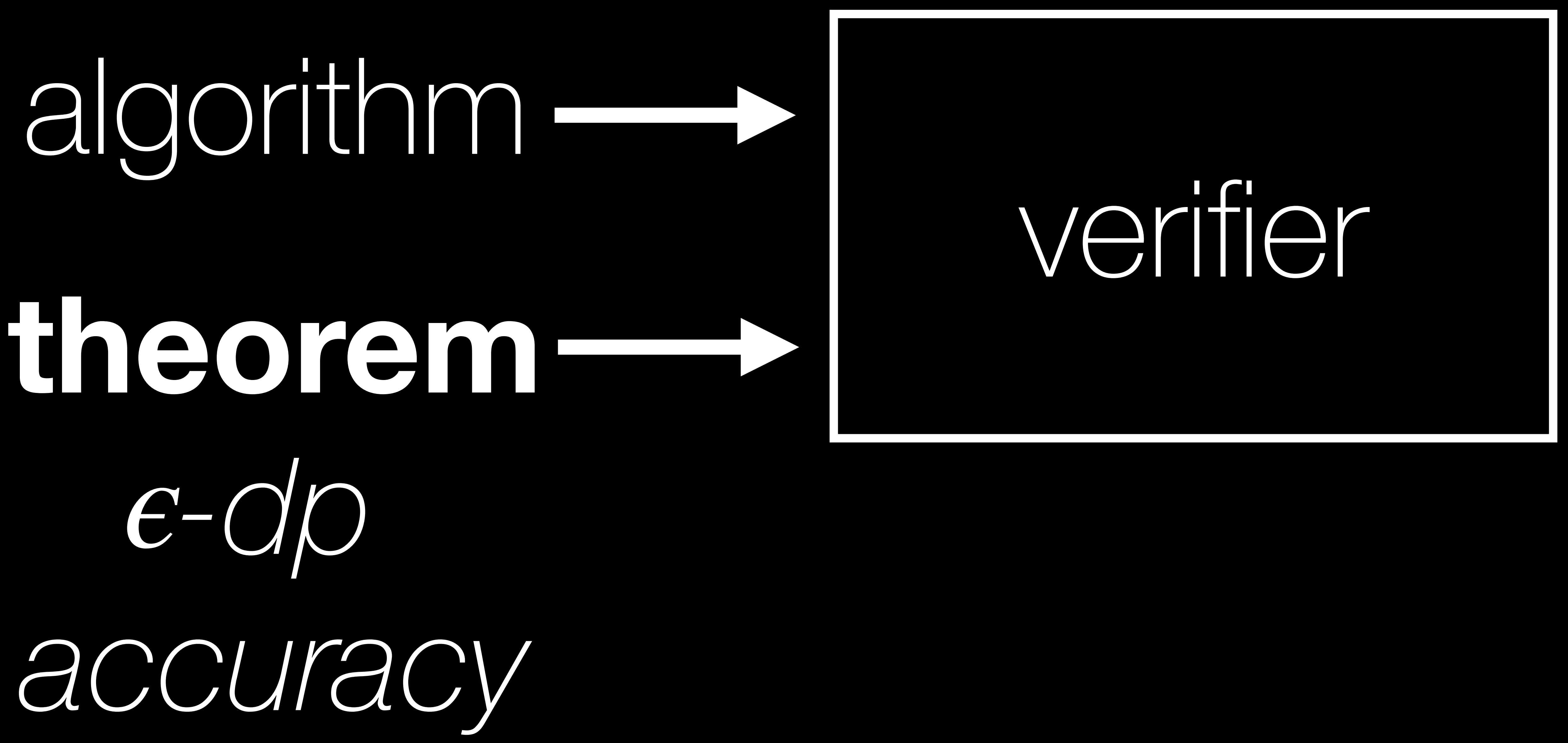


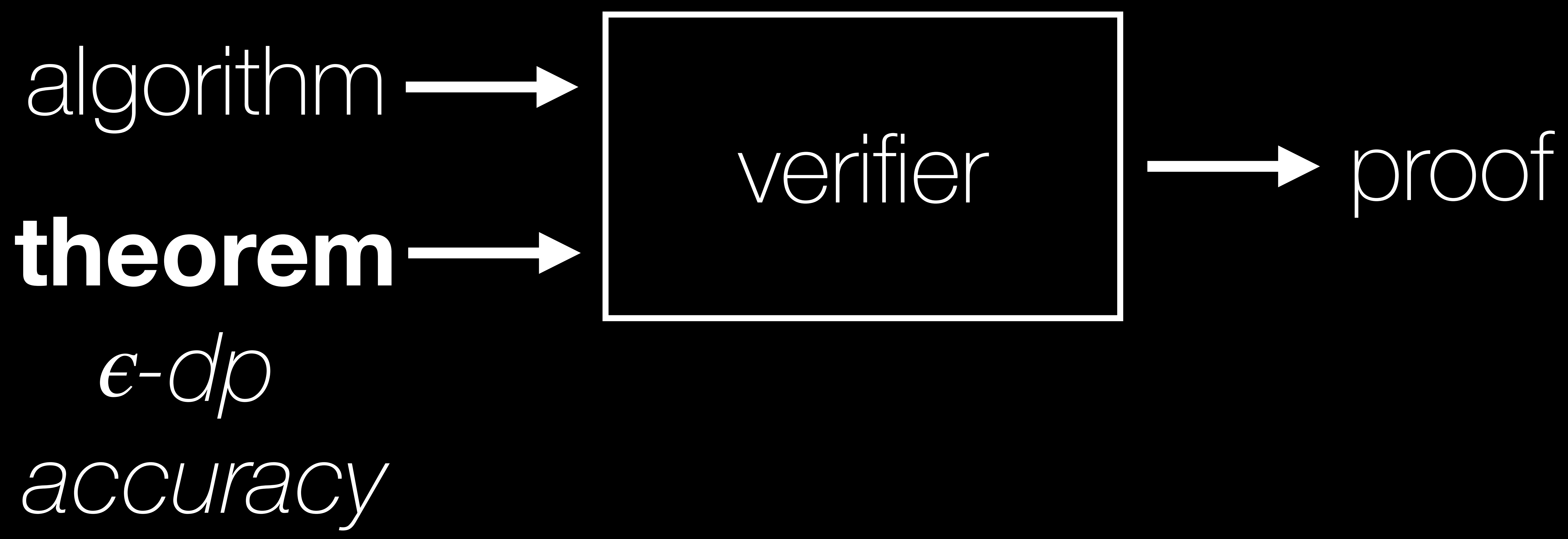
google chrome
apple iOS
uber internally
census bureau
...



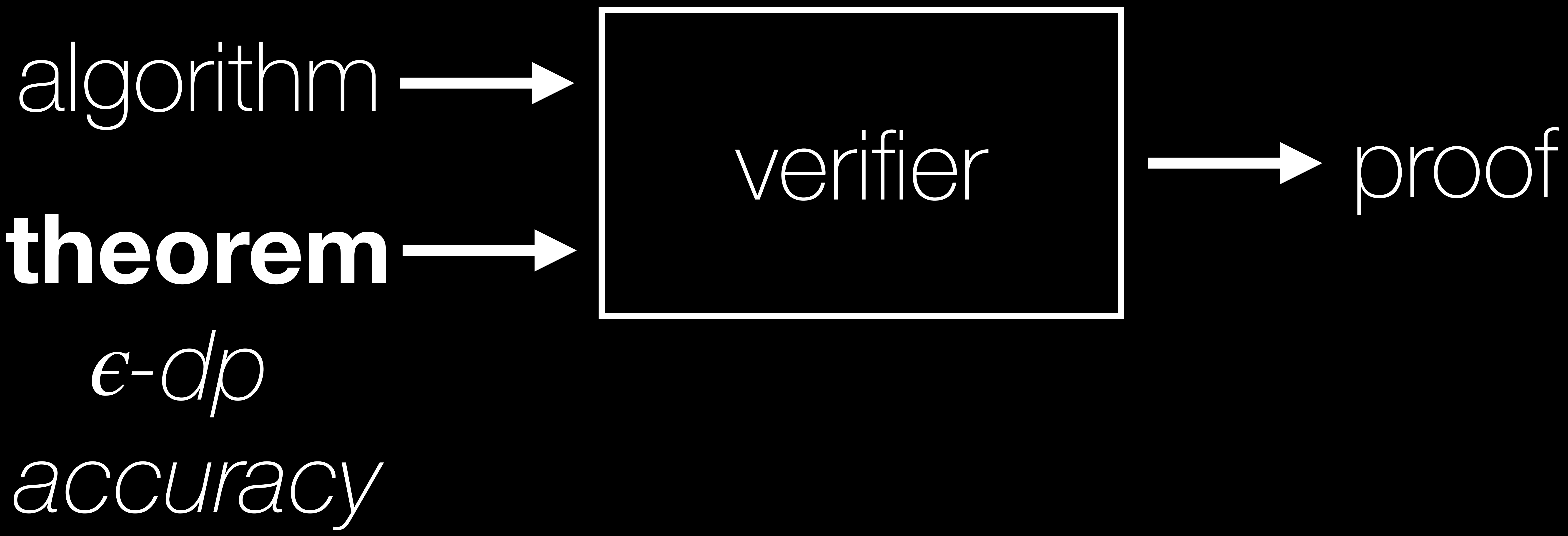
A 6
B 6
C 2
A 6
B 7
C 2
p
p


~
p
p


algorithm
theorem
ϵ-*dp*
accuracy
verifier

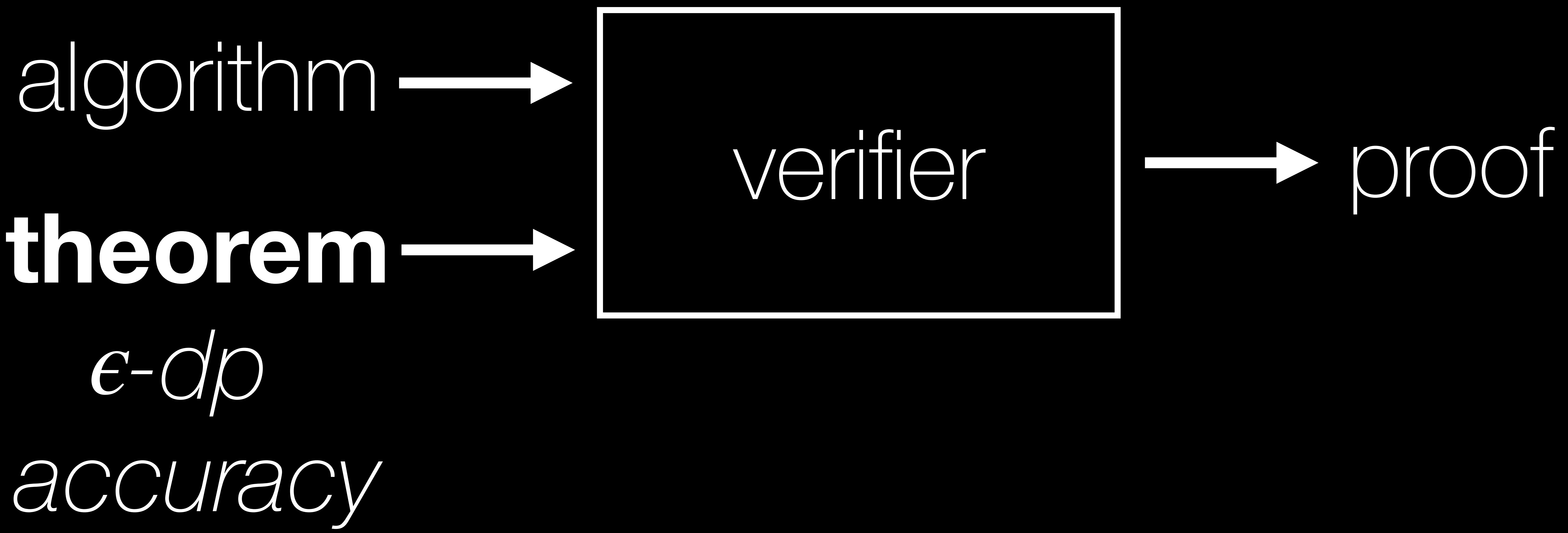

algorithm
theorem
ϵ-dp
accuracy
verifier
proof

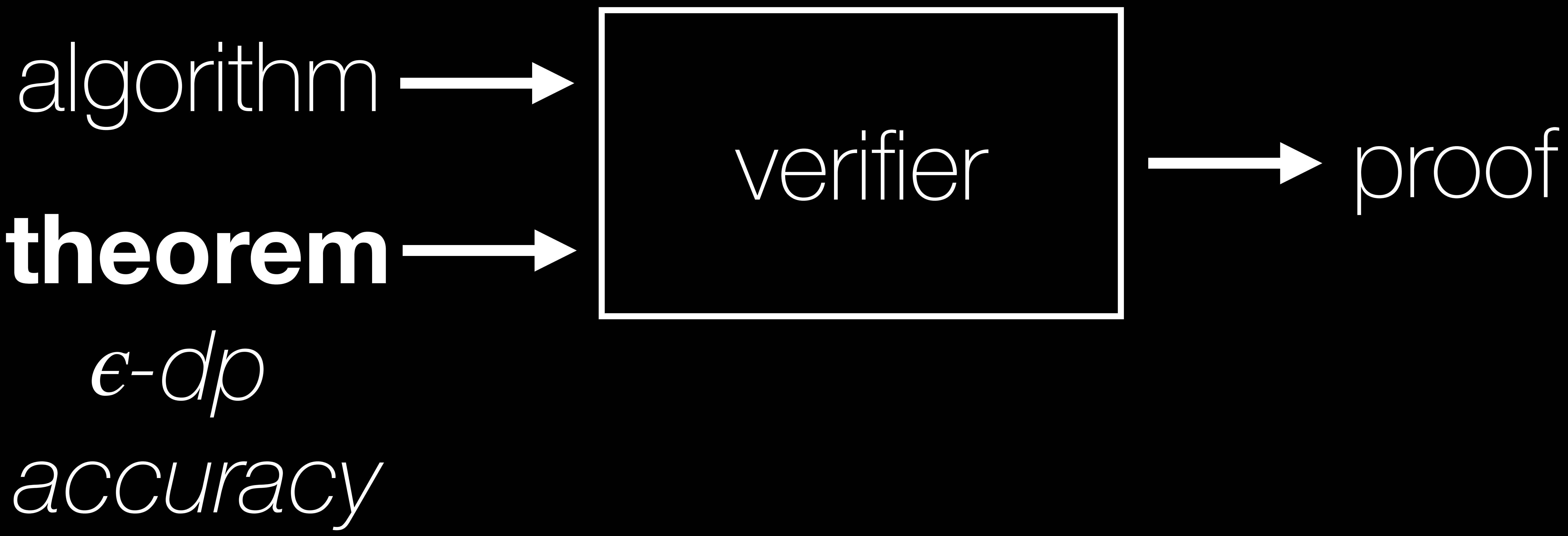
$$\forall d \sim d' \,.\, \mathbb{P}[p(d) = a] \leq e^{\epsilon} \cdot \mathbb{P}[p(d') = a]$$

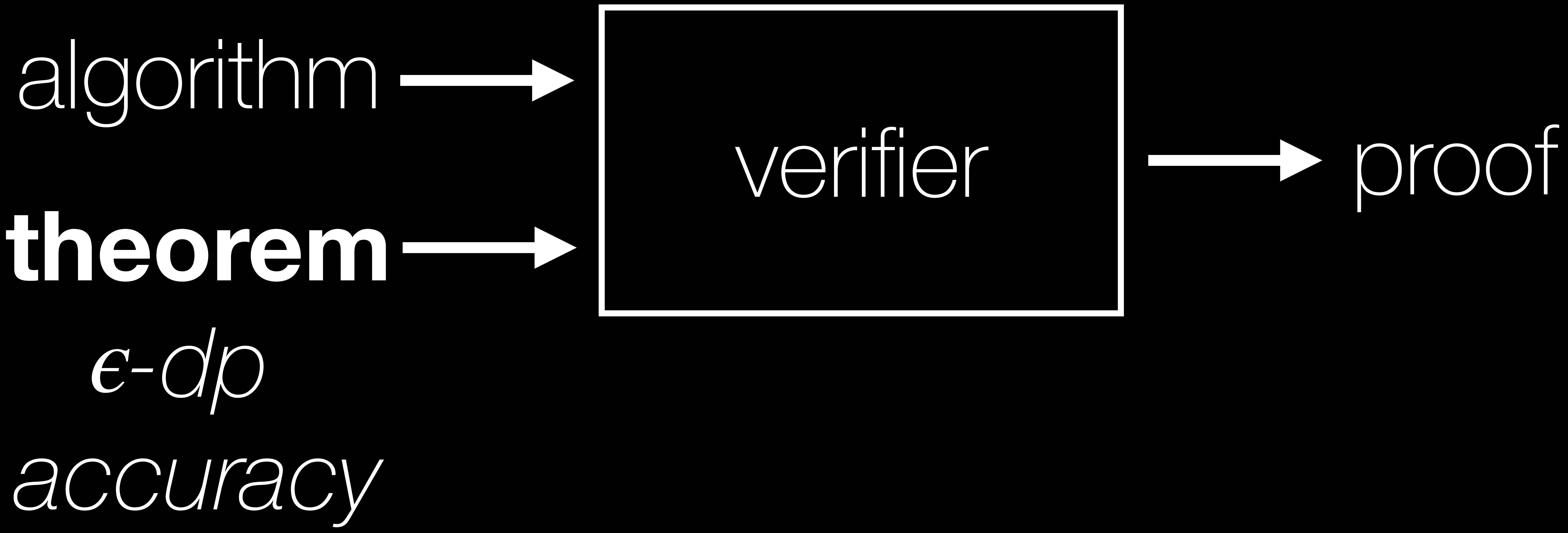


$$\forall d \sim d' .\, \mathbb{P}[p(d) = a] \leq e^{\epsilon} \cdot \mathbb{P}[p(d') = a]$$

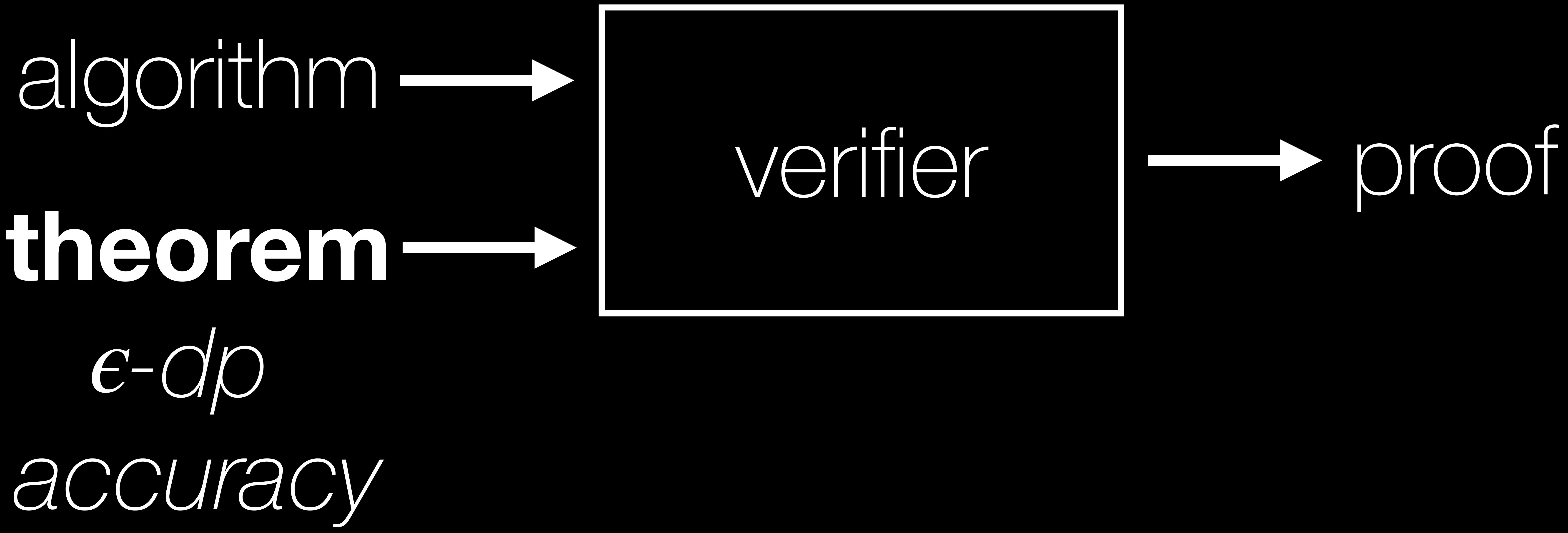
$$p(d) - p_{exact}(d) \leq \theta \quad \text{except with prob. } \delta$$




algorithm
theorem
ϵ-dp
accuracy
verifier
proof



short-term vision aid algorithm designers



short-term vision aid algorithm designers

long-term vision put theorists out of work

proof of **report noisy max**

Proof. Fix $D = D' \cup \{a\}$. Let c, respectively c', denote the vector of counts when the database is D, respectively D'. We use two properties:

1. *Monotonicity of Counts.* For all $j \in [m]$, $c_j \geq c'_j$; and
2. *Lipschitz Property.* For all $j \in [m]$, $1 + c'_j \geq c_j$.

Fix any $i \in [m]$. We will bound from above and below the ratio of the probabilities that i is selected with D and with D'.

Fix r_{-i}, a draw from $[\mathrm{Lap}(1/\varepsilon)]^{m-1}$ used for all the noisy counts except the ith count. We will argue for each r_{-i} independently. We

use the notation $\Pr[i|\xi]$ to mean the probability that the output of the Report Noisy Max algorithm is i, conditioned on ξ.

We first argue that $\Pr[i|D, r_{-i}] \leq e^{\varepsilon} \Pr[i|D', r_{-i}]$. Define

$$r^* = \min_{r_i} : c_i + r_i > c_j + r_j \ \forall j \neq i.$$

Note that, having fixed r_{-i}, i will be the output (the argmax noisy count) when the database is D if and only if $r_i \geq r^*$.
We have, for all $1 \leq j \neq i \leq m$:

$$\begin{aligned} c_i + r^* &> c_j + r_j \\ \Rightarrow (1 + c'_i) + r^* \geq c_i + r^* &> c_j + r_j \geq c'_j + r_j \\ \Rightarrow c'_i + (r^* + 1) &> c'_j + r_j. \end{aligned}$$

Thus, if $r_i \geq r^* + 1$, then the ith count will be the maximum when the database is D' and the noise vector is (r_i, r_{-i}). The probabilities below are over the choice of $r_i \sim \mathrm{Lap}(1/\varepsilon)$.

$$\begin{aligned} \Pr[r_i \geq 1 + r^*] \geq e^{-\varepsilon} \Pr[r_i \geq r^*] = e^{-\varepsilon} \Pr[i|D, r_{-i}] \\ \Rightarrow \Pr[i|D', r_{-i}] \geq \Pr[r_i \geq 1 + r^*] \geq e^{-\varepsilon} \Pr[r_i \geq r^*] = e^{-\varepsilon} \Pr[i|D, r_{-i}], \end{aligned}$$

which, after multiplying through by e^{ε}, yields what we wanted to show: $\Pr[i|D, r_{-i}] \leq e^{\varepsilon} \Pr[i|D', r_{-i}]$.

We now argue that $\Pr[i|D', r_{-i}] \leq e^{\varepsilon} \Pr[i|D, r_{-i}]$. Define

$$r^* = \min_{r_i} : c'_i + r_i > c'_j + r_j \ \forall j \neq i.$$

Note that, having fixed r_{-i}, i will be the output (argmax noisy count) when the database is D' if and only if $r_i \geq r^*$.
We have, for all $1 \leq j \neq i \leq m$:

$$\begin{aligned} c'_i + r^* &> c'_j + r_j \\ \Rightarrow 1 + c'_i + r^* &> 1 + c'_j + r_j \\ \Rightarrow c'_i + (r^* + 1) &> (1 + c'_j) + r_j \\ \Rightarrow c_i + (r^* + 1) \geq c'_i + (r^* + 1) &> (1 + c'_j) + r_j \geq c_j + r_j. \end{aligned}$$

Thus, if $r_i \geq r^* + 1$, then i will be the output (the argmax noisy count) on database D with randomness (r_i, r_{-i}). We therefore have, with probabilities taken over choice of r_i:

$$\Pr[i|D, r_{-i}] \geq \Pr[r_i \geq r^* + 1] \geq e^{-\varepsilon} \Pr[r_i \geq r^*] = e^{-\varepsilon} \Pr[i|D', r_{-i}],$$

[dwork/roth book]

proof of **report noisy max**

Proof. Fix $D = D' \cup \{a\}$. Let c, respectively c', denote the vector of counts when the database is D, respectively D'. We use two properties:

1. *Monotonicity of Counts.* For all $j \in [m]$, $c_j \geq c'_j$; and
2. *Lipschitz Property.* For all $j \in [m]$, $1 + c'_j \geq c_j$.

Fix any $i \in [m]$. We will bound from above and below the ratio of the probabilities that i is selected with D and with D'.

Fix r_{-i}, a draw from $[\mathrm{Lap}(1/\varepsilon)]^{m-1}$ used for all the noisy counts except the ith count. We will argue for each r_{-i} independently. We

use the notation $\Pr[i|\xi]$ to mean the probability that the output of the Report Noisy Max algorithm is i, conditioned on ξ.

We first argue that $\Pr[i|D, r_{-i}] \leq e^{\varepsilon} \Pr[i|D', r_{-i}]$. Define

$$r^* = \min_{r_i} : c_i + r_i > c_j + r_j \ \forall j \neq i.$$

Note that, having fixed r_{-i}, i will be the output (the argmax noisy count) when the database is D if and only if $r_i \geq r^*$.
We have, for all $1 \leq j \neq i \leq m$:

$$\begin{aligned} c_i + r^* &> c_j + r_j \\ \Rightarrow (1 + c'_i) + r^* \geq c_i + r^* &> c_j + r_j \geq c'_j + r_j \\ \Rightarrow c'_i + (r^* + 1) &> c'_j + r_j. \end{aligned}$$

Thus, if $r_i \geq r^* + 1$, then the ith count will be the maximum when the database is D' and the noise vector is (r_i, r_{-i}). The probabilities below are over the choice of $r_i \sim \mathrm{Lap}(1/\varepsilon)$.

$$\begin{aligned} \Pr[r_i \geq 1 + r^*] &\geq e^{-\varepsilon} \Pr[r_i \geq r^*] = e^{-\varepsilon} \Pr[i|D, r_{-i}] \\ \Rightarrow \Pr[i|D', r_{-i}] \geq \Pr[r_i \geq 1 + r^*] &\geq e^{-\varepsilon} \Pr[r_i \geq r^*] = e^{-\varepsilon} \Pr[i|D, r_{-i}], \end{aligned}$$

which, after multiplying through by e^{ε}, yields what we wanted to show: $\Pr[i|D, r_{-i}] \leq e^{\varepsilon} \Pr[i|D', r_{-i}]$.

We now argue that $\Pr[i|D', r_{-i}] \leq e^{\varepsilon} \Pr[i|D, r_{-i}]$. Define

$$r^* = \min_{r_i} : c'_i + r_i > c'_j + r_j \ \forall j \neq i.$$

Note that, having fixed r_{-i}, i will be the output (argmax noisy count) when the database is D' if and only if $r_i \geq r^*$.
We have, for all $1 \leq j \neq i \leq m$:

$$\begin{aligned} c'_i + r^* &> c'_j + r_j \\ \Rightarrow 1 + c'_i + r^* &> 1 + c'_j + r_j \\ \Rightarrow c'_i + (r^* + 1) &> (1 + c'_j) + r_j \\ \Rightarrow c_i + (r^* + 1) \geq c'_i + (r^* + 1) &> (1 + c'_j) + r_j \geq c_j + r_j. \end{aligned}$$

Thus, if $r_i \geq r^* + 1$, then i will be the output (the argmax noisy count) on database D with randomness (r_i, r_{-i}). We therefore have, with probabilities taken over choice of r_i:

$$\Pr[i|D, r_{-i}] \geq \Pr[r_i \geq r^* + 1] \geq e^{-\varepsilon} \Pr[r_i \geq r^*] = e^{-\varepsilon} \Pr[i|D', r_{-i}],$$

proof of **exponential mech**

Proof. For clarity, we assume the range $\mathcal{R}$ of the exponential mechanism is finite, but this is not necessary. As in all differential privacy proofs, we consider the ratio of the probability that an instantiation

of the exponential mechanism outputs some element $r \in \mathcal{R}$ on two neighboring databases $x \in \mathbb{N}^{|\mathcal{X}|}$ and $y \in \mathbb{N}^{|\mathcal{X}|}$ (i.e., $\|x - y\|_1 \leq 1$).

$$\begin{aligned} \frac{\Pr[\mathcal{M}_E(x, u, \mathcal{R}) = r]}{\Pr[\mathcal{M}_E(y, u, \mathcal{R}) = r]} &= \frac{\left(\frac{\exp(\frac{\varepsilon u(x,r)}{2\Delta u})}{\sum_{r' \in \mathcal{R}} \exp(\frac{\varepsilon u(x,r')}{2\Delta u})}\right)}{\left(\frac{\exp(\frac{\varepsilon u(y,r)}{2\Delta u})}{\sum_{r' \in \mathcal{R}} \exp(\frac{\varepsilon u(y,r')}{2\Delta u})}\right)} \\ &= \left(\frac{\exp(\frac{\varepsilon u(x,r)}{2\Delta u})}{\exp(\frac{\varepsilon u(y,r)}{2\Delta u})}\right) \cdot \left(\frac{\sum_{r' \in \mathcal{R}} \exp(\frac{\varepsilon u(y,r')}{2\Delta u})}{\sum_{r' \in \mathcal{R}} \exp(\frac{\varepsilon u(x,r')}{2\Delta u})}\right) \\ &= \exp\left(\frac{\varepsilon(u(x,r') - u(y,r'))}{2\Delta u}\right) \\ &\quad \cdot \left(\frac{\sum_{r' \in \mathcal{R}} \exp(\frac{\varepsilon u(y,r')}{2\Delta u})}{\sum_{r' \in \mathcal{R}} \exp(\frac{\varepsilon u(x,r')}{2\Delta u})}\right) \\ &\leq \exp\left(\frac{\varepsilon}{2}\right) \cdot \exp\left(\frac{\varepsilon}{2}\right) \cdot \left(\frac{\sum_{r' \in \mathcal{R}} \exp(\frac{\varepsilon u(x,r')}{2\Delta u})}{\sum_{r' \in \mathcal{R}} \exp(\frac{\varepsilon u(x,r')}{2\Delta u})}\right) \\ &= \exp(\varepsilon). \end{aligned}$$

Similarly, $\frac{\Pr[\mathcal{M}_E(y,u)=r]}{\Pr[\mathcal{M}_E(x,u)=r]} \geq \exp(-\varepsilon)$ by symmetry. □

[dwork/roth book]

logics

barthe et al. 2018
chadha et al. 2007
den harton 2002
rand and zdancewic 2015
…

martingales

chakarov and sankaranarayan 2013
chatterjee et al. 2016{a, b}, 2017
mclver et al. 2018
…

pre-expectation calculus

kozen 1985
morgan et al. 1996
…

probabilistic model checking

survey by katoen 2016
hermanns et al. 2008
kattenbelt et al. 2009, 2010
tiege and fränzle 2011
…

…and more!

logics

not automated

martingales

chakarov and sankaranarayan 2013
chatterjee et al. 2016{a, b}, 2017
mclver et al. 2018
. . .

pre-expectation calculus

kozen 1985
morgan et al. 1996
. . .

probabilistic model checking

survey by katoen 2016
hermanns et al. 2008
kattenbelt et al. 2009, 2010
tiege and fränzle 2011
. . .

. . .and more!

logics
not automated

martingales
require additional manual reasoning

pre-expectation calculus

kozen 1985
morgan et al. 1996
…

probabilistic model checking

survey by katoen 2016
hermanns et al. 2008
kattenbelt et al. 2009, 2010
tiege and fränzle 2011
…

…and more!

logics
not automated

martingales
require additional manual reasoning

pre-expectation calculus
complex integrals

probabilistic model checking

survey by katoen 2016
hermanns et al. 2008
kattenbelt et al. 2009, 2010
tiege and fränzle 2011
. . .

. . .and more!

logics
not automated

martingales
require additional manual reasoning

pre-expectation calculus
complex integrals

probabilistic model checking
(mostly) finite state and no symbolic distributions

…and more!

1 automatic proofs of accuracy [POPL19]

1 automatic proofs of accuracy [POPL19]

2 automatic proofs of differential privacy [POPL18]

1 automatic proofs of accuracy [POPL19]

2 automatic proofs of differential privacy [POPL18]

synthesis of privacy-preserving algorithms [ICFP 19]

1 automatic proofs of accuracy [POPL19]

2 automatic proofs of differential privacy [POPL18]

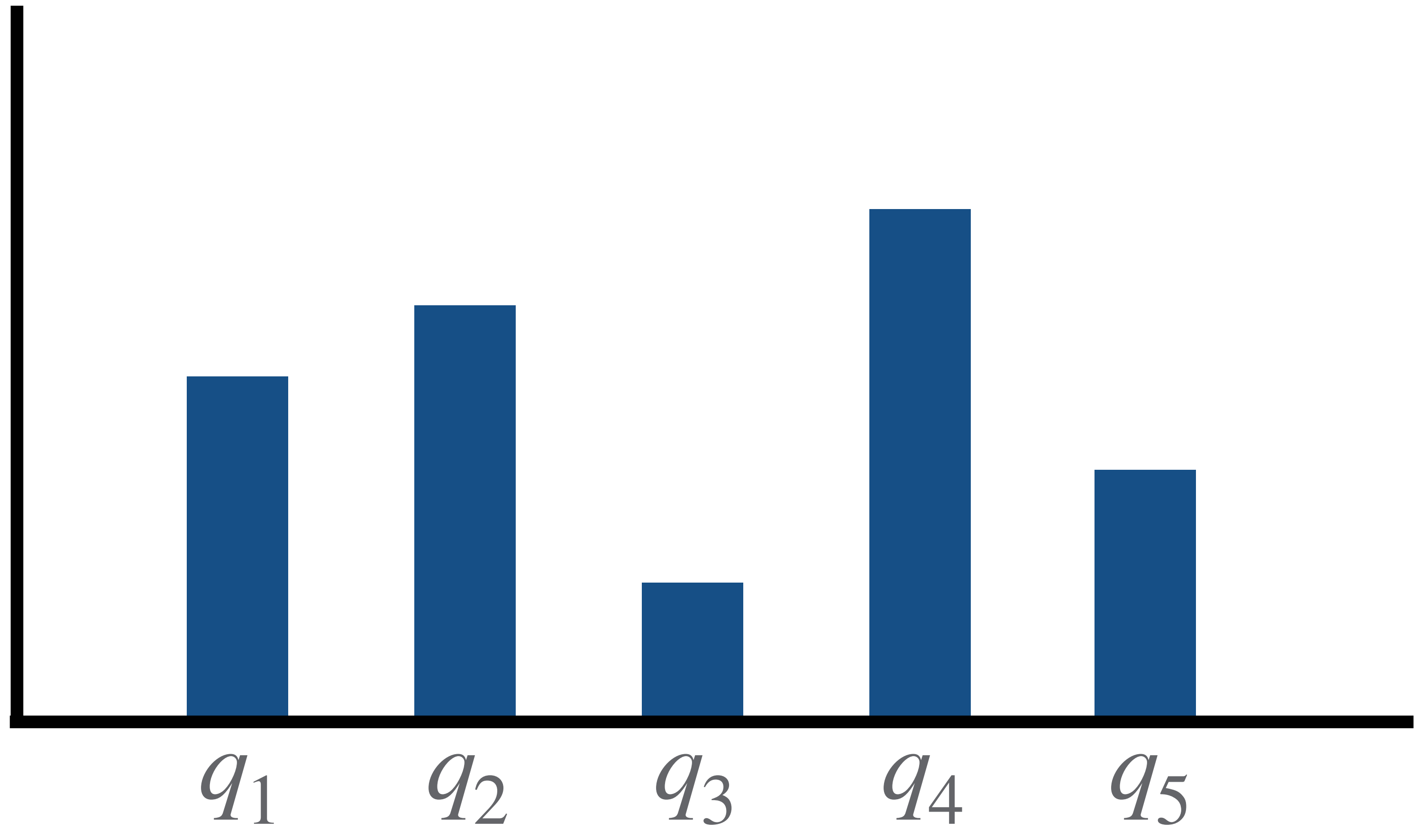
synthesis of privacy-preserving algorithms [ICFP 19]

theme get rid of probability! long live logic!

```
def rnm(q):
  i, best, r = 0
  while i < |q|
   d ~ Lap(q[i], 2/ε)

   if d > best || i = 0
    r = i
    best = d

   i = i + 1
  return r
```

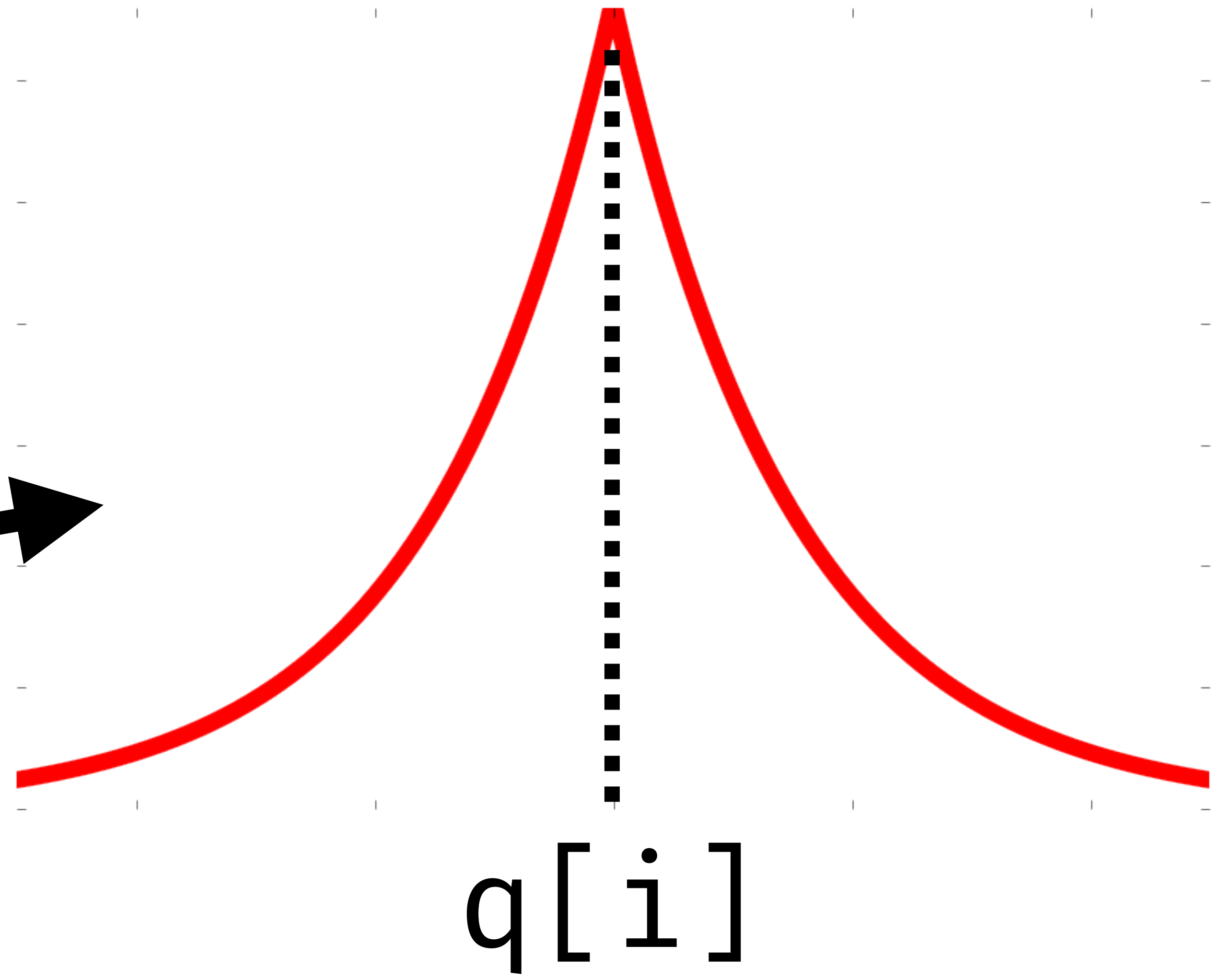
[dwork & roth 14]

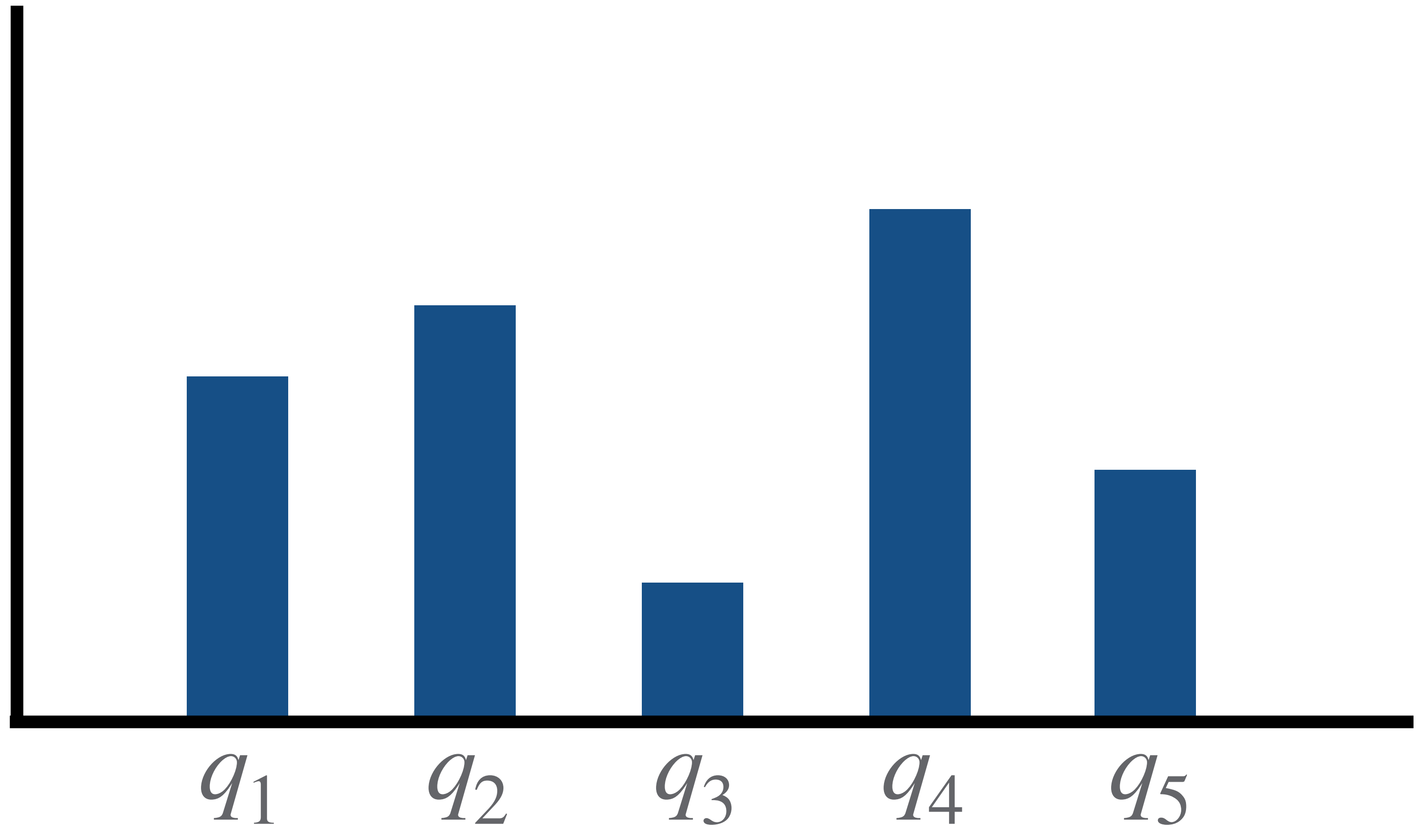
```
def rnm(q):
  i, best, r = 0
  while i < |q|
   d ~ Lap(q[i], 2/ε)

   if d > best || i = 0
    r = i
    best = d

   i = i + 1
  return r
```



[dwork & roth 14]

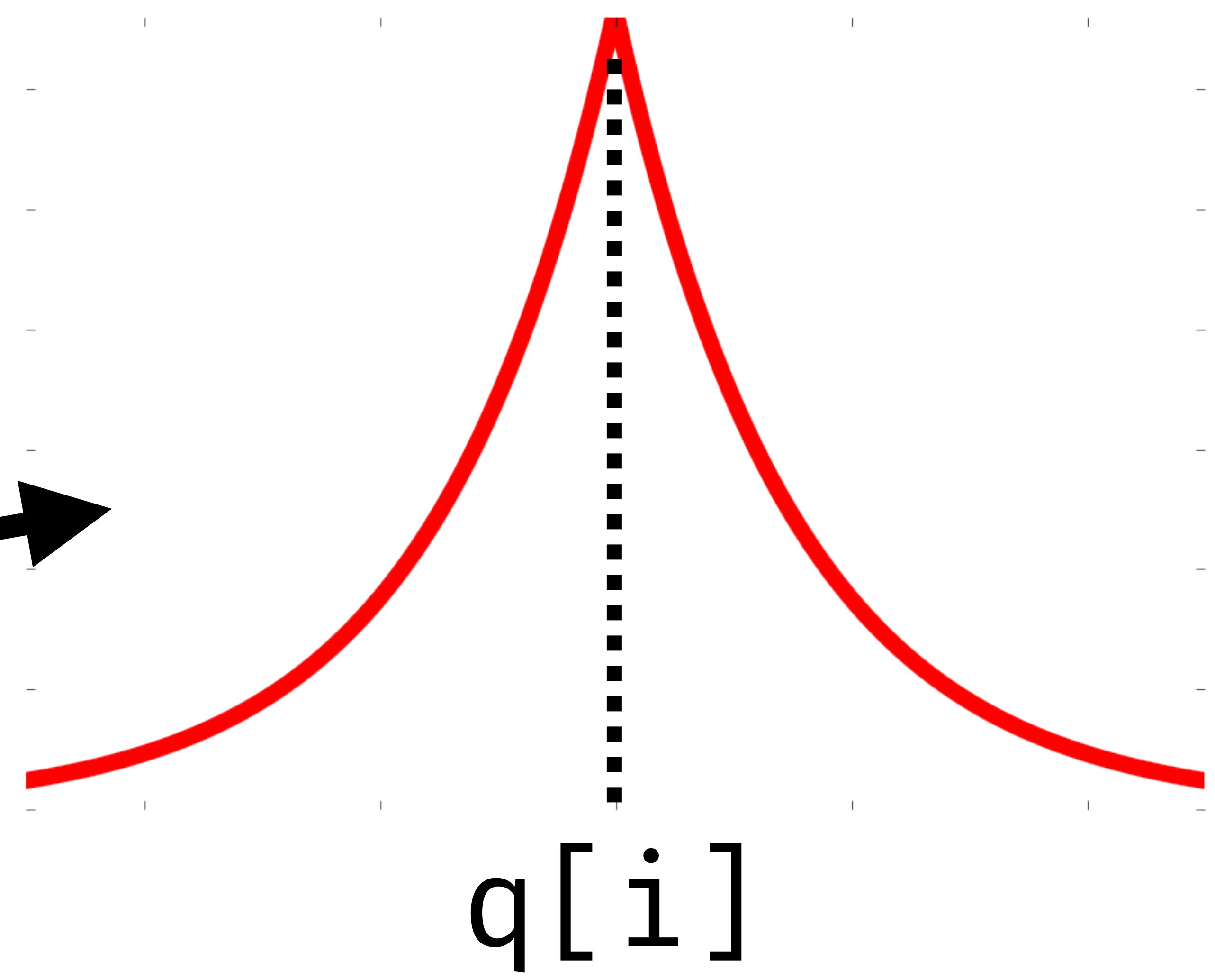
```
def rnm(q):
  i, best, r = 0
  while i < |q|
   d ~ Lap(q[i], 2/ε)

   if d > best || i = 0
    r = i
    best = d

   i = i + 1
  return r
```

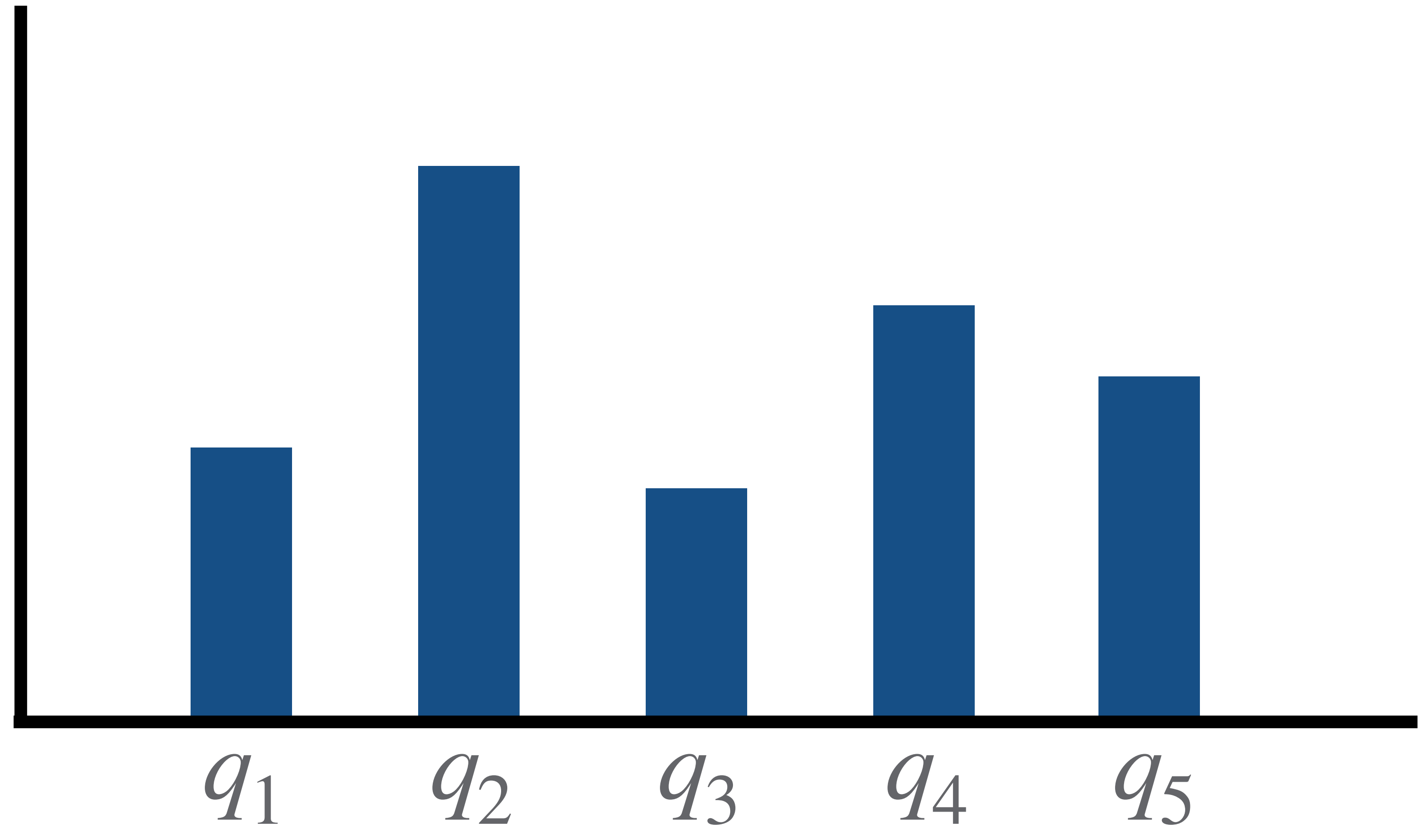




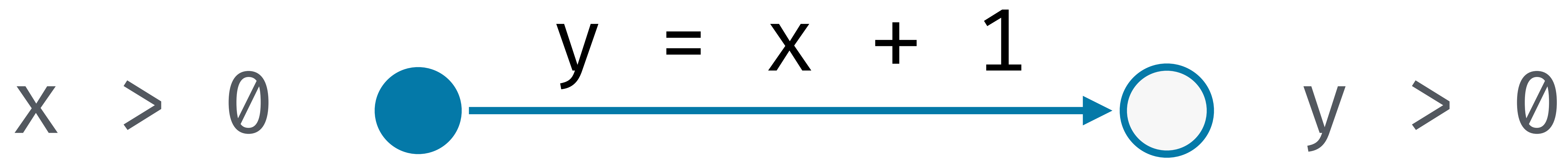
[dwork & roth 14]

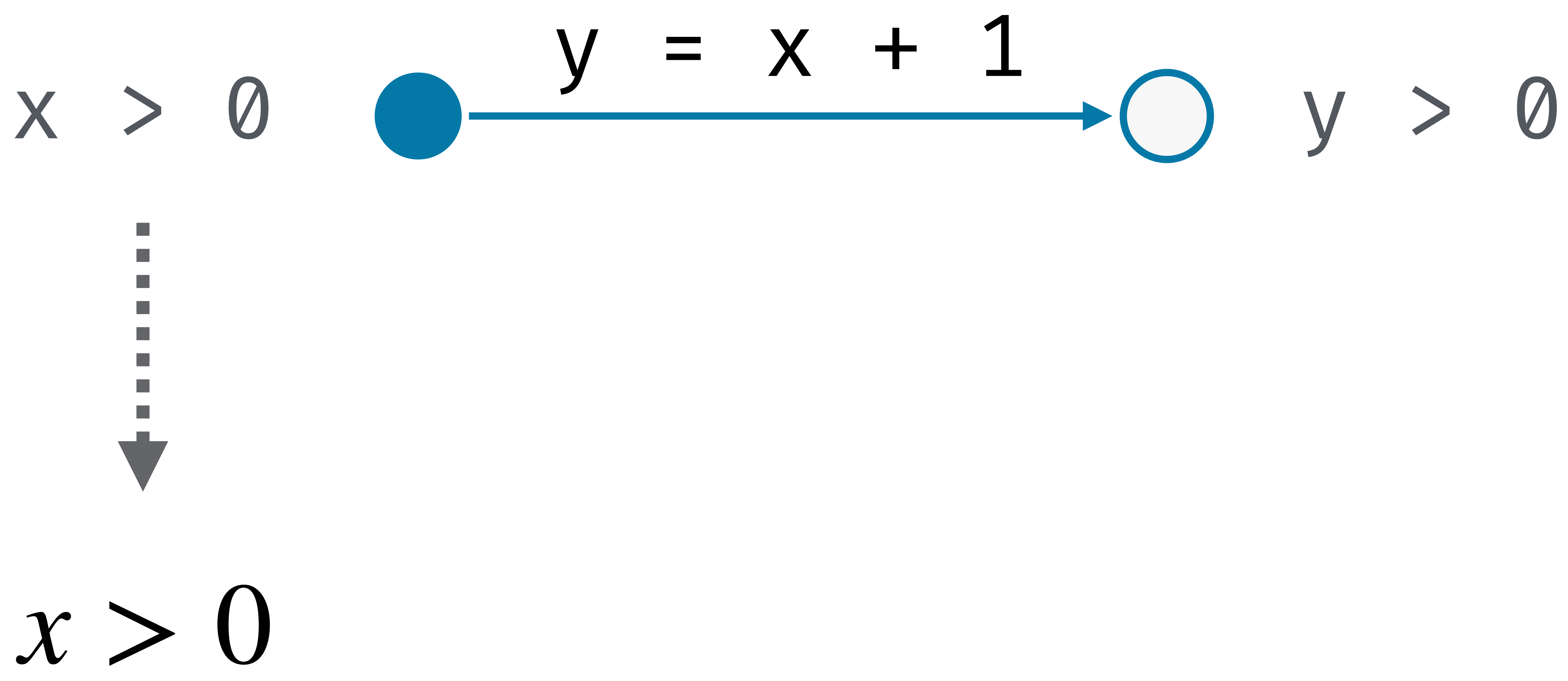
```
def rnm(q):
  i, best, r = 0
  while i < |q|
    d ~ Lap(q[i], 2/ε)

    if d > best || i = 0
      r = i
      best = d

    i = i + 1
  return r
```

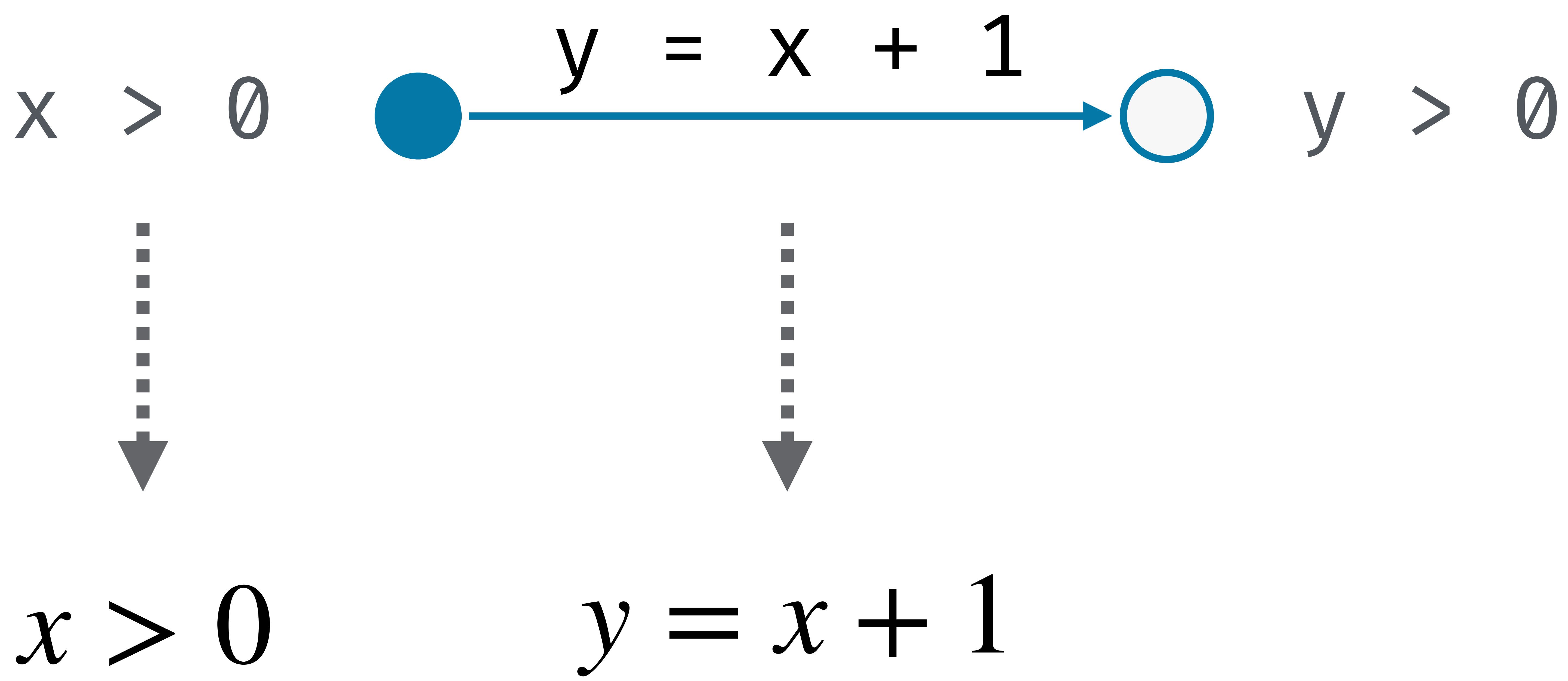


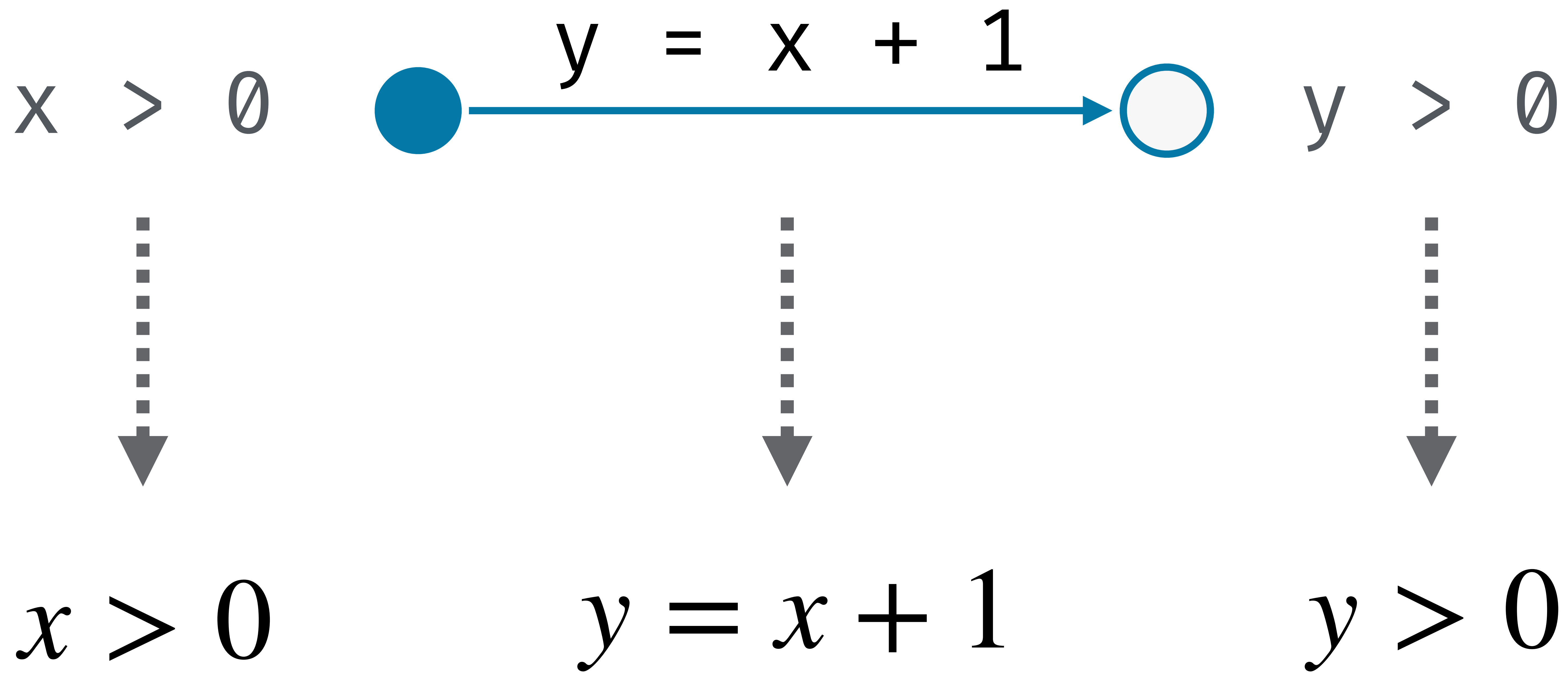


[dwork & roth 14]

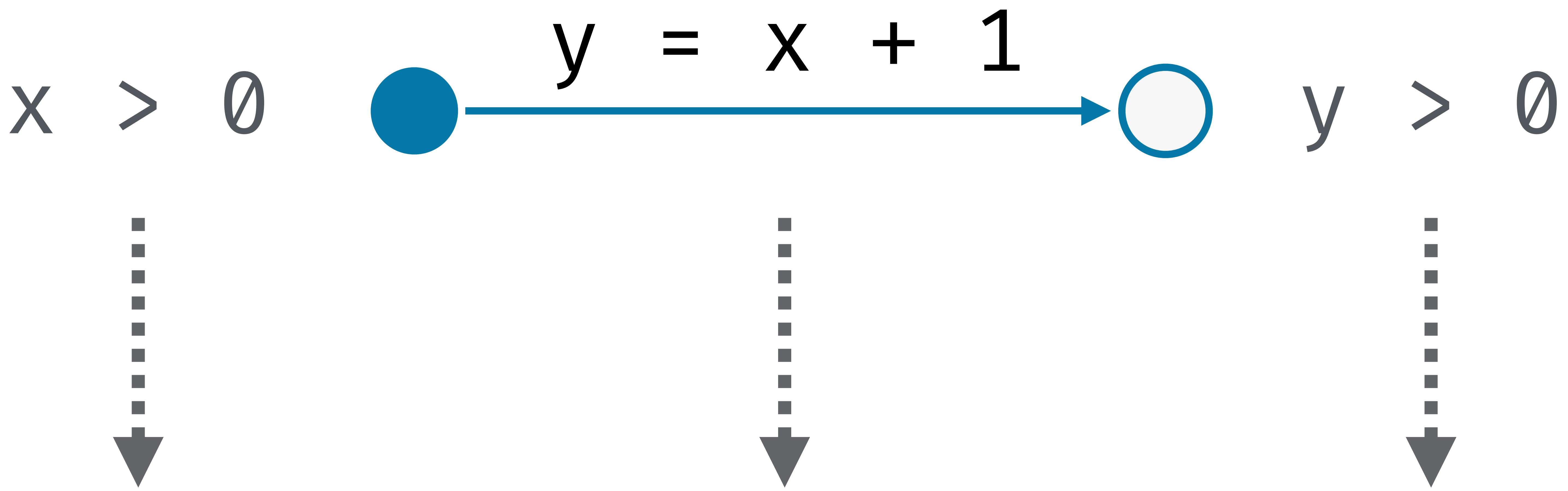

x > 0
y = x + 1
y > 0


y = x + 1
x > 0
y > 0
$x > 0$

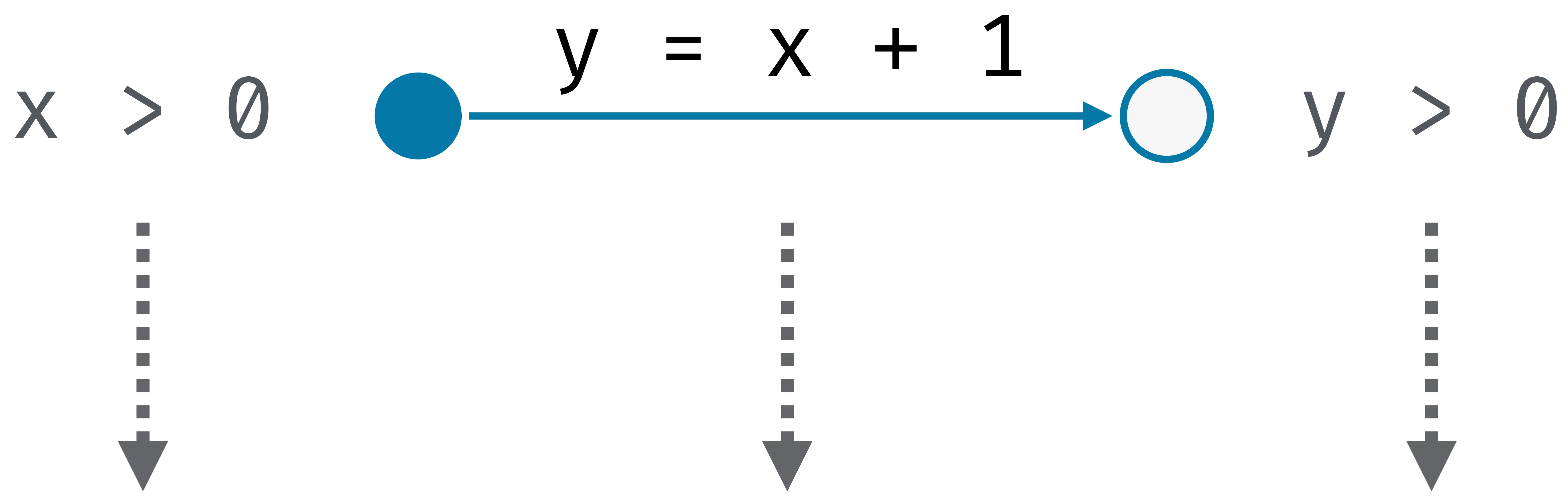

x > 0
y = x + 1
y > 0
x > 0
y = x + 1



x > 0
y = x + 1
y > 0
x > 0
y = x + 1
y > 0

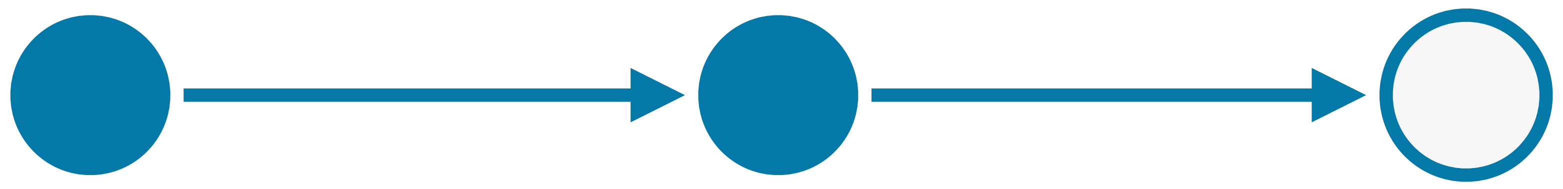


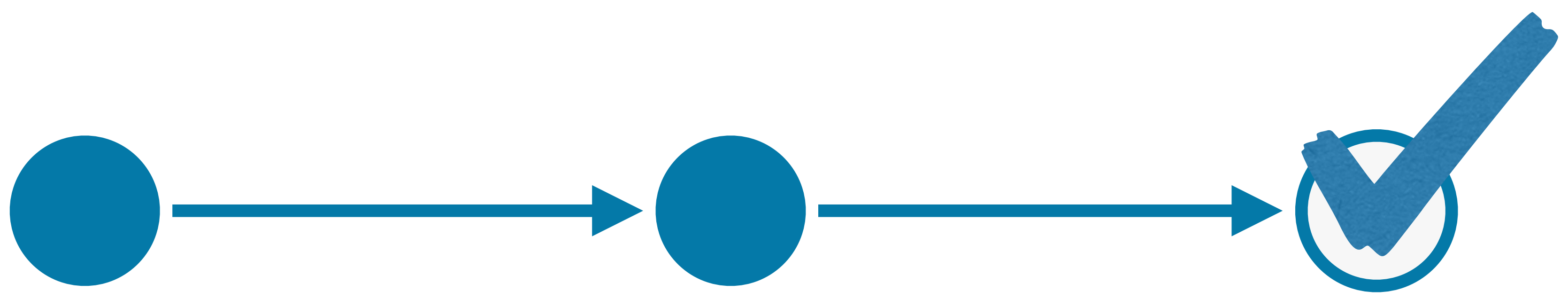
$$x > 0 \quad \wedge \quad y = x + 1 \quad \Rightarrow \quad y > 0$$

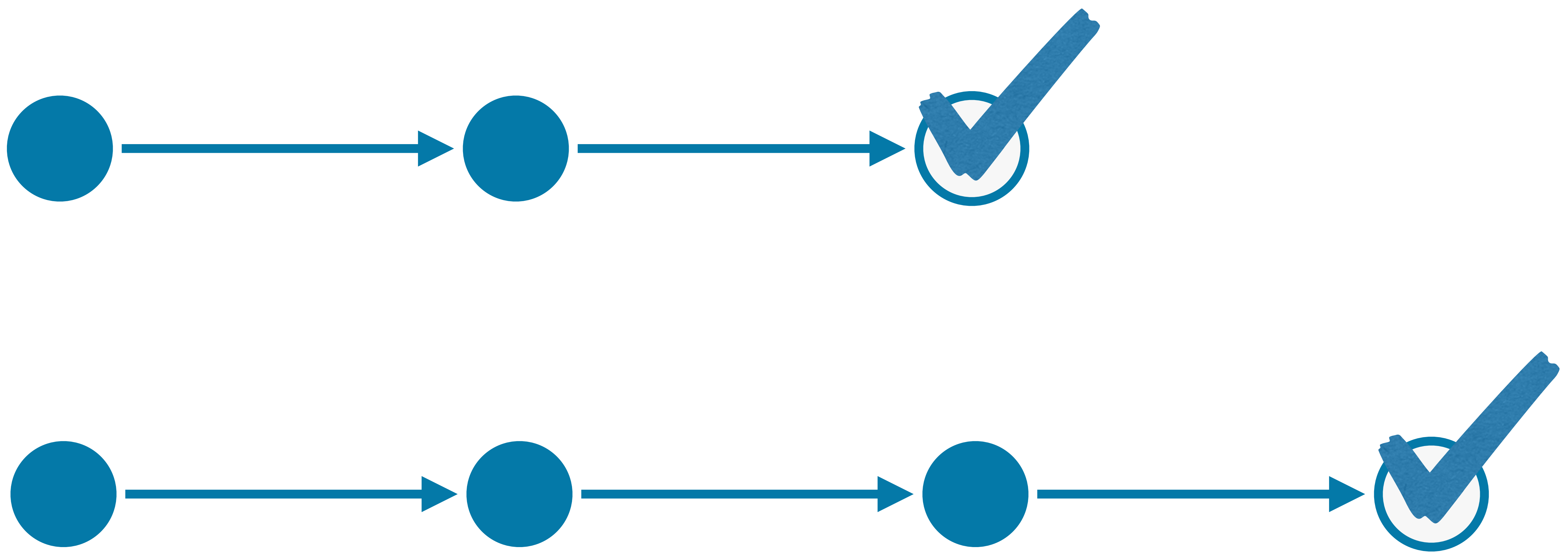


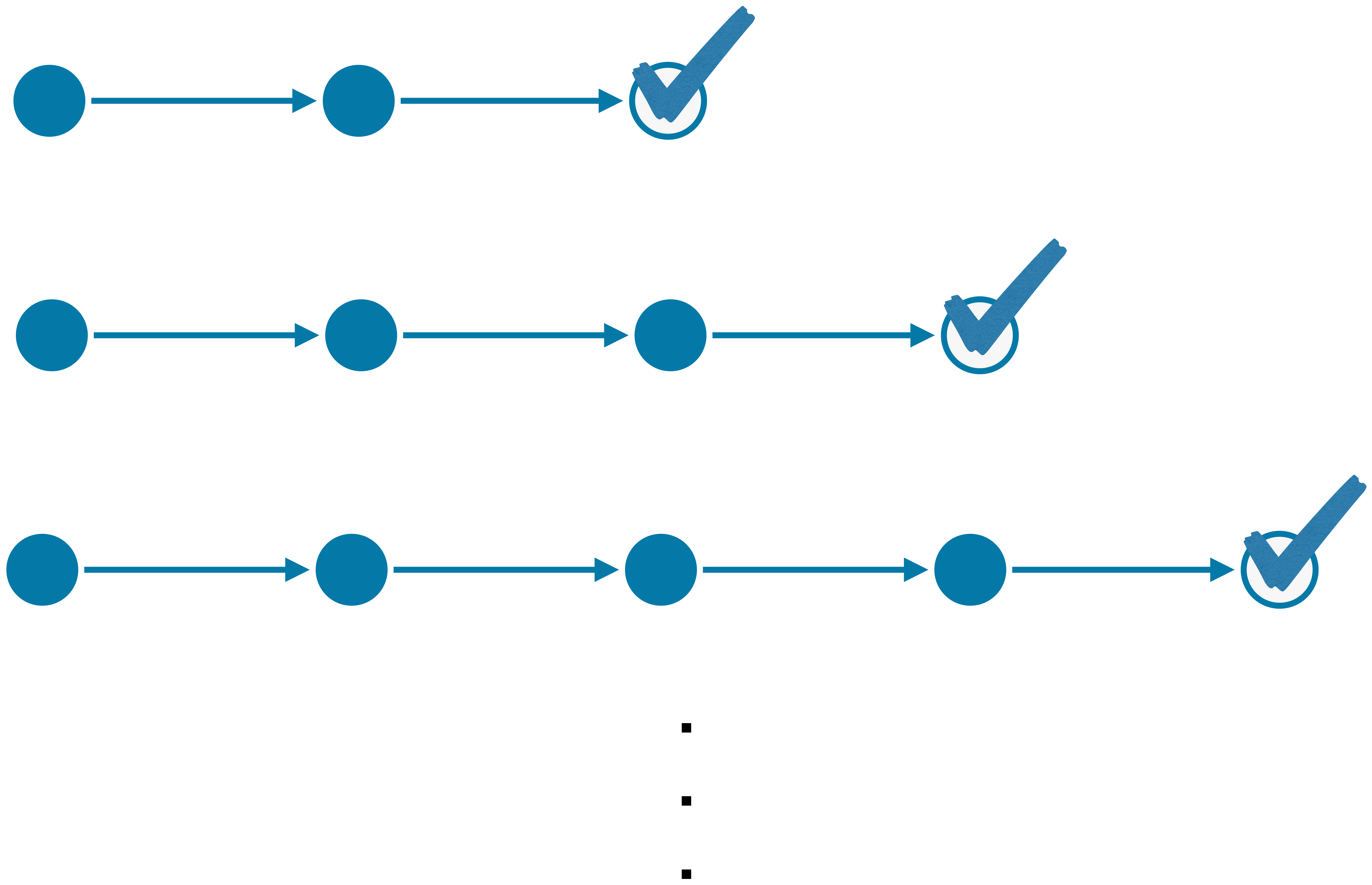
$$x > 0 \;\wedge\; y = x + 1 \;\Rightarrow\; y > 0$$

solve with a **SAT/SMT** solver

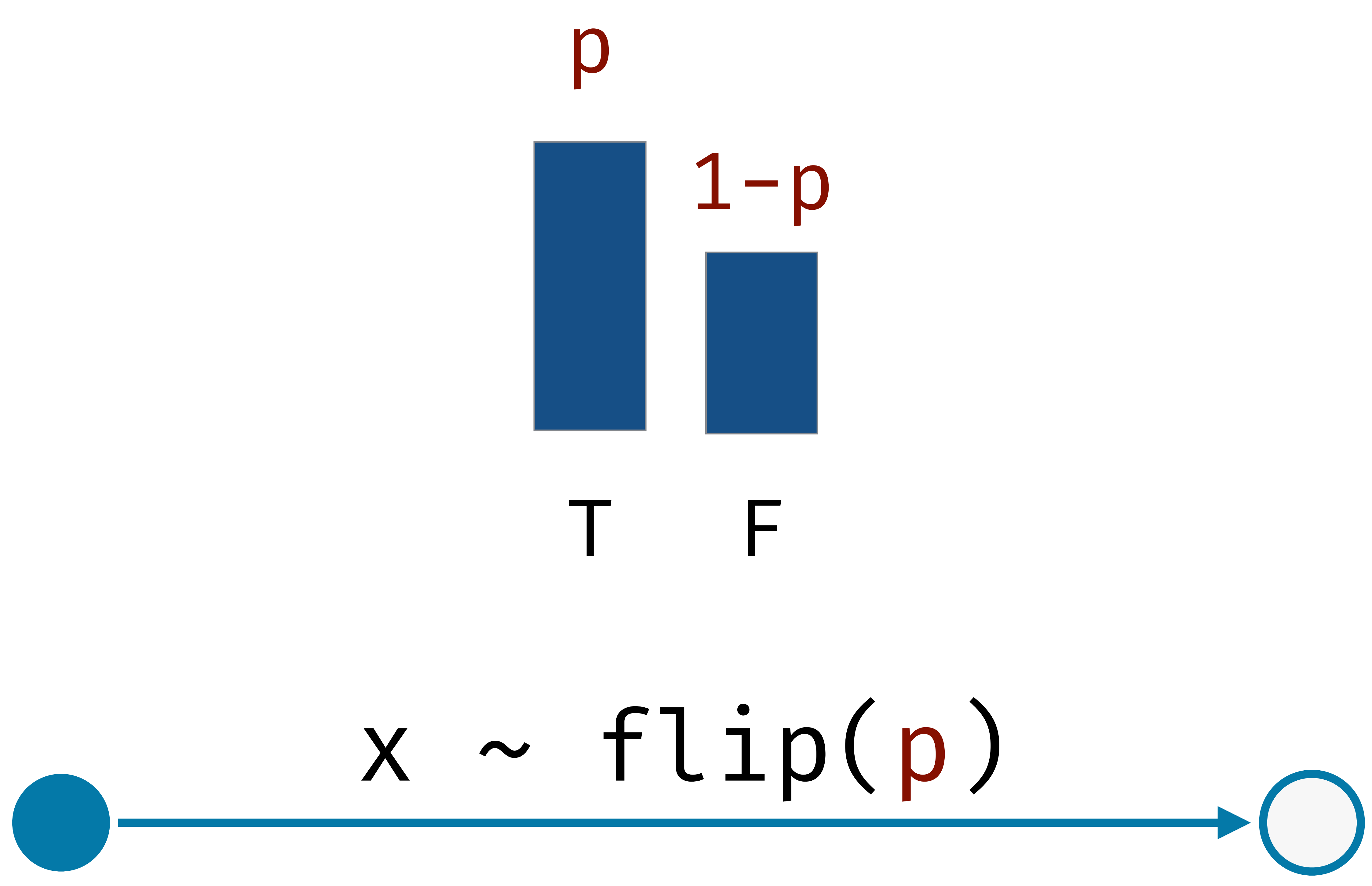


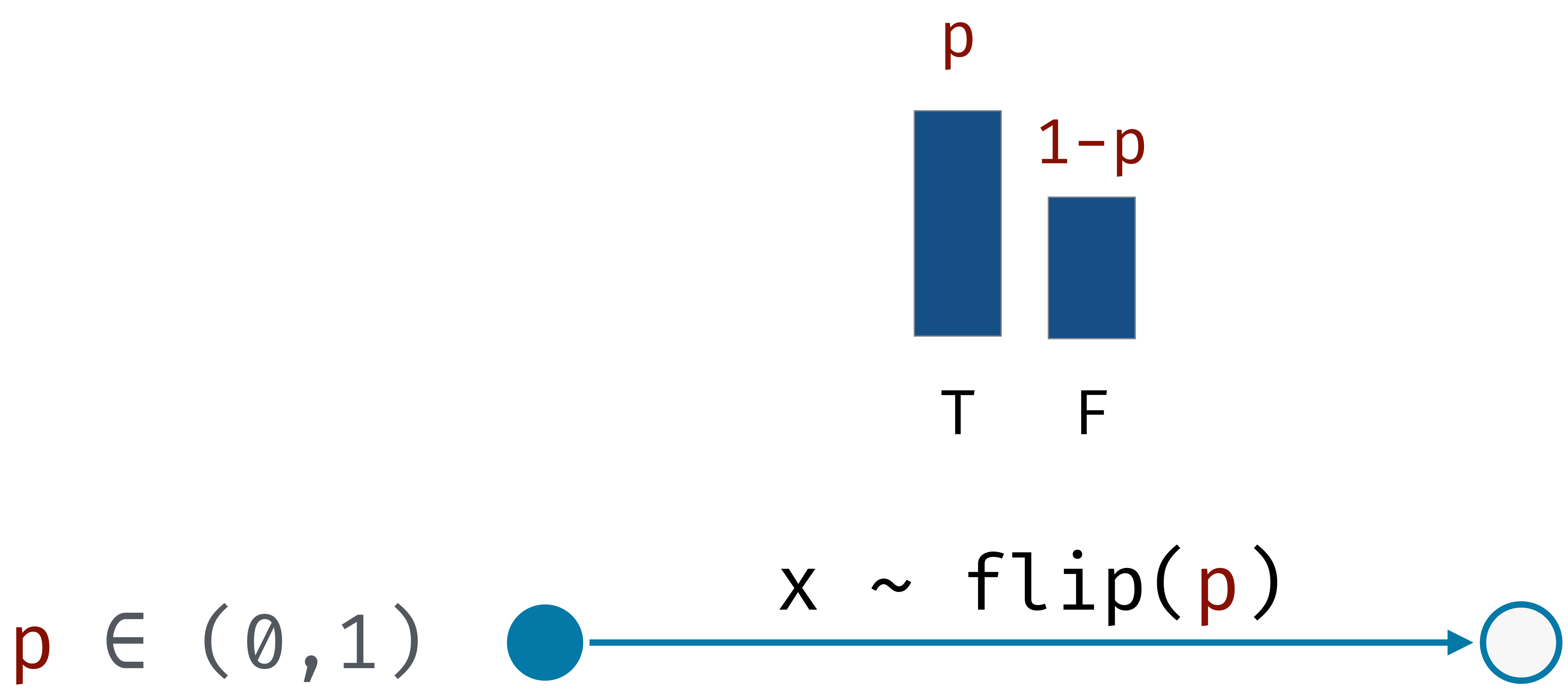


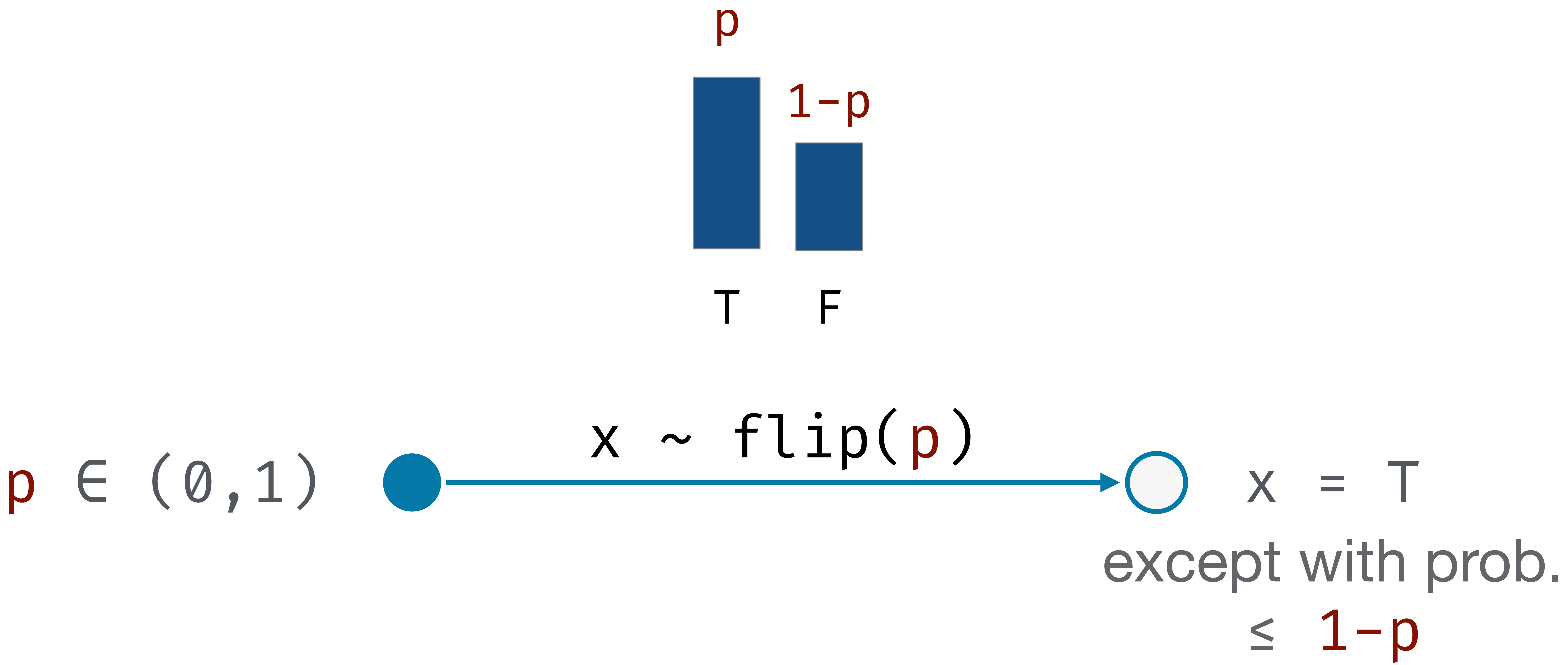


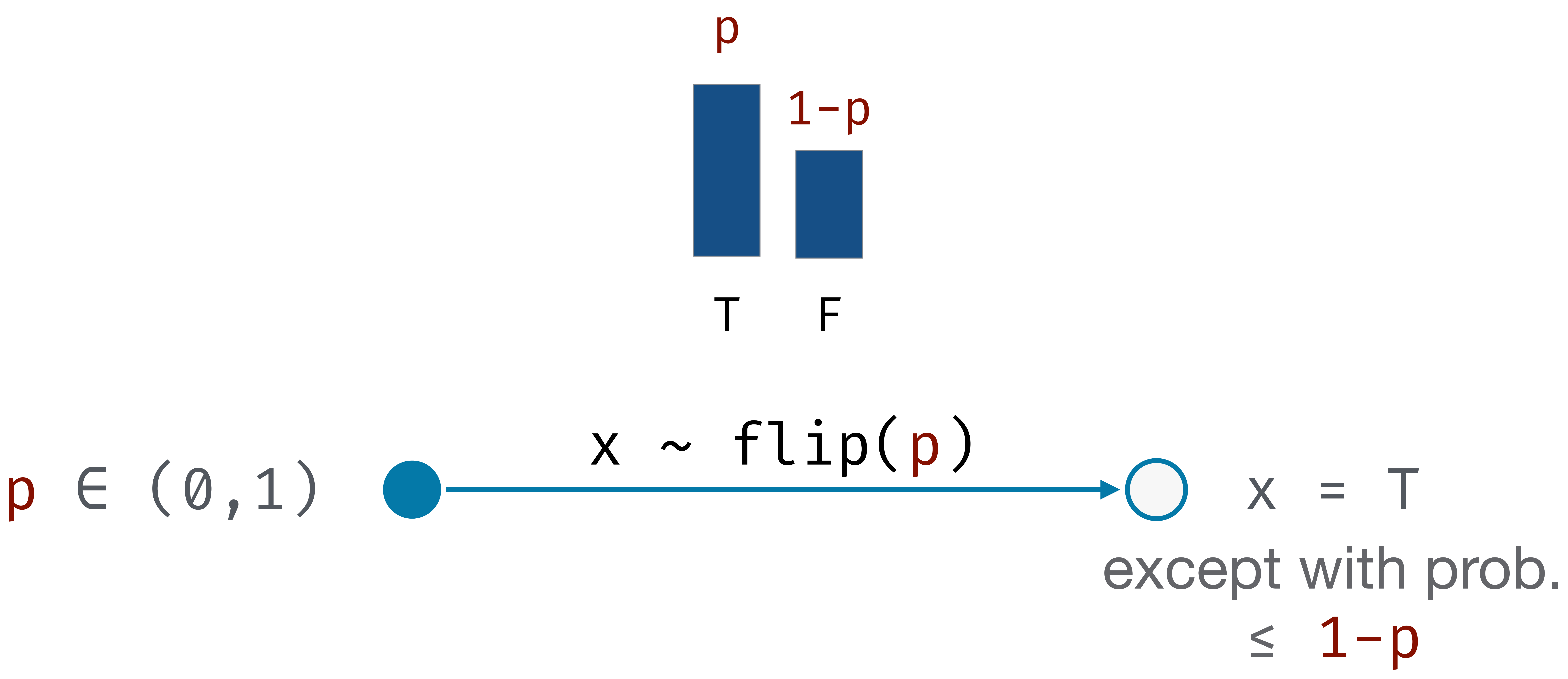



x ~ flip(p)

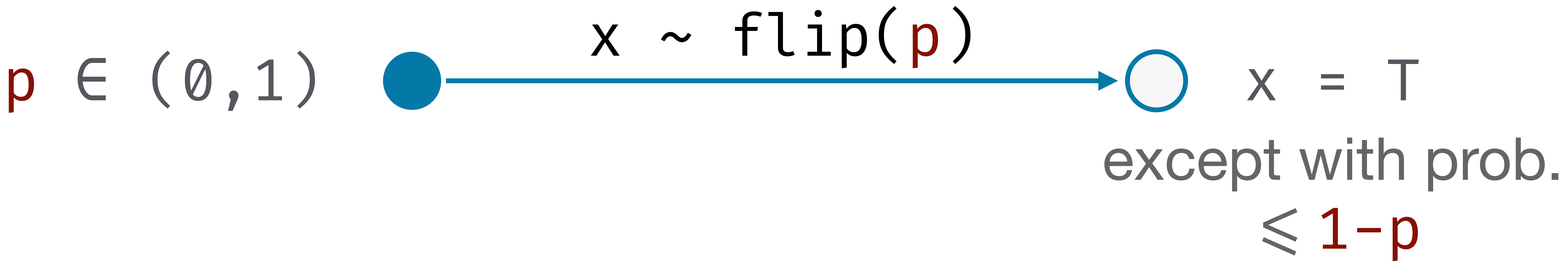

p
1-p
T
F
x ~ flip(p)

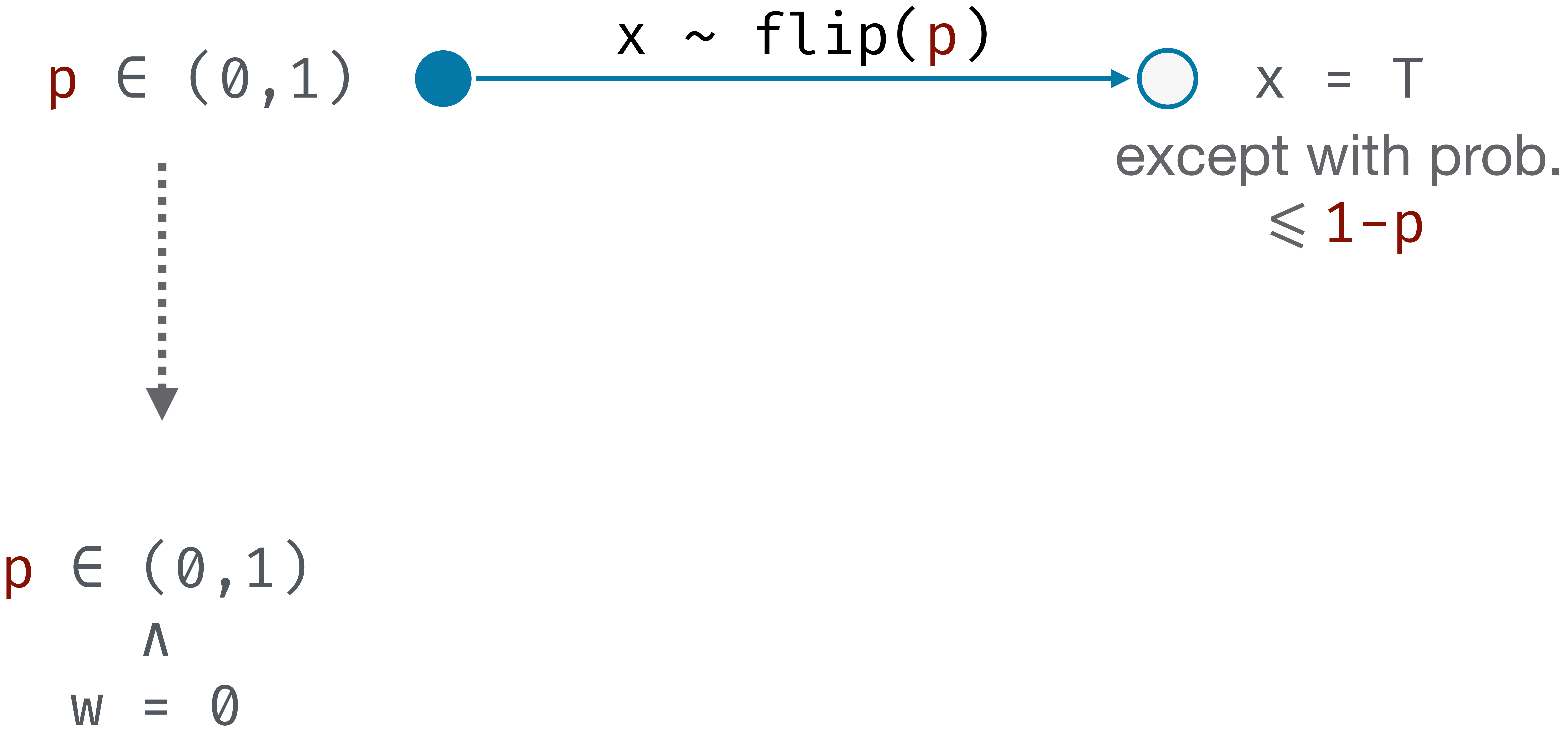

p
1-p
T
F
x ~ flip(p)
p ∈ (0,1)


p
1-p
T
F
x ~ flip(p)
p ∈ (0,1)
x = T
except with prob.
≤ 1-p

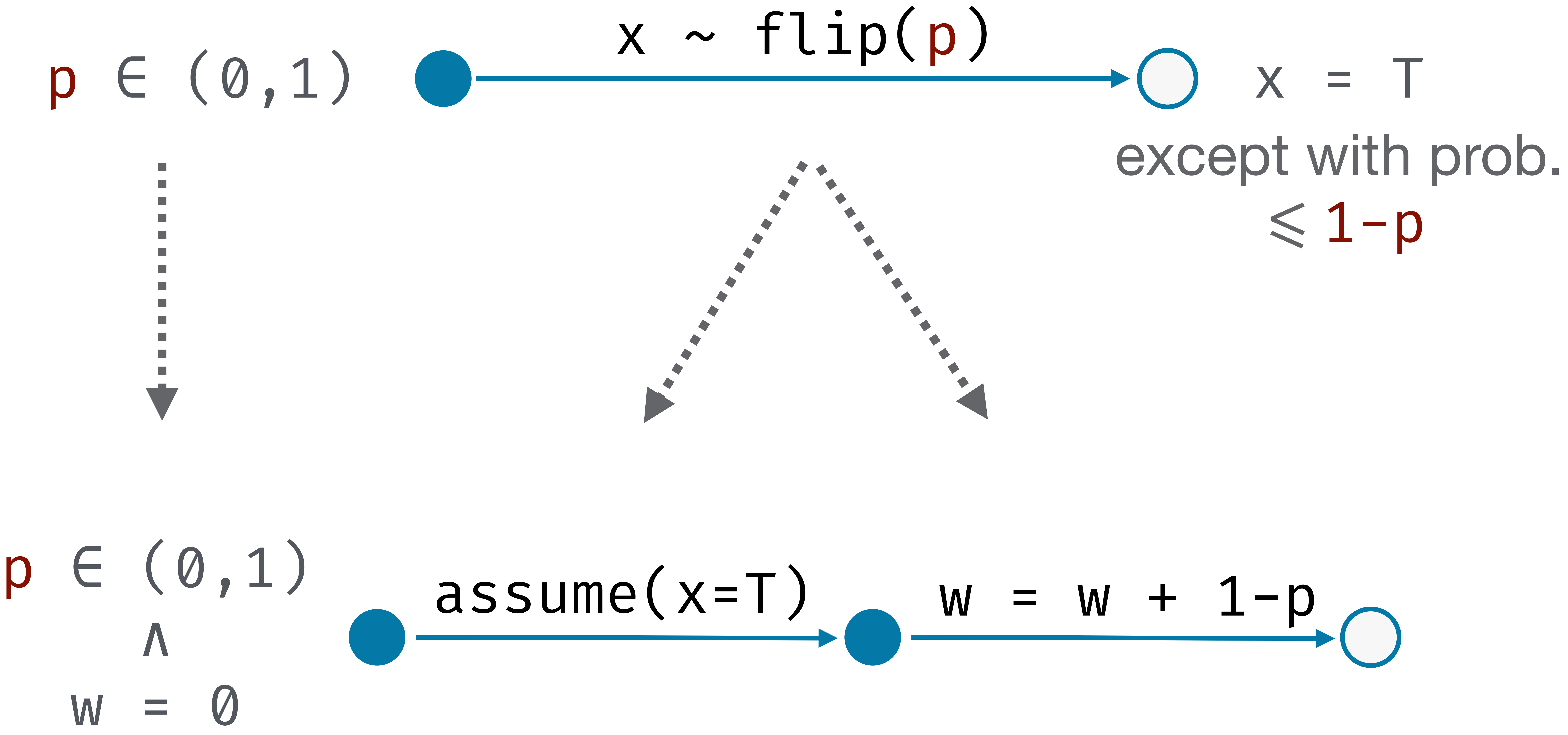


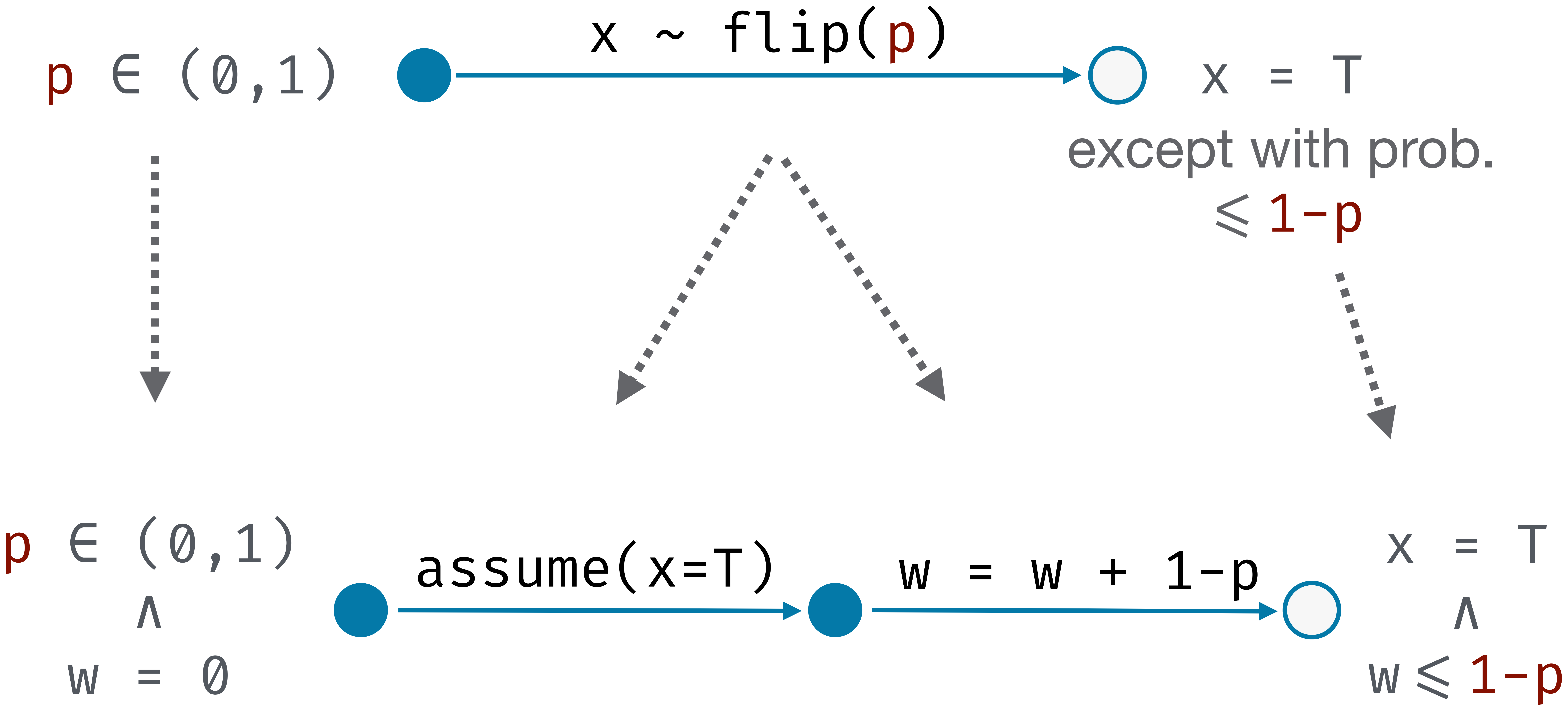
challenge how do we check this with first-order logic?

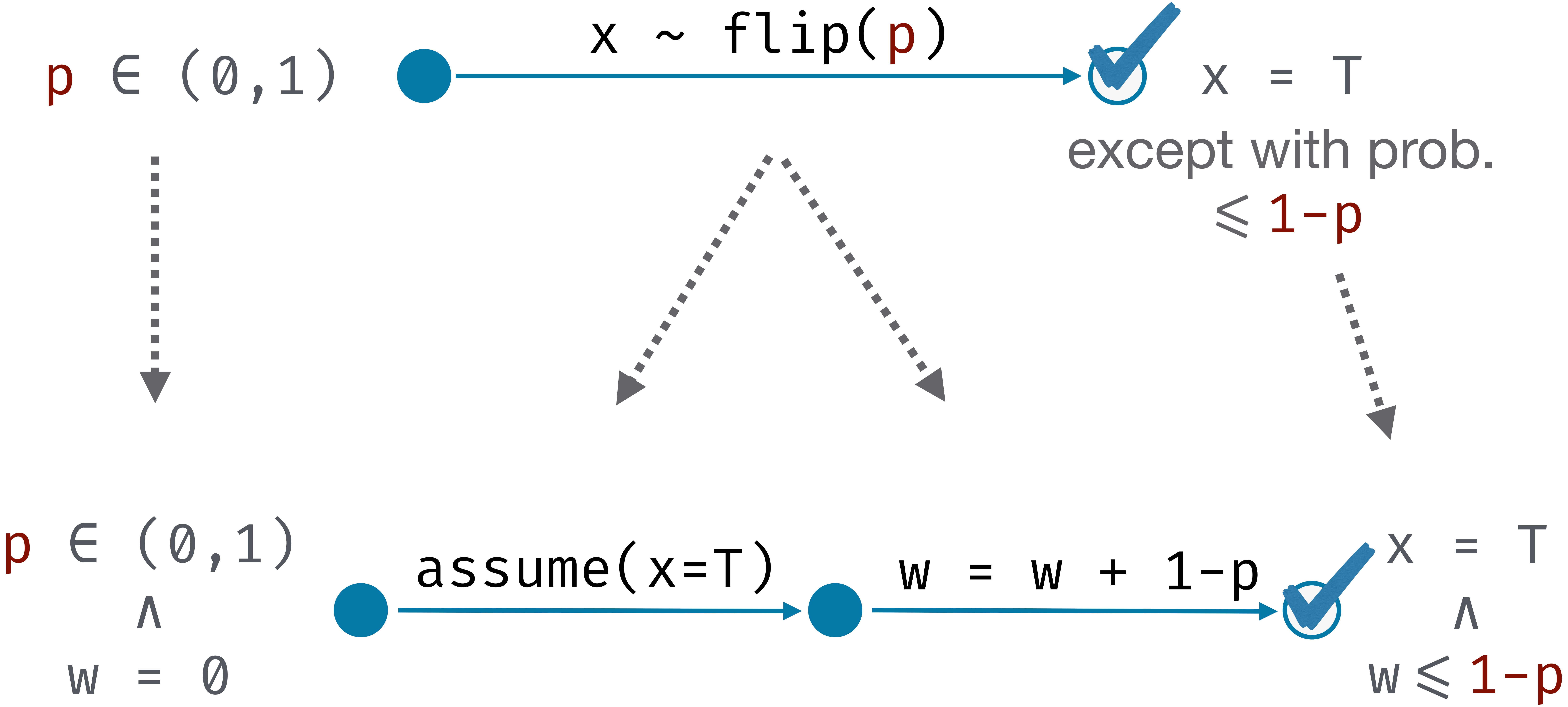

p ∈ (0,1)
x ~ flip(p)
x = T
except with prob.
⩽ 1-p



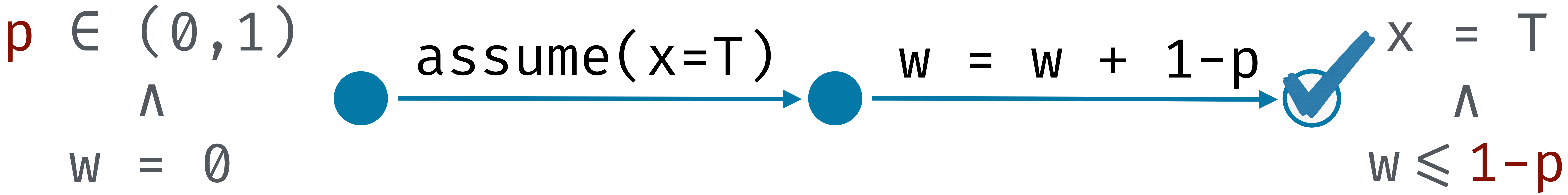
p ∈ (0,1)
x ~ flip(p)
x = T
except with prob.
⩽ 1-p
p ∈ (0,1)
∧
w = 0


p ∈ (0,1)
x ~ flip(p)
x = T
except with prob.
⩽ 1-p
p ∈ (0,1)
∧
w = 0
assume(x=T)
w = w + 1-p

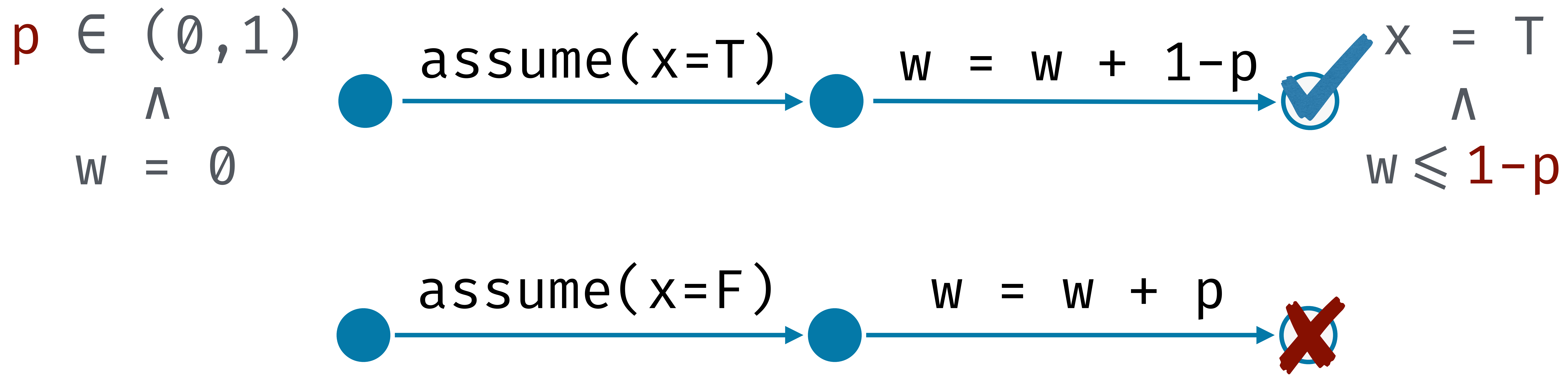

p ∈ (0,1)
x ~ flip(p)
x = T
except with prob.
⩽ 1-p
p ∈ (0,1)
∧
w = 0
assume(x=T)
w = w + 1-p
x = T
∧
w ⩽ 1-p



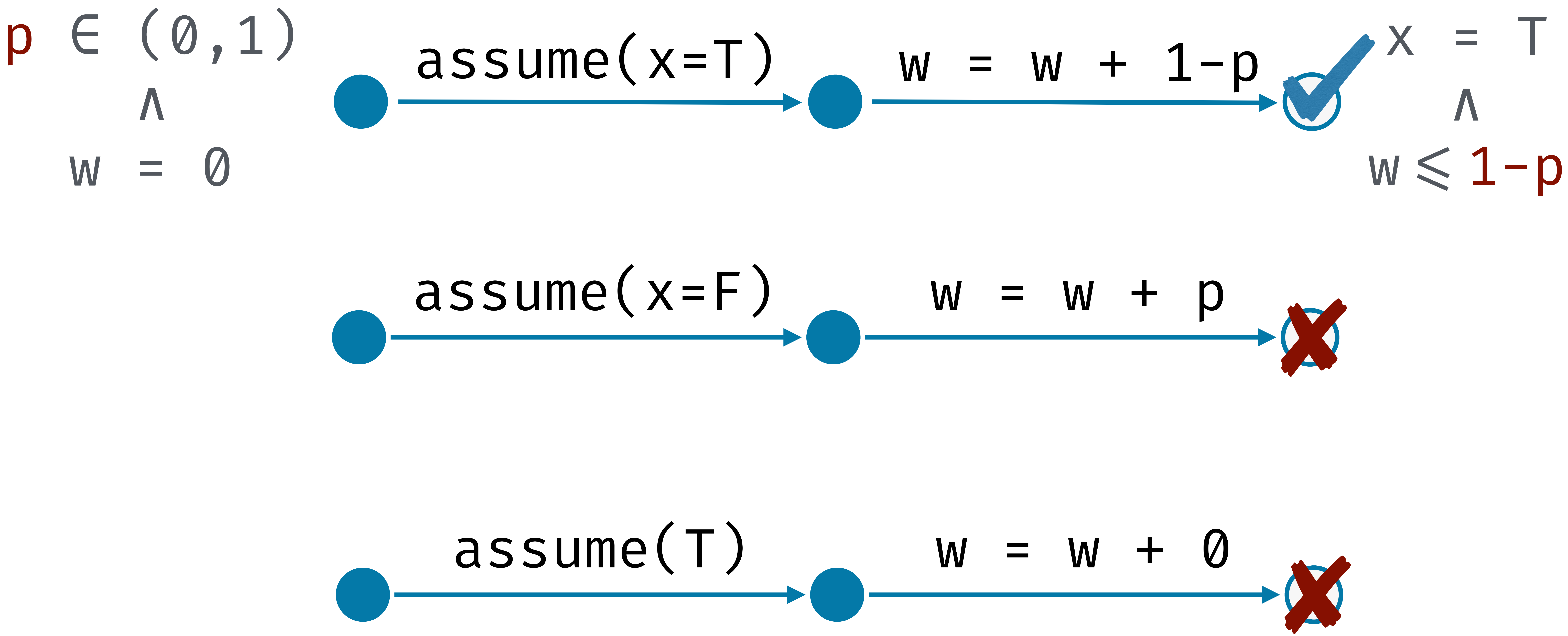
p ∈ (0,1)
x ~ flip(p)
x = T
except with prob.
⩽ 1-p
p ∈ (0,1)
∧
w = 0
assume(x=T)
w = w + 1-p
x = T
∧
w ⩽ 1-p



challenge many different axiomatizations

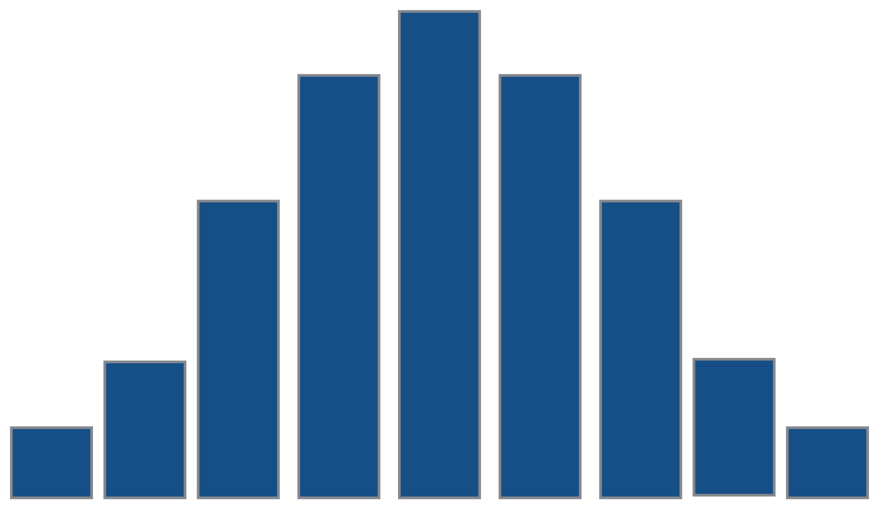


challenge many different axiomatizations



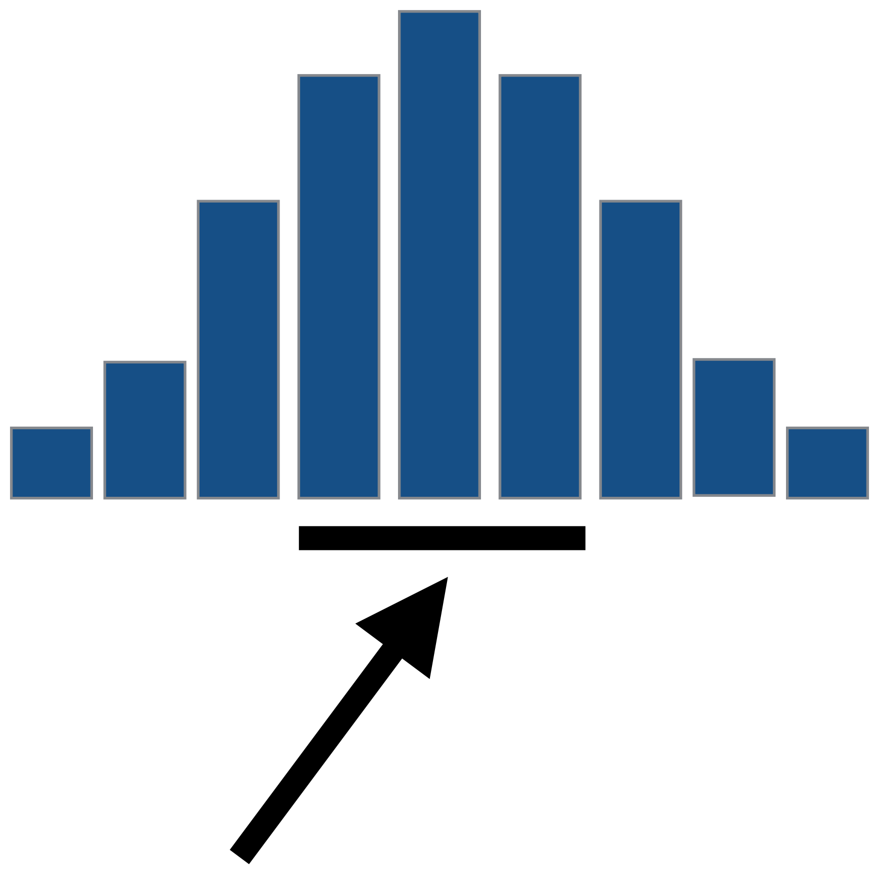
challenge many different axiomatizations

```
x ~ dist
```



challenge many different axiomatizations

```
x ~ dist
```

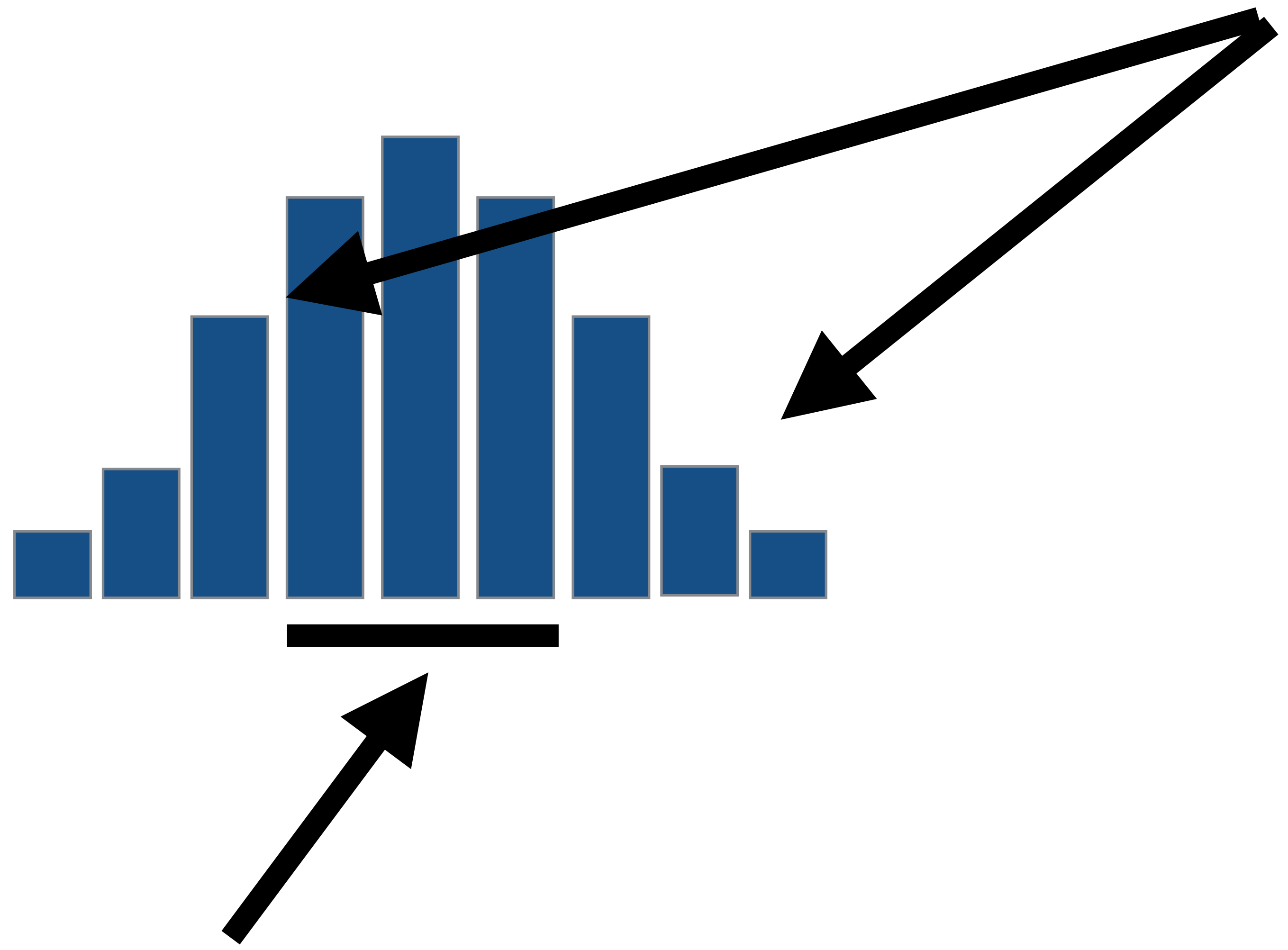


assume **x** is one of those 3 values

challenge many different axiomatizations

`x ~ dist`

failure probability is

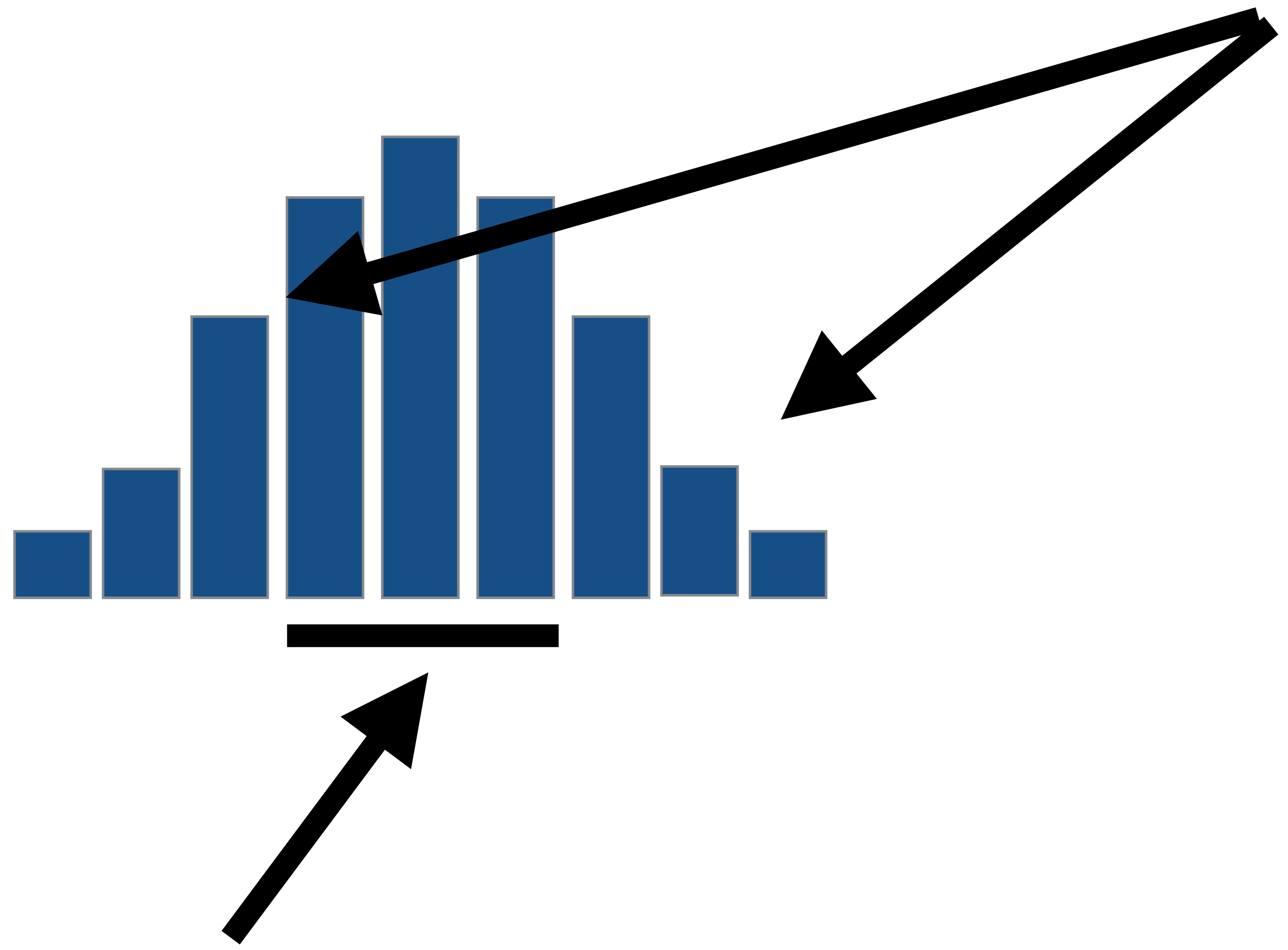


assume **x** is one of those 3 values

challenge many different axiomatizations

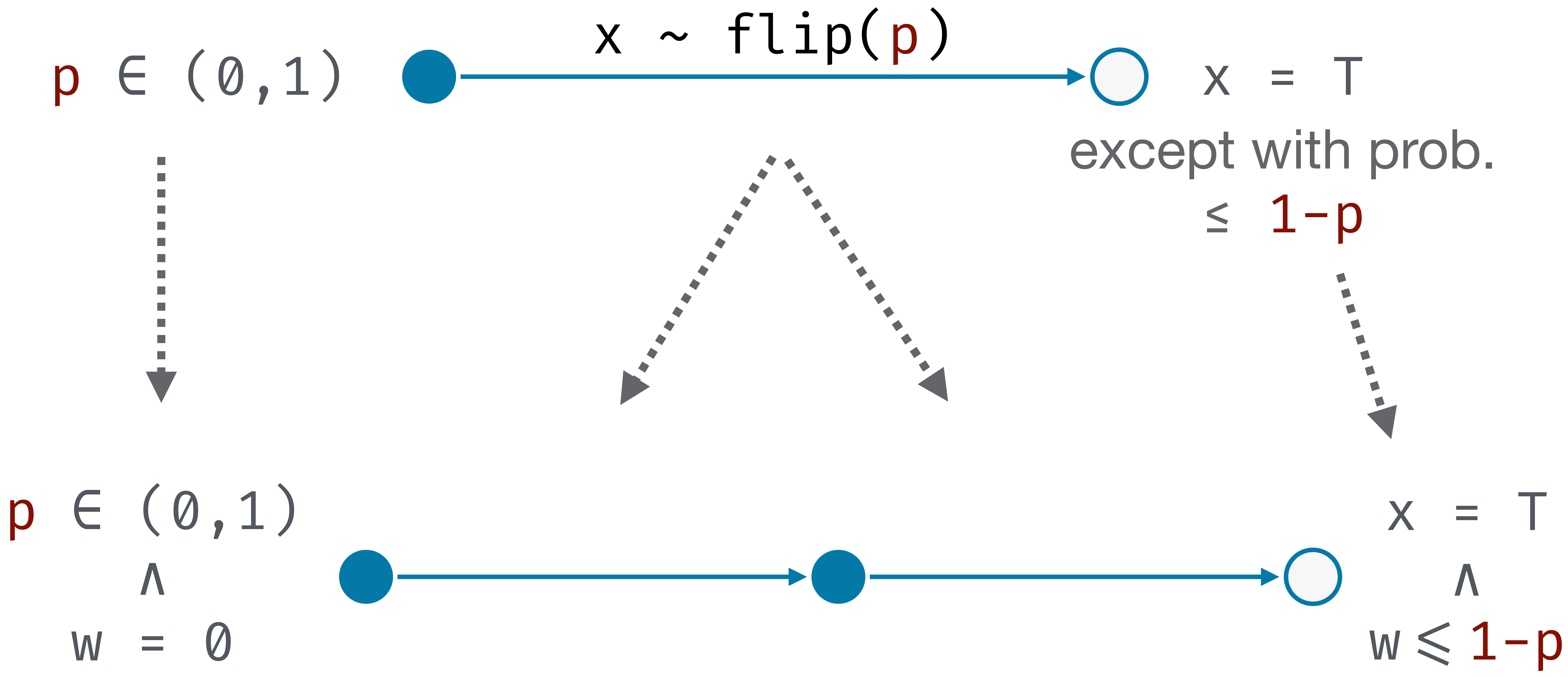
`x ~ dist`

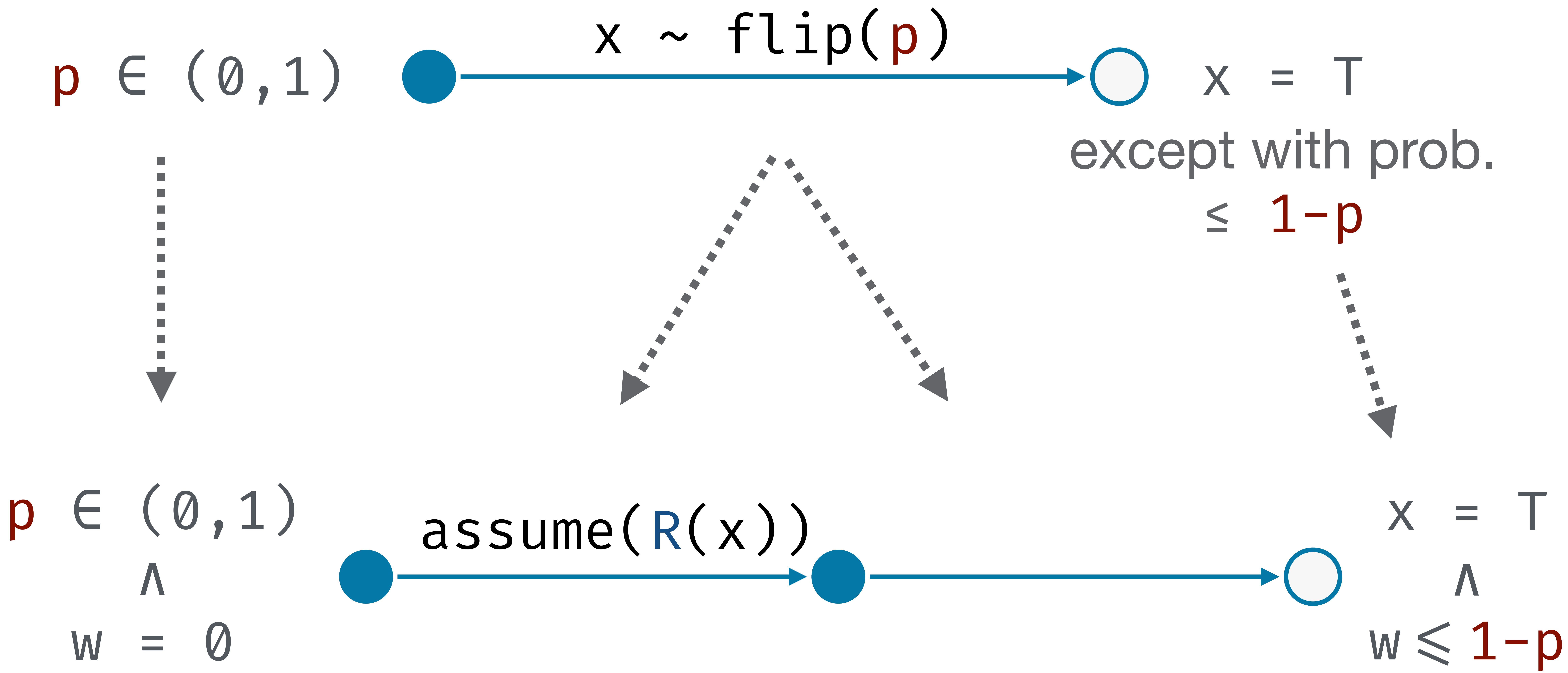
failure probability is

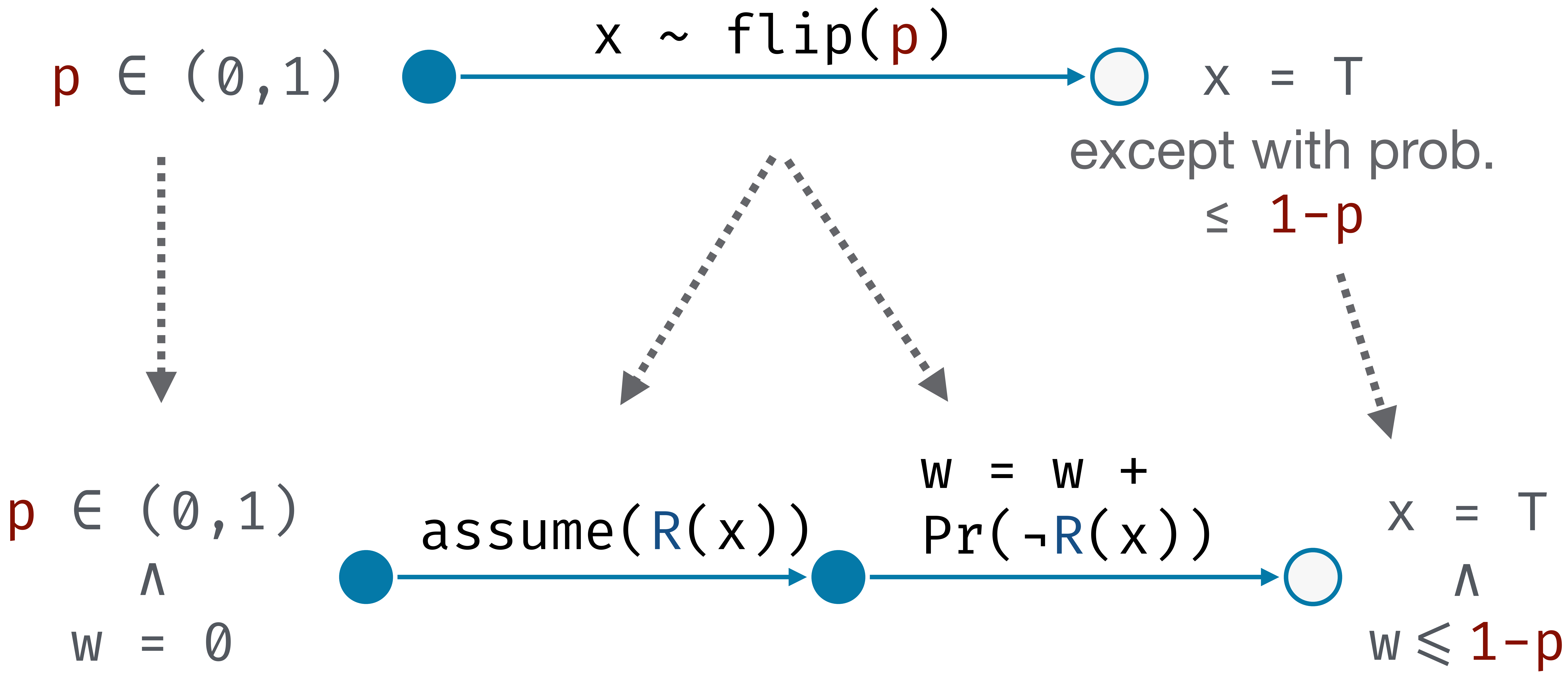


assume **x** is one of those 3 values

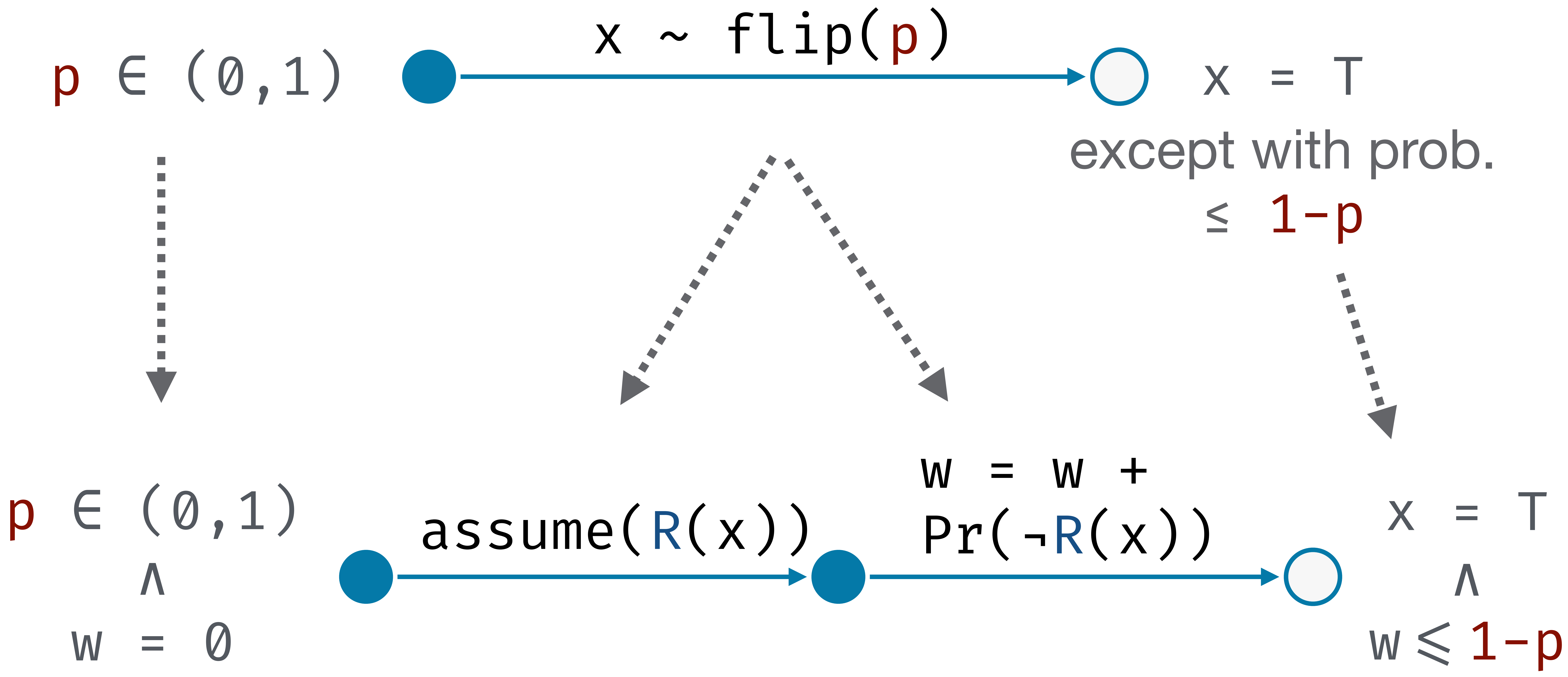
challenge many different axiomatizations


x ~ flip(p)
p ∈ (0,1)
x = T
except with prob.
≤ 1-p
p ∈ (0,1)
∧
w = 0
x = T
∧
w ≤ 1-p

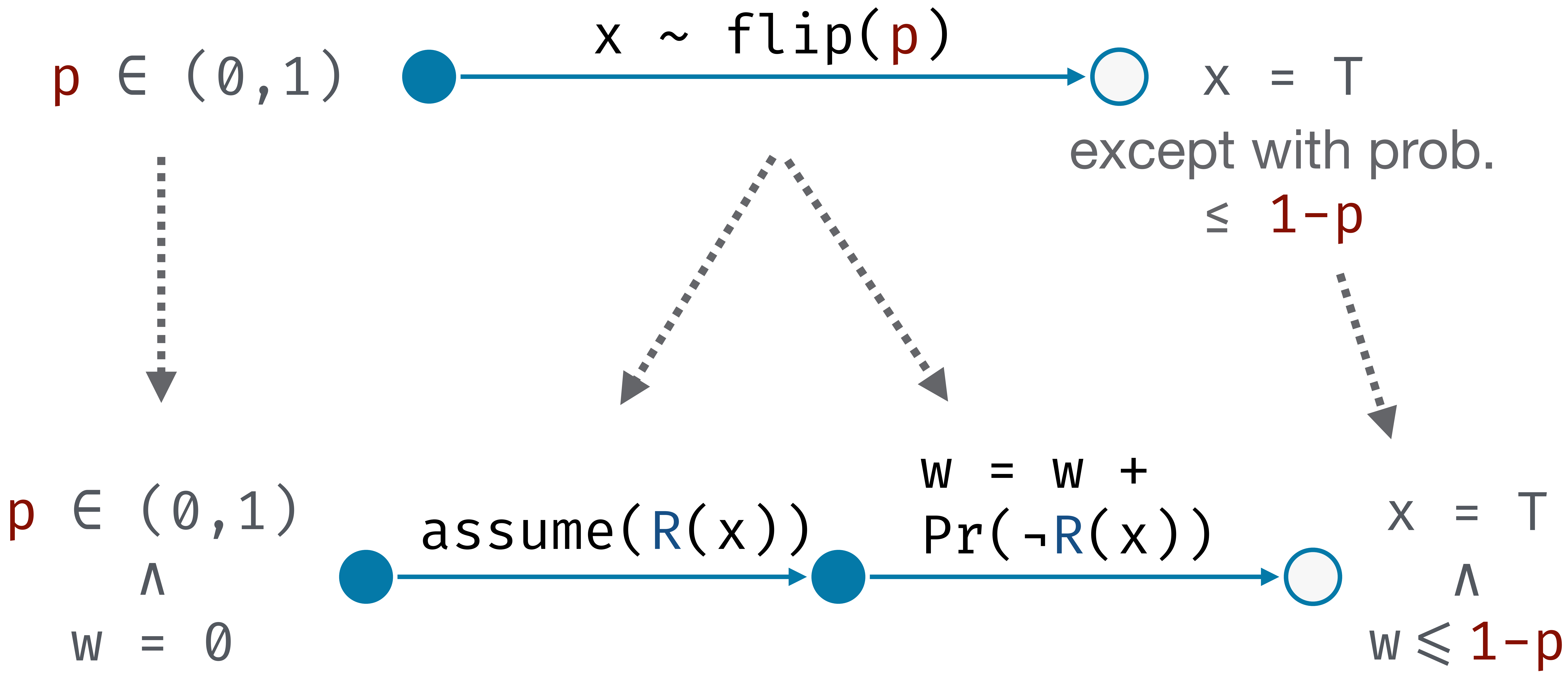

p ∈ (0,1)
x ~ flip(p)
x = T
except with prob.
≤ 1-p
p ∈ (0,1)
∧
w = 0
assume(R(x))
x = T
∧
w ⩽ 1-p



p ∈ (0,1)
x ~ flip(p)
x = T
except with prob.
≤ 1-p
p ∈ (0,1)
∧
w = 0
assume(R(x))
w = w +
Pr(¬R(x))
x = T
∧
w ≤ 1-p

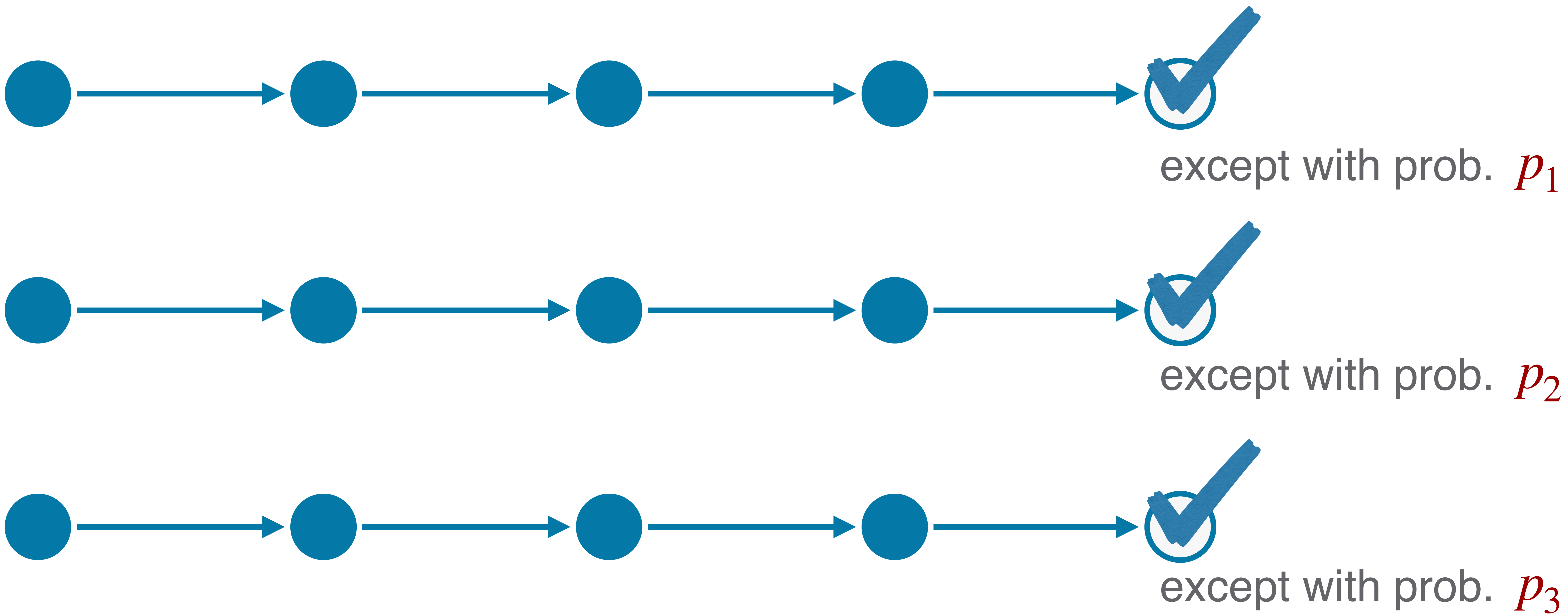


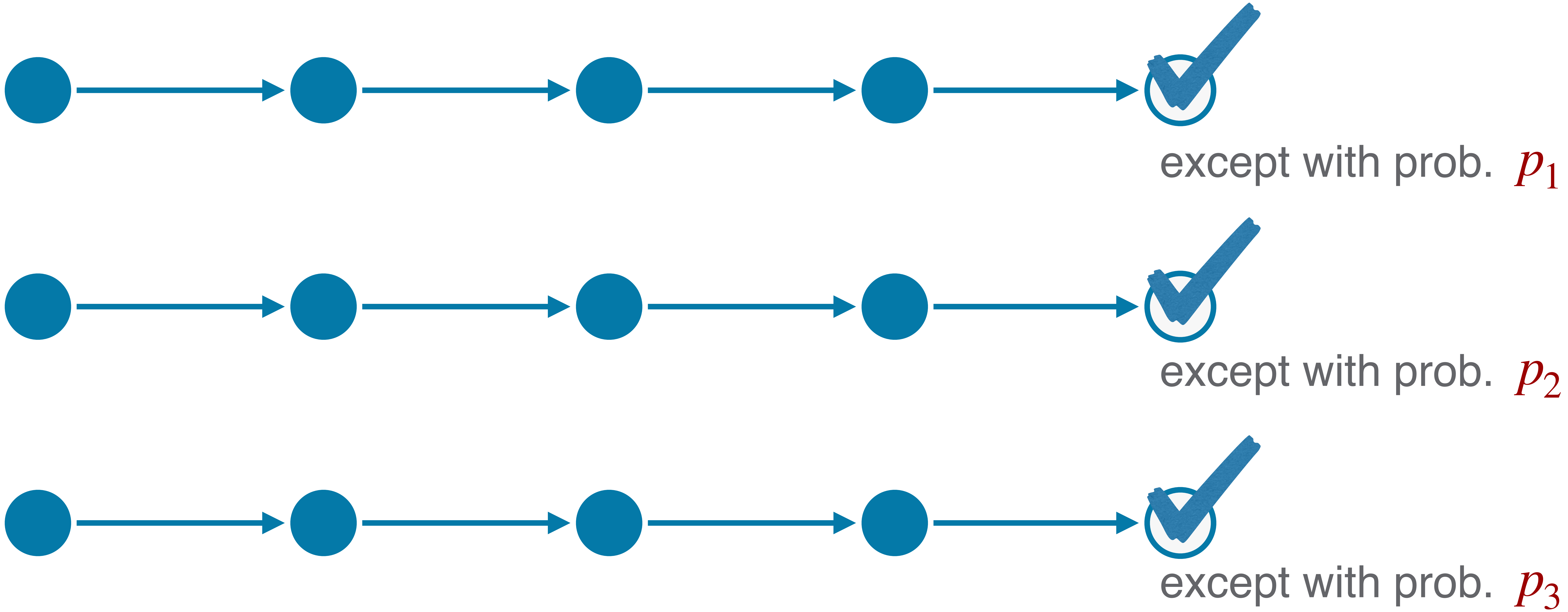
$$\exists R \,.\, \forall w, w', x \,.$$



$$\exists R\,.\,\forall w, w', x\,.$$

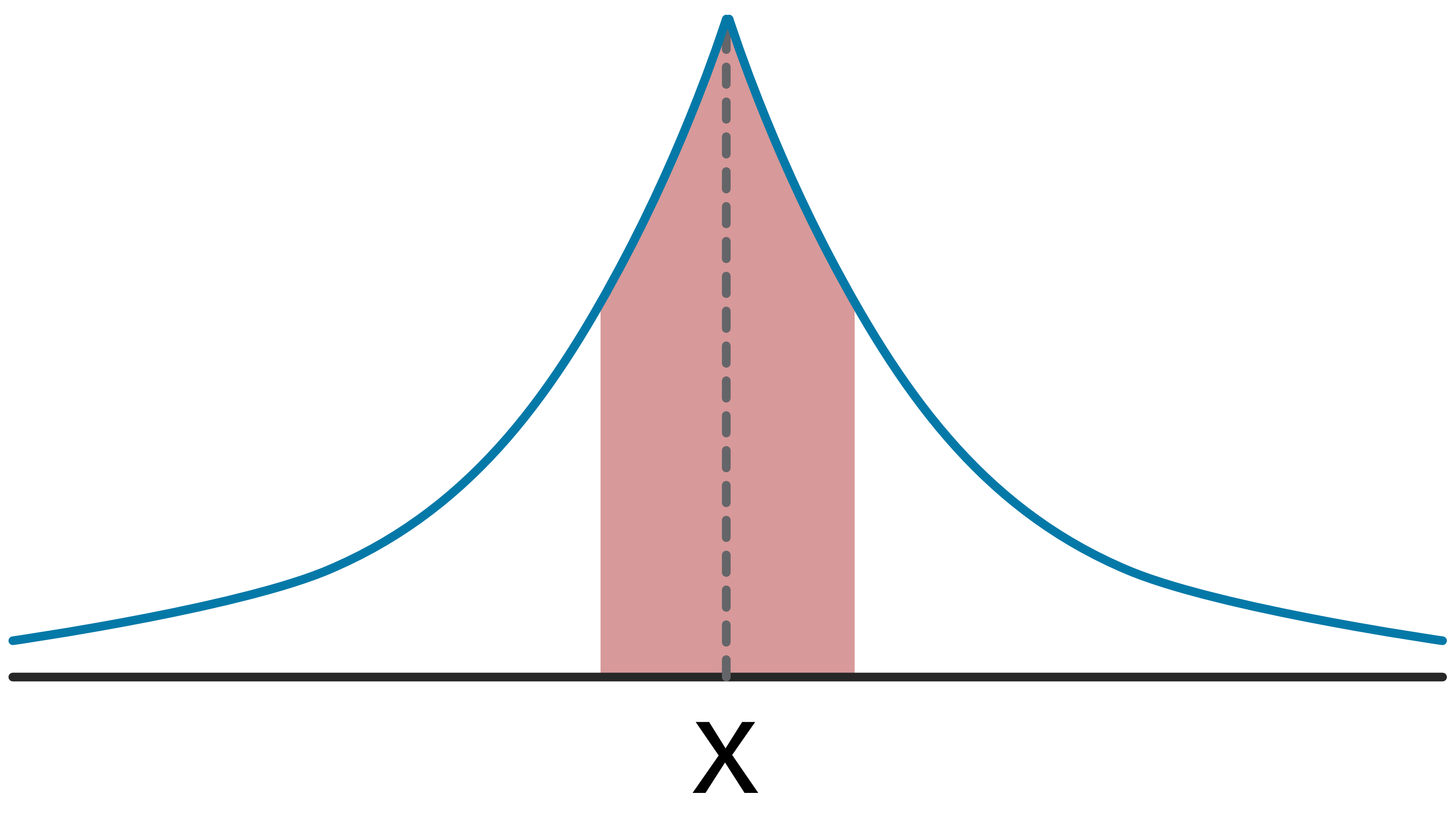
$$w = 0 \wedge R(x) \wedge w' = w + Pr(\neg R(x)) \implies x \wedge w' \leq 1 - p$$


except with prob. p_1
except with prob. p_2
except with prob. p_3

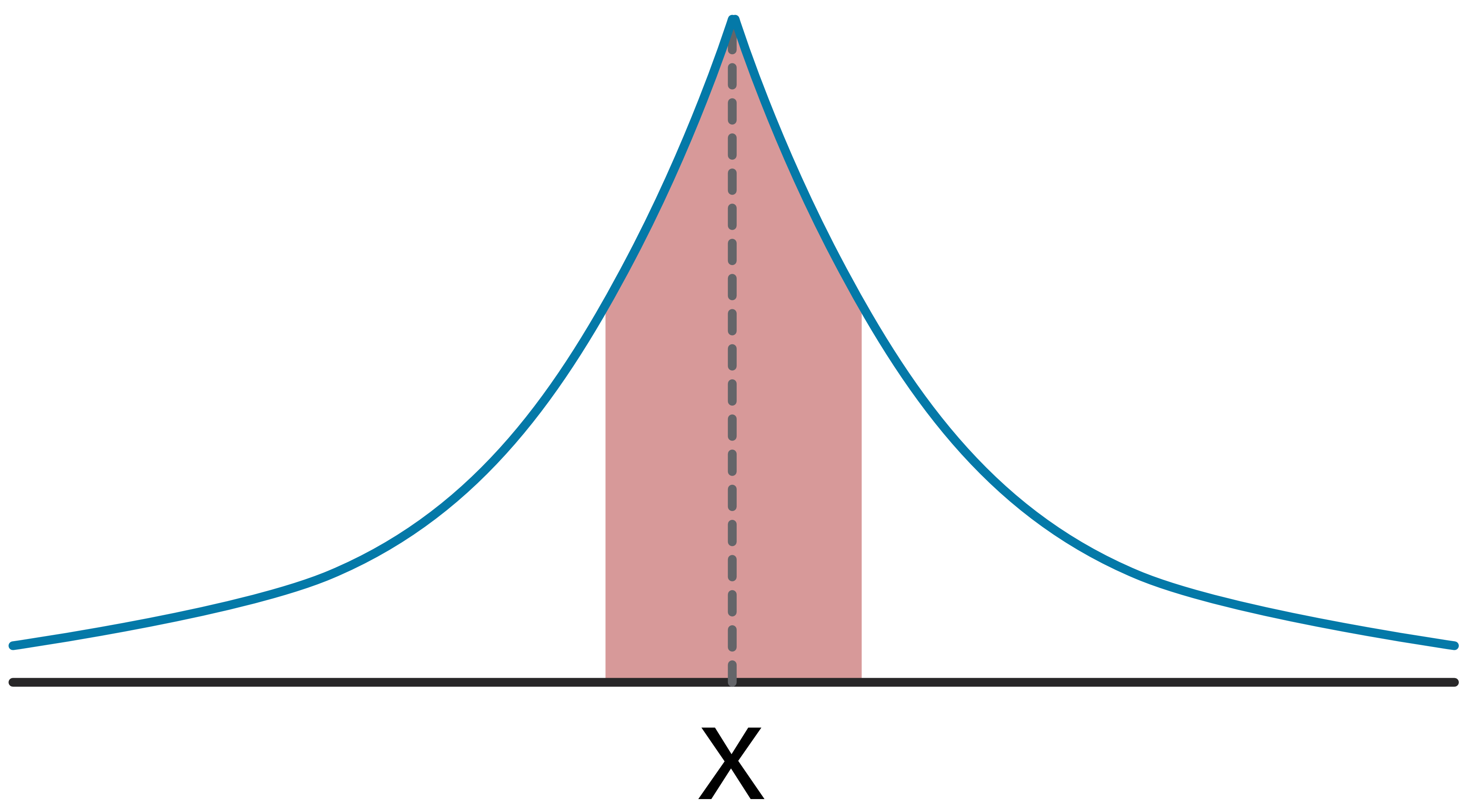


simply, total failure probability is $\sum_i p_i$

y ~ Lap(x,s)



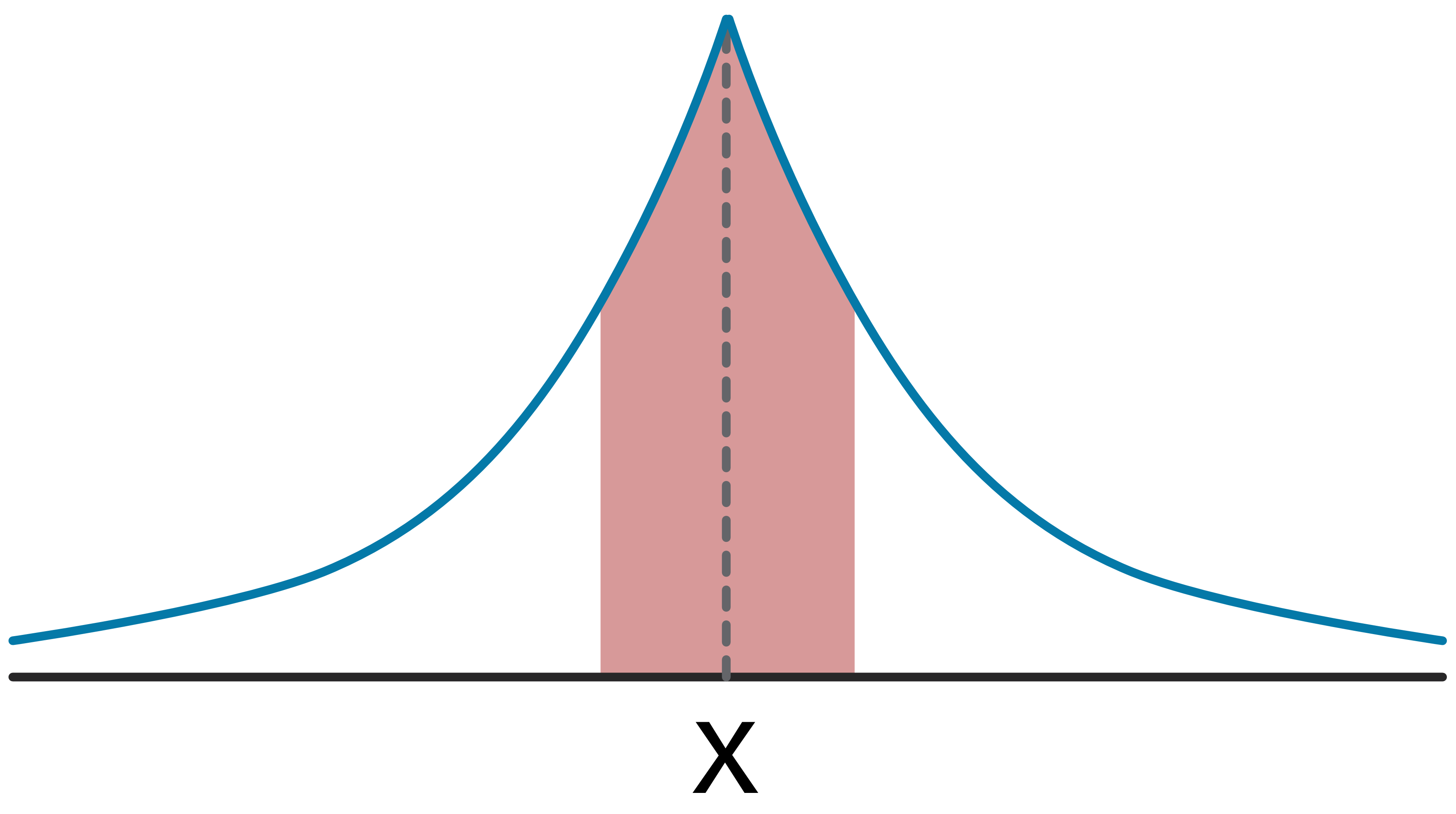
`y ~ Lap(x,s)`



axiom family

$$|x - y| \leq s \cdot log\left(\frac{1}{f(V_I)}\right)$$

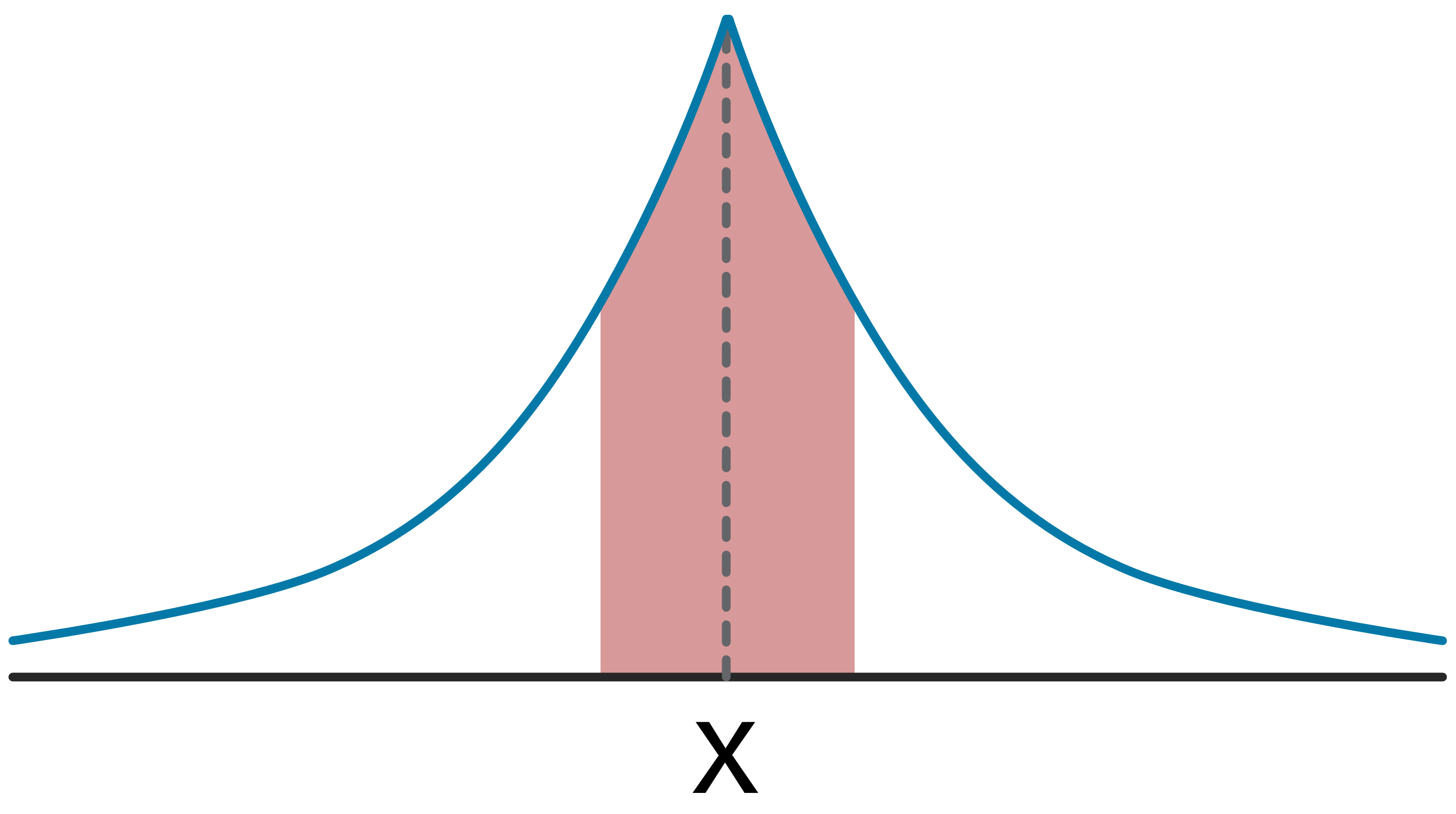
`y ~ Lap(x,s)`



axiom family

$$|x - y| \leq s \cdot log\left(\frac{1}{f(V_I)}\right)$$

`y ~ Lap(x,s)`



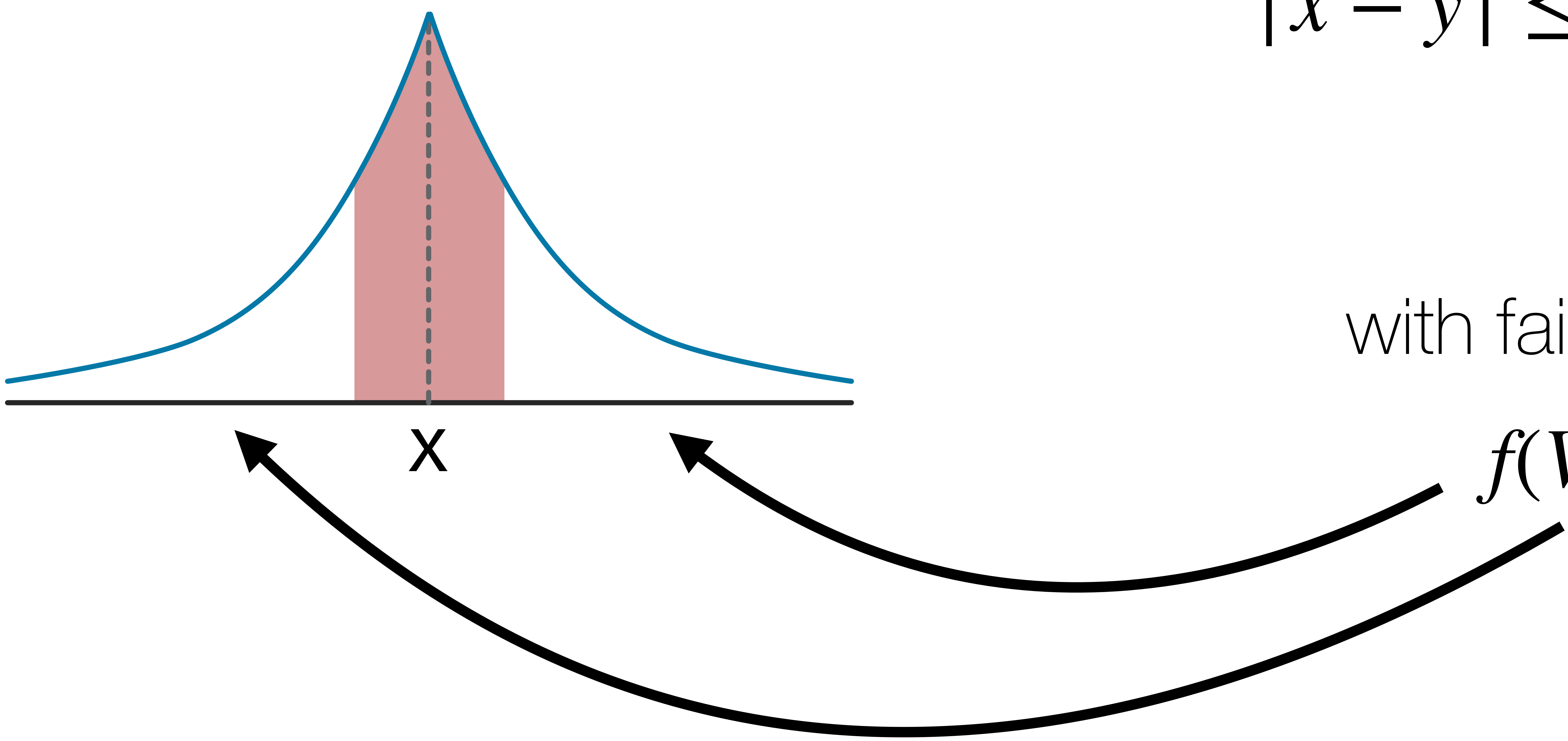
axiom family

$$|x - y| \leq s \cdot log\left(\frac{1}{f(V_I)}\right)$$

with failure probability

$$f(V_I) \in (0,1]$$

`y ~ Lap(x,s)`



axiom family

$$|x - y| \leq s \cdot log\left(\frac{1}{f(V_I)}\right)$$

with failure probability

$$f(V_I) \in (0,1]$$

```
def rnm(q):
  i, best, r = 0
  while i < |q|
   d ~ Lap(q[i], 2/ε)

   if d > best || i = 0
    r = i
    best = d

   i = i + 1
  return r
```

$p \in (0,1)$

```
def rnm(q):
  i, best, r = 0
  while i < |q|
    d ~ Lap(q[i], 2/ε)

    if d > best || i = 0
      r = i
      best = d

    i = i + 1
  return r
```

```
p ∈ (0,1)
```

```
def rnm(q):
  i, best, r = 0
  while i < |q|
    d ~ Lap(q[i], 2/ε)

    if d > best || i = 0
      r = i
      best = d

    i = i + 1
  return r
```

```
∀j. q[r] ⩾ q[j] - 4/ε*log(|q|/p)
```
except with prob. ⩽ p

$p \in (0,1)$

```
def rnm(q):
  i, best, r = 0
  while i < |q|
    d ~ Lap(q[i], 2/ε)

    if d > best || i = 0
      r = i
      best = d

    i = i + 1
  return r
```

$\forall j.\ q[r] \geqslant q[j] - 4/\varepsilon * \log(|q|/p)$
except with prob. $\leqslant p$

$p \in (0,1)$

```
def rnm(q):
  i, best, r = 0
  while i < |q|
    d ~ Lap(q[i], 2/ε)

    if d > best || i = 0
      r = i
      best = d

    i = i + 1
  return r
```

$$|q[i] - d| \leq \frac{2}{\epsilon} \cdot log\left(\frac{|q|}{p}\right)$$

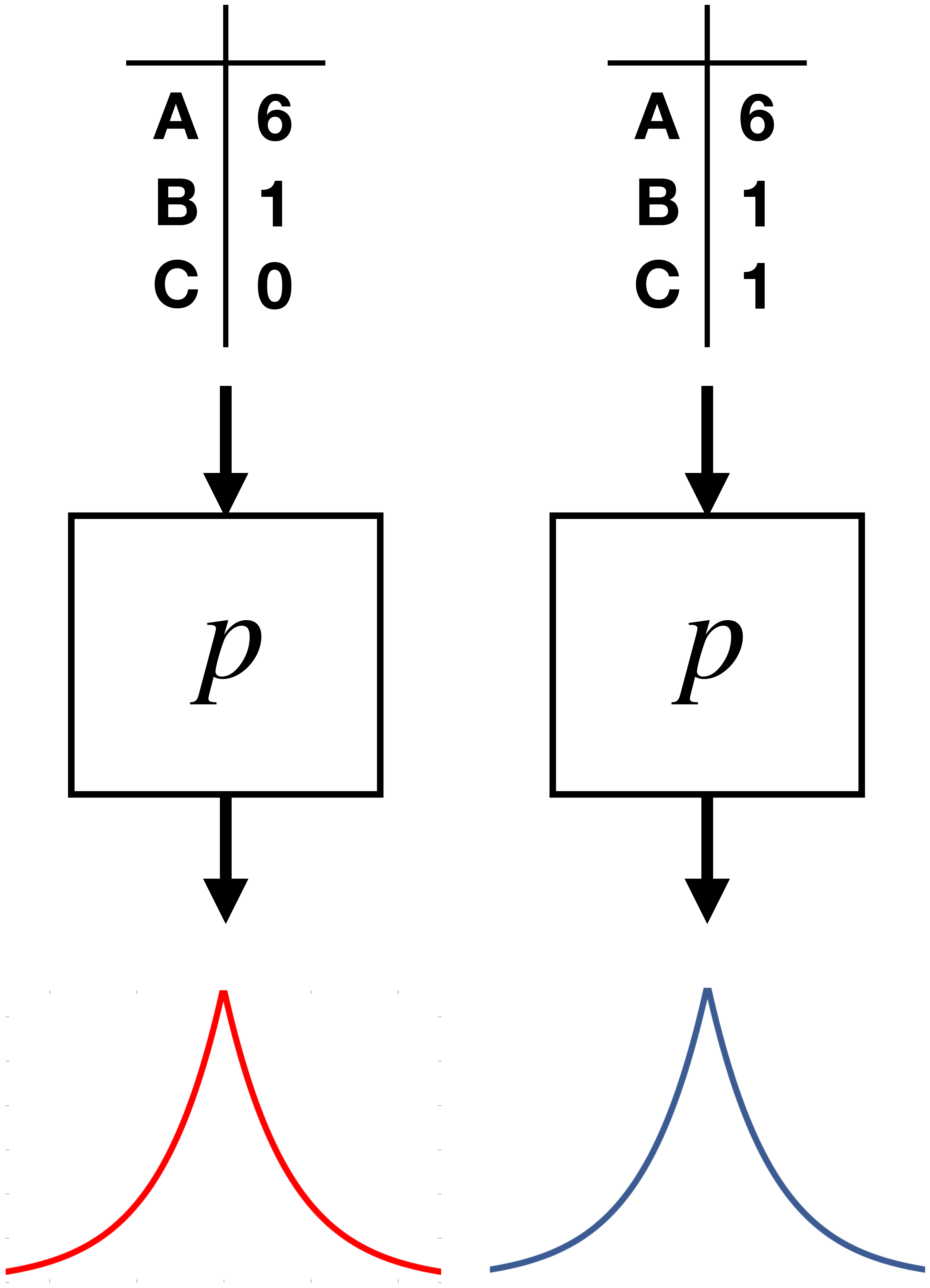
with failure probability $\frac{p}{|q|}$

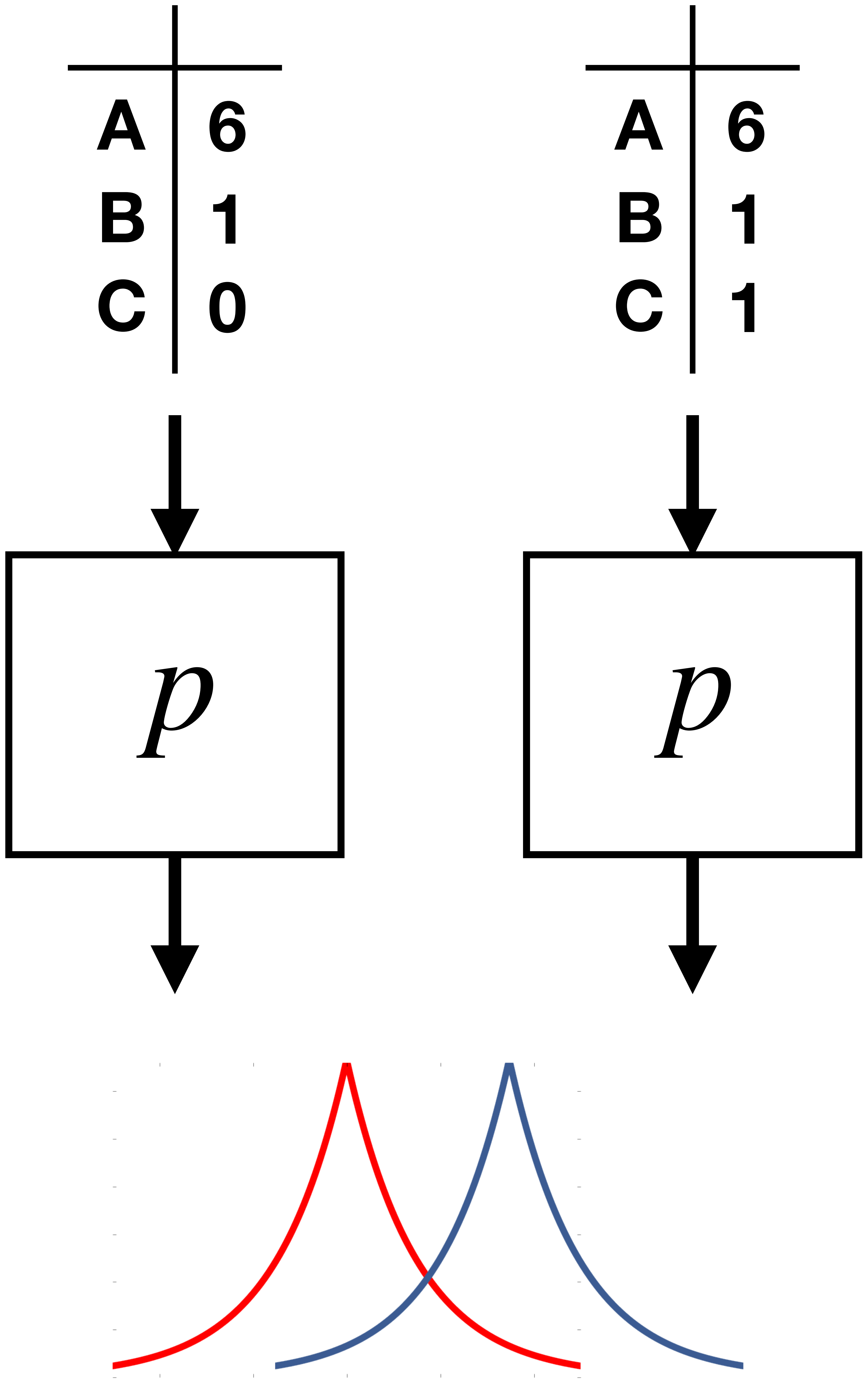
∀j. q[r] ⩾ q[j] - 4/ε*log(|q|/p)
except with prob. ⩽ p

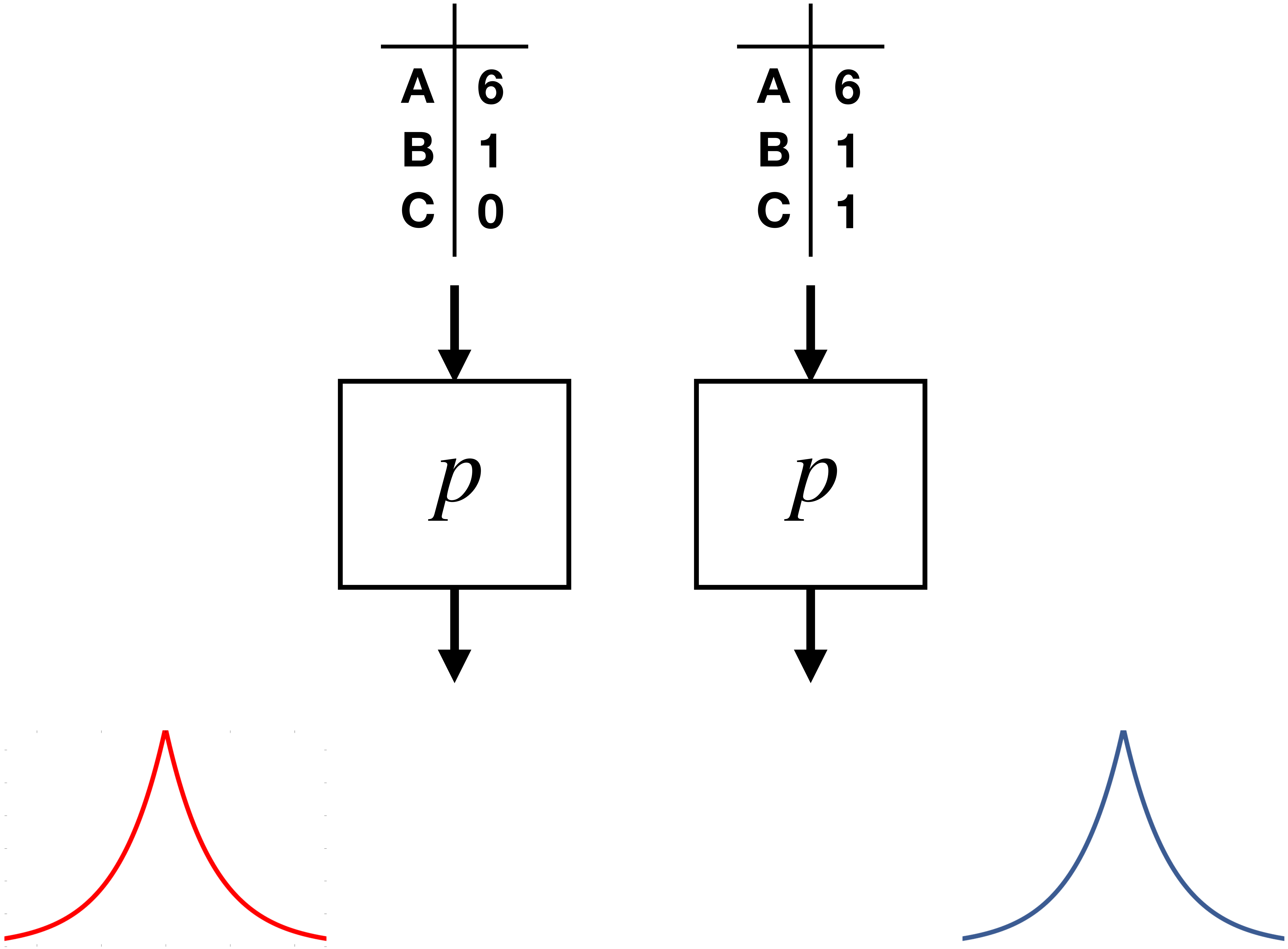
1 automatic proofs of accuracy [POPL19]

2 automatic proofs of differential privacy [POPL18]

theme get rid of probability! long live logic!


A 6
B 1
C 0
A 6
B 1
C 1
p
p


A 6
B 1
C 0
A 6
B 1
C 1
p
p



A 6
B 1
C 0
A 6
B 1
C 1
p
p

$$\forall d \sim d', a, \epsilon \,.$$

$$\forall d \sim d', a, \epsilon \,.$$

$$\mathbb{P}[p(d) = a] \leq e^{\epsilon} \cdot \mathbb{P}[p(d') = a]$$

$$\forall d \sim d', a, \epsilon \,.$$

$$\mathbb{P}[p(d) = a] \leq \boxed{e^{\epsilon}} \cdot \mathbb{P}[p(d') = a]$$

problems

problems

proving differential privacy is hard and error-prone [lyu et al. 16]

problems

proving differential privacy is hard and error-prone [lyu et al. 16]

existing automated techniques only work for simple algorithms

problems

proving differential privacy is hard and error-prone [lyu et al. 16]

existing automated techniques only work for simple algorithms

goal

automatically prove differential privacy of advanced algorithms

key ideas

key ideas

view differential privacy **coupling proofs as games**

key ideas

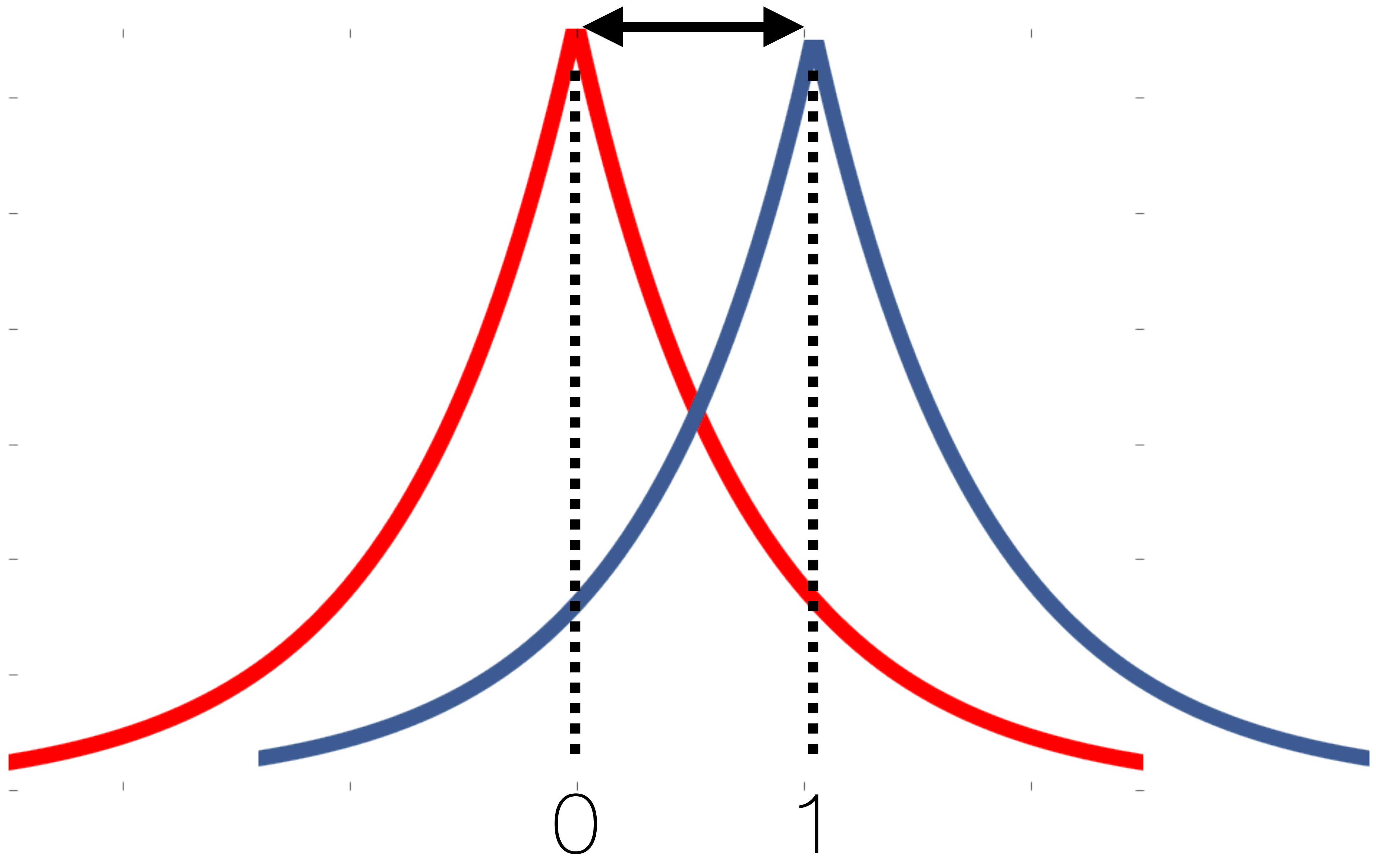
view differential privacy **coupling proofs as games**

solve a program **synthesis/verification problem**

$$\exists q \,.\, \forall x \,.\, \varphi(q, x)$$

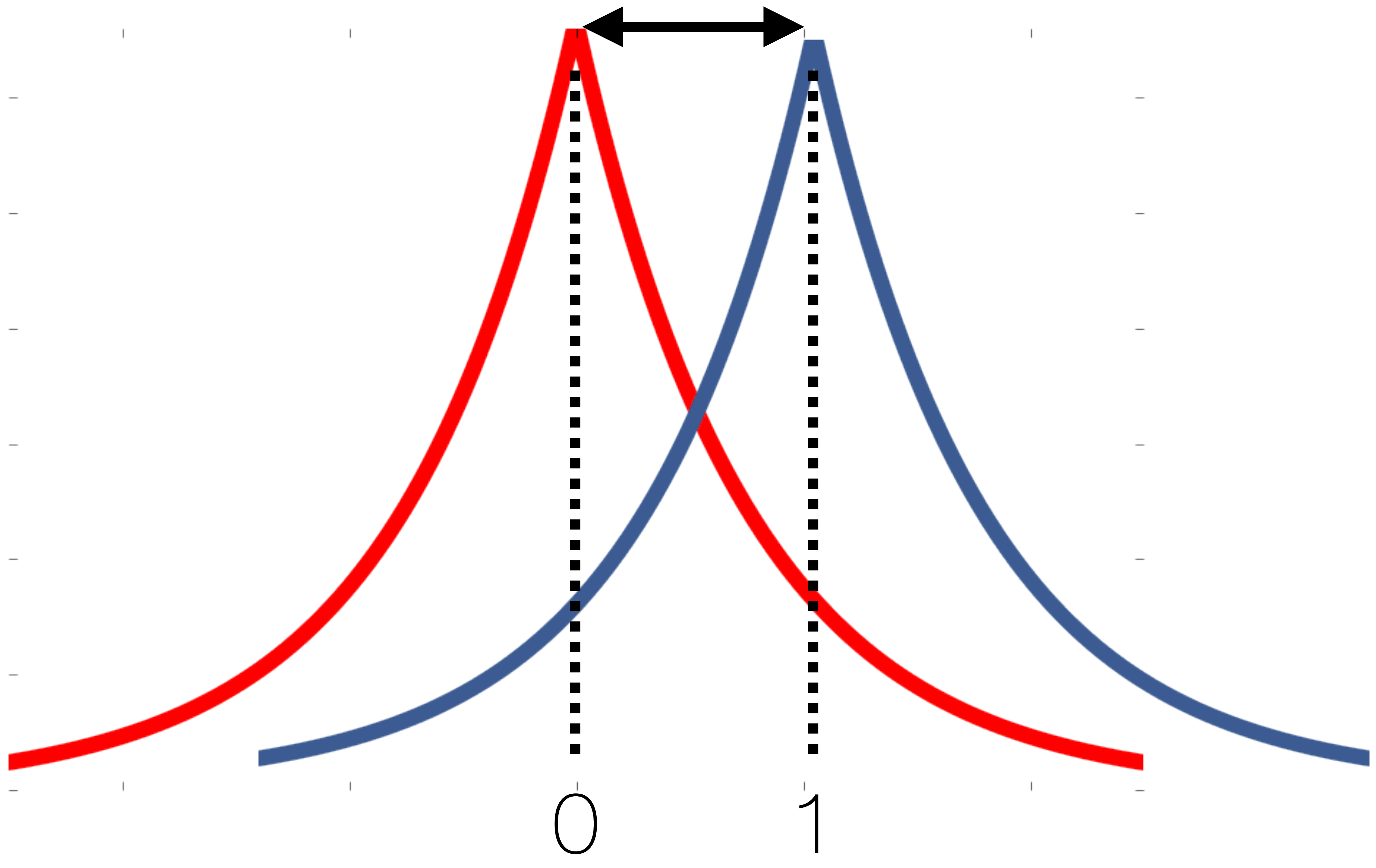
variable approximate couplings

scale of distributions is ***1/y***



variable approximate couplings

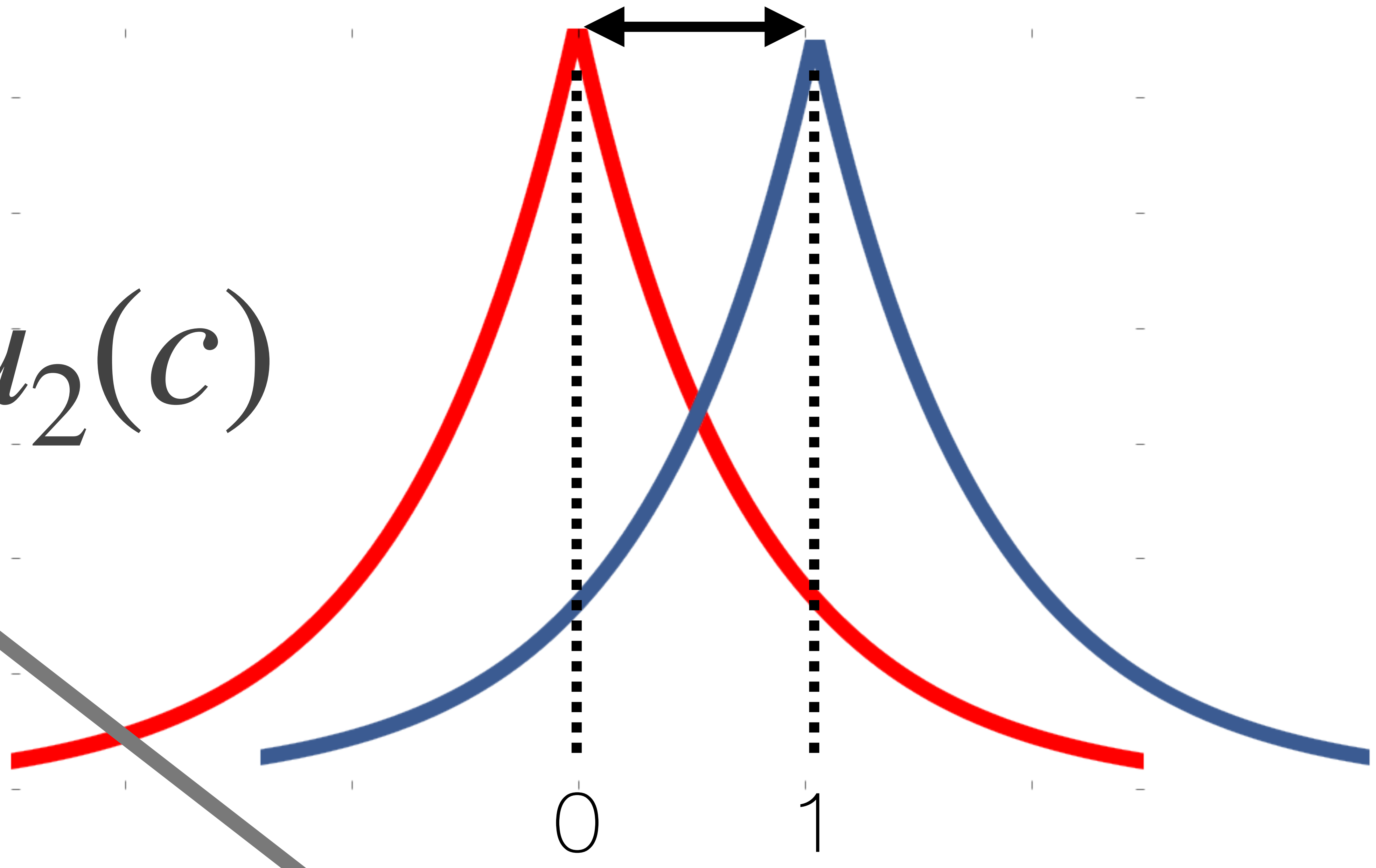
scale of distributions is ***1/y***



$$\{(c, c, y) \mid c \in \mathbb{Z}\}$$

variable approximate couplings

scale of distributions is ***1/y***

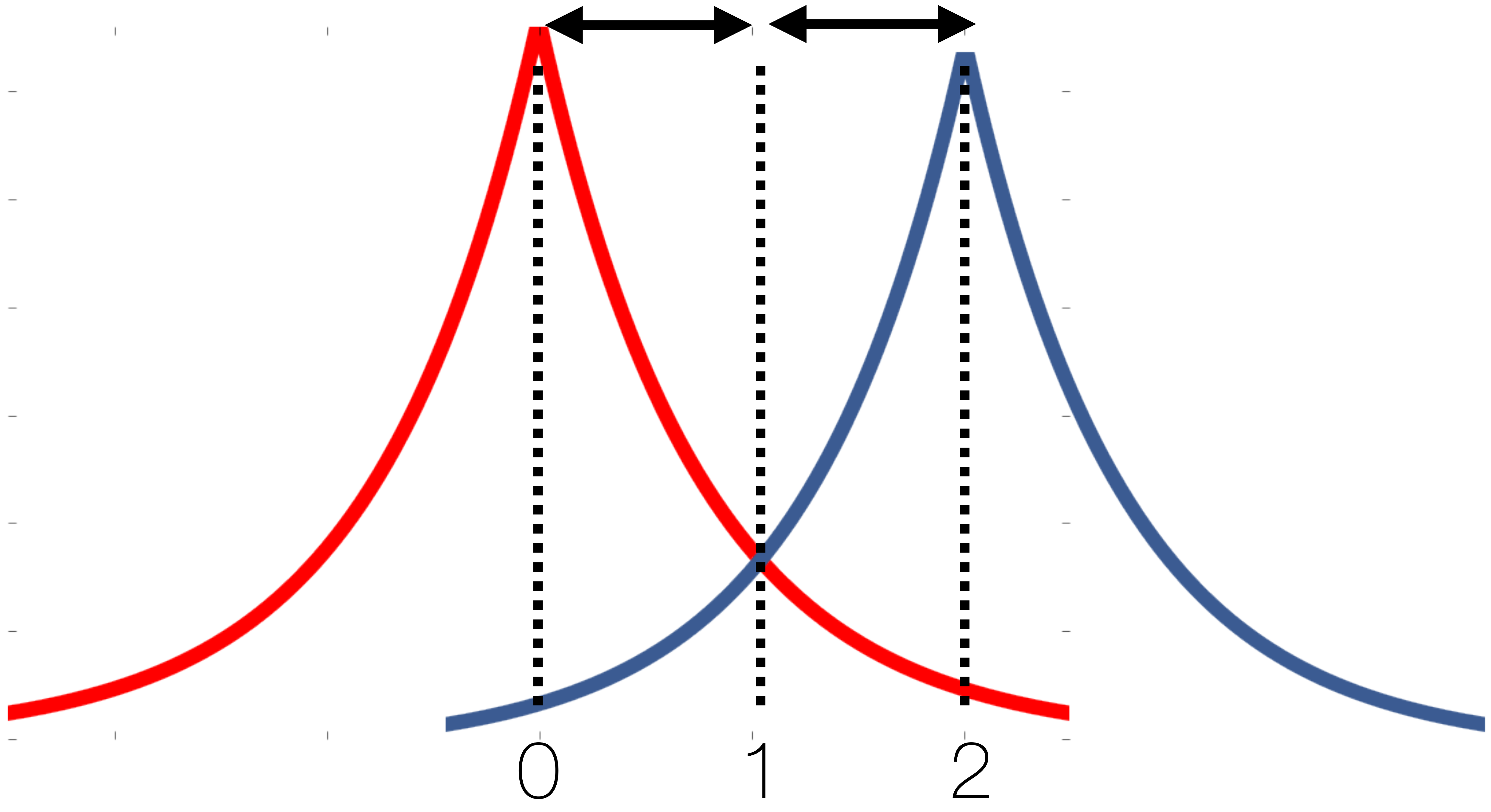


$$\mu_1(c) \leq e^y \cdot \mu_2(c)$$

$$\{(c, c, y) \mid c \in \mathbb{Z}\}$$

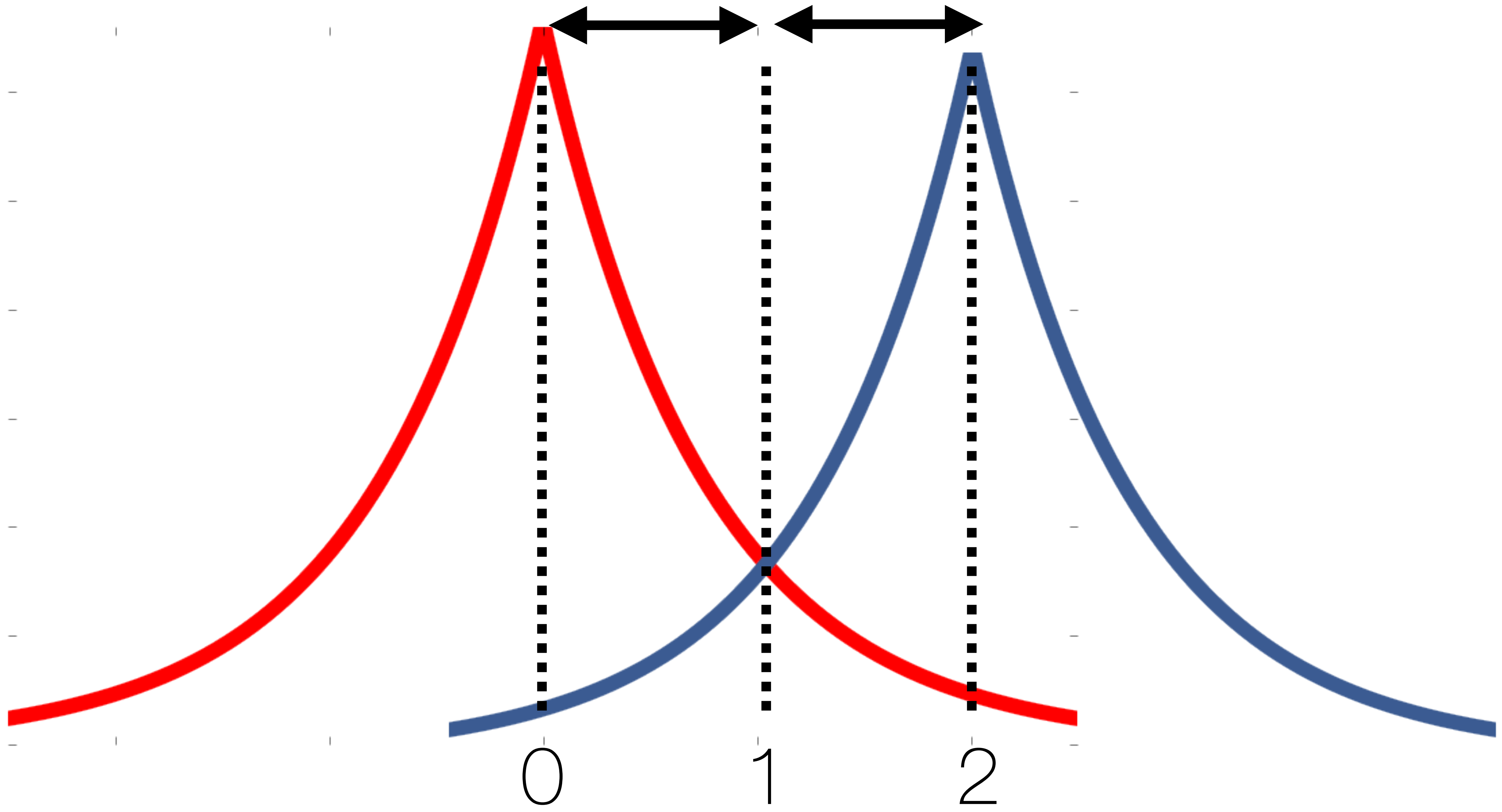
variable approximate couplings

scale of distributions is ***1/y***



variable approximate couplings

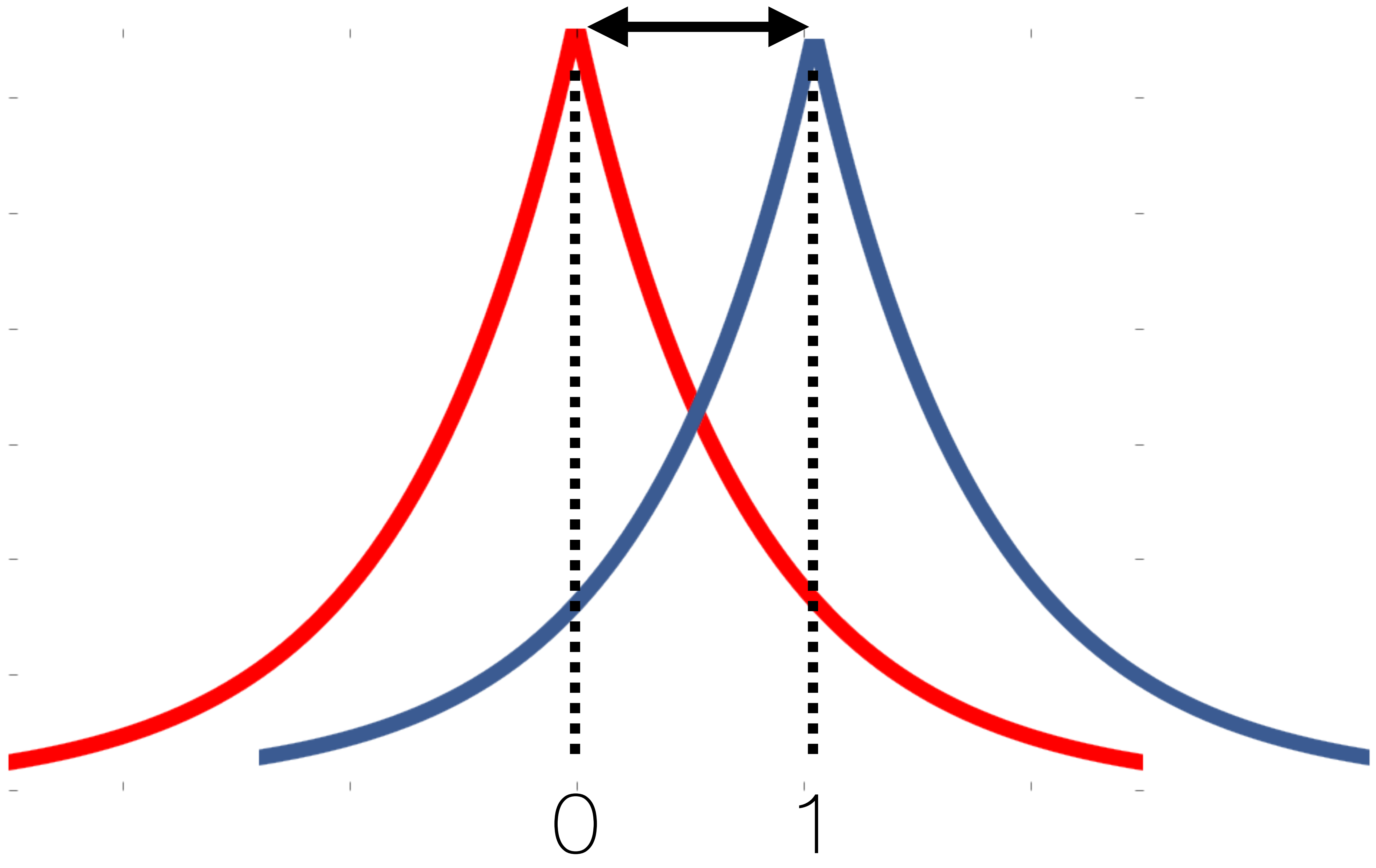
scale of distributions is ***1/y***



$$\{(c, c, 2y) \mid c \in \mathbb{Z}\}$$

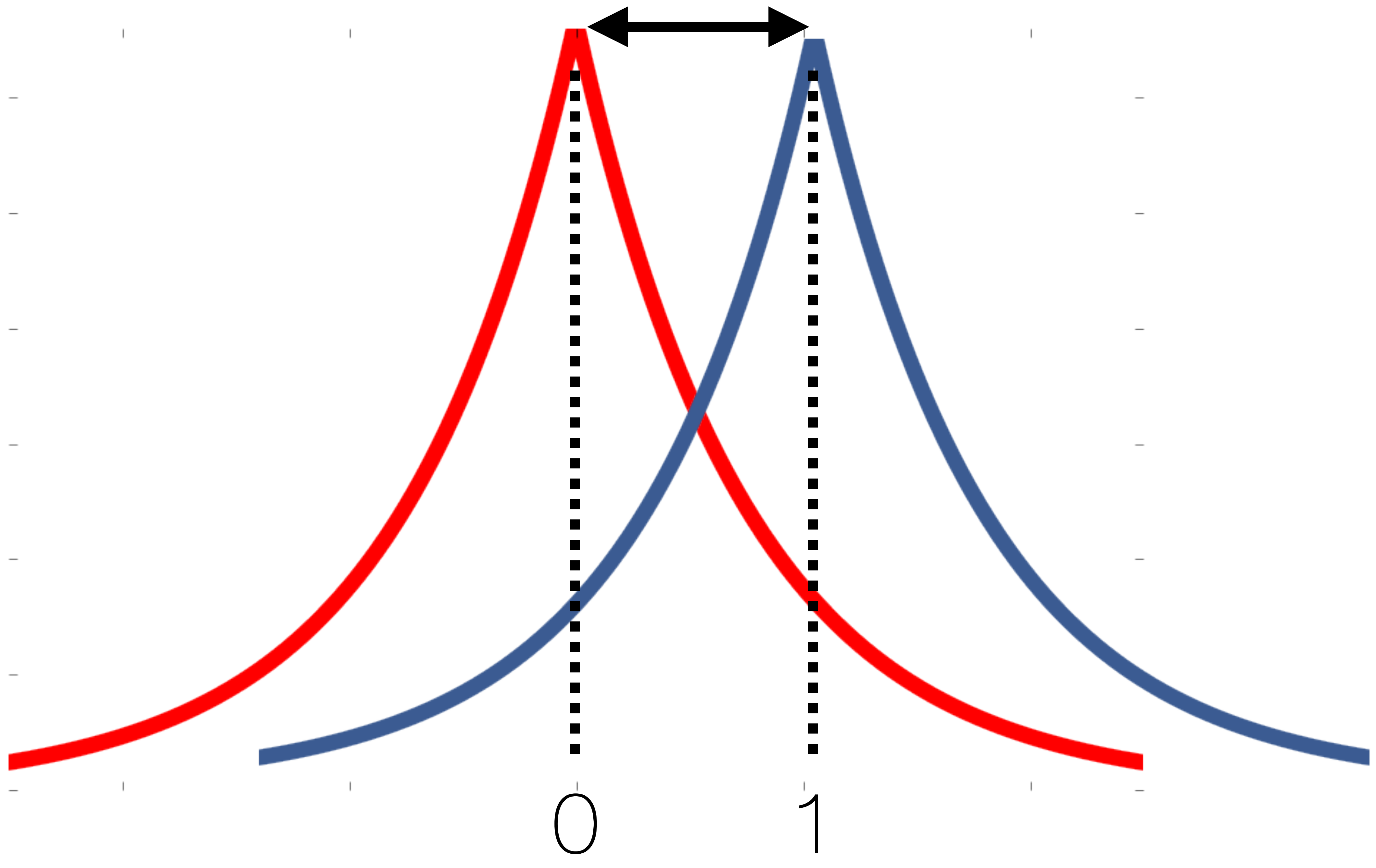
variable approximate couplings

scale of distributions is ***1/y***



variable approximate couplings

scale of distributions is ***1/y***



$$\{(c, c+1, 0) \mid c \in \mathbb{Z}\}$$

proof rule

p is DP if $\forall d \sim d', \epsilon \,.\, \exists \mathscr{C} \,.$

proof rule

p is DP if $\forall d \sim d', \epsilon \,.\, \exists \mathscr{C} \,.$

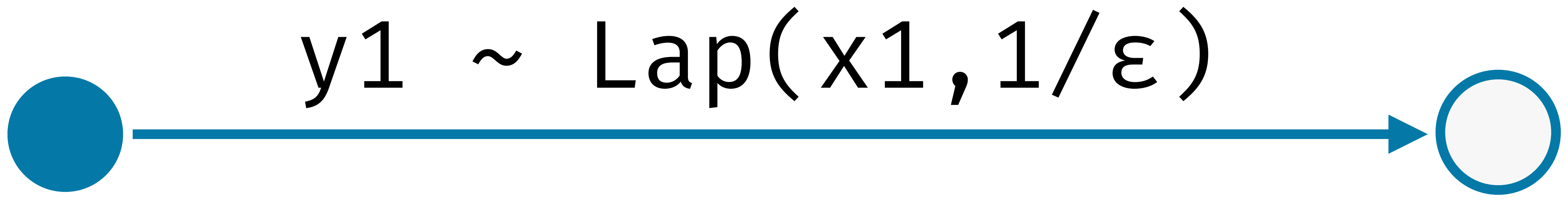
$\mathscr{C}$ couples $p(d), p(d')$

proof rule

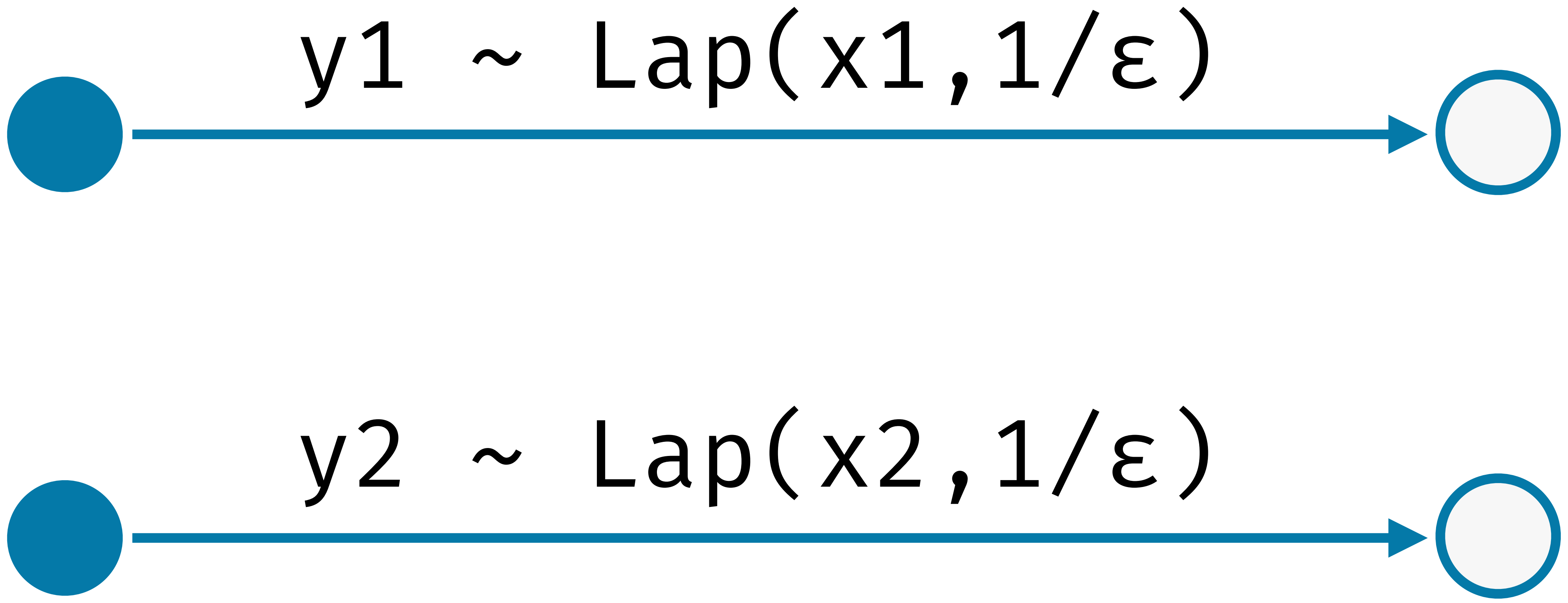
p is DP if $\forall d \sim d', \epsilon \,.\, \exists \mathscr{C} \,.$

$\mathscr{C}$ couples $p(d), p(d')$

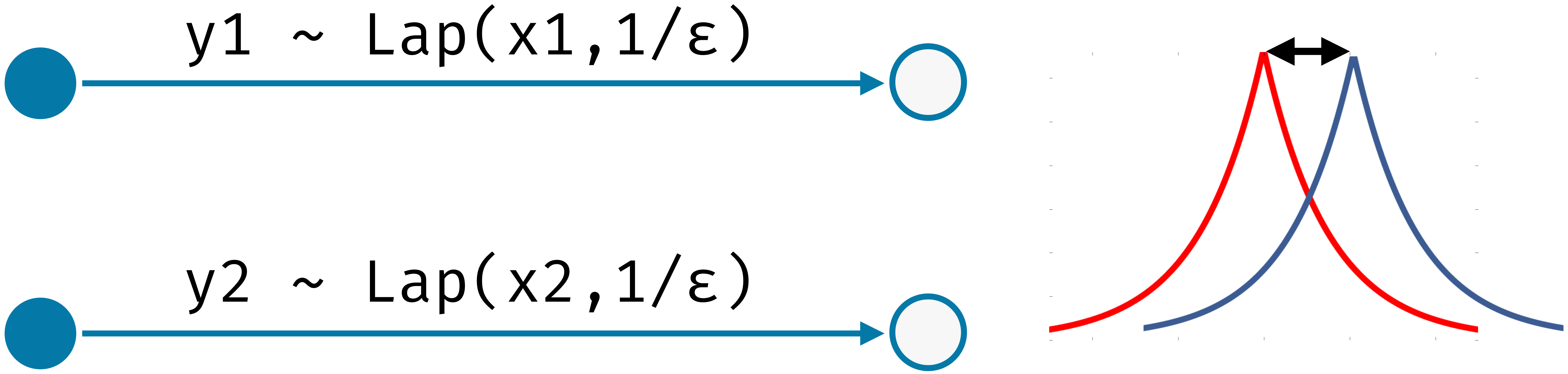
$\mathscr{C} = \{(c, c, y) \mid y \leq \epsilon\}$

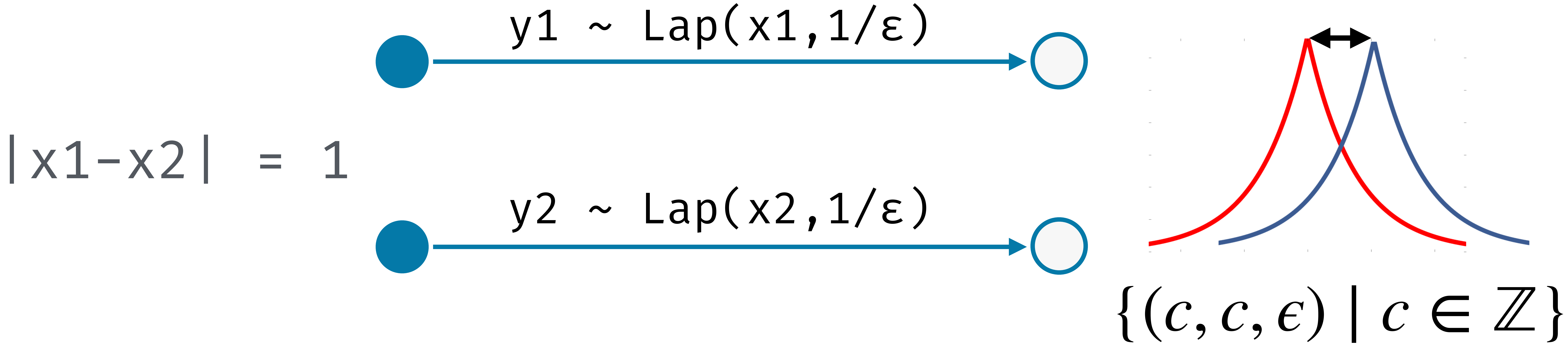

y1 ~ Lap(x1,1/ε)

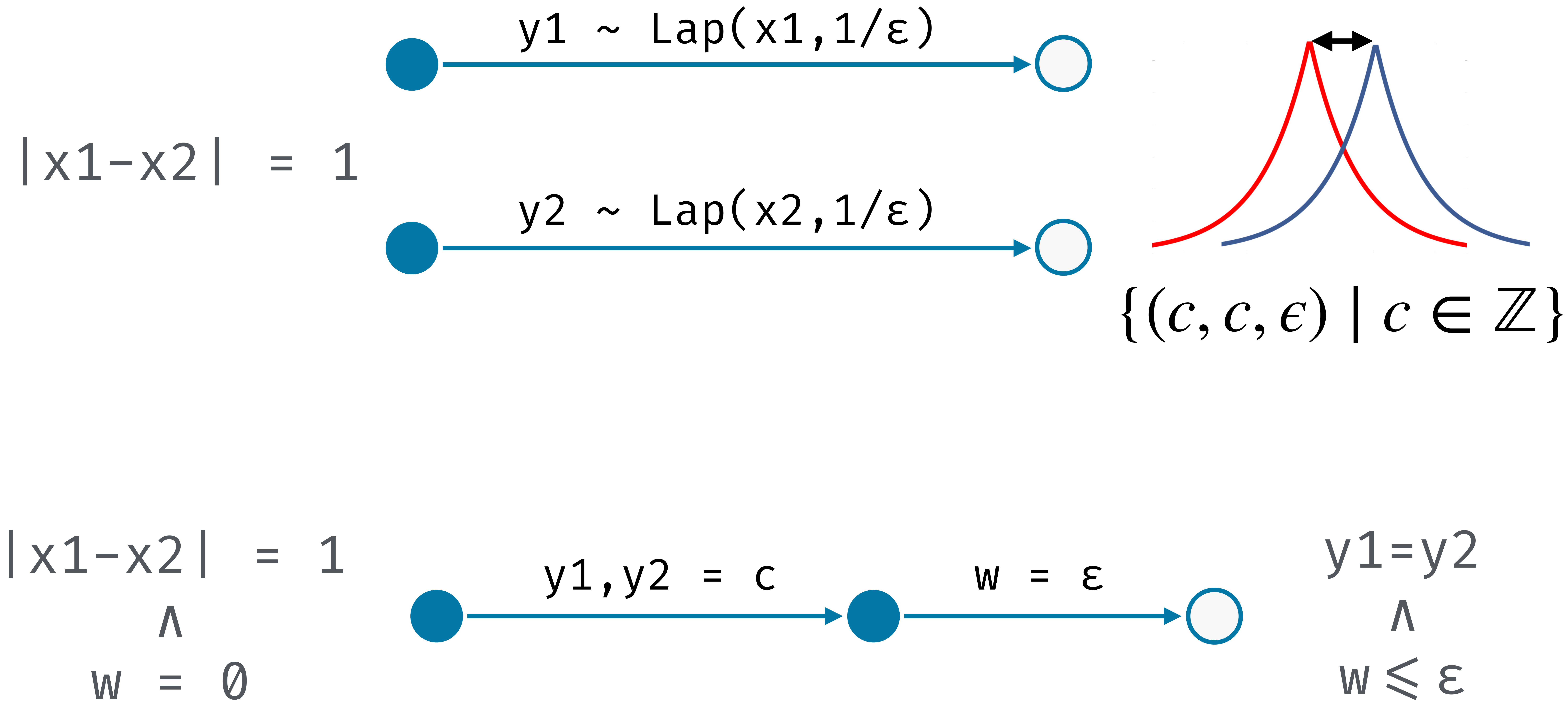
$|x1-x2| = 1$

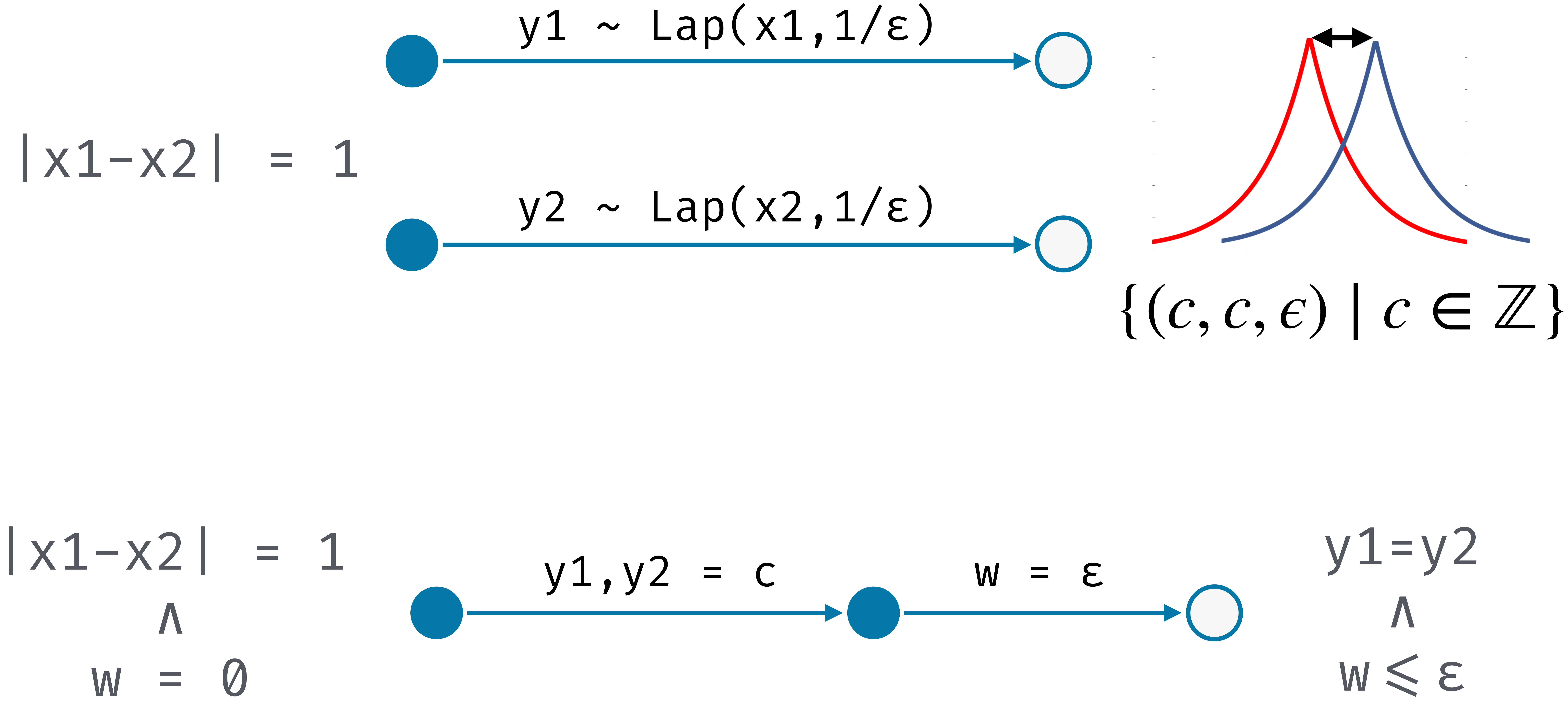


|x1-x2| = 1



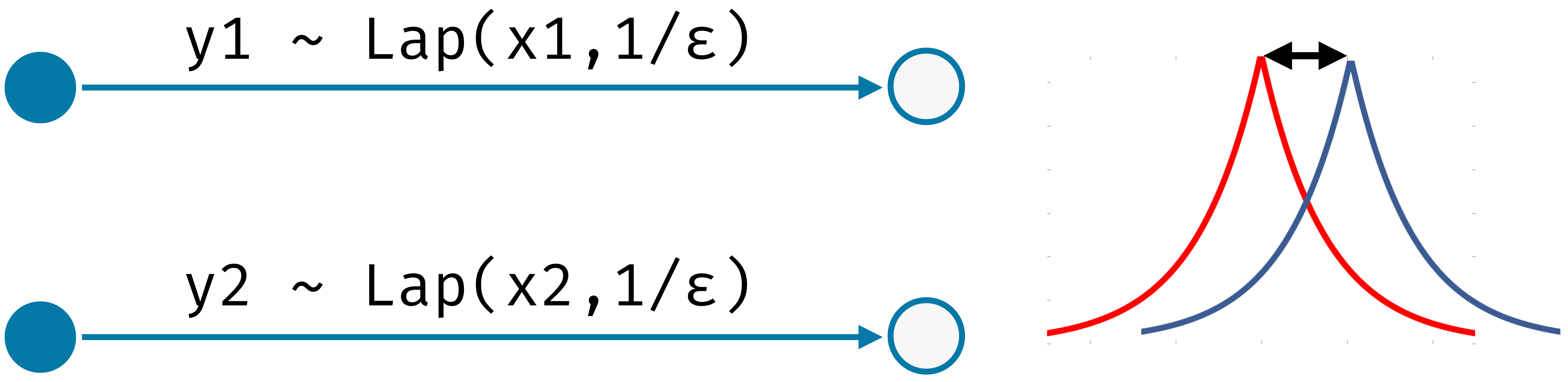

|x1-x2| = 1
y1 ~ Lap(x1,1/ε)
y2 ~ Lap(x2,1/ε)
$\{(c, c, \epsilon) \mid c \in \mathbb{Z}\}$


|x1-x2| = 1
y1 ~ Lap(x1,1/ε)
y2 ~ Lap(x2,1/ε)
{(c, c, ϵ) | c ∈ ℤ}
|x1-x2| = 1
∧
w = 0
y1,y2 = c
w = ε
y1=y2
∧
w ≤ ε



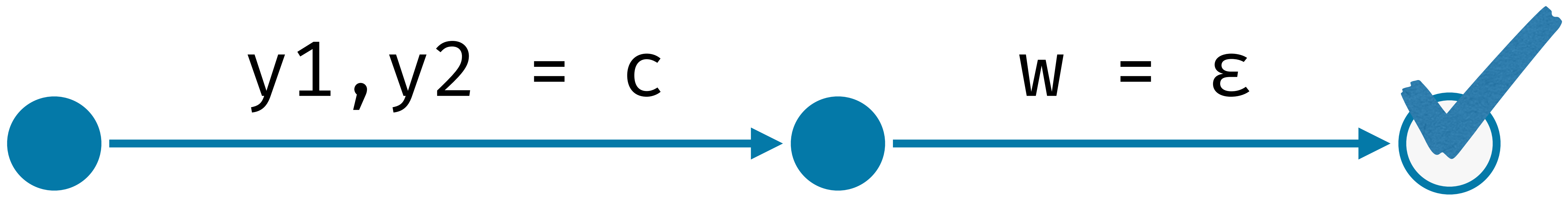
the triple **(y1,y2,w)** is a coupling!

|x1-x2| = 1



$\{(c, c, \epsilon) \mid c \in \mathbb{Z}\}$

|x1-x2| = 1
∧
w = 0



y1=y2
∧
w ⩽ ε

the triple **(y1,y2,w)** is a coupling!

let’s play!

```
def rnm(q):
  i, best, r = 0

  while i < |q|
   d ~ Lap(q[i], 2/ε)

   if d > best || i = 0
    r = i
    best = d

   i = i + 1

  return r
```

[dwork & roth 14]

```
def rnm(q):
  i, best, r = 0

  while i < |q|
    d ~ Lap(q[i], 2/ε)

    if d > best || i = 0
      r = i
      best = d

    i = i + 1

  return r
```

[dwork & roth 14]

```
q1 = [9, 0]          q2 = [10, 1]
```

```
def rnm(q):
  i, best, r = 0

  while i < |q|
   d ~ Lap(q[i], 2/ε)

   if d > best || i = 0
    r = i
    best = d

   i = i + 1

  return r
```

[dwork & roth 14]

```
q1 = [9, 0]          q2 = [10, 1]
          w = 0
```

```
def rnm(q):
  i, best, r = 0

  while i < |q|
   d ~ Lap(q[i], 2/ε)

   if d > best || i = 0
    r = i
    best = d

   i = i + 1

  return r
```

[dwork & roth 14]

q1 = [9, 0]   q2 = [10, 1]

w = 0

$r1 = r2 \wedge w \leqslant \varepsilon$

```
def rnm(q):
  i, best, r = 0

  while i < |q|
    d ~ Lap(q[i], 2/ε)

    if d > best || i = 0
      r = i
      best = d

    i = i + 1

  return r
```

[dwork & roth 14]

q1 = [9, 0]   q2 = [10, 1]

w = 0

r1 = r2 ∧ w ⩽ ε

```
def rnm(q):
  i, best, r = 0

  while i < |q|
    d ~ Lap(q[i], 2/ε)

    if d > best || i = 0
      r = i
      best = d

    i = i + 1

  return r
```

[dwork & roth 14]

q1 = [9, 0]   q2 = [10, 1]

w = 0

r1 = 0   r2 = 0

$r1 = r2 \wedge w \leqslant \varepsilon$

```
def rnm(q):
  i, best, r = 0

  while i < |q|
    d ~ Lap(q[i], 2/ε)

    if d > best || i = 0
      r = i
      best = d

    i = i + 1

  return r
```

[dwork & roth 14]

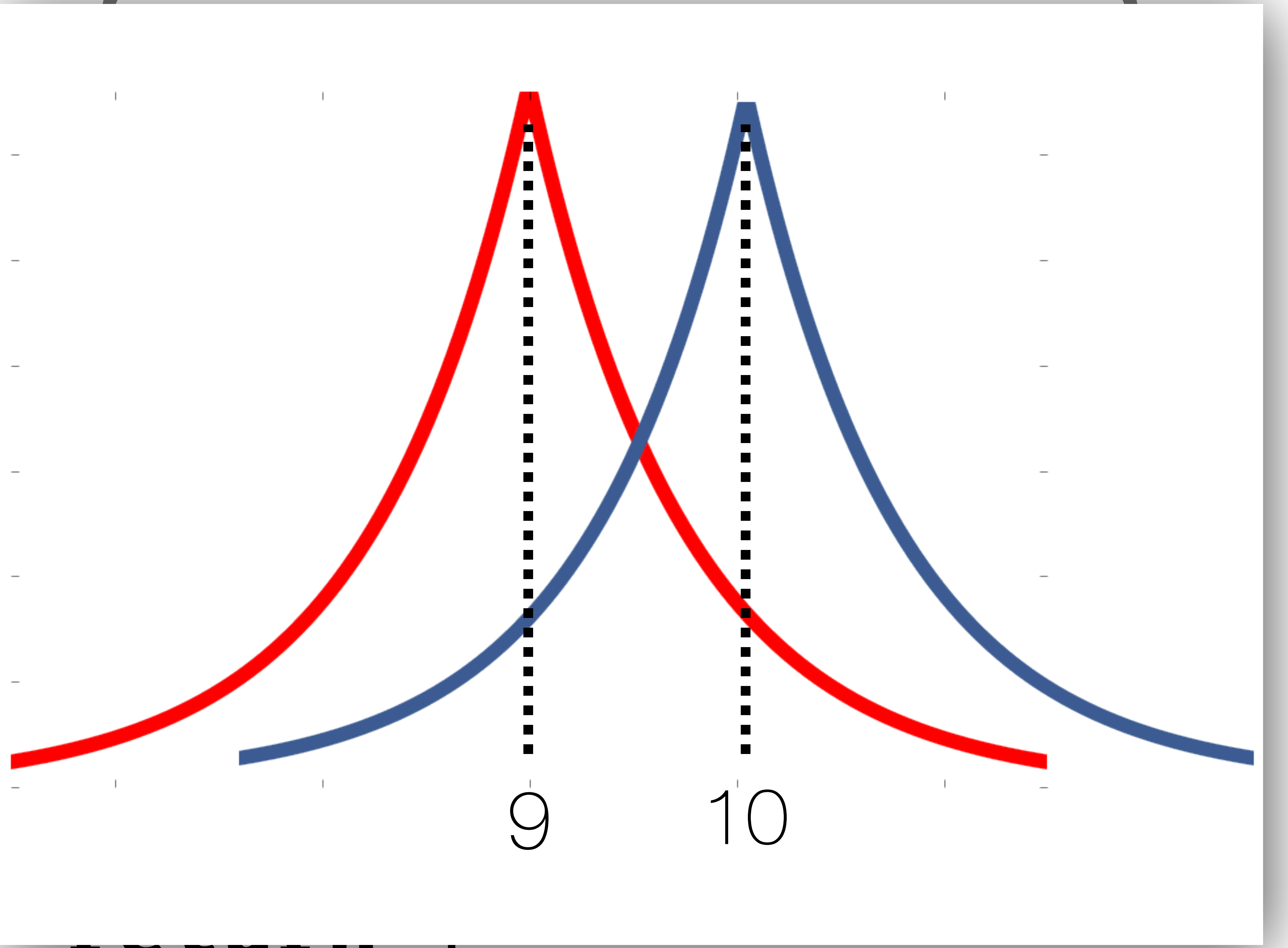
q1 = [9, 0] q2 = [10, 1]

w = 0

r1 = 0 r2 = 0

$r1 = r2 \wedge w \leqslant \varepsilon$

```
def rnm(q):
  i, best, r = 0

  while i < |q|
```

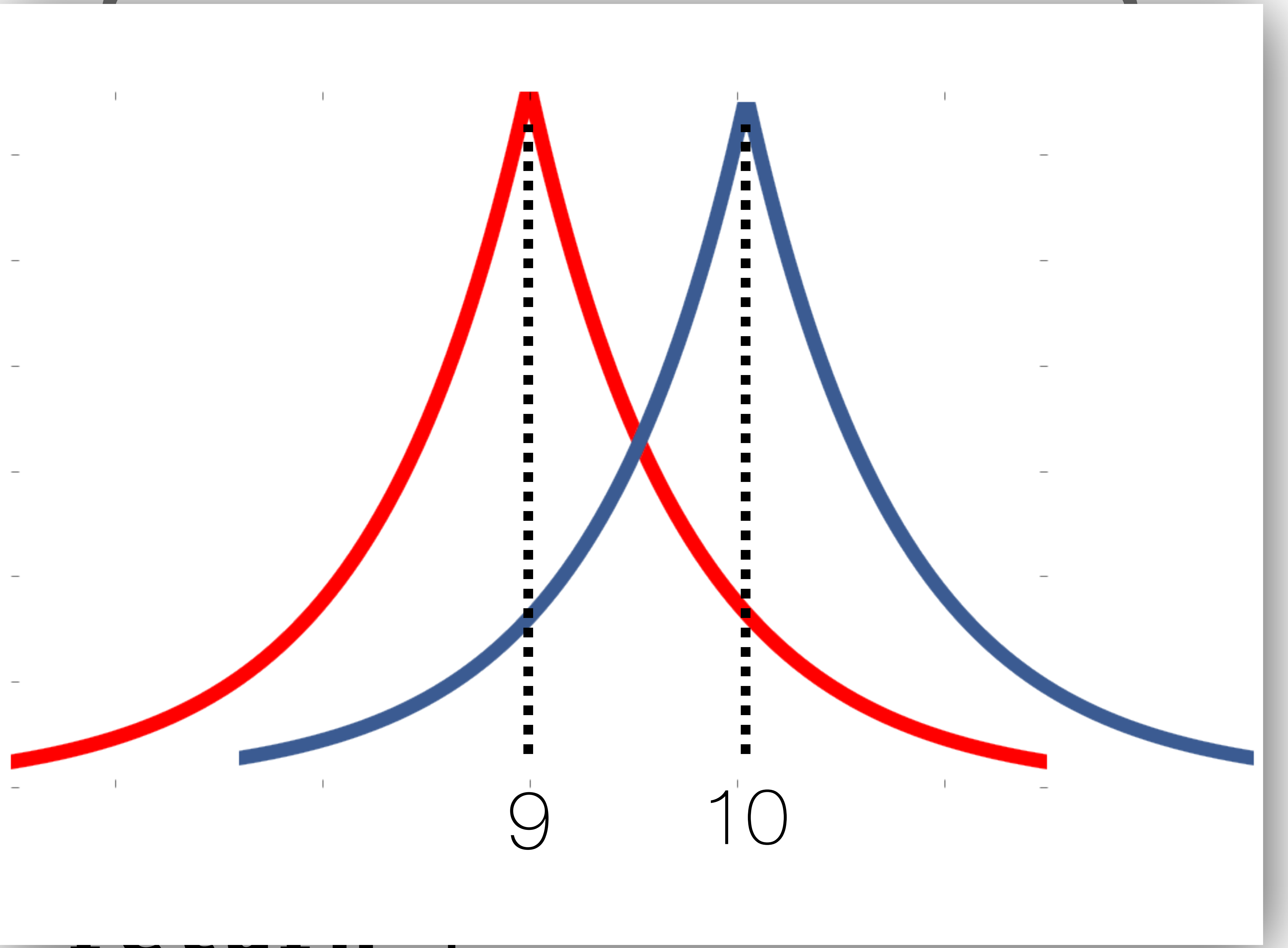


[dwork & roth 14]

```
q1 = [9, 0]          q2 = [10, 1]
            w = 0
r1 = 0               r2 = 0
```

$r1 = r2 \wedge w \leqslant \varepsilon$

```
def rnm(q):
  i, best, r = 0

  while i < |q|
```



[dwork & roth 14]

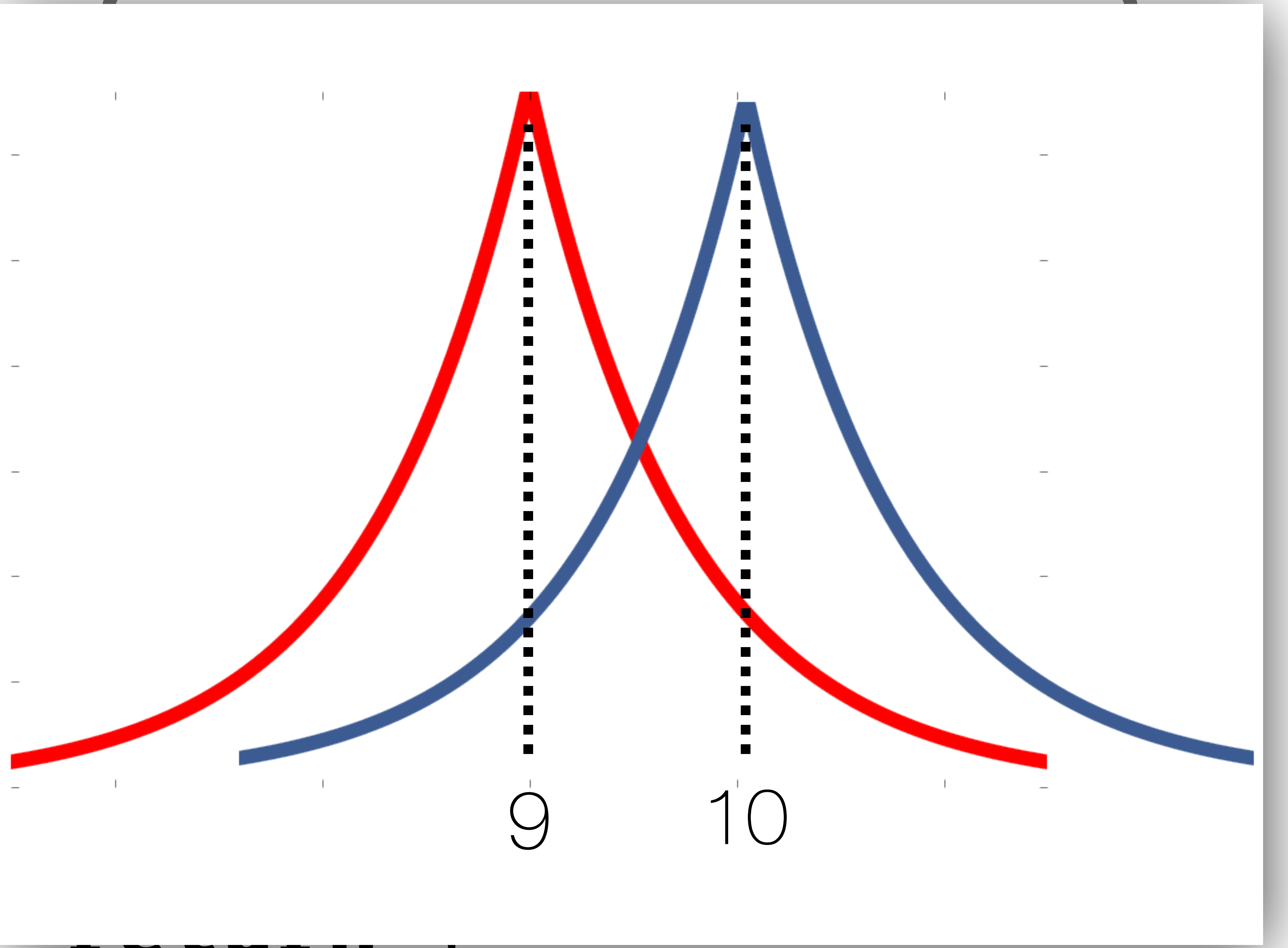
```
q1 = [9, 0]          q2 = [10, 1]
           w = 0
r1 = 0               r2 = 0
```

non-deterministically pick from

$$\{(c, c, \epsilon/2) \mid c \in \mathbb{Z}\}$$

$r1 = r2 \wedge w \leqslant \varepsilon$

```
def rnm(q):
  i, best, r = 0

  while i < |q|
```



[dwork & roth 14]

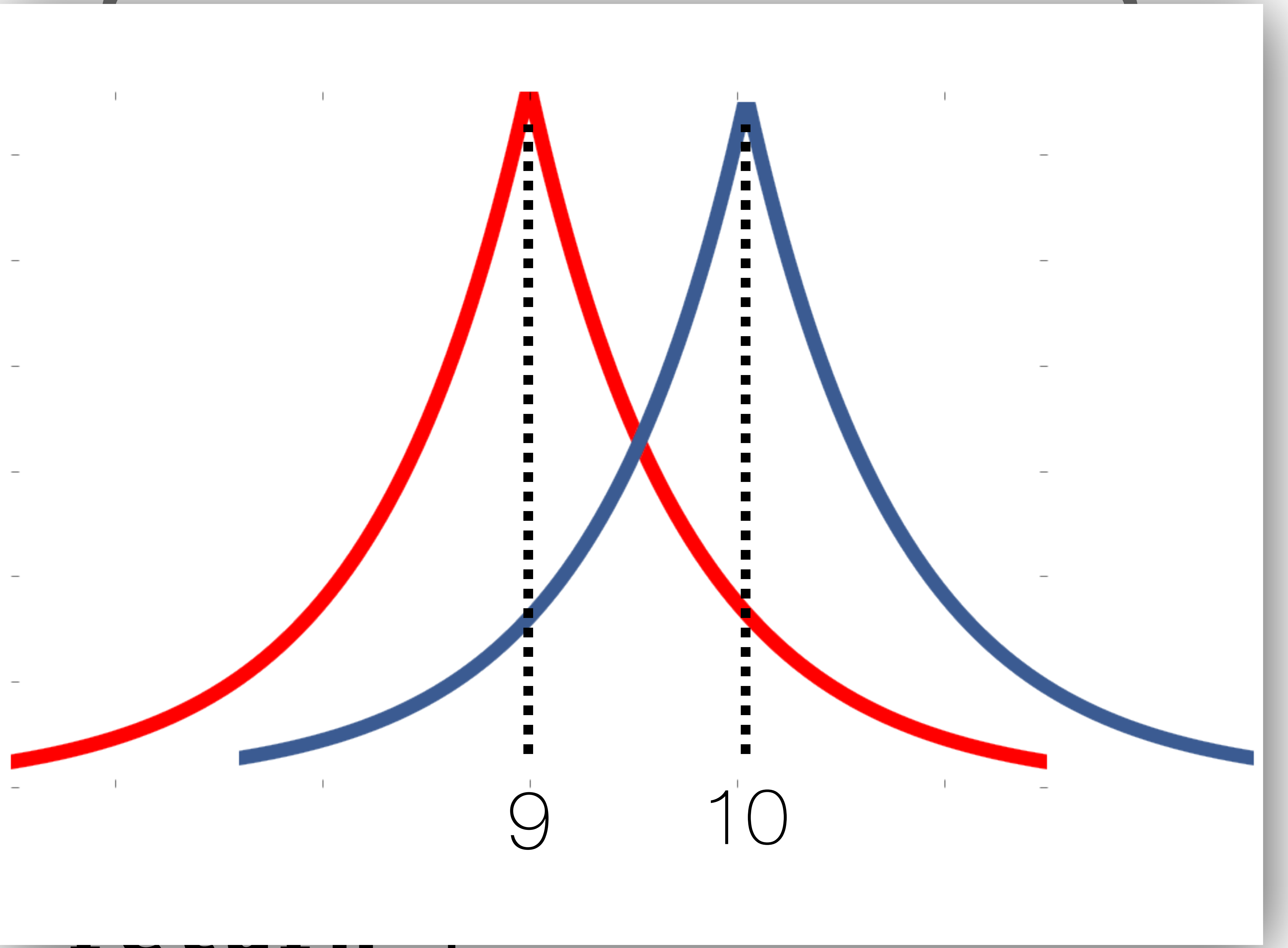
```
q1 = [9, 0]          q2 = [10, 1]
           w = 0
r1 = 0               r2 = 0

r1 = 0               r2 = 0
d1 = c               d2 = c
```

non-deterministically pick from

$$\{(c, c, \epsilon/2) \mid c \in \mathbb{Z}\}$$

$r1 = r2 \wedge w \leqslant \epsilon$

```
def rnm(q):
  i, best, r = 0

  while i < |q|
```



[dwork & roth 14]

```
q1 = [9, 0]          q2 = [10, 1]
             w = 0
r1 = 0               r2 = 0

r1 = 0               r2 = 0
d1 = c               d2 = c
            w = ε/2
```

non-deterministically pick from

$$\{(c, c, \epsilon/2) \mid c \in \mathbb{Z}\}$$

$r1 = r2 \land w \leqslant \epsilon$

```
def rnm(q):
  i, best, r = 0

  while i < |q|
    d ~ Lap(q[i], 2/ε)

    if d > best || i = 0
      r = i
      best = d

    i = i + 1

  return r
```

[dwork & roth 14]

q1 = [9, 0]   q2 = [10, 1]

w = 0

r1 = 0   r2 = 0

r1 = 0   r2 = 0
d1 = c   d2 = c

w = ε/2

$r1 = r2 \wedge w \leqslant \varepsilon$

```
def rnm(q):
  i, best, r = 0

  while i < |q|
    d ~ Lap(q[i], 2/ε)

    if d > best || i = 0
      r = i
      best = d

    i = i + 1

  return r
```

[dwork & roth 14]

```
q1 = [9, 0]          q2 = [10, 1]
           w = 0

r1 = 0               r2 = 0


r1 = 0               r2 = 0
d1 = c               d2 = c

          w = ε/2

             ⋮

           w = ε
```

$r1 = r2 \wedge w \leqslant \varepsilon$

our game strategy

in every iteration, couple samples using

$$\{(c, c, \epsilon/2) \mid c \in \mathbb{Z}\}$$

our game strategy

in every iteration, couple samples using

$$\{(c, c, \epsilon/2) \mid c \in \mathbb{Z}\}$$

$$\frac{n \cdot \epsilon}{2}$$ differential privacy

our game strategy

in every iteration, couple samples using

$$\{(c, c, \epsilon/2) \mid c \in \mathbb{Z}\}$$

$$\frac{n \cdot \epsilon}{2}$$ differential privacy

a winning strategy

use this coupling in 1 iteration only

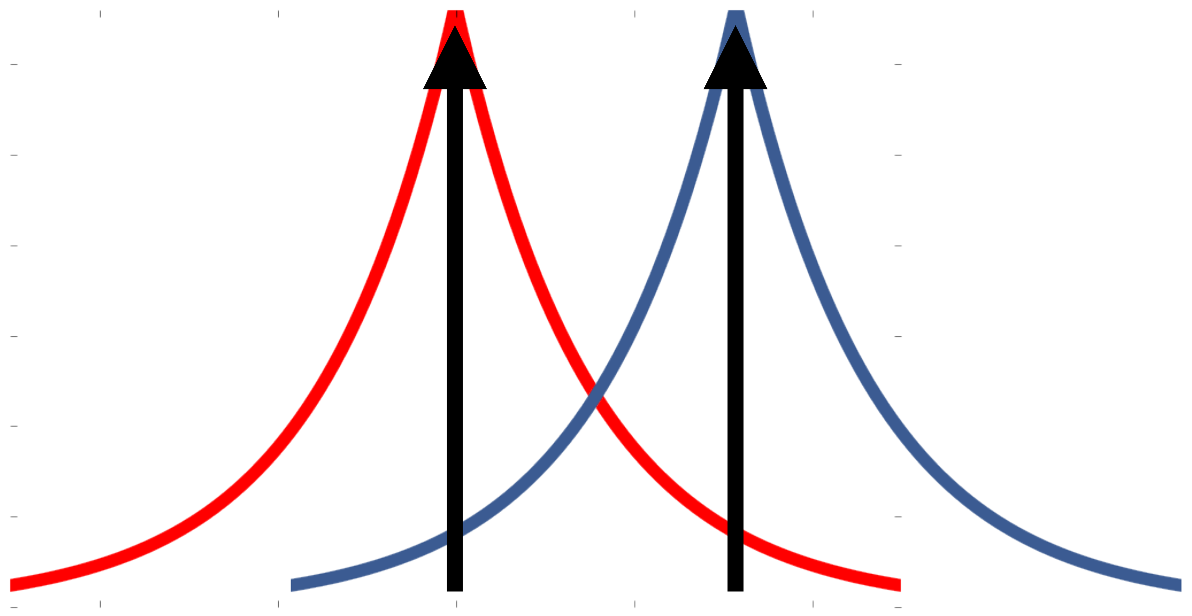
$$\{(c, c+1, \epsilon) \mid c \in \mathbb{Z}\}$$

a winning strategy

use this coupling in 1 iteration only

$$\{(c, c+1, \epsilon) \mid c \in \mathbb{Z}\}$$

in all other iterations pay **zero cost**



winning strategies are programs

if *condition*
 use *coupling C1*
else
 use *coupling C2*

evaluation

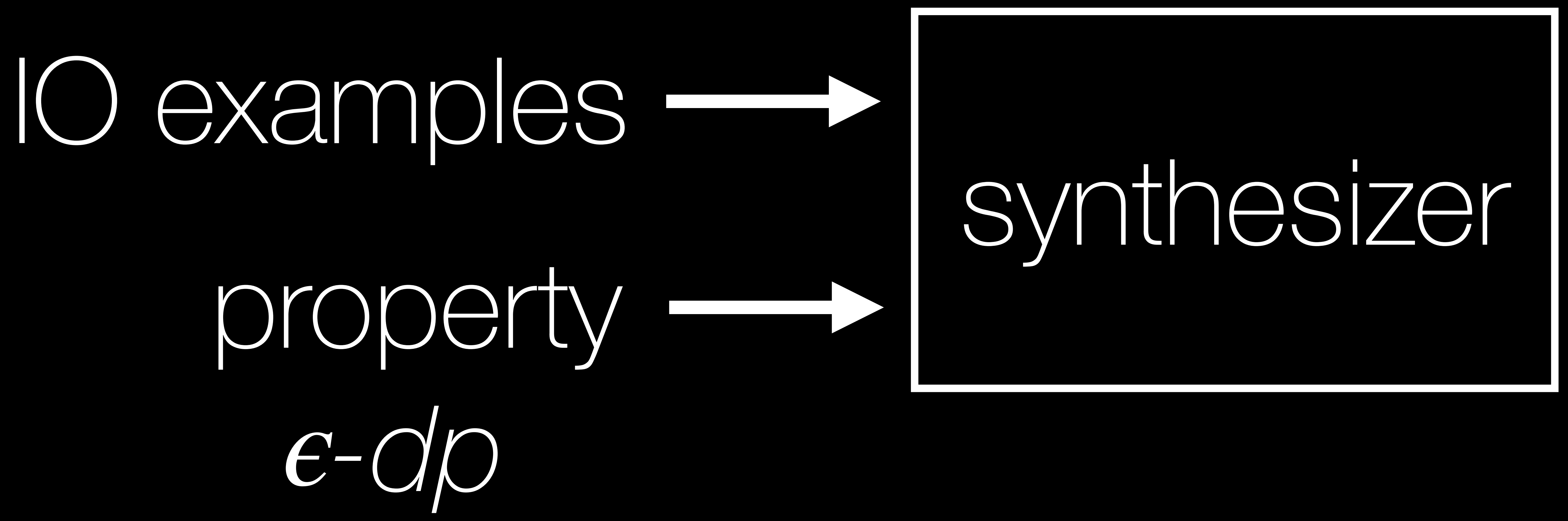
PartialSum	Compute the noisy sum of a list of queries.
PrefixSum	Compute the noisy sum for every prefix of a list of queries.
SmartSum	Advanced version of PrefixSum that chunks the list [Chan et al. 2011; Dwork et al. 2010].
ReportNoisyMax	Find the element with the highest quality score [Dwork and Roth 2014].
ExpMech	Variant of ReportNoisyMax using the exponential distribution [Dwork and Roth 2014; McSherry and Talwar 2007].
AboveThreshold	Find the index of the first query above threshold [Dwork and Roth 2014].
AboveThresholdN	Find the indices of the first N queries with answer above threshold [Dwork and Roth 2014; Lyu et al. 2017].
NumericSparse	Return the index and answer of the first query above threshold [Dwork and Roth 2014].
NumericSparseN	Return the indices and answers of the first N queries above threshold [Dwork and Roth 2014; Lyu et al. 2017].

…and more!

1 automatic proofs of accuracy [POPL19]

2 automatic proofs of differential privacy [POPL18]

theme get rid of probability! long live logic!


IO examples
property
ϵ-*dp*
synthesizer


IO examples
property
ϵ-*dp*
synthesizer
algorithm
+
proof

```
function IDC
  (iter : Nat[i]) (eps : num[e])
  (db :[2 * i * e] db_type) (qs : query bag)
  (PA : (query bag) -> approx_db
      -> db_type -o[e] Circle query)
  (DUA : approx_db -> query -> num -> approx_db)
  (eval_q : query -> db_type -o[1] num)
  : Circle approx_db {
  case iter of
    0     => return init_approx
  | n + 1 =>
      sample approx = (IDC n eps db qs PA DUA);
      sample q = PA qs approx db;
      sample actual = add_noise eps (eval_q q db);
      return (DUA approx q actual)
  }
```

Figure 11. Iterative Database Function in *DFuzz*

[Gupta, Roth, Ullman, TCC 2012]

```
function IDC
  (iter : Nat[i]) (eps : num[e])
  (db :[2 * i * e] db_type) (qs : query bag)
  (PA : (query bag) -> approx_db
      -> db_type -o[e] Circle query)
  (DUA : approx_db -> query -> num -> approx_db)
  (eval_q : query -> db_type -o[1] num)
  : Circle approx_db {
  case iter of
    0      => return init_approx
  | n + 1 =>
      sample approx = (IDC n eps db qs PA DUA);
      sample q = PA qs approx db;
      sample actual = add_noise eps (eval_q q db);
      return (DUA approx q actual)
  }
```

Figure 11. Iterative Database Function in *DFuzz*

[Gupta, Roth, Ullman, TCC 2012]