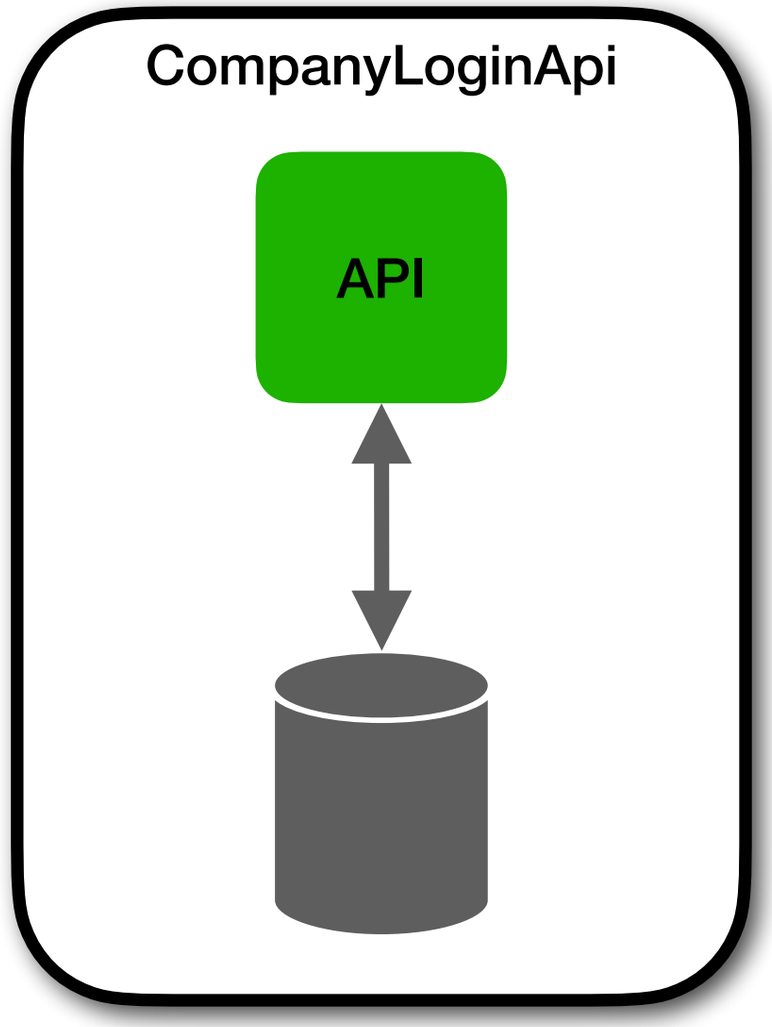
A Gradual Probabilistic Lambda Calculus

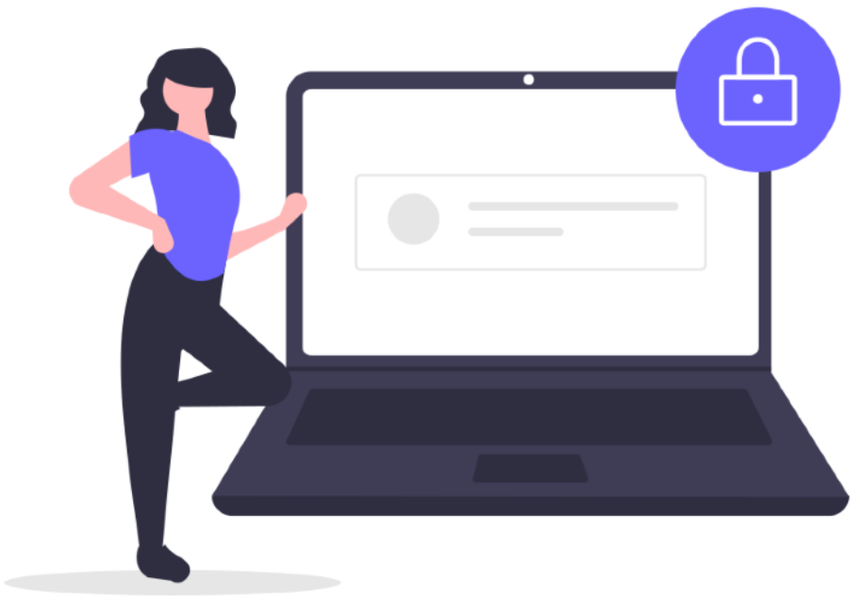
Wenjia Ye Matías Toro Federico Olmedo

香 港 大 學
THE UNIVERSITY OF HONG KONG

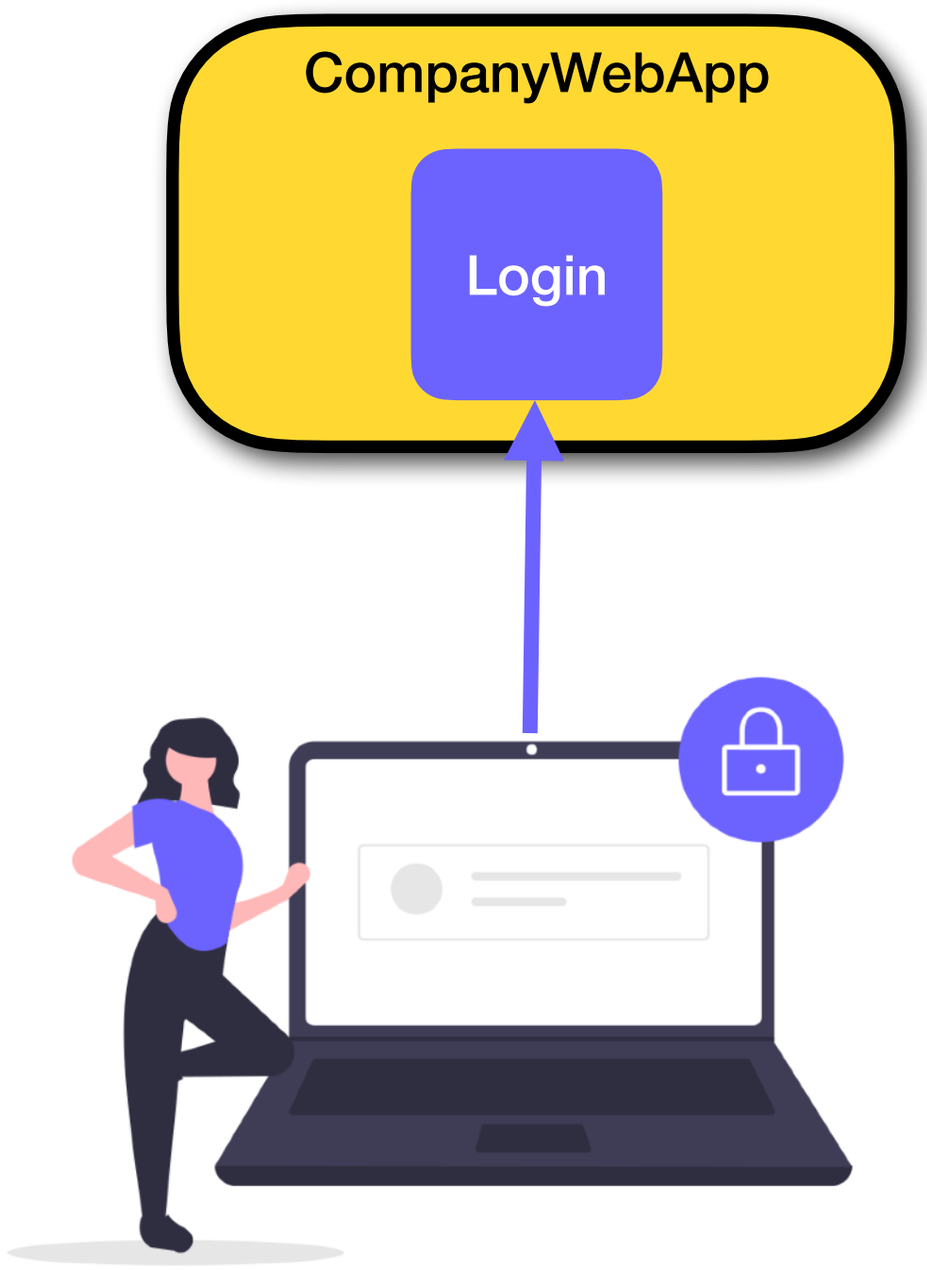
UNIVERSIDAD DE CHILE

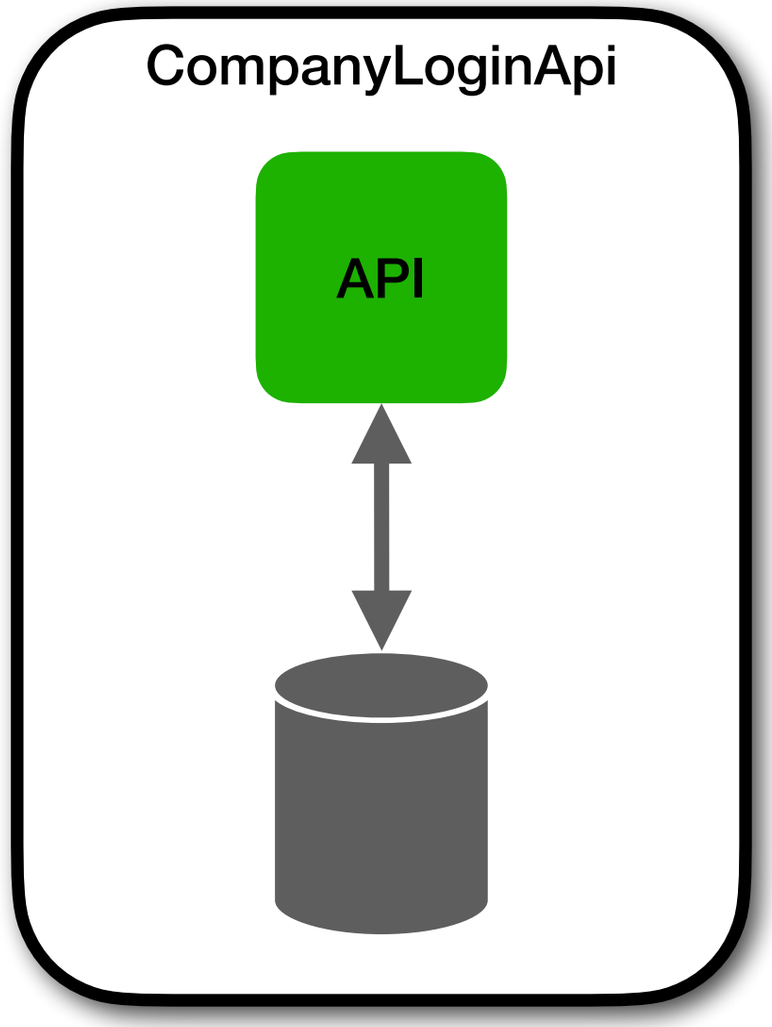
A Simple App with Authentication



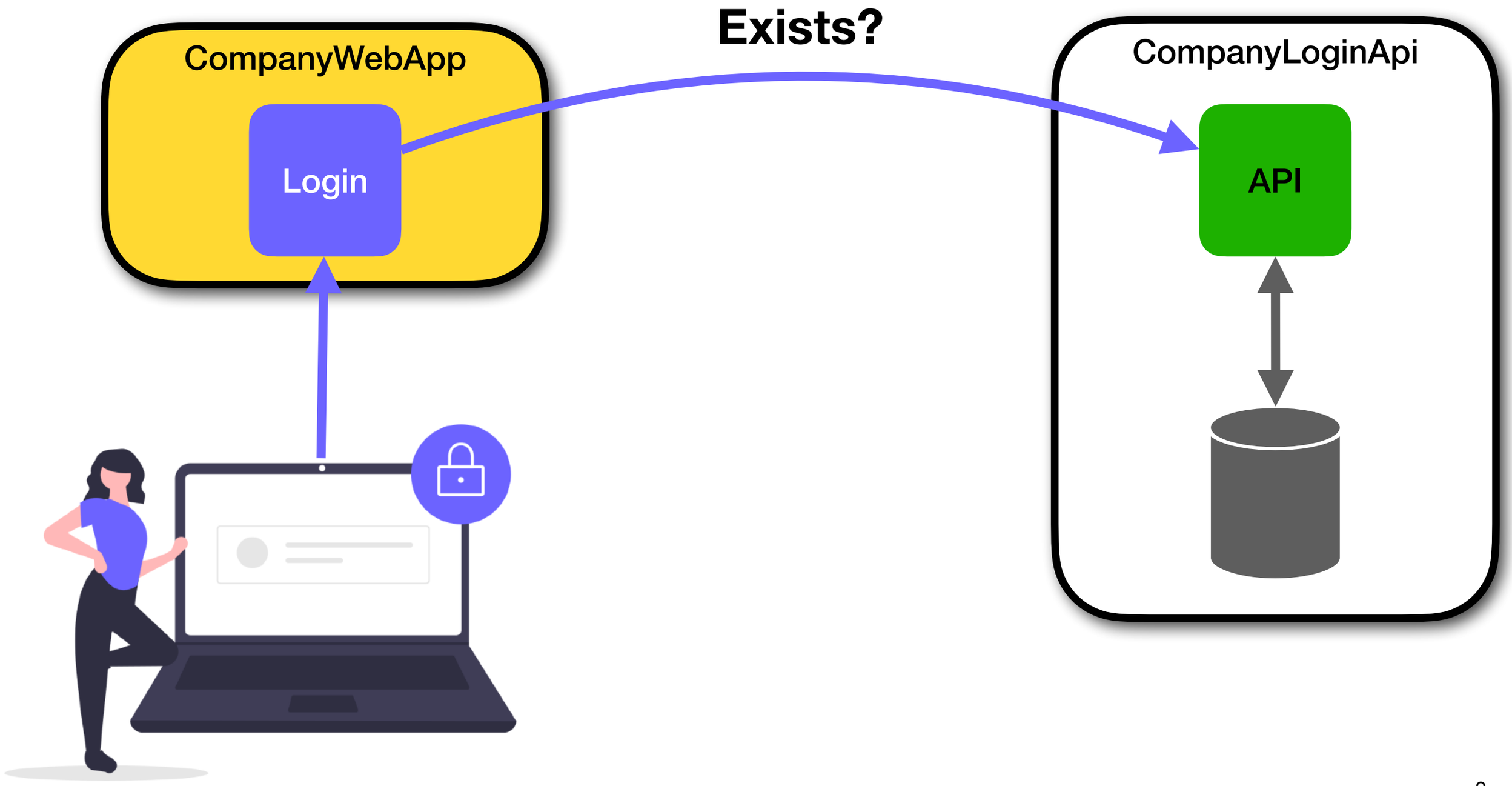


A Simple App with Authentication

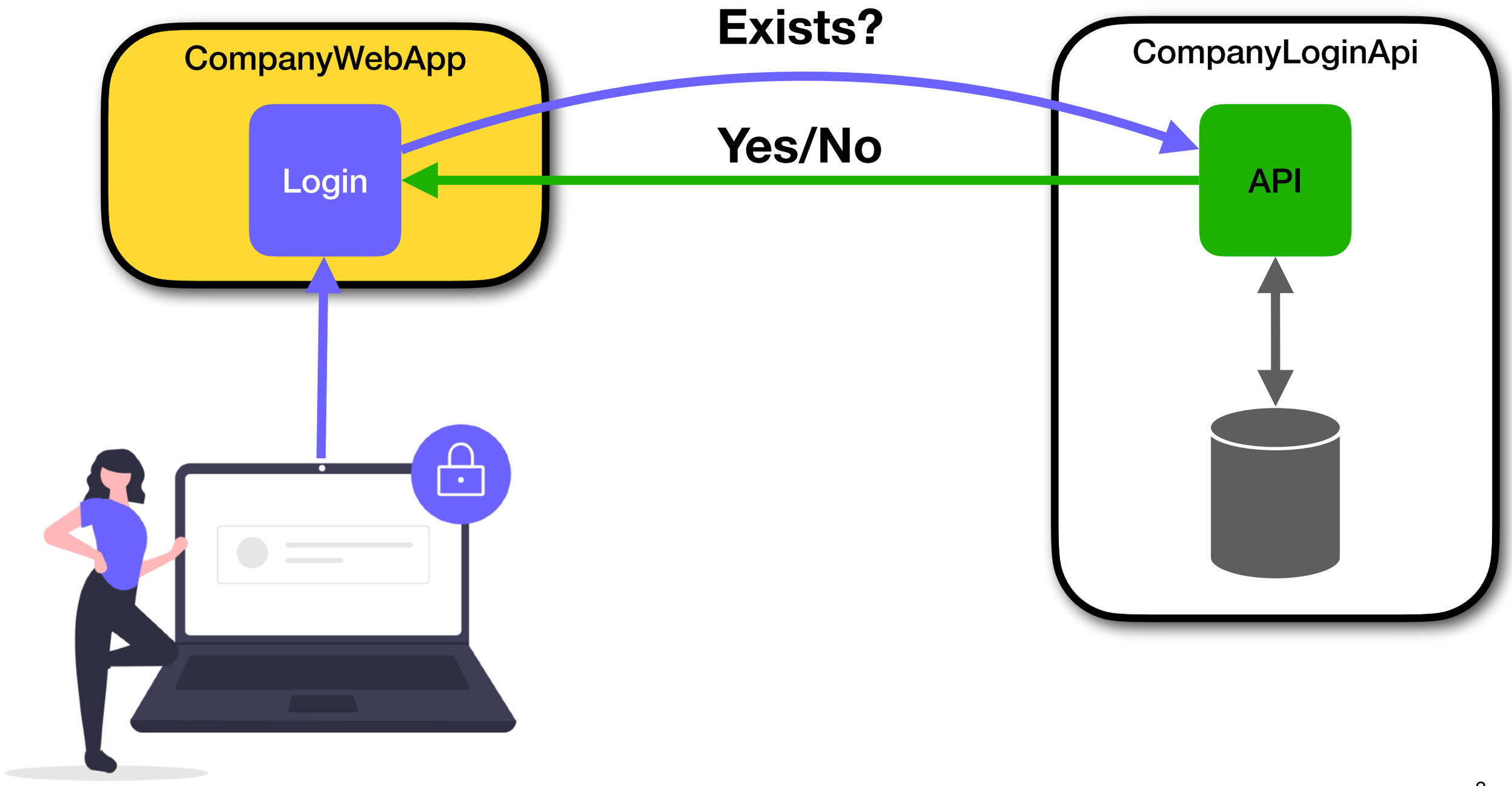




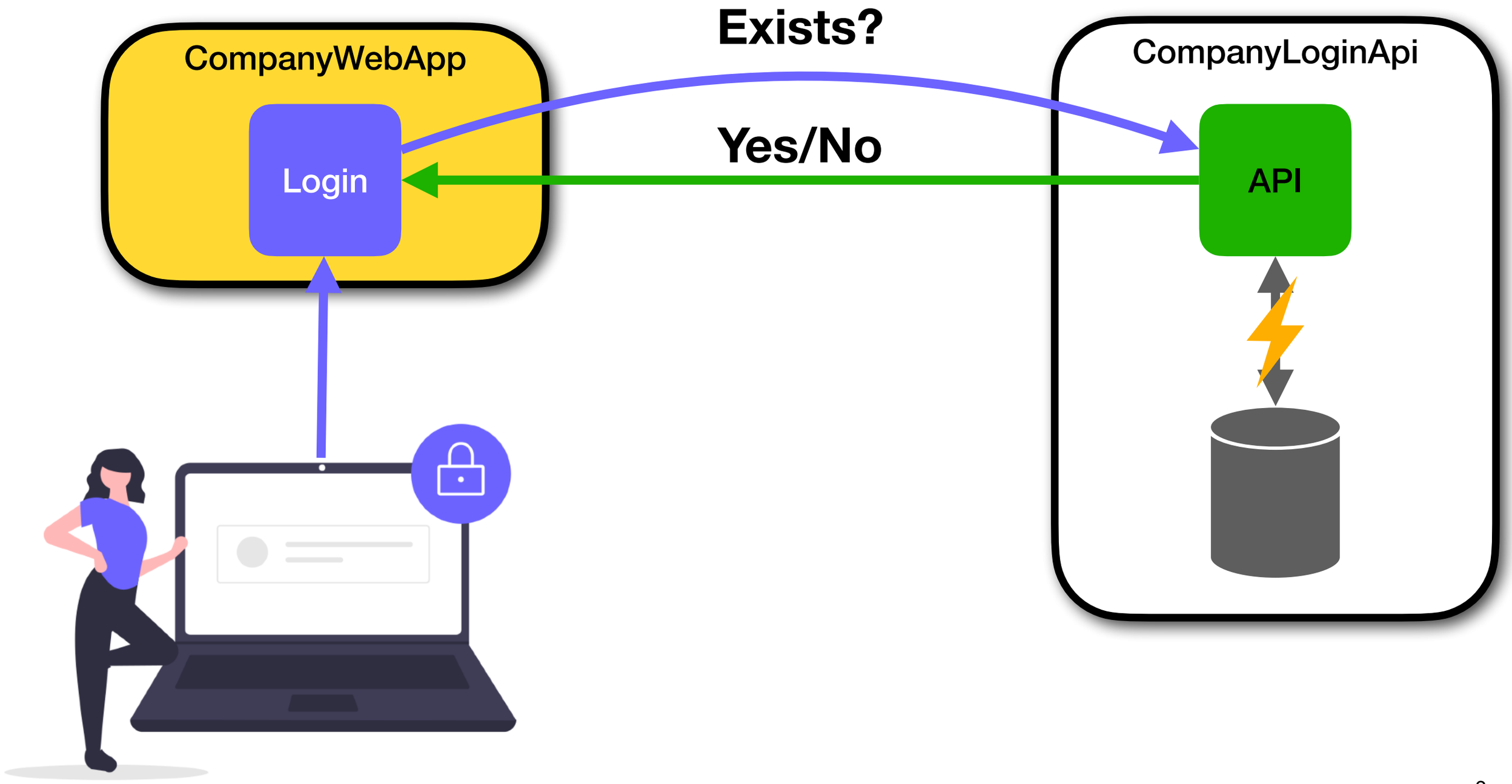
A Simple App with Authentication



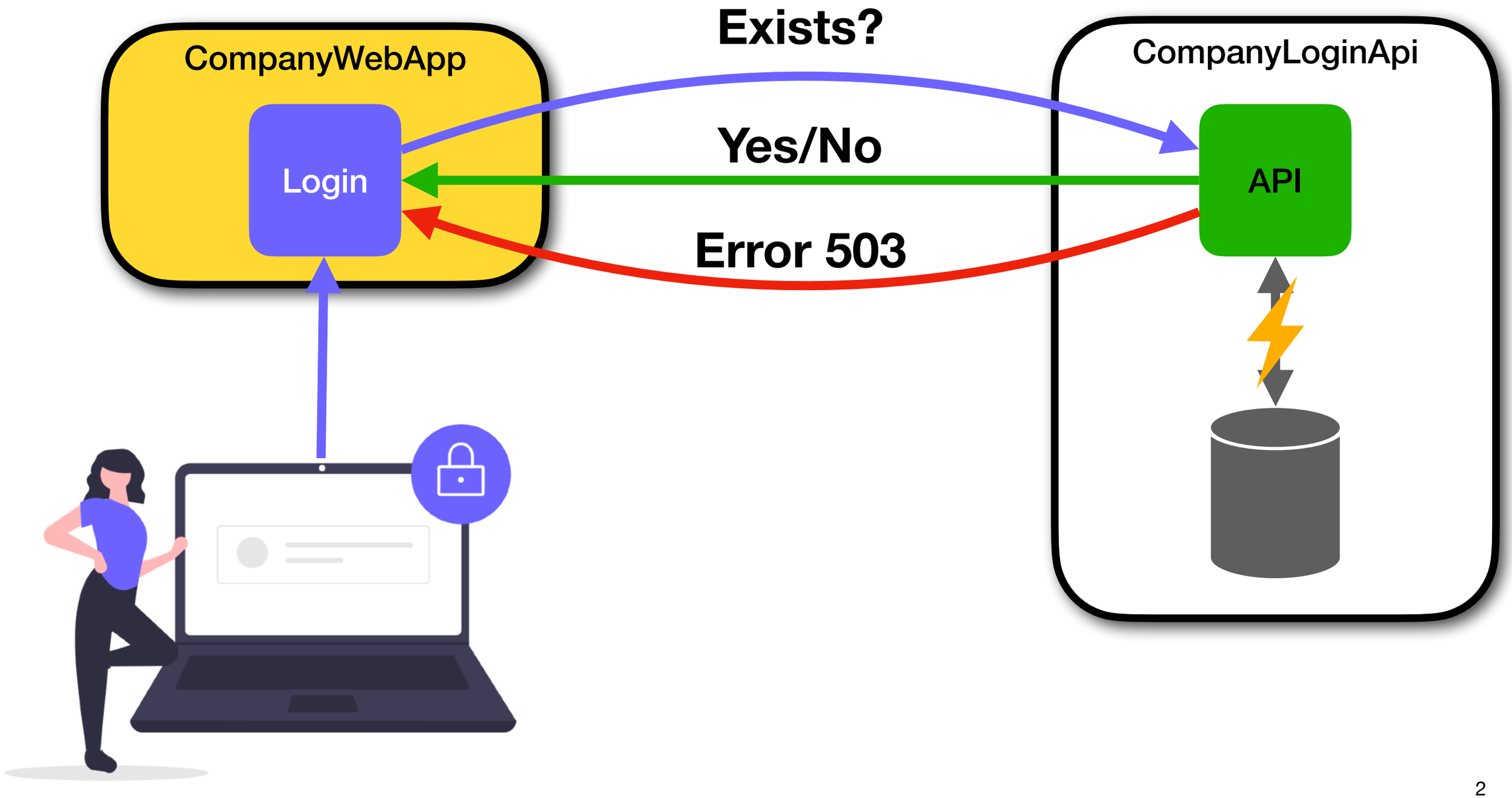
A Simple App with Authentication



A Simple App with Authentication



A Simple App with Authentication



Let’s zoom over the Login module

Development mode vs production mode

Let’s zoom over the Login module

Development mode vs production mode

```
login("Bob", "mypassword")
```

Let’s zoom over the Login module

Development mode vs production mode

```
login("Bob", "mypassword")
```

```
def isUserActive(user) = {
  if (prod)
    externalCall(user)
  else
    localCall(user)
}
def login(user, pass) = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

Let’s zoom over the Login module

Development mode vs production mode

```
login("Bob", "mypassword")
```

```
def isUserActive(user) = {
  if (prod)
    externalCall(user)
  else
    localCall(user)
}
def login(user, pass) = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

We check if the user belongs to the company

Let's zoom over the Login module

Development mode vs production mode

```
login("Bob", "mypassword")
```

```
def isUserActive(user) = {
  if (prod)
    externalCall(user)
  else
    localCall(user)
}
def login(user, pass) = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

If we are in production mode, query the official API

We check if the user belongs to the company

Let's zoom over the Login module

Development mode vs production mode

```
login("Bob", "mypassword")
```

```
def isUserActive(user) = {
  if (prod)
    externalCall(user)
  else
    localCall(user)
}
def login(user, pass) = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

If we are in production mode, query the official API

If we are in development mode, query a Mock implementation

We check if the user belongs to the company

Setting an uptime specification

```
login("Bob", "mypassword")
```

```
def isUserActive(user) = {
  if (prod)
    externalCall(user)
  else
    localCall(user)
}
def login(user, pass) = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

Setting an uptime specification

```
login("Bob", "mypassword")
```

```
def isUserActive(user) = {
  if (prod)
    externalCall(user)
  else
    localCall(user)
}
def login(user, pass) = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

Empirically: 90% uptime

Setting an uptime specification

```
login("Bob", "mypassword")
```

```
def isUserActive(user) = {
  if (prod)
    externalCall(user)
  else
    localCall(user)
}
def login(user, pass) = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

Empirically: 90% uptime

Setting an uptime specification

```
login("Bob", "mypassword")
```

```
def isUserActive(user) = {
  if (prod)
    externalCall(user)
  else
    localCall(user)
}
def login(user, pass) = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

Empirically: 90% uptime

We fail to detect the impossibility to comply with the requirement

Modeling the specification

Probabilistic choice to the rescue

Modeling the specification

Probabilistic choice to the rescue

$$m \bigoplus_p n$$

Modeling the specification

Probabilistic choice to the rescue

$$m \oplus_p n$$

Behaves like m with probability p,
and like n with probability $1 - p$

Modeling the specification

Probabilistic choice to the rescue

$$m \oplus_p n$$

Behaves like m with probability p,
and like n with probability $1 - p$

$$\{\overline{\tau_i^{pi}}\}$$

Modeling the specification

Probabilistic choice to the rescue

$$m \oplus_p n$$

Behaves like m with probability p, and like n with probability $1 - p$

$$\{\overline{\tau_i^{p_i}}\}$$

Type τ_i occurs with probability p_i

Modeling the specification

Probabilistic choices + types to the rescue

$$m \bigoplus_p n$$

$$\{\overline{\tau_i^{pi}}\}$$

Modeling the specification

Probabilistic choices + types to the rescue

$$m \oplus_p n$$

$$\{\overline{\tau_i^{pi}}\}$$

```
def externalCall(user: String): {Bool^90/100, Error503^10/100} = ...
```

Modeling the specification

Probabilistic choices + types to the rescue

$$m \oplus_p n$$

$$\{\overline{\tau_i^{pi}}\}$$

```
def externalCall(user: String): {Bool^90/100, Error503^10/100} = ...
```

It returns a boolean with 90% probability, and errors with 10% probability

Modeling the specification

Probabilistic choices + types to the rescue

$$m \oplus_p n$$

$$\{\overline{\tau_i^{pi}}\}$$

```
def externalCall(user: String): {Bool^90/100, Error503^10/100} = ...
```

It returns a boolean with 90% probability, and errors with 10% probability

```
def localCall(user: String): {Bool^95/100, Error503^5/100} = {
    true ⊕_95/100 Error503()
}
```

Limitations of Static Typing

Overly conservative (and cumbersome)

```
def externalCall(user: String): {Bool^90/100, Error503^10/100} = …
def localCall(user: String): {Bool^95/100, Error503^5/100} = …

def isUserActive(user: String): {Bool^95/100, Error503^5/100} = {
  if (prod)
    externalCall(user)
  else
    localCall(user)
}
def login(user: String, pass: String): {Bool^95/100, Error503^5/100} = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

Limitations of Static Typing

Overly conservative (and cumbersome)

```
def externalCall(user: String): {Bool^90/100, Error503^10/100} = …
def localCall(user: String): {Bool^95/100, Error503^5/100} = …

def isUserActive(user: String): {Bool^95/100, Error503^5/100} = {
  if (prod)
    externalCall(user)
  else
    localCall(user)
}
def login(user: String, pass: String): {Bool^95/100, Error503^5/100} = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

Type error

Gradual Typing at Rescue

Introducing imprecision on types

```
def externalCall(user: ?): {Bool^90/100, Error503^10/100} = …
def localCall(user: ?): {Bool^95/100, Error503^5/100} = …

def isUserActive(user: ?): ? = {
  if (prod)
    externalCall(user) :: ?
  else
    localCall(user) :: ?
}
def login(user: ?, pass: ?): {Bool^95/100, Error503^5/100} = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

Gradual Typing at Rescue

Introducing imprecision on types

```
def externalCall(user: ?): {Bool^90/100, Error503^10/100} = …
def localCall(user: ?): {Bool^95/100, Error503^5/100} = …

def isUserActive(user: ?): ? = {
  if (prod)
    externalCall(user) :: ?
  else
    localCall(user) :: ?
}
def login(user: ?, pass: ?): {Bool^95/100, Error503^5/100} = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

? can represent any type

Gradual Typing at Rescue

Introducing imprecision on types

```
def externalCall(user: ?): {Bool^90/100, Error503^10/100} = …
def localCall(user: ?): {Bool^95/100, Error503^5/100} = …

def isUserActive(user: ?): ? = {
  if (prod)
    externalCall(user) :: ?
  else
    localCall(user) :: ?
}
def login(user: ?, pass: ?): {Bool^95/100, Error503^5/100} = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

? can represent any type

Type checks!!

Gradual Typing at Rescue

Introducing imprecision on types

```
def externalCall(user: ?): {Bool^90/100, Error503^10/100} = …
def localCall(user: ?): {Bool^95/100, Error503^5/100} = …

def isUserActive(user: ?): ? = {
  if (prod)
    externalCall(user) :: ?
  else
    localCall(user) :: ?
}
def login(user: ?, pass: ?): {Bool^95/100, Error503^5/100} = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

Gradual Typing at Rescue

Introducing imprecision on types

```
def externalCall(user: ?): {Bool^{90/100}, Error503^{10/100}} = …
def localCall(user: ?): {Bool^{95/100}, Error503^{5/100}} = …

def isUserActive(user: ?): ? = {
  if (prod)
    externalCall(user) :: ?
  else
    localCall(user) :: ?
}
def login(user: ?, pass: ?): {Bool^{95/100}, Error503^{5/100}} = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

If `prod==false`, it runs successfully

Gradual Typing at Rescue

Introducing imprecision on types

```
def externalCall(user: ?): {Bool^90/100, Error503^10/100} = …
def localCall(user: ?): {Bool^95/100, Error503^5/100} = …

def isUserActive(user: ?): ? = {
  if (prod)
    externalCall(user) :: ?
  else
    localCall(user) :: ?
}
def login(user: ?, pass: ?): {Bool^95/100, Error503^5/100} = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

If `prod==true`, it fails on runtime

If `prod==false`, it runs successfully

Gradual Typing at Rescue

Consistency is not transitive

$$\{\mathsf{Bool}^{\frac{90}{100}}, \mathsf{Error503}^{\frac{10}{100}}\} \sim ?$$

$$? \sim \{\mathsf{Bool}^{\frac{95}{100}}, \mathsf{Error503}^{\frac{5}{100}}\}$$

but

$$\{\mathsf{Bool}^{\frac{90}{100}}, \mathsf{Error503}^{\frac{10}{100}}\} \not\sim \{\mathsf{Bool}^{\frac{95}{100}}, \mathsf{Error503}^{\frac{5}{100}}\}$$

Gradual Typing at Rescue

Increasing precision can make the term ill-typed

```
def externalCall(user: ?): {Bool^{90/100}, Error503^{10/100}} = …
def localCall(user: ?): {Bool^{95/100}, Error503^{5/100}} = …

def isUserActive(user: ?): {Bool^{90/100}, Error503^{10/100}} = {
  if (prod)
    externalCall(user) :: ?
  else
    localCall(user) :: ?
}
def login(user: ?, pass: ?): {Bool^{95/100}, Error503^{5/100}} = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

Gradual Typing at Rescue

Increasing precision can make the term ill-typed

```
def externalCall(user: ?): {Bool^{90/100}, Error503^{10/100}} = …
def localCall(user: ?): {Bool^{95/100}, Error503^{5/100}} = …

def isUserActive(user: ?): {Bool^{90/100}, Error503^{10/100}} = {
  if (prod)
    externalCall(user) :: ?
  else
    localCall(user) :: ?
}
def login(user: ?, pass: ?): {Bool^{95/100}, Error503^{5/100}} = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

Gradual Typing at Rescue

Increasing precision can make the term ill-typed

```
def externalCall(user: ?): {Bool^{90/100}, Error503^{10/100}} = …
def localCall(user: ?): {Bool^{95/100}, Error503^{5/100}} = …

def isUserActive(user: ?): {Bool^{90/100}, Error503^{10/100}} = {
  if (prod)
    externalCall(user) :: ?
  else
    localCall(user) :: ?
}
def login(user: ?, pass: ?): {Bool^{95/100}, Error503^{5/100}} = {
  if (isUserActive(user))
    user.pass == pass
  else
    false
}
```

Now, it doesn’t type check!!

Imprecise Probabilities

What if we want to specify that `externalCall` have an uptime of ***at least*** 95%?

Imprecise Probabilities

What if we want to specify that `externalCall` have an uptime of ***at least*** 95%?

$$\{\mathrm{Bool}^{\frac{95}{100}}, \mathrm{Bool}^{?}, \mathrm{Error503}^{?}\}$$

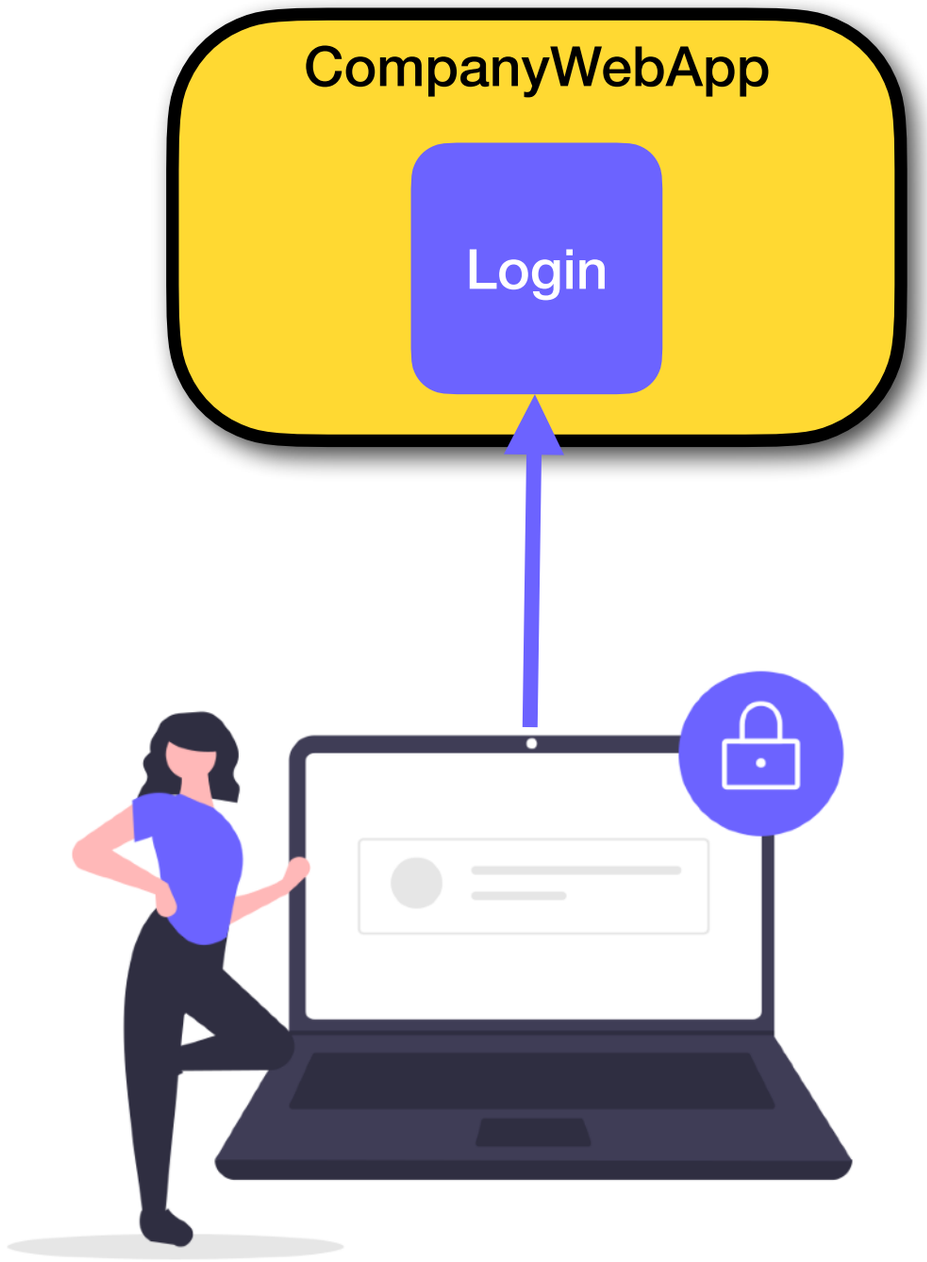
Imprecise Probabilities

What if we want to specify that `externalCall` have an uptime of ***at least*** 95%?

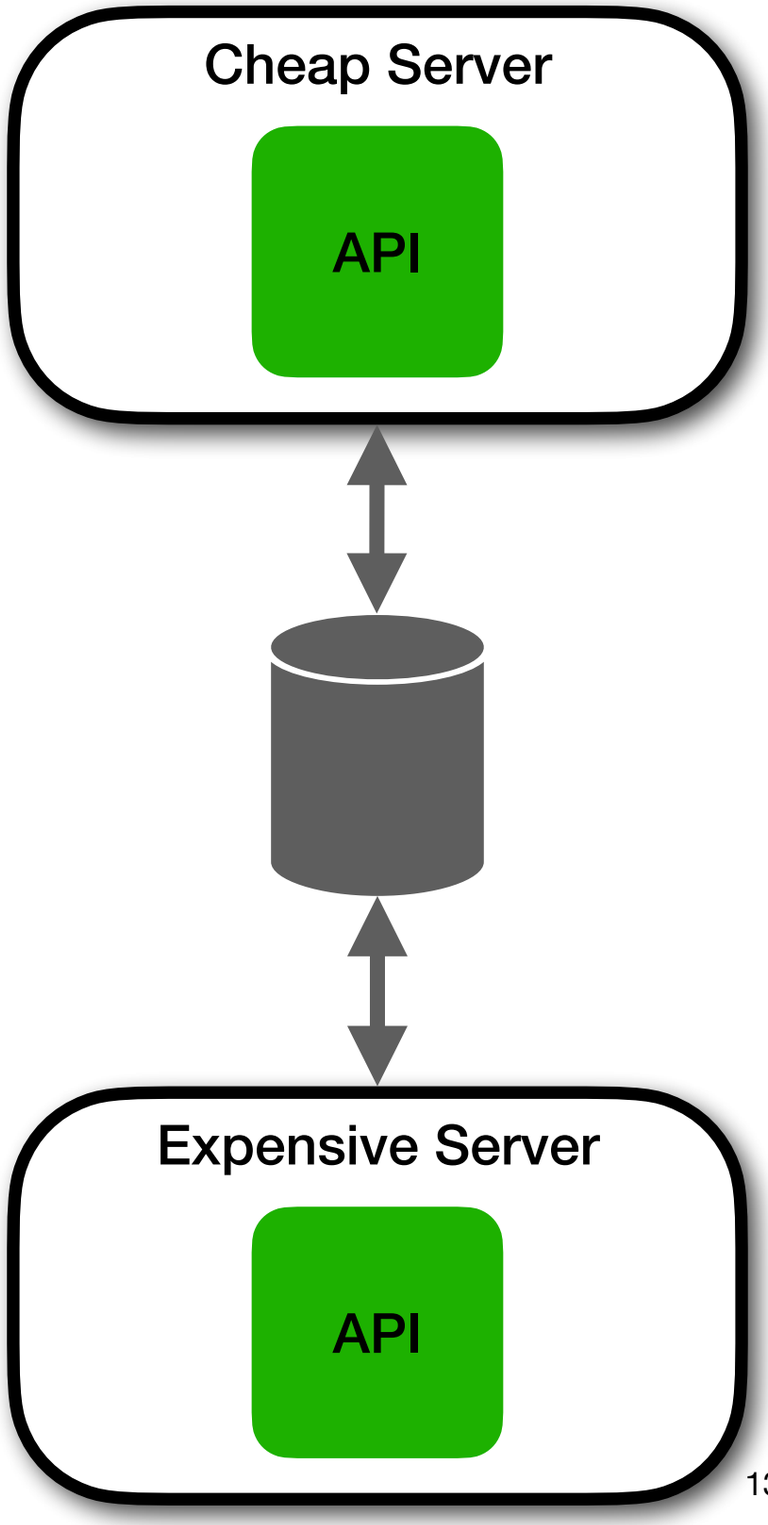
Unknown probability, but $\frac{95}{100} + ? + ? = 1$

$$\{\text{Bool}^{\frac{95}{100}}, \text{Bool}^{?}, \text{Error503}^{?}\}$$

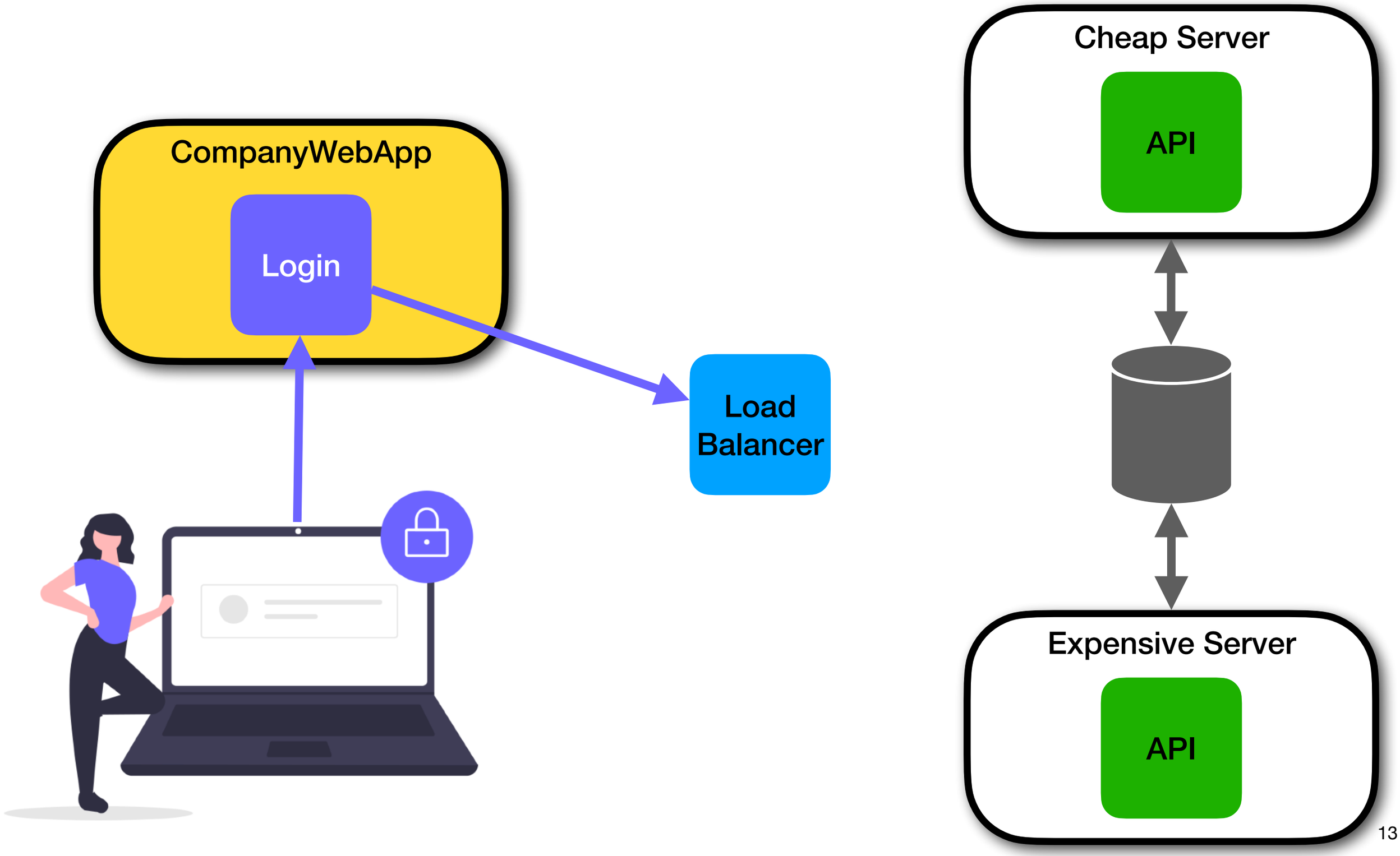
Synthesizing Probabilities



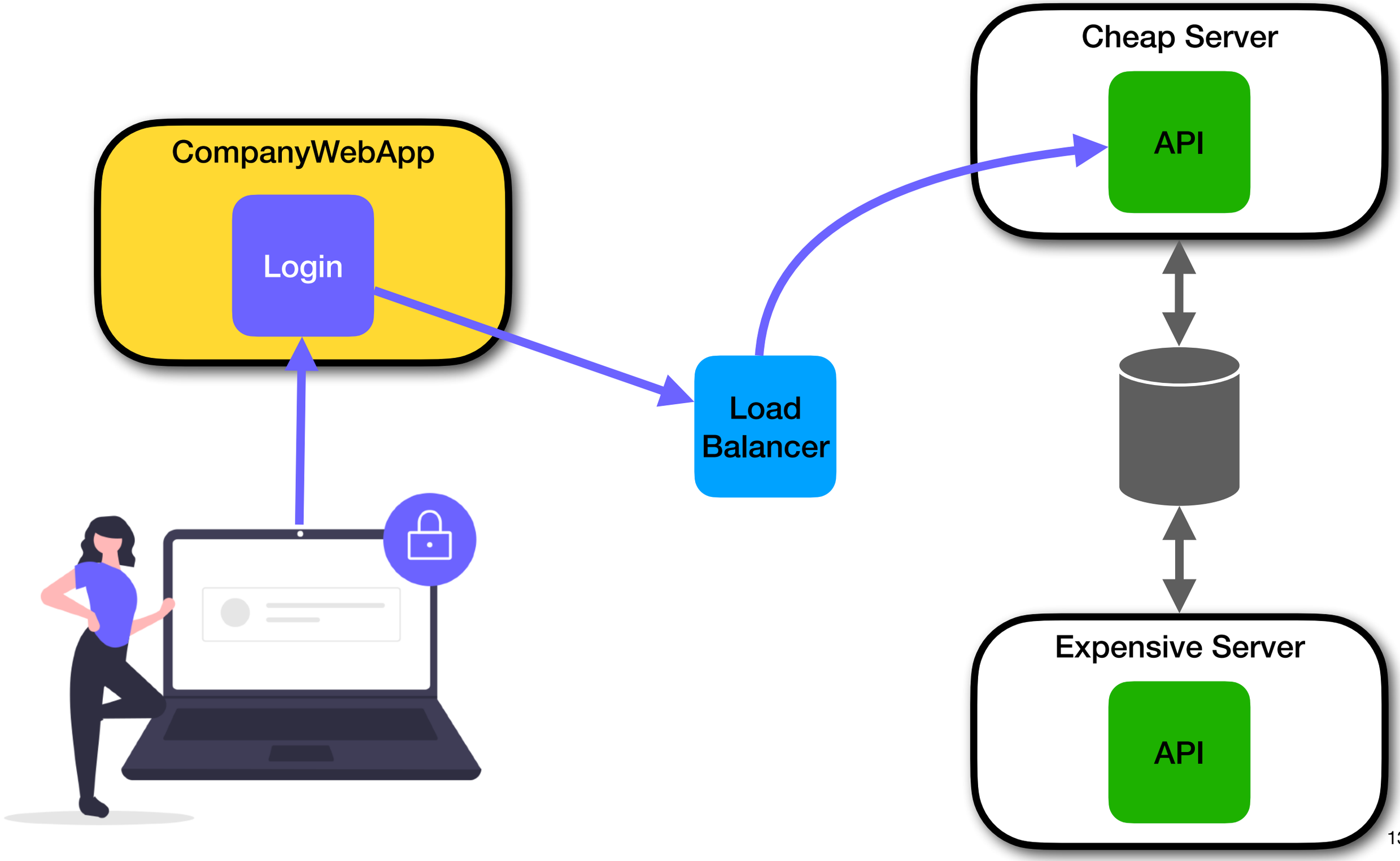
Load Balancer



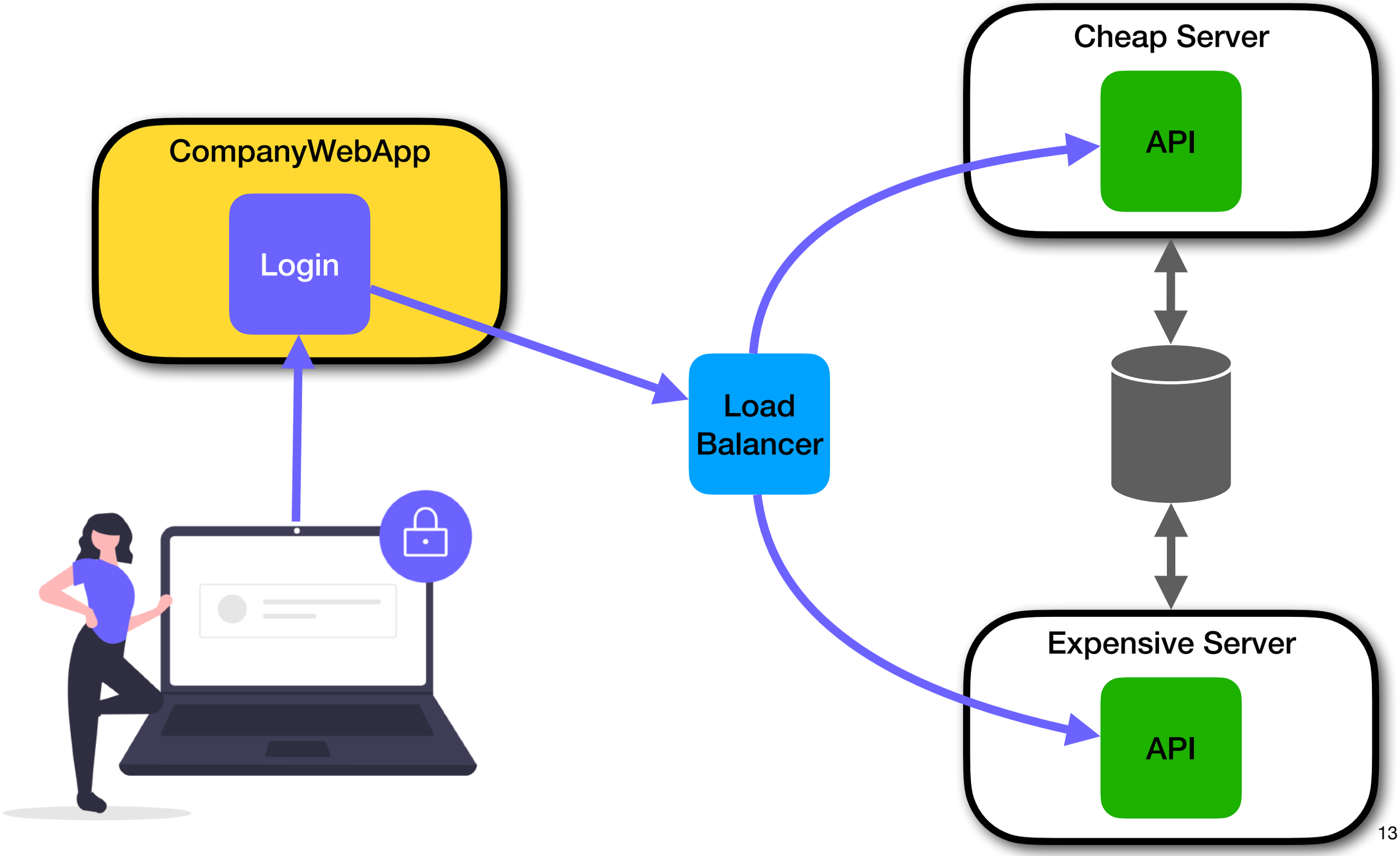
Synthesizing Probabilities



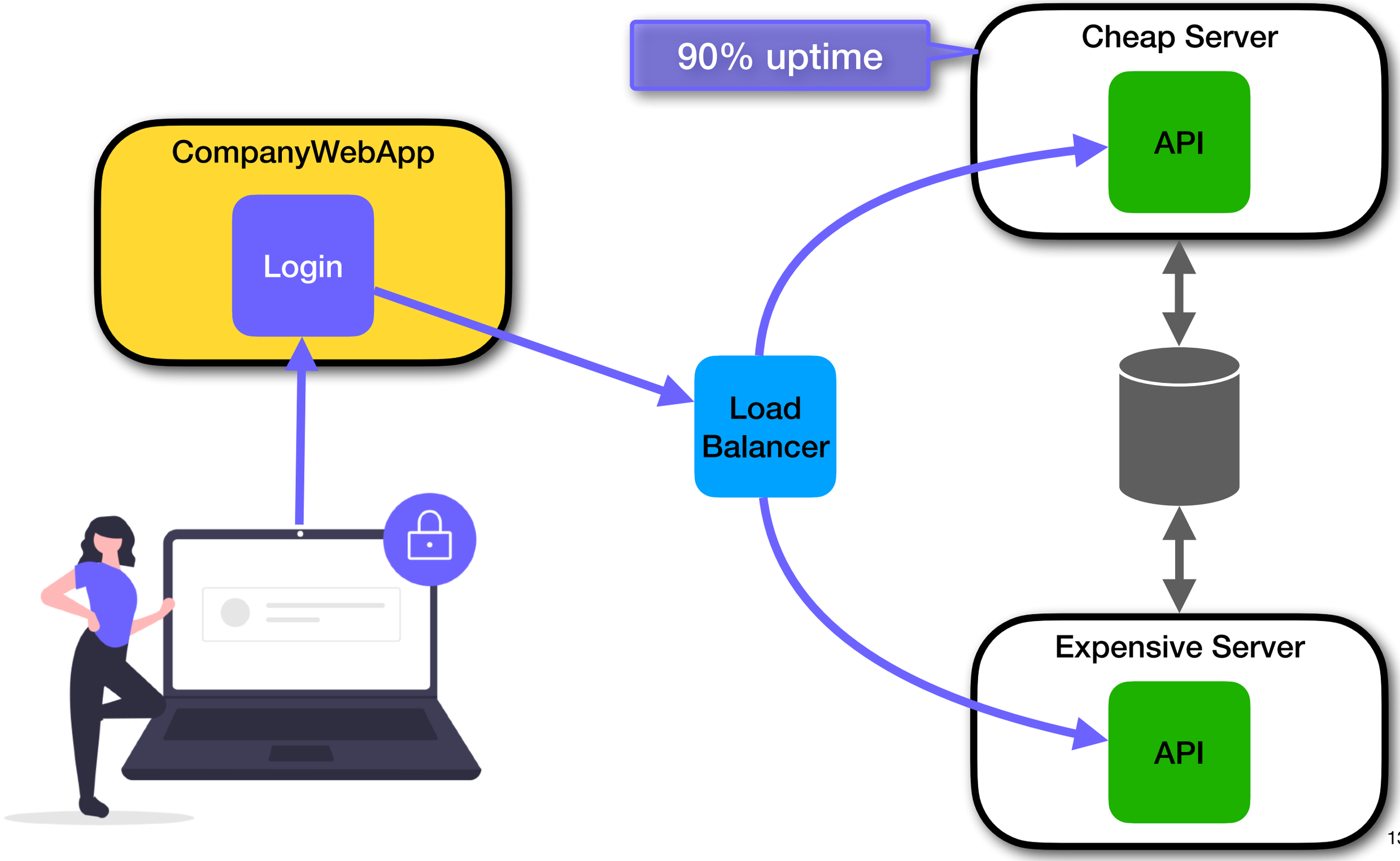
Synthesizing Probabilities



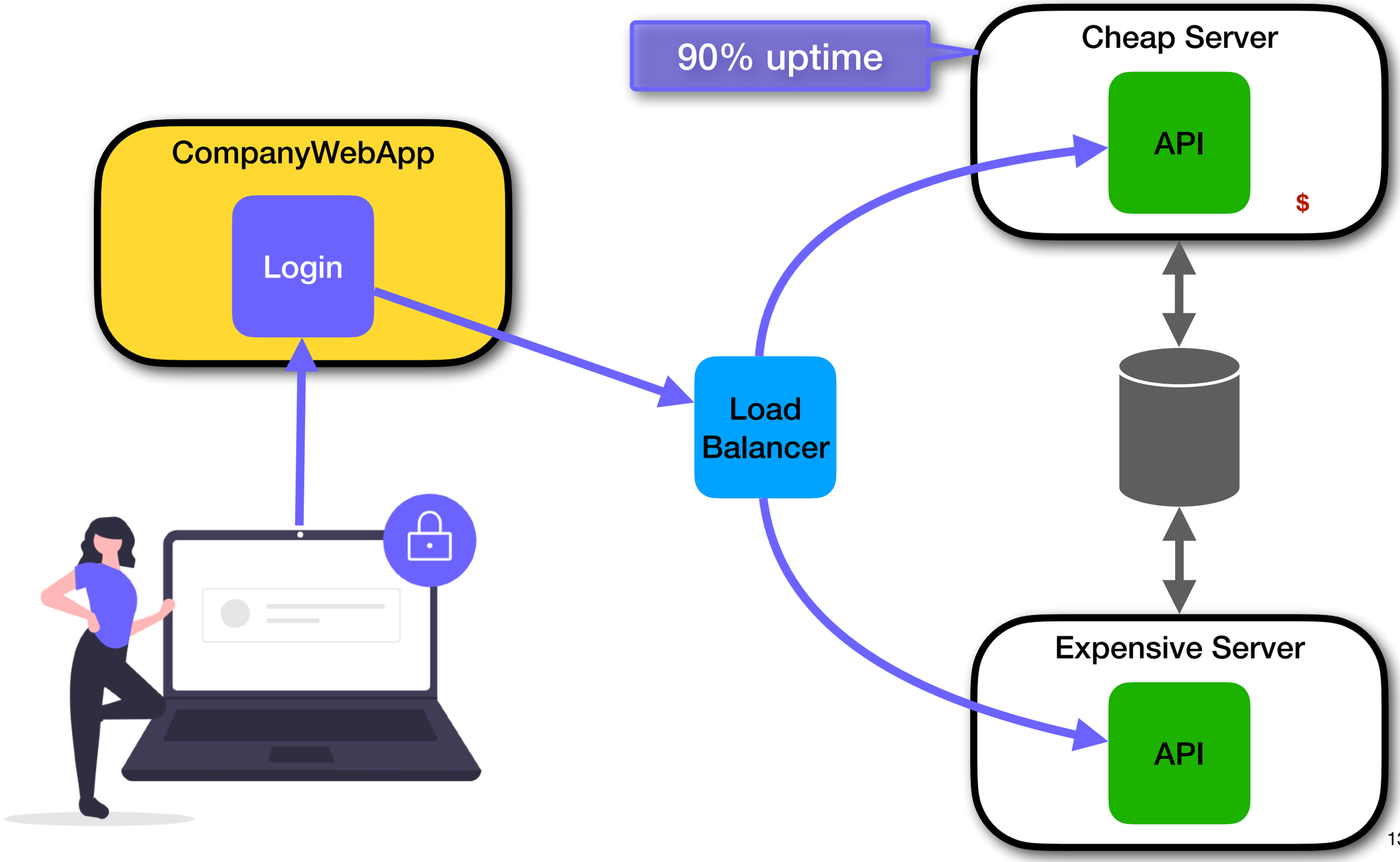
Synthesizing Probabilities



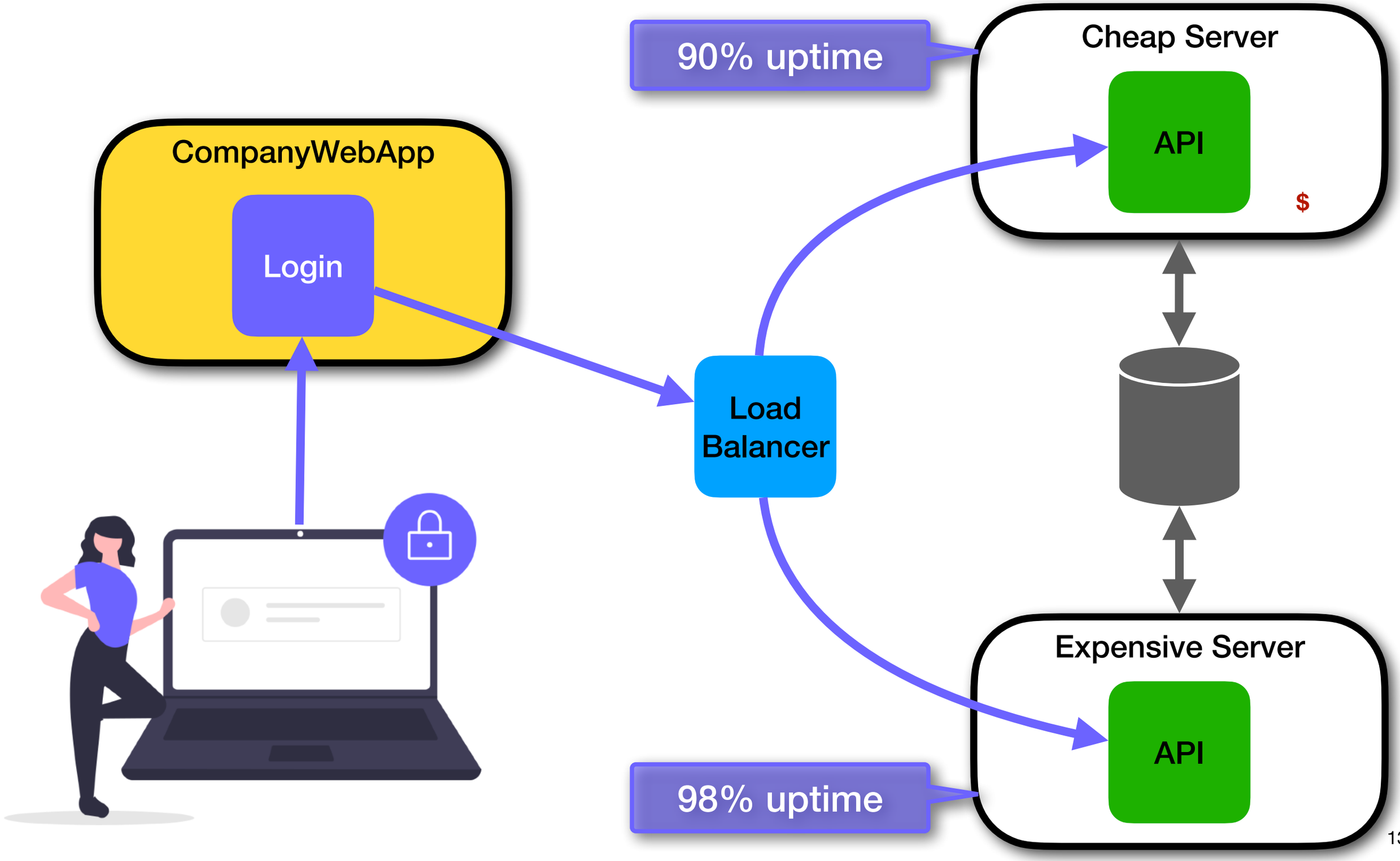
Synthesizing Probabilities



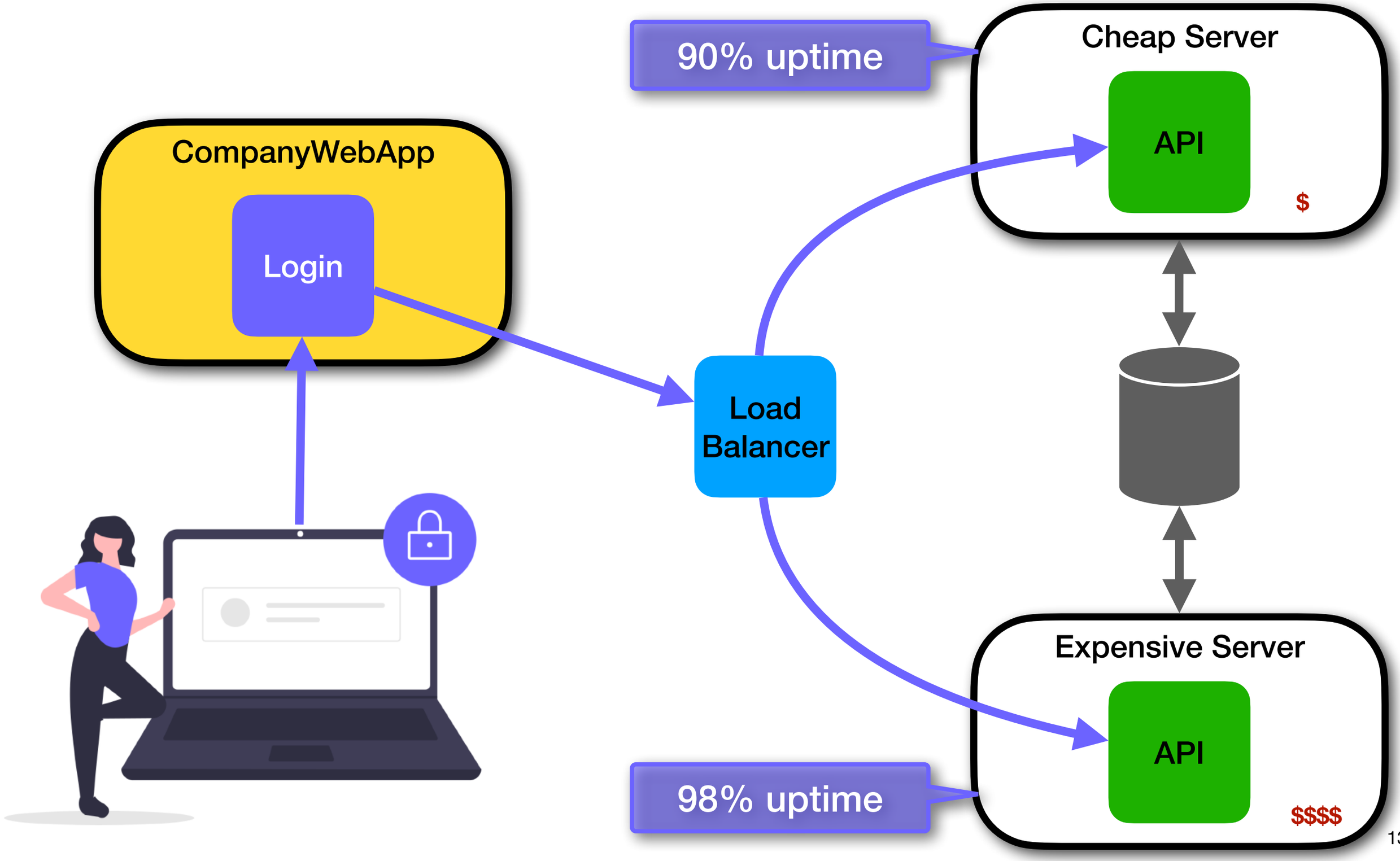
Synthesizing Probabilities



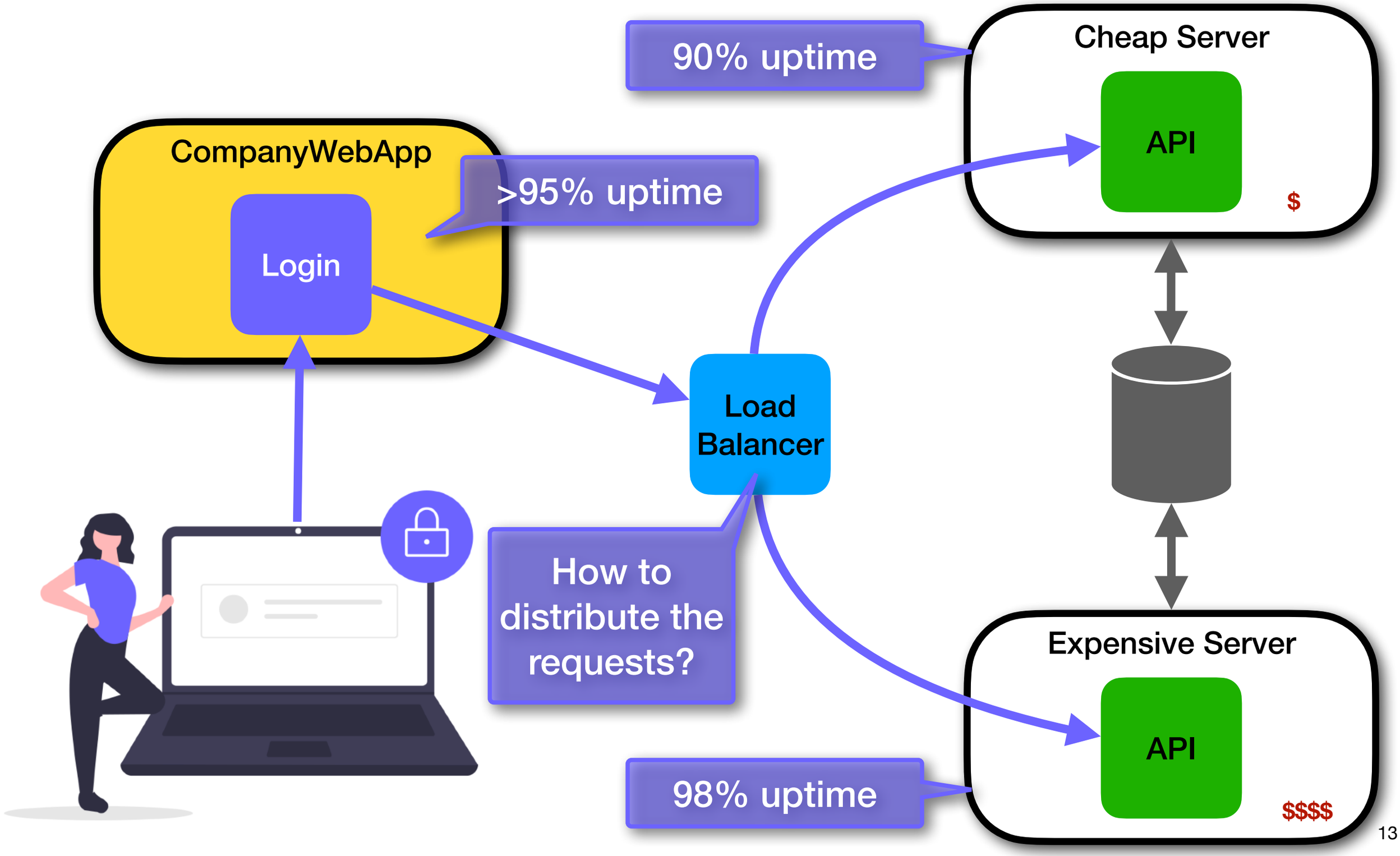
Synthesizing Probabilities



Synthesizing Probabilities



Synthesizing Probabilities



Synthesizing Probabilities

```
val cheapServer: ? → {Bool^90/100, Error503^10/100} = ...
val expensiveServer: ? → {Bool^98/100, Error503^2/100} = ...

def externalCall(user: ?): {Bool^95/100, Bool^?, Error503^?} = {
  cheapServer(user) ⊕? expensiveServer(user)
}
```

Synthesizing Probabilities

```
val cheapServer: ? → {Bool^{90/100}, Error503^{10/100}} = ...
val expensiveServer: ? → {Bool^{98/100}, Error503^{2/100}} = ...

def externalCall(user: ?): {Bool^{95/100}, Bool^?, Error503^?} = {
  cheapServer(user) ⊕? expensiveServer(user)
}
```

Type checks!

Synthesizing Probabilities

```
val cheapServer: ? → {Bool^{90/100}, Error503^{10/100}} = ...
val expensiveServer: ? → {Bool^{98/100}, Error503^{2/100}} = ...

def externalCall(user: ?): {Bool^{95/100}, Bool^?, Error503^?} = {
  cheapServer(user) ⊕? expensiveServer(user)
}
```

We use *symbolic variables* to represent unknown probabilities

Type checks!

Couplings as a key center piece

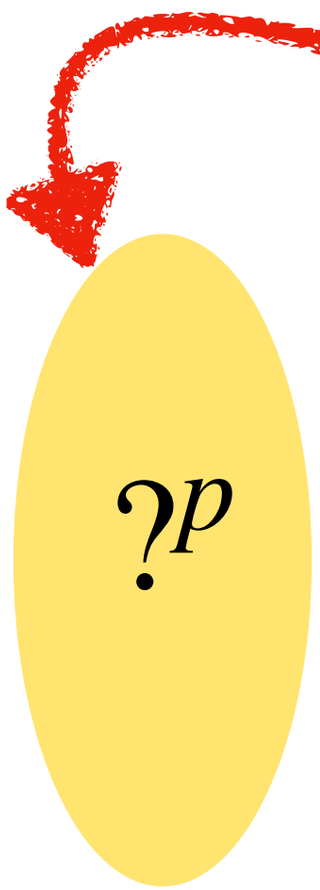
Plausibility = couplings + constraints + satisfiability

$$\{?^{?}\} \sim \{\mathbb{R}^{\frac{1}{2}}, \mathbb{B}^{\frac{1}{2}}\}$$

Couplings as a key center piece

Plausibility = couplings + constraints + satisfiability

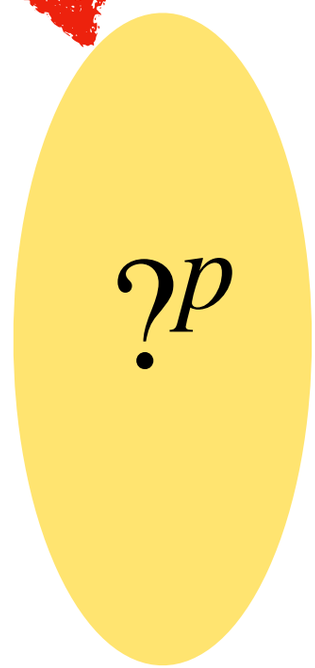
$$\{?^{?}\} \sim \{\mathbb{R}^{\frac{1}{2}}, \mathbb{B}^{\frac{1}{2}}\}$$

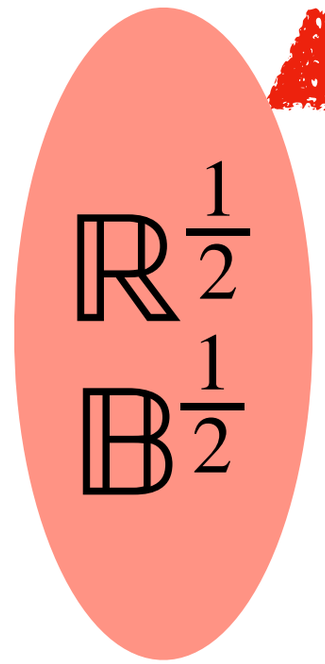


Couplings as a key center piece

Plausibility = couplings + constraints + satisfiability

$$\{?^{?}\} \sim \{\mathbb{R}^{\frac{1}{2}}, \mathbb{B}^{\frac{1}{2}}\}$$

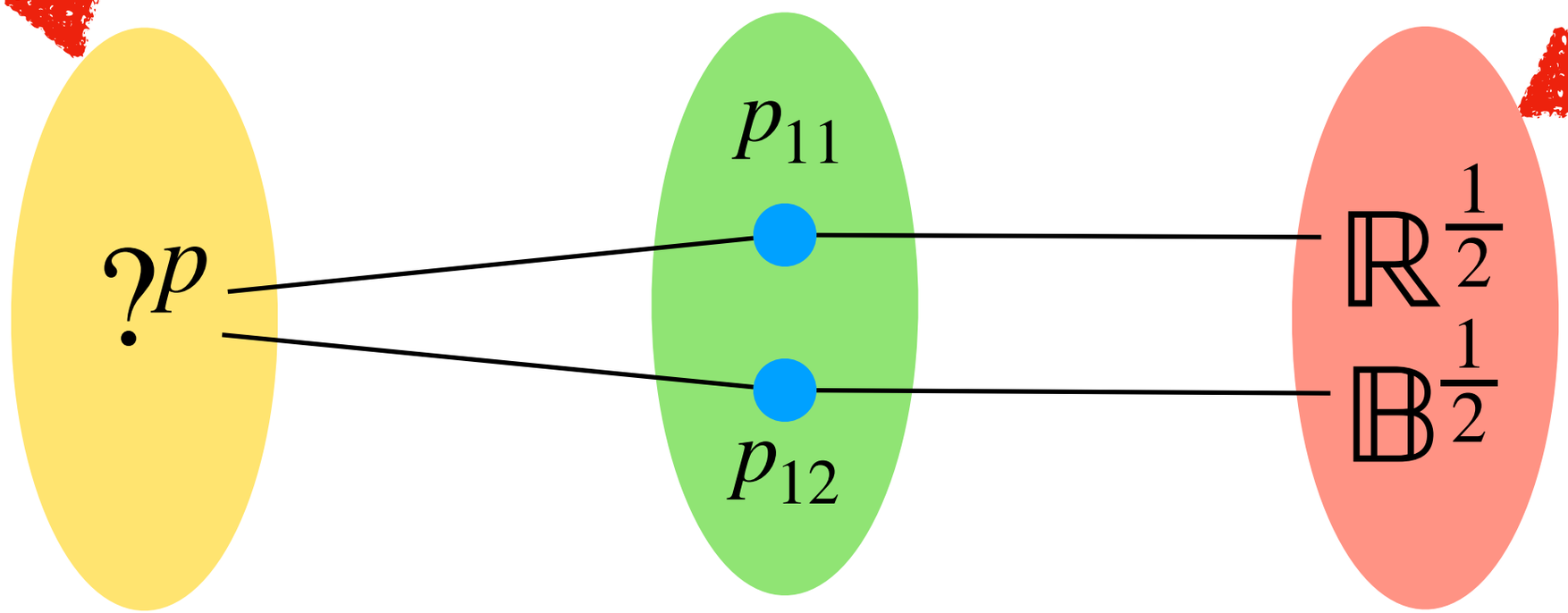




Couplings as a key center piece

Plausibility = couplings + constraints + satisfiability

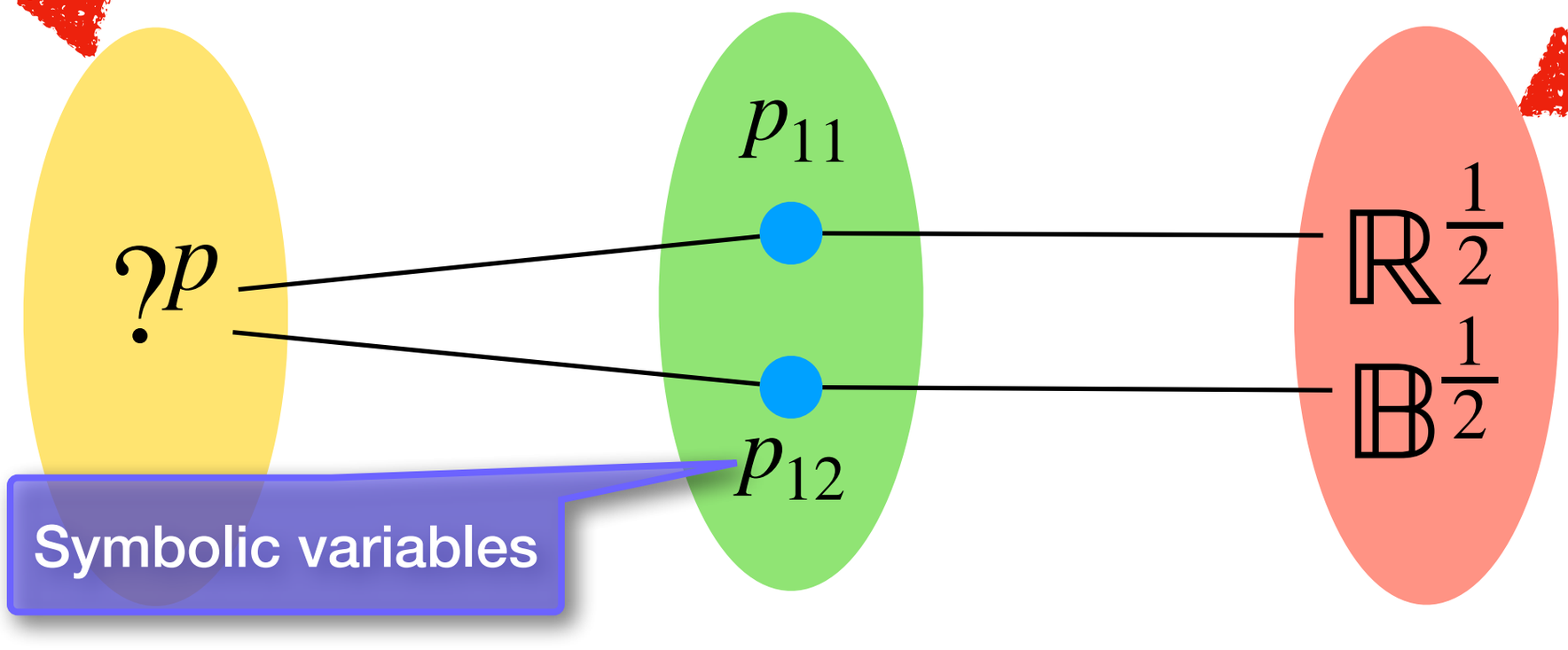
$$\{?^{?}\} \sim \{\mathbb{R}^{\frac{1}{2}}, \mathbb{B}^{\frac{1}{2}}\}$$



Couplings as a key center piece

Plausibility = couplings + constraints + satisfiability

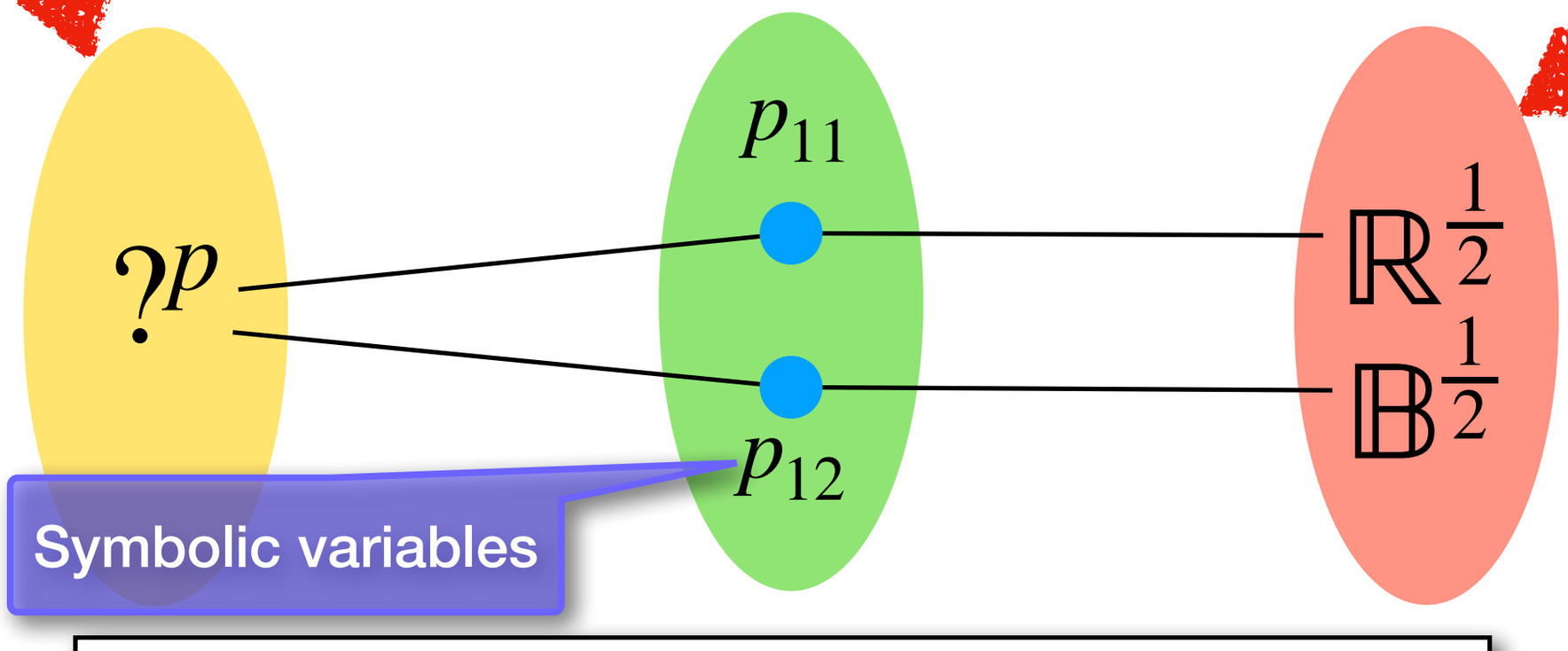
$$\{?^{?}\} \sim \{\mathbb{R}^{\frac{1}{2}}, \mathbb{B}^{\frac{1}{2}}\}$$



Couplings as a key center piece

Plausibility = couplings + constraints + satisfiability

$$\{?^{?}\} \sim \{\mathbb{R}^{\frac{1}{2}}, \mathbb{B}^{\frac{1}{2}}\}$$

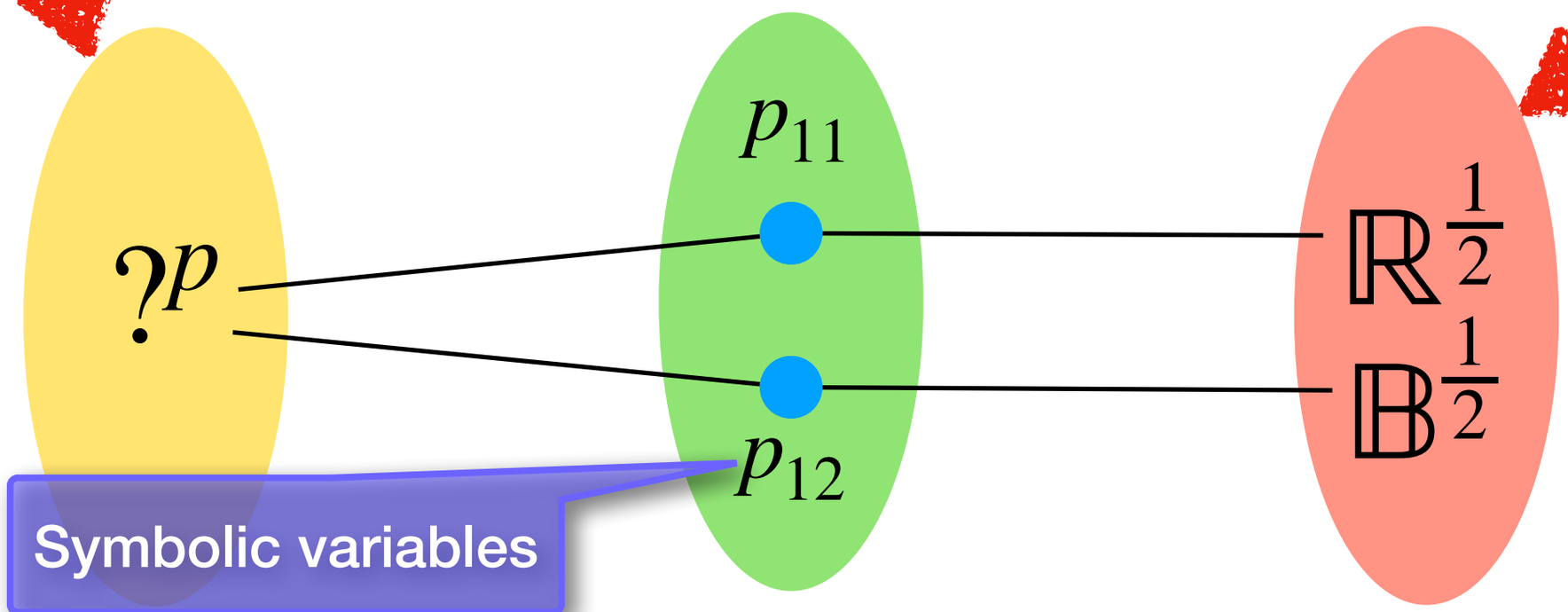


$$p_{11} + p_{12} = p \wedge p = 1$$
$$p_{11} = 1/2 \wedge p_{12} = 1/2$$
$$p_{11} + p_{12} = 1$$

Couplings as a key center piece

Plausibility = couplings + constraints + satisfiability

$$\{?^{?}\} \sim \{\mathbb{R}^{\frac{1}{2}}, \mathbb{B}^{\frac{1}{2}}\}$$



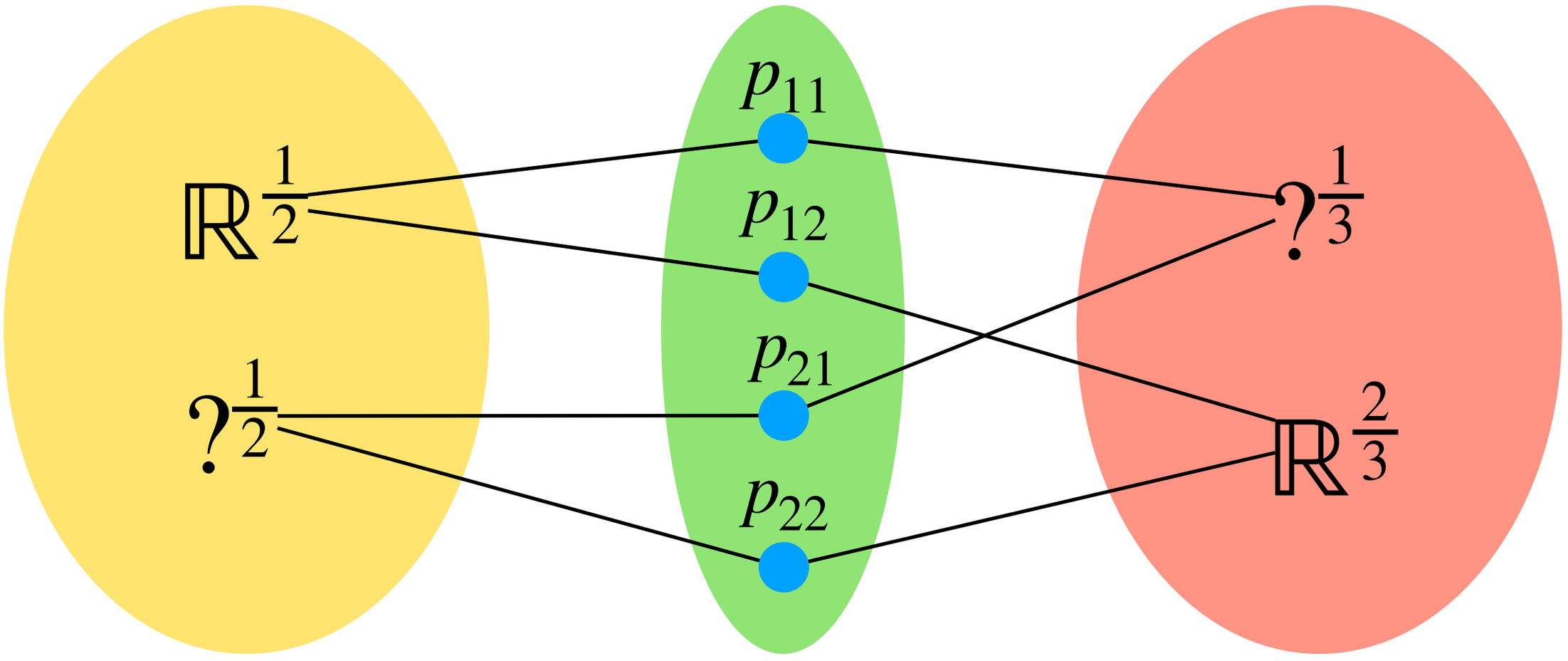
$$p_{11} + p_{12} = p \wedge p = 1$$
$$p_{11} = 1/2 \wedge p_{12} = 1/2$$
$$p_{11} + p_{12} = 1$$

Is it satisfiable?
Yes!

Couplings as a key center piece

We capture on runtime all possible solutions

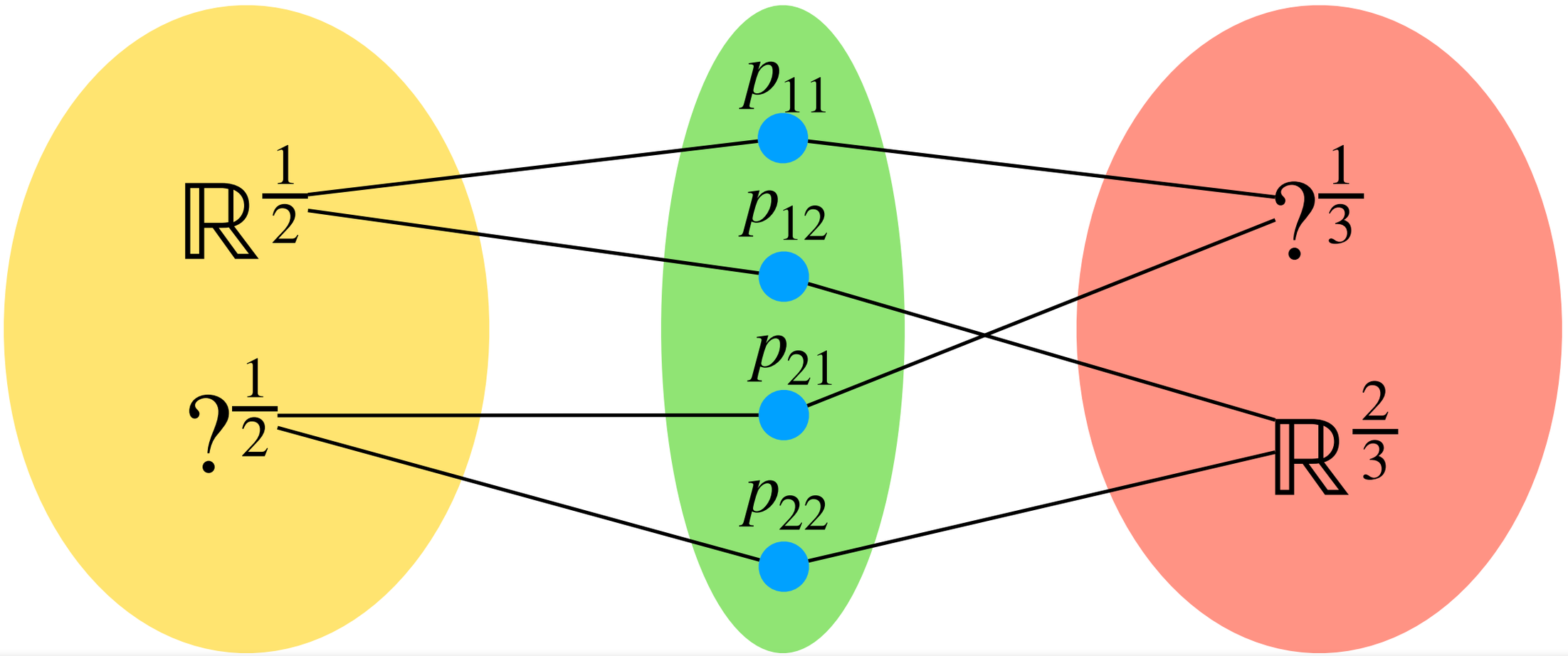
$$\{\mathbb{R}^{\frac{1}{2}}, ?^{\frac{1}{2}}\} \sim \{\mathbb{R}^{\frac{1}{3}}, ?^{\frac{2}{3}}\}$$



Couplings as a key center piece

We capture on runtime all possible solutions

$$\{\mathbb{R}^{\frac{1}{2}}, ?^{\frac{1}{2}}\} \sim \{\mathbb{R}^{\frac{1}{3}}, ?^{\frac{2}{3}}\}$$



Set of constraints satisfiable by

$$p_{11} \in [0,1] \wedge p_{12} = \frac{1}{2} - p_{11} \wedge p_{21} = \frac{1}{3} - p_{11} \wedge p_{22} = p_{11} + \frac{1}{6}$$

Inspecting couplings at runtime

```
val cheapServer: ? → {Bool^{90/100}, Error503^{10/100}} = ...
val expensiveServer: ? → {Bool^{98/100}, Error503^{2/100}} = ...

def externalCall(user: ?): {Bool^{95/100}, Bool^?, Error503^?} = {
  cheapServer(user) ⊕? expensiveServer(user)
}
```

Inspecting couplings at runtime

```
val cheapServer: ? → {Bool^{90/100}, Error503^{10/100}} = …
val expensiveServer: ? → {Bool^{98/100}, Error503^{2/100}} = …

def externalCall(user: ?): {Bool^{95/100}, Bool^{?}, Error503^{?}} = {
  cheapServer(user) ⊕? expensiveServer(user)
}
```

$$p_{\text{choice}} \leq \frac{37.5}{100}$$

What’s more in the paper

What’s more in the paper

- Formalization

What’s more in the paper

- Formalization
  - A source and a target language

What’s more in the paper

- Formalization
  - A source and a target language
  - How to *lift* relations on gradual types to relations on distribution types (couplings)

What’s more in the paper

- Formalization
  - A source and a target language
  - How to *lift* relations on gradual types to relations on distribution types (couplings)
  - Operational semantics defined over distributions of values

What's more in the paper

- Formalization
  - A source and a target language
  - How to *lift* relations on gradual types to relations on distribution types (couplings)
  - Operational semantics defined over distributions of values
- Properties

What’s more in the paper

- Formalization
  - A source and a target language
  - How to *lift* relations on gradual types to relations on distribution types (couplings)
  - Operational semantics defined over distributions of values
- Properties
  - Type Safety

What's more in the paper

- Formalization
  - A source and a target language
  - How to *lift* relations on gradual types to relations on distribution types (couplings)
  - Operational semantics defined over distributions of values
- Properties
  - Type Safety
  - **Gradual Guarantees**

The Dynamic Gradual Guarantee

Reduction is monotone with respect to imprecision

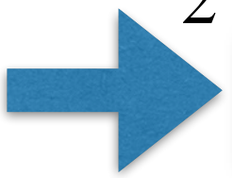
```
def foo(x: Bool): {Bool^1/2, Int^1/2} = {
    x ⊕_1/2 (x::?+1)
}
foo(true)
```

The Dynamic Gradual Guarantee

Reduction is monotone with respect to imprecision

```
def foo(x: Bool): {Bool^1/2, Int^1/2} = {
    x ⊕_1/2 (x::?+1)
}
foo(true)
```

$$p = \frac{1}{2}$$



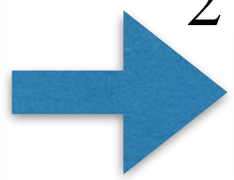
```
true
```

The Dynamic Gradual Guarantee

Reduction is monotone with respect to imprecision

```
def foo(x: Bool): {Bool^1/2, Int^1/2} = {
    x ⊕_1/2 (x::?+1)
}
foo(true)
```

$p = \frac{1}{2}$



`true`

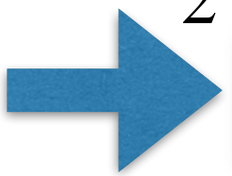
```
def foo(x: ?): {?^1/2, Int^1/2} = {
    x ⊕_1/2 (x::?+1)
}
foo(true)
```

The Dynamic Gradual Guarantee

Reduction is monotone with respect to imprecision

```
def foo(x: Bool): {Bool^1/2, Int^1/2} = {
    x ⊕_1/2 (x::?+1)
}
foo(true)
```

$p = \frac{1}{2}$



`true`

Also defined using couplings

$\sqsubseteq$

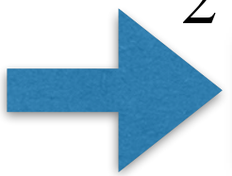
```
def foo(x: ?): {?^1/2, Int^1/2} = {
    x ⊕_1/2 (x::?+1)
}
foo(true)
```

The Dynamic Gradual Guarantee

Reduction is monotone with respect to imprecision

```
def foo(x: Bool): {Bool^1/2, Int^1/2} = {
    x ⊕_1/2 (x::?+1)
}
foo(true)
```

$p = \frac{1}{2}$



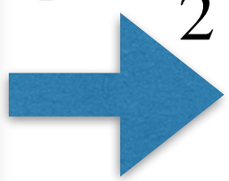
`true`

Also defined using couplings

$\sqsubset\!\sqsupset$

```
def foo(x: ?): {?^1/2, Int^1/2} = {
    x ⊕_1/2 (x::?+1)
}
foo(true)
```

$p = \frac{1}{2}$



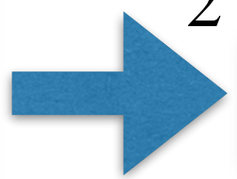
`error`

The Dynamic Gradual Guarantee

Reduction is monotone with respect to imprecision

```
def foo(x: Bool): {Bool^1/2, Int^1/2} = {
    x ⊕_1/2 (x::?+1)
}
foo(true)
```

$p = \frac{1}{2}$



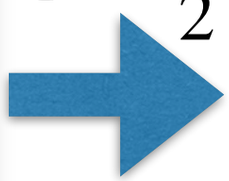
`true`

Also defined using couplings

$\sqsubseteq$

```
def foo(x: ?): {?^1/2, Int^1/2} = {
    x ⊕_1/2 (x::?+1)
}
foo(true)
```

$p = \frac{1}{2}$



`error`

It seems that the DGG doesn't hold

The Dynamic Gradual Guarantee

Reduction is monotone with respect to imprecision

```
def foo(x: Bool): {Bool^1/2, Int^1/2} = {
    x ⊕_1/2 (x::?+1)
}
foo(true)
```

$p = \frac{1}{2}$

`true`

$\{\texttt{true}^{\frac{1}{2}}, \texttt{error}^{\frac{1}{2}}\}$

Also defined using couplings

$\sqsubseteq$

```
def foo(x: ?): {?^1/2, Int^1/2} = {
    x ⊕_1/2 (x::?+1)
}
foo(true)
```

$p = \frac{1}{2}$

`error`

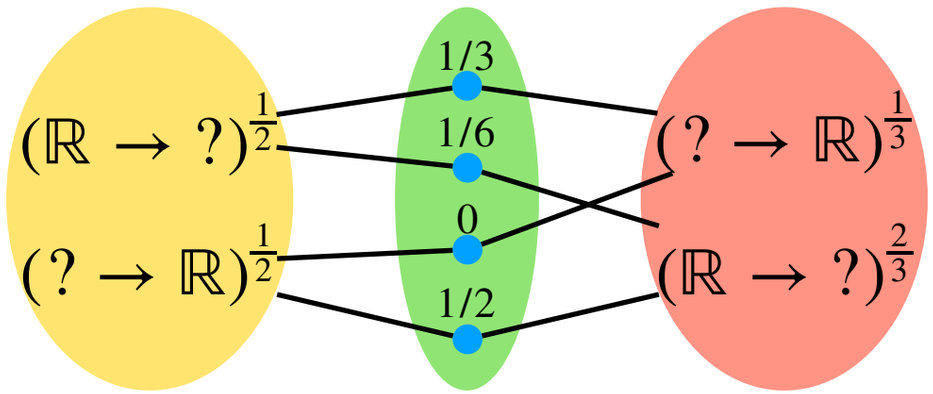
$\{(\texttt{true} :: ?)^{\frac{1}{2}}, \texttt{error}^{\frac{1}{2}}\}$

$\sqsubseteq$

It seems that the DGG doesn't hold

It holds if we analyze value distributions!

Thank you!


$(\mathbb{R} \to ?)^{\frac{1}{2}}$
$(? \to \mathbb{R})^{\frac{1}{2}}$
1/3
1/6
0
1/2
$(? \to \mathbb{R})^{\frac{1}{3}}$
$(\mathbb{R} \to ?)^{\frac{2}{3}}$

Thank you!

$$1 \oplus_{\frac{1}{3}} \texttt{true} : \{\texttt{Int}^{\overline{3}}, \texttt{Bool}^{\overline{3}}\}$$