Expectation Credits

Resourceful expected value reasoning for higher-order probabilistic programs

Alejandro Aguirre
Lars Birkedal
Kwing Hei Li
Philipp G. Haselwarter

Markus de Medeiros
Simon Oddershede Gregersen
Joseph Tassarotti

AARHUS UNIVERSITY

NYU

Probabilistic Programs

Rich functional language, including:

- Higher-order functions
- (Higher-order) state
- Generic recursion $(\mathsf{rec}\ f\ x\ =\ \ldots\ f\ \ldots)$
- Primitive random sampling $\mathsf{rand}(N)$

Probabilistic Programs

Rich functional language, including:

- Higher-order functions
- (Higher-order) state
- Generic recursion $(\text{rec } f\ x\ =\ \ldots\ f\ \ldots)$
- Primitive random sampling $\text{rand}(N)$

Cryptography,

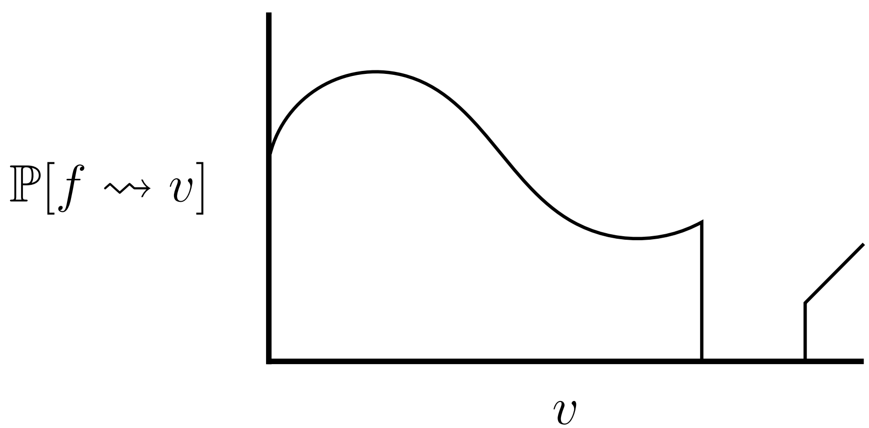
Differential privacy,

Random data structures

...

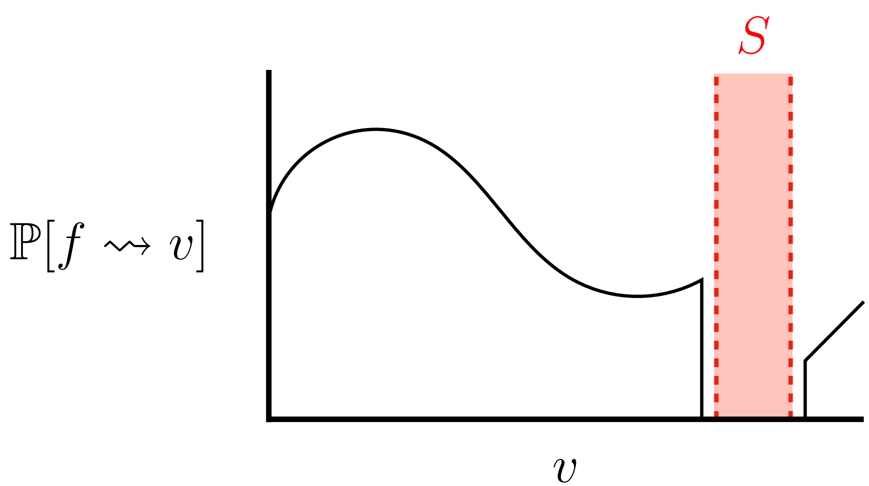
Probabilistic Programs

Executing a probabilistic program produces a subdistribution over states



Probabilistic Programs

Executing a probabilistic program produces a subdistribution over states



$$C(v) = \begin{cases} v \in S & 1 \\ v \notin S & 0 \end{cases}$$

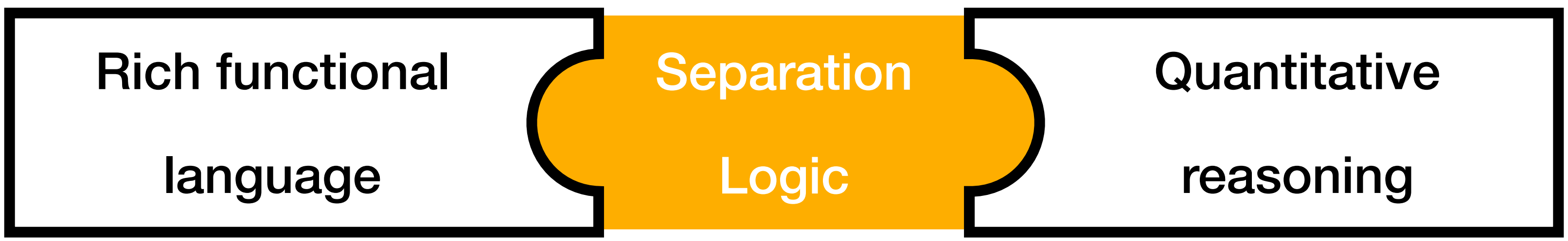
Safety: $\mathbb{E}[C] = 0$

Quantitative bounds ⇒ properties of the program

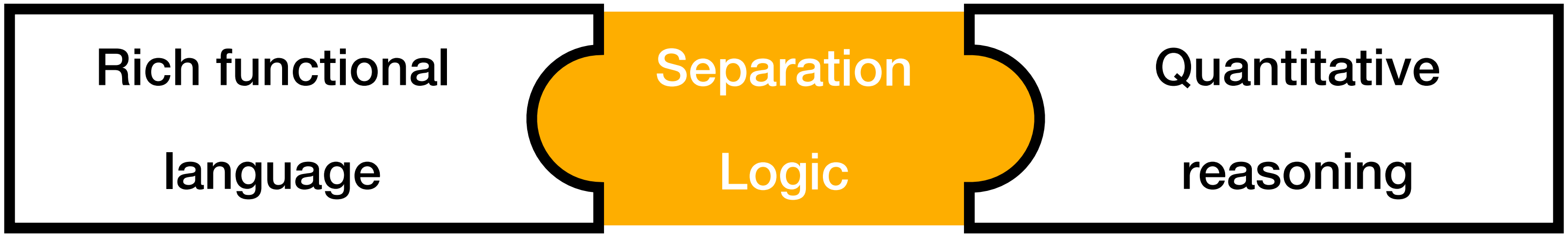

Rich functional language
Quantitative reasoning

- Compositionality issues
- Limited language features


Rich functional language
Quantitative reasoning



Expected values as state



Expected values as state

Challenge 1. **Approximate Correctness**

Challenge 2. **Almost-Sure Termination**

Challenge 3. **Expected Cost Bounds**

Challenge 1.

Approximate Correctness

Approximate Specifications

$$\text{hash} \ : A \to \text{int64}$$

$$\text{collide} \ : A \to A \to \text{bool}$$

$$\text{collide}\ x\ y = (\text{hash}\ x = \text{hash}\ y)$$

Approximate Specifications

$$\text{hash} \; : A \to \text{int64}$$

$$\text{collide} \; : A \to A \to \text{bool}$$

$$\text{collide}\ x\ y = (\text{hash}\ x = \text{hash}\ y)$$

$$\{x \neq y\}\ \text{collide}\ x\ y\ \{b.\, b = \text{false}\}_{\approx}$$

Approximate Specifications

aHL

$$\{x \neq y\}\ \mathsf{collide}\ x\ y\ \{b.\, b = \mathsf{false}\}_{2^{-64}}$$

Approximate Specifications

aHL

$$\{x \neq y\}\ \textsf{collide}\ x\ y\ \{b.\, b = \textsf{false}\}_{2^{-64}}$$

Useful reasoning principles,

Union Bound

$$\frac{\{P\}\, e_1\, \{Q\}_{\epsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\epsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\epsilon_1 + \epsilon_2}}$$

Sampling

$$\frac{\Pr_{x \sim D}[x \notin S] \leq \epsilon}{\{\textsf{True}\}\ \textsf{sample}(D)\ \{x.\, x \in S\}_{\epsilon}}$$

Approximate Specifications

aHL

$$\{x \neq y\}\ \mathsf{collide}\ x\ y\ \{b.\, b = \mathsf{false}\}_{2^{-64}}$$

Useful reasoning principles,

Union Bound

$$\frac{\{P\}\, e_1\, \{Q\}_{\epsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\epsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\epsilon_1 + \epsilon_2}}$$

Sampling

$$\frac{\Pr_{x \sim D}[x \notin S] \leq \epsilon}{\{\mathsf{True}\}\ \mathsf{sample}(D)\ \{x.\, x \in S\}_{\epsilon}}$$

Approximate Specifications

aHL

$$\{x \neq y\}\ \mathsf{collide}\ x\ y\ \{b.\, b = \mathsf{false}\}_{2^{-64}}$$

Useful reasoning principles,

Union Bound

$$\frac{\{P\}\, e_1\, \{Q\}_{\epsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\epsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\epsilon_1 + \epsilon_2}}$$

Sampling

$$\frac{\Pr_{x \sim D}[x \notin S] \leq \epsilon}{\{\mathsf{True}\}\ \mathsf{sample}(D)\ \{x.\, x \in S\}_{\epsilon}}$$

Approximate Specifications

aHL

$$\{x \neq y\}\ \mathsf{collide}\ x\ y\ \{b.\, b = \mathsf{false}\}_{2^{-64}}$$

Useful reasoning principles, **but limited compositionality.**

Approximate Specifications

aHL

$$\{x \neq y\}\ \mathsf{collide}\ x\ y\ \{b.\, b = \mathsf{false}\}_{2^{-64}}$$

Useful reasoning principles, **but limited compositionality.**

Limitation 1

$$\frac{\forall a.\, \{\ldots\}\ f\ a\ \{\ldots\}_{\epsilon(a)}}{\{\ldots\}\ \mathsf{map}\ f\ L\ \{\ldots\}_{?}}$$

Approximate Specifications

aHL

$$\{x \neq y\}\ \text{collide}\ x\ y\ \{b.\, b = \text{false}\}_{2^{-64}}$$

Useful reasoning principles, **but limited compositionality.**

Limitation 1

$$\frac{\forall a.\ \{\ldots\}\ f\ a\ \{\ldots\}_{\epsilon(a)}}{\{\ldots\}\ \text{map}\ f\ L\ \{\ldots\}_{\sum_{a \in L} \epsilon(a)}}$$

Approximate Specifications

aHL

$$\{x \neq y\}\ \mathsf{collide}\ x\ y\ \{b.\, b = \mathsf{false}\}_{2^{-64}}$$

Useful reasoning principles, **but limited compositionality.**

Limitation 1

$$\frac{\forall a.\, \{\ldots\}\ f\ a\ \{\ldots\}_{\epsilon(a)}}{\{\ldots\}\ \mathsf{map}\ f\ L\ \{\ldots\}_{\sum_{a \in L} \epsilon(a)}}$$

error specifications propagate

Approximate Specifications

aHL

$$\{x \neq y\}\ \mathsf{collide}\ x\ y\ \{b.\, b = \mathsf{false}\}_{2^{-64}}$$

Useful reasoning principles, **but limited compositionality.**

Limitation 2

$$\{\top\}\ G\ d\ \{d.\, P\}_0$$
$$\{\top\}\ F\ d\ \{d.\, P\}_{1/100}$$

$$\begin{aligned}\mathsf{test}\ d = \ & \mathsf{if}\ \mathsf{decide}\ d \\ & \ \mathsf{then}\ (\mathsf{true}, G\ d) \\ & \ \mathsf{else}\ (\mathsf{false}, F\ d)\end{aligned}$$

Approximate Specifications

aHL

$$\{x \neq y\}\ \textsf{collide}\ x\ y\ \{b.\, b = \textsf{false}\}_{2^{-64}}$$

Useful reasoning principles, **but limited compositionality.**

Limitation 2

$$\{\top\}\ G\ d\ \{d.\, P\}_{0}$$
$$\{\top\}\ F\ d\ \{d.\, P\}_{1/100}$$

$$\textsf{test}\ d = \ \textsf{if decide}\ d\ \textsf{then}\ (\textsf{true}, G\ d)\ \textsf{else}\ (\textsf{false}, F\ d)$$

Approximate Specifications

aHL

$$\{x \neq y\}\ \mathsf{collide}\ x\ y\ \{b.\, b = \mathsf{false}\}_{2^{-64}}$$

Useful reasoning principles, **but limited compositionality.**

Limitation 2

$$\{\top\}\ G\ d\ \{d.\, P\}_0$$
$$\{\top\}\ F\ d\ \{d.\, P\}_{1/100}$$

$$\begin{aligned} \mathsf{test}\ d = \ & \mathsf{if}\ \mathsf{decide}\ d \\ & \mathsf{then}\ (\mathsf{true}, G\ d) \\ & \mathsf{else}\ (\mathsf{false}, F\ d) \end{aligned}$$

Approximate Specifications

aHL

$$\{x \neq y\}\ \text{collide}\ x\ y\ \{b.\, b = \text{false}\}_{2^{-64}}$$

Useful reasoning principles, **but limited compositionality.**

Limitation 2

$$\{\top\}\ G\ d\ \{d.\, P\}_0$$
$$\{\top\}\ F\ d\ \{d.\, P\}_{1/100}$$

$$\begin{aligned} \text{test}\ d = \ & \text{if decide}\ d \\ & \text{then}\ (\text{true}, G\ d) \\ & \text{else}\ (\text{false}, F\ d) \end{aligned}$$

$$\{\top\}\ \text{test}\ d\ \{(v, d).\, P\}_?$$

Approximate Specifications

aHL

$$\{x \neq y\}\ \text{collide}\ x\ y\ \{b.\, b = \text{false}\}_{2^{-64}}$$

Useful reasoning principles, **but limited compositionality.**

Limitation 2

$$\{\top\}\ G\ d\ \{d.\, P\}_0$$

$$\{\top\}\ F\ d\ \{d.\, P\}_{1/100}$$

$$\begin{aligned}\text{test}\ d = \ & \text{if decide}\ d \\ & \text{then}\ (\text{true}, G\ d) \\ & \text{else}\ (\text{false}, F\ d)\end{aligned}$$

$$\{\top\}\ \text{test}\ d\ \{(v, d).\, P\}_?$$

error depends on return value

Approximate Specifications

aHL

$$\{x \neq y\}\ \mathsf{collide}\ x\ y\ \{b.\, b = \mathsf{false}\}_{2^{-64}}$$

Error Credits

Eris

$$\{x \neq y\}\ \textsf{collide}\ x\ y\ \{b.\, b = \textsf{false}\}_{2^{-64}}$$

$$\{↯(2^{-64}) * x \neq y\}\ \textsf{collide}\ x\ y\ \{b.\, b = \textsf{false}\}$$

Error Credits

Eris

$$\{x \neq y\}\ \text{collide}\ x\ y\ \{b.\, b = \text{false}\}_{2^{-64}}$$

$$\{\text{↯}(2^{-64}) * x \neq y\}\ \text{collide}\ x\ y\ \{b.\, b = \text{false}\}$$

Expected Error Bounds as a Resource

Error Credits

Eris

Expected Error Bounds as a Resource

$$\vdash \{\text{↯}(\epsilon)\}\ f\ \{v.\, P\}$$

If f terminates with value v,
$P\ v$ holds with probability $1 - \epsilon$.

Iris

Step-indexed & higher-order
Mechanized in Rocq

Error Credits

Eris

Expected Error Bounds as a Resource

$$\vdash \{ \lightning(\epsilon) \}\ f\ \{v.\, P\}$$

$$\frac{\{P\}\ f\ \{Q\}}{\{P * \lightning(\epsilon)\}\ f\ \{Q * \lightning(\epsilon)\}}$$

$$(\triangleright P \Rightarrow P) \vdash P$$

$$\Big\{\ \{P * \lightning(\epsilon)\}\ f\ \{Q\}\ \Big\}\ g\ \{R\}$$

Iris

Step-indexed & higher-order
Mechanized in Rocq

The Eris Logic

Limitation 1

The Eris Logic

Limitation 1

Standard higher-order specification:

$$\frac{\forall a,\ \{P\ a\}\ f\ a\ \{Q\ a\}}{\left\{\mathop{\Large\ast}_{a \in L} (P\ a)\right\}\ \mathsf{map}\ f\ L\ \left\{L'.\ \mathop{\Large\ast}_{a \in L'} (Q\ a)\right\}}$$

The Eris Logic

Limitation 1

Standard higher-order specification:

$$\frac{\forall a,\ \{P\ a\}\ f\ a\ \{Q\ a\}}{\left\{\mathop{\ast}_{a \in L} (P\ a)\right\}\ \mathsf{map}\ f\ L\ \left\{L'.\ \mathop{\ast}_{a \in L'} (Q\ a)\right\}}$$

Derived error-aware specification:

$$\frac{\forall y,\ \{↯(2^{-64})\}\ \mathsf{hash}\ y\ \{v.\, v \neq v'\}}{\left\{\mathop{\ast}_{a \in L} ↯(2^{-64})\right\}\ \mathsf{map\ hash}\ L\ \left\{L'.\ \mathop{\ast}_{a \in L'} a \neq v'\right\}}$$

The Eris Logic

Limitation 2

$$\{\top\}\, G\ d\, \{d.\, P\}_{0}$$

$$\{\top\}\, F\ d\, \{d.\, P\}_{1/100}$$

$$\textsf{test}\ d = \ \textsf{if decide}\ d\ \textsf{then}\ (\textsf{true}, G\ d)\ \textsf{else}\ (\textsf{false}, F\ d)$$

$$\{\top\}\, \textsf{test}\ d\, \{(v, d).\, P\}_{?}$$

The Eris Logic

Limitation 2

$$\{\top\}\, G\ d\, \{d.\, P\}$$
$$\{↯(1/100)\}\, F\ d\, \{d.\, P\}$$

$$\mathsf{test}\ d = \mathsf{if\ decide}\ d\ \mathsf{then}\ (\mathsf{true}, G\ d)\ \mathsf{else}\ (\mathsf{false}, F\ d)$$

State-dependent specification:

$$\left\{\ ↯(1/100)\ \right\}\mathsf{test}\ d\left\{(v, d).\, P * \begin{pmatrix} \mathsf{if}\ v \\ \mathsf{then}\ ↯(1/100) \\ \mathsf{else}\ \top \end{pmatrix}\right\}$$

Error Credits

Core Rules

Error Credits

Core Rules

Spending $\lightning(1) \vdash \bot$

Error Credits

Core Rules

Spending $\lightning(1) \vdash \bot$

Splitting $\lightning(\epsilon_1 + \epsilon_2) \dashv\vdash \lightning(\epsilon_1) * \lightning(\epsilon_2)$

Error Credits

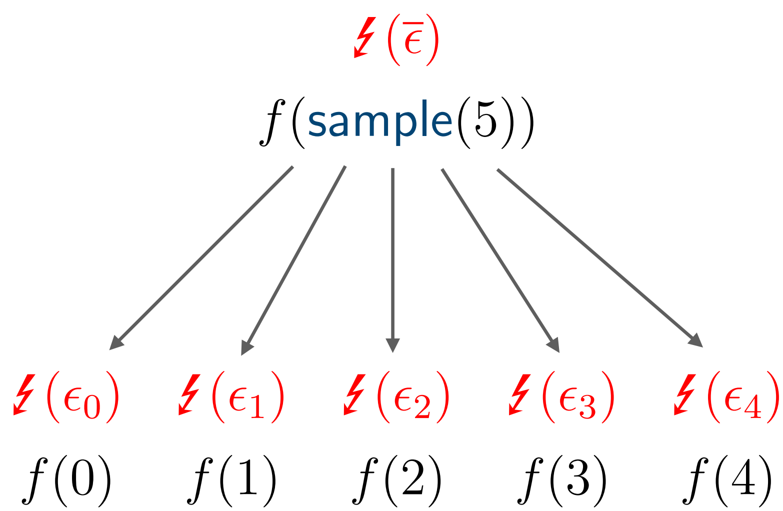
Core Rules

Spending $\lightning(1) \vdash \bot$

Splitting $\lightning(\epsilon_1 + \epsilon_2) \dashv\vdash \lightning(\epsilon_1) * \lightning(\epsilon_2)$

Averaging

$$\frac{\mathbb{E}_{x \sim D}[\epsilon_x] = \overline{\epsilon}}{\{\lightning(\overline{\epsilon})\}\ \mathsf{sample}(D)\ \{x.\, \lightning(\epsilon_x)\}}$$



Error Credits

Derived Rules

aHL Union Bound

$$\frac{\{P\}\, e_1\, \{Q\}_{\epsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\epsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\epsilon_1 + \epsilon_2}}$$

Error Credits

Derived Rules

aHL Union Bound

$$\frac{\{P\}\, e_1\, \{Q\}_{\epsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\epsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\epsilon_1 + \epsilon_2}}$$

$$\{ \lightning(\epsilon_1) * P \}\, e_1\, \{Q\}$$

$$\{ \lightning(\epsilon_2) * Q \}\, e_2\, \{R\}$$

Error Credits

Derived Rules

aHL Union Bound

$$\frac{\{P\}\, e_1\, \{Q\}_{\epsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\epsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\epsilon_1 + \epsilon_2}}$$

$$\{\lightning(\epsilon_1) * P\}\, e_1\, \{Q\}$$

$$\{\lightning(\epsilon_2) * Q\}\, e_2\, \{R\}$$

$$\lightning(\epsilon_1 + \epsilon_2) * P$$

$$e_1;\ e_2$$

Error Credits

Derived Rules

aHL Union Bound

$$\frac{\{P\}\, e_1\, \{Q\}_{\epsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\epsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\epsilon_1 + \epsilon_2}}$$

$$\{↯(\epsilon_1) * P\}\, e_1\, \{Q\}$$

$$\{↯(\epsilon_2) * Q\}\, e_2\, \{R\}$$

$$↯(\epsilon_1) * ↯(\epsilon_2) * P$$

$$e_1;\ e_2$$

Splitting

Error Credits

Derived Rules

aHL Union Bound

$$\frac{\{P\}\, e_1\, \{Q\}_{\epsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\epsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\epsilon_1 + \epsilon_2}}$$

$$\lightning(\epsilon_1) * \lightning(\epsilon_2) * P$$

Splitting

$$\{\lightning(\epsilon_2) * Q\}\, e_2\, \{R\}$$

$$e_1;\; e_2$$

$$\{\lightning(\epsilon_1) * P\}\, e_1\, \{Q\}$$

Frame Rule

$$\lightning(\epsilon_2) * Q$$

$$e_2$$

Error Credits

Derived Rules

aHL Union Bound

$$\frac{\{P\}\, e_1\, \{Q\}_{\epsilon_1} \qquad \{Q\}\, e_2\, \{R\}_{\epsilon_2}}{\{P\}\, e_1; e_2\, \{R\}_{\epsilon_1 + \epsilon_2}}$$

$$\lightning(\epsilon_1) * \lightning(\epsilon_2) * P$$

Splitting

$$e_1;\ e_2$$

Frame Rule

$$\{\lightning(\epsilon_1) * P\}\, e_1\, \{Q\}$$

$$\lightning(\epsilon_2) * Q$$

$$e_2$$

$$\{\lightning(\epsilon_2) * Q\}\, e_2\, \{R\}$$

$$R$$

Error Credits

Derived Rules

aHL Sampling

$$\frac{\Pr_{x \sim D}[x \notin S] < \epsilon}{\{\mathsf{True}\}\ \mathsf{sample}(D)\ \{x.\, x \in S\}_{\epsilon}}$$

Error Credits

Derived Rules

aHL Sampling

$$\frac{\Pr_{x \sim D}[x \notin S] < \epsilon}{\{\mathsf{True}\}\ \mathsf{sample}(D)\ \{x.\, x \in S\}_{\epsilon}}$$

$$↯(1/5)$$

$$f(\mathsf{sample}(5))$$

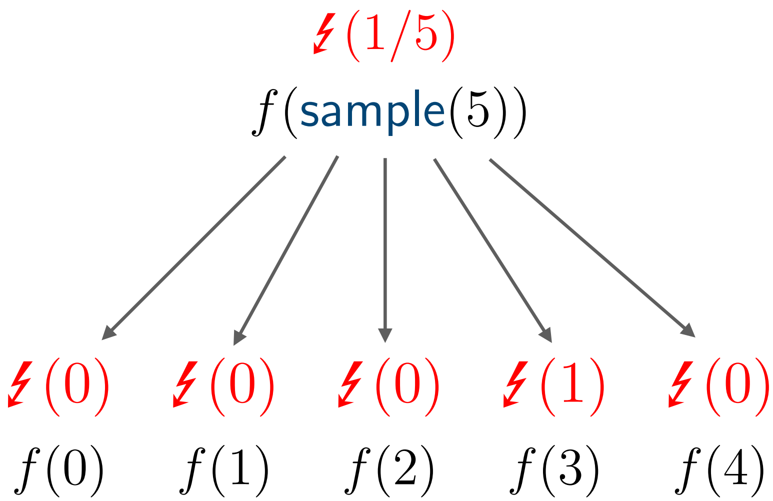
Error Credits

Derived Rules

aHL Sampling

$$\frac{\Pr_{x\sim D}[x \notin S] < \epsilon}{\{\mathsf{True}\}\ \mathsf{sample}(D)\ \{x.\, x \in S\}_{\epsilon}}$$

Averaging



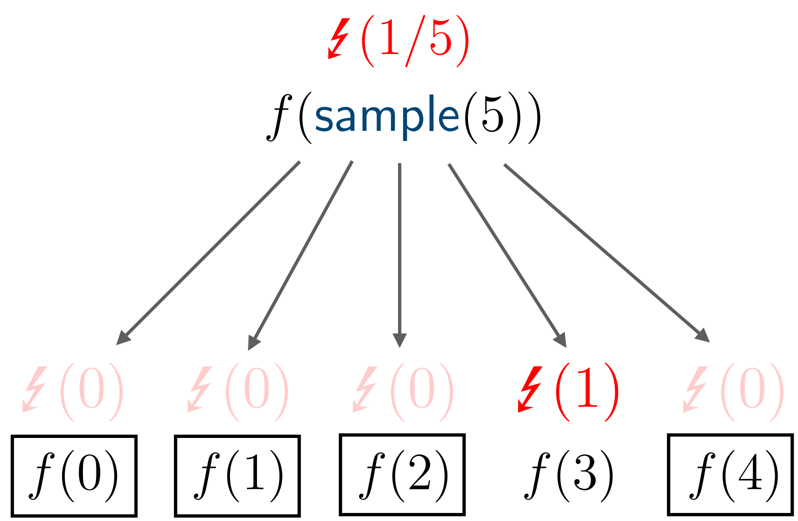
Error Credits

Derived Rules

aHL Sampling

$$\frac{\Pr_{x \sim D}[x \notin S] < \epsilon}{\{\mathsf{True}\}\ \mathsf{sample}(D)\ \{x.\, x \in S\}_{\epsilon}}$$

Averaging

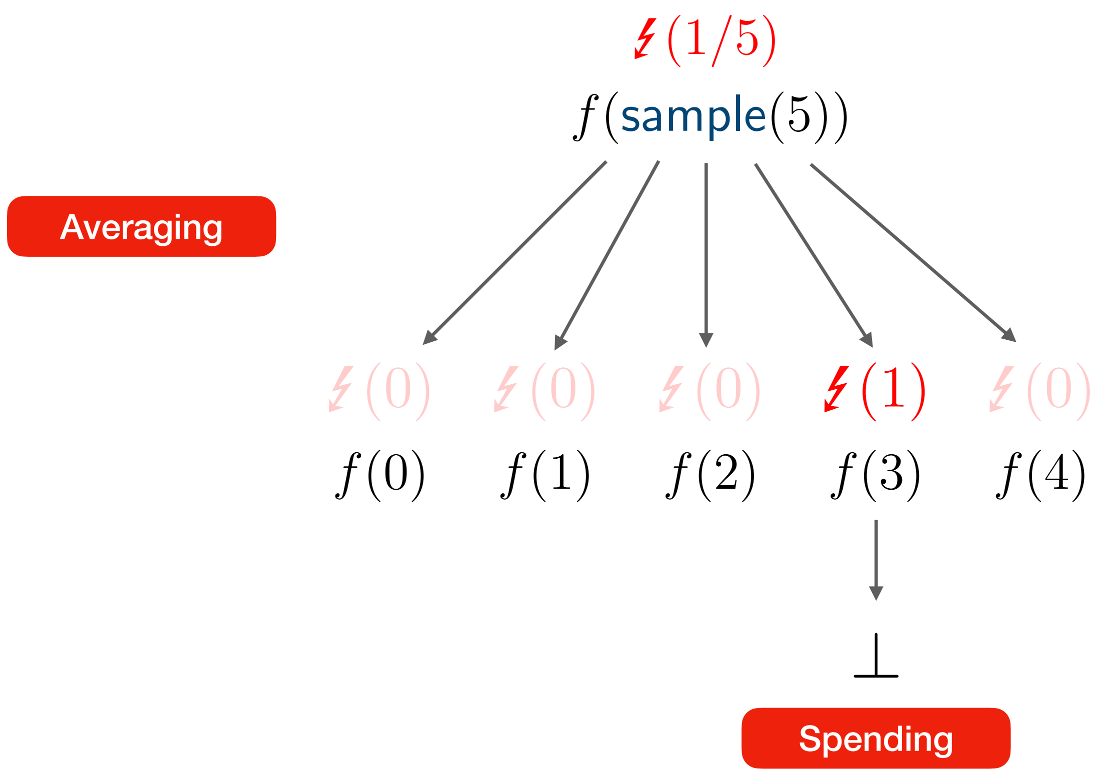


Error Credits

Derived Rules

aHL Sampling

$$\frac{\Pr_{x \sim D}[x \notin S] < \epsilon}{\{\mathsf{True}\}\ \mathsf{sample}(D)\ \{x.\, x \in S\}_{\epsilon}}$$



Hash-based authentication in Eris

Hash Collisions

$$\text{hash} : A \rightarrow \text{int64}$$

$$\begin{aligned}\text{hash } x = \; & \text{match get } x \text{ with} \\ & \quad \text{Some } (v) \Rightarrow v \\ & \mid \text{None} \Rightarrow \; \text{let } v = \text{sample}(2^{64}) \text{ in} \\ & \qquad\qquad\quad \text{set } x \; v; \\ & \qquad\qquad\quad v \\ & \text{end}\end{aligned}$$

Hash Collisions

$$
\begin{aligned}
&\text{hash} : A \to \text{int64} \\
&\text{hash } x = \ \text{match get } x \text{ with} \\
&\qquad\qquad \text{Some } (v) \Rightarrow v \\
&\qquad\quad\ \mid \text{None} \Rightarrow \ \text{let } v = \text{sample}(2^{64}) \text{ in} \\
&\qquad\qquad\qquad\qquad\quad \text{set } x\ v; \\
&\qquad\qquad\qquad\qquad\quad v \\
&\qquad\quad\ \text{end}
\end{aligned}
$$

Property: collisionFree N

- **Map is collision-free**
- **At most N hashes**

Hash Collisions

hash : $A \rightarrow$ int64

$$
\begin{aligned}
\text{hash } x = \ & \text{match get } x \text{ with} \\
& \quad \text{Some } (v) \Rightarrow v \\
& \ | \ \text{None} \Rightarrow \ \text{let } v = \text{sample}(2^{64}) \text{ in} \\
& \qquad\qquad\quad \text{set } x \ v; \\
& \qquad\qquad\quad v \\
& \text{end}
\end{aligned}
$$

Property: collisionFree N

Hash Collisions

$$\left\{ \begin{array}{c} \text{collisionFree } N * \\ \lightning\,(?) \end{array} \right\} \text{hash } x \left\{ v.\ \begin{array}{c} \text{collisionFree } (N+1) * \\ \text{get } x = v \end{array} \right\}$$

hash : $A \rightarrow$ int64

```
hash x =  match get x with
            Some (v) ⇒ v
          | None ⇒  let v = sample(2^64) in
                    set x v;
                    v
          end
```

Property: collisionFree N

Hash Collisions

hash : $A \to$ int64

```
hash x =  match get x with
            | Some (v) => v
            | None =>  let v = sample(2^64) in
                       set x v;
                       v
          end
```

$$\left\{ \begin{array}{c} \text{collisionFree } N \,* \\ \text{⚡}(?) \end{array} \right\} \text{hash } x \left\{ v.\ \begin{array}{c} \text{collisionFree } (N+1) \,* \\ \text{get } x = v \end{array} \right\}$$

- **Already Hashed**

Property: collisionFree N

Hash Collisions

$\text{hash} : A \to \text{int64}$

```
hash x =  match get x with
            Some (v) ⇒ v
          | None ⇒  let v = sample(2^64) in
                    set x v;
                    v
          end
```

$$\left\{ \begin{array}{c} \text{collisionFree } N \ * \\ ⚡(?) \end{array} \right\} \text{hash } x \left\{ v. \begin{array}{c} \text{collisionFree } (N+1) \ * \\ \text{get } x = v \end{array} \right\}$$

- **Already Hashed**

⚡(0)

Property: collisionFree N

Hash Collisions

$\text{hash} : A \to \text{int64}$

$$
\begin{aligned}
\text{hash } x = \; & \text{match get } x \text{ with} \\
& \;\; \text{Some } (v) \Rightarrow v \\
& | \; \text{None} \Rightarrow \; \text{let } v = \text{sample}(2^{64}) \text{ in} \\
& \qquad\qquad\;\; \text{set } x \; v; \\
& \qquad\qquad\;\; v \\
& \text{end}
\end{aligned}
$$

$$
\left\{ \begin{array}{c} \text{collisionFree } N * \\ ⚡(?) \end{array} \right\} \text{hash } x \left\{ v. \begin{array}{c} \text{collisionFree } (N+1) * \\ \text{get } x = v \end{array} \right\}
$$

- **Already Hashed**

  $⚡(0)$

- **New Hash**

Property: collisionFree N

Hash Collisions

$$\left\{ \begin{array}{c} \text{collisionFree } N \ * \\ ↯(?) \end{array} \right\} \text{hash } x \left\{ v. \begin{array}{c} \text{collisionFree } (N+1) \ * \\ \text{get } x = v \end{array} \right\}$$

hash : $A \rightarrow$ int64

```
hash x =  match get x with
            Some (v) => v
          | None => let v = sample(2^64) in
                    set x v;
                    v
          end
```

- **Already Hashed**

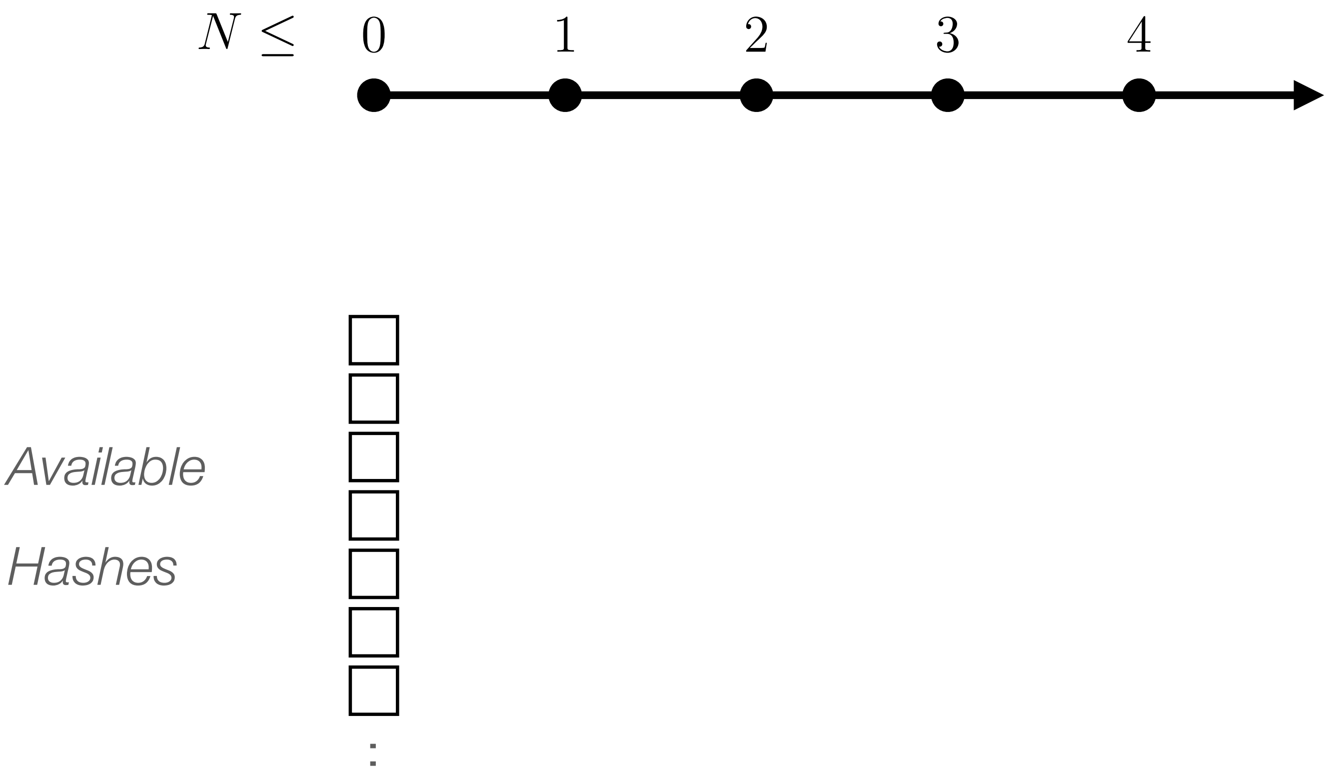
  $↯(0)$

- **New Hash**

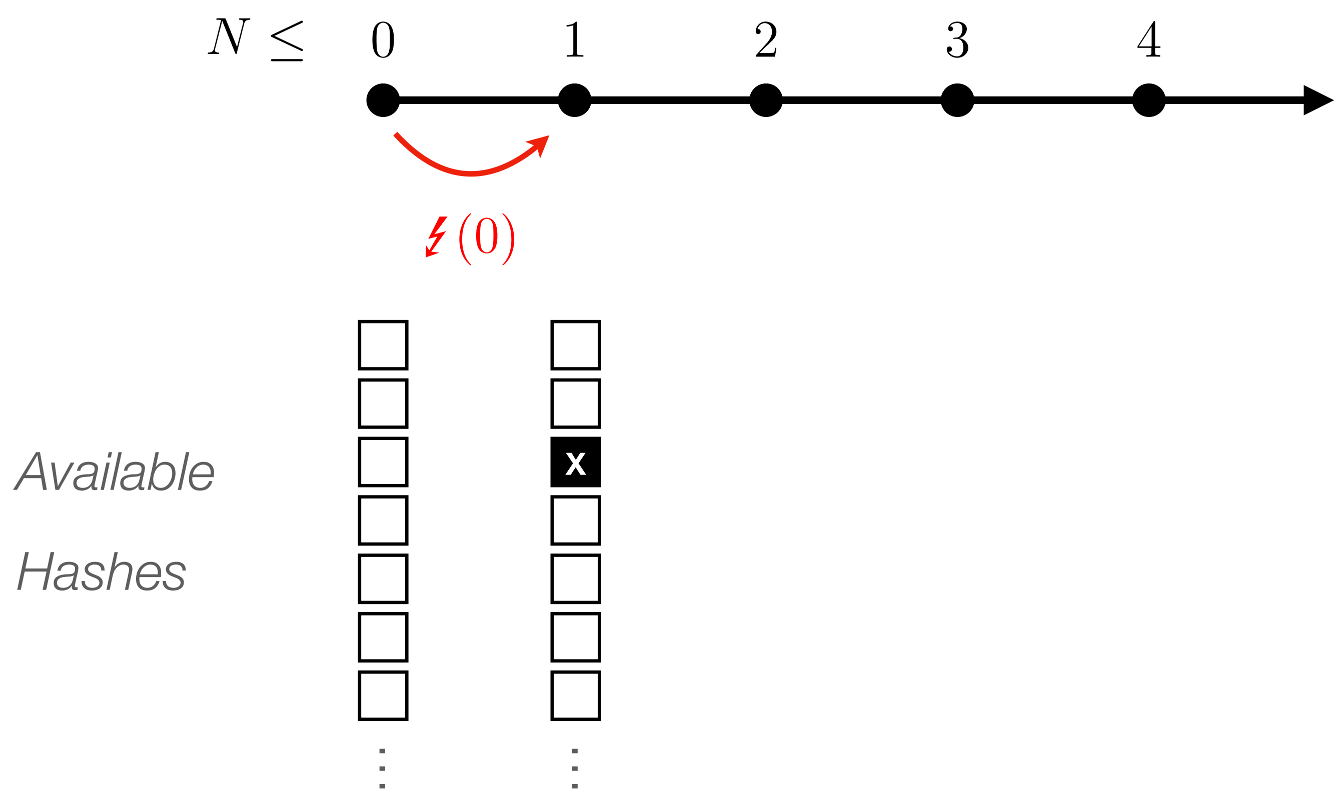
  $↯(?)$

Property: collisionFree N

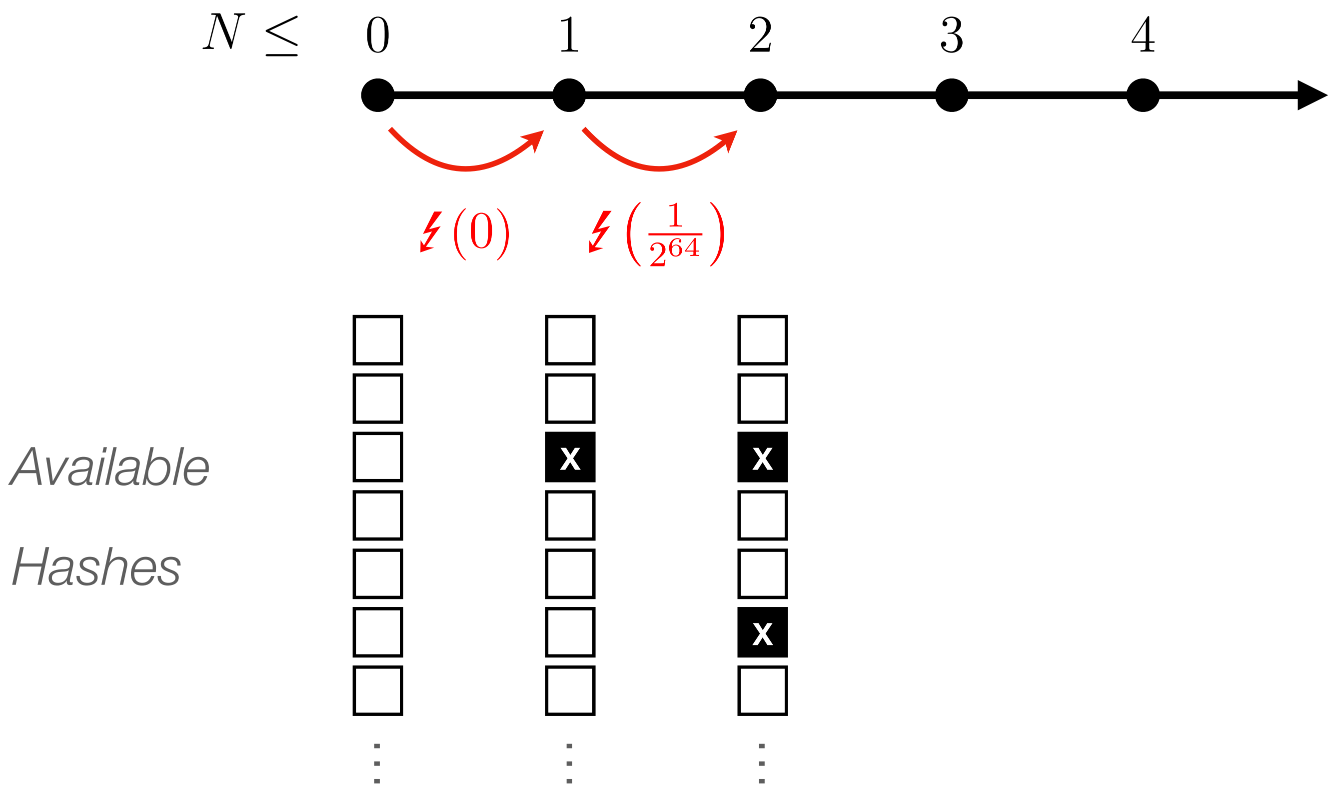
Preserving Collision Freedom



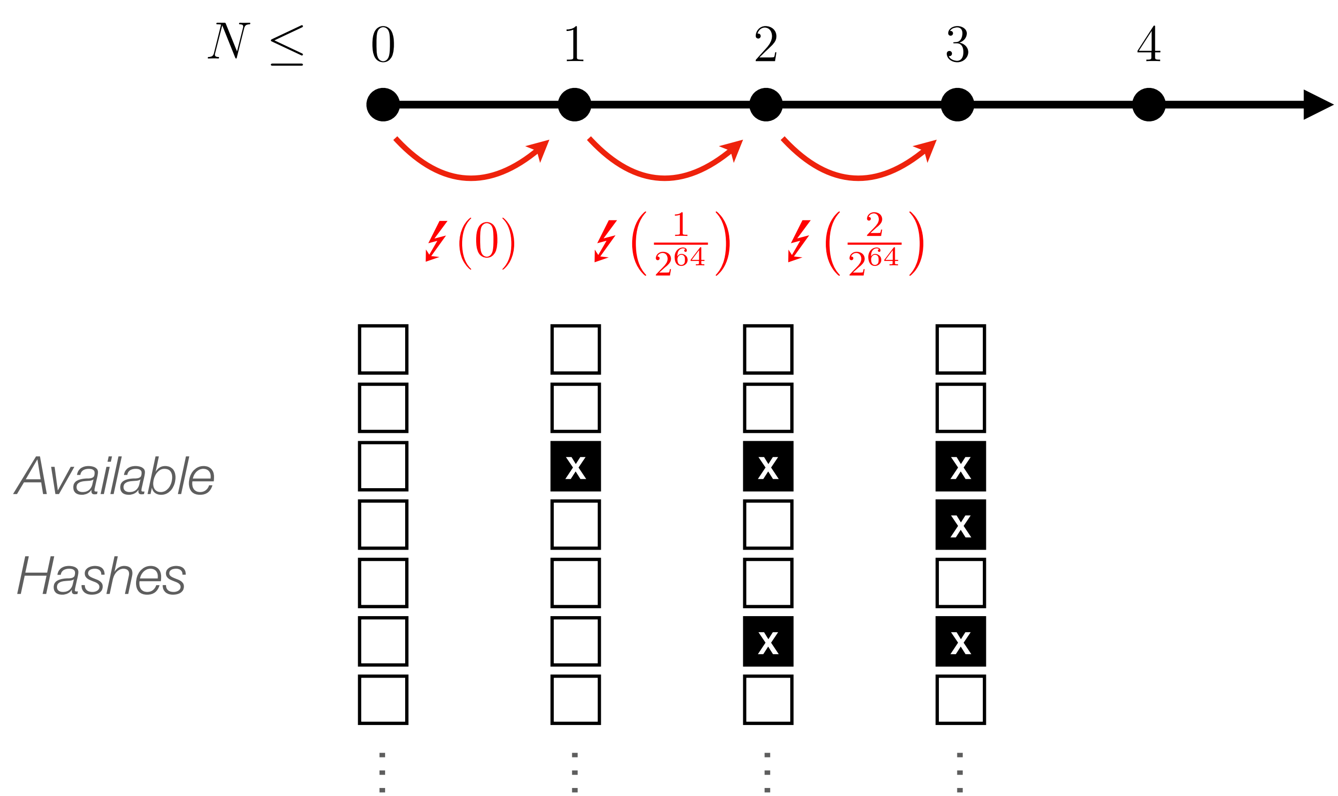
Preserving Collision Freedom



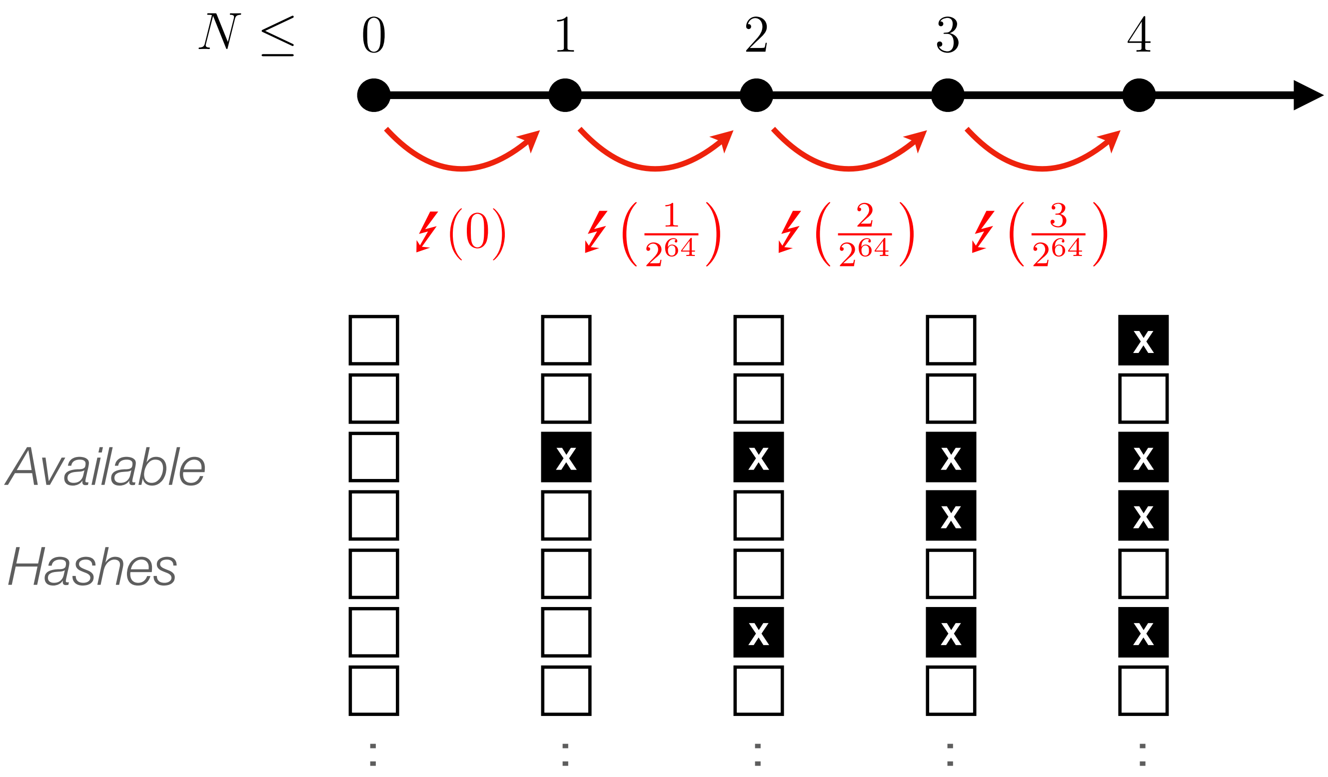
Preserving Collision Freedom



Preserving Collision Freedom



Preserving Collision Freedom



Hash Collisions

$$\left\{ \begin{array}{c} \text{collisionFree } N \ast \\ ⚡(?) \end{array} \right\} \text{hash } x \left\{ v. \begin{array}{c} \text{collisionFree } (N+1) \ast \\ \text{get } x = v \end{array} \right\}$$

$\text{hash} : A \to \text{int64}$

$\text{hash } x =$ match get x with
  Some $(v) \Rightarrow v$
  | None $\Rightarrow$ let $v = \text{sample}(2^{64})$ in
    set x v;
    v
  end

- **Already Hashed**

  ⚡(0)

- **New Hash**

  ⚡$\left(\frac{N}{2^{64}}\right)$

Property: collisionFree N

Hash Collisions

hash : $A \to$ int64

hash $x =$ match get x with
  Some $(v) \Rightarrow v$
  | None $\Rightarrow$ let $v =$ sample(2^{64}) in
    set x v;
    v
  end

$$\left\{ \begin{array}{c} \text{collisionFree } N \ * \\ \text{↯}(N \cdot 2^{-64}) \end{array} \right\} \text{hash } x \left\{ v.\ \begin{array}{c} \text{collisionFree } (N+1) \ * \\ \text{get } x = v \end{array} \right\}$$

Property: collisionFree N

Hash Collisions

hash : $A \to$ int64

hash $x =$ match get x with
  Some $(v) \Rightarrow v$
  | None $\Rightarrow$ let $v =$ sample(2^{64}) in
    set x v;
    v
  end

Amortize over M hashes

$$\left\{ \begin{array}{c} \text{collisionFree } N \ * \\ \text{↯}(N \cdot 2^{-64}) \end{array} \right\} \text{ hash } x \left\{ v. \begin{array}{c} \text{collisionFree } (N+1) \ * \\ \text{get } x = v \end{array} \right\}$$

Simplify client dependency on N?

Property: collisionFree N

Hash Collisions

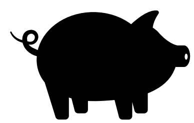
hash : $A \to$ int64

```
hash x =  match get x with
            Some (v) ⇒ v
          | None ⇒  let v = sample(2^64) in
                    set x v;
                    v
          end
```

Amortize over M hashes

$$\left\{ \begin{array}{c} \text{collisionFree } N \ * \\ I(N) * N < M * ↯(k) \end{array} \right\} \text{hash } x \left\{ \begin{array}{c} \text{collisionFree } (N+1) \ * \\ I(N+1) \end{array} \right\}$$

Derived!

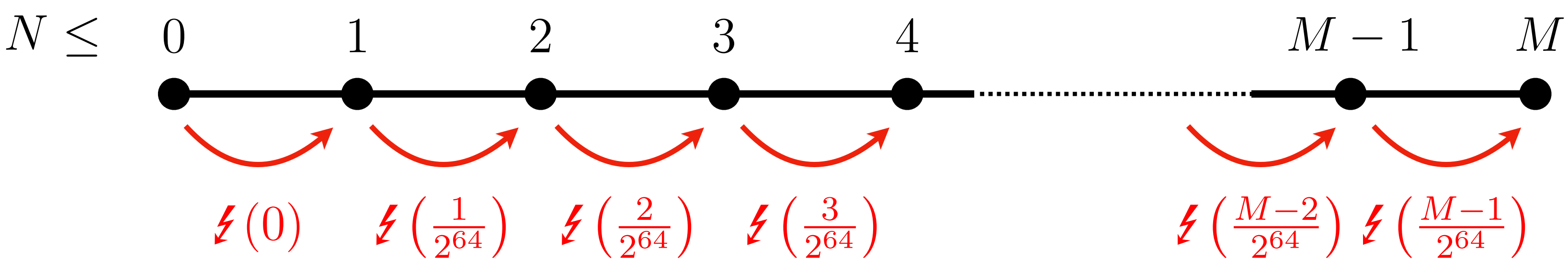


$$I(N) \triangleq (N \leq M) * ↯(\Delta_N)$$

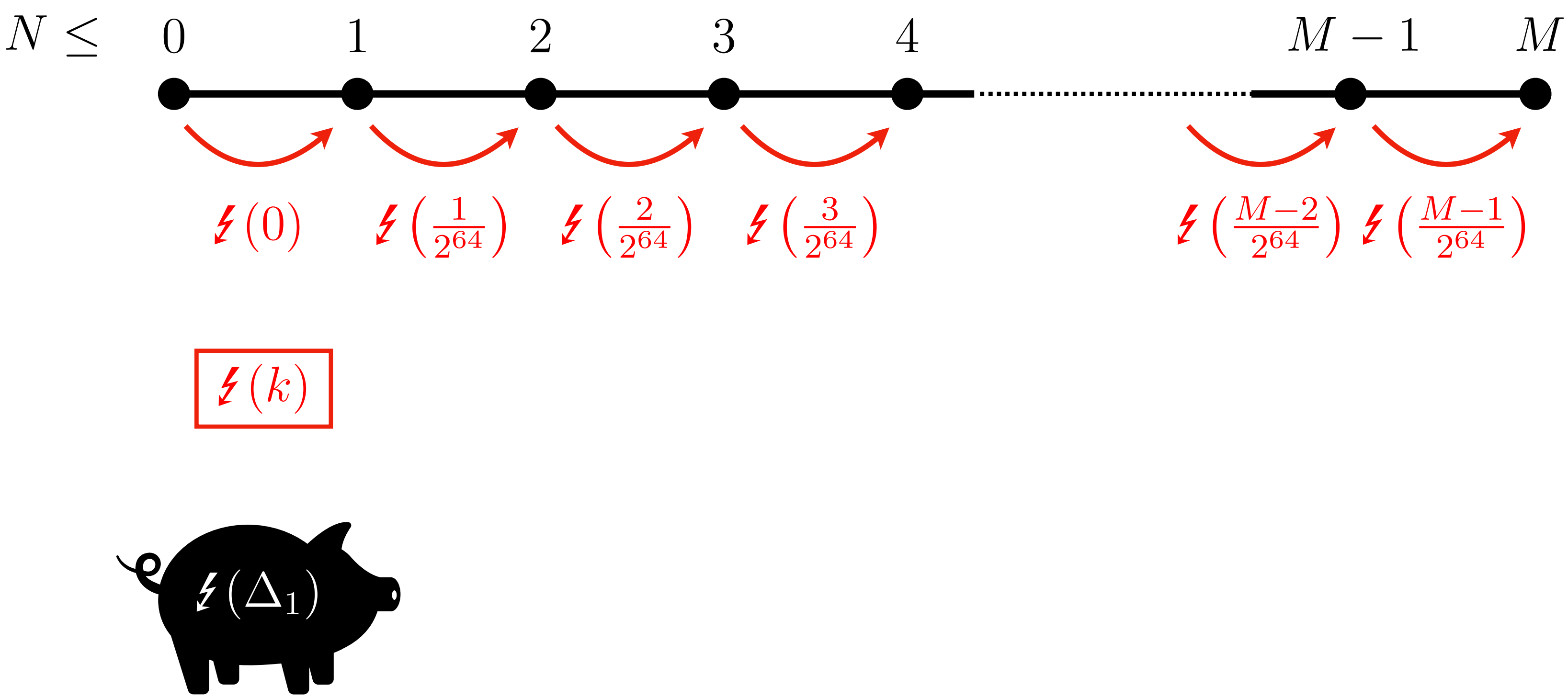
$$\left\{ \begin{array}{c} \text{collisionFree } N \ * \\ ↯(N \cdot 2^{-64}) \end{array} \right\} \text{hash } x \left\{ v. \begin{array}{c} \text{collisionFree } (N+1) \ * \\ \text{get } x = v \end{array} \right\}$$

Property: collisionFree N

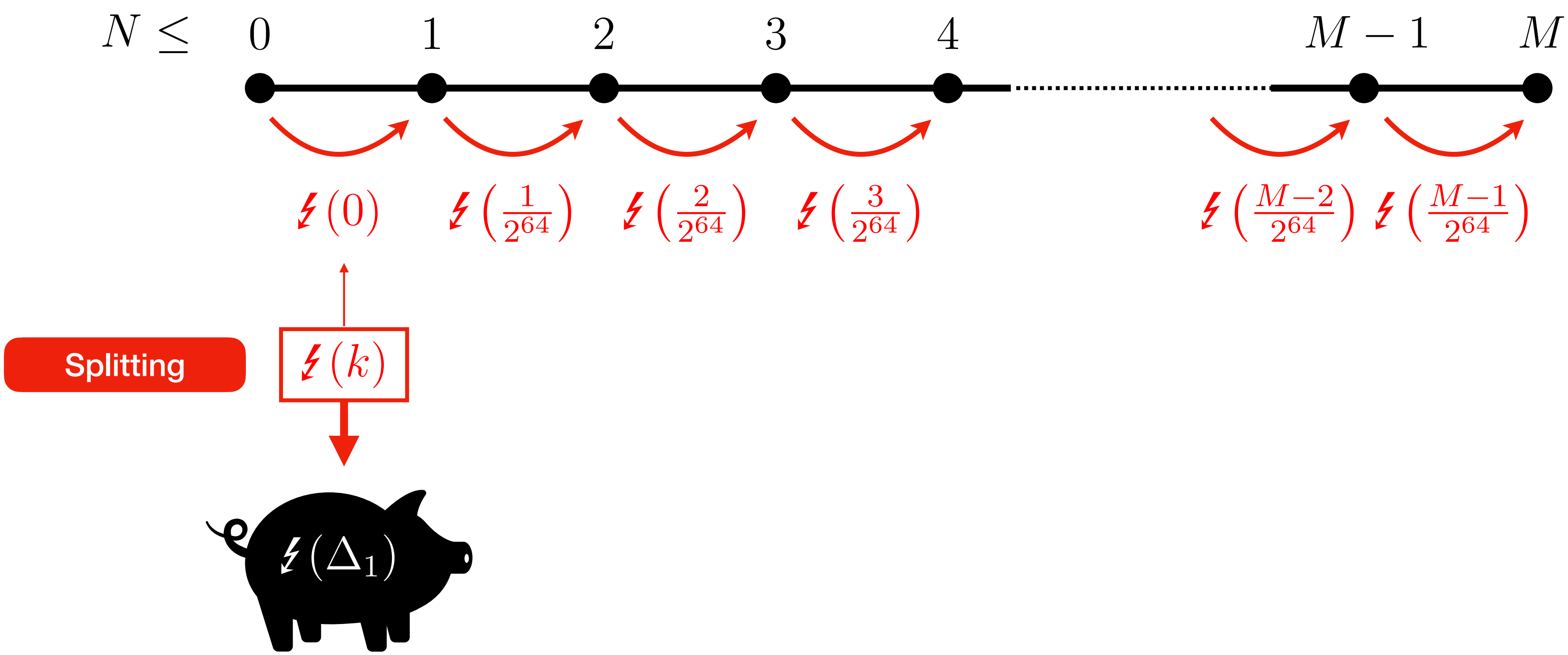
Amortized Credit Arithmetic



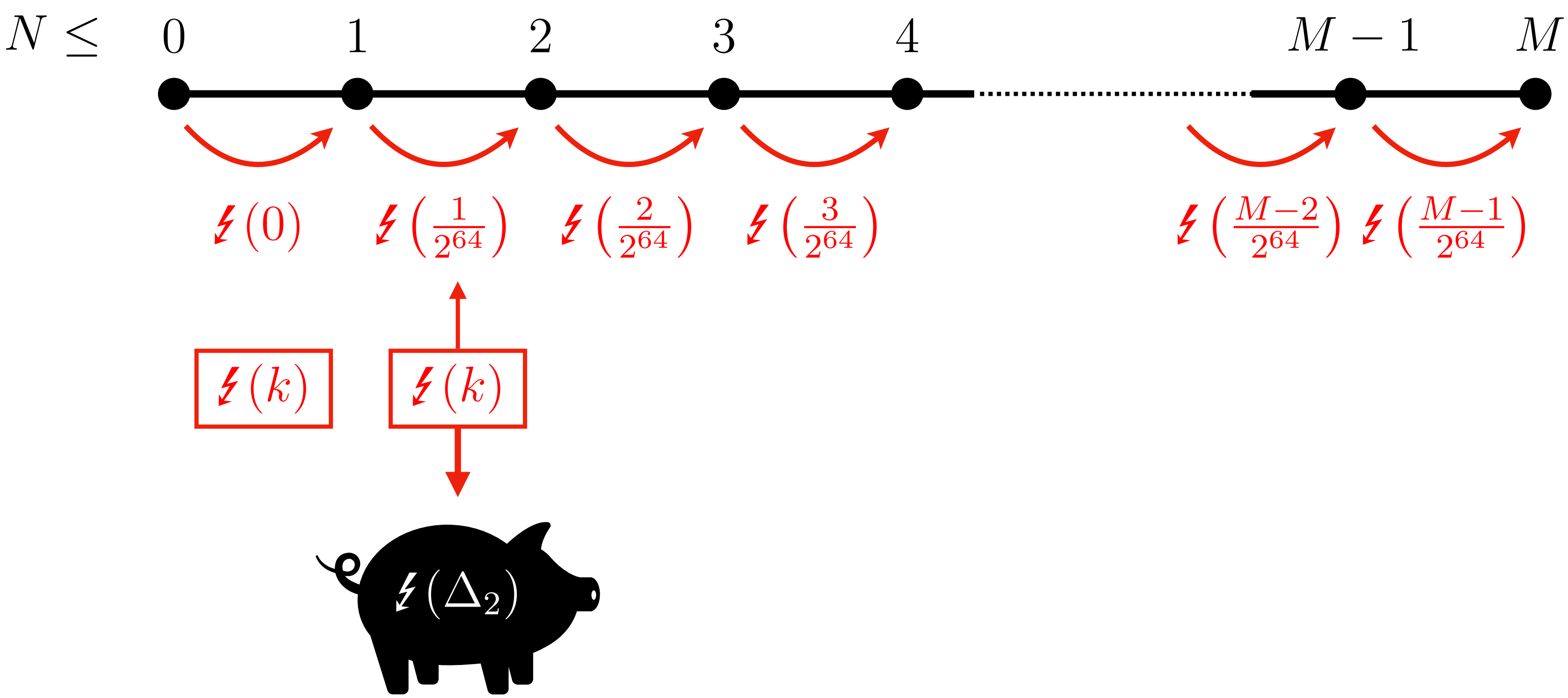
Amortized Credit Arithmetic



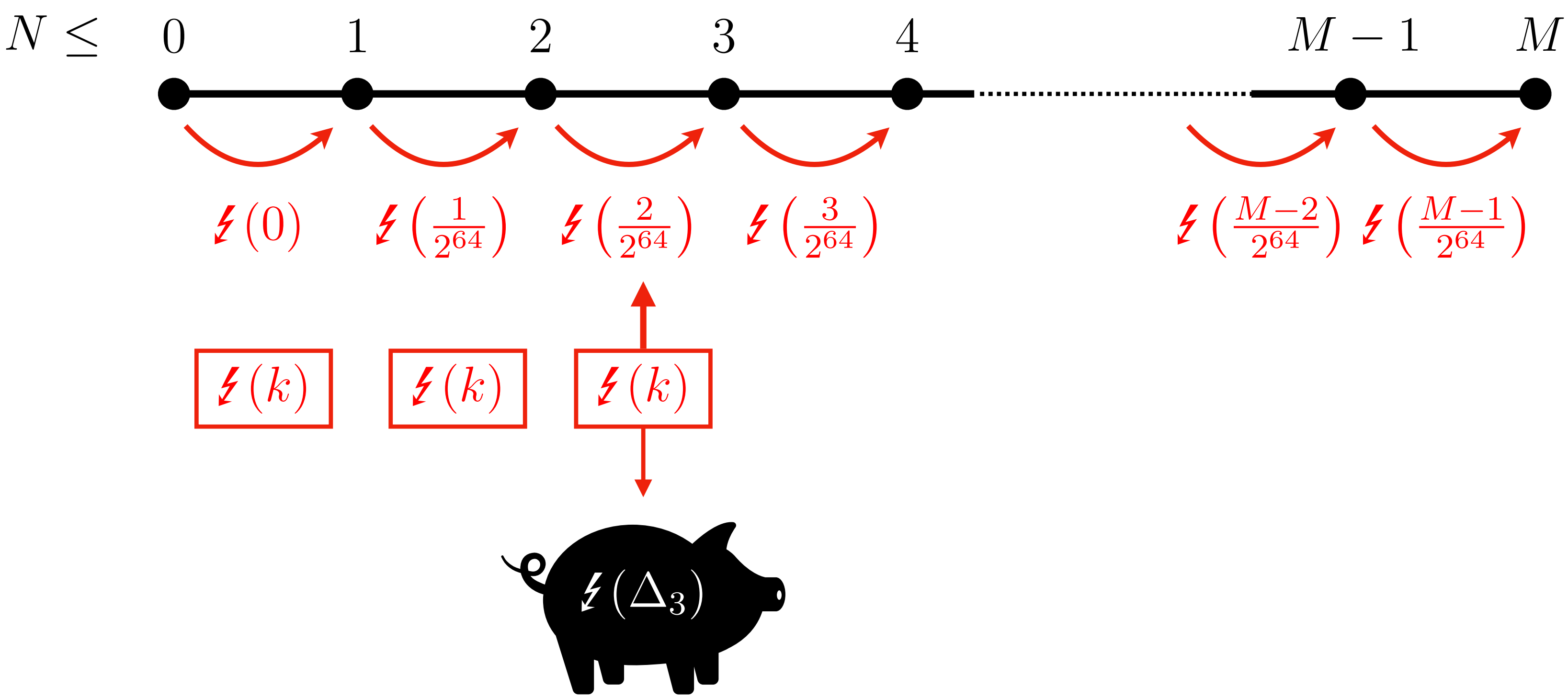
Amortized Credit Arithmetic



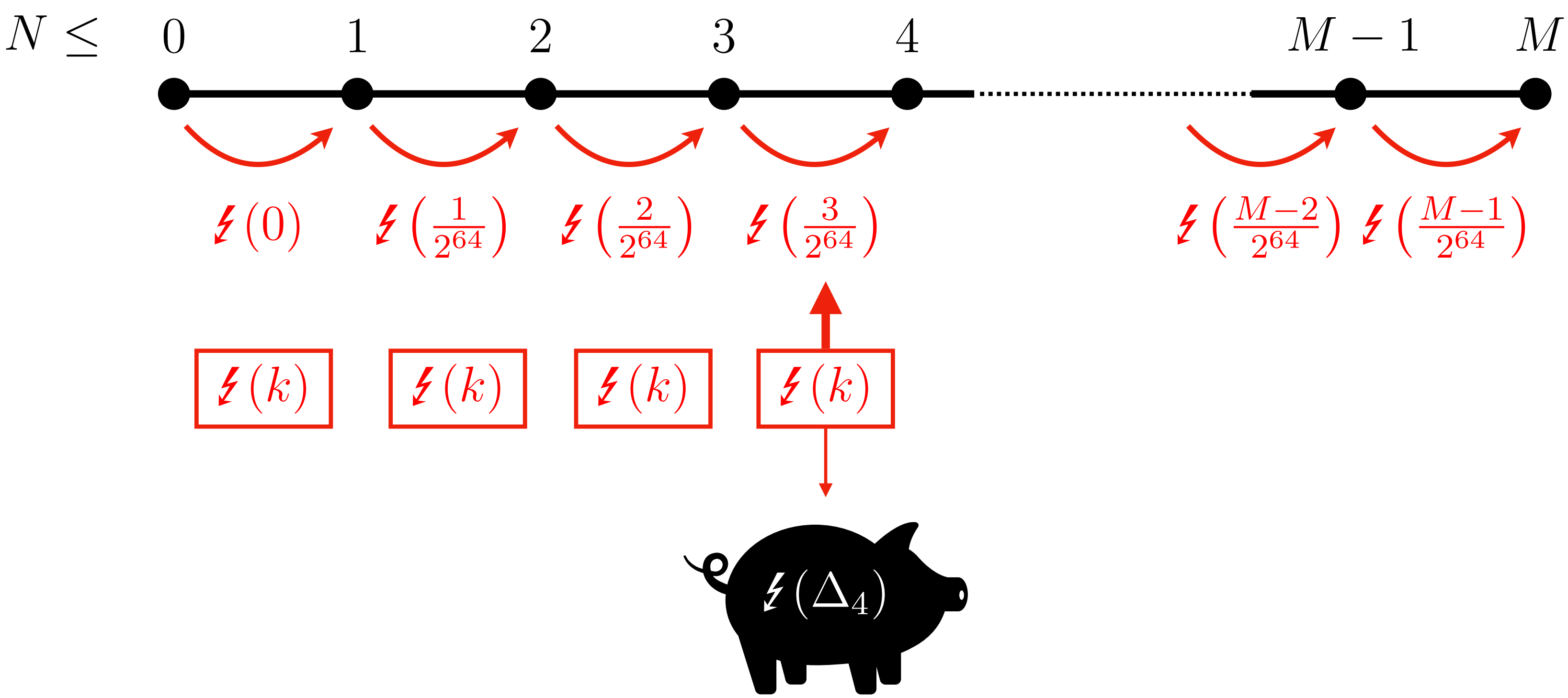
Amortized Credit Arithmetic



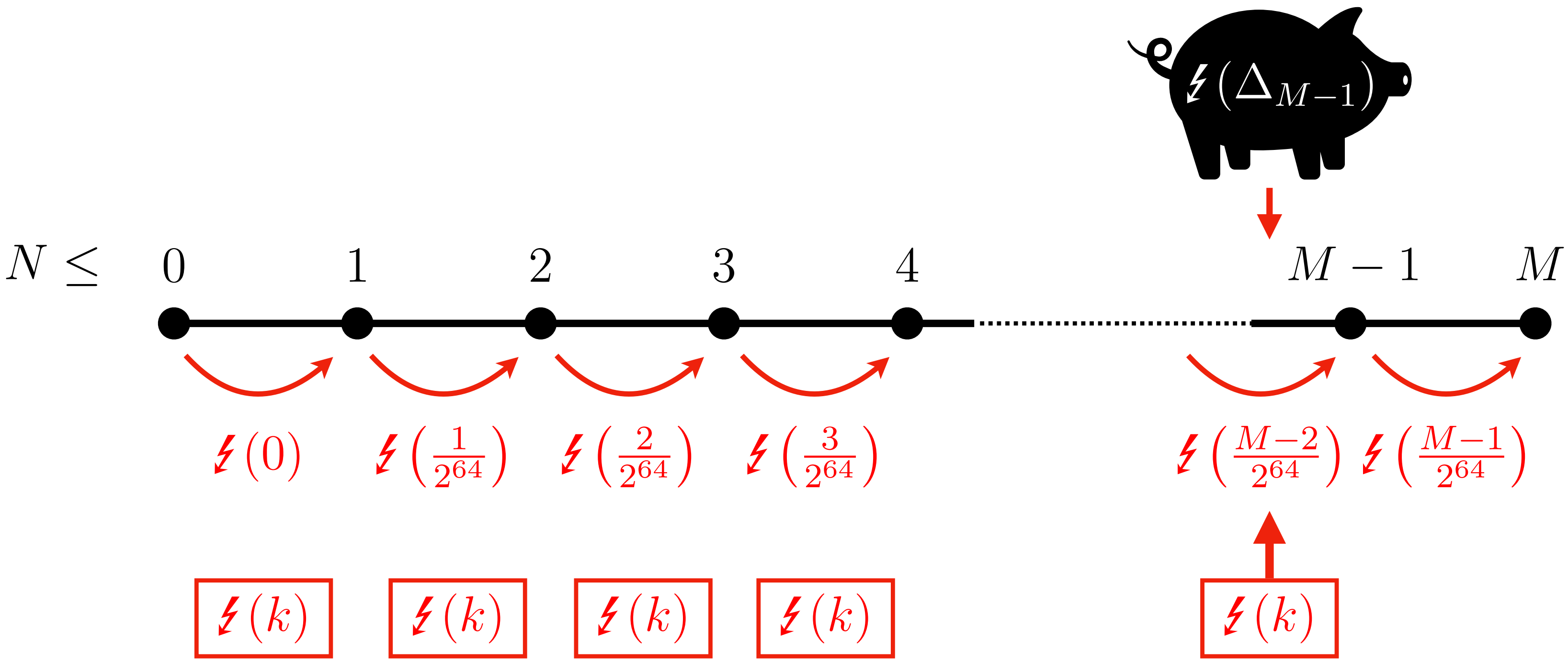
Amortized Credit Arithmetic



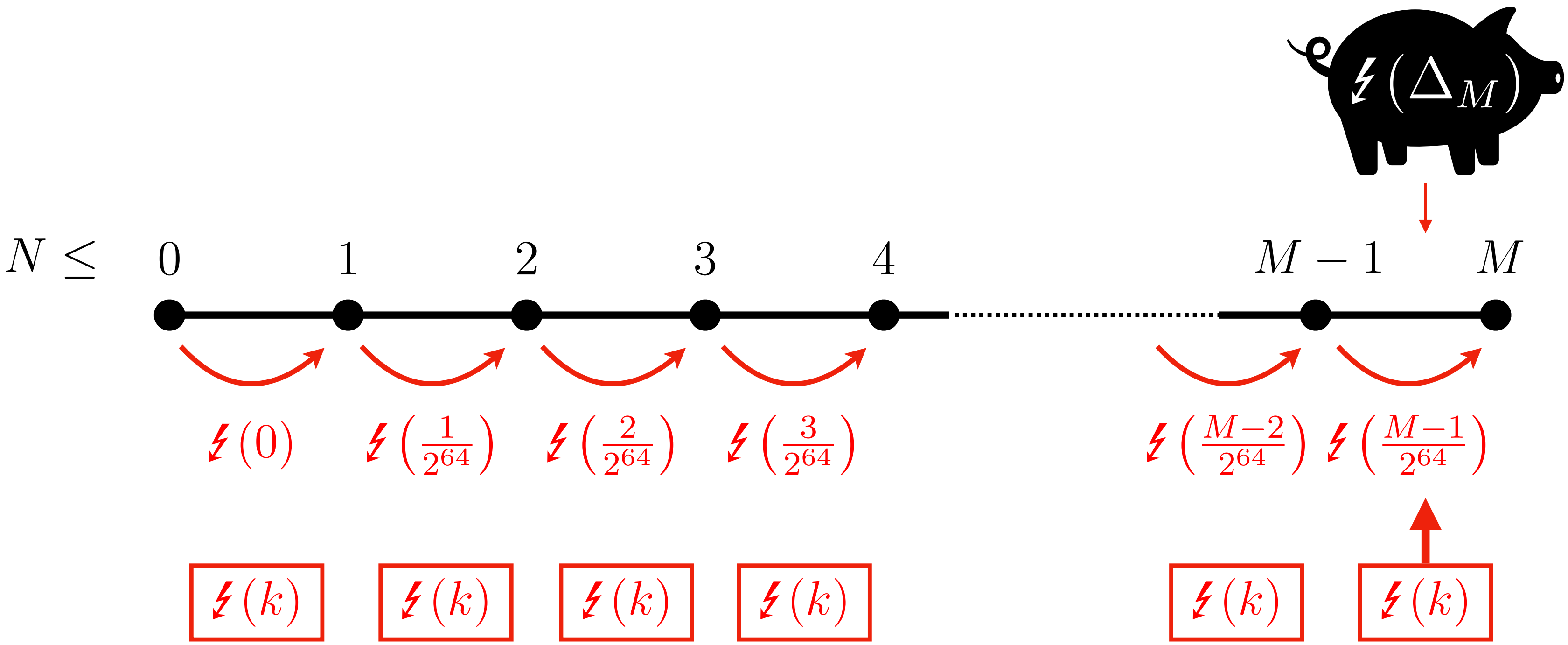
Amortized Credit Arithmetic



Amortized Credit Arithmetic



Amortized Credit Arithmetic



Hash Collisions

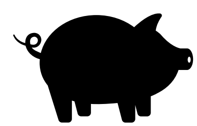
hash : $A \rightarrow$ int64

hash $x =$ match get x with
  Some $(v) \Rightarrow v$
  | None $\Rightarrow$ let $v =$ sample(2^{64}) in
    set x v;
    v
end

Amortize over M hashes

$$\left\{ \begin{array}{c} \text{collisionFree } N \ * \\ I(N) * N < M * ⚡(k) \end{array} \right\} \text{hash } x \left\{ \begin{array}{c} \text{collisionFree } (N+1) \ * \\ I(N+1) \end{array} \right\}$$

Derived!

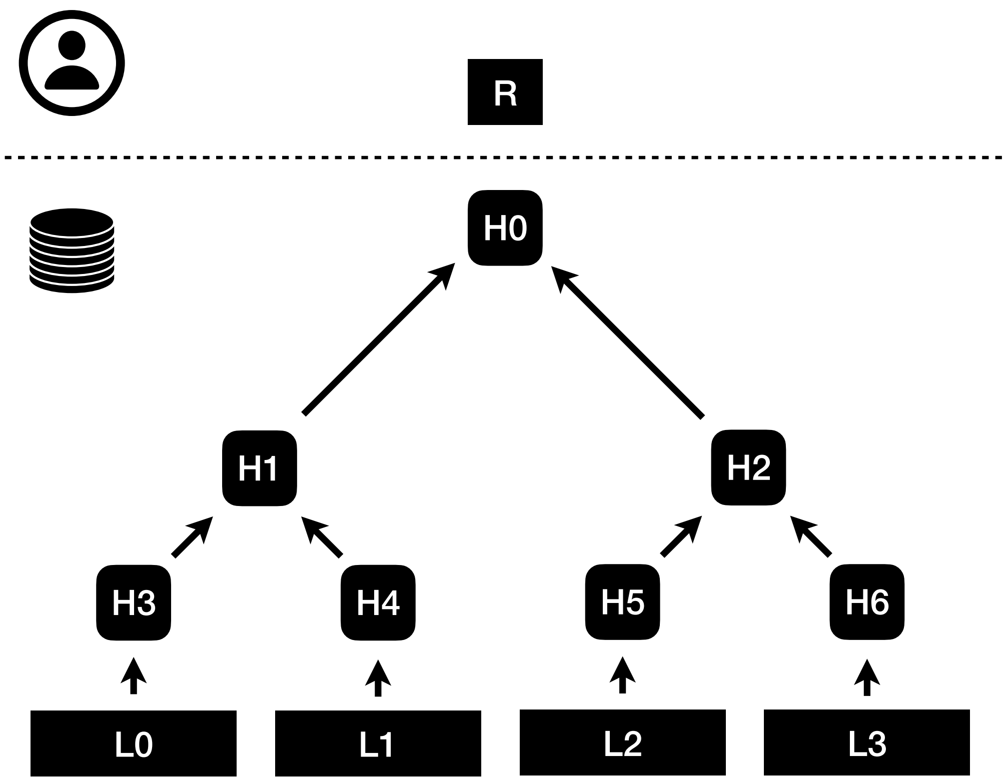


$$I(N) \triangleq (N \leq M) * ⚡(\Delta_N)$$

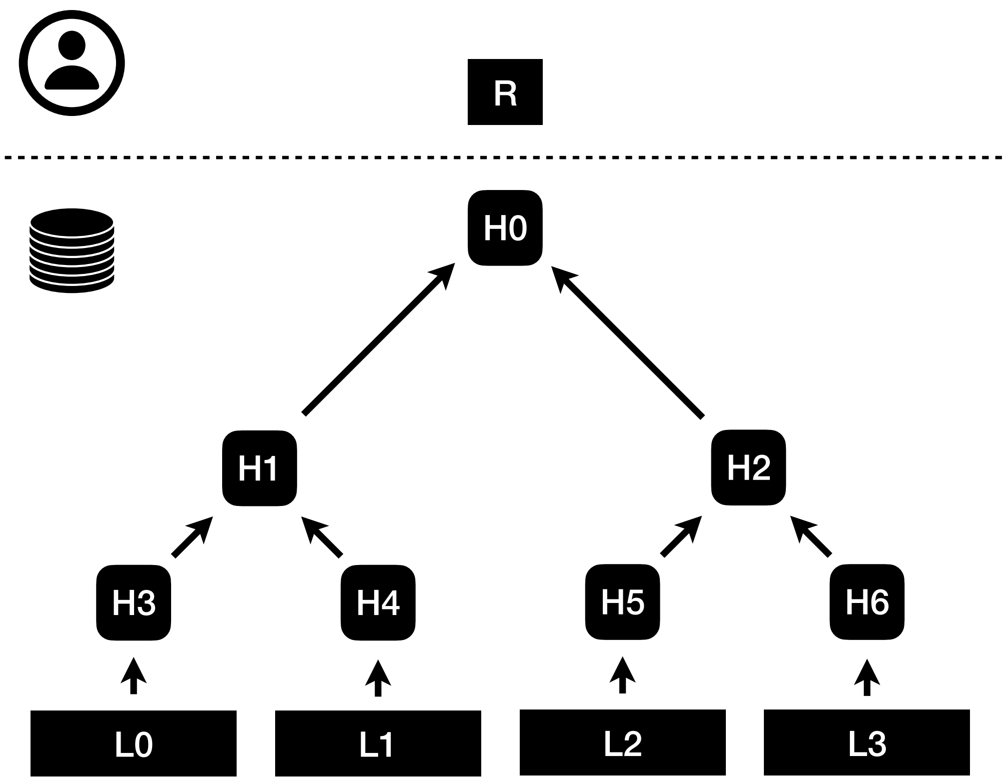
$$\left\{ \begin{array}{c} \text{collisionFree } N \ * \\ ⚡(N \cdot 2^{-64}) \end{array} \right\} \text{hash } x \left\{ v. \begin{array}{c} \text{collisionFree } (N+1) \ * \\ \text{get } x = v \end{array} \right\}$$

Property: collisionFree N

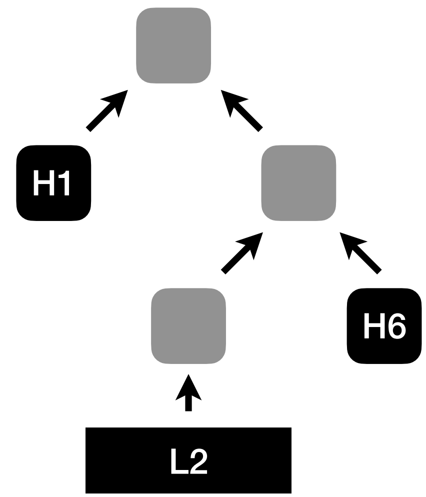
Merkle Tree



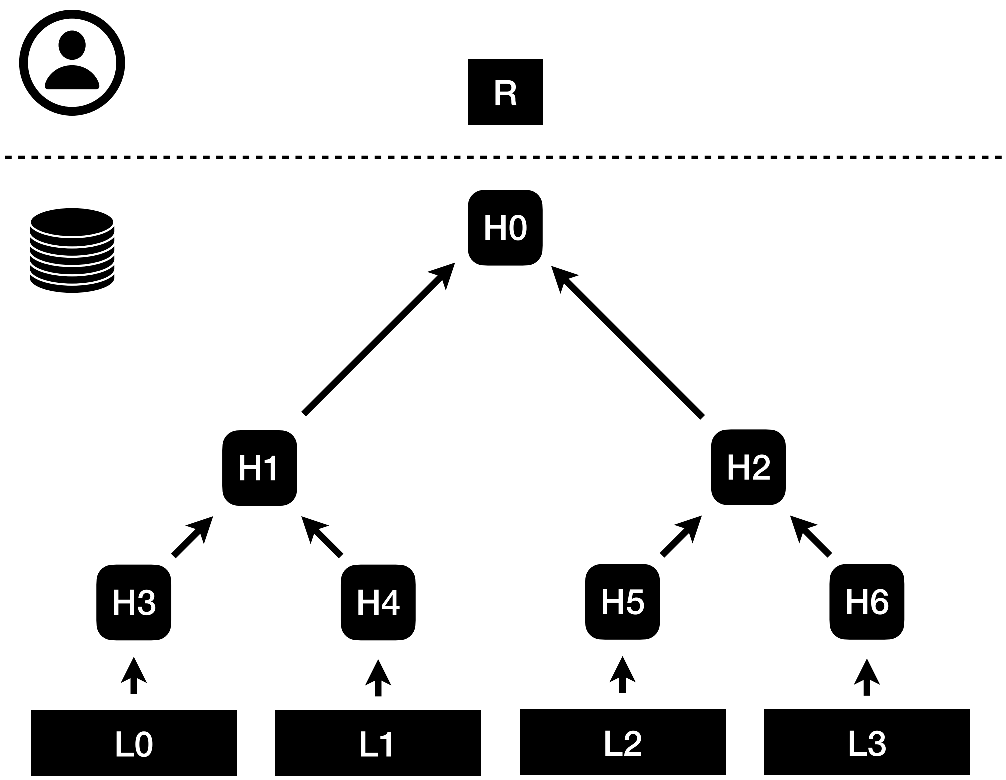
Merkle Tree



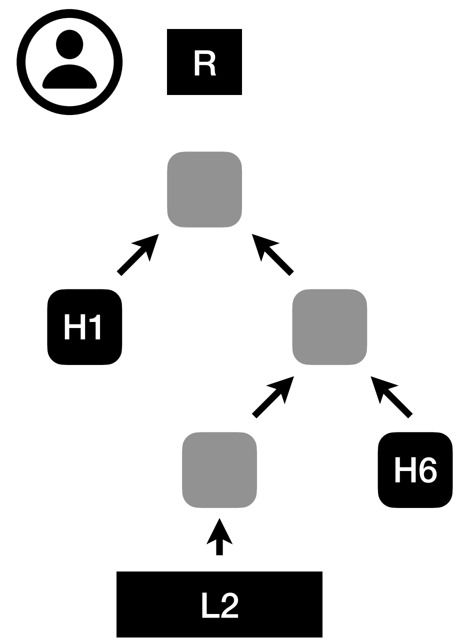
query(L2) =



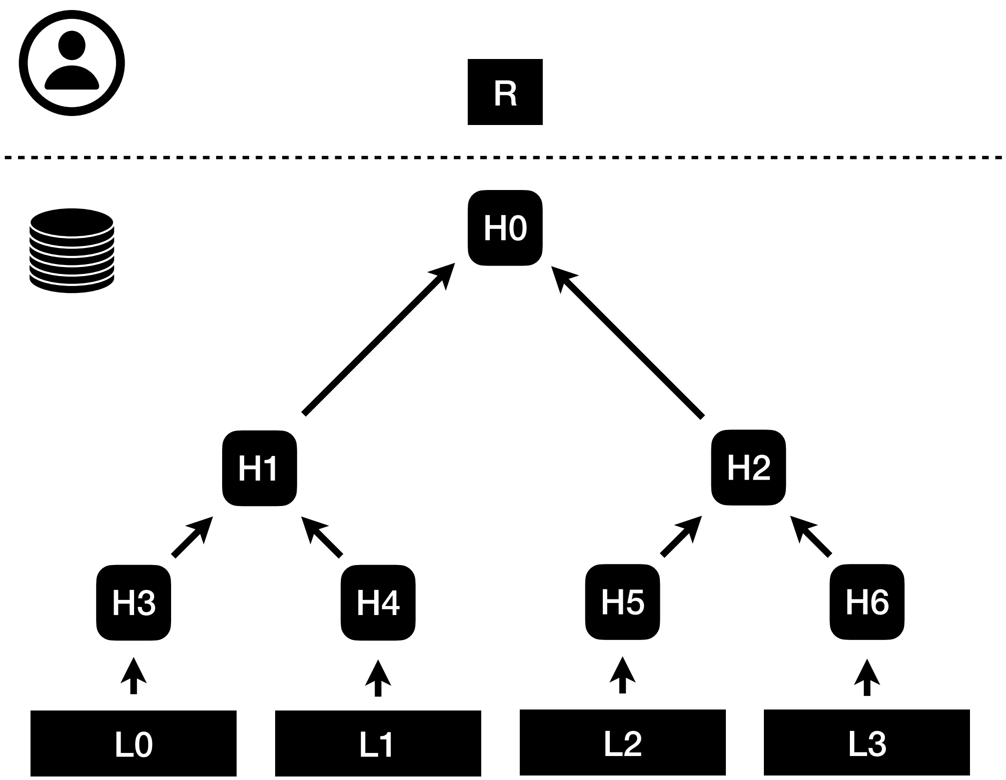
Merkle Tree



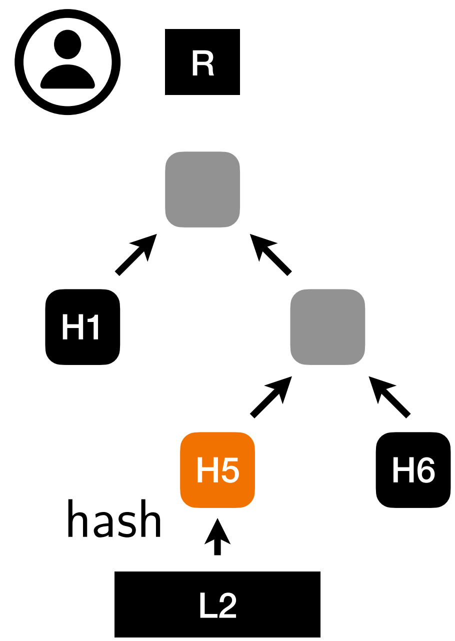
query(L2) =



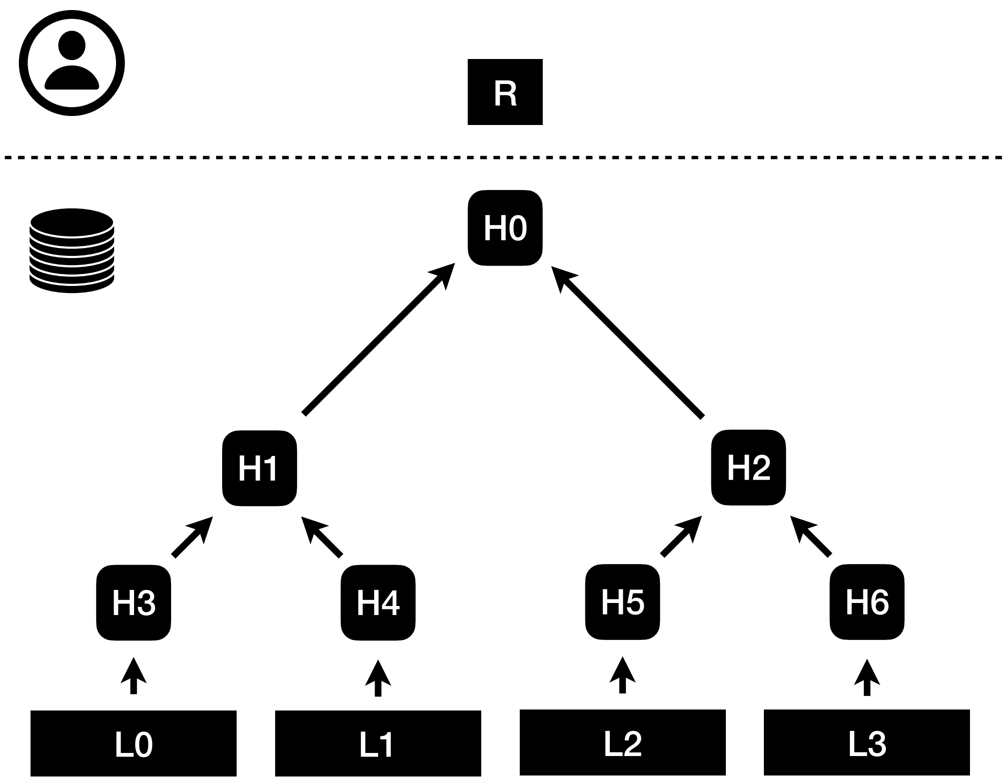
Merkle Tree



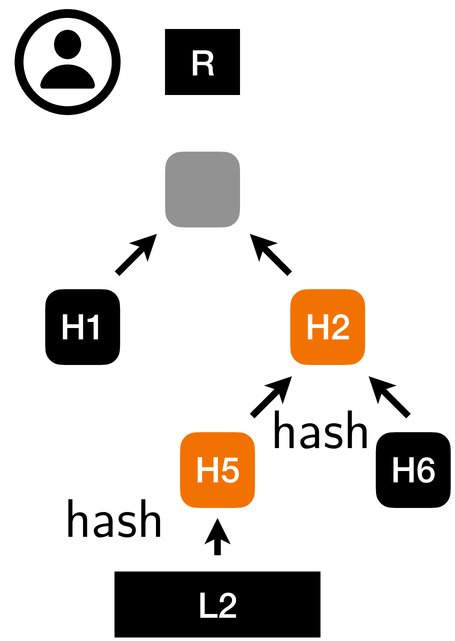
query(L2) =



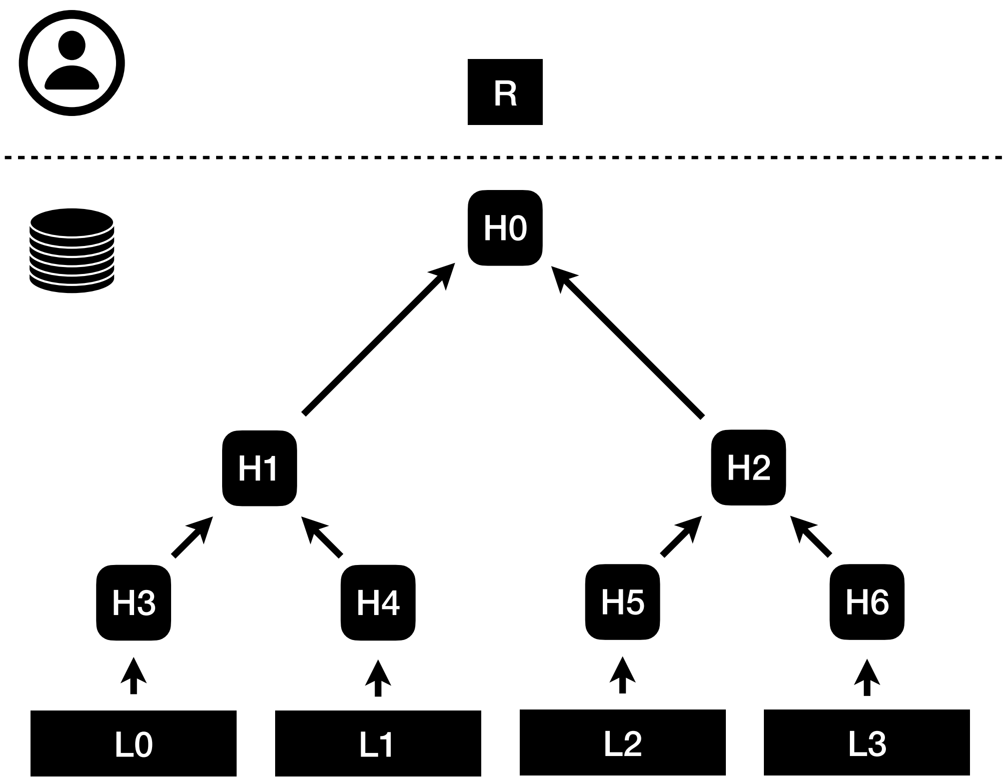
Merkle Tree



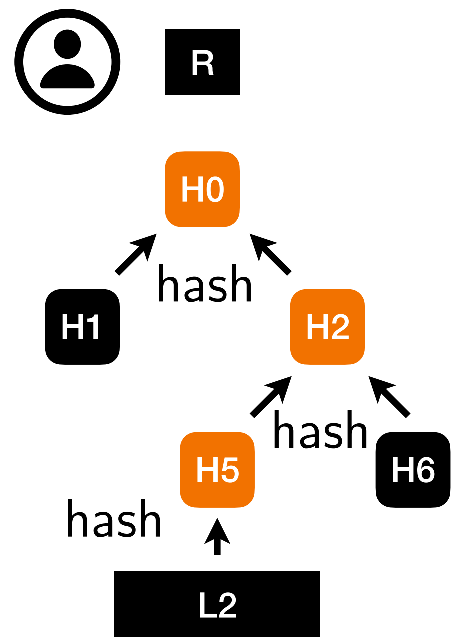
query(L2) =



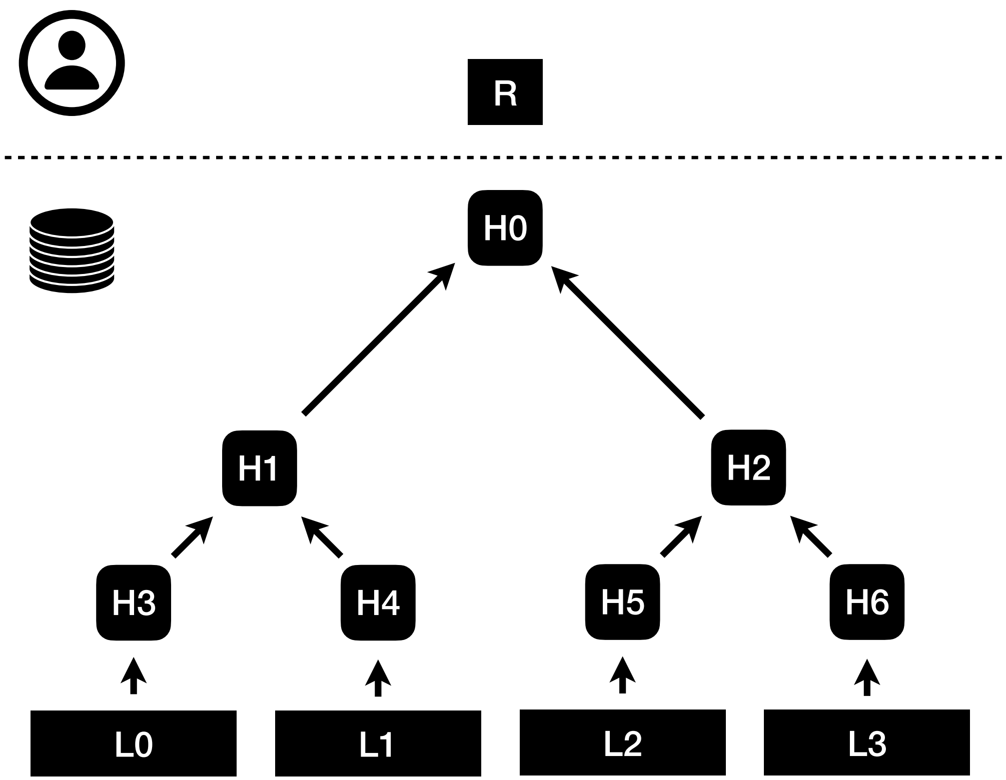
Merkle Tree



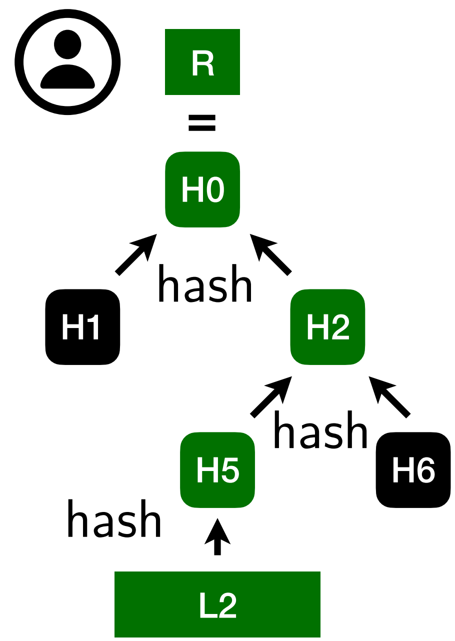
query(L2) =



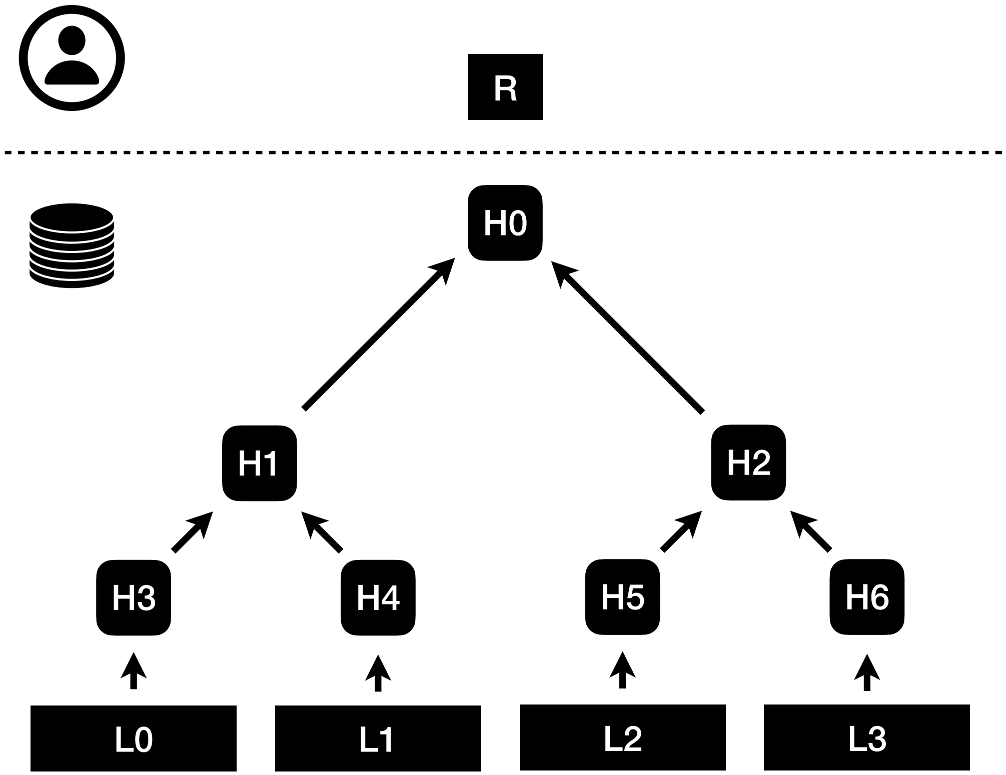
Merkle Tree



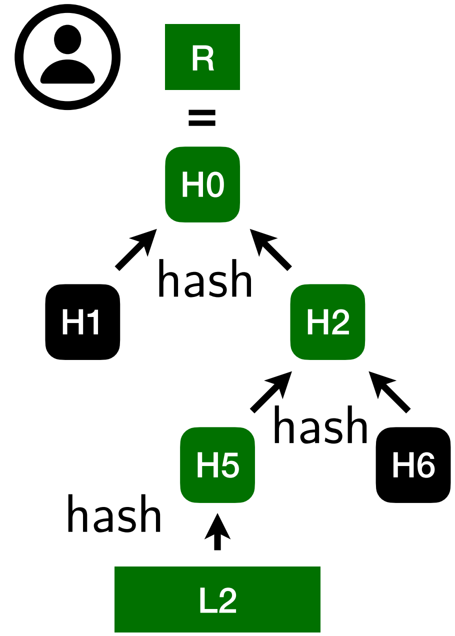
query(L2) =



Merkle Tree



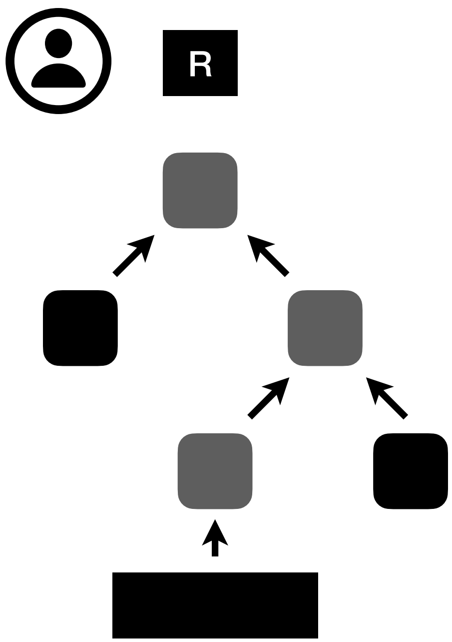
query(L2) =



What are the chances that arbitrarily corrupted data will pass this check?

Merkle Tree

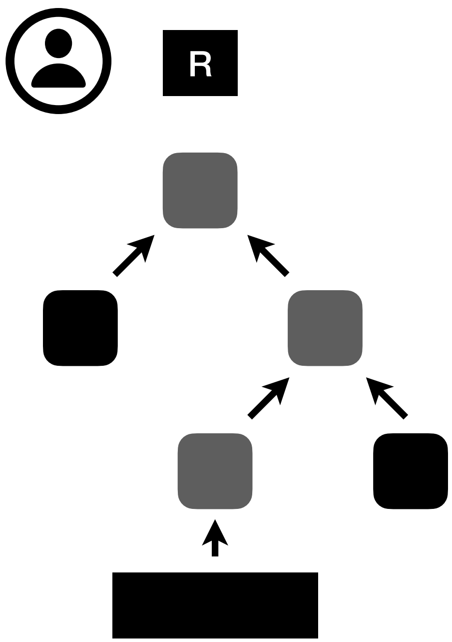
What are the chances that arbitrarily corrupted data will pass this check?



- **Validation program** check

Merkle Tree

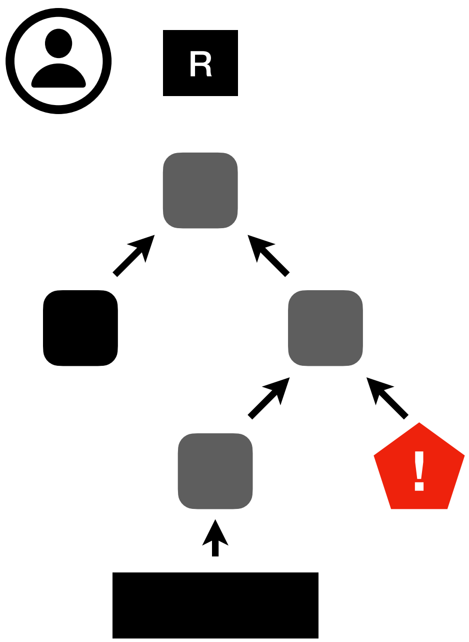
What are the chances that arbitrarily corrupted data will pass this check?



- **Validation program** check
- **Collision free** ⇒ check **is sound**

Merkle Tree

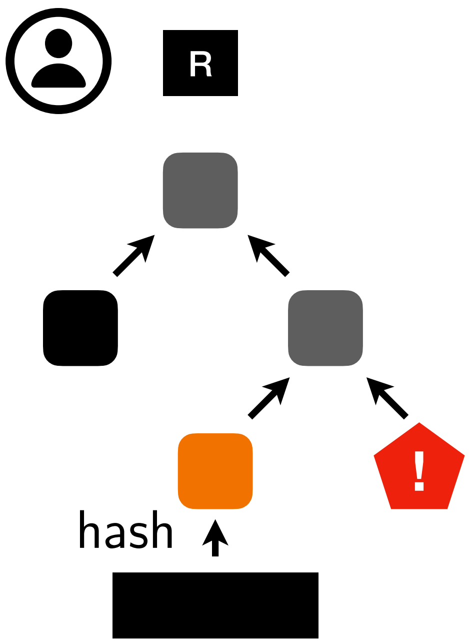
What are the chances that arbitrarily corrupted data will pass this check?



- **Validation program** check
- **Collision free** $\Rightarrow$ check **is sound**

Merkle Tree

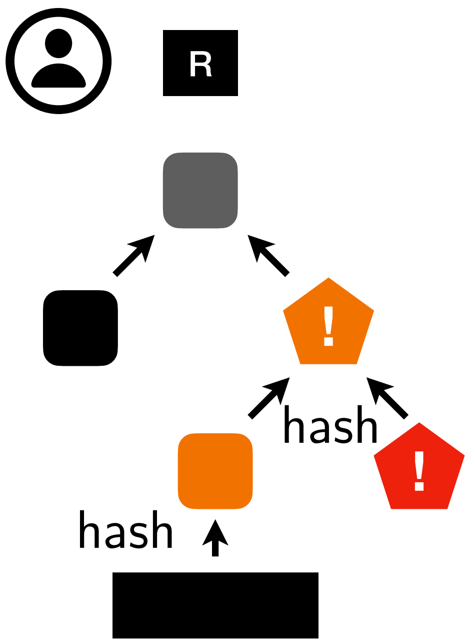
What are the chances that arbitrarily corrupted data will pass this check?



- **Validation program** check
- **Collision free** $\Rightarrow$ check **is sound**

Merkle Tree

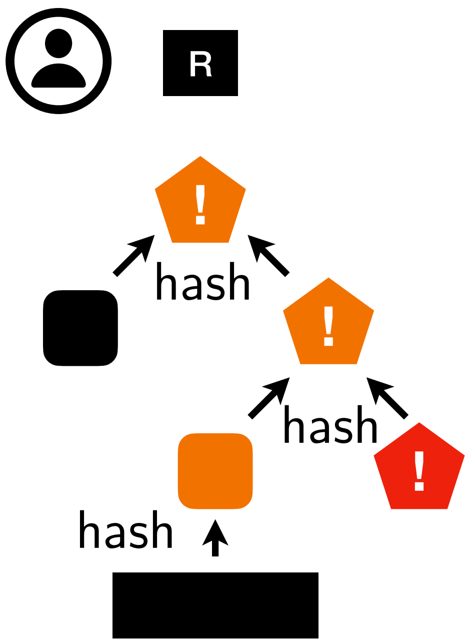
What are the chances that arbitrarily corrupted data will pass this check?



- **Validation program** check
- **Collision free ⇒** check **is sound**

Merkle Tree

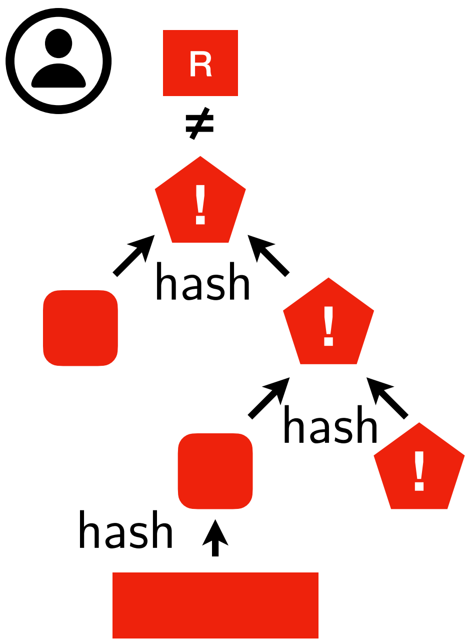
What are the chances that arbitrarily corrupted data will pass this check?



- **Validation program** check
- **Collision free ⇒** check **is sound**

Merkle Tree

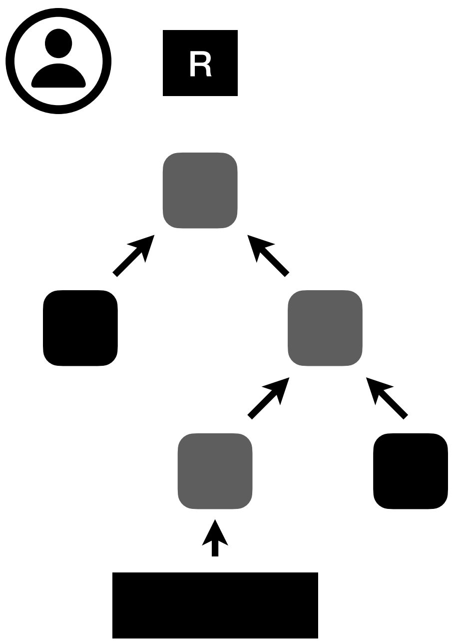
What are the chances that arbitrarily corrupted data will pass this check?



- **Validation program** check
- **Collision free ⇒** check **is sound**

Merkle Tree

What are the chances that arbitrarily corrupted data will pass this check?

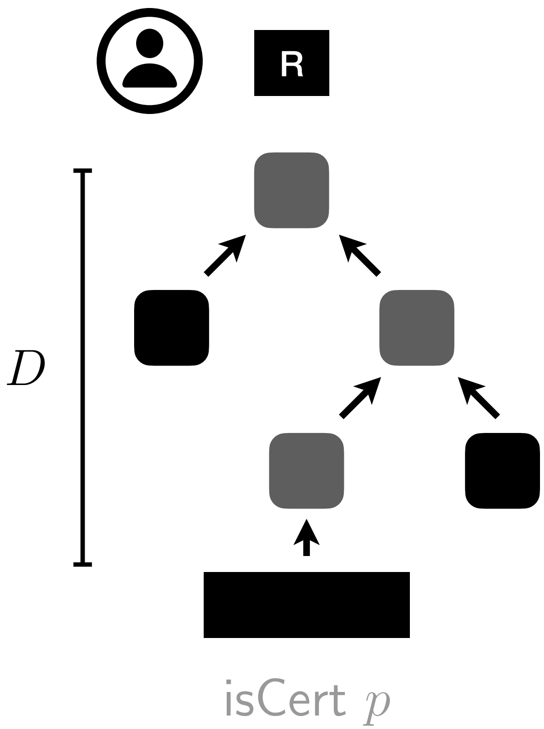


- **Validation program** check
- **Collision free ⇒** check **is sound**

$$\left\{ \begin{array}{c} \text{collisionFree } N * \\ I(N) * N < M * \lightning(k) \end{array} \right\} \text{hash } x \left\{ \begin{array}{c} \text{collisionFree } (N+1) * \\ I(N+1) \end{array} \right\}$$

Merkle Tree

What are the chances that arbitrarily corrupted data will pass this check?

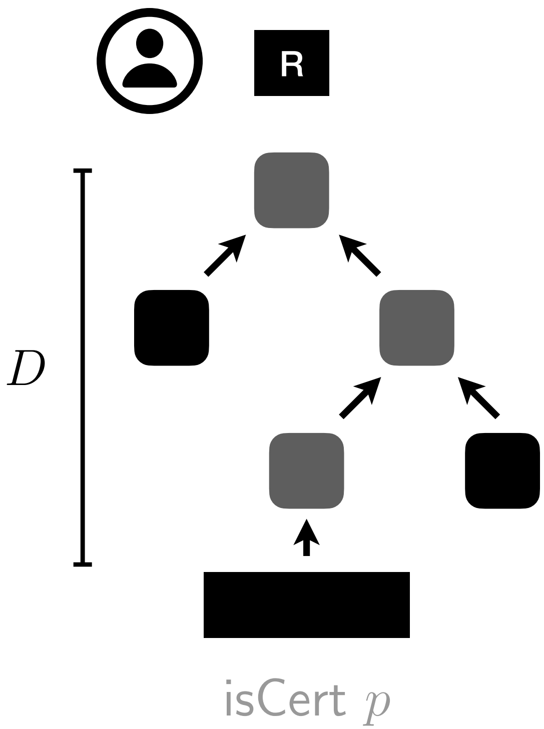


- **Validation program** check
- **Collision free ⇒** check **is sound**

$$\left\{ \begin{array}{c} \text{collisionFree } N * \\ I(N) * N < M * \lightning(k) \end{array} \right\} \text{hash } x \left\{ \begin{array}{c} \text{collisionFree } (N+1) * \\ I(N+1) \end{array} \right\}$$

Merkle Tree

What are the chances that arbitrarily corrupted data will pass this check?

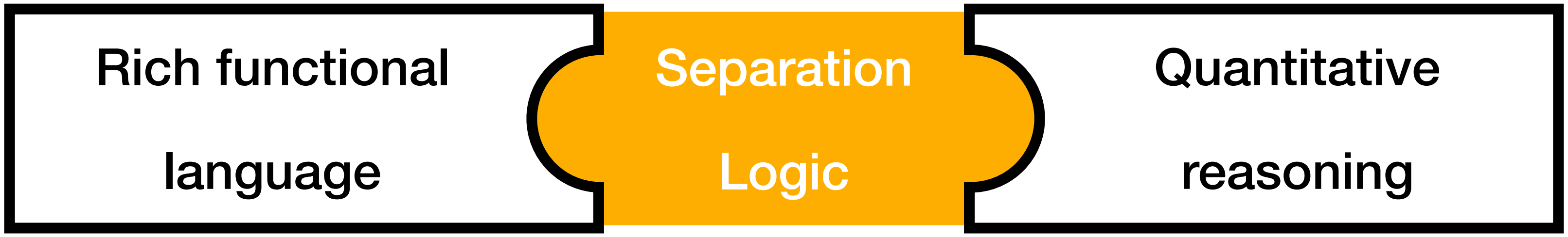


- **Validation program** check
- **Collision free** $\Rightarrow$ check **is sound**

$$\left\{ \begin{array}{c} \text{collisionFree } N \ast \\ I(N) \ast N < M \ast \lightning(k) \end{array} \right\} \text{hash } x \left\{ \begin{array}{c} \text{collisionFree } (N+1) \ast \\ I(N+1) \end{array} \right\}$$

$$\left\{ \begin{array}{c} \text{collisionFree } N \ast I(N) \ast \\ N + D < M \ast \text{isCert } p \ast \\ \lightning(k \cdot D) \end{array} \right\} \text{check } p \left\{ \begin{array}{c} \text{collisionFree } (N+D) \ast \\ I(N+D) \end{array} \right\}$$

At most $\lightning(k \cdot D)$



Expected values as state

Challenge 1.

Approximate Correctness

- Expected error bounds as a separation logic resource
- Derived aHL rules, amortized reasoning
- Modular proofs of approximate correctness

Challenge 2.

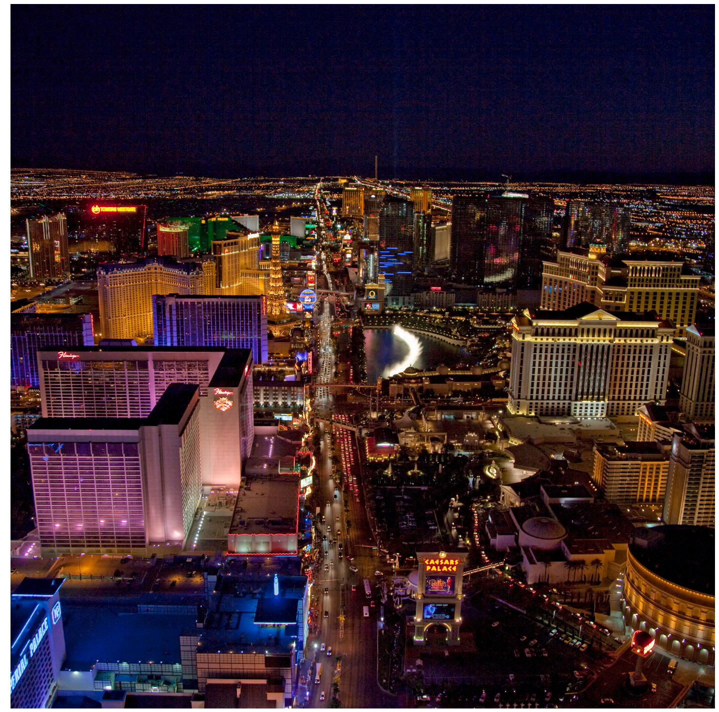
Almost-Sure Termination

Monte Carlo

- *Always terminates*
- *May be incorrect*

Monte Carlo

- *Always terminates*
- *May be incorrect*



Las Vegas

- *May not terminate*
- *Always correct*

Total Error Credits

Total Eris

Termination Bounds as a Resource

$$\vdash [\lightning(\epsilon)]\ f\ [v.\, P]$$

f terminates with value v <u>and</u>

$P\,v$ holds, with probability $1 - \epsilon$.

Iris

Step-indexed & higher-order

Mechanized in Rocq

Eris

$\vdash \{ \lightning(\epsilon) \}\ f\ \{ P \}$

Total Eris

$\vdash [\lightning(\epsilon)]\ f\ [P]$

Iris **Step-indexed & higher-order**

Error Credits with **Spending** **Splitting** **Averaging**

Eris

$\vdash \{ ↯(\epsilon) \} \; f \; \{ P \}$

Total Eris

$\vdash [↯(\epsilon)] \; f \; [P]$

Iris Step-indexed & higher-order

Error Credits with Spending Splitting Averaging

Recursion rule:

To prove

$$\vdash \{P\} \, (\mathsf{rec} \; f \; x \; = \; e) \; v \; \{Q\}$$

assume

$$\forall w. \{P\} \, (\mathsf{rec} \; f \; x \; = \; e) \; w \; \{Q\}$$

and show

$$\vdash \{P\} \; e[v/x][(\mathsf{rec} \; f \; x \; = \; e)/f] \; \{Q\}$$

Eris

$$\vdash \{\lightning(\epsilon)\}\ f\ \{P\}$$

Total Eris

$$\vdash [\lightning(\epsilon)]\ f\ [P]$$

Iris **Step-indexed & higher-order**

Error Credits with Spending Splitting Averaging

Recursion rule:

To prove

$$\vdash \{P\}\ (\text{rec}\ f\ x = e)\ v\ \{Q\}$$

assume

$$\forall w. \{P\}\ (\text{rec}\ f\ x = e)\ w\ \{Q\}$$

and show

$$\vdash \{P\}\ e[v/x][(\text{rec}\ f\ x = e)/f]\ \{Q\}$$

Recursion rule does not hold!

Error Induction

Rejection Sampling

$\text{rec roll5}\ \_ =$

$\quad \text{let roll} = 1 + \text{sample}\, 6\ \text{in}$

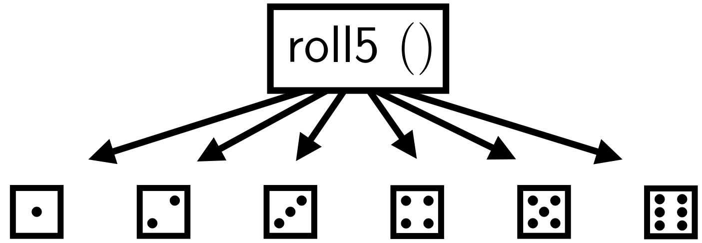
$\quad \text{if}\ (\text{roll} < 6)$

$\quad\quad \text{then roll}$

$\quad\quad \text{else roll5}\ ()$

Error Induction

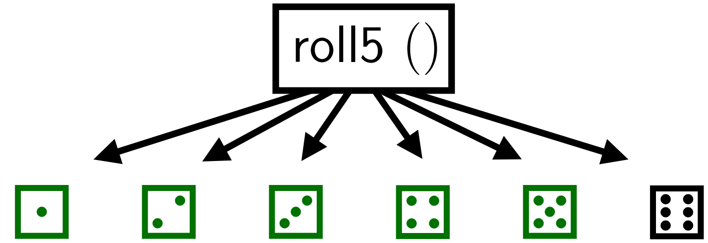
Rejection Sampling




```
rec roll5 _ =
  let roll = 1 + sample 6 in
  if (roll < 6)
    then roll
    else roll5 ()
```

Error Induction

Rejection Sampling



$\text{rec roll5 } \_ =$

$\quad \text{let roll} = 1 + \text{sample } 6 \text{ in}$

$\quad \text{if } (\text{roll} < 6)$

$\qquad \text{then roll}$

$\qquad \text{else roll5 } ()$

Error Induction

Rejection Sampling

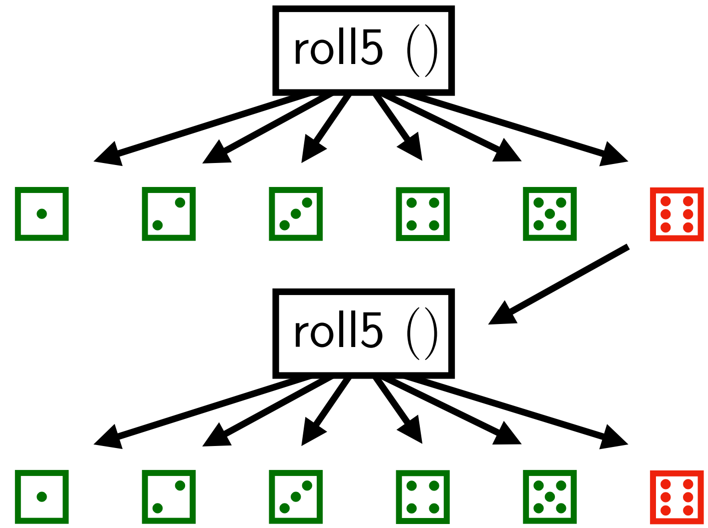
$\mathsf{rec}\ \mathsf{roll5}\ \_ =$

$\quad \mathsf{let}\ \mathsf{roll} = 1 + \mathsf{sample}\ 6\ \mathsf{in}$

$\quad \mathsf{if}\ (\mathsf{roll} < 6)$

$\quad\quad \mathsf{then}\ \mathsf{roll}$

$\quad\quad \mathsf{else}\ \mathsf{roll5}\ ()$



Error Induction

Rejection Sampling

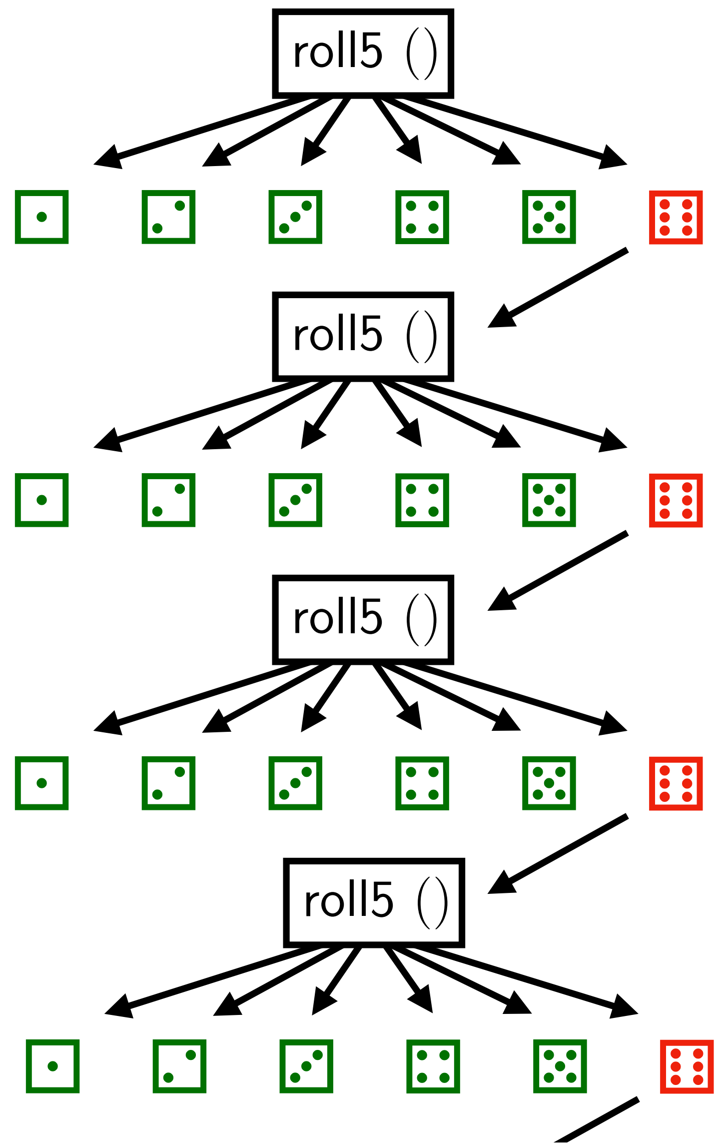
$\text{rec roll5 } \_ =$

$\quad \text{let roll} = 1 + \text{sample } 6 \text{ in}$

$\quad \text{if } (\text{roll} < 6)$

$\quad\quad \text{then roll}$

$\quad\quad \text{else roll5 } ()$



Error Induction

Rejection Sampling

$\mathsf{rec}\ \mathsf{roll5}\ \_ =$

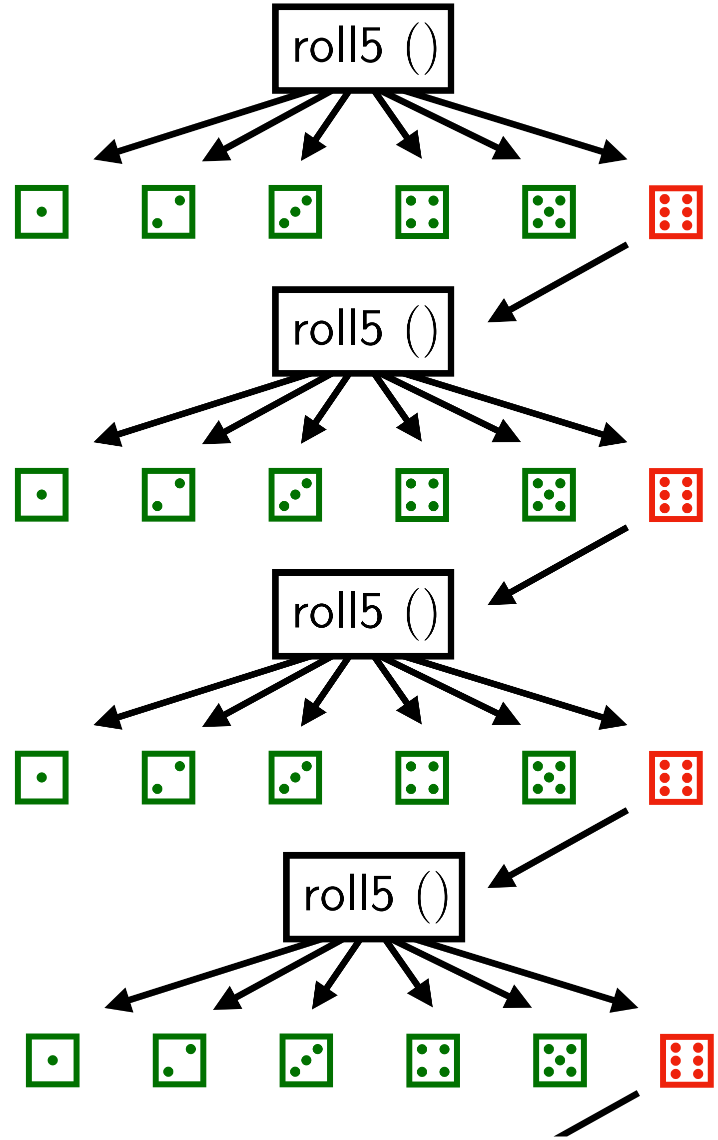
$\quad \mathsf{let}\ \mathsf{roll} = 1 + \mathsf{sample}\ 6\ \mathsf{in}$

$\quad \mathsf{if}\ (\mathsf{roll} < 6)$

$\quad\quad \mathsf{then}\ \mathsf{roll}$

$\quad\quad \mathsf{else}\ \mathsf{roll5}\ ()$

Prove $[\top]\ \mathsf{roll5}\ ()\ [v.\, v < 6]$ **?**



roll5 $()$

↯ (ϵ)

Error Induction

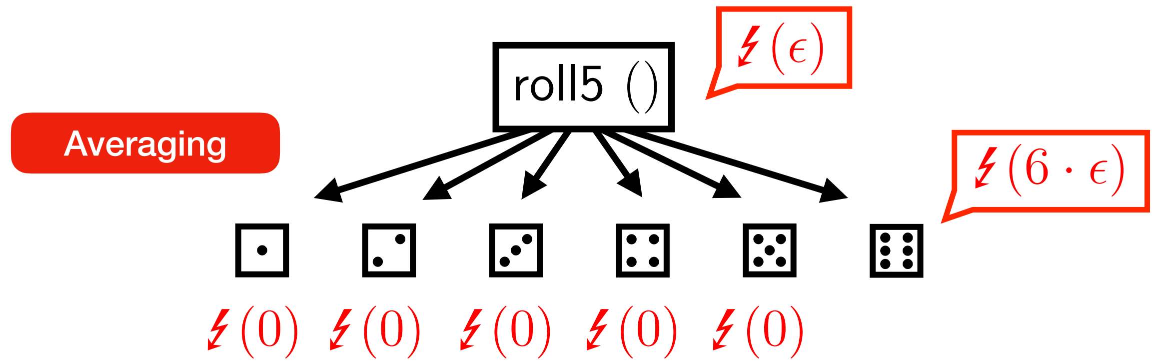
Rejection Sampling

Prove that for all $0 < \epsilon$

$$[↯(\epsilon)]\ \text{roll5}\ ()\ [v.\, v < 6]$$

Error Induction

Rejection Sampling

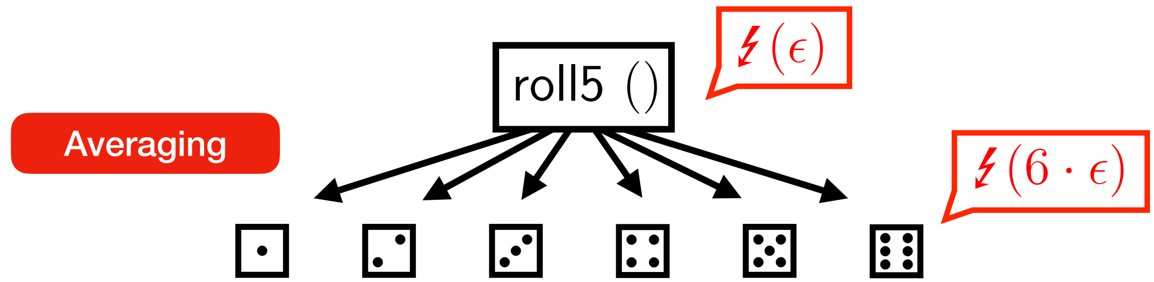


Prove that for all $0 < \epsilon$

$$[\lightning(\epsilon)]\ \mathsf{roll5}\ ()\ [v.\, v < 6]$$

Error Induction

Rejection Sampling



Prove that for all $0 < \epsilon$

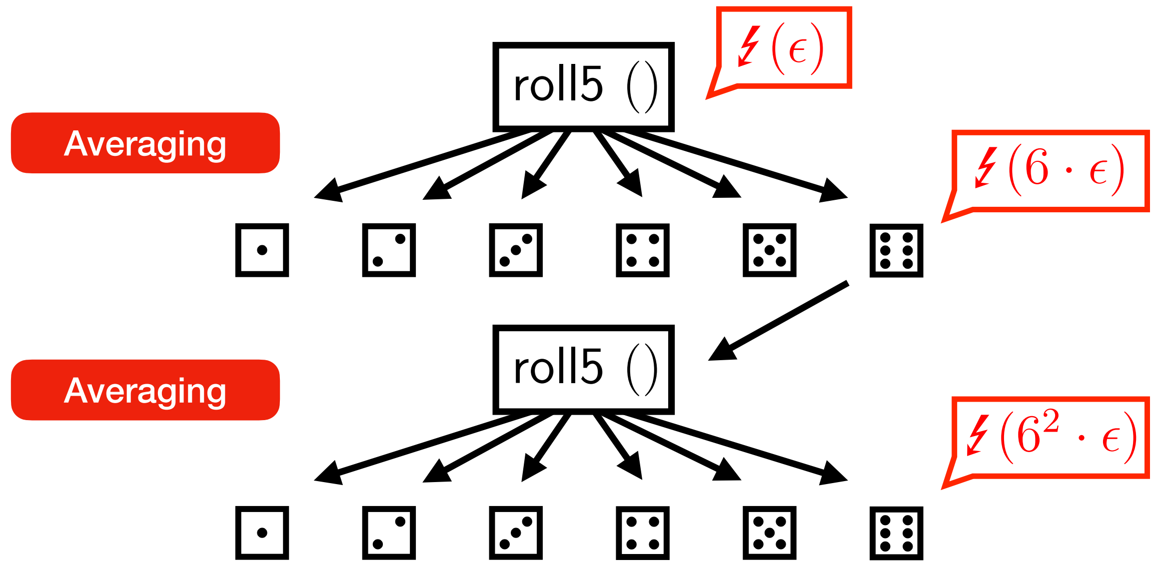
$$[\lightning(\epsilon)]\ \text{roll5}\ ()\ [v.\, v < 6]$$

Error Induction

Rejection Sampling

Prove that for all $0 < \epsilon$

$$[\lightning(\epsilon)]\ \text{roll5}\ ()\ [v.\, v < 6]$$



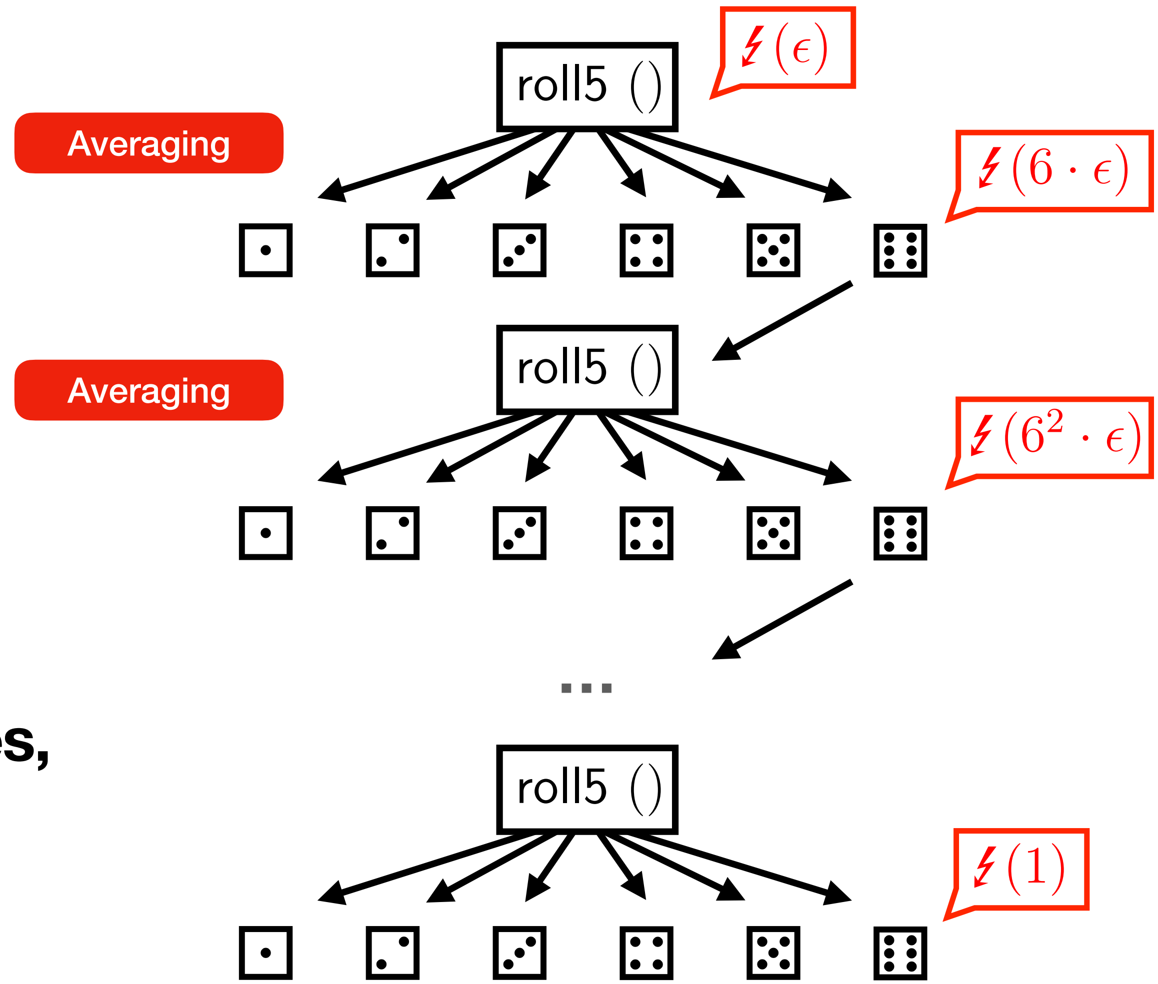
Error Induction

Rejection Sampling

Prove that for all $0 < \epsilon$

$$[\lightning(\epsilon)]\ \text{roll5}\ ()\ [v.\, v < 6]$$

Apply Averaging $\log_6(1/\epsilon)$ **times,**



Error Induction

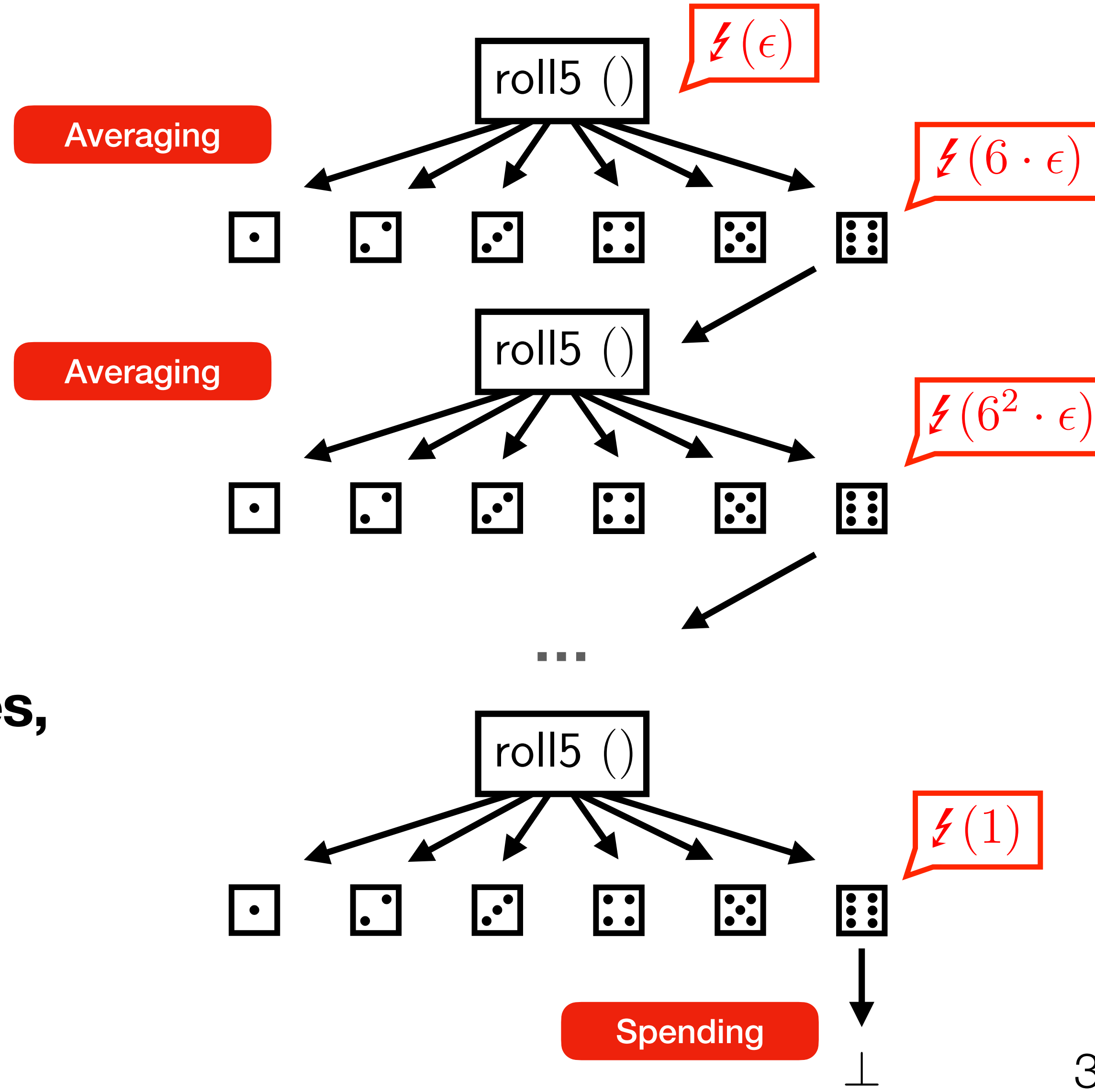
Rejection Sampling

Prove that for all $0 < \epsilon$

$$[⚡(\epsilon)]\ \text{roll5}\ ()\ [v.\, v < 6]$$

Apply Averaging $\log_6(1/\epsilon)$ **times,**

Apply Spending **once.**



Error Induction

Rejection Sampling

$$\forall \epsilon > 0, \vdash [\lightning(\epsilon)]\ \mathsf{roll5}\ ()\ [v.\, v < 6]$$

Error Induction

Rejection Sampling

Total Eris $\vdash [\lightning(\epsilon)]\ f\ [P]$

f terminates with value v and $P\,v$ holds, with probability $1 - \epsilon$

"roll5 terminates with a value less than 6 with arbitrarily high probability"

$$\forall \epsilon > 0, \vdash [\lightning(\epsilon)]\ \textsf{roll5}\ ()\ [v.\, v < 6]$$

Error Induction

Rejection Sampling

Total Eris $\vdash [\lightning(\epsilon)]\; f\; [P]$

f terminates with value v and $P\,v$ holds, with probability $1 - \epsilon$

"roll5 terminates with a value less than 6 with arbitrarily high probability"

$$\forall \epsilon > 0, \vdash [\lightning(\epsilon)]\ \mathsf{roll5}\ ()\ [v.\, v < 6]$$

Continuity ⇓

$$\vdash [\top]\ \mathsf{roll5}\ ()\ [v.\, v < 6]$$

"roll5 terminates with a value less than 6 with probability 1"

Error Induction

*Assume a **nonzero amount of credit**,*

*Prove that the **error increases** in every recursive case,*

*Perform induction on the **number of rounds**,*

*Conclude by **continuity**.*

Ask me about general forms!

Error Induction

WalkSAT

$$F \triangleq (\overline{\varphi_2} \vee \varphi_3 \vee \varphi_4) \wedge (\varphi_2 \vee \varphi_3 \vee \varphi_4) \wedge (\overline{\varphi_1} \vee \varphi_2 \vee \overline{\varphi_3})$$

Error Induction

WalkSAT

$$s \mapsto \begin{matrix} \varphi_1 & \varphi_2 & \varphi_3 & \varphi_4 \\ [\text{true}; & \text{false}; & \text{true}; & \text{false}] \end{matrix}$$

$$F \triangleq (\overline{\varphi_2} \vee \varphi_3 \vee \varphi_4) \wedge (\varphi_2 \vee \varphi_3 \vee \varphi_4) \wedge (\overline{\varphi_1} \vee \varphi_2 \vee \overline{\varphi_3})$$

Error Induction

WalkSAT

$$s \mapsto \begin{matrix} \varphi_1 & \varphi_2 & \varphi_3 & \varphi_4 \\ [\textsf{true}; & \textsf{false}; & \textsf{true}; & \textsf{false}] \end{matrix}$$

$$F \triangleq (\overline{\varphi_2} \lor \varphi_3 \lor \varphi_4) \land (\varphi_2 \lor \varphi_3 \lor \varphi_4) \land (\overline{\varphi_1} \lor \varphi_2 \lor \overline{\varphi_3})$$

"Pick a random variable in the first UNSAT clause, and flip it"

Error Induction

WalkSAT

$$
\begin{array}{ccccc} & \varphi_1 & \varphi_2 & \varphi_3 & \varphi_4 \\ s \mapsto & [\text{true}; & \text{false}; & \text{true}; & \text{false}] \end{array}
$$

$$
F \triangleq (\overline{\varphi_2} \vee \varphi_3 \vee \varphi_4) \wedge (\varphi_2 \vee \varphi_3 \vee \varphi_4) \wedge \boxed{(\overline{\varphi_1} \vee \varphi_2 \vee \overline{\varphi_3})}
$$

“Pick a random variable in the first UNSAT clause, and flip it”

Error Induction

WalkSAT

$$s \mapsto \begin{matrix} \varphi_1 & \varphi_2 & \varphi_3 & \varphi_4 \\ [\textsf{true}; & \textsf{false}; & \textsf{true}; & \textsf{false}] \end{matrix}$$

$$F \triangleq (\overline{\varphi_2} \vee \varphi_3 \vee \varphi_4) \wedge (\varphi_2 \vee \varphi_3 \vee \varphi_4) \wedge (\overline{\varphi_1} \vee \varphi_2 \vee \overline{\varphi_3})$$

"Pick a random variable in the first UNSAT clause, and flip it"

Error Induction

WalkSAT

$$\begin{matrix} & \varphi_1 & \varphi_2 & \varphi_3 & \varphi_4 \\ s \mapsto & [\text{true}; & \text{false}; & \text{false}; & \text{false}] \end{matrix}$$

$$F \triangleq (\overline{\varphi_2} \vee \varphi_3 \vee \varphi_4) \wedge (\varphi_2 \vee \varphi_3 \vee \varphi_4) \wedge (\overline{\varphi_1} \vee \varphi_2 \vee \overline{\varphi_3})$$

"Pick a random variable in the first UNSAT clause, and flip it"

Error Induction

WalkSAT

$$s \mapsto \begin{matrix} \varphi_1 & \varphi_2 & \varphi_3 & \varphi_4 \\ [\mathsf{true}; & \mathsf{false}; & \mathsf{false}; & \mathsf{false}] \end{matrix}$$

$$F \triangleq (\overline{\varphi_2} \vee \varphi_3 \vee \varphi_4) \wedge (\varphi_2 \vee \varphi_3 \vee \varphi_4) \wedge (\overline{\varphi_1} \vee \varphi_2 \vee \overline{\varphi_3})$$

"Pick a random variable in the first UNSAT clause, and flip it"

Error Induction

WalkSAT

$$s \mapsto \overset{\varphi_1}{[\textsf{true}};\ \overset{\varphi_2}{\textsf{true}};\ \overset{\varphi_3}{\textsf{false}};\ \overset{\varphi_4}{\textsf{false}]}$$

$$F \triangleq (\overline{\varphi_2} \vee \varphi_3 \vee \varphi_4) \wedge (\varphi_2 \vee \varphi_3 \vee \varphi_4) \wedge (\overline{\varphi_1} \vee \varphi_2 \vee \overline{\varphi_3})$$

"Pick a random variable in the first UNSAT clause, and flip it"

Error Induction

WalkSAT

$$\begin{array}{cccc} \varphi_1 & \varphi_2 & \varphi_3 & \varphi_4 \end{array}$$
$$s \mapsto [\text{true};\ \text{true};\ \text{false};\ \text{false}]$$

$$F \triangleq (\overline{\varphi_2} \vee \varphi_3 \vee \varphi_4) \wedge (\varphi_2 \vee \varphi_3 \vee \varphi_4) \wedge (\overline{\varphi_1} \vee \varphi_2 \vee \overline{\varphi_3})$$

"Pick a random variable in the first UNSAT clause, and flip it"

Error Induction

WalkSAT

$$s \mapsto \overset{\varphi_1}{[\text{true};}\ \overset{\varphi_2}{\text{true};}\ \overset{\varphi_3}{\text{false};}\ \overset{\varphi_4}{\text{true}]}$$

SAT

$$F \triangleq (\overline{\varphi_2} \lor \varphi_3 \lor \varphi_4) \land (\varphi_2 \lor \varphi_3 \lor \varphi_4) \land (\overline{\varphi_1} \lor \varphi_2 \lor \overline{\varphi_3})$$

"Pick a random variable in the first UNSAT clause, and flip it"

Error Induction

WalkSAT

$$s \mapsto [\overset{\varphi_1}{\text{true}};\ \overset{\varphi_2}{\text{true}};\ \overset{\varphi_3}{\text{false}};\ \overset{\varphi_4}{\text{true}}]$$

SAT

$$F \triangleq (\overline{\varphi_2} \vee \varphi_3 \vee \varphi_4) \wedge (\varphi_2 \vee \varphi_3 \vee \varphi_4) \wedge (\overline{\varphi_1} \vee \varphi_2 \vee \overline{\varphi_3})$$

If F is solvable, WalkSAT finds a solution with probability 1.

"Pick a random variable in the first UNSAT clause, and flip it"

Error Induction

WalkSAT

Let s' be a solution to F

s the current assignment

$0 < \epsilon_3$

Error Induction

WalkSAT

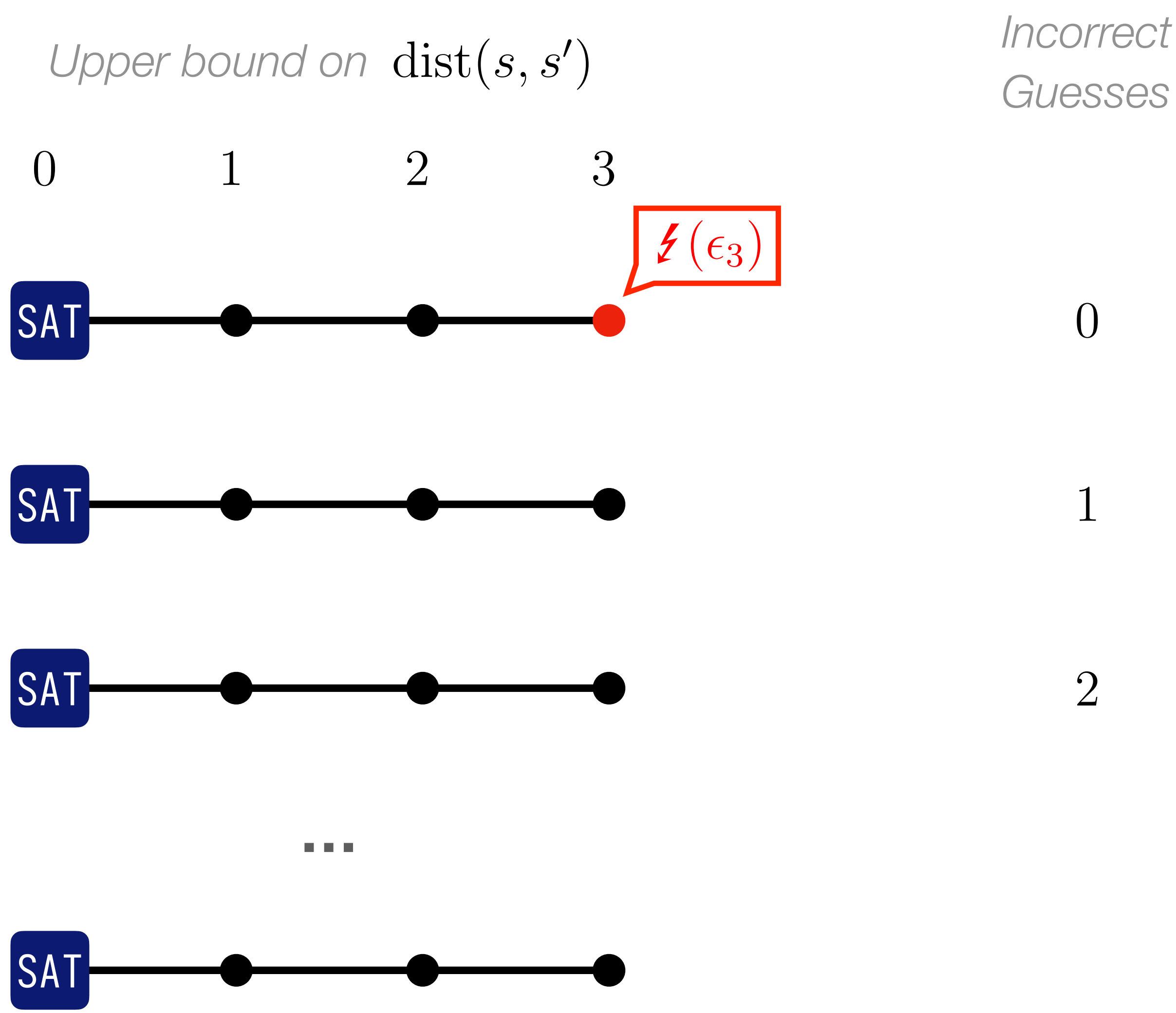
Let s' be a solution to F

s the current assignment

$0 < \epsilon_3$

Flip variable in UNSAT clause:

-
-
-



Error Induction

WalkSAT

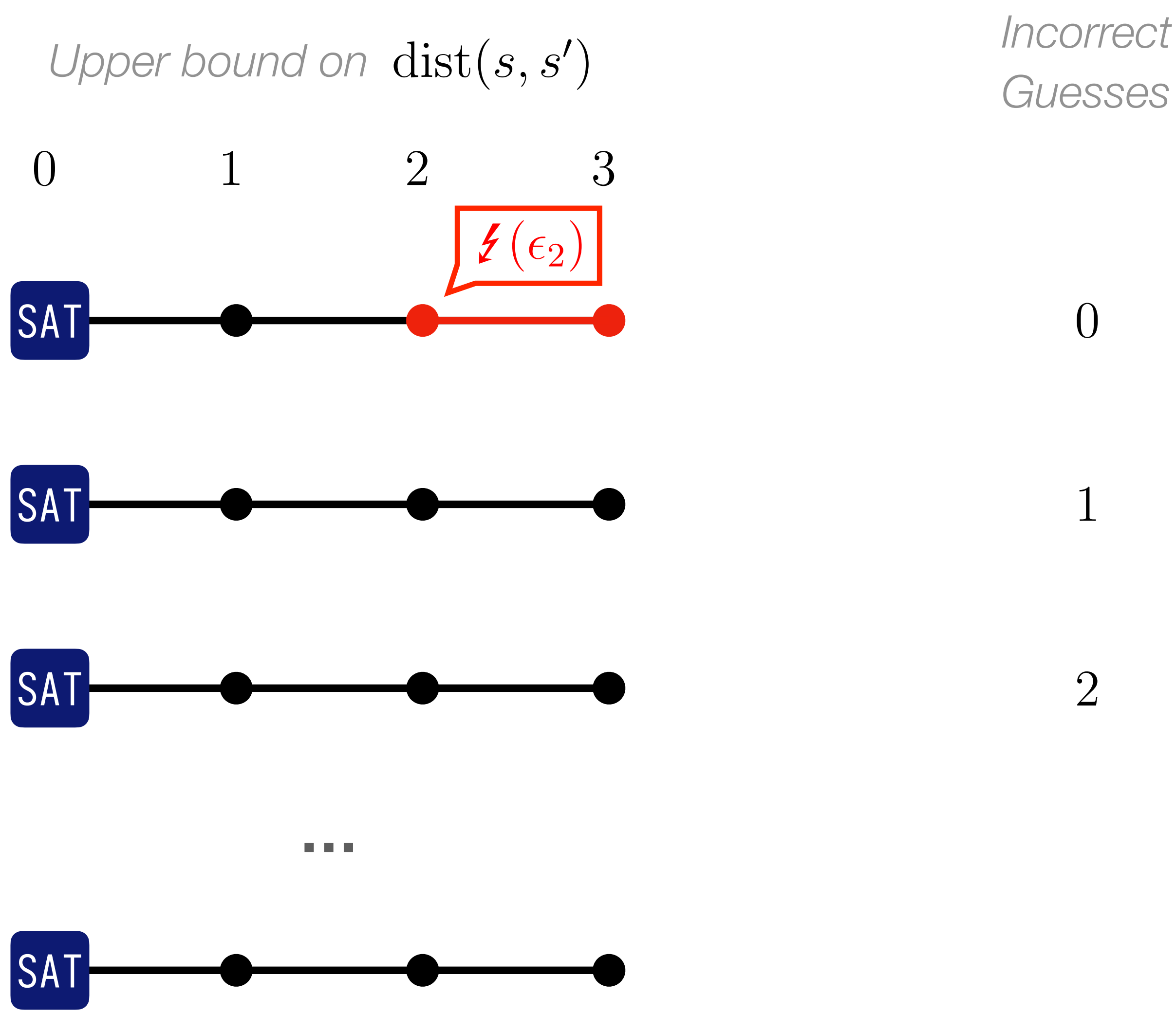
Let s' **be a solution to** F

s **the current assignment**

$0 < \epsilon_3$

Flip variable in UNSAT clause:

- **Reduce** $\mathrm{dist}(s, s')$
-
-



Error Induction

WalkSAT

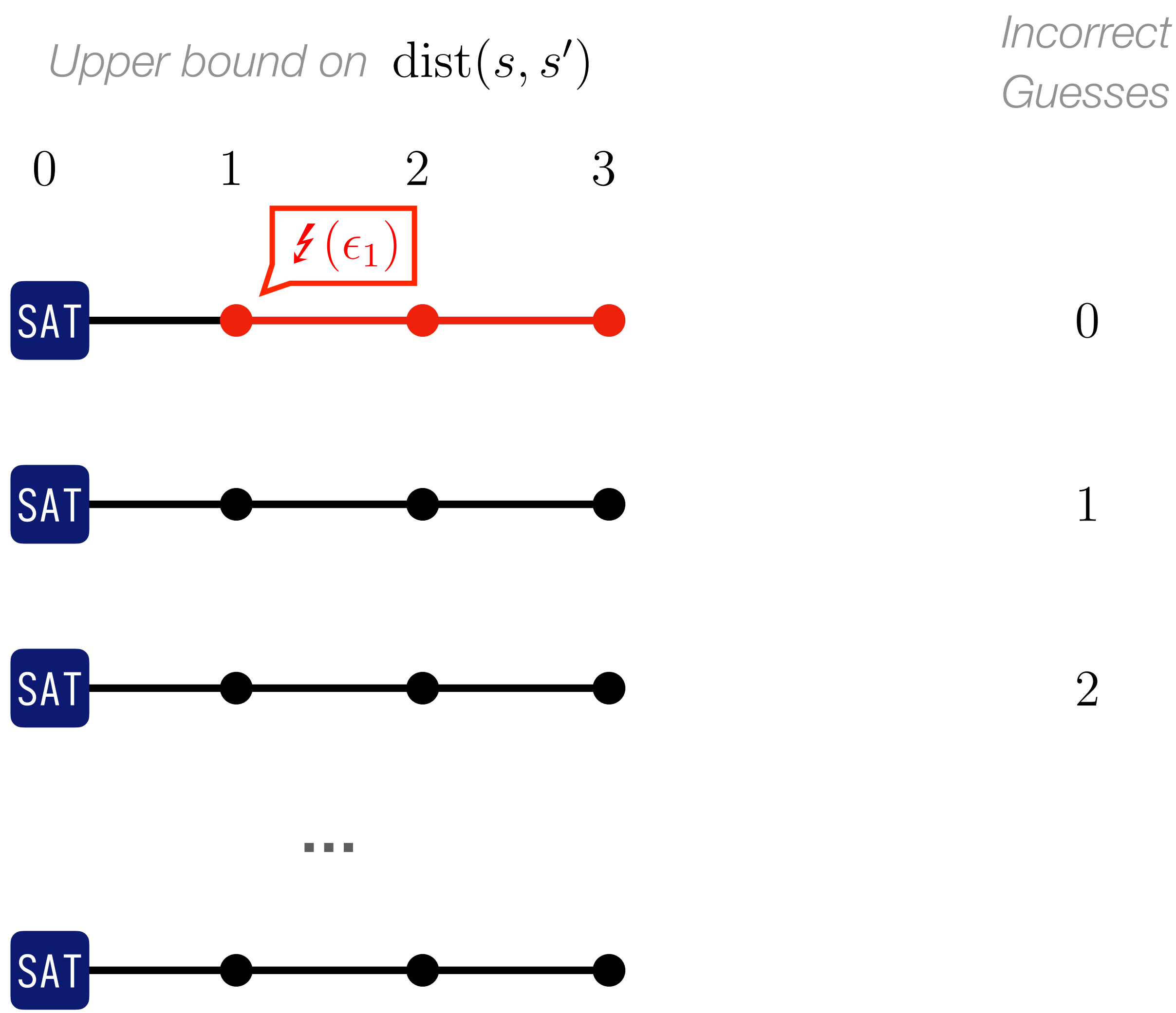
Let s' **be a solution to** F

s **the current assignment**

$0 < \epsilon_3$

Flip variable in UNSAT clause:

- **Reduce** $\mathrm{dist}(s, s')$
-
-



Error Induction

WalkSAT

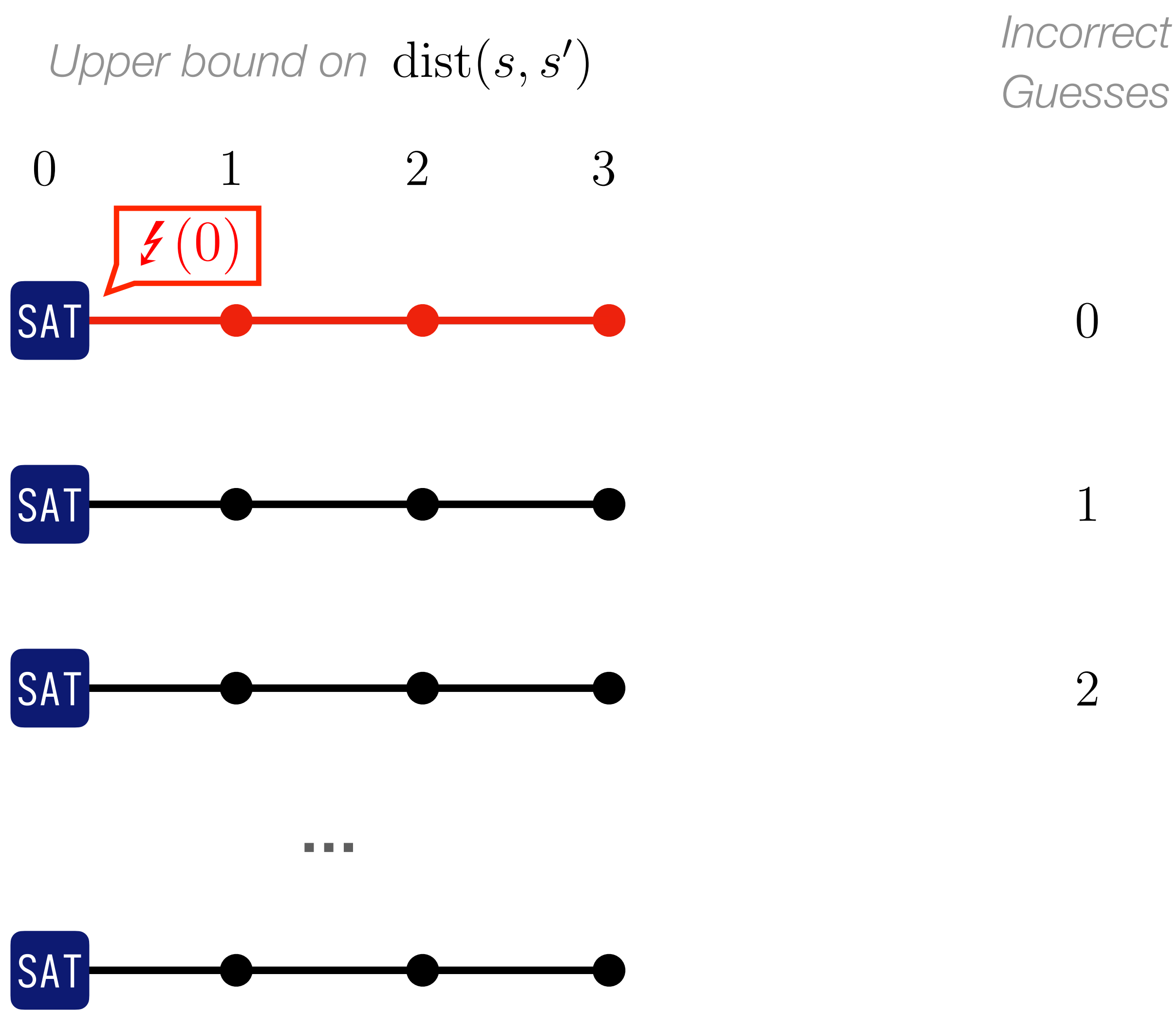
Let s' **be a solution to** F

s **the current assignment**

$0 < \epsilon_3$

Flip variable in UNSAT clause:

- **Reduce** $\text{dist}(s, s')$
-
-



Error Induction

WalkSAT

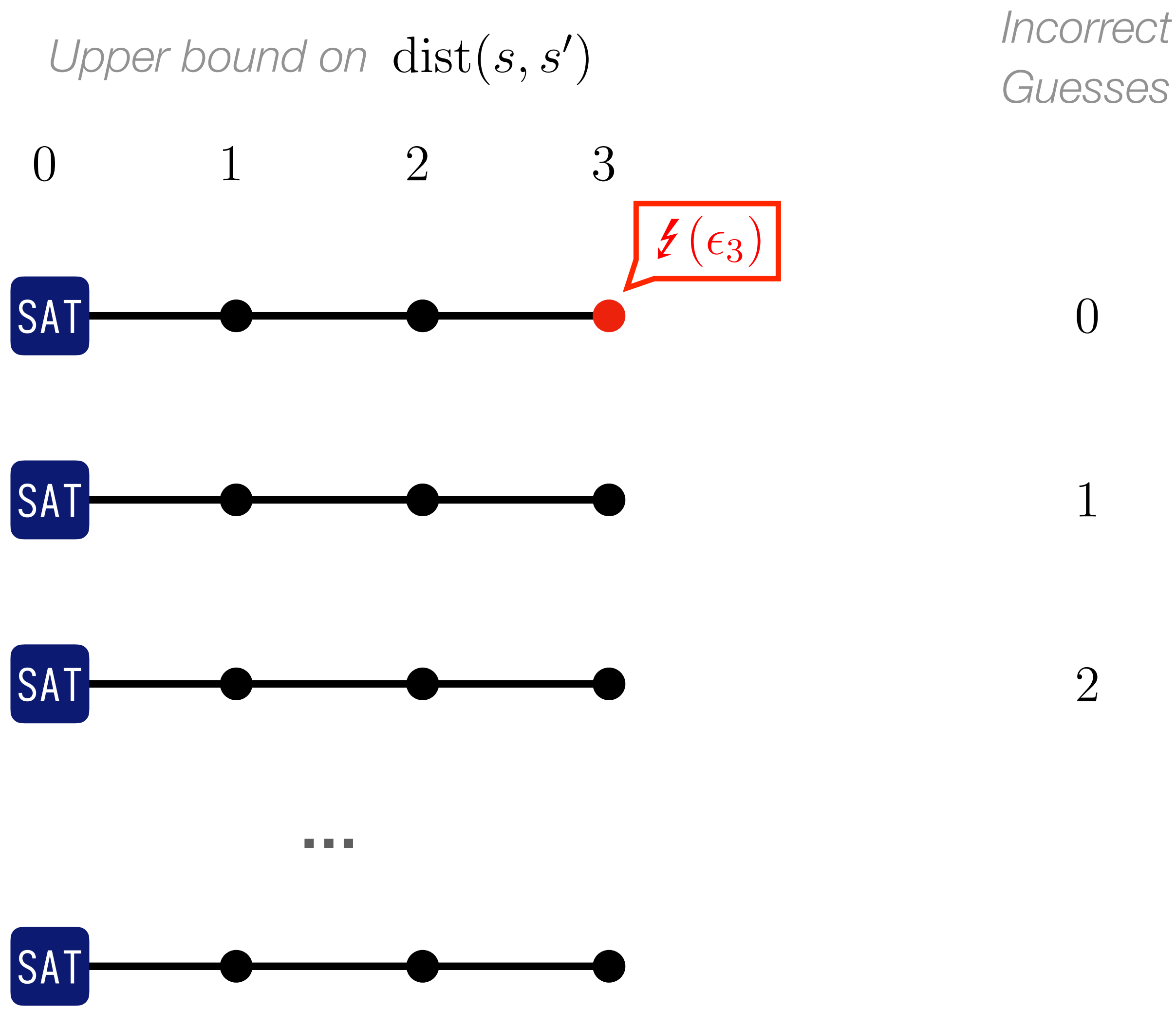
Let s' **be a solution to** F

s **the current assignment**

$0 < \epsilon_3$

Flip variable in UNSAT clause:

- **Reduce** $\text{dist}(s, s')$
-
-



Error Induction

WalkSAT

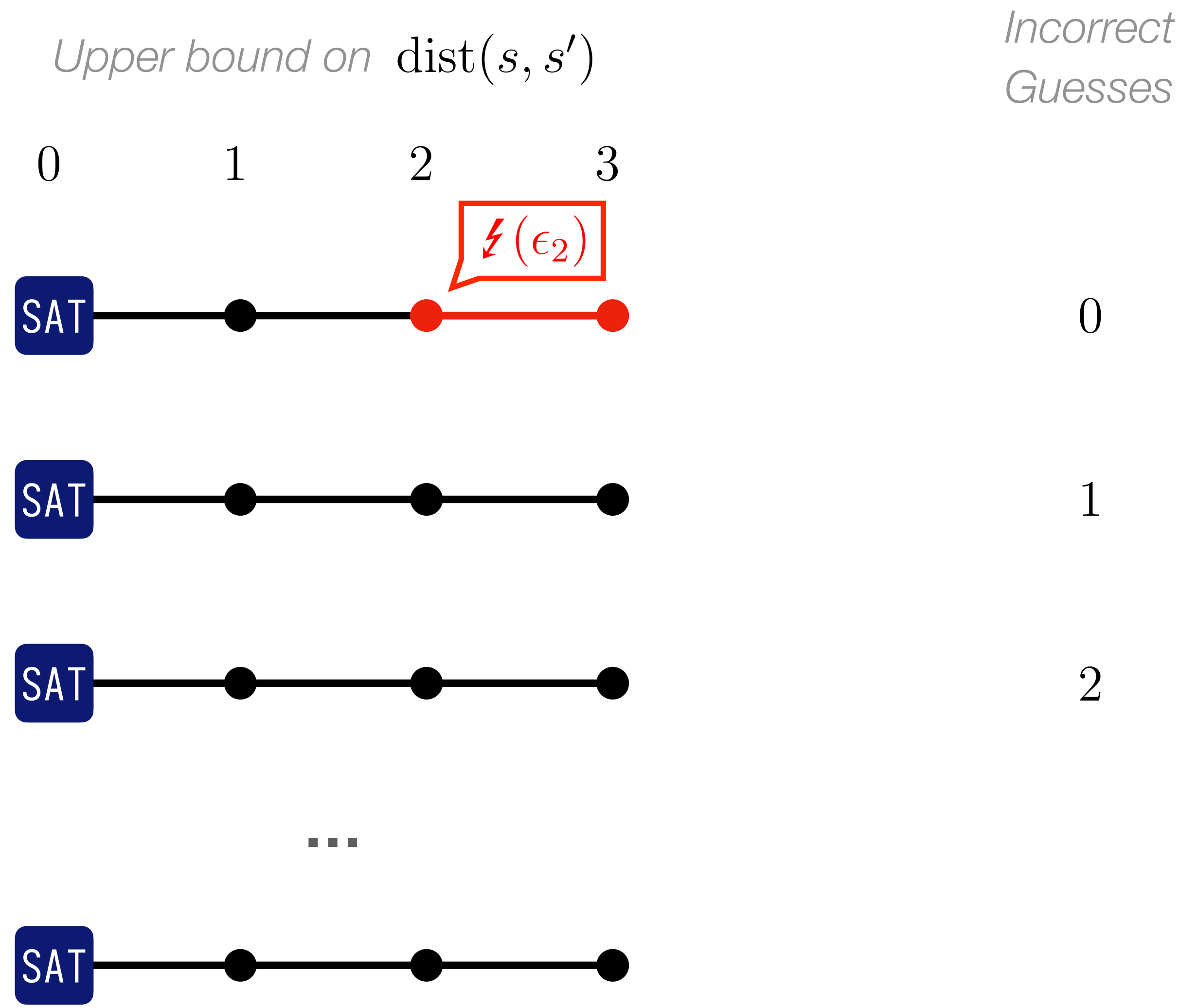
Let s' **be a solution to** F

s **the current assignment**

$0 < \epsilon_3$

Flip variable in UNSAT clause:

- **Reduce** $\text{dist}(s, s')$
-
-



Error Induction

WalkSAT

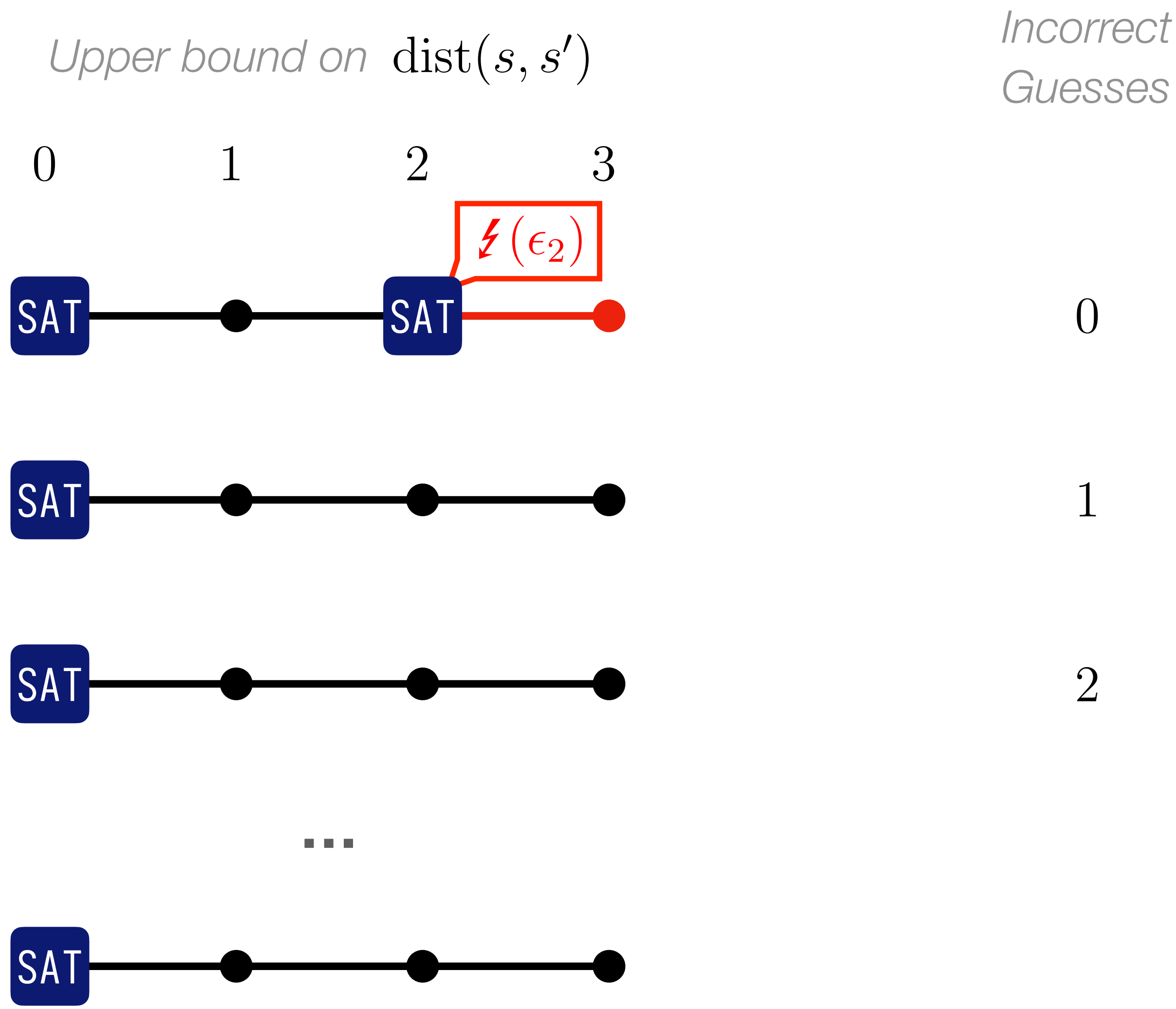
Let s' **be a solution to** F

s **the current assignment**

$0 < \epsilon_3$

Flip variable in UNSAT clause:

- **Reduce** $\text{dist}(s, s')$
- **Lucky SAT**
-



Error Induction

WalkSAT

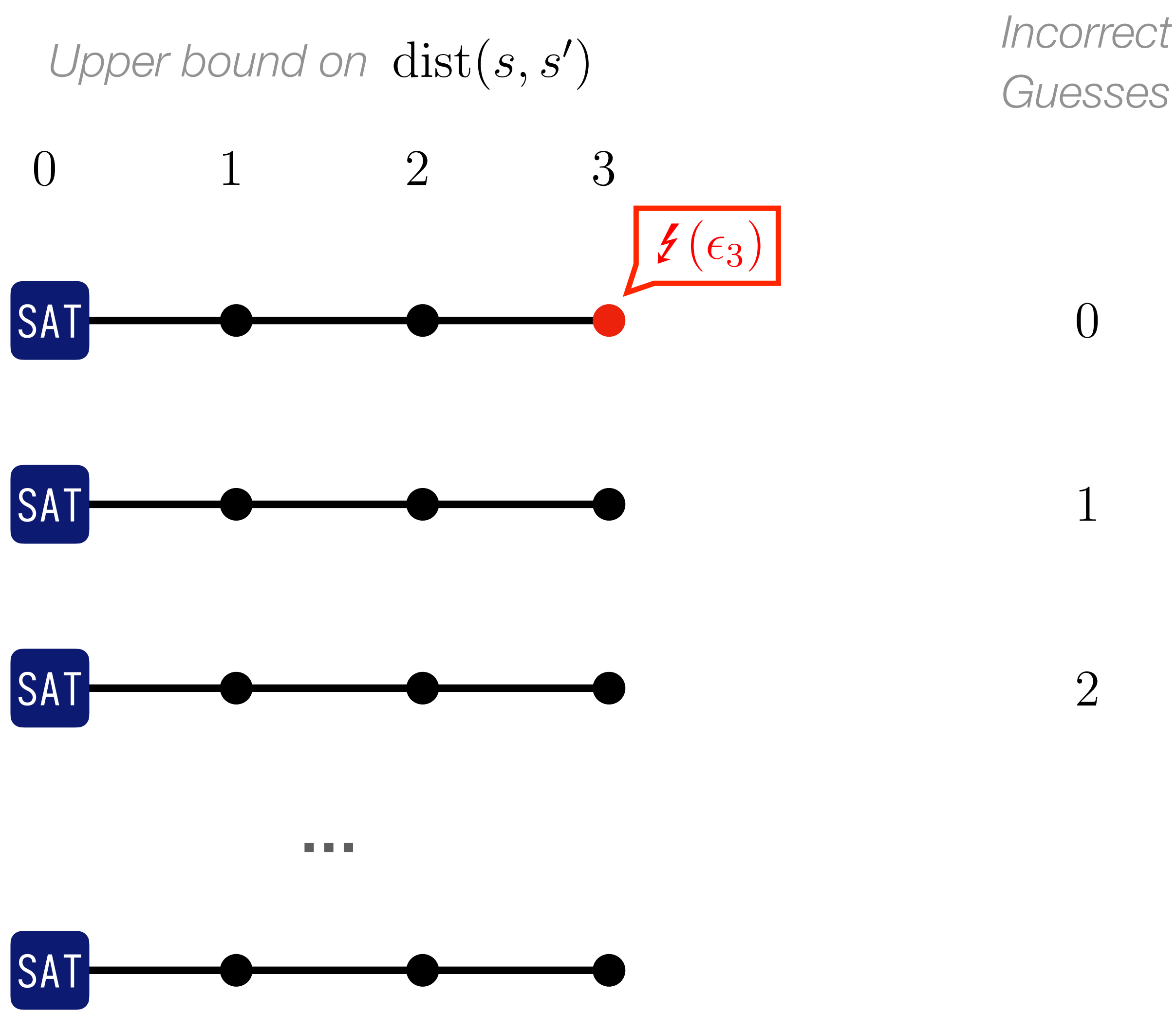
Let s' **be a solution to** F

s **the current assignment**

$0 < \epsilon_3$

Flip variable in UNSAT clause:

- **Reduce** $\mathrm{dist}(s, s')$
- **Lucky SAT**
-



Error Induction

WalkSAT

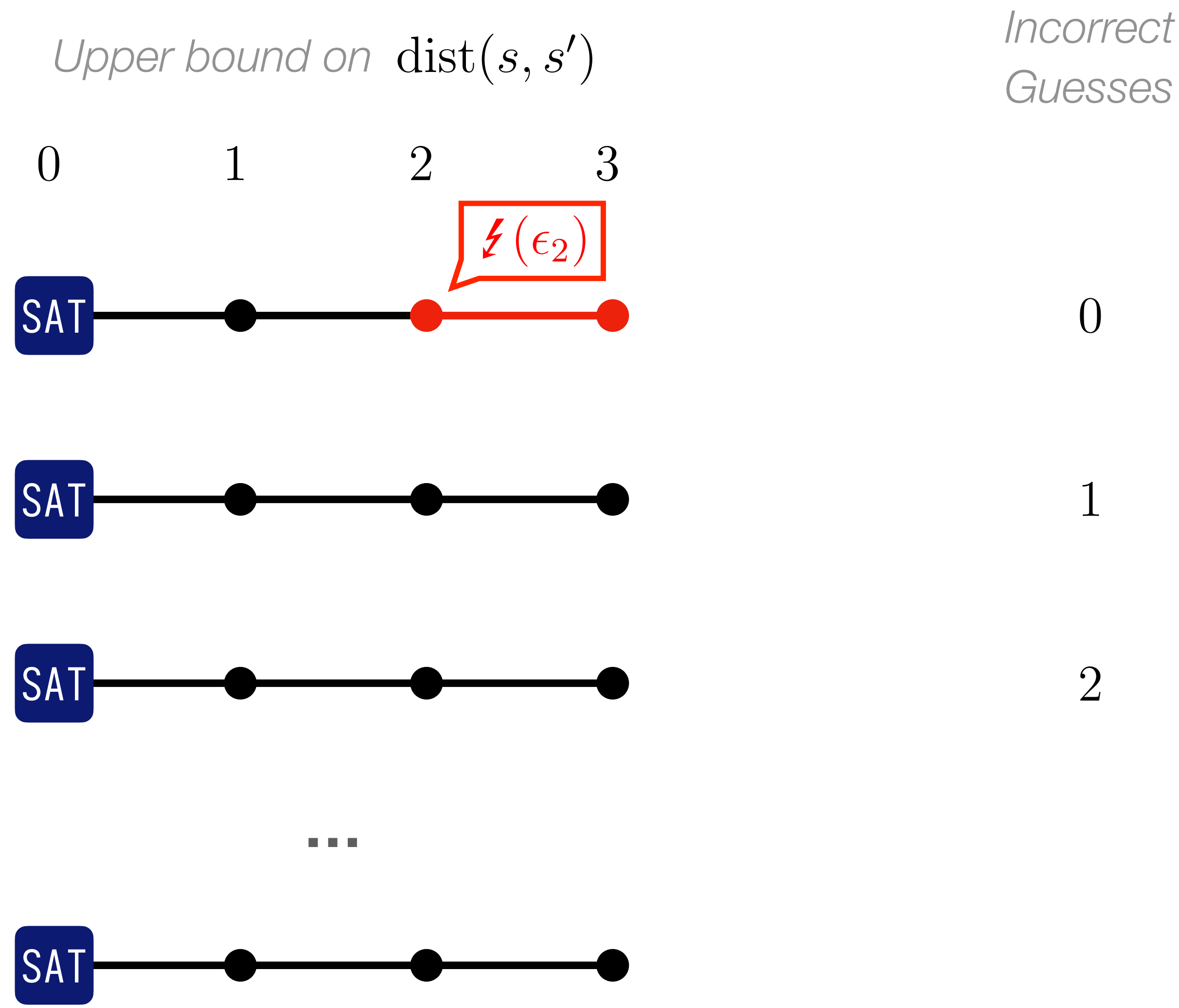
Let s' **be a solution to** F

s **the current assignment**

$0 < \epsilon_3$

Flip variable in UNSAT clause:

- **Reduce** $\text{dist}(s, s')$
- **Lucky SAT**
-



Error Induction

WalkSAT

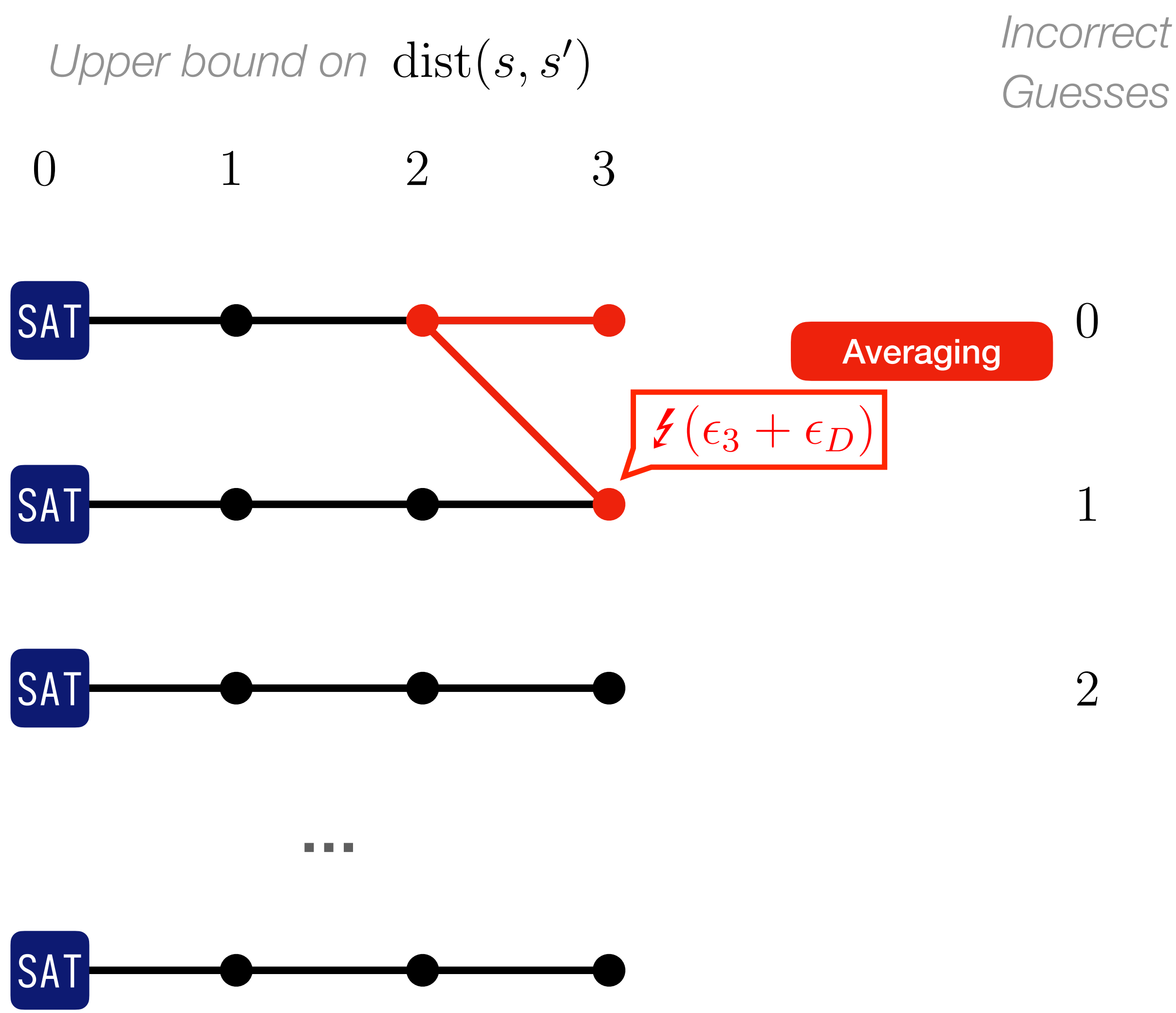
Let s' **be a solution to** F

s **the current assignment**

$0 < \epsilon_3$

Flip variable in UNSAT clause:

- **Reduce** $\text{dist}(s, s')$
- **Lucky SAT**
- **Increase** $\text{dist}(s, s')$



Error Induction

WalkSAT

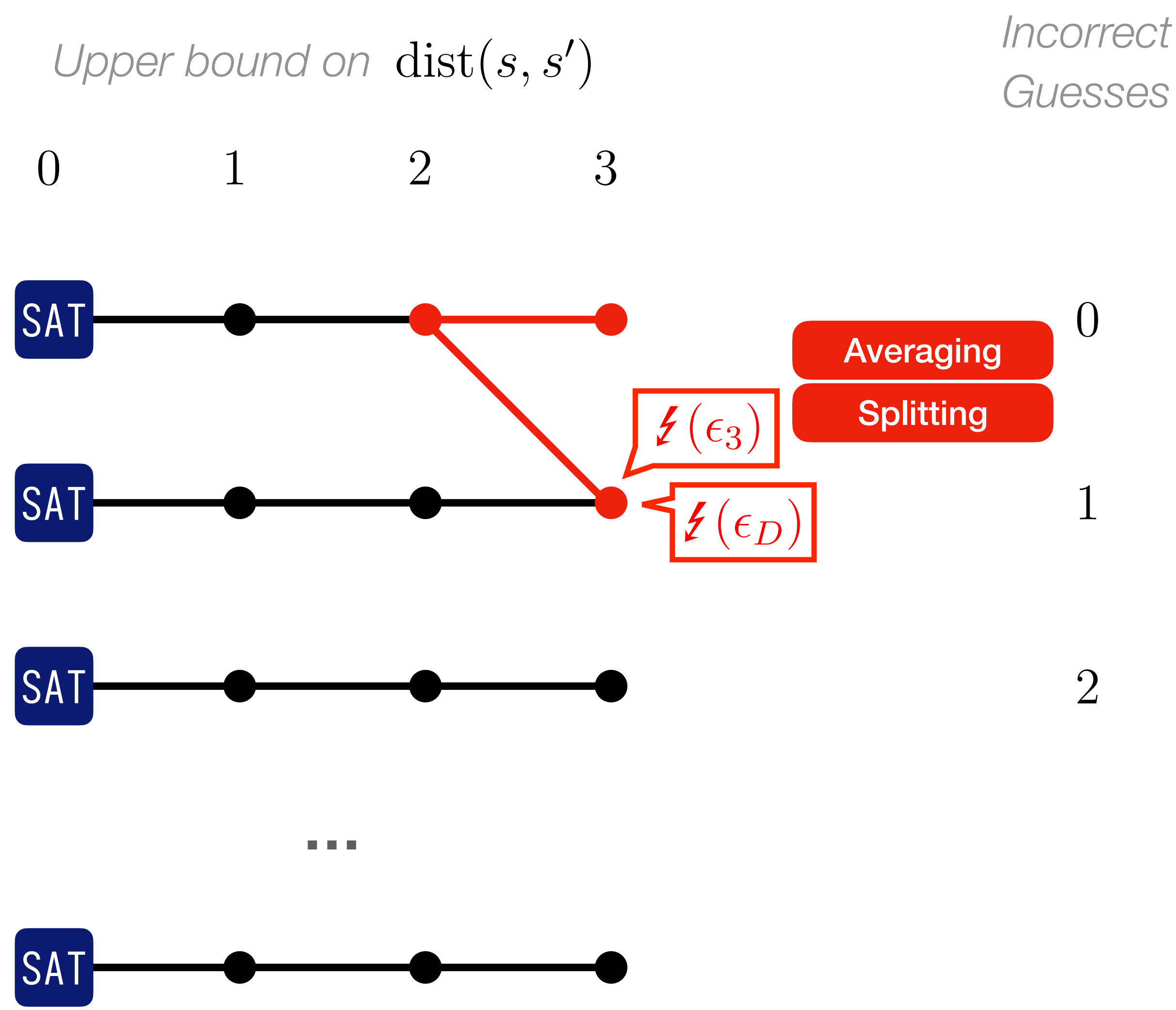
Let s' **be a solution to** F

s **the current assignment**

$0 < \epsilon_3$

Flip variable in UNSAT clause:

- **Reduce** $\mathrm{dist}(s, s')$
- **Lucky SAT**
- **Increase** $\mathrm{dist}(s, s')$



Error Induction

WalkSAT

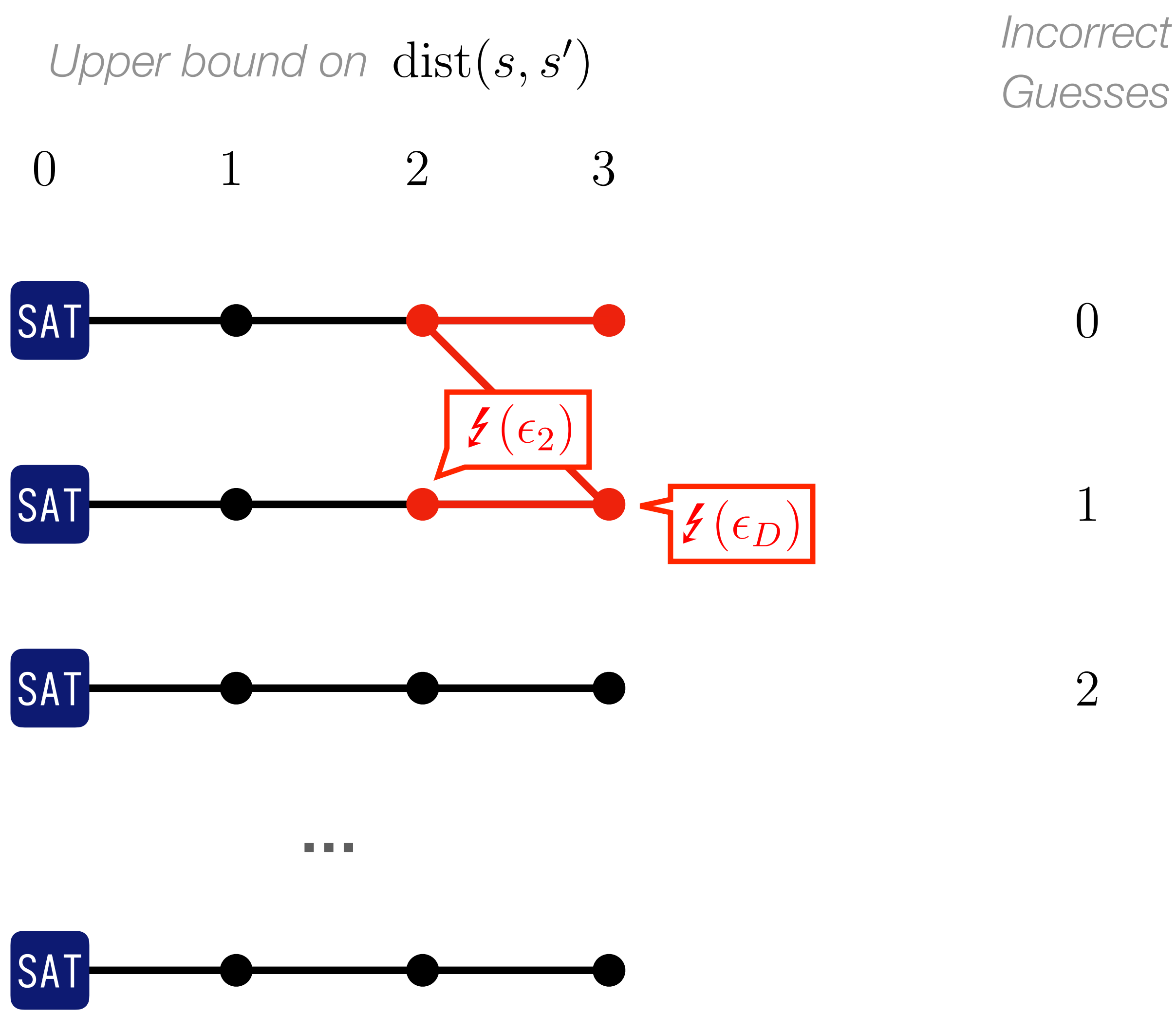
Let s' be a solution to F

s the current assignment

$0 < \epsilon_3$

Flip variable in UNSAT clause:

- **Reduce** $\text{dist}(s, s')$
- **Lucky SAT**
- **Increase** $\text{dist}(s, s')$



Error Induction

WalkSAT

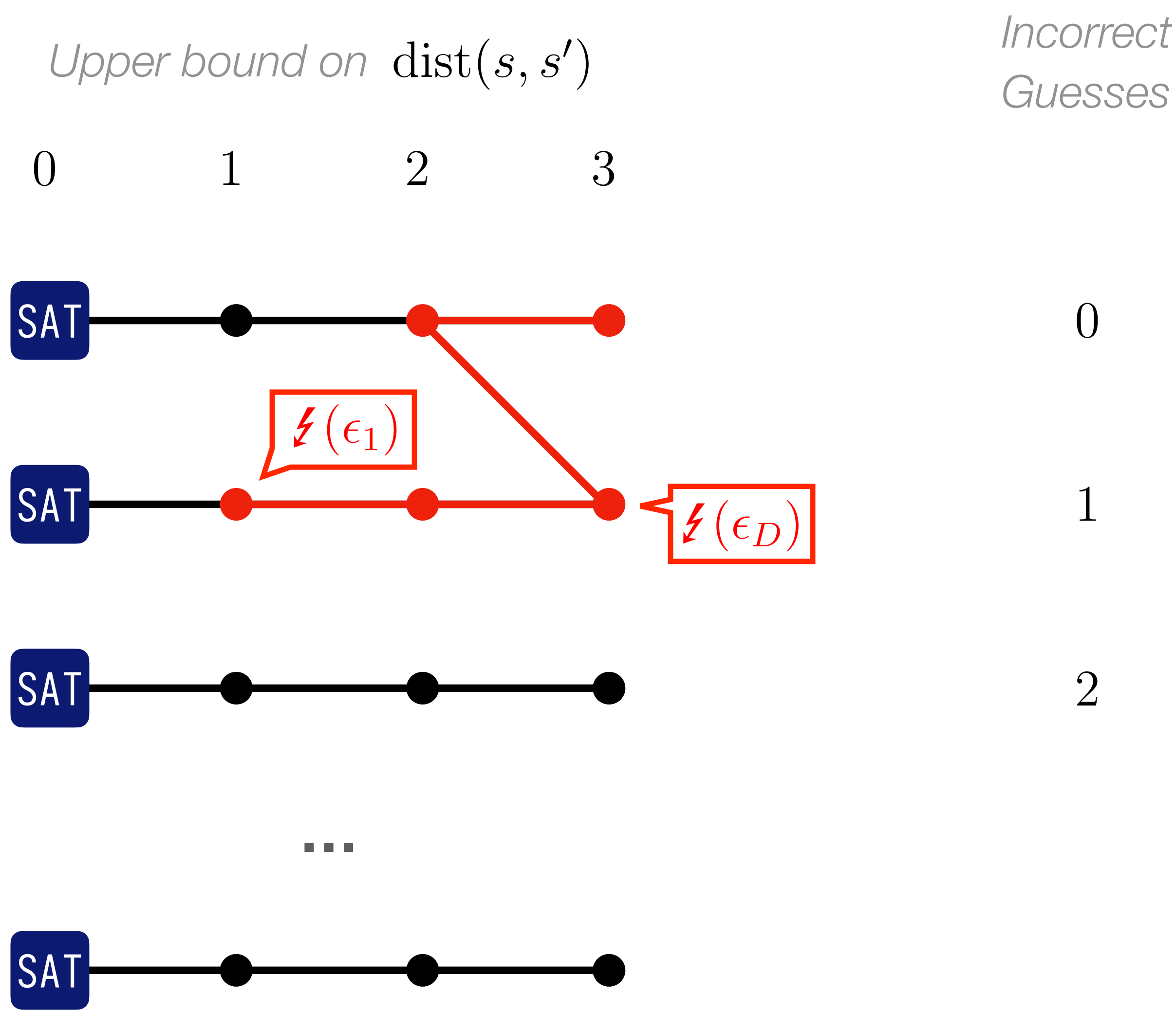
Let s' **be a solution to** F

s **the current assignment**

$0 < \epsilon_3$

Flip variable in UNSAT clause:

- **Reduce** $\text{dist}(s, s')$
- **Lucky SAT**
- **Increase** $\text{dist}(s, s')$



Error Induction

WalkSAT

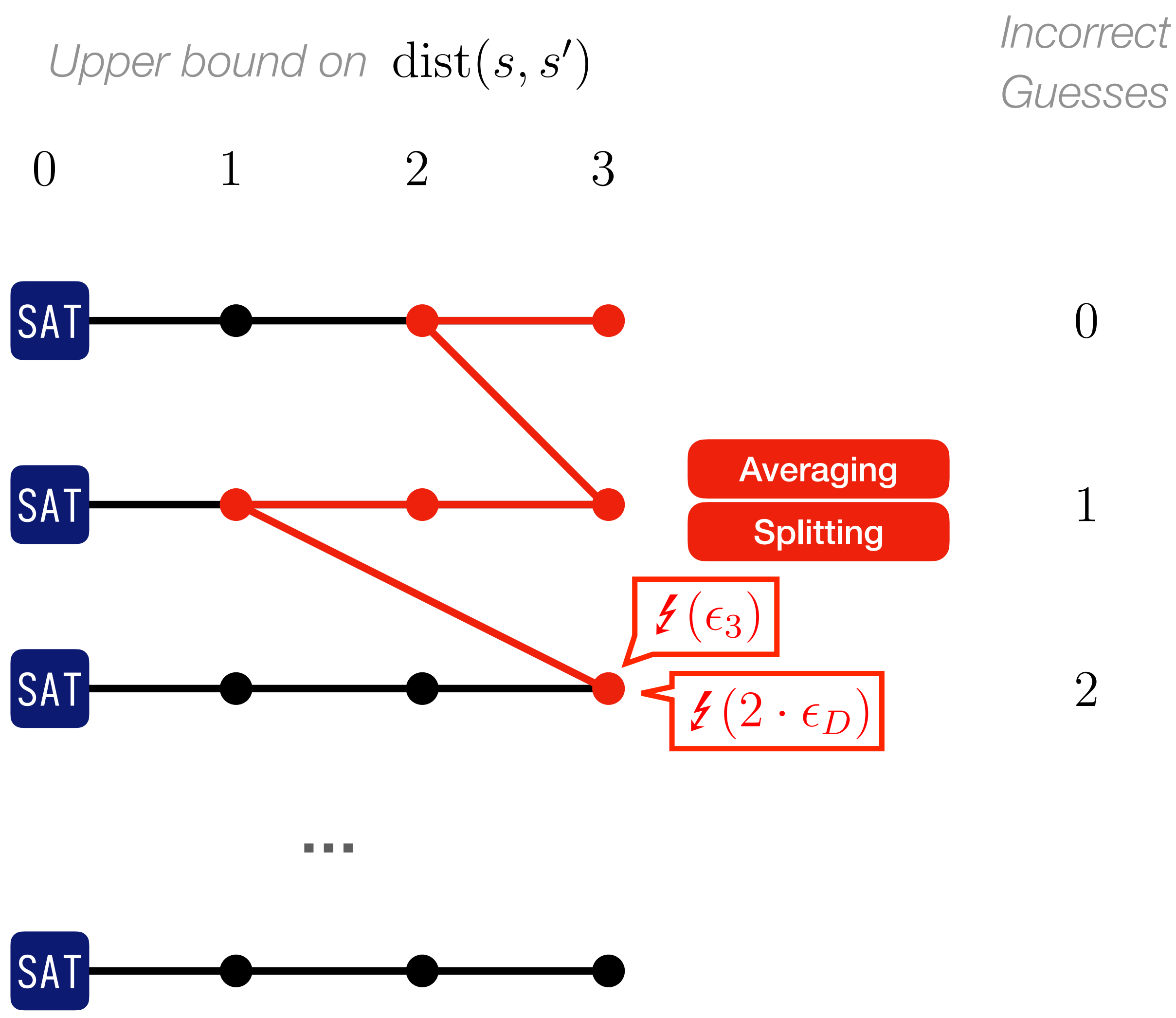
Let s' **be a solution to** F

s **the current assignment**

$0 < \epsilon_3$

Flip variable in UNSAT clause:

- **Reduce** $\mathrm{dist}(s, s')$
- **Lucky SAT**
- **Increase** $\mathrm{dist}(s, s')$



Error Induction

WalkSAT

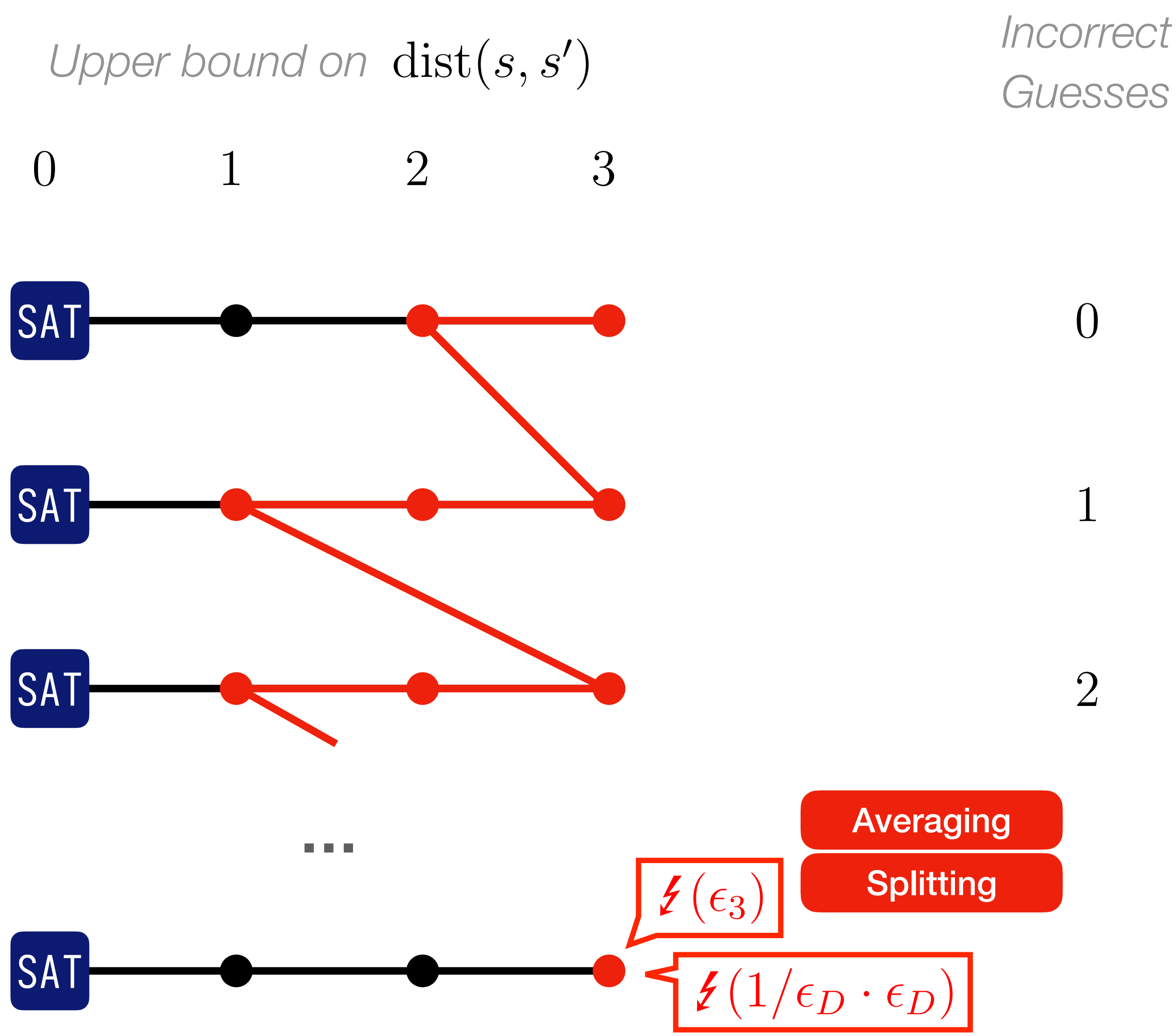
Let s' **be a solution to** F

s **the current assignment**

$0 < \epsilon_3$

Flip variable in UNSAT clause:

- **Reduce** $\mathrm{dist}(s, s')$
- **Lucky SAT**
- **Increase** $\mathrm{dist}(s, s')$



Error Induction

WalkSAT

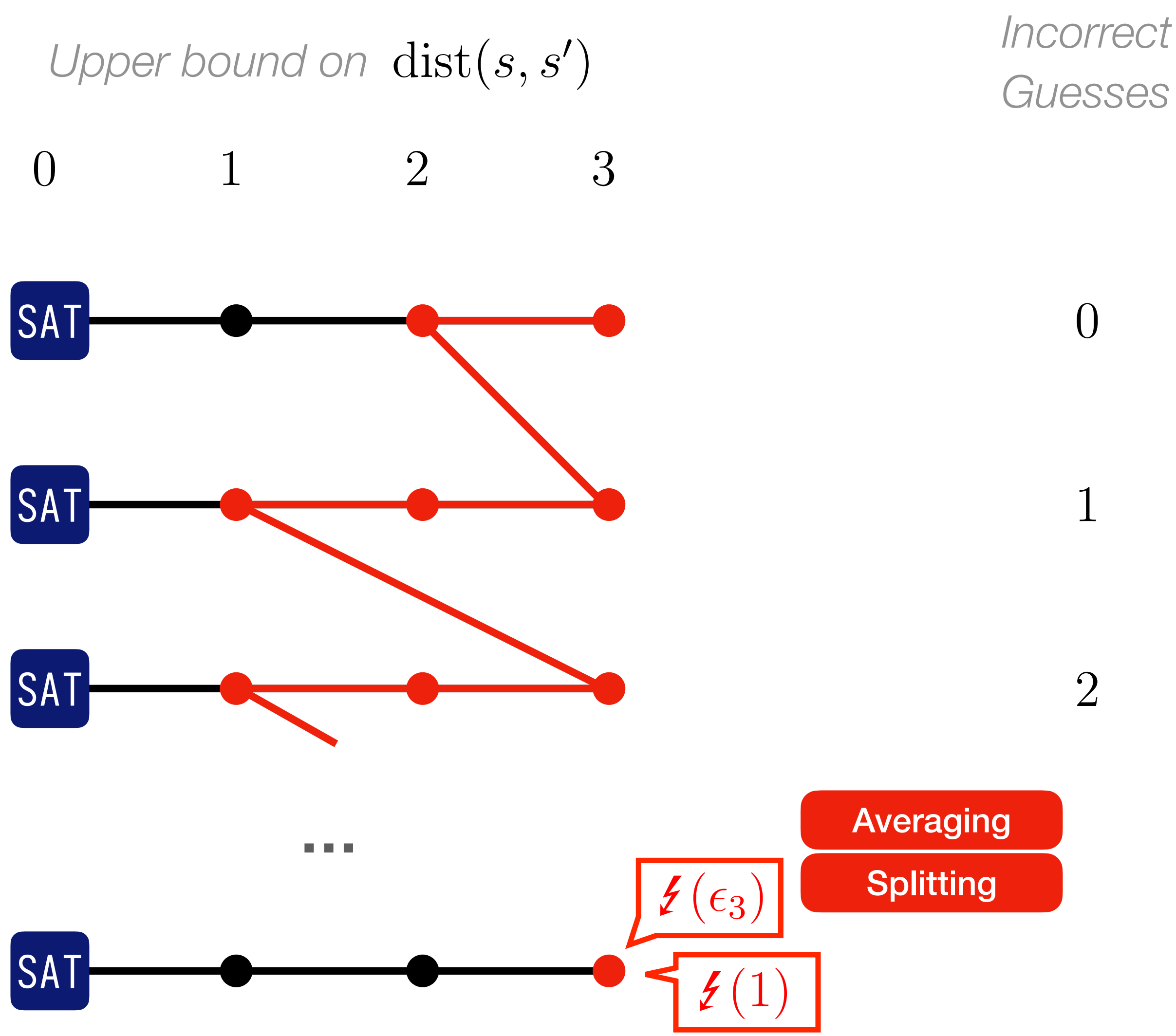
Let s' be a solution to F

s the current assignment

$0 < \epsilon_3$

Flip variable in UNSAT clause:

- **Reduce** $\mathrm{dist}(s, s')$
- **Lucky SAT**
- **Increase** $\mathrm{dist}(s, s')$



Error Induction

WalkSAT

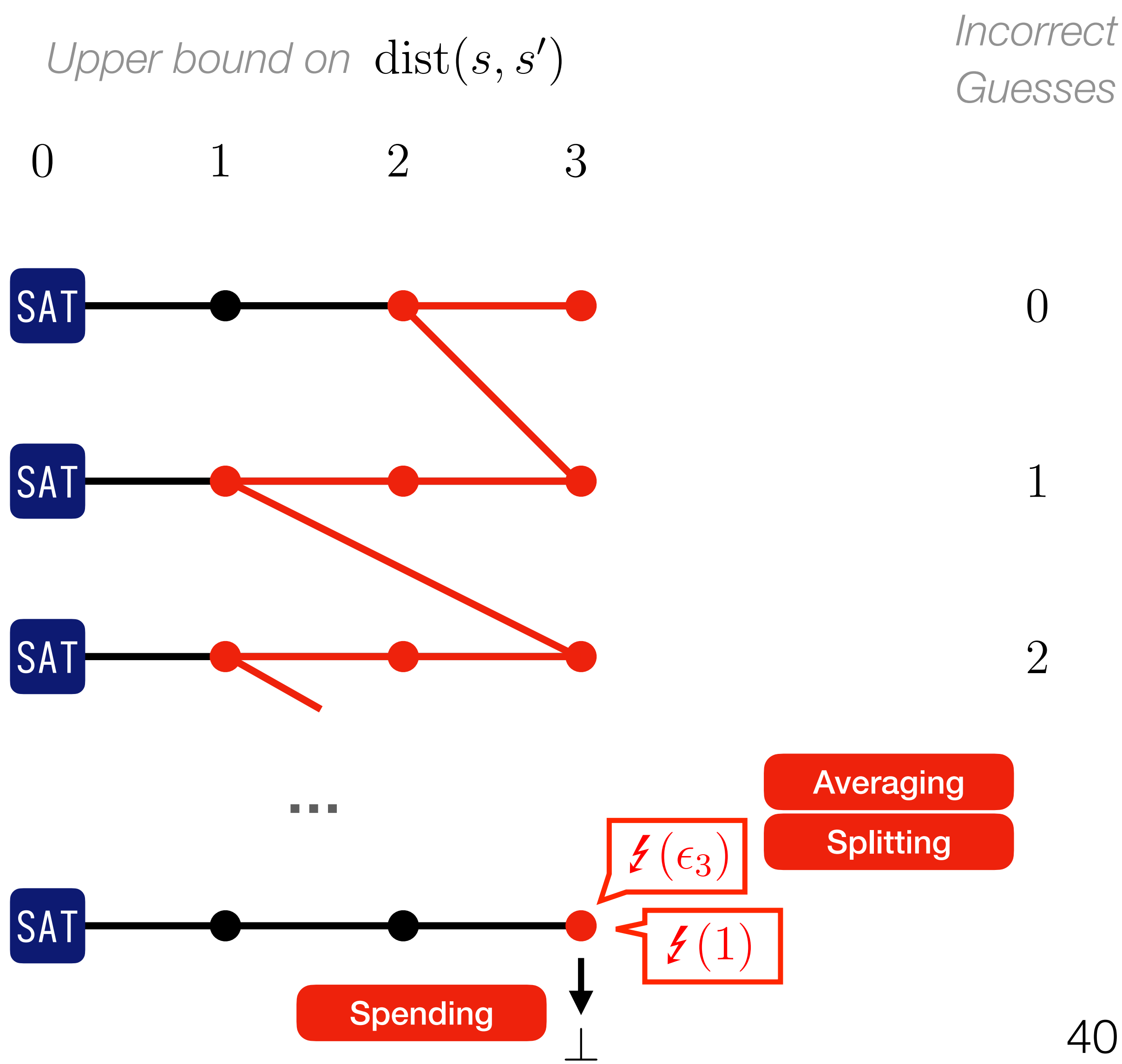
Let s' **be a solution to** F

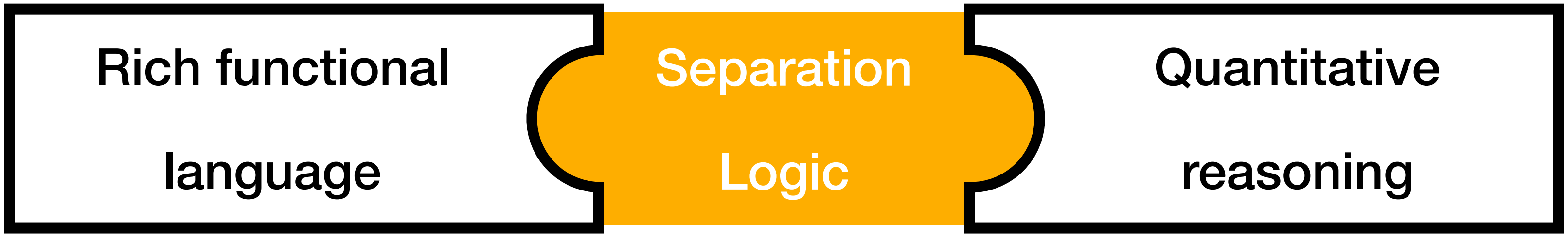
s **the current assignment**

$0 < \epsilon_3$

Flip variable in UNSAT clause:

- **Reduce** $\mathrm{dist}(s, s')$
- **Lucky SAT**
- **Increase** $\mathrm{dist}(s, s')$





Expected values as state

Challenge 2.

Almost-Sure Termination

- Error credits in a total logic
- Error induction + continuity for recursive programs
- Handles higher-order, stateful programs

Challenge 3.

Expected Cost Bounds

Expected Time Bounds

Expected Time Bounds

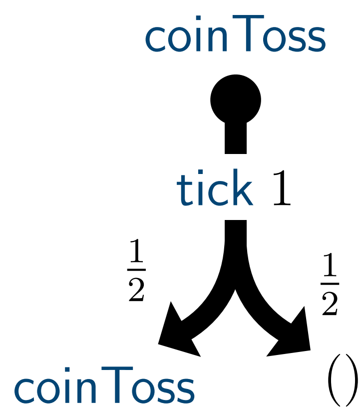
rec coinToss _ =

tick 1;

if flip

then $()$

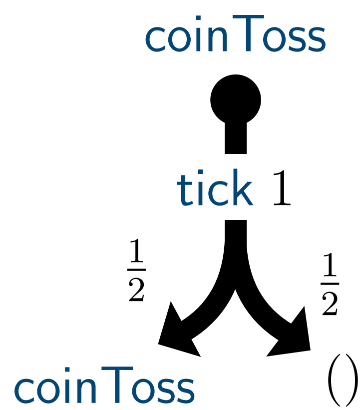
else coinToss $()$



Expected Time Bounds

How many times is $\mathsf{tick}\ 1$ called?

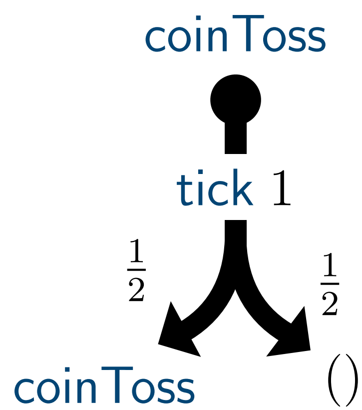
$$\mathsf{rec}\ \mathsf{coinToss}\ \_ =$$
$$\quad \mathsf{tick}\ 1;$$
$$\quad \mathsf{if}\ \mathsf{flip}$$
$$\quad\quad \mathsf{then}\ ()$$
$$\quad\quad \mathsf{else}\ \mathsf{coinToss}\ ()$$



Expected Time Bounds

How many times is $\mathsf{tick}\ 1$ called?

$\mathsf{rec}\ \mathsf{coinToss}\ \_ =$
$\quad \mathsf{tick}\ 1;$
$\quad \mathsf{if}\ \mathsf{flip}$
$\qquad \mathsf{then}\ ()$
$\qquad \mathsf{else}\ \mathsf{coinToss}\ ()$

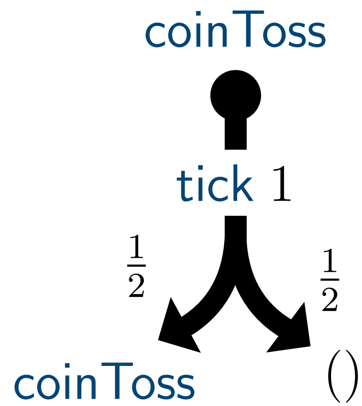


$$\mathbb{E}[T] = 1 + (1/2) \cdot 0 + (1/2) \cdot \mathbb{E}[T]$$

Expected Time Bounds

How many times is tick 1 called?

```
rec coinToss _ =
    tick 1;
    if flip
        then ()
        else coinToss ()
```



$$\mathbb{E}[T] = 1 + (1/2)\cdot 0 + (1/2)\cdot \mathbb{E}[T]$$

$$\boxed{\mathbb{E}[T] = 2}$$

Time Credits: ***Deterministic***

Pioneered by Bob Atkey *Amortised Resource Analysis with Separation Logic*

Separation logic plus time credits $\$(x)$

Time Credits: ***Deterministic***

Pioneered by Bob Atkey *Amortised Resource Analysis with Separation Logic*

Separation logic plus time credits $\$(x)$

Soundness: $\vdash \left\{P * \$(x)\right\} e \{Q\} \quad \Rightarrow$ runtime bound of x

Time Credits: ***Deterministic***

Pioneered by Bob Atkey *Amortised Resource Analysis with Separation Logic*

Separation logic plus time credits $\$(x)$

Soundness: $\vdash \{P * \$(x)\}\, e\, \{Q\} \quad \Rightarrow$ runtime bound of x

$\$(x) * \$(y) \dashv\vdash \$(x+y)$

$\vdash \{\$(1)\}\ \textsf{tick}\ 1\ \{\top\}$

$$\frac{\{P\}\, e\, \{Q\}}{\{P * \$(x)\}\, e\, \{Q * \$(x)\}}$$

- *Derived rules for amortization*

Time Credits: ***Deterministic***

Pioneered by Bob Atkey *Amortised Resource Analysis with Separation Logic*

Separation logic plus time credits $\$(x)$

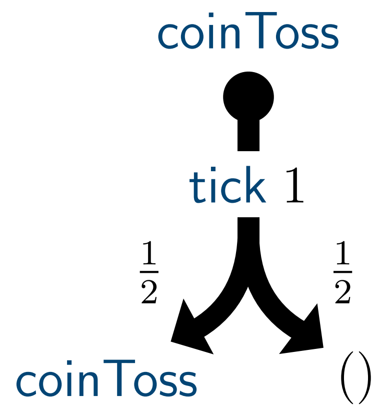
(Some) Subsequent Work

- *Time Credits and Time Receipts in Iris (2019)*
  Mével, Jourdan, and Pottier
- *Thunks and Debits in Separation Logic with Time Credits (2024)*
  Pottier, Guéneau, Jourdan, Mével

Time Credits: *Probabilistic?*

$$\mathbb{E}[T] = 2$$

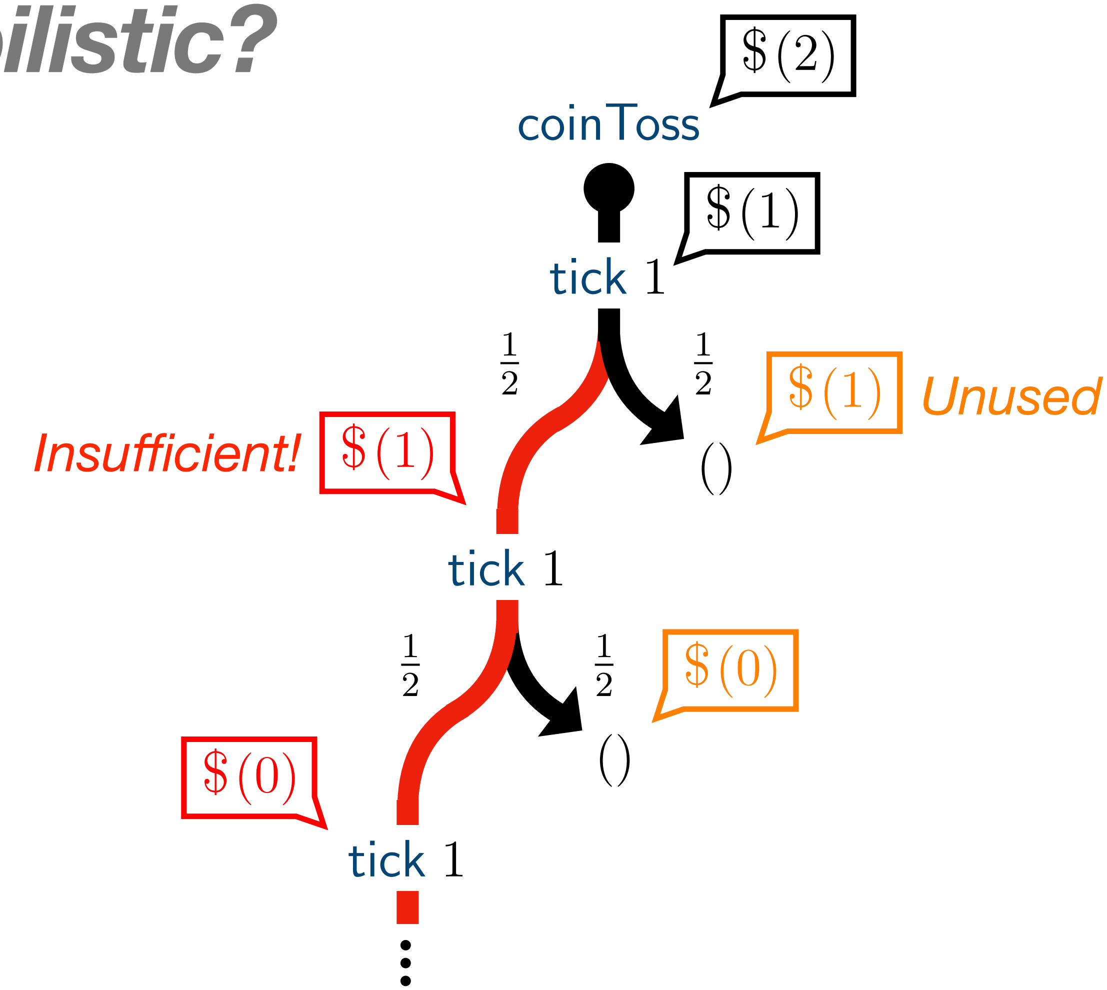
```
rec coinToss _ =
    tick 1;
   if flip
      then ()
      else coinToss ()
```



Time Credits: ***Probabilistic?***

$$\mathbb{E}[T] = 2$$

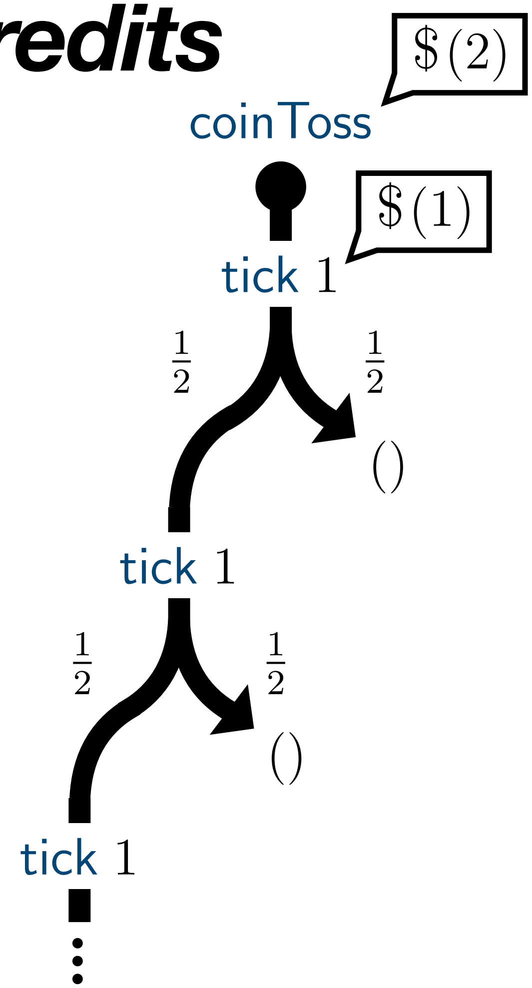
```
rec coinToss _ =
   tick 1;
  if flip
    then ()
    else coinToss ()
```



Expected Cost Credits

$$\mathbb{E}[T] = 2$$

```
rec coinToss _ =
  tick 1;
  if flip
    then ()
    else coinToss ()
```

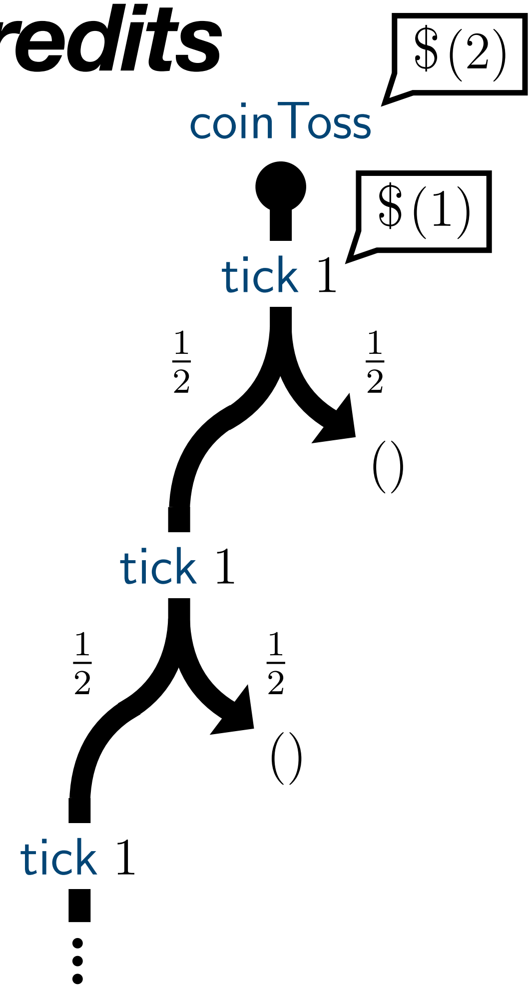


TACHIS Expected Cost Credits

$\mathbb{E}[T] = 2$

```
rec coinToss _ =
   tick 1;
  if flip
    then ()
    else coinToss ()
```

$$\frac{\{\$(2) * P\}\ \mathsf{false}\ \{Q\} \quad \{\$(0) * P\}\ \mathsf{true}\ \{Q\}}{\{\$(1) * P\}\ \mathsf{flip}\ \{Q\}}$$

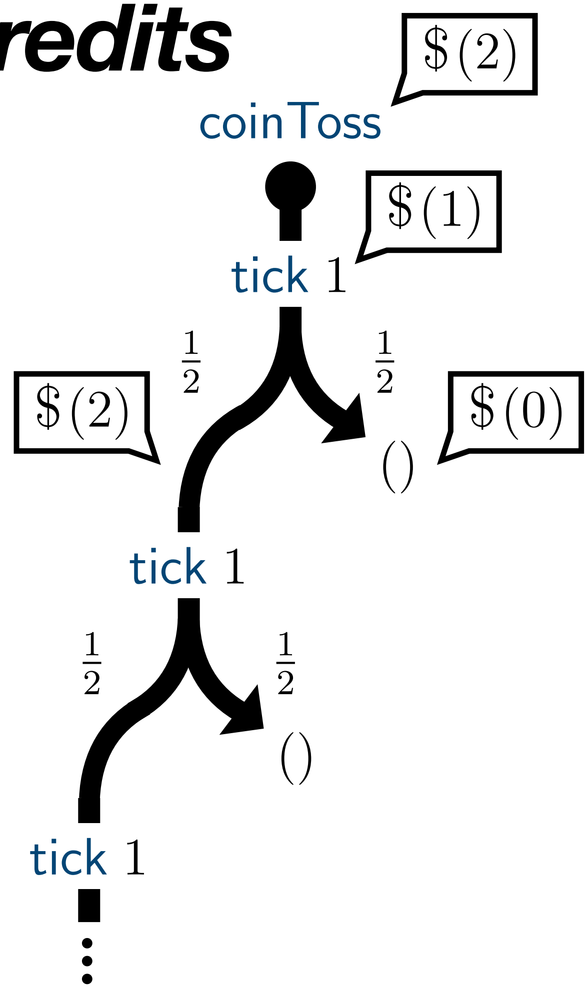


TACHIS Expected Cost Credits

$$\mathbb{E}[T] = 2$$

```
rec coinToss _ =
  tick 1;
  if flip
    then ()
    else coinToss ()
```

$$\frac{\{\$(2) * P\}\ \mathsf{false}\ \{Q\} \quad \{\$(0) * P\}\ \mathsf{true}\ \{Q\}}{\{\$(1) * P\}\ \mathsf{flip}\ \{Q\}}$$

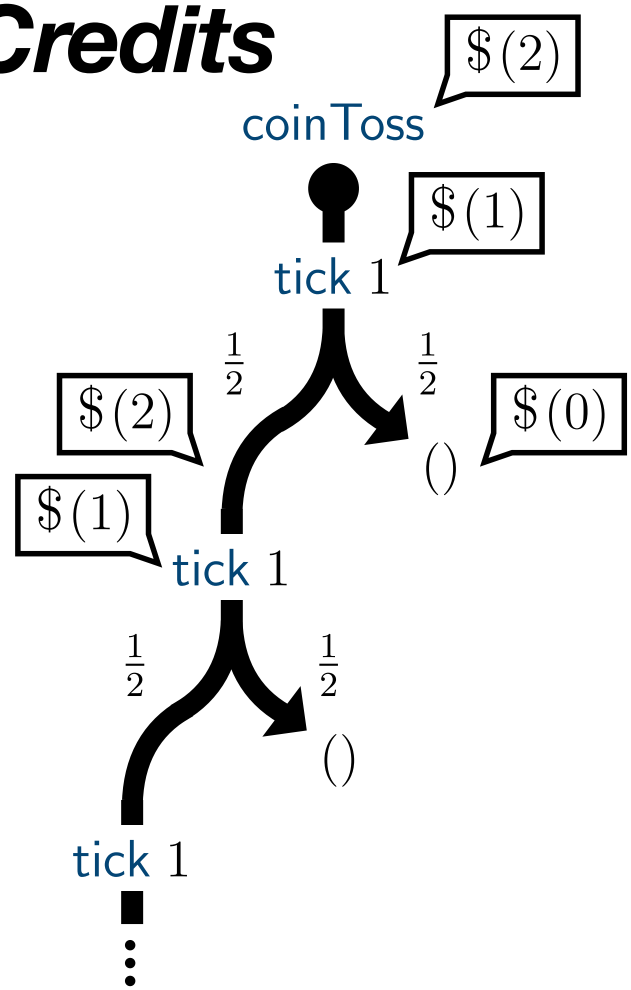


TACHIS Expected Cost Credits

$$\mathbb{E}[T] = 2$$

```
rec coinToss _ =
  tick 1;
  if flip
    then ()
    else coinToss ()
```

$$\frac{\{\$(2) * P\}\ \mathsf{false}\ \{Q\} \quad \{\$(0) * P\}\ \mathsf{true}\ \{Q\}}{\{\$(1) * P\}\ \mathsf{flip}\ \{Q\}}$$

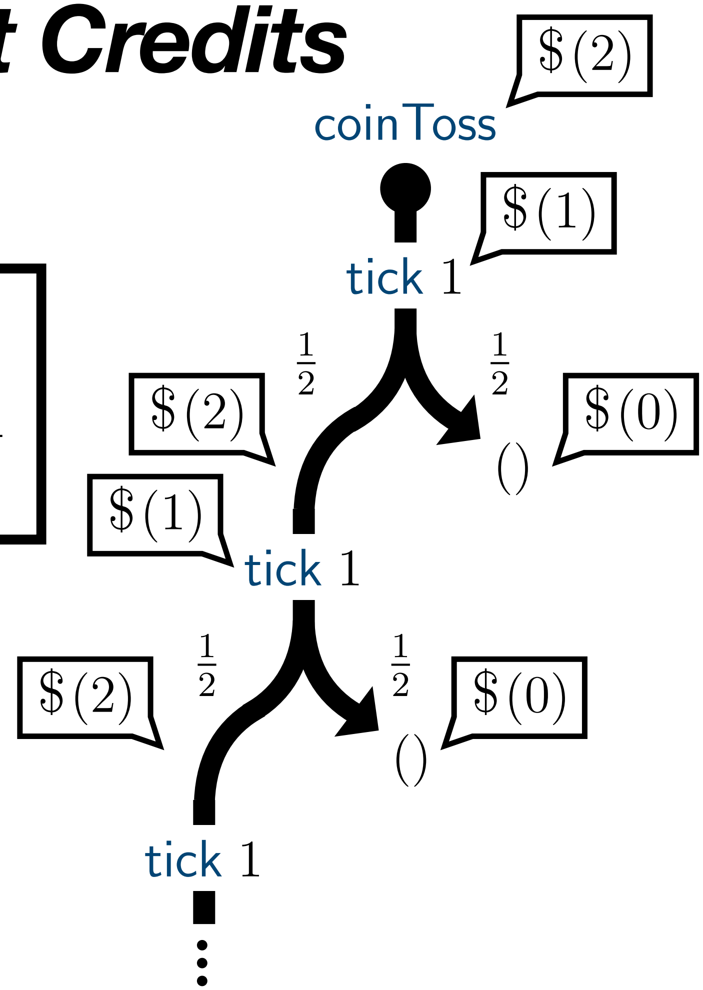


TACHIS Expected Cost Credits

$$\mathbb{E}[T] = 2$$

```
rec coinToss _ =
  tick 1;
  if flip
    then ()
    else coinToss ()
```

$$\frac{\{\$(2) * P\}\ \mathsf{false}\ \{Q\} \quad \{\$(0) * P\}\ \mathsf{true}\ \{Q\}}{\{\$(1) * P\}\ \mathsf{flip}\ \{Q\}}$$

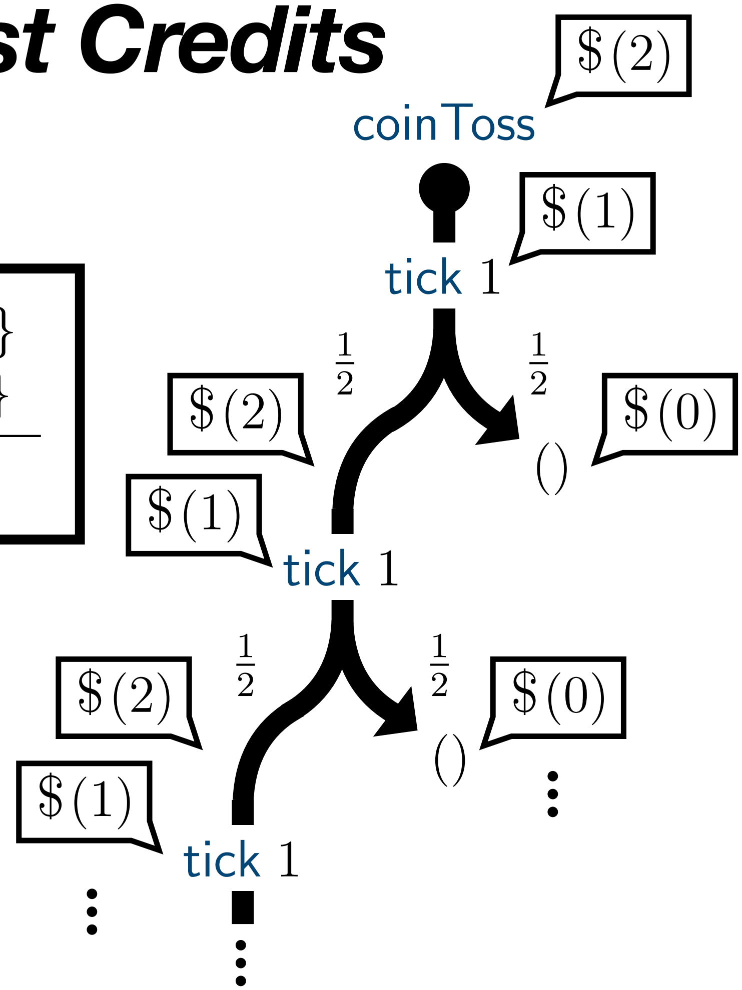


Expected Cost Credits

$$\mathbb{E}[T] = 2$$

```
rec coinToss _ =
   tick 1;
   if flip
     then ()
     else coinToss ()
```

$$\frac{\{\$(2) * P\}\ \mathsf{false}\ \{Q\} \quad \{\$(0) * P\}\ \mathsf{true}\ \{Q\}}{\{\$(1) * P\}\ \mathsf{flip}\ \{Q\}}$$



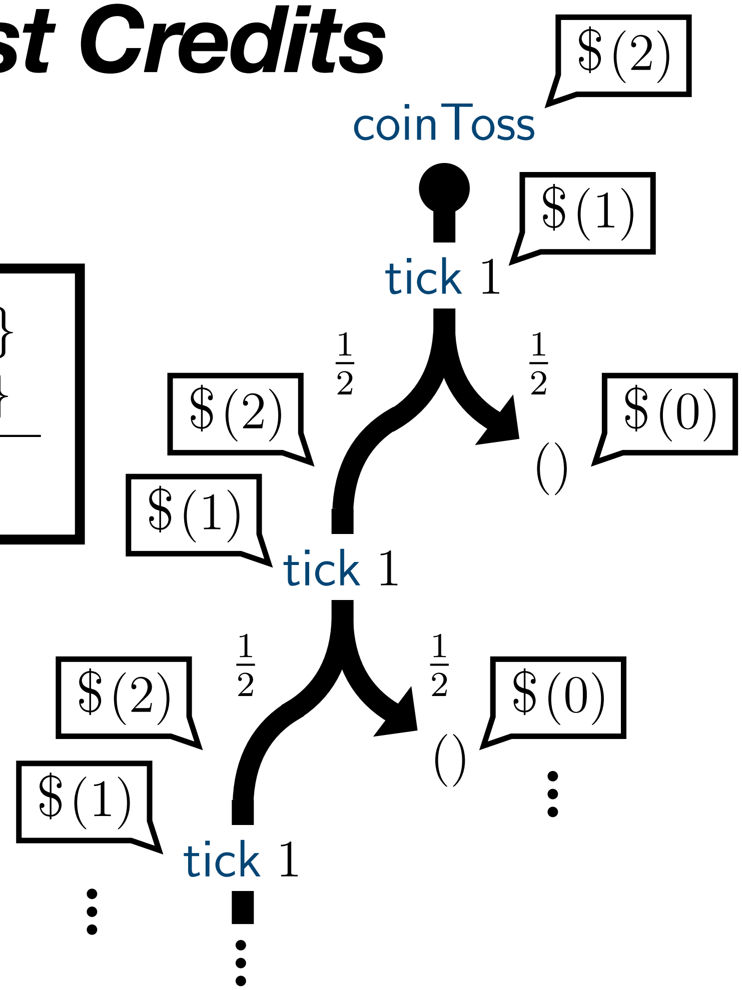
TACHIS Expected Cost Credits

$$\mathbb{E}[T] = 2$$

```
rec coinToss _ =
  tick 1;
  if flip
    then ()
    else coinToss ()
```

$$\frac{\{\$(2) * P\}\ \mathsf{false}\ \{Q\} \quad \{\$(0) * P\}\ \mathsf{true}\ \{Q\}}{\{\$(1) * P\}\ \mathsf{flip}\ \{Q\}}$$

$$\vdash \{\$(2)\}\ \mathsf{coinToss}\ \{\top\}$$



TACHIS Expected Cost Credits

Expected Cost Bounds as a Resource

$$\vdash \{\$(x)\}\ f\ \{v.\,P\}$$

The expected cost of f is x,

and $P\,v$ holds on its result.

Iris

Step-indexed & higher-order

Mechanized in Rocq

TACHIS

Expected Cost Credits

Expected Cost Bounds as a Resource

$$\vdash \{\$(x)\}\ f\ \{v.\, P\}$$

- Averaging rule
- User-defined cost models
- Generalizes rules from Iris$

Iris

Step-indexed & higher-order
Mechanized in Rocq

Example: Batch Sampling

- *Sample a sequence of coin flips with access to only* randByte

    eg. /dev/random

TACHIS Example: Batch Sampling

- *Sample a sequence of coin flips with access to only* $\mathsf{randByte}$

  eg. /dev/random

$$\mathsf{let}\ \mathsf{s} = \mathsf{randByte}\ \mathsf{in}\ \mathsf{s}\ \&\ 1$$

TACHIS Example: Batch Sampling

- *Sample a sequence of coin flips with access to only* $\mathsf{randByte}$

  eg. /dev/random

$$\mathsf{let}\ \mathsf{s} = \mathsf{randByte}\ \mathsf{in}\ \mathsf{s}\ \&\ 1$$

- *Wastes entropy!*

TACHIS Example: Batch Sampling

- *Sample a sequence of coin flips with access to only* randByte

  eg. /dev/random

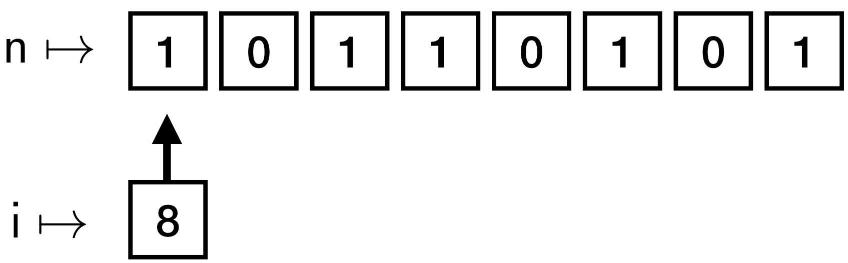
$$\{\$(8)\}\ \mathsf{let\ s} = \mathsf{randByte\ in\ s}\ \&\ 1\ \{\top\}$$

- *Wastes entropy!*

  Entropy model $cost((\mathsf{rand}\ N), \cdot) = \log_2(N)$

TACHIS Example: Batch Sampling

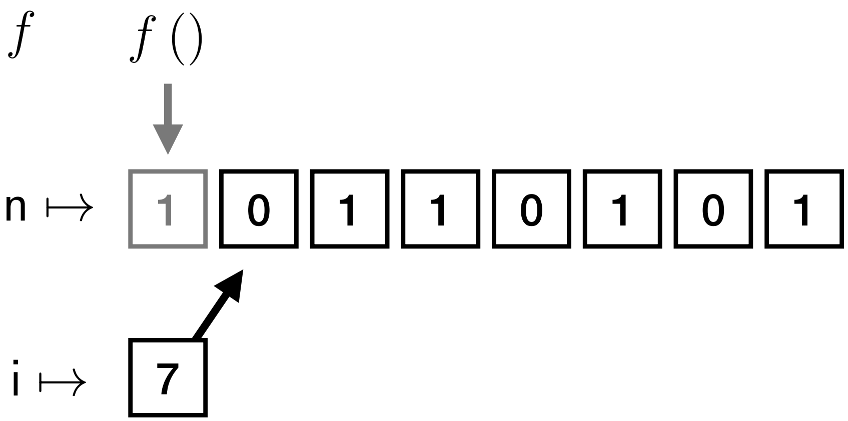
f




```
batchFlip ≜
  let n = ref(randByte) in
  let i = ref(8) in
  (λ_.
    if (!i = 0) {n ← randByte; i ← 8; }
    i ← (!i − 1);
    (!n >> i) & 1)
```

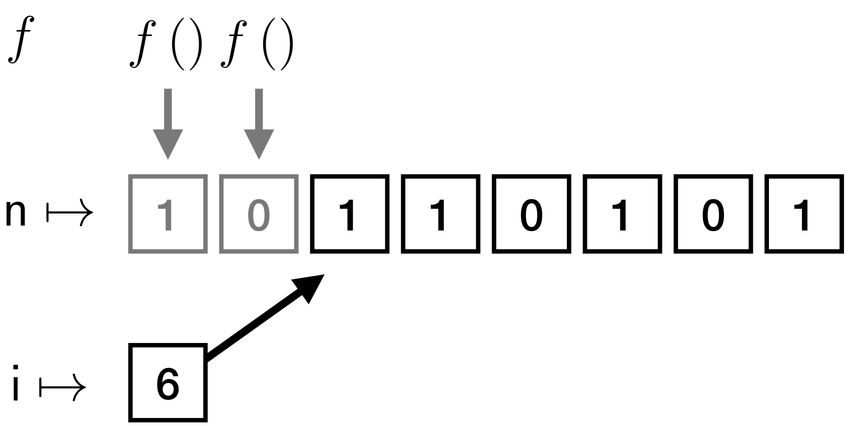
TACHIS

Example: Batch Sampling



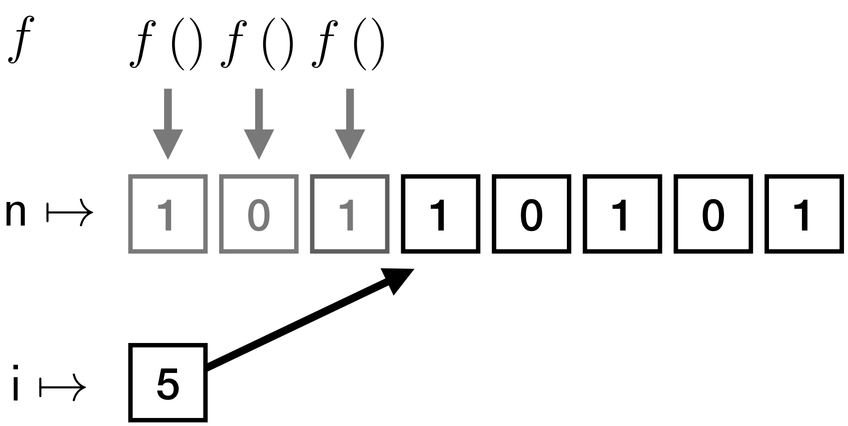
$$
\begin{array}{l}
\mathsf{batchFlip} \triangleq \\
\quad \mathsf{let}\ \mathsf{n} = \mathsf{ref}(\mathsf{randByte})\ \mathsf{in} \\
\quad \mathsf{let}\ \mathsf{i} = \mathsf{ref}(8)\ \mathsf{in} \\
\quad (\lambda\_. \\
\qquad \mathsf{if}\ \ (!\,\mathsf{i} = 0)\ \{\mathsf{n} \leftarrow \mathsf{randByte};\ \ \mathsf{i} \leftarrow 8;\}\\
\qquad \mathsf{i} \leftarrow (!\,\mathsf{i} - 1); \\
\qquad (!\,\mathsf{n}\ \ >> i)\ \&\ 1)
\end{array}
$$

Example: Batch Sampling



$$
\begin{aligned}
&\mathsf{batchFlip} \triangleq \\
&\quad \mathsf{let}\ \mathsf{n} = \mathsf{ref}(\mathsf{randByte})\ \mathsf{in} \\
&\quad \mathsf{let}\ \mathsf{i} = \mathsf{ref}(8)\ \mathsf{in} \\
&\quad (\lambda \_. \\
&\qquad \mathsf{if}\ (!\,\mathsf{i} = 0)\ \{\mathsf{n} \leftarrow \mathsf{randByte};\ \mathsf{i} \leftarrow 8;\} \\
&\qquad \mathsf{i} \leftarrow (!\,\mathsf{i} - 1); \\
&\qquad (!\,\mathsf{n}\ >> i)\ \&\ 1)
\end{aligned}
$$

TACHIS Example: Batch Sampling



$\mathsf{batchFlip} \triangleq$

$\quad \mathsf{let\ n} = \mathsf{ref}(\mathsf{randByte})\ \mathsf{in}$

$\quad \mathsf{let\ i} = \mathsf{ref}(8)\ \mathsf{in}$

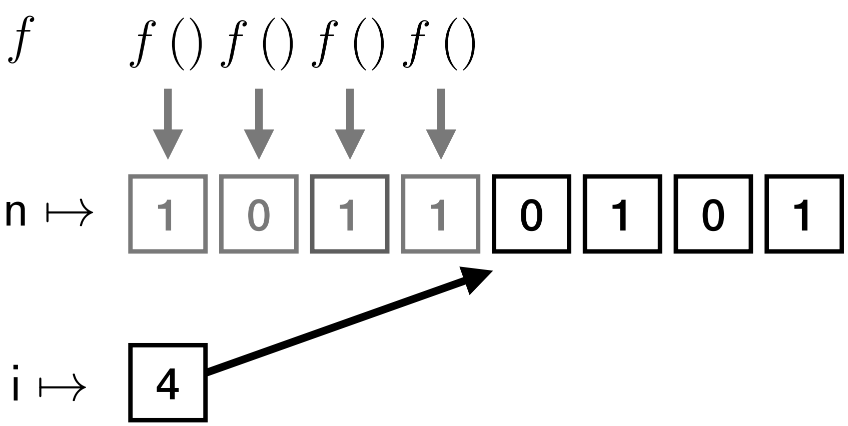
$\quad (\lambda\_.$

$\qquad \mathsf{if}\ \ (!\,\mathsf{i} = 0)\ \{\mathsf{n} \leftarrow \mathsf{randByte};\ \ \mathsf{i} \leftarrow 8;\}$

$\qquad \mathsf{i} \leftarrow (!\,\mathsf{i} - 1);$

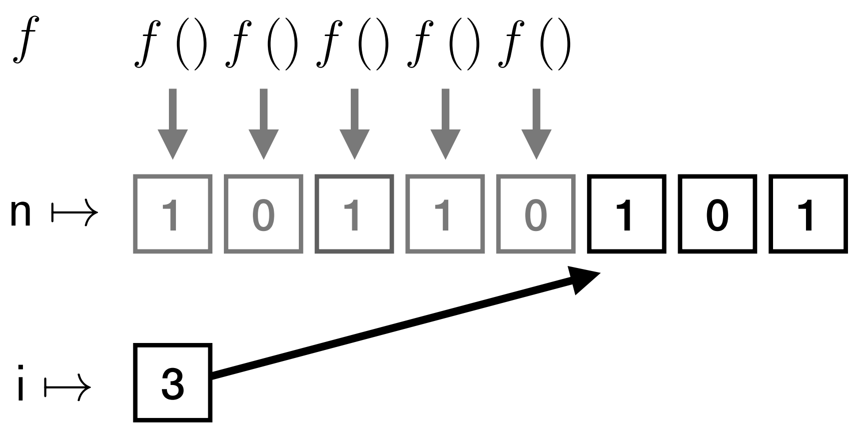
$\qquad (!\,\mathsf{n}\ \ >> i)\ \&\ 1)$

TACHIS Example: Batch Sampling



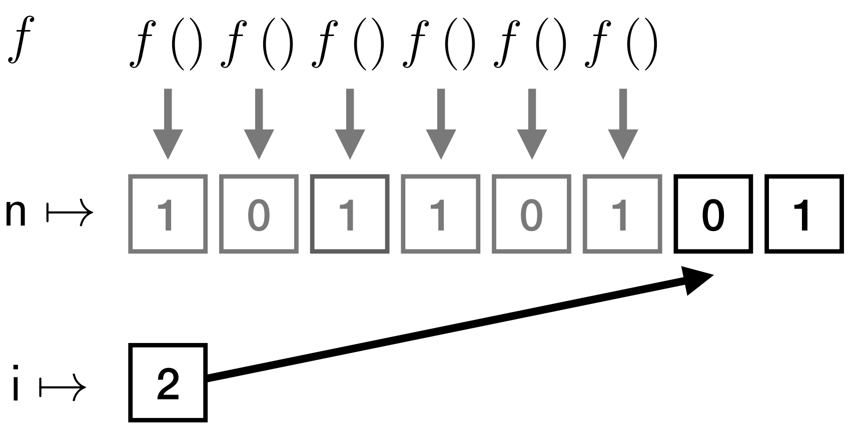

```
batchFlip ≜
  let n = ref(randByte) in
  let i = ref(8) in
  (λ_.
    if (!i = 0) {n ← randByte; i ← 8; }
    i ← (!i − 1);
    (!n >> i) & 1)
```

Example: Batch Sampling



$$
\begin{array}{l}
\mathsf{batchFlip} \triangleq \\
\quad \mathsf{let\ n} = \mathsf{ref}(\mathsf{randByte})\ \mathsf{in} \\
\quad \mathsf{let\ i} = \mathsf{ref}(8)\ \mathsf{in} \\
\quad (\lambda\_. \\
\qquad \mathsf{if}\ (!\,\mathsf{i} = 0)\ \{\mathsf{n} \leftarrow \mathsf{randByte};\ \mathsf{i} \leftarrow 8;\} \\
\qquad \mathsf{i} \leftarrow (!\,\mathsf{i} - 1); \\
\qquad (!\,\mathsf{n}\ >>\ i)\ \&\ 1)
\end{array}
$$

TACHIS Example: Batch Sampling



$\mathsf{batchFlip} \triangleq$

$\quad \mathsf{let\ n} = \mathsf{ref}(\mathsf{randByte})\ \mathsf{in}$

$\quad \mathsf{let\ i} = \mathsf{ref}(8)\ \mathsf{in}$

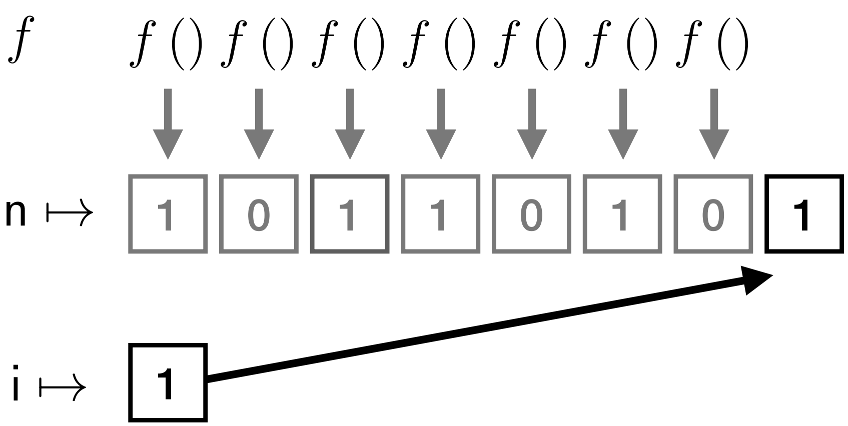
$\quad (\lambda\_.$

$\quad\quad \mathsf{if}\ (!\,\mathsf{i} = 0)\ \{\mathsf{n} \leftarrow \mathsf{randByte};\ \mathsf{i} \leftarrow 8;\}$

$\quad\quad \mathsf{i} \leftarrow (!\,\mathsf{i} - 1);$

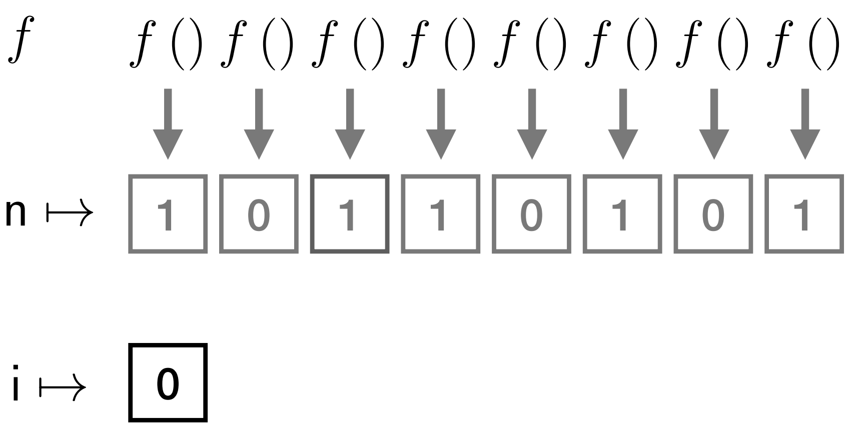
$\quad\quad (!\,\mathsf{n}\ >> i)\ \&\ 1)$

Example: Batch Sampling



$$
\begin{array}{l}
\mathsf{batchFlip} \triangleq \\
\quad \mathsf{let}\ \mathsf{n} = \mathsf{ref}(\mathsf{randByte})\ \mathsf{in} \\
\quad \mathsf{let}\ \mathsf{i} = \mathsf{ref}(8)\ \mathsf{in} \\
\quad (\lambda\_. \\
\qquad \mathsf{if}\ \ (!\,\mathsf{i} = 0)\ \{\mathsf{n} \leftarrow \mathsf{randByte};\ \ \mathsf{i} \leftarrow 8;\}\\
\qquad \mathsf{i} \leftarrow (!\,\mathsf{i} - 1); \\
\qquad (!\,\mathsf{n}\ \ >> i)\ \&\ 1)
\end{array}
$$

Example: Batch Sampling




```
batchFlip ≜
  let n = ref(randByte) in
  let i = ref(8) in
  (λ_.
    if (! i = 0) {n ← randByte; i ← 8; }
    i ← (! i − 1);
    (! n  >> i) & 1)
```

TACHIS Example: Batch Sampling

f

n $\mapsto$ | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |

i $\mapsto$ | 0 |

$$
\begin{aligned}
&\mathsf{batchFlip} \triangleq \\
&\quad \mathsf{let\ n} = \mathsf{ref}(\mathsf{randByte})\ \mathsf{in} \\
&\quad \mathsf{let\ i} = \mathsf{ref}(8)\ \mathsf{in} \\
&\quad (\lambda\_. \\
&\qquad \mathsf{if}\ \ (!\,\mathsf{i} = 0)\ \{\mathsf{n} \leftarrow \mathsf{randByte};\ \ \mathsf{i} \leftarrow 8;\}\\
&\qquad \mathsf{i} \leftarrow (!\,\mathsf{i} - 1); \\
&\qquad (!\,\mathsf{n}\ \ >> i)\ \&\ 1)
\end{aligned}
$$

TACHIS Example: Batch Sampling

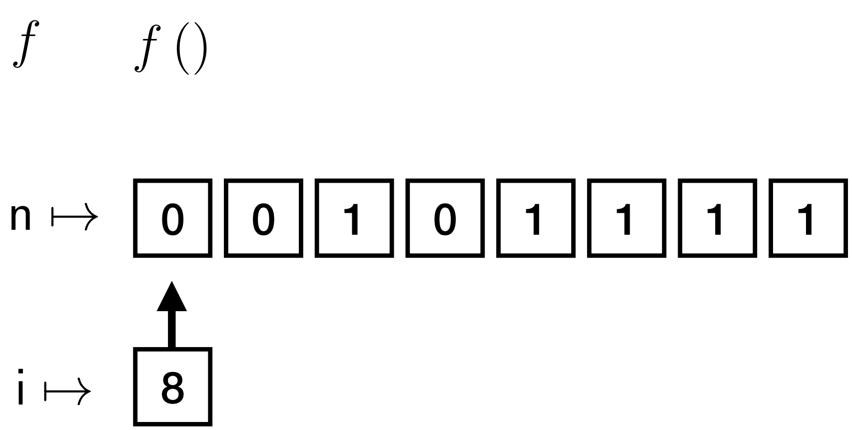
f $\quad f\,()$

n $\mapsto$ | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |

i $\mapsto$ | 0 |

$$
\begin{aligned}
&\mathsf{batchFlip} \triangleq \\
&\quad \mathsf{let\ n} = \mathsf{ref}(\mathsf{randByte})\ \mathsf{in} \\
&\quad \mathsf{let\ i} = \mathsf{ref}(8)\ \mathsf{in} \\
&\quad (\lambda\_. \\
&\qquad \mathsf{if}\ \ (!\,\mathsf{i} = 0)\ \{\mathsf{n} \leftarrow \mathsf{randByte};\ \ \mathsf{i} \leftarrow 8;\}\\
&\qquad \mathsf{i} \leftarrow (!\,\mathsf{i} - 1); \\
&\qquad (!\,\mathsf{n}\ \ >> i)\ \&\ 1)
\end{aligned}
$$

TACHIS Example: Batch Sampling



$\mathsf{batchFlip} \triangleq$

$\quad \mathsf{let}\ \mathsf{n} = \mathsf{ref}(\mathsf{randByte})\ \mathsf{in}$

$\quad \mathsf{let}\ \mathsf{i} = \mathsf{ref}(8)\ \mathsf{in}$

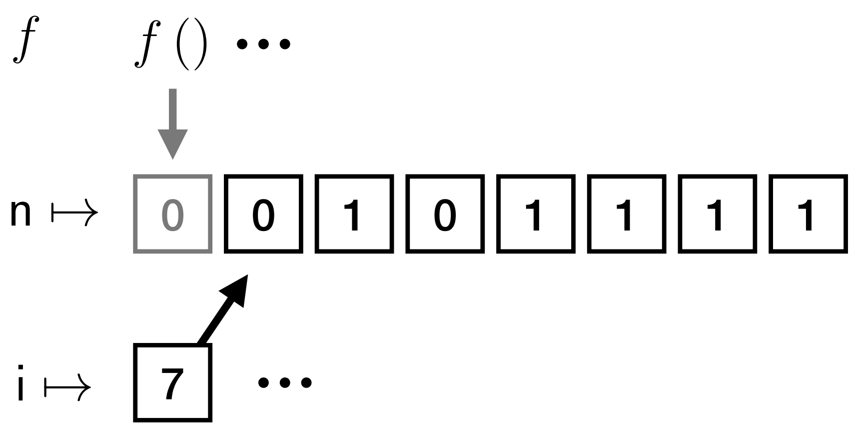
$\quad (\lambda\_.$

$\qquad \mathsf{if}\ (!\,\mathsf{i} = 0)\ \{\mathsf{n} \leftarrow \mathsf{randByte};\ \mathsf{i} \leftarrow 8;\}$

$\qquad \mathsf{i} \leftarrow (!\,\mathsf{i} - 1);$

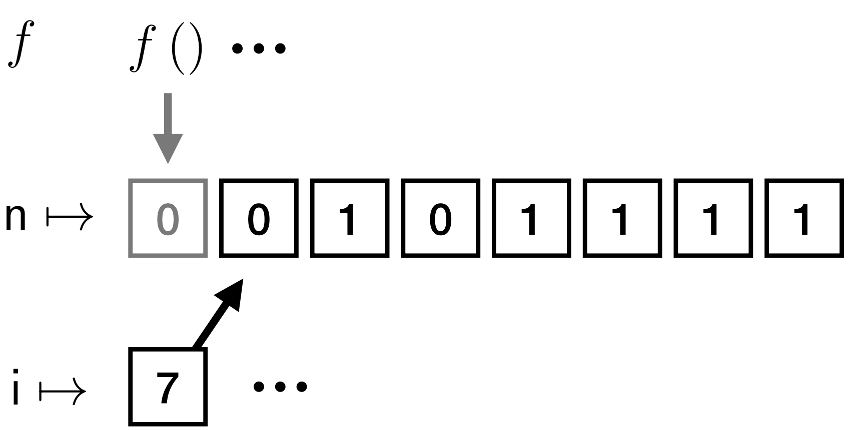
$\qquad (!\,\mathsf{n}\ >>\ i)\ \&\ 1)$

TACHIS Example: Batch Sampling




```
batchFlip ≜
  let n = ref(randByte) in
  let i = ref(8) in
  (λ_.
    if (!i = 0) {n ← randByte; i ← 8;}
    i ← (!i − 1);
    (!n >> i) & 1)
```

TACHIS Example: Batch Sampling




```
batchFlip ≜
  let n = ref(randByte) in
  let i = ref(8) in
  (λ_.
    if (!i = 0) {n ← randByte; i ← 8; }
    i ← (!i − 1);
    (!n >> i) & 1)
```

Verify expected entropy use in Tachis?

TACHIS Example: Batch Sampling

Entropy model $cost((\mathsf{rand}\ N), \cdot) = \log_2(N)$

$\{\$(8)\}\ \mathsf{batchFlip}\ \{f.\, I * S_f\}$

$$
\begin{aligned}
&\mathsf{batchFlip} \triangleq \\
&\quad \mathsf{let\ n} = \mathsf{ref}(\mathsf{randByte})\ \mathsf{in} \\
&\quad \mathsf{let\ i} = \mathsf{ref}(8)\ \mathsf{in} \\
&\quad (\lambda\_. \\
&\qquad \mathsf{if}\ \ (!\,\mathsf{i} = 0)\ \{\mathsf{n} \leftarrow \mathsf{randByte};\ \ \mathsf{i} \leftarrow 8;\}\\
&\qquad \mathsf{i} \leftarrow (!\,\mathsf{i} - 1); \\
&\qquad (!\,\mathsf{n}\ \ >> i)\ \&\ 1)
\end{aligned}
$$

TACHIS Example: Batch Sampling

Entropy model $cost((\text{rand } N), \cdot) = \log_2(N)$

$$\{\$(8)\}\ \text{batchFlip}\ \{f.\, I * S_f\}$$

$$S_f \triangleq \{\$(1) * I\}\ f\ ()\ \{I\}$$

$$
\begin{array}{l}
\text{batchFlip} \triangleq \\
\quad \text{let n} = \text{ref}(\text{randByte}) \text{ in} \\
\quad \text{let i} = \text{ref}(8) \text{ in} \\
\quad (\lambda\_. \\
\quad\quad \text{if } (!\,\text{i} = 0)\ \{\text{n} \leftarrow \text{randByte};\ \text{i} \leftarrow 8;\} \\
\quad\quad \text{i} \leftarrow (!\,\text{i} - 1); \\
\quad\quad (!\,\text{n} >> i)\ \&\ 1)
\end{array}
$$

TACHIS Example: Batch Sampling

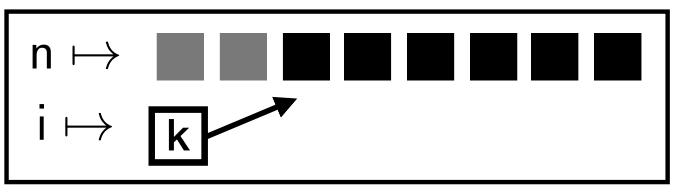
Entropy model $cost((\mathsf{rand}\ N), \cdot) = \log_2(N)$

$$\{\$(8)\}\ \mathsf{batchFlip}\ \{f.\, I * S_f\}$$

$$S_f \triangleq \{\$(1) * I\}\ f\ ()\ \{I\}$$

$$I \triangleq \exists k < 8.$$

$$\$(8-k) *$$




```
batchFlip ≜
  let n = ref(randByte) in
  let i = ref(8) in
  (λ_.
    if (!i = 0) {n ← randByte; i ← 8;}
    i ← (!i − 1);
    (!n >> i) & 1)
```

Example: Batch Sampling

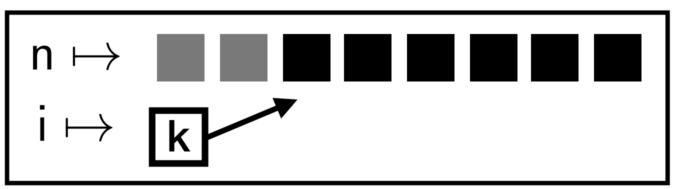
Entropy model $cost((\mathsf{rand}\ N), \cdot) = \log_2(N)$

$$\{\$(8)\}\ \mathsf{batchFlip}\ \{f.\, I * S_f\}$$

$$S_f \triangleq \{\$(1)\ * I\}\ f\ ()\ \{I\}$$

$$I \triangleq \exists k < 8.$$

$$\$(8-k)\ *$$

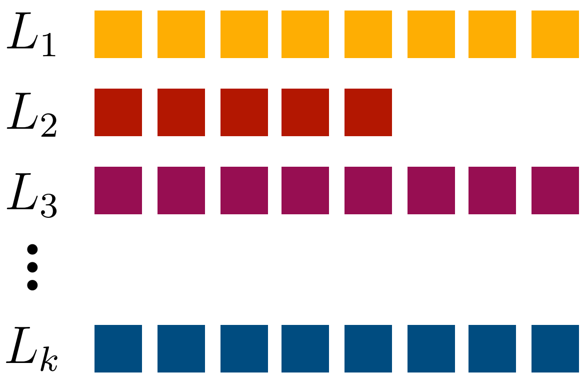



```
batchFlip ≜
  let n = ref(randByte) in
  let i = ref(8) in
  (λ_.
    if (!i = 0) {n ← randByte; i ← 8; }
    i ← (!i − 1);
    (!n >> i) & 1)
```

- *Amortize entropy consumption of* $\mathsf{randByte}$
- *Higher-order, stateful specification*

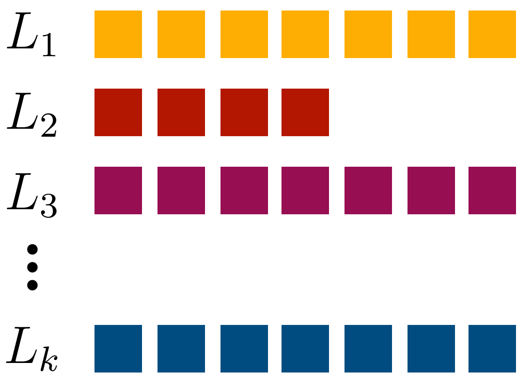
TACHIS Example: K-way merge

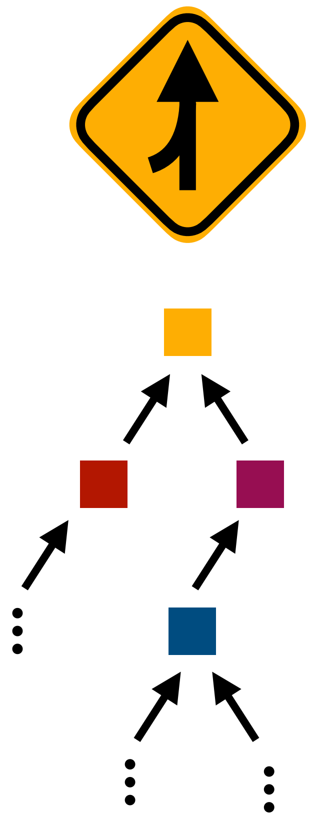
- *K sorted lists*



Example: K-way merge

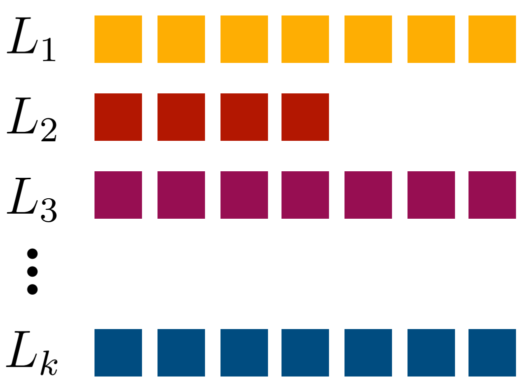
- *K sorted lists*

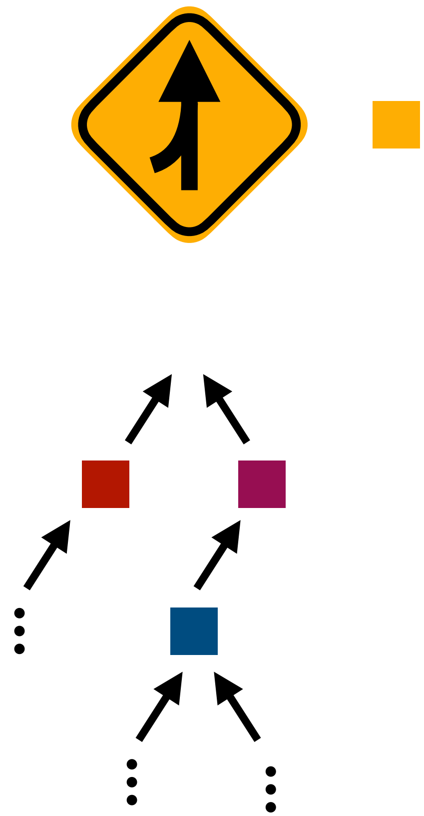




Example: K-way merge

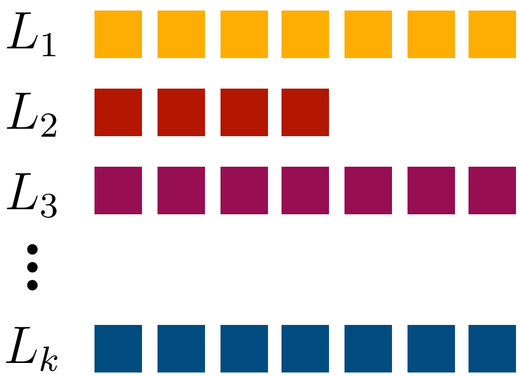
- *K sorted lists*

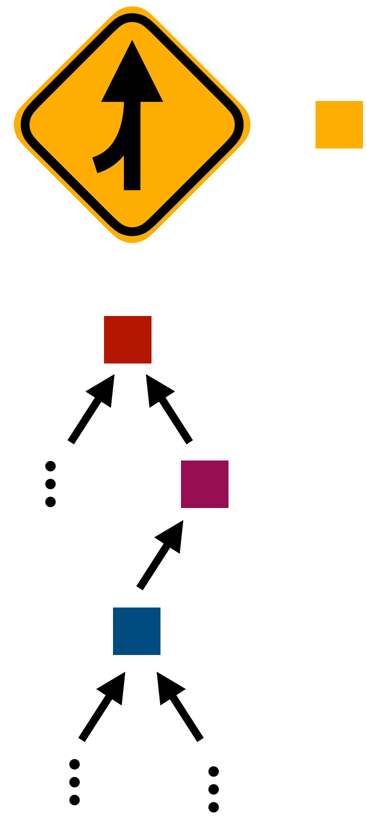




TACHIS Example: K-way merge

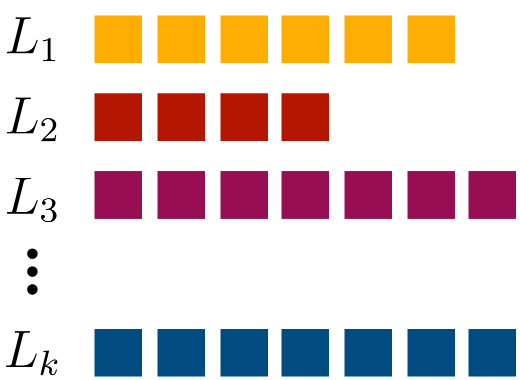
- *K sorted lists*

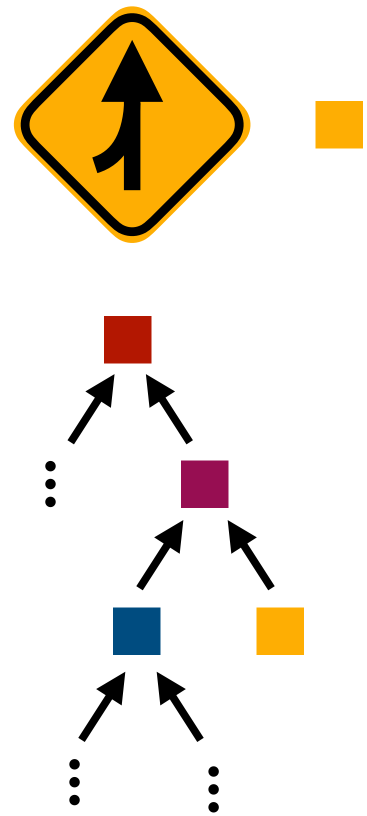




TACHIS Example: K-way merge

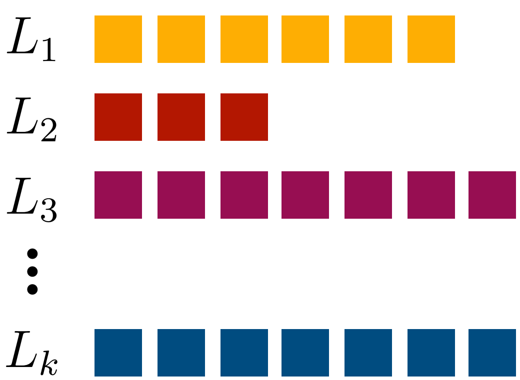
- *K sorted lists*

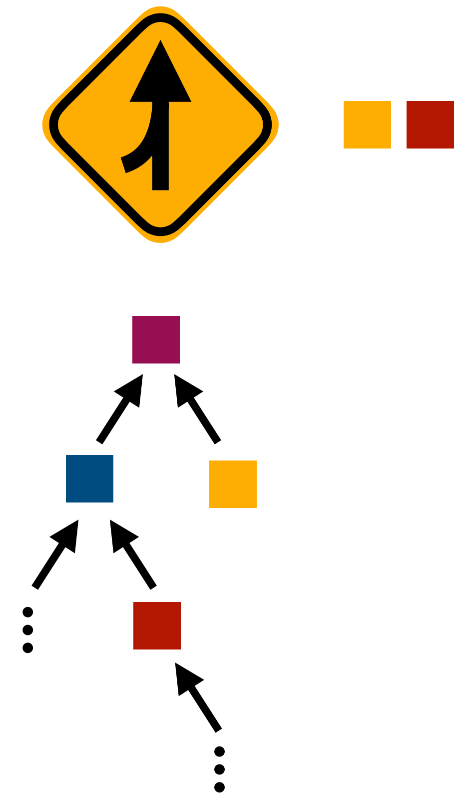




Example: K-way merge

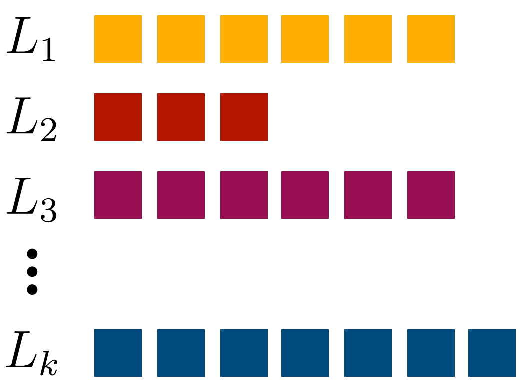
- *K sorted lists*

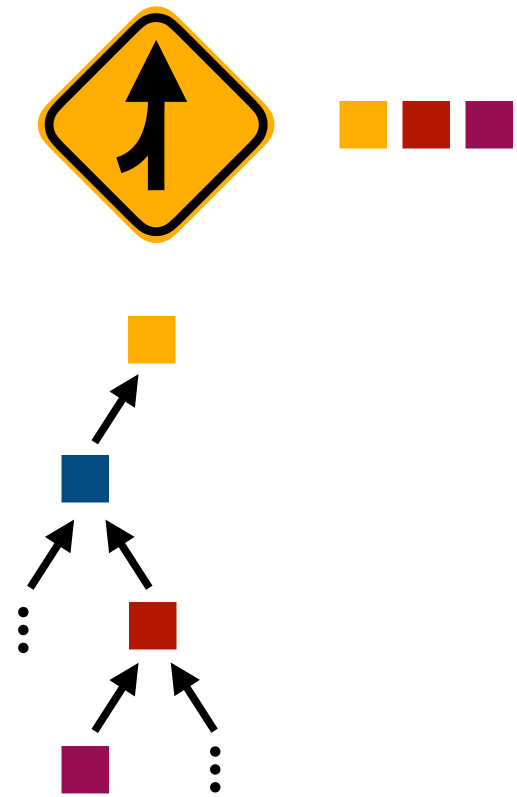




TACHIS Example: K-way merge

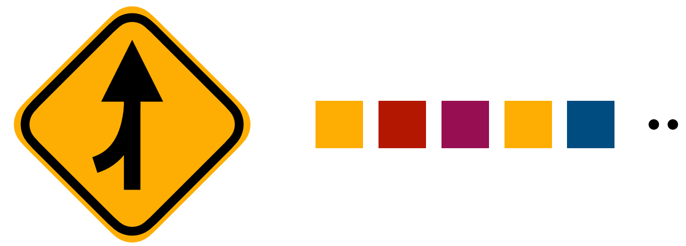
- *K sorted lists*





Example: K-way merge

- *K sorted lists*



L_1

L_2

L_3

L_k

Example: K-way merge

Runtime?

- *K sorted lists*

L_1

L_2

L_3

L_k

TACHIS Example: K-way merge

$\{\$(\mathcal{O}(\log k)) \ * \ \ldots\}$ insert $v \ h \ \{\ldots\}$

$\{\$(\mathcal{O}(\log k)) \ * \ \ldots\}$ remove $h \ \{\ldots\}$

$\{\$(\mathcal{O}(n \log k)) \ * \ \ldots\}$ kWayMerge $[L_1, L_2, \ldots, L_k] \ \{\ldots\}$

$n = \sum_i |L_i|$

TACHIS Example: K-way merge

$\{\$(\mathcal{O}(\log k)) \ * \ \ldots\}$ insert $v\ h\ \{\ldots\}$

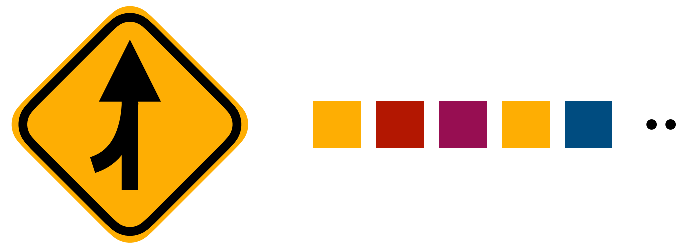
$\{\$(\mathcal{O}(\log k)) \ * \ \ldots\}$ remove $h\ \{\ldots\}$

Where is the randomness?

$\{\$(\mathcal{O}(n \log k)) \ * \ \ldots\}$ kWayMerge $[L_1, L_2, \ldots, L_k]\ \{\ldots\}$

$n = \sum_i |L_i|$

TACHIS Example: K-way merge



$\{\$(\mathcal{O}(\log k)) \ * \ \ldots\}$ insert $v \; h \; \{\ldots\}$

$\{\$(\mathcal{O}(\log k)) \ * \ \ldots\}$ remove $h \; \{\ldots\}$

Where is the randomness?

- ***Encapsulated!***

$\{\$(\mathcal{O}(n \log k)) \ * \ \ldots\}$ kWayMerge $[L_1, L_2, \ldots, L_k] \; \{\ldots\}$

$n = \sum_i |L_i|$

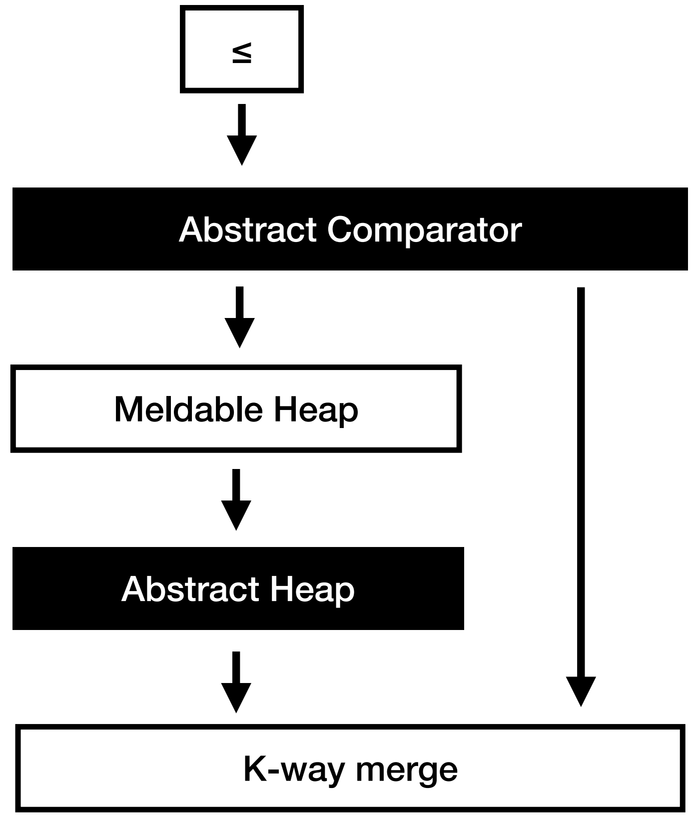
Example: K-way merge

$$\mathsf{isComp}(K, \mathsf{cmp}, \mathsf{hasKey}) \triangleq \exists R : K \to K \to \mathbb{B}, x : \mathbb{R}_{\geq 0}.\ \mathsf{PreOrder}(R) \land \mathsf{Total}(R) \land$$
$$\{\mathsf{hasKey}(k_1, v_2) * \mathsf{hasKey}(k_2, v_2) * \$(x)\}$$
$$\mathsf{cmp}\ v_1\ v_2$$
$$\{b.\ b = R(k_1, k_2) * \mathsf{hasKey}(k_1, v_2) * \mathsf{hasKey}(k_2, v_2)\}$$

Fig. 6. A specification for an abstract comparator.

$$\mathsf{isComp}(K, \mathsf{cmp}, \mathsf{hasKey}) \Rightarrow$$
$$\exists\, \mathsf{isHeap} : \mathit{List}(K) \to \mathit{Val} \to \mathit{iProp}, X_i, X_r : \mathbb{N} \to \mathbb{R}_{\geq 0}.$$
$$(\forall n, m.\ n \leq m \Rightarrow X_i(n) \leq X_i(m)) \land (\forall n, m.\ n \leq m \Rightarrow X_r(n) \leq X_r(m))$$
$$\land\ \{\mathsf{True}\}\ \mathsf{new}\ ()\ \{v.\ \mathsf{isHeap}([], v)\}$$
$$\land\ \{\mathsf{isHeap}(l, v) * \mathsf{hasKey}(k, w) * \$(X_i(|l|))\}\ \mathsf{insert}\ v\ w\ \{\_.\ \exists l'.\ \mathsf{isHeap}(l', v) * l \equiv_p (k :: l')\}$$
$$\land\ \{\mathsf{isHeap}(l, v) * \$(X_r(|l|))\}$$
$$\mathsf{remove}\ v$$
$$\left\{ w.\ \begin{array}{l} (w = \mathsf{None} * l = [] * \mathsf{isHeap}([], v)) \\ \lor\ (\exists u, k, l'.\ w = \mathsf{Some}\ u * l \equiv_p (k :: l') * \min(k, l) * \mathsf{hasKey}(k, u) * \mathsf{isHeap}(l', v)) \end{array} \right\}$$

Fig. 7. An abstract specification for a min-heap.



Challenge 3.

Expected Cost Bounds

- Expected cost bounds as a separation logic resource
- Generic cost model
- Encapsulated probabilistic reasoning

Rich functional language | Separation Logic | Quantitative reasoning

Expected values as state

Approximate Correctness — **Eris**

Almost-Sure Termination — **Total Eris**

Expected Cost Bounds — **Tachis**

TACHIS **_Implementation_**

$\mathrm{Auth}(\mathbb{R}_{\geq 0}, +)$ Iris

Cost Credit $\$\,(x)$

$\$_{\bullet}\,(x)$ **Cost Interpretation**

$\{P\}\, e\, \{Q\}$

TACHIS Implementation

$\mathsf{Auth}(\mathbb{R}_{\geq 0}, +)$ Iris

Cost Credit $\$\,(x)$

$\$_\bullet\,(x)$ **Cost Interpretation**

$\{P\}\, e\, \{Q\}$

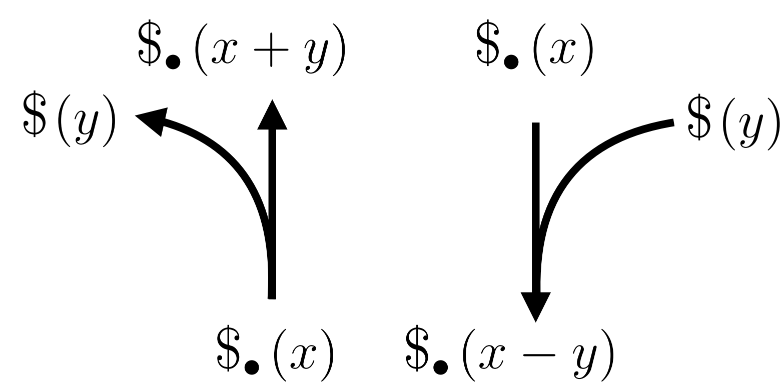
Splitting $\$\,(x+y) \dashv\vdash \$\,(x) \ast \$\,(y)$

Agreement $\$\,(x_1) \ast \$_\bullet\,(x_2) \vdash x_1 \leq x_2$

Local, Higher-order specs, Invariants…

Iris

Expected cost credit upper bound



TACHIS Implementation

$$\frac{cost(\textsf{tick}\ 1, \cdot) = 1}{(\textsf{tick}\ 1, \sigma) \to_1 ((), \sigma)}$$

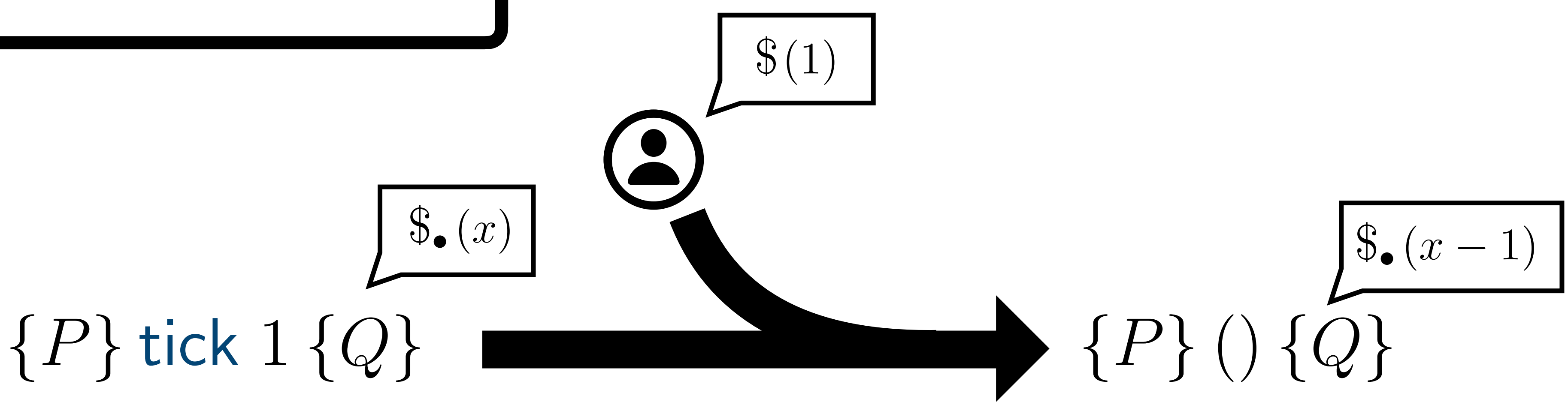
$\$_\bullet(x)$

$\{P\}\ \textsf{tick}\ 1\ \{Q\}$ → $\{P\}\ ()\ \{Q\}$

$\$_\bullet(x-1)$

TACHIS Implementation

$$\frac{cost(\textsf{tick}\ 1, \cdot) = 1}{(\textsf{tick}\ 1, \sigma) \rightarrow_1 ((), \sigma)}$$

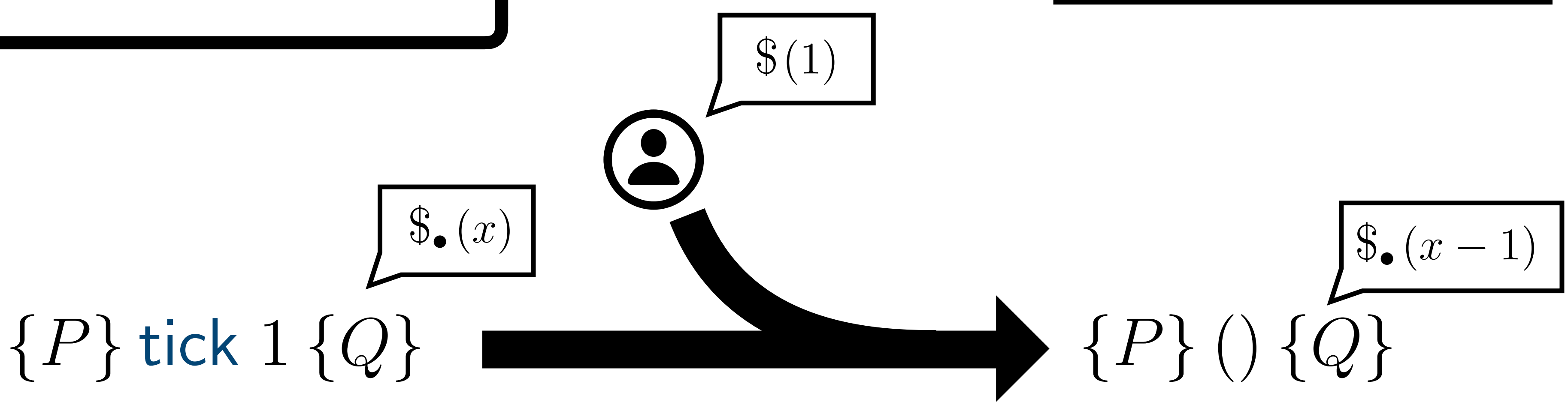


Implementation

$$cost(\textsf{tick}\ 1, \cdot) = 1$$

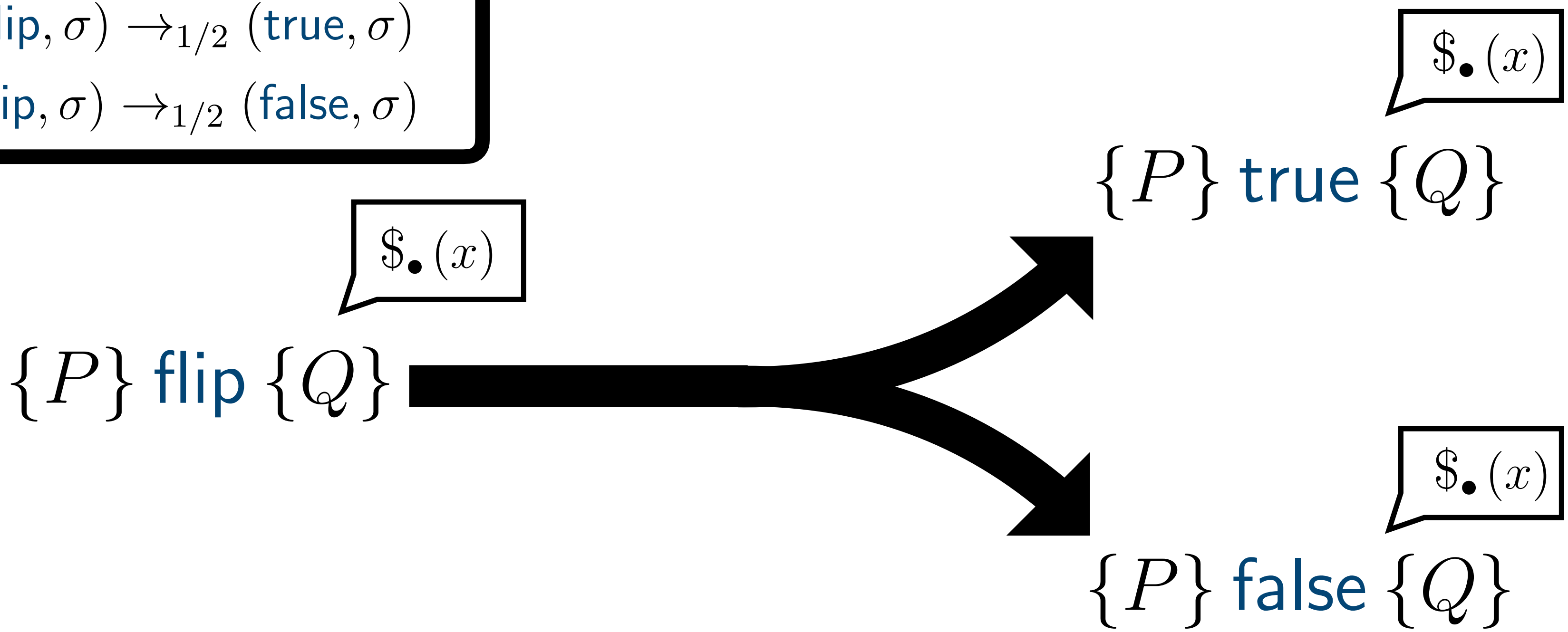
$$(\textsf{tick}\ 1, \sigma) \to_1 ((), \sigma)$$

$$\frac{\{\$(1) * P\}\ \textsf{tick}\ 1\ \{Q\}}{\{P\}\ ()\ \{Q\}}$$



TACHIS Implementation

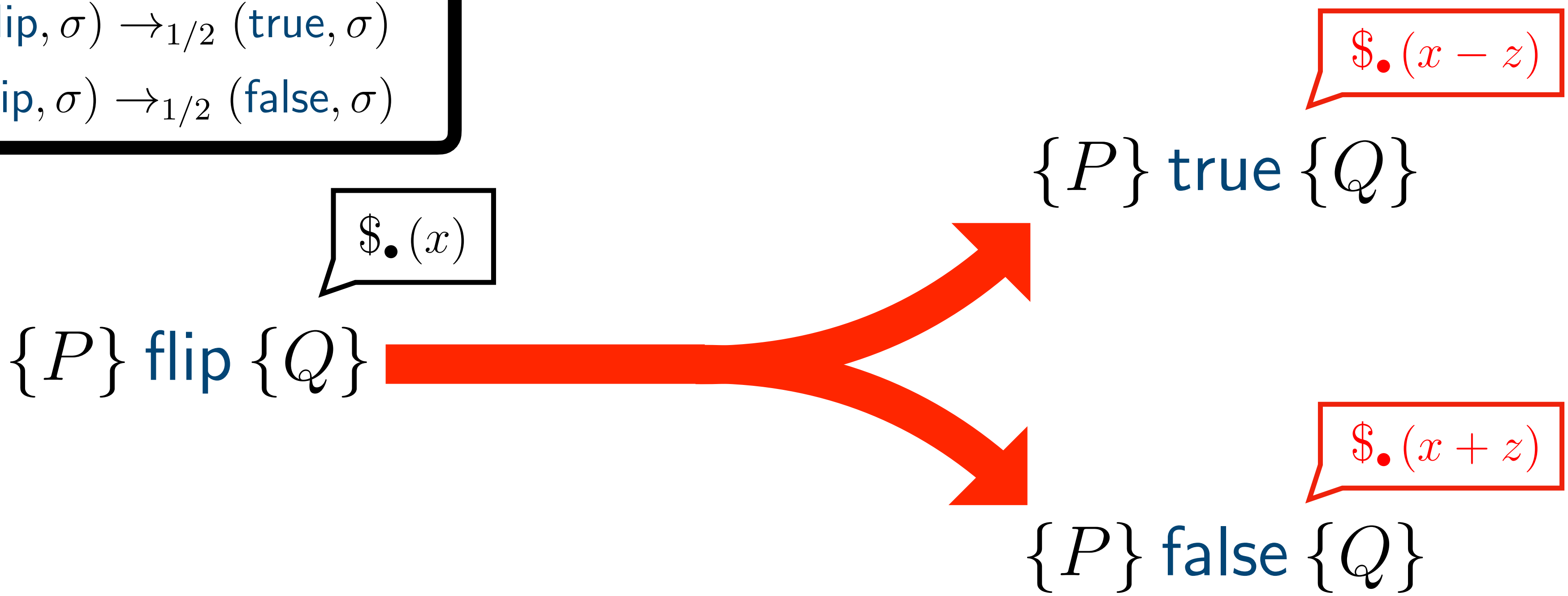
$$\frac{cost(\textsf{flip}, \cdot) = 0}{\begin{array}{c}(\textsf{flip}, \sigma) \to_{1/2} (\textsf{true}, \sigma) \\ (\textsf{flip}, \sigma) \to_{1/2} (\textsf{false}, \sigma)\end{array}}$$

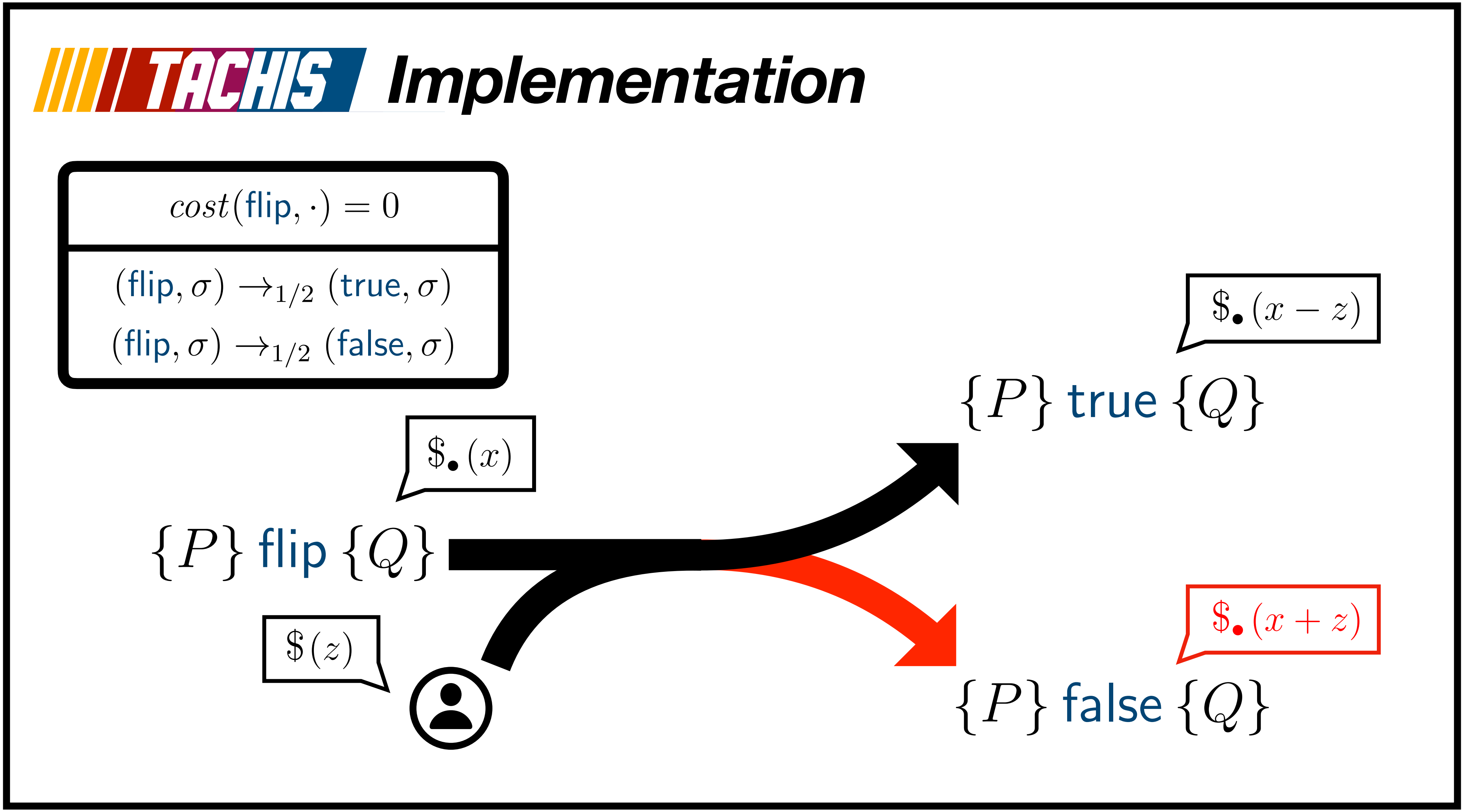


TACHIS Implementation

$$cost(\mathsf{flip}, \cdot) = 0$$

$$(\mathsf{flip}, \sigma) \to_{1/2} (\mathsf{true}, \sigma)$$
$$(\mathsf{flip}, \sigma) \to_{1/2} (\mathsf{false}, \sigma)$$

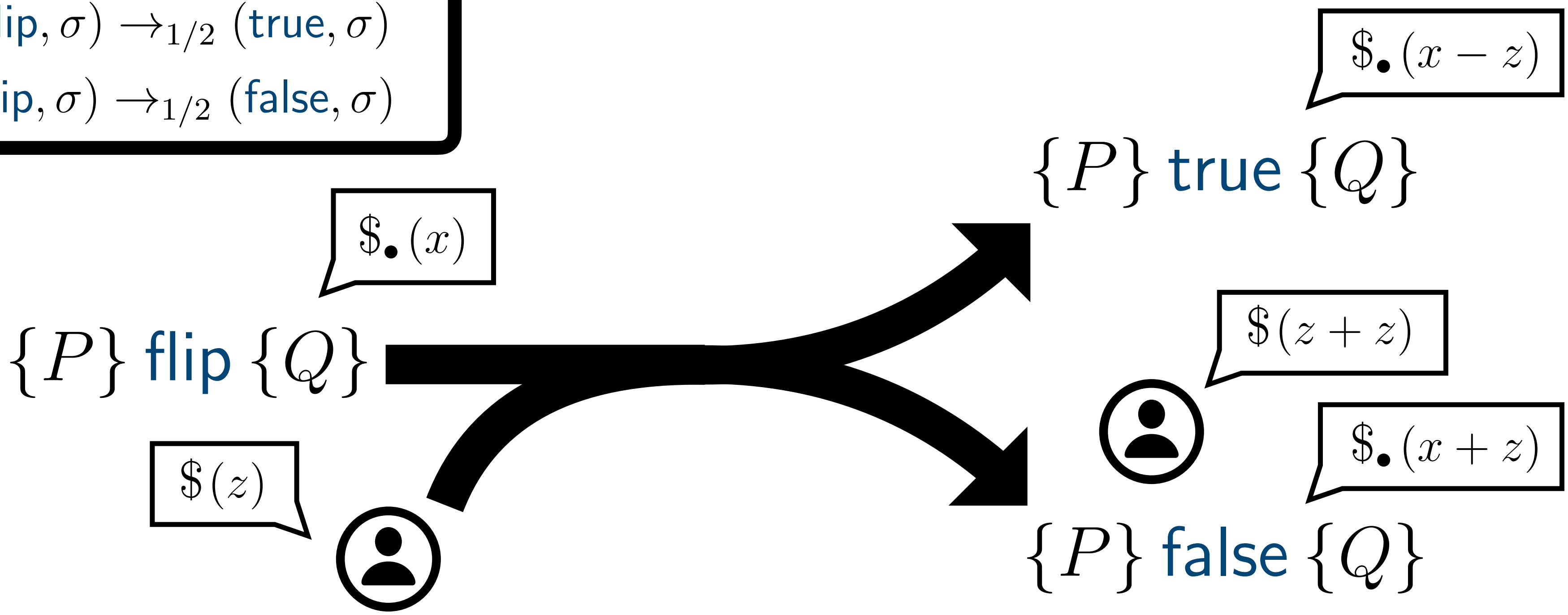




TACHIS
Implementation
$cost(\mathsf{flip}, \cdot) = 0$
$(\mathsf{flip}, \sigma) \to_{1/2} (\mathsf{true}, \sigma)$
$(\mathsf{flip}, \sigma) \to_{1/2} (\mathsf{false}, \sigma)$
$\$_\bullet(x - z)$
$\{P\}\ \mathsf{true}\ \{Q\}$
$\$_\bullet(x)$
$\{P\}\ \mathsf{flip}\ \{Q\}$
$\$(z)$
$\$_\bullet(x + z)$
$\{P\}\ \mathsf{false}\ \{Q\}$

TACHIS Implementation

$$\frac{cost(\mathsf{flip}, \cdot) = 0}{\begin{array}{c}(\mathsf{flip}, \sigma) \to_{1/2} (\mathsf{true}, \sigma) \\ (\mathsf{flip}, \sigma) \to_{1/2} (\mathsf{false}, \sigma)\end{array}}$$



TACHIS Implementation

$$cost(\textsf{flip}, \cdot) = 0$$

$$(\textsf{flip}, \sigma) \to_{1/2} (\textsf{true}, \sigma)$$
$$(\textsf{flip}, \sigma) \to_{1/2} (\textsf{false}, \sigma)$$

$$\frac{\mathbb{E}[f] = x \quad \{\$(f(\textsf{true})) * P\}\ \textsf{true}\ \{Q\} \quad \{\$(f(\textsf{false})) * P\}\ \textsf{false}\ \{Q\}}{\{\$(x) * P\}\ \textsf{flip}\ \{Q\}}$$