Data-driven Invariant Learning for Probabilistic Programs

Jialu Bao

PLDG, March 9th, 2022

Collaborated work with Nitesh Trivedi, Drashti Pathak, Justin Hsu, Subhajit Roy

Motivation

Motivation

Ex. 1

$z \leftarrow z + 1 \; [p] \; \mathrm{skip}$

Motivation

Ex. 1

$z \leftarrow z + 1\ [p]\ \mathrm{skip}$

What is the expected value of z at the end of program in term of program variables' value at the initialization?

Motivation

Ex. 1 $\mathbb{E}(z) = z + p$

$z \leftarrow z + 1 \; [p] \; \text{skip}$

What is the expected value of z at the end of program in term of program variables' value at the initialization?

Motivation

Ex. 1 $\mathbb{E}(z) = z + p$

$z \leftarrow z + 1 \; [p] \; \text{skip}$

Ex. 2

$\text{while } (\text{flip} == 0) \text{ do}$

$\quad \text{flip} \leftarrow 1 \; [p] \; z \leftarrow z + 1$

What is the expected value of z at the end of program in term of program variables' value at the initialization?

Motivation

Ex. 1 $\mathbb{E}(z) = z + p$

$z \leftarrow z + 1\ [p]\ \text{skip}$

What is the expected value of z at the end of program in term of program variables' value at the initialization?

Ex. 2 $\mathbb{E}(z) = z + [\text{flip} == 0] \cdot (1 - p)/p$

$\text{while } (\text{flip} == 0) \text{ do}$

$\quad \text{flip} \leftarrow 1\ [p]\ z \leftarrow z + 1$

Motivation

Ex. 1 $\mathbb{E}(z) = z + p$

$z \leftarrow z + 1 \; [p] \; \text{skip}$

What is the expected value of z at the end of program in term of program variables' value at the initialization?

Ex. 2 $\mathbb{E}(z) = z + [\text{flip} == 0] \cdot (1 - p)/p$

$\textbf{while} \; (\text{flip} == 0) \; \textbf{do}$

$\quad \text{flip} \leftarrow 1 \; [p] \; z \leftarrow z + 1$

Can we automatically find this answer?

Background: Weakest Pre-condition Calculus

Background: Weakest Pre-condition Calculus

- Edsger Dijkstra, 1975

Background: Weakest Pre-condition Calculus

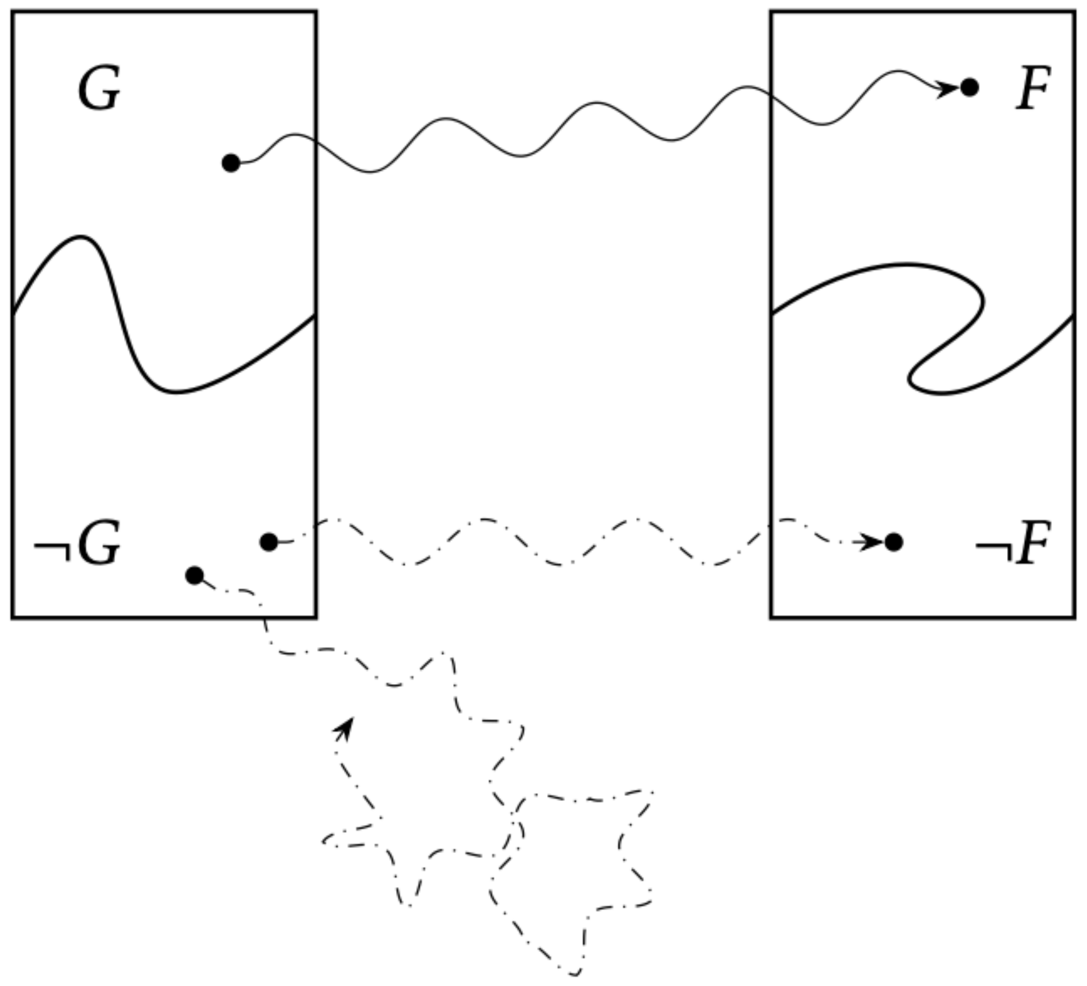
- Edsger Dijkstra, 1975
- $wpc : \text{Programs} \times \text{Assertions} \rightarrow \text{Assertions}$

Background: Weakest Pre-condition Calculus

- Edsger Dijkstra, 1975
- $wpc : \text{Programs} \times \text{Assertions} \to \text{Assertions}$
  - Say $wpc(C, F) = G$

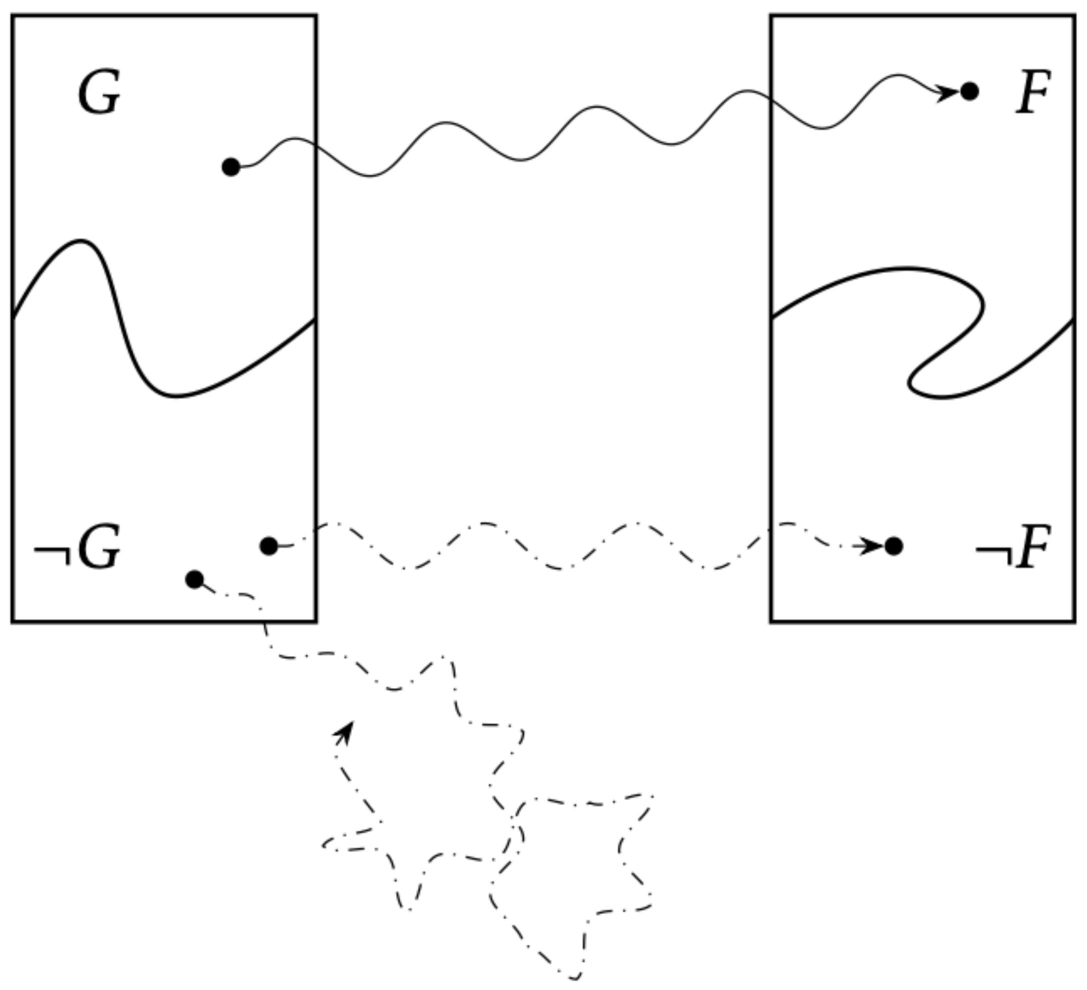
Background: Weakest Pre-condition Calculus

- Edsger Dijkstra, 1975
- wpc : Programs $\times$ Assertions $\rightarrow$ Assertions
  - Say $wpc(C, F) = G$



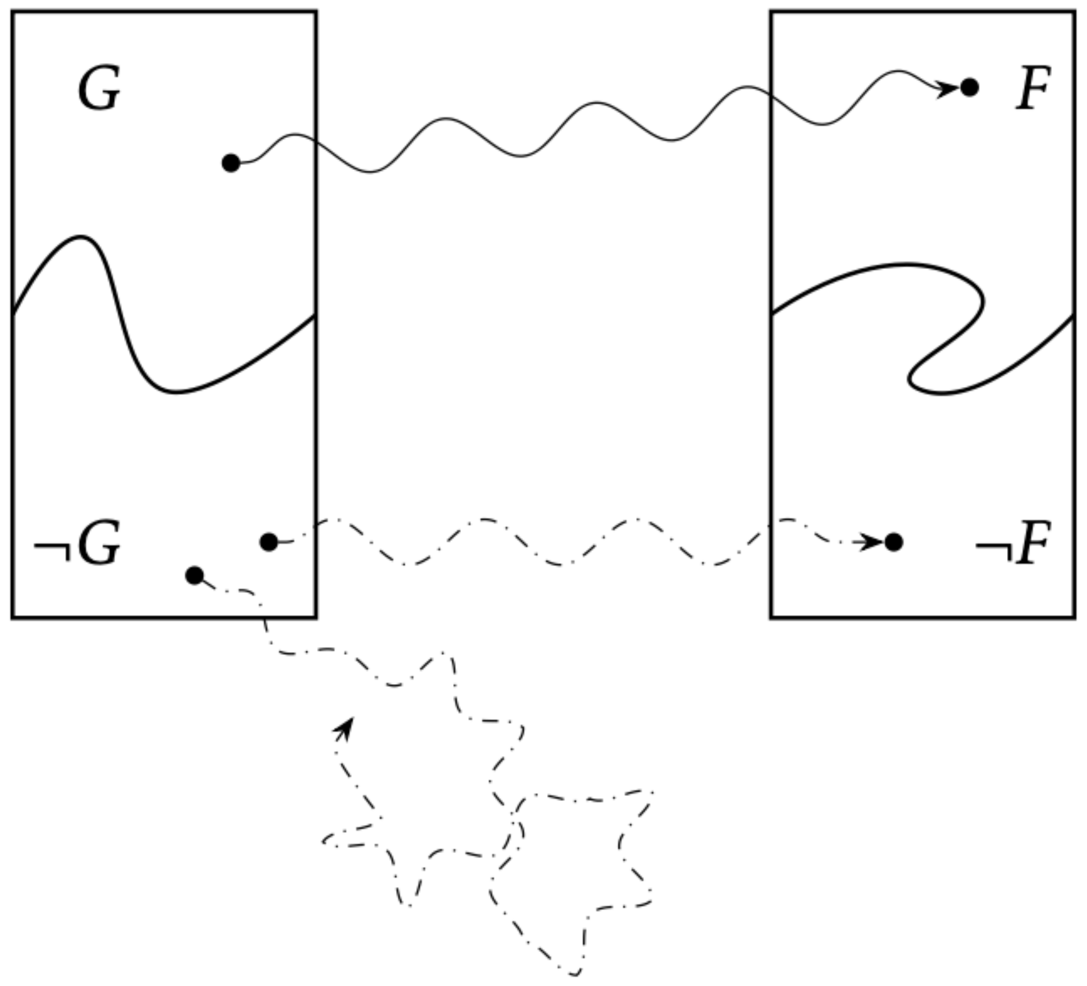
Background: Weakest Pre-condition Calculus

- Edsger Dijkstra, 1975
- wpc : Programs $\times$ Assertions $\rightarrow$ Assertions
  - Say $wpc(C, F) = G$



Background: Weakest Pre-condition Calculus

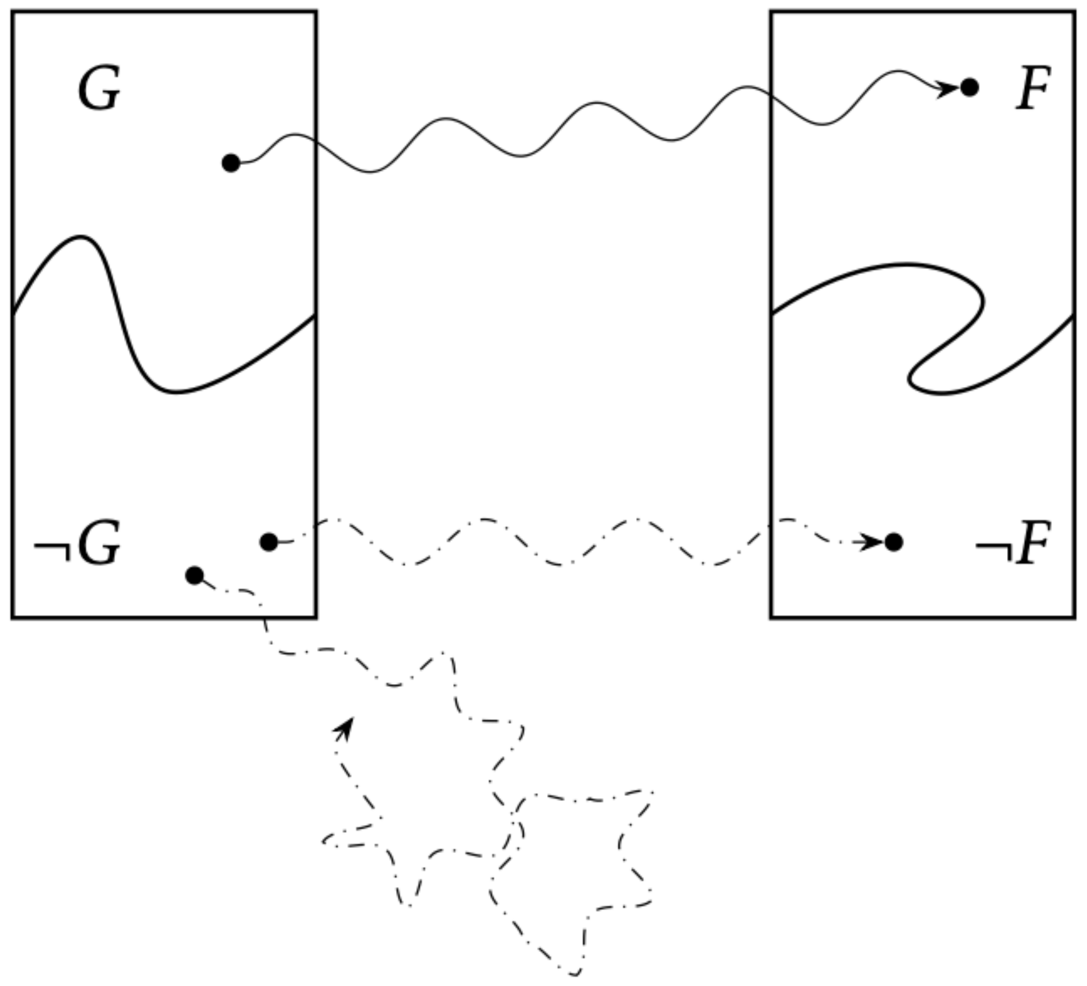
- Edsger Dijkstra, 1975
- wpc : Programs $\times$ Assertions $\rightarrow$ Assertions
  - Say $wpc(C, F) = G$



- Example rules:

Background: Weakest Pre-condition Calculus

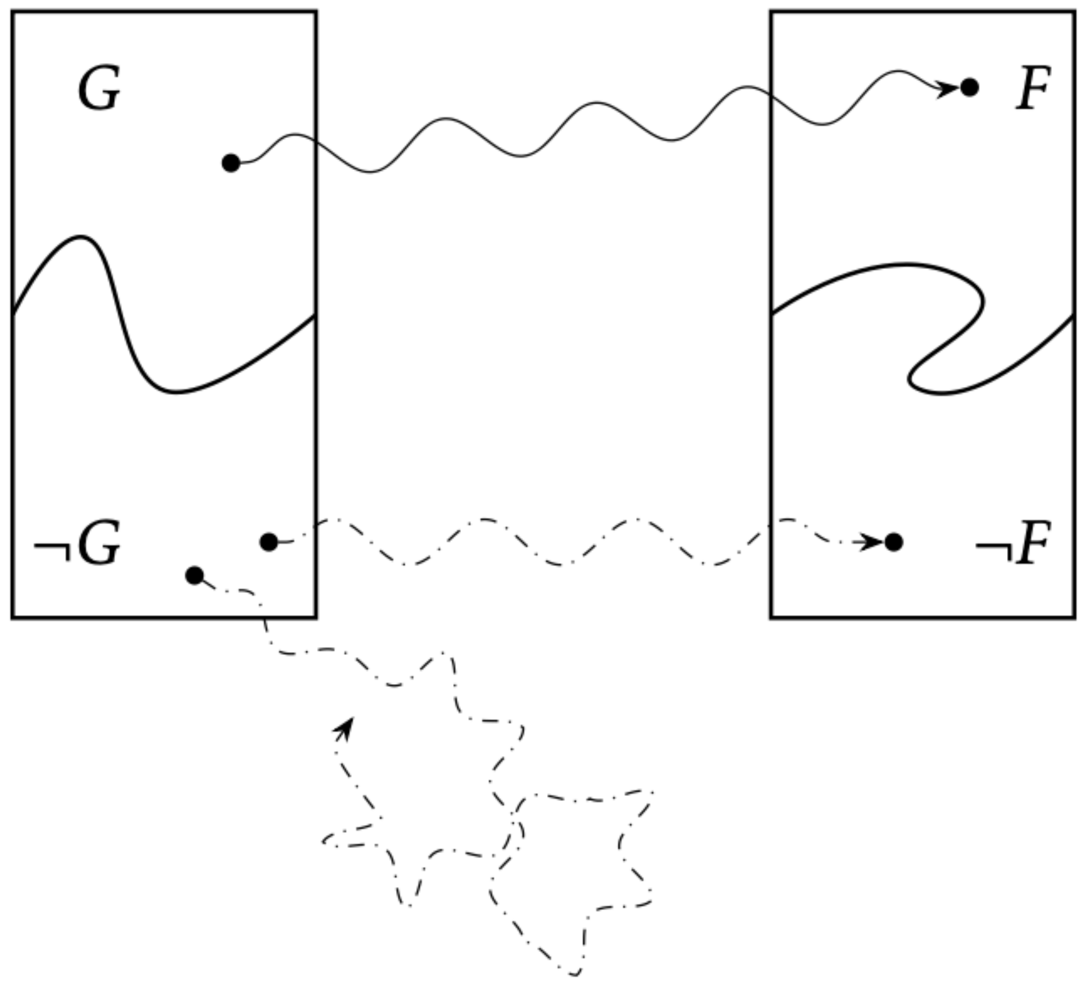
- Edsger Dijkstra, 1975
- wpc : Programs $\times$ Assertions $\rightarrow$ Assertions
  - Say $wpc(C, F) = G$



- Example rules:
  - Assignment: $wpc(x \leftarrow a, F) := F[a/x]$

Background: Weakest Pre-condition Calculus

- Edsger Dijkstra, 1975
- wpc : Programs × Assertions → Assertions
  - Say $wpc(C, F) = G$



- Example rules:
  - Assignment: $wpc(x \leftarrow a, F) := F[a/x]$
  - Sequencing: $wpc(P; Q, F) := wpc(P, wpc(Q, F))$

Background: Weakest Pre-expectation Calculus

Background: Weakest Pre-expectation Calculus

- Kozen, ~1985; McIver & Morgan and others, 1990s~today

Background: Weakest Pre-expectation Calculus

- Kozen, ~1985; McIver & Morgan and others, 1990s~today
- $\mathit{wpe} : \text{Programs} \times \text{Expectations} \to \text{Expectations}$

Background: Weakest Pre-expectation Calculus

- Kozen, ~1985; McIver & Morgan and others, 1990s~today
- $wpe : \text{Programs} \times \text{Expectations} \to \text{Expectations}$
- Expectations are maps from program states to numbers

Background: Weakest Pre-expectation Calculus

- Kozen, ~1985; McIver & Morgan and others, 1990s~today
- $wpe : \text{Programs} \times \text{Expectations} \to \text{Expectations}$
- Expectations are maps from program states to numbers
    - Ex. $\mathsf{st} \mapsto 2 \cdot \mathsf{st}[x]$, or $\mathsf{st} \mapsto \mathsf{st}[x] + \mathsf{st}[y]$

Background: Weakest Pre-expectation Calculus

- Kozen, ~1985; McIver & Morgan and others, 1990s~today
- $wpe : \mathrm{Programs} \times \mathrm{Expectations} \to \mathrm{Expectations}$
- Expectations are maps from program states to numbers
    - Ex. $\mathsf{st} \mapsto 2 \cdot \mathsf{st}[x]$, or $\mathsf{st} \mapsto \mathsf{st}[x] + \mathsf{st}[y]$
        - We simply write $2 \cdot x$, or $x + y$

Background: Weakest Pre-expectation Calculus

- Kozen, ~1985; McIver & Morgan and others, 1990s~today
- $wpe : \text{Programs} \times \text{Expectations} \to \text{Expectations}$
- Expectations are maps from program states to numbers
  - Ex. $\mathsf{st} \mapsto 2 \cdot \mathsf{st}[x]$, or $\mathsf{st} \mapsto \mathsf{st}[x] + \mathsf{st}[y]$
    - We simply write $2 \cdot x$, or $x + y$
  - Iverson bracket: $[G]$ maps states where the assertion G holds to 1 and maps other states to 0

Reasoning about Quantities

Reasoning about Quantities

- We let $wpe(C, e)$ be the expected value of expectation e after running C

Reasoning about Quantities

- We let $wpe(C, e)$ be the expected value of expectation e after running C
  1. It generalizes weakest pre-condition calculus, e.g.,

Reasoning about Quantities

- We let $wpe(C, e)$ be the expected value of expectation e after running C

  1. It generalizes weakest pre-condition calculus, e.g.,

     - $wpc(x \leftarrow x + 1, x = y)$ is $x + 1 = y$

Reasoning about Quantities

- We let $wpe(C, e)$ be the expected value of expectation e after running C
  1. It generalizes weakest pre-condition calculus, e.g.,
     - $wpc(x \leftarrow x + 1,\ x = y)$ is $x + 1 = y$
     - $wpe(x \leftarrow x + 1, [x = y])$ is $[x + 1 = y]$

Reasoning about Quantities

- We let $wpe(C, e)$ be the expected value of expectation e after running C

1. It generalizes weakest pre-condition calculus, e.g.,
    - $wpc(x \leftarrow x + 1,\ x = y)$ is $x + 1 = y$
    - $wpe(x \leftarrow x + 1, [x = y])$ is $[x + 1 = y]$
2. Expected values are useful.

Reasoning about Quantities

- We let $wpe(C, e)$ be the expected value of expectation e after running C

1. It generalizes weakest pre-condition calculus, e.g.,
   - $wpc(x \leftarrow x + 1,\ x = y)$ is $x + 1 = y$
   - $wpe(x \leftarrow x + 1, [x = y])$ is $[x + 1 = y]$
2. Expected values are useful.
3. It can encode probability of an event Ev in the output distribution:

Reasoning about Quantities

- We let $wpe(C, e)$ be the expected value of expectation e after running C

1. It generalizes weakest pre-condition calculus, e.g.,
    - $wpc(x \leftarrow x + 1,\ x = y)$ is $x + 1 = y$
    - $wpe(x \leftarrow x + 1, [x = y])$ is $[x + 1 = y]$
2. Expected values are useful.
3. It can encode probability of an event Ev in the output distribution:
    - Let the expectation e be $[Ev]$.

Weakest Pre-expectation Calculus

reason about expected values!

Definition [Morgan and McIver]

Weakest Pre-expectation Calculus

reason about expected values!

Definition [Morgan and McIver]

$$wpe(x \leftarrow a, e) := e[a/x]$$

Weakest Pre-expectation Calculus

reason about expected values!

Definition [Morgan and McIver]

$$wpe(x \leftarrow a, e) := e[a/x]$$

$$wpe(P; Q, e) := wpe(P, wpe(Q, e))$$

Weakest Pre-expectation Calculus

reason about expected values!

Definition [Morgan and McIver]

$$wpe(x \leftarrow a, e) := e[a/x]$$

$$wpe(P; Q, e) := wpe(P, wpe(Q, e))$$

$$wpe(C[p]C', e) := p \cdot wpe(C, e) + (1 - p) \cdot wpe(C', e)$$

Weakest Pre-expectation Calculus

reason about expected values!

Definition [Morgan and McIver]

$$wpe(x \leftarrow a, e) := e[a/x]$$

$$wpe(P; Q, e) := wpe(P, wpe(Q, e))$$

$$wpe(C[p]C', e) := p \cdot wpe(C, e) + (1 - p) \cdot wpe(C', e)$$

$$wpe(\text{while } b \text{ do } body, e) := \mathsf{lfp}(\lambda x\,.\,[b] \cdot wpe(body, x) + [\neg b] \cdot e)$$

Weakest Pre-expectation Calculus

reason about expected values!

Definition [Morgan and McIver]

$$wpe(x \leftarrow a, e) := e[a/x]$$

$$wpe(P; Q, e) := wpe(P, wpe(Q, e))$$

$$wpe(C[p]C', e) := p \cdot wpe(C, e) + (1 - p) \cdot wpe(C', e)$$

$$wpe(\text{while } b \text{ do } body, e) := \mathsf{lfp}(\lambda x . [b] \cdot wpe(body, x) + [\neg b] \cdot e)$$

$$\ldots$$

Weakest Pre-expectation Calculus

reason about expected values!

Definition [Morgan and McIver]

$$wpe(x \leftarrow a, e) := e[a/x]$$

$$wpe(P; Q, e) := wpe(P, wpe(Q, e))$$

$$wpe(C[p]C', e) := p \cdot wpe(C, e) + (1 - p) \cdot wpe(C', e)$$

$$wpe(\text{while } b \text{ do } body, e) := \text{lfp}(\lambda x . [b] \cdot wpe(body, x) + [\neg b] \cdot e)$$

$$\ldots$$

Theorem. $wpe(C, e) = \lambda s\,.$ expected value of e after running C from s

WPE Calculus on Motivating Examples

WPE Calculus on Motivating Examples

Ex. 1

$$wpe(\mathrm{z} \leftarrow \mathrm{z} + 1 \; [\mathrm{p}] \; \mathrm{skip}, \mathrm{z})$$

$$= \mathrm{p} \cdot wpe(\mathrm{z} \leftarrow \mathrm{z} + 1, \mathrm{z}) + (1 - \mathrm{p}) \cdot wpe(\mathrm{skip}, \mathrm{z})$$

$$= \mathrm{p} \cdot (\mathrm{z} + 1) + (1 - \mathrm{p}) \cdot \mathrm{z}$$

$$= \mathrm{z} + \mathrm{p}$$

WPE Calculus on Motivating Examples

Ex. 1

$wpe(\mathrm{z} \leftarrow \mathrm{z} + 1 \; [\mathrm{p}] \; \mathrm{skip}, \mathrm{z})$

$= \mathrm{p} \cdot wpe(\mathrm{z} \leftarrow \mathrm{z} + 1, \mathrm{z}) + (1 - \mathrm{p}) \cdot wpe(\mathrm{skip}, \mathrm{z})$

$= \mathrm{p} \cdot (\mathrm{z} + 1) + (1 - \mathrm{p}) \cdot \mathrm{z}$

$= \mathrm{z} + \mathrm{p}$

Ex. 2

$wpe(\mathbf{while}\ (\mathrm{flip} == 0)\ \mathbf{do}\ (\mathrm{flip} \leftarrow 1 \; [\mathrm{p}] \; \mathrm{z} \leftarrow \mathrm{z} + 1), \mathrm{z})$?

WPE Calculus on Motivating Examples

Ex. 1

$wpe(\mathrm{z} \leftarrow \mathrm{z} + 1 \; [\mathrm{p}] \; \mathrm{skip}, \mathrm{z})$

$= \mathrm{p} \cdot wpe(\mathrm{z} \leftarrow \mathrm{z} + 1, \mathrm{z}) + (1 - \mathrm{p}) \cdot wpe(\mathrm{skip}, \mathrm{z})$

$= \mathrm{p} \cdot (\mathrm{z} + 1) + (1 - \mathrm{p}) \cdot \mathrm{z}$

$= \mathrm{z} + \mathrm{p}$

$wpe(\text{while } b \text{ do } body, e) := \mathrm{lfp}(\lambda x \,.\, [b] \cdot wpe(body, x) + [\neg b] \cdot e)$

Ex. 2

$wpe(\text{while } (\mathrm{flip} == 0) \text{ do } (\mathrm{flip} \leftarrow 1 \; [\mathrm{p}] \; \mathrm{z} \leftarrow \mathrm{z} + 1), \mathrm{z})$?

WPE Calculus on Motivating Examples

Ex. 1

$wpe(z \leftarrow z + 1\ [p]\ \text{skip}, z)$

$= p \cdot wpe(z \leftarrow z + 1, z) + (1 - p) \cdot wpe(\text{skip}, z)$

$= p \cdot (z + 1) + (1 - p) \cdot z$

$= z + p$

The least fixed point is indirect and hard to work with!

$wpe(\text{while } b \text{ do } body, e) := \text{lfp}(\lambda x \,.\, [b] \cdot wpe(body, x) + [\neg b] \cdot e)$

Ex. 2

$wpe(\text{while } (\text{flip} == 0) \text{ do } (\text{flip} \leftarrow 1\ [p]\ z \leftarrow z + 1), z)?$

WPE Calculus on Motivating Examples

Ex. 1

$wpe(\text{z} \leftarrow \text{z} + 1\ [\text{p}]\ \text{skip}, \text{z})$

$= \text{p} \cdot wpe(\text{z} \leftarrow \text{z} + 1, \text{z}) + (1 - \text{p}) \cdot wpe(\text{skip}, \text{z})$

$= \text{p} \cdot (\text{z} + 1) + (1 - \text{p}) \cdot \text{z}$

$= \text{z} + \text{p}$

The least fixed point is indirect and hard to work with!

$$wpe(\text{while } b \text{ do } body, e) := \text{lfp}\underbrace{(\lambda x\,.\,[b] \cdot wpe(body, x) + [\neg b] \cdot e)}_{\phi}$$

Ex. 2

$wpe(\text{while } (\text{flip} == 0) \text{ do } (\text{flip} \leftarrow 1\ [\text{p}]\ \text{z} \leftarrow \text{z} + 1), \text{z})$?

Problem Statement

Problem Statement

- **Given: a loop while G do P and an expectation e.**

Problem Statement

- **Given:** a loop **while** G **do** P and an expectation e.
- **Goal:** find expectation I such that $I = [G] \cdot wpe(P, I) + [\neg G] \cdot e$.

Problem Statement

- **Given:** **a loop while G do P and an expectation e.**
- **Goal:** find expectation I such that $I = [G] \cdot wpe(P, I) + [\neg G] \cdot e$.
- We call e the postexpectation, and call I an exact invariant of the loop.

Why find exact invariant?

Why find exact invariant?

- For <u>simple</u> cases, there is unique fixed point for ϕ, so I is a fixed point of ϕ iff I is the least fixed point.

Why find exact invariant?

- For simple cases, there is unique fixed point for ϕ, so I is a fixed point of ϕ iff I is the least fixed point.
- In other words, in simple cases, I is an exact invariant iff $I = wpe(\text{while } G \text{ do } P, e)$.

Why find exact invariant?

- For simple cases, there is unique fixed point for ϕ, so I is a fixed point of ϕ iff I is the least fixed point.
- In other words, in simple cases, I is an exact invariant iff $I = wpe(\text{while } G \text{ do } P, e)$.
- Simple: the loop is almost surely terminating and e is upper bounded by a constant

Casting into a Learning Problem

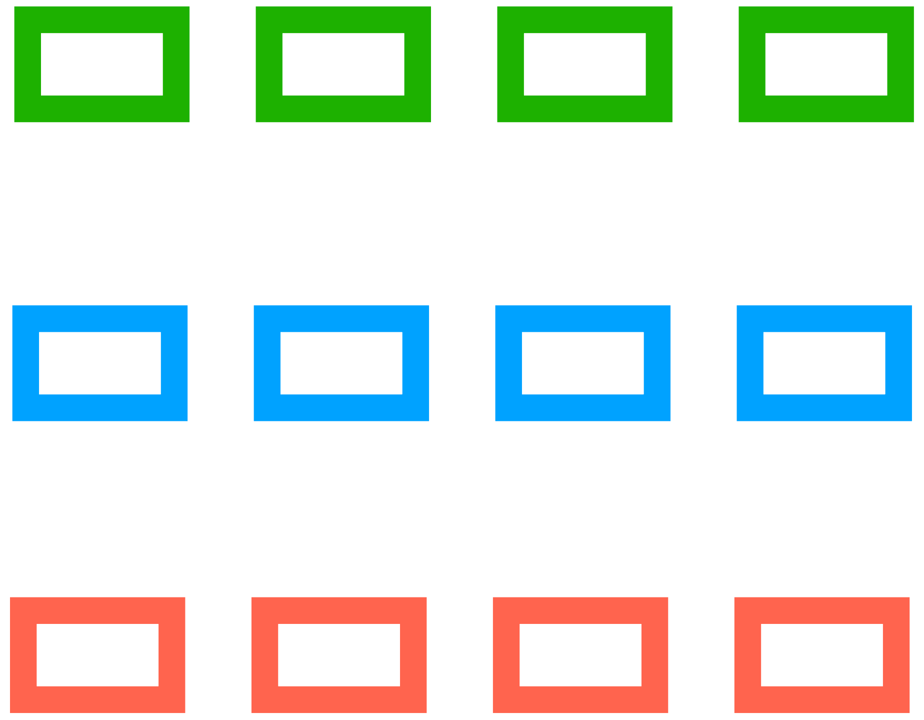
Casting into a Learning Problem

- $I = wpe(\text{while } G \text{ do } P, e)$ is a map from program states to numbers

Casting into a Learning Problem

- $I = wpe(\text{while } G \text{ do } P, e)$ is a map from program states to numbers

Program states | **Numbers**



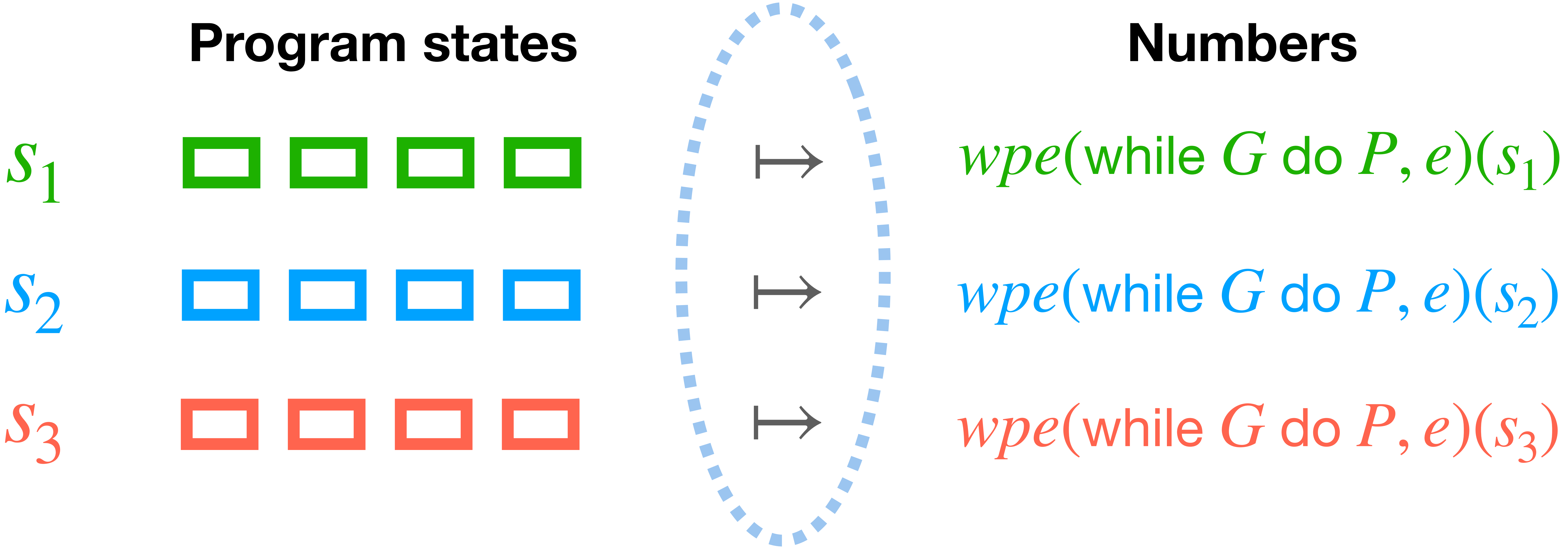
$s_1 \mapsto wpe(\text{while } G \text{ do } P, e)(s_1)$

$s_2 \mapsto wpe(\text{while } G \text{ do } P, e)(s_2)$

$s_3 \mapsto wpe(\text{while } G \text{ do } P, e)(s_3)$

Casting into a Learning Problem

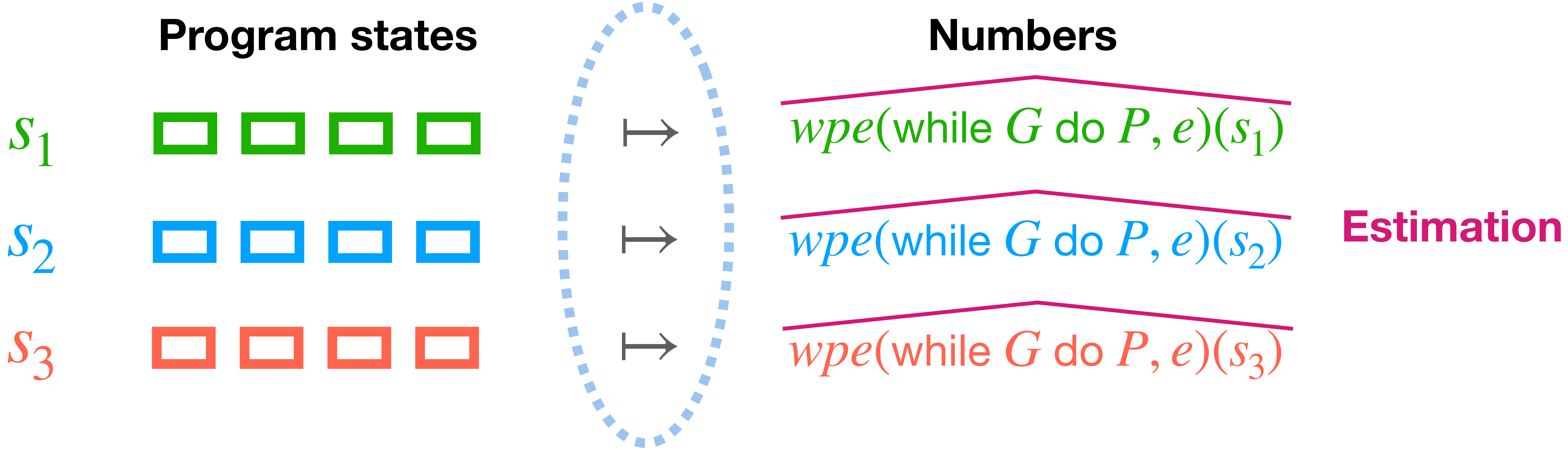
- $I = wpe(\text{while } G \text{ do } P, e)$ is a map from program states to numbers



If we can collect data to learn this map, it coincides with $wpe(\text{while } G \text{ do } P, e)$ on sampled program states

Casting into a Learning Problem

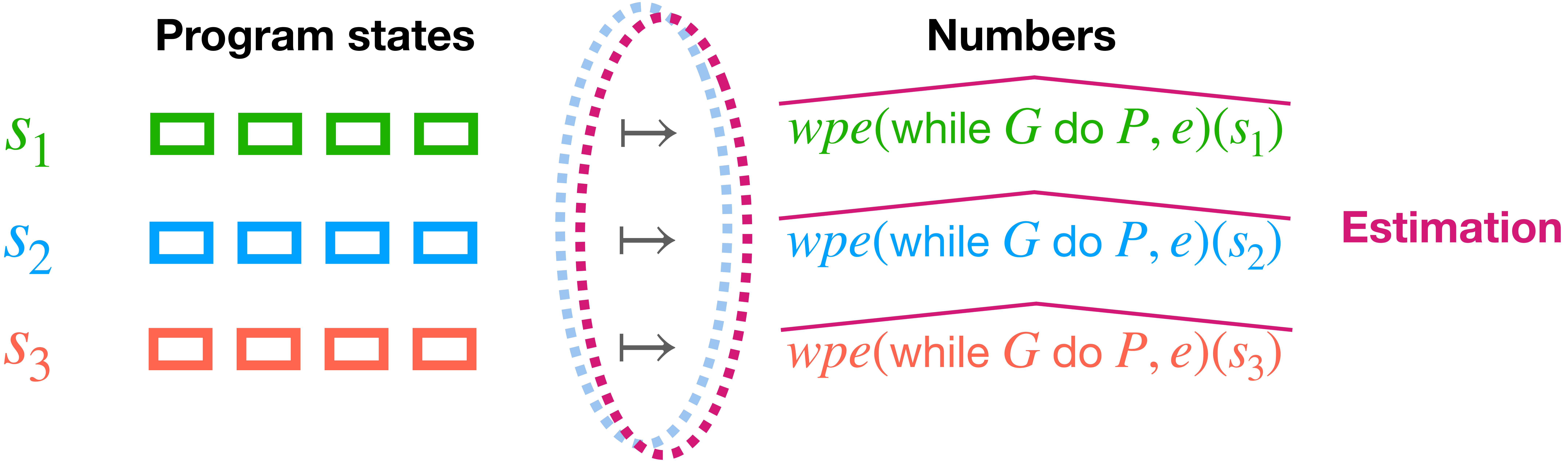
- $I = wpe(\text{while } G \text{ do } P, e)$ is a map from program states to numbers



If we can collect data to learn this map, it coincides with $wpe(\text{while } G \text{ do } P, e)$ on sampled program states

Casting into a Learning Problem

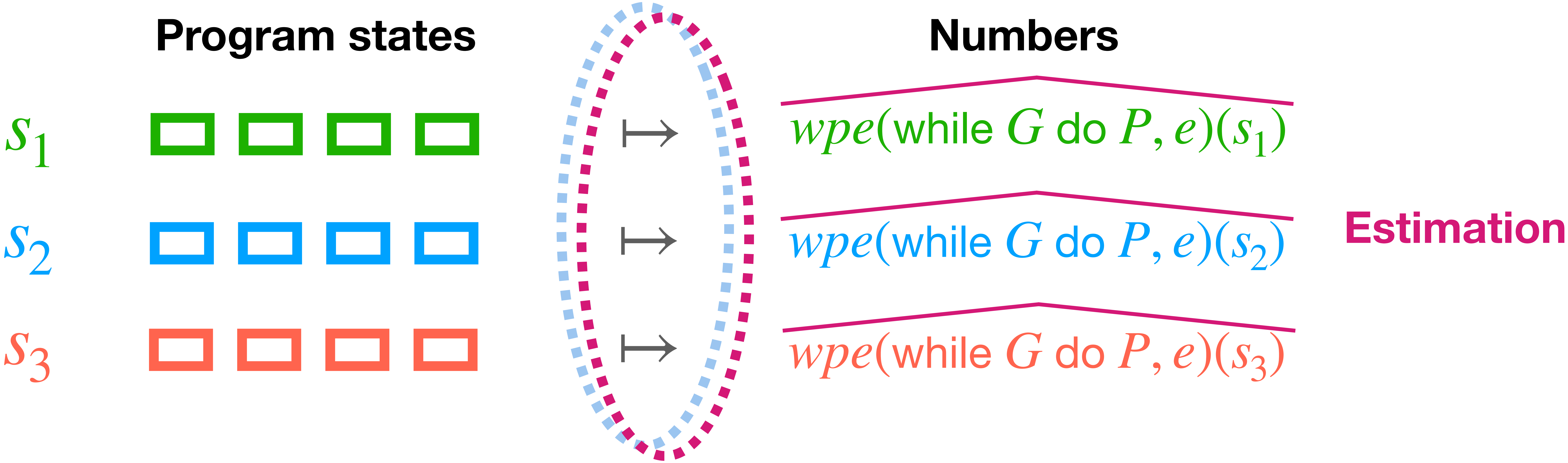
- $I = wpe(\text{while } G \text{ do } P, e)$ is a map from program states to numbers



If we learn this map, it approximates $wpe(\text{while } G \text{ do } P, e)$ on sampled program states

Casting into a Learning Problem

- $I = wpe(\text{while } G \text{ do } P, e)$ is a map from program states to numbers



If we learn this map, it approximates $wpe(\text{while } G \text{ do } P, e)$ on sampled program states

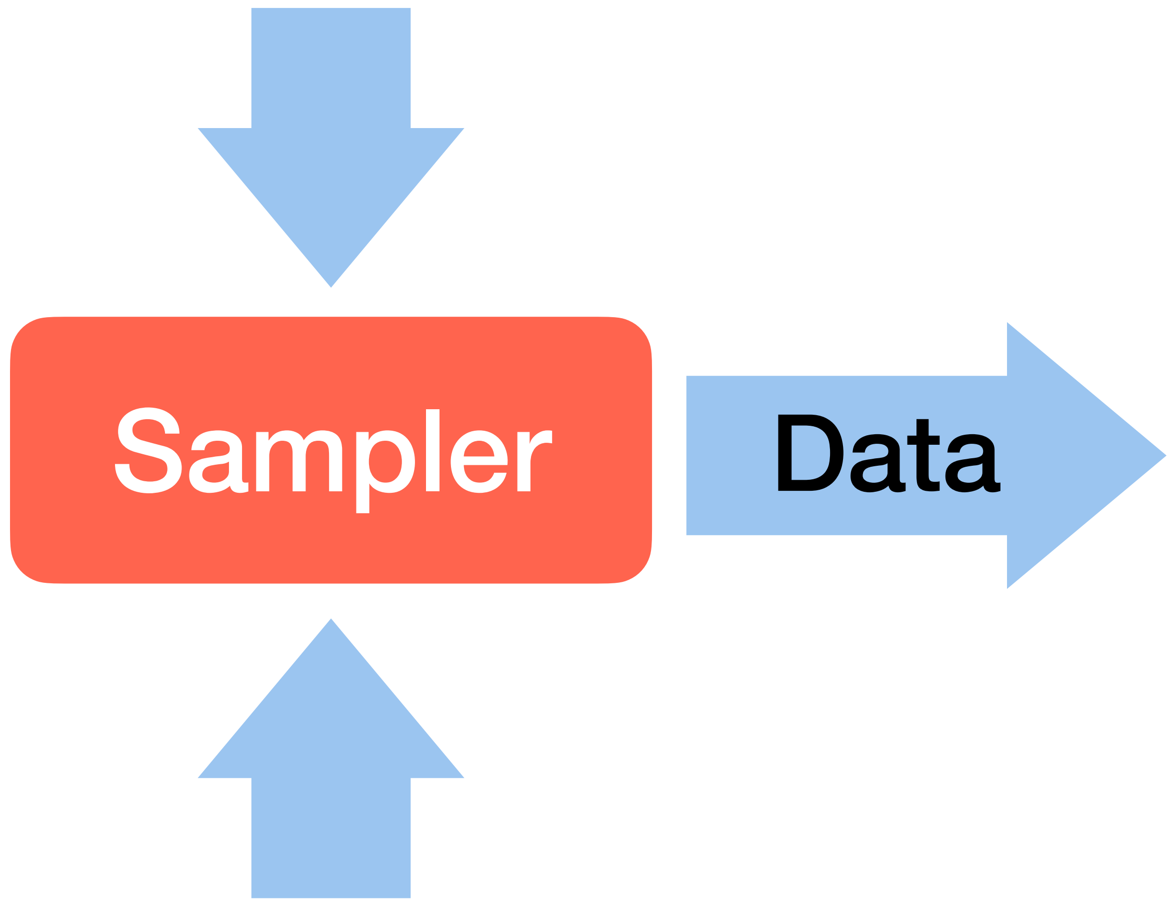
The learned map may not be $wpe(\text{while } G \text{ do } P, e)$ but we can check whether it is an exact invariant.

Method Overview

Method Overview

Estimate the map

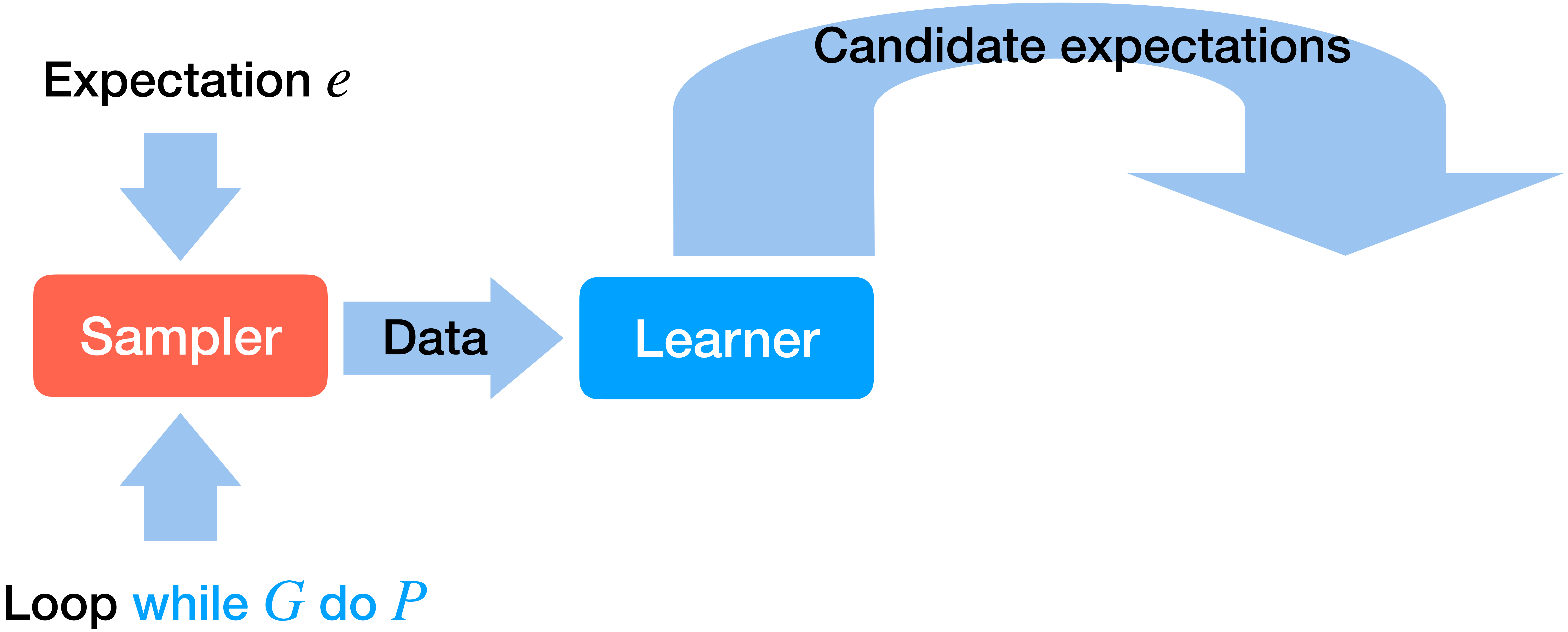
Expectation e



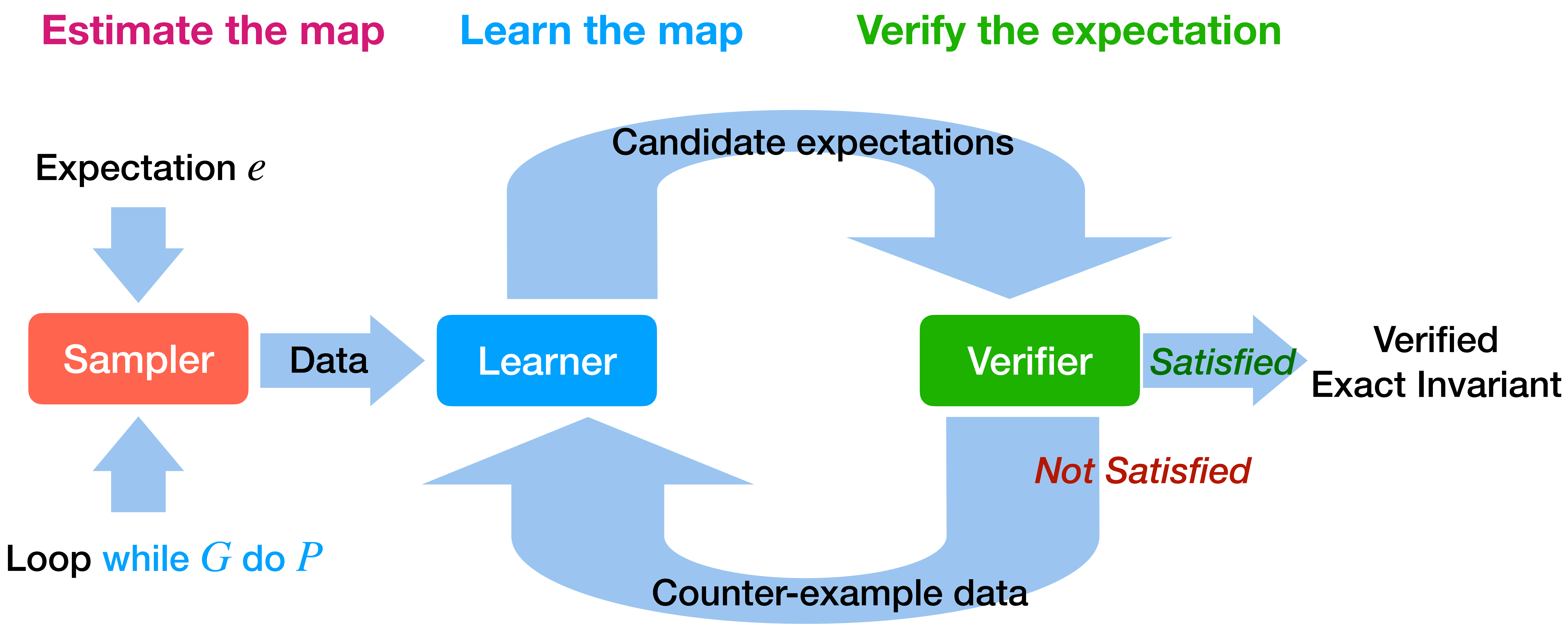
Loop while G do P

Method Overview

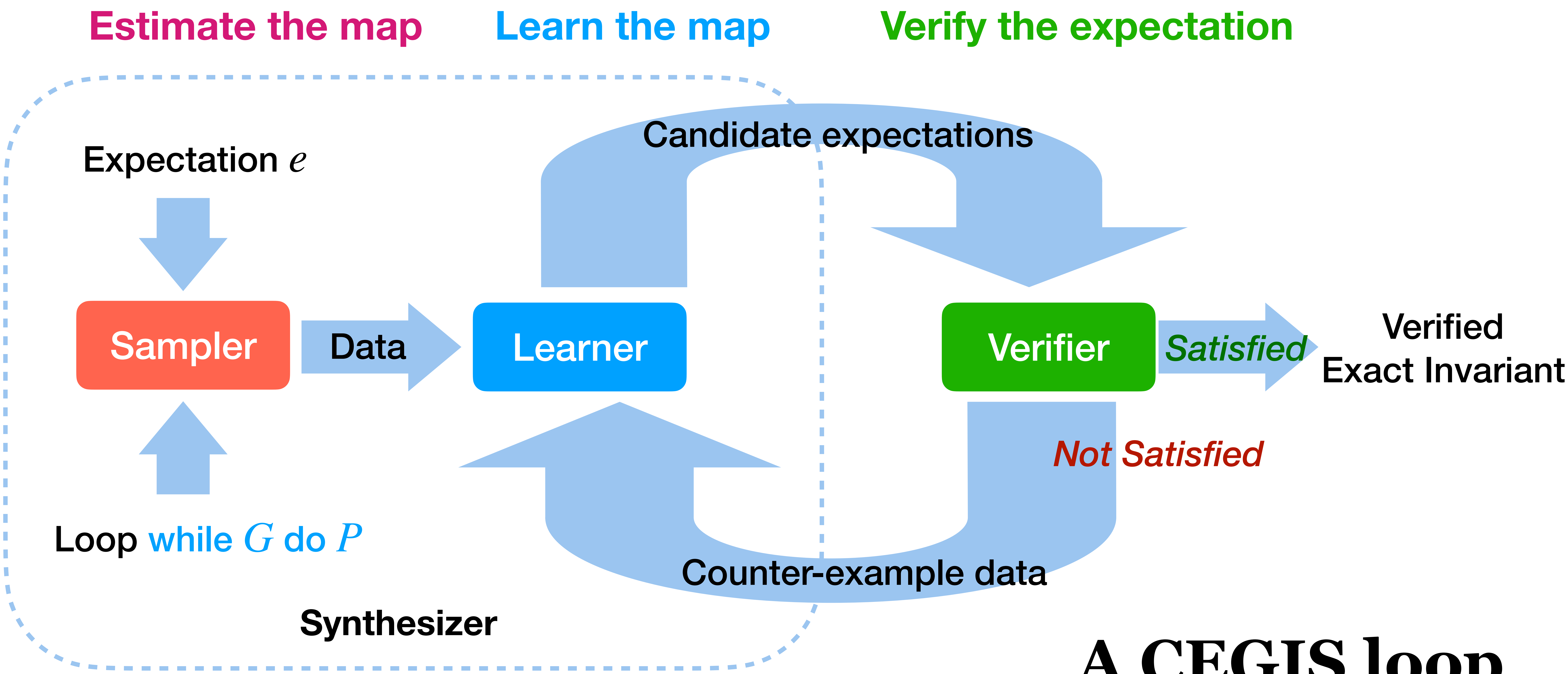
Estimate the map   Learn the map



Method Overview



Method Overview



A CEGIS loop

Sampler

Sampler

- **How to estimate $wpe(\text{while } G \text{ do } P, e)(s)$?**
  - It is the expected value of e on the distribution obtained from running while G do P from s.
  - We can approximate expected values by empirical means.

Sampler: Sample Program States

Sampler: Sample Program States

- To sample a program state, we sample each program variable’s value from a range:

Sampler: Sample Program States

- To sample a program state, we sample each program variable's value from a range:
  - bool: uniformly from $\{0,1\}$

Sampler: Sample Program States

- To sample a program state, we sample each program variable's value from a range:
  - bool: uniformly from $\{0,1\}$
  - int: uniformly from $\{0,1,\ldots,20\}$

Sampler: Sample Program States

- To sample a program state, we sample each program variable's value from a range:
  - bool: uniformly from $\{0,1\}$
  - int: uniformly from $\{0,1,\ldots,20\}$
  - prob: uniformly from $[0.01,0.99]$

Sampler: Sample Program States

- To sample a program state, we sample each program variable's value from a range:
  - bool: uniformly from $\{0,1\}$
  - int: uniformly from $\{0,1,\ldots,20\}$
  - prob: uniformly from $[0.01,0.99]$
  - …

Sampler: Sample Program States

- To sample a program state, we sample each program variable's value from a range:
  - bool: uniformly from $\{0,1\}$
  - int: uniformly from $\{0,1,\ldots,20\}$
  - prob: uniformly from $[0.01,0.99]$
  - $\ldots$
- Sample M program states in total

Sampler: Estimate wpe on program states

Sampler: Estimate wpe on program states

- Generate a list of features

Sampler: Estimate wpe on program states

- Generate a list of features
- For each sampled program state s, run the loop for N times and record the postexpectation's value when the loop exits

Sampler: Estimate wpe on program states

- Generate a list of features
- For each sampled program state s, run the loop for N times and record the postexpectation's value when the loop exits

Feature

N

initial p	initial z	initial flip	final z
0.4	0	0	1
0.4	0	0	2
0.4	0	0	0
0.4	0	0	3
0.5	1	0	2
0.5	1	0	3
0.5	1	0	1
0.5	1	0	2
...	…	…	…

Sampler: Estimate *wpe* on program states

- Generate a list of features
- For each sampled program state s, run the loop for N times and record the postexpectation's value when the loop exits

Feature

N

initial p	initial z	initial flip	final z
0.4	0	0	1
0.4	0	0	2
0.4	0	0	0
0.4	0	0	3
0.5	1	0	2
0.5	1	0	3
0.5	1	0	1
0.5	1	0	2
...	…	…	…

Averaging

initial p	initial z	initial flip	Average final z
0.4	0	0	1.5
0.5	1	0	2
…			

Sampler: Estimate *wpe* on program states

- Generate a list of features
- For each sampled program state s, run the loop for N times and record the postexpectation's value when the loop exits

Feature

N

initial p	initial z	initial flip	final z
0.4	0	0	1
0.4	0	0	2
0.4	0	0	0
0.4	0	0	3
0.5	1	0	2
0.5	1	0	3
0.5	1	0	1
0.5	1	0	2
...	...	...	...

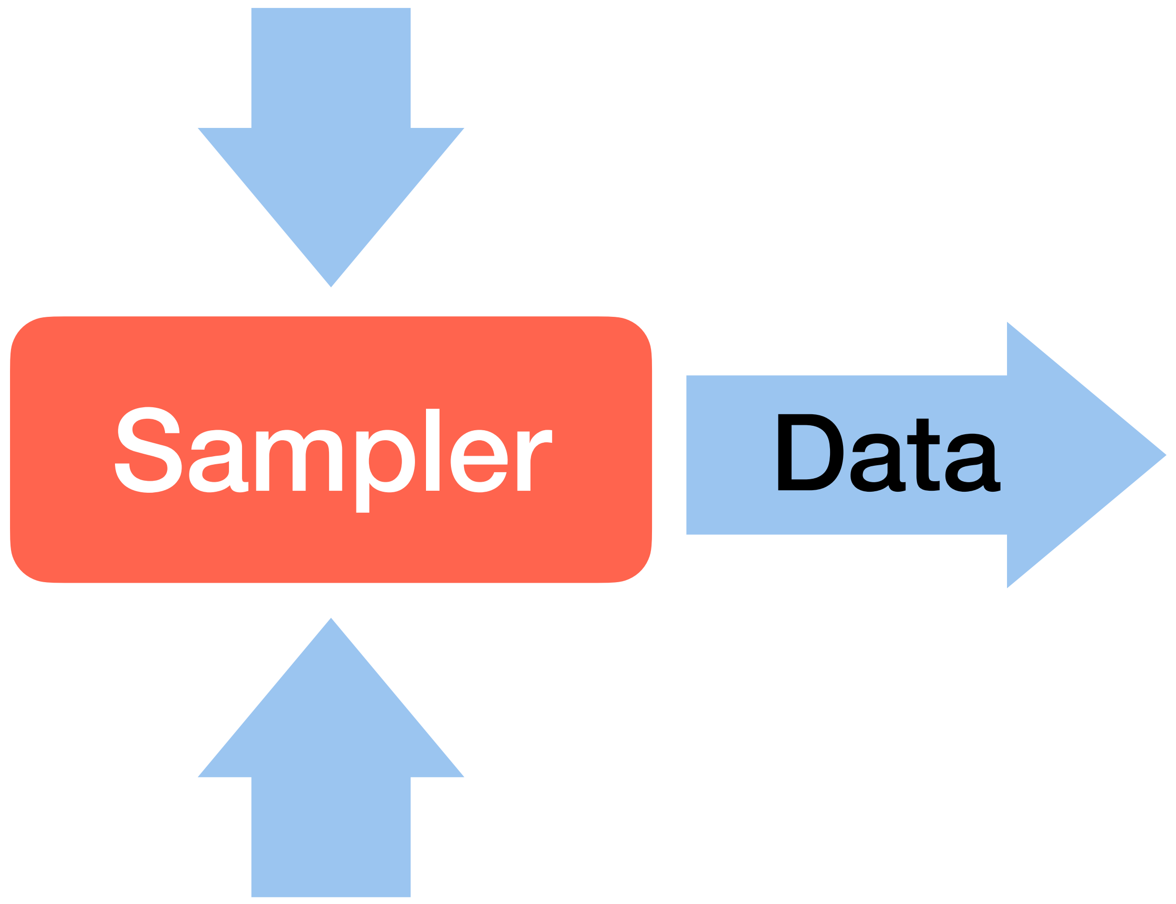
Averaging

initial p	initial z	initial flip	Average final z
0.4	0	0	1.5
0.5	1	0	2
...			

$wpe(\text{while } flip = 0 \text{ do } \ldots, z)(s)$

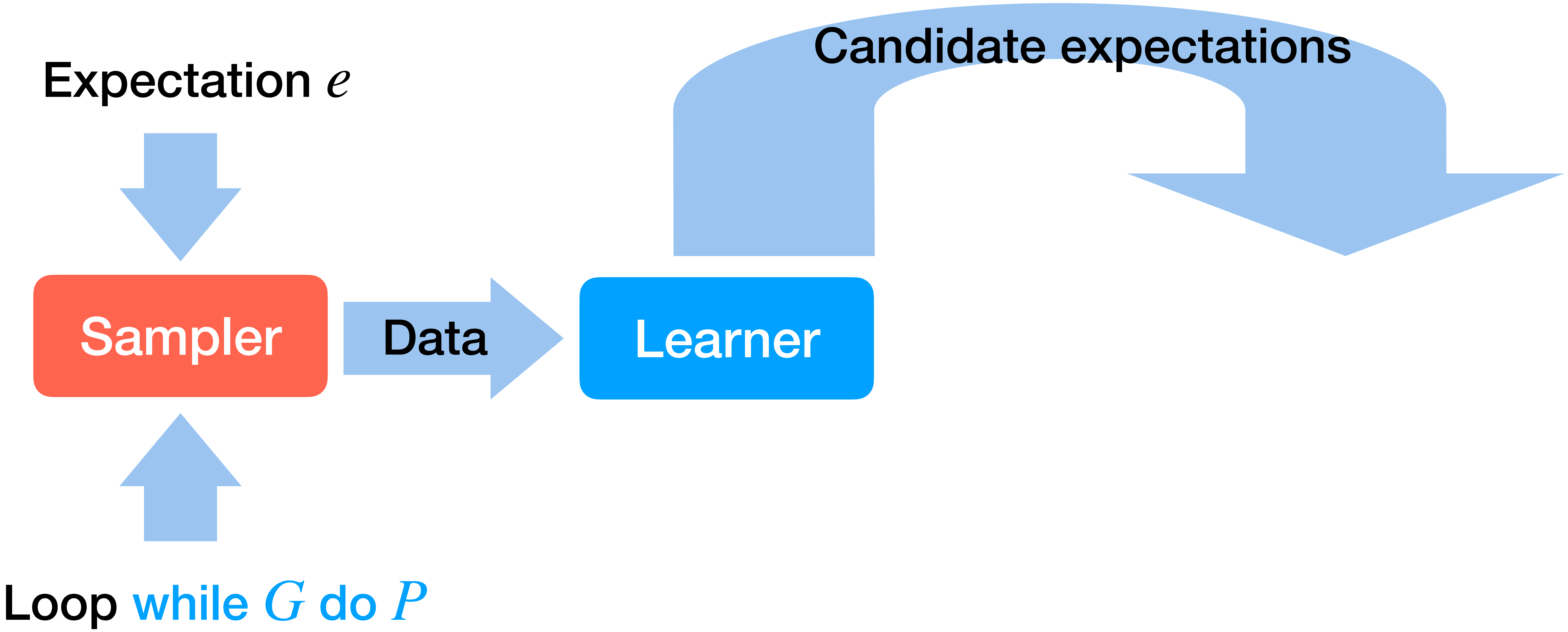
Estimate the map

Expectation e

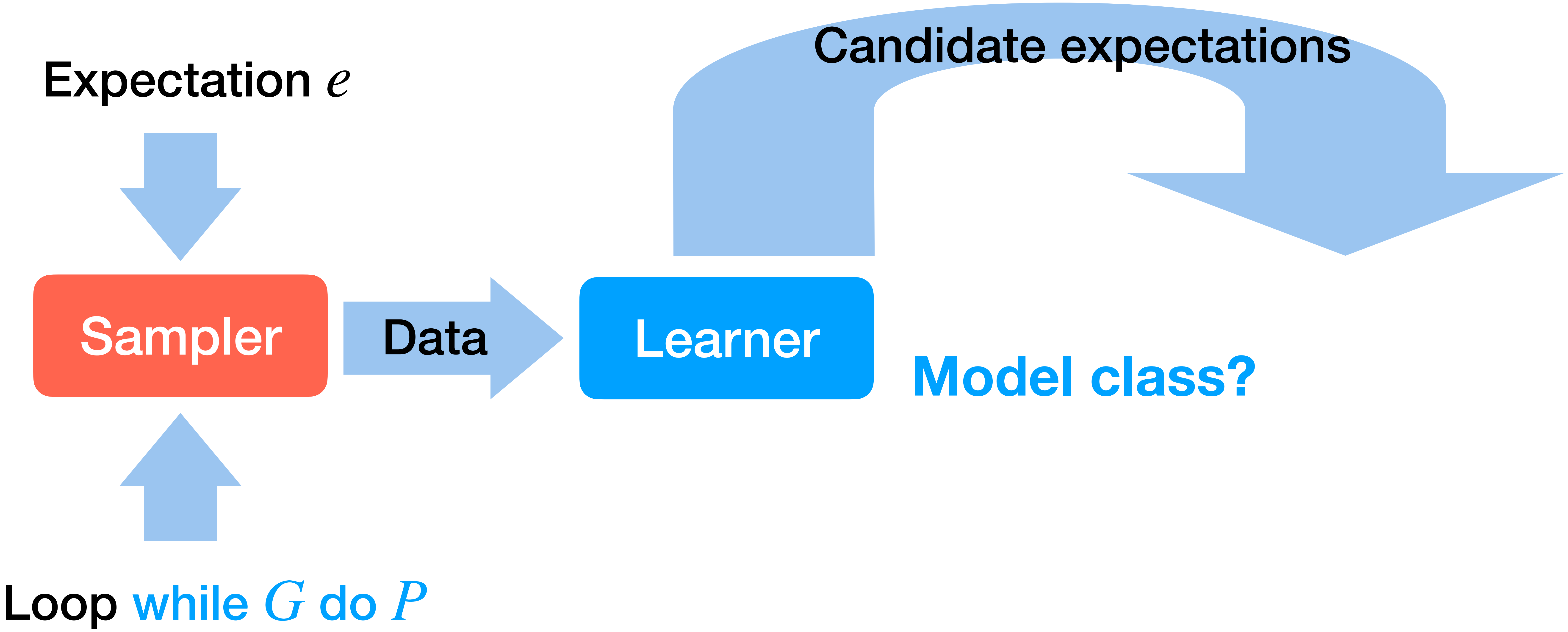


Loop while G do P

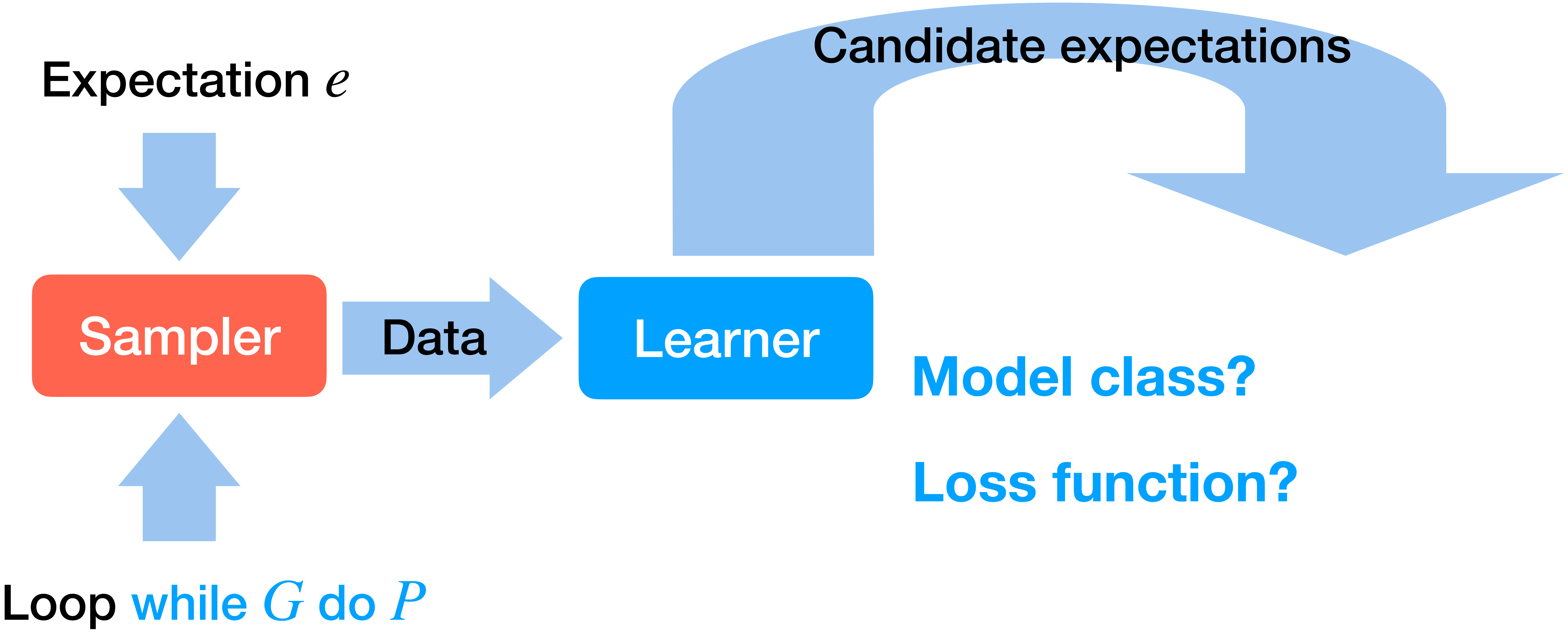
Estimate the map **Learn the map**



Estimate the map **Learn the map**



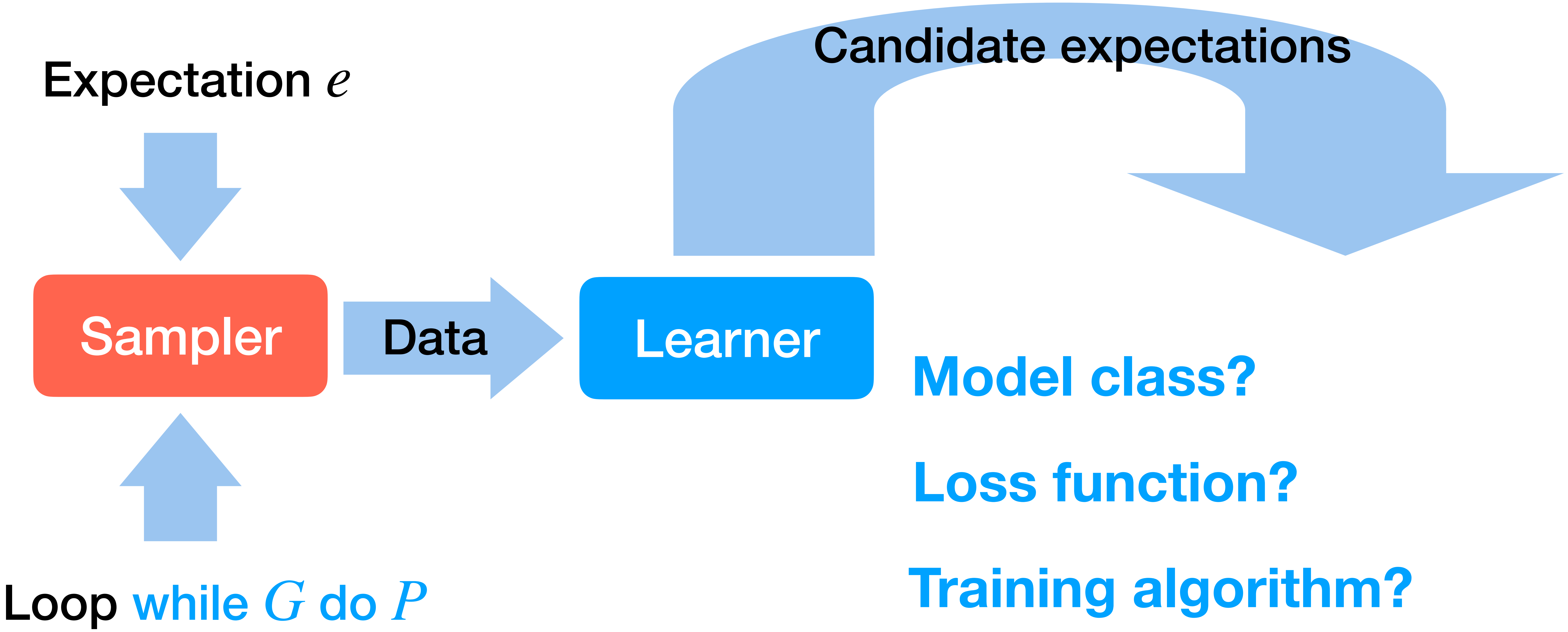
Estimate the map **Learn the map**



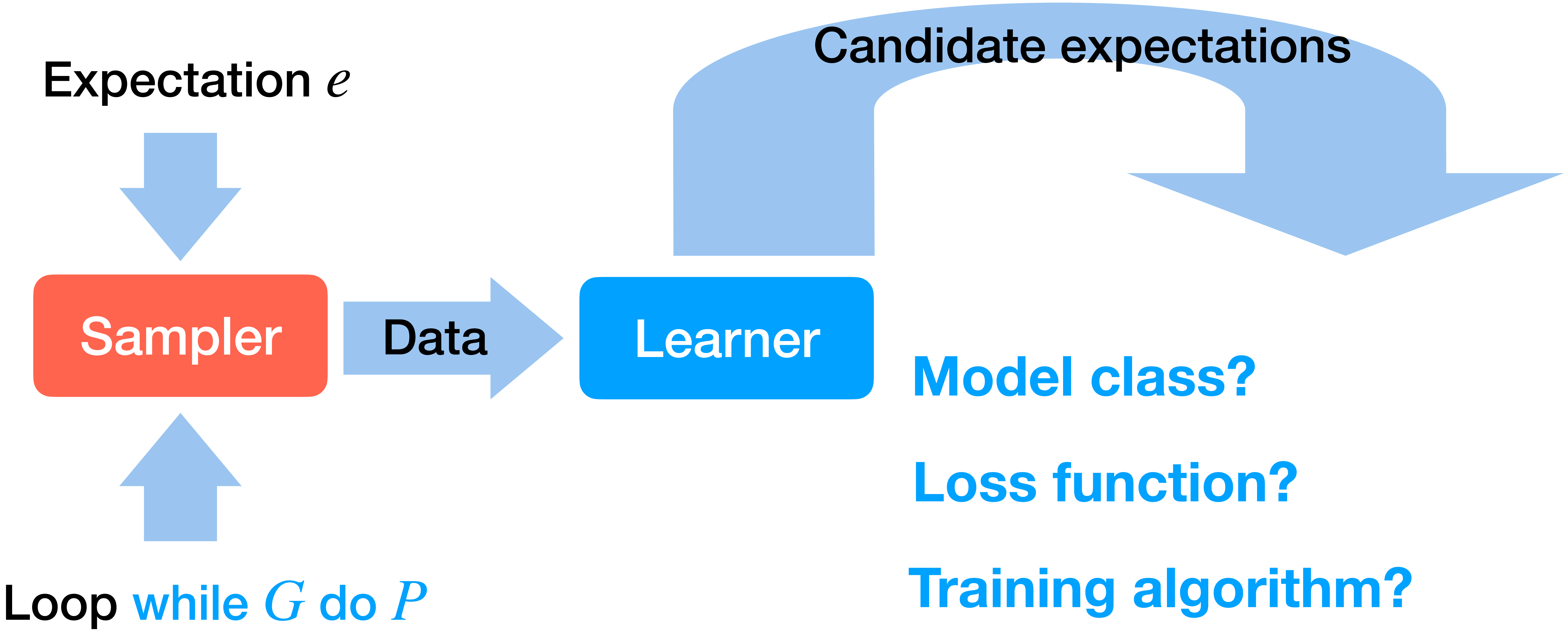
Model class?

Loss function?

Estimate the map **Learn the map**



Estimate the map **Learn the map**



Model class?

Loss function?

Training algorithm?

How to formulate the expectations?

Learner: Model Class

Learner: Model Class

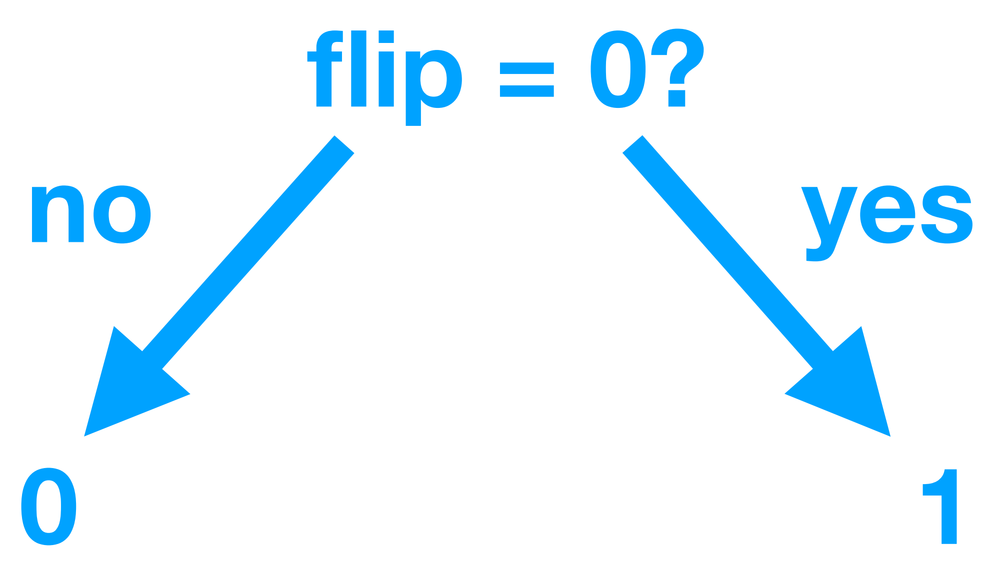
- Model trees generalize decision trees with a model at each leaf.

Learner: Model Class

- Model trees generalize decision trees with a model at each leaf.
  - Ex.

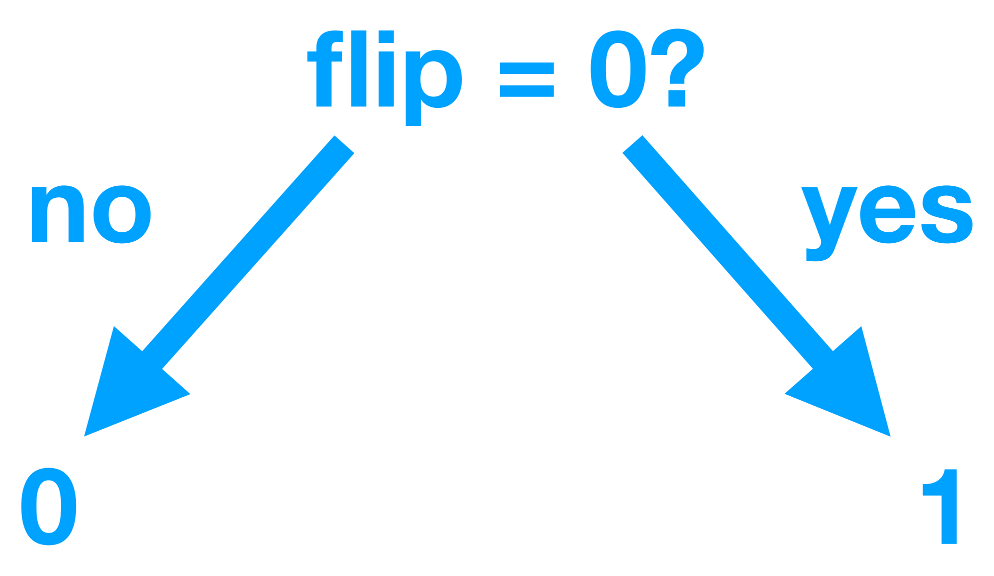
Learner: Model Class

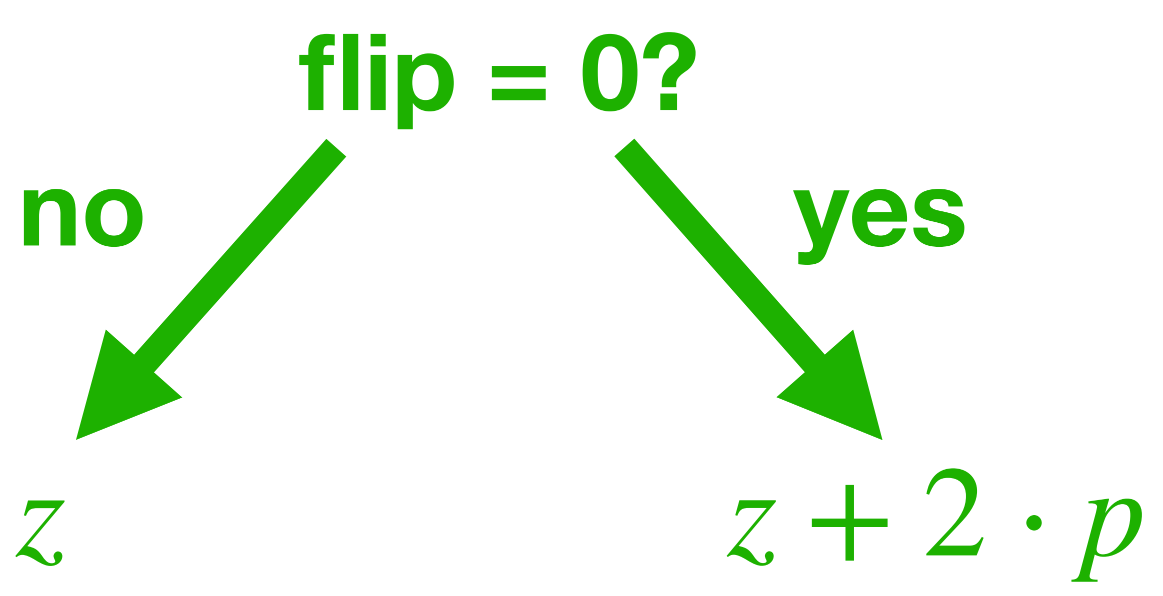
- Model trees generalize decision trees with a model at each leaf.
  - Ex.



Learner: Model Class

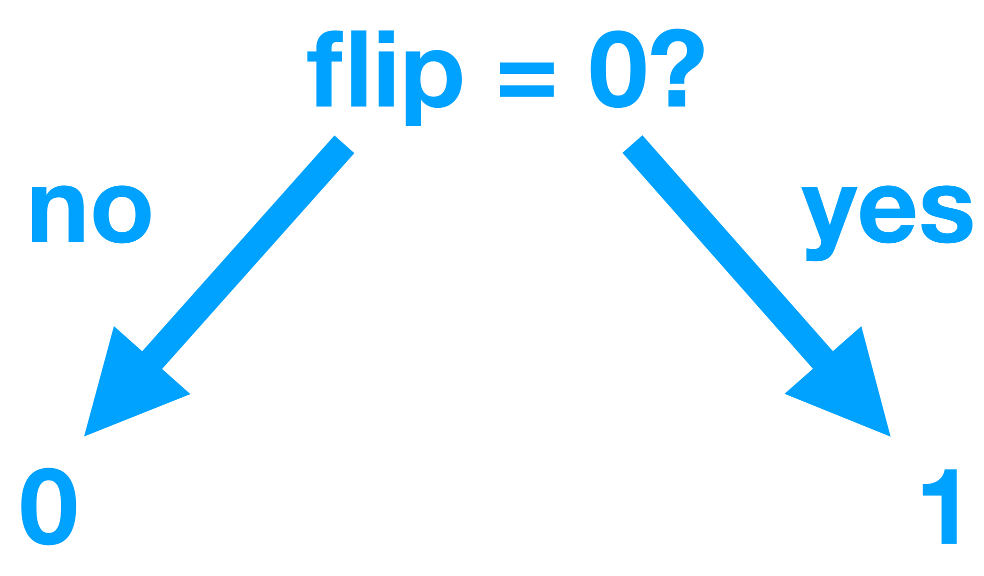
- Model trees generalize decision trees with a model at each leaf.
  - Ex.

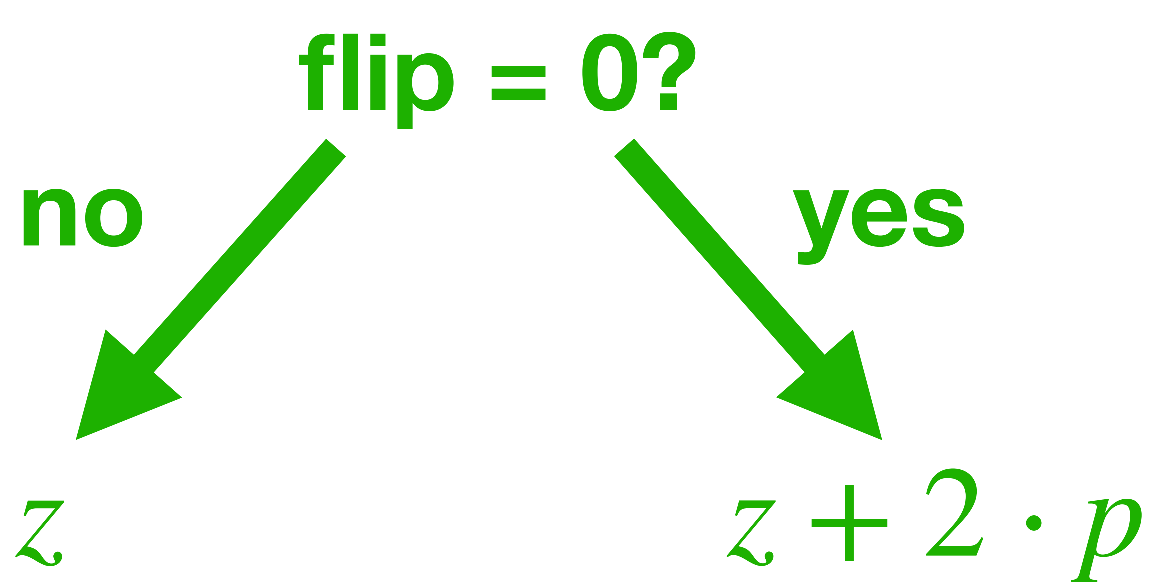




Learner: Model Class

- Model trees generalize decision trees with a model at each leaf.
  - Ex.





- Leaf models: linear functions or cumulative products of features

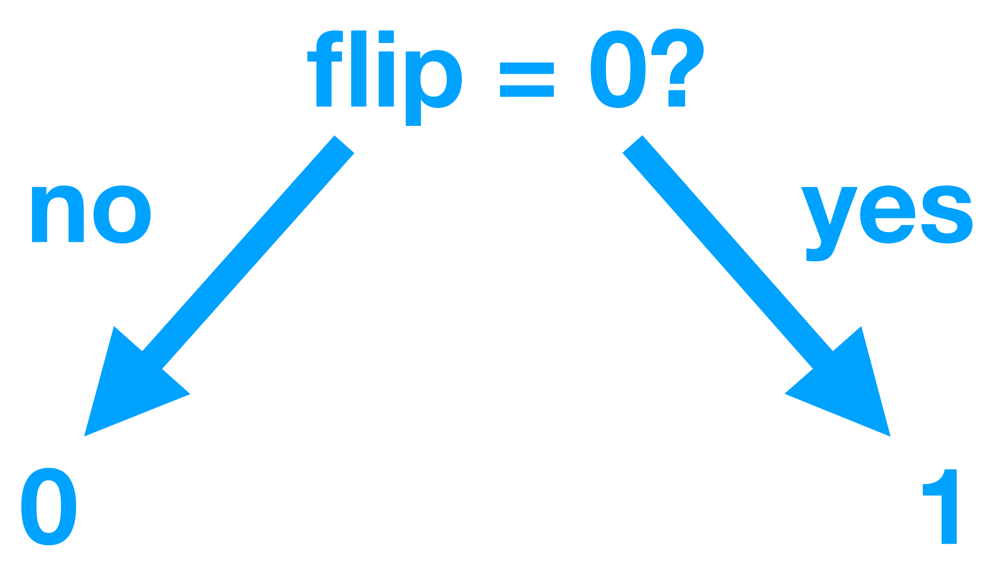
Learner: Model Class

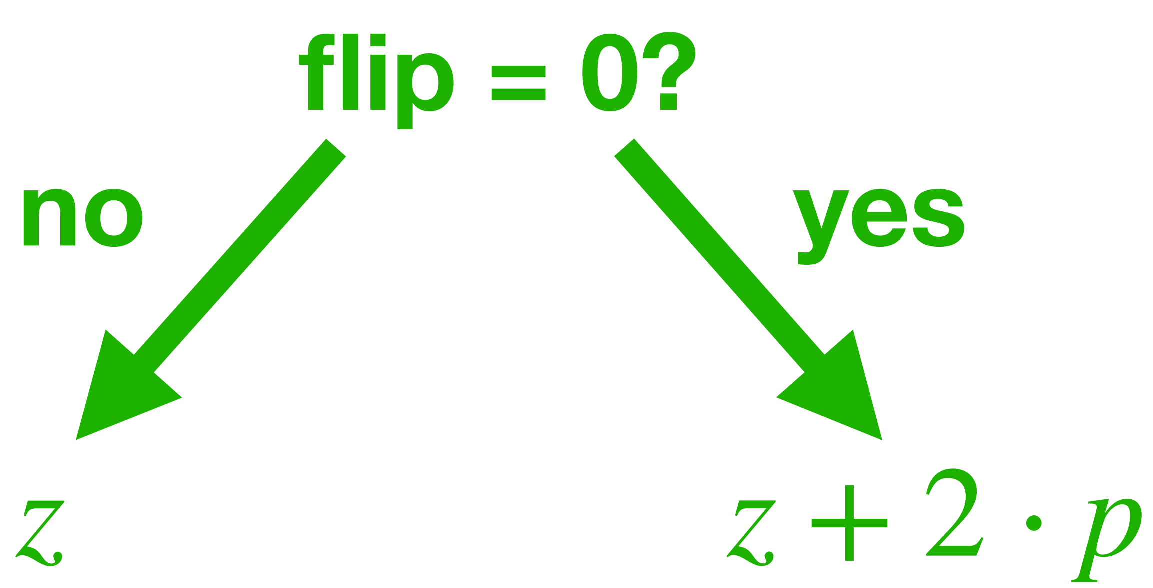
- Model trees generalize decision trees with a model at each leaf.
  - Ex.

- Leaf models: linear functions or cumulative products of features
- Advantages:

Learner: Model Class

- Model trees generalize decision trees with a model at each leaf.
  - Ex.

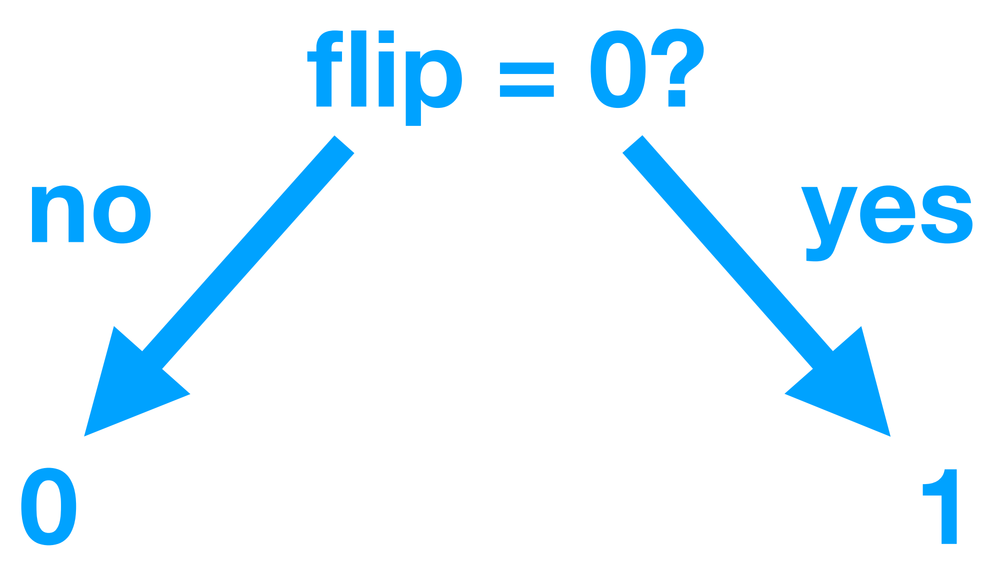


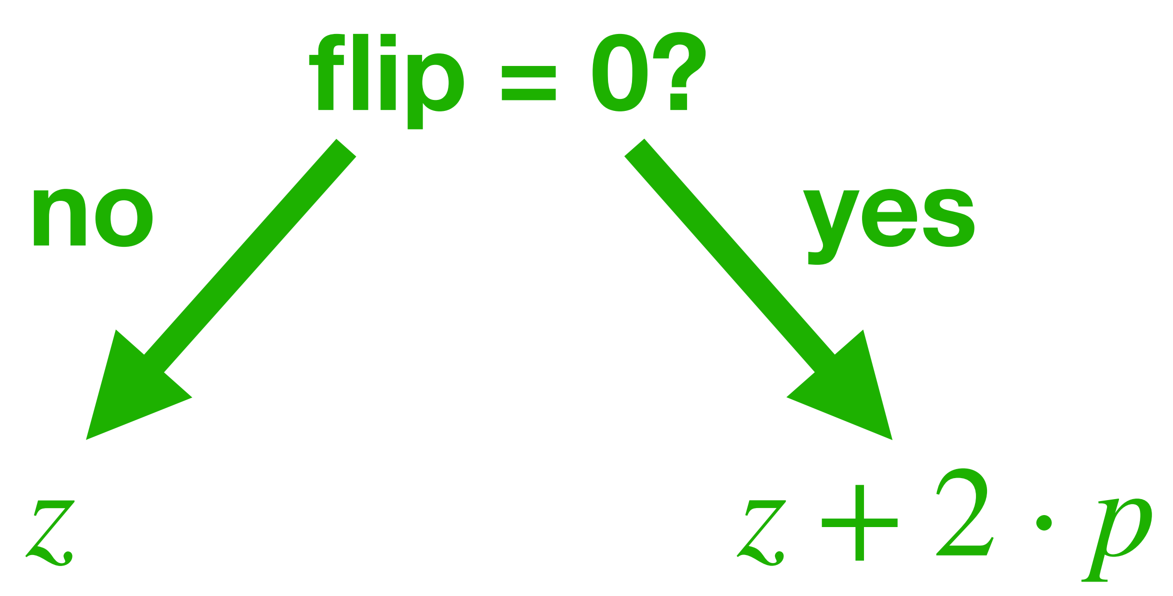


- Leaf models: linear functions or cumulative products of features
- Advantages:
  - Easy for human to interpret

Learner: Model Class

- Model trees generalize decision trees with a model at each leaf.
  - Ex.





- Leaf models: linear functions or cumulative products of features
- Advantages:
  - Easy for human to interpret
  - Easy for verifier to manipulate

Learner: Loss and Training

Learner: Loss and Training

- **Loss** of a model tree T on collected data D

$$\text{Loss}(T, D) = \sum_{(f,v) \in D} (T(f) - v)^2$$

Learner: Loss and Training

- **Loss of a model tree T on collected data D**

$$\text{Loss}(T, D) = \sum_{(f,v) \in D} (T(f) - v)^2$$

initial p	initial z	initial flip	Average post z
0.4	0	0	1.5
0.5	1	0	2
…			

f : feature vector on s

v : $\overline{wpe(\textbf{while}\ flip = 0\ \textbf{do}\ \ldots, z)(s)}$

Learner: Loss and Training

- **Loss of a model tree T on collected data D**

$$\text{Loss}(T, D) = \sum_{(f,v) \in D} (T(f) - v)^2$$

- Fit a model tree T to minimize the loss

initial p	initial z	initial flip	Average post z
0.4	0	0	1.5
0.5	1	0	2
…			

f : feature vector on s v : $wpe($**while** $flip = 0$ **do** $\ldots, z)(s)$

Learner: Loss and Training

- **Loss of a model tree T on collected data D**

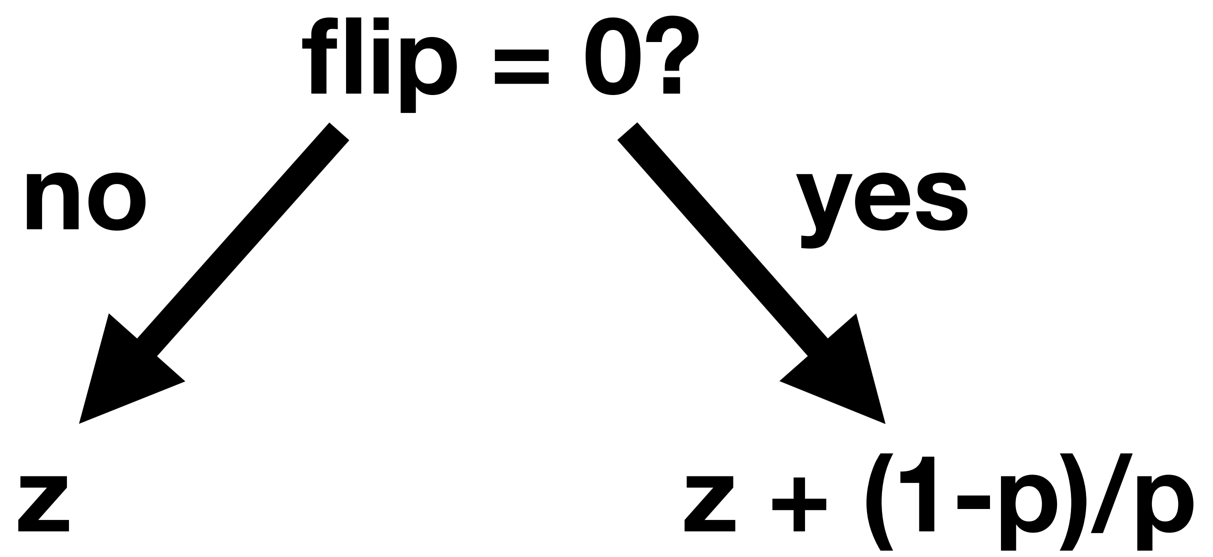
$$\text{Loss}(T, D) = \sum_{(f,v) \in D} (T(f) - v)^2$$

- Fit a model tree T to minimize the loss

initial p	initial z	initial flip	Average post z
0.4	0	0	1.5
0.5	1	0	2
…			

f : feature vector on s $\quad v : wpe(\textbf{while } flip = 0 \textbf{ do } \ldots, z)(s)$

Off-the-shelf model tree learning



Learner: Extract Expectations

Learner: Extract Expectations

- A view change

Learner: Extract Expectations

- A view change
  - Model Tree T: $\mathit{Features} \rightarrow \mathit{Numbers}$

Learner: Extract Expectations

- A view change
  - Model Tree T: $\mathit{Features} \rightarrow \mathit{Numbers}$
  - Expectations: $\mathit{Program\ States} \rightarrow \mathit{Numbers}$

Learner: Extract Expectations

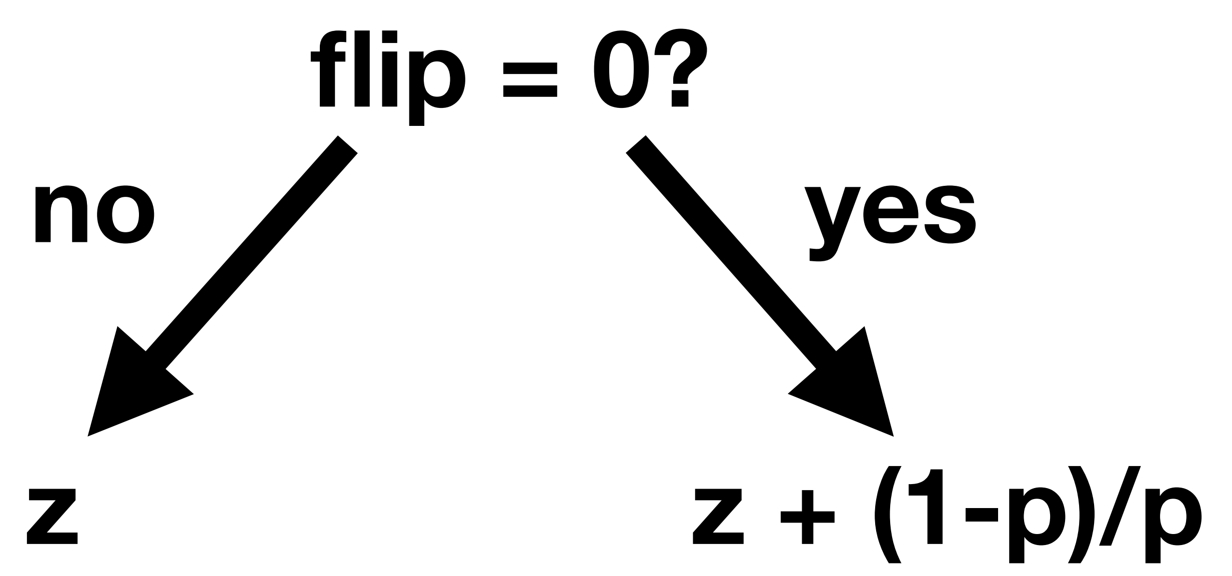
- A view change
  - Model Tree T: $\mathit{Features} \to \mathit{Numbers}$
  - Expectations: $\mathit{Program\ States} \to \mathit{Numbers}$
  - Let $M : \mathit{Program\ States} \to \mathit{Features}$, then $T \circ M$ is an expectation

Learner: Extract Expectations

- A view change
  - Model Tree T: $\mathit{Features} \rightarrow \mathit{Numbers}$
  - Expectations: $\mathit{Program\ States} \rightarrow \mathit{Numbers}$
  - Let $M : \mathit{Program\ States} \rightarrow \mathit{Features},$ then $T \circ M$ is an expectation
- Represent as piece-wise mathematical expression

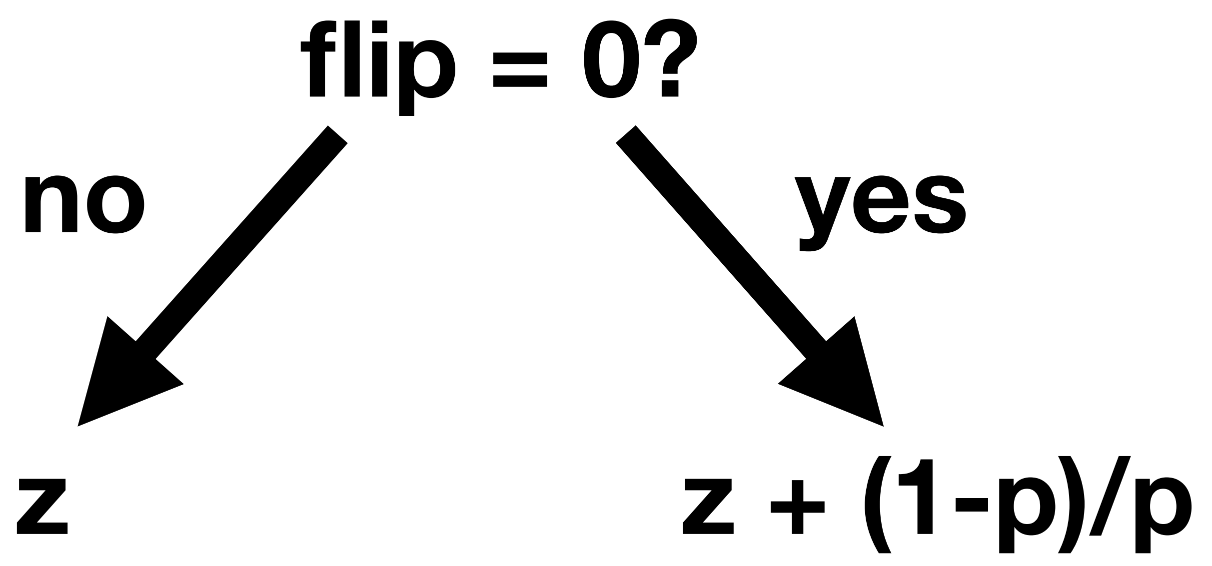
Learner: Extract Expectations

- A view change
  - Model Tree T: $\mathit{Features} \to \mathit{Numbers}$
  - Expectations: $\mathit{Program\ States} \to \mathit{Numbers}$
  - Let $M : \mathit{Program\ States} \to \mathit{Features}$, then $T \circ M$ is an expectation
- Represent as piece-wise mathematical expression



Learner: Extract Expectations

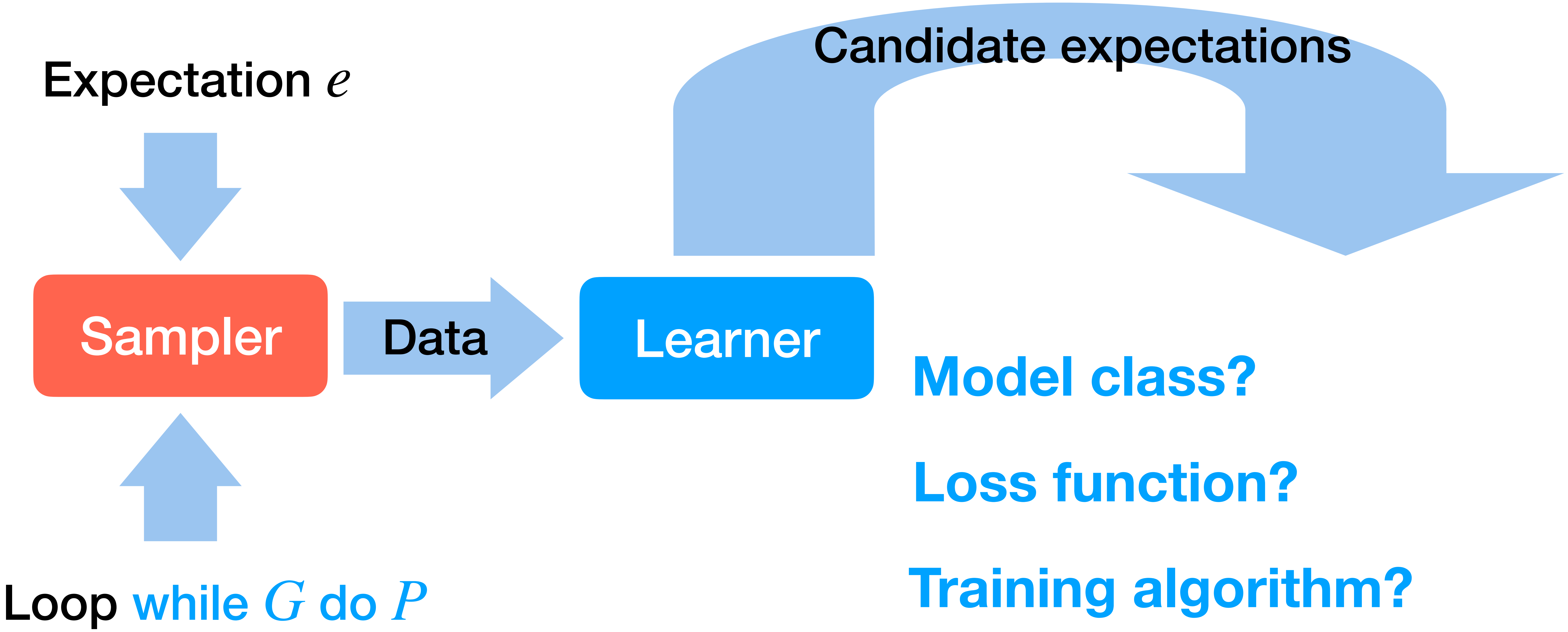
- A view change
    - Model Tree T: $\textit{Features} \to \textit{Numbers}$
    - Expectations: $\textit{Program States} \to \textit{Numbers}$
    - Let $M : \textit{Program States} \to \textit{Features},$ then $T \circ M$ is an expectation
- Represent as piece-wise mathematical expression



$$[flip \neq 0] \cdot (z) + [flip = 0] \cdot (z + \frac{1-p}{p})$$

Estimate the map

Learn the map



Model class?

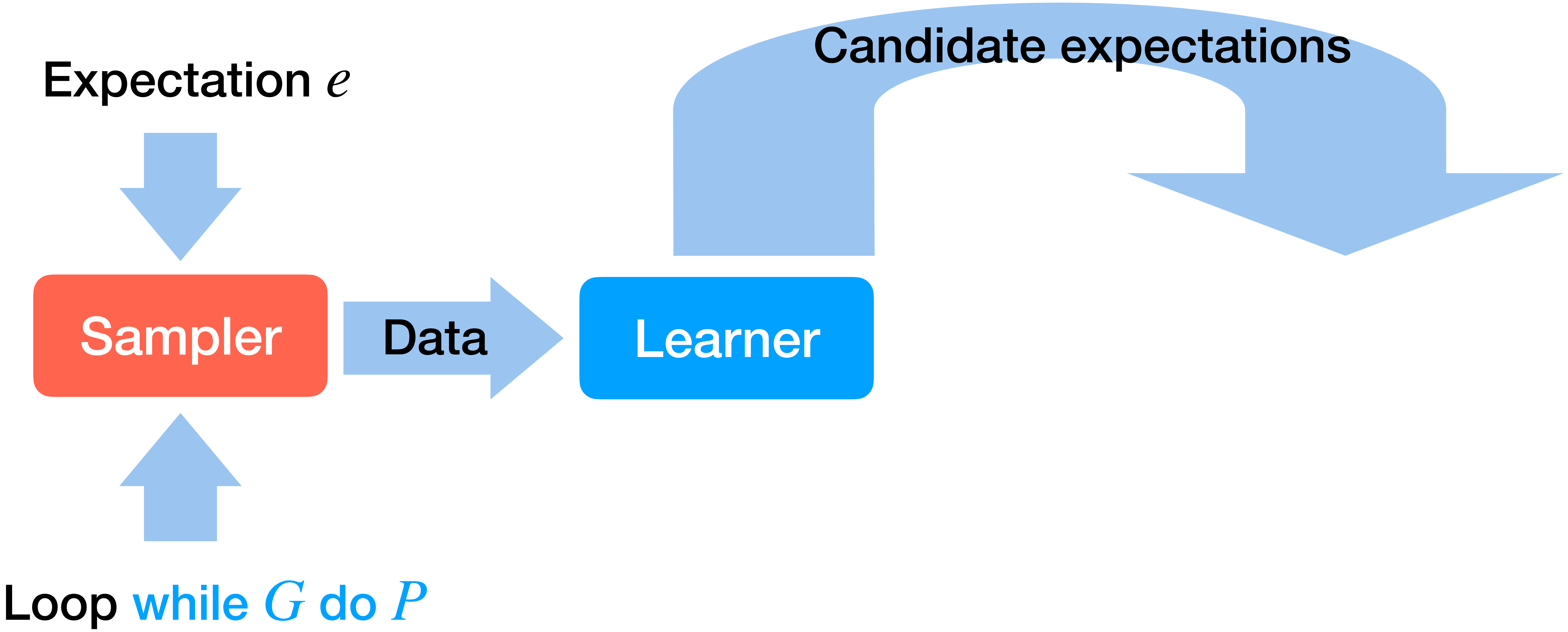
Loss function?

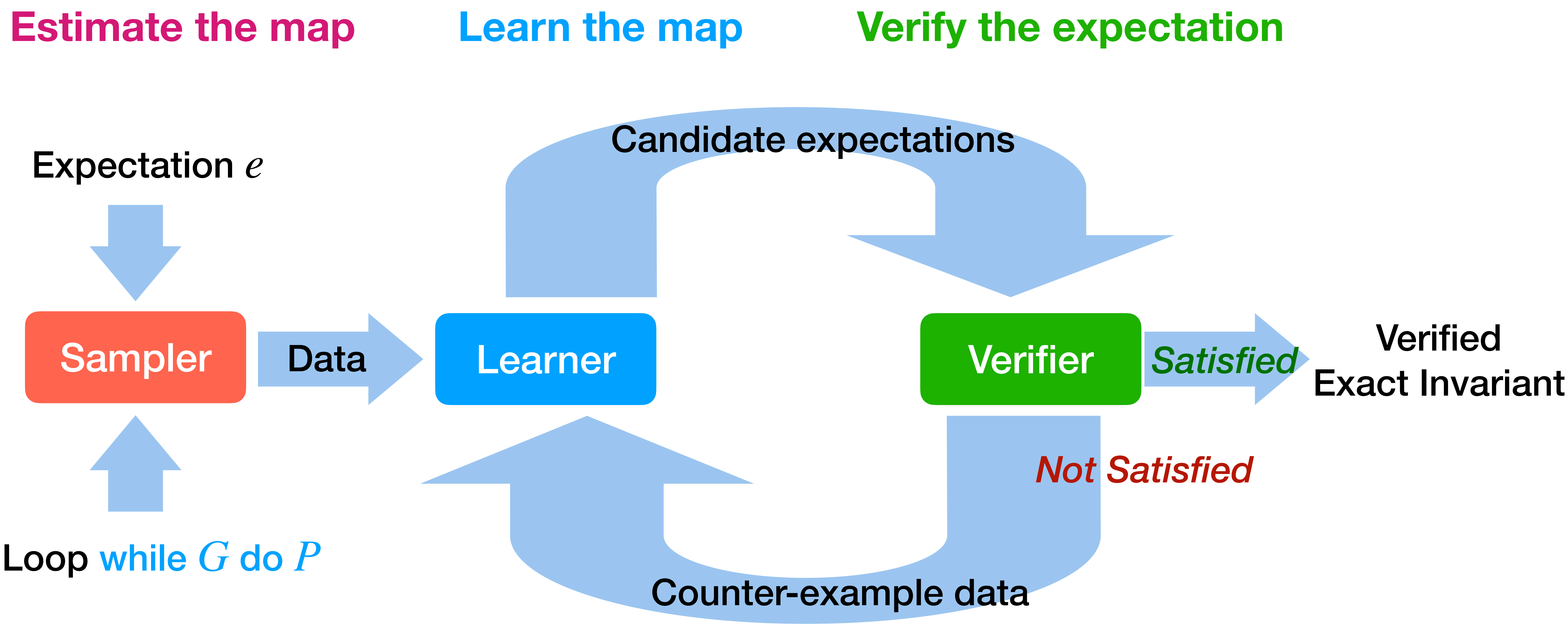
Training algorithm?

How to formulate the expectations?

Estimate the map

Learn the map





Estimate the map
Learn the map
Verify the expectation
Expectation e
Candidate expectations
Sampler
Data
Learner
Verifier
Satisfied
Verified
Exact Invariant
Not Satisfied
Loop while G do P
Counter-example data

Verifier: Check Candidate Expectations

Verifier: Check Candidate Expectations

- **Recall:** In simple cases, $I = [G] \cdot wpe(P, I) + [\neg G] \cdot e$ iff $I = wpe($while G do $P, e)$.

Verifier: Check Candidate Expectations

- **Recall:** In simple cases, $I = [G] \cdot wpe(P, I) + [\neg G] \cdot e$ iff $I = wpe($while G do $P, e)$.

- **Observation:** if P is loop less, then it's possible to calculate $wpe(P, I)$ syntactically

Verifier: Check Candidate Expectations

- **Recall:** In simple cases, $I = [G] \cdot wpe(P, I) + [\neg G] \cdot e$ iff $I = wpe($while G do $P, e)$.

- **Observation:** if P is loop less, then it's possible to calculate $wpe(P, I)$ syntactically

- Given candidate expectation I', we use a solver to check if $I' = [G] \cdot wpe(P, I') + [\neg G] \cdot e$

WolframAlpha

Verifier: Find Counter-examples

Verifier: Find Counter-examples

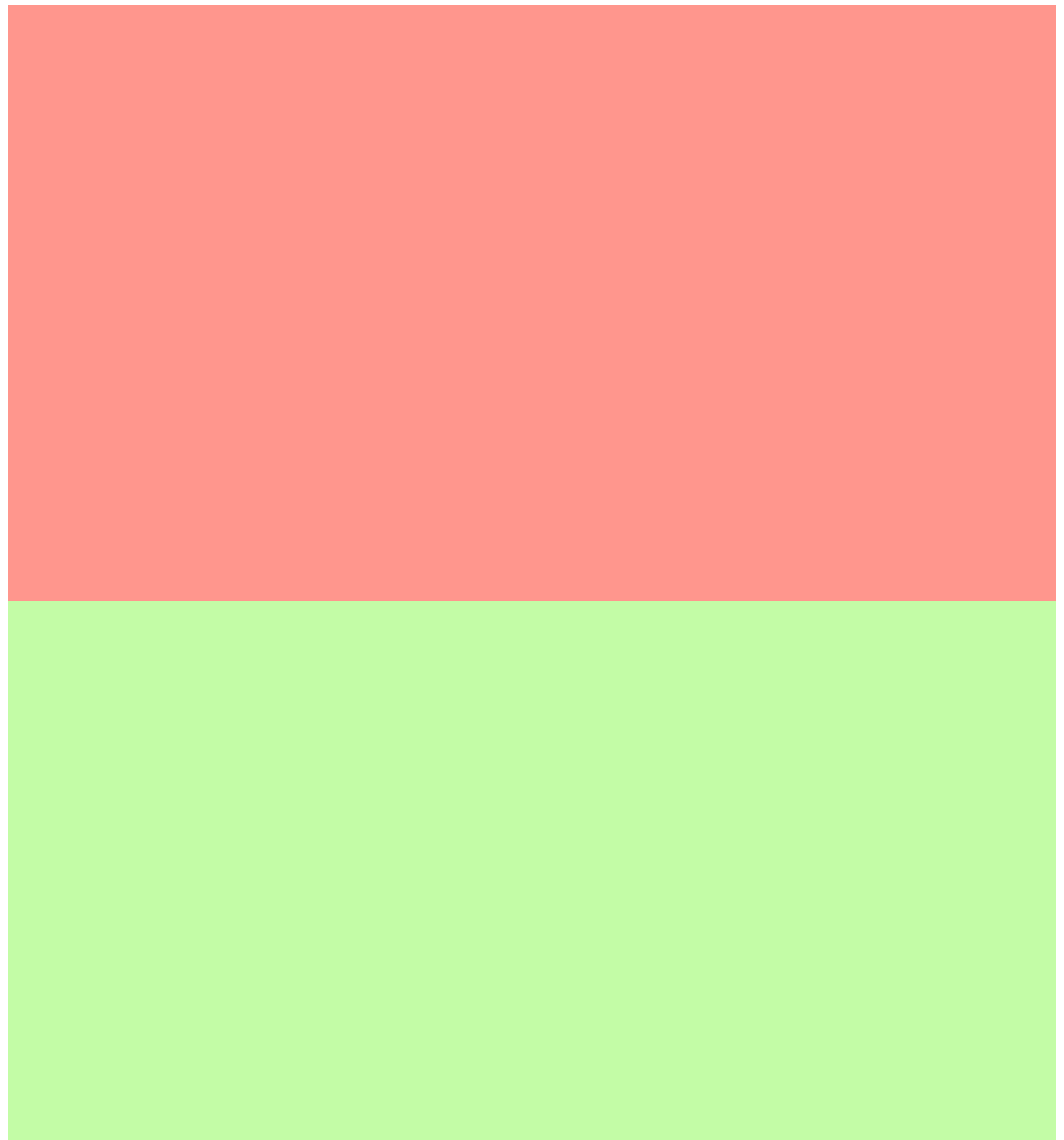
- One counter-example may not change the learning process enough

Verifier: Find Counter-examples

- One counter-example may not change the learning process enough
- Solution:
  - Find multiple counter-examples
  - Find counter-example that maximize the violation

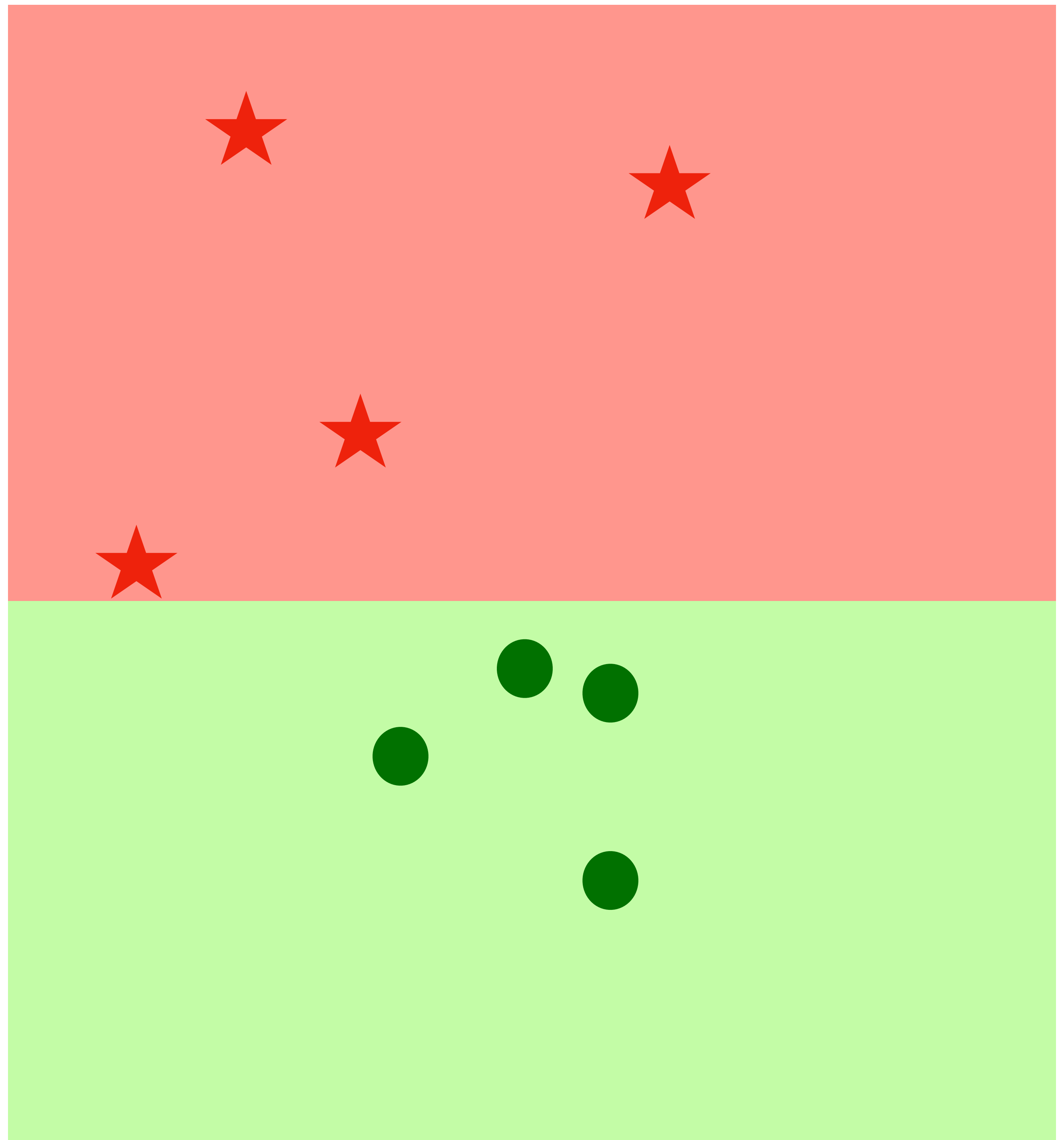
Verifier: Find Counter-examples

- One counter-example may not change the learning process enough
- Solution:
  - Find multiple counter-examples
  - Find counter-example that maximize the violation



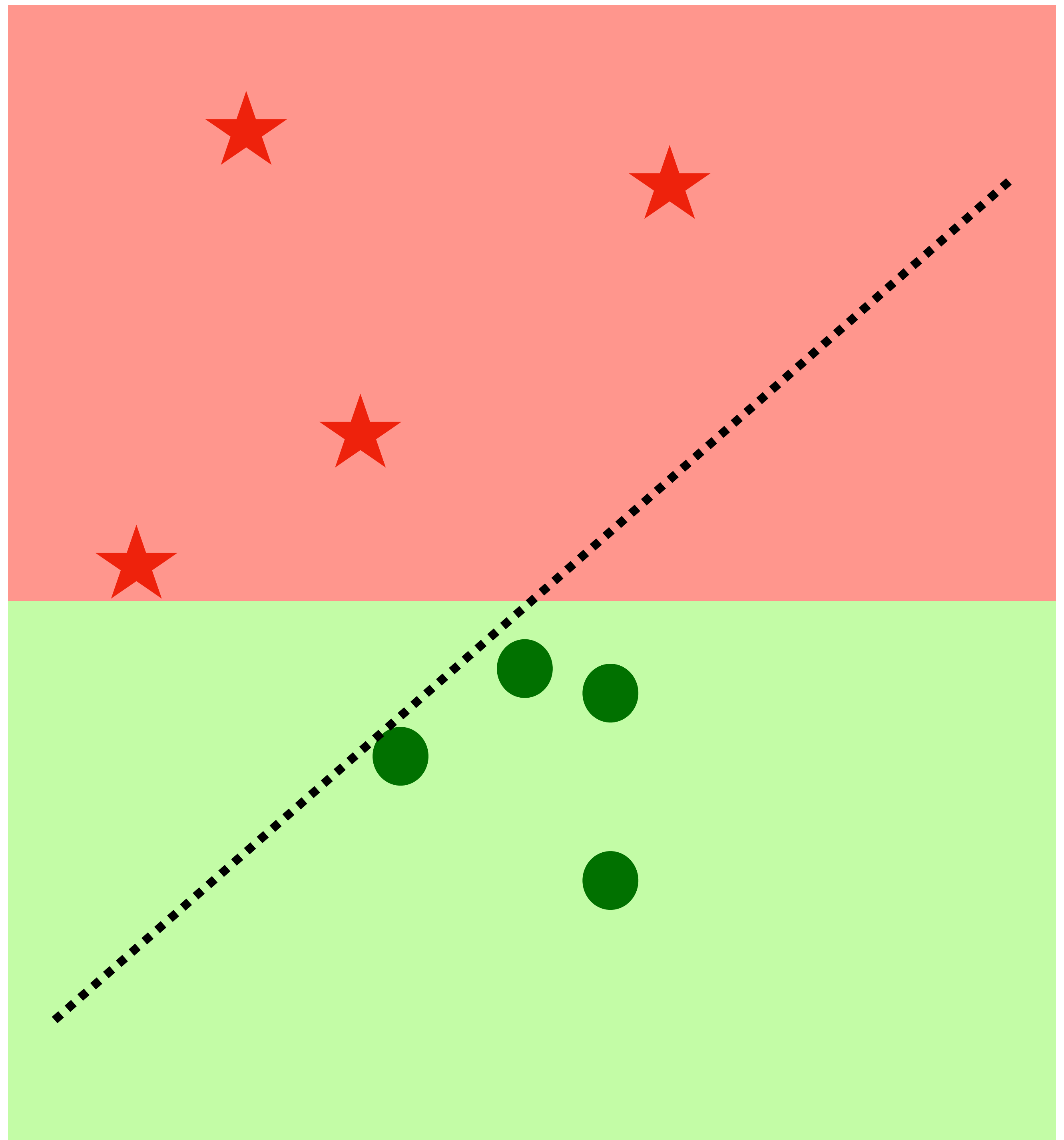
Verifier: Find Counter-examples

- One counter-example may not change the learning process enough
- Solution:
  - Find multiple counter-examples
  - Find counter-example that maximize the violation



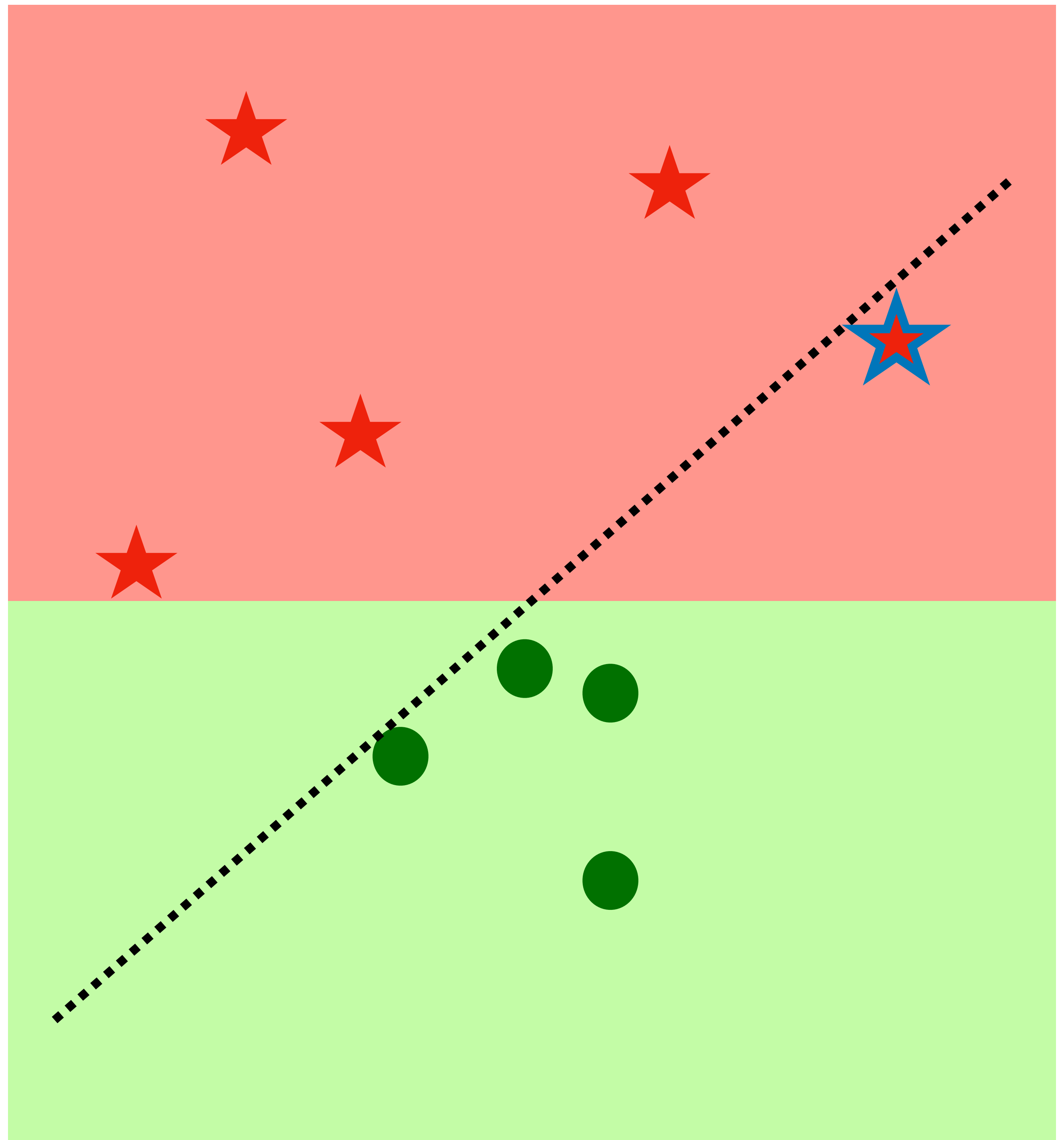
Verifier: Find Counter-examples

- One counter-example may not change the learning process enough
- Solution:
  - Find multiple counter-examples
  - Find counter-example that maximize the violation



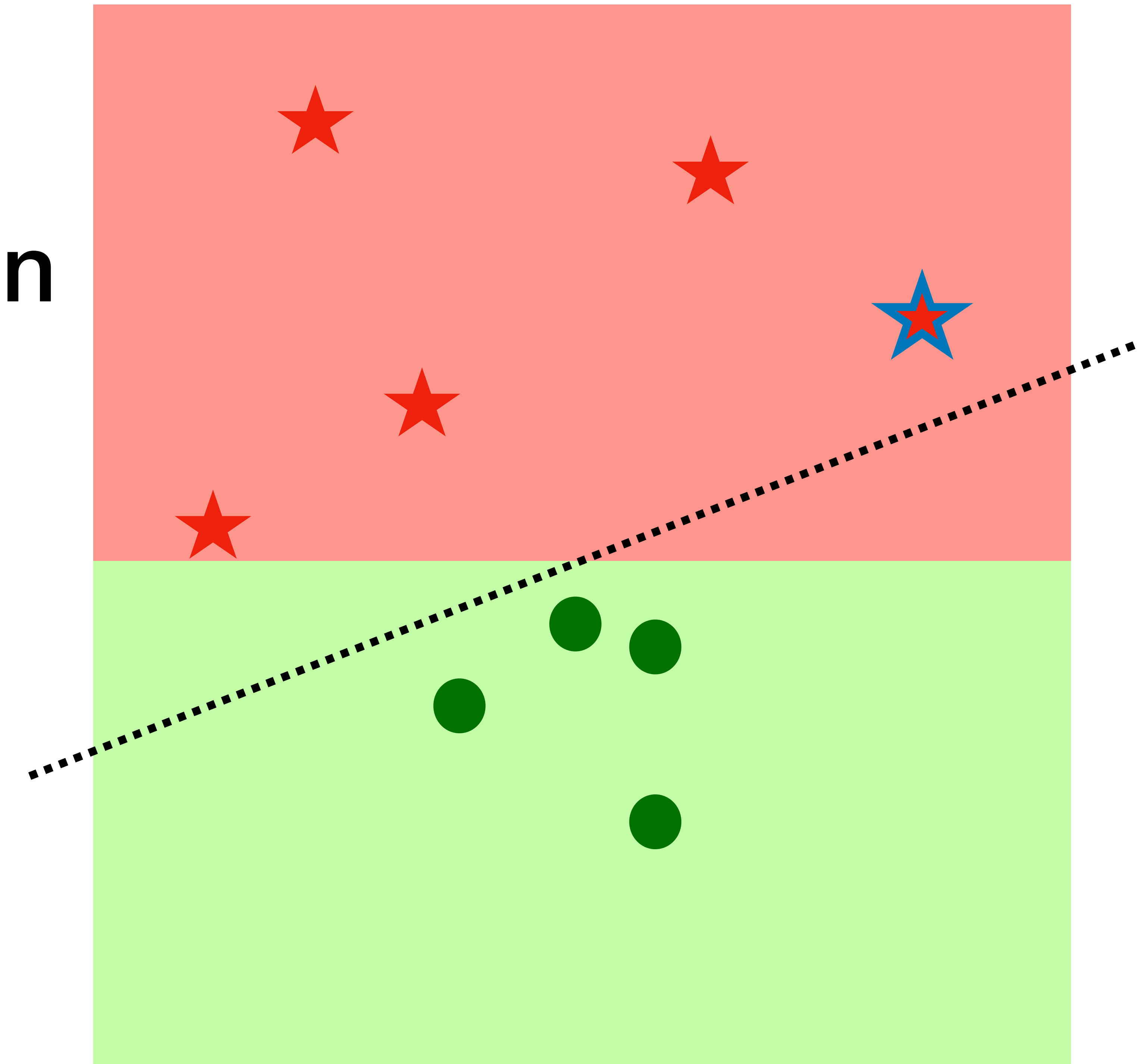
Verifier: Find Counter-examples

- One counter-example may not change the learning process enough
- Solution:
  - Find multiple counter-examples
  - Find counter-example that maximize the violation



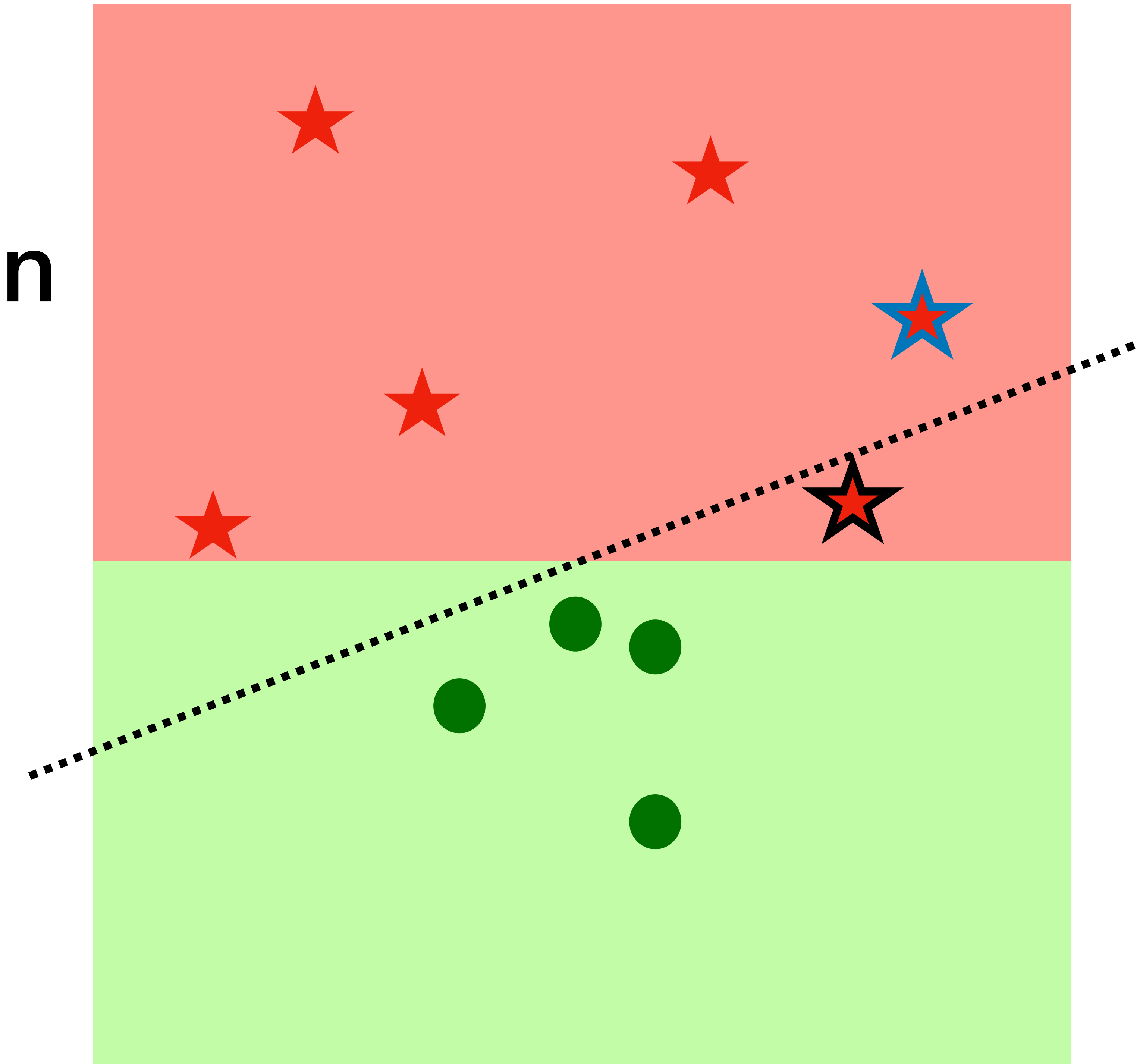
Verifier: Find Counter-examples

- One counter-example may not change the learning process enough
- Solution:
  - Find multiple counter-examples
  - Find counter-example that maximize the violation



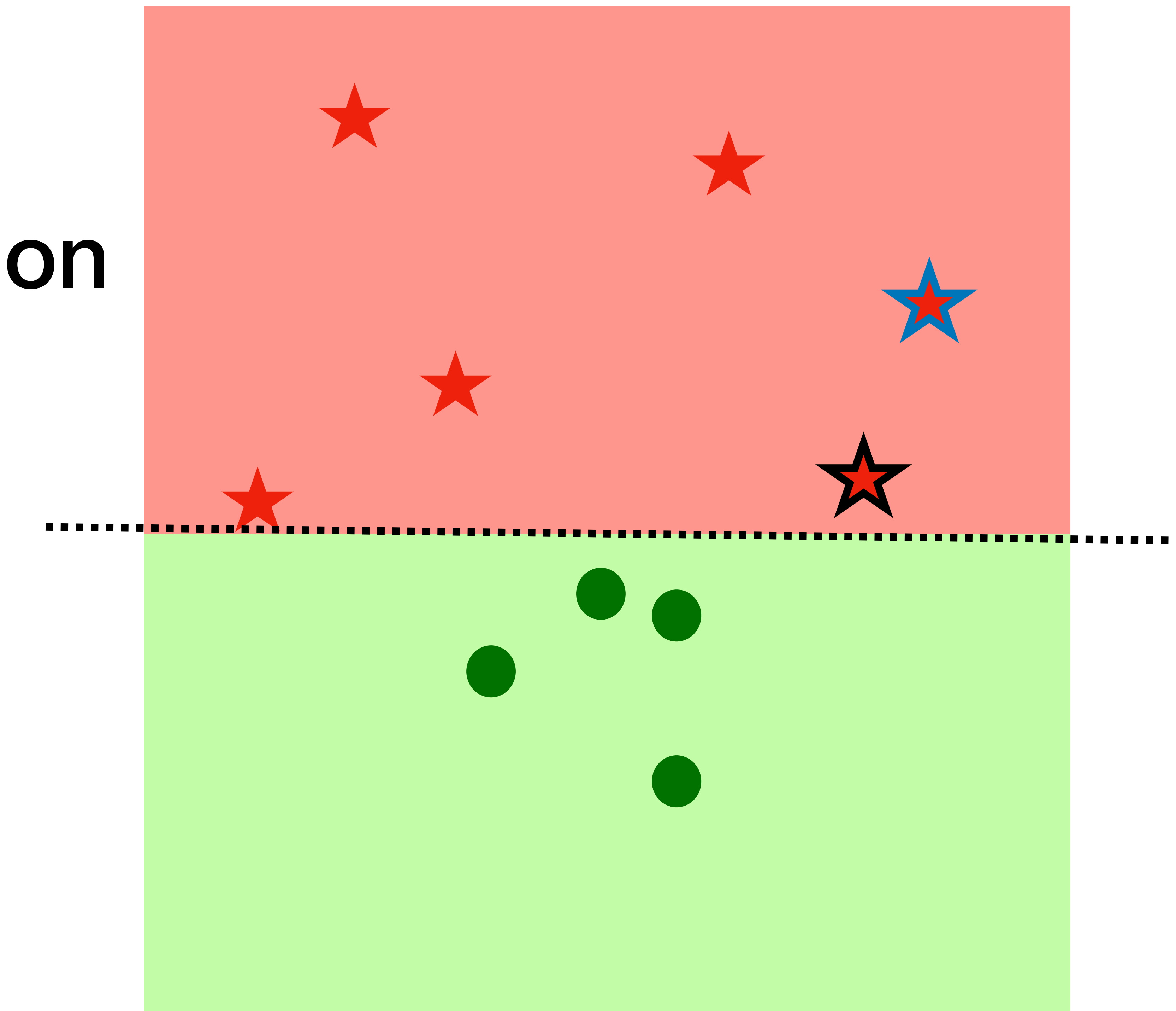
Verifier: Find Counter-examples

- One counter-example may not change the learning process enough
- Solution:
  - Find multiple counter-examples
  - Find counter-example that maximize the violation



Verifier: Find Counter-examples

- One counter-example may not change the learning process enough
- Solution:
  - Find multiple counter-examples
  - Find counter-example that maximize the violation



Implementation

Implementation

- We implemented a prototype in Python, using Wolfram Alpha Engine for verifier

Implementation

- We implemented a prototype in Python, using Wolfram Alpha Engine for verifier
  - Sample 500 program states

Implementation

- We implemented a prototype in Python, using Wolfram Alpha Engine for verifier
    - Sample 500 program states
    - Run 500 times from each program state

Implementation

- We implemented a prototype in Python, using Wolfram Alpha Engine for verifier
  - Sample 500 program states
  - Run 500 times from each program state
  - Timeout after 10 mins

Evaluations

Evaluations

- We evaluated on 18 benchmarks collected from prior work

Evaluations

- We evaluated on 18 benchmarks collected from prior work
  - Successfully generate invariants for 15 out of 18 benchmarks before timeout

Evaluations

- We evaluated on 18 benchmarks collected from prior work
  - Successfully generate invariants for 15 out of 18 benchmarks before timeout
  - Time cost: 3-299 sec

Evaluations

- We evaluated on 18 benchmarks collected from prior work
  - Successfully generate invariants for 15 out of 18 benchmarks before timeout
  - Time cost: 3-299 sec
    - Dominated by sampling time

Limitations

Limitations

- Fail when the ground truth exact invariant is too complicated

Limitations

- Fail when the ground truth exact invariant is too complicated
  - Too many digits, e.g., $I = z + [n > 0] \cdot 2.625 \cdot n$

Limitations

- Fail when the ground truth exact invariant is too complicated
  - Too many digits, e.g., $I = z + [n > 0] \cdot 2.625 \cdot n$
    - Our learner oscillates between expectations like $[n > 0] \cdot 2.63 \cdot n - 0.02$ and $[n > 0] \cdot 2.62 \cdot n + 0.01$

Limitations

- Fail when the ground truth exact invariant is too complicated
  - Too many correlated terms, e.g.,

    $$x \cdot y + [n > 0] \cdot (0.25 \cdot n^2 + 0.5 \cdot n \cdot x + 0.5 \cdot n \cdot y - 0.25 \cdot n)$$

    - Our learner generates

      $$\ldots(0.25 \cdot n^2 + 0.5 \cdot n \cdot x + 0.5 \cdot n \cdot y - 0.27 \cdot n - 0.01 \cdot x + 0.12)$$

Limitations

More data may help

- Fail when the ground truth exact invariant is too complicated
  - Too many correlated terms, e.g.,
    $x \cdot y + [n > 0] \cdot (0.25 \cdot n^2 + 0.5 \cdot n \cdot x + 0.5 \cdot n \cdot y - 0.25 \cdot n)$
    - Our learner generates
      $\ldots(0.25 \cdot n^2 + 0.5 \cdot n \cdot x + 0.5 \cdot n \cdot y - 0.27 \cdot n - 0.01 \cdot x + 0.12)$

Limitations

- Fail when the ground truth exact invariant is too complicated
  - The verifier gets stuck, e.g.,

$$[c = 0] \cdot [t = 0] \cdot \frac{p1}{p1 + p2 - p1 \cdot p2} +$$

$$[c = 0] \cdot [t = 0] \cdot \frac{(1 - p2) \cdot p1}{p1 + p2 - p1 \cdot p2} + [c = 0] \cdot (t)$$

Limitations

Not only need more data but also more powerful verifier

- Fail when the ground truth exact invariant is too complicated
  - The verifier gets stuck, e.g.,

$$[c=0]\cdot[t=0]\cdot\frac{p1}{p1+p2-p1\cdot p2} +$$

$$[c=0]\cdot[t=0]\cdot\frac{(1-p2)\cdot p1}{p1+p2-p1\cdot p2}+[c=0]\cdot(t)$$

Stepping Back …

Stepping Back …

- Previously, we generate exact invariant by learning I that approximates $wpe(\text{while } G \text{ do } P, e)$ and then check if $I = [G] \cdot wpe(P, I) + [\neg G] \cdot e$.

Stepping Back …

- Previously, we generate exact invariant by learning I that approximates $wpe(\text{while } G \text{ do } P, e)$ and then check if $I = [G] \cdot wpe(P, I) + [\neg G] \cdot e$.
- Question: Can we generate subinvariants, i.e., I such that $I \leq [G] \cdot wpe(P, I) + [\neg G] \cdot e$?
  - Sometimes the exact invariant is too complicated
  - I is a subinvariant implies $I \leq wpe(\text{while } G \text{ do } P, e)$, not the other way

New Problem: Learning subinvariants

New Problem: Learning subinvariants

- **Given:** a loop while G do P and two expectation pre and e.

New Problem: Learning subinvariants

- **Given:** a loop while G do P and two expectation pre and e.
- **Goal:** find expectation I such that $I \leq [G] \cdot wpe(P, I) + [\neg G] \cdot e$ and $pre \leq I$.

New Problem: Learning subinvariants

- **Given:** a loop while G do P and two expectation pre and e.
- **Goal:** find expectation I such that $I \leq [G] \cdot wpe(P, I) + [\neg G] \cdot e$ and $pre \leq I$.
  - Equivalent conditions:
    $$\bigwedge_s I(s) \leq [G] \cdot wpe(P, I)(s) + [\neg G] \cdot e(s) \wedge \bigwedge_s pre(s) \leq I(s)$$

Casting into a Learning Problem

Casting into a Learning Problem

- **The condition**
  - $\bigwedge_{s} I(s) \leq [G] \cdot wpe(P, I)(s) + [\neg G] \cdot e(s) \wedge \bigwedge_{s} preE(s) \leq I(s)$

Casting into a Learning Problem

- **The condition**
  - $\bigwedge_s I(s) \leq [G] \cdot wpe(P, I)(s) + [\neg G] \cdot e(s) \wedge \bigwedge_s preE(s) \leq I(s)$
- **The ideal loss**
  - $Loss'(I) = \sum_s \max(0, I(s) - G(s) \cdot wpe(P, I)(s) - (1 - G(s)) \cdot e(s))$
    $+ \sum_s \max(0, preE(s) - I(s))$

Casting into a Learning Problem

- **The condition**
  - $\bigwedge_s I(s) \le [G] \cdot wpe(P, I)(s) + [\neg G] \cdot e(s) \wedge \bigwedge_s preE(s) \le I(s)$
- **The ideal loss**
  - $Loss'(I) = \sum_s \max(0, I(s) - G(s) \cdot wpe(P, I)(s) - (1 - G(s)) \cdot e(s))$ $+ \sum_s \max(0, preE(s) - I(s))$

In practice: we only sum over sampled states

Casting into a Learning Problem

- **The condition**
  - $\bigwedge_s I(s) \leq [G] \cdot wpe(P, I)(s) + [\neg G] \cdot e(s) \wedge \bigwedge_s preE(s) \leq I(s)$
- **The ideal loss**
  - $Loss'(I) = \sum_s \max(0, I(s) - G(s) \cdot wpe(P, I)(s) - (1 - G(s)) \cdot e(s))$
    $+ \sum_s \max(0, preE(s) - I(s))$

In practice: we only sum over sampled states

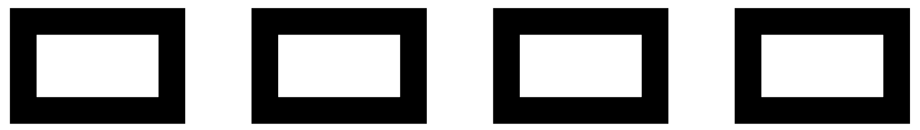
In practice: we need to estimate $\lambda I \, . \, wpe(P, I)(s)$

How to estimate $\lambda I \,.\, wpe(P, I)(s)$

How to estimate $\lambda I \,.\, wpe(P, I)(s)$

Initial state

s



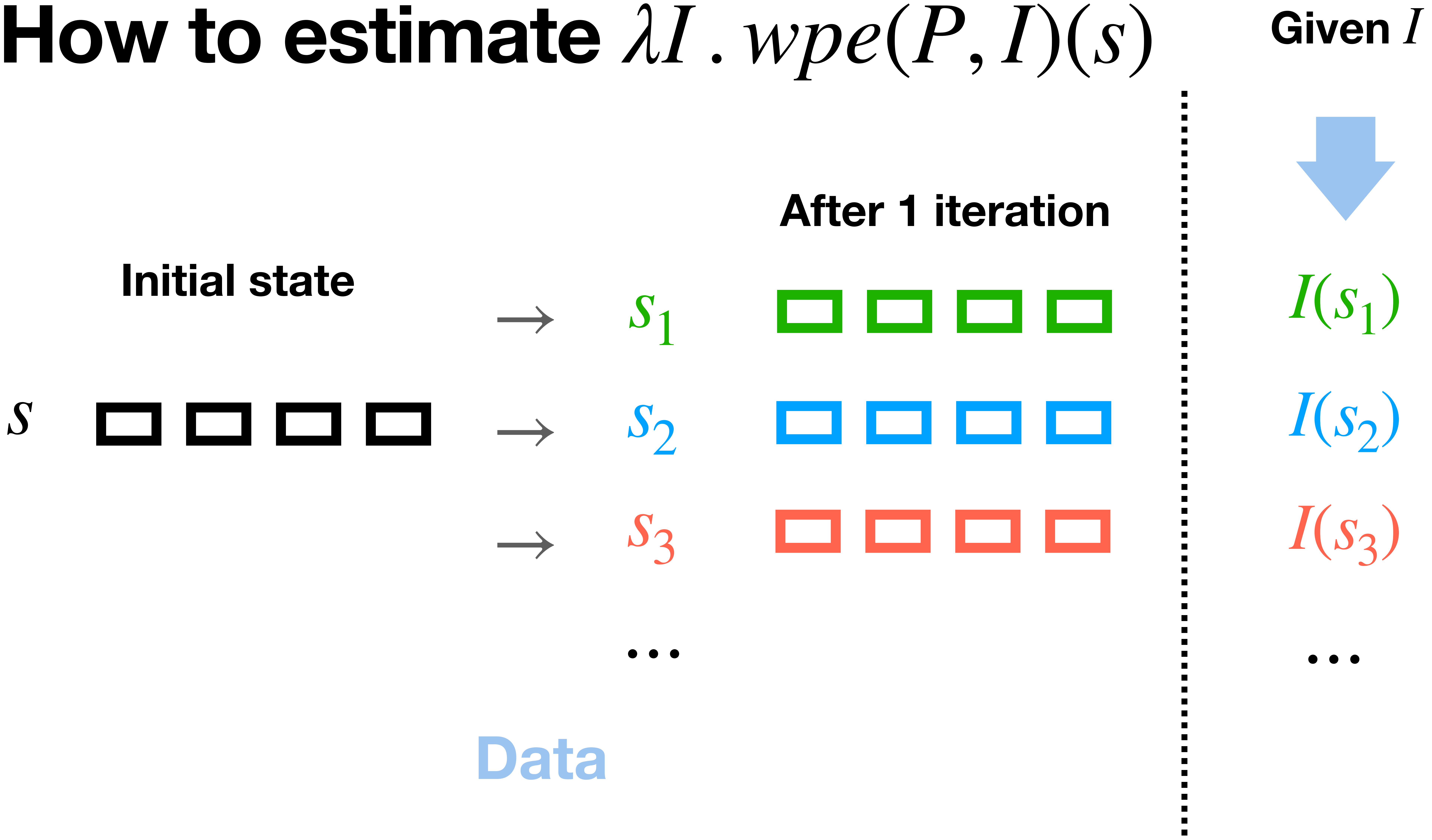
How to estimate $\lambda I \,.\, wpe(P, I)(s)$

How to estimate $\lambda I\,.\,wpe(P, I)(s)$

How to estimate $\lambda I\,.\,wpe(P, I)(s)$

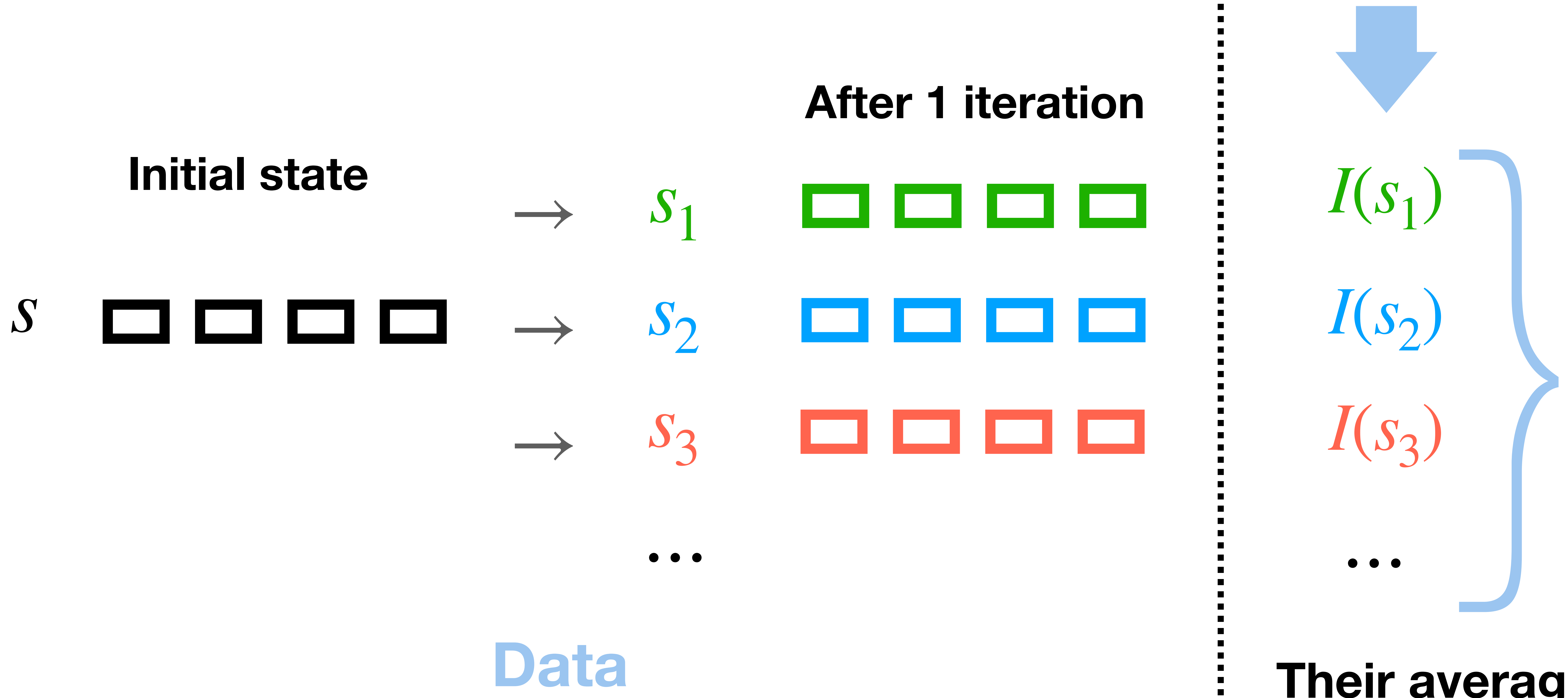
Given I

How to estimate $\lambda I\,.\,wpe(P, I)(s)$



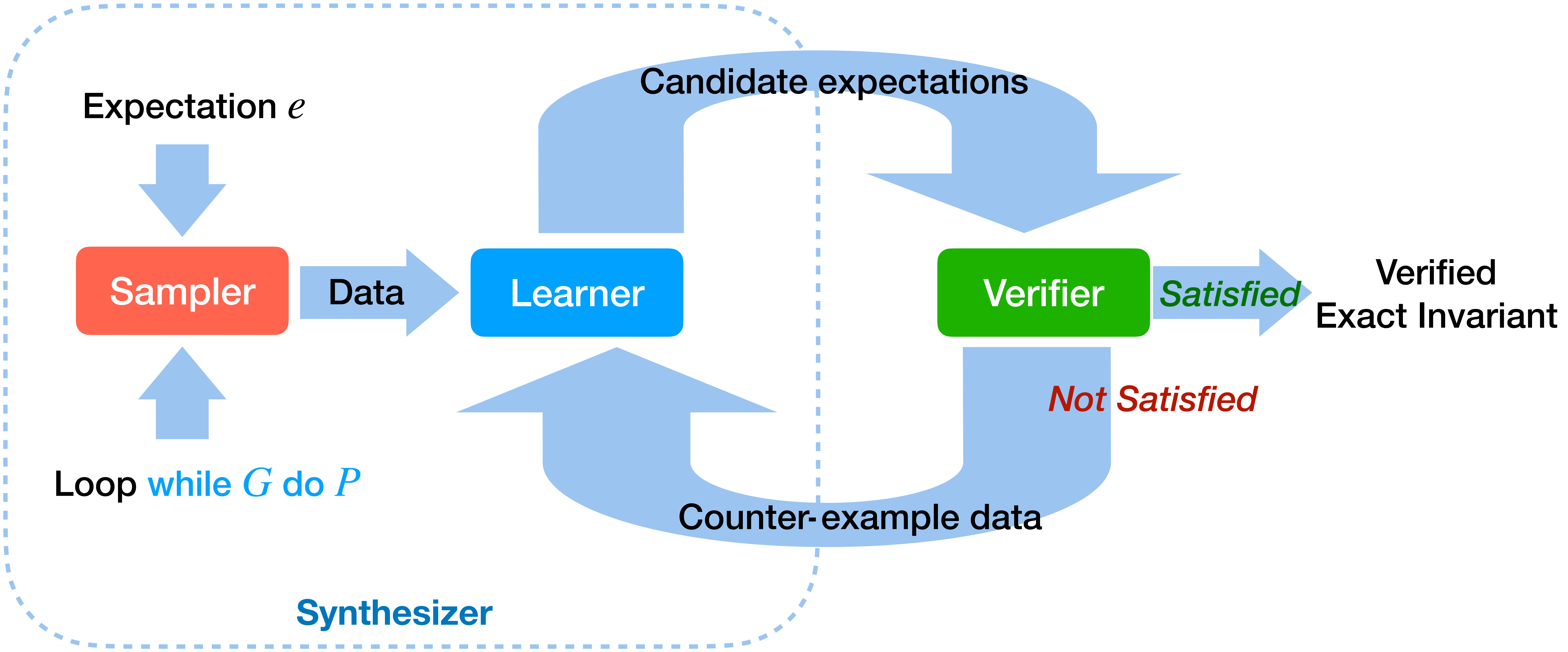
How to estimate $\lambda I \,.\, wpe(P, I)(s)$

Given I



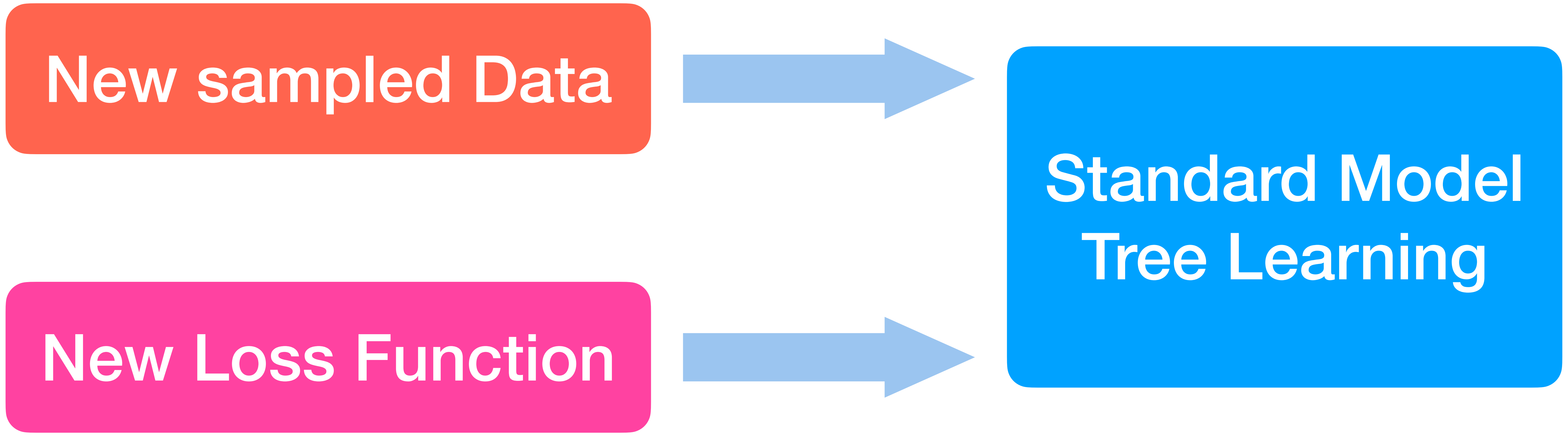
Their average estimates $wpe(P, I)(s)$

Same Method, Different Implementations

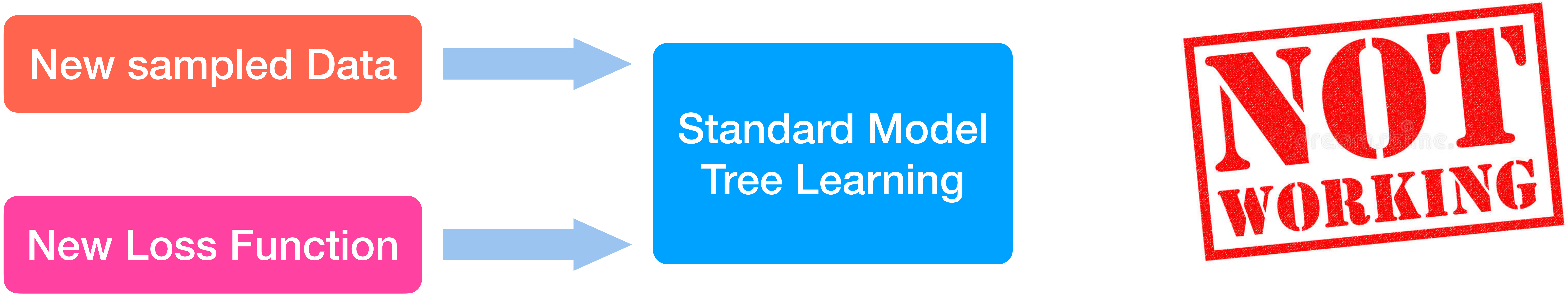


Challenge

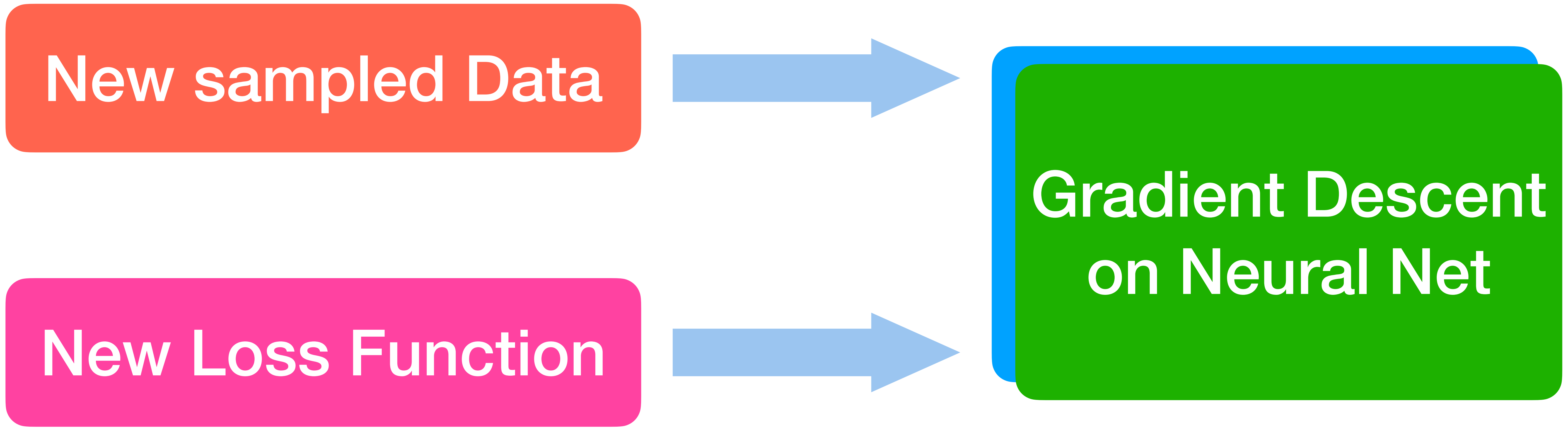
Challenge


New sampled Data
New Loss Function
Standard Model
Tree Learning

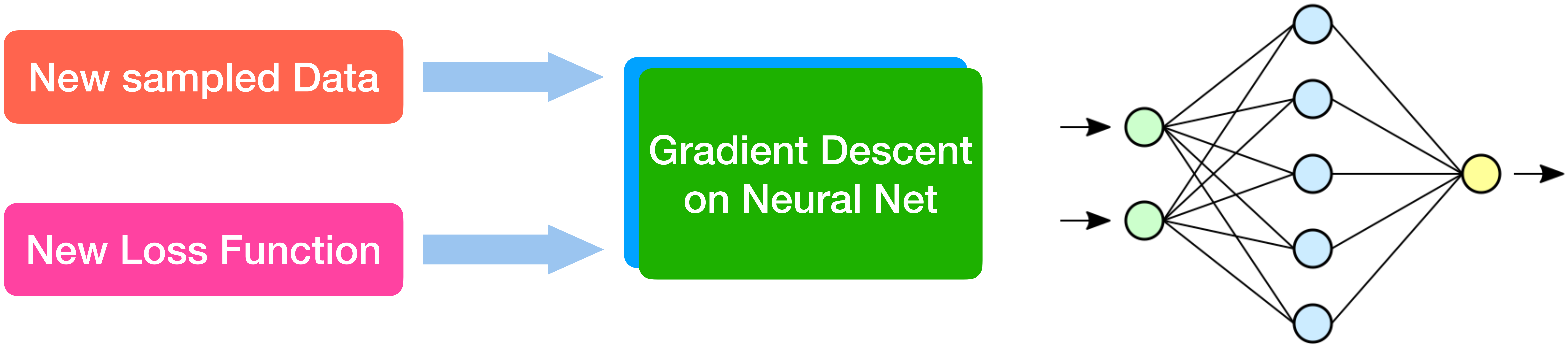
Challenge


New sampled Data
New Loss Function
Standard Model
Tree Learning
NOT
WORKING

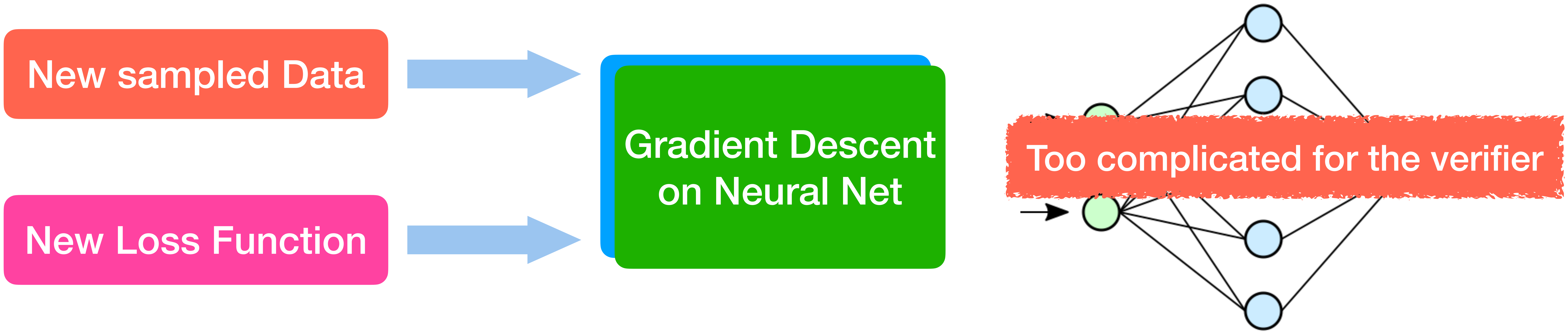
Challenge


New sampled Data
New Loss Function
Gradient Descent on Neural Net

Challenge


New sampled Data
New Loss Function
Gradient Descent on Neural Net

Challenge



New sampled Data
New Loss Function
Gradient Descent on Neural Net
Too complicated for the verifier

Our Solution

New sampled Data →

New Loss Function →

Our Solution

New sampled Data

New Loss Function

Gradient Descent on neural encodings of model trees

Our Solution

New sampled Data

New Loss Function

→ Gradient Descent on <u>neural encodings of model trees</u>

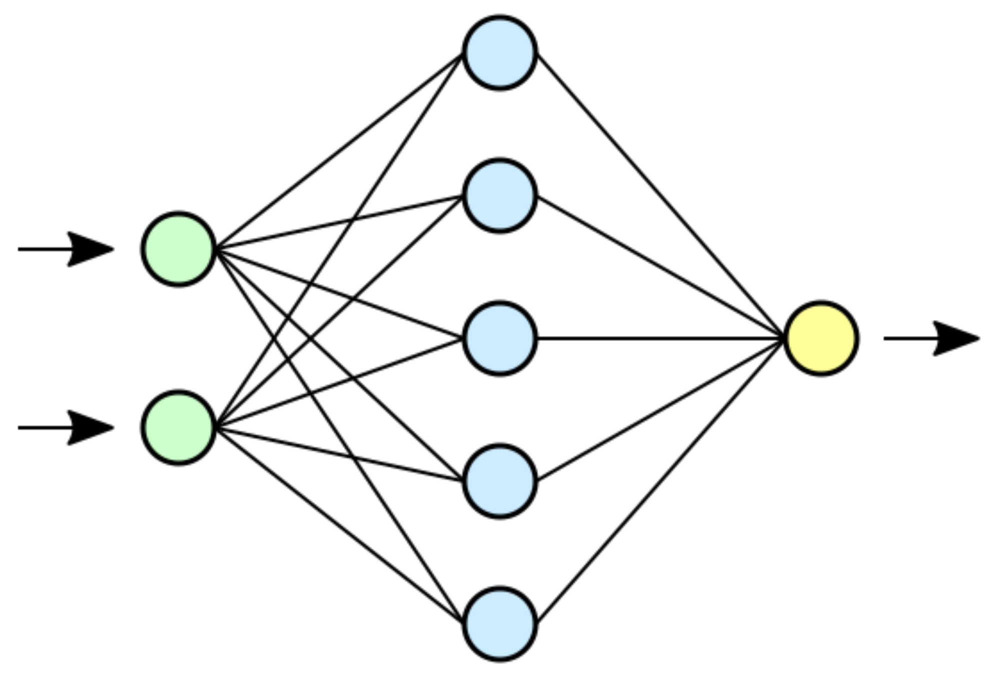
i.e., differentiable approximation of model trees

Our Solution

New sampled Data

New Loss Function

Gradient Descent on <u>neural encodings of model trees</u>



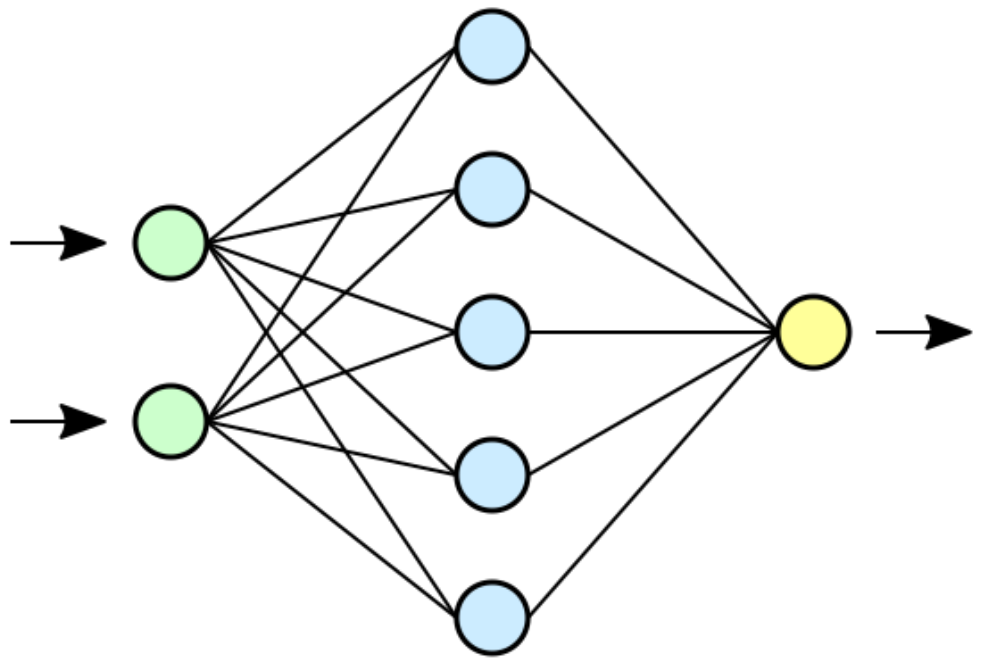
i.e., differentiable approximation of model trees

Our Solution

New sampled Data

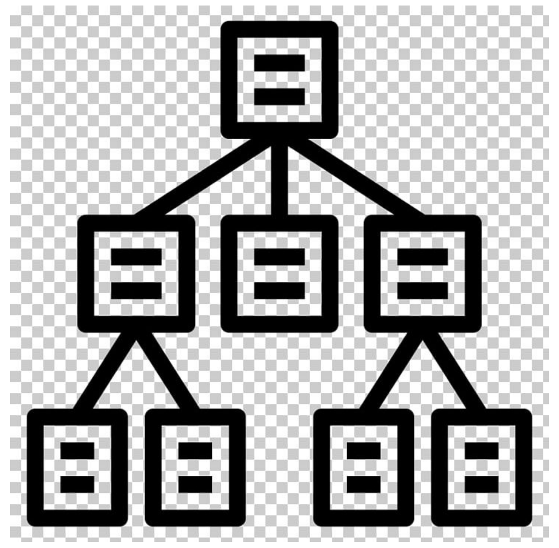
New Loss Function

Gradient Descent on <u>neural encodings of model trees</u>



i.e., differentiable approximation of model trees

A Model Tree



Evaluations

Evaluations

- We constructed 32 benchmarks using the 18 programs we collected from prior work and different preexpectations

Evaluations

- We constructed 32 benchmarks using the 18 programs we collected from prior work and different preexpectations
    - Successfully generate subinvariants for 25 out of 32 benchmarks before timeout

Evaluations

- We constructed 32 benchmarks using the 18 programs we collected from prior work and different preexpectations
  - Successfully generate subinvariants for 25 out of 32 benchmarks before timeout
  - Time cost: 33-196 sec

Evaluations

- We constructed 32 benchmarks using the 18 programs we collected from prior work and different preexpectations
  - Successfully generate subinvariants for 25 out of 32 benchmarks before timeout
  - Time cost: 33-196 sec
    - Dominated by learning time

Limitations

Limitations

- Noise in sample data

Limitations

- Noise in sample data
- Gradient descent seems to get stuck in local minima

Limitations

- Noise in sample data
- Gradient descent seems to get stuck in local minima
  - It seems a hard learning program.

The Learning Problem

The Learning Problem

- Symbolic Regression:

The Learning Problem

- Symbolic Regression:
  - Given: a dataset (X, y) where each point has inputs $X_i \in \mathbb{R}^n$ and response $y_i \in R$:

The Learning Problem

- Symbolic Regression:
  - Given: a dataset (X, y) where each point has inputs $X_i \in \mathbb{R}^n$ and response $y_i \in R$:
  - Goal: find a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ that best fits the data set, where f is a short closed-form mathematical expression.

The Learning Problem

- Symbolic Regression:
  - Given: a dataset (X, y) where each point has inputs $X_i \in \mathbb{R}^n$ and response $y_i \in R$:
  - Goal: find a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ that best fits the data set, where f is a short closed-form mathematical expression.
- Our learning problem in exact invariant generation is almost the same.

The Learning Problem

- Symbolic Regression:
  - Given: a dataset (X, y) where each point has inputs $X_i \in \mathbb{R}^n$ and response $y_i \in R$:
  - Goal: find a function $f : \mathbb{R}^n \to \mathbb{R}$ that best fits the data set, where f is a short closed-form mathematical expression.
- Our learning problem in exact invariant generation is almost the same.
- Our learning problem in subinvariant generation is a bit more general.

The State of Art of Symbolic Regression

Table 1: Recovery rate of several algorithms on the Nguyen benchmark problem set across 100 independent training runs. Results of our algorithm are obtained using PQT; slightly lower recovery rates were obtained using VPG and RSPG training (see Table 3 for comparisons).

		Recovery rate (%)					
Benchmark	Expression	Ours	DSR	PQT	VPG	GP	Eureqa
Nguyen-1	x^3+x^2+x	100	100	100	96	100	100
Nguyen-2	$x^4+x^3+x^2+x$	100	100	99	47	97	100
Nguyen-3	$x^5+x^4+x^3+x^2+x$	100	100	86	4	100	95
Nguyen-4	$x^6+x^5+x^4+x^3+x^2+x$	100	100	93	1	100	70
Nguyen-5	$\sin(x^2)\cos(x)-1$	100	72	73	5	45	73
Nguyen-6	$\sin(x)+\sin(x+x^2)$	100	100	98	100	91	100
Nguyen-7	$\log(x+1)+\log(x^2+1)$	97	35	41	3	0	85
Nguyen-8	$\sqrt{x}$	100	96	21	5	5	0
Nguyen-9	$\sin(x)+\sin(y^2)$	100	100	100	100	100	100
Nguyen-10	$2\sin(x)\cos(y)$	100	100	91	99	76	64
Nguyen-11	x^y	100	100	100	100	7	100
Nguyen-12	$x^4-x^3+\frac{1}{2}y^2-y$	0	0	0	0	0	0
	Average	**91.4**	83.6	75.2	46.7	60.1	73.9

From *Symbolic Regression via Neural-Guided Genetic Programming Population Seeding* [Neurips 2021]

Take Away

Take Away

PL problems

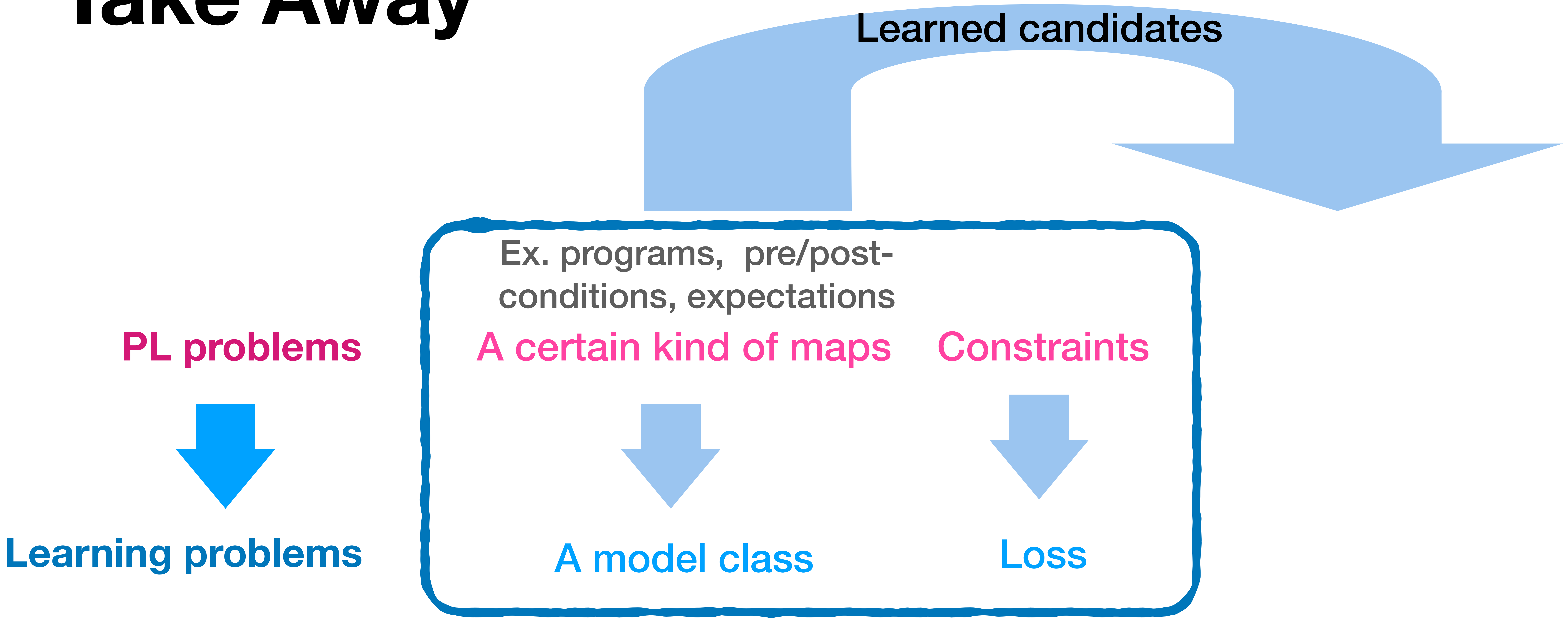
Learning problems

Take Away

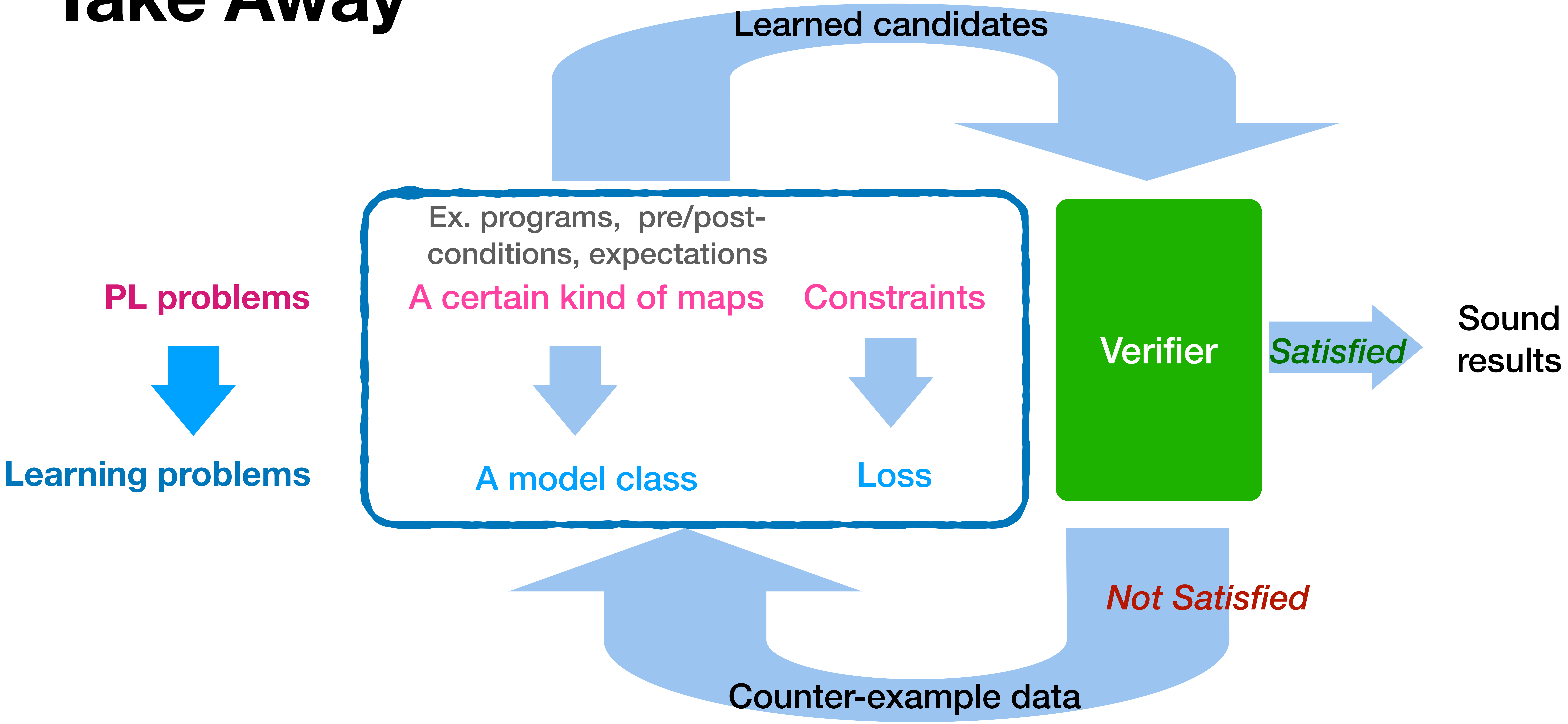
Ex. programs, pre/post-conditions, expectations

PL problems	A certain kind of maps	Constraints
↓	↓	↓
Learning problems	A model class	Loss

Take Away


Learned candidates
Ex. programs, pre/post-conditions, expectations
PL problems
A certain kind of maps
Constraints
Learning problems
A model class
Loss

Take Away


Learned candidates
Ex. programs, pre/post-conditions, expectations
PL problems
A certain kind of maps
Constraints
Learning problems
A model class
Loss
Verifier
Satisfied
Sound results
Not Satisfied
Counter-example data