Invariant Generation via Genetic Programming

Soonho Kong
soon@ropas.snu.ac.kr

12 Mar 2010
ROPAS Show&Tell

Problem:

Inductive Invariant Generation

For the annotated loop

$$\{\delta\}\ \texttt{while}\ \rho\ \texttt{do}\ S\ \texttt{end}\ \{\epsilon\}$$

Find an invariant I satisfying the following conditions:

(A) $\delta \Rightarrow \iota$ (ι holds when entering the loop)

(B) $\iota \wedge \rho \Rightarrow Pre(\iota, S)$ (ι holds at each iteration)

(C) $\iota \wedge \neg\rho \Rightarrow \epsilon$ (ι gives ϵ after leaving the loop)

$$\delta \Rightarrow I \Rightarrow \epsilon \vee \rho$$

$\underline{\iota}$ strongest under-approximation of an invariant

$\overline{\iota}$ weakest over-approximation of an invariant

Precondition

$\{\ phase = \mathtt{F} \land success = \mathtt{F} \land give\_up = \mathtt{F} \land cutoff = 0 \land count = 0\ \}$

Loop Body

$\mathtt{while}\ \neg(success \lor give\_up)\ \mathtt{do}$
$\quad entered\_phase := \mathtt{F};$
$\quad \mathtt{if}\ \neg phase\ \mathtt{then}$
$\quad\quad \mathtt{if}\ cutoff = 0\ \mathtt{then}\ cutoff := 1;$
$\quad\quad \mathtt{else\ if}\ cutoff = 1 \land maxcost > 1\ \mathtt{then}\ cutoff := maxcost;$
$\quad\quad\quad \mathtt{else}\ phase := \mathtt{T};\ entered\_phase := \mathtt{T};\ cutoff := 1000;$
$\quad\quad \mathtt{if}\ cutoff = maxcost \land \neg search\ \mathtt{then}\ give\_up := \mathtt{T};$
$\quad \mathtt{else}$
$\quad\quad count := count + 1;$
$\quad\quad \mathtt{if}\ count > words\ \mathtt{then}\ give\_up := \mathtt{T};$
$\quad \mathtt{if}\ entered\_phase\ \mathtt{then}\ count := 1;$
$\quad linkages := \mathtt{nondet};$
$\quad \mathtt{if}\ linkages > 5000\ \mathtt{then}\ linkages := 5000;$
$\quad canonical := 0;\ valid := 0;$
$\quad \mathtt{if}\ linkages \neq 0\ \mathtt{then}$
$\quad\quad valid := \mathtt{nondet};\ \mathtt{assume}\ 0 \leq valid \land valid \leq linkages;$
$\quad\quad canonical := linkages;$
$\quad \mathtt{if}\ valid > 0\ \mathtt{then}\ success := \mathtt{T};$
$\mathtt{end}$

Postcondition

$\{\ (valid > 0 \lor count > words \lor (cutoff = maxcost \land \neg search)) \land$
$valid \leq linkages \land canonical = linkages \land linkages \leq 5000\ \}$

Fig. 3. A Sample Loop in SPEC2000 Benchmark PARSER

with 20 Atomic Propositions (Building Blocks)

$\{\ phase = \mathtt{F} \wedge success = \mathtt{F} \wedge give\_up = \mathtt{F} \wedge cutoff = 0 \wedge count = 0\ \}$

```
while ¬(success ∨ give_up) do
    entered_phase := F;
    if ¬phase then
        if cutoff = 0 then cutoff := 1;
        else if cutoff = 1 ...
```

An Invariant ✓

$$
\begin{aligned}
&success \Rightarrow (valid \le linkages \wedge linkages \le 5000 \wedge canonical = linkages) \bigwedge \\
&success \Rightarrow (\neg search \vee count > words \vee valid \neq 0) \bigwedge \\
&success \Rightarrow (count > words \vee cutoff = maxcost \vee (canonical \neq 0 \wedge valid \neq 0 \wedge linkages \neq 0)) \bigwedge \\
&give\_up \Rightarrow ((valid = 0 \wedge linkages = 0 \wedge canonical = linkages) \vee \\
&\qquad (canonical \neq 0 \wedge valid \le linkages \wedge linkages \le 5000 \wedge canonical = linkages)) \bigwedge \\
&give\_up \Rightarrow (cutoff = maxcost \vee count > words \vee \\
&\qquad (canonical \neq 0 \wedge valid \neq 0 \wedge linkages \neq 0)) \bigwedge \\
&give\_up \Rightarrow (\neg search \vee count > words \vee valid \neq 0)
\end{aligned}
$$

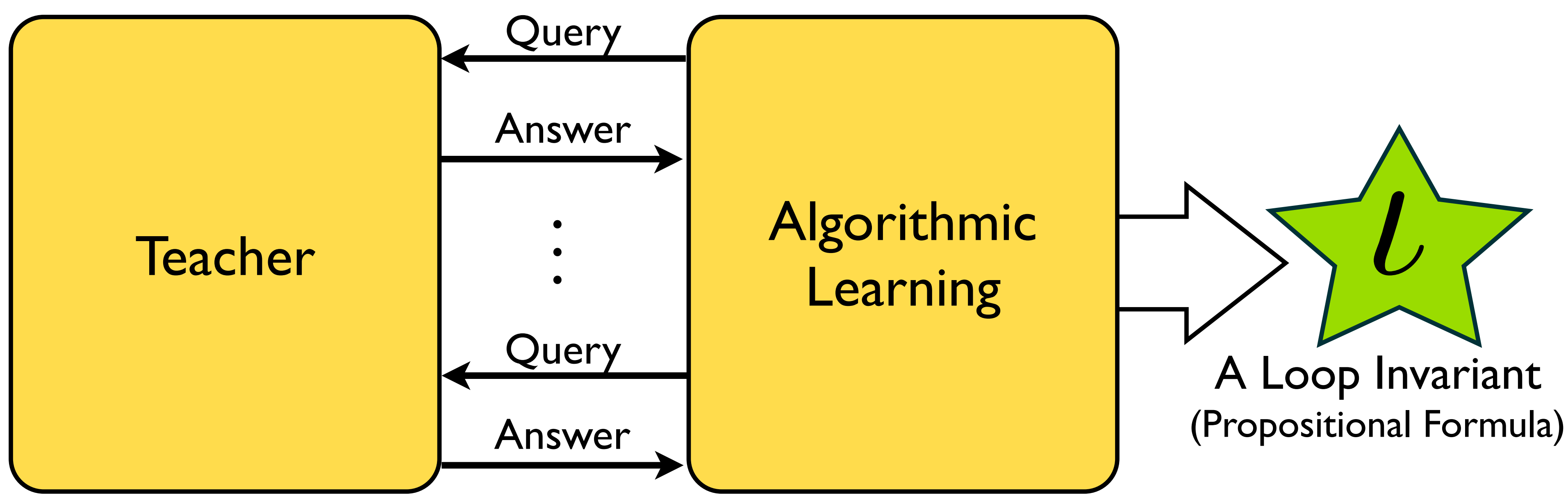
```
        valid := nondet; assume 0 ≤ valid ∧ valid ≤ linkages;
        canonical := linkages;
    if valid > 0 then success := T;
end
```

$\{\ (valid > 0 \vee count > words \vee (cutoff = maxcost \wedge \neg search)) \wedge$
$valid \le linkages \wedge canonical = linkages \wedge linkages \le 5000\ \}$

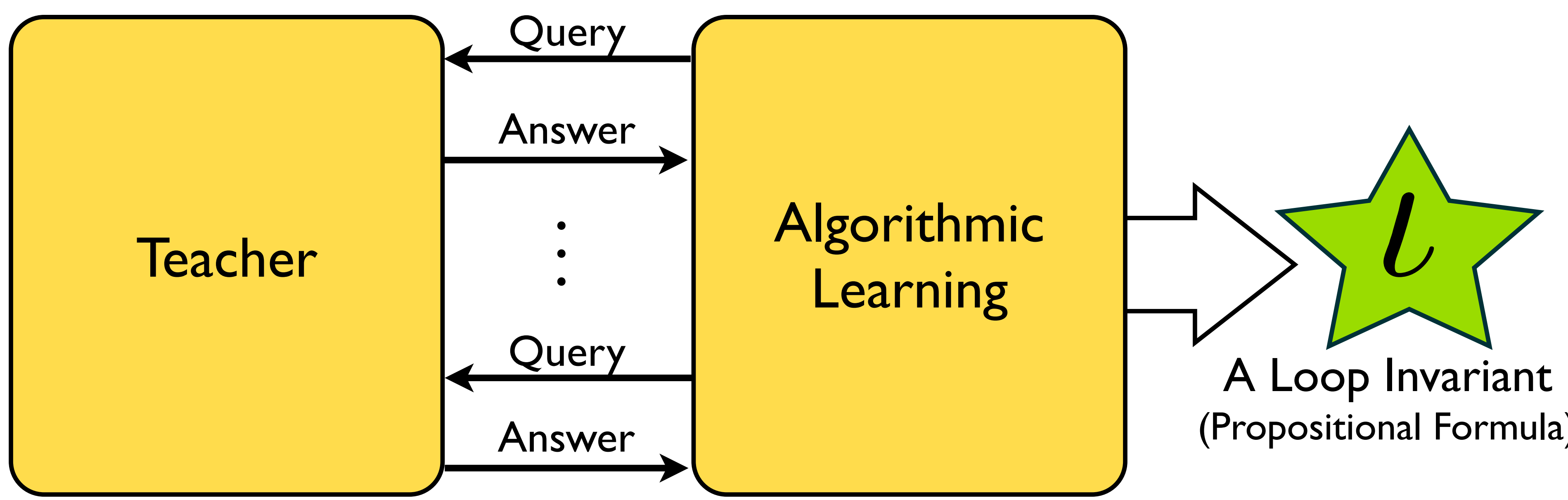
Fig. 3. A Sample Loop in SPEC2000 Benchmark PARSER

with 20 Atomic Propositions (Building Blocks)

Observations on Algorithmic Learning Approach



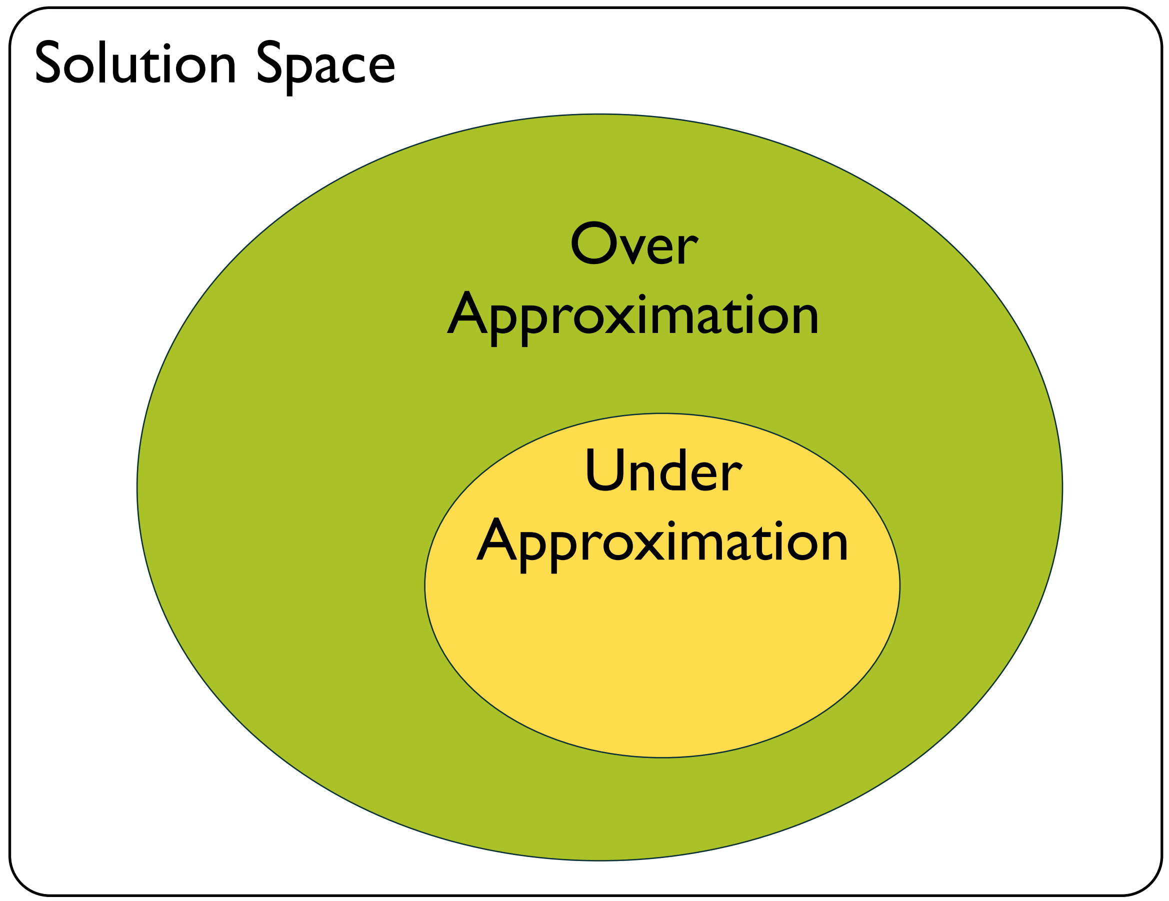
Observations on Algorithmic Learning Approach



Problem #1: **Active Learning**

Teacher can guide the learner only through the form of answers.

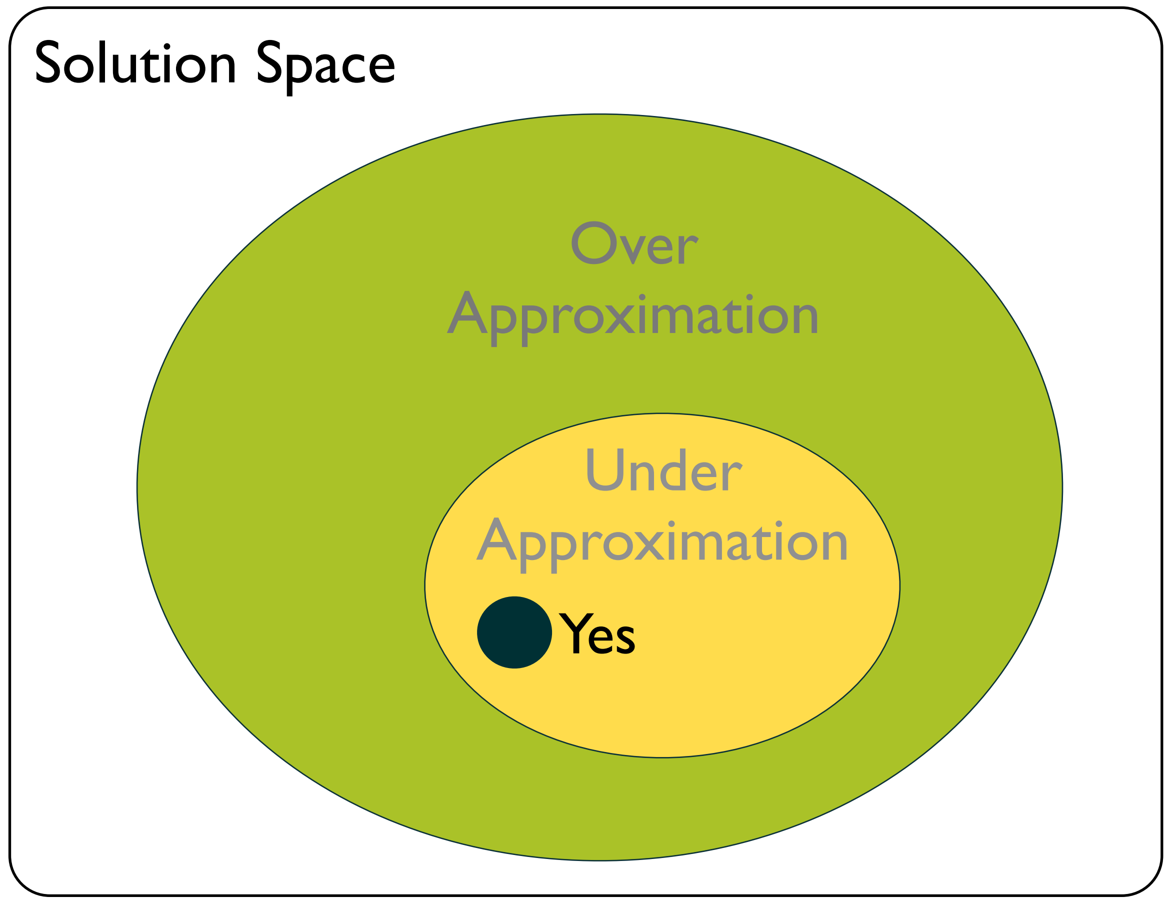
Observations on Algorithmic Learning Approach



Problem #1: **Active Learning**

Teacher can guide the learner only through the form of answers.

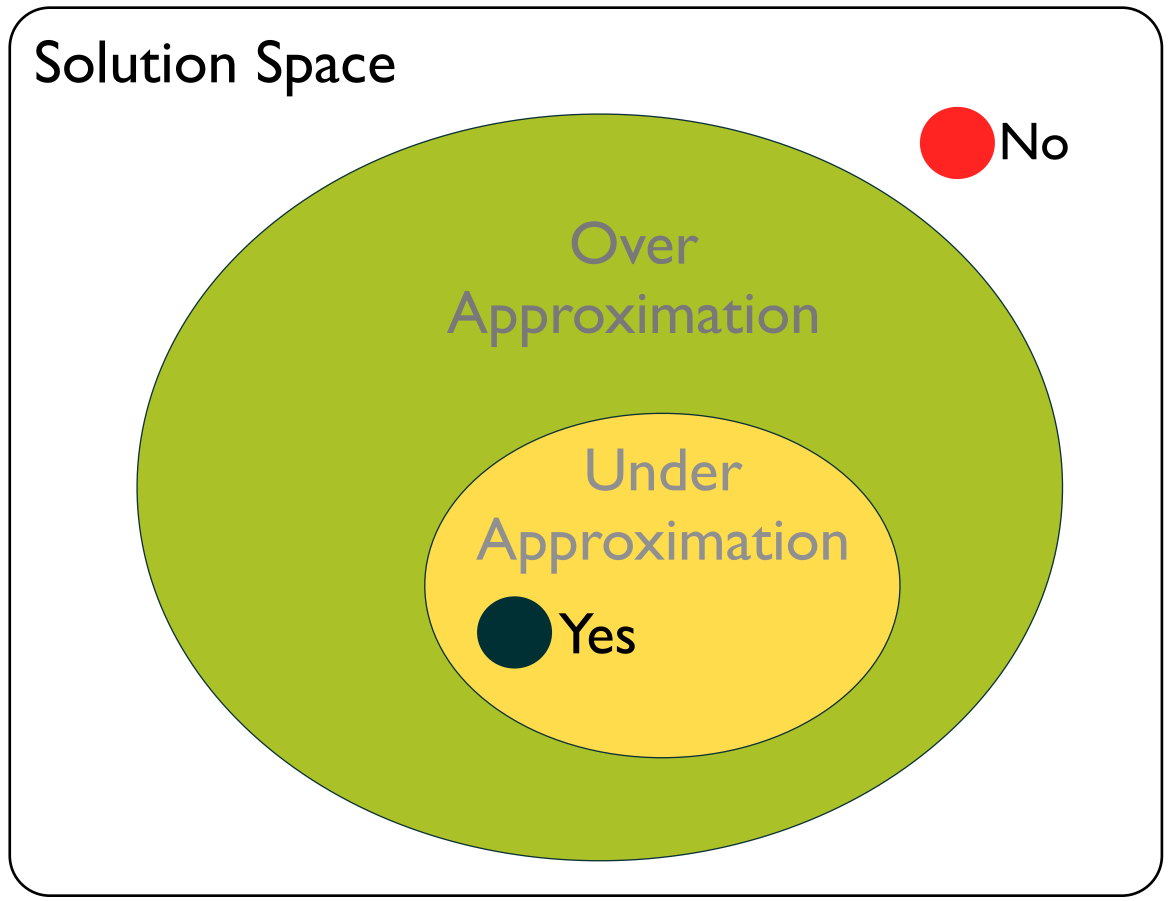
Observations on Algorithmic Learning Approach



Problem #1: **Active Learning**

Teacher can guide the learner only through the form of answers.

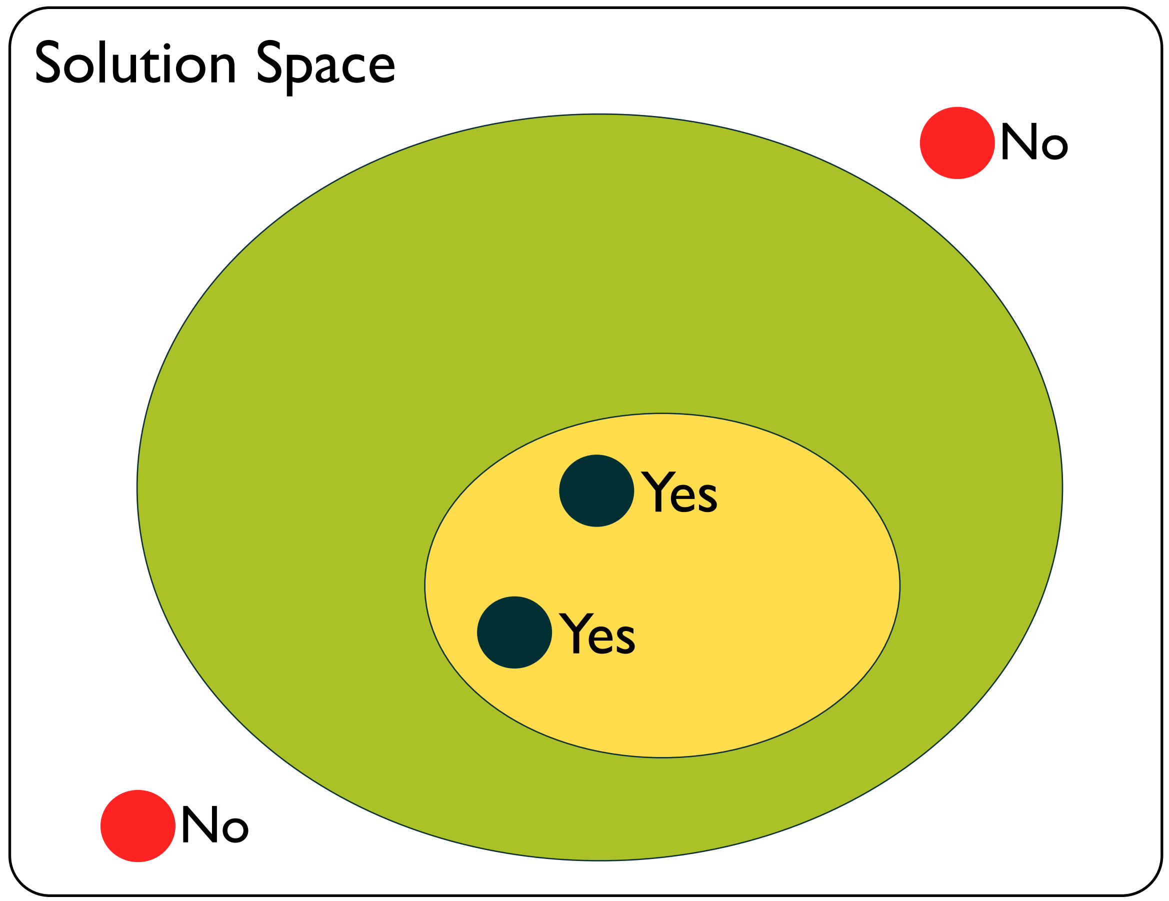
Observations on Algorithmic Learning Approach



Problem #1: **Active Learning**

Teacher can guide the learner only through the form of answers.

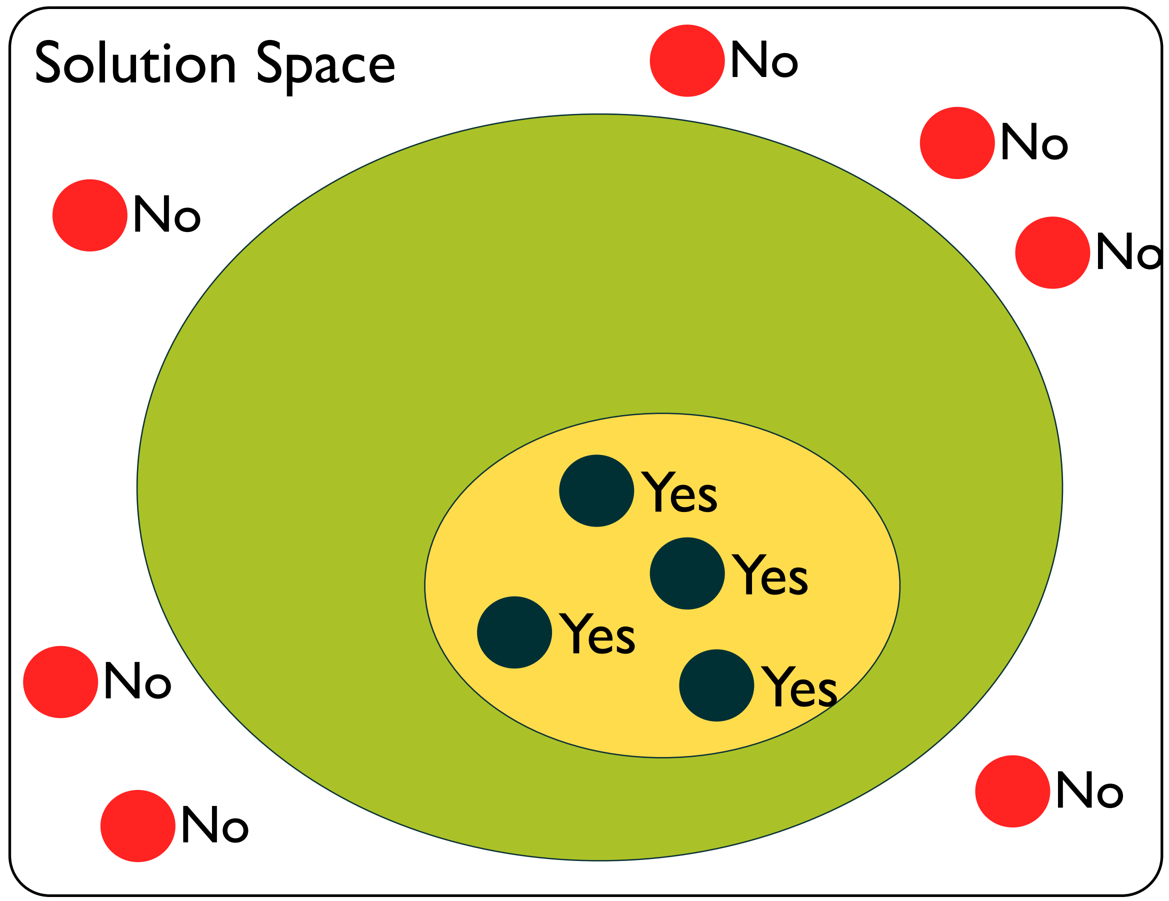
Observations on Algorithmic Learning Approach



Problem #1: **Active Learning**

Teacher can guide the learner only through the form of answers.

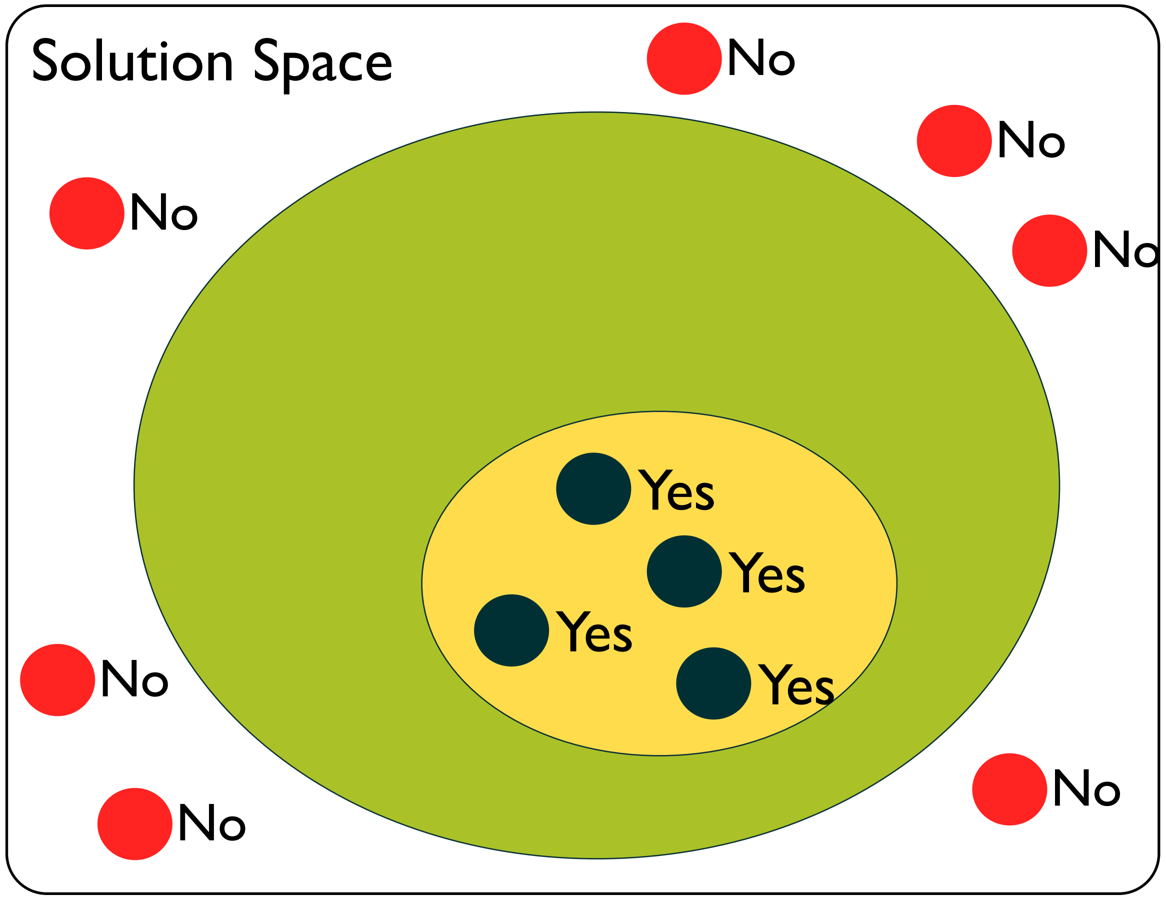
Observations on Algorithmic Learning Approach



Problem #1: **Active Learning**

Teacher can guide the learner only through the form of answers.

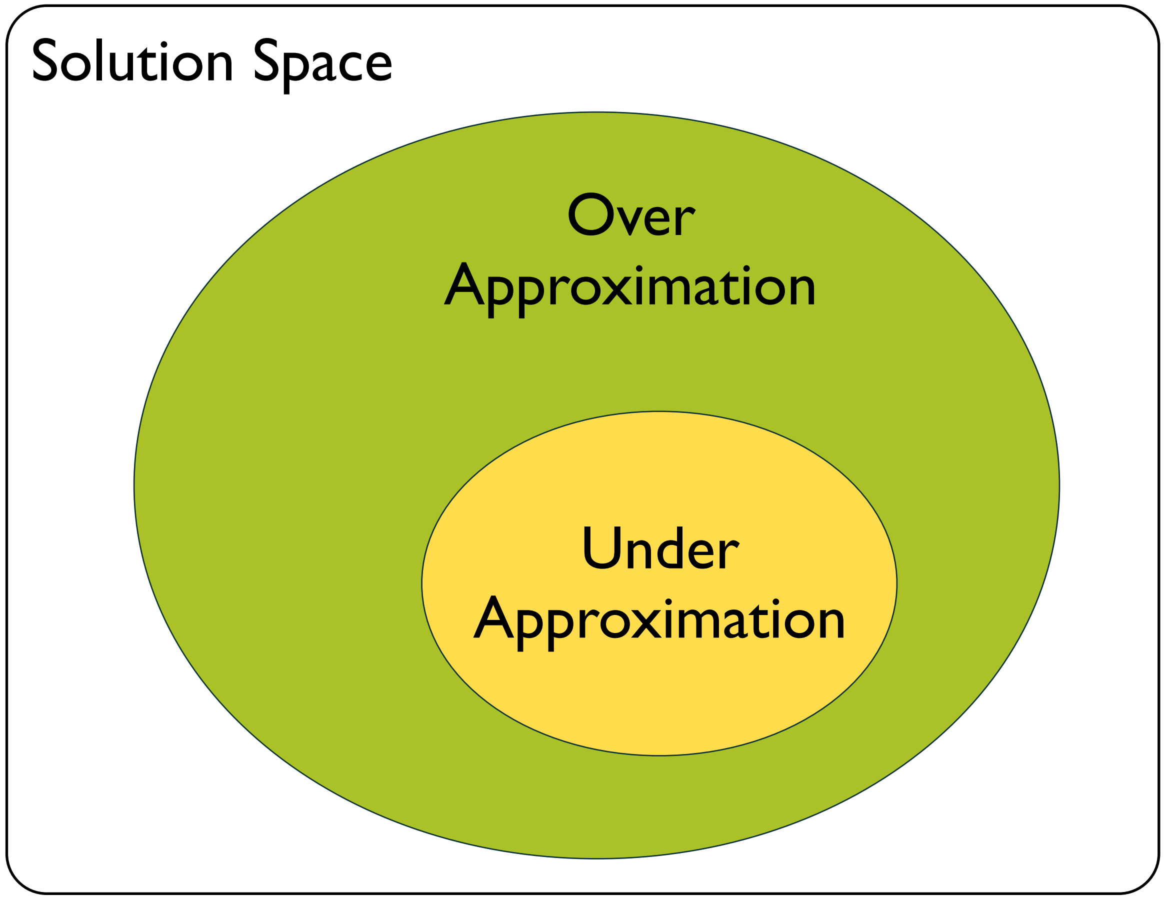
Observations on Algorithmic Learning Approach



Problem #1: **Active Learning**

Invariant approximations are not directly utilized.

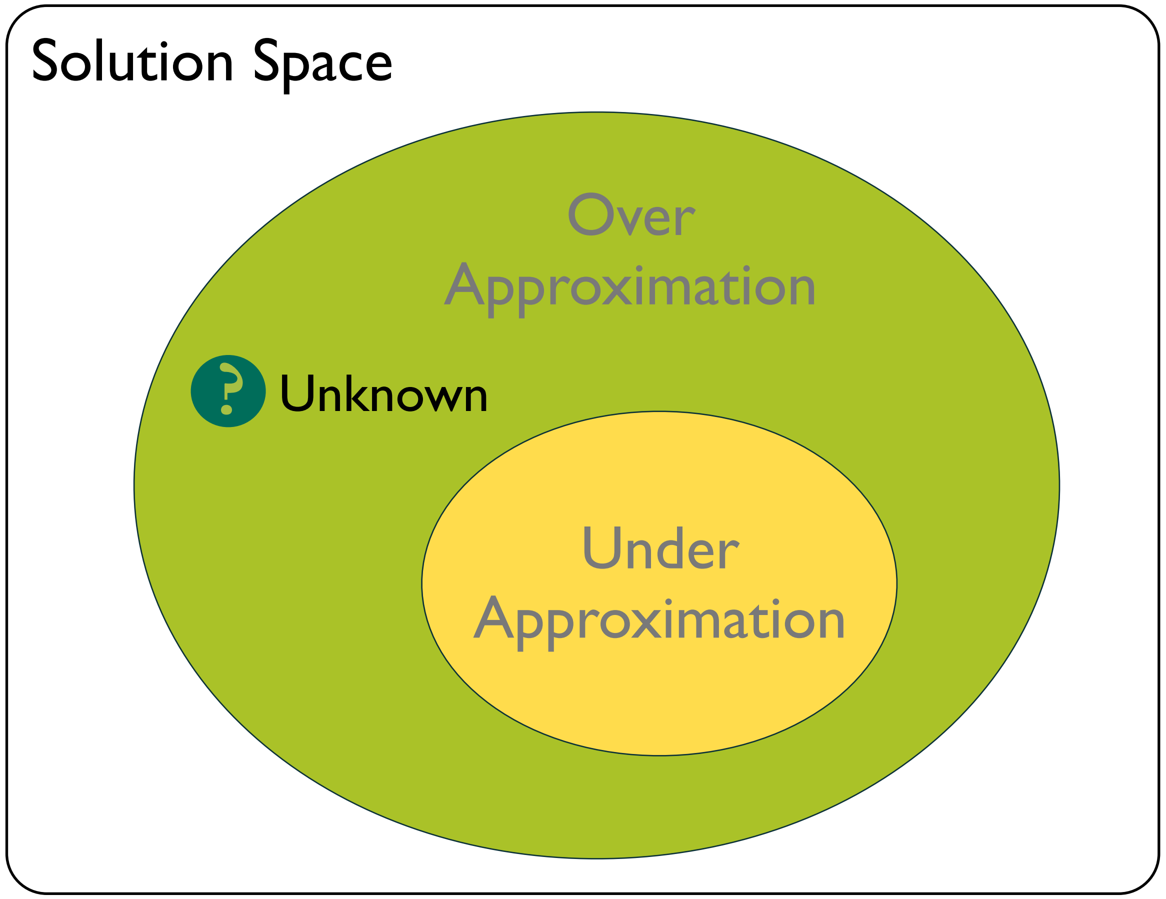
Observations on Algorithmic Learning Approach



Problem #2: **Naive Randomized Mechanism**

Teacher solely relies on naive randomized mechanism when approximations are not helpful.

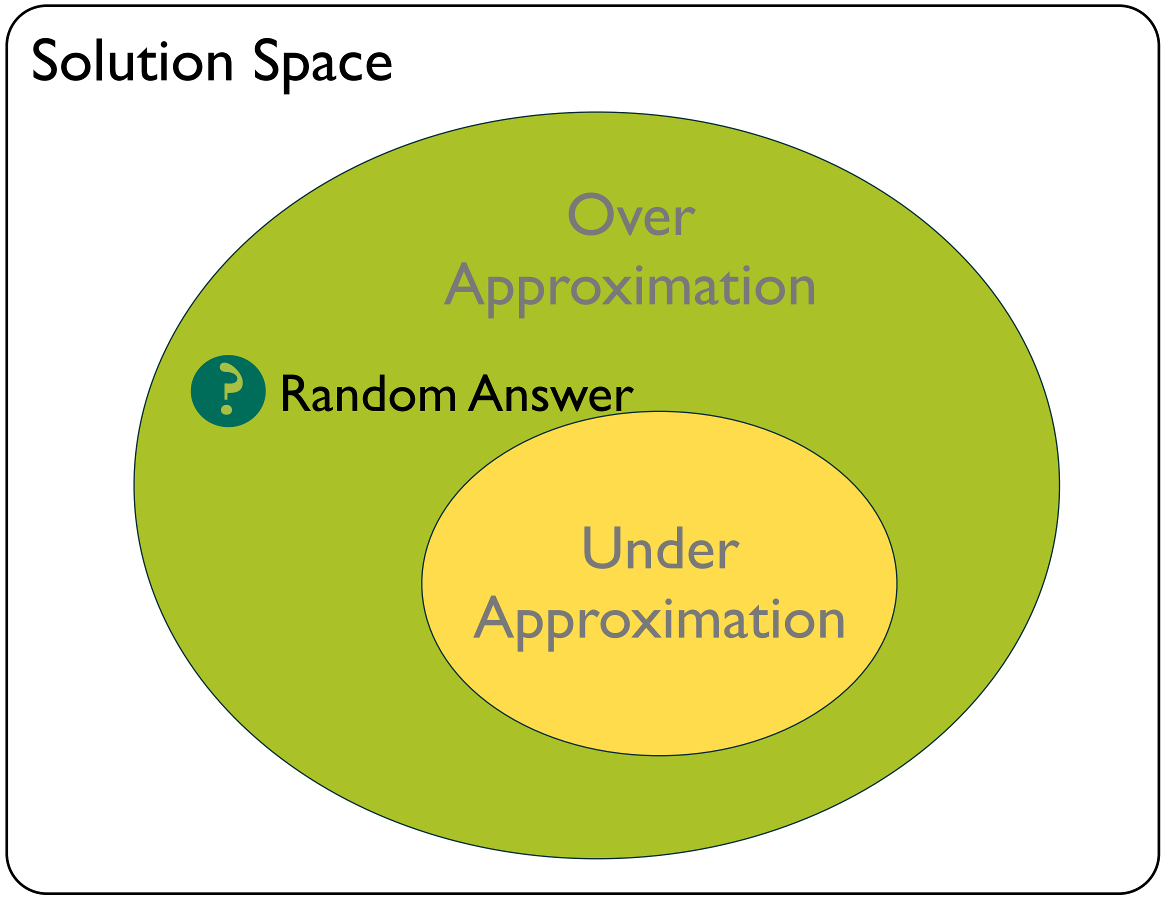
Observations on Algorithmic Learning Approach



Problem #2: **Naive Randomized Mechanism**

Teacher solely relies on naive randomized mechanism when approximations are not helpful.

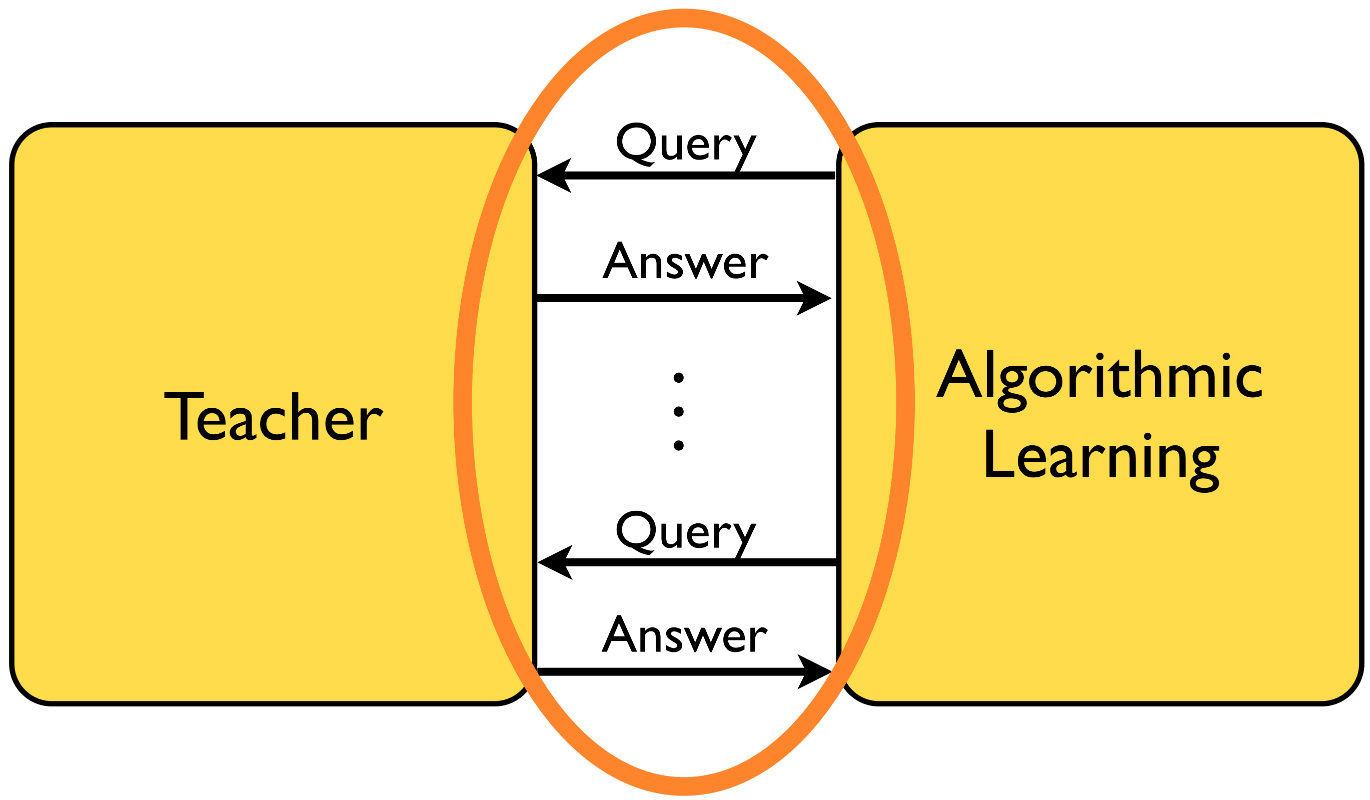
Observations on Algorithmic Learning Approach



Problem #2: Naive Randomized Mechanism

Teacher solely relies on naive randomized mechanism when approximations are not helpful.

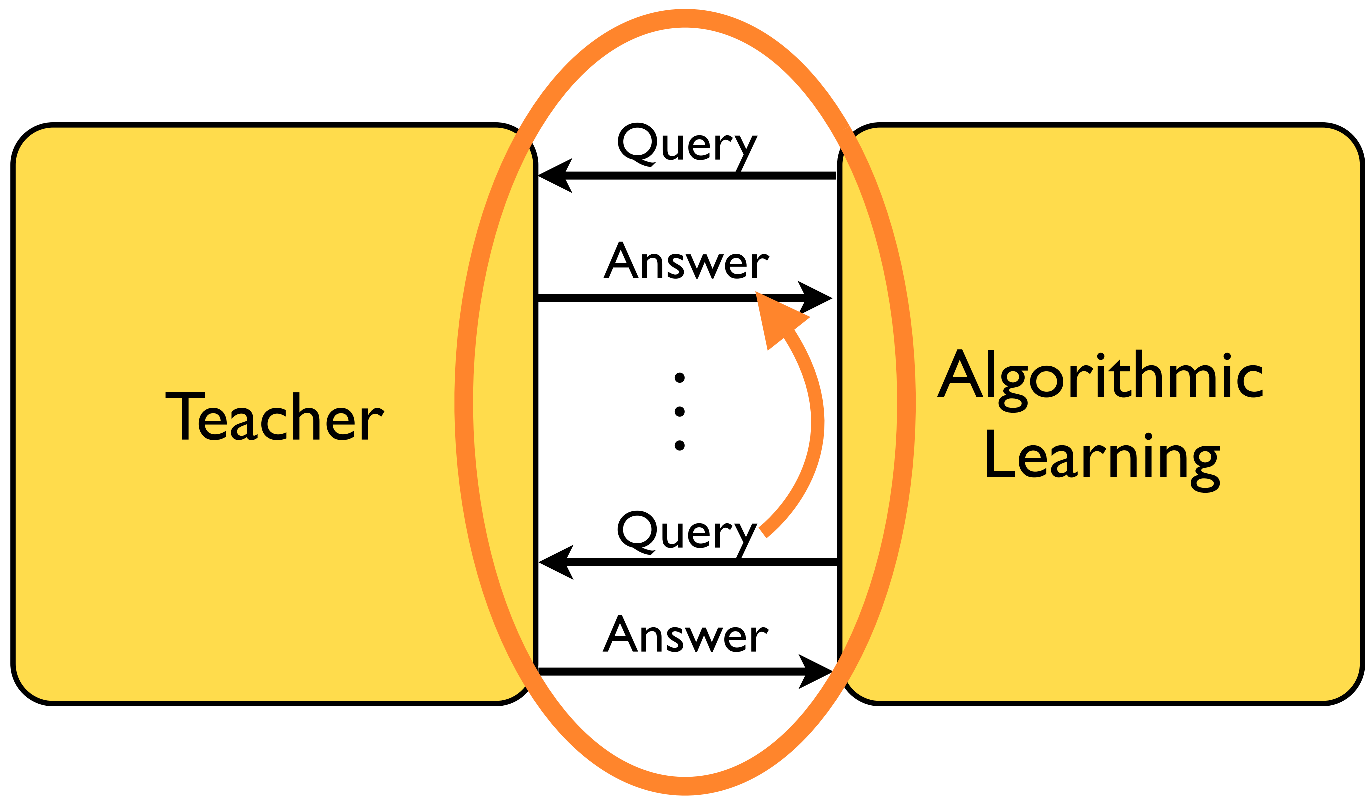
Observations on Algorithmic Learning Approach



Problem #3: **Serialized Query-Answer Process**

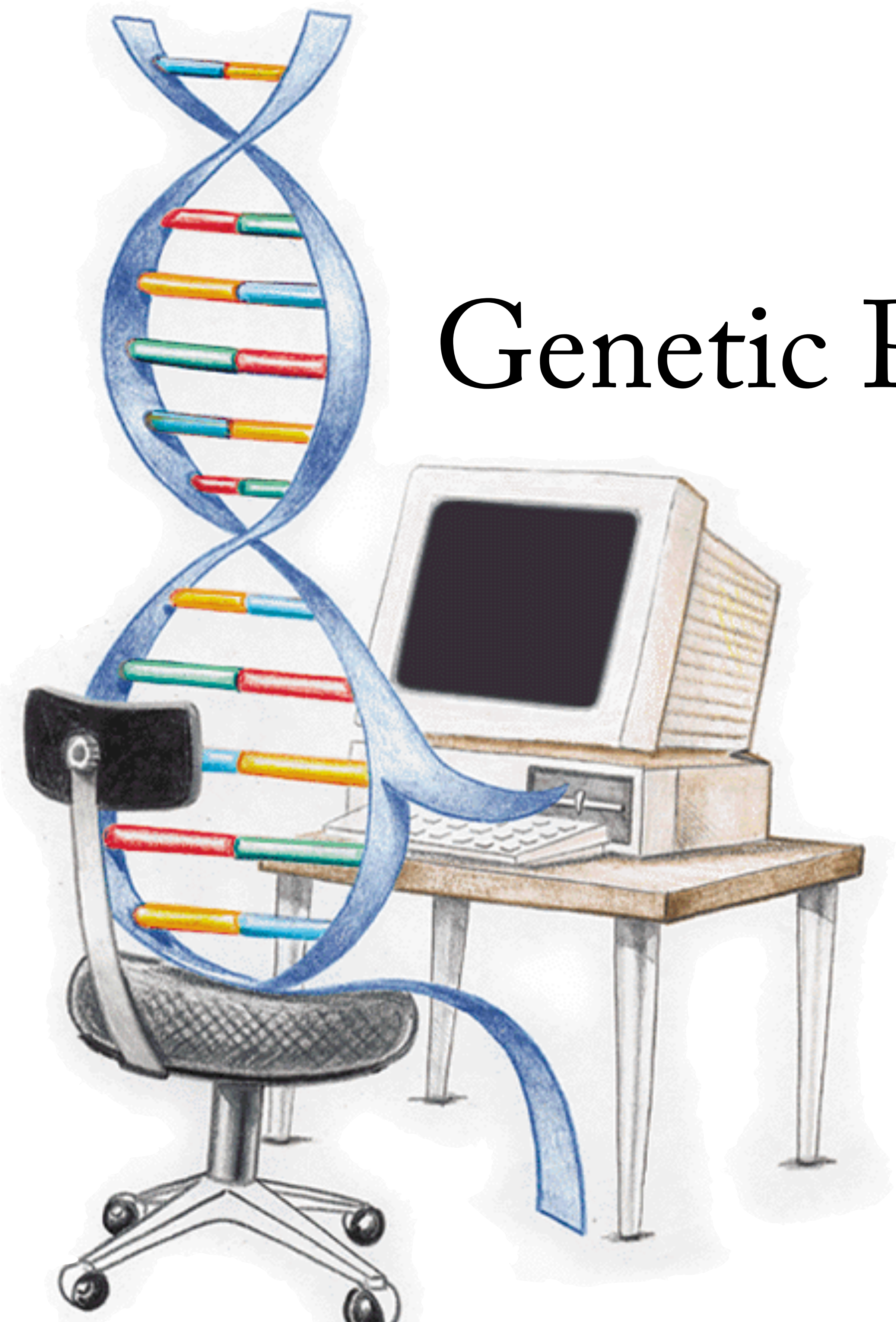
Queries are dependent on the previous query/answer pairs and make parallelization difficult.

Observations on Algorithmic Learning Approach



Problem #3: **Serialized Query-Answer Process**

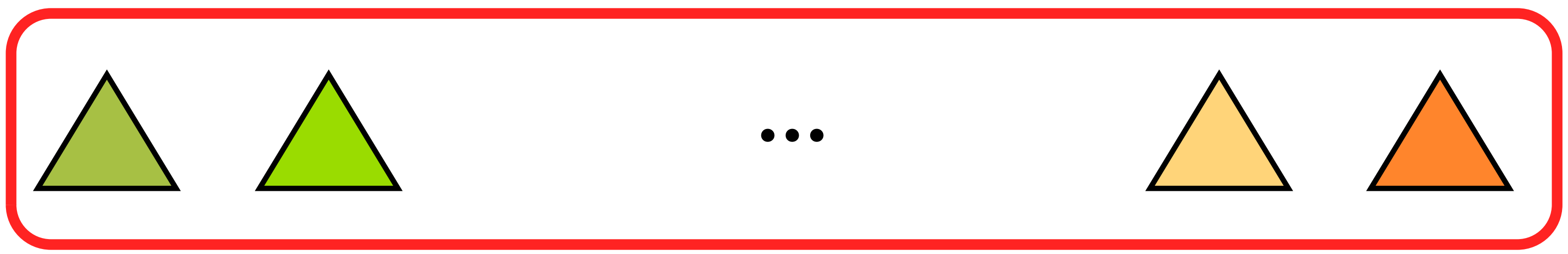
Queries are dependent on the previous query/answer pairs and make parallelization difficult.



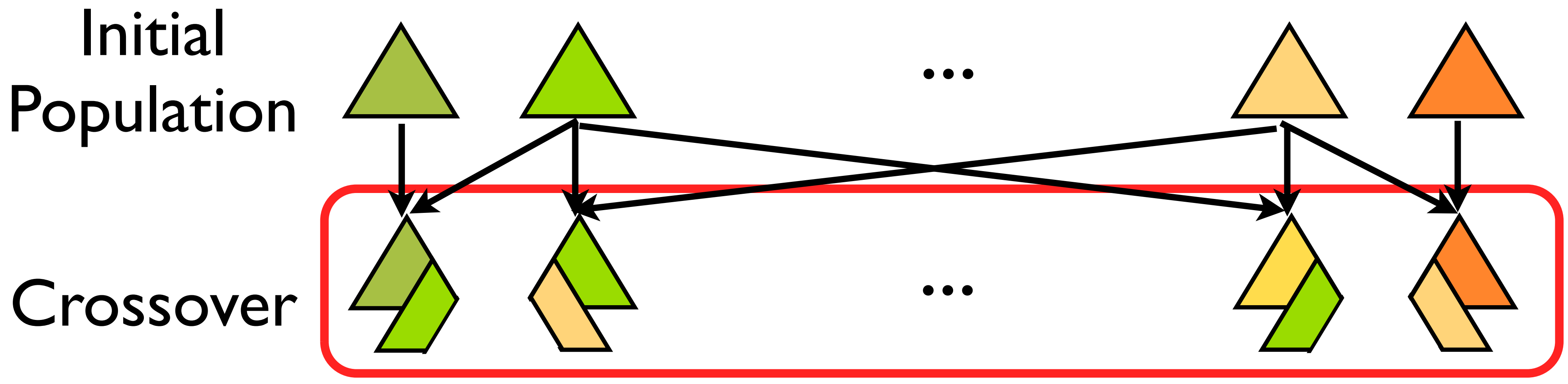
Genetic Programming

Genetic Programming

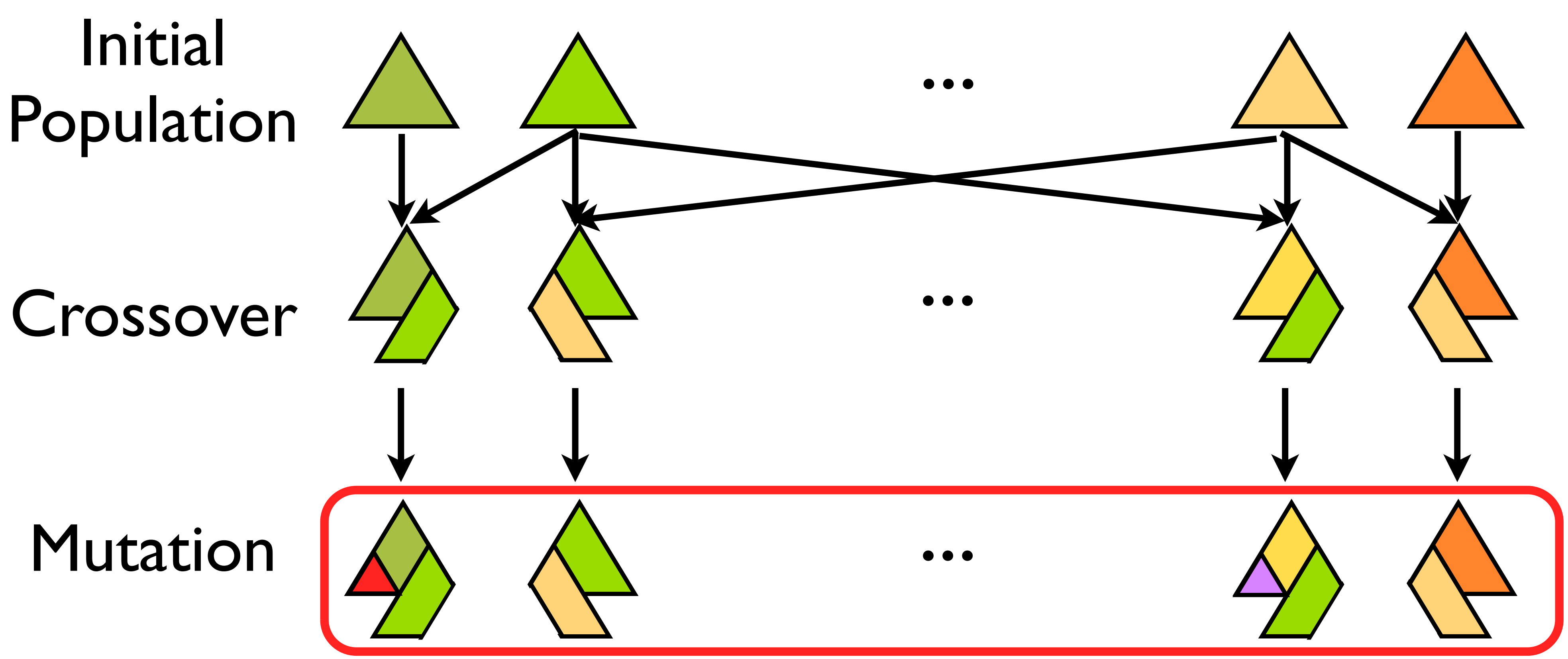
Initial Population



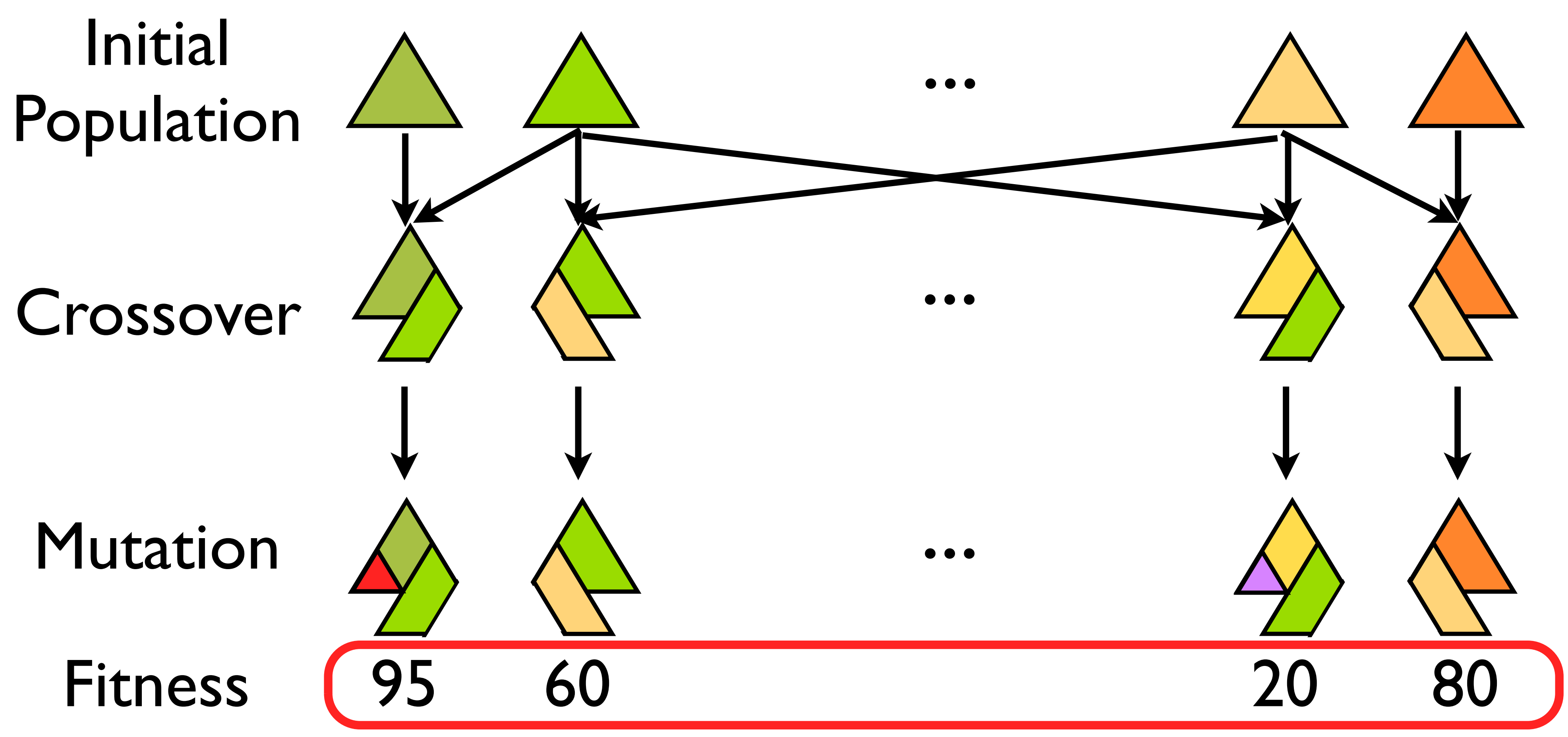
Genetic Programming



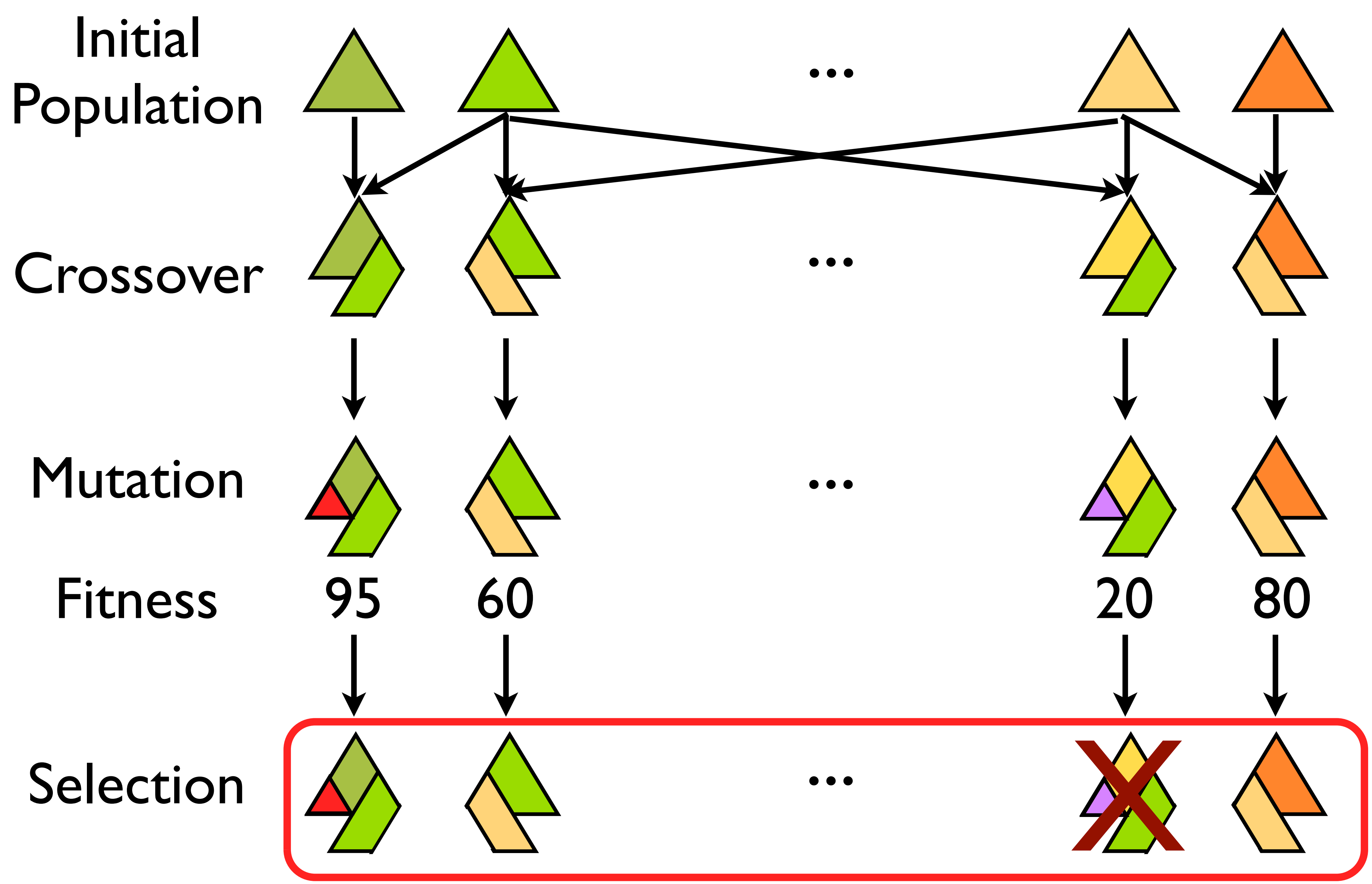
Genetic Programming



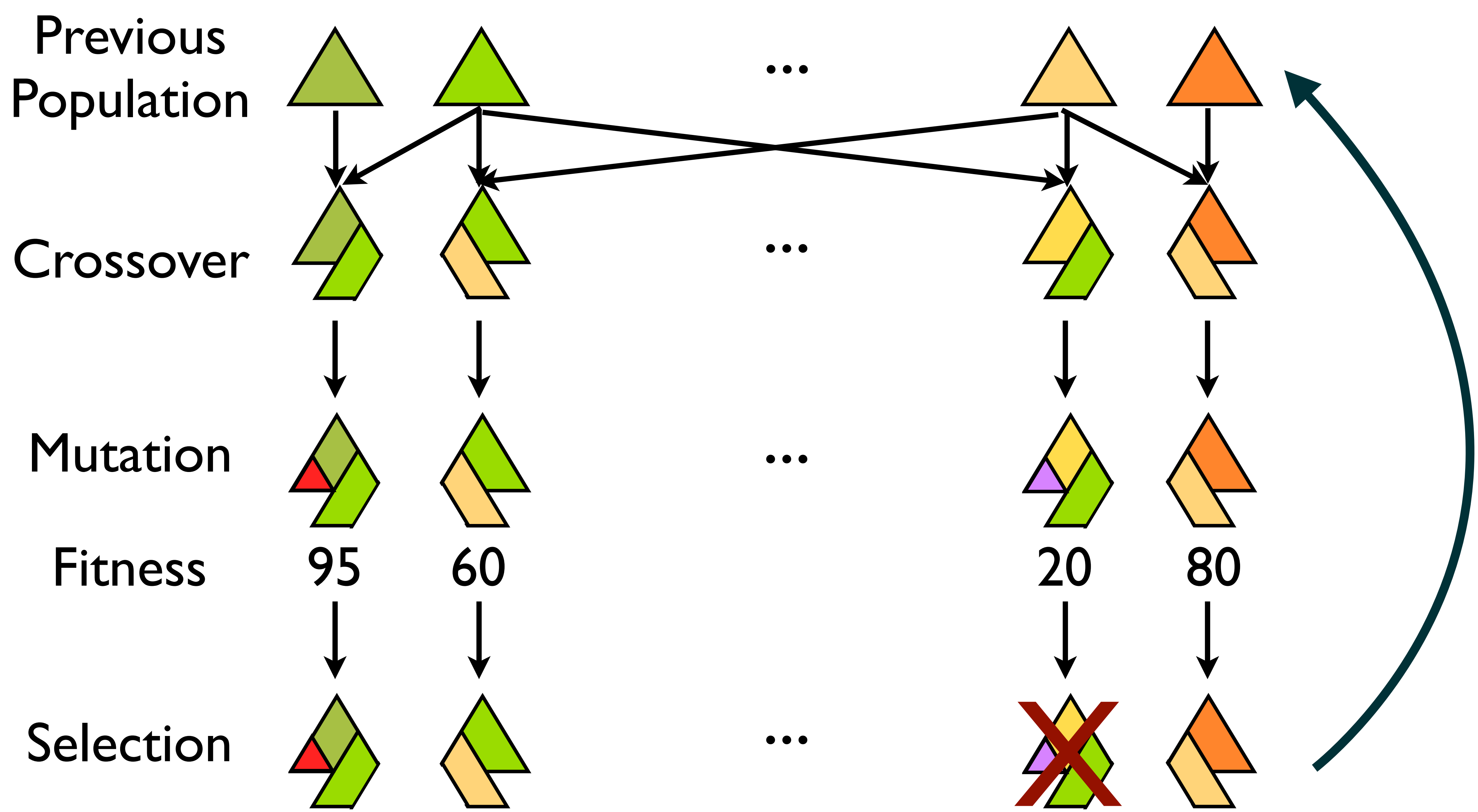
Genetic Programming



Genetic Programming



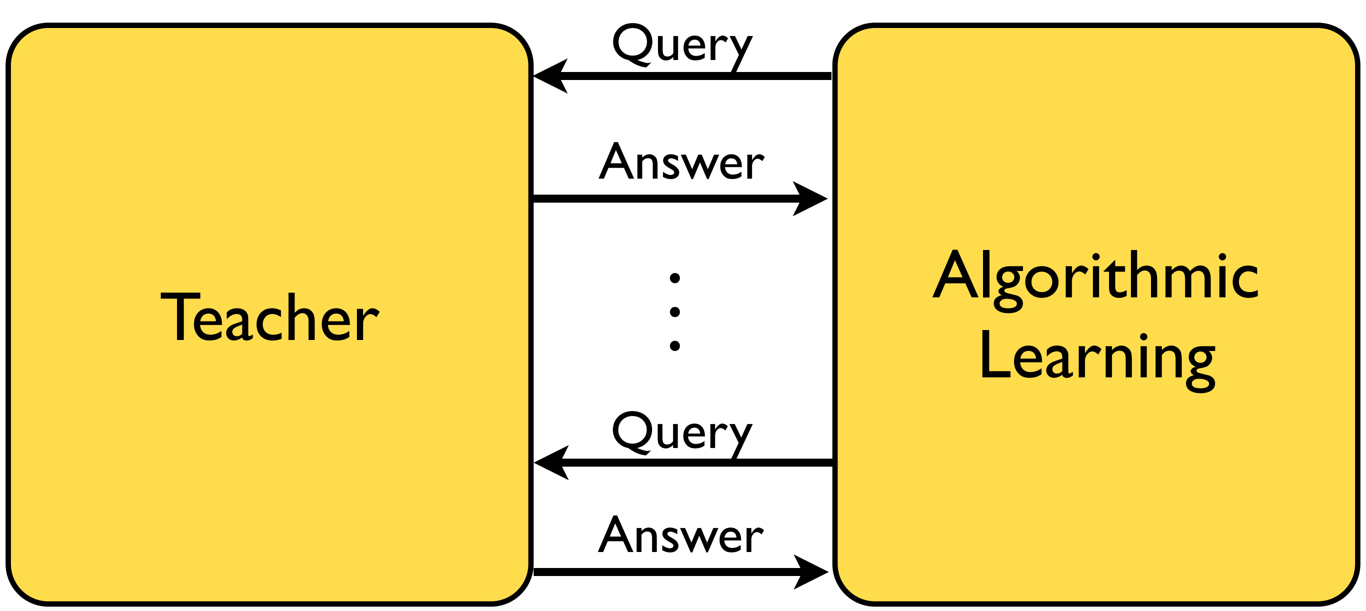
Genetic Programming



Genetic Programming as a Solution

Problem #1: **Active Learning**

Teacher can guide the learner only through the form of answers.



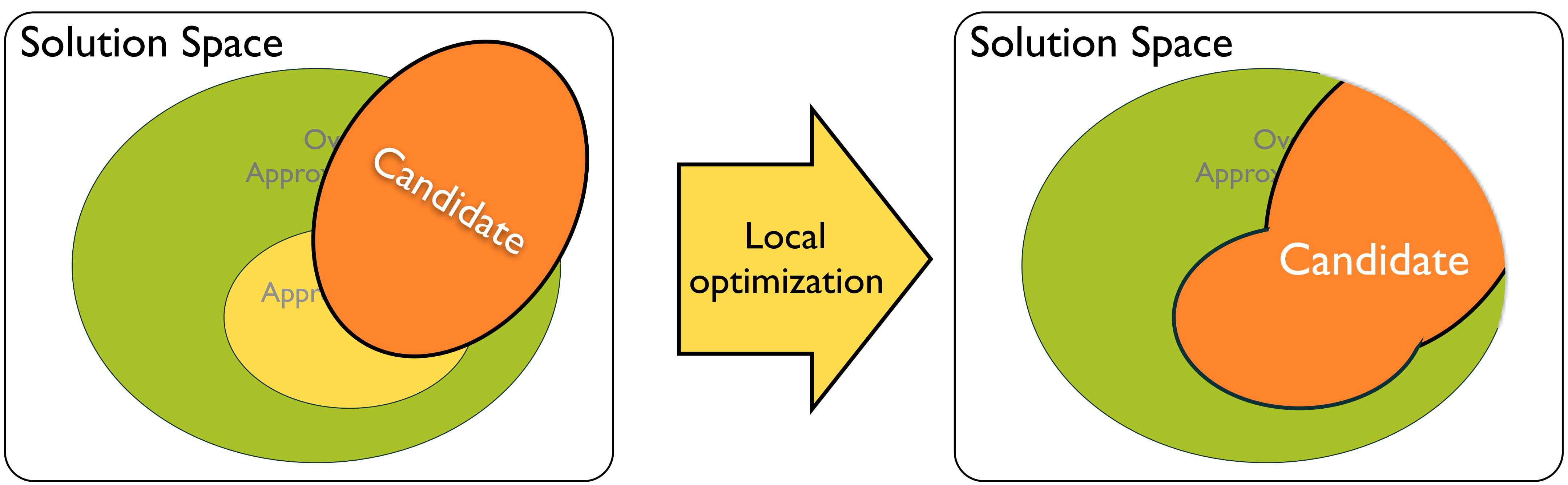
Genetic Programming as a Solution

Problem #1: **Active Learning**

Teacher can guide the learner only through the form of answers.

Solution: Active Teaching

In genetic programming, we can manipulate the formula directly: $c \Rightarrow (c \vee \underline{\iota}) \wedge \bar{\iota}$



Genetic Programming as a Solution

Problem #2: **Naive Randomized Mechanism**

Teacher solely relies on naive randomized mechanism when approximations are not helpful.

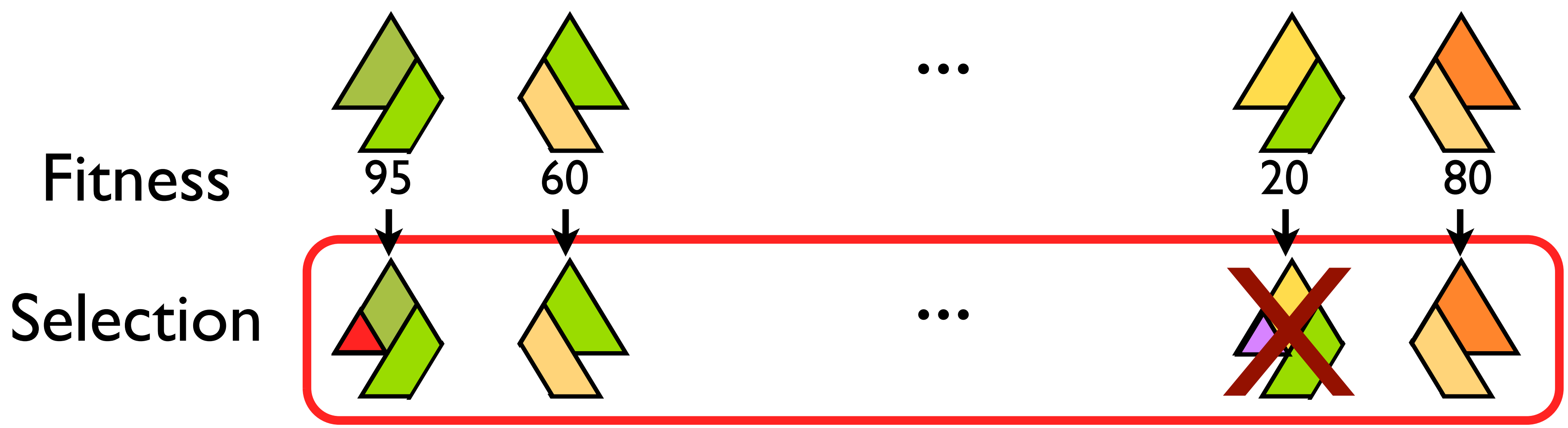
Genetic Programming as a Solution

Problem #2: **Naive Randomized Mechanism**

Teacher solely relies on naive randomized mechanism when approximations are not helpful.

Solution: **Natural Selection**

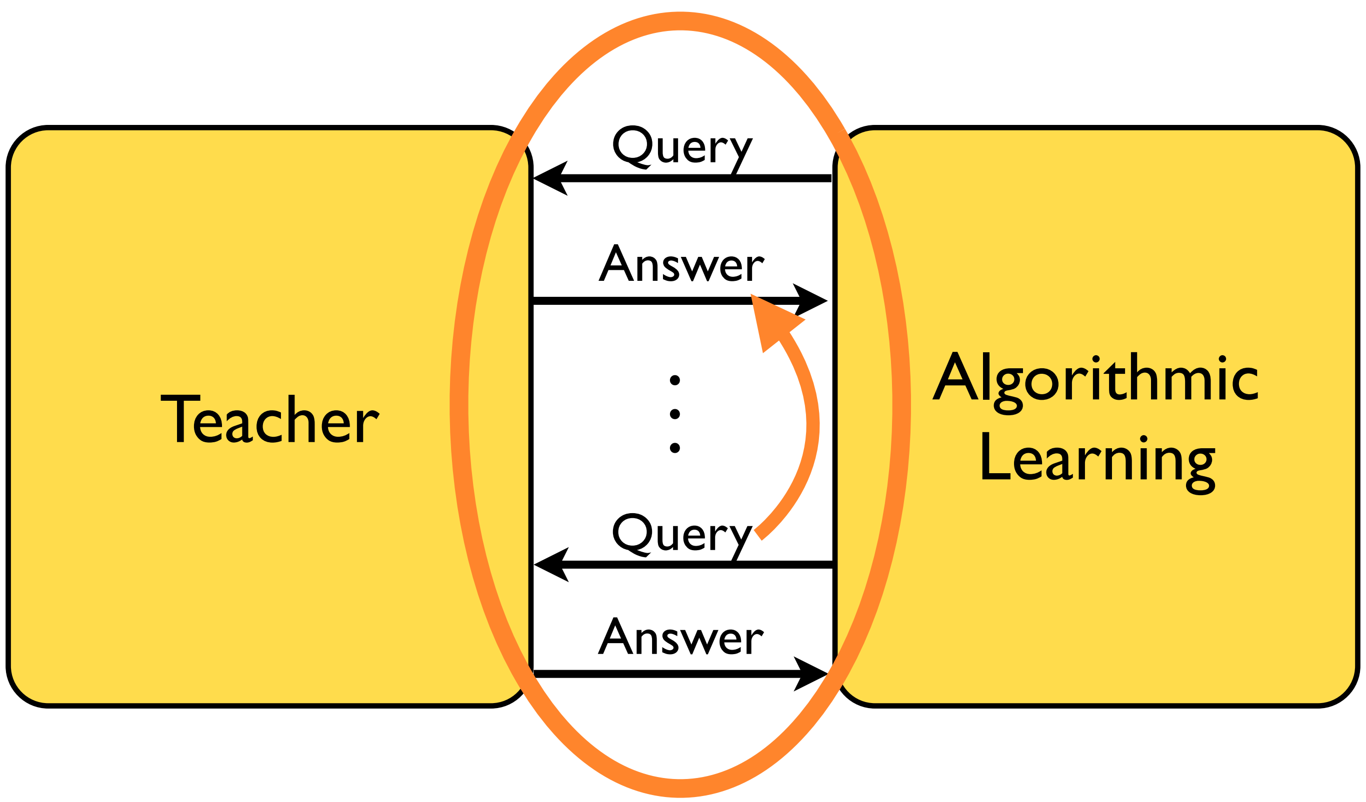
Genetic programming favors better solutions.



Genetic Programming as a Solution

Problem #3: **Serialized Query-Answer Process**

Queries are dependent on the previous query/ answer pairs and make parallelization difficult.

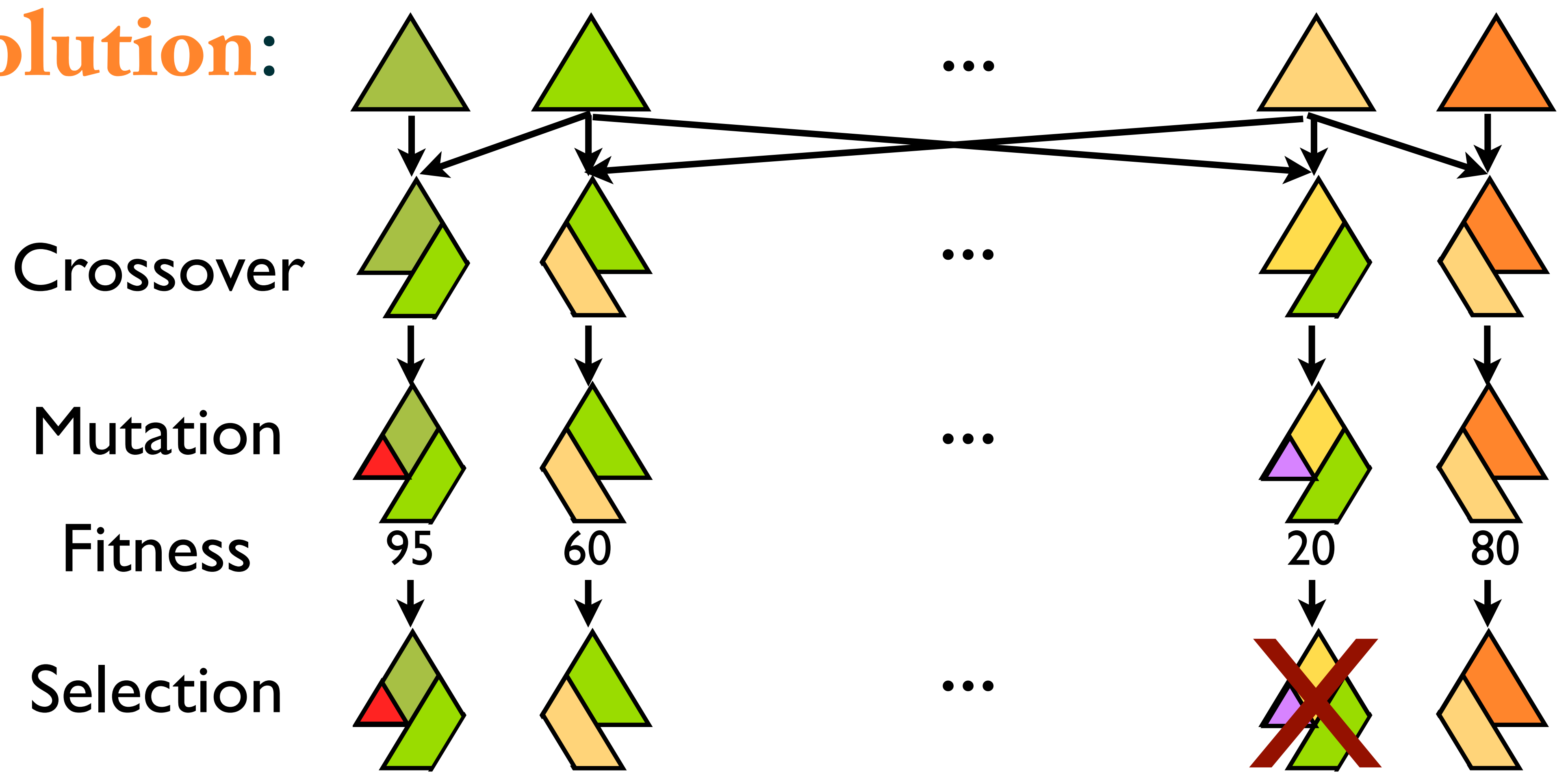


Genetic Programming as a Solution

Problem #3: **Serialized Query-Answer Process**

Queries are dependent on the previous query/ answer pairs and make parallelization difficult.

Solution:

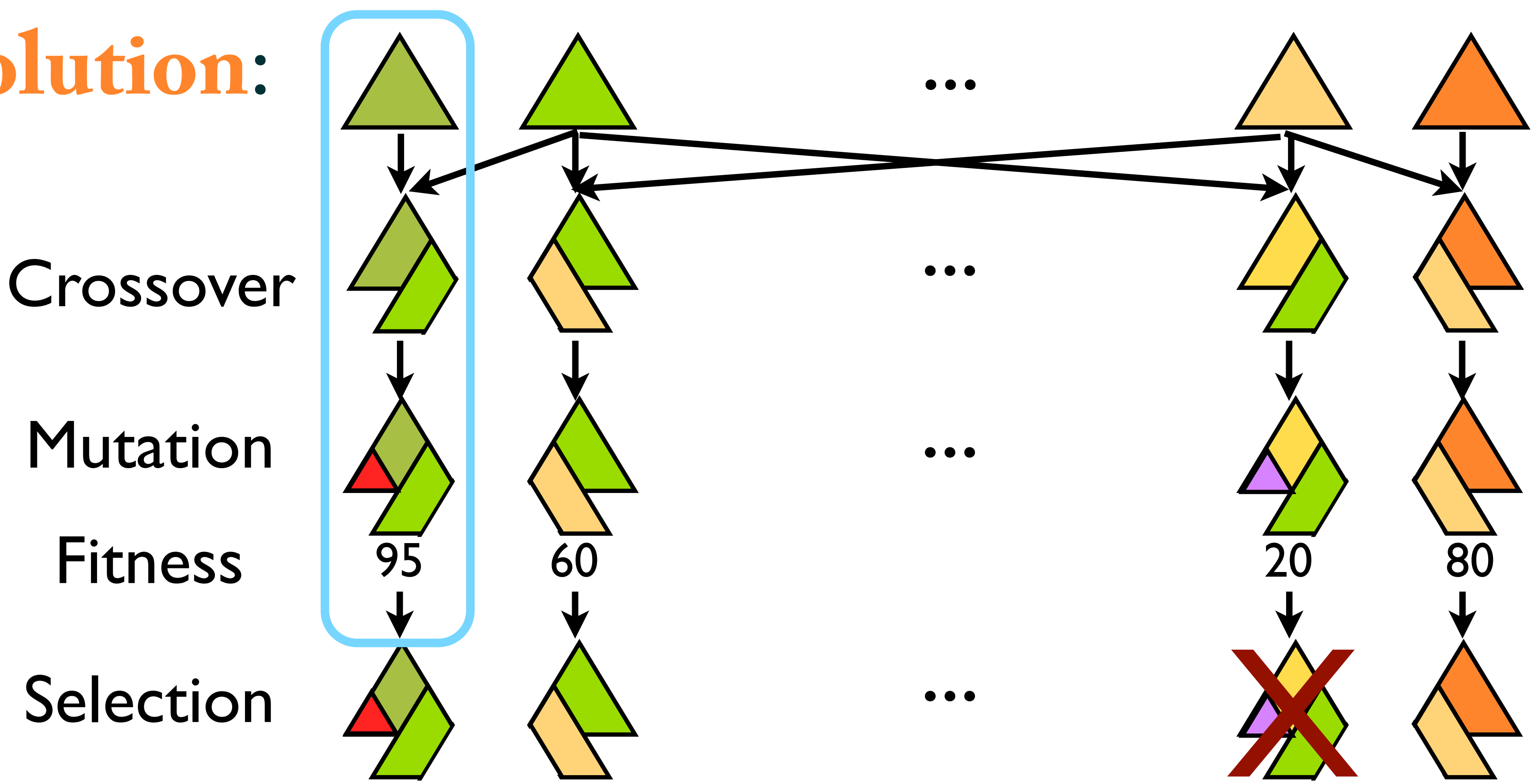


Genetic Programming as a Solution

Problem #3: **Serialized Query-Answer Process**

Queries are dependent on the previous query/answer pairs and make parallelization difficult.

Solution:

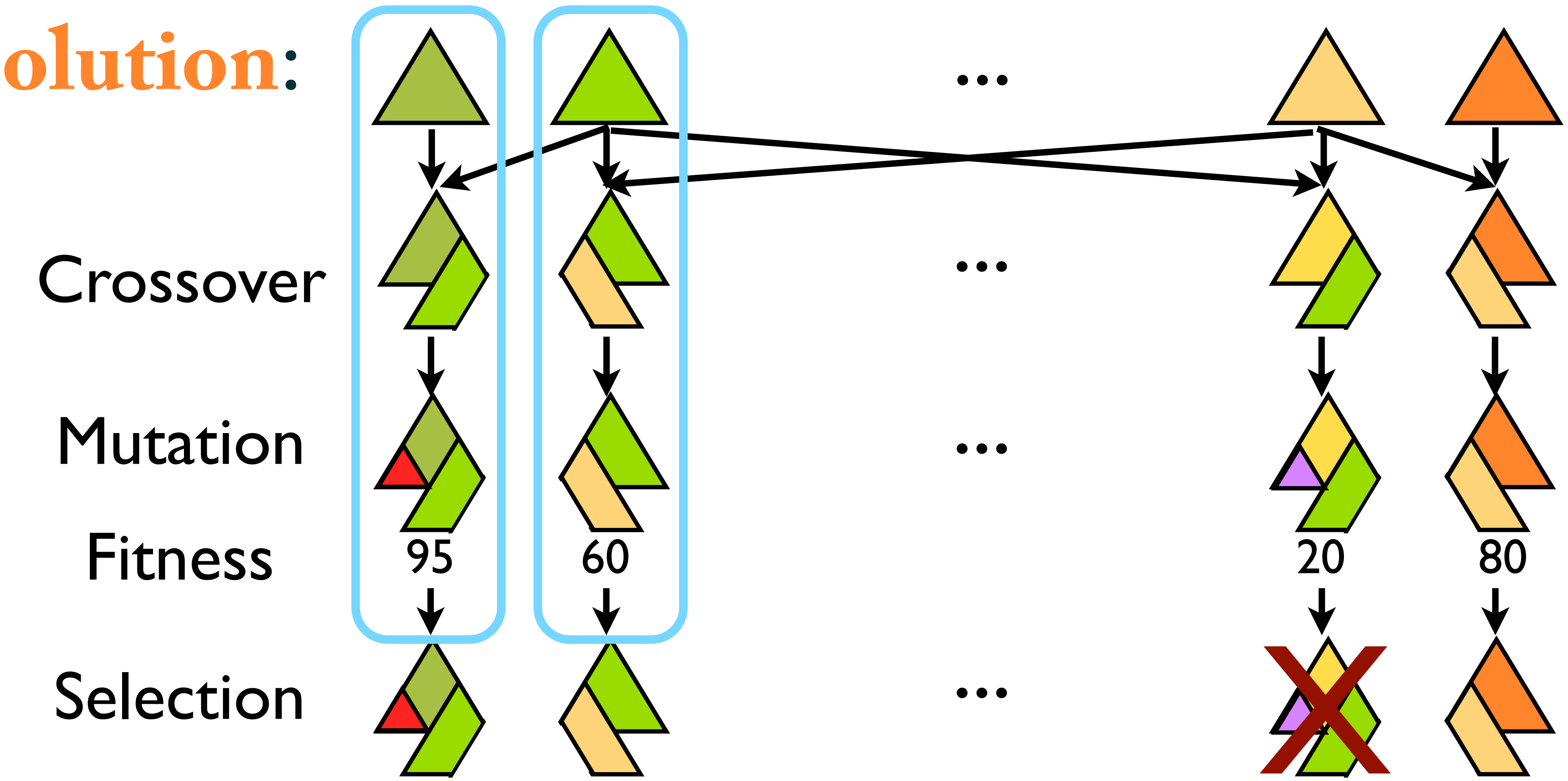


Genetic Programming as a Solution

Problem #3: **Serialized Query-Answer Process**

Queries are dependent on the previous query/ answer pairs and make parallelization difficult.

Solution:

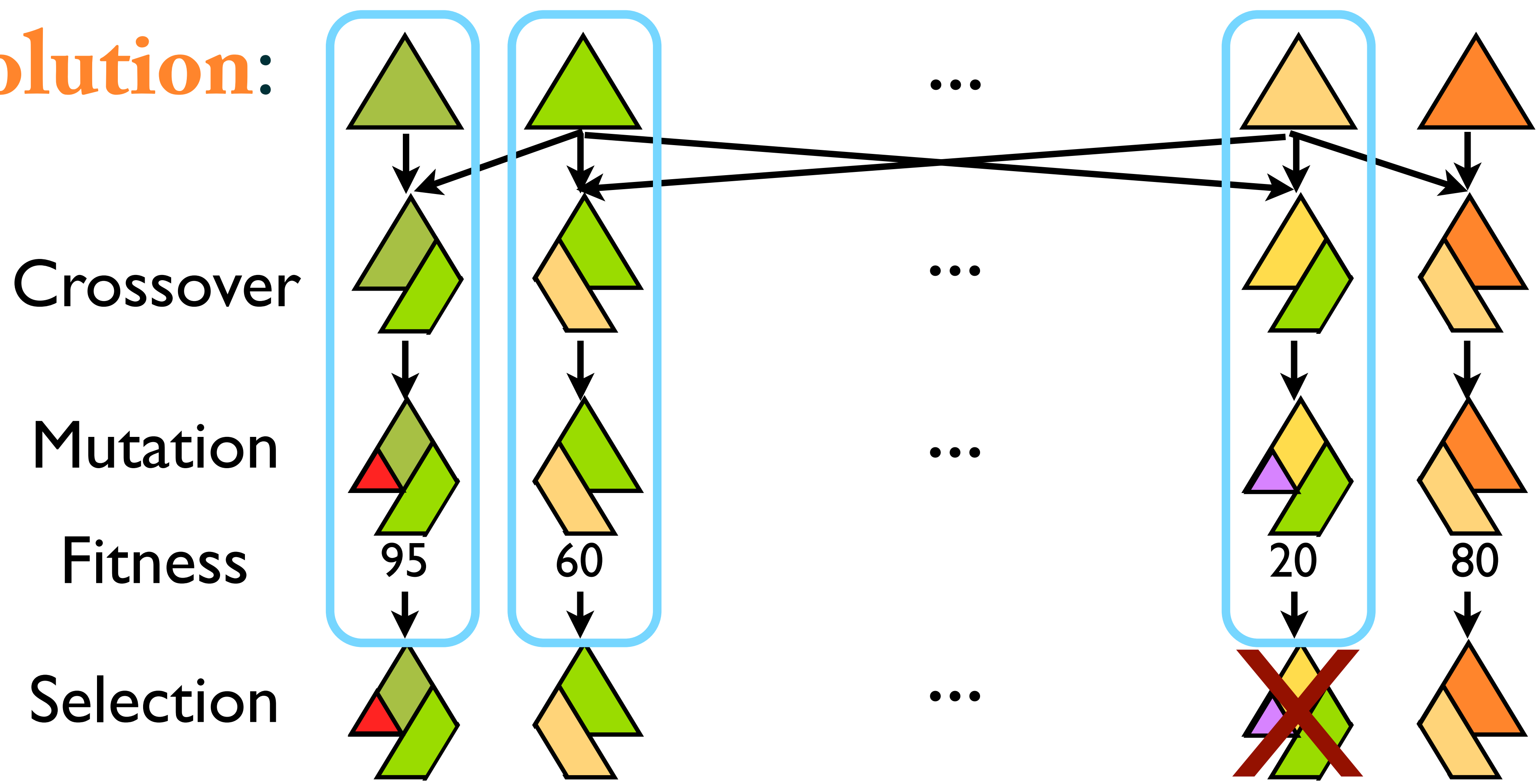


Genetic Programming as a Solution

Problem #3: **Serialized Query-Answer Process**

Queries are dependent on the previous query/answer pairs and make parallelization difficult.

Solution:

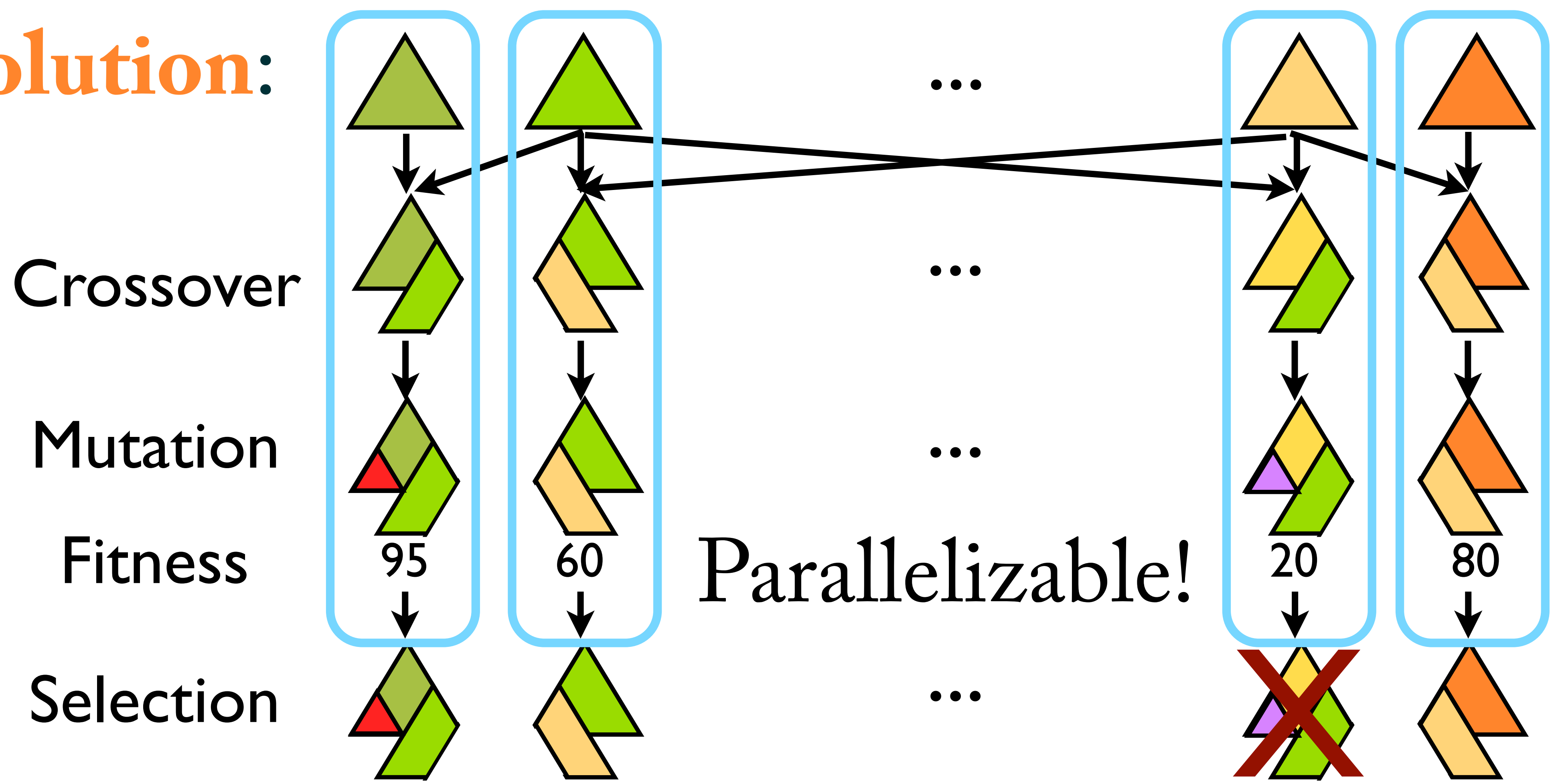


Genetic Programming as a Solution

Problem #3: **Serialized Query-Answer Process**

Queries are dependent on the previous query/ answer pairs and make parallelization difficult.

Solution:

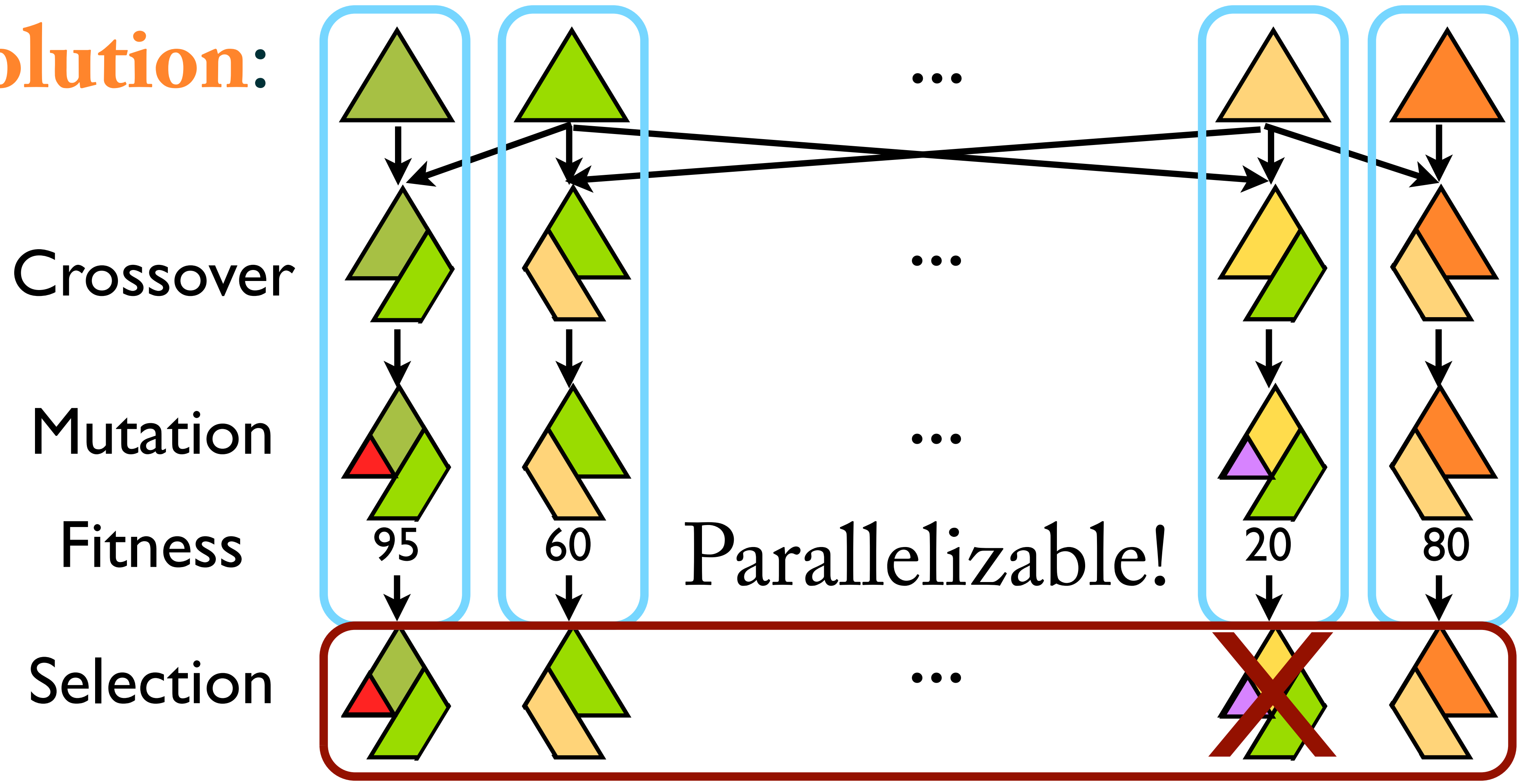


Genetic Programming as a Solution

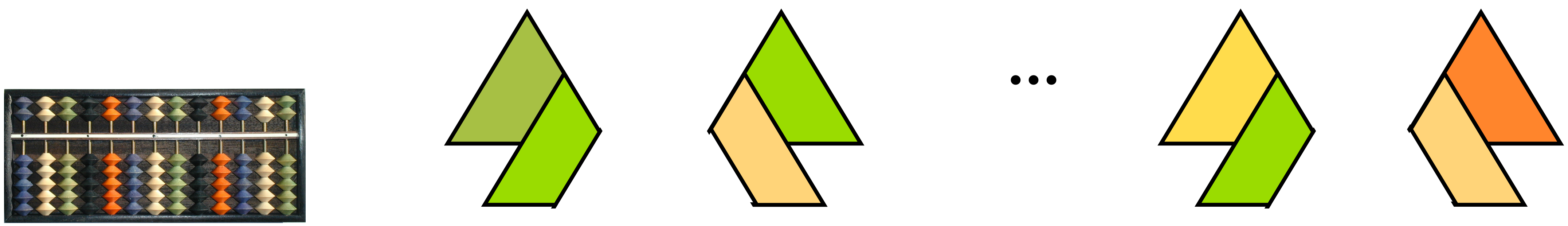
Problem #3: **Serialized Query-Answer Process**

Queries are dependent on the previous query/answer pairs and make parallelization difficult.

Solution:



Key Issue: How to define Fitness Function?



Fitness 95 60 ... 20 80

Role of Fitness Function

For the annotated loop

$$\{\delta\}\ \texttt{while}\ \rho\ \texttt{do}\ S\ \texttt{end}\ \{\epsilon\}$$

Find an invariant I satisfying the following conditions:

(A) $\underline{\iota} \Rightarrow \iota$ (ι holds when entering the loop)

(B) $\iota \wedge \rho \Rightarrow Pre(\iota, S)$ (ι holds at each iteration)

(C) $\iota \Rightarrow \overline{\iota}$ (ι gives ϵ after leaving the loop)

Role of Fitness Function

For the annotated loop

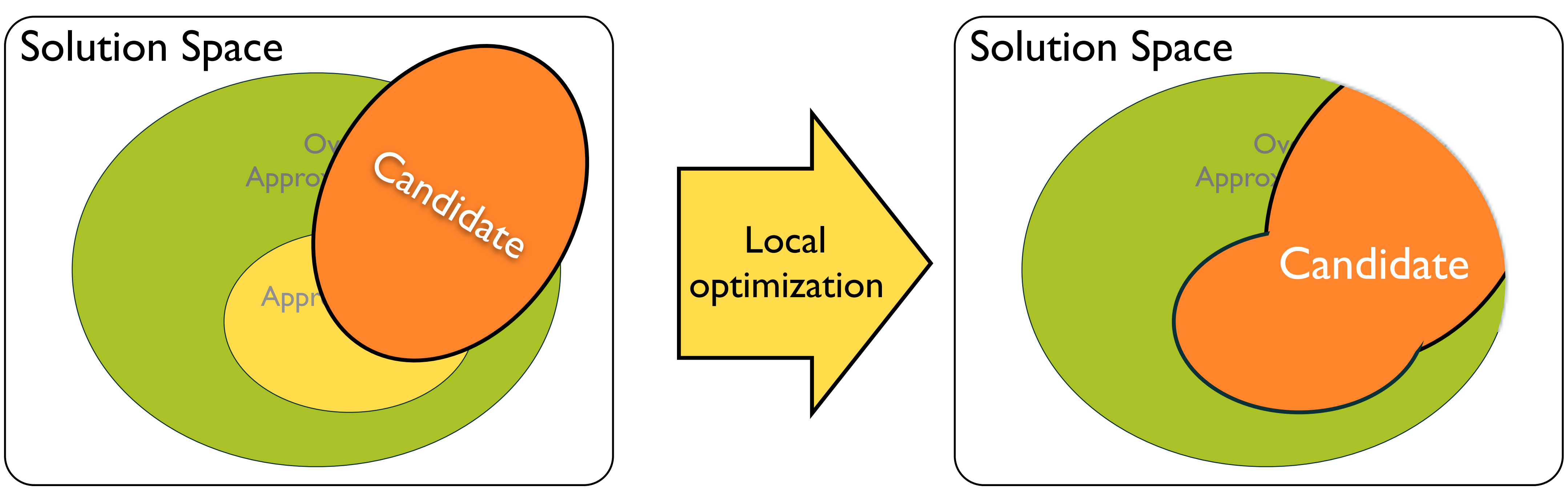
$$\{\delta\}\ \texttt{while}\ \rho\ \texttt{do}\ S\ \texttt{end}\ \{\epsilon\}$$

Find an invariant *I* satisfying the following conditions:

(A) $\underline{\iota} \Rightarrow \iota$ (ι holds when entering the loop)

(B) $\iota \wedge \rho \Rightarrow Pre(\iota, S)$ (ι holds at each iteration)

(C) $\iota \Rightarrow \overline{\iota}$ (ι gives ϵ after leaving the loop)



Role of Fitness Function

For the annotated loop

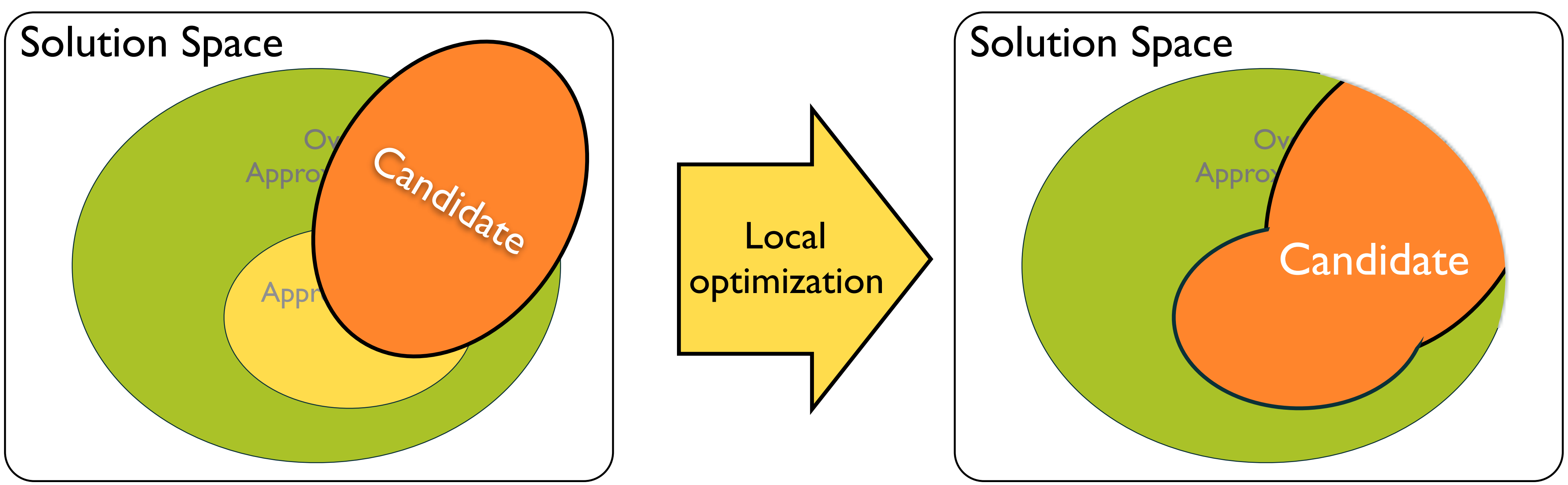
$$\{\delta\}\ \texttt{while}\ \rho\ \texttt{do}\ S\ \texttt{end}\ \{\epsilon\}$$

Find an invariant I satisfying the following conditions:

✓ (A) $\underline{\iota} \Rightarrow \iota$ (ι holds when entering the loop)

(B) $\iota \wedge \rho \Rightarrow Pre(\iota, S)$ (ι holds at each iteration)

✓ (C) $\iota \Rightarrow \overline{\iota}$ (ι gives ϵ after leaving the loop)



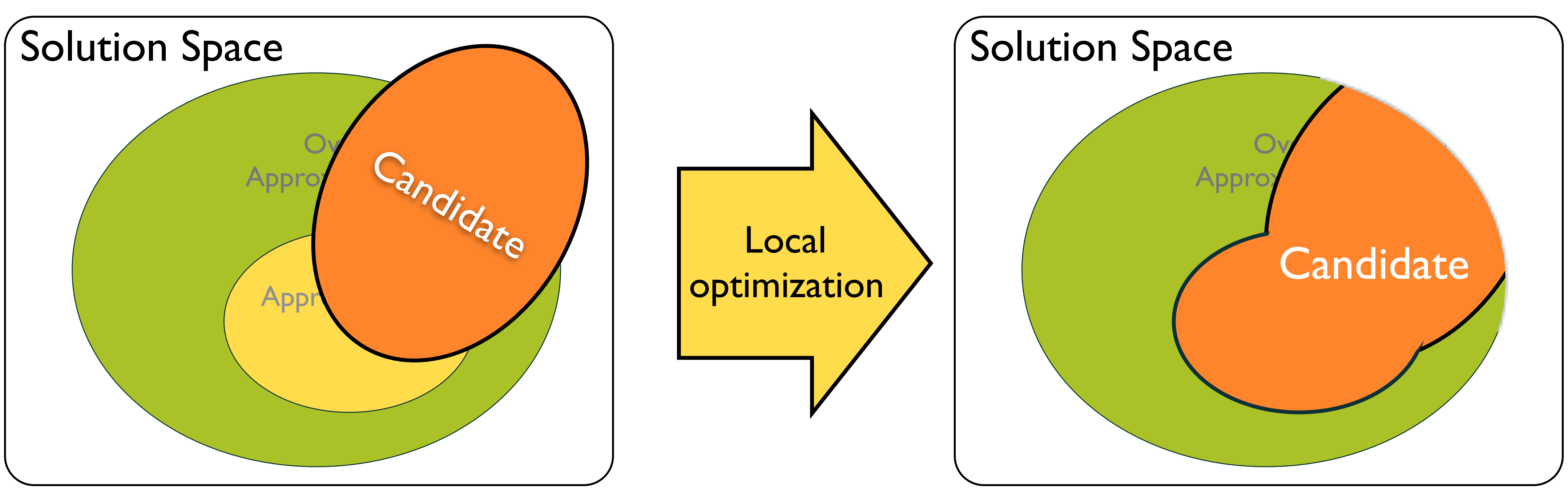
Role of Fitness Function

For the annotated loop

$$\{\delta\}\ \texttt{while}\ \rho\ \texttt{do}\ S\ \texttt{end}\ \{\epsilon\}$$

Find an invariant I satisfying the following conditions:

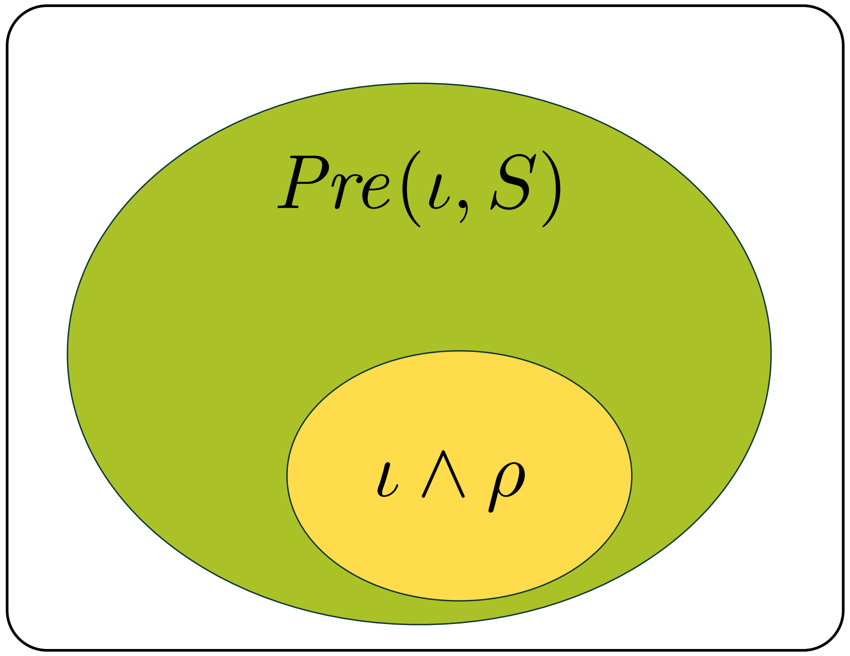
- ✓ (A) $\underline{\iota} \Rightarrow \iota$ (ι holds when entering the loop)
- (B) $\iota \wedge \rho \Rightarrow Pre(\iota, S)$ (ι holds at each iteration)
- ✓ (C) $\iota \Rightarrow \overline{\iota}$ (ι gives ϵ after leaving the loop)



Role of Fitness Function

(B) $\iota \wedge \rho \Rightarrow Pre(\iota, S)$ (ι holds at each iteration)

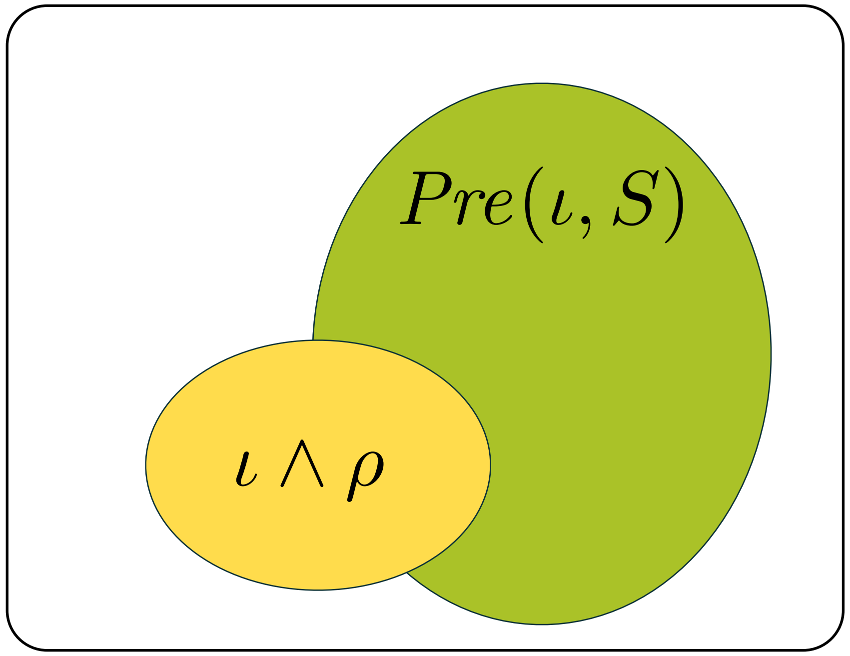
If (B) holds, then we find an invariant.



Role of Fitness Function

(B) $\iota \wedge \rho \Rightarrow Pre(\iota, S)$ (ι holds at each iteration)

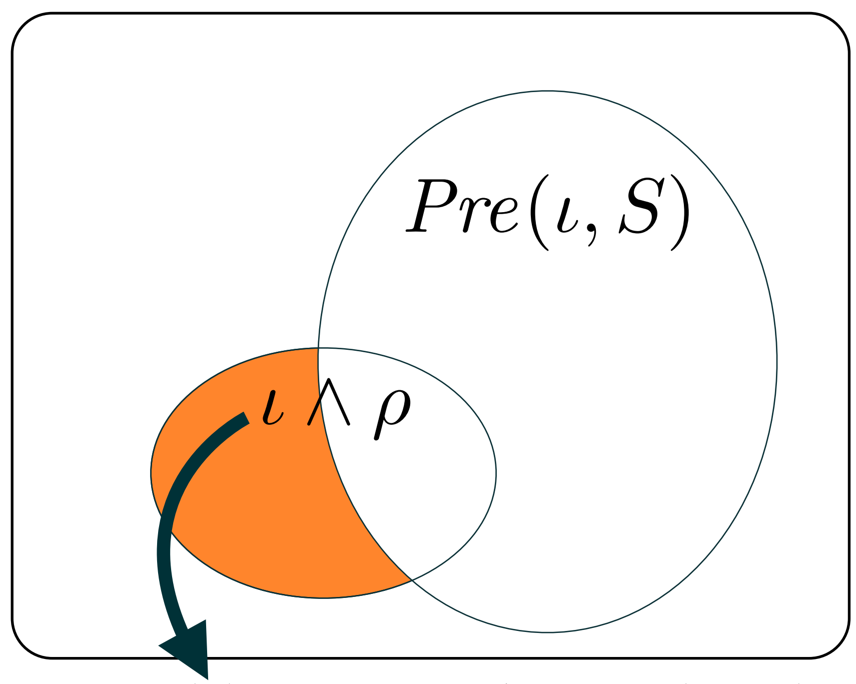
If (B) does not hold,
we need to quantify how bad the candidate is.



Role of Fitness Function

(B) $\iota \wedge \rho \Rightarrow Pre(\iota, S)$ (ι holds at each iteration)

If (B) does not hold,
we need to quantify how bad the candidate is.



$$Diff(\iota) = \iota \wedge \rho \setminus Pre(\iota, S)$$

Fitness Function Property

$$Diff(\iota_1) \subseteq Diff(\iota_2) \Rightarrow Fit(\iota_1) \leq Fit(\iota_2)$$

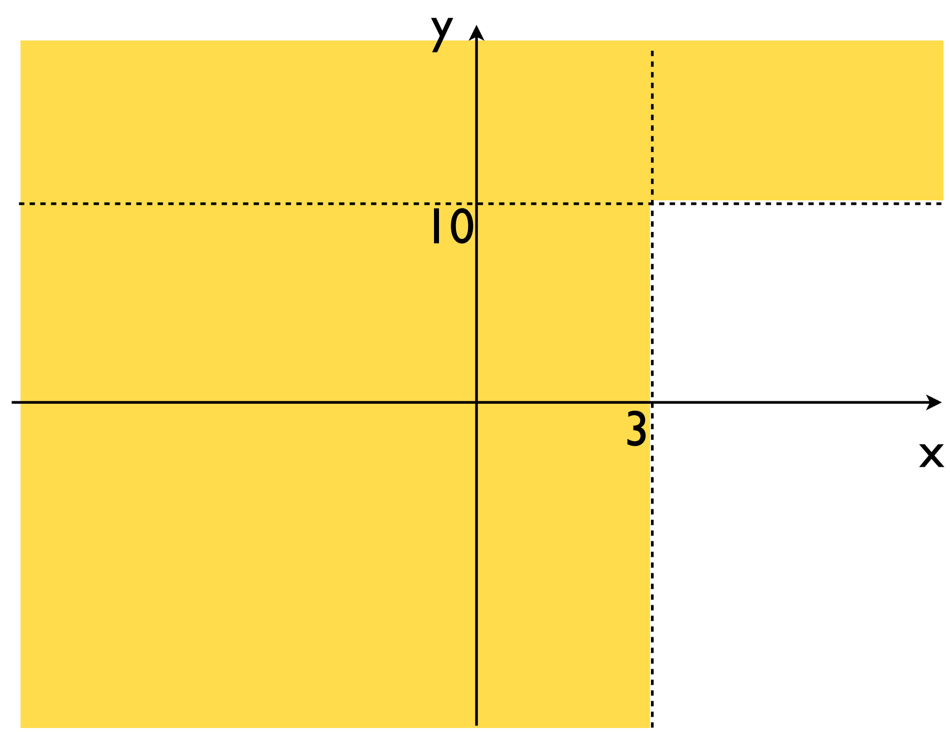
Infeasible Fitness Function

Count the number of models in **concrete domain**.

Example

$$Diff(\iota) = x < 3 \vee y > 10$$

$$Fit(\iota) = |\{(2, 11), (10, 15), (0, 0), \dots\}|$$



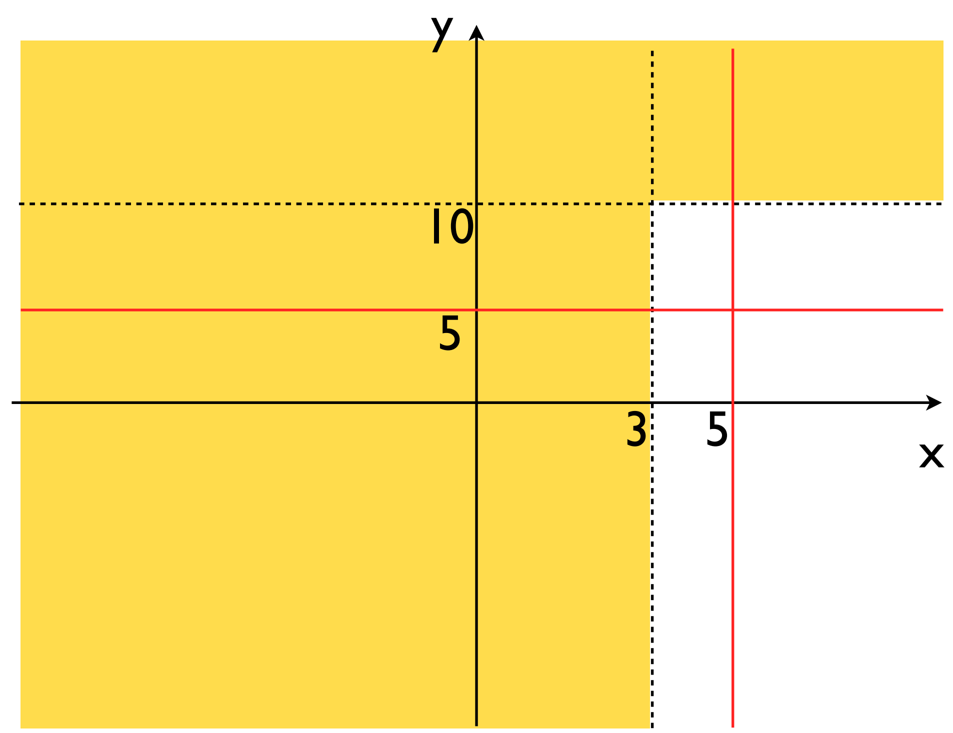
Can **not** count them all in general.

Feasible Fitness Function

Count the number of models in **abstract domain**.

Example

$$Diff(\iota) = x < 3 \vee y > 10$$



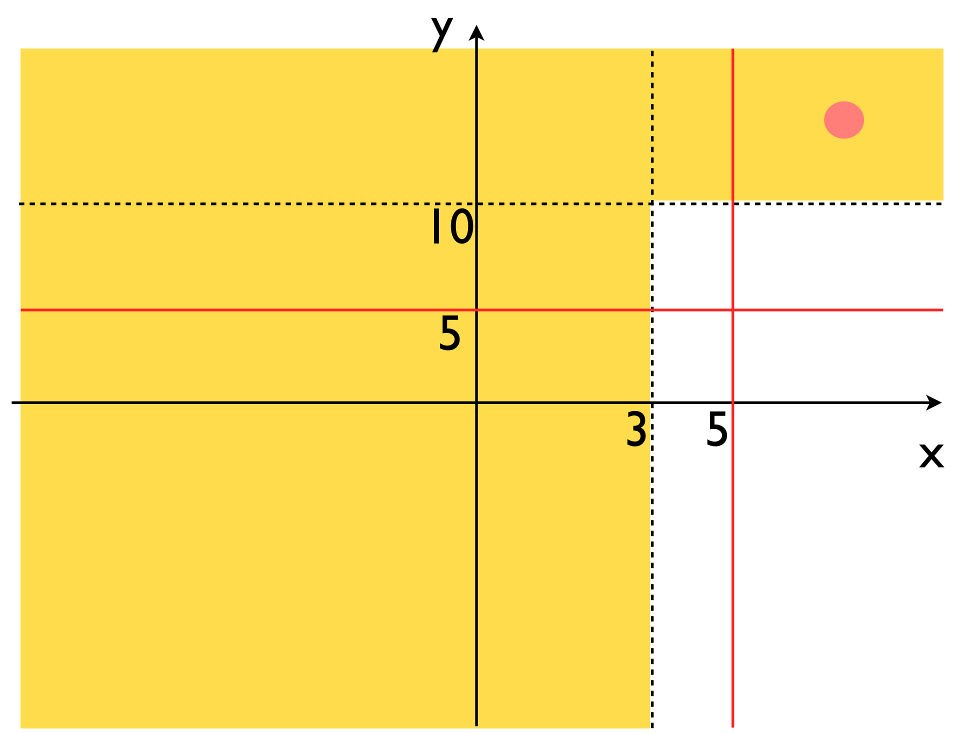
Feasible Fitness Function

Count the number of models in **abstract domain**.

Example

$$Diff(\iota) = x < 3 \vee y > 10$$

$$\text{SMT: } (10, 15) \models Diff(\iota)$$



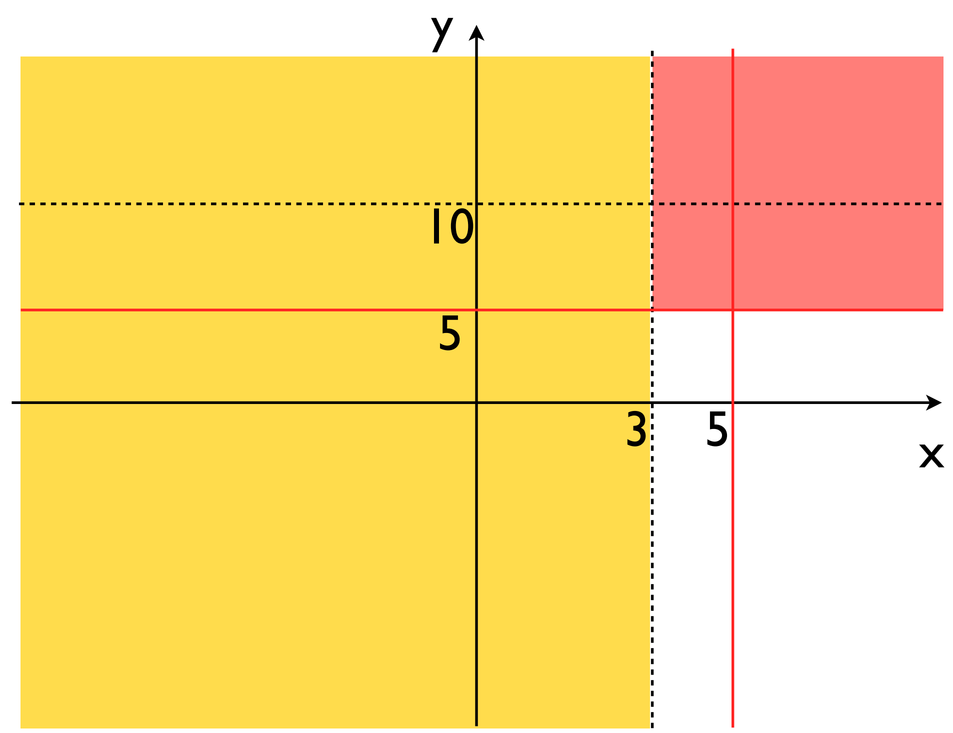
Feasible Fitness Function

Count the number of models in **abstract domain**.

Example

$$Diff(\iota) = x < 3 \vee y > 10$$

$$\text{SMT: } (10, 15) \models Diff(\iota)$$



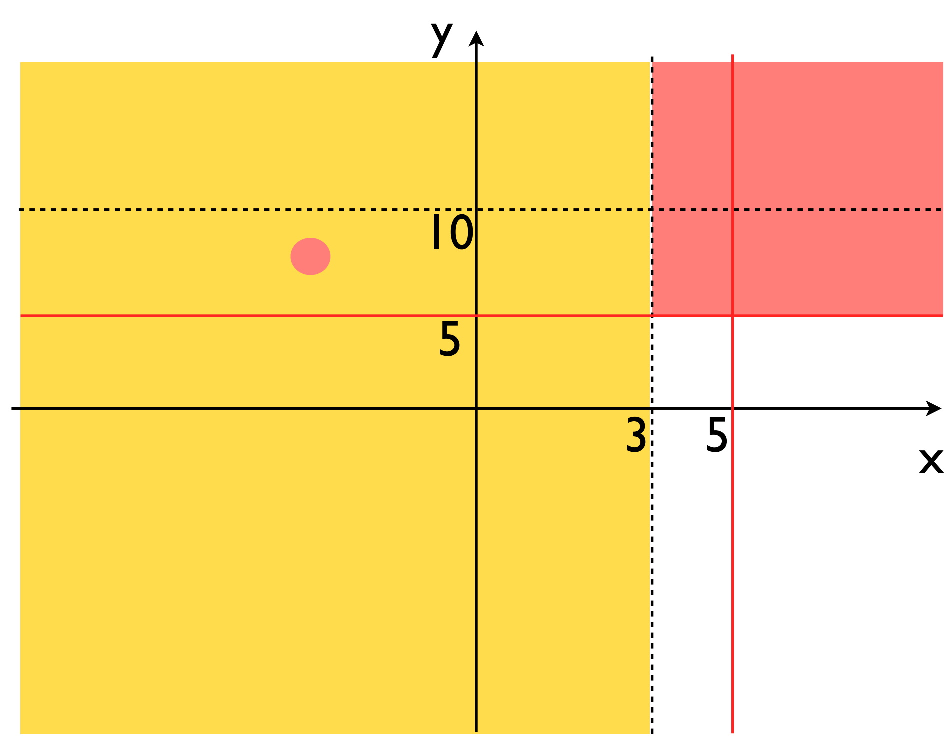
Feasible Fitness Function

Count the number of models in **abstract domain**.

Example

$$Diff(\iota) = x < 3 \vee y > 10$$

SMT: $(-3, 7) \models Diff(\iota) \wedge \neg[(x > 5) \wedge (y > 5)]$



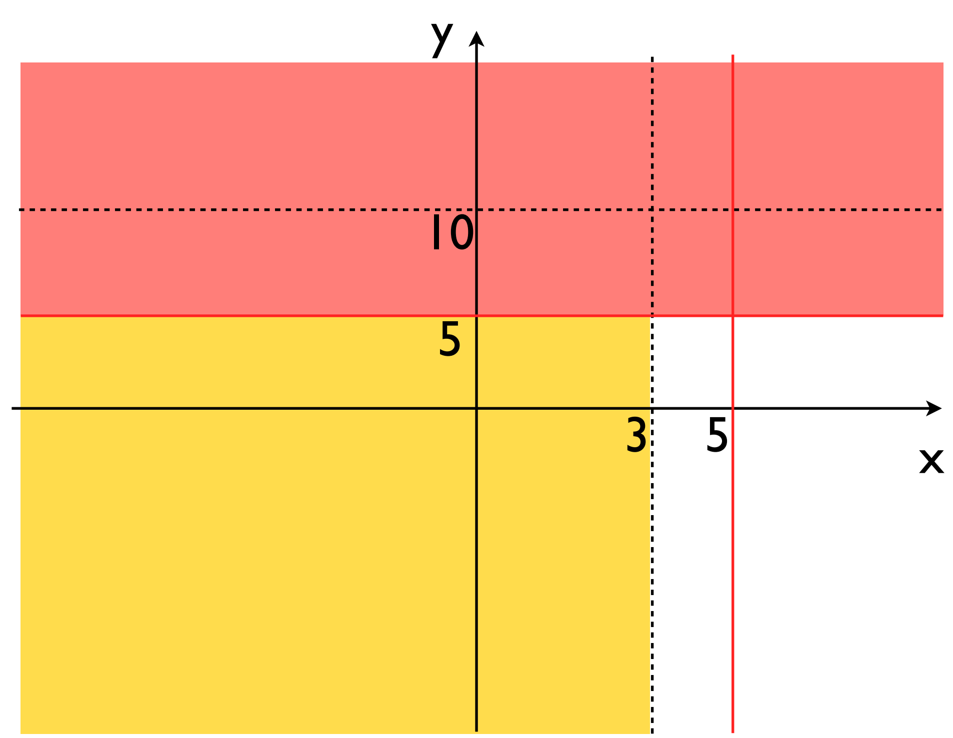
Feasible Fitness Function

Count the number of models in **abstract domain**.

Example

$$Diff(\iota) = x < 3 \vee y > 10$$

SMT: $(-3, 7) \models Diff(\iota) \wedge \neg[(x > 5) \wedge (y > 5)]$



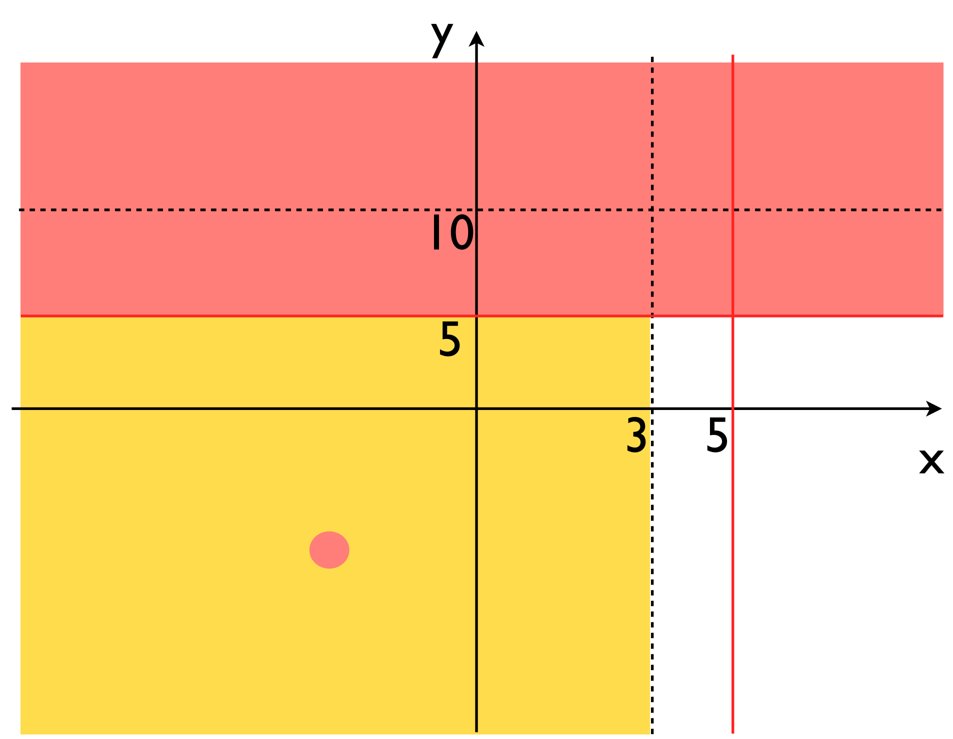
Feasible Fitness Function

Count the number of models in **abstract domain**.

Example

$$Diff(\iota) = x < 3 \vee y > 10$$

SMT: $(-5, -5) \models Diff(\iota) \wedge \neg[(x > 5) \wedge (y > 5)] \wedge \neg[(x <= 5) \wedge (y > 5)]$



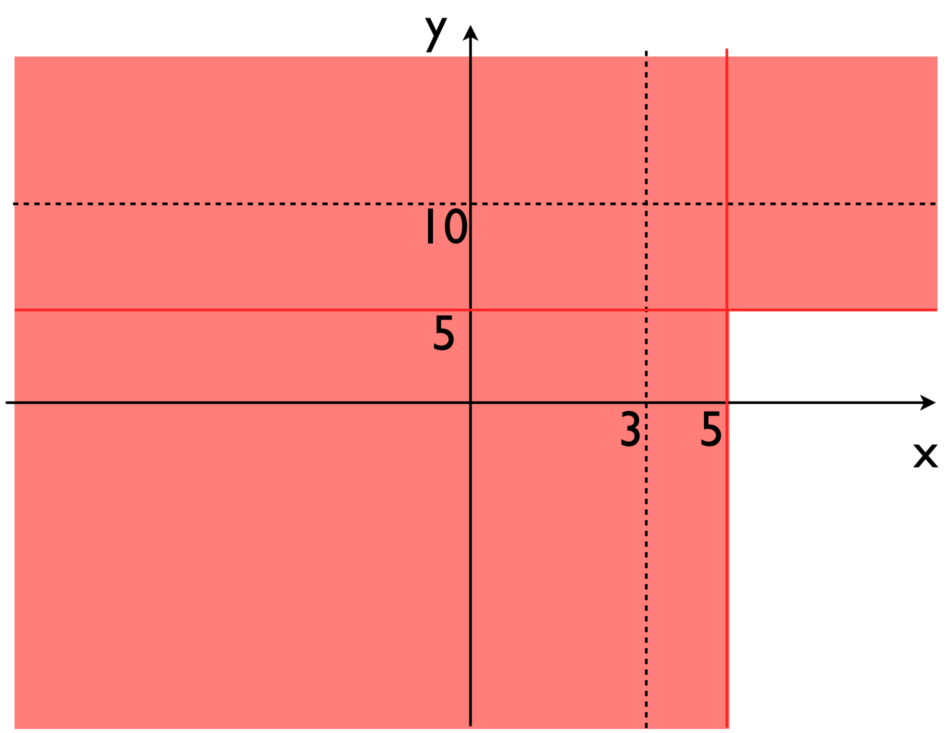
Feasible Fitness Function

Count the number of models in **abstract domain**.

Example

$$Diff(\iota) = x < 3 \vee y > 10$$

SMT: $(-5, -5) \models Diff(\iota) \wedge \neg[(x > 5) \wedge (y > 5)] \wedge \neg[(x <= 5) \wedge (y > 5)]$



Feasible Fitness Function

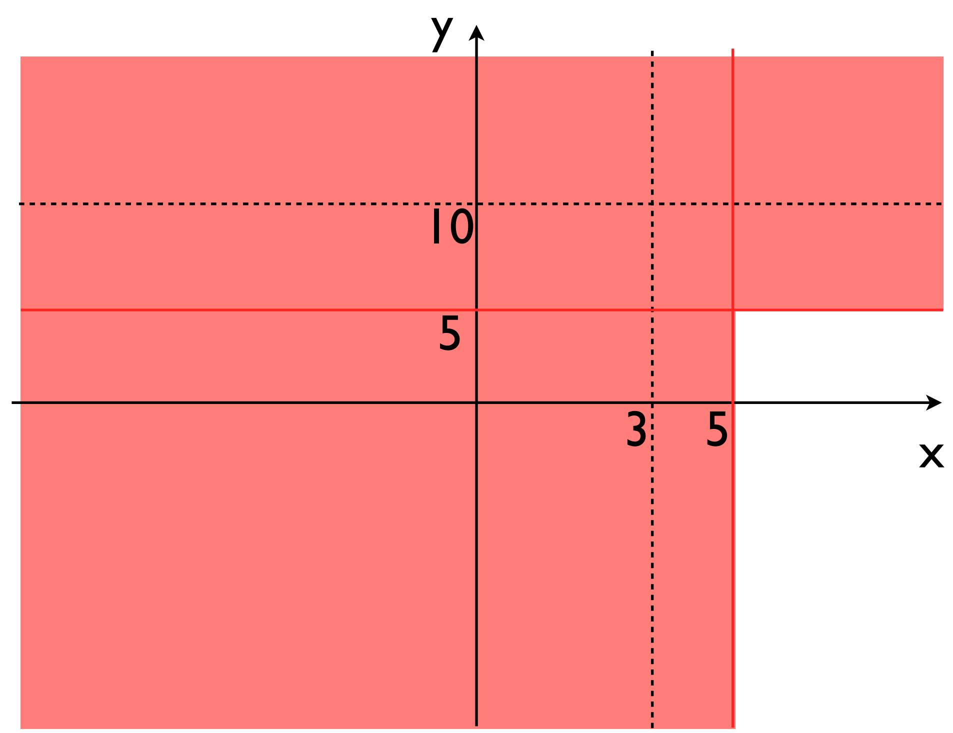
Count the number of models in **abstract domain**.

Example

$$Diff(\iota) = x < 3 \vee y > 10$$

$$SMT(Diff(\iota) \wedge \neg[(x > 5) \wedge (y > 5)] \wedge \neg[(x <= 5) \wedge (y > 5)] \wedge \neg[(x <= 5) \wedge (y <= 5)])$$

= UNSAT



Feasible Fitness Function

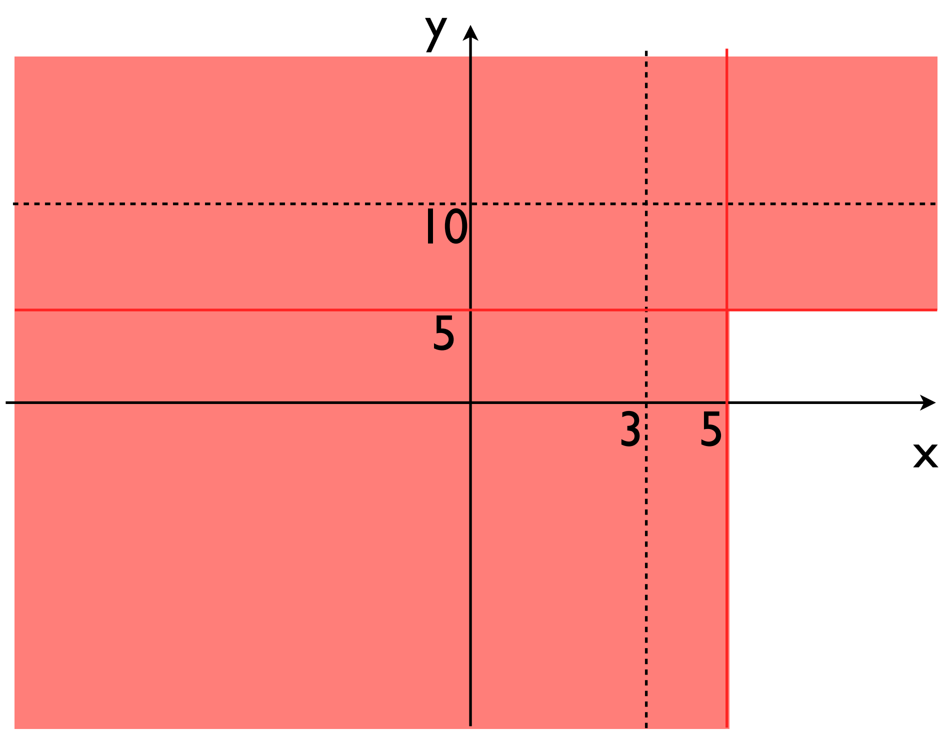
Count the number of models in **abstract domain**.

Example

$$Diff(\iota) = x < 3 \vee y > 10$$

$$SMT(Diff(\iota) \wedge \neg[(x > 5) \wedge (y > 5)] \wedge \neg[(x <= 5) \wedge (y > 5)] \wedge \neg[(x <= 5) \wedge (y <= 5)])$$

= UNSAT



$$\widehat{Fit}(\iota) = 3$$

Feasible Fitness Function

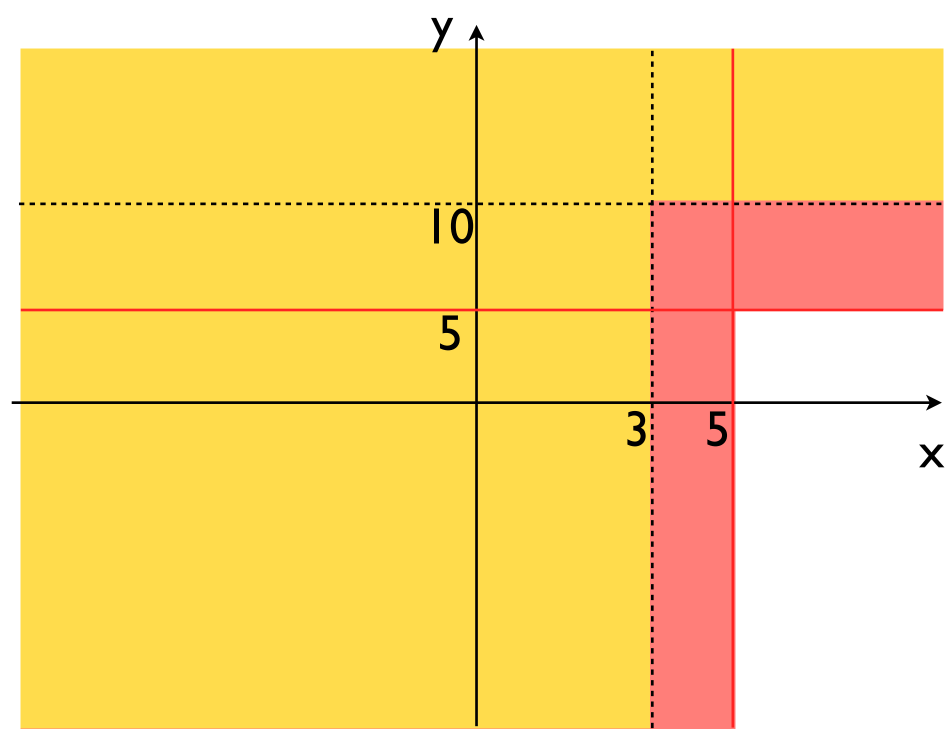
Count the number of models in **abstract domain**.

Example

$$Diff(\iota) = x < 3 \vee y > 10$$

$$SMT(Diff(\iota) \wedge \neg[(x > 5) \wedge (y > 5)] \wedge \neg[(x <= 5) \wedge (y > 5)] \wedge \neg[(x <= 5) \wedge (y <= 5)])$$

= UNSAT



Fitness Function Property ✓

$$Diff(\iota_1) \subseteq Diff(\iota_2) \Rightarrow Fit(\iota_1) \leq Fit(\iota_2)$$

Complexity of Fitness Function

Number of SMT calls could raise up to 2^n if we have n atomic propositions in predicate abstraction.

Complexity of Fitness Function

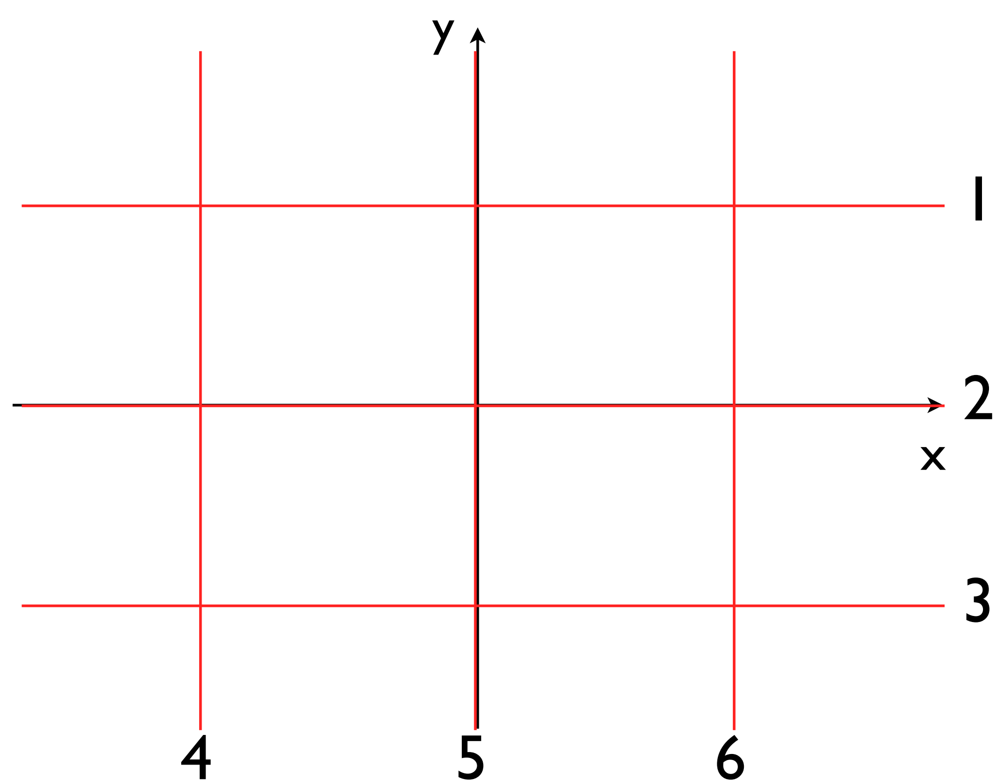
Number of SMT calls could raise up to 2^n if we have n atomic propositions in predicate abstraction.

In practice, that is **not** the case because of dependencies among atomic propositions.

Complexity of Fitness Function

Number of SMT calls could raise up to 2^n if we have n atomic propositions in predicate abstraction.

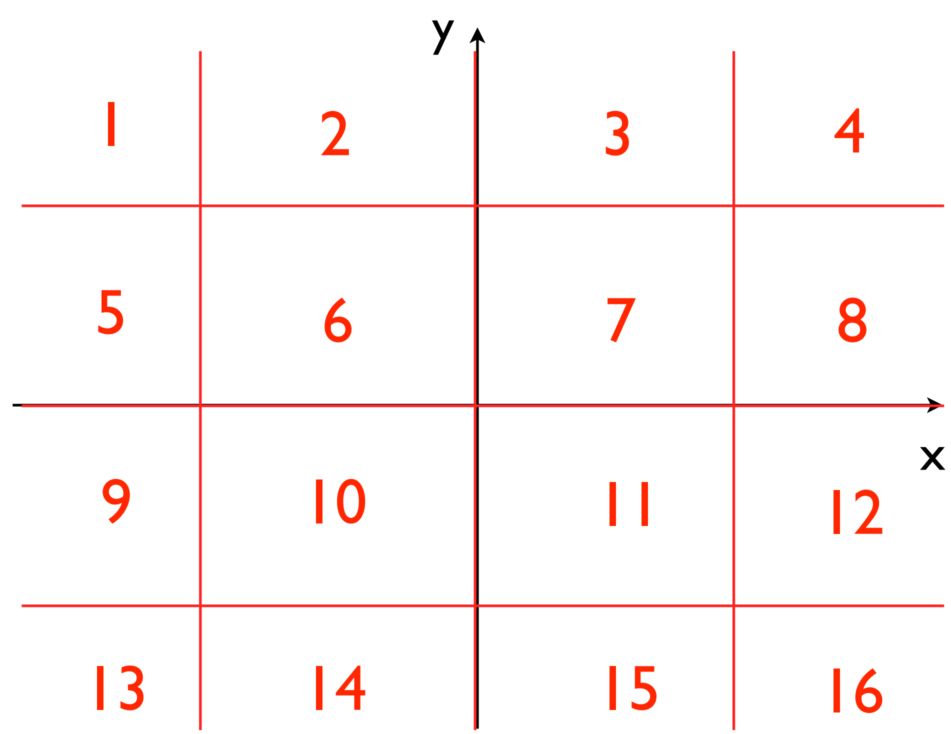
In practice, that is **not** the case because of dependencies among atomic propositions.



Complexity of Fitness Function

Number of SMT calls could raise up to 2^n if we have n atomic propositions in predicate abstraction.

In practice, that is **not** the case because of dependencies among atomic propositions.

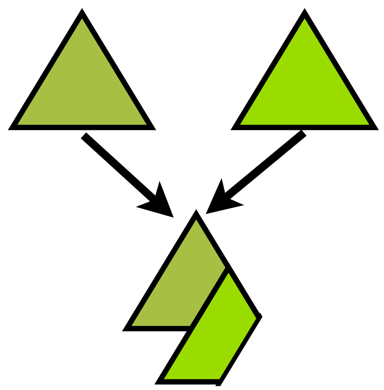


$$16 \leq 2^6 = 64$$

Issue: Formula Size

Formula size can grow infinitely.

1. Crossover operation



2. Fitness Function

SMT: $(10, 15) \models \mathit{Diff}(\iota)$

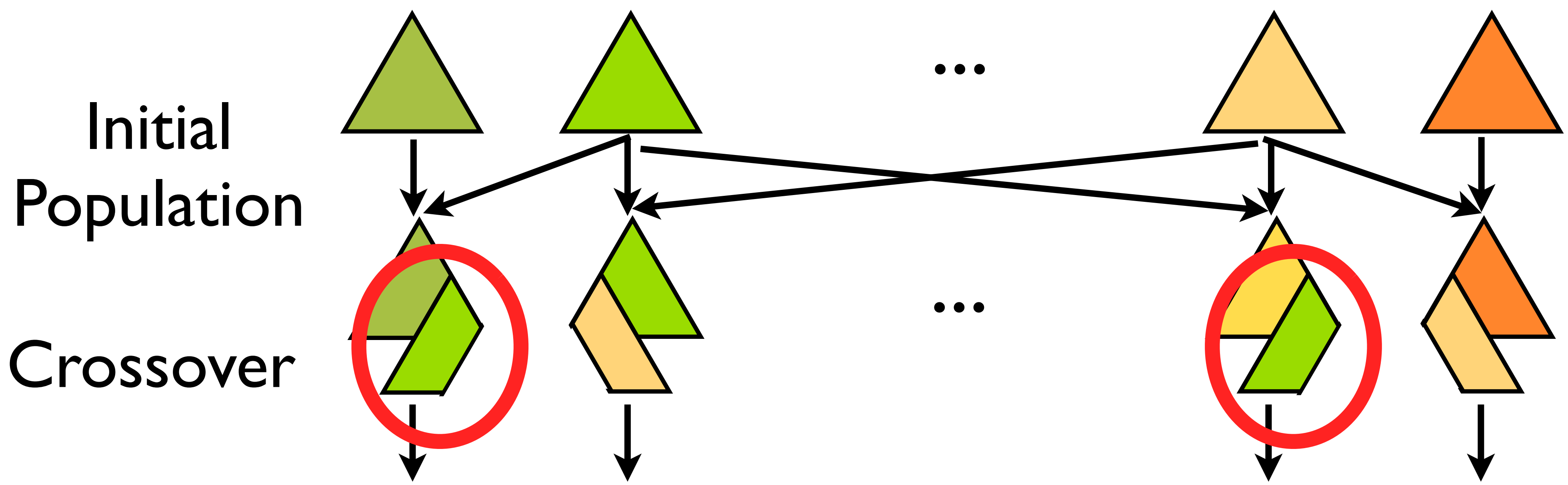
$(-3, 7) \models \mathit{Diff}(\iota) \wedge \neg[(x > 5) \wedge (y > 5)]$

$(-5, -5) \models \mathit{Diff}(\iota) \wedge \neg[(x > 5) \wedge (y > 5)] \wedge \neg[(x <= 5) \wedge (y > 5)]$

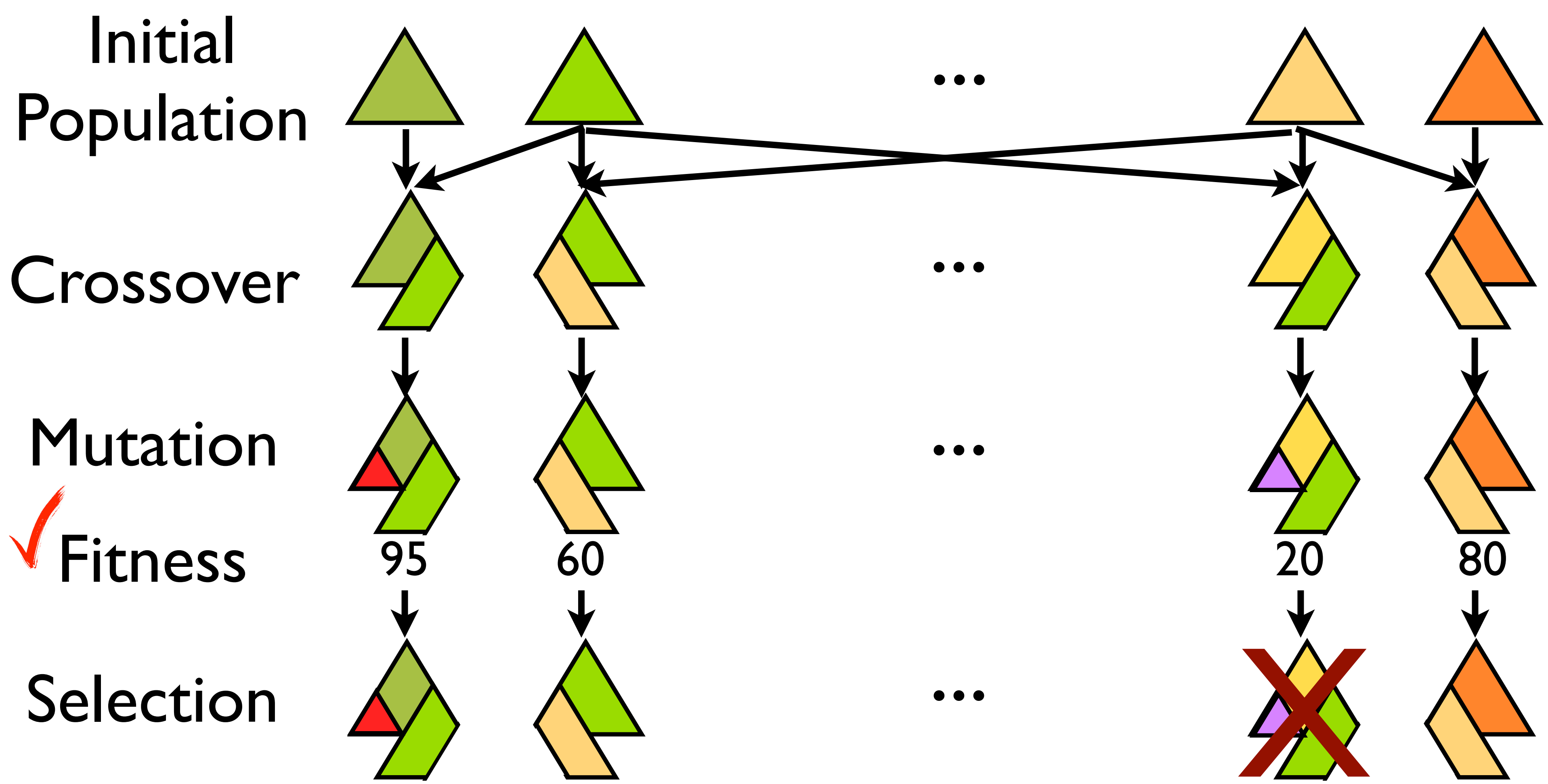
Solution: Simplify formulae.

Progress

1. Design and implement abstract data type for formula to share subformulae
(Heejae Shin & Wonchan Lee)



Plan


Initial
Population
Crossover
Mutation
Fitness
Selection
...
...
...
...
95
60
20
80

Thank you