Optimizing Synthesis with Metasketches

Artifact • Consistent • Complete • Well Documented • Easy to Reuse • Evaluated • POPL • AEC

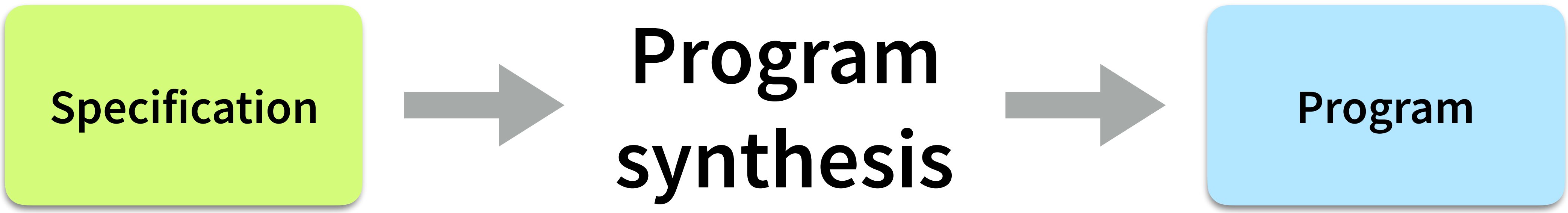
James Bornholt
Emina Torlak
Dan Grossman
Luis Ceze
University of Washington

SYNAPSE

Program synthesis

Specification

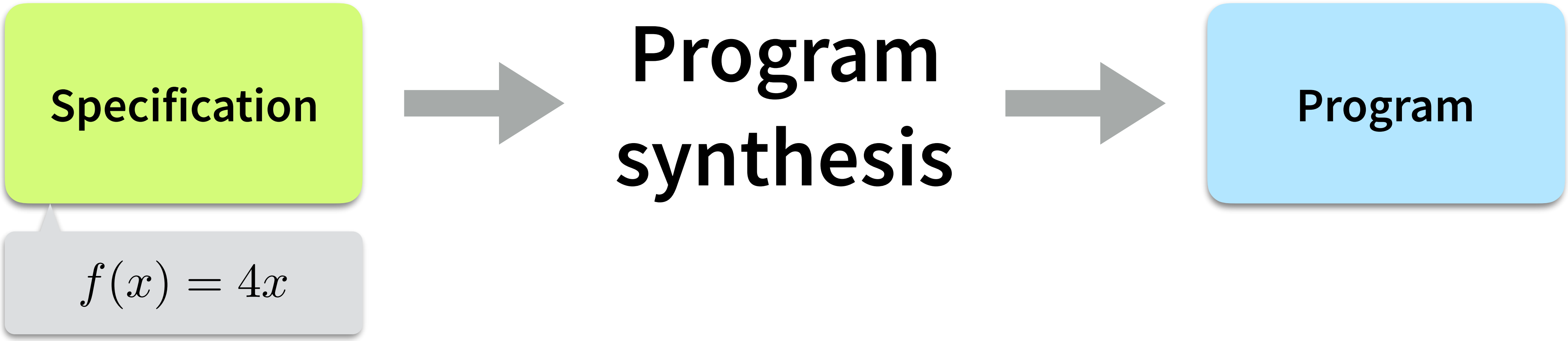
Program synthesis



Specification

Program synthesis

Program

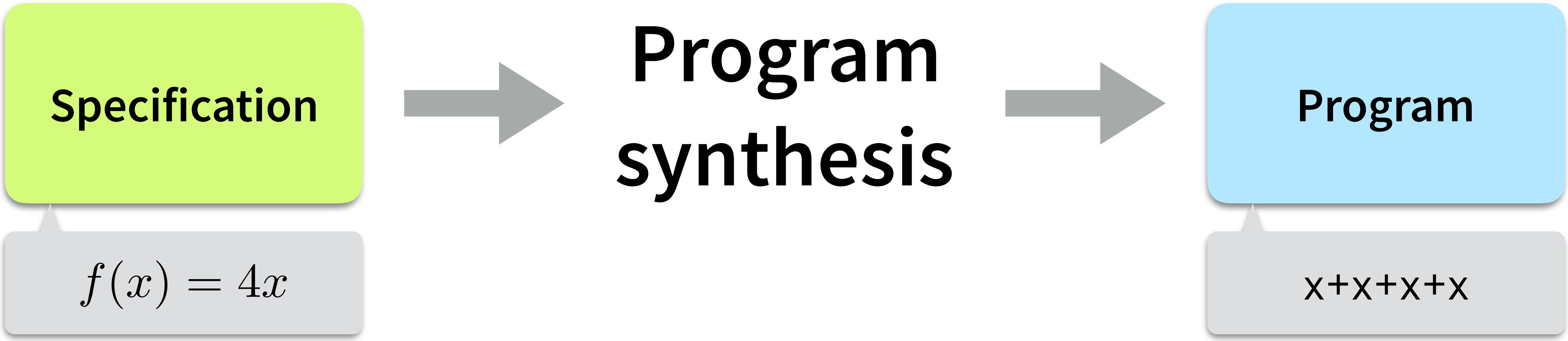


Specification

$f(x) = 4x$

Program synthesis

Program



Specification

$f(x) = 4x$

Program synthesis

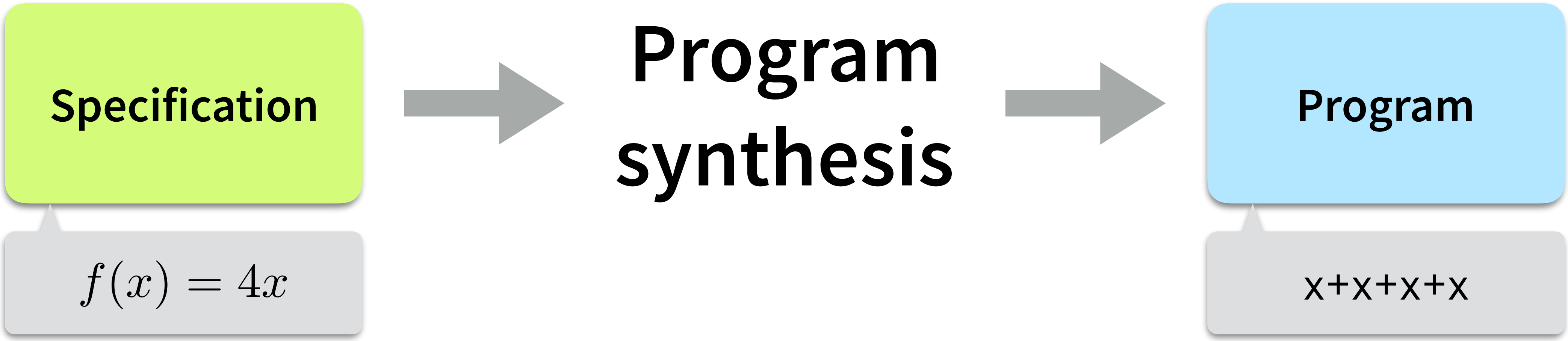
Program

x+x+x+x

Compilation
[PLDI’14]

Data Structures
[PLDI’15]

End-user Programming
[POPL’11]



Specification

$f(x) = 4x$

Program synthesis

Program

x+x+x+x

Executable Biology
[POPL’13]

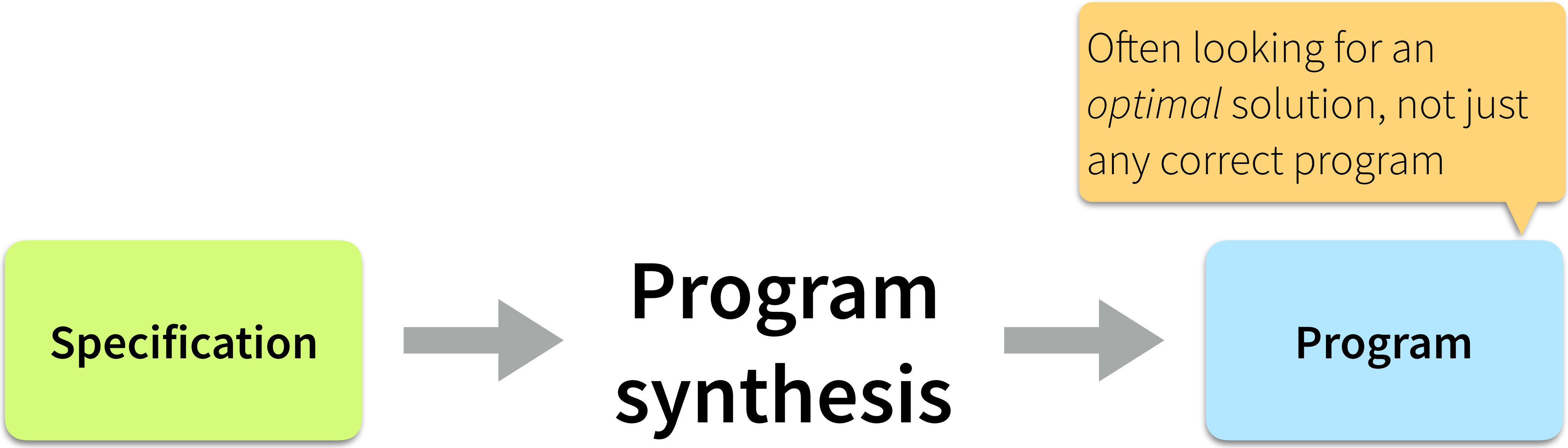
Browser Layout
[PPoPP’13]

Cache Protocols
[PLDI’13]

Compilation
[PLDI’14]

Data Structures
[PLDI’15]

End-user Programming
[POPL’11]



Often looking for an *optimal* solution, not just any correct program

Specification

Program synthesis

Program

Executable Biology
[POPL’13]

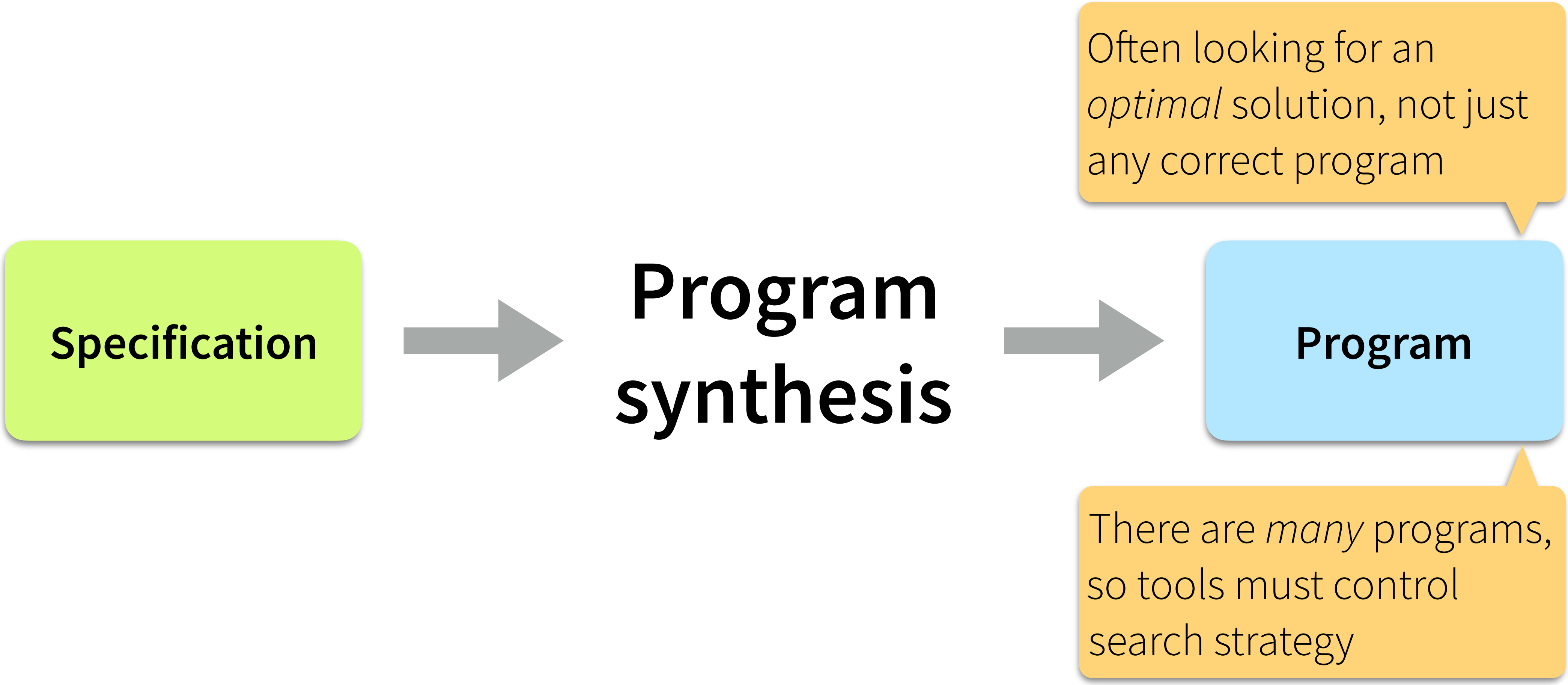
Browser Layout
[PPoPP’13]

Cache Protocols
[PLDI’13]

Compilation [PLDI'14]

Data Structures [PLDI'15]

End-user Programming [POPL'11]



Often looking for an *optimal* solution, not just any correct program

Specification

Program synthesis

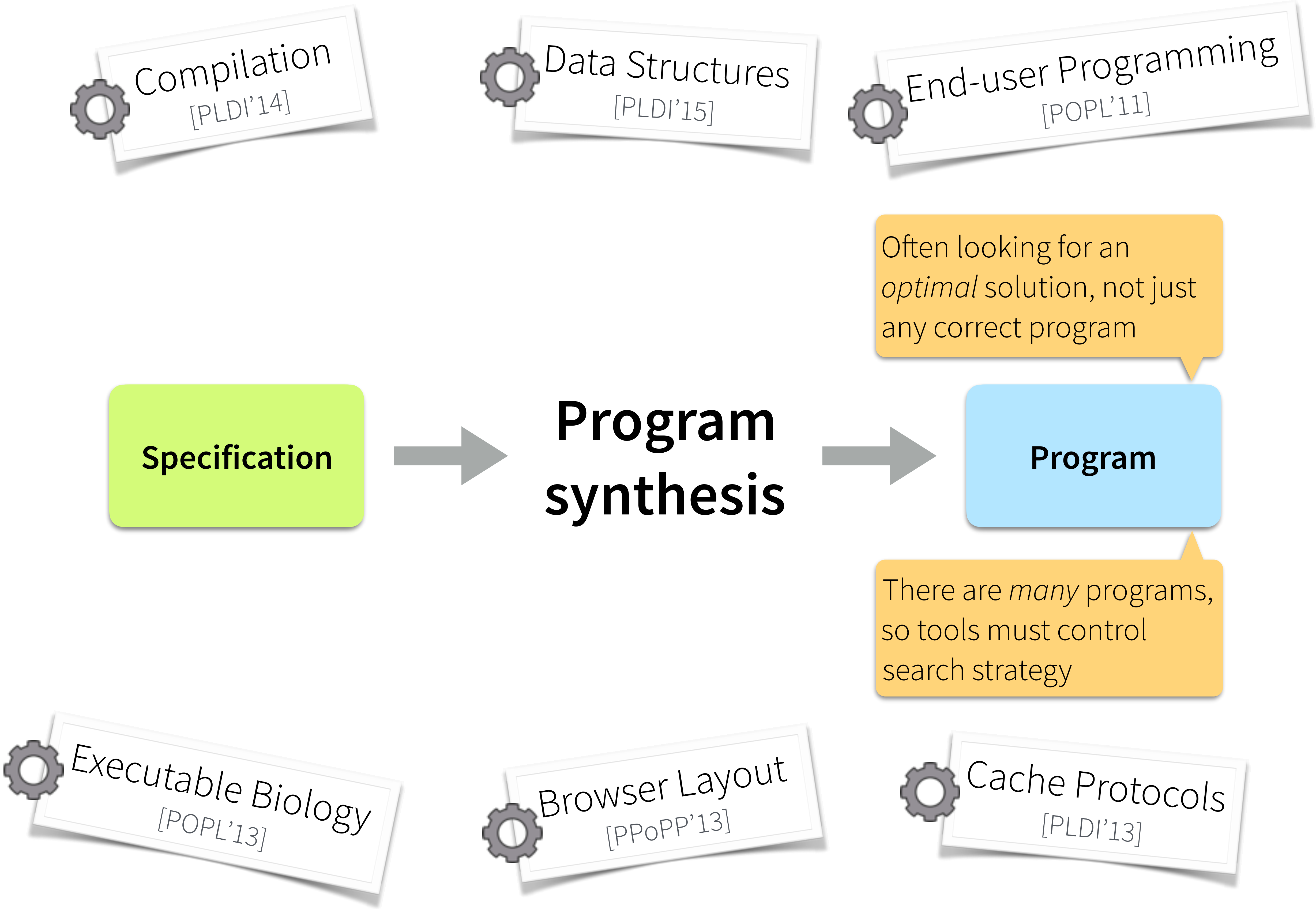
Program

There are *many* programs, so tools must control search strategy

Executable Biology [POPL'13]

Browser Layout [PPoPP'13]

Cache Protocols [PLDI'13]



Compilation [PLDI'14]

Data Structures [PLDI'15]

End-user Programming [POPL'11]

Often looking for an *optimal* solution, not just any correct program

Specification

Program synthesis

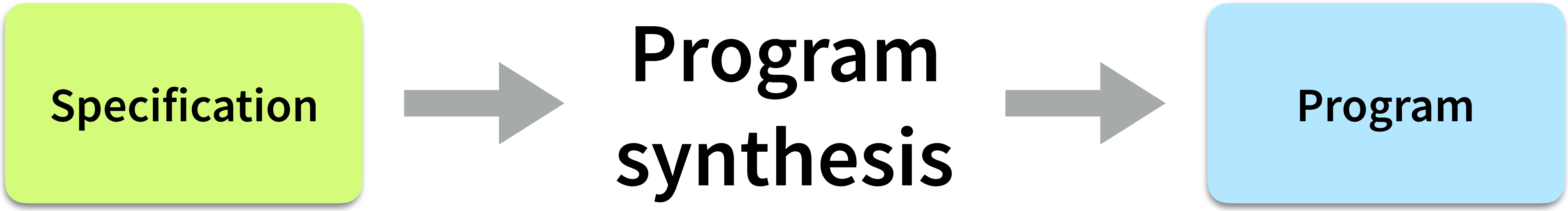
Program

There are *many* programs, so tools must control search strategy

Executable Biology [POPL'13]

Browser Layout [PPoPP'13]

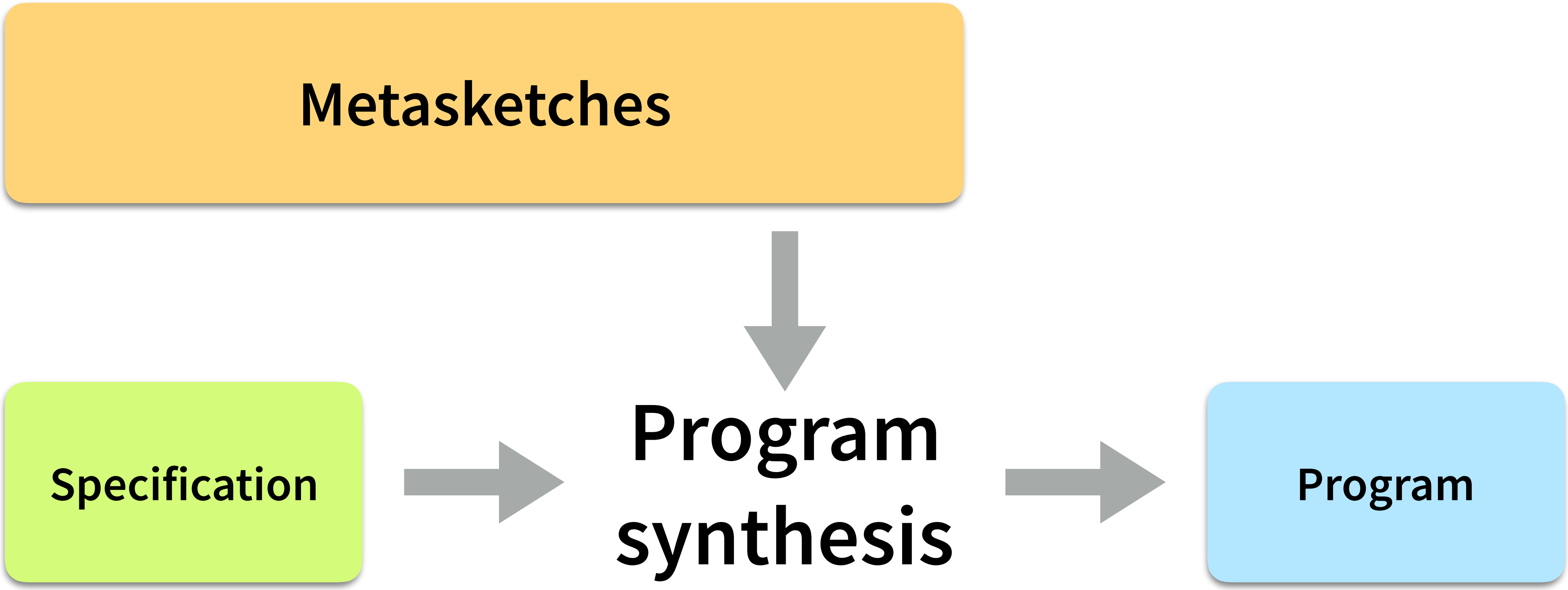
Cache Protocols [PLDI'13]



Specification

Program synthesis

Program

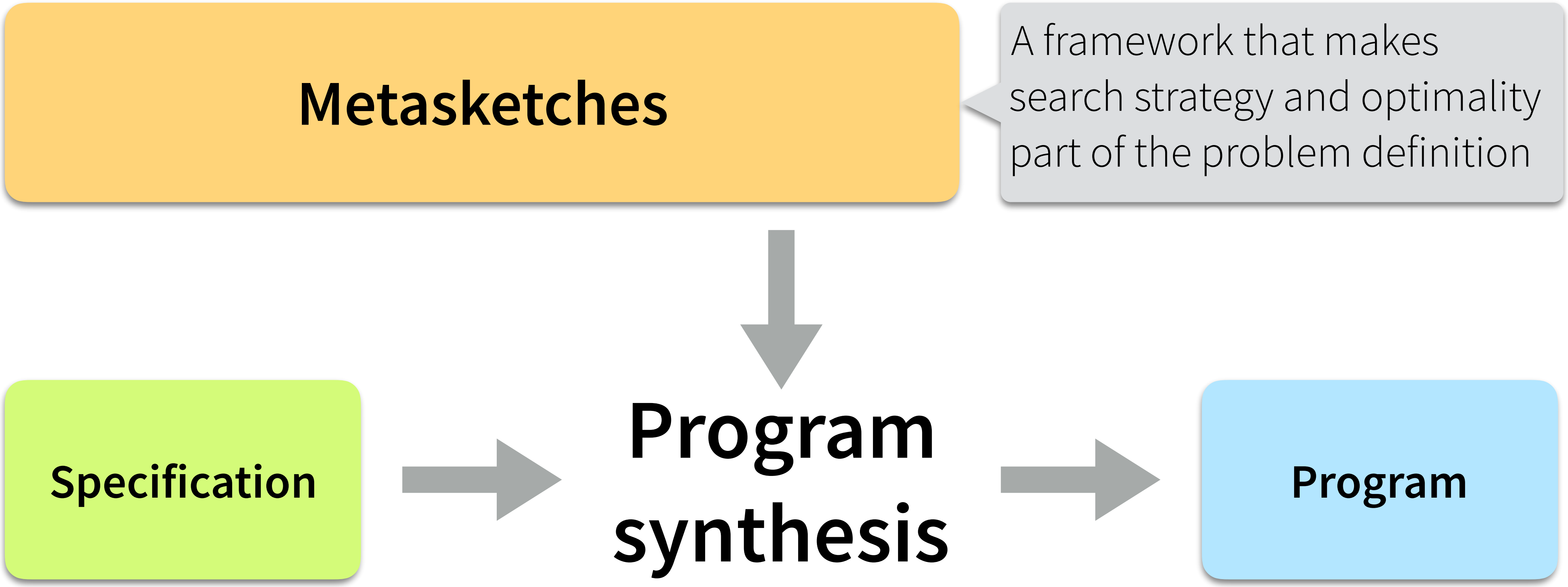


Metasketches

Specification

Program synthesis

Program



Metasketches

A framework that makes search strategy and optimality part of the problem definition

Specification

Program synthesis

Program

Metasketches

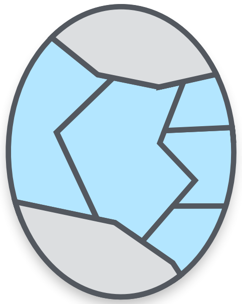
Design and structure

Metasketches

Design and structure

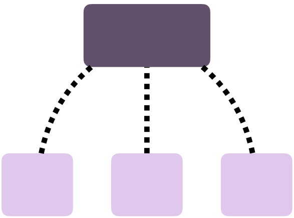
Synapse

A metasketch solver



Metasketches

Design and structure

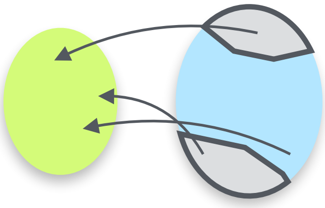


Synapse

A metasketch solver

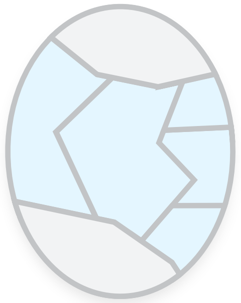
Results

Better solutions, faster



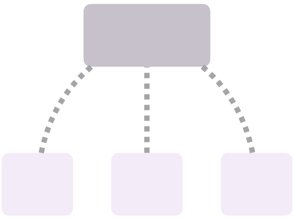
Background

Syntax-guided synthesis



Metasketches

Design and structure



Synapse

A metasketch solver

Results

Better solutions, faster

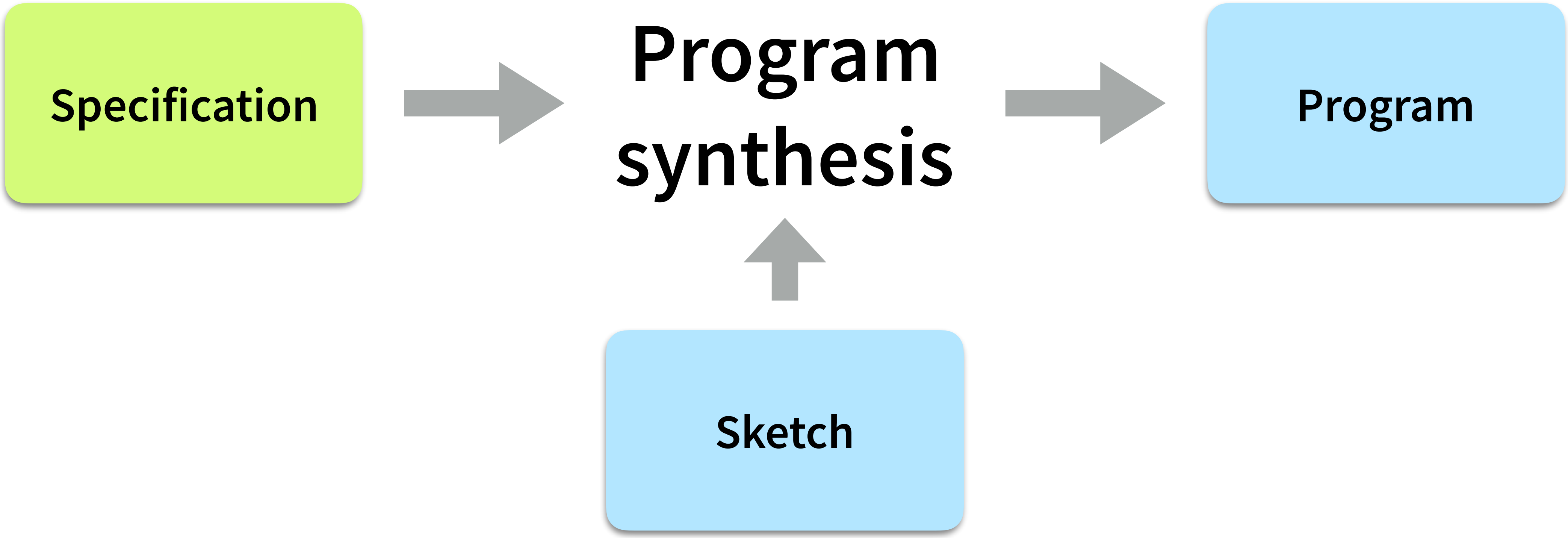
Syntax-guided synthesis

Specification

Program synthesis

Program

Syntax-guided synthesis



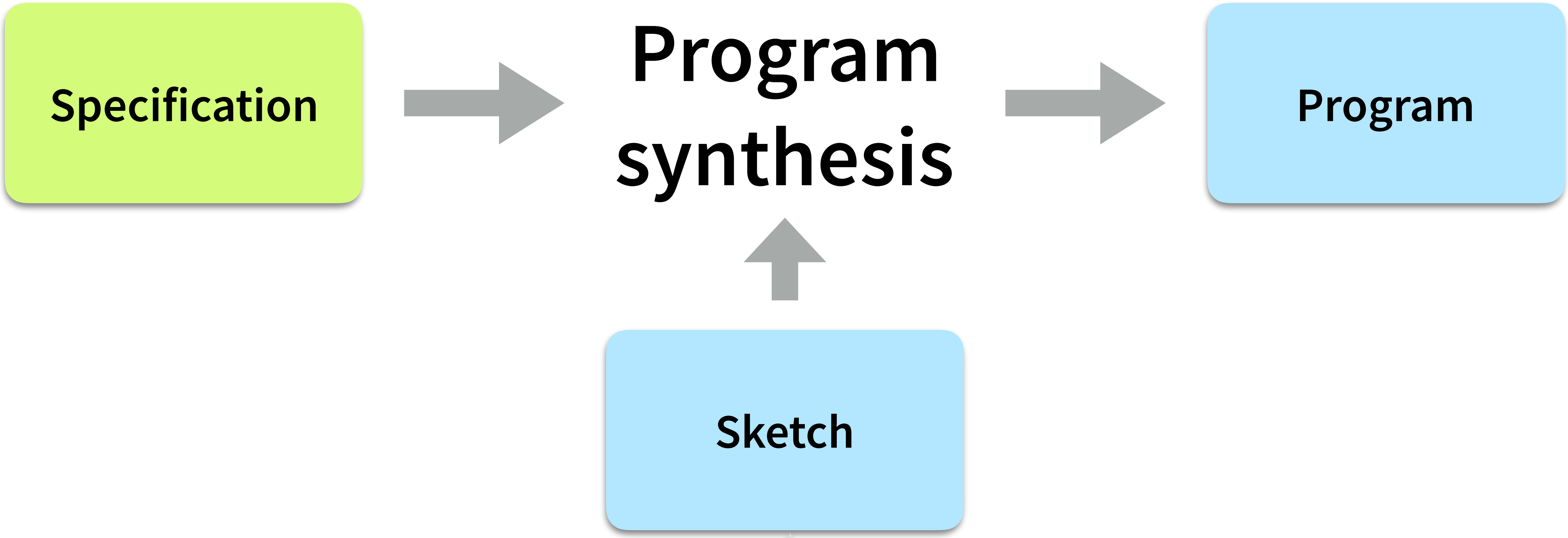
Specification

Program synthesis

Program

Sketch

Syntax-guided synthesis



Specification

Program synthesis

Program

Sketch

```
def f(x):
  return Expr

Expr := x | ?? | Expr op Expr
  op := + | * | - | >> | <<
  ?? := integer constant
```

Syntax-guided synthesis: guess, check, learn

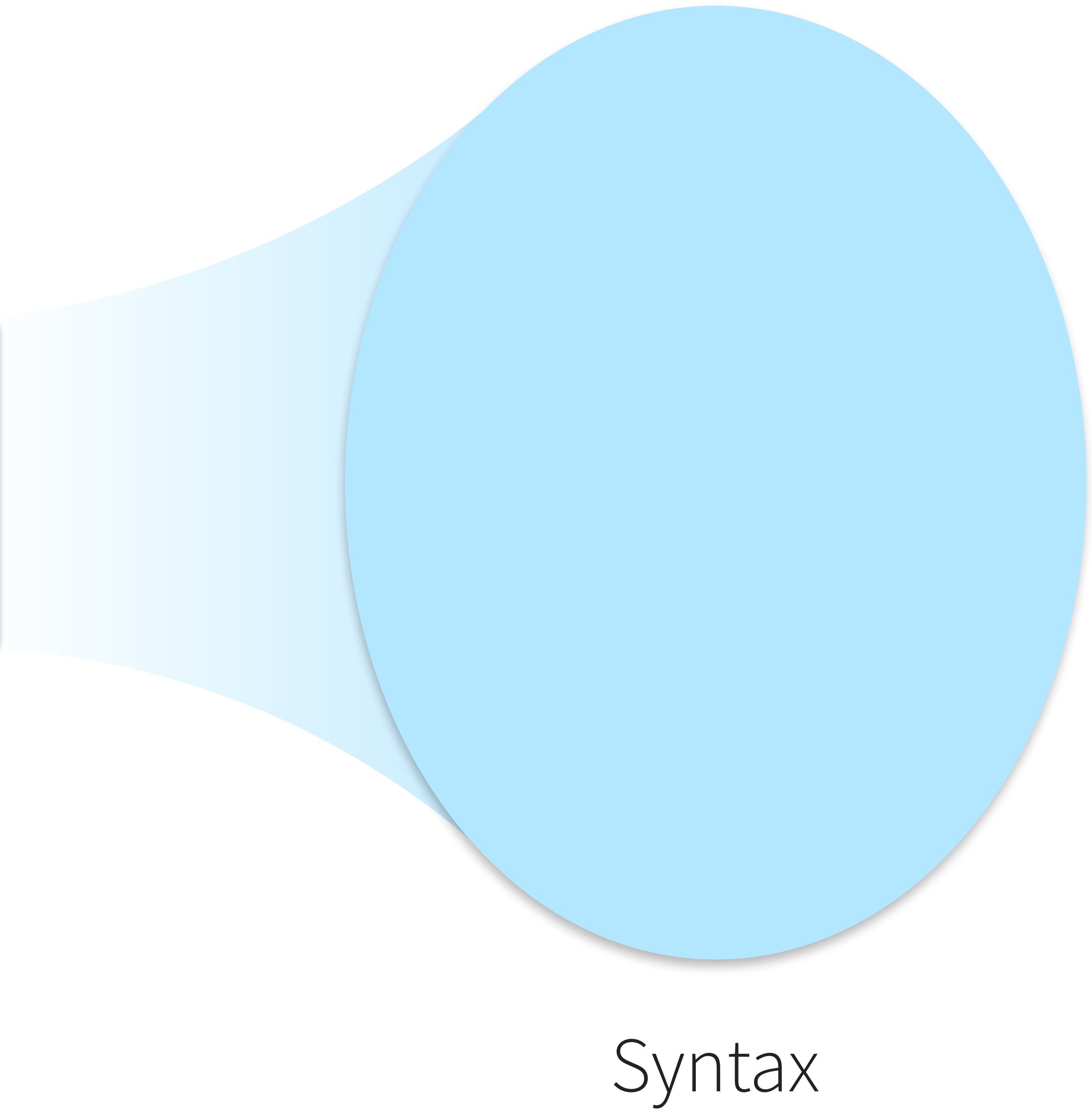
```
def f(x):
  return Expr

Expr := x | ?? | Expr op Expr
  op := + | * | - | >> | <<
  ?? := integer constant
```

Syntax-guided synthesis: guess, check, learn

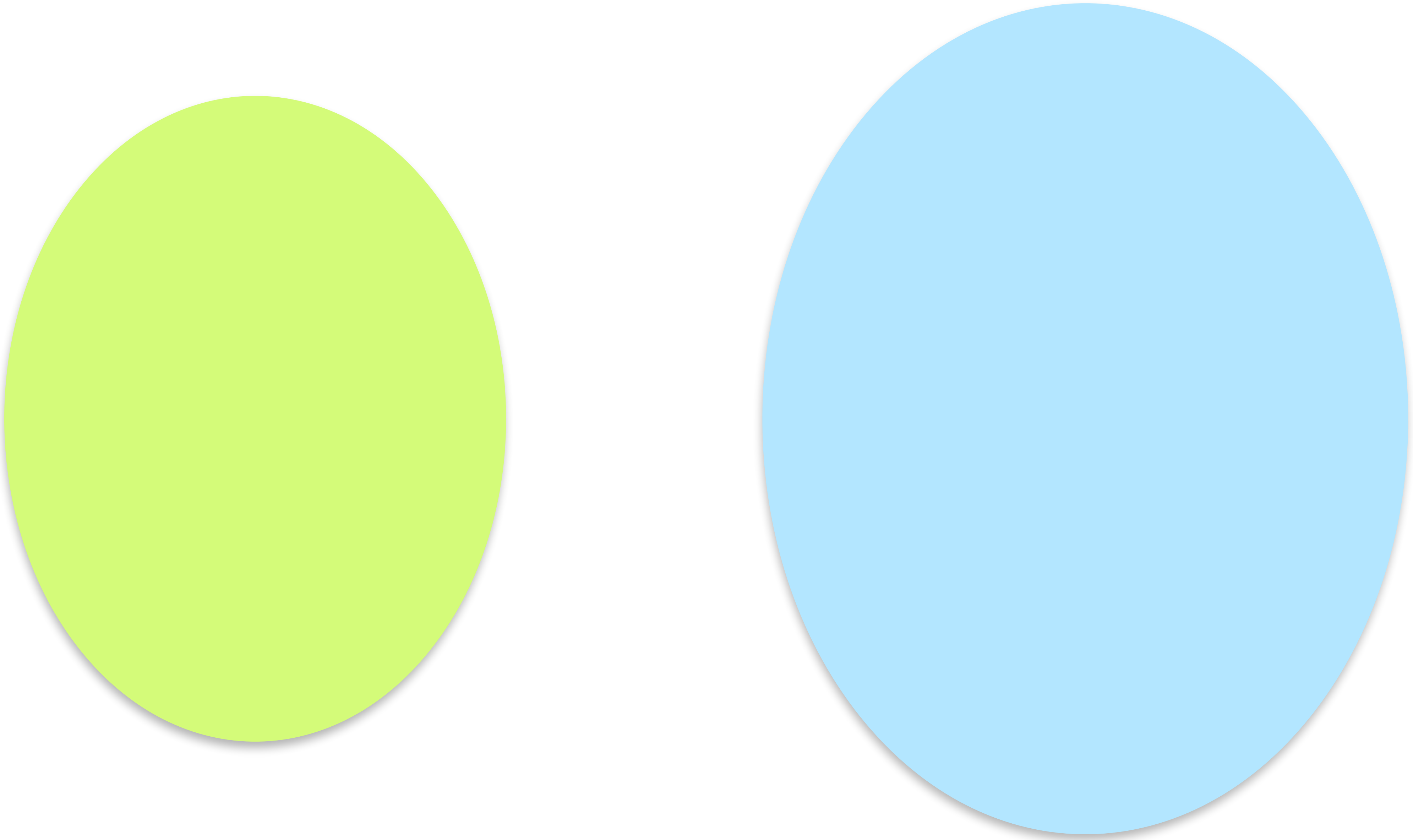
```
def f(x):
  return Expr

Expr := x | ?? | Expr op Expr
  op := + | * | - | >> | <<
  ?? := integer constant
```



Syntax-guided synthesis: guess, check, learn

Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]

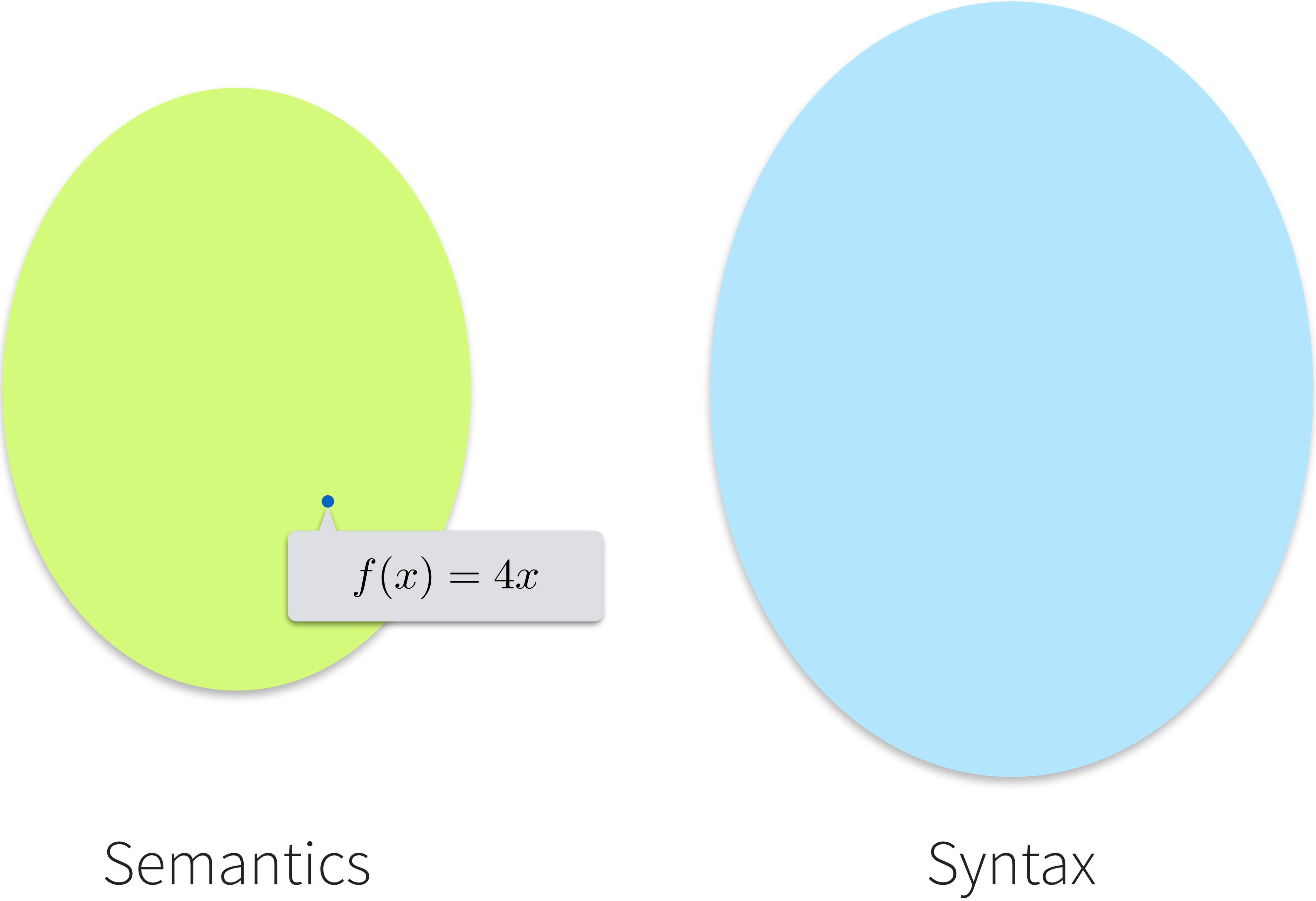


Semantics

Syntax

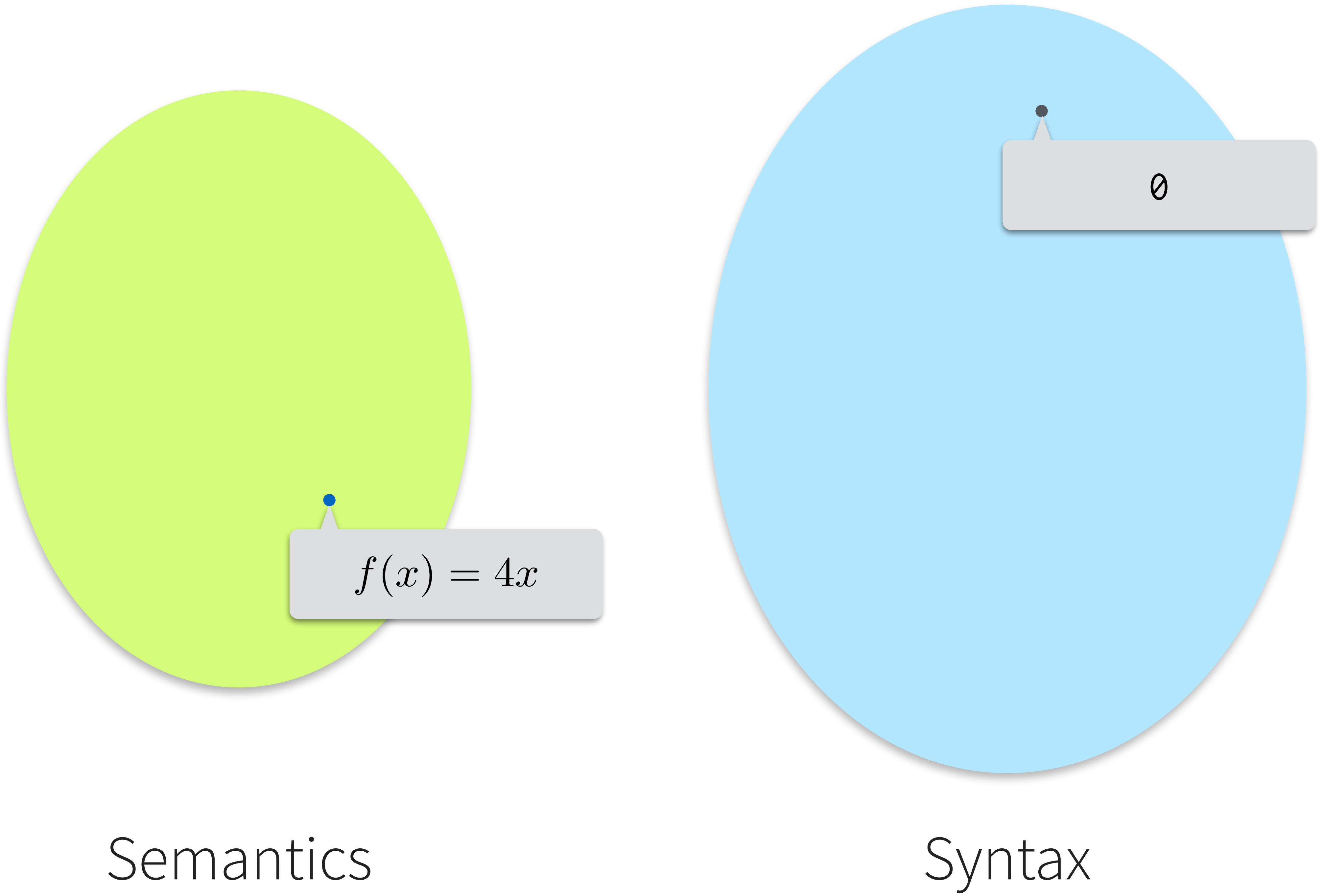
Syntax-guided synthesis: guess, check, learn

Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]



Syntax-guided synthesis: guess, check, learn

Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]



$f(x) = 4x$

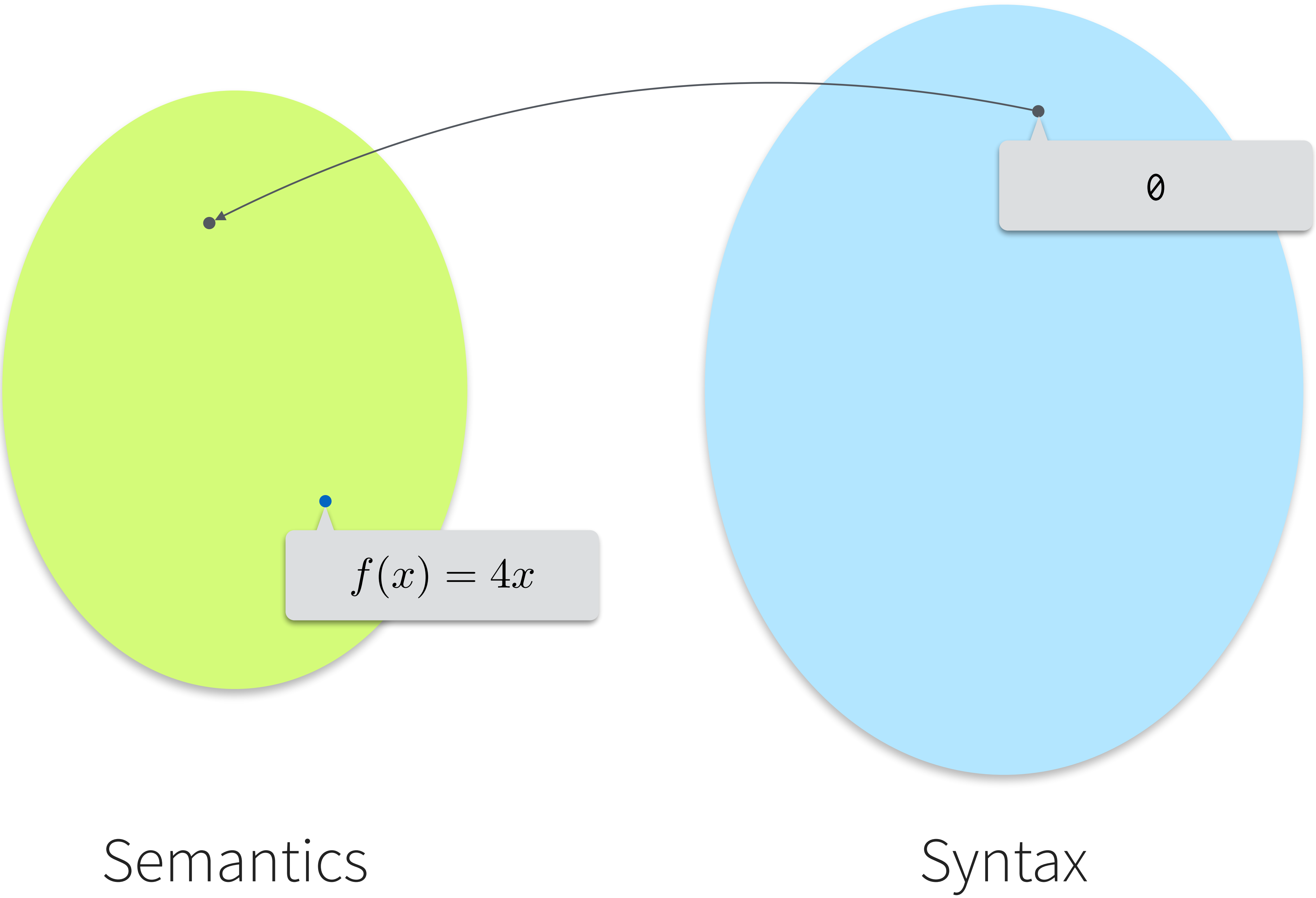
`0`

Semantics

Syntax

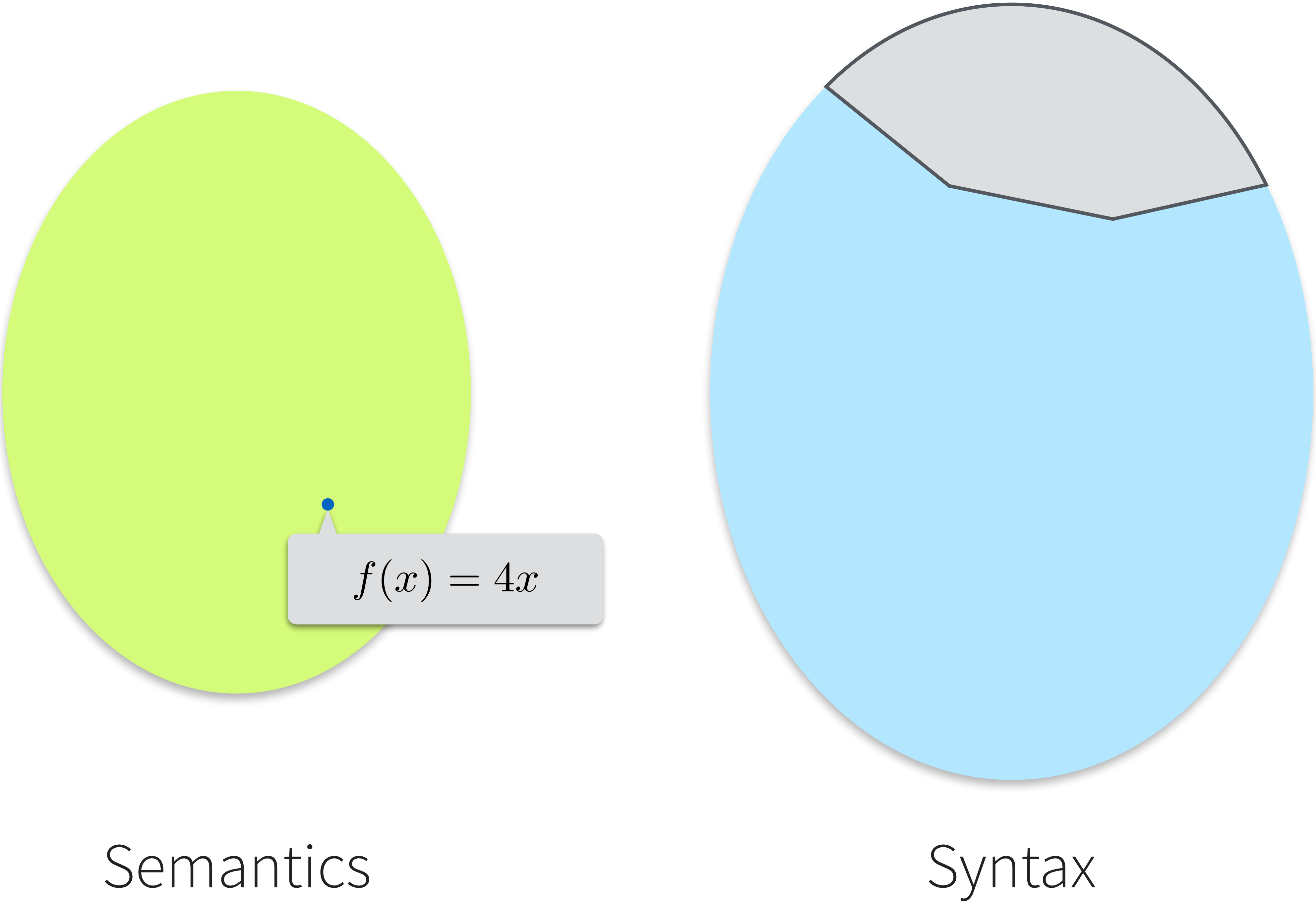
Syntax-guided synthesis: guess, check, learn

Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]



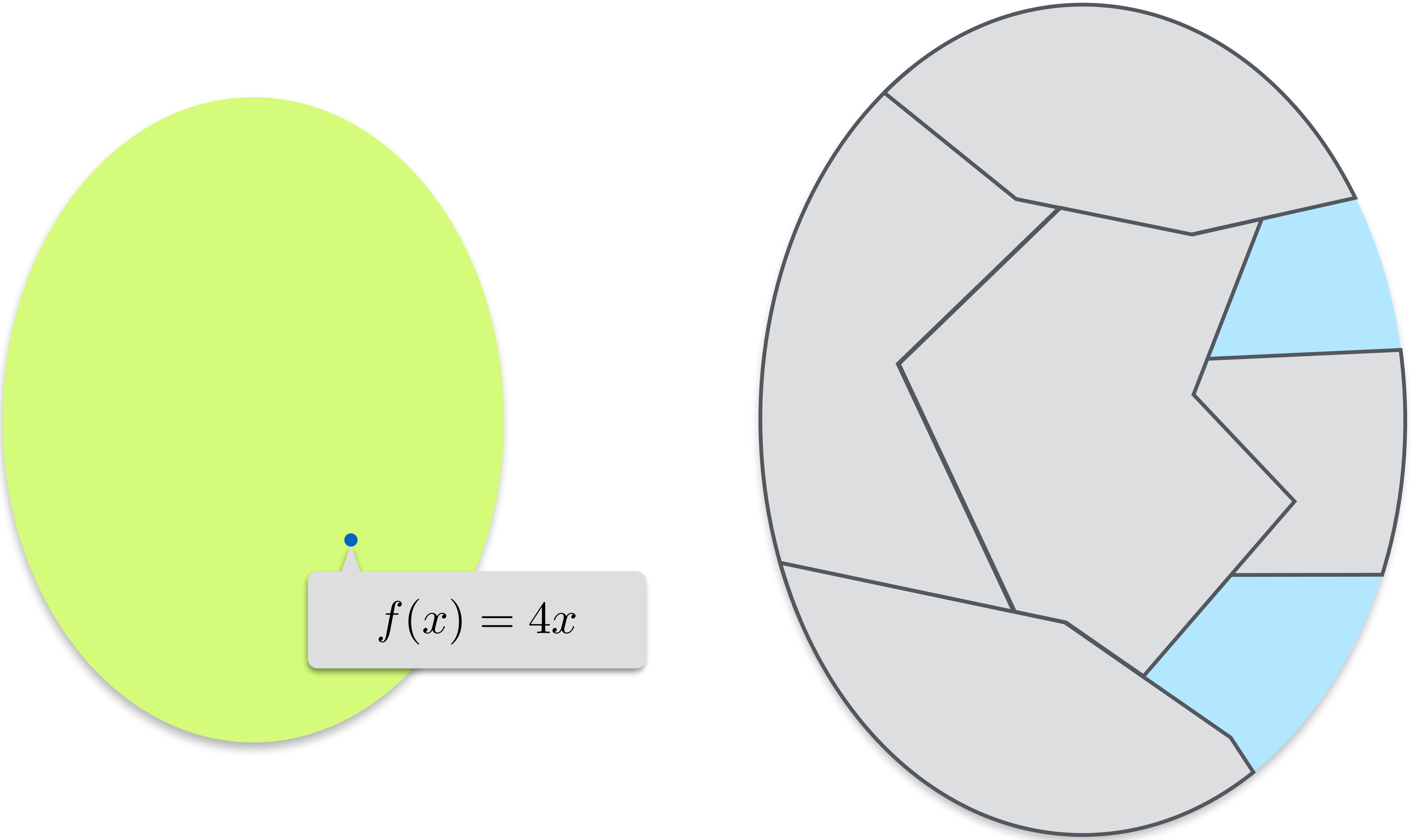
Syntax-guided synthesis: guess, check, learn

Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]



Syntax-guided synthesis: guess, check, learn

Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]

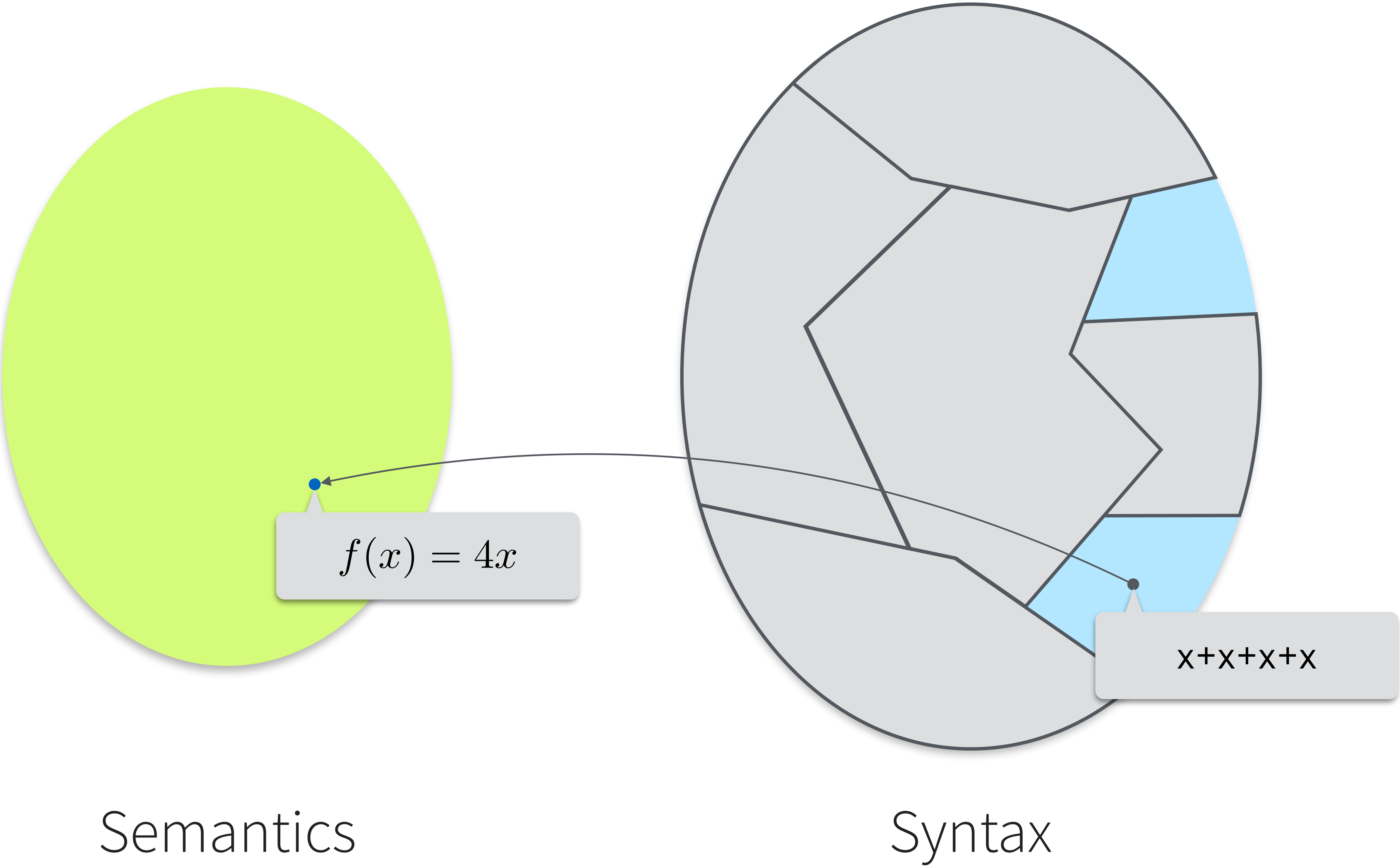


Semantics

Syntax

Syntax-guided synthesis: guess, check, learn

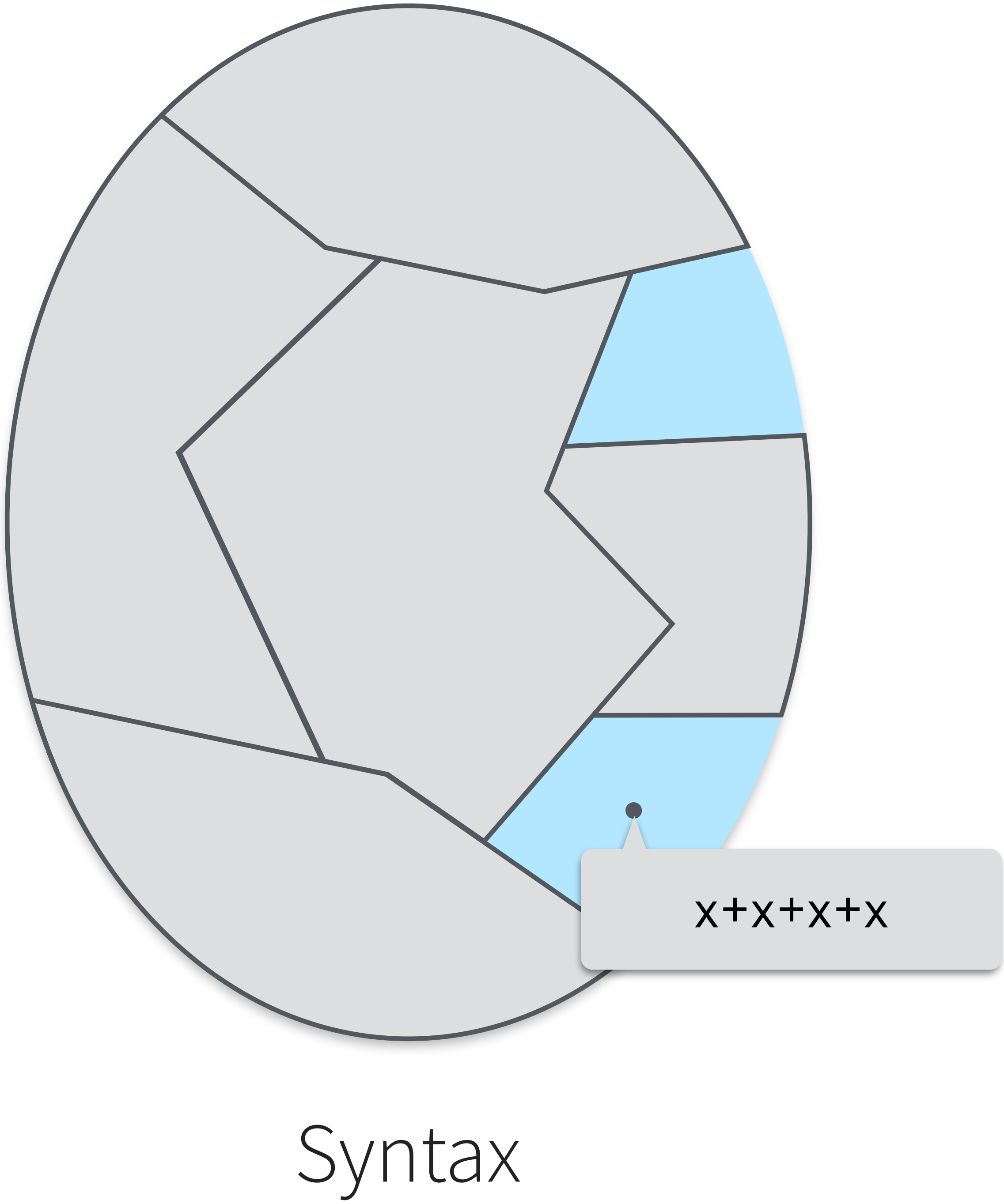
Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]



Syntax-guided synthesis: guess, check, learn

Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]

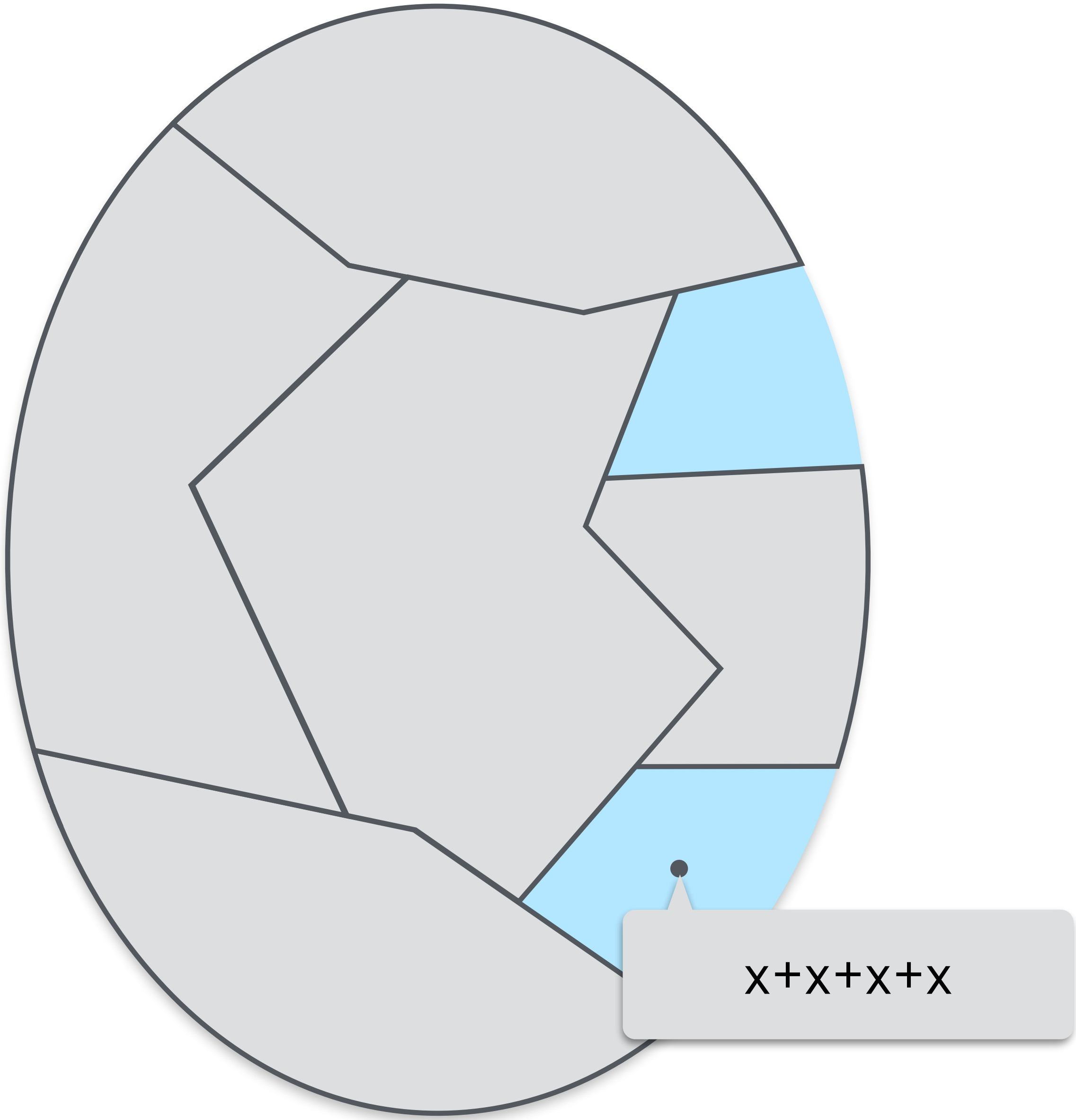
1. **Search order** is critical



Syntax-guided synthesis: guess, check, learn

Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]

1. **Search order** is critical
2. Desire **optimal** solutions

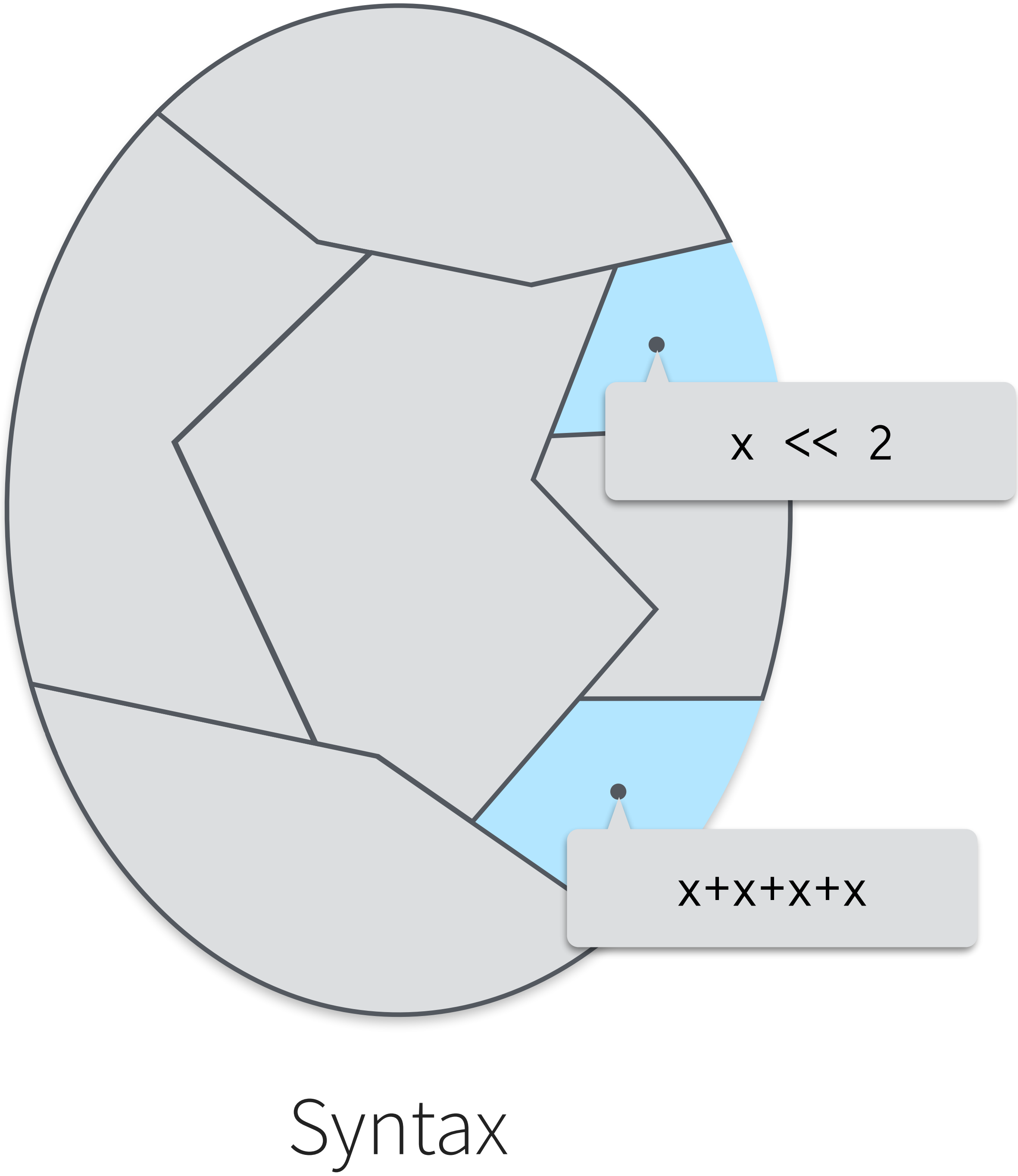


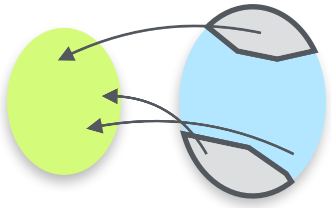
Syntax

Syntax-guided synthesis: guess, check, learn

Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]

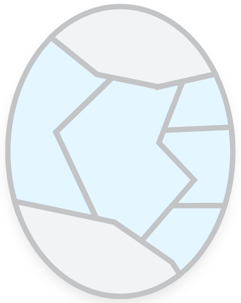
1. **Search order** is critical
2. Desire **optimal** solutions





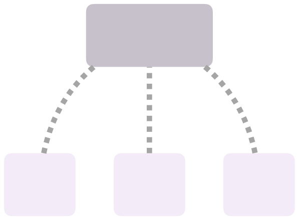
Background

Syntax-guided synthesis



Metasketches

Design and structure

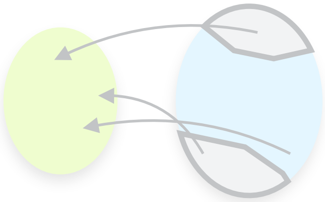


Synapse

A metasketch solver

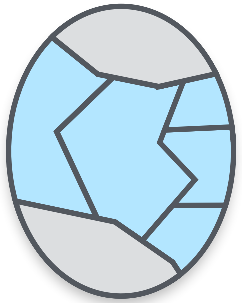
Results

Better solutions, faster



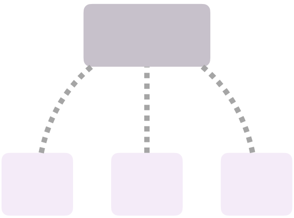
Background

Syntax-guided synthesis



Metasketches

Design and structure



Synapse

A metasketch solver

Results

Better solutions, faster

Metasketches express structure and strategy

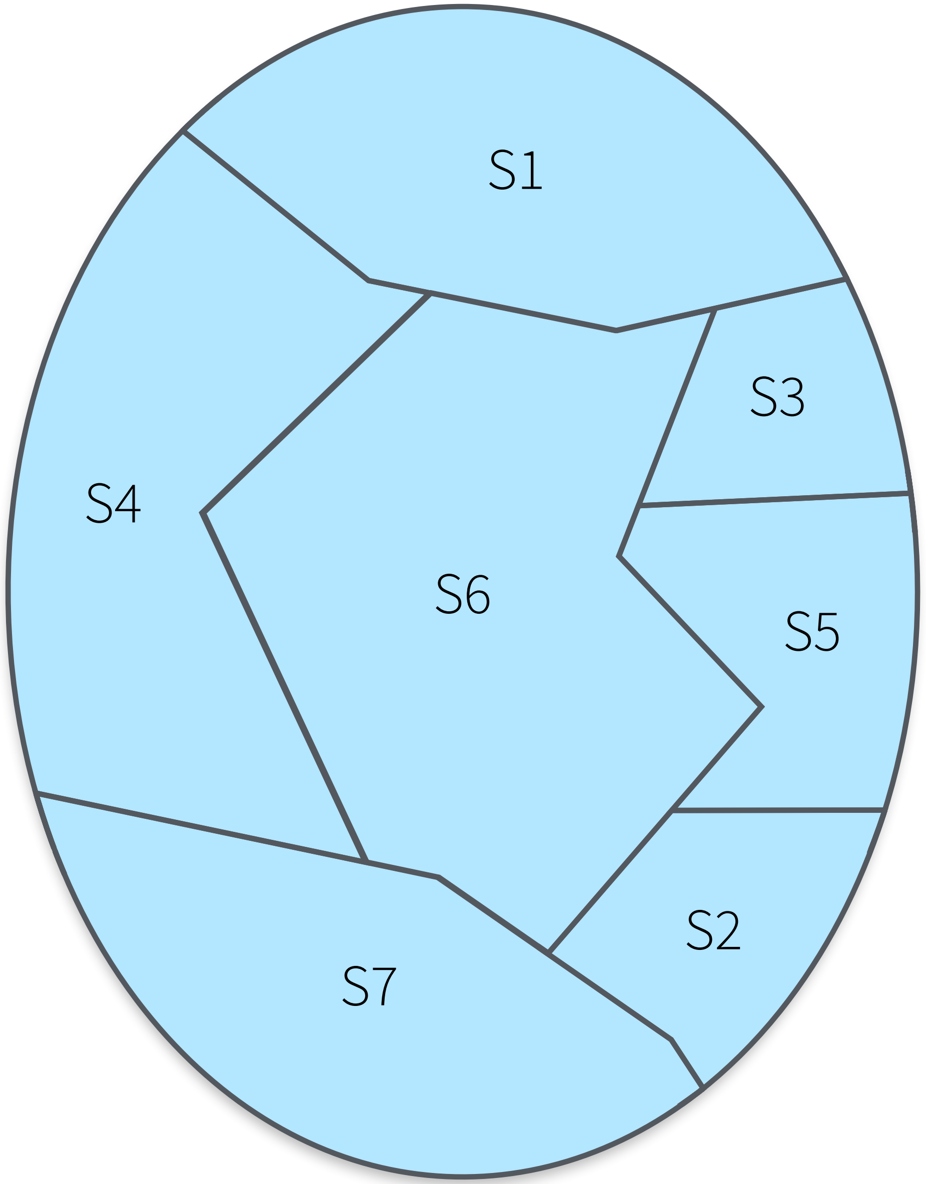
1. **Search order** is critical
2. Desire **optimal** solutions

Metasketches express structure and strategy

1. **Search order** is critical
2. Desire **optimal** solutions

A metasketch contains:

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)



Syntax

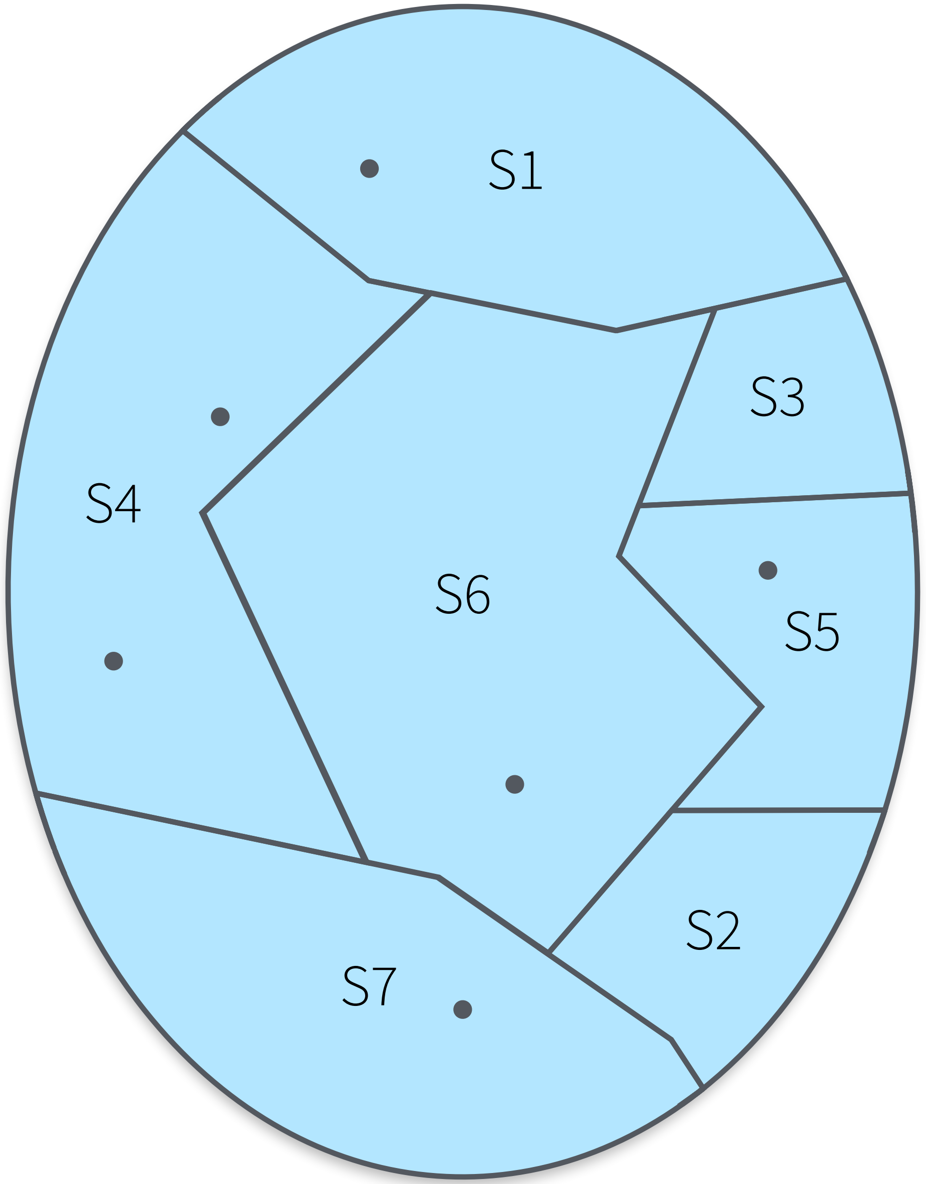
Metasketches express structure and strategy

1. **Search order** is critical
2. Desire **optimal** solutions

<u>**A metasketch contains:**</u>

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

2. cost function (κ)



Syntax

Metasketches express structure and strategy

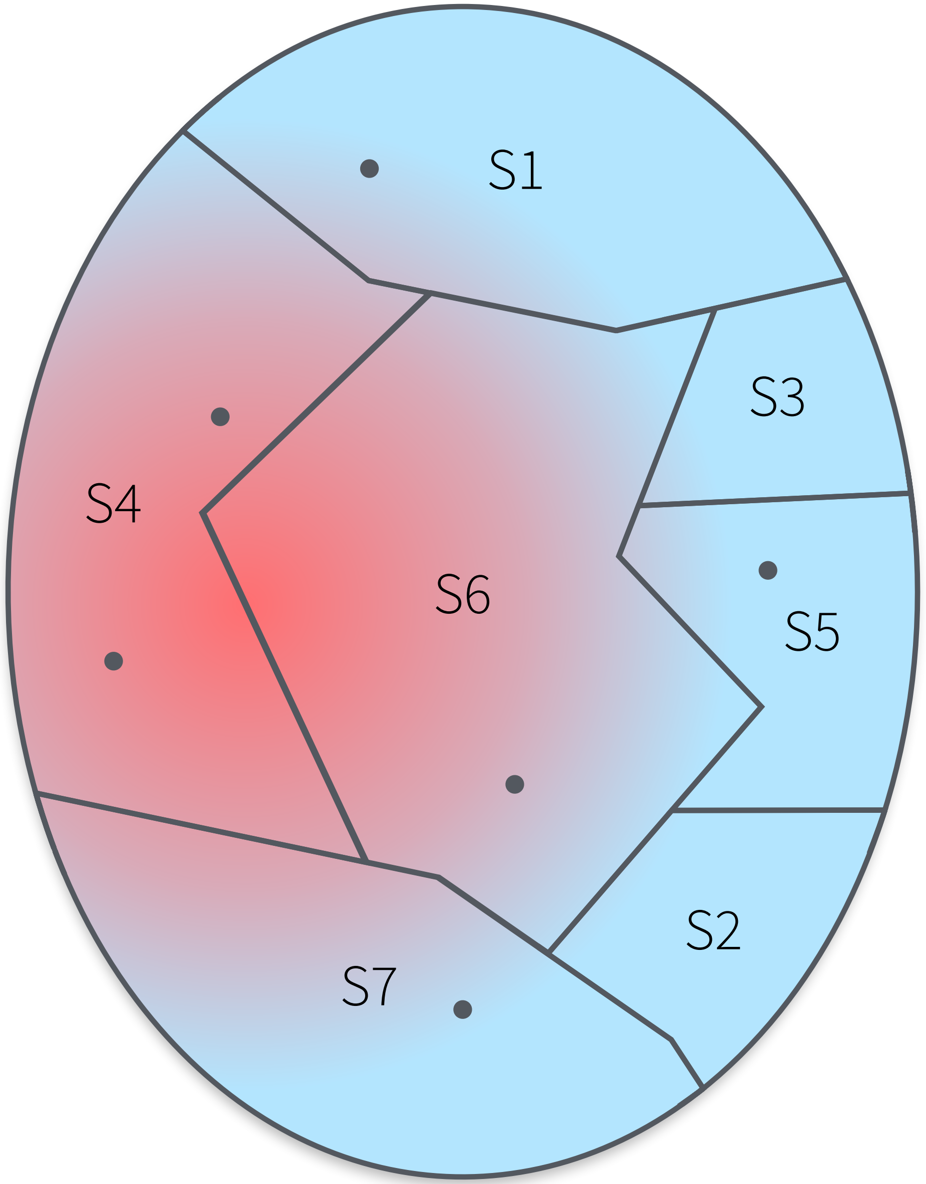
1. **Search order** is critical
2. Desire **optimal** solutions

<u>A metasketch contains:</u>

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

2. cost function (κ)

3. gradient function (g)



Syntax

Metasketches express structure and strategy

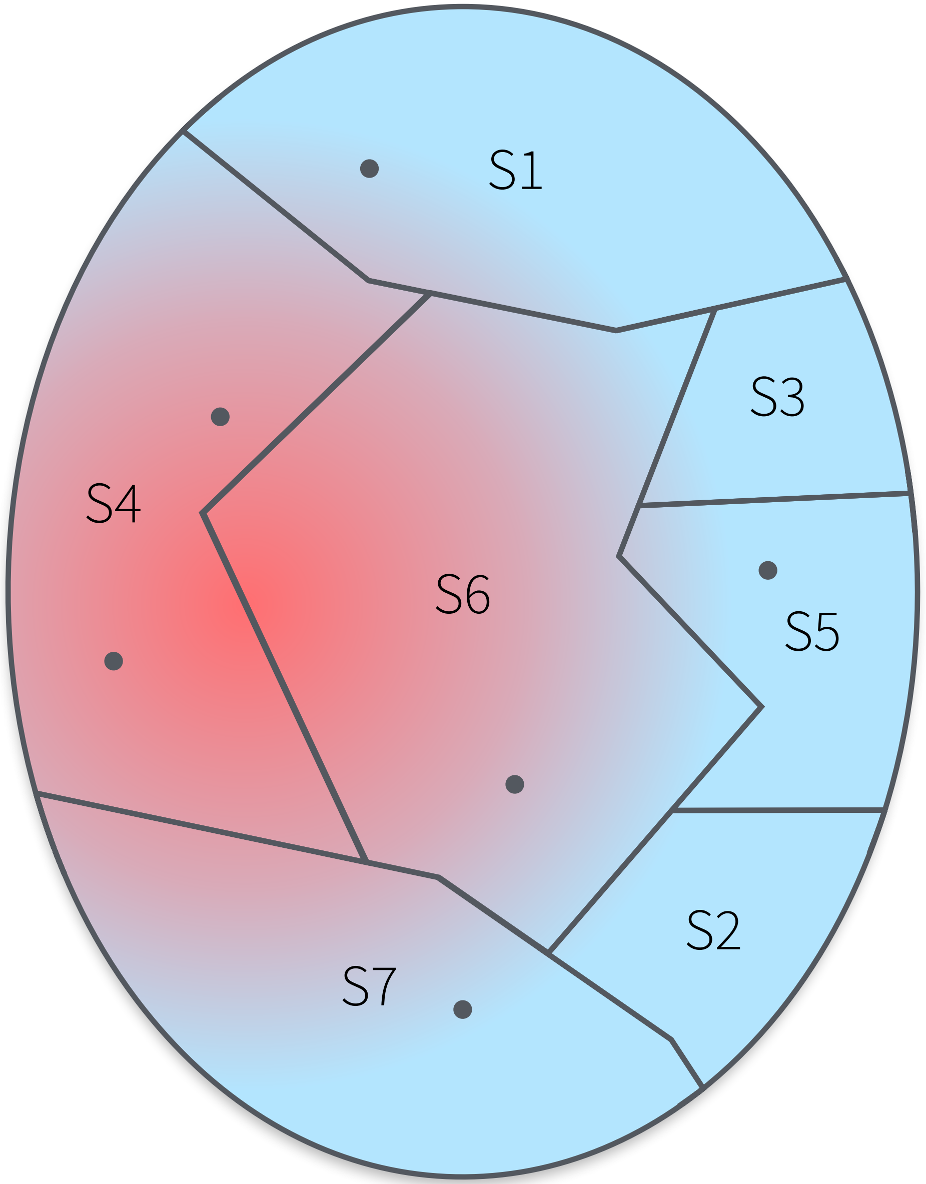
1. **Search order** is critical
2. Desire **optimal** solutions

<u>**A metasketch contains:**</u>

1. structured candidate space ($\mathcal{S}$, ≼)

2. cost function (κ)

3. gradient function (g)



Syntax

Metasketches express structure and strategy

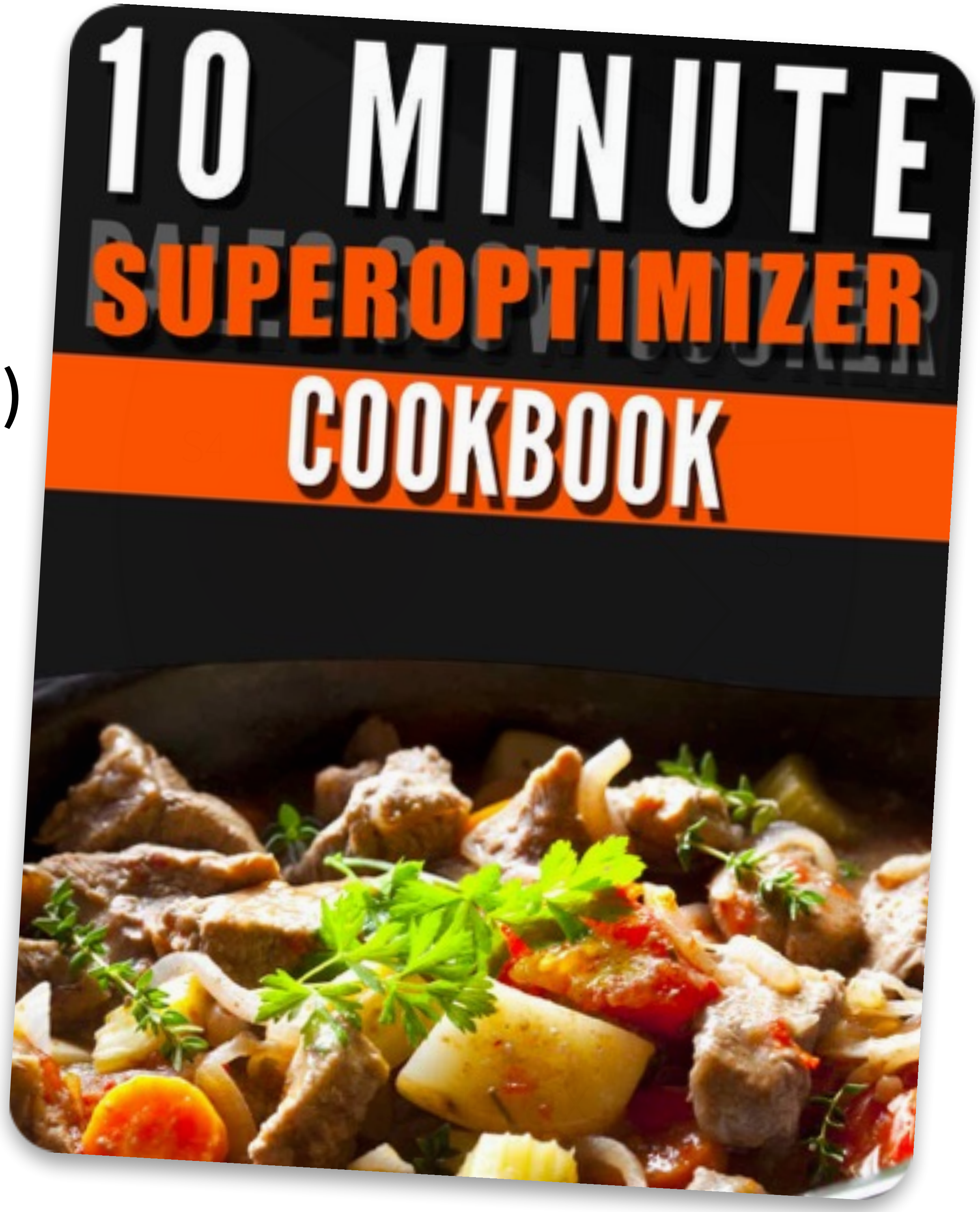
1. **Search order** is critical
2. Desire **optimal** solutions

A metasketch contains:

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

2. cost function (κ)

3. gradient function (g)



Metasketches express structure and strategy

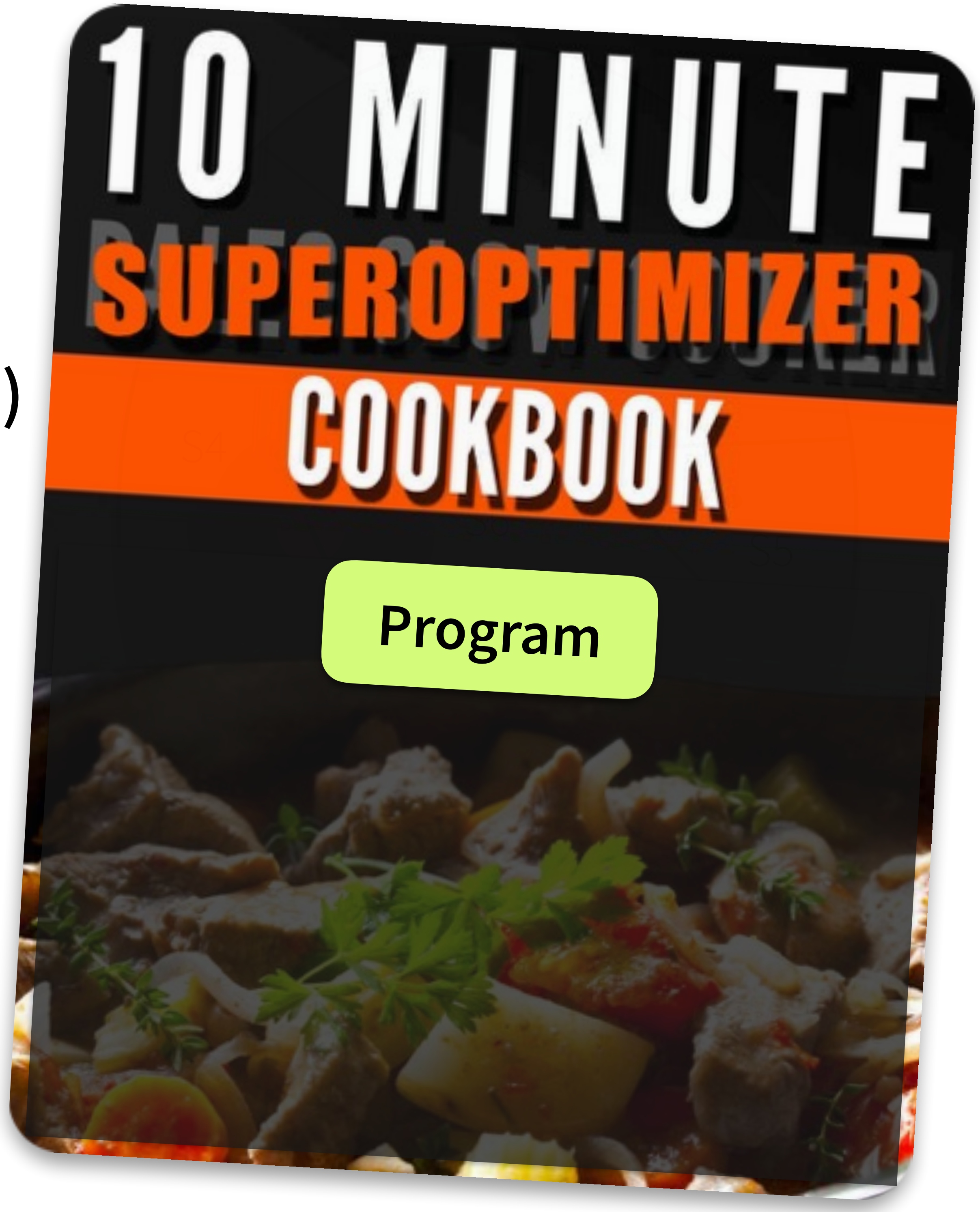
1. **Search order** is critical
2. Desire **optimal** solutions

<u>**A metasketch contains:**</u>

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

2. cost function (κ)

3. gradient function (g)



Metasketches express structure and strategy

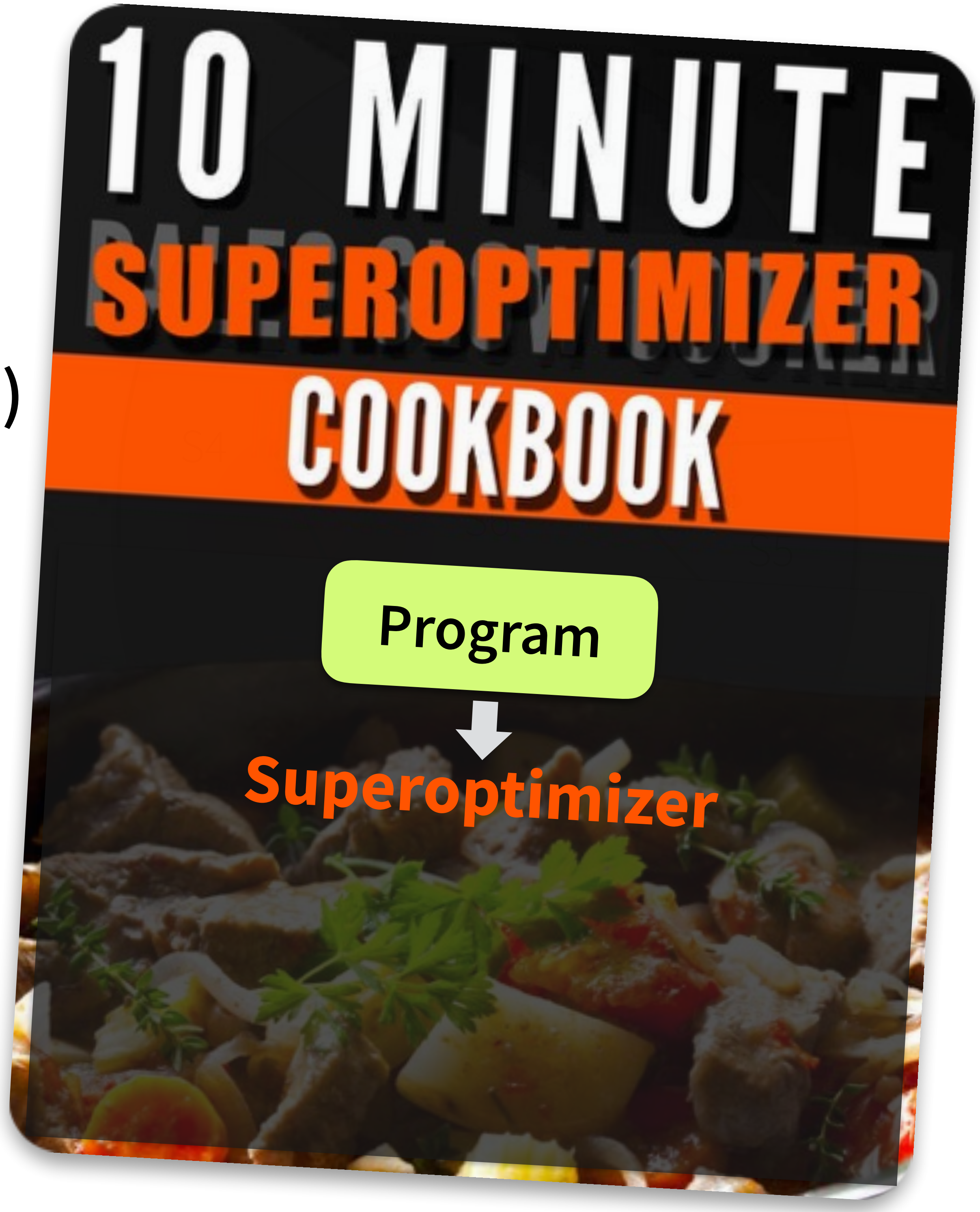
1. **Search order** is critical
2. Desire **optimal** solutions

A metasketch contains:

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

2. cost function (κ)

3. gradient function (g)



Metasketches express structure and strategy

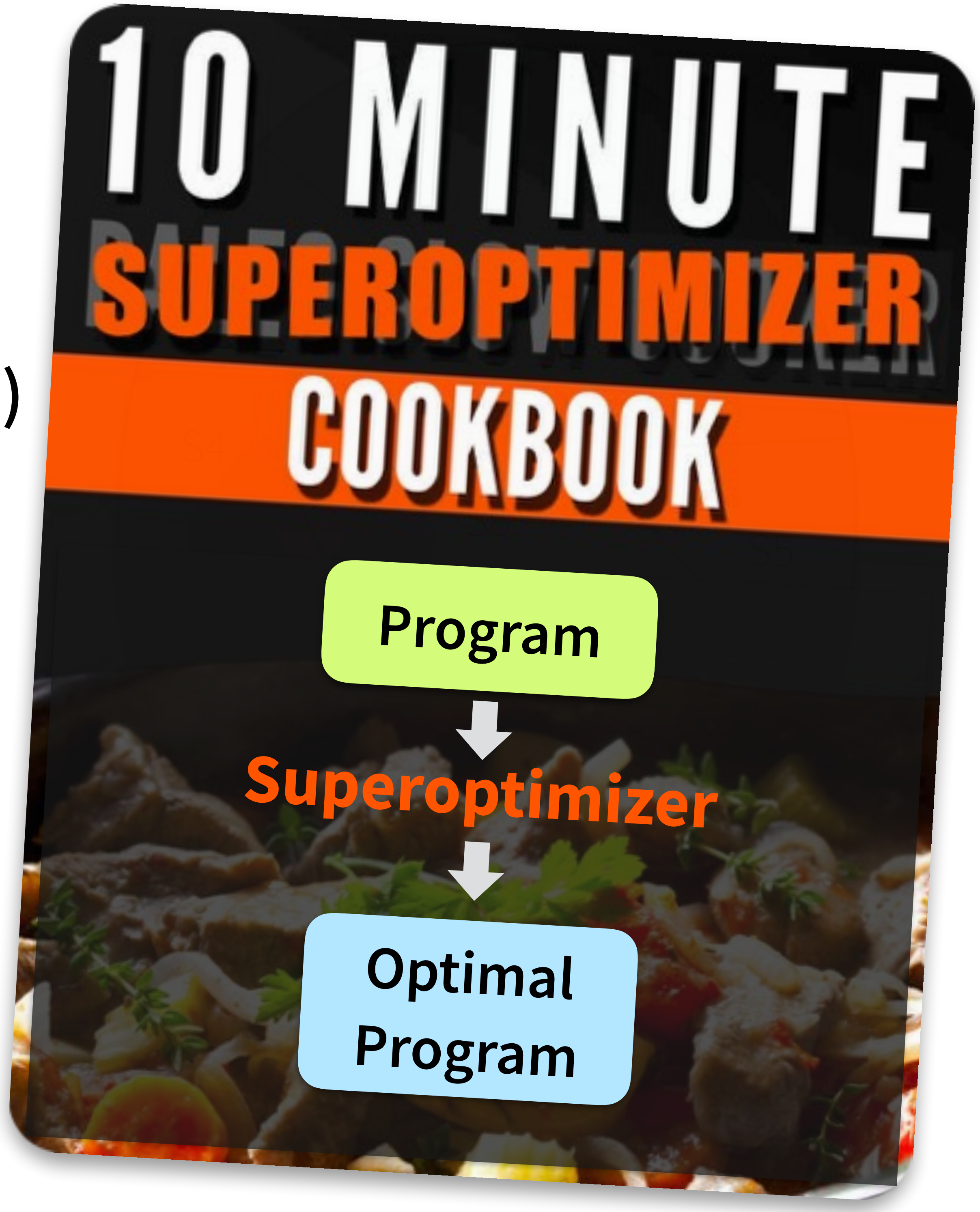
1. **Search order** is critical
2. Desire **optimal** solutions

<u>A metasketch contains:</u>

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

2. cost function (κ)

3. gradient function (g)



Program

Superoptimizer

Optimal Program

Metasketches express structure and strategy

1. structured candidate space $(\mathcal{S}, \preccurlyeq)$

2. cost function (κ)

3. gradient function (g)

Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

A fragmentation of the candidate space, and an ordering on those fragments.

2. cost function (κ)

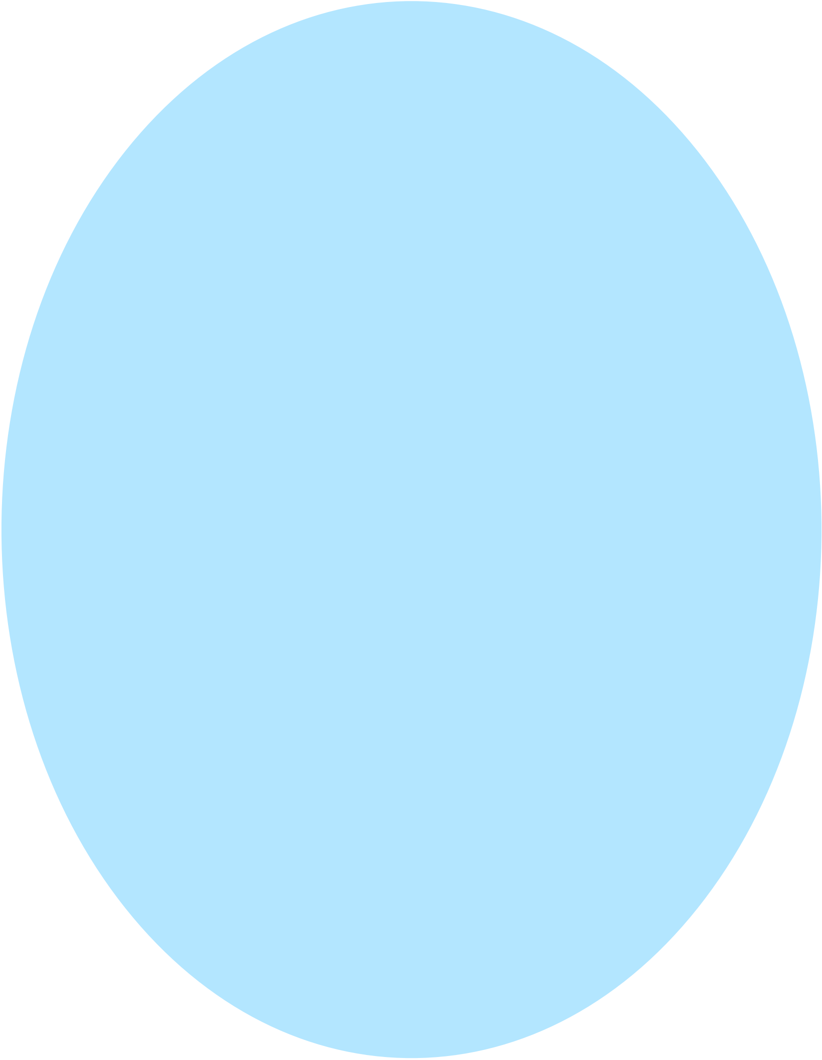
3. gradient function (g)

Metasketches express structure and strategy

1. structured candidate space $(\mathcal{S}, \preccurlyeq)$

A fragmentation of the candidate space, and an ordering on those fragments.

$\mathcal{S}$ = set of all SSA programs



2. cost function (κ)

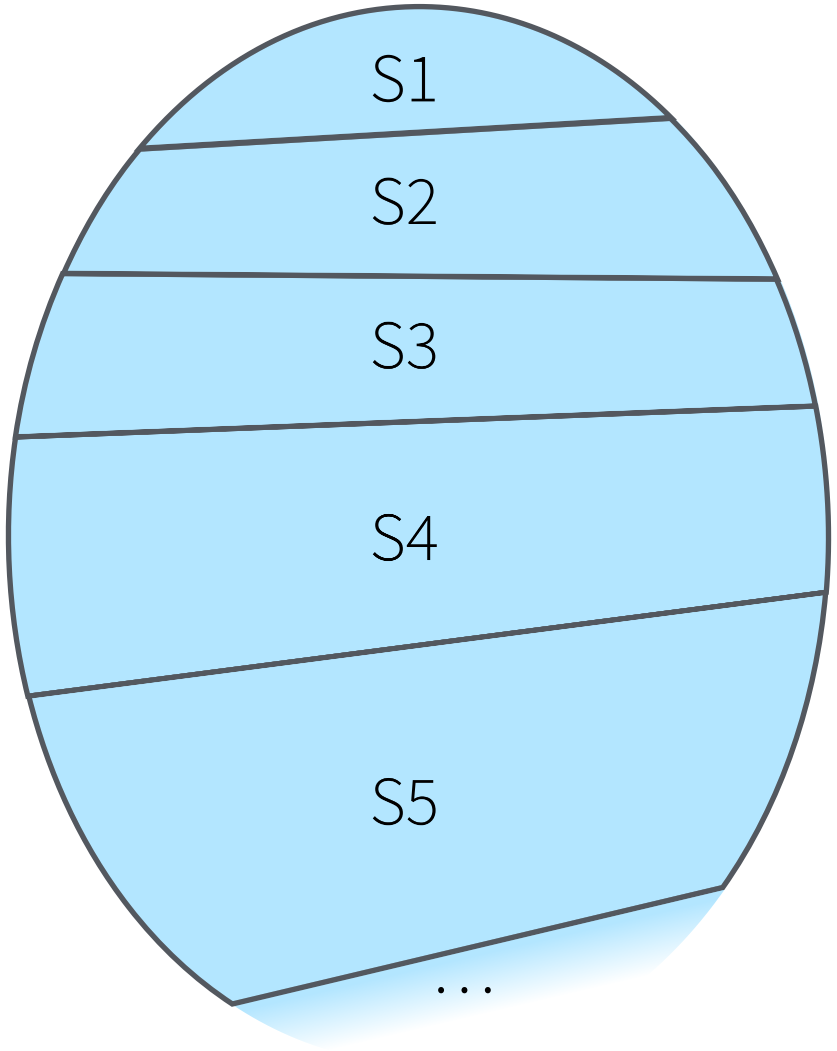
3. gradient function (g)

Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

A fragmentation of the candidate space, and an ordering on those fragments.

$\mathcal{S}$ = set of all SSA programs



2. cost function (κ)

3. gradient function (g)

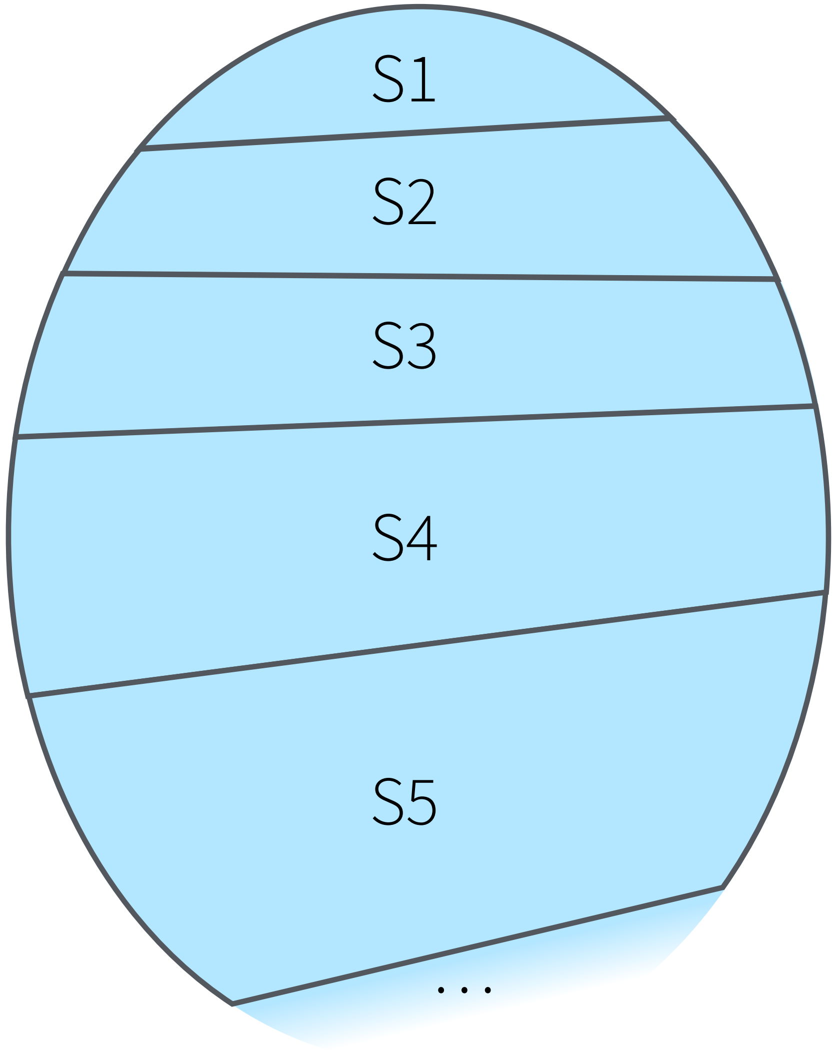
Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

- a countable set $\mathcal{S}$ of sketches
- a total order $\preccurlyeq$ on $\mathcal{S}$

A fragmentation of the candidate space, and an ordering on those fragments.

$\mathcal{S}$ = set of all SSA programs



2. cost function (κ)

3. gradient function (g)

Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

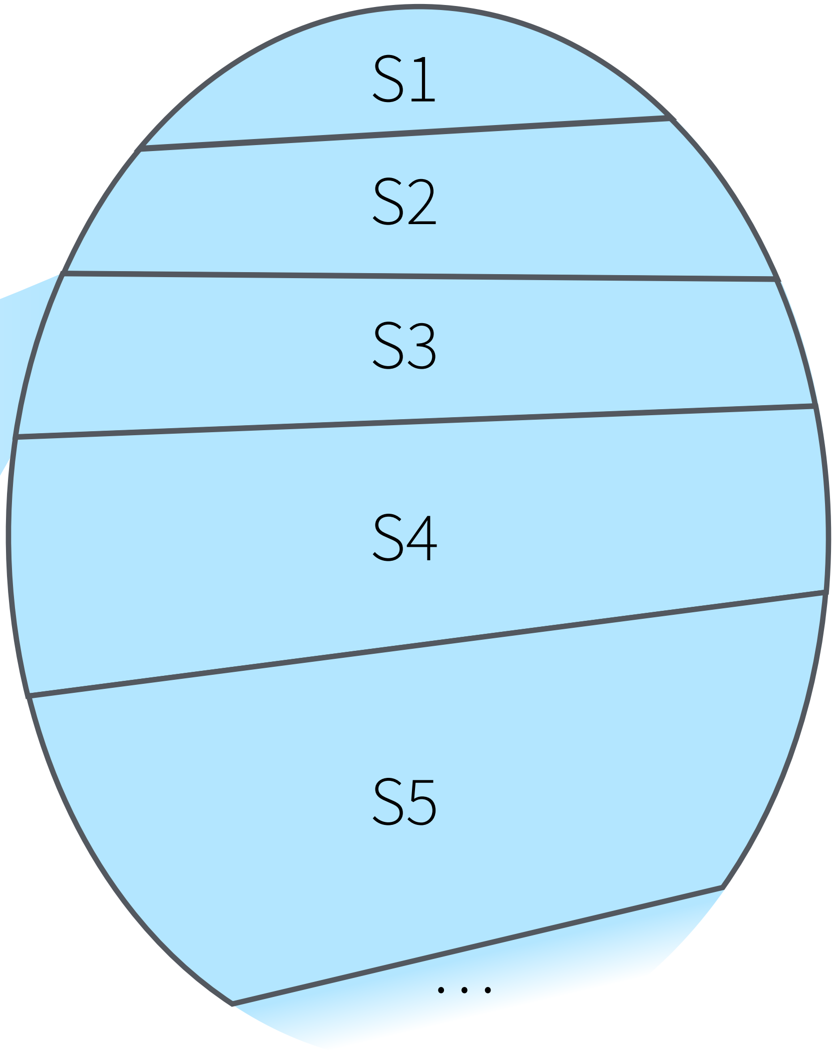
- a countable set $\mathcal{S}$ of sketches
- a total order $\preccurlyeq$ on $\mathcal{S}$

A fragmentation of the candidate space, and an ordering on those fragments.

S₃ (SSA programs of length 3)

```
def f(x):
  r1 = ??op(??{x})
  r2 = ??op(??{x,r1})
  r3 = ??op(??{x,r1,r2})
  return r3
```

$\mathcal{S}$ = set of all SSA programs



2. cost function (κ)

3. gradient function (g)

Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

- a countable set $\mathcal{S}$ of sketches
- a total order $\preccurlyeq$ on $\mathcal{S}$

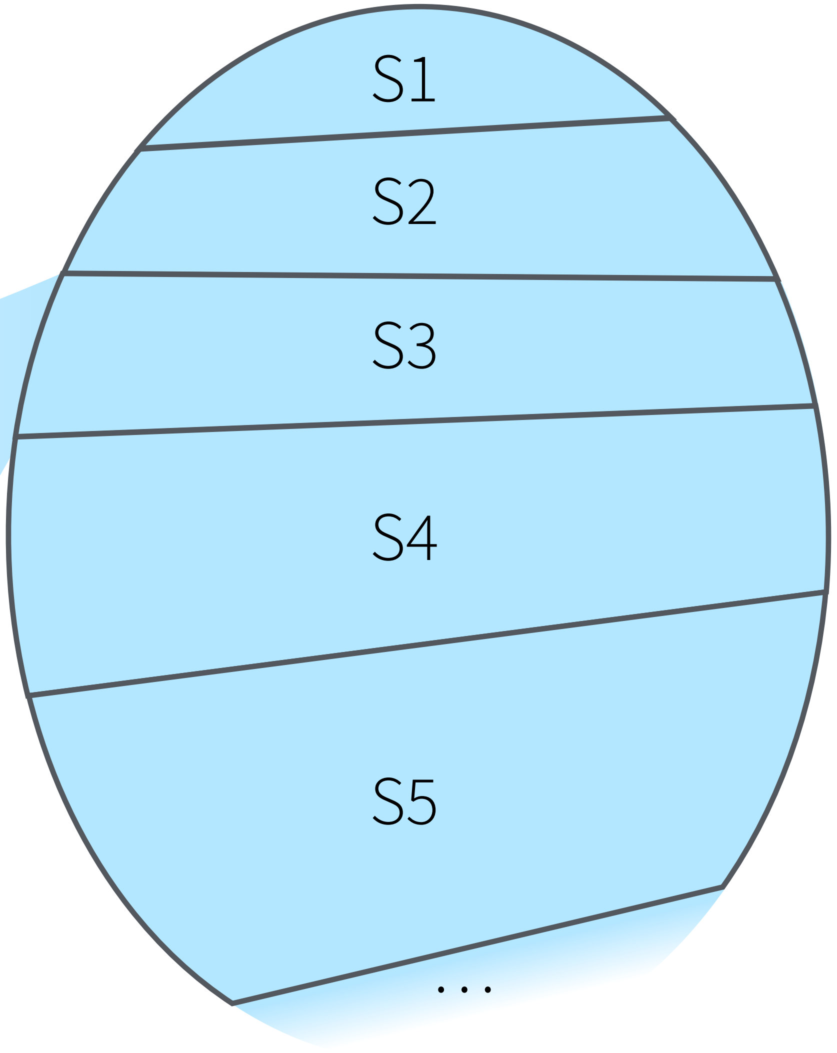
A fragmentation of the candidate space, and an ordering on those fragments.

S_3 (SSA programs of length 3)

+, -, <, if, . . .

```
def f(x):
   r1 = ??op(??{x})
   r2 = ??op(??{x,r1})
   r3 = ??op(??{x,r1,r2})
   return r3
```

$\mathcal{S}$ = set of all SSA programs



2. cost function (κ)

3. gradient function (g)

Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

- a countable set $\mathcal{S}$ of sketches
- a total order $\preccurlyeq$ on $\mathcal{S}$

A fragmentation of the candidate space, and an ordering on those fragments.

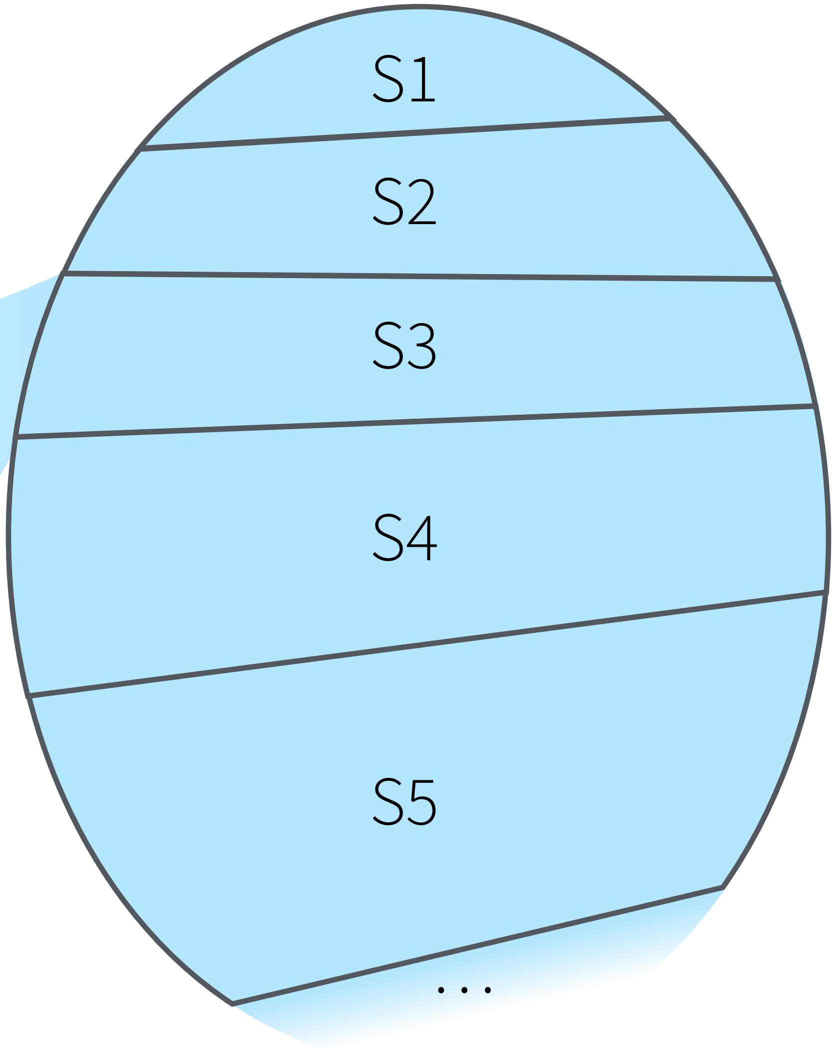
S_3 (SSA programs of length 3)

```
def f(x):
   r1 = ??_op(??_{x})
   r2 = ??_op(??_{x,r1})
   r3 = ??_op(??_{x,r1,r2})
   return r3
```

+, -, <, if, . . .

Vars & constants

$\mathcal{S}$ = set of all SSA programs



2. cost function (κ)

3. gradient function (g)

Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

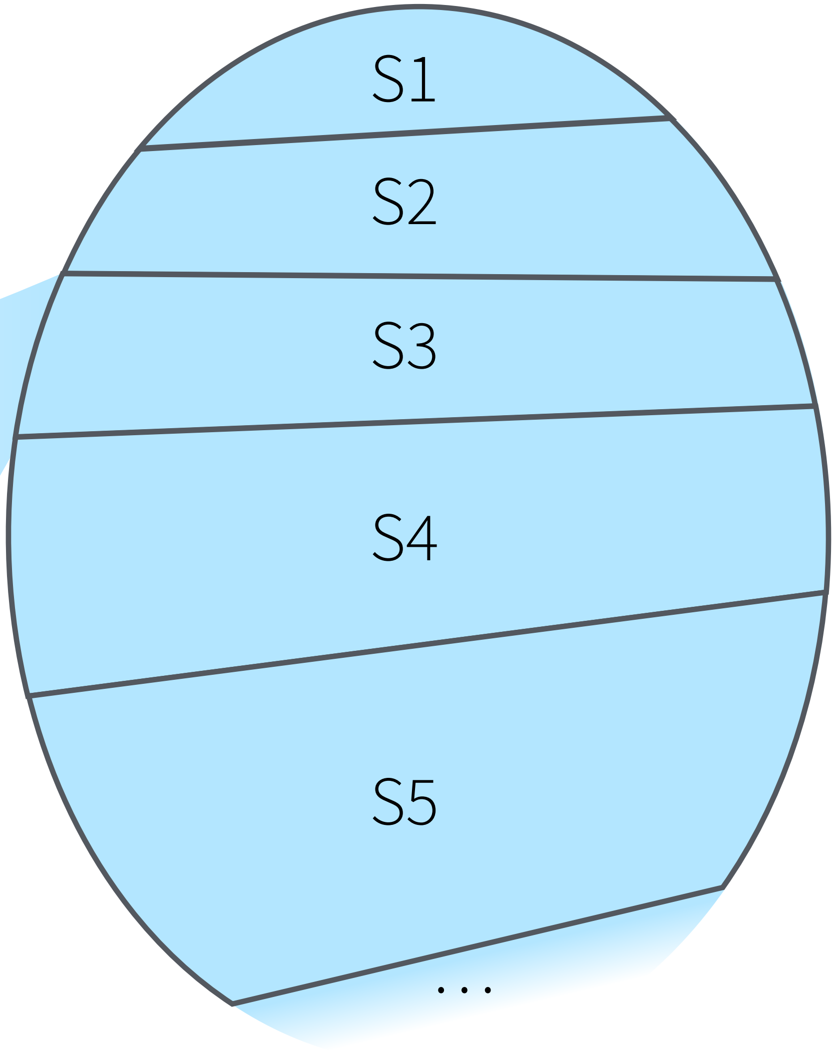
- a countable set $\mathcal{S}$ of sketches
- a total order $\preccurlyeq$ on $\mathcal{S}$

A fragmentation of the candidate space, and an ordering on those fragments.

S_3 (SSA programs of length 3)

```
def f(x):
   r1 = ??op(??{x})
   r2 = ??op(??{x,r1})
   r3 = ??op(??{x,r1,r2})
   return r3
```

$\mathcal{S}$ = set of all SSA programs



2. cost function (κ)

3. gradient function (g)

Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

- a countable set $\mathcal{S}$ of sketches
- a total order $\preccurlyeq$ on $\mathcal{S}$

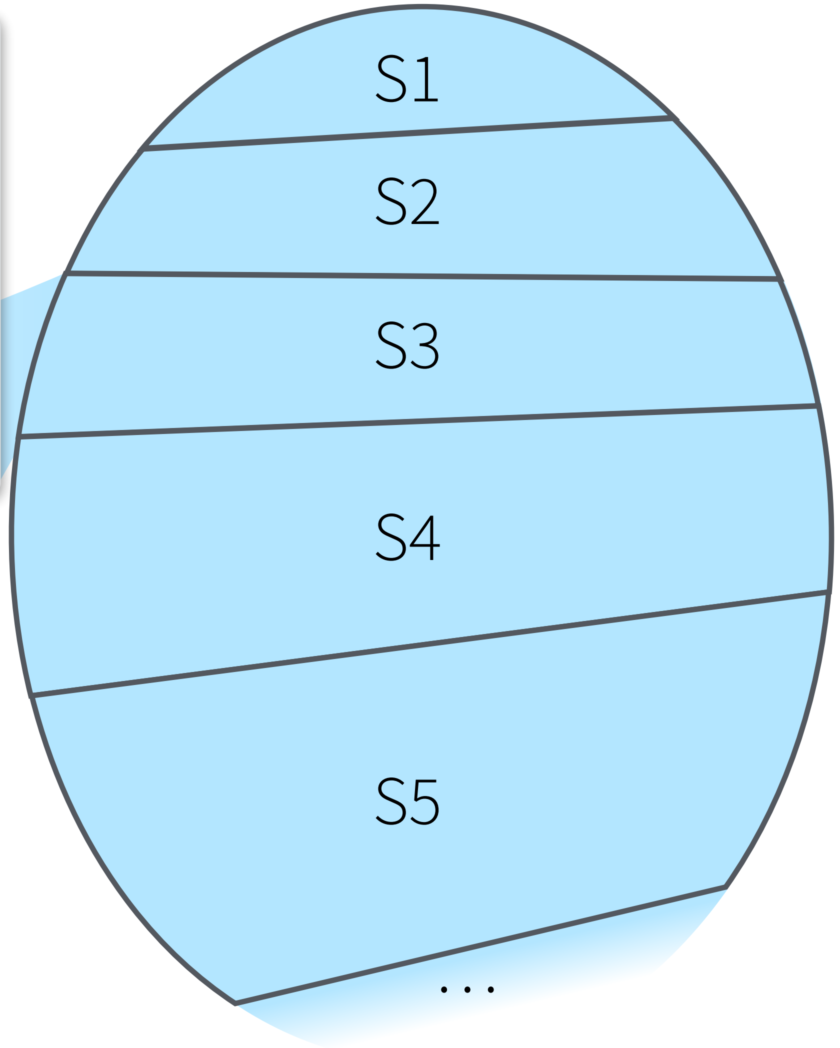
Ordering expresses high-level search strategy.

A fragmentation of the candidate space, and an ordering on those fragments.

S_3 (SSA programs of length 3)

```
def f(x):
   r1 = ??op(??{x})
   r2 = ??op(??{x,r1})
   r3 = ??op(??{x,r1,r2})
   return r3
```

$\mathcal{S}$ = set of all SSA programs



2. cost function (κ)

3. gradient function (g)

Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

- a countable set $\mathcal{S}$ of sketches
- a total order $\preccurlyeq$ on $\mathcal{S}$

A fragmentation of the candidate space, and an ordering on those fragments.

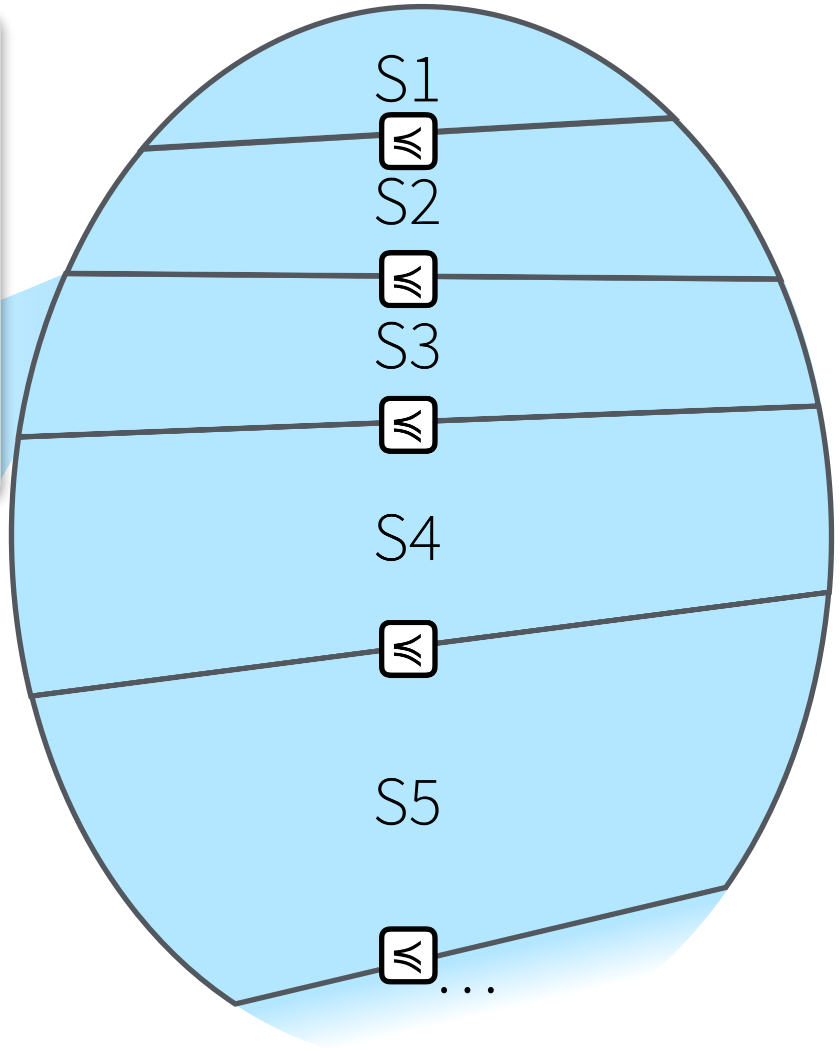
Ordering expresses high-level search strategy.

Here, $\preccurlyeq$ expresses iterative deepening.

S_3 (SSA programs of length 3)

```
def f(x):
   r_1 = ??_op(??_{x})
   r_2 = ??_op(??_{x,r_1})
   r_3 = ??_op(??_{x,r_1,r_2})
   return r_3
```

$\mathcal{S}$ = set of all SSA programs



2. cost function (κ)

3. gradient function (g)

Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

- a countable set $\mathcal{S}$ of sketches
- a total order $\preccurlyeq$ on $\mathcal{S}$

A fragmentation of the candidate space, and an ordering on those fragments.

$\mathcal{S}$ = set of all SSA programs

S_1

```
def f(x):
  r1 = ??_op(??_{x})
  return r1
```

S_2

```
def f(x):
  r1 = ??_op(??_{x})
  r2 = ??_op(??_{x,r1})
  return r2
```

S_3

```
def f(x):
  r1 = ??_op(??_{x})
  r2 = ??_op(??_{x,r1})
  r3 = ??_op(??_{x,r1,r2})
  return r3
```

...

Implemented as a generator that returns the next sketch in the space

2. cost function (κ)

3. gradient function (g)

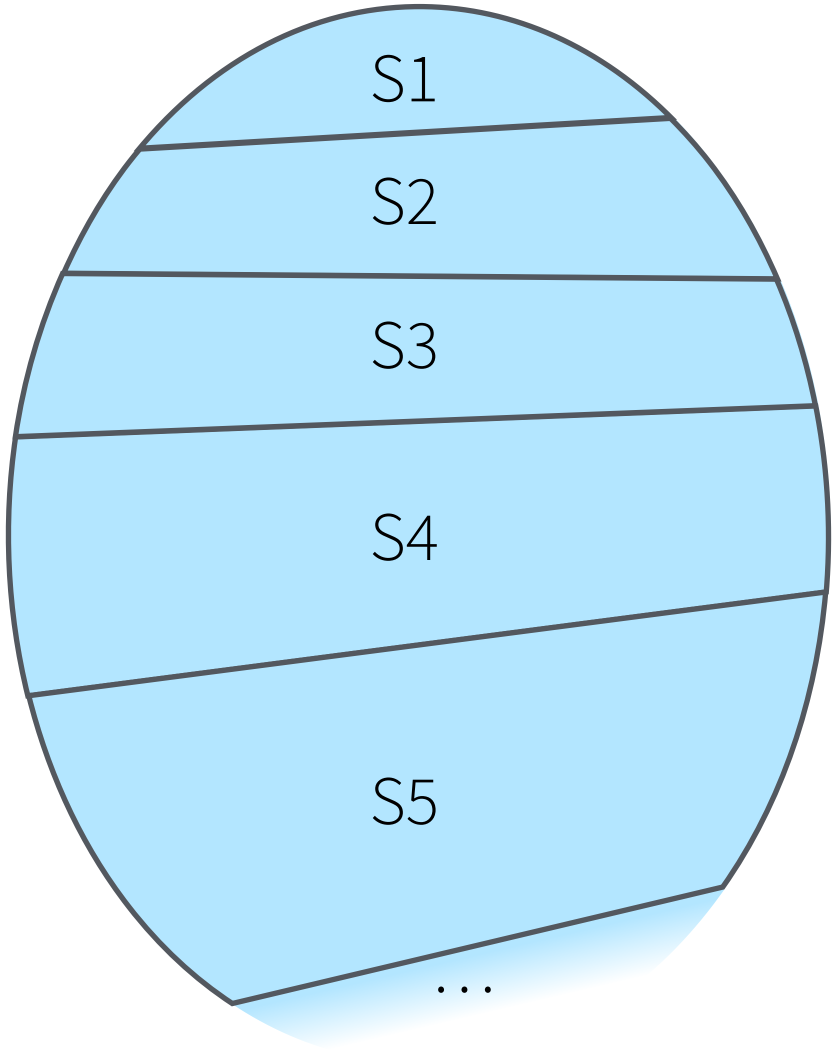
Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

- a countable set $\mathcal{S}$ of sketches
- a total order $\preccurlyeq$ on $\mathcal{S}$

Semantics

$\mathcal{S}$ = set of all SSA programs



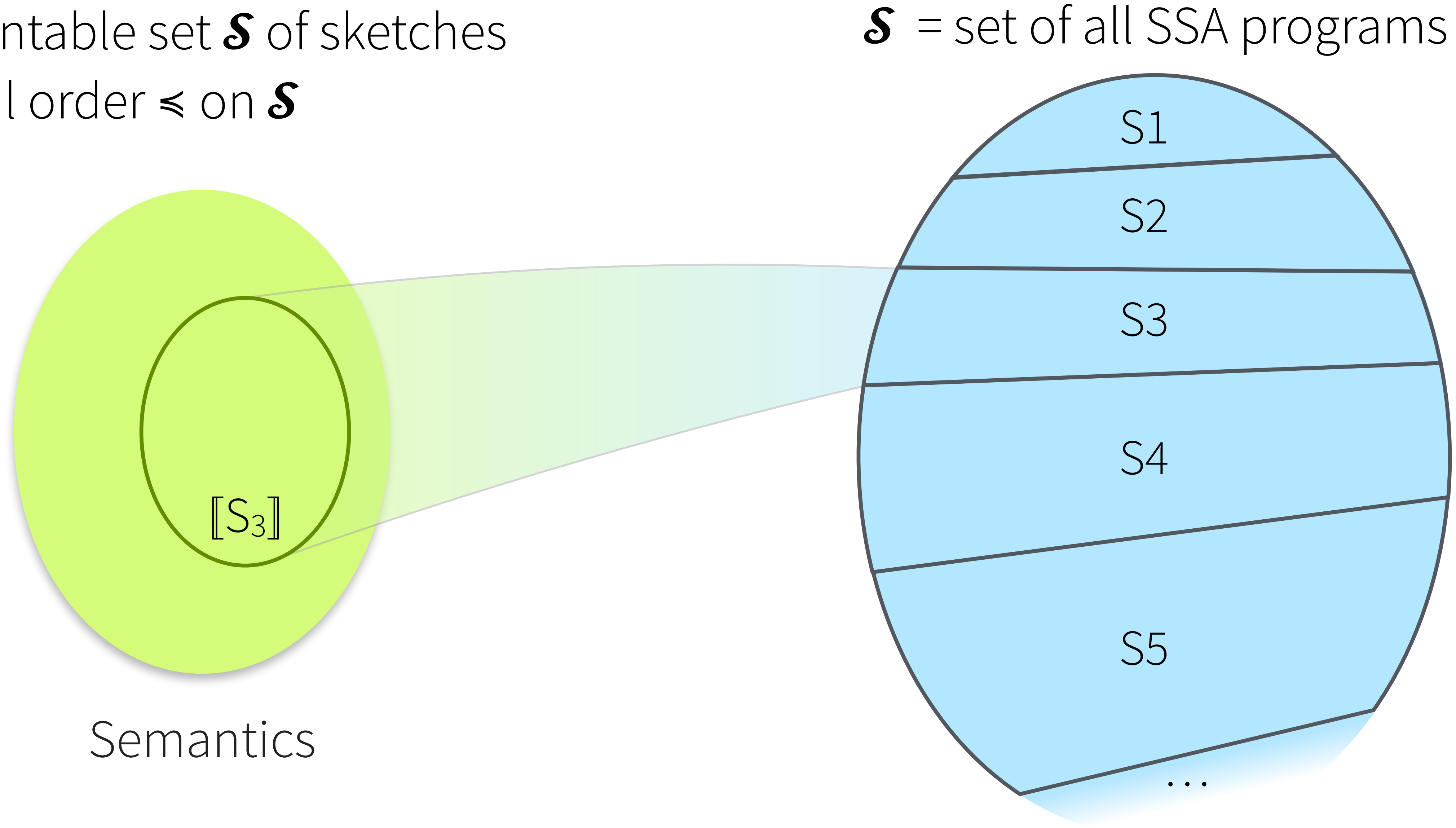
2. cost function (κ)

3. gradient function (g)

Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

- a countable set $\mathcal{S}$ of sketches
- a total order $\preccurlyeq$ on $\mathcal{S}$



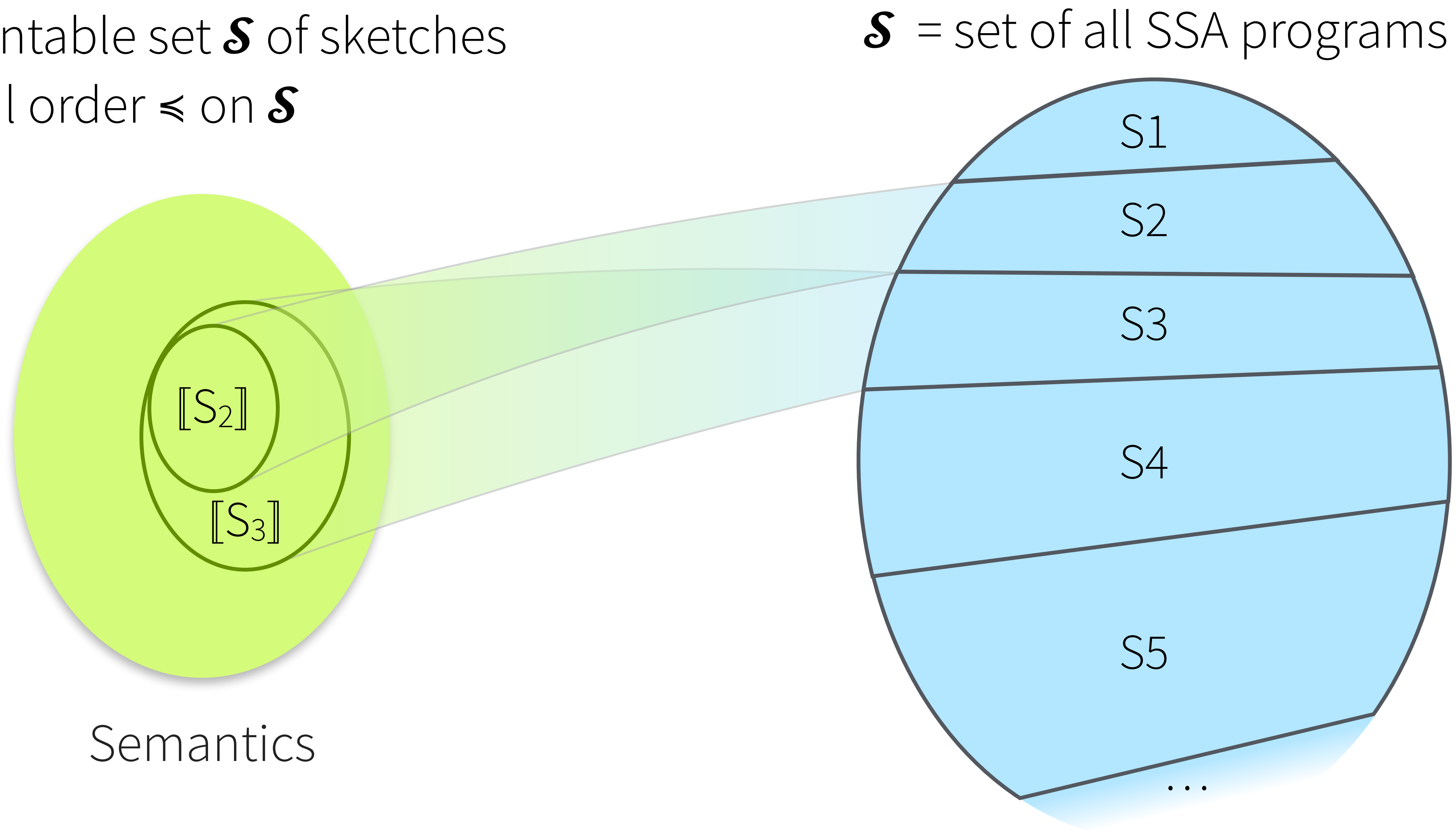
2. cost function (κ)

3. gradient function (g)

Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, ≼)

- a countable set $\mathcal{S}$ of sketches
- a total order ≼ on $\mathcal{S}$



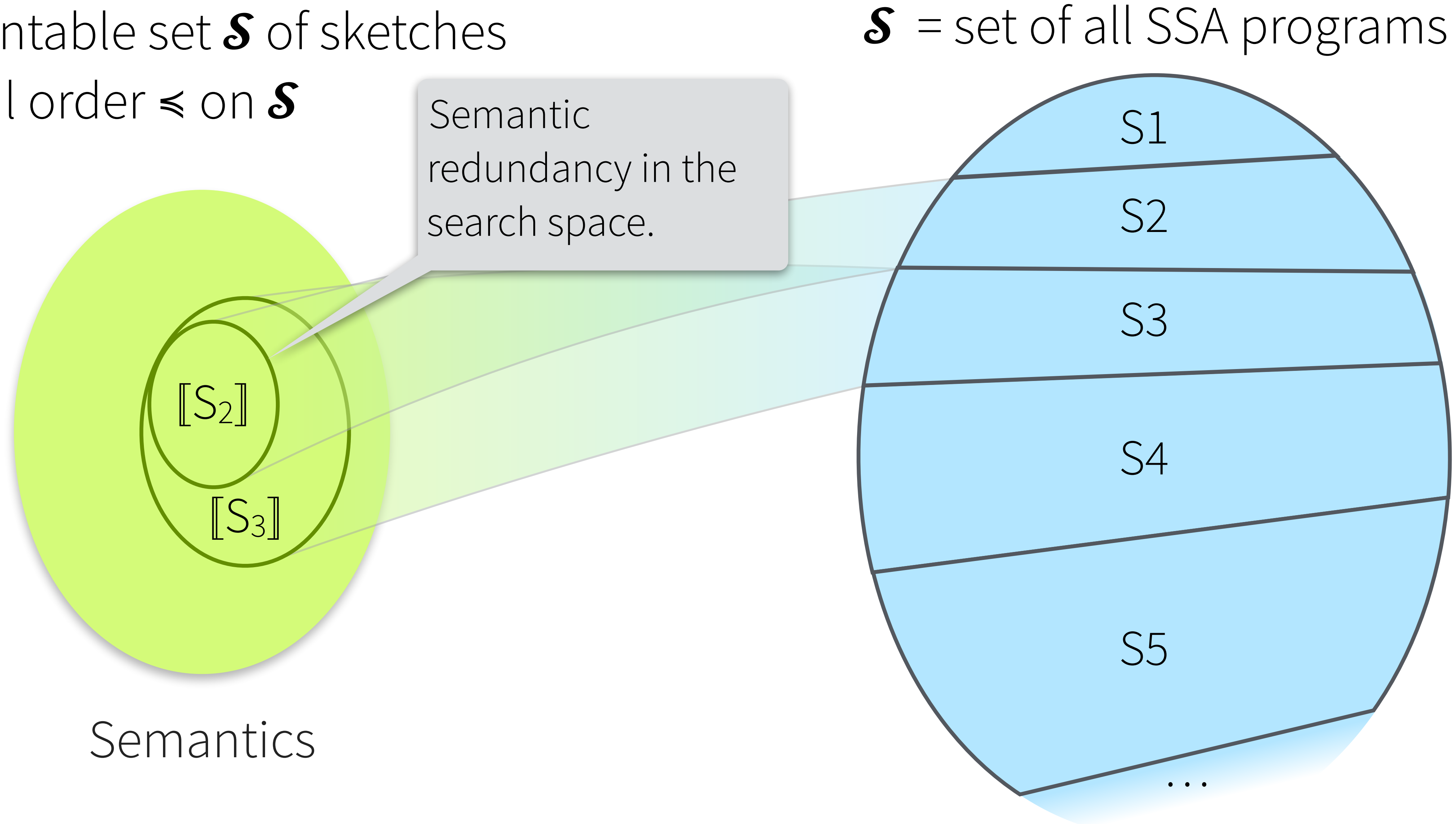
2. cost function (κ)

3. gradient function (g)

Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, ≼)

- a countable set $\mathcal{S}$ of sketches
- a total order ≼ on $\mathcal{S}$



$\mathcal{S}$ = set of all SSA programs
Semantic redundancy in the search space.
⟦S₂⟧
⟦S₃⟧
S1
S2
S3
S4
S5
...

Semantics

2. cost function (κ)

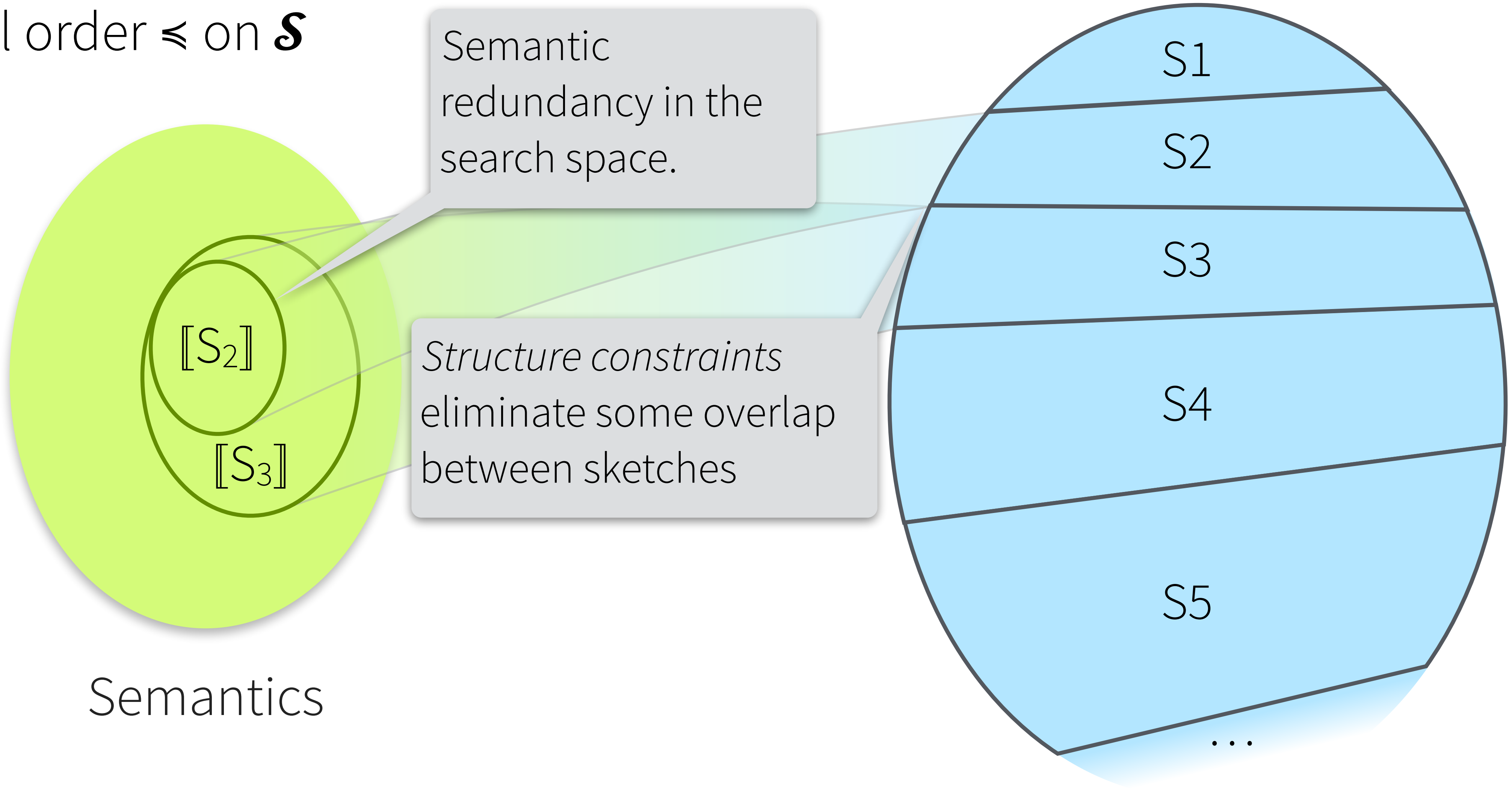
3. gradient function (g)

Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

- a countable set $\mathcal{S}$ of sketches
- a total order $\preccurlyeq$ on $\mathcal{S}$

$\mathcal{S}$ = set of all SSA programs



Semantic redundancy in the search space.

Structure constraints eliminate some overlap between sketches

2. cost function (κ)

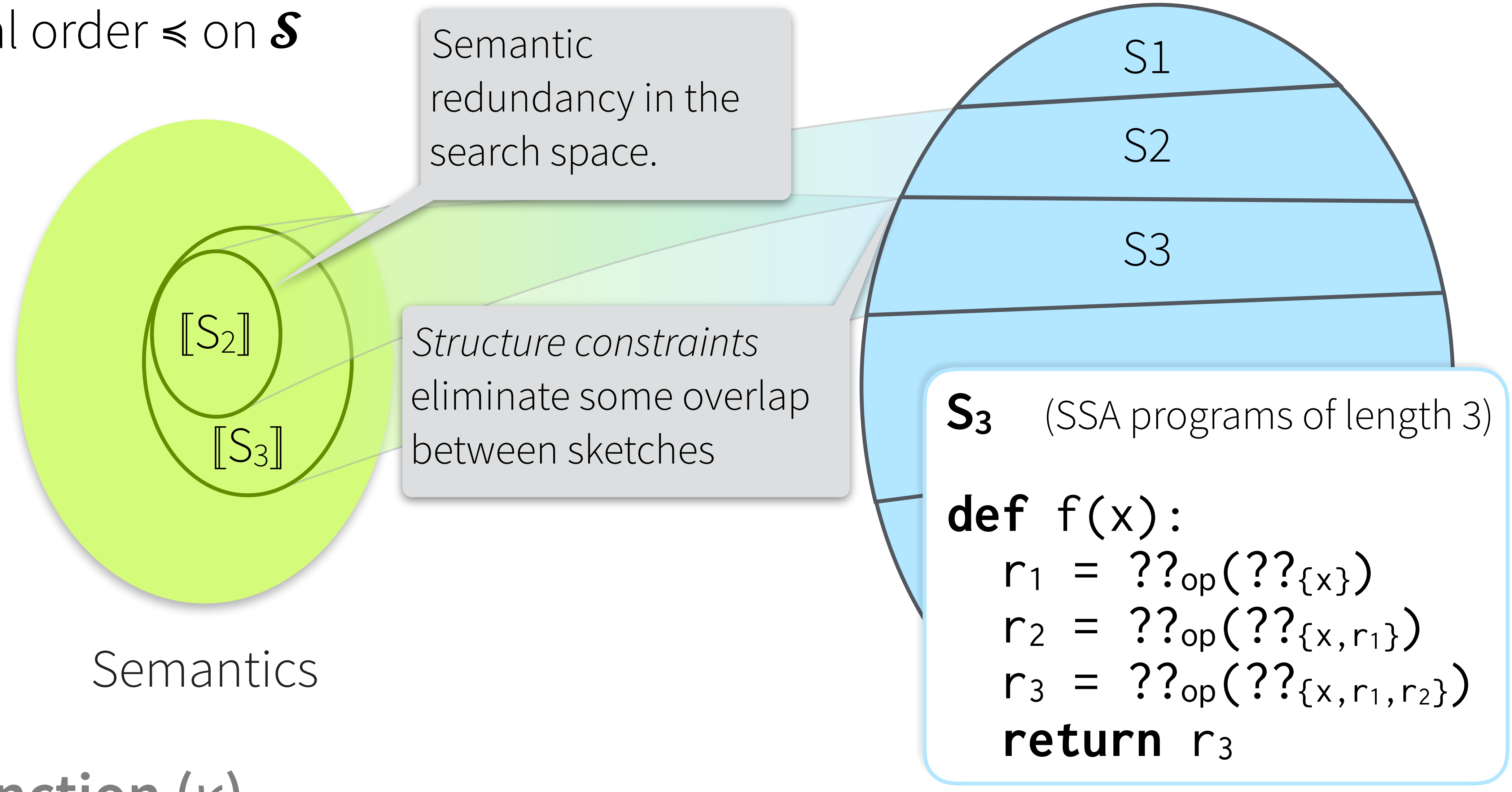
3. gradient function (g)

Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

- a countable set $\mathcal{S}$ of sketches
- a total order $\preccurlyeq$ on $\mathcal{S}$

$\mathcal{S}$ = set of all SSA programs



Semantic redundancy in the search space.

Structure constraints eliminate some overlap between sketches

S_3 (SSA programs of length 3)

```
def f(x):
  r1 = ??op(??{x})
  r2 = ??op(??{x,r1})
  r3 = ??op(??{x,r1,r2})
  return r3
```

2. cost function (κ)

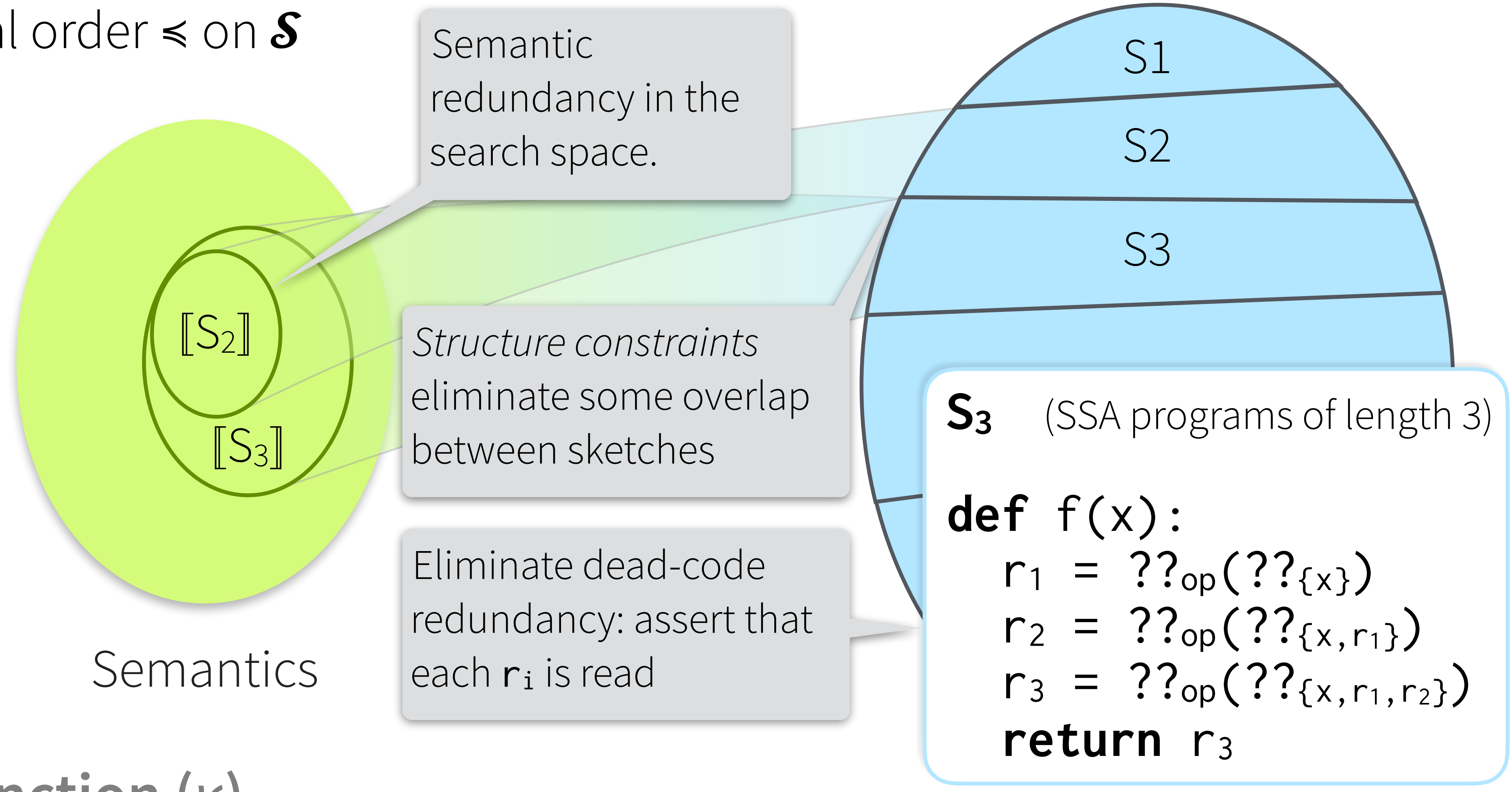
3. gradient function (g)

Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)

- a countable set $\mathcal{S}$ of sketches
- a total order $\preccurlyeq$ on $\mathcal{S}$

$\mathcal{S}$ = set of all SSA programs



Semantic redundancy in the search space.

Structure constraints eliminate some overlap between sketches

Eliminate dead-code redundancy: assert that each r_i is read

S_3 (SSA programs of length 3)

```
def f(x):
   r1 = ??op(??{x})
   r2 = ??op(??{x,r1})
   r3 = ??op(??{x,r1,r2})
   return r3
```

2. cost function (κ)

3. gradient function (g)

Cost functions rank candidate programs

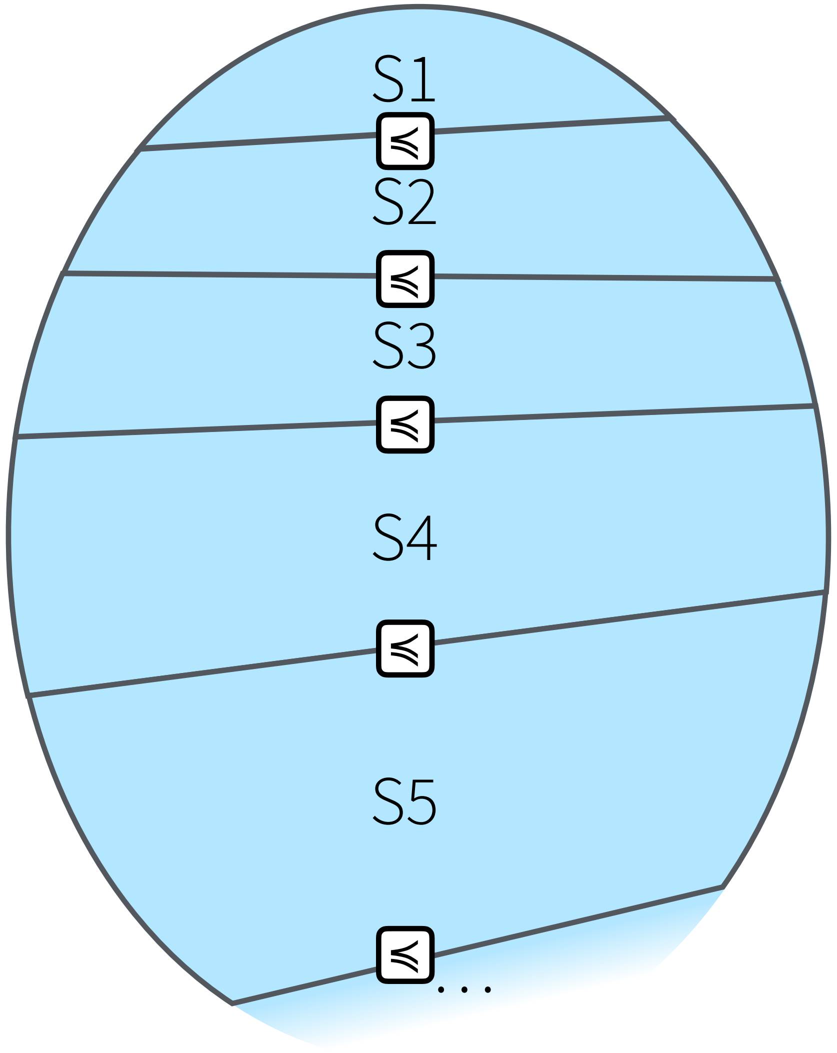
1. **structured candidate space ($\mathcal{S}$, ≼)**

2. **cost function (κ)**

   $\kappa : \mathcal{L} \to \mathbb{R}$

   assigns a numeric cost to each program in the language $\mathcal{L}$

$\mathcal{S}$ = set of all SSA programs



3. **gradient function (g)**

Cost functions rank candidate programs

1. structured candidate space ($\mathcal{S}$, ≼)

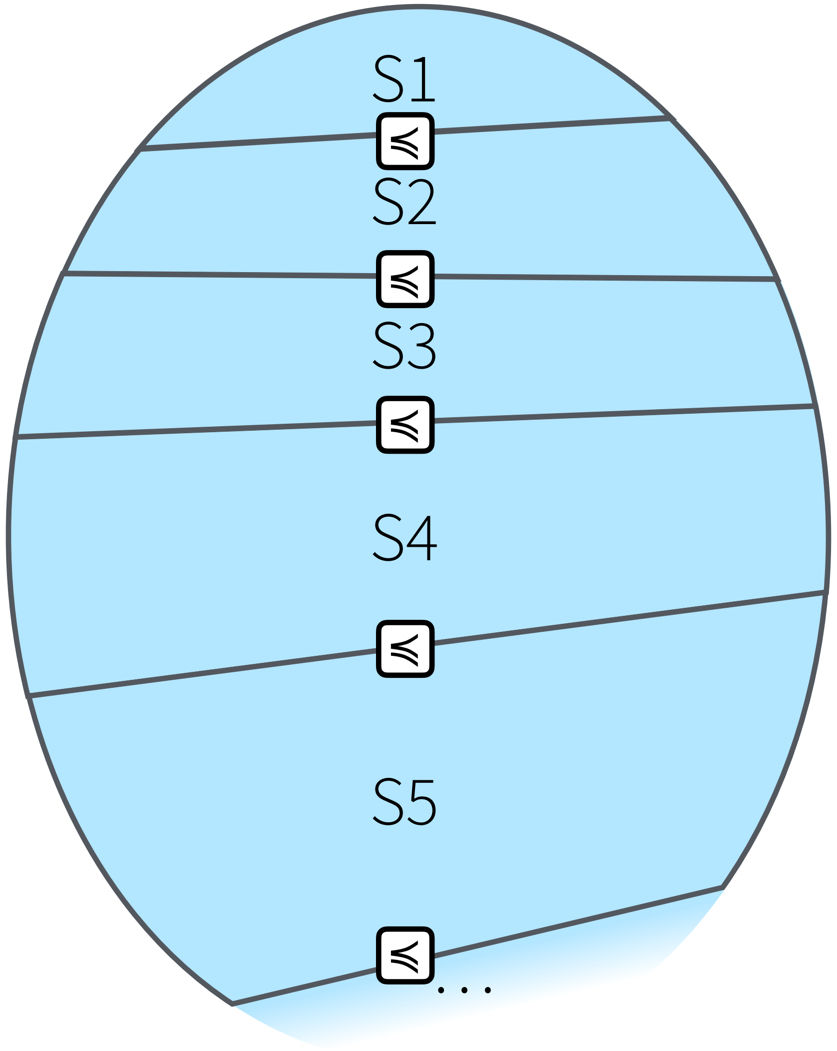
2. cost function (κ)

$\kappa : \mathcal{L} \rightarrow \mathbb{R}$

assigns a numeric cost to each program in the language $\mathcal{L}$

Cost functions can be based on both syntax and semantics (dynamic behavior)

$\mathcal{S}$ = set of all SSA programs



3. gradient function (g)

Cost functions rank candidate programs

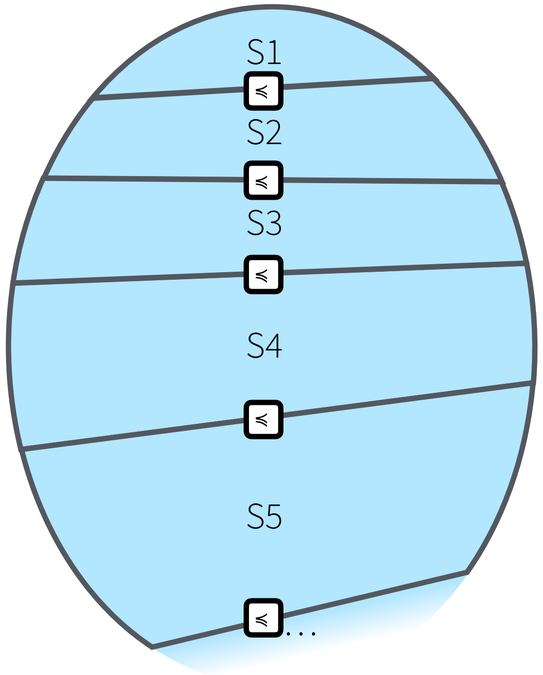
1. **structured candidate space ($\mathcal{S}$, ≼)**

2. **cost function (κ)**

$\kappa : \mathcal{L} \rightarrow \mathbb{R}$
assigns a numeric cost to each program in the language $\mathcal{L}$

Cost functions can be based on both syntax and semantics (dynamic behavior)

$\mathcal{S}$ = set of all SSA programs



$\kappa(P) = i \quad$ for $P \in S_i \in \mathcal{S}$

The number of variables defined in P

3. **gradient function (g)**

Gradient functions provide cost structure

1. structured candidate space ($\mathcal{S}$, ≼)

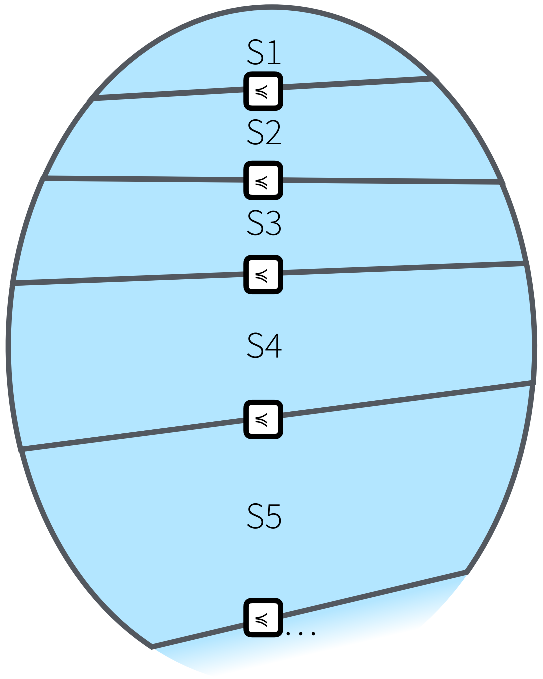
2. cost function (κ)

3. **gradient function (g)**

$g : \mathbb{R} \to 2^{\mathcal{S}}$

$g(c)$ is the set of sketches in $\mathcal{S}$ that *may* contain a solution P with $\kappa(P) < c$

$\mathcal{S}$ = set of all SSA programs



$\kappa(P) = i$ for $P \in S_i \in \mathcal{S}$

Gradient functions provide cost structure

1. **structured candidate space ($\mathcal{S}$, ≼)**

2. **cost function (κ)**

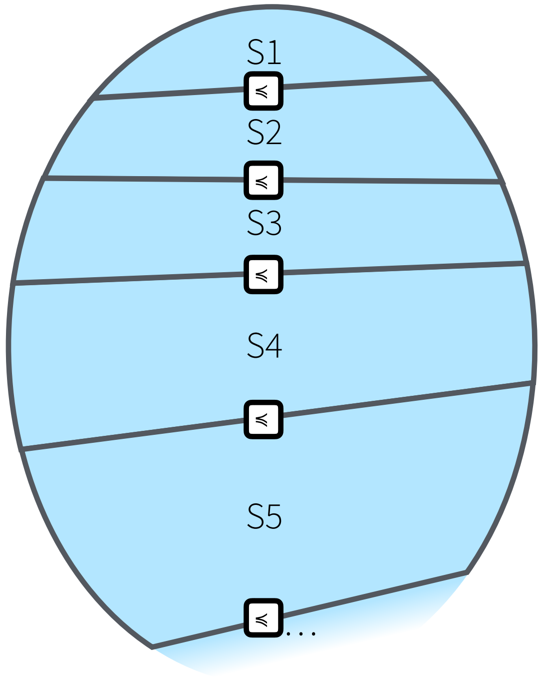
3. **gradient function (g)**

$g : \mathbb{R} \rightarrow 2^{\mathcal{S}}$

g(c) is the set of sketches in $\mathcal{S}$ that *may* contain a solution P with $\kappa(P) < c$

The gradient function overapproximates the behavior of **κ** on $\mathcal{S}$

$\mathcal{S}$ = set of all SSA programs



$\kappa(P) = i$ for $P \in S_i \in \mathcal{S}$

Gradient functions provide cost structure

1. **structured candidate space ($\mathcal{S}$, $\preccurlyeq$)**

2. **cost function (κ)**

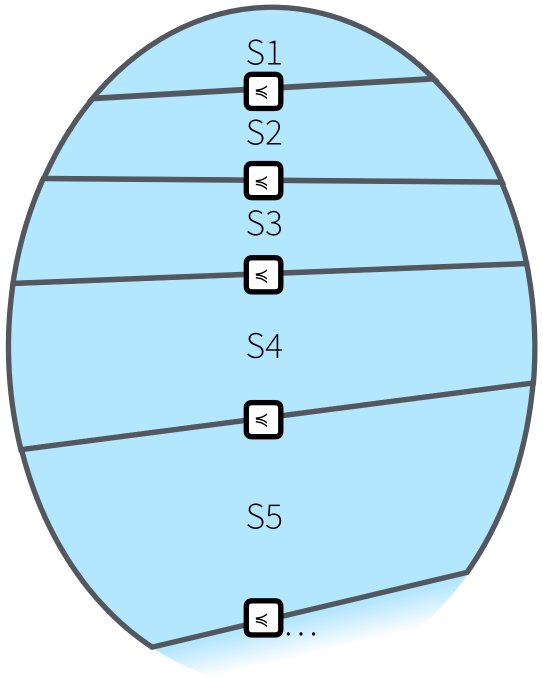
3. **gradient function (g)**

$g : \mathbb{R} \to 2^{\mathcal{S}}$

g(c) is the set of sketches in $\boldsymbol{\mathcal{S}}$ that *may* contain a solution P with $\kappa(P) < c$

The gradient function overapproximates the behavior of **κ** on $\boldsymbol{\mathcal{S}}$

$\boldsymbol{\mathcal{S}}$ = set of all SSA programs



$\kappa(P) = i$ for $P \in S_i \in \boldsymbol{\mathcal{S}}$

$g(c) = \{\, S_i \in \boldsymbol{\mathcal{S}} \mid i < c \,\}$

Gradient functions provide cost structure

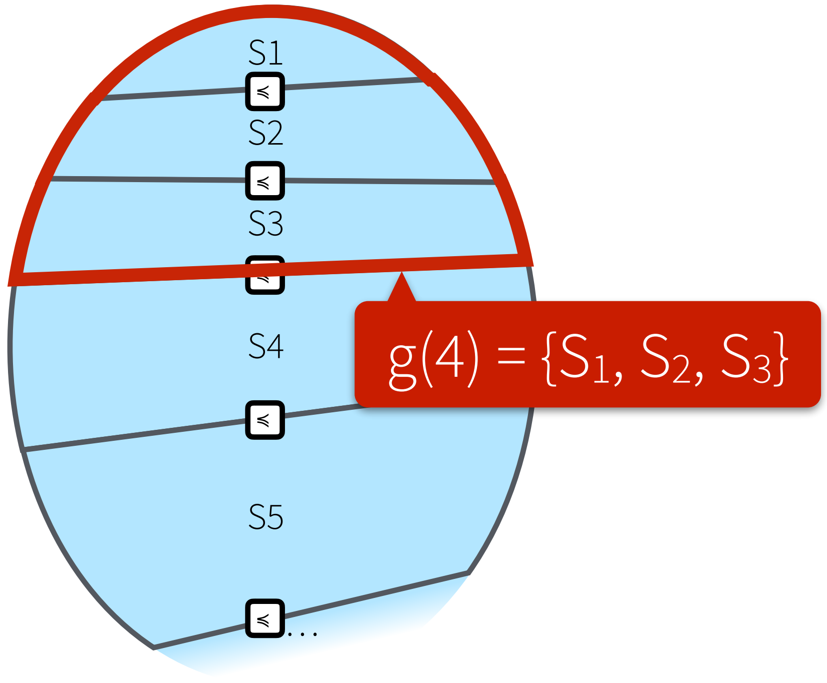
1. structured candidate space ($\mathcal{S}$, $\preccurlyeq$)
2. cost function (κ)
3. **gradient function (g)**

$g : \mathbb{R} \to 2^{\mathcal{S}}$

g(c) is the set of sketches in $\mathcal{S}$ that *may* contain a solution P with $\kappa(P) < c$

The gradient function overapproximates the behavior of κ on $\mathcal{S}$

$\mathcal{S}$ = set of all SSA programs



$g(4) = \{S_1, S_2, S_3\}$

$\kappa(P) = i \quad \text{for } P \in S_i \in \mathcal{S}$

$g(c) = \{\, S_i \in \mathcal{S} \mid i < c \,\}$

Gradient functions provide cost structure

1. **structured candidate space ($\mathcal{S}$, $\preccurlyeq$)**

2. **cost function (κ)**

3. **gradient function (g)**

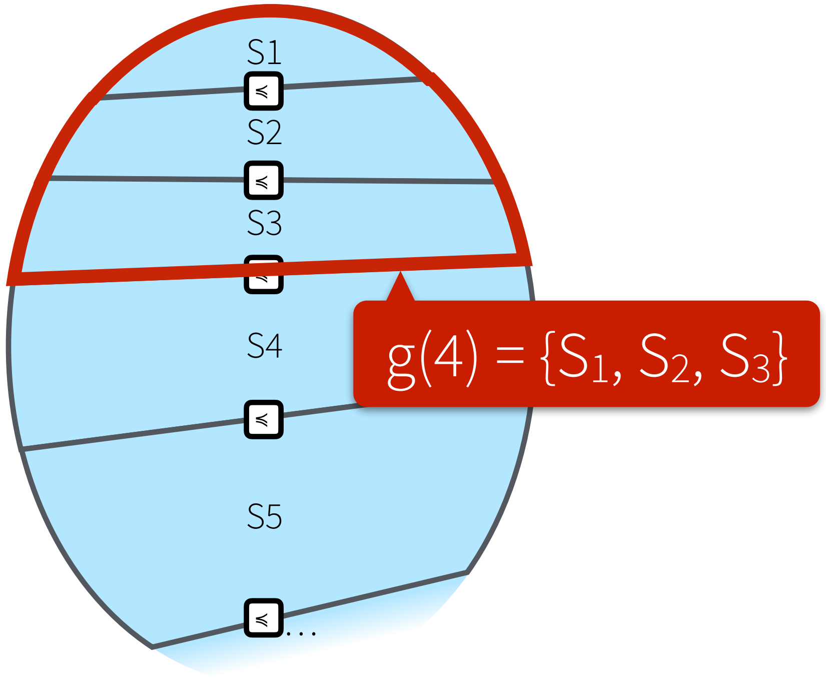
$g : \mathbb{R} \rightarrow 2^{\mathcal{S}}$

g(c) is the set of sketches in $\mathcal{S}$ that *may* contain a solution P with $\kappa(P) < c$

The gradient function overapproximates the behavior of κ on $\mathcal{S}$

Always sound for g to return all of $\mathcal{S}$ if a tighter bound is unavailable.

$\mathcal{S}$ = set of all SSA programs



$\kappa(P) = i \quad \text{for } P \in S_i \in \mathcal{S}$

$g(c) = \{ S_i \in \mathcal{S} \mid i < c \}$

Gradient functions provide cost structure

1. **structured candidate space ($\mathcal{S}$, $\preccurlyeq$)**

2. **cost function (κ)**

3. **gradient function (g)**

$g : \mathbb{R} \rightarrow 2^{\boldsymbol{\mathcal{S}}}$

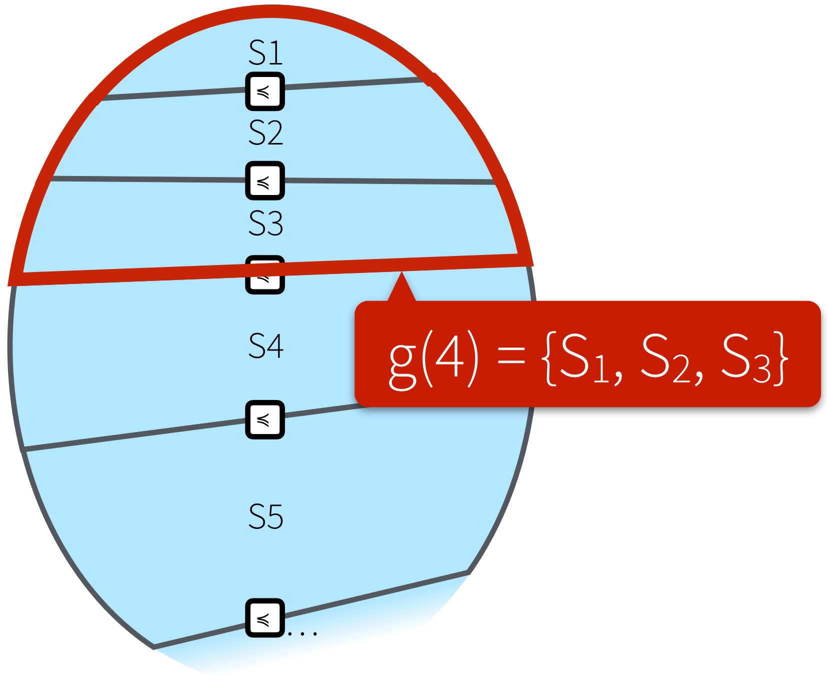
g(c) is the set of sketches in $\boldsymbol{\mathcal{S}}$ that *may* contain a solution P with $\kappa(P) < c$

The gradient function overapproximates the behavior of κ on $\boldsymbol{\mathcal{S}}$

Always sound for g to return all of $\boldsymbol{\mathcal{S}}$ if a tighter bound is unavailable.

g(c) always being finite is sufficient (not necessary) to guarantee termination.

$\boldsymbol{\mathcal{S}}$ = set of all SSA programs



$\kappa(P) = i \quad \text{for } P \in S_i \in \boldsymbol{\mathcal{S}}$

$g(c) = \{\, S_i \in \boldsymbol{\mathcal{S}} \mid i < c \,\}$

Metasketches

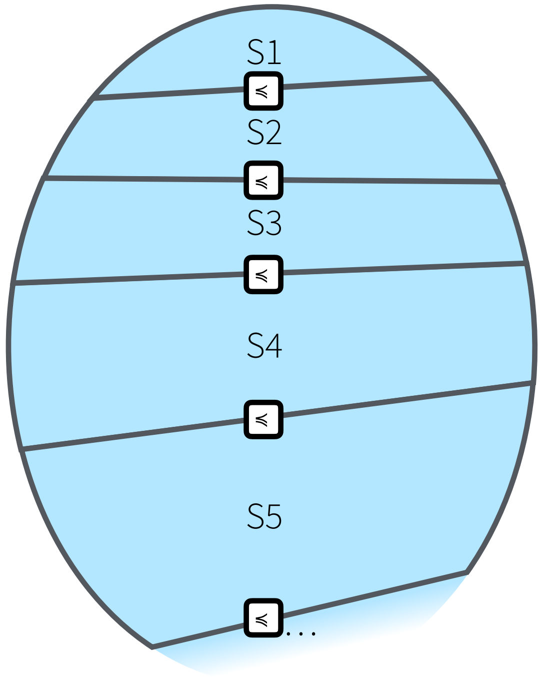
Design and structure

1. **structured candidate space ($\mathcal{S}$, ≼)**

2. **cost function (κ)**

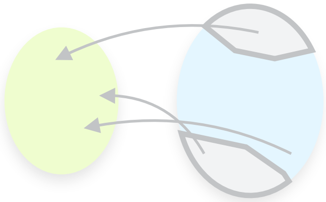
3. **gradient function (g)**

$\boldsymbol{\mathcal{S}}$ = set of all SSA programs



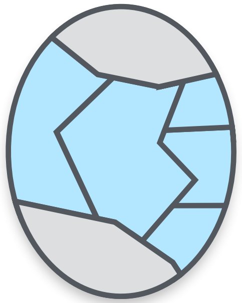
$\kappa(P) = i \quad \text{for } P \in S_i \in \boldsymbol{\mathcal{S}}$

$g(c) = \{\, S_i \in \boldsymbol{\mathcal{S}} \mid i < c \,\}$



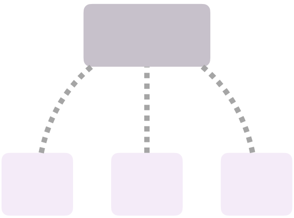
Background

Syntax-guided synthesis



Metasketches

Design and structure

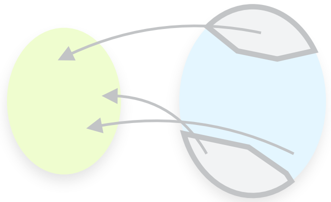


Synapse

A metasketch solver

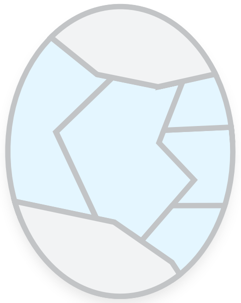
Results

Better solutions, faster



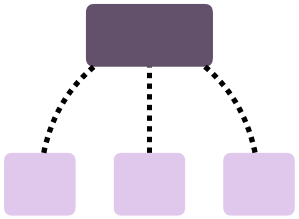
Background

Syntax-guided synthesis



Metasketches

Design and structure



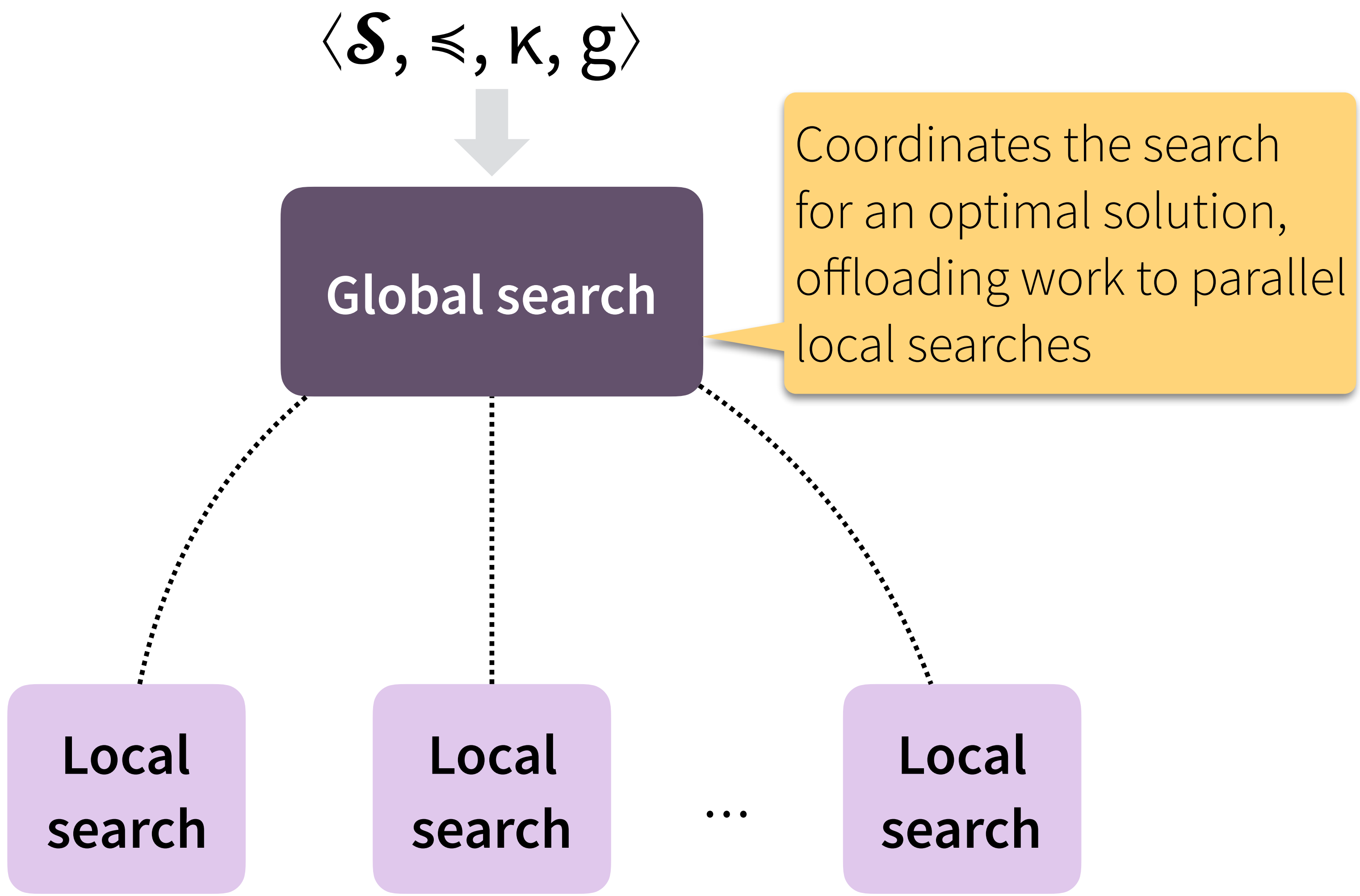
Synapse

A metasketch solver

Results

Better solutions, faster

Solving with two cooperative searches



$\langle \mathcal{S}, \preccurlyeq, \kappa, g \rangle$

Global search

Local search

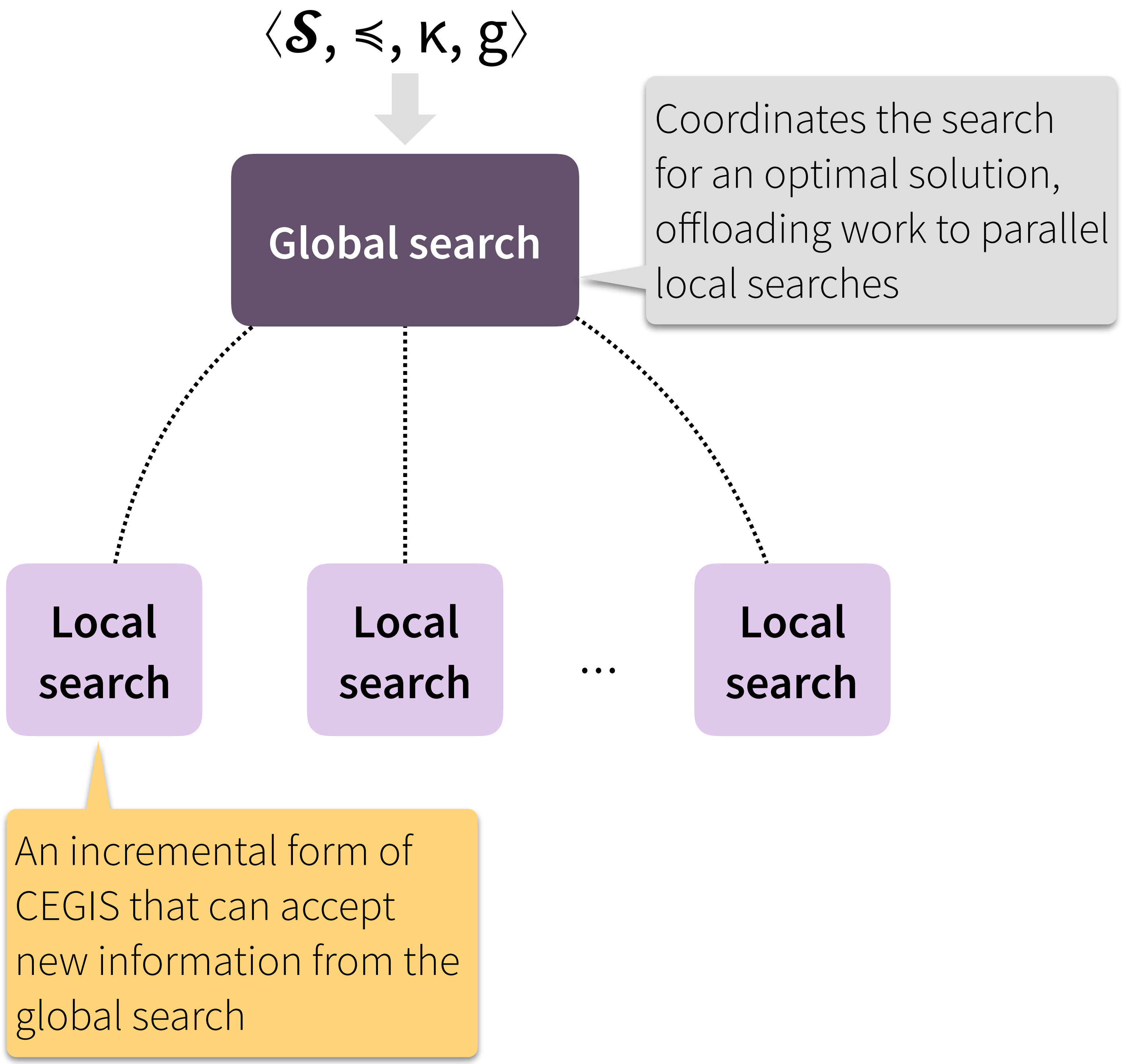
Local search

…

Local search

Coordinates the search for an optimal solution, offloading work to parallel local searches

Solving with two cooperative searches



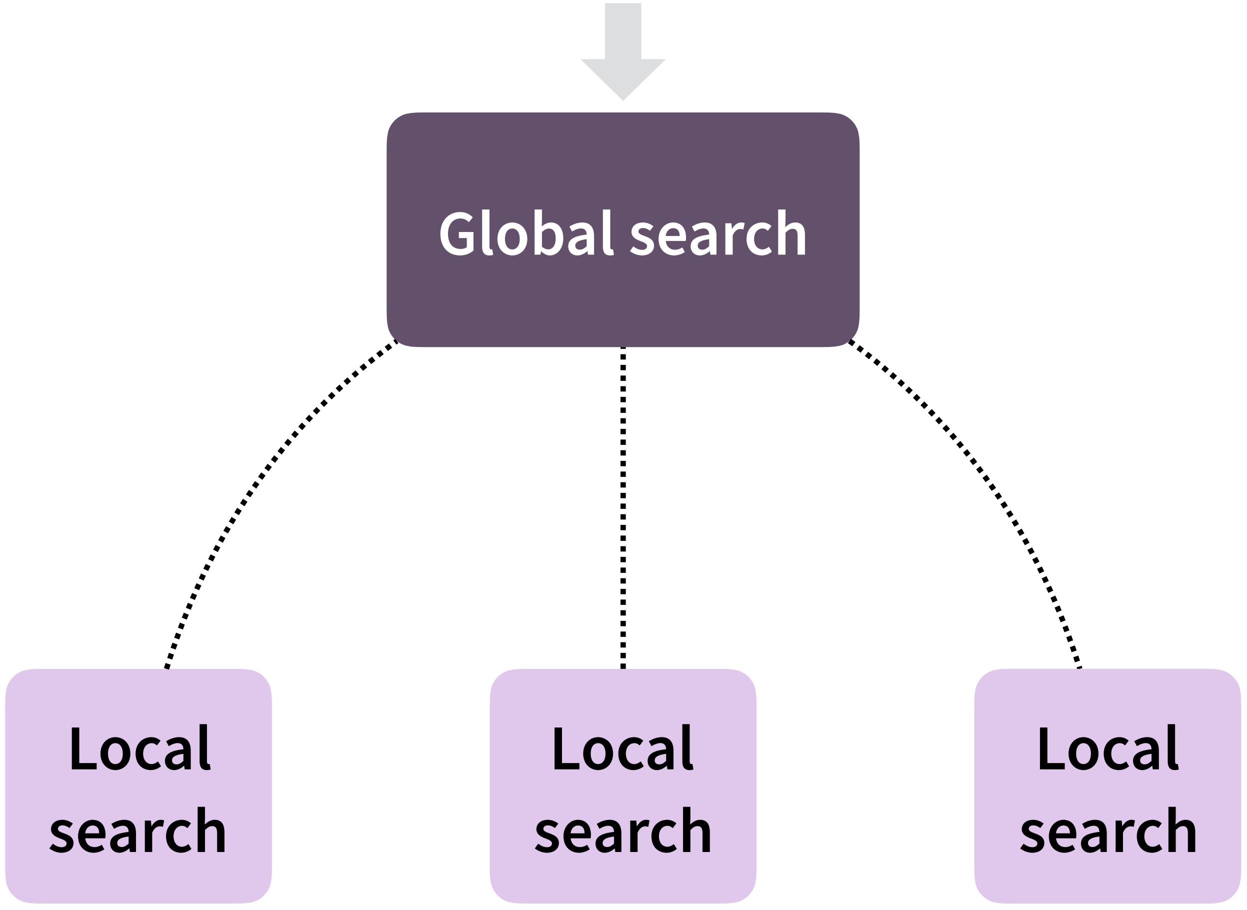
$\langle \mathcal{S}, \preccurlyeq, \kappa, g \rangle$

Coordinates the search for an optimal solution, offloading work to parallel local searches

An incremental form of CEGIS that can accept new information from the global search

Solving with two cooperative searches

$\langle \boldsymbol{\mathcal{S}}, \preccurlyeq, \kappa, g \rangle$

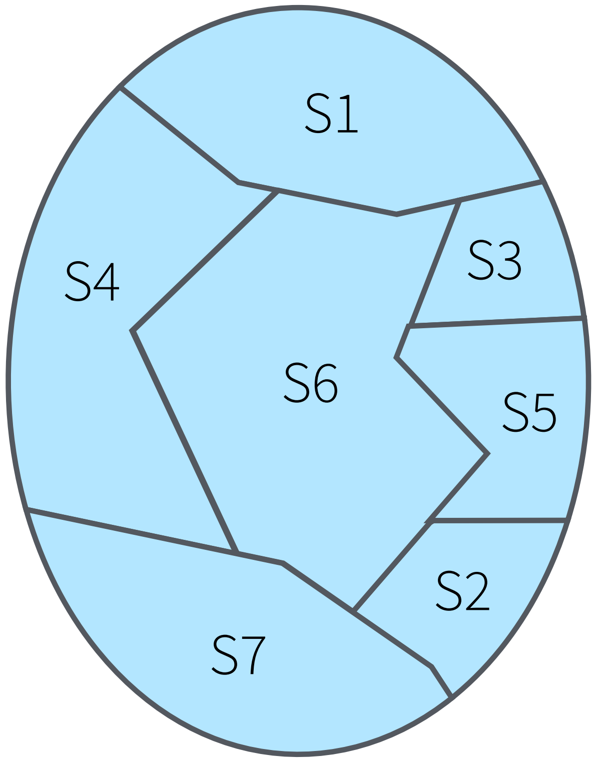


Global search

Local search

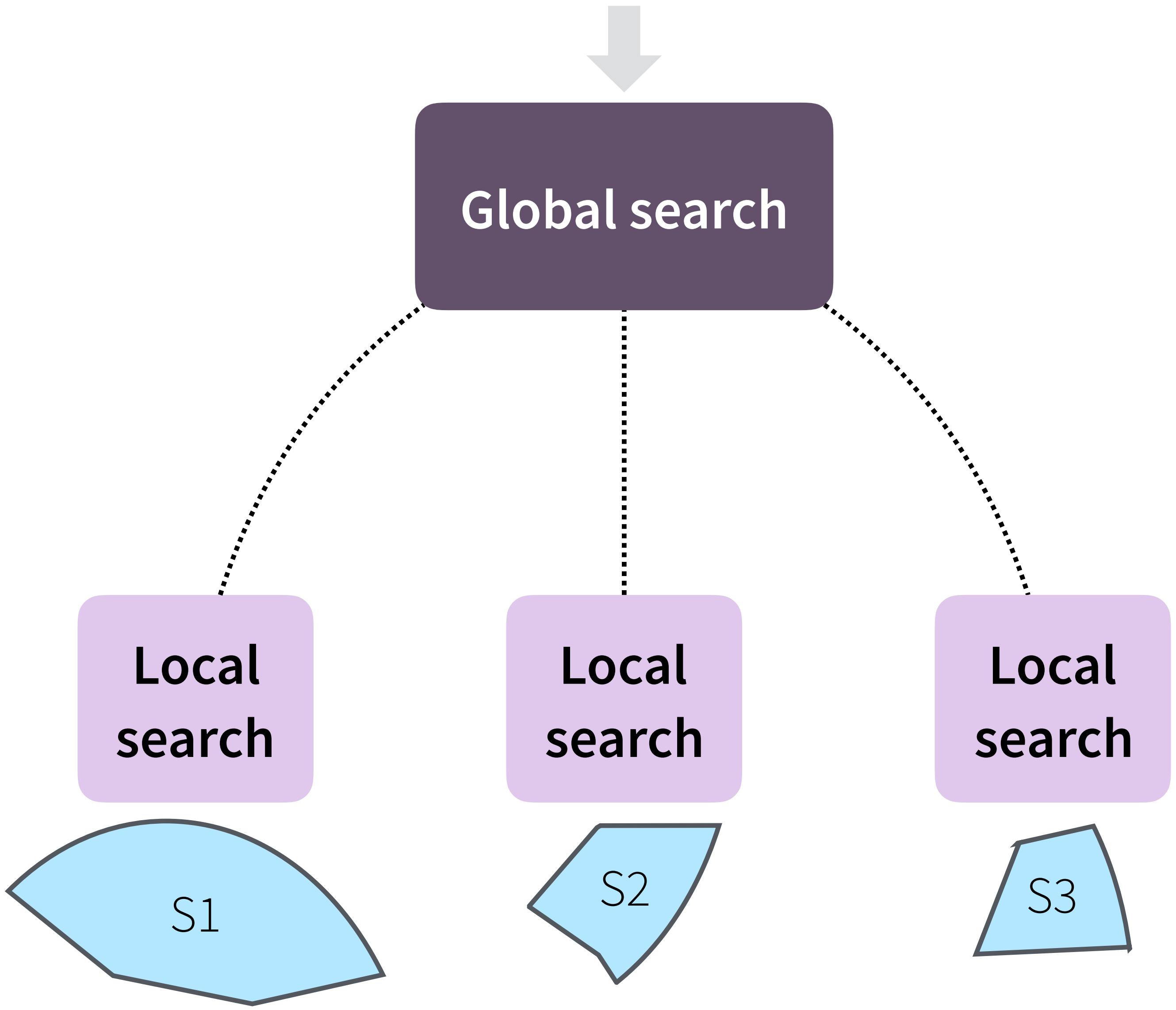
Local search

Local search



Solving with two cooperative searches

$\langle \mathcal{S}, \preccurlyeq, \kappa, g \rangle$

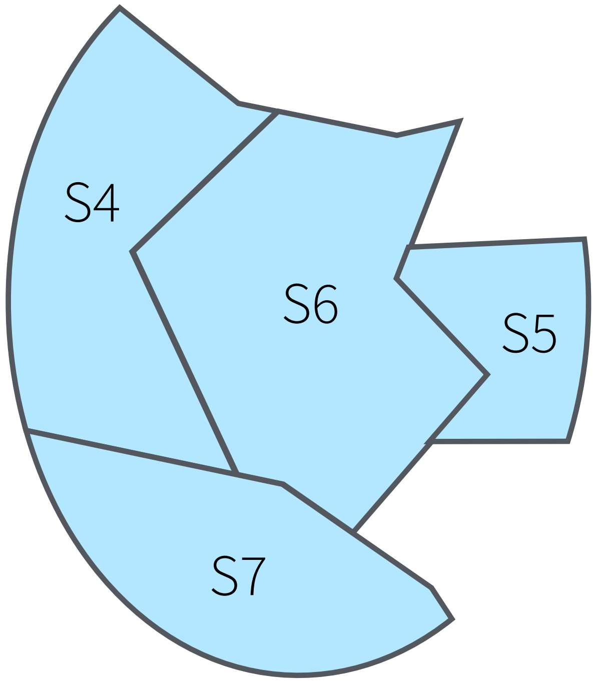


Global search

Local search

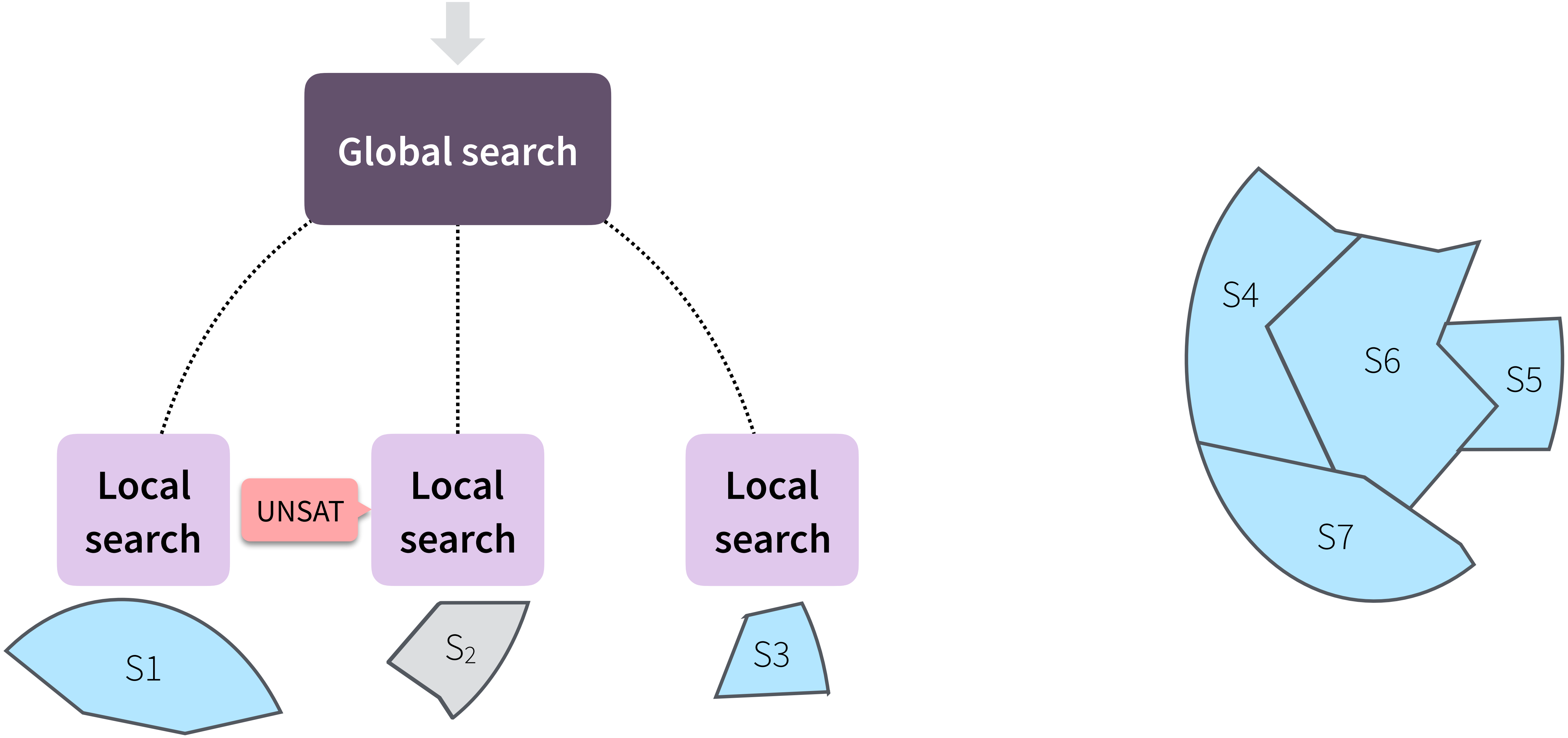
Local search

Local search

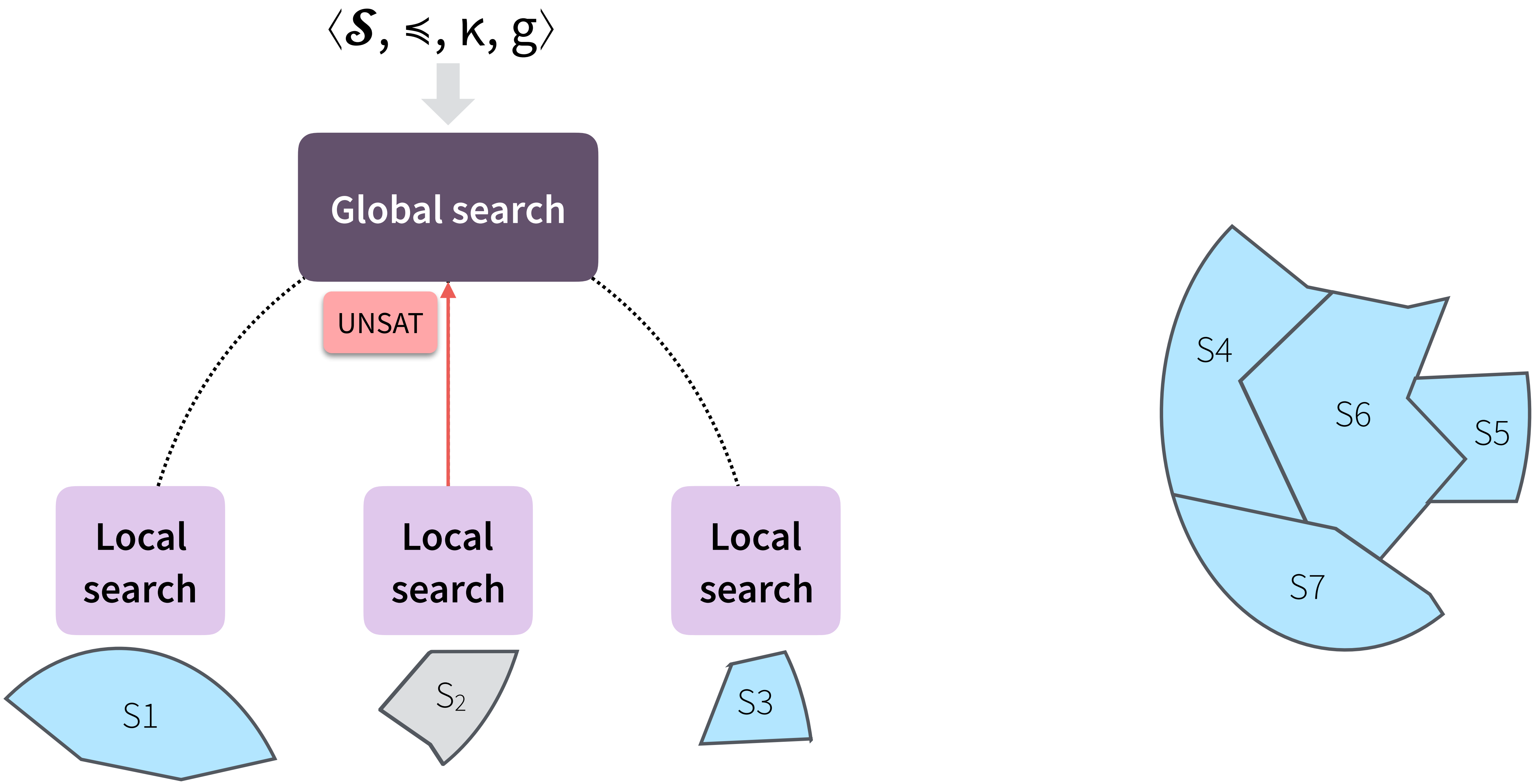


Solving with two cooperative searches

$\langle \mathcal{S}, \preccurlyeq, \kappa, g \rangle$

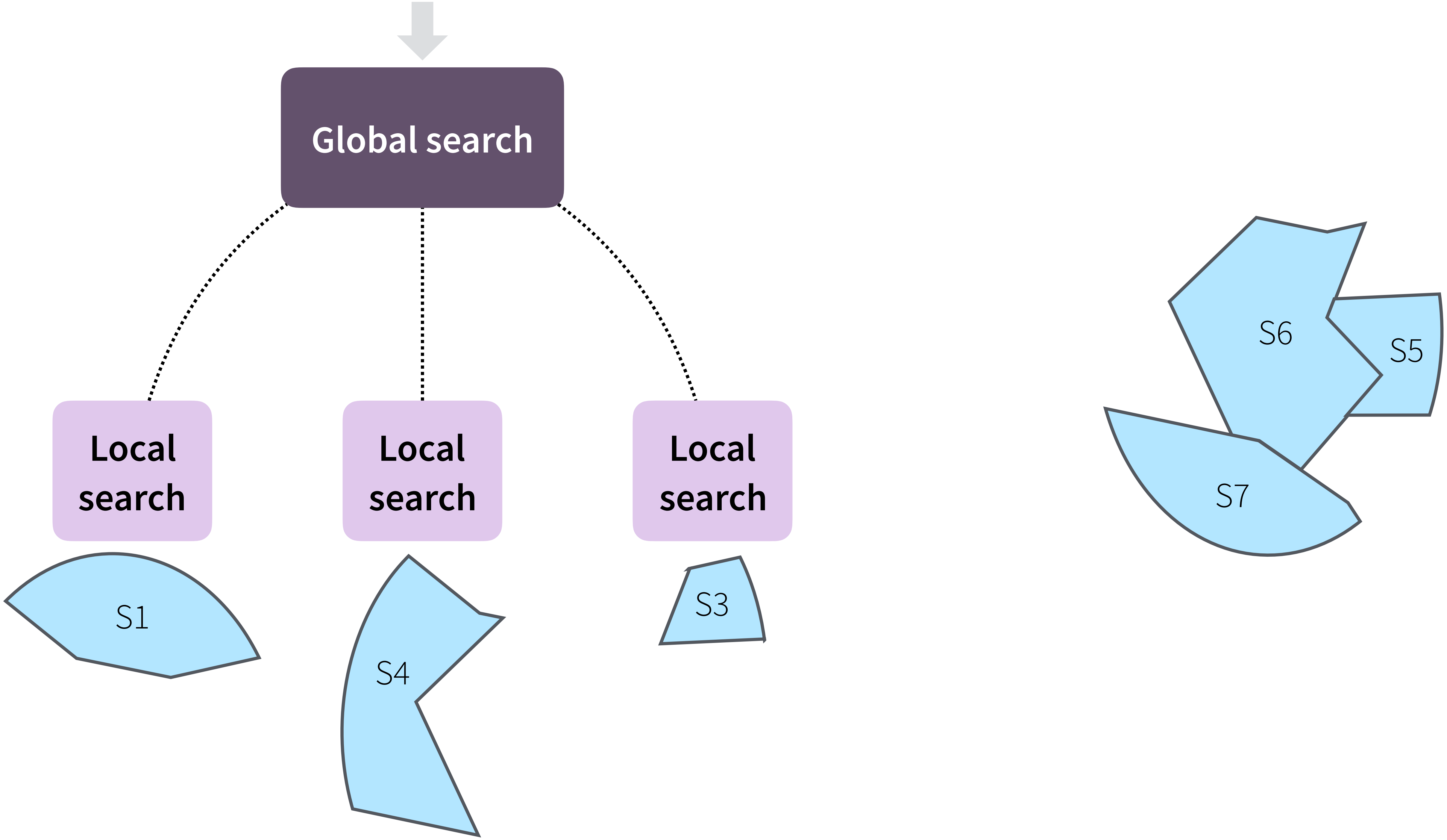


Solving with two cooperative searches

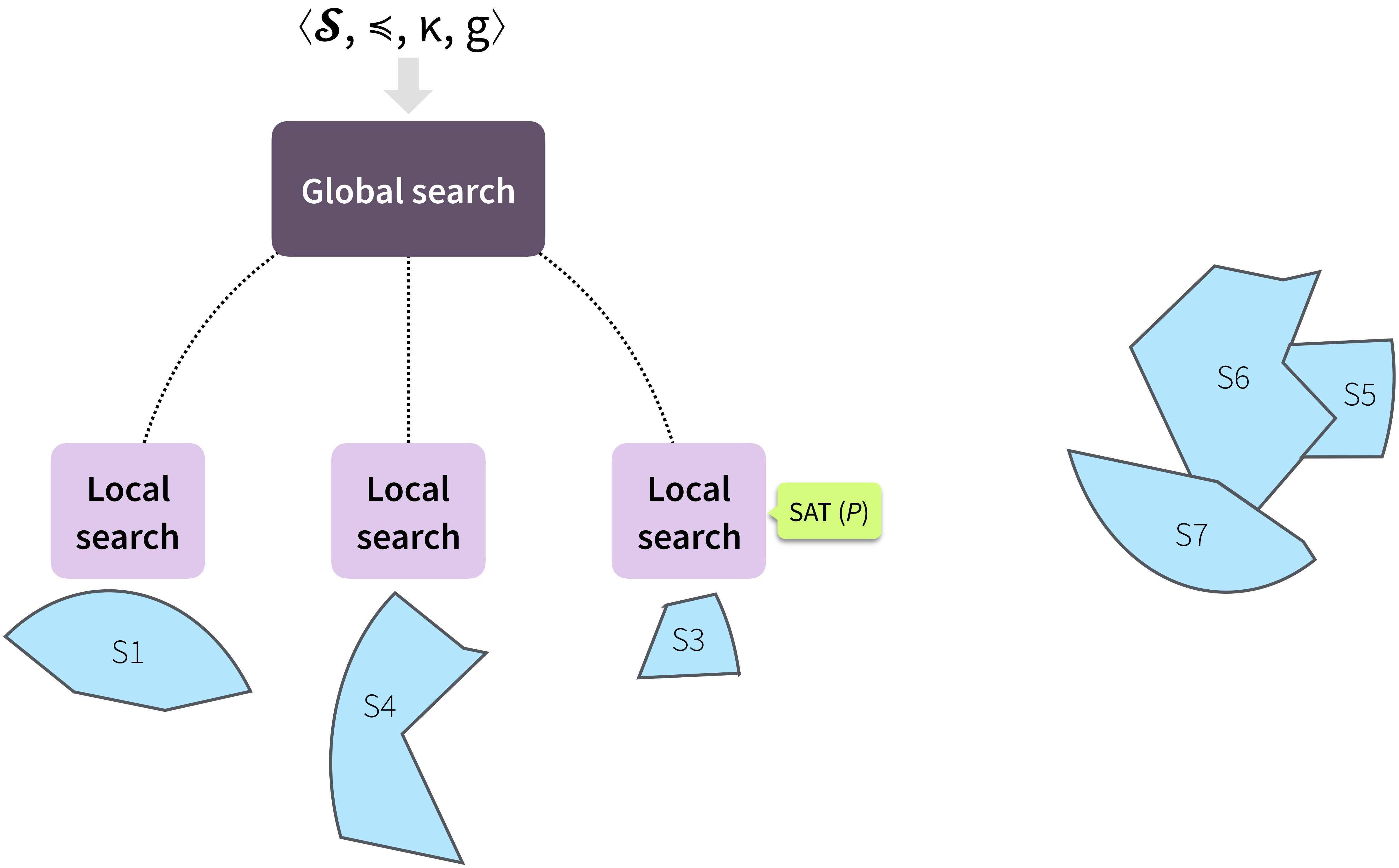


Solving with two cooperative searches

$\langle \boldsymbol{\mathcal{S}}, \preccurlyeq, \kappa, g \rangle$



Solving with two cooperative searches



$\langle \mathcal{S}, \preccurlyeq, \kappa, g \rangle$

Global search

Local search

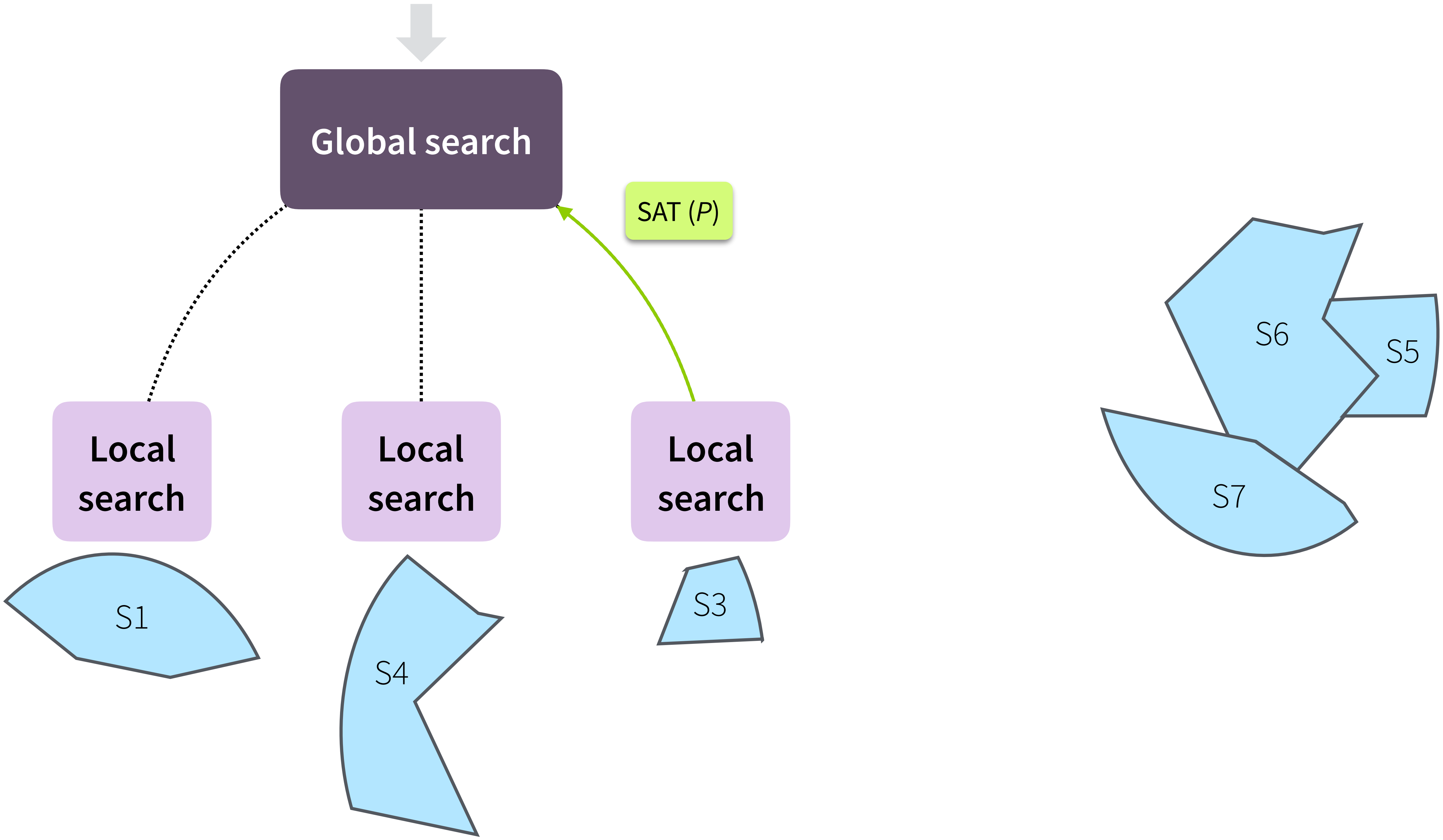
Local search

Local search

SAT (*P*)

Solving with two cooperative searches

$\langle \mathcal{S}, \preccurlyeq, \kappa, g \rangle$



Global search

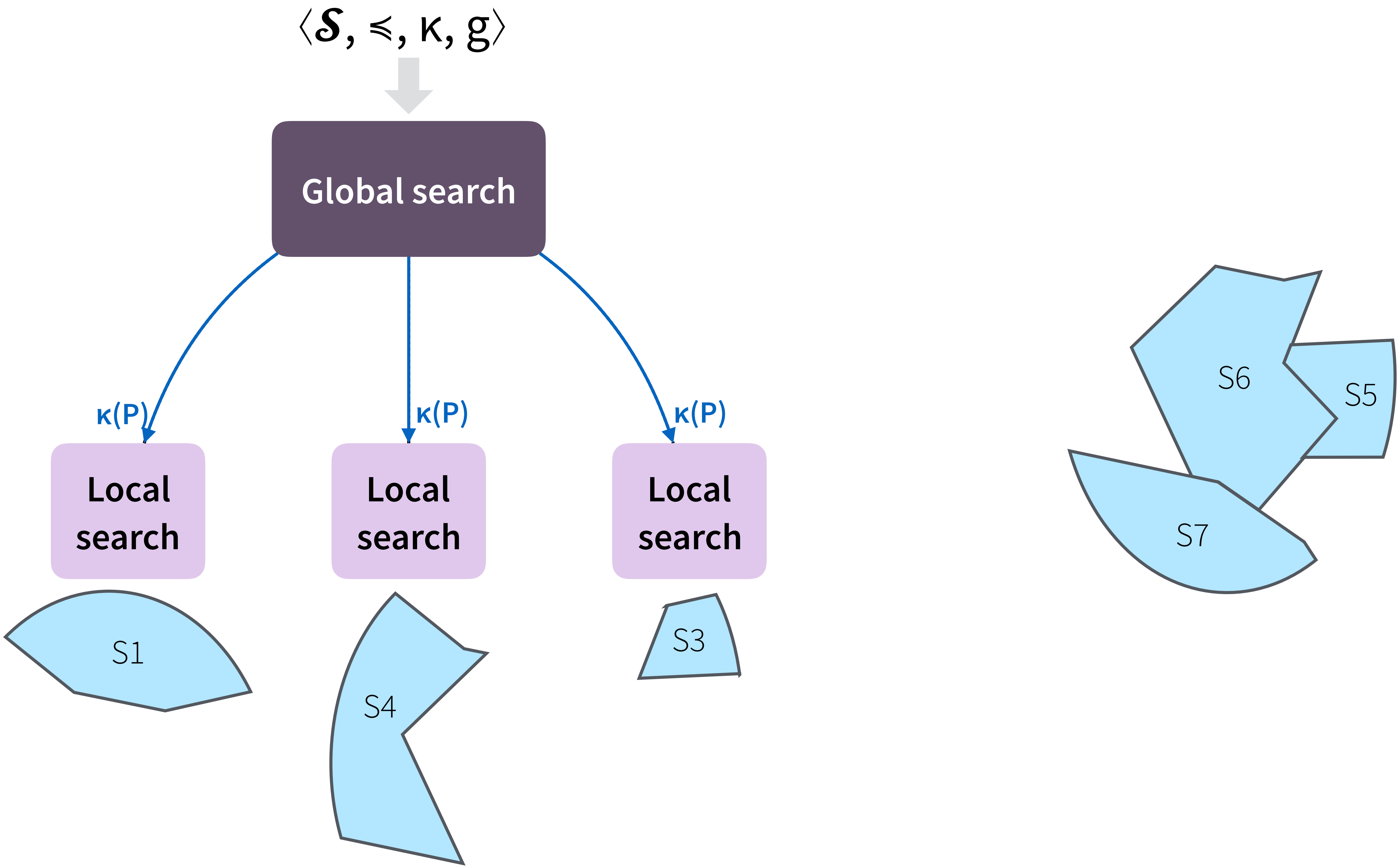
SAT (*P*)

Local search

Local search

Local search

Solving with two cooperative searches



$\langle \boldsymbol{\mathcal{S}}, \preccurlyeq, \kappa, g \rangle$

Global search

$\kappa(P)$

$\kappa(P)$

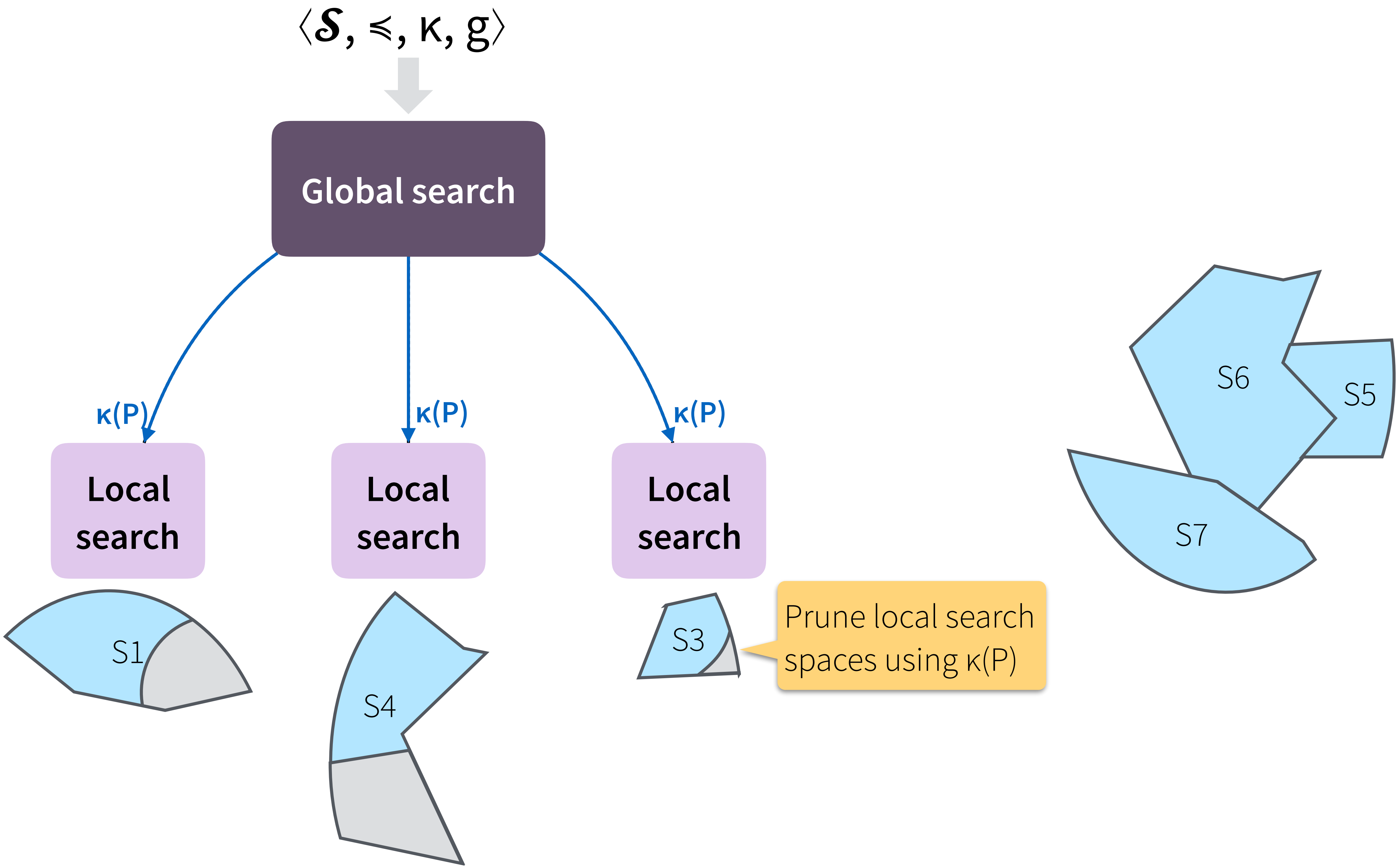
$\kappa(P)$

Local search

Local search

Local search

Solving with two cooperative searches



$\langle \mathcal{S}, \preccurlyeq, \kappa, g \rangle$

Global search

κ(P)

κ(P)

κ(P)

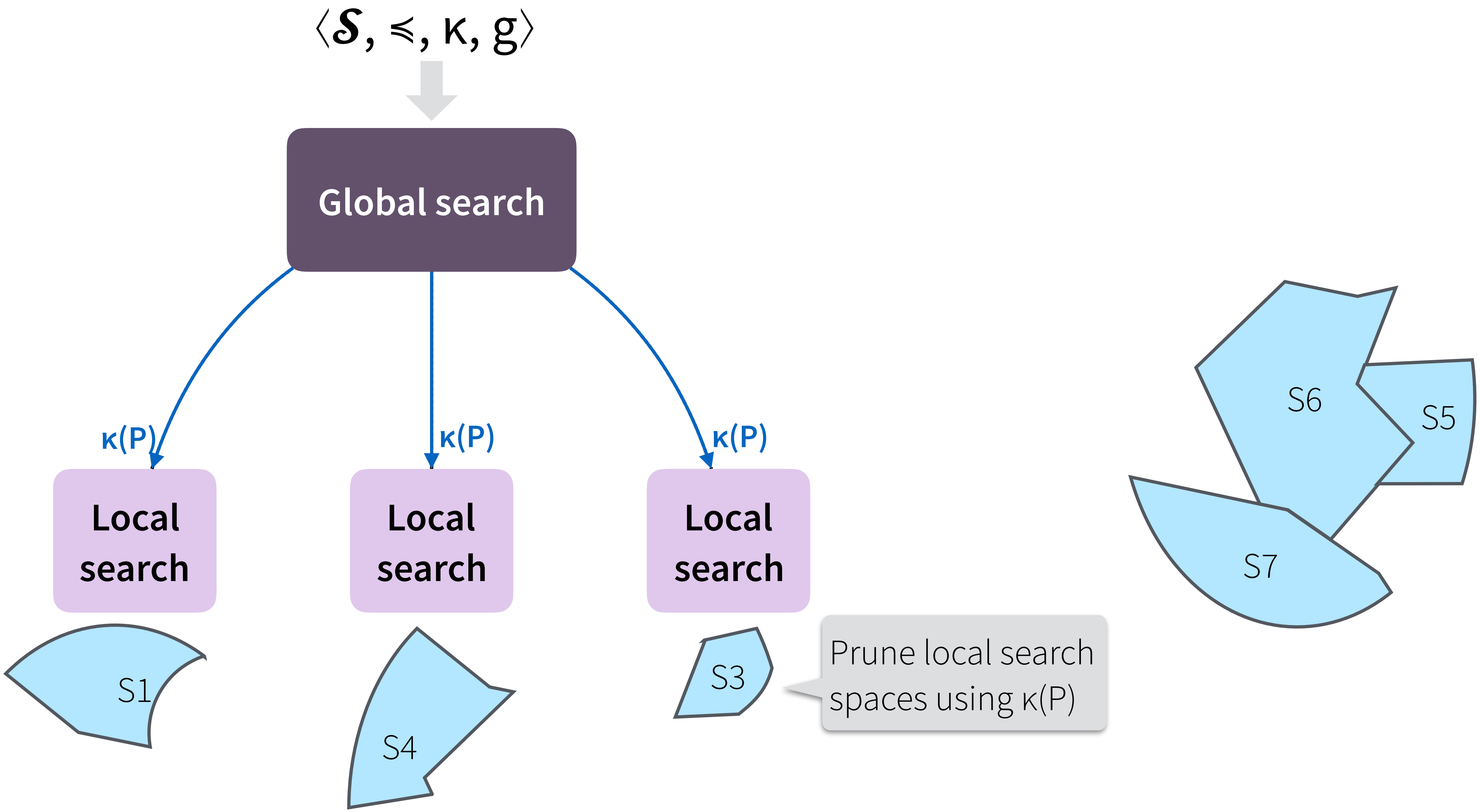
Local search

Local search

Local search

Prune local search spaces using κ(P)

Solving with two cooperative searches



$\langle \mathcal{S}, \preccurlyeq, \kappa, g \rangle$

Global search

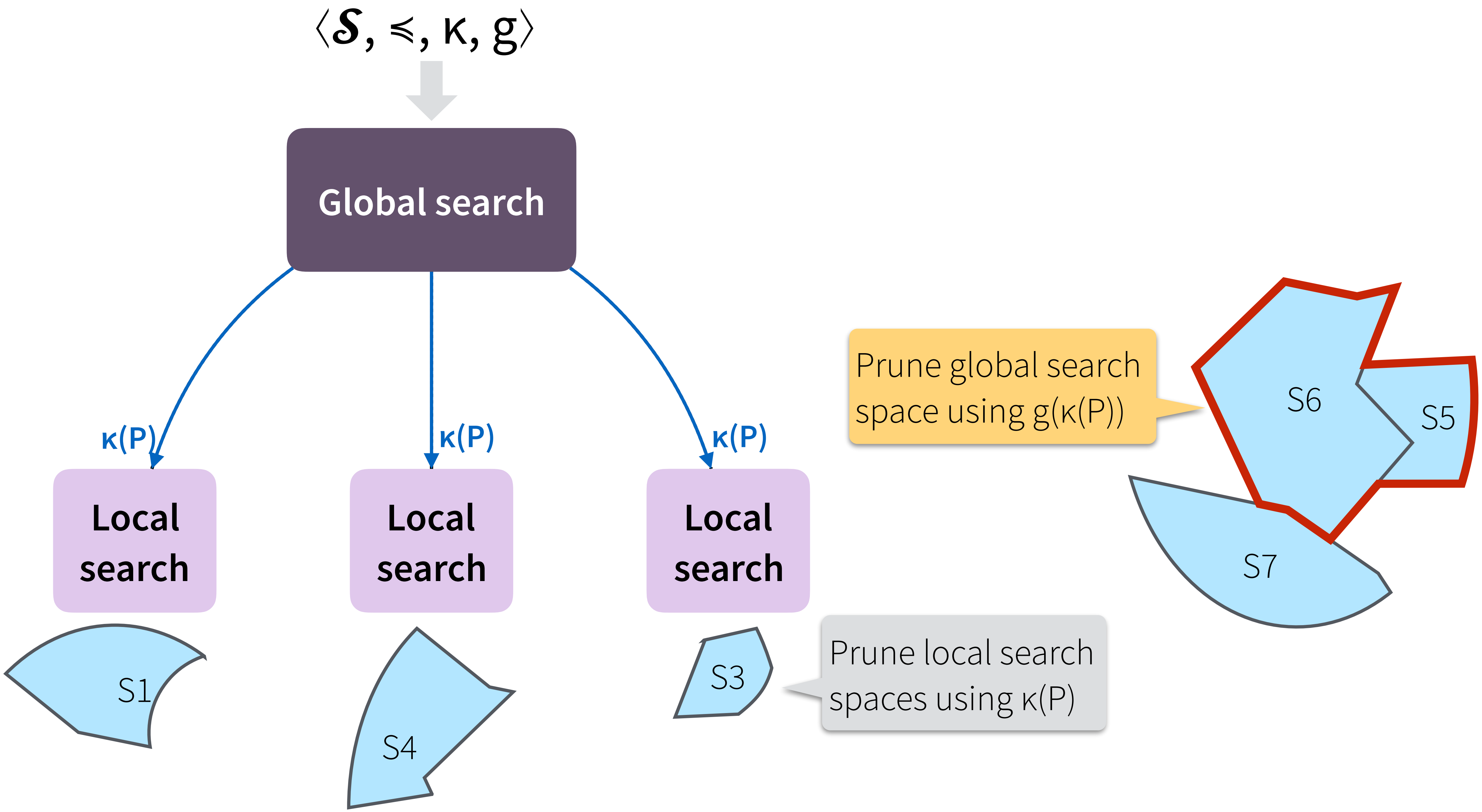
Local search

Local search

Local search

Prune local search spaces using κ(P)

Solving with two cooperative searches

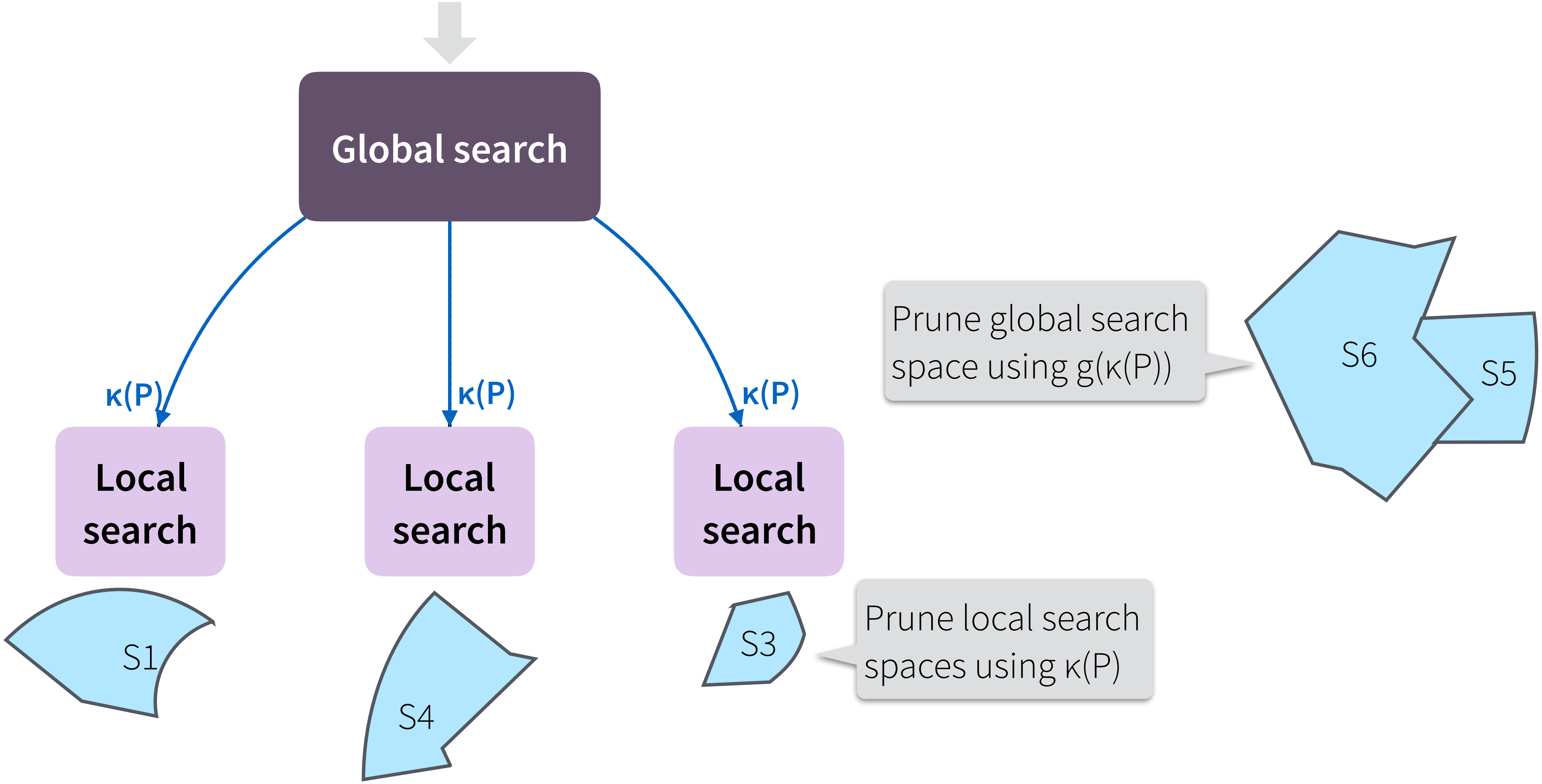


Prune global search space using g(κ(P))

Prune local search spaces using κ(P)

Solving with two cooperative searches

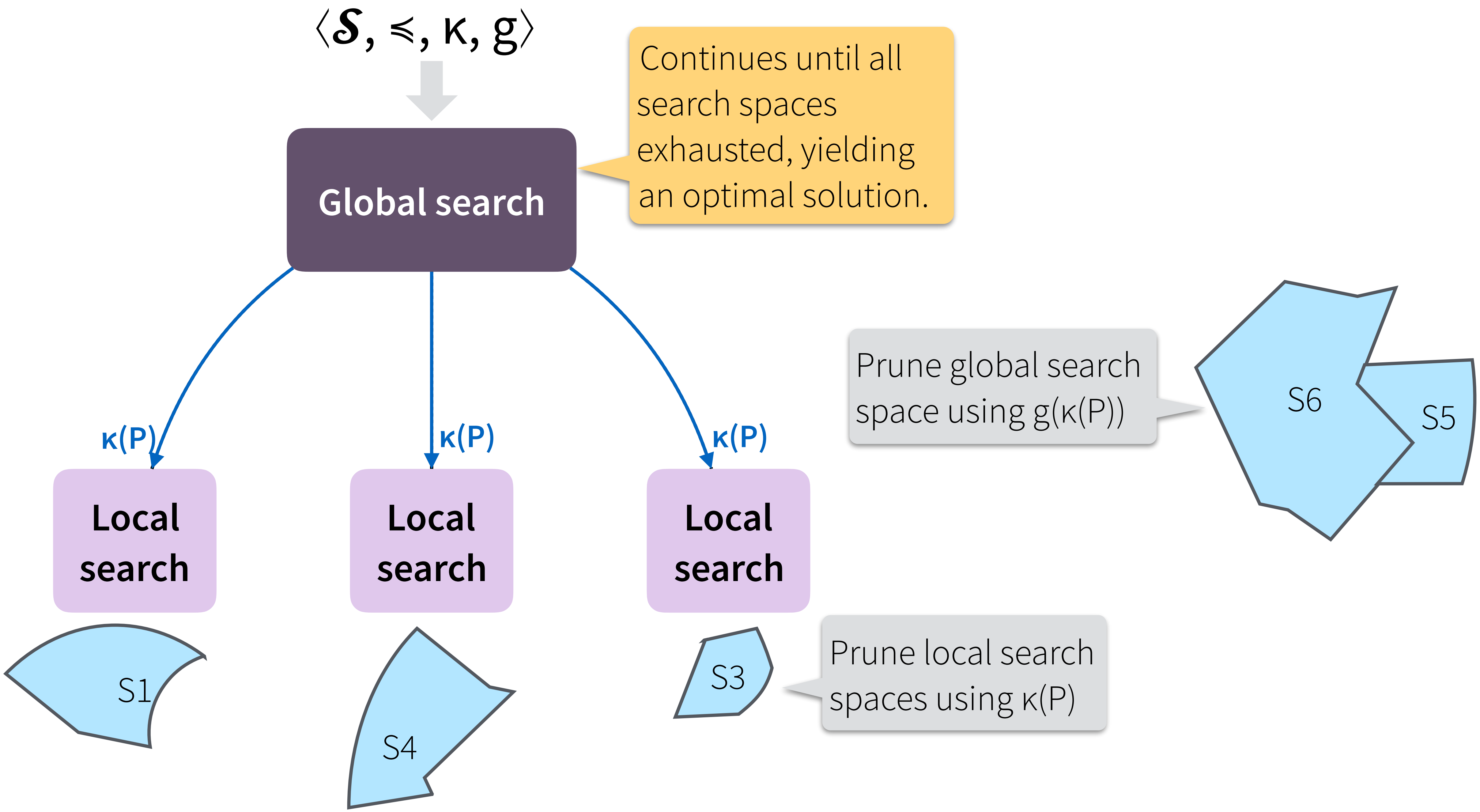
$\langle \mathcal{S}, \preccurlyeq, \kappa, g \rangle$



Prune global search space using g(κ(P))

Prune local search spaces using κ(P)

Solving with two cooperative searches



$\langle \mathcal{S}, \preccurlyeq, \kappa, g \rangle$

Global search

Continues until all search spaces exhausted, yielding an optimal solution.

κ(P)

Local search

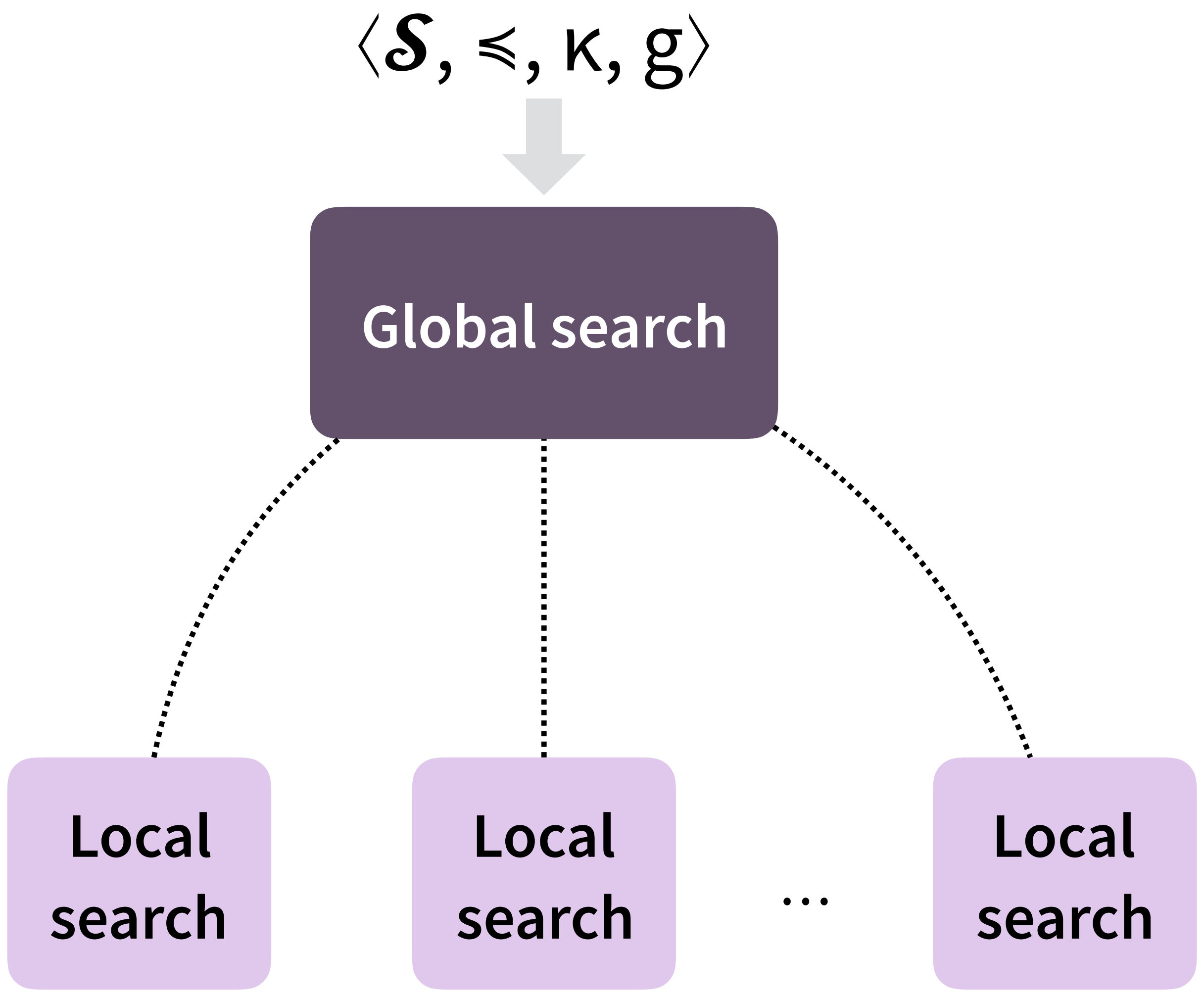
Local search

Local search

Prune global search space using g(κ(P))

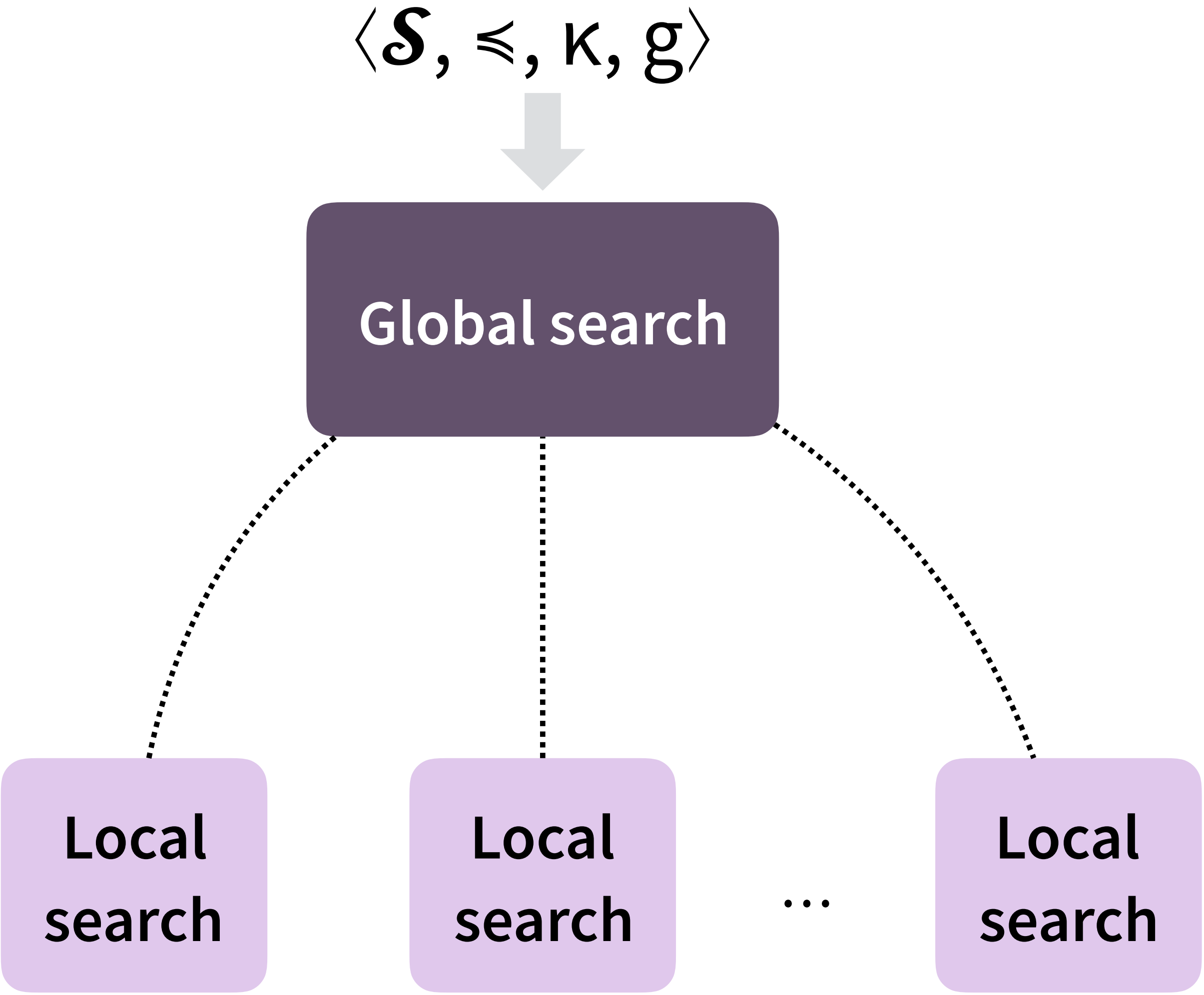
Prune local search spaces using κ(P)

Synapse implementation



Implemented in Rosette, a solver-aided extension of Racket

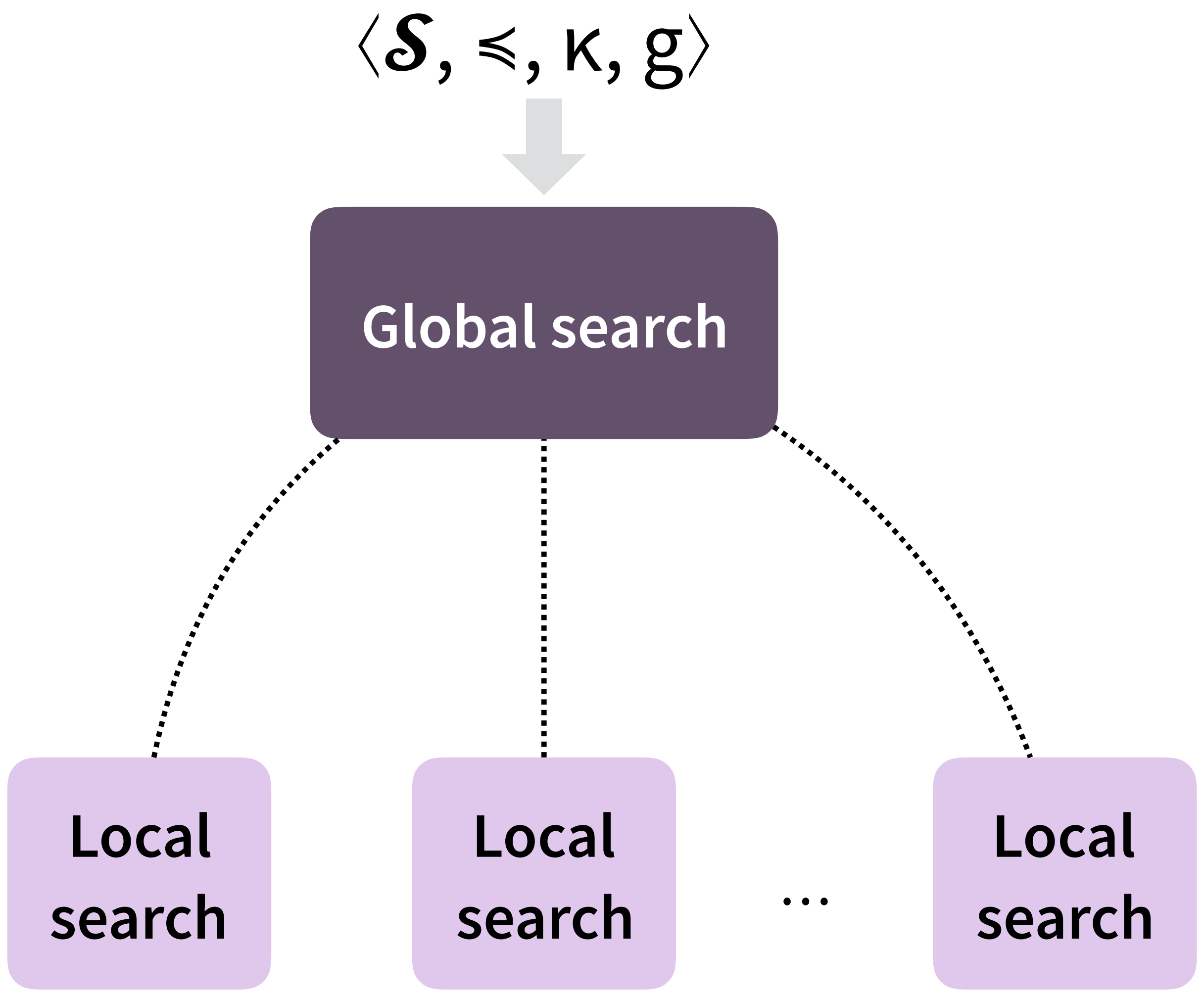
Synapse implementation



Implemented in Rosette, a solver-aided extension of Racket

Local CEGIS searches can share counterexamples

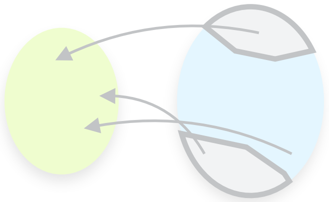
Synapse implementation



Implemented in Rosette, a solver-aided extension of Racket

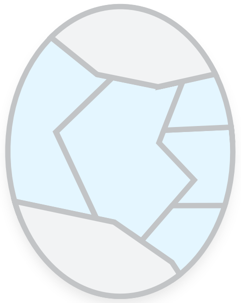
Local CEGIS searches can share counterexamples

Local searches can time out, which weakens optimality



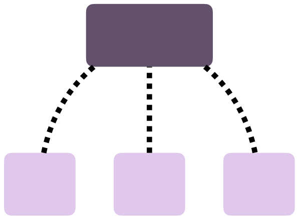
Background

Syntax-guided synthesis



Metasketches

Design and structure

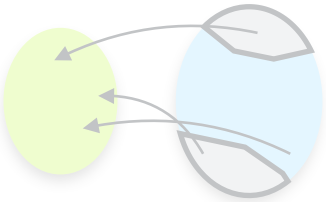


Synapse

A metasketch solver

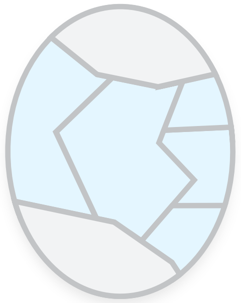
Results

Better solutions, faster



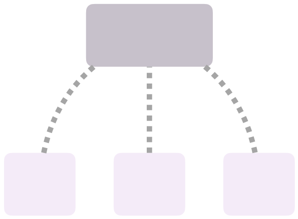
Background

Syntax-guided synthesis



Metasketches

Design and structure



Synapse

A metasketch solver

Results

Better solutions, faster

Evaluation questions

Is Synapse a practical approach to solving different kinds of synthesis problems?

- Approximate computing, array programs

Evaluation questions

Is Synapse a practical approach to solving different kinds of synthesis problems?

Approximate computing, array programs

Can Synapse reason about complex cost functions?

Evaluation questions

Is Synapse a practical approach to solving different kinds of synthesis problems?

Approximate computing, array programs

Can Synapse reason about complex cost functions?

In the paper:

- Parallel speedup
- Optimizations (structure constraints, sharing)
- More kinds of problems
- More complex cost functions

Synapse solves previously-intractable problems

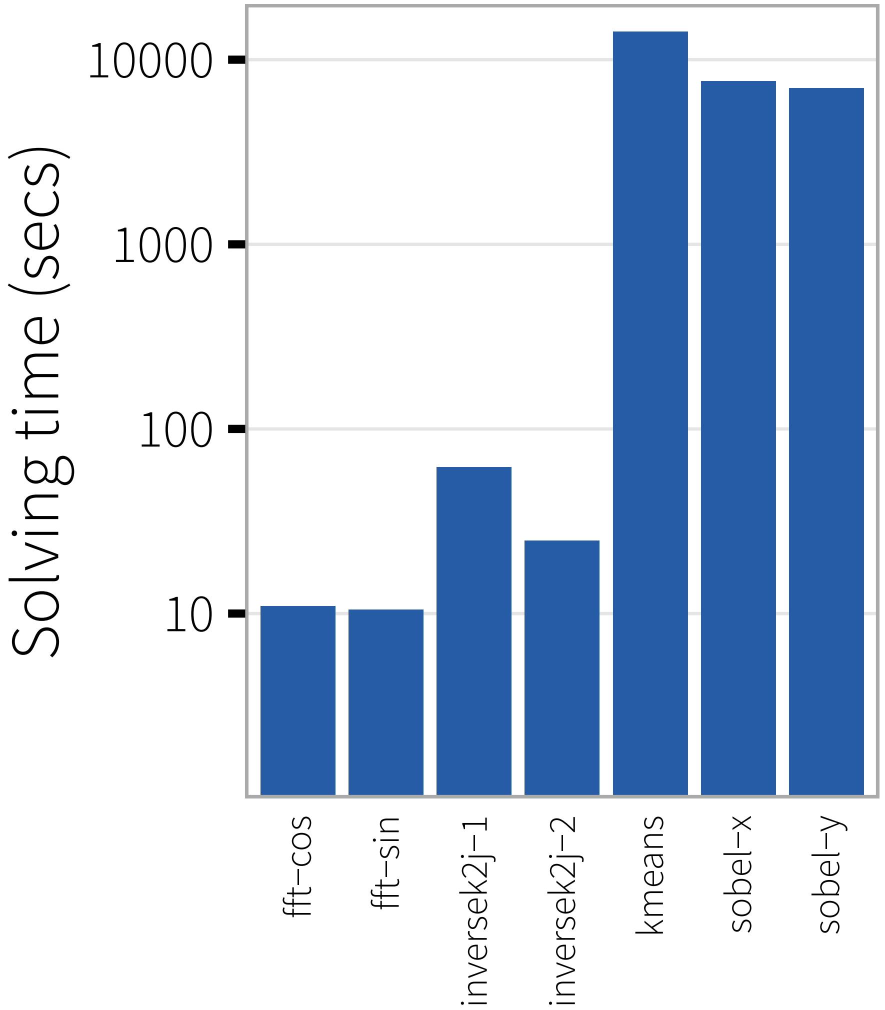
Parrot benchmarks from approximate computing [Esmaelizadeh et al., 2012]

Find the most efficient approximate program within an error bound

Synapse solves previously-intractable problems

Parrot benchmarks from approximate computing [Esmaelizadeh et al., 2012]

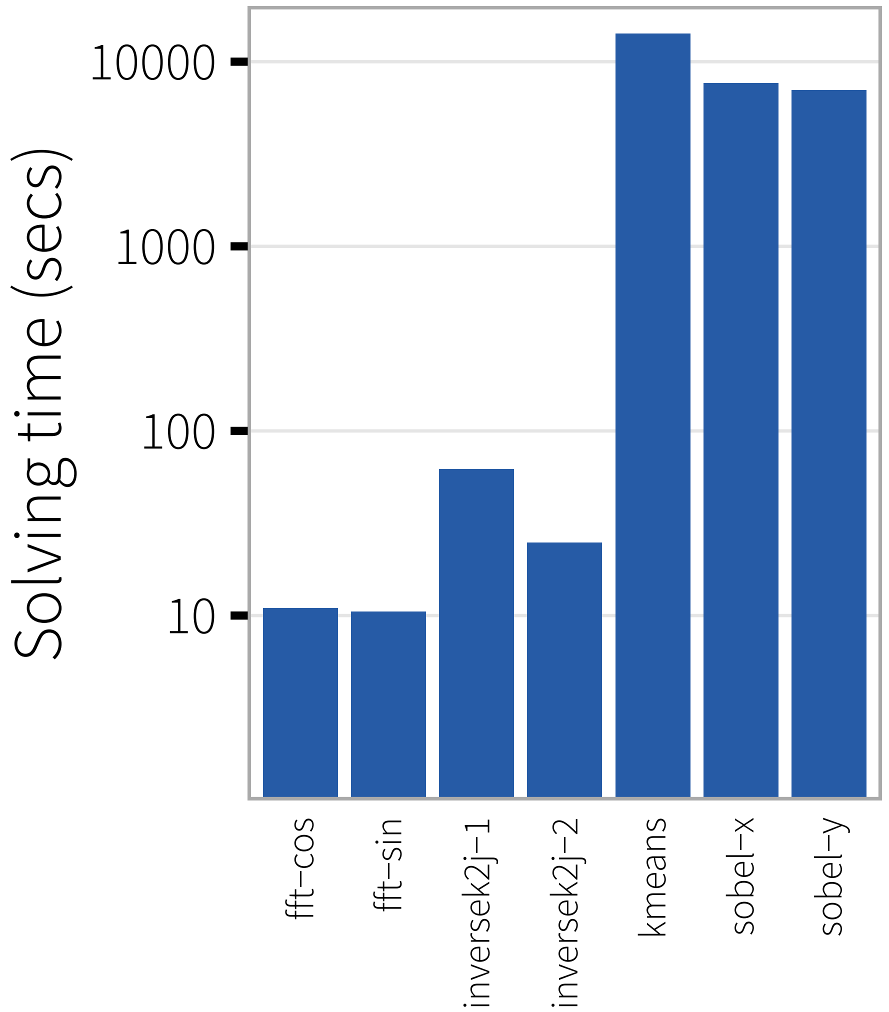
Find the most efficient approximate program within an error bound



Synapse solves previously-intractable problems

Parrot benchmarks from approximate computing [Esmaelizadeh et al., 2012]

Find the most efficient approximate program within an error bound



All intractable to Sketch and Stoke

Synapse solves standard benchmarks optimally

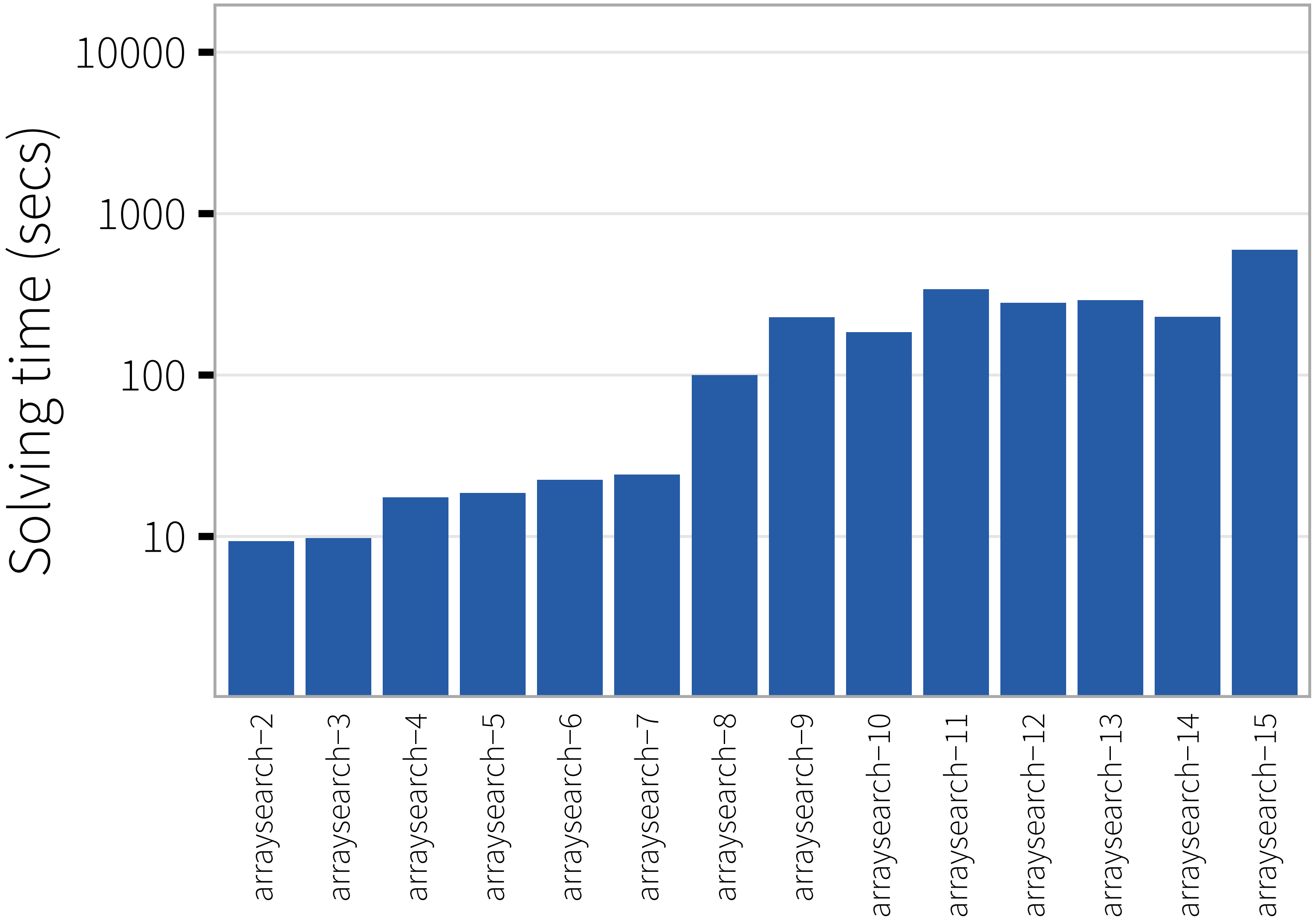
Array Search benchmarks from the syntax-guided synthesis (SyGuS) competition [Alur et al., 2015]

arraysearch-*n*: find program that searches lists of length *n*

Synapse solves standard benchmarks optimally

Array Search benchmarks from the syntax-guided synthesis (SyGuS) competition [Alur et al., 2015]

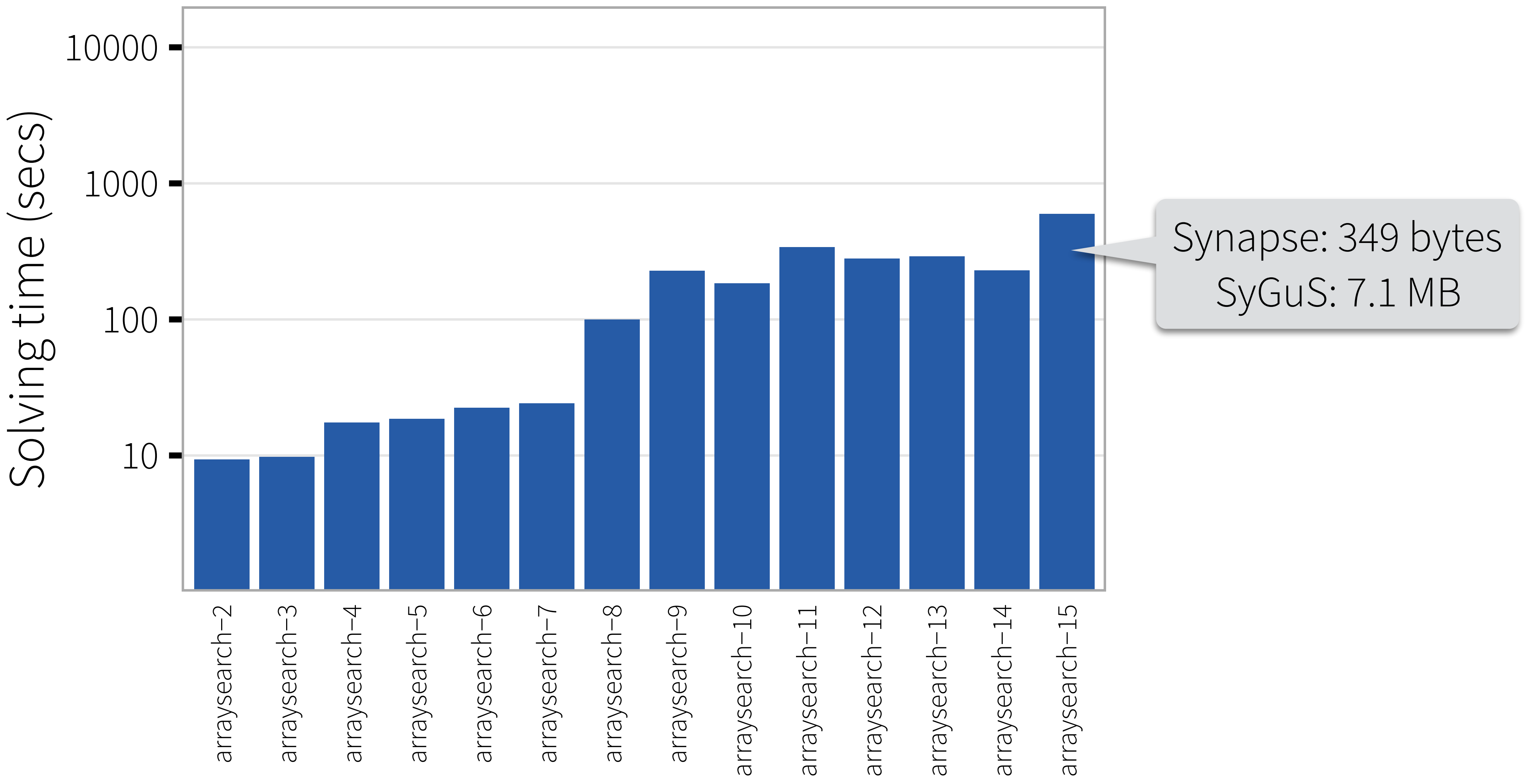
arraysearch-*n*: find program that searches lists of length *n*



Synapse solves standard benchmarks optimally

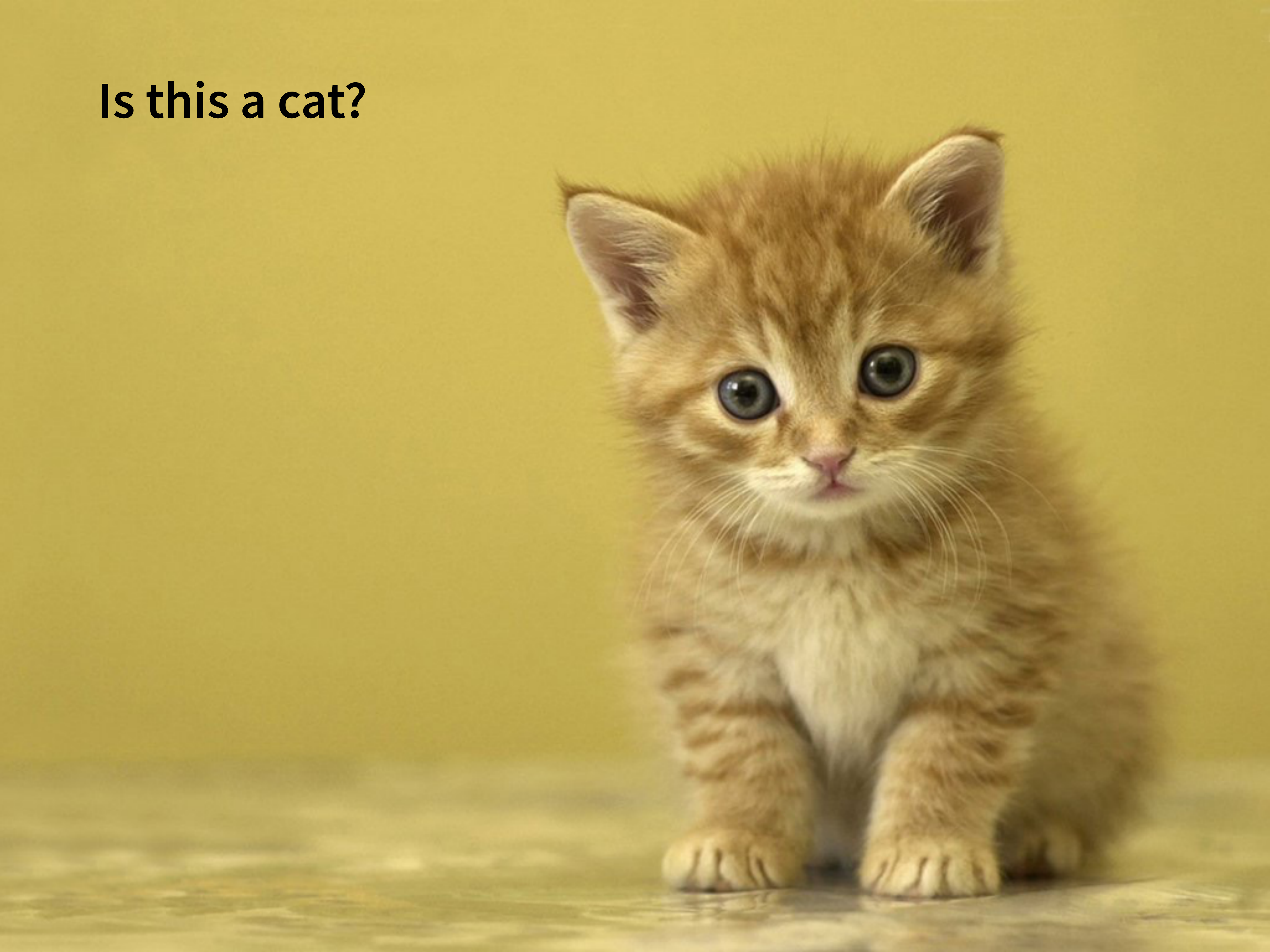
Array Search benchmarks from the syntax-guided synthesis (SyGuS) competition [Alur et al., 2015]

arraysearch-*n*: find program that searches lists of length *n*

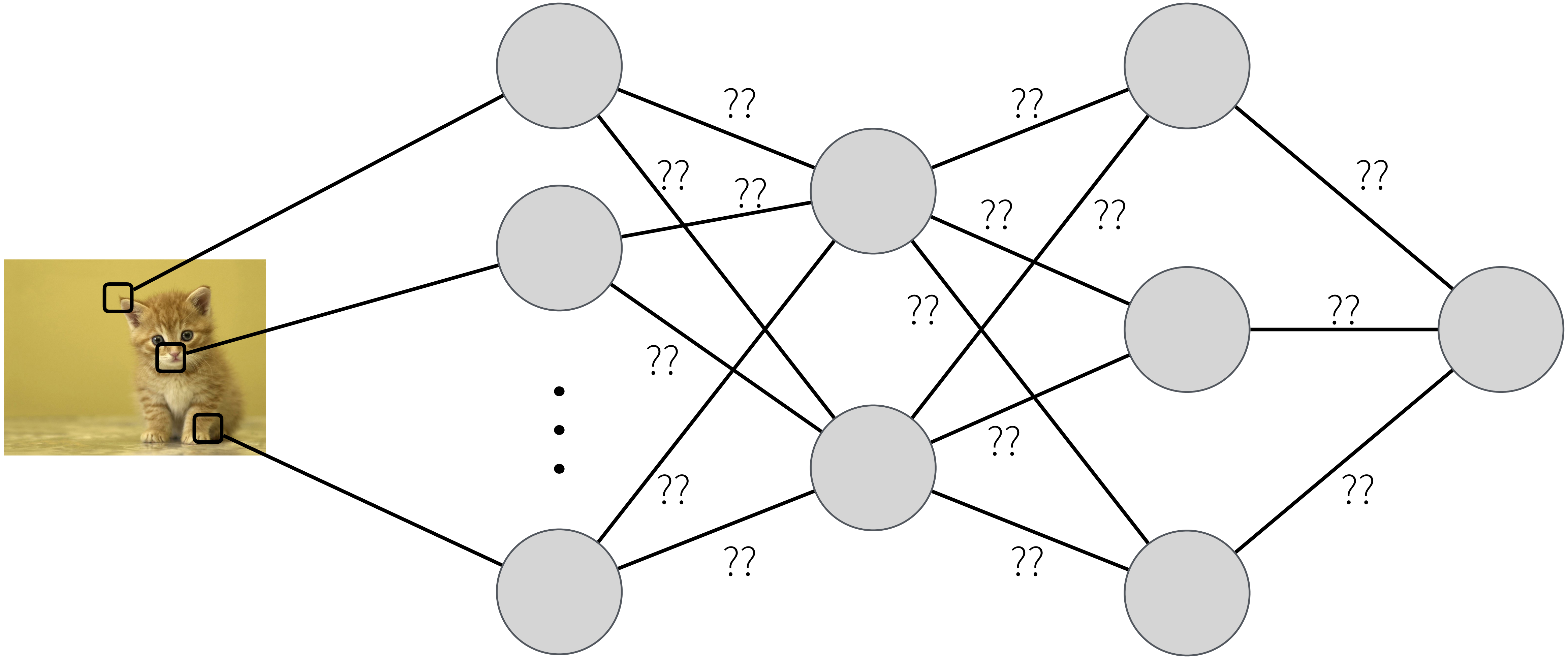


Synapse: 349 bytes
SyGuS: 7.1 MB

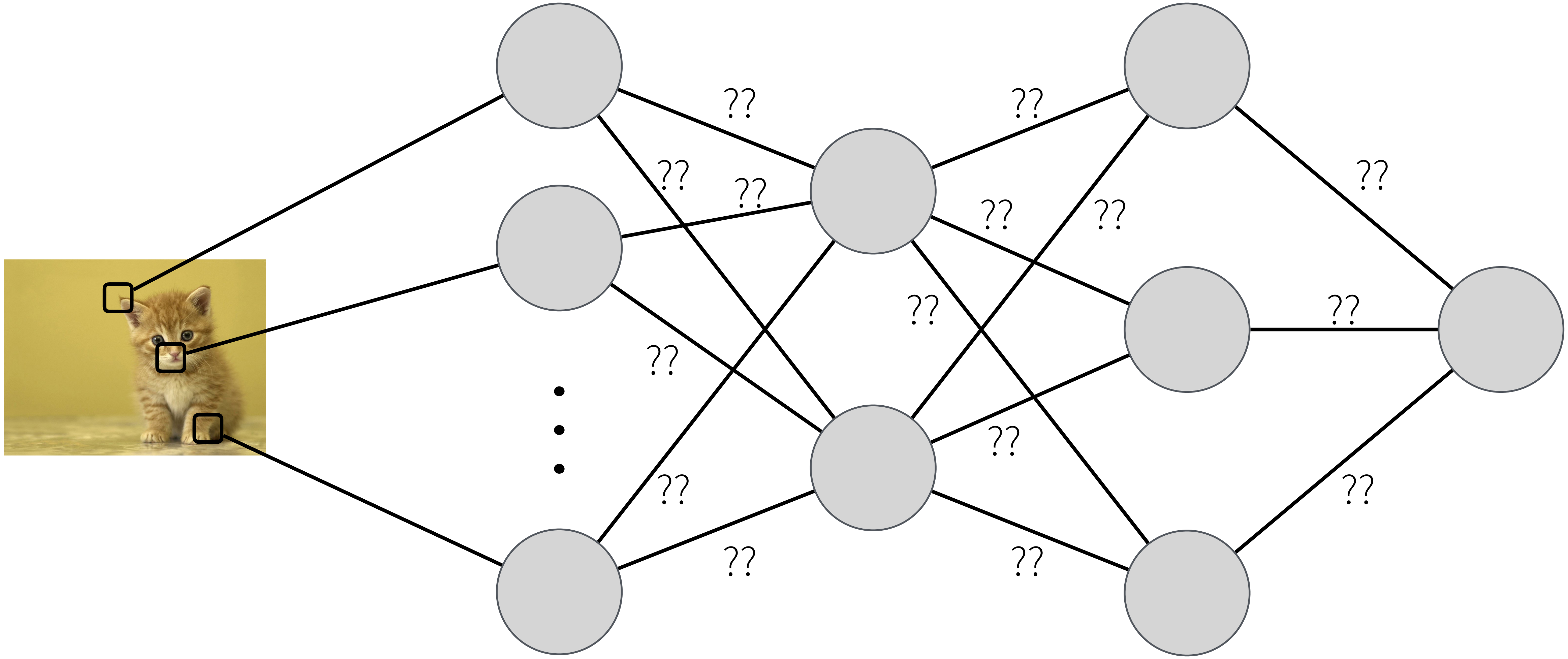
Is this a cat?



Synapse reasons about complex costs



Synapse reasons about complex costs



$$\kappa(P) = \sum_i |P(x_i) - y_i|$$

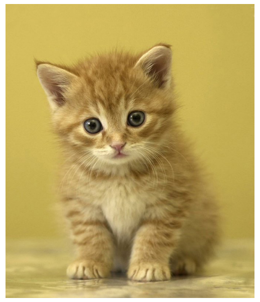
Classification error executes the program for each point in the training set

Metasketches

Design and structure

Synapse

A metasketch solver



Results

Better solutions, faster

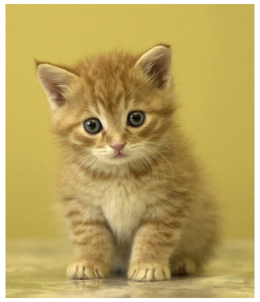
Metasketches

Design and structure

Synapse

A metasketch solver

`synapse.uwplse.org`



Results

Better solutions, faster