Optimizing synthesis with metasketches

James Bornholt, Emina Torlak,
Dan Grossman, and Luis Ceze

University of Washington

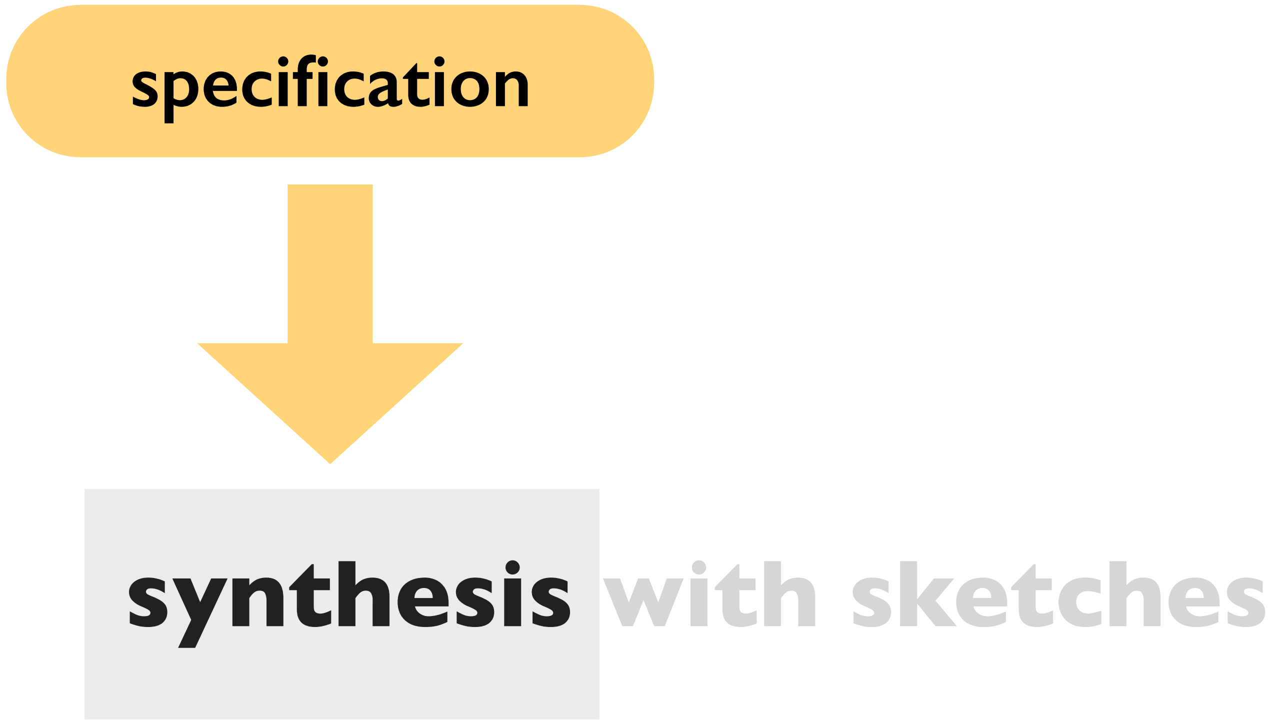
outline

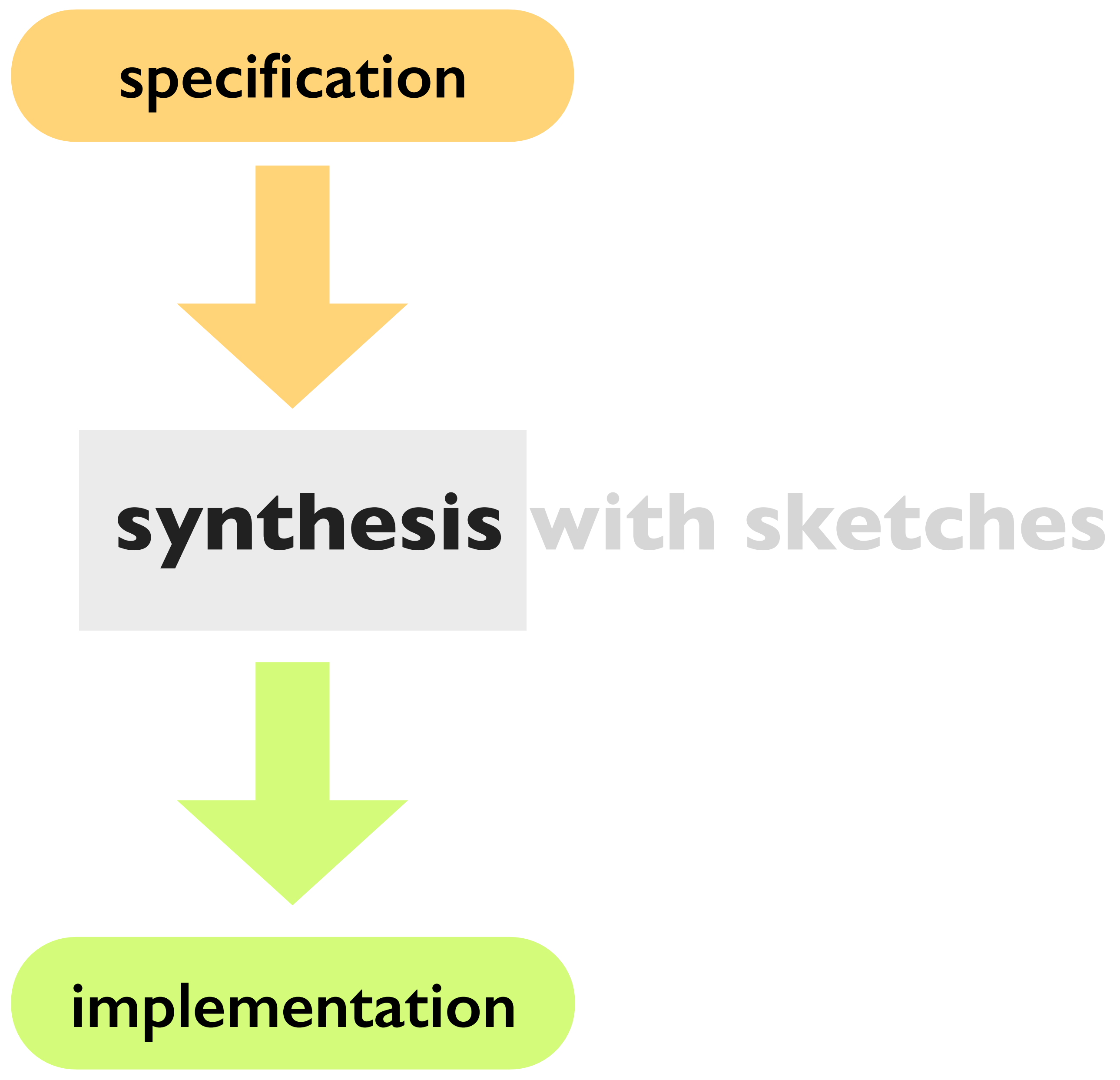
1. introduction
2. metaskeches
3. synapse
4. results

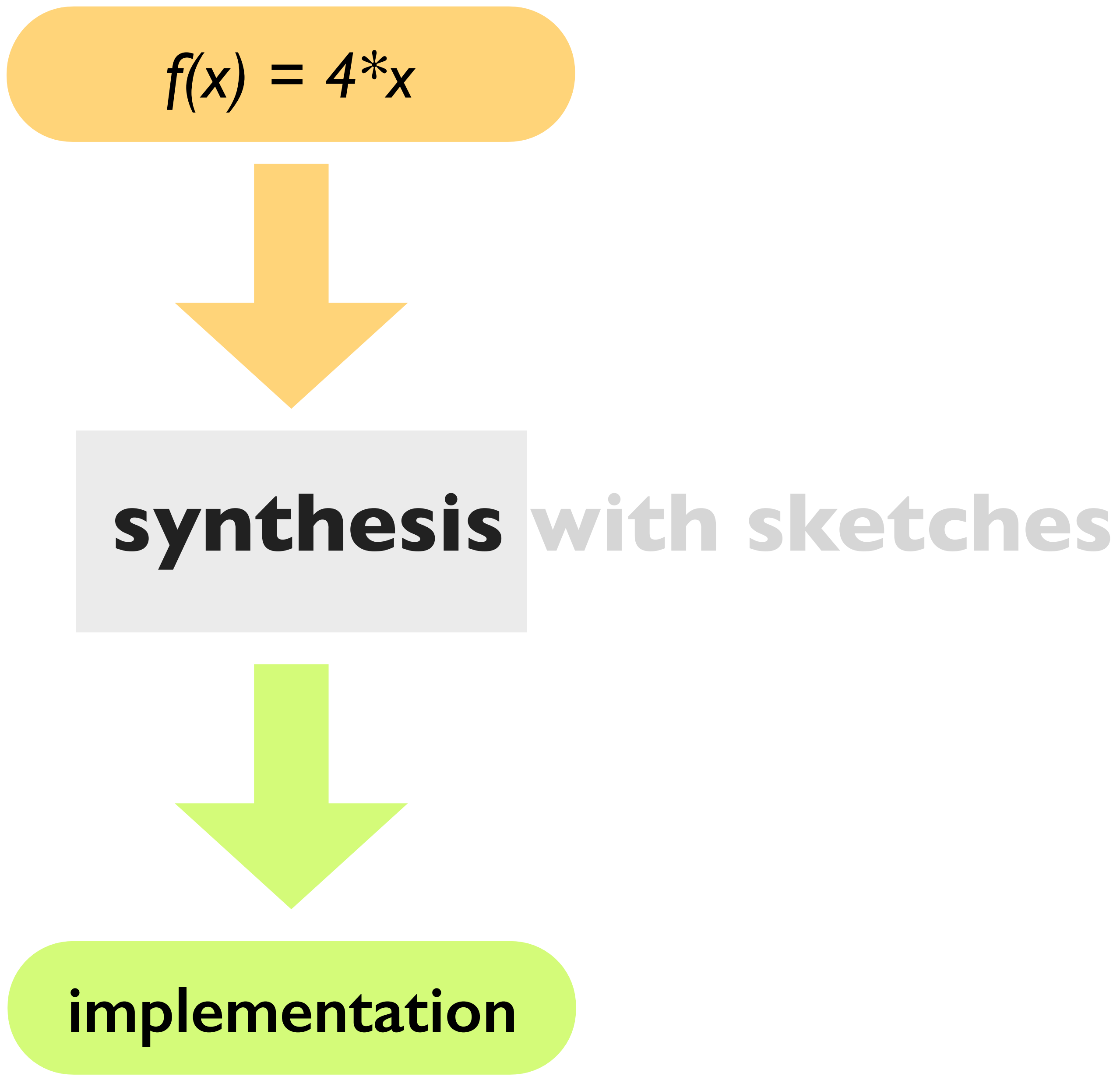
intro

synthesis with sketches

synthesis with sketches


specification
synthesis with sketches

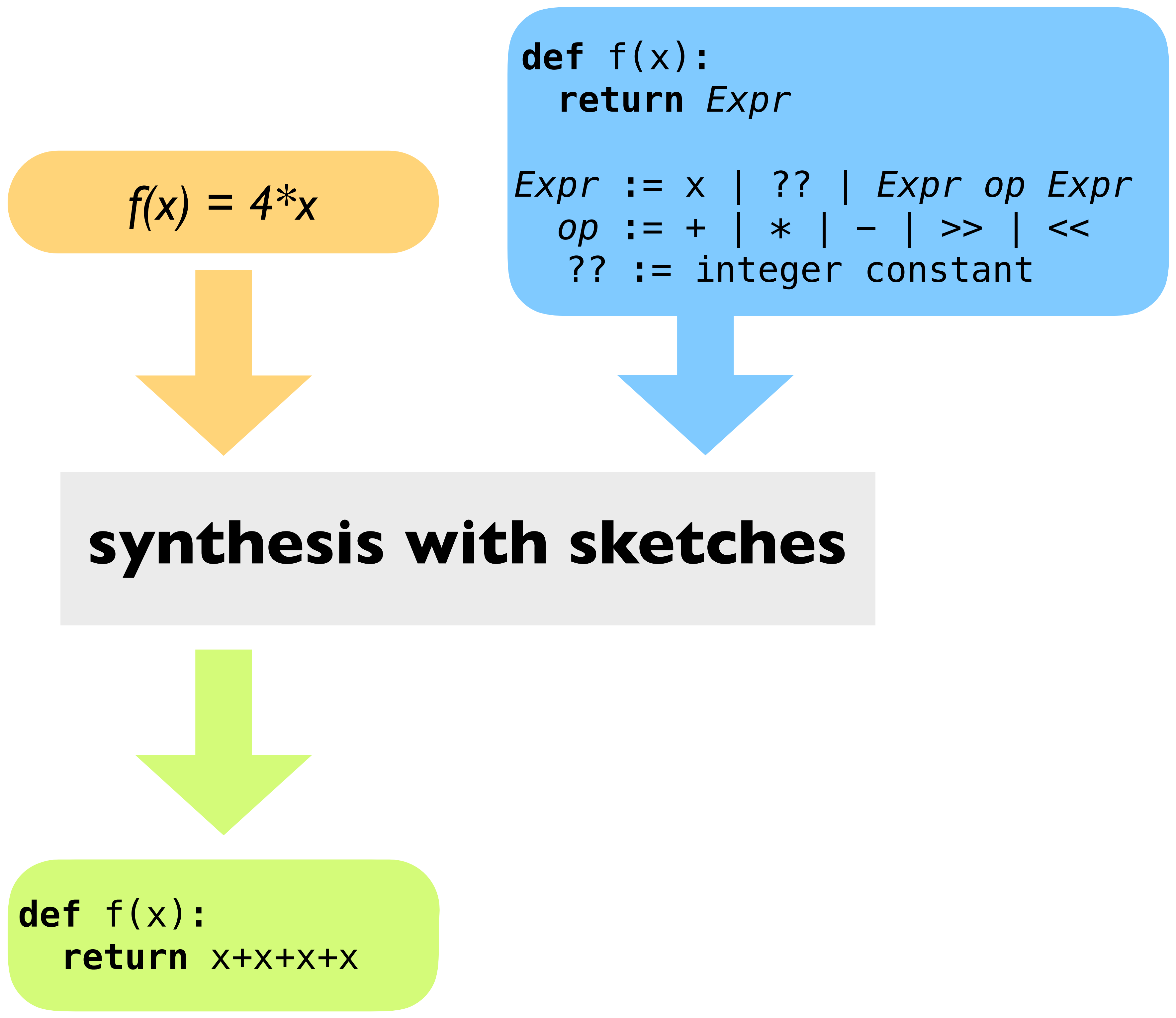

specification
synthesis with sketches
implementation


$f(x) = 4*x$
synthesis with sketches
implementation

f(*x*) = 4**x*

synthesis with sketches

```
def f(x):
  return x+x+x+x
```

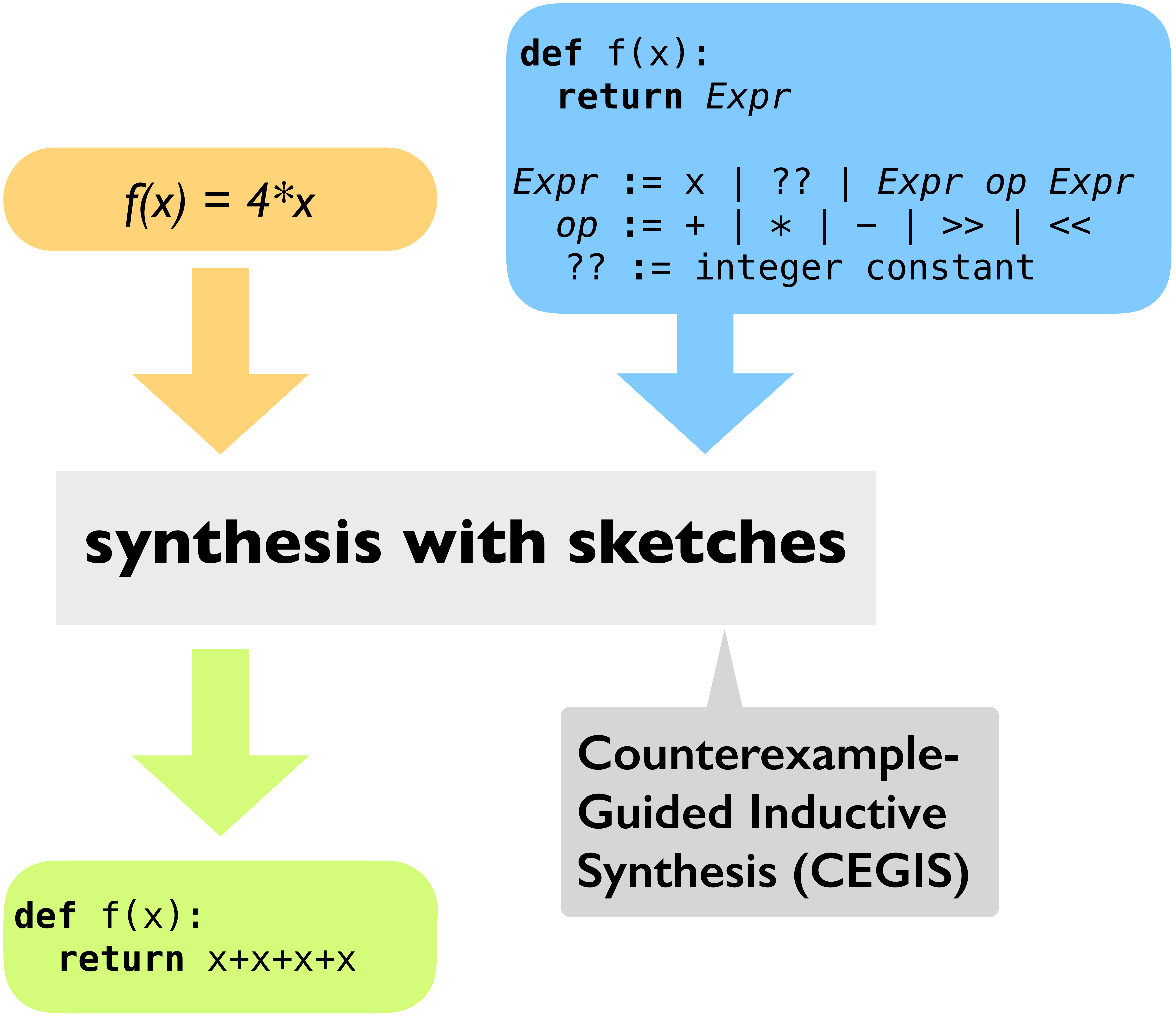


$f(x) = 4*x$

```
def f(x):
  return Expr

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

synthesis with sketches

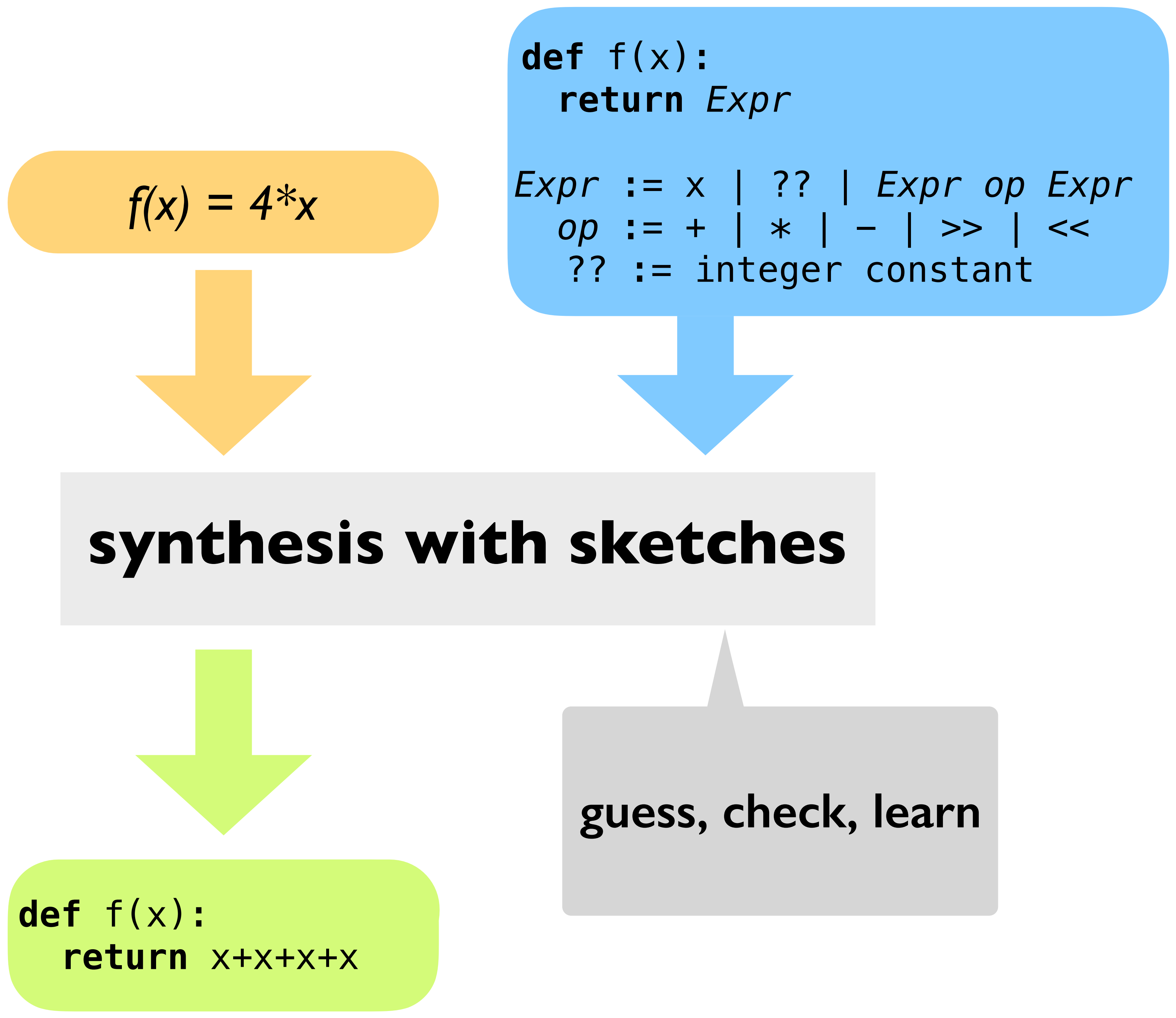
```
def f(x):
  return x+x+x+x
```



$f(x) = 4*x$

```
def f(x):
  return Expr

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

```
def f(x):
  return x+x+x+x
```



```
def f(x):
  return Expr

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

```
def f(x):
  return x+x+x+x
```

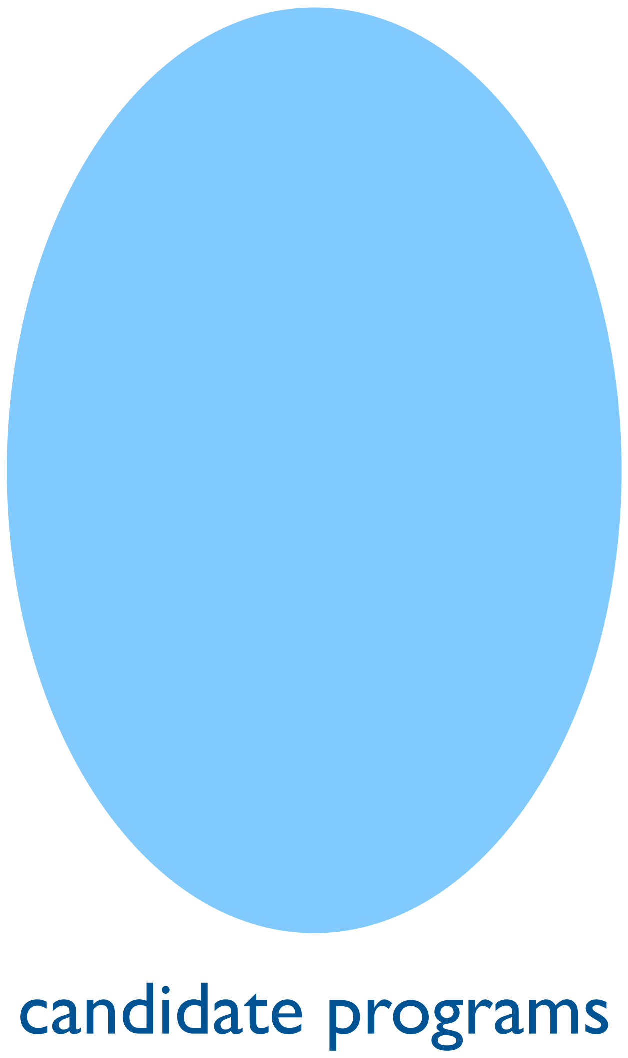
f(*x*) = 4**x*

```
def f(x):
  return Expr

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

synthesis with sketches

guess, check, learn

```
def f(x):
  return x+x+x+x
```



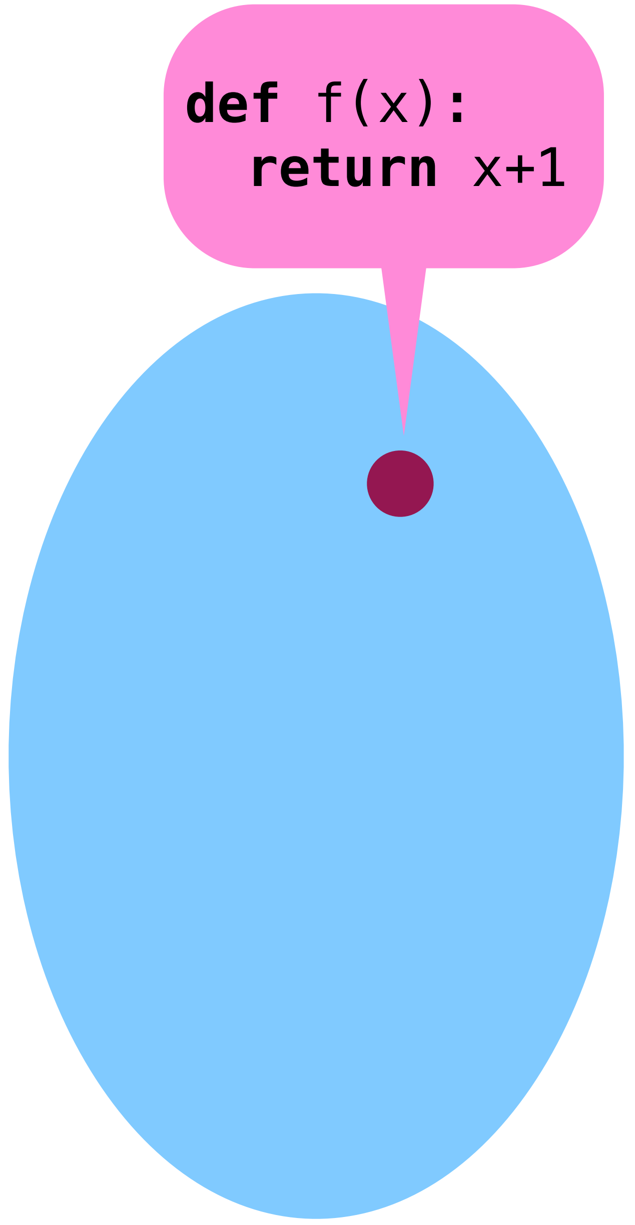
f(*x*) = *4***x*

```
def f(x):
  return Expr

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

synthesis with sketches

guess, check, learn

```
def f(x):
  return x+x+x+x
```



```
def f(x):
  return x+1
```

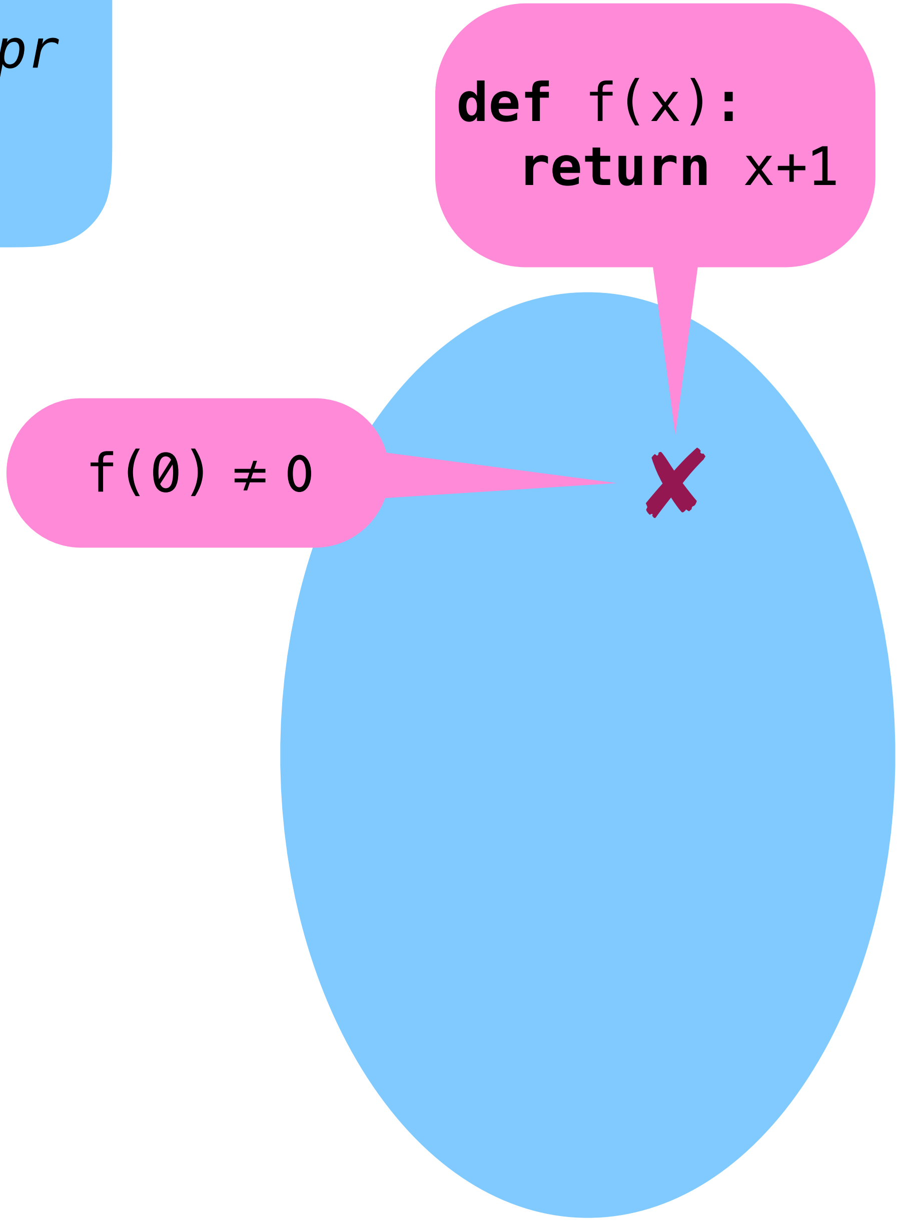
*f(x) = 4*x*

```
def f(x):
  return Expr

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

synthesis with sketches

guess, check, learn

```
def f(x):
  return x+x+x+x
```



```
def f(x):
  return x+1
```

candidate programs

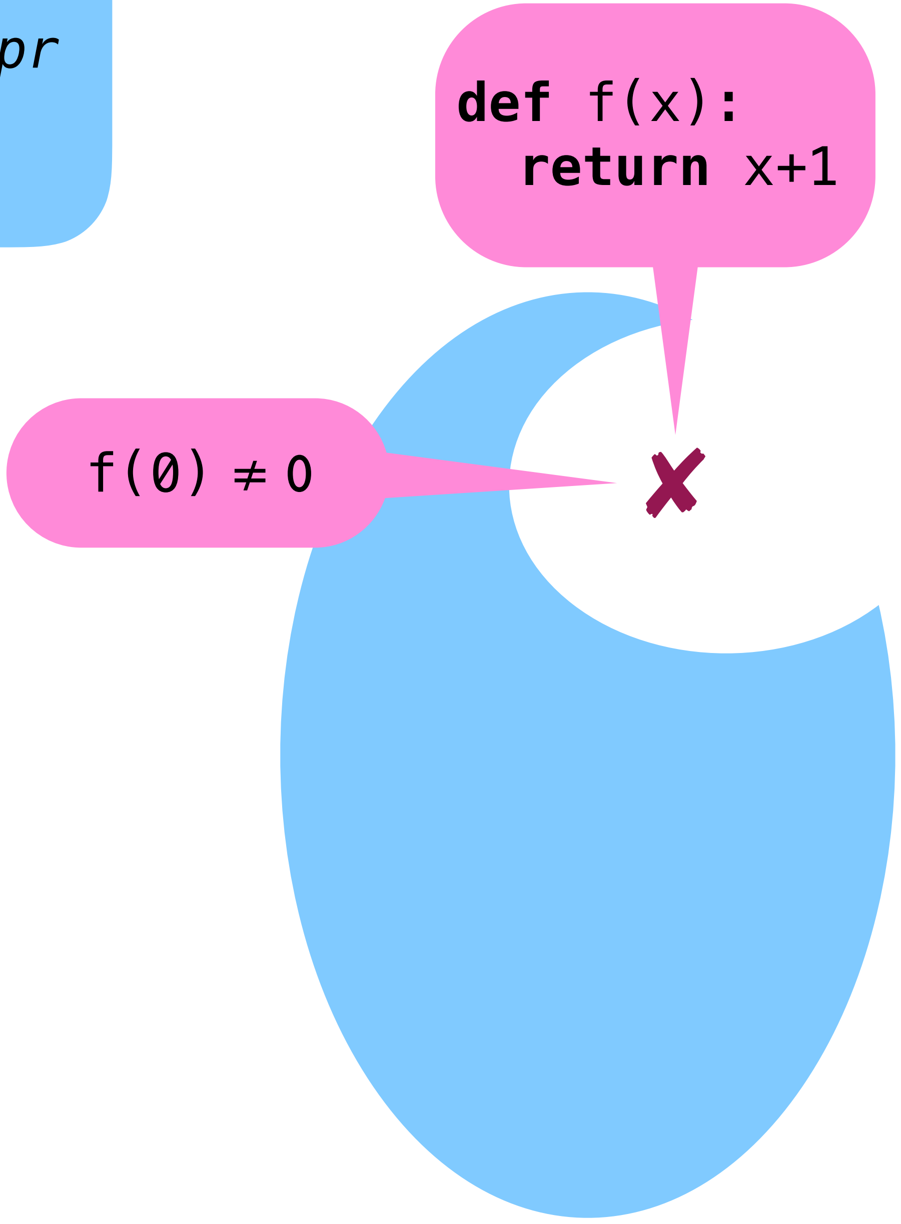
*f(x) = 4*x*

```
def f(x):
  return Expr

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

synthesis with sketches

```
def f(x):
  return x+x+x+x
```

guess, check, learn



```
def f(x):
  return x+1
```

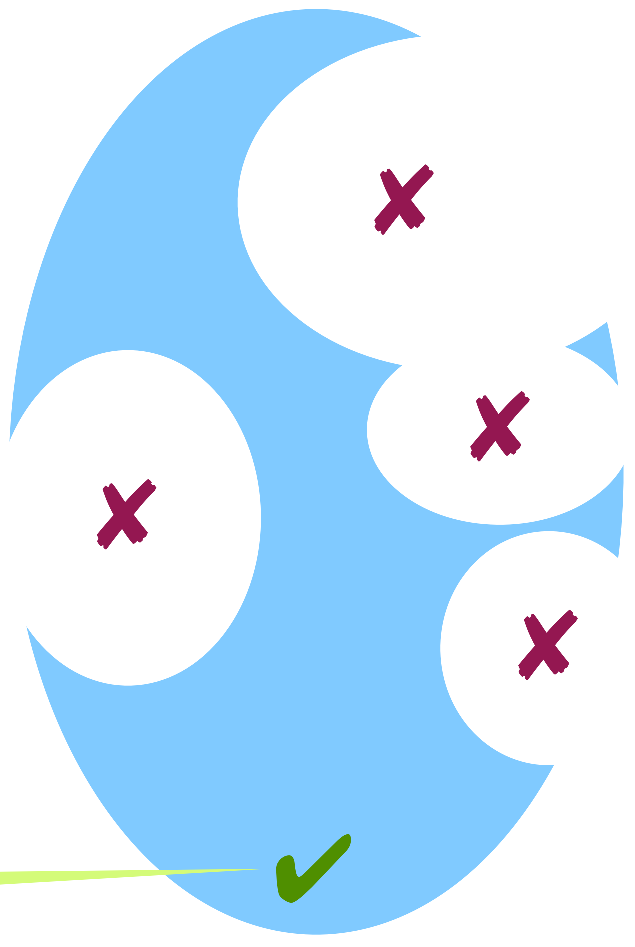
candidate programs

*f(x) = 4*x*

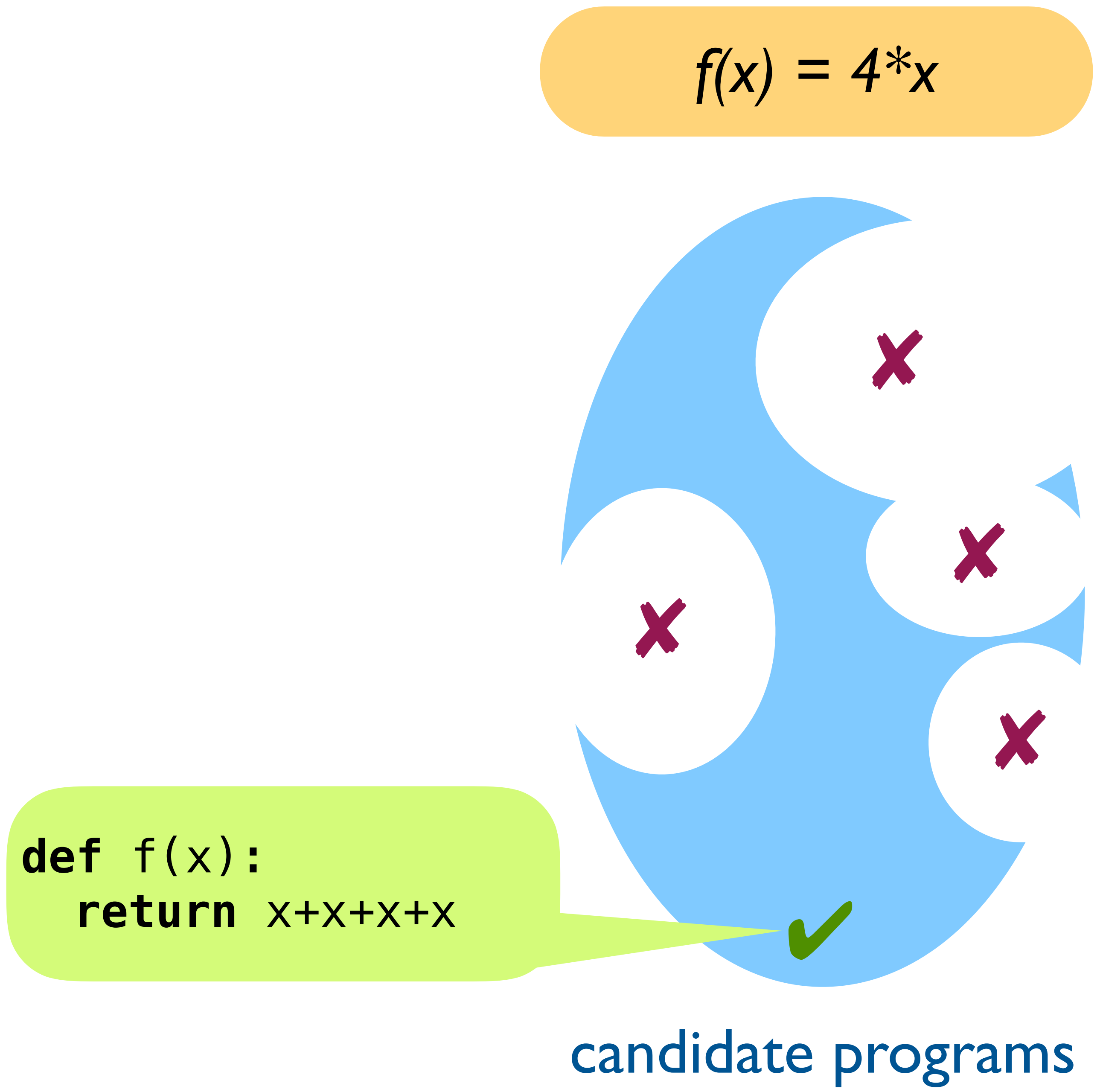
```
def f(x):
  return Expr

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

synthesis with sketches

```
def f(x):
  return x+x+x+x
```


candidate programs

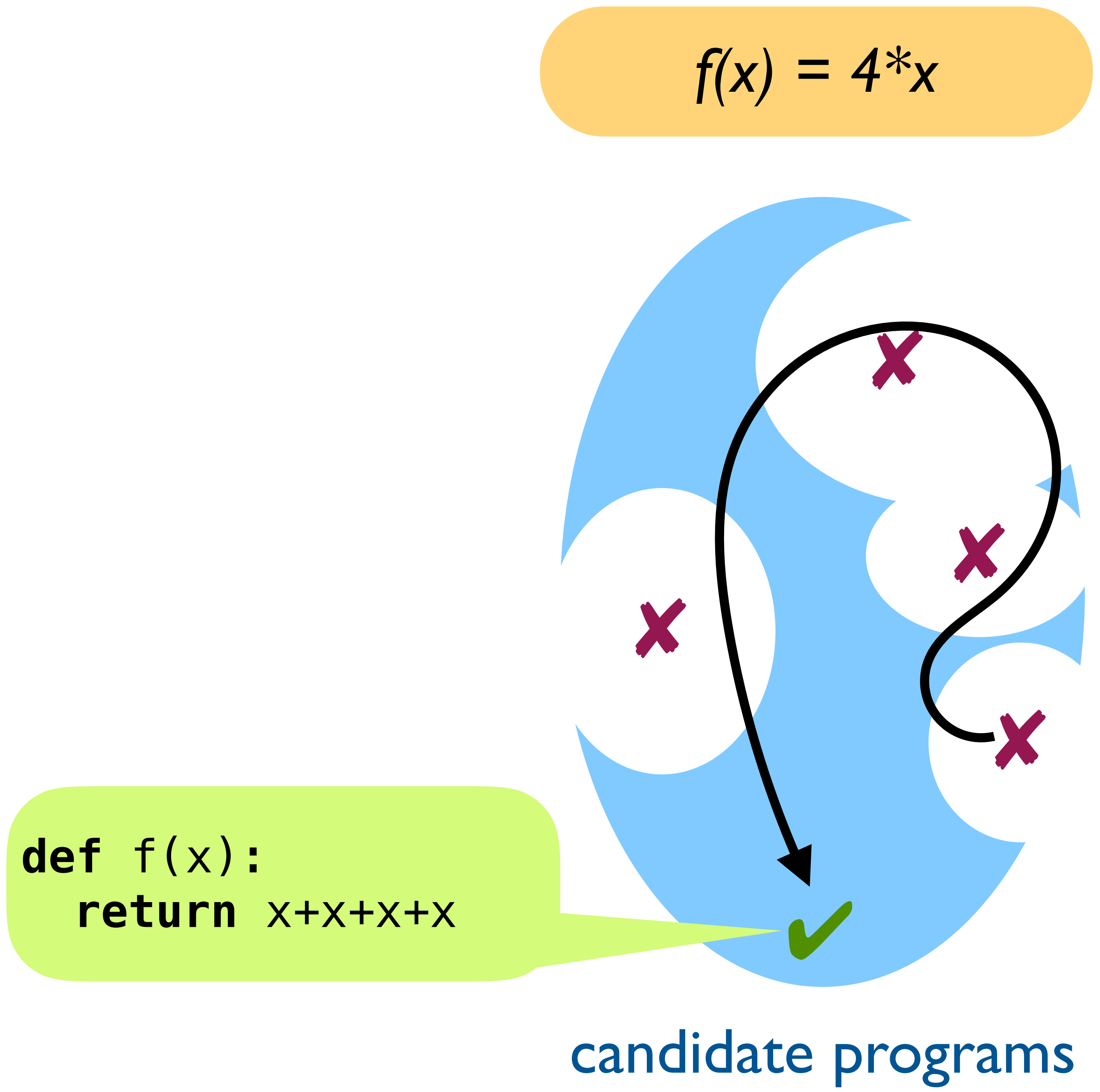
building a practical synthesizer is hard …



```
def f(x):
  return x+x+x+x
```

building a practical synthesizer is hard ...

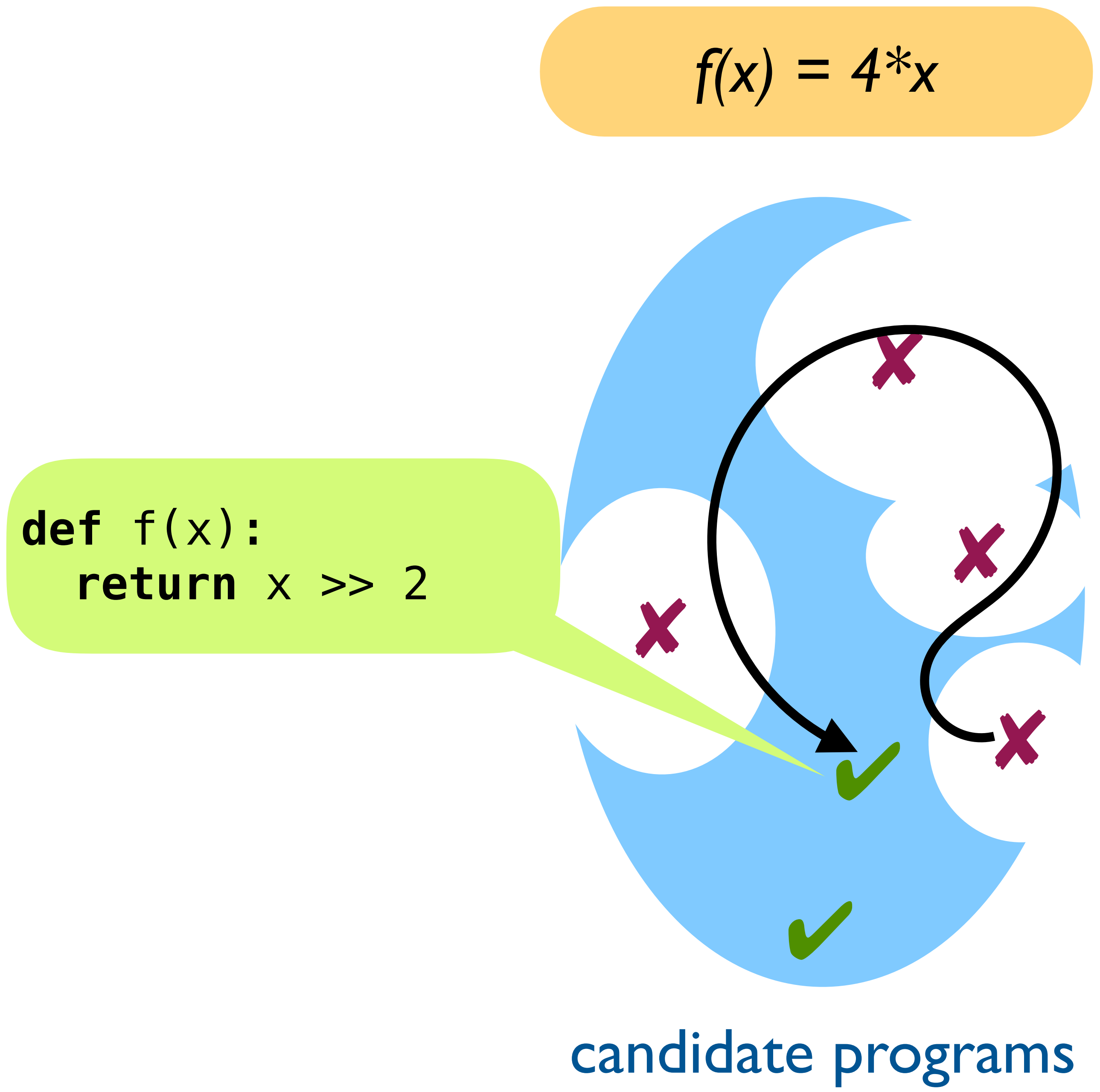
1. pick the right search strategy



```
def f(x):
  return x+x+x+x
```

building a practical synthesizer is hard …

1. pick the right search strategy
2. find the *best* correct program

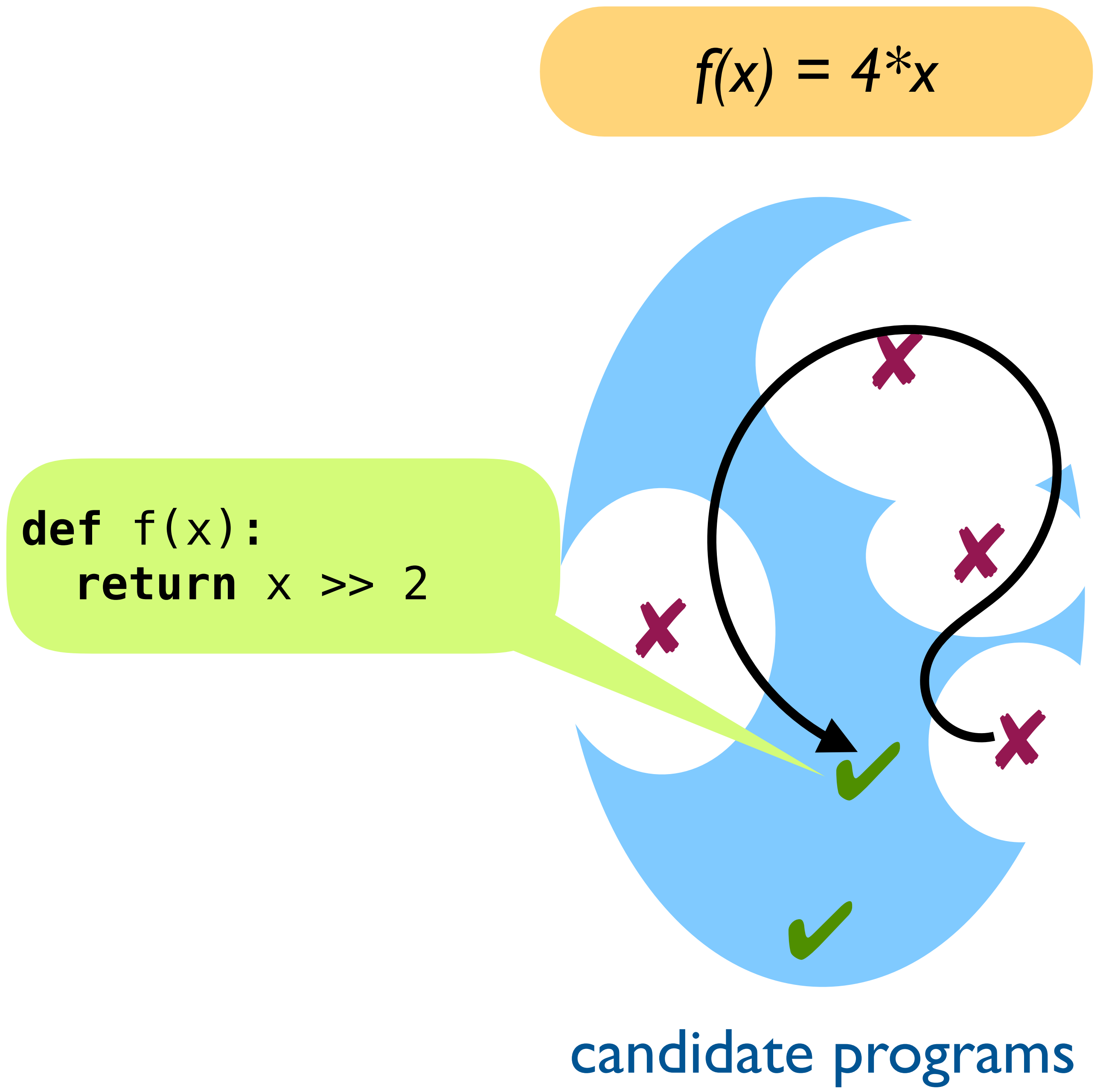


```
def f(x):
  return x >> 2
```

building a practical synthesizer is hard …

Existing tools hardcode both the search strategy and cost metric (if any), so are difficult to reuse for new domains.

1. **pick the right search strategy**
2. **find the *best* correct program**



```
def f(x):
  return x >> 2
```

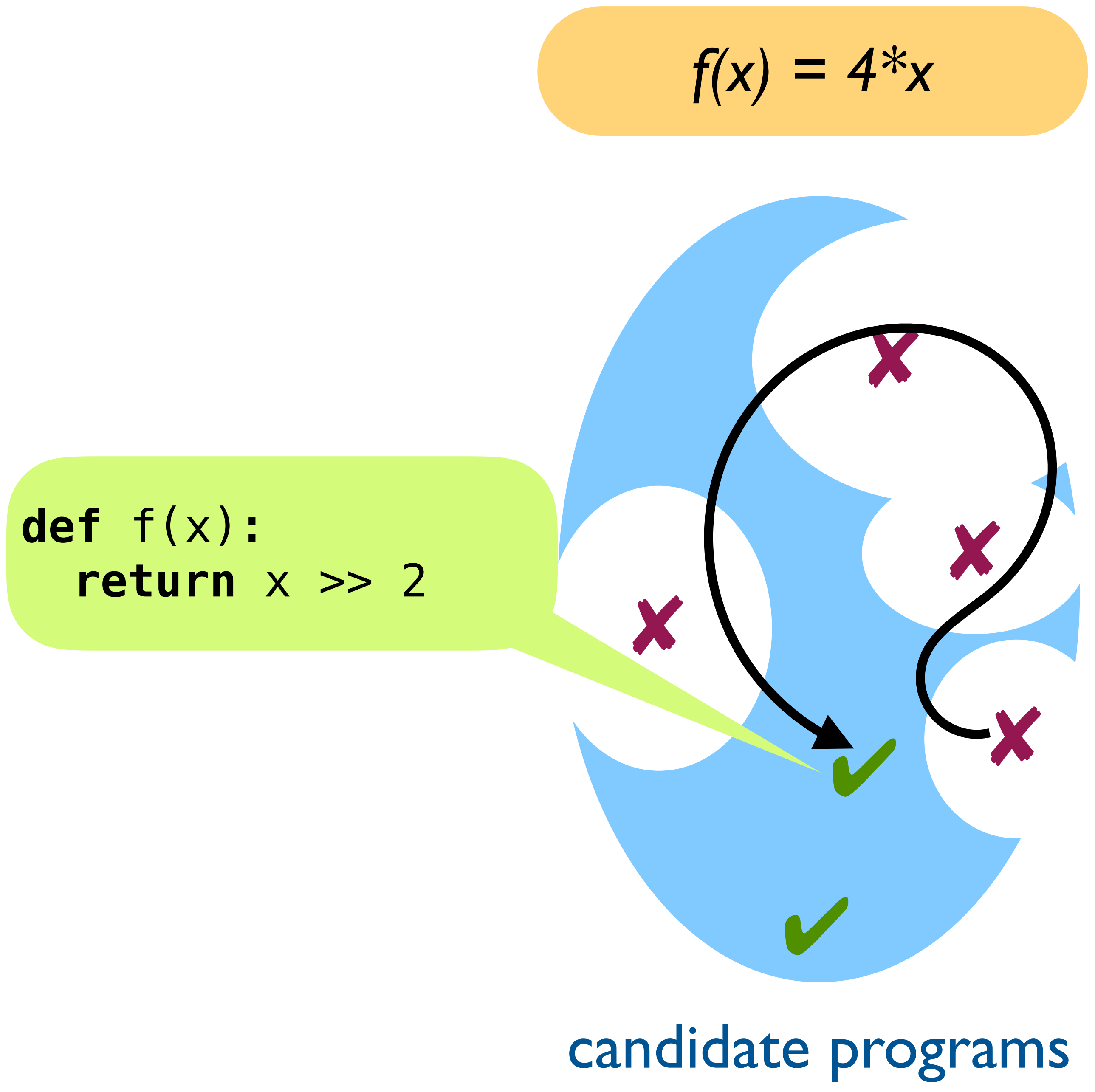
building a practical synthesizer is hard …

Existing tools hardcode both the search strategy and cost metric (if any), so are difficult to reuse for new domains.

1. pick the right search strategy

2. find the *best* correct program

Metasketches are a new way to express synthesis problems, making the search strategy and the cost function explicit in the problem definition.



```
def f(x):
  return x >> 2
```

design

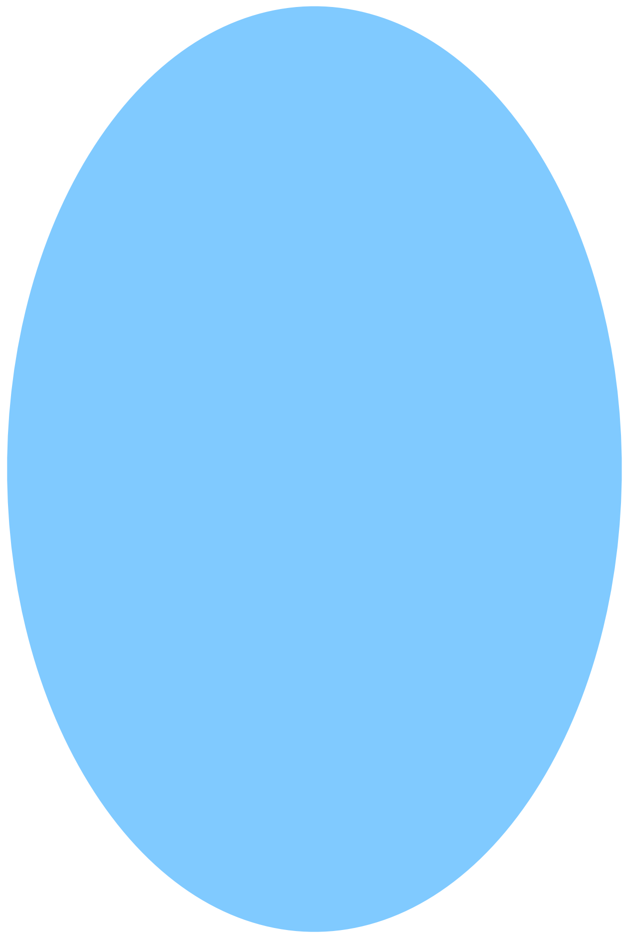
metasketches

anatomy of a metasketch

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

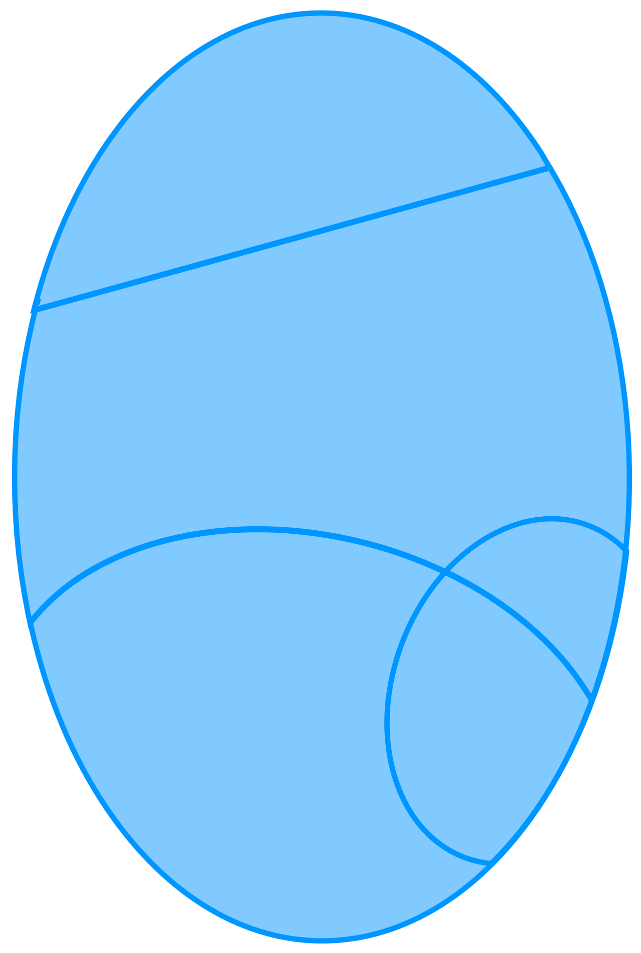
anatomy of a metasketch

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



anatomy of a metasketch

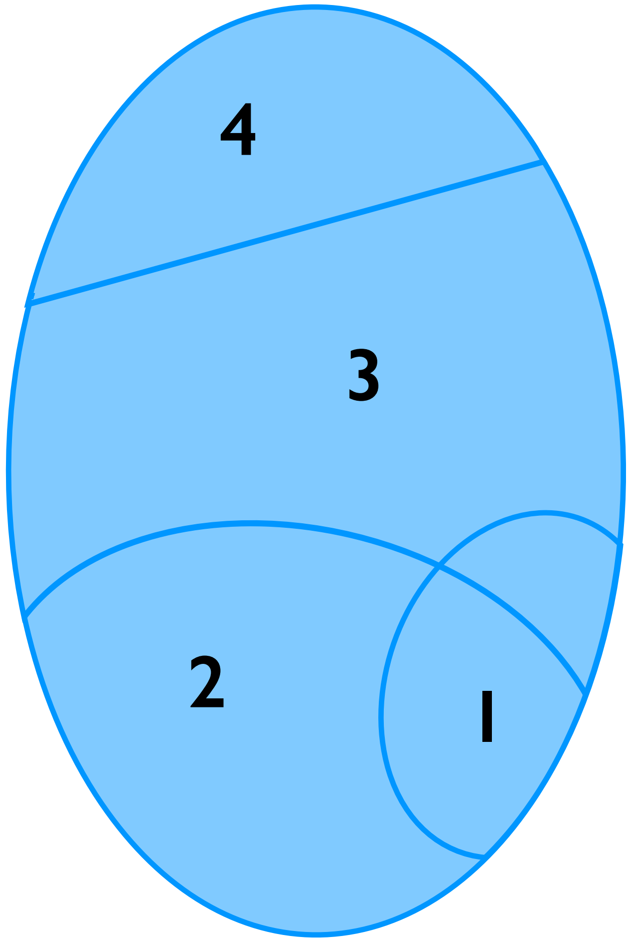
1. **structured candidate space ($\mathcal{S}$, $\preccurlyeq$)**
2. **cost function (κ)**
3. **gradient function (g)**



candidate programs

anatomy of a metasketch

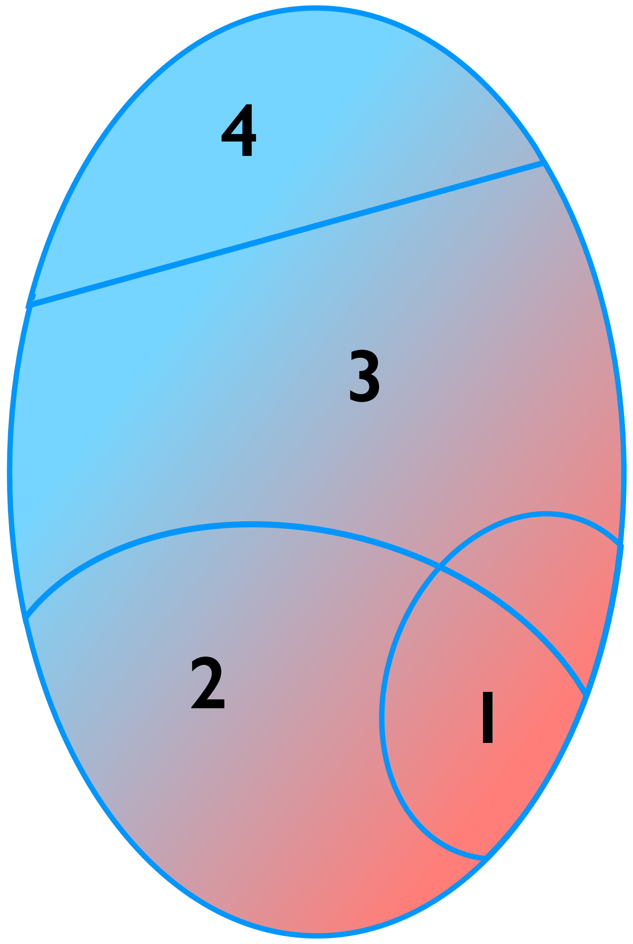
1. structured candidate space $(\mathcal{S}, \preccurlyeq)$
2. cost function (κ)
3. gradient function (g)



candidate programs

anatomy of a metasketch

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



candidate programs

anatomy of a metasketch (1)

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)**

anatomy of a metasketch (1)

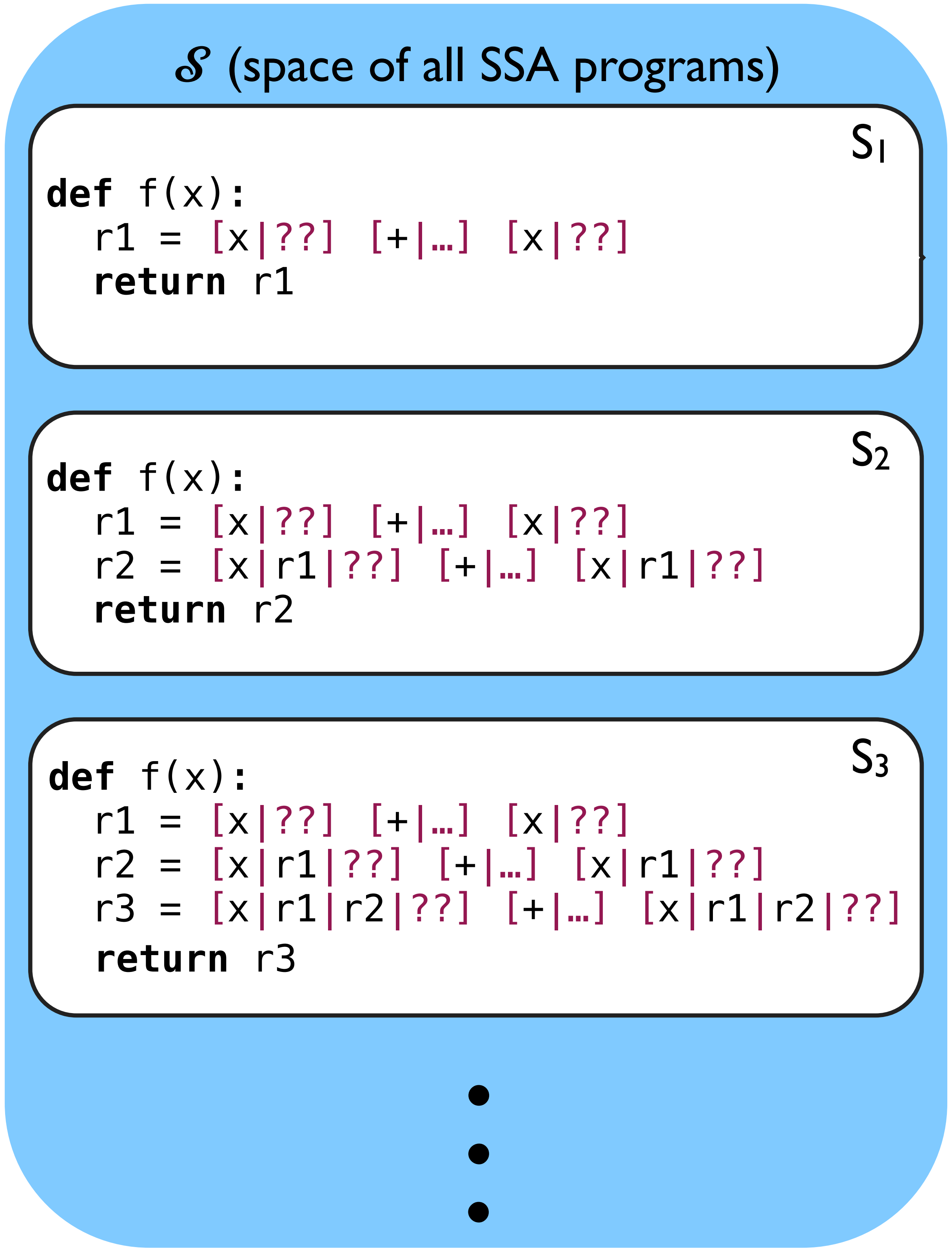
1. **structured candidate space ($\mathcal{S}$, $\preccurlyeq$)**
   - a countable set $\mathcal{S}$ of sketches
   - a total order $\preccurlyeq$ on $\mathcal{S}$
2. **cost function (κ)**
3. **gradient function (g)**

A set of sketches $\mathcal{S}$ can express candidate spaces that cannot be expressed with a single sketch or a CFG.

anatomy of a metasketch (1)

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)**

A set of sketches $\mathcal{S}$ can express candidate spaces that cannot be expressed with a single sketch or a CFG.



```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  return r1
```

```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  r2 = [x|r1|??] [+|…] [x|r1|??]
  return r2
```

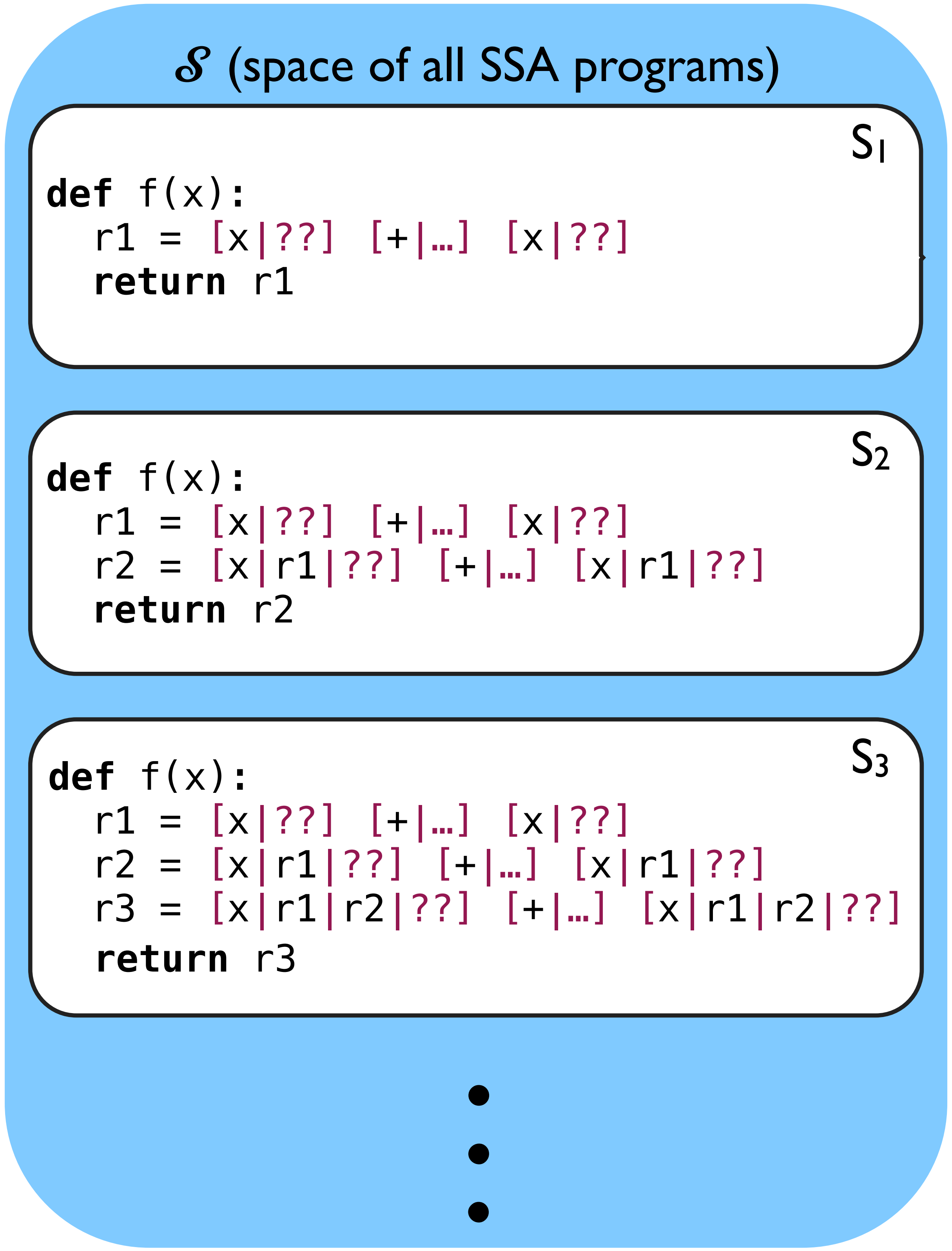
```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  r2 = [x|r1|??] [+|…] [x|r1|??]
  r3 = [x|r1|r2|??] [+|…] [x|r1|r2|??]
  return r3
```

anatomy of a metasketch (1)

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)**

A set of sketches $\mathcal{S}$ can express candidate spaces that cannot be expressed with a single sketch or a CFG.

The ordering $\preccurlyeq$ on sketches expresses a high-level search strategy.


$\mathcal{S}$ (space of all SSA programs)

S_1
```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  return r1
```

S_2
```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  r2 = [x|r1|??] [+|…] [x|r1|??]
  return r2
```

S_3
```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  r2 = [x|r1|??] [+|…] [x|r1|??]
  r3 = [x|r1|r2|??] [+|…] [x|r1|r2|??]
  return r3
```

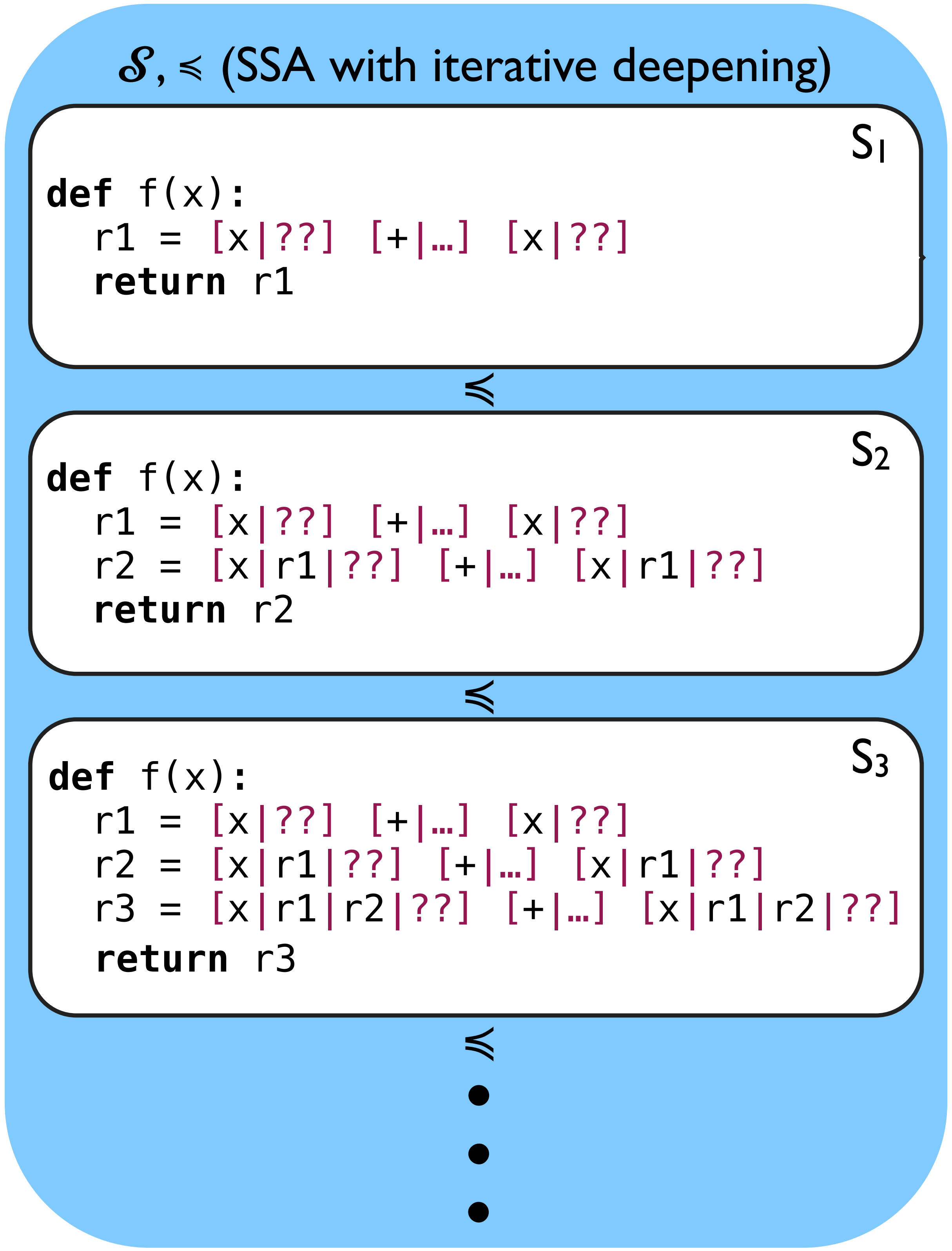
⋮

anatomy of a metasketch (1)

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)**

A set of sketches $\mathcal{S}$ can express candidate spaces that cannot be expressed with a single sketch or a CFG.

The ordering $\preccurlyeq$ on sketches expresses a high-level search strategy.


$\mathcal{S}$, $\preccurlyeq$ (SSA with iterative deepening)
S_1
$\preccurlyeq$
S_2
$\preccurlyeq$
S_3
$\preccurlyeq$

```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  return r1
```

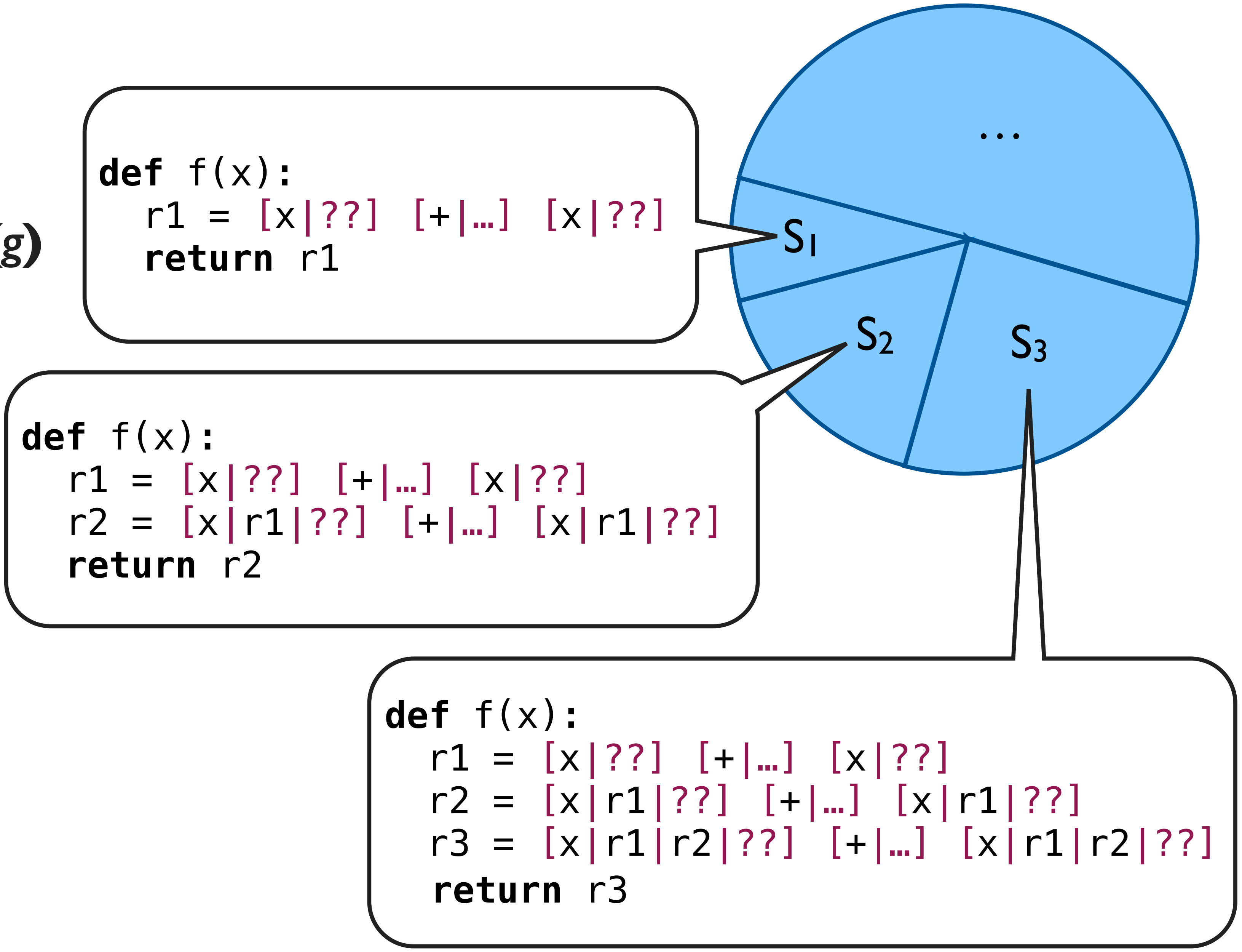
```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  r2 = [x|r1|??] [+|…] [x|r1|??]
  return r2
```

```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  r2 = [x|r1|??] [+|…] [x|r1|??]
  r3 = [x|r1|r2|??] [+|…] [x|r1|r2|??]
  return r3
```

anatomy of a metasketch (1)

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)**

$\mathcal{S}$, $\preccurlyeq$ (SSA with iterative deepening)



```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  return r1
```

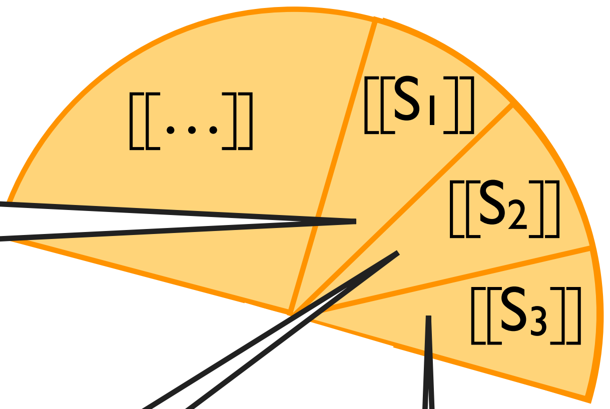
```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  r2 = [x|r1|??] [+|…] [x|r1|??]
  return r2
```

```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  r2 = [x|r1|??] [+|…] [x|r1|??]
  r3 = [x|r1|r2|??] [+|…] [x|r1|r2|??]
  return r3
```

anatomy of a metasketch (1)

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)**

$\mathcal{S}$, $\preccurlyeq$ (SSA with iterative deepening)



```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  return r1
```

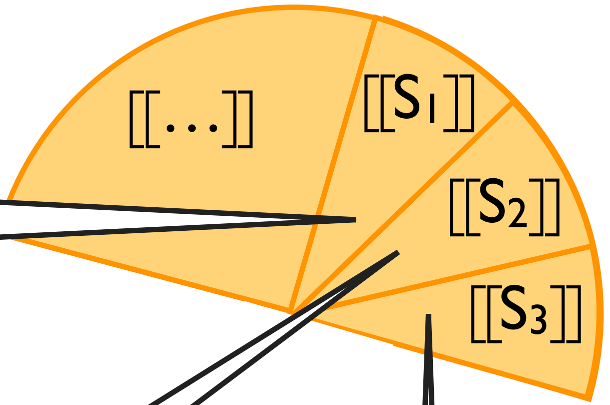
```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  r2 = [x|r1|??] [+|…] [x|r1|??]
  return r2
```

```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  r2 = [x|r1|??] [+|…] [x|r1|??]
  r3 = [x|r1|r2|??] [+|…] [x|r1|r2|??]
  return r3
```

anatomy of a metasketch (1)

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)**

$\mathcal{S}$, $\preccurlyeq$ (SSA with iterative deepening)



```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  return r1
```

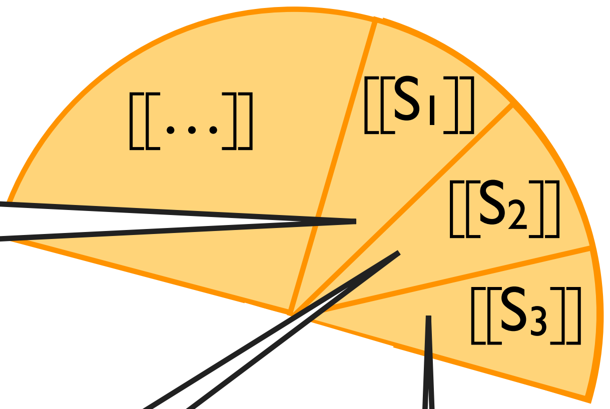
```
def f(x):
  r1 = e11 [+|…] e12
  r2 = e21 [+|…] e22
  return r2
```

```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  r2 = [x|r1|??] [+|…] [x|r1|??]
  r3 = [x|r1|r2|??] [+|…] [x|r1|r2|??]
  return r3
```

anatomy of a metasketch (1)

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)**

$\mathcal{S}$, $\preccurlyeq$ (SSA with iterative deepening)



```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  return r1
```

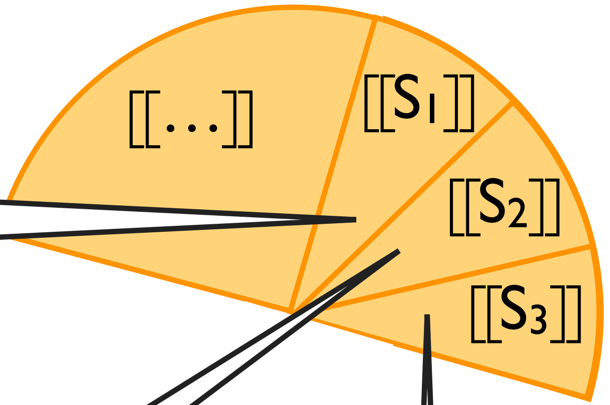
```
def f(x):
  r1 = e11 [+|…] e12
  r2 = e21 [+|…] e22
  assert r1 == e21 || r1 == e22
  return r2
```

```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  r2 = [x|r1|??] [+|…] [x|r1|??]
  r3 = [x|r1|r2|??] [+|…] [x|r1|r2|??]
  return r3
```

anatomy of a metasketch (1)

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)**

$\mathcal{S}$, $\preccurlyeq$ (SSA with iterative deepening)



```
def f(x):
  r1 = [x|??] [+|…] [x|??]
  return r1
```

```
def f(x):
  r1 = e11 [+|…] e12
  r2 = e21 [+|…] e22
  assert r1 == e21 || r1 == e22
  return r2
```

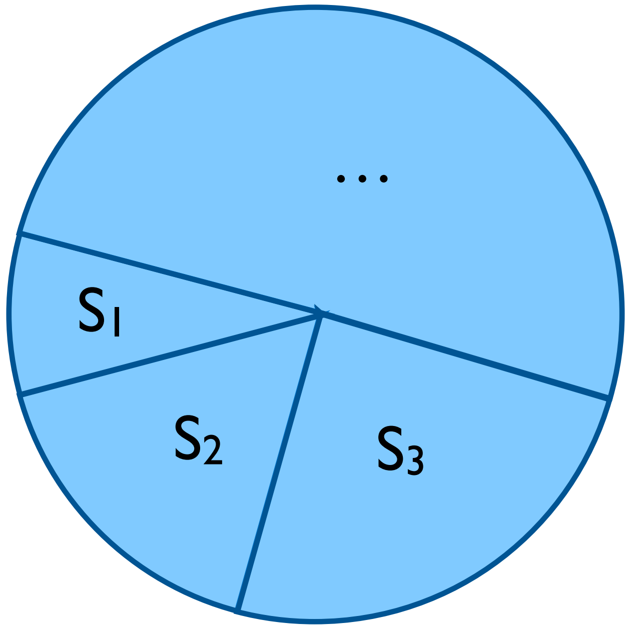
```
def f(x):
  r1 = e11 [+|…] e12
  r2 = e21 [+|…] e22
  r3 = e31 [+|…] e32
  assert r1 == e21 || … || r1 == e32
  assert r2 == e31 || r2 == e32
  return r3
```

Structure constraints help reduce semantic redundancies in the search space.

anatomy of a metasketch (2)

1. **structured candidate space ($\mathcal{S}$, $\preccurlyeq$)**
2. **cost function (κ)**
   - $\kappa : \{ P \mid P \in S_i \land S_i \in \mathcal{S} \} \rightarrow \mathbb{R}$
   - assigns a numeric cost to each candidate
3. **gradient function (g)**

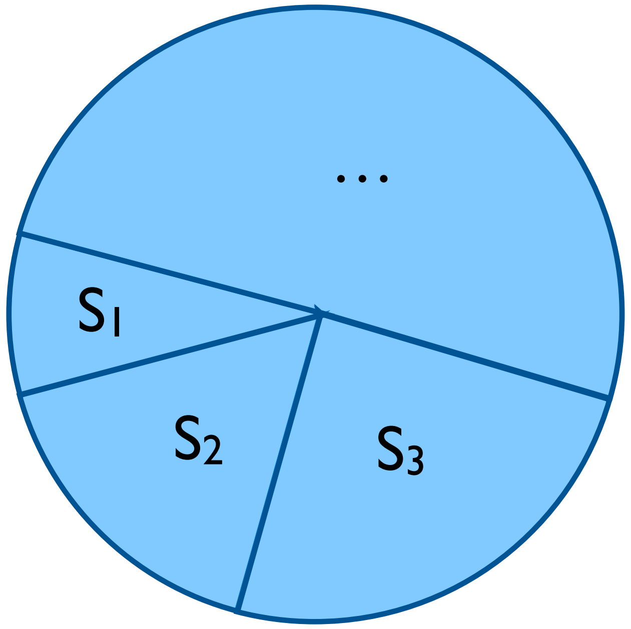
$\mathcal{S}$, $\preccurlyeq$ (SSA with iterative deepening)



anatomy of a metasketch (2)

1. **structured candidate space ($\mathcal{S}$, $\preccurlyeq$)**
2. **cost function (κ)**
   - $\kappa : \{ P \mid P \in S_i \wedge S_i \in \mathcal{S} \} \rightarrow \mathbb{R}$
   - assigns a numeric cost to each candidate
3. **gradient function (g)**

$\mathcal{S}$, $\preccurlyeq$ (SSA with iterative deepening)



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

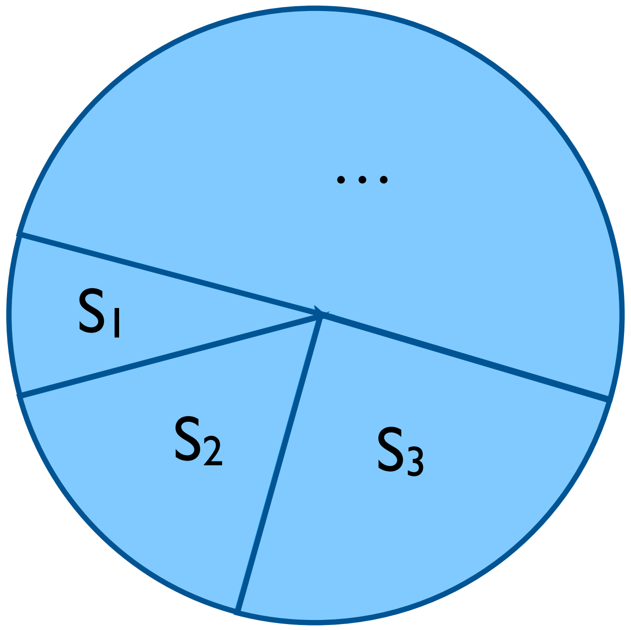
Counts the number of defined variables in the candidate program *P*.

anatomy of a metasketch (2)

1. **structured candidate space ($\mathcal{S}$, $\preccurlyeq$)**
2. **cost function (κ)**
   - $\kappa : \{ P \mid P \in S_i \wedge S_i \in \mathcal{S} \} \rightarrow \mathbb{R}$
   - assigns a numeric cost to each candidate
3. **gradient function (g)**

The cost function κ can be *static* (based on program syntax) or *dynamic* (based on runtime behavior).

$\mathcal{S}$, $\preccurlyeq$ (SSA with iterative deepening)



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

Counts the number of defined variables in the candidate program *P*.

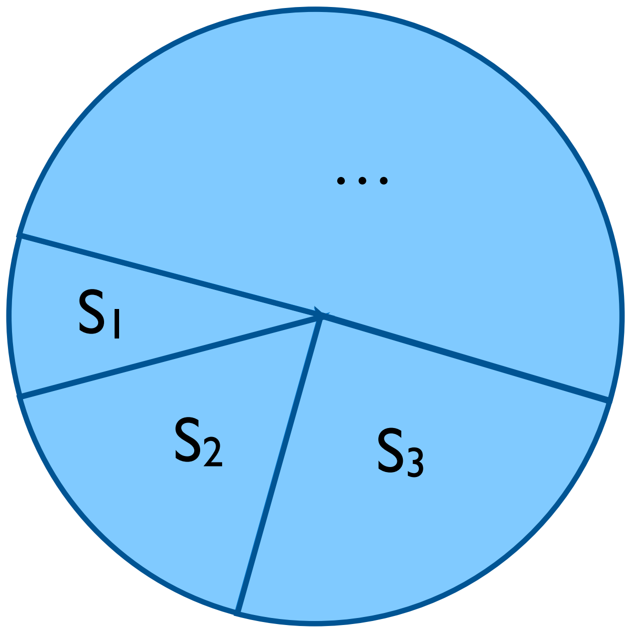
anatomy of a metasketch (2)

1. **structured candidate space ($\mathcal{S}$, $\preccurlyeq$)**
2. **cost function (κ)**
   - $\kappa : \{ P \mid P \in S_i \wedge S_i \in \mathcal{S} \} \to \mathbb{R}$
   - assigns a numeric cost to each candidate
3. **gradient function (g)**

The cost function κ can be *static* (based on program syntax) or *dynamic* (based on runtime behavior).

Any cost function κ can be used as long as the result of evaluating κ on a program *P* (and possibly its inputs) can be expressed as a term in a decidable theory.

$\mathcal{S}$, $\preccurlyeq$ (SSA with iterative deepening)



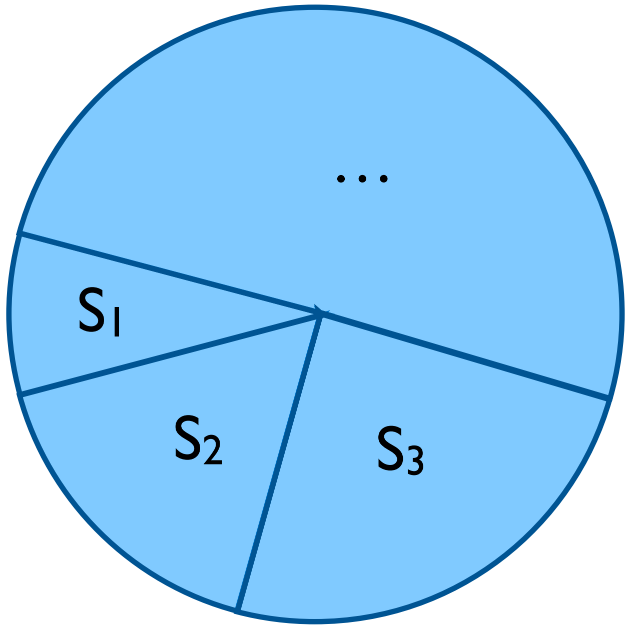
$\kappa(P) = i$ for $P \in S_i \in \mathcal{S}$

Counts the number of defined variables in the candidate program *P*.

anatomy of a metasketch (3)

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 all sketches in $\mathcal{S}$ that *may* contain a program P with $\kappa(P) < c$

$\mathcal{S}$, $\preccurlyeq$ (SSA with iterative deepening)



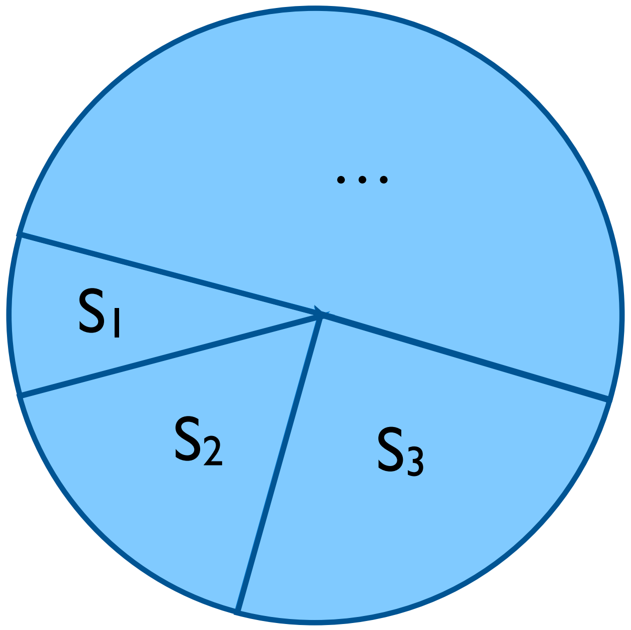
$\kappa(P) = i$ for $P \in S_i \in \mathcal{S}$

anatomy of a metasketch (3)

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 all sketches in $\mathcal{S}$ that *may* contain a program P with $\kappa(P) < c$

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

$\mathcal{S}$, $\preccurlyeq$ (SSA with iterative deepening)



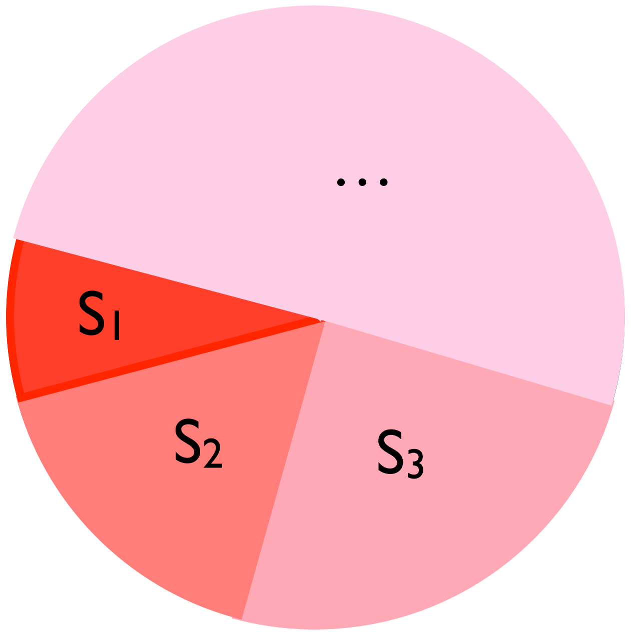
$\kappa(P) = i$ for $P \in S_i \in \mathcal{S}$

anatomy of a metasketch (3)

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 all sketches in $\mathcal{S}$ that *may* contain a program P with $\kappa(P) < c$

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

$\mathcal{S}$, $\preccurlyeq$ (SSA with iterative deepening)



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

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

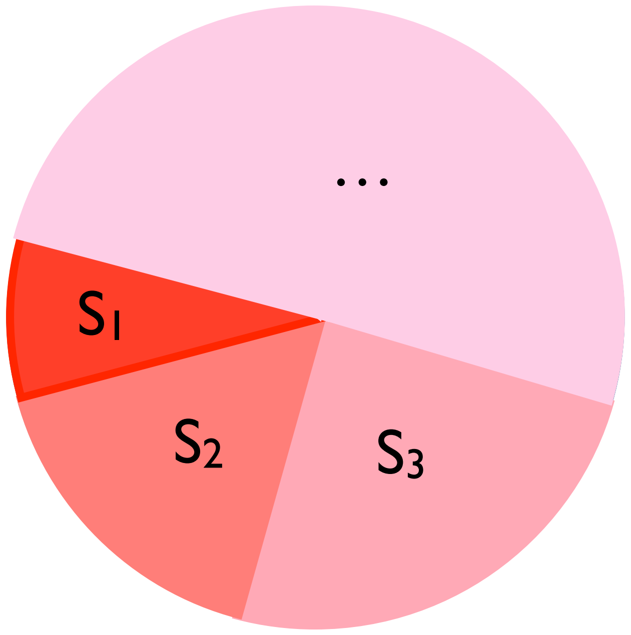
anatomy of a metasketch (3)

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 all sketches in $\mathcal{S}$ that *may* contain a program P with $\kappa(P) < c$

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

It is always sound to return the trivial gradient $g(c) = \mathcal{S}$.

$\mathcal{S}$, $\preccurlyeq$ (SSA with iterative deepening)



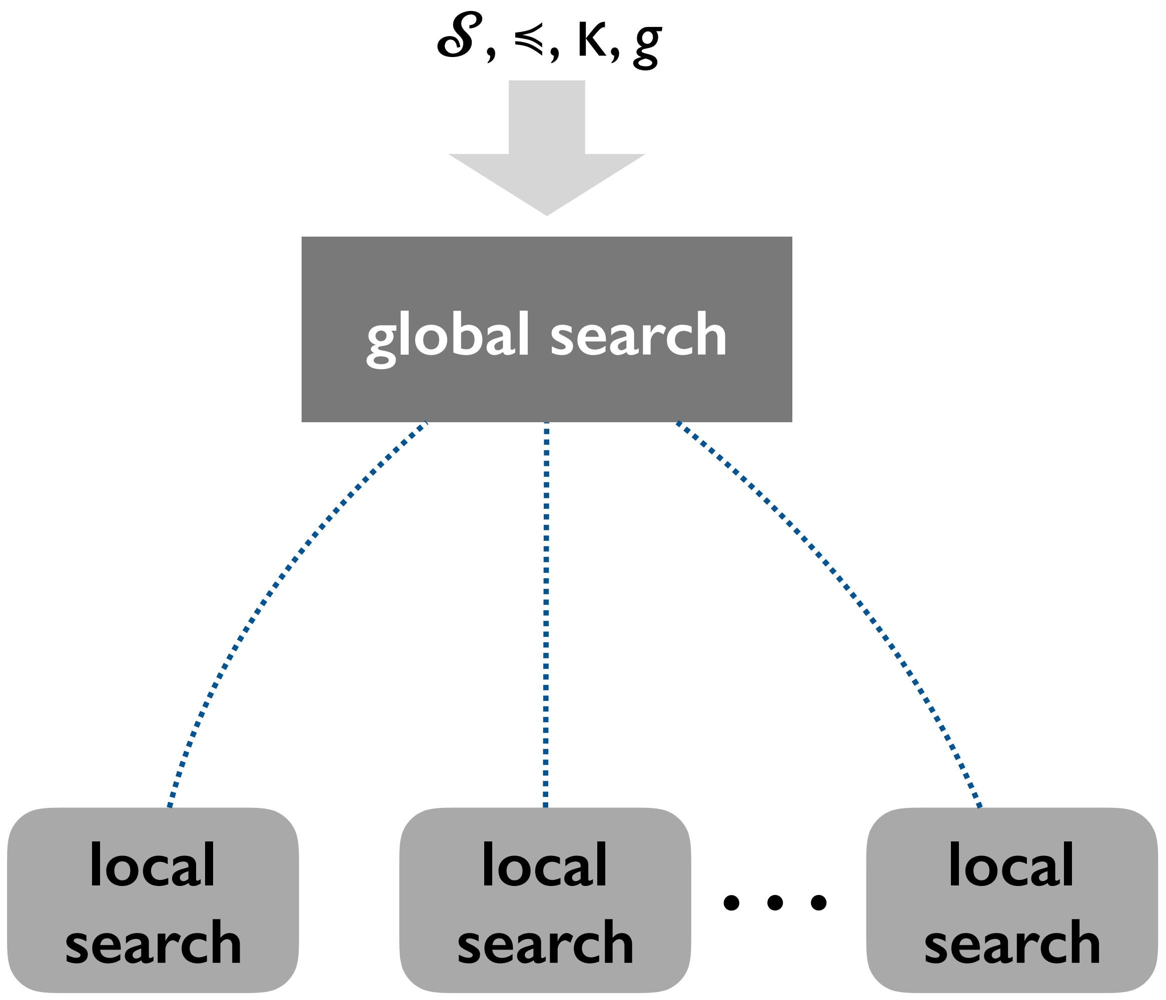
$\kappa(P) = i$ for $P \in S_i \in \mathcal{S}$

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

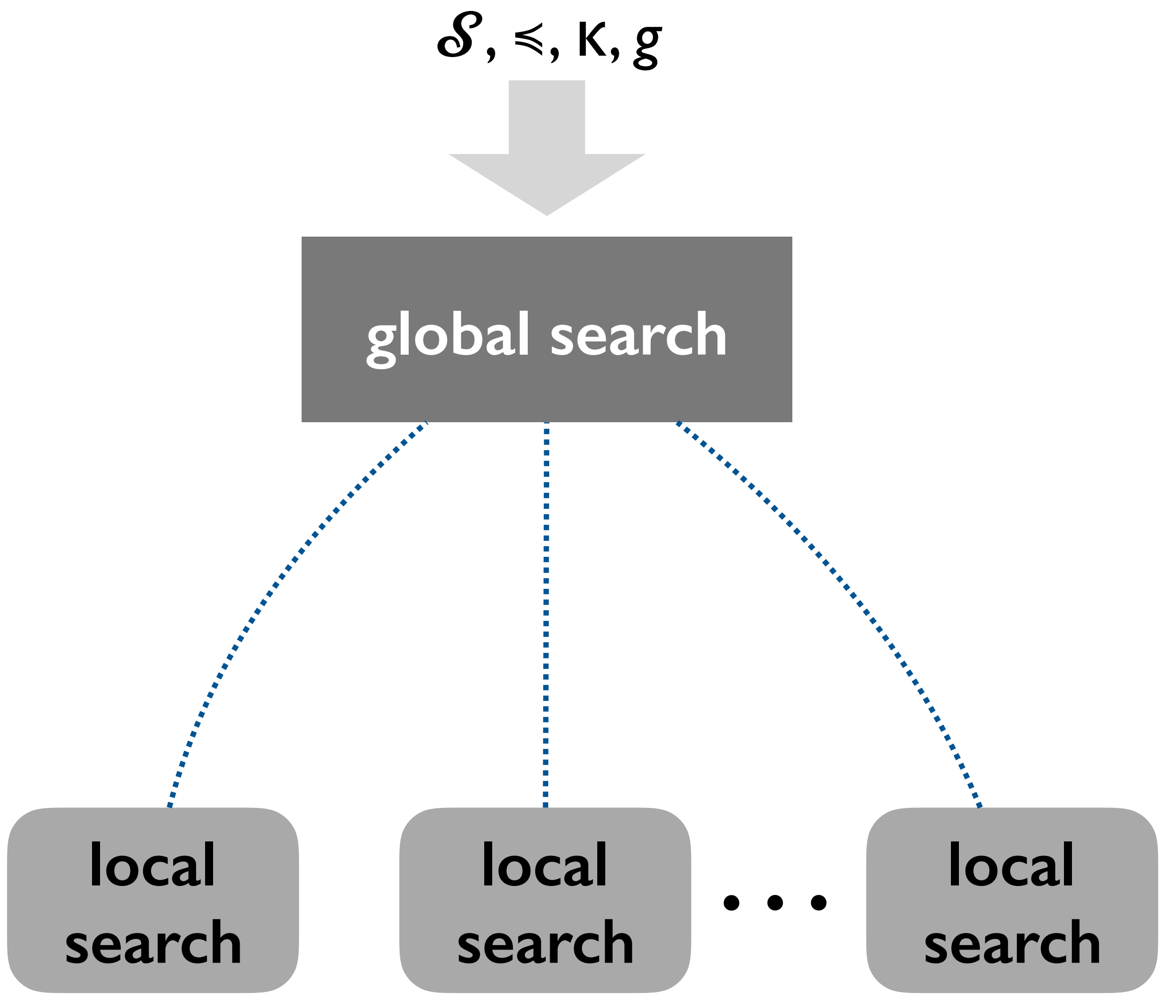
tool

SYNAPSE

basic idea: two cooperating search algorithms

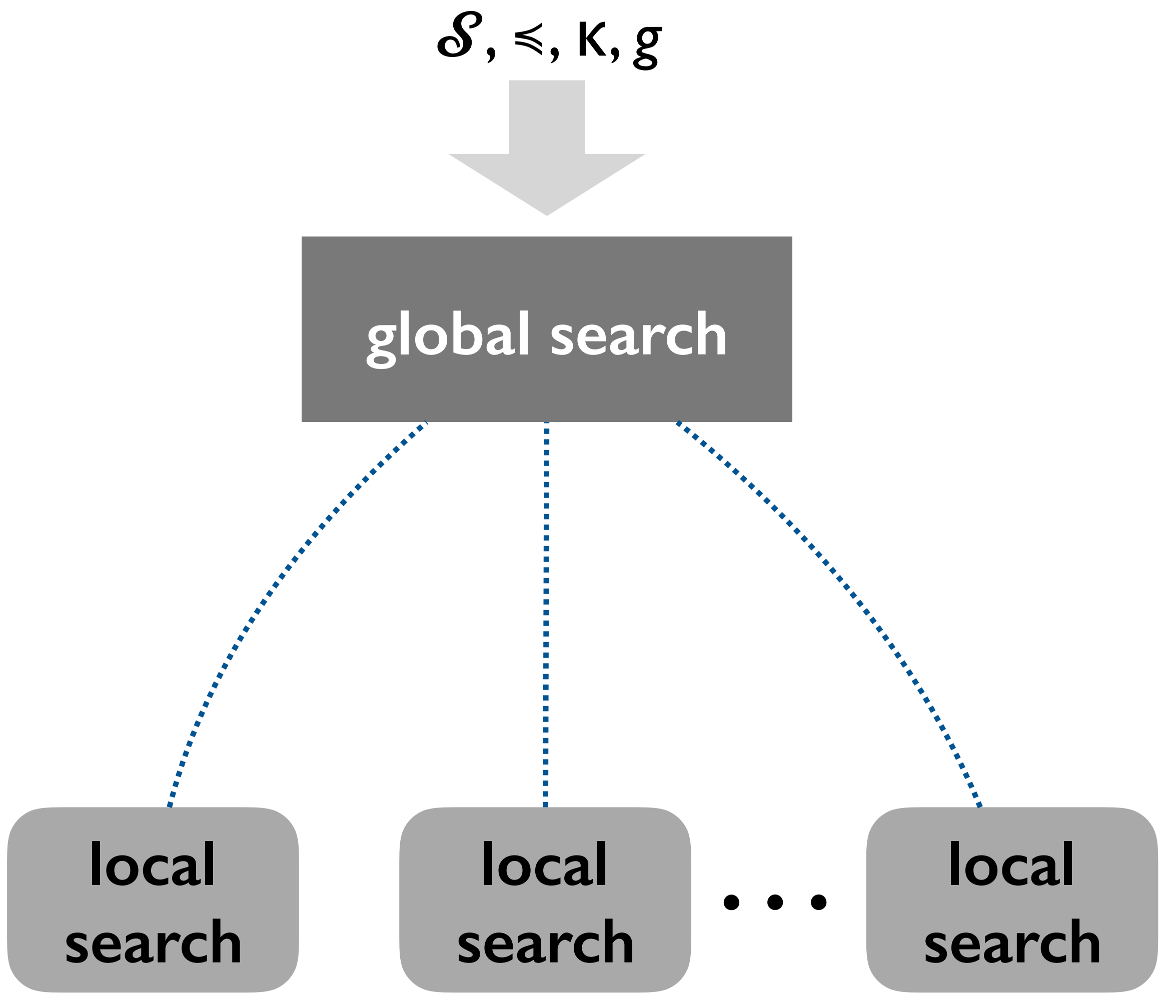


basic idea: two cooperating search algorithms



Global optimizing search coordinates the activities of local searches running in parallel on individual sketches in $\mathcal{S}$.

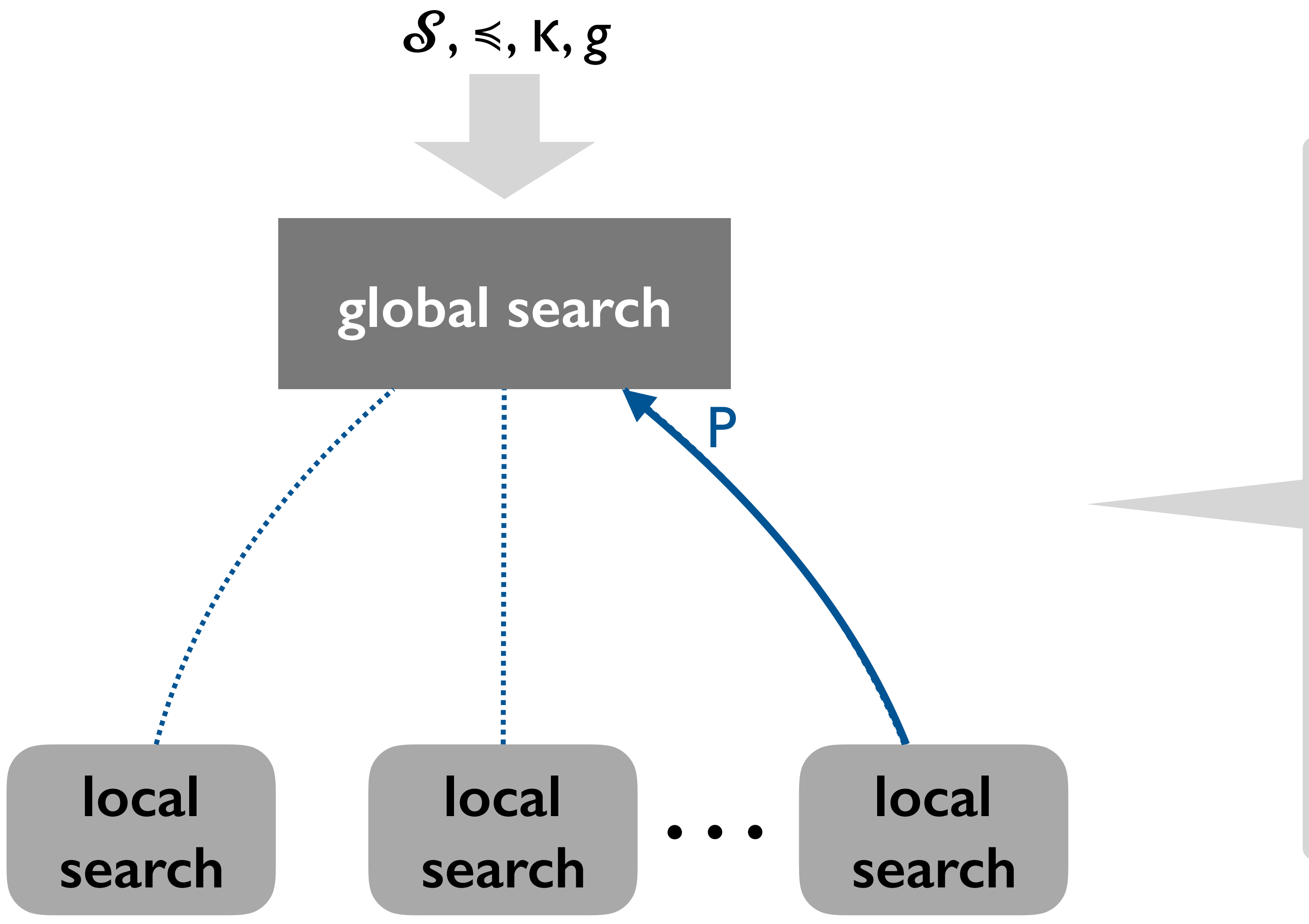
basic idea: two cooperating search algorithms



Global optimizing search coordinates the activities of local searches running in parallel on individual sketches in $\mathcal{S}$.

Local combinatorial search employs an incremental form of CEGIS to incorporate the information sent by the global search.

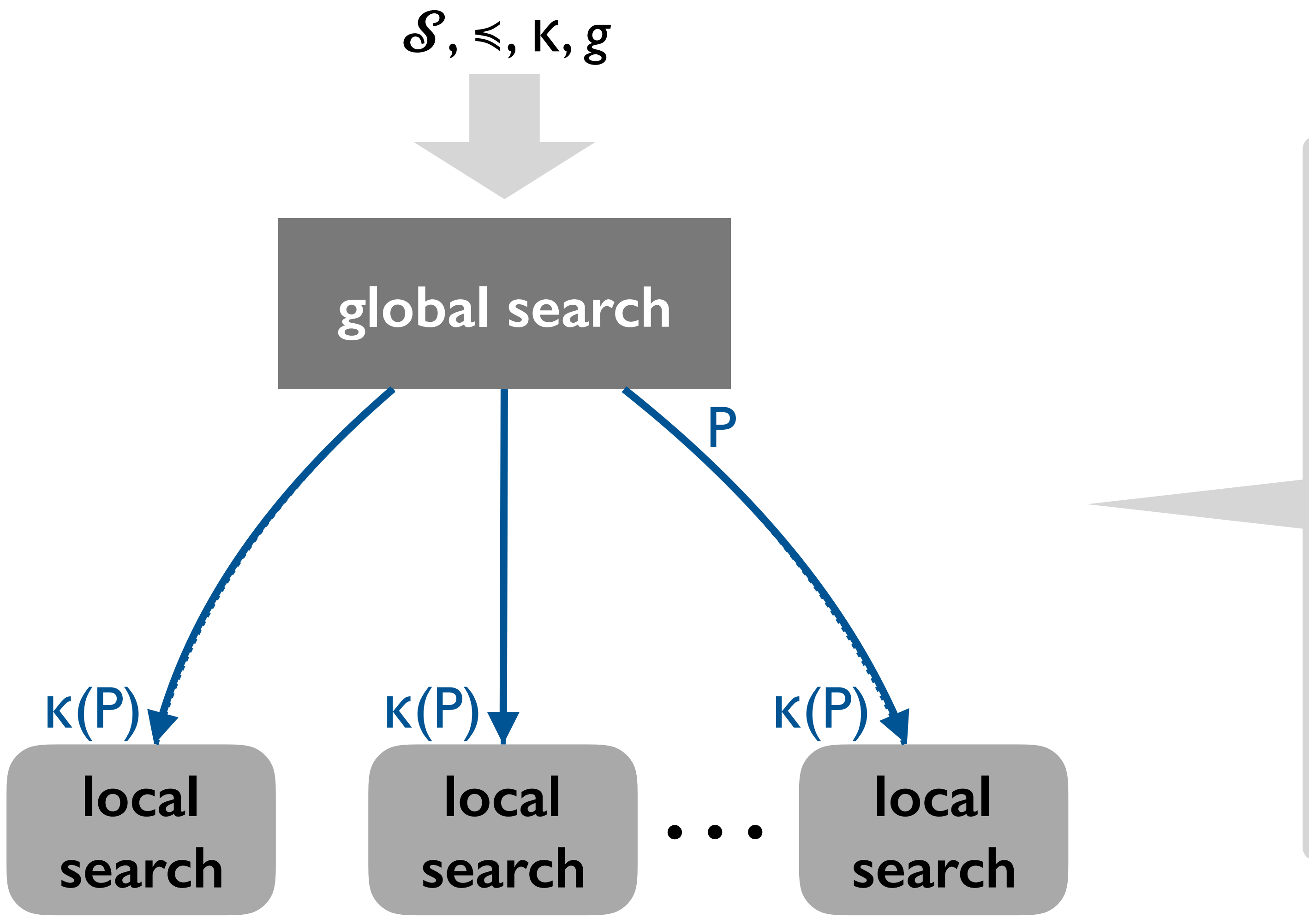
basic idea: two cooperating search algorithms



Global optimizing search coordinates the activities of local searches running in parallel on individual sketches in $\mathcal{S}$.

Local combinatorial search employs an incremental form of CEGIS to incorporate the information sent by the global search.

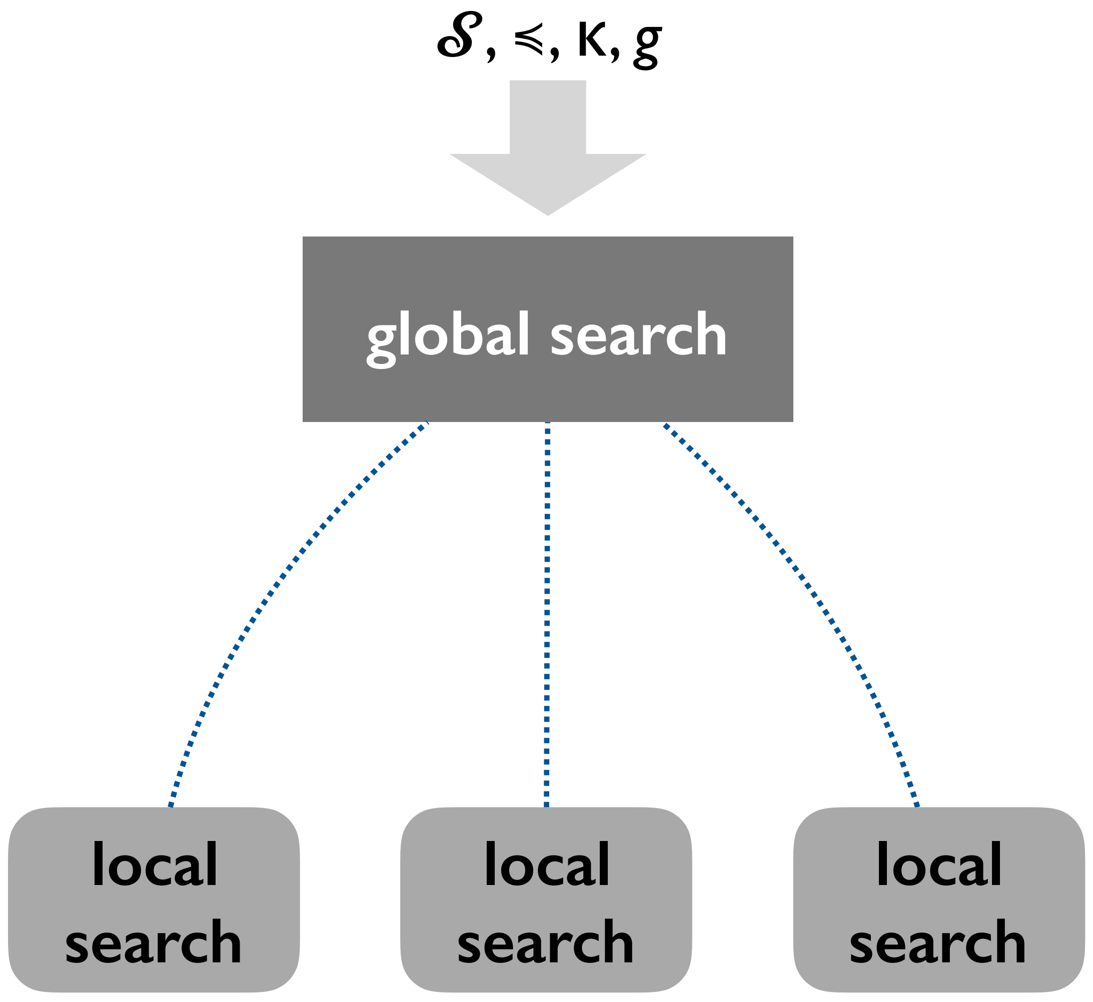
basic idea: two cooperating search algorithms

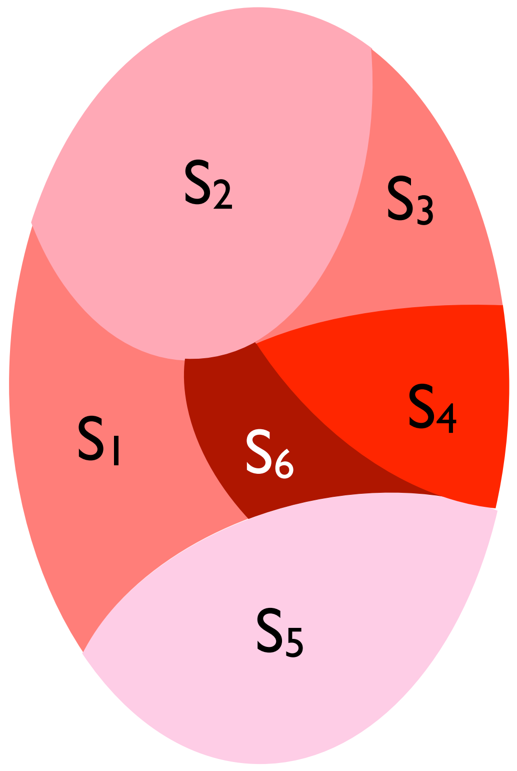


Global optimizing search coordinates the activities of local searches running in parallel on individual sketches in $\mathcal{S}$.

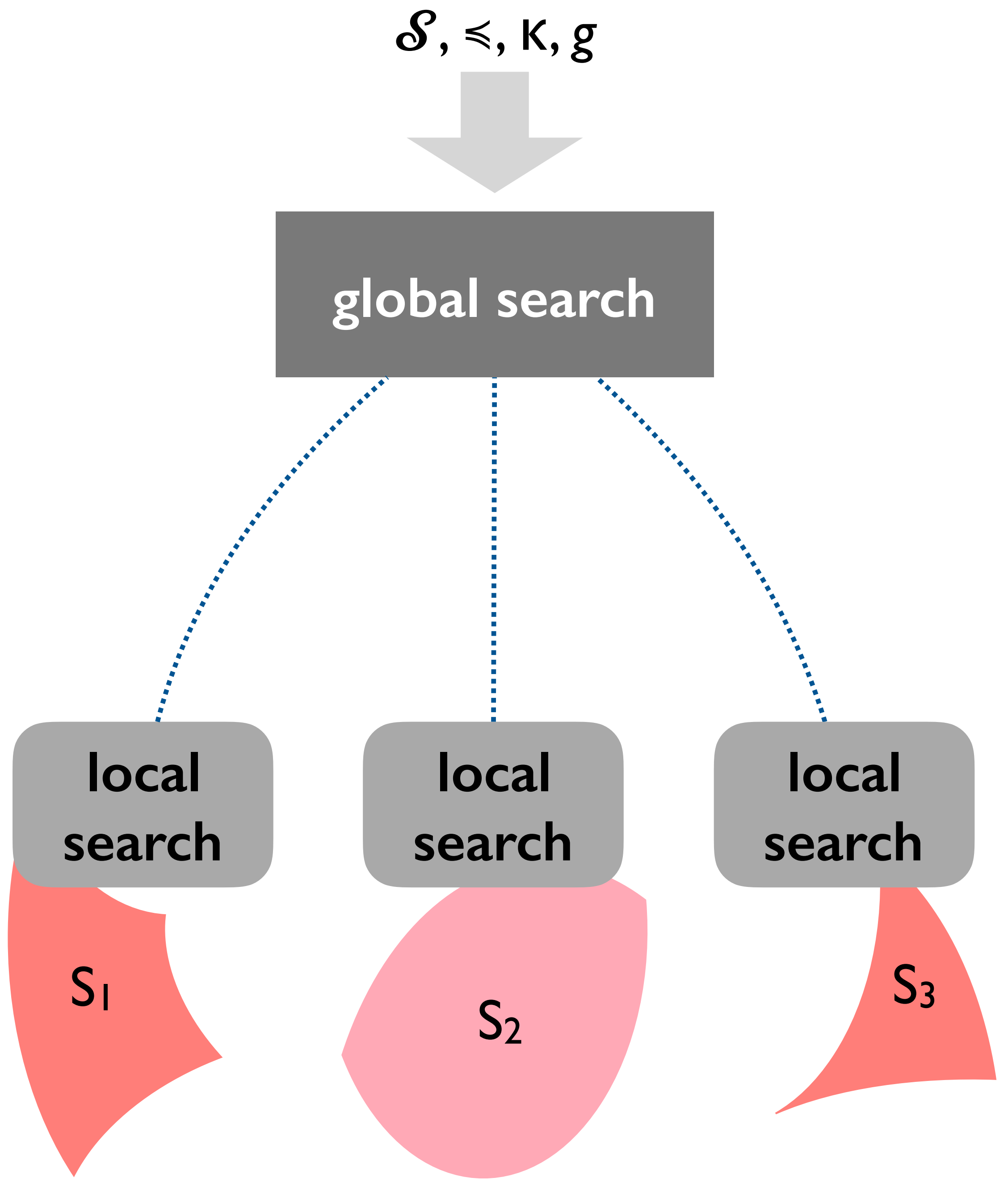
Local combinatorial search employs an incremental form of CEGIS to incorporate the information sent by the global search.

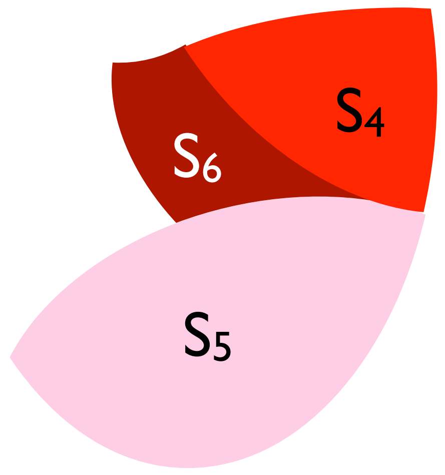
synapse by example



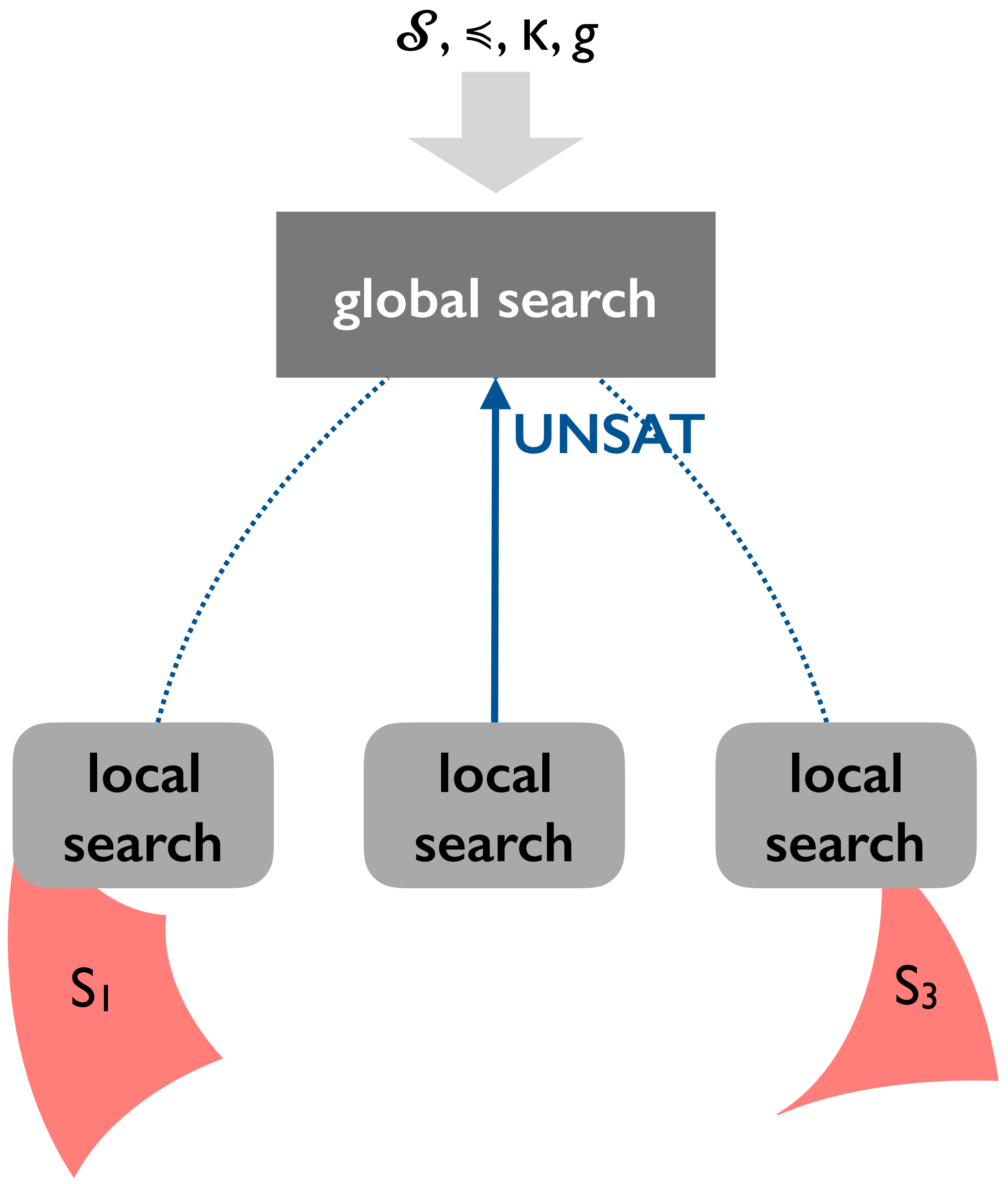


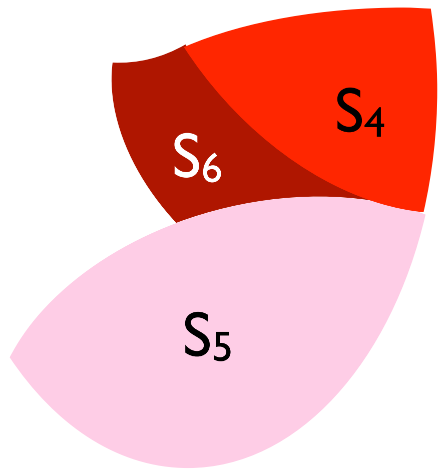
synapse by example



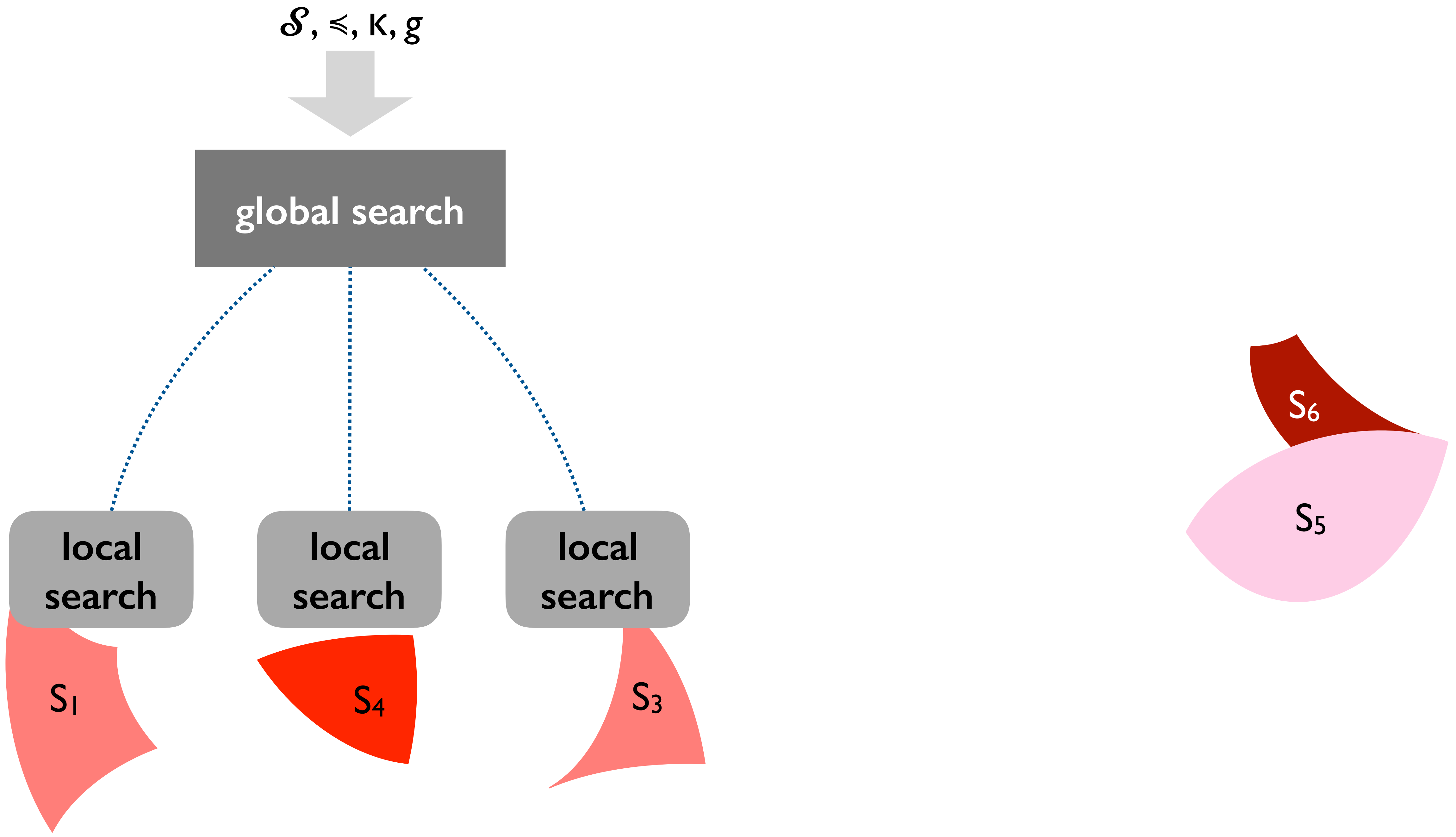


synapse by example

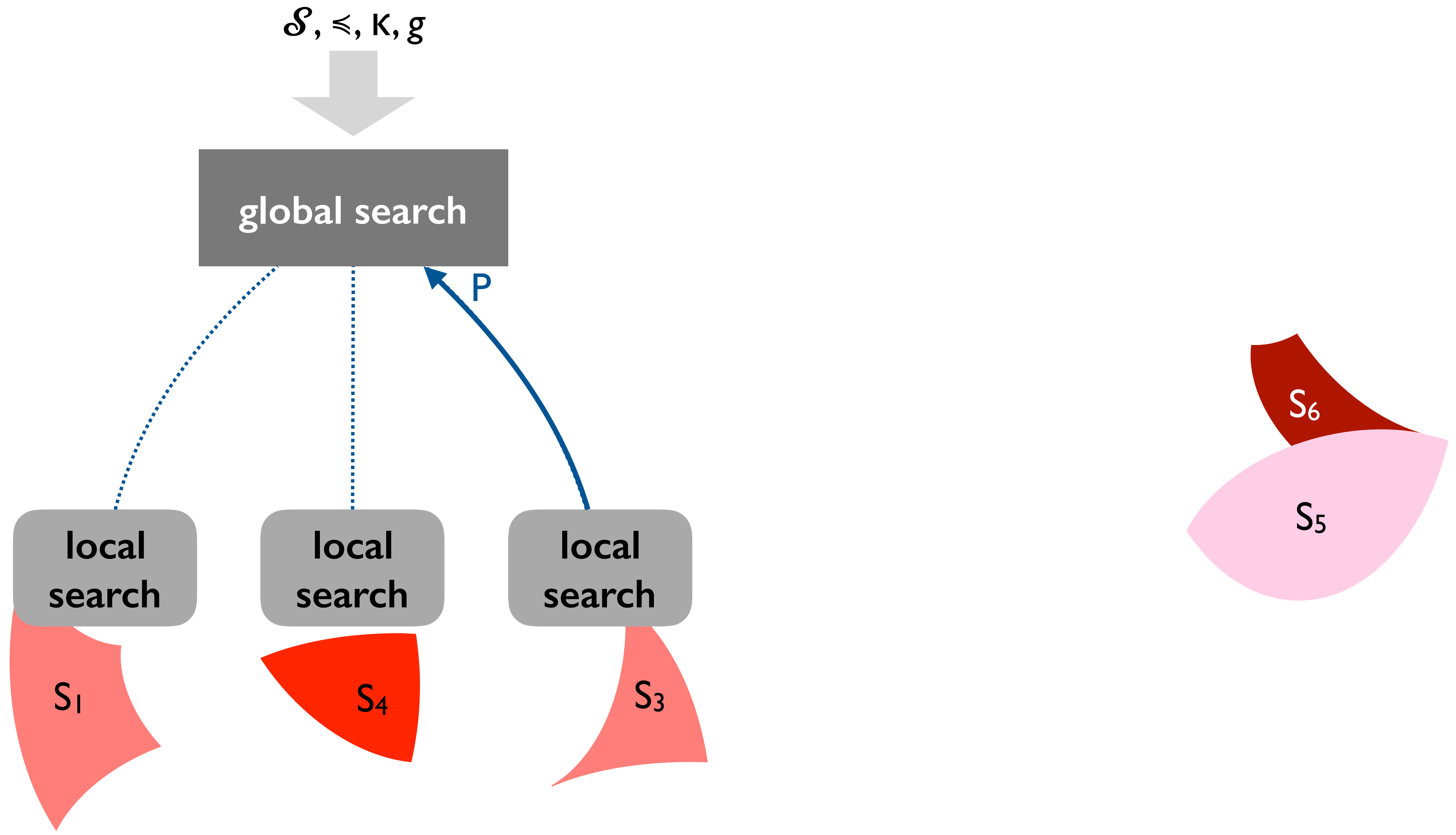




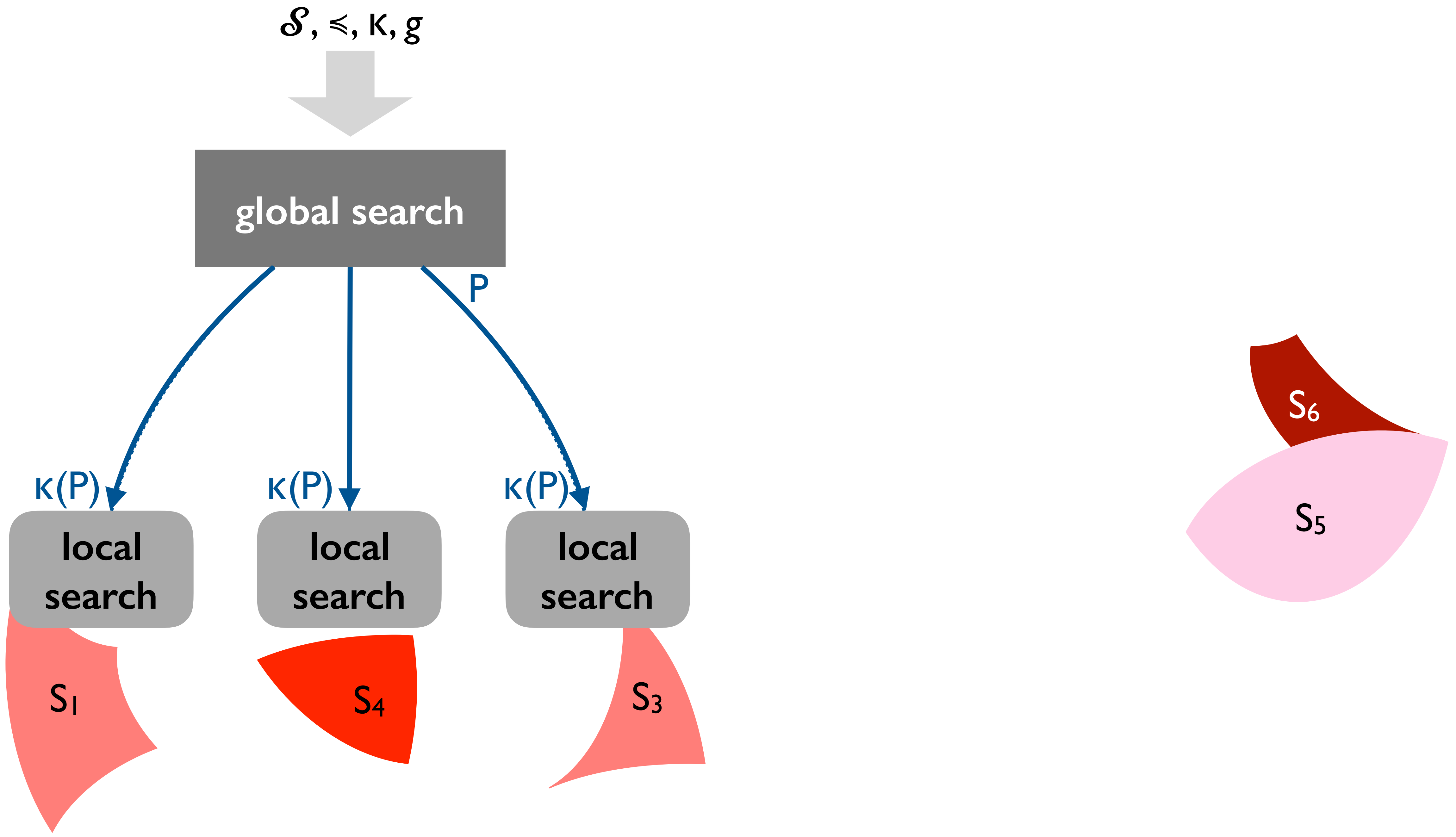
synapse by example



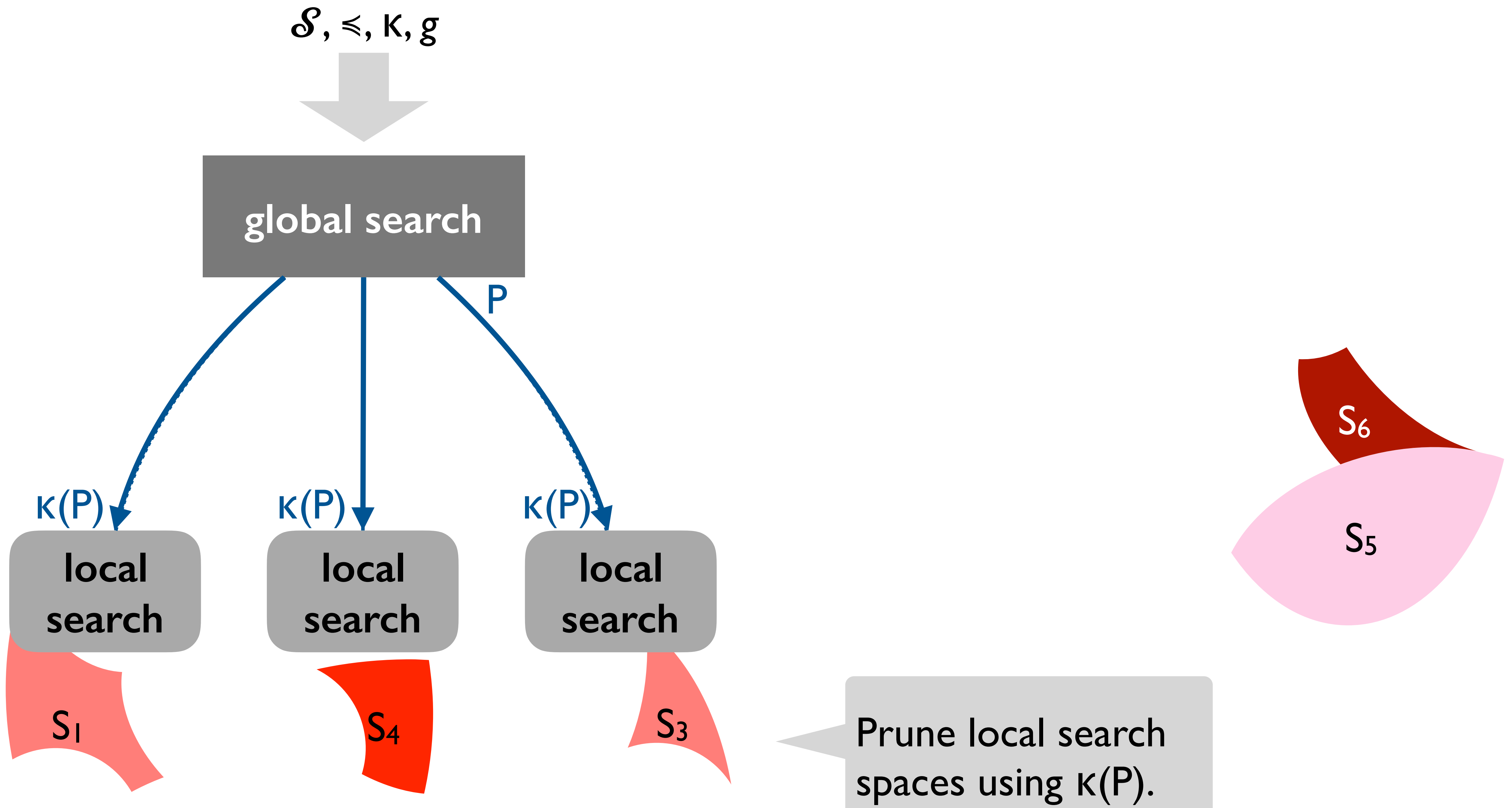
synapse by example



synapse by example

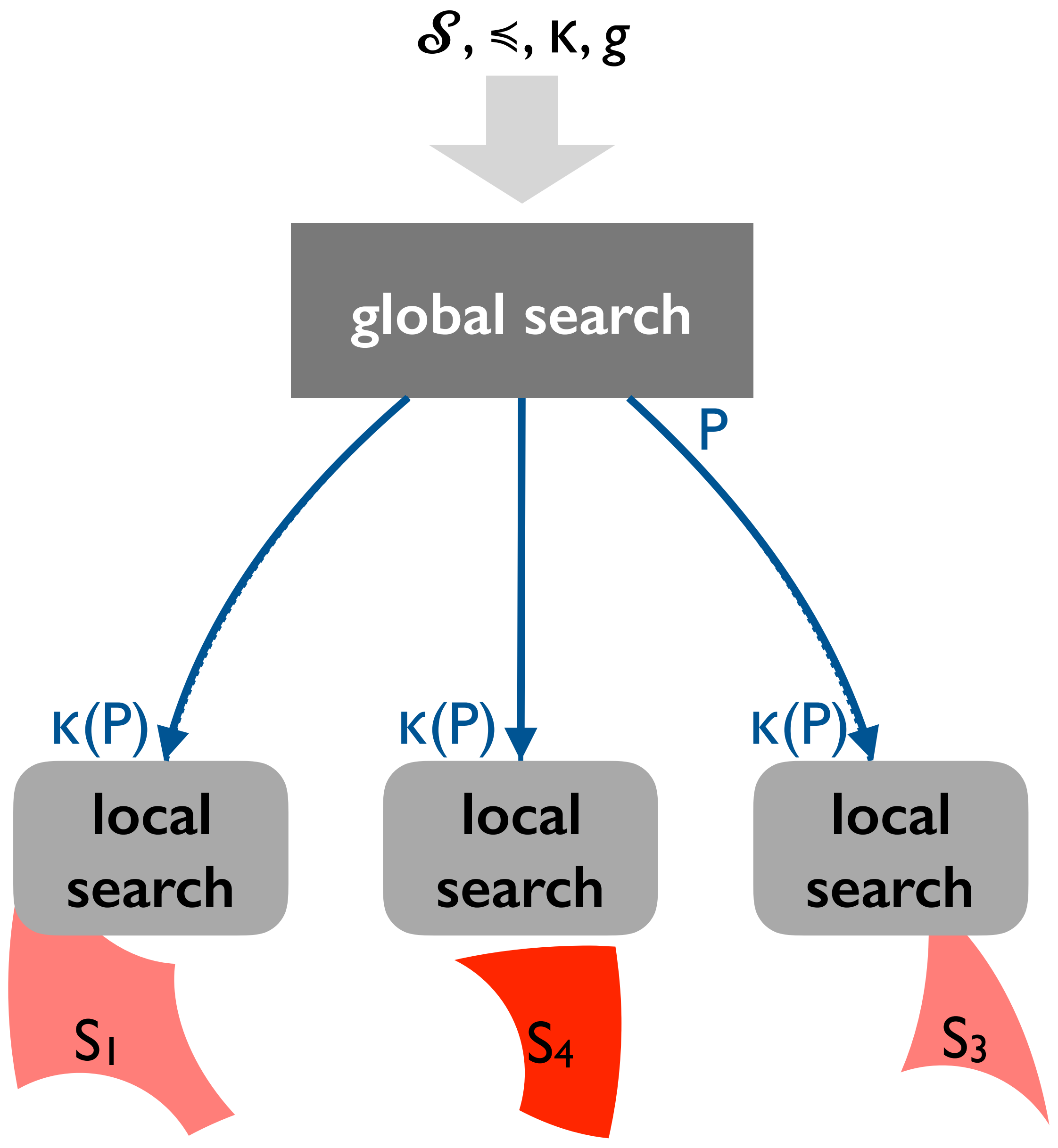


synapse by example



Prune local search spaces using κ(P).

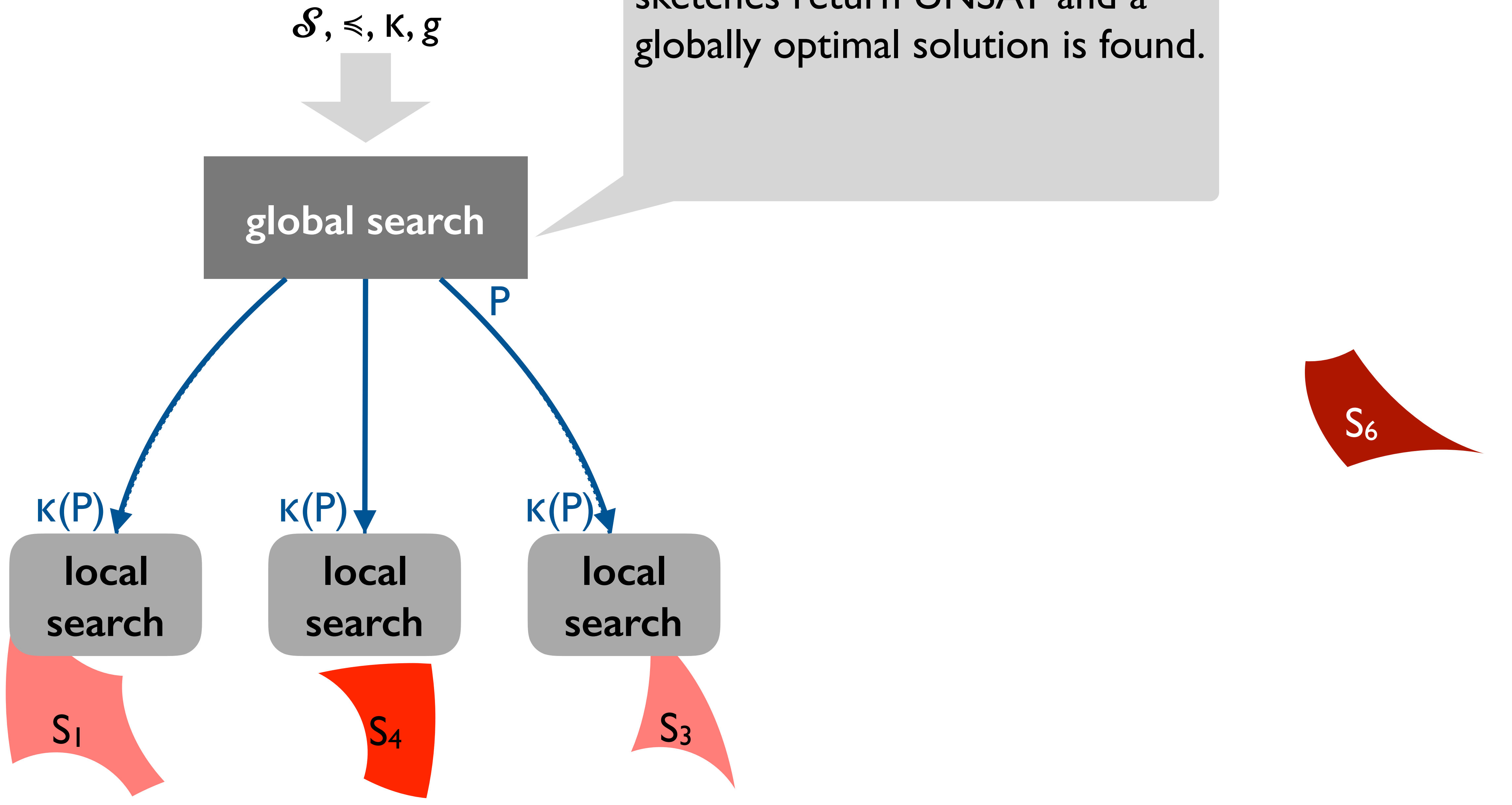
synapse by example



Prune the global search space to g(κ(P)).

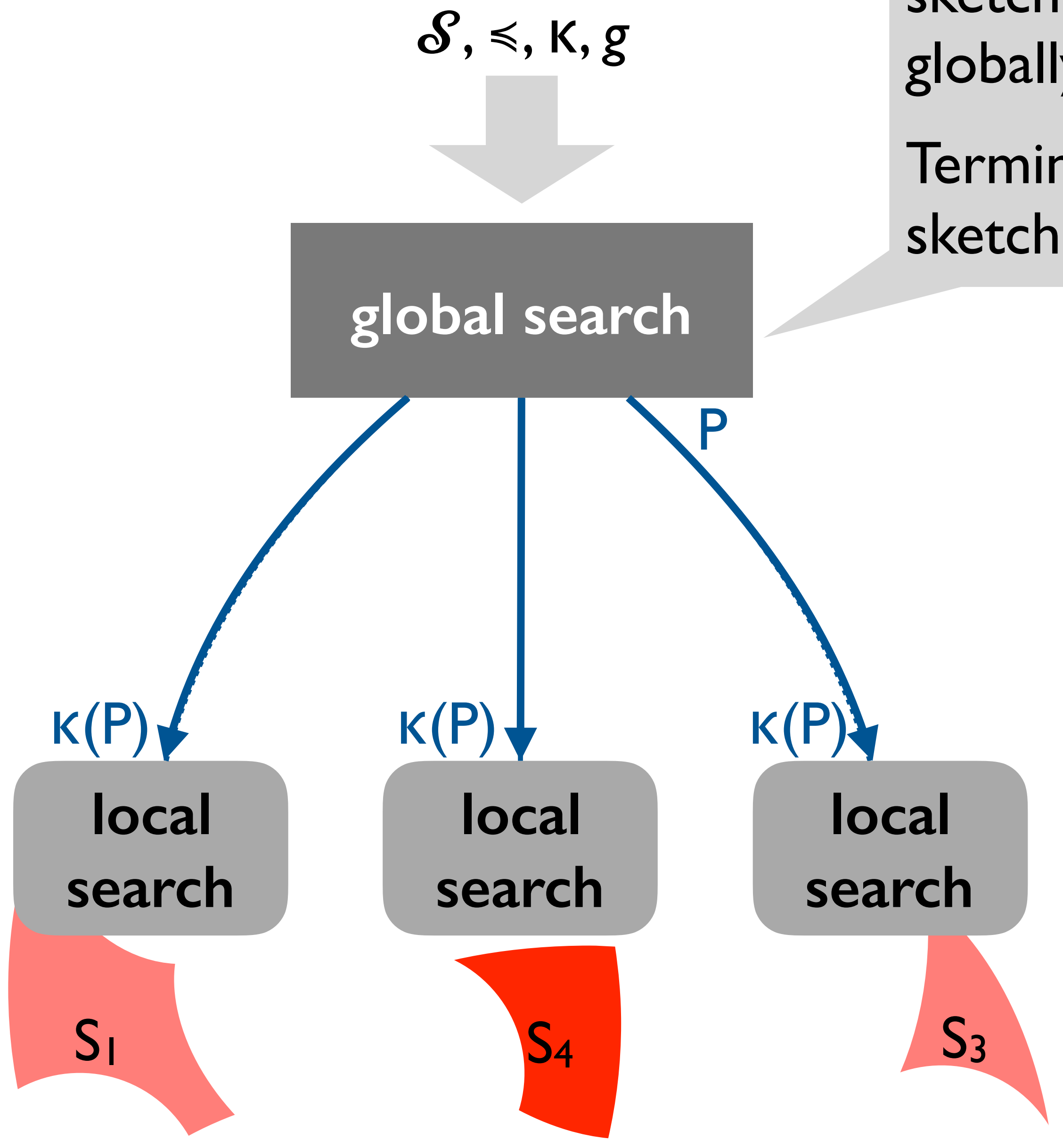
Prune local search spaces using κ(P).

synapse by example



Continues until all remaining sketches return UNSAT and a globally optimal solution is found.

synapse by example



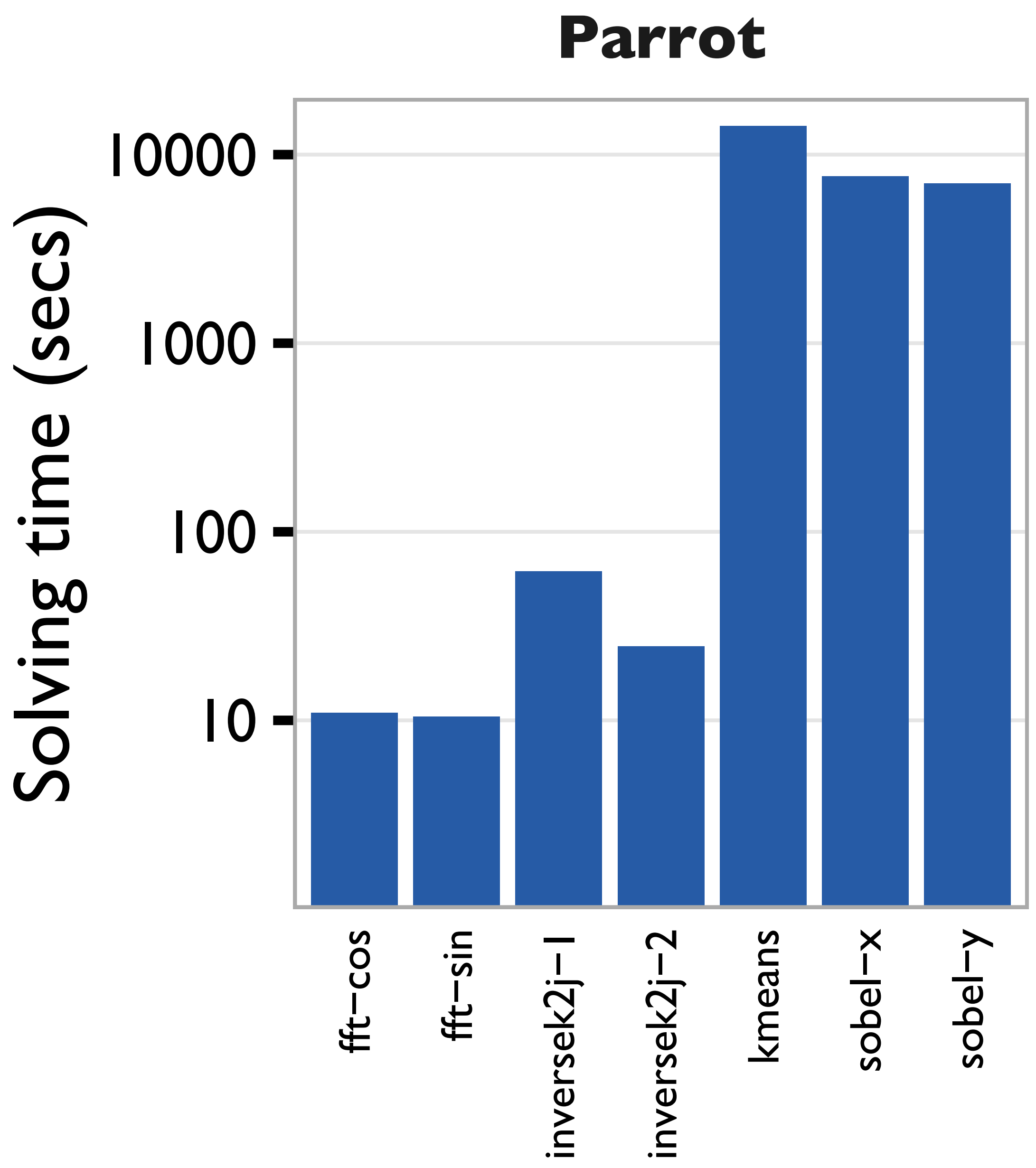
Continues until all remaining sketches return UNSAT and a globally optimal solution is found.

Terminates if $g(c)$ is a finite set of sketches for all c.

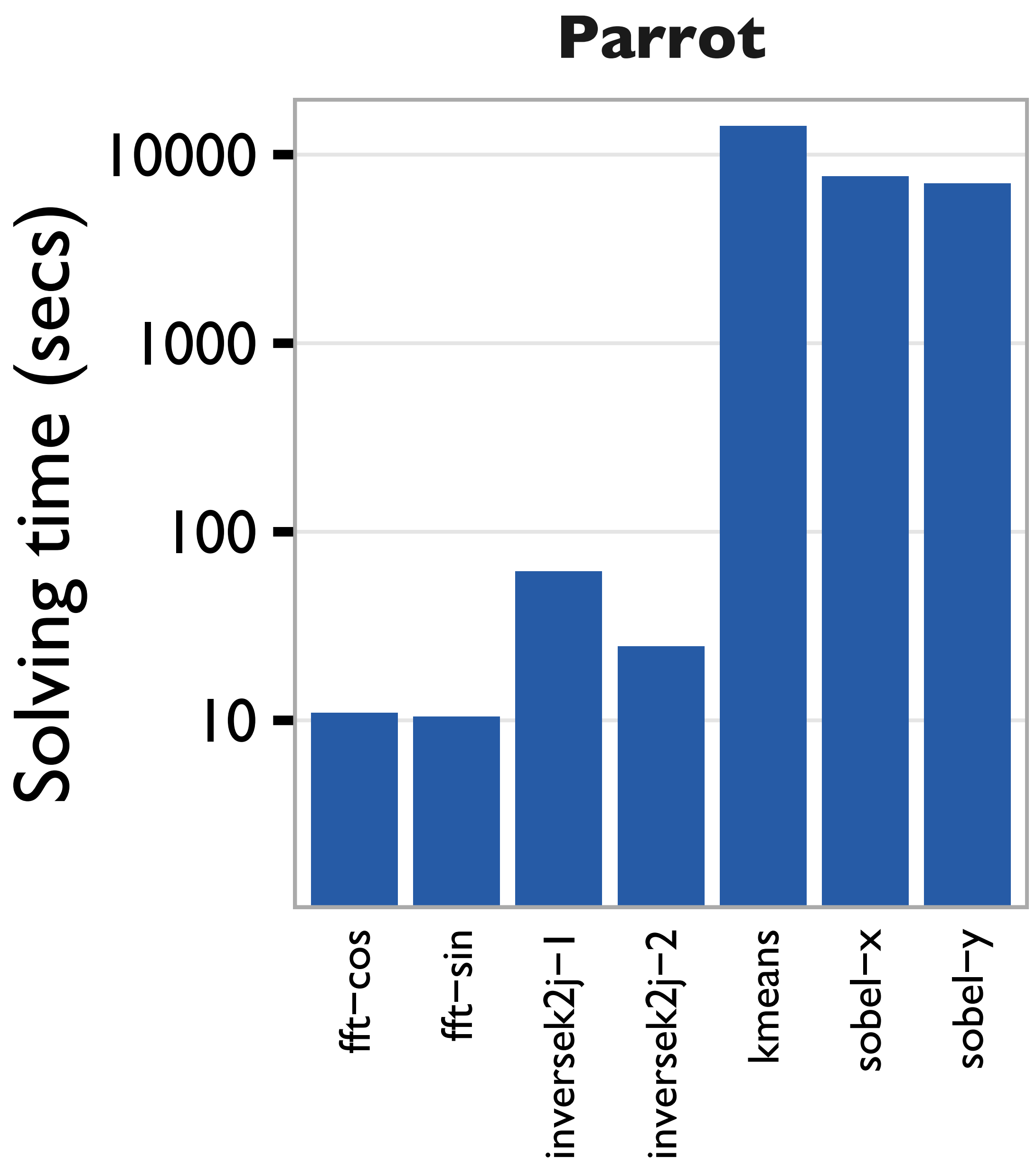
results

evaluation

synapse can solve new classes of problems

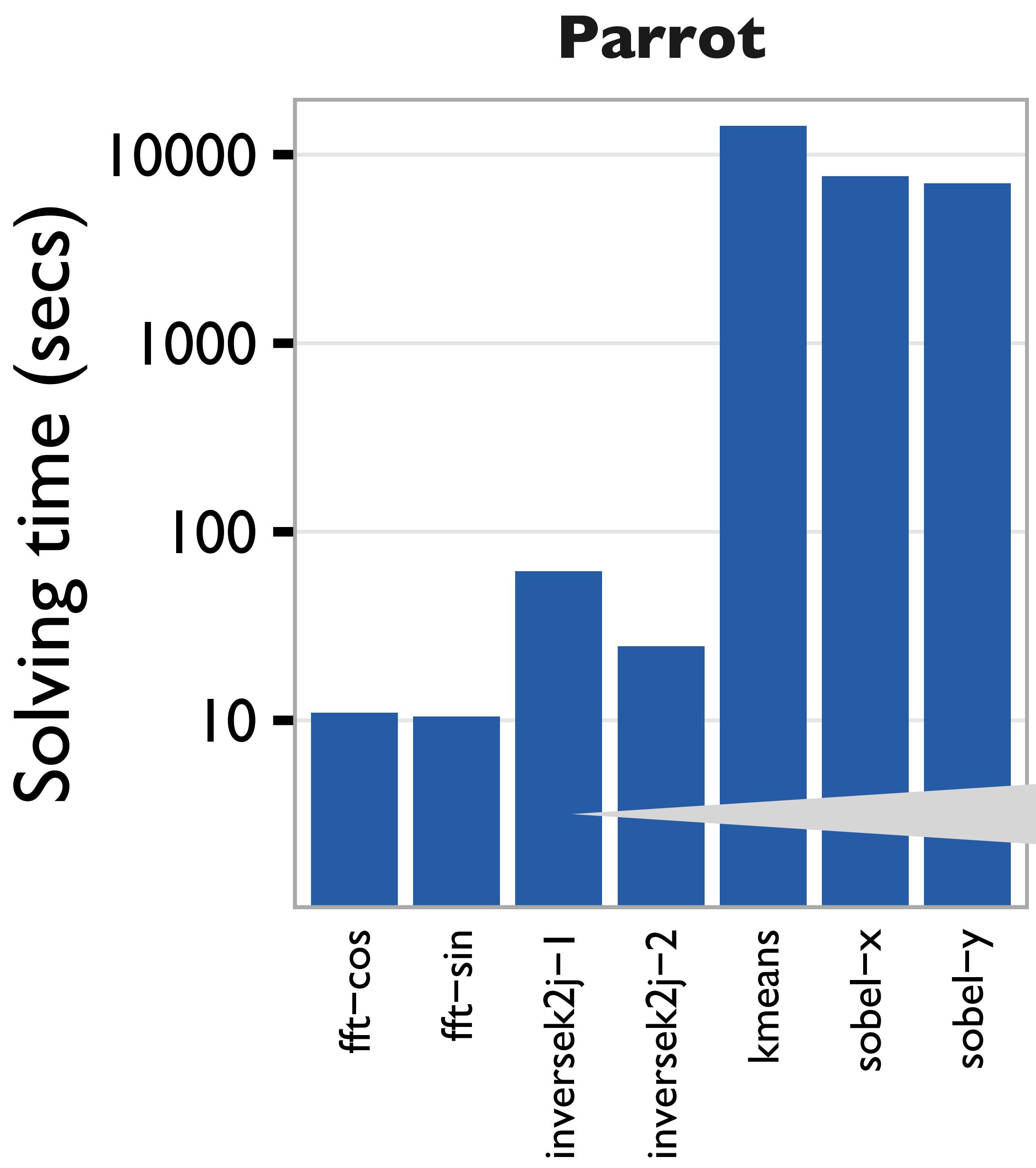


synapse can solve new classes of problems



Finds the cheapest program that approximates a given reference program with respect to an application-specific error bound.

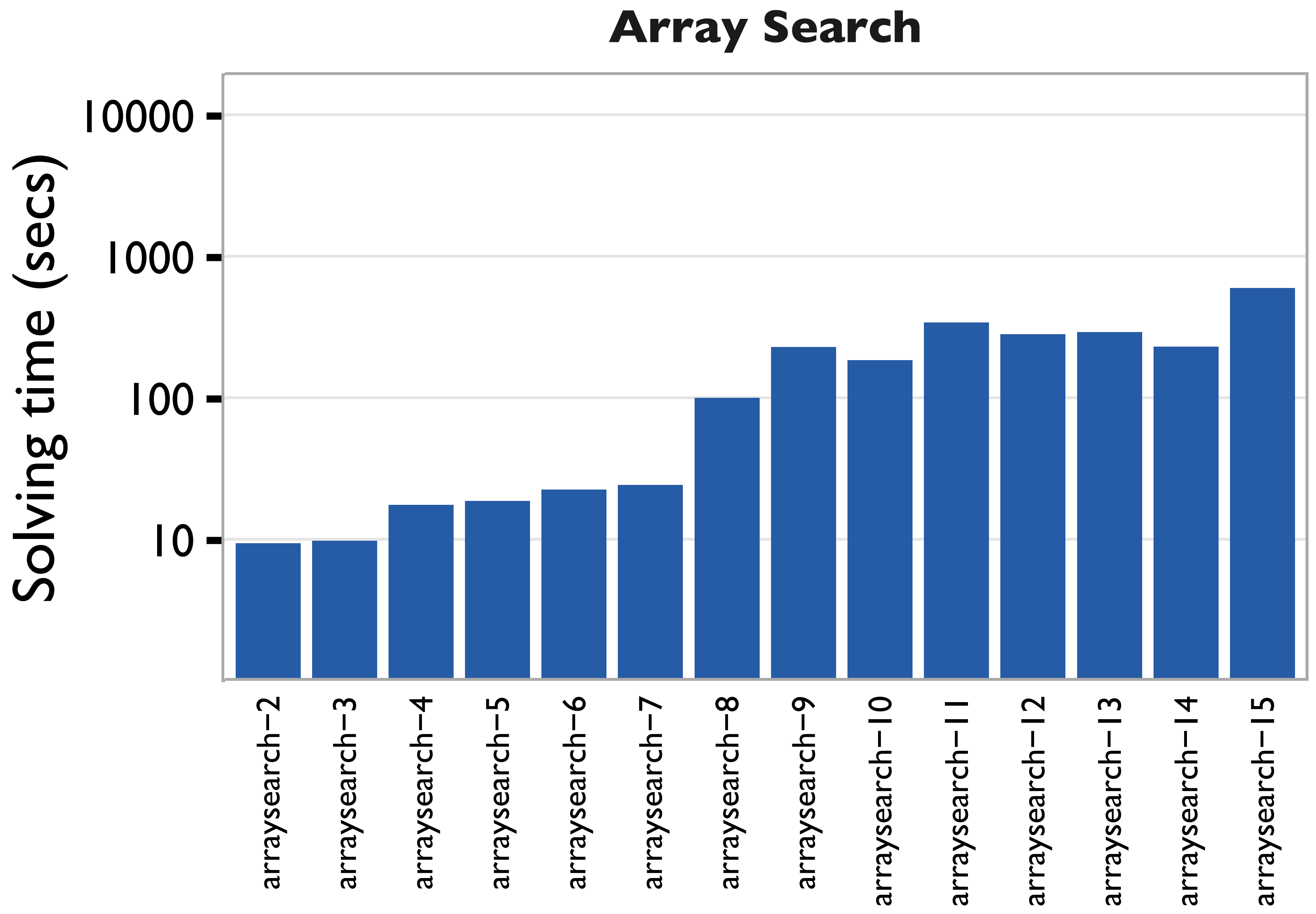
synapse can solve new classes of problems



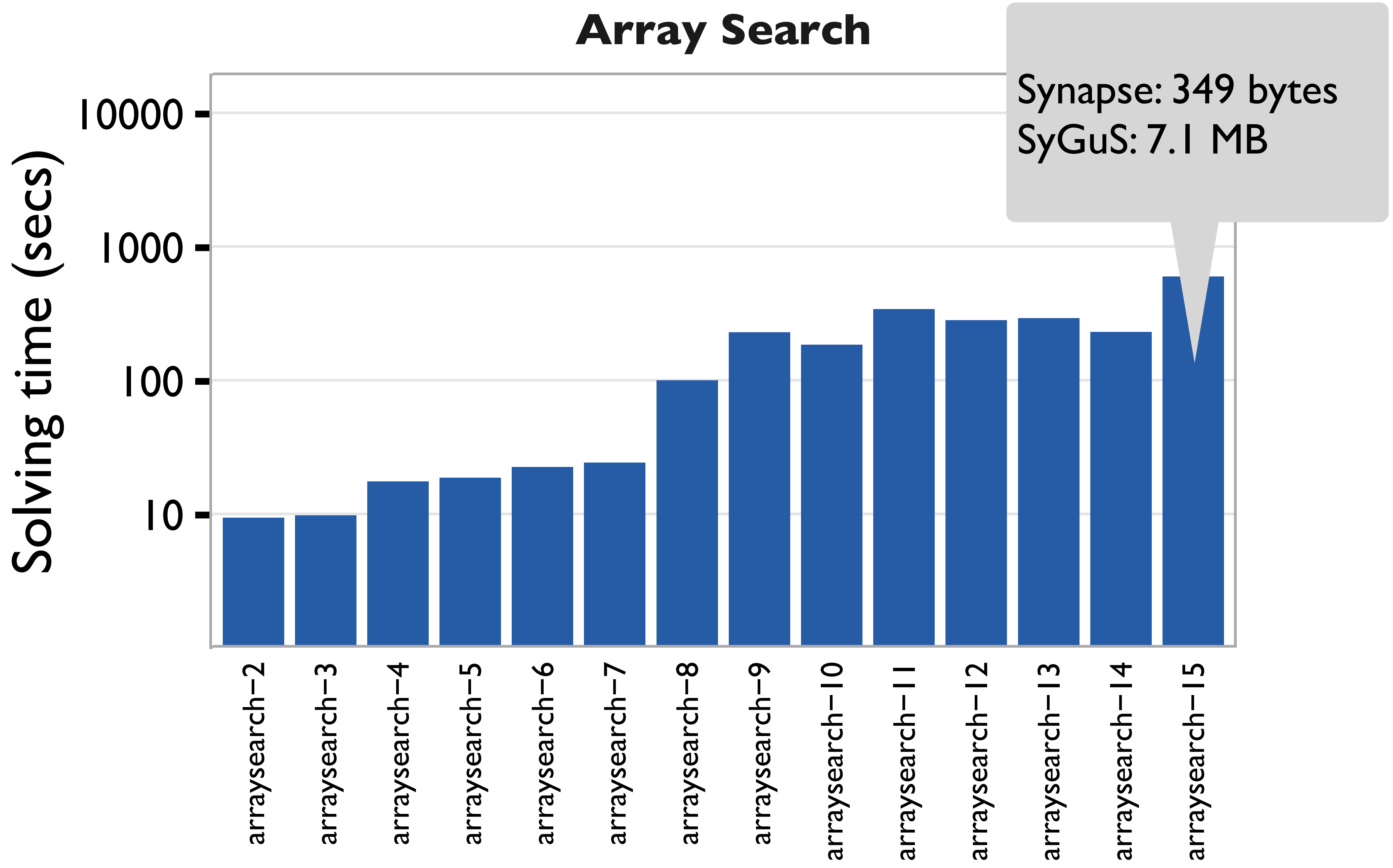
Finds the cheapest program that approximates a given reference program with respect to an application-specific error bound.

```
def inversek2j(float x, float y):
  th2 = acos(((x*x) + (y*y) - 0.5) / 0.5)
  th1 = asin((y * (0.5 + 0.5*cos(*th2)) -
              0.5*x*sin(*th2)) /
              (x*x + y*y))
  return th1
```

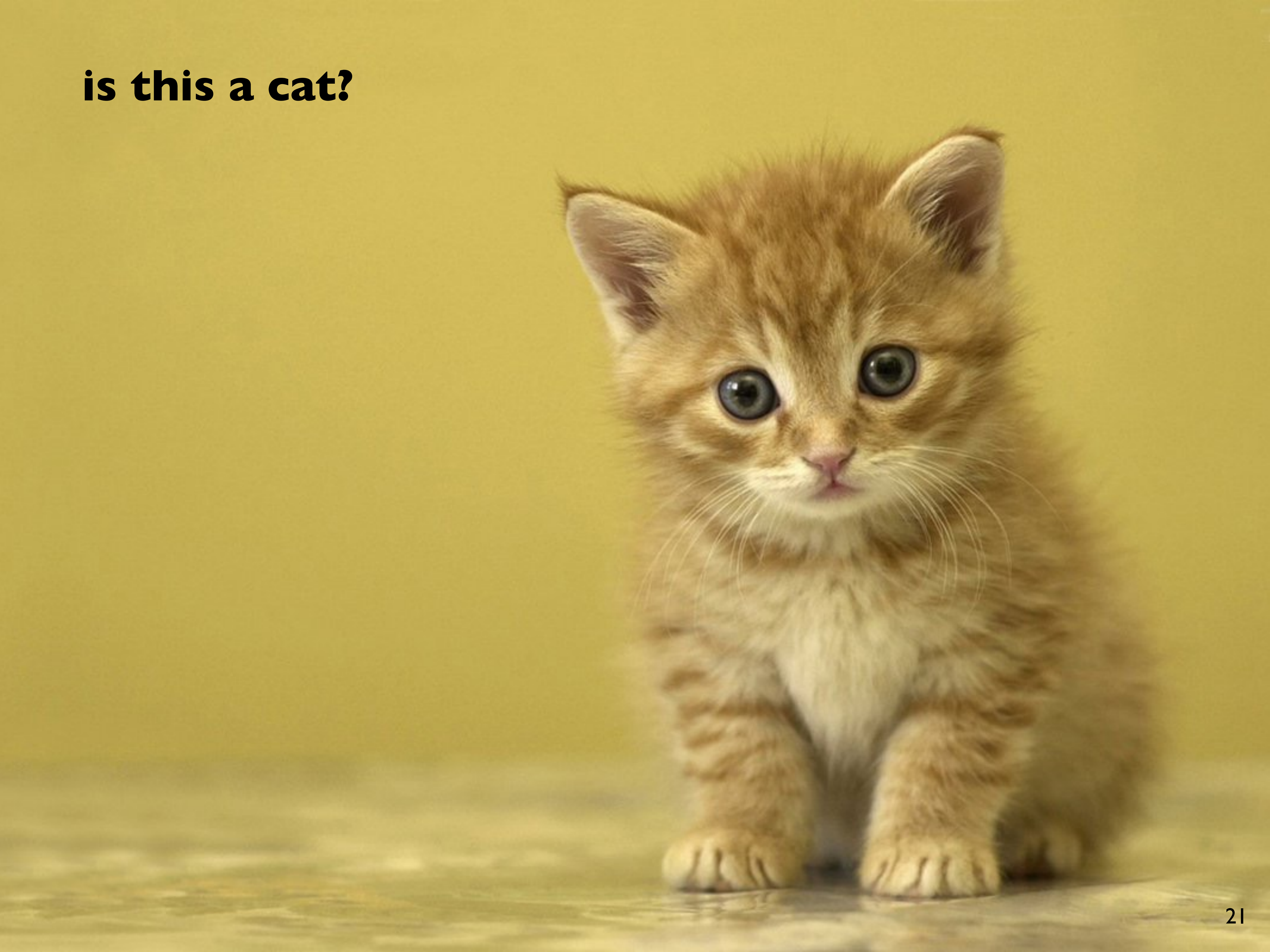
synapse solves standard benchmarks optimally



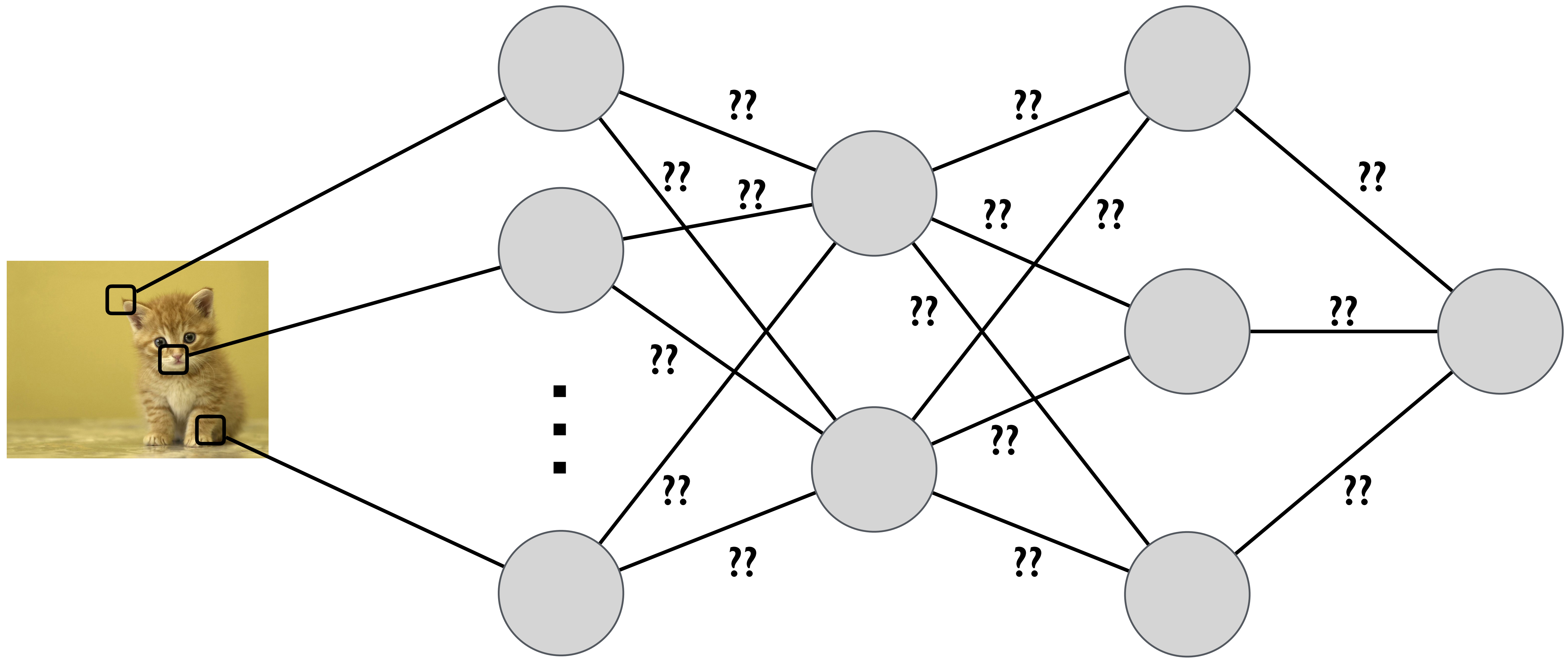
synapse solves standard benchmarks optimally



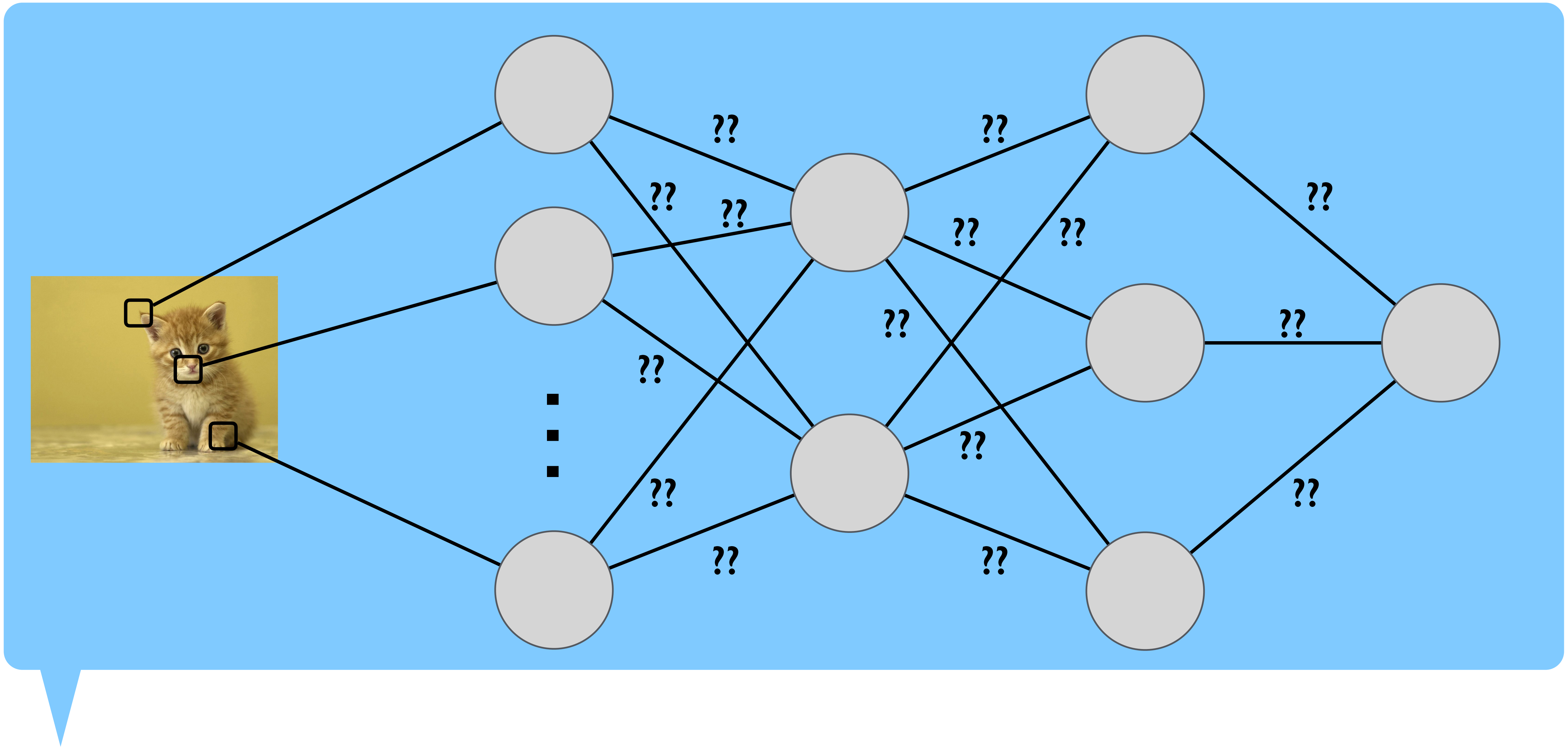
is this a cat?



synapse can reason about complex costs



synapse can reason about complex costs



$\mathcal{S}, \preccurlyeq$: a finite set of network topologies

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

$g(c) = \mathcal{S}$

thanks!

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

http://synapse.uwplse.org

is this a cat?

yes!