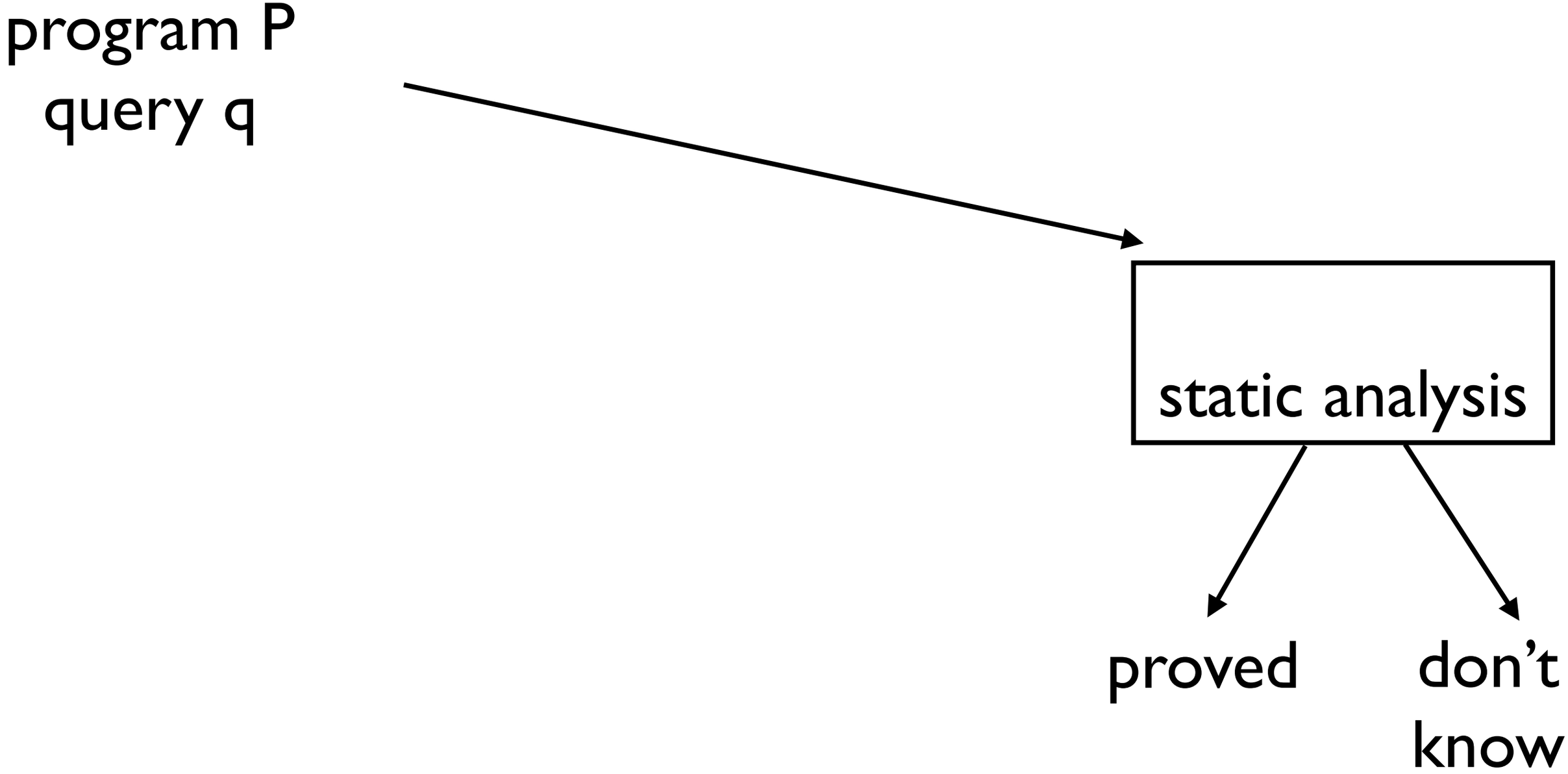
Abstractions from Tests

Mayur Naik (Georgia Institute of Technology)
Hongseok Yang (University of Oxford)
Ghila Castelnuovo (Tel-Aviv University)
Mooly Sagiv (Tel-Aviv University)

Motivation

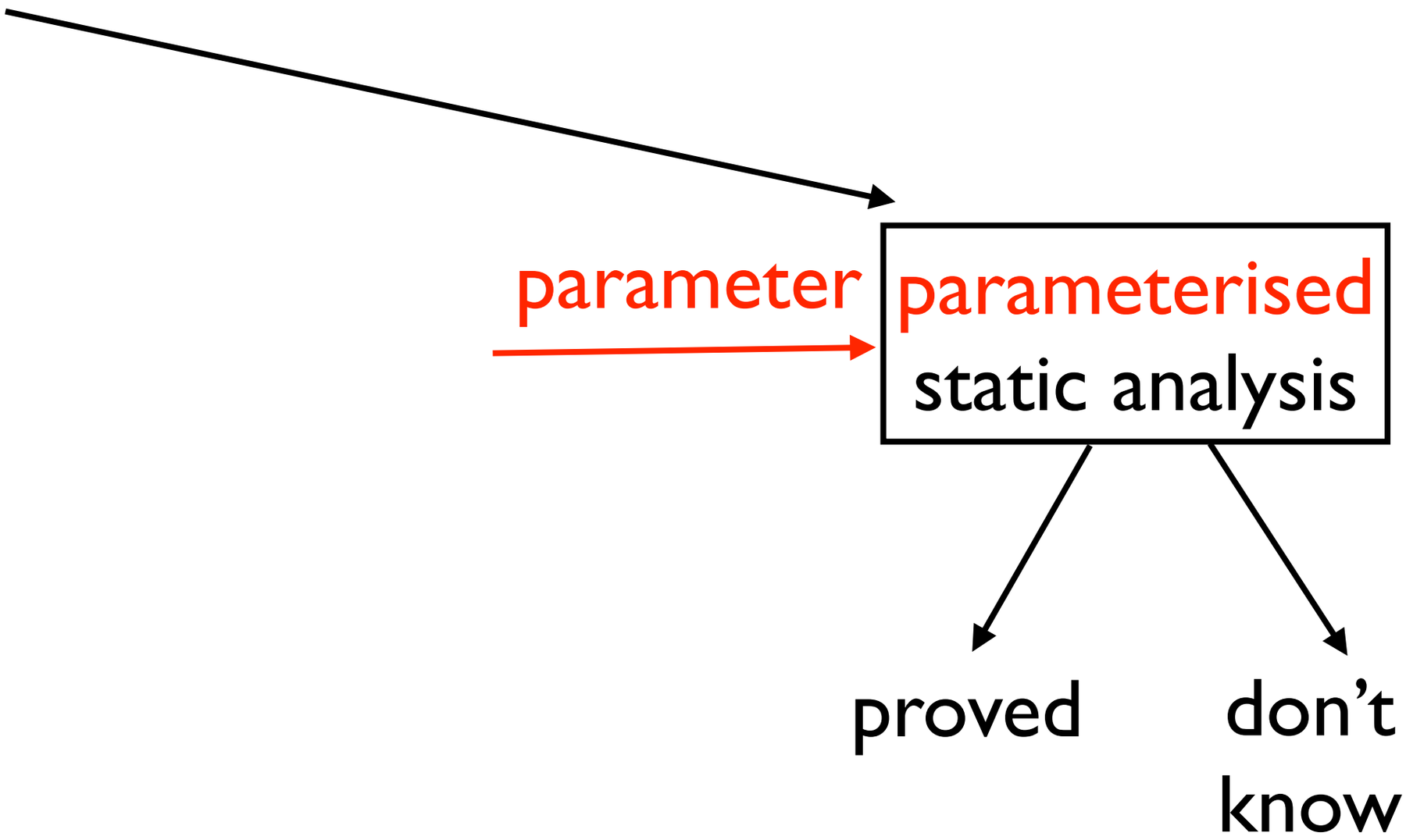
- Great success stories in automatic program verification based on static analysis techniques (SDV, Astree, etc).
- Yet balancing precision and performance of a static analysis is still an art.
- We want to do this balancing automatically.

Typical static analysis

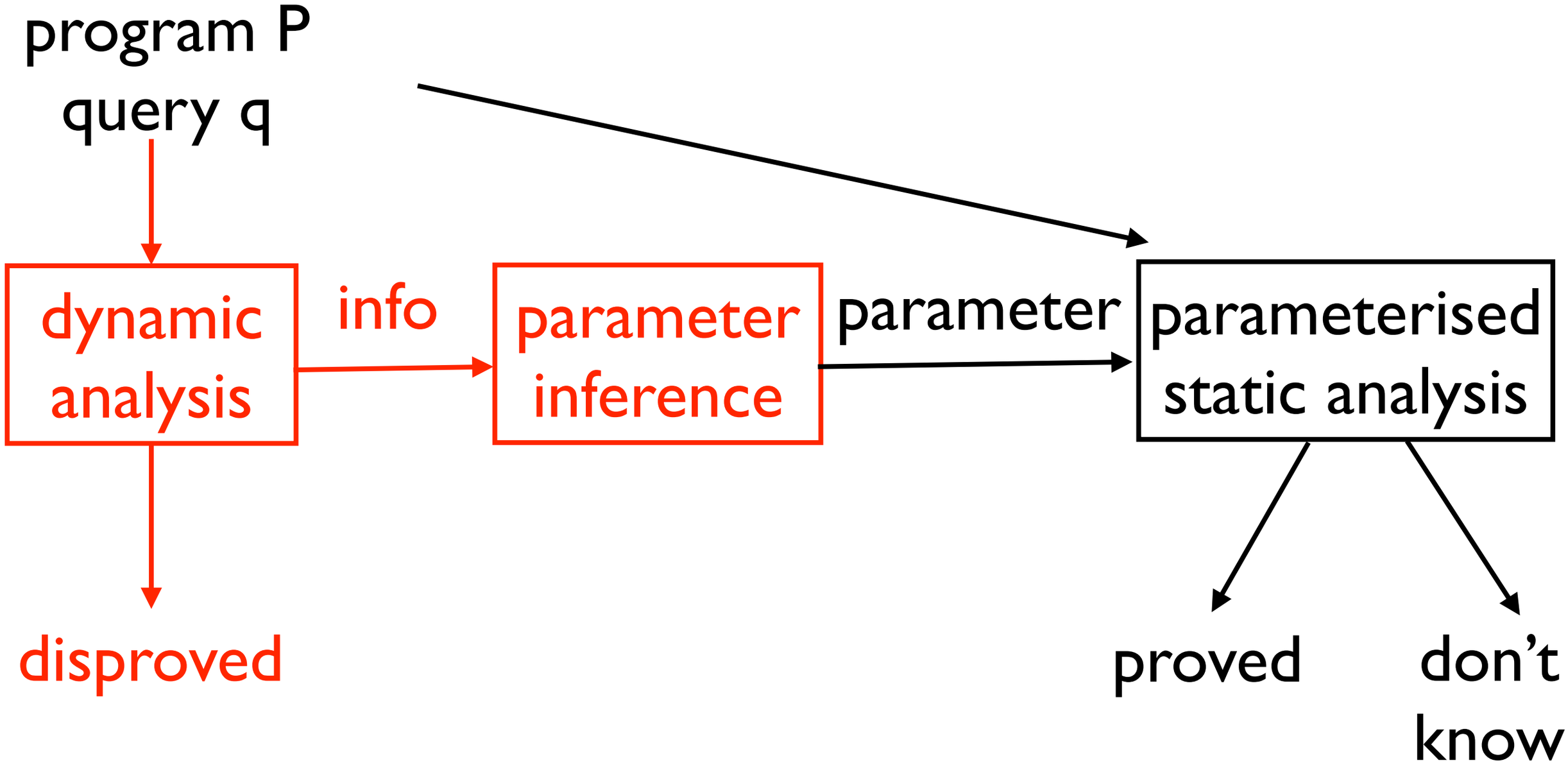


Our approach

program P
query q



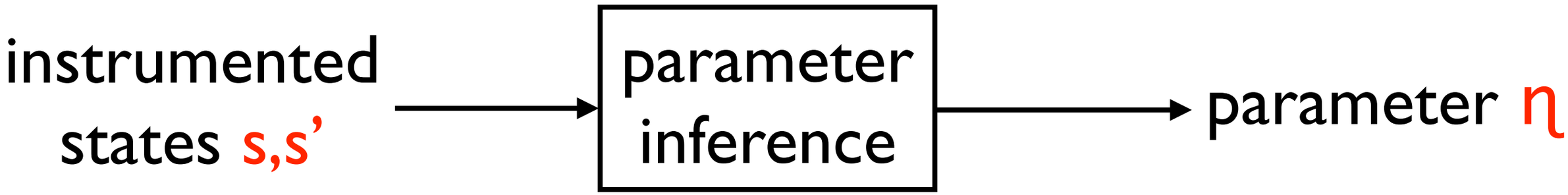
Our approach

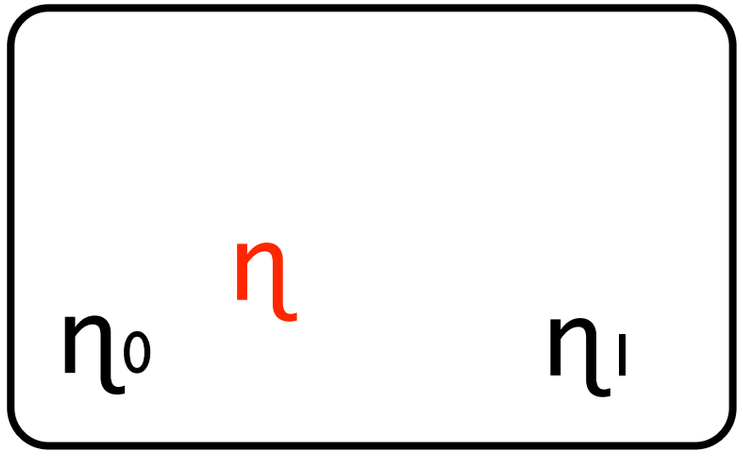


Hypothesis

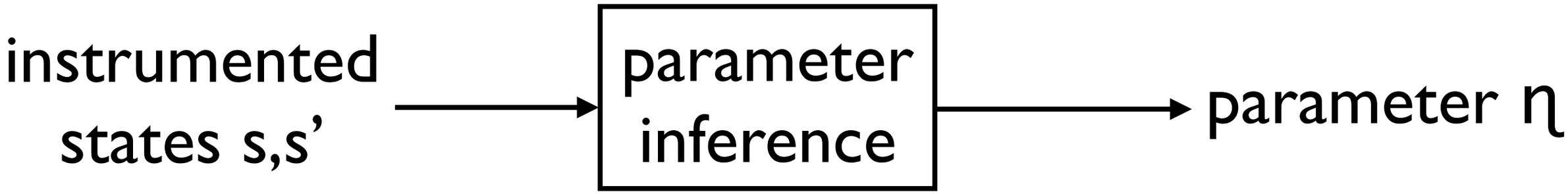
- If a query is simple, we can find why the query holds simply by looking at a few execution traces.

Parameter inference based on separability and minimality



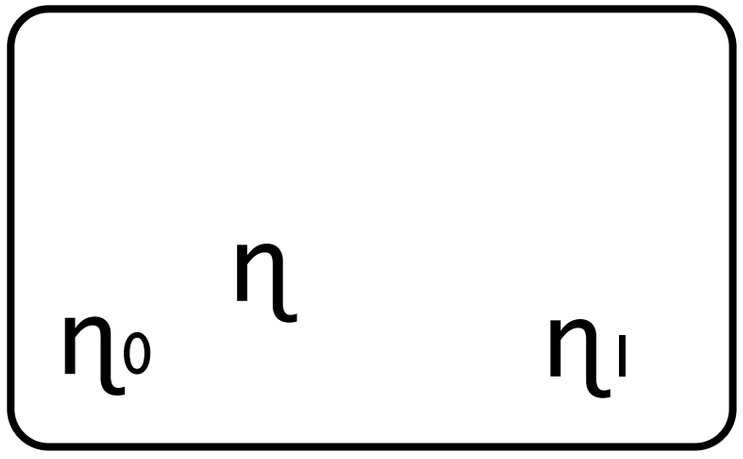


Parameter inference based on separability and minimality

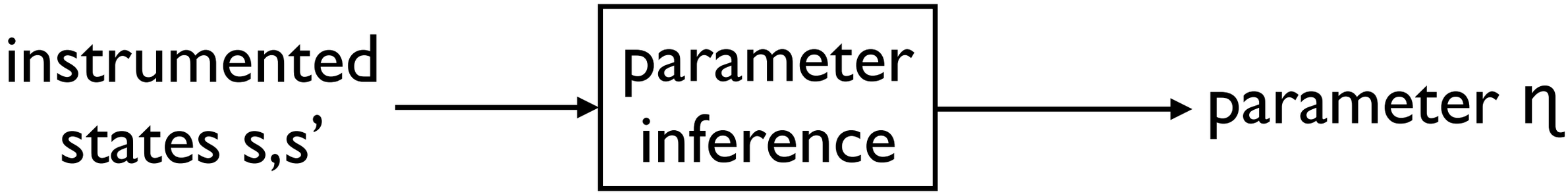


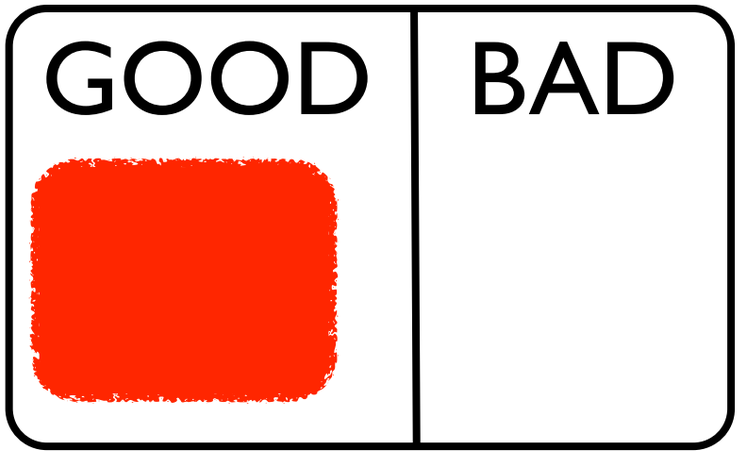
GOOD	BAD
s,s'	

Can separate?

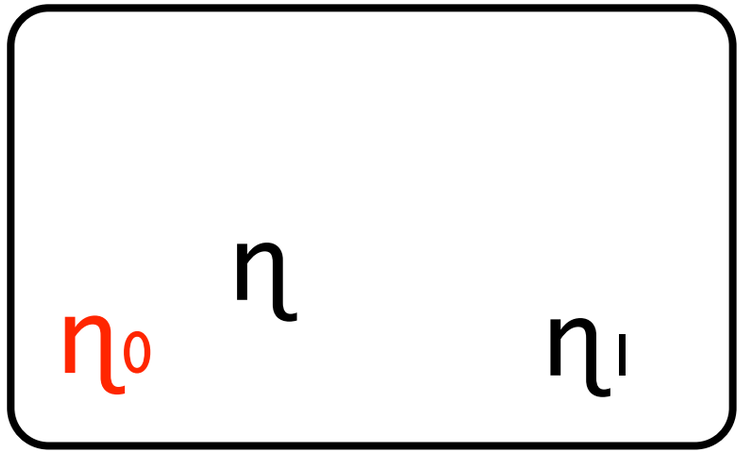


Parameter inference based on separability and minimality

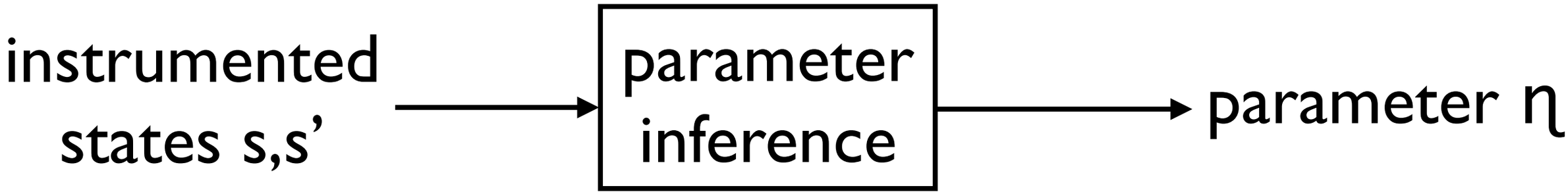




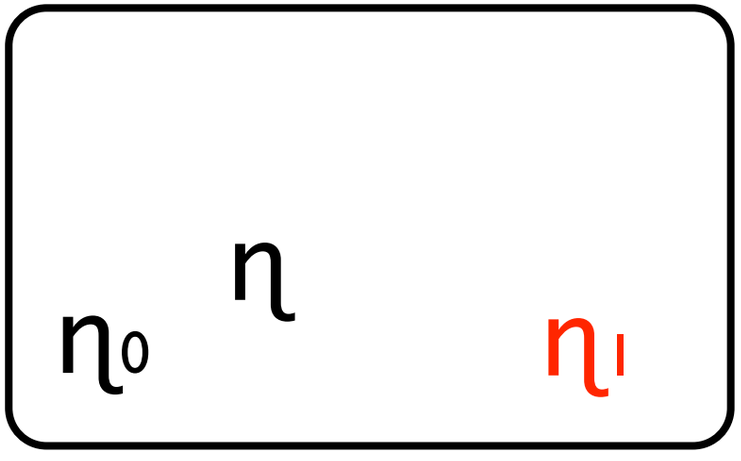
Can separate?



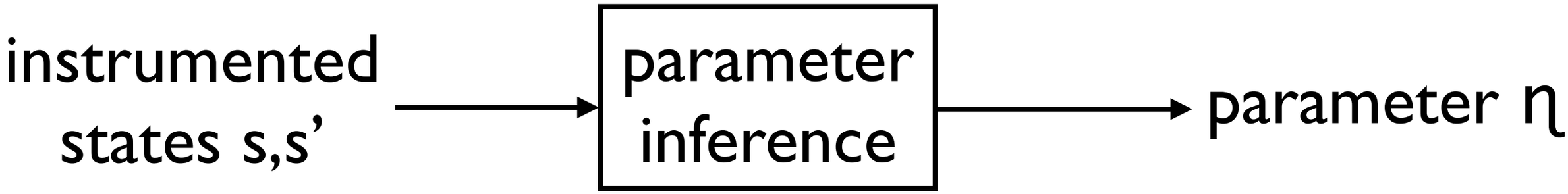
Parameter inference based on separability and minimality



Can separate?

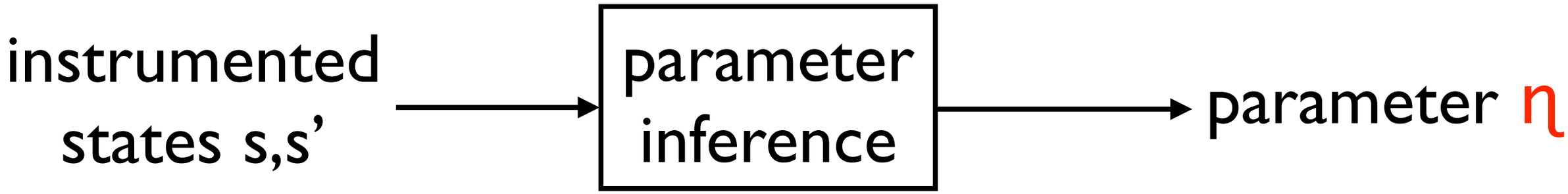


Parameter inference based on separability and minimality



- Computes a separability condition.

Parameter inference based on separability and minimality



- Computes a separability condition.
- Among separable η_i's, choose a minimal η according to an order (which approximately reflects precision).

Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

```
for (i = 0; i < n; i++) {
  x0 = new h0;
  x1 = new h1; x1.f1 = x0;
  x2 = new h2; x2.f2 = x1;
  x3 = new h3; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```

Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

```
for (i = 0; i < n; i++) {
  x0 = new h0;
  x1 = new h1; x1.f1 = x0;
  x2 = new h2; x2.f2 = x1;
  x3 = new h3; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```

Thread-escape query

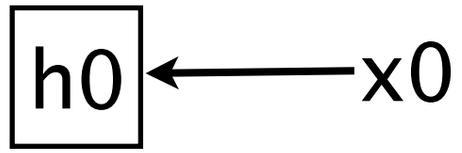
- Does a local variable point to an object that cannot be reached from other threads?

```
for (i = 0; i < n; i++) {
  x0 = new h0;
  x1 = new h1; x1.f1 = x0;
  x2 = new h2; x2.f2 = x1;
  x3 = new h3; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```

Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

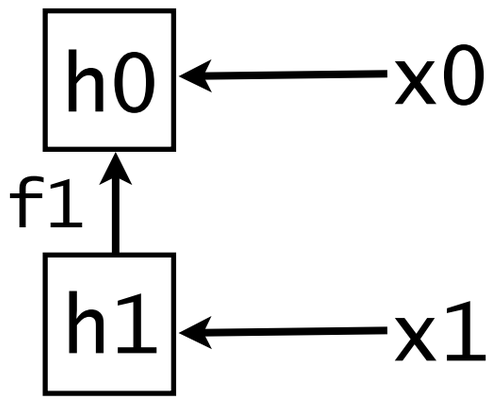
```
for (i = 0; i < n; i++) {
  x0 = new h0;
  x1 = new h1; x1.f1 = x0;
  x2 = new h2; x2.f2 = x1;
  x3 = new h3; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

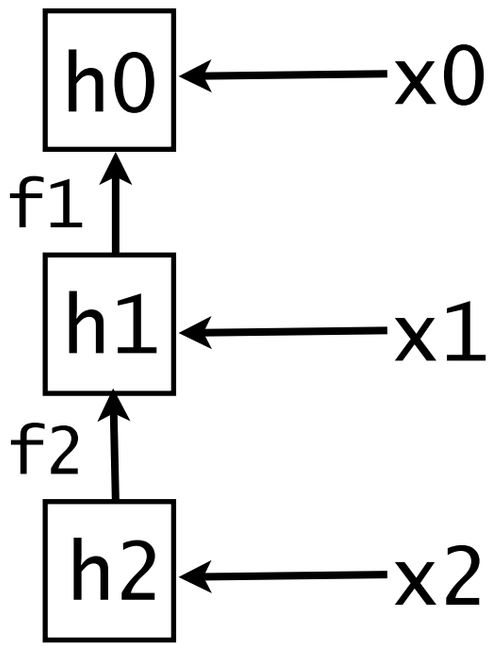
```
for (i = 0; i < n; i++) {
  x0 = new h0;
  x1 = new h1; x1.f1 = x0;
  x2 = new h2; x2.f2 = x1;
  x3 = new h3; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

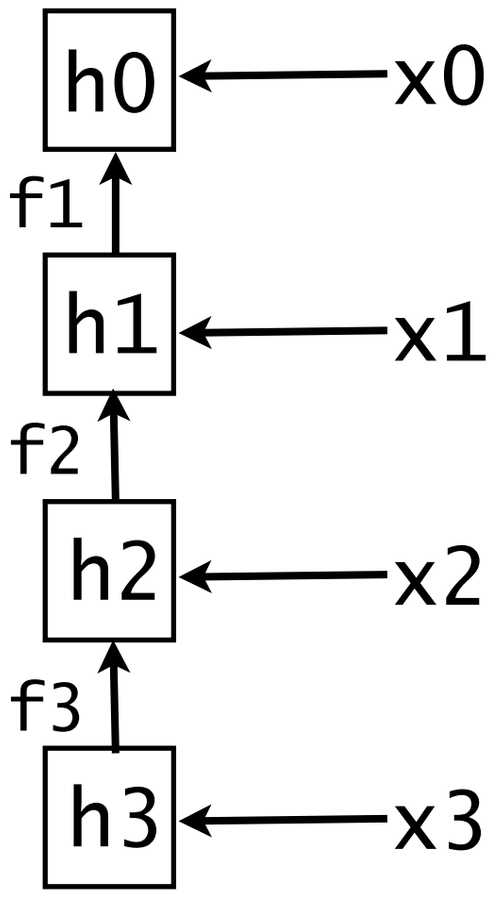
```
for (i = 0; i < n; i++) {
  x0 = new h0;
  x1 = new h1; x1.f1 = x0;
  x2 = new h2; x2.f2 = x1;
  x3 = new h3; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

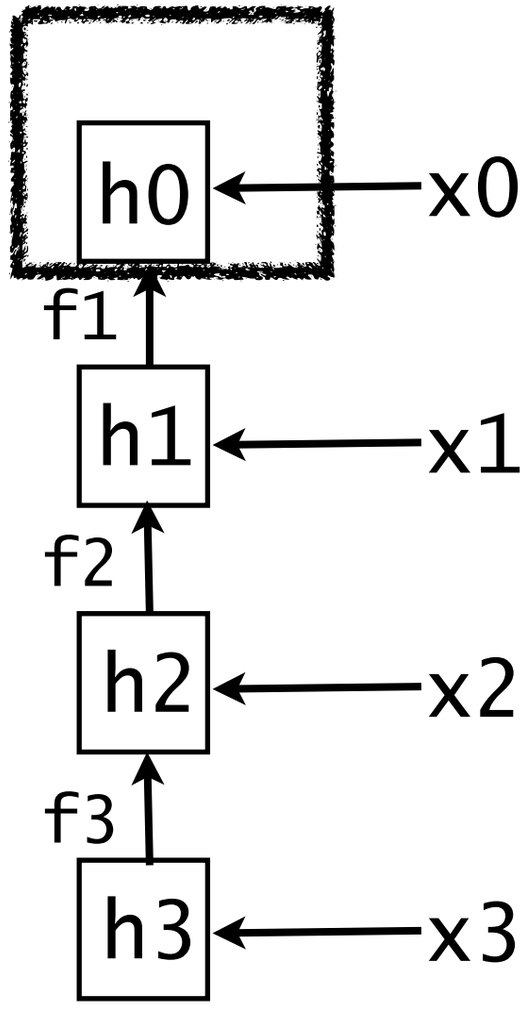
```
for (i = 0; i < n; i++) {
  x0 = new h0;
  x1 = new h1; x1.f1 = x0;
  x2 = new h2; x2.f2 = x1;
  x3 = new h3; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

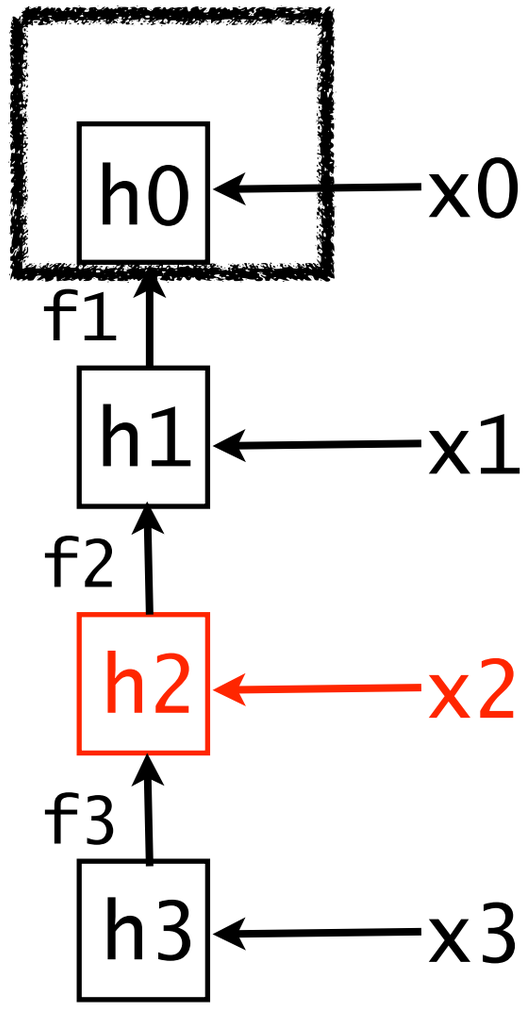
```
for (i = 0; i < n; i++) {
  x0 = new h0;
  x1 = new h1; x1.f1 = x0;
  x2 = new h2; x2.f2 = x1;
  x3 = new h3; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

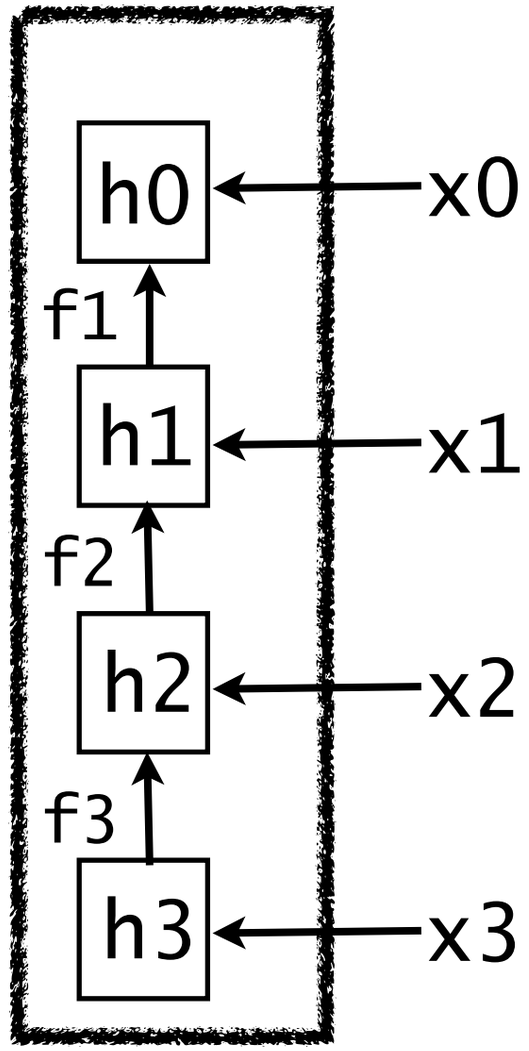
```
for (i = 0; i < n; i++) {
  x0 = new h0;
  x1 = new h1; x1.f1 = x0;
  x2 = new h2; x2.f2 = x1;
  x3 = new h3; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

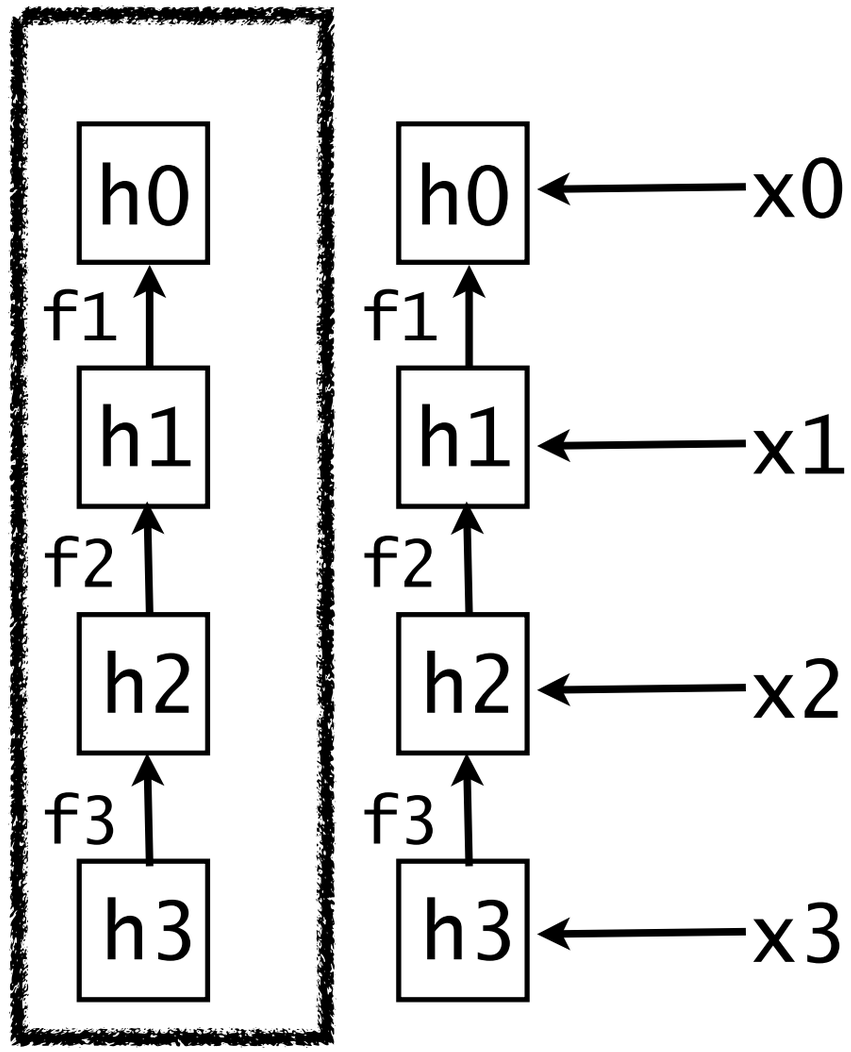
```
for (i = 0; i < n; i++) {
  x0 = new h0;
  x1 = new h1; x1.f1 = x0;
  x2 = new h2; x2.f2 = x1;
  x3 = new h3; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

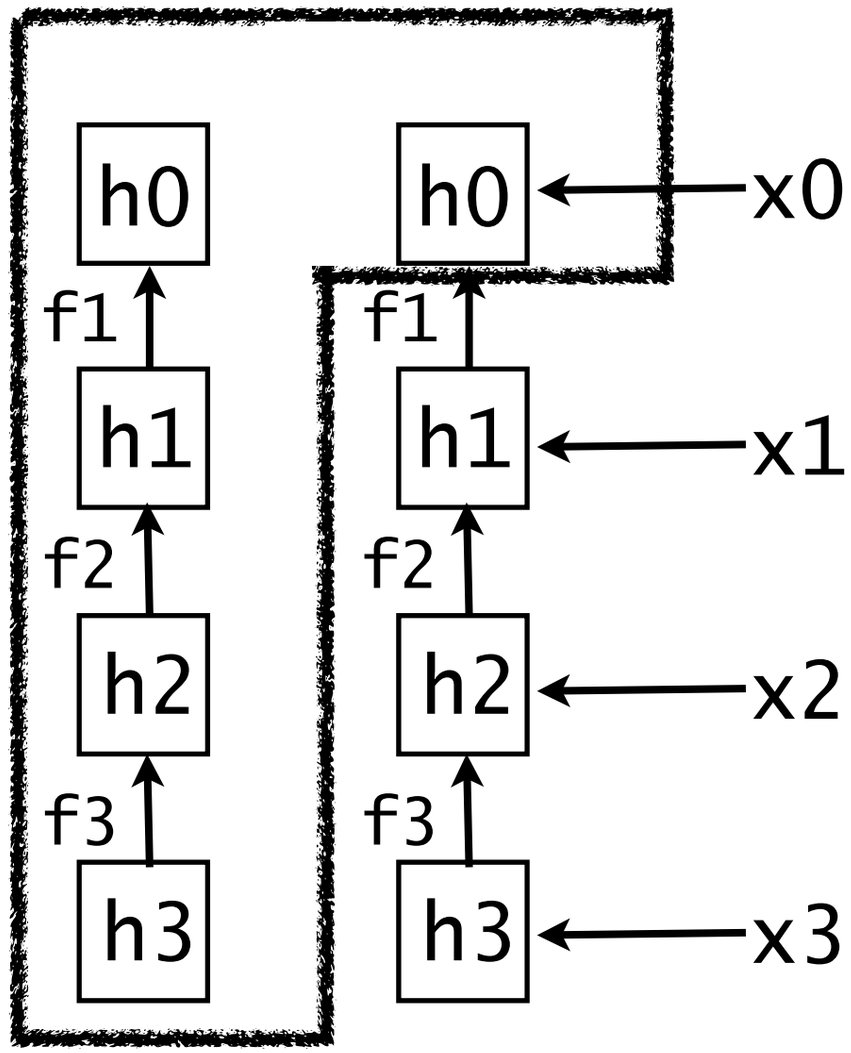
```
for (i = 0; i < n; i++) {
   x0 = new h0;
   x1 = new h1; x1.f1 = x0;
   x2 = new h2; x2.f2 = x1;
   x3 = new h3; x3.f3 = x2;
   x0.start();
pc: x2.id = i; //local(x2)?
   x3.start();
}
```



Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

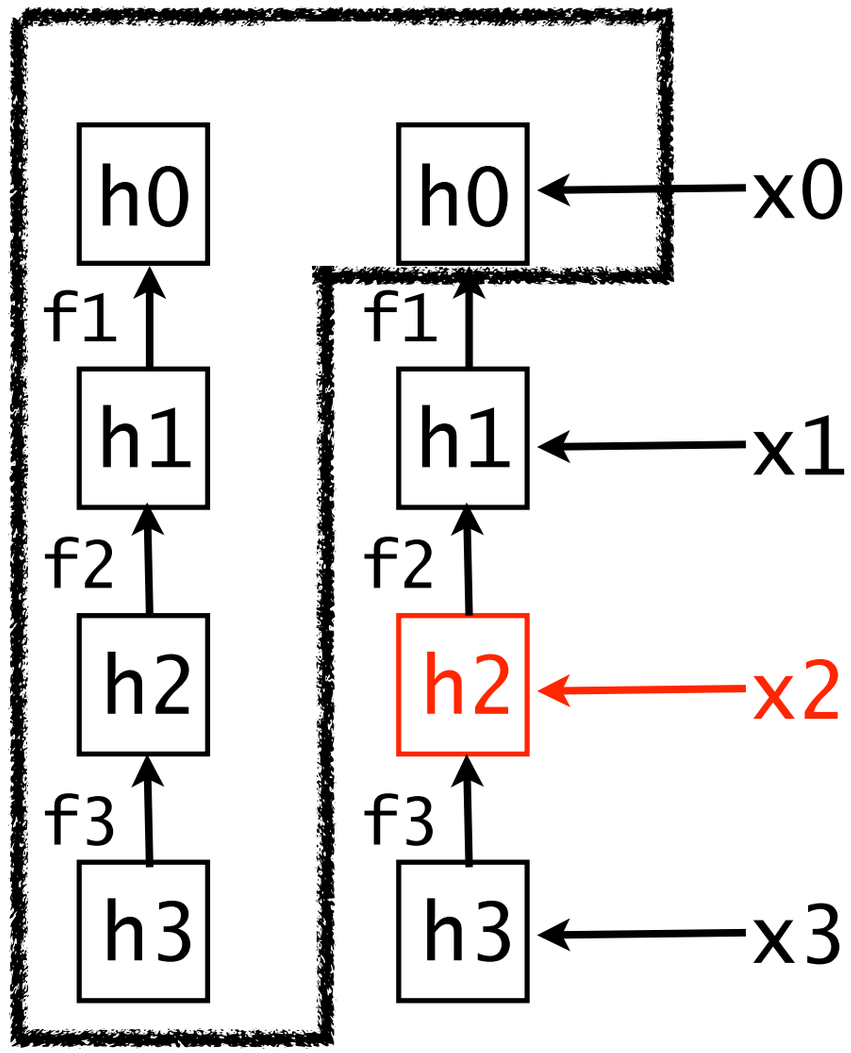
```
for (i = 0; i < n; i++) {
  x0 = new h0;
  x1 = new h1; x1.f1 = x0;
  x2 = new h2; x2.f2 = x1;
  x3 = new h3; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape query

- Does a local variable point to an object that cannot be reached from other threads?

```
for (i = 0; i < n; i++) {
  x0 = new h0;
  x1 = new h1; x1.f1 = x0;
  x2 = new h2; x2.f2 = x1;
  x3 = new h3; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape analysis

- Summarise all heap objects with only two abstract nodes E and L.
- $\gamma(E)$ consists of all the thread-escaping objects and possibly more.
- $\gamma(L)$ contains only thread-local objects.

Parameterisation

$$\text{Param} = \text{AllocSite} \to \{\mathrm{L}, \mathrm{E}\}$$

- For each allocation site, it decides whether L or E is used to summarise allocated objects.
- Changes the transfer function of "x=new h_i".
- Objects summarised by L can move to E, but not vice versa.

Thread-escape analysis

- Parameter η = [{h0,h1}↦E, {h2,h3}↦L]

```
for (i = 0; i < n; i++) {
  x0 = new h0;
  x1 = new h1; x1.f1 = x0;
  x2 = new h2; x2.f2 = x1;
  x3 = new h3; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```

Thread-escape analysis

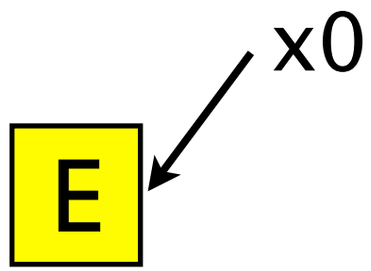
- Parameter η = [{h0,h1}↦E, {h2,h3}↦L]

```
for (i = 0; i < n; i++) {
  x0 = new h0/E;
  x1 = new h1/E; x1.f1 = x0;
  x2 = new h2/L; x2.f2 = x1;
  x3 = new h3/L; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```

Thread-escape analysis

- Parameter η = [{h0,h1}↦E, {h2,h3}↦L]

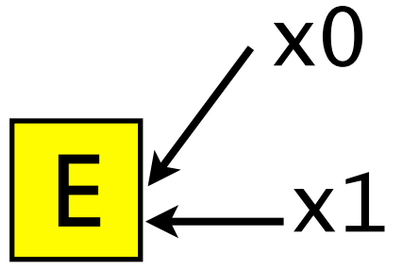
```
for (i = 0; i < n; i++) {
  x0 = new h0/E;
  x1 = new h1/E; x1.f1 = x0;
  x2 = new h2/L; x2.f2 = x1;
  x3 = new h3/L; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape analysis

- Parameter η = [{h0,h1}↦E, {h2,h3}↦L]

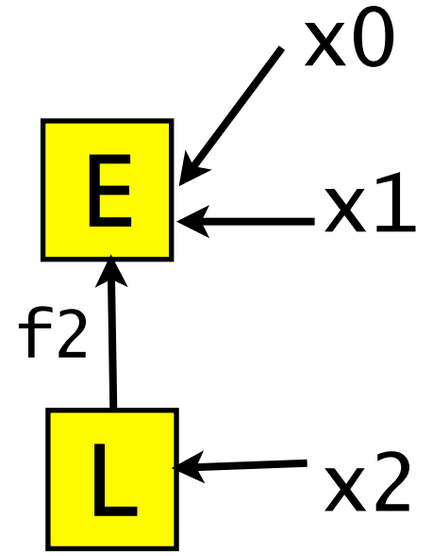
```
for (i = 0; i < n; i++) {
  x0 = new h0/E;
  x1 = new h1/E; x1.f1 = x0;
  x2 = new h2/L; x2.f2 = x1;
  x3 = new h3/L; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape analysis

- Parameter η = [{h0,h1}↦E, {h2,h3}↦L]

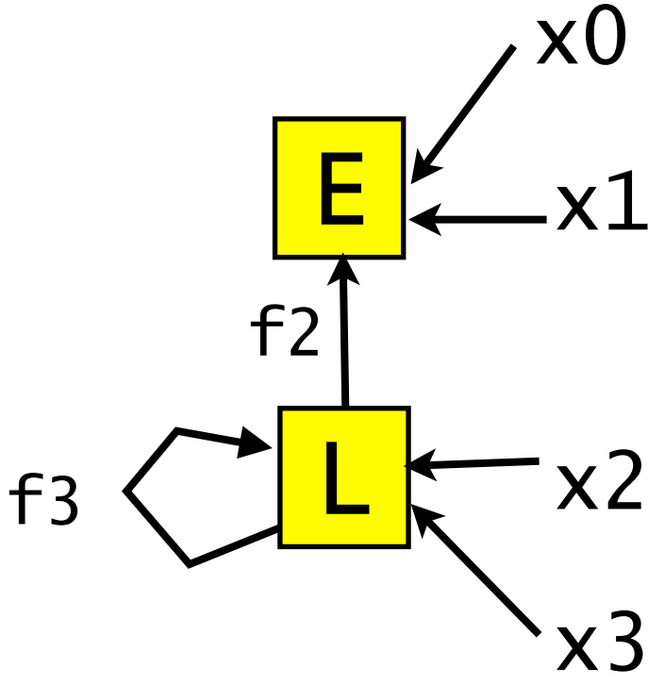
```
for (i = 0; i < n; i++) {
  x0 = new h0/E;
  x1 = new h1/E; x1.f1 = x0;
  x2 = new h2/L; x2.f2 = x1;
  x3 = new h3/L; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape analysis

- Parameter η = [{h0,h1}↦E, {h2,h3}↦L]

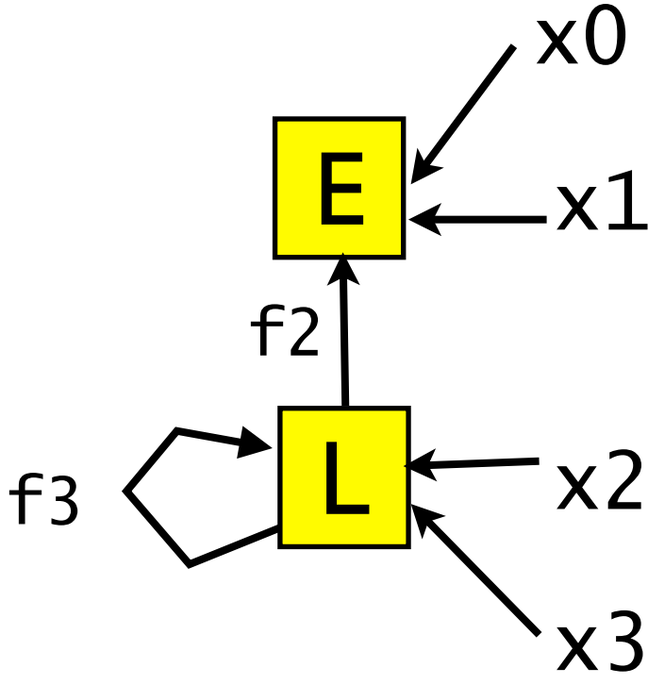
```
for (i = 0; i < n; i++) {
  x0 = new h0/E;
  x1 = new h1/E; x1.f1 = x0;
  x2 = new h2/L; x2.f2 = x1;
  x3 = new h3/L; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape analysis

- Parameter η = [{h0,h1}↦E, {h2,h3}↦L]

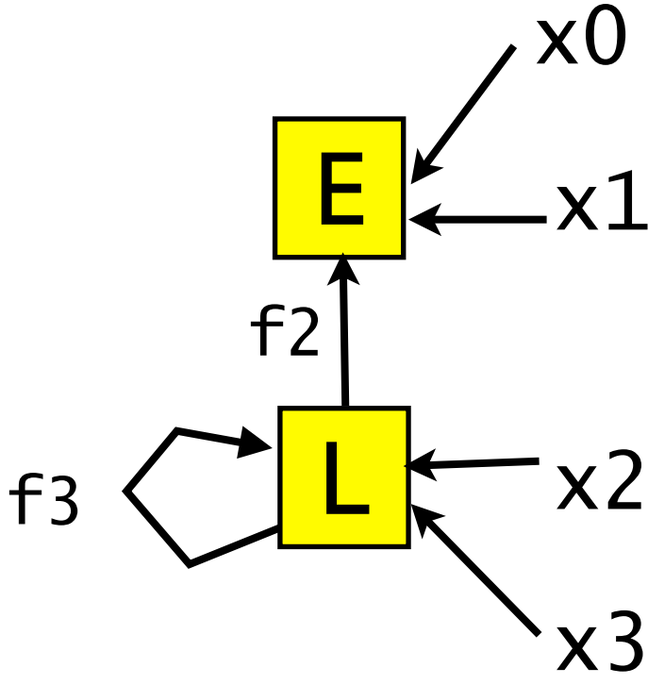
```
for (i = 0; i < n; i++) {
  x0 = new h0/E;
  x1 = new h1/E; x1.f1 = x0;
  x2 = new h2/L; x2.f2 = x1;
  x3 = new h3/L; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape analysis

- Parameter η = [{h0,h1}↦E, {h2,h3}↦L]

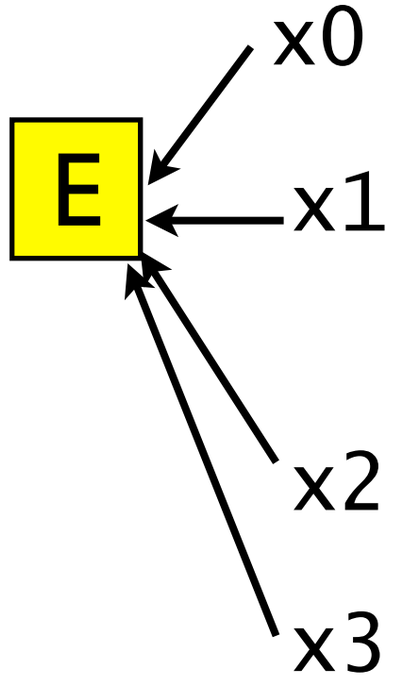
```
for (i = 0; i < n; i++) {
  x0 = new h0/E;
  x1 = new h1/E; x1.f1 = x0;
  x2 = new h2/L; x2.f2 = x1;
  x3 = new h3/L; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape analysis

- Parameter η = [{h0,h1}↦E, {h2,h3}↦L]

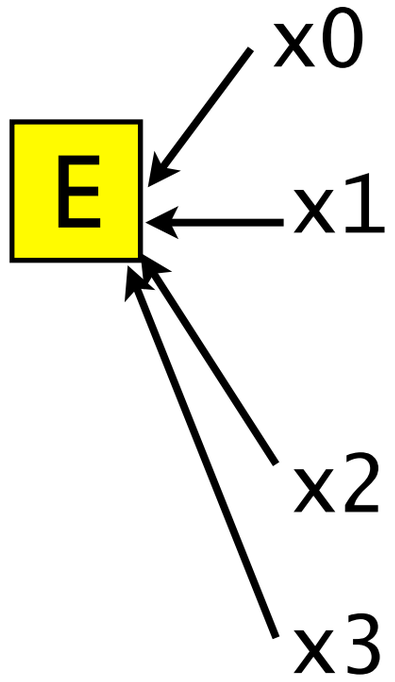
```
for (i = 0; i < n; i++) {
  x0 = new h0/E;
  x1 = new h1/E; x1.f1 = x0;
  x2 = new h2/L; x2.f2 = x1;
  x3 = new h3/L; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape analysis

- Parameter η = [{h0,h1}↦E, {h2,h3}↦L]

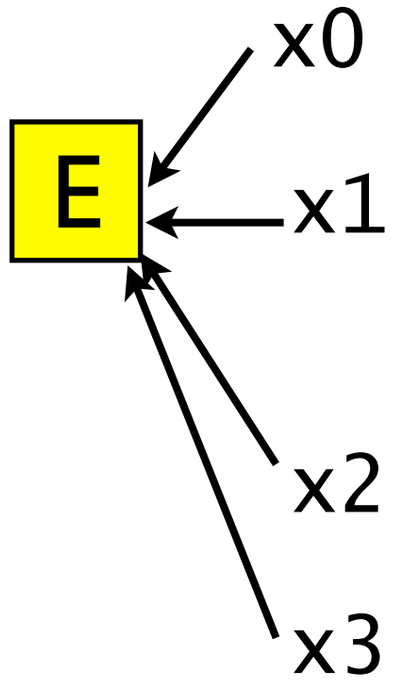
```
for (i = 0; i < n; i++) {
  x0 = new h0/E;
  x1 = new h1/E; x1.f1 = x0;
  x2 = new h2/L; x2.f2 = x1;
  x3 = new h3/L; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape analysis

- Parameter η = [{h0,h1}↦E, {h2,h3}↦L]

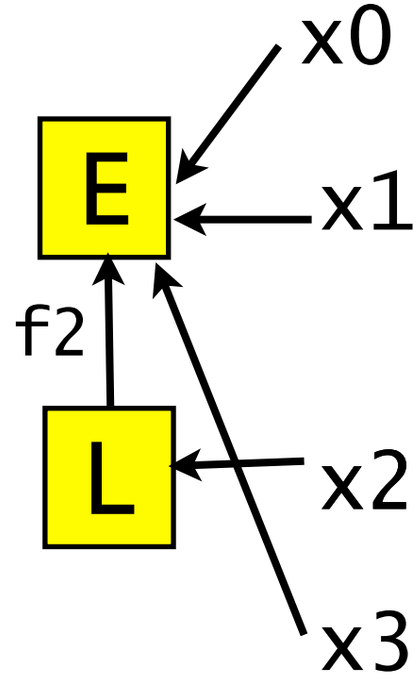
```
for (i = 0; i < n; i++) {
  x0 = new h0/E;
  x1 = new h1/E; x1.f1 = x0;
  x2 = new h2/L; x2.f2 = x1;
  x3 = new h3/L; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape analysis

- Parameter η = [{h0,h1}↦E, {h2,h3}↦L]

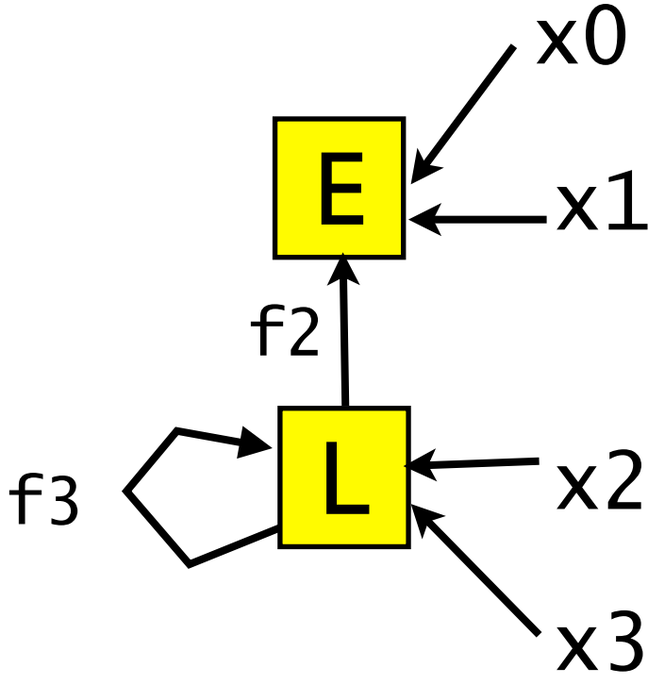
```
for (i = 0; i < n; i++) {
  x0 = new h0/E;
  x1 = new h1/E; x1.f1 = x0;
  x2 = new h2/L; x2.f2 = x1;
  x3 = new h3/L; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape analysis

- Parameter η = [{h0,h1}↦E, {h2,h3}↦L]

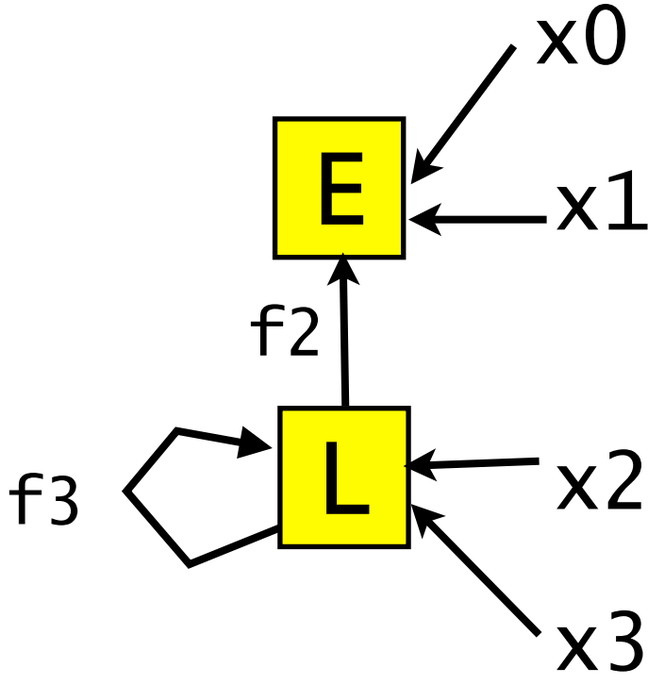
```
for (i = 0; i < n; i++) {
  x0 = new h0/E;
  x1 = new h1/E; x1.f1 = x0;
  x2 = new h2/L; x2.f2 = x1;
  x3 = new h3/L; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape analysis

- Parameter η = [{h0,h1}↦E, {h2,h3}↦L]

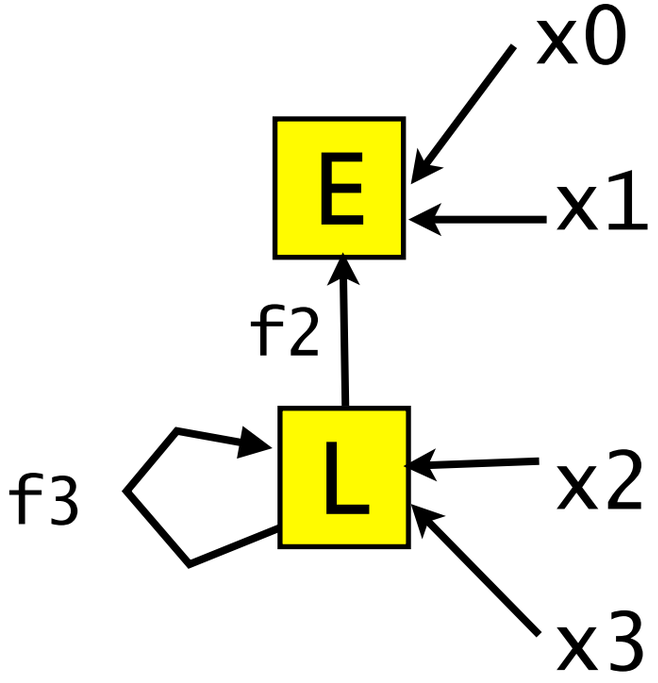
```
for (i = 0; i < n; i++) {
  x0 = new h0/E;
  x1 = new h1/E; x1.f1 = x0;
  x2 = new h2/L; x2.f2 = x1;
  x3 = new h3/L; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



Thread-escape analysis

- Parameter η = [{h0,h1}↦E, {h2,h3}↦L]

```
for (i = 0; i < n; i++) {
  x0 = new h0/E;
  x1 = new h1/E; x1.f1 = x0;
  x2 = new h2/L; x2.f2 = x1;
  x3 = new h3/L; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```



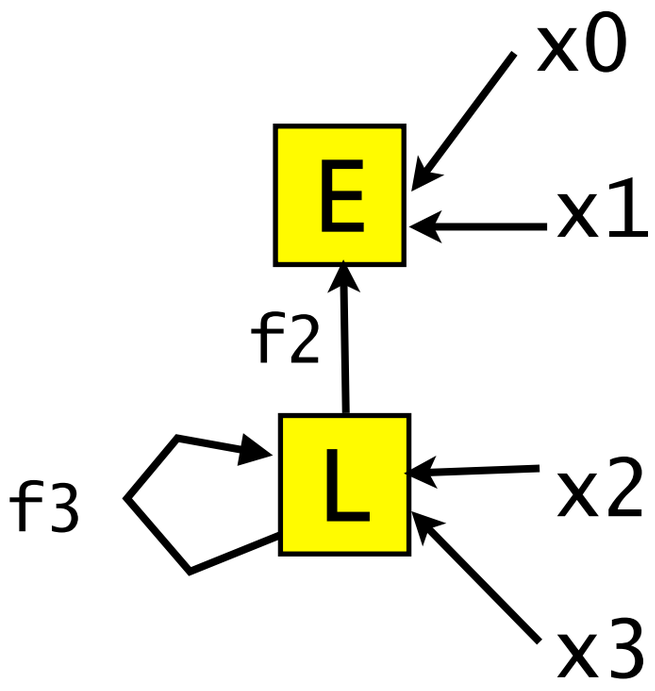
Difficulties in choosing a good parameter

- Using more L makes the analysis more expensive.

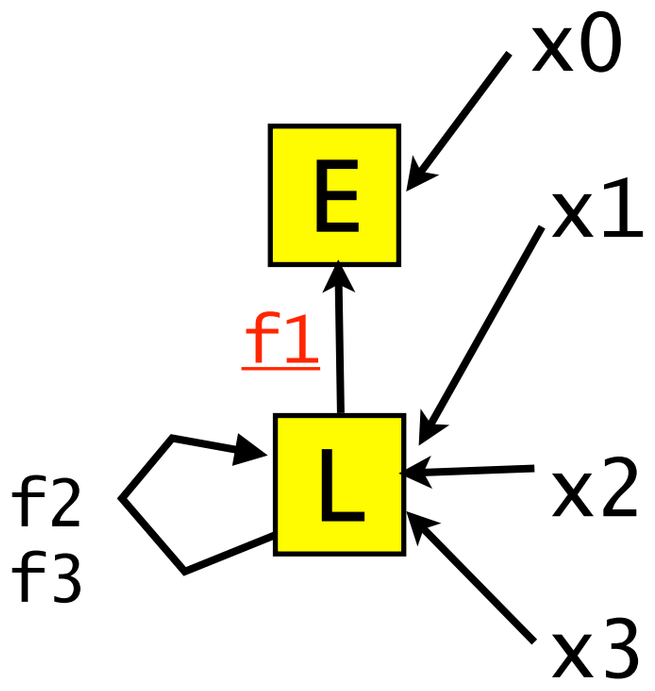
Difficulties in choosing a good parameter

- Using more L makes the analysis more expensive.

[{h0,h1}↦E, {h2,h3}↦L]



[{h0}↦E, {h1,h2,h3}↦L]



Difficulties in choosing a good parameter

- Using more L makes the analysis more expensive.
- More L doesn't always mean more precision.

Difficulties in choosing a good parameter

- Using more L makes the analysis more expensive.
- More L doesn't always mean more precision.

```
for (i = 0; i < n; i++) {
  x0 = new h0;
  x1 = new h1; x1.f1 = x0;
  x2 = new h2; x2.f2 = x1;
  x3 = new h3; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```

Difficulties in choosing a good parameter

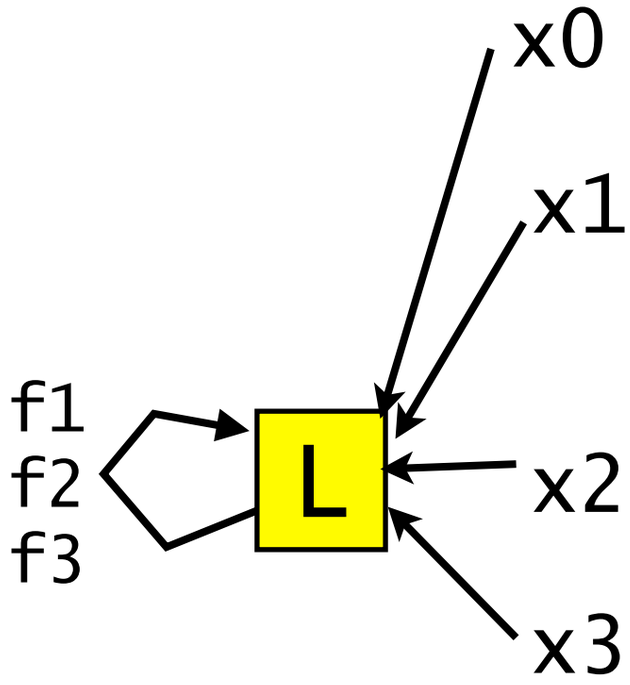
- Using more L makes the analysis more expensive.
- More L doesn't always mean more precision.

```
for (i = 0; i < n; i++) {
  x0 = new h0/L;
  x1 = new h1/L; x1.f1 = x0;
  x2 = new h2/L; x2.f2 = x1;
  x3 = new h3/L; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```

Difficulties in choosing a good parameter

- Using more L makes the analysis more expensive.
- More L doesn't always mean more precision.

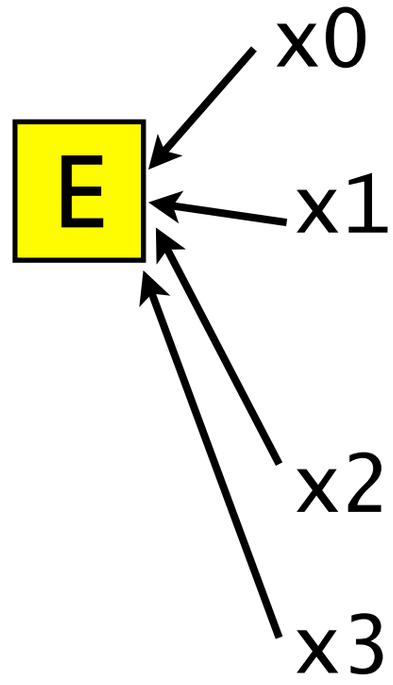
```
for (i = 0; i < n; i++) {
   x0 = new h0/L;
   x1 = new h1/L; x1.f1 = x0;
   x2 = new h2/L; x2.f2 = x1;
   x3 = new h3/L; x3.f3 = x2;
   x0.start();
pc: x2.id = i; //local(x2)?
   x3.start();
}
```



Difficulties in choosing a good parameter

- Using more L makes the analysis more expensive.
- More L doesn't always mean more precision.

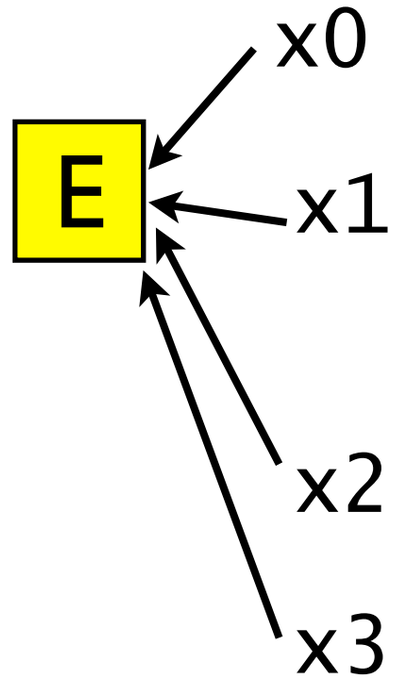
```
for (i = 0; i < n; i++) {
   x0 = new h0/L;
   x1 = new h1/L; x1.f1 = x0;
   x2 = new h2/L; x2.f2 = x1;
   x3 = new h3/L; x3.f3 = x2;
   x0.start();
pc: x2.id = i; //local(x2)?
   x3.start();
}
```



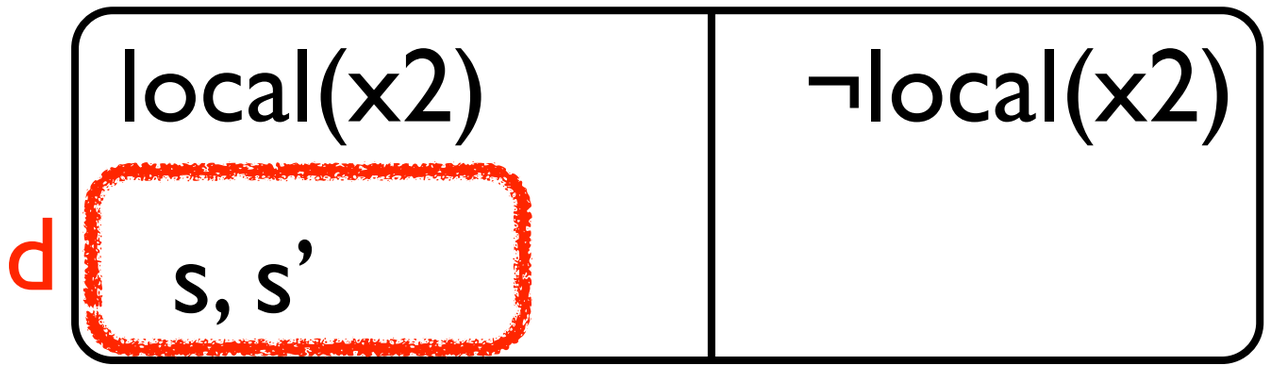
Difficulties in choosing a good parameter

- Using more L makes the analysis more expensive.
- More L doesn't always mean more precision.

```
for (i = 0; i < n; i++) {
  x0 = new h0/L;
  x1 = new h1/L; x1.f1 = x0;
  x2 = new h2/L; x2.f2 = x1;
  x3 = new h3/L; x3.f3 = x2;
  x0.start();
pc: x2.id = i; //local(x2)?
  x3.start();
}
```

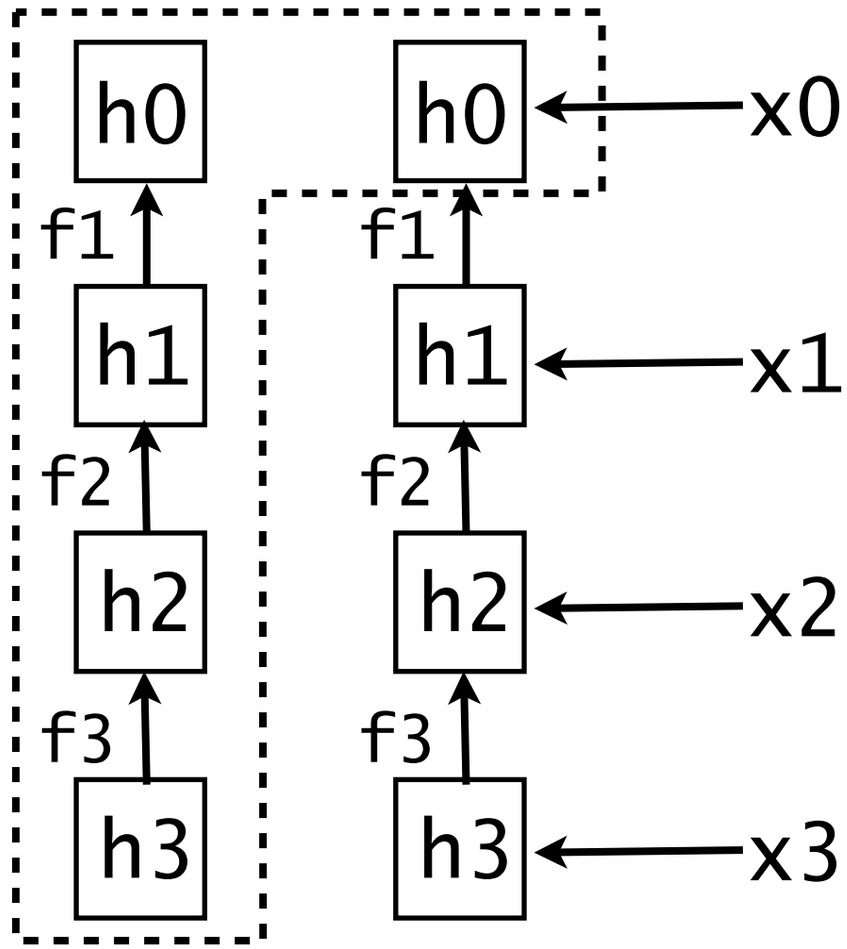


Separability question



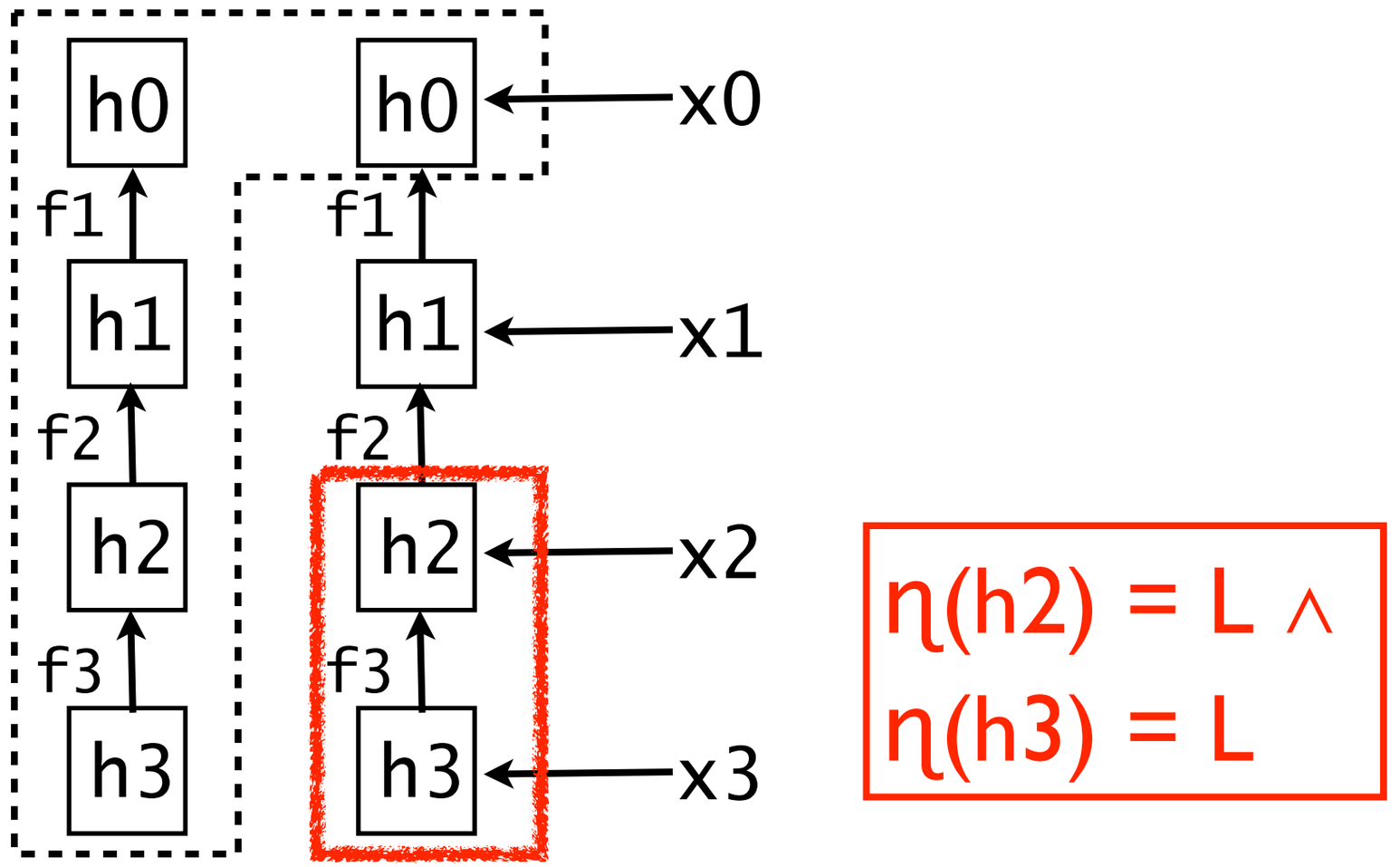
- Does analysis(η) have an abstract element d separating {s, s’} from ¬local(x2)?
- We use a generic answer to this question during our parameter inference.

Separability from ¬local(x2)



- This state satisfies local(x2).

Separability from ¬local(x2)

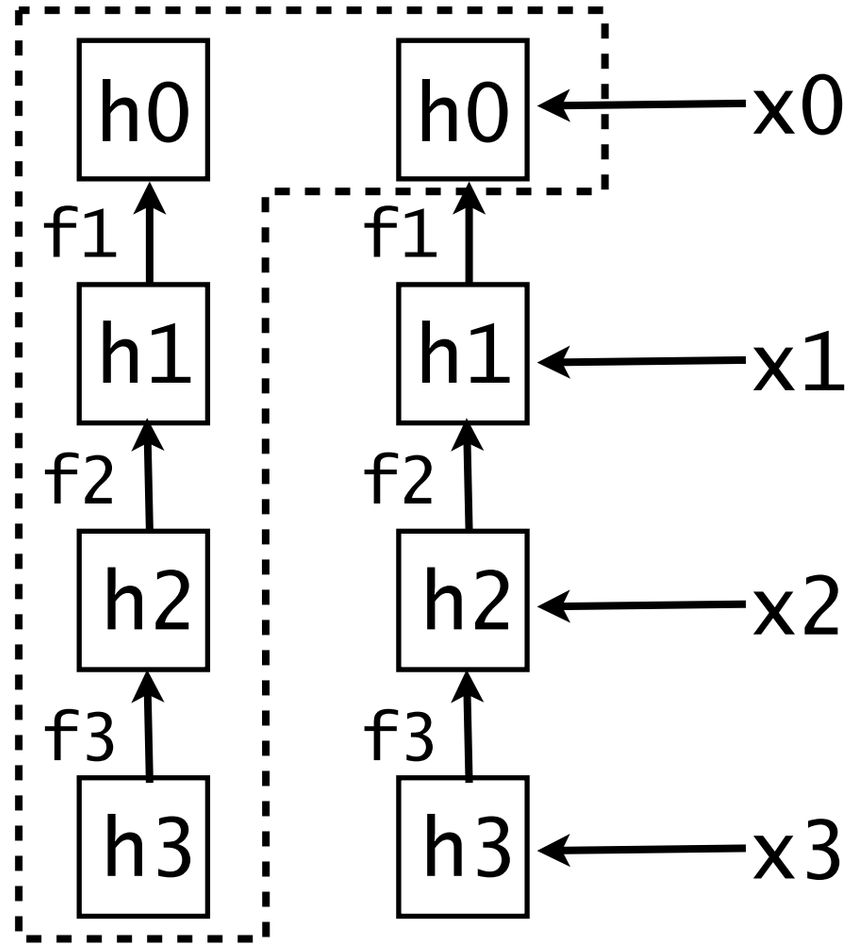


η(h2) = L ∧
η(h3) = L

- This state satisfies local(x2).
- Separated from ¬local(x2) by analysis(η) iff (η ∘ allocSite ∘ backReach)(x2) = {L}.

Parameter inference

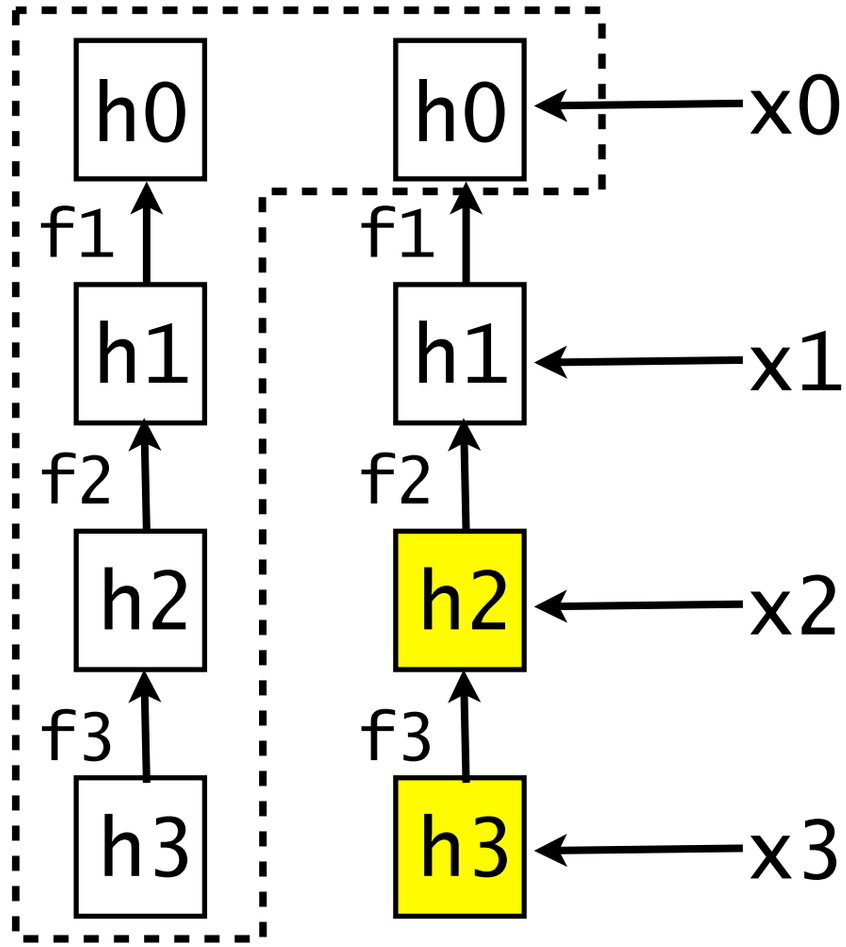
1. Testing gives states where local(x2) holds.



Parameter inference

1. Testing gives states where local(x2) holds.

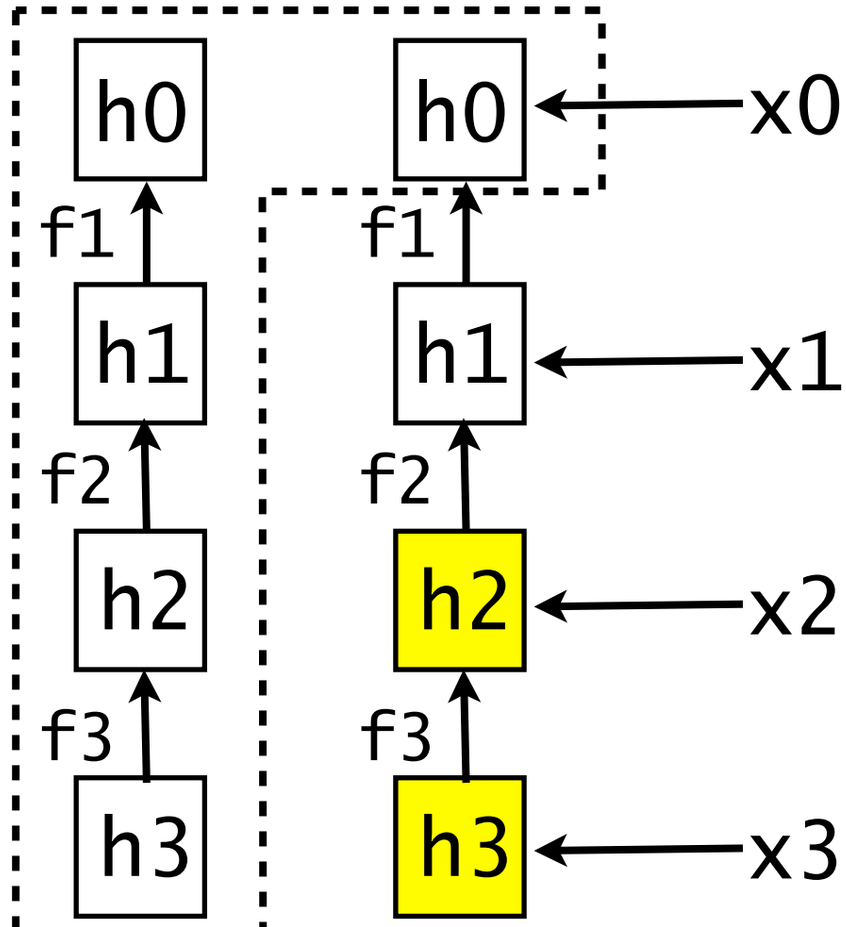
2. Compute the alloc. sites H of objects that can reach x2.



H = {h2, h3}

Parameter inference

1. Testing gives states where local(x2) holds.
2. Compute the alloc. sites H of objects that can reach x2.
3. $\eta(h) = L$, if h is in H;
   $\eta(h) = E$, otherwise.



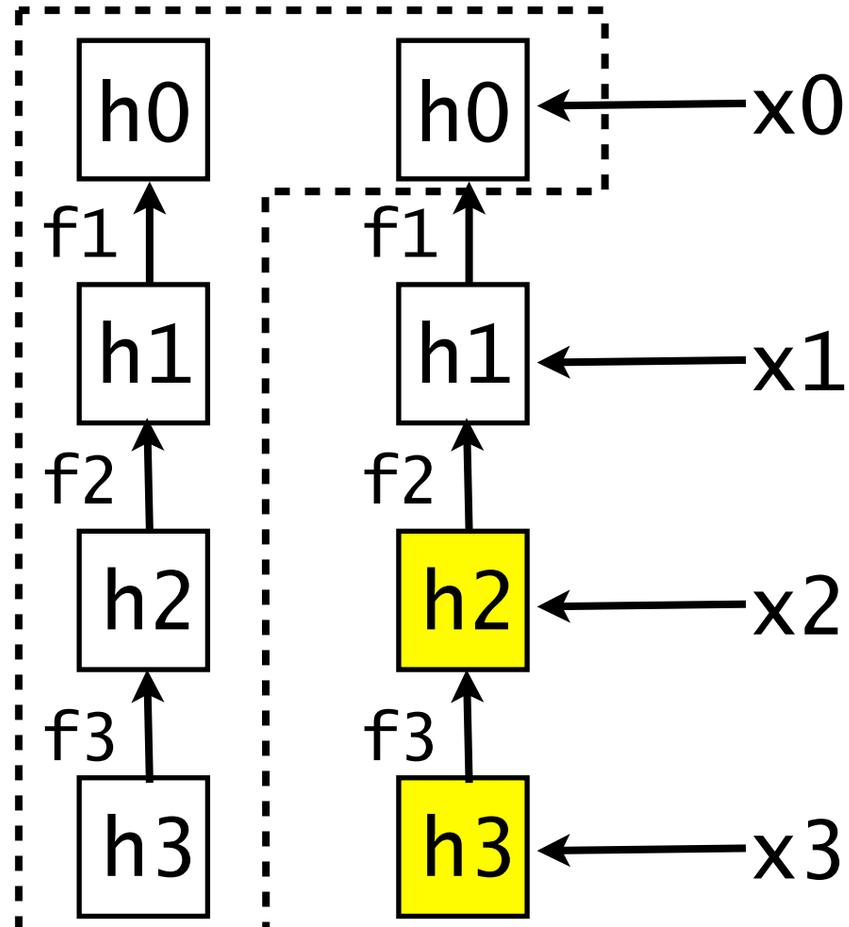
$H = \{h2, h3\}$

$\eta = [\{h0,h1\} \mapsto E, \{h2,h3\} \mapsto L]$

Parameter inference

1. Testing gives states where local(x2) holds.
2. Compute the alloc. sites H of objects that can reach x2.
3. η(h) = L, if h is in H;
   η(h) = E, otherwise.

separability



H = {h2, h3}

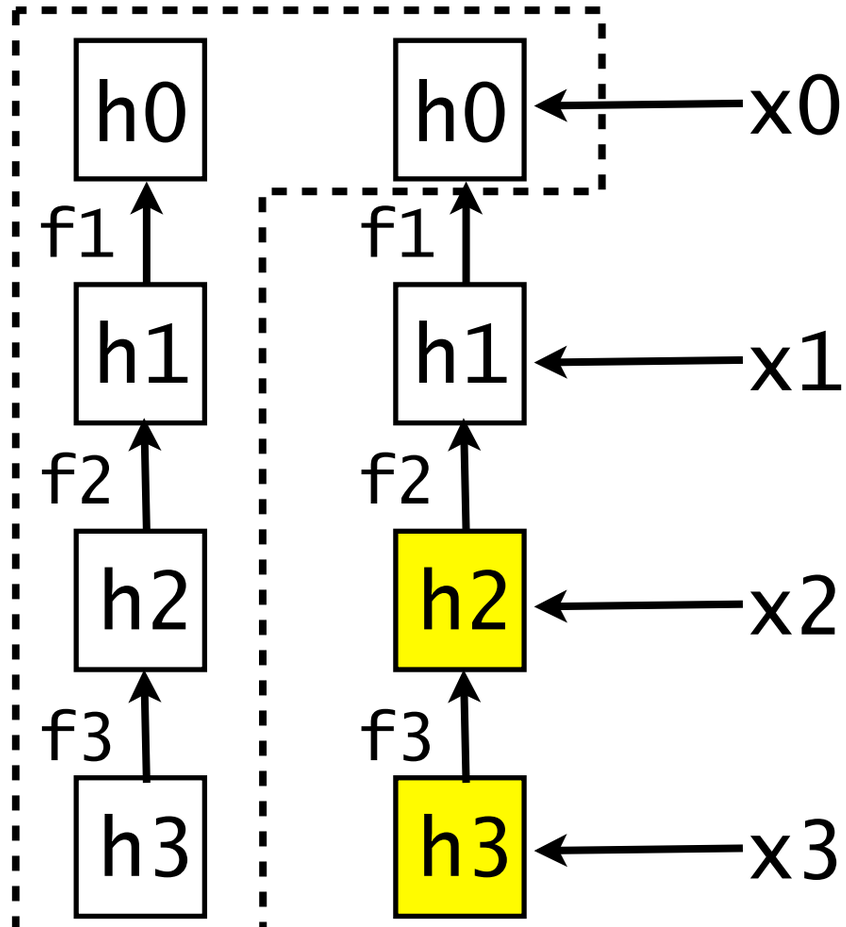
η = [{h0,h1}↦E, {h2,h3}↦L]

Parameter inference

1. Testing gives states where local(x2) holds.
2. Compute the alloc. sites H of objects that can reach x2.
3. η(h) = L, if h is in H;
   η(h) = E, otherwise.

separability

minimality



H = {h2, h3}

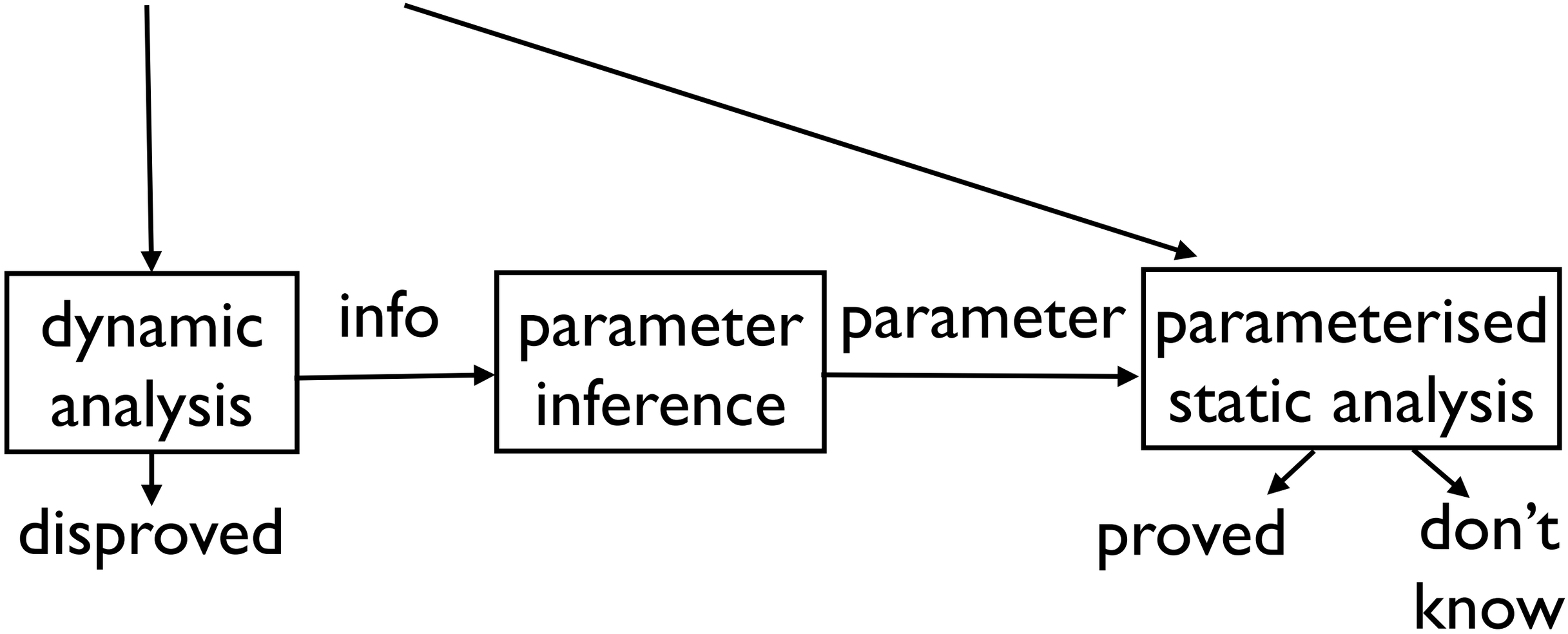
η = [{h0,h1}↦E, {h2,h3}↦L]

Does it work?

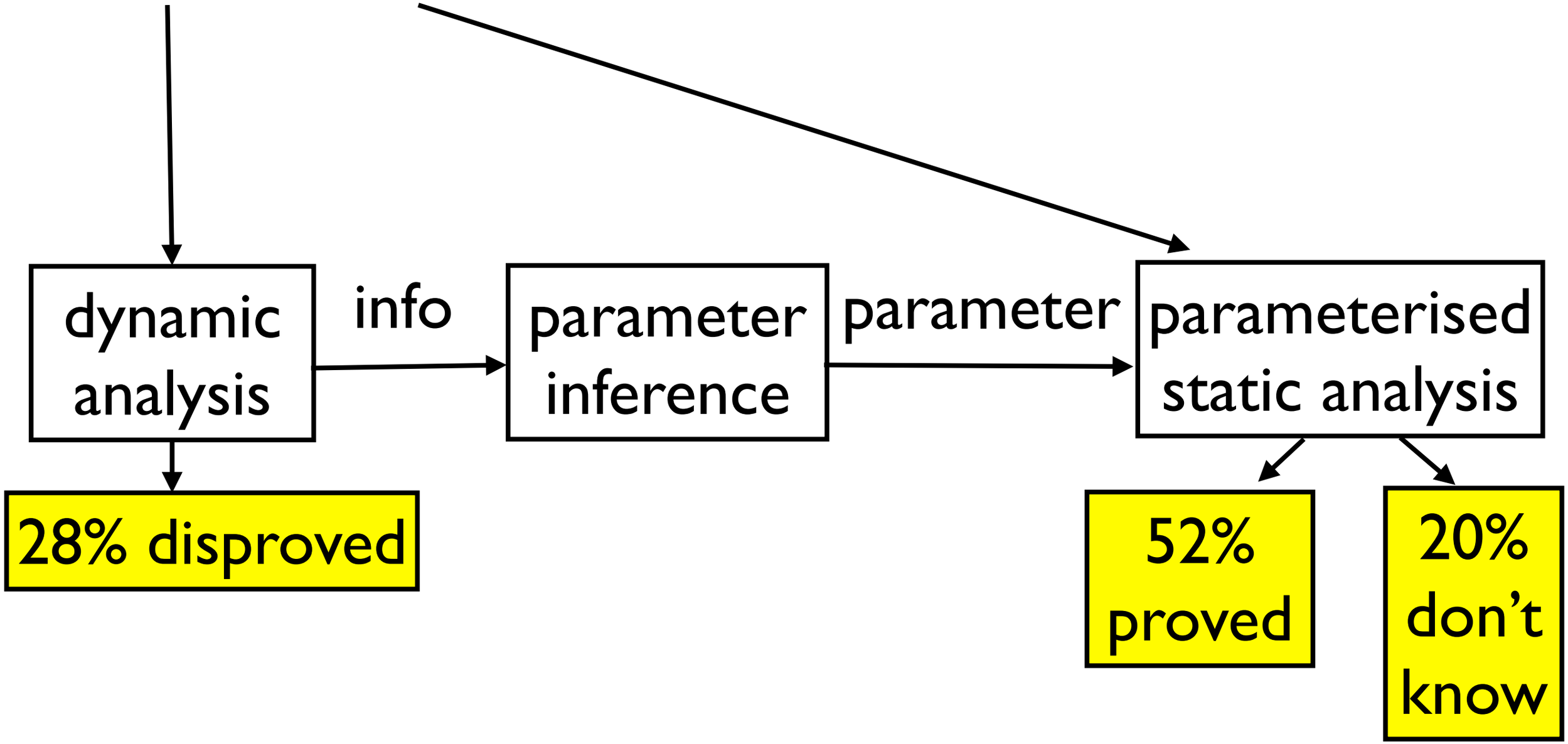
Setting of experiments

- 6 concurrent Java programs from Dacapo:
  - 161K - 491K bytecode (including analysed JDK).
    - Up to 5K allocation sites per program.
- 47K queries, but only 17K(37%) reached during testing.
- Considered only these reachable queries.

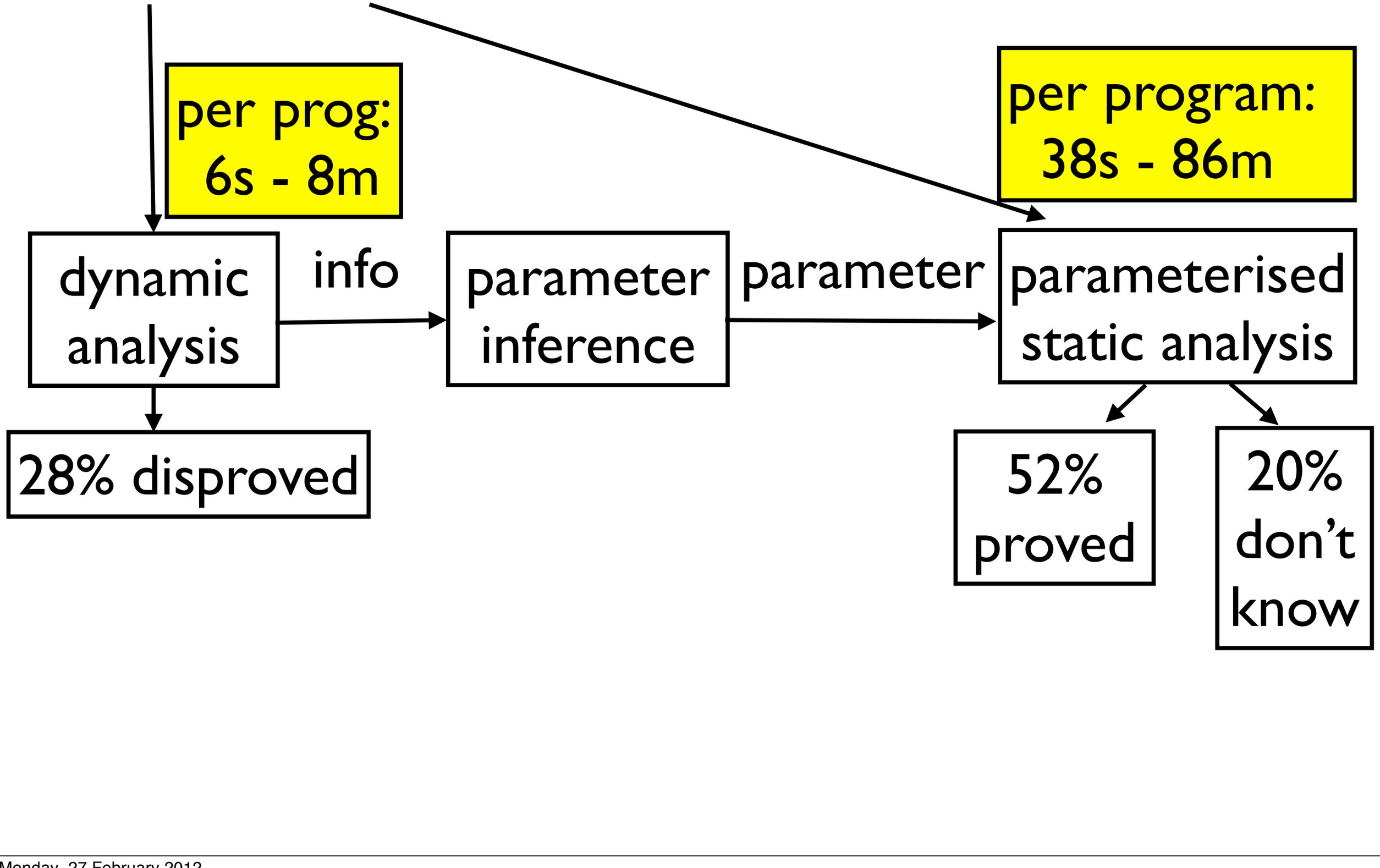
6 Java prog. (161K-491K) up to 5K sites
17K queries



6 Java prog. (161K-491K) up to 5K sites
17K queries



6 Java prog. (161K-491K) up to 5K sites
17K queries



6 Java prog. (161K-491K) up to 5K sites
17K queries

per prog:
6s - 8m

per program:
38s - 86m

dynamic analysis

info

parameter inference

parameter

parameterised static analysis

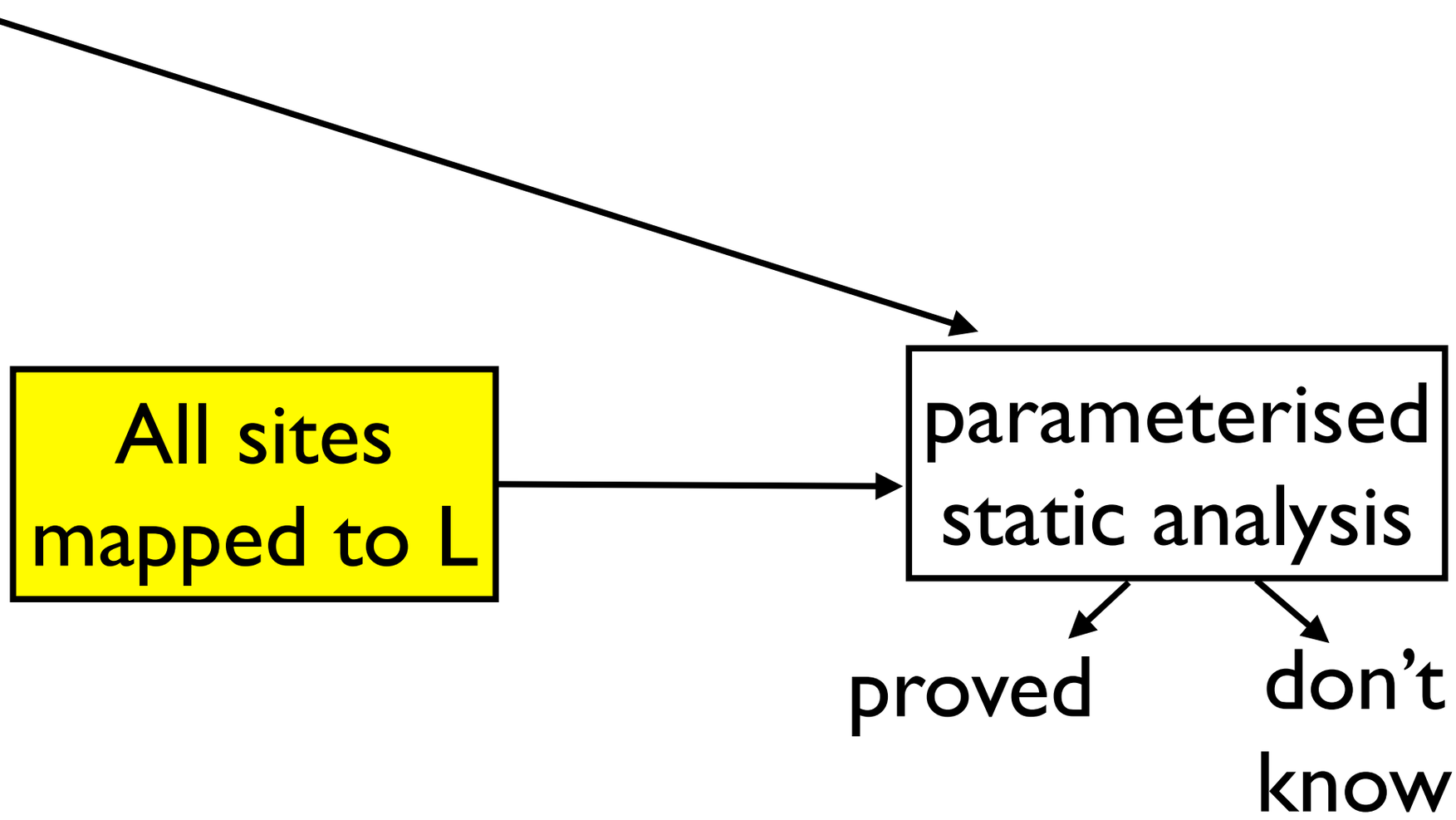
28% disproved

L-mapped sites:
avg 4.8, max 195

52% proved

20% don't know

6 Java prog. (161K-491K) up to 5K sites
17K queries



6 Java prog. (161K-491K) up to 5K sites
17K queries