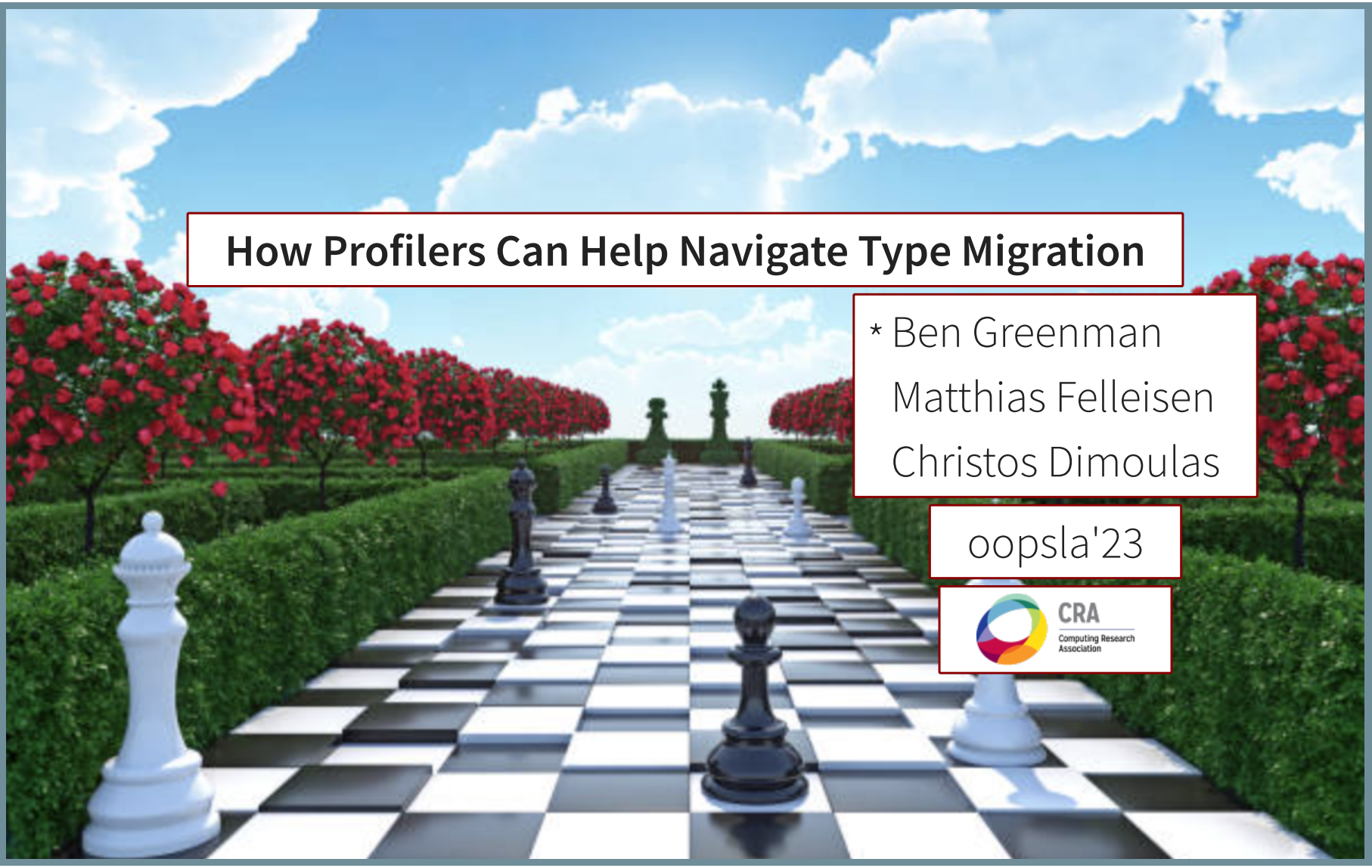

How Profilers Can Help Navigate Type Migration
* Ben Greenman
Matthias Felleisen
Christos Dimoulas
oopsla'23
CRA
Computing Research Association

How Profilers Can Help Navigate Type Migration

How Profilers Can Help Navigate Type Migration

How to avoid **runtime costs** using **off-the-shelf tools**?

How Profilers Can Help Navigate Type Migration

How to avoid **runtime costs** using **off-the-shelf tools**?

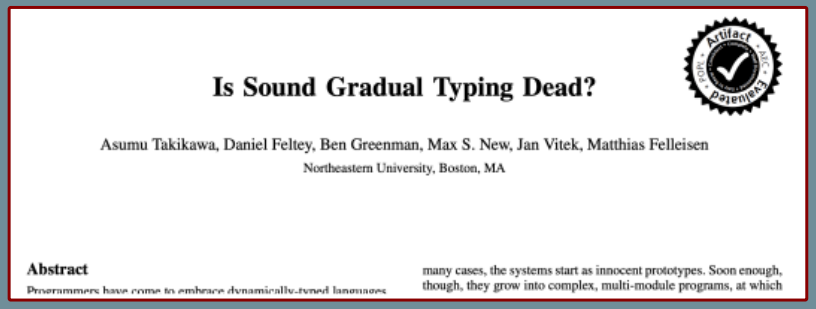
costs ~ gradual types

tools ~ statistical profilers

Old Problem, New Idea

Old Problem, New Idea

popl'16: 10x slowdowns are common,
but fast points exist!



Is Sound Gradual Typing Dead?

Asumu Takikawa, Daniel Feltey, Ben Greenman, Max S. New, Jan Vitek, Matthias Felleisen
Northeastern University, Boston, MA

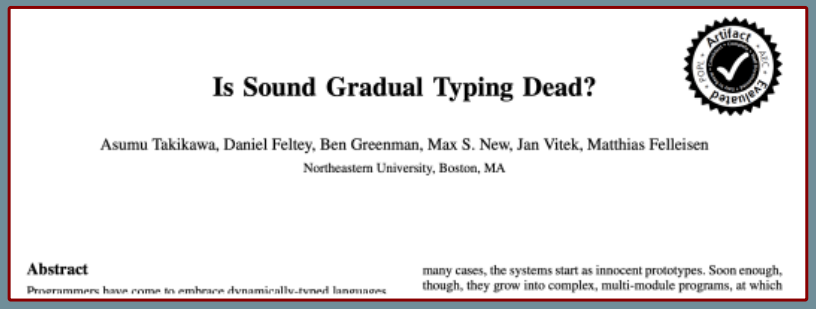
Abstract

Programmers have come to embrace dynamically-typed languages

many cases, the systems start as innocent prototypes. Soon enough, though, they grow into complex, multi-module programs, at which

Old Problem, New Idea

popl'16: 10x slowdowns are common,
but fast points exist!



Is Sound Gradual Typing Dead?

Asumu Takikawa, Daniel Feltey, Ben Greenman, Max S. New, Jan Vitek, Matthias Felleisen

Northeastern University, Boston, MA

Abstract

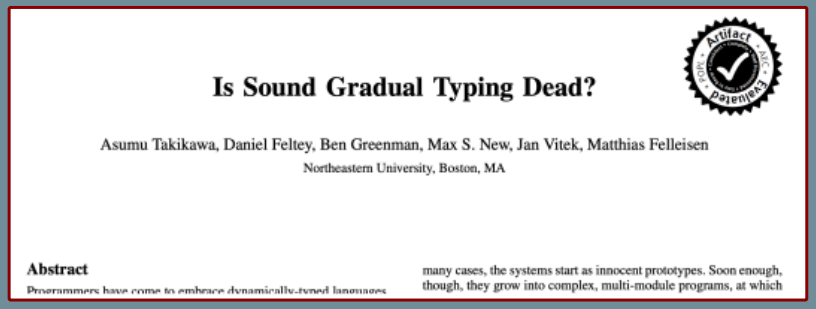
Programmers have come to embrace dynamically-typed languages

many cases, the systems start as innocent prototypes. Soon enough, though, they grow into complex, multi-module programs, at which

How to find??

Old Problem, New Idea

popl'16: 10x slowdowns are common,
but fast points exist!



Is Sound Gradual Typing Dead?

Asumu Takikawa, Daniel Feltey, Ben Greenman, Max S. New, Jan Vitek, Matthias Felleisen

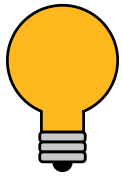
Northeastern University, Boston, MA

Abstract

Programmers have come to embrace dynamically-typed languages

many cases, the systems start as innocent prototypes. Soon enough, though, they grow into complex, multi-module programs, at which

How to find??



Rational Programmer
method (icfp'21)

Gradual Types + Costs

Gradual Types + Costs

```
def avg(g):
  return mean(get_column(g, "score"))
```

```
def mean(nums):
  ....
```

```
def get_column(table, col_name):
  ....
```

```
avg(quiz_1_grades)
avg(recipe_book)
avg(42)
```

Gradual Types + Costs

```
avg : Gradebook -> Num
def avg(g):
  return mean(get_column(g, "score"))
```

```
def mean(nums):
  ....
```

```
def get_column(table, col_name):
  ....
```

```
avg(quiz_1_grades)
avg(recipe_book)
avg(42)
```

Add types, code still runs

Gradual Types + Costs

```
avg : Gradebook -> Num
def avg(g):
  return mean(get_column(g, "score"))
```

```
avg(quiz_1_grades)
avg(recipe_book)
avg(42)
```

Gradual Types + Costs

```
avg : Gradebook -> Num
def avg(g):
  return mean(get_column(g, "score"))
```

```
avg(quiz_1_grades)
avg(recipe_book)
avg(42)
```

Type soundness ◀ Runtime checks

Gradual Types + Costs

```
avg : Gradebook -> Num
def avg(g):
  return mean(get_column(g, "score"))
```

```
avg(quiz_1_grades)   ✓
avg(recipe_book)     ■
avg(42)              ■
```

Type soundness ➤ Runtime checks

Costs depend ...

Guarded semantics

(**deep** types)

Contract @ boundary

Transient semantics

(**shallow** types)

Asserts in typed code

Gradual Types + Costs

```
avg : Gradebook -> Num
def avg(g):
  return mean(get_column(g, "score"))
```

```
avg(quiz_1_grades)   ✓
avg(recipe_book)     ■
avg(42)              ■
```

Type soundness ◀ Runtime checks

Costs depend ...

deep

check full gradebook

9x

shallow

check book shape, numbers

~1x

```
avg : Gradebook -> Num
def avg(g):
  return mean(get_column(g, "score"))
```

```
avg(quiz_1_grades)
avg(recipe_book)
avg(42)
```

```
avg : Gradebook -> Num
def avg(g):
  return mean(get_column(g, "score"))
```

```
avg(quiz_1_grades)
```

```
avg : Gradebook -> Num
def avg(g):
  return mean(get_column(g, "score"))
```

```
avg(quiz_1_grades)
```

deep

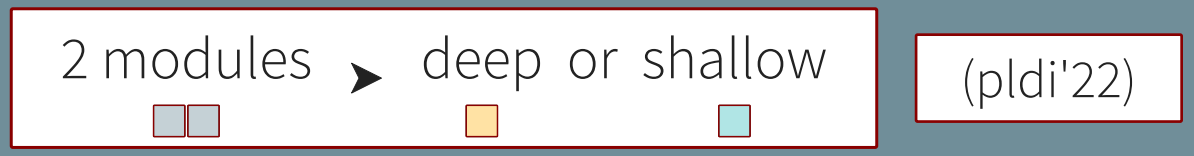
no boundaries!

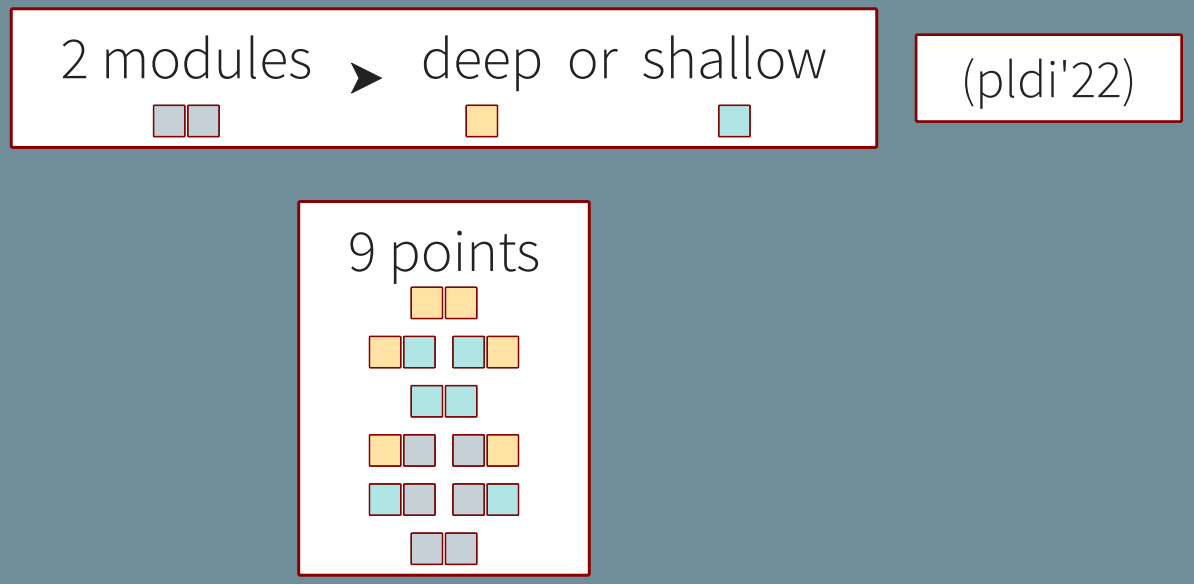
1x

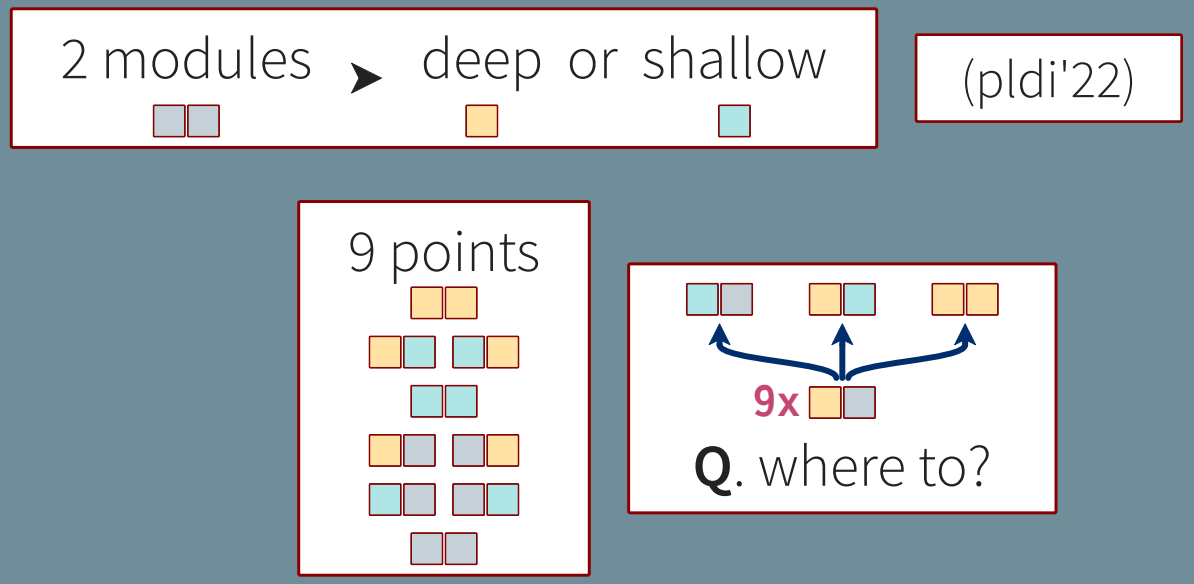
shallow

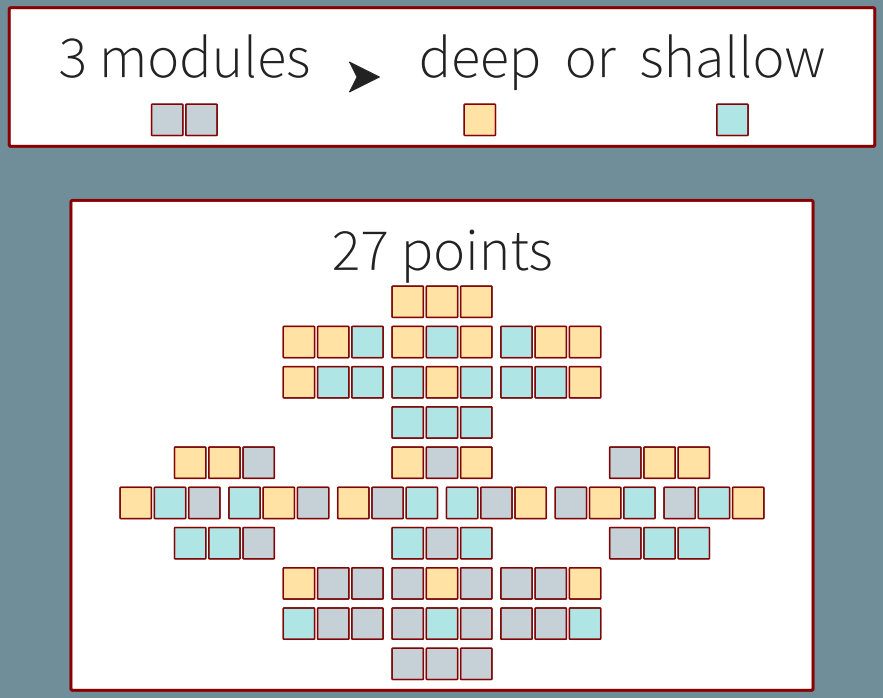
more types, more checks

2x

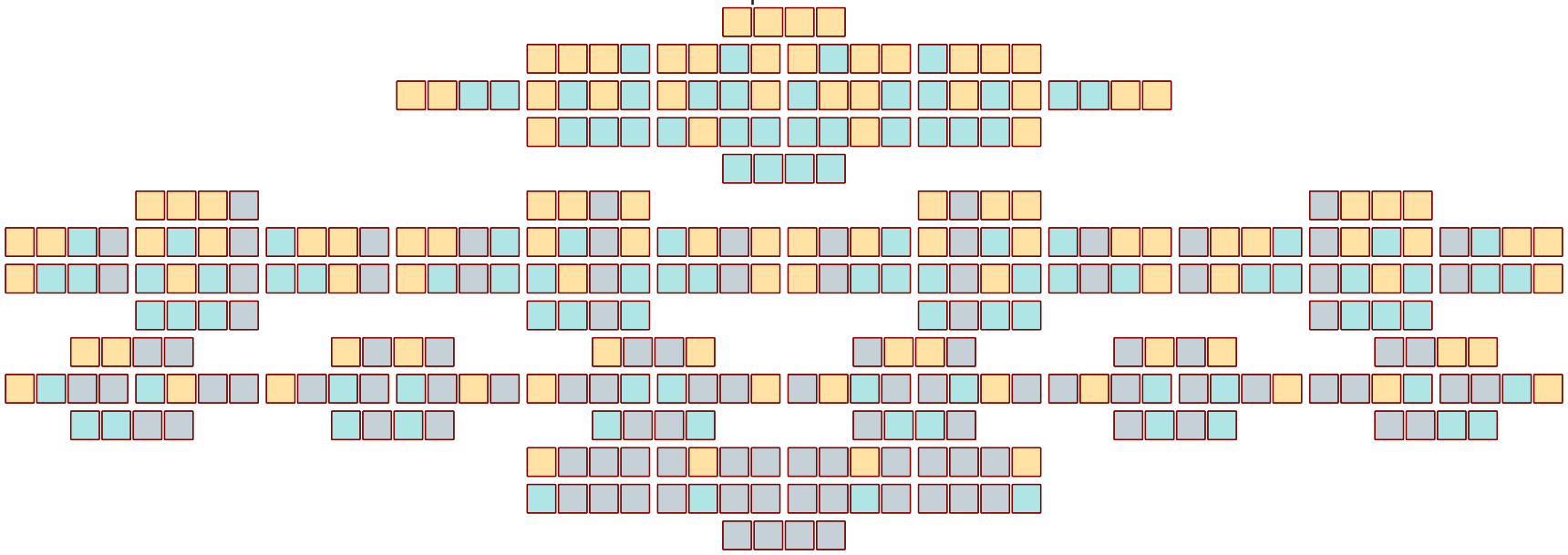

2 modules
deep or shallow
(pldi'22)

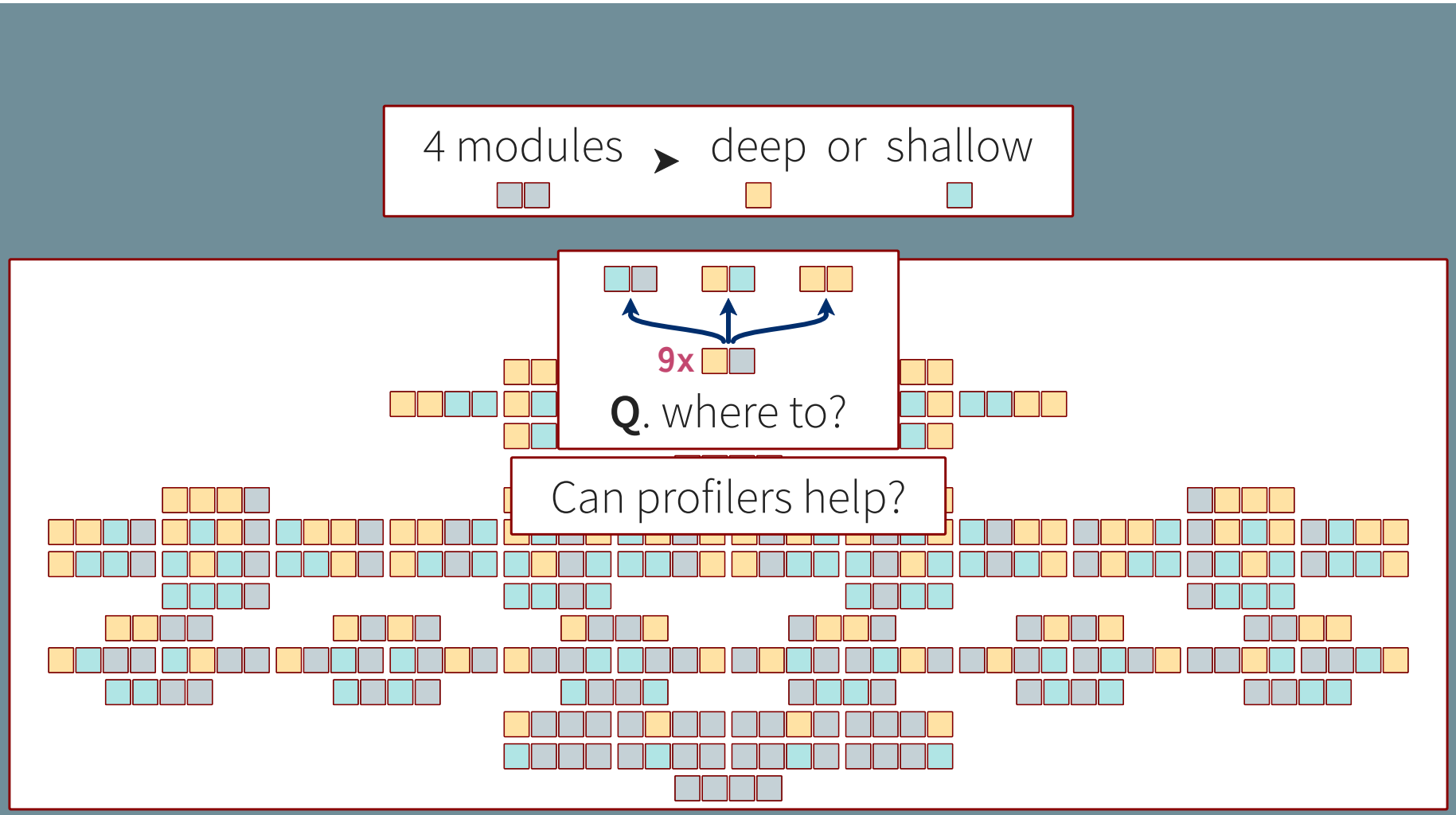

2 modules
deep or shallow
(pldi'22)
9 points


2 modules
deep or shallow
(pldi'22)
9 points
9x
Q. where to?


3 modules
deep or shallow
27 points

4 modules ➤ deep or shallow




4 modules
deep or shallow
9x
Q. where to?
Can profilers help?

Profilers

Profilers

Statistical Profiler

Contract Profiler

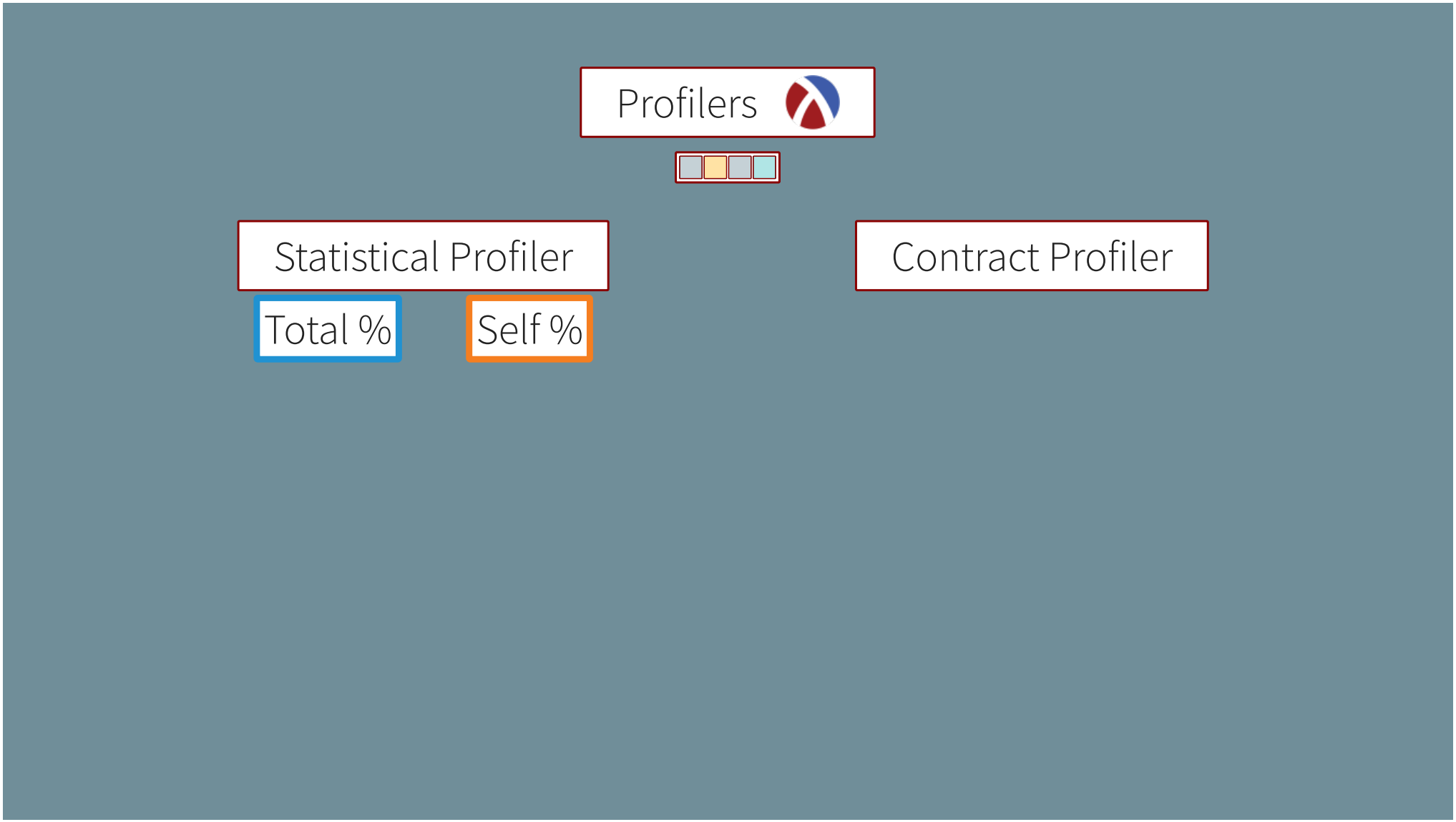
Profilers

Statistical Profiler

```
  Total cpu time observed: 1192ms (out of 1236ms)
  Number of samples taken: 23 (once every 52ms)

================================================
         ■          ■          Caller
 Idx    Total      Self       Name+src
        ms(pct)    ms(pct)    Callee
================================================
                              ??? [12]
                              evolve [17]
[17]  818(68.6%)    0(0.0%)   evolve main
                              evolve [17]
                              shuffle-vector [19]
                              death-birth [18]
                              ??? [20]
------------------------------------------------
                              match-up* [22]
                              shuffle-vector [19]
[24]  152(12.7%)  152(12.7%)  contract-wrapper
------------------------------------------------
```

Contract Profiler


Profilers
Statistical Profiler
Total %
Self %
Contract Profiler

Profilers

Statistical Profiler

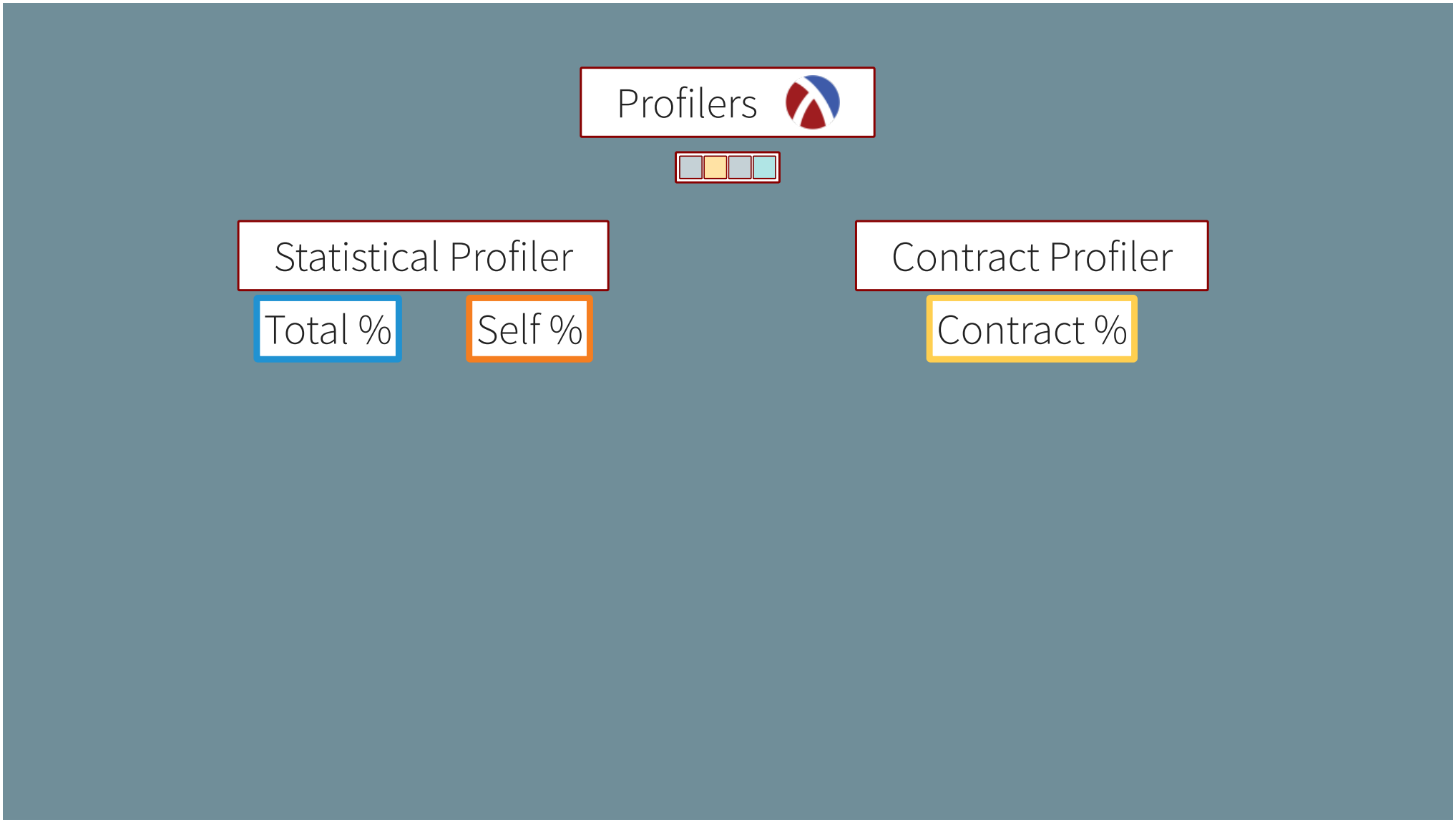
Total %

Self %

Contract Profiler

```
cpu time: 984 real time: 984 gc time: 155
Running time is 18.17% contracts
253/1390 ms

(interface:death-birth pop main)
  142 ms
  (->* ((cons/c (vectorof automaton?)
                (vectorof automaton?))
        any/c)
       (#:random any/c)
       (cons/c (vectorof automaton?)
               (vectorof automaton?)))
(interface:match-up* pop main)
  81.5 ms
  (-> ....)
(interface:population-payoffs pop main)
  29 ms
  (-> ....)
```


Profilers
Statistical Profiler
Total %
Self %
Contract Profiler
Contract %

Deep types
Contract @ boundary

Contract %

Total %

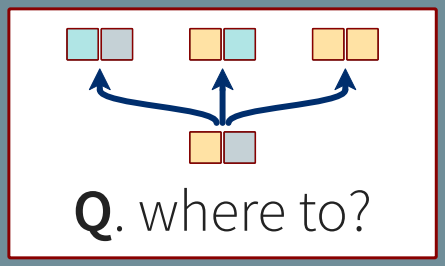
Self %

Shallow types
Asserts in typed code

Contract %

Total %

Self %



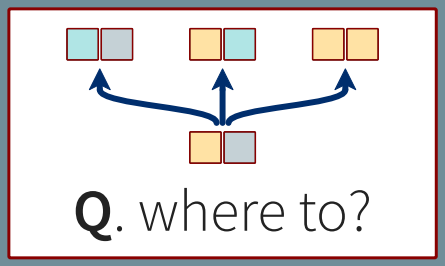
Q. where to?

Q. how to find a boundary?

Contract %

Total %

Self %



Q. where to?

A. Rational Programmer experiment

Q. how to find a boundary?

Contract %

Total %

Self %

Rational Programmer

Rational Programmer

Identify strategies, let them compete.

Rational Programmer

Identify strategies, let them compete.

Deep ()

Rational Programmer

Identify strategies, let them compete.

Deep ( )

Shallow

...

Rational Programmer

Identify strategies, let them compete.

Deep ( )

Shallow

Type-Aware Deep

1. 
2. 

Rational Programmer

Identify strategies, let them compete.

Deep ( )

Shallow

… 

Type-Aware Deep

1. 
2. 

Type-Aware Shallow

… 

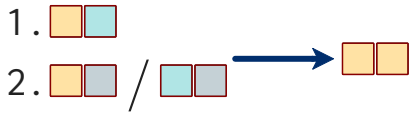
Rational Programmer

Identify strategies, let them compete.

Deep ( )

Shallow

Type-Aware Deep



Type-Aware Shallow

Lattice[S; D] count #typed, choose Deep or Shallow

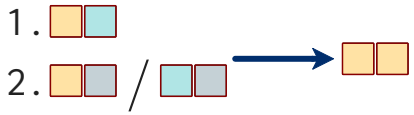
Rational Programmer

Identify strategies, let them compete.

Deep ( )

Shallow

Type-Aware Deep



Type-Aware Shallow

Lattice[S; D] count #typed, choose Deep or Shallow

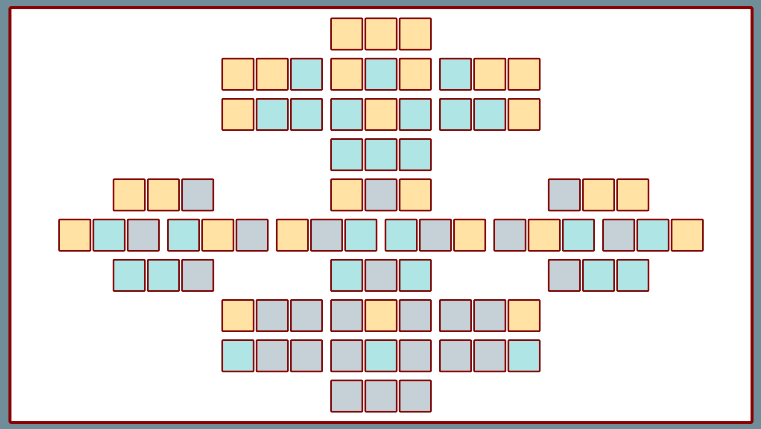
null, **pldi22** = baselines

Rational Programmer

Identify strategies, let them compete.

Rational Programmer

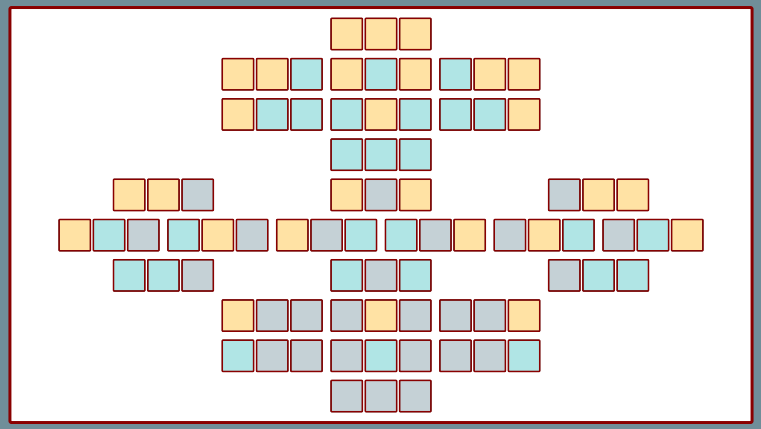
Identify strategies, let them compete.



For all starting points,
Goal = **path** to a fast config

Rational Programmer

Identify strategies, let them compete.



For all starting points,
Goal = **path** to a fast config

strict = faster each step

k **loose** = k slower steps

99x ➤ 99x ➤ 3x ➤ **1x**

3x ➤ **99x** ➤ **1x**

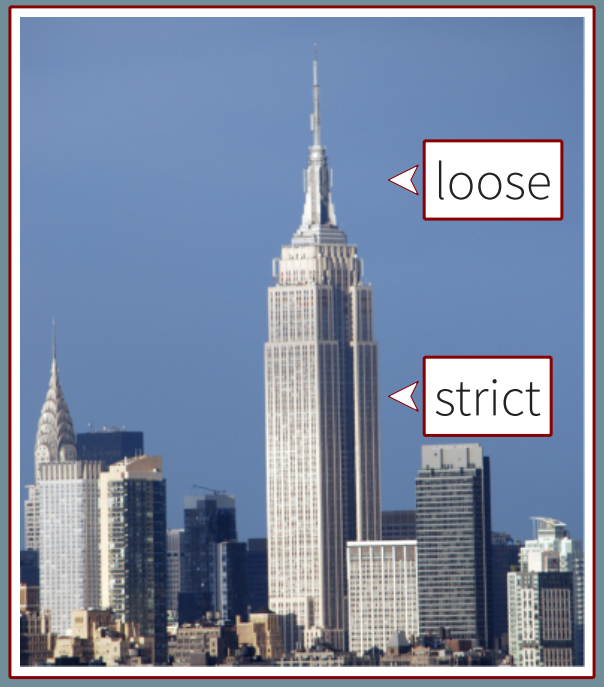
Dataset

16	GTP Benchmarks
116 K	starting points
1.2 M	measurements
5 GB	output
10	months on CloudLab

How often do the strategies succeed?

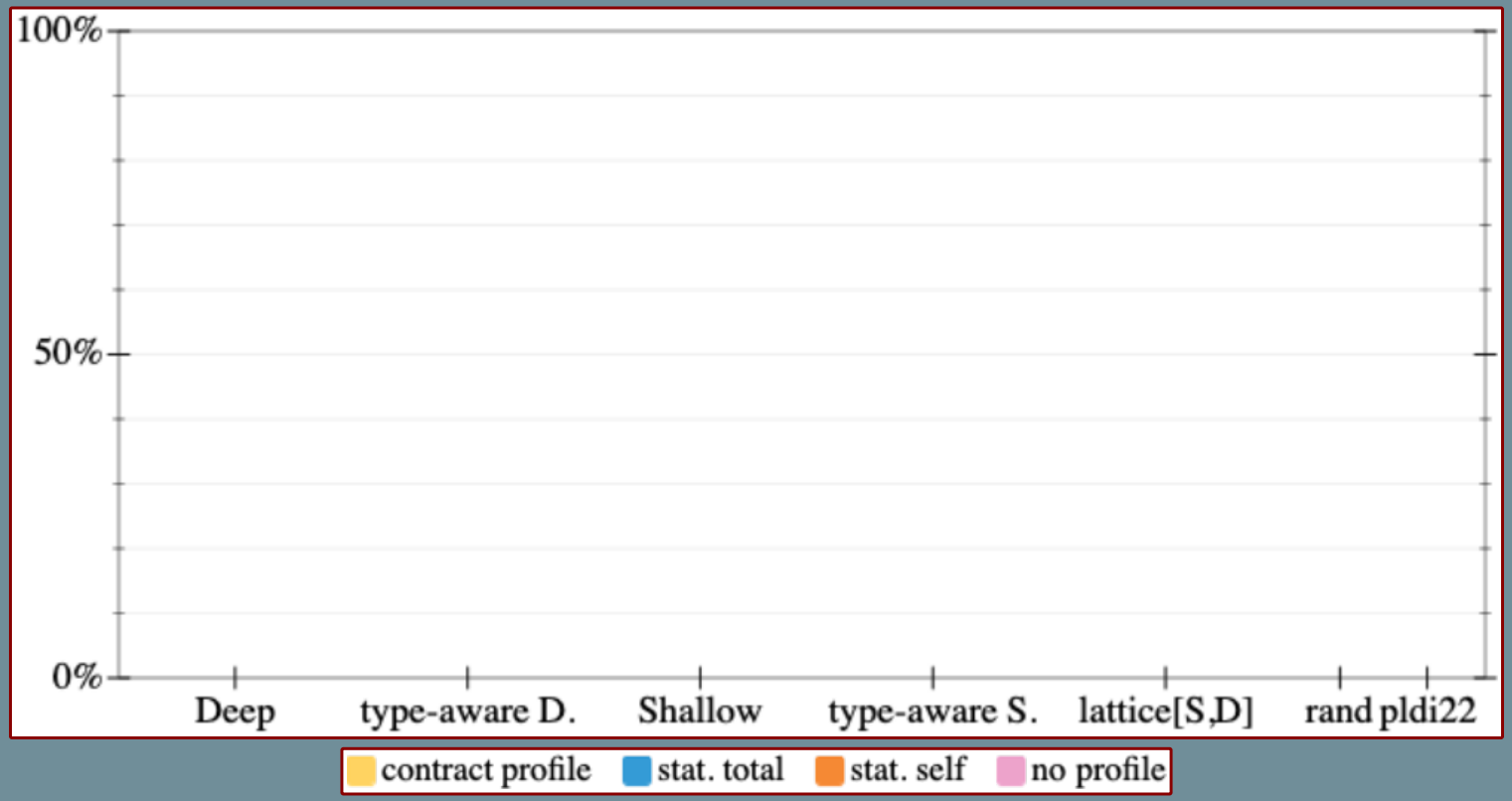
How often do the strategies succeed?

How often do the strategies succeed?



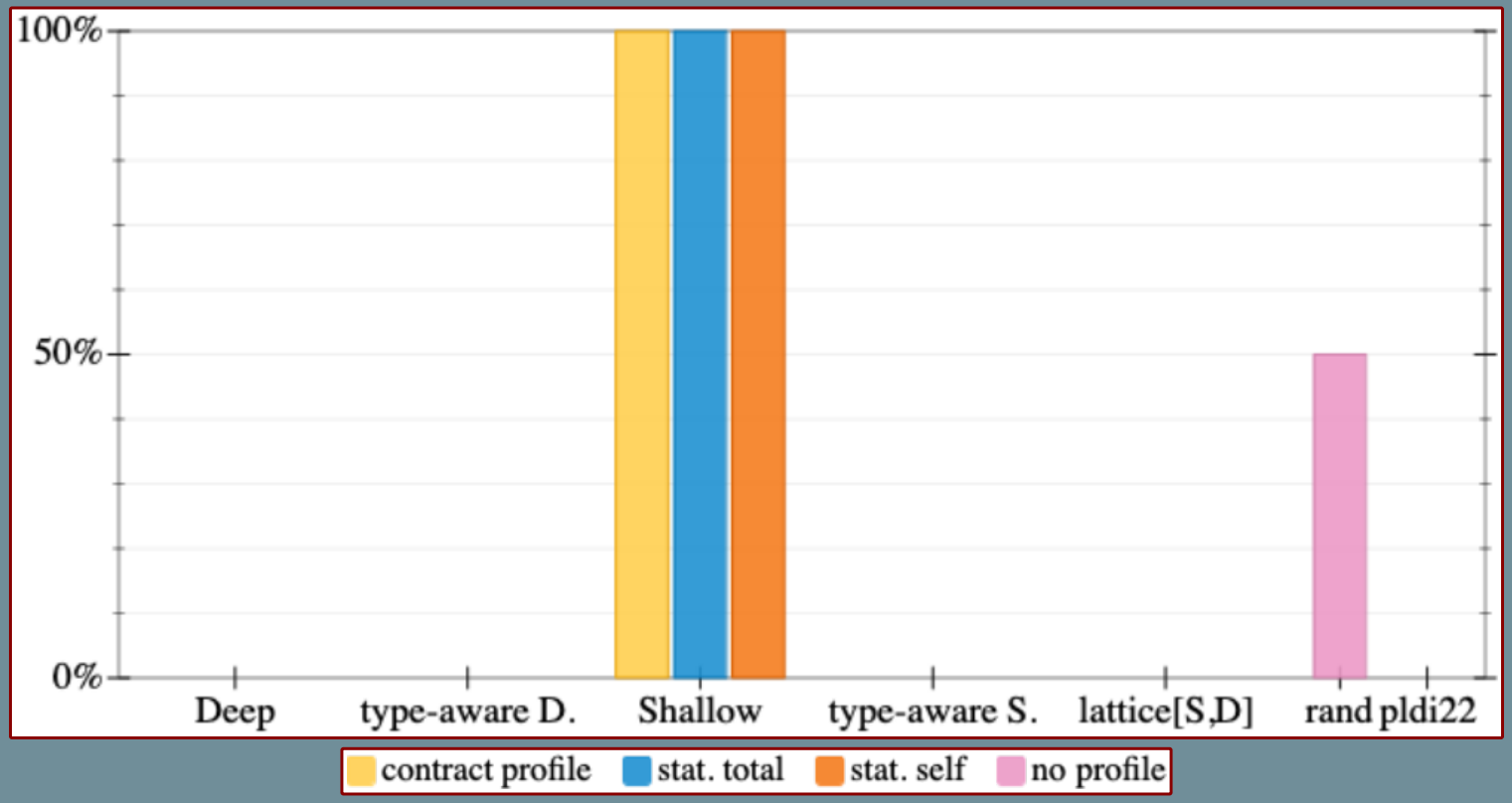
How often do the strategies succeed?

X = strategies, Y = % scenarios



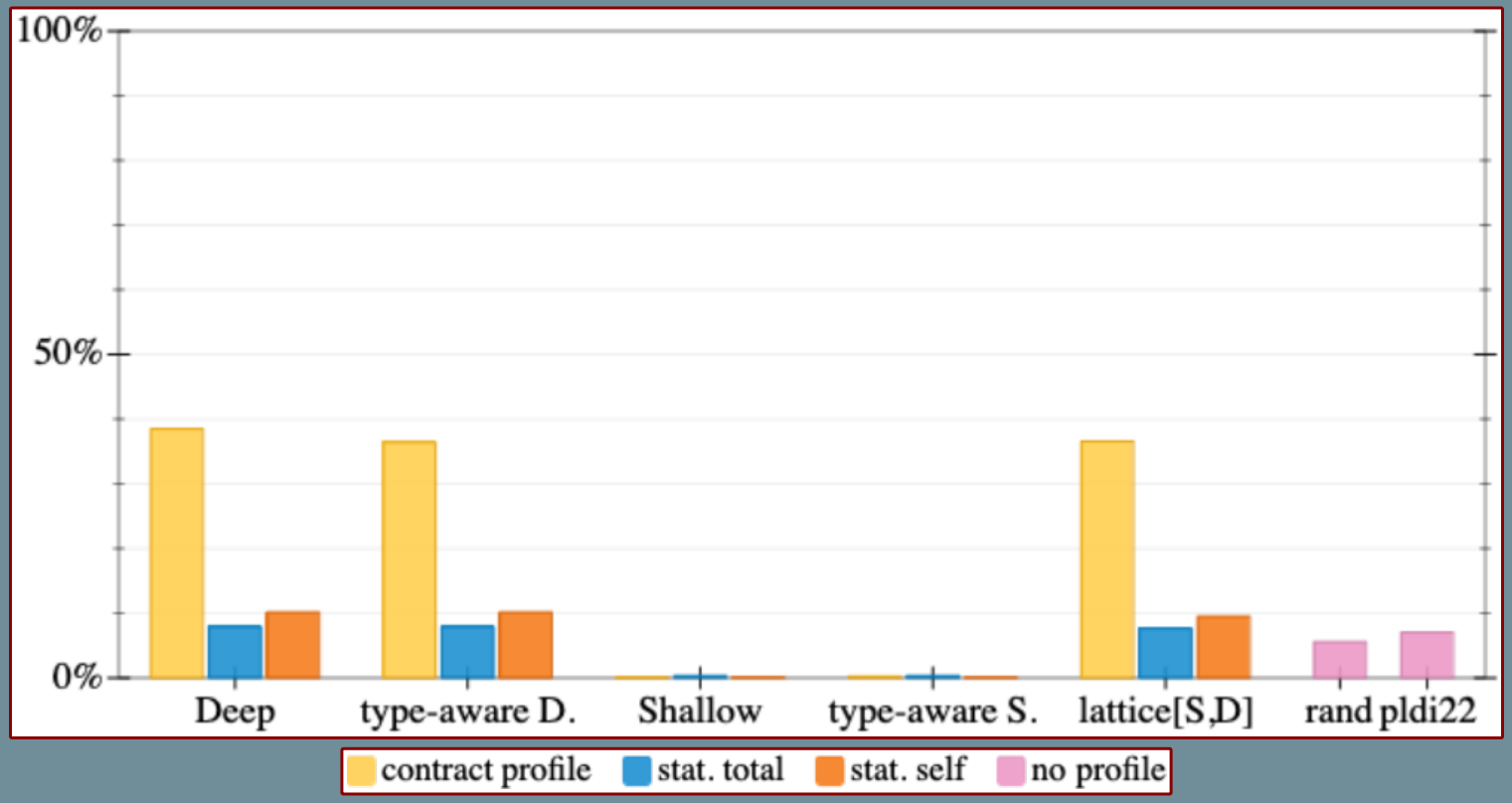
How often do the strategies succeed?

example data



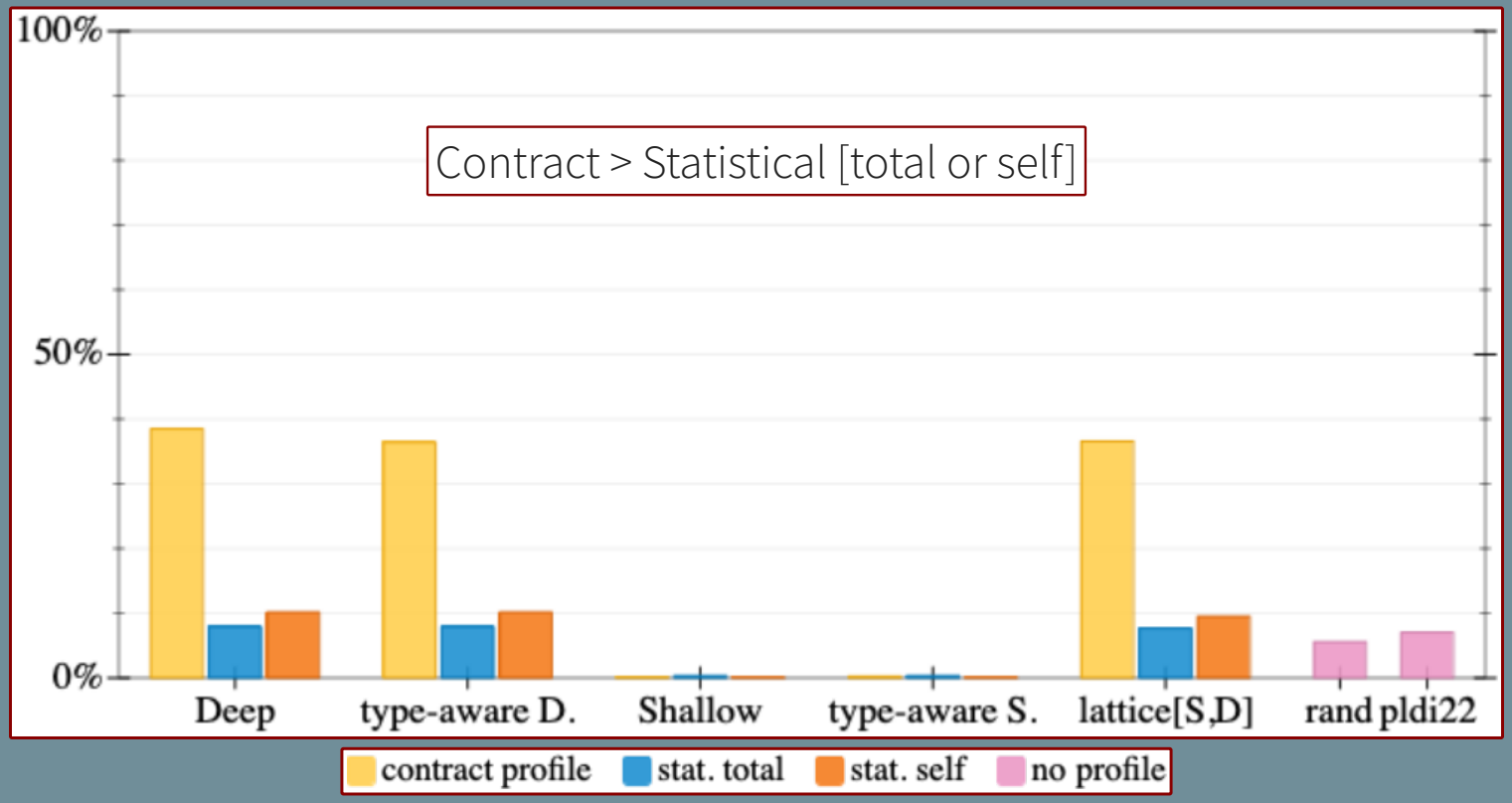
How often do the strategies succeed?

strict success



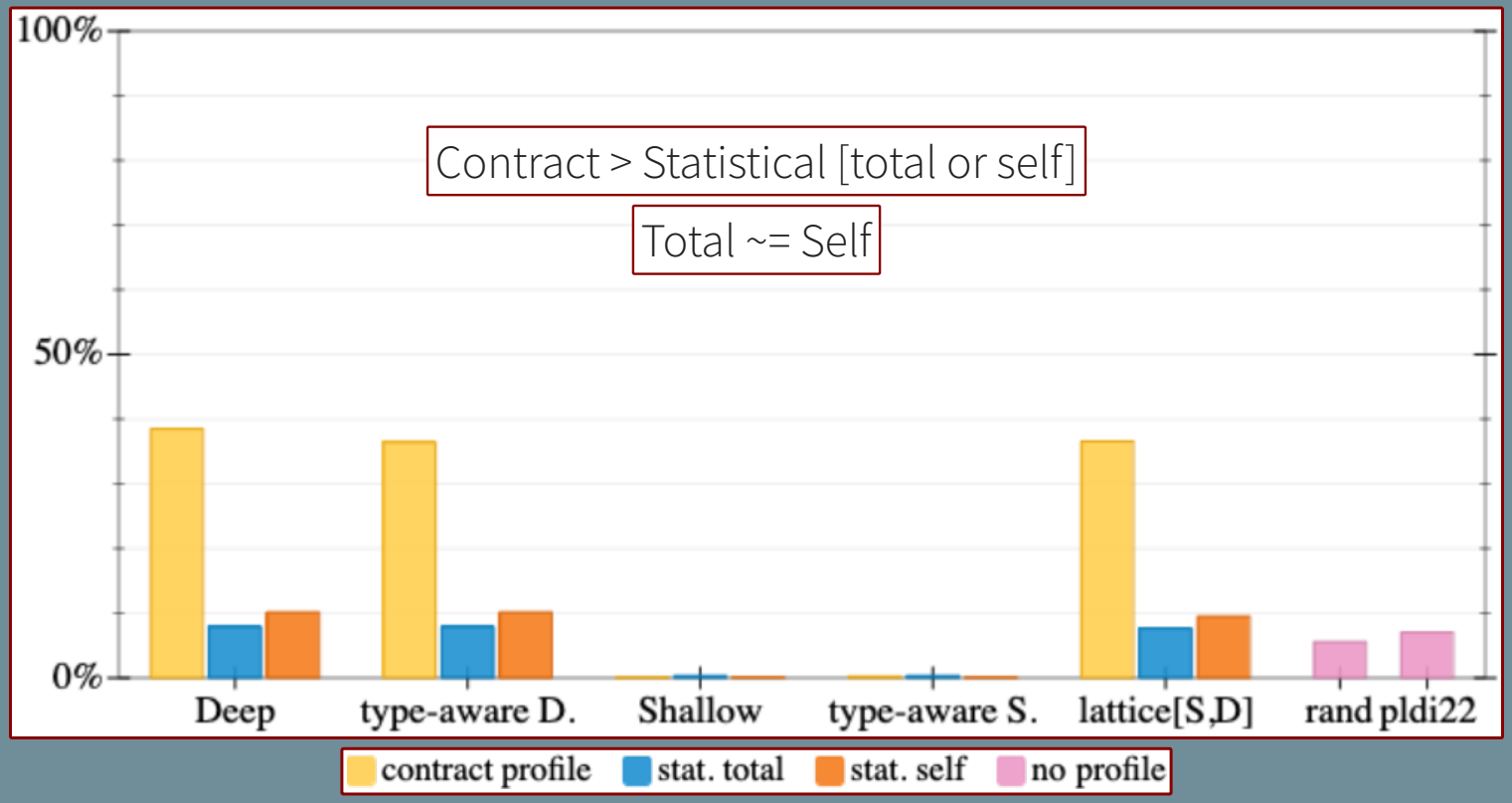
How often do the strategies succeed?

strict success



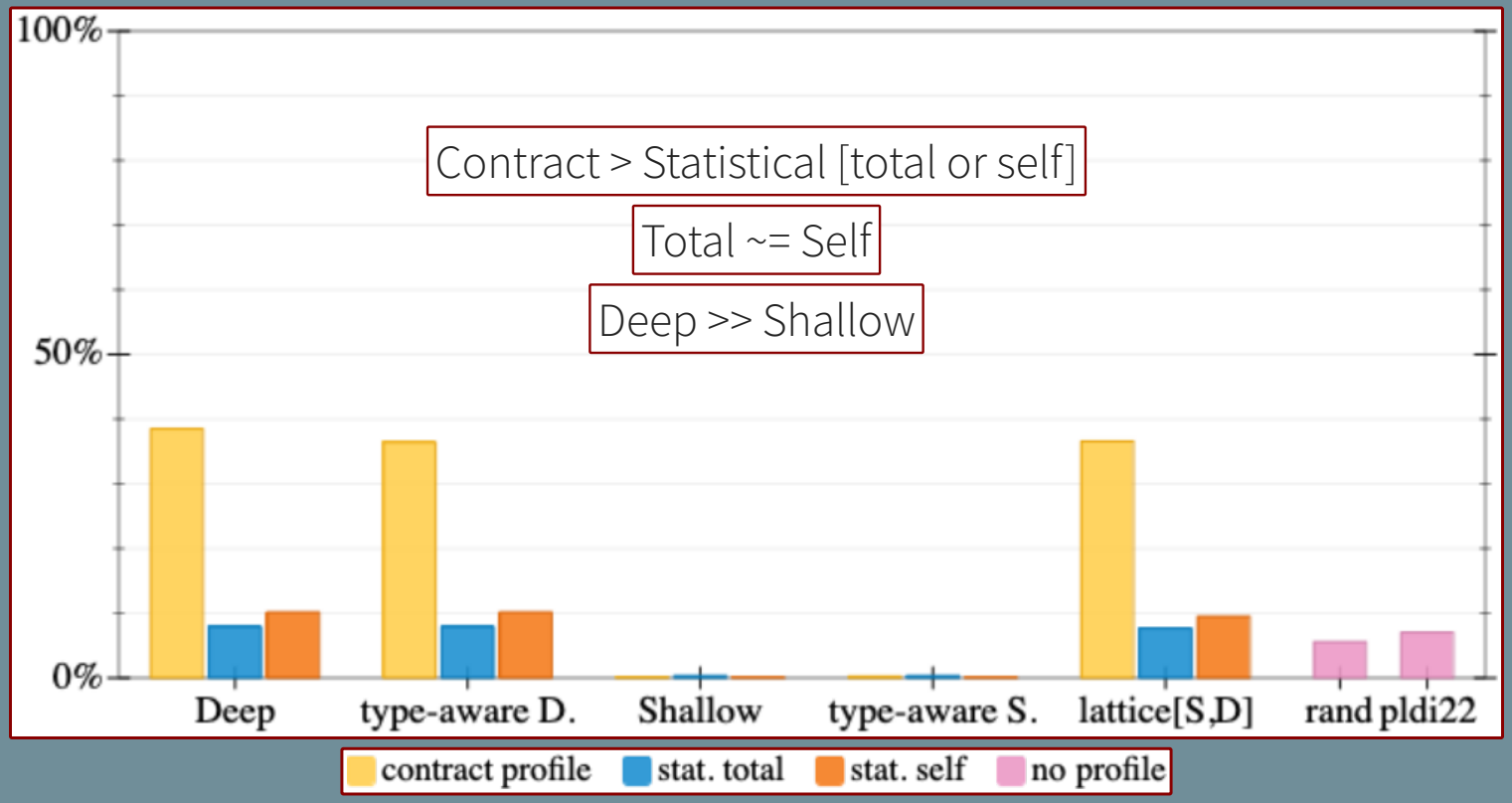
How often do the strategies succeed?

strict success



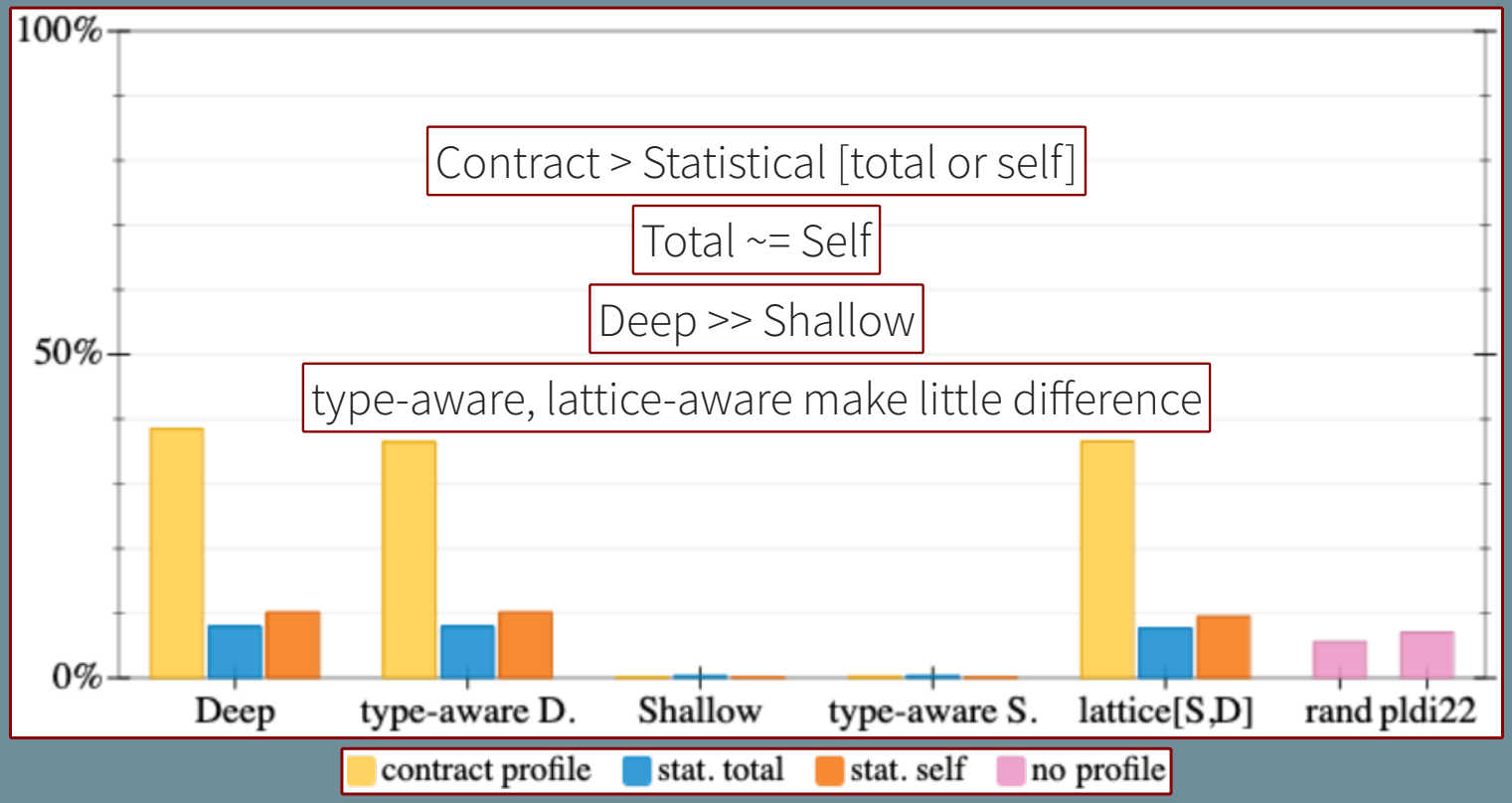
How often do the strategies succeed?

strict success



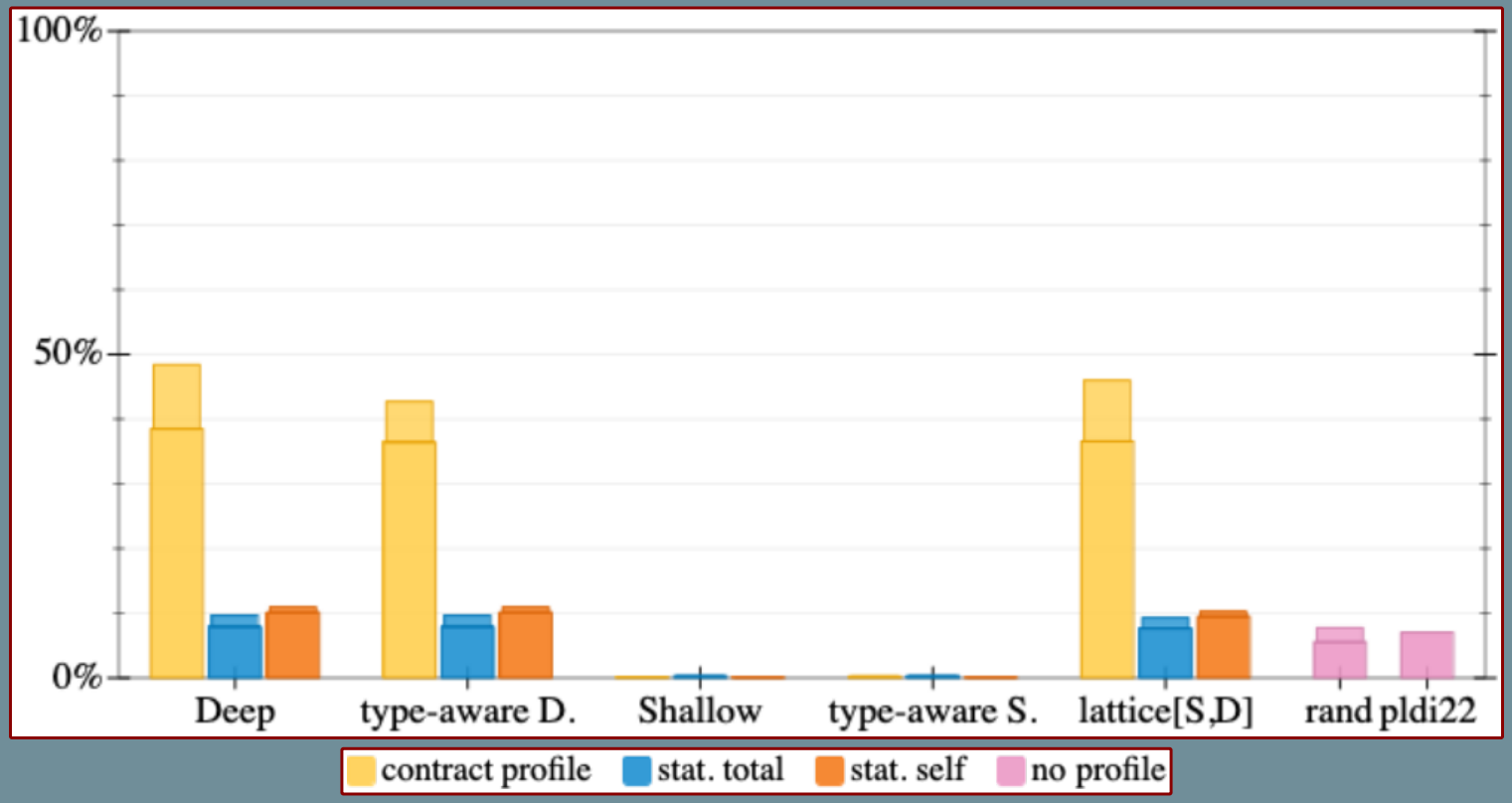
How often do the strategies succeed?

strict success



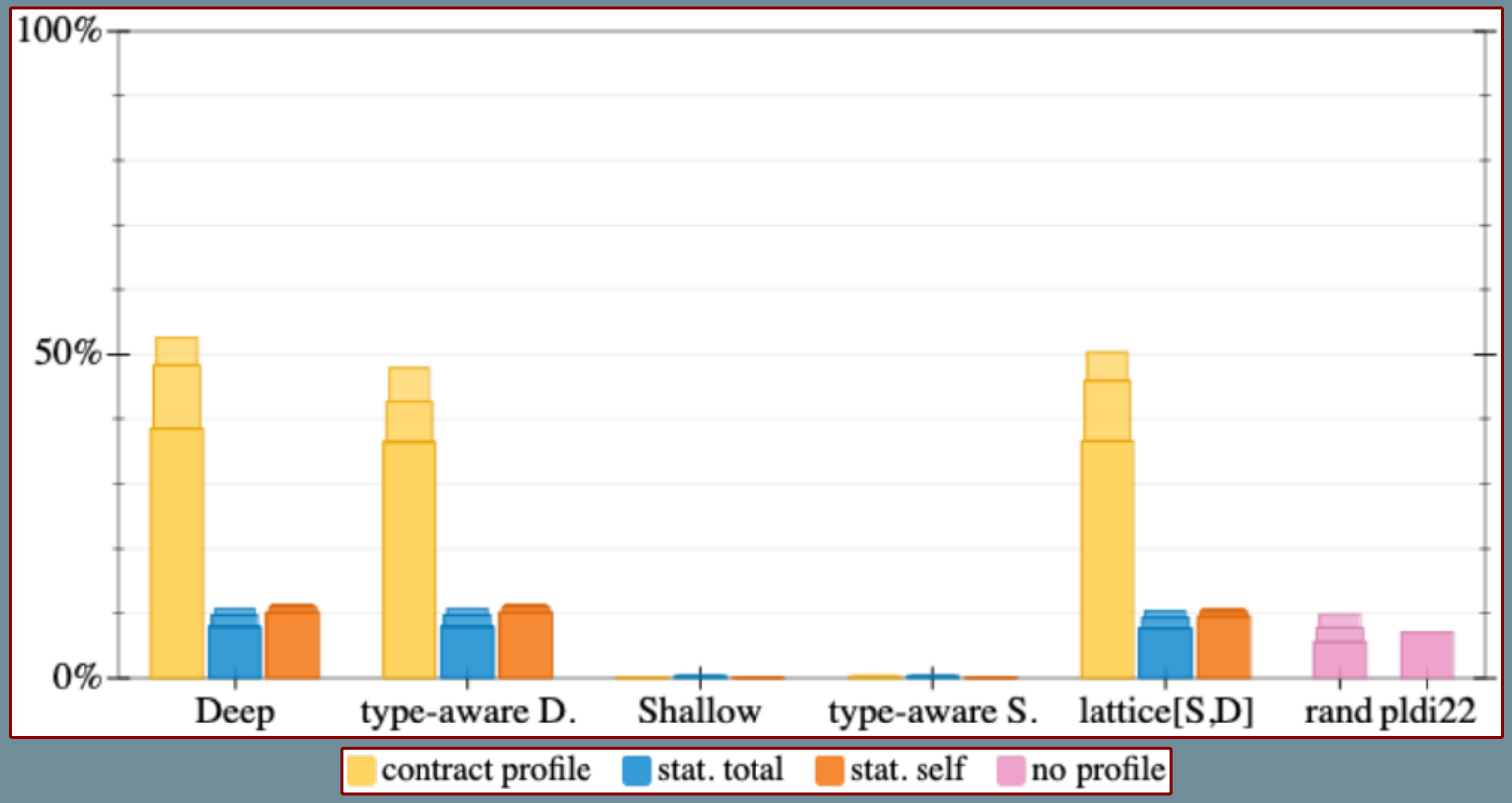
How often do the strategies succeed?

strict success | 1 loose



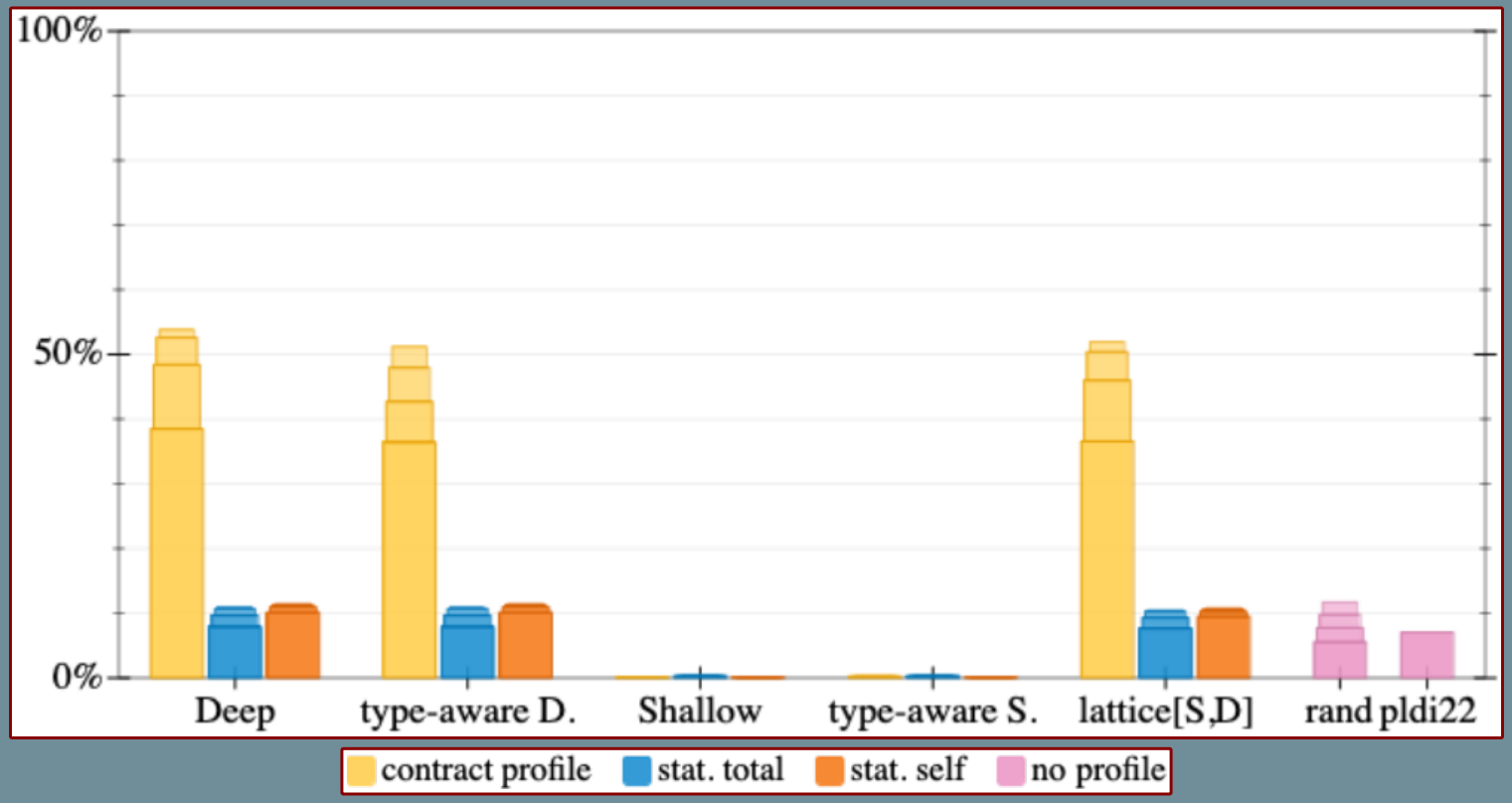
How often do the strategies succeed?

strict success | 1 loose | 2 loose



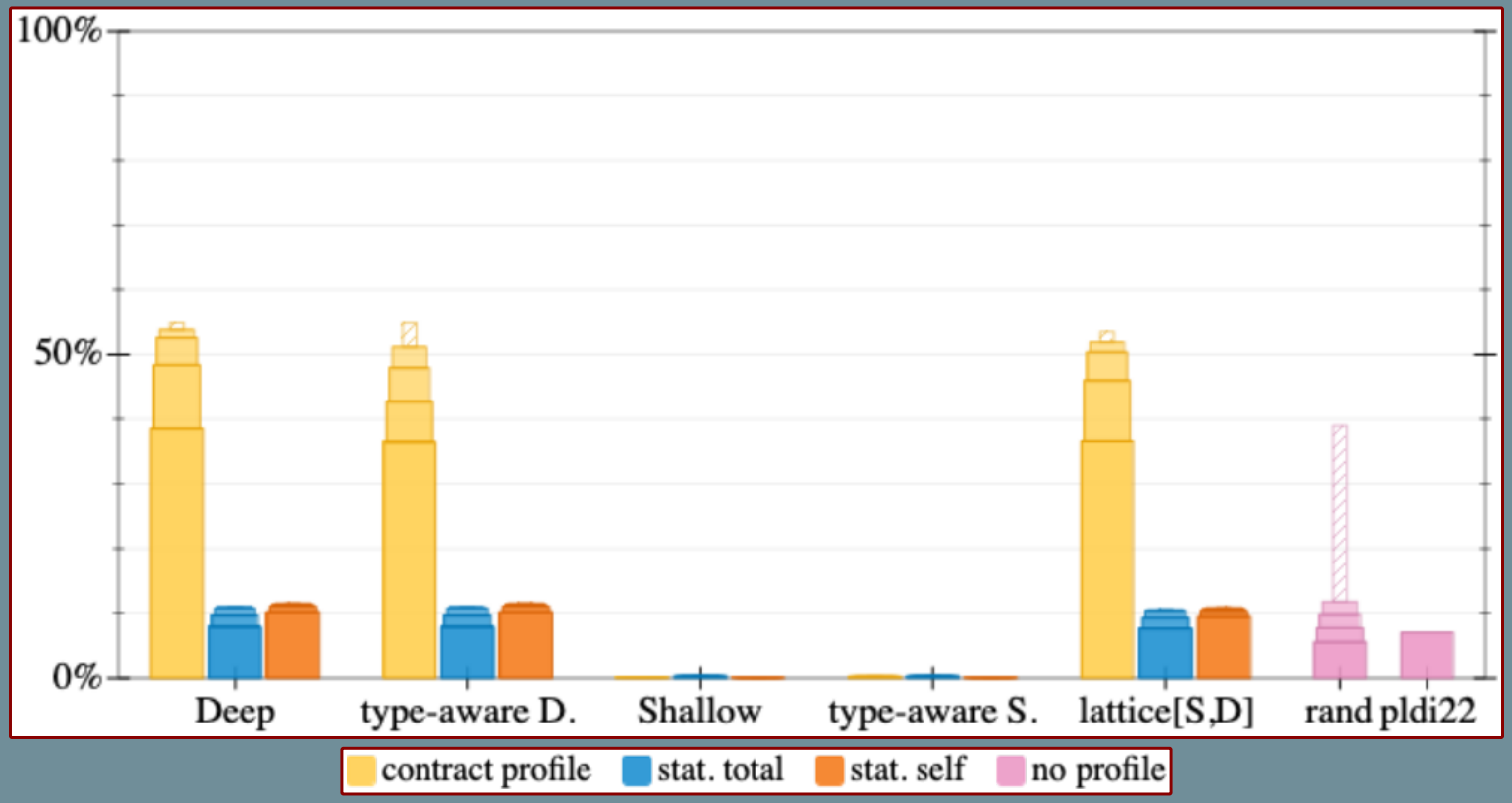
How often do the strategies succeed?

strict success | 1 loose | 2 loose | 3 loose



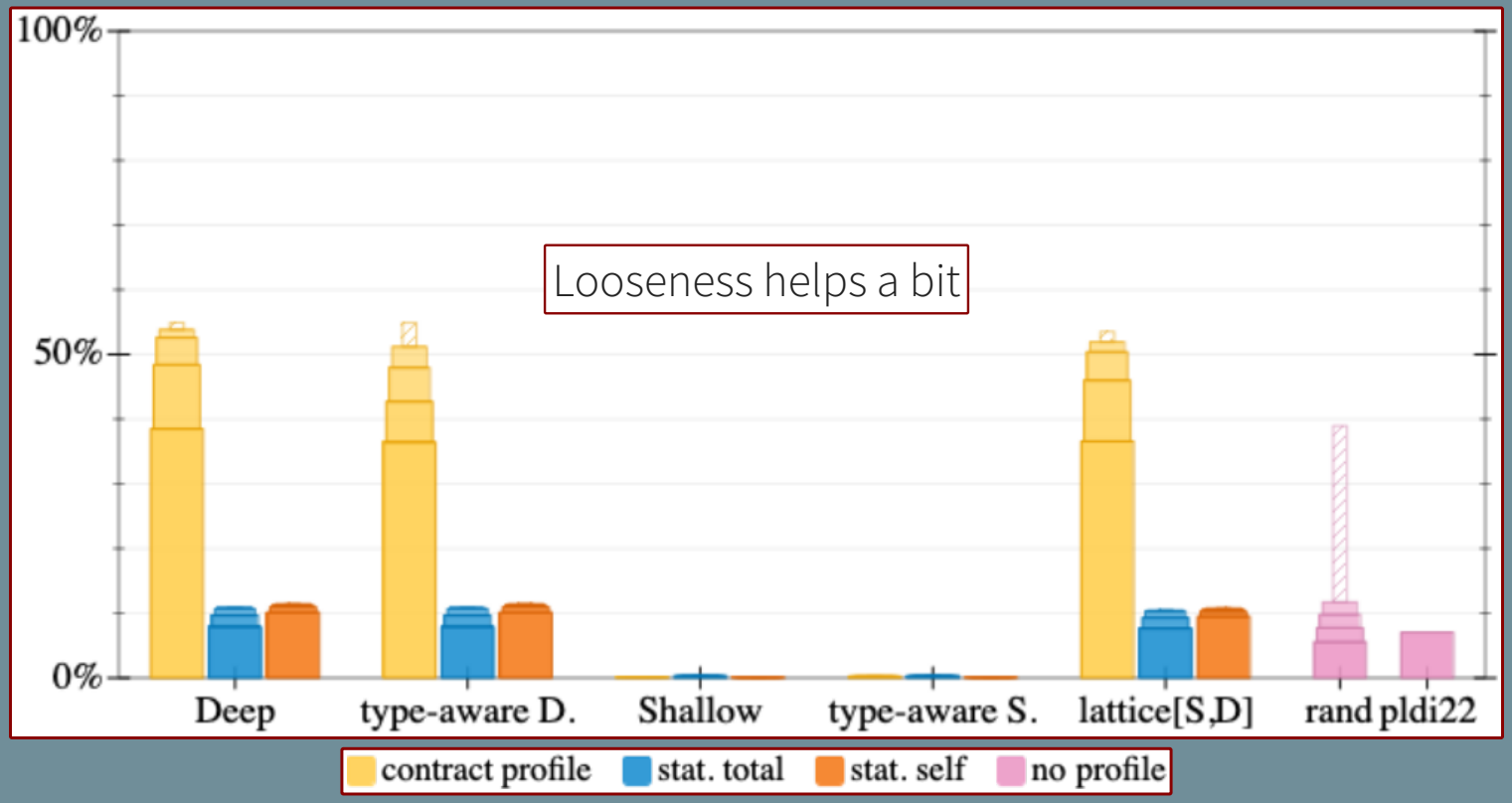
How often do the strategies succeed?

strict success | 1 loose | 2 loose | 3 loose | N loose



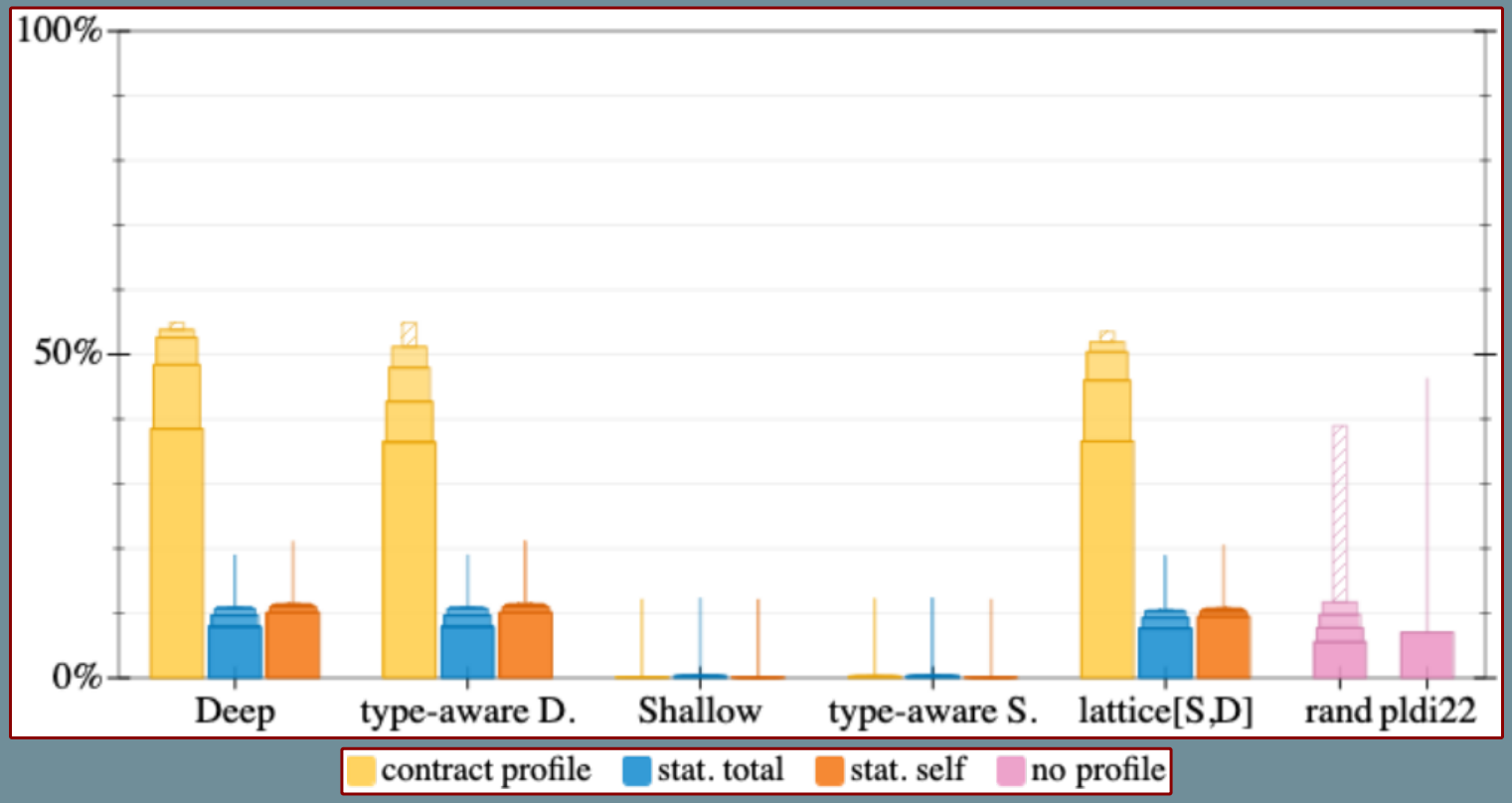
How often do the strategies succeed?

strict success | 1 loose | 2 loose | 3 loose | N loose



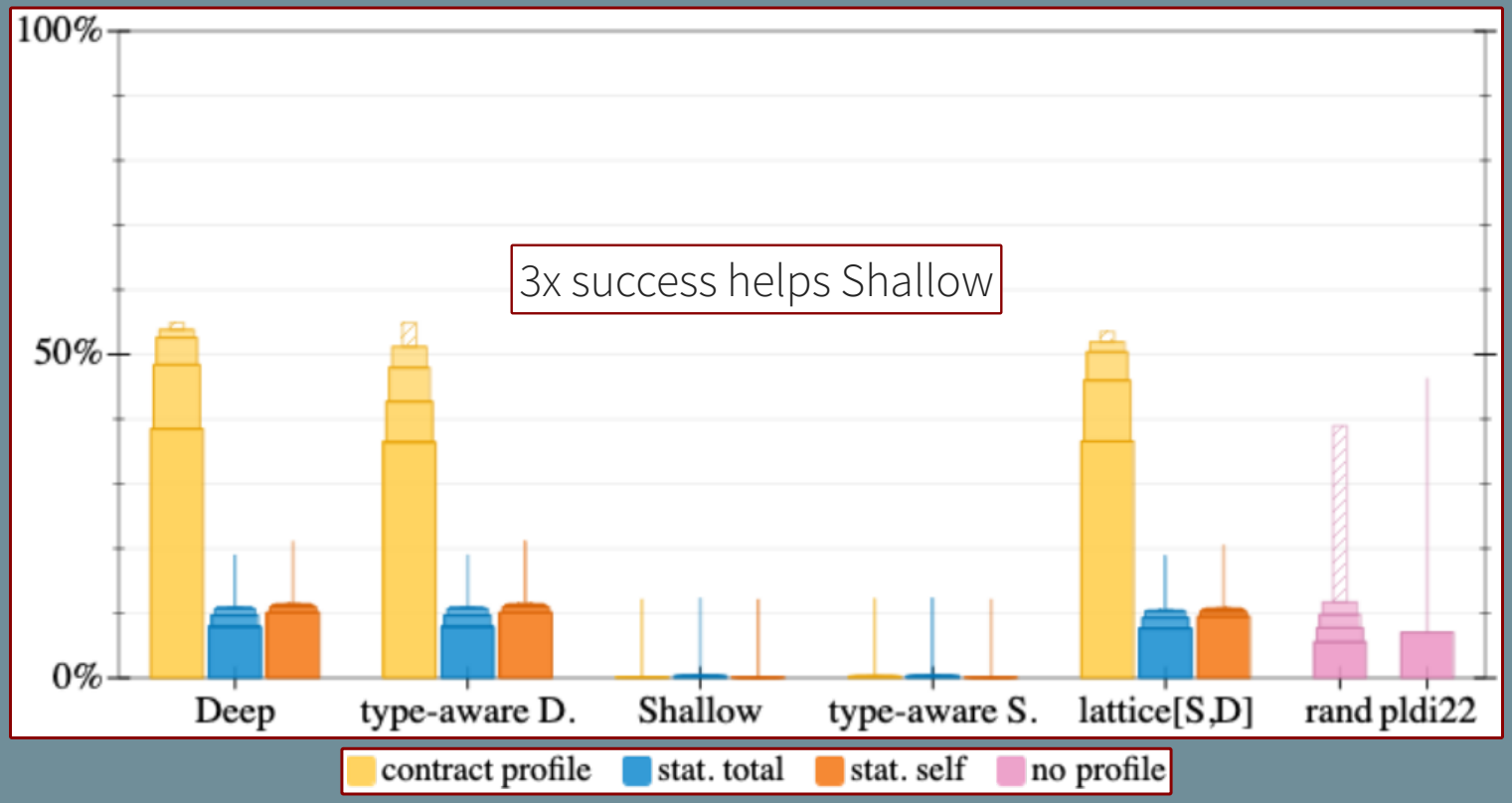
How often do the strategies succeed?

strict success | 1 loose | 2 loose | 3 loose | N loose | strict 3x



How often do the strategies succeed?

strict success | 1 loose | 2 loose | 3 loose | N loose | strict 3x



Takeaways

Takeaways

* **contract** profiling + **deep** types
  = **best** for type migration

Takeaways

* **contract** profiling + **deep** types
  = **best** for type migration
* shallow types do not help

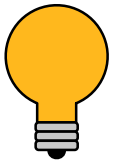
Takeaways

* **contract** profiling + **deep** types
  = **best** for type migration

* shallow types do not help

Q. hybrid strategies, shallow profilers?

Takeaways

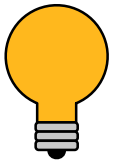


* the **rational programmer** method
  enables rigorous **experiments**

* **contract** profiling + **deep** types
  = **best** for type migration

* shallow types do not help

Takeaways



* the **rational programmer** method enables rigorous **experiments**

errors testing?
perf debugging?

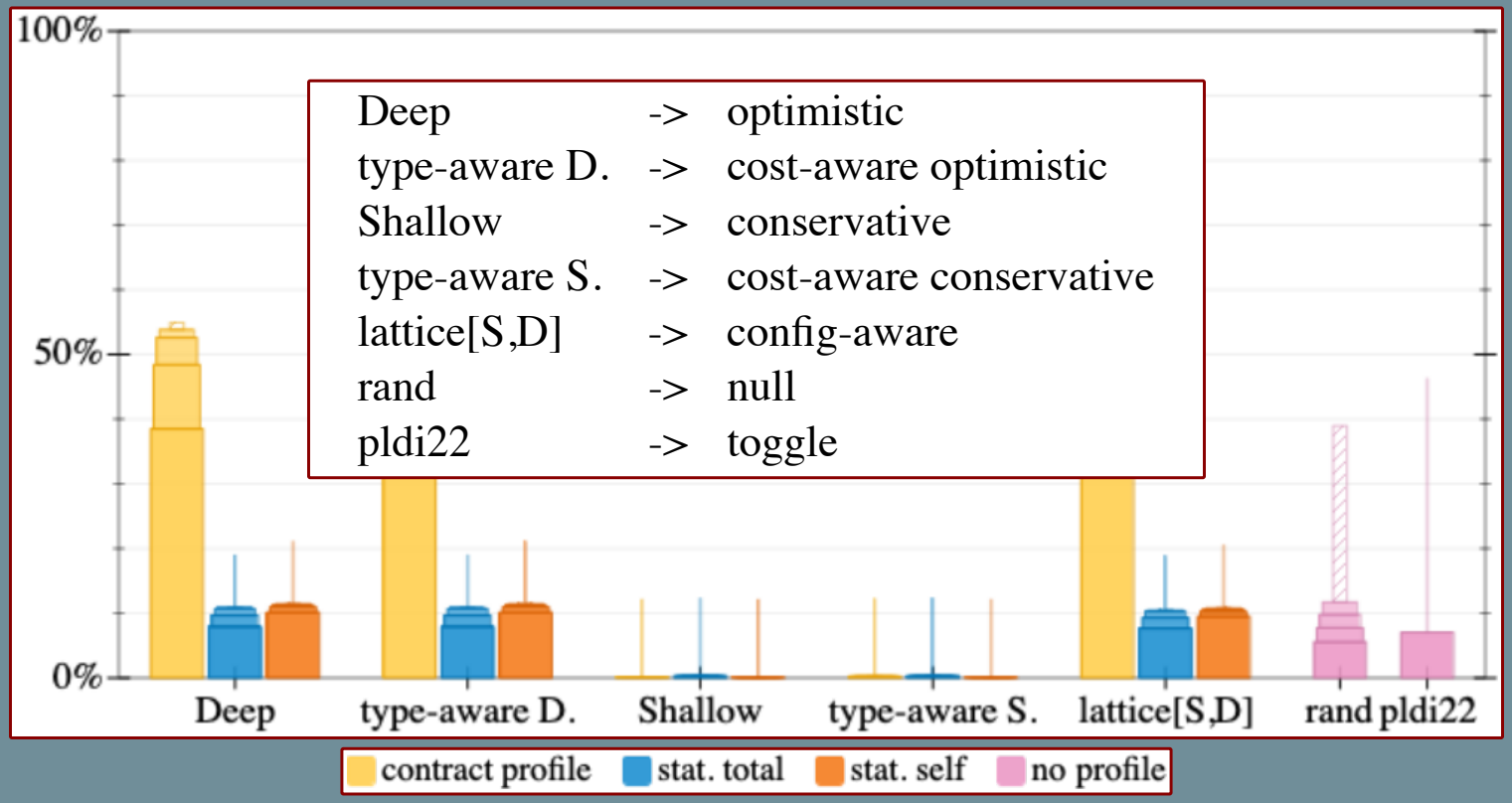
* **contract** profiling + **deep** types = **best** for type migration

* shallow types do not help

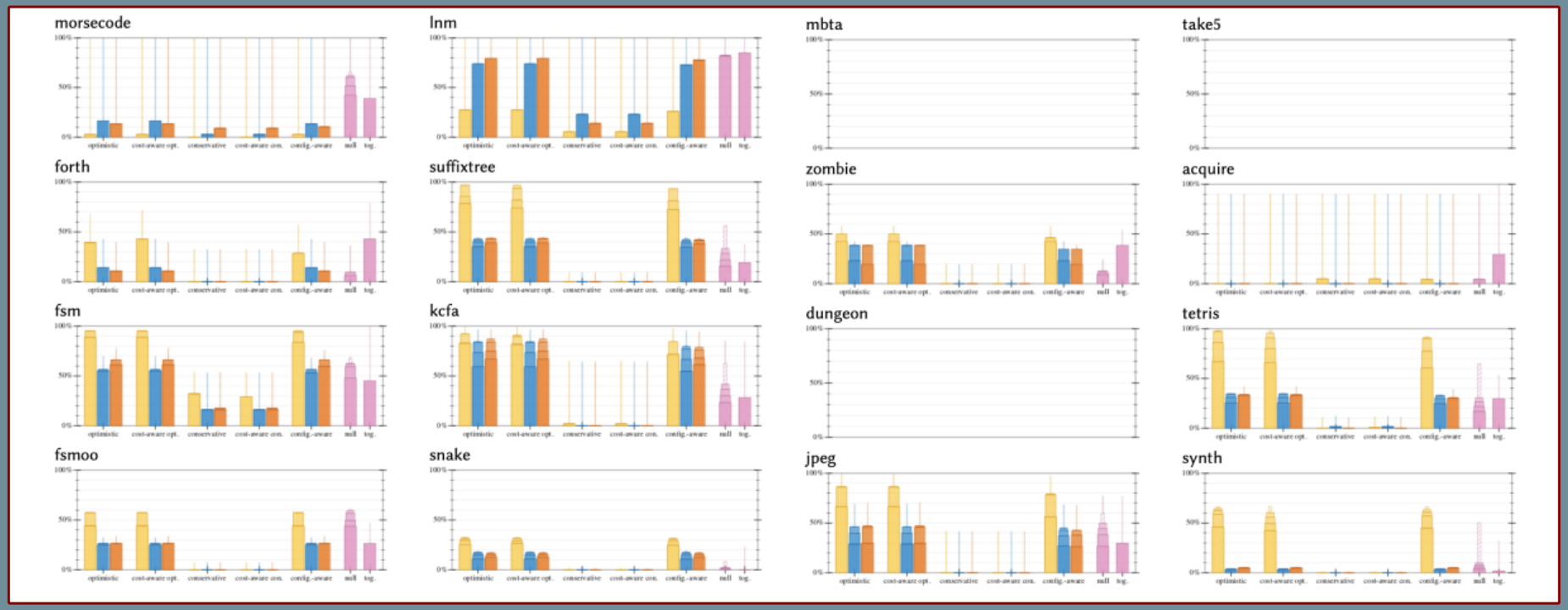
https://github.com/bennn/rational-deep-shallow

Translation: talk -> paper

strict success | 1 loose | 2 loose | 3 loose | N loose | strict 3x



Skylines per Benchmark

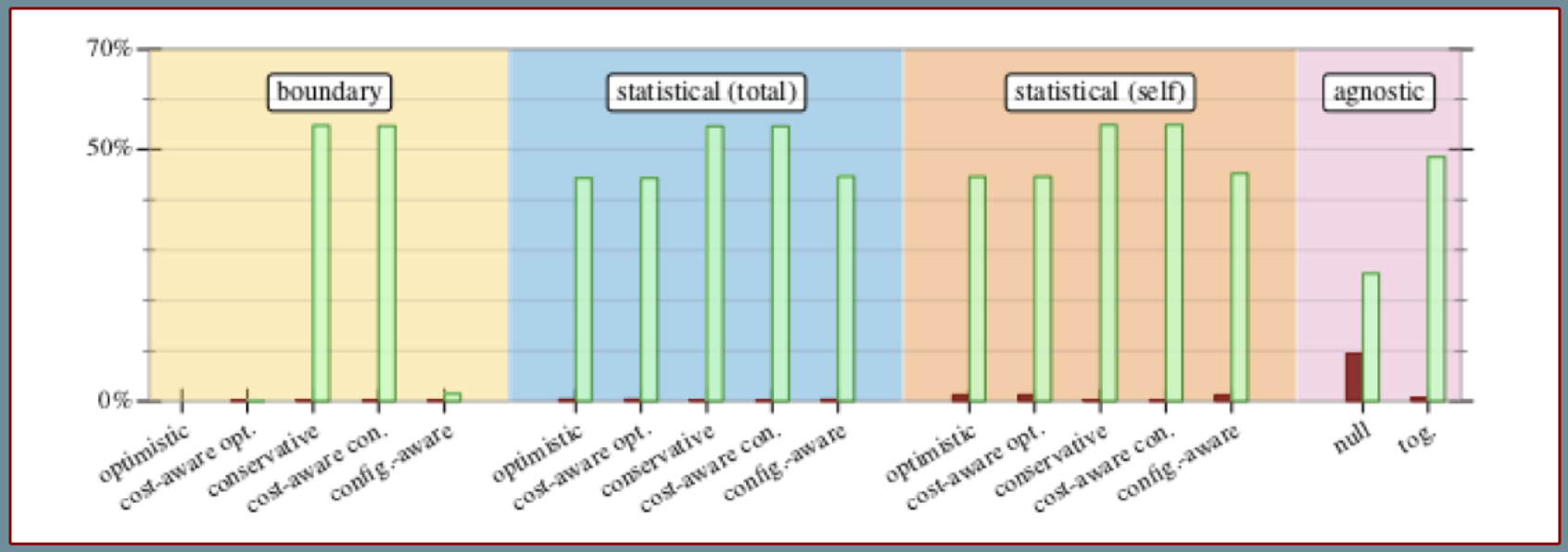


Hopeful Scenarios

Table 3. How many scenarios can possibly reach 1x without removing types?

Benchmark	# Scenario	% Hopeful	Benchmark	# Scenario	% Hopeful
morsecode	67	100.00 %	lnm	295	100.00 %
forth	76	36.84 %	suffixtree	718	100.00 %
fsm	62	100.00 %	kcfa	2,031	100.00 %
fsmoo	68	100.00 %	snake	6,559	100.00 %
mbta	72	0.00 %	take5	6,558	0.00 %
zombie	74	35.14 %	acquire	19,532	5.45 %
dungeon	242	0.00 %	tetris	18,791	100.00 %
jpeg	230	100.00 %	synth	59,046	100.00 %

Opt-Boundary vs. the others



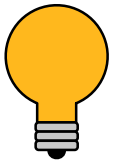
Where are the Fast Configs?

Table 4. Which levels of the migration lattice have any acceptable configurations?

Benchmark	#acceptable					
morsecode	1	2	4	4	3	
forth	1	2	1	1	0	
fsm	1	3	4	7	4	
fsmoo	1	2	4	2	4	
mbta	1	4	4	0	0	
zombie	1	2	3	1	0	
dungeon	1	0	0	0	0	0
jpeg	1	2	1	1	4	4

Benchmark	#acceptable by lattice level										
lnm	1	9	38	93	138	116	39				
suffixtree	1	1	0	0	1	4	4				
kcfa	1	8	22	33	24	24	29	15			
snake	1	0	0	0	0	0	0	0	1		
take5	1	2	0	0	0	0	0	0	0		
acquire	1	8	28	51	45	16	2	0	0	0	
tetris	1	12	56	121	169	128	118	133	112	42	
synth	1	1	0	0	0	0	0	0	0	0	1

Takeaways



* the **rational programmer** method
  enables rigorous **experiments**

* **contract** profiling + **deep** types
  = **best** for type migration

* shallow types do not help