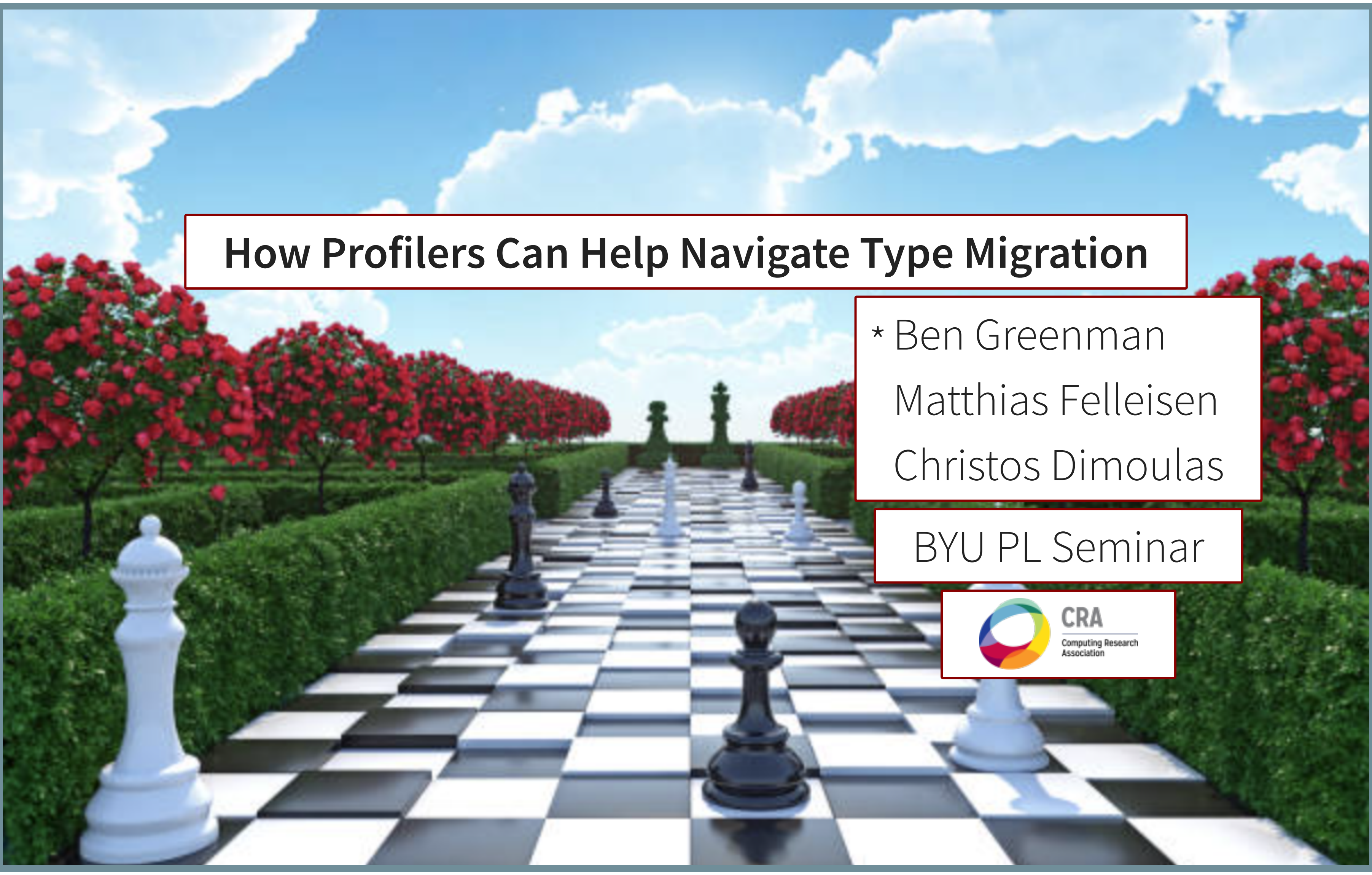


How Profilers Can Help Navigate Type Migration

* Ben Greenman

Matthias Felleisen

Christos Dimoulas

BYU PL Seminar

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**?

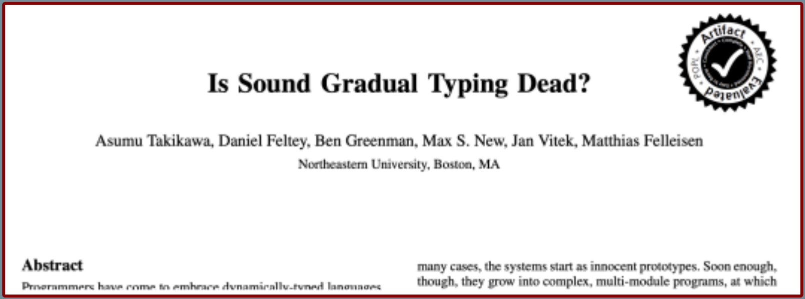
costs ~ gradual types

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

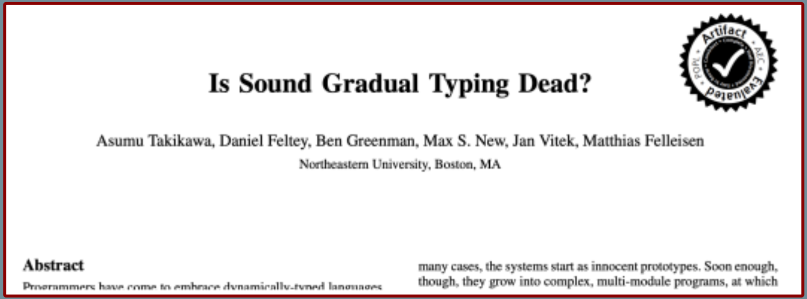
Programmers have come to embrace dynamically-typed languages

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

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 needs 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 needs Runtime checks

Costs depend ...

$$\begin{tikzcd} & \text{Costs depend ...} \arrow[dl, bend right] \arrow[dr, bend left] & \\ \text{Guarded semantics} & & \text{Transient semantics} \end{tikzcd}$$

Guarded semantics	Transient semantics
deep types	**shallow** types
Contract @ boundary	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 needs Runtime checks

Costs depend ...

$$\begin{tikzcd} & \text{Costs depend ...} \arrow[dl, bend right] \arrow[dr, bend left] & \\ \textbf{deep} & & \textbf{shallow} \end{tikzcd}$$

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	**shallow**
no boundaries!	more types, more checks
1x	2x

2 modules ➤ deep or shallow

(pldi'22)

2 modules ➤ deep or shallow

⬜⬜ 🟨 🟦

(pldi'22)

9 points

🟨🟨

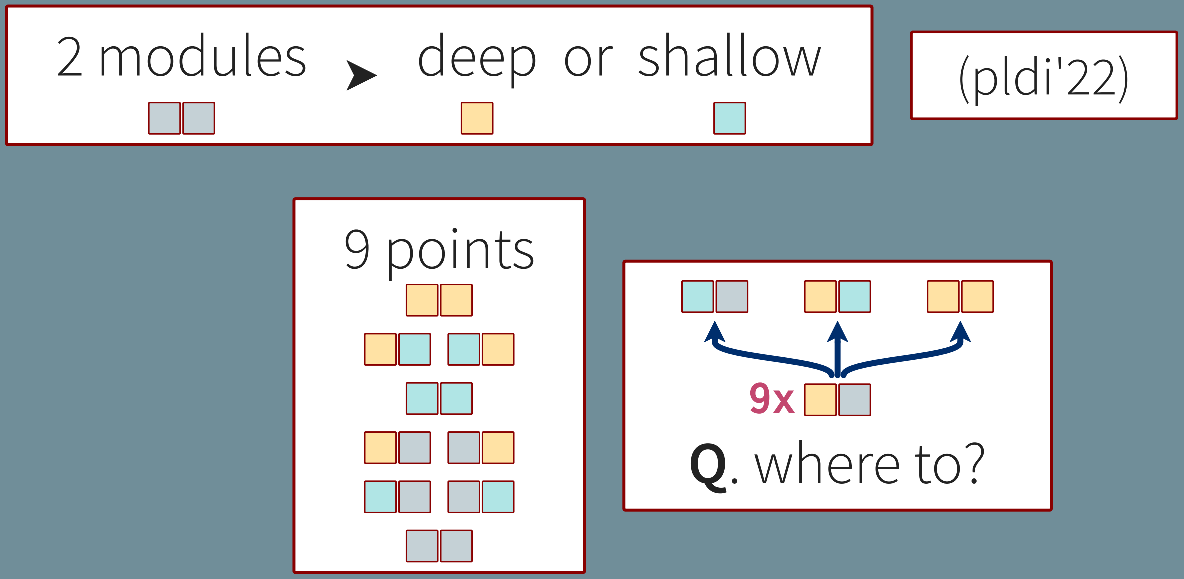
🟨🟦 🟦🟨

🟦🟦

🟨⬜ ⬜🟨

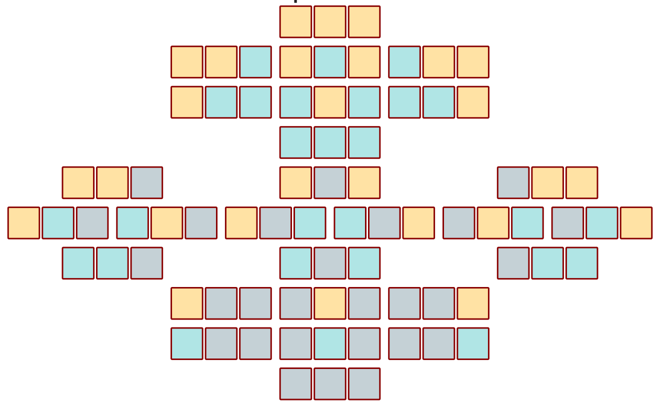
🟦⬜ ⬜🟦

⬜⬜

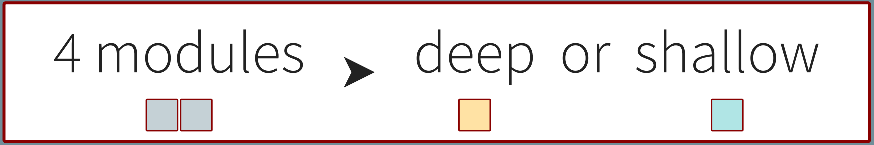

2 modules
➤
deep or shallow
(pldi'22)
9 points
9x
Q. where to?

3 modules ➤ deep or shallow

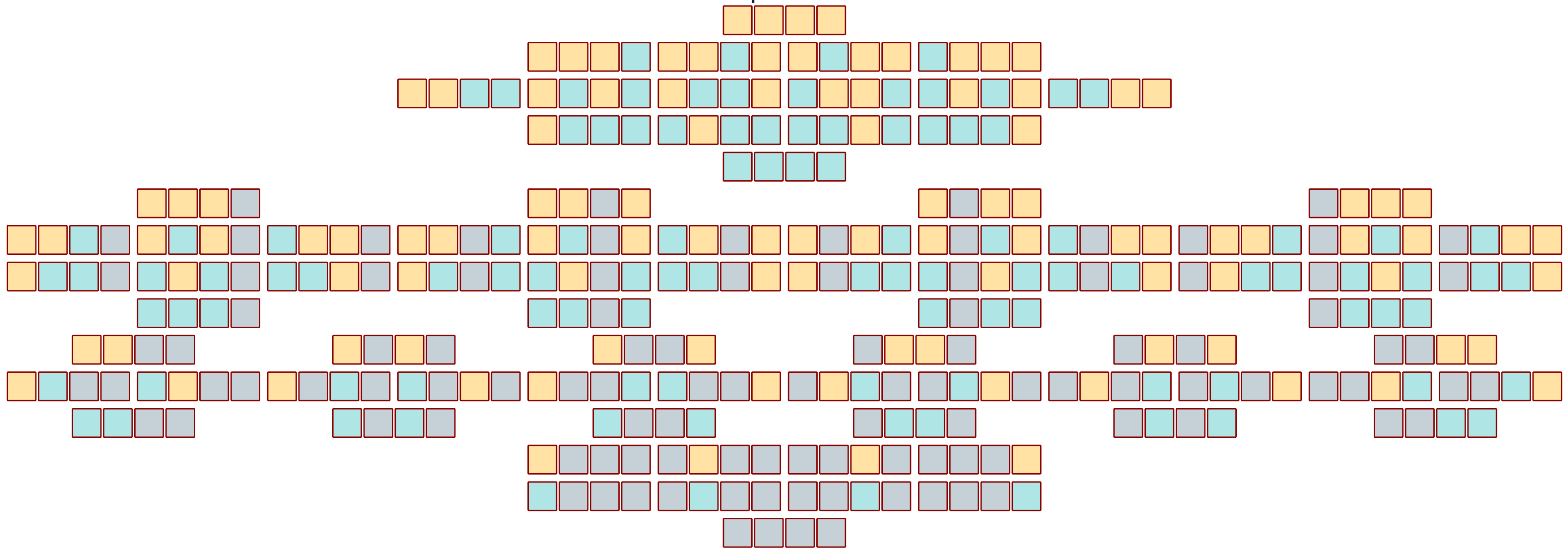
27 points



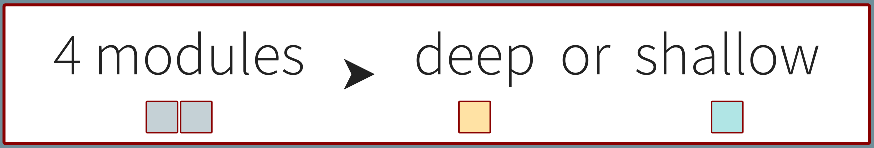
4 modules ➤ deep or shallow

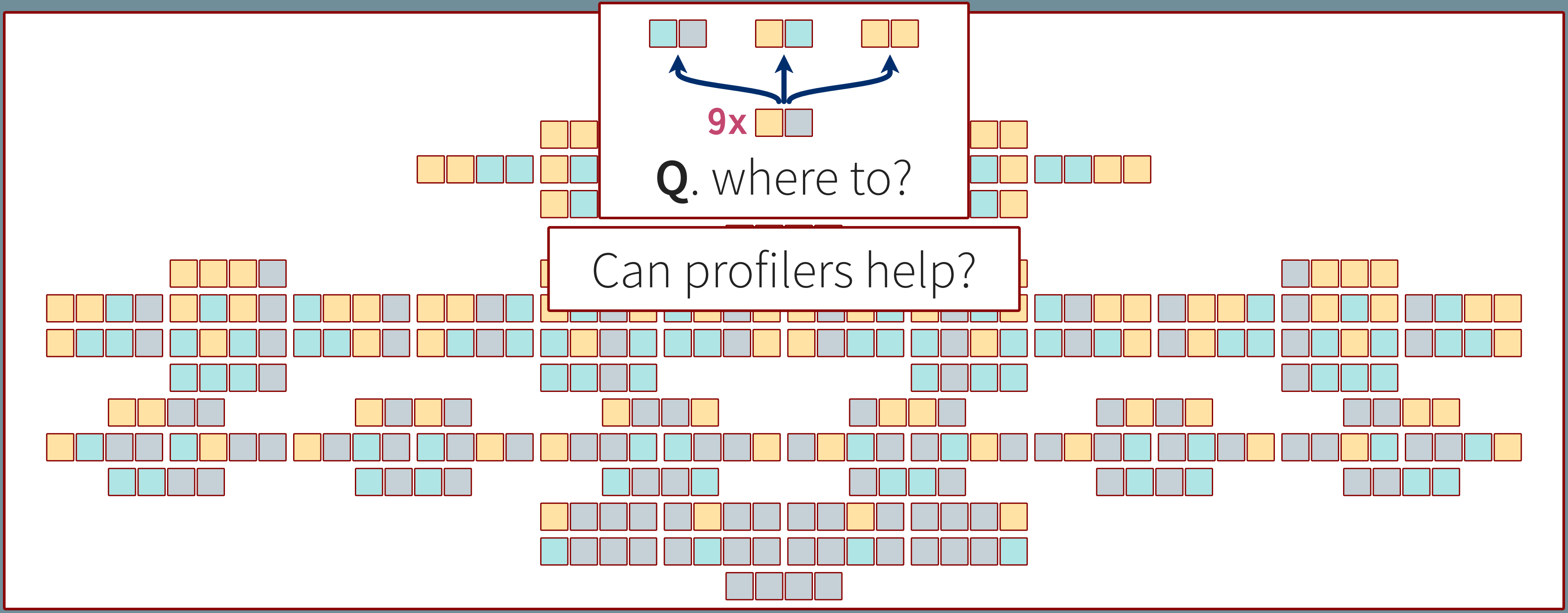


81 points



4 modules ➤ deep or shallow





Profilers

Profilers

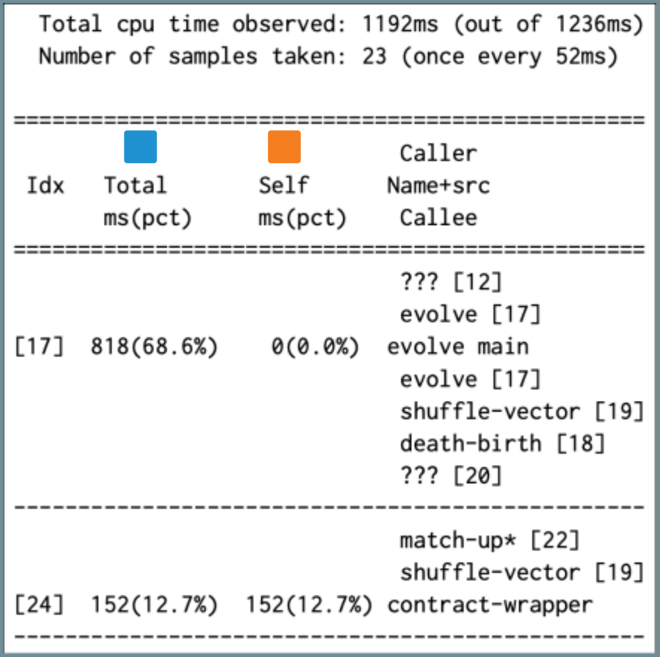
Statistical Profiler

Contract Profiler

Profilers

Statistical Profiler

Contract 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
--------------------------------------------------
```

Profilers

Statistical Profiler

Total %

Self %

Contract Profiler

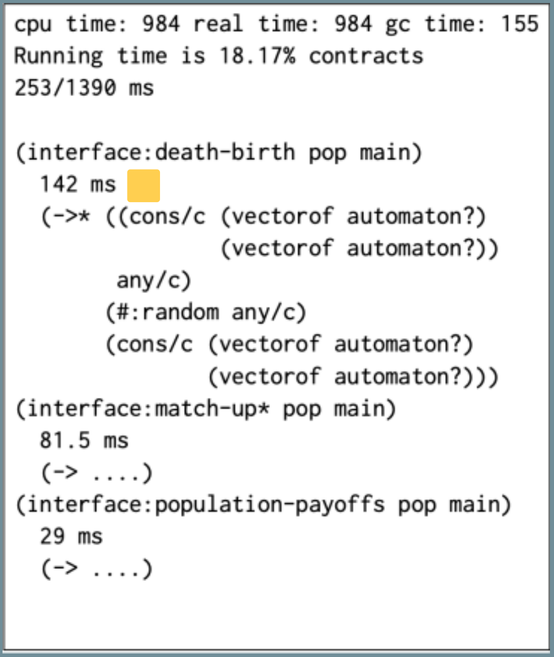
Profilers

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 %

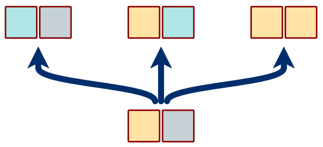
Self %

Contract Profiler

Contract %

Deep types Contract @ boundary	**Shallow** types Asserts in typed code
✓ Contract %	🟥 Contract %
Total %	✓ Total %
Self %	✓ Self %

The Problem



Q. where to?

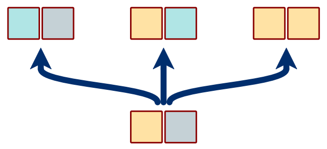
Q. how to find a boundary?

Contract %

Total %

Self %

The Problem



Q. where to?

A. Rational Programmer experiment

Q. how to find a boundary?

Contract %

Total %

Self %

Rational Programmer

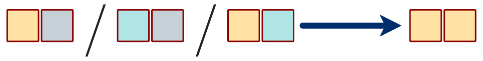
Rational Programmer

Identify strategies, let them compete.

Rational Programmer

Identify strategies, let them compete.

Deep (🟨🟦🟧)



Rational Programmer

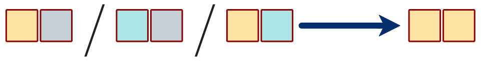
Identify strategies, let them compete.

Deep (🟨🟦🟧) **Shallow**

Rational Programmer

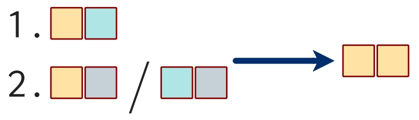
Identify strategies, let them compete.

Deep (🟨🟦🟧)



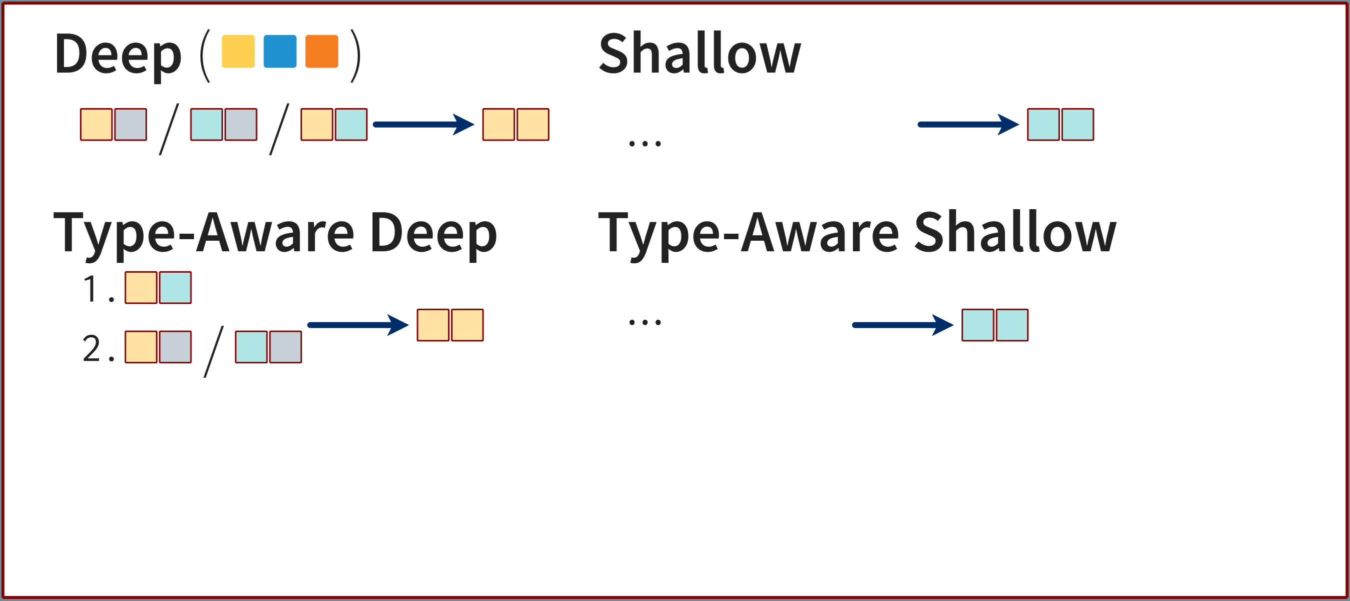
Shallow

Type-Aware Deep



Rational Programmer

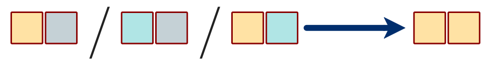
Identify strategies, let them compete.



Rational Programmer

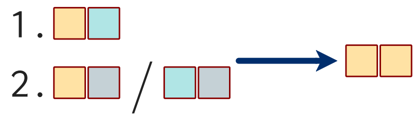
Identify strategies, let them compete.

Deep (🟨🟦🟧)



Shallow

Type-Aware Deep



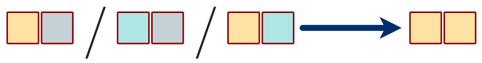
Type-Aware Shallow

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

Rational Programmer

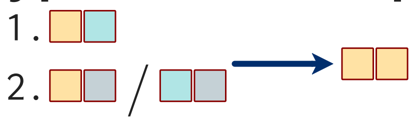
Identify strategies, let them compete.

Deep (🟨🟦🟧)



Shallow

Type-Aware Deep



Type-Aware Shallow

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

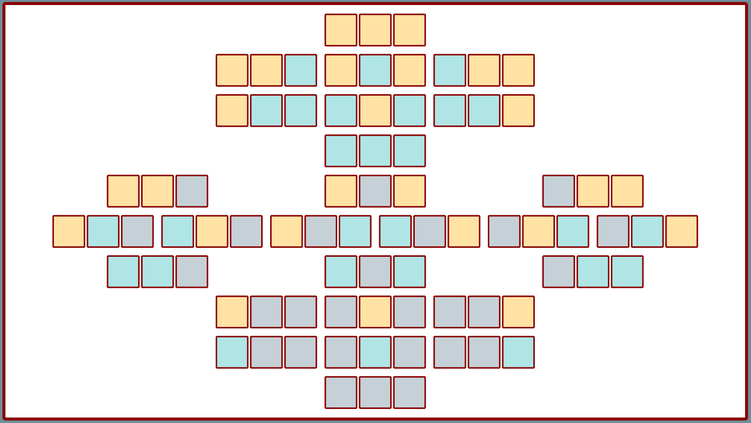
null, **pldi22** baselines

Rational Programmer

Identify strategies, let them compete.

Rational Programmer

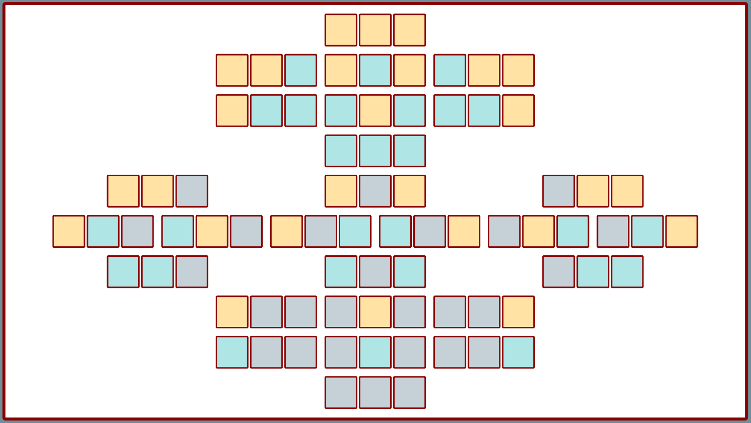
Identify strategies, let them compete.



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 = never slow down
k **loose** = k slower steps

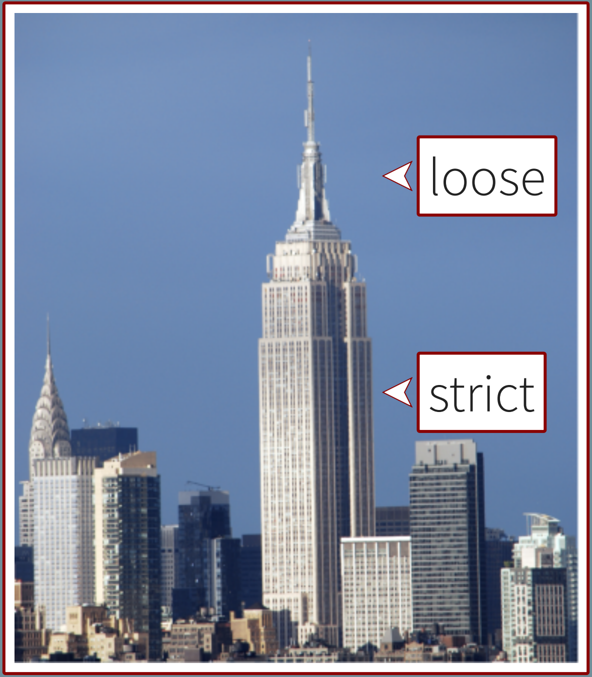
99x ➤ 99x ➤ 3x ➤ **1x**
3x ➤ **99x** ➤ **1x**

Dataset

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

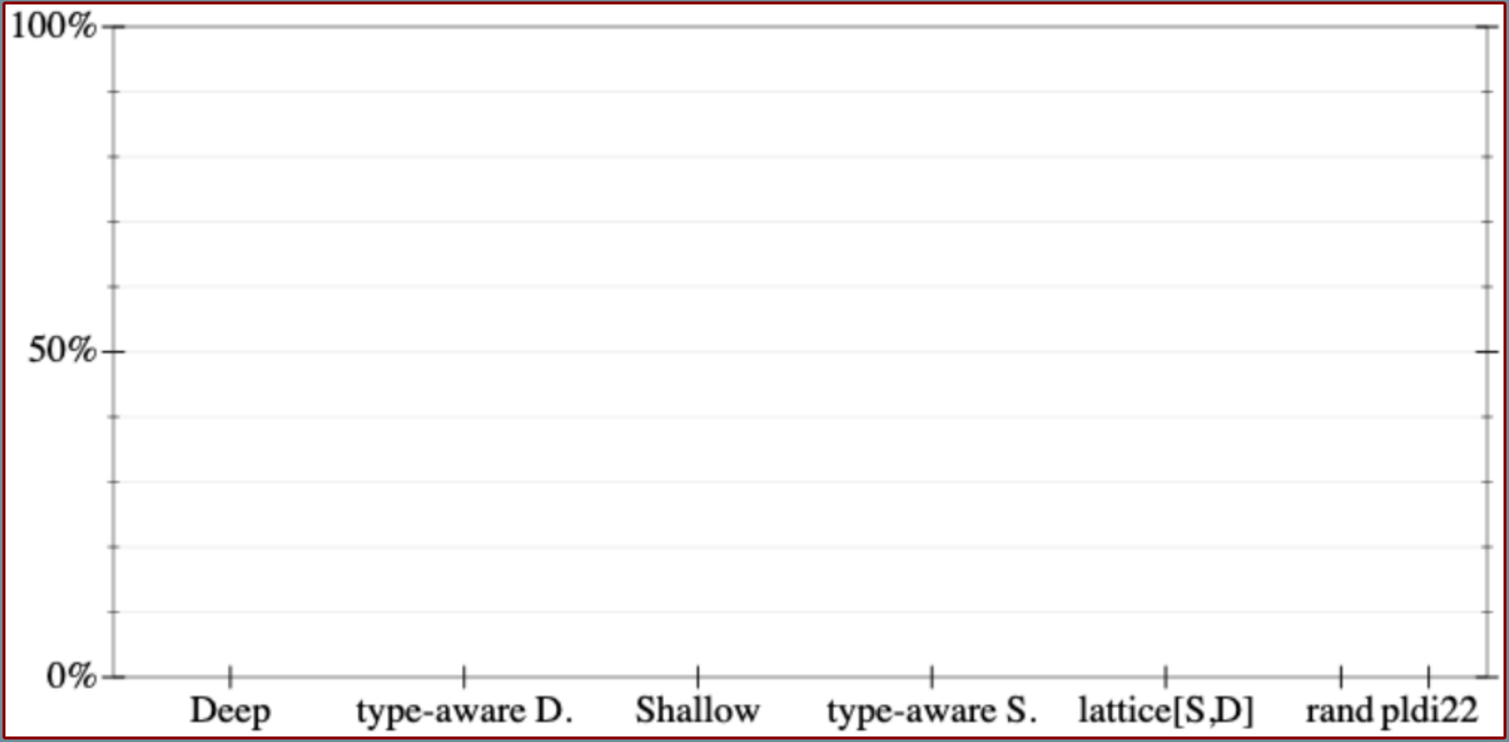
How often do the strategies succeed?

How often do the strategies succeed?



How often do the strategies succeed?

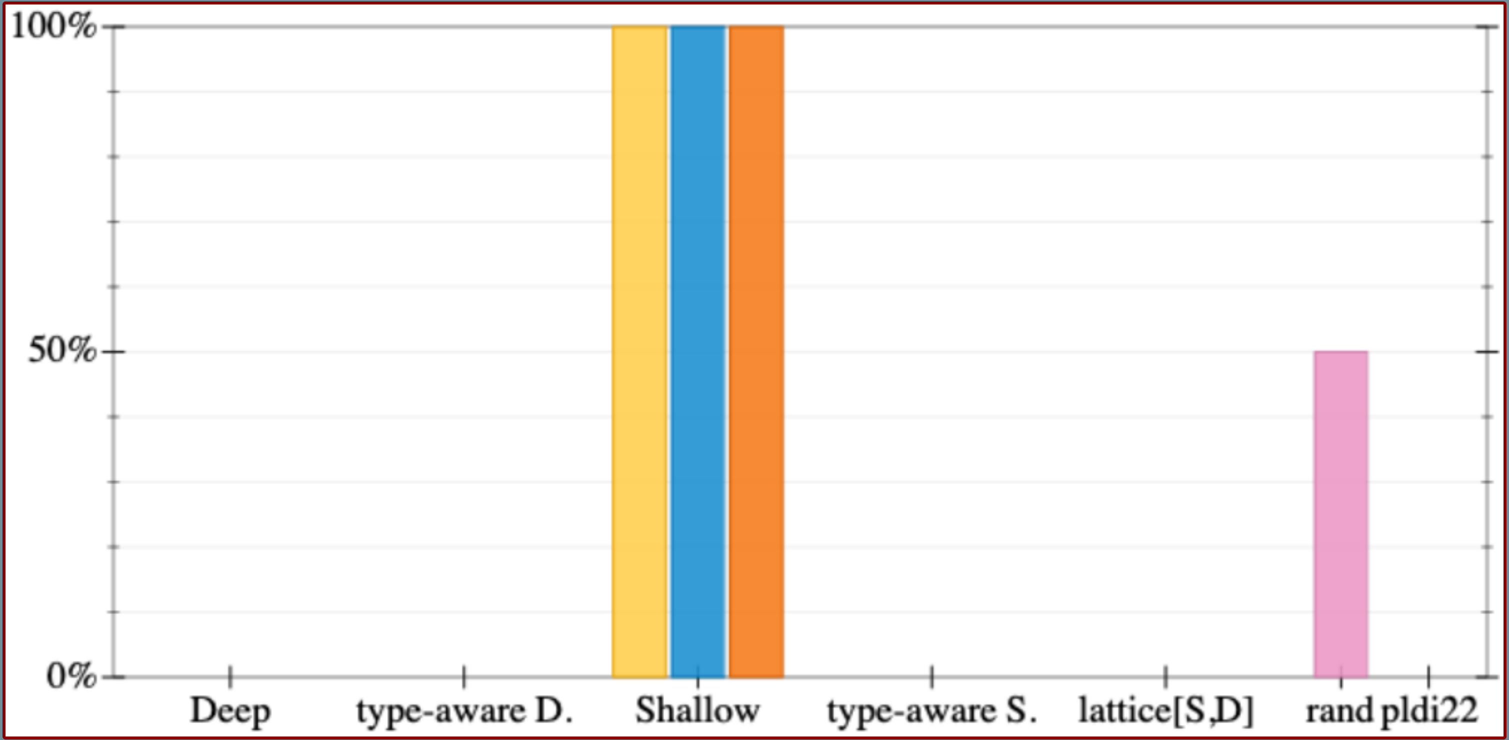
X = strategies, Y = % scenarios



🟨 contract profile 🟦 stat. total 🟧 stat. self 🟪 no profile

How often do the strategies succeed?

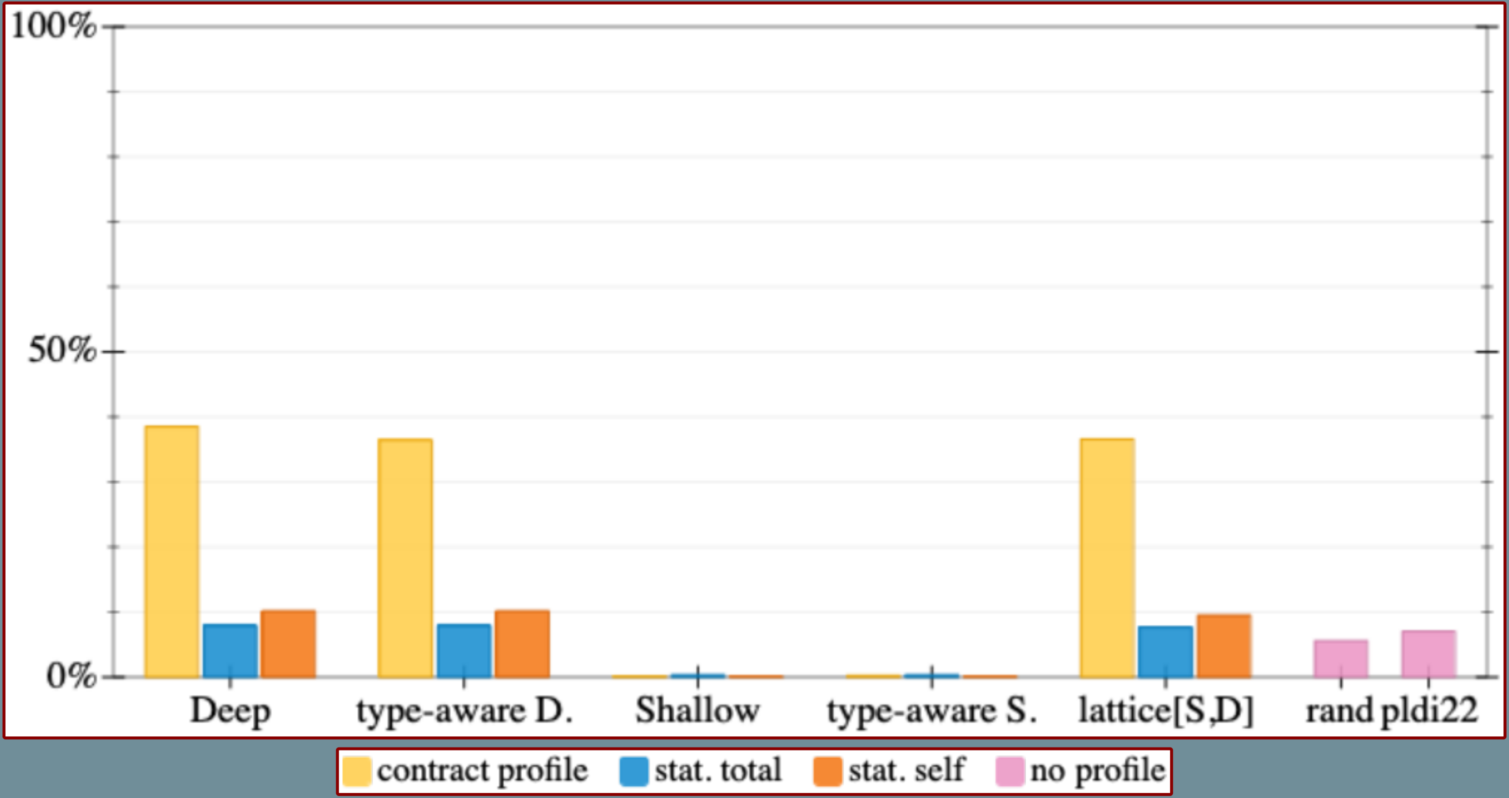
example data



🟨 contract profile 🟦 stat. total 🟧 stat. self 🟪 no profile

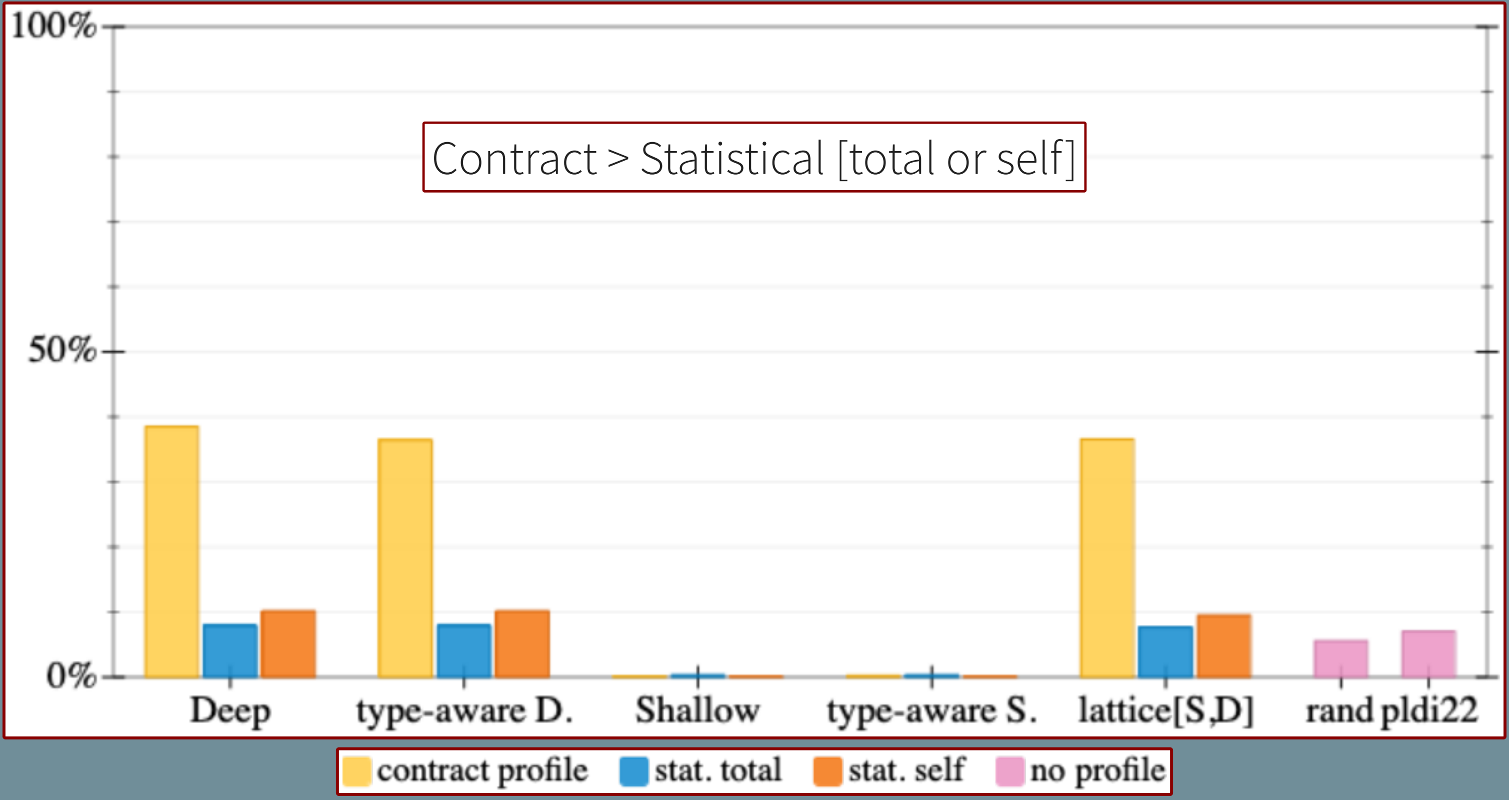
How often do the strategies succeed?

strict success



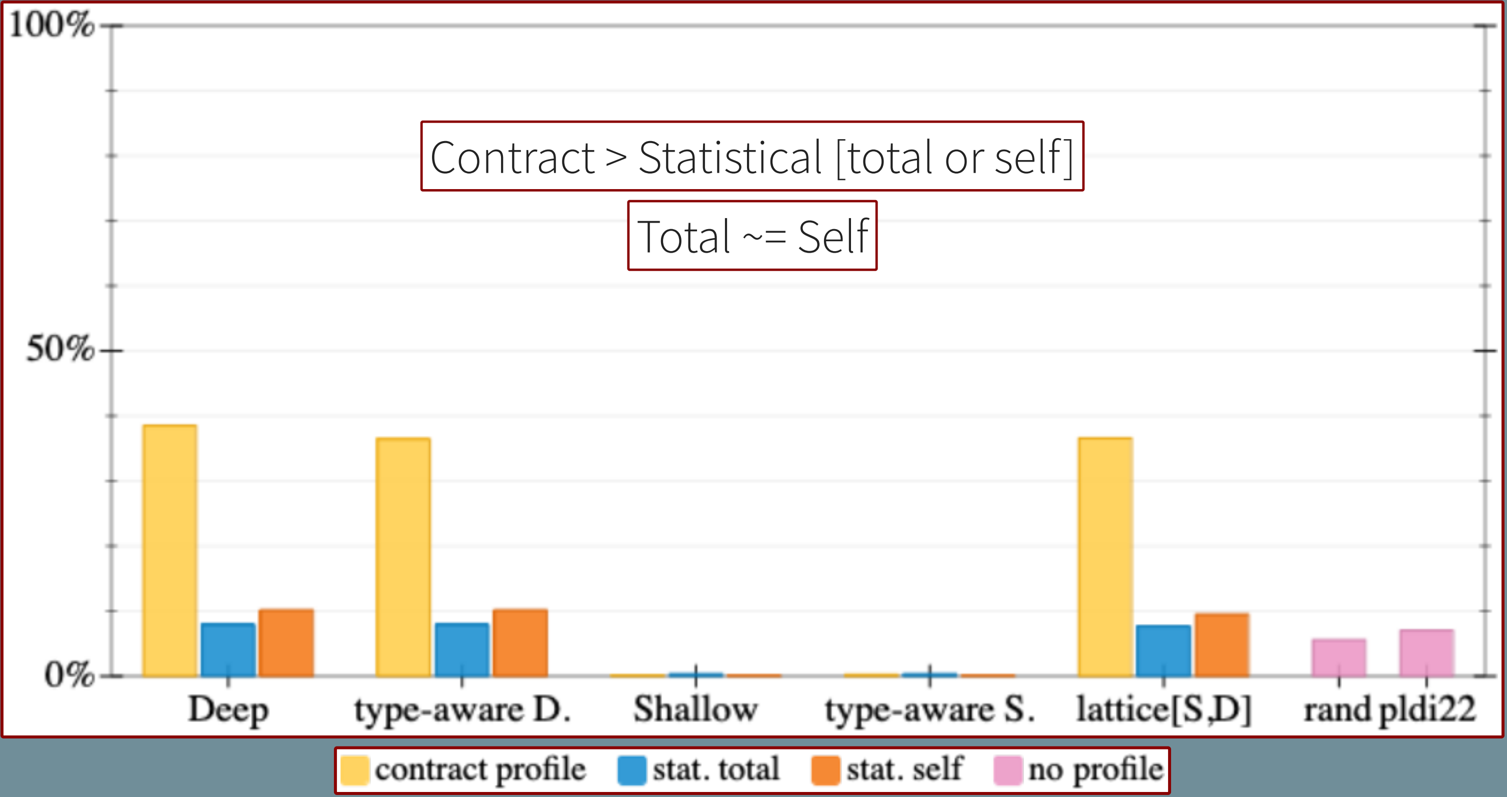
How often do the strategies succeed?

strict success



How often do the strategies succeed?

strict success

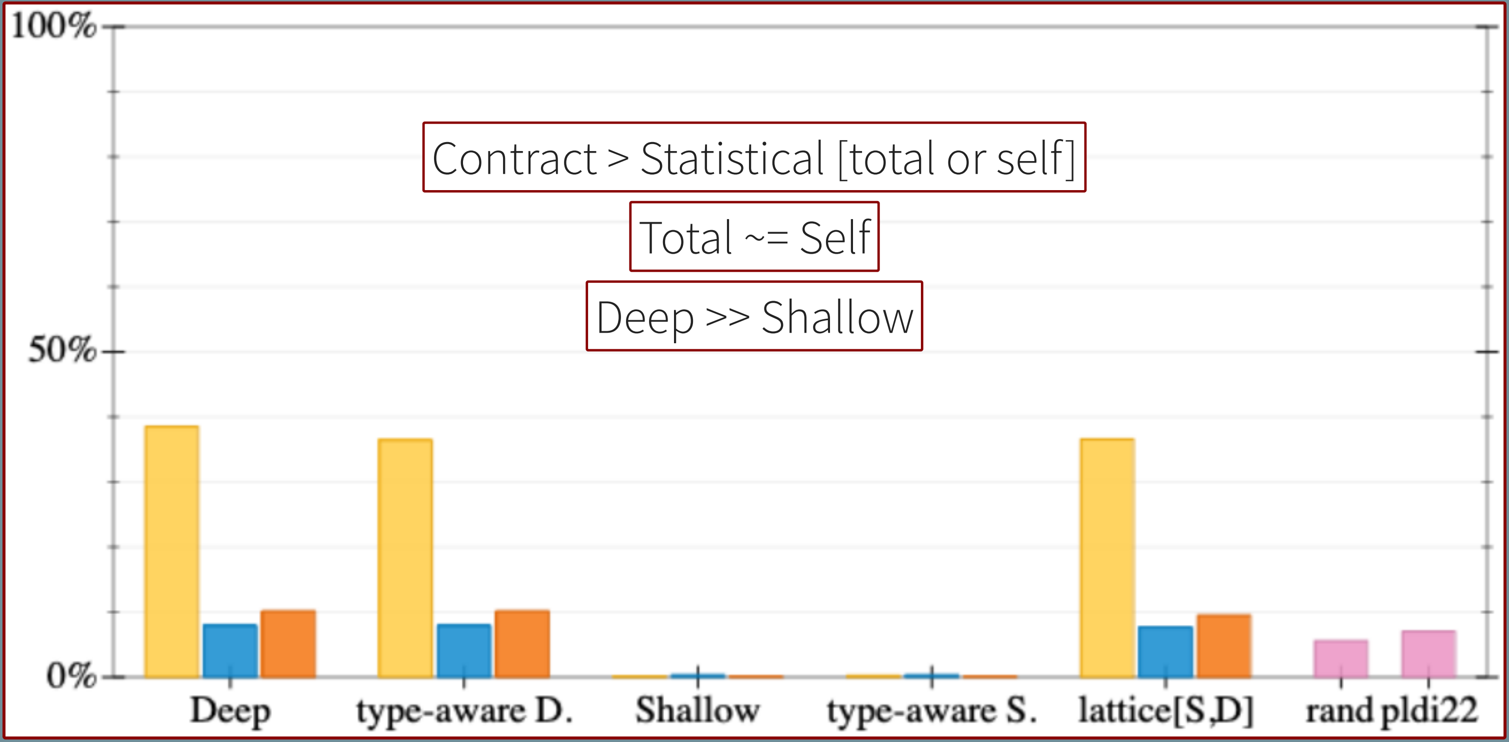


Contract > Statistical [total or self]

Total ~= Self

How often do the strategies succeed?

strict success



Contract > Statistical [total or self]

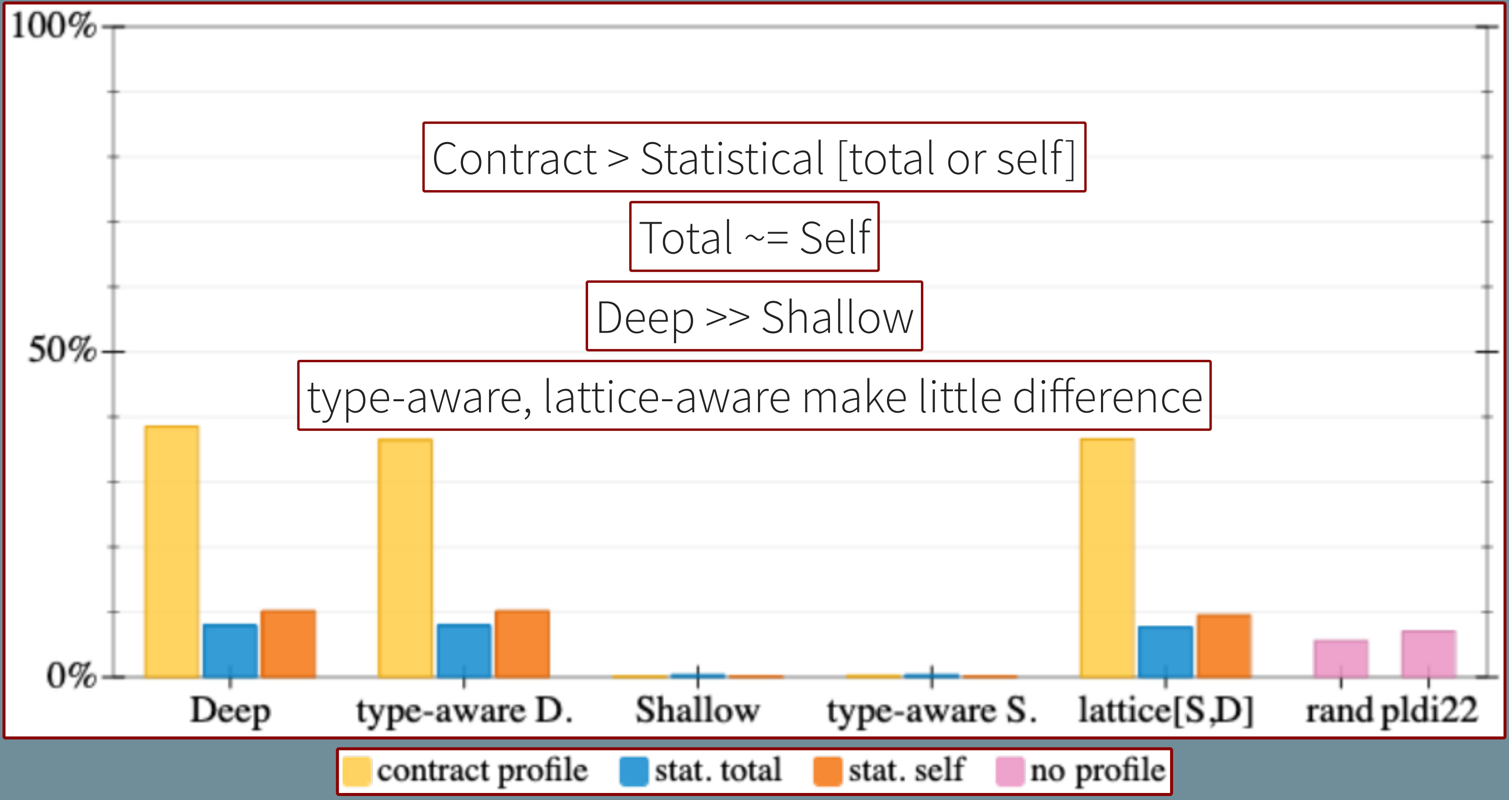
Total ~= Self

Deep >> Shallow

🟨 contract profile 🟦 stat. total 🟧 stat. self 🟪 no profile

How often do the strategies succeed?

strict success



Contract > Statistical [total or self]

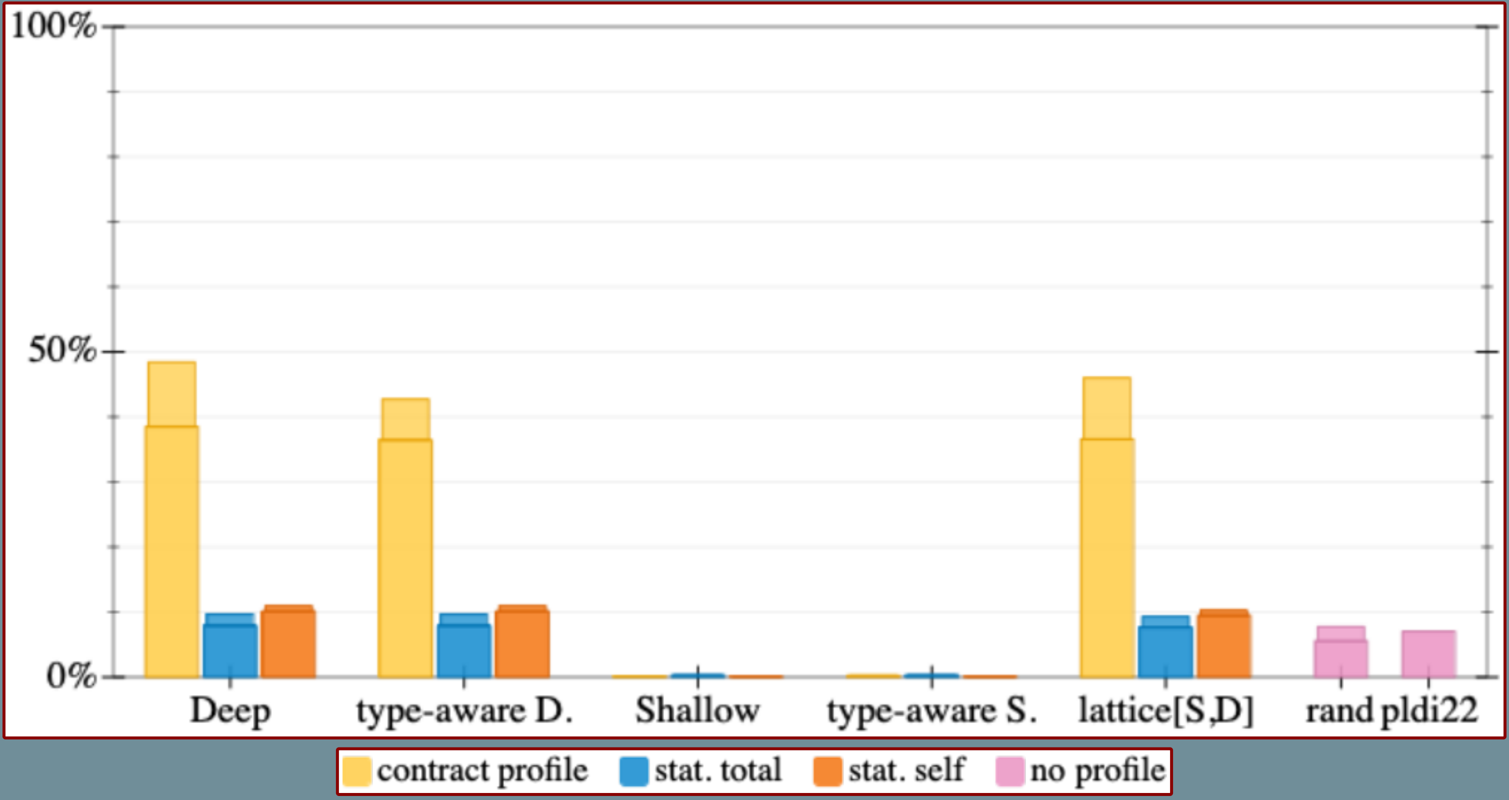
Total ~= Self

Deep >> Shallow

type-aware, lattice-aware make little difference

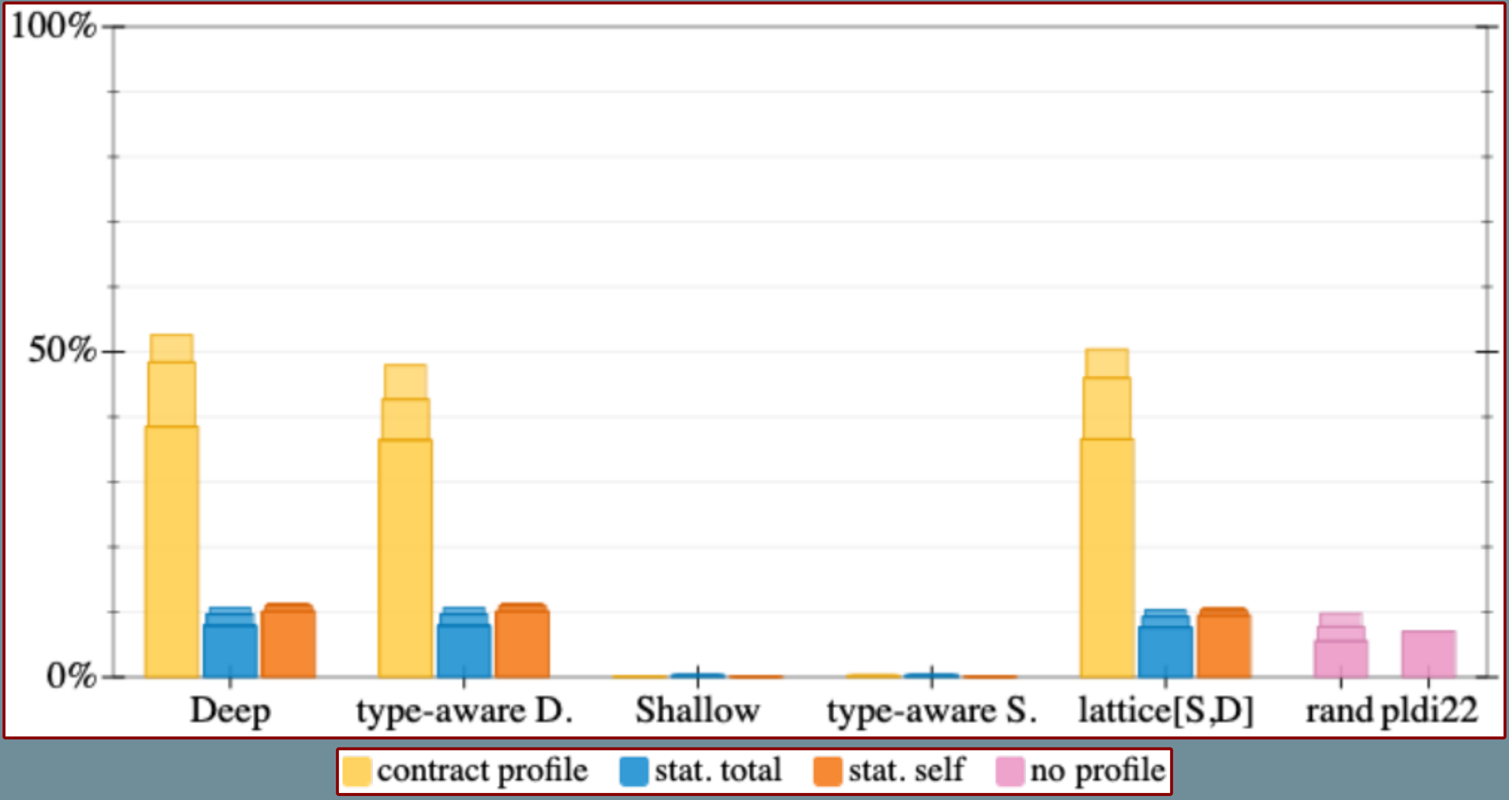
How often do the strategies succeed?

strict success 1 loose



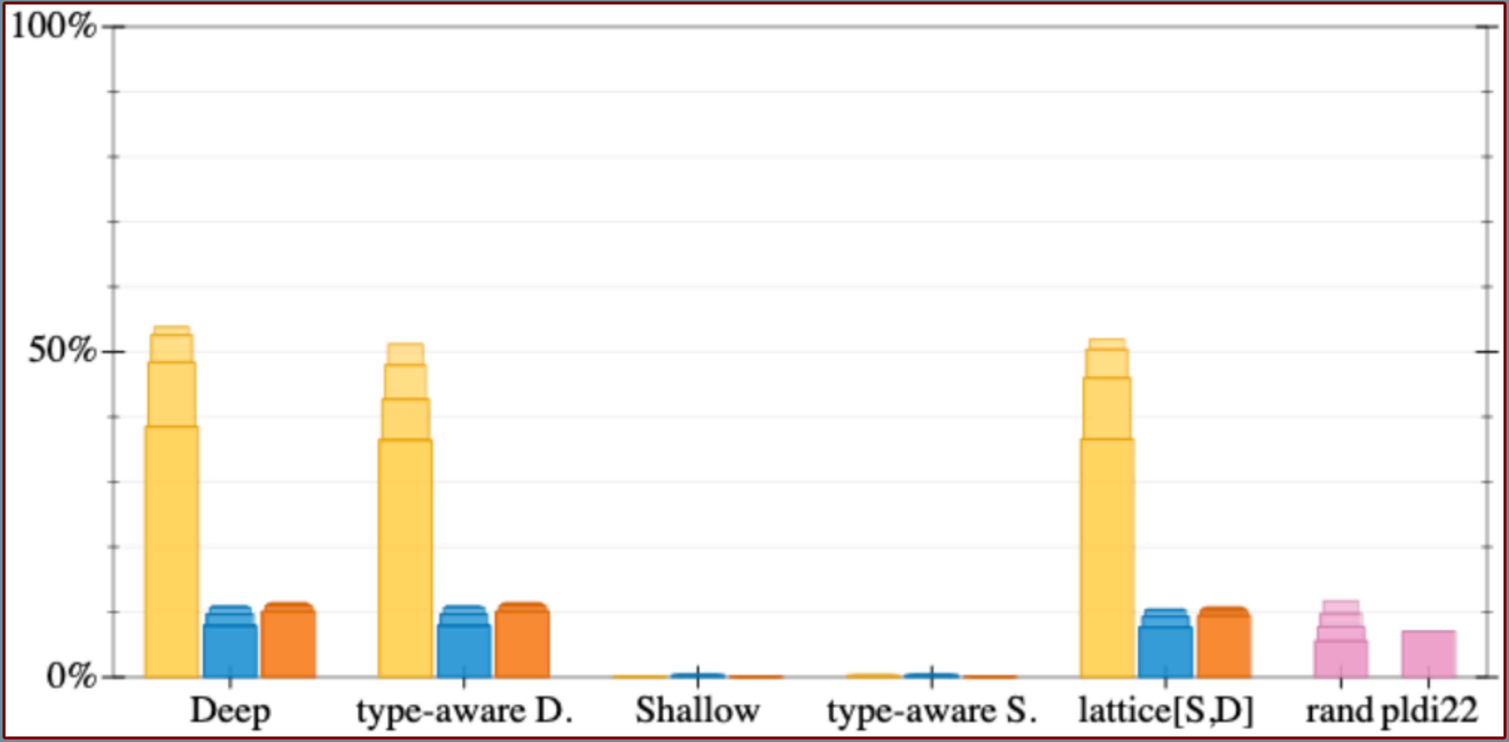
How often do the strategies succeed?

strict success | 1 loose | 2 loose



How often do the strategies succeed?

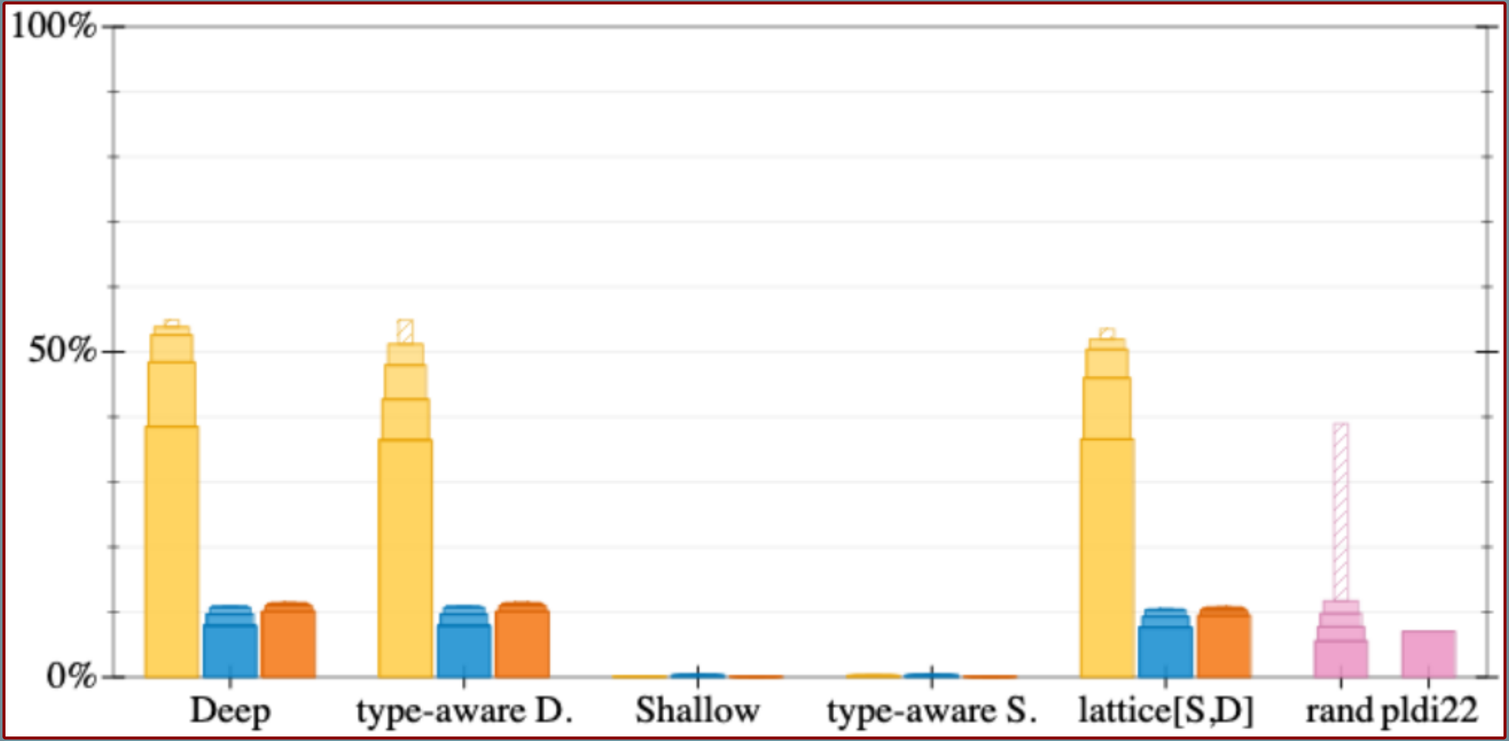
strict success | 1 loose | 2 loose | 3 loose



🟨 contract profile 🟦 stat. total 🟧 stat. self 🟪 no profile

How often do the strategies succeed?

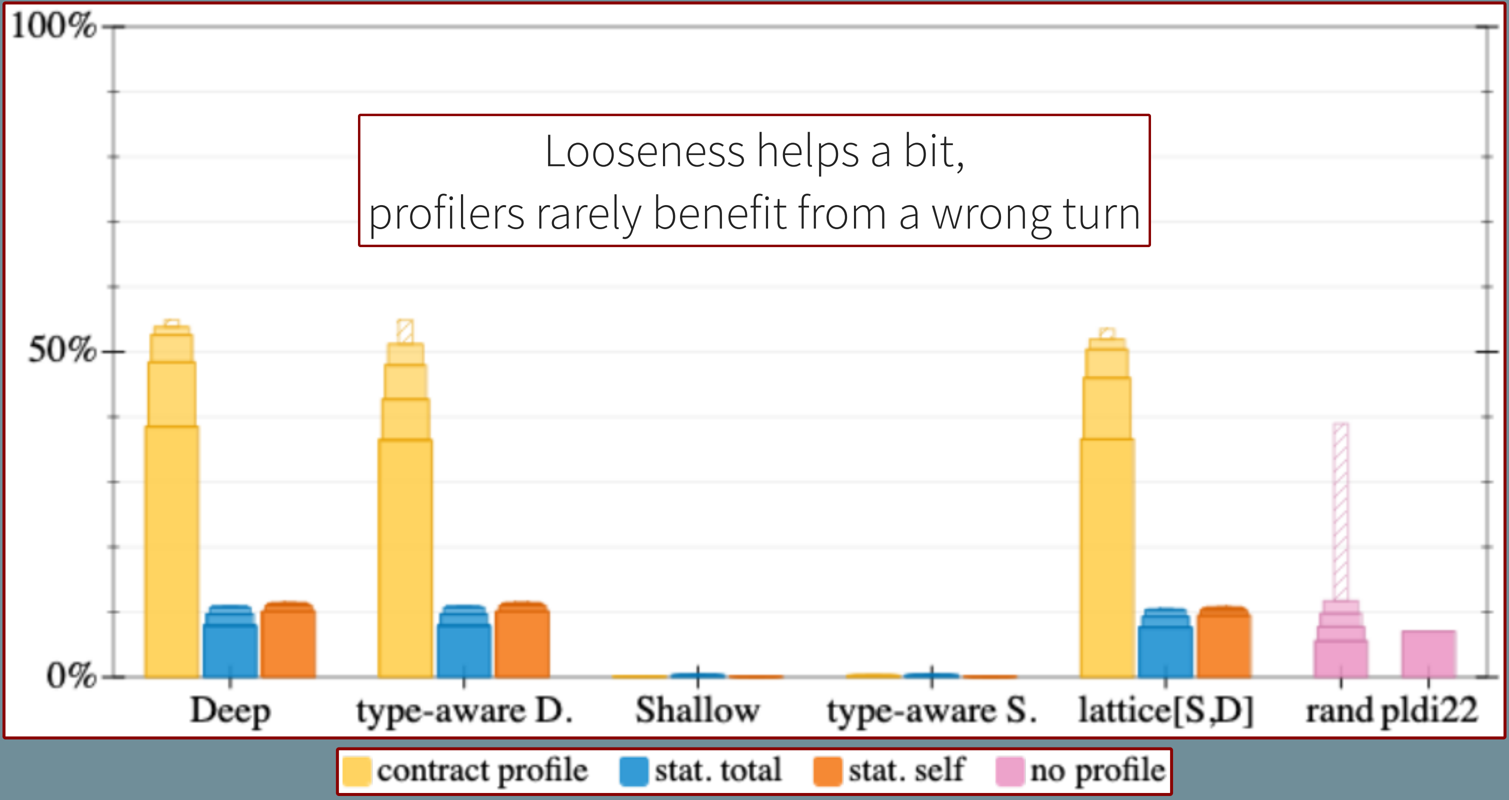
strict success | 1 loose | 2 loose | 3 loose | N loose



🟨 contract profile 🟦 stat. total 🟧 stat. self 🟪 no profile

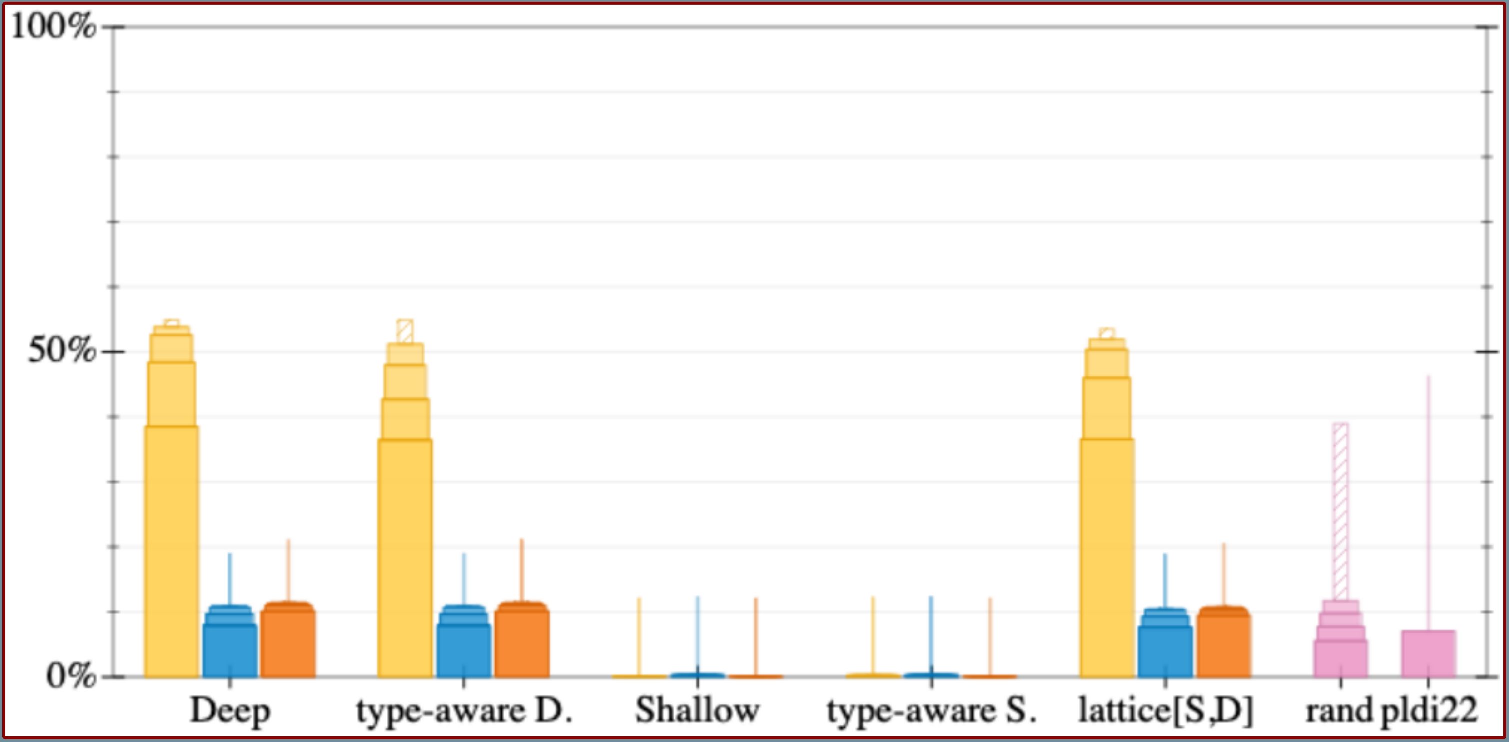
How often do the strategies succeed?

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



How often do the strategies succeed?

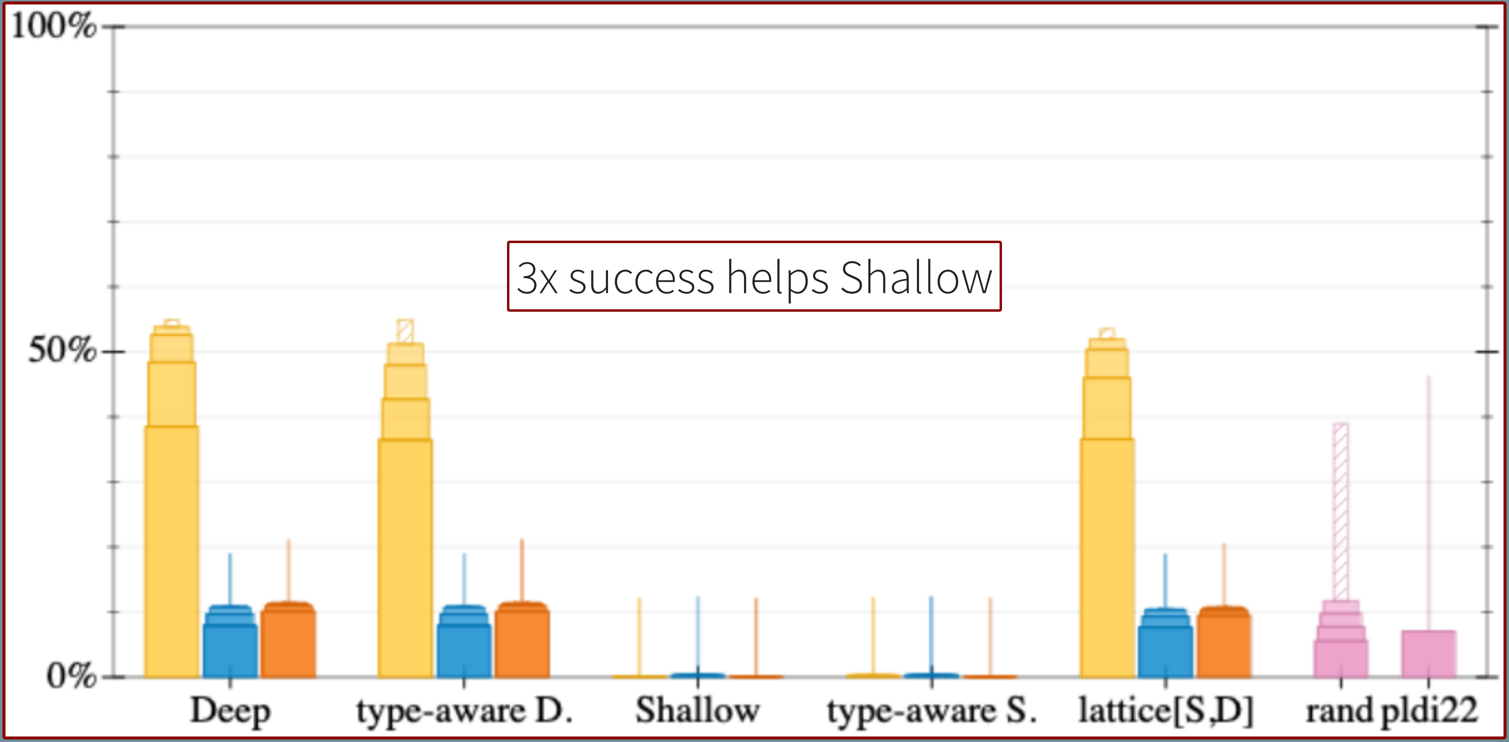
strict success | 1 loose | 2 loose | 3 loose | N loose | strict 3x



🟨 contract profile 🟦 stat. total 🟧 stat. self 🟪 no profile

How often do the strategies succeed?

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



🟨 contract profile 🟦 stat. total 🟧 stat. self 🟪 no profile

Takeaways

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

Q. hybrid strategies, shallow profilers?

Takeaways

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

errors	testing?
perf	debugging?

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

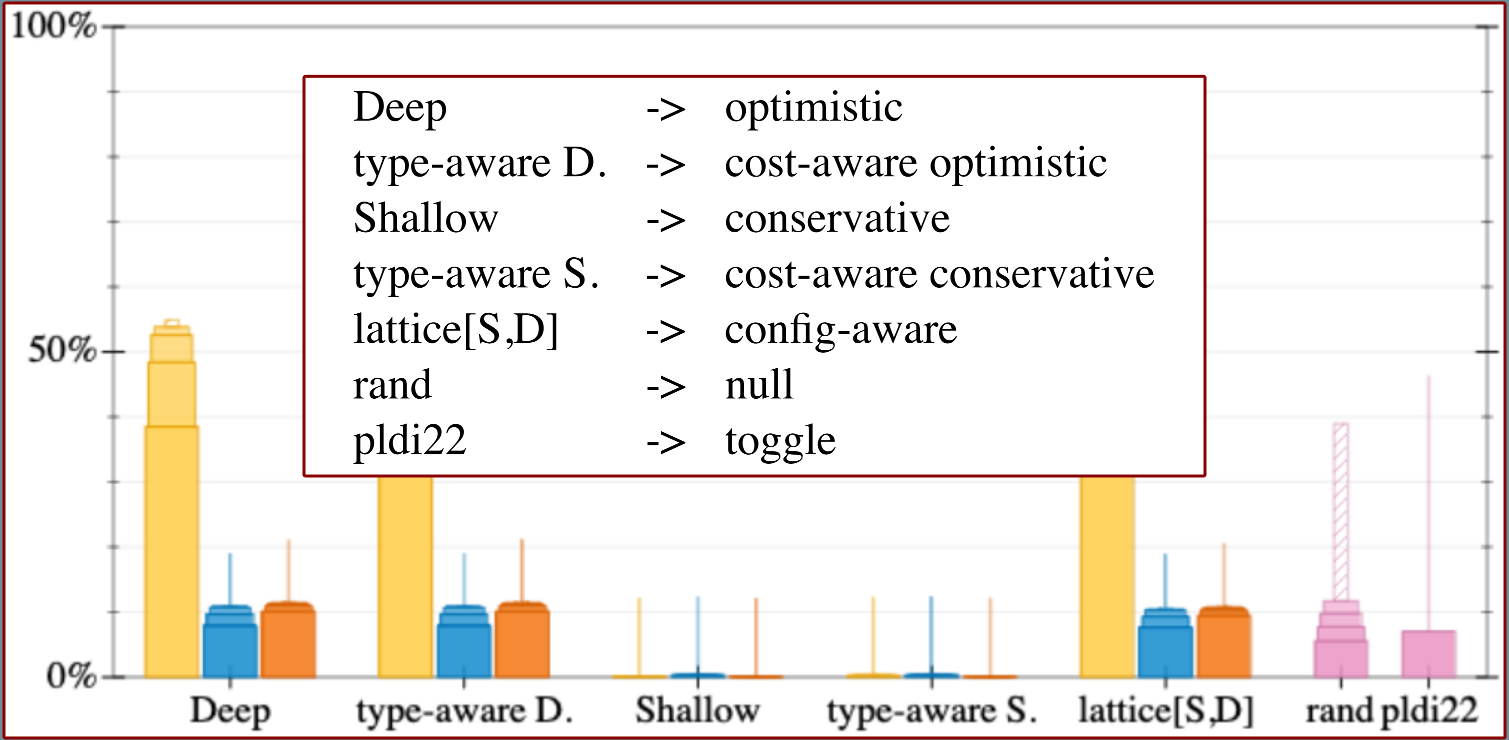
* shallow types do not help

Q. hybrid strategies, shallow profilers?

https://github.com/bennn/gfd-oopsla-2023

Translation: talk to paper

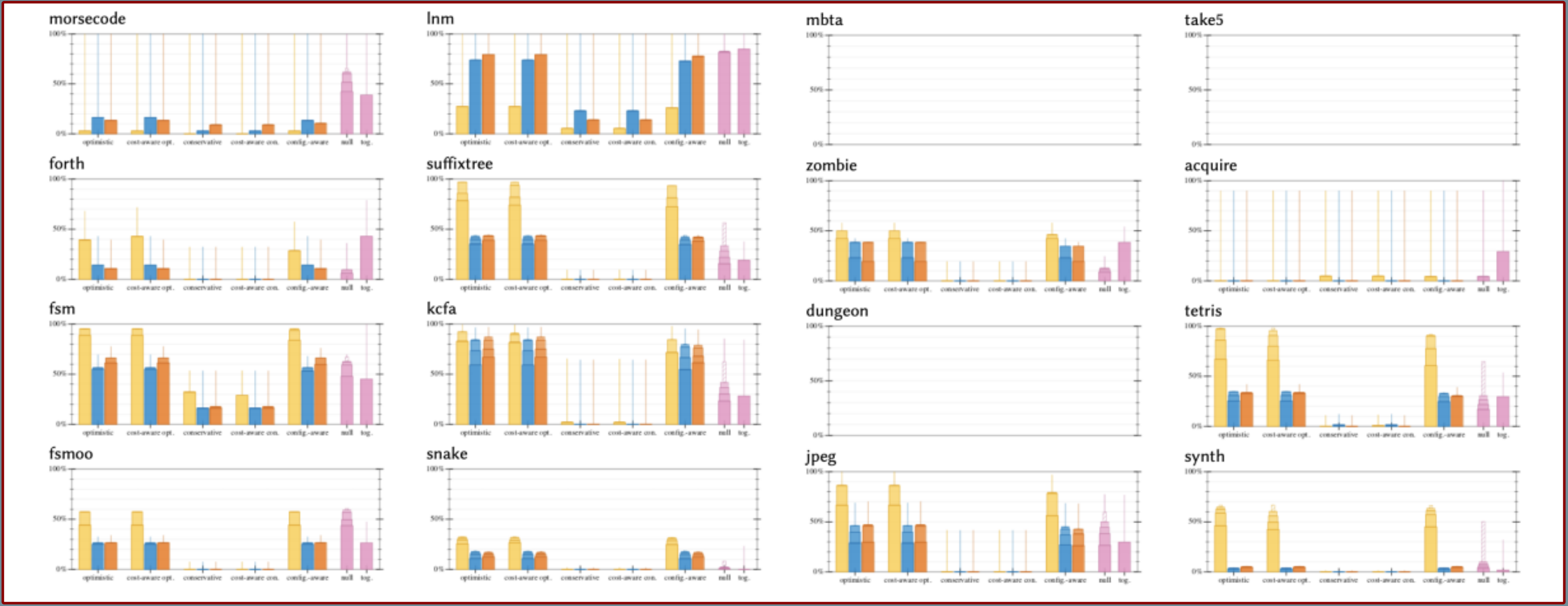
strict success | 1 loose | 2 loose | 3 loose | N loose | strict 3x



Deep	->	optimistic
type-aware D.	->	cost-aware optimistic
Shallow	->	conservative
type-aware S.	->	cost-aware conservative
lattice[S,D]	->	config-aware
rand	->	null
pldi22	->	toggle

🟨 contract profile 🟦 stat. total 🟧 stat. self 🟪 no profile

Skylines per Benchmark



Hopeful Scenarios

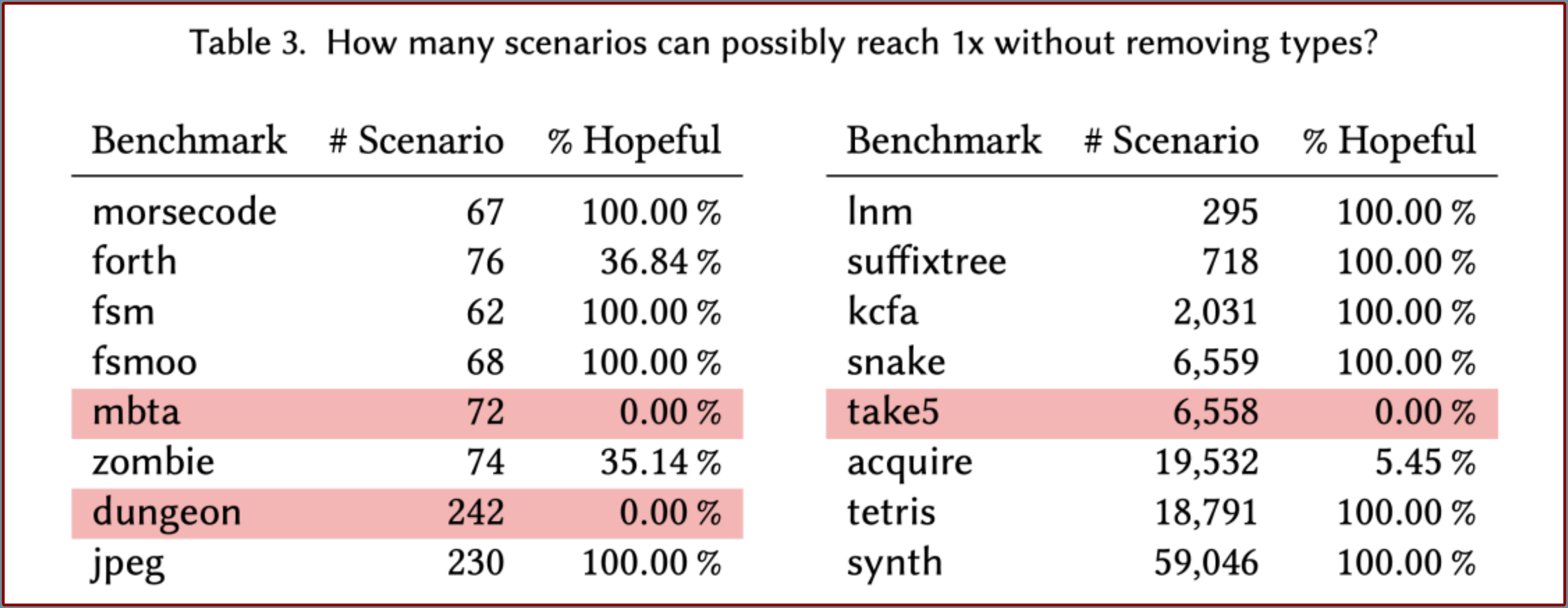
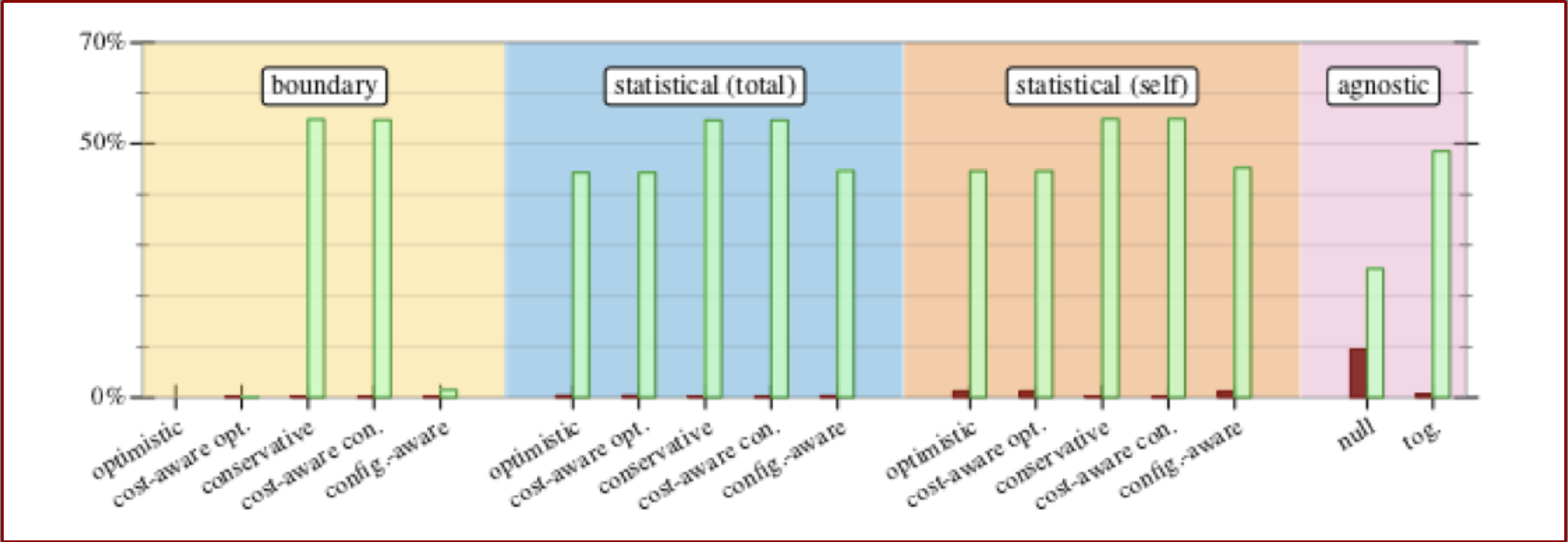


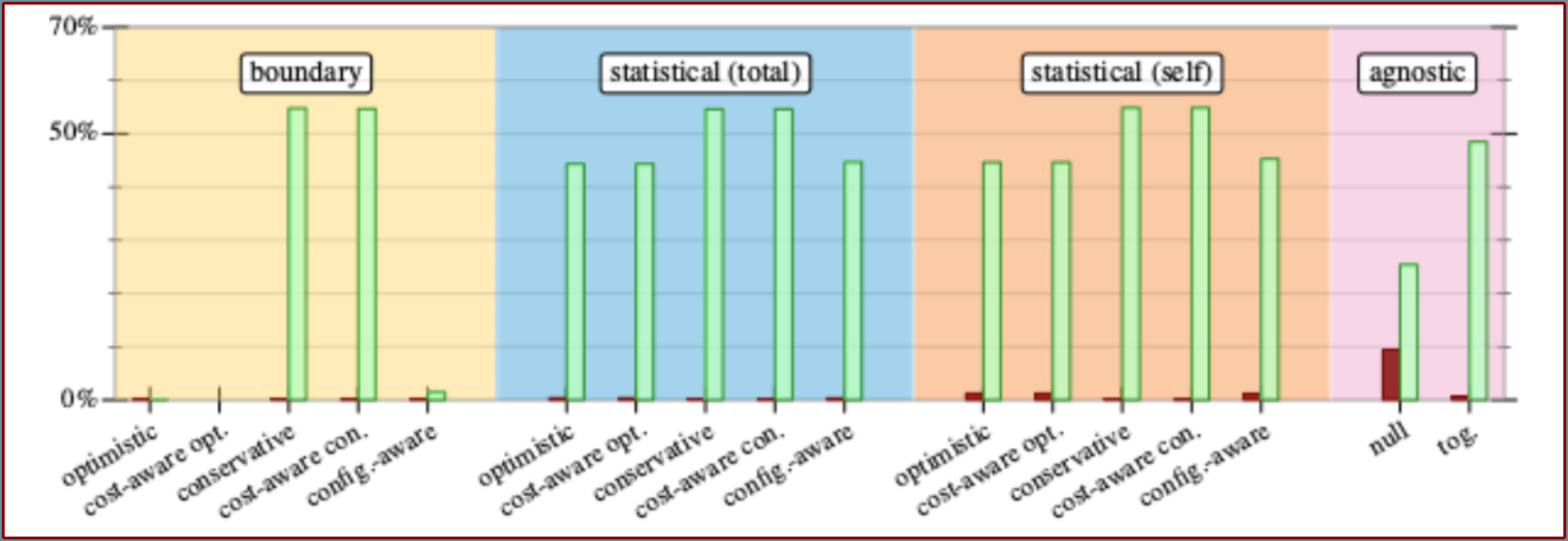
Table 3. How many scenarios can possibly reach 1x without removing types?

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



Type-Aware Boundary vs. the others



Where are the Fast Configs?

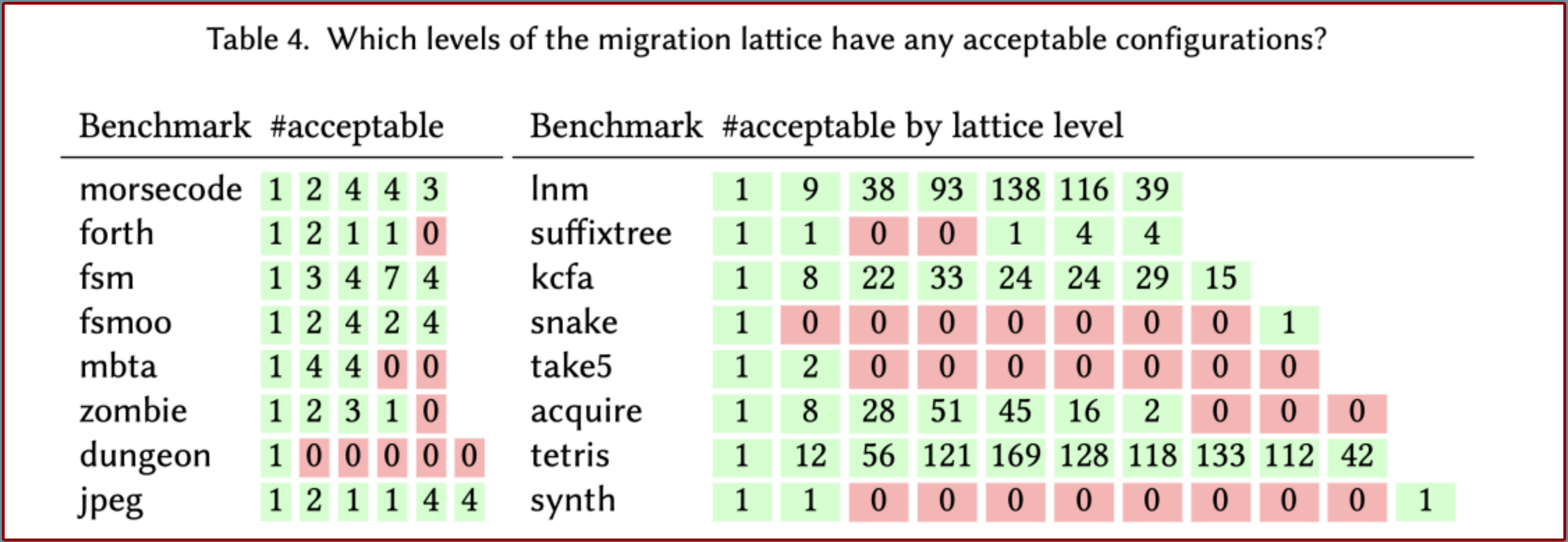
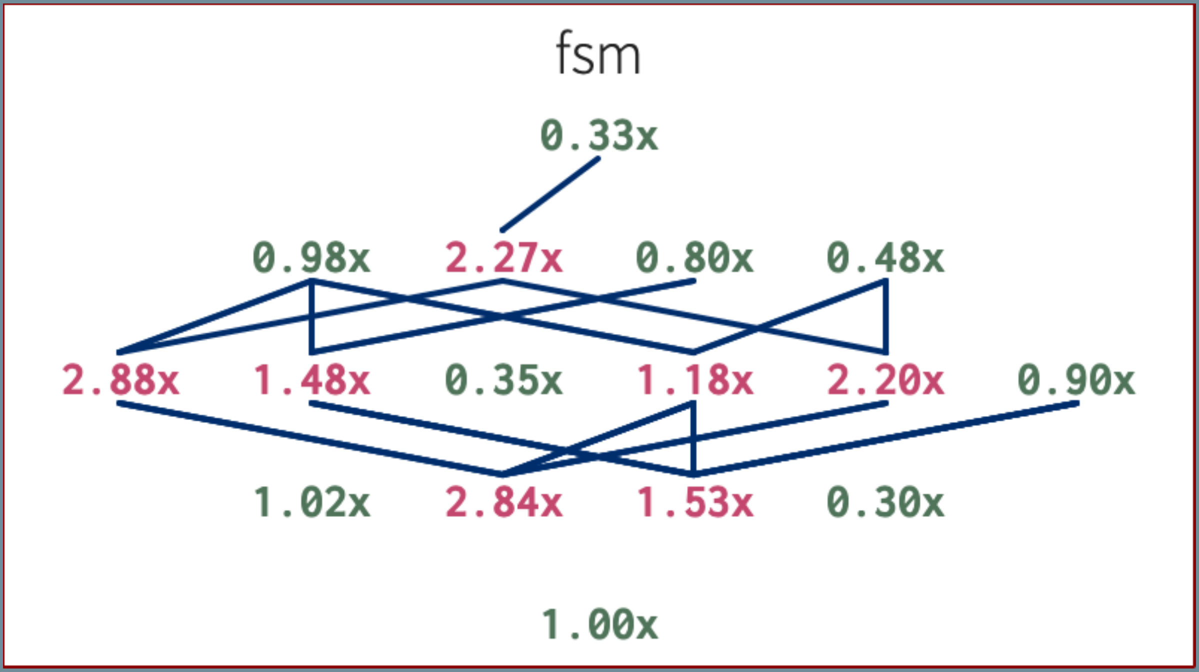


Table 4. Which levels of the migration lattice have any acceptable configurations?

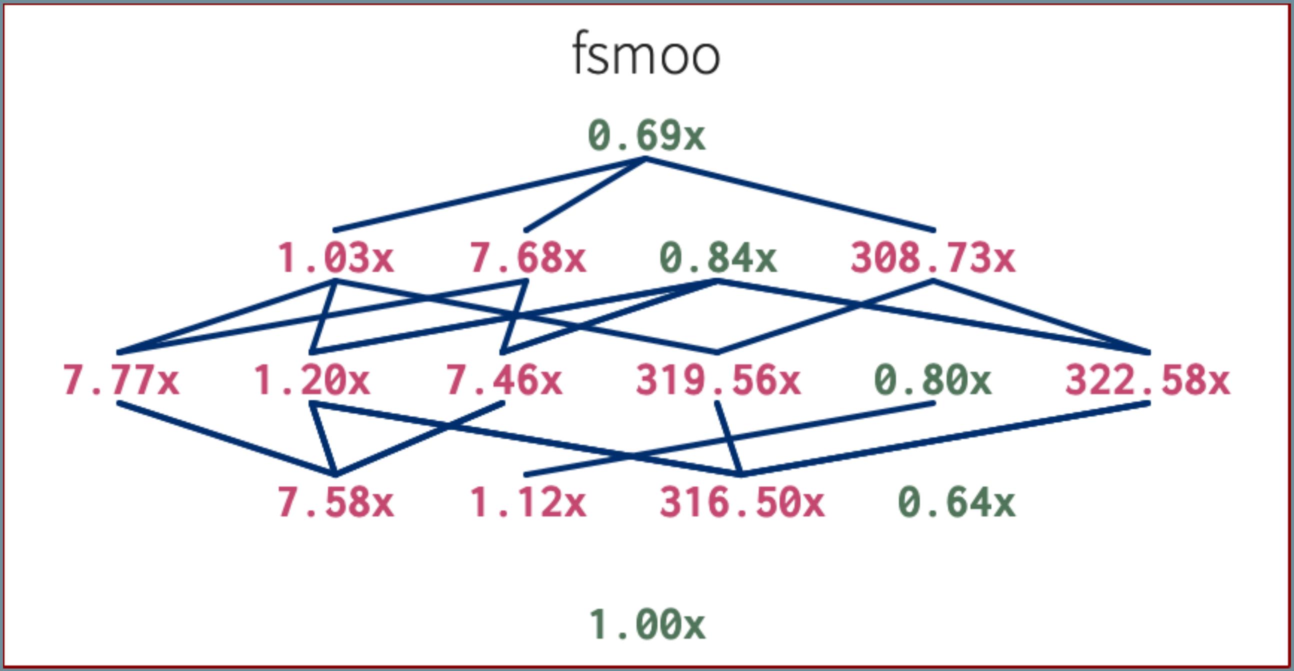
Benchmark	#acceptable					
morsecode	1	2	4	4	3	
forth	1	2	1	1	0	
fsm	1	3	4	7	4	
fsmoo	1	2	4	2	4	
mbta	1	4	4	0	0	
zombie	1	2	3	1	0	
dungeon	1	0	0	0	0	0
jpeg	1	2	1	1	4	4

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

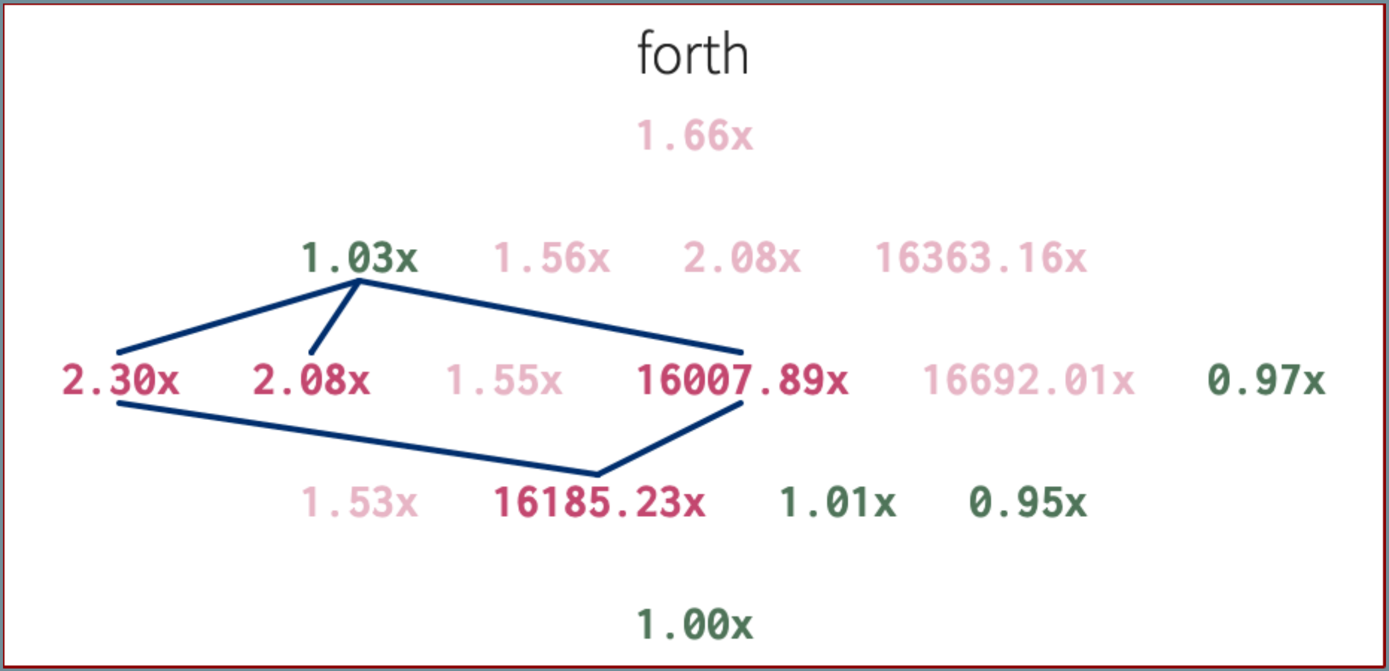
Best-Case Lattice



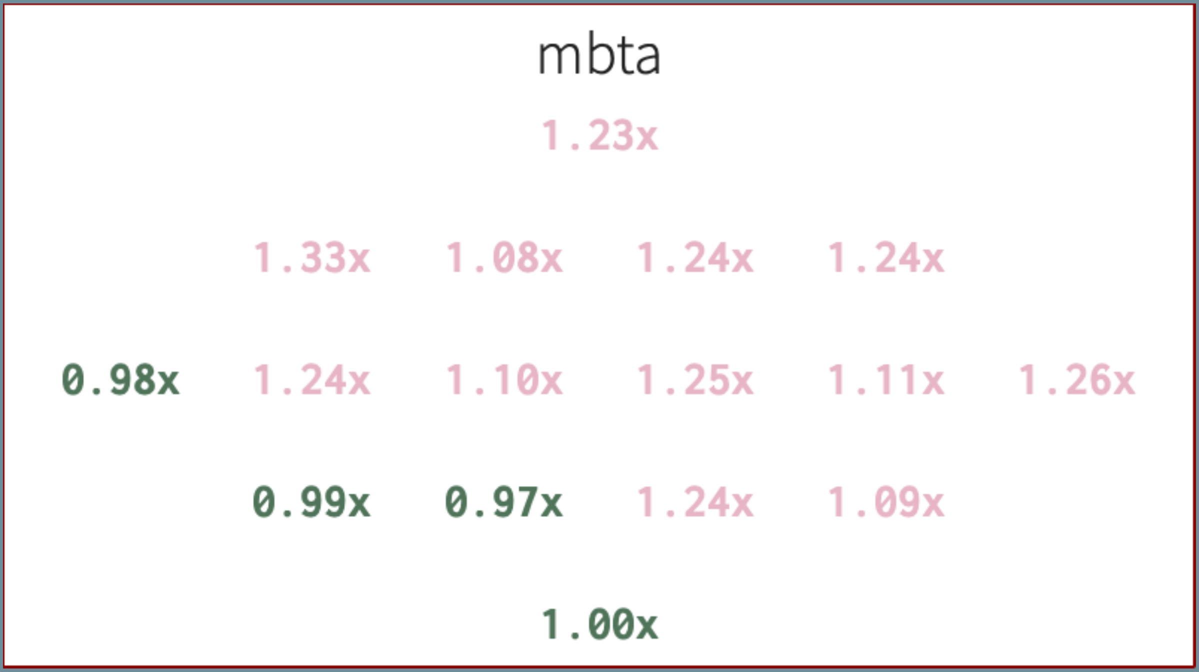
Best-Case Lattice



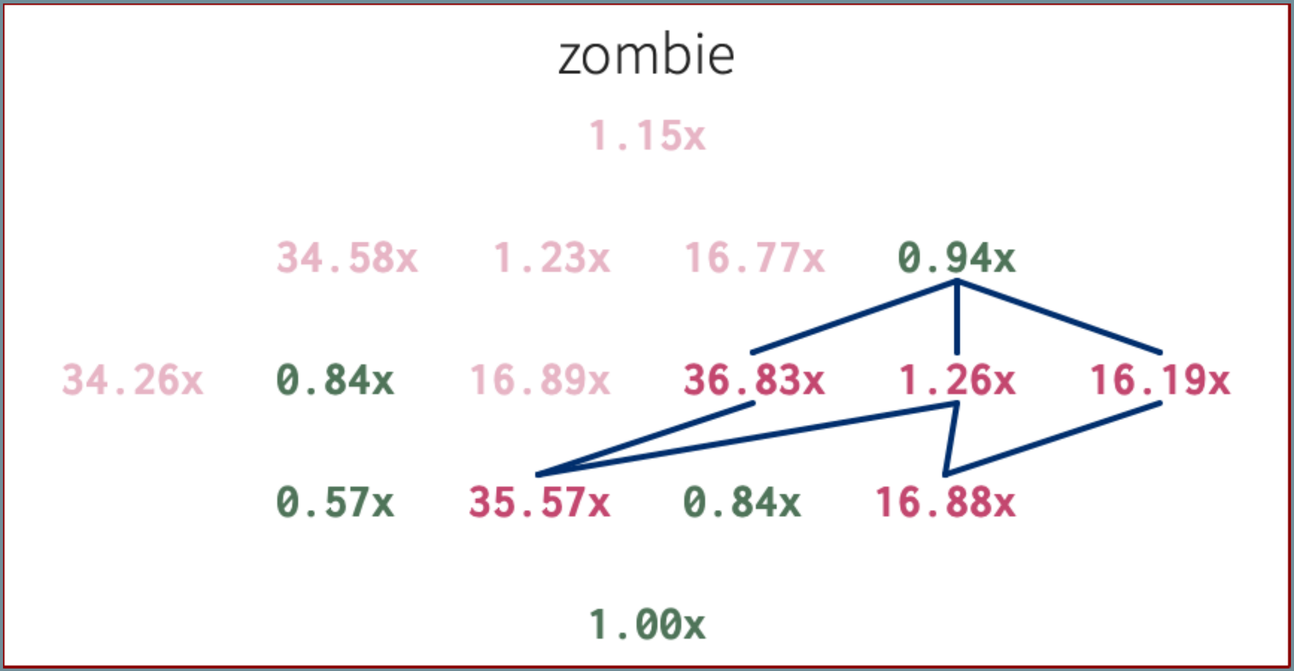
Best-Case Lattice



Best-Case Lattice



Best-Case Lattice



Best-Case Lattice

dungeon

1.30x

2.33x 1.21x 4.01x 472.06x 1.47x

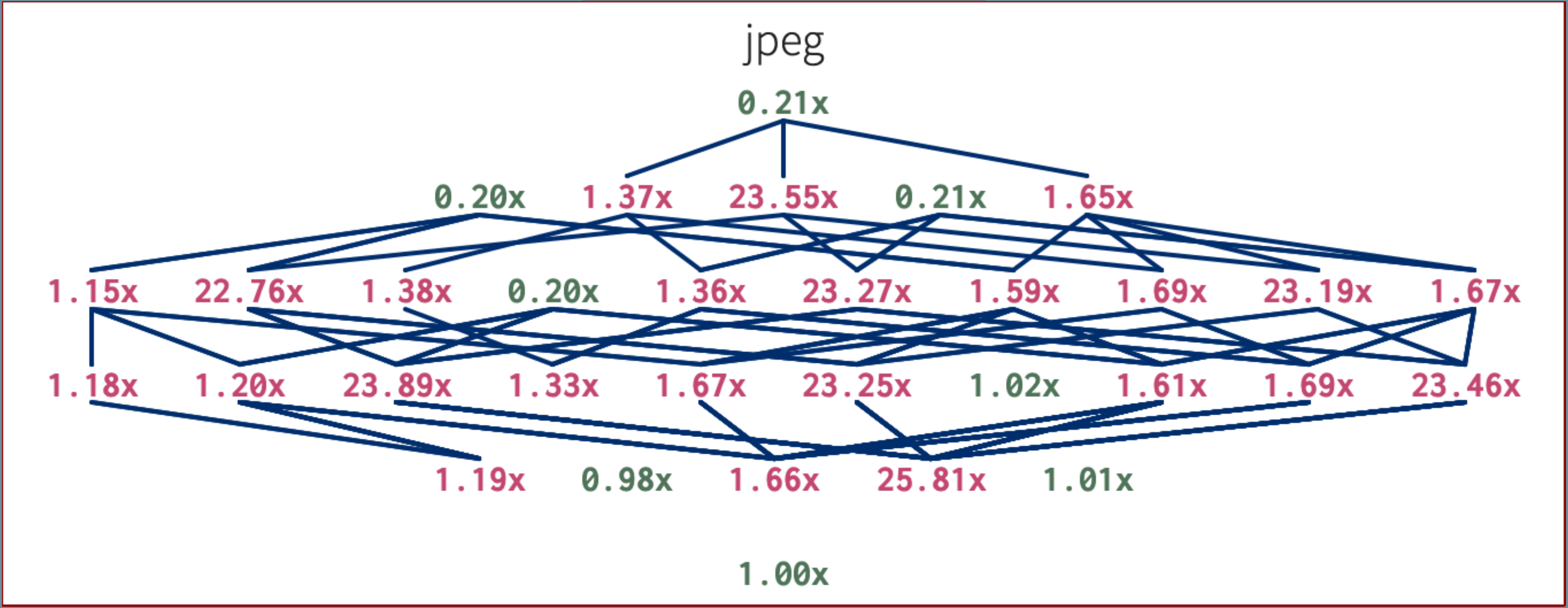
2.18x 3.98x 3.97x 492.31x 478.60x 1.56x 2.52x 1.44x 3.61x 498.27x

3.91x 483.09x 1.65x 1.58x 2.53x 3.63x 3.56x 493.17x 497.65x 1.15x

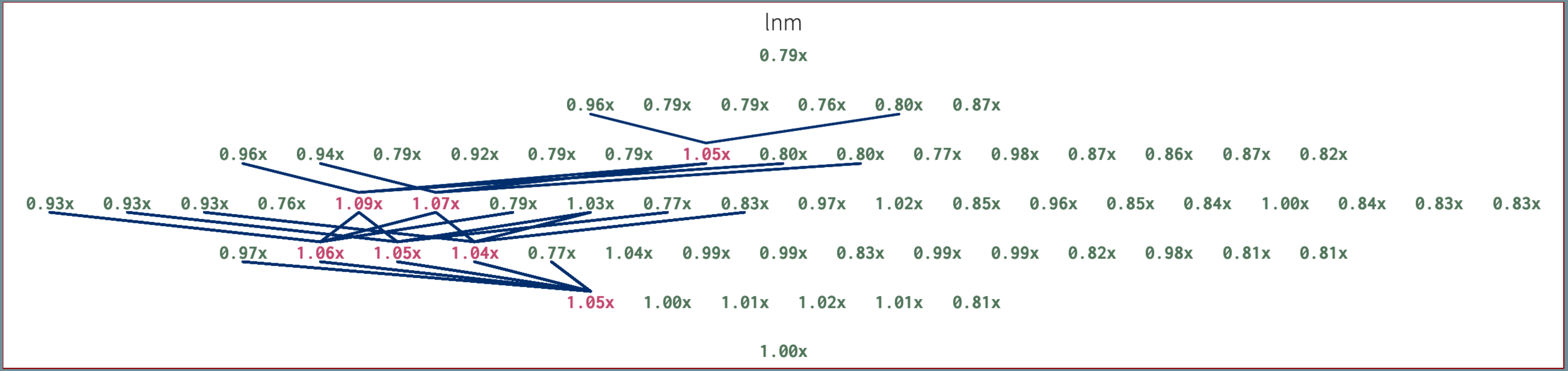
1.55x 3.62x 497.28x 1.13x 1.12x

1.00x

Best-Case Lattice

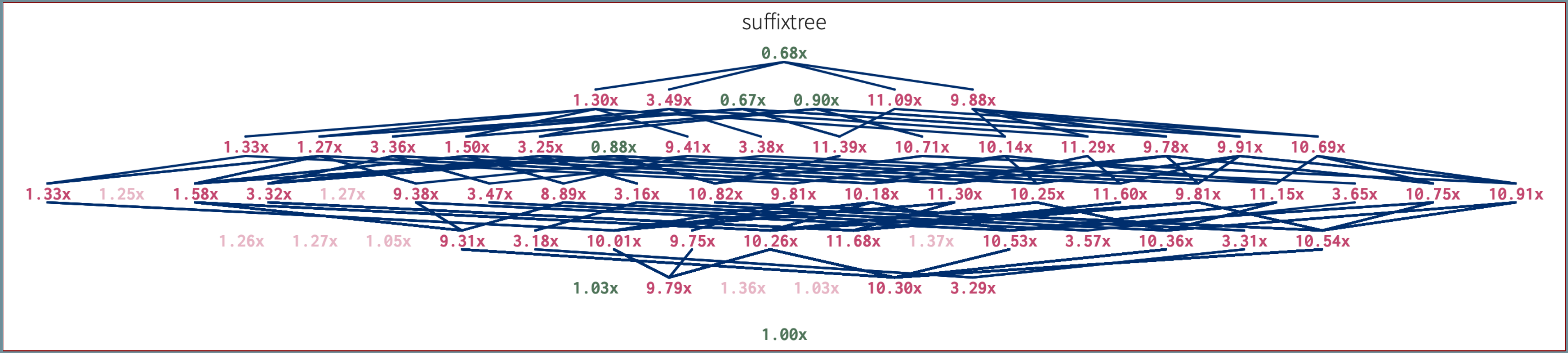


Best-Case Lattice



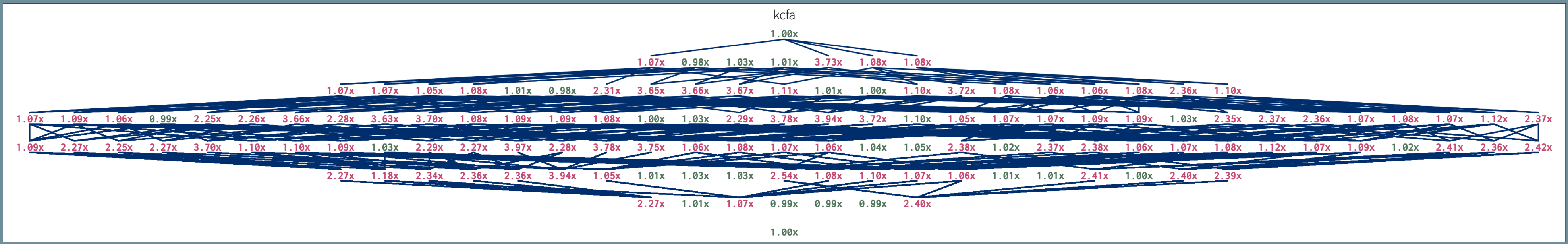
lnm
0.79x
0.96x 0.79x 0.79x 0.76x 0.80x 0.87x
0.96x 0.94x 0.79x 0.92x 0.79x 0.79x 1.05x 0.80x 0.80x 0.77x 0.98x 0.87x 0.86x 0.87x 0.82x
0.93x 0.93x 0.93x 0.76x 1.09x 1.07x 0.79x 1.03x 0.77x 0.83x 0.97x 1.02x 0.85x 0.96x 0.85x 0.84x 1.00x 0.84x 0.83x 0.83x
0.97x 1.06x 1.05x 1.04x 0.77x 1.04x 0.99x 0.99x 0.83x 0.99x 0.99x 0.82x 0.98x 0.81x 0.81x
1.05x 1.00x 1.01x 1.02x 1.01x 0.81x
1.00x

Best-Case Lattice



suffixtree
0.68x
1.30x 3.49x 0.67x 0.90x 11.09x 9.88x
1.33x 1.27x 3.36x 1.50x 3.25x 0.88x 9.41x 3.38x 11.39x 10.71x 10.14x 11.29x 9.78x 9.91x 10.69x
1.33x 1.25x 1.58x 3.32x 1.27x 9.38x 3.47x 8.89x 3.16x 10.82x 9.81x 10.18x 11.30x 10.25x 11.60x 9.81x 11.15x 3.65x 10.75x 10.91x
1.26x 1.27x 1.05x 9.31x 3.18x 10.01x 9.75x 10.26x 11.68x 1.37x 10.53x 3.57x 10.36x 3.31x 10.54x
1.03x 9.79x 1.36x 1.03x 10.30x 3.29x
1.00x

Best-Case Lattice


kcfa
1.00x
1.07x 0.98x 1.03x 1.01x 3.73x 1.08x 1.08x
1.07x 1.07x 1.05x 1.08x 1.01x 0.98x 2.31x 3.65x 3.66x 3.67x 1.11x 1.01x 1.00x 1.10x 3.72x 1.08x 1.06x 1.06x 1.08x 2.36x 1.10x
1.07x 1.09x 1.06x 0.99x 2.25x 2.26x 3.66x 2.28x 3.63x 3.70x 1.08x 1.09x 1.09x 1.08x 1.00x 1.03x 2.29x 3.78x 3.94x 3.72x 1.10x 1.05x 1.07x 1.07x 1.09x 1.09x 1.03x 2.35x 2.37x 2.36x 1.07x 1.08x 1.07x 1.12x 2.37x
1.09x 2.27x 2.25x 2.27x 3.70x 1.10x 1.10x 1.09x 1.03x 2.29x 2.27x 3.97x 2.28x 3.78x 3.75x 1.06x 1.08x 1.07x 1.06x 1.04x 1.05x 2.38x 1.02x 2.37x 2.38x 1.06x 1.07x 1.08x 1.12x 1.07x 1.09x 1.02x 2.41x 2.36x 2.42x
2.27x 1.18x 2.34x 2.36x 2.36x 3.94x 1.05x 1.01x 1.03x 1.03x 2.54x 1.08x 1.10x 1.07x 1.06x 1.01x 1.01x 2.41x 1.00x 2.40x 2.39x
2.27x 1.01x 1.07x 0.99x 0.99x 0.99x 2.40x
1.00x

Takeaways

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

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

* shallow types do not help