



GTP Benchmarks for Gradual Typing Performance

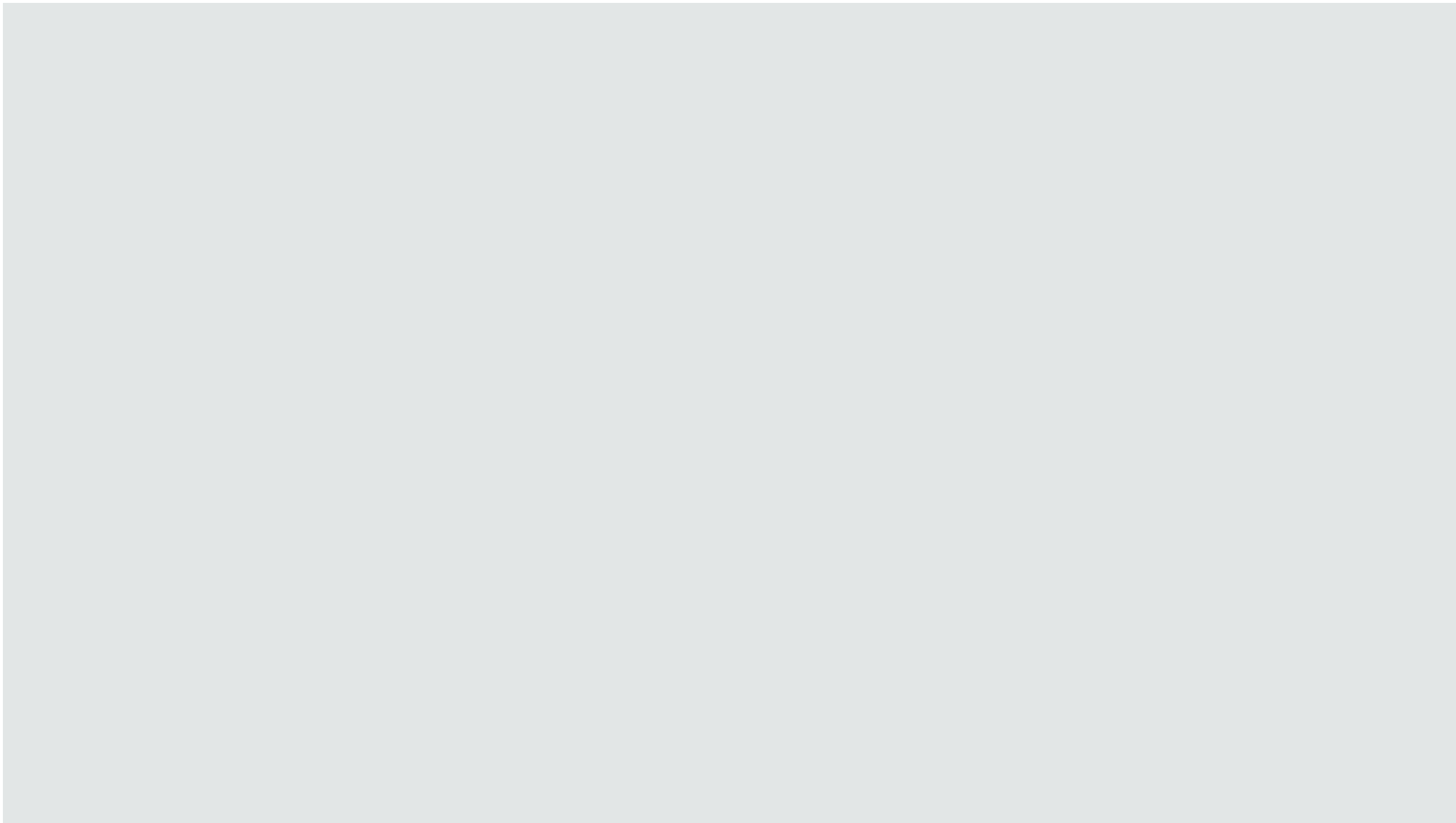
Ben Greenman



BROWN



THE
UNIVERSITY
OF UTAH



Benchmarks + Experiments are important.

Benchmarks + Experiments are important.

Must be:

Relevant

Rigorous

Reproducible

Benchmarks + Experiments are important.

Must be:

Relevant

Rigorous

Reproducible

Example:



Benchmarks + Experiments are important.

Must be:

Relevant

Rigorous

Reproducible

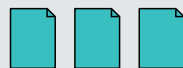
Benchmarks + Experiments are important.

Must be:

Relevant

Rigorous

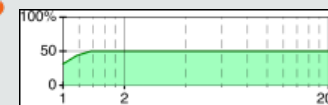
Reproducible



benchmarks



measurement



visualization

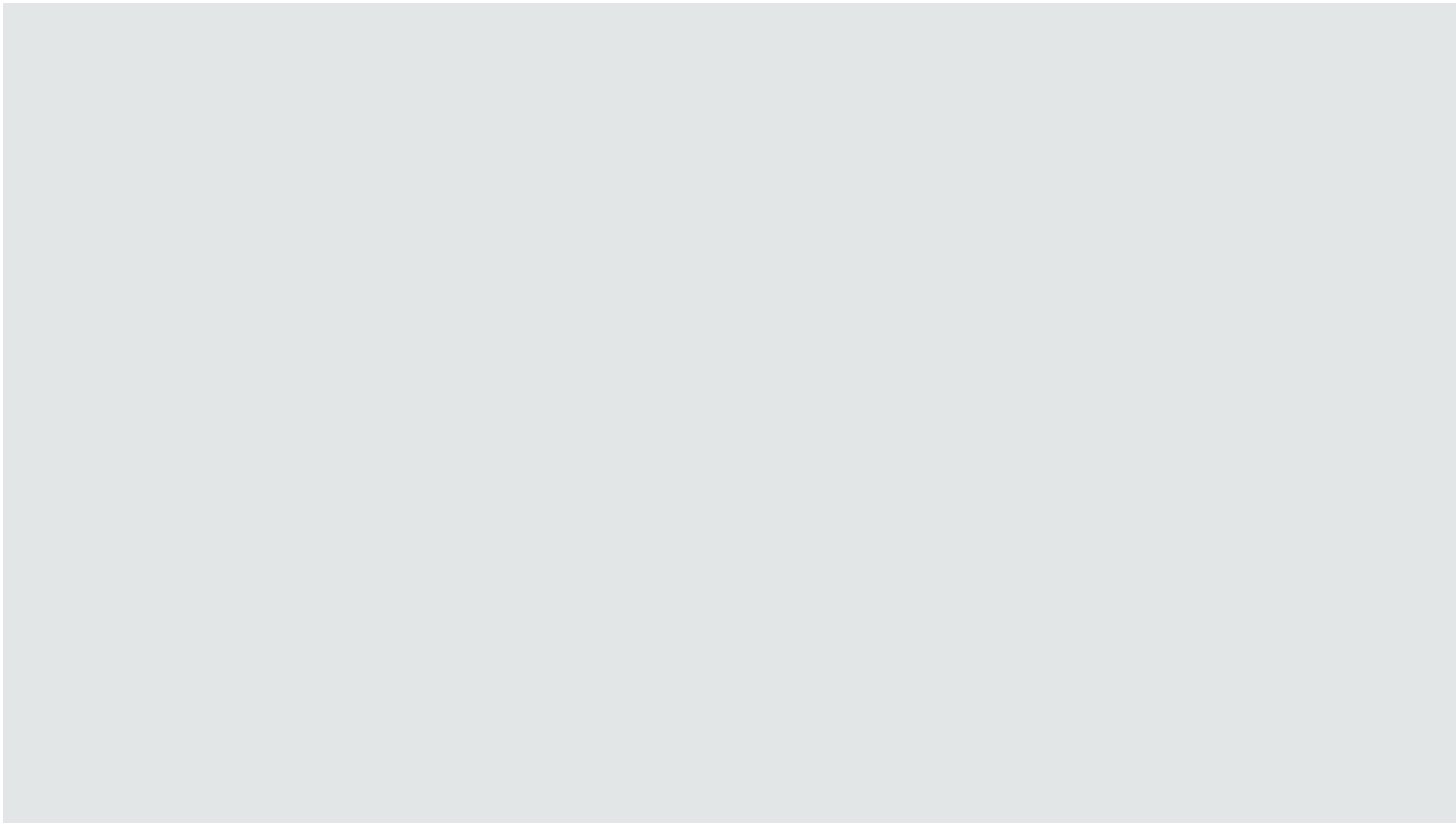
How to encourage **domain-specific** benchmarks?

How to encourage **domain-specific** benchmarks?

Main takeaway: **think like a practitioner**

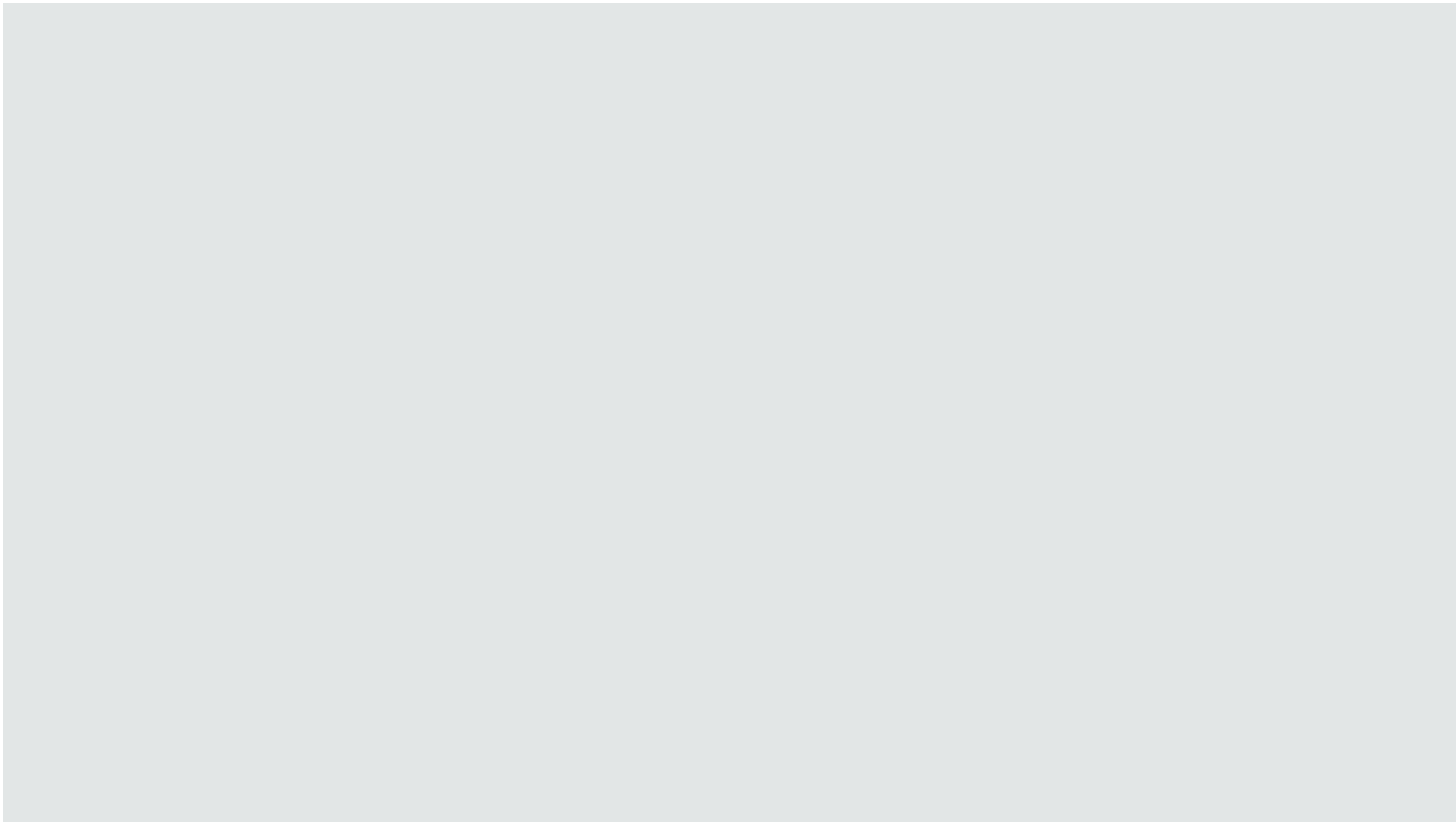








Gradual
GTP = Typing
Performance



Gradual Typing

Untyped



Typed

Gradual Typing

Untyped



Typed

```
def join(d0,d1,sort,how):  
    ....
```

DataFrame

bool

Left|Right

```
def join(d0:DataFrame,  
        d1:DataFrame,  
        sort:bool,  
        how:Left|Right)  
    -> DataFrame:  
    ....
```

Types where useful ... and nowhere else!

Gradual Typing

Untyped



Typed

Gradual Typing

Untyped



Typed

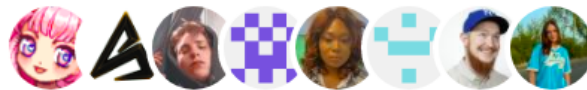


TypeScript is JavaScript
with syntax for types.

Used by

19.6m

DefinitelyTyped



+ 19,600,849

Gradual Typing Performance?

Untyped



Typed

Gradual Typing Performance?

Untyped



Typed

Run-time cost of sound types

Gradual Typing Performance?

Untyped



Typed

Run-time cost of sound types

```
def join(d0:DataFrame,  
        d1:DataFrame,  
        sort:bool,  
        how:Left|Right)  
  -> DataFrame:  
  ....
```

Gradual Typing Performance?

Untyped



Typed

Run-time cost of sound types

?? `join(x,y,z)` How to validate?

```
def join(d0:DataFrame,  
         d1:DataFrame,  
         sort:bool,  
         how:Left|Right)  
  -> DataFrame:  
  ....
```

Gradual Typing Performance?

Untyped



Typed

Run-time cost of sound types

?? `join(x,y,z)` How to validate?

```
def join(d0:DataFrame,  
        d1:DataFrame,  
        sort:bool,  
        how:Left|Right)  
  -> DataFrame:  
  ....
```



(TypeScript does not validate)

Gradual Typing Performance?

Untyped



Typed

Run-time cost of sound types

?? `join(x,y,z)` How to validate?

```
def join(d0:DataFrame,  
         d1:DataFrame,  
         sort:bool,  
         how:Left|Right)  
  -> DataFrame:  
  ....
```

Gradual Typing Performance?

Untyped

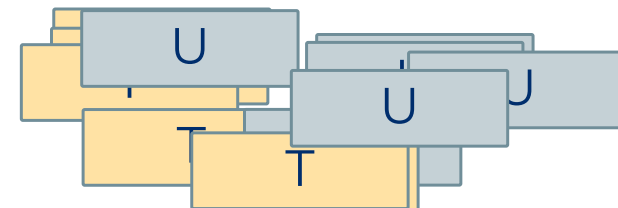


Typed

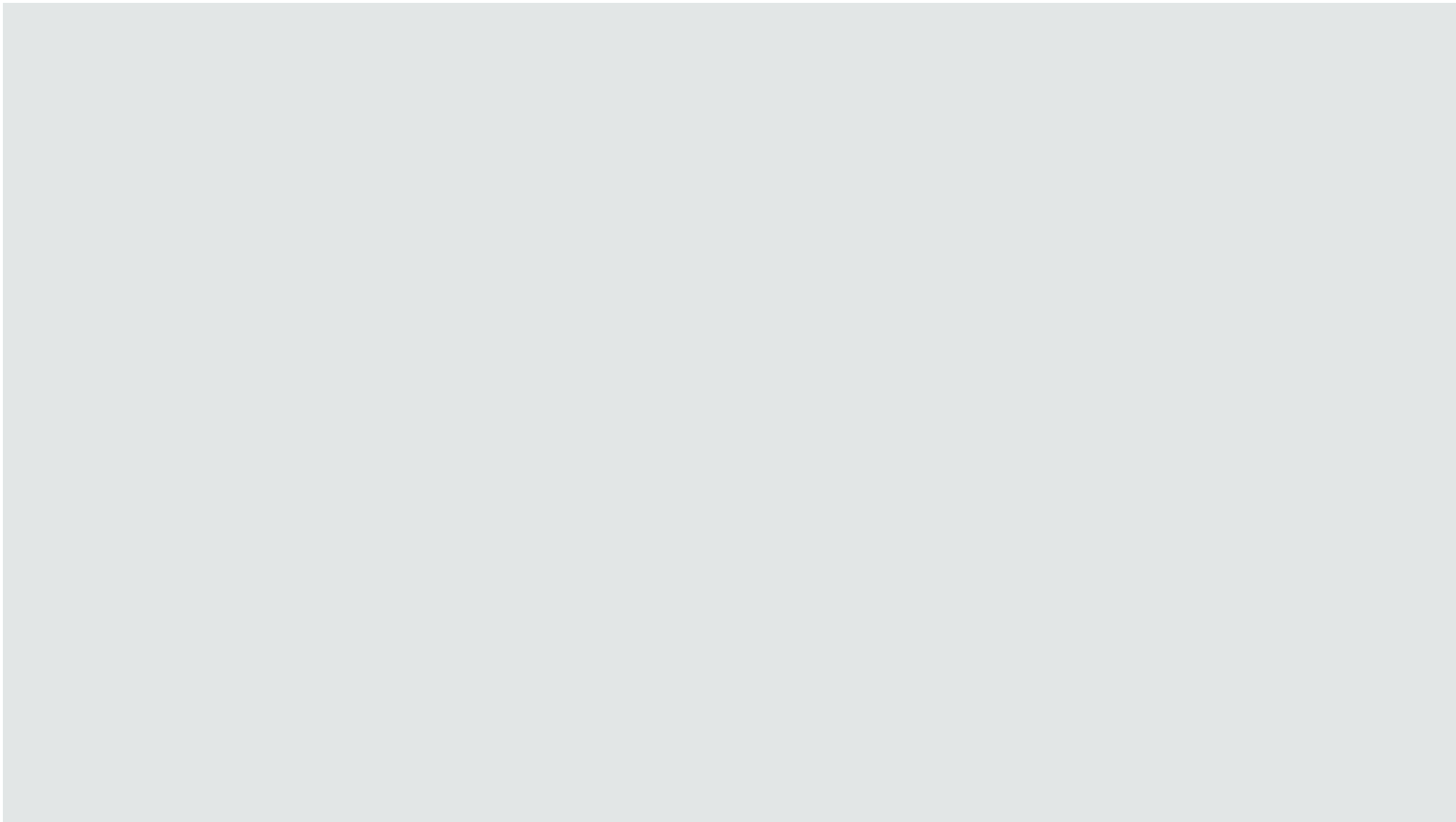
Run-time cost of sound types

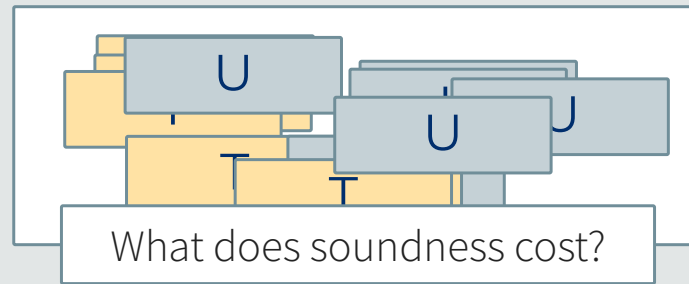
?? `join(x,y,z)` How to validate?

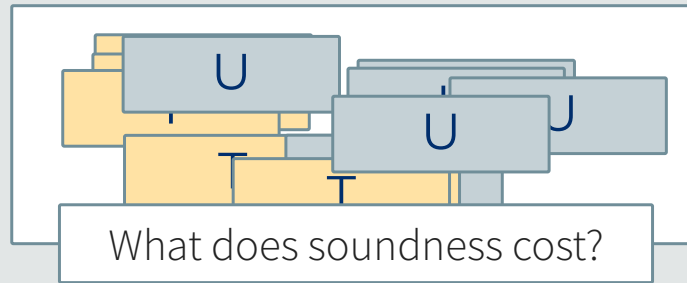
```
def join(d0:DataFrame,  
        d1:DataFrame,  
        sort:bool,  
        how:Left|Right)  
  -> DataFrame:  
  ....
```



Many interactions,
Maybe high costs



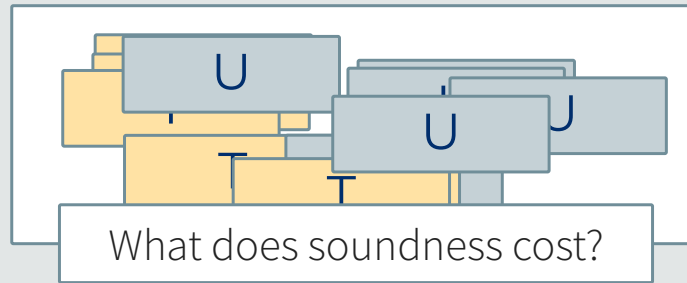




Typed Racket

- + object types, function types, ...
- + type-driven optimizer

Worst-case slowdown: **1.4x**
ecoop '15

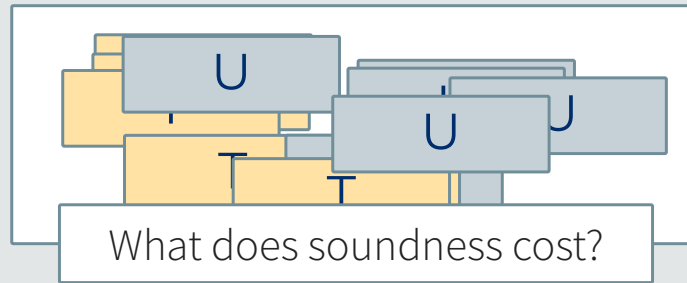


Typed Racket

- + object types, function types, ...
- + type-driven optimizer

Worst-case slowdown: **1.4x**
e coop '15





Typed Racket

- + object types, function types, ...
- + type-driven optimizer

Worst-case slowdown: **1.4x**
ecoop '15

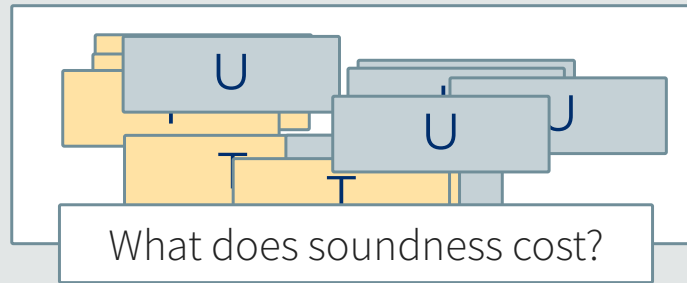
2x



30x

12,000x

(1ms to 12sec)



Typed Racket

- + object types, function types, ...
- + type-driven optimizer

Worst-case slowdown: **1.4x**
ecoop '15



2x

30x

12,000x

(1ms to 12sec)

warning on use trie functions in #lang racket?



johnbclements
to Racket Users

This program constructs a trie containing exactly two keys; ea
... ..

What does soundness cost?

Need a way to measure!



GTP Benchmarks



GTP Benchmarks

What to measure?

Cost of sound types



GTP Benchmarks

What to measure?

Which programs?

Cost of sound types

... Any



GTP Benchmarks

What to measure?

Cost of sound types

Which programs?

... Any

How fast is good enough? ???



GTP Benchmarks

What to measure?	Cost of sound types
Which programs?	... Any
How fast is good enough?	???
What is a benchmark?	???



GTP Benchmarks

What to measure?	Cost of sound types
Which programs?	... Any
How fast is good enough?	???
What is a benchmark?	???



Think like a practitioner

What is a gradual typing benchmark?

Untyped code?

```
def join(d0,d1,sort,how):
```

Not enough.

Typed code?

```
def join(d0:DataFrame, ...):
```

Not enough.

What is a gradual typing benchmark?

Untyped code?

```
def join(d0,d1,sort,how):
```

Not enough.

Typed code?

```
def join(d0:DataFrame, ...):
```

Not enough.



GT promise: can mix typed + untyped code

Need to measure **all configurations**

What is a gradual typing benchmark?

1. Start with a program

```
def join(d0,d1,sort,how):  
    ....
```

What is a gradual typing benchmark?

1. Start with a program

```
def join(d0,d1,sort,how):  
    ....
```

2. Add types

```
def join(d0:DataFrame,  
        d1:DataFrame,  
        sort:bool,  
        how:Left|Right)  
    -> DataFrame:  
    ....
```

What is a gradual typing benchmark?

1. Start with a program

```
def join(d0,d1,sort,how):  
    ....
```

2. Add types

```
def join(d0:DataFrame,  
        d1:DataFrame,  
        sort:bool,  
        how:Left|Right)  
    -> DataFrame:  
    ....
```

3. Explore all configurations

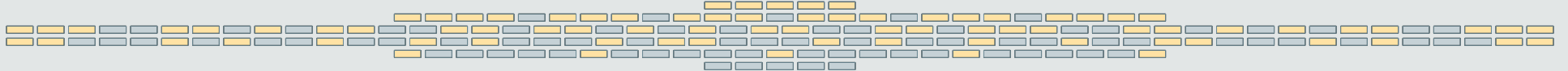


What is a gradual typing benchmark?



Explore by **module**

5 modules, 32 configurations

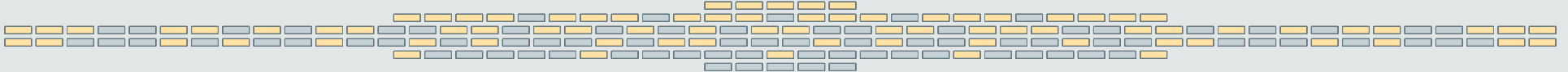


What is a gradual typing benchmark?



Explore by **module**

5 modules, 32 configurations



2 modules, 4 configurations

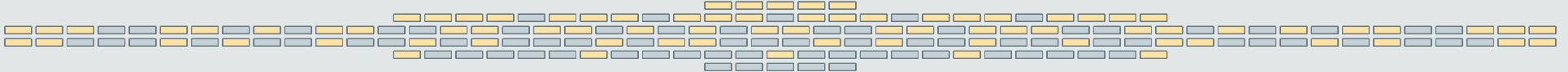


What is a gradual typing benchmark?



Explore by **module**

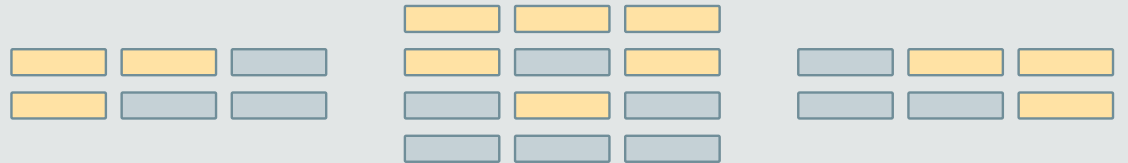
5 modules, 32 configurations



2 modules, 4 configurations



3 modules, 8 configurations



Where to find benchmarks?

Where to find benchmarks?



Wherever people share code



Where to find benchmarks?



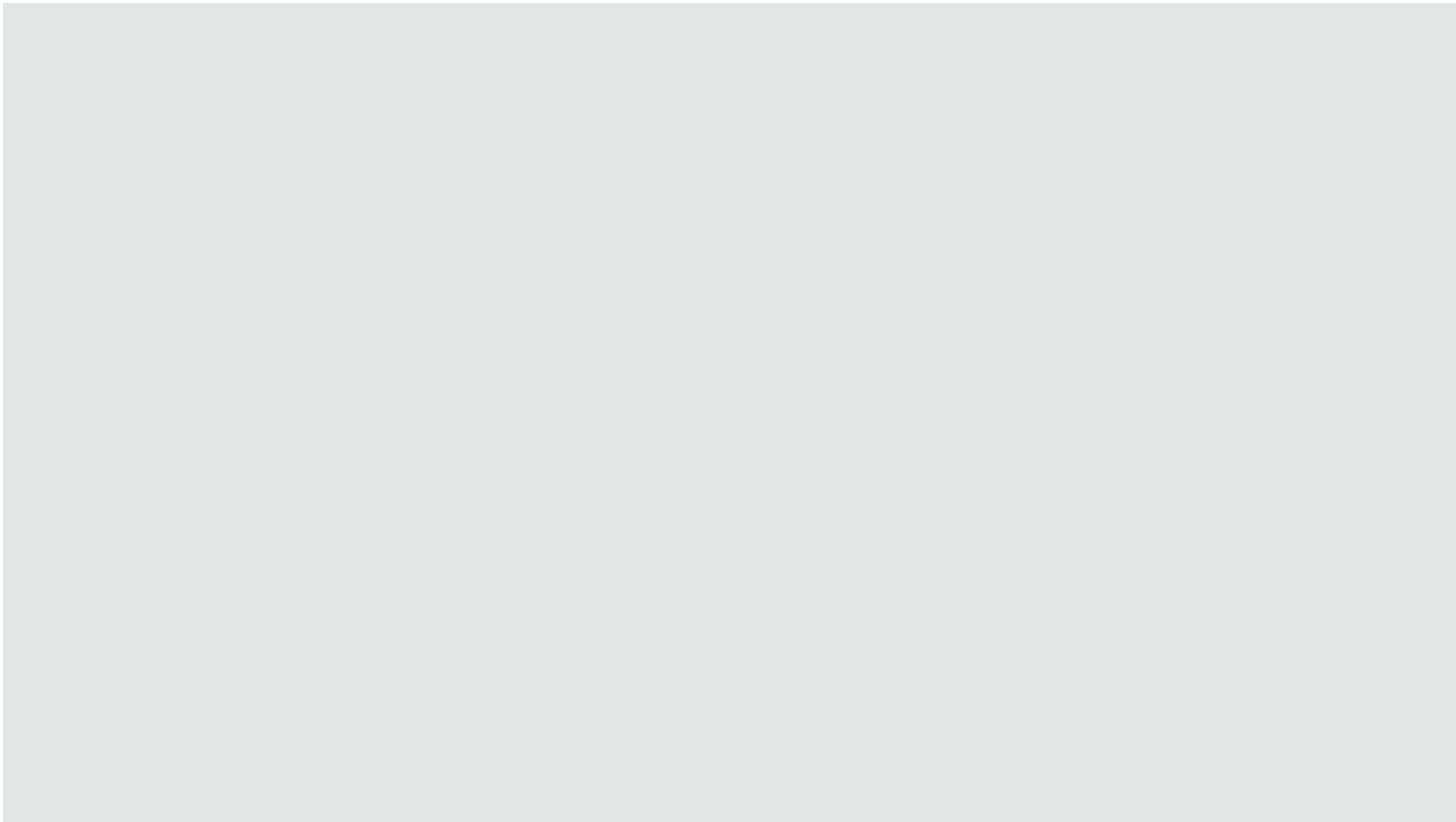
Wherever people share code



Current status: 21 benchmarks, +40k configurations

Table 1: Benchmarks overview: purpose and characteristics

Benchmark	Purpose	T Init	U Lib	T Lib	Adapt	HOF	Poly	Rec	Mut	Imm	Obj	Cls
sieve	<i>prime generator</i>	○	○	○	●	○	○	●	○	●	○	○
forth	<i>Forth interpreter</i> [51]	○	○	○	○	○	○	●	○	●	●	●
fsm	<i>economy simulation</i> [33]	○	○	○	○	○	○	○	●	●	○	○
fsmoo	<i>economy simulation</i> [34]	○	○	○	○	○	○	○	●	●	●	○
mbta	<i>subway map</i>	●	●	○	○	○	○	○	○	○	●	○
morsecode	<i>Morse code trainer</i> [23, 148]	○	○	○	○	○	○	○	●	○	○	○
zombie	<i>HTDP game</i> [151]	○	○	○	●	●	○	●	○	●	○	○
zordoz	<i>bytecode tools</i> [53]	○	●	○	●	●	○	●	●	●	○	○
dungeon	<i>maze generator</i>	○	○	○	○	●	●	●	●	●	●	●
image	<i>image tools</i> [161]	○	○	○	○	○	○	○	○	○	○	○



How to analyze the data?

How to analyze the data?

How to summarize?

How to compare?

How to scale?

How to analyze the data?

How to summarize?

How to analyze the data?

How to summarize?

Some ideas:
worst-case? average? median?

How to analyze the data?

How to summarize?

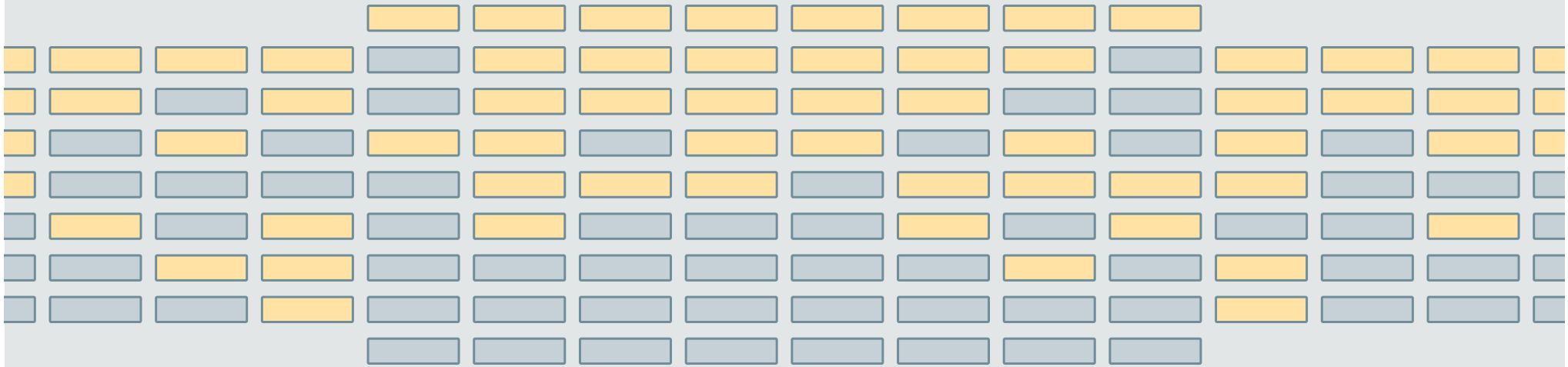
worst-case



median?



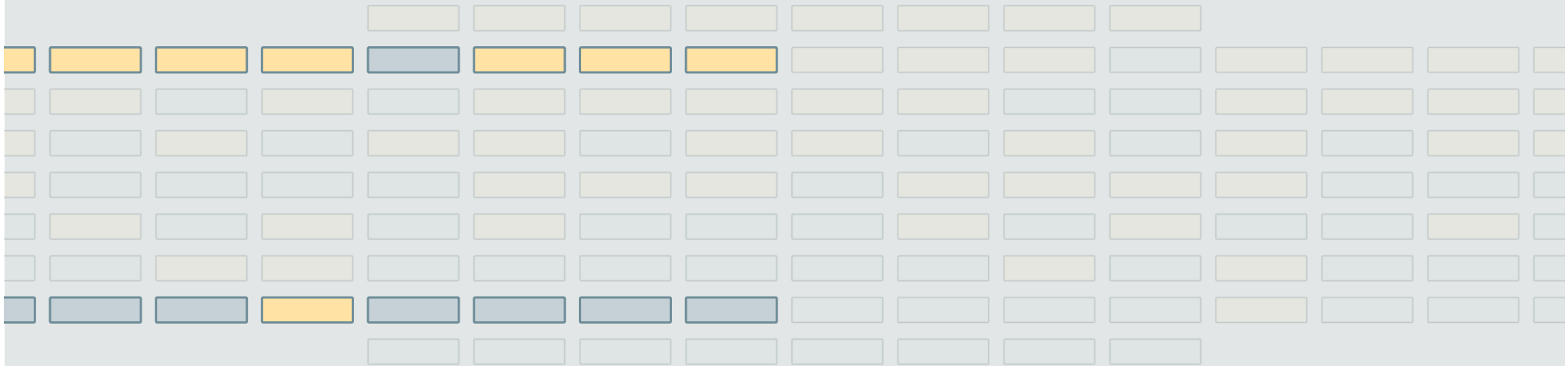
How to analyze the data?





How to analyze the data?

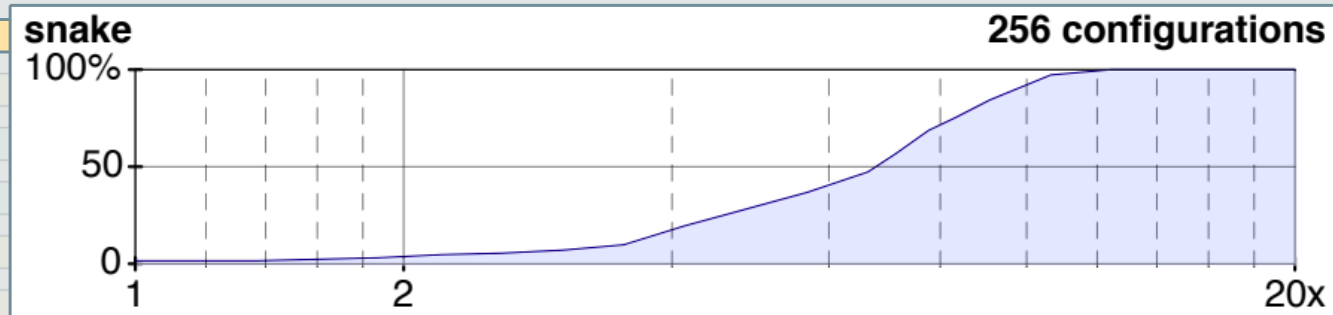
Too slow = useless!





How to analyze the data?

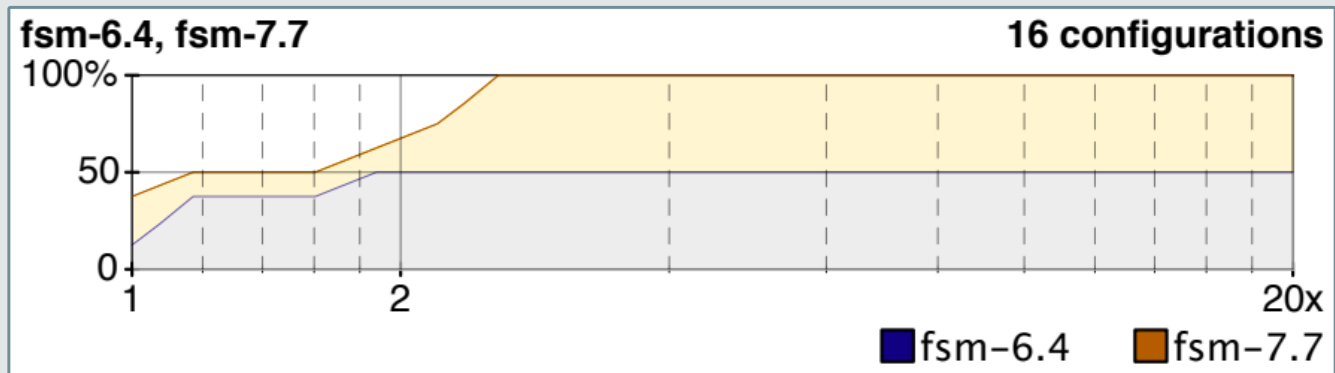
Too slow = useless!



x-axis = limit for "too slow" vs. untyped code (log scale)
y-axis = % usable configs.

How to compare

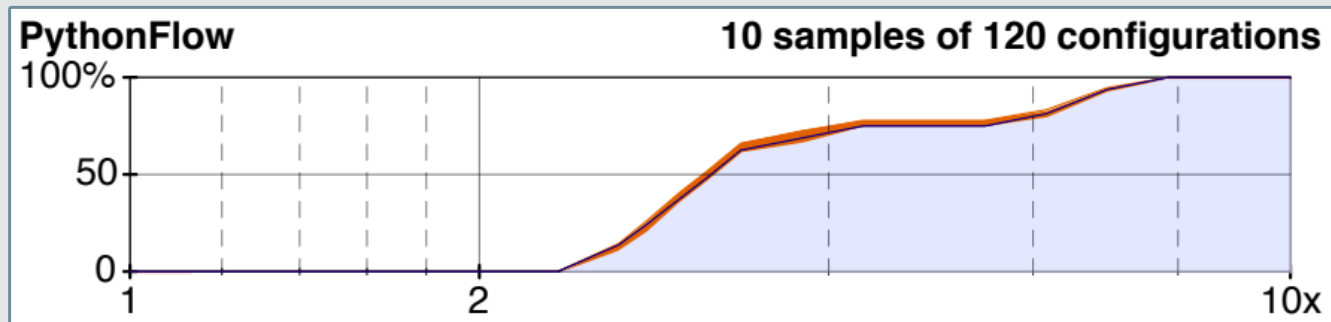
How to compare

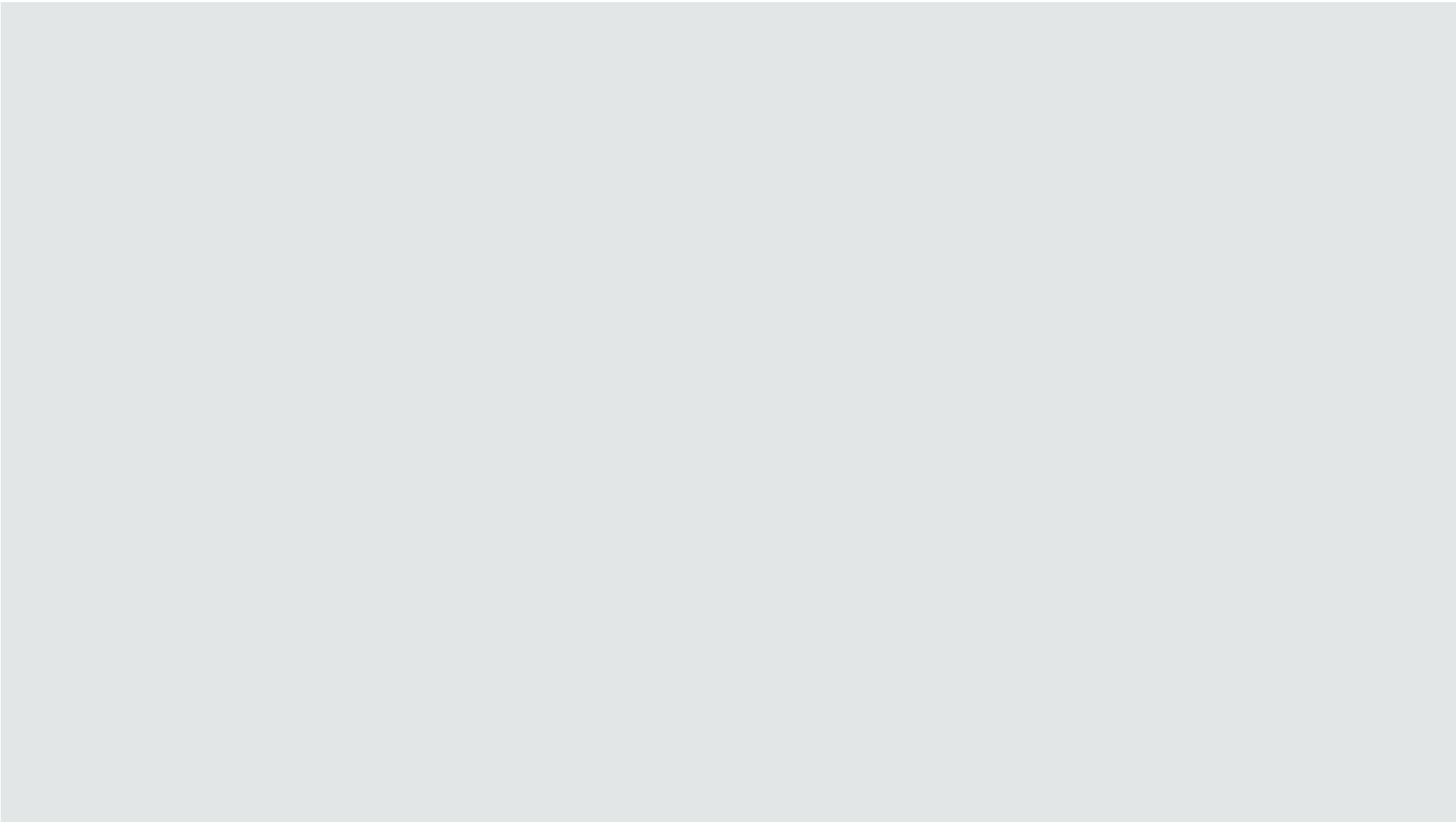


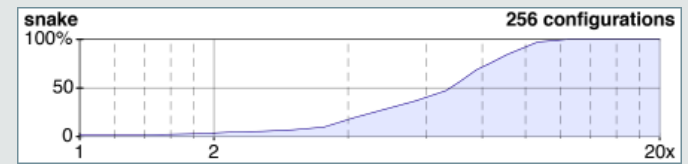
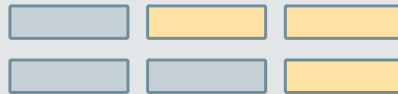
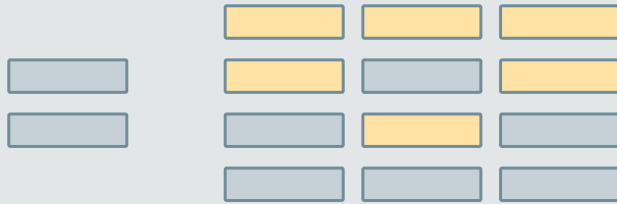
How to scale

How to scale

Linear-size random samples

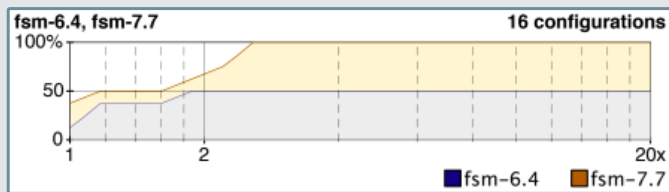




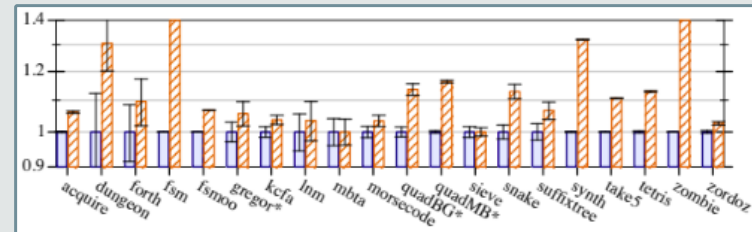
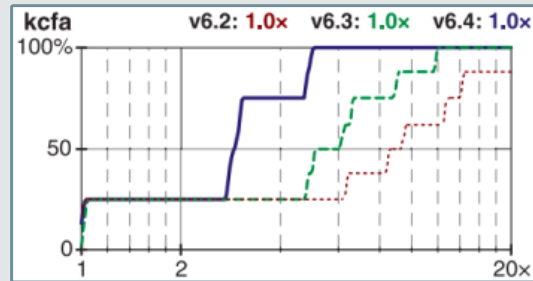
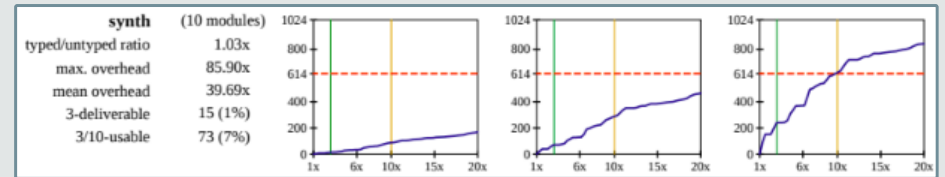
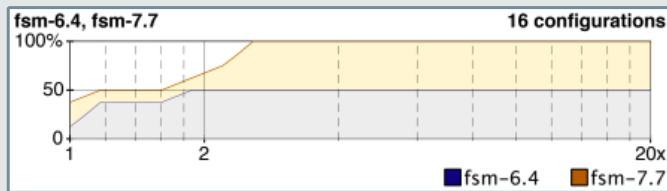




Software for Measurement



Software for Measurement



Software for Measurement

Software for Measurement

[top](#) [< prev](#) [up](#) [next >](#)

► **GTP measure**

GTP measure

1 Command-line: `raco gtp-measure`

1.1 Stages of measurement

1.2 Configuration and Data Files

2 GTP targets

2.1 Typed-Untyped Configuration

v.7.8.0.6

GTP measure

by Ben Greenman

`(require gtp-measure)` `package: gtp-measure`

For benchmarking.

1 Command-line: `raco gtp-measure`

The `gtp-measure` `raco` command is a tool for measuring the performance of a set of [gtp-](#)

Software for Measurement

The screenshot shows the manual page for 'GTP measure' in the Raco documentation. On the left is a sidebar with a search bar and a table of contents. The main content area on the right includes the version number 'v.7.8.0.6', the title 'GTP measure', the author 'by Ben Greenman', a code block for the package requirement, a description 'For benchmarking.', and a section for the command-line usage.

...search manuals...

top ← prev up next →

► GTP measure

GTP measure

1 Command-line: `raco gtp-measure`

1.1 Stages of measurement

1.2 Configuration and Data Files

2 GTP targets

2.1 Typed-Untyped Configuration

v.7.8.0.6

GTP measure

by Ben Greenman

```
(require gtp-measure)
```

package: gtp-measure

For benchmarking.

1 Command-line: `raco gtp-measure`

The `gtp-measure` `raco` command is a tool for measuring the performance of a set of [gtp-](#)

Interruptible! Space-Efficient. Configurable.

Software for Measurement

...search manuals...

top ← prev up next →

► GTP measure

GTP measure

1 Command-line: `raco gtp-measure`

1.1 Stages of measurement

1.2 Configuration and Data Files

2 GTP targets

2.1 Typed-Untyped Configuration

v.7.8.0.6

GTP measure

by Ben Greenman

```
(require gtp-measure)
```

For benchmarking.

1 Command-line: `raco gtp-measure`

The `gtp-measure` `raco` command is a tool for measuring the performance of Racket programs. It is designed to be interruptible and space-efficient.

- `key:bin` = `"/Users/ben/code/racket/fork/racket/bin/"`
- `key:iterations` = `8`
- `key:jit-warmup` = `1`
- `key:num-samples` = `10`
- `key:sample-factor` = `10`
- `key:cutoff` = `9`
- `key:entry-point` = `"main.rkt"`
- `key:start-time` = `0`
- `key:time-limit` = `#f`

Interruptible! Space-Efficient

Software for Measurement

Software for Measurement

Tiny DSL for experiments

```
#lang gtp-measure/manifest

#:config #hash(
  (bin . "/home/gtp/racket-8.8/bin/")
  (cutoff . 6)
  (num-samples . 10))

/home/gtp/benchmarks/morsecode
/home/gtp/benchmarks/take5
```

Software for Measurement

Software for Measurement

DSL for data

```
#lang gtp-measure/output/typed-untyped
("00000" ("cpu time: 566 real time: 567 gc time: 62" ....))
("00001" ("cpu time: 820 real time: 822 gc time: 46" ....))
("00010" ("cpu time: 561 real time: 562 gc time: 46" ....))
("00011" ("cpu time: 805 real time: 807 gc time: 47" ....))
....
```

Software for Measurement

DSL for data

```
#lang gtp-measure/output/typed-untyped
("00000" ("cpu time: 566 real time: 567 gc time: 46" ....))
("00001" ("cpu time: 820 real time: 822 gc time: 46" ....))
("00010" ("cpu time: 561 real time: 562 gc time: 46" ....))
("00011" ("cpu time: 805 real time: 807 gc time: 47" ....))
....
```

5.3 Output Data: Typed-Untyped Target

```
#lang gtp-measure/output/typed-untyped
```

package: gtp-measure

Output data for a [gtp typed-untyped target](#).

[Each line is the result for one configuration. The first element is the name of the](#)

Software for Measurement

DSL for data

```
#lang gtp-measure/output/typed-untyped
("00000" ("cpu time: 566 real time: 567 ...))
("00001" ("cpu ...))
("00010" ("cpu ...))
("00011" ("cpu ...))
....
```

Running an output file prints a summary:

```
$ racket jpeg-2020-08-17.rktd
dataset info:
- num configs: 32
- num timings: 256
- min time: 110 ms
- max time: 8453 ms
- total time: 968537 ms
```

5.3 Output Data: Typed-Untyped Target

```
#lang gtp-measure/output/typed-untyped
```

package: gtp-measure

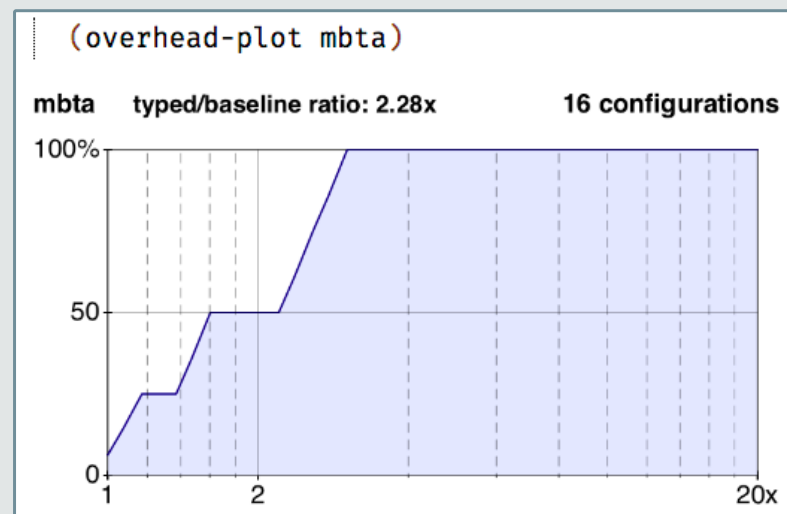
Output data for a [gtp typed-untyped target](#).

the result for one configuration. The first element is the name of the

```
me: 46" ....))
me: 46" ....))
me: 47" ....))
```

Software for Visualization

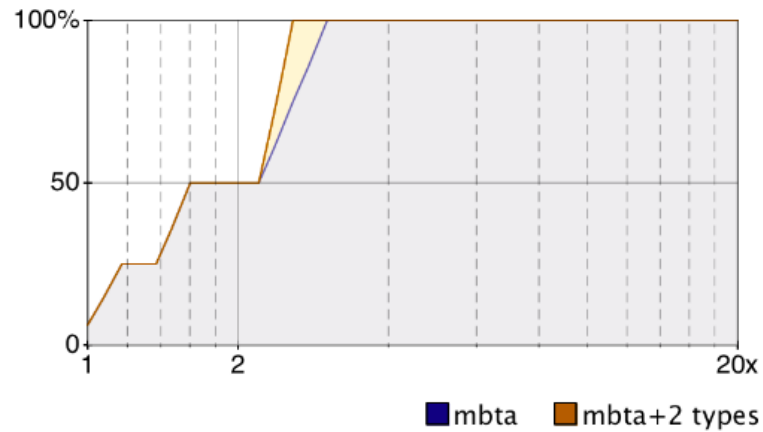
Software for Visualization



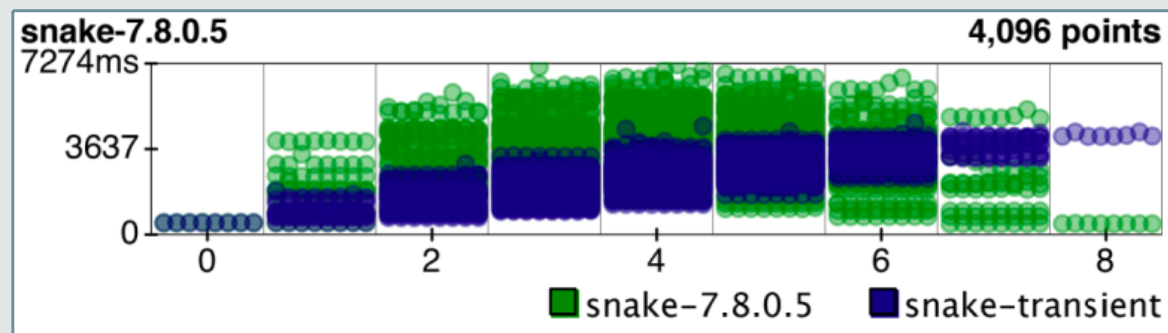
Software for Visualization

```
(parameterize ((*OVERHEAD-SHOW-RATIO* #f))  
  (overhead-plot (list mbta (typed-racket-info%best-typed-path mbta 2))))
```

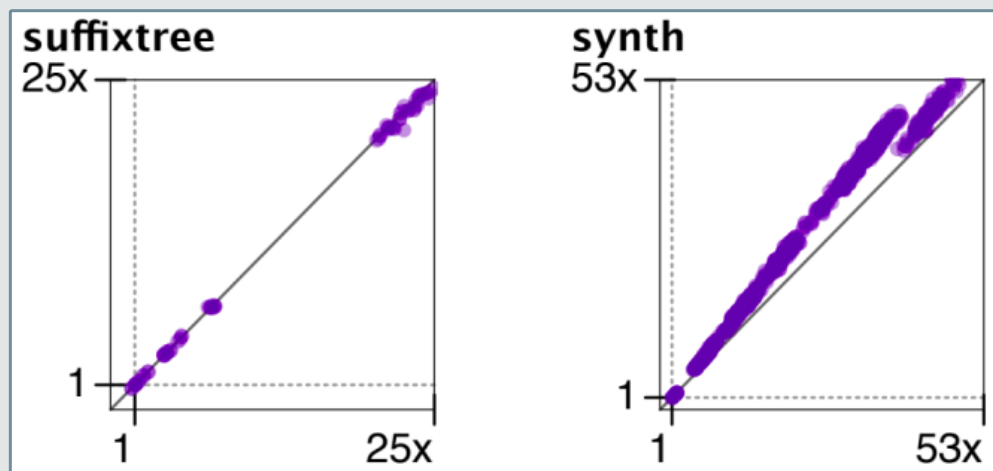
mbta, mbta+2 types 16 configurations



Software for Visualization



Software for Visualization

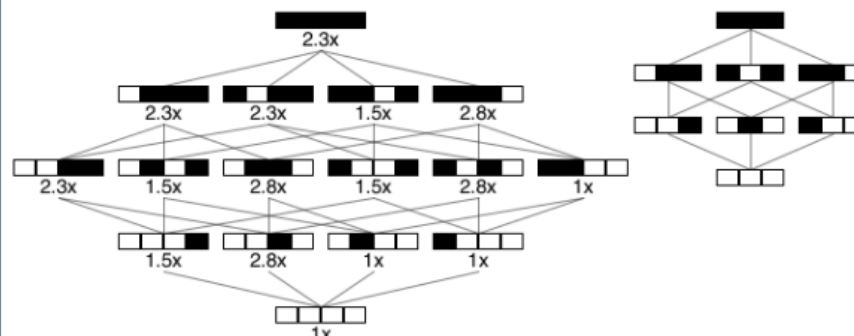


Software for Visualization

```
(performance-lattice pi) → pict? procedure
pi : (or/c performance-info? natural?)
```

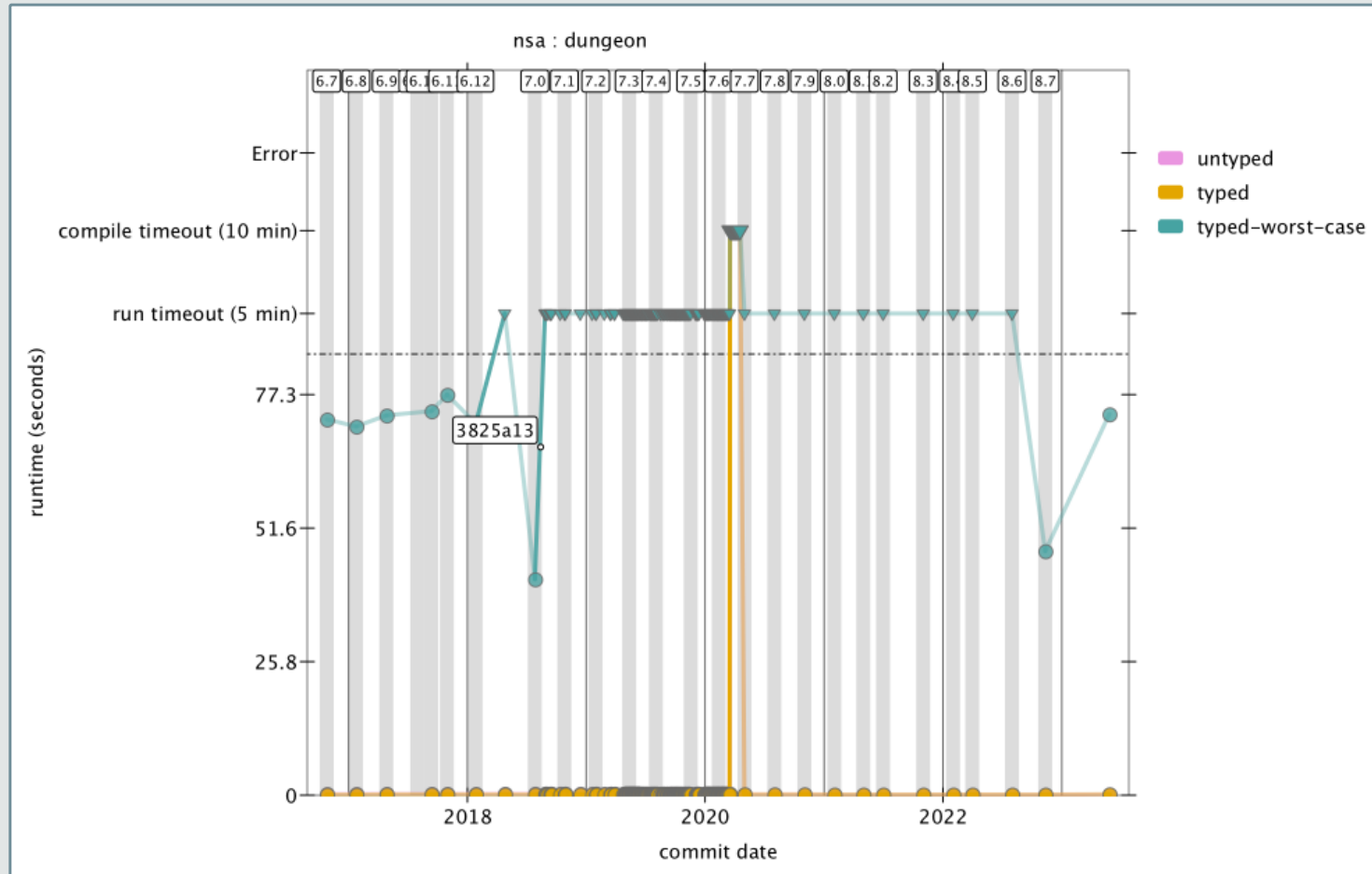
Given a `performance-info` structure, shows the overhead of every configuration in a lattice. Given a number, render an unlabeled lattice.

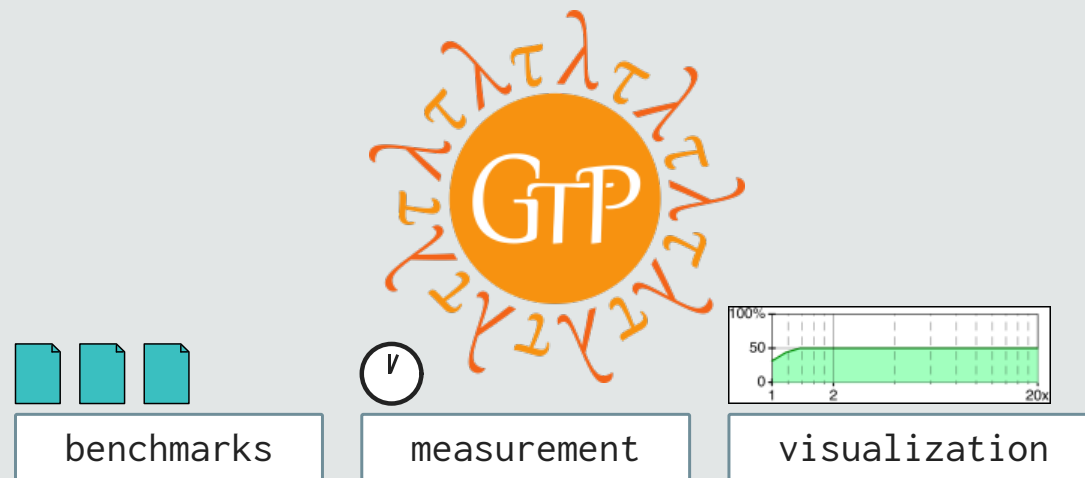
```
(parameterize ([*FONT-SIZE* 14]
               [*LATTICE-UNIT-WIDTH* 16]
               [*LATTICE-UNIT-HEIGHT* 12]
               [*LATTICE-CONFIG-X-MARGIN* 10]
               [*LATTICE-CONFIG-Y-MARGIN* 25]
               [*LATTICE-LINES? #true]
               [*LATTICE-LINE-ALPHA* 0.5])
(ht-append 4
 (performance-lattice mbta)
 (performance-lattice 3)))
```

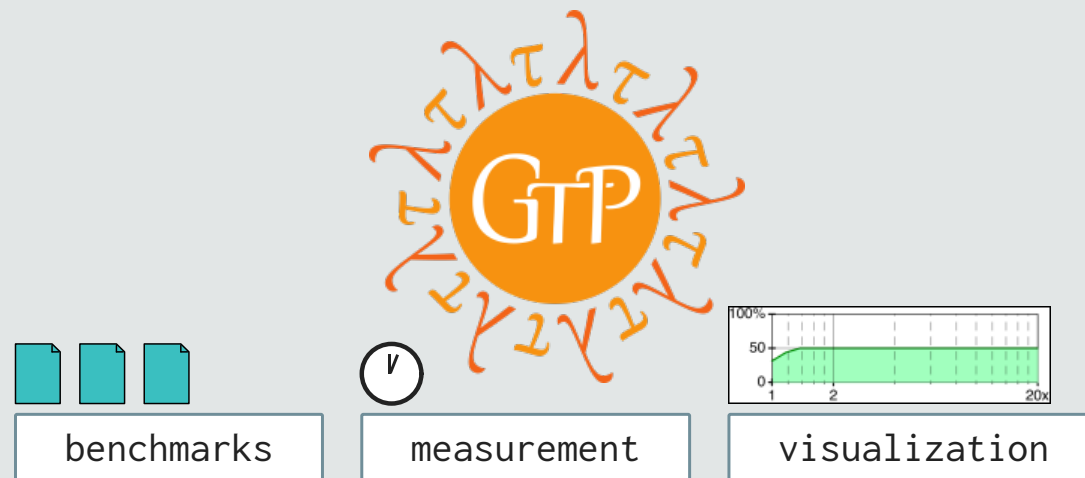


Continuous Testing

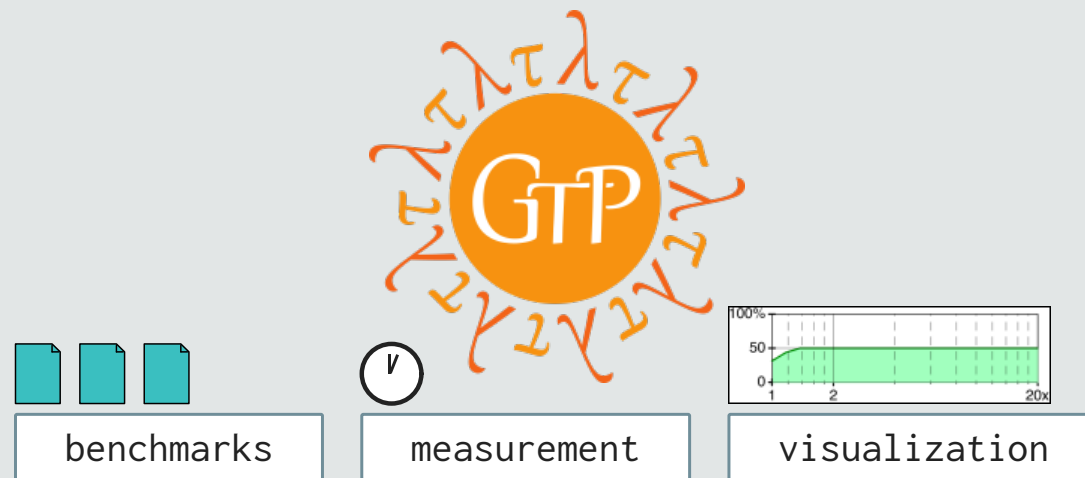
Continuous Testing







All 3 important ... but not to everyone



All 3 important ... but not to everyone

Lesson 2: loose coupling helps adoption

Still ... low adoption

Still ... low adoption

2014: few experiments,
~2 gradual configurations

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 for prototyping and delivering large and complex systems. When it comes to maintaining and evolving these systems, the lack of explicit static typing becomes a bottleneck. In response, researchers

many cases, the systems start as innocent prototypes. Soon enough, though, they grow into complex, multi-module programs, at which point the engineers realize that they are facing a maintenance nightmare, mostly due to the lack of reliable type information.

Gradual typing [21, 26] proposes a language-based solution to this problem: *gradual static typing*. The idea is to extend

Still ... low adoption

2014: few experiments,
~2 gradual configurations

Lately: few experiments,
but thorough

Ok?

Is Sound Gradual Typing Dead?

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

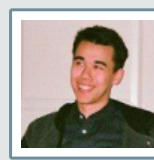
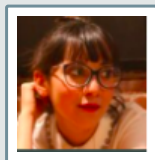
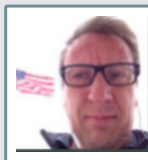
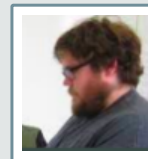
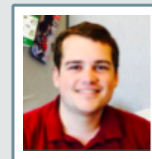
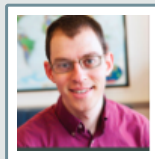
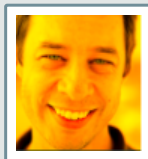


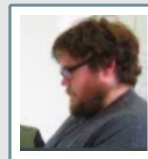
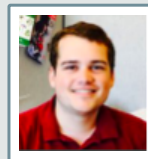
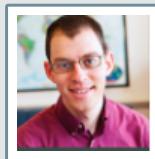
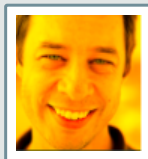
Abstract

Programmers have come to embrace dynamically-typed languages for prototyping and delivering large and complex systems. When it comes to maintaining and evolving these systems, the lack of explicit static typing becomes a bottleneck. In response, researchers

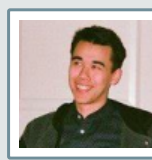
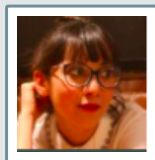
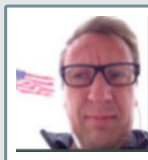
many cases, the systems start as innocent prototypes. Soon enough, though, they grow into complex, multi-module programs, at which point the engineers realize that they are facing a maintenance nightmare, mostly due to the lack of reliable type information.

Gradual typing [21, 26] proposes a language-based solution to this problem: *gradual static typing*. The idea is to extend





Thank You



Lessons

How to encourage **domain-specific** benchmarks?

Lessons

How to encourage **domain-specific** benchmarks?



Think like a practitioner

Lessons

How to encourage **domain-specific** benchmarks?



Think like a practitioner



Separate benchmarks from analysis tools

Lessons

How to encourage **domain-specific** benchmarks?



Think like a practitioner



Separate benchmarks from analysis tools



Borrow nodes

<https://github.com/utahplt/gtp-benchmarks>

<https://github.com/utahplt/gtp-measure>

<https://github.com/utahplt/gtp-plot>

