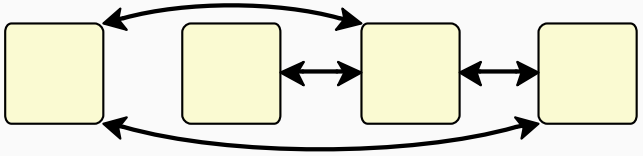
How to Evaluate the Performance of Gradual Type Systems

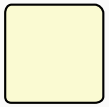
Ben Greenman
Asumu Takikawa
Max S. New
Daniel Feltey
Robert Bruce Findler
Jan Vitek
Matthias Felleisen

Contribution:

Our paper presents the first systematic **method** to measure the **performance implications** of a gradual typing system.

Notation

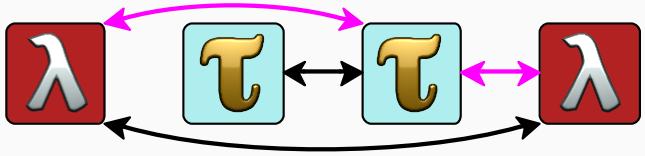
Program 

Component 

Dependency 

Gradual Typing

Mixed-Typed Program



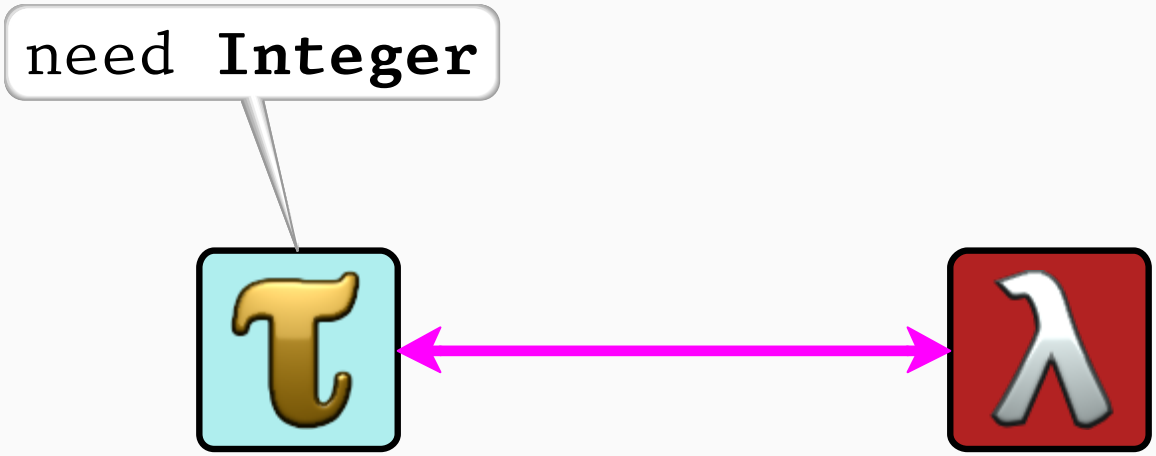
Statically-typed Component

Dynamically-typed Component

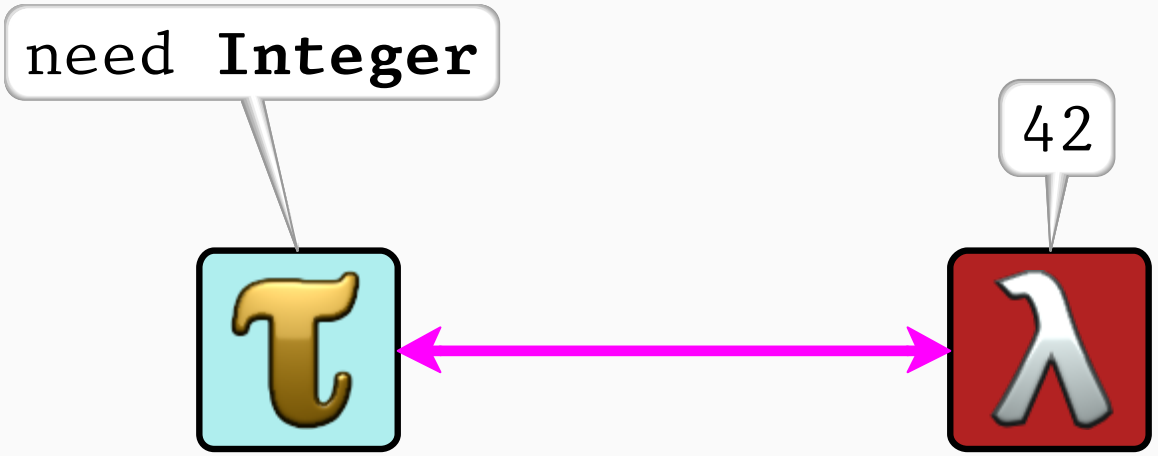
Type Boundary

Typed–Untyped Interaction

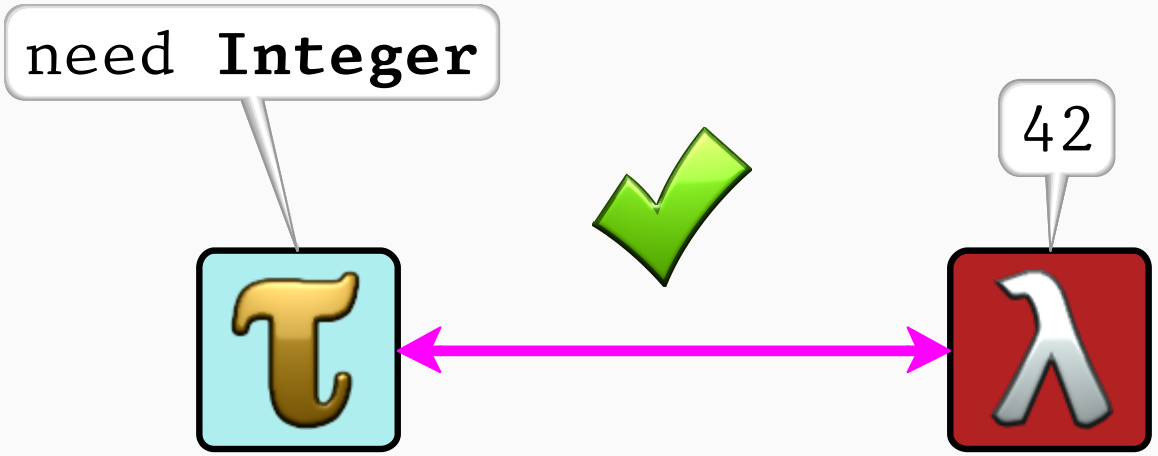
Typed–Untyped Interaction



Typed–Untyped Interaction

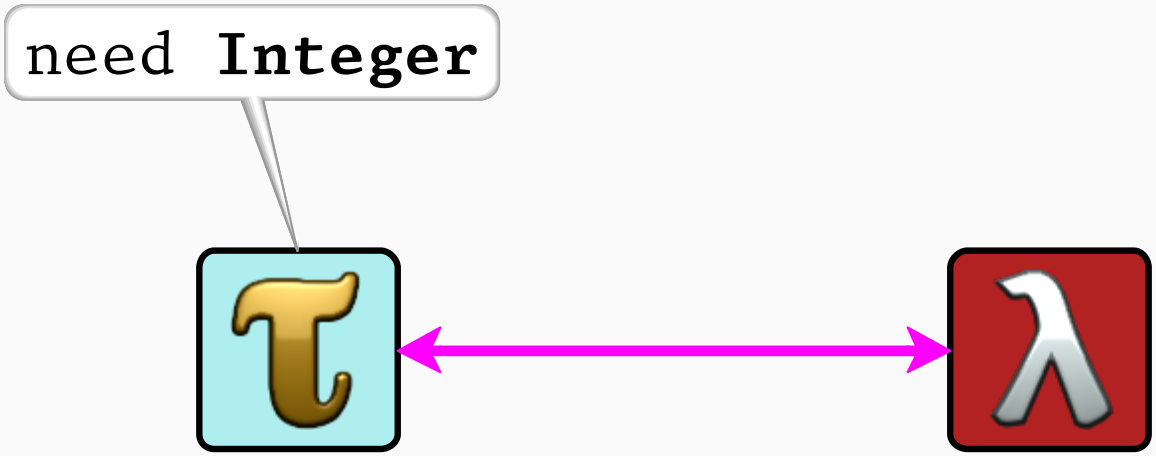


Typed–Untyped Interaction

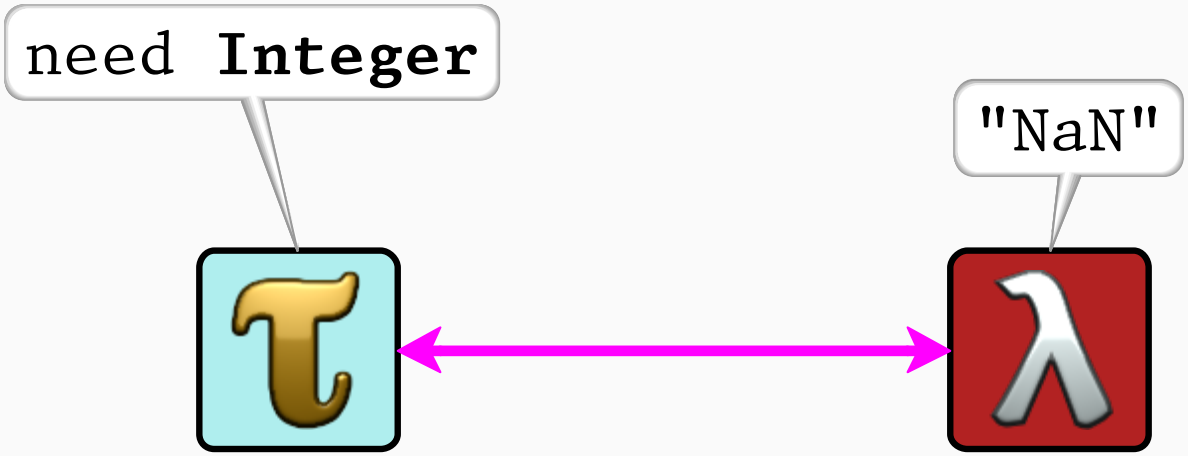


Typed–Untyped Interaction

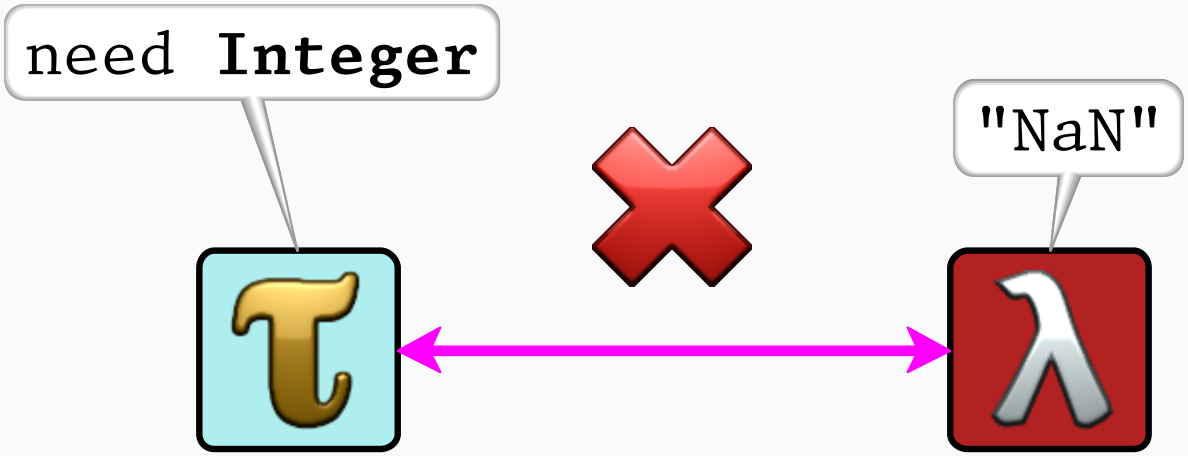
Typed–Untyped Interaction



Typed–Untyped Interaction

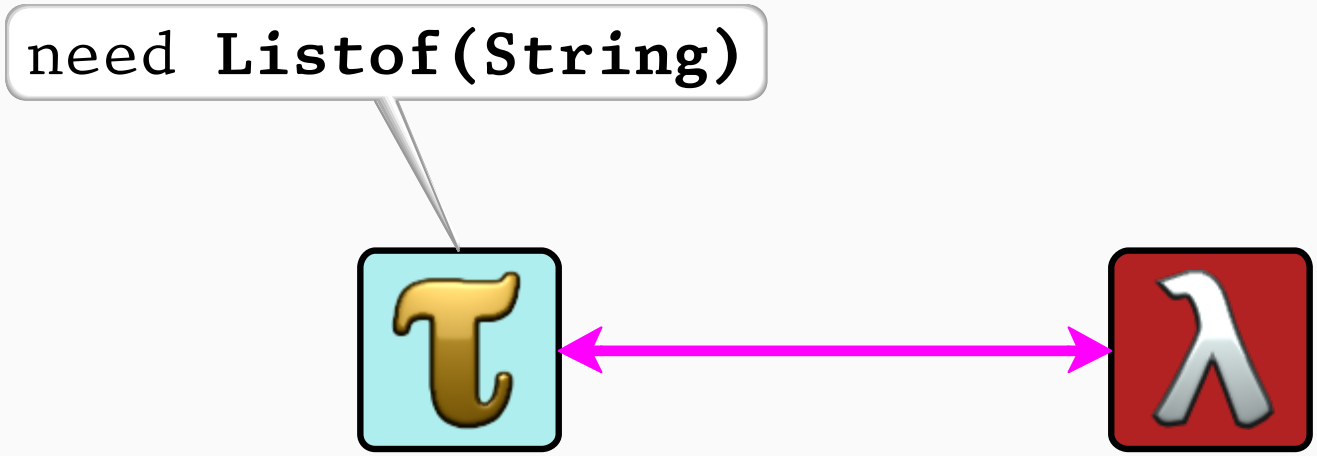


Typed–Untyped Interaction

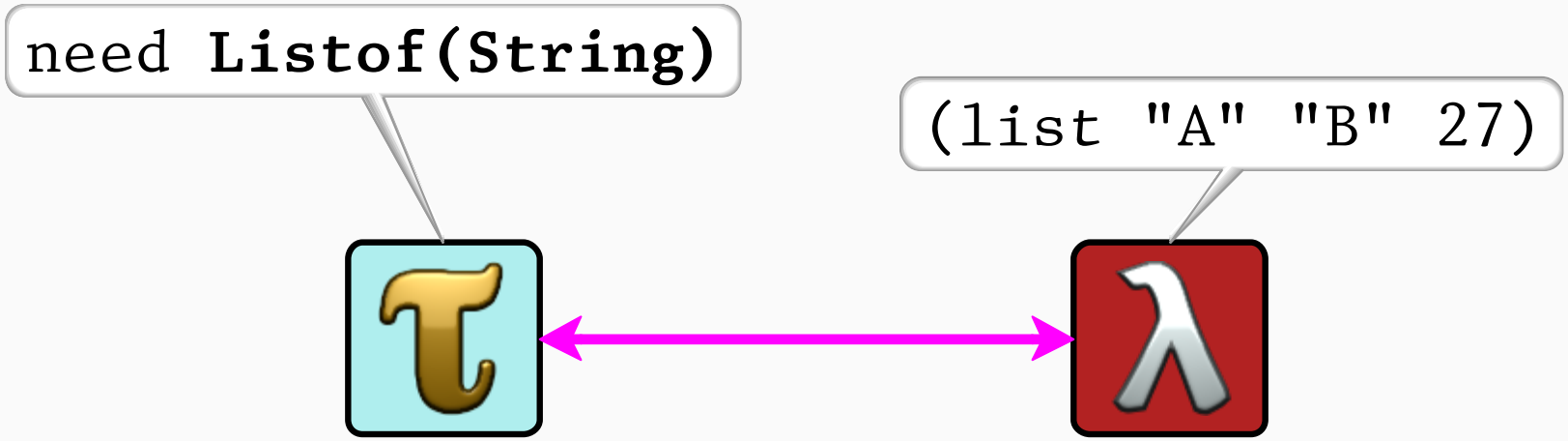


Typed–Untyped Interaction

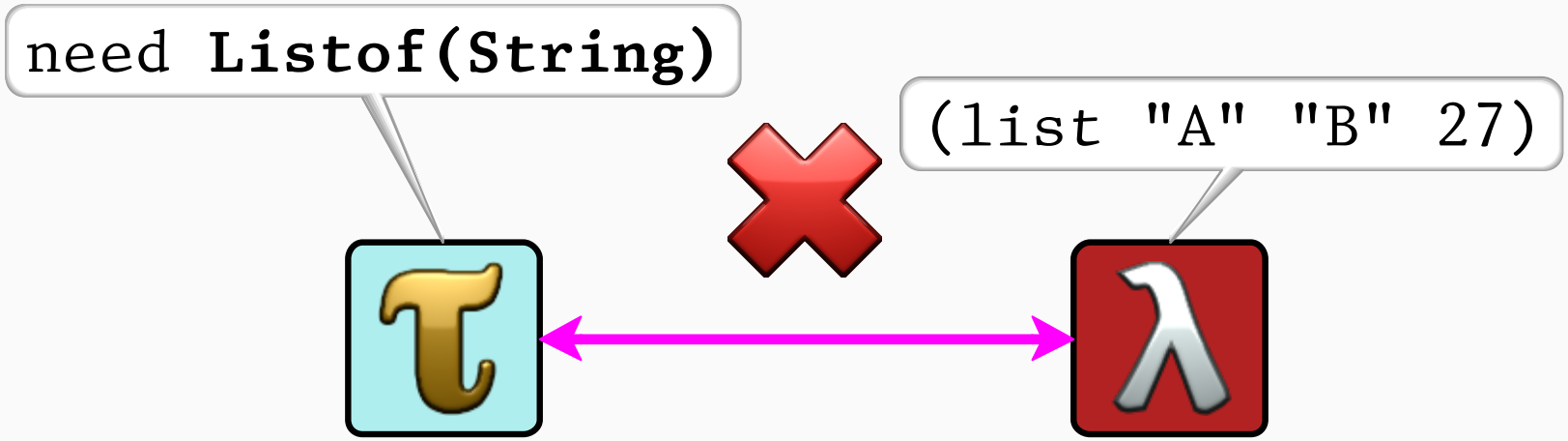
Typed-Untyped Interaction



Typed-Untyped Interaction

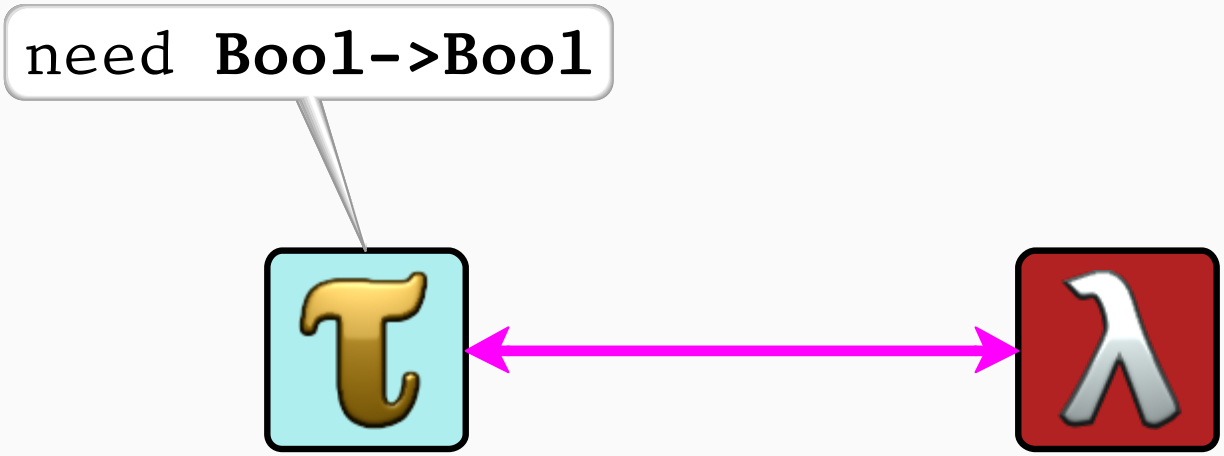


Typed-Untyped Interaction

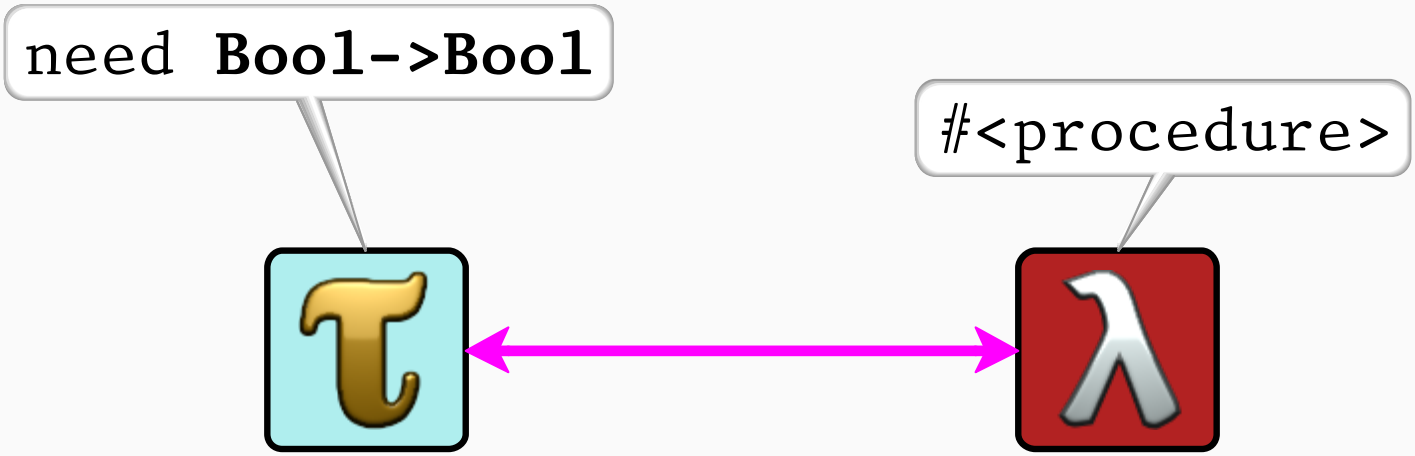


Typed–Untyped Interaction

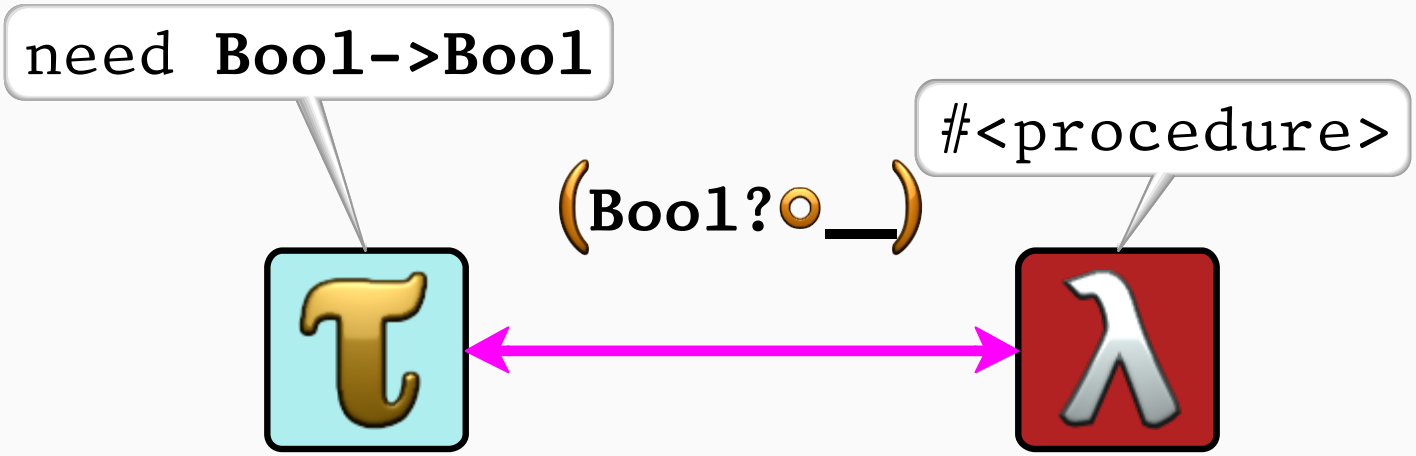
Typed–Untyped Interaction



Typed–Untyped Interaction



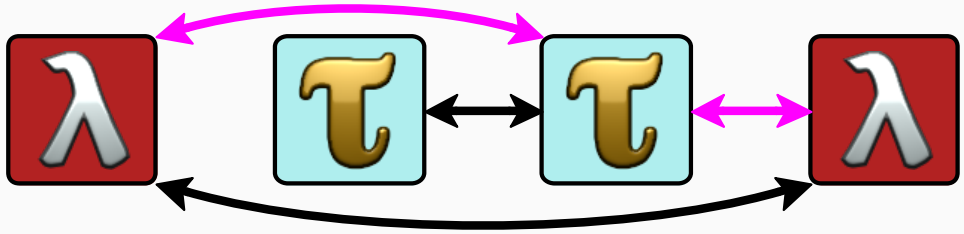
Typed–Untyped Interaction

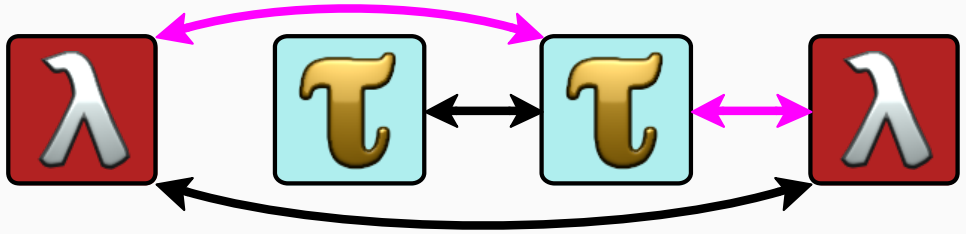


Typed-Untyped Interaction

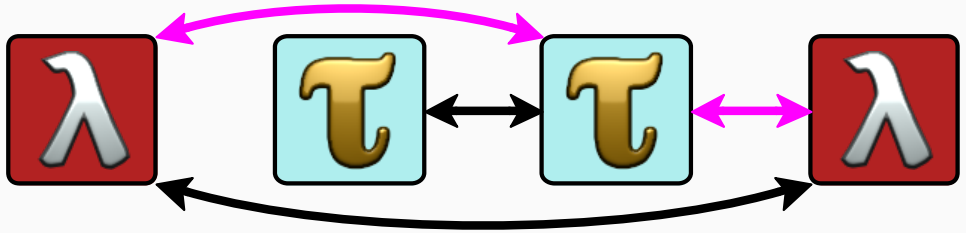
Type boundaries impose a run-time cost!

(Some mixed-typed languages do not enforce types. For these languages, the performance of type boundaries is not an issue.)


λ
τ
τ
λ



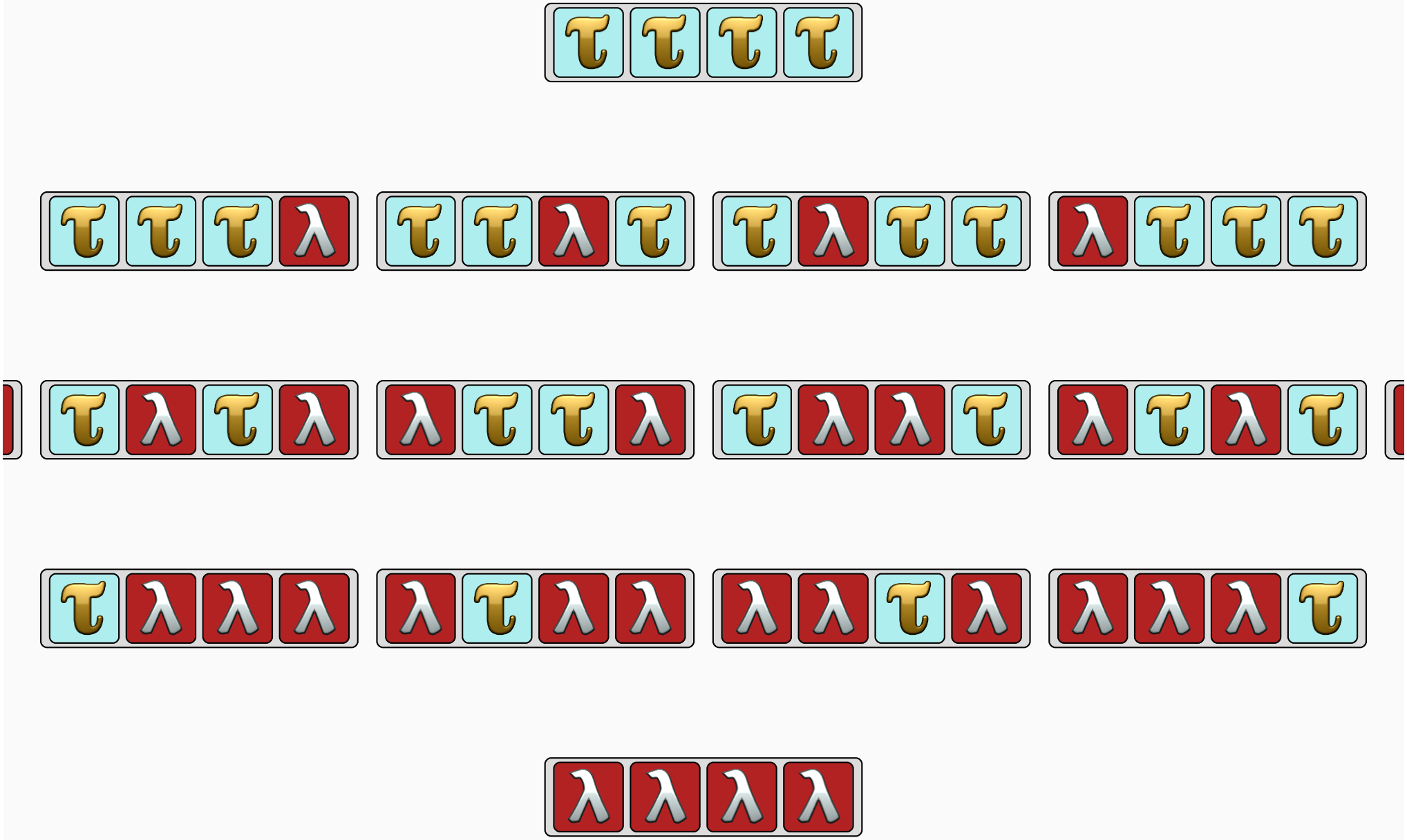
Q. What is the overall cost of boundaries in a **gradual typing system**?



Q. What is the overall cost of boundaries in a **gradual typing system**?

Need a **method** to measure and evaluate the performance implications of a gradual typing system

The Method


τ τ τ τ
τ τ τ λ
τ τ λ τ
τ λ τ τ
λ τ τ τ
τ λ τ λ
λ τ τ λ
τ λ λ τ
λ τ λ τ
τ λ λ λ
λ τ λ λ
λ λ τ λ
λ λ λ τ
λ λ λ λ

529 ms

602 ms

829,048 ms

821,285 ms

891 ms

829,779 ms

963 ms

575 ms

711,000 ms

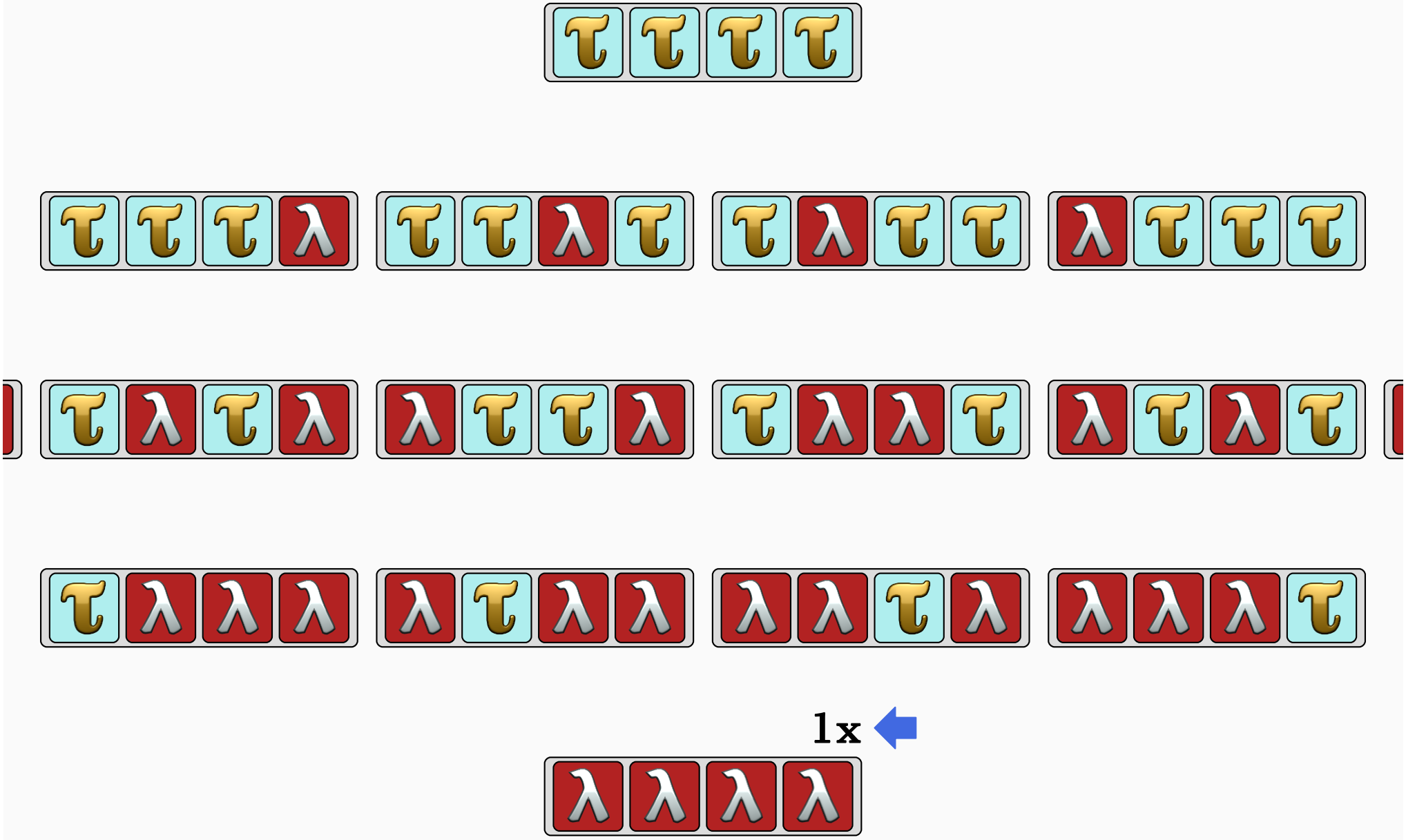
592 ms

709,770 ms

716,637 ms

560 ms

548 ms


1x

0.97x

1x

1,512x

1,498x

1x

x

1,513x

1x

1x

1,297x

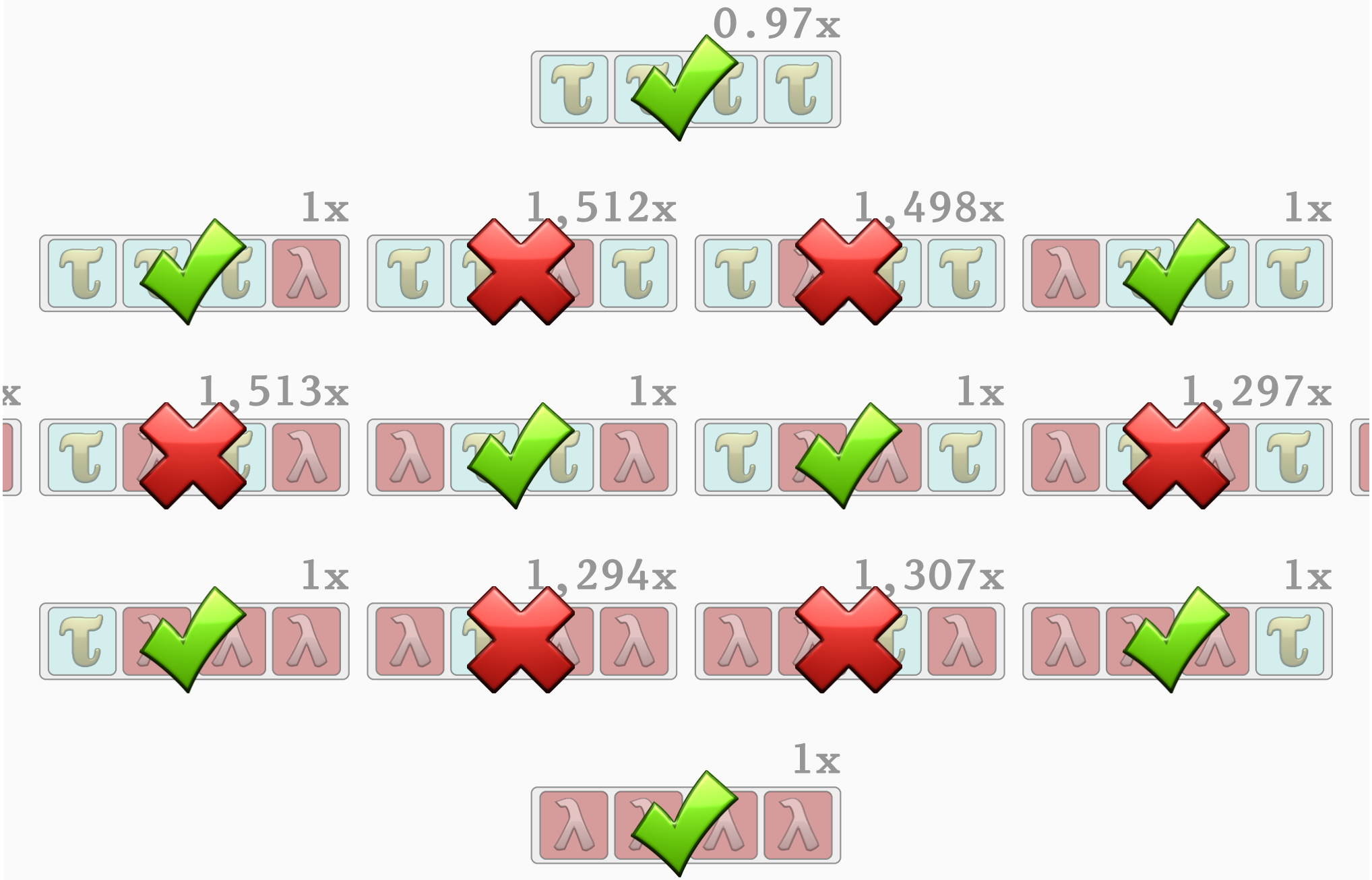
1x

1,294x

1,307x

1x

1x

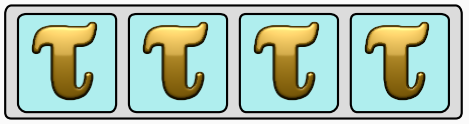

0.97x
1x
1,512x
1,498x
1x
x
1,513x
1x
1x
1,297x
1x
1,294x
1,307x
1x
1x

D-deliverable

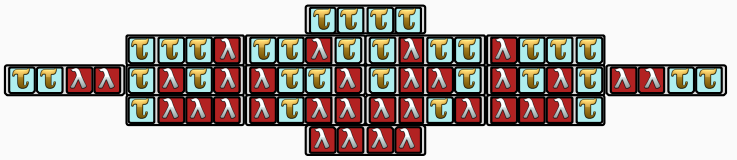
A configuration is **D**-*deliverable* if its performance is no worse than a factor of **D** slowdown compared to the baseline

Method: Exhaustive Perf. Eval.

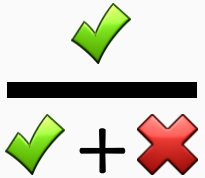
1. Typed program



2. Measure all configurations

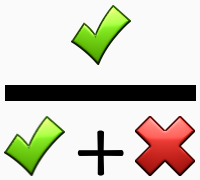


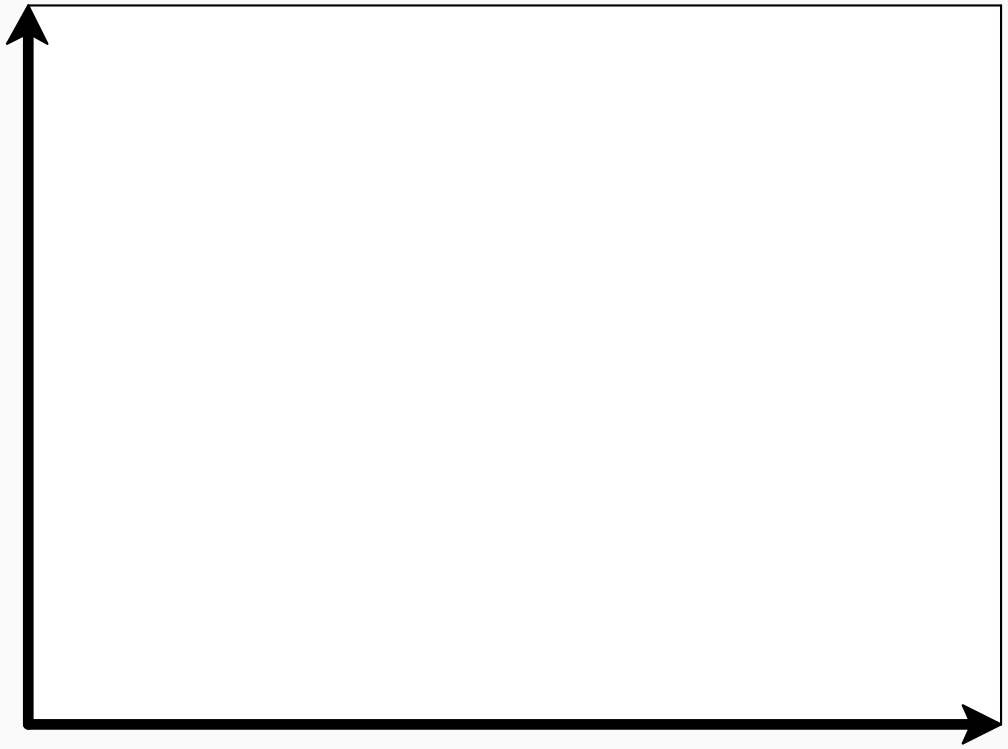
3. Count **D**-deliverable cfgs.



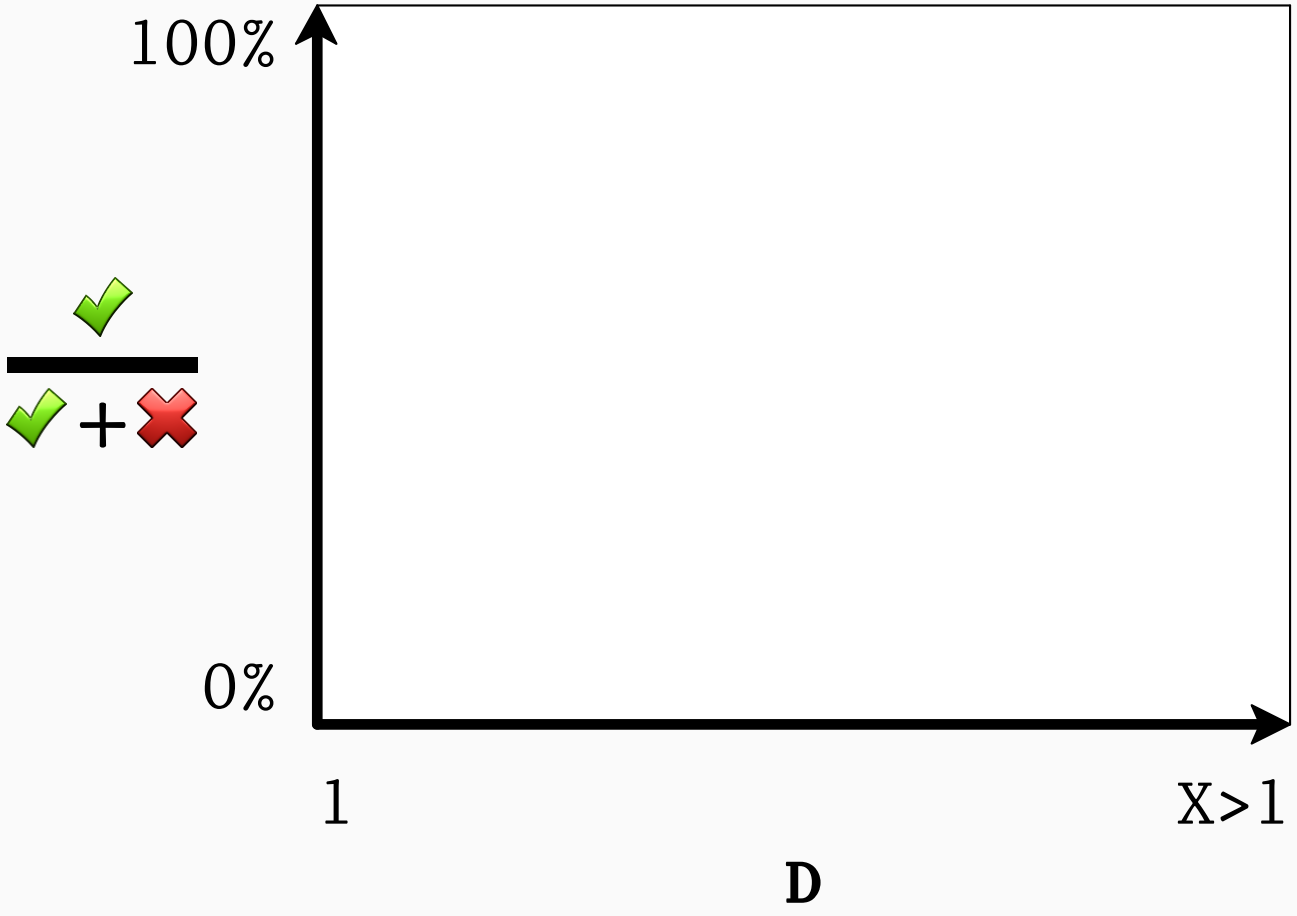
Repeat for other programs

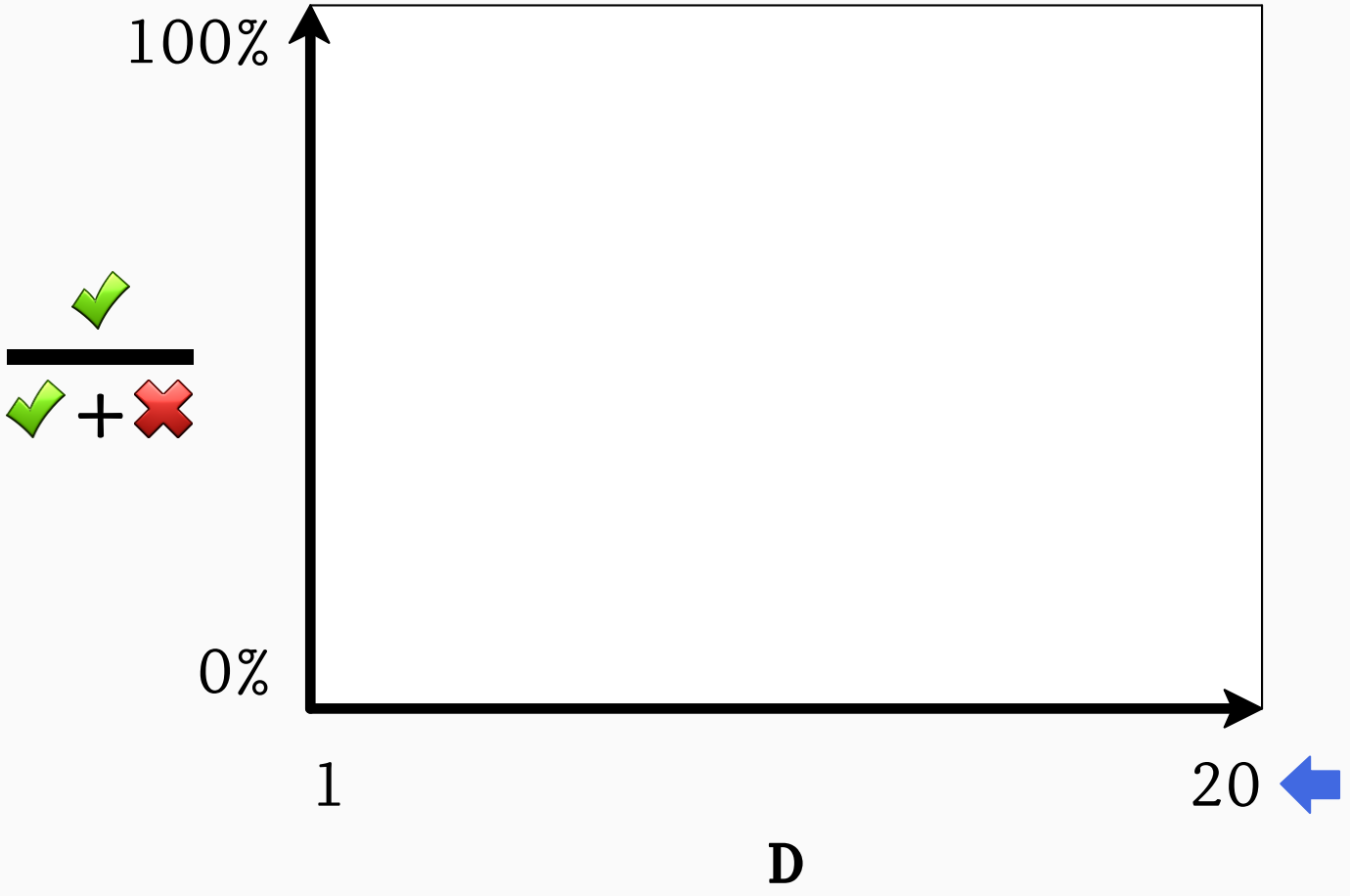
A Method for Presenting the Data

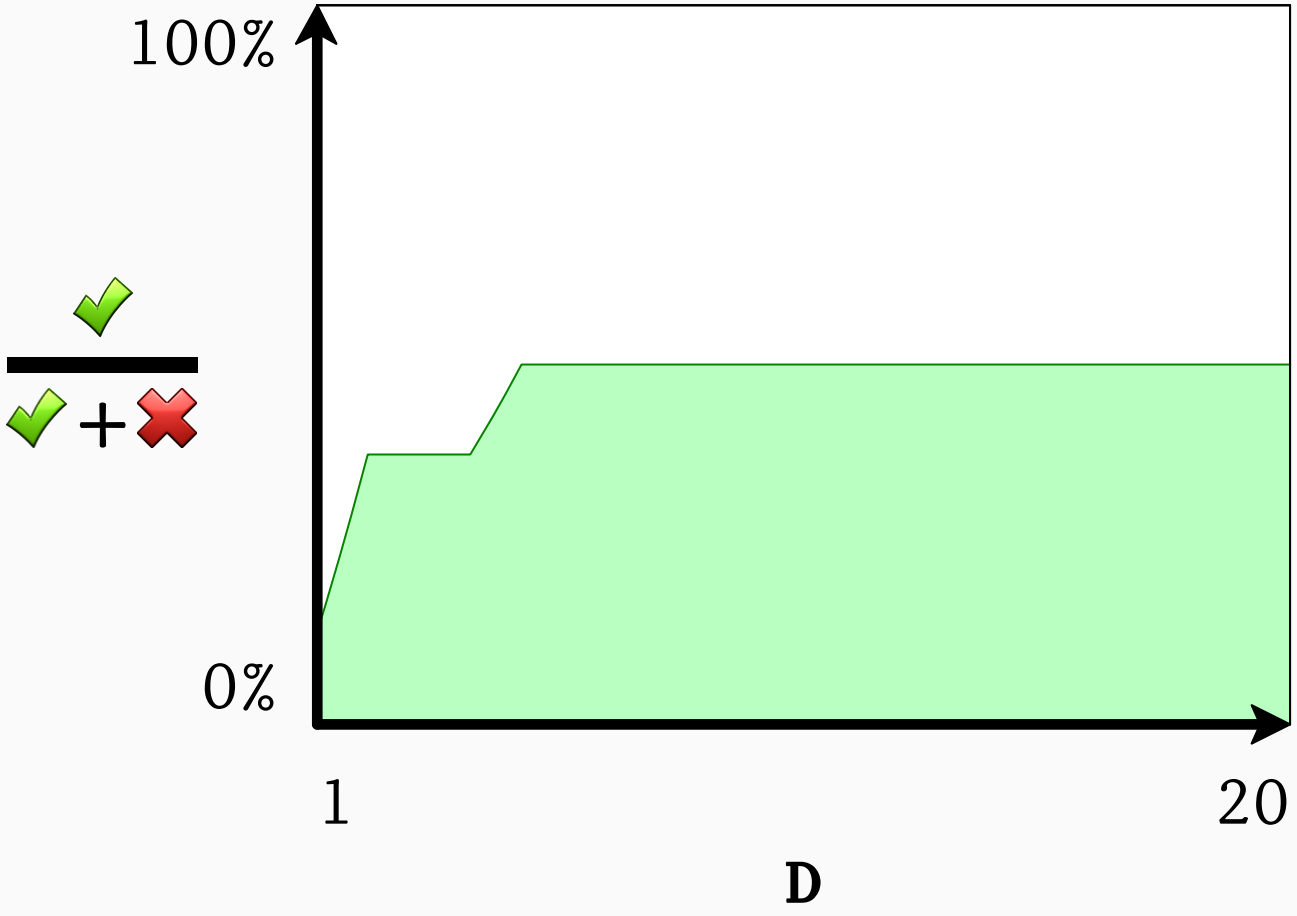




D


100%
0%
1
X>1
D

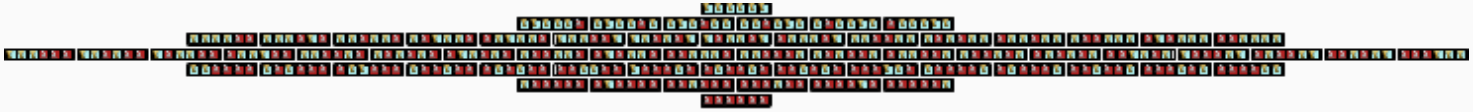

100%
0%
1
20
D


100%
0%
1
20
D

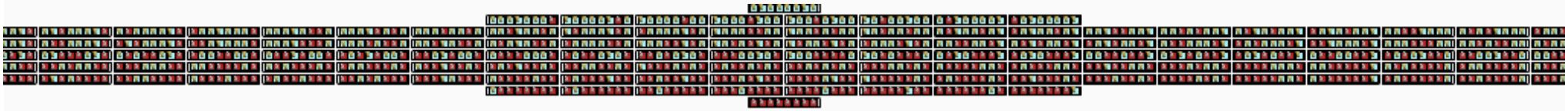
Scaling the Method

Exponential Blowup

4 components



6 components



8 components

Exponential Blowup

N components => 2^N configurations

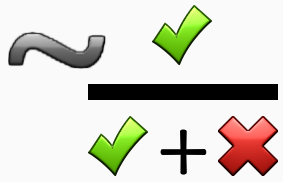
Simple Random Sampling

Simple Random Sampling

1. Sample O(N) configurations
   - N = number of components

Simple Random Sampling

1. Sample O(N) configurations
   - N = number of components
2. Count **D**-deliverable cfgs. in the sample



More in Paper

- justification for O(N) sampling
  - N = number of components
- exhaustive method applied to Typed Racket
  - 20 benchmarks, docs.racket-lang.org/gtp-benchmarks
- comparison: TR v6.2, v6.3, & v6.4
  - the method quantifies improvements
- discussion of pathologies

Thank you

Sam Tobin-Hochstadt

For Typed Racket, and for significant improvements to v6.3, v6.4, and beyond.

How to Evaluate the Performance of Gradual Type Systems

Ben Greenman *
Max S. New
Robert Bruce Findler
Matthias Felleisen
Asumu Takikawa
Daniel Feltey
Jan Vitek