Debugging Dynamic Language Features in a Multi-Tier Virtual Machine

Smoke and Mirror

Meetesh K Mehta

Advisor: Dr Manas Thakur

IIT Bombay

June 2, 2023

Dynamic Languages are **great**!

The **R** programming language, JS, lua...

Dynamic Languages are **great**!

The **R** programming language, JS, lua...

- Dynamic typing

Dynamic Languages are **great**!

The **R** programming language, JS, lua...

- Dynamic typing
- First-class functions/environments

Dynamic Languages are **great**!

The **R** programming language, JS, lua...

- Dynamic typing
- First-class functions/environments
- Lazy evaluation

Dynamic Languages are **great**!

The **R** programming language, JS, lua...

- Dynamic typing
- First-class functions/environments
- Lazy evaluation
- Runtime reification of environments

Dynamic Languages are **great**!

The **R** programming language, JS, lua...

- Dynamic typing
- First-class functions/environments
- Lazy evaluation
- Runtime reification of environments
- Access to runtime stack

Dynamic Languages are **great**!

The **R** programming language, JS, lua...

- Dynamic typing
- First-class functions/environments
- Lazy evaluation
- Runtime reification of environments
- Access to runtime stack
- Eval

Dynamic Languages are **great**!

The **R** programming language, JS, lua...

- Dynamic typing
- First-class functions/environments
- Lazy evaluation
- Runtime reification of environments
- Access to runtime stack
- Eval

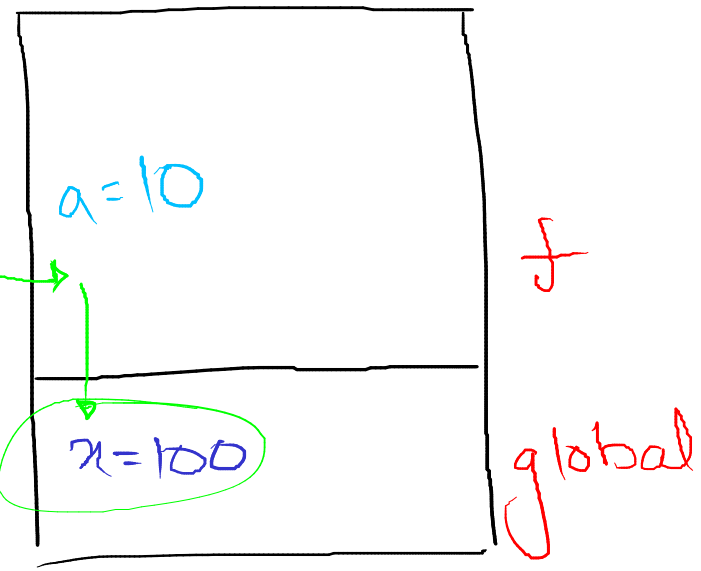
Slow and **complex** runtimes

The problem with optimizing dynamic languages

```
1   # Parent Scope
2   x <- 100
3   f <- function(a) {
4       a;
5       # Function Scope
6       print(x);
7   }
8
9
10
11
12
13
14
15
16
17
```

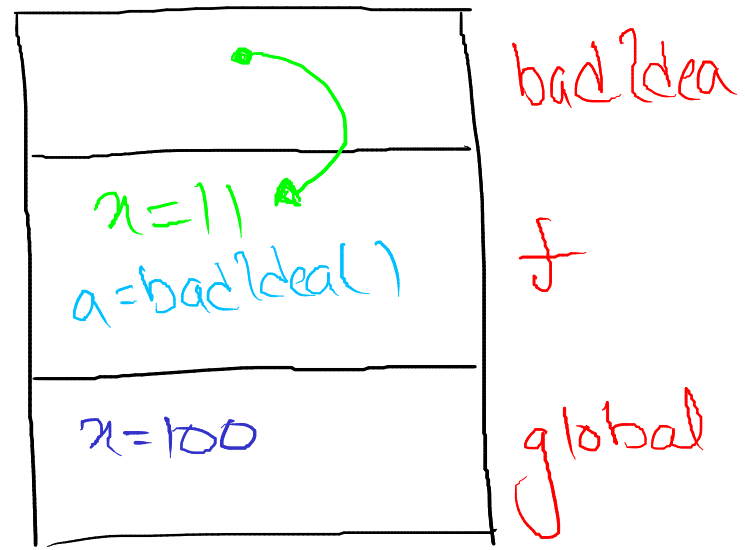
The problem with optimizing dynamic languages

```
1   # Parent Scope
2   x <- 100
3   f <- function(a) {
4       a;
5       # Function Scope
6       print(x);
7   }
8
9   f(10)
10
11  # Output: 100
12
13
14
15
16
17
```



The problem with optimizing dynamic languages

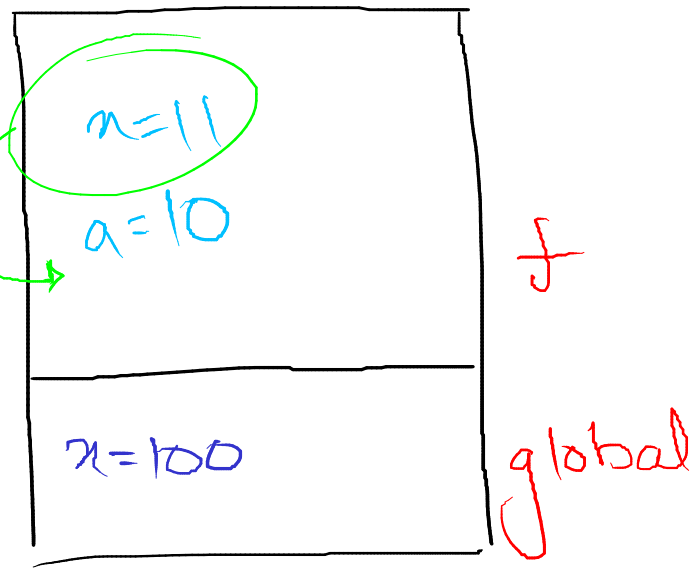
```r
# Parent Scope
x <- 100
f <- function(a) {
    a;
    # Function Scope
    print(x);
}
```




```r
badIdea <- function() { assign("x", 11, sys.frame(-1)); 1; }
f(badIdea())

# Output: 11
```

The problem with optimizing dynamic languages

```
1   # Parent Scope
2   x <- 100
3   f <- function(a) {
4       a;
5       # Function Scope
6       print(x);
7   }
8
9
10
11
12
13  badIdea <- function() { assign("x", 11, sys.frame(-1)); 1; }
14  f(badIdea())
15
16  # Output: 11
17
```



The $\check{\mathbf{R}}$ JIT compiler

Optimize for the most common cases.

- Runtime profiling
  - Collect information about **types**, **call sites**, **branches**.

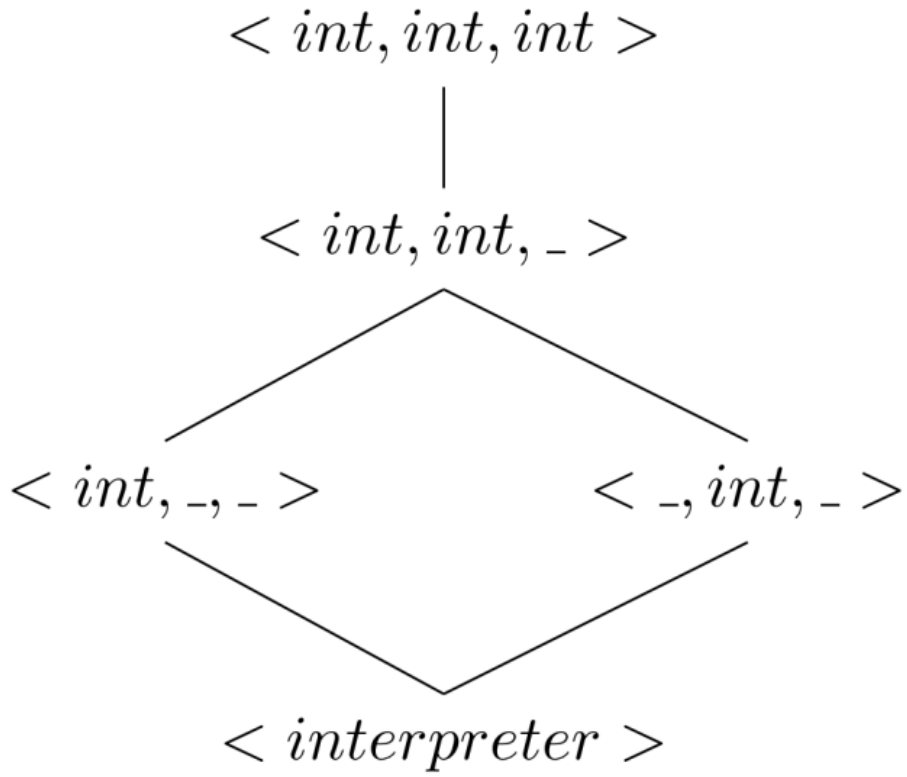
The Řr JIT compiler

Optimize for the most common cases.

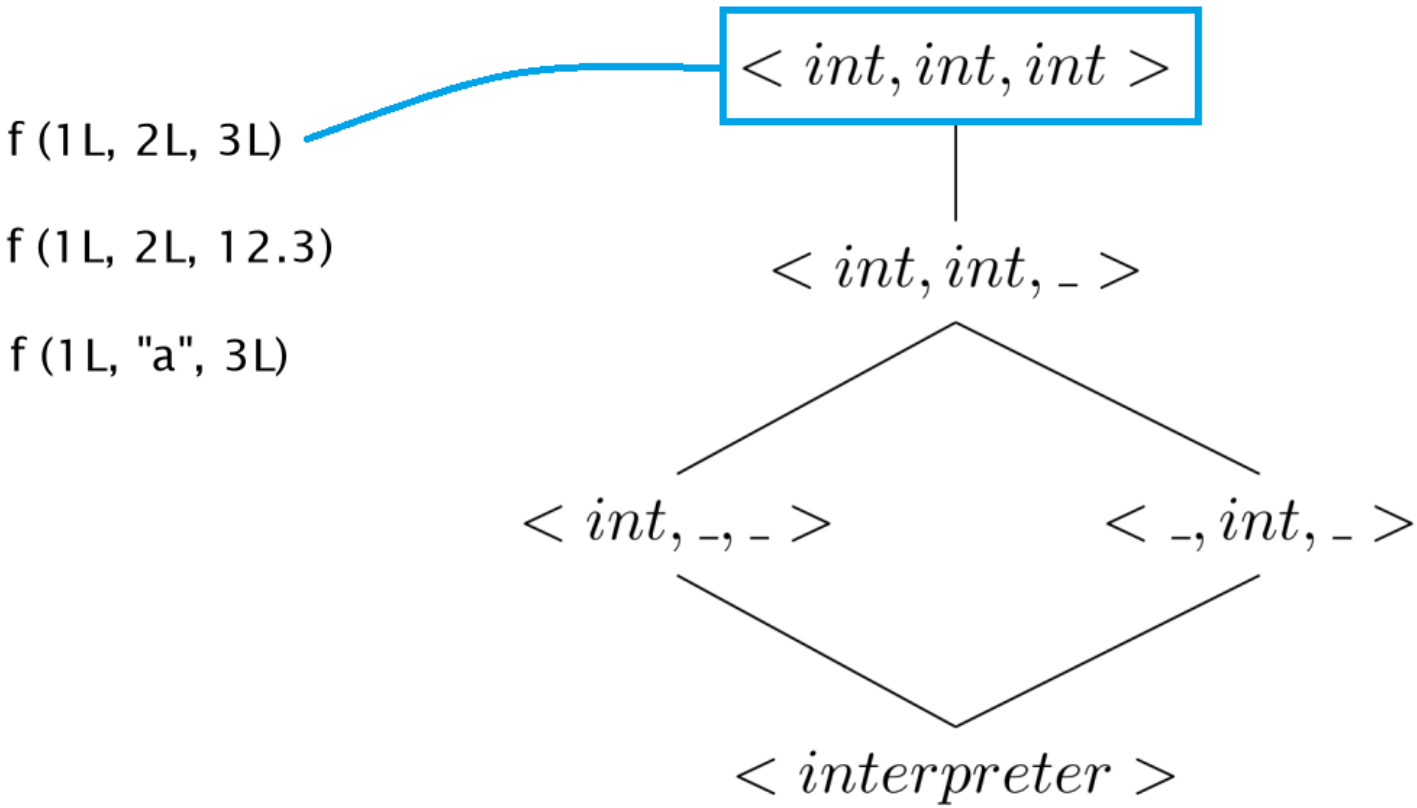
- Runtime profiling
  - Collect information about **types**, **call sites**, **branches**.
- Contextual Specialization
  - Optimize for **classes of behaviours**.

Contextual Dispatch

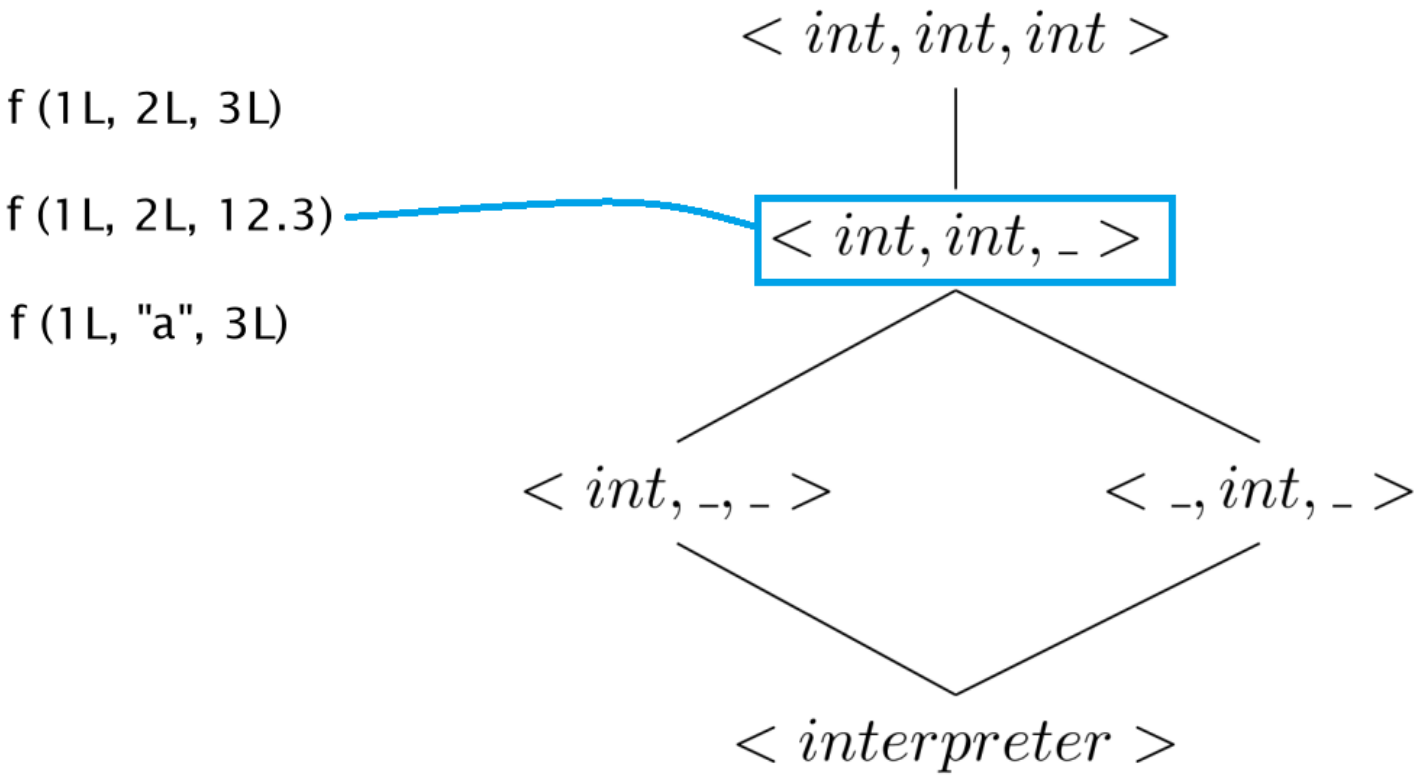
```
f (1L, 2L, 3L)

f (1L, 2L, 12.3)

f (1L, "a", 3L)
```



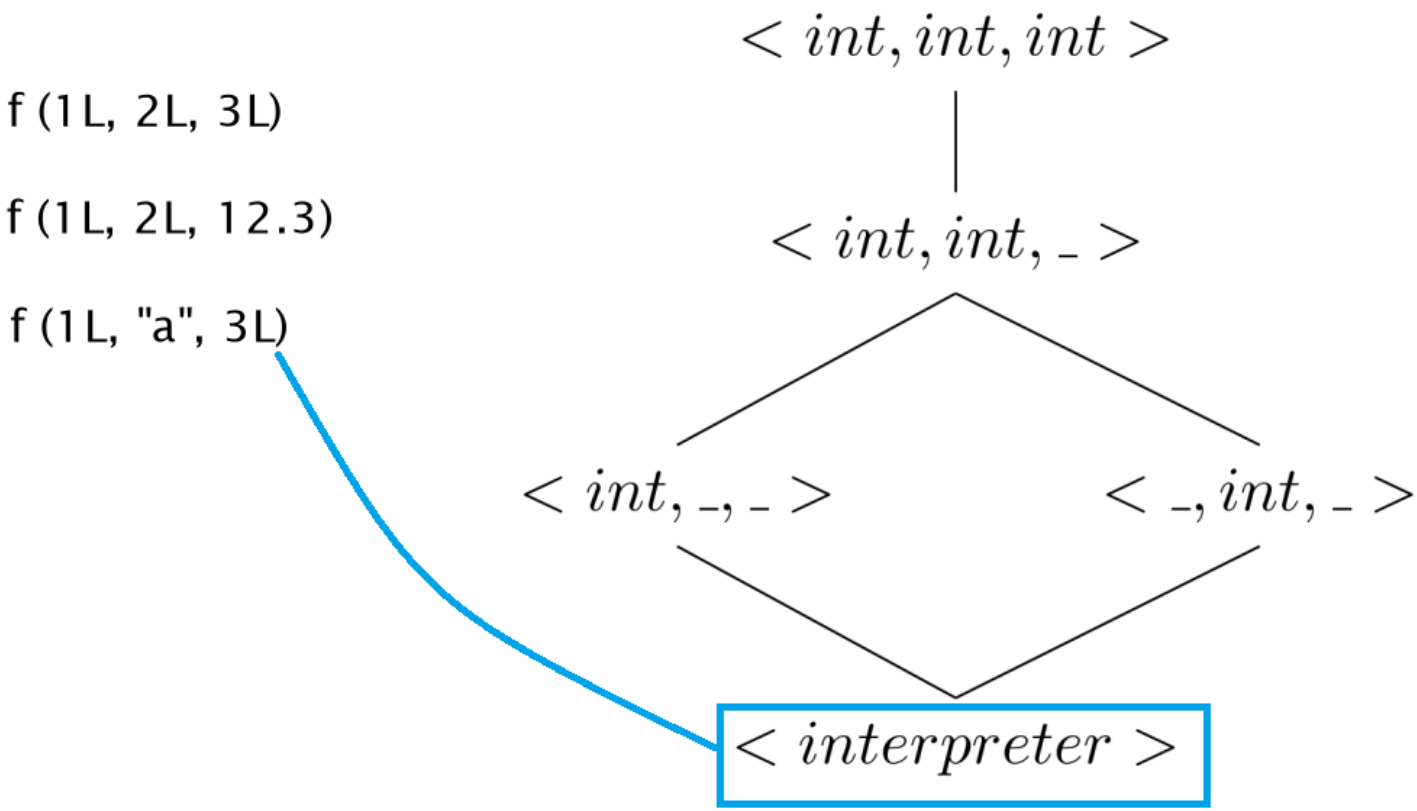
Contextual Dispatch



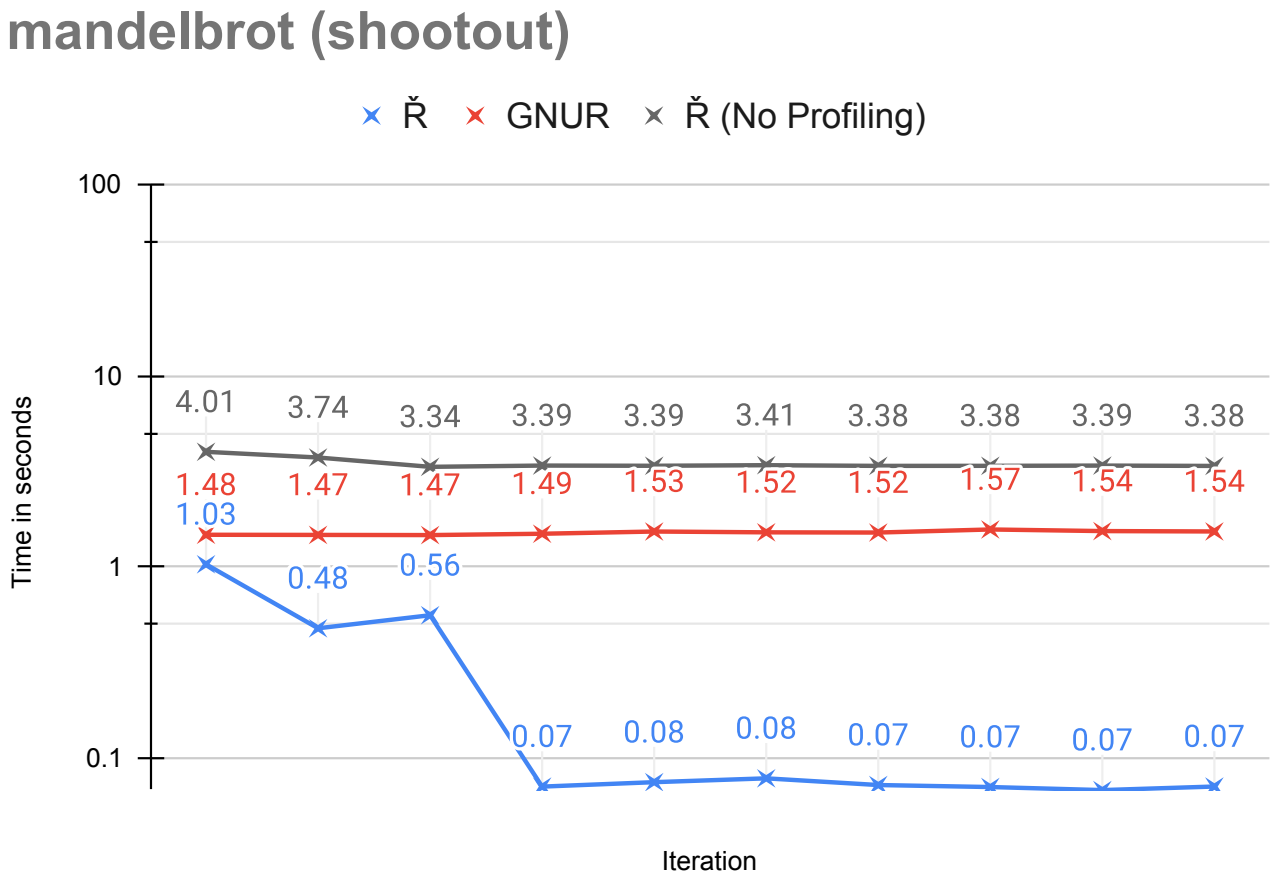
Contextual Dispatch



Contextual Dispatch



Performance



The regressions

reg-s4.R

- Normal GNUR: *$\tilde{0}$.35 sec*

The regressions

reg-s4.R

- Normal GNUR: *0̃.35 sec*
- Řˇ: **2̃36 sec**

The regressions

reg-s4.R

- Normal GNUR: *õ.35 sec*
- Řˇ: **õ236 sec**

About the program.

- 11 lines of R code.

The regressions

reg-s4.R

- Normal GNUR: *$\tilde{0}$.35 sec*
- Řˇ: **$\tilde{2}$36 sec**

About the program.

- 11 lines of R code.
- Nothing complex.

The regressions

reg-s4.R

- Normal GNUR: *$\tilde{0}$.35 sec*
- $\check{\mathrm{R}}$: **$\tilde{2}$36 sec**

About the program.

- 11 lines of R code.
- Nothing complex.
- Creates one class object.

The regressions

reg-s4.R

- Normal GNUR: *$\tilde{0}$.35 sec*
- Řˇ: **$\tilde{2}$36 sec**

About the program.

- 11 lines of R code.
- Nothing complex.
- Creates one class object.

Let the debugging begin.

- **Data Collection:** What tools to use? GDB, Valgrind, rr?

The regressions

reg-s4.R

- Normal GNUR: *$\tilde{0}$.35 sec*
- $\check{\mathrm{R}}$: **$\tilde{2}$36 sec**

About the program.

- 11 lines of R code.
- Nothing complex.
- Creates one class object.

Let the debugging begin.

- **Data Collection:** What tools to use? GDB, Valgrind, rr?
- **Hypothesis Testing:** Realistic insights, **solid evidence**.

The regressions

reg-s4.R

- Normal GNUR: *$\tilde{0}$.35 sec*
- Řℝ: **$\tilde{2}$36 sec**

About the program.

- 11 lines of R code.
- Nothing complex.
- Creates one class object.

Let the debugging begin.

- **Data Collection:** What tools to use? GDB, Valgrind, rr?
- **Hypothesis Testing:** Realistic insights, **solid evidence**.
- **Insights:** Real research begins **here**.

Usually the hammer works

Existing state of the art: fast and reliable

- Segfaults: GDB

Usually the hammer works

Existing state of the art: fast and reliable

- Segfaults: GDB
- Memory Leaks: Valgrind

Usually the hammer works

Existing state of the art: fast and reliable

- Segfaults: GDB
- Memory Leaks: Valgrind
- Wrong results: Compiler pass logs, rr

Usually the hammer works

Existing state of the art: fast and reliable

- Segfaults: GDB
- Memory Leaks: Valgrind
- Wrong results: Compiler pass logs, rr

The quick and dirty solutions are the best!

- **Trace visualizer:** $\check{\mathrm{R}}$-viz ($\tilde{2}$ weeks)

Usually the hammer works

Existing state of the art: fast and reliable

- Segfaults: GDB
- Memory Leaks: Valgrind
- Wrong results: Compiler pass logs, rr

The quick and dirty solutions are the best!

- **Trace visualizer:** $\check{\mathrm{R}}$-viz ($\tilde{2}$ weeks)
- **Runtime debuggers:** Rsh Dynamic Visualizer and Debugger (BTech MTP @ IITMandi)

Usually the hammer works

Existing state of the art: fast and reliable

- Segfaults: GDB
- Memory Leaks: Valgrind
- Wrong results: Compiler pass logs, rr

The quick and dirty solutions are the best!

- **Trace visualizer:** $\check{\mathrm{R}}$-viz ($\tilde{2}$ weeks)
- **Runtime debuggers:** Rsh Dynamic Visualizer and Debugger (BTech MTP @ IITMandi)
- **Event querying:** General Event Query Engine ($\tilde{2}$ days)

A little bit of **React**, **C++**, **JS** and some **sockets**.

Query: What creates so many contexts; are they even useful?

Hypothesis: Contextual dispatch is a bad idea for the real world!

Query: What creates so many contexts; are they even useful?

Hypothesis: Contextual dispatch is a bad idea for the real world!

$\check{\mathrm{R}}$-viz demo: https://compl-research.github.io/r-viz/

Query: What creates so many contexts; are they even useful?

Hypothesis: Contextual dispatch is a bad idea for the real world!

$\check{\mathrm{R}}$-viz demo: https://compl-research.github.io/r-viz/

Reality: Number of contexts are usually manageable, but there are too many compilations!

Query: What creates so many contexts; are they even useful?

Hypothesis: Contextual dispatch is a bad idea for the real world!

$\check{\mathrm{R}}$-viz demo: https://compl-research.github.io/r-viz/

Reality: Number of contexts are usually manageable, but there are too many compilations!

Insight: Compilation heuristic warrants deeper investigation.

Data Collection

Hypothesis: All compiled functions are equally responsible for peak performance.

Insight: Not always, sometimes inlining is dominant and most compilations are deprecated quickly.

Data Collection

Hypothesis: All compiled functions are equally responsible for peak performance.

> **Insight:** Not always, sometimes inlining is dominant and most compilations are deprecated quickly.

Hypothesis: Compilation heuristics are buggy, leading to unnecessary compilations.

> **Insight:** Yes, there is evidence to prove that.

Data Collection

Hypothesis: All compiled functions are equally responsible for peak performance.

> **Insight:** Not always, sometimes inlining is dominant and most compilations are deprecated quickly.

Hypothesis: Compilation heuristics are buggy, leading to unnecessary compilations.

> **Insight:** Yes, there is evidence to prove that.

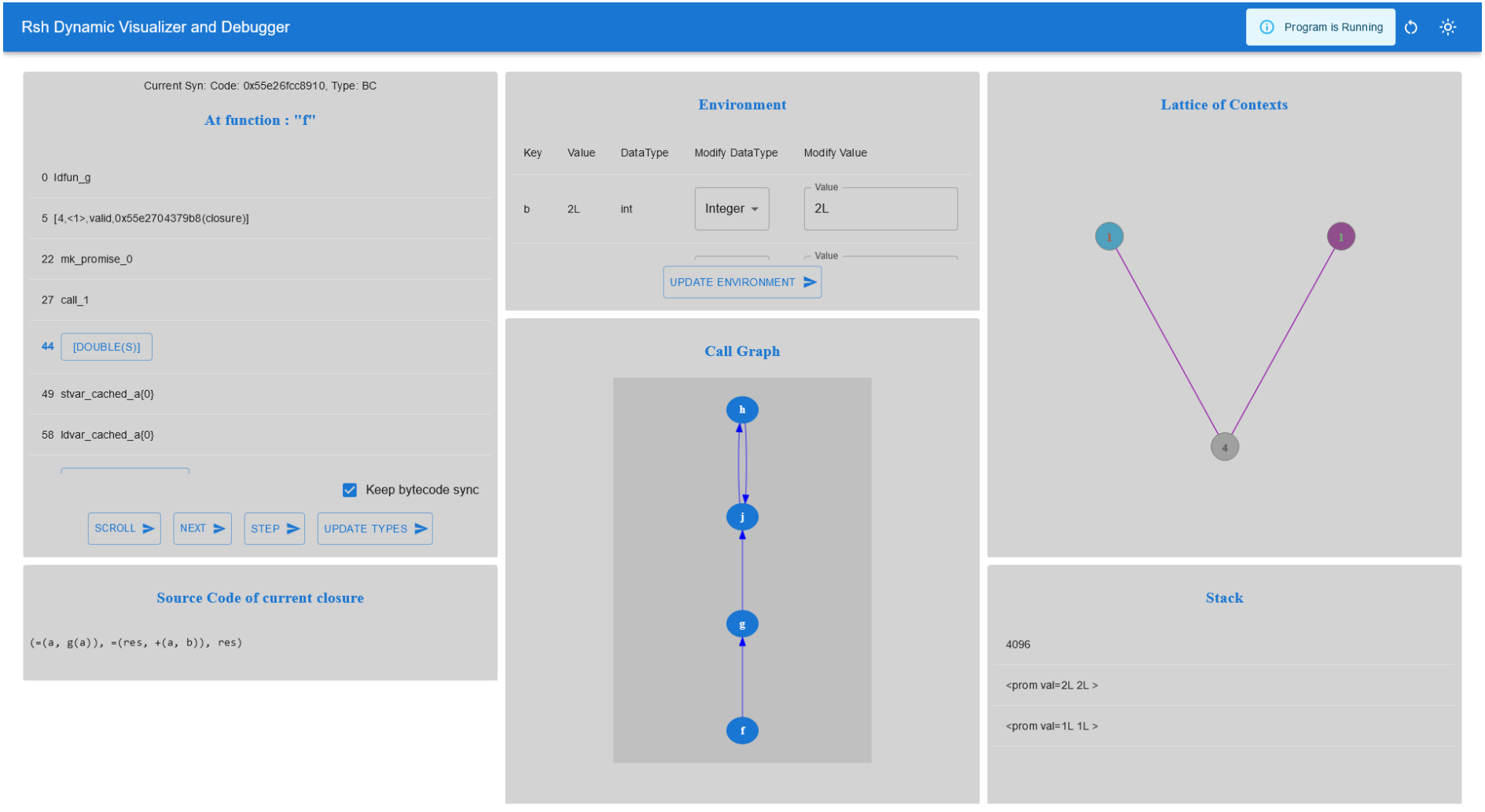
Hypothesis: Contextual compilation sounds good in theory, but in reality, only a generic context becomes dominant.

> **Insight:** No, in most cases the call-box shows that most contexts are used equally throught the runtime.

Runtime Debugger

Goal: Learn about the **internal working** of the system.

Team: Aayush Sharma and Anmolpreet Singh (**just-in**-graduation-**time**).



General Event Query Engine

Goal: A simple and fast query for runtime events.

General Event Query Engine

Goal: A simple and fast query for runtime events.

Motivation: Most things look right, but there is something suspicious.

General Event Query Engine

Goal: A simple and fast query for runtime events.

Motivation: Most things look right, but there is something suspicious.

GEQE: https://meetesh06.github.io/General-Event-Query-Engine/

Conclusion

The good

- Specialized tools are really useful in gaining meaningful insights.
- Mature frameworks like React are great way to write reusable code.

Conclusion

The good

- Specialized tools are really useful in gaining meaningful insights.
- Mature frameworks like React are great way to write reusable code.

The bad

- There are no one-size-fits-all solution for these problems.
- Things move fast and documentation is hard.

Conclusion

The good

- Specialized tools are really useful in gaining meaningful insights.
- Mature frameworks like React are great way to write reusable code.

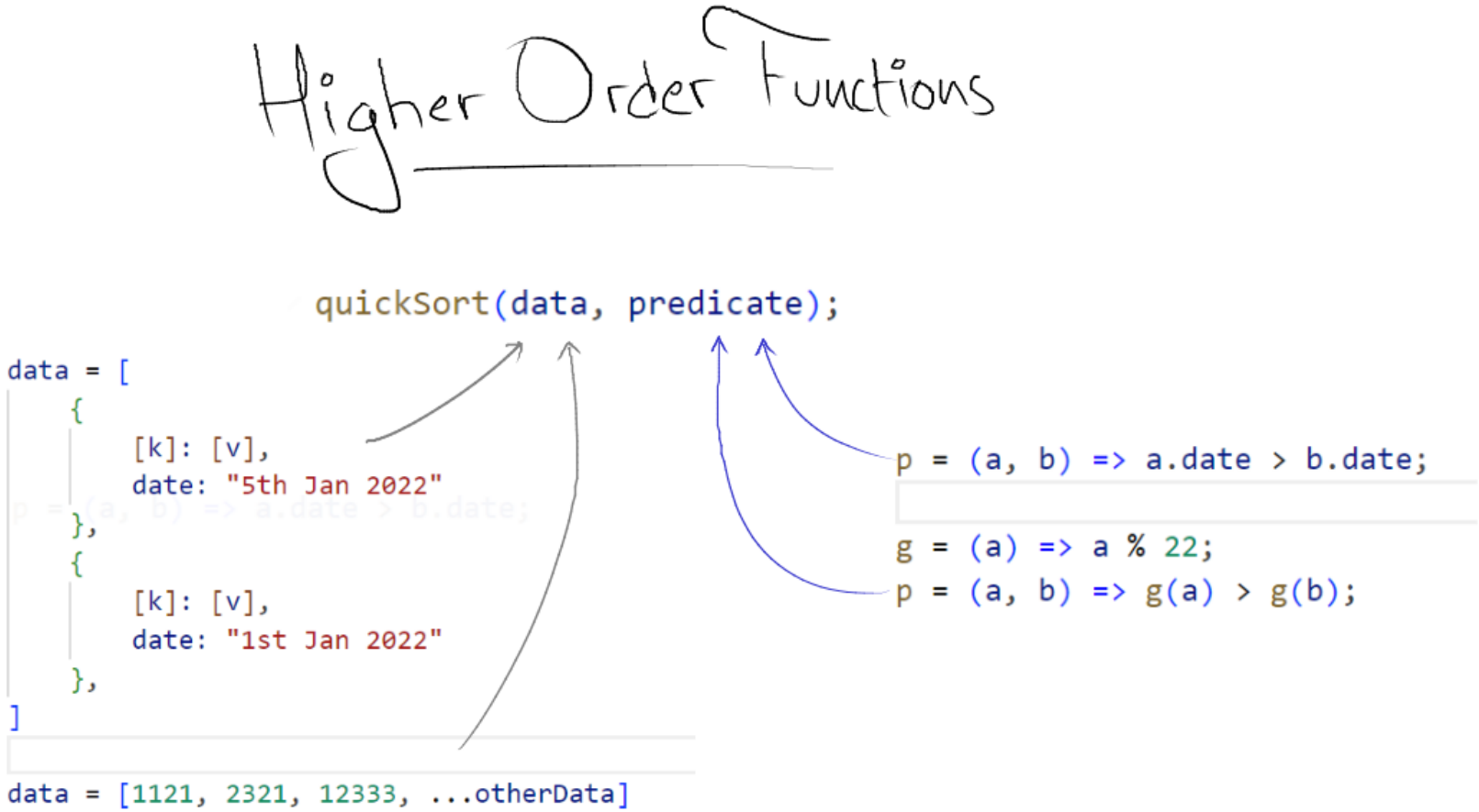
The bad

- There are no one-size-fits-all solution for these problems.
- Things move fast and documentation is hard.

Road ahead

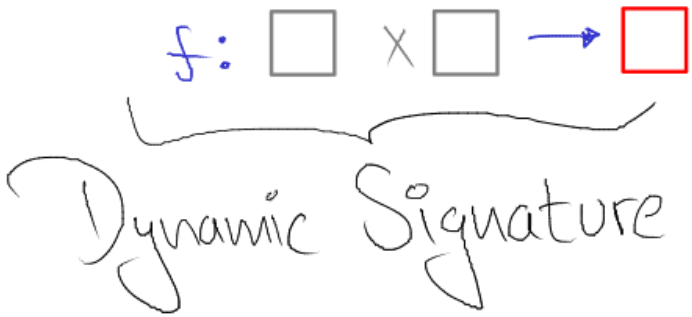
- These tools can be quickly retrofit to other use cases.

Dynamic languages are **great**!



Dynamic languages are **great**!


```
f <- function(a, b) {
    if (isS4(a, b)) {
        add.s4(a.rep, b.rep);
    } else if (isS3(a, b)){
        add.s3(a, b)
    } else {
        add.default(a,b)
    }
}
```



Dynamic languages are **great**!