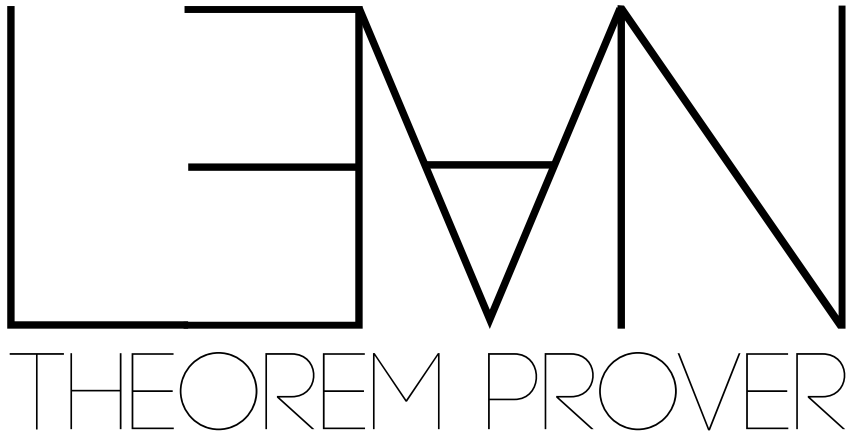
Karlsruhe Institute of Technology

# Lean 4: State of the U

**Sebastian Ullrich**

Programming paradigms group - IPD Snelting



# A brief history of Lean

- Lean 0.1 (2014)
- Lean 2 (2015)
  - first official release
  - fixed tactic language
- Lean 3 (2017)
  - make Lean a meta-programming language: build tactics in Lean
  - backed by a bytecode interpreter
- Lean 4 (201X)
  - make Lean a general-purpose language: native back end, FFI, ...
  - reimplement Lean in Lean

# A brief history of Lean

- Lean 0.1 (2014)
- Lean 2 (2015)
  - first official release
  - fixed tactic language
- Lean 3 (2017)
  - make Lean a meta-programming language: build tactics in Lean
  - backed by a bytecode interpreter
- Lean 4 (201X)
  - make Lean a general-purpose language: native back end, FFI, ...
  - reimplement Lean in Lean

$$X \geq 10$$

# The Lean dream team

Karlsruhe Institute of Technology

- Leonardo de Moura: everything, really
- Sebastian Ullrich: macros, interpreter

# The Lean dream team

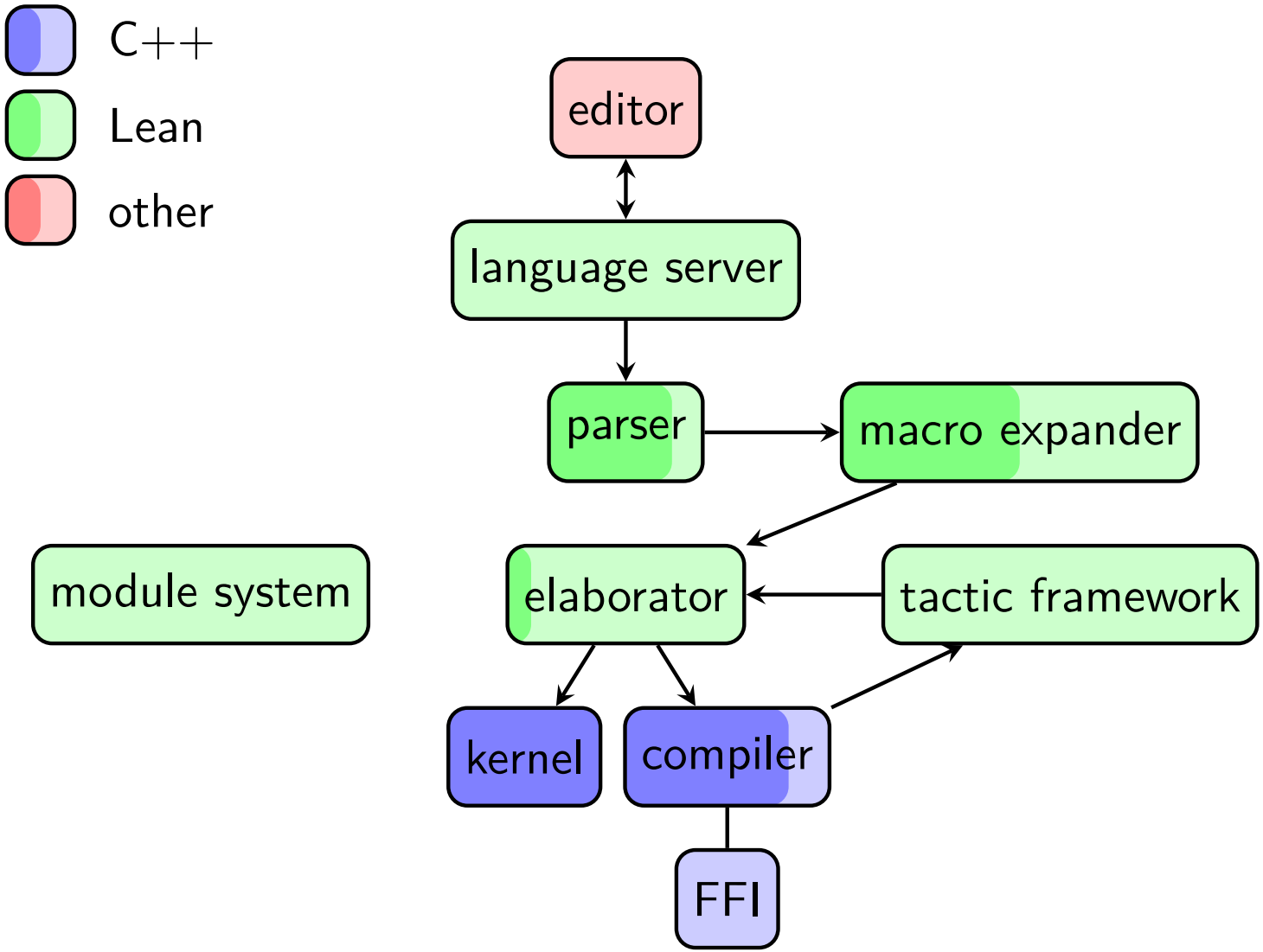
Karlsruhe Institute of Technology

- Leonardo de Moura: everything, really
- Sebastian Ullrich: macros, interpreter
- Daniel Selsam: new typeclass resolution

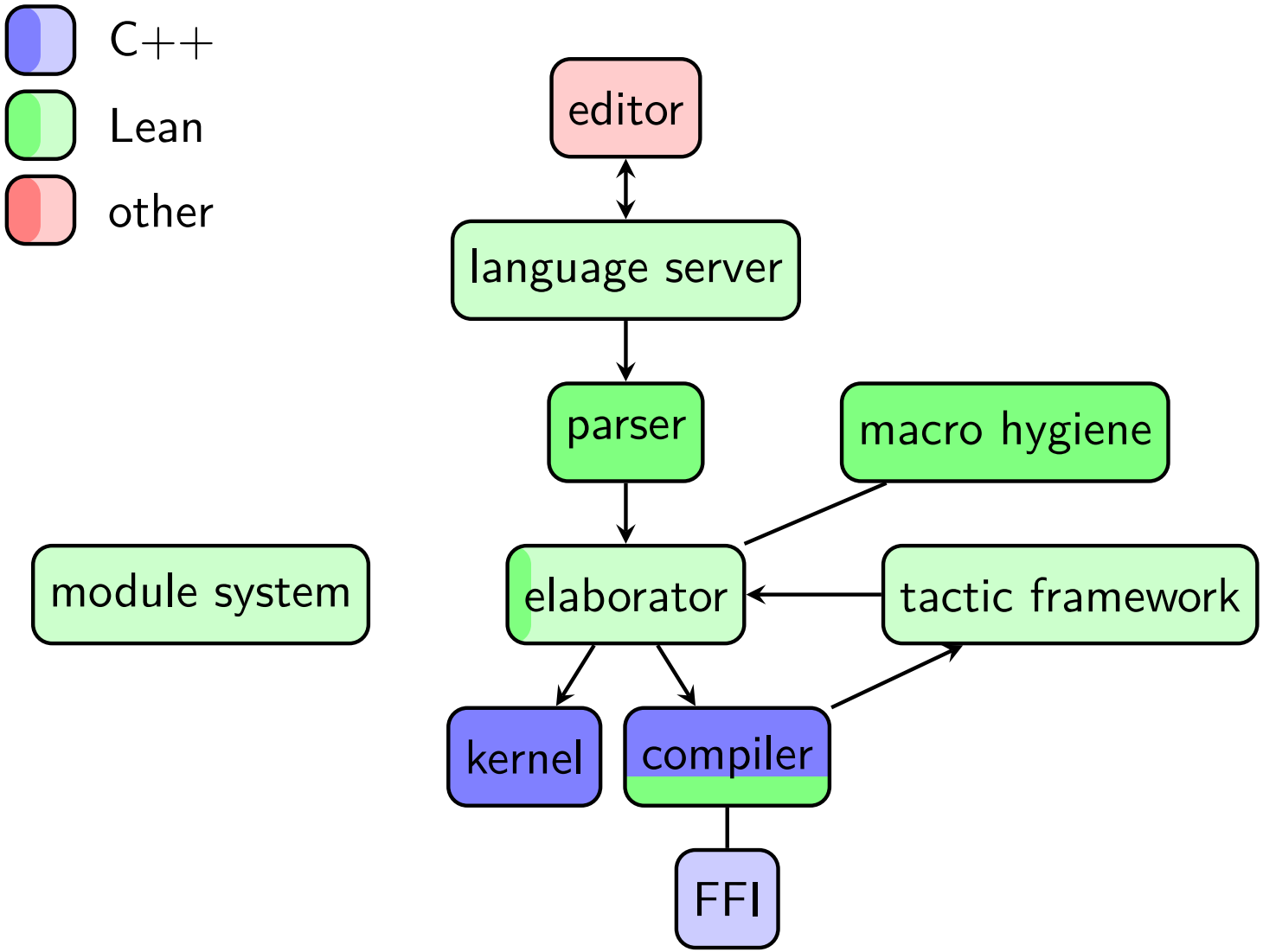
# The Lean dream team

- Leonardo de Moura: everything, really
- Sebastian Ullrich: macros, interpreter
- Daniel Selsam: new typeclass resolution
- Simon Hudon: language server

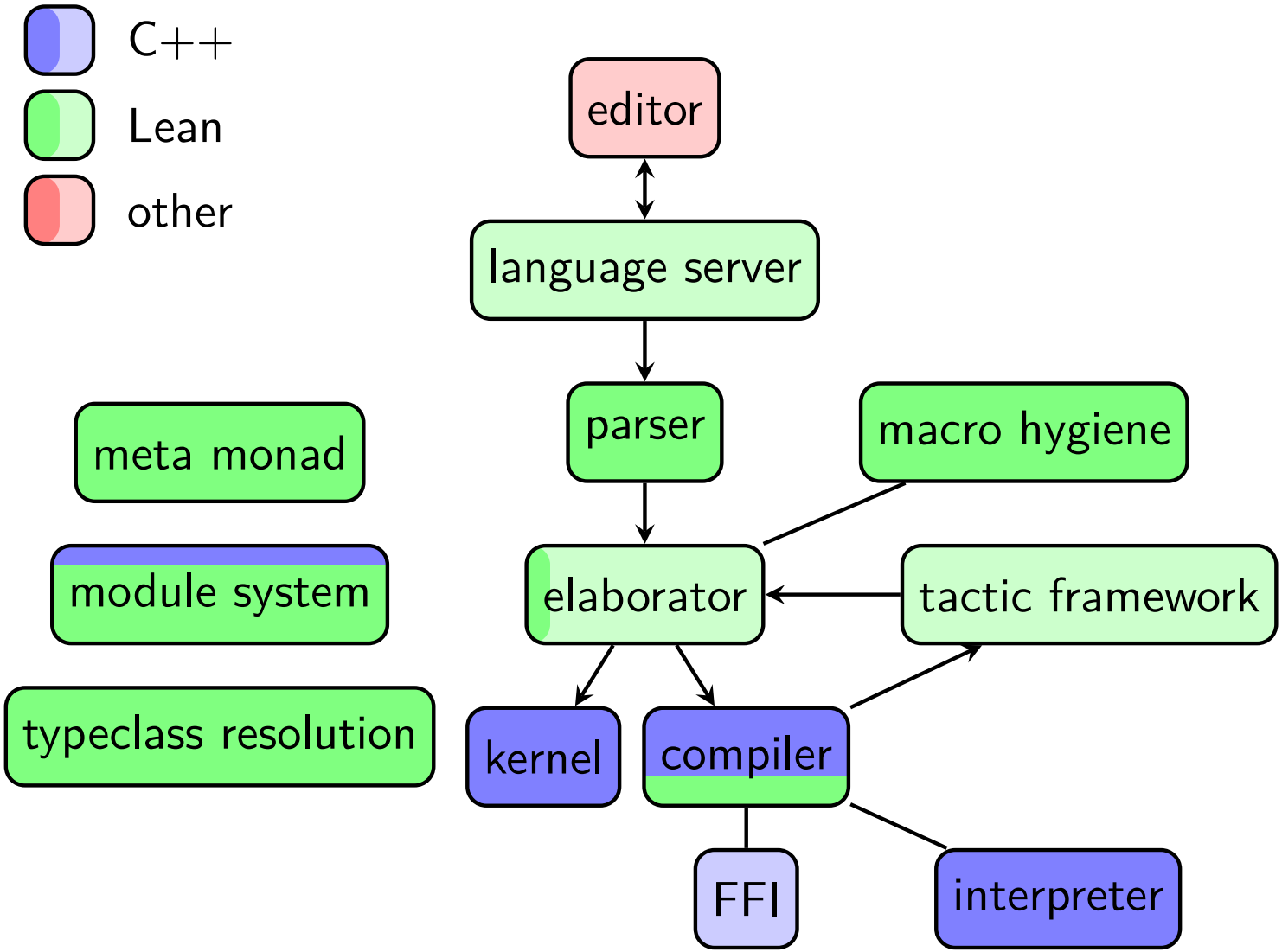
# Lean 4 progress: Jan 2019



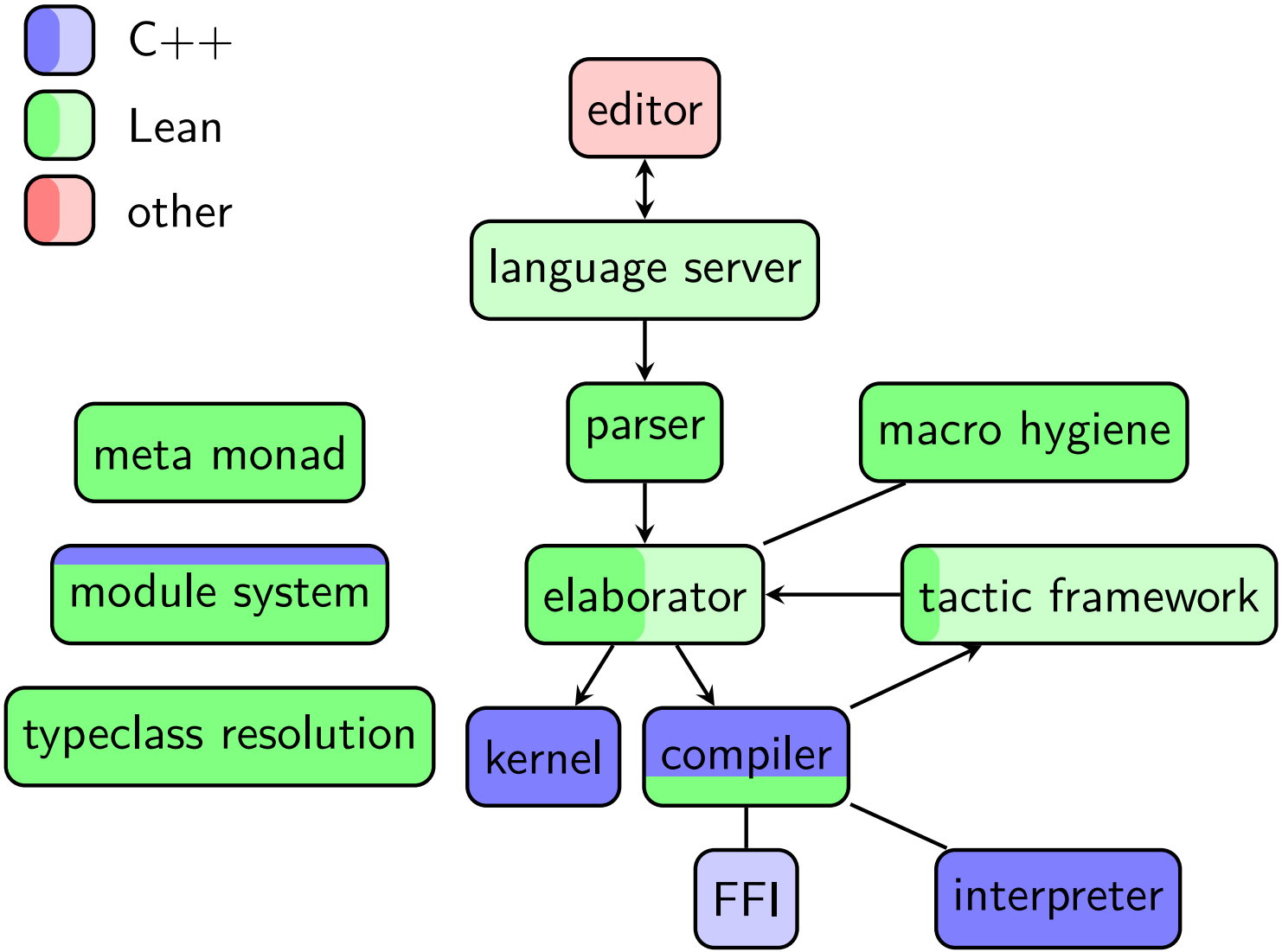
# Lean 4 progress: Jun 2019



# Lean 4 progress: Dec 2019



# Lean 4 progress: Jan 2020



# Cosmetics

Karlsruhe Institute of Technology

Minor syntax changes to make Lean a more consistent and pleasant language

- naming convention: `TypeName`, `ModuleName`, `termName`

# Cosmetics

Karlsruhe Institute of Technology

Minor syntax changes to make Lean a more consistent and pleasant language

- naming convention: `TypeName`, `ModuleName`, `termName`
  - lemma convention `termName_property_of_assumption`?

# Cosmetics

Minor syntax changes to make Lean a more consistent and pleasant language

- naming convention: `TypeName`, `ModuleName`, `termName`
  - lemma convention `termName_property_of_assumption`?
- consistent pattern syntax

```
def hiThere : ...
| pat1, ... => ...
| ...
```

```
match ... with
| pat1, ... => ...
| ...
```

# Cosmetics

Minor syntax changes to make Lean a more consistent and pleasant language

- naming convention: `TypeName`, `ModuleName`, `termName`
  - lemma convention `termName_property_of_assumption`?
- consistent pattern syntax

```
def hiThere : ...
| pat1, ... => ...
| ...
```

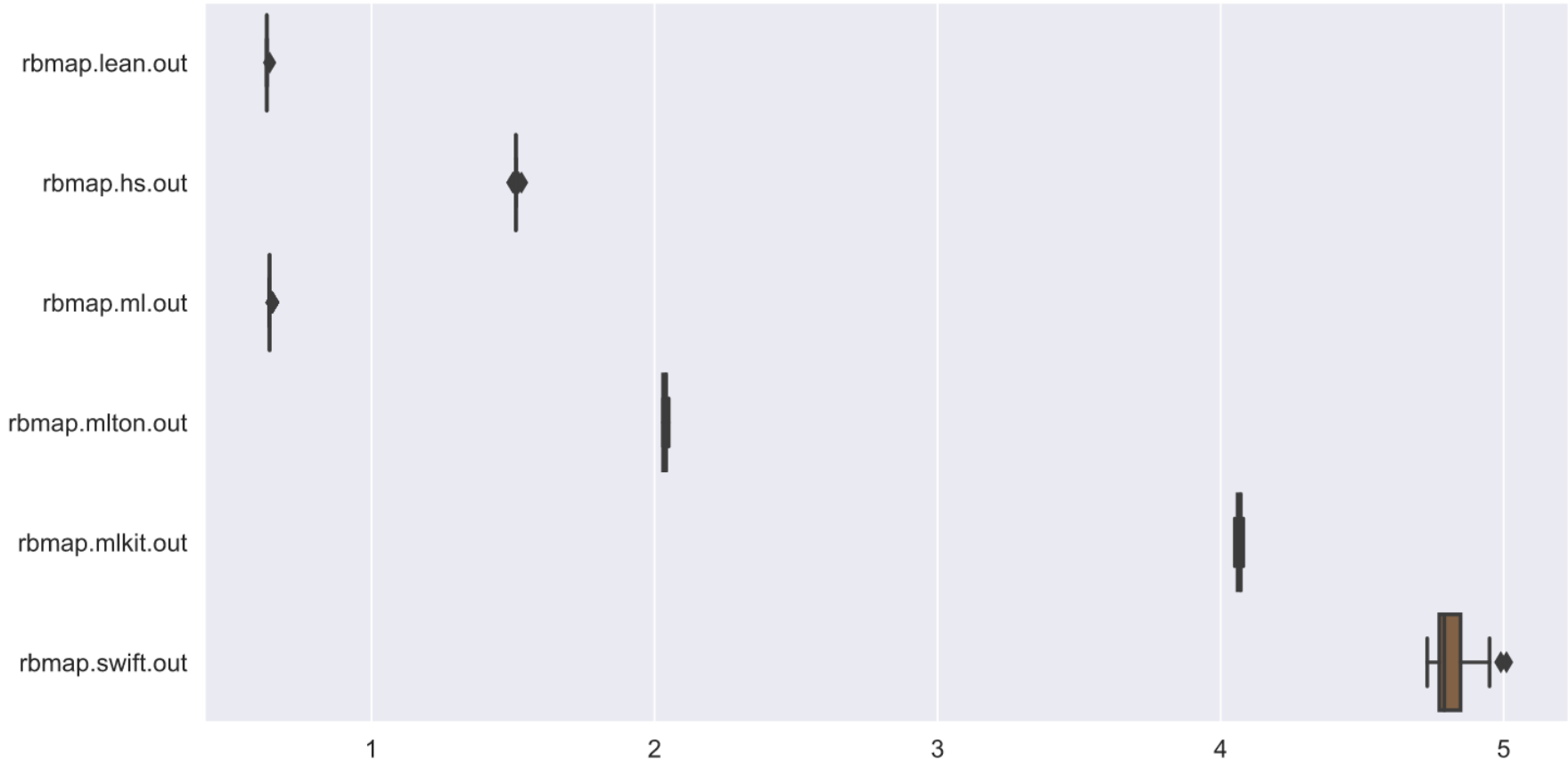
```
match ... with
| pat1, ... => ...
| ...
```

- etc...

```
fun x =>
  let y := 1;
  do a; b
```

# Compiler

Ullrich and de Moura. *Counting Immutable Beans: Reference Counting Optimized for Purely Functional Programming*. IFL'19.



# New typeclass resolution

Karlsruhe Institute of Technology

Performance issues with the old implementation:

- *diamonds* can lead to exponential runtime
- cycles can lead to nontermination

# New typeclass resolution

Karlsruhe Institute of Technology

Performance issues with the old implementation:

- *diamonds* can lead to exponential runtime
- cycles can lead to nontermination

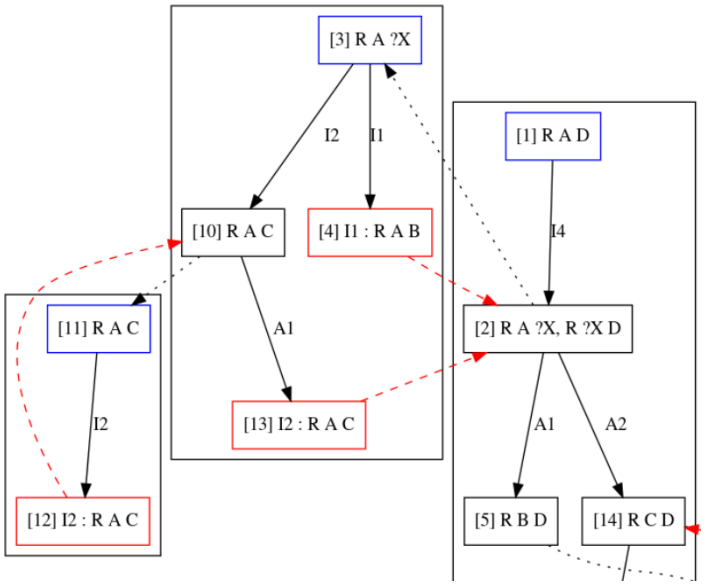
Typeclass resolution follows a “Prolog-like search”

# New typeclass resolution

Performance issues with the old implementation:

- *diamonds* can lead to exponential runtime
- cycles can lead to nontermination

Typeclass resolution follows a “Prolog-like search”
⇒ adapt known Prolog optimization, *tabled resolution*, to Lean!




Guarantees termination if size of typeclass problems is bounded

# State of the ⋃

```
notation "⋃ " binders ", " r:(scoped f, Union f) := r
```

# State of the ⋃

```
notation "⋃ " binders ", " r:(scoped f, Union f) := r
```

```
expected ':='
```

# State of the ⋃

Karlsruhe Institute of Technology

```
notation "⋃ " b ", " r := Union (fun b => r)


#check ⋃ x,              x = x
#check ⋃ (x : Set Unit), x = x
#check ⋃ x ∈ univ,       x = x   -- error
```

# State of the ⋃

```
notation "⋃ " b ", " r := Union {b | r}


#check ⋃ x,              x = x
#check ⋃ (x : Set Unit), x = x
#check ⋃ x ∈ univ,       x = x   -- works!
```

# State of the ⋃

```
syntax "⋃ " term ", " term : term
macro `(⋃ $b, $r) => `(Union {$b | $r})

#check ⋃ x,              x = x
#check ⋃ (x : Set Unit), x = x
#check ⋃ x ∈ univ,       x = x   -- works!
```

# State of the ⋃

```
syntax "⋃ " term ", " term : term
macro `(⋃ $b, $r) => `(Union {$b | $r})

#check ⋃ x,              x = x
#check ⋃ (x : Set Unit), x = x
#check ⋃ x ∈ univ,       x = x   -- works!
```

```
syntax "{" term " | " term "}" : term
macro
| `({$x ∈ $s | $p}) => `(setOf (fun $x => $x ∈ $s ∧ $p))
| `({$x ≤ $e | $p}) => `(setOf (fun $x => $x ≤ $e ∧ $p))

| `({$b      | $r}) => `(setOf (fun $b => $r))
```

# State of the ⋃

```
syntax "⋃ " setIdx ", " term : term
macro `(⋃ $b, $r) => `(Union {$b | $r})

#check ⋃ x,              x = x
#check ⋃ x : Set Unit,   x = x  -- works!
#check ⋃ x ∈ univ,       x = x
```

```
syntax "{" term " | " term "}" : term
macro
| `({$x ∈ $s | $p}) => `(setOf (fun $x => $x ∈ $s ∧ $p))
| `({$x ≤ $e | $p}) => `(setOf (fun $x => $x ≤ $e ∧ $p))

| `({$b      | $r}) => `(setOf (fun $b => $r))
```

# State of the ⋃

```
syntax "⋃ " setIdx ", " term : term
macro `(⋃ $b, $r) => `(Union {$b | $r})

#check ⋃ x,              x = x
#check ⋃ x : Set Unit,   x = x  -- works!
#check ⋃ x ∈ univ,       x = x
```

```
declare_syntax_cat setIdx
syntax term                      : setIdx
syntax ident " : " term          : setIdx
syntax "{" setIdx " | " term "}" : term
macro
| `({$x ∈ $s | $p}) => `(setOf (fun $x => $x ∈ $s ∧ $p))
| `({$x ≤ $e | $p}) => `(setOf (fun $x => $x ≤ $e ∧ $p))
| `({$x : $t | $r}) => `(setOf (fun ($x : $t) => $r))
| `({$b      | $r}) => `(setOf (fun $b => $r))
```

# Thoughts about eventual porting of Lean 3 code

Karlsruhe Institute of Technology

- syntax changes: mostly superficial, automatable

# Thoughts about eventual porting of Lean 3 code

- syntax changes: mostly superficial, automatable

  One possible path: Incrementally reimplement Lean 3 syntax as macros first, then unfold them as final step

```
$ lean --plugin lean3-compat mathlib/src/...
```

# Thoughts about eventual porting of Lean 3 code

- syntax changes: mostly superficial, automatable

  One possible path: Incrementally reimplement Lean 3 syntax as macros first, then unfold them as final step

```
$ lean --plugin lean3-compat mathlib/src/...
```

- elaborator changes: probably not too drastic

# Thoughts about eventual porting of Lean 3 code

Karlsruhe Institute of Technology

- syntax changes: mostly superficial, automatable

  One possible path: Incrementally reimplement Lean 3 syntax as macros first, then unfold them as final step

```
$ lean --plugin lean3-compat mathlib/src/...
```

- elaborator changes: probably not too drastic
- library changes: mostly *missing* API, needs to be reimplemented
  - but not necessarily in the stdlib