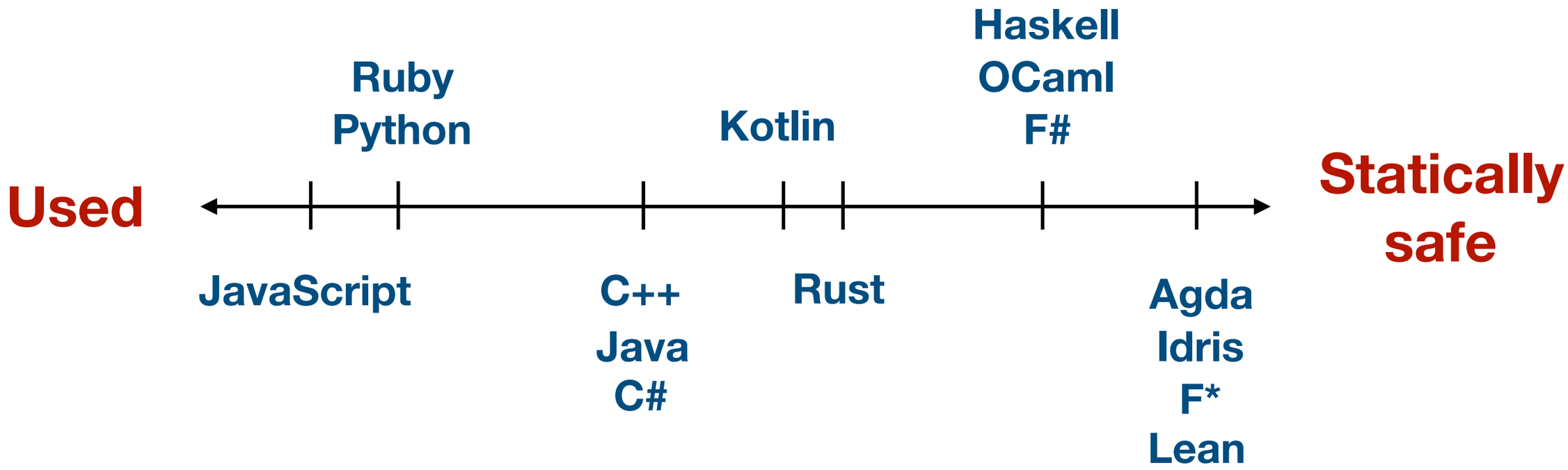
Refinement Types:

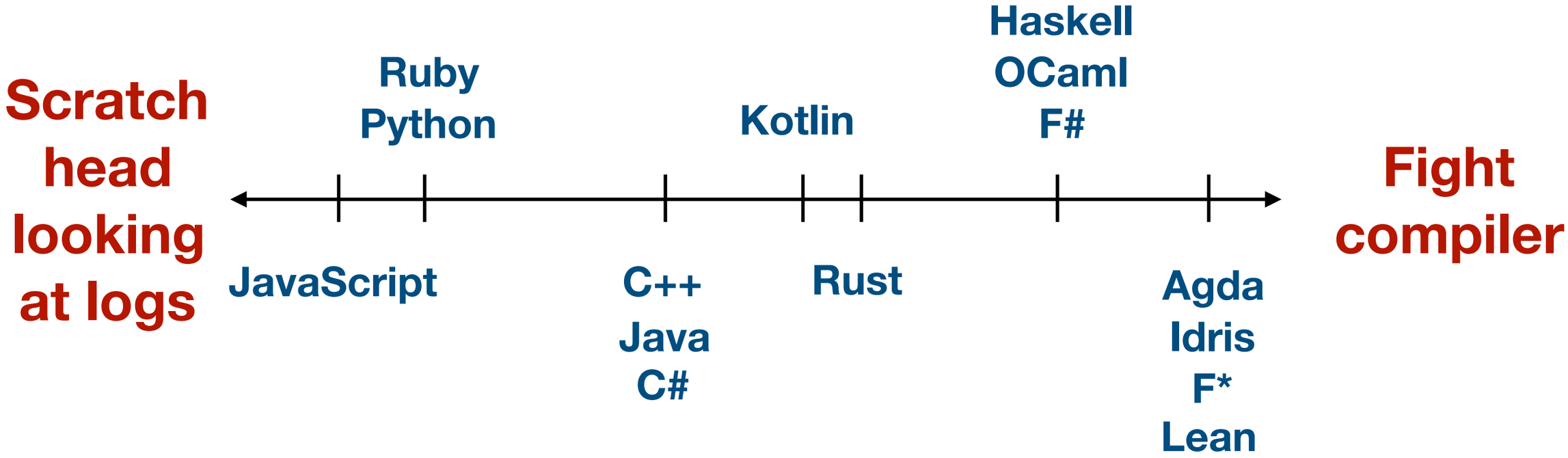
Future of typing, now

Mistral Contrastin

Safer you are, lonelier you get



Safety is hard work



Traditional types are rigid

- Every type is different

  `List α ≠ NonEmpty α ≠ Singleton α`

- ```
  max :: NonEmpty Int -> Int
  max = …

  max (xs :: List Int)        -- type error
  max (xs :: NonEmpty  Int) -- type checks
  max (xs :: Singleton Int) -- type error
  ```
  ```

Refinements are fluid

- Refinement types are related with **subtyping**

```
List  α
⊆
NonEmpty α = { v:List α | length v > 0 }
⊆
Singleton α  = { v:List α | length v = 1 }
```

- ```
  max (xs :: List Int)      -- type error
  max (xs :: NonEmpty  Int) -- type checks
  max (xs :: Singleton Int) -- type checks
  ```
  ```

Type checker with a grade school degree

What does it take to type check `max (xs :: Singleton Int)`?
Prove…

$$\texttt{Singleton Int} \subseteq \texttt{NonEmpty Int}$$

$$\Leftrightarrow$$

$$\texttt{\{ v:List } \alpha \texttt{ | len v = 1 \}}$$
$$\subseteq \texttt{\{ v:List } \alpha \texttt{ | len v > 0 \}}$$

$$\Leftrightarrow$$

$$\texttt{(len v = 1)} \Rightarrow \texttt{(len v > 0)}$$

$$\Leftrightarrow$$

$$\texttt{1 > 0}$$

Satisfiability Modulo Theories can solve 1>0

- SMT decides on certain logical formulae
  - Boolean logic, e.g., $x \wedge y \Rightarrow x \vee y$
  - Theory of linear arithmetic, e.g., $x + y > 0 \Rightarrow 2x > -3y$
  - Theory of arrays, e.g., $arr[x \mapsto 42][x \mapsto 10] \neq 42$
  - Theory of uninterpreted functions, e.g., $f(g(x)) = f(y)$
  - …

Type checking is now context sensitive

```
if (2 * x > 2)
    then …
else …
```

$x > 1$

$x \leq 1$

$$(-498 \times -1307) + 601$$

Abstract interpretation

×

$\times^{\#}$	0	+	-
0	0	0	0
+	0	+	-
-	0	-	+

+

$+^{\#}$	0	+	-
0	0	+	-
+	+	+	?
-	-	?	-

Demo

github.com/madgen/refinement-types-seminar