The Fuzion Language

Fridtjof Siebert

Who is this guy?

Fridtjof Siebert

email: fridi@fridi.de
github: fridis
twitter: @fridi_s

‘90-‘94	AmigaOberon, AMOK PD
‘97	FEC Eiffel Compiler Sparc / Solaris
‘98-‘99	OSF: TurboJ Java Compiler
‘00-‘01	PhD on realtime GC
‘02-‘19	JamaicaVM realtime JVM based on CLASSSPATH / OpenJDK, VeriFlux static analysis tool
‘20-…	Fuzion

Motivation

Many languages overloaded with concepts like classes, methods, interfaces, constructors, traits, records, structs, packages, values, …

⇨ Fuzion has one concept: a feature

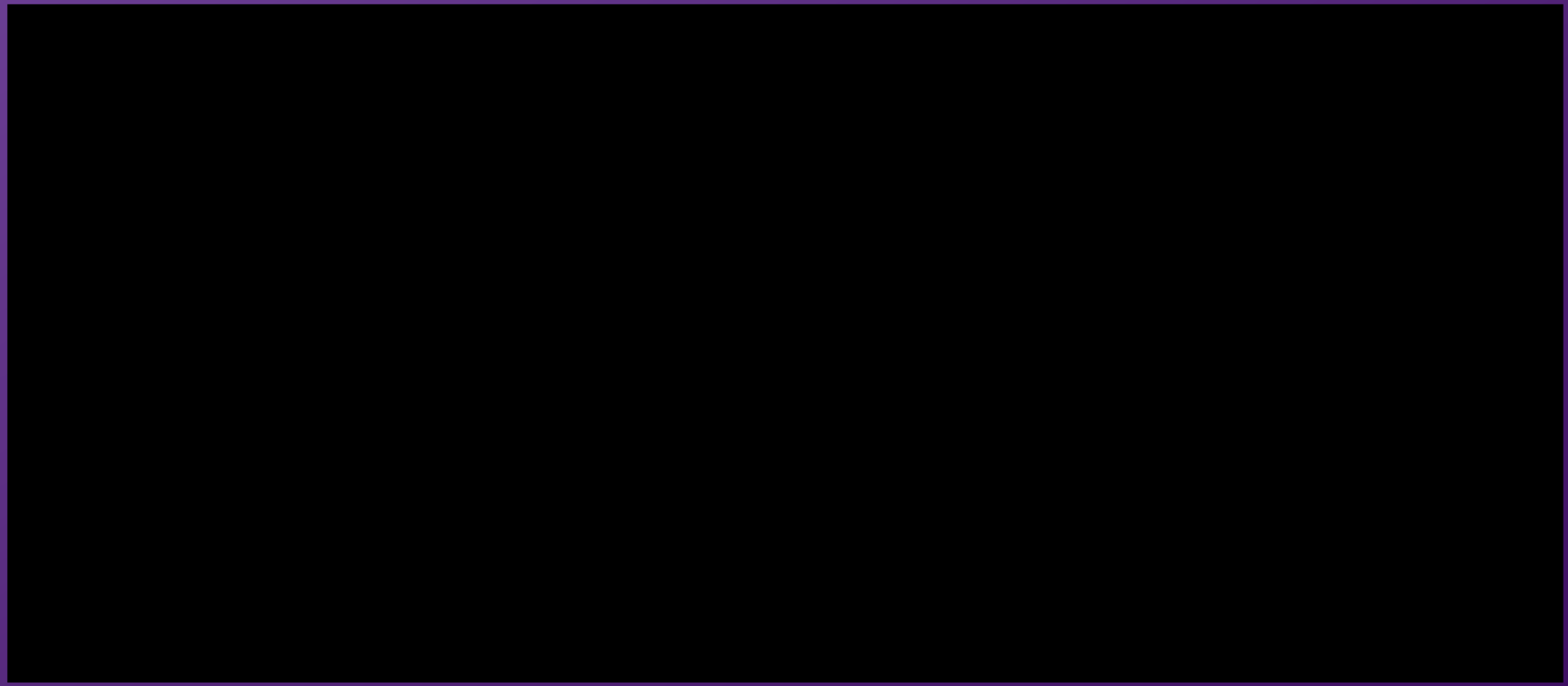
Compilers and tools are more powerful

⇨ Tools make better decisions

Systems are safety-critical

⇨ we need to ensure correctness

Hello World!



Hello World!

```
HelloWorld is
  stdout.println("Hello, World!")
```

Feature Declaration

```
HelloWorld is
  stdout.println("Hello, World!")
```

Feature Call

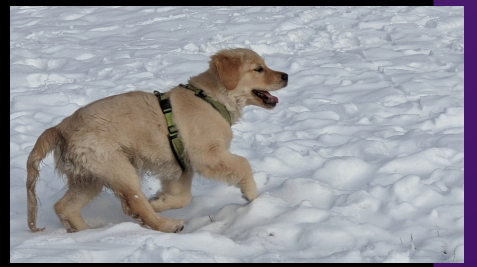
```
HelloWorld is
  stdout.println("Hello, World!")
```

Feature Call

```
HelloWorld is
  stdout.println("Hello, World!")
```

Feature Declaration

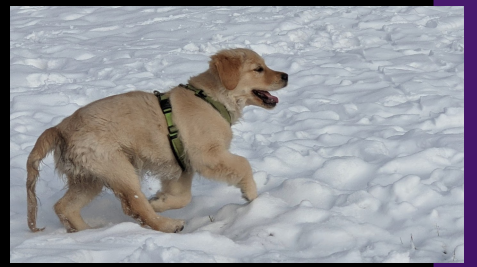
```
myUtils is
  myStack<T>(maxSize i64)
    : lifo<T>, streamable<T>
    pre
      maxSize > 0
    post
      isEmpty
  is
    size u64 = 0
    isEmpty => size = 0
    push(x T) myStack<T>
    is …
```



Feature Declaration

⇨ a name

```
myUtils is
  myStack<T>(maxSize i64)
    : lifo<T>, streamable<T>
    pre
      maxSize > 0
    post
      isEmpty
  is
    size u64 = 0
    isEmpty => size = 0
    push(x T) myStack<T>
    is …
```

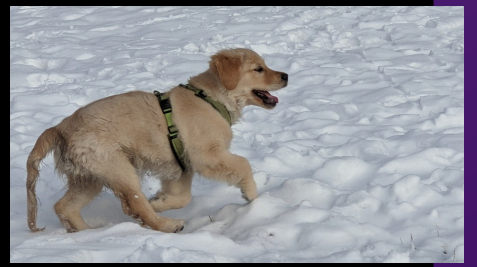


Feature Declaration

⇨ a name

⇨ formal args, result type

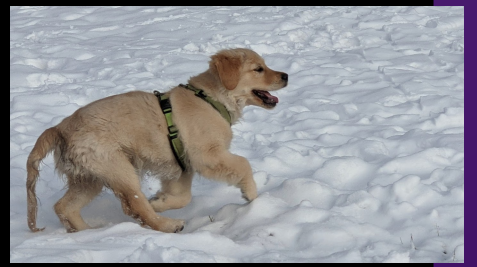
```
myUtils is
  myStack<T>(maxSize i64)
    : lifo<T>, streamable<T>
    pre
      maxSize > 0
    post
      isEmpty
  is
    size u64 = 0
    isEmpty => size = 0
    push(x T) myStack<T>
    is …
```



Feature Declaration

- ⇨ a name
- ⇨ formal args, result type
- ⇨ an outer feature

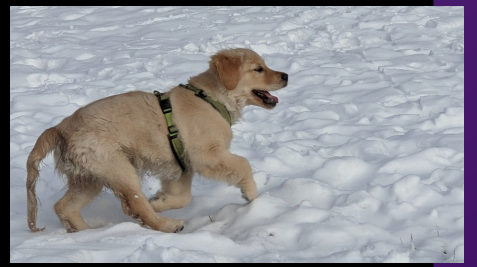
```
myUtils is
  myStack<T>(maxSize i64)
    : lifo<T>, streamable<T>
    pre
      maxSize > 0
    post
      isEmpty
  is
    size u64 = 0
    isEmpty => size = 0
    push(x T) myStack<T>
    is …
```



Feature Declaration

- ⇨ a name
- ⇨ formal args, result type
- ⇨ an outer feature
- ⇨ list of formal generics

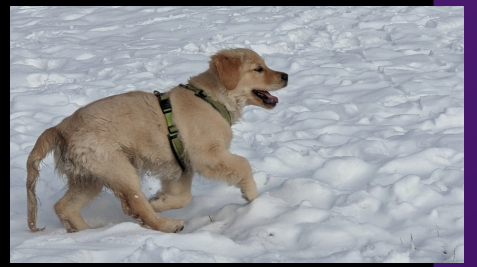
```
myUtils is
  myStack<T>(maxSize i64)
    : lifo<T>, streamable<T>
    pre
      maxSize > 0
    post
      isEmpty
  is
    size u64 = 0
    isEmpty => size = 0
    push(x T) myStack<T>
    is …
```



Feature Declaration

- ⇨ a name
- ⇨ formal args, result type
- ⇨ an outer feature
- ⇨ list of formal generics
- ⇨ list of parent feature calls

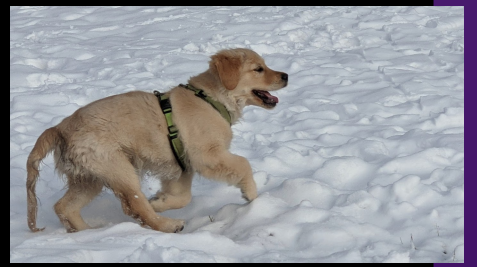
```
myUtils is
  myStack<T>(maxSize i64)
    : lifo<T>, streamable<T>
      pre
        maxSize > 0
      post
        isEmpty
  is
    size u64 = 0
    isEmpty => size = 0
    push(x T) myStack<T>
    is …
```



Feature Declaration

- ⇨ a name
- ⇨ formal args, result type
- ⇨ an outer feature
- ⇨ list of formal generics
- ⇨ list of parent feature calls
- ⇨ a contract

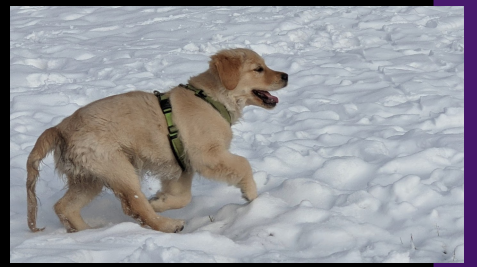
```
myUtils is
  myStack<T>(maxSize i64)
   : lifo<T>, streamable<T>
    pre
      maxSize > 0
    post
      isEmpty
  is
    size u64 = 0
    isEmpty => size = 0
    push(x T) myStack<T>
    is …
```



Feature Declaration

- ⇨ a name
- ⇨ formal args, result type
- ⇨ an outer feature
- ⇨ list of formal generics
- ⇨ list of parent feature calls
- ⇨ a contract
- ⇨ an implementation
  - → inner features

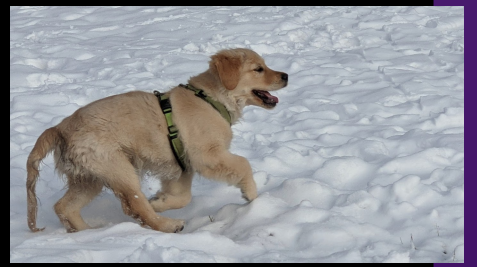
```
myUtils is
  myStack<T>(maxSize i64)
   : lifo<T>, streamable<T>
    pre
      maxSize > 0
    post
      isEmpty
  is
    size u64 => 0
    isEmpty => size = 0
    push(x T) myStack<T>
    is …
```



Feature Declaration

- ⇨ a name
- ⇨ formal args, result type
- ⇨ an outer feature
- ⇨ list of formal generics
- ⇨ list of parent feature calls
- ⇨ a contract
- ⇨ an implementation
  - → inner features

```
myUtils is
  myStack<T>(maxSize i64)
   : lifo<T>, streamable<T>
    pre
      maxSize > 0
    post
      isEmpty
  is
    size u64 => 0
    isEmpty => size = 0
    push(x T) myStack<T>
    is …
```



Syntactic Sugar

mapped to feature declarations and calls

Syntactic Sugar

mapped to feature declarations and calls

⇨ loops tail-recursive features

Syntactic Sugar

mapped to feature declarations and calls

- ⇨ loops — tail-recursive features
- ⇨ first class functions — feature **`function`** heirs

Syntactic Sugar

mapped to feature declarations and calls

⇨ loops	tail-recursive features
⇨ first class functions	feature **`function`** heirs
⇨ union types	feature **`choice`** with generics

Syntactic Sugar

mapped to feature declarations and calls

⇨ loops	tail-recursive features
⇨ first class functions	feature **`function`** heirs
⇨ union types	feature **`choice`** with generics
⇨ tuples	feature **`tuple`** with generics

Syntactic Sugar

mapped to feature declarations and calls

- ⇨ loops — tail-recursive features
- ⇨ first class functions — feature `function` heirs
- ⇨ union types — feature **`choice`** with generics
- ⇨ tuples — feature **`tuple`** with generics

See Tutorial at https://flang.dev

Immutability

Fields can be assigned a value only once

- ⇨ great for static analysis
- ⇨ great for optimizations
- ⇨ multi-threading becomes race-free without locks
- ⇨ this scares many developers

Immutability in Fuzion

Most fields are immutable

⇨ immutable field declaration syntax simple

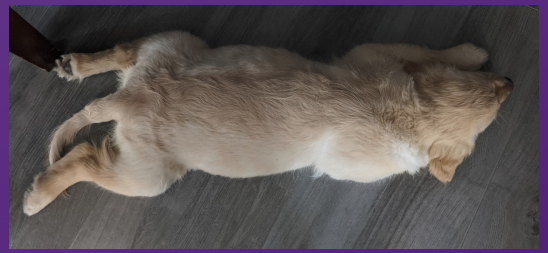
```
x := expr()
```

vs.

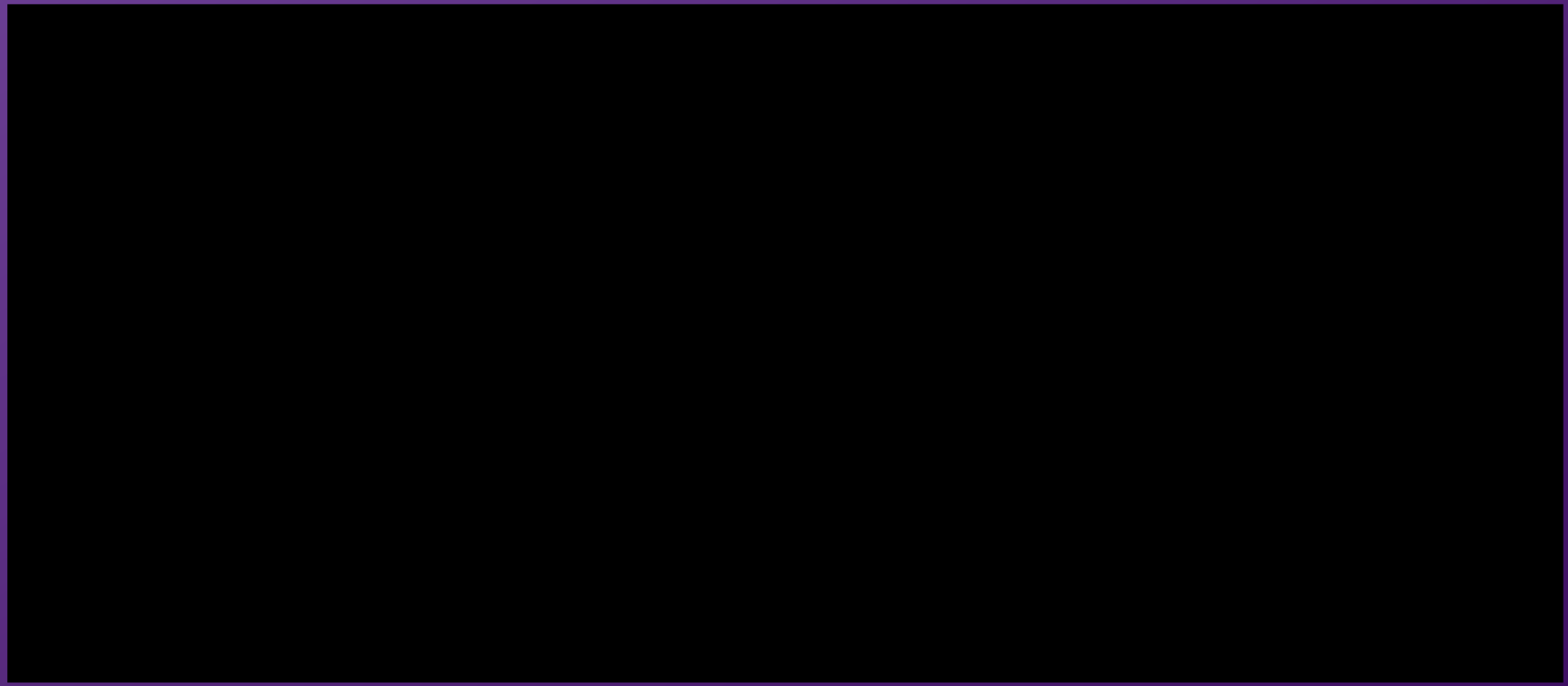
```
x T
set x := expr()
```

⇨ loops converted to tail-recursive functions

⇨ fields shared by threads must be immutable



Memory Management



Memory Management

```
Complex(re, im f64) is
  abs => sqrt(re*re + im*im)
```

Memory Management

```
Complex(re, im f64) is
  abs => sqrt(re*re + im*im)

a := Complex(3,4).abs

f(Complex(5,12))
```

Memory Management

```
Complex(re, im f64) is
  abs => sqrt(re*re + im*im)

a := Complex(3,4).abs    # register or stack

f(Complex(5,12))
```

Memory Management

```
Complex(re, im f64) is
  abs => sqrt(re*re + im*im)

a := Complex(3,4).abs     # register or stack

f(Complex(5,12))          # stack or heap
```

Memory Management

Memory Management

Static analysis detects lifespan of values

Memory Management

Static analysis detects lifespan of values

⇨ register / stack allocation when possible

Memory Management

Static analysis detects lifespan of values

⇨ register / stack allocation when possible

⇨ heap allocation otherwise

Memory Management

Static analysis detects lifespan of values

⇨ register / stack allocation when possible

⇨ heap allocation otherwise

⇨ realtime GC handles heap deallocation

Memory Management

Static analysis detects lifespan of values

⇨ register / stack allocation when possible

⇨ heap allocation otherwise

⇨ realtime GC handles heap deallocation

→ bounded allocation time

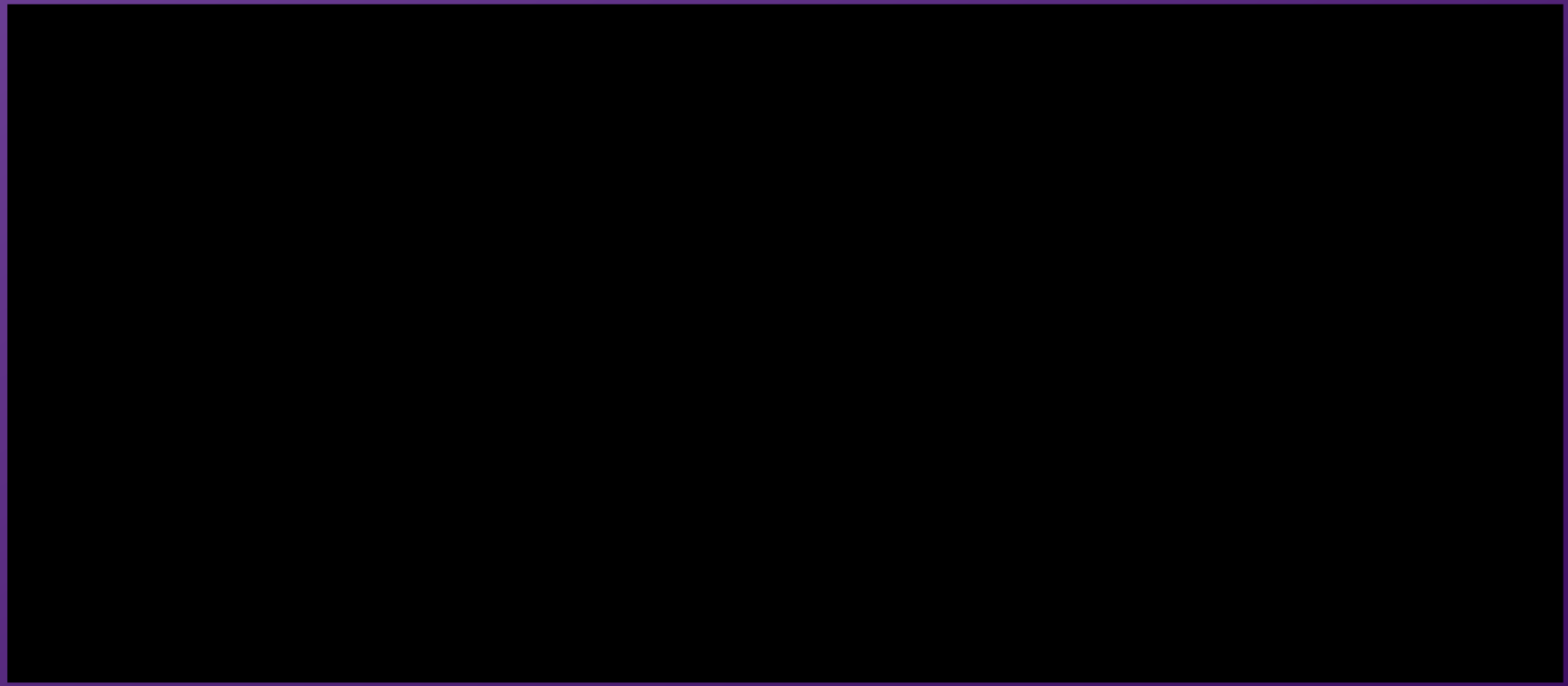
→ fragmentation solved

Memory Management

Realtime GC

- ⇨ parallel mark and sweep standard algorithm
- ⇨ write barrier ensuring strong invariant
- ⇨ segregated size classes to reduce fragmentation
- ⇨ tile-based compaction to fix fragmentation
- ⇨ parallel implementation to avoid pauses
- ⇨ work-based as fallback

Type Inference



Type Inference

```
a := x + 3                    # expression to type
```

Type Inference

```
a := x + 3                  # expression to type

f => if x<y a else b  # union of types
```

Type Inference

```
a := x + 3                  # expression to type

f => if x<y a else b  # union of types

flags u128 = 1 << 124 # reverse inference
```

Type Inference

```
a := x + 3                  # expression to type

f => if x<y a else b        # union of types

flags u128 = 1 << 124 # reverse inference

flags := 1 << 24            # default integer type
```

Type Inference

Expression to assigned field or function result

⇨ type is expression type

⇨ performed lazily during type inferencing phase

⇨ cycles detected, cause error

Type Inference

Expression to assigned field or function result

⇨ type is expression type

⇨ performed lazily during type inferencing phase

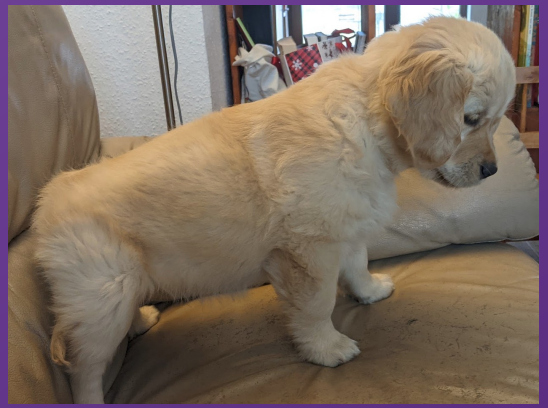
⇨ cycles detected, cause error

Constant numeric expressions

⇨ type inferred from assignment target

⇨ default type, if not present (i64, f64)

Fuzion Implementation State



Fuzion is still in a proof-of-concept state

- ⇨ Interpreter on top of JVM
- ⇨ C code backend under development
- ⇨ Analysis tools still under development

Help wanted

Resources related to Fuzion

- ⇨ web: https://flang.dev
- ⇨ github: https://github.com/fridis/fuzion
- ⇨ give feedback


The End