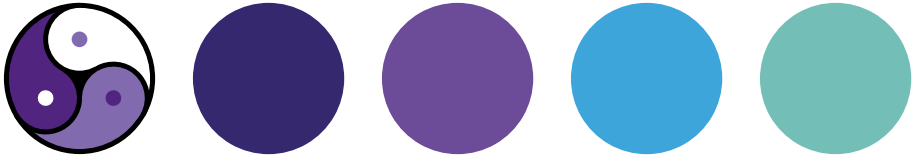
Effects Everywhere

Error Handling and Design-By-Contract in Fuzion

Fridtjof Siebert
Tokiwa Software GmbH

FOSDEM 2025, 2. Feb 2025, Brussels

TOKIWA software



Who is this guy?

Fridtjof Siebert

siebert@tokiwa.software

fridis

@fridis.bsky.social

@fridi@mastodon.social

fridi_si@instagram

1990	AmigaOberon, AMOK PD
1997	FEC Eiffel Sparc / Solaris
1998	OSF: TurboJ Java Compiler
2000	PhD on real-time GC
2002	JamaicaVM real-time JVM based on CLASSSPATH / OpenJDK, VeriFlux static analysis tool
2020	Fuzion
2021	Tokiwa Software

TOKIWA software

Talk Overview

- ➡ Fuzion quick intro
- ➡ Example
- ➡ Design-by-Contract
- ➡ Effect Handlers
- ➡ Fallible Hierarchy
- ➡ Type Constraints
- ➡ Status

TOKIWA software

Talk Overview

- Fuzion quick intro
- Example
- Design-by-Contract
- Effect Handlers
- Fallible Hierarchy
- Type Constraints
- Status

TOKIWA software

Motivation: Fuzion Language

- ➡ One concept: a feature
- ➡ Systems are safety-critical
- ➡ Tools make developer's life easier
- ➡ Fuzion is
  - statically typed
  - polymorphic: union types, parametric types, inheritance
  - pure using effects

Talk Overview

- Fuzion quick intro
- Example
- Design-by-Contract
- Effect Handlers
- Fallible Hierarchy
- Type Constraints
- Status

TOKIWA software

Introductory Example

```
square(x N : numeric) ⇒ "{x}² = {x*x}"
```


TOKIWA software

Introductory Example

```
square(x N : numeric) ⇒ "{x}² = {x*x}"

square(N type : numeric, x N) String ⇒ "{x}² = {x*x}"
```


Introductory Example

```
square(x N : numeric) ⇒ "{x}² = {x*x}"
```


TOKIWA software

Introductory Example

```
square(x N : numeric) ⇒ "{x}² = {x*x}"

square 32                  |> say
square 1/-/3               |> say
square (num.complex 1 1)   |> say
```


Introductory Example

```
square(x N : numeric) ⇒ "{x}² = {x*x}"

square 32                  |> say
square 1/-/3               |> say
square (num.complex 1 1)   |> say
```

```
> fz square.fz
32² = 1024
1/3² = 1/9
1+1i² = 0+2i
>
```


Introductory Example

```
square(x N : numeric) ⇒ "{x}² = {x*x}"

square 32                    |> say
square 1/-/3                 |> say
square (num.complex 1 1)     |> say
square (u8 30)               |> say
```

```
>
```


TOKIWA software

Introductory Example

```
square(x N : numeric) ⇒ "{x}² = {x*x}"

square 32                    |> say
square 1/-/3                 |> say
square (num.complex 1 1)     |> say
square (u8 30)               |> say
```

```
> fz square.fz
32² = 1024
1/3² = 1/9
1+1i² = 0+2i

error 1: FATAL FAULT
`precondition`: debug:
(numeric.this *! other)
Call stack:
fuzion.sys.fatal_fault#2
fuzion.type.runtime.type.fault.t
ype.install_default.λ.call#1
fuzion.runtime.fault.cause#1
```


TOKIWA software

Introductory Example

```
square(x N : numeric) ! panic ⇒
  if x *! x                # *! checks if * would succeed without overflow
    "{x}² = {x*x}"
  else
    panic "overflow for $x”

square (u8 30) |> say
```


TOKIWA software

Introductory Example

```
square(x N : numeric) ! panic ⇒
  if x *! x                  # *! checks if * would succeed without overflow
    "{x}² = {x*x}"
  else
    panic "overflow for $x”

square (u8 30) |> say
```


TOKIWA software

Introductory Example

```
square(x N : numeric) ! panic ⇒
  if x *! x                # *! checks if * would succeed without overflow
    "{x}² = {x*x}"
  else
    panic "overflow for $x”

square (u8 30) |> say
```


TOKIWA software

Introductory Example

```
square(x N : numeric) ! panic ⇒
  if x *! x                # *! checks if * would succeed without overflow
    "{x}² = {x*x}"
  else
    panic "overflow for $x”

square (u8 30) |> say
```


TOKIWA software

Introductory Example

```
square(x N : numeric) ! panic ⇒
  if x *! x                # *! checks if * would succeed without overflow
    "{x}² = {x*x}"
  else
    panic "overflow for $x”

square (u8 30) |> say
```

```
> fz square.fz

error 1: FATAL FAULT `*** panic ***`: overflow for 30
Call stack:
fuzion.sys.fatal_fault#2
fuzion.type.runtime.type.fault.type.install_default.λ.call#1
fuzion.runtime.fault.cause#1
panic.type.install_default.λ.call#1
```

Introductory Example

```
square(x N : numeric) ! panic ⇒
  if x *! x                # *! checks if * would succeed without overflow
    "{x}² = {x*x}"
  else
    panic "overflow for $x"

square (u8 30) |> say
```

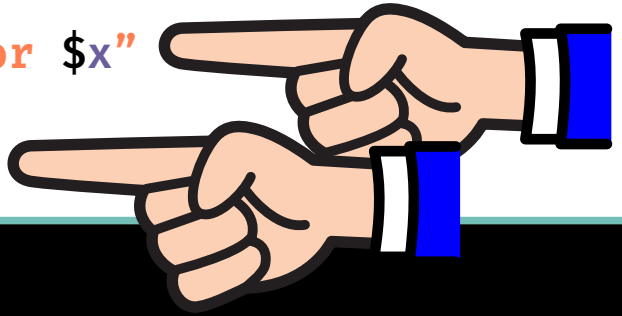
```
> fz square.fz

error 1: FATAL FAULT `*** panic ***`: overflow for 30
Call stack:
fuzion.sys.fatal_fault#2
fuzion.type.runtime.type.fault.type.install_default.λ.call#1
fuzion.runtime.fault.cause#1
panic.type.install_default.λ.call#1
```

Introductory Example

```
square(x N : numeric) ! panic ⇒
  if x *! x                # *! checks if * would succeed without overflow
    "{x}² = {x*x}"
  else
    panic "overflow for $x"

square (u8 30) |> say
```



```
> fz square.fz

error 1: FATAL FAULT `*** panic ***`: overflow for 30
Call stack:
fuzion.sys.fatal_fault#2
fuzion.type.runtime.type.fault.type.install_default.λ.call#1
fuzion.runtime.fault.cause#1
panic.type.install_default.λ.call#1
```

Talk Overview

- Fuzion quick intro
- Example
- Design-by-Contract
- Effect Handlers
- Fallible Hierarchy
- Type Constraints
- Status

TOKIWA software

Design-by-Contract

Routines formally define

- ➡ preconditions: What does the caller need to provide?
- ➡ postconditions: What does the routine guarantee?

A formal contract offered by routine to the caller

- ➡ validated via
  - runtime checks, or
  - static analysis
  - (auto-generated) tests

Design-by-Contract

```
square(x N : numeric)
  pre
    debug: x *! x
  post
    debug: result ≠ ""
⇒
  "{x}² = {x*x}"

square (u8 30) |> say
```


TOKIWA software

Design-by-Contract

```
square(x N : numeric)
  pre
    debug: x *! x
  post
    debug: result ≠ ""
⇒
  "{x}² = {x*x}"

square (u8 30) |> say
```


TOKIWA software

Design-by-Contract

```
square(x N : numeric)
  pre
    debug: x *! x
  post
    debug: result ≠ ""
⇒
  "{x}² = {x*x}"

square (u8 30) |> say
```


TOKIWA software

Design-by-Contract

```
square(x N : numeric)
  pre
    debug: x *! x
  post
    debug: result ≠ ""
⇒
  "{x}² = {x*x}"

square (u8 30) |> say
```


TOKIWA software

Design-by-Contract

```
square(x N : numeric)
  pre
    debug: x *! x
  post
    debug: result ≠ ""
⇒
  "{x}² = {x*x}"

square (u8 30) |> say
```

```
> fz square.fz
```

Design-by-Contract

```
square(x N : numeric)
  pre
    debug: x *! x
  post
    debug: result ≠ ""
⇒
  "{x}² = {x*x}"

square (u8 30) |> say
```

```
> fz square.fz
error 1: FATAL FAULT `precondition`: debug: x *! x
[ ... ]
pre square u8
precall square u8
universe
```

Talk Overview

- Fuzion quick intro
- Example
- Design-by-Contract
- Effect Handlers
- Fallible Hierarchy
- Type Constraints
- Status

TOKIWA software

Effects Handlers

Effect Handlers are Fuzion features

- ➡ define set of operations
  - have (side-) effects
  - produce a result
  - abort
- ➡ must be instated to be used

Effects Handler: `fallible`

Provides features try and catch:

```
public fallible(ERROR type, h ERROR→void) : effect is

  # try -- run code and handle fault of type `fallible.this`.
  #
  public type.try(code_try ()→T) eff.try ERROR fallible.this

  # cause fault with the given error
  #
  public cause(e ERROR) void
```


Effects Handler: `fallible`

Provides features try and catch:

```
public fallible(ERROR type, h ERROR→void) : effect is

  # try -- run code and handle fault of type `fallible.this`.
  #
  public type.try(code_try ()→T) eff.try ERROR fallible.this

  # cause fault with the given error
  #
  public cause(e ERROR) void
```


TOKIWA software

Effects Handler: `fallible`

Provides features try and catch:

```
public fallible(ERROR type, h ERROR→void) : effect is

  # try -- run code and handle fault of type `fallible.this`.
  #
  public type.try(code_try ()→T) eff.try ERROR fallible.this

  # cause fault with the given error
  #
  public cause(e ERROR) void
```


Effects Handler: `fallible`

```
my_fallible : eff.fallible String is
   ...

my_fallible.try ()→
     ...
        my_fallible.env.cause “some error”
     ...
  .catch msg→
     ...
        # handle failure
     ...
```


TOKIWA software

Effects Handler: `fallible`

```
my_fallible : eff.fallible String is
   ...

my_fallible.try ()→
     ...
        my_fallible.env.cause “some error”
     ...
  .catch msg→
     ...
        # handle failure
     ...
```

TOKIWA software

Effects Handler: `fallible`

```
my_fallible : eff.fallible String is
   ...

my_fallible.try ()→
     ...
        my_fallible.env.cause “some error”
     ...
   .catch msg→
     ...
        # handle failure
     ...
```


TOKIWA software

Effects Handler: `fallible`

```
my_fallible : eff.fallible String is
   ...

my_fallible.try ()→
     ...
        my_fallible.env.cause “some error”
     ...
   .catch msg→
     ...
        # handle failure
     ...
```


TOKIWA software

Effects Handlers: `panic`

```
square(x N : numeric) ! panic ⇒
  if x *! x              # *! checks if * would succeed without overflow
    "{x}² = {x*x}"
  else
    panic "overflow for $x”
```


TOKIWA software

Effects Handlers: `panic`

```
square(x N : numeric) ! panic ⇒
  if x *! x              # *! checks if * would succeed without overflow
    "{x}² = {x*x}"
  else
    panic "overflow for $x”

panic.try ()→
    square (u8 30) |> say
```


TOKIWA software

Effects Handlers: `panic`

```
square(x N : numeric) ! panic ⇒
  if x *! x              # *! checks if * would succeed without overflow
    "{x}² = {x*x}"
  else
    panic "overflow for $x”

panic.try ()→
    square (u8 30) |> say
  .catch msg→
    say "got '$msg', trying uint:"
    square (uint 30) |> say
```


TOKIWA software

Effects Handlers: `panic`

```
square(x N : numeric) ! panic ⇒
  if x *! x              # *! checks if * would succeed without overflow
    "{x}² = {x*x}"
  else
    panic "overflow for $x”

panic.try ()→
    square (u8 30) |> say
  .catch msg→
    say "got '$msg', trying uint:"
    square (uint 30) |> say
```

```
> fz square.fz
```


TOKIWA software

Effects Handlers: `panic`

```
square(x N : numeric) ! panic ⇒
  if x *! x              # *! checks if * would succeed without overflow
    "{x}² = {x*x}"
  else
    panic "overflow for $x”

panic.try ()→
    square (u8 30) |> say
  .catch msg→
    say "got '$msg', trying uint:"
    square (uint 30) |> say
```

```
> fz square.fz
got 'overflow for u8 30',
trying uint:
30² = 900
>
```

TOKIWA software

Effects Handlers: contracts

```
square(x N : numeric)
  pre
    debug: x *! x
⇒
  "{x}² = {x*x}"
```


TOKIWA software

Effects Handlers: contracts

```
square(x N : numeric)
  pre
    debug: x *! x
⇒
  "{x}² = {x*x}"

fuzion.runtime.pre_fault.try ()→
    square (u8 30) |> say
```


TOKIWA software

Effects Handlers: contracts

```
square(x N : numeric)
  pre
    debug: x *! x
⇒
  "{x}² = {x*x}"

fuzion.runtime.pre_fault.try ()→
    square (u8 30) |> say
  .catch msg→
    say "got '$msg', trying uint:"
    square (uint 30) |> say
```

TOKIWA software

Effects Handlers: contracts

```
square(x N : numeric)
  pre
    debug: x *! x
⇒
  "{x}² = {x*x}"

fuzion.runtime.pre_fault.try ()→
    square (u8 30) |> say
  .catch msg→
    say "got '$msg', trying uint:"
    square (uint 30) |> say
```

```
> fz square.fz
```


Effects Handlers: contracts

```
square(x N : numeric)
  pre
    debug: x *! x
⇒
  "{x}² = {x*x}"

fuzion.runtime.pre_fault.try ()→
    square (u8 30) |> say
  .catch msg→
    say "got '$msg', trying uint:"
    square (uint 30) |> say
```

```
> fz square.fz
got 'debug: x *! x', trying
uint:
30² = 900
>
```


TOKIWA software

Talk Overview

- Fuzion quick intro
- Example
- Design-by-Contract
- Effect Handlers
- **Fallible Hierarchy**
- Type Constraints
- Status

TOKIWA software

Fallible Hierarchy

We have a number of different faults

- ➡ `panic`, `pre_fault`, `post_fault`, `check_fault`, user defined, ...

We need some hierarchy

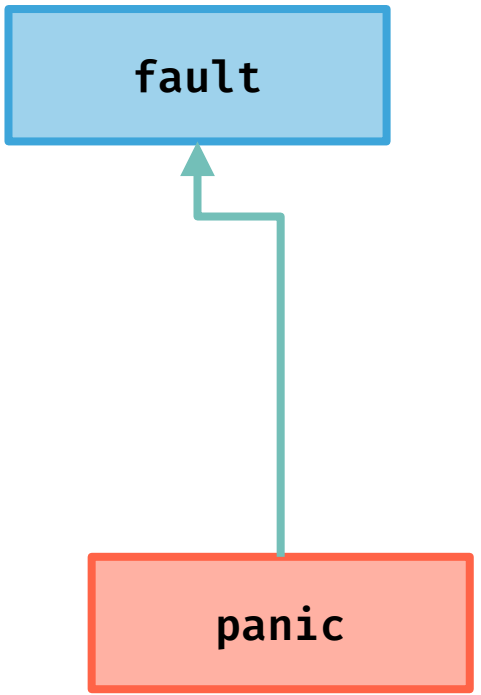
- ➡ e.g., Java: `Throwable`, `Error`, `Exception`, `IOException`

Solution

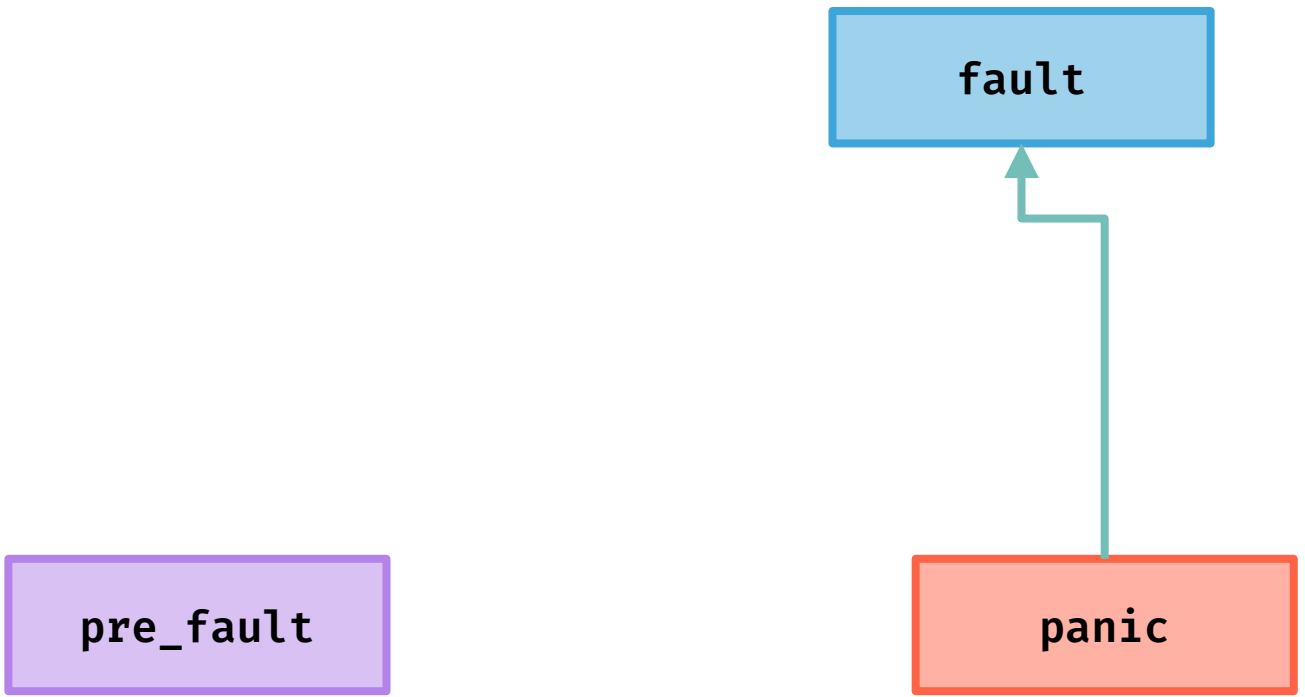
- ➡ cascading fallibles

Fallible Hierarchy

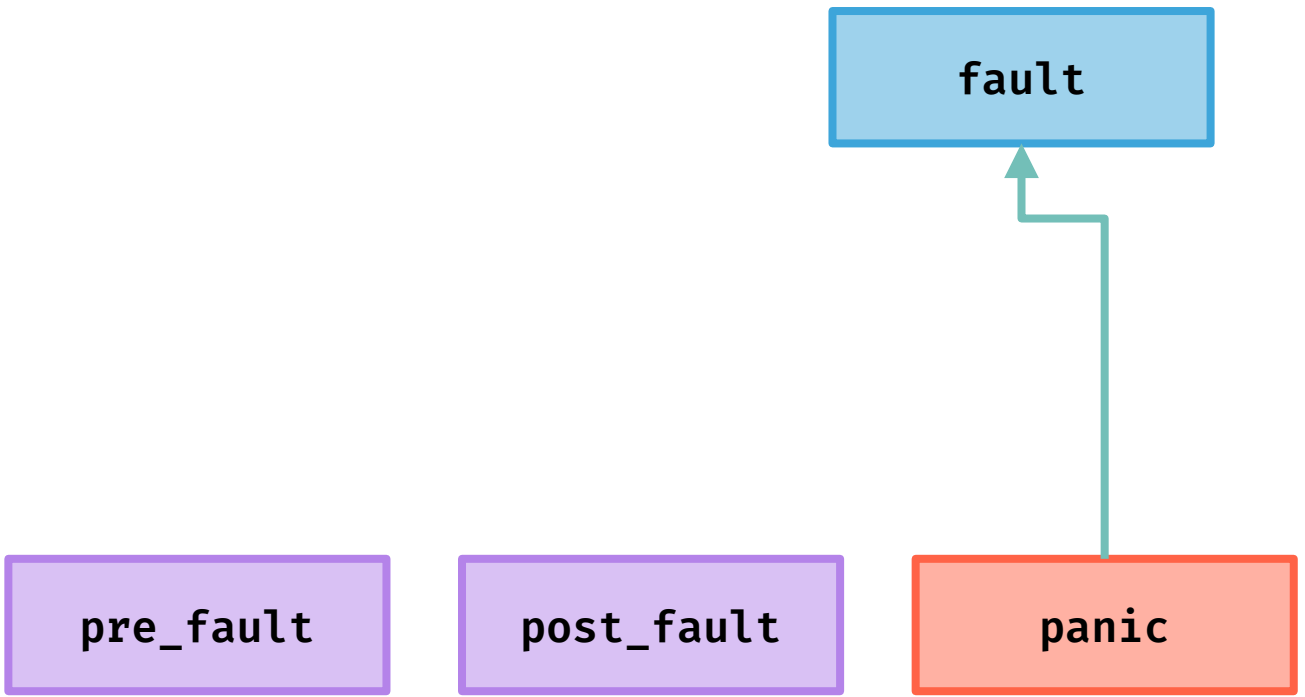
Fallible Hierarchy



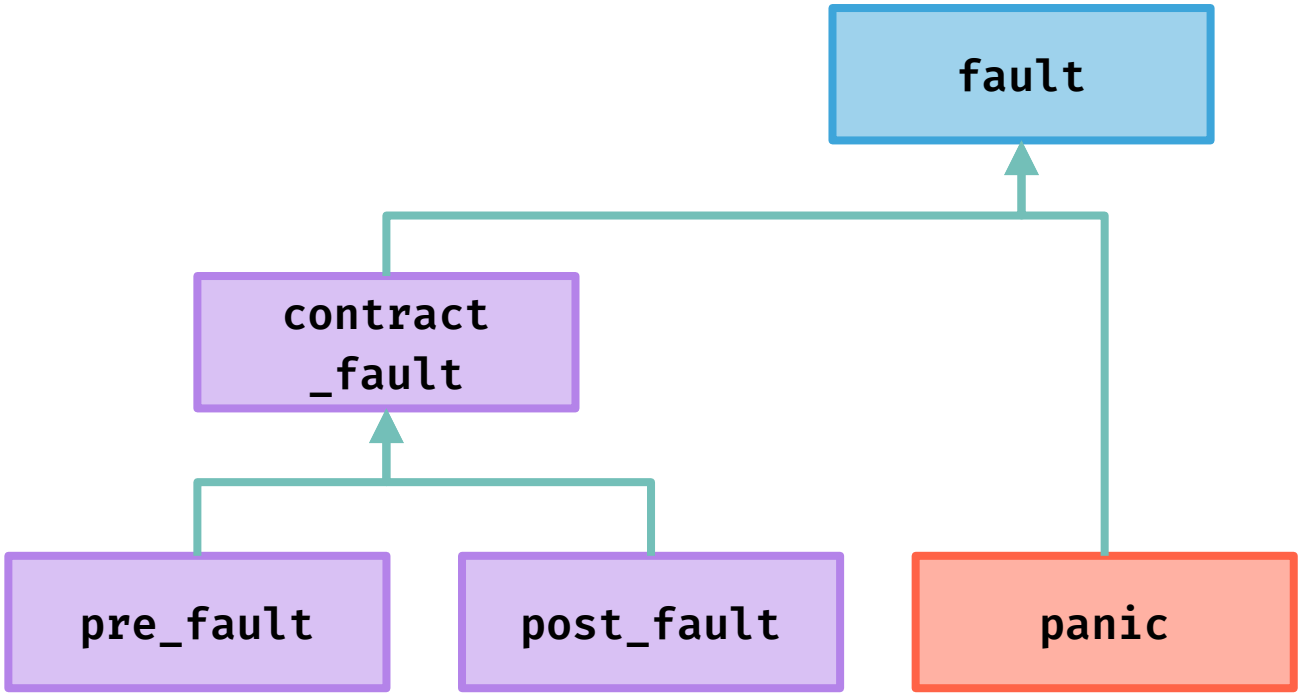
Fallible Hierarchy



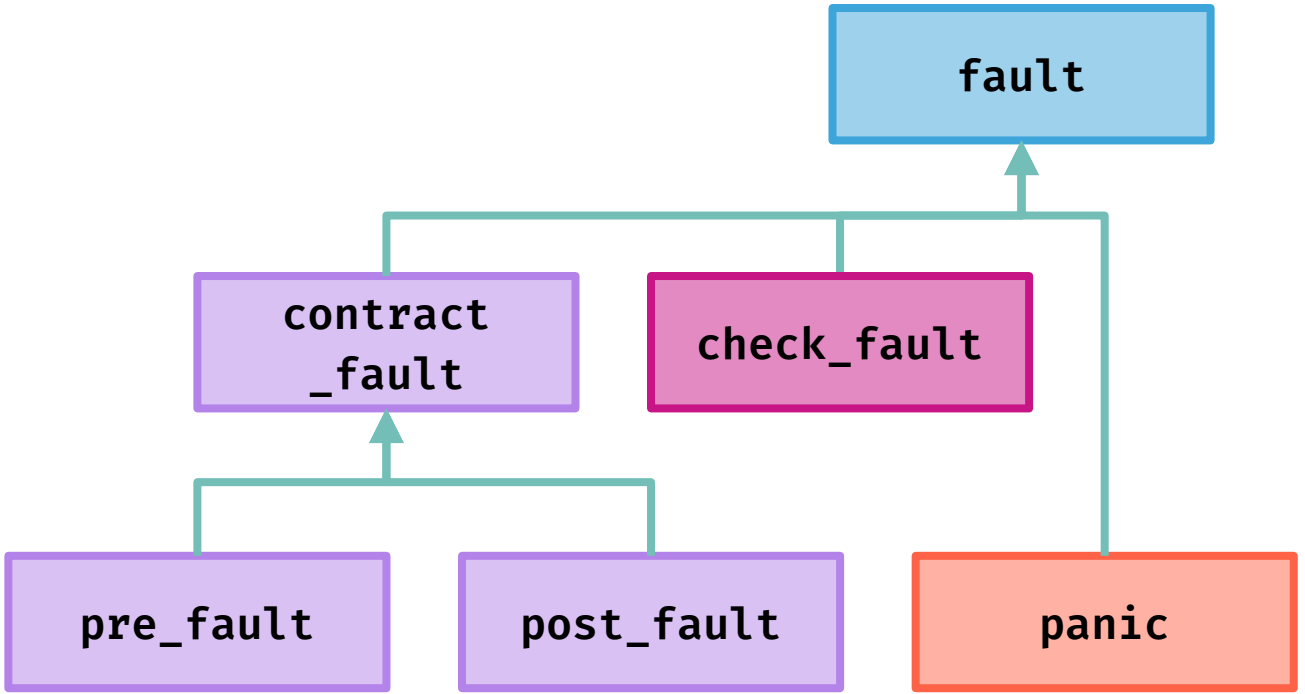
Fallible Hierarchy



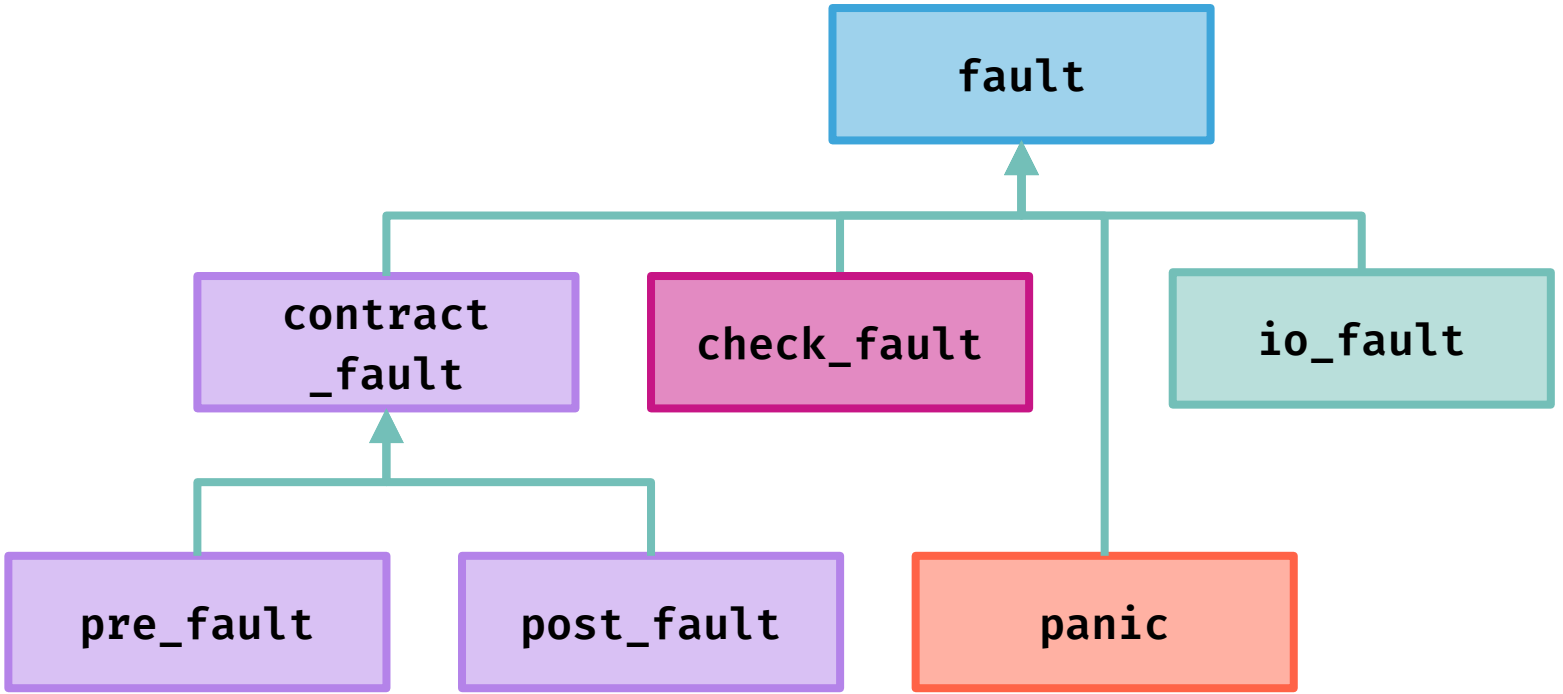
Fallible Hierarchy



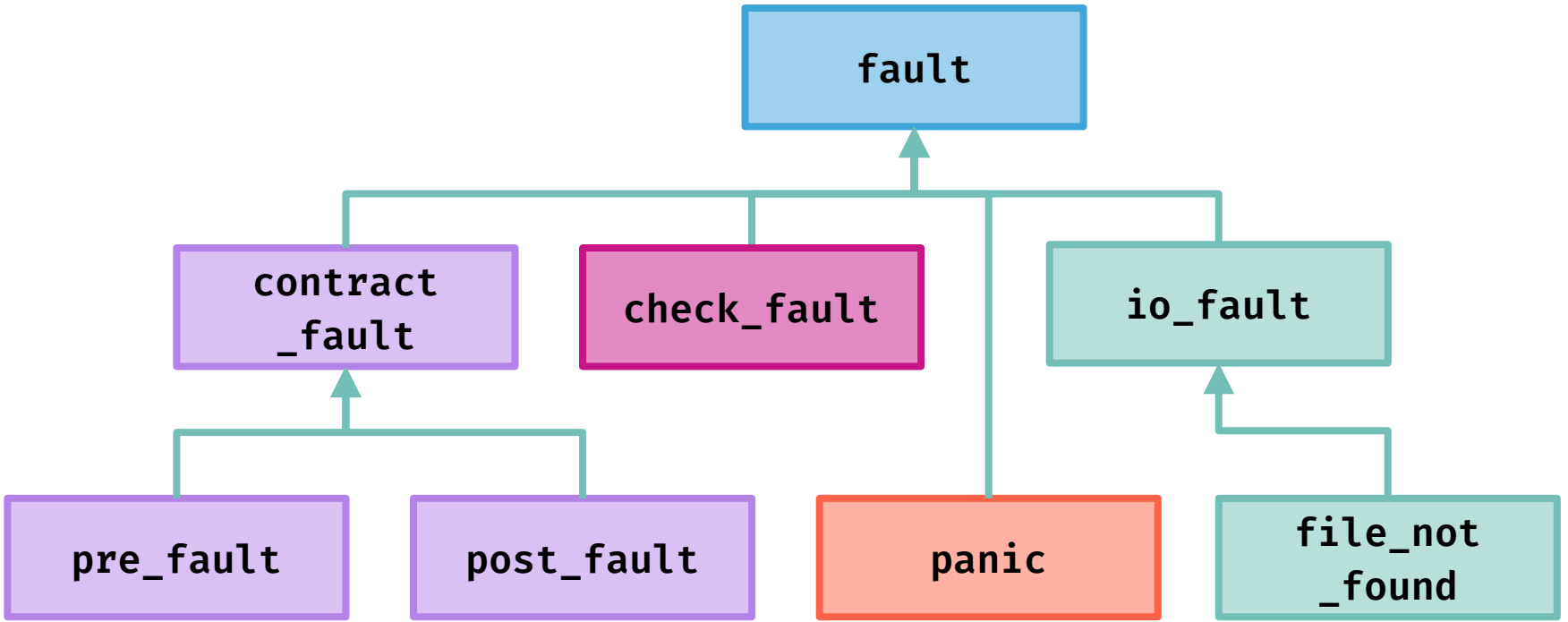
Fallible Hierarchy



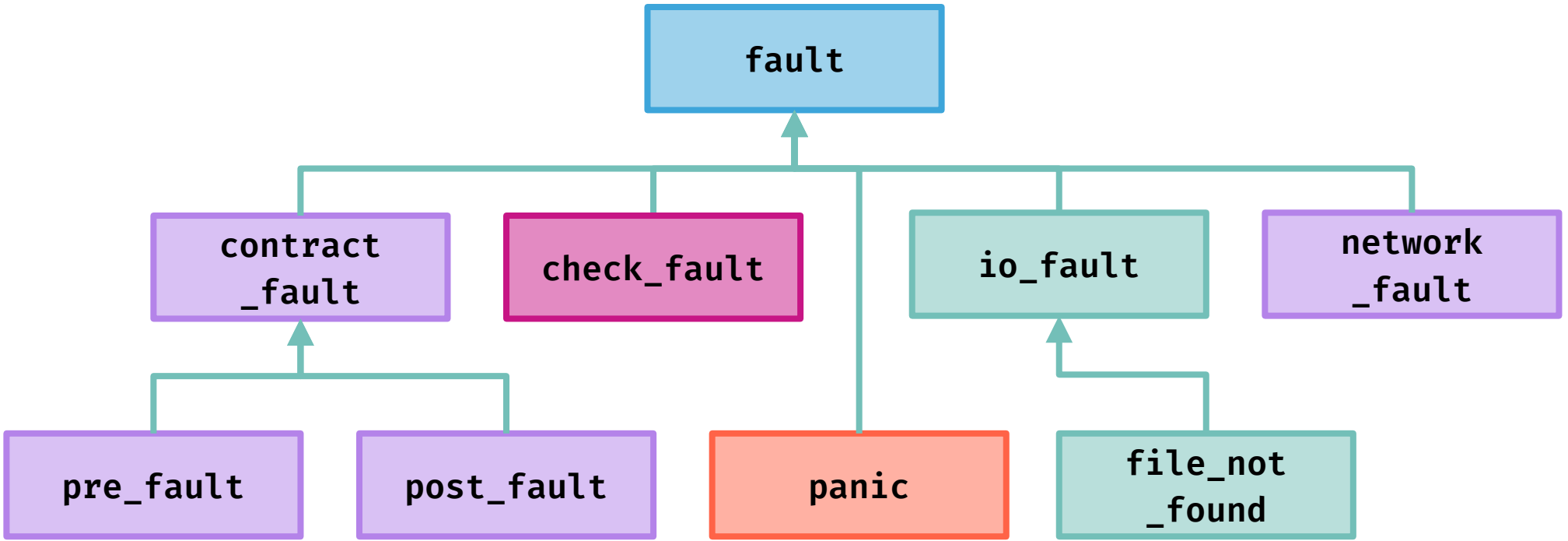
Fallible Hierarchy



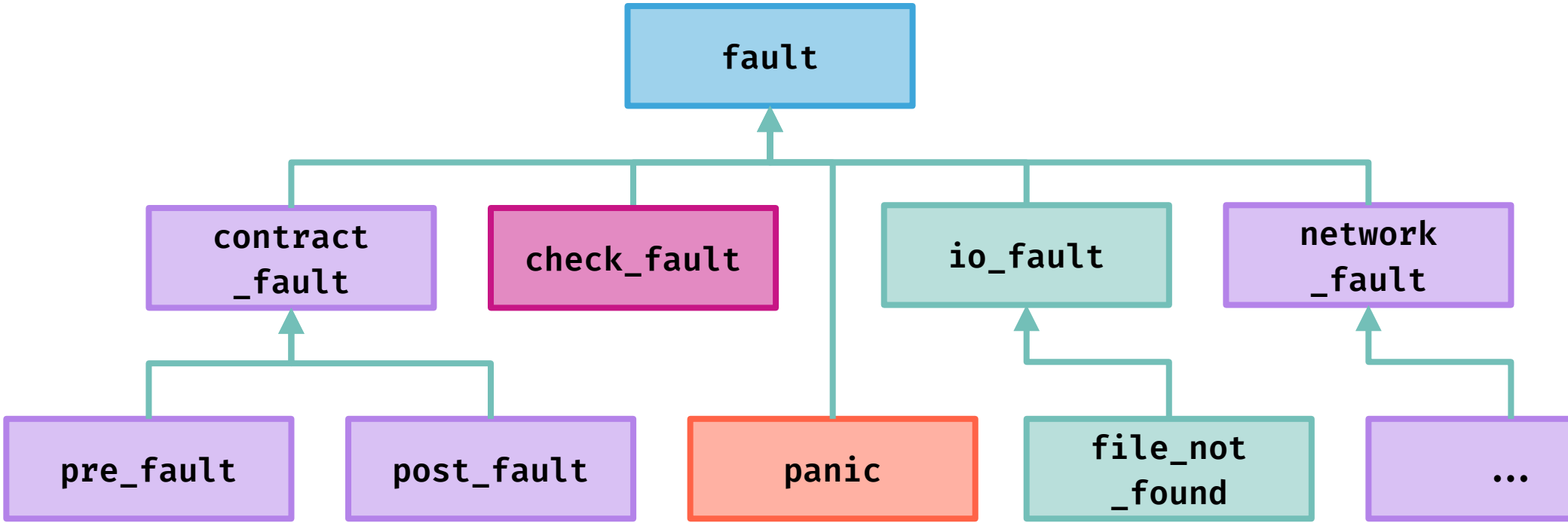
Fallible Hierarchy



Fallible Hierarchy



Fallible Hierarchy



Talk Overview

- Fuzion quick intro
- Example
- Design-by-Contract
- Effect Handlers
- Fallible Hierarchy
- Type Constraints
- Status

TOKIWA software

Type Constraints

```
square(x N : numeric)
  pre
    debug: x *! x
⇒
  "{x}² = {x*x}"

square 3.2E200 |> say
```

Type Constraints

```
square(x N : numeric)
  pre
    debug: x *! x
⇒
  "{x}² = {x*x}"

square 3.2E200 |> say
```

```
> fz square.fz
3.2E200² = Infinity
>
```


Type Constraints

```
square(x N : numeric)
  pre
    debug: x *! x
    debug: (if N : float
            then x*x ≠ N.infinity
            else true)
⇒
  "{x}² = {x*x}"

square 3.2E200 |> say
```

Type Constraints

```
square(x N : numeric)
  pre
    debug: x *! x
    debug: (if N : float
            then x*x ≠ N.infinity
            else true)
⇒
  "{x}² = {x*x}"

square 3.2E200 |> say
```

```
> fz square.fz

error 1: FATAL FAULT
`precondition`: debug:
       (if N : float
        then x*x ≠ N.infinity
        else true)
Call stack:
 ...
```


TOKIWA software

Talk Overview

- Fuzion quick intro
- Example
- Design-by-Contract
- Effect Handlers
- Fallible Hierarchy
- Type Constraints
- Status

TOKIWA software

Fuzion: Status

Fuzion still early prototype

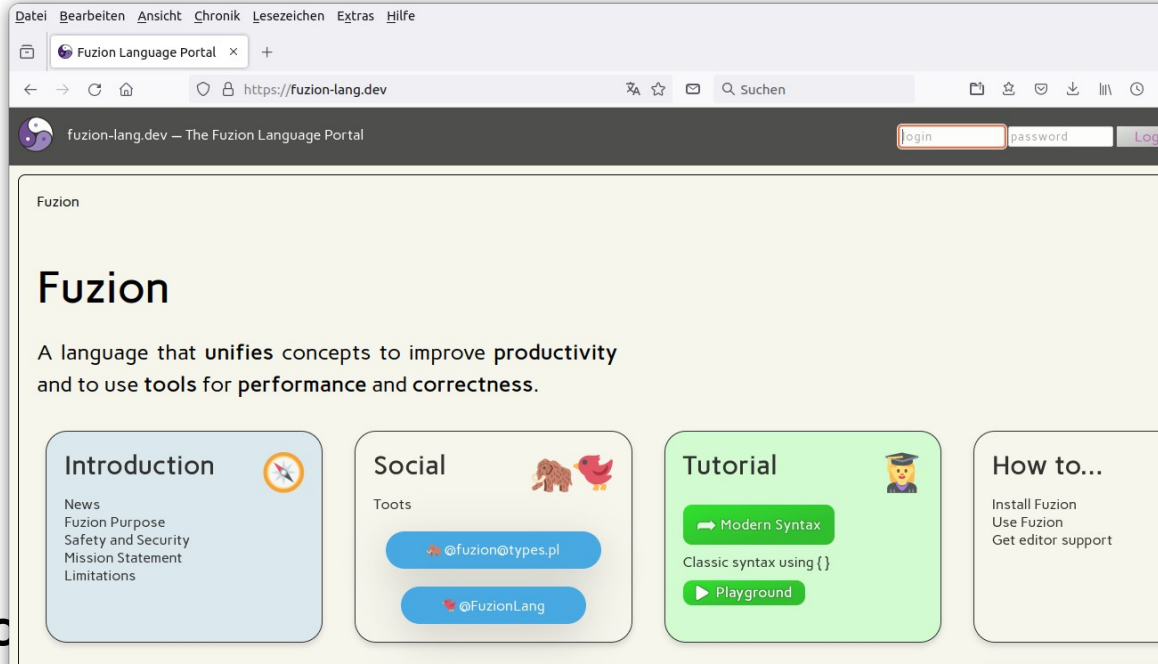
- ➡ language definition slowly getting more stable
- ➡ JVM and C backends and FFIs
- ➡ basic libraries
- ➡ static analysis tools
- ➡ first application being developed
- ➡ part of greencode.ai

TOKIWA software

Thank you. Any questions?

Please follow and stay informed

- github.com/tokiwa-software/fuzion
- web https://fuzion-lang.dev
- @fuzionlang@bsky.social
- @Fuzion@types.pl



TOKIWA software