Understanding your monad

an intuitive approach to monads

Stephan Roslen, 2025-02-05

Motivation

What is a monad?

In addition to it being good and useful, it is also cursed and the curse of the monad is that once you get the epiphany, once you understand - "oh that's what it is" - you lose the ability to explain it to anybody else.

Douglas Crockford, YUIConf 2012 Evening Keynote

Why?

How to solve it?

Foundations

Functional Programming

Functions

- Maps input to output
  - $f :: int \rightarrow int$
  - $f\,x = 23 + x$
- No side effects, same input => same output, no side channels

Type theory

Basic Types

- $bool \equiv 2$
- $char \equiv 2^8 = 256$
- $unit\ (void) \equiv 1$
- $actual\ void \equiv 0$

Type theory

Calculating with types

- $variant < a, b > = a \,|\, b \equiv a + b$
- $optional < a > = Maybe\ a = variant < a, unit > = a \,|\, unit \equiv a + 1$
- $tuple < a, b > = (a, b) \equiv a * b$
- $list < a > = () \,|\, (a) \,|\, (a, a) \,|\, \cdots \equiv 1 + a + a^2 + \cdots + a^\infty = \sum_{n=0}^{\infty} a^n$

Type theory

Functions

- $a :: b \to e$
- $a \equiv e^b$

b :: bool	**e :: int**
false	23
true	42

Type theory

Multi Argument Functions

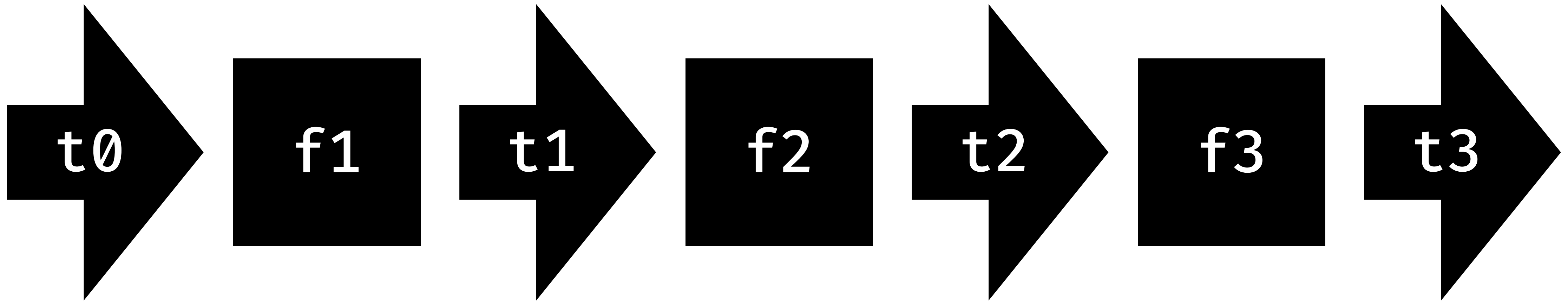
- $a :: b \rightarrow c \rightarrow e$
- $a :: (b, c) \rightarrow e$
- $\equiv e^{(b*c)}$
- $= (e^b)^c$
- $\Rightarrow a(b, c) = (a\ b)\ c$
- $a :: b \rightarrow (c \rightarrow e)$

(b,c) :: (bool,bool)	e :: int
(false,false)	23
(false,true)	42
(true,false)	93
(true,true)	1337

Functional Programming

Function composition

- Composition
  - $f_3(f_2(f_1 x))$
  - $f = f_3 \cdot f_2 \cdot f_1$
- $f_n :: t_{n-1} \rightarrow t_n$



Types

Type Mapping

- $F :: a \to t\,a$
- $Maybe\ a = Just\ a \,|\, Nothing \equiv a + 1$
- $List\ a = () \,|\, (a) \,|\, (a, a) \,|\, \cdots \equiv \sum_{n=0}^{\infty} a^n$

Monads

Monads

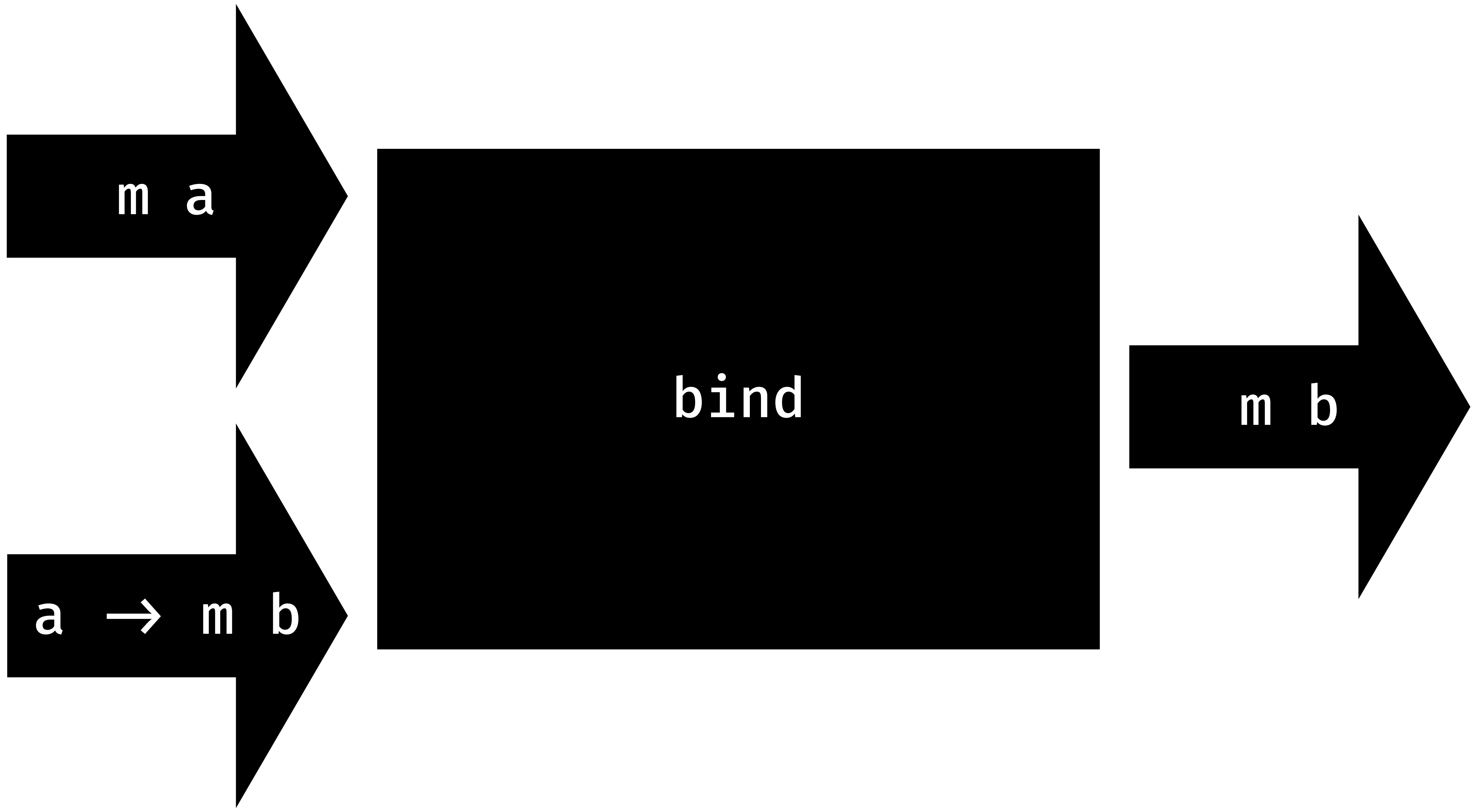
Type mapping

- $m\ a$
- $return :: a \rightarrow m\ a$

Monads

bind

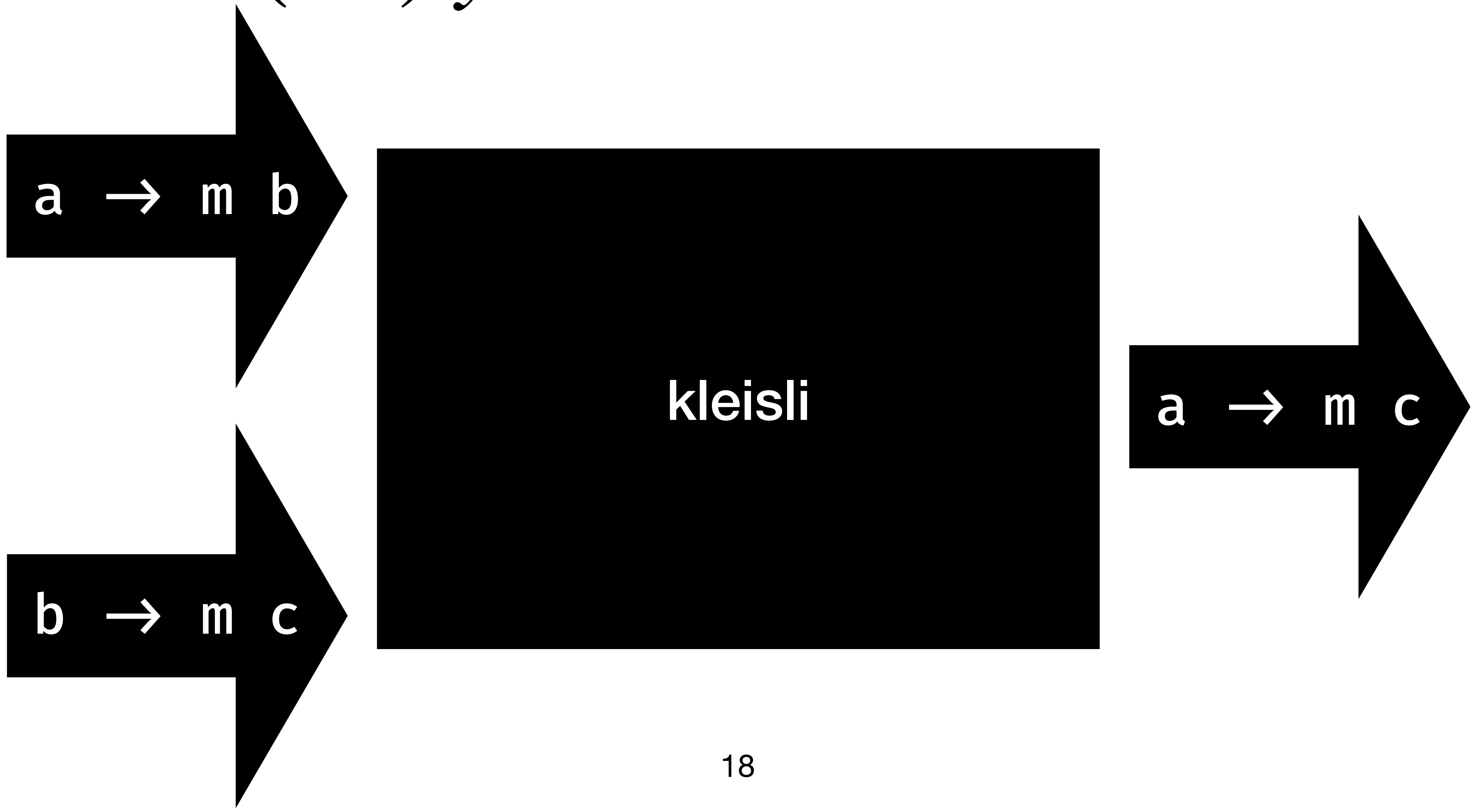
- $bind :: m\ a \rightarrow (a \rightarrow m\ b) \rightarrow m\ b$
  - also ≫= operator



Monads

Kleisli operator

- $kleisli :: (a \to m\, b) \to (b \to m\, c) \to (a \to m\, c)$
  - also `>=>` operator
  - $(kleisli\ x\ y)\ z = bind\ (x\ z)\ y$



Monads

From Bind to Kleisli in C++

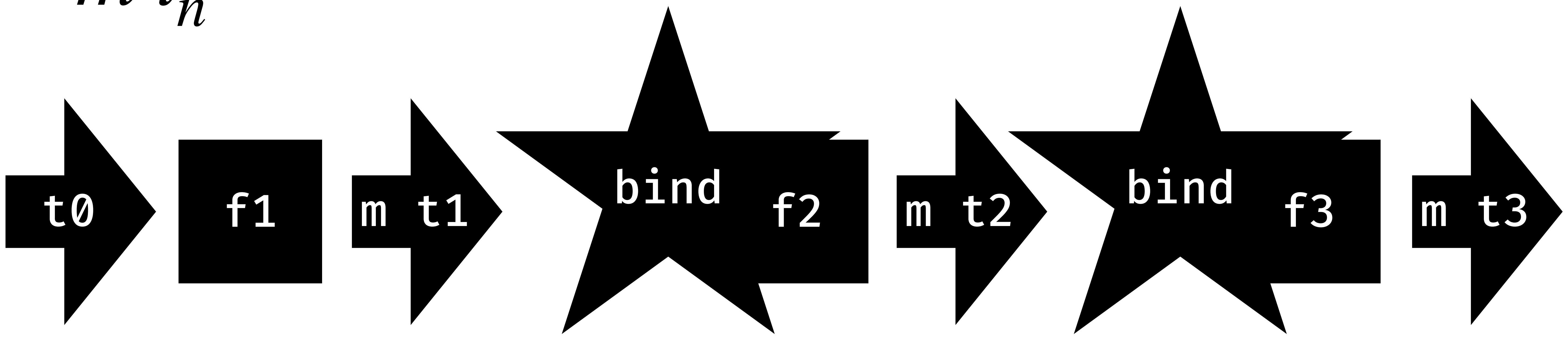
$$(kleisli\ x\ y)\ z = bind\ (x\ z)\ y$$

```
template<typename MorphismA, typename MorphismB>
static auto mkleisli(MorphismA&& morphisma_, MorphismB&& morphismb_) {
  return [morphisma = std::forward<MorphismA>(morphisma_),
          morphismb = std::forward<MorphismB>(morphismb_)]<typename A>(A&& a) {
    return MonadBase::mbind(std::move(morphisma)(std::forward<A>(a)), std::move(morphismb));
  };
}
```

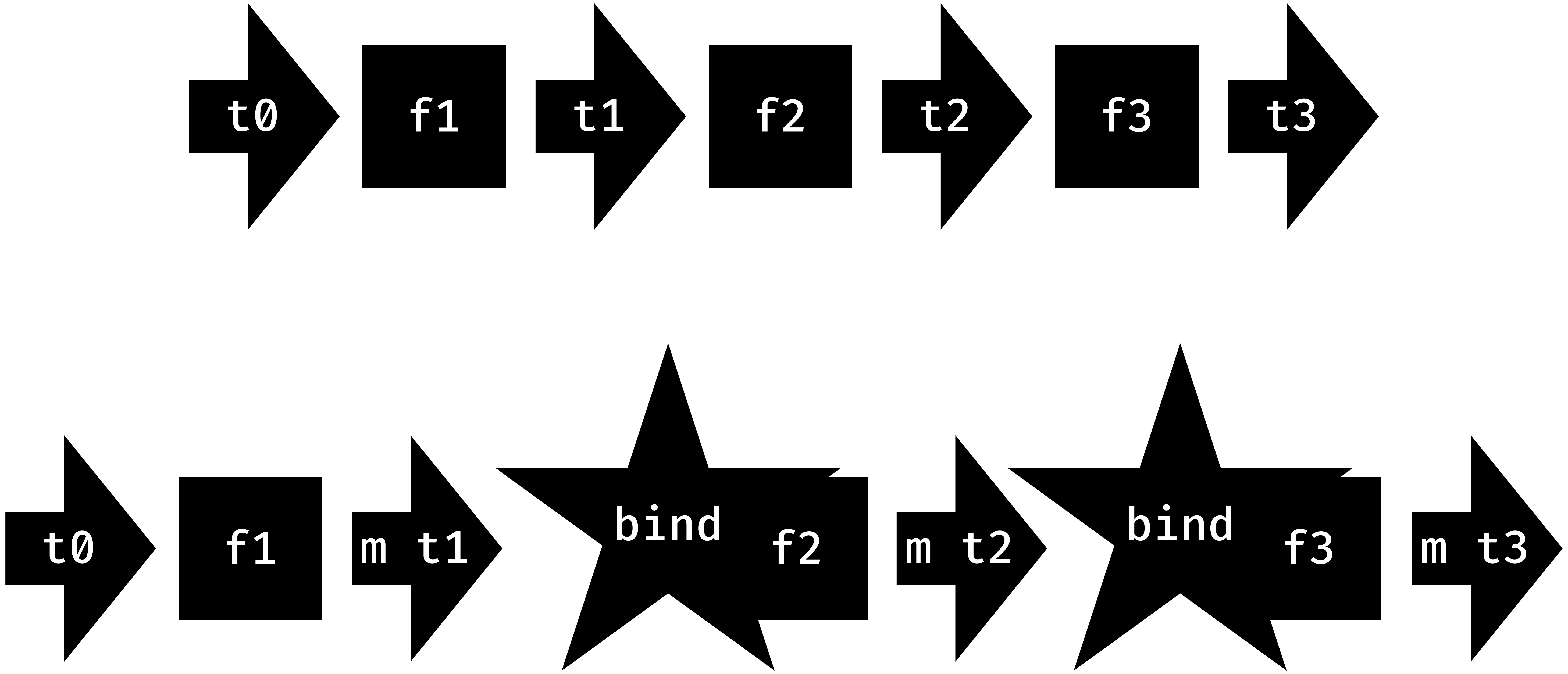
Monads

Composition

- $bind\ (bind\ (x\ f_1)\ f_2)\ f_3$
  - $f = f_3 \implies f_2 \implies f_1$
  - $x :: m\ t_{n-1}$
  - $f_n :: t_{n-1} \rightarrow m\ t_n$



Composition compared



Example: Maybe Monad

Maybe Monad

An Example

- $m\ a = Nothing\,|\,Just\ a$

Maybe Monad

Return function

- $return :: a \rightarrow m\ a$
- $return\ a = Just\ a$

Maybe Monad

Kleisli Operator >=>

- $kleisli :: (a \to m\ b) \to (b \to m\ c) \to (a \to m\ c)$
- $= (a \to Nothing\,|\,Just\ b) \to (b \to Nothing\,|\,Just\ c) \to (a \to Nothing\,|\,Just\ c)$
- Or just $(kleisli\ x\ y)\ z = bind\ (x\ z)\ y$

Maybe Monad

Bind Operator >>= in C++

$$Nothing\,|\,Just\ a \rightarrow (a \rightarrow Nothing\,|\,Just\ b) \rightarrow Nothing\,|\,Just\ b$$

```
template<typename MA, typename Morphism>
static constexpr MBindResultType<MaybeMonadBase, MA, Morphism> mbind(MA&& ma,
    Morphism&& morphism) noexcept(noexcept(morphism(*ma.data))) {
  if (!ma.data) {
    return nothing<InverseTypeConstructor<MBindResultType<MaybeMonadBase, MA, Morphism>>>();
  }
  return morphism(*ma.data);
}
```

Example: Error Monad

Error Monad

An Example

- $m\ a = Ok\ a \,|\, Error\ b$
- Behaves similar to the Maybe Monad
- Remember Kleisli Categories?

Example: State Monad

State Monad

An Example

- $(a, state) \to (b, state)$
- $= a \to state \to (b, state)$
- $= a \to (state \to (b, state))$
- $\Rightarrow m\ b = state \to (b, state)$
- $a \to m\ b$

State Monad

Return function

- $return :: a \rightarrow m\ a = a \rightarrow (state \rightarrow (a, state))$

State Monad

Return function in C++

$$a \to (state \to (a, state))$$

```
template<typename A>
static MReturnResultType<StateMonadBase, A> mreturn(A&& a_) {
  return MReturnResultType<StateMonadBase, A>{[a = std::forward<A>(a_)](StateDesc s) {
    return ResultStateDesc{std::move(a), std::move(s)};
  }};
}
```

State Monad

Kleisli Operator >=>

- $kleisli :: (a \to m\ b) \to (b \to m\ c) \to (a \to m\ c)$
- $= (a \to (state \to (b, state)) \to (b \to (state \to (c, state)) \to (a \to (state \to (c, state))$
- $= ((a, state) \to (b, state)) \to ((b, state) \to (c, state)) \to ((a, state) \to (c, state))$
- Or just $(kleisli\ x\ y)\ z = bind\ (x\ z)\ y$

State Monad

Bind Operator >>=

- $bind :: m\ a \rightarrow (a \rightarrow m\ b) \rightarrow m\ b$
- $= Nothing\,|\,Just\ a \rightarrow (a \rightarrow Nothing\,|\,Just\ b) \rightarrow Nothing\,|\,Just\ b$

State Monad

Bind Operator >>=

- $bind :: m\ a \to (a \to m\ b) \to m\ b$
- $= (state \to (a, state)) \to (a \to (state \to (b, state))) \to (state \to (b, state))$
- $= (state \to (a, state)) \to ((a, state) \to (b, state)) \to (state \to (b, state))$

State Monad

Bind Operator >>= in C++

$$(state \to (a, state)) \to (a \to (state \to (b, state))) \to (state \to (b, state))$$

```
template<typename MA, typename Morphism>
static MBindResultType<StateMonadBase, MA, Morphism> mbind(MA&& ma_, Morphism&& morphism_) {
  return MBindResultType<StateMonadBase, MA, Morphism>{[ma = std::forward<MA>(ma_), morphism
      = std::forward<Morphism>(morphism_)](StateDesc s) {
    auto rsd  = std::move(ma.data)(std::move(s));
    auto tmps = std::move(morphism)(std::move(rsd.result));
    return std::move(tmps.data)(std::move(rsd.stateDesc));
  }};
}
```

State Monad

What’s the Kleisli Category of the State Monad?

Summary and outlook

Thank you for your kind attention!

Questions?