Four More Domain Modeling Lenses

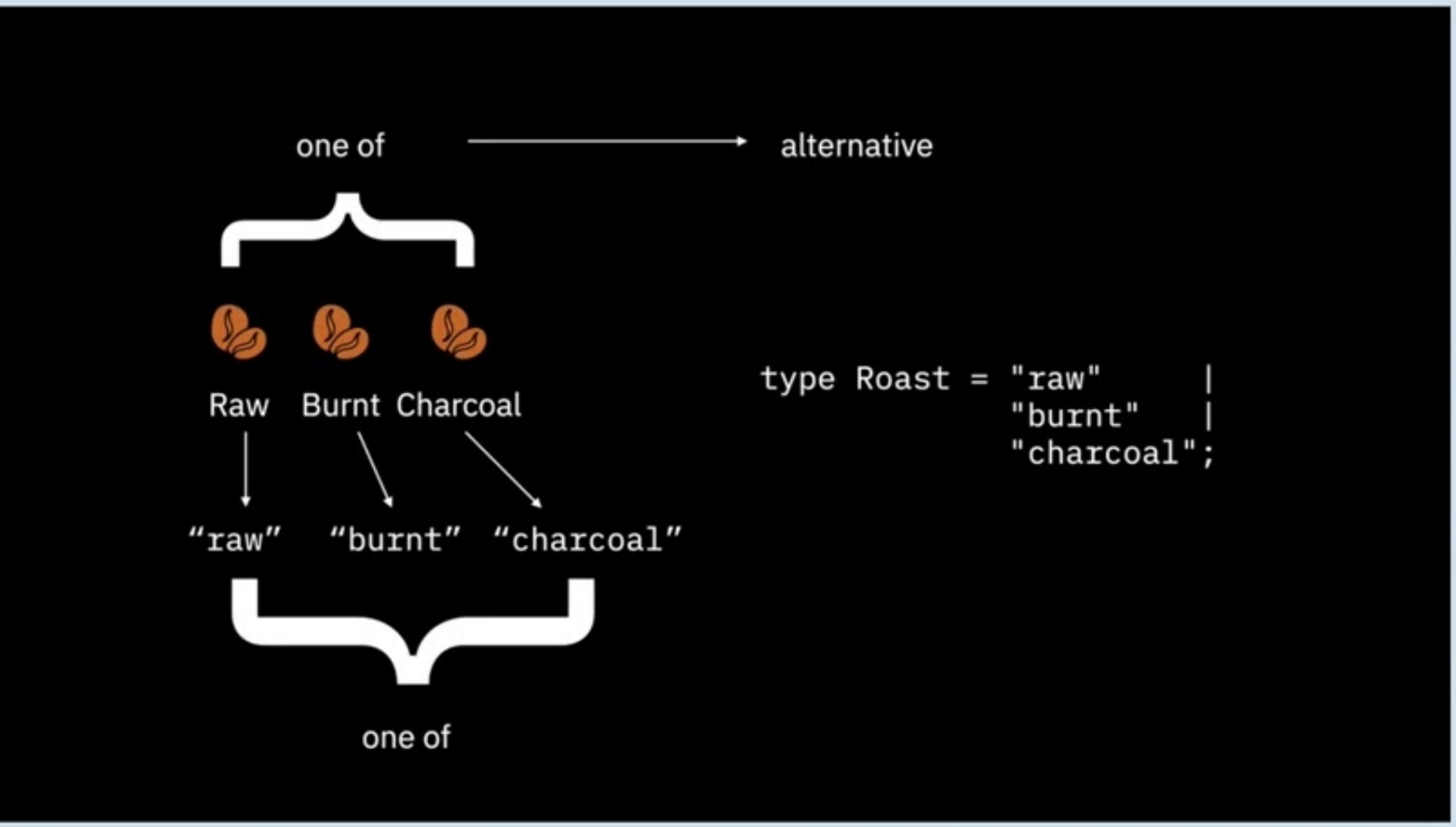
Eric Normand - Houston Functional Programming User Group

Software design is subtle

Good information → good decisions → good design

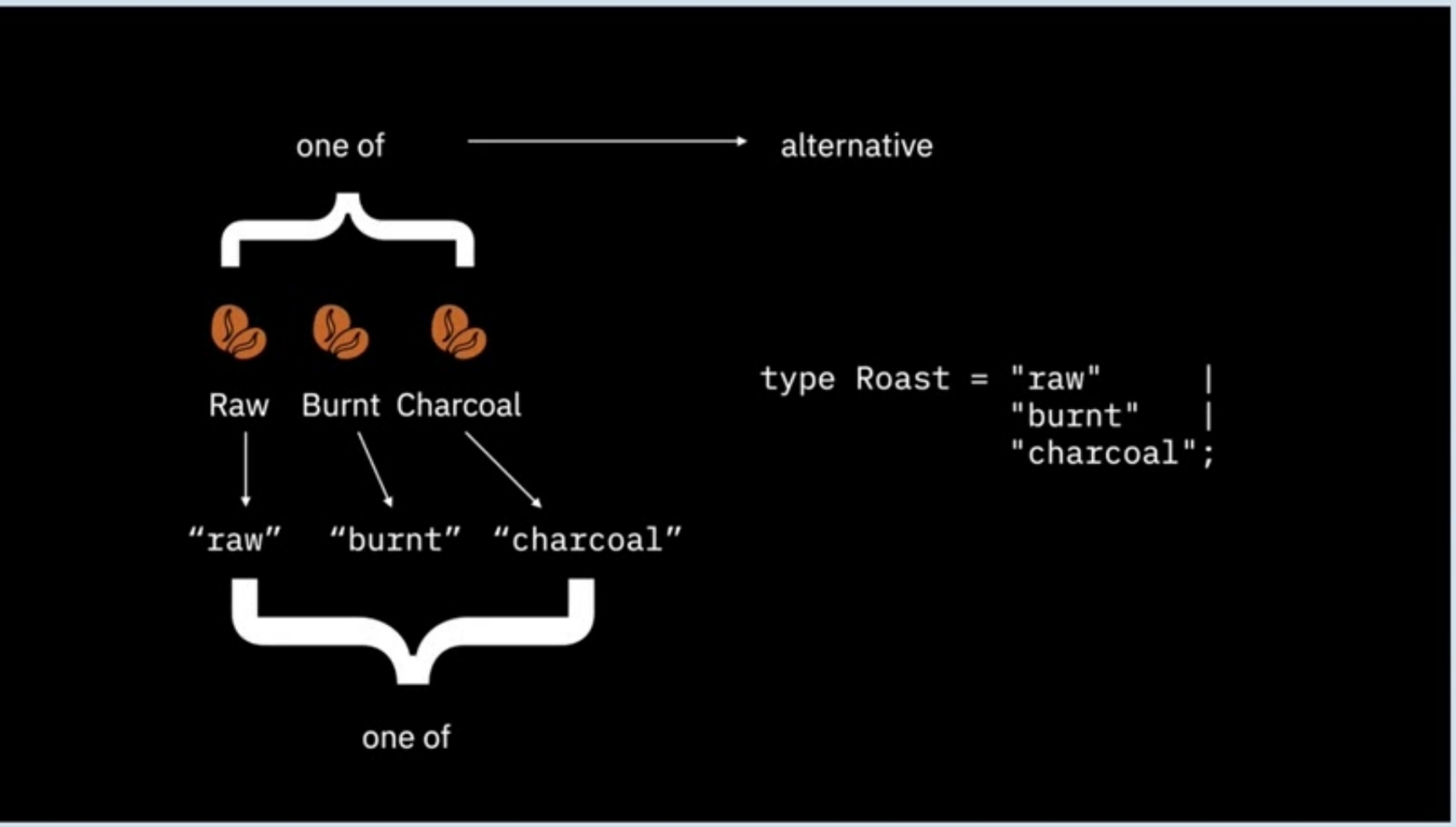
- Data
- Operations
- Composition
- Time
- Domain
- Scope
- Platform
- Volatility
- Runnable specifications

- Data
- Operations
- Composition
- Time
- Domain
- Scope
- Platform
- Volatility
- Runnable specifications



Better Software Design with Domain Modeling

https://ericnormand.me/speaking/func-prog-sweden-2023



Better Software Design with Domain Modeling

https://ericnormand.me/speaking/func-prog-sweden-2023

- Data
- Operations
- Composition
- Time
- Domain
- Scope
- Platform
- Volatility
- Runnable specifications

- Data
- Operations
- Composition
- Time
- Domain
- Scope
- Platform
- Volatility
- Runnable specifications

Super Mega Galactic

Super Mega Galactic

Raw Burnt Charcoal

Super Mega Galactic

Raw Burnt Charcoal

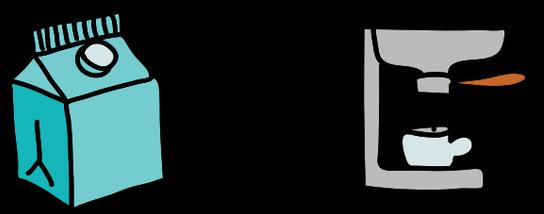
Soy milk

Espresso

Hazelnut Chocolate Almond

Super Mega Galactic

Raw Burnt Charcoal



Soy milk Espresso

Hazelnut Chocolate Almond

```
{
  “size”: “super”,

  “roast”: “burnt”,

  “add-ins”: {“espresso” : 1,
              “soy”      : 2}
}
```

Composition

Composition is two or more calls to operations working together.

```
let coffee = newCoffee();
coffee = setSize(coffee, “galactic”);
coffee = setSize(coffee, “mega”);
coffee = setRoast(coffee, “burnt”);
coffee = addAddIn(coffee, “soy”);
coffee = removeAddIn(coffee, “almond”);
```

Example-based tests

Example-based tests

```
let coffee = newCoffee();
coffee = setSize(coffee, “galactic”);
assert(coffee.size === “galactic”);
```

Example-based tests

```
let coffee = newCoffee();
coffee = setSize(coffee, “galactic”);
assert(coffee.size === “galactic”);
```

```
let coffee = newCoffee();
coffee = setRoast(coffee, “charcoal”);
assert(coffee.roast === “charcoal”);
```

Example-based tests

```
let coffee = newCoffee();
coffee = setSize(coffee, “galactic”);
assert(coffee.size === “galactic”);
```

```
let coffee = newCoffee();
coffee = setRoast(coffee, “charcoal”);
assert(coffee.roast === “charcoal”);
```

BORING!

Look for relationships between operations.

Look for relationships between operations.

```
let coffee = newCoffee();
```

Look for relationships between operations.

```
let coffee = newCoffee();
let addIns1 = [“soy”, “almond”, “espresso”];
```

Look for relationships between operations.

```
let coffee = newCoffee();
let addIns1 = [“soy”, “almond”, “espresso”];
let addIns2 = [“almond”, “espresso”, “soy”];
```

Look for relationships between operations.

```
let coffee = newCoffee();
let addIns1 = [“soy”, “almond”, “espresso”];
let addIns2 = [“almond”, “espresso”, “soy”];
let coffee1 = addIns1.reduce(addAddIn, coffee, addIns1);
```

Look for relationships between operations.

```
let coffee = newCoffee();
let addIns1 = [“soy”, “almond”, “espresso”];
let addIns2 = [“almond”, “espresso”, “soy”];
let coffee1 = addIns1.reduce(addAddIn, coffee, addIns1);
let coffee2 = addIns2.reduce(addAddIn, coffee, addIns2);
```

Look for relationships between operations.

```
let coffee = newCoffee();
let addIns1 = [“soy”, “almond”, “espresso”];
let addIns2 = [“almond”, “espresso”, “soy”];
let coffee1 = addIns1.reduce(addAddIn, coffee, addIns1);
let coffee2 = addIns2.reduce(addAddIn, coffee, addIns2);
assert(sameCoffee(coffee1, coffee2));
```

Look for relationships between operations.

```
let coffee = anyCoffee();
let addIns1 = [“soy”, “almond”, “espresso”];
let addIns2 = [“almond”, “espresso”, “soy”];
let coffee1 = addIns1.reduce(addAddIn, coffee, addIns1);
let coffee2 = addIns2.reduce(addAddIn, coffee, addIns2);
assert(sameCoffee(coffee1, coffee2));
```

Look for relationships between operations.

```
let coffee = anyCoffee();
let addIns1 = arrayOf(anyAddIn);
let addIns2 = [“almond”, “espresso”, “soy”];
let coffee1 = addIns1.reduce(addAddIn, coffee, addIns1);
let coffee2 = addIns2.reduce(addAddIn, coffee, addIns2);
assert(sameCoffee(coffee1, coffee2));
```

Look for relationships between operations.

```
let coffee = anyCoffee();
let addIns1 = arrayOf(anyAddIn);
let addIns2 = anyOrderOf(addIns1);
let coffee1 = addIns1.reduce(addAddIn, coffee, addIns1);
let coffee2 = addIns2.reduce(addAddIn, coffee, addIns2);
assert(sameCoffee(coffee1, coffee2));
```

Look for relationships between operations.

```
for(let i = 0; i < 100; i++) {
  let coffee = anyCoffee();
  let addIns1 = arrayOf(anyAddIn);
  let addIns2 = anyOrderOf(addIns1);
  let coffee1 = addIns1.reduce(addAddIn, coffee, addIns1);
  let coffee2 = addIns2.reduce(addAddIn, coffee, addIns2);
  assert(sameCoffee(coffee1, coffee2));
}
```

Look for relationships between operations.

```
for(let i = 0; i < 100; i++) {
  let coffee = anyCoffee();
  let addIns1 = arrayOf(anyAddIn);
  let addIns2 = anyOrderOf(addIns1);
  let coffee1 = addIns1.reduce(addAddIn, coffee
                                          ns1);
  let coffee2 = addIns2.reduce(addAddIn, coffee
                                          ns2);
  assert(sameCoffee(coffee1, coffee2));
}
```

Add-ins can be added in any order.

```
let coffee = newCoffee();
let addIns1 = [“soy”, “almond”, “espresso”];
let addIns2 = [“almond”, “espresso”, “soy”];
let coffee1 = addIns1.reduce(addAddIn, coffee, addIns1);
let coffee2 = addIns2.reduce(addAddIn, coffee, addIns2);
assert(sameCoffee(coffee1, coffee2));
```

```
let coffee = newCoffee();
let addIns1 = [“soy”, “almond”, “espresso”];
let addIns2 = [“almond”, “espresso”, “soy”];
let coffee1 = addIns1.reduce(addAddIn, coffee, addIns1);
let coffee2 = addIns2.reduce(addAddIn, coffee, addIns2);
assert(sameCoffee(coffee1, coffee2));
```

```
for(let i = 0; i < 100; i++) {
  let coffee = anyCoffee();
  let addIns1 = arrayOf(anyAddIn);
  let addIns2 = anyOrderOf(addIns1);
  let coffee1 = addIns1.reduce(addAddIn, coffee, addIns1);
  let coffee2 = addIns2.reduce(addAddIn, coffee, addIns2);
  assert(sameCoffee(coffee1, coffee2));
}
```

```
let coffee = newCoffee();
let addIns1 = [“soy”, “almond”, “espresso”];
let addIns2 = [“almond”, “espresso”, “soy”];
let coffee1 = addIns1.reduce(addAddIn, coffee, addIns1);
let coffee2 = addIns2.reduce(addAddIn, coffee, addIns2);
assert(sameCoffee(coffee1, coffee2));
```

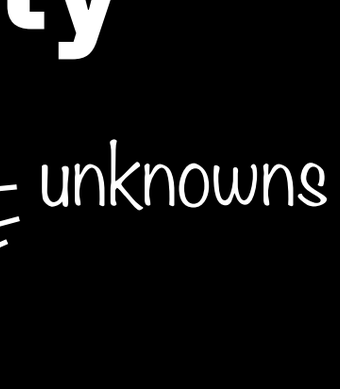
```
for(let i = 0; i < 100; i++) {
  let coffee = anyCoffee();
  let addIns1 = arrayOf(anyAddIn);
  let addIns2 = anyOrderOf(addIns1);
  let coffee1 = addIns1.reduce(addAddIn, coffee, addIns1);
  let coffee2 = addIns2.reduce(addAddIn, coffee, addIns2);
  assert(sameCoffee(coffee1, coffee2));
}
```

unknowns

```
let coffee = newCoffee();
let addIns1 = [“soy”, “almond”, “espresso”];
let addIns2 = [“almond”, “espresso”, “soy”];
let coffee1 = addIns1.reduce(addAddIn, coffee, addIns1);
let coffee2 = addIns2.reduce(addAddIn, coffee, addIns2);
assert(sameCoffee(coffee1, coffee2));
```

Algebraic property

```
for(let i = 0; i < 100; i++) {
  let coffee = anyCoffee();
  let addIns1 = arrayOf(anyAddIn);
  let addIns2 = anyOrderOf(addIns1);
  let coffee1 = addIns1.reduce(addAddIn, coffee, addIns1);
  let coffee2 = addIns2.reduce(addAddIn, coffee, addIns2);
  assert(sameCoffee(coffee1, coffee2));
}
```



Commutativity

Order of function calls doesn’t matter

```
for(let i = 0; i < 100; i++) {
  let coffee = anyCoffee();
  let addIns1 = arrayOf(anyAddIn);
  let addIns2 = anyOrderOf(addIns1);
  let coffee1 = addIns1.reduce(addAddIn, coffee, addIns1);
  let coffee2 = addIns2.reduce(addAddIn, coffee, addIns2);
  assert(sameCoffee(coffee1, coffee2));
}
```

Commutativity

Order of function calls doesn’t matter

Commutativity

Order of function calls doesn’t matter

```
for(let i = 0; i < 100; i++) {
```

Commutativity

Order of function calls doesn’t matter

```
for(let i = 0; i < 100; i++) {
  let coffee = anyCoffee();
```

Commutativity

Order of function calls doesn’t matter

```
for(let i = 0; i < 100; i++) {
  let coffee = anyCoffee();
  let addInA = anyAddIn();
```

Commutativity

Order of function calls doesn’t matter

```
for(let i = 0; i < 100; i++) {
  let coffee = anyCoffee();
  let addInA = anyAddIn();
  let addInB = anyAddIn();
```

Commutativity

Order of function calls doesn’t matter

```
for(let i = 0; i < 100; i++) {
  let coffee = anyCoffee();
  let addInA = anyAddIn();
  let addInB = anyAddIn();
  assert(sameCoffee(
```

Commutativity

Order of function calls doesn’t matter

```
for(let i = 0; i < 100; i++) {
  let coffee = anyCoffee();
  let addInA = anyAddIn();
  let addInB = anyAddIn();
  assert(sameCoffee(
    coffee.add(addInA).add(AddInB),
```

Commutativity

Order of function calls doesn’t matter

```
for(let i = 0; i < 100; i++) {
  let coffee = anyCoffee();
  let addInA = anyAddIn();
  let addInB = anyAddIn();
  assert(sameCoffee(
    coffee.add(addInA).add(AddInB),
    coffee.add(addInB).add(AddInA)
```

Commutativity

Order of function calls doesn’t matter

```
for(let i = 0; i < 100; i++) {
  let coffee = anyCoffee();
  let addInA = anyAddIn();
  let addInB = anyAddIn();
  assert(sameCoffee(
    coffee.add(addInA).add(AddInB),
    coffee.add(addInB).add(AddInA)
  ));
```

Commutativity

Order of function calls doesn’t matter

```
for(let i = 0; i < 100; i++) {
  let coffee = anyCoffee();
  let addInA = anyAddIn();
  let addInB = anyAddIn();
  assert(sameCoffee(
    coffee.add(addInA).add(AddInB),
    coffee.add(addInB).add(AddInA)
  ));
}
```

Commutativity

Order of function calls doesn’t matter

```
for(let i = 0; i < 100; i++) {
  let coffee = anyCoffee();
  let addInA = anyAddIn();
  let addInB = anyAddIn();
  assert(sameCoffee(
    coffee.add(addInA).add(AddInB),
    coffee.add(addInB).add(AddInA)
  ));
}
```

```
g(f(a)) = f(g(a))
```

Commutativity

Order of arguments doesn’t matter

Commutativity

Order of arguments doesn’t matter

```
{espresso: 1}
```

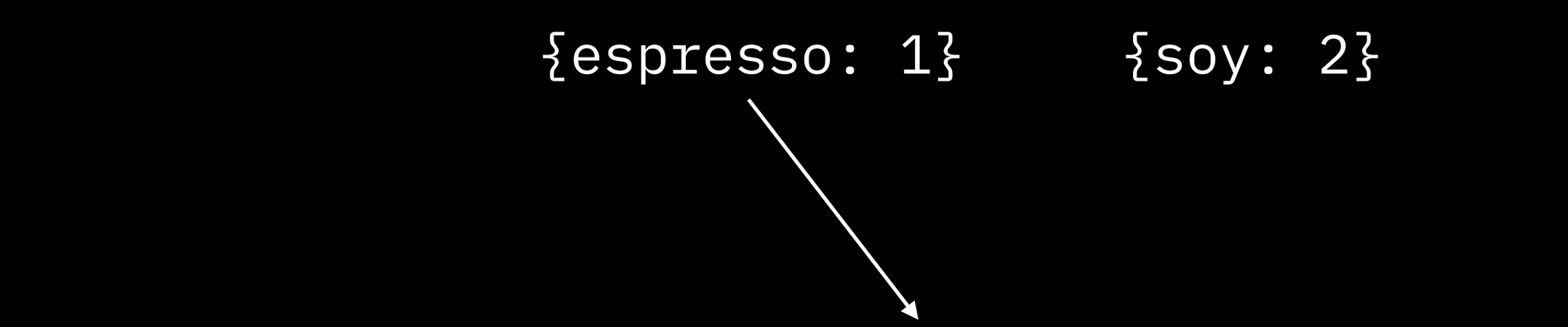
Commutativity

Order of arguments doesn’t matter

```
{espresso: 1}     {soy: 2}
```

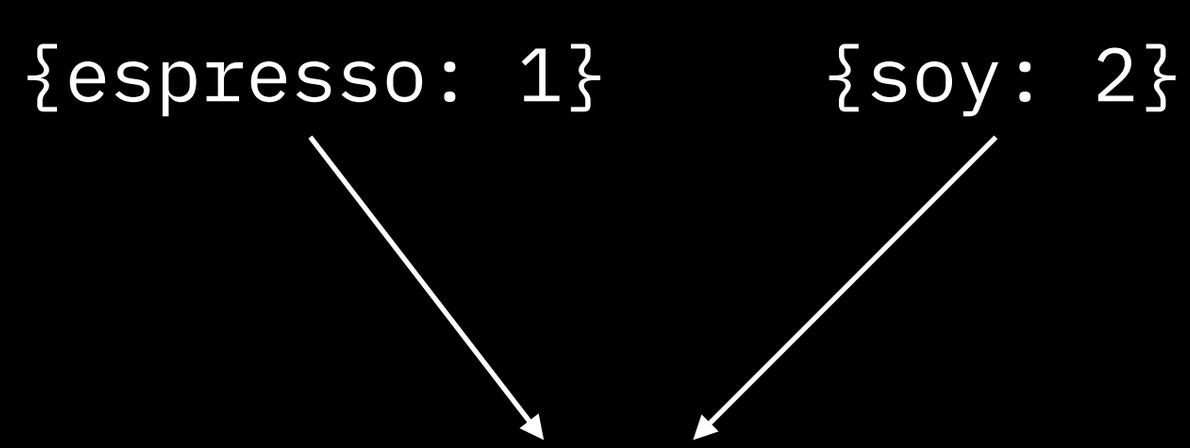
Commutativity

Order of arguments doesn’t matter



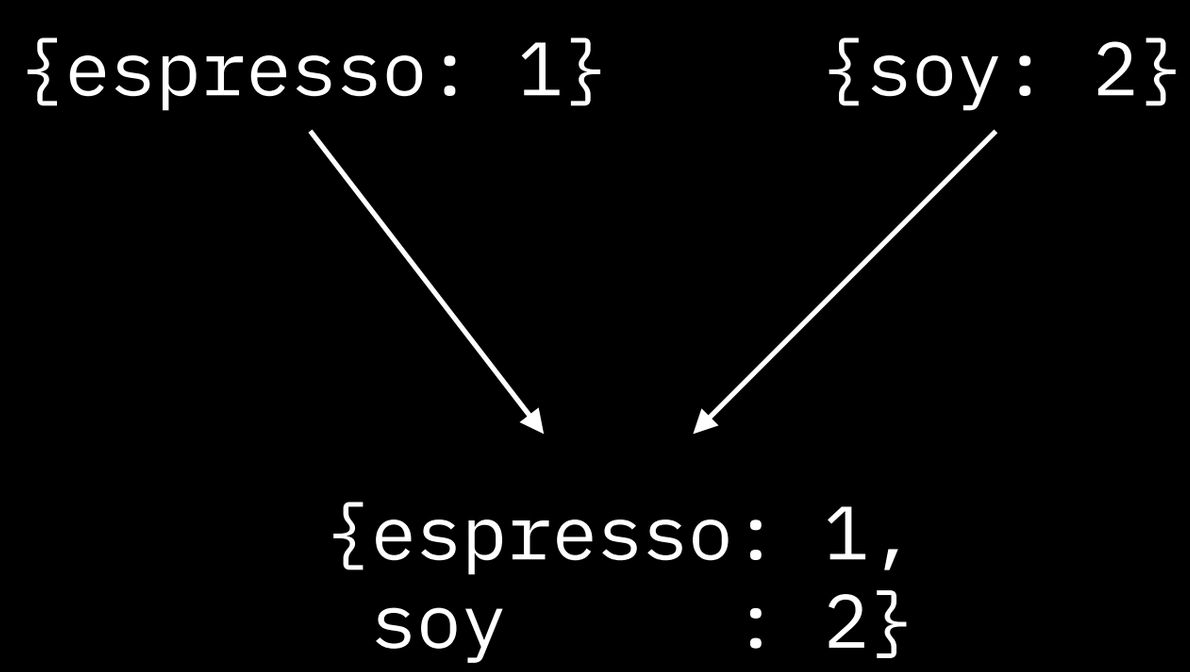
Commutativity

Order of arguments doesn’t matter



Commutativity

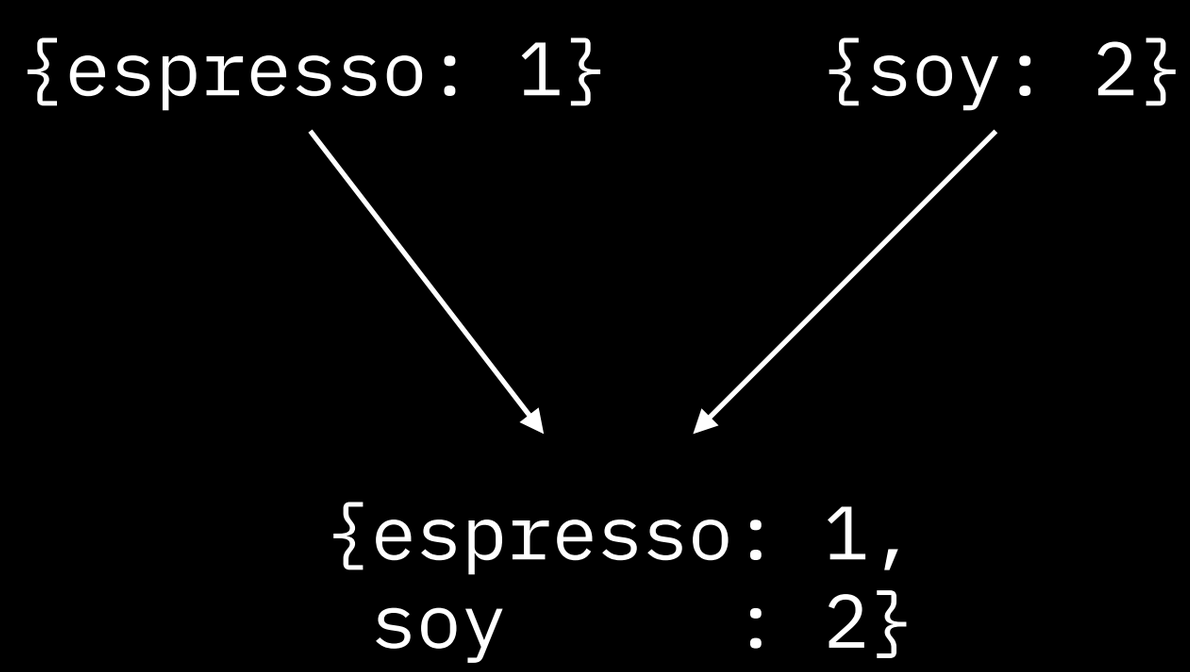
Order of arguments doesn’t matter



Commutativity

Order of arguments doesn't matter

```
let addInsA = arrayOf(anyAddIn);
let addInsB = arrayOf(anyAddIn);
assert(sameAddIns(
  combineAddIns(addInsA, addInsB),
  combineAddIns(addInsB, addInsA)
));
```

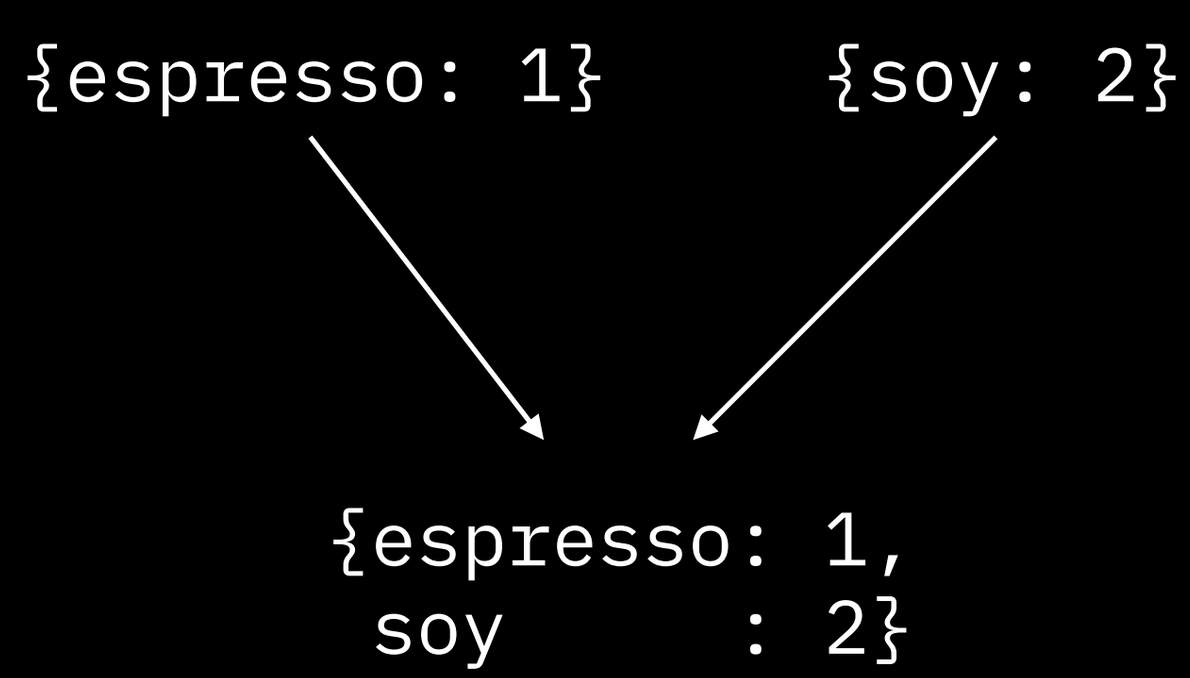


Commutativity

Order of arguments doesn't matter

```
let addInsA = arrayOf(anyAddIn);
let addInsB = arrayOf(anyAddIn);
assert(sameAddIns(
  combineAddIns(addInsA, addInsB),
  combineAddIns(addInsB, addInsA)
));
```

$$f(a, b) = f(b, a)$$



Idempotence

Doing a thing twice is the same as doing it once.

Idempotence

Doing a thing twice is the same as doing it once.

```
let coffee = anyCoffee();
```

Idempotence

Doing a thing twice is the same as doing it once.

```
let coffee = anyCoffee();
let size   = anySize();
```

Idempotence

Doing a thing twice is the same as doing it once.

```
let coffee = anyCoffee();
let size   = anySize();
assert(sameCoffee(
```

Idempotence

Doing a thing twice is the same as doing it once.

```
let coffee = anyCoffee();
let size   = anySize();
assert(sameCoffee(
  setSize(coffee, size),
```

Idempotence

Doing a thing twice is the same as doing it once.

```
let coffee = anyCoffee();
let size   = anySize();
assert(sameCoffee(
  setSize(coffee, size),
  setSize(setSize(coffee, size), size)
```

Idempotence

Doing a thing twice is the same as doing it once.

```
let coffee = anyCoffee();
let size   = anySize();
assert(sameCoffee(
  setSize(coffee, size),
  setSize(setSize(coffee, size), size)
));
```

Idempotence

Doing a thing twice is the same as doing it once.

```
let coffee = anyCoffee();
let size   = anySize();
assert(sameCoffee(
  setSize(coffee, size),
  setSize(setSize(coffee, size), size)
));
```

$$f(a) = f(f(a))$$

Idempotence

Doing a thing twice is the same as doing it once.

```
let coffee = anyCoffee();
let size   = anySize();
assert(sameCoffee(
  coffee.setSize(size),
  coffee.setSize(size).setSize(size)
));
```

$$f(a) = f(f(a))$$

Associative

Order of operations doesn’t matter

Associative

Order of operations doesn’t matter

```
let a = anyAddIns();
```

Associative

Order of operations doesn’t matter

```
let a = anyAddIns();
let b = anyAddIns();
```

Associative

Order of operations doesn’t matter

```
let a = anyAddIns();
let b = anyAddIns();
let c = anyAddIns();
```

Associative

Order of operations doesn’t matter

```
let a = anyAddIns();
let b = anyAddIns();
let c = anyAddIns();
assert(sameAddIns(
```

Associative

Order of operations doesn’t matter

```
let a = anyAddIns();
let b = anyAddIns();
let c = anyAddIns();
assert(sameAddIns(
  a.combine(b).combine(c),
```

Associative

Order of operations doesn’t matter

```
let a = anyAddIns();
let b = anyAddIns();
let c = anyAddIns();
assert(sameAddIns(
  a.combine(b).combine(c),
  a.combine(b.combine(c))
```

Associative

Order of operations doesn’t matter

```
let a = anyAddIns();
let b = anyAddIns();
let c = anyAddIns();
assert(sameAddIns(
  a.combine(b).combine(c),
  a.combine(b.combine(c))
));
```

Associative

Order of operations doesn’t matter

```
let a = anyAddIns();
let b = anyAddIns();
let c = anyAddIns();
assert(sameAddIns(
  a.combine(b).combine(c),
  a.combine(b.combine(c))
));
```

$$(a \cdot b) \cdot c = a \cdot (b \cdot c)$$

Inverse

You can undo an operation

Inverse

You can undo an operation

```
let coffee = anyCoffee();
```

Inverse

You can undo an operation

```
let coffee = anyCoffee();
let addIn  = anyAddIn();
```

Inverse

You can undo an operation

```
let coffee = anyCoffee();
let addIn  = anyAddIn();
assert(sameCoffee(
```

Inverse

You can undo an operation

```
let coffee = anyCoffee();
let addIn  = anyAddIn();
assert(sameCoffee(
  coffee,
```

Inverse

You can undo an operation

```
let coffee = anyCoffee();
let addIn  = anyAddIn();
assert(sameCoffee(
  coffee,
  coffee.add(addIn).remove(addIn)
```

Inverse

You can undo an operation

```
let coffee = anyCoffee();
let addIn  = anyAddIn();
assert(sameCoffee(
  coffee,
  coffee.add(addIn).remove(addIn)
));
```

Inverse

You can undo an operation

```
let coffee = anyCoffee();
let addIn  = anyAddIn();
assert(sameCoffee(
  coffee,
  coffee.add(addIn).remove(addIn)
));
```

$$g(f(a)) = a$$

Business rule: maximum # of add-ins

How does it affect our property?

We can't sell a coffee with
more than 5 add-ins.

Business rule: maximum # of add-ins

How does it affect our property?

We can't sell a coffee with
more than 5 add-ins.

```
function addAddIn(coffee, addIn) {
   if(countAddIns(coffee) >= 5)
      return coffee;
   else
      …
}
```

```
let coffee = anyCoffee();
let addIn = anyAddIn();
assert(sameCoffee(
  coffee,
  coffee.add(addIn).remove(addIn)
));
```

```
g(f(a)) = a
```

```
let coffee = anyCoffee();
let addIn = anyAddIn();
assert(sameCoffee(
  coffee,
  coffee.add(addIn).remove(addIn)
));
```

$$g(f(a)) = a$$

```
let coffee = newCoffee();
```

```
let coffee = anyCoffee();
let addIn = anyAddIn();
assert(sameCoffee(
  coffee,
  coffee.add(addIn).remove(addIn)
));
```

$$g(f(a)) = a$$

```
let coffee = newCoffee();
coffee = coffee.add(“soy”)
```

```
let coffee = anyCoffee();
let addIn = anyAddIn();
assert(sameCoffee(
  coffee,
  coffee.add(addIn).remove(addIn)
));
```

```
g(f(a)) = a
```

```
let coffee = newCoffee();
coffee = coffee.add(“soy”)
.add(“soy”)
```

```
let coffee = anyCoffee();
let addIn = anyAddIn();
assert(sameCoffee(
  coffee,
  coffee.add(addIn).remove(addIn)
));
```

```
g(f(a)) = a
```

```
let coffee = newCoffee();
coffee = coffee.add(“soy”)
.add(“soy”)
```

```
let coffee = anyCoffee();
let addIn = anyAddIn();
assert(sameCoffee(
  coffee,
  coffee.add(addIn).remove(addIn)
));
```

```
g(f(a)) = a
```

```
let coffee = newCoffee();
coffee = coffee.add(“soy”)
.add(“soy”)
.add(“soy”)
```

```
let coffee = anyCoffee();
let addIn = anyAddIn();
assert(sameCoffee(
  coffee,
  coffee.add(addIn).remove(addIn)
));
```

$$g(f(a)) = a$$

```
let coffee = newCoffee();
coffee = coffee.add(“soy”)
.add(“soy”)
.add(“soy”)
.add(“soy);
```

```
let coffee = anyCoffee();
let addIn = anyAddIn();
assert(sameCoffee(
  coffee,
  coffee.add(addIn).remove(addIn)
));
```

```
g(f(a)) = a
```

```
let coffee = newCoffee();
coffee = coffee.add(“soy”)
               .add(“soy”)
               .add(“soy”)
               .add(“soy);
let coffee2 = coffee.add(“soy”) // no op
```

```
let coffee = anyCoffee();
let addIn = anyAddIn();
assert(sameCoffee(
  coffee,
  coffee.add(addIn).remove(addIn)
));
```

```
g(f(a)) = a
```

```
let coffee = newCoffee();
coffee = coffee.add(“soy”)
               .add(“soy”)
               .add(“soy”)
               .add(“soy);
let coffee2 = coffee.add(“soy”) // no op
                    .remove(“soy”);
```

```
let coffee = anyCoffee();
let addIn = anyAddIn();
assert(sameCoffee(
  coffee,
  coffee.add(addIn).remove(addIn)
));
```

```
g(f(a)) = a
```

```
let coffee = newCoffee();
coffee = coffee.add(“soy”)
               .add(“soy”)
               .add(“soy”)
               .add(“soy);
let coffee2 = coffee.add(“soy”) // no op
                    .remove(“soy”);
```

```
let coffee = anyCoffee();
let addIn = anyAddIn();
assert(sameCoffee(
  coffee,
  coffee.add(addIn).remove(addIn)
));
```

```
g(f(a)) = a
```

```
let coffee = newCoffee();
coffee = coffee.add(“soy”)
               .add(“soy”)
               .add(“soy”)
               .add(“soy);
let coffee2 = coffee.add(“soy”) // no op
                    .remove(“soy”);

assert(sameCoffee(coffee, coffee2));
```

```
let coffee = anyCoffee();
let addIn = anyAddIn();
assert(sameCoffee(
  coffee,
  coffee.add(addIn).remove(addIn)
));
```

```
g(f(a)) = a
```

```
let coffee = newCoffee();
coffee = coffee.add(“soy”)
               .add(“soy”)
               .add(“soy”)
               .add(“soy);
let coffee2 = coffee.add(“soy”) // no op
                    .remove(“soy”);

assert(sameCoffee(coffee, coffee2));
```

Total/partial property

We prefer total properties

Total/partial property

We prefer total properties

```
let coffee = anyCoffee();
if(countAddIns(coffee) < 5) {
  let addIn = anyAddIn();
  assert(sameCoffee(
    coffee,
    coffee.add(addIn).remove(addIn)
  ));
}
```

Total/partial property

We prefer total properties

```
let coffee = anyCoffee();
if(countAddIns(coffee) < 5) {
  let addIn = anyAddIn();
  assert(sameCoffee(
    coffee,
    coffee.add(addIn).remove(addIn)
  ));
}
```

partial property

Scope

```
function addAddIn(coffee, addIn) //=> coffee
function removeAddIn(coffee, addIn) //=> coffee

function normalize(coffee) //=> coffee
function isValid(coffee) //=> boolean
```

```
function addAddIn(coffee, addIn) //=> coffee
function removeAddIn(coffee, addIn) //=> coffee

function normalize(coffee) //=> coffee
function isValid(coffee) //=> boolean
```

Coffee Shops

```
function addAddIn(coffee, addIn) //=> coffee
function removeAddIn(coffee, addIn) //=> coffee

function normalize(coffee) //=> coffee
function isValid(coffee) //=> boolean
```

Our Coffee Shop

Coffee Shops

```
function addAddIn(coffee, addIn) //=> coffee
function removeAddIn(coffee, addIn) //=> coffee

function normalize(coffee) //=> coffee
function isValid(coffee) //=> boolean
```

```
function isValid(coffee) //=> boolean
```

Our Coffee Shop

Coffee Shops

```
function addAddIn(coffee, addIn) //=> coffee
function removeAddIn(coffee, addIn) //=> coffee

function normalize(coffee) //=> coffee
function isValid(coffee) //=> boolean
```

Our Coffee Shop

Coffee Shops

Our Coffee Shop

Coffee Shops

```
function addAddIn(coffee, addIn) {
  // if(countAddIns(coffee) >= 5)
  //  return coffee;
  // else
    …
}
```

Coffee Shops

comment out

```
function addAddIn(coffee, addIn) {
  // if(countAddIns(coffee) >= 5)
  //  return coffee;
  // else
    …
}
```

```
function isValid(coffee) {
  return countAddIns(coffee) <= 5 && …
}
```

Our Coffee Shop

Coffee Shops

```
function addAddIn(coffee, addIn) {
  // if(countAddIns(coffee) >= 5)
  //  return coffee;
  // else
    …
}
```

```
function isValid(coffee) {
  return countAddIns(coffee) <= 5 && …
}
```

business rules

Our Coffee Shop

Coffee Shops

```
function addAddIn(coffee, addIn) {
  // if(countAddIns(coffee) >= 5)
  //  return coffee;
  // else
    …
}
```

comment out

Defer a decision to a higher layer

If it helps you eliminate a corner case/partial property

- Only if you can justify it semantically

Go down to a lower layer

If it helps you organize complication

Go down to a lower layer

If it helps you organize complication

- Coupons

Go down to a lower layer

If it helps you organize complication

- Coupons
- Weekly promotions

Go down to a lower layer

If it helps you organize complication

- Coupons
- Weekly promotions
- Loyalty perks

Go down to a lower layer

If it helps you organize complication

- Coupons
- Weekly promotions
- Loyalty perks
- Percentage discounts

Go down to a lower layer

If it helps you organize complication

- Coupons
- Weekly promotions
- Loyalty perks
- Percentage discounts
- Fixed discount

Go down to a lower layer

If it helps you organize complication

- Coupons
- Weekly promotions
- Loyalty perks
- Percentage discounts
- Fixed discount
- Buy one get one free

Go down to a lower layer

If it helps you organize complication

- Coupons
- Weekly promotions
- Loyalty perks
- Percentage discounts
- Fixed discount
- Buy one get one free
- Discounts on individual coffees

Go down to a lower layer

If it helps you organize complication

- Coupons
- Weekly promotions
- Loyalty perks
- Percentage discounts
- Fixed discount
- Buy one get one free
- Discounts on individual coffees
- Complex Requirements

Go down to a lower layer

If it helps you organize complication

- Coupons
- Weekly promotions
- Loyalty perks

- Percentage discounts
- Fixed discount
- Buy one get one free
- Discounts on individual coffees

- Complex Requirements
  - Buy 5, get 6th free

Go down to a lower layer

If it helps you organize complication

- Coupons
- Weekly promotions
- Loyalty perks

- Percentage discounts
- Fixed discount
- Buy one get one free
- Discounts on individual coffees

- Complex Requirements
  - Buy 5, get 6th free
  - Any latte

Go down to a lower layer

If it helps you organize complication

- Coupons
- Weekly promotions
- Loyalty perks

- Percentage discounts
- Fixed discount
- Buy one get one free
- Discounts on individual coffees

- Complex Requirements
  - Buy 5, get 6th free
  - Any latte
  - Coffee with almond and coffee with soy

Go down a layer

Our Discounts

Discounts

Go down a layer

Our Discounts

Discounts

```
type DiscountAmount = NoDiscount        |
                      PercentDiscount   |
                      FixedDiscount     |
                      BothDiscounts     |
                      HighestDiscount ;
```

Go down a layer

Our Discounts

Discounts

```
type DiscountAmount = NoDiscount        |
                      PercentDiscount   |
                      FixedDiscount     |
                      BothDiscounts     |
                      HighestDiscount ;

function discount(order)                //=> DiscountAmount
```

Go down a layer

Our Discounts

Discounts

```
type DiscountAmount = NoDiscount        |
                      PercentDiscount   |
                      FixedDiscount     |
                      BothDiscounts     |
                      HighestDiscount ;

function discount(order)          //=> DiscountAmount
function guardDiscount(discount)  //=> discount
```

Go down a layer

Our Discounts

Discounts

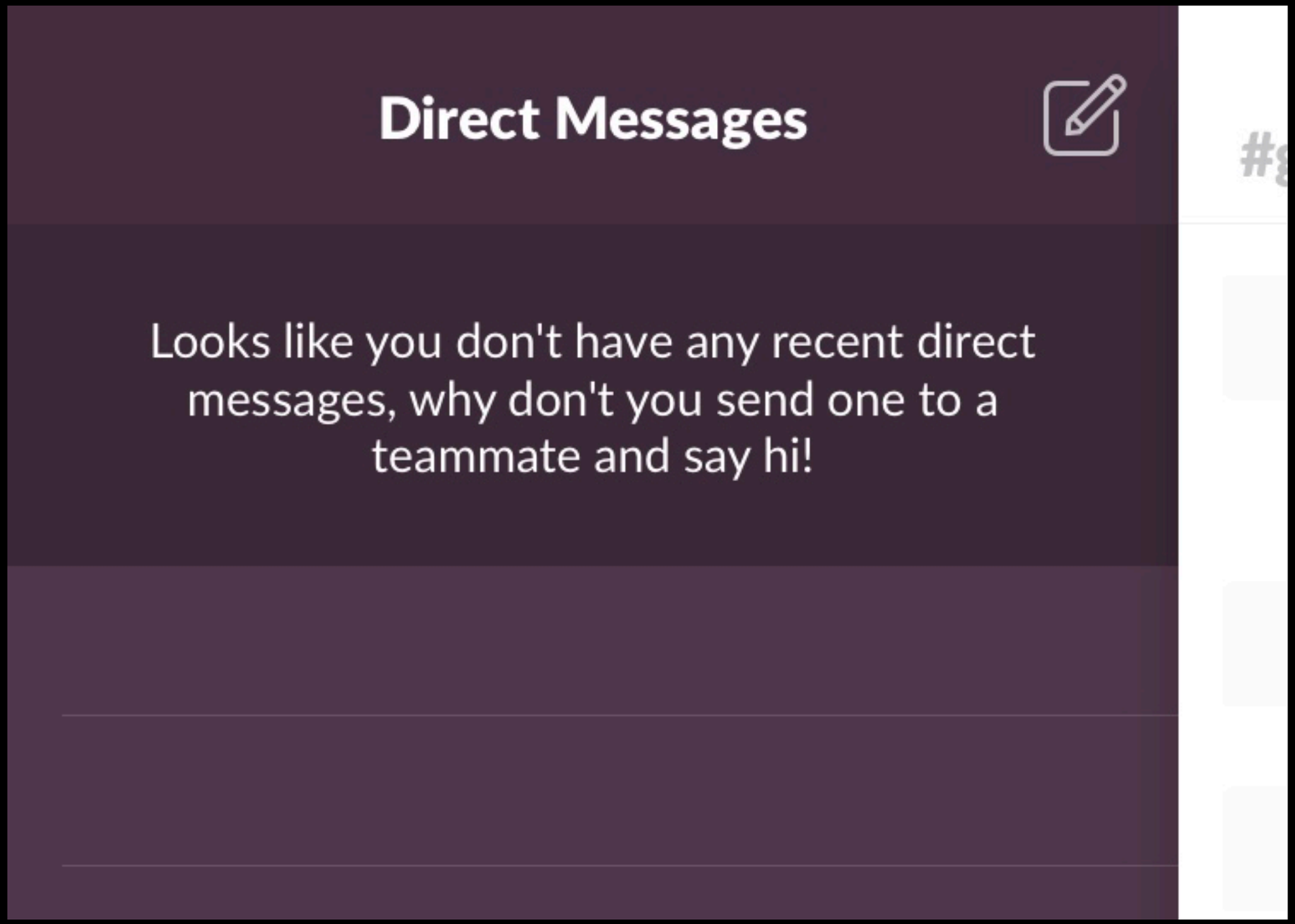
```
type DiscountAmount = NoDiscount        |
                      PercentDiscount   |
                      FixedDiscount     |
                      BothDiscounts     |
                      HighestDiscount ;

function discount(order)           //=> DiscountAmount
function guardDiscount(discount)   //=> discount
function combineDiscounts(d1, d2) //=> discount
```

Platform

Sometimes your platform is complex enough to model as a subdomain

AJAX Loading example

A common bug that needs a solution



http://blog.jenkster.com/2016/06/how-elm-slays-a-ui-antipattern.html

```
let messages = [];
```

```
let messages = [];
```

```
let messages = [];

ajax.get(“/messages”, data => messages = data.messages);
```

```
let messages = [];

ajax.get(“/messages”, data => messages = data.messages);


 function view({messages}) {
```

```
let messages = [];

ajax.get(“/messages”, data => messages = data.messages);


 function view({messages}) {
   if(!messages)
```

```
let messages = [];

ajax.get(“/messages”, data => messages = data.messages);


 function view({messages}) {
  if(!messages)
    return <div>No messages</div>;
```

```
let messages = [];

ajax.get(“/messages”, data => messages = data.messages);


 function view({messages}) {
  if(!messages)
    return <div>No messages</div>;
  return (<ul>
```

```
let messages = [];

ajax.get(“/messages”, data => messages = data.messages);


 function view({messages}) {
if(!messages)
  return <div>No messages</div>;
return (<ul>
 {messages.map(message => <li>{message}</li>)}
```

```
let messages = [];

ajax.get(“/messages”, data => messages = data.messages);


 function view({messages}) {
if(!messages)
  return <div>No messages</div>;
return (<ul>
 {messages.map(message => <li>{message}</li>)}
</ul>);
```

```
let messages = [];

ajax.get(“/messages”, data => messages = data.messages);


 function view({messages}) {
   if(!messages)
     return <div>No messages</div>;
   return (<ul>
    {messages.map(message => <li>{message}</li>)}
   </ul>);
}
```

AJAX is more complex than that

AJAX is more complex than that

What about loading state?

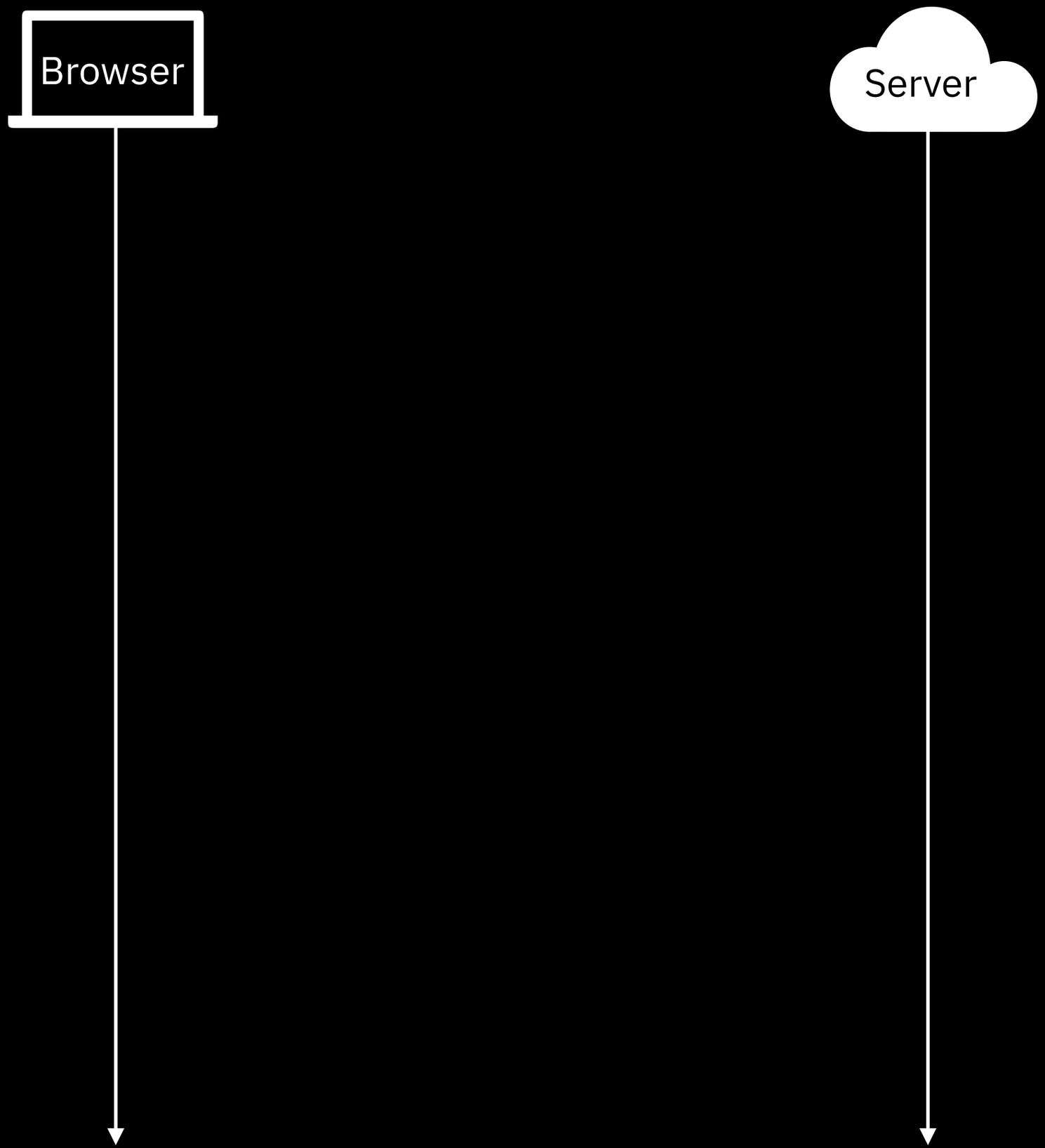
AJAX is more complex than that

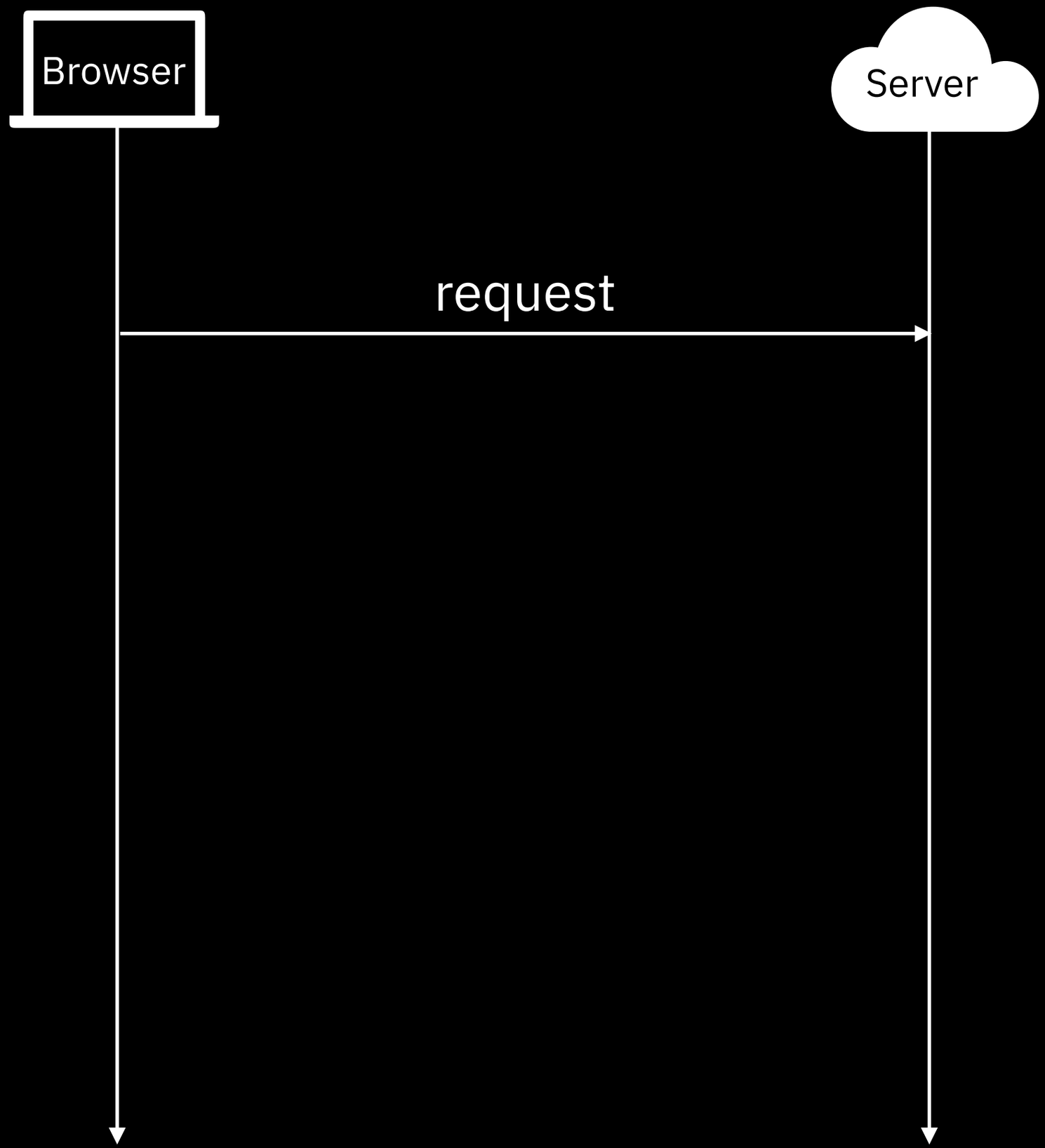
What about loading state?
What about errors?

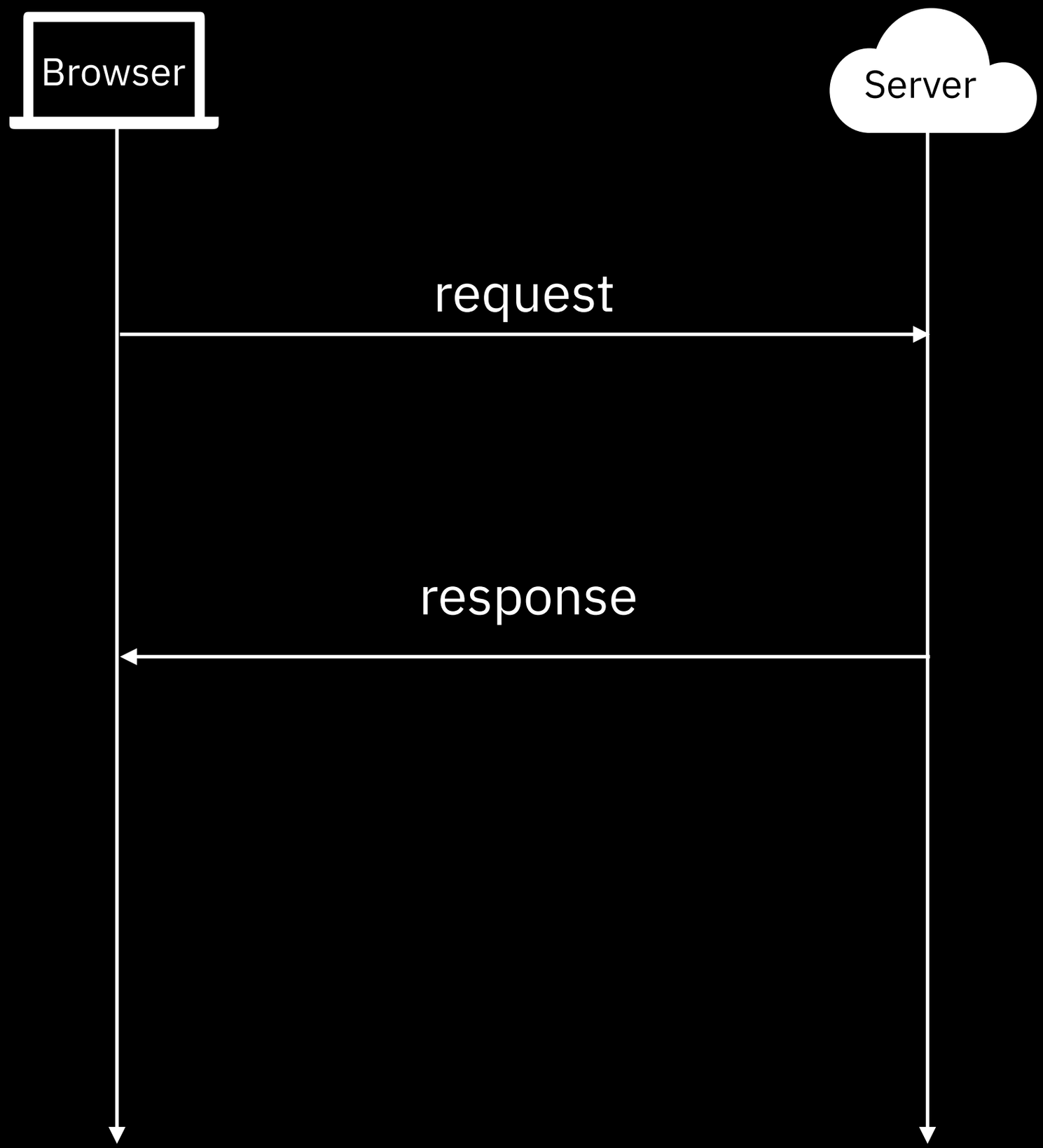
AJAX is more complex than that

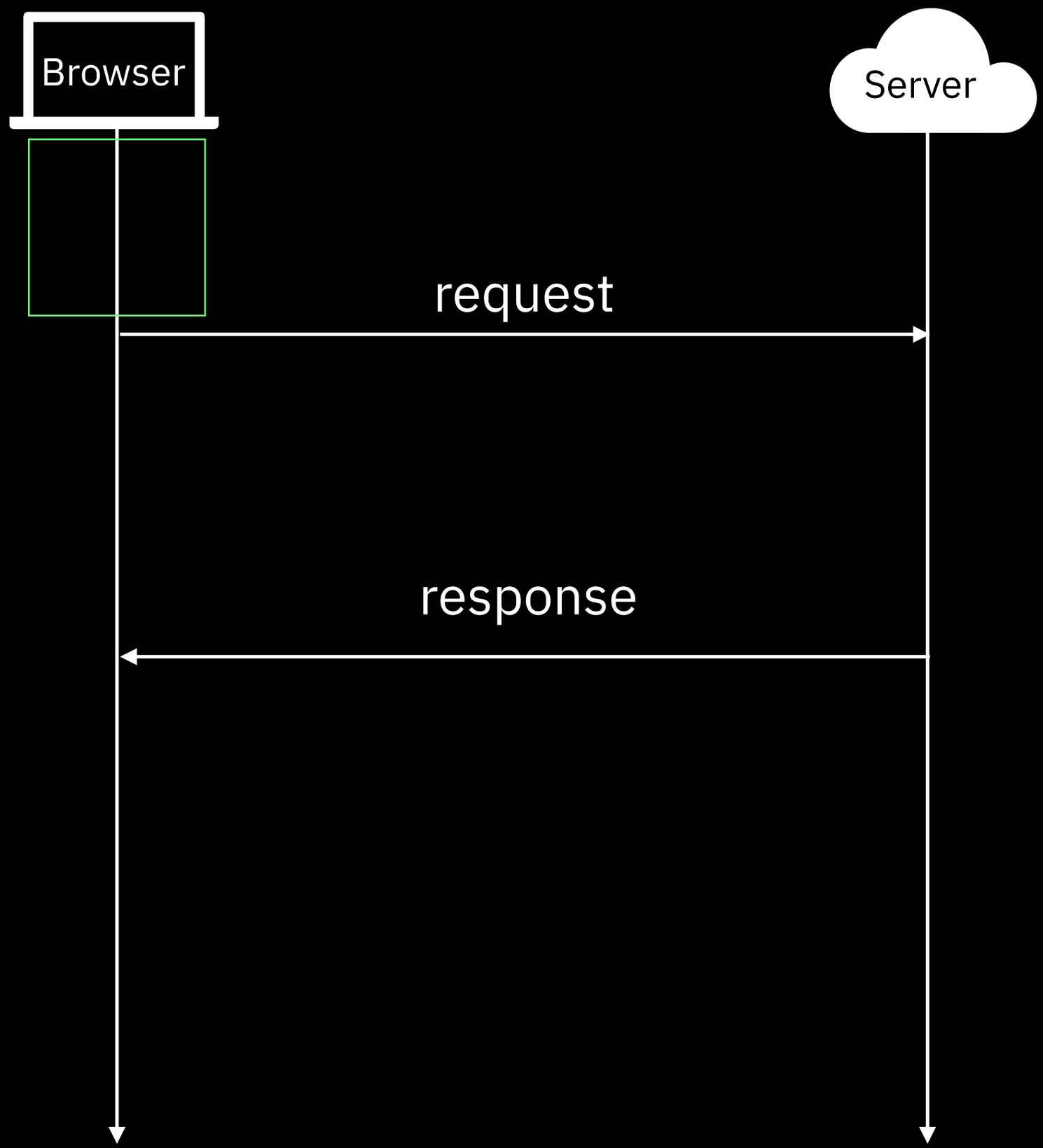
What about loading state?
What about errors?

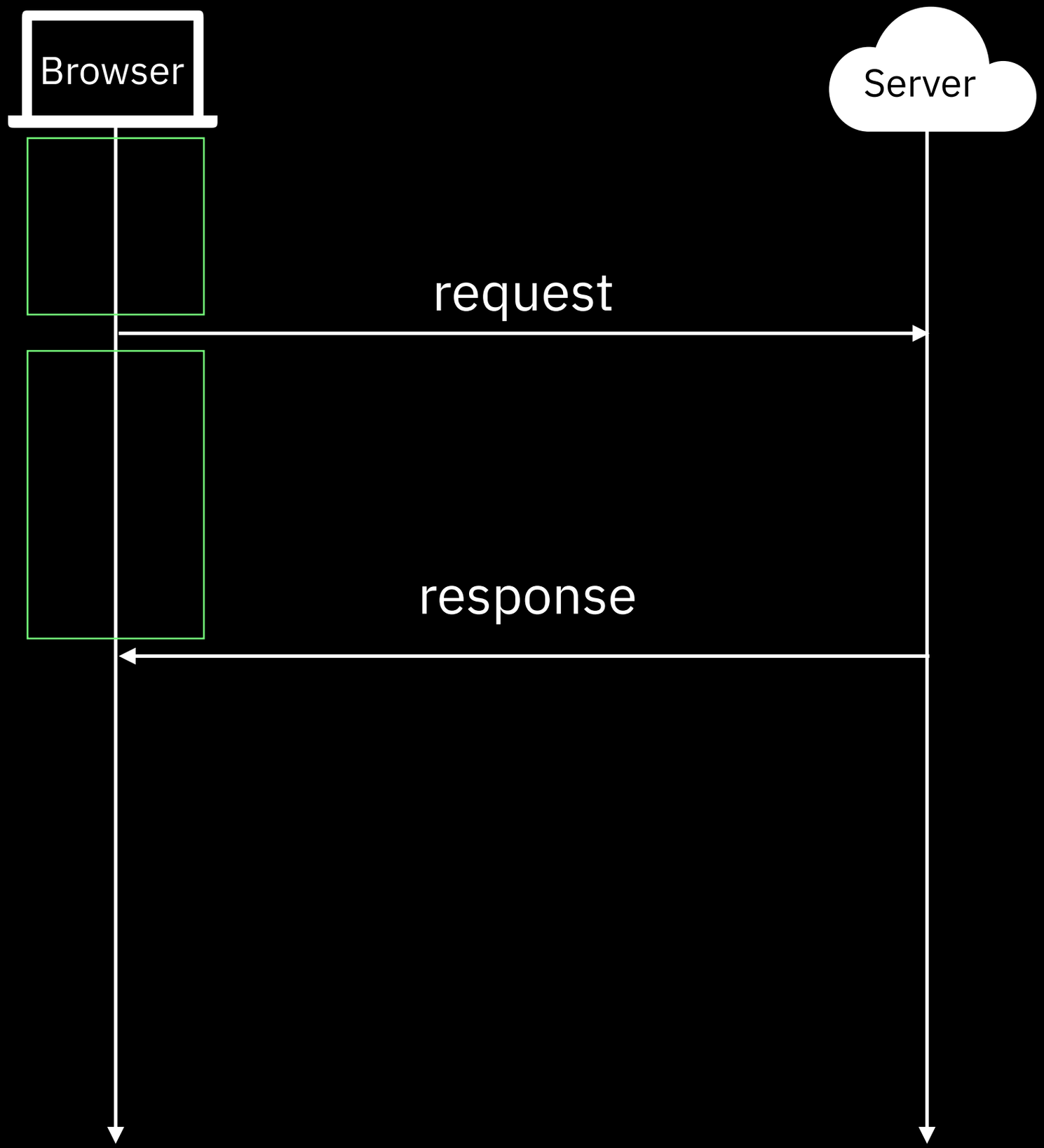
Let’s model the AJAX value!

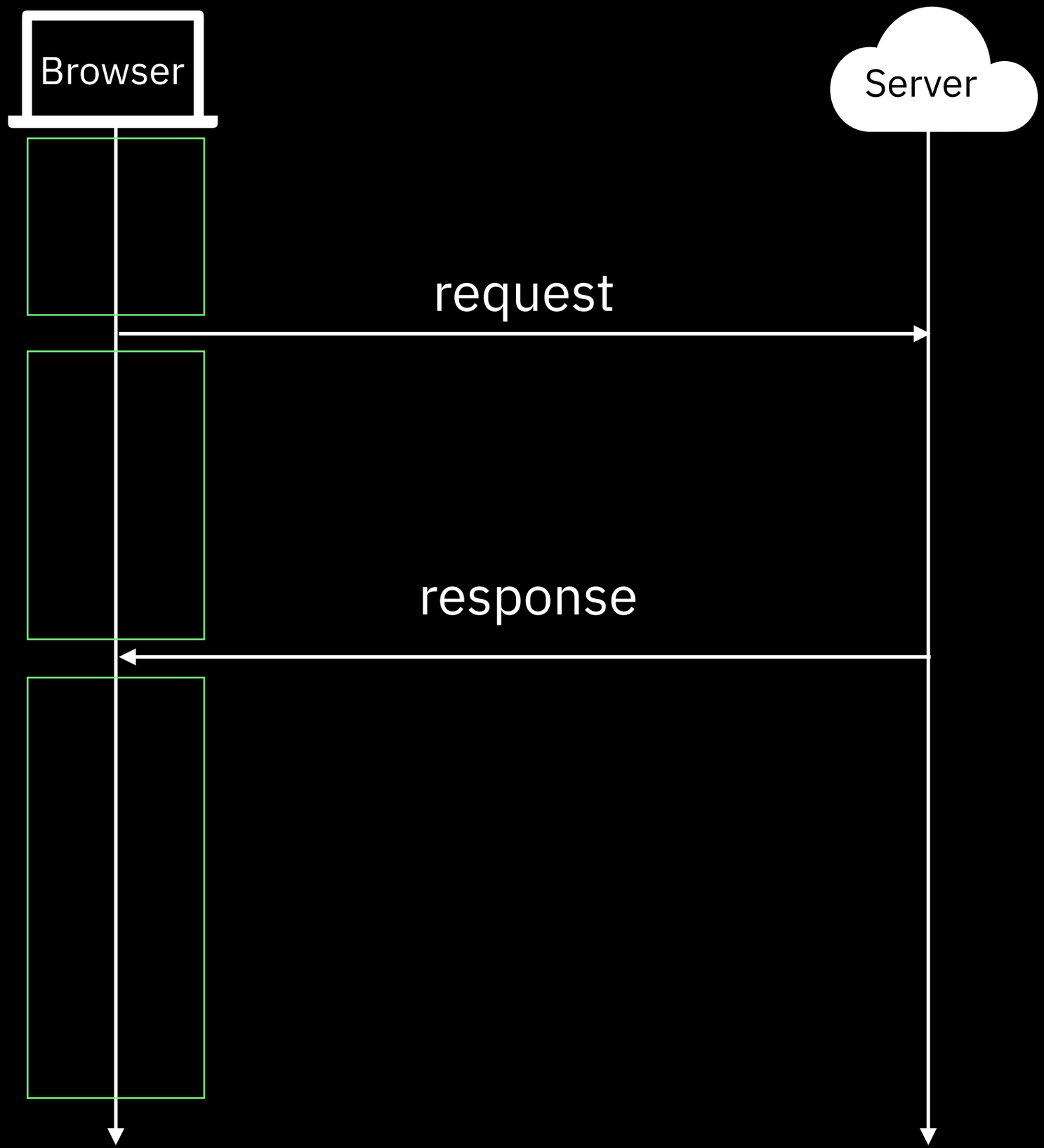

Browser
Server

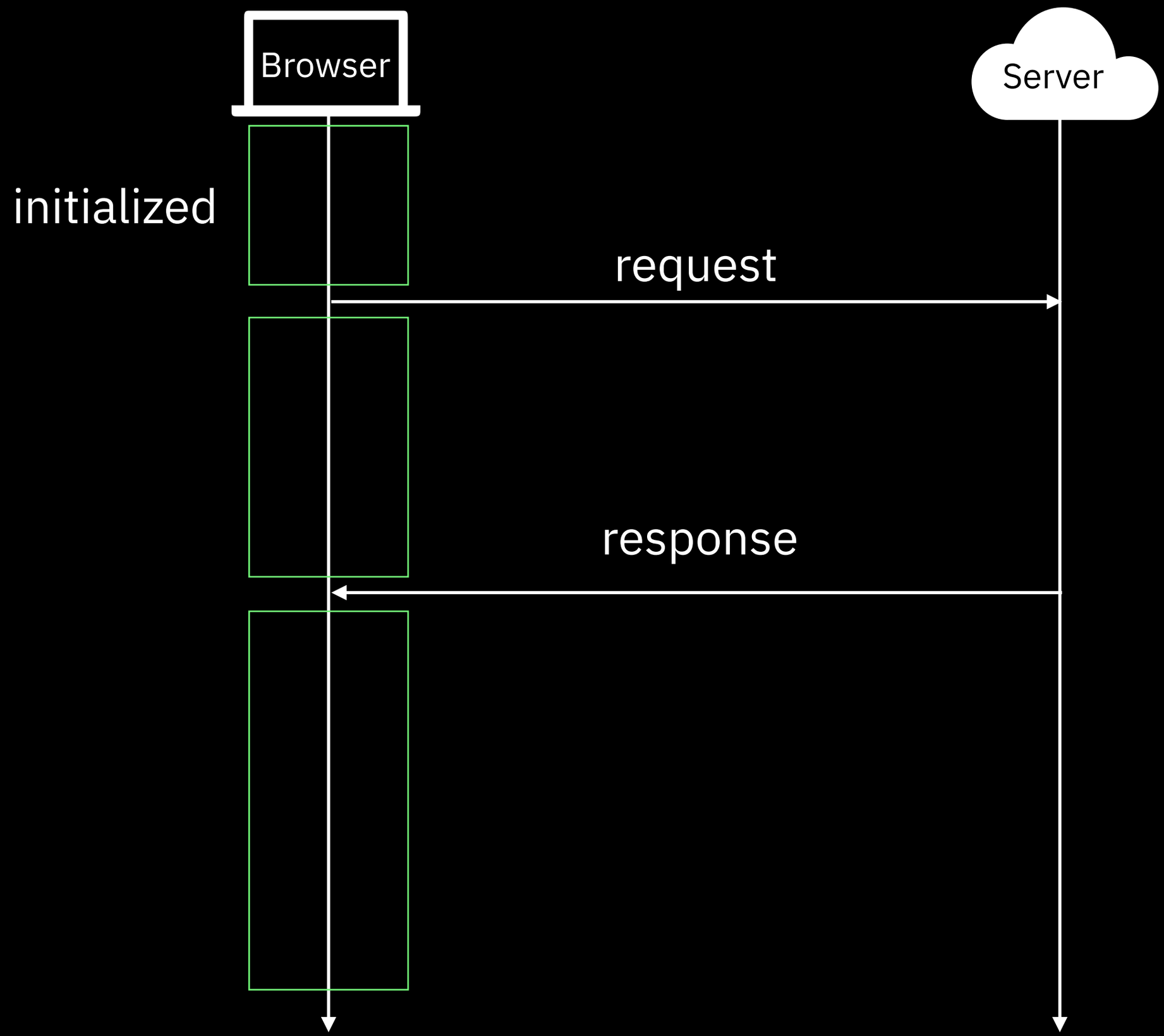

Browser
Server
request

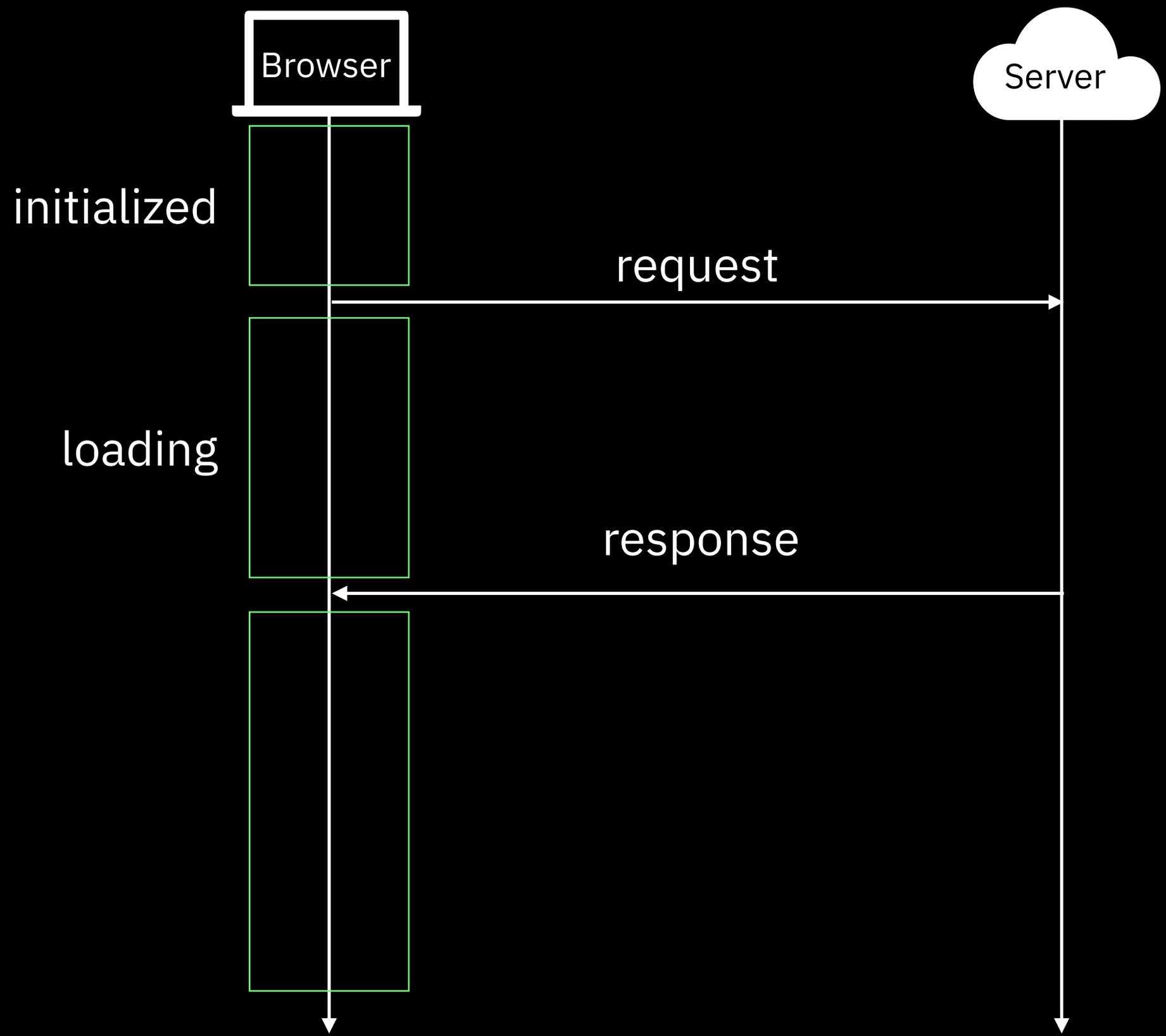

Browser
Server
request
response

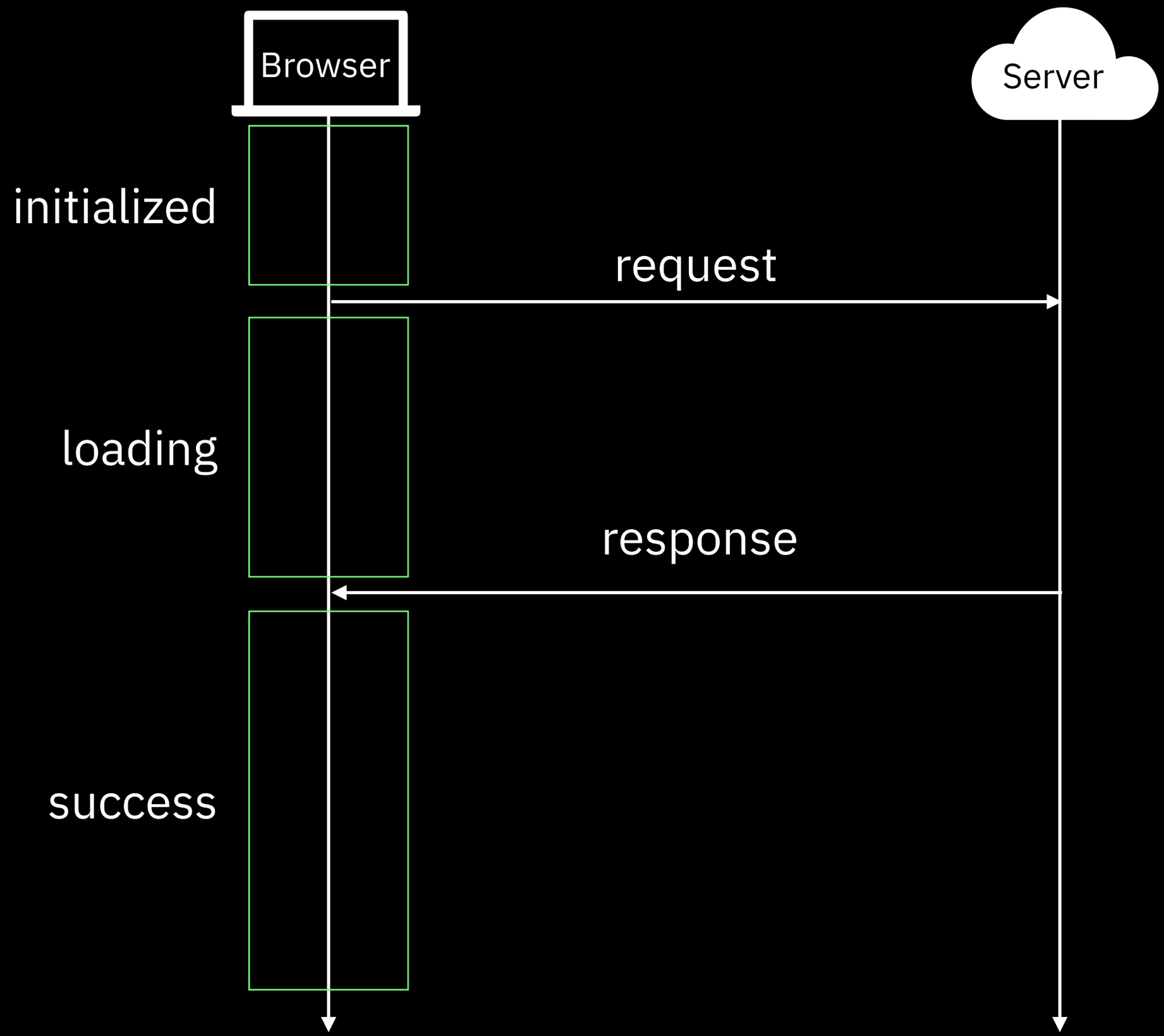

Browser
Server
request
response

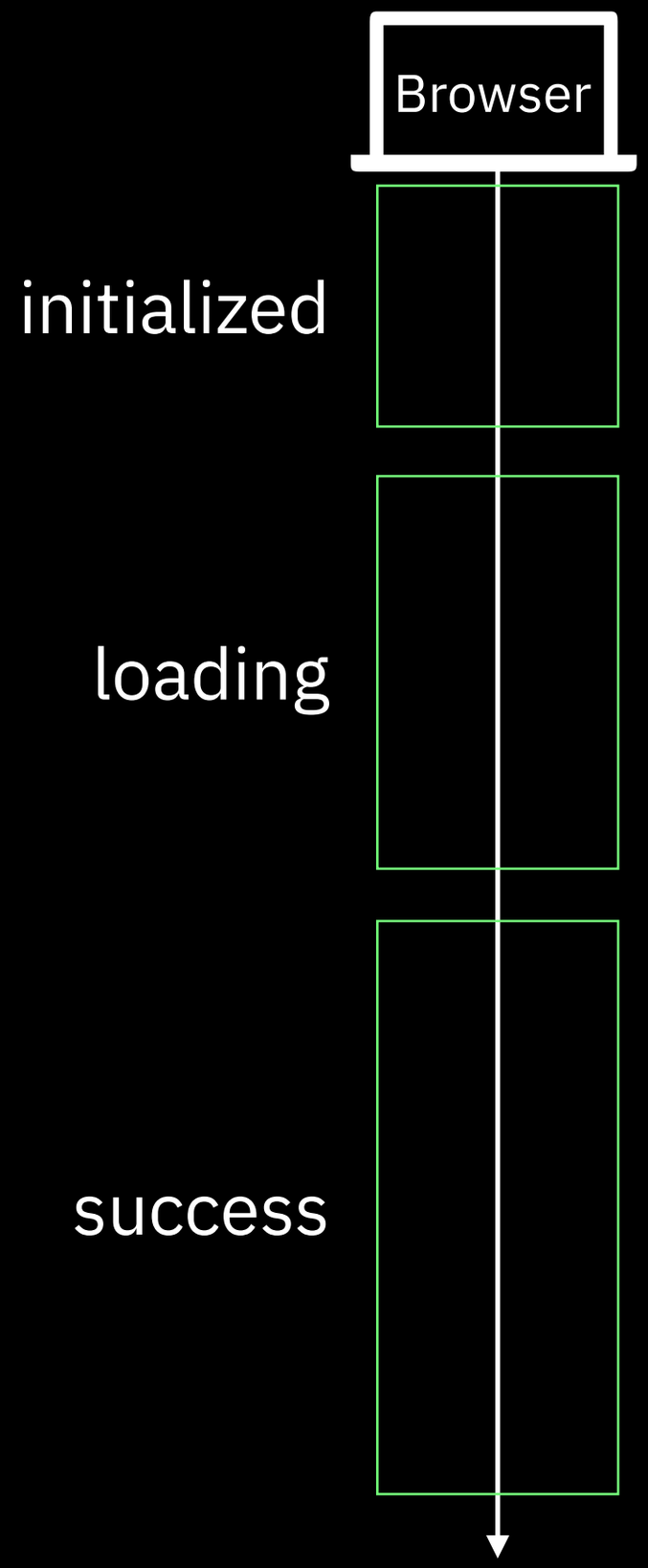

Browser
Server
request
response

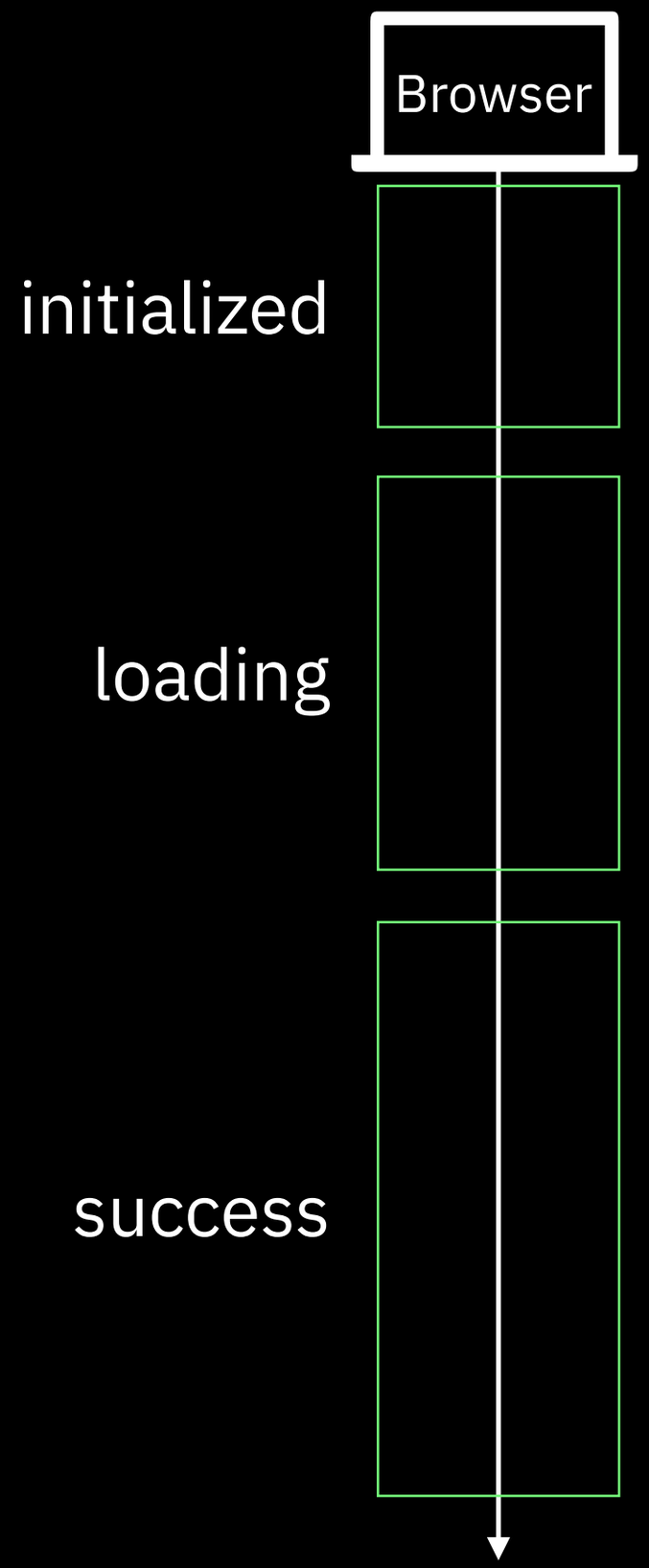

Browser
Server
request
response


Browser
Server
initialized
request
response

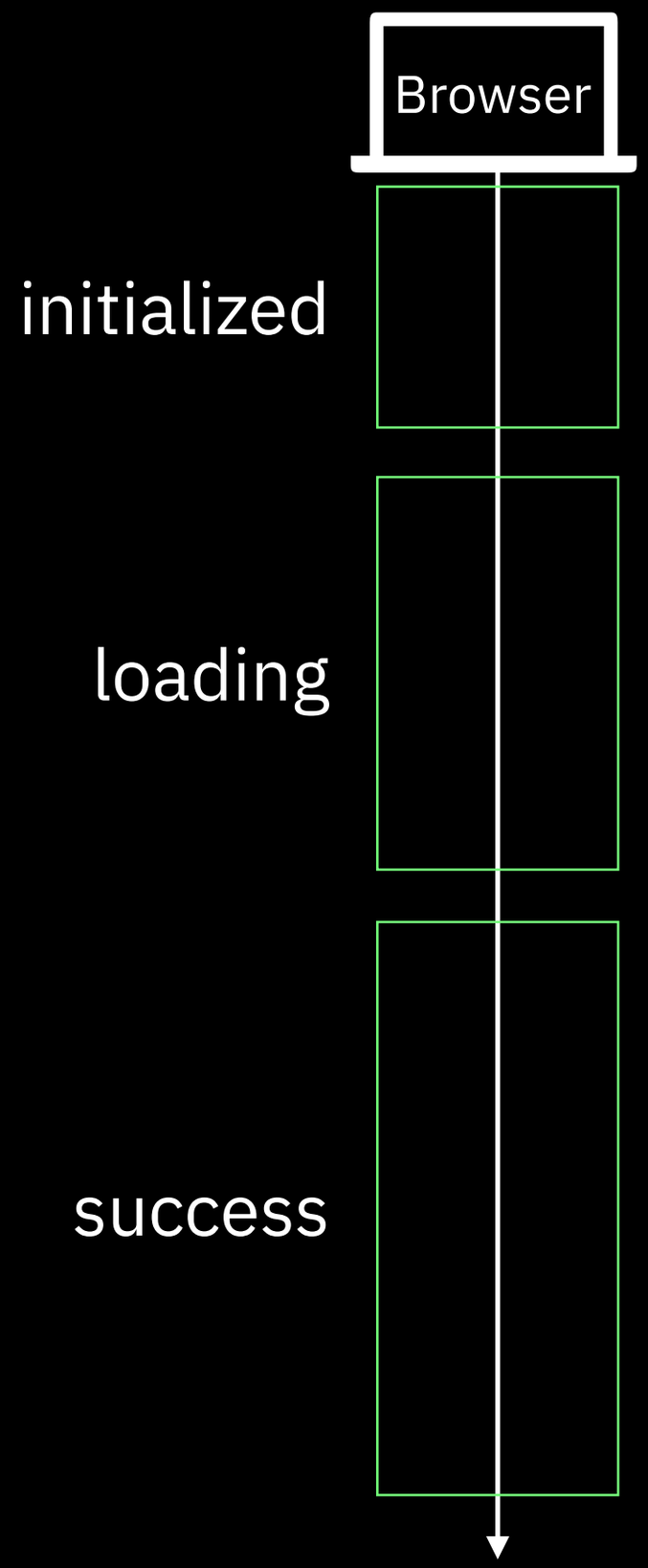

Browser
Server
initialized
request
loading
response


Browser
Server
initialized
request
loading
response
success

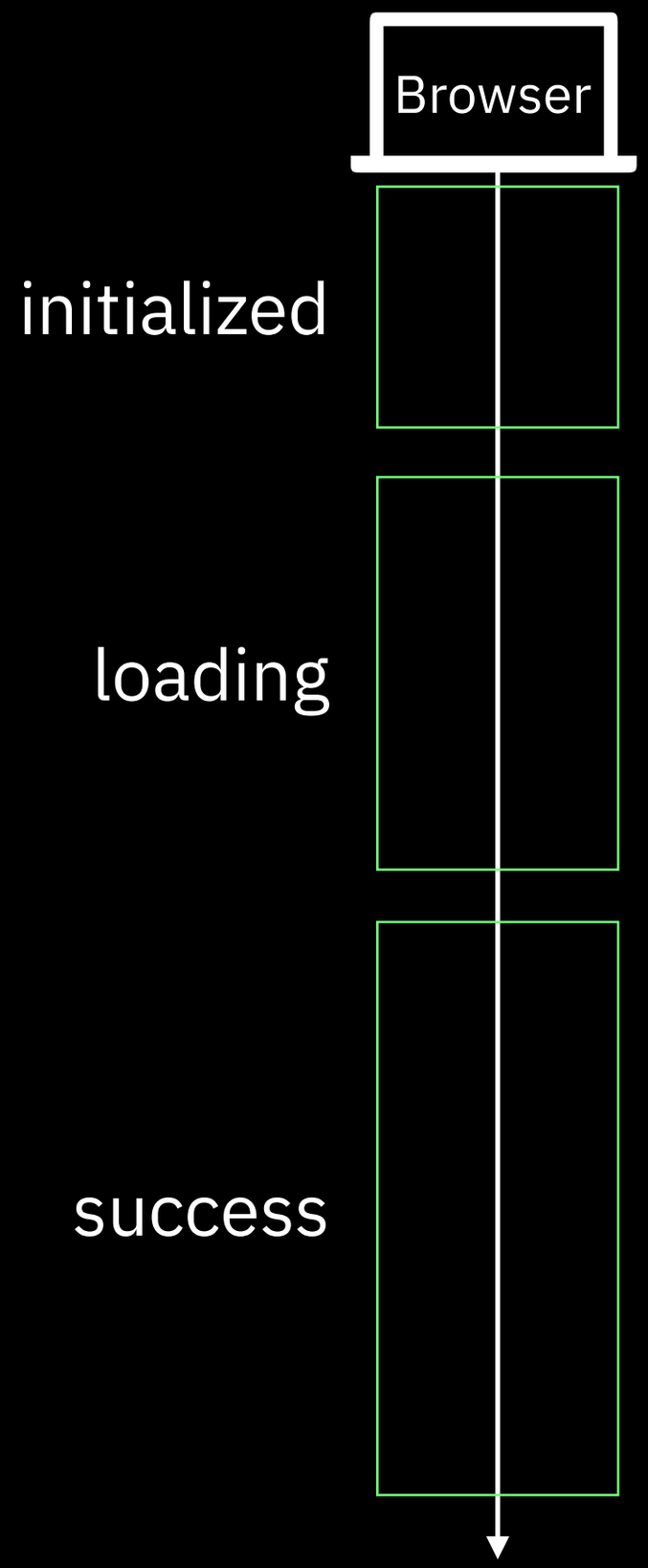

Browser
initialized
loading
success



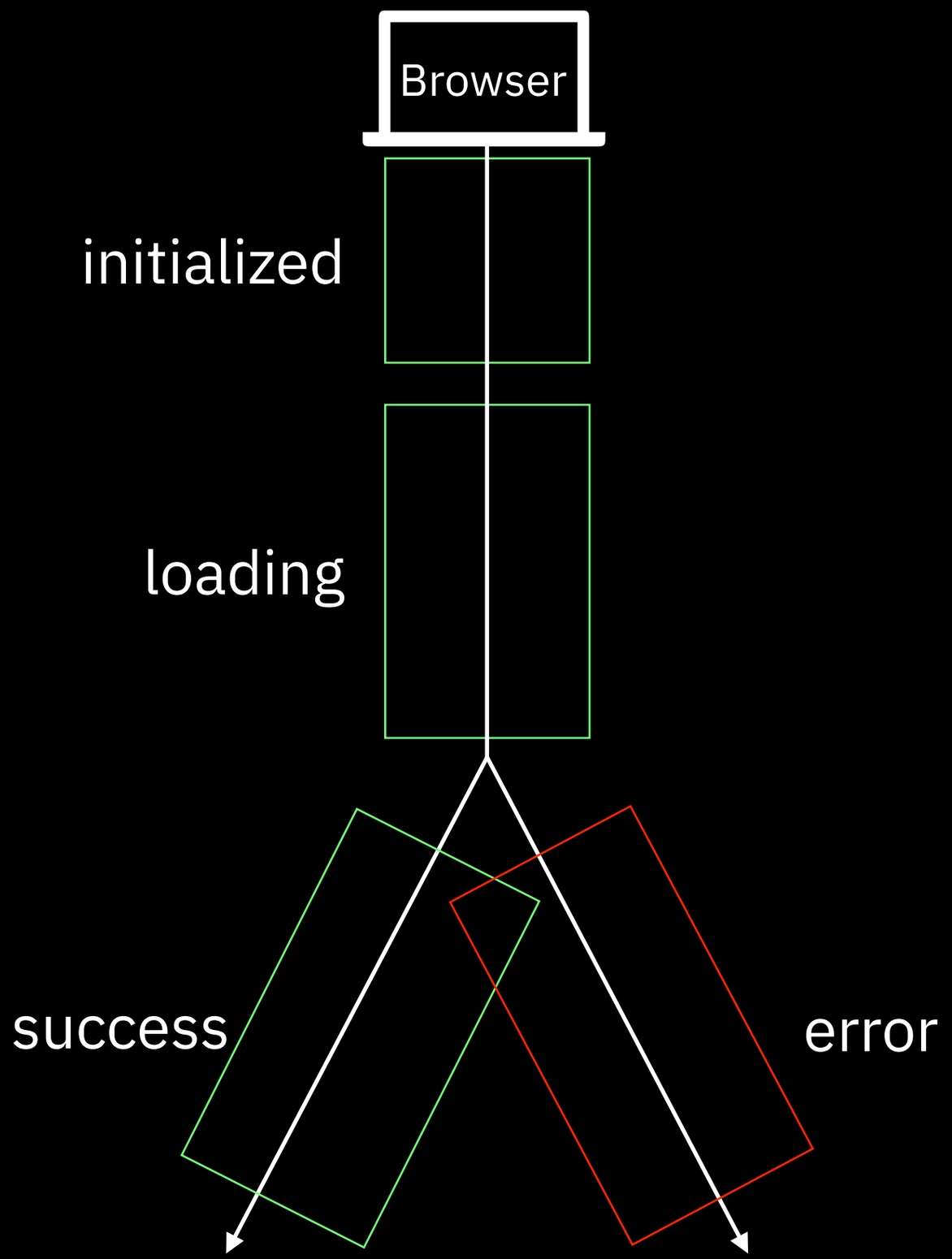

```
{
  status: “initialized”
}
```




```
{
  status: “initialized”
}

{
  status: “loading”
}
```




```
{
  status: “initialized”
}

{
  status: “loading”
}

{
  status: “success”,
  value: […]
}
```




```
{
  status: “initialized”
}

{
  status: “loading”
}

{
  status: “success”,
  value: […]
}

{
  status: “error”,
  message: “404”
}
```

```
function newAjaxValue() {
```

```
function newAjaxValue() {
  return {status: “initialized”};
```

```
function newAjaxValue() {
  return {status: “initialized”};
}
```

```
function newAjaxValue() {
  return {status: “initialized”};
}
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return {status: “loading”};
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return {status: “loading”};
}
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return {status: “loading”};
}
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return {status: “loading”};
}

function succeed(ajaxValue, value) {
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return {status: “loading”};
}

function succeed(ajaxValue, value) {
return {status: “success”,
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return {status: “loading”};
}

function succeed(ajaxValue, value) {
  return {status: “success”,
          value: value};
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return {status: “loading”};
}

function succeed(ajaxValue, value) {
  return {status: “success”,
          value: value};
}
```

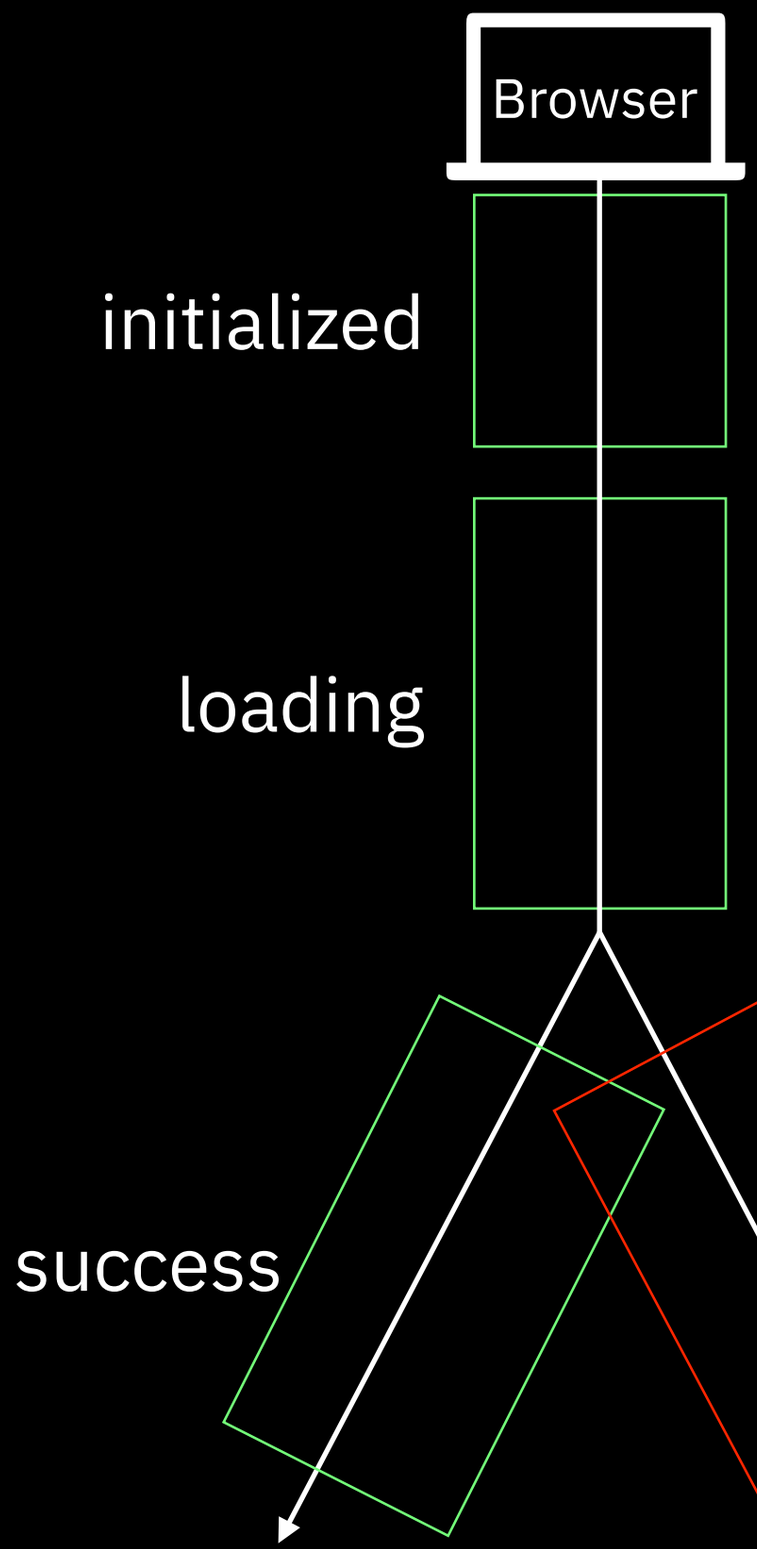
```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return {status: “loading”};
}

function succeed(ajaxValue, value) {
  return {status: “success”,
          value: value};
}
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return {status: “loading”};
}

function succeed(ajaxValue, value) {
  return {status: “success”,
          value: value};
}

function error(ajaxValue, message) {
```

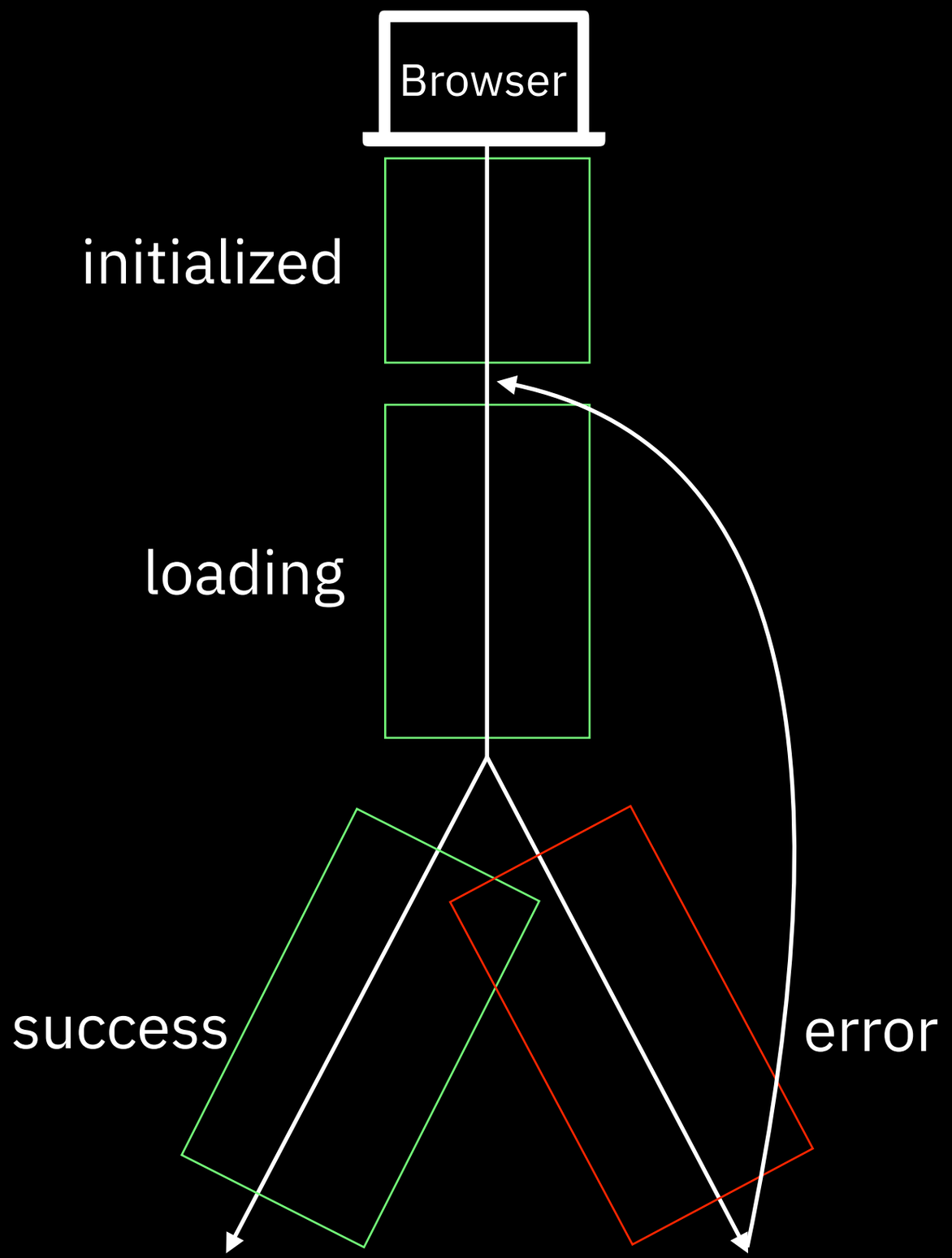
```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return {status: “loading”};
}

function succeed(ajaxValue, value) {
  return {status: “success”,
          value: value};
}

function error(ajaxValue, message) {
  return {status: “error”,
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return {status: “loading”};
}

function succeed(ajaxValue, value) {
  return {status: “success”,
          value: value};
}

function error(ajaxValue, message) {
  return {status: “error”,
          message: message};
```

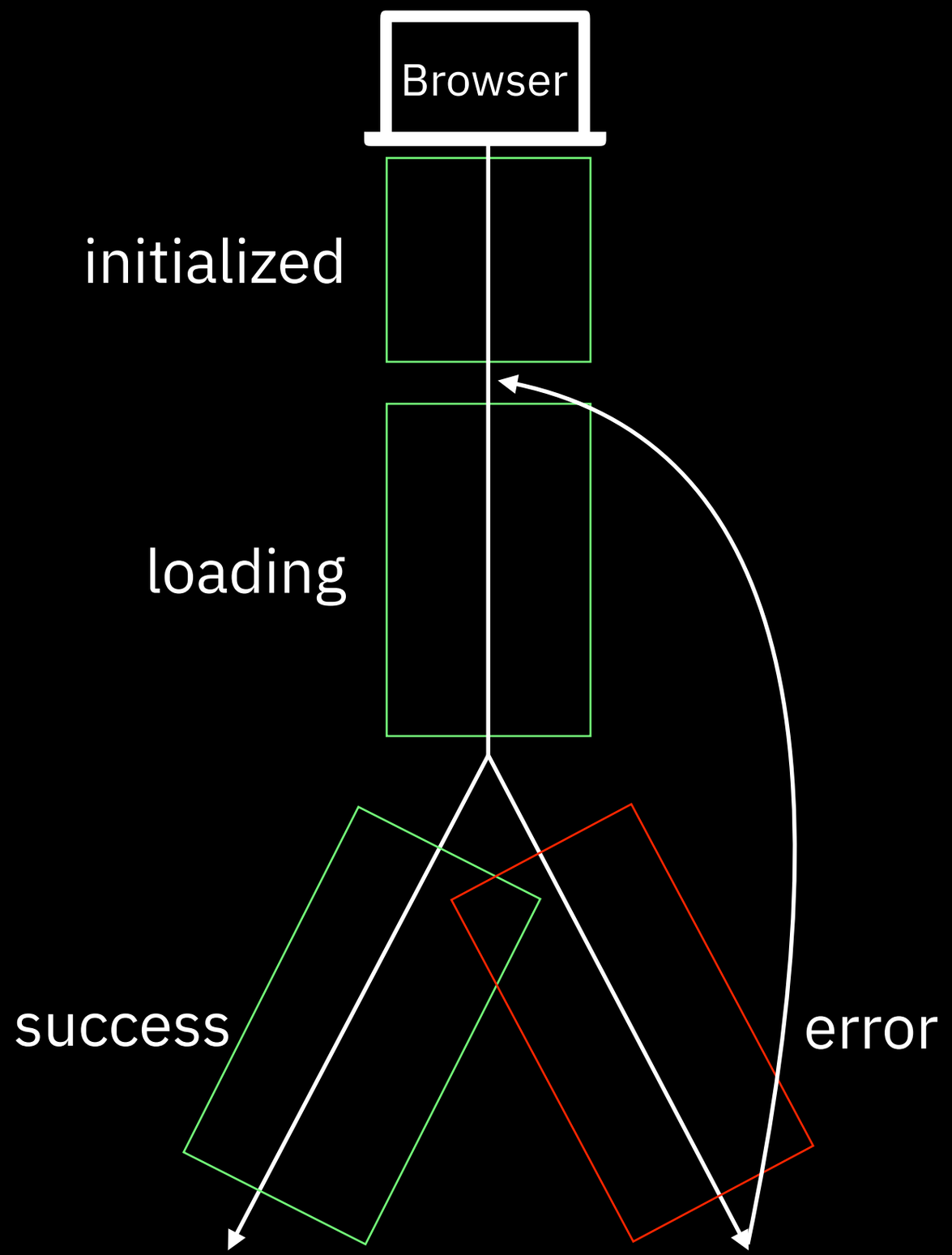
```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return {status: “loading”};
}

function succeed(ajaxValue, value) {
  return {status: “success”,
          value: value};
}

function error(ajaxValue, message) {
  return {status: “error”,
          message: message};
}
```



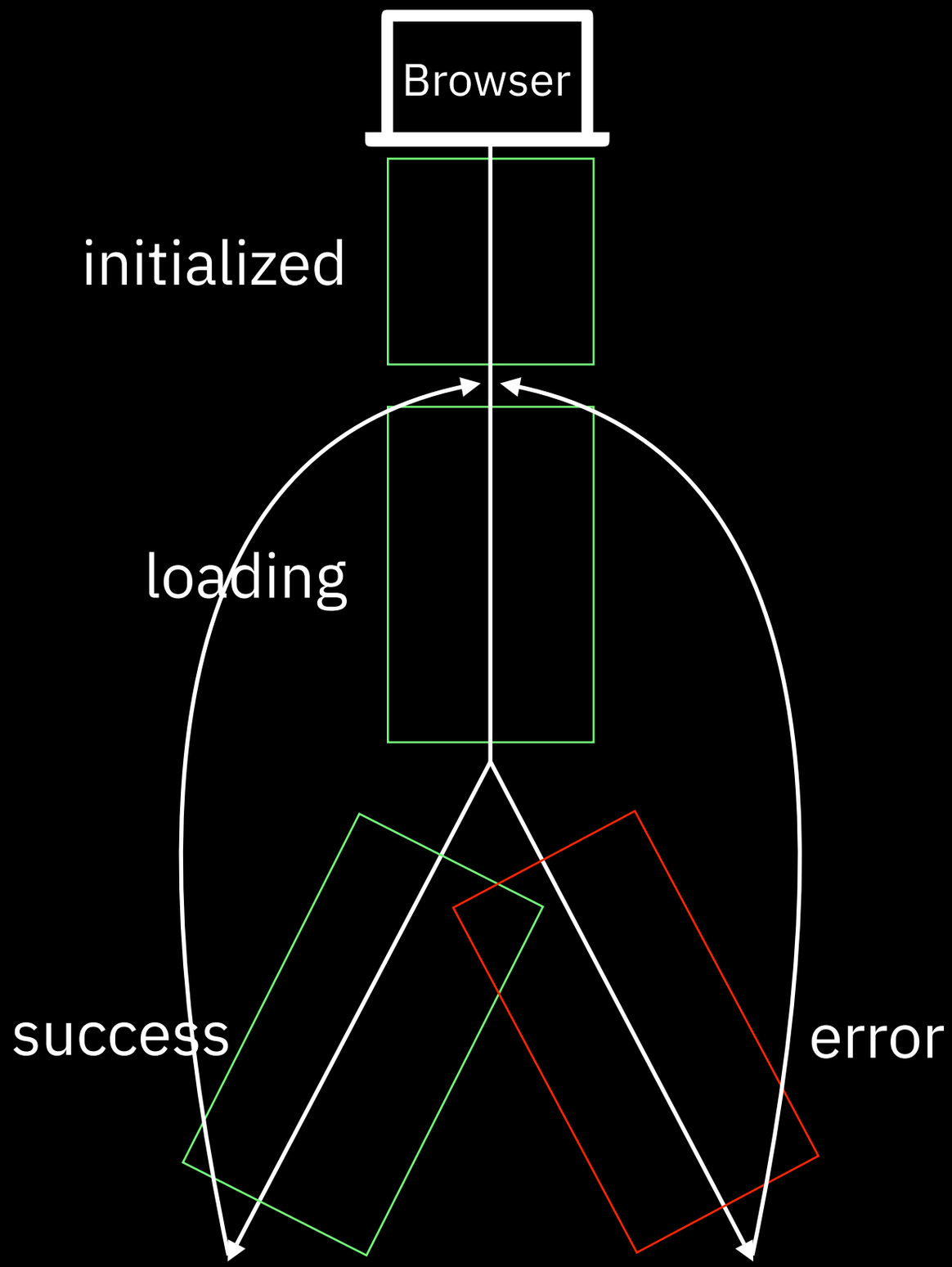

```
{
  status: “initialized”
}

{
  status: “loading”
}

{
  status: “success”,
  value: […]
}

{
  status: “error”,
  message: “404”
}
```



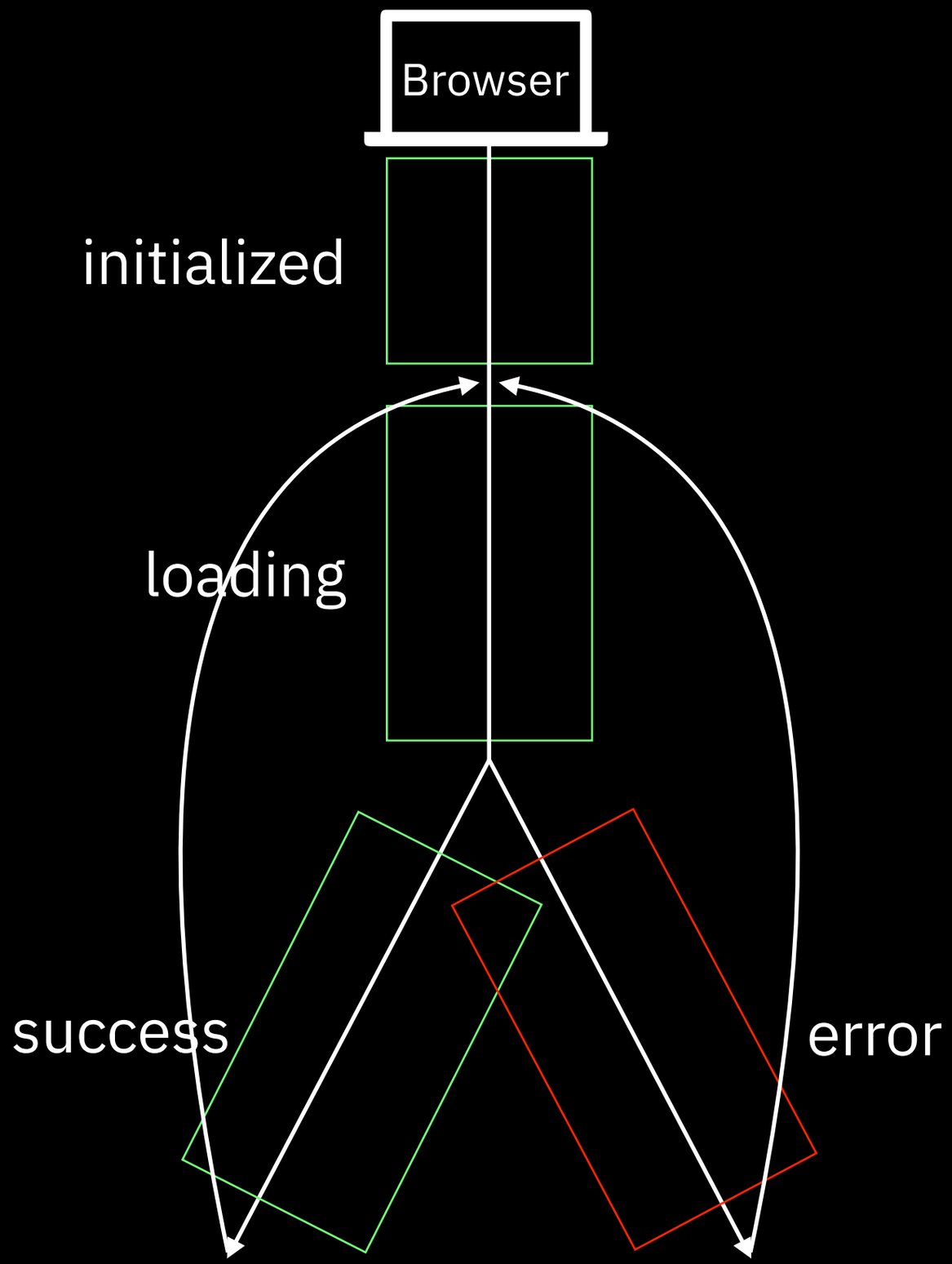

```
{
  status: “initialized”
}

{
  status: “loading”
}

{
  status: “success”,
  value: […]
}

{
  status: “error”,
  message: “404”
}
```




```
{
  status: “initialized”
}

{
  status: “loading”
}

{
  status: “success”,
  value: […]
}

{
  status: “error”,
  message: “404”,
  value?: […]
}
```




```
{
  status: “initialized”
}

{
  status: “loading”
}

{{
  status: “success”,
  value: […]
}}

{
  status: “error”,
  message: “404”,
  value?: […]
}
```




```
{
  status: “initialized”
}

{
  status: “loading”,
  error?: “404”,
  value?: […]
}

{
  status: “success”,
  value: […]
}

{
  status: “error”,
  message: “404”,
  value?: […]
}
```

```
function newAjaxValue() {
```

```
function newAjaxValue() {
  return {status: “initialized”};
```

```
function newAjaxValue() {
  return {status: “initialized”};
}
```

```
function newAjaxValue() {
  return {status: “initialized”};
}
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return Object.assign({},
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return Object.assign({},
                       ajaxValue,
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return Object.assign({},
                       ajaxValue,
                       {status: “loading”});
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return Object.assign({},
                       ajaxValue,
                       {status: “loading”});
}
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return Object.assign({},
                       ajaxValue,
                       {status: “loading”});
}
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return Object.assign({},
                       ajaxValue,
                       {status: “loading”});
}

function succeed(ajaxValue, value) {
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return Object.assign({},
                       ajaxValue,
                       {status: “loading”});
}

function succeed(ajaxValue, value) {
  return {status: “success”,
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return Object.assign({},
                       ajaxValue,
                       {status: “loading”});
}

function succeed(ajaxValue, value) {
  return {status: “success”,
          value: value};
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return Object.assign({},
                       ajaxValue,
                       {status: “loading”});
}

function succeed(ajaxValue, value) {
  return {status: “success”,
          value: value};
}
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return Object.assign({},
                       ajaxValue,
                       {status: “loading”});
}

function succeed(ajaxValue, value) {
  return {status: “success”,
          value: value};
}
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return Object.assign({},
                       ajaxValue,
                       {status: “loading”});
}

function succeed(ajaxValue, value) {
  return {status: “success”,
          value: value};
}

function error(ajaxValue, message) {
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return Object.assign({},
                       ajaxValue,
                       {status: “loading”});
}

function succeed(ajaxValue, value) {
  return {status: “success”,
          value: value};
}

function error(ajaxValue, message) {
  return Object.assign({},
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return Object.assign({},
                       ajaxValue,
                       {status: “loading”});
}

function succeed(ajaxValue, value) {
  return {status: “success”,
          value: value};
}

function error(ajaxValue, message) {
  return Object.assign({},
                       ajaxValue,
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return Object.assign({},
                       ajaxValue,
                       {status: “loading”});
}

function succeed(ajaxValue, value) {
  return {status: “success”,
          value: value};
}

function error(ajaxValue, message) {
  return Object.assign({},
                       ajaxValue,
                       {status: “error”,
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return Object.assign({},
                       ajaxValue,
                       {status: “loading”});
}

function succeed(ajaxValue, value) {
  return {status: “success”,
          value: value};
}

function error(ajaxValue, message) {
  return Object.assign({},
                       ajaxValue,
                       {status: “error”,
                        message: message});
```

```
function newAjaxValue() {
  return {status: “initialized”};
}

function request(ajaxValue) {
  return Object.assign({},
                       ajaxValue,
                       {status: “loading”});
}

function succeed(ajaxValue, value) {
  return {status: “success”,
          value: value};
}

function error(ajaxValue, message) {
  return Object.assign({},
                       ajaxValue,
                       {status: “error”,
                        message: message});
}
```

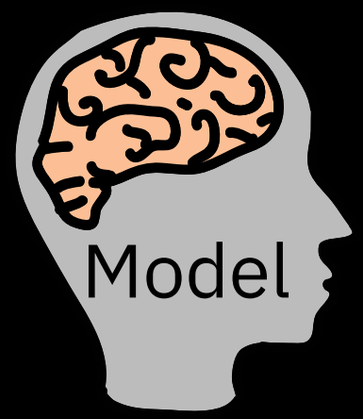
Any aspect of your platform can be modeled

Model the complexity

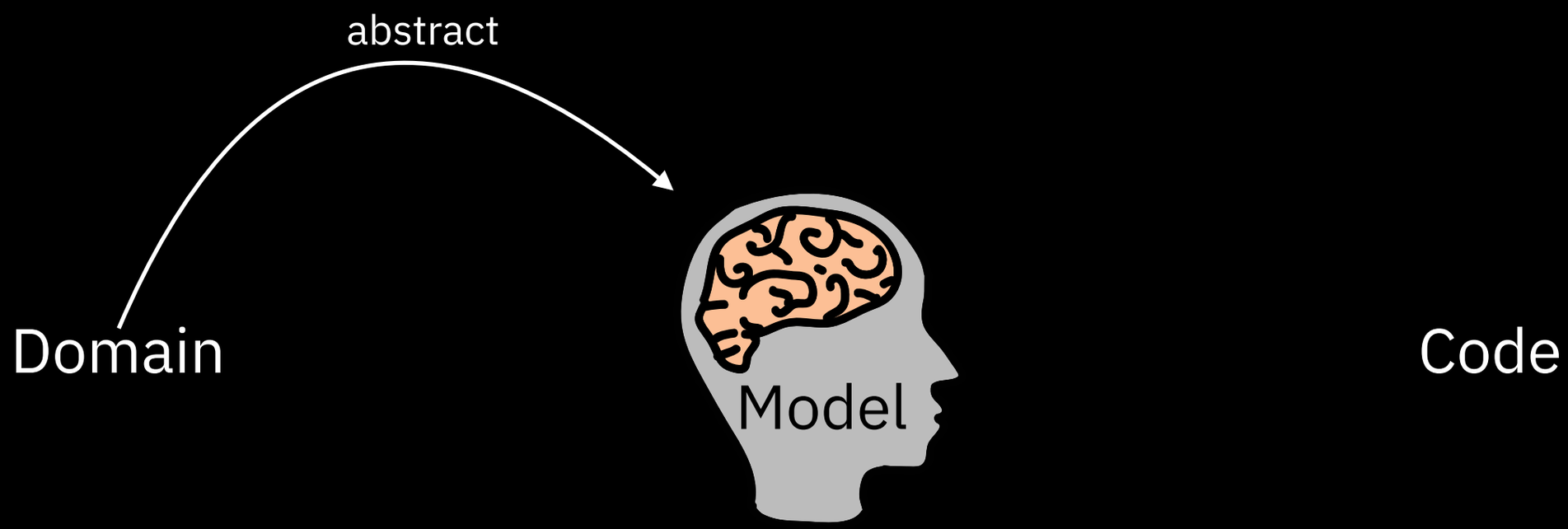
- File IO
- Databases
- Threads/concurrency
- Exceptions

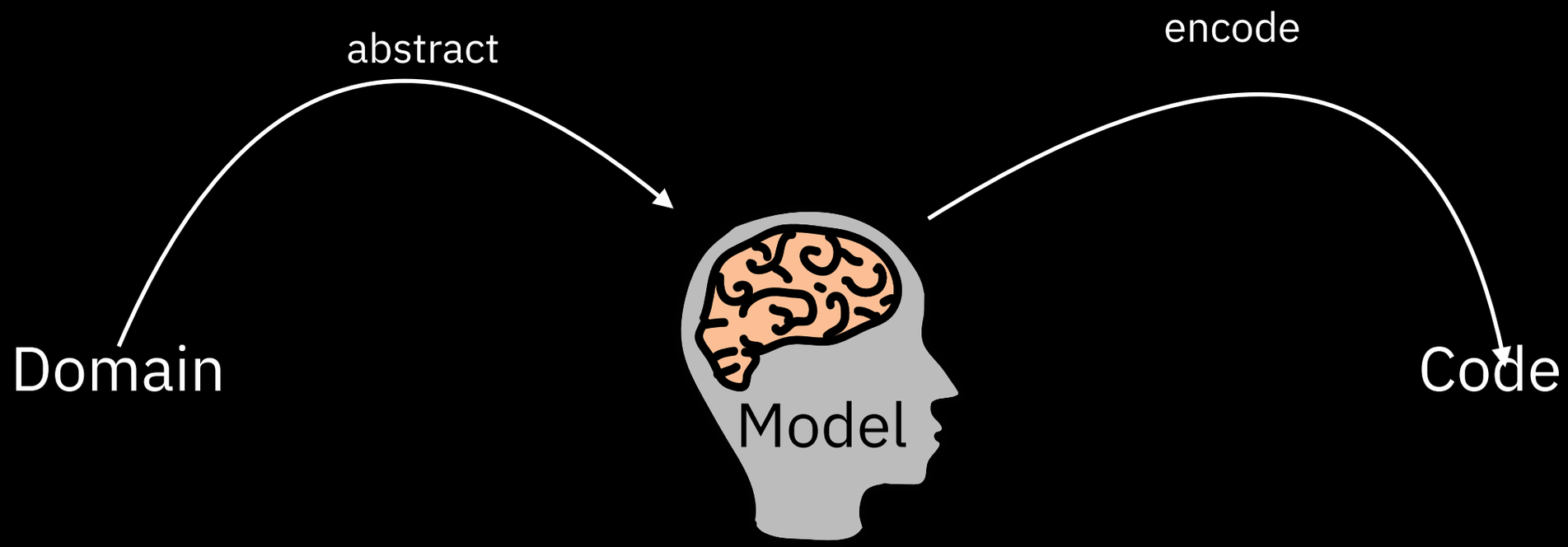
Runnable specifications

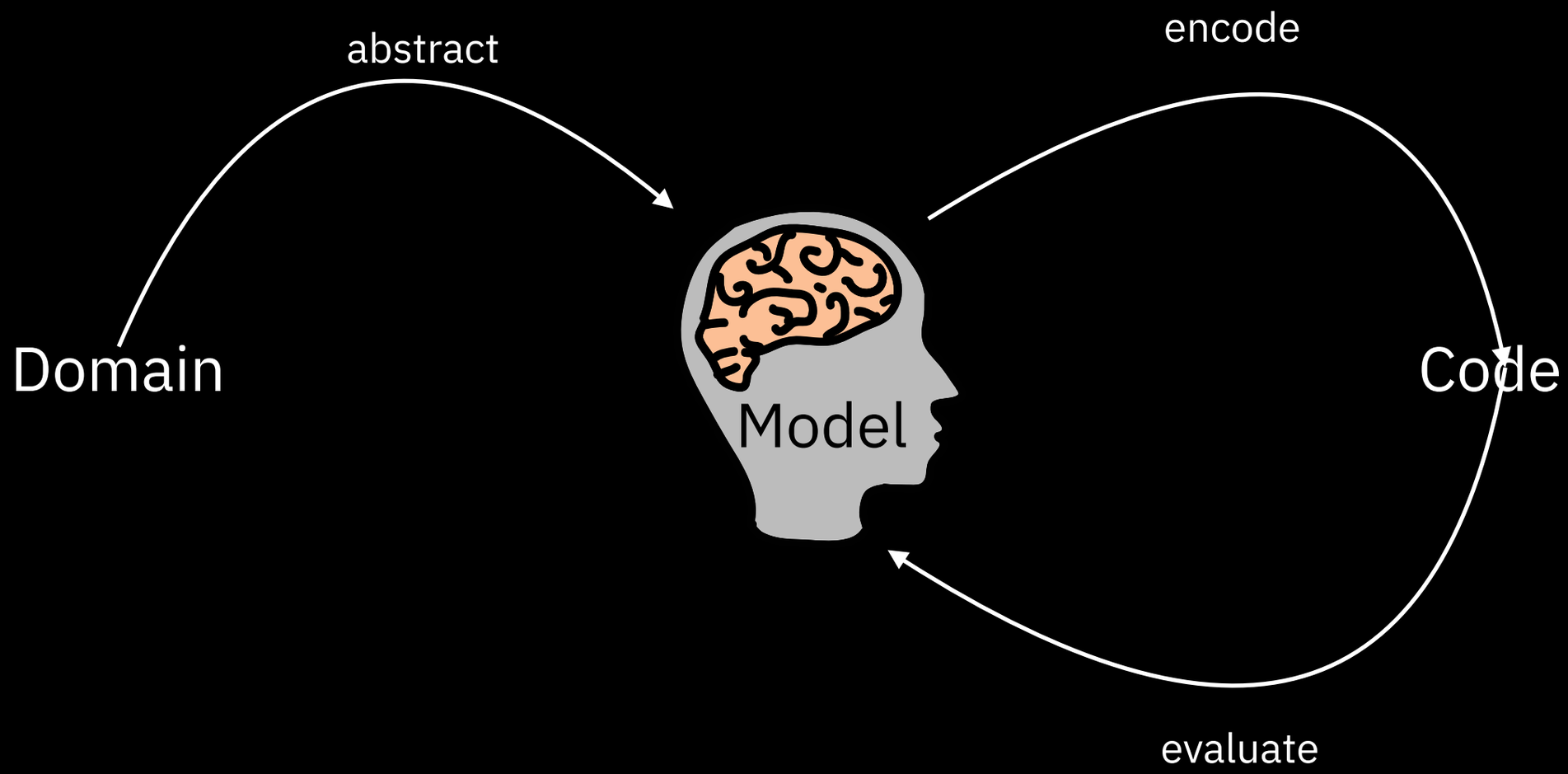
Domain

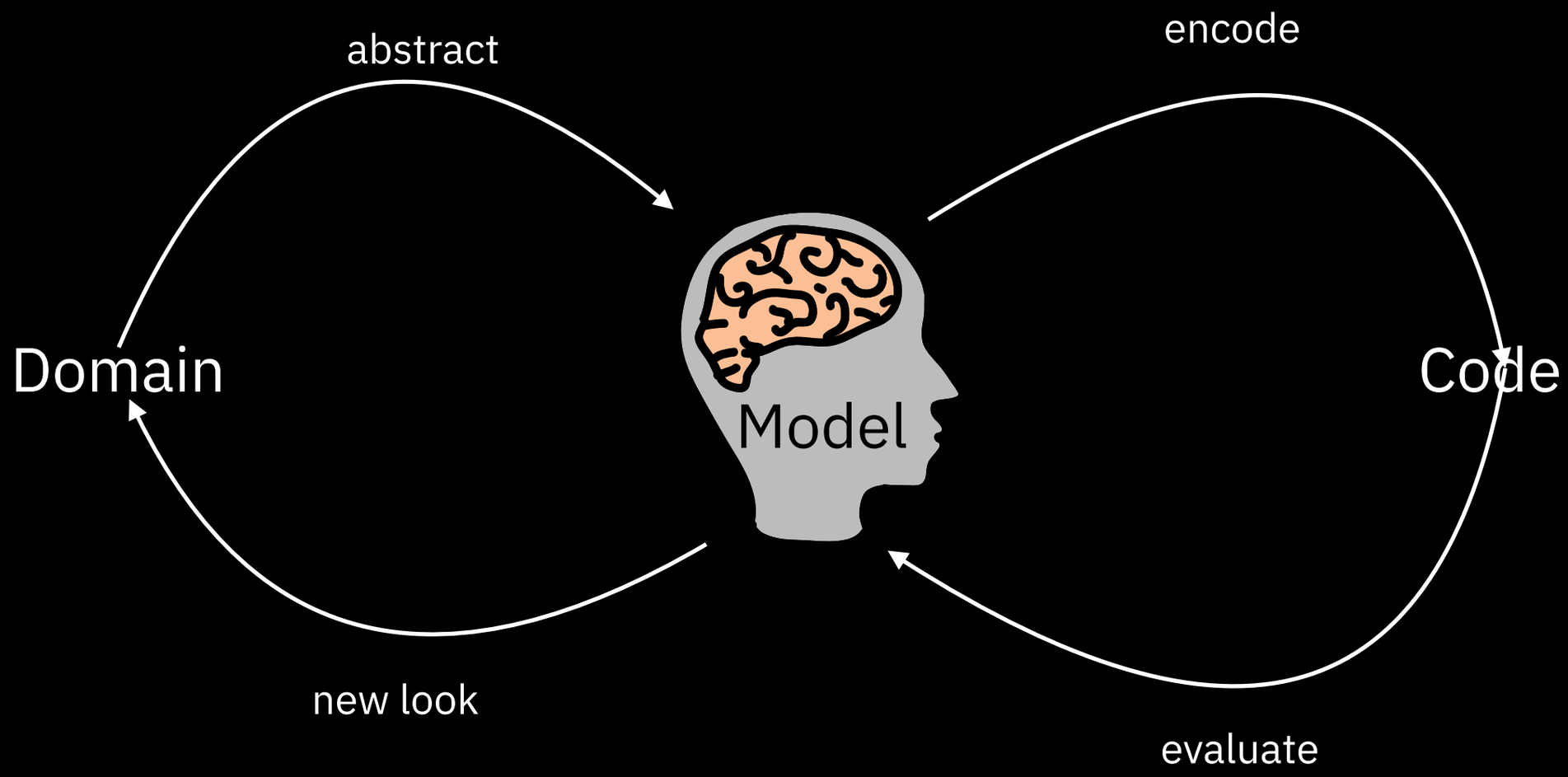


Code


abstract
Domain
Model
Code


abstract
encode
Domain
Model
Code

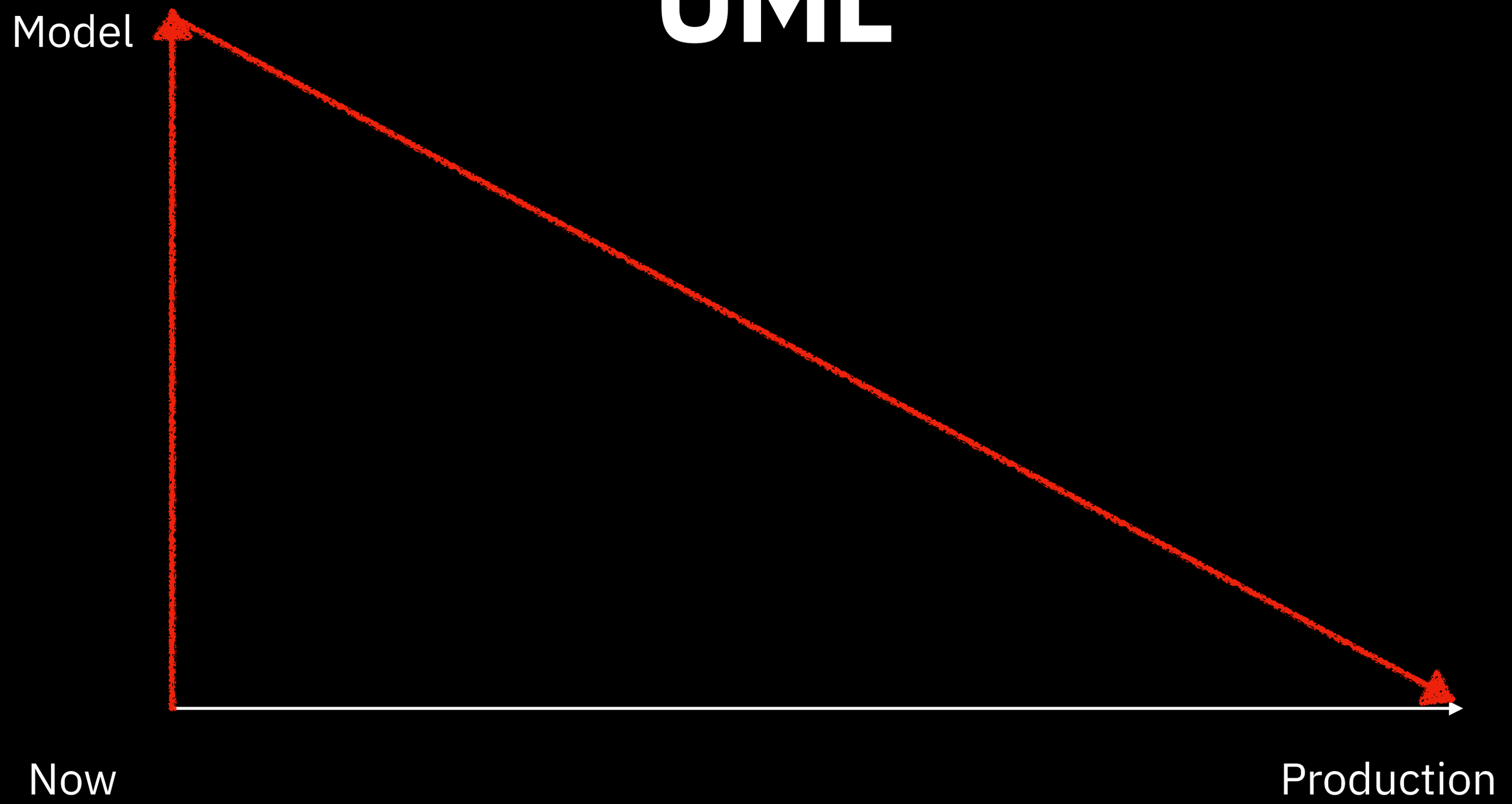

abstract
encode
Domain
Model
Code
evaluate


abstract
encode
Domain
Model
Code
new look
evaluate

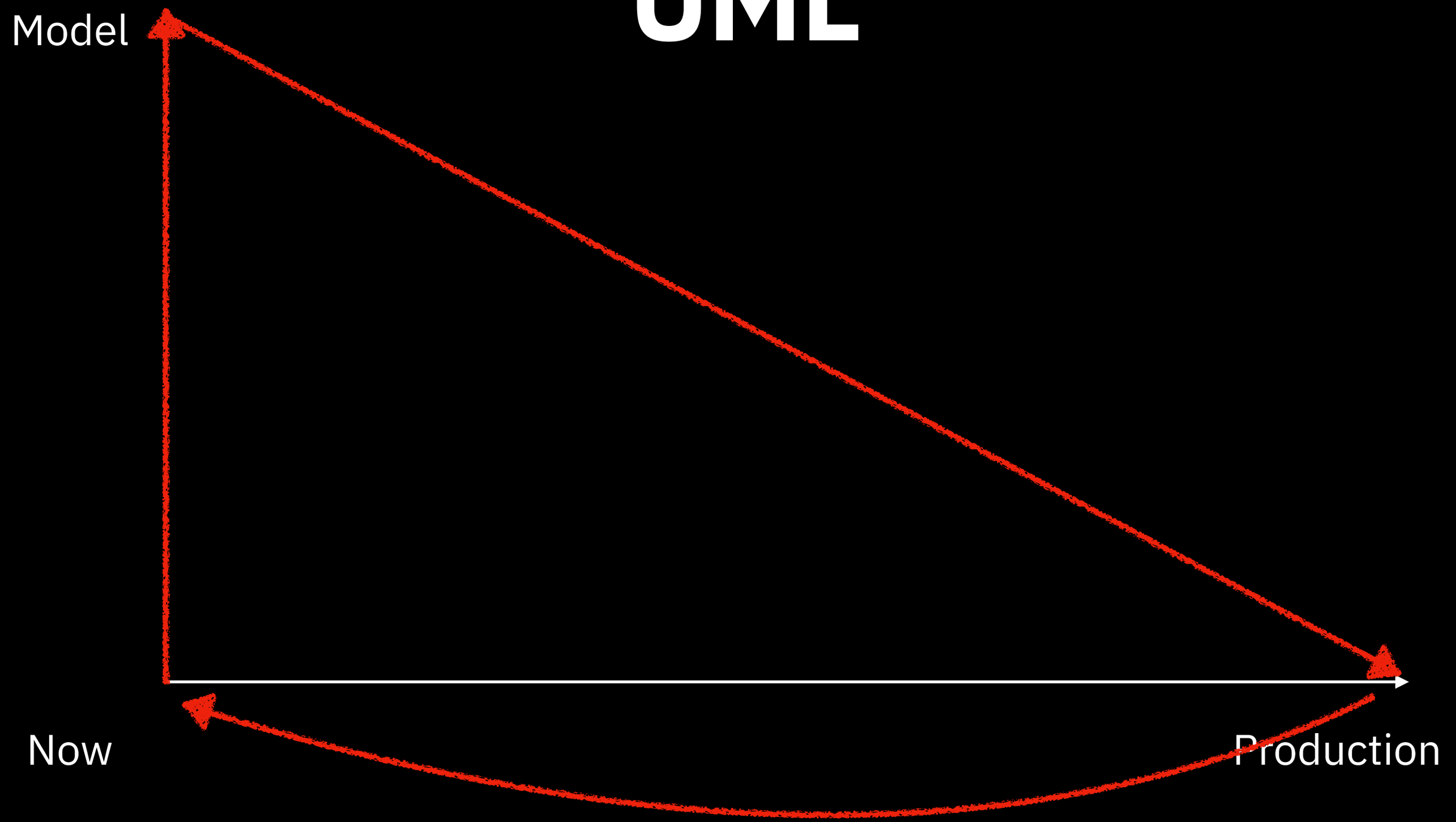
UML

UML

UML



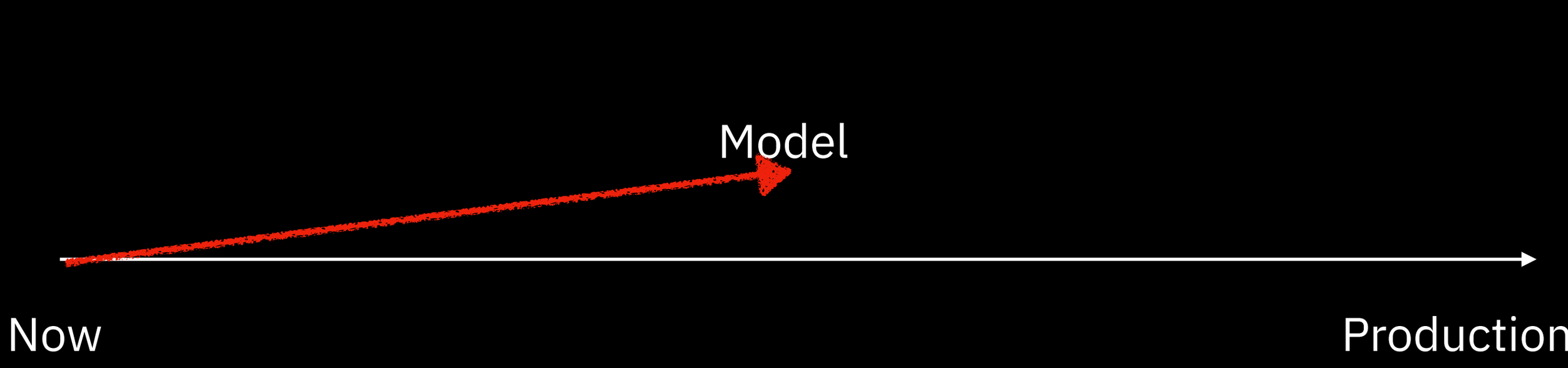
UML



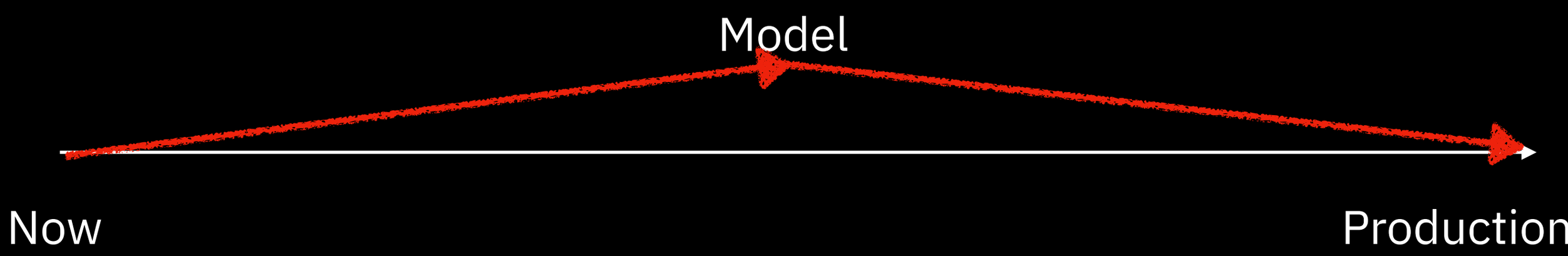
UML

Runnable Specifications

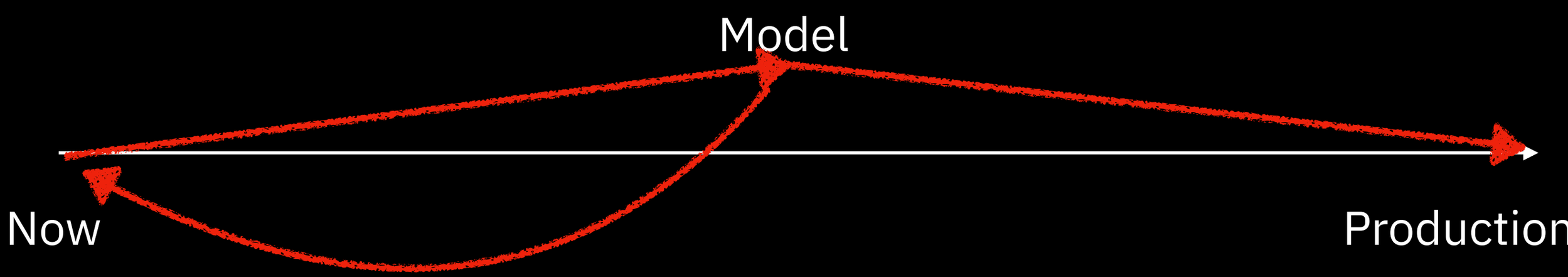
Runnable Specifications



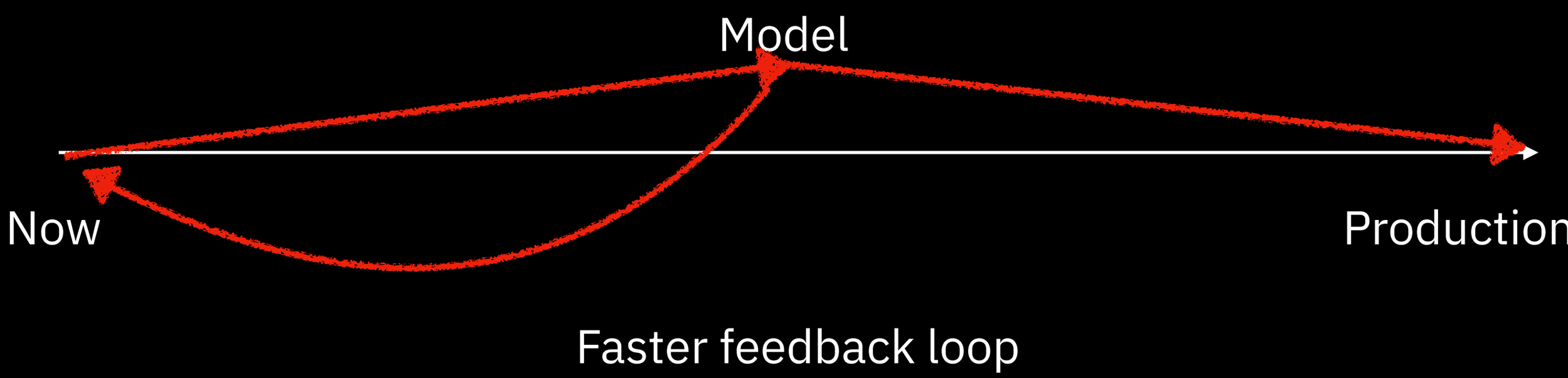
Runnable Specifications



Runnable Specifications



Runnable Specifications



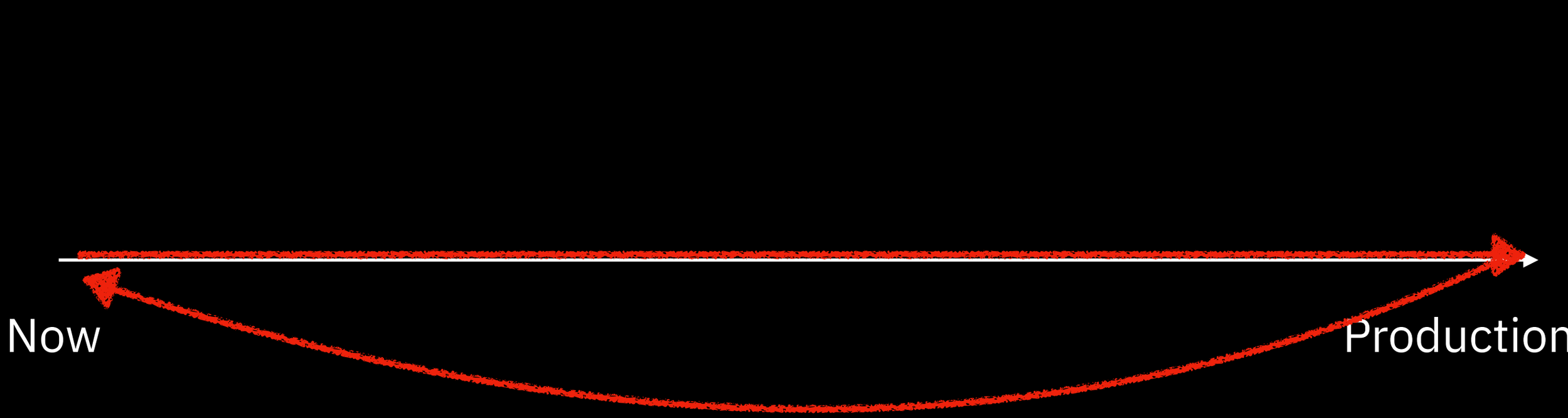
No design

No design

Now

Production

No design



No design

Runnable specifications

Doing design in your language

Runnable specifications

Doing design in your language

- Make in-memory model

Runnable specifications

Doing design in your language

- Make in-memory model
- Run tests on the model

Runnable specifications

Doing design in your language

- Make in-memory model
- Run tests on the model
- Refactor model into a production implementation

Runnable specifications

Doing design in your language

- Make in-memory model
- Run tests on the model
- Refactor model into a production implementation
- Use model as an oracle to test implementation

Make in-memory models

Make in-memory models

- Skip platform-specific stuff

Make in-memory models

- Skip platform-specific stuff
  - No DB, no async, etc. Too complicated

Make in-memory models

- Skip platform-specific stuff
  - No DB, no async, etc. Too complicated
- Use in-memory data and functions

Make in-memory models

- Skip platform-specific stuff
  - No DB, no async, etc. Too complicated
- Use in-memory data and functions
  - JSON, types, objects, structs, etc

Make in-memory models

- Skip platform-specific stuff
  - No DB, no async, etc. Too complicated
- Use in-memory data and functions
  - JSON, types, objects, structs, etc
- Figure out how to tell if they’re working

Make in-memory models

- Skip platform-specific stuff
  - No DB, no async, etc. Too complicated
- Use in-memory data and functions
  - JSON, types, objects, structs, etc
- Figure out how to tell if they’re working
  - visualize them

Make in-memory models

- Skip platform-specific stuff
  - No DB, no async, etc. Too complicated
- Use in-memory data and functions
  - JSON, types, objects, structs, etc
- Figure out how to tell if they’re working
  - visualize them
  - tests

Run tests on model

Ensure the behavior and properties you want

Run tests on model

Ensure the behavior and properties you want

- Manual testing

Run tests on model

Ensure the behavior and properties you want

- Manual testing
- Automated testing

Run tests on model

Ensure the behavior and properties you want

- Manual testing
- Automated testing
- Fast to run lots of tests

Run tests on model

Ensure the behavior and properties you want

- Manual testing
- Automated testing
- Fast to run lots of tests
- Little investment if problem is found

Refactor model into implementation

Refactor model into implementation

- Stepwise convert

Refactor model into implementation

- Stepwise convert
  - JSON into DB schemas

Refactor model into implementation

- Stepwise convert
  - JSON into DB schemas
  - Function calls into AJAX calls

Refactor model into implementation

- Stepwise convert
  - JSON into DB schemas
  - Function calls into AJAX calls
  - etc.

Refactor model into implementation

- Stepwise convert
  - JSON into DB schemas
  - Function calls into AJAX calls
  - etc.
- Ideally, this wouldn’t exist

Use model to test implementation

Use model to test implementation

- Run model and implementation with same input

Use model to test implementation

- Run model and implementation with same input
- Compare output (should be the same)

Newsletter

ericnormand.substack.com

Coffee cup Image by Freepik

https://www.freepik.com/free-vector/list-different-types-coffee_951047.htm