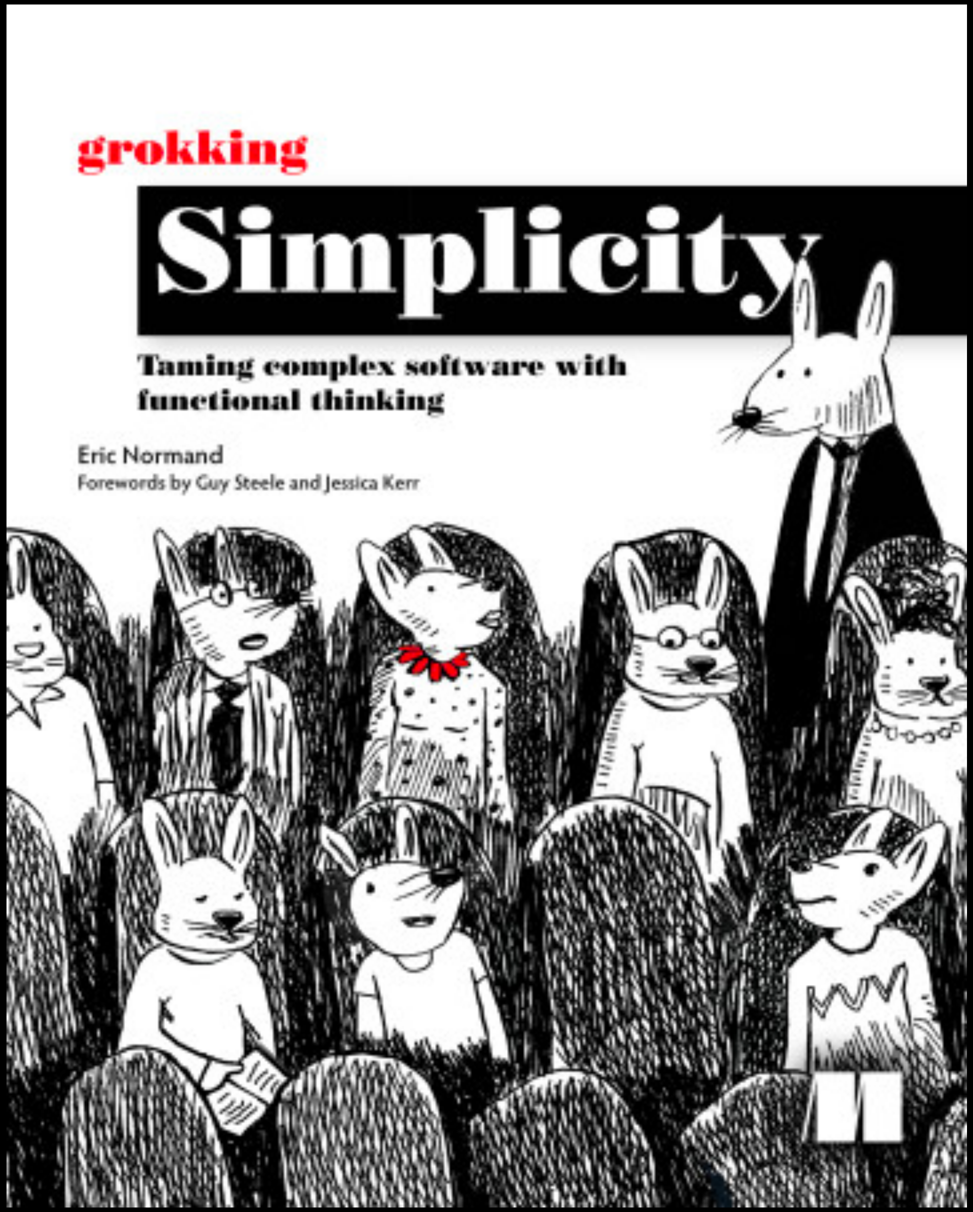
Stratified design and functional architecture

Eric Normand - Øredev 2023


grokking
Simplicity
Taming complex software with functional thinking
Eric Normand
Forewords by Guy Steele and Jessica Kerr

ericnormand.me/gs

40% off: `grokdev23`

Pure functions ✚ Stratified design → Onion architecture

```
sendEmail(to, from, subject, body)

saveUserDB(user)

getCurrentTime()
```

```
sum(numbers)

stringLength(str)
```

```
{“firstname”: “Eric”,
 “lastname”: “Normand”}

[1, 10, 2, 45, 3, 98]
```

```
sendEmail(to, from, subject, body)

saveUserDB(user)

getCurrentTime()
```

```
sum(numbers)

stringLength(str)


{“firstname”: “Eric”,
 “lastname”: “Normand”}

[1, 10, 2, 45, 3, 98]
```

Actions

```
sendEmail(to, from, subject, body)

saveUserDB(user)

getCurrentTime()
```

```
sum(numbers)

stringLength(str)
```

```
{“firstname”: “Eric”,
 “lastname”: “Normand”}

[1, 10, 2, 45, 3, 98]
```

Actions

```
sendEmail(to, from, subject, body)

saveUserDB(user)

getCurrentTime()
```

```
sum(numbers)

stringLength(str)
```

```
{“firstname”: “Eric”,
 “lastname”: “Normand”}

[1, 10, 2, 45, 3, 98]
```

Actions

```
sendEmail(to, from, subject, body)

saveUserDB(user)

getCurrentTime()
```

Calculations

```
sum(numbers)

stringLength(str)
```

```
{“firstname”: “Eric”,
 “lastname”: “Normand”}

[1, 10, 2, 45, 3, 98]
```

Actions

```
sendEmail(to, from, subject, body)

saveUserDB(user)

getCurrentTime()
```

Calculations

```
sum(numbers)

stringLength(str)
```

Data

```
{“firstname”: “Eric”,
 “lastname”: “Normand”}

[1, 10, 2, 45, 3, 98]
```

Data

Facts about events.

Data

Facts about events.

- Numbers

Data

Facts about events.

- Numbers
- Strings

Data

Facts about events.

- Numbers
- Strings
- Enums

Data

Facts about events.

- Numbers
- Strings
- Enums
- Collections

Data

Facts about events.

- Numbers
- Strings
- Enums
- Collections
- Etc.

Calculations

Computations from input to output.

Calculations

Computations from input to output.

- Also known as “pure functions” or “mathematical functions”.

Calculations

Computations from input to output.

- Also known as “pure functions” or “mathematical functions”.
- Examples

Calculations

Computations from input to output.

- Also known as “pure functions” or “mathematical functions”.
- Examples
    - +, *, -, /

Calculations

Computations from input to output.

- Also known as “pure functions” or “mathematical functions”.
- Examples
    - +, *, -, /
    - Math.abs()

Calculations

Computations from input to output.

- Also known as “pure functions” or “mathematical functions”.
- Examples
  - +, *, -, /
  - Math.abs()
  - String concatenation

Calculations

Computations from input to output.

- Also known as “pure functions” or “mathematical functions”.
- Examples
  - +, *, -, /
  - Math.abs()
  - String concatenation
  - Validate an email address

Actions

Affect or are affected by the outside world.

Actions

Affect or are affected by the outside world.

- Also known as “*im*pure functions”, “side-effecting functions”, “functions with side-effects”.

Actions

Affect or are affected by the outside world.

- Also known as “*im*pure functions”, “side-effecting functions”, “functions with side-effects”.

- Rule of thumb: Depend on how many times or when they are run.

Actions

Affect or are affected by the outside world.

- Also known as “*im*pure functions”, “side-effecting functions”, “functions with side-effects”.
- Rule of thumb: Depend on how many times or when they are run.
- Examples

Actions

Affect or are affected by the outside world.

- Also known as “*im*pure functions”, “side-effecting functions”, “functions with side-effects”.
- Rule of thumb: Depend on how many times or when they are run.
- Examples
  - Send an email

Actions

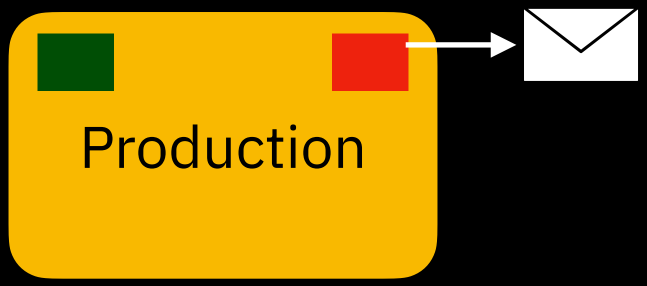
Affect or are affected by the outside world.

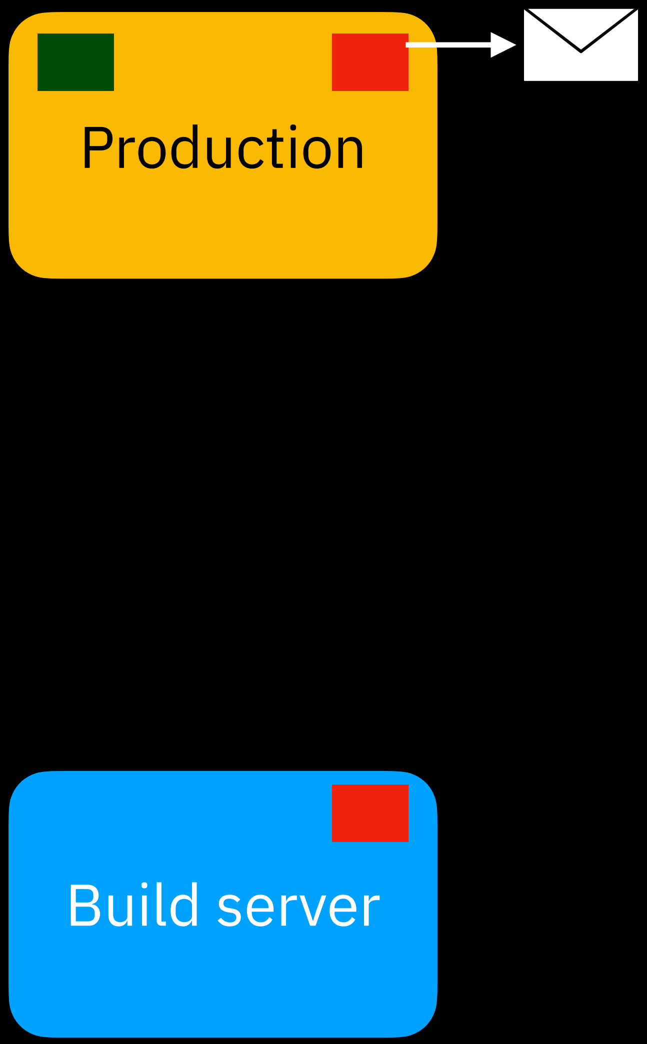
- Also known as “*im*pure functions”, “side-effecting functions”, “functions with side-effects”.
- Rule of thumb: Depend on how many times or when they are run.
- Examples
  - Send an email
  - Read from a database

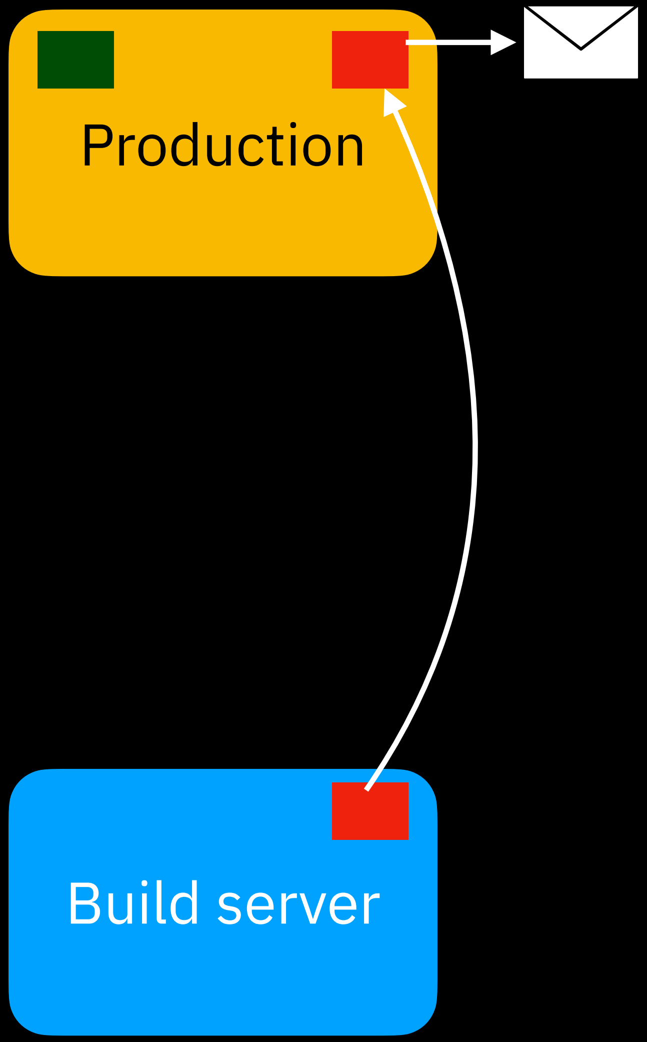
Actions

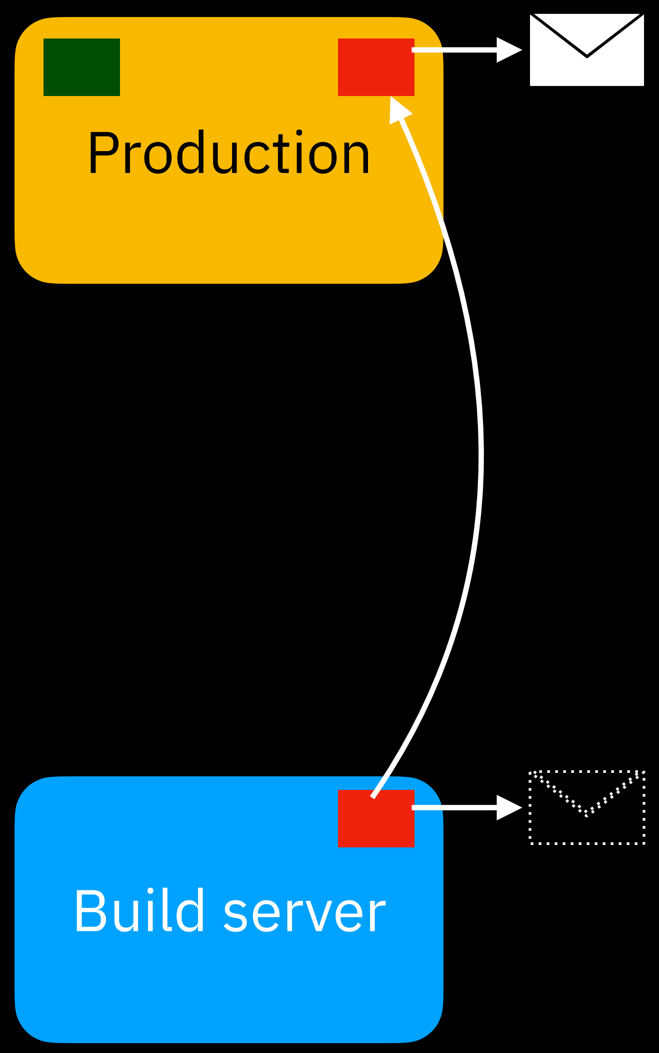
Affect or are affected by the outside world.

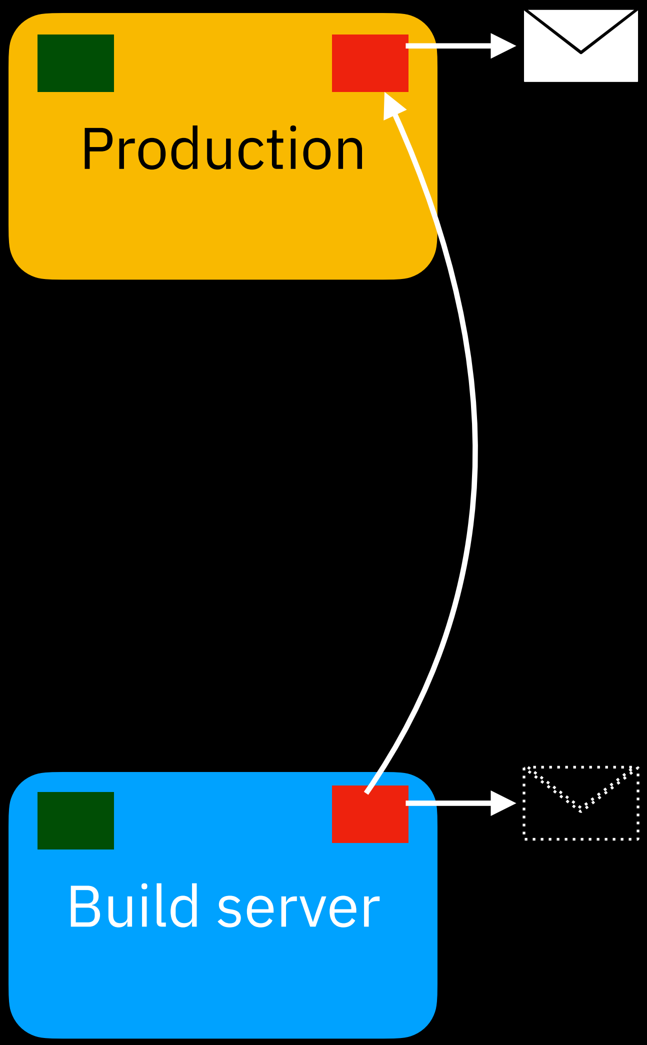
- Also known as “*im*pure functions”, “side-effecting functions”, “functions with side-effects”.
- Rule of thumb: Depend on how many times or when they are run.
- Examples
  - Send an email
  - Read from a database
  - Write to a file

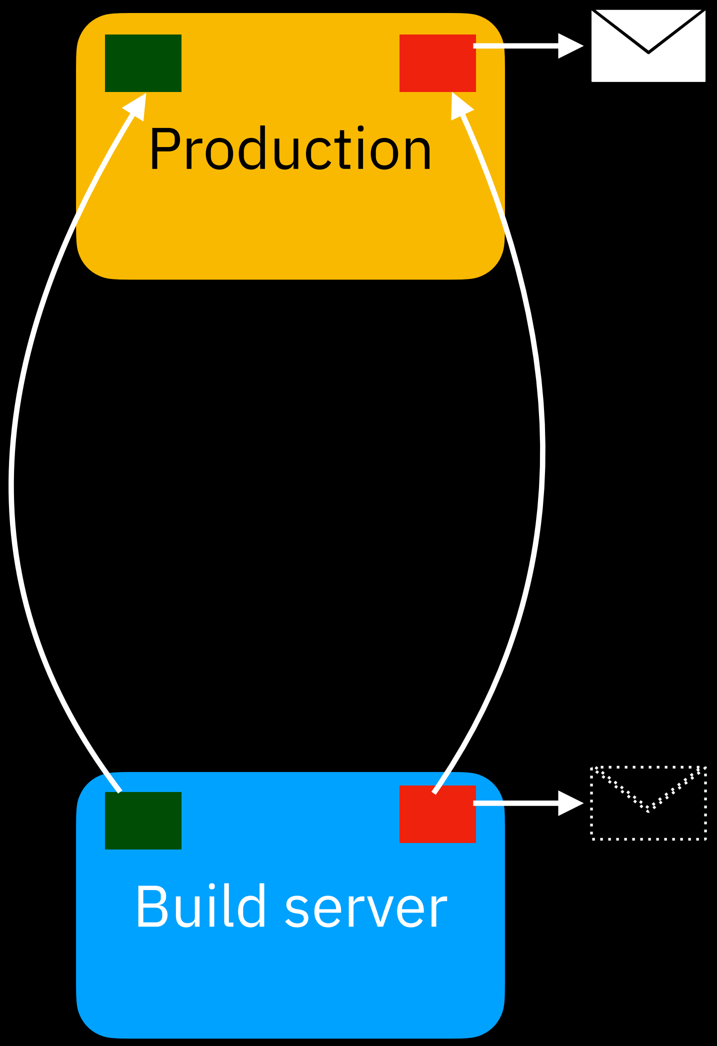

Production


Production
Build server


Production
Build server


Production
Build server


Production
Build server

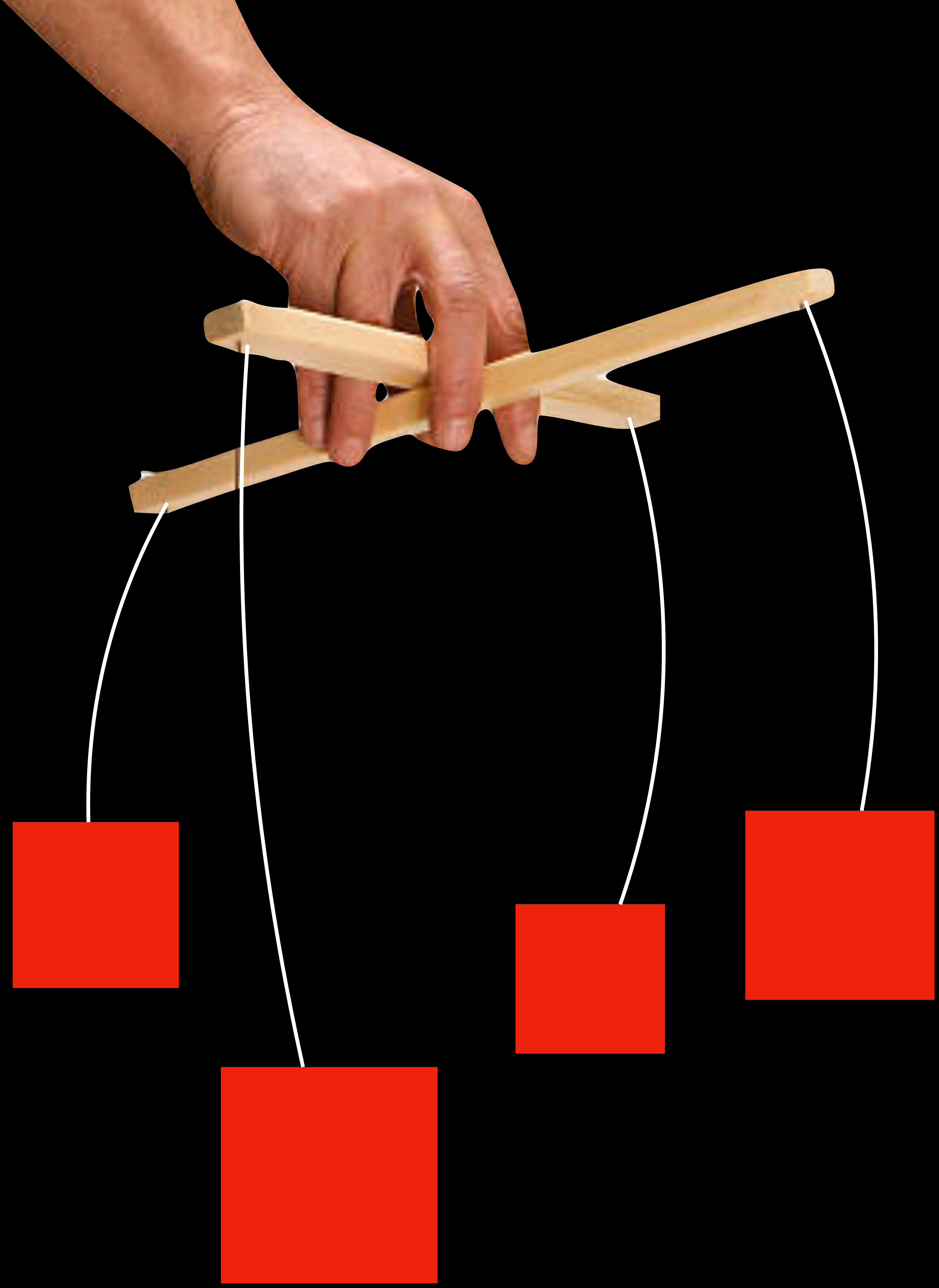

Production
Build server

Actions are harder to run safely in production

Actions are harder to debug

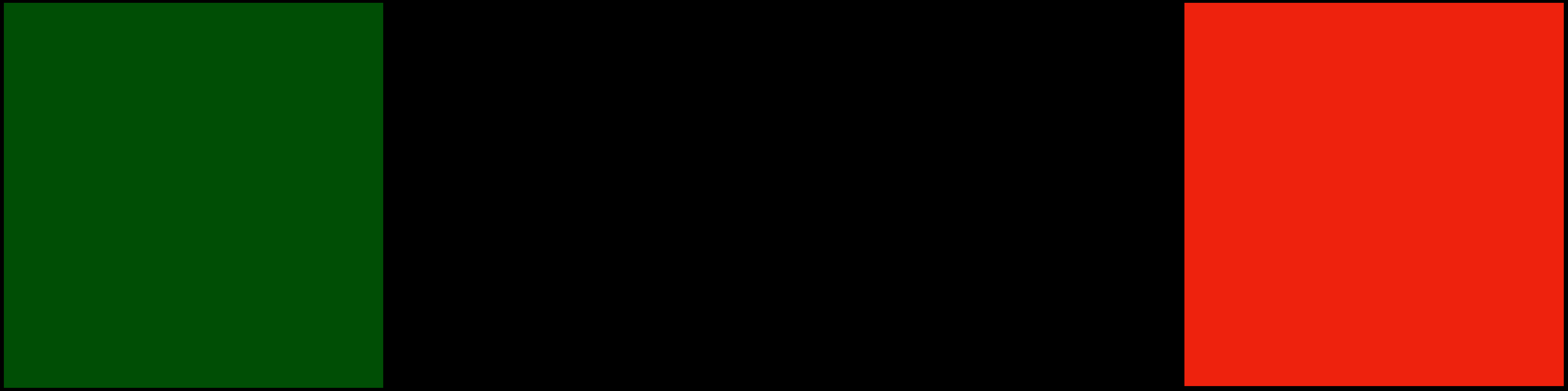
Calculations ☝️ Actions

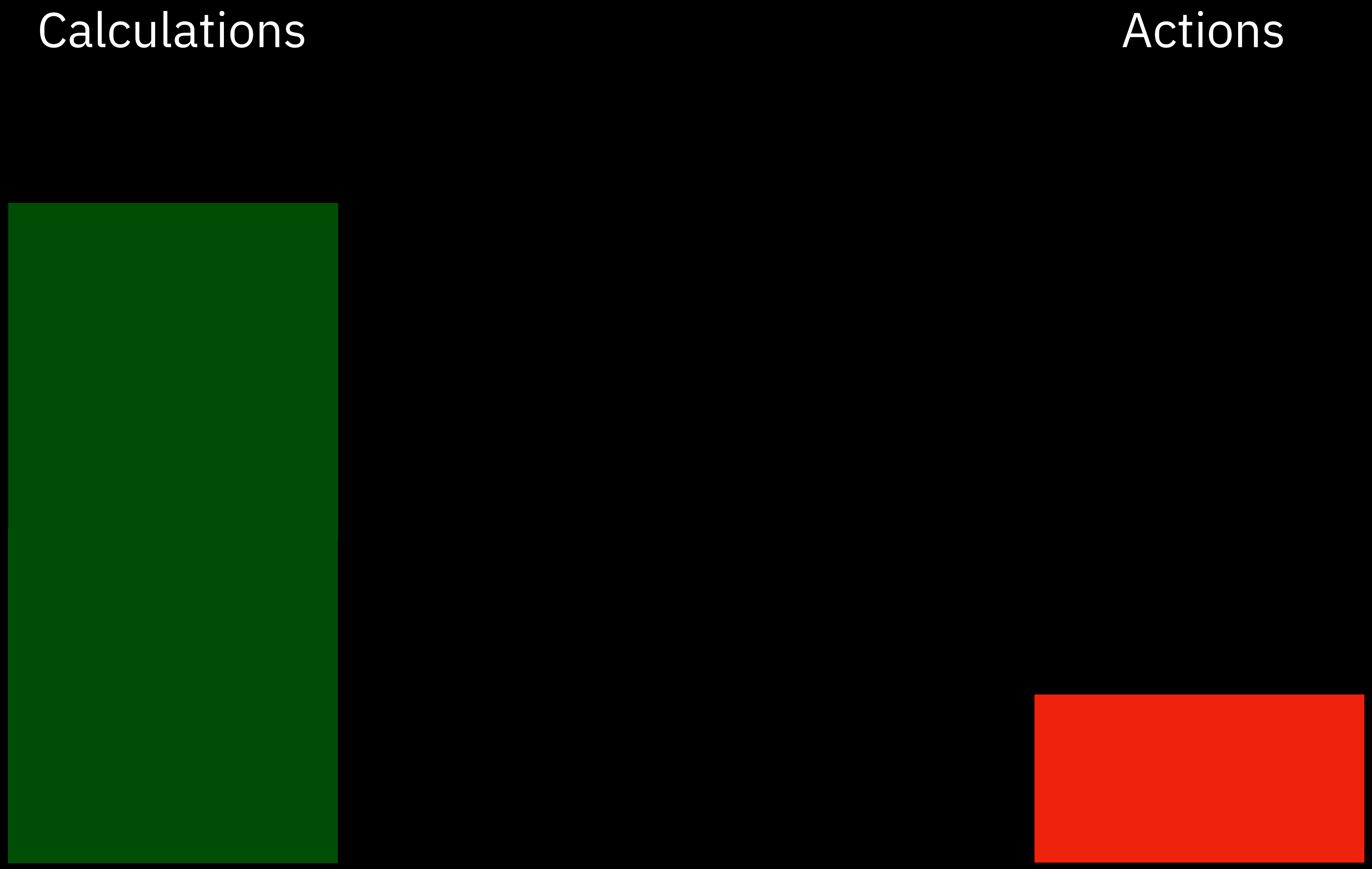
Calculations 👈 Actions



Calculations

Actions




Calculations
Actions

Spreading rule

```
function figurePayout(affiliate) {
  var owed = affiliate.sales * affiliate.commission;
  if(owed > 100)
    sendPayout(affiliate.bank_code, owed);
}

function affiliatePayout(affiliates) {
  for(var a = 0; a < affiliates.length; a++)
    figurePayout(affiliates[a]);
}

function main(affiliates) {
  affiliatePayout(affiliates);
}
```

Spreading rule

```
function figurePayout(affiliate) {
  var owed = affiliate.sales * affiliate.commission;
  if(owed > 100)
    sendPayout(affiliate.bank_code, owed);
}

function affiliatePayout(affiliates) {
  for(var a = 0; a < affiliates.length; a++)
    figurePayout(affiliates[a]);
}

function main(affiliates) {
  affiliatePayout(affiliates);
}
```

Spreading rule

```
function figurePayout(affiliate) {
  var owed = affiliate.sales * affiliate.commission;
  if(owed > 100)
    sendPayout(affiliate.bank_code, owed);
}

function affiliatePayout(affiliates) {
  for(var a = 0; a < affiliates.length; a++)
    figurePayout(affiliates[a]);
}

function main(affiliates) {
  affiliatePayout(affiliates);
}
```

Spreading rule

```
function figurePayout(affiliate) {
  var owed = affiliate.sales * affiliate.commission;
  if(owed > 100)
    sendPayout(affiliate.bank_code, owed);
}

function affiliatePayout(affiliates) {
  for(var a = 0; a < affiliates.length; a++)
    figurePayout(affiliates[a]);
}

function main(affiliates) {
  affiliatePayout(affiliates);
}
```

Spreading rule

```
function figurePayout(affiliate) {
  var owed = affiliate.sales * affiliate.commission;
  if(owed > 100)
    sendPayout(affiliate.bank_code, owed);
}

function affiliatePayout(affiliates) {
  for(var a = 0; a < affiliates.length; a++)
    figurePayout(affiliates[a]);
}

function main(affiliates) {
  affiliatePayout(affiliates);
}
```

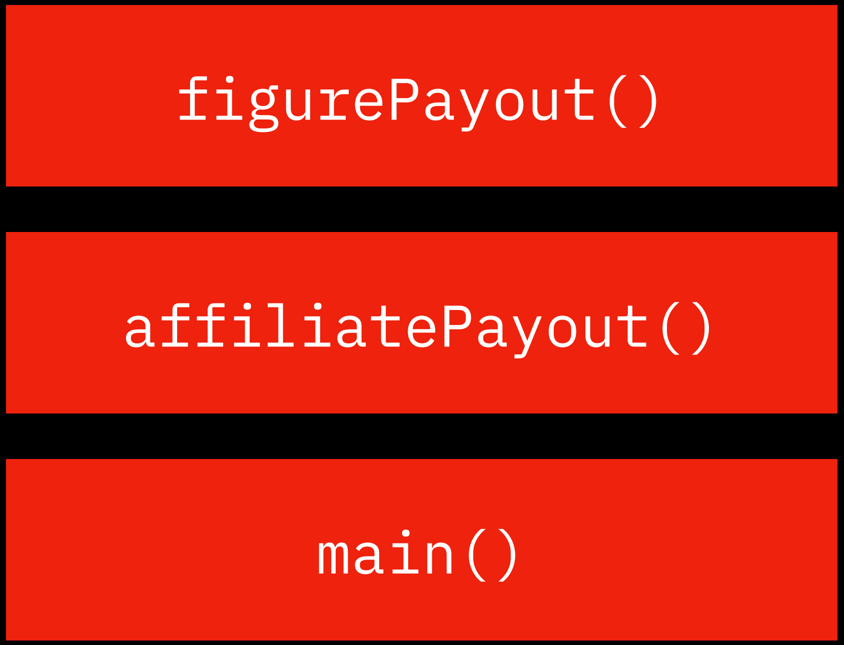
Spreading rule

```
function figurePayout(affiliate) {
  var owed = affiliate.sales * affiliate.commission;
  if(owed > 100)
    sendPayout(affiliate.bank_code, owed);
}
```

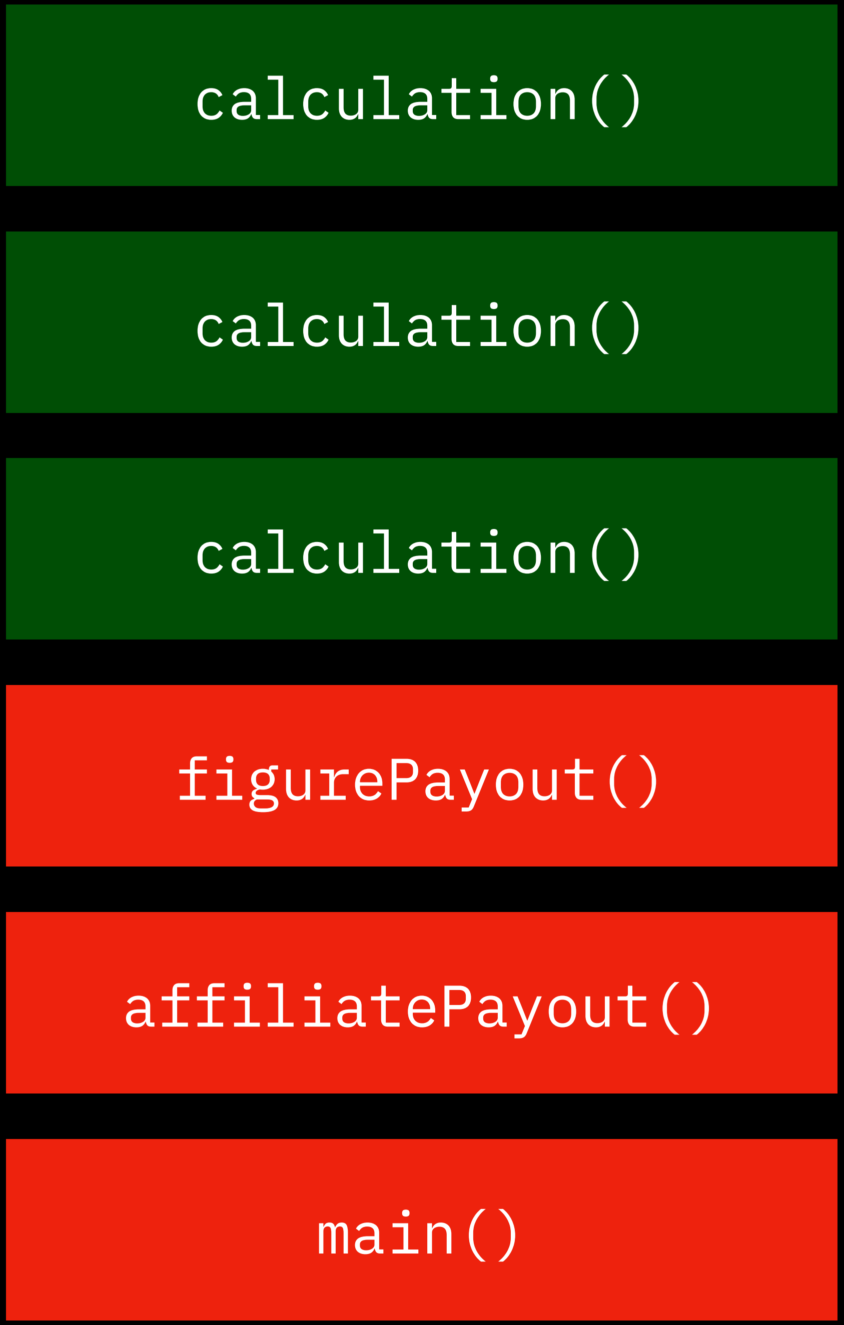
```
function affiliatePayout(affiliates) {
  for(var a = 0; a < affiliates.length; a++)
    figurePayout(affiliates[a]);
}
```

```
function main(affiliates) {
  affiliatePayout(affiliates);
}
```

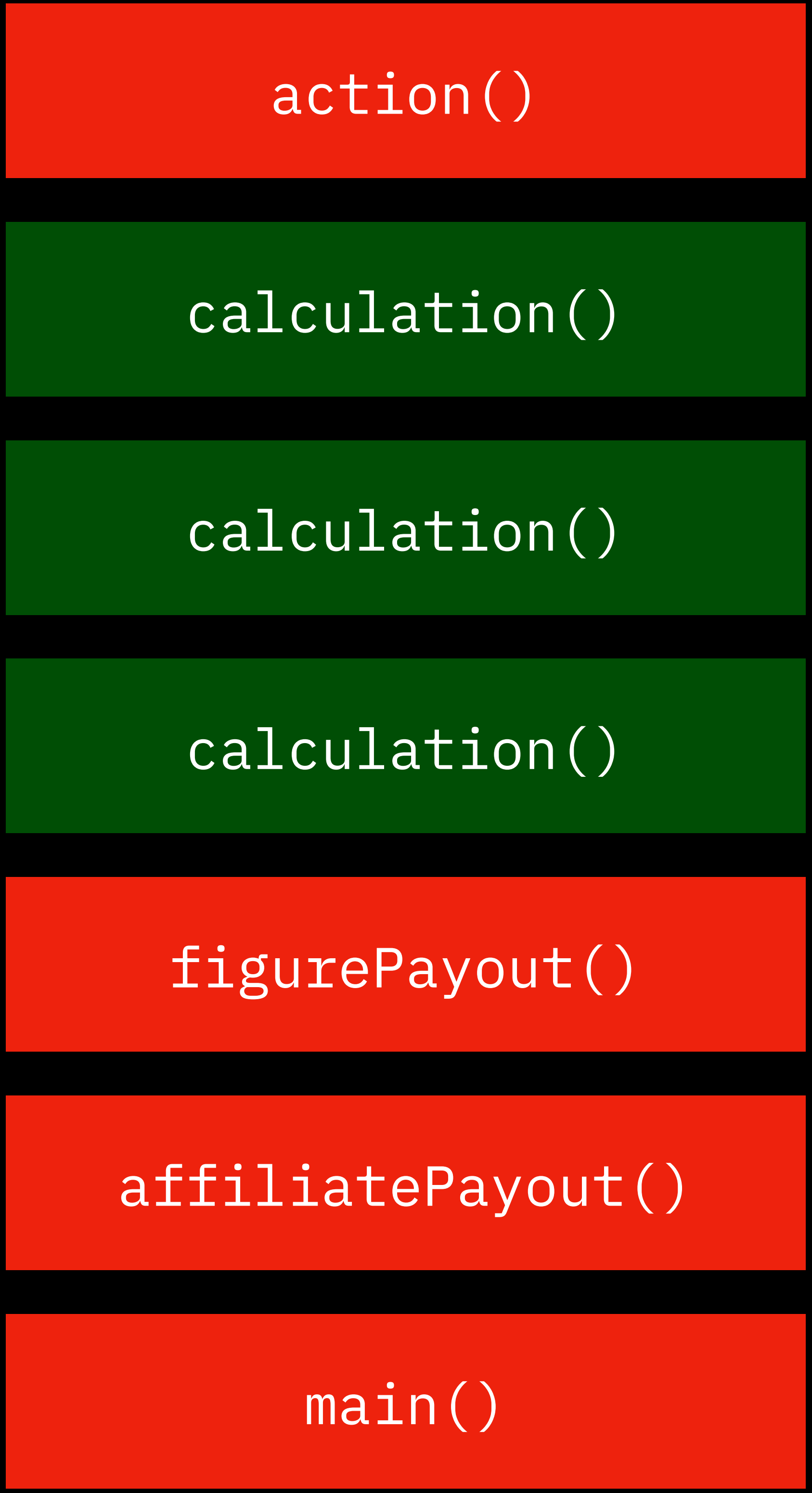
Call stack



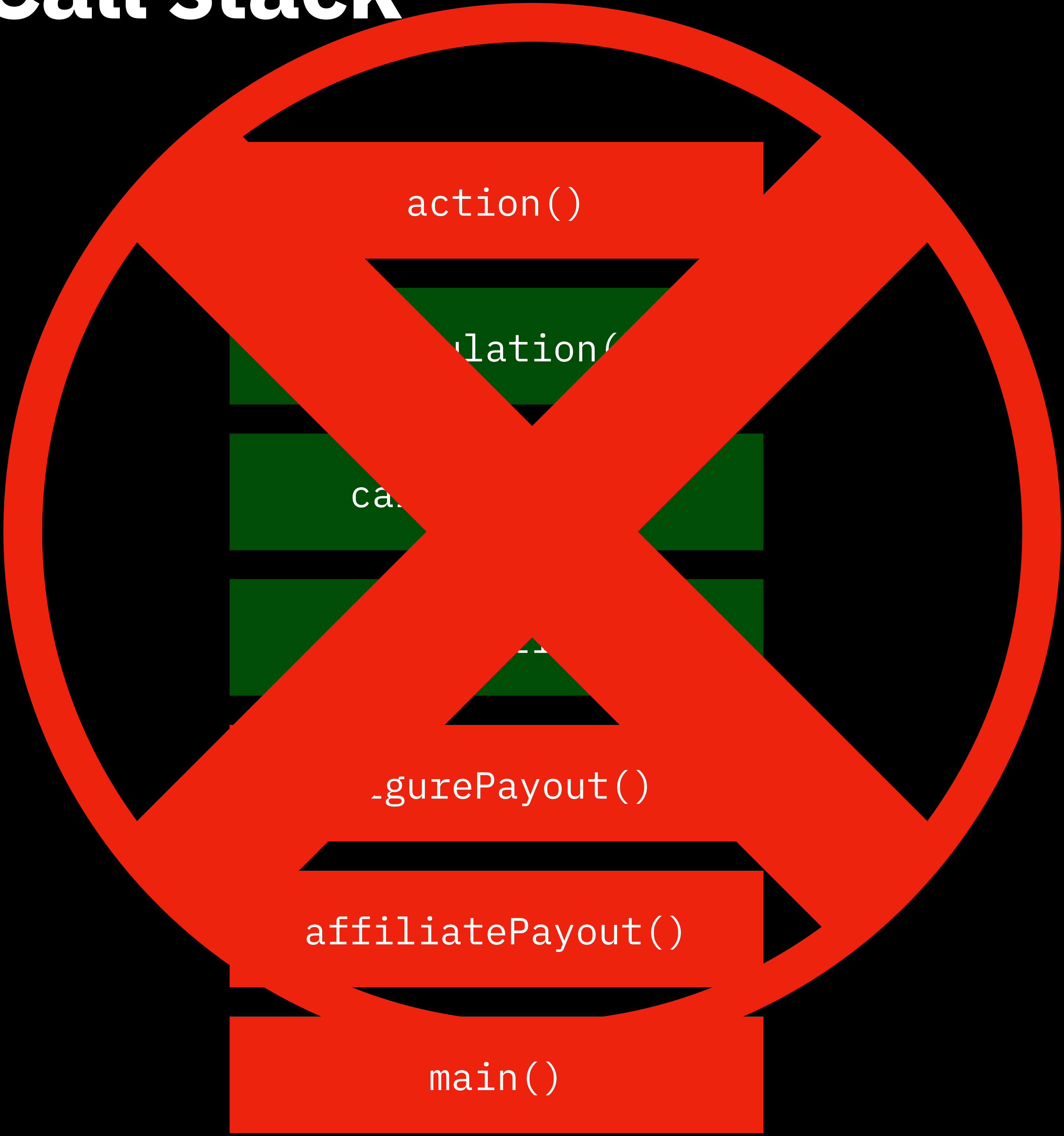
Call stack


calculation()
calculation()
calculation()
figurePayout()
affiliatePayout()
main()

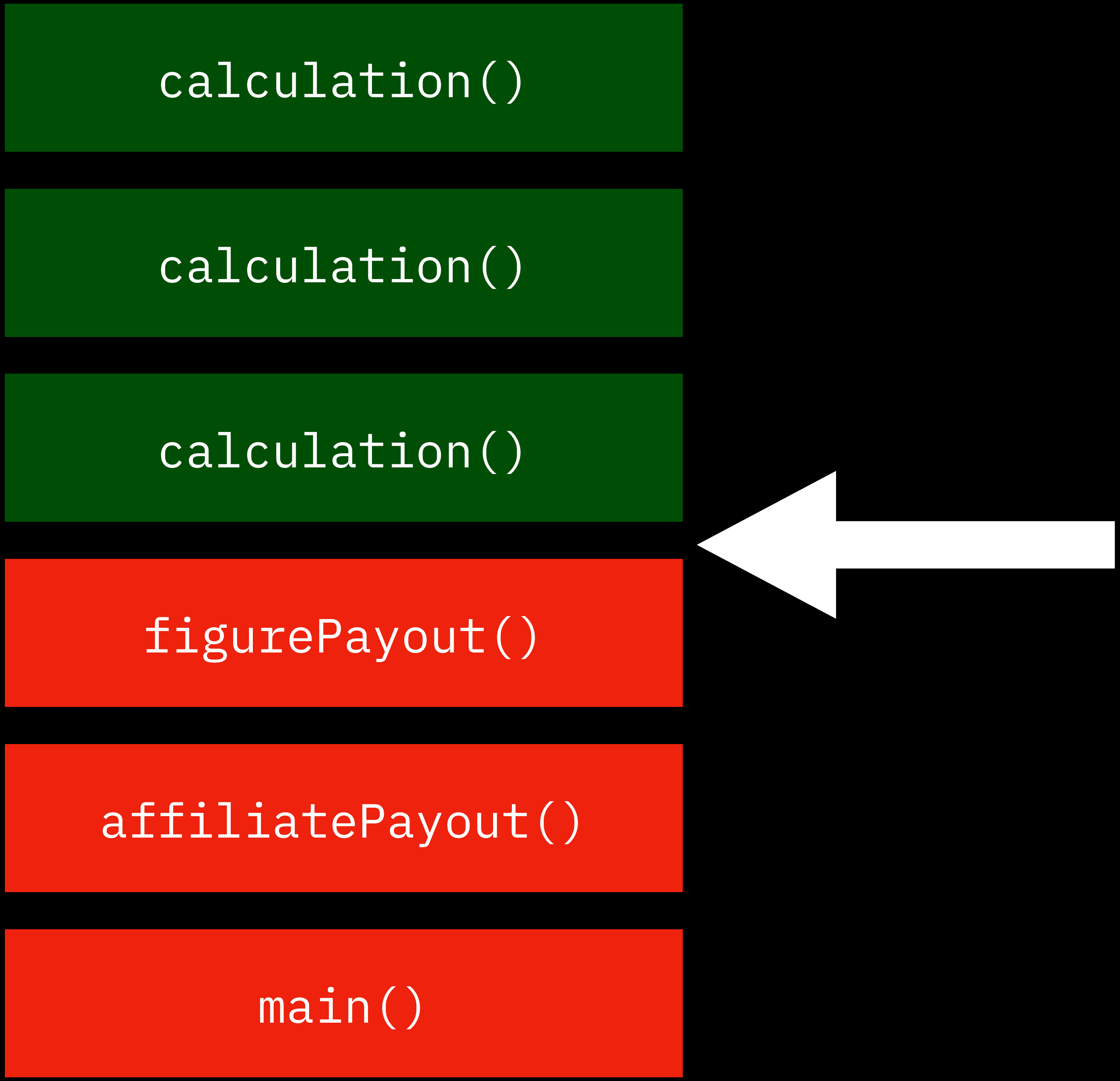
Call stack


action()
calculation()
calculation()
calculation()
figurePayout()
affiliatePayout()
main()

Call stack



Call stack


calculation()
calculation()
calculation()
figurePayout()
affiliatePayout()
main()

Extracting calculations

CouponDog

```
function sendIssue() {
  const coupons     = fetchCouponsFromDB();
  const subscribers = fetchSubscribersFromDB();
  subscribers.forEach((s) => {
    emailSystem.send({
      from: “newsletter@coupondog.co",
      to: s.email,
      subject: “Your best weekly coupons inside”,
      body: “Here are the best coupons: “ +
            coupons.join(“, “)
    });
  });
}
```

Extracting calculations

CouponDog

```
function sendIssue() {
  const coupons     = fetchCouponsFromDB();
  const subscribers = fetchSubscribersFromDB();
  subscribers.forEach((s) => {
    emailSystem.send({
      from: “newsletter@coupondog.co",
      to: s.email,
      subject: “Your best weekly coupons inside”,
      body: “Here are the best coupons: “ +
            coupons.join(“, “)
    });
  });
}
```

Extracting calculations

CouponDog

```
function sendIssue() {
  const coupons     = fetchCouponsFromDB();
  const subscribers = fetchSubscribersFromDB();
  subscribers.forEach((s) => {
    emailSystem.send({
      from: “newsletter@coupondog.co",
      to: s.email,
      subject: “Your best weekly coupons inside”,
      body: “Here are the best coupons: “ +
            coupons.join(“, “)
    });
  });
}
```

```
function emailForSubscriber(subscriber, coupons) {
  return {
    from: “newsletter@coupondog.co",
    to: subscriber.email,
    subject: “Your best weekly coupons inside”,
    body: “Here are the best coupons: “ +
            coupons.join(“, “)
  };
}


function sendIssue() {
  const coupons     = fetchCouponsFromDB();
  const subscribers = fetchSubscribersFromDB();
  subscribers.forEach((s) => {
    emailSystem.send(
      emailForSubscriber(s, coupons)
    );
  });
}
```

```
function emailForSubscriber(subscriber, coupons) {
  return {
    from: “newsletter@coupondog.co",
    to: subscriber.email,
    subject: “Your best weekly coupons inside”,
    body: “Here are the best coupons: “ +
            coupons.join(“, “)
  };
}


function sendIssue() {
  const coupons     = fetchCouponsFromDB();
  const subscribers = fetchSubscribersFromDB();
  const emails      = subscribers.map(
    (s) => emailForSubscriber(s, coupons)
  );
  emails.forEach((e) => emailSystem.send(e));

}
```

Common questions

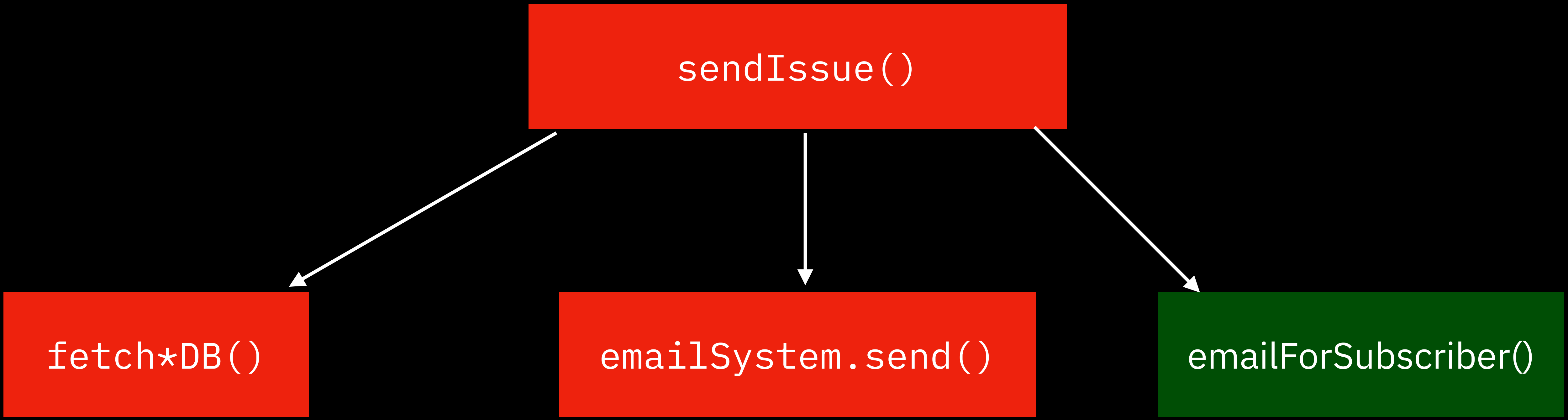
Isn’t it inefficient to create every email?
What if we have billions of users?

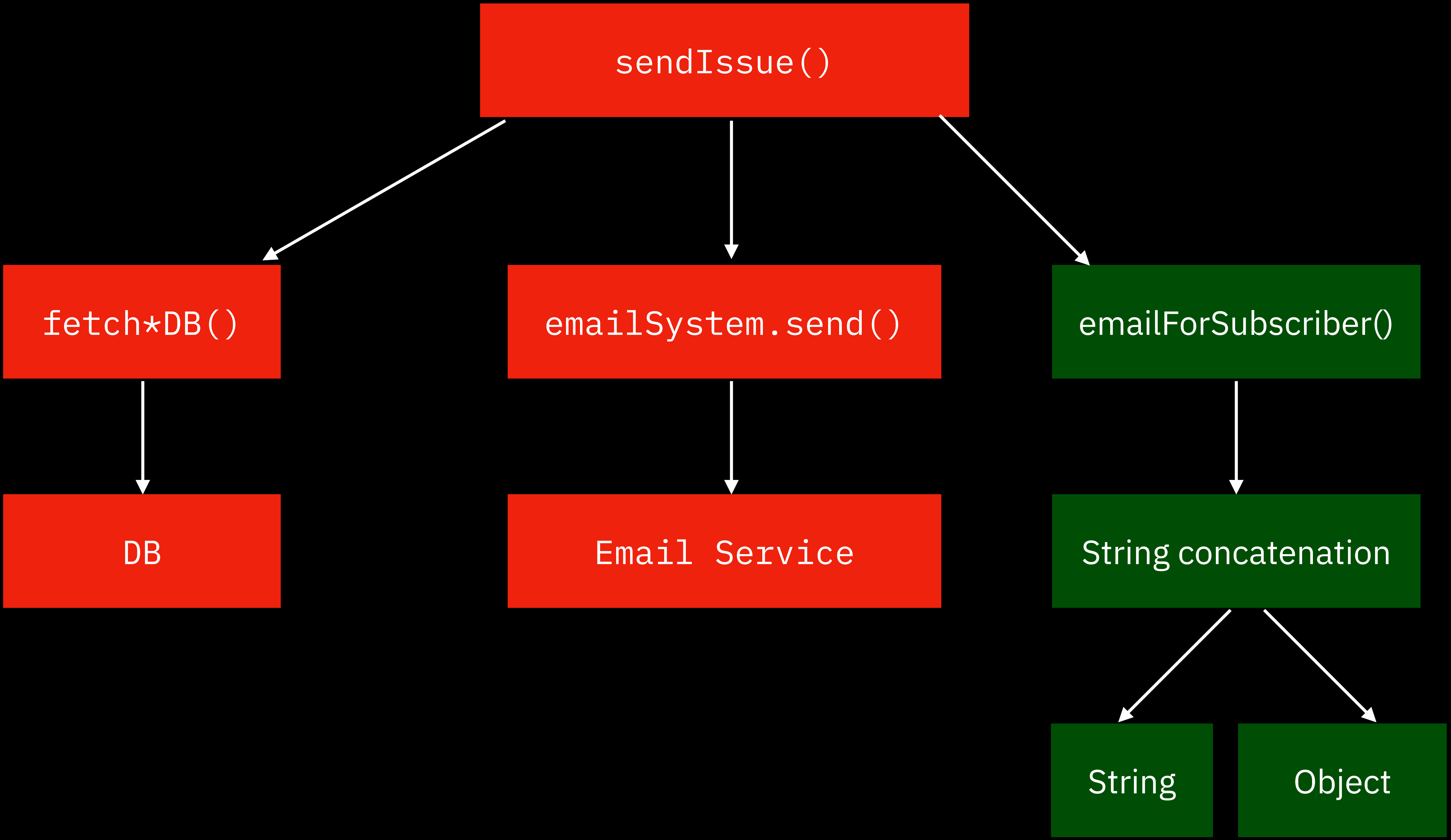
```
function emailForSubscriber(subscriber, coupons) {
  return {
    from: “newsletter@coupondog.co",
    to: subscriber.email,
    subject: “Your best weekly coupons inside”,
    body: “Here are the best coupons: “ +
            coupons.join(“, “)
  };
}


function sendIssue() {
  const coupons     = fetchCouponsFromDB();
  const subscribers = fetchSubscribersFromDB();
  const emails      = subscribers.map(
    (s) => emailForSubscriber(s, coupons)
  );
  emails.forEach((e) => emailSystem.send(e));

}
```

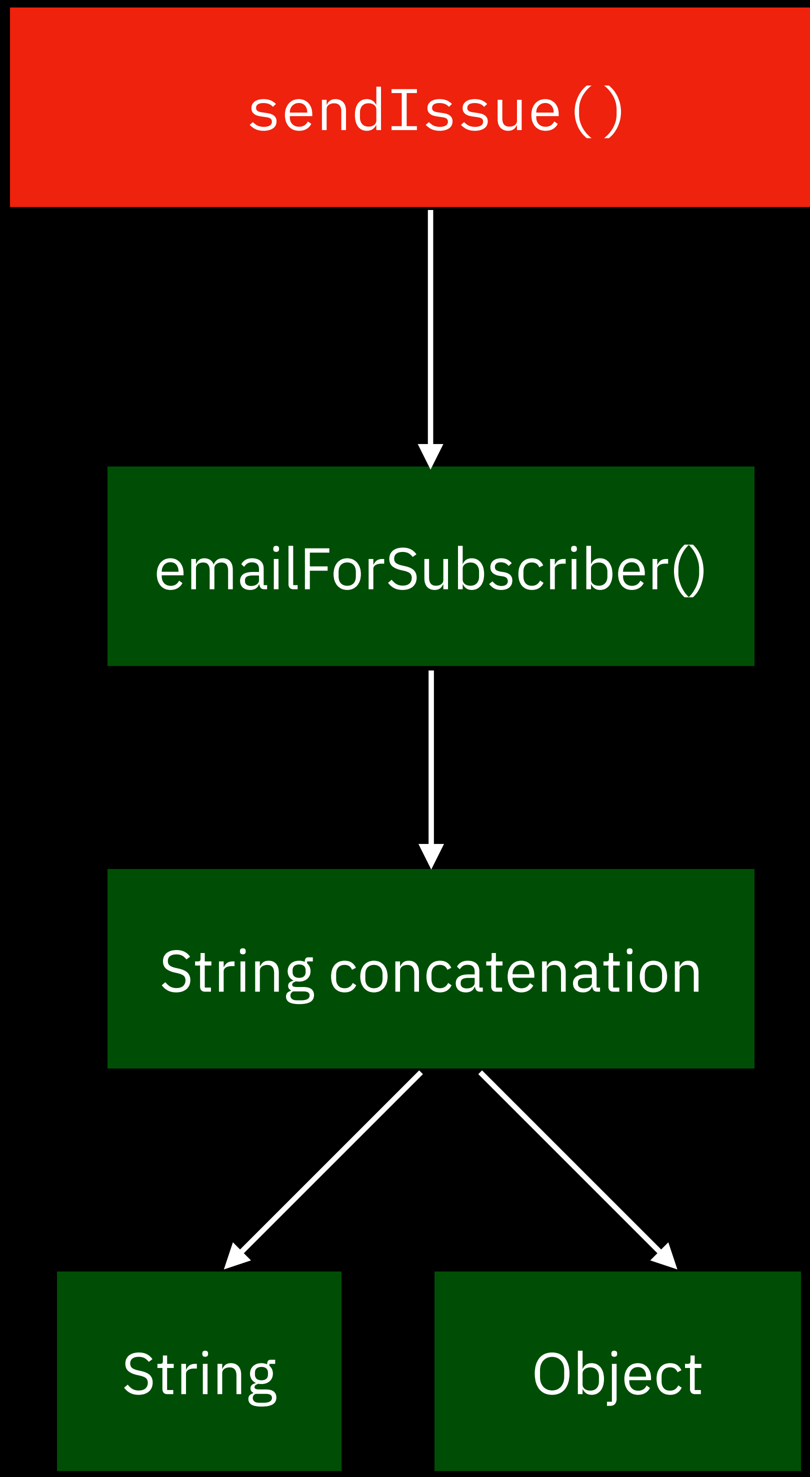
```
function emailForSubscriber(subscriber, coupons) {
  return {
    from: “newsletter@coupondog.co",
    to: subscriber.email,
    subject: “Your best weekly coupons inside”,
    body: “Here are the best coupons: “ +
            coupons.join(“, “)
  };
}
function sendIssue() {
  const coupons = fetchCouponsFromDB();
  let page = 0;
  let subscribers = fetchSubscribersFromDB(page);
  while(subscribers.length > 0) {
    const emails = subscribers.map(
      (s) => emailForSubscriber(s, coupons)
    );
    emails.forEach((e) => emailSystem.send(e));
    page += 1;
    subscribers = fetchSubscribersFromDB(page);
  }
}
```

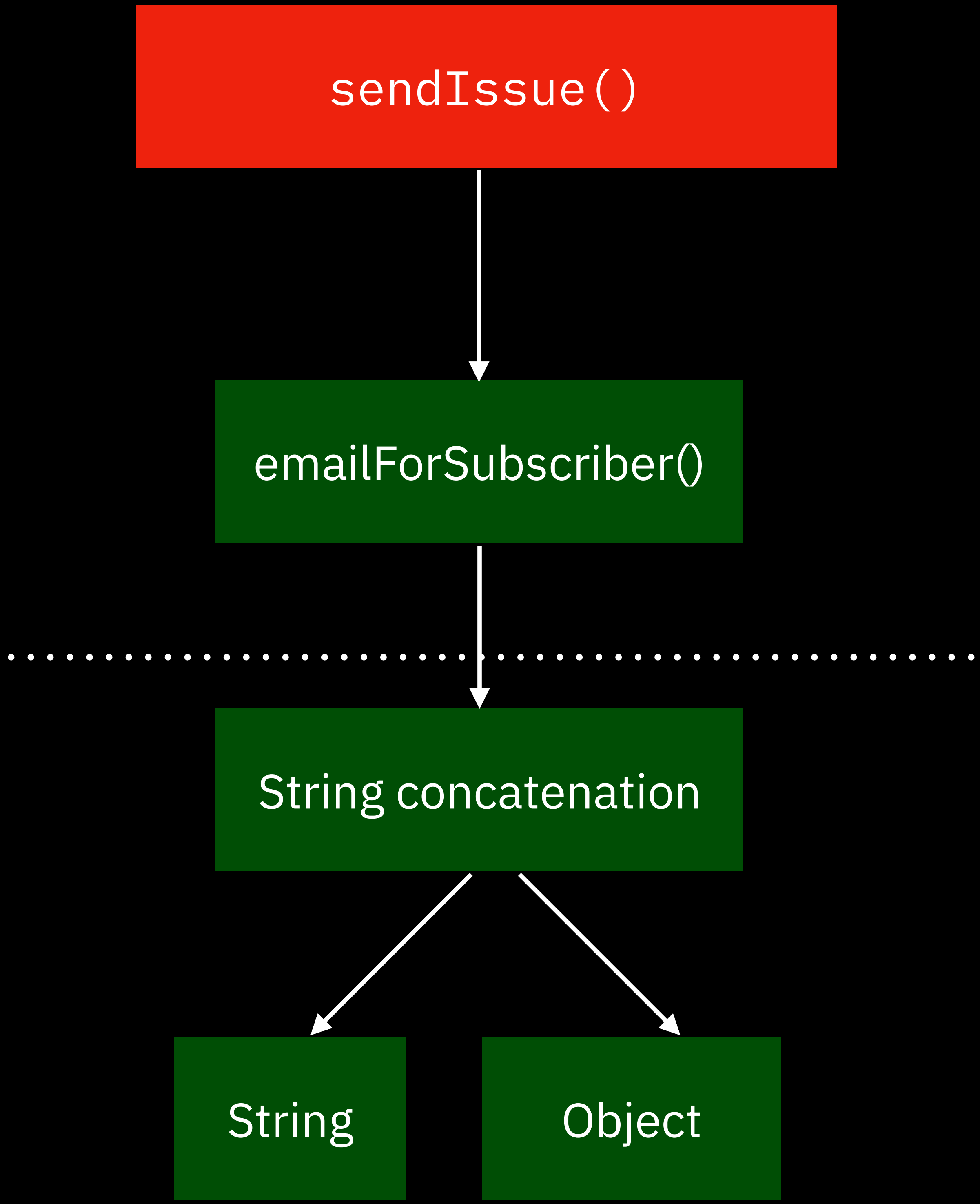
✓

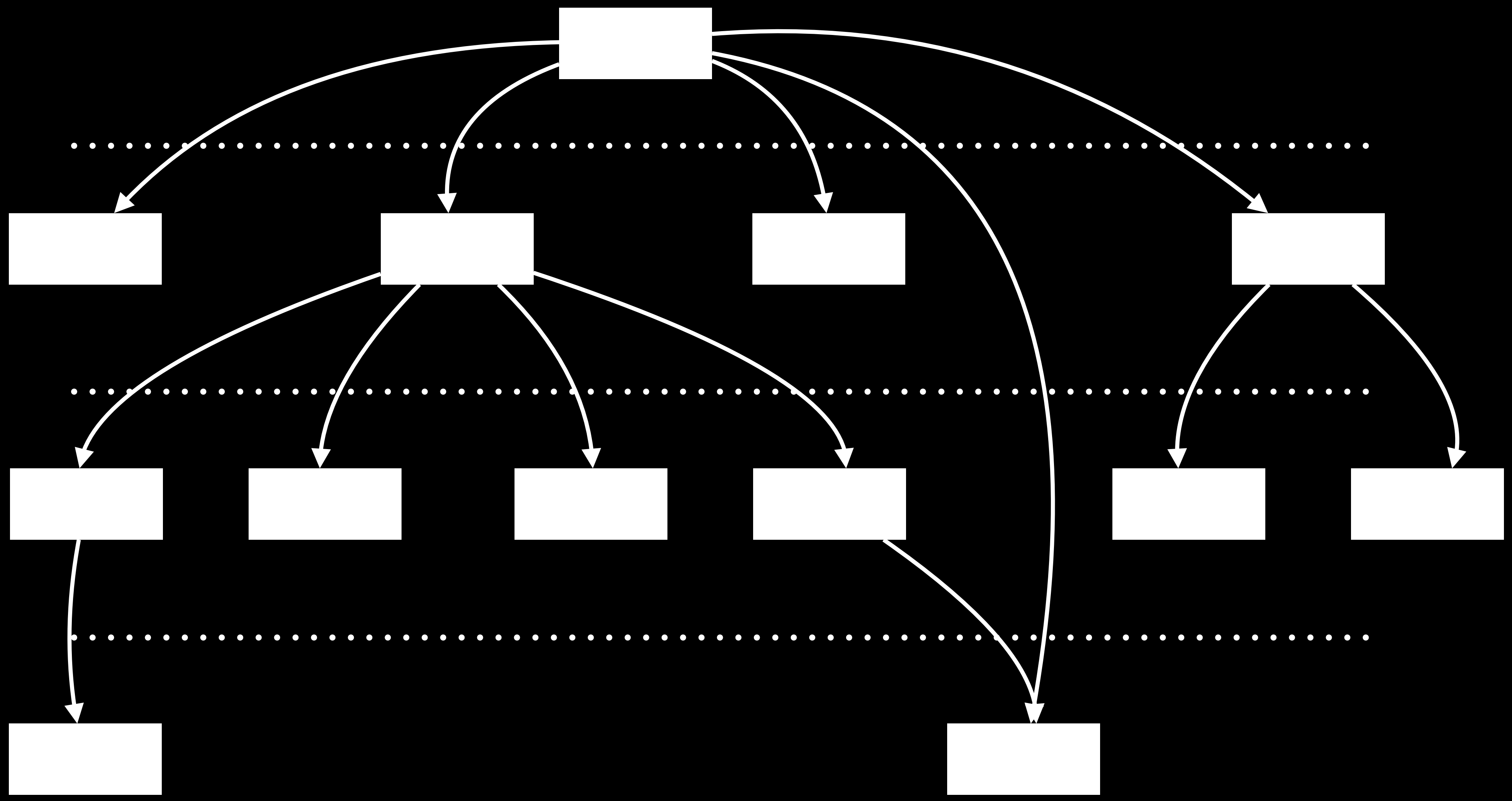
Pure functions ➕ Stratified design → Onion architecture


sendIssue()
fetch*DB()
emailSystem.send()
emailForSubscriber()

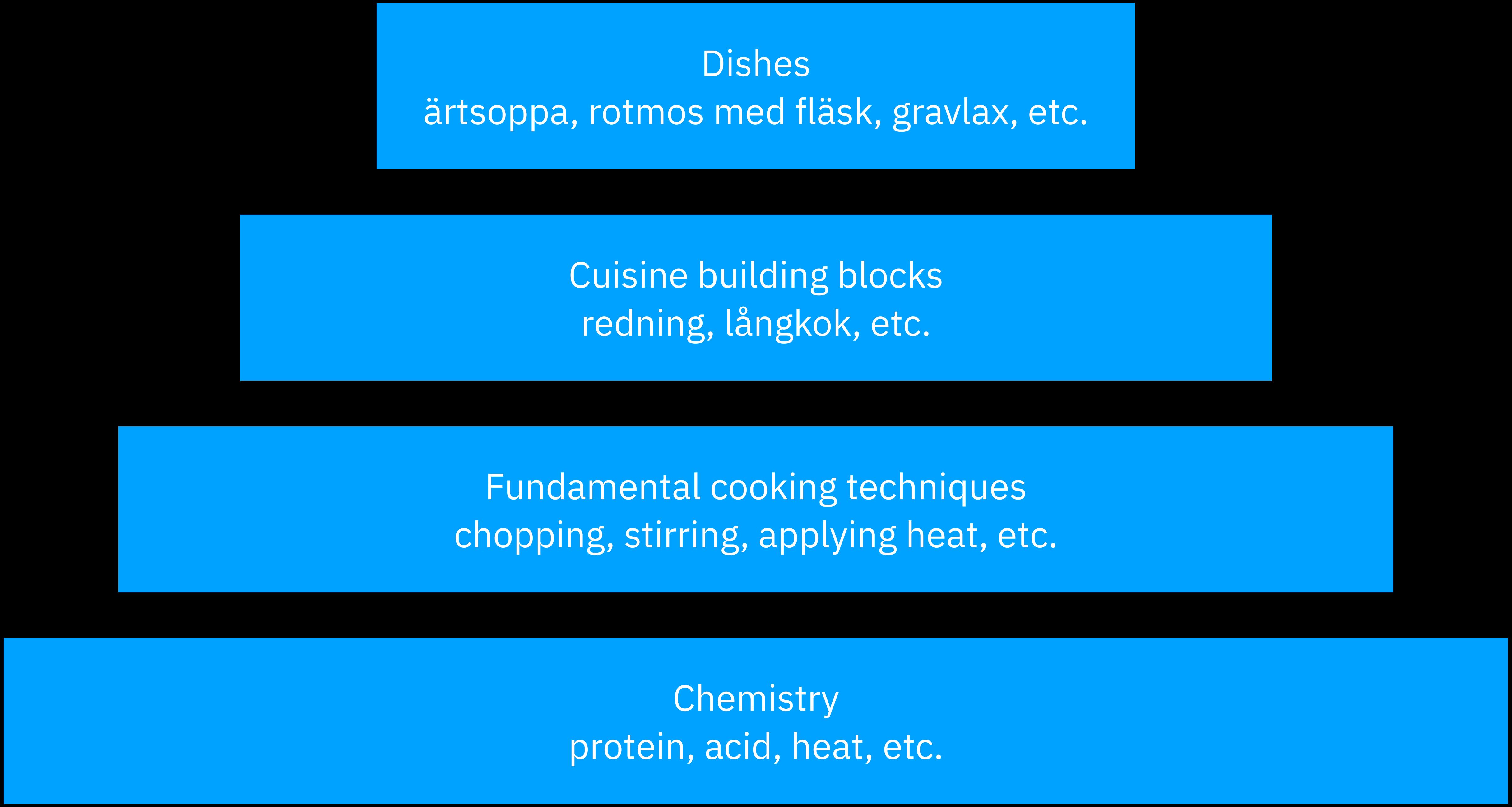

sendIssue()
fetch*DB()
emailSystem.send()
emailForSubscriber()
DB
Email Service
String concatenation
String
Object


sendIssue()
emailForSubscriber()
String concatenation
String
Object

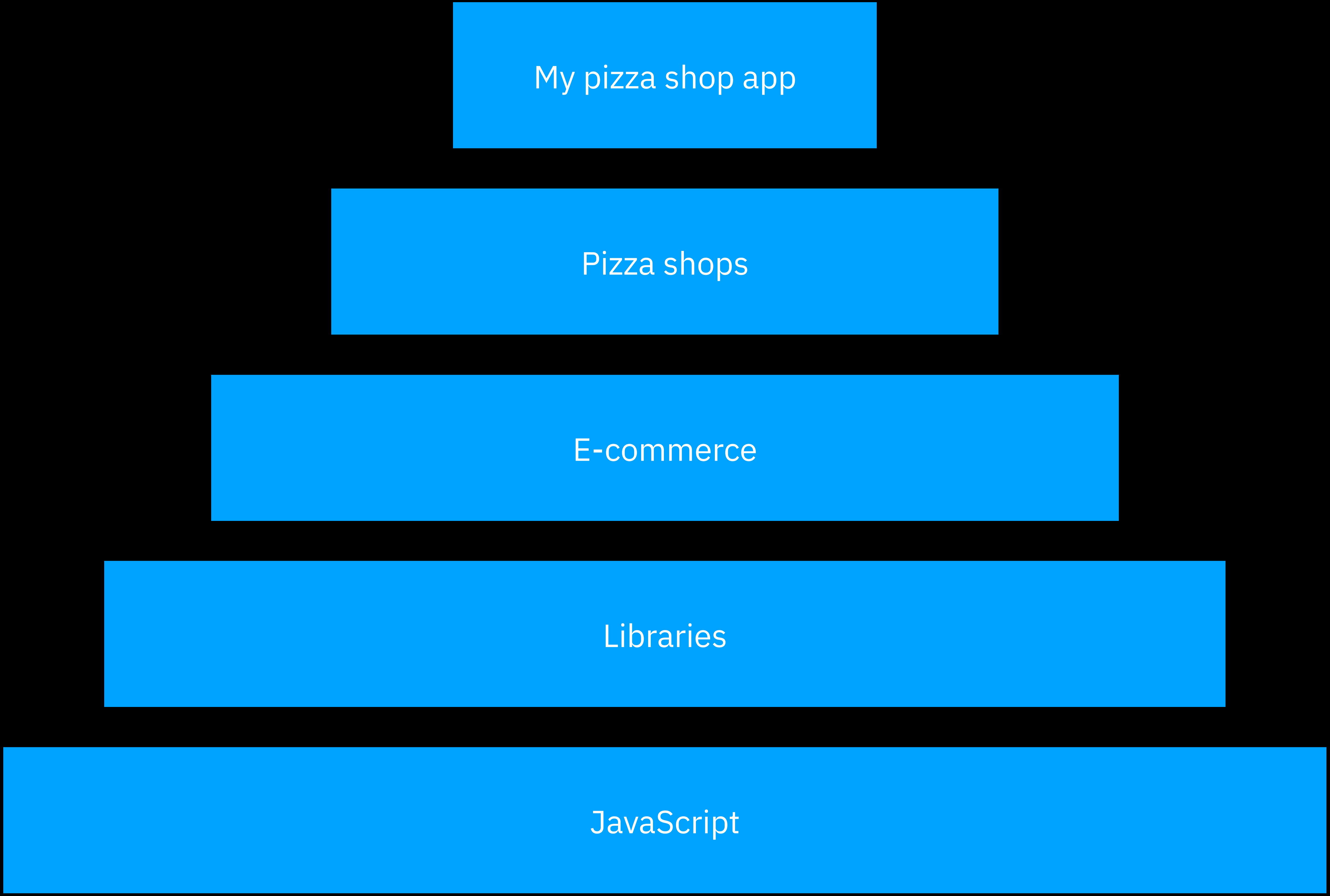

sendIssue()
emailForSubscriber()
String concatenation
String
Object

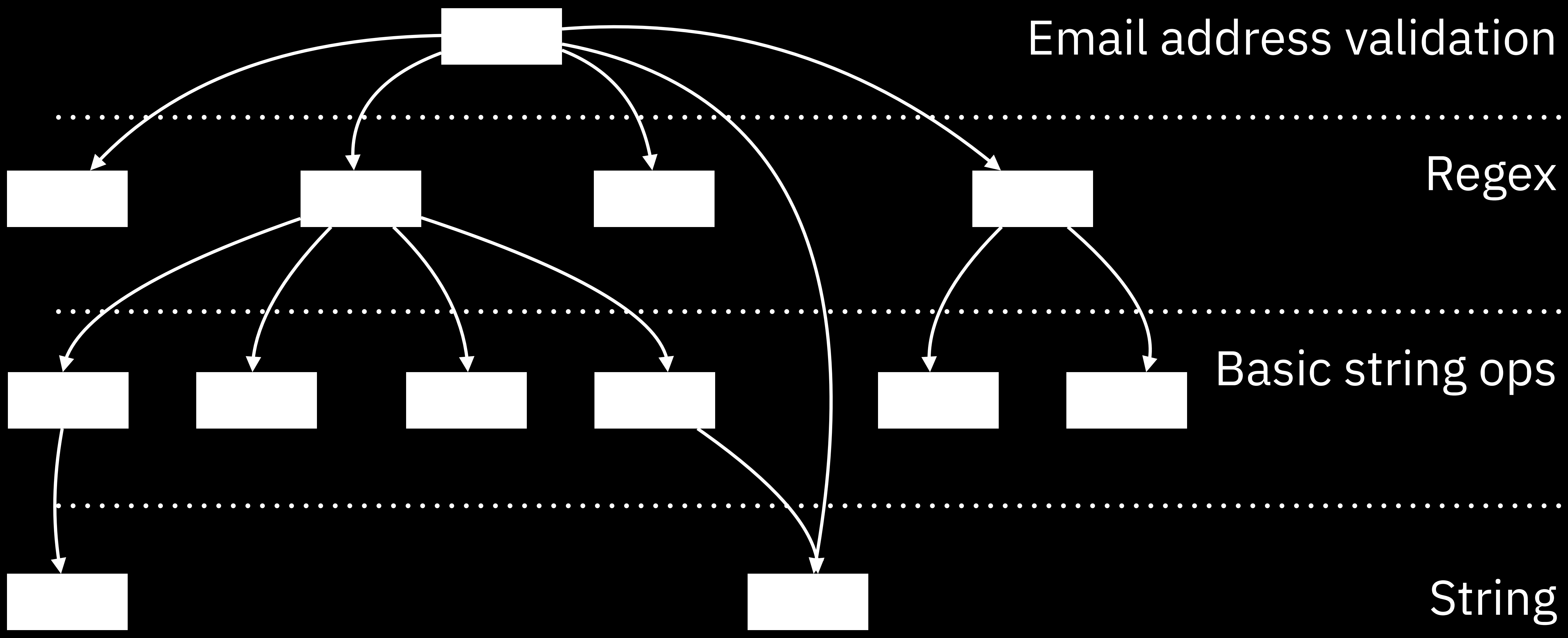


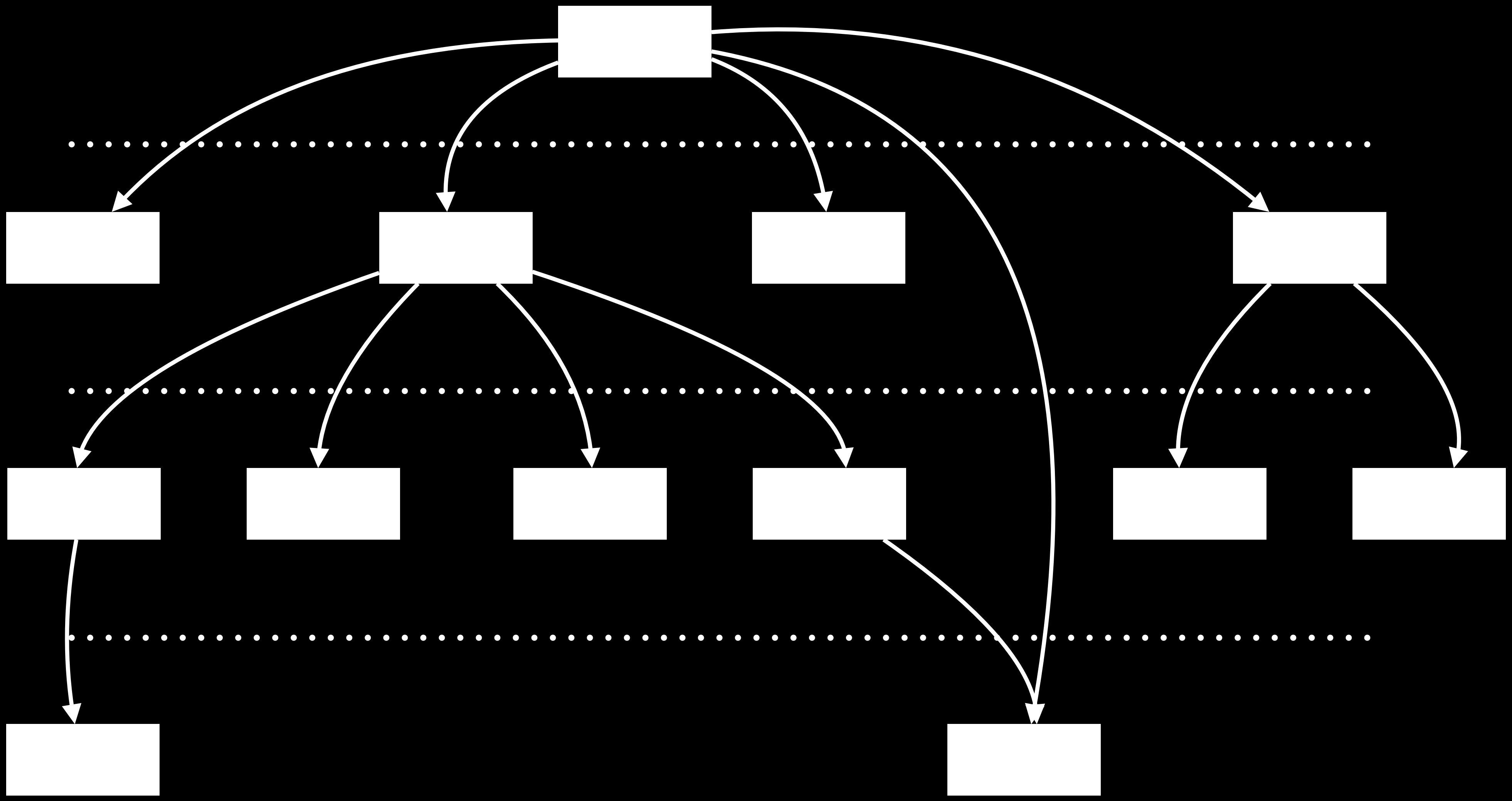
Stratified design

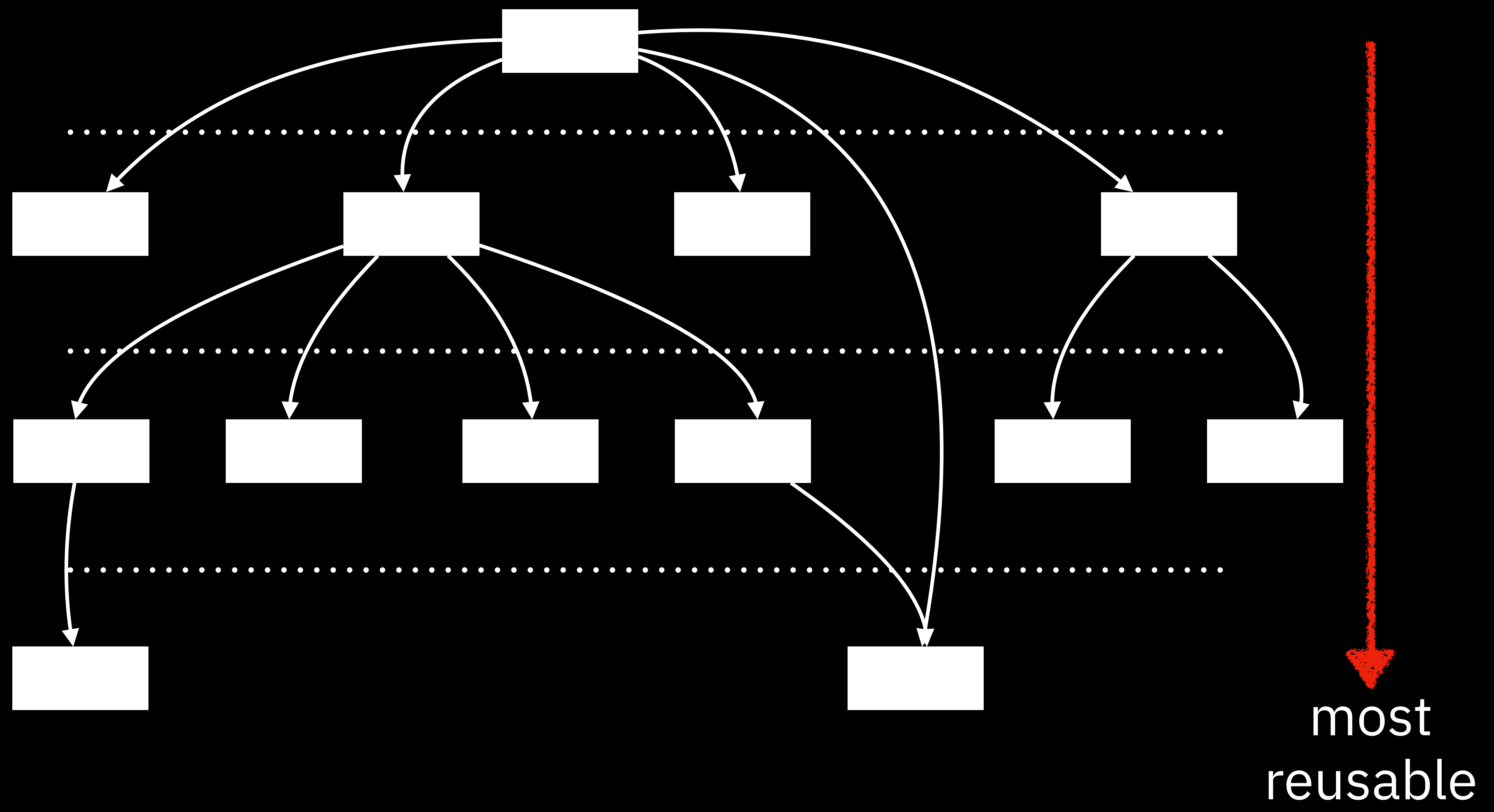

Dishes
ärtsoppa, rotmos med fläsk, gravlax, etc.
Cuisine building blocks
redning, långkok, etc.
Fundamental cooking techniques
chopping, stirring, applying heat, etc.
Chemistry
protein, acid, heat, etc.

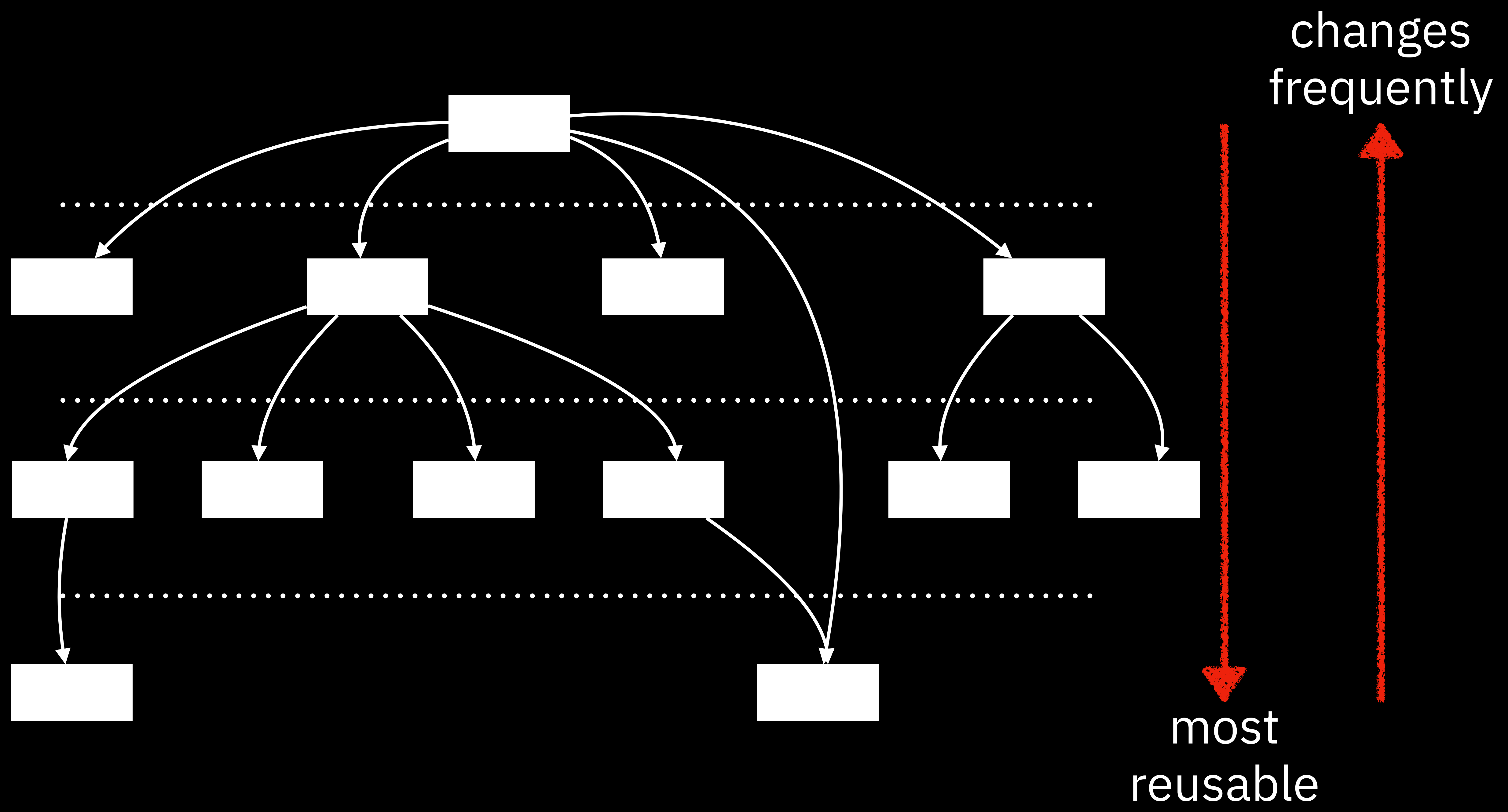
Stratified design

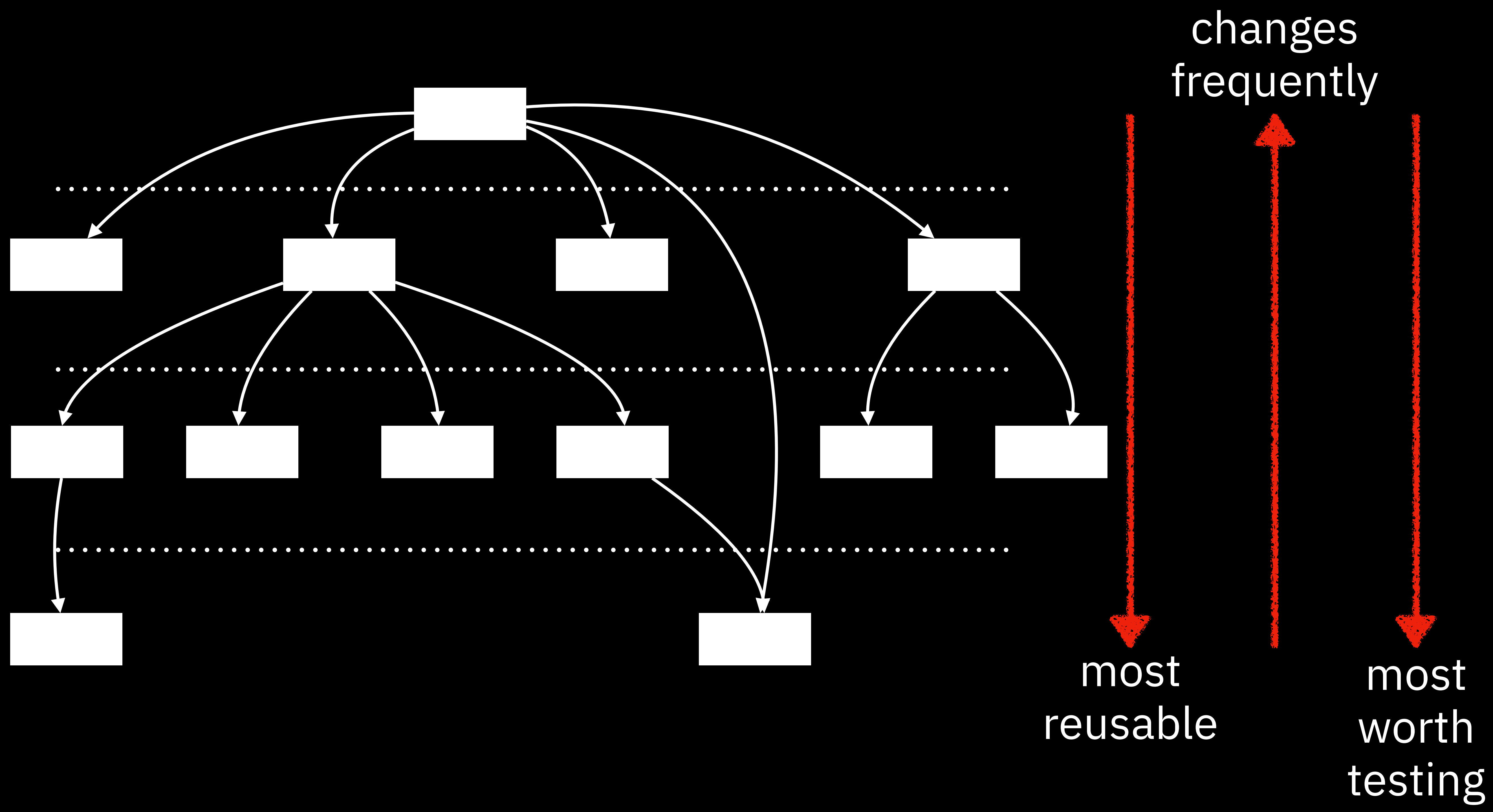


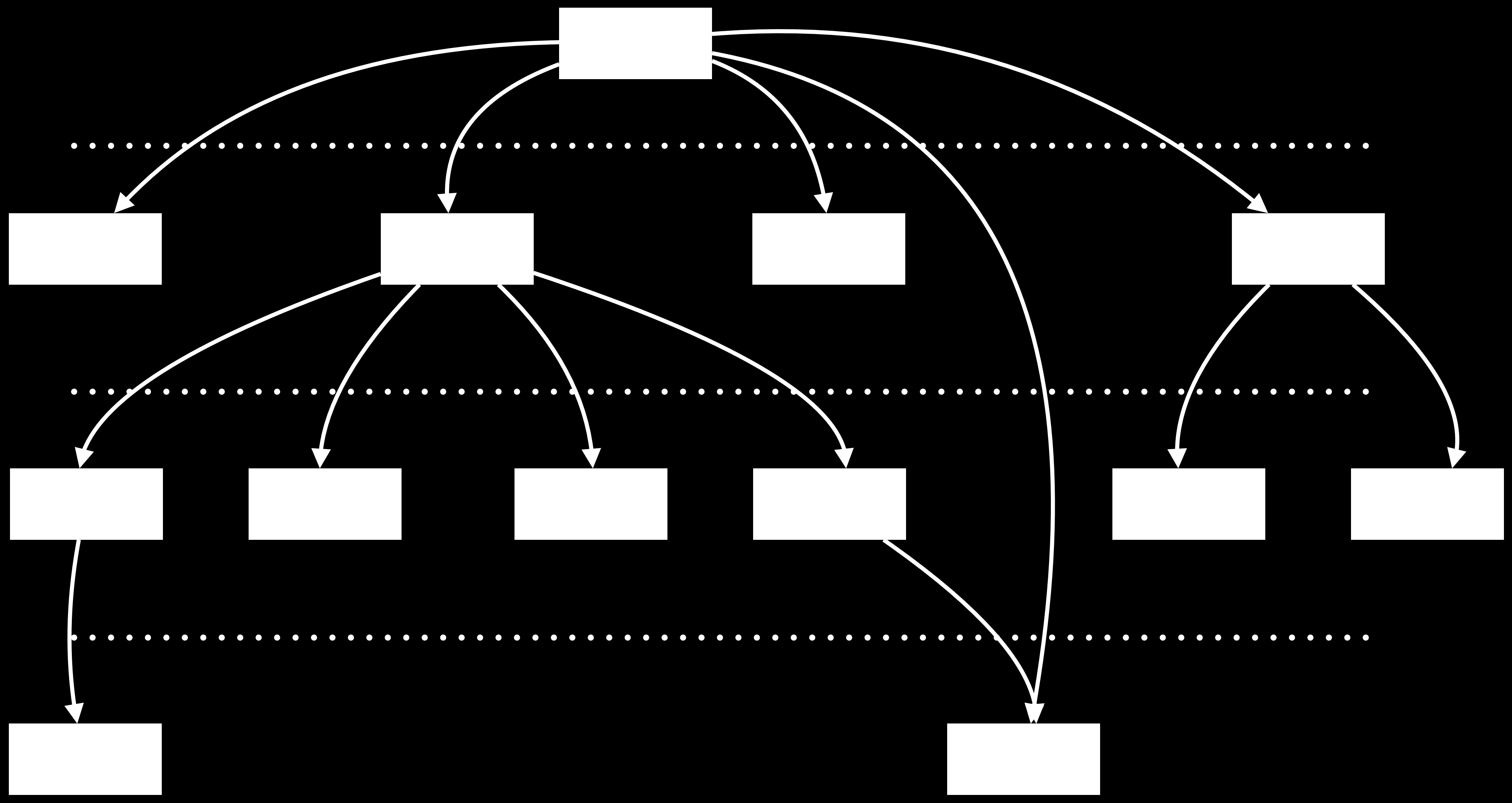

Email address validation
Regex
Basic string ops
String

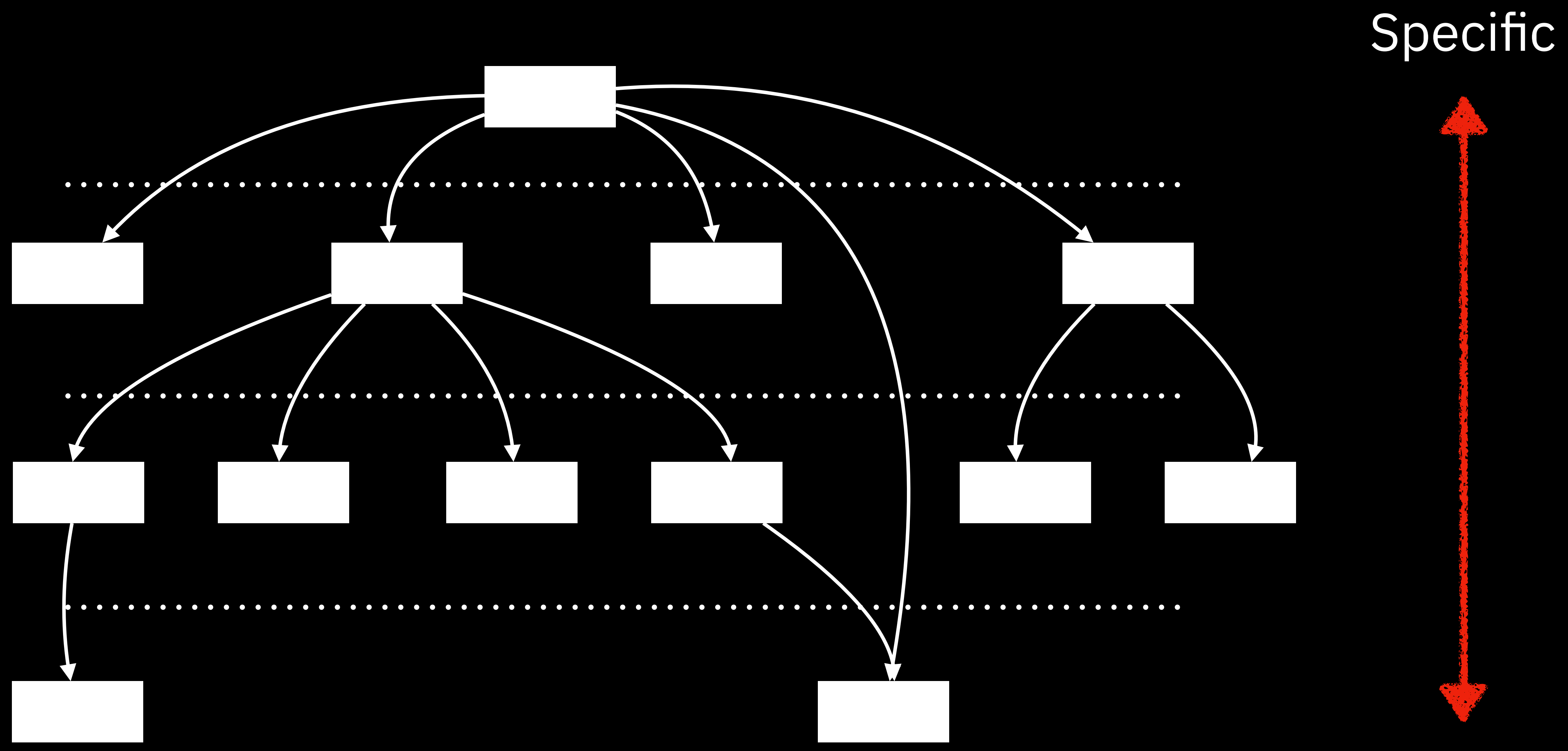



most
reusable


changes frequently
most reusable

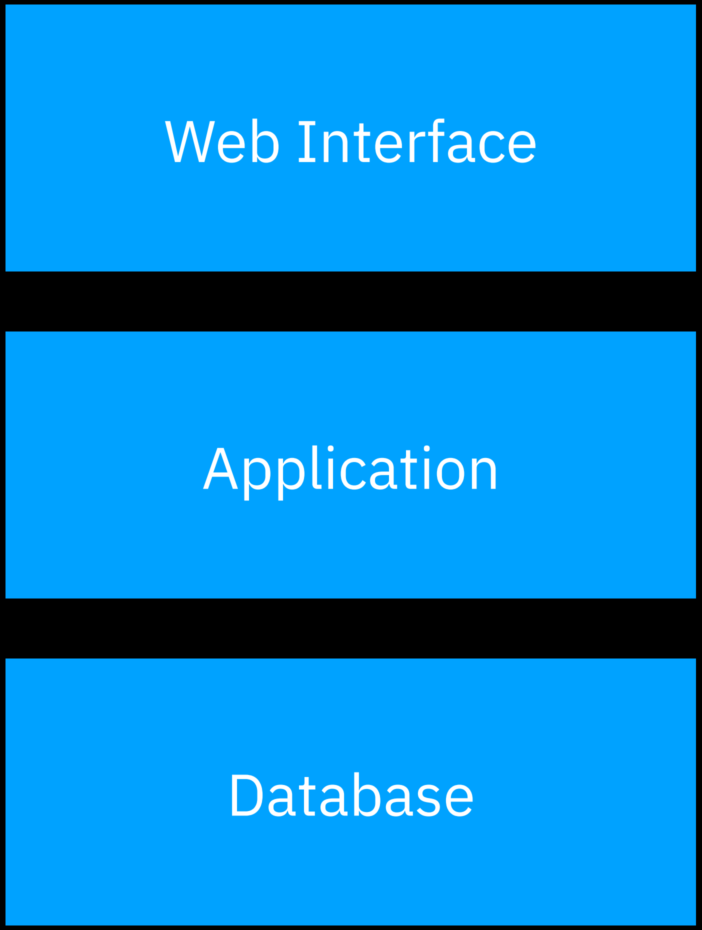

changes frequently
most reusable
most worth testing



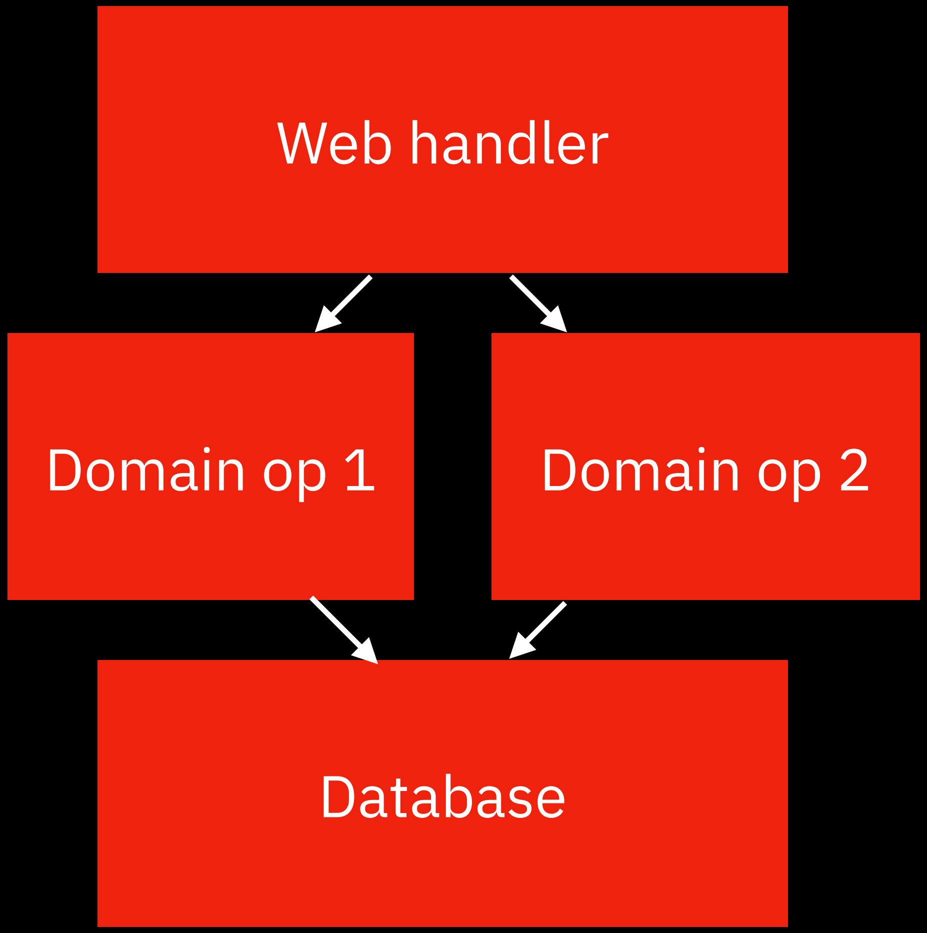

Specific
General

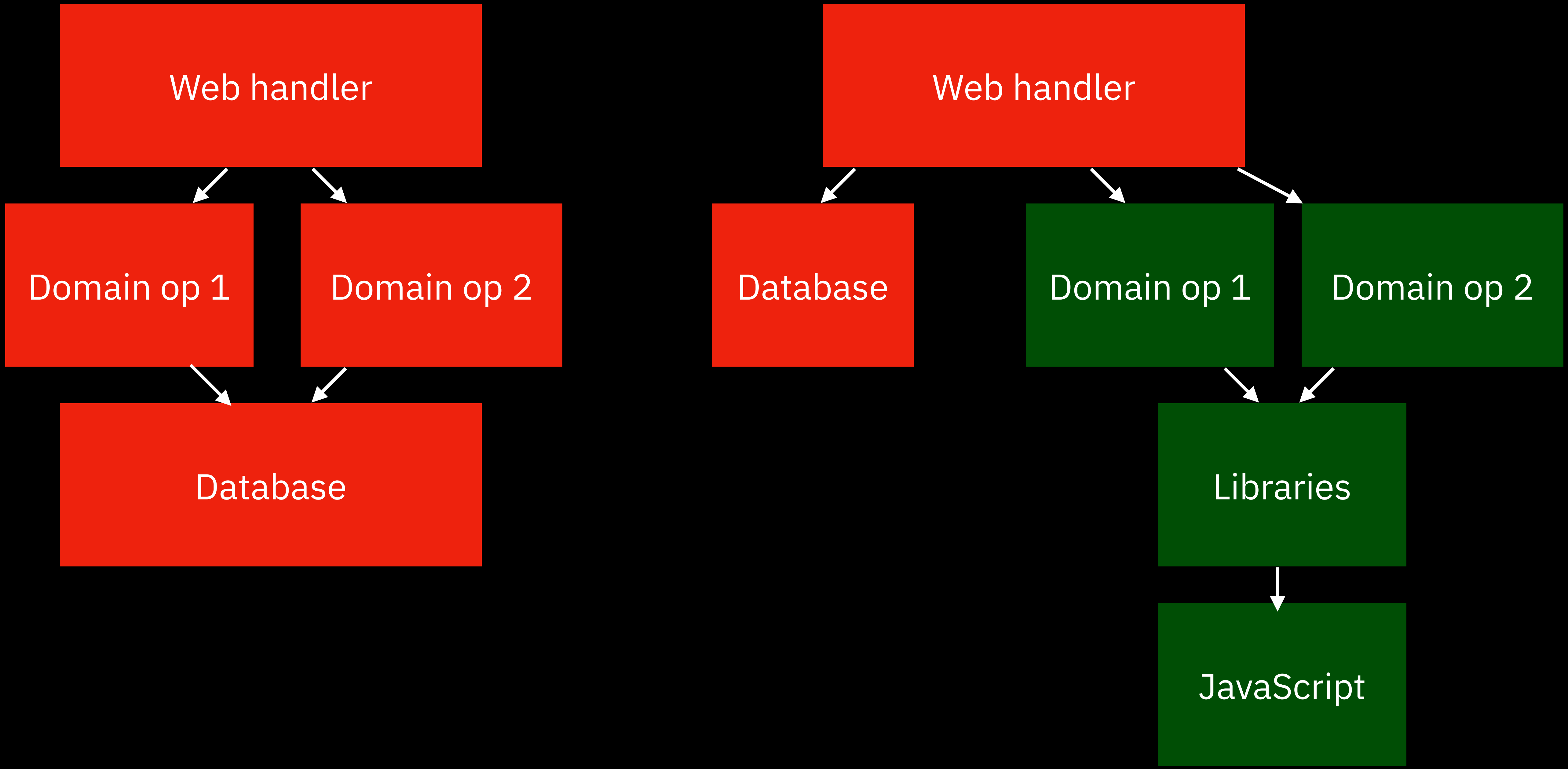
Pure functions ✓ + Stratified design ✓ → Onion architecture

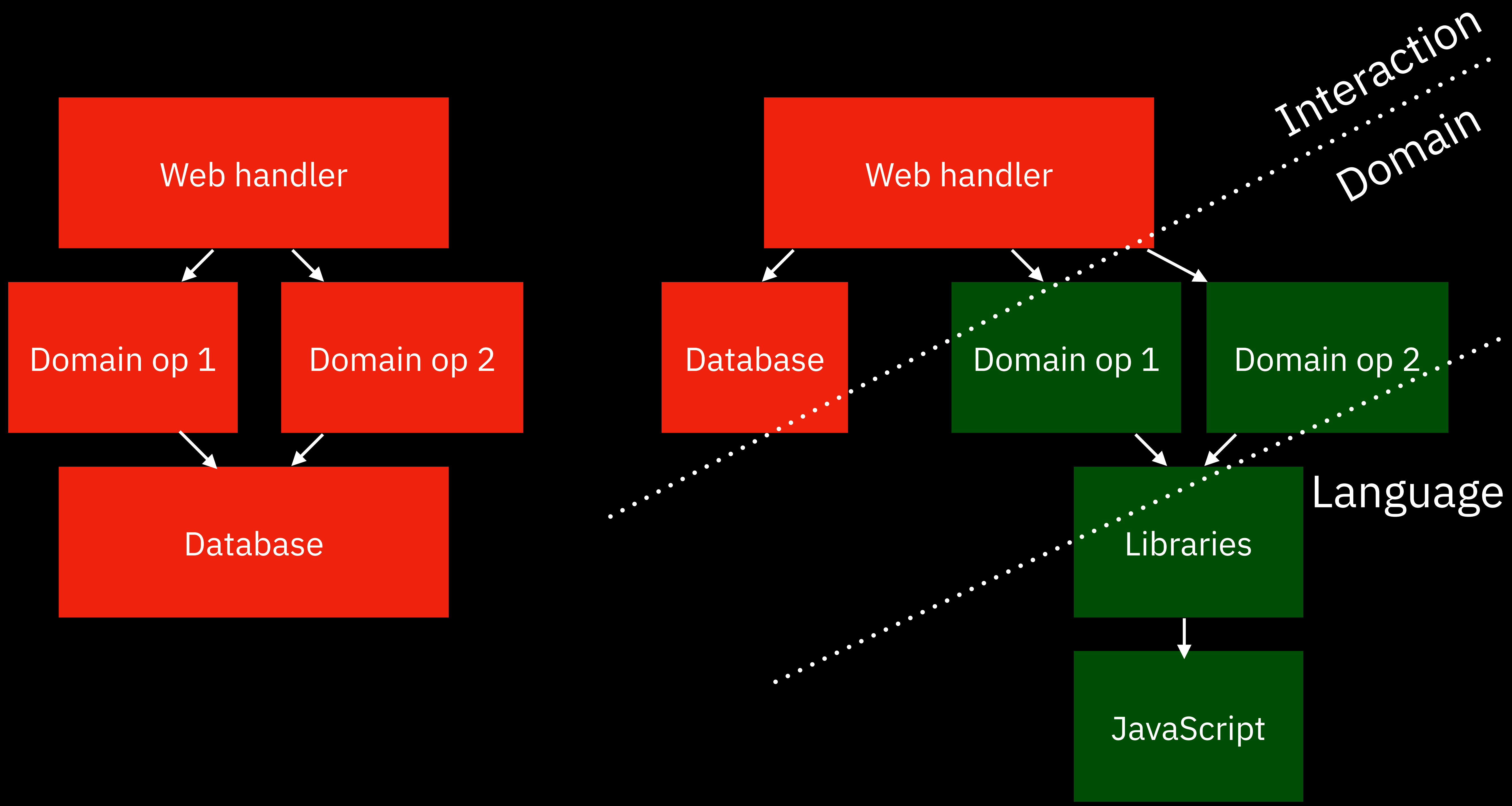
Traditional layered architecture



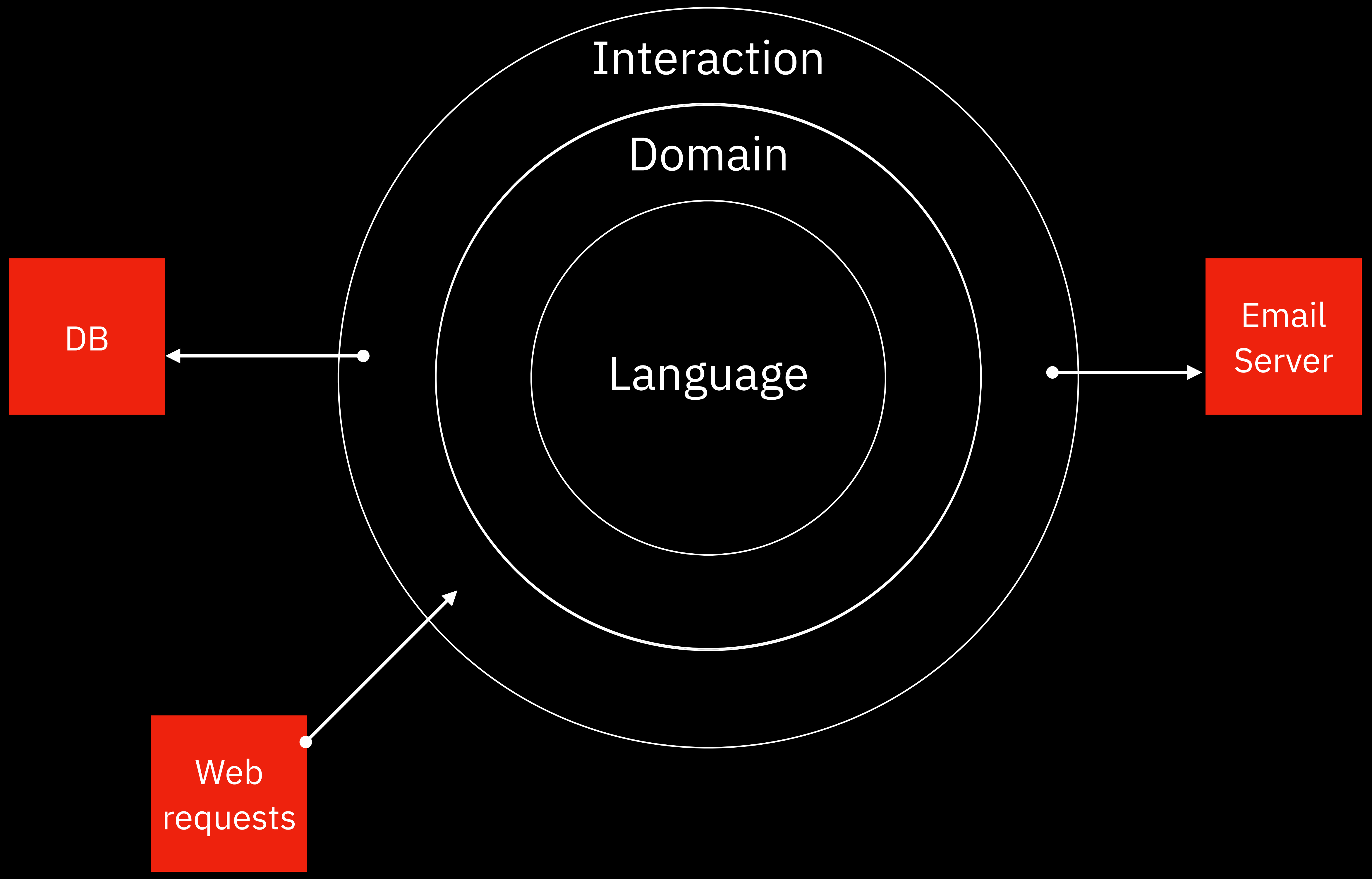
Traditional layered architecture




Web handler
Domain op 1
Domain op 2
Database
Web handler
Database
Domain op 1
Domain op 2
Libraries
JavaScript


Web handler
Domain op 1
Domain op 2
Database
Web handler
Database
Domain op 1
Domain op 2
Libraries
JavaScript
Interaction
Domain
Language

Onion architecture



Onion architecture

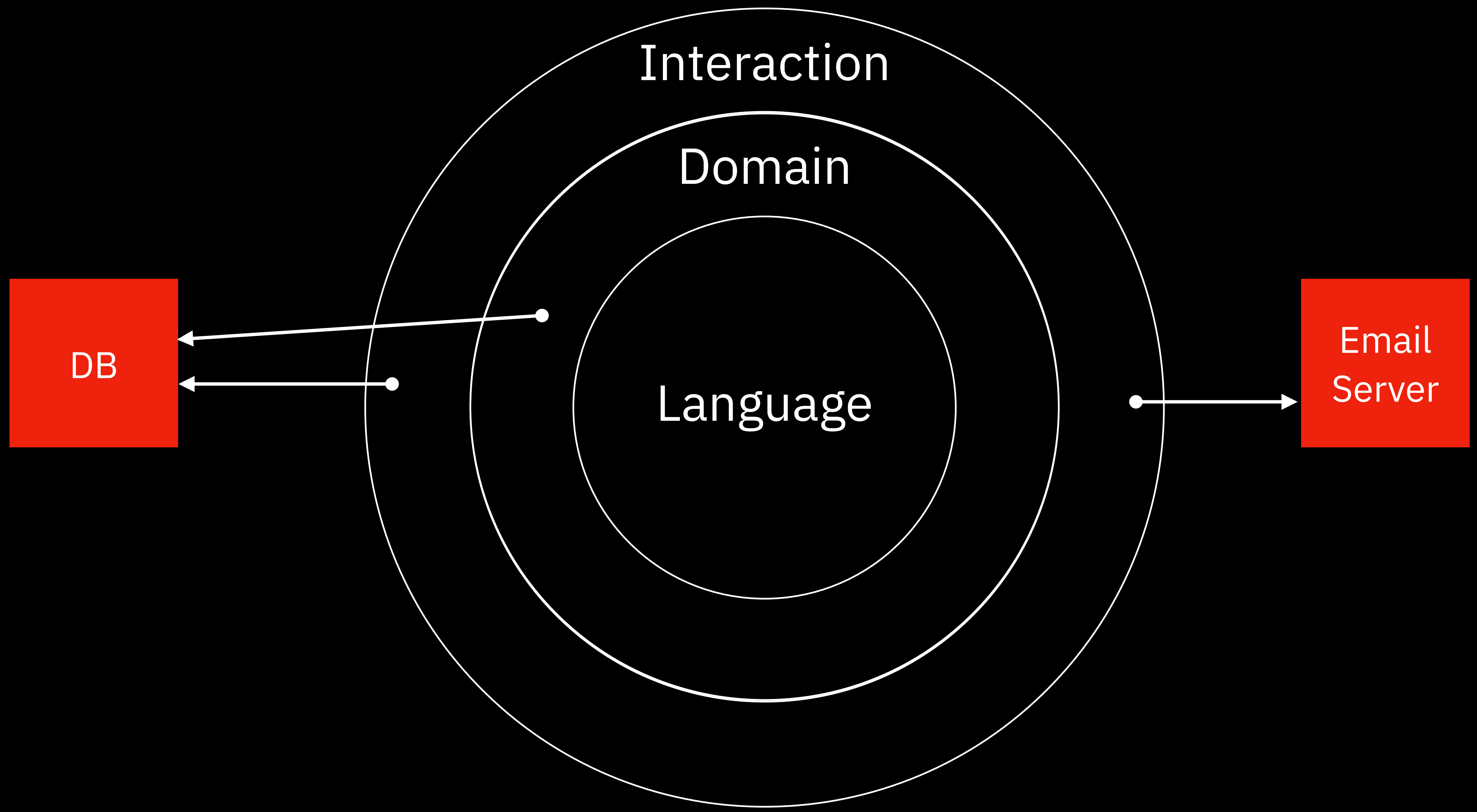
also known as

- Ports and adapters
- Hexagonal architecture
- Functional core, imperative shell

Common questions

What if your domain rule needs to ask the DB?

Onion architecture



Is it really a domain rule?

```
var image = newImageDB.getImage(‘123’);

if(image === undefined)
  image = oldImageDB.getImage(‘123’);
```

Domain terms:

product, image, price, discount

```
var image = newImageDB.getImage(‘123’);

if(image === undefined)
  image = oldImageDB.getImage(‘123’);
```

Non-domain terms:

database, old, new

It belongs in the interaction layer.

```
function generateReport(products) {
  return reduce(products, “”, (report, product) =>
    report + product.name + “ “ + product.price + “\n”);
}

const productsLastYear = db.fetchProducts(‘last year’);
const reportLastYear   = generateReport(productsLastYear);
```

```
function generateReport(products) {
  return reduce(products, “”, (report, product) =>
    report + product.name + “ “ + product.price + “\n”);
}

const productsLastYear = db.fetchProducts(‘last year’);
const reportLastYear   = generateReport(productsLastYear);
```

```
{                                  {
  name: “shoes”,                     name: “watch”,
  price: 3.99,                       price: 223.43,
  discountID: ‘2311’                 discountID: null
}                                  }
```

```
function generateReport(products) {
  return reduce(products, “”, (report, product) =>
    report + product.name + “ “ + product.price +
     “ discount: “ + (product.discount || 0) + “%\n”);
}

const productsLastYear = db.fetchProducts(‘last year’);

const productsWithDiscounts = map(productsLastYear, (product) => {
  if(product.discountID)
    product.discount = db.fetchDiscount(product.discountID);
  return product;
});

const reportLastYear   = generateReport(productsWithDiscounts);
```

Don’t overcomplicate

gifbin.com

gifbin.com

Actions

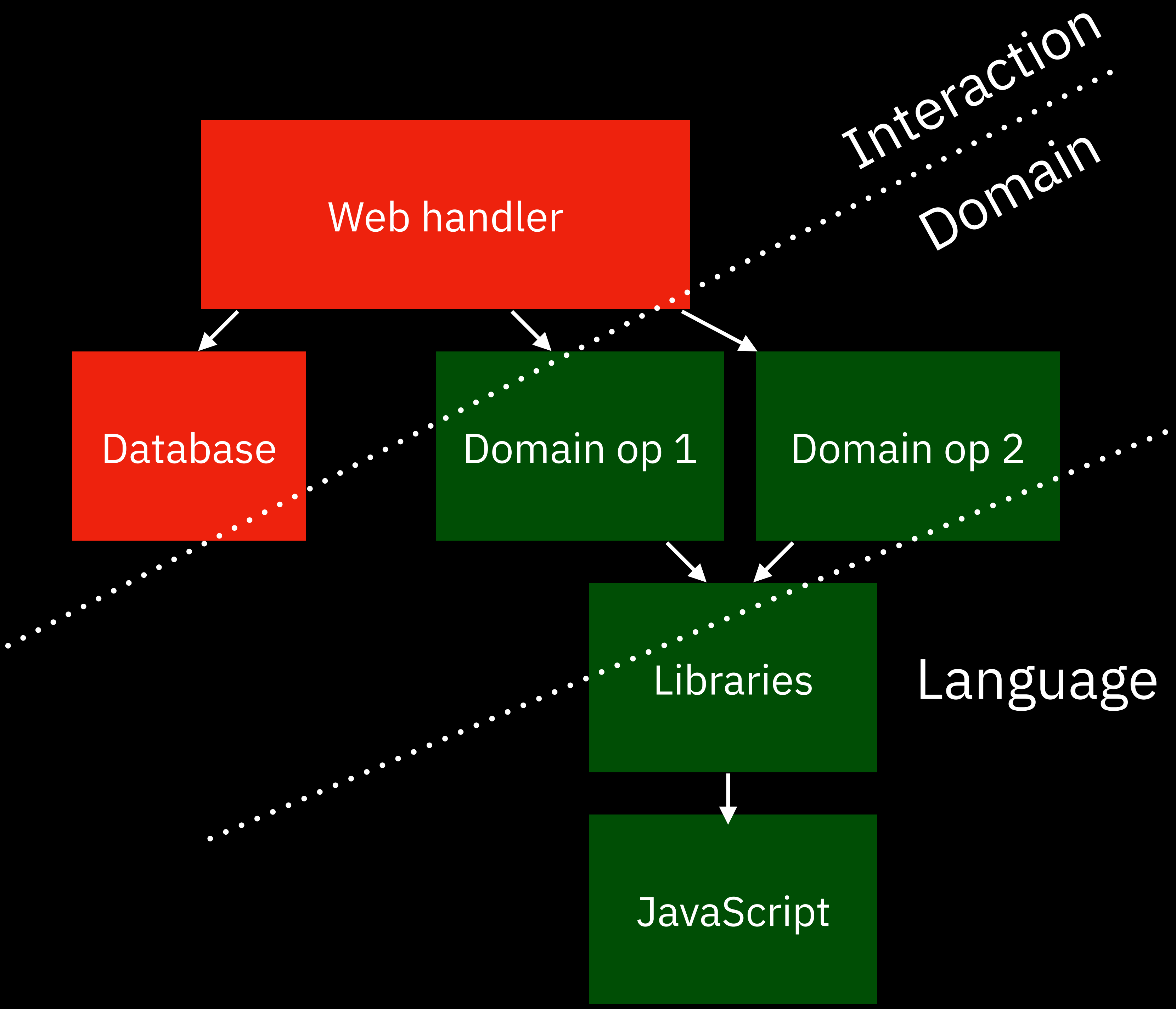
```
sendEmail(to, from, subject, body)

saveUserDB(user)

getCurrentTime()
```

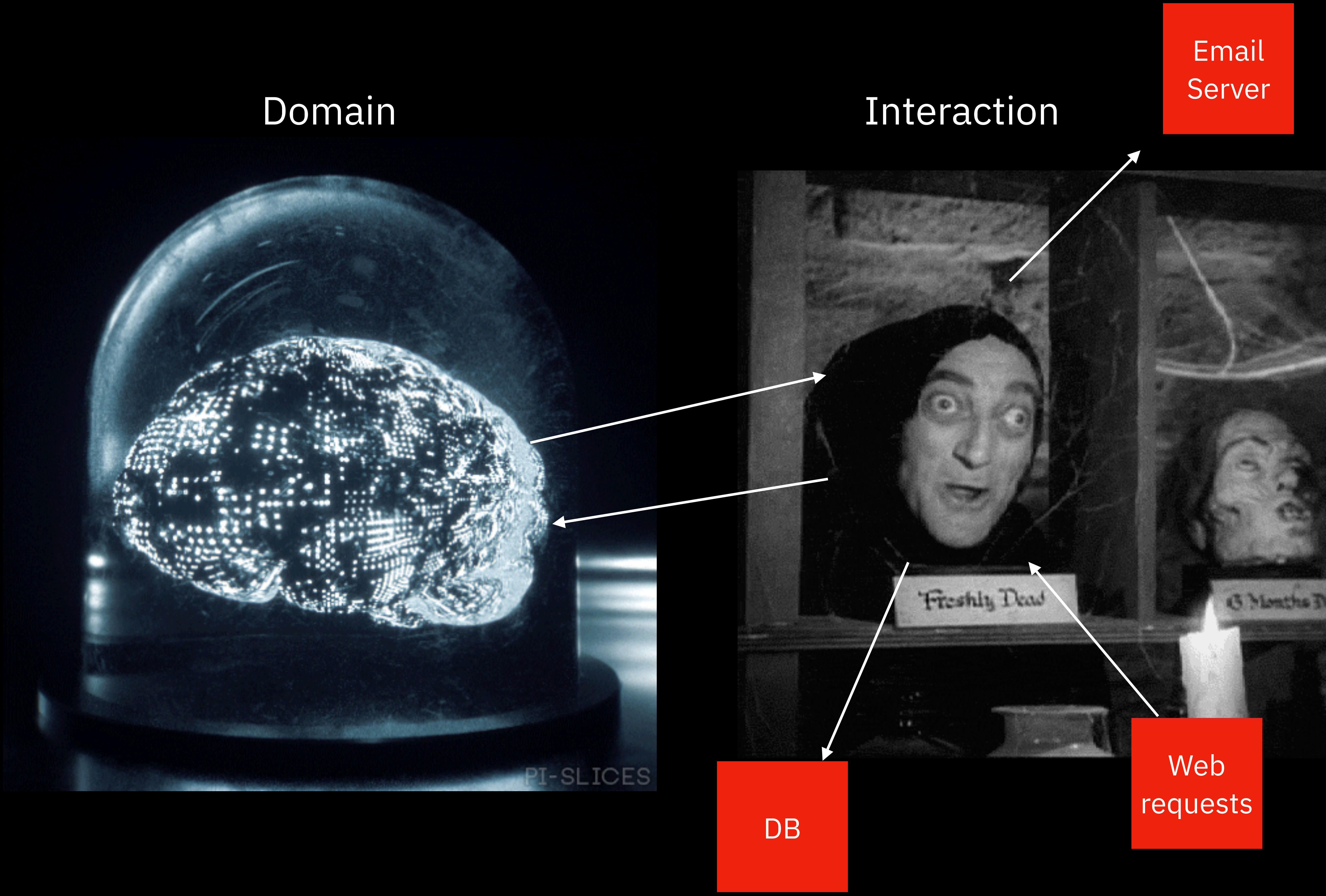
Calculations

```
sum(numbers)

stringLength(str)
```

Data

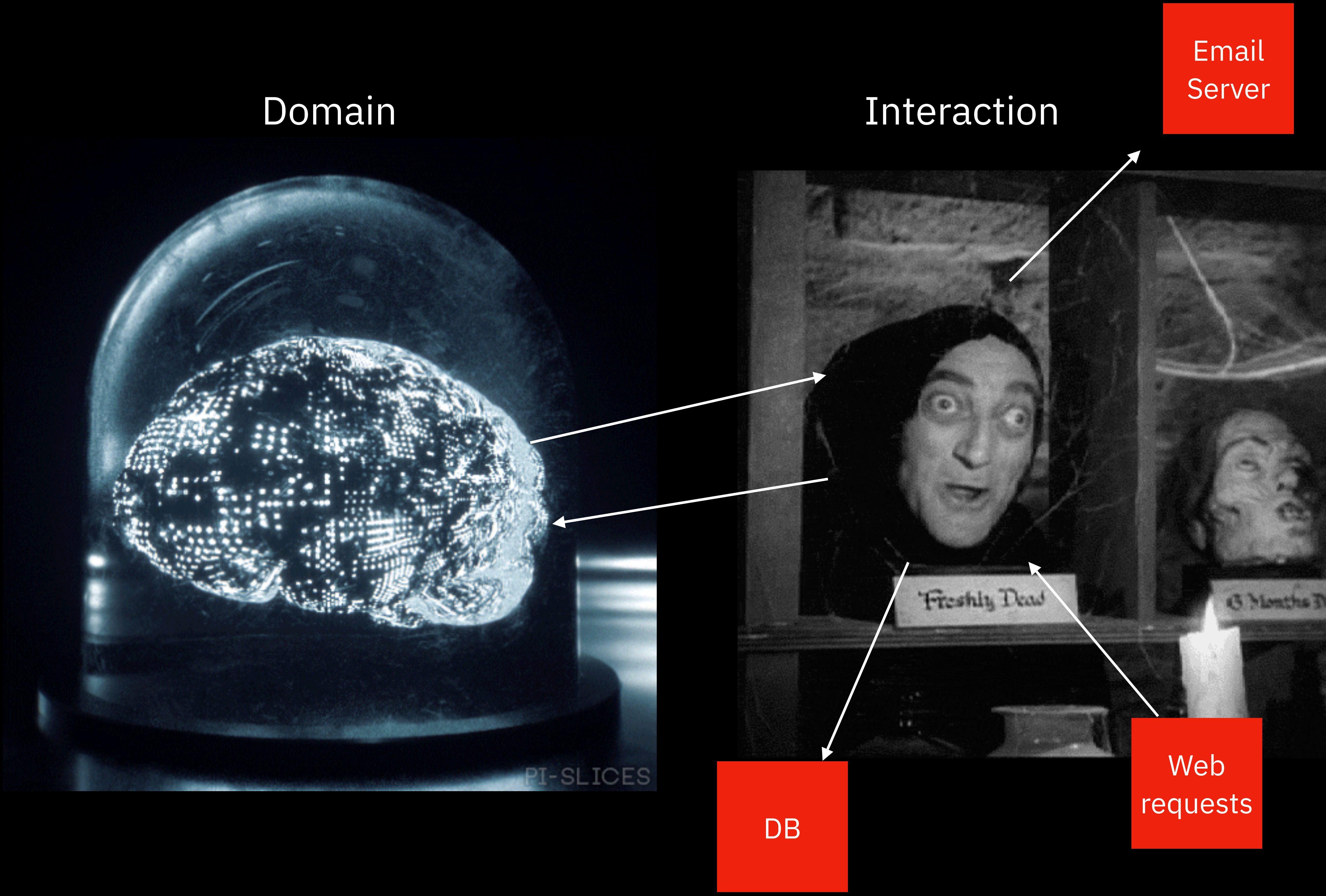
```
{“firstname”: “Eric”,
 “lastname”: “Normand”}

[1, 10, 2, 45, 3, 98]
```

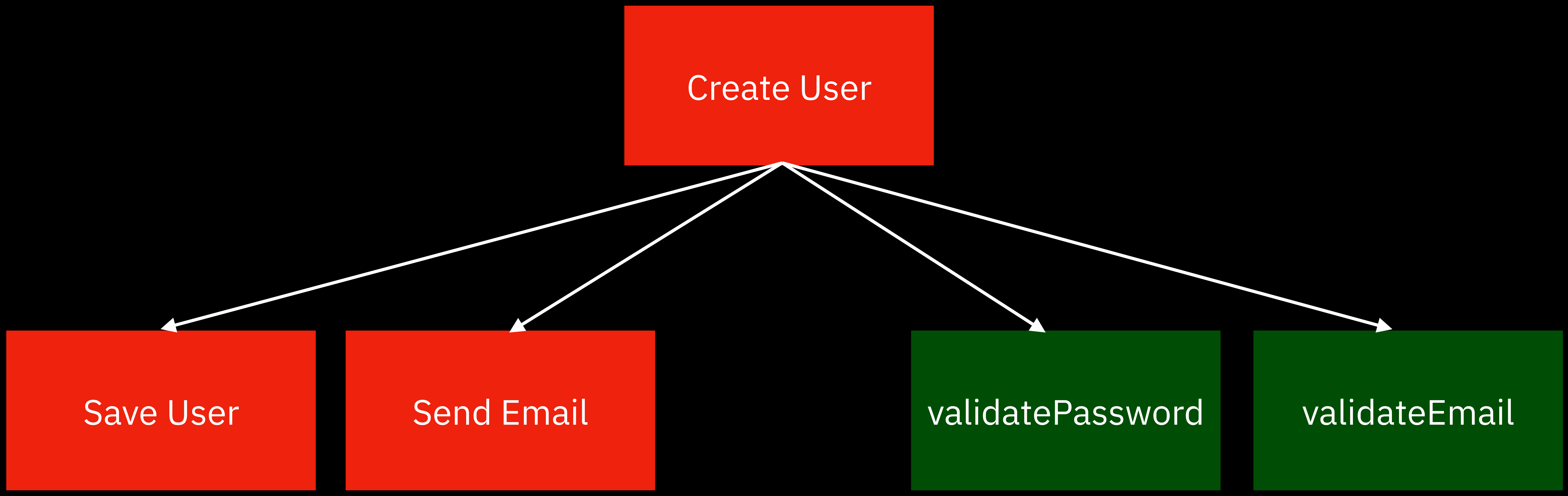

Interaction
Domain
Language
Web handler
Database
Domain op 1
Domain op 2
Libraries
JavaScript


Domain
Interaction
Email Server
PI-SLICES
Freshly Dead
DB
Web requests

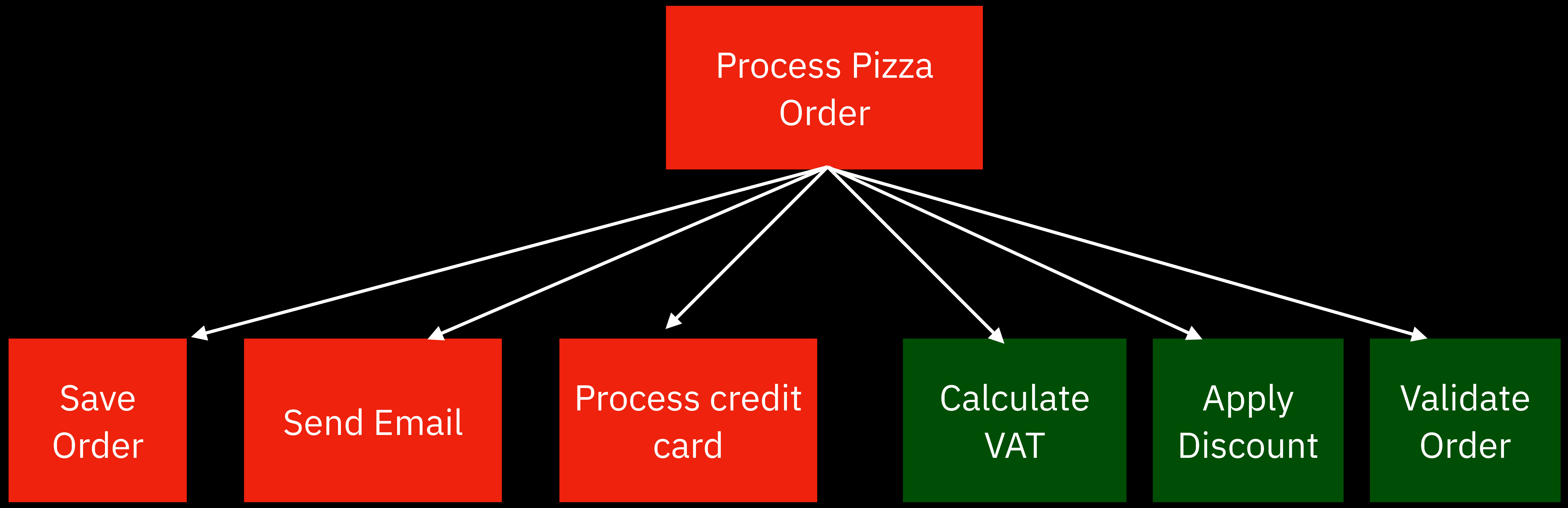
Domain

Interaction



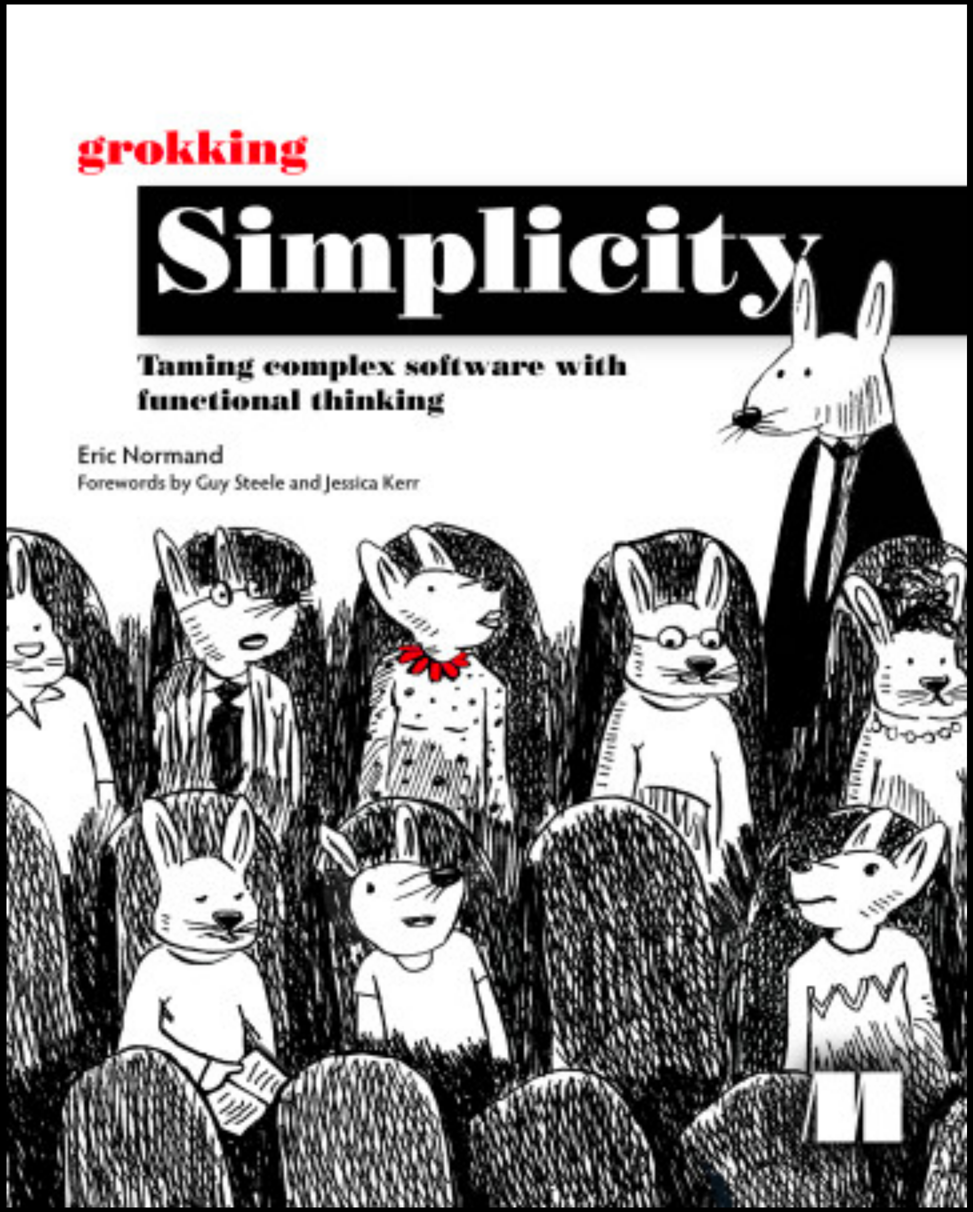


Onion Architecture



Onion Architecture

✓ Pure functions + ✓ Stratified design → ✓ Onion architecture


grokking
Simplicity
Taming complex software with functional thinking
Eric Normand
Forewords by Guy Steele and Jessica Kerr

ericnormand.me/gs

40% off: `grokdev23`

ericnormand.me