The Anatomy of a Distributed JavaScript Runtime

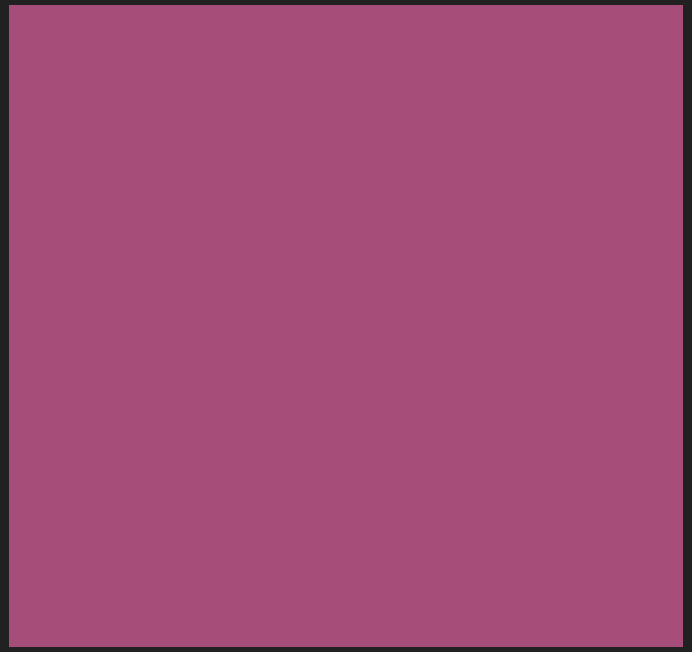
Masking Technology
hello@masking.tech
linkedin.com/company/maskingtechnology

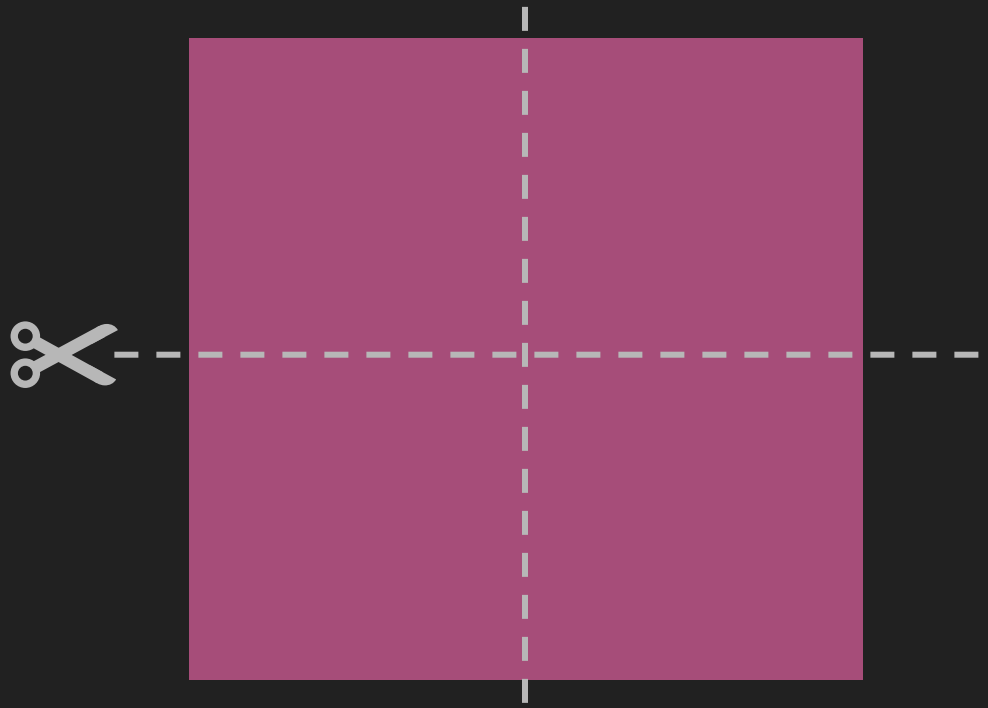
Masking Technology

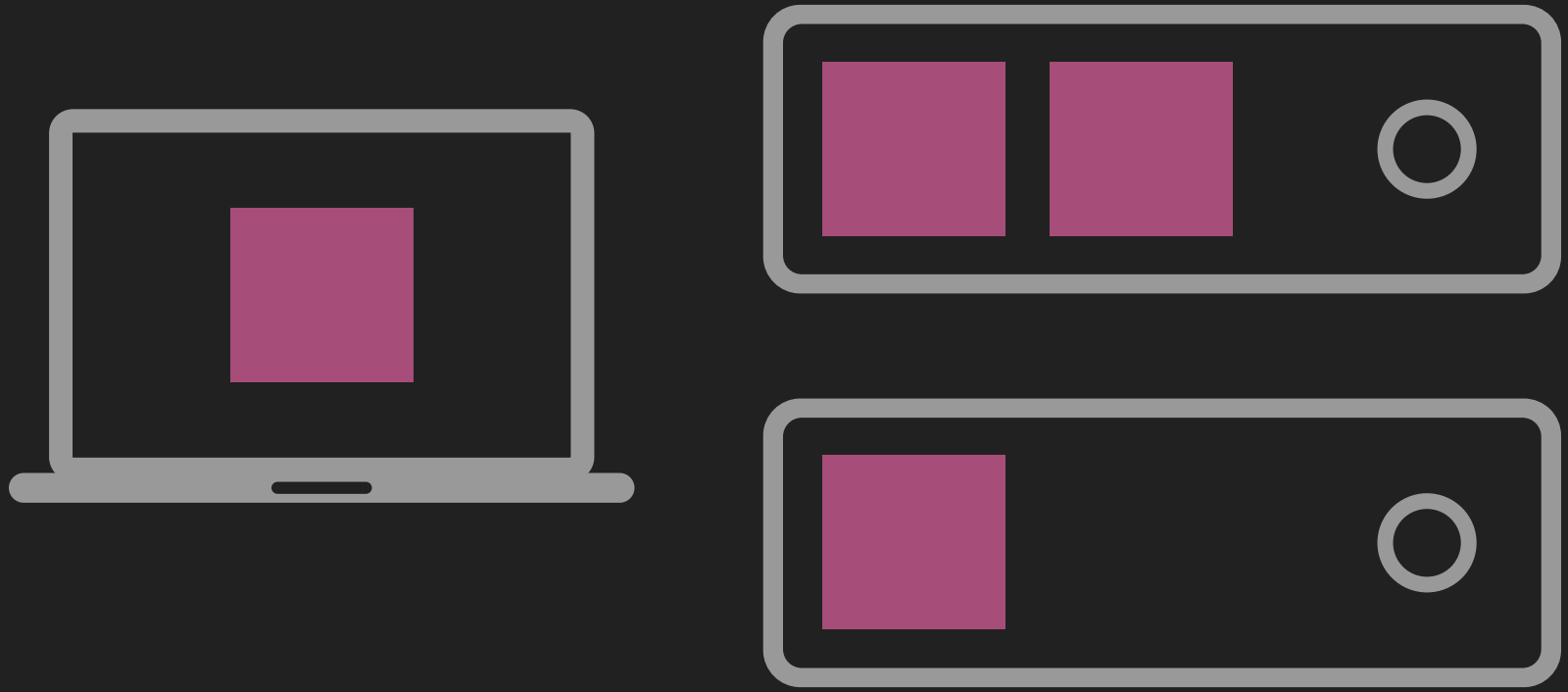
Jitar

Abbreviation:
Just-In-Time ArchitectuRe

https://jitar.dev


Full-stack
Application



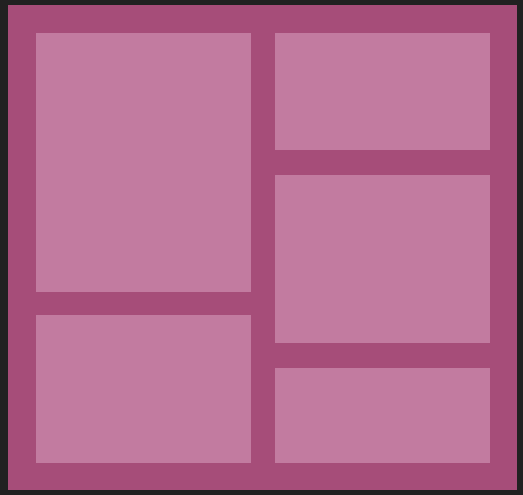



#1 Motivation & goals
#2 Splitting applications
#3 Running applications
#4 Distributing applications
#5 Conclusions & considerations

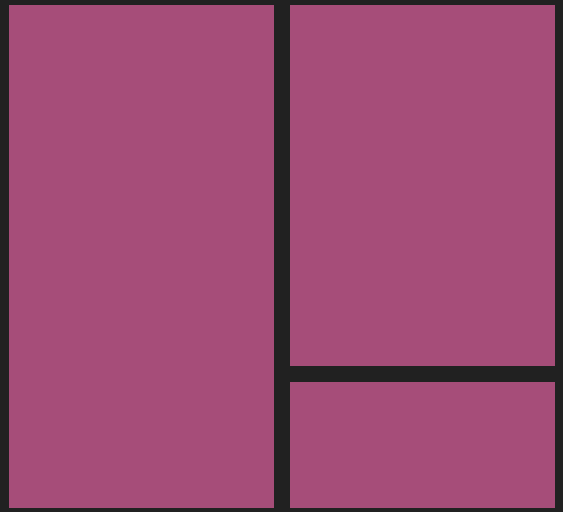
#1

Motivation

Freedom of deployment

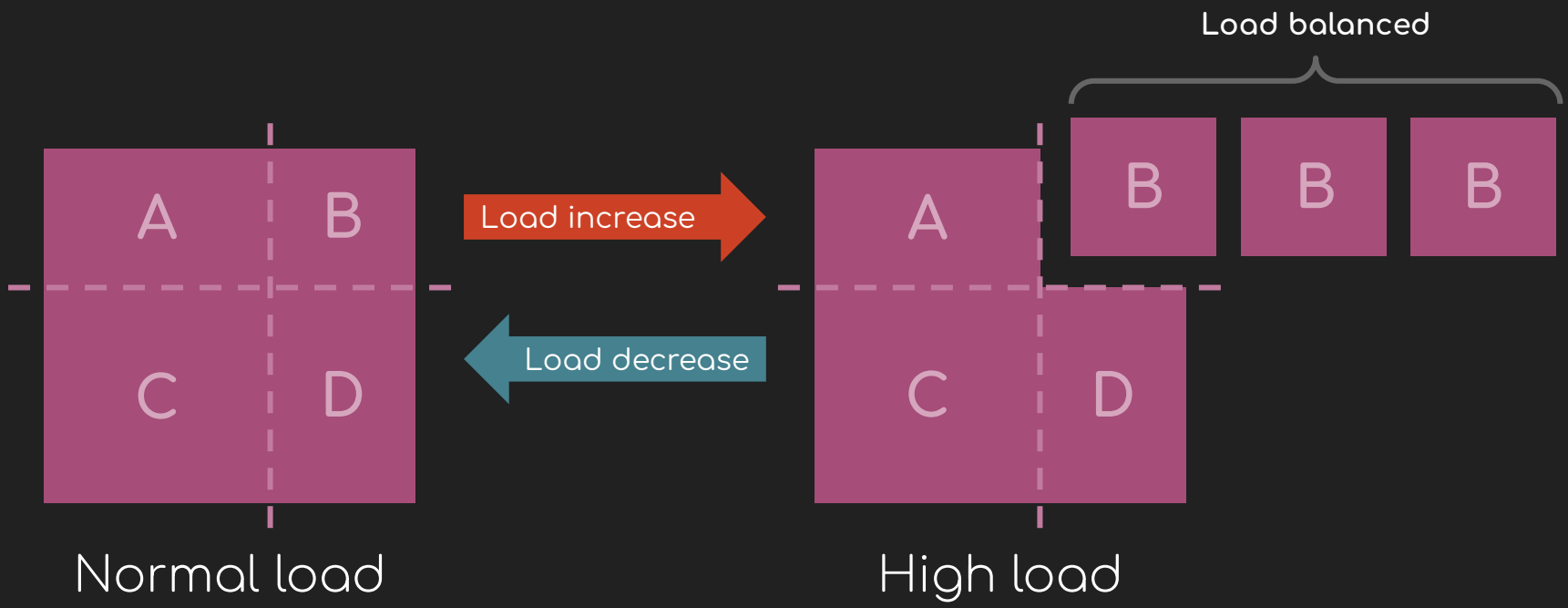


Development
(logical decomposition)

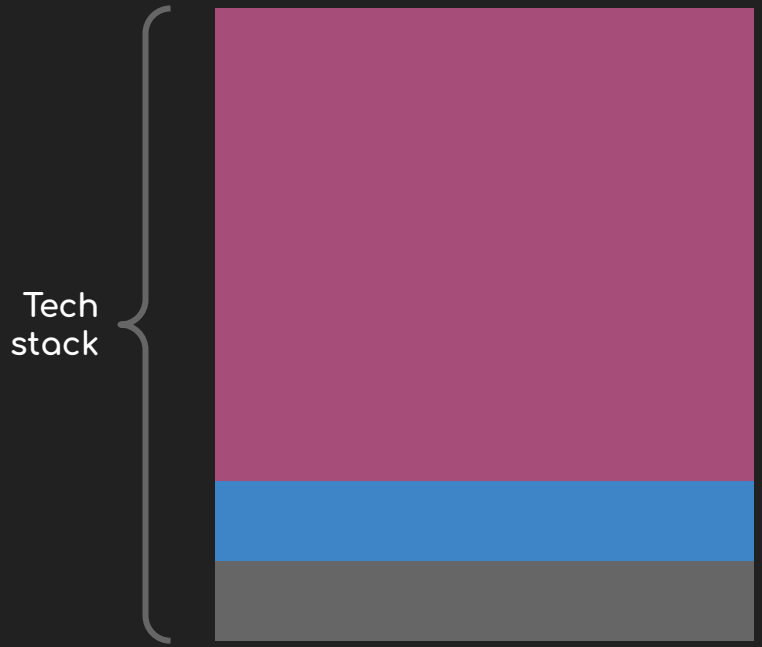


Deployment
(physical decomposition)

Optimal infrastructure use



Simplified overall development



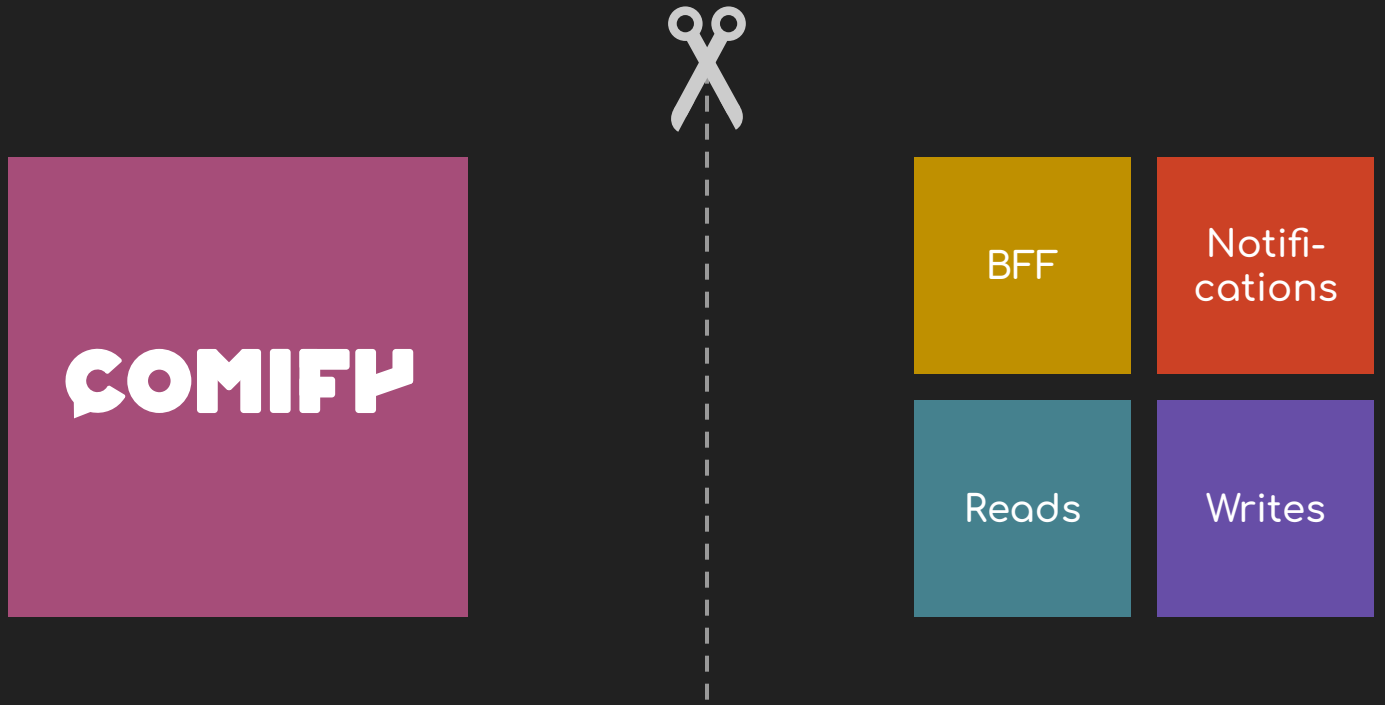
Application
Plain JavaScript without additional dependencies.

Runtime
Handles distribution concerns for the application.

Platform
Runs everywhere: web browser, cloud, container, etc.

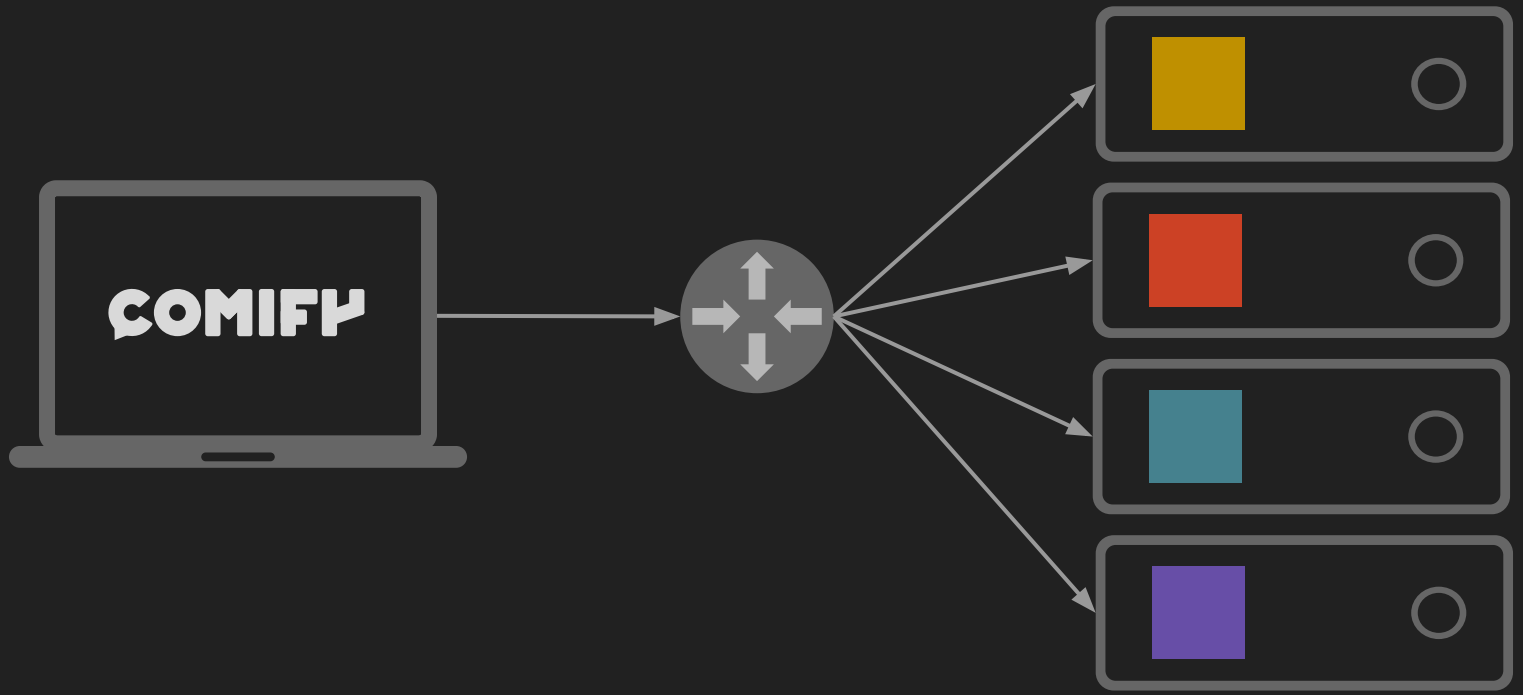

Demo time!

Distribution plan



Standalone deployment

Distributed deployment



#2

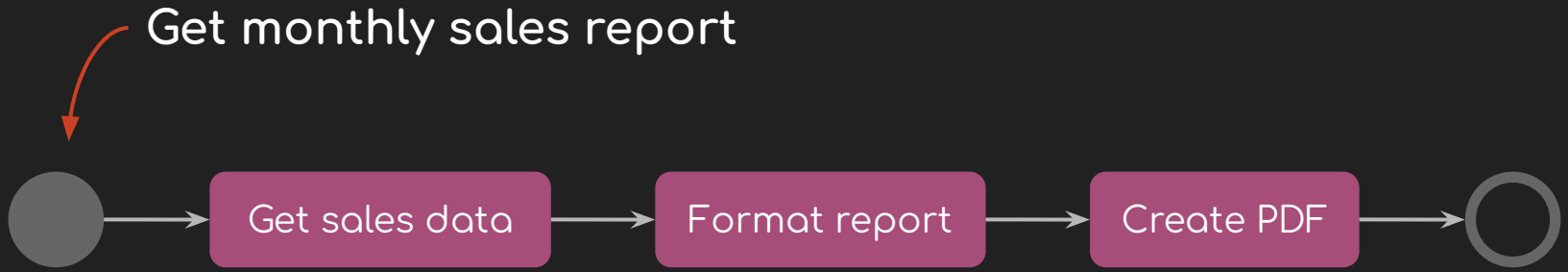
Splitting

Prerequisites

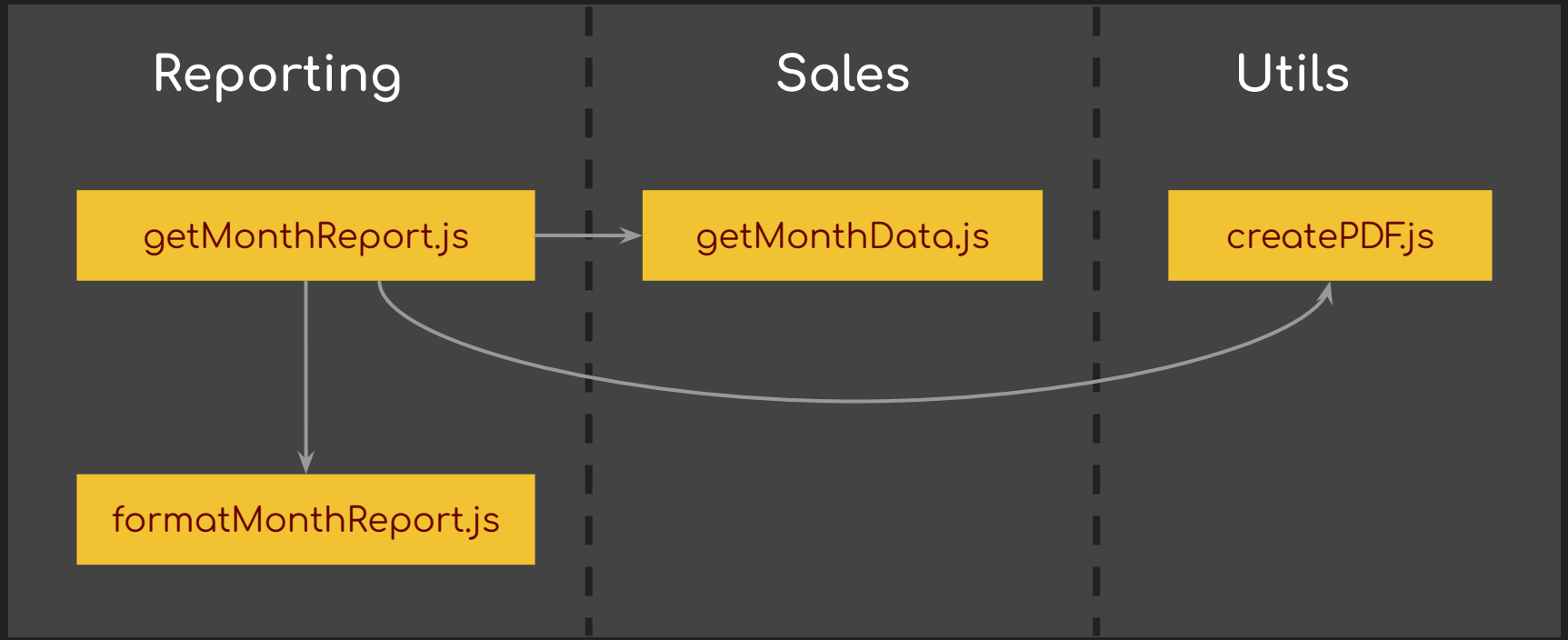
JavaScript
runtime

ECMAScript
modules

The application



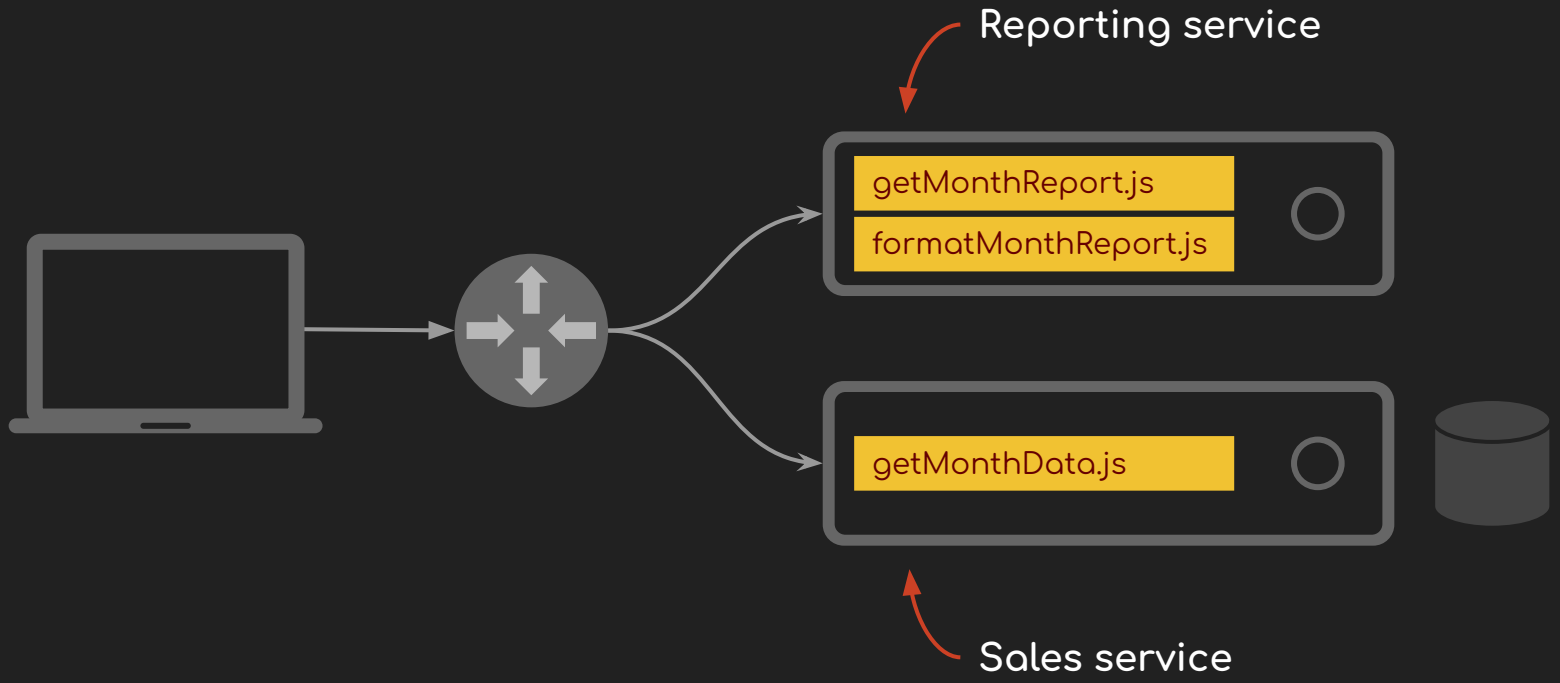
Application structure



Process implementation

```
// reporting/getMonthReport.js

import createPDF from '../utils/createPDF.js';
import getMonthData from '../sales/getMonthData.js';
import formatMonthReport from './formatMonthReport.js';

export default async function getMonthReport(year, month)
{
  const data = await getMonthData(year, month);
  const report = await formatMonthReport(data);

  return createPDF(report);
}
```

Distribution plan



Deployment configuration

```
// reporting.json
{
  "./reporting/getMonthReport.js": {
    "default": {
      "access": "public"
    }
  },
  "./reporting/formatMonthReport.js": {
    "default": {
      "access": "private"
    }
  }
}
```

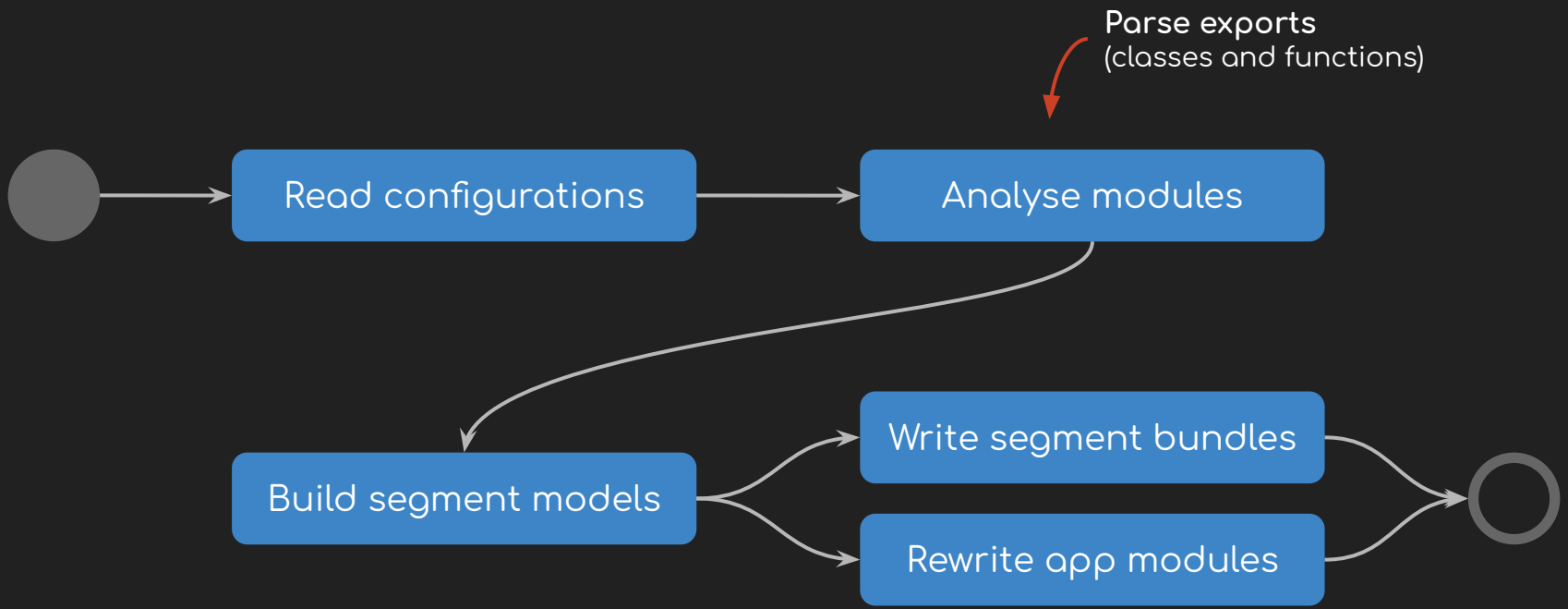
```
// sales.json
{
  "./sales/getMonthData.js": {
    "default": {
      "access": "protected"
    }
  }
}
```

$ jitar build

OUTPUT

```
[INFO] Built reporting segment (2 modules, 2 procedures, 0 classes)
[INFO] Built sales segment (1 modules, 1 procedure, 0 classes)
```

Build process

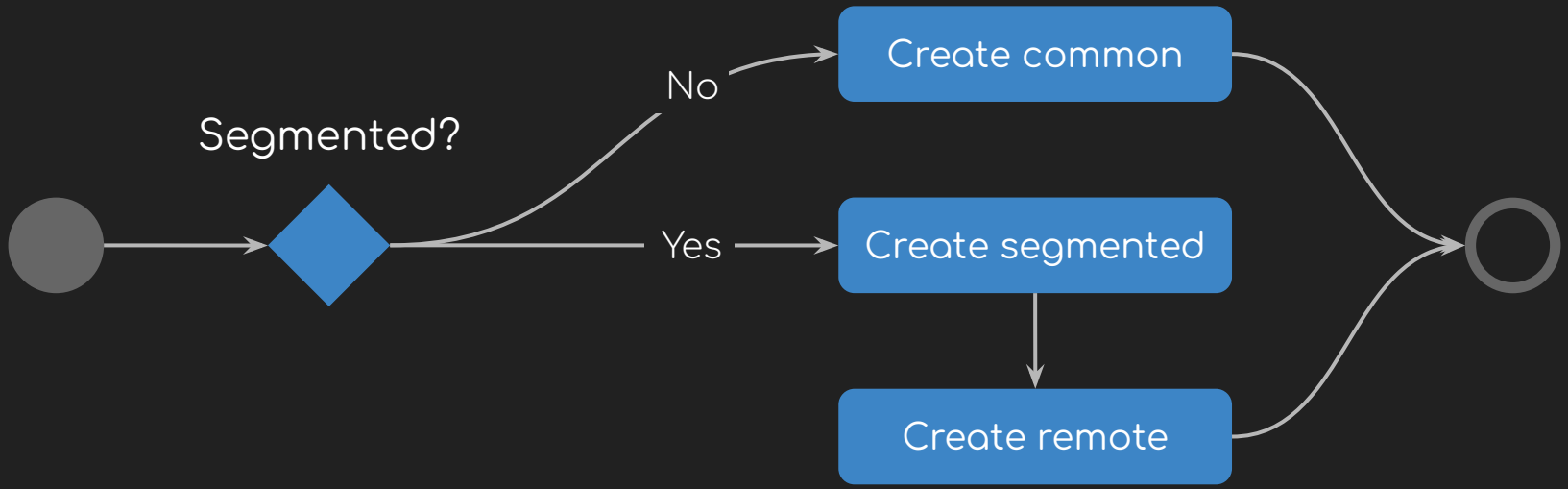


Segment bundle

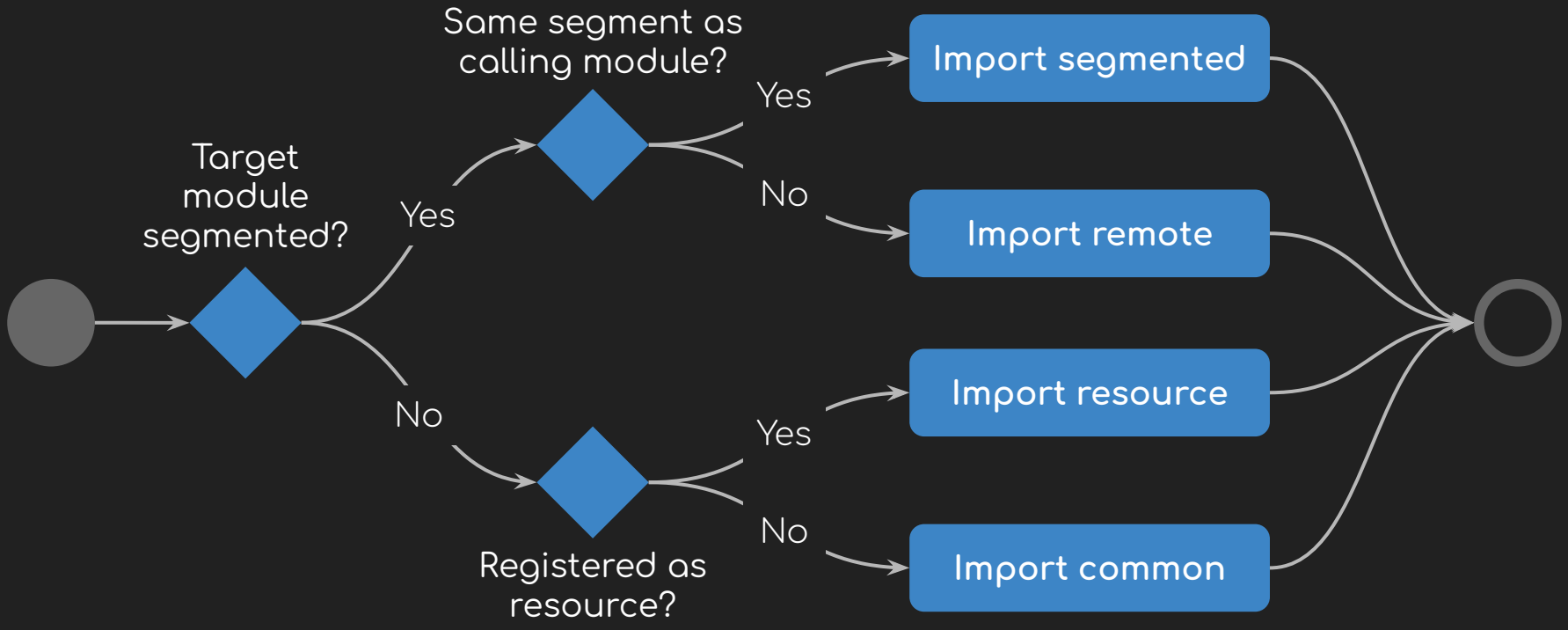
```
import {Segment, Procedure, Implementation, Version, NamedParameter} from "jitar";
import $1 from "./reporting/getMonthReport.reporting.js";
import $2 from "./reporting/formatMonthReport.reporting.js";

export default new Segment("reporting")
  .addProcedure(new Procedure("reporting/getMonthReport")
    .addImplementation(new Implementation(new Version(0, 0, 0), "public",
      [new NamedParameter("year", false), new NamedParameter("month", false)], $1))
    )
  .addProcedure(new Procedure("reporting/formatMonthReport")
    .addImplementation(new Implementation(new Version(0, 0, 0), "private",
      [new NamedParameter("data", false)], $2))
    )
```

FQN

App module rewriting



Import rewriting



Segmented module

```
// reporting/getMonthReport.reporting.js

import createPDF from '../utils/createPDF.js';
import getMonthData from "../sales/getMonthData.remote.js";
import formatMonthReport from "./formatMonthReport.reporting.js";

export default async function getMonthReport(year, month)
{
  const data = await getMonthData(year, month);
  const report = await formatMonthReport(data);

  return createPDF(report);
}
```

Remote module

```
// sales/getMonthData.remote.js

export default async function getMonthData(year, month)
{
  return __run('sales/getMonthData', '0.0.0', {'year':year, 'month':month}, this);
}
```

FQN

Version

Args

Context

#3

Running

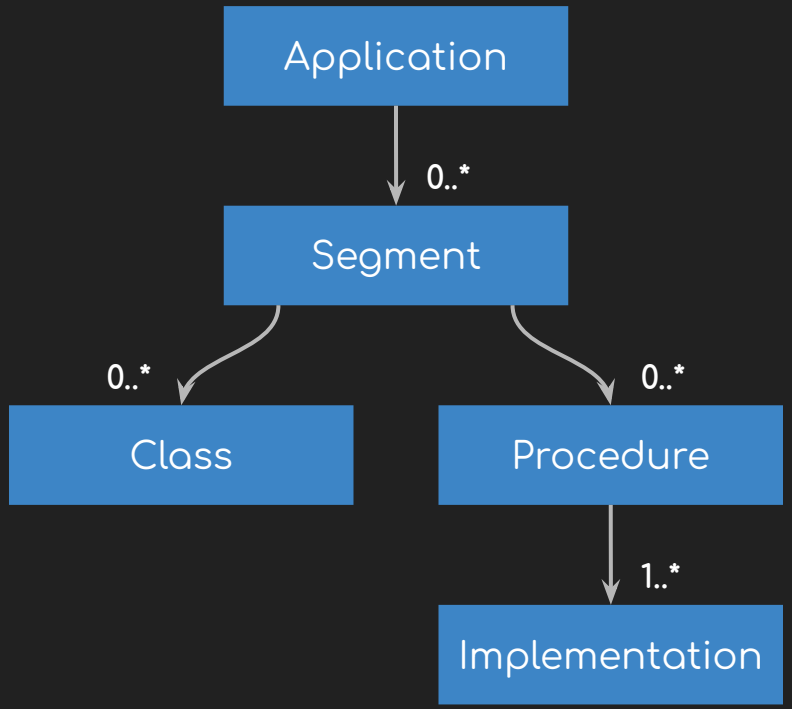
$ jitar start

--service=reporting.json

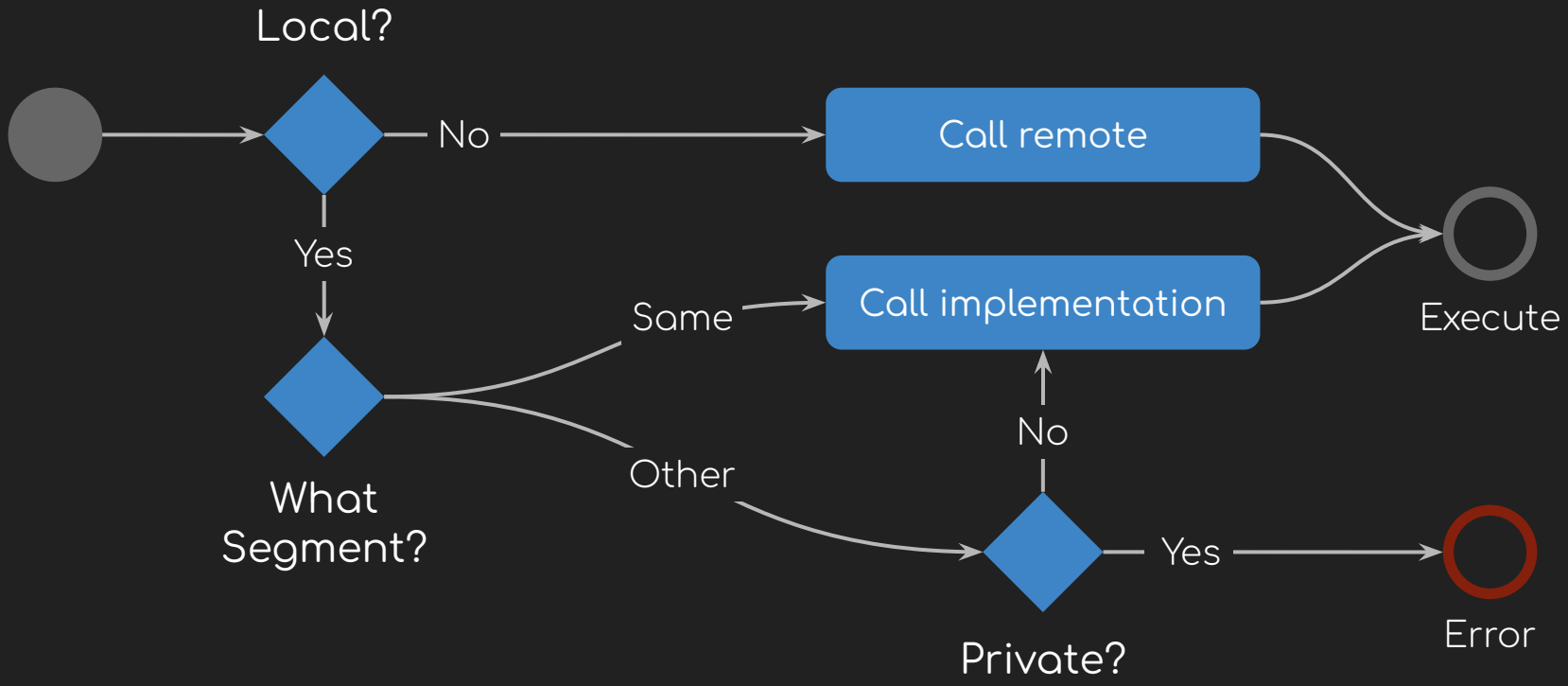
OUTPUT

```
[INFO] Server started at https://example.com:3000
[INFO] RPC procedures: [
  reporting/getMonthReport
]
```

Execution model



__run

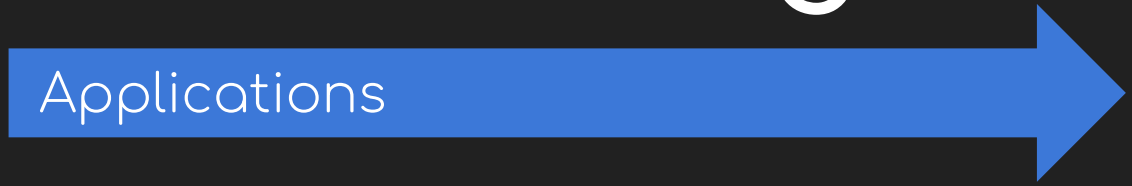

Local?
No
Call remote
Yes
Same
Call implementation
Execute
What
Segment?
Other
No
Yes
Private?
Error

RPC API


```
POST https://example.com/rpc/reporting/getMonthReport
content-type: application/json

{
    "year": 2025,
    "month": "April"
}
```

#4

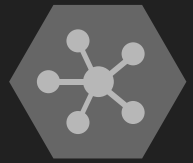
Distributing



Requirements

Routing

Discovery



Interaction

Balancing

Monitoring

Security

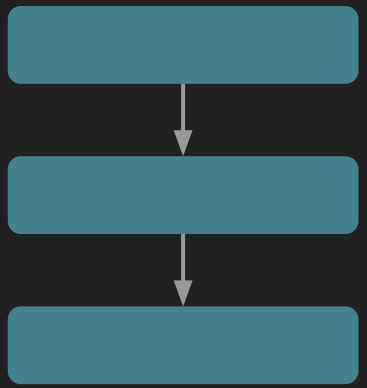
Prevention

Tolerance

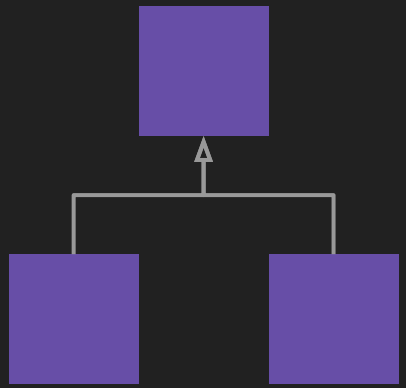
#5

Conclusions

Programming paradigms

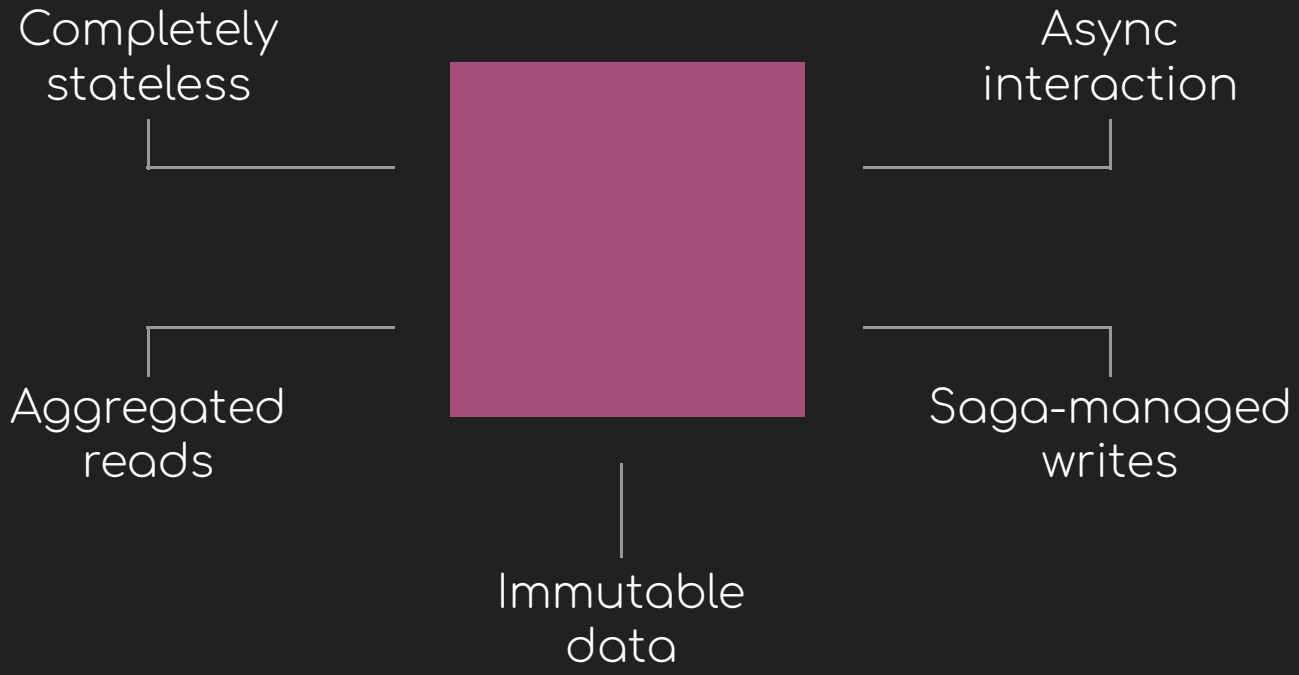


Procedural
(PRC)

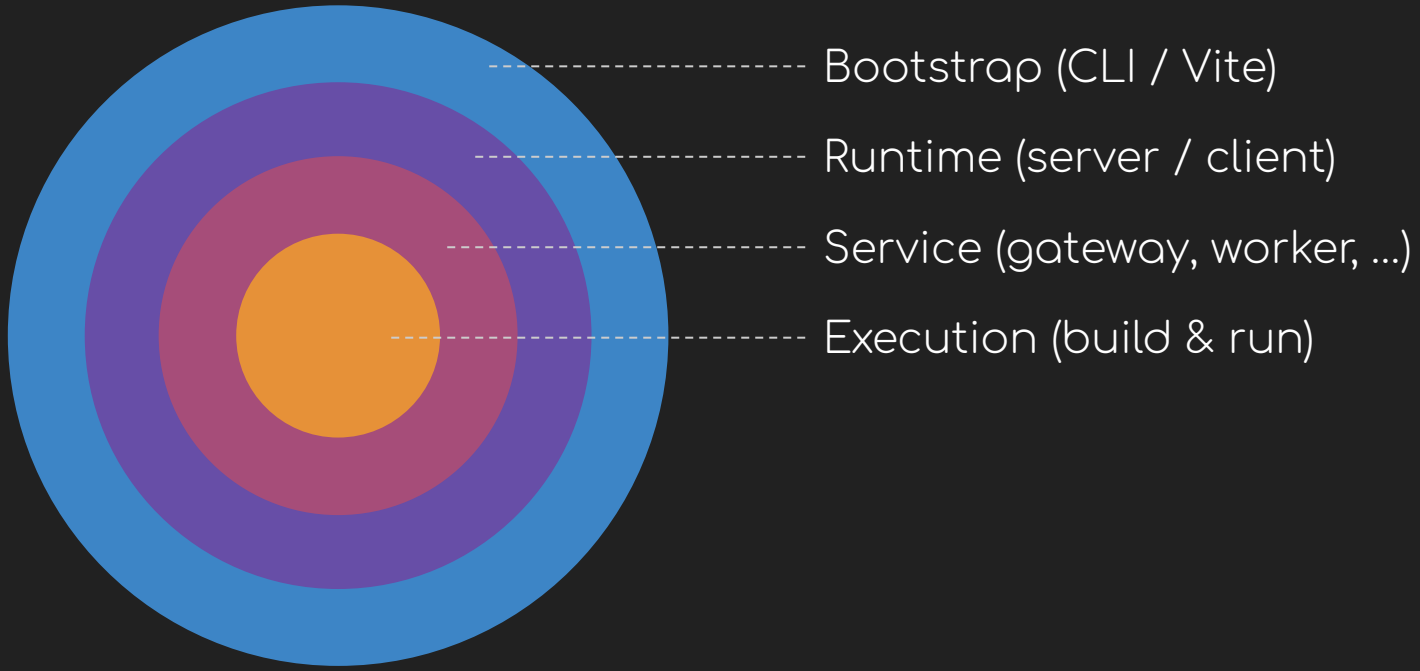


Object-oriented
(serialization)

Application constraints



Overall architecture



Thanks!

https://masking.tech