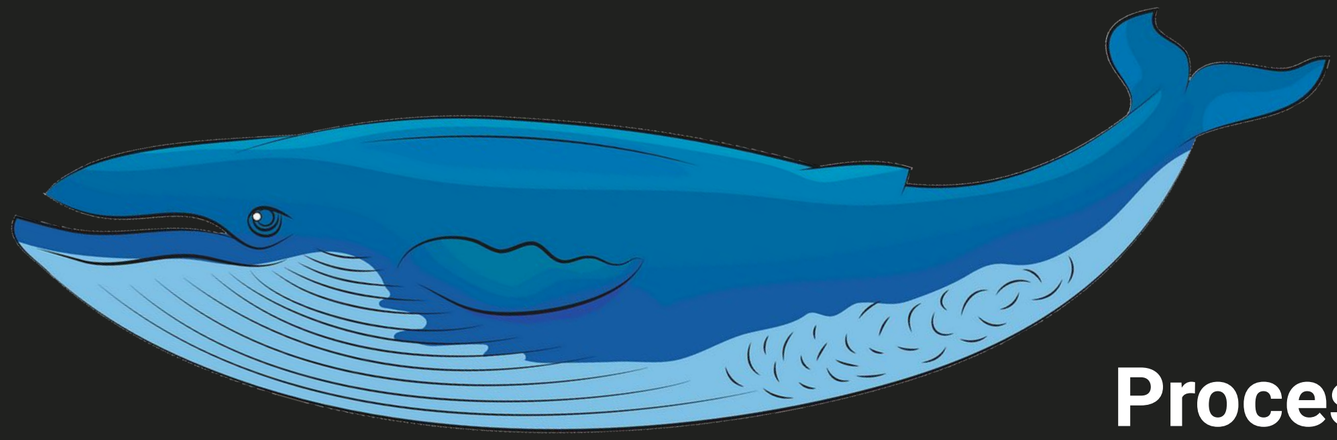
JavaScript
fwdays

Metarhia Technology Stack for Node.js

TIMUR SHEMSEDINOV

CTO at Salucyber, Lecturer at KPI and KSE,
Chief Architect at Metarhia

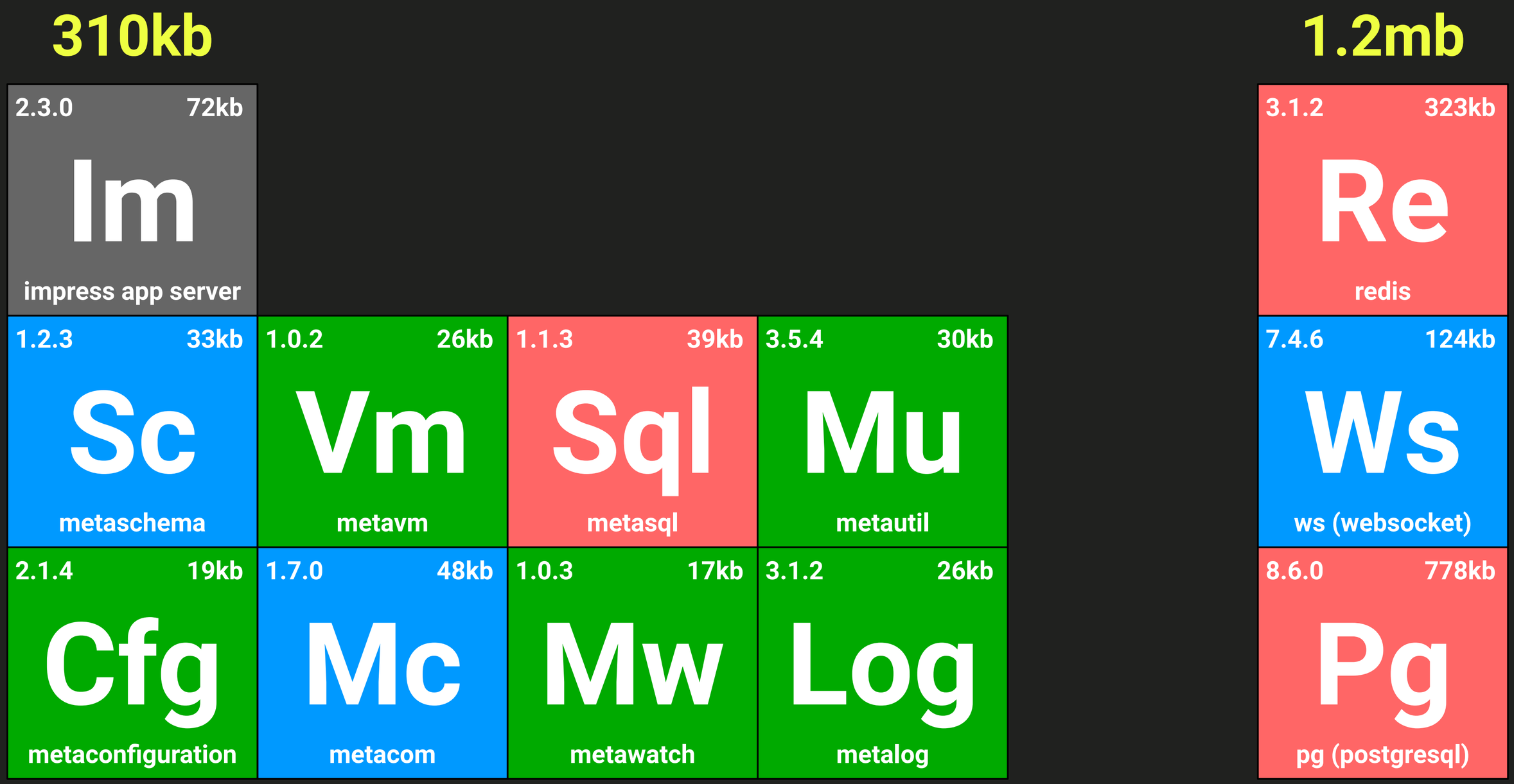
Scale
with threads



Process

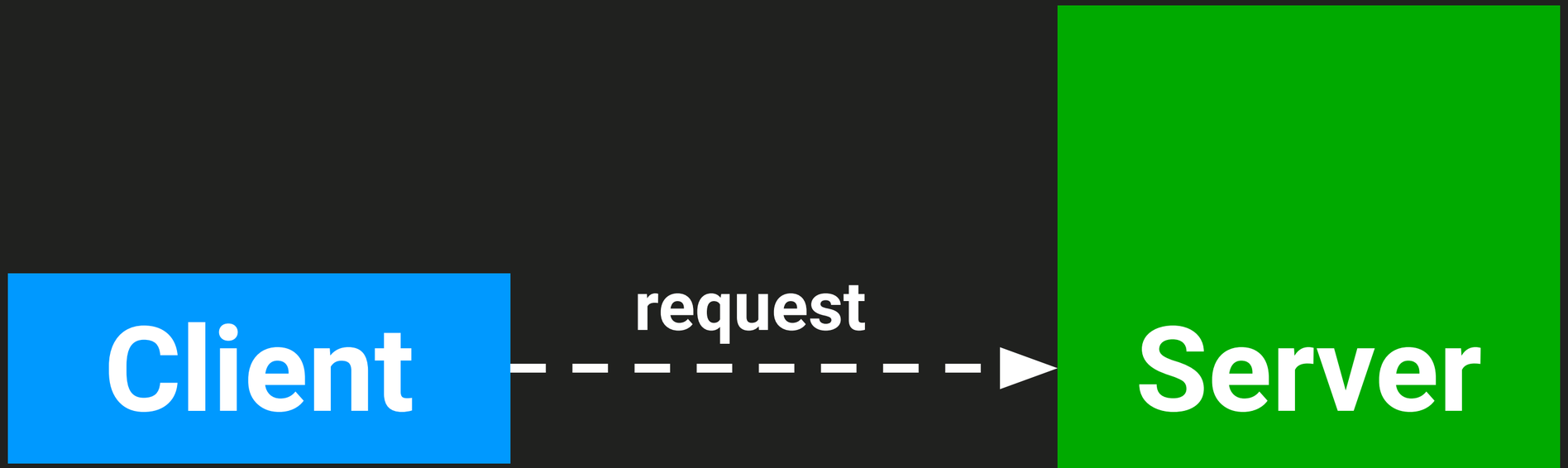
Thread

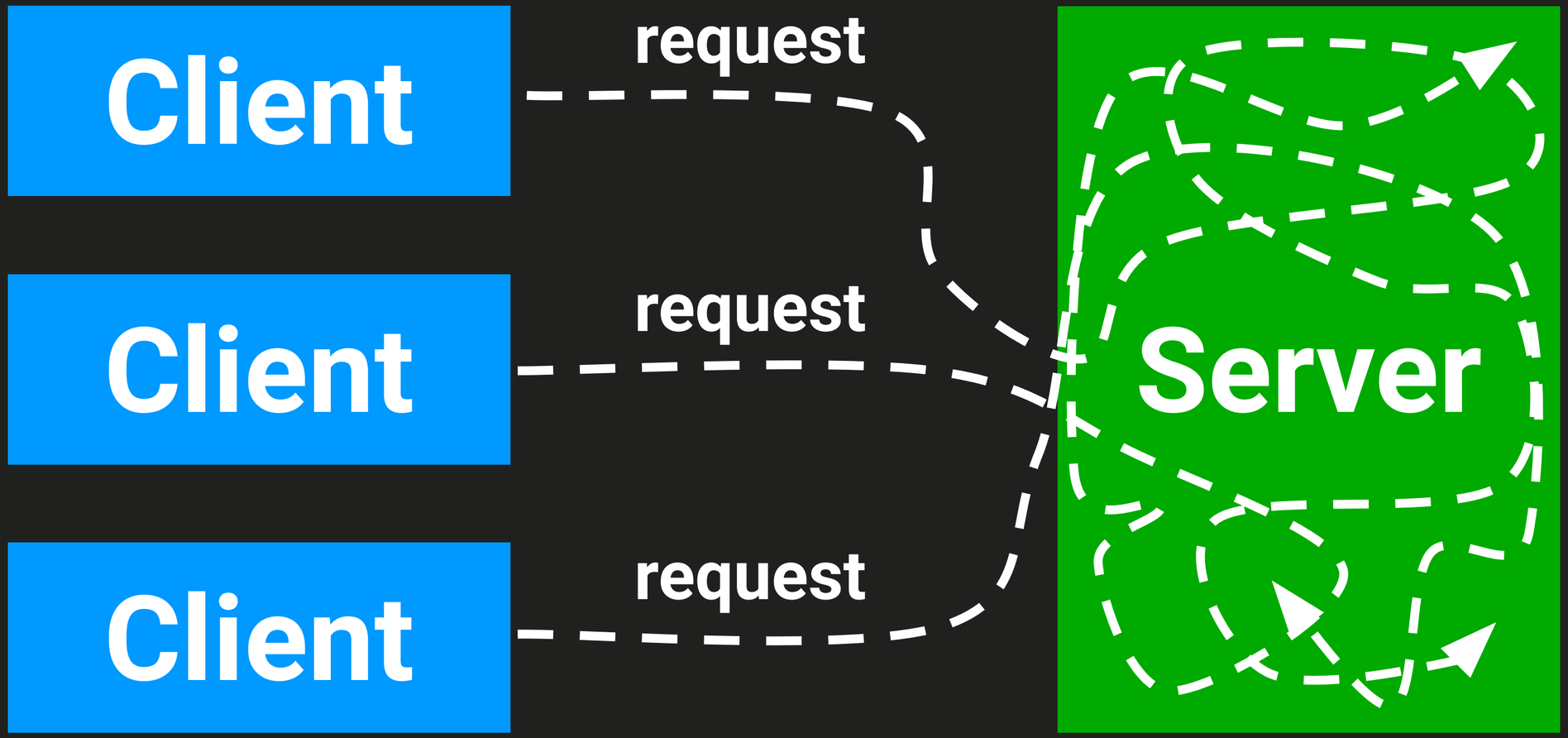
Application Server
For enterprise backend
Security, Reliability
DDD, Clean architecture
GRASP, SOLID, GoF
Separate system and applier layers
Simplicity, Quick start
Tech stack agnostic

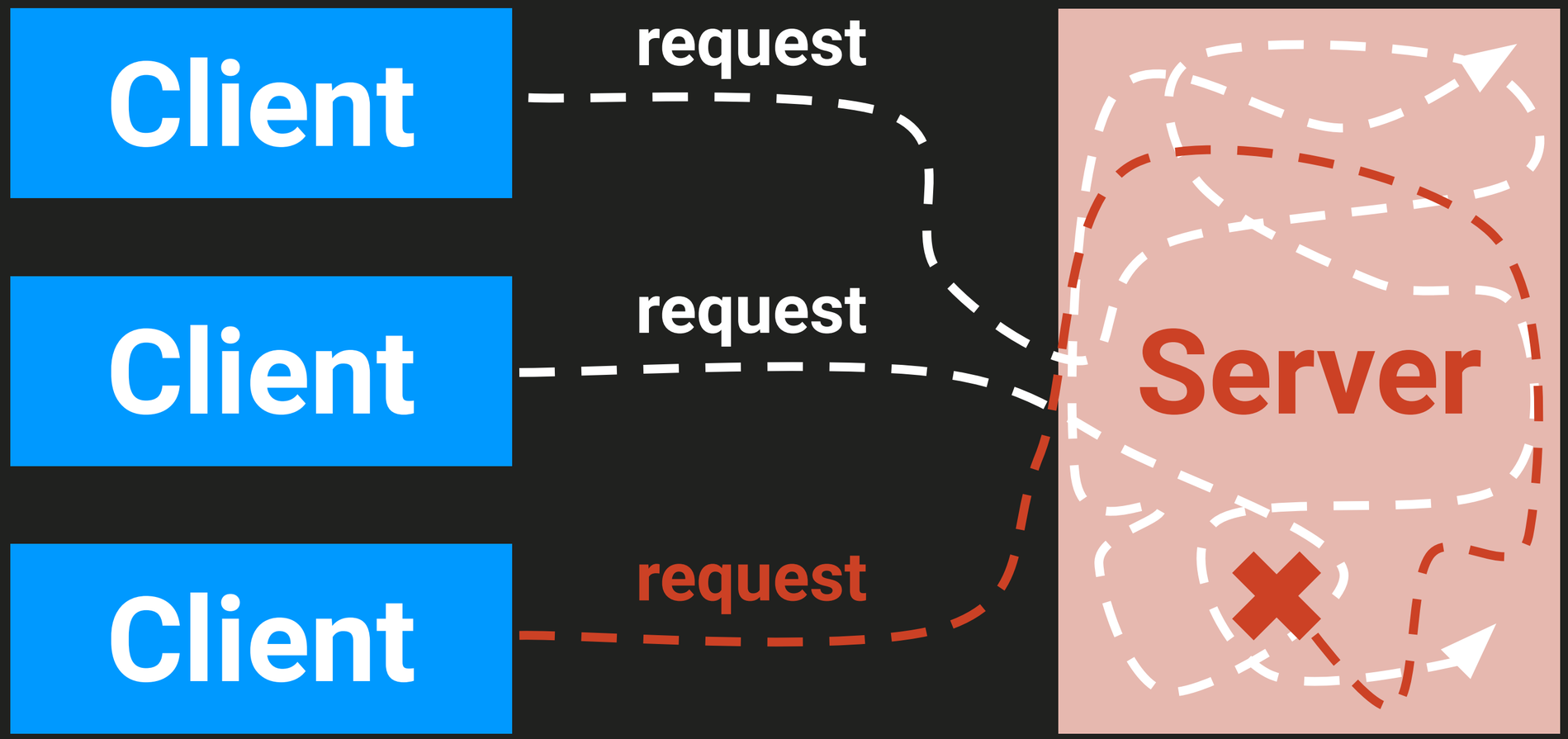

310kb
1.2mb
2.3.0
72kb
Im
impress app server
3.1.2
323kb
Re
redis
1.2.3
33kb
Sc
metaschema
1.0.2
26kb
Vm
metavm
1.1.3
39kb
Sql
metasql
3.5.4
30kb
Mu
metautil
7.4.6
124kb
Ws
ws (websocket)
2.1.4
19kb
Cfg
metaconfiguration
1.7.0
48kb
Mc
metacom
1.0.3
17kb
Mw
metawatch
3.1.2
26kb
Log
metalog
8.6.0
778kb
Pg
pg (postgresql)

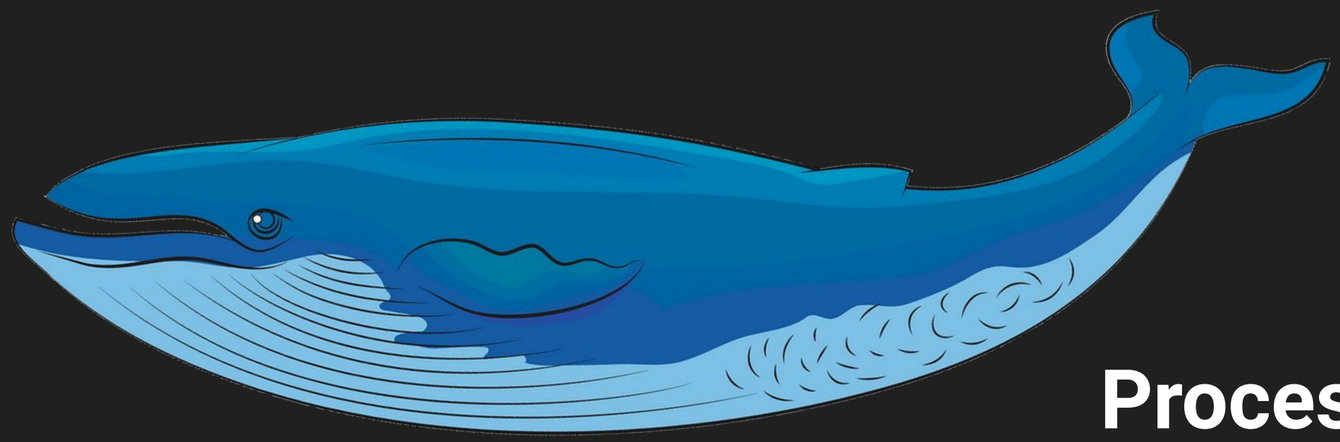
No middlewares
No mixins
No memory leaks
No race conditions
No integration tasks
No system programming
No global
No require

Isolation


Client
request
Server


Client
Client
Client
request
request
request
Server


Client
Client
Client
request
request
request
Server



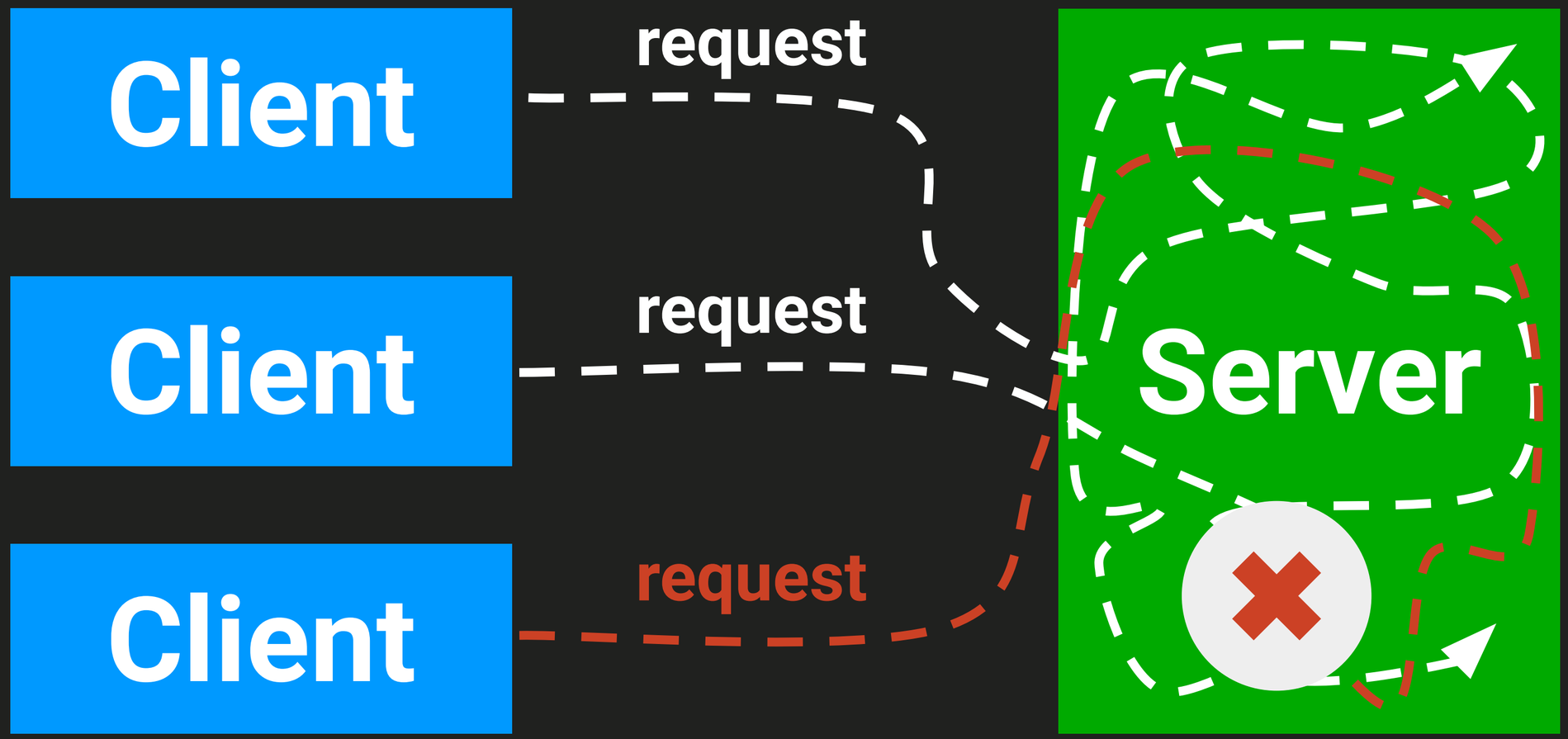
Process

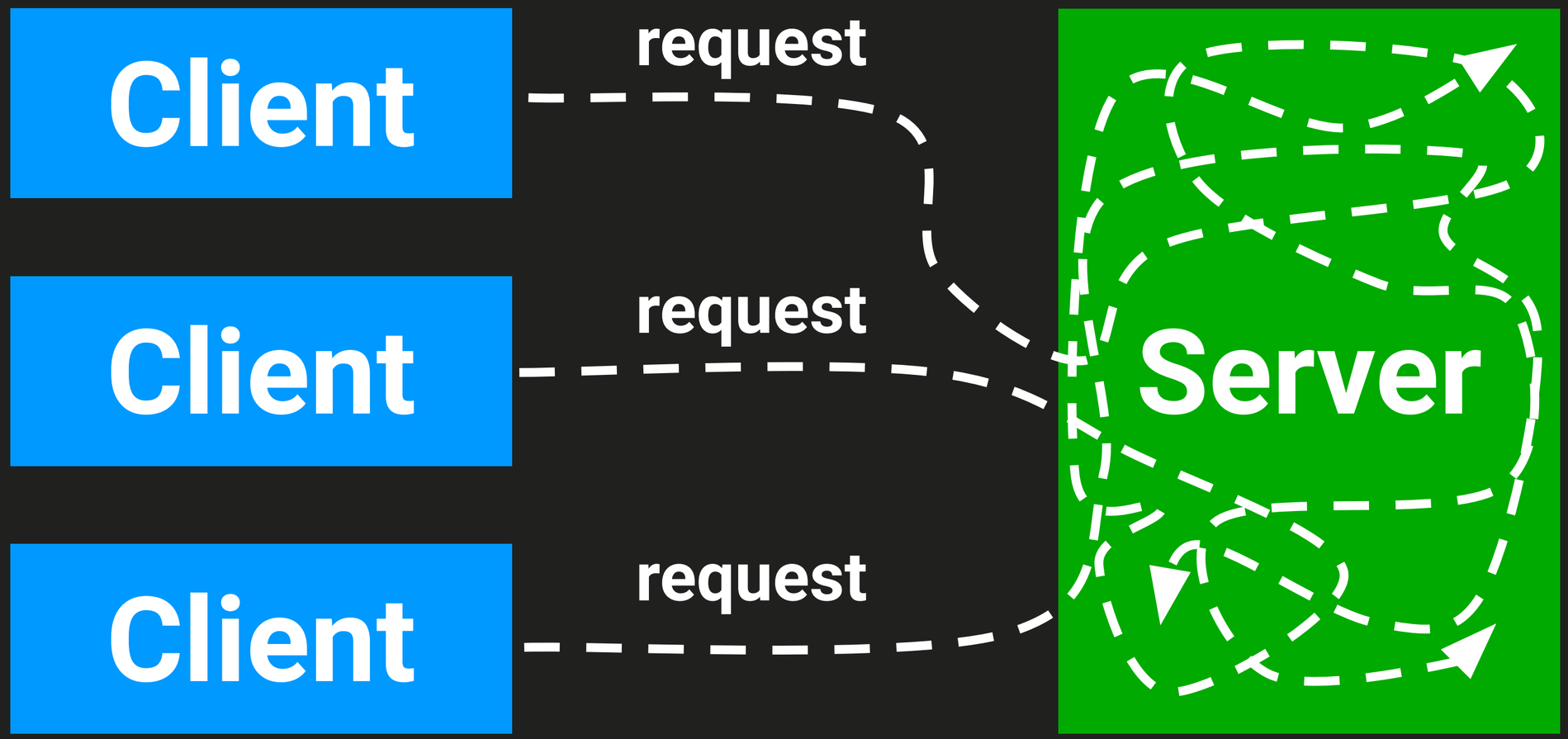
Thread

v8::Isolate

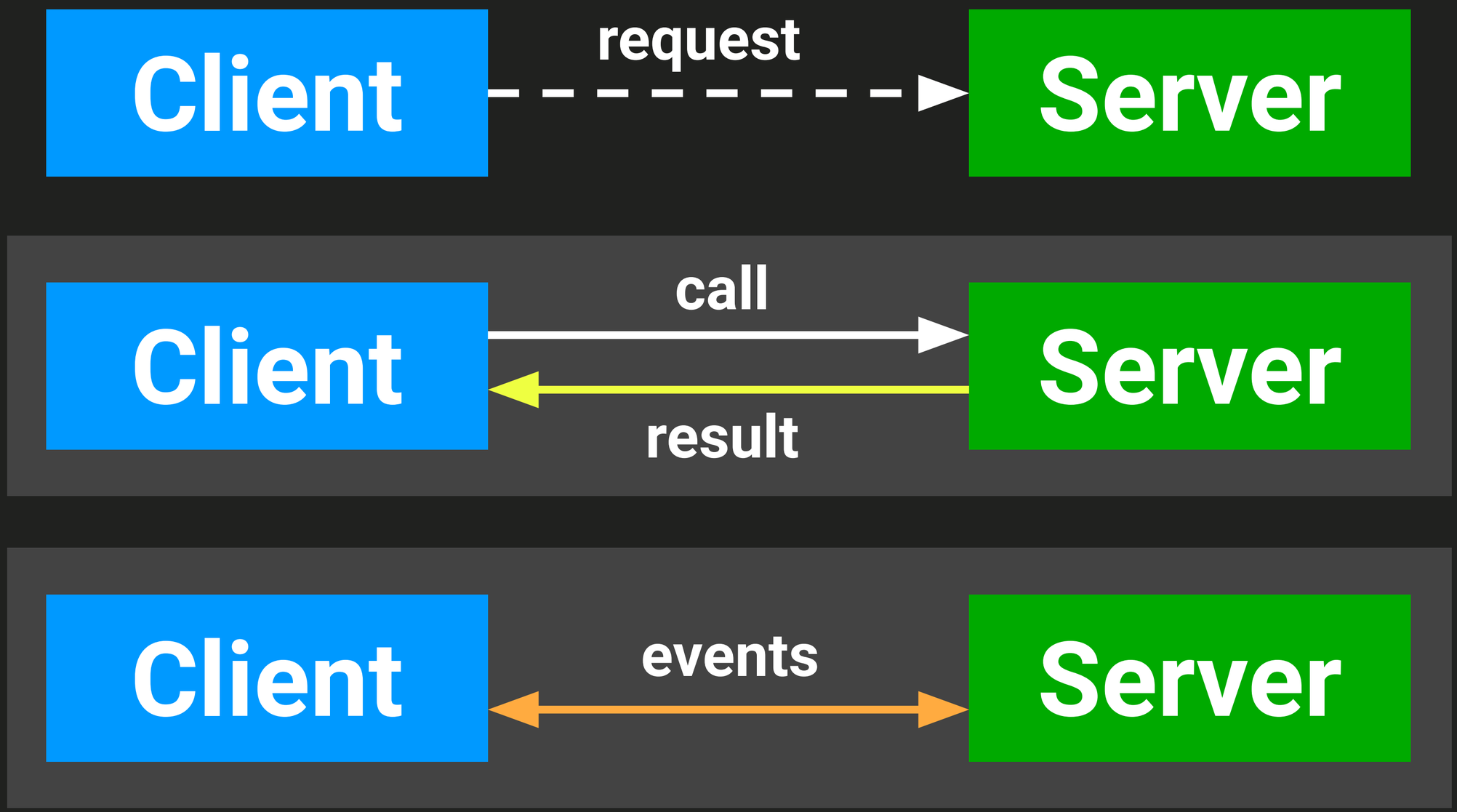
v8::Context

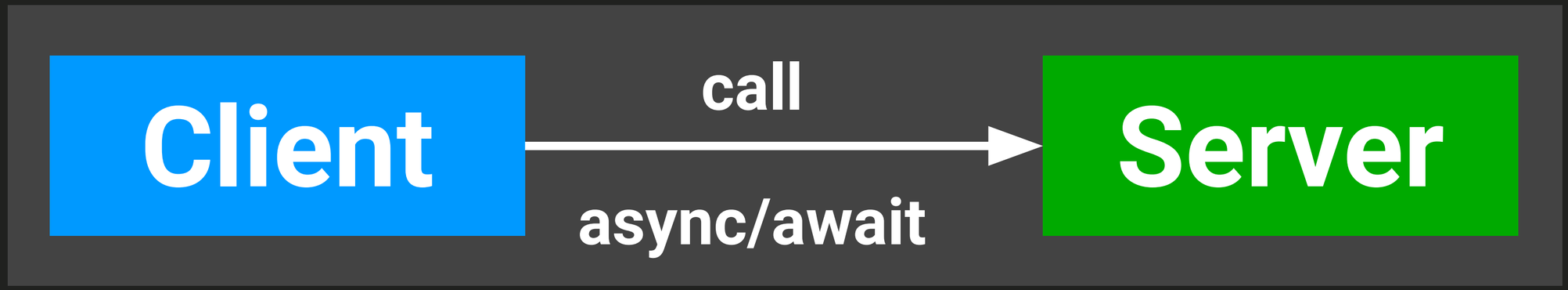
Closure


Client
request
Client
request
Client
request
Server


Client
Client
Client
request
request
request
Server

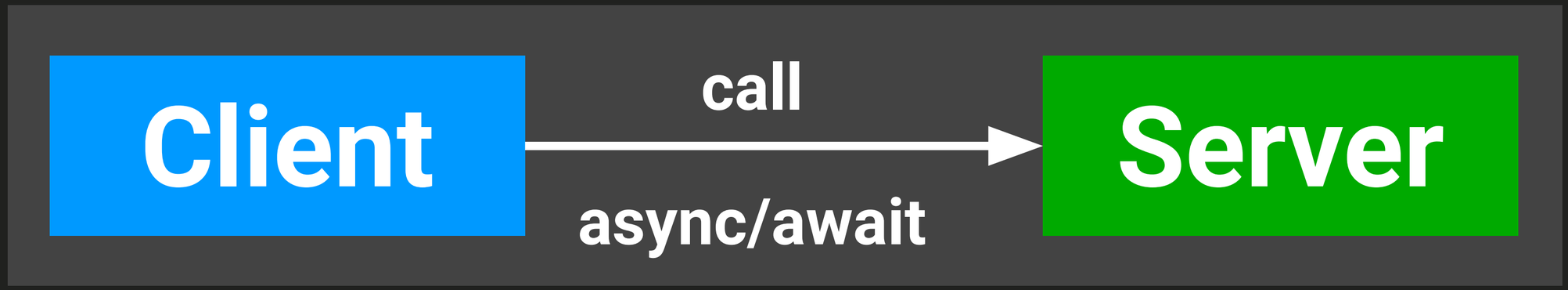
Simple


Client
request
Server
Client
call
result
Server
Client
events
Server



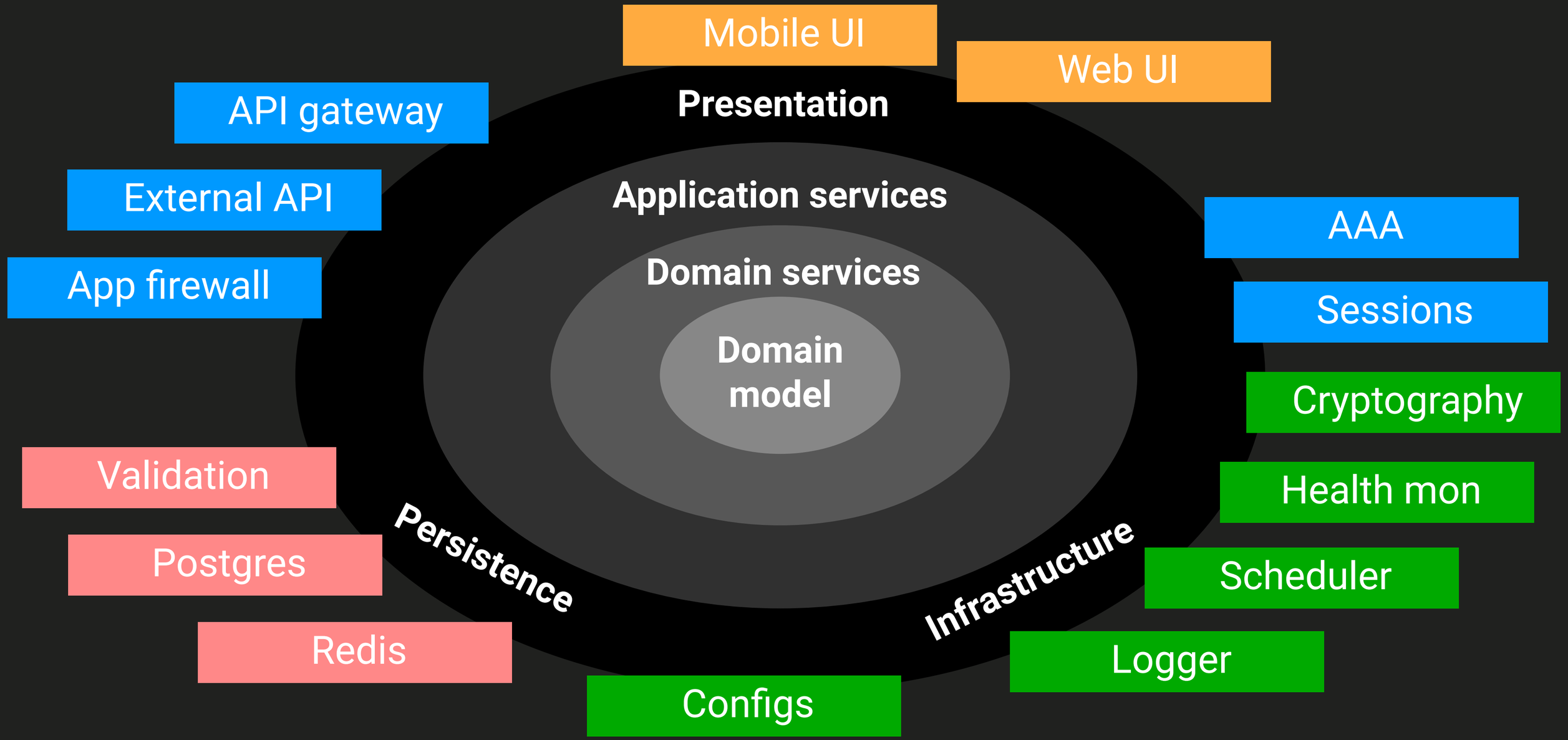

```
const c = await api.math.sum({ a, b });
```

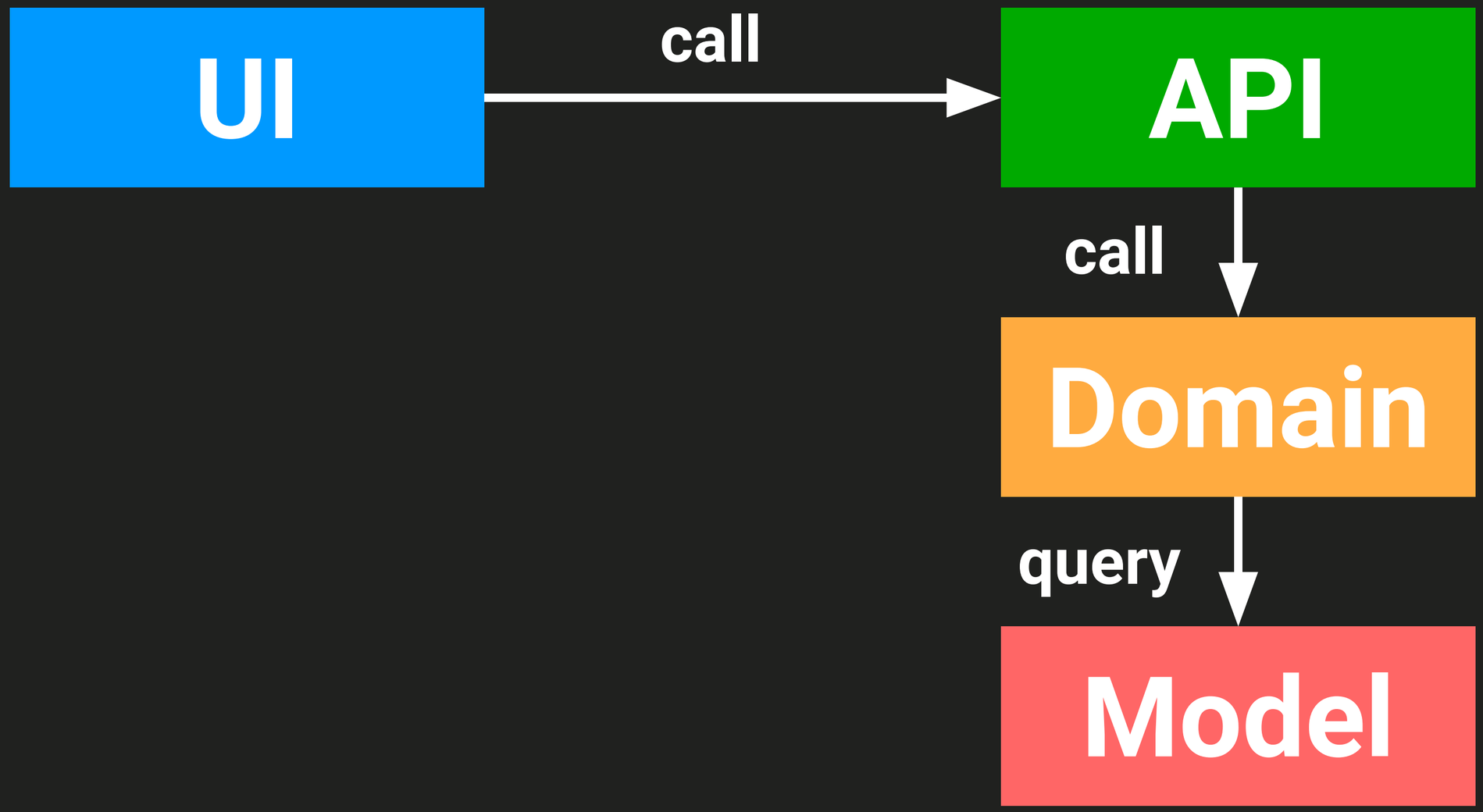
```
// application/api/math/sum.js
async ({ a, b }) => {
  return a + b;
};
```



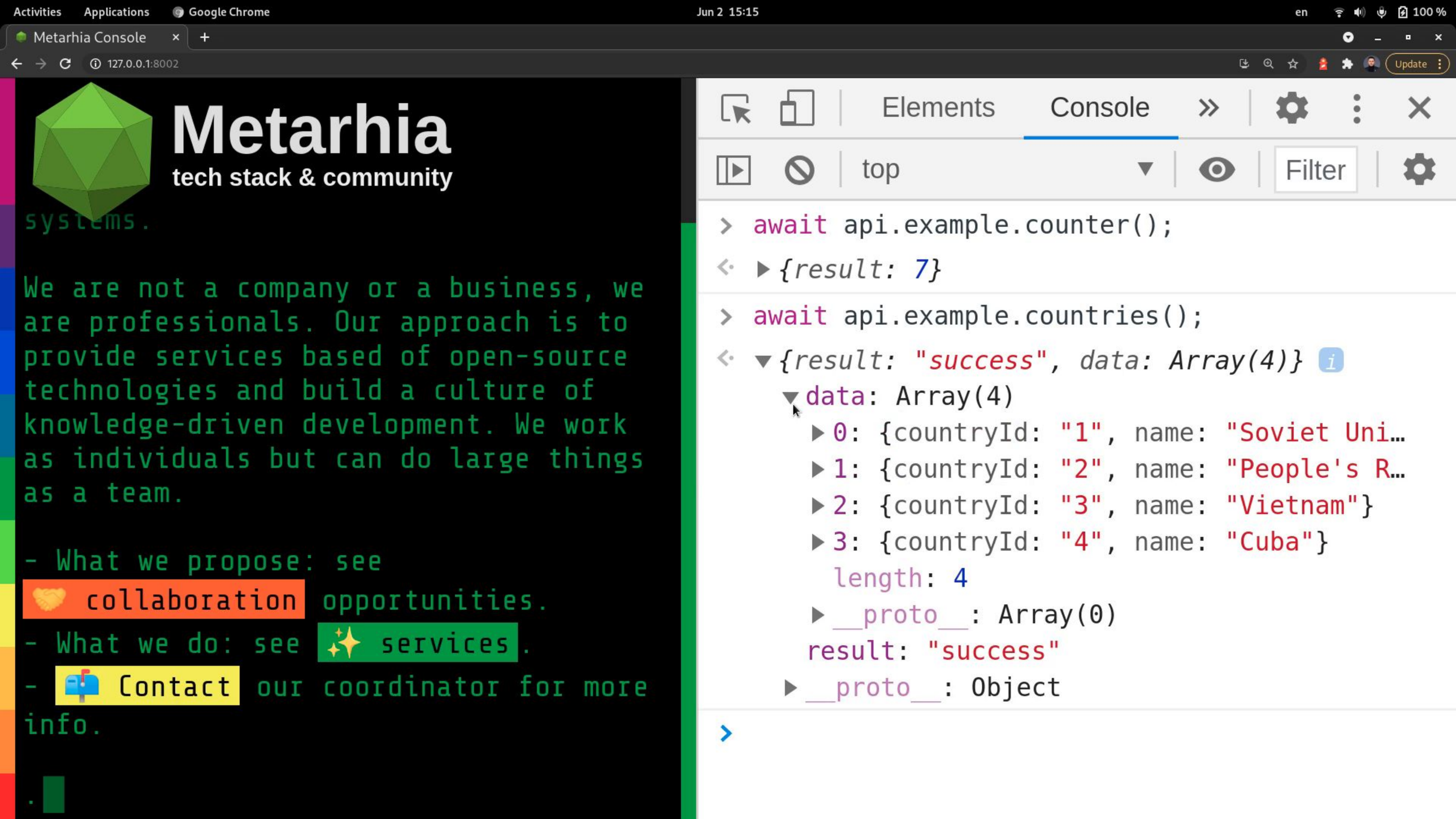
Communication-agnostic
Transport-agnostic
Serialization-agnostic

Clean Architecture


Mobile UI
Web UI
Presentation
API gateway
External API
App firewall
Application services
Domain services
Domain model
AAA
Sessions
Cryptography
Health mon
Scheduler
Logger
Validation
Postgres
Redis
Persistence
Infrastructure
Configs


UI
call
API
call
Domain
query
Model

```
async ({ accountId, address }) => {
  const account = await domain.getAccount(accountId);
  if (!account) {
    return { result: 'account not found' };
  }
  if (account.addresses.includes(address)) {
    return { result: 'already known address' };
  }
  await domain.addAccountAddress(address);
  return { result: 'successfully added' };
};
```


Activities
Applications
Google Chrome
Jun 2 15:15
en
100 %
Metarhia Console
127.0.0.1:8002
Update
Metarhia
tech stack & community
systems.
We are not a company or a business, we are professionals. Our approach is to provide services based of open-source technologies and build a culture of knowledge-driven development. We work as individuals but can do large things as a team.
- What we propose: see
collaboration
opportunities.
- What we do: see
services
.
-
Contact
our coordinator for more info.
Elements
Console
top
Filter
await api.example.counter();
{result: 7}
await api.example.countries();
{result: "success", data: Array(4)}
data: Array(4)
0: {countryId: "1", name: "Soviet Uni…
1: {countryId: "2", name: "People's R…
2: {countryId: "3", name: "Vietnam"}
3: {countryId: "4", name: "Cuba"}
length: 4
__proto__: Array(0)
result: "success"
__proto__: Object

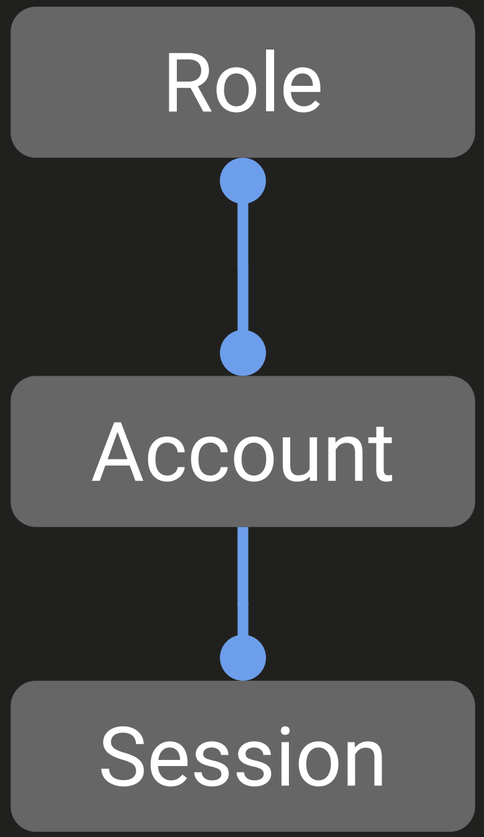
Contracts

```
({
  parameters: {
    a: 'number',
    b: 'number',
  },

  async method({ a, b }) => {
    return a + b;
  },

  returns: 'number',
});
```

```
({
  parameters: {
    name: 'string',
    age: { type: 'number', min: 16 },
    address: {
      city: 'string',
      street: '?string',
      building: '?number',
    },
  },

  async method({ name, age, address }) => {
    const processed = await db.select('Account', { name });
    if (processed) return { exists: true, status: 'registered' };
    const cityId = await db.select('City', { address.city });
    const accountId = await db.insert('Account', { name, age, city });
    return { exists: true, status: 'inserted' };
  },

  returns: {
    exists: 'boolean',
    status: { enum: ['inserted', 'restricted', 'registered'] },
  },
});
```

Schemas

```
// application/schemas/Account.js
({
  login: { type: 'string', length: { min: 8, max: 64 }, unique: true },
  password: { type: 'string', note: 'Password hash' },
  roles: { many: 'Role' },
});

// application/schemas/Session.js
({
  account: 'Account',
  token: { type: 'string', unique: true },
  ip: 'ip',
  data: 'json',
});

// application/schemas/Role.js
({
  name: { type: 'string', unique: true },
});
```



```ts
interface Role {
  roleId: number;
  name: string;
}

interface Account {
  accountId: number;
  login: string;
  password: string;
}

interface Session {
  sessionId: number;
  accountId: number;
  token: string;
  ip: string;
  data: string;
}
```

```sql
CREATE TABLE "Role" (
  "roleId" bigint generated always as identity,
  "name" varchar NOT NULL
);
ALTER TABLE "Role" ADD CONSTRAINT "pkRole" PRIMARY KEY ("roleId");

CREATE TABLE "Account" (
  "accountId" bigint generated always as identity,
  "login" varchar(64) NOT NULL,
  "password" varchar NOT NULL
);
ALTER TABLE "Account" ADD CONSTRAINT "pkAccount" PRIMARY KEY ("accountId");

CREATE TABLE "AccountRole" (
  "accountId" bigint NOT NULL,
  "roleId" bigint NOT NULL
);
ALTER TABLE "AccountRole" ADD CONSTRAINT "pkAccountRole" PRIMARY KEY ("accountId", "roleId");
ALTER TABLE "AccountRole" ADD CONSTRAINT "fkAccountRoleAccount" FOREIGN KEY ("accountId") REFERENCES "Account"
("accountId");
ALTER TABLE "AccountRole" ADD CONSTRAINT "fkAccountRoleRole" FOREIGN KEY ("roleId") REFERENCES "Role" ("roleId");

CREATE TABLE "Session" (
  "sessionId" bigint generated always as identity,
  "accountId" bigint NOT NULL,
  "token" varchar NOT NULL,
  "ip" inet NOT NULL,
  "data" jsonb NOT NULL
);
ALTER TABLE "Session" ADD CONSTRAINT "pkSession" PRIMARY KEY ("sessionId");
ALTER TABLE "Session" ADD CONSTRAINT "fkSessionAccount" FOREIGN KEY ("accountId") REFERENCES "Account" ("accountId");
```

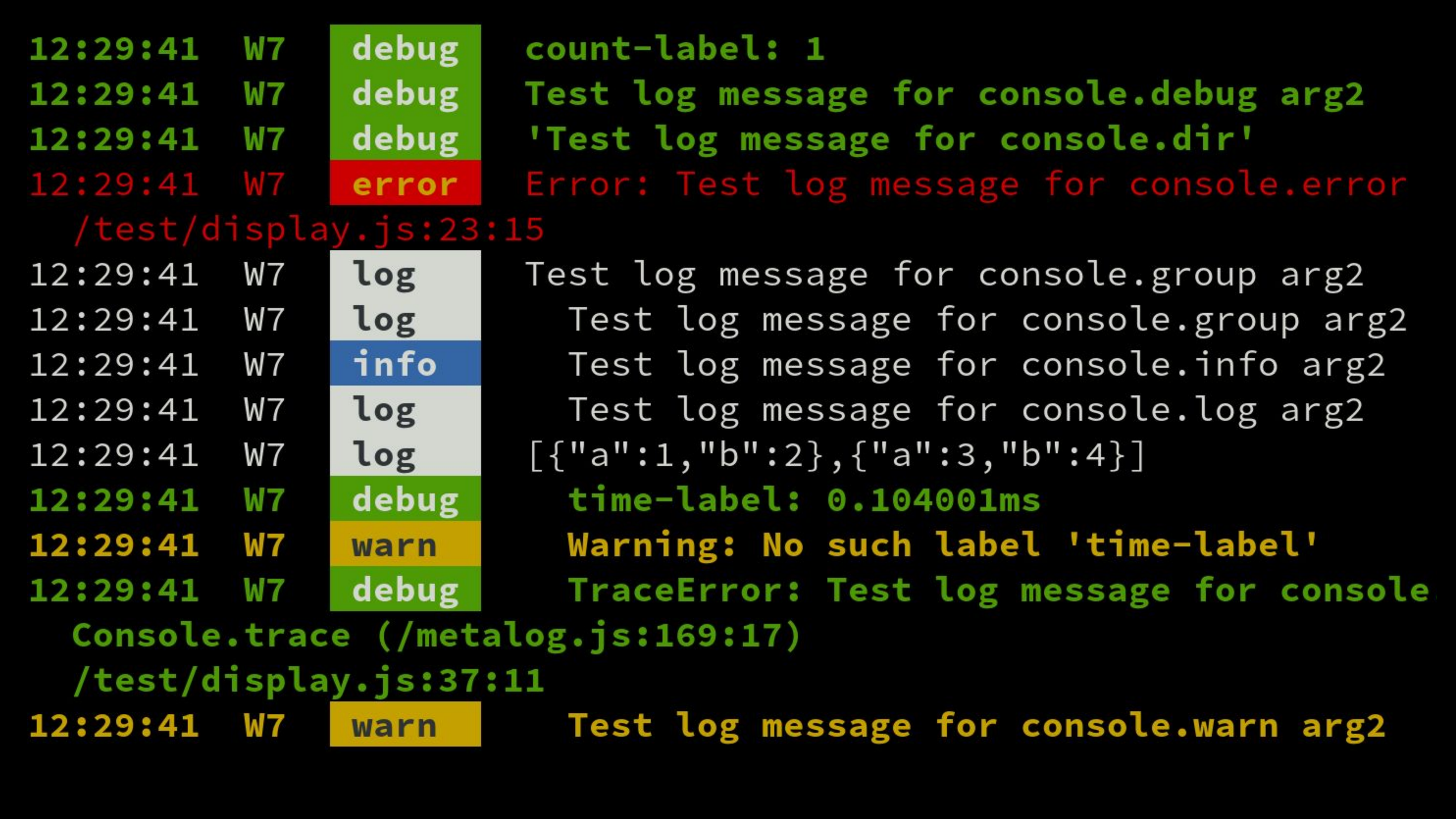
Configuration

```
// application/config/server.js
({
  host: '0.0.0.0',
  balancer: 80,
  protocol: 'https',
  ports: [8001, 8002],
  nagle: false,
  timeouts: {
    bind: 2000, start: 30000, stop: 5000,
    request: 5000, watch: 1000,
  },
  queue: { concurrency: 1000, size: 2000, timeout: 3000 },
  workers: { pool: 2 },
});
```

```
// application/config/database.js
({
  host: '127.0.0.1',
  port: 5432,
  database: 'application',
  user: 'marcus',
  password: 'marcus',
});

// application/domain/database/start.js
async () => {
  const options = { ...config.database, logger: console };
  const database = new metarhia.metasql.Database(options);
  domain.db = database;
};
```

Logging

```
// application/config/log.js
({
  keepDays: 100,
  writeInterval: 3000,
  writeBuffer: 64 * 1024,
  toFile: ['error', 'warn', 'info', 'debug', 'log'],
  toStdout: ['error', 'warn', 'info'],
});

// How to write logs: use Console interface
.clear      .assert     .count      .countReset
.debug      .dir        .error      .warn
.info       .log        .table      .trace
.time       .timeEnd    .timeLog
.group      .groupEnd   .groupCollapsed
```


12:29:41 W7 debug count-label: 1
12:29:41 W7 debug Test log message for console.debug arg2
12:29:41 W7 debug 'Test log message for console.dir'
12:29:41 W7 error Error: Test log message for console.error
/test/display.js:23:15
12:29:41 W7 log Test log message for console.group arg2
12:29:41 W7 log Test log message for console.group arg2
12:29:41 W7 info Test log message for console.info arg2
12:29:41 W7 log Test log message for console.log arg2
12:29:41 W7 log [{"a":1,"b":2},{"a":3,"b":4}]
12:29:41 W7 debug time-label: 0.104001ms
12:29:41 W7 warn Warning: No such label 'time-label'
12:29:41 W7 debug TraceError: Test log message for console
Console.trace (/metalog.js:169:17)
/test/display.js:37:11
12:29:41 W7 warn Test log message for console.warn arg2

Features

Live reload
Graceful shutdown

Auto load on start:
Dependencies,
/config, /api, /domain, /lib modules
/resources and /static
/schemas

```
node.http

await node.fs.promises.readFile('contacts.csv', 'utf8')

new metarhia.metacom.Metacom('wss://domainname.com/')

config.database

api.example.getCities({ countryId: 5 })

lib.utils.parse(data)

domain.pay.transfer(from, to, amount)

application.resources.get('fileName.ext')
```

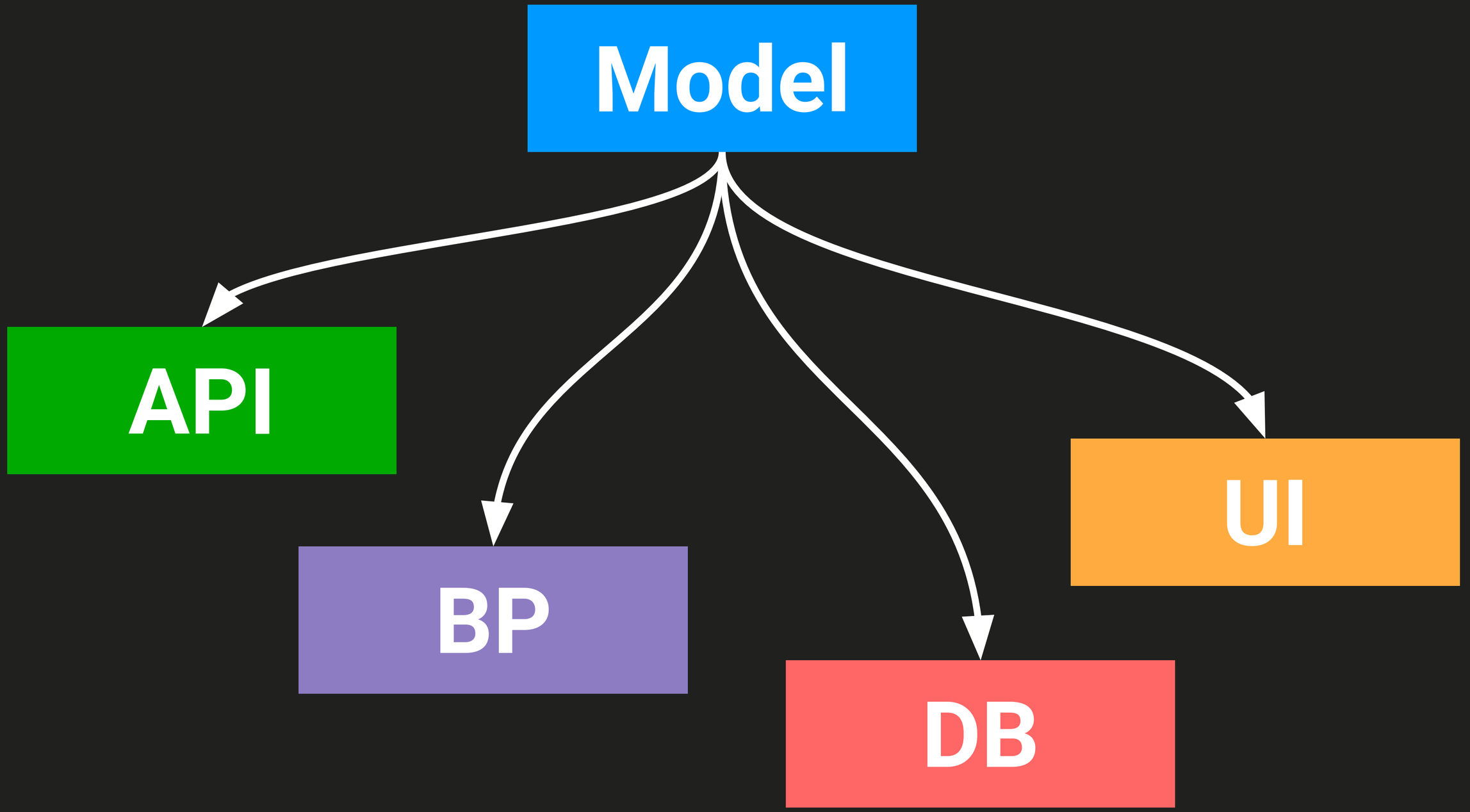
```
11:07:07  W4   info    Application started in worker 4
11:07:07  W1   debug   Start example plugin
11:07:07  W5   info    Application started in worker 5
11:07:07  W3   info    Application started in worker 3
11:07:07  W1   debug   Connect to redis
11:07:07  W1   debug   Connect to pg
11:07:07  W1   info    Application started in worker 1
11:07:07  W2   info    Application started in worker 2
11:07:09  W1   debug   Reload: /api/example.1/rooms.js
11:07:10  W1   debug   Deleted: /api/example.1/rooms.js
11:07:12  W1   debug   Reload: /api/example.1/rooms.js
```

```
# touch rooms.js
# rm rooms.js
# cp countries.js rooms.js
[marcus@localhost example.1]$
```

Dependency injection
Concurrency control
Execution timeout
Error handling
Auto routing
Plugable auth module
Persistent sessions
Built-in query builder

Roadmap

Automated db migrations
Health monitoring
Task scheduling
File streams
Distributed state management
Visualization (BPMN, ERD)
Support HTTP/2 and HTTP/3
Support pure TCP and TLS


Model
API
BP
DB
UI

Support and community

Metarhia is a Community since 2013 (8 years)
~27.000 developers
269 meetups, >200 lectures

60 collaborators, >100 contributors

multiple use cases including fintech, healthtech, interactive TV, games, and government...

Do not use for:

Web
Blogs
Sites
Content publishing

Use Metarhia for:

API
Domain logic
Interactive
IoT
Games
Enterprise
Stateful

github.com/metarhia

github.com/HowProgrammingWorks
youtube.com/TimurShemsedinov
github.com/tshemsedinov

github.com/metarhia/Example