Node.js in 2020
readiness for enterprise solutions

github.com/HowProgrammingWorks

Тимур Шемсединов

Chief Technology Architect at Metarhia
Lecturer at Kiev Polytechnic Institute

github.com/tshemsedinov

Who we are?

- Metarhia is a Community since 2013 (7 years)
  >15000 developers, we did 265 meetups and
  154 lectures, we have 5 node.js contributors

- Metarhia is a Technology stack for highload apps
  44 collaborators and >100 contributors, multiple
  use cases including interactive TV and government







四川电视节
Sichuan TV Festival


计 算 机 学 院
COMPUTER SCHOOL

THE
EVENTH
SENSE

github.com/tshemsedinov
github.com/HowProgrammingWorks/Index

https://youtube.com/TimurShemsedinov

meetup.com/HowProgrammingWorks
meetup.com/NodeUA

t.me/HowProgrammingWorks
t.me/NodeUA

Node.js in 2020

State of the platform and future

Node.js уже 10 лет: v0.0.1 — 27 мая 2019

0.10.x и 0.12.x — (2013 - 2016)
Io.js 1.x, 2.x, 3.x — (2014 - 2015)

Argon 4.x (2015 - 2018), 5.x (2015 - 2016),
Boron 6.x (2016 - 2019), 7.x (2016 - 2017),
Carbon 8.x (2017 - 2019), 9.x (2017 - 2018),
Dubnium 10.x (2018 - 2021), 11.x (2018 - 2019),
Erbium 12.x (2019 - 2022), 13.x (to June 2020)
14.x (April 2020 - April 2023), 15.x(2020-)

Node.js features

8.x V8 6.0, async/await, TurboFan and Ignition

10.x V8 6.6, HTTP/2, fs.promises, BigInt, npm 6

12.x V8 7.8, TLS 1.3, OpenSSL 1.1.1c, npm 6.10.3

js: #, static, async/await, async stack, динамическая куча, llhttp и llparser, threads, DOS в HTTP/2, startup, fs.rmdir & fs.Dir, process.resourceUsage()

Node.js features

13.x V8 7.8, npm 6.13.6, libuv 1.34.1
WASI, worker.resourceLimits, vm.Module
Source map, Advanced Serialization API

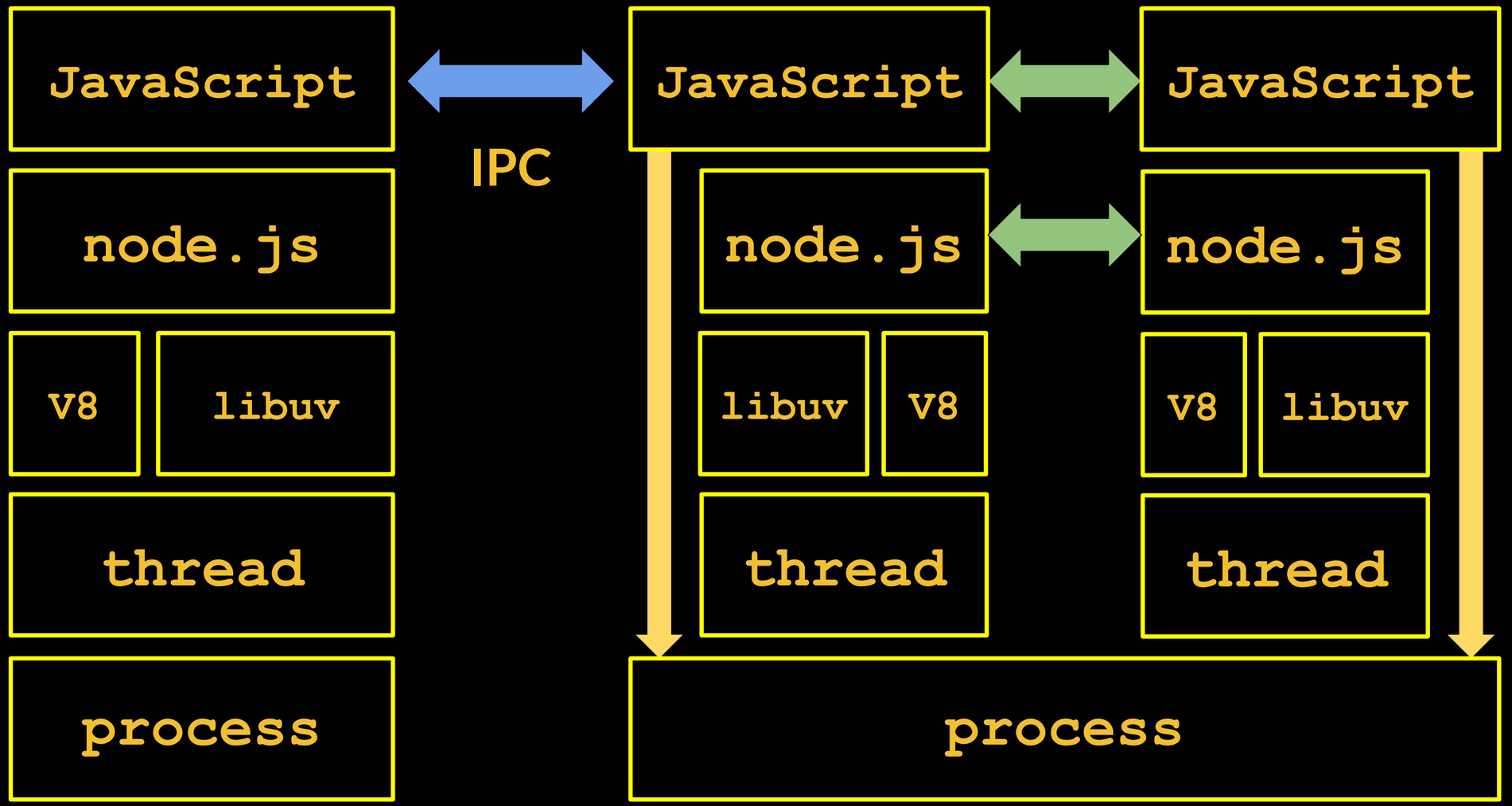
14.x Ожидания: V8 8.x
HTTP/3 (HTTP over IETF QUIC)
Внедрение промисов во все API
Web locks API

Shared memory and Atomics

Появилось в Node.js 9

- SharedArrayBuffer
- Atomics
  - add, sub, and, or, xor
  - store, load, exchange, compareExchange
  - notify, wait, ~~wake~~ (deprecated)

Threads vs Processes



Why Isolation?

- Ошибки
- Утечки памяти и других ресурсов
- Приложение: данные, соединения с БД
- Файловая система и корневой каталог
- Окружение ОС, PID, IPC
- Безопасность ОС: пользователи, группы
- Сеть: дескрипторы сокетов, порты, хосты

Execution Strategy Problems

- Недостаточная изоляция исполнения запросов к серверу друг от друга
- Один неудачный запрос может убить все параллельно исполняемые
- В асинхронной среде сложно найти и связать ошибку с запросом
- Изоляция приложений и организаций в SaaS

Execution Isolation

- VPS (виртуальная машина)
- Контейнер (Docker)
- Провесс (node)
- Поток (встроенный модуль worker_threads)
- Песочница (vm.createContext, vm.Script)
- Программная абстракция (объект или замыкание)

Execution Strategy

- 1 процесс, 1 поток JS, 1 запрос в потоке
- 1 процесс, 1 поток JS, N запросов в нем
- 1 процесс, N потоков JS, 1 запрос в каждом
- 1 процесс, N потоков JS, N запрос в каждом
- N процессов, 1 поток JS, 1 запрос в каждом
- N процессов, 1 поток, N запросов в каждом
- N процессов, N потоков, 1 запрос в каждом
- N процессов, N потоков, N запрос в каждом

Execution Strategy

- 1 процесс, 1 поток JS, 1 запрос в потоке
- 1 процесс, 1 поток JS, N запросов в нем
- 1 процесс, N потоков JS, 1 запрос в каждом
- 1 процесс, N потоков JS, N запрос в каждом
- N процессов, 1 поток JS, 1 запрос в каждом
- N процессов, 1 поток, N запросов в каждом
- N процессов, N потоков, 1 запрос в каждом
- N процессов, N потоков, N запрос в каждом

Links

Node	https://bit.ly/2lFgqSN (видео) https://node.green/ https://nodejs.org/api/worker_threads.html
JS	https://wicg.github.io/web-locks
Deno	https://github.com/denoland/deno https://youtu.be/z6JRlx5NC9E

Deno

- Безопасность:
  файловая система,сеть, окружение
- V8, TypeScript
- Rust вместо C++
- Tokio (event loop, I/O scheduler)
- Встроенный менеджер пакетов

Node.js

Готовность ноды для серьезных систем

Проблемы ноды

- Безопасность, заражения, зависимости
- Потерянные ошибки, утечки, перезапуски
- Асинхронность и стектрейс

Перспективы платформы

Questions?

ES.Next
ECMAScript 2020

ECMA Script versions

ES1	Jun 1997
ES2	Jun 1998
ES3	Dec 1999
ES5	Dec 2009
ES5.1	Jun 2011
ES6	ES2015
ES7	ES2016
ES8	ES2017
ES9	ES2018
ES10	ES2019
ES11	ES2020
ES.Next	

Links

ES2020	https://tc39.es/ecma262/
Node Green	https://node.green/
Proposals	https://github.com/tc39/proposals
Can I use	https://caniuse.com/

Array

Array.prototype.includes(value[, fromIndex])

Array.prototype.flat([depth])
Array.prototype.flatMap(callback[, thisArg])

Array.prototype.sort([compareFunction])
    QuickSort to TimSort

Object

Object.values(object)
Object.keys(object)
Object.entries(object)
Object.fromEntries(object)

String

String.prototype.padStart(targetLength [, padString])
String.prototype.padEnd(targetLength [, padString])
String.prototype.trimStart()
String.prototype.trimEnd()

Operators

Rest

```
const f = (a, b, ...array) => {};
const g = ({ a, b, ...array }) => {};
const { name, ...rest } = obj;
```

Spread

```
f(a, b, ...array);
const obj2 = { name, ...obj1 };
const clone = { ...obj };
```

Operators

Exponentiation

```
Math.pow(x, y)    x ** y    x **= y    x = x ** y
```

Optional chaining

```
const spqr = {
  emperor: { name: 'Marcus' }
};
console.log(spqr.emperor?.name);
console.log(spqr.president?.name);
```

Operators

Exponentiation

```
Math.pow(x, y)    x ** y    x **= y    x = x ** y
```

Optional chaining (still waiting v8 8.x in Node.js)

```
const spqr = {
  emperor: { name: 'Marcus' }
};
console.log(spqr.emperor?.name);
console.log(spqr.president?.name);
```

Asynchronous function: async/await

```
const fn = async (a, b, c) => {
  // do something
  await callSomething();
  // do something
  return aValue;
};
```

Trailing commas

```
const fn = (arg1, arg2, arg3,) => {
  console.log({
    arg1,
    arg2,
    arg3,
  });
};
fn(...['val1', 'val2', ], 'val3',);
```

Asynchronous iterable contract

Symbol.iterator

```
iterable[Symbol.iterator]()
```

Symbol.asyncIterator

```
asyncIterable[Symbol.asyncIterator]()
```

Try...catch

```
try {
  throw new Error('message');
} catch {
  console.log('no arguments catched');
}
```

Function

```
((a, b) => {
    const c = a + b; // hello there
    return c;
}).toString()
"(a, b) => {
    const c = a + b; // hello there
    return c;
}"
```

Symbol

```
const sym = Symbol('description');

console.log(sym);
// Symbol(description);

console.log(sym.description);
// description
```

Promise.finally

```
new Promise(executor)
  .then(onFulfilled[, onRejected])
  .catch(onRejected)
  .finally(onFinally);
```

Promise.allSettled

```
const p1 = Promise.resolve('p1');
const p2 = new Promise((resolve, reject) => {
  setTimeout(resolve, 1000, 'p2');
});
const p3 = new Promise((resolve, reject) => {
 setTimeout(reject, 100, 'p3');
});

Promise.all([p1, p2, p3]).then(values => {
 console.log(values);
});
```

Promise.allSettled

```
Promise.all([p1, p2, p3]).then(values => {
 console.log(values);
});
```

```
node:26549) UnhandledPromiseRejectionWarning: p3
(node:26549) UnhandledPromiseRejectionWarning: Unhandled promise
rejection. This error originated either by throwing inside of an
async function without a catch block, or by rejecting a promise
which was not handled with .catch(). (rejection id: 1)
(node:26549) [DEP0018] DeprecationWarning: Unhandled promise
rejections are deprecated. In the future, promise rejections that
are not handled will terminate the Node.js process with
a non-zero exit code.
```

Promise.allSettled

```
const p1 = Promise.resolve('p1');
const p2 = new Promise((resolve, reject) => {
  setTimeout(resolve, 1000, 'p2');
});
const p3 = new Promise((resolve, reject) => {
 setTimeout(reject, 100, 'p3');
});

Promise.allSettled([p1, p2, p3]).then(values => {
 console.log(values);
});
```

Promise.allSettled

```
Promise.allSettled([p1, p2, p3]).then(values => {
 console.log(values);
});

[
  { status: 'fulfilled', value: 'p1' },
  { status: 'fulfilled', value: 'p2' },
  { status: 'rejected', reason: 'p3' }
]
```

More features

Atomics
SheredArrayBuffer
Set, Map, WeakSet, WeakMap
globalThis
Private fields
Static fields

Questions?

How Race Conditions in single threaded JS are possible?

Concurrency Problems

- Race condition
- Deadlock
- Livelock
- Resource starvation
- Resource leaks

Solutions

javascruptissinglethreaded

Solutions

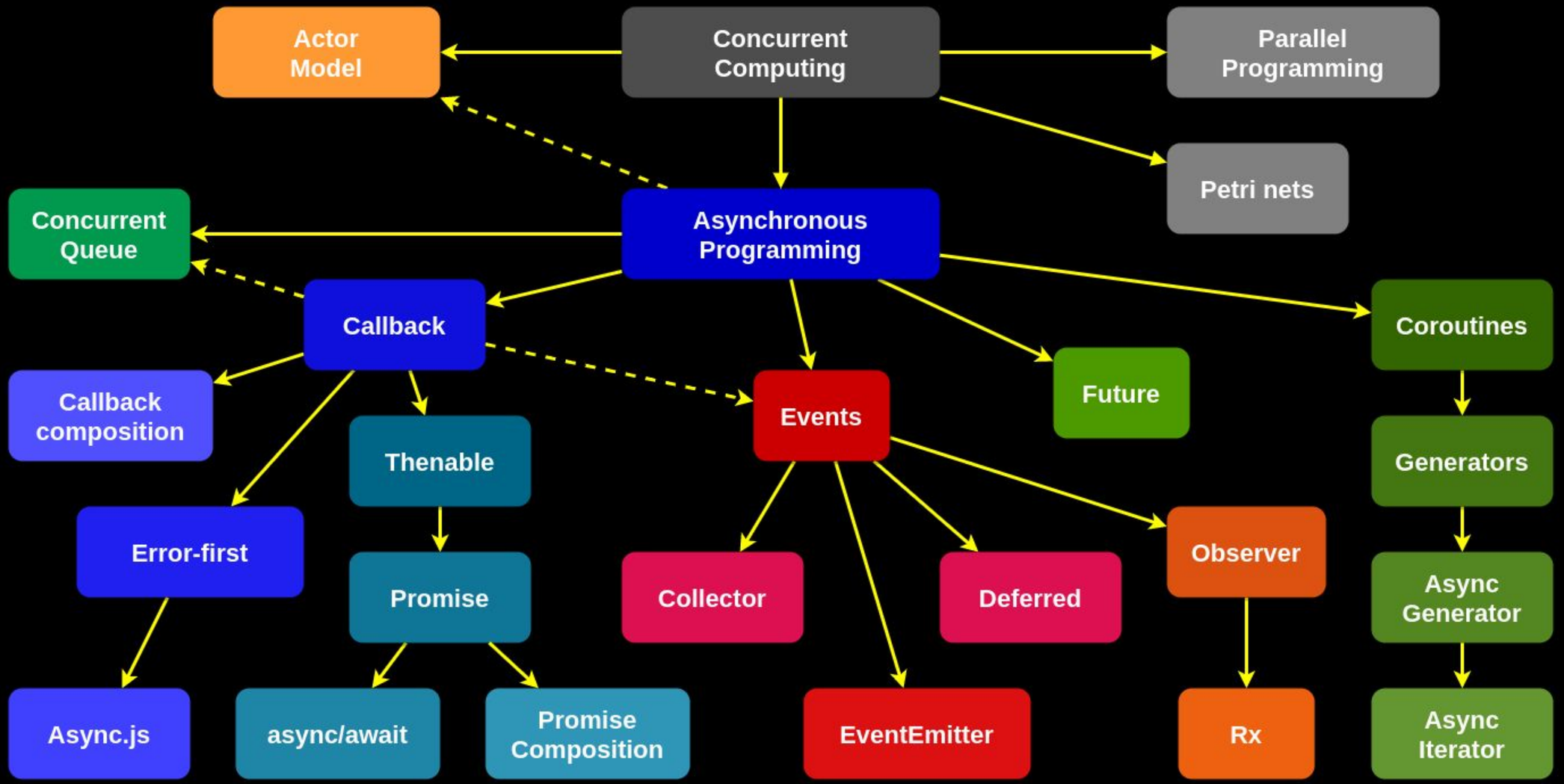
nodejsissinglethreaded

Solutions

Promises

async/await

Concurrent Computing



Race Condition

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  async move(dx, dy) {
    this.x = await add(this.x, dx);
    this.y = await add(this.y, dy);
  }
}
```

Race Condition

```
const random = (min, max) => Math
  .floor(Math.random() * (max - min + 1)) + min;

const add = (x, dx) => new Promise(resolve => {
  setTimeout(() => {
    resolve(x + dx);
  }, random(20, 100));
});
```

Race Condition

```
const p1 = new Point(10, 10);
console.log(p1);

p1.move(5, 5);
p1.move(6, 6);
p1.move(7, 7);
p1.move(8, 8);

setTimeout(() => {
  console.log(p1);
}, 1000);
```

Race Condition

Initial

```
Point { x: 10, y: 10 }
```

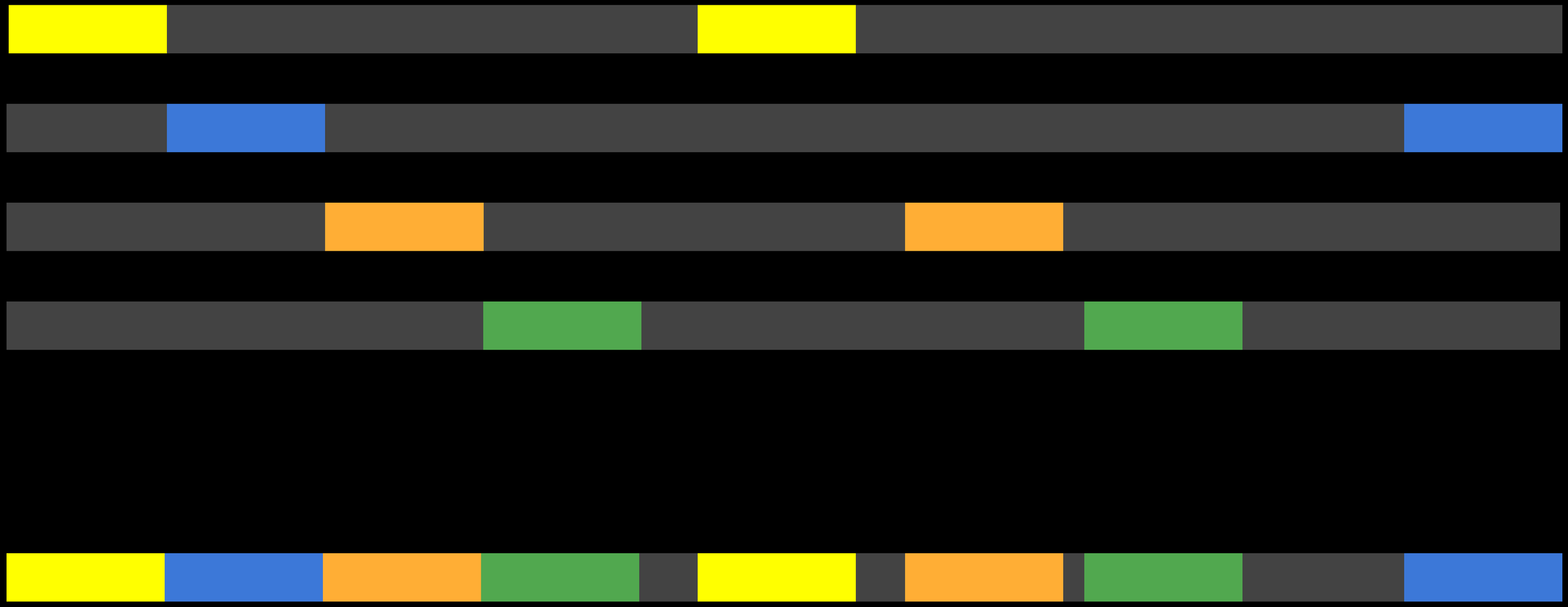
Expected

```
Point { x: 36, y: 36 }
```

Actual

```
Point { x: 18, y: 25 }
```

Race Condition



Possible Solutions

- Synchronization
- Resource locking
- Special control flow organization
- Queuing theory
- Actor model
- Use DBMS transactions

Synchronization Primitives

Semaphore
Binary semaphore
Counting semaphore
Condition variable
Spinlock

Mutex
Timed mutex
Shared mutex
Recursive mutex
Monitor
Barrier

Semaphore

```
class Semaphore {
  constructor()
  enter(callback)
  leave()
}
semaphore.enter(() => {
  // do something
  semaphore.leave();
});
```

```
class Semaphore {
  constructor()
  async enter()
  leave()
}
await semaphore.enter();
// do something
semaphore.leave();
```

github.com/HowProgrammingWorks/Semaphore

Mutex

```
class Mutex {
  constructor()
  async enter()
  leave()
}

await mutex.enter();
// do something with shared resources
mutex.leave();
```

https://github.com/HowProgrammingWorks/Mutex

Resource Locking

```
class Lock {
  constructor() {
    this.active = false;
    this.queue = [];
  }

  leave() {
    if (!this.active) return;
    this.active = false;
    const next = this.queue.pop();
    if (next) next();
  }
}
```

```
enter() {
  return new Promise(resolve => {
    const start = () => {
      this.active = true;
      resolve();
    };
    if (!this.active) {
      start();
      return;
    }
    this.queue.push(start);
  });
}
```

Resource Locking

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
    this.lock = new Lock();
  }

  async move(dx, dy) {
    await this.lock.enter();
    this.x = await add(this.x, dx);
    this.y = await add(this.y, dy);
    this.lock.leave();
  }
}
```

Resource Locking

```
const p1 = new Point(10, 10);
console.log(p1);

p1.move(5, 5);
p1.move(6, 6);
p1.move(7, 7);
p1.move(8, 8);

setTimeout(() => {
  console.log(p1);
}, 1000);
```

Resource Locking

Initial

```
Point { x: 10, y: 10 }
```

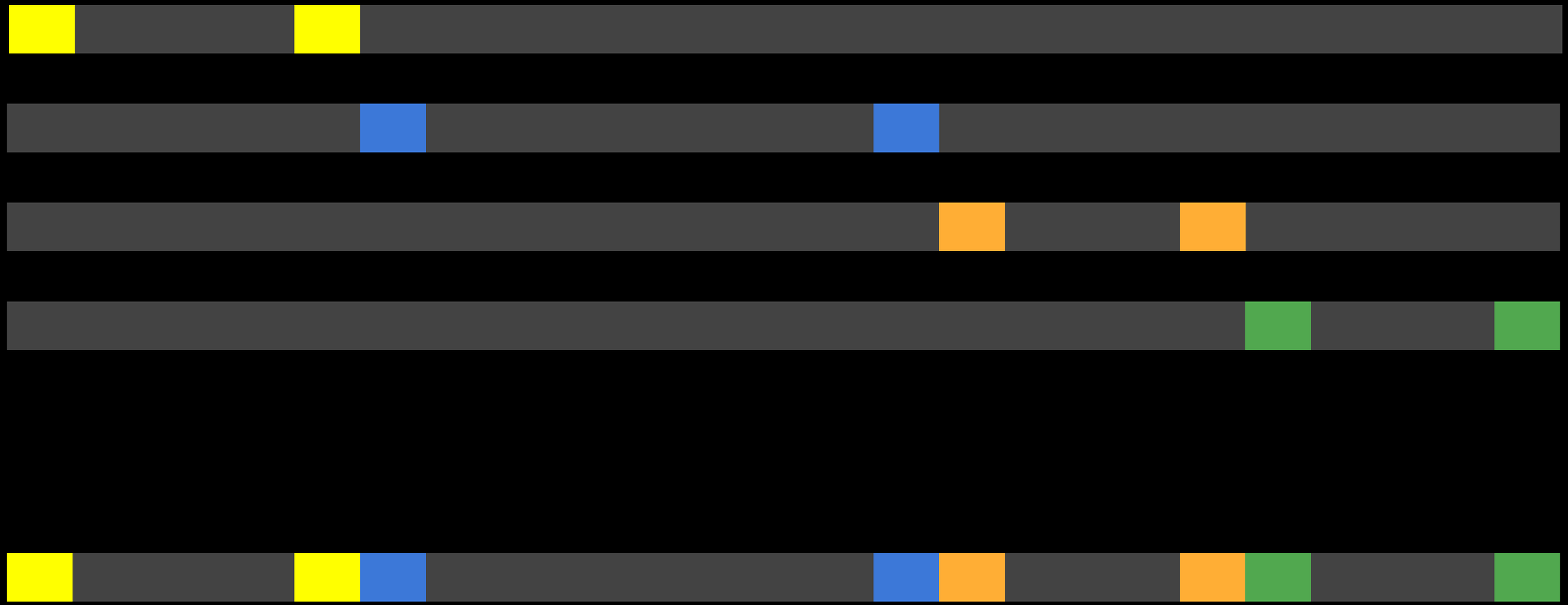
Expected

```
Point { x: 36, y: 36 }
```

Actual

```
Point { x: 36, y: 36 }
```

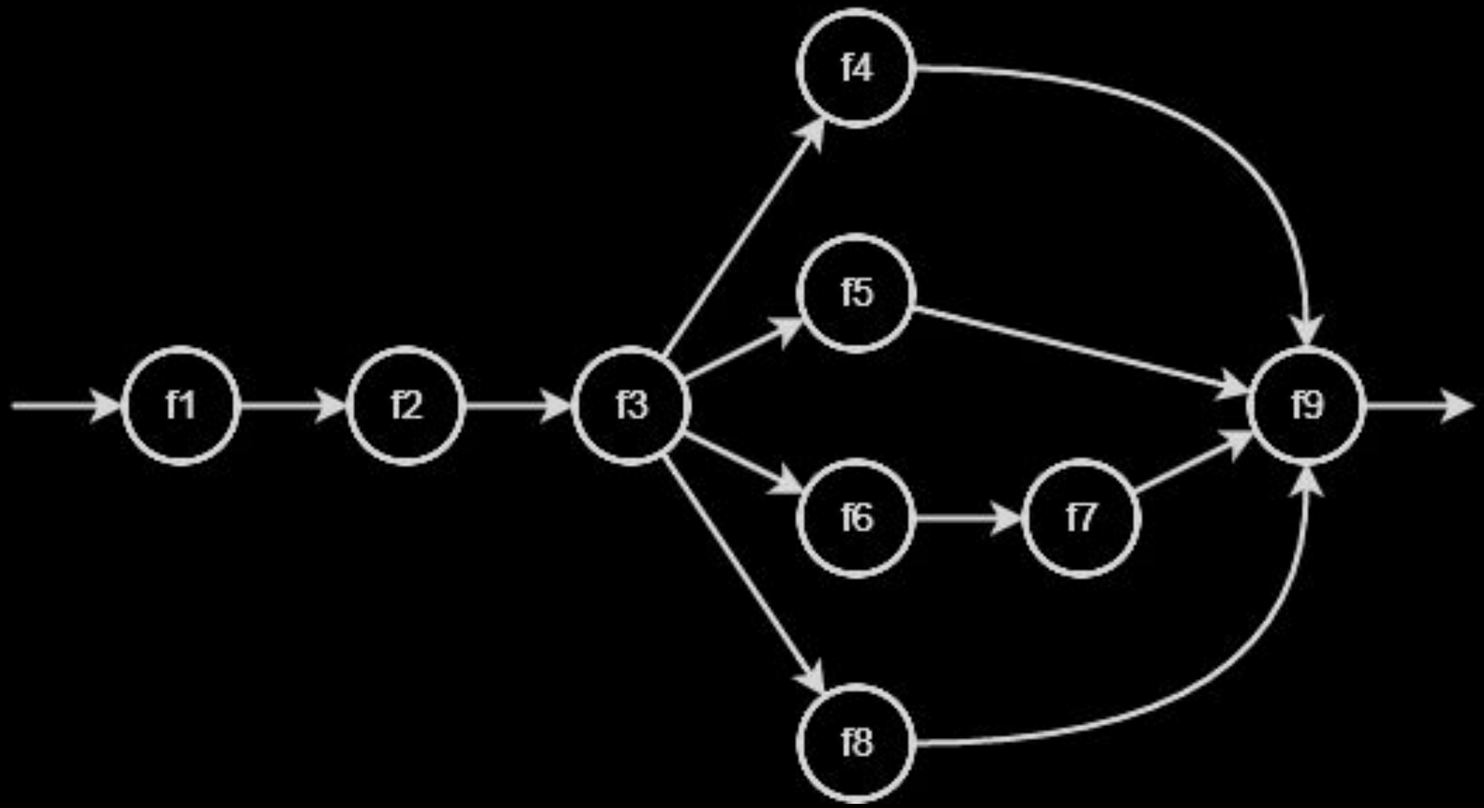
Resource Locking



Asynchronous Res Locks

Flow Commutation

like in electronics




```
const fx = metasync(
  [f1, f2, f3, [[f4, f5, [f6, f7], f8]], f9]
);
```

Real-life Example

Warehouse API

- Check balances
- Ship goods
- Lock balances

github.com/HowProgrammingWorks/RaceCondition

Web Locks API

```
locks.request('resource', opt, async lock => {
  if (lock) {
    // critical section for `resource`
    // will be released after return
  }
});
```

https://wicg.github.io/web-locks/

Web Locks: await

```
(async () => {
  await something();
  await locks.request('resource', async lock => {
    // critical section for `resource`
  });
  await somethingElse();
})();
```

Web Locks: Promise

```
locks.request('resource', lock => new Promise(
  (resolve, reject) => {
    // you can store or pass
    // resolve and reject here
  }
));
```

Web Locks: Thenable

```
locks.request('resource', lock => ({
  then((resolve, reject) => {
    // critical section for `resource`
    // you can call resolve and reject here
  })
}));
```

Web Locks: Abort

```
const controller = new AbortController();
setTimeout(() => controller.abort(), 200);

const { signal } = controller;

locks.request('resource', { signal }, async lock => {
  // lock is held
}).catch(err => {
  // err is AbortError
});
```

Web Locks for Node.js

github.com/nodejs/node/issues/22702
Open

github.com/nodejs/node/pull/22719
Closed

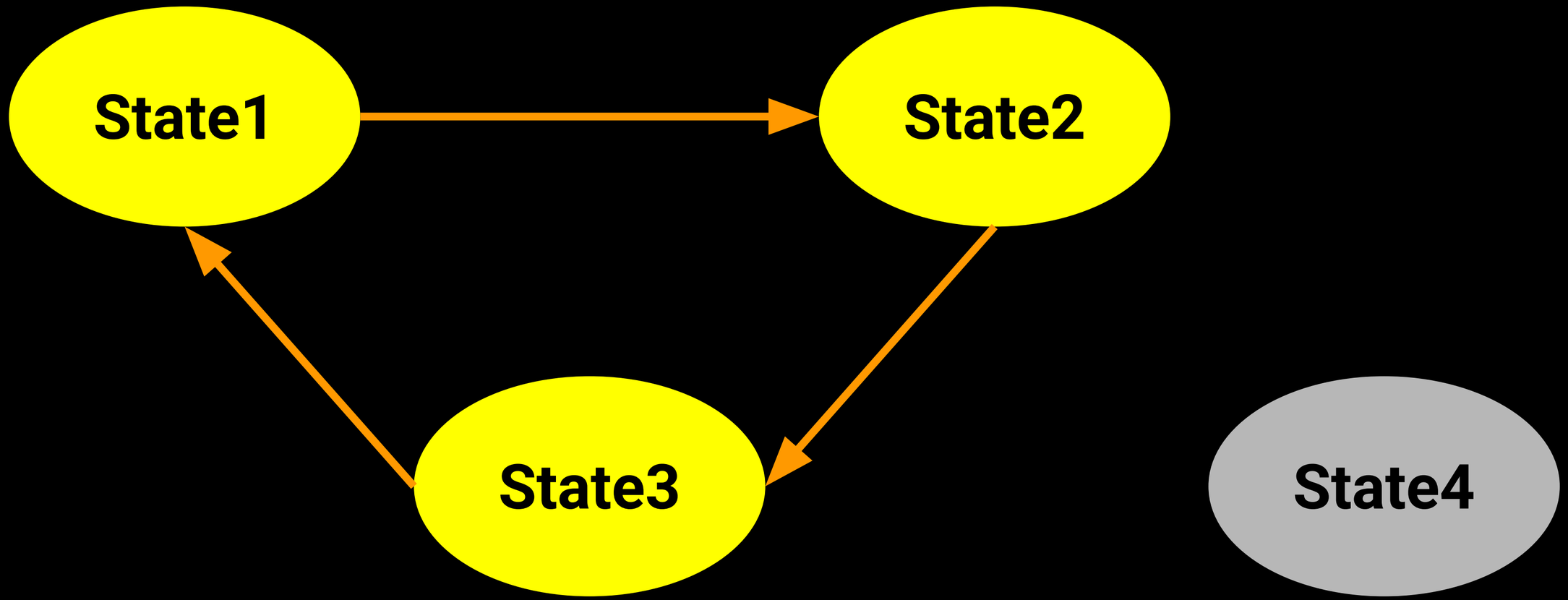
Safe data structures

- Low-level structures
  e.g. Register, Counter, Buffer, Array, Lists, etc.
- Abstract structures
  e.g. Queue, Graph, Polyline, etc.
- Subject-domain classes
  e.g. Sensors, Payment, Biometric data, etc.
- Resources and handles
  e.g. Sockets, Connections, Streams, etc.

Deadlock

```
(async () => {
  await locks.request('A', async lock => {
    await locks.request('B', async lock => {
    });
  });
})(); (async () => {
  await locks.request('B', async lock => {
    await locks.request('A', async lock => {
    });
  });
})();
```

Livelock



Alternative Solutions

- Thread safe data structures
- Lock-free data structures
- Wait-free algorithms
- Conflict-free data structures

Links

habr.com/ru/post/452974/
github.com/HowProgrammingWorks/RaceCondition
github.com/HowProgrammingWorks/Semaphore
github.com/HowProgrammingWorks/Mutex
github.com/metarhia/metasync
wicg.github.io/web-locks

Questions?

GRASP, SOLID, GoF

Как это может быть связано с JavaScript?

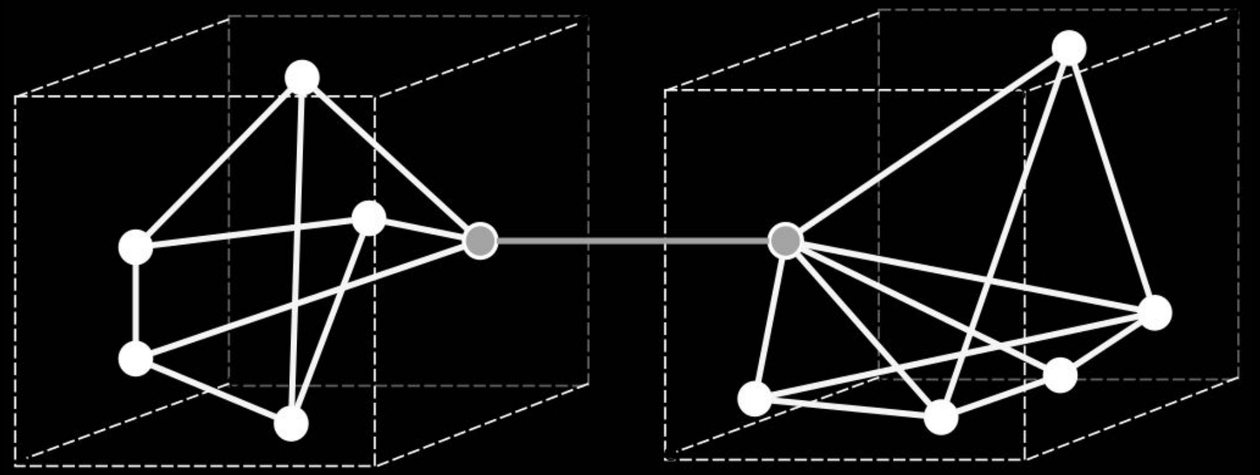
GRASP

General responsibility assignment software patterns (распределение ответственности)

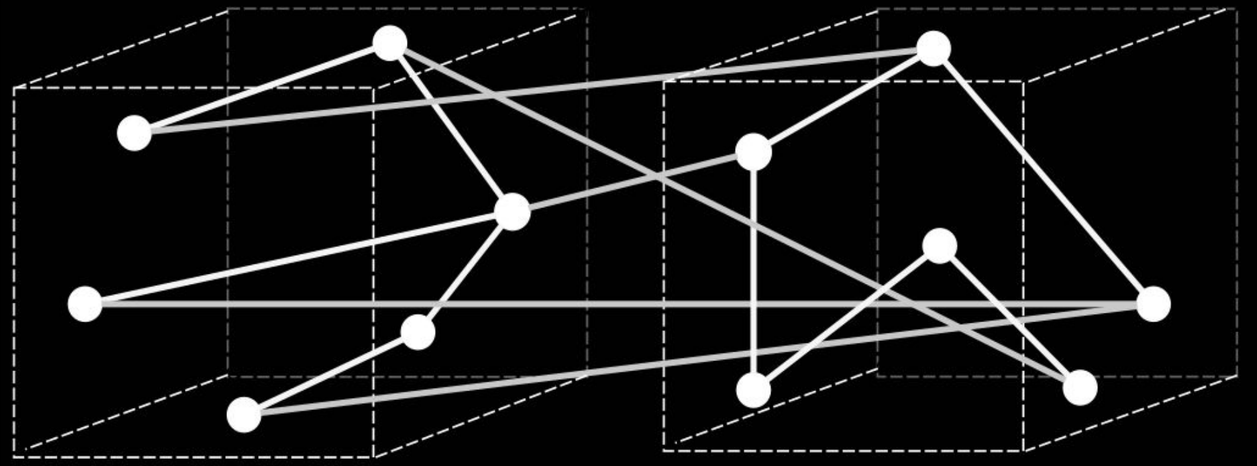
Книга “Применение UML и шаблонов проектирования” // Крэг Ларман

GRASP: Coupling and cohesion

Cohesion (связность) внутри модуля или программного компонента



Coupling (зацепление) между модулями



GRASP:

General responsibility assignment software patterns (распределение ответственности)

Low Coupling
Information Expert
Controller
Pure Fabrication
Protected Variations

High Cohesion
Creator
Polymorphism
Indirection

SOLID: задачи

Майкл Фэзерс (Michael Feathers)
Роберт Мартин (Robert Martin, Uncle Bob)

Что они дают:

- Облегчение модификации и расширения
- Улучшение владения кодом и TTM
- Способность быстро понимать друг друга

SOLID: 5 принципов

- The Single Responsibility Principle
- The Open Closed Principle
- The Liskov Substitution Principle
- The Interface Segregation Principle
- The Dependency Inversion Principle
  (не путать с dependency injection и inversion of control)

Шаблоны GoF

Gang of Four (GoF): Эрих Гамм, Ричард Хелм, Ральф Джонсон, Джон Влиссидес

Design Patterns — Elements of Reusable Object-Oriented Software (23 шаблона)

Классификация шаблонов

Порождающие: абстрактная фабрика, строитель, фабричный метод, пул, синглтон...
Структурные: адаптер, мост, компоновщик, декоратор или обертка, фасад, прокси...
Поведенческие: цепочка обязанностей, команда, обсервер, итератор, стратегия...
Коммуникационные: cqs, cqrs, event sourcing...

Questions?

Node.js Security in 2020

Node.js Security Aspects

- Concurrency model for I/O
- Common path traversal vulnerability
- SQL injection, XSRF, XSS etc.
- Resource leaks (memory, handlers, etc.)
- Passwords should be stored as hash with salt
- Load control (DoS/DDoS)

Security issues

- Dependencies: see your node_modules
  - Unreliable dependencies
  - Malicious modules from NPM
- Sandbox escaping (vm)
- Buffer vulnerabilities
- Regular expressions

Tools Node.js

- Linter may help github.com/nodesecurity/eslint-plugin-security
- We have npm audit and it may even fix multiple problems automatically npm audit fix
- Special tools: ~~nsp~~, snyk
- Githib have built-in Security Alert

SQLI (SQL Injection)

Hello! See what they say about you:
http://bank-web-site.com/accounts?
name='marcus'%20OR%201=1%20--%20

https://bit.ly/2XZpJMt

“SELECT * from Accounts where name=” + name

XSRF (Cross-Site Request Forgery)

They can send you:

Hello! See what they say about you:
http://payment-system.com/api/transfer?
amount=1000&destination=card-number

XSS (Cross-Site Scripting)

They can send you:

Hello! See what they say about you:
http://control-panel.com/help.php?q=
%3Cscript%3Ealert('Hello');%3C/script%3E

CSP (Content Security Policy)

Browser have built-in layer to create security policy to solve XSS problem

https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

Path traversal

```
const serveFile = fileName => {
  const filePath = path.join(STATIC_PATH, fileName);
  return fs.createReadStream(filePath);
};

http.createServer((req, res) => {
  const url = decodeURI(req.url);
  serveFile(url).pipe(res);
}).listen(8000);
```

```
curl -v http://127.0.0.1:8000/%2e%2e/1-traversal.js
```

Path traversal fixed

```
const serveFile = fileName => {
  const filePath = path.join(STATIC_PATH, fileName);
  if (!filePath.startsWith(STATIC_PATH)) {
    throw new Error(`Access denied: ${name}`);
  }
  return fs.createReadStream(filePath);
};
http.createServer((req, res) => {
  const url = decodeURI(req.url);
  serveFile(url).pipe(res);
}).listen(8000);
```

What’s next:

- httponly cookies
  https://www.owasp.org/index.php/HttpOnly
- HTTP Headers:
  - X-XSS-Protection
  - X-Frame-Options
  - X-Content-Type-Options
    etc.

OWASP (Open Web App. Security Project)

See this site:

https://owasp.org/

Questions?

Node.js

Patterns and Antipatterns

Node.js antipatterns

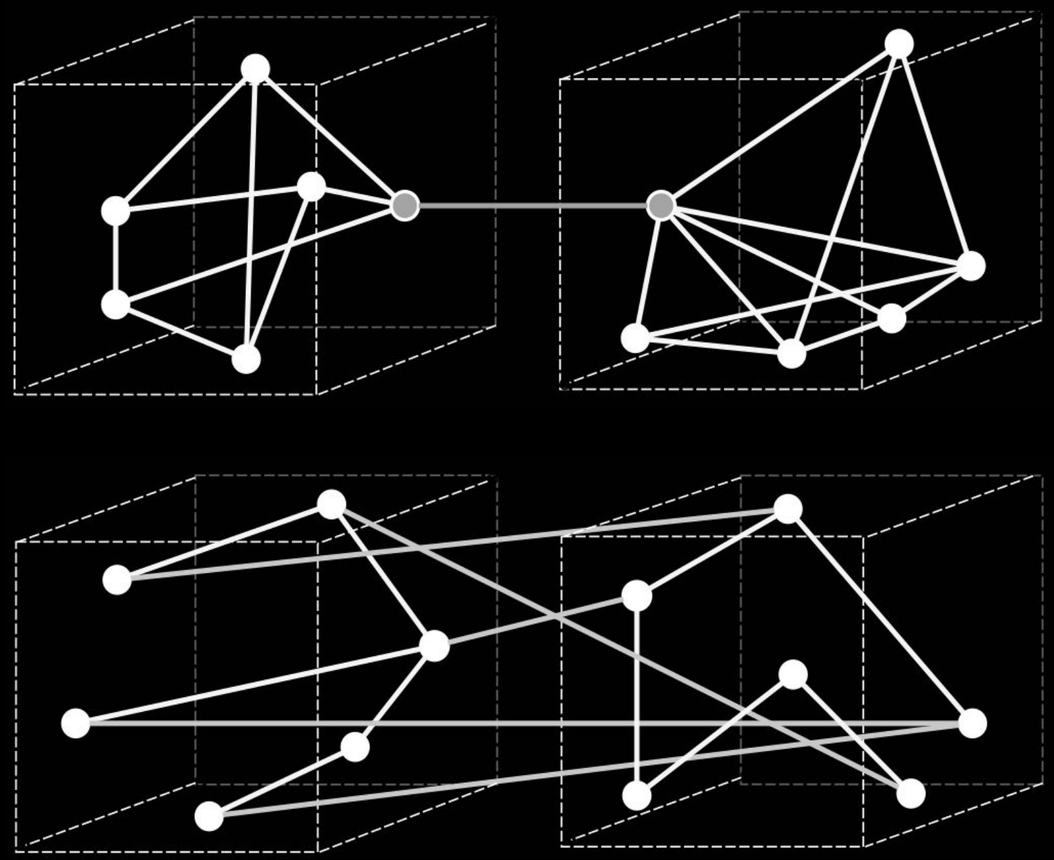
- Structure and arch.
- Initialization
- Dependency issues
- Application state
- Middlewares
- Context isolation
- Security issues
- Asynchronity issues
- Blocking operations
- Memory leaks
- Databases and ORM
- Error handling

No layers, everything mixed

- Configuration and Dependency management
- Network protocols related code (http, tcp, tls...)
- Request parsing, Cookies, Sessions
- Logging, Routing, Business-logic
- I/O: fs, Database queries
- Generating responses and error generation
- Templating, etc.

Middlewares

Middlewares is an extremely bad idea for low coupling and high cohesion



Middlewares changes:

- Socket state
- Db connection state
- Server state

Don’t create global state

```
let groupName;

app.use((req, res, next) => {
  groupName = 'idiots'; next();
});

app.get('/user', (req, res) => {
  if (groupName === 'idiots') {
    res.end('I know you!');
  }
});
```

Don’t mixin to req, res, ctx

```
app.use((req, res, next) => {
  res.groupName = 'idiots';
  next();
});

app.get('/user', (req, res) => {
  if (res.groupName === 'idiots') {
    res.end('I know you!');
  }
});
```

Special place for the state: res.locals

```
app.use((req, res, next) => {
  res.locals.groupName = 'idiots';
  next();
});

app.get('/user', (req, res) => {
  if (res.locals.groupName === 'idiots') {
    res.end('I know you!');
  }
});
```

Don’t mixin methods

```
app.get('/user/:id', (req, res, next) => {
  req.auth = (login, password) => { /* auth */ };
  next();
});

app.get('/user/:id', (req, res) => {
  if (req.auth(req.params.id, '111')) {
    res.end('I know you!');
  }
});
```

Don’t require in middleware / handler

```
app.get((req, res, next) => {
  req.db = new require('pg').Client();
  req.db.connect();
  next();
});

app.get('/user/:id', (req, res) => {
  req.db.query('SELECT * from USERS', (e, r) => {
    ...
  });
});
```

Don’t connect DB from handlers

```
app.get((req, res, next) => {
  req.db = new Pool(config);
  next();
});

app.get('/user/:id', (req, res) => {
  req.db.query('SELECT * from USERS', (e, r) => {
  });
});
```

Connection leaks is easy

```
const db = new Pool(config);

app.get('/user/:id', (req, res) => {
  req.db.query('SELECT * from USERS', (err, r) => {
    if (err) throw err;
    // Prepare data to reply client
  });
});
```

Don’t use blocking operations

- Sync calls like fs.readFileSync
- Console output like console.log
- Remember that require is synchronous
- Long loops (including for..of and for await)
- Serialization: JSON.parse, JSON.stringify
- Iteration: loops, Array.prototype.map, etc.
- CPU-intensive: zlib, crypto

Loop: for await of is blocking

```
(async () => {
  let ticks = 0;
  const timer = setInterval(() => ticks++, 10);
  const numbers = new Array(1000000).fill(1);
  let i = 0;
  for await (const number of numbers) i++;
  clearInterval(timer);
  console.dir({ i, ticks });
})();

// { i: 1000, ticks: 0 }
```

AsyncArray (short version)

```
class AsyncArray extends Array {
  [Symbol.asyncIterator]() {
    let i = 0;
    return {
      next: () => new Promise(resolve => {
        setTimeout(() => resolve({
          value: this[i], done: i++ === this.length
        }), 0);
      })
    };
  }
}  // github.com/HowProgrammingWorks/NonBlocking
```

Loop: for await of + AsyncArray

```
(async () => {
  let ticks = 0;
  const timer = setInterval(() => ticks++, 10);
  const numbers = new AsyncArray(10000000).fill(1);
  let i = 0;
  for await (const number of numbers) i++;
  clearInterval(timer);
  console.dir({ i, ticks });
})();
```

```
// { i: 10000, ticks: 1163 }
```

https://github.com/HowProgrammingWorks/NonBlocking

Memory leaks

- References
  - Global variables
  - Mixins to built-in Classes
  - Singletons, Caches
- Closures / Function contexts
  - Recursive closures
  - Require in the middle of code
  - Functions in loops

Memory leaks

- OS and Language Objects
  - Descriptors: files, sockets...
  - Timers: setTimeout, setInterval
- Events / Subscription / Promises
  - EventEmitter
  - Callbacks, Not resolved promises

Links

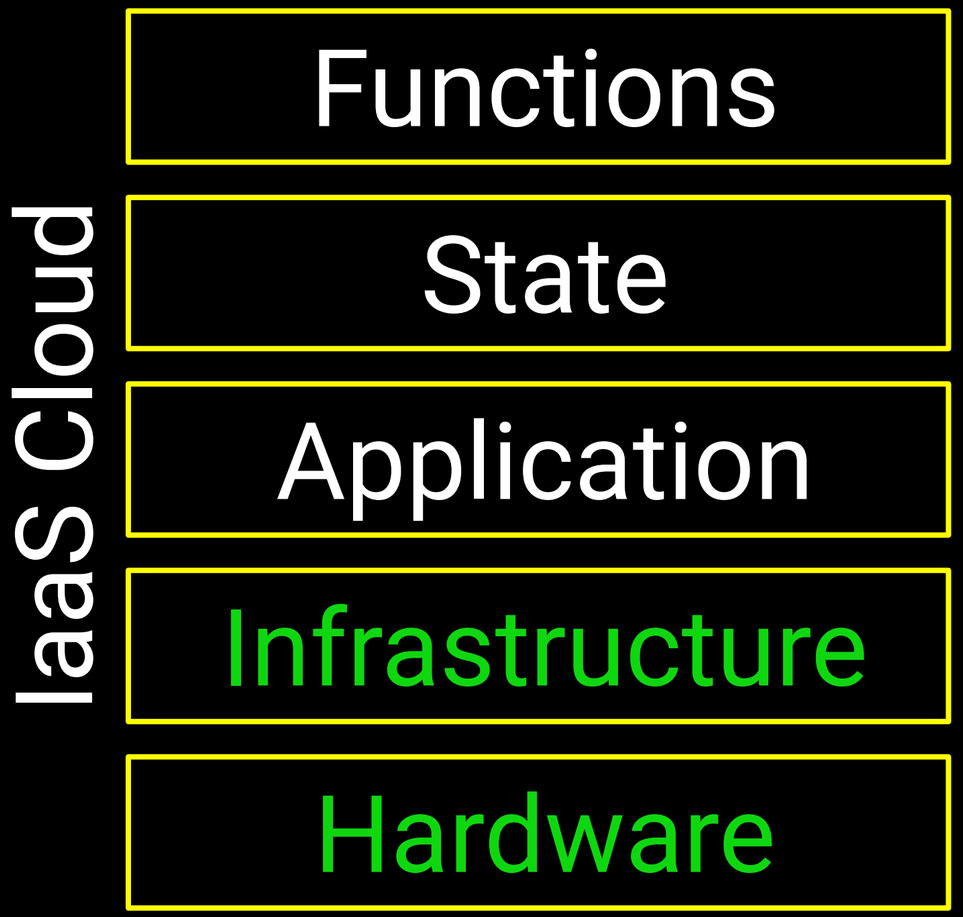
github.com/HowProgrammingWorks/AbstractionLayers

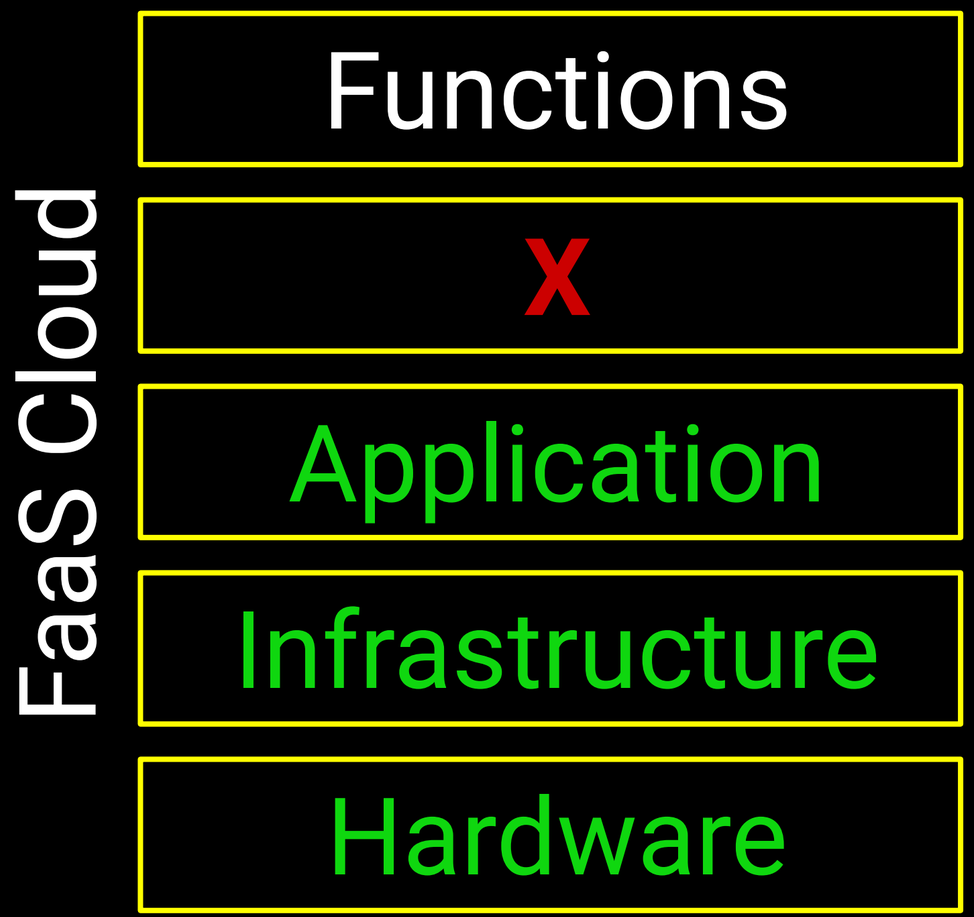
github.com/HowProgrammingWorks/MemoryLeaks

Questions?

FaaS
Serverless Clouds
and Node.js

Infrastructure Clouds vs App Clouds





Infrastructure Types

	Infra	Runtime	Storage
• Bare metal	none	stateful	stateful
• Shared hosting	slice	stateless	stateful
• Virtualization	slice	stateful	stateful
• Containerization	slice	stateful	stateful
• Clusterization	join	stateful	stateless
• Serverless	join	stateless	stateless

Infrastructure Aspects

	Sec	Lock	TTM
• Bare metal	++++	no	+
• Shared hosting	+	tech	++
• Virtualization	+++	no	+++
• Containerization	+++	tech	+++
• Clusterization	+++	tech	++++
• Serverless	++	vendor	+++++

Serverless Benefits

- Service price? (evangelists told us...)
- Efficiency: Performance? Speed? Latency?
- Easy to test, deploy, maintain?
- Security? Reliability? Flexibility? Quality?
- Quick development?
- Reduces development cost?
- Scalability?

What we pay for?

We pay for:

- lack of available professionals
- lack of competencies
- lack of available technologies
- lack of funding for our projects
- lack of time

What can be automated?

Yes

- Infrastructure
- Monitoring
- Networking
- Integration (CI)
- Deployment (CD)

No

- Scalability, sharding
- Performance
- Quality
- Security
- Interactivity

Cost Optimization Cases

- Small services, sometimes cold
  Can reduce cost x10 (great: $10 to $1)

- Highload >100k online, always warm
  Single bare metal can hold load
  Try to calculate serverless cost...

Serverless Disadvantages

- High resource consumption
- Stateless nature and no application integrity
- Interactivity issue (separate solution needed)
- Development and debug issues
- Deploy and maintain issues
- Vendor lock, not open source
- Where is no promised simplicity

Middleware Madness

```
router.get('/user/:id', (req, res, next) => {
  const id = parseInt(req.params.id);
  const query = 'SELECT * FROM users WHERE id = $1';
  pool.query(query, [id], (err, data) => {
    if (err) throw err;
    res.status(200).json(data.rows);
    next();
  });
});
```

Code Structure and Patterns

```
exports.handler = (event, context, callback) => {
  const { Client } = require('pg');
  const client = new Client();
  client.connect();
  const id = parseInt(event.pathParameters.id);
  const query = 'SELECT * FROM users WHERE id = $1';
  client.query(query, [id], (err, data) => {
    callback(err, { statusCode: 200,
      body: JSON.stringify(data.rows)});
  });
};
```

What do we want?

```
async (arg1, arg2, arg3) => {
  const [data1, data2] = await Promise.all(
    [getData(arg1), getData(arg2)]
  );
  const data3 = await getData(arg3);
  if (!data3) throw new Error('Message');
  return await processData(data1, data2, data3);
}
```

What do we want?

```
async (arg1, arg2, arg3) => {
  const data1 = await getData(arg1);
  if (!data1) throw new Error('Message');
  const [data2, data3] = await Promise.all(
    [getData(arg2), getData(arg3)]
  );
  return await processData(data1, data2, data3);
}
```

Equivalent example

```
id => application
  .database
  .select('users')
  .where({ id });
```

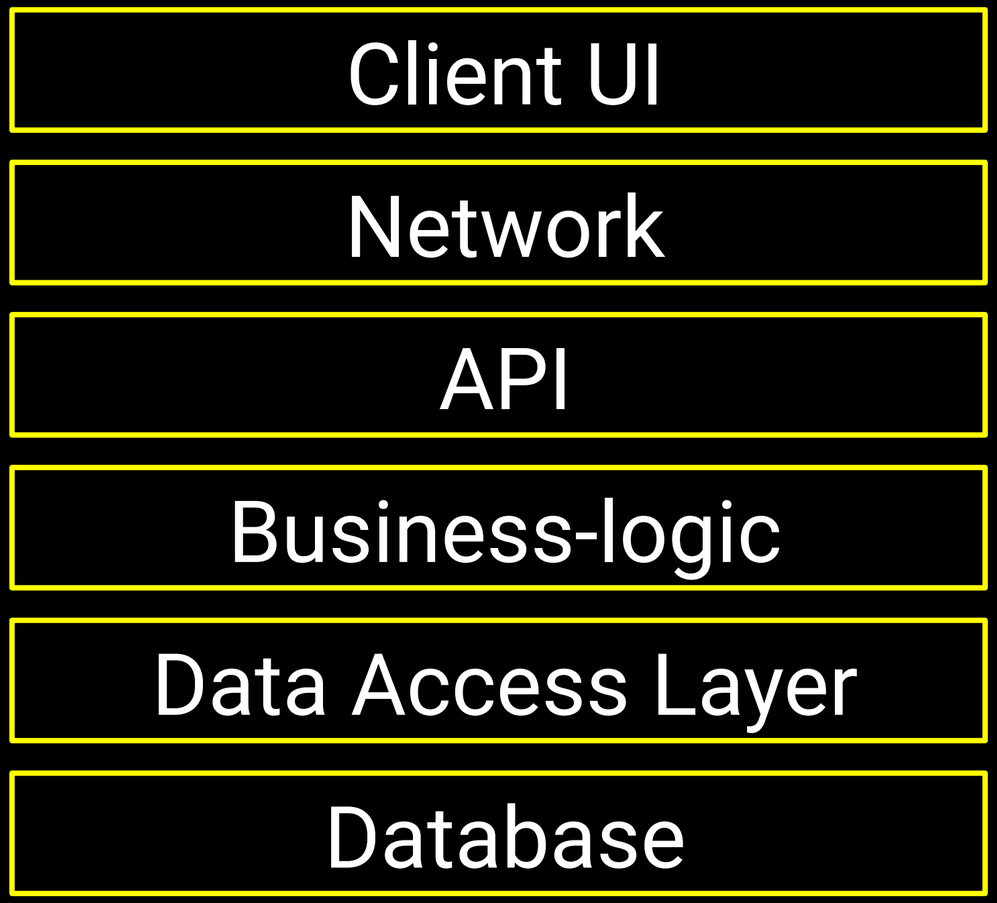
Complex Query

```
id => application.database
  .select('users')
  .where({ id })
  .cache({ timeout: 30000, invalidate: { id })
  .projection({
    name: ['name', toUpperCase],
    age: ['birth', toAge],
    place: ['address', getCity, getGeocode],
  });
```

Layered Architecture

Server-side

- Layered
- Microservices
- Serverless



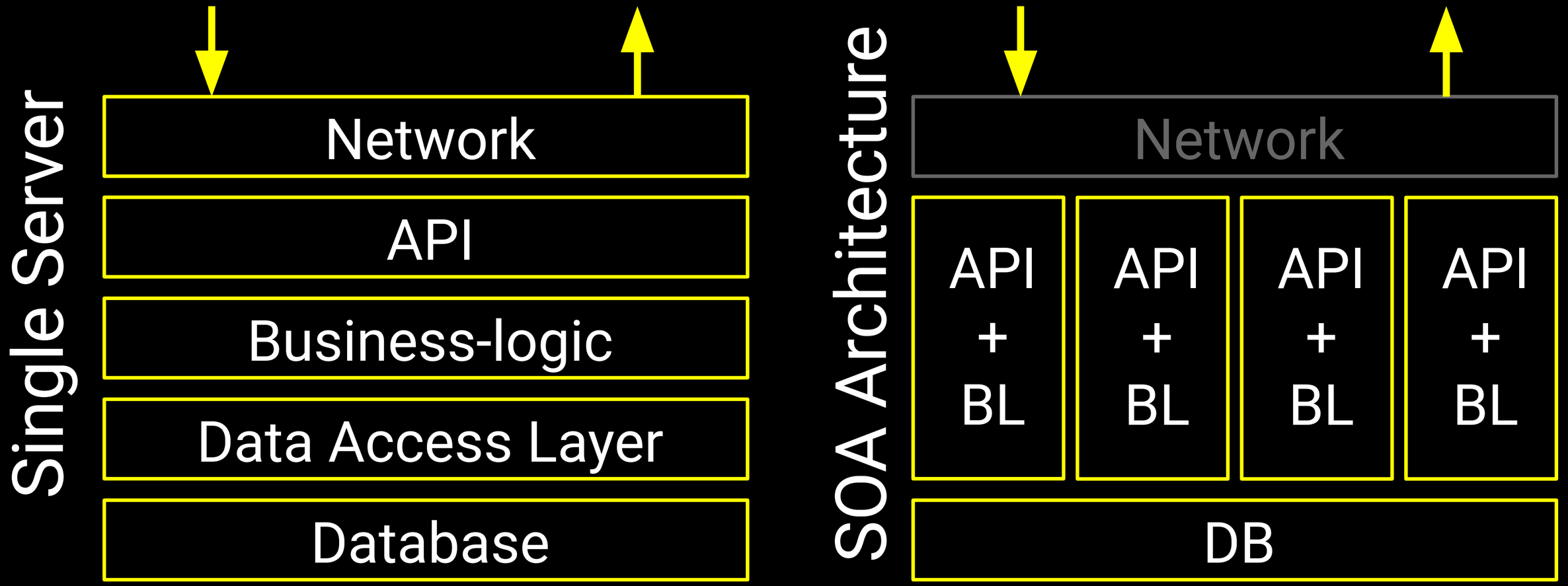
What do we want?

- Apps consolidation
- Stateful cloud applications
- Interactivity (Websockets, TCP, TLS support)
- No vendor lock
- Private clouds
- Do not overpay for clouds

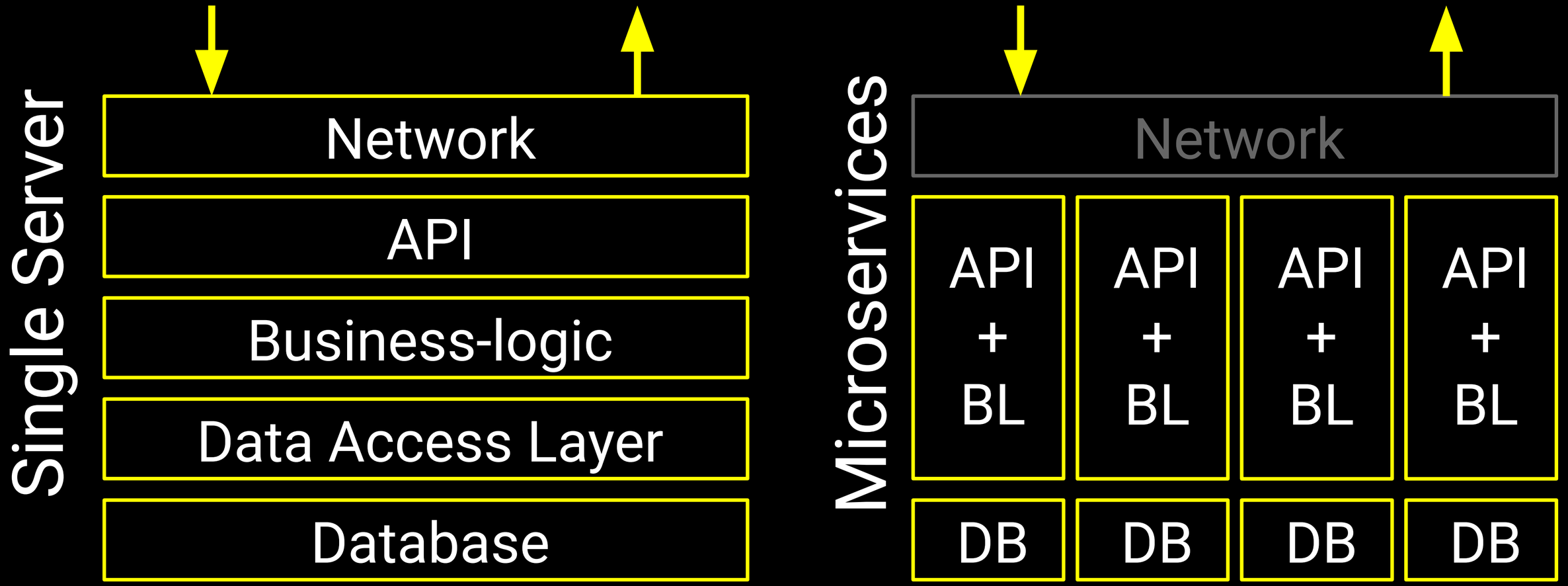
How do we achieve this?

- Architecture and layered approach
- Async I/O for business-logic parallelization
- Long-lived processes: in-memory, reuse
- Server inside application (not vice versa)
- Minimize IPC and serialization
- Open source
- But we need request isolation

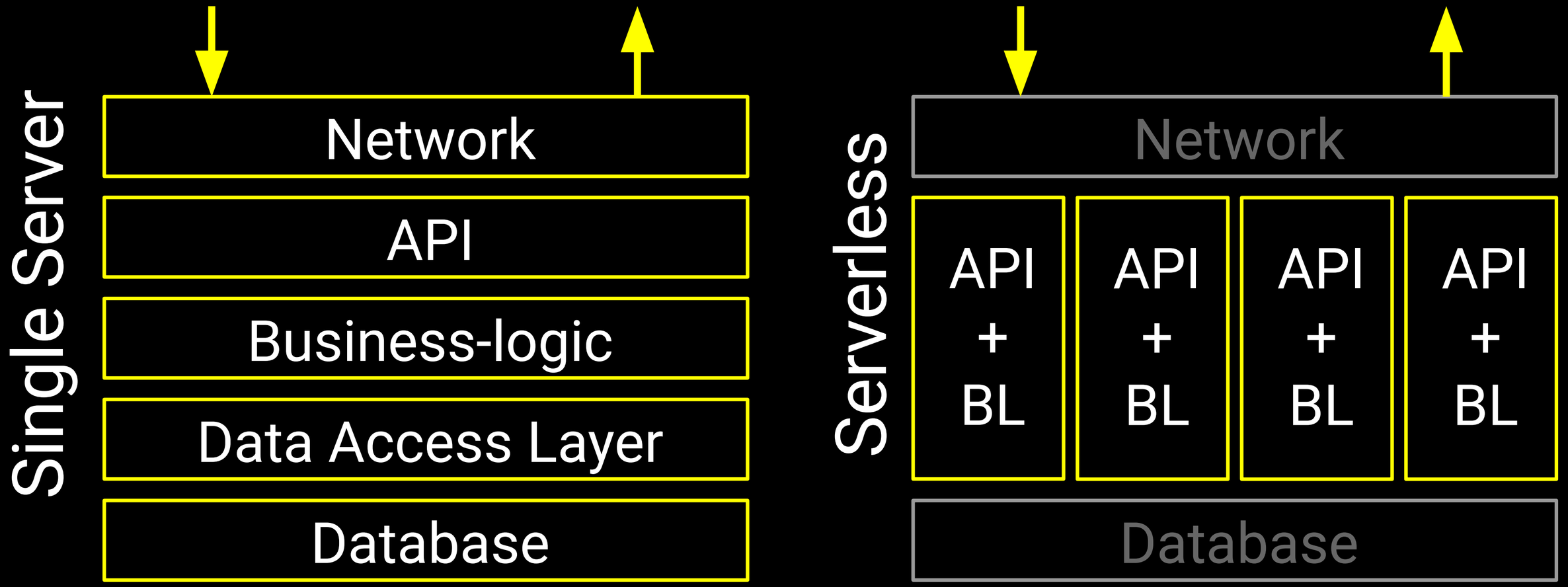
SOA Architecture



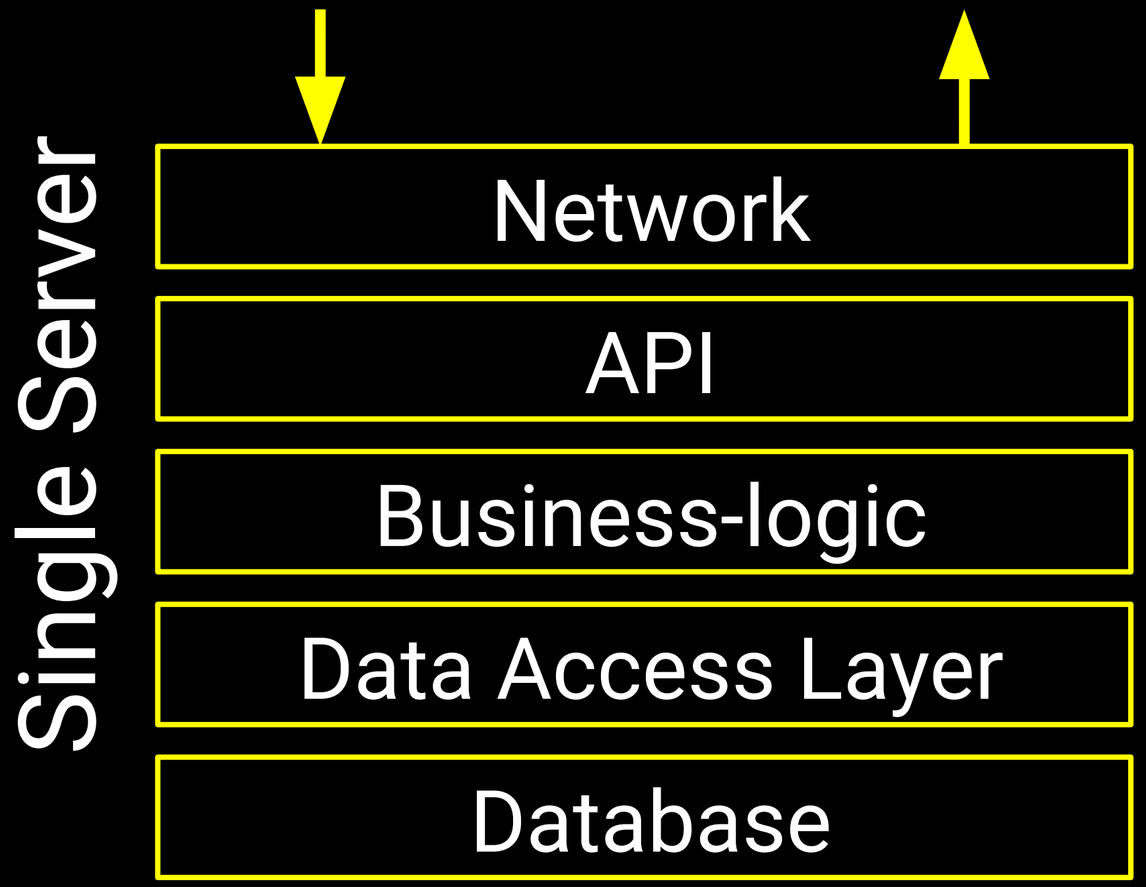
Microservices Architecture

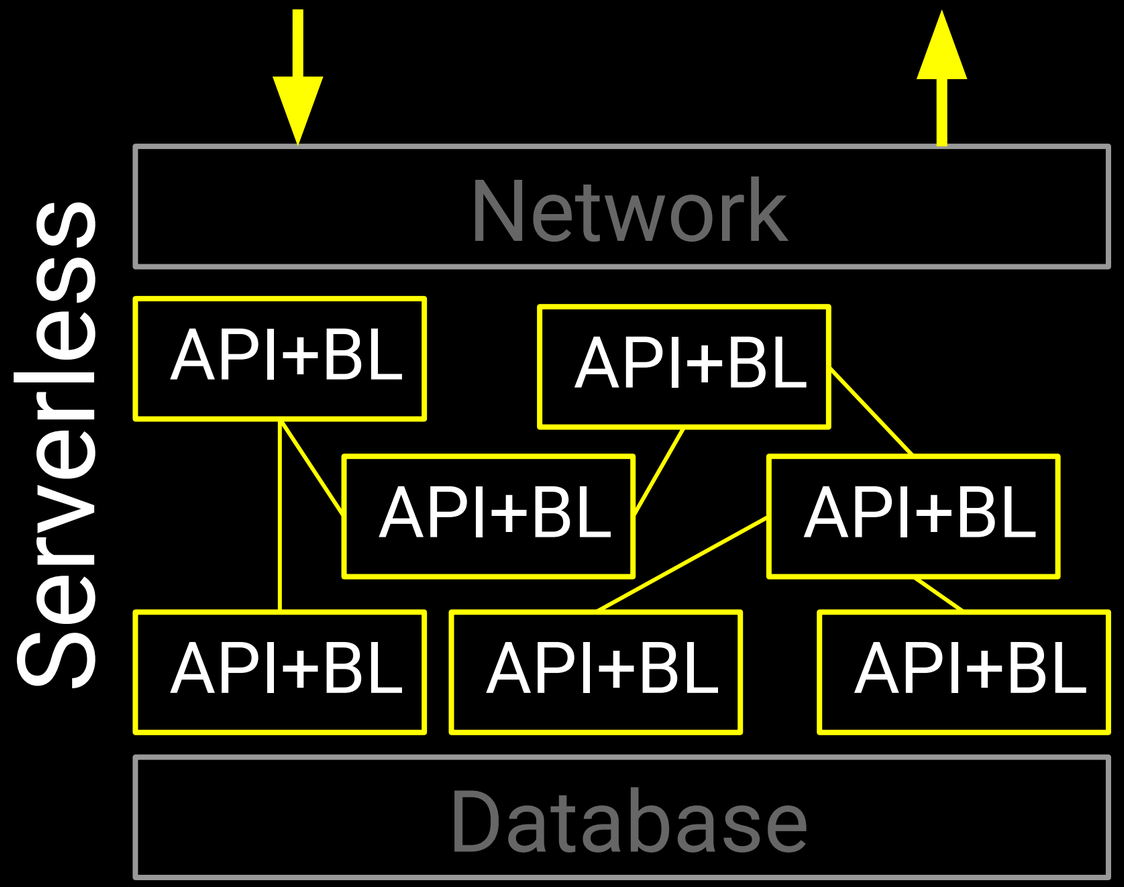


Serverless Architecture

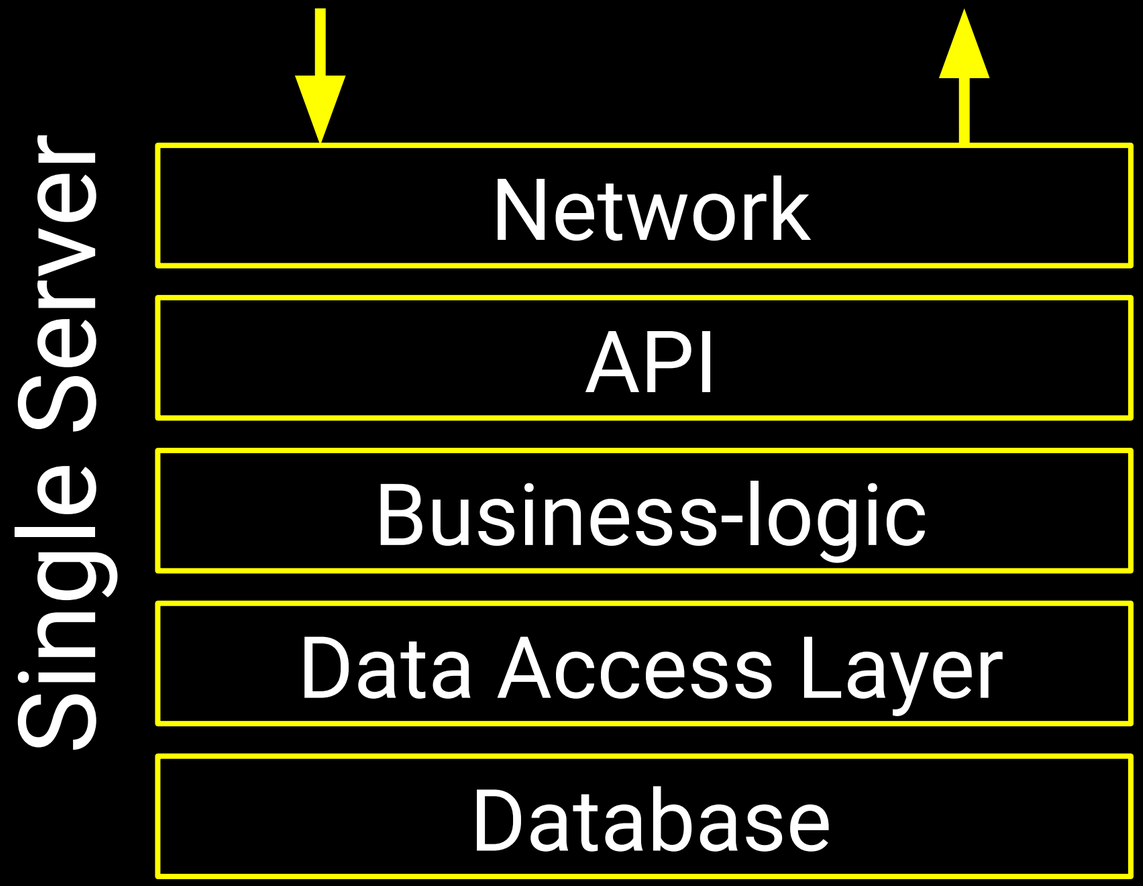


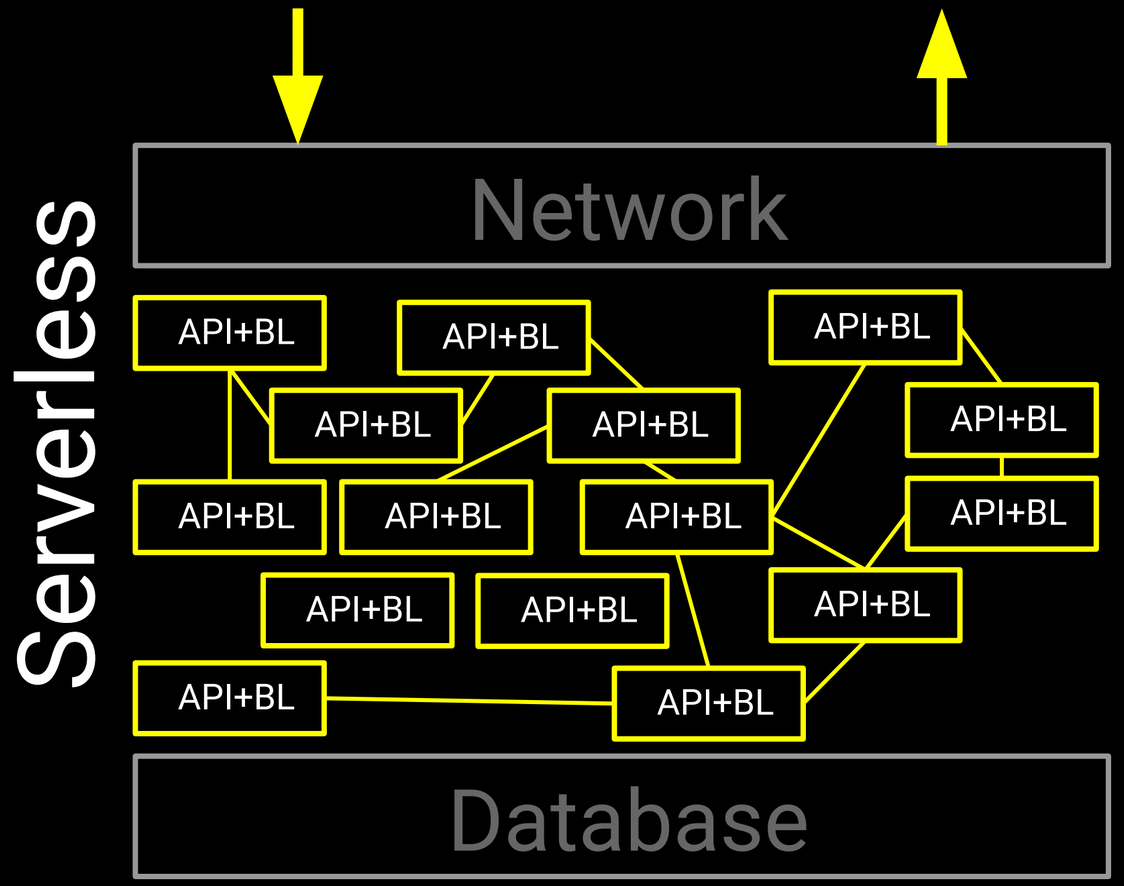
Serverless Architecture



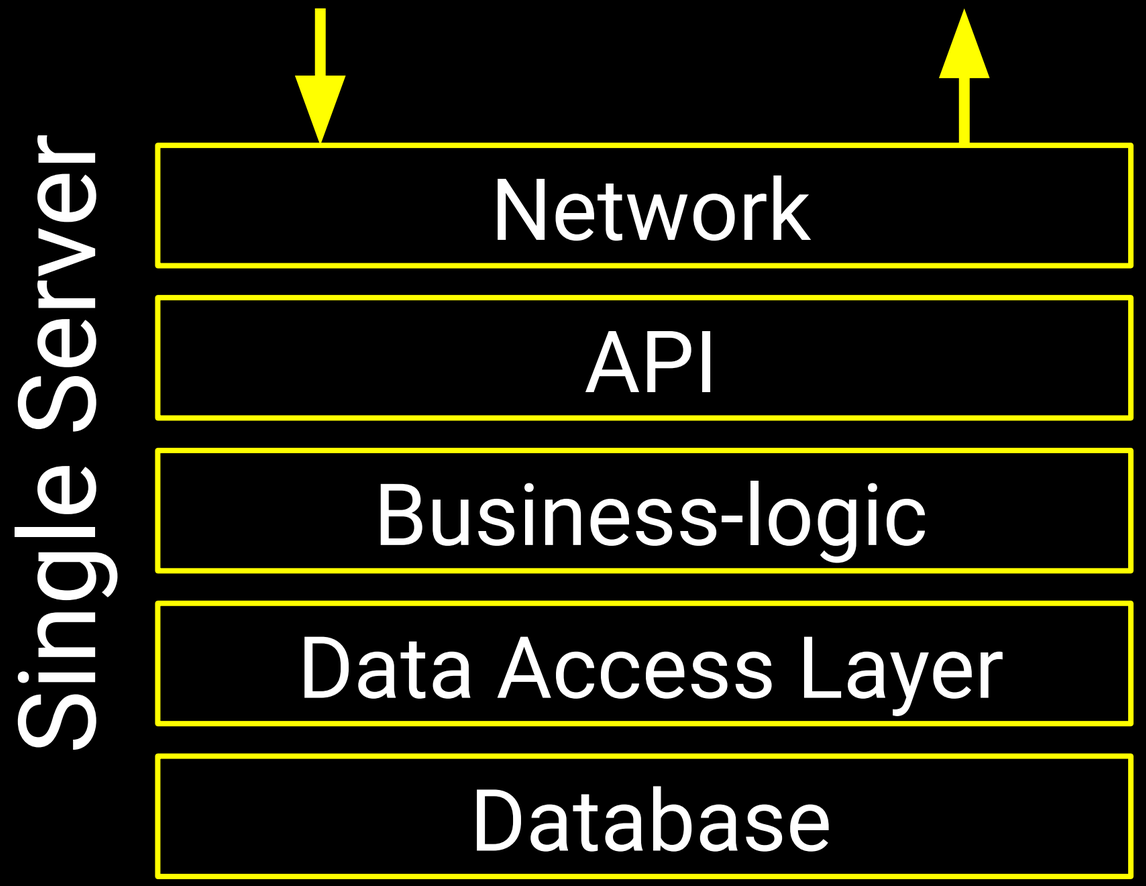


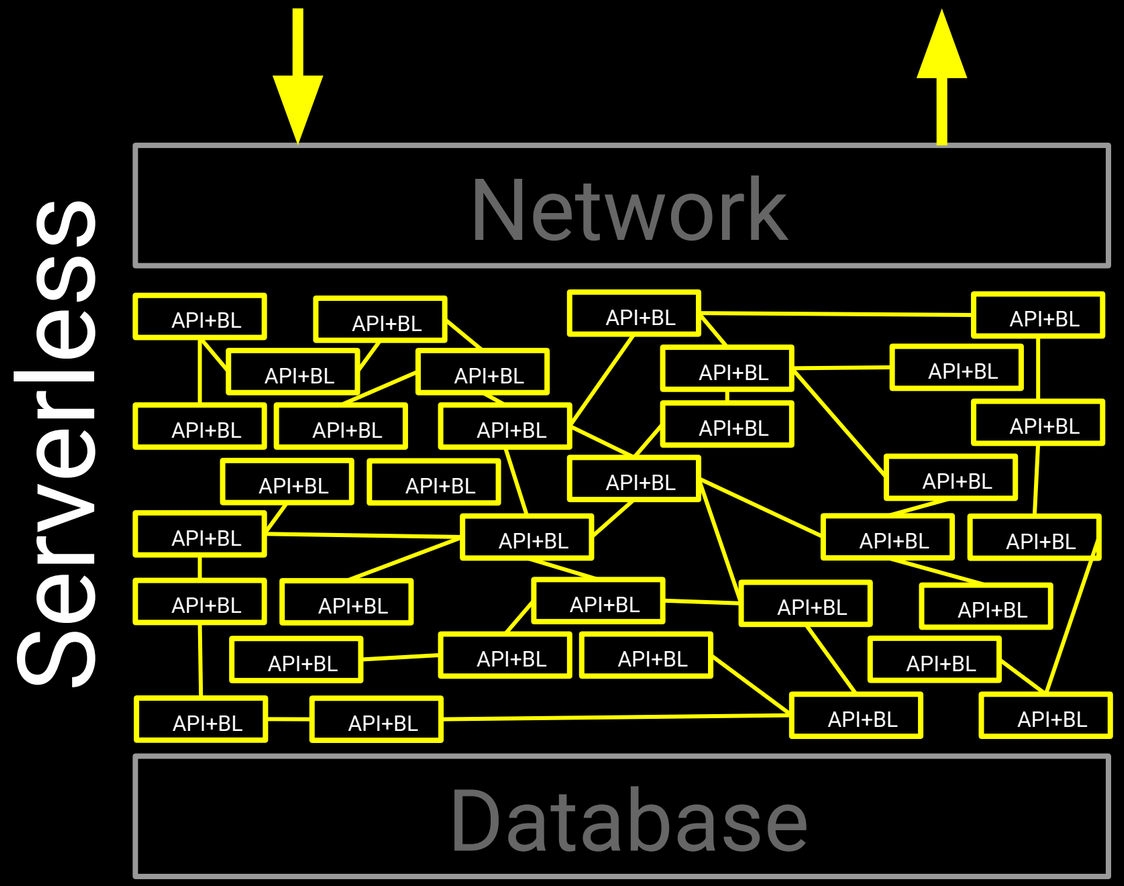
Serverless Architecture





Serverless Architecture





Metaserverless Experiments

- Application is not a separate functions, application has distributed in-memory state
- Functions can be executed sequentially and parallely in asynchronous style
- Applications have long life and structure
- Interactivity (Websockets, TCP, TLS support)
- No vendor lock, Private clouds, Open Source

Vendor Lock Prevention Checklist

- Wrap vendor services
- Concentrate on architecture: layers
- Code quality and competencies
- Think twice before following hype and trends
- Remove dependencies if possible

Vendor Loyalty Checklisk

- Use everything as a service
- Follow guidelines
- Cut risky developments
- Relax
- Share your income

Questions?

github.com/tshemsedinov
https://youtube.com/TimurShemsedinov
github.com/HowProgrammingWorks/Index

Весь курс по ноде (>35.5 часов)
https://habr.com/ru/post/485294/

t.me/HowProgrammingWorks
t.me/NodeUA

timur.shemsedinov@gmail.com