Node.js in 2020 #3

Выйди и зайди нормально

github.com/HowProgrammingWorks

Timur Shemsedinov

Chief Technology Architect at Metarhia
Lecturer at Kiev Polytechnic Institute

github.com/tshemsedinov

Limit concurrency

to avoid resource starvation in high-intensive servers

How to limit concurrency?

- Counter variable
- Array, Linked list
- Asynchronous Queue
- Counting Semaphore
- Event Stream
- External ballancer + monitoring
- It works somehow

Asynchronous Queue Usage

```
const q1 = metasync.queue(3).priority()
  .process((item, cb) => {});

const q2 = metasync.queue(1).wait(100).timeout(200)
  .process((item, cb) => {});

q1.pipe(q2);

q1.add({ id: 1 }, 0);
q1.add({ id: 3 }, 1);
```

Counting Semaphore

```
const semaphore = new CountingSemaphore(concurrency);

const handler = async (req, res) => {
  await semaphore.enter();
  ...
  semaphore.leave();
};
```

Reference implementations

Асинхронная очередь

https://github.com/metarhia/metasync/blob/master/lib/queue.js

Семафор со счетчиком

https://github.com/HowProgrammingWorks/NodejsStarterKit/blob/master/lib/semaphore.js

Graceful shutdown

after fatal errors and for reload applications

When we need to exit?

- On fatal errors, unhandled exceptions
- Update infrastructure or platform
- Scheduled restart, release leaks, etc.
- Manual stop or restart, maintenance

How to shutdown graceful?

- Close server network ports
- Notify all clients with special events
- Wait for timeout
- Close all connections with socket.destroy()
- Save all data and release critical resources
- exit 0

fs.watch

fs.watchFile, fs.FSWatcher and live code reload

Counting Semaphore

```
const fs = require('fs');

fs.watch(dirPath, (event, fileName) => {
  const filePath = path.join(dirPath, fileName);
  reloadFile(filePath);
});
```

Tricks

- Wait for 3-5 sec. timeout after fs.watch event
- Put all changes to collection to reload
- Ignore temporary and unneeded files

Multi-cure support

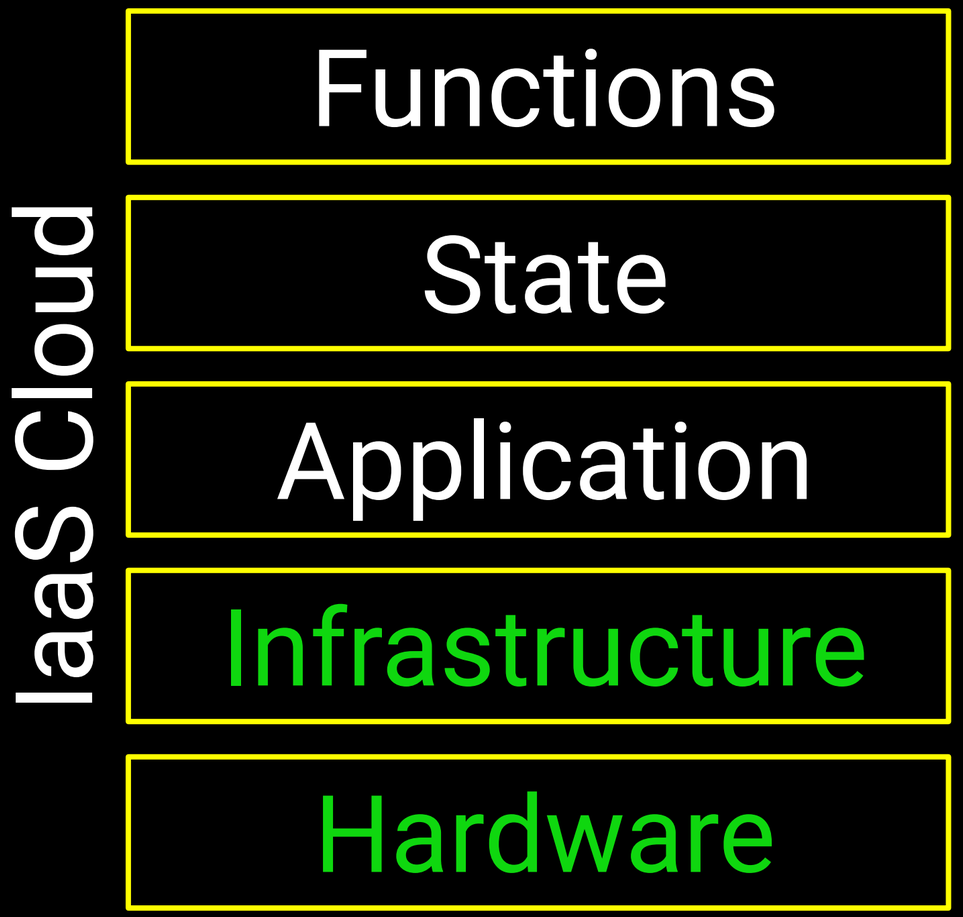
child_process, cluster and worker_threads

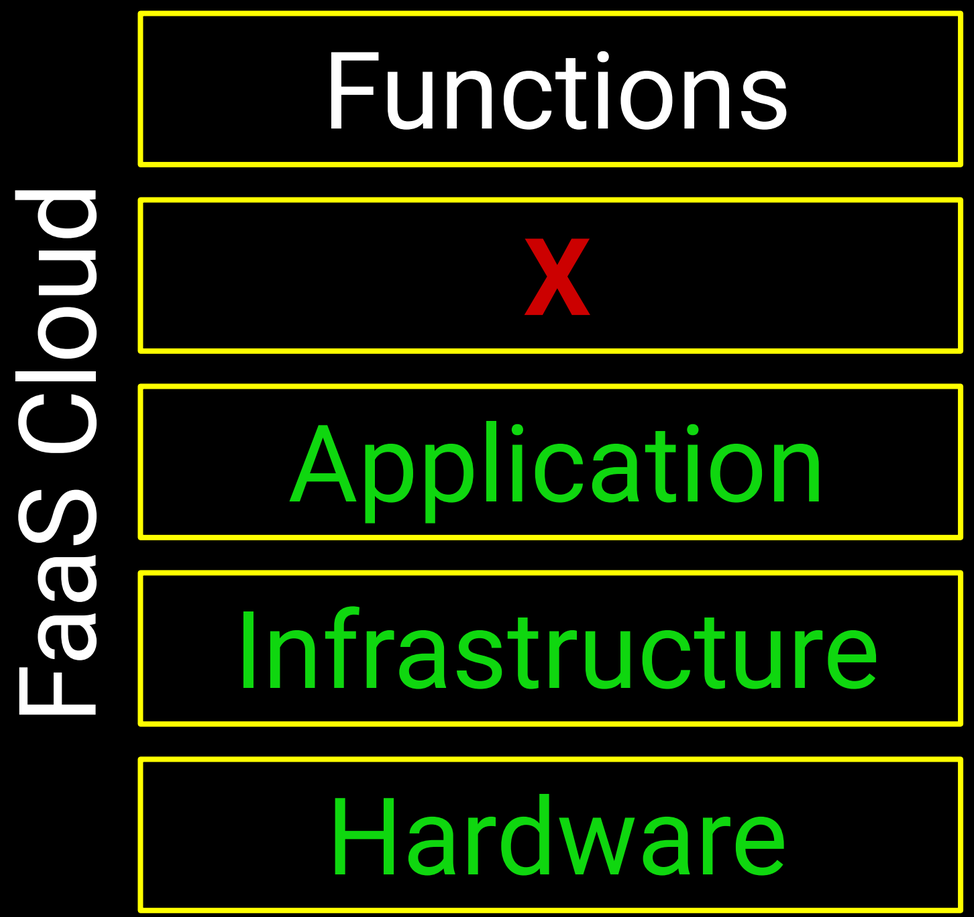
How to use workers_threads for net load

- Don’t run business-logic in main thread
- Spawn one thread per one network port
- Spawn os.cpus().length / 2 threads
- Stick client to certain thread/port
- Separate log file for each thread
- Use shared memory for cache
  (for example static files)

FaaS
Serverless Clouds
and Node.js

Infrastructure Clouds vs App Clouds





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

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 });
```

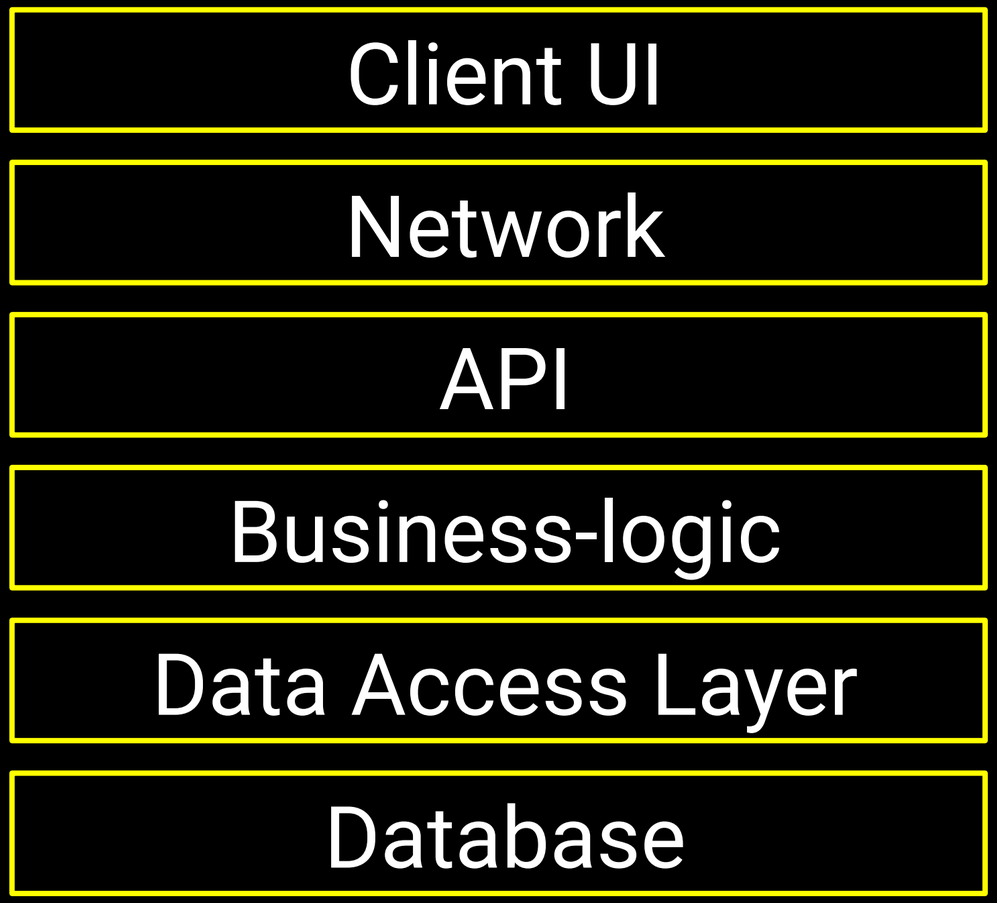
Complex Query

```
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



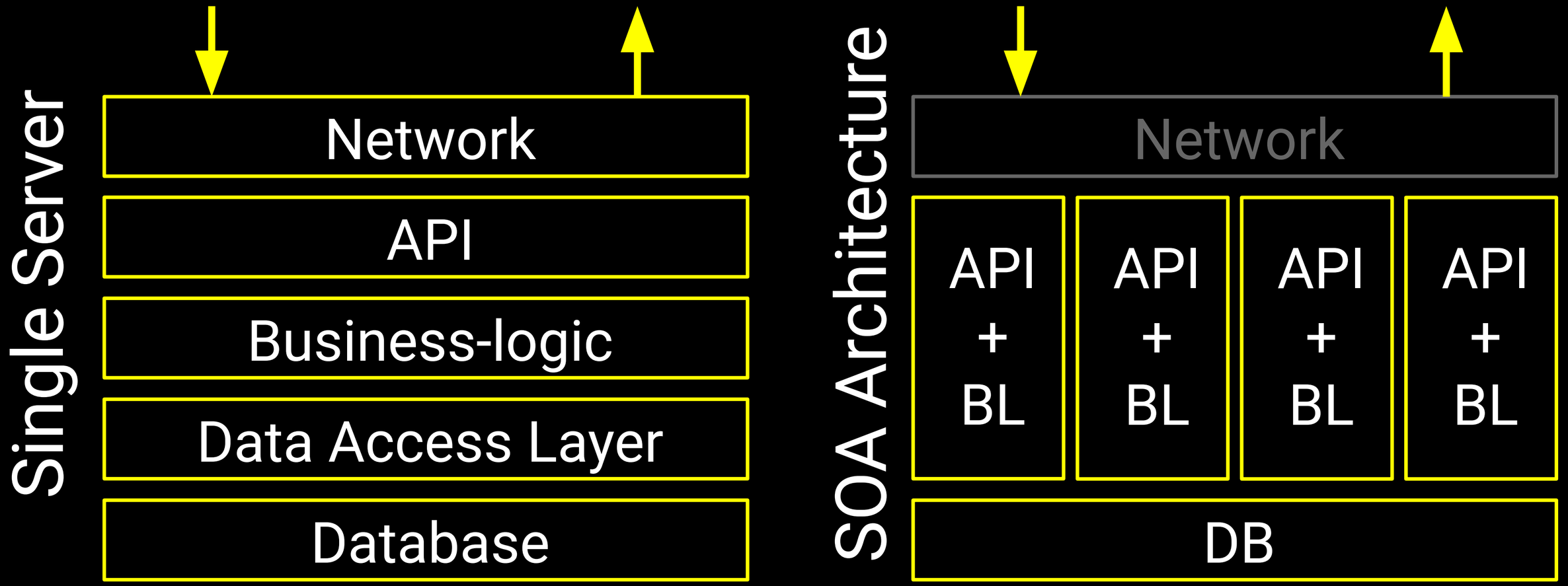
What do we want?

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

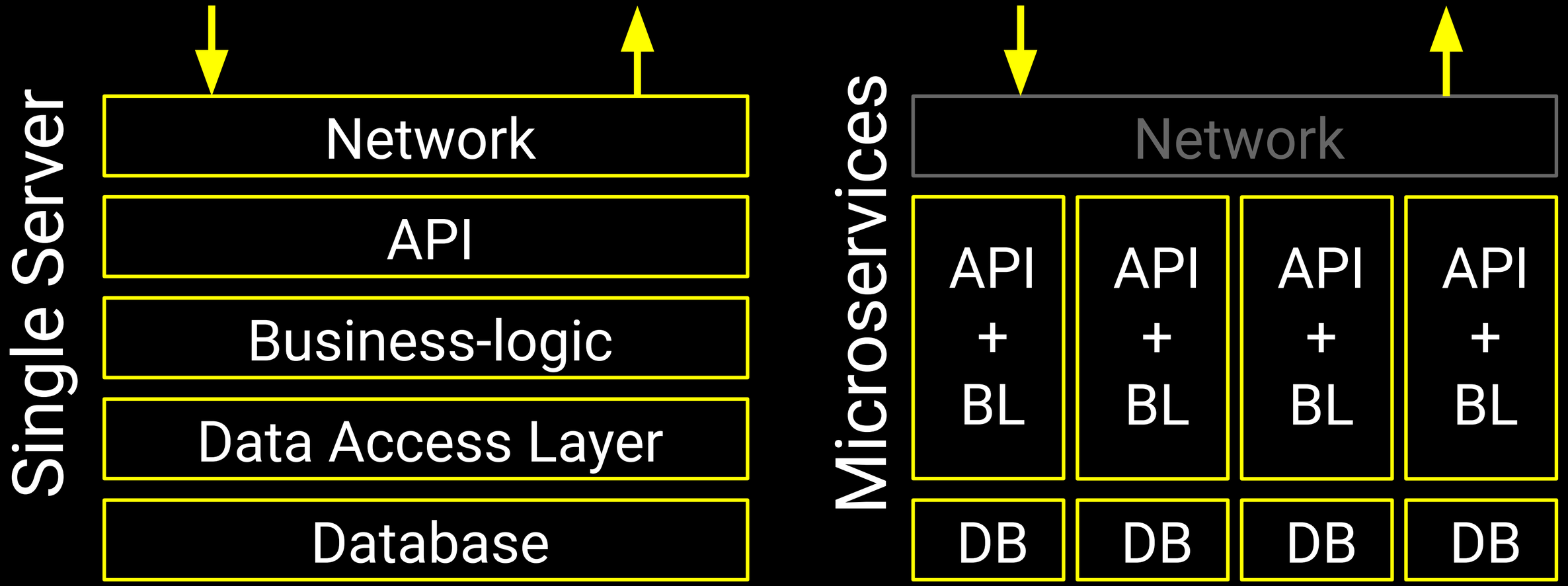
How do we achieve this?

- 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

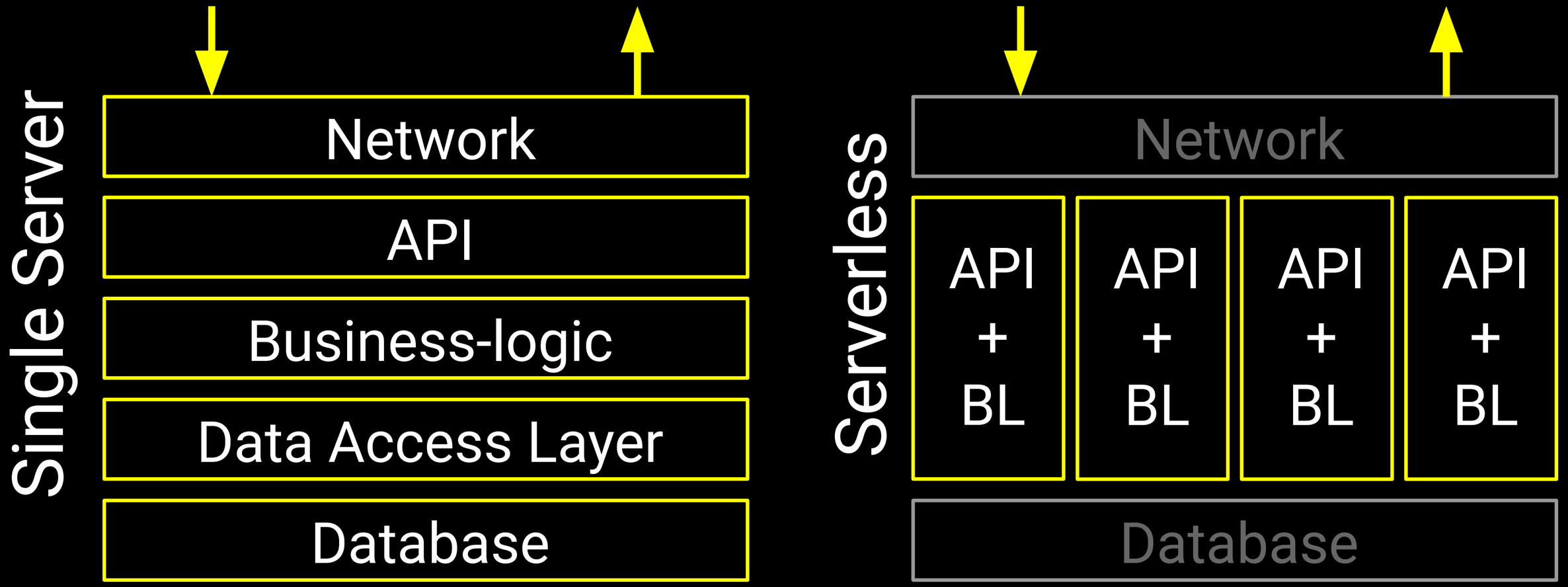
SOA Architecture



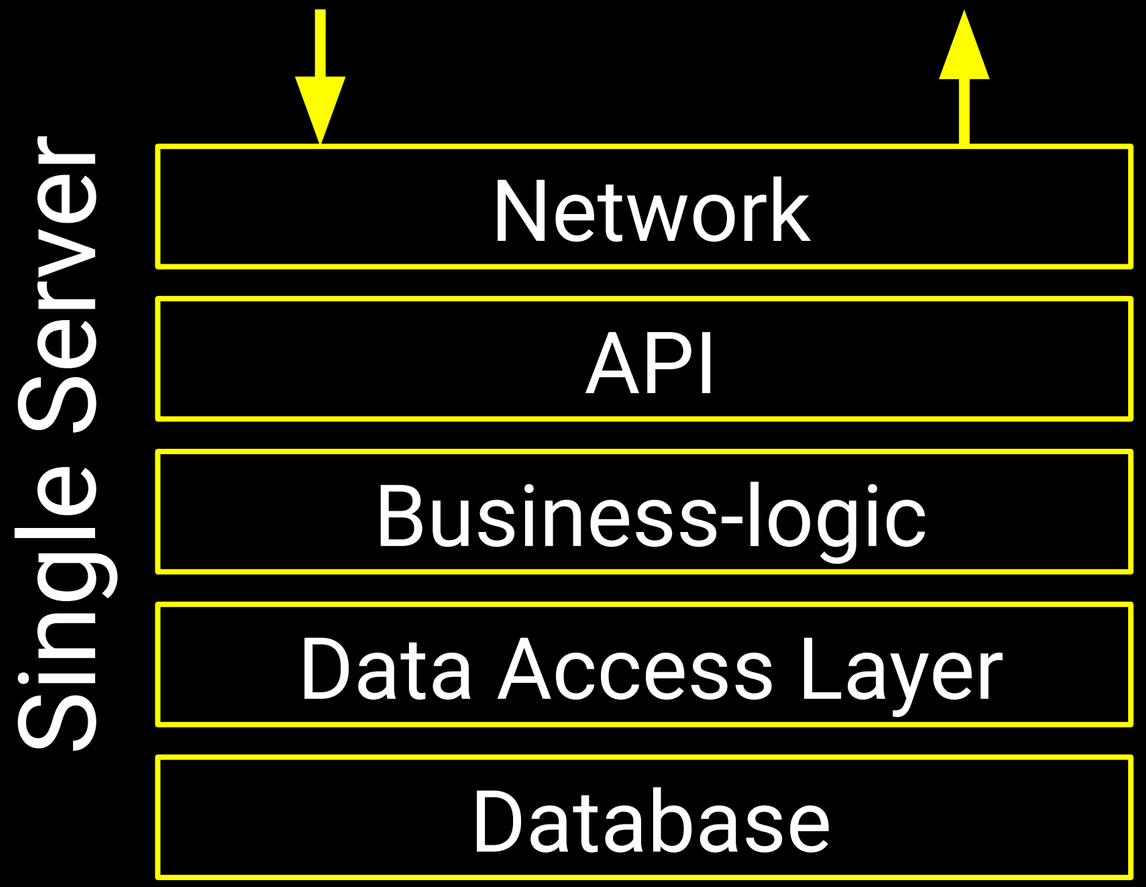
Microservices Architecture

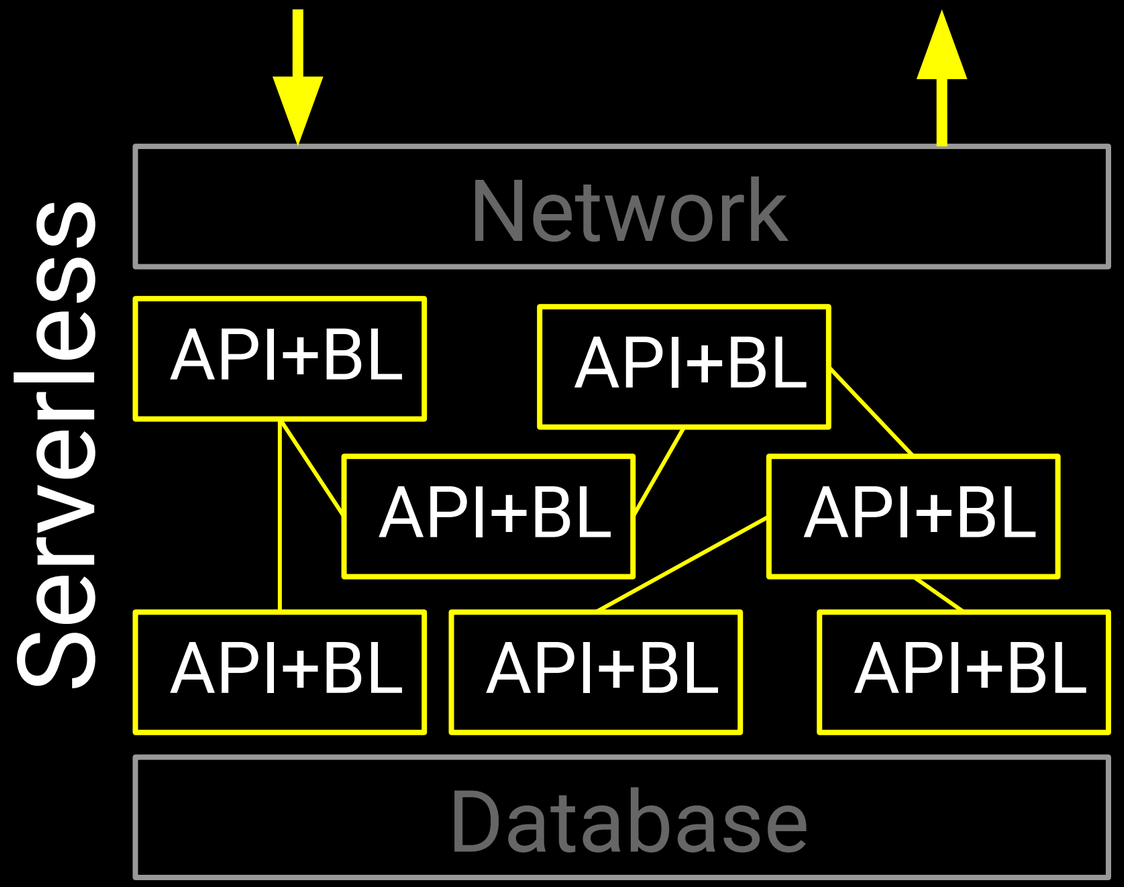


Serverless Architecture

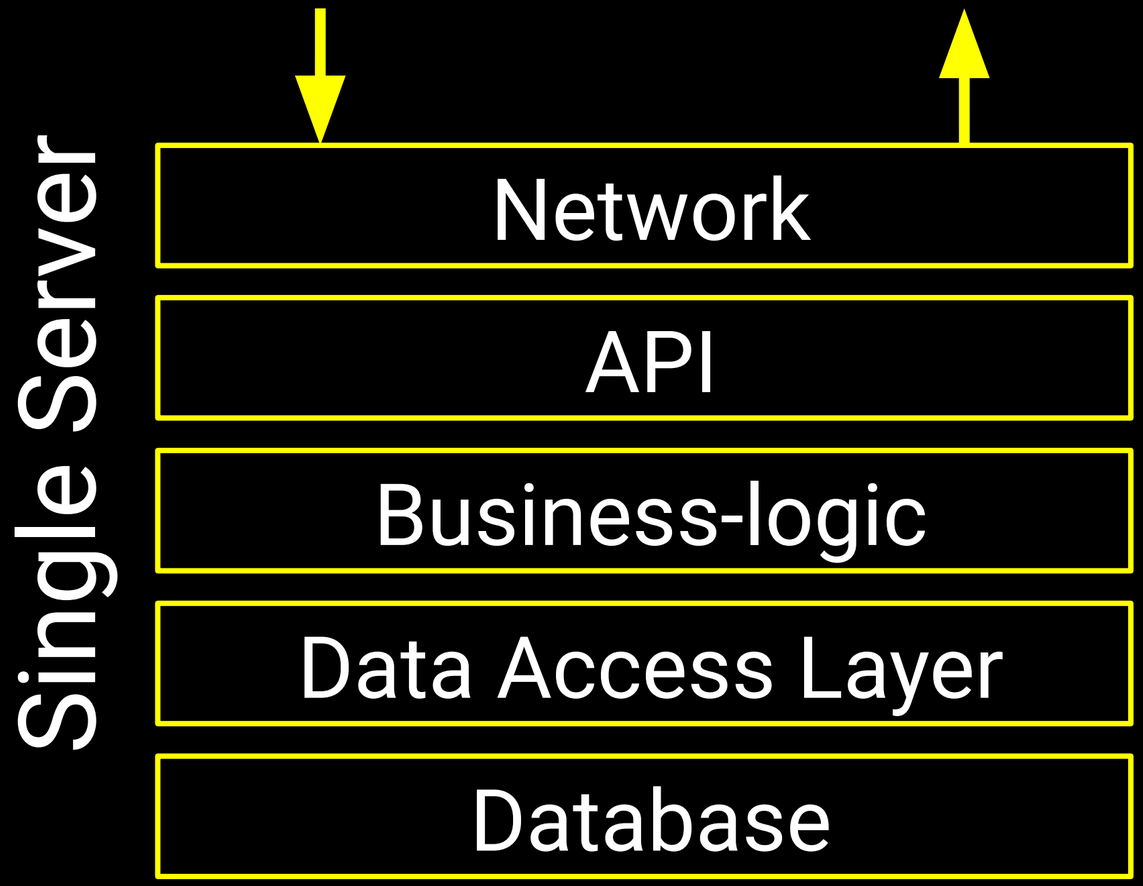


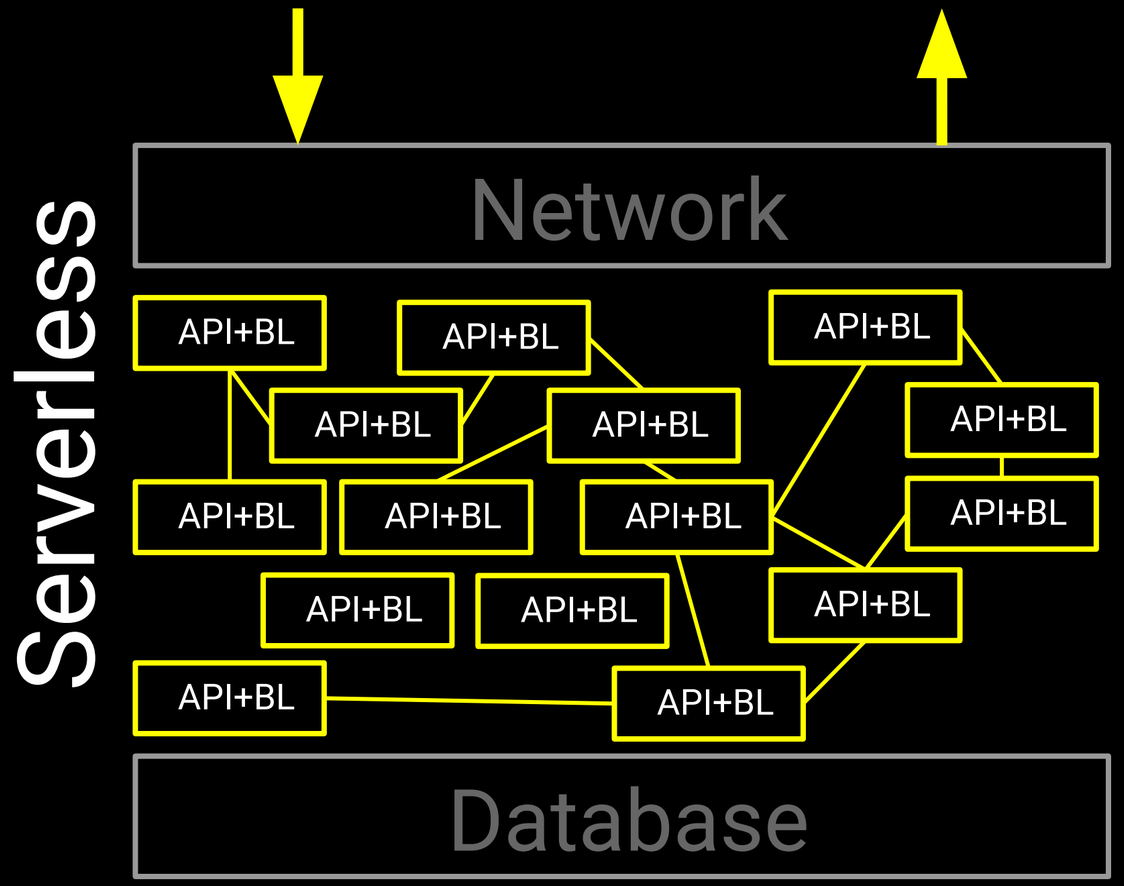
Serverless Architecture



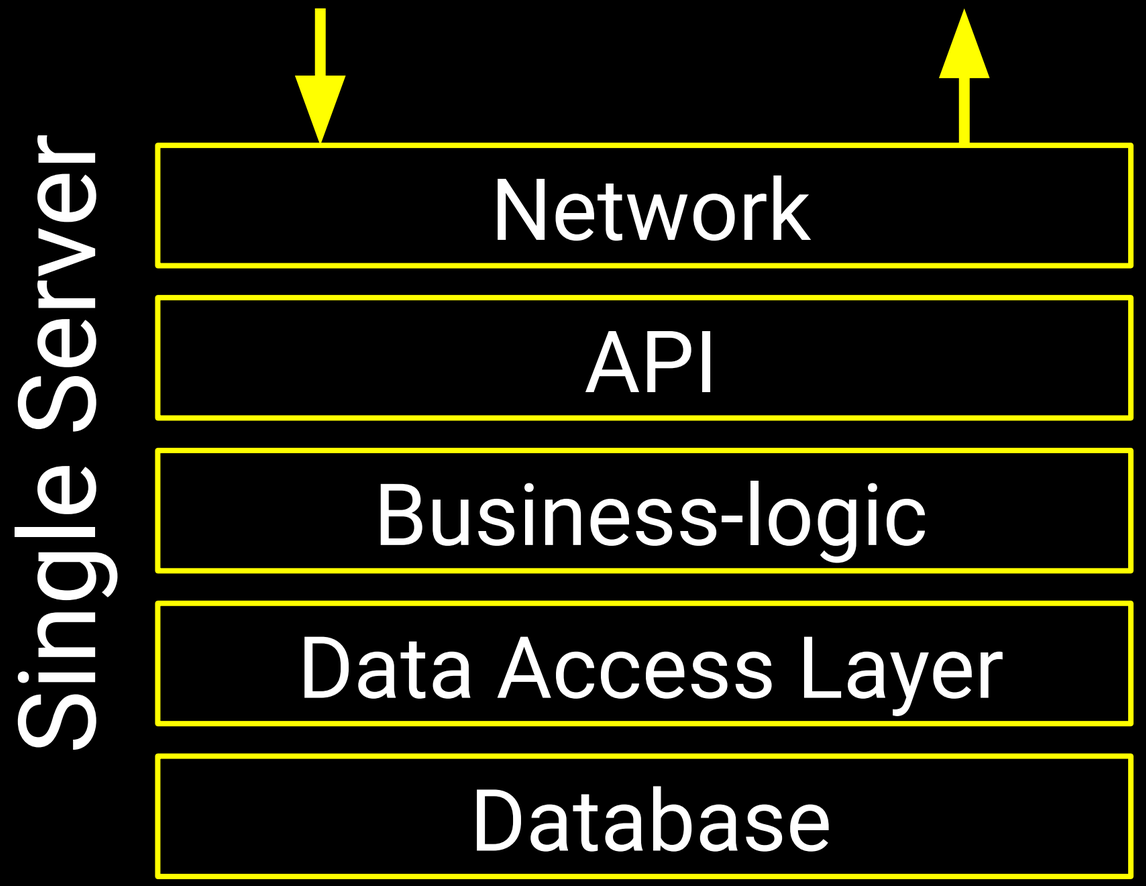


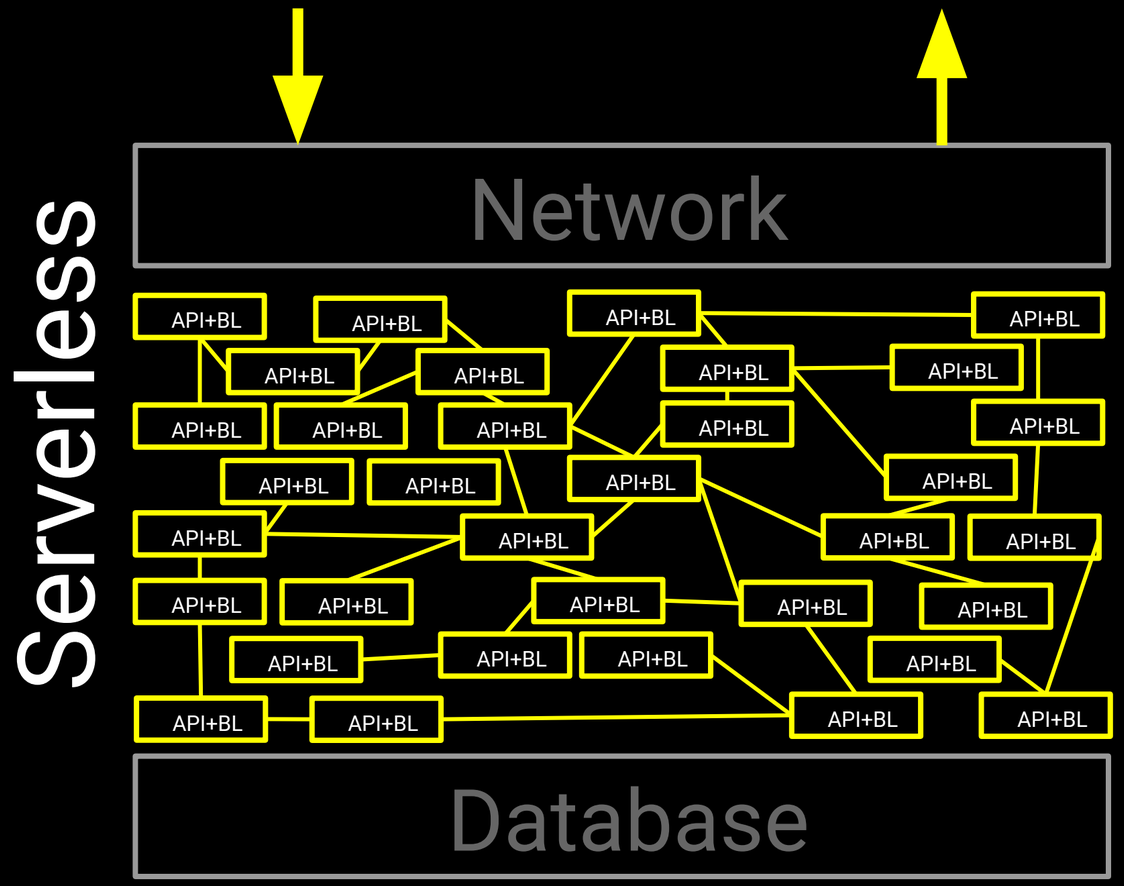
Serverless Architecture





Serverless Architecture





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

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

Node.js Starter Kit

no dependencies, 20 kb with pg drivers + 1.2 mb and ws + 0.24 mb

Starter Kit Key Ideas

- Минимум кода и зависимостей
- Минимизация I/O, отдача всего из памяти
- Безопасность и изоляция контекстов
- Структура и архитектура приложения
- Разделение системного и прикладного слоя
- Все на контрактах (interface)
- Балансировка, таймауты и очередь запросов

Starter Kit Feature List

- Автороутинг API и поддержка HTTP(S), WS(S)
- Подгрузка изменений на лету через fs.watch
- Загрузчик конфигурации и Graceful shutdown
- Утилизация CPU, кластеризация через потоки
- Слой доступа к данным DAL: Postgresql
- Сессии с сохраняемым состоянием
- Песочницы, потоки, масштабирование, тесты

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