Node.js Microservice Architecture

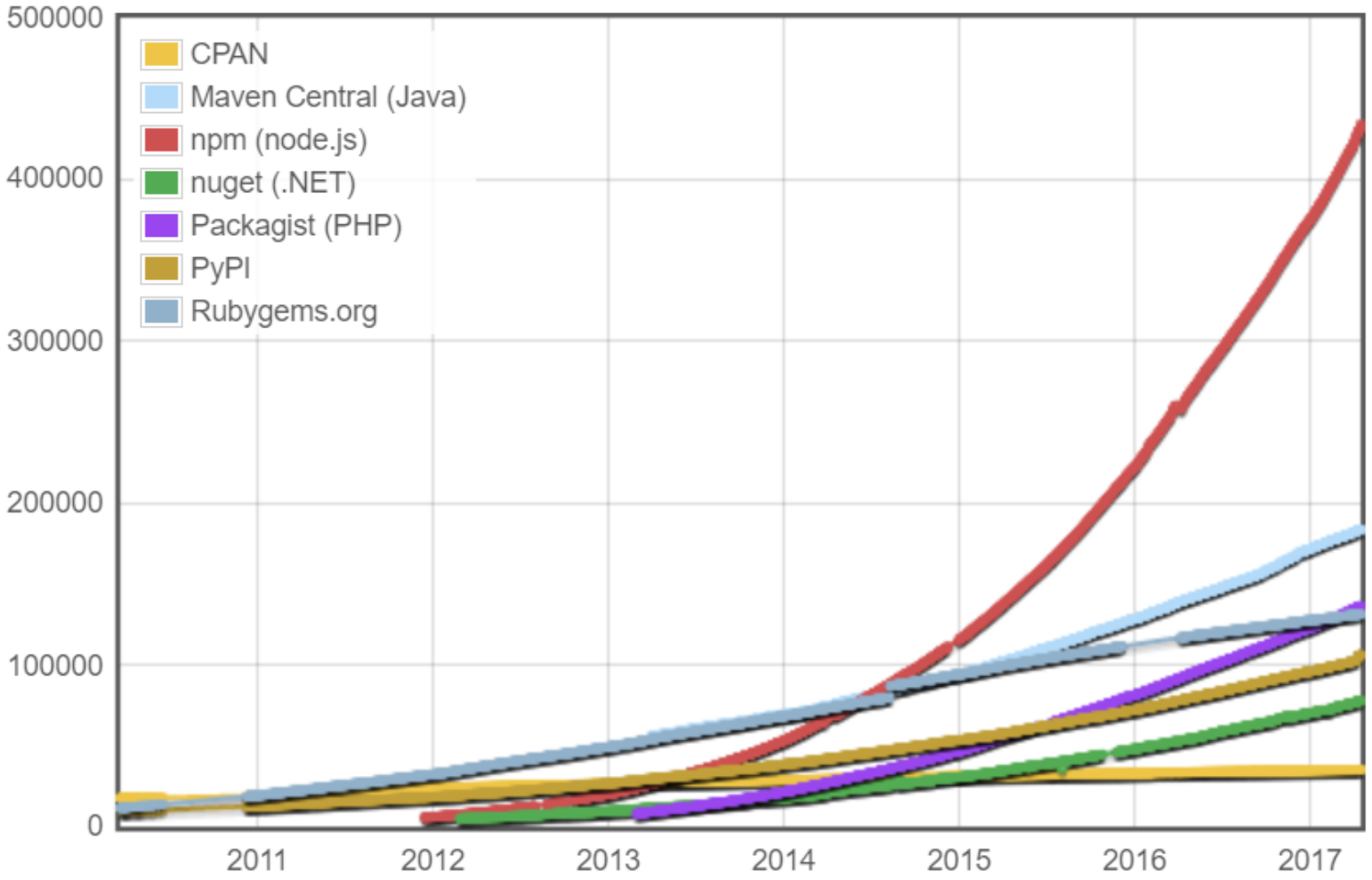
Libor Vilímek – Ackee

Node.js quick summary

- One thread – no deadlocks, not expensive thread creation
- Npm install – as easy as possible
- Javascript – allows almost anything
- Public modules – always open source, huge community
- Asynchronous – no worker threads

The hype is real

Module Counts



No more copy-paste solutions

Microservices in 2017

- Price for services - cheap
- Docker – easy to build, configure
- Cloud – easy to scale, maintain
- Node.js – starts in few seconds

History of Loose Coupling and High Cohesion

- Loose Coupling – do not “couple” things that does not belong to each other
- High Cohesion – Keep close things that should be together
- History: classes, OOP, Design Patterns, Enterprise Service Bus
- “New”: Microservices

Data Transport

- WSDL + SOAP – Bad readability, too complex
- REST API – no dependencies, good readability
- Queues – library/service dependent, can be safer

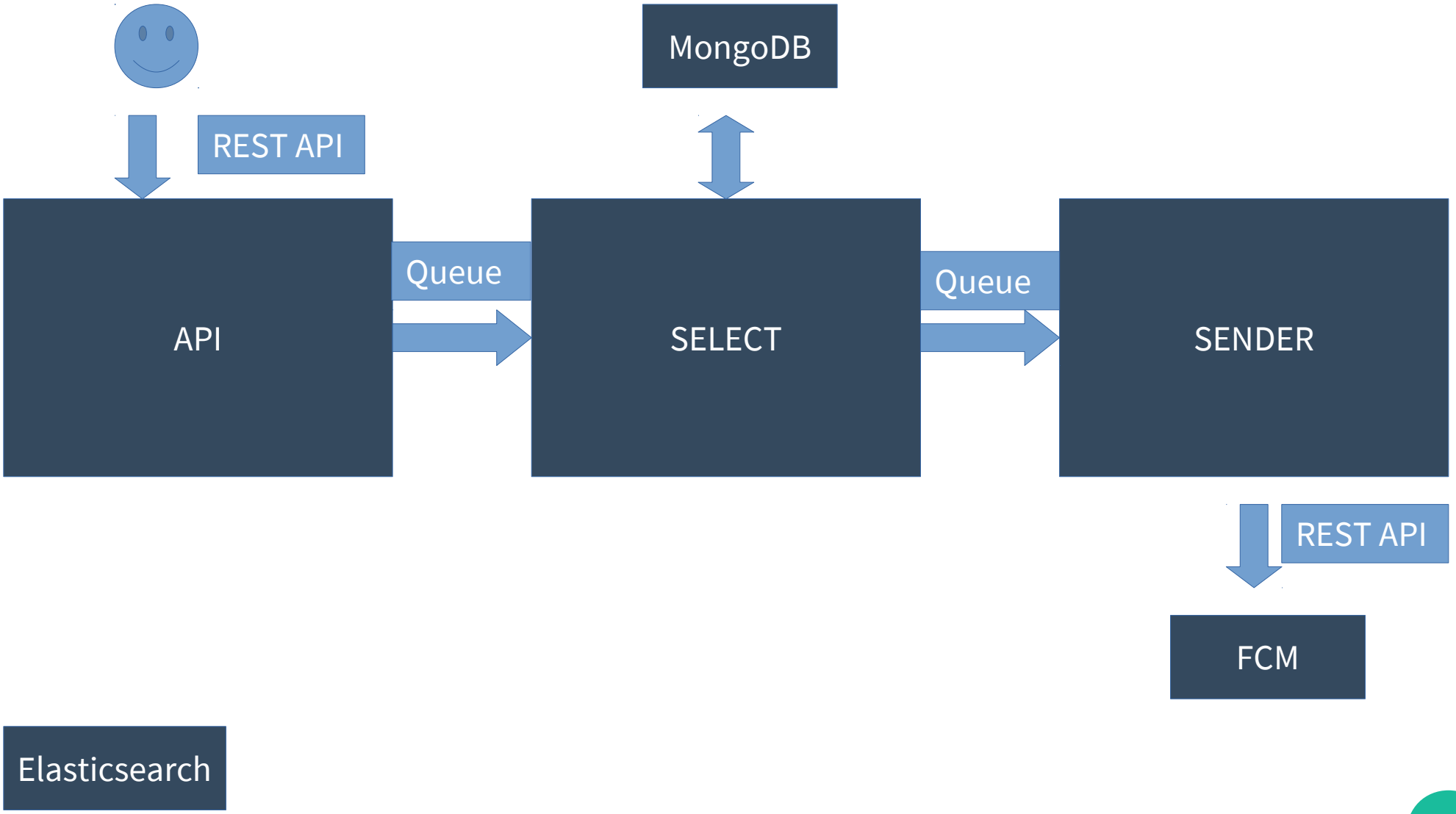
Ackee Push Server – How it started

- At January 2017 - Parse was turned off
- If already developed “push service” is not viable for Facebook, can it be viable for another company or startup?
- How hard can it be to implement simple Push Server using FCM (Firebase Cloud Messaging)?

Ackee Push Server - Requirements

- Must always deliver push
- Downtime is not acceptable (several apps would depend on it)
- Developers should get info about progress
- Server must be able to target devices by custom variables

Ackee Push Server - Solution



Ackee Push Server - Solution

- API – no logic, targets message to right queue, can return 400 or 202
- SELECT – have database, select or save devices
- SENDER - communicate with FCM
- All microservices are using Elastic for logging data for developers
- Communication between services go through Cloud Pub/Sub queues

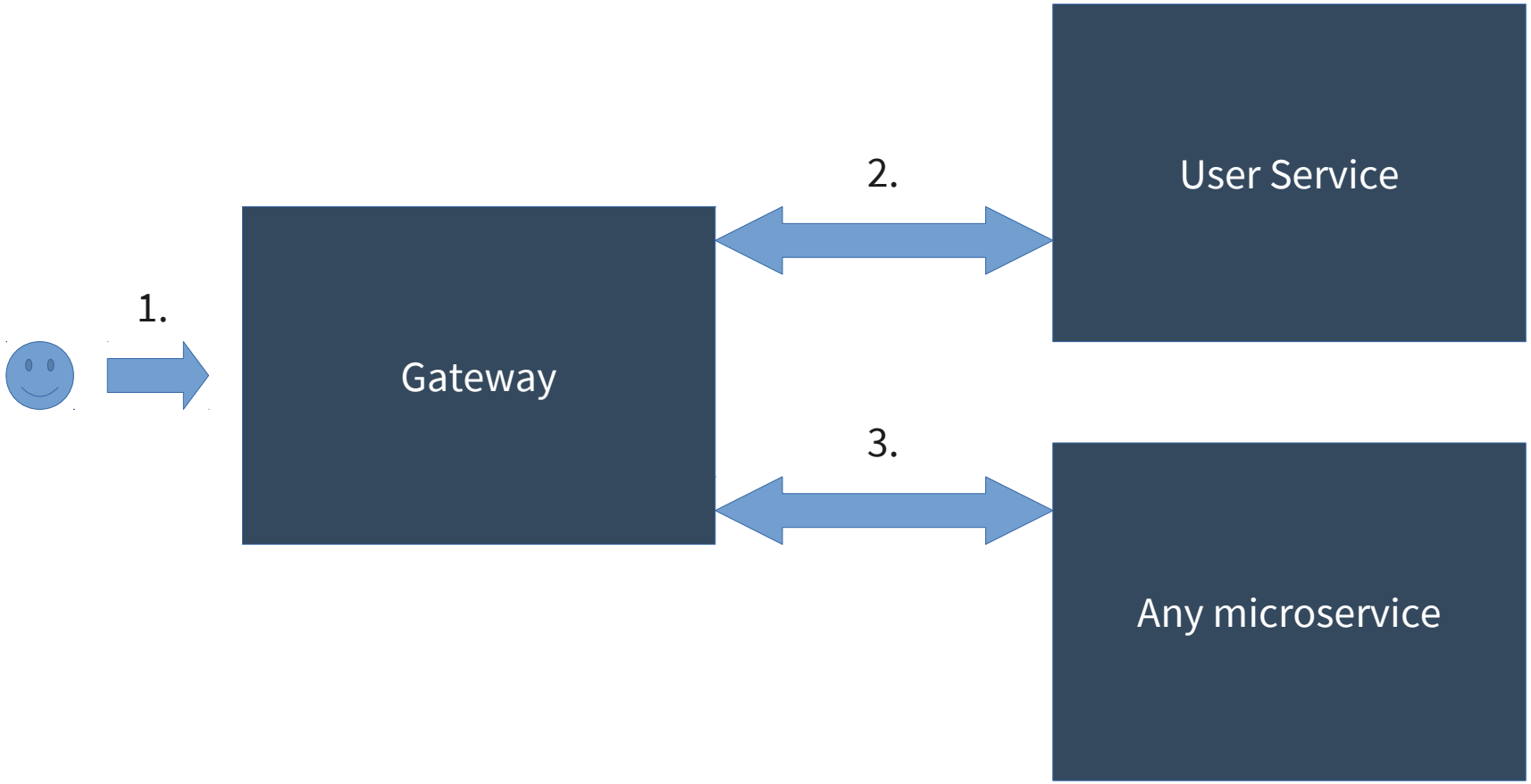
Ackee Push Server - Results

- Queues – once message get in, it is not resolved until everything finishes as expected
- API – having as simple as possible REST API helps with minimize chance of any bug
- MongoDB – easy for creating new “columns”
- Winston-Elastic – good for structured data
- Microservices – low codebase, updates affects less
- Cloud and Load Balancer – no downtime

User Service – how it started

- We already had Node.js template for our typical use-cases – register, log-in, refreshing tokens
- Used in multiple project – how update easily?
- Do we need authetincation data after their validation?
- How we handle users through multiple micorservices?

User Service – Solution



User Service – Solution

- 1) Gateway – Use Authorization header and send it to User Service
- 2) User Service – handles authorization and returns user or 401
- 3) Gateway – pack user into JWT token and send in Authorization header
- 4) Any microservice – unpack JWT token

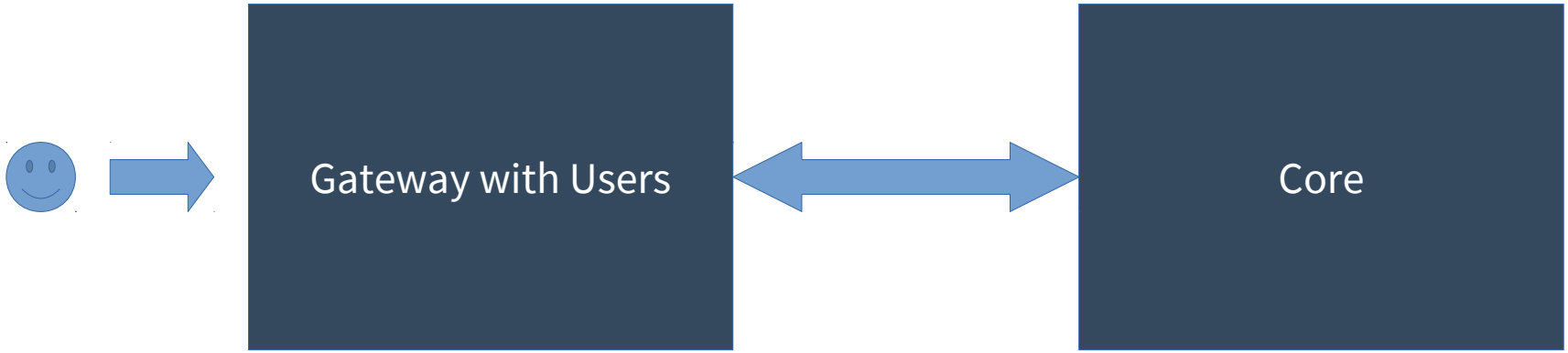
User Service - advantages

- Changes in our Template can be easily “copy-pasted” into User Service used in already existing projects
- Keep lower codebase
- Bcrypt password generation/comparison can be CPU heavy – separates it from the other services

Microservice Architecture - Stateless

- Do NOT try to do transactions through (micro)services
- Design application for “This part does not have to know about something NOW”
- Use lazy-load
- Anything can fail “in middle” - have architecture ready for it

Fails – integrate User Service into Gateway



Fails – integrate User Service into Gateway

- No stateless – user creation that failed “in middle” could lead to inconsistency
- Transaction through microservices – user must be in both microservices at once or not at all
- This approach lead us to extensive ammount of new code in Gateway
- Solution was too specific for current project – cannot be reused

Expect the Unexpected

Even Node.js cant save you by itself

```
exports.createPayment = function(object, cb) {
    var rand = Math.abs((Math.random() * (9999999999 - 1) + 1) | 0).toString();
    // console.log(rand);
    Payment.findOne({vs: rand}, function(err, obj) {
        if (obj) {
            rand = Math.abs((Math.random() * (9999999999 - 1) + 1) | 0).toString();
            // console.log(rand);
            Payment.findOne({vs: rand}, function(err, obj) {
                if (obj) {
                    rand = Math.abs((Math.random() * (9999999999 - 1) + 1) | 0).toString();
                    // console.log(rand);
                    Payment.findOne({vs: rand}, function(err, obj) {
                        if (obj) {
                            rand = Math.abs((Math.random() * (9999999999 - 1) + 1) | 0).toString();
                            // console.log(rand);
                            Payment.findOne({vs: rand}, function(err, obj) {
                                if (obj) {
                                    rand = Math.abs((Math.random() * (9999999999 - 1) + 1) | 0).toString();
                                    // console.log(rand);
                                    Payment.findOne({vs: rand}, function(err, obj) {
                                        if (obj) {
                                            cb(1, null);
                                        } else {
                                            object.vs = rand;
                                            var create = new Payment(object);
                                            create.save(function(err, payment) {
                                                cb(err, payment);
                                            });
                                        }
                                    });
                                } else {
                                    object.vs = rand;
                                    var create = new Payment(object);
                                    create.save(function(err, payment) {
                                        cb(err, payment);
                                    });
                                }
                            });
                        } else {
                            object.vs = rand;
                            var create = new Payment(object);
                            create.save(function(err, payment) {
                                cb(err, payment);
                            });
                        }
                    });
                } else {
                    object.vs = rand;
                    var create = new Payment(object);
                    create.save(function(err, payment) {
                        cb(err, payment);
                    });
                }
            });
        } else {
            object.vs = rand;
            var create = new Payment(object);
            create.save(function(err, payment) {
                cb(err, payment);
            });
        }
    });
};
```

Live demo

- Live demo

Thank you for your attention

Questions?

github.com/AckeeCZ/nodejs-berlin-demo

linkedin.com/in/libor-vilimek/

ackee.de