Node.js

발표의 목적

Node.js의 애플리케이션을 개발하는 View 도구를 간단히 살펴본다.

어디에 Node.js를 적용하면 좋을지 그 특징을 살펴본다.

express 프레임워크를 사용하여, 얼마나 간단히 Node.js 애플리케이션이 생성되는 알게된다.

Node.js 목차

- front
  - Vue & express & ejs & pug에 대한 고민
- back(node.js)
  - Node.js 특징
  - CPU/Core/Thread???
  - Node.js Application 작동방식
  - IO Non-Blocking과 Blocking
  - 비동기/동기
  - Good 유즈케이스
  - Bad 유즈케이스
  - express Application 만들고, 작동 방식 이해하기

Front for Node.js

Express 프레임워크

- Express는 node.js를 위한 빠르고 간편하게 웹 어플리케이션 생성
- 가장 많이 사용.
- node.js를 위한 express 사용...

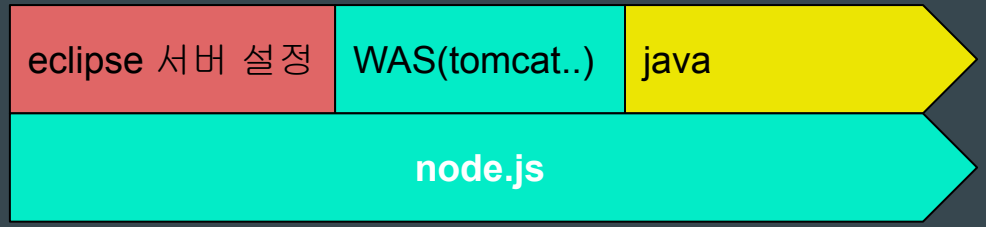
Vue.js

- React와 Vue 모두 **가상의 DOM추상화를 사용하여 이 작업을 수행**하며 두가지 구현 모두 거의 동일하게 작동한다.
- DOM 조작에 가능한 적은 오버헤드(순수 javascript계산)만 가한다. Vue만의 특징

React.js

Node.js의 특징

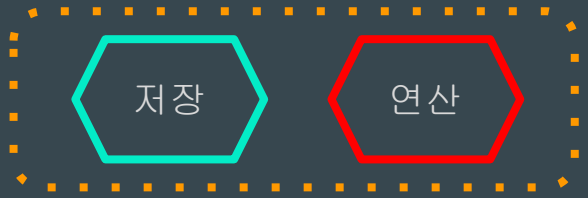
- JavaScript를 사용한다.
- 탁월한 생산성 보장한다.



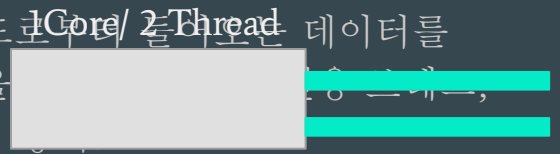
- 순차방식 프로그래밍이 아닌 이벤트기반의 프로그래밍 모델을 사용한다.
- Node.js 는 Single Thread 기반으로 동작하는 비동기 IO(Async/Non-blocking)를 지원하는 네트워크 서버임.

단일 스레드 구조 하나의 작업이 시간이 많이 소요될 경우 스레드가 처리해야 하는 작업이 밀리게 되고 전체적인 시스템 성능이 낮아진다. 하나의 작업시간이 작은 것 위주로 처리해야 한다.

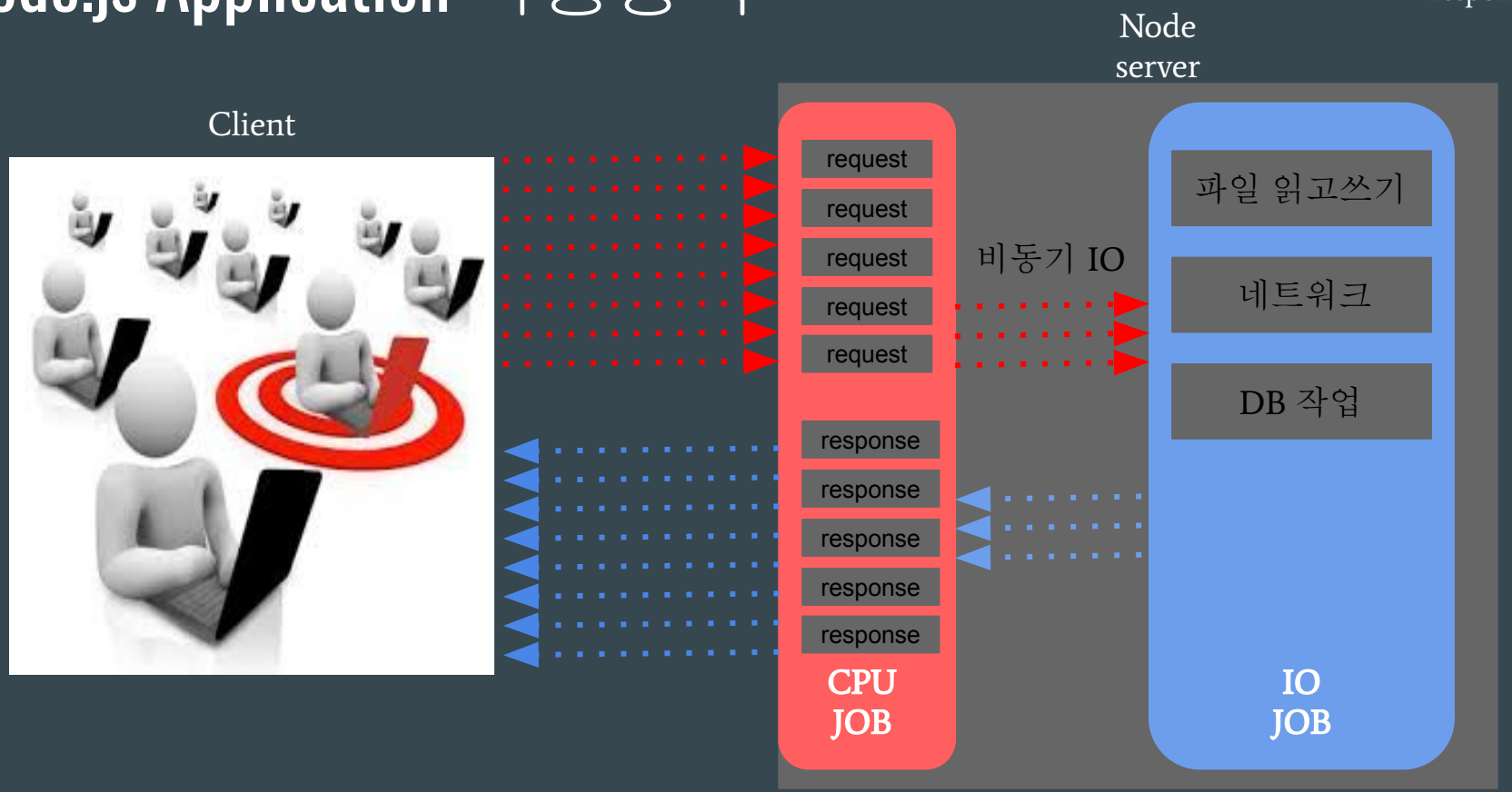
CPU/Thread/Core???



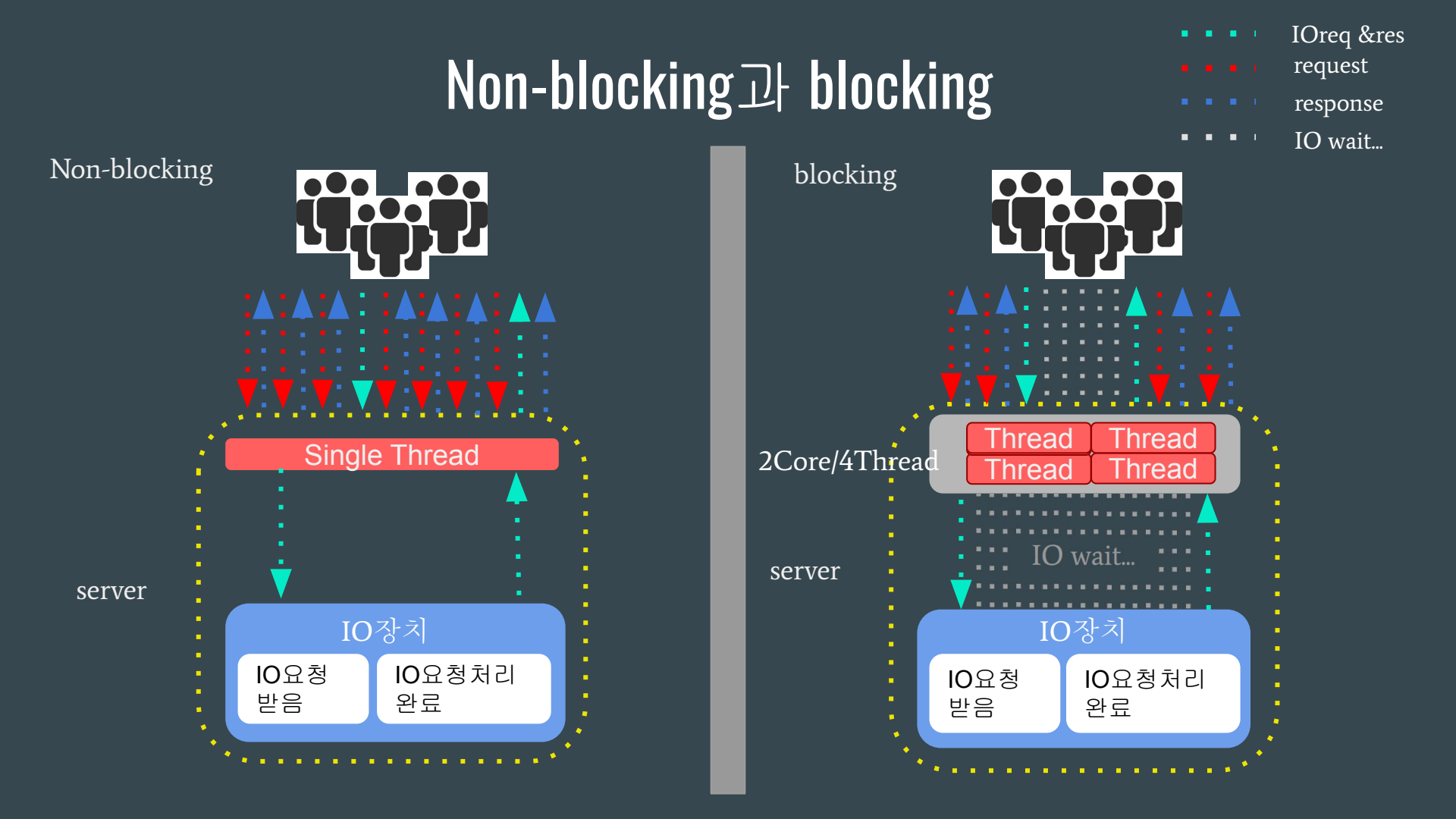
- CPU (=Core)
  - 계산기
  - 컴퓨터를 사용할 때 단일 코어 연산만을 지원하는 소프트웨어를 이용하거나 하나의 작업만 처리한다면, 1코어든 2코어든 차이를 크게 느끼지 못하고, 1코어가 오히려 빠를 수 있음.
  - Node.js Application은 단일 스레드 1코어만 사용한다.
- 클럭
  - 클럭이란 얼마나 빠르게 연산을 처리하느냐를 결정한다.
  - 클럭이 높으면
  - 쿼드코어는 동작이 느린 4개의 연산기를 가지고 작업량을 나누어서 처리
  - 듀얼코어는 동작이 빠른 2개의 연산기를 가지고 작은 작업량을 효율적으로 처리
- 쓰레드
  - 쓰레드는 데이터의 실행 흐름이다. 데이터가 Core를 지나가는 길로 생각하면된다.
  - 쓰레드는 데이터가 송신되고 수신되는 길이다. Core는 쓰레드로부터 들어오는 데이터를 처리해서 쓰레드를 통해서 내보낸다. 한개의 통로로 송수신을 [redacted] 수신용 쓰레드를 둬서 데이터를 송수신하는 것이 작업효율이 [redacted]
  - 2코어 4쓰레드, 4코어 8쓰레드처럼 1코어당 2쓰레드란 개념은 CPU 최적화를 위해 만들어진



Node.js Application 작동방식



Non-blocking과 blocking



비동기/동기

Good 유즈케이스

- 짧은 시간 작업 + 대량 트래픽 환경에 강함
- 하나의 작업에 많은 시간이 소요되는 프로젝트에는 지양.. 유튜브같은 거..

작은 데이터 많은 데이터 처리에 강점... 몽고디비도 이 환경에 특화!

- JSON API
- 페이지가 단 하나인 앱,
  - Gmail이 대표적인 페이지가 단 하나인 앱
  - Ajax가 난무하는 페이지가 된다.
  - 많은 요청 처리, 빠른 응답 그리고 클라이언트와 서버 사이에서 검증하는 코드같은 거도 공유가능.
  - 클라이언트의 한 페이지에서 많은 것을 해야하는 웹앱에 좋다.
- 유닉스 툴을 이용하는 앱
- 스트리밍
  - 파일 실시간 업로드, node.js는 다양한 데이터 레이어를 위한 프록시를 만들 때 매우 좋다.

스프트 리얼타임 어플리케이션

Bad 유즈케이스

- 단일 스레드이므로 CPU 사용이 많은 앱
  - IO 작업이 많은 앱
- 단순 CRUD /HTML 앱
  - Node가 확장성이 좋지만, node.js를 사용했다는 이유로 트래픽 성능이 좋지는 않다.
- NoSQL + Node.js + Buzzword Bullshit

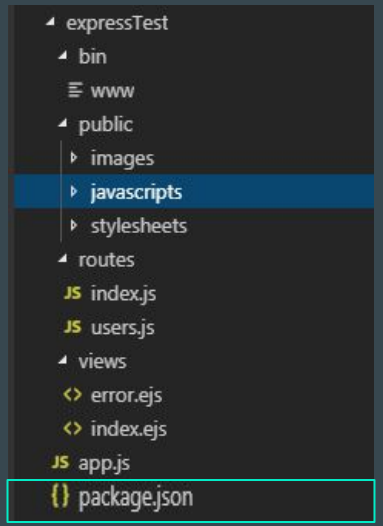
다음 프로젝트에서 NoSQL DB를 사용하려고 준비중이라면 잠깐 멈추고 이 것부터 읽었으면 좋겠다.

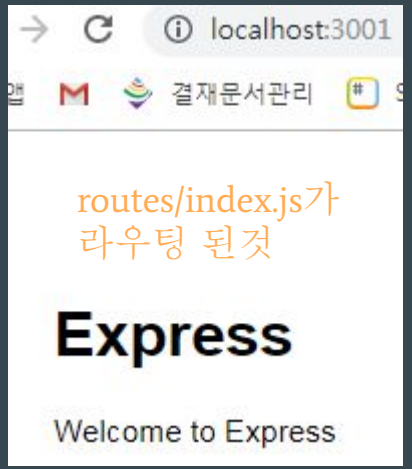
Redis, CouchDB, MongoDB,Riak, Casandra등은 정말 매력적이다. 원래 이브는 빨간 사과를 거절하지 못한다. 지금 node.js를 사용하는 기술적 모험을 감행하고 있다면 더 이상의 모험을 하는 것은 좋지 않다. 아직 완전히 이해하지 못하는 기술들은 서로 위험을 증폭 시킨다.

물론 문서 지향 데이터베이스가 적합한 유즈케이스도 있다. 그러나 지금 만드는 소프트웨어로 사업을 할 것이라면 데이터베이스 기술은 보수적으로 가져가는 것이 좋다. 적어도 덕질*satisfying your inner nerd*보다, 친구들에게 자랑하는 것보다 중요하다.

간단한 express 어플리케이션 만들어보기

```
> npm install -g express-generator
> express --view=ejs [프레임워크 파일 생성명]
> npm install
> node bin/www
http://localhost:3000
```

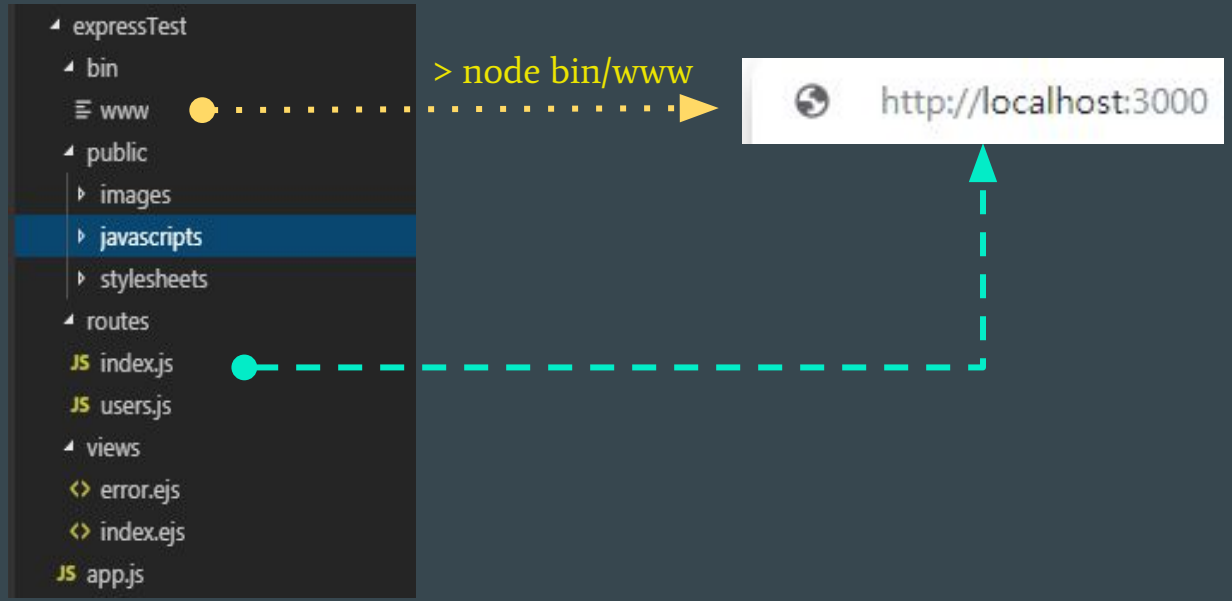




express framework 구동원리

node server start

rendering



bin/www

```
var app = require('../app');
var http = require('http');
/**
 * Get port from environment and store in Express.
 */
var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);
/**
 * Create HTTP server.
 */
var server = http.createServer(app);
```

app.js

```
2  var express = require('express');
7  var indexRouter = require('./routes/index');
8  var usersRouter = require('./routes/users');
10 var app = express();
12 // view engine setup
13 app.set('views', path.join(__dirname, 'views'));
14 app.set('view engine', 'ejs');
15
16 app.use(logger('dev'));
17 app.use(express.json());
18 app.use(express.urlencoded({ extended: false }));
19 app.use(cookieParser());
20 app.use(express.static(path.join(__dirname, 'public')));
21
22 app.use('/', indexRouter);
23 app.use('/users', usersRouter);
```

routes/index.js 랜더링로직

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

랜더링대상 views/index.ejs

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
  </body>
</html>
```

1. 서버로직(app.js) 객체 생성
2. http 모듈 객체 생성
3. 서버로직 객체에 port 할당
4. 서버로직 객체(app)로 서버 생성
5. 서버 작동

npm의 역할

npm(node package manager)

이것은 최고의 의존성 해결과 또한 수많은 빌드 툴체인이 자동화되도록 한다.

npm 자주 사용하는 명령어

끗

vue.js 공식참조문서

https://kr.vuejs.org/v2/guide/index.html

유즈케이스참조

http://pismute.github.io/nodeguide.com/convincing_the_boss.html

다음 주제는 ....

Node.js 핵심 모듈

- Buffer
- Event
- Stream
- Process