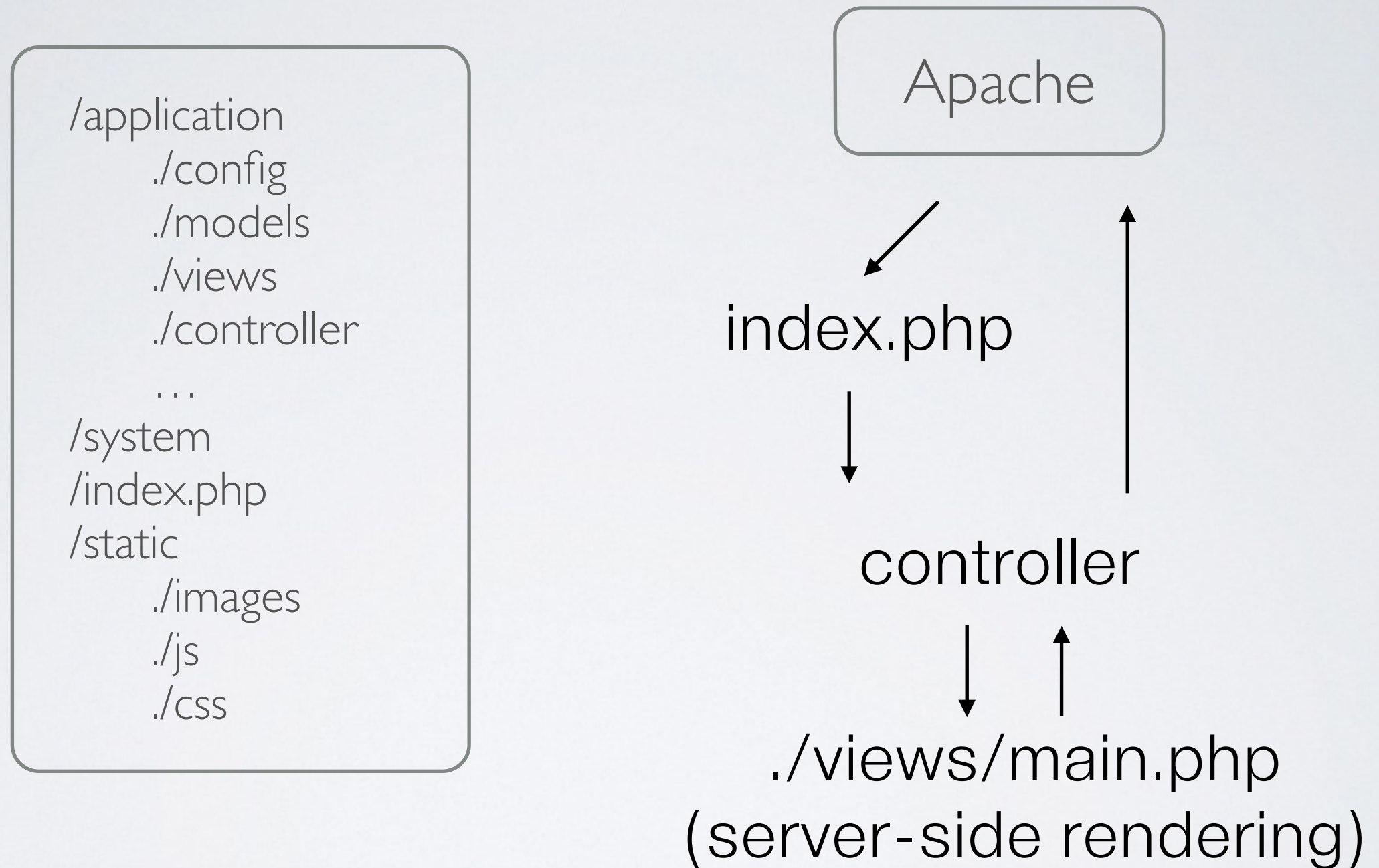




FLASK WEB APPLICATION

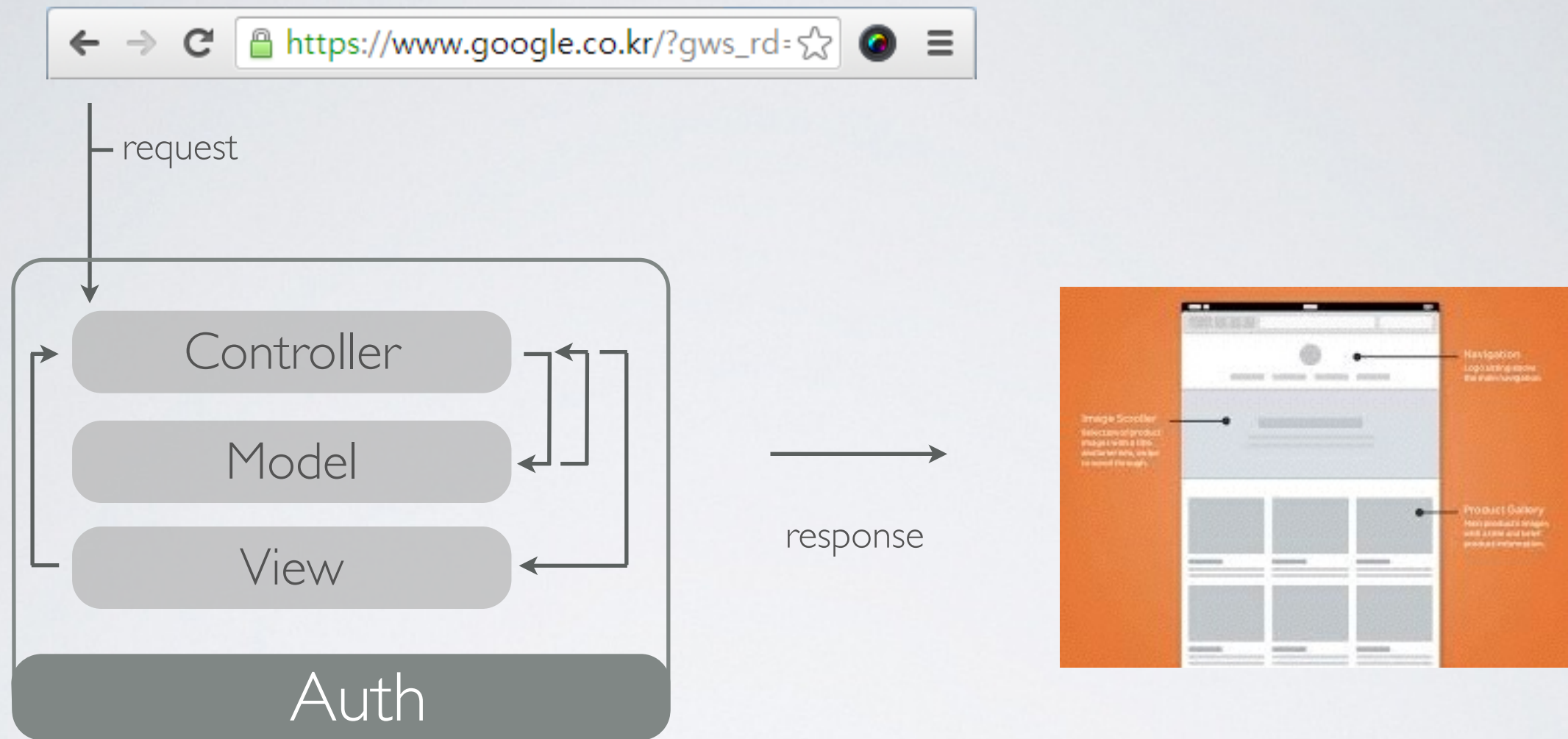
WITH ANGULARJS

도입 전 시스템 구조 (1세대)



폼으로만 통신

flow

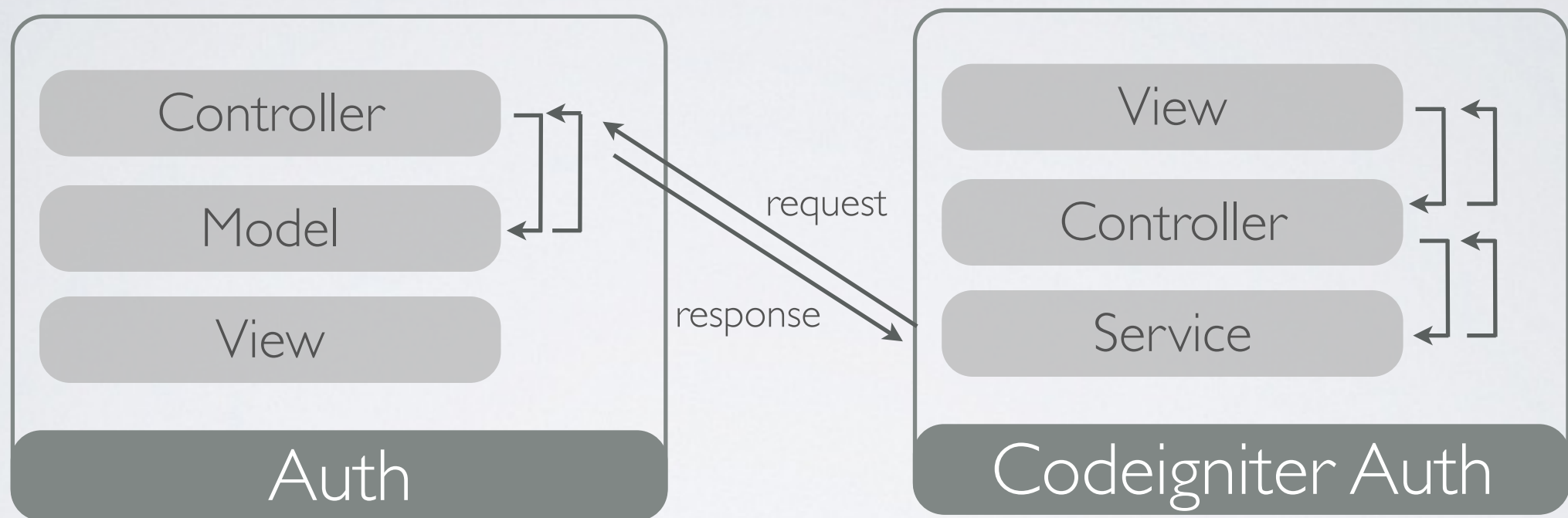


Codeigniter

도입 전 시스템 구조 (2세대)

ajax flow

페이지 리로드 없는 즉각적인 반응 요구로 인해 ajax 작업 증가



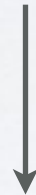
Codeigniter



코드 혼재

도입 전 시스템 구조 (2.5세대)

- `<script src="/main.js"></script>` 호출 방식
php의 controller 내에서 관리
html 내에서 직접 script src 작성
- `<script> alert(1); </script>` inline 방식
`<script>`
 `var date = <? = $date ?>;`
`</script>`
php와 섞어 쓰거나, 섞어쓰지 않더라도 html내에 직접 작성



못참겠다! 스크립트는 스크립트가 관리하게 하자!
requireJS 도입

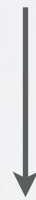
도입 전 시스템 구조 (2.5세대)



- 도입 후 장점
 - 스크립트 관리의 일원화
 - javascript의 캡슐화, 모듈화
- 도입 후 단점
 - 타 팀원들이 작성 방식에 어려움을 느낌
 - 복잡도 증가. 코드 길이 상승

도입 전 시스템 구조 (2.5세대)

- ajax 리턴을 json만 주게 해서 view 변경 시 서버 단 코드 수정을 없게 하자! (디자인 변화 영향을 view 단에 한정)
- view단 client-side 템플릿 엔진 및 javascript MVC 분리하자



BACKBONE.JS 도입

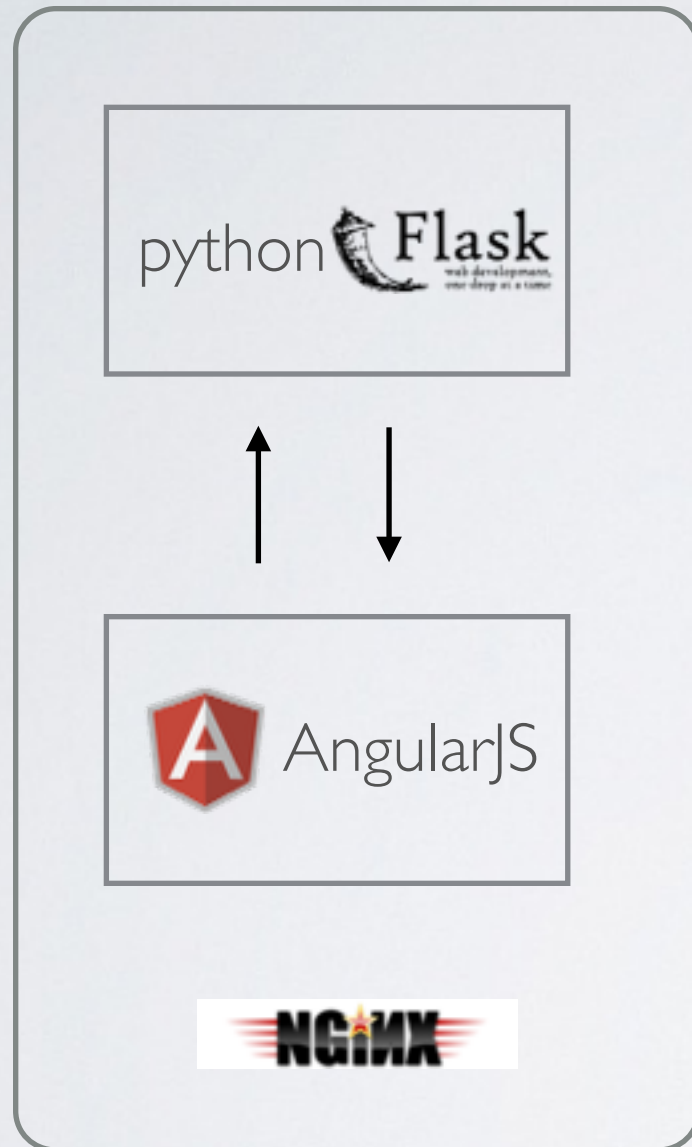
도입 전 시스템 구조 (2.5세대)



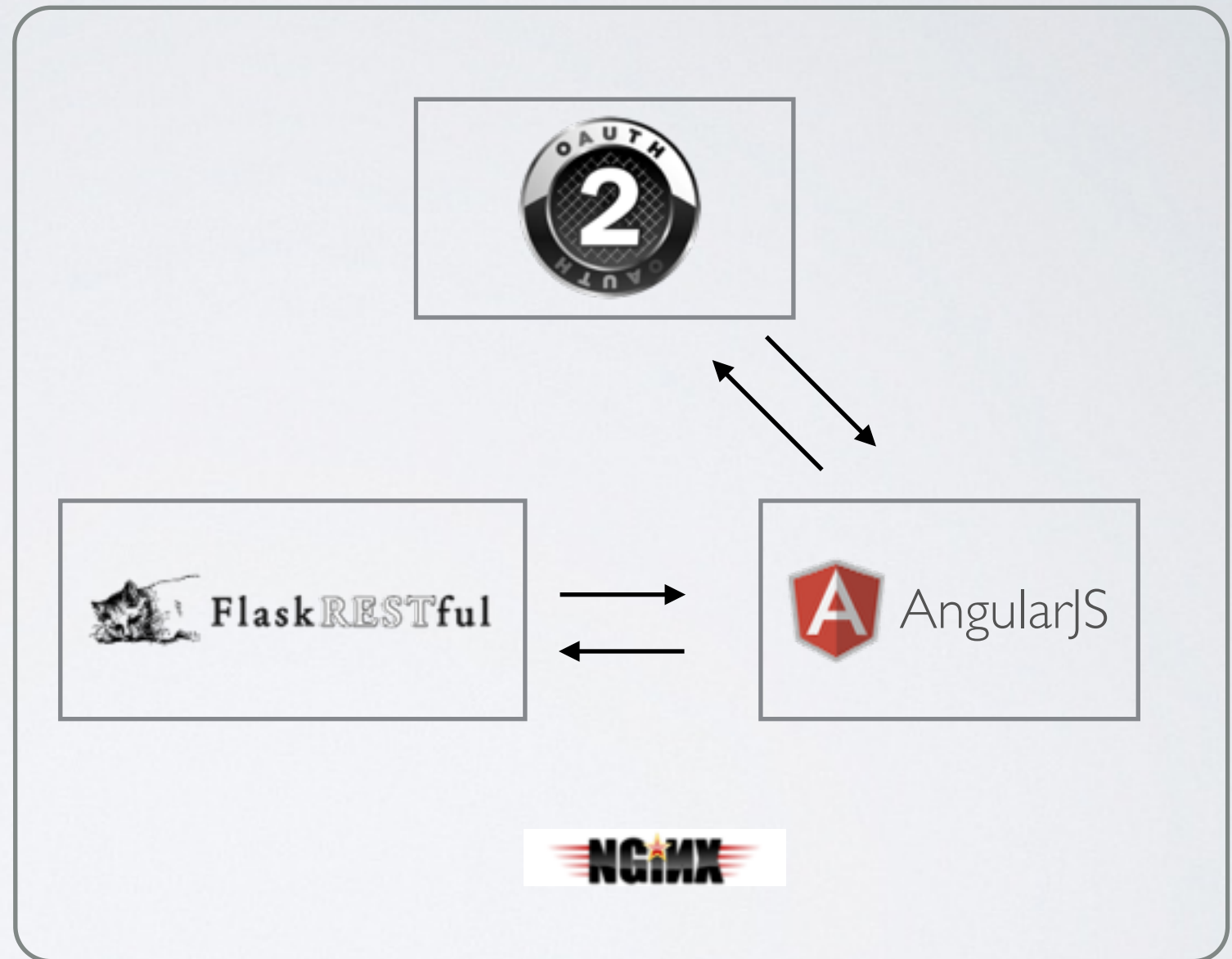
- 도입 후 장점
 - 코드가 어려워 보이는게.. 뭔가 있어보임
 - MVC 분리가 잘됨
- 도입 후 단점
 - 나 이외에는 수정을 못함. 욕먹음
 - 나도 오랫동안 보면 수정하기 힘들

도입 시스템 구조 (3세대)

최초 생각



변경



web application

도입 시스템 구조 (3세대)



Oauth2 도입 배경



- 로그인 통합

- 인증 관련 문제 발생 시 해당 repo만 관리

도입 시스템 구조 (3세대)



FlaskRESTful

- 웹서버는 API만 기재
- stateless
- 간결한 사용법 (<http://flask-restful.readthedocs.org/en/latest/quickstart.html#full-example>)
- 간결한 세팅
 - python 설치
 - pip install flask flask-restful
 - vim test.py -> ctrl+c, ctrl+v -> python test.py
 - 간결한 구조

/app

./config

./models

./resources

./helpers


runserver.py

도입 시스템 구조 (3세대)



FlaskRESTful

- RESTful API는 텀블러, 인스타그램을 참고
- CRUD와 HTTP Method를 완벽하게 매핑

GET /users  resource
 /users/1

POST /users

PUT /users/1

DELETE /users/1

- Nginx 설정 분리

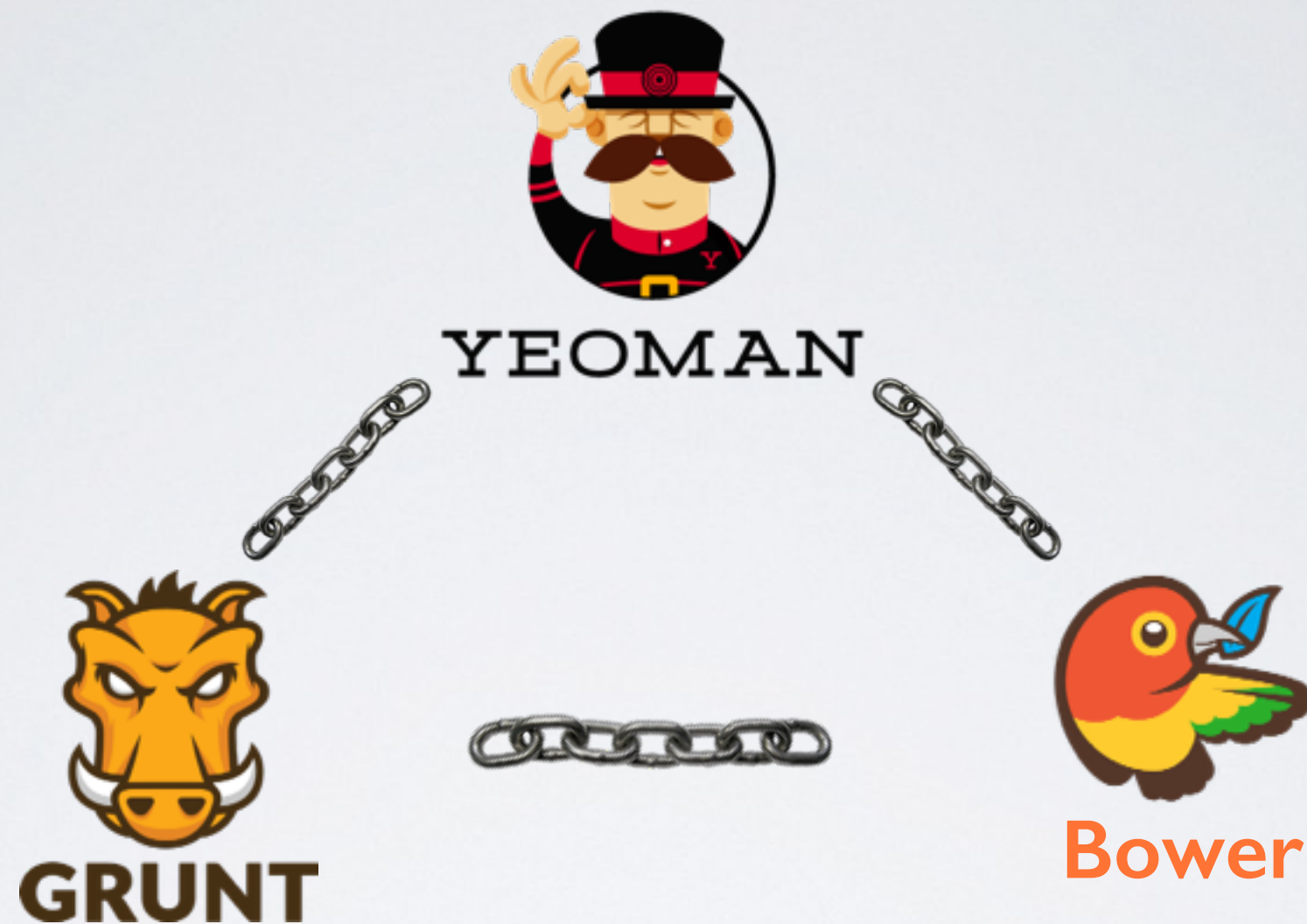
api.conf

```
server {  
    server_name  api.service.com;  
    location / {  
        Access-Control-Allow-Origin;  
        uwsgi_pass unix:/..sock;  
    }  
}
```

front_end.conf

```
server {  
    server_name  service.com www.service.com;  
    location / {  
        root /www/service.com/dist;  
        index index.html;  
    }  
}
```


Nodejs Front-end 환경 도입



- sprite 자동화
- Less (Ruby 환경 도입이 괜찮다면 Sass도..), CoffeeScript 도입 장벽의 하락
- 서버 언어, 프레임워크와 독립되어 언어 변경에도 제약 없음

Nodejs Front-end 환경 도입



Beyond GruntJS => GulpJS
스트리밍을 통한 성능 향상
Grunt에 비해 간결한 설정

Yeoman 팀에서 지원 사격 중

AngularJS 내장 기능 사용법

- controller
데이터 조작과 관련된 화면에 적합
- directive
UI 조작에 특화된 모든 것. UI 모듈
- service
 - service - 싱글톤. 내부에서 this로 외부 노출. 인젝션 시 new로 생성
 - factory - 리빌딩 모듈 패턴으로 리턴. 가장 많이 사용
 - provider - \$get을 갖고 있는 factory. config에서 환경 설정 가능
 - constant - 상수
 - value - 한번 변경이 가능한 변수
 - decorator - service 확장. 3th party 라이브러리 확장 시 유용

경험한 문제점 & 주의점

- Full RESTful API => async => promise로 해결
- Angular의 미래는... 2버전 부터 IE11 이상 지원, ES6 도입, Mobile First...
- 배포에서의 고민. 클라이언트 환경에서 배포할지, 서버 환경에서 build 할지 (모든 상황을 고려해서 서버에서도 build 환경을 만들어 놓는 것이 좋을 듯 클라이언트 배포 및 서버 배포 두가지 다 가능하도록)
- datepicker등 사용 시 client 시간을 믿을 수 없어 시간 가져오기용 api를 만들어야함 -.-
- meta를 구워서 줄 수가 없기 때문에 phantomJS 도입이 필수가 됨...