SSR환경에서의 Micro-Frontend 구현과 퍼포먼스 향상을 위한 캐시전략

박찬진
COUPANG

NAVER DEVIEW 2023

책임의 한계와 법적고지

여기에 포함된 정보는 임직원으로서 저자 본인의 개인적 경험을 바탕으로 한 것으로 쿠팡의 견해나 의견을 나타내는 것이 아님을 밝혀 둡니다. 쿠팡은 여기에 포함된 정보의 적절성이나 공정성, 정확성, 완전성에 대해 확인하지 않았으며 그에 대해 어떠한 진술도 하지 않습니다.

CONTENTS

1. Monolithic Frontend의 한계

1.1 Why Micro-frontend?

NAVER
DEVIEW
2023

마이크로 프론트엔드는 꽤 복잡합니다

어떤 문제가 있었길래 이 기술이 필요했을까?

1.2 Onsite Merchandising System

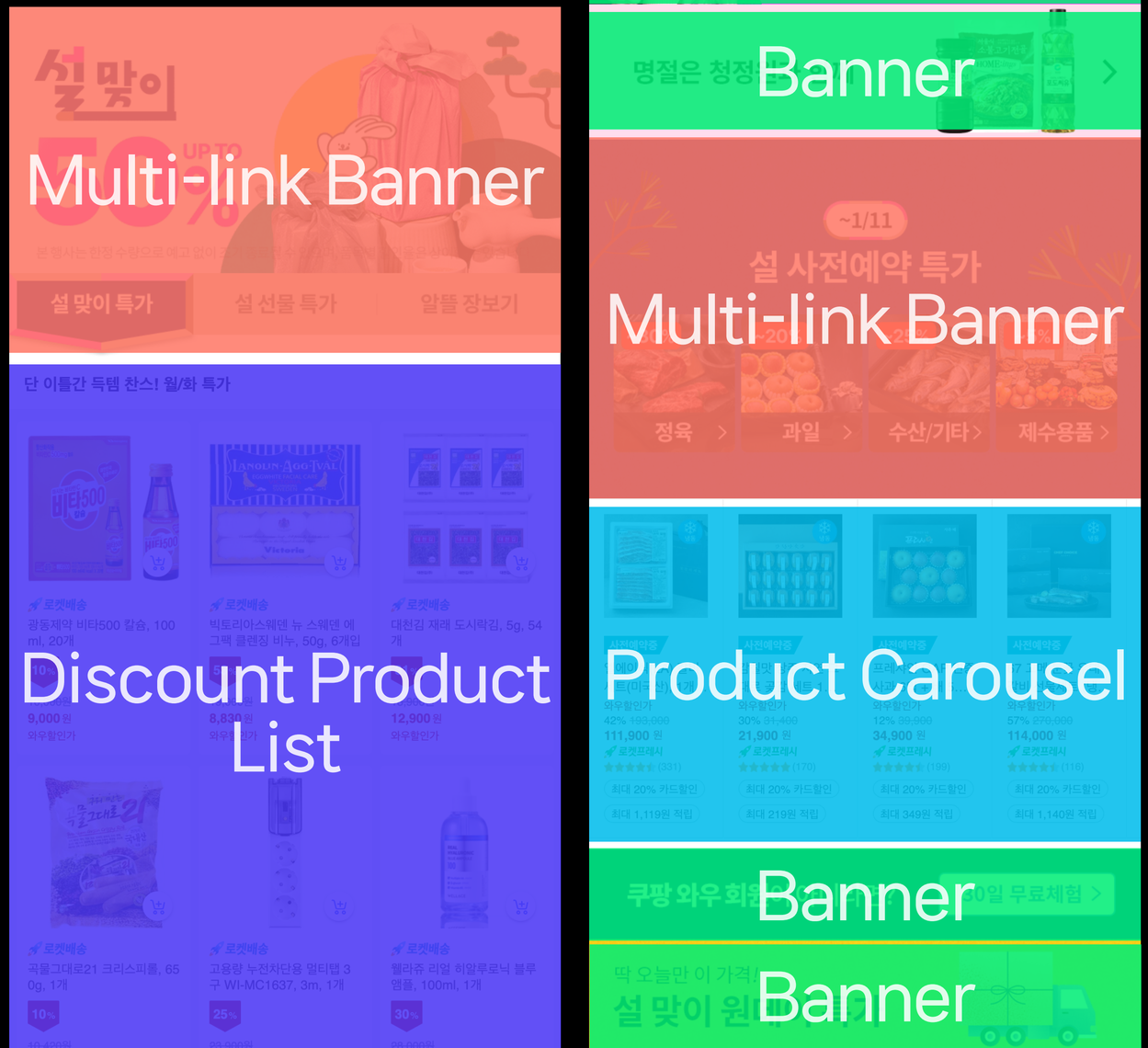
저희 서비스를 소개합니다

광고 및 프로모션 랜딩 페이지 제작 / 관리

- 여러개의 위젯으로 구성
- 각 위젯별로 다양한 옵션
- 간단히 페이지를 생성할 수 있음
- 다양한 목적에 유용함!

↓

사용하고자 하는 팀 증가 = 신규 기능 개발 요구 증가



NAVER DEVIEW 2023

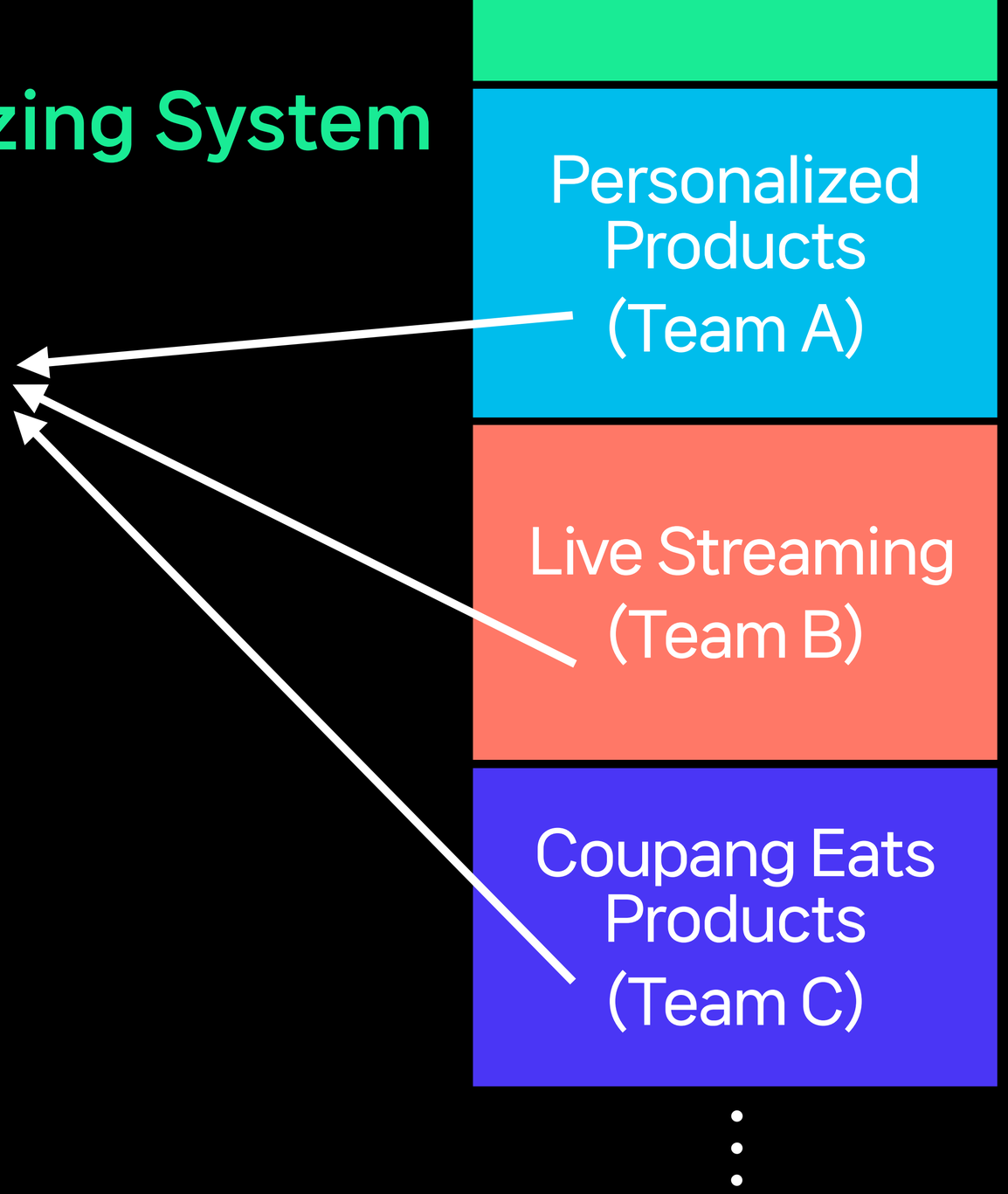
1.2 Onsite Merchandazing System

NAVER
DEVIEW
2023

외부 팀으로부터 신규 위젯 개발 요청

- 관련된 외부팀의 개수가 점점 늘어남
- 각자의 일정 / 각자의 요구사항
- 한정된 팀 리소스

직접 만들어 주세요



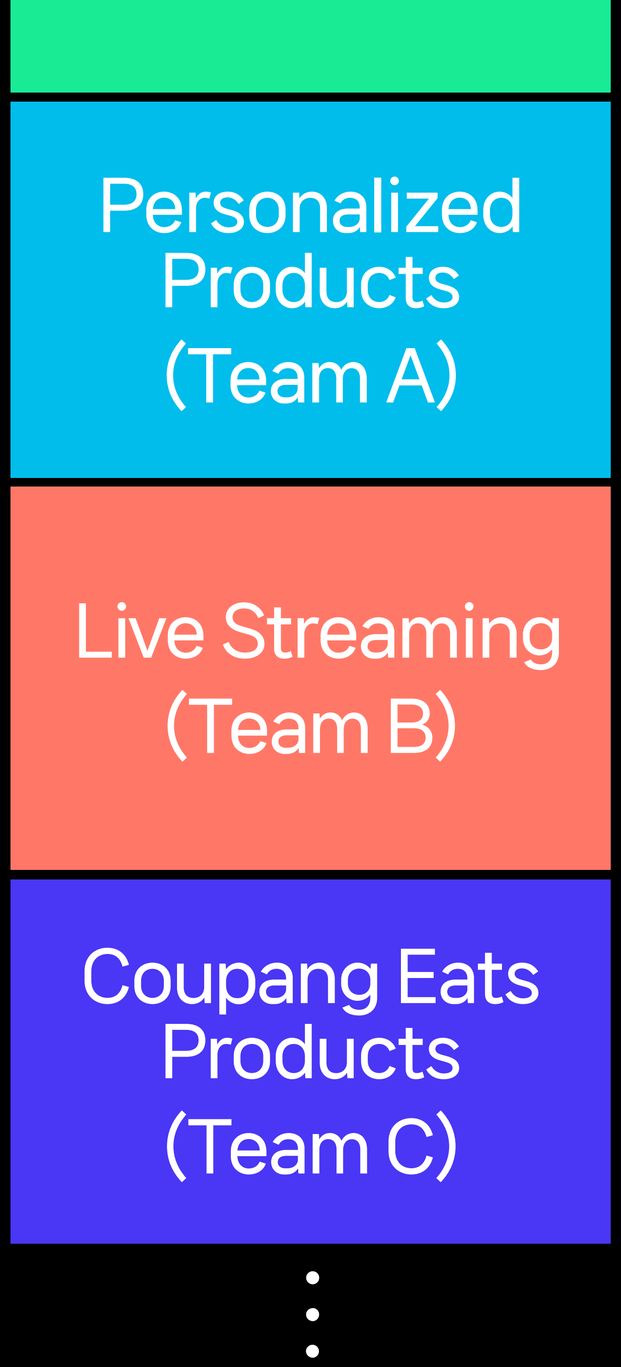
1.2 Onsite Merchandazing System

NAVER
DEVIEW
2023

단일 패키지 서비스 코드에 외부팀이 코드 기여

- 시스템 온보딩 세션 : 전체 시스템 이해 필요
- 트러블 슈팅 : 매우 빈번함
- 코드리뷰, 머지 : 변경 사항 있을 때 마다, 컨플릭 많음
- 테스트, QA : 단일 QA 팀이 모든 배포 대응

× N



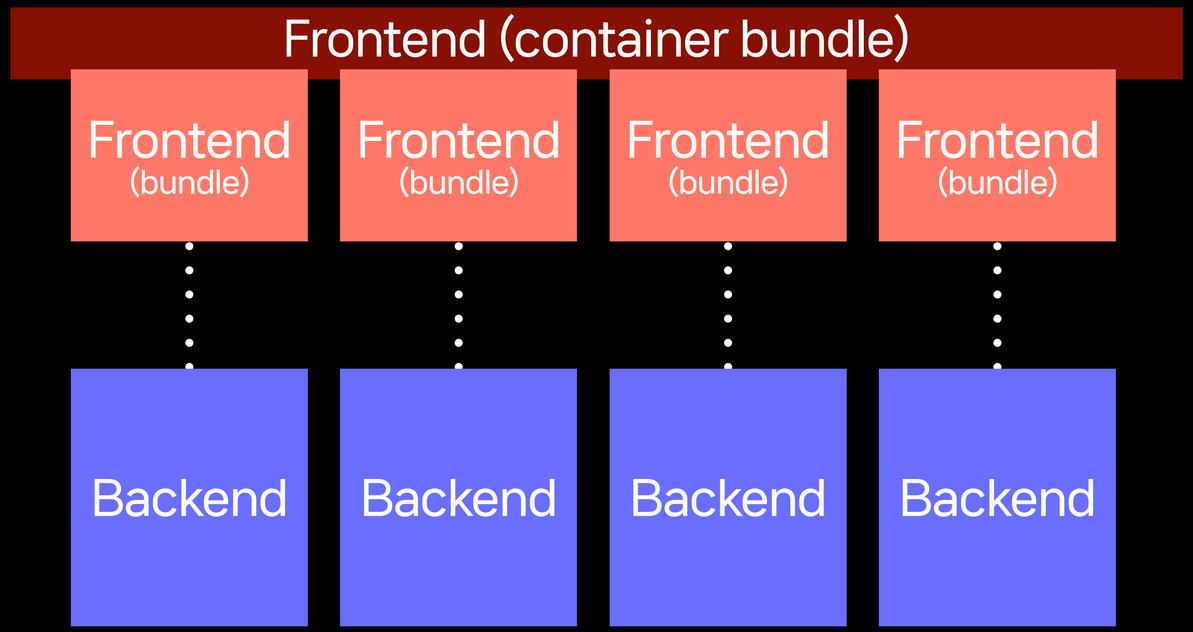
1.3 Content Management Platform

NAVER
DEVIEW
2023

1차 개선사항 목표

Mono-repo

- package화 된 컴포넌트들
- 격리된 코드 구간
- 각 팀이 별도의 repo에서 백엔드 API 운영



- 시스템 온보딩 세션 : 인터페이스 중심으로
- 트러블 슈팅 : 감소
- 코드리뷰, 머지 : 변경 사항 있을 때 마다, 컨플릭 거의 없음
- 테스트, QA : 각 도메인 QA 팀이 유닛 테스트 (+플랫폼 팀 통합테스트)

× N

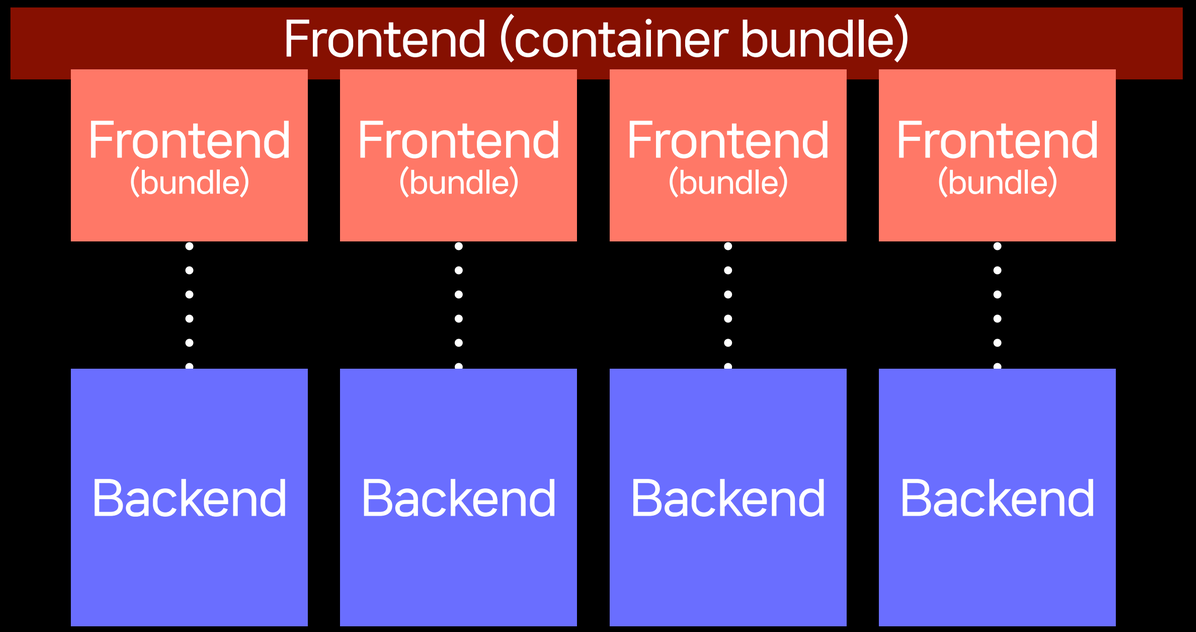
1.3 Content Management Platform

NAVER
DEVIEW
2023

1차 개선사항 목표

배포는?

- 개별 모듈이 추가 및 변경될때마다
  컨테이너의 통합 빌드 및 배포 필요
- QA 및 배포 일정을 팀간에 협의해야 함
- 특정 모듈을 롤백해야 할 때는?
  배포순서나 브랜치 가 꼬이면??
  (A -> B 순으로 배포 후 A에 문제가 생긴 상황)
- 커뮤니케이션 비용이 여전히 높다



1.3 Content Management Platform

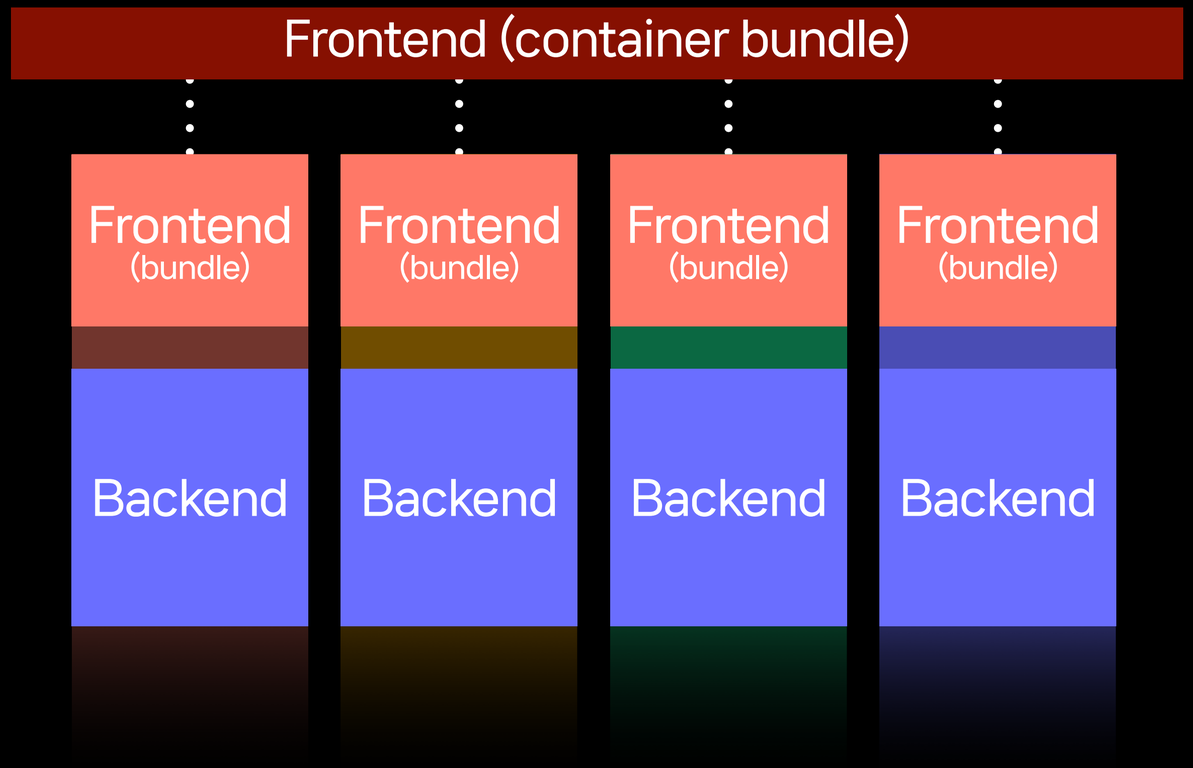
NAVER DEVIEW 2023

2차 개선사항 목표

배포는?

여러 팀 간의 배포 독립성 확보

Micro-Frontend



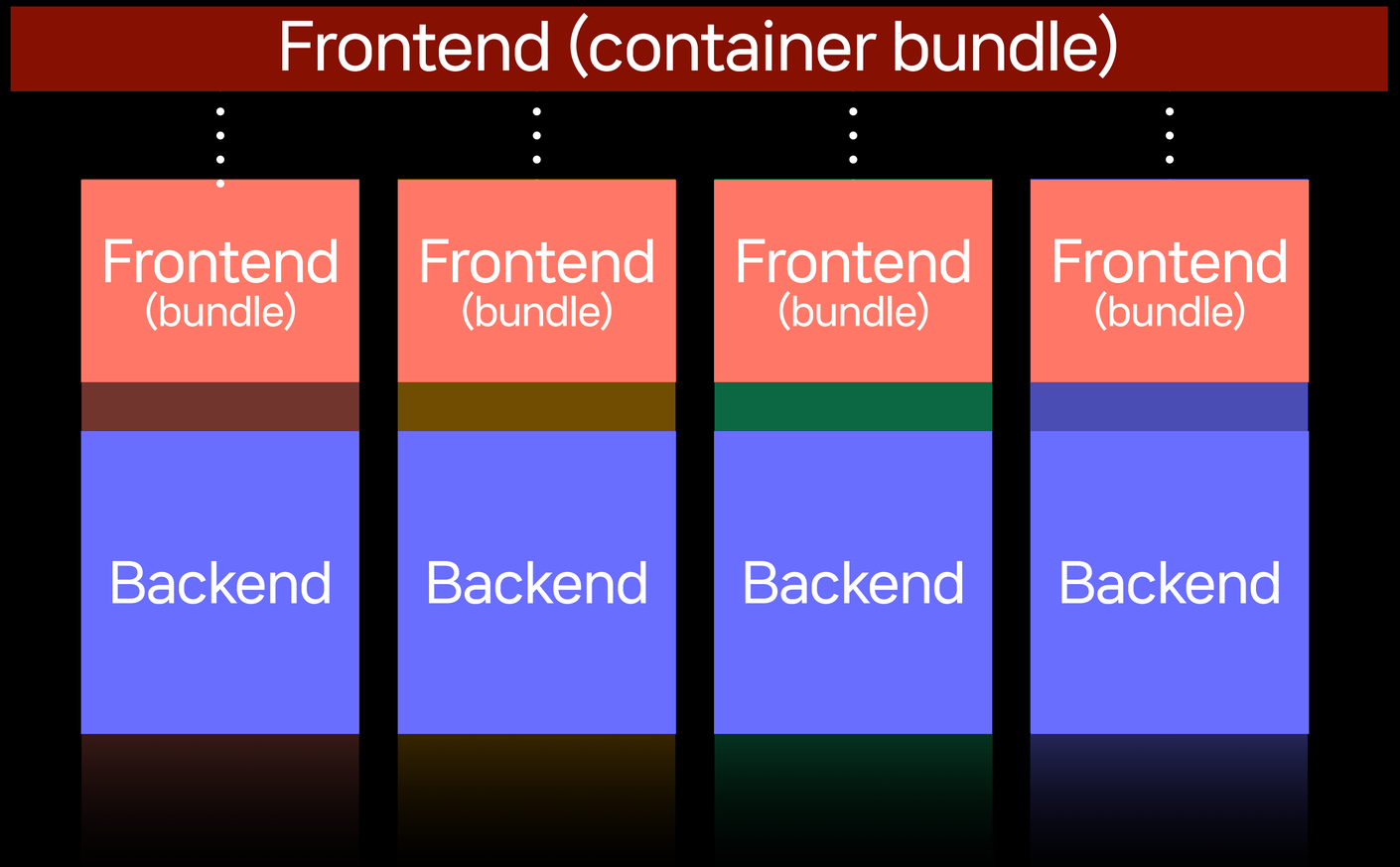
- 각자 의 일정에 스스로 테스트하고 배포
- 플랫폼 팀은 배포나 서버 재시작 필요 없음
- 각 마이크로 서비스별 독립적으로 롤백 가능

2. Micro-Frontend (MFE)

2.1 Micro-Frontend의 기본 구조

NAVER
DEVIEW
2023

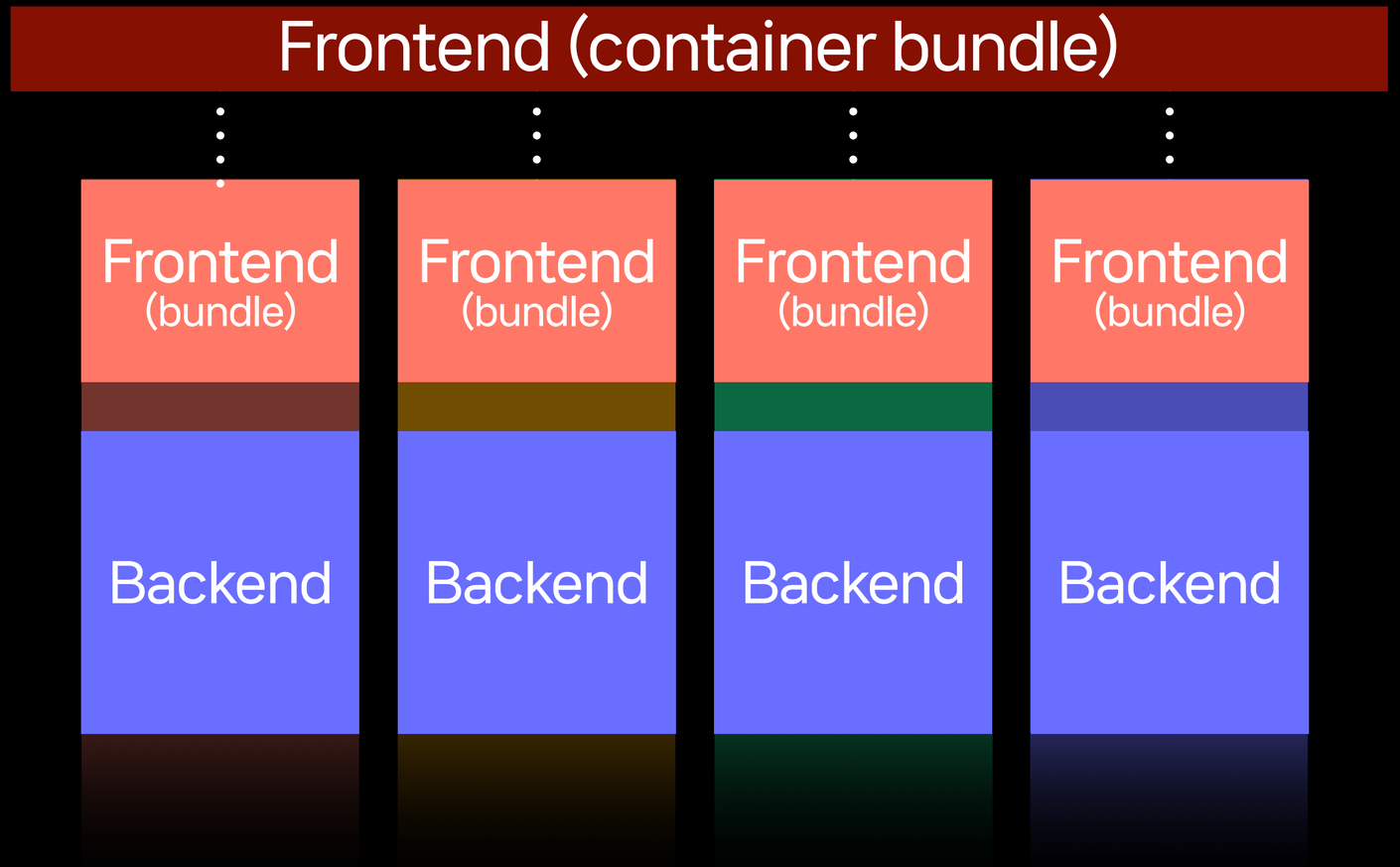
- 컨테이너 중심으로 각 파트를 통합
- 인터페이스를 기반으로 느슨한 결합 (직접 참조 X)
- 각 마이크로 프론트엔드의 번들을 독립적으로 배포



2.1 Micro-Frontend의 기본 구조

NAVER
DEVIEW
2023

핵심은 코드를 통합하는 방법 - **런타임** / **원격** / **동적으로**



2.2 런타임 원격 코드 동적 로딩

NAVER DEVIEW 2023

<이상>

- import로 위젯별 컴포넌트의 원격 번들 파일을 로딩
- 각 번들을 위젯 이름 키로 해서 저장
- 페이지 정보에 나열된 위젯 컴포넌트를 각각 렌더링

```
import Page from 'component/Page';

const componentRegistry = {
  ProductList: await import('https://my-cdn.com/ProductList/bundle.a23fd.js'),
  ProductCarousel: await import('https://my-cdn.com/ProductCarousel.daf3e.js'),
  BannerImage: await import('https://my-cdn.com/BannerImage.l4iuf.js'),
};

const widgets:{ name, data }[]  = await fetch(`/pages/${id}`);

<Page>
  {widgets.map(({name, data}) => <componentRegistry[name] data={widget.data} />)}
</Page>
```

2.2 런타임 원격 코드 동적 로딩

NAVER
DEVIEW
2023

<현실>

1. Can I use ESM??

```
import Page from 'component/Page';

const componentRegistry = {
  ProductList: await import('https://my-cdn.com/ProductList/bundle.a23fd.js'),
  ProductCarousel: await import('https://my-cdn.com/ProductCarousel.daf3e.js'),
  BannerImage: await import('https://my-cdn.com/BannerImage.l4iuf.js'),
};

const widgets:{ name, data }[]  = await fetch(`/pages/${id}`);

<Page>
  {widgets.map(({name, data}) => <componentRegistry[name] data={widget.data} />)}
</Page>
```

3. 번들러에서 remote import가 되나?

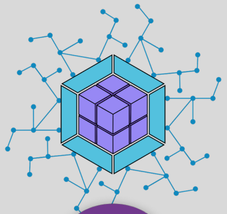
4. 리모트 번들 파일 버전이 계속 바뀔텐데?

2. Vue template / JSX 컴파일은? → 번들러를 사용

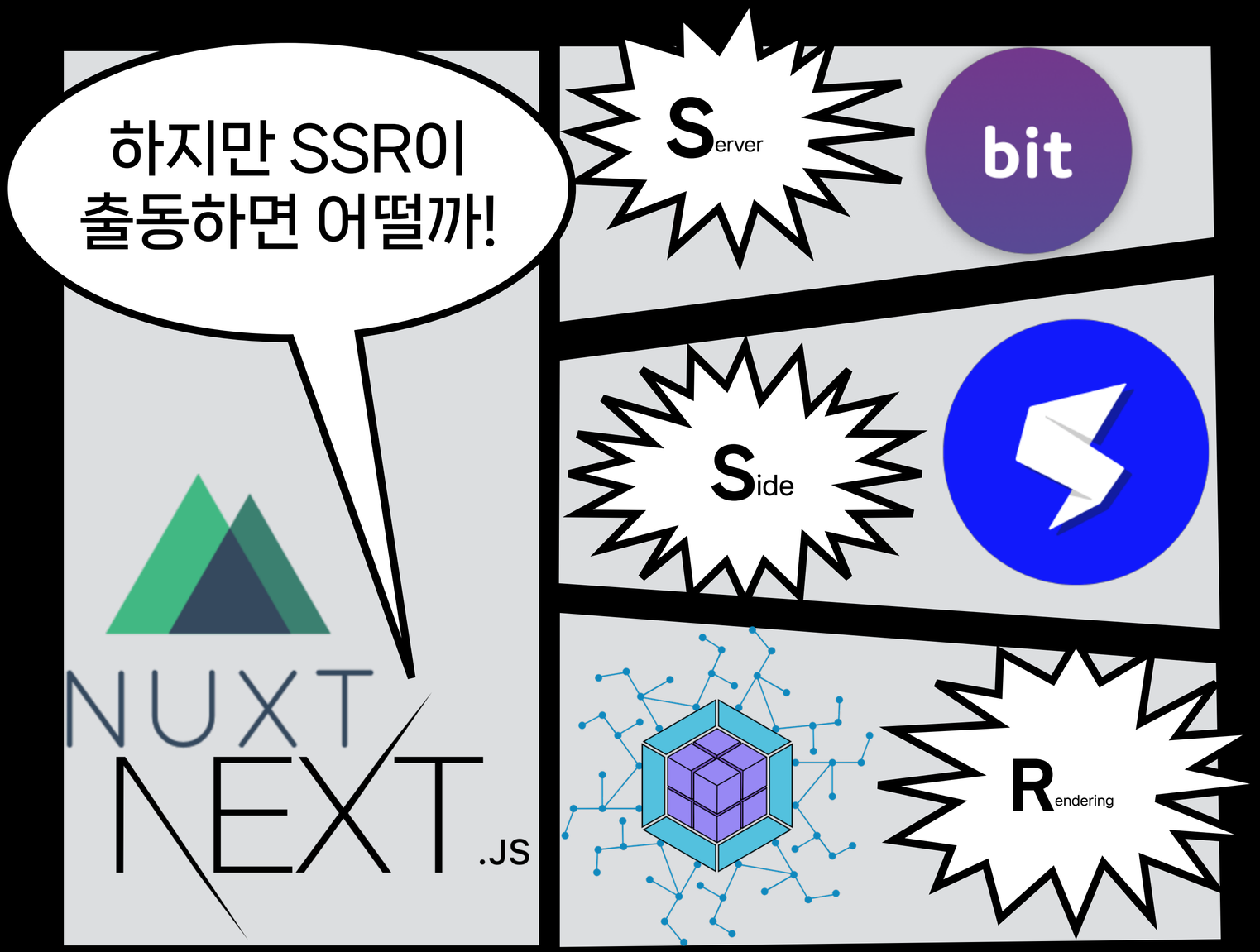
2.2 런타임 원격 코드 동적 로딩

NAVER DEVIEW 2023

- 이런 문제를 해결해주기 위해 다양한 레이어에서 작동하는 라이브러리들이 있음
- 대개 정적인 remote entry를 가지고 있음
- entry에 참조된 remote URL의 manifest 파일 에서 최신 번들 파일의 URL를 다시 참조

Library / Framework	 Single SPA
Based on Builder / Bundler	 **Webpack Module Federation (run-time 통합)**  Bit.dev (build-time 통합)
Based on JS Loader	• SystemJs, RequireJS • `<script src="http://">` + Global Namespace

(※ 위 표는 정확성 보다는 빠른 이해를 돕기 위해 다소 거친 기준으로 구분되었음)


하지만 SSR이
출동하면 어떨까!
NUXT
NEXT.JS
Server
bit
Side
Rendering

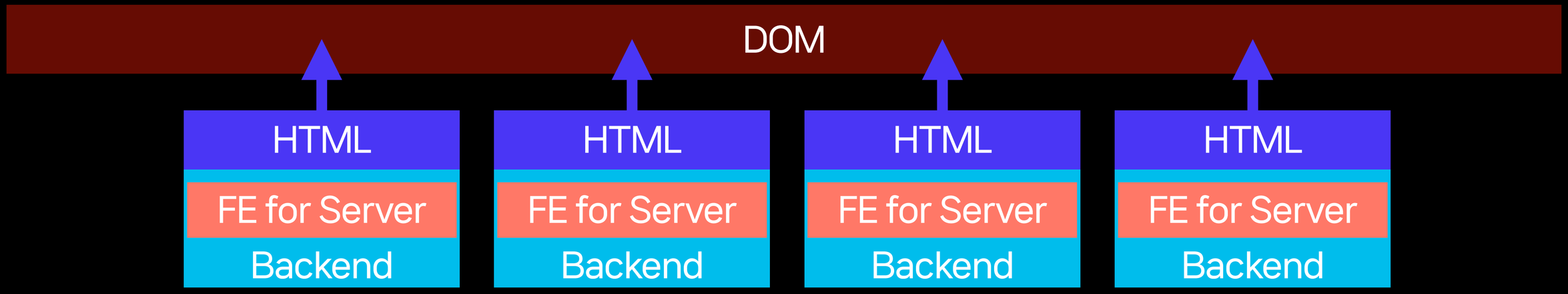
2.4 Module Federation 과 SSR

NAVER
DEVIEW
2023

@module-federation/nextjs-mf

Federated SSR / Dynamic HTML streaming

- Next.js 에서 Federated SSR을 지원
- 여러개의 Monolithic SSR 서버의 렌더링 결과를 런타임에 묶는 방식



2.4 Module Federation 과 SSR

NAVER
DEVIEW
2023

@module-federation/nextjs-mf

Federated SSR / Dynamic HTML streaming

- Paid solution (이 프로젝트를 진행하던 시점 기준)
- 새로운 remote를 동적으로 추가할 수는 없음
- Vue.js는 지원이 불가

정적인 remote entry / manifest URL에서 최신 번들 파일의 URL를 참조 (ref 2.2)

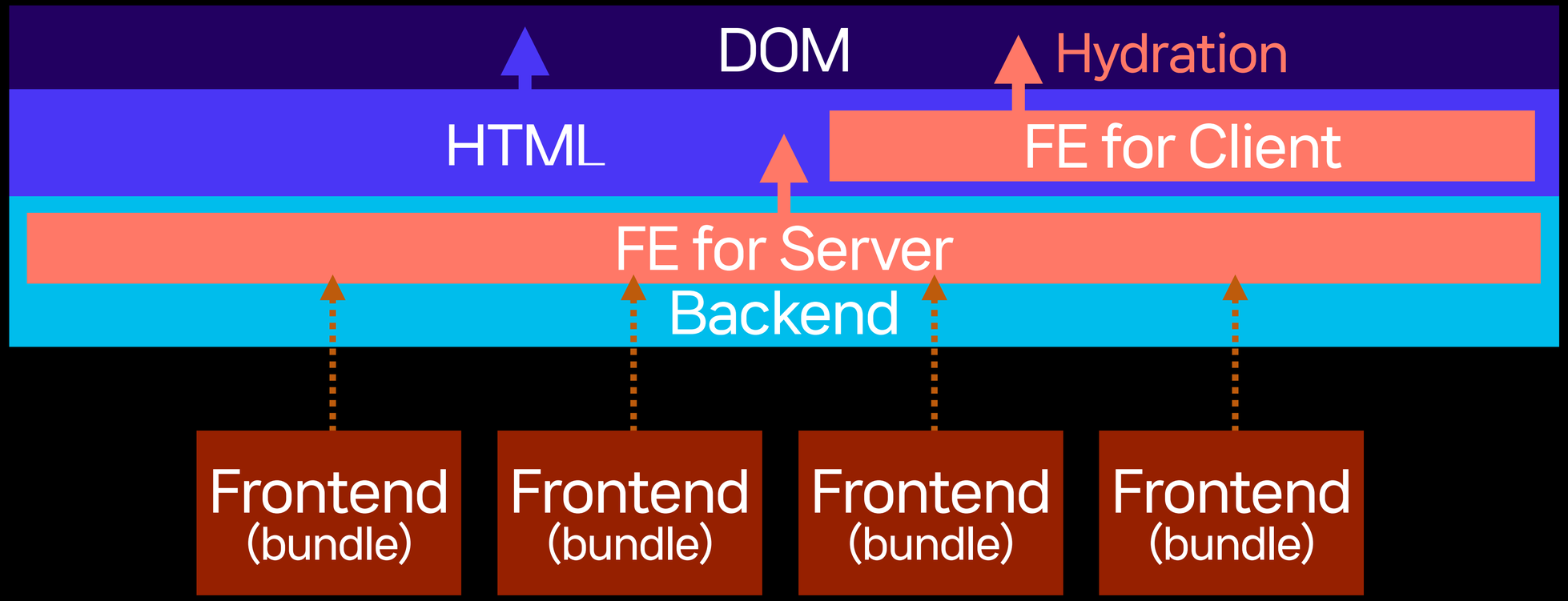
- **각각의 widget 팀마다 서버를 따로 운영해야 함**

우리 서비스와는 잘 맞지 않는다..

2.5 우리가 원했던 방식

NAVER
DEVIEW
2023

각각 따로 빌드된 ~~HTML~~ 컴포넌트 번들을 SSR할때 런타임에 불러온다



단일 Server에서 통합

2.5 우리가 원했던 방식

NAVER
DEVIEW
2023

직접 구현한다면.. 아까 나왔던 문제점들을 해결해야함

Vue template / JSX 컴파일은?

Can I use ESM??

번들러를 사용하면 되지 않을까?

번들러/Node.js에서 remote / dynamic import가 되나?

리모트 번들 파일 버전이 계속 바뀔텐데?

헉..

```
import Page from 'component/Page';

const componentRegistry = {
  ProductList: await import('https://my-cdn.com/ProductList/bundle.a23fd.js'),
  ProductCarousel: await import('https://my-cdn.com/ProductCarousel.daf3e.js'),
  BannerImage: await import('https://my-cdn.com/BannerImage.l4iuf.js'),
};

const widgets:{ name, data }[]  = await fetch(`/pages/${id}`);

<Page>
  {widgets.map(({name, data}) => <componentRegistry[name] data={widget.data} />)}
</Page>
```

2.6 런타임 원격 코드 동적 로딩 (server)

NAVER
DEVIEW
2023

Node.JS에서 remote import가 되나?

- Download 받아서 local로 만든다.

번들러에서 dynamic import가 되나?

- 번들러는 빌드타임에 import문과 dependency들을 미리 트랜스파일링
- 가능한 경우에대해 미리 대비 해둘 뿐, 완전한 dynamic import가 어렵다
- Eval 로 import문의 transpile을 캔슬하고 저수준에서 직접 처리한다.
  import 자체만 eval, import 대상 코드를 eval하는 것이 아님

```
const { filePath } = (await download('https://.../my-module.1.mjs');
const component = (await eval(`import('${filePath}')`)).default;
```

string

2.6 런타임 원격 코드 동적 로딩 (client)

NAVER DEVIEW 2023

- Can I use ESM??
- CommonJs + 브라우저용 모듈 지원 라이브러리를 사용한다.
  - SystemJS
  - @paciolan/remote-module-loader

```
import * as Vue from "vue"
import createLoadRemoteModule, { createRequires } from "@paciolan/remote-module-loader"

const requires = createRequires({ vue: Vue })
const loadRemoteModule = createLoadRemoteModule({ requires })
const component = await loadRemoteModule('https://cdn.com/MyWidget/bundle.asdf234.js')
```

CommonJs Build

2.6 런타임 원격 코드 동적 로딩

NAVER DEVIEW 2023

```
const widgets: { name, data }[] = await fetch(`/pages/${id}`);
const modules = await fetch(`/modules`);

const components = widgets.map(({ name, data }) => {
  const {
    version, // aw3ref
    server,  // './.bundles/aw3ref.mjs'
    client,  // 'https://cdn.com/.../aw3ref.mjs'
    css
  } = modules[name];
  return {
    data,
    css,
    Component: isServer
      ? (await eval(`import('${server}')`)).default
      : await loadRemoteModule(client)
  };
});

<Page>
  {components.map(({ data, css, Component }) => <>
    <link rel="stylesheet" href={css} />
    <Component data={data} />
  </>}
</Page>
```

- 다소 요약된 버전
- 실제 코드는 각 프레임웍 구조에 나누어서 Plugin이나 Module형태로적용

nodejs에서 remote import가 되나? Download 받아서 local로 만든다.

번들러에서 dynamic import가 되나? Eval 로 import문의 transpile을 캔슬하고 저수준에서 직접 처리한다.

Can I use ESM?? CommonJs + 브라우저용 모듈 지원 라이브러리를 사용한다.

2.7 남은 문제

```
                asdr234.js
                w3fbddf.js
                qd3trwe.js
                df44wf2.js
const {         alfie73.js
  version, // aw3ref
  server,  // './.bundles/aw3ref.mjs'
  client,  // 'https://cdn.com/.../aw3ref.mjs'
  css
} = modules[name];
```

리모트 번들 파일 버전이 계속 바뀔텐데?

다음 장에서..

3. Module Registry

리모트 번들 파일 관리 시스템

3.1 Module Registry

모듈 개발/배포 프로세스의 탈중앙화

모든 것을 알아서 하게 둬도 될까?

3.1 Module Registry

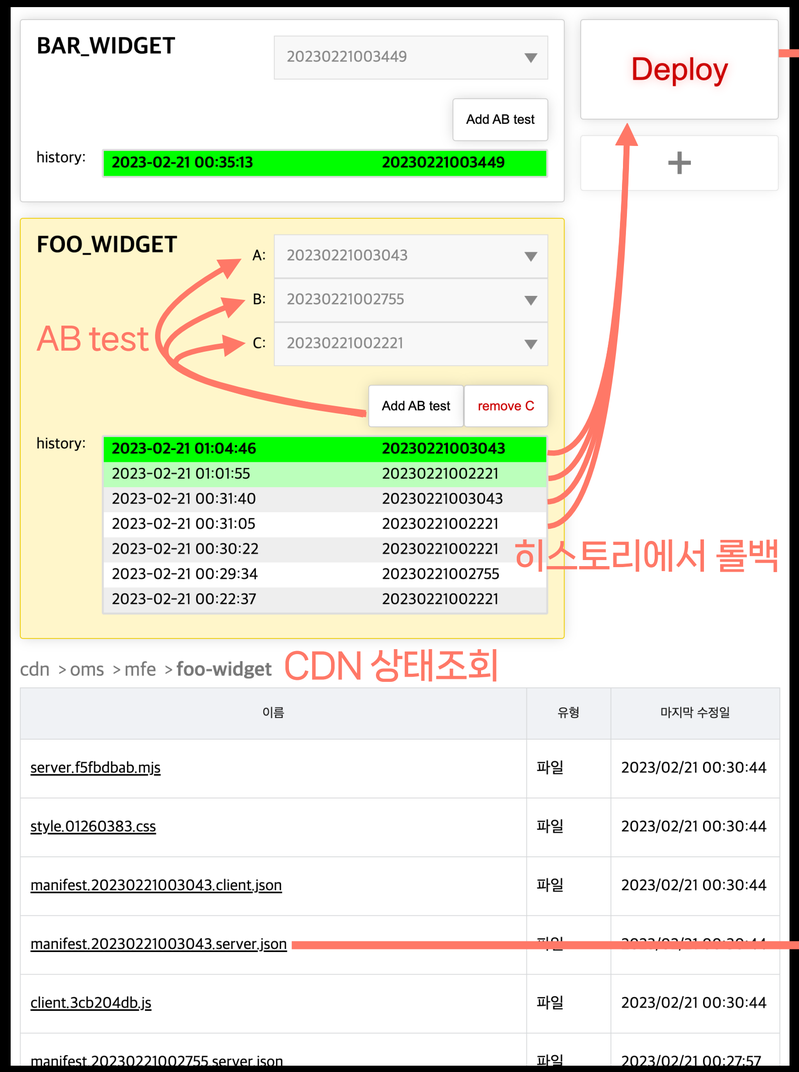
모듈 개발/배포 프로세스는 탈중앙화

플랫폼 관리 기능은 중앙에서 컨트롤

- **각 모듈의 현재 버전에 대한 참조**
- 버전 변경 이벤트를 컨테이너 서버에 실시간으로 발행
- 배포 히스토리 및 메타데이터를 관리
- UI를 통한 손쉬운 배포 및 롤백
- AB test 프레임웍과 통합하여 test 및 점진적 배포 지원
- CDN 빌드 상태 조회

3.2 Demo - UI / Data flow

Module Registry



Deploy

+

히스토리에서 롤백

Redis Pub/Sub

다중 서버로 배포 이벤트 발송

Container


```
{
  "src/index.vue": {
    "file": "server.f5fbdbab.mjs",
    "src": "src/index.vue",
    "isEntry": true,
    "css": [
      "index.13b18b8b.css"
    ]
  }
}
```

manifest.20230221003043.server.json

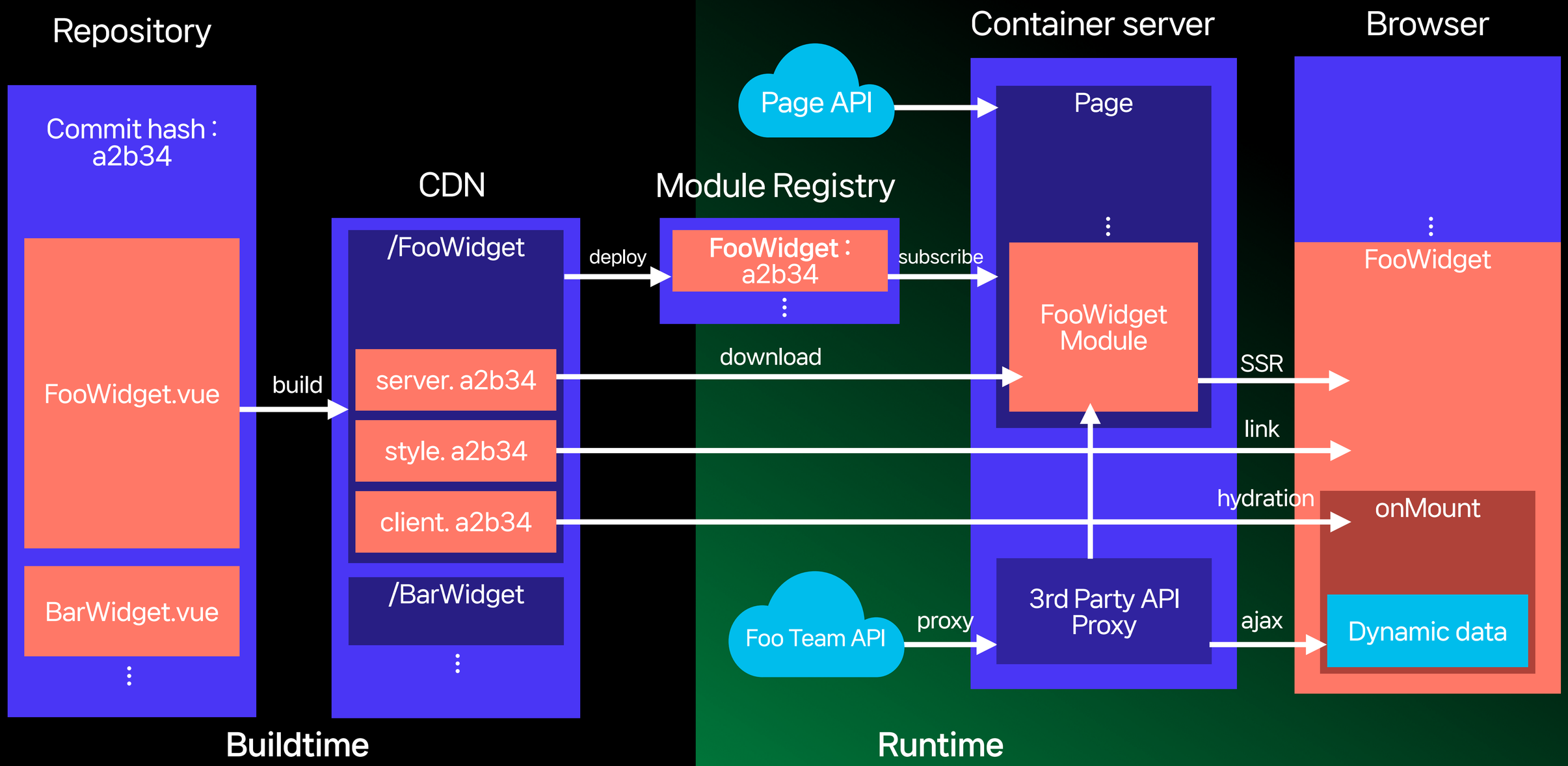
선택된 버전의 manifest 파일에서 컴포넌트 번들 참조

페이지 구성 데이터

```
const widgets =[
  {
    id: 1,
    mfeType: 'BAR_WIDGET',
    config: {
      title: 'Bar1',
    },
  },
  {
    id: 2,
    mfeType: 'BAR_WIDGET',
    config: {
      title: 'Bar2',
    },
  },
  {
    id: 3,
    mfeType: 'FOO_WIDGET',
    config: {
      title: "…"
    }
  },
];
```

3.3 Rendering Walk-through

NAVER DEVIEW 2023



4. MFE를 위한 캐시전략

4.1 무엇을 캐시할 것인가?

NAVER
DEVIEW
2023

모듈별 독립배포

모듈별 AB Test

개인화된 자동 페이지 구성

4.1 무엇을 캐시할 것인가?

NAVER
DEVIEW
2023

N개의 모듈을 AB 테스트 하고자 할 때 증가되는 캐시의 사이즈가

2^n → n

전체 페이지 캐시 → 마이크로 파트별 캐시

모듈별 독립배포

모듈별 AB Test

개인화된 자동 페이지 구성

4.2 캐시 구현 - 라이브러리 통합 (vue 기준)

NAVER
DEVIEW
2023

Nuxt2 (Vue2) Component Level HTML Cache

https://v2.ssr.vuejs.org/guide/caching.html#component-level-caching

Nuxt3 (Vue3) 에서는?

nuxt-multi-cache https://nuxt-multi-cache.dulnan.net

또는 직접 구현

```
export default defineNuxtConfig({
  nitro: {
    alias: {
      "vue/server-renderer": resolve(__dirname, './component-caching-renderer.ts') // nitro alias 를 자체 구현으로 대체
    }
  }, ...
})

// component-caching-renderer.ts
import {renderToString as _renderToString} from '@vue/server-renderer'

export const renderToString = (...args) => {
  // 여기에 캐시 전략 decorating
  return _renderToString(...args);
}
```

4.2 캐시 구현 - key 구성

NAVER
DEVIEW
2023

- 어떤 방식의 구현 (또는 라이브러리)이든 컴포넌트 별로 캐시 키 설계가 필요하다
- 컨테이너 / 각 모듈이 새로 배포되었을 때 기존 캐시를 invalidation 하는 부분이 중요함

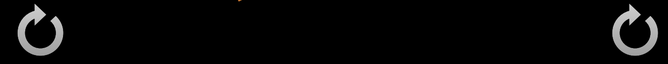
```
serverCacheKey : (…) => [
                   type, // fooWidet - 위젯 종류
                   dataId,  // 해당 위젯 렌더링 데이터 hash or UUID
                   moduleVersion, // 모듈 빌드 버전
                   containerVersion,  // 컨테이너 빌드 버전
                   … // 기타 필요한 키들 (device, language, …)
                 ].join(':')
```

4.3 스트레스 테스트

NAVER
DEVIEW
2023

한번에 많은 수의 캐시가 생성되는 경우?

```
[type, moduleVersion, containerVersion, partId].join(':')
```



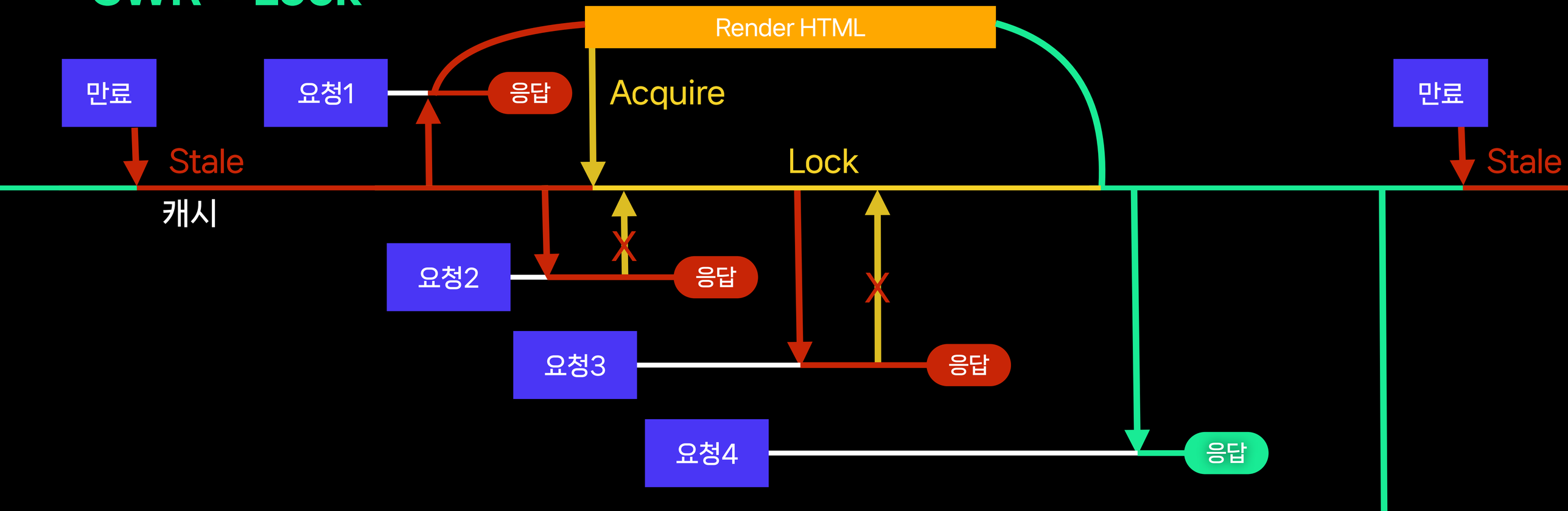
- 새 버전 배포가 일어날 때
- 새 모듈이 배포되어도 모듈별로 캐시하고 있기 때문에
  각 모듈 타입에 대해서만 캐시를 생성하면 됨
- 컨테이너 배포시 현재 요청되고 있는 unique 활성 페이지 수만큼 캐시 리프레시가 일어난다
- 이를 고려해서 충분한 트래픽을 감당할 수 있도록 CPU 자원을 모니터링하며 테스트해야 함

4.4 동시성 문제

NAVER
DEVIEW
2023

- 특정 페이지를 생성하면 캐시를 먼저 만들고 캐시로부터 서빙 시작
- 캐시가 만료되어 리프레시하는 와중에도 대량의 트래픽이 몰린다면?

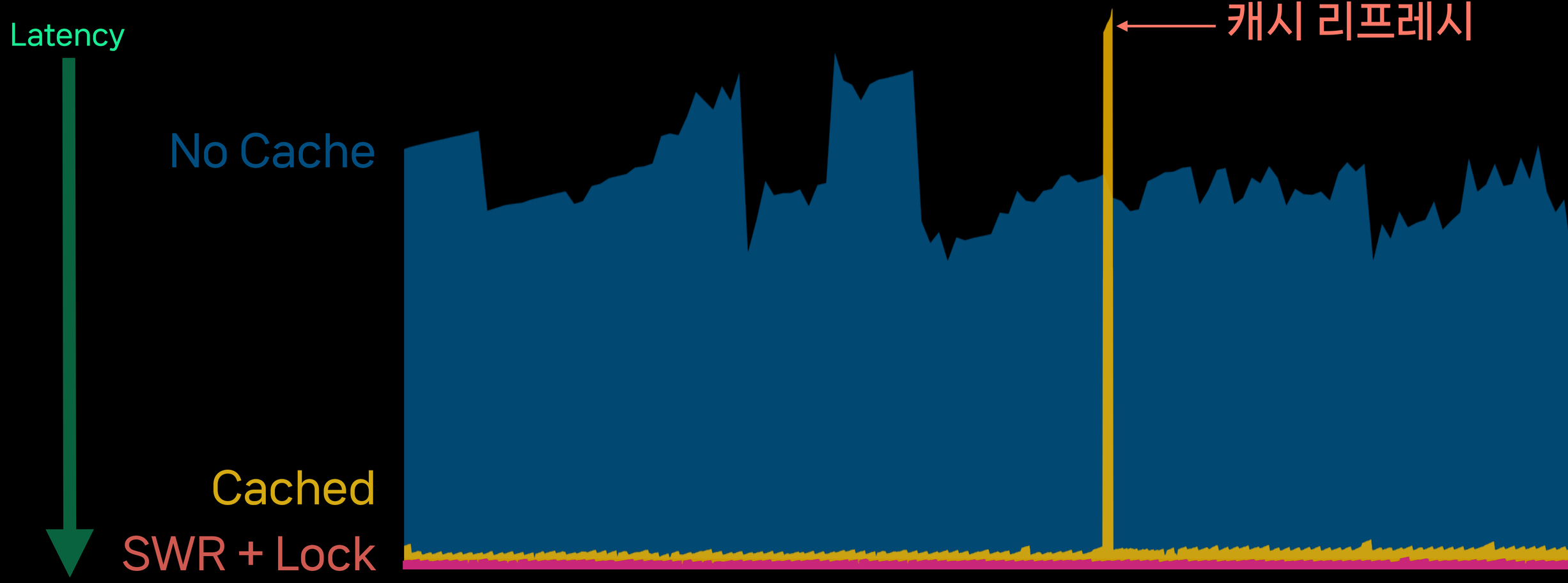
SWR + Lock



4.4 동시성 문제

NAVER
DEVIEW
2023

트래픽이 많은 경우, SWR + Lock 메커니즘이 적용 안되었을 때
레이턴시가 회복되기까지 매우 오래 걸릴 수 있음



4.5 Storage

NAVER
DEVIEW
2023

in-memory LRU cache Or Redis

- 캐시를 저장할 공간은 필요한 성능과 서버 구성 등의 상황을 고려해서 선택
  - 키 갯수
  - 키당 사이즈
  - 여러 서버 인스턴스 사이의 캐시 공유범위
  - 네트워크 레이턴시

4.5 Storage

NAVER
DEVIEW
2023

in-memory LRU cache
(1차)

+

Redis
(2차)

현재 인기있는 페이지에 대해서
더 빠르다, Redis call count 감소

대량의 캐시를 운영할 수 있다 →
SWR available

5. 결론 / 회고

5.1 이 변화를 통해 얻은 것

NAVER
DEVIEW
2023

- n개의 팀이 참여해 n개의 위젯을 개발할 때 코드, 개발, QA, 빌드, 배포의 복잡도가
  기하급수적으로 증가한다 → 증가하지 않는다 (n/n) → 전체 개발 퍼포먼스 향상

- 하나의 모듈을 롤백할 때 함께 통합되거나 이후에 배포된 다른 모듈에
  영향을 줄 수 있다 → 독립적이다

- 각 모듈에서 버그 발생시 롤백 속도
  빌드 시간 + 배포시간 → 1초 미만

- N개의 모듈을 AB 테스트 하고자 할 때 증가되는 캐시의 사이즈가
  $2^n \rightarrow n$ 에 비례

- 서비스가 더 다이내믹 해졌지만, 전체 페이지를 캐시하던 버전에 비해 레이턴시가
  오히려 향상됨 (LRU 1차 캐시 도입 + 추가적인 SSR 최적화)

5.2 결론

NAVER
DEVIEW
2023

- MFE는 시스템 복잡성을 높인다. 이득이 있는지 따져보고 적용한다

- SSR에서의 MFE는 서비스 특성에 따라 필요한 구현방식이 다양할 수 있으므로
  특정 라이브러리를 통한 일반적인 해결보다는 요구사항에 최적화된 설계가 필요할 수 있다

- 하지만 서비스가 가진 문제의 특수성에 따라서 신중하게 설계하고 도입한다면
  확실한 이점을 얻을 수 있다

- MFE에서 HTML 캐시는 마이크로 서비스별로 이루어지는 것이 좋다
- 캐시 키 로직을 신중하게 결정해야 하며 동시성 문제를 조심스럽게 다루어야 한다

5.3 회고

NAVER
DEVIEW
2023

- **개발 경험이 월등히 나아졌다** : 프로젝트 초기에 하나의 외부 모듈을 이전 방식과 새로운 방식으로 두번 구현해 비교한 결과, 처음 구현시보다 확연히 빠르게 온보딩 및 개발 가능했고 이후 유지관리도 팀간에 최소한의 간섭으로 독립적으로 이루어짐

- **여전히 팀간의 소통은 중요하다** : 외부팀 동료들이 쉽게 온보딩하고 운영할 수 있도록 개발 경험에 대한 피드백을 경청하고 개발 환경과 프로세스를 지속적으로 자동화하며 꾸준히 개선해야함

Q & A

Thank You