# Loading 40 MB of JSON on initial load

Lukáš Huvar

# Just don’t do it

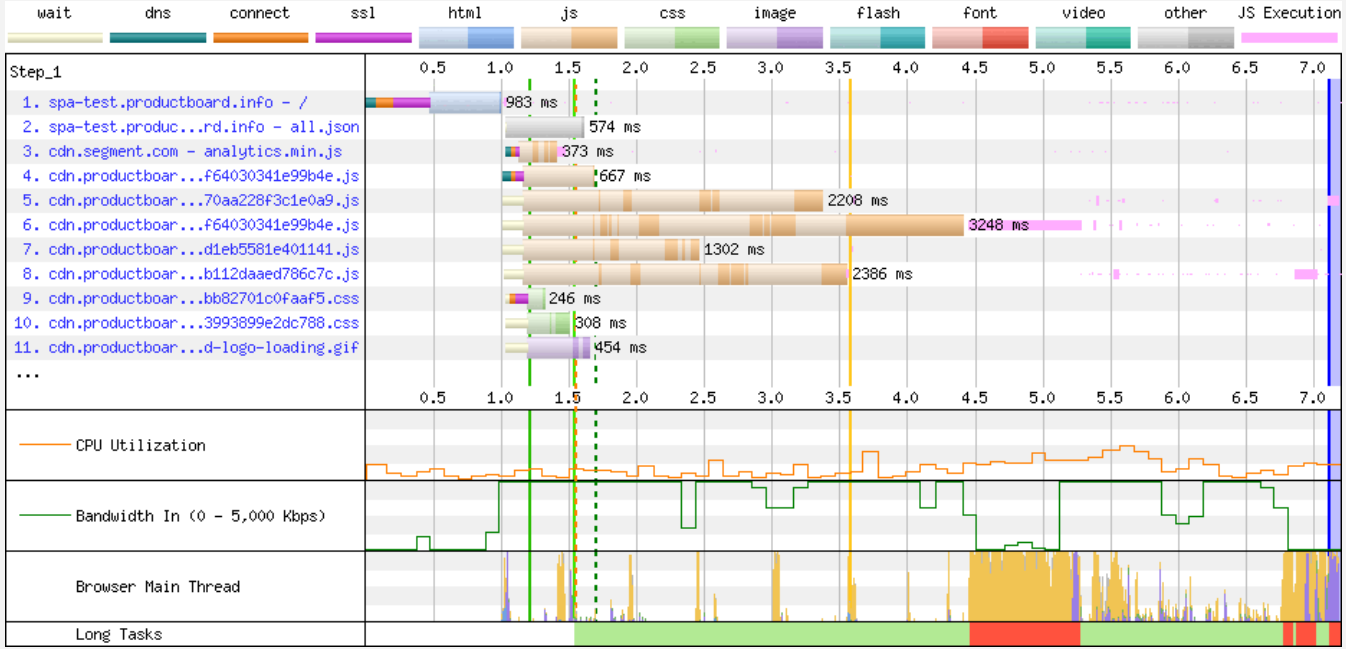
# Lukáš Huvar

**Software engineer** productboard

huv1k huv1k https://huvik.dev/

Previously Vercel Prisma

# Frontend performance at Productboard



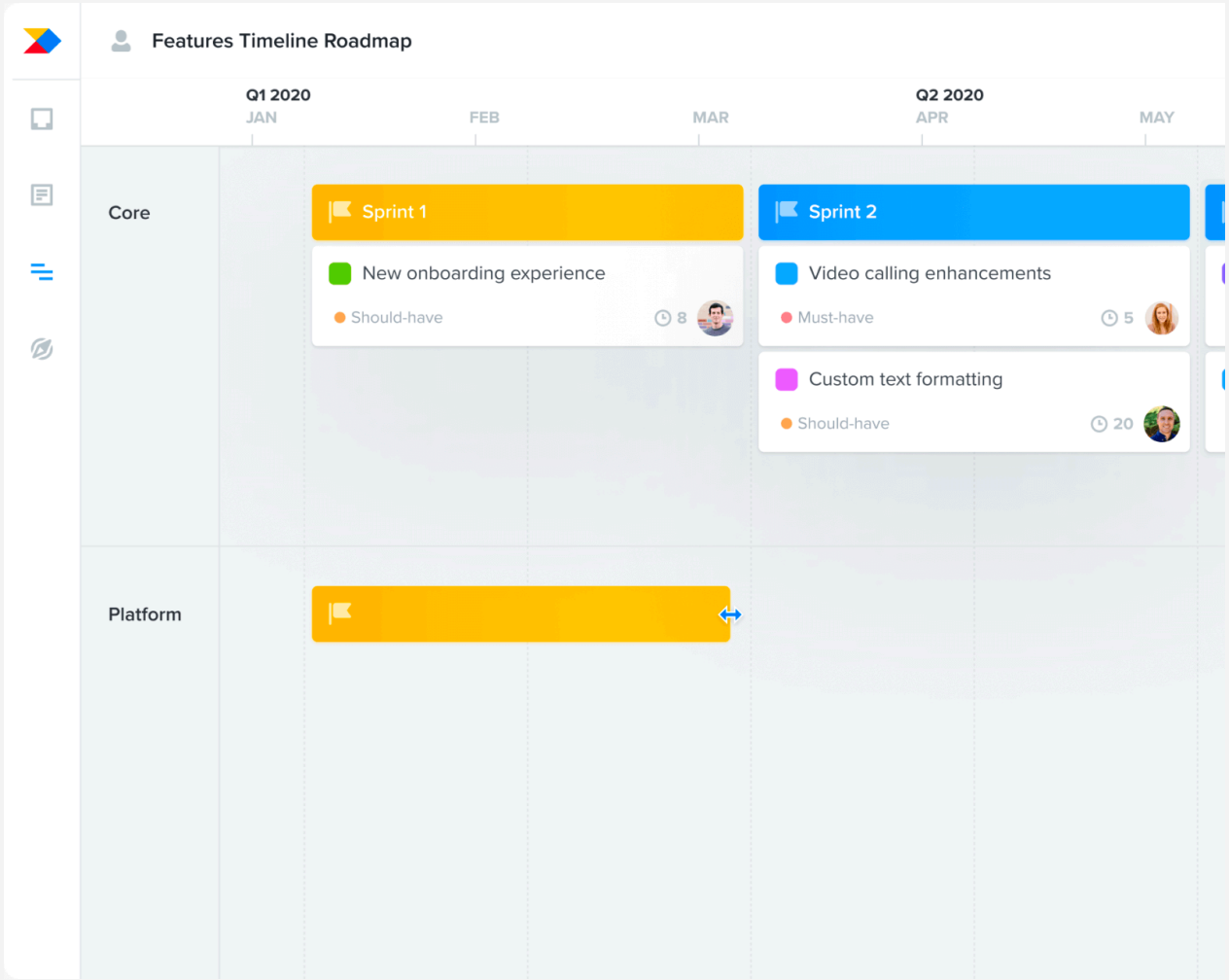
# Agenda

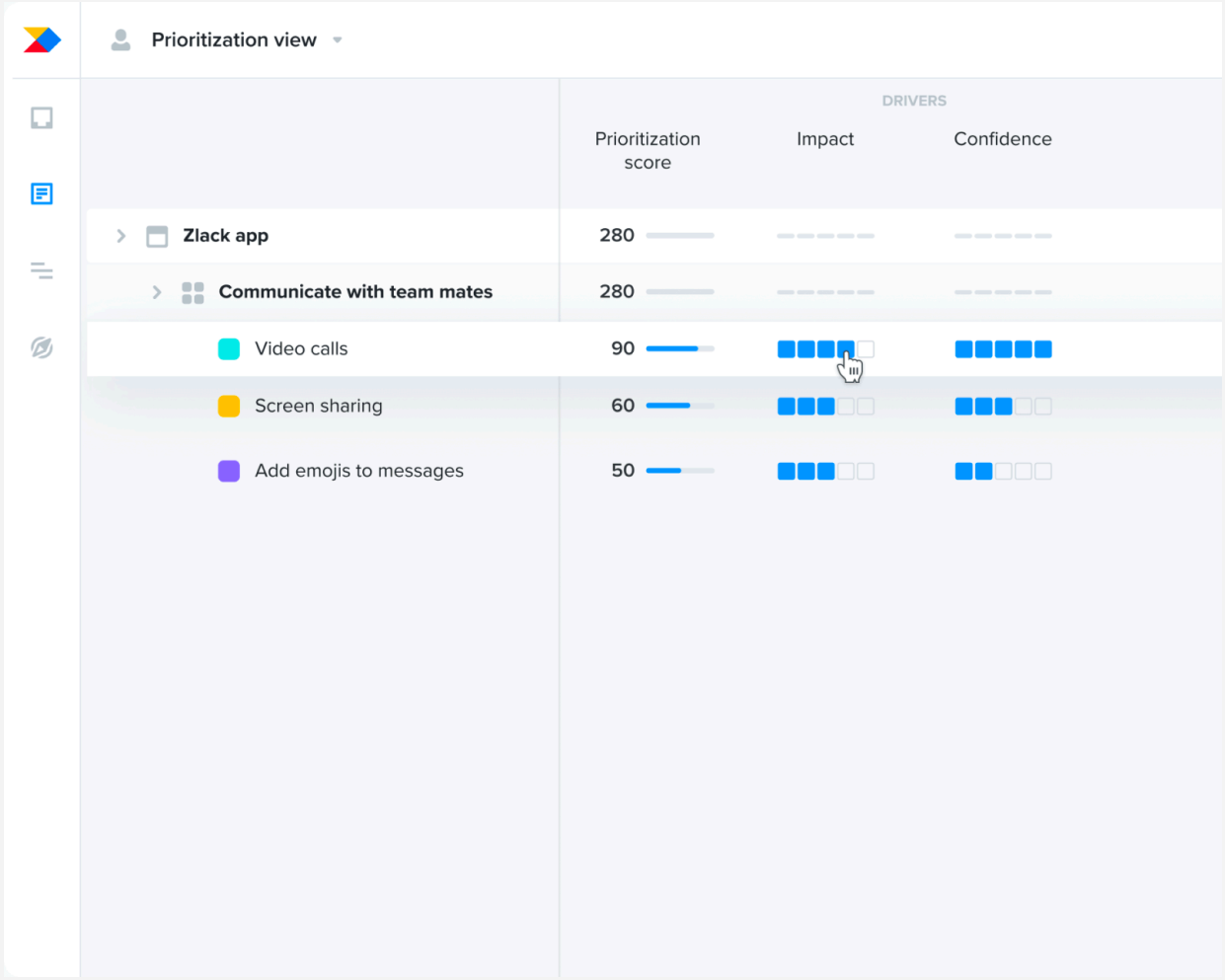
- What is Productboard ❓
- Data fetching evolution 🚀
- Squeezing the lemon 🍋

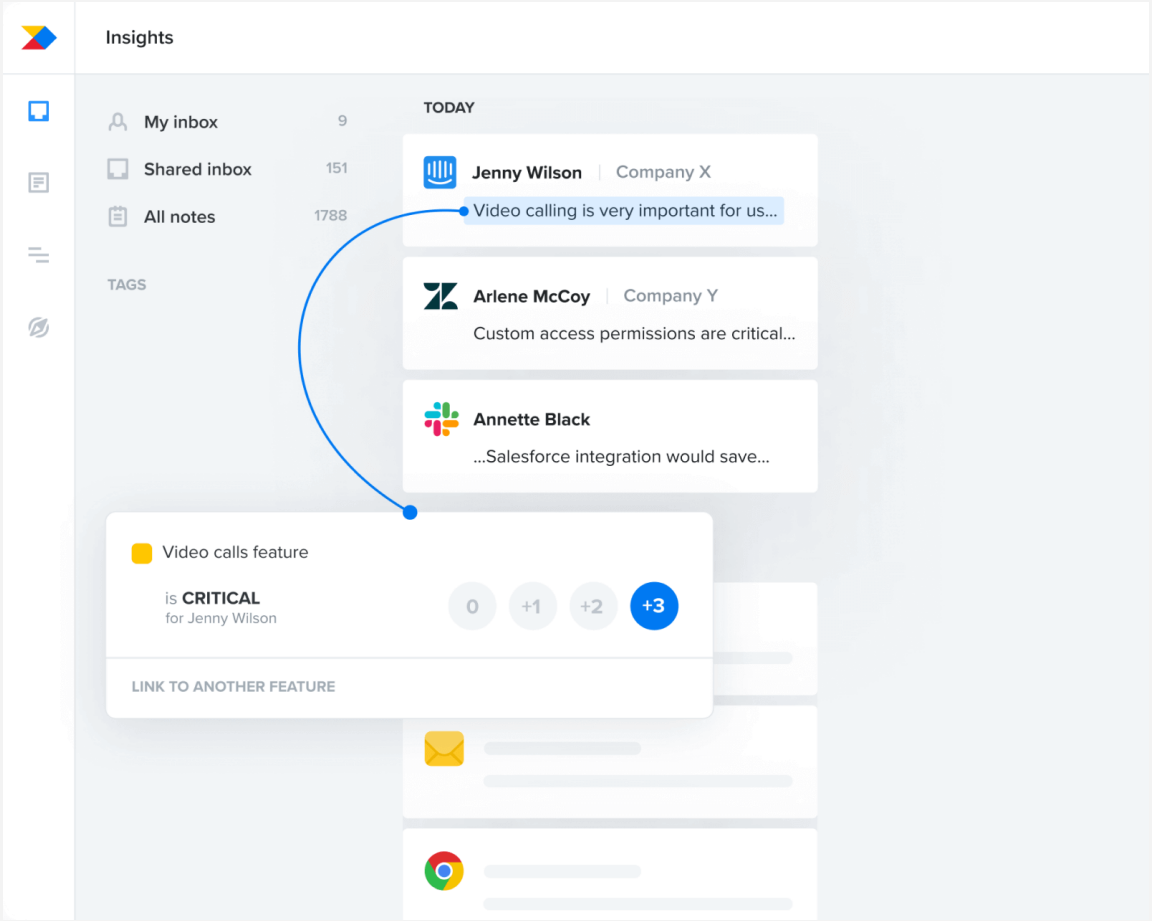
# What is Productboard ?

# The customer-centric product management platform

Productboard helps product managers understand what customers need, prioritize what to build next, and rally everyone around the roadmap.


Features Timeline Roadmap
Q1 2020
JAN
FEB
MAR
Q2 2020
APR
MAY
Core
Sprint 1
Sprint 2
New onboarding experience
Should-have
8
Video calling enhancements
Must-have
5
Custom text formatting
Should-have
20
Platform


Prioritization view
DRIVERS
Prioritization score
Impact
Confidence
Zlack app
280
Communicate with team mates
280
Video calls
90
Screen sharing
60
Add emojis to messages
50


Insights
My inbox 9
Shared inbox 151
All notes 1788
TAGS
TODAY
Jenny Wilson | Company X
Video calling is very important for us...
Arlene McCoy | Company Y
Custom access permissions are critical...
Annette Black
...Salesforce integration would save...
Video calls feature
is CRITICAL
for Jenny Wilson
0
+1
+2
+3
LINK TO ANOTHER FEATURE

Public Portal

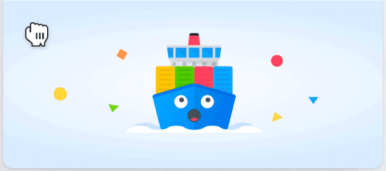
UNDER CONSIDERATION | IN PROGRESS | RELEASED

## Integrations

**Multi-team channel segments** 27

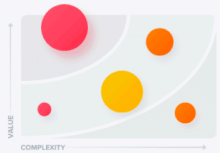
There are multiple ways of name a same feature. While searching for a feature to link an insight, I need to find all features regardless of the synonym I use to search.

**Bulk import your feature ideas** 38

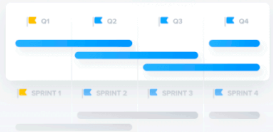


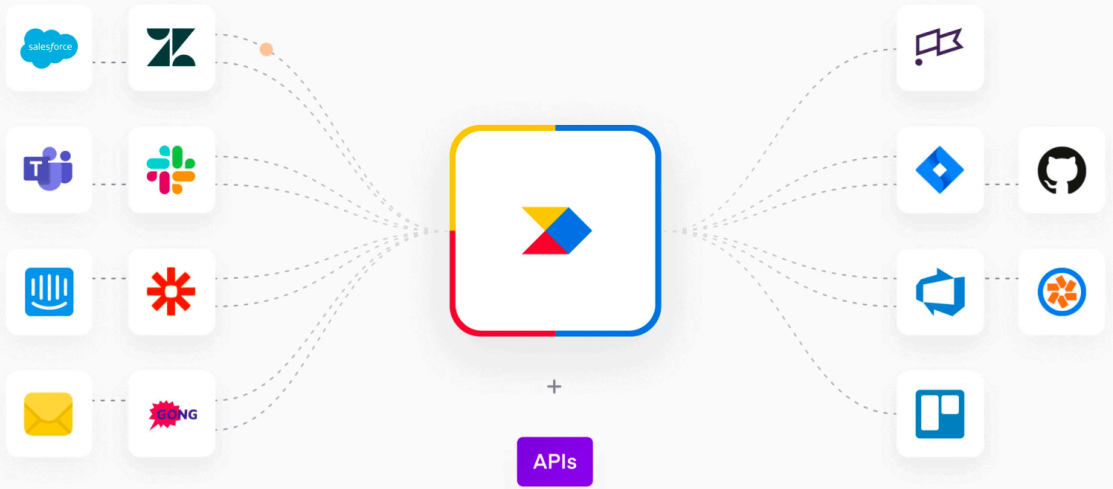
## Scaling up

**Interactive matrix prioritization** 22



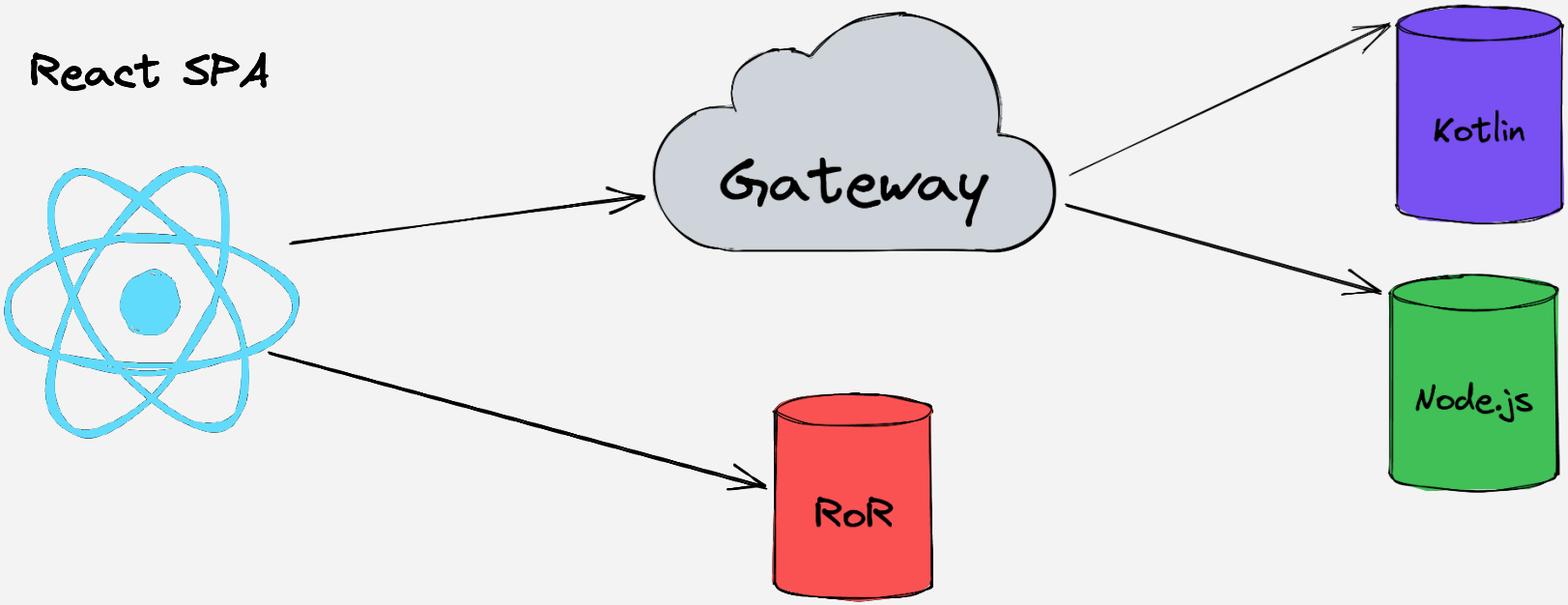

**Release groups** 85



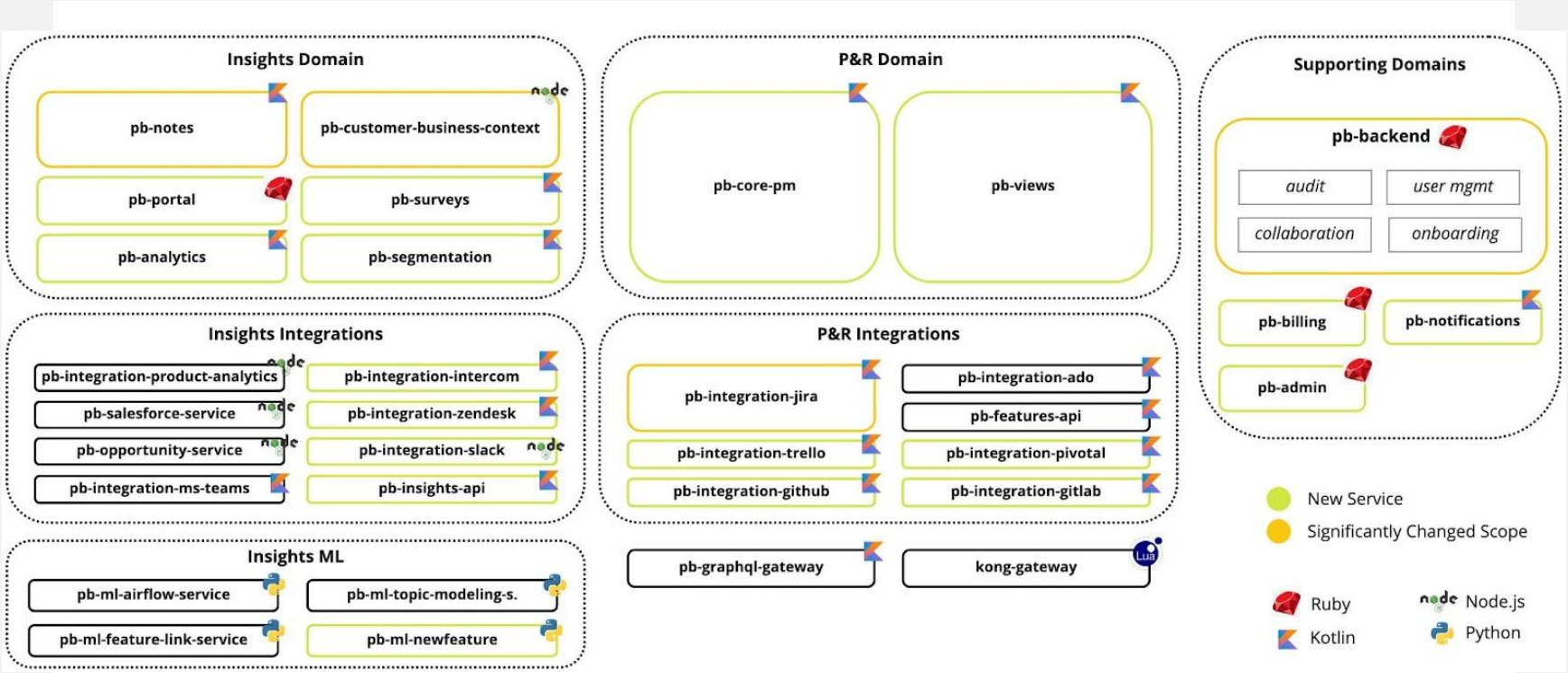

salesforce
GONG
+
APIs

# Data fetching evolution 🚀

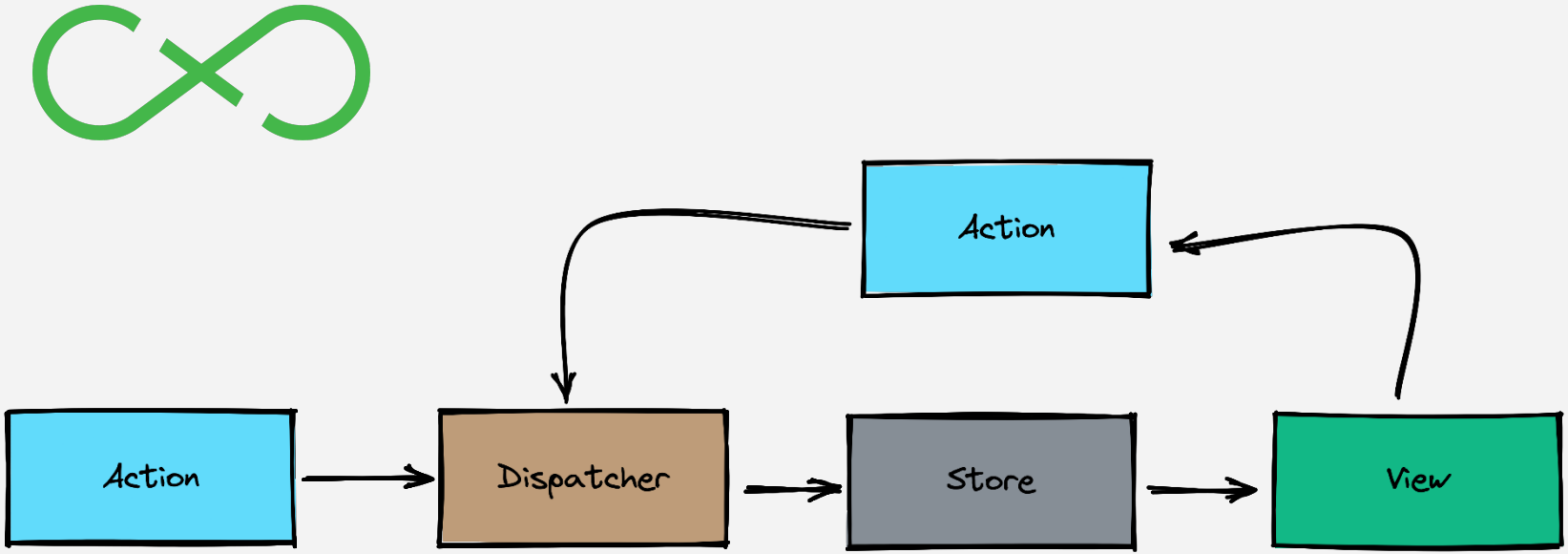
# Brief architecture overview



# Backend services & stacks



# Frontend architecture

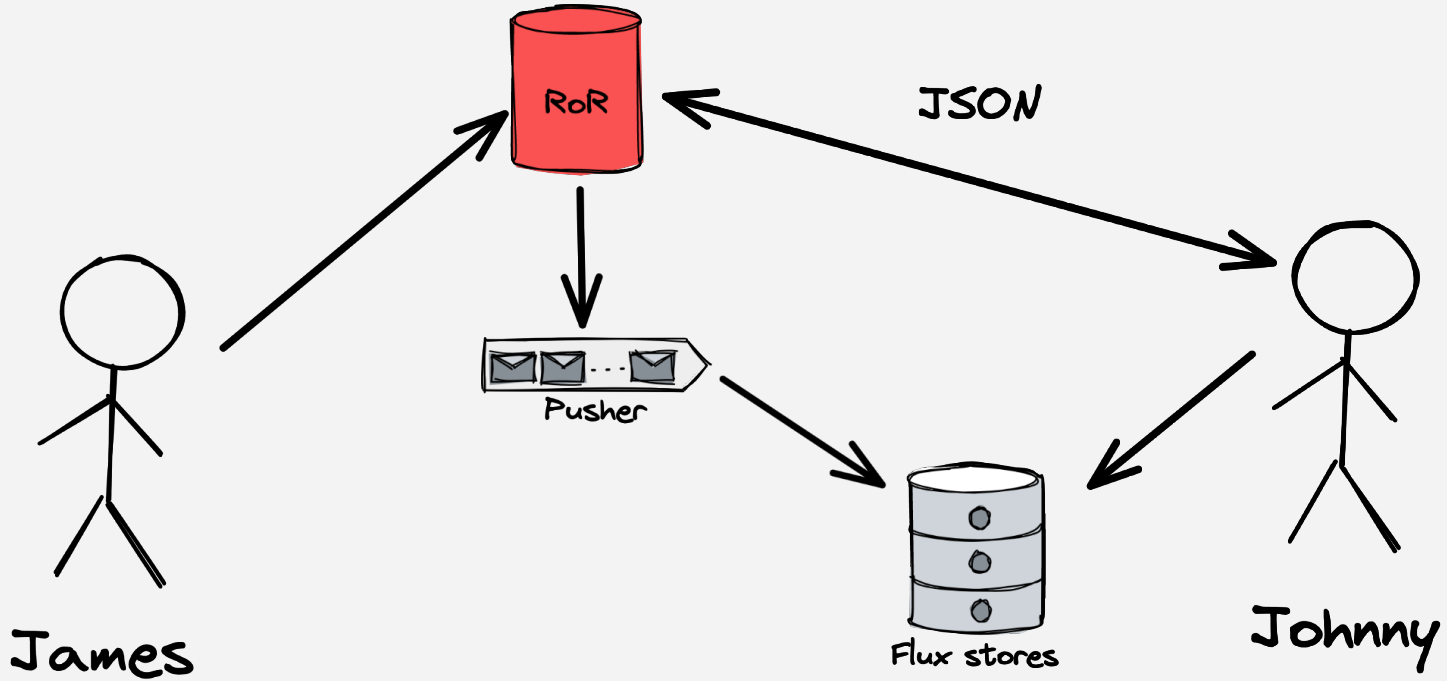


# RestCollectionStore

```ts
// Active record model "features"
export type Feature = {
  id: string;
  archived: boolean;
  name: string;
  // etc ...
}
```

```ts
export class FeatureStore extends RestCollectionStore<Feature> {
  constructor() {
    super('features');
  }

  // From RestCollectionStore
  get(id: ID): Feature {
    return this._records[id]
  }
}
```

# Full picture



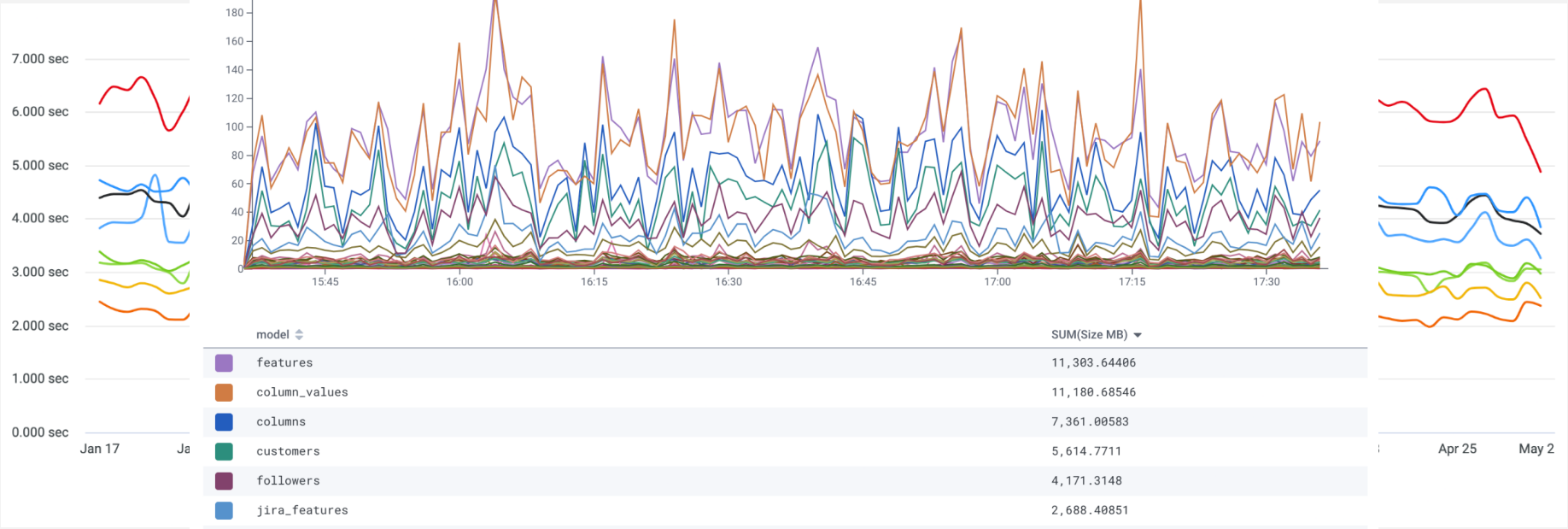
## Pros 👍

- Developer velocity
- Real time updates
- Normalized data

## Cons 👎

- Initial load
- Fat client
- Over fetching

# Proper monitoring in place




May 2 2022, 3:36:56 PM – May 2 2022, 5:36:56 PM (Granularity: 1 min)

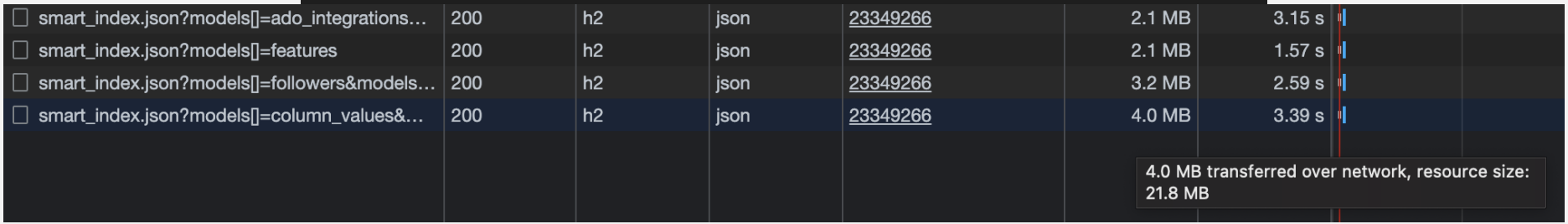
SUM(Size MB)



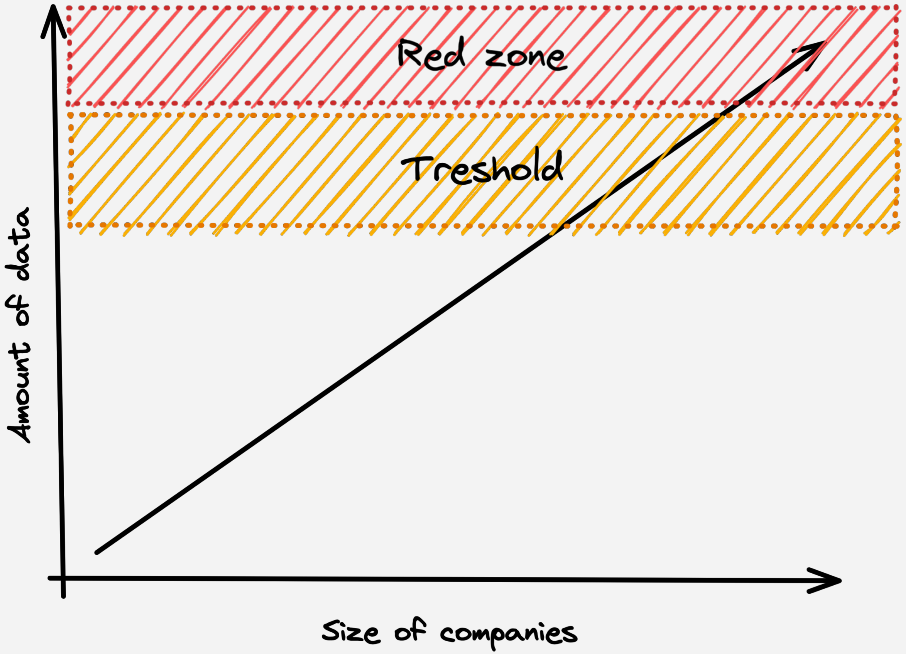

| model | SUM(Size MB) |
|---|---|
| features | 11,303.64406 |
| column_values | 11,180.68546 |
| columns | 7,361.00583 |
| customers | 5,614.7711 |
| followers | 4,171.3148 |
| jira_features | 2,688.40851 |
| release_assignments | 1,800.96084 |
| taggings | 968.07565 |
| views | 871.88977 |

# 40 MB of JSON

```json
{
  "features": [
    { "id": 1, "name": "Lazy loading" },
    { "id": 2, "name": "Webpack 5" }
  ],
  "users": [
    { "id": 1, "name": "Lukáš Huvar"}
  ]
}
```



# What to do?




1. Complete rewrite of data fetching
2. Incremental adoption new strategy

# Reduction of all.json

- Currently we have **106 data models**

Dead ends ❌

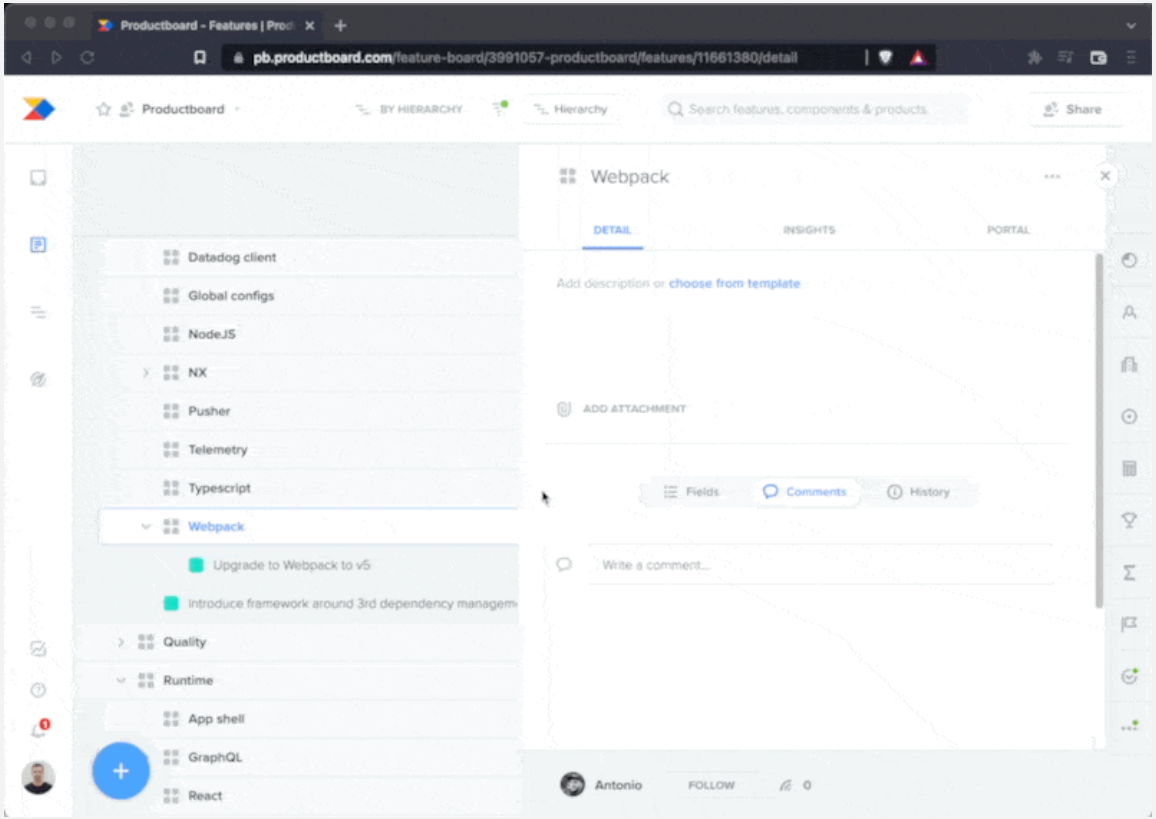

Backends for Frontends BFF

# GraphQL

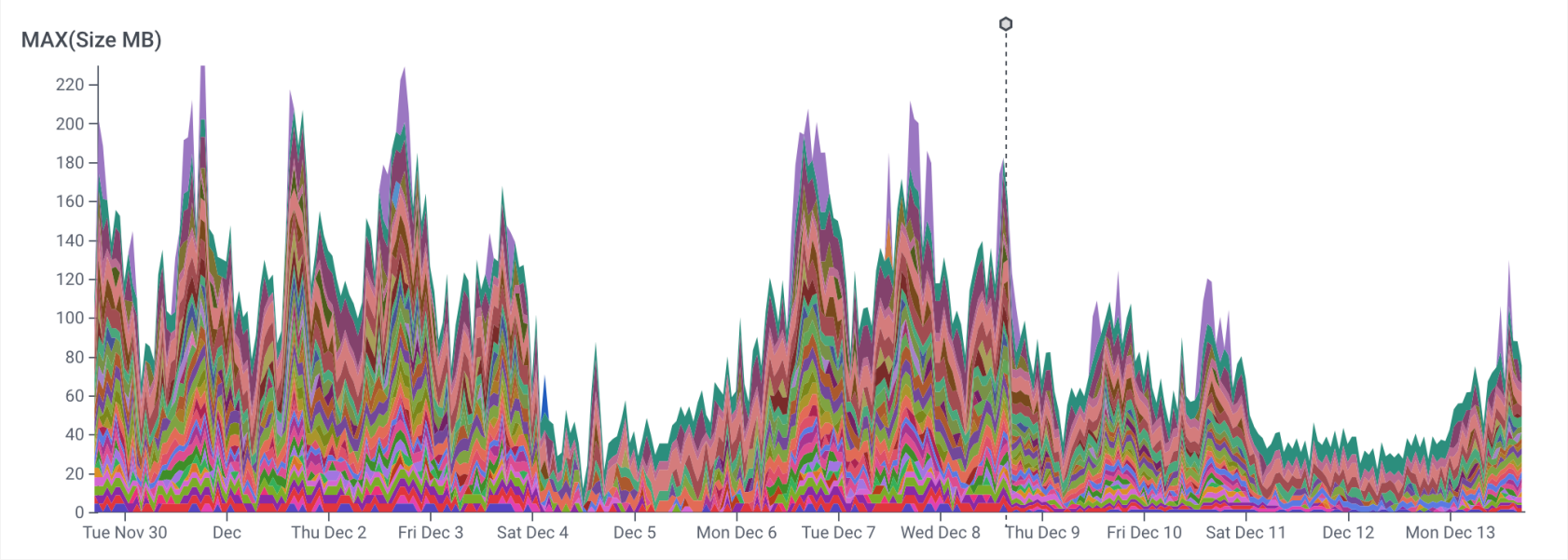
- Federated GraphQL
- Relay

# Squeezing the lemon 🍋

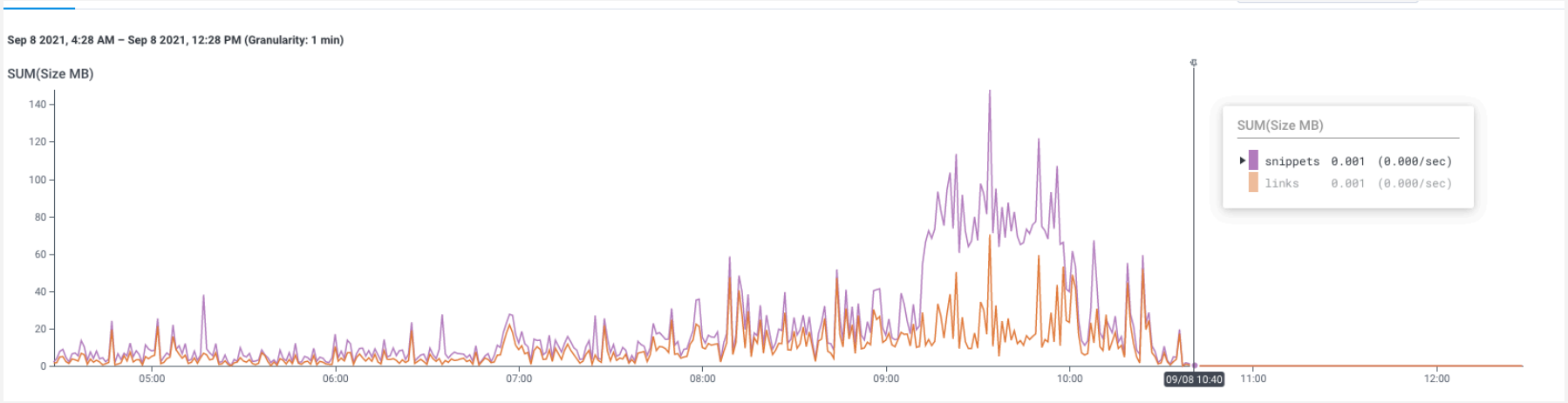
# Lazy loading part of store



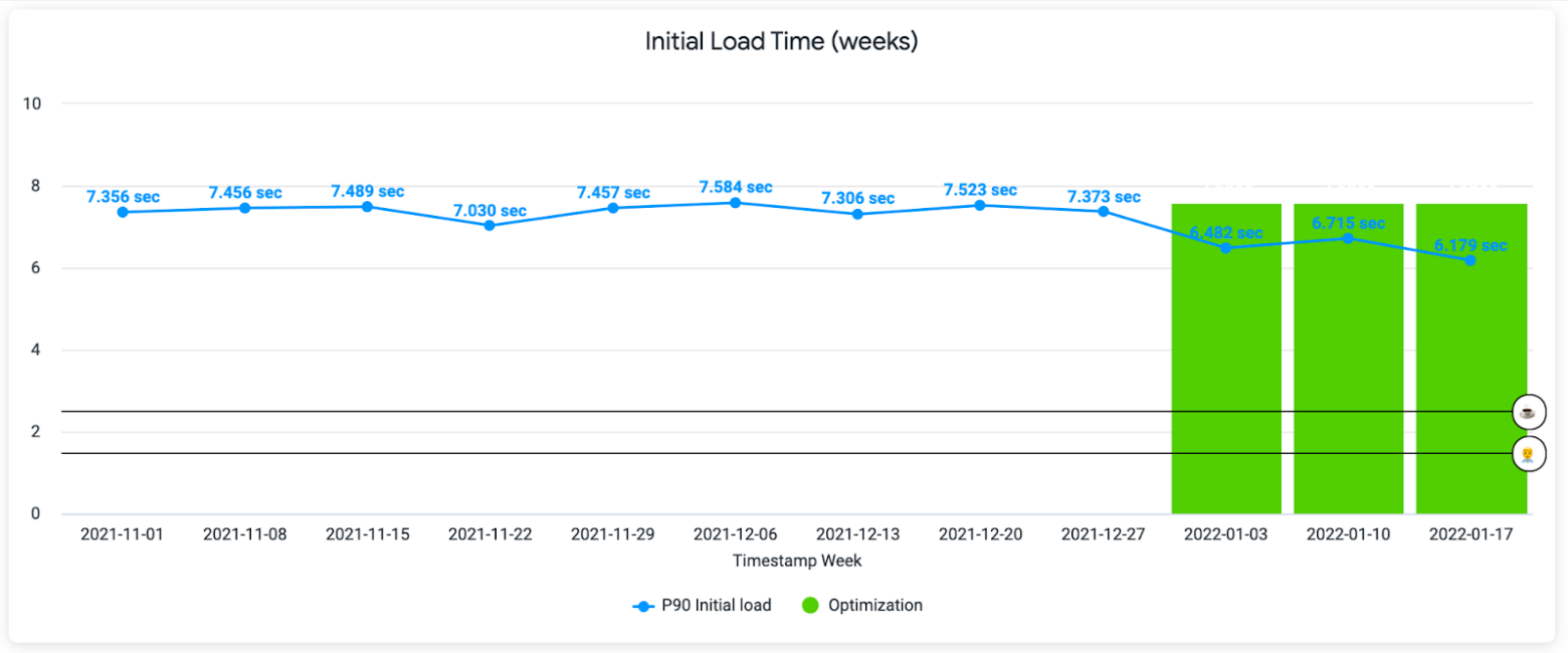
# Lazy loading part of store



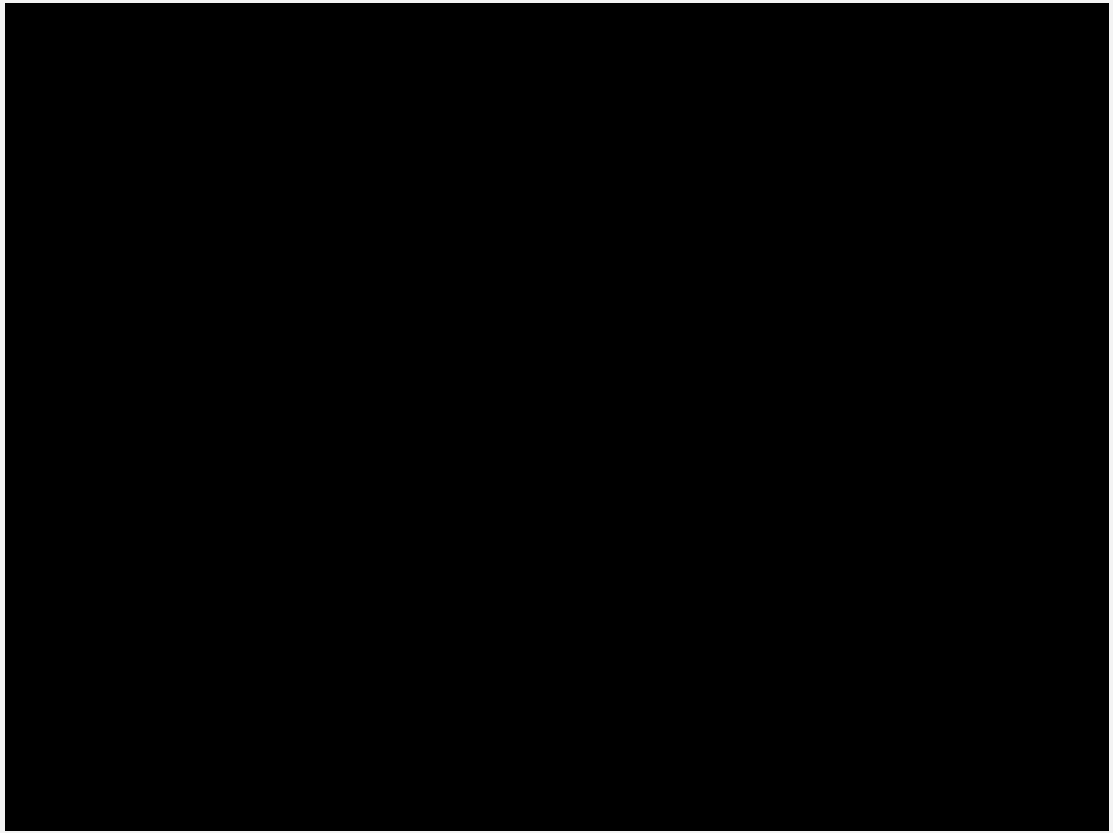
# Lazy loading stores



# One request to multiple requests



# Minimal application



# Prefetching data request

```
const DEFAULT_MODELS = [
  ['features'],
  ['followers', 'customer', 'taggings'],
  ['column_values', 'columns'],
];

const fetchModels = models ⇒
  fetch(
    `/api/all/smart_index.json?${models
      .map(model ⇒ `models[]=${model}`)
      .join('&')}`,
  ).then(response ⇒ response.json());

Promise.all(DEFAULT_MODELS.map(models ⇒ f
  .then(insertDataToStores)
  .then(hideLoading);
```

```
<!DOCTYPE html>
<html lang="en" prefix="og: https://ogp.me/ns#">
  <head>
    <link
      rel="preload"
      href="/api/all/smart_index.json?models[]=features"
      as="fetch"
    />
    <link
      rel="preload"
      href="/api/all/smart_index.json?models[]=followers&models[]=customers&models[]=taggings"
      as="fetch"
    />
    <link
      rel="preload"
      href="/api/all/smart_index.json?models[]=column_values&models[]=columns"
      as="fetch"
    />
  </head>
  <!— Rest of index.html ... —>
</html>
```

# Fetching only data we need for a screen

```
export const mapModelsToRoutes = [
  {
    route: new RegExp(/(\/$)|(\/insights\/.*)/),
    requiredModels: ['users', 'tags', 'customers'],
  },
];

const { pathname } = window.location;
const matchedRoute = mapModelsToRoutes.find(route => route.route.exec(pathname));

if (matchedRoute) {
  const optionalModels = removeRequiredModels(
    DEFAULT_MODELS,
    matchedRoute.requiredModels,
  );

  fetchModels(matchedRoute.requiredModels)
    .then(insertDataToStores)
    .then(hideLoading);

  Promise.all(optionalModels)
    .map(models => fetchModels(models))
    .then(insertDataToStores);
} else {
  Promise.all(DEFAULT_MODELS.map(models => fetchModels(models)))
    .then(insertDataToStores)
    .then(hideLoading);
}
```

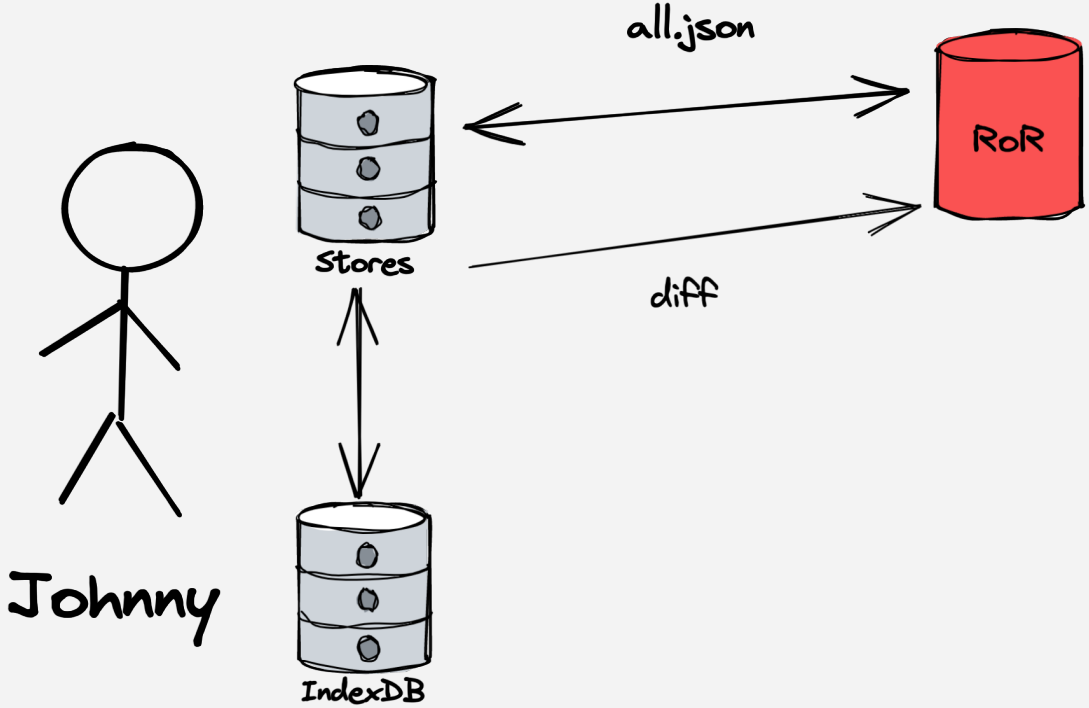
# Smart prefetching by route

```
addEventListener('fetch', (event) => {
  event.respondWith(handleEvent(event))
})

async function handleEvent(event) {
  const cache = caches.default

  let response = await cache.match(event.request)

  if (!response) {
    const template = await TEMPLATES.get(':current')
    response = new Response(template)

    event.waitUntil(cache.put(event.request, response.clon
  }

  return new HTMLRewriter()
    .on(`link[rel="preload"]`, new PreloadHandler(pathname
    .transform(response)
}
```

```
const generatePreloadHref = (models) => {
  const modelsString = models.map(model => `models[]=${model}`).join('&');

  return `<link rel="preload" href="/api/all/smart_index.json?${modelsString}" as="fetch">`;
};

export class PreloadHandler {
  constructor(pathname) {
    this.pathname = pathname;
  }

  element(element) {
    const preloadRoute = mapModelsToRoutes.find(({ route }) =>
      route.exec(this.pathname),
    );

    if (preloadRoute) {
      const optionalModels = removeRequiredModels(
        DEFAULT_MODELS,
        preloadRoute.requiredModels,
      );
      optionalModels.forEach(models => {
        element.after(generatePreloadHref(models), { html: true });
      });
      element.after(generatePreloadHref(preloadRoute.requiredModels), {
        html: true,
      });
    } else {
      DEFAULT_MODELS.forEach(models => {
        element.after(generatePreloadHref(models), { html: true });
      });
    }
  }
}
```

# Precompute optimal fetching

# Local cache with conflict resolution



# Summary

- **Developer velocity** is key 🗝️
- There is no **silver bullet** 🚄
- **Prolong** migration runway 🛬

# Q & A

## Lukáš Huvar

**Software engineer** productboard

huv1k huv1k https://huvik.dev/