コンパウンド戦略を支える
フロントエンド基盤設計

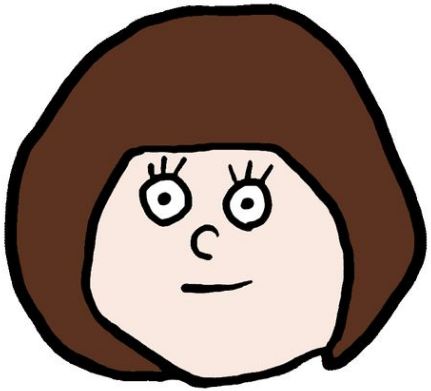
2024/11/06 Webフロントエンドを軸に、幅を広げたエンジニアたちの仕事

株式会社ナレッジワーク @yoshiko

_KNOWLEDGE WORK

自己紹介

よしこ @yoshiko_pg



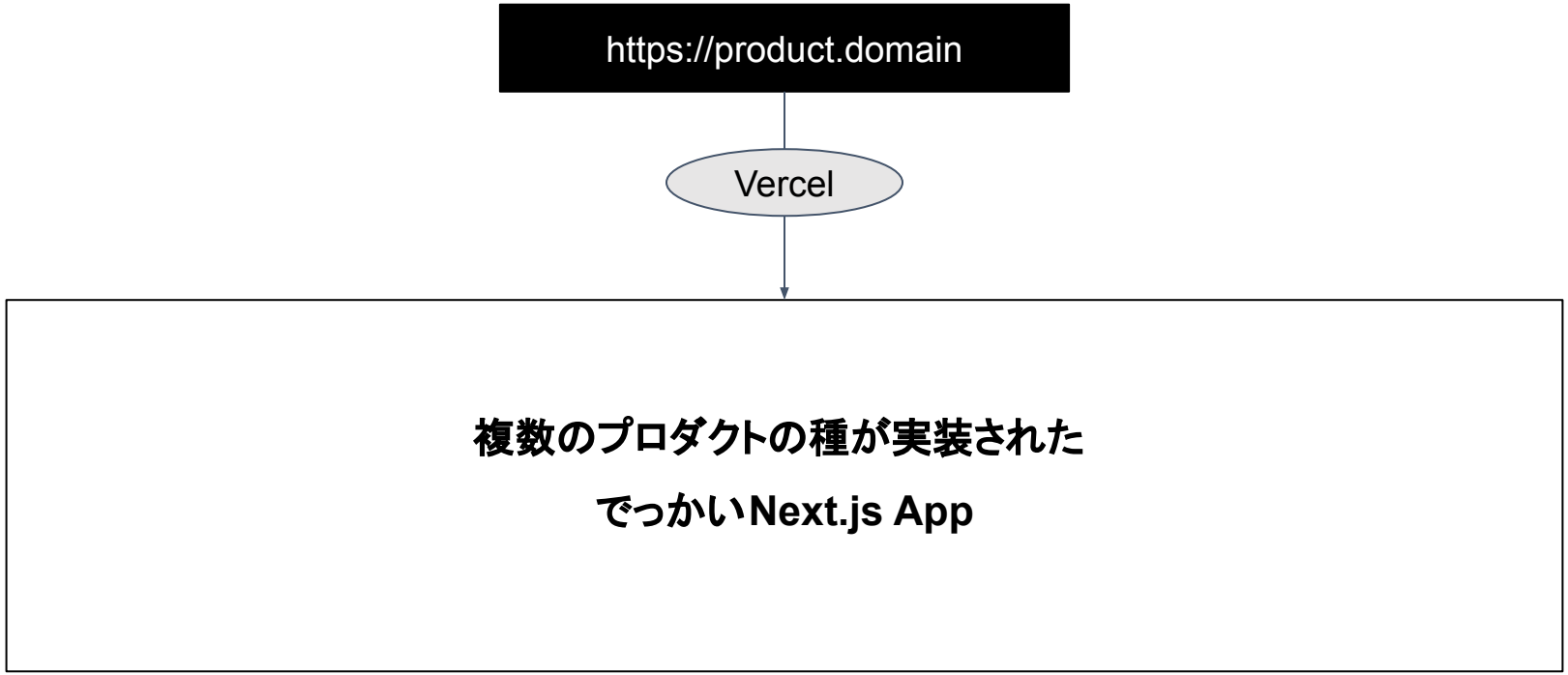
株式会社ナレッジワークのフロントエンドエンジニア
GUIをSPAとして作るのが好き。

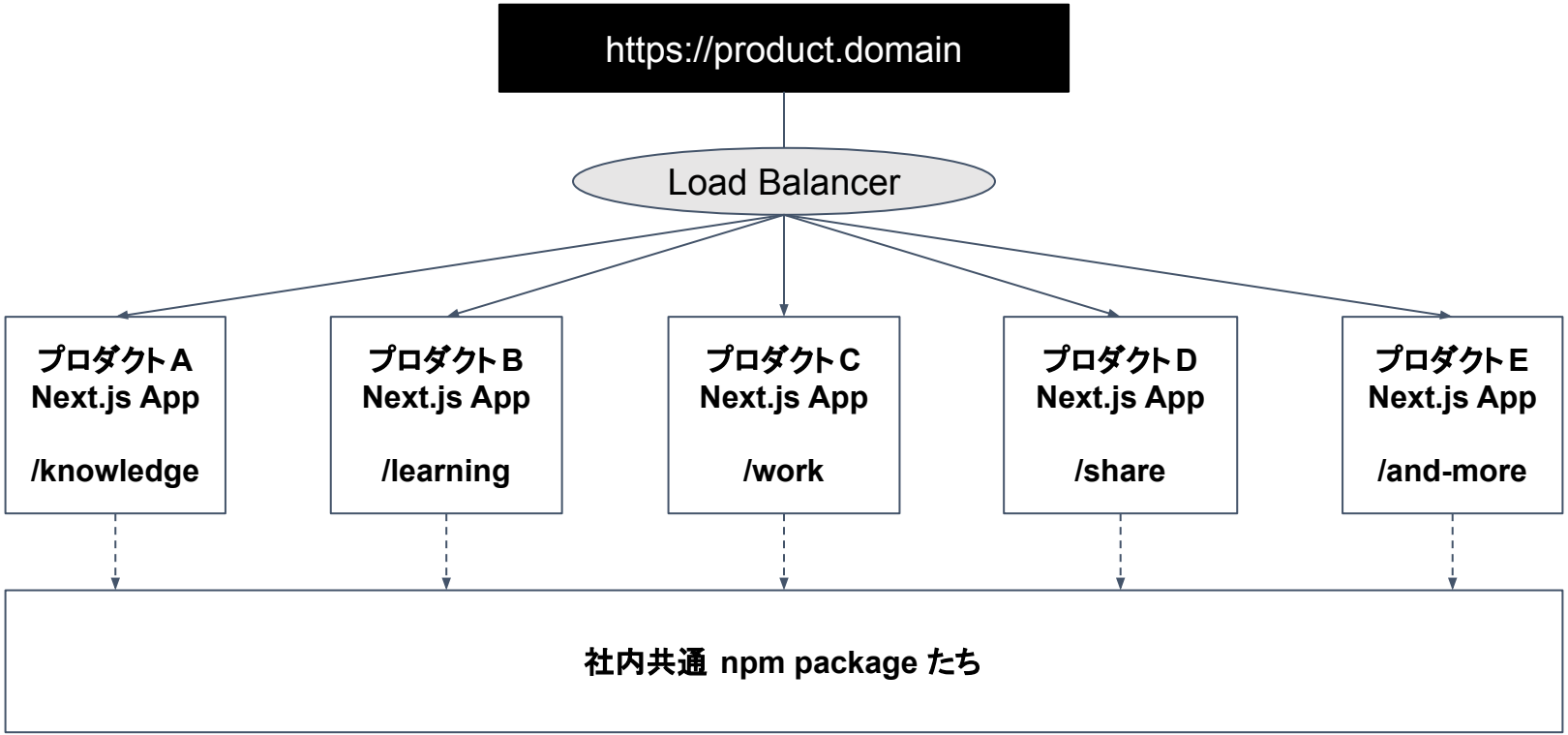
自社での技術スタックや設計を Zennで公開しています。
https://zenn.dev/yoshiko

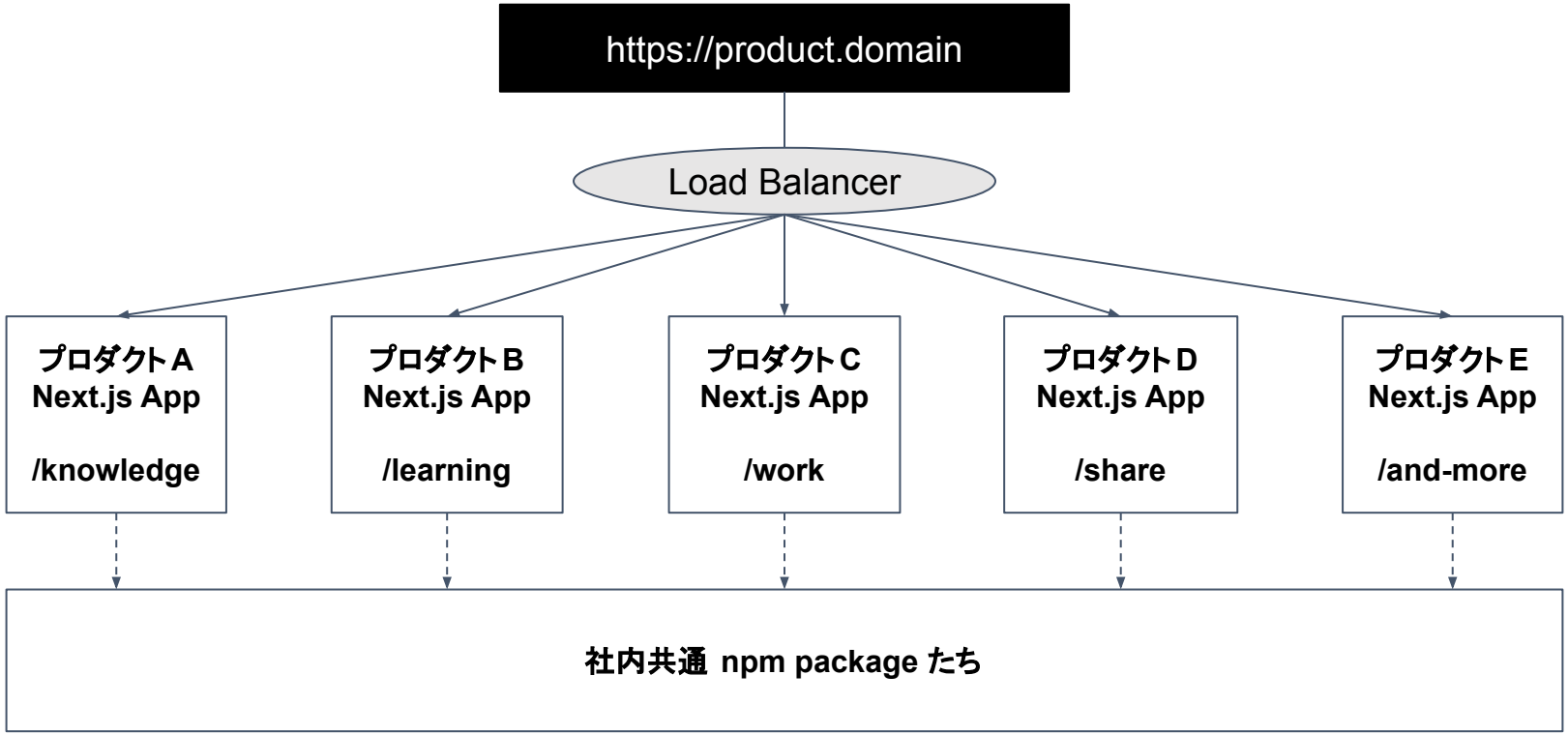
Agenda 目次

- 技術基盤とか構成どんな感じなの？
- なんでそうしたいの？
- 具体的な移行の話

技術基盤とか構成どんな感じなの？

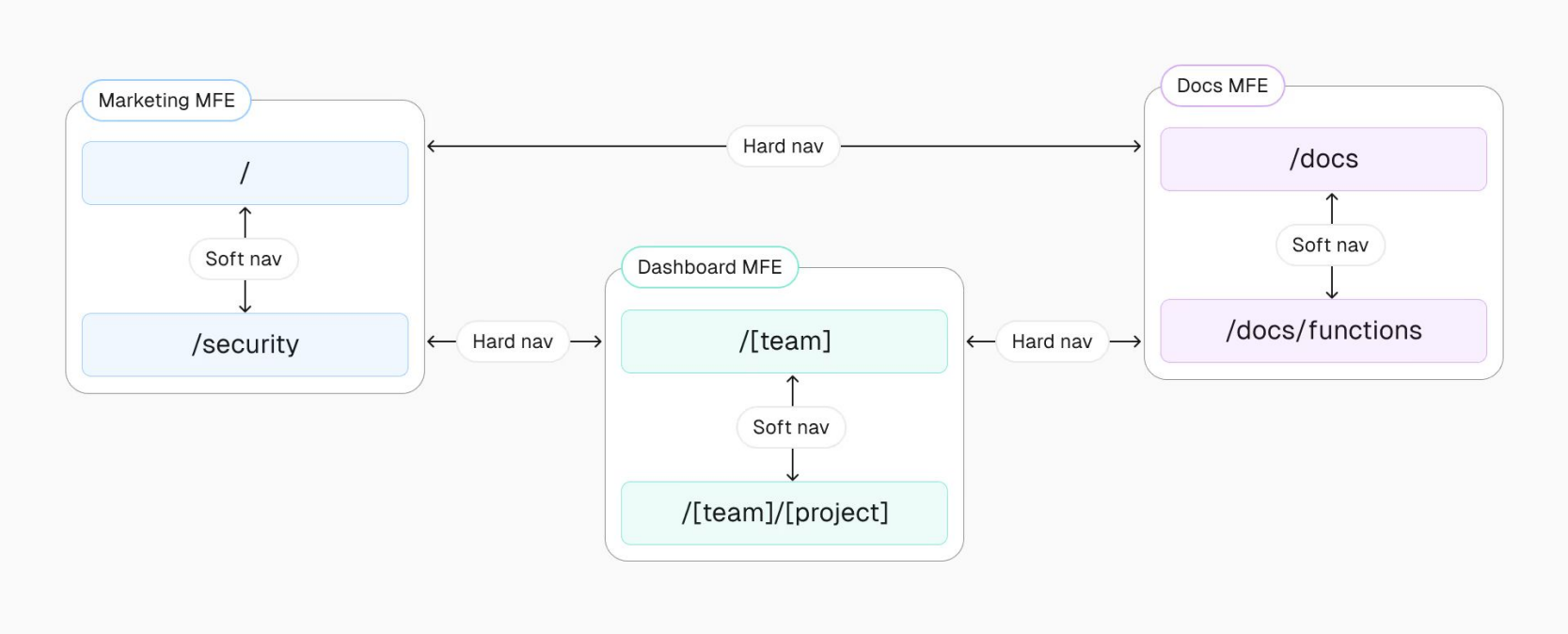






に、しようとしています（進捗 80%）

垂直分割マイクロフロントエンド



引用：[How Vercel adopted microfrontends]

（先月出た記事ですが、構成や実施内容がほとんど一緒でした）

なんでそうしたいの？

「フロントエンド技術基盤」が単体で存在するわけではない

プロダクト全体の 技術基盤 、

ひいては 会社全体の 技術戦略 から繋がってくるものですよね

「事業戦略の実現」を目的におきたい

事業戦略を理解する必要がある

事業戦略 ＝ 顧客価値をどう創出するか？

顧客価値のブレイクダウン

社内戦略資料より抜粋

_KNOWLEDGE WORK

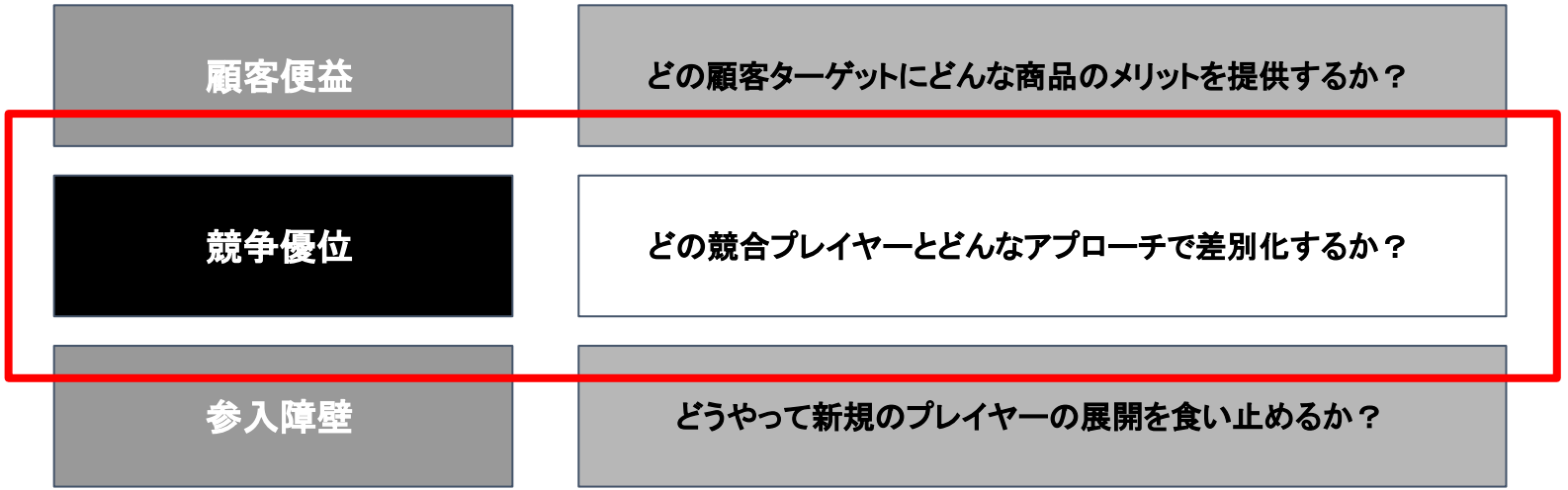
顧客便益	どの顧客ターゲットにどんな商品のメリットを提供するか？
競争優位	どの競合プレイヤーとどんなアプローチで差別化するか？
参入障壁	どうやって新規のプレイヤーの展開を食い止めるか？

「顧客便益」「競争優位」「参入障壁」により、
顧客価値（＝顧客が自社の商品を選ぶ理由）を確立できます

顧客価値のブレイクダウン

社内戦略資料より抜粋

_KNOWLEDGE WORK



「顧客便益」「競争優位」「参入障壁」により、
顧客価値（＝顧客が自社の商品を選ぶ理由）を確立できます

競争優位のブレイクダウン

社内戦略資料より抜粋

_KNOWLEDGE WORK

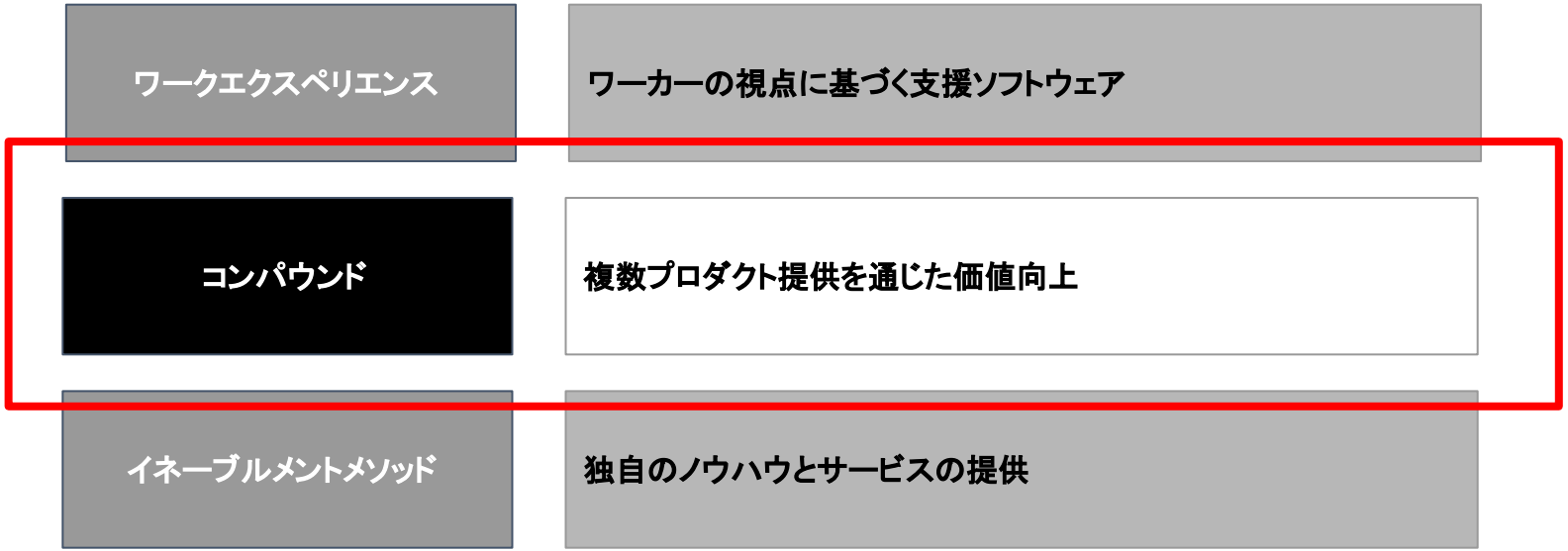
ワークエクスペリエンス	ワーカーの視点に基づく支援ソフトウェア
コンパウンド	複数プロダクト提供を通じた価値向上
イネーブルメントメソッド	独自のノウハウとサービスの提供

ナレッジワークは 3つの競争優位を実現します

競争優位のブレイクダウン

社内戦略資料より抜粋

_KNOWLEDGE WORK



ナレッジワークは 3つの競争優位を実現します

コンパウンドとは？

コンパウンド戦略

製品を複数提供することで乗数的に顧客価値を向上させる

※コンパウンド＝複利

コンパウンド戦略

製品を複数提供することで <u>乗数的に顧客価値を向上させる</u>

※コンパウンド＝複利

マルチプロダクト戦略

製品を複数提供すること

ちなみに今5個は進んでいます

創業から3年経ち、新しく打ち立てられた
この事業戦略を実現する技術基盤とは？

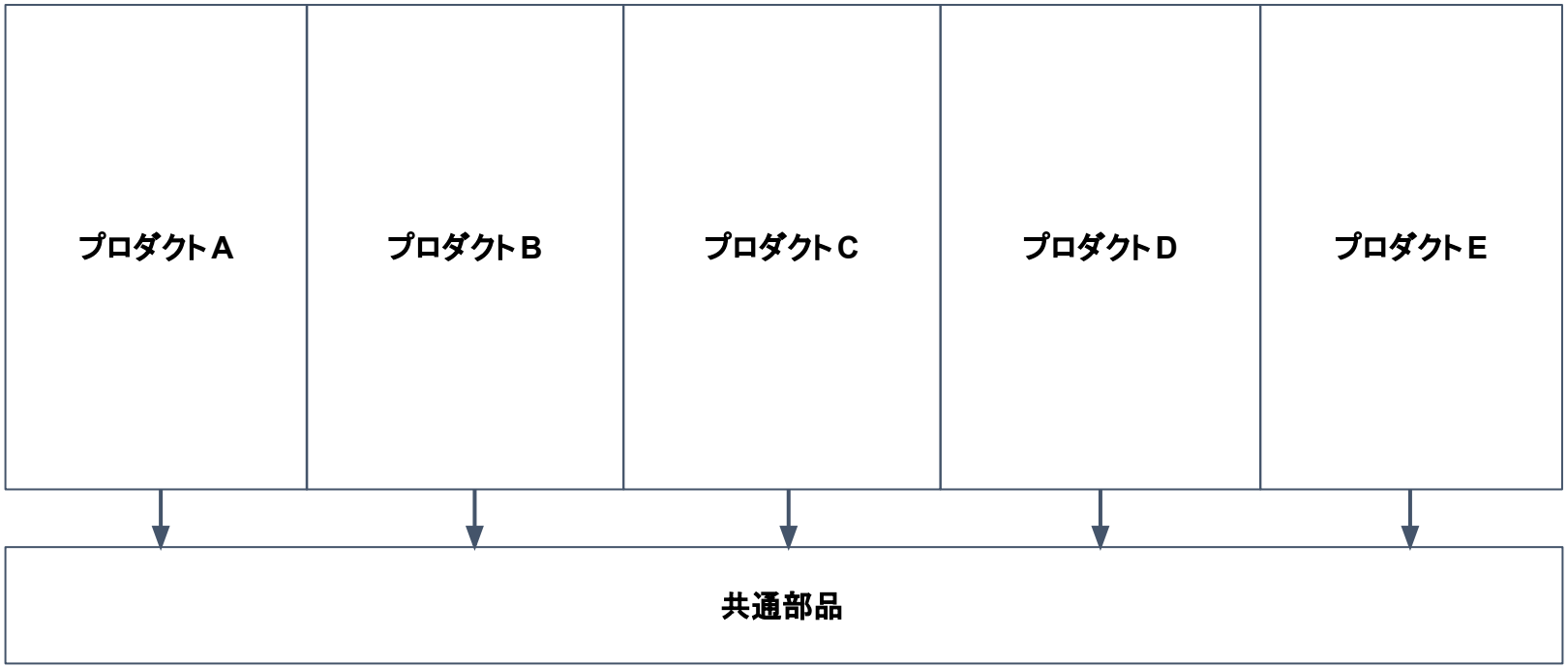
- **3年で**
  - **新しく作らないと存在しないものに集中する**
  - **既にあるものを作らない**
    - 社外はもちろん社内でこそ車輪の再発明を減らす
    - **プロダクト間で使える共通部品を最大化する**
      - 採用技術やフレームワークは同じものにする前提
- **10個**
  - **1チームが掛け持ちは無理。開発チーム数も増える**
  - **チーム間で開発フローの依存があるとスピードが下がる**
    - 例：全チーム開発完了してないとデプロイできない
  - **チーム間での依存を疎にし、自律性を最大化する**

共通部品は
最大だけど

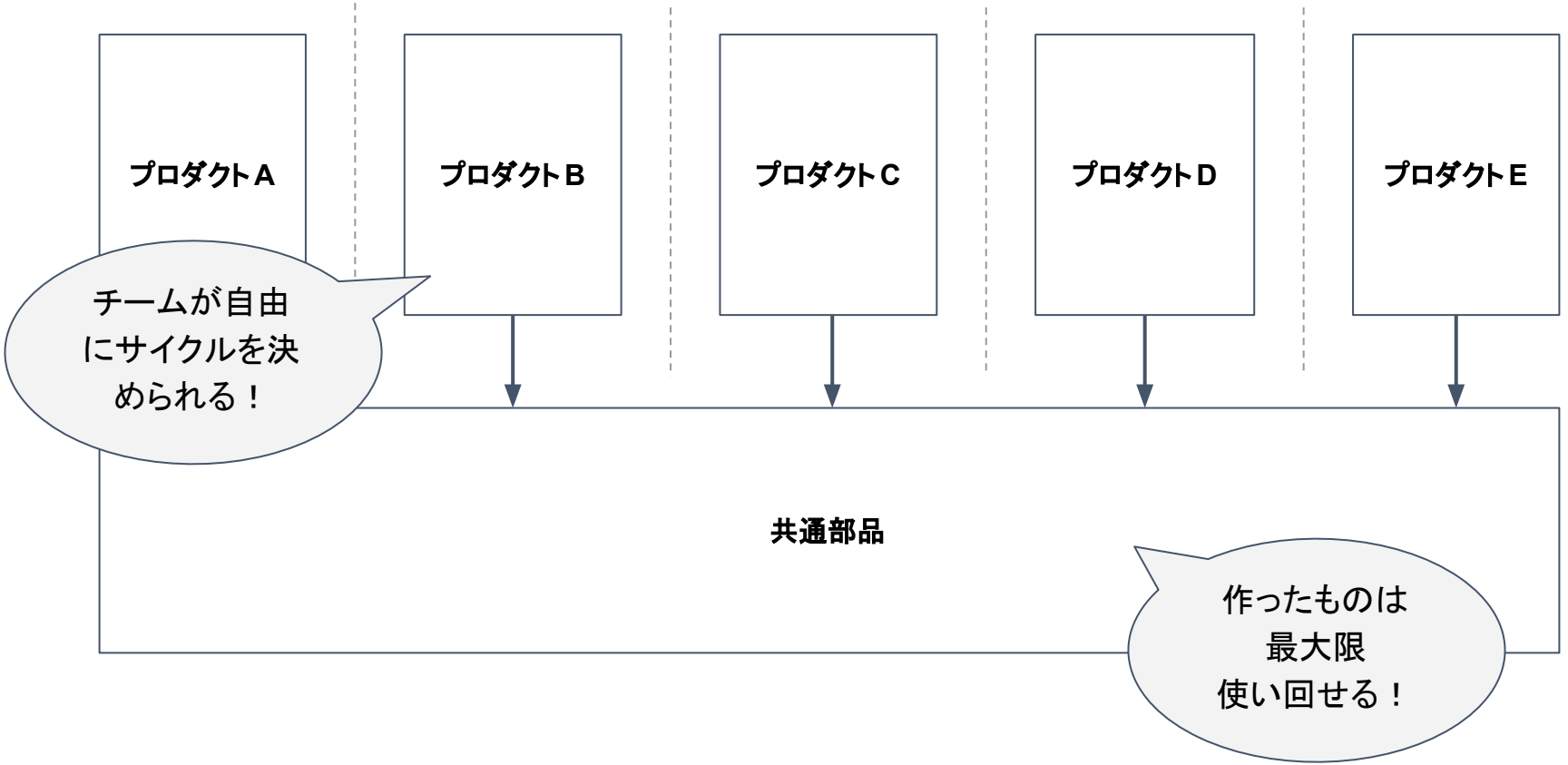
複数のプロダクトの種を内包した

でっかいモノリシックアプリケーション

全チーム足並み
揃わないとデプ
ロイできない…



こうしたい



プロダクト分割

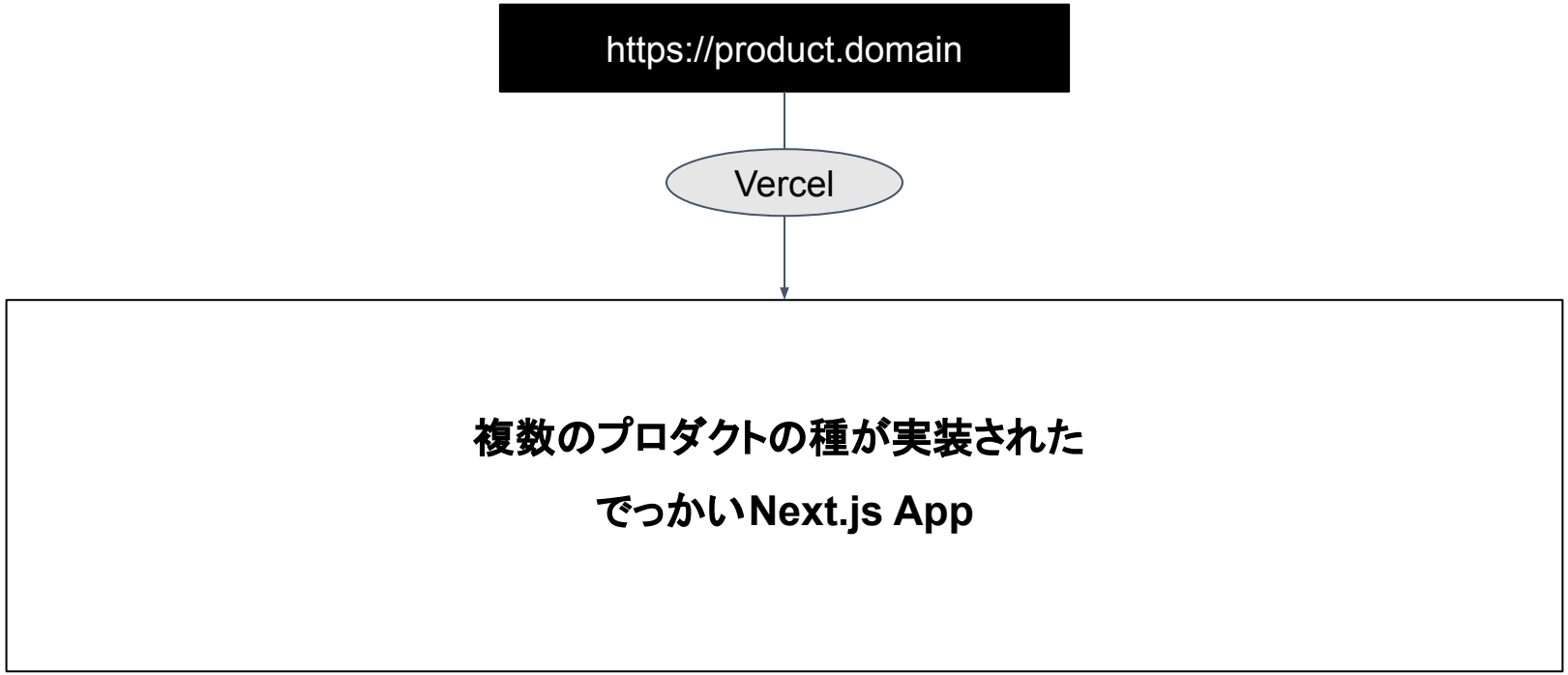
創業時はコンパウンドとか構想になかったのでシンプルな作りだったところから

事業戦略の変更に技術も追従

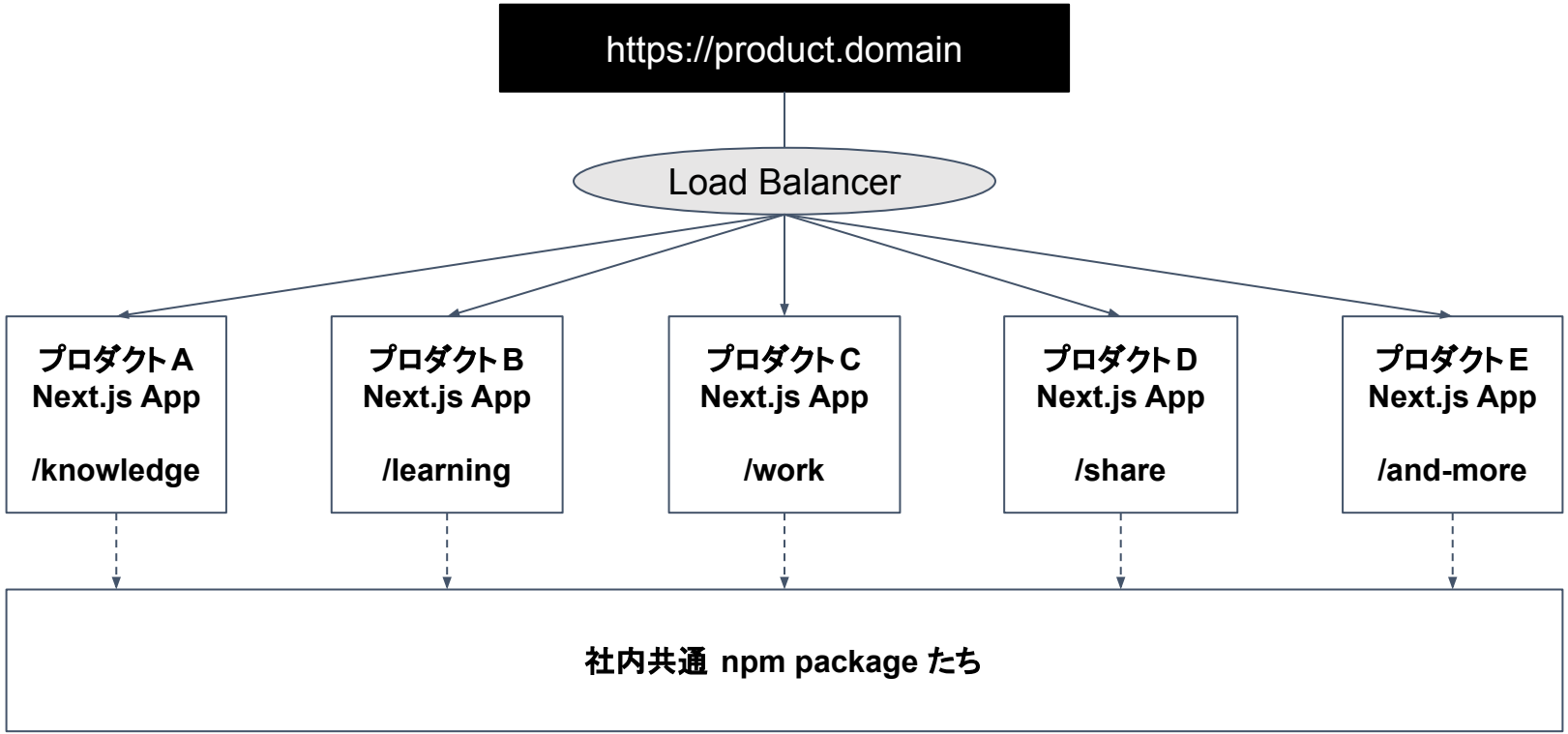
フロントエンドでは：
垂直分割マイクロフロントエンドへ

具体的な移行の話

これを



こうする



複数のプロダクトの種が実装された

でっかいNext.js App

俺達の戦いはこれからだ（本当に）

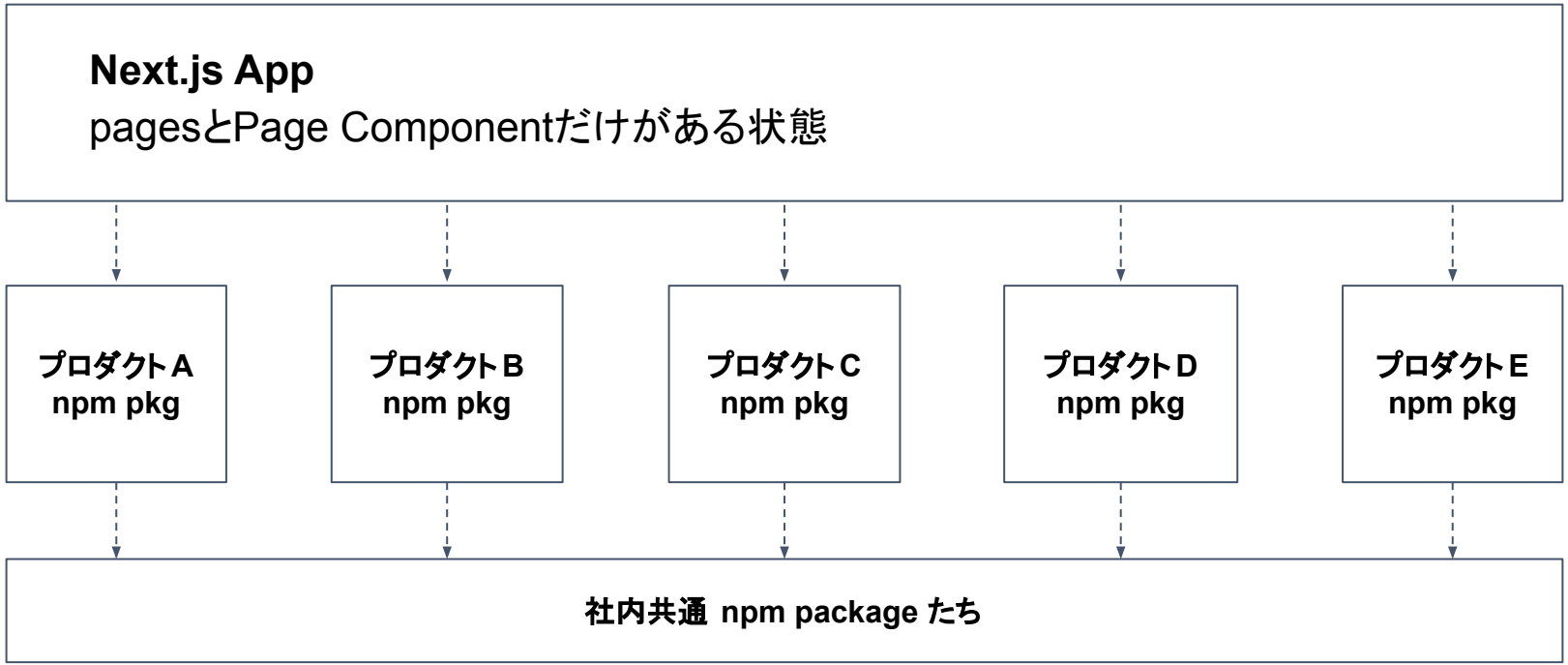
/application
  /product-a
  /product-b
  /…

/middleware
  /core
  /config
  /…

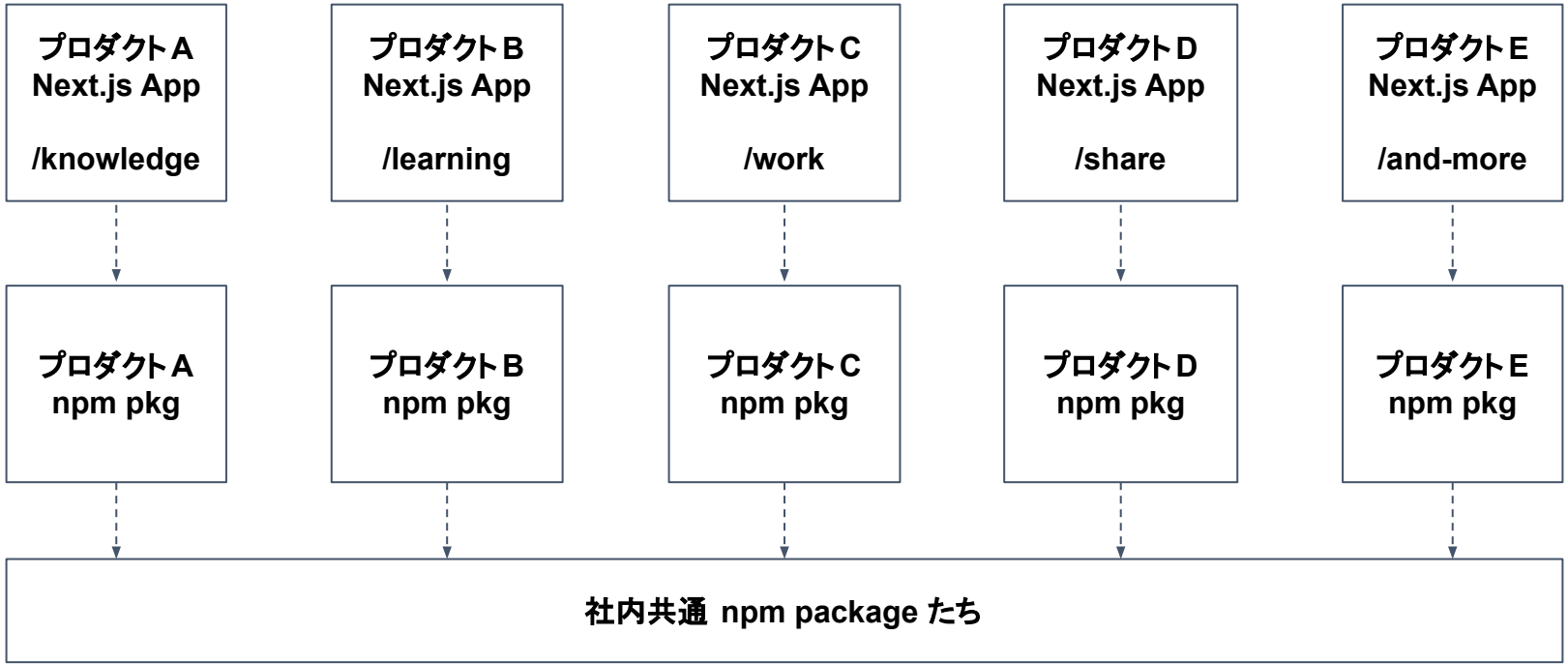
/packages
  /package-a
  /package-b
  /…

内部が分割整理された
でっかいNext.js App

App内で、分割予定単位でのディレクトリの整理・依存違反解消



ほとんどのコードを npm packageとしてpnpm workspaceで分離



最終形

- **pnpm workspace**
  - pnpmで同一パッケージの容量を集約
  - workspace:* 指定でコード変更に伴うversion up + publishが不要
- **Turborepo**
  - 依存によるbuild順のコントロールなど
  - remote cacheの活用はこれから
- **Vite**
  - ビルドの細かい調整がききやすい
  - Storybook, Vitestなどエコシステムサポートが充実

- **全プロダクトで共通利用する Global State**
  - プロダクトを超えて共有したいGlobal StateはlocalStorageに永続化して対応
- **高頻度デプロイとユーザー側での再読み込み**
  - フロントエンドのデプロイを検知して、新ソースを再読み込みできるように
    プロダクト内リンクも動的にハードナビゲーションに切り替える(参考)
- **CSSのビルド**
  - 当初計画していたtsupでCSS Modulesの解決とビルドが難しかった
  - Viteで進めたが、CSSの適用順が異なっていたりしてスタイルがデグレした
  - 現在は暫定的にAppのtsconfigの対象を各packageのComponentまで広げて
    ComponentだけApp側からbuildしている。将来的にはCSS in JSへ移行検討中

- **Storybook + Visual Regression Test**
  - Storybookは未分割。引き続き全ページをカバー
  - package化前後もVRTによりビジュアル差分チェック
  - アプリケーション分割が完了したらStorybookも分割したい
    - 立ち上げがアプリケーションより重いので..
- **E2Eテスト**
  - 分割関係なくユーザー視点でアプリケーションとして操作
  - 体験が変化していないことの担保

- **依存関係の整理ができた**
  - ドメインモデルの依存関係がごちゃごちゃしていたところを
    package分割したことであるべき形に整理ができた
- **新規プロダクト立ち上げの際のビルドが軽い、他への影響リスク低い**
  - 10月から分割後構成でのプロダクト開発がひとつ開始
  - 身軽！！！

App分割が完了すればさらにチーム開発の速度に効いてくる期待

まだまだ課題も山積み

- **共通コアパッケージでかすぎ問題**
  - 既存の共通部分を一旦全部コアとして切り出したのでデカい
  - 型チェックや動作に必要なビルドが重い
  - **vite内部がrolldownになったら差分ビルド可能になる …？**
- **依存ライブラリのバージョン揃えないと CI落ちる問題**
  - プロダクト間で同一パッケージのバージョンがずれるとbuildやtestが落ちる
  - **pnpm catalog使いたい。renovateの対応待ち…**
- **ハードナビゲーション問題**
  - プロダクト間の移動がハードナビゲーションになってしまい遅い
  - **プリフェッチなどの工夫が必要**

基盤もプロダクトと同じ、回せフィードバックループ

最後に

マルチプロダクト戦略であれ、
コンパウンド戦略であれ、

一つ一つのプロダクトに価値があることが最も大切

プロダクトの価値を高速で検証できるように

MVPを高速に作れる基盤を提供していくぜ！

そんなプロダクトと基盤を
一緒に作っていってくれる方を募集しています！

カジュアル面談はこちらから

スポンサーブースで
お話ししましょう！
（toggleホール）

質問も歓迎です！

できる喜びが巡る日々を届ける

Deliver the joy of enablement