プロダクトライン開発の本質
～その様々な開発形態の共通性と可変性～

2013年08月23日

林 好一（SRA）

Agenda

- PLE とは？
- 様々な形
- 何が違うのか？
- 派生開発 → PLE の移行
- まとめ

PLE とは？

派生開発では既存のアプリケーションを元として
そこから別のアプリケーションを派生させる

PLE ではコア資産というアプリケーション間で共通に使う成果物を基にしてそこからそれぞれのアプリケーションを派生させる

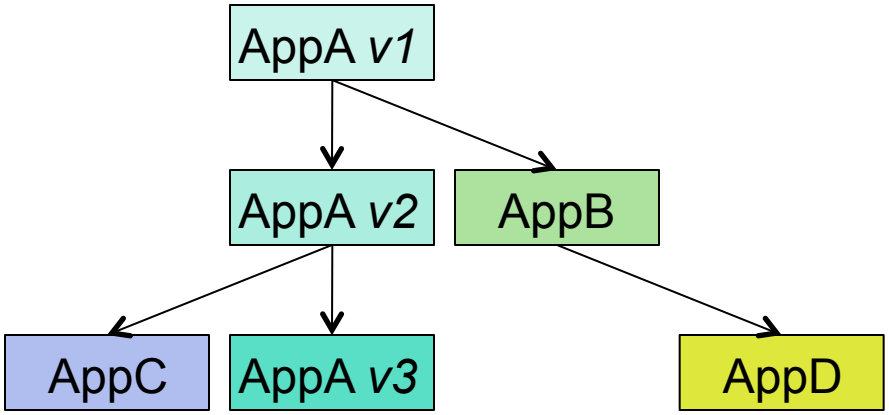
どちらも派生

違いは？

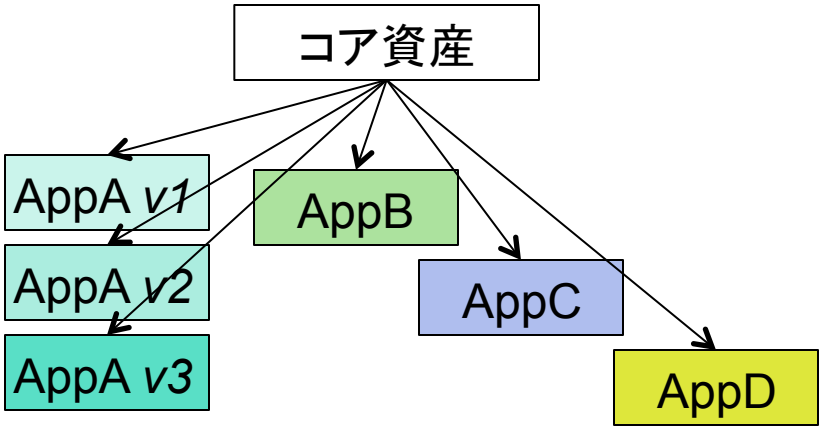
派生開発と PLE

派生開発：差分開発、改造開発、改良保守、等ともいう

派生開発の典型例



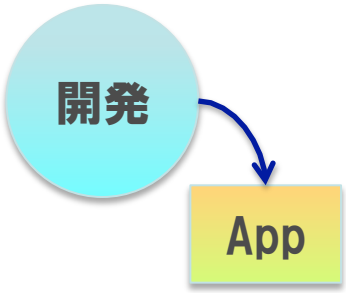
SPLEの基本形



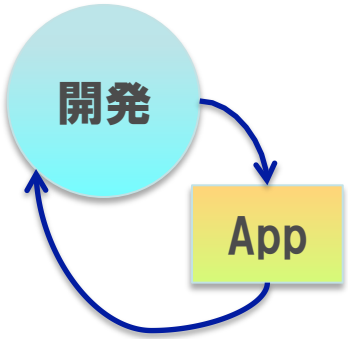
- 過去のアプリケーションを基に再利用を検討する
  - 作ったものを再利用する
- 将来のアプリケーションを基に再利用を検討する
  - 再利用するものを作る

様々な形

開発形態三種

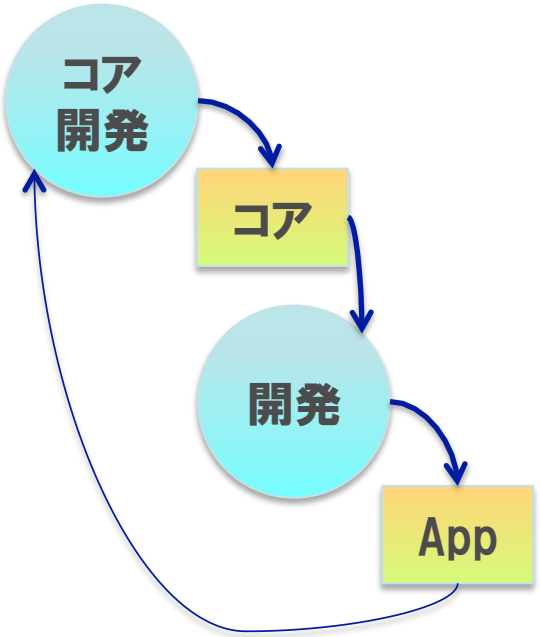


単一開発



派生開発

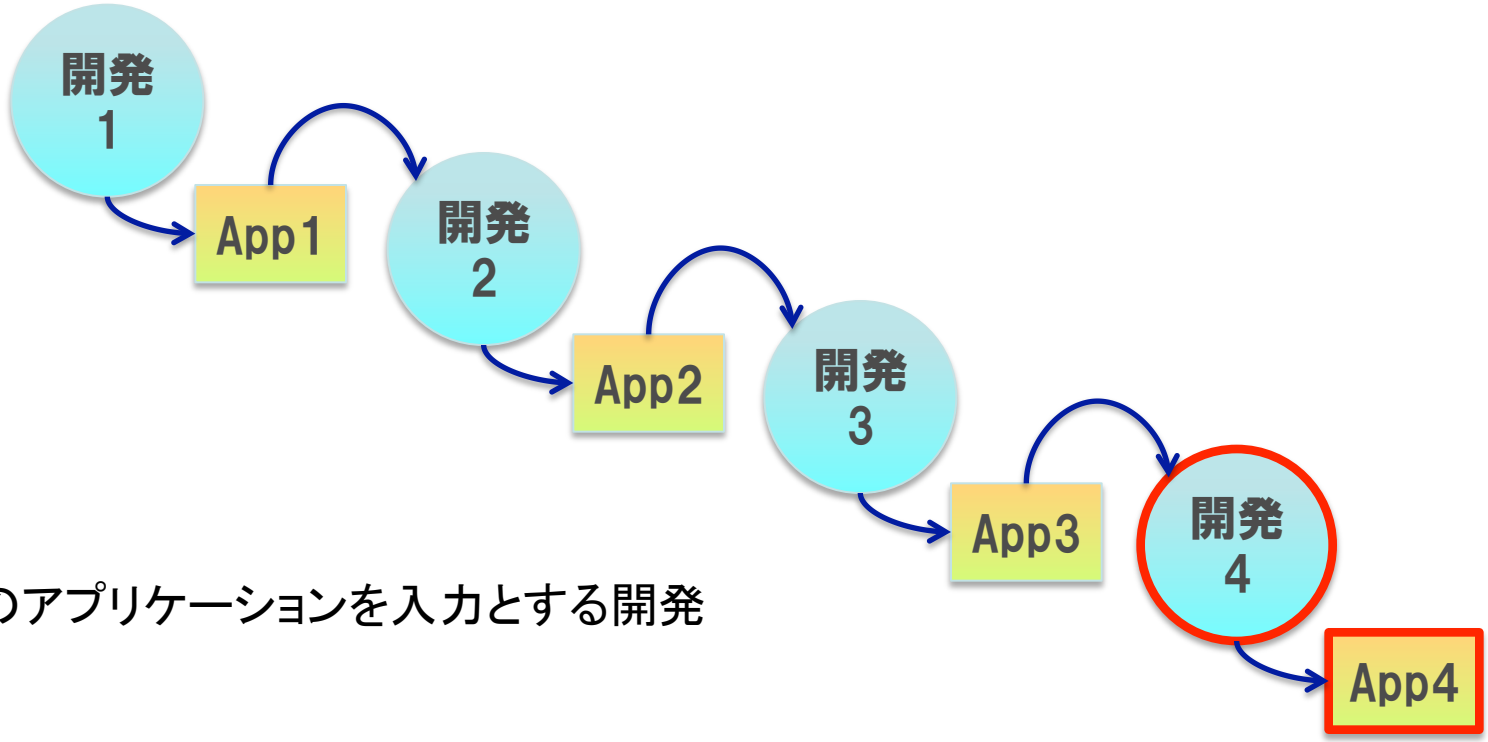
- 既存のアプリケーションを入力とする開発



SPLE

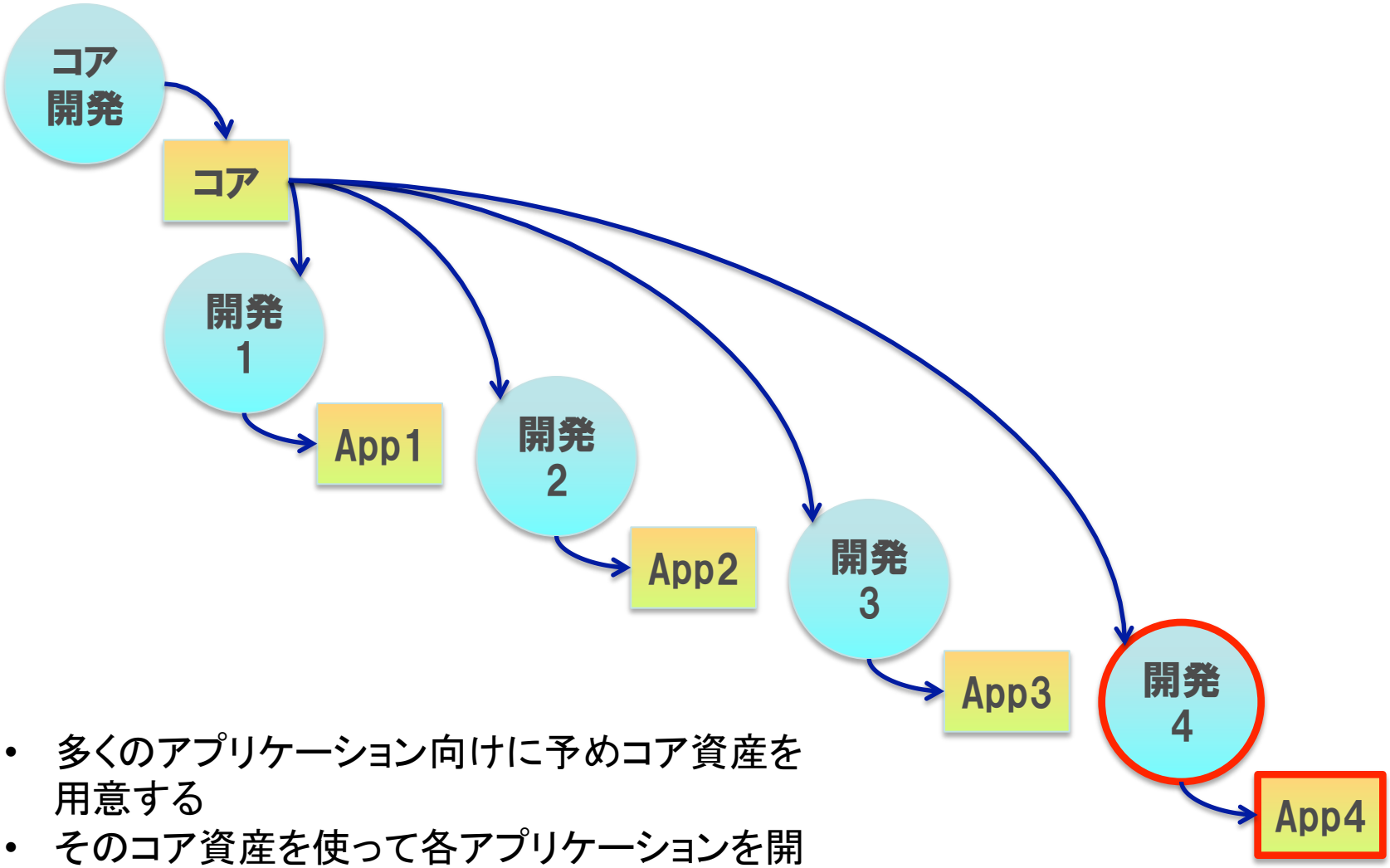
- アプリケーション開発はコア資産を入力とし、コア資産開発には既存アプリケーションの一部が使われることもある

派生開発の様子



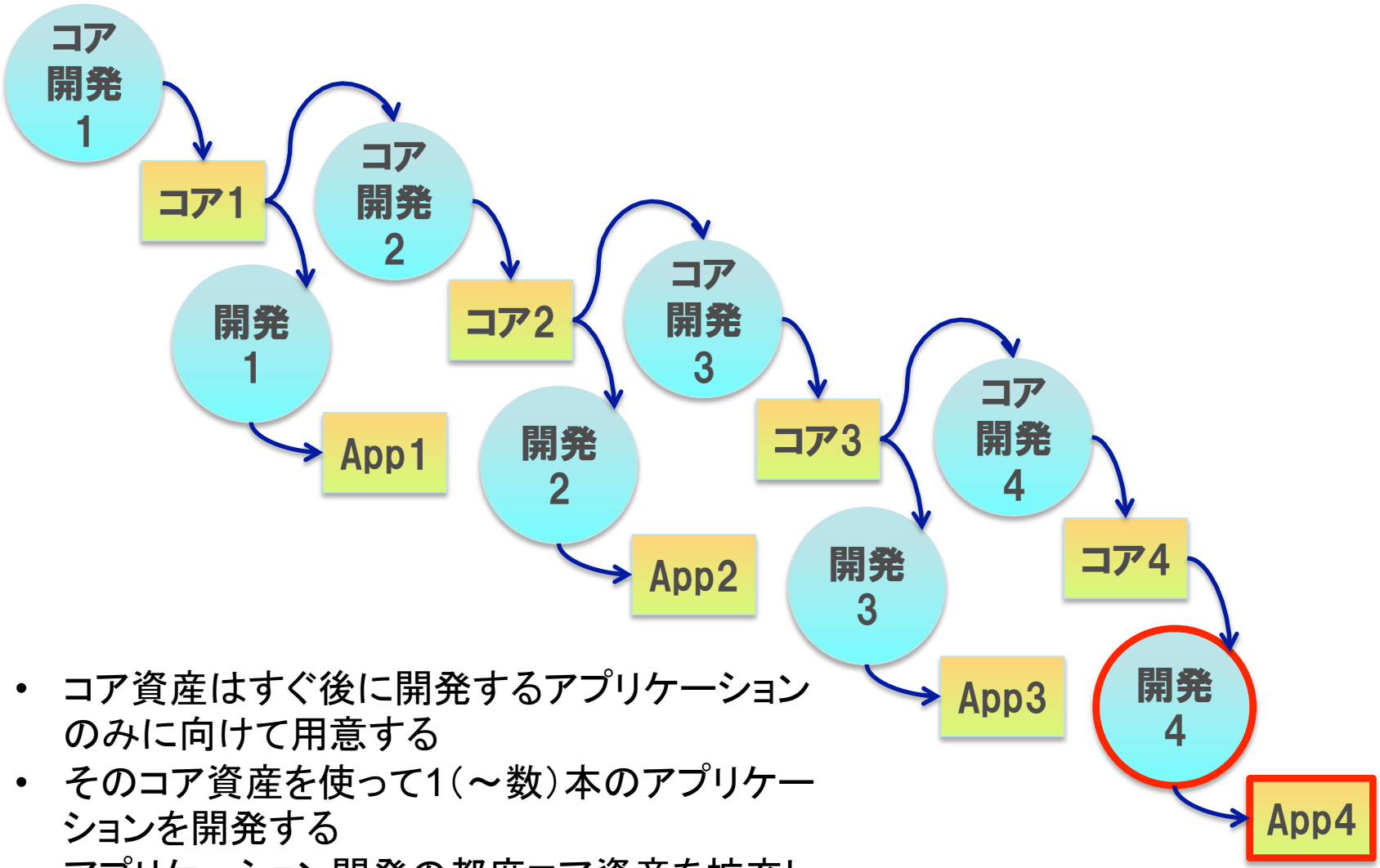
- 既存のアプリケーションを入力とする開発

PLE（事前準備式）の様子



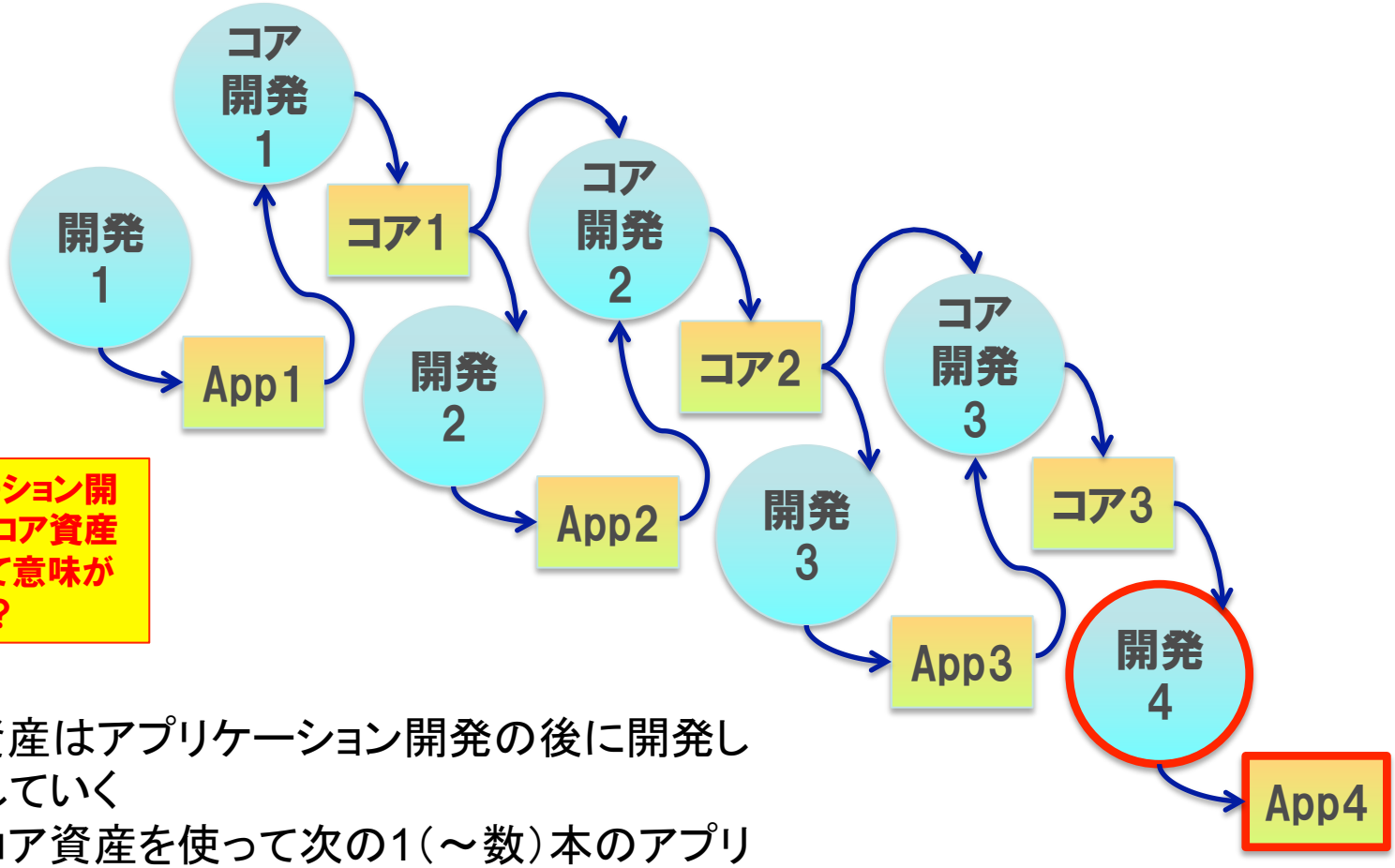
- 多くのアプリケーション向けに予めコア資産を用意する
- そのコア資産を使って各アプリケーションを開発する

PLE(事前都度対応式)の様子



- コア資産はすぐ後に開発するアプリケーションのみに向けて用意する
- そのコア資産を使って1(～数)本のアプリケーションを開発する
- アプリケーション開発の都度コア資産を拡充していく

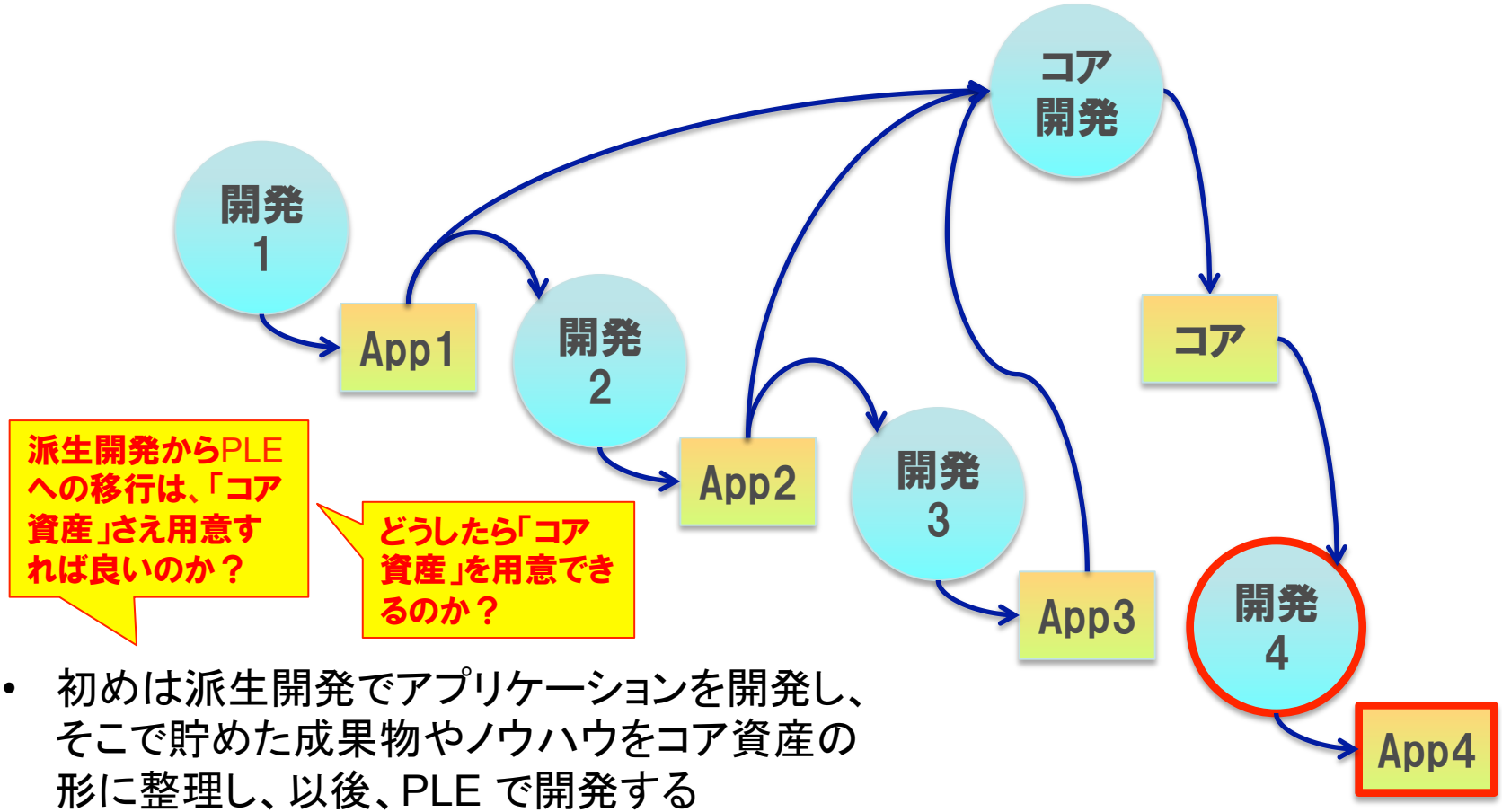
PLE(事後都度対応式)の様子



アプリケーション開発の後にコア資産を拡充して意味があるのか？

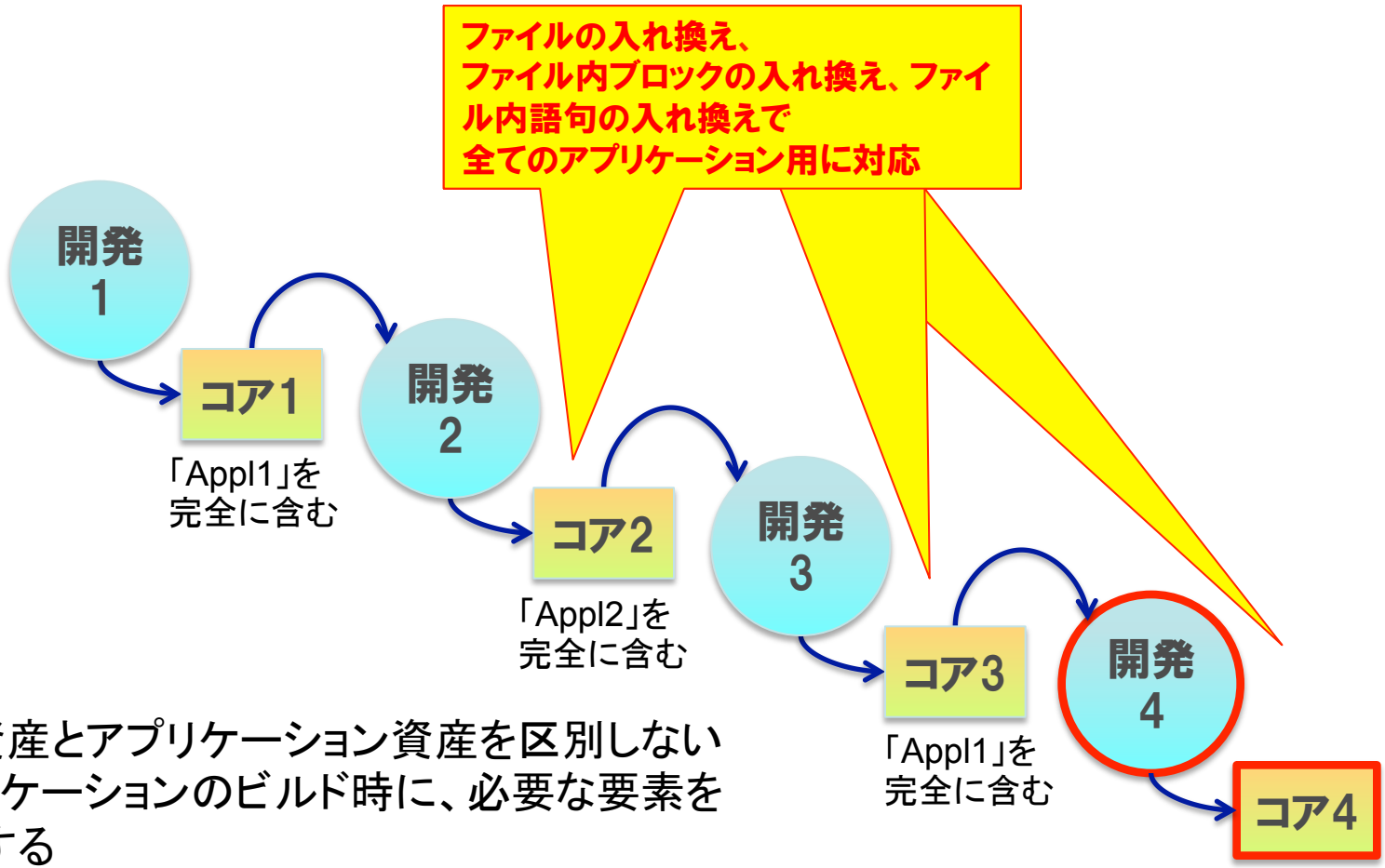
- コア資産はアプリケーション開発の後に開発し拡充していく
- そのコア資産を使って次の1(～数)本のアプリケーションを開発する

派生開発→SPLE 移行の様子(例)



- 初めは派生開発でアプリケーションを開発し、そこで貯めた成果物やノウハウをコア資産の形に整理し、以後、PLE で開発する

PLE(統合・都度対応式)の様子



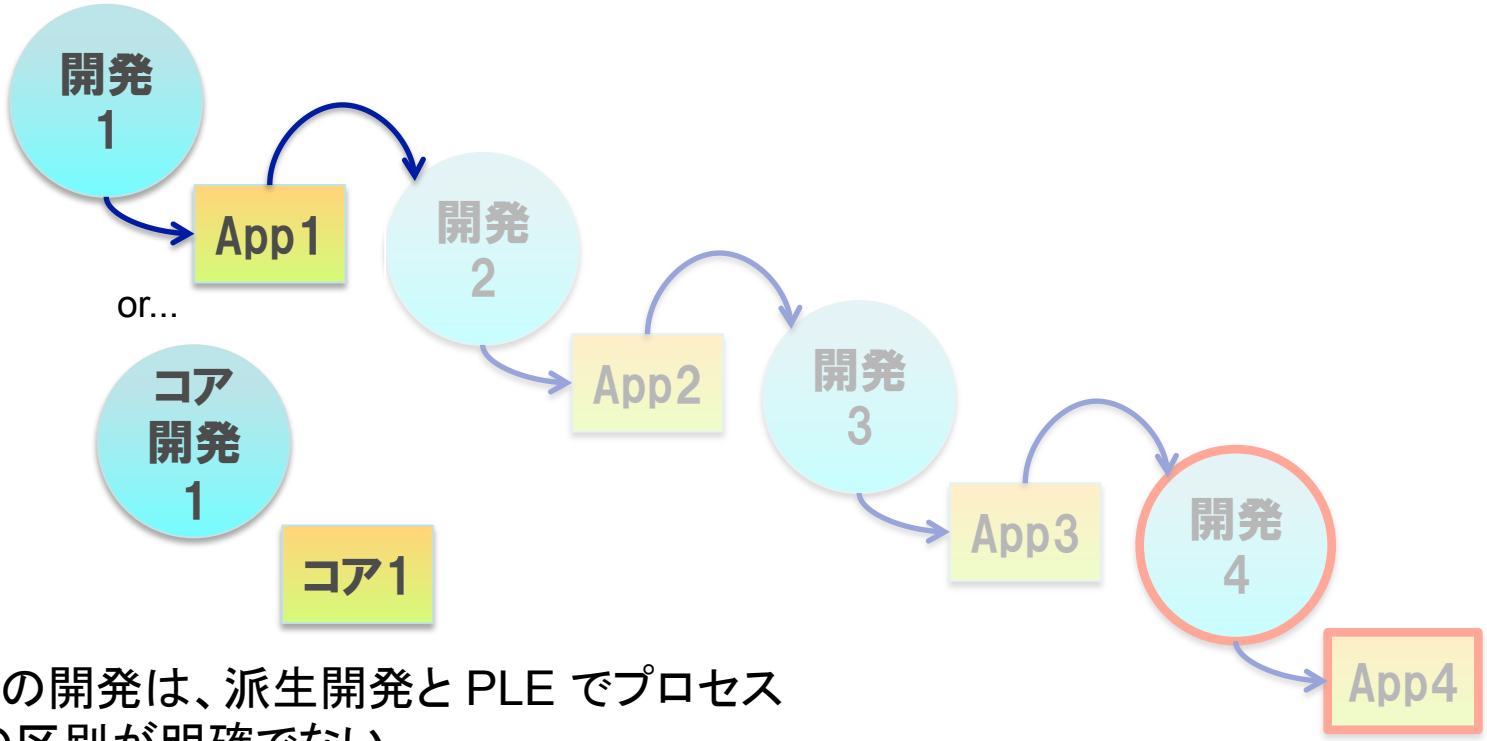
- コア資産とアプリケーション資産を区別しない
- アプリケーションのビルド時に、必要な要素を構成する

PLE では各アプリケーションを共通資産(コア資産)から作る

派生開発ではアプリケーション間の共通資産を前提としない

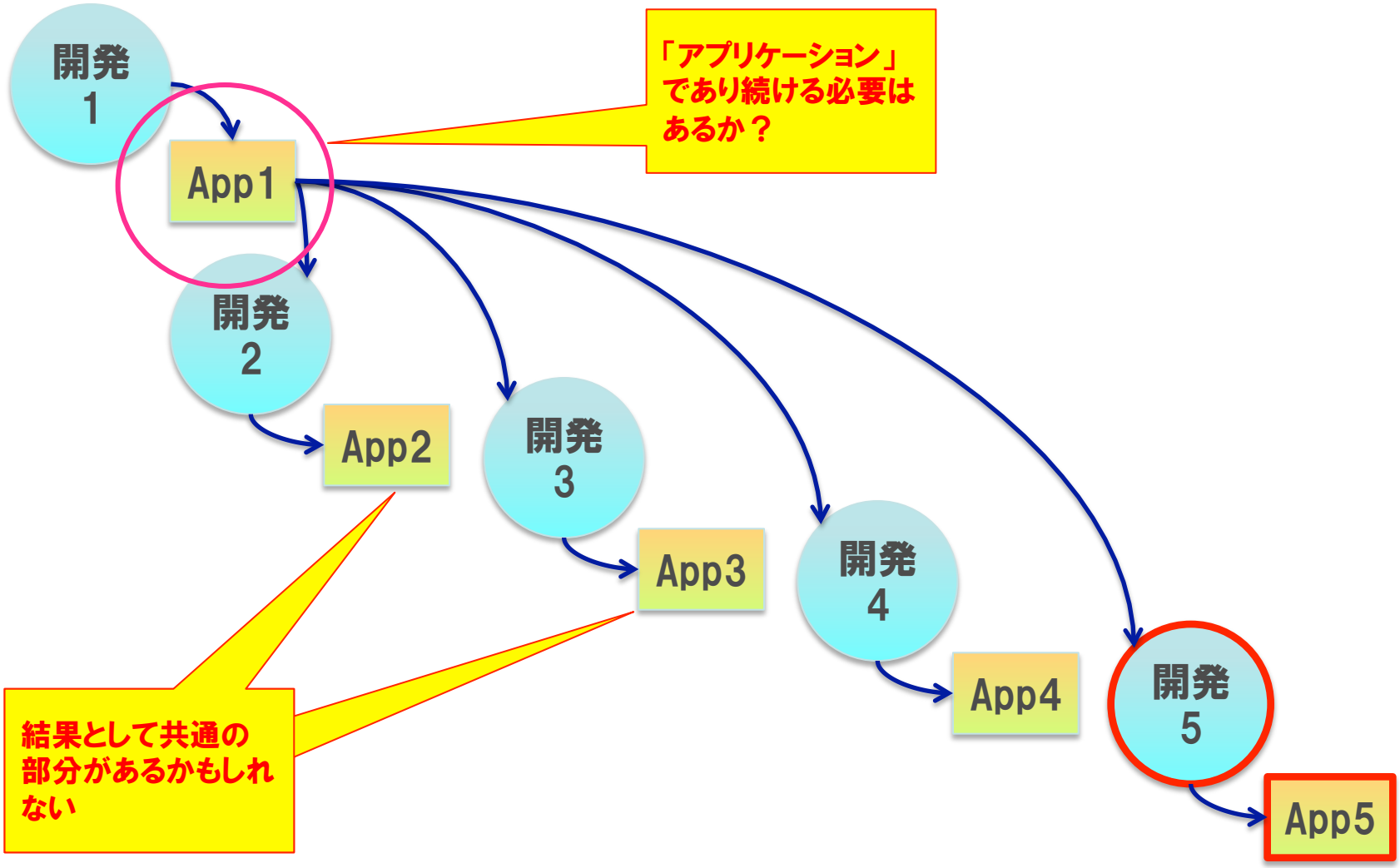
だが派生開発でもアプリケーション間の共通性はある…

派生開発？ PLE?



- 1回目の開発は、派生開発と PLE でプロセス上での区別が明確でない

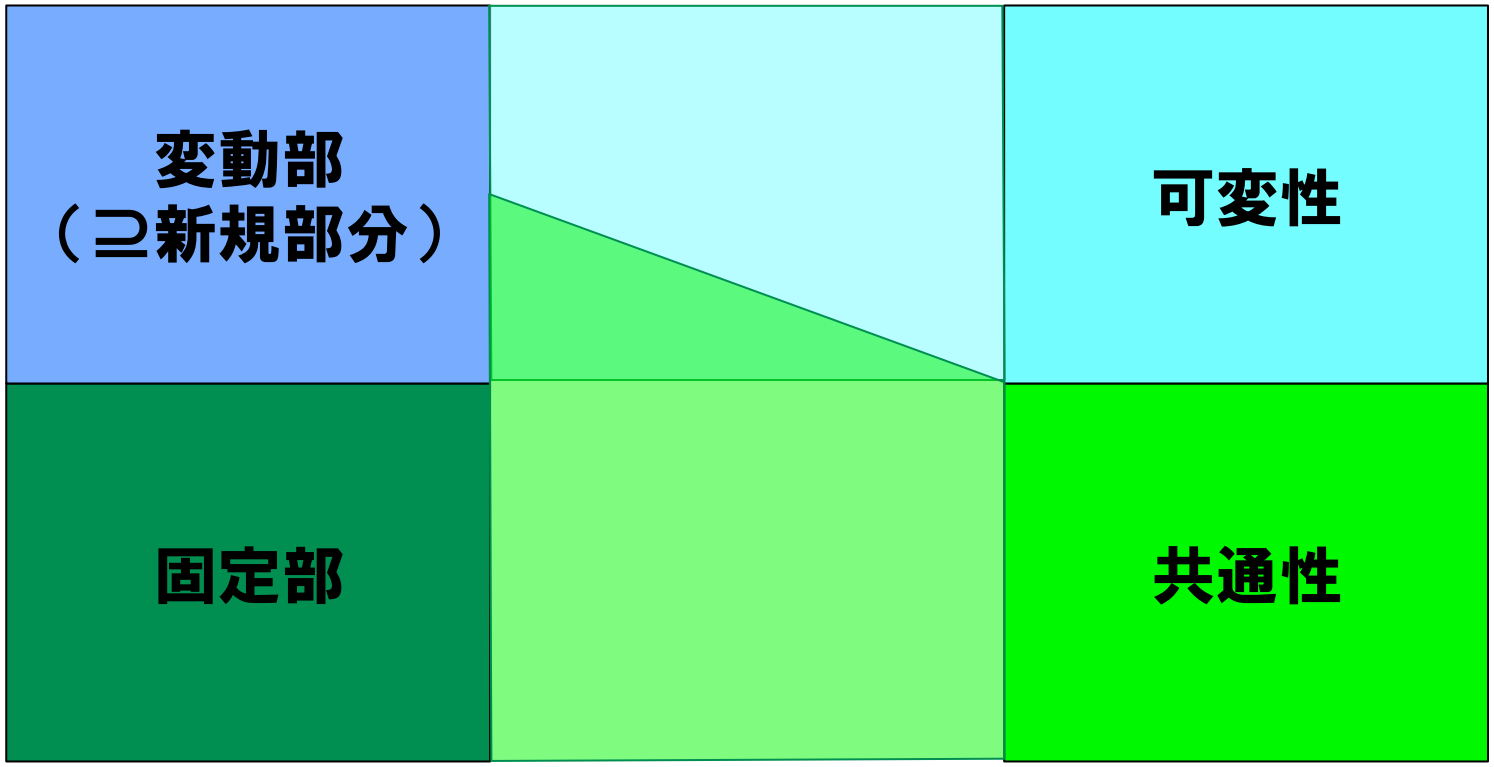
並行派生開発



何が違うのか？

共通性とは？

「固定部・変動部」との違い



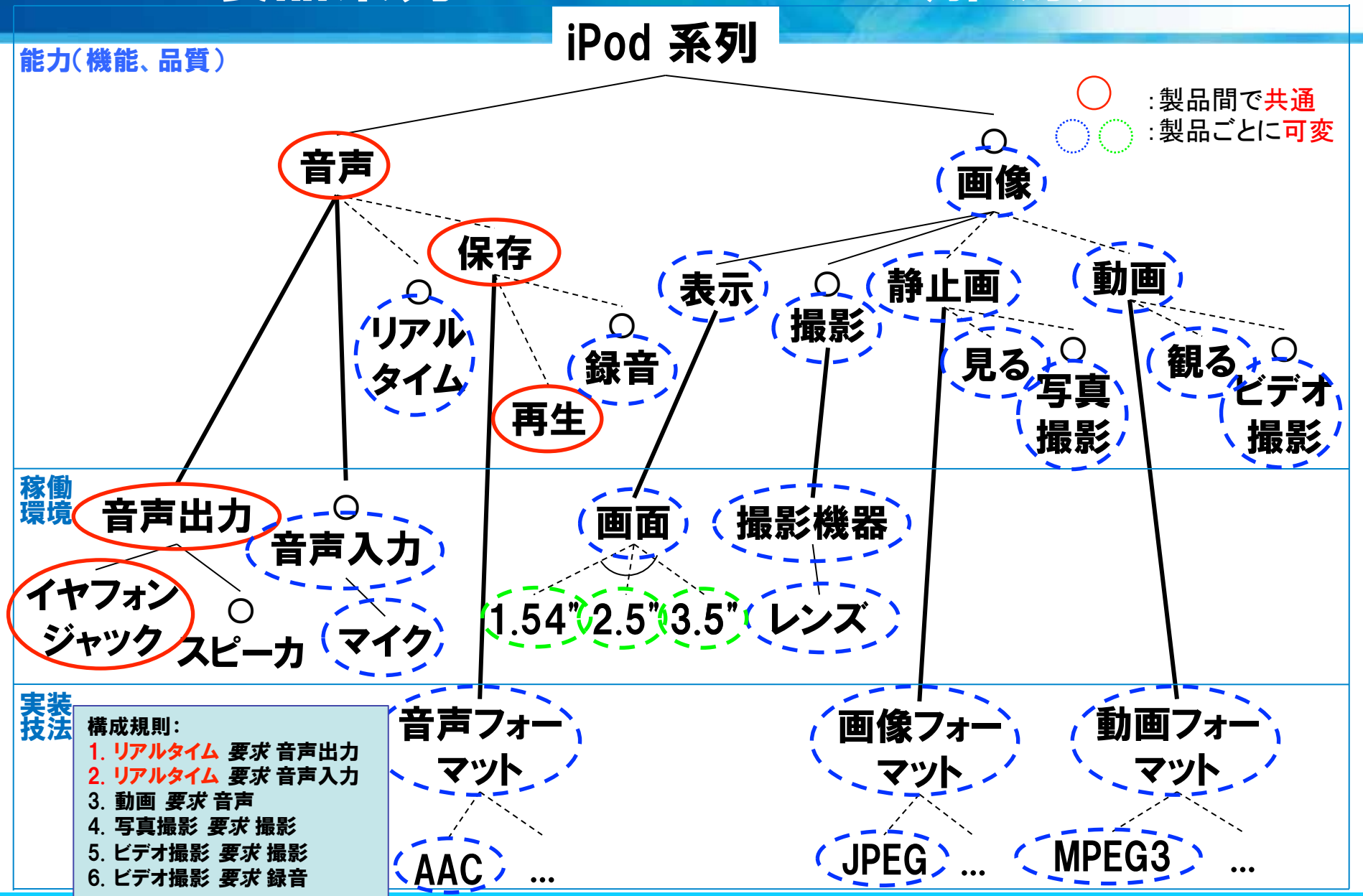
何が共通化がわかっていれば
開発が効率化できる

フィーチャ、フィーチャモデル

- フィーチャ
  - アプリケーションを特徴づけるものとして、よく「フィーチャ」という単位が使われる
  - そのドメインにおける専門家やユーザの「語彙」の一つ一つが「フィーチャ」の候補となりうる
    - 自動車ドメインでの例：ドア、エンジン、カーナビ、ブレーキ、ブレーキ制御、…
    - AV 製品ドメインでの例：再生、録画、お任せ録画、メディア、番組表、…
- フィーチャモデル
  - フィーチャ間の関係を表現したモデル
  - 対象がプロダクトライン（製品系列等）の場合、複数のアプリケーションを表現することになるので、共通なものとそうでないものを区別する必要がある

iPod 製品系列のフィーチャモデル（推測）

（部分）



フィーチャモデリングのアプローチ

- ドメイン指向
  - そのドメインにある要素をモデリングする
  - 例：エンジン、ハンドル、ドア、運転、加速、制動、走行性能、安全性、…
  - 全体の把握
- アプリケーション指向(製品指向(
  - 個々のアプリケーションにある(べき)要素をモデリングする
  - そのドメインをモデリングしたものに近づいていく
- トップダウンとボトムアップ
  - ドメイン熟達者がいればドメインモデルから
  - そうでなければアプリケーション間差異を取り込んで
- 事前準備と都度対応
  - アプリケーションフィーチャを予め集積して事前準備を行なうこともある
  - ドメインフィーチャを都度対応しながら都度対応する事もある

派生開発 → PLE の移行

PLE への移行に際して必要な事

- 次の三つが SPL を形作る
  - 共通性と可変性
    - 製品系列内で共通な点と製品ごとに異なる点

    →フィーチャ分析

    →可変性実現技術
  - コア資産とアプリケーション
    - 製品系列で共有するために作るものと、それを基にして作る各製品

    →プロセス構築技術
  - アーキテクチャとコンポーネント
    - 全体の大枠・仕組みと中身

    →アーキテクチャ

- 現状の評価
  - フィーチャ分析（必須）
    - 大きな共通性を見いだしかつ対応すべき可変性を見定める能力
  - 柔軟なアーキテクチャ…
    - …を設計する能力（新規開発）or
    - …に再構築する能力（既存資産活用）
  - 既存資産を…
    - 再利用可能な形に改修する能力
  - プロセス構築
    - コア資産とアプリケーションを開発・管理していく能力

アーキテクティング能力について

- 従来から、PLE にはアーキテクチャが死活的に重要であると言われている
- しかしアーキテクティングは PLE でなくても重要
- この能力が高くないから PLE が実施できないということはない
- もし、今までアプリケーションのバリエーションを生み出せているのだとしたら、そこにコア資産と、フィーチャモデルと、そこからの資産への対応を加えれば、開発を効率化する PLE となる
- ここでは柔軟性の高いアーキテクチャは必須ではなく、むしろ「可変性を持つアーキテクチャ」が重要となる

参考：可変性のメカニズム例（アーキテクチャレベル）

Svahnberg* らがアーキテクチャレベルで可変性をサポートするメカニズムを分類した[Svahnberg00]

- 継承：
  - 製品ごとに異なる/拡張された振る舞いのメソッドを持たせる
- 拡張と拡張点：
  - コンポーネントの一部を更なる振る舞い/機能で増強する
- パラメータ化：
  - コンポーネントの振る舞いがパラメータによって抽象化され、ビルド時にそれを固定する。マクロやテンプレートを利用
- 構成およびモジュール間接続のための言語：
  - ビルド時の構造を定義する
- 生成：
  - コンポーネントの特性を定義する高次言語が使える
- 異なる実装のコンパイル時選択：
  - コンポーネント中で #ifdef によって変数を参照して異なる実装が選べるようにする

* M. Svahnberg and J. Bosch, “Issues Concerning Variability in Software Product Lines,” pp. 50-60, Proc. Third International Workshop on Software Architectures for Product Families. Las Palmas de Gran Canaria, Spain, March 15-17, 2000. Heidelberg, Germany: Springer LNCS, 2000

参考：可変性のメカニズム例（コンポーネントレベル）

Anastasopoulos* らがコンポーネントレベルで可変性をサポートするメカニズムを分類した[Anastasopoulos00]

- 集約/委譲
  - 構成要素の一部や他のオブジェクトにフィーチャを実現させる
- 継承
  - 前ページに同じ
- パラメータ化
  - 前ページに同じ
- オーバーロード
- Delphi言語における特性
  - 属性値や属性の種類によって可変性を実現
- Java における動的クラスロード
- 静的ライブラリ
  - ライブラリを替える
- 動的リンクライブラリ
  - ライブラリを動的に替える
- 条件付きコンパイル
  - コンパイルスイッチ等で
- フレーム技術
- リフレクション
- アスペクト指向プログラミング
- 設計パターン

Anastasopoulos, M. & Gacek, C., “Implementing Product Line Variabilities” (ISES-Report No 089.00/E, Version 1.0), Kaiserslautern, Germany: Fraunhofer Institut Experimentelles Software Engineering, November 6, 2000

まとめ

- PLE のプロセス
  - 派生開発にプロセスに似ているところがある
  - だが共通性を基にしたコア資産は PLE の特徴
- コア資産
  - アプリケーション間の共通性を把握することが効率につながる
  - その上でどこが可変なのかを把握する
  - そのためにフィーチャモデルがある
- フィーチャモデル
  - 各アプリケーションを特徴づける要素を整理したもの
  - 整理は対象ドメインの分析から、または各アプリケーションの特徴から、またはその両方から考えていく事ができる
- PLE への対応
  - 事前準備式、都度対応式、そしてそれらの組合せの形態がある
  - 自分達の能力、条件等にそって選べば良い