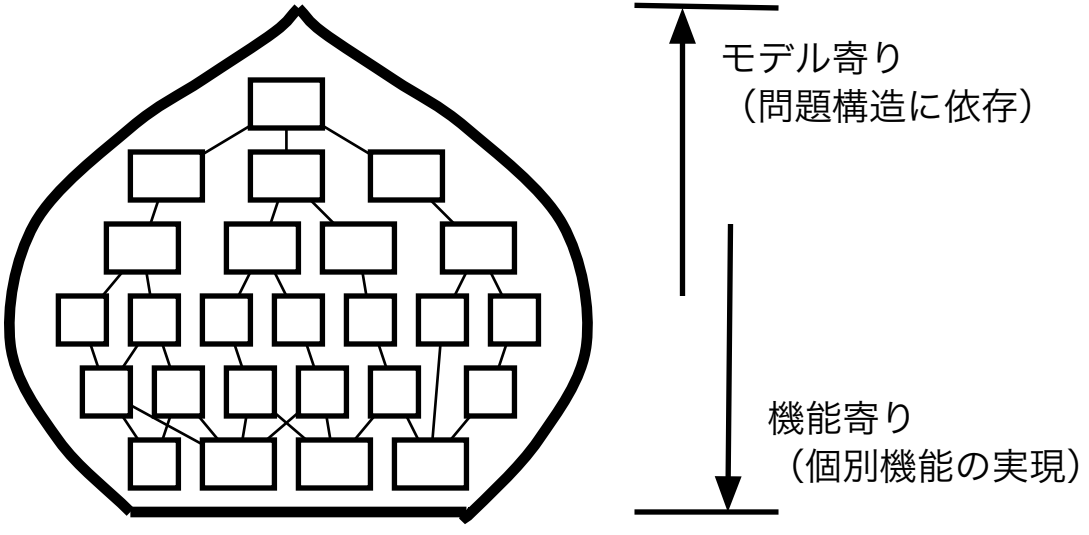
ソフトウェア再利用のための分散作業支援方式の研究

荻原剛志，庭山直幹，上嶋明（神戸大学）
{ogihara, nya, uejima}@**cs25.scitec.**kobe-u.ac.jp

ソフトウェアモジュールのモスク構造



◎上位モジュールは、そのソフトウェアの問題構造に深く依存しており、汎用性は低い。（再利用は抽象度の高いデザインパターンで）

◎下位モジュールは、上位モジュールを実現するための道具立てを提供しており、汎用性が高い。（コードレベルの再利用がしやすい）

（関数モジュールでも、オブジェクト指向モデルでも同じ）

実世界

要求

抽象化

OOA

詳細化

追加

追加

OOD

UI

問題領域

データ管理

タスク管理

実装

再利用

OOP

再利用

計算機

オブジェクト指向は再利用しやすいか？

◎モデルを構成している主要なオブジェクトは、そのソフトウェアの問題構造に深く依存しており、汎用性は低い。

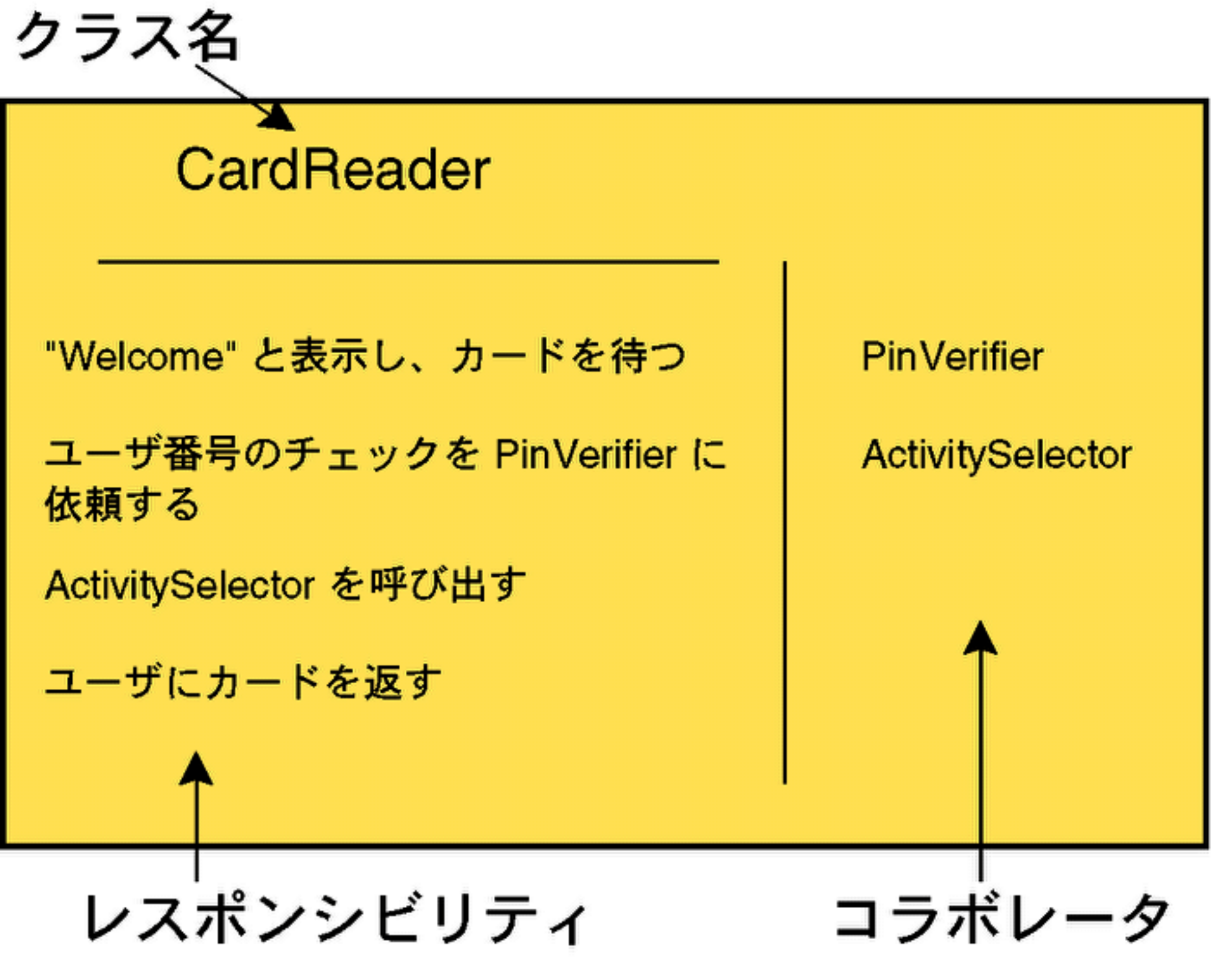
◎主要なオブジェクトを実現するために用意されたオブジェクトは、汎用性が高い。

◆オブジェクト指向モデルでも、すべてのクラスが再利用可能なわけではない。

◆オブジェクトは相互の関係の上で意味を持つため、個々のオブジェクト（クラス）のみでは再利用性が低い。

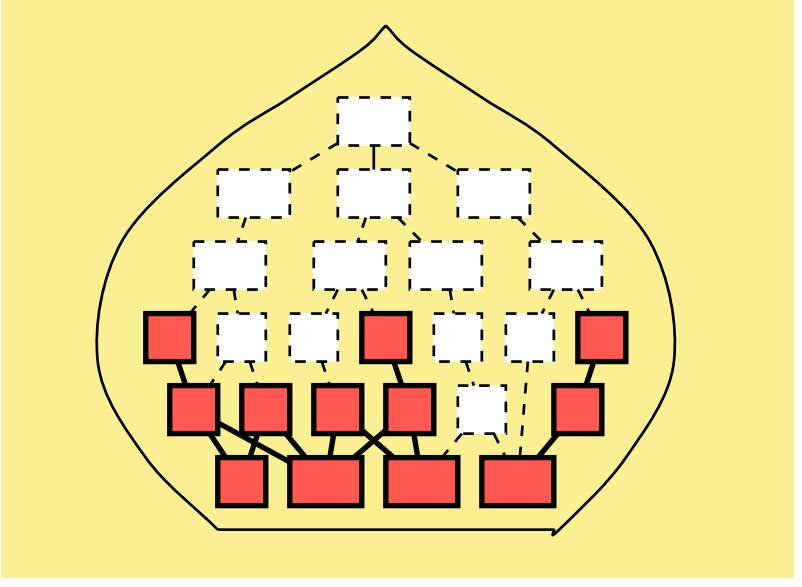
→ ボトムアップ的な要素が大きい再利用では不利？

→ 型の制限の厳しい言語は特に再利用が困難？


クラス名
CardReader
"Welcome" と表示し、カードを待つ
ユーザ番号のチェックを PinVerifier に依頼する
ActivitySelector を呼び出す
ユーザにカードを返す
PinVerifier
ActivitySelector
レスポンシビリティ
コラボレータ

フレームワークとは？

◎ある特定の問題領域において典型的によく利用される「道具立て」としてのオブジェクト、およびその半完成形モデル。



再利用の実体験例

画像ビューア＆操作ツール「***ToyViewer***」

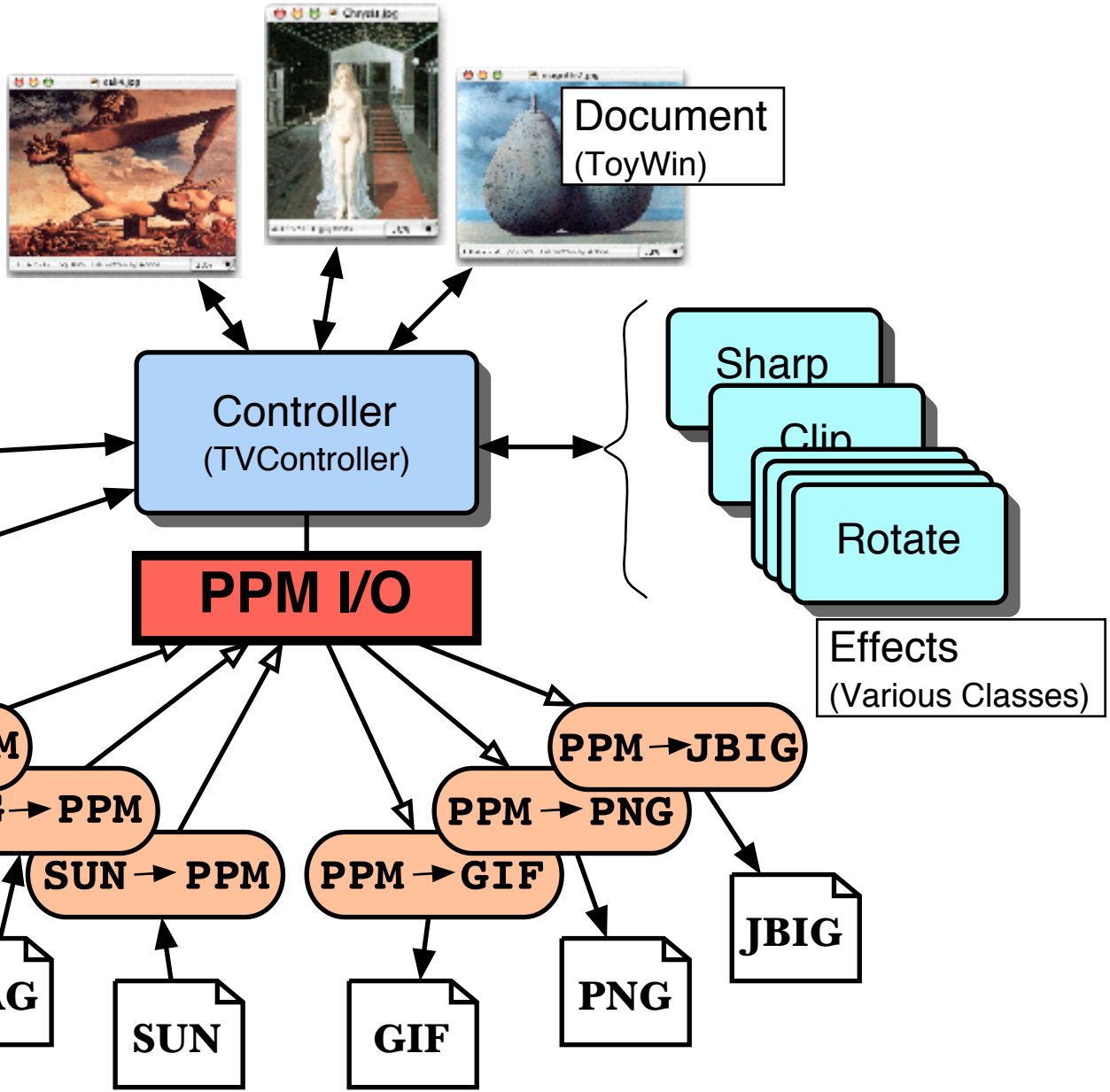
機能

・各種画像ファイルの表示、形式変換して保存
・画像に対する各種操作（強調、濃度変化、拡大・縮小など）
・その他

開発作業

・NeXTstep, OPENSTEP, Mac OS X 上で開発
・Ｃおよび Objective-C を使用（約３万行）
・UI部分には Interface Builder を利用
・画像操作部分のルーチンは再利用できないか？
・各種画像ファイルの入出力は再利用できないか？

Overview of *ToyViewer*



画像操作部分の再利用

・結果的には全くうまくいっていない

・原因

+ 画像の内部形式が異なる
+ ツールとしての使い勝手に合致しない
+ 再利用しようという気にならないコードも多い
（動かない、冗長、汚い、明らかなバグ）

画像ファイルの入出力の再利用

・かなりうまく実現できている

・PPM(Portable PixMap)形式へのフィルタプログラムを利用

+ プロセスを起動し、パイプでデータをやりとりする
+ PPMを一部拡張している

・入力画像形式は、設定ファイルにフィルタ名を追加するだけ

・出力画像形式は、プログラム自体に記述を追加する必要あり

クラスの再利用

・うまく再利用できたのは次の２種類

1. 独立性の極めて高いクラス
    例. RGB型のような基本的なデータ型
    例. 行列演算のように抽象度の高い操作を行うクラス

2. 共通したフレームワーク内のクラスの機能を強化したもの
    例. ウィンドウクラスに全画面表示の機能を付け加えたもの
    例. 項目がドラッグで移動できるようにしたテーブルクラス

その他のクラスや関数などは、考え方や実現方法を参考にして、
自分でコーディングし直した方が結局近道であった。

再利用は可能なのか？

・特定のソフトウェアのコンテキストに依存しない、

あるいは

・共通した既知のフレームワークに従っている

のであれば、ソースコードレベルの再利用がしやすい。

そうでなければ、

・モデルの全体像（ソフトウェア・アーキテクチャ）を参考にする

・デザインパターンを抽出する

・使っているアルゴリズムを参考にする

・コードに手を加えて再利用可能な部分を切り出す

など。

コマンドライン・プログラムへの注目

コマンドライン・プログラムは再利用の条件に合致している

- ・コンテキスト独立
  - 基本的にそれ自体のみで動作する
- ・既知のフレームワーク
  - UNIXの場合：
    - ・標準入出力とリダイレクション、パイプライン
    - ・オプションの記述方法
    - ・環境変数の使い方

シェルスクリプト、あるいは Apple Script は、実は同様な再利用で
成功している例では？

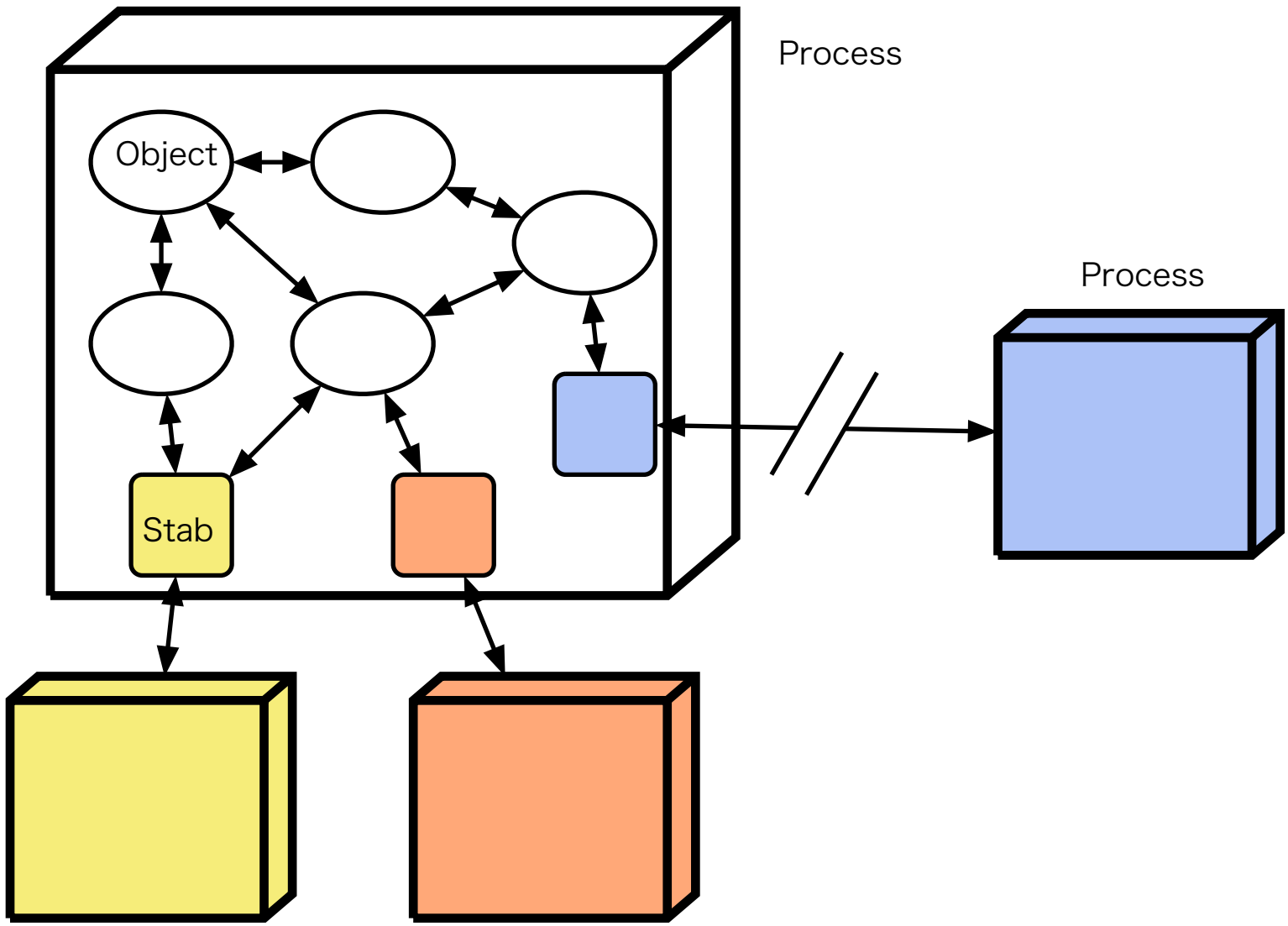
コマンドライン・プログラム利用のメリット

1. **インタフェースが簡単**
   - 反面、再利用できるのはフィルタなどに限定されやすい
2. **ドキュメントが付属している**
   ・ネットワークで検索する場合にも有利
3. **簡単に利用できる**
   ・簡単に試用、テストができる
   ・テスト用のドライバやスタブが不要
4. **プラットフォームに依存する部分が少ない**
5. **広く使われていれば「枯れた」コードが期待できる**

コマンドライン・プログラム利用のデメリット

1. **インタフェースが簡単すぎる**
   - 単純なフィルタ程度しか再利用の対象にできない
2. **不必要な機能も抱え込んでいることがある**
3. **キャラクタ端末に依存する部分がある**
   - メッセージやエラー表示、ユーザ入力を求める部分
4. **かなりUNIX文化に依存する**
   - 最近は GUI流行りで、CLIは減少傾向？
5. **プロセスの起動はやや重い**

プロセス単位の再利用のイメージ



コマンドライン・プログラム再利用に向けて

1. **使用方法の統一的な記述方式の提案**
    - 入出力データの形式
    - （入出力以外の）機能、作用の記述
    - オプションの指定方法
    - XMLなどによる記述？

2. **マニュアルなどのドキュメントの利用**

3. **検索方法**
    - キーワードによる検索
    - 入出力データの表現に対するマッチング

4. **再利用を支援するためのフレームワークの提供**

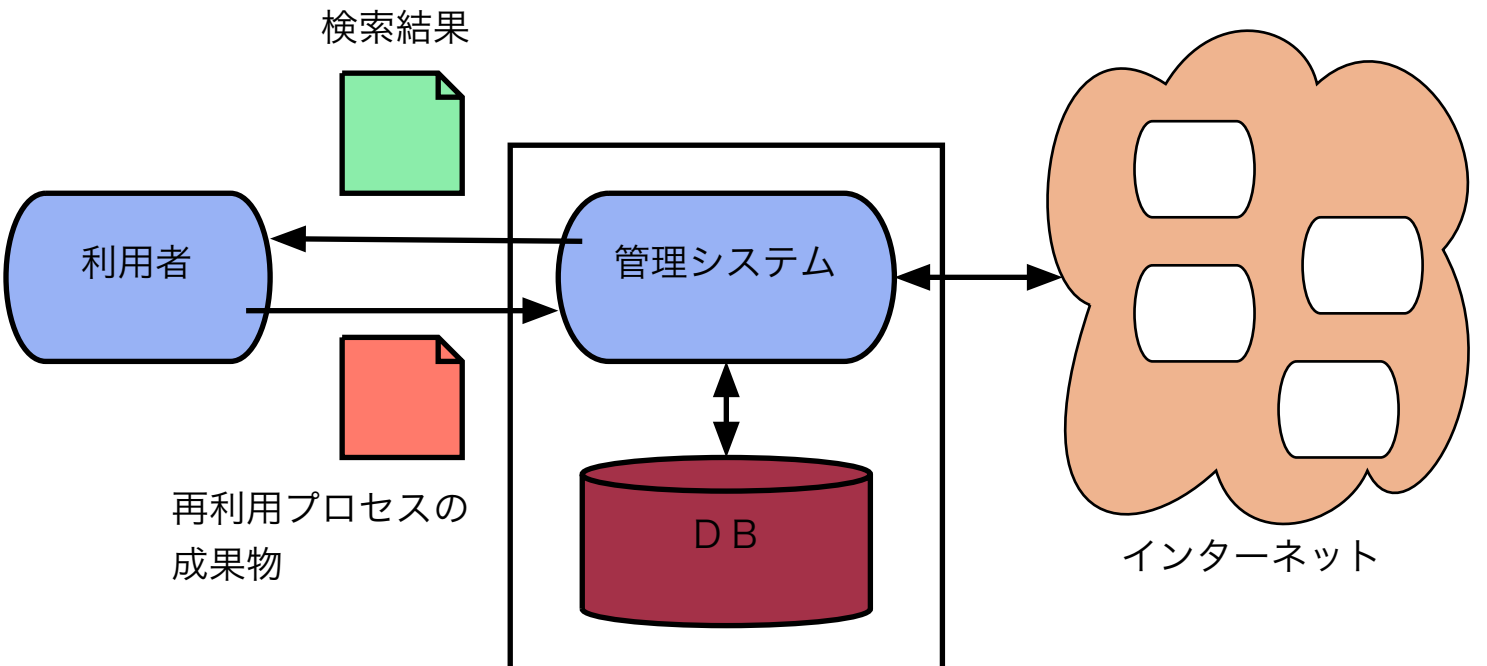
ソフトウェア再利用のための分散作業支援

ソフトウェア再利用を、
・再利用する側と
・コード（に関する情報）を提供する側
の共同作業と考える

再利用のプロセスで、
・そのコードが役にたったかどうかの評価
・コンテキスト独立性の高いプロダクトをフィードバック
してもらう仕組みが必要

利用側のメリット：再利用可能なコードが入手できる
提供側のメリット：コードの再利用性を高めることができる

分散作業のイメージ



ソフトウェア再利用のための分散作業支援*(cont.)*

課題

コードを、再利用資源として定量的に特徴付ける方法は？

現在の再利用は全面的に人手に頼っている
→ 定型業務（ワークフロー）化して支援できないか？