アーキテクチャーの様式

FU14 ソフトウェア工学概論

第5回

吉岡 廉太郎

前回の内容

- 設計は創造的プロセス
  - 課題に対して解決策を創造する → 始めに、アーキテクチャーを決める
- アーキテクチャーとは
  - システムを構成するソフトウェアユニットの関係＝「モデル」
- アーキテクチャーを決定する第一ステップ
  - 課題を意味的単位に分解する

  A) 過去の解決策を利用する

  B) 分解手法を利用する
- 過去の解決策を活用する
  - クローン、参照モデル
- 手法に従って分解する
  - アーキテクチャ・スタイル： 分解方法の方針
  - 分解手法： ソフトウェアユニットに分解する具体的ガイドライン
  - アーキテクチャの観点： 一つの分解手法では捉えきれない性質

今日の内容

- アーキテクチャ設計＝モジュール化
  - 構築するシステムをモジュールの組合せとして表現すること
- アーキテクチャの様式
  - 代表的なソフトウェアユニットの構成法を学ぶ

前回は課題の“分解”が中心

- 課題を意味的単位に分解する考え方

- 今日は、アーキテクチャの様式
  - システムをソフトウェアユニットで構成する具体的なパターン(=様式)を学びます

アーキテクチャの様式

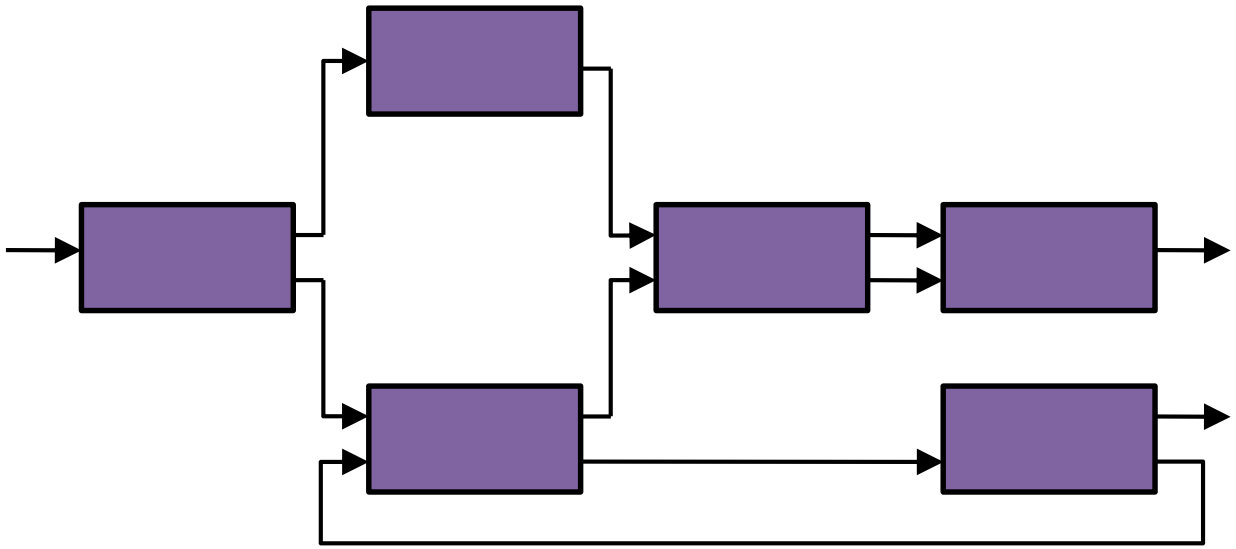
9.4.3

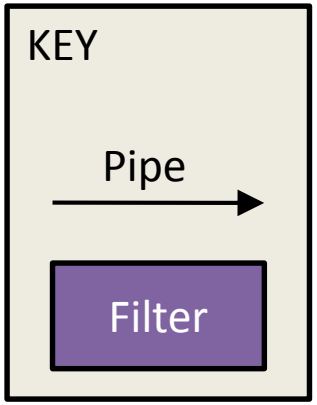
1. Pipe-Filter
2. Client-Server
3. Peer-to-Peer
4. Publish-Subscribe
5. Repository
6. Layering

※一般に代表的な“アーキテクチャ”として知られているのがこれら様式です

Pipe-Filter

- 構成
  - Pipe: 入出力のデータストリーム
  - Filter: データの変換





Pipe-Filter

- 特徴
  - システムの処理をフィルターの重ね合わせと考える
  - 各フィルターは独立した単位＝再利用が可能
  - システムの保守と機能的拡張が容易
  - 並列実行を表しやすい

- 欠点
  - バッチ処理的システムに陥りやすい
  - インタラクティブなシステムには向かない
  - フィルター間で共通する処理に冗長性が生まれる

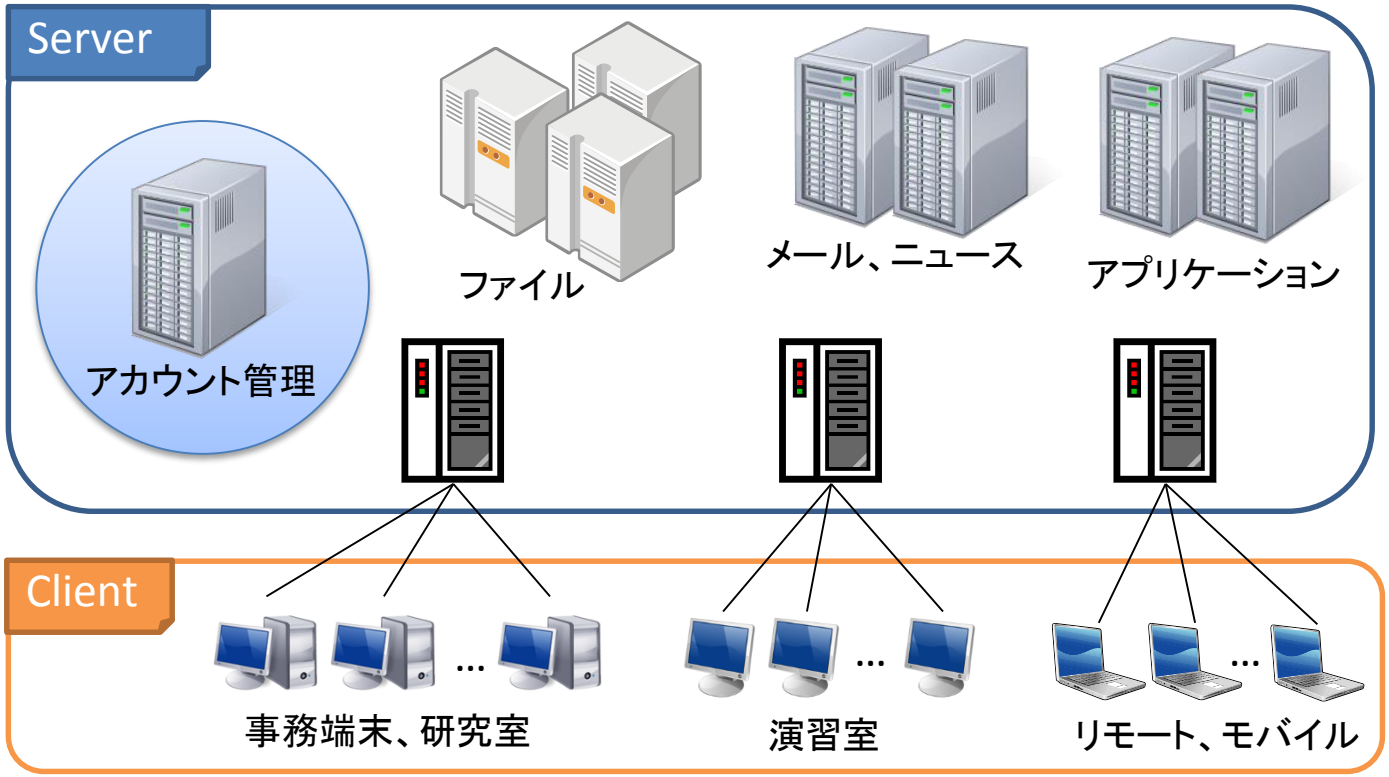
Client-Server

- 構成要素
  - Server：サービスを提供
  - Client：サービスを利用

- Serverとclient間でのやりとりの方法
  - 要求・応答プロトコル
    - Clientが要求し、Serverが答える
  - コールバック
    - Clientは、Serverの準備ができたら呼び出してもらう

Client-Server例 : AINS

- 会津大学の情報システム
  - 全学生および教職員が利用
  - 教育系、研究系、事務系
  - 24時間利用
  - 中央制御と分散処理の両方が必要



Peer-to-Peer (P2P)

- 独立して動作するコンポーネントの集合
  - コンポ.はServerとClientの役割を両方備える
  - すべてのコンポ.が、他のコンポ.に対して要求を出せる

- 特徴
  - 拡張性が高い（特に規模的な）
  - 高い処理能力のシステムを実現しやすい
  - 耐障害性が非常に高い

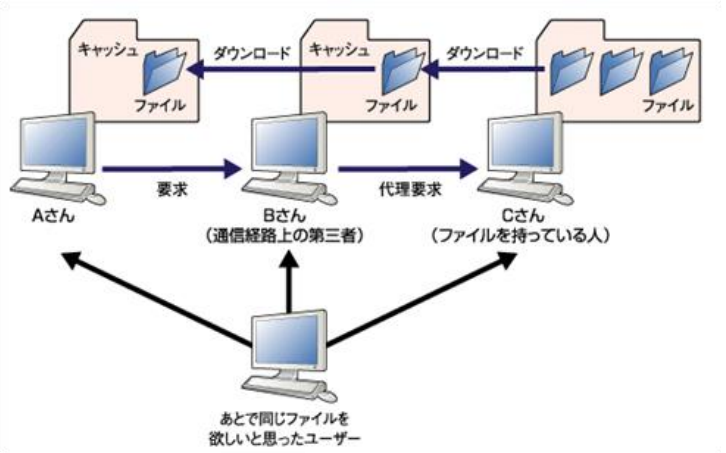
- 例：ファイル共有ソフト
  - Napster、Winny、BitTorrent、等

P2P例：Napster

- 各Peer（コンポーネント）はユーザのPC
  - 自分のPCにソフトをインストールすることでPeerになる
- Peerの特性
  - 固定IPが無い
  - ネットワークに接続している時間や接続の品質はばらばら
  - ユーザは技術者でなく利用者

- Peerのサーバー機能
  - 機能：コンテンツの管理、要求の交通整理
  - ユーザが登録するコンテンツを管理
  - 登録されたコンテンツを他のPeer上にも分散して保管する

- Client-Serverが有利な場合
  - コンテンツが頻繁に更新される
  - コンテンツへのアクセスを高速化する
  - コンテンツの品質を確保する
  - Peer間での信頼が必要



Publish-Subscribe（発行者-読者）

- ブロードキャストとイベント応答でコンポーネント同士がやりとりする
  - 興味のあるコンポーネントのイベントに登録できる
  - 登録をすると、イベントが起きたときに通知をもらえる

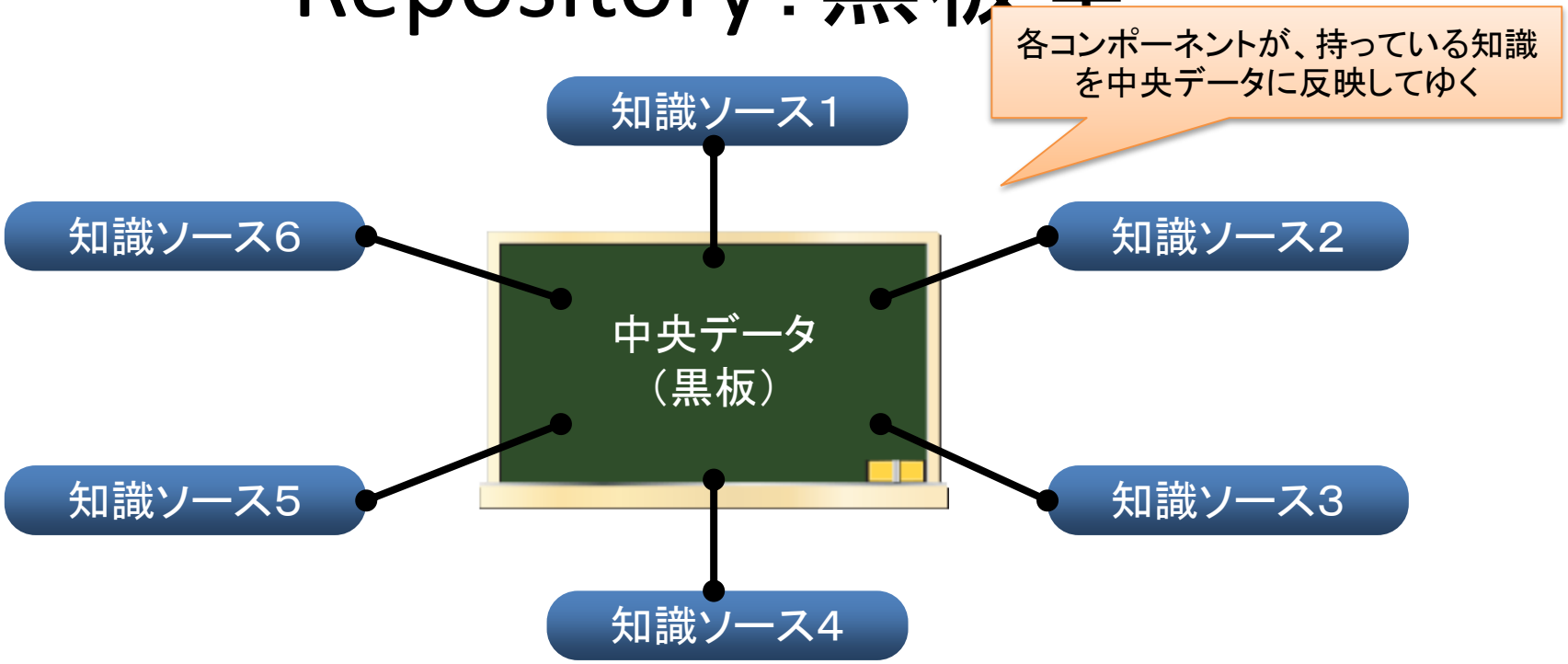
- 例：Javaのイベント・リスナー、暗黙的呼出
  - 関数とイベントを関連付けし、イベント発生時に呼び出す仕組み

- 特徴
  - システムの拡張が容易：コンポ.の“着脱”が他に影響しない
  - 再利用性に優れている
  - データ通信が難しい：共有メモリが必要
  - 妥当性検証が難しい：通知する側は受信側の事情を知らない

Repository

- 構成要素
  - 中央データ
  - データを操作する独立したコンポーネント
    - データの格納、取得、更新を行う
- コンポ.間のやりとりの実現方法
  - データベース型
    - データベースはServerの役割を果たす
    - コンポーネントは能動的に動作する
  - 黒板型
    - 中央データの状態に応じてコンポーネントが動作する
    - データを取得し、処理し、戻す（例、Wikipedia）
    - コンポーネントはデータに対して受動的に動作する

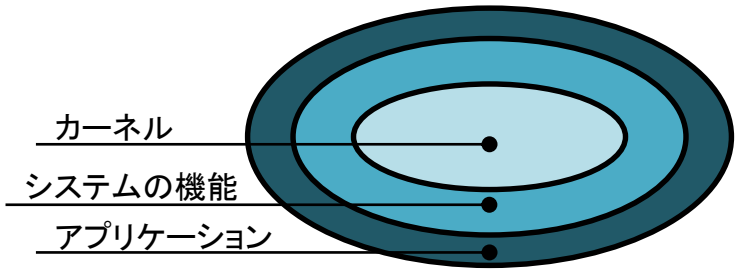
Repository：黒板型



- 特徴
  - 高い公開性：　広く利用できる
  - 中央管理がしやすい：品質、セキュリティー
  - 広く利用しやすいデータ構造を維持するのが難しい

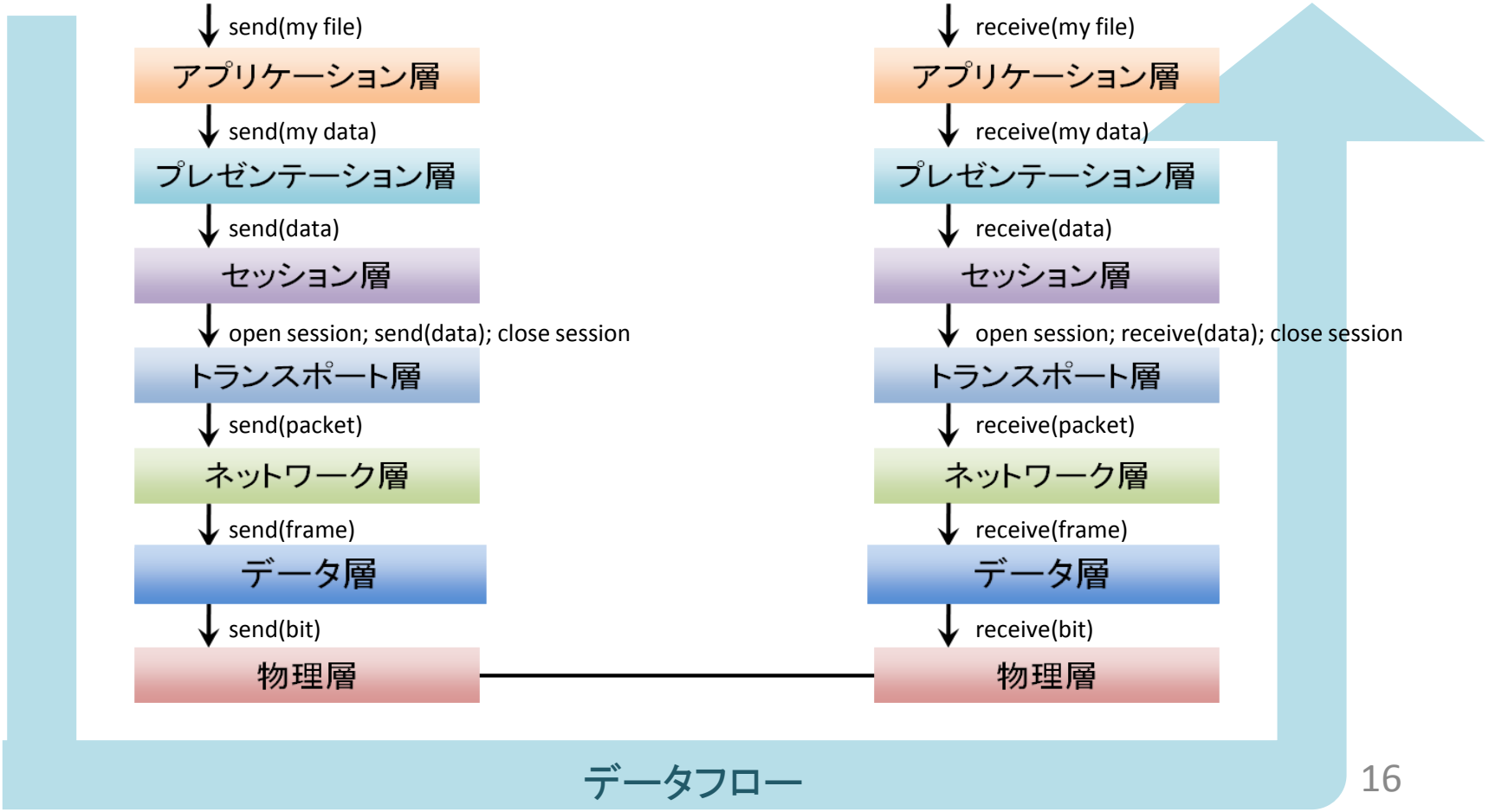
Layering

- 階層的に整理する
  - 各層は、外側の層に対するServerとなり、内側の層のClientとなる
  - 複数の層をまたがる通信も設計できる
- 通信方法
  - 層間でプロトコルを決める
- 利点
  - 抽象度での整理がしやすい：複雑な問題を階層に分割する
  - 拡張が容易：依存関係が前後の層に限られる
- 欠点
  - 階層構造に整理するのは難しい
  - パフォーマンス：層ごとに通信することが負荷や遅延になる



Layering

- 例：OSIモデル （Open Systems Interconnection）

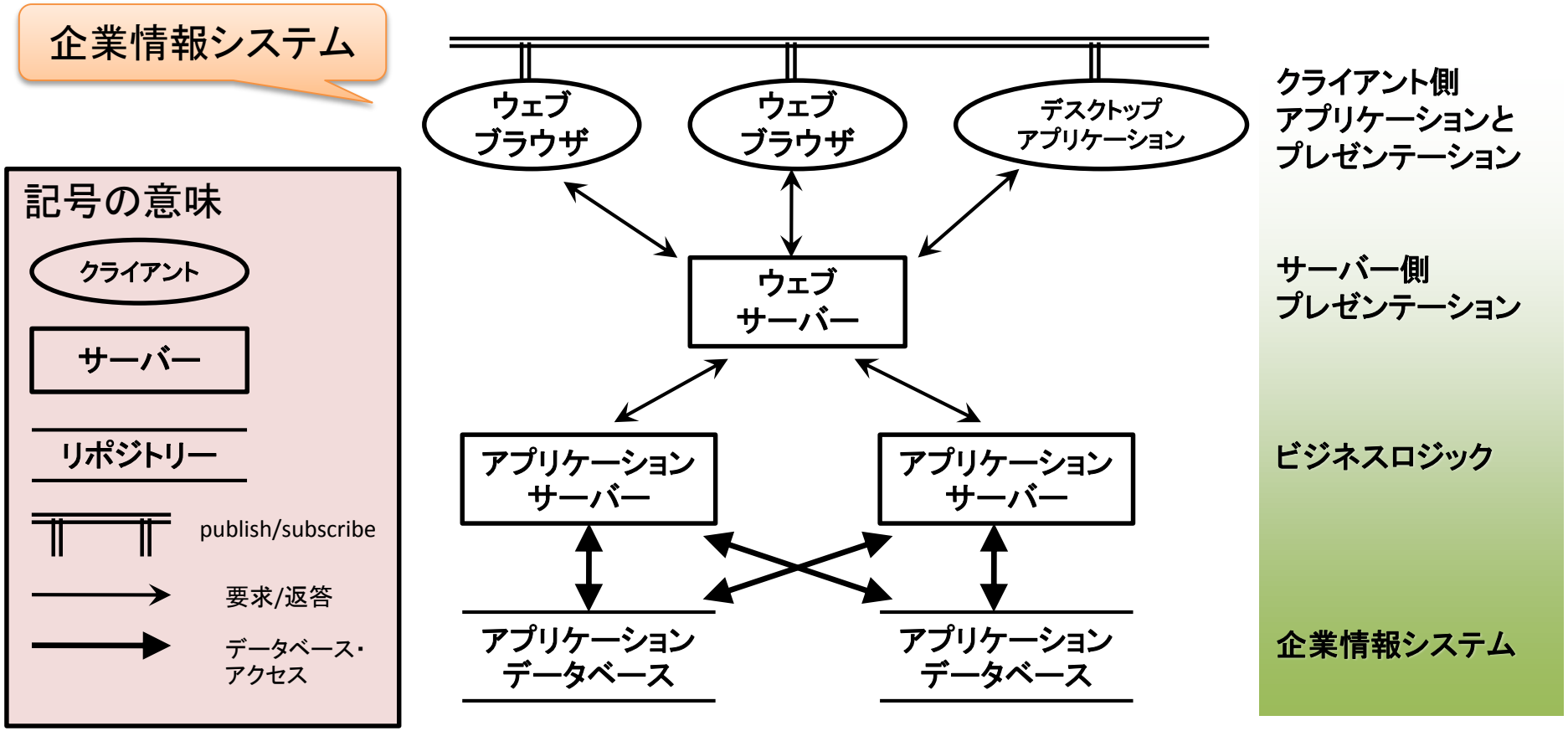


複数の様式を組み合わせる

- 一つの様式からなるアーキテクチャーは少ない
- 複数の様式の組み合わせ方
  - 全体構造と内部構造に異なる様式を用いる
    - 例）Client-Serverで、ServerをLayeringで構成する
  - 特定の機能、振る舞いに様式を当てはめる
    - 例）コンポーネント間の通信をイベント駆動にする
- 開発では、誰でも分かる説明が重要
  - アーキテクチャーを説明するのに、いずれかの様式に当てはめると分かりやすい
  - 様式に当てはめると整理される

様式の組み合わせ例

- 例）Publish-subscribe、client-server、repository



本日のまとめ

- アーキテクチャーの様式
  - システムの全体構成を考えるためのモデル
  - 最低限知っておくべき、一般的常識、共通言語
  - 通常は、複数の様式を組み合わせて使う

次回講義の事前学習：
講義資料を予習してきてください