分散システム基礎とクラウドでの活用

第１回(後): 実現基盤の概観

国立情報学研究所
石川　冬樹
f-ishikawa@nii.ac.jp

今回の内容

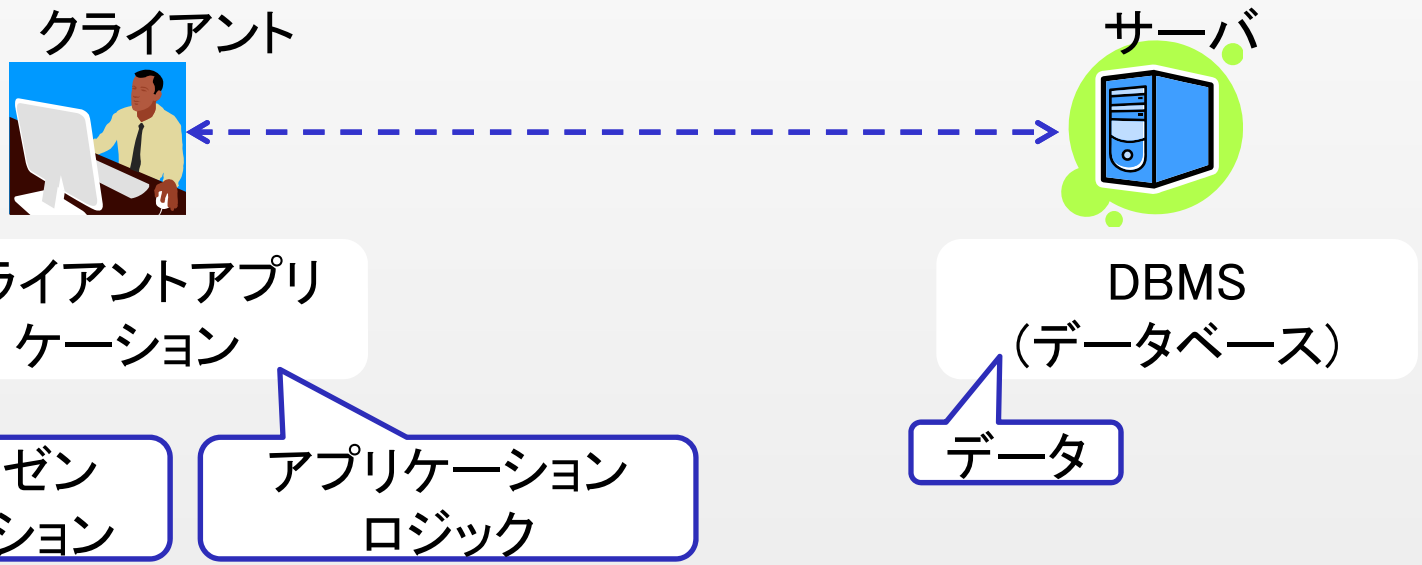
- 分散システムの実現基盤のための代表的な技術を概観することにより，基本的な概念や考え方を確認する
  - 分散オブジェクト（CORBA）
  - Webサービスにおける標準仕様

目次

- 分散システムの実現基盤
  - 分散オブジェクト
  - CORBA
  - Webサービス

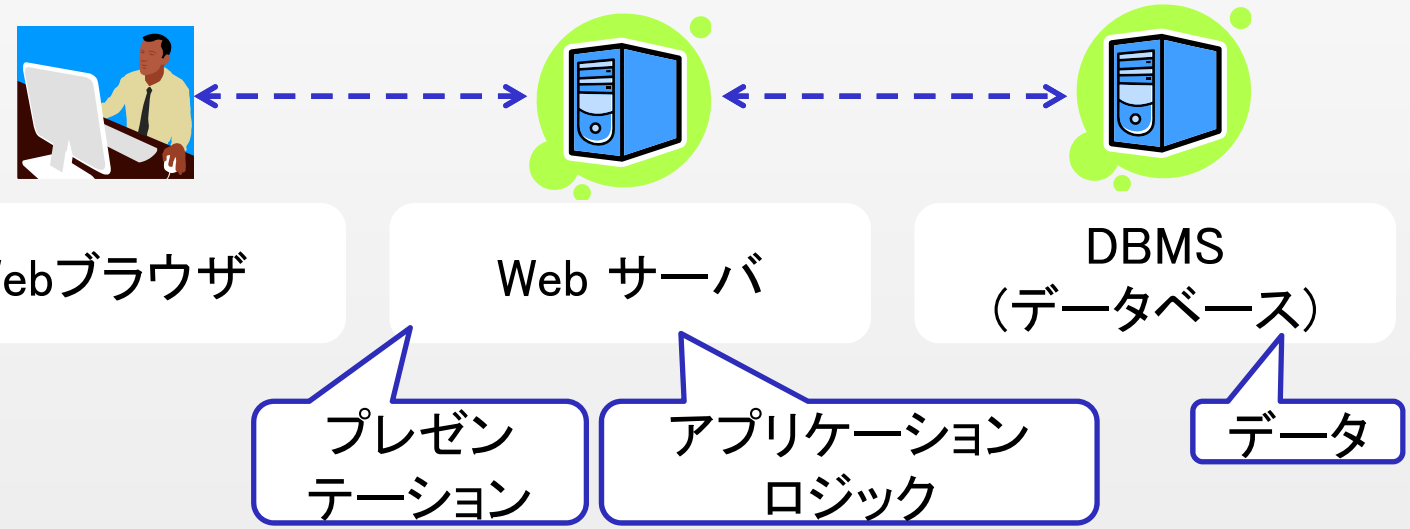
「分散」のモデル

- 基本クライアントサーバモデル



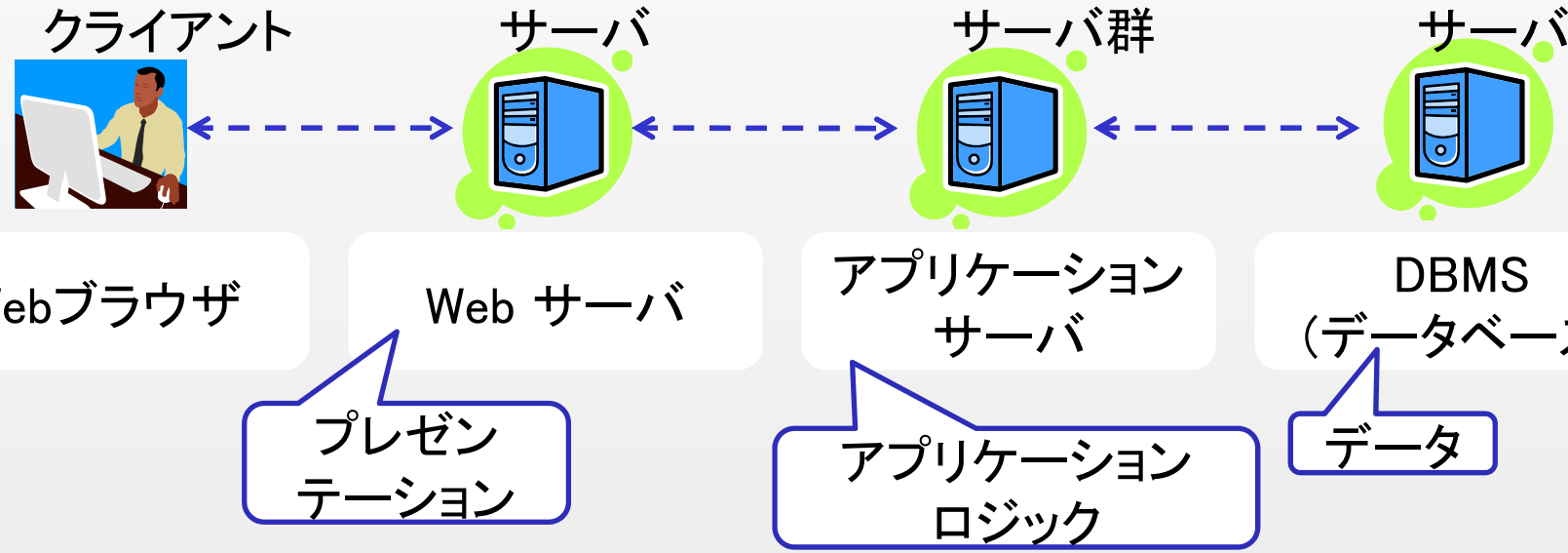
「分散」のモデル

- 初期のWebアプリケーションモデル（例：CGI）



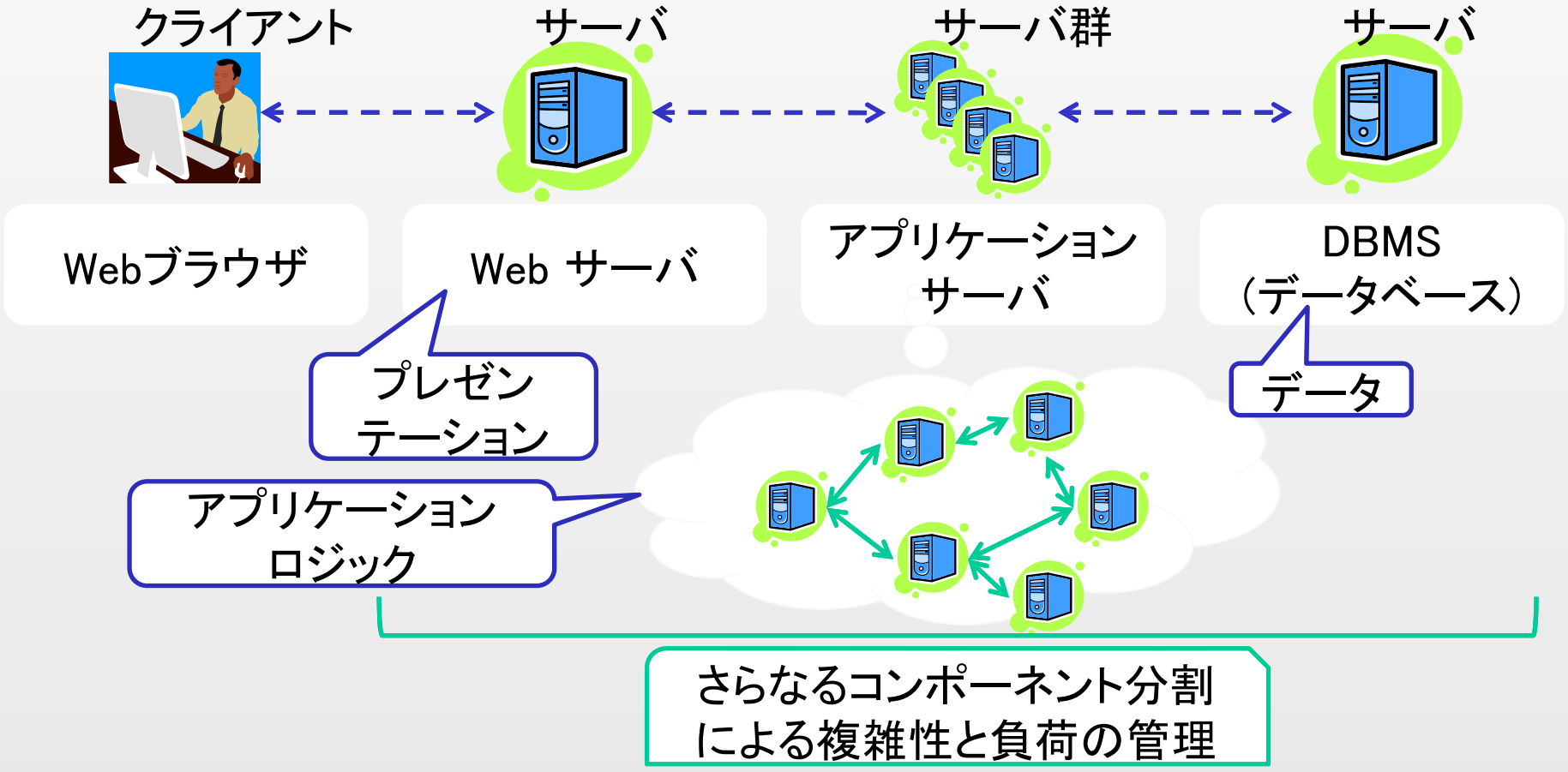
「分散」のモデル

3-Tier Webアプリケーションモデル



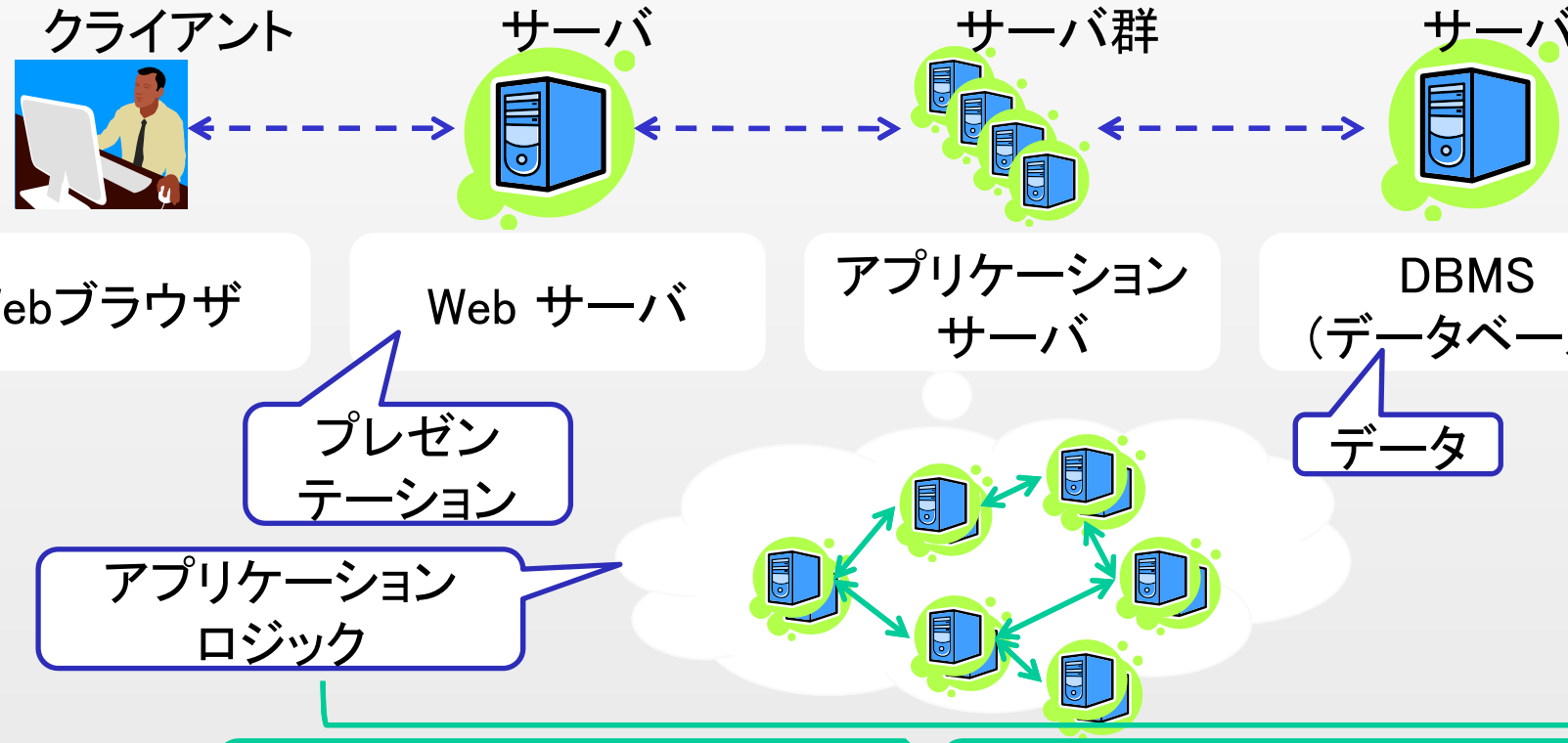
「分散」のモデル

N-Tierモデル



「分散」のモデル

- N-Tierモデル（垂直分散）＋ 水平分散



分散オブジェクト

- オブジェクト指向の利点は分散システムでも重要
  - データとそれに関する機能を併せての部品化
  - カプセル化
- 最も重要な基盤技術として，遠隔でのメソッド呼び出しを実現する「標準となる」仕組みが必要
  - プラットフォーム非依存な形式でのメッセージ交換
  - その他，ネットワーク上での識別方式や，典型的な非機能要求のための処理方式など

分散オブジェクトにおける関連用語

- ORB: Object Request Broker
  - ネットワーク経由でのメッセージング（メソッド呼び出し）を可能とするミドルウェア部品
- Marshalling
  - 異なるプラットフォーム間でのメッセージングのためのデータ形式変換
- IDL: Interface Definition Language
  - オブジェクトのインターフェース（利用可能なメソッドなど）を定義する言語

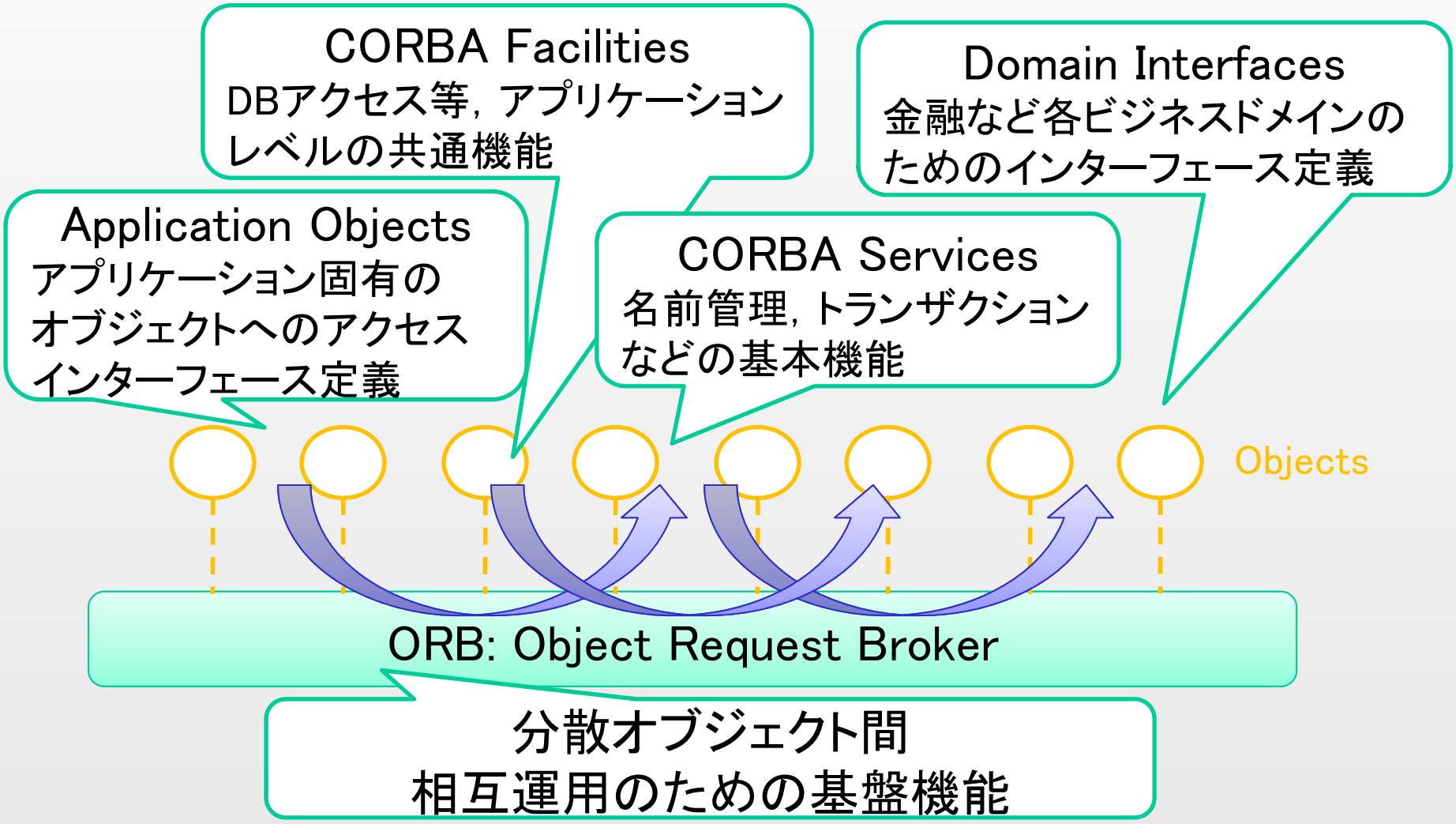
目次

- 分散システムの実現基盤
  - 分散オブジェクト
  - CORBA
  - Webサービス

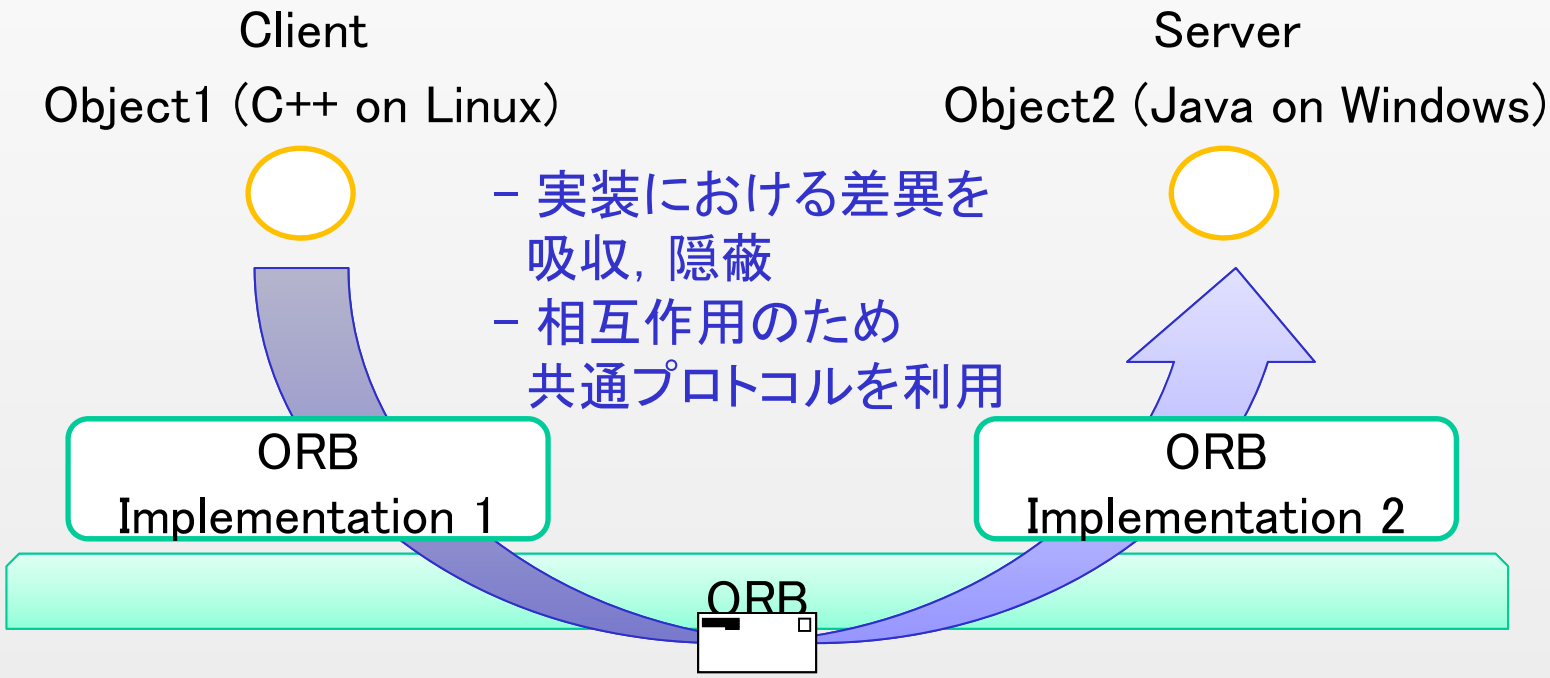
CORBA：概要

- CORBA：
  Common Object Request Broker Architecture
  - 分散オブジェクト間相互運用のための基盤に関する仕様（実装ではない）
  - OMG (Object Management Group)による標準
  - 多くの言語による実装
    - Java, C++, .NET, Python, Ruby, ・・・
  - 1991年ver. 1.0以降発展
    （2011年11月に ver. 3.2）

CORBA：概要

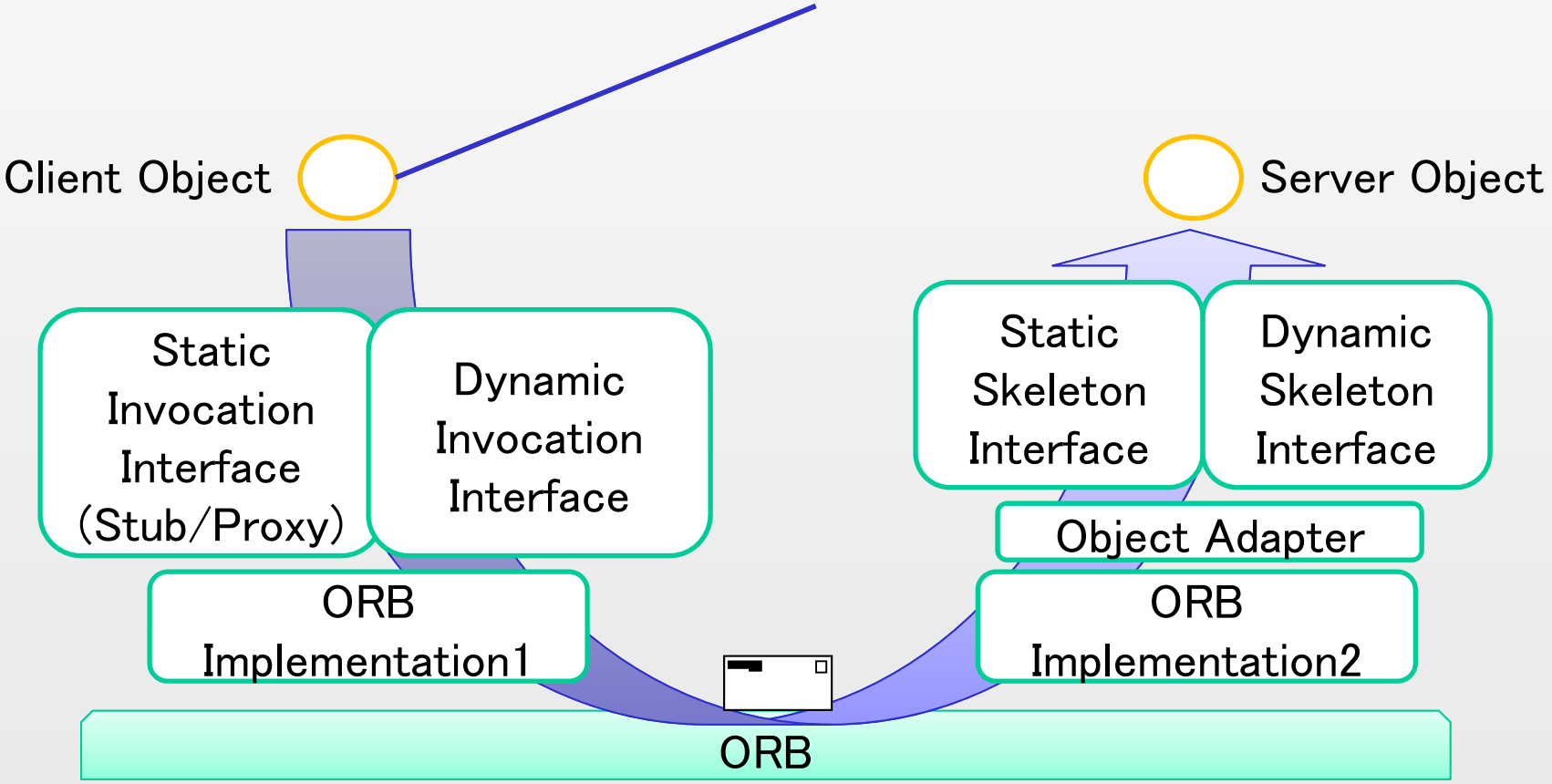


CORBA：ORBの動作概観



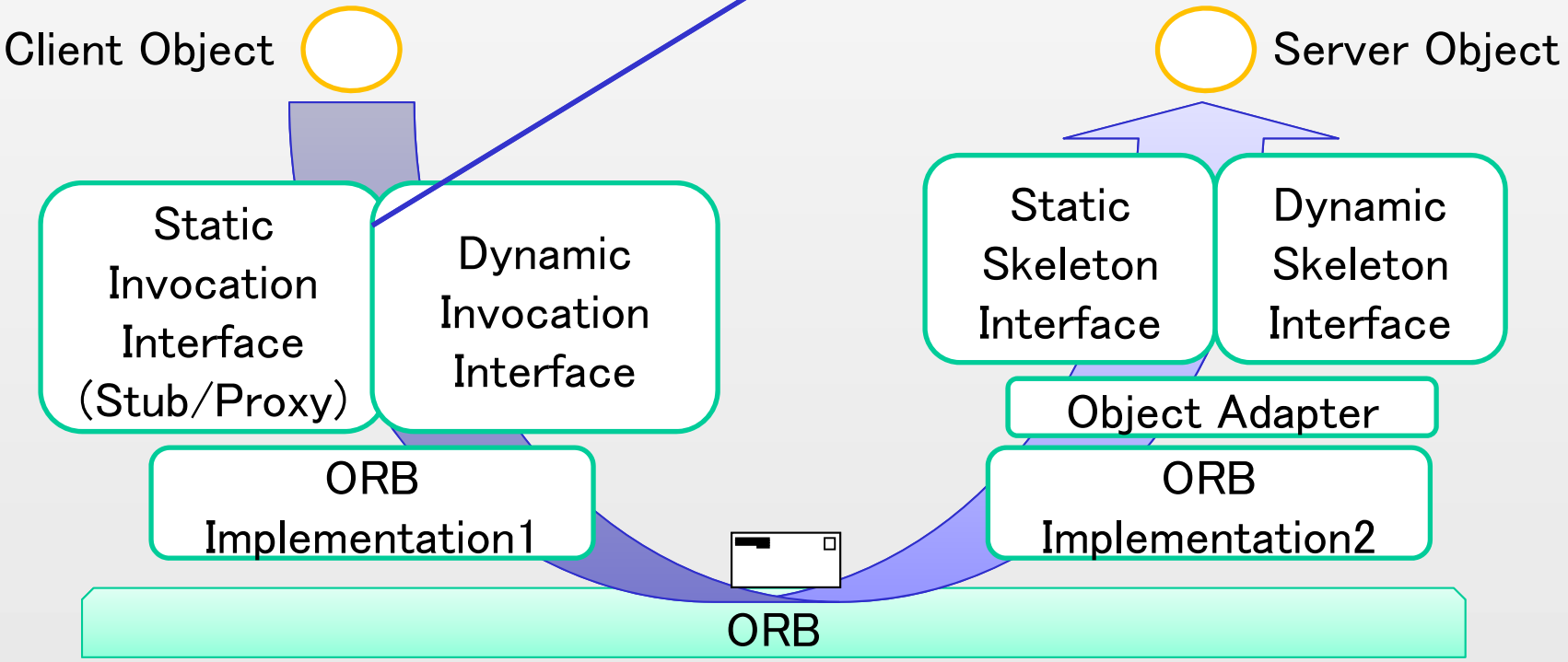
CORBA：ORBの動作概観

1. クライアントプログラムは，naming serviceを用いて
   サーバオブジェクトの参照を取得



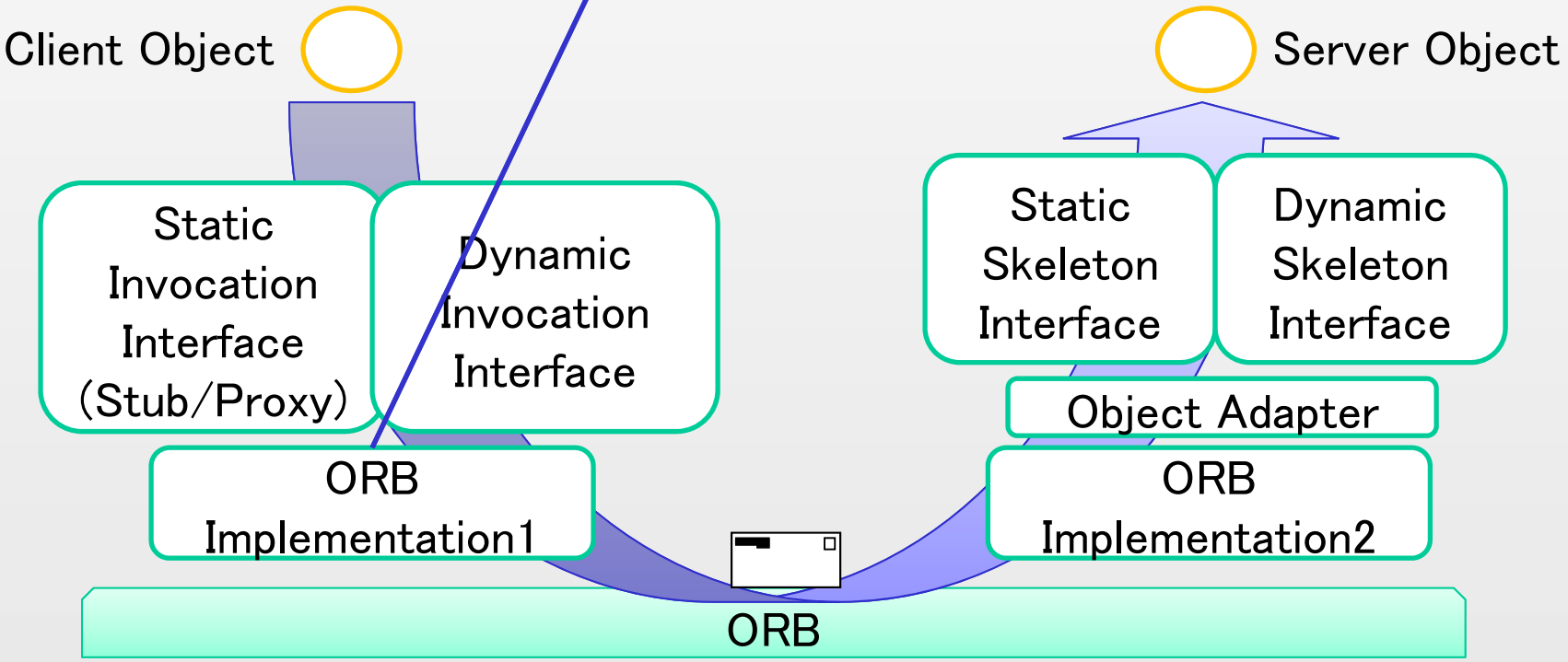
CORBA：ORBの動作概観

2. クライアントプログラムはいずれかを利用して送信
- Static Invocation Interface（事前生成）
- Dynamic Invocation Interface（オンデマンドで利用）



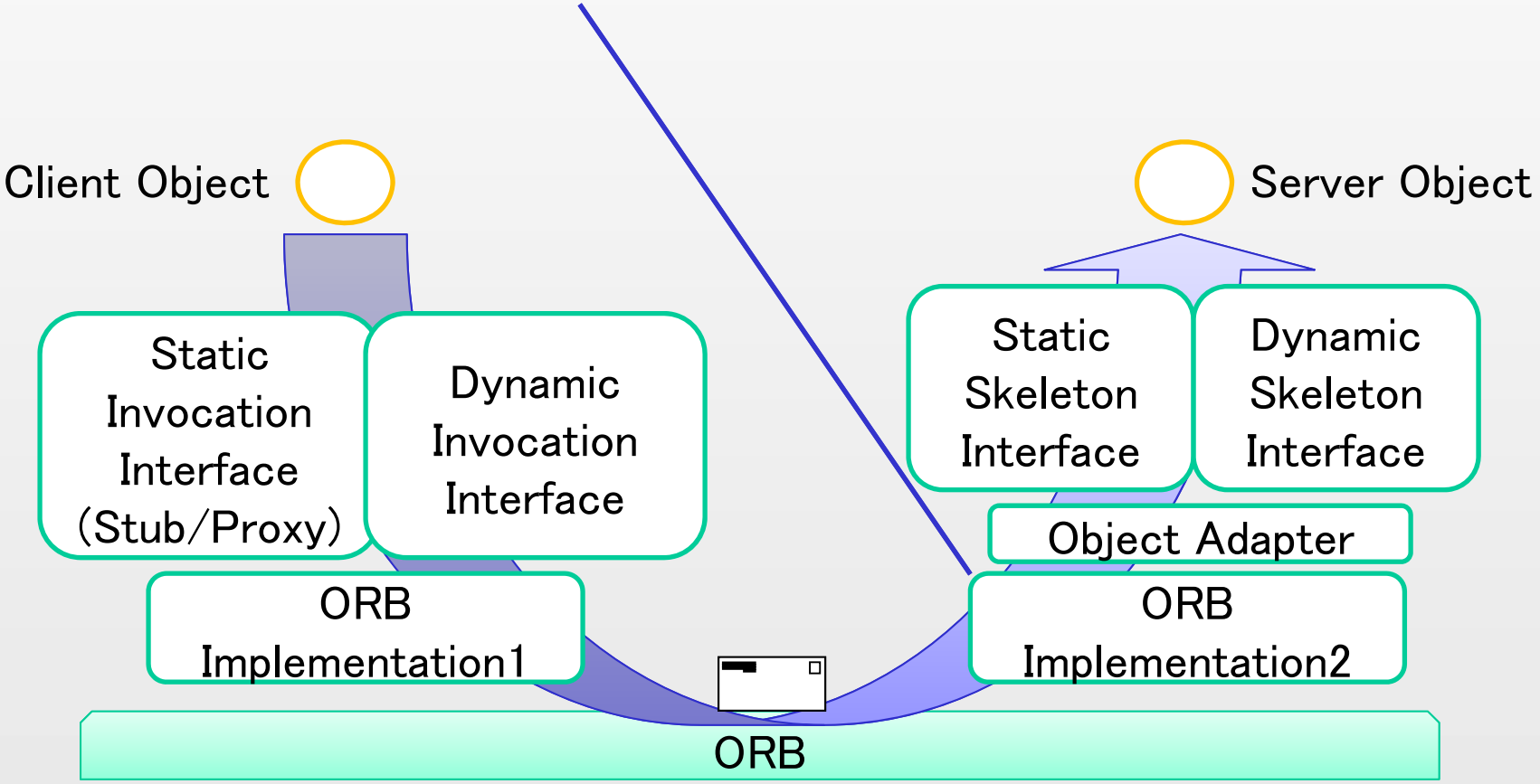
CORBA：ORBの動作概観

3. ORBがプラットフォーム依存の形式から共通のネットワーク用共通形式に変換



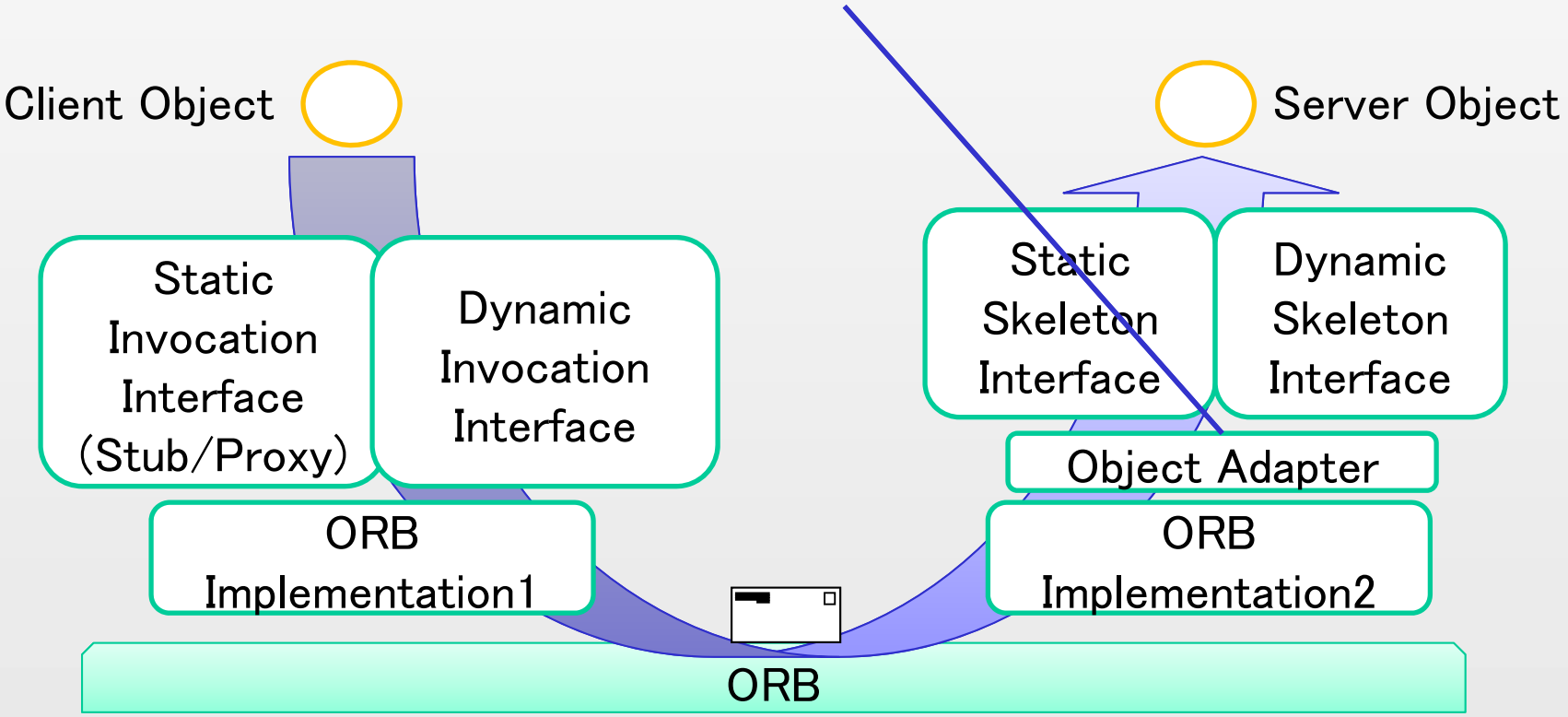
CORBA：ORBの動作概観

4. ORBがネットワーク用共通形式から プラットフォーム依存の形式に変換



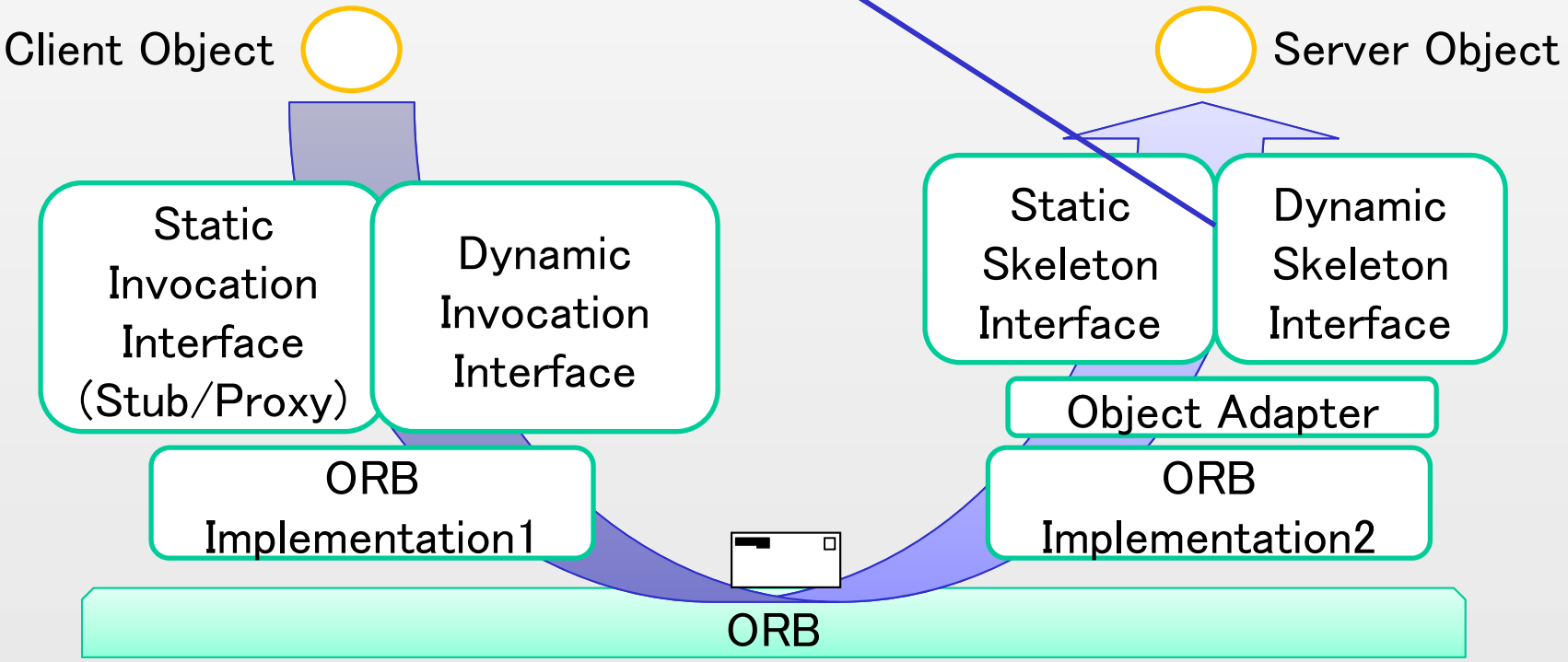
CORBA：ORBの動作概観

5. オブジェクトのライフサイクルを管理する
   object adapterが，必要に応じ永続化・保存されている
   オブジェクトを取得，メモリ上に展開



CORBA：ORBの動作概観

6. サーバプログラムは下記いずれかを利用して受信
- Static Skelton Interface（事前生成）
- Dynamic Skelton Interface（オンデマンドで利用）

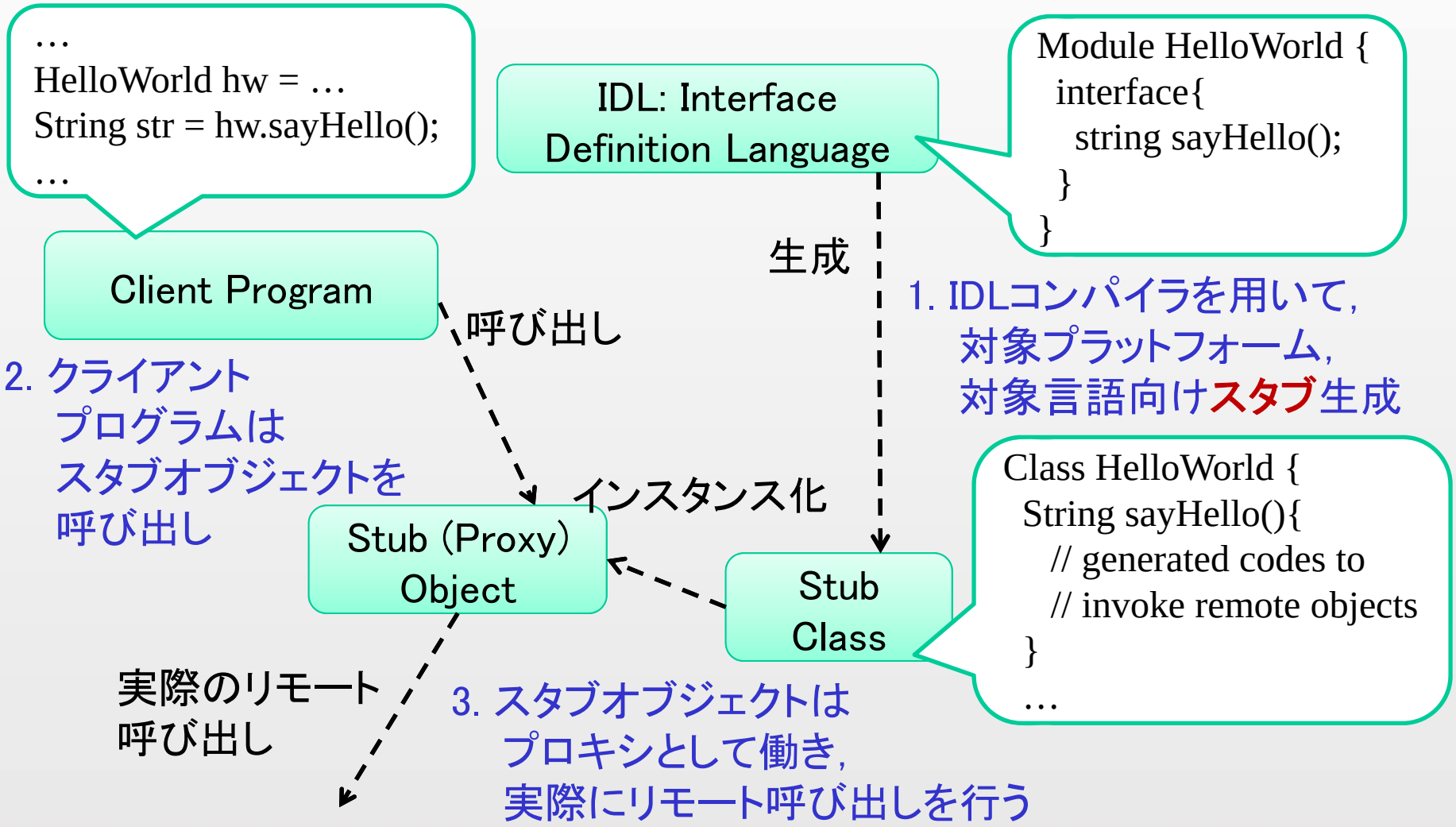


CORBA: Static/Dynamic Interface

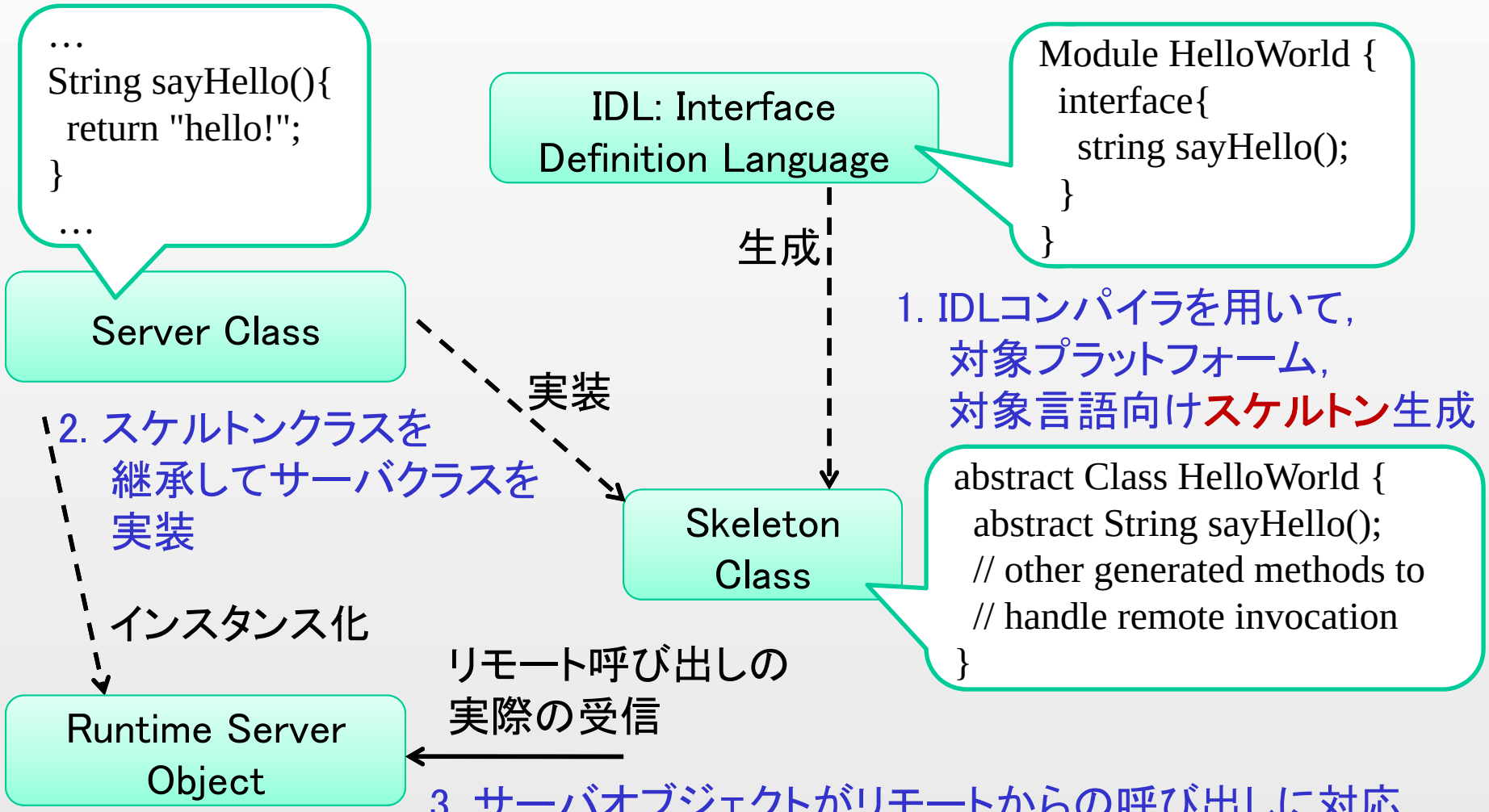
- Static Interface
  - 固定されたインターフェース間の相互作用(のみ)を前もって手軽に実装
  - スタブ・スケルトンによる実装が典型的(後述)
  - 高速に実行可能で，プログラミング言語における型チェックも利用可能
- Dynamic Interface

  invokeRemoteMethod(objRef, methodName, args)
  - 実行時のオブジェクト，メソッドの発見や，動的な割り当てなど，柔軟な実装向け

スタブによるクライアントの実装



スケルトンによるサーバの実装



補足：最近の実装方式

- 最近の（Webサービス向け）フレームワークでは，サーバ側においてスケルトンを使わなくてよいことも多い
  1. 通常のクラス定義を行う
  2. 設定ファイルや，Javaのアノテーション（コンパイラへの指示を出す特別なコメント）などを使い，ネットワークアクセスされる（Webサービスとなる）メソッドを指定
  3. ツールが，サーバコードおよび，IDL記述を生成する

CORBA：その他

- 呼び出しモデルの定義（同期，一方向，非同期）
- 名前管理の方針
  - URL，URI，DNS，XML Namespaceと同様の，階層構造による一意な識別子の構築
- 一般的なメッセージフォーマット
  （GIOP： General Inter-ORB Protocol）
- そのTCP/IPへのマッピング
  （IIOP： Internet Inter-ORB Protocol）

目次

- 分散システムの実現基盤
  - 分散オブジェクト
  - CORBA
  - Webサービス

Webサービスへの展開(2000年前後～)

- Web技術の普及(HTML/XML, HTTP)
  - それと比べると相互運用性に課題(CORBA, DCOMなどの複数の標準ができてしまった)
  - ファイアウォール設定も面倒だった
- XML(とたいていHTTP)を用いたメッセージング
  - XML-RPC(Userland and Microsoft, 1998)
  - SOAP(広く使われる標準となったver. 1.1, 2000)
- XMLベースの様々な標準
- 異論・反動(位置づけの異なる技術)も出て今に至る

Webサービス：定義

- 「Webサービス」（W3C，2002）
  - URIを用いた識別
  - XMLを用いたインターフェース・バインディングの定義，記述，発見
  - 他のソフトウェアアプリケーションと，インターネットプロトコル・XMLメッセージでの相互作用

「Web経由で提供されるサービス（価値や利益の伝達）」という一般名詞でも用いられるが・・・

Webサービス：基本仕様（と呼ばれたもの）

- メッセージング：SOAP
  - 様々な非機能要求への対応や拡張性が不要な場合，無駄に「重い」ため，REST と呼ばれる形式（？ ← 詳細後述）も用いられる
- インターフェース記述言語（IDL）：WSDL
  - 無難に用いられている（スタブ生成など）
- ディレクトリアクセス・同期仕様：UDDI
  - 大企業数社により運用されていたグローバルディレクトリ（世界のWebサービス情報を集約）は閉じてしまい，存在感は薄くなっている

Webサービス：SOAP

- SOAP

  （以前はSimple Object Access Protocol, ver. 1.2よりSOAPという固有名詞）

  - XMLメッセージの封筒構造（HeaderとBody）を定義
  - RPC（Remote Procedure Call）の方法も定義し，その形式でよく用いられるが，それに限らない
  - ネットワークプロトコルに非依存（とはいえHTTP上での実現は規定されており，よく用いられる）

Webサービス：SOAP

SOAP外部で規定される様々なヘッダ情報を埋め込む枠と，その説明方法（属性）を規定

Envelop

Header

Routing info　*mustUnderstand*="false"
…

Security info　*mustUnderstand*="true"
…

Body

getStockInfo
companyName
…

最終的な受け取り手に向けたメッセージ（例：RPCのためのメソッド名と引数）

REST/RESTfulサービス？？？

- SOAPのような，非機能要求の考慮と拡張性（とそれゆえの重さ）はいらないことも多い
- 「REST/RESTfulサービス」
  - HTTPコマンドを直接使う（GET，POSTなど）
  - Webアプリケーションにも似ているが，（入）出力にHTMLではなくXMLを用いる
  - それだけの意味のことも時折あるが・・・
    - SOAP-RPC/HTTP getUser(ishikawa)　ではなく
    - HTTP GET /getUser?name=ishikawa　ならよい？
    - HTTP GET /users/ishikawa　だと？

本来のREST

- REST: Representational State Transfer

  [Ph.D. Thesis by Roy T. Fielding, 2000]
  （HTTP仕様やApacheへの貢献者の1人）

  - アーキテクチャの原則を定義
    - Client-server, Stateless, Cacheable, Uniform interface, Layered system, Code-on-demand (optional)
  - Web系企業では推奨（Google, Facebookなど）
    - ただし，実際には一部の原則を選択して採用することが多い

本来のRESTにおける原則(１)

- Client-server：ユーザインターフェースとデータストレージの関心事を分離
  - ユーザインタフェースのポータビリティ，サーバのスケーラビリティ
- Stateless：リクエスト内にそれを理解するための情報を全て含み，サーバ側に状態を持たせない
  - スケーラビリティ，監視の容易性，故障からの復帰の容易性
- Cacheable：キャッシュ可能なものを明確に区別
  - 効率，スケーラビリティ

本来のRESTにおける原則（２）

- Uniform interface：単一インターフェースに統一
  - 単純さ，インターフェースと機能の分離
- Layered system：階層構造による不要な詳細の隠蔽
  - 複雑さの軽減，依存性の低減
- Code-on-demand：必要な機能の，クライアント側へのオンデマンドでの読み込み（オプション）
  - 実行時の拡張性

REST/RESTfulサービス？？？

- 適当にHTTPコマンドを使うだけでは，そもそもコマンドの意図にも合わず，キャッシュ可能かどうかも判別できない
  - HTTP GET /setUser?name=ishikawa
- URLは動詞ではなく名詞（Resource）として設計
  - HTTP POST /users
    <?xml><user><name>ishikawa</name></user>
  - HTTP GET /users/ishikawa

Webサービス: WSDL

- WSDL: Web Service Description Language
  - インターフェース記述言語

types XML Schemaによる
用いられるデータ型の定義

interface インターフェース定義
（提供される操作の集合）

binding アクセス方法定義
（例: SOAP/HTTP）

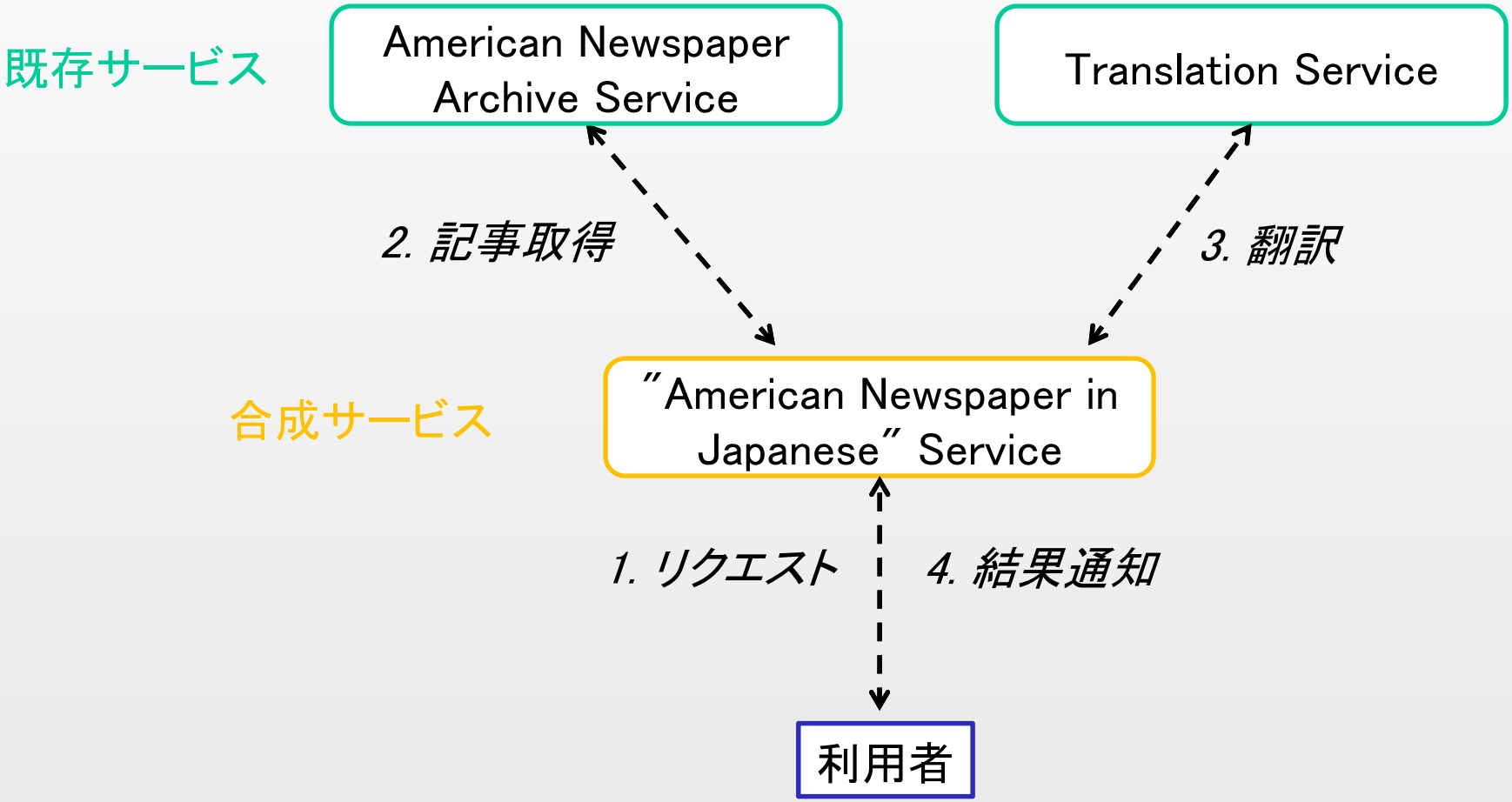
service エンドポイント定義
（IPアドレスとポート）

再利用可能な抽象的定義

具体的な実体定義

注: ver. 2.0の語彙

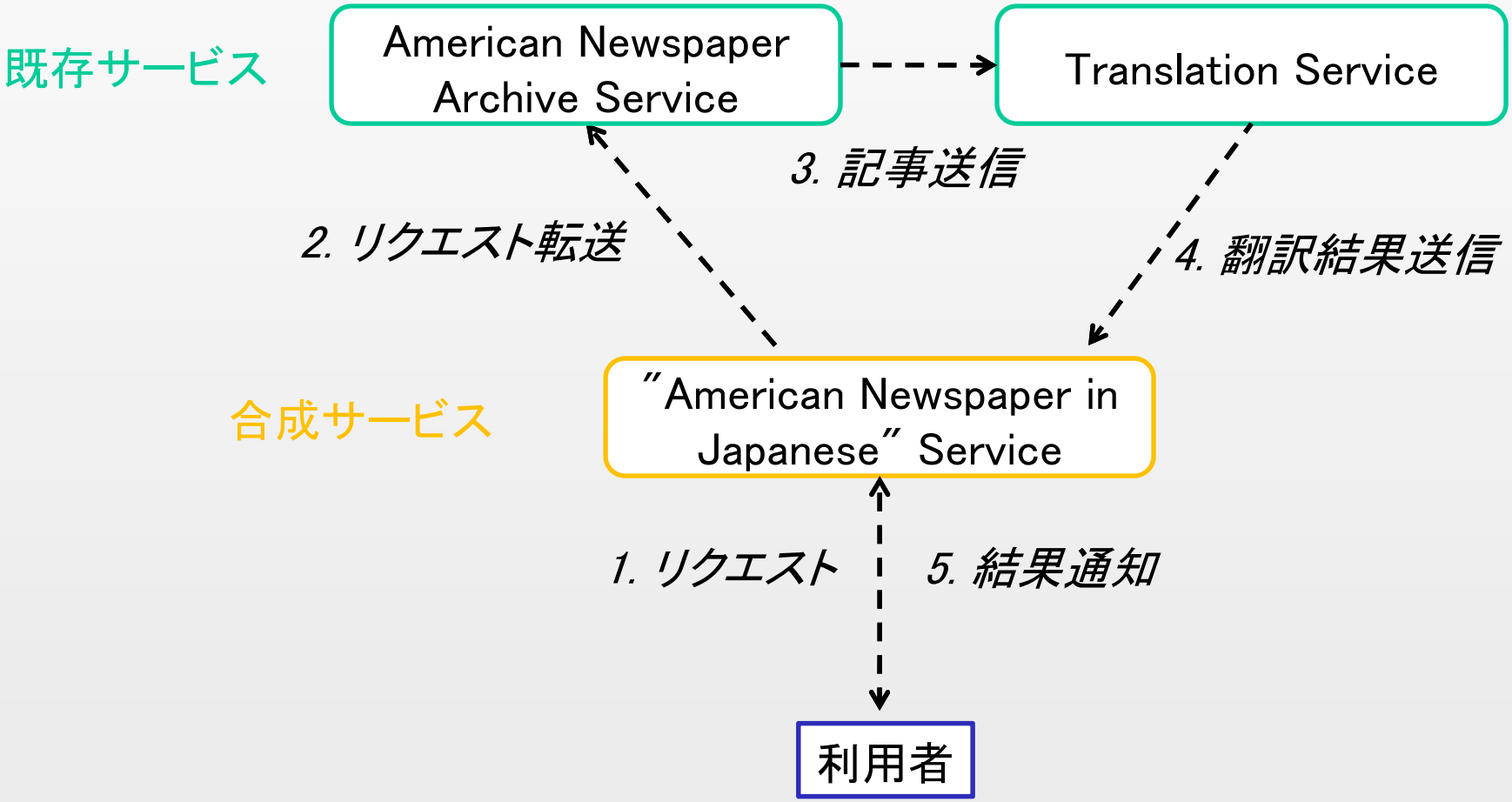
Webサービス：連携の実現方法（中央型）



Webサービス：連携の実現方法（中央型）

- 中央集約型の制御（オーケストレーション）
  - 合成者のみがロジックを把握していればよい
  - 利用される既存サービスは合成を意識する必要なく，単に合成者を自身の利用者の1つと考えればよい
  - 通信がすべて合成者を経由するオーバーヘッドがある

Webサービス：連携の実現方法（非中央型）



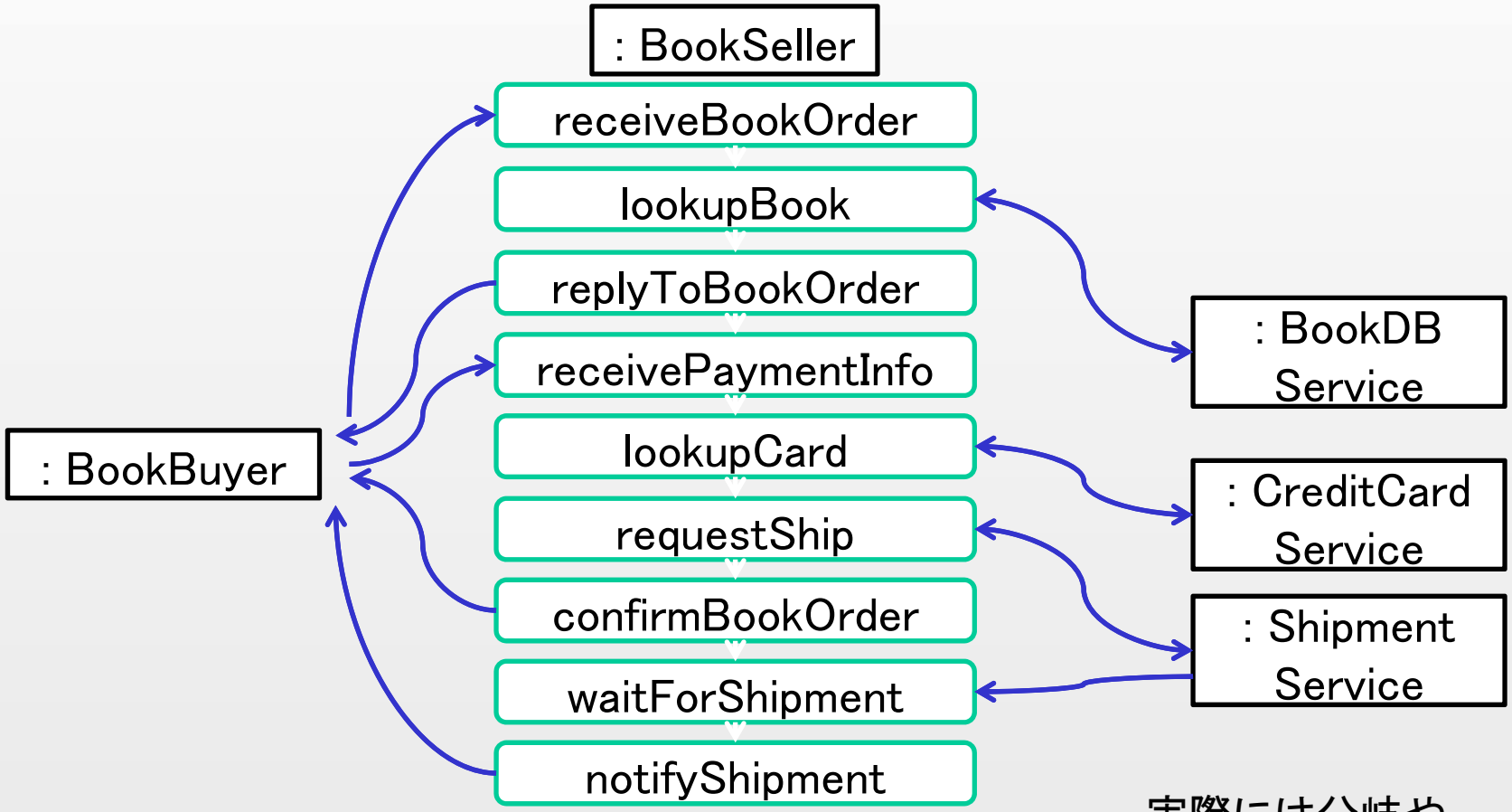
Webサービス：連携の実現方法（非中央型）

- 非中央集約型の制御（コレオグラフィ）
  - 各参加者が，その合成における自身の役割を把握している（もしくはそういう指示を受ける）
  - 通信を最小化できる

Webサービス：WS-BPELとWS-CDL

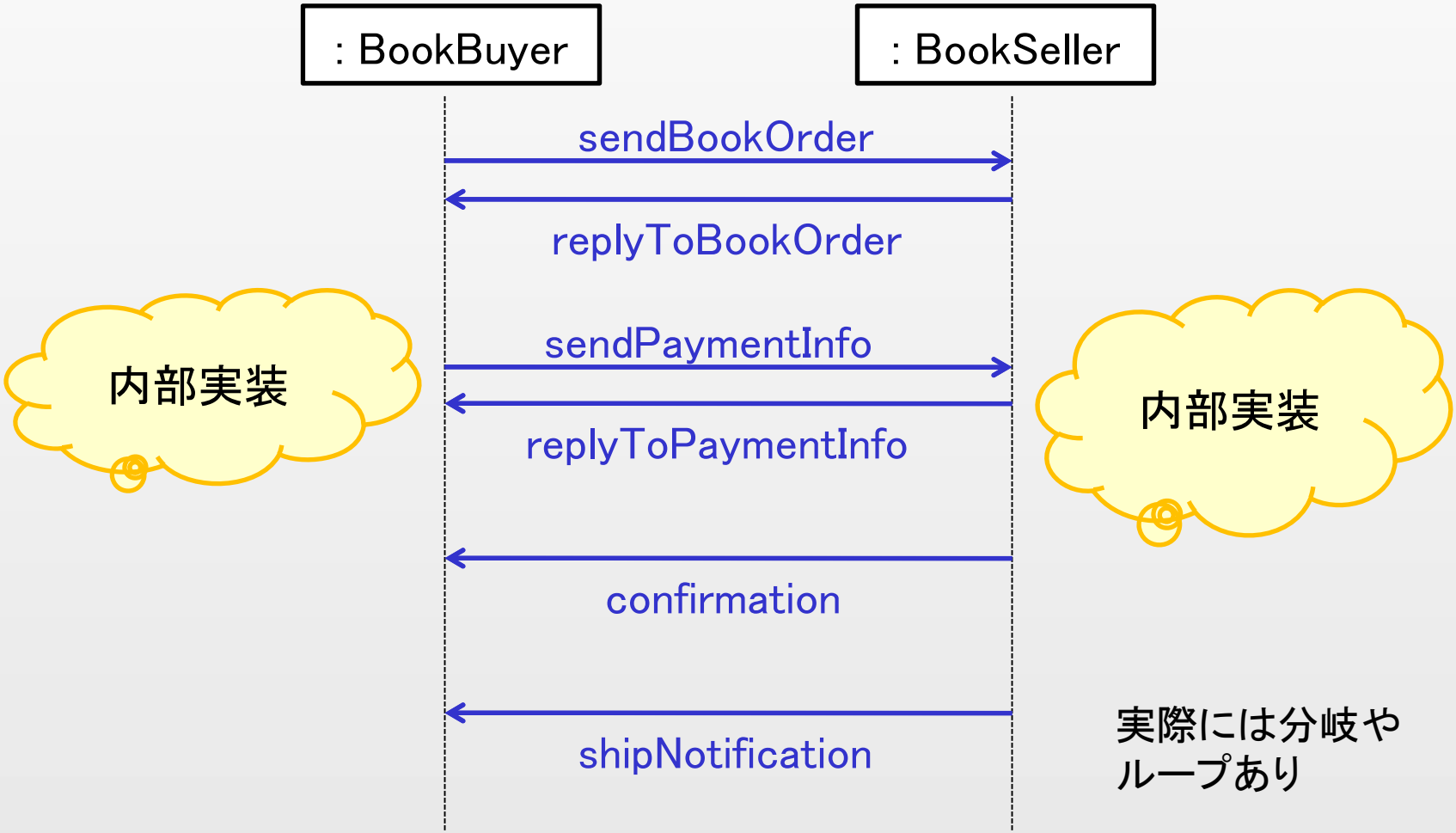
- これらの連携方法のいずれにおいても，
  - 各ノードの動作定義が必要
    （Javaなどで書いてもよいが，サービスを組み合わせるというもっと抽象的なロジックでも書ける）
  - ➡ WS-BPEL: Web Service Business Process Execution Language
  - 各ノード間で起きうる通信に関する約束事（プロトコル）定義が必要
  - ➡ WS-CDL: Web Service Choreography Language

Webサービス：WS-BPEL



実際には分岐やループあり

Webサービス：WS-CDL



Webサービス：その他様々な仕様

- メッセージ配信のルーティングや信頼性
  - WS-Addressing
  - WS-Reliability/WS-ReliableMessaging
- トランザクション関連
  - WS-Coordination, WS-AtomicTransaction, WS-BusinessActivity
- セキュリティ関連
  - WS-Security, WS-Federation, WS-Trust, …

などなど

（すべてがよく用いられているわけではない）

補足：相互運用性

- 決めて皆が従った範囲は，相互運用性がある
- Webサービス標準仕様は拡張性重視で，決めないことも多い
  - 例：SOAPはHTTPに限らずどのネットワークプロトコルにのせてもよい
  - 例：SOAPのヘッダ内の構造は別途規定
  - 例：WS-Securityには暗号化方式の指定欄
- 実際の相互運用には＋αの「決め」が必要
  - WS-I (Web Services Interoperability)によるプロファイル規定

今回のまとめ

- 分散システムを「構築し動かす」ための基盤技術は確立され，広く使えるようになっている
  - 分散オブジェクト，Webサービス：異種混合の環境や非機能要求を考慮できる枠組みを紹介
  - 閉じたシステム部品の内部実装であれば，プログラミング言語を固定し，ネットワークプロトコルの特性を活用してより高速に
- 次回：基本的な問題を通して，同期や一貫性，耐故障性を議論する