「サービス・クラウド技術をSEに活かす」

国立情報学研究所 石川 冬樹
情報通信研究機構 村上 陽平，田仲 正弘
（サービスコンピューティング時限専門研究委員会）

本チュートリアルの目的

Webサービス・クラウドを使うための基礎

特に「アプローチは何となくわかるが，実際の利用・開発（プログラミング）を行ったことがない」人の勉強・活用の第一歩に

- 一般的な基礎知識を解説する
- 「分散処理（大量・耐故障）」の例題を通して活用のための基礎技術を解説する
- 例題を中心に，「特長」と「手軽さ」，そして「難しさ」を議論する

SC研究会の紹介

- サービスコンピューティング研究会（2009～）

  http://langrid2.nict.go.jp/sc/

  （電子情報通信学会 情報・システムソサエティ）

  サービス・クラウドの連携やビジネスプロセスにおける信頼性確保，ポリシー，経済的問題などに横断的・総合的に取り組む

  - SEは（AIと並んで）一つの大きな技術観点での軸
    - SOAやBPMなど問題領域の定義と取り組み
    - 要求分析，形式検証，MDAなどの技術応用
  - SIG-SEウィンターワークショップ内のセッションなど随時イベント開催中

目次

- クラウドとは
- Webサービスの技術的基礎
- 例題： AWS上のLOC
- 議論・おわりに
- 参考文献

クラウド： 定義

- 「計算資源へのアクセス」のためのモデル
  - On-demand self-service： 提供者との人手を介したやりとりなく，自動で取得可能である
  - Broad network access： 様々なプラットフォームから標準的な方法で利用可能である
  - Resource pooling： 詳細が隠蔽された形でプールされ，複数の利用者に提供される
  - Rapid elasticity： 迅速に，伸縮可能に提供されており，無限に見えるものから必要な分だけ取得する
  - Measured Service： 抽象的な指標での測定に基づき，利用が可視化された形で制御，最適化される

[The NIST Definition of Cloud Computing (Draft), Last update 2011]

クラウド：サービスモデル

- SaaS (Software-as-a-Service)
  - 構成済みのパッケージソフトウェア
  - Gmail，Salesforce CRM，Microsoft Online Services等
- PaaS (Platform-as-a-Service)
  - 特定用途・特定プログラミング言語向けの，自動管理ミドルウェアを含むプラットフォーム
  - Force.com，Google App Engine，Windows Azure等
- IaaS (Infrastructure-as-a-Service)
  - 仮想マシンやストレージを自由に（自身で中身を設定して）使うための基盤
  - Amazon EC2，Nifty Cloud等

クラウド：活用の方向性？

- よく登場するIaaS/PaaSにおける提供・活用

大量処理・データを複数サーバに
分散配置・複製し，高速に・高信頼に！

  - 大量のビジネスデータやセンサデータの解析など
  - 安い（普通の）ハードを大量に使い，どんどん落ちる・失敗する前提で実行するという考え方
    - 内部の運用管理もその考え方で
  - 技術的な「クセ」
    - 関数型の処理記述やキューの利用など，複製（並行化・故障復帰）しやすい処理記述が求められる
    - 複製に関して担保される整合性について，理解し対応することが求められることも

クラウド：他の特長（スケールメリット）

- リソースプールを一括で購入・（自動）運用
- ➡ 各自が個別に行うより低コスト
  - 組織内で（プライベートクラウド）
  - 外部の専用業者が（パブリッククラウド）
- 例：Amazon EC2（計算資源）でのデフォルト
  - 仮想マシンタイプSmall でLinux ：$0.085/hour
    （1.0-1.2GHz Opteron/Xeon程度，メモリ1.7GB）
  - ➡ *× 24h × 30days = $61.2/month*
    （実際は必要な分だけ起動すればよい）
  - EC2からのダウンロード10TBまで：$0.12/GB
  - ストレージ：$0.1/GB，I/O 1Mリクエスト：$0.1/GB

[http://aws.amazon.com/jp/ec2/]

クラウド：他の特長（スケールメリット）

「多く買うほど安い」

	1,000 servers	50,000 servers
Network (per 1M/sec)	$95	$13
Storage (per 1G)	$2.2	$0.4
Management (per 1 supervisor)	140 servers	1000 servers

[Above the clouds: A berkeley view of cloud computing, 2009]

- Amazonのクラウドへのトラフィックは，「本」業のものより多くなっている
  - 「本」業の基盤のコスト減少にもますます効く

クラウド：他の特長（Programmable）

- Programmable：
  - プログラムから「Webサービス」として（ネットワーク越しに）呼び出し可能

    （人間がWebインターフェースなどから利用するだけでなく）
  - 自身の要求に合った制御を実現することができる
    - アプリケーションとして
    - フレームワーク・ライブラリとして

目次

- クラウドとは
- Webサービスの技術的基礎
- 例題： AWS上のLOC
- 議論・おわりに
- 参考文献

Webサービス： 定義と技術

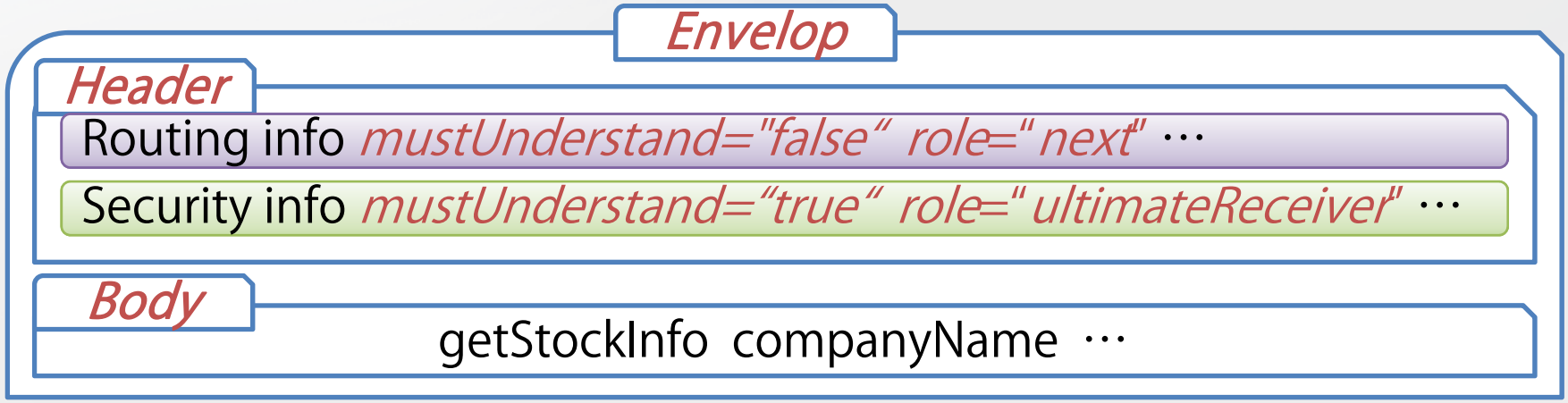
「Webサービス」
- URIを用いた識別
- XMLを用いたインターフェース・バインディングの定義，記述，発見
- 他のソフトウェアアプリケーションと，インターネットプロトコル・XMLメッセージでの相互作用

[W3C, 2002]

- W3CやOASISにおける多くの標準仕様
  - メッセージング（SOAP）
  - インターフェース記述（WSDL）
  - 実行プロセス定義（WS-BPEL）
  - その他セキュリティ（WS-Security）などなど

Webサービス： SOAP

- SOAP： XMLによるメッセージングプロトコル
  （元はSimple Object Access Protocol，今は固有名詞）
  - 様々な（他仕様で定義された）メタ情報を入れるためのヘッダを含む封筒形式を定義



  - RPCでのBody形式・HTTP上でのエンコーディングも標準として定めているが，それらに限らない
    - P2PやMANETへの利用，SMTPの利用など

Webサービス： RESTサービス？？？

- SOAPのような拡張性（と引き替えの面倒さ）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）
  - CGIプログラミングにも似ているが，出力にHTMLではなくXMLを用いる（ときには入力にも）
  - 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？

Webサービス： REST Architecture

- Representational State Transfer （REST）
  - 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
  - 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ずスケーラビリティ・耐故障性向上を容易に
  - Cacheable： キャッシュ可能なものを明確に区別
  - Uniform interface： 単一のインターフェースに統一
  - Layered system： 階層構造による不要な詳細の隠蔽
  - 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）

[Roy T. Fielding, Ph.D. Thesis, 2000]

Webサービス： WSDL

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

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

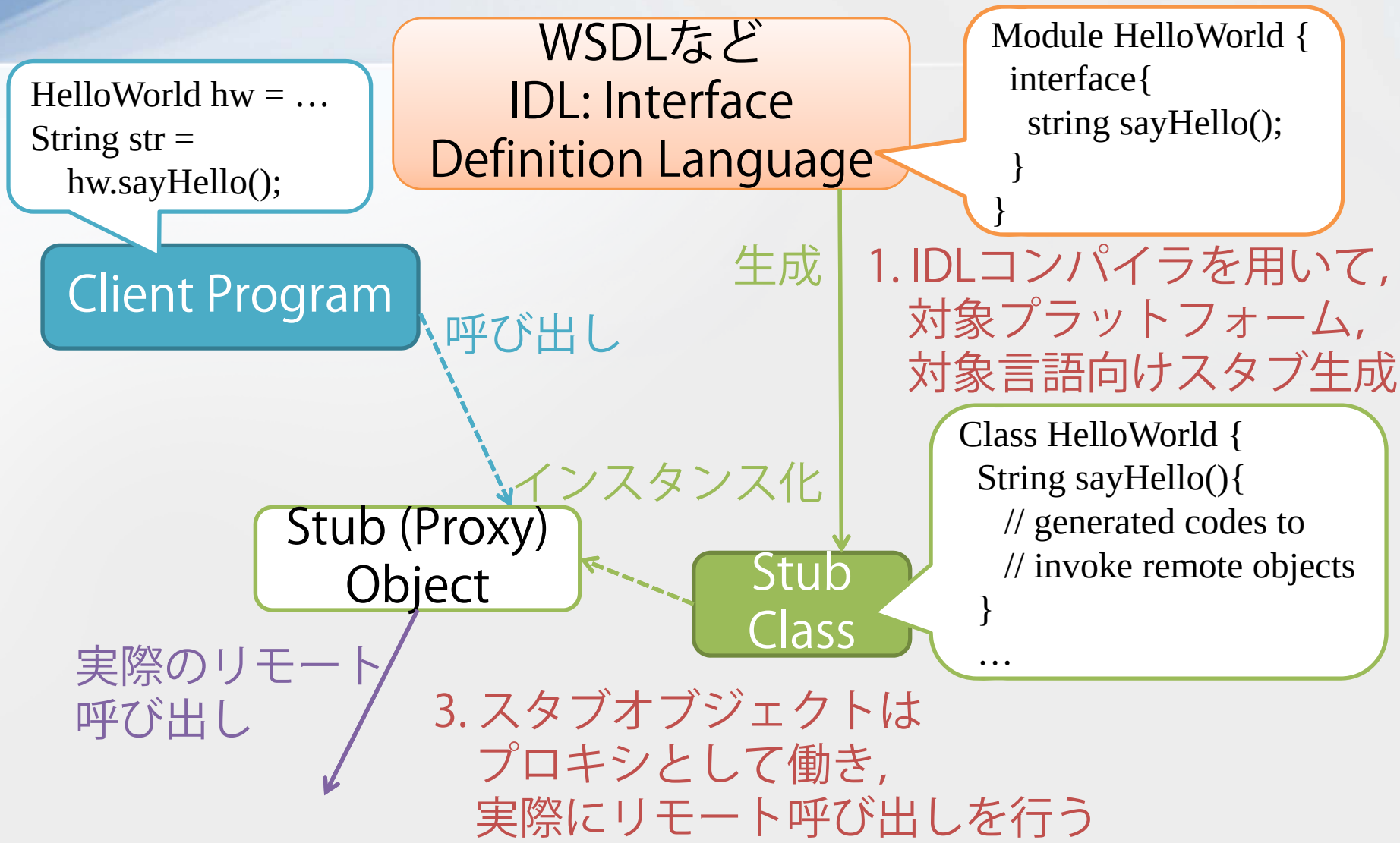
具体的な実体定義

Webサービス： 実装方法

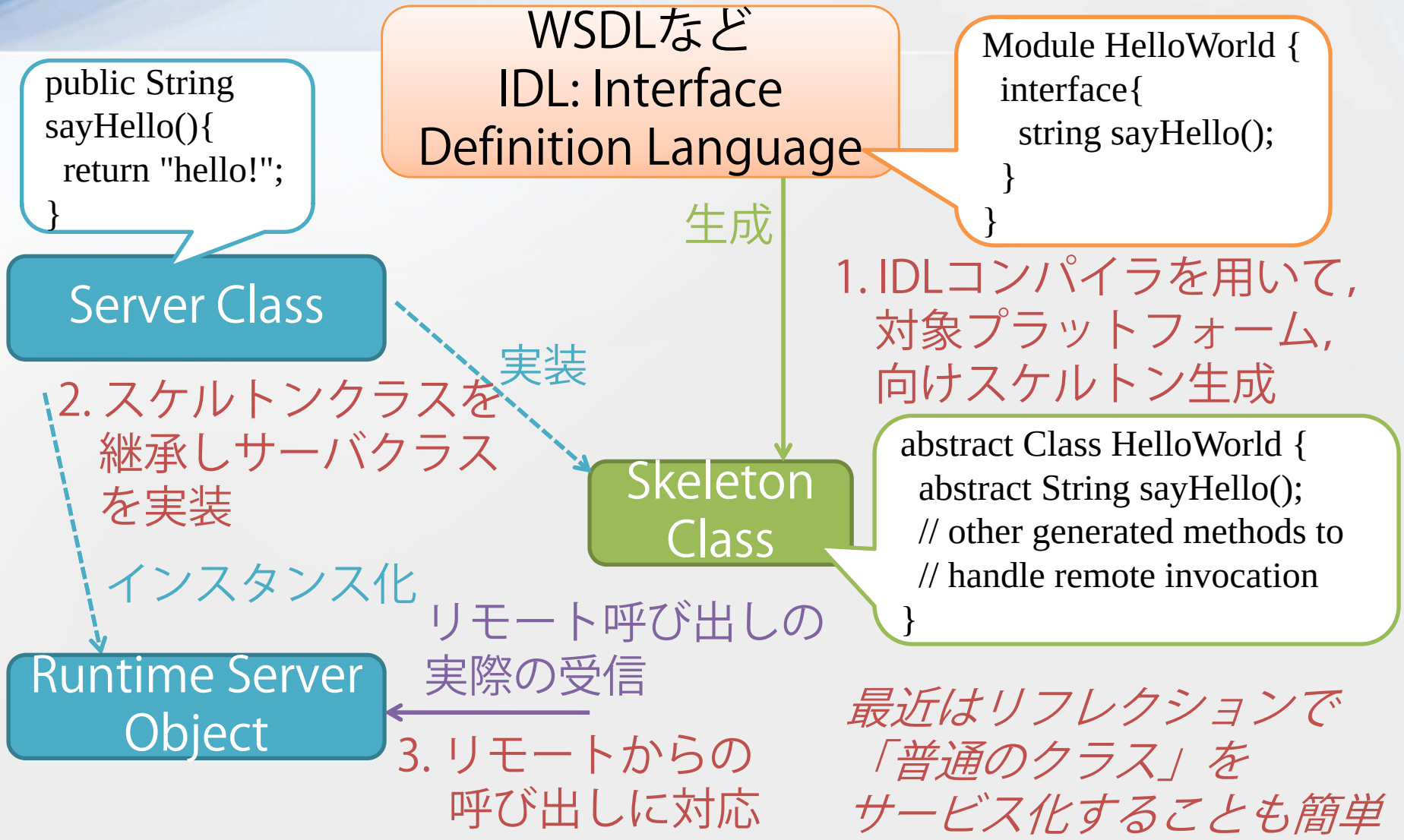
クライアント・サーバ側ともに，

- （HTTP/XMLライブラリを直接使う）
- Webサービスフレームワーク・ライブラリを使う
  - Java SDKや.NETなどには標準で含まれる
  - Apache Axis2
  - その他多数
- 特定Webサービスが提供する，そのサービス専用のフレームワーク・ライブラリを使う

Webサービス：相互運用実装の基本（１）



Webサービス：相互運用実装の基本（2）



目次

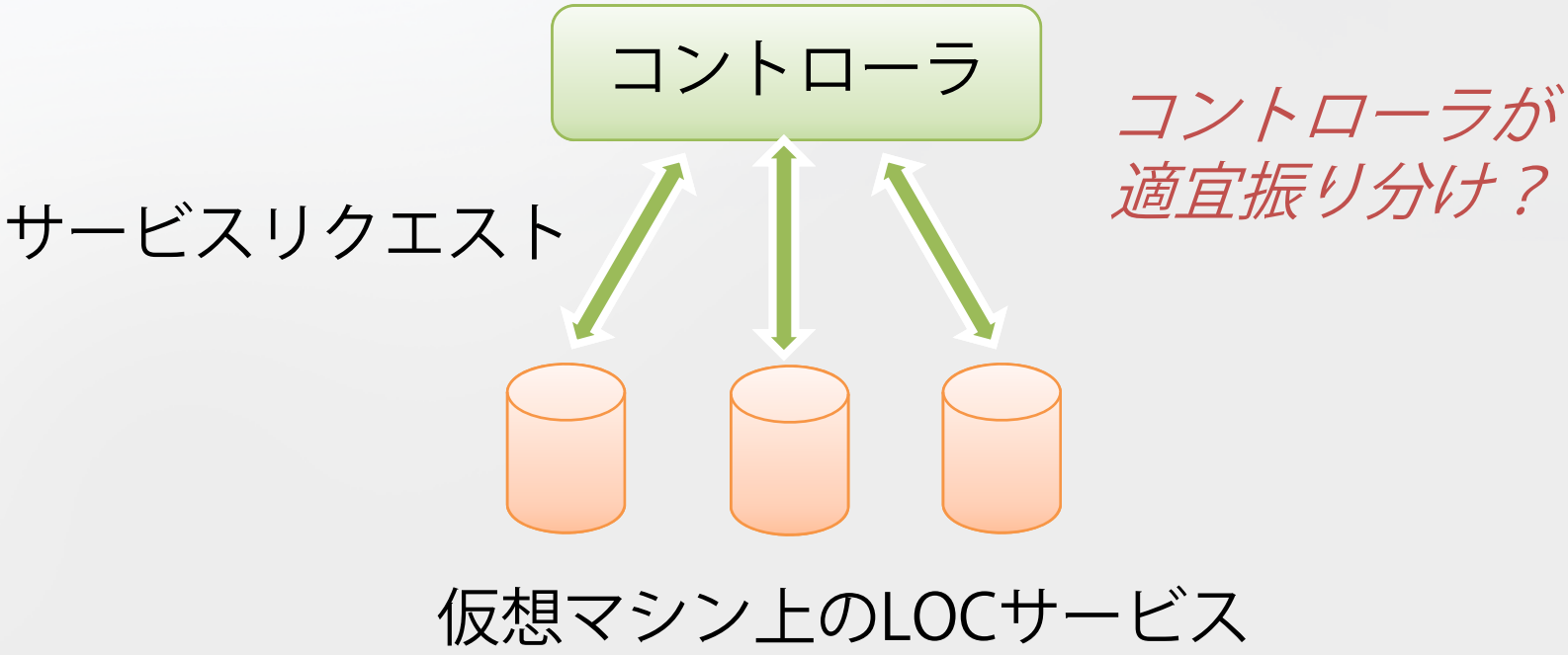
- クラウドとは
- Webサービスの技術的基礎
- 例題： AWS上のLOC
- 議論・おわりに
- 参考文献

例題： 概要

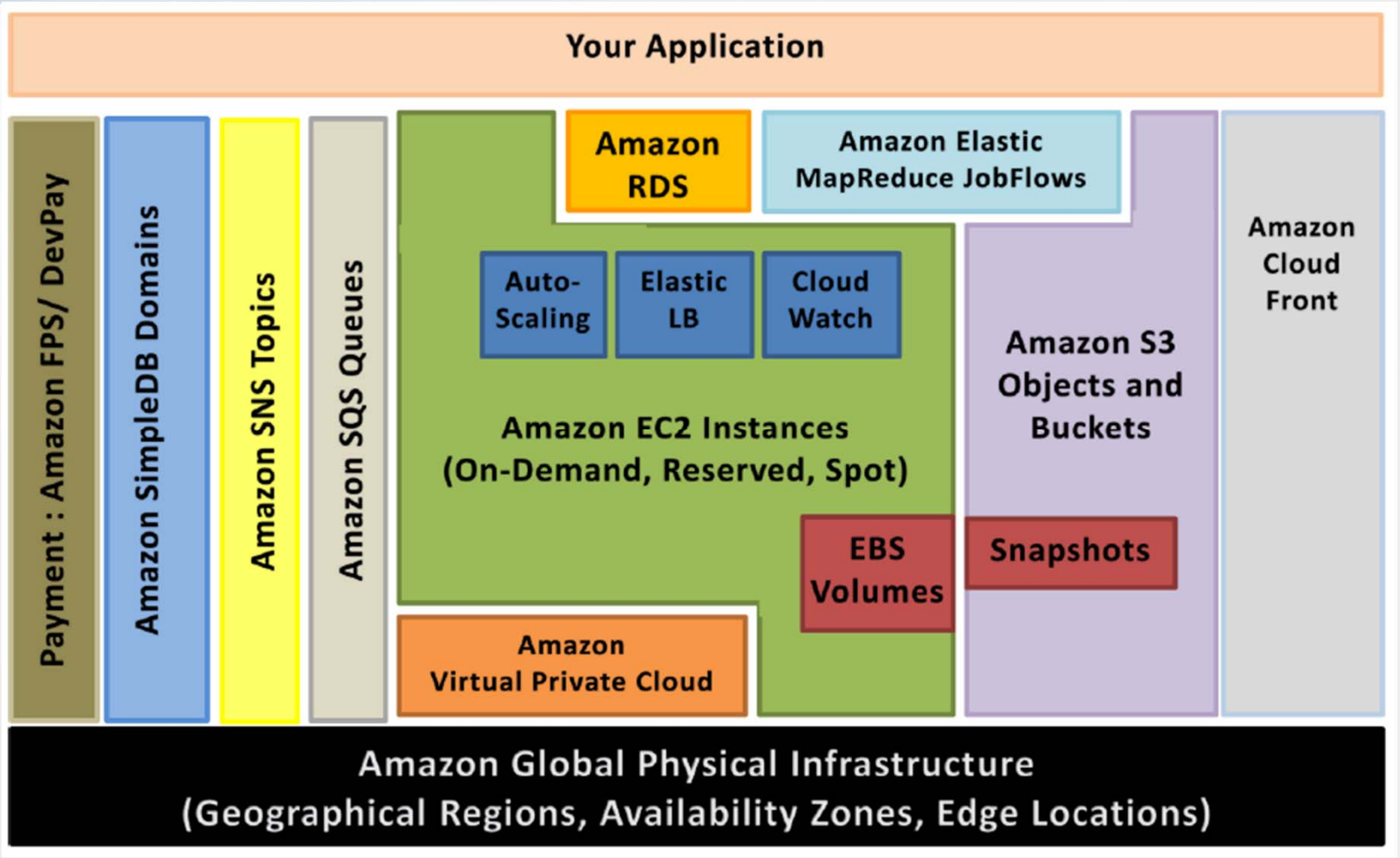
- 簡単なものを作ってみよう！
  - やっぱり大量のソースコードへのバッチ処理？
    （ビルド，インテグレーション，テスト，静的検証，
    メトリクス解析，リファクタリング，・・・）
  - ⇨今回は単純にLOC（Lines of Code）を計測してみる
    - コードファイル単位で並行処理可能
    - Java1.6のソース（7000強のファイル）で動作確認
  - AmazonのIaaSを中心に
    - Map Reduce（Hadoop）は今回はパス
      （Word Countが典型例題として使われている）
    - ⇨仮想マシン立ち上げなど，中身の動きがもう少し
      見える作り方をしてみる

例題：アーキテクチャ？？

- こんな感じ？



Amazon Web Services： 全体像



[J. Varia, Architecting for The Cloud: Best Practices, 2010]

Amazon Web Services： 構成（１）

- 計算資源（仮想マシン）

  EC2 （Elastic Computing Cloud）

  - その場その場での利用（On-Demand）だけでなく，安くなる長期利用契約（Reserved），さらに安い余剰資源への入札（Spot）
  - RegionとZoneを指定： 遅延に影響する「東海岸」などのデータセンターの物理的位置と，さらにその中での分離（同時に障害が起きにくい）
  - Elastic IP： 固定IPの発行と割り当て
  - Cloud Watch，Auto Scaling Group，Elastic Load balancing： 監視結果に基づき自動スケーリングと負荷分散

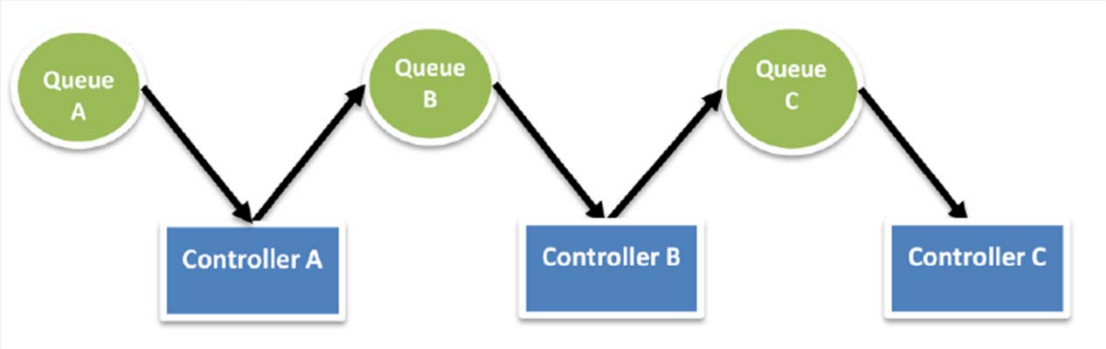
Amazon Web Services： 構成（２）

- 各種ストレージ
  - Erastic Block Storage： EC2インスタンスに割り当てられるストレージ
  - S3（Simple Storage Service）： 複製・分散されるデータストア（CloudFront： 世界中に分散も可能）
  - SimpleDB： Key-Valueペアによるデータストア
  - RDS（Relational Data Service）： RDBサービス
- メッセージング
  - SQS（Simple Queue Service）： 分散キュー
  - SNS（Sinple Notifications Service）： public-subscribeによるトピックベースのメッセージング

クラウド上でのアーキテクチャ（1）

Amazonのドキュメントを見てみると・・・

- コンポーネントの分割
  - 各コンポーネントの失敗や遅れが互いに影響しないように疎結合を
  - ⇨ 特にバッチ処理の場合など，水平（同機能の）スケールのためには非同期アーキテクチャに



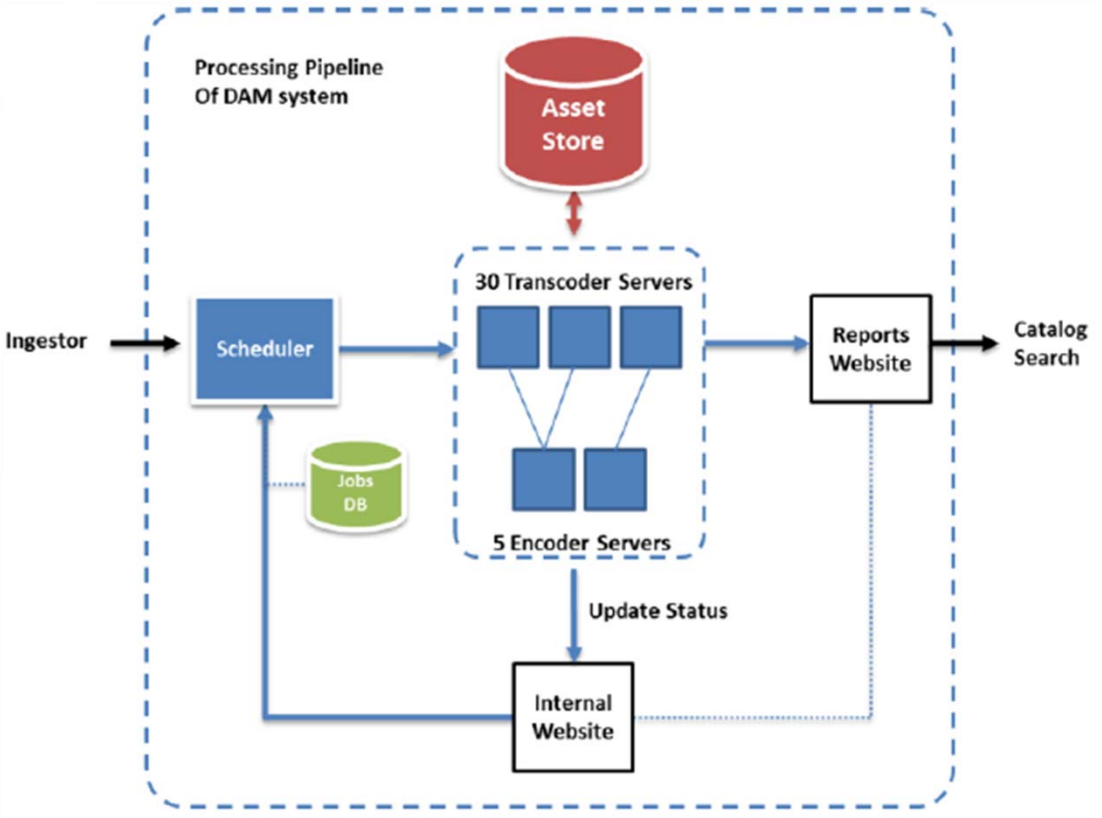
  - ⇨ 複製による並行処理・耐故障化を容易に

[J. Varia, Architecting for The Cloud: Best Practices, 2010]

クラウド上でのアーキテクチャ（2）

Amazonのドキュメントを見てみると・・・

- バッチシステムの移行例（before）

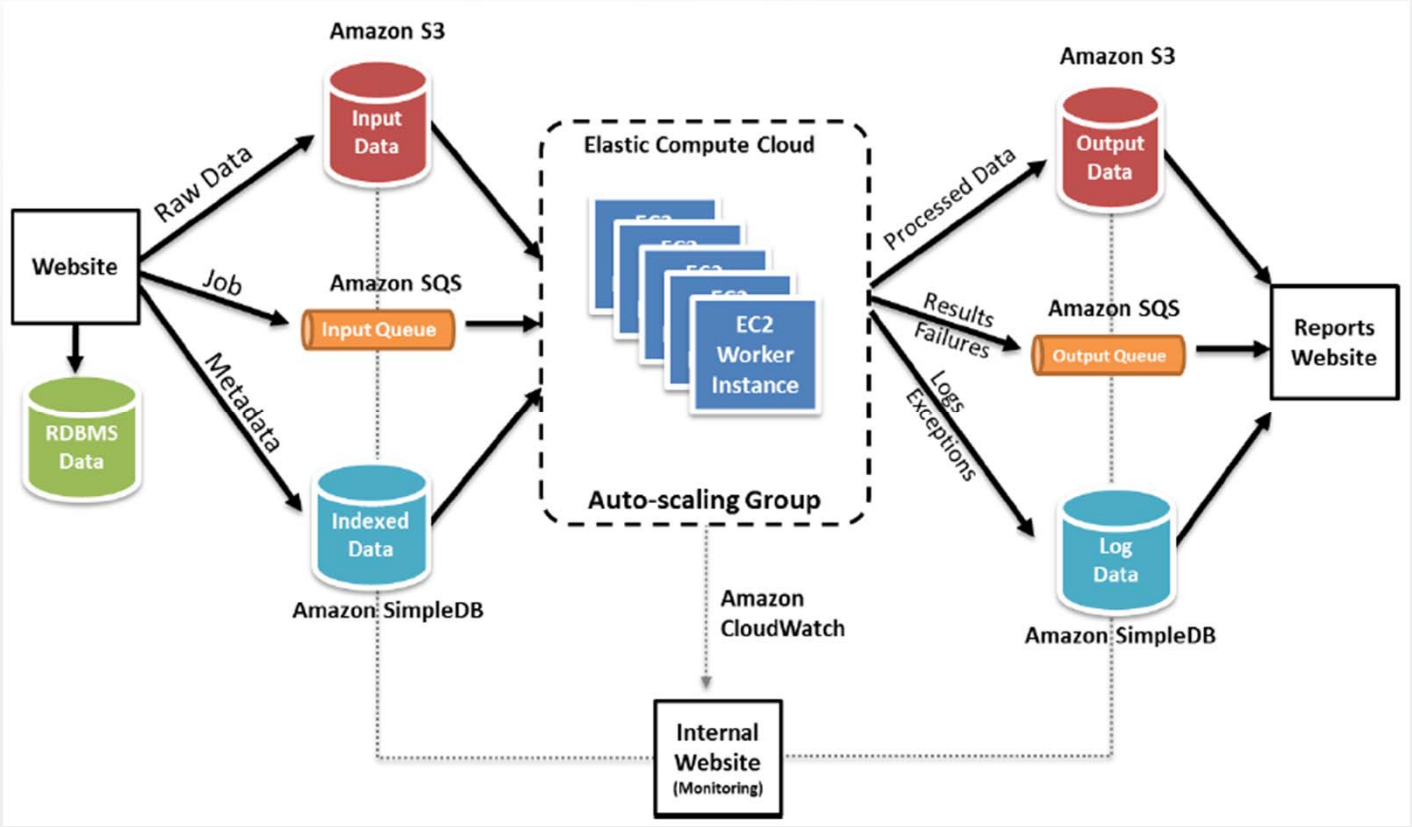


[J. Varis, Migrating your Existing Applications to the AWS Cloud, Oct 2010]

クラウド上でのアーキテクチャ（2）

Amazonのドキュメントを見てみると・・・

- バッチシステムの移行例（after）

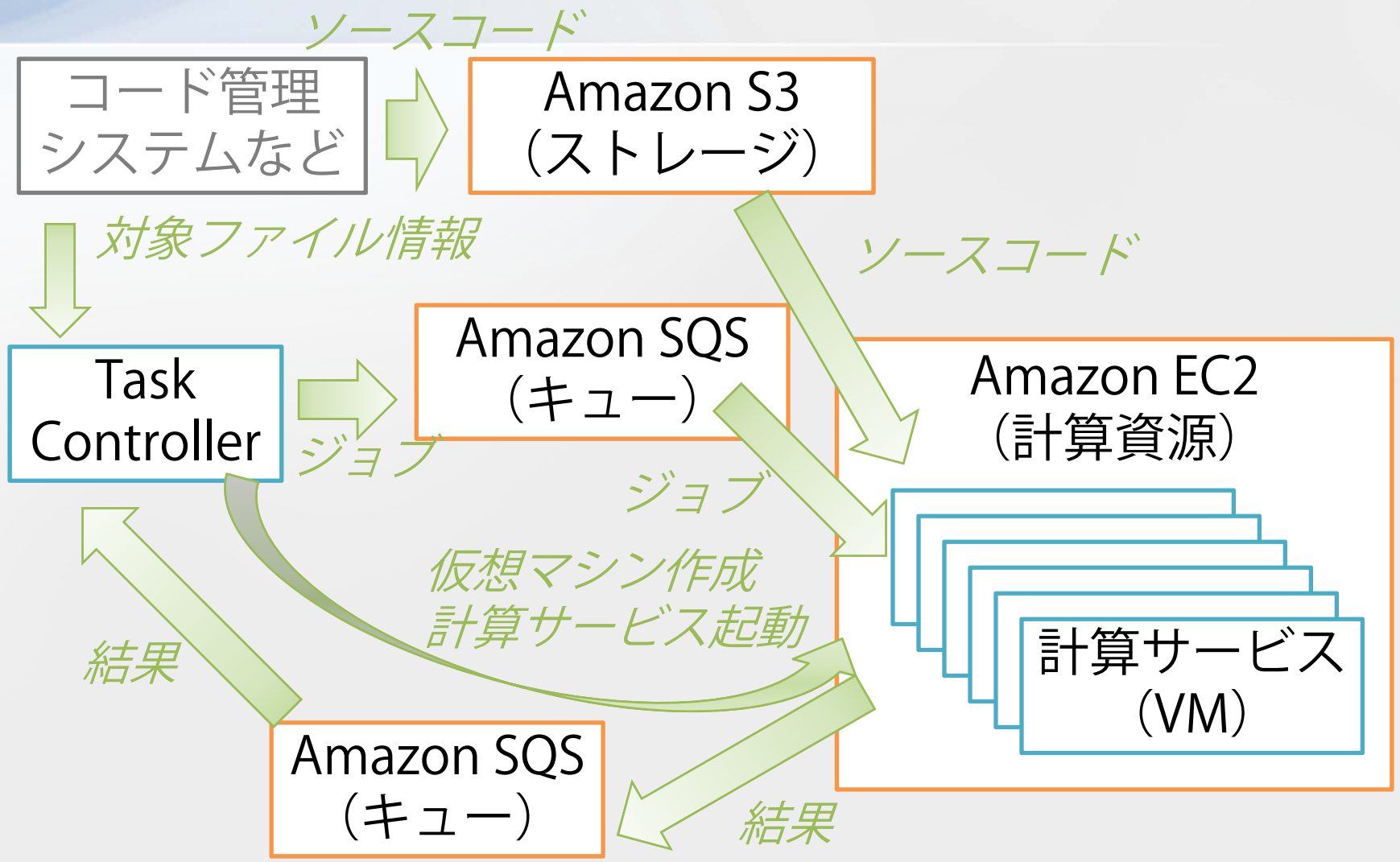


[J. Varis, Migrating your Existing Applications to the AWS Cloud, Oct 2010]

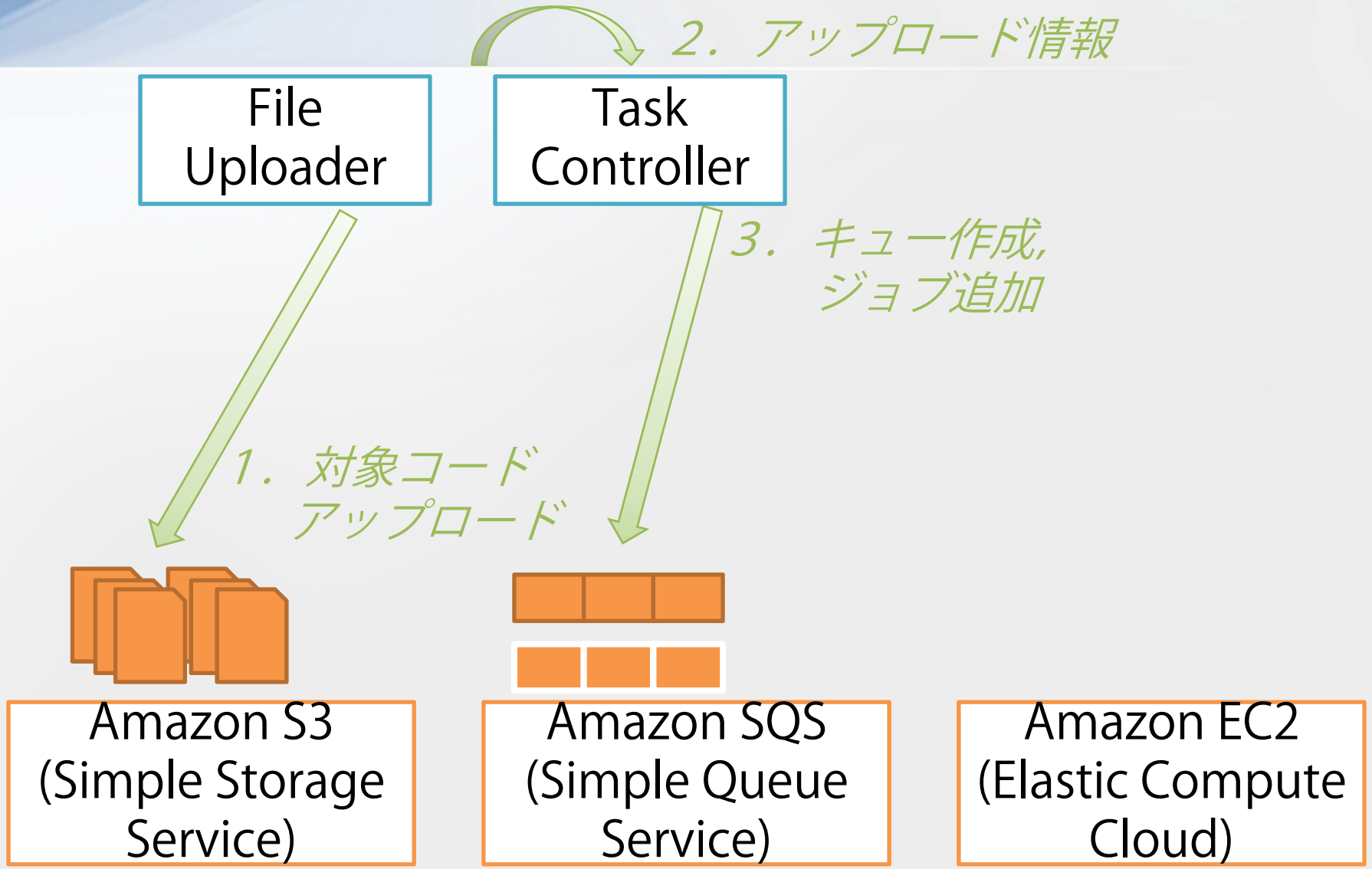
メモ： Scale Upと Scale Out

- 大量の処理（リクエストなど）をさばくには
  - Scale Up： 強力なサーバを用いる
    - 古典的なオンプレミスサーバ
  - Scale Out： 管理ソフトウェアツールとともに，大量のサーバを用いる
    - Webスケールのデータに対して（例：サーチエンジン）
    - 落ちているサーバが常にあるという仮定に基づいた運用
    - データや機能について，並行実行・障害復帰しやすい特定の形式を用いる
      （例：key-value DBやMap Reduce)

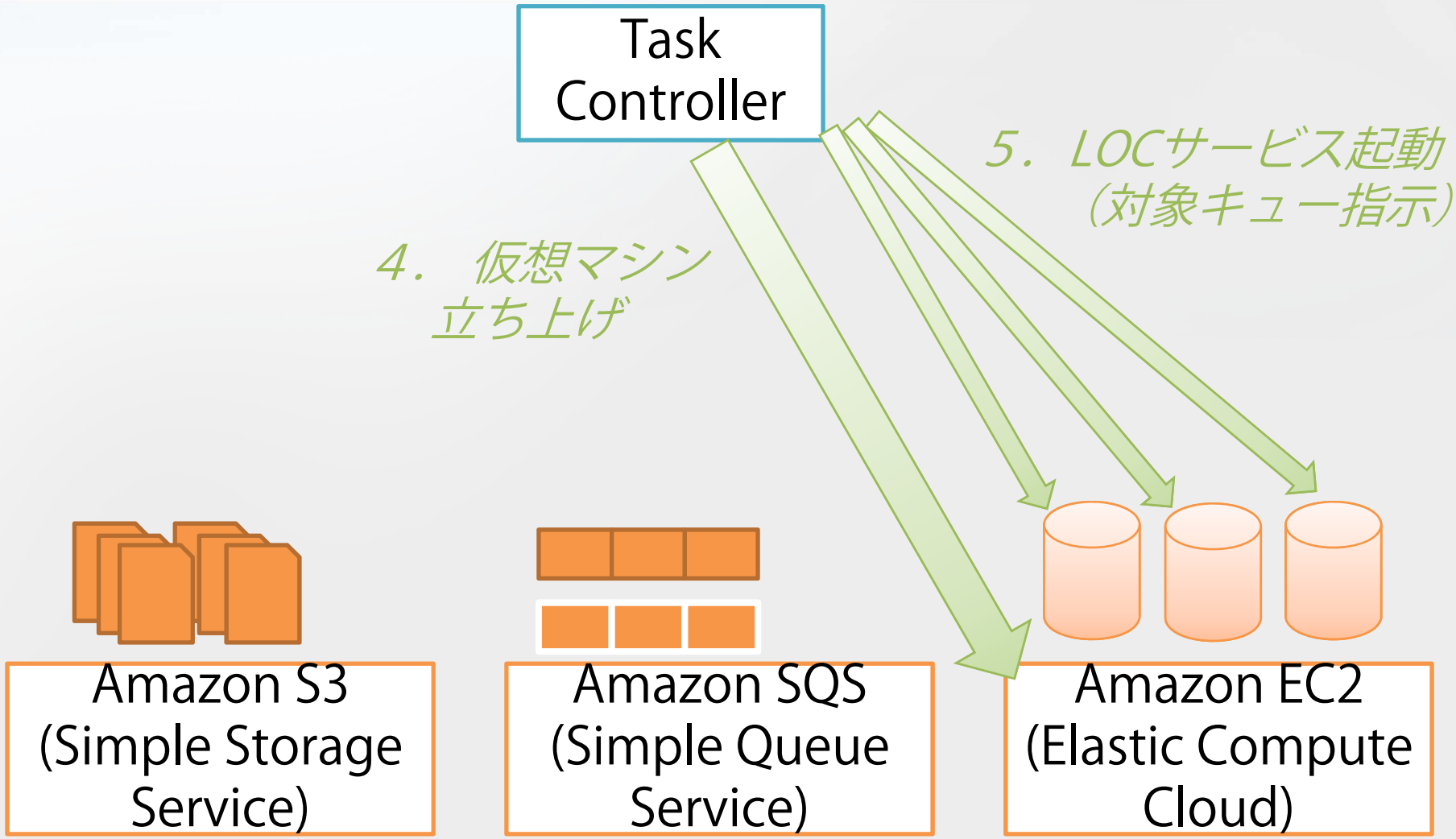
例題： 構成概要


ソースコード
コード管理
システムなど
Amazon S3
（ストレージ）
対象ファイル情報
ソースコード
Task
Controller
Amazon SQS
（キュー）
ジョブ
Amazon EC2
（計算資源）
ジョブ
仮想マシン作成
計算サービス起動
計算サービス
（VM）
結果
Amazon SQS
（キュー）
結果

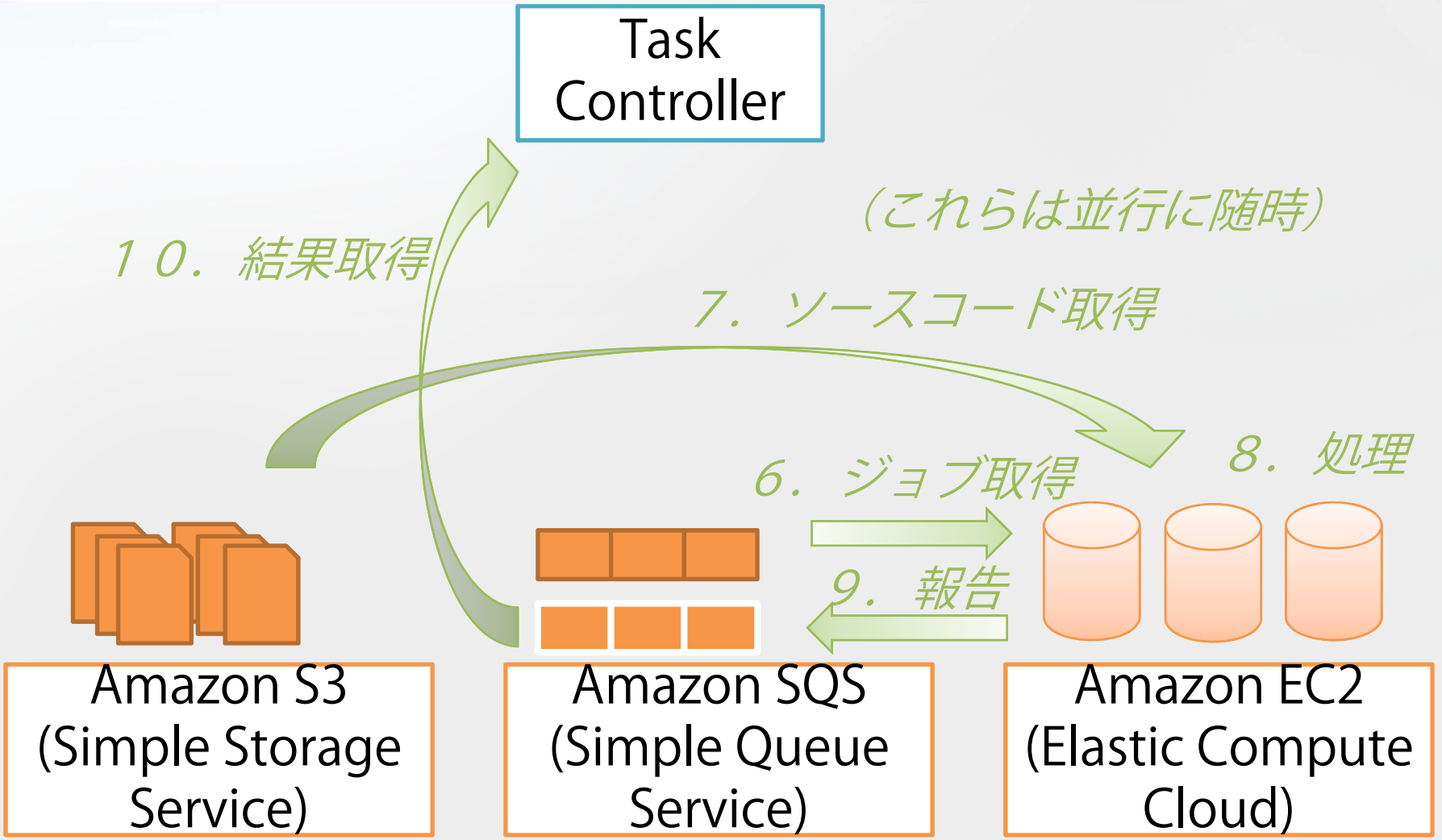
例題：動作概要（1）


2．アップロード情報
File Uploader
Task Controller
3．キュー作成，ジョブ追加
1．対象コード アップロード
Amazon S3 (Simple Storage Service)
Amazon SQS (Simple Queue Service)
Amazon EC2 (Elastic Compute Cloud)

例題：動作概要（２）


Task Controller
5．LOCサービス起動（対象キュー指示）
4．仮想マシン立ち上げ
Amazon S3 (Simple Storage Service)
Amazon SQS (Simple Queue Service)
Amazon EC2 (Elastic Compute Cloud)

例題：動作概要（３）


Task Controller
（これらは並行に随時）
１０．結果取得
７．ソースコード取得
８．処理
６．ジョブ取得
９．報告
Amazon S3 (Simple Storage Service)
Amazon SQS (Simple Queue Service)
Amazon EC2 (Elastic Compute Cloud)

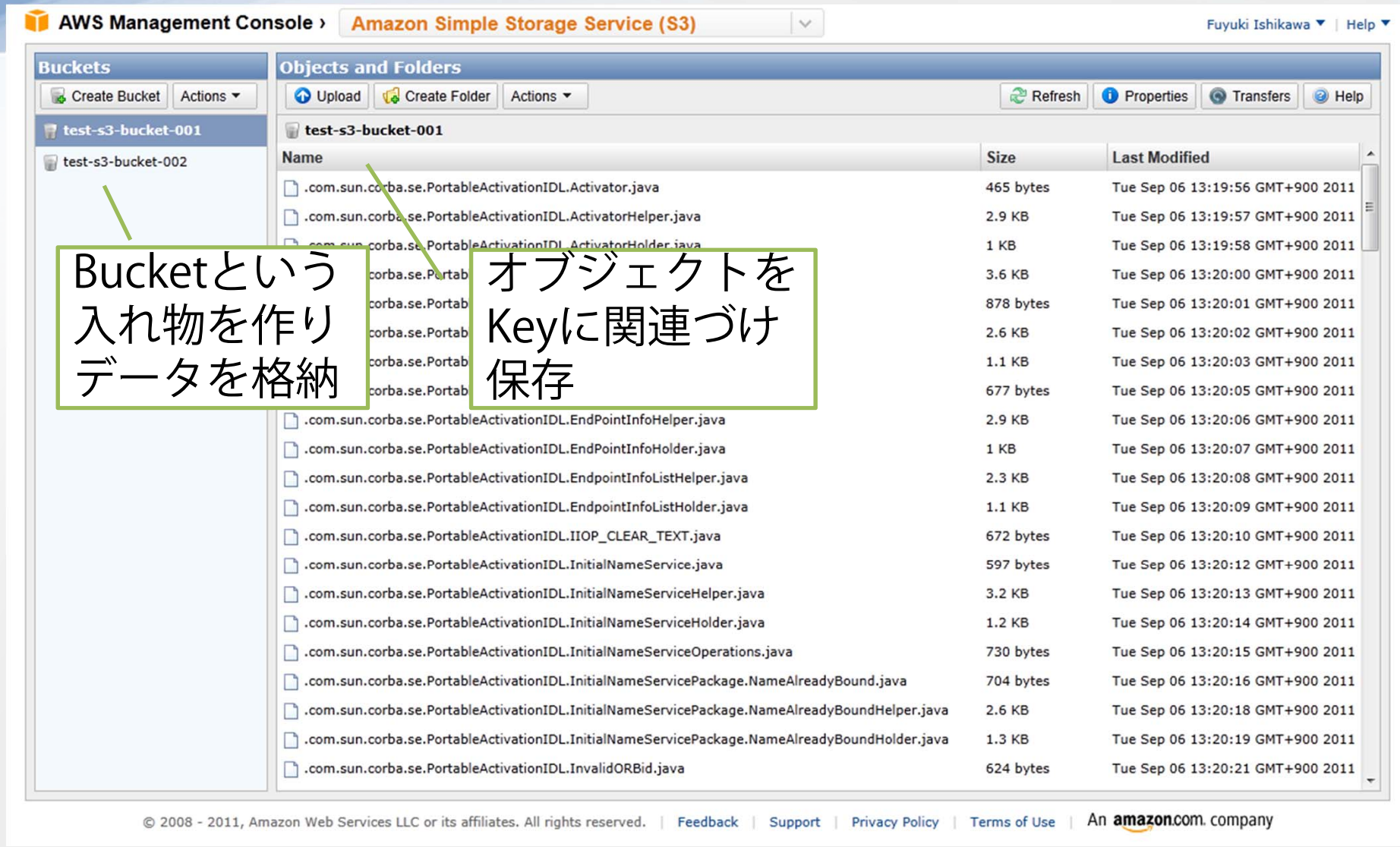
例題： 利用ライブラリ

- 今回はJavaベースで
  - Amazon Web ServicesのJavaライブラリを利用
    - http://aws.amazon.com/jp/sdkforjava/
  - LOCカウンタは，Axis2を用いてサービス化
    - http://ws.apache.org/axis2/
  - クライアントは別のライブラリ（JDK内のJAX-WS）を用いてみる
    - http://jax-ws.java.net/

例題：作業手順一例（１）

1. Amazon Web Serviceアカウント作成
   - クレジットカード登録，電話での自動確認がある
2. 1つのローカルファイルに対するLOCカウンタ作成・テスト
3. S3上のファイルを対象とするようにLOCカウンタを拡張
   - S3へのファイルアップローダ作成・テスト
   - S3上のファイルを読み出すようLOCカウンタ拡張・テスト

例題： Webインターフェース（S3）



例題：実際の作業（S3の利用コード）

```
AmazonS3 s3 = new AmazonS3Client(…);
```

認証情報を与えスタブ作成

```
try {
  s3.putObject(new PutObjectRequest(bucketName, key, file));
} catch (AmazonServiceException e) {
  e.printStackTrace();
} catch (AmazonClientException e) {
  e.printStackTrace();
}
```

ファイルのPUT

Internal Server Errorなど

ネットワークエラーなど

以後例外処理省略

```
  S3Object obj = s3.getObject(
                      new GetObjectRequest(bucketName, key));
  InputStream is = is. getObjectContent();
  …
```

ファイルのGET

[http://aws.amazon.com/jp/documentation/s3/]

例題：メモ（S3）

- データ移動の遅延・課金
  - 今回（デフォルトの東海岸利用）
    - 国内のある程度速いであろう場所からのアップロードに1ファイル1秒弱，7000個強で2時間弱
    - 全体で70MB程度なので，通信課金は1円未満
  - 今回省いたが
    - 必要ならRegionも指定すべき
    - 送る側も並行化しどんどん送るべき
    - 圧縮してあちら側で展開し保存するなど，工夫すべき（データセンター内の通信は無料・速い）
    - 本当にデータが膨大なら「USBディスクを郵送」などの方が安い・速い

例題：作業手順一例（２）

5. SQS経由で複数ファイルを扱うように拡張
   - アップロードした各ファイルのS3上でのキーを，SQS上のリクエストキューに送るタスクコントローラを作成
   - LOCカウンタにおいて，SQS上のリクエストキューから1つずつキーを取得し，それを用いてS3上の各ファイルを処理するように拡張・テスト
6. 結果もSQSを経由するように拡張
   - LOCカウンタにおいて，各ファイルの処理結果をSQS上の結果キューに1つずつ送るように拡張
   - タスクコントローラにおいて，結果キューから結果を読み取るように拡張・テスト

例題：実際の作業（SQSの利用コード）

```
AmazonSQS sqs = new AmazonSQSClient(…);
```

認証情報を与えスタブ作成

```
queueURL = sqs.createQueue(
                    new CreateQueueRequest(queueName)
                ).getQueueUrl();
```

キューの作成

```
sqs.sendMessage(
        new SendMessageRequest(queueURL, message));
```

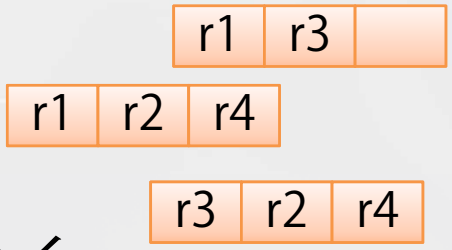
メッセージ送信

```
List<Message> messages =
   sqs.receiveMessage(
            new ReceiveMessageRequest(queueURL));
for(Message mes: messages){ … }
```

メッセージ受信

[http://aws.amazon.com/jp/documentation/sqs/]

例題：メモ（SQS）



- SQSのキューは複製分散されている
  - 全てのキューが（順序はさておき）同じデータを持つ状態に「収束」していく
  - キューを読み出したとき，（絶対的な時間で）その前に送られたメッセージが含まれるとは限らない
- 「キューを空にしたら終了」と書いてしまうと，全タスクが実行されないことがある
  - タスクコントローラ側では，送ったタスクと受け取ったタスクを照らし合わせて終了判断可能
  - 複数インスタンスがあるLOCカウンタ側は，タスクをすべて裁いたかは自身だけでは判断不可能

    （今回はいずれもタイムアウトを設けている）

例題：メモ（SQS）

- SQSのキューからメッセージを読み出しても，キューにメッセージが残る
  - 読み出したノードは，処理後にメッセージを明示的に削除する必要がある
  - 読み出されたメッセージは見えなくなるが，一定期間後に再び見えるようになる
    - 読み出したノードが，処理完了せずに落ちてしまい削除が行われないような場合を想定
- 「少なくとも1回実行」を目指している仕組み
  - 「高々1回実行（ちょうど1回実行）」が必要な場合は支援の範囲外（アプリケーション側で同期する）

例題：作業手順一例（３）

7. LOCカウンタをサービス化

- LOCカウンタをサービス化しローカルホスト上で実行
- 呼び出しクライアントを作成・テスト
- タスクコントローラは，LOCカウンタをサービスとしてHTTP経由で呼び出すように変更・テスト

例題： 実際の作業（Webサービス化）

- サービス側： Apache Axis2を利用
  1. 通常のJavaクラスとしてLOCカウンタ実装
  2. サービス設定ファイルにて，サービス操作として公開するクラス・メソッドやその名前空間などを指定
  3. クラスファイルと設定ファイルをAxis2インストールディレクトリ内に移動，自動で取り込ませる
- クライアント側： JDKを利用
  1. 上記#3にて生成されたWSDLファイルから，wsimportコマンドにてスタブを生成
  2. そのスタブを用いてLOCカウンタを呼び出すようなクライアント側の動作を実装

（実装は双方であえて変えてみている）

例題：実際の作業（Axis2サービス設定）

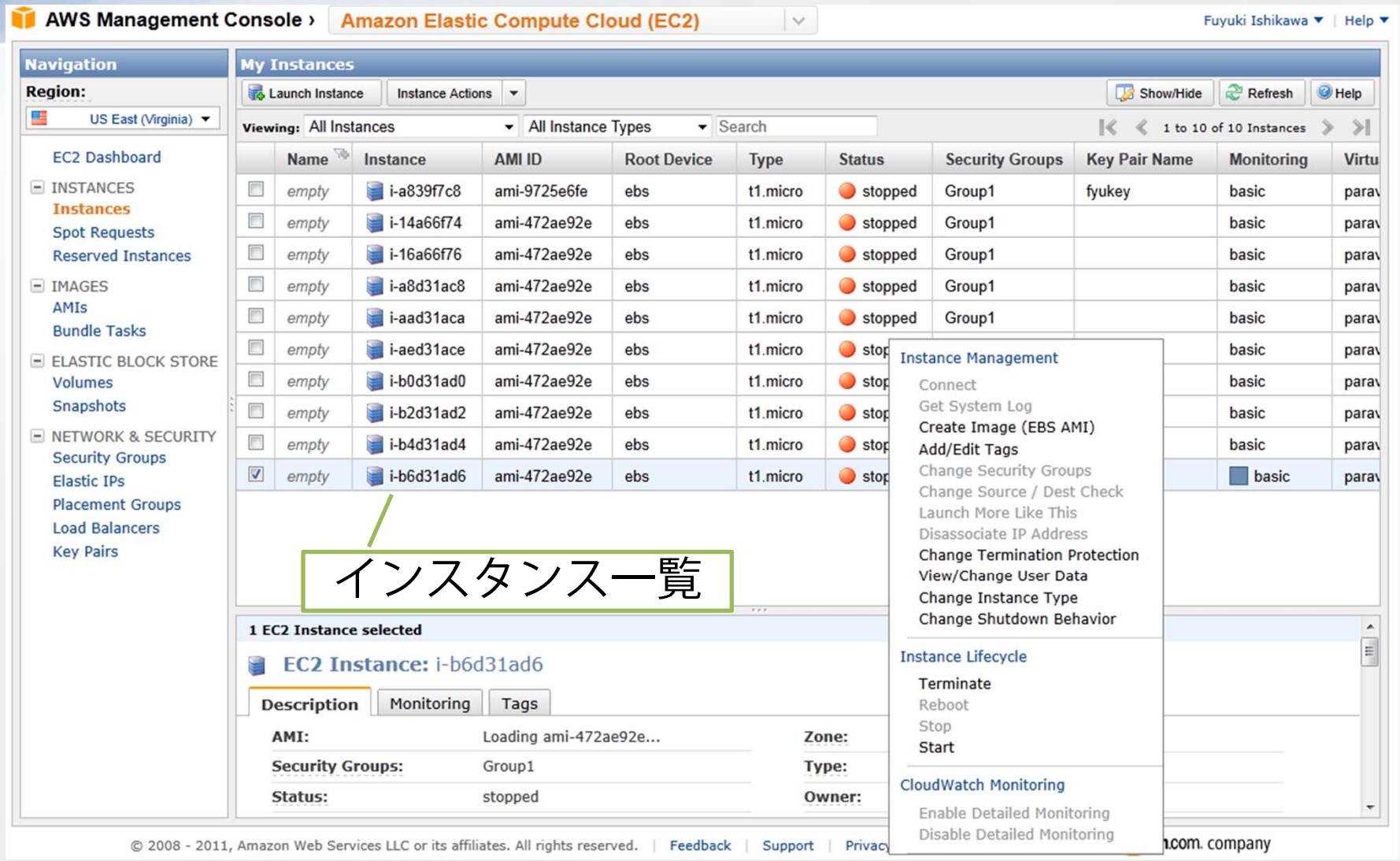
```
<service name="LocCountService“
    targetNamespace="http://service.cloudloc/">
 <description>Count Lines of Code of a file on
                  Amazon S3</description>
 <schema schemaNamespace="http://service.cloudloc/"/>
 <parameter name="ServiceClass">
    cloudloc.service.LocCountService
 </parameter>
 <messageReceivers>
  <messageReceiver
     mep="http://www.w3.org/2004/08/wsdl/in-only"
     class="org.apache. … .RPCInOnlyMessageReceiver"/>
 </messageReceivers>
</service>
```

このクラスをサービス化
（publicメソッドを公開）

例題：作業手順一例（４）

8. LOCカウンタサービスをEC2上に移行
   - LOCカウンタサービスを走らせる仮想マシンを作成
   - タスクコントローラからの呼び出し先を仮想マシン上のサービスに変更
9. LOCカウンタサービスを含むVMを複数立ち上げてテスト

例題： Webインターフェース（EC2）



例題： 実際の作業（VM作成）

1. 出来合のVMイメージ（AMI： Amazon Machine Image）から適当に選択し，インスタンスを起動
   - Amazon製の基本Linuxなど
2. そのインスタンスのアドレスを取得し，sshやリモートデスクトップで接続
3. Tomcat/Axis2などのインストールやファイアウォール（ホスト内，Amazon両方）設定後，LOCサービスをセットアップし，動作確認
4. その時点でのVMイメージを新しく保存
   - 以後何個でもインスタンスを作成可能に

例題：作業手順一例（５）

10.VM立ち上げを自動で行うように拡張

- タスクコントローラはVMを立ち上げ，そのアドレスに対してLOCサービス呼び出しを行うようにするとともに，一定時間経過後にVMの終了も行う

例題：実際の作業（EC2の利用コード）

```
AmazonEC2 ec2 = new AmazonEC2Client(…);
```

認証情報を与えスタブ作成

```
RunInstancesRequest req
    = new RunInstancesRequest(amiID, minNum, maxNum);
req.setInstanceType(instanceType);
req.withSecurityGroupIds(securityGroupID);
RunInstancesResult res = ec2.runInstances(req);
```

各種の設定とともにインスタンス起動

- イメージID
- 立ち上げ個数（最小・最大）：同時立ち上げ可能な数の制限と照らし合わせて実際の個数が決定
- インスタンスタイプ（CPU/Memoryのサイズを表す）
- セキュリティグループ（ファイアウォール設定を指定）

[http://aws.amazon.com/jp/documentation/ec2/]

例題：実際の作業（EC2の利用コード）

```
AmazonEC2 ec2 = new AmazonEC2Client(…);
```

認証情報を与えスタブ作成

```
DescribeInstancesResult res
    = ec2.describeInstances(
                new DescribeInstancesRequest());
```

現在起動中のインスタンス情報の取得
（例：立ち上げたインスタンスの起動処理が終わり利用可能になっているかチェックするため）

```
ec2.stopInstances(new StopInstancesRequest(instanceIDs));
```

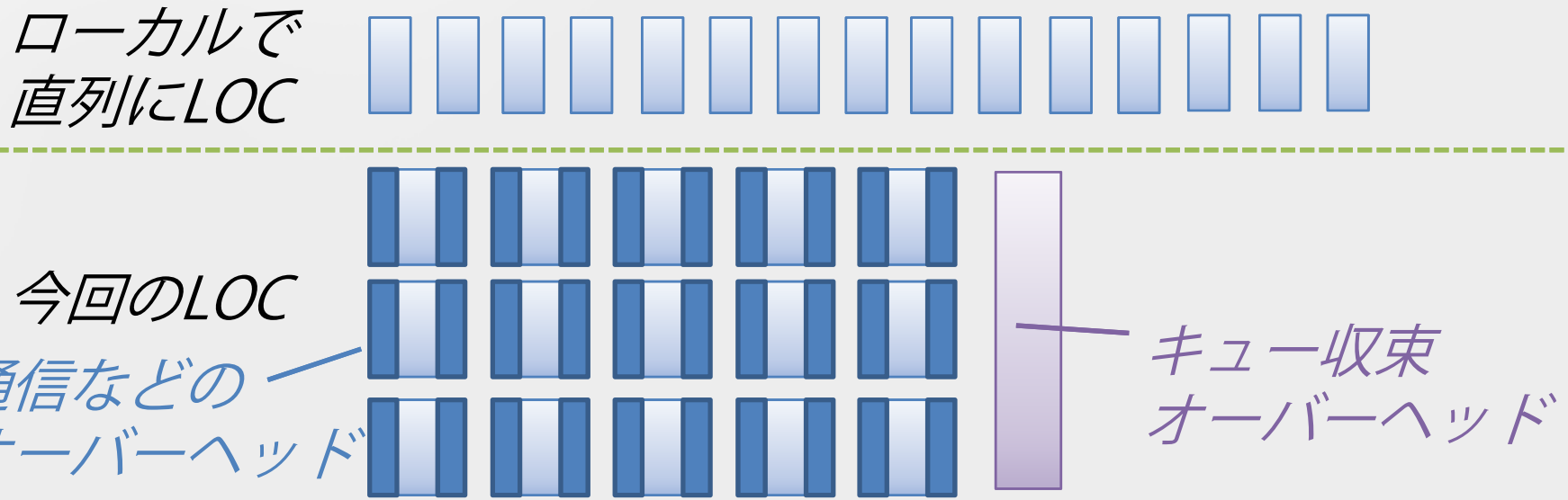
インスタンス停止

[http://aws.amazon.com/jp/documentation/ec2/]

実行結果

- Java 6のソースコード（ファイル7000個超）
  - S3へのアップロード（並行化せず）：2時間弱
  - キューへの送信（並行化せず）：約20分
  - VM10個を立ち上げ，全結果が集まるまで：約1時間

注：性能最適化していない，LOC処理が小さいので通信などのオーバーヘッドが相対的に大きい



目次

- クラウドとは
- Webサービスの技術的基礎
- 例題： AWS上のLOC
- 議論・おわりに
- 参考文献

例題の限界

- EC2サービスを直接使わずとも，自動スケーリングなど多様な機能が利用できる
  - Amazon自体のサービス
  - サードパーティのサービス・ライブラリ
  - 他の提供者によるIaaS/PaaS
- VMの設定はもっと柔軟・動的にもできる
  - 指定されたサービス設定を起動時に読み込み，必要なサービス配備を動的に行う，など
- 処理時間が非常に短い
  - 並列化効果に対して，キューの情報反映や通信のオーバーヘッドが大きくなりがち

おわりに： SEに活かす？（1）

クラウドが活きる特徴とSEにおけるタスク

- 分割して独立に捌ける処理を大量に行う
  - ビルド，インテグレーション，テスト，静的検証，メトリクス解析，リファクタリング，・・・

    （分割できる？どう分割？）
- 様々な仮想マシンを手軽に自動で利用する
  - 異なるCPU速度・メモリ量，異なるOSやパッチ状況，異なるソフトウェア（ブラウザなど），異なる環境設定ごとのテストを行いやすい

おわりに： SEに活かす？（２）

クラウドが活きる特徴とSEにおけるタスク

- データ転送（コスト・オーバーヘッド）が少ない方がよい
  - コードのデータ量はたかがしれている
    （が，取り得る入出力やたどりうる実行パスなど，組み合わせでテストなどの処理は膨大になり得る）
- 時間順序や最新情報の正確な反映が必ずしも必要でない場合の方がよい
  - SEにおけるタスクはWeb検索などと異なり，そんなことはない？
  - バックアップやアーカイブを対象としたメトリクス分析など？

おわりに

今回は本当に基礎的な内容

- これくらいでもぜひご自身で触ってみて下さい（これくらいならほぼ無料で済む）
- それが難しくとも，本チュートリアルが何か考えるための「実感」を提供できたなら幸いです

ご自身の「専門領域」でぜひサービス・クラウドとの関連を考えてみて下さい！

- 自分のツールをサービス化・公開？
- 研究・開発にWebサービス・クラウドを利用？
- 特にクラウドにおける課題を同定，研究対象に？

目次

- クラウドとは
- Webサービスの技術的基礎
- 例題： AWS上のLOC
- 議論・おわりに
- 参考文献

参考文献（クラウド全般）

- *The NIST Definition of Cloud Computing (Draft)*

  Special Publication 800-145, NIST, Jan 2011
  http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.pdf

- *Above the clouds: A berkeley view of cloud computing*

  M. Armbrust et. al., Technical Report No. UCB/EECS-2009-28, Feb 2009

- *クラウド大全 サービス詳細から基盤技術まで（第２版）*

  日経BP社出版局，2010

- *Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility*

  R. Buyya et. al., Future Generation Computer Systems, 2009

参考文献（Amazon Web Services）

- *Amazon Web Services*

  http://aws.amazon.com/jp/
- *AWS Documentation*

  http://aws.amazon.com/jp/documentation/
- *AWS SDK for Java*

  http://aws.amazon.com/jp/sdkforjava/
- *Architecting for The Cloud: Best Practices*

  J. Varia, http://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf
- *Migrating your Existing Applications to the AWS Cloud*

  J. Varia, *http://media.amazonwebservices.com/CloudMigration-main.pdf*

参考文献（Webサービス）

- *W3C Web Services Activity*
  http://www.w3.org/2002/ws/
- *Apache Axis2 / Java*
  http://axis.apache.org/axis2/java/core/
- *JAX-WS Reference Implementation — Java.net*
  http://jax-ws.java.net/
- *Webサービスコンピューティング*
  青木 利晴（監修），電子情報通信学会，2005
- *Webサービスプラットフォームアーキテクチャ*
  S. Weerawaranaら，エスアイビーアクセス，2006
- *RESTful Webサービス*
  L. Richardsonら，オライリー・ジャパン，2007