Asakusa Framework Day 2016

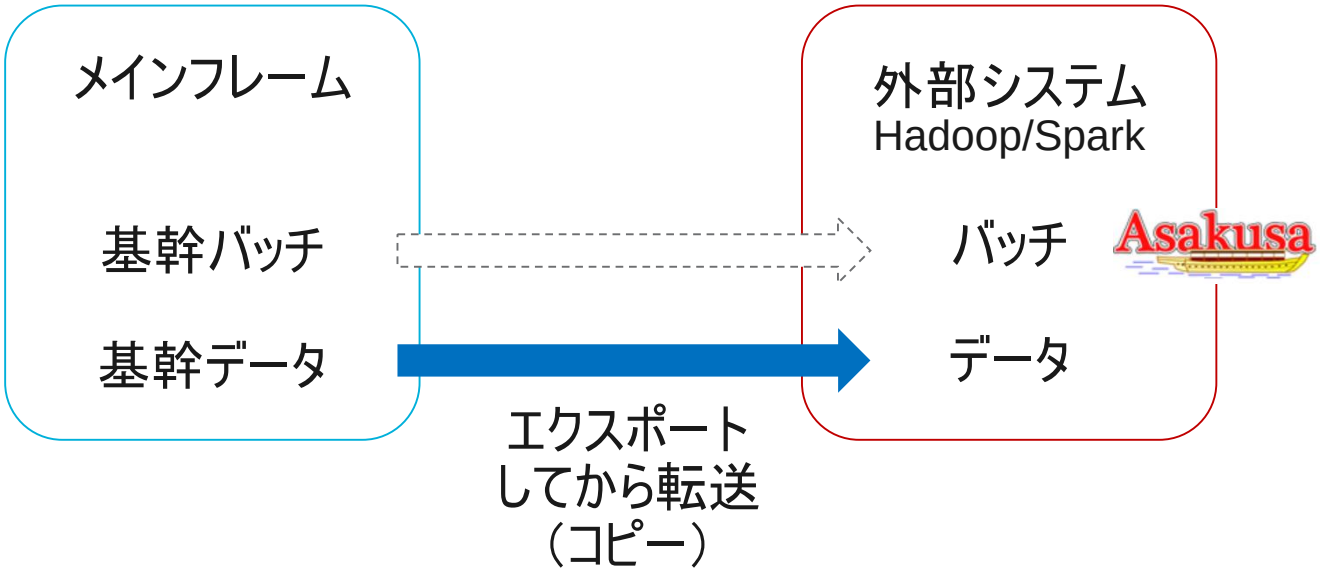
Asakusa Frameworkをメインフレーム上に食わせるとどうなるか？

2016年11月25日

日本アイ・ビー・エム株式会社
IBM Systems ハードウェア事業本部
藤岡英典（hfujioka@jp.ibm.com）

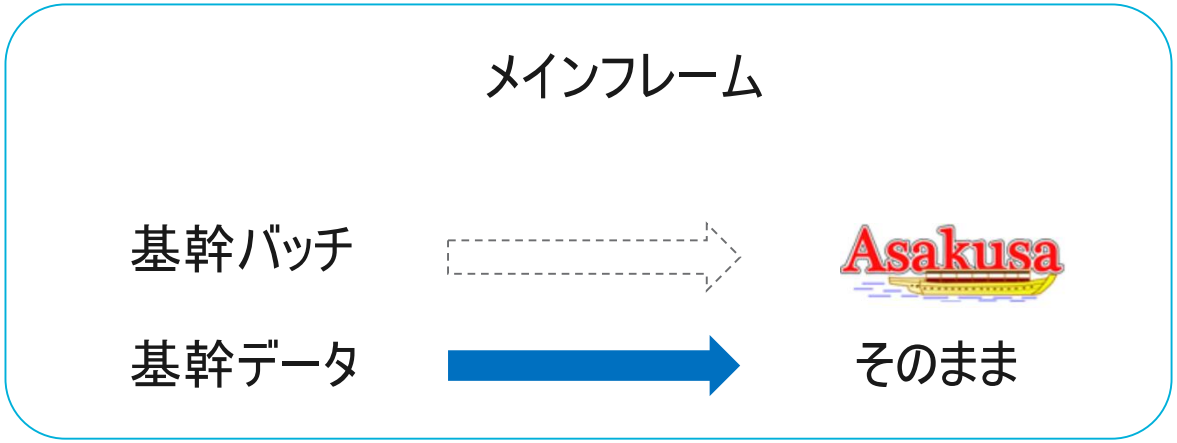
IBM

メインフレームの基幹バッチをAsakusa環境で実行・・・？



普通の発想は、データを外部にコピーして処理させる

でも、もしできるならシンプルにしたいですよね？



本当にできるの？

本当にやる意味あるの？

ビジネス上の必然性はあるか？

セキュリティ

処理の外出しを検討したが、基幹データなので外部システムにエクスポートしたくない。

スキル

これからはJavaのような、主流なスキルセットを持つ人財にも適した仕組みが必要。

コスト

CPU消費を抑えることで、月額のコストを削減できないか。

これらのビジネス課題を　Asakusa+メインフレーム　で解決できるかを検証しました。

しかし、この二つは相性が合うのでしょうか？

→メインフレームについて、少し振り返ることから始めます

ホストとは？ メインフレームとは？汎用機とは？

- ホスト（コンピューター）とは
  企業の基幹業務システムなどに用いられる汎用大型コンピュータ。電源やCPU、記憶装置を始めとするほとんどのパーツが多重化されており、並列処理による処理性能の向上と対故障性能の向上が図られている。

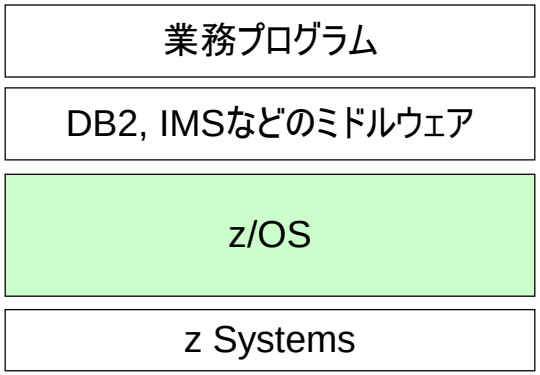
- メインフレームとは
  高速、大容量のコンピュータの総称。個人レベルでの所有を基本とするパーソナルコンピュータに対し、複数のクライアントコンピュータからの要求を集中的に処理する大型コンピュータなどを指す。

- 汎用機とは
  1964年に発表されたIBMのS/360は、それ以前は別々のコンピューターで処理していた「商用計算（10進数演算）」と「科学技術計算（浮動小数点演算）」を両方処理できるという特徴を持っていた。この両方のタイプの演算が行えるS/360は「汎用的な計算ができる」ため「汎用機」または「汎用コンピュータ」と呼ばれた。

「ホスト」＝「メインフレーム」＝「汎用機」

z/OSの役割

- IBMが1964年にS/360用のOSとして発表。
- メインフレーム用OSとして普及。
- もともとは、MVSと呼ばれていて、それがOS/390、z/OSと進化。



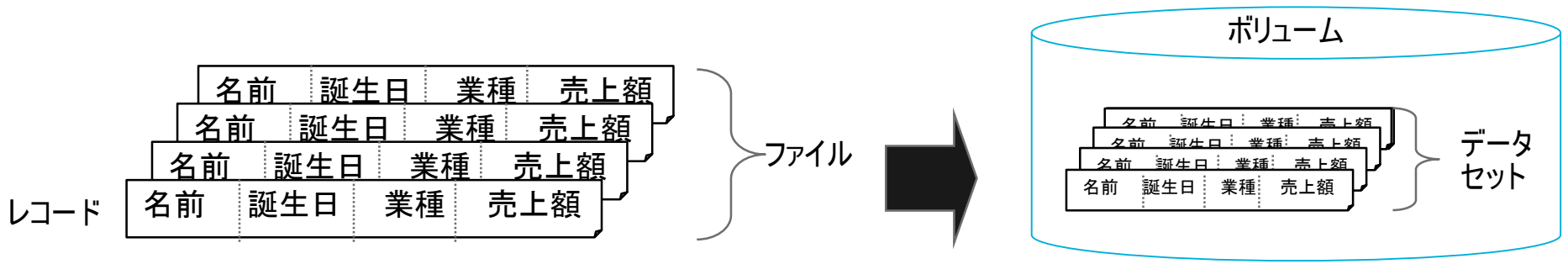
- **記憶域管理**
  物理メモリーとディスク上のページ領域を使用して、複数のジョブが物理メモリーの容量以上にメモリーを使えるようにする。
- **対話型処理**
  TSO（Time Sharing Option）と呼ばれるユーザーインターフェース。
- **入出力管理**
  データを読み込む際に、ディスク上の物理的な位置を意識することなくアクセスできるようにしたり、端末やプリンタなどの出力装置の管理を行う。
- **多重プログラミング**
  CPUを効率よく使用するように、同時に複数のプログラムを実行させる機能。
- **ジョブ管理**
  バッチ処理やオンライン・トランザクション、その他システム系のジョブの制御を行う。
- **システム資源管理**
  ハードウェア資源を最も有効利用できるように処理の優先度をコントロールする。

バッチアプリケーションは大得意

z/OSのデータの構造

- データセット
  “関連のあるデータの集合体”。（パソコンでいうとファイルとフォルダを合わせて考えるイメージです。）

顧客の情報が「名前」「生年月日」「業種」「売上額」といった項目に整理されている場合、1顧客分の情報に相当する一連のデータ群を**レコード**と呼び、複数のレコードが集まったものを**ファイル**と呼びます。このファイルは通常磁気ディスク装置や磁気テープに保存されます。このように磁気装置に記録されたファイルを、ホストの世界では**データセット**と呼びます。



データ＋ファイル＋ディレクトリー　＝　データセット

z/OSのデータの構造

- データセットの種類
  - アクセスの仕方や処理の方法により、以下に挙げられる種類のデータセットがあります。

編成	用途	アクセス方式	処理方法	概要
順次編成	中間ファイル	SAM Sequential Access Method	順次処理	一連のレコードを物理的な順番に連続して処理する。なお、テープ装置に記録する情報はすべてSAMになる。
直接編成	あまり使われない	DAM Direct Access Method	直接処理	ディスク上のアドレスを用いて処理。ランダムにアクセスを行う処理に向いているが容量の無駄がでる。
区分編成	ライブラリ	PAM Partitioned Access Method	直接処理	同一属性を持ったレコードの集まりである“メンバー”から構成されている。メンバーは直接選択する事が出来るがメンバー内のレコードは順次処理される。 “AAAA.X.GYOMU32.JCLLIB(#BINDALL)” こういう書き方をした場合は”～JCLLIB”までを区分DS、”#BINDALL”をメンバーと呼ぶ。 ※パソコンのフォルダとファイルに近いイメージ
VSAM編成	データベース	VSAM Virtual Storage Access Method	直接処理/順次処理	上記編成の長所を併せ持つ。アクセスが早く容量の無駄もない。データベースの実体がこのVSAMファイルである。

オープン系とは別世界。文字コードは「EBCDIC」系

z/OSのバッチジョブ管理

- JCLとJES
  - ユーザーから見て、システムに実行させる仕事の基本的な単位をジョブと呼びます。
    例：支店での帳票1起動が1ジョブ。地域全支店の連続起動も1ジョブ。DB作成も1ジョブ。コンパイルも1ジョブ。
  - ジョブはJCL（Job Control Language）と呼ばれるジョブ制御言語で記述され実行される
  - JCLはソースと同じようにコーディングして作成。ジョブの実行の事をサブミットと呼ぶ。
  - サブミットされたジョブはJES（Job Entry Subsystem）で管理され、スプールと呼ばれる一時保管場所にコピーされて実行までの順番を待つ。

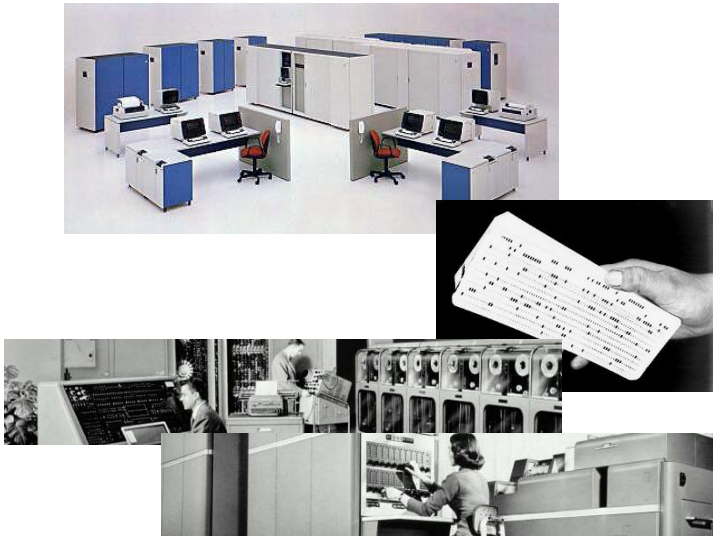
```
//AABBCCDD JOB  'ACCT#',
//         CLASS=D,MSGCLASS=X,COND=(4000,LE),
//           REGION=4M,TIME=1440
//* ジョブネット名称 :                                                *
//*===============================================================
//OUT1     OUTPUT CHARS=(N16N,M32F),PRMODE=SOSI1,PAGEDEF=V06483,
//           PIMSG=(NO,O)
//*
//S015     EXEC PGM=IKJEFT1B,DYNAMNBR=20,COND=(8,LE)
//STEPLIB  DD DSN=AAAA.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD DSN=AAAA.BBBB.CC.CNTLST,DISP=(,CATLG),
//            SPACE=(CYL,(1,1),RLSE),UNIT=SYSDA
//SYSUDUMP DD SYSOUT=*
//SYSTSIN  DD *
  DSN SYSTEM(DBJ3) RUN PROGRAM(JSNTEP2) PLAN(JSNTEP2) -
      LIB('DBJ3.RUNLIB.LOAD')
  END
/*
//SYSIN    DD DSN=AA    .BBBB.CC.CNTLIB(PGM010),DISP=SHR
```

ロードモジュール「JSNTEP2」を実行せよ

この場所に記録してあるファイルを読み込んでパラメータとせよ

メインフレーム＝ジョブ（+データ）管理システム

オープン・メインフレームとしての進化



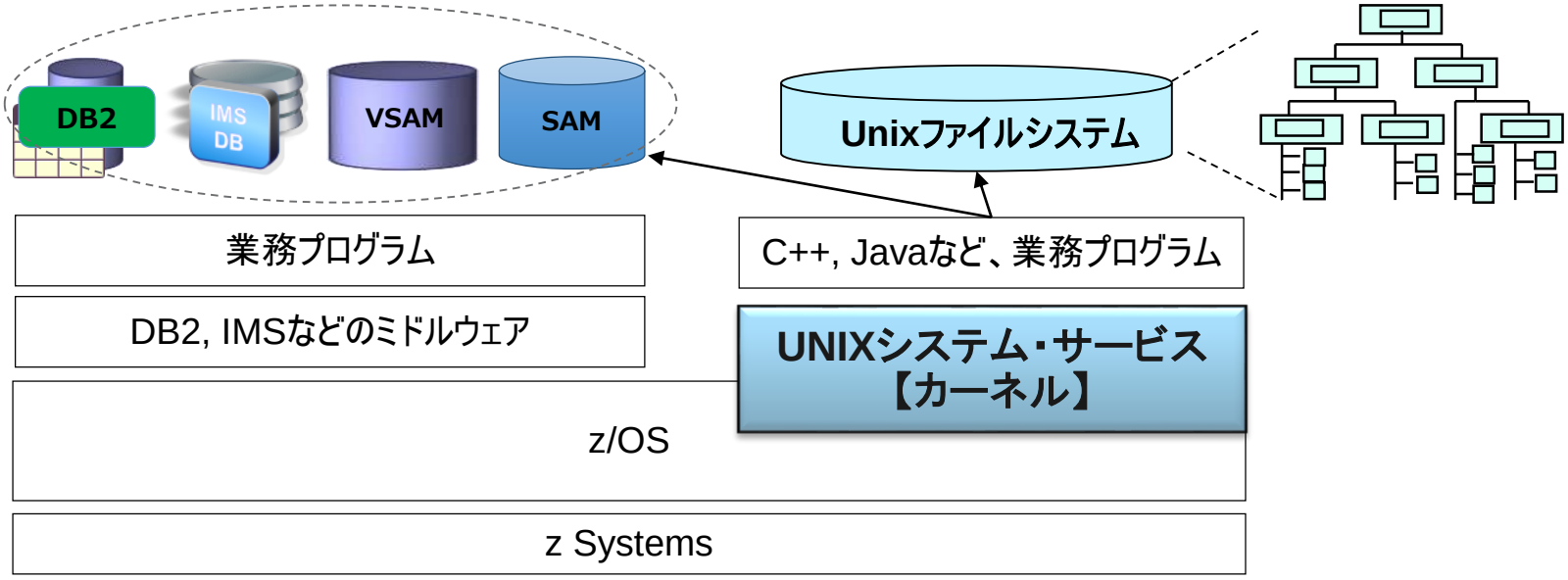
基幹システム（レガシー）としての使われ方 + オープン化

- これまで蓄積したお客様の重要な業務アプリケーション資産を継承し、
- 仮想化技術やUnix、Linux技術の採用でオープン化をはかり、
- クラウド、アナリティクスのインフラストラクチャーを構築する上で、
- 絶対的に必要となる可用性、セキュリティ、スケーラビリティ、柔軟性、管理性

を兼ね備えた、**オープン・メインフレーム**として進化

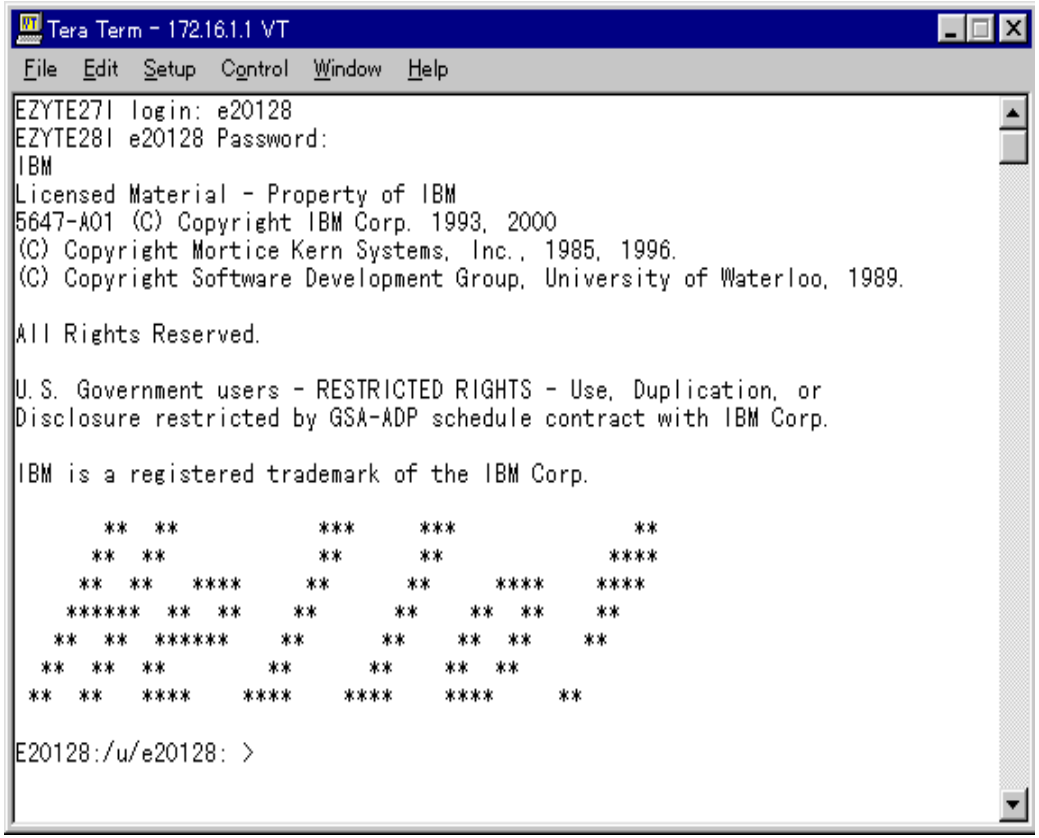
z/OSの中にいるUNIXの顔

- z/OS Unixシステムサービス(USS)はz/OSの一機能として、z/OS上でUNIXを実行
  - 分散系で開発された製品のポーティングが容易なため、開発コストが安くなり、かつ品質的にもメリット
    - TCP/IP
    - Java
- USSだけを使っている場合には、ユーザーには他のUNIXとほとんど同じ環境を提供
  - シェル環境の提供
  - シェルスクリプトの実行
  - UNIXと同様のディレクトリ構造



USSログイン・シェル

- ASCII端末から直接のログイン
  - TCP/IP アプリケーションの利用
    - telnet
    - ssh
- UNIXサーバーと同様の使い勝手
  - UNIXエディタが使用可能
    - vi
- Linuxのシェルとの違い
  - データ特性
    - 表示される文字コードはEBCDIC。必要に応じてASCII 変換する。
  - bash
    - USSのデフォルトはBシェル。別途、bashを導入できる。Sparkはbash上で実行。



IBM z/OS Platform for Apache Spark

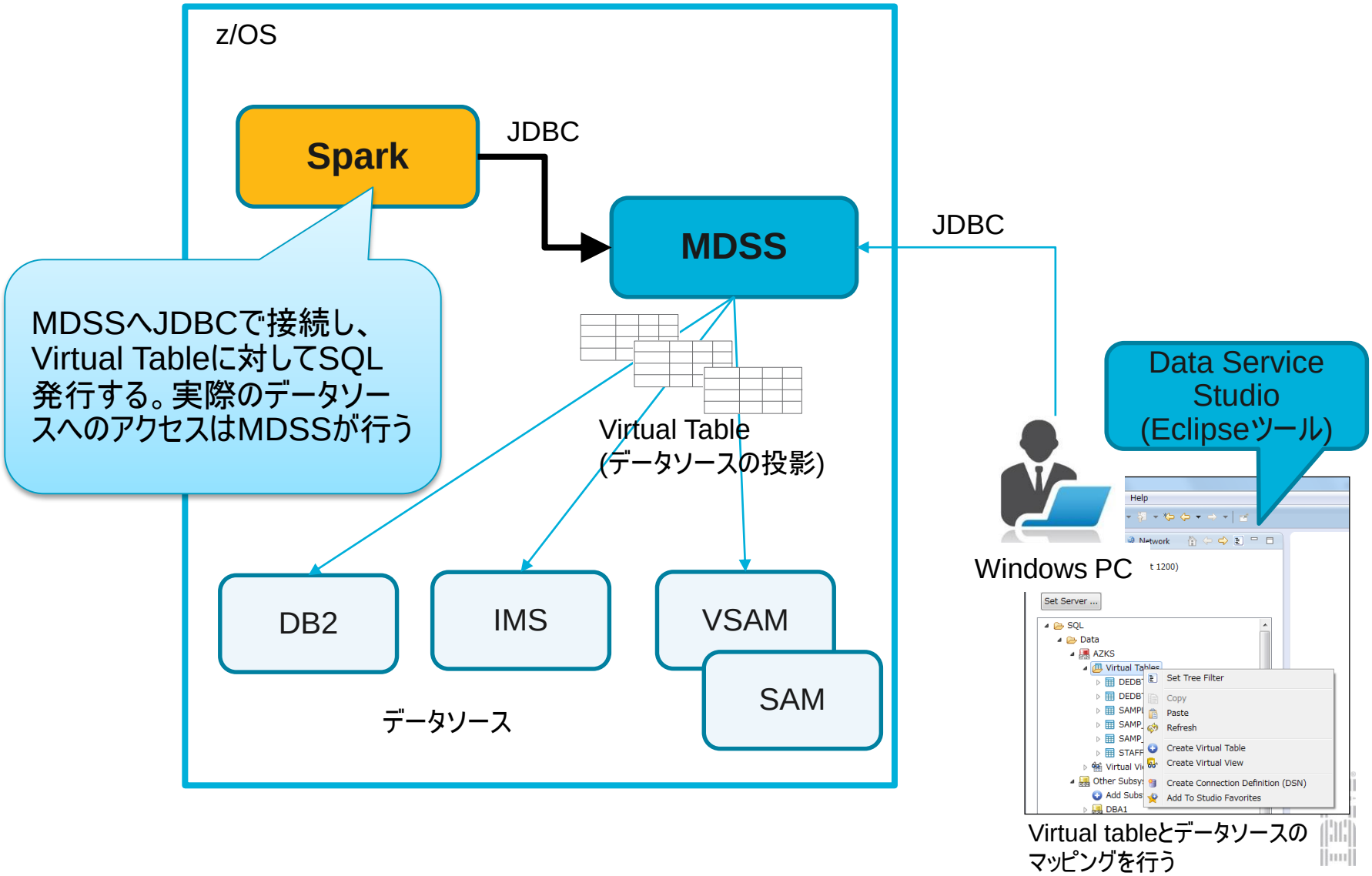
- USS上で稼働するApache Spark
  - 2016/3/25出荷開始。
  - Apache Spark単体（コミュニティ版）と、後述するMDSSとのセットになった製品版がある。

- 製品版には各種データソースへのアクセスが可能な Mainframe Data Service for Apache Spark z/OS (**MDSS**) を同梱
  - 表中のデータソースに対するANSI 92レベルのSQLアクセスを提供
    - データソースにアクセスするためのJDBCドライバーを含む
  - Spark でDB2、IMS、VSAM、SAMなどのメインフレーム・データや分散システムのデータを利用する事が可能

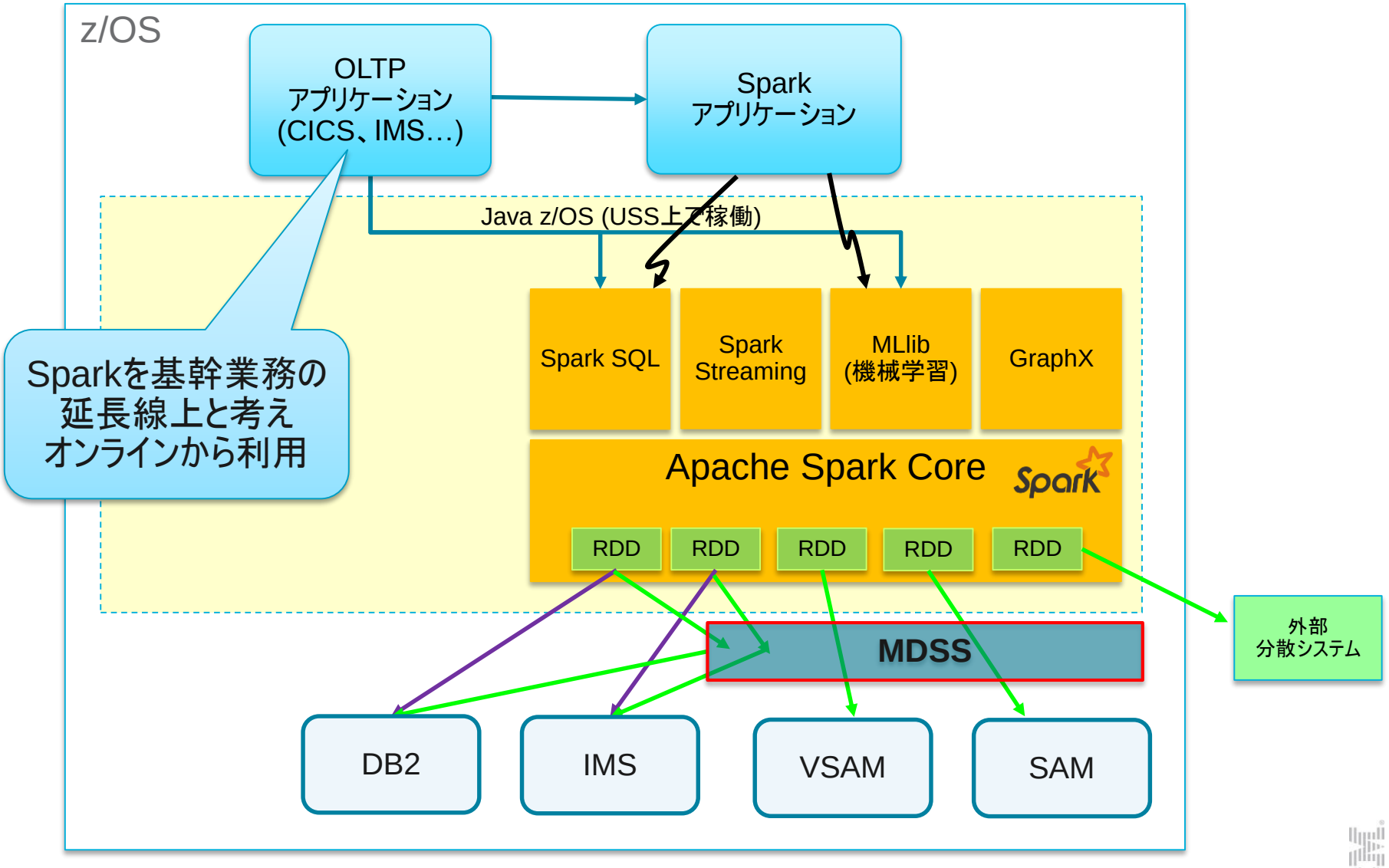
メインフレーム・データ	分散(Linux/Unix/Windows)上のデータベース
• IBM DB2 • IBM IMS • VSAMファイル • シーケンシャルファイル • Software AG Adabas	• IBM DB2 • Apache Derby • IBM Informix • Oracle • Microsoft SQL Server

・・・以降、Spark on z/OSと略記

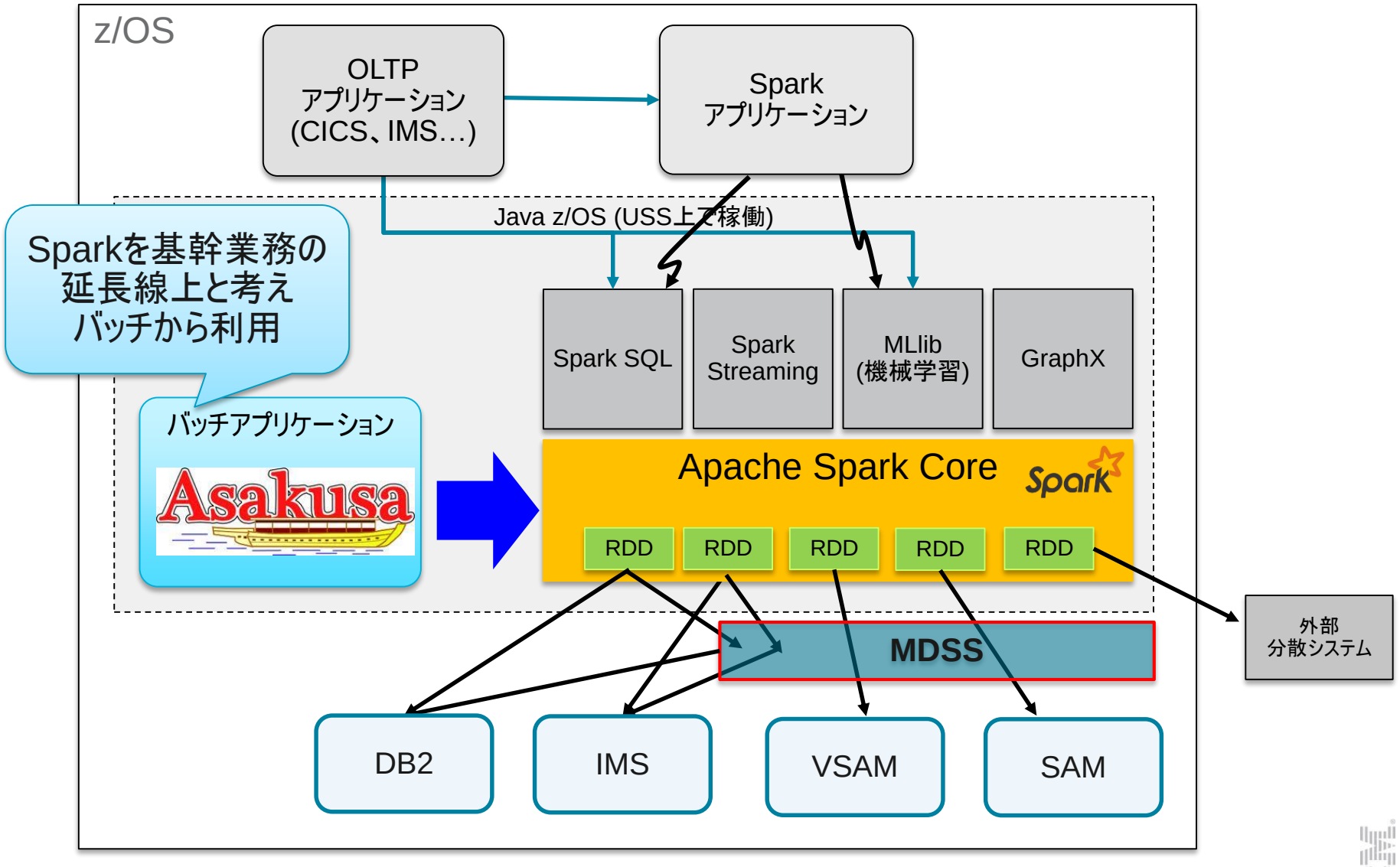
SparkとメインフレームデータをMDSS



Spark on z/OSと基幹オンラインの連携



Asakusa FrameworkによってSparkは基幹バッチにも適用できる（はず）



それ、高額になるのでは？

Q： CPUもメモリーも多く使いそうだけど・・・

A： z Systemsの戦略的に、高額にならないような仕掛けがあります。

zIIP* という特殊なCPUとメモリーについて、Spark on z/OSの用途向けのオファリングがあります（詳細はお問い合わせください）

Though the standard pricing for IBM System z Integrated Information Processors (zIIPs) and memory might not be competitive, as part of the IBM z/OS Platform for Apache Spark offering IBM is offering specially priced zIIPs and memory.

http://www.redbooks.ibm.com/redbooks/pdfs/sg248325.pdf

zIIPとは： Javaなどをオフロードすることができる、専用CPU。zIIPはソフトウェア課金の対象外なので、Spark実行時のコスト増大を抑えることができる。
なお、zIIPは通常のCPU(GCP)と比較して割安に価格設定されている。

その他、気になること

Q： Sparkはあることがわかったけれど、Hadoopは？

A： z/OSでは提供されていません。今回はAsakusa FrameworkをHadoopなしで実行します。

Q： では、HDFSに処理対象のデータは置かない？

A： 置きません。AsakusaのWindGate機能から、MDSS-JDBC経由のメインフレームデータ、もしくはUNIXファイルシステムのデータを使用します。

Q： データの文字コードは？

A： z/OSデータセットに関しては、EBCDICです。日本語の文字も扱うことができます。
Unixファイルに関しては、Asakusaで入出力するデータはUTF-8です。

Asakusaとのバージョン整合性は厳密に合わせること

Q： Spark on z/OSのバージョンは？

A： 1.5.2です(2016/11現在)。これに対応するAsakusaのバージョンは0.7.6ですので、今回はこれを使用しました。

開発環境			実行環境	
Asakusa Framework	Asakusa on Spark	Gradle	Spark	Hadoop
0.8.0	0.3.0	2.12	1.6.1	2.x
0.7.6	**0.2.2**	**2.8**	**1.5.2**	**2.x**

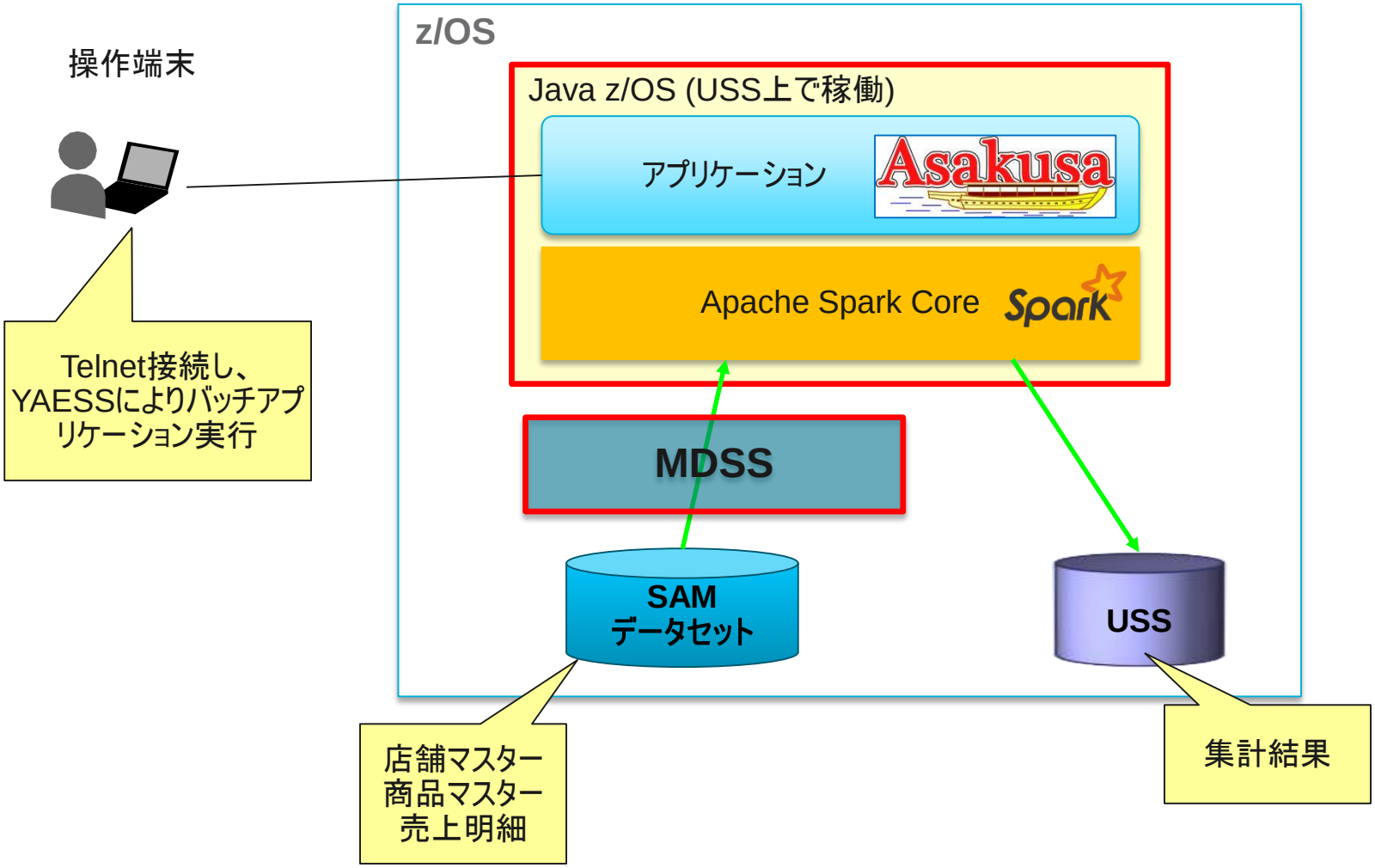
Spark 1.5.2に適合する
Asakusa 0.7.6を使用

アプリ開発・実行

- 結論： Asakusa on z/OSのアプリケーションが正常実行できました。
  - hadoopコマンドがないというメッセージは出るものの、実行自体には問題ない
- 開発： 売上データの振り分け・マスターデータとの結合・集計を行うサンプルアプリケーション（チュートリアル）をベースに、入力データをz/OSデータセットに変換・配置して実行
  - http://docs.asakusafw.com/basic-tutorial/
- 詳細： いくつかのノウハウがあります。Slideshare* にて公開中。
- 性能： 想定通り、zIIPでほとんどのCPU処理を行うことができ、実用的なスループットが得られました。（次ページに検証）

* http://ibm.biz/spark-on-zos-asakusa-framework

Asakusa on z/OS検証環境



データソースをホストデータとして準備

- チュートリアルのシナリオにて、売り上げ情報のレコード数を増幅してアプリケーションを実行
- 入力データをz/OSデータセットとして準備し、MDSS経由で読み取り

- 商品マスタ
  AZK.ASAKUSA.SAMPLE.SEQFILES(ITEMINFO)

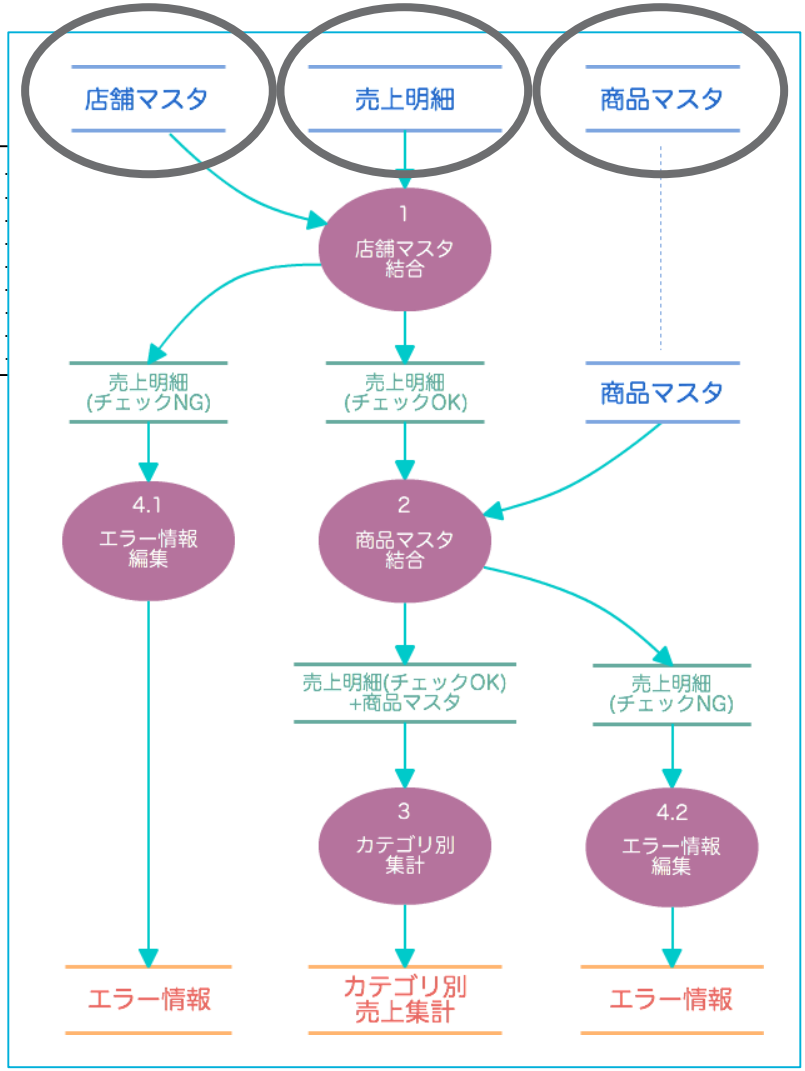
```
922010001000Milk Chocolate M          110Snack    1600ChocolateSnack  ..
922010001001PREMIUM Chocolate         110Snack    1600ChocolateSnack  ..
922010001002Almond Crunch mini        110Snack    1600ChocolateSnack  ..
922020002000CupNoodle Shoyu           130Food     1401CupNoodle       ..
922020002001CupNoodle Salt            130Food     1401CupNoodle       ..
922020002002CupNoodle Curry           130Food     1401CupNoodle       ..
922030003000CupIce YubariMelon        110Snack    1300IceCream        ..
922030003001Maccha Sundae             110Snack    1300IceCream        ..
922030003002RockIce Ichigo            110Snack    1300IceCream        ..
```

- 店舗マスタ
  AZK.ASAKUSA.SAMPLE.SEQFILES(STORINFO)

```
0000SuperMakuhari-HQ
0001SuperMakuhari-Shinagawa
0002SuperMakuhari-Shibuya
0055SuperMakuhari-Heiwajima
0004SuperMakuhari-Nishiarai
```

- 売上明細
  AZK.ASAKUSA.SAMPLE.SEQFILE4

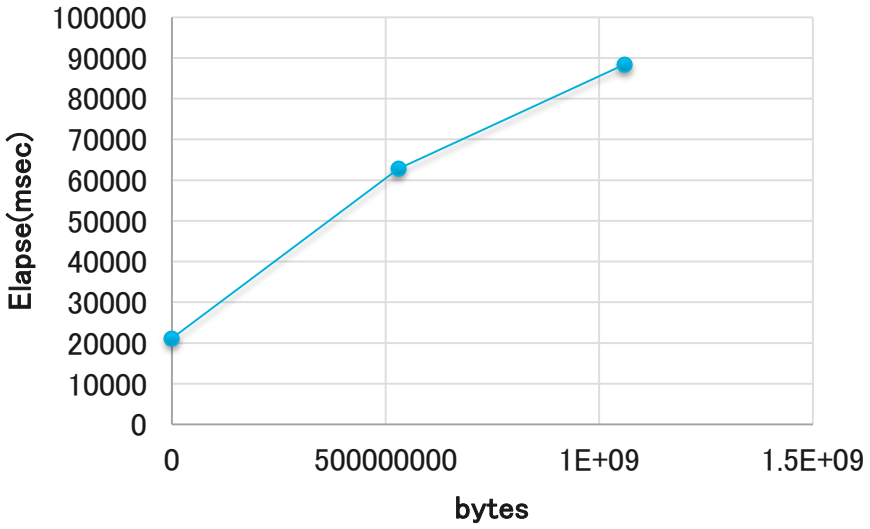
```
2011-04-01 10:30:0000014922010001000.....ッ...ッDUMMY
2011-04-01 10:31:0000014922020002000.....ｲ...DDUMMY
2011-04-01 10:32:0000014922030003000.....ｴ....DUMMY
2011-04-01 10:30:0000014922010001000.....ッ...ッDUMMY
2011-04-01 10:31:0000014922020002000.....ｲ...DDUMMY
2011-04-01 10:32:0000014922030003000.....ｴ....DUMMY
...
```

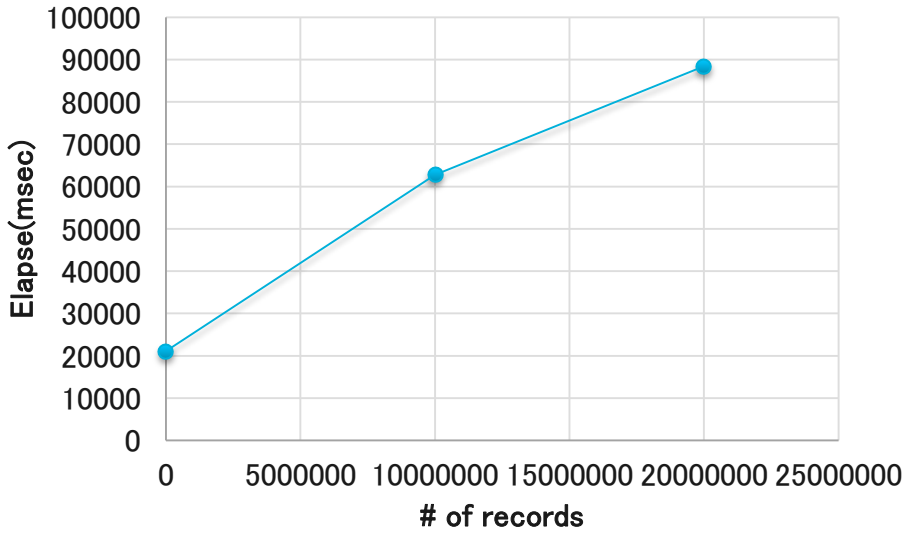


結果：処理スループットは現実的

- YAESS実行ログに出力される処理時間で比較

ケース#	レコード数	データサイズ(bytes)	Elapse(msec)
1	5	265	21,129
2	10,000,002	530,000,106	62,779
3	20,000,004	1,060,000,212	88,430



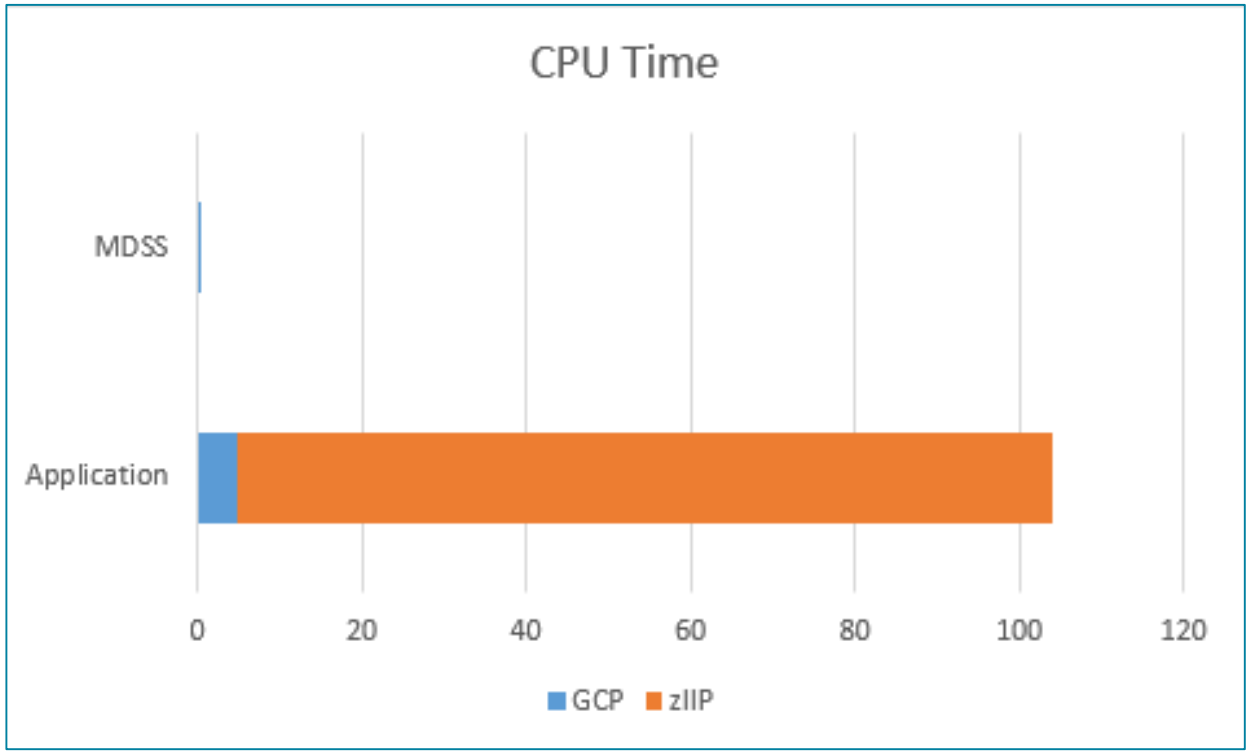


結果：CPU使用は、大部分をzIIPに逃がせられた

- 前頁のケース3（レコード数20,000,004件）でのCPU時間
  - RMF(Workload Activity Report) より

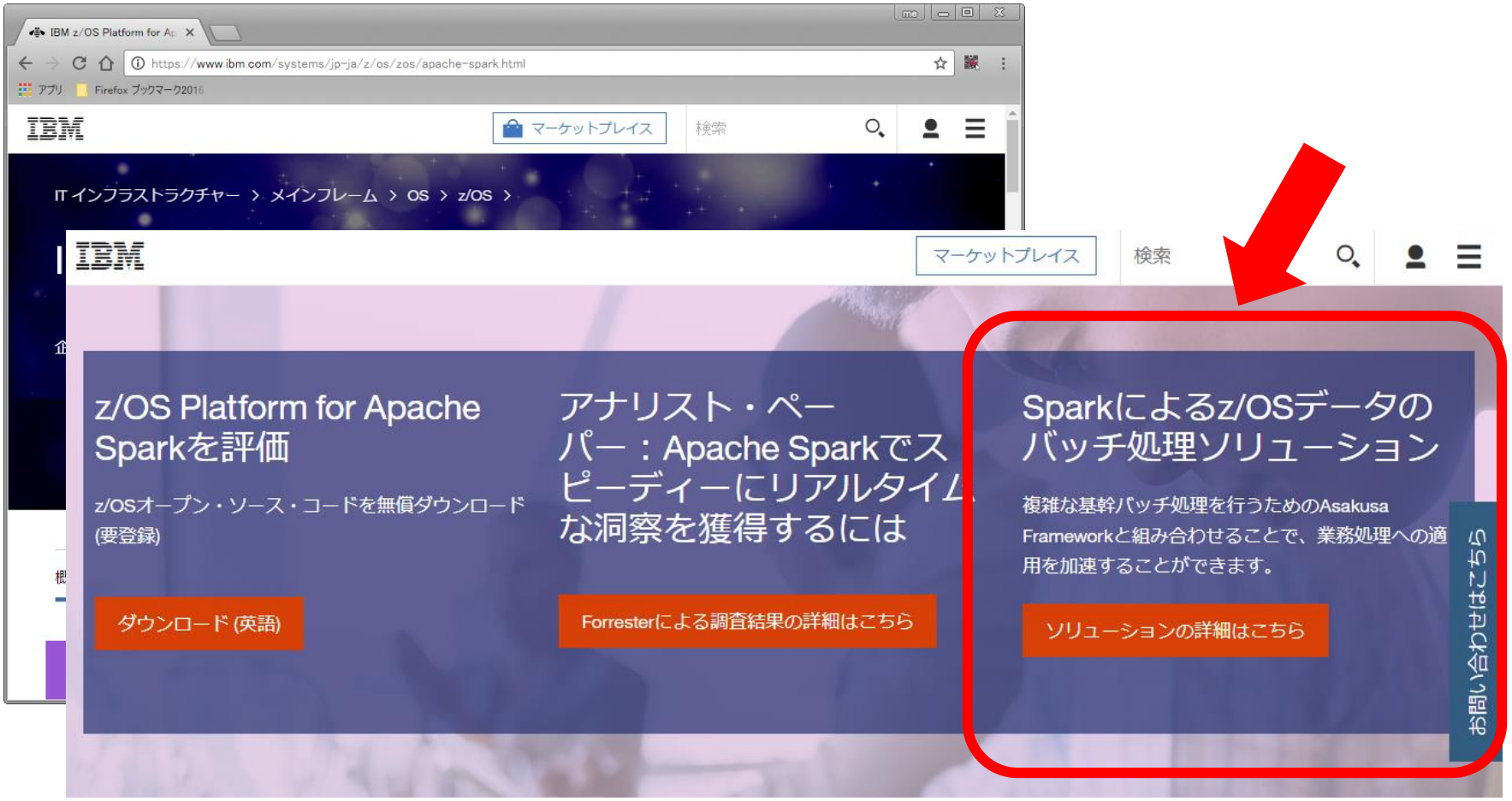
	Application (Spark)	MDSS
GCP(汎用CPU)	4.847	0.035
zIIP	99.178	0

単位：秒



日本IBM Webサイト

z/OS Spark　検索



https://www.ibm.com/systems/jp-ja/z/os/zos/apache-spark.html

まとめ

- Asakusa Frameworkによるバッチアプリケーションをメインフレーム（z/OS）上で実行することができた。
- セキュリティ、スキル、コストの観点で、バッチアプリケーションを進化できるソリューションが提案できそう。
- IBMはSpark on z/OSに注力しており、Asakusa Frameworkとのコラボも今後進めていく予定。