

Java VMへの処方箋

～ 先進のメモリ管理技術とは～

13-B-4

中島 恵

株式会社 日立製作所 ソフトウェア事業部
第2AP基盤ソフト設計部 担当部長

Contents

1. Javaのメモリ管理の課題とは
2. GC (ガベージコレクション)の仕組み
3. 「FullGCレス」を実現する最新技術
4. 「FullGCレス」の効果 (デモンストレーション)
5. メモリトラブルを起こさないためには

Contents

1. Javaのメモリ管理の課題とは
2. GC (ガベージコレクション) の仕組み
3. 「FullGCレス」を実現する最新技術
4. 「FullGCレス」の効果 (デモンストレーション)
5. メモリトラブルを起こさないためには

Webシステムの安定性・堅牢性を高めるためには

Webシステムは、アプリケーションサーバおよびJavaによる開発生産性の向上によって、容易に構築が可能となった

→業務量ピーク時など、**性能上のトラブル**が後を絶たない

●システムの性能劣化につながる要因

1. CPUネック

2. メモリネック

JavaVMのメモリ管理機能(GC)に起因

3. バックエンドシステムによるネック

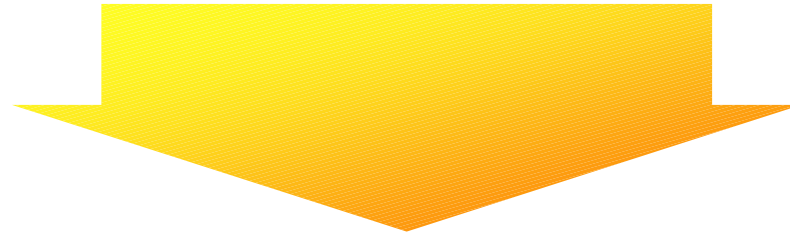
Webシステムの扱うデータサイズの肥大化

Webシステムが参照・更新するデータのサイズが
どんどん大きくなってきている

→ 64ビット化で対処はできるが、メモリ領域サイズの
増大に伴ってGCでの停止時間が長時間し、
システム性能劣化に陥る

JavaVMのメモリ管理機能(GC)に起因

JavaVMのメモリ管理機能(GC)に起因した問題が山積



JavaVMのメモリ管理機能(GC)として、これらの問題を解消する「**次の一手**」が、課題解決のためには必要

→ JavaVMを大きく改良することによって、課題を解決するアプローチが見出せるのでは？

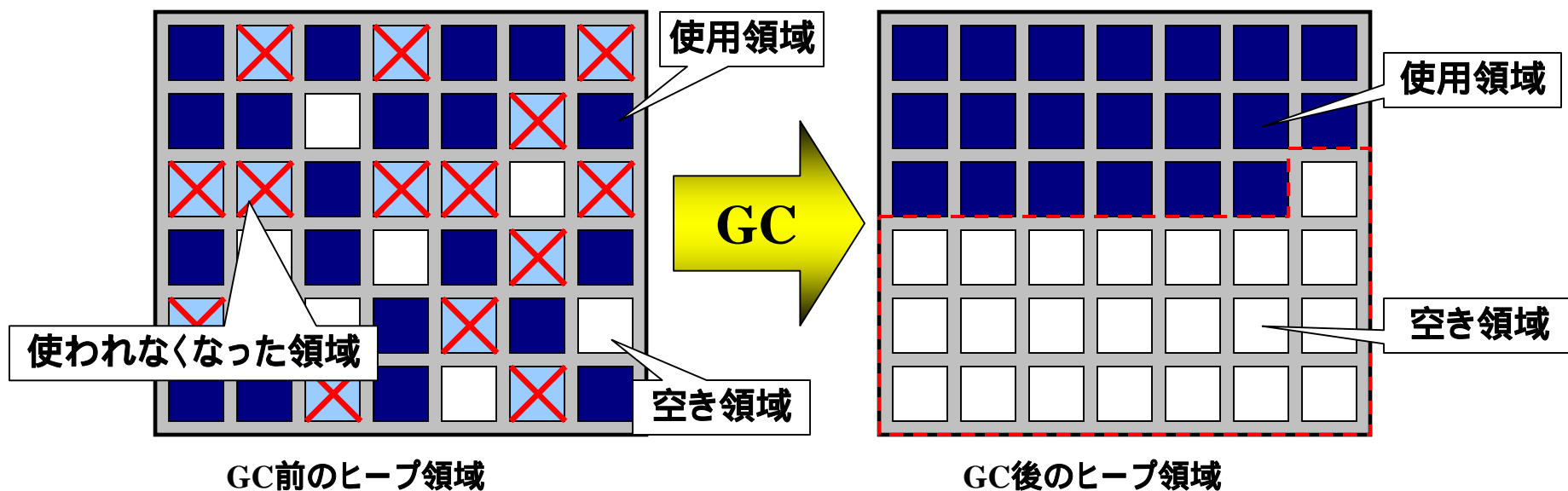
Contents

1. Javaのメモリ管理の課題とは
2. GC (ガベージコレクション)の仕組み
3. 「FullGCレス」を実現する最新技術
4. 「FullGCレス」の効果 (デモンストレーション)
5. メモリトラブルを起こさないためには

2. ガベージコレクションの仕組み

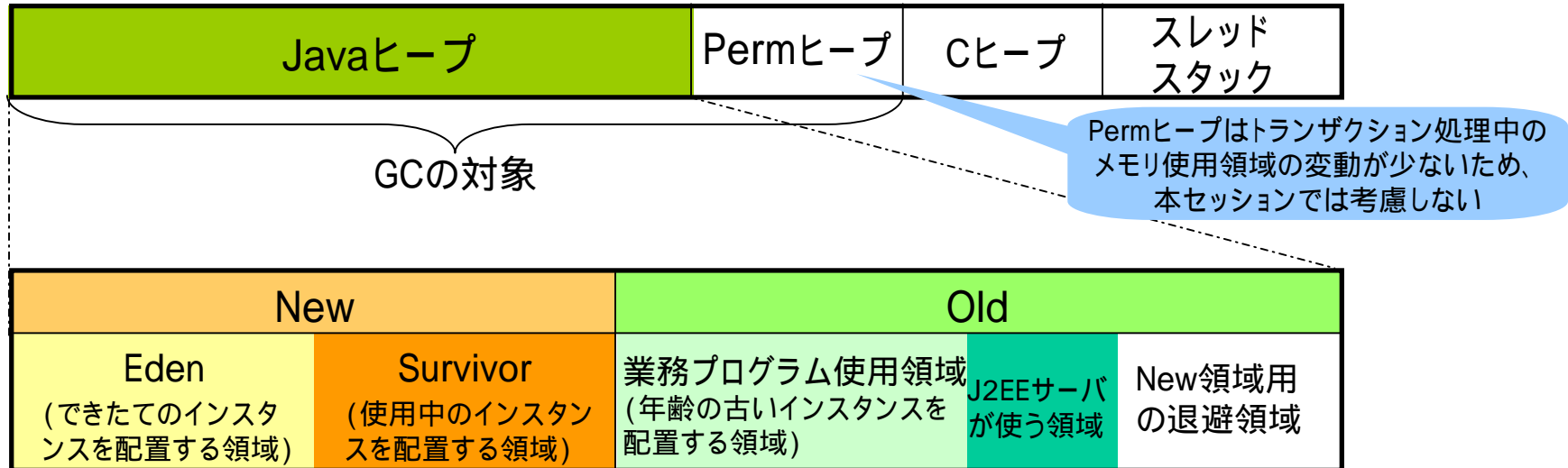
● ガベージコレクション (GC) とは...

JavaVMが管理するメモリ領域中の使用済みのメモリ領域を破棄し、
空き領域を作ること



2.ガベージコレクションの仕組み -Javaヒープとは-

●Javaプロセスのメモリの内訳



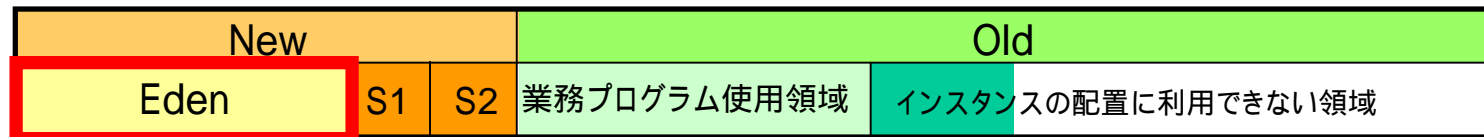
- Javaヒープ Javaアプリケーションが使用するメモリ領域
- Permヒープ クラスなどのメタ情報を格納する領域
- Cヒープ JavaVMやCプログラムが使用するヒープ領域
- スレッドスタック スレッドごとに保持するスタック領域

2. ガベージコレクションの仕組み -CopyGCとFullGC-

● GCには2種類ある

CopyGC...New領域を対象に、使用済みのインスタンスを全て削除する

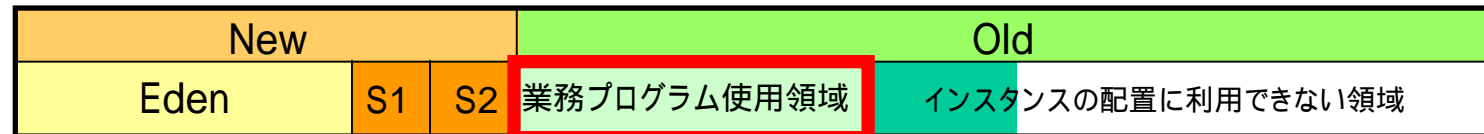
使用中のインスタンスは隣の領域へ移動する



Edenの領域がいっぱいになるとCopyGCが起こる

使用中のものは隣の領域へ

FullGC...全ての領域を対象に、使用済みのインスタンスを全て削除する



Oldの業務プログラム使用領域がいっぱいになるとFullGCが起こる

2. ガベージコレクションの仕組み -GCの問題点-

● 2つのGCの比較

| | 範囲 | 発生タイミング | 実行にかかる時間 |
|--------|-------|---------------------------|-------------|
| CopyGC | New領域 | Eden領域がいっぱいになった時 | 0.01 ~ 0.7秒 |
| FullGC | 全領域 | Oldの業務プログラム使用領域がいっぱいになった時 | 1秒 ~ 数十秒 |

・FullGCは特に時間がかかる

・Javaヒープのサイズにより実行にかかる時間は増加する

GCはメモリの空き容量を確保するために必要な処理

しかし、GCが発生すると、GC実行中は業務処理が停止する

ノーマルGC

GCのアルゴリズムを
変えてみる？
対処して見る？

GCアルゴリズムは
複雑になるほど
コントロール不可
となる

コンカレントGC

2. ガベージコレクションの仕組み -GCの問題点-

● 2つのGCの比較

| | 範囲 | 発生タイミング | 実行にかかる時間 |
|--------|-------|---------------------------|-------------|
| CopyGC | New領域 | Eden領域がいっぱいになった時 | 0.01 ~ 0.7秒 |
| FullGC | 全領域 | Oldの業務プログラム使用領域がいっぱいになった時 | 1秒 ~ 数十秒 |

・FullGCは特に時間がかかる

・Javaヒープのサイズにより実行にかかる時間は増加する

GCはメモリの空き容量を確保するために必要な処理

しかし、GCが発生すると、GC実行中は業務処理が停止する



FullGCの発生を抑止することを目的に

発想の転換を行った

→メモリ領域を1つの枠組みで管理することに

そもそも無理があるのでは？

2. ガベージコレクションの仕組み - 明示管理ヒープ方式の位置付け -

ヒープ管理方式

世代別ヒープ管理方式
(世代別GC方式)

+

明示管理ヒープ方式

NEW

特許出願済

GC方式

ノーマルGC

パラレルGC

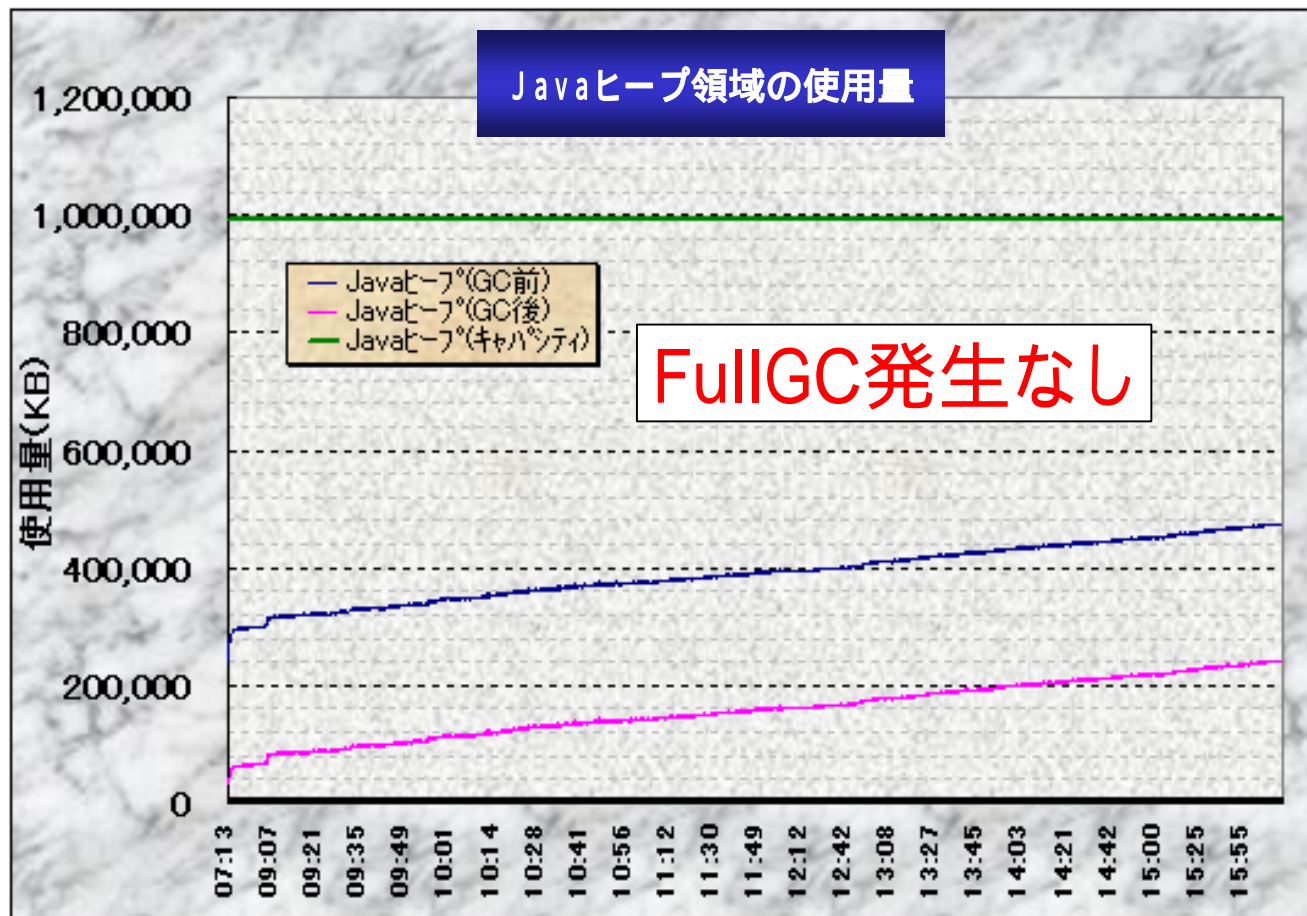
コンカレントGC

⋮

GCアルゴリズムの変更ではFullGC発生は回避不可のため、
ヒープ管理方式として **明示管理ヒープ方式**
(Explicit Heap方式: 略称 **Eヒープ方式**) を開発!

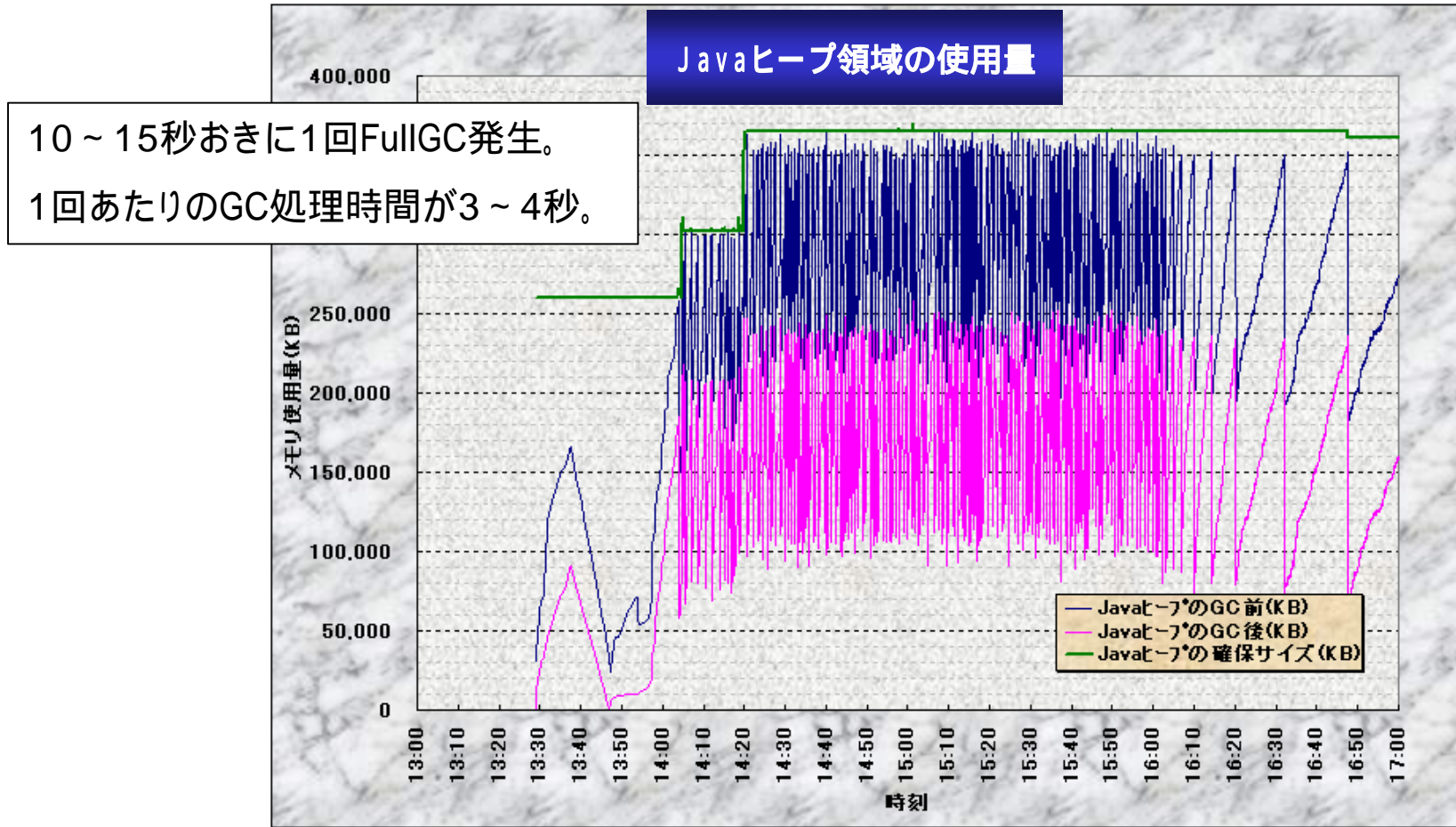
2. ガベージコレクションの仕組み

-安定時のメモリ使用状況(グラフ)-



安定時のグラフ (非月末)

2.ガベージコレクションの仕組み -GC多発時のメモリ使用状況(グラフ)-



GC多発時のグラフ(月末)

Contents

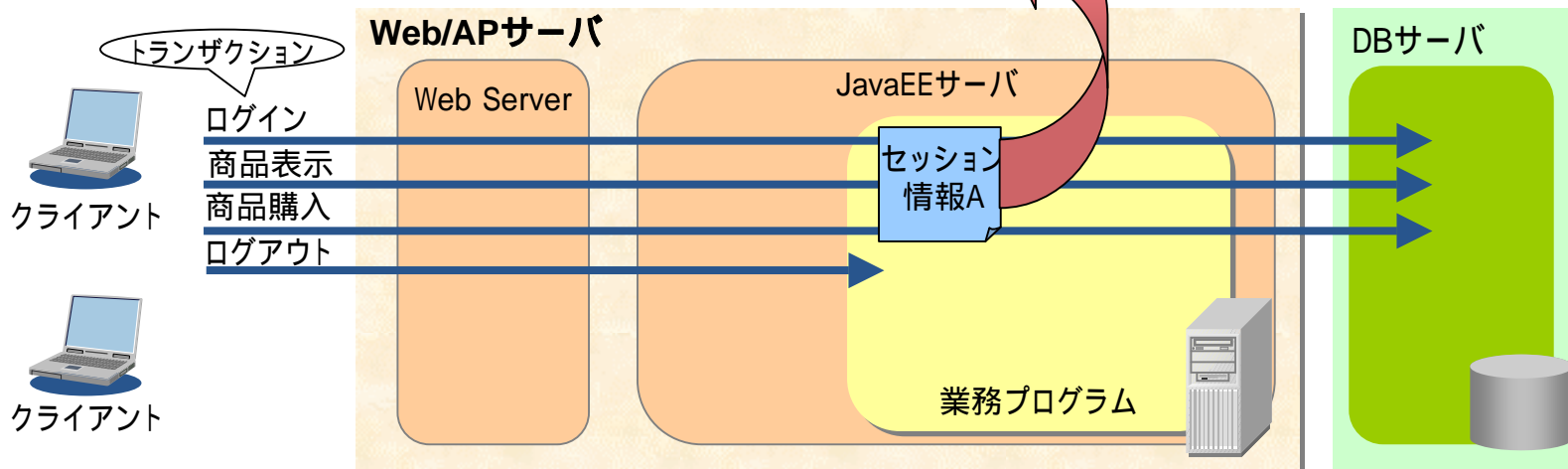
1. Javaのメモリ管理の課題とは
2. GC (ガベージコレクション)の仕組み
3. 「FullGCレス」を実現する最新技術
4. 「FullGCレス」の効果 (デモンストレーション)
5. メモリトラブルを起こさないためには

3. 「FullGCレス」を実現する最新技術

-ヒープ領域ごとの用途-

- New領域およびOld領域に格納されるインスタンスの違いに着目

| | New | | Old | | |
|-------------|-------------------------------------|----------|------------------------|----------------|-------------|
| Javaヒープ構成 | Eden | Survivor | 業務プログラム使用領域 | JavaEEサーバが使う領域 | New領域用の退避領域 |
| 格納されるインスタンス | 短命なインスタンス (トランザクション処理で使用するメモリなど) | | 長命なインスタンス (セッション情報) | JavaEEサーバが使う領域 | New領域用の退避領域 |

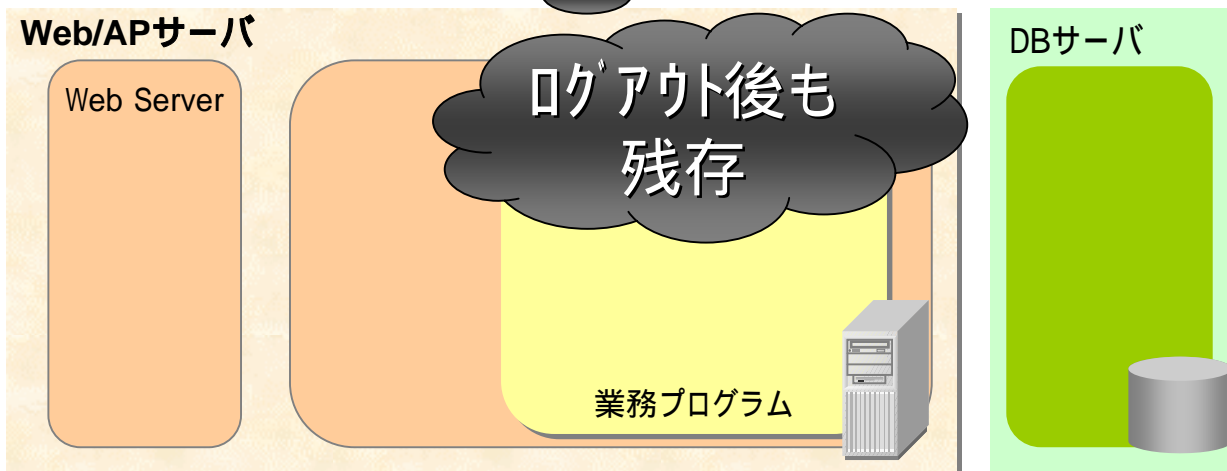


3. 「FullGCレス」を実現する最新技術

-ヒープ領域ごとの用途-

- New領域およびOld領域に格納されるインスタンスの違いに着目

| | | New | | Old | | |
|-----------------|--|---|----------|------------------------|--------------------|-----------------|
| Javaヒープ 構成 | | Eden | Survivor | 業務プログラム使用領域 | JavaEEサーバ が使う領域 | New領域用の 退避領域 |
| 格納される インスタンス | | 短命なインスタンス (トランザクション処理で 使用するメモリなど) | | 長命なインスタンス (セッション情報) | JavaEEサーバ が使う領域 | New領域用の 退避領域 |

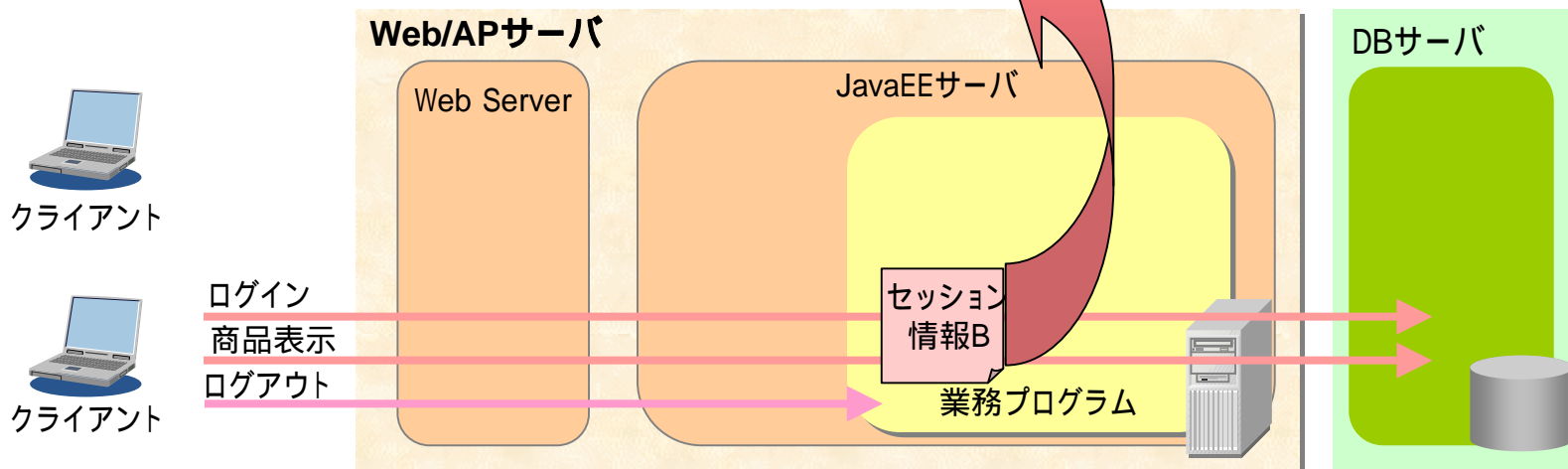


3. 「FullGCレス」を実現する最新技術

-ヒープ領域ごとの用途-

- New領域およびOld領域に格納されるインスタンスの違いに着目

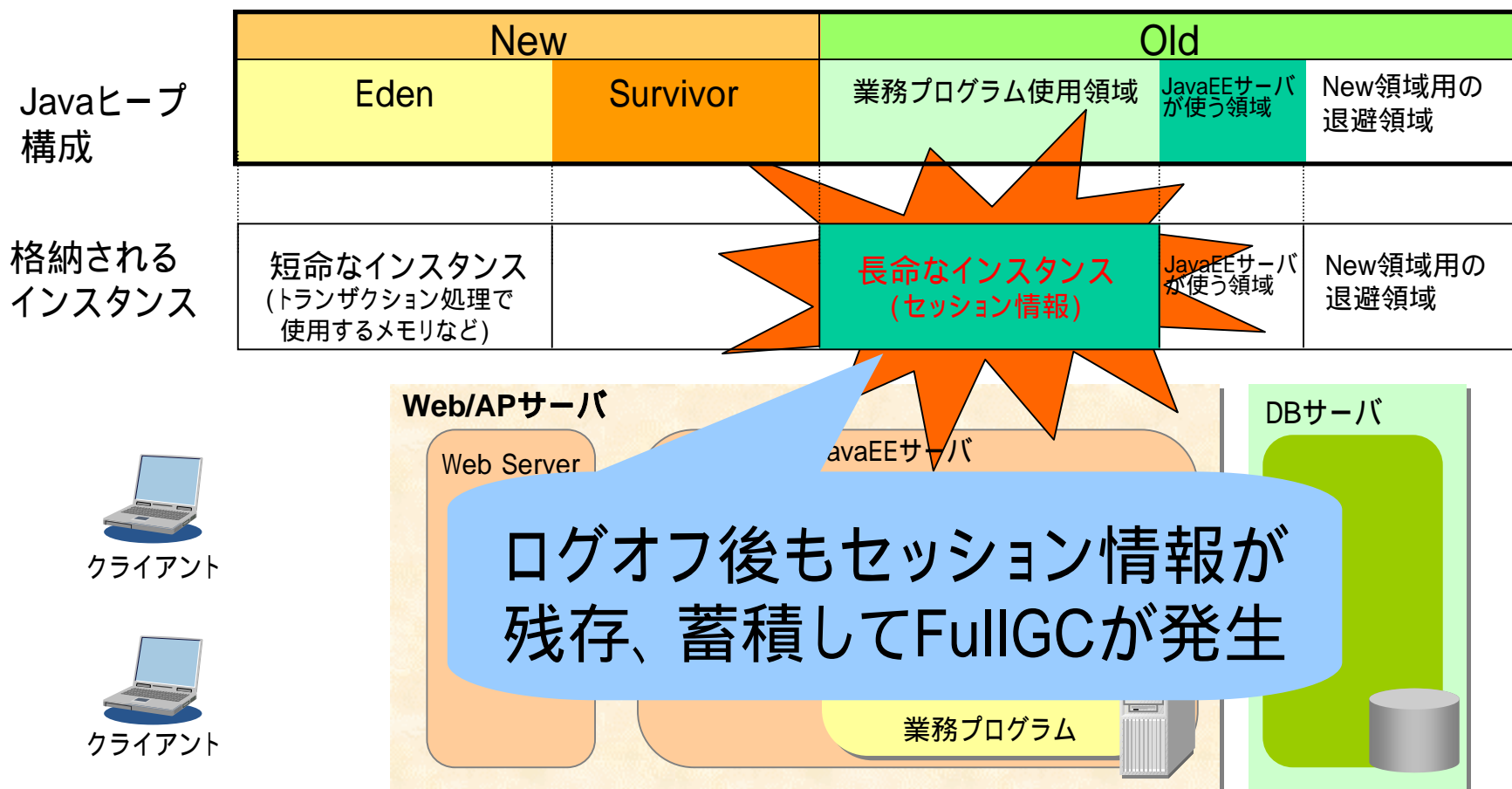
| | | New | | Old | | |
|-------------|--|-------------------------------------|----------|------------------------|----------------|-------------|
| Javaヒープ構成 | | Eden | Survivor | 業務プログラム使用領域 | JavaEEサーバが使う領域 | New領域用の退避領域 |
| 格納されるインスタンス | | 短命なインスタンス (トランザクション処理で使用するメモリなど) | | 長命なインスタンス (セッション情報) | JavaEEサーバが使う領域 | New領域用の退避領域 |



3. 「FullGCレス」を実現する最新技術

-ヒープ領域ごとの用途-

- New領域およびOld領域に格納されるインスタンスの違いに着目



3. 「Full GCレス」を実現する最新技術

- Full GCレスでStop The Worldを解消！ -

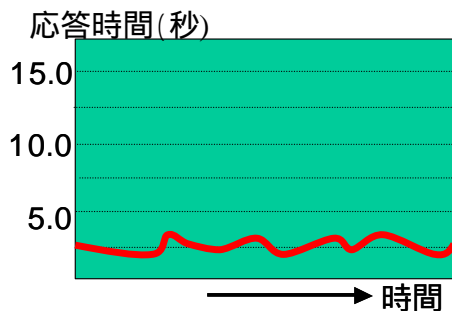
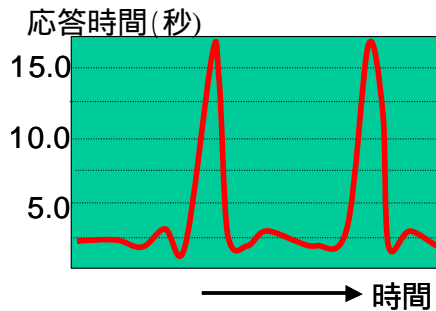


Full GCによるオンライン業務の一時的な停止は、何とかならないものだろうか…

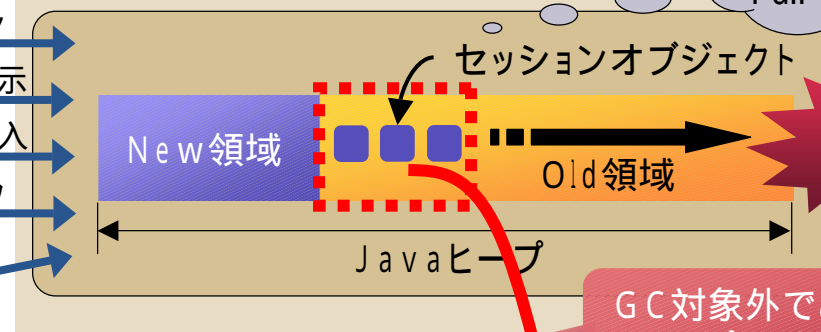
業界初！(*)
Full GCレスを実現

ポイント

- Webアプリケーションの変更無く、Full GCの発生を抑止。Full GCによるオンライン業務の一時停止を解消します。



従来方式

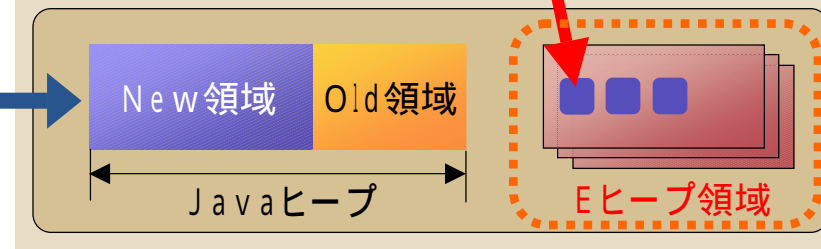


ログオフ後もセッション情報が残存。Full GCで解放。

GC対象外であるEヒープ領域に配置しFull GC発生を抑止

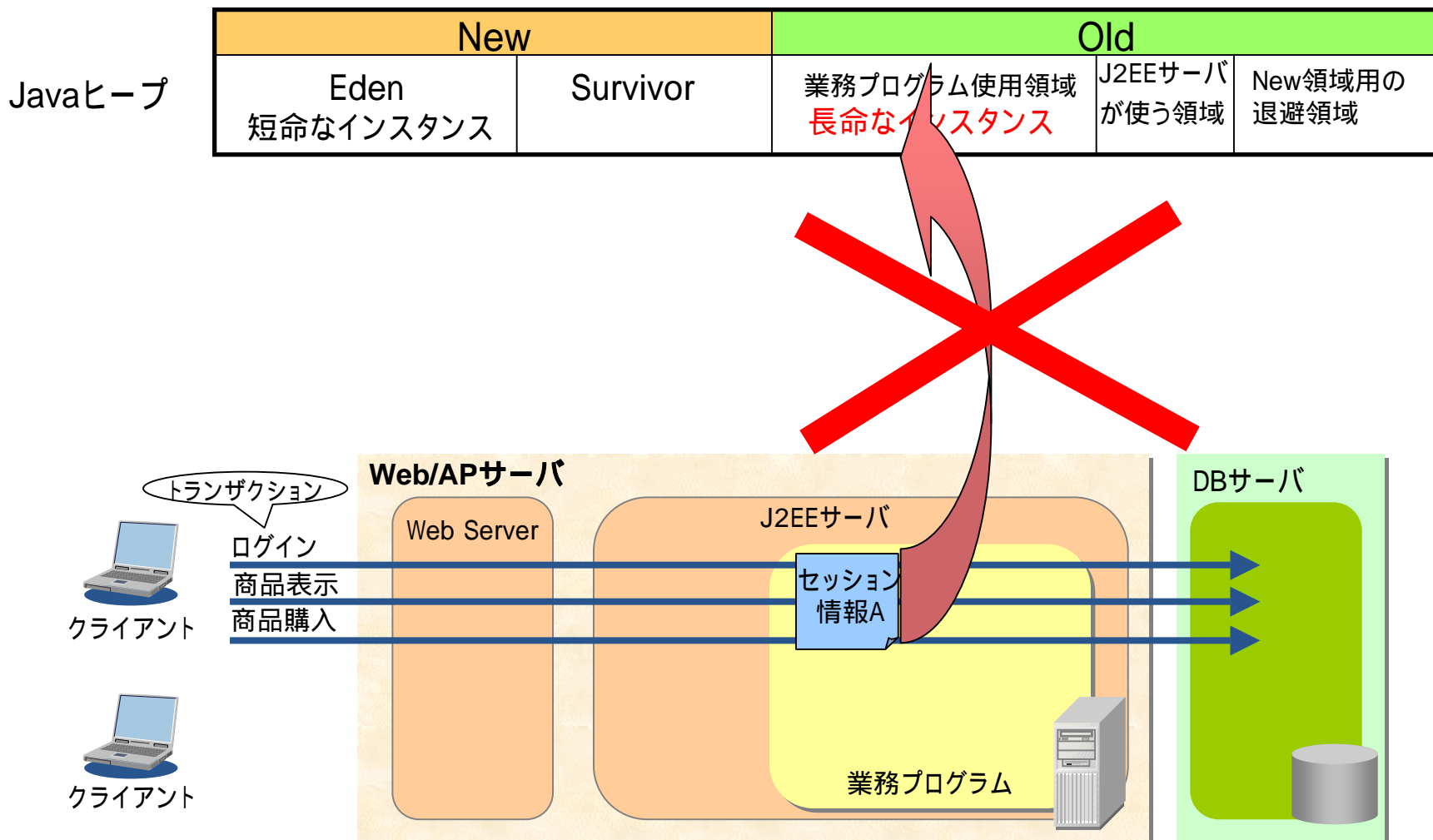


新方式



3. 「FullGCレス」を実現する最新技術

-FullGCレス機能適用時の動き-



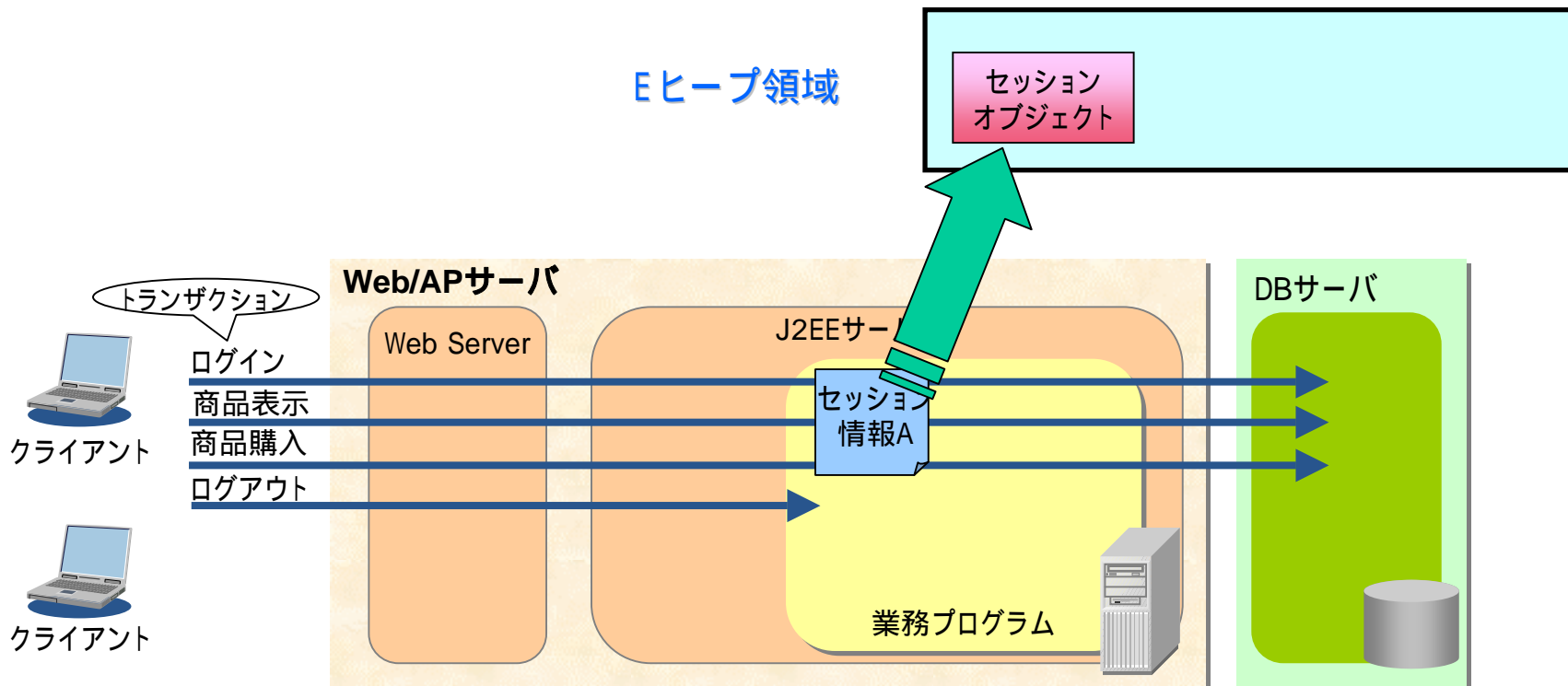
3. 「FullGCレス」を実現する最新技術

-FullGCレス機能適用時の動き-

| New | | Old | |
|-------------------|----------|--------------------------|-----------------------------|
| Eden 短命なインスタンス | Survivor | 業務プログラム使用領域 長命なインスタンス | J2EEサーバが使う領域 New領域用の退避領域 |

Eヒープ領域

セッション
オブジェクト



3. 「FullGCレス」を実現する最新技術

-FullGCレス機能適用時の動き-

Javaヒープ

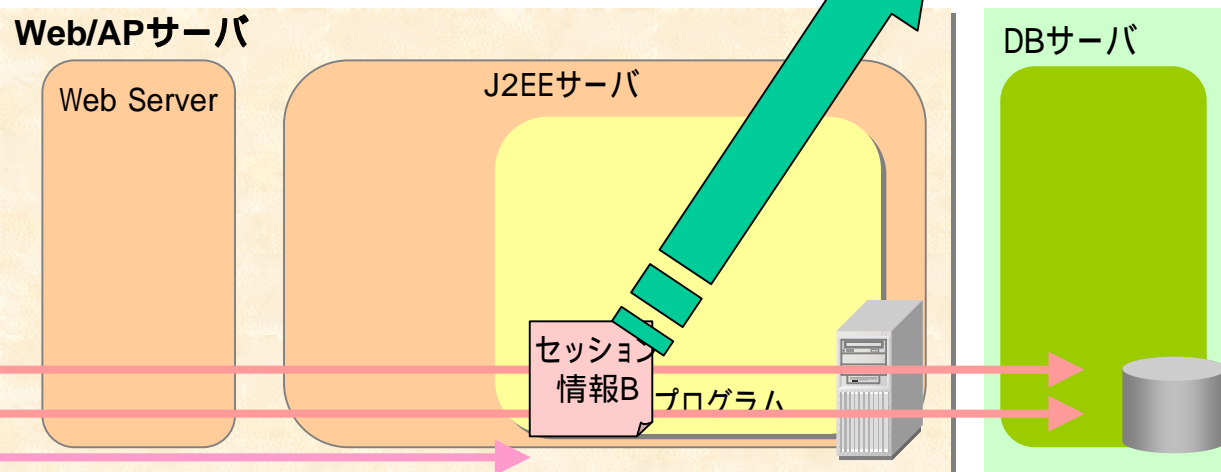
| New | | Old | | |
|-------------------|----------|--------------------------|------------------|-----------------|
| Eden 短命なインスタンス | Survivor | 業務プログラム使用領域 長命なインスタンス | J2EEサーバ が使う領域 | New領域用の 退避領域 |

Eヒープ領域

セッション
オブジェクト



ログイン
商品表示
ログアウト



3. 「FullGCレス」を実現する最新技術

-FullGCレス機能適用時の動き-

Javaヒープ

| New | | Old |
|-------------------|----------|---|
| Eden 短命なインスタンス | Survivor | 業務プログラム使用領域 長命なインスタンス J2EEサーバが使う領域 New領域用の退避領域 |

セッション情報が蓄積せず
FullGCレス！



3. 「FullGCレス」を実現する最新技術

-FullGCレス機能の仕組みとは-

業務プログラム

(1)セッションの生成

```
HttpSession session = request.getSession();
```

(2)オブジェクトの生成およびセッションへのオブジェクトの格納

```
Object obj1 = new UserObject1();  
session.setAttribute(Key1, obj1);
```

(3)オブジェクトの生成およびセッションへのオブジェクトの格納

```
Object obj2 = new UserObject2();  
session.setAttribute(Key2, obj2);
```

(4)セッションの終了

```
session.invalidate();
```

Cosminexus

セッションの割り当て

明示管理ヒープ
領域確保

メモリ領域

Eヒープ



3. 「FullGCレス」を実現する最新技術

-FullGCレス機能の仕組みとは-

業務プログラム

(1)セッションの生成

```
HttpSession session = request.getSession();
```

(2)オブジェクトの生成およびセッションへのオブジェクトの格納

```
Object obj1 = new UserObject1();
session.setAttribute(Key1, obj1);
```

(3)オブジェクトの生成およびセッションへのオブジェクトの格納

```
Object obj2 = new UserObject2();
session.setAttribute(Key2, obj2);
```

(4)セッションの終了

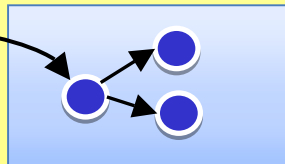
```
session.invalidate();
```

Cosminexus

メモリ領域

Eヒープ

New領域



3. 「FullGCレス」を実現する最新技術

-FullGCレス機能の仕組みとは-

業務プログラム

(1)セッションの生成

```
HttpSession session = request.getSession();
```

(2)オブジェクトの生成およびセッションへのオブジェクトの格納

```
Object obj1 = new UserObject1();
session.setAttribute(Key1, obj1);
```

(3)オブジェクトの生成およびセッションへのオブジェクトの格納

```
Object obj2 = new UserObject2();
session.setAttribute(Key2, obj2);
```

(4)セッションの終了

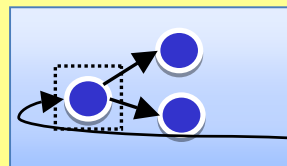
```
session.invalidate();
```

Cosminexus

Obj1とセッションを関連付け

メモリ領域

New領域



Eヒープ



3. 「FullGCレス」を実現する最新技術

-FullGCレス機能の仕組みとは-

業務プログラム

(1)セッションの生成

```
HttpSession session = request.getSession();
```

(2)オブジェクトの生成およびセッションへの
オブジェクトの格納

```
Object obj1 = new UserObject1();  
session.setAttribute(Key1, obj1);
```

(3)オブジェクトの生成およびセッションへの
オブジェクトの格納

```
Object obj2 = new UserObject2();  
session.setAttribute(Key2, obj2);
```

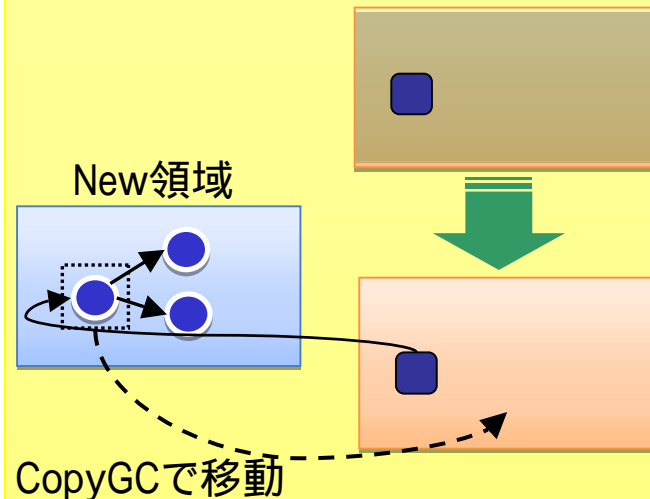
(4)セッションの終了

```
session.invalidate();
```

Cosminexus

メモリ領域

Eヒープ



3. 「FullGCレス」を実現する最新技術

-FullGCレス機能の仕組みとは-

業務プログラム

(1)セッションの生成

```
HttpSession session = request.getSession();
```

(2)オブジェクトの生成およびセッションへのオブジェクトの格納

```
Object obj1 = new UserObject1();
session.setAttribute(Key1, obj1);
```

(3)オブジェクトの生成およびセッションへのオブジェクトの格納

```
Object obj2 = new UserObject2();
session.setAttribute(Key2, obj2);
```

(4)セッションの終了

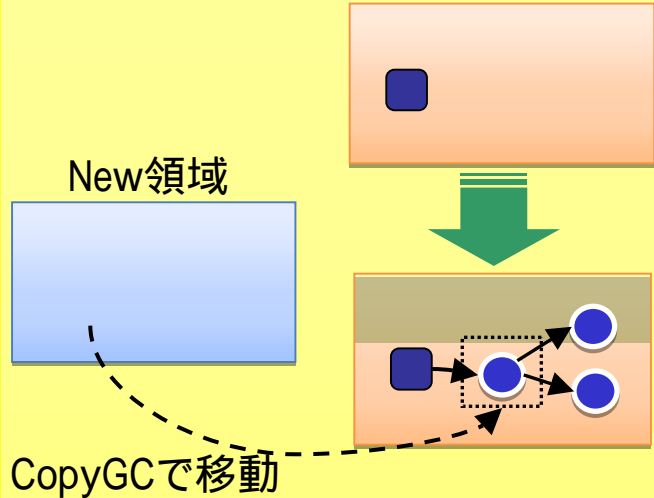
```
session.invalidate();
```

Cosminexus

メモリ領域

Eヒープ

New領域



3. 「FullGCレス」を実現する最新技術

-FullGCレス機能の仕組みとは-

業務プログラム

(1)セッションの生成

```
HttpSession session = request.getSession();
```

(2)オブジェクトの生成およびセッションへのオブジェクトの格納

```
Object obj1 = new UserObject1();  
session.setAttribute(Key1, obj1);
```

(3)オブジェクトの生成およびセッションへのオブジェクトの格納

```
Object obj2 = new UserObject2();  
session.setAttribute(Key2, obj2);
```

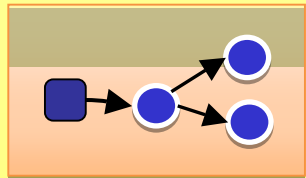
(4)セッションの終了

```
session.invalidate();
```

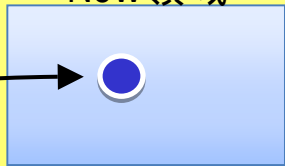
Cosminexus

メモリ領域

Eヒープ



New領域



3. 「FullGCレス」を実現する最新技術

-FullGCレス機能の仕組みとは-

業務プログラム

(1)セッションの生成

```
HttpSession session = request.getSession();
```

(2)オブジェクトの生成およびセッションへのオブジェクトの格納

```
Object obj1 = new UserObject1();  
session.setAttribute(Key1, obj1);
```

(3)オブジェクトの生成およびセッションへのオブジェクトの格納

```
Object obj2 = new UserObject2();  
session.setAttribute(Key2, obj2);
```

(4)セッションの終了

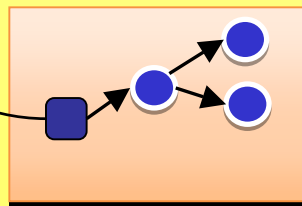
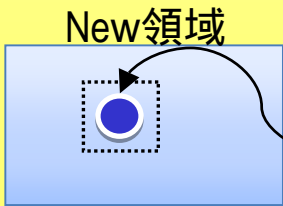
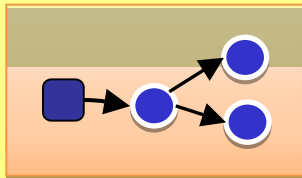
```
session.invalidate();
```

Cosminexus

Obj2とセッションを関連付け

メモリ領域

Eヒープ



3. 「FullGCレス」を実現する最新技術

-FullGCレス機能の仕組みとは-

業務プログラム

(1)セッションの生成

```
HttpSession session = request.getSession();
```

(2)オブジェクトの生成およびセッションへのオブジェクトの格納

```
Object obj1 = new UserObject1();
session.setAttribute(Key1, obj1);
```

(3)オブジェクトの生成およびセッションへのオブジェクトの格納

```
Object obj2 = new UserObject2();
session.setAttribute(Key2, obj2);
```

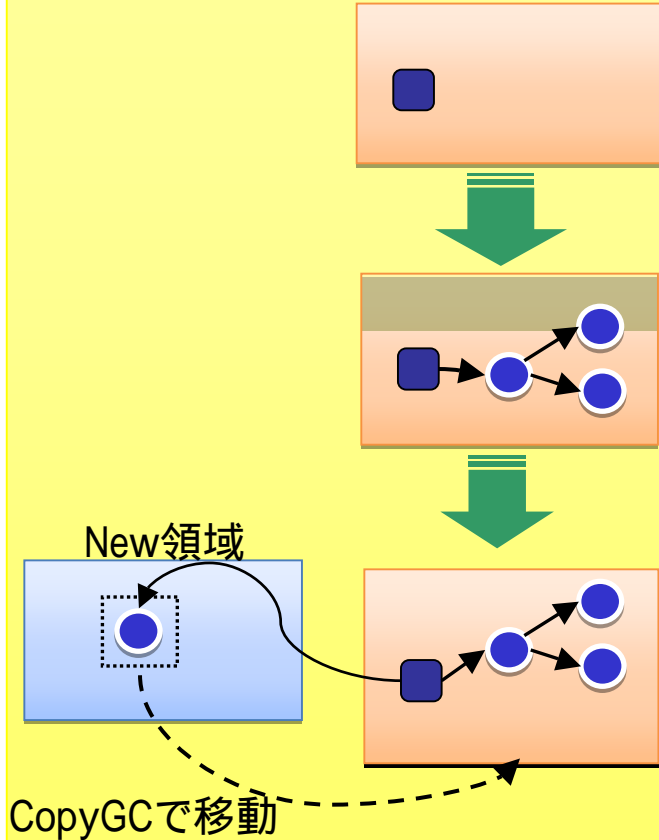
(4)セッションの終了

```
session.invalidate();
```

Cosminexus

メモリ領域

Eヒープ



3. 「FullGCレス」を実現する最新技術

-FullGCレス機能の仕組みとは-

業務プログラム

(1)セッションの生成

```
HttpSession session = request.getSession();
```

(2)オブジェクトの生成およびセッションへのオブジェクトの格納

```
Object obj1 = new UserObject1();
session.setAttribute(Key1, obj1);
```

(3)オブジェクトの生成およびセッションへのオブジェクトの格納

```
Object obj2 = new UserObject2();
session.setAttribute(Key2, obj2);
```

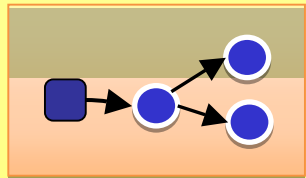
(4)セッションの終了

```
session.invalidate();
```

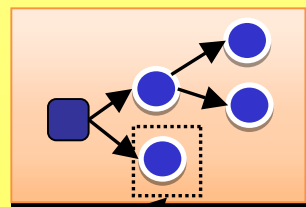
Cosminexus

メモリ領域

Eヒープ



New領域



CopyGCで移動

3. 「FullGCレス」を実現する最新技術

-FullGCレス機能の仕組みとは-

業務プログラム

(1)セッションの生成

```
HttpSession session = request.getSession();
```

(2)オブジェクトの生成およびセッションへのオブジェクトの格納

```
Object obj1 = new UserObject1();
session.setAttribute(Key1, obj1);
```

(3)オブジェクトの生成およびセッションへのオブジェクトの格納

```
Object obj2 = new UserObject2();
session.setAttribute(Key2, obj2);
```

(4)セッションの終了

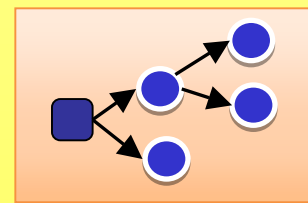
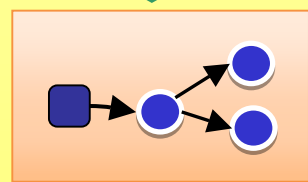
```
session.invalidate();
```

Cosminexus

明示管理ヒープ
領域削除

メモリ領域

Eヒープ

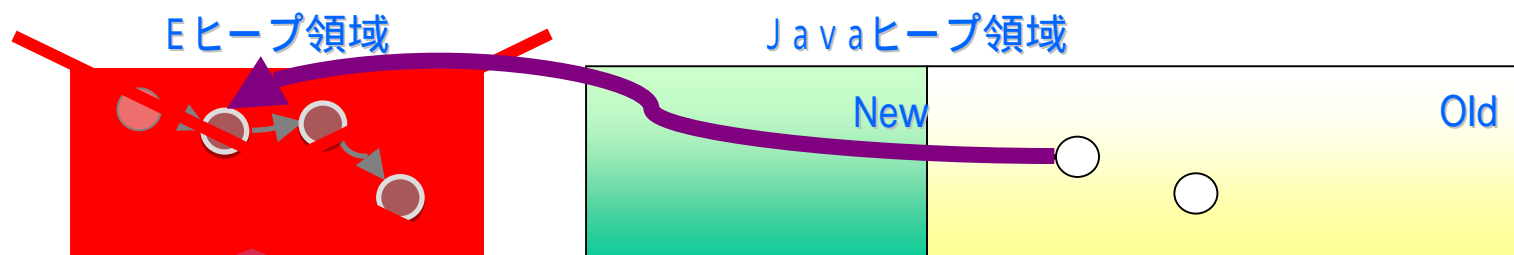


アプリ変更無しにセッションオブジェクトをEヒープで管理

3. 「FullGCレス」を実現する最新技術

-「Eヒープ領域削除」の堅牢化-

- 削除するEヒープ内に共通データなど使用中データが存在する場合
→単純な削除ではトラブル(領域外参照)が発生

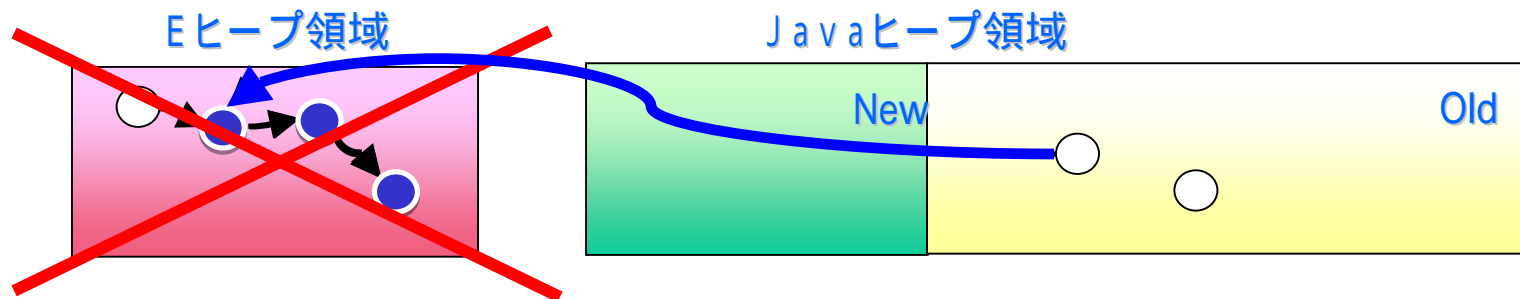


単純な削除ではアベンド発生！

3. 「FullGCレス」を実現する最新技術

-「Eヒープ領域削除」の堅牢化-

- 削除するEヒープ内に共通データなど使用中データが存在する場合
→単純な削除ではトラブル(領域外参照)が発生

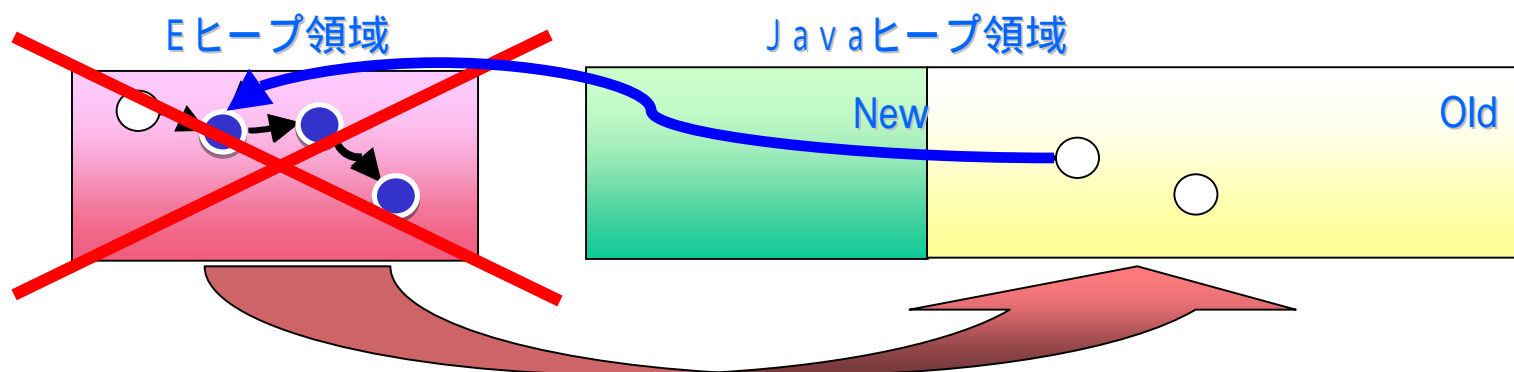


領域削除時に使用中
オブジェクトを検出して移動

3. 「FullGCレス」を実現する最新技術

-「Eヒープ領域削除」の堅牢化-

- 削除するEヒープ内に共通データなど使用中データが存在する場合
→単純な削除ではトラブル(領域外参照)が発生

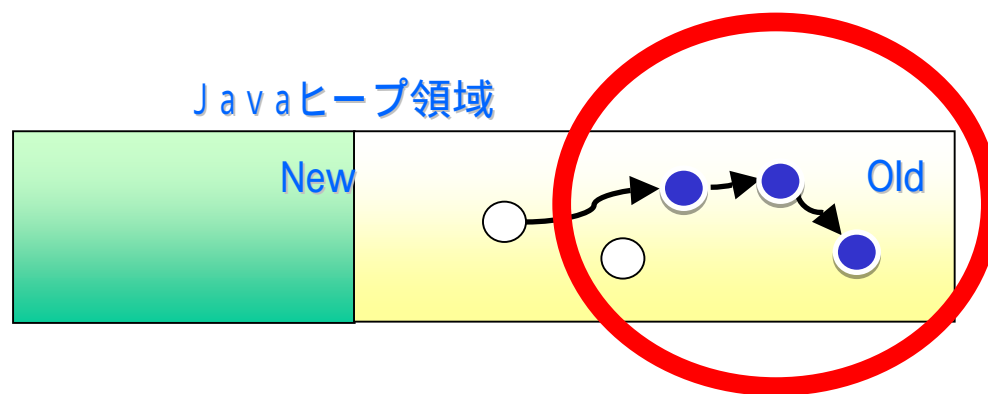


領域削除時に使用中
オブジェクトを検出して移動

3. 「FullGCレス」を実現する最新技術

-「Eヒープ領域削除」の堅牢化-

- 削除するEヒープ内に共通データなど使用中データが存在する場合
→単純な削除ではトラブル(領域外参照)が発生



領域削除時に使用中
オブジェクトを検出して移動

Eヒープ内に使用中データが存在する場合にはJavaヒープへの移動を行い、そのEヒープを削除してしまっても問題なく動作

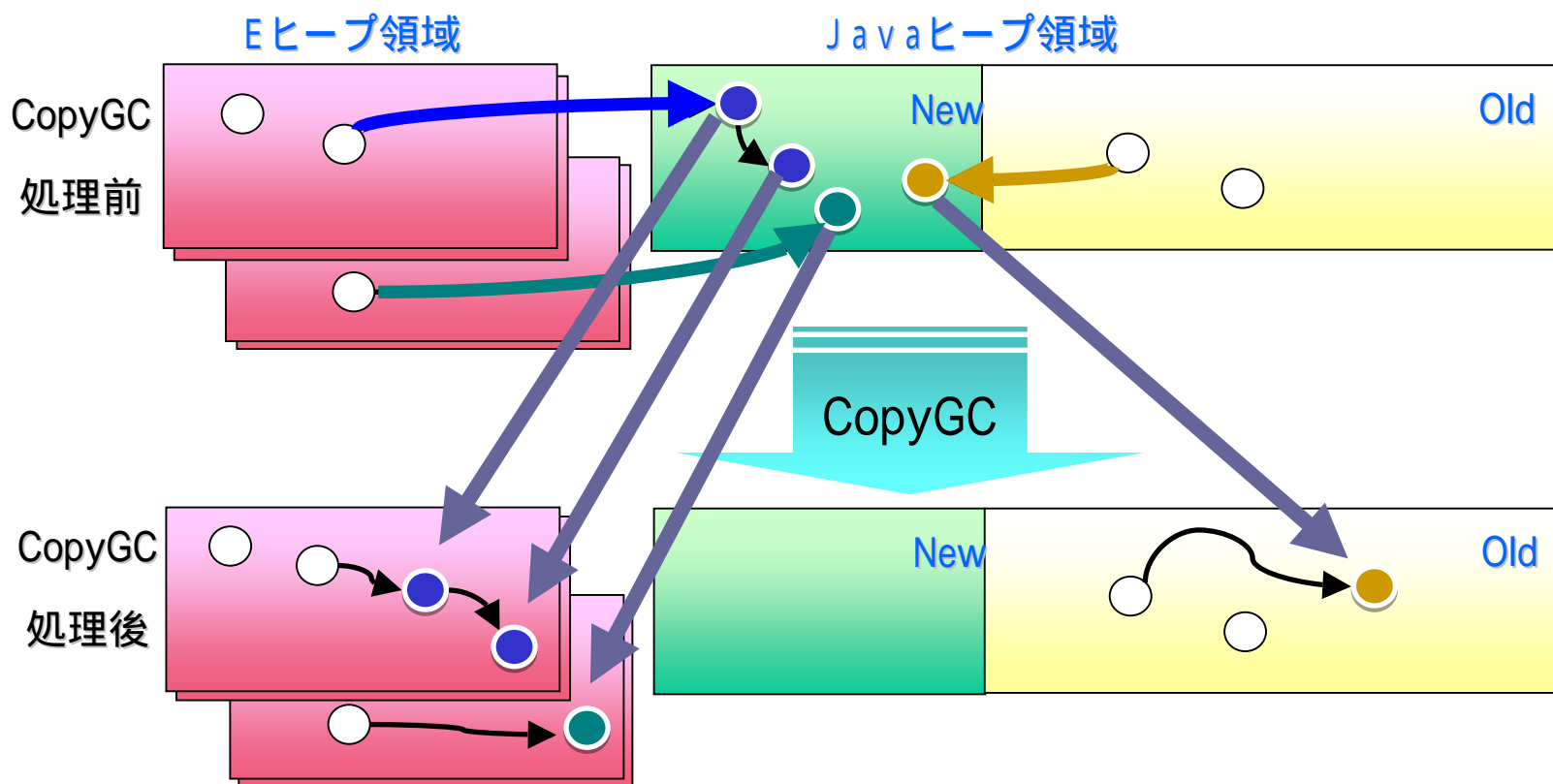
→ システム堅牢性を確保

3. 「FullGCレス」を実現する最新技術

-FullGCレス機能適用時のオーバーヘッド-

- FullGCレス機能適用時CopyGC処理のコスト

従来のCopyGC処理時間の**最大30%増**



CopyGC時にEヒープ領域への移動候補の解析及び移動先がJavaヒープかEヒープかの移動先判定を行う

3. 「FullGCレス」を実現する最新技術

-FullGCレス機能適用時のオーバーヘッド-

特許出願済

[単価極小(数十ms ~ 数百ms)のCopyGCに対して30%増のオーバーヘッドのみ]

FullGCの停止処理コストを広く浅くオンライン業務処理全域に割り振るのではなく、FullGC対象領域を縮小させる という抜本的な方式のため、処理全体に占める増加オーバーヘッドはCopyGCオーバーヘッド増という限定されたものとなる。

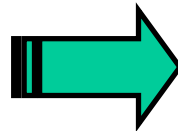
[CopyGC時間増によるスループット劣化の試算例]

測定単位時間: 10:00 ~ 15:45(20700秒)

CopyGC回数: 1658回

平均CopyGC時間: 0.253秒/回

処理件数: 548819件



平均CopyGC時間: 0.328秒/回 (30%増)

処理件数: 545486件 (0.61%劣化)

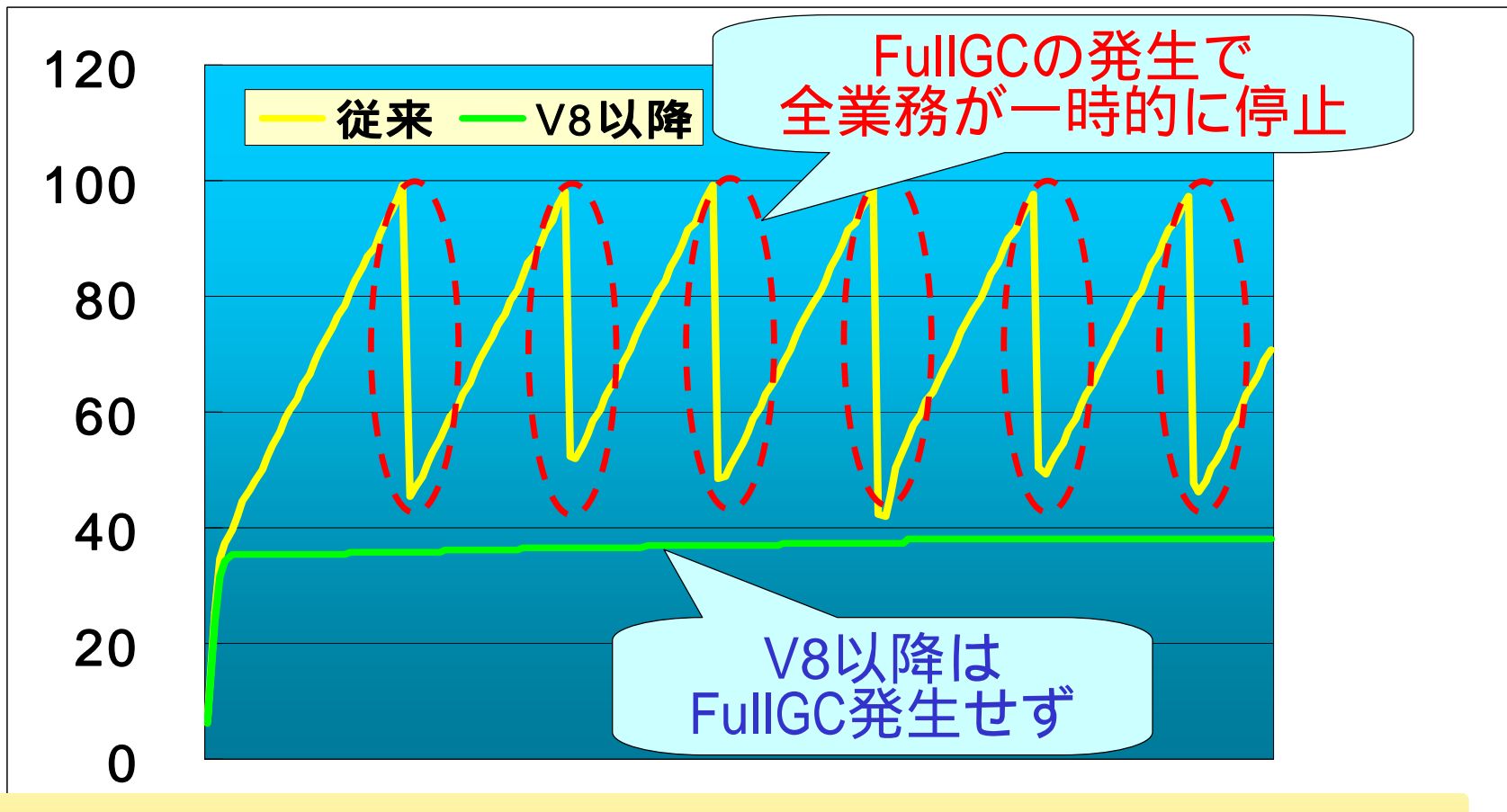
処理性能への影響

1%未満

3. 「FullGCレス」を実現する最新技術

-FullGCレス機能の効果例-

- Cosminexus動作時のメモリ使用量変化: A銀行殿縮小模擬環境



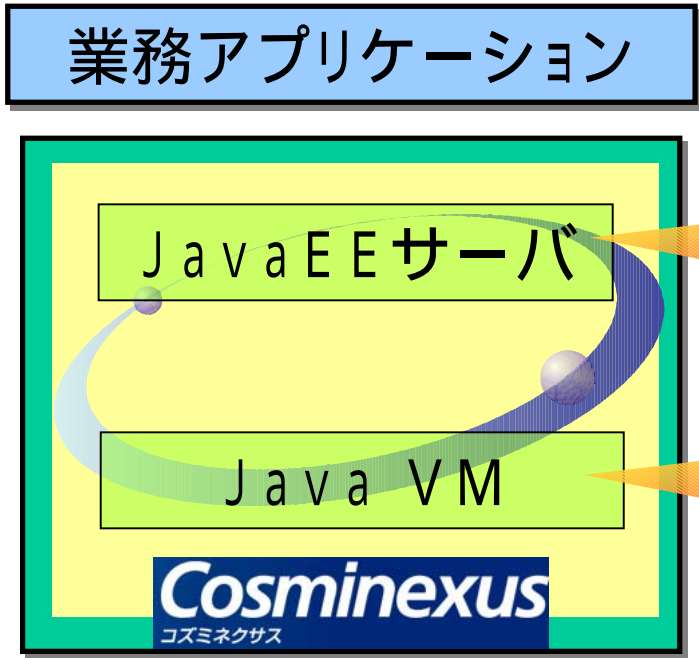
「FullGCレス」を実現！

GC対象外領域でメモリ大容量化対応！

3. 「FullGCレス」を実現する最新技術

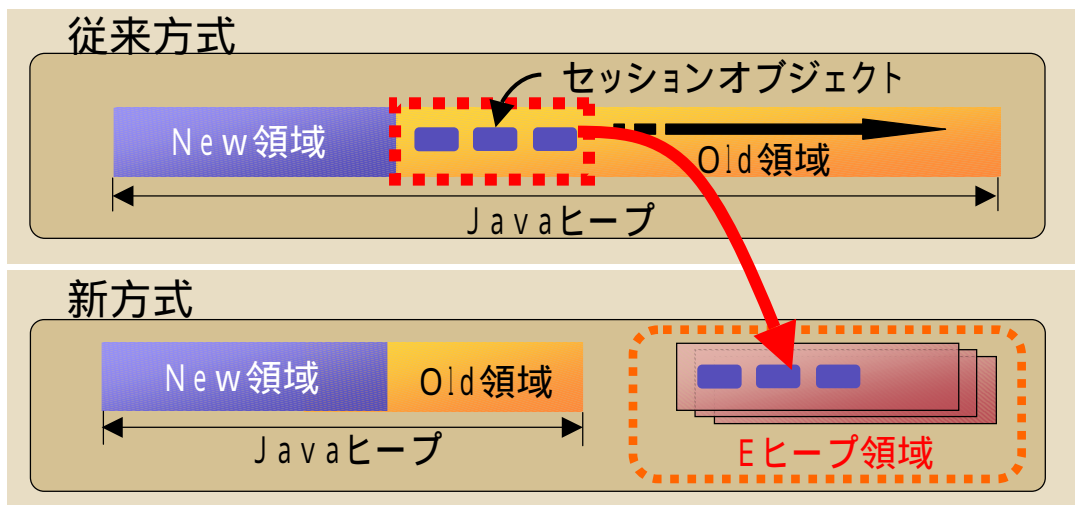
-Cosminexusでしか実現できない理由-

JavaEEサーバと
JavaVMの連携により
業務アプリの変更無しに
「FullGCレス」を
実現可能に！



セッションオブジェクトを
自動Eヒープ化

Eヒープ領域を追加



3. 「FullGCレス」を実現する最新技術

-Cosminexusでしか実現できない理由-

業務アプリケーション

Cosminexusのサポート
プラットフォーム全てで
「FullGCレス」を実現！

日立Java VM

Cosminexus
コスミネクス

全プラットフォームに
独自Java VMを開発
(Cosminexus V5より)

OS

HP-UX 11i

HP-UX

Windows

Windows



Linux



AIX

Solaris

Solaris

ハードウェア



Power



Itanium



x86/x64



Sparc

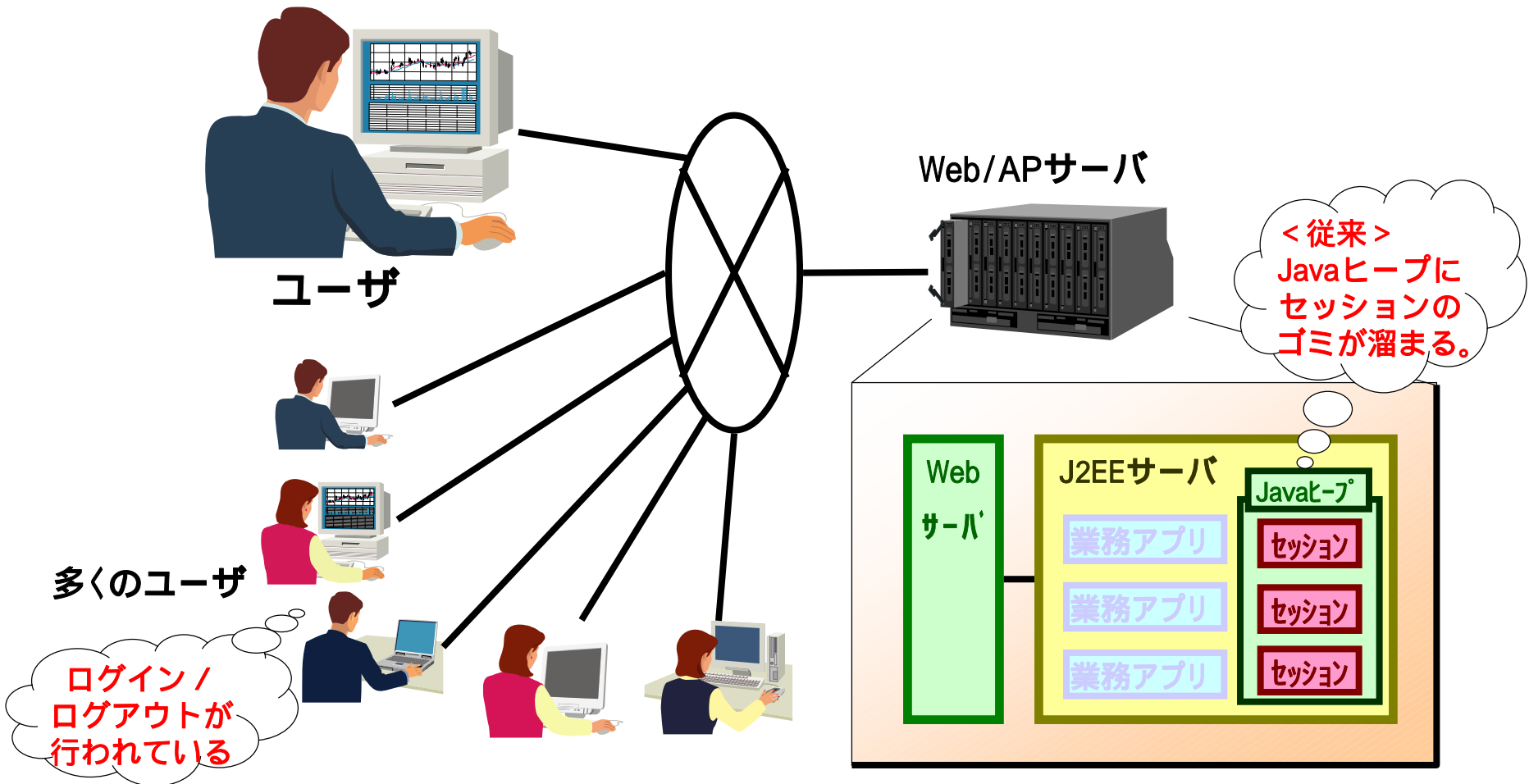
Contents

1. Javaのメモリ管理の課題とは
2. GC (ガベージコレクション) の仕組み
3. 「FullGCレス」を実現する最新技術
4. 「FullGCレス」の効果 (デモンストレーション)
5. メモリトラブルを起こさないためには

4. 「FullIGCレス」の効果 (デモンストレーション)

-デモンストレーションの構成-

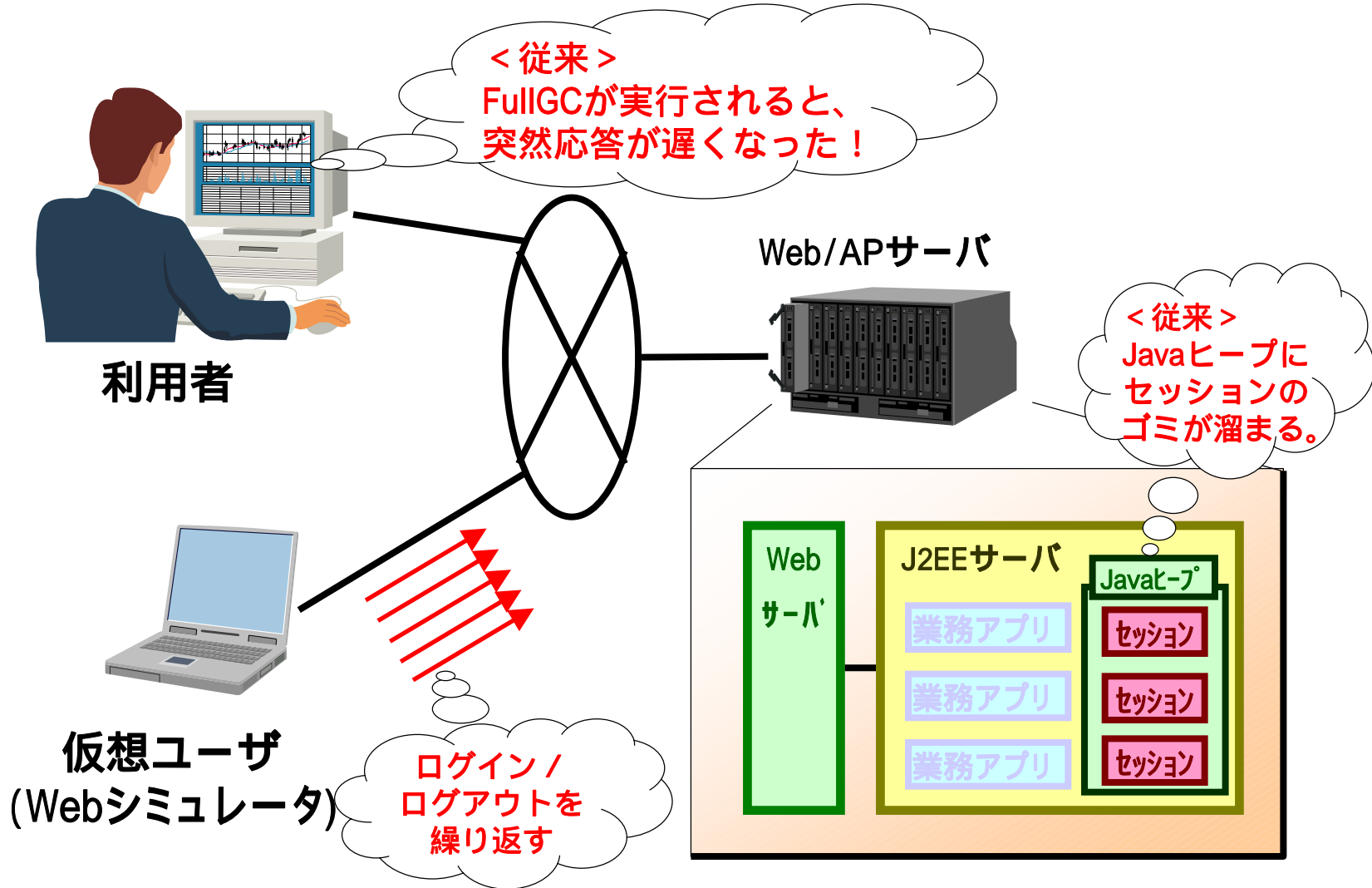
現実のシステムでは、多くのユーザがサーバにログインします。



4. 「FullIGCレス」の効果 (デモンストレーション)

-デモンストレーションの構成-

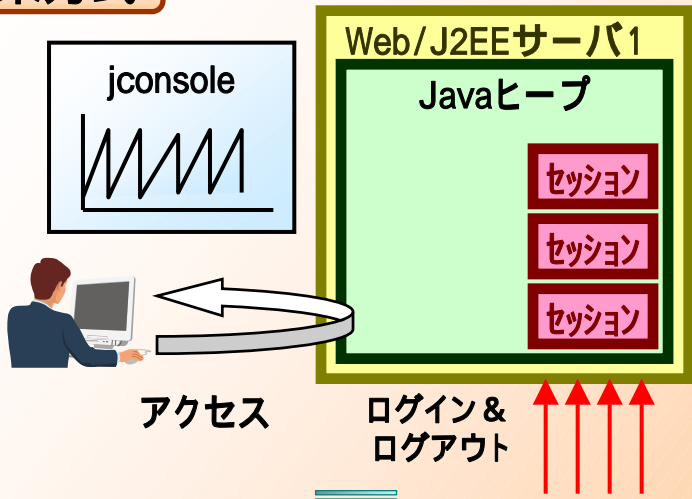
ここでは、シミュレータを使って多くの仮想ユーザでログインします。



4. 「FullGCレス」の効果 (デモンストレーション)

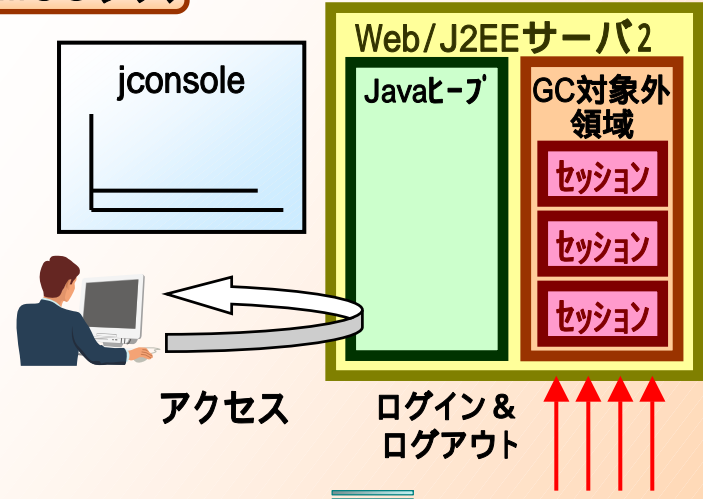
-デモンストレーションの内容-

従来方式



ブラウザ(画面)
時間: [ms]

FullGCレス



ブラウザ(画面)
時間: × × [ms]

比較

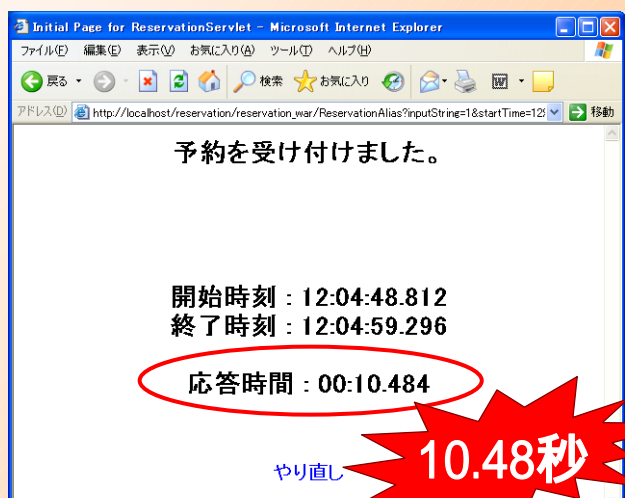
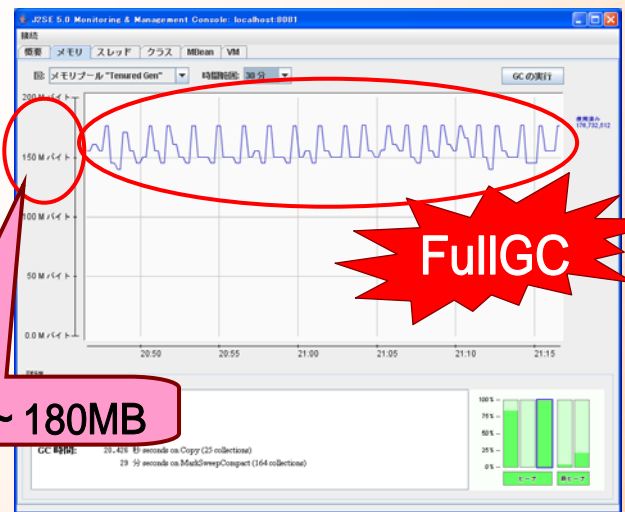
4. 「FullIGCレス」の効果(デモンストレーション)

スクリーンをご覧ください。

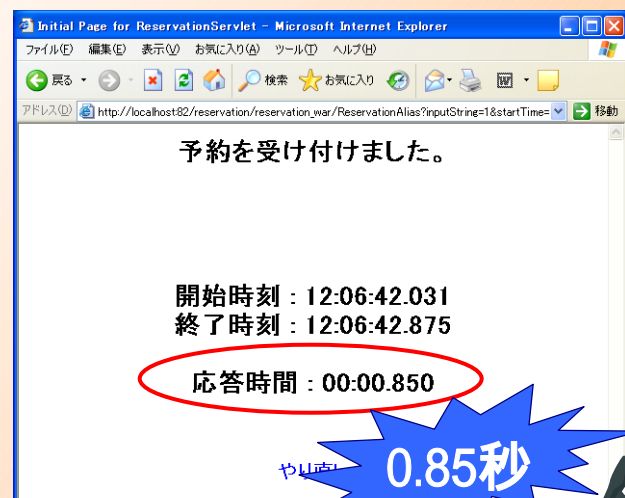
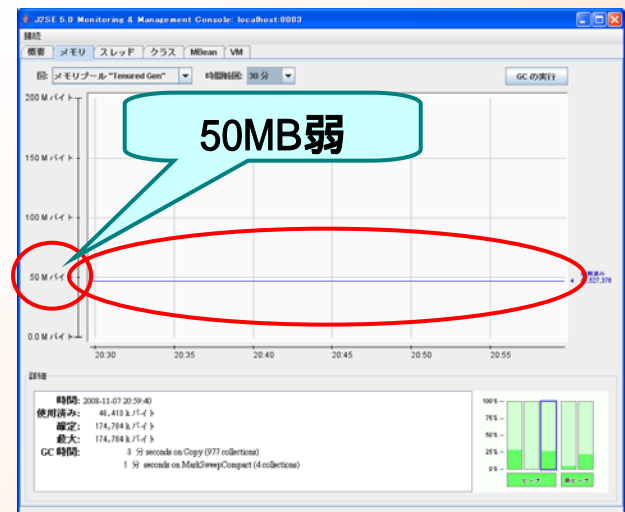
4. 「FullGCレス」の効果(デモンストレーション)

-デモンストレーションの結果まとめ-

従来方式



FullGCレス



「FullGCレス」によりレスポンス遅延解消!



Contents

1. Javaのメモリ管理の課題とは
2. GC (ガベージコレクション)の仕組み
3. 「FullGCレス」を実現する最新技術
4. 「FullGCレス」の効果 (デモンストレーション)
5. メモリトラブルを起こさないためには

5.メモリトラブルを起こさないためには -パフォーマンス見積もりシート-

- 「パフォーマンス見積もりシート」を用いてラクラク見積もり
赤枠の項目の一部を入力すると、必要なメモリサイズ、CPU数などを自動サイジング

| Microsoft Excel - 見積もりシート.xls | | | | | | | | | | |
|------------------------------------|----------|-----------------------|------|----------------|----------|----------|------|---|-----------|---------------|
| A | B | C | D | E | F | G | H | I | J | |
| マシンサイジング | | | | | | | | | | |
| ■システム(全体)パフォーマンス | | | | | | | | | | |
| CPU数算出 | | | | | | | | | | |
| 性能要件 | | | | | | | | | | |
| 1秒あたりのリクエスト数(件/秒) | | | 20 | | | | | システム全体に必要なCPU数(個) | 1,485,714 | 1秒あたりのリクエ |
| 目標レスポンス時間(秒) | | | 2 | | | | | | | |
| CPU利用率(%) | | | 70% | | | | | | | |
| 想定処理性能 | | | | | | | | | | |
| 実行CPU時間(秒) | | | 0.04 | | | | | | | |
| 内部保留時間(秒) | | | 2 | | | | | ← ネットワーク時間を加えて、目標レスポンス時間(秒)を満たしているか確認しておきます | | |
| メモリサイズ算出 | | | | | | | | | | |
| Javaヒープ算出の性能要件、想定処理性能 | | | | | | | | | | |
| 1リクエストあたりの使用メモリ(MB) | | | 2 | | | | | システム全体に必要なEden領域サイズ(MB) | 520 | 1リクエストあたり |
| 1秒あたりのリクエスト数(件/秒) | | | 20 | | | | | システム全体に必要なJavaヒープサイズ(MB) | 1950 | (Eden領域サイズ+ |
| CopyGC発生許容間隔(秒) | | | 10 | | | | | 秒に1回の発生 | | |
| Explicitヒープ算出の性能要件、想定処理性能 | | | | | | | | | | |
| 1セッションあたりの使用メモリ(MB) | | | 0.1 | | | | | システム全体に必要なExplicitヒープサイズ(MB) | 58.5 | 1セッションあたり |
| 最大同時ログイン数(件) | | | 450 | | | | | | | |
| ■マシン1台のパフォーマンス | | | | | | | | | | |
| マシン台数の決定 | | | | | | | | | | |
| マシン台数 | | | 2 | | | | | マシン1台あたりのCPU数(個) | 1 | システム全体に必 |
| 固定値 | | | | | | | | | | 条件: 4個までが妥当 |
| Cosminexusが使用するJavaヒープサイズ(MB) | | | 100 | | | | | Cosminexus使用に必要なJavaヒープサイズ | 300 | (Cosminexusが使 |
| Cosminexusが使用するExplicitヒープサイズ(MB) | | | 20 | | | | | マシン1台あたりに必要なJavaヒープサイズ(MB) | 975 | システム全体に必 |
| Survivor比率(SurvivorをとしたときのEdenの比率) | | | 8 | | | | | | | |
| New比率(NewをとしたときのOldの比率) | | | 2 | | | | | マシン1台あたりのJavaヒープサイズ(MB) | 975 | システム全体に必 |
| | | | | | | | | マシン1台あたりのExplicitヒープサイズ(MB) | 50 | システム全体に必 |
| | | | | | | | | マシン1台あたりのJ2EEサーバが使用するメモリサイズ(MB) | 1665 | マシン1台あたりの |
| | | | | | | | | | | 条件: 最大2048MB |
| | | | | | | | | マシン1台あたりのAPサーバが使用するメモリサイズ(MB) | 2415 | Webサーバ(20MB) |
| 参考: 見積もり後のJ2EEサーバのメモリ構成(MB) | | | | | | | | | | |
| Javaヒープ | | | | | Explicit | Perm | Cヒープ | | | |
| New | | Old | | ヒープ | ヒープ | スレッドスタック | | | | |
| Eden | Survivor | セッション情報以外の業務プログラム使用領域 | | Cosminexus使用領域 | | | | | | |
| | | New領域用の回避領域 | | | | | | | | |
| | | | | | 50 | 128 | 512 | | | |
| | 325 | | | 650 | | | | | | |
| | 260 | 65 | 225 | 100 | | | | | | |

①入力欄の値を変更します。(グレーのセルは固定値のため、変更不要です。)

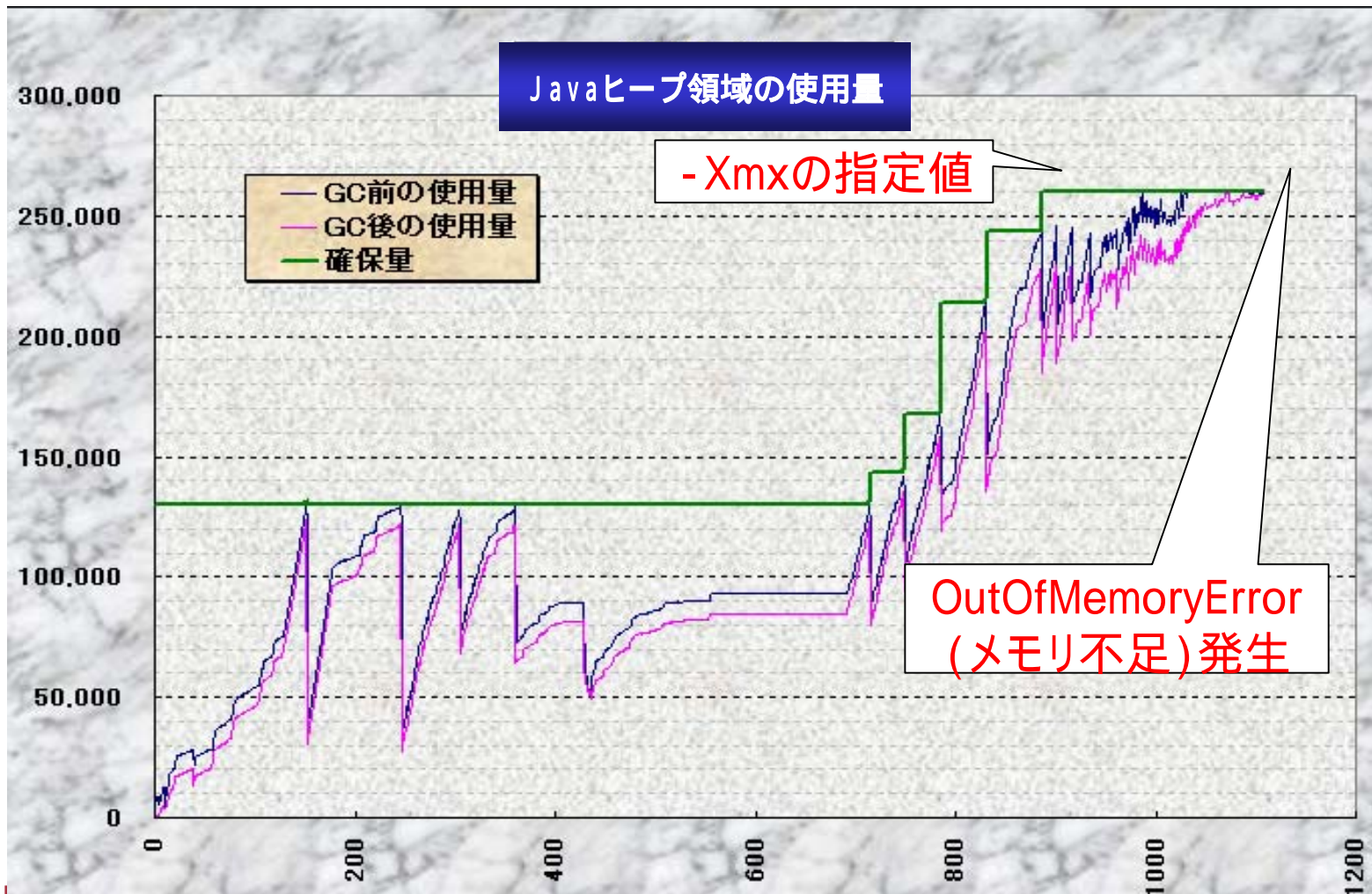
②CPU数・メモリサイズの条件に合うよう、マシン台数を調節します。

5.メモリトラブルを起こさないためには

-メモリリーク-

●メモリリークとは...

プログラマが予想しないところでオブジェクトの参照が残ってしまい、GCを行っても解放されないオブジェクトが継続的に増加してしまうこと



5.メモリトラブルを起こさないためには -メモリリーク原因の特定- 「ヒーププロファイル機能」

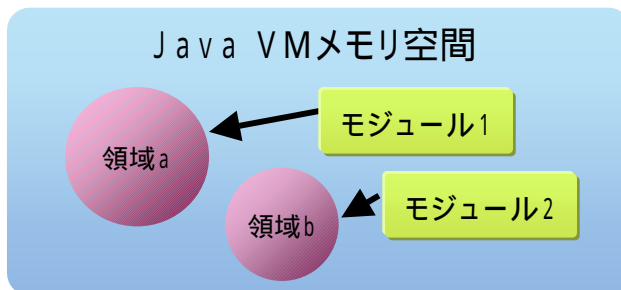
●メモリリークしているが、原因が特定できない。

ポイント

- メモリの解放モレ(オブジェクト参照の解除モレ)の原因を容易に追求することが可能になります。
- JavaVM自身で情報出力を行うため、**高い精度で解析**を行うことが可能です。
- 通常実行時のオーバーヘッドはゼロ。**
- 情報取得時のオーバーヘッドはFullGC 1回分程度の低オーバーヘッド。
- サーバーを**再起動することなく情報を取得**できます。

領域単位のメモリ消費量の把握

領域aが多くのメモリを消費していることを確認します。



領域を保持しているモジュールを把握

モジュール1が領域aを保持している。仕様/バグを調査

実際は領域/モジュールともJavaのクラスインスタンスになります。

Total Size of Instances

| Size | Instances | Class |
|---------|-----------|---------------------------------|
| 1437424 | 15809 | [Ljava.lang.Class; |
| 525120 | 7408 | java.util.HashMap\$Entry |
| 502000 | 7094 | java.util.HashMap |
| 500248 | 7012 | [Ljava.util.HashMap\$Entry; |
| 486336 | 4017 | java.lang.ref.SoftReference |
| 394760 | 4658 | java.util.HashSet |
| 394328 | 4648 | java.lang.Shutdown\$WrappedHook |
| ... | | |

Reference of class classC

```
classA(0x10766840)[Eden]
  classX(0x10766998)[Eden]
-----
classB(0x10766840)[Eden]
  classC(0x10766858)[Tenured]
  classD(0x10766968)[Eden]
  classX(0x10766a28)[Survivor]
-----
classE(0x10766840)[Eden]
  classA(0x10766920)[Survivor]
  classX(0x06aa0020)[EM(eid=1)]
```

classCをポイントしているすべてのクラスが一覧として出力されます。ヒープ領域名も表示します

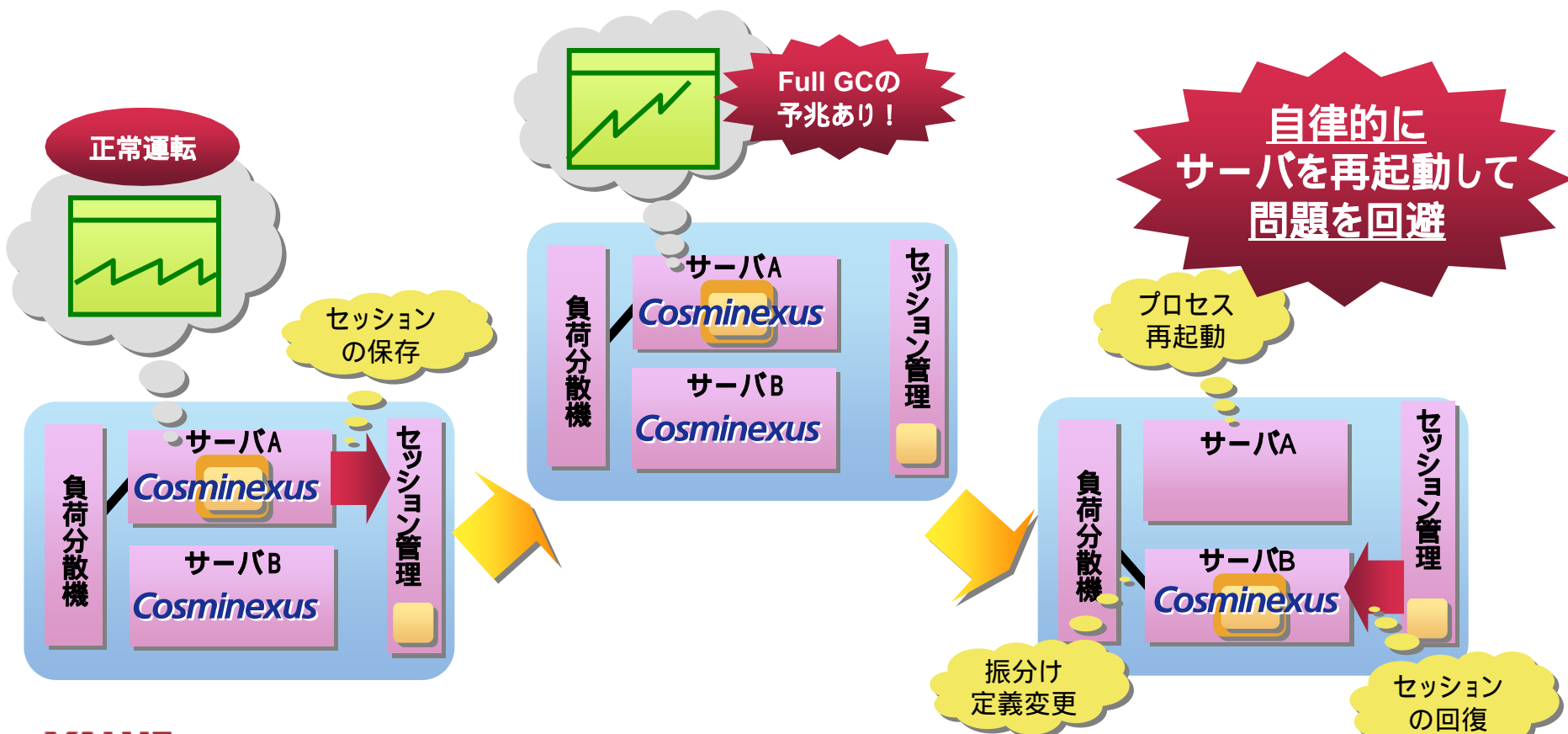
5.メモリトラブルを起こさないためには

-FullGC発生によるスローダウンの回避-

- FullGC発生によるスローダウンを事前に検知して回避したい。

ポイント

- FullGCの発生を事前に検知し自律的に再起動することで、トラブルを未然に防止。
- セッション情報を引継ぐため、業務を継続することができる。



- Cosminexus V8は、Webシステムの“Stop the world”解消のために「FullGCレス」を実現する機能を新たに搭載した。アプリの変更なく、誰でも恩恵を享受できる。
- Cosminexus 日立JavaVMの明示管理ヒープと、アプリケーションサーバとの密連携という高度な独自技術の結晶として、「FullGCレス機能」は実現されている。
- メモリリークなどのアプリ不良によって、メモリトラブルが発生する場合もある。これについての調査方法は確立されており、トラブルを暫定回避する機能もある。

Cosminexus ホームページ

<http://www.hitachi.co.jp/cosminexus/>

<http://www.cosminexus.com/>

Cosminexus 認定資格講座

<http://www.hitachi.co.jp/Prod/comp/soft1/certification/cosminexus/kouza.html>

謝辞および他社所有名称に対する表示

《他社所有名称に対する表示》

- Java 及びすべてのJava関連の商標及びロゴは、米国及びその他の国における米国Sun Microsystems, Inc.の商標または登録商標です。
- その他記載の会社名、製品名は、それぞれの会社の商号、商標もしくは登録商標です。

END