Oracle
DBA & Developer Days 2011

日本オラクル、今年最大の技術トレーニングイベント

2011年11月9日(水)～11月11日(金) シェラトン都ホテル東京

ORACLE

アプリケーションサーバ運用管理

Javaパフォーマンス・トラブル解決の実際

日本オラクル株式会社 Fusion Middleware 事業統括本部
シニアセールスコンサルタント　二川 秀智

以下の事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。以下の事項は、マテリアルやコード、機能を提供することをコミットメント（確約）するものではないため、購買決定を行う際の判断材料になさらないで下さい。オラクル製品に関して記載されている機能の開発、リリースおよび時期については、弊社の裁量により決定されます。

はじめに

- 私はWebLogic Server/Coherence/JRockitといったOracleのミドルウェアの中でも基盤となる製品の提案や導入支援を担当しております。

- お客様からWebLogic ServerやJavaアプリケーション周りの性能トラブルの相談を受けることがたびたびありますが、最近はOracle JRockit JVMに付属する監視ツール(JRockit Flight Recorderなど)を活用して、原因不明のトラブルをスムーズに解決できたケースが増えています。

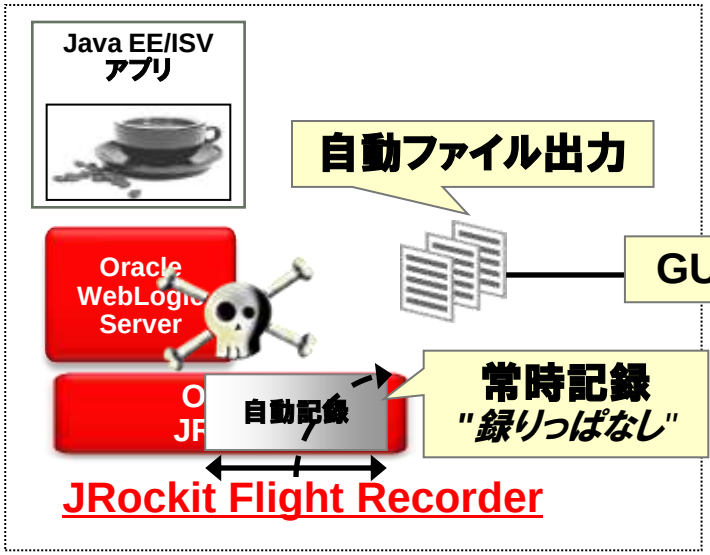
- 今回はそれらの中でも特徴的だったいくつかのケースについて詳細な解析フローを紹介します。

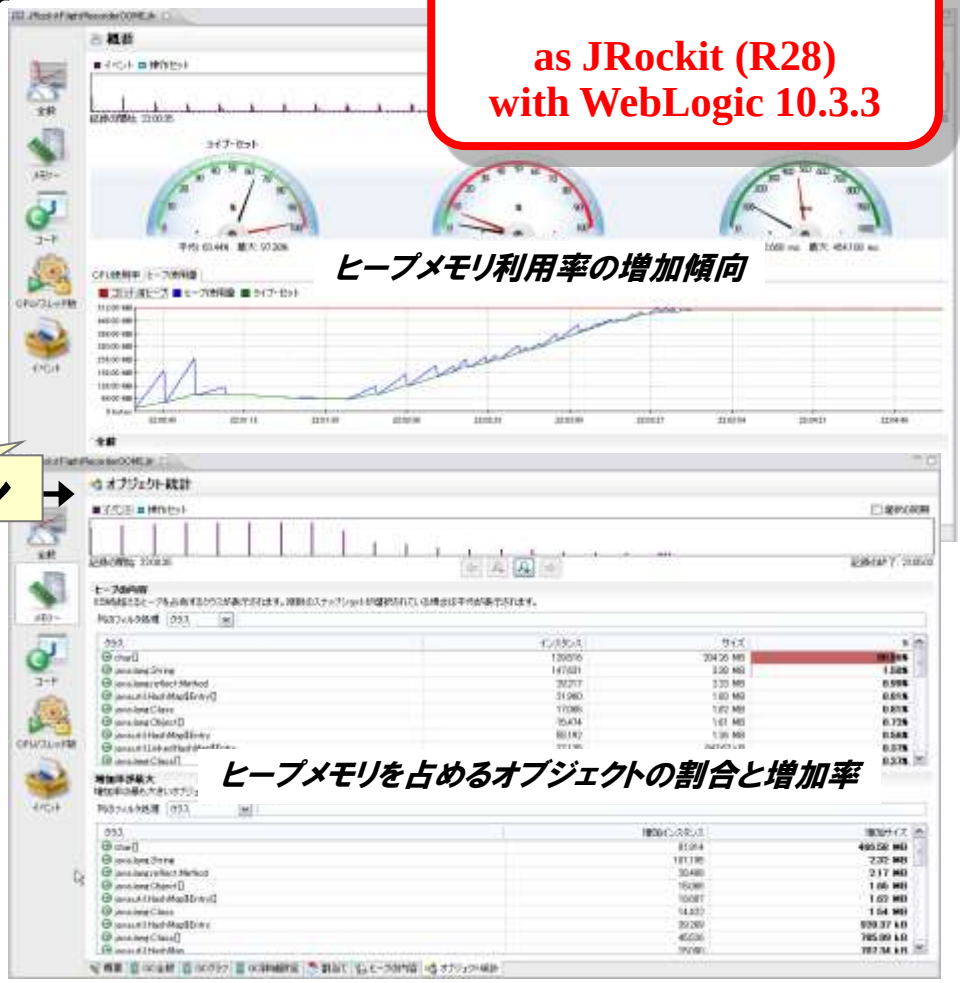
JRockit Flight Recorder

2010.07.06 New Release!

as JRockit (R28) with WebLogic 10.3.3

- 問題解決を強力に支援するOracle JRockitの新機能





- ✓ 独自の低負荷技術により本番環境での常時記録を実現
  - ✓ 記録した情報を自動ファイル出力
  - ✓ 迅速な原因究明を支援するGUIツール

1. トラブルの確実な原因追究を「遡って」実施可能
2. 障害発生→改善のサイクルと手間を大きく短縮化

ここからの話はノンフィクションですが・・・

- ここでは実際にお客様から相談のあった性能トラブルを3つ紹介します。いずれもお客様側でいろいろ調査をしたものの原因が分からず、やみくもに設定変更をしても効果が出ず、弊社に相談がありました。

- お客様の実際のシステムで発生した事象に基づくものですが、分かりやすくするためにシステムの構成は単純化しています。
- 登場するJRockit Flight RecorderやJRockit Memory Leak Detectorの取得例もお客様のシステムから直接取得したものではなく、弊社で同じ事象を再現させたものです。

アジェンダ

- 性能トラブル事例1
- 性能トラブル事例2
- 性能トラブル事例3
- まとめ
- (Appendix) JRockit Flight Recorder概要

ちなみに・・・

- このセッションの要点（かなり短縮版）はWebLogic Channelの以下のコラムでもご覧になれます。
  - 【前編】外部ライブラリに起因する問題も確実に追跡
    http://www.oracle.co.jp/campaign/weblogic/columns/column01/column06/post_25.html
  - 【後編】原因不明のOutOfMemoryエラー、性能劣化の問題も一挙に解消
    http://www.oracle.co.jp/campaign/weblogic/columns/column01/column06/outofmemory.html

エンタープライズ Java アプリケーションの開発、管理、運用基盤といえば

WebLogic Channel

http://www.oracle.co.jp/weblogic/

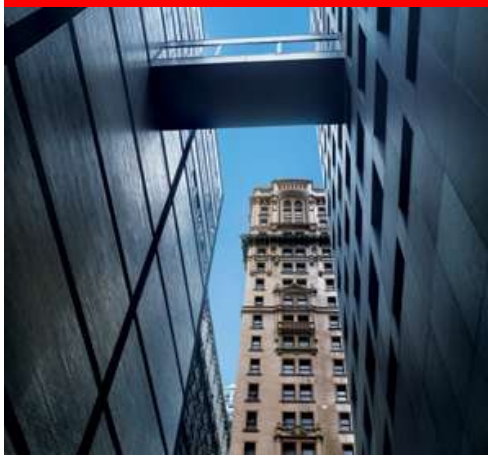
【後編】原因不明のOutOfMemoryエラー、性能劣化の問題も一挙に解消 [2011/10/17]

WebLogic Server Enterprise Editionでは、システム監視機能「JRoc...

【前編】外部ライブラリに起因する問題も確実に追跡 [2011/10/13]

「Oracle WebLogic Server Enterprise Edition」に備わるシステ...

- このセッションでは上記コラムに書ききれなかったことも含めて詳細に紹介します。



性能トラブル事例1(A社)

A社システムで発覚した性能問題

システム概要	•WebLogic Serverを利用してオンラインサービスを提供中 •高速レスポンス要件のためWebLogic Serverの背後でCoherenceをデータ・キャッシュとして利用 •非同期で外部システムと連携
トラブル状況	•高負荷時に 　•サービスのレスポンスが極端に悪化 　•外部システムへの連携のターンアラウンドタイムも極端に悪化 •WebLogic・外部システムともにCPU/Diskリソースには十分余裕があり、それらはネックとは考えにくい

お客様が事前に実施したこと

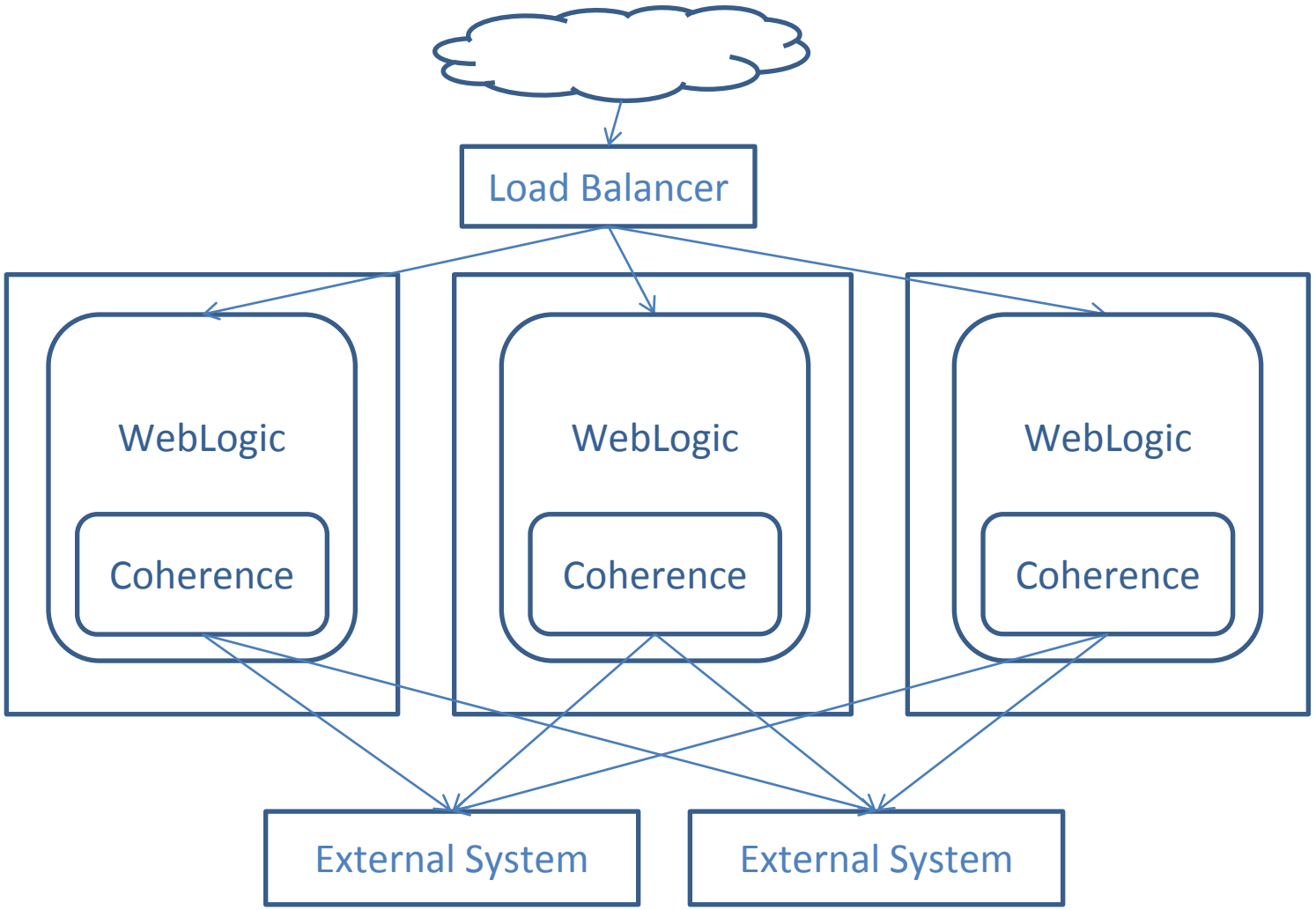
- WebLogic Serverの実行スレッド数を変えてみる
- Coherenceのワーカースレッド数を変えてみる

⇒しかし、いずれも効果なし・・・

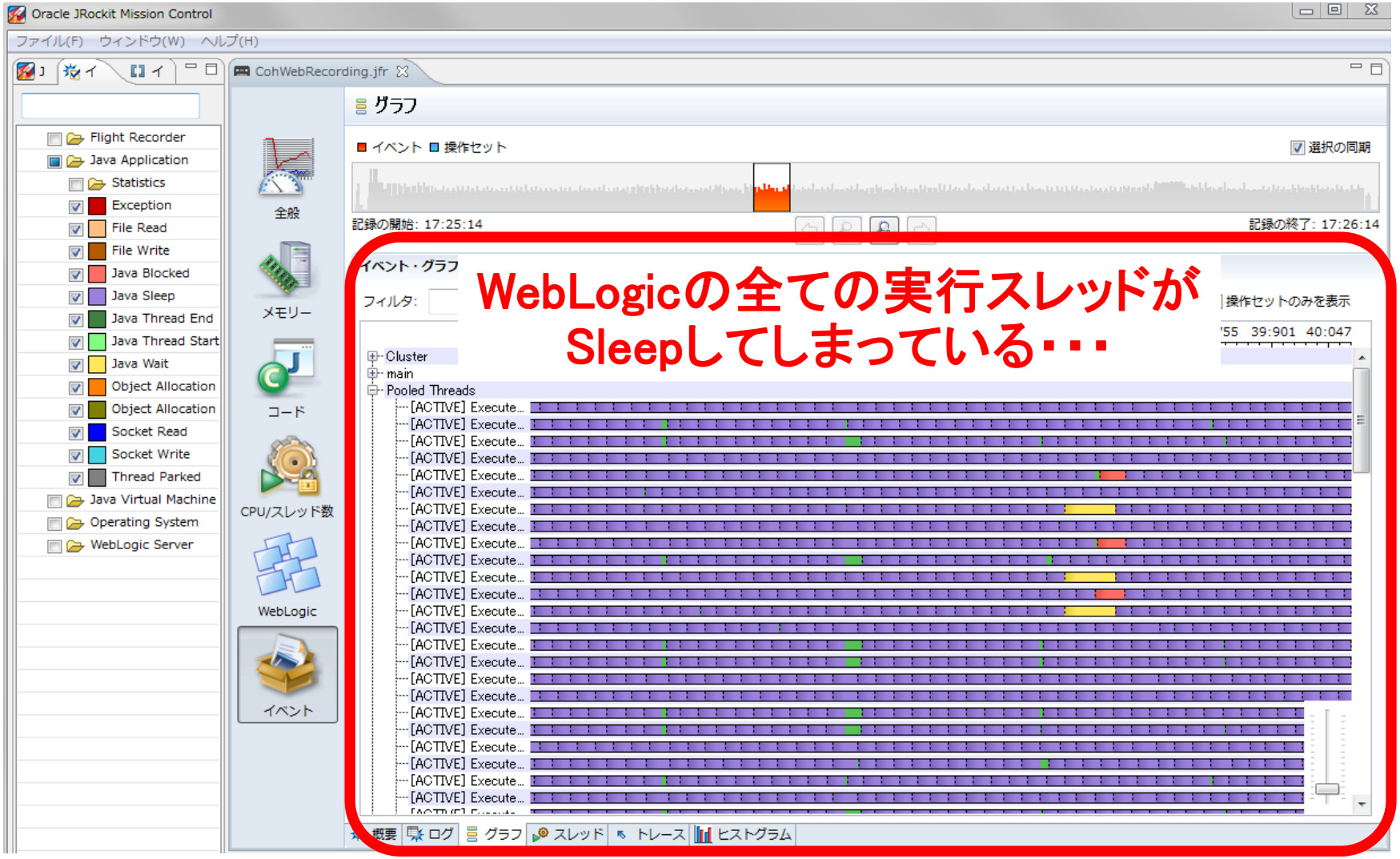
その後の解決までの流れ

- システム構成概要のヒアリング
- JRockit Flight Recorderの取得と解析
- 改善策の提案
- 改善策の実施

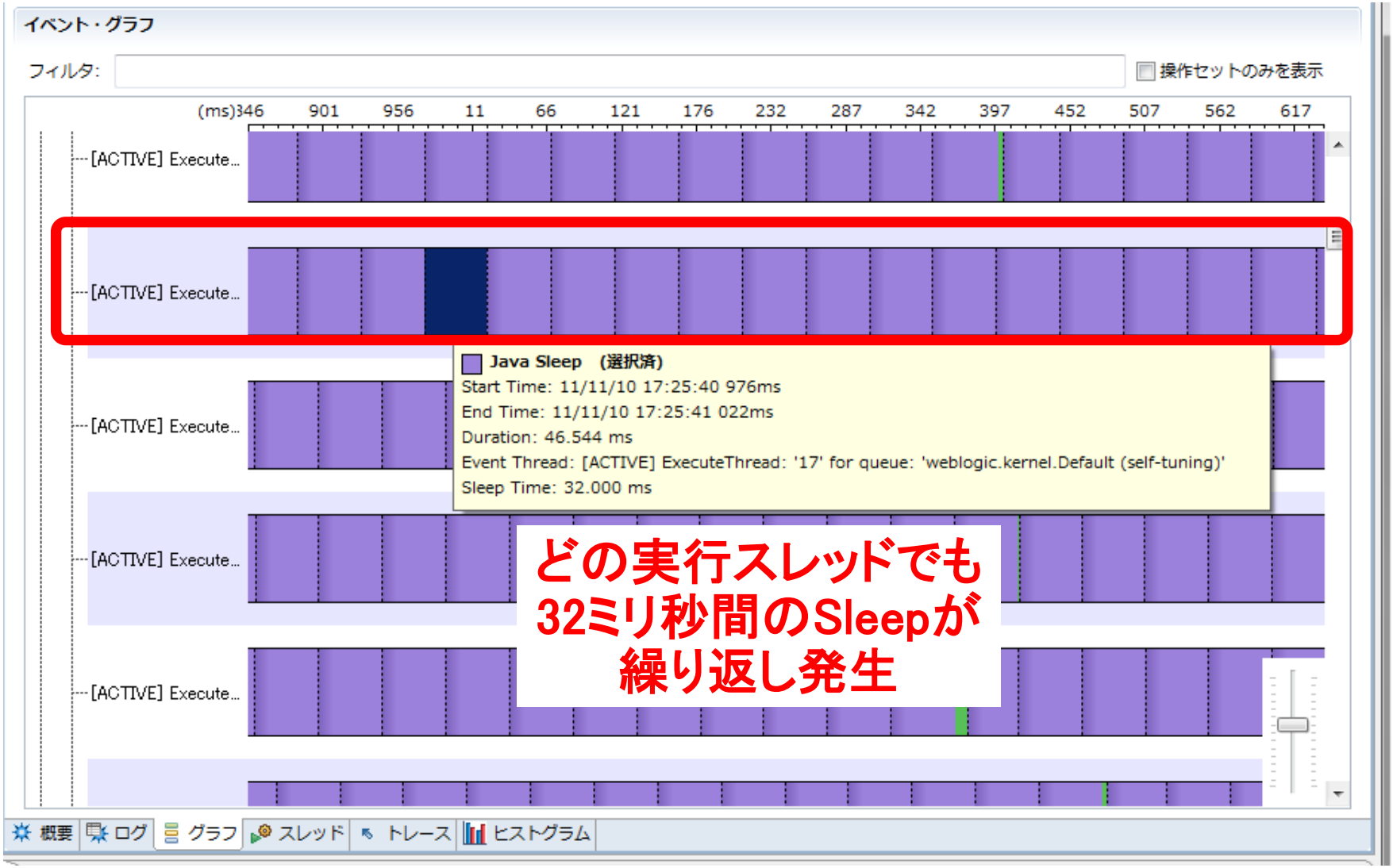
システム構成概要



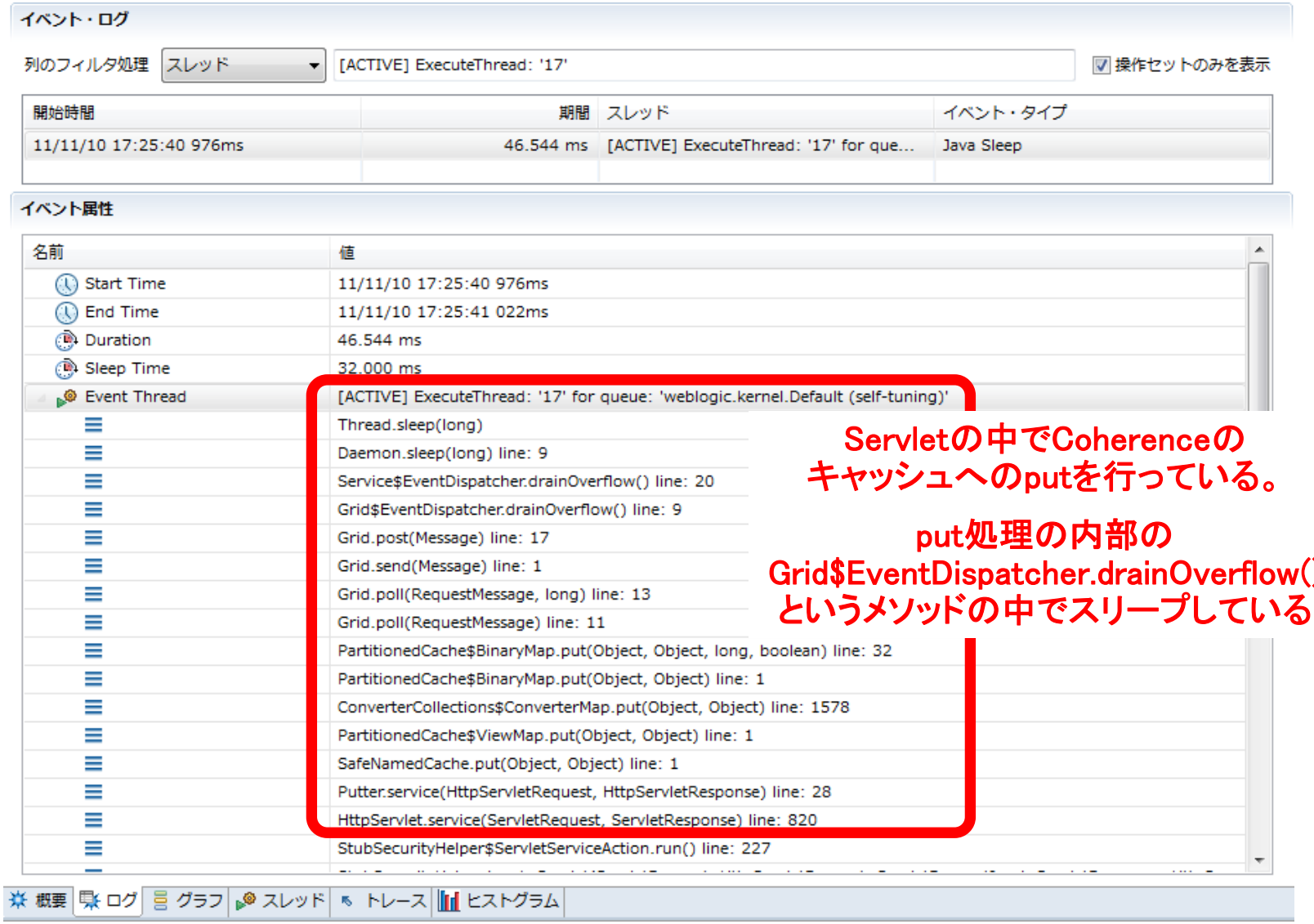
Flight Recorder イベントグラフ



イベントグラフを拡大して真相を追求



イベントログで遅延発生元のメソッドを特定



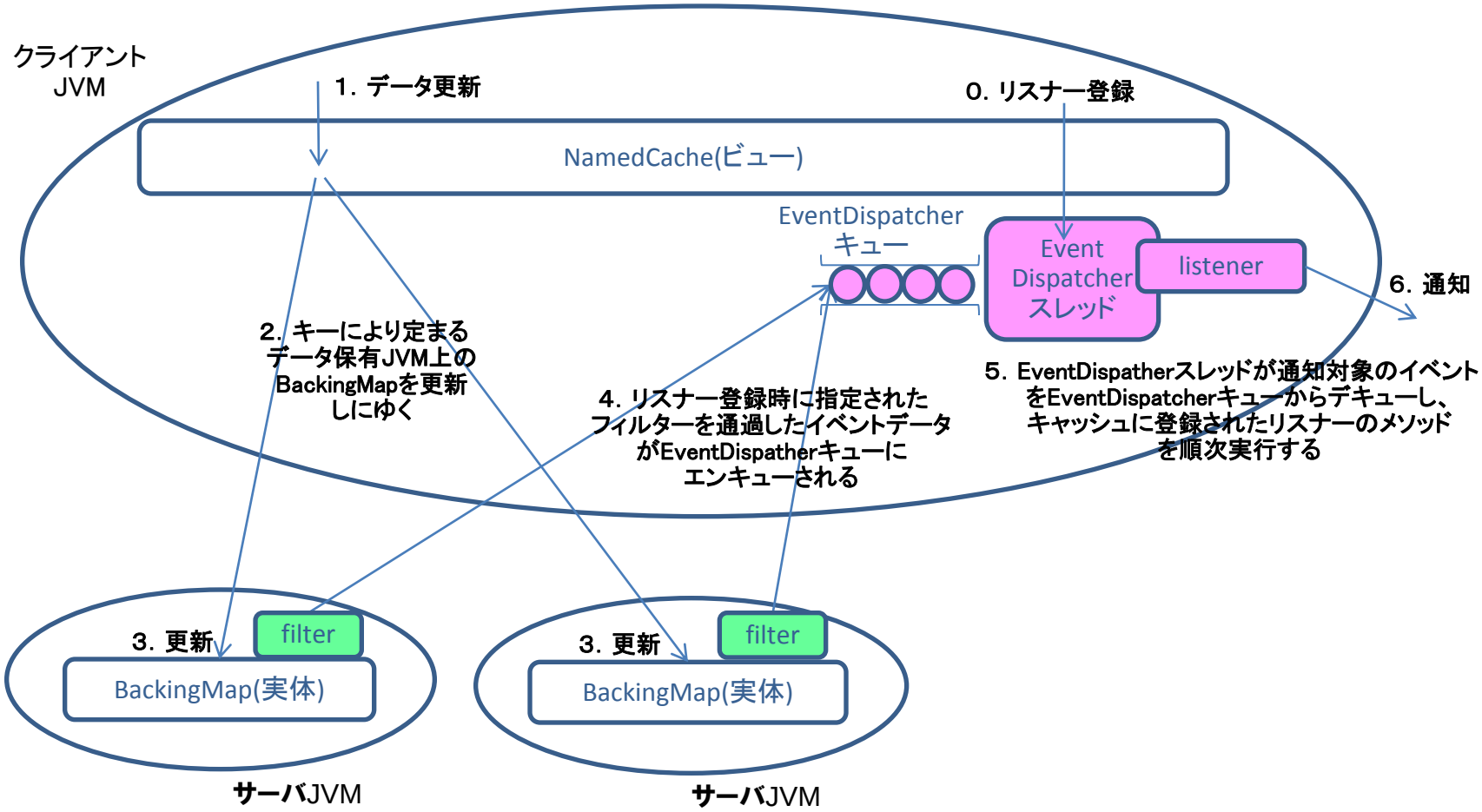
Servletの中でCoherenceの
キャッシュへのputを行っている。

put処理の内部の
Grid$EventDispatcher.drainOverflow()
というメソッドの中でスリープしている

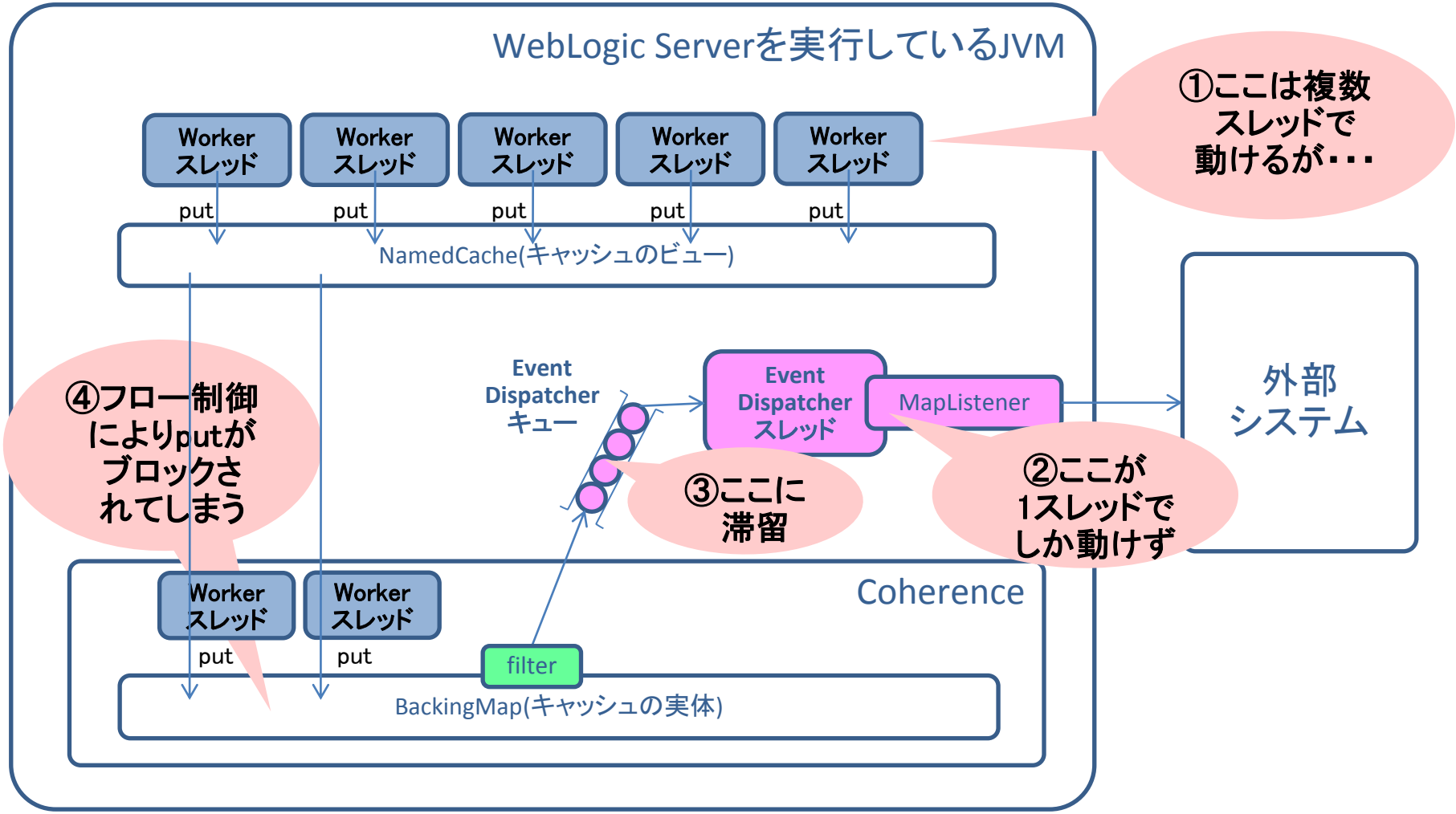
Grid$EventDispatcher.drainOverflow() ???

- このメソッドでサポート・ナレッジを探したところ正体が判明
  - Coherence Client Threads Sleep While in drainOverflow [ID 739404.1]
- Coherenceにはキャッシュ更新のタイミングで任意の処理を非同期で実行できるMapListenerという仕組みがある
- MapListnerへ処理を依頼するディスパッチ・キューがある
- ディスパッチ・キューに大量の依頼(デフォルトでは1024件以上)が滞留すると、Coherenceは自身の正常稼働を保つために、キャッシュへの更新を一時的にストップさせるようなフロー制御を行っている
  - **これがGrid$EventDispatcher.drainOverflow()の正体**

【参考】Coherence MapListenerの仕組み

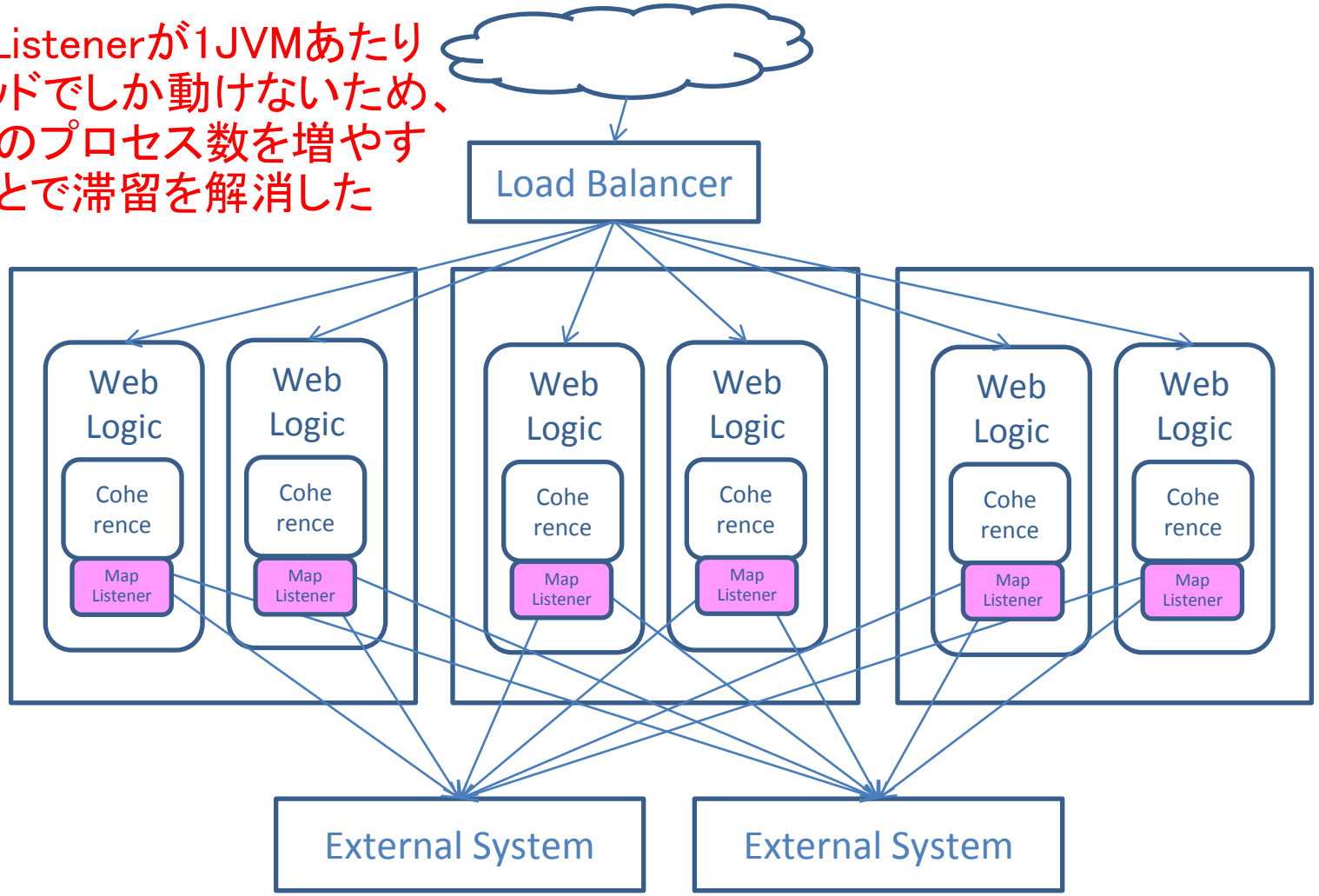


遅延の原因はMapListenerのボトルネック

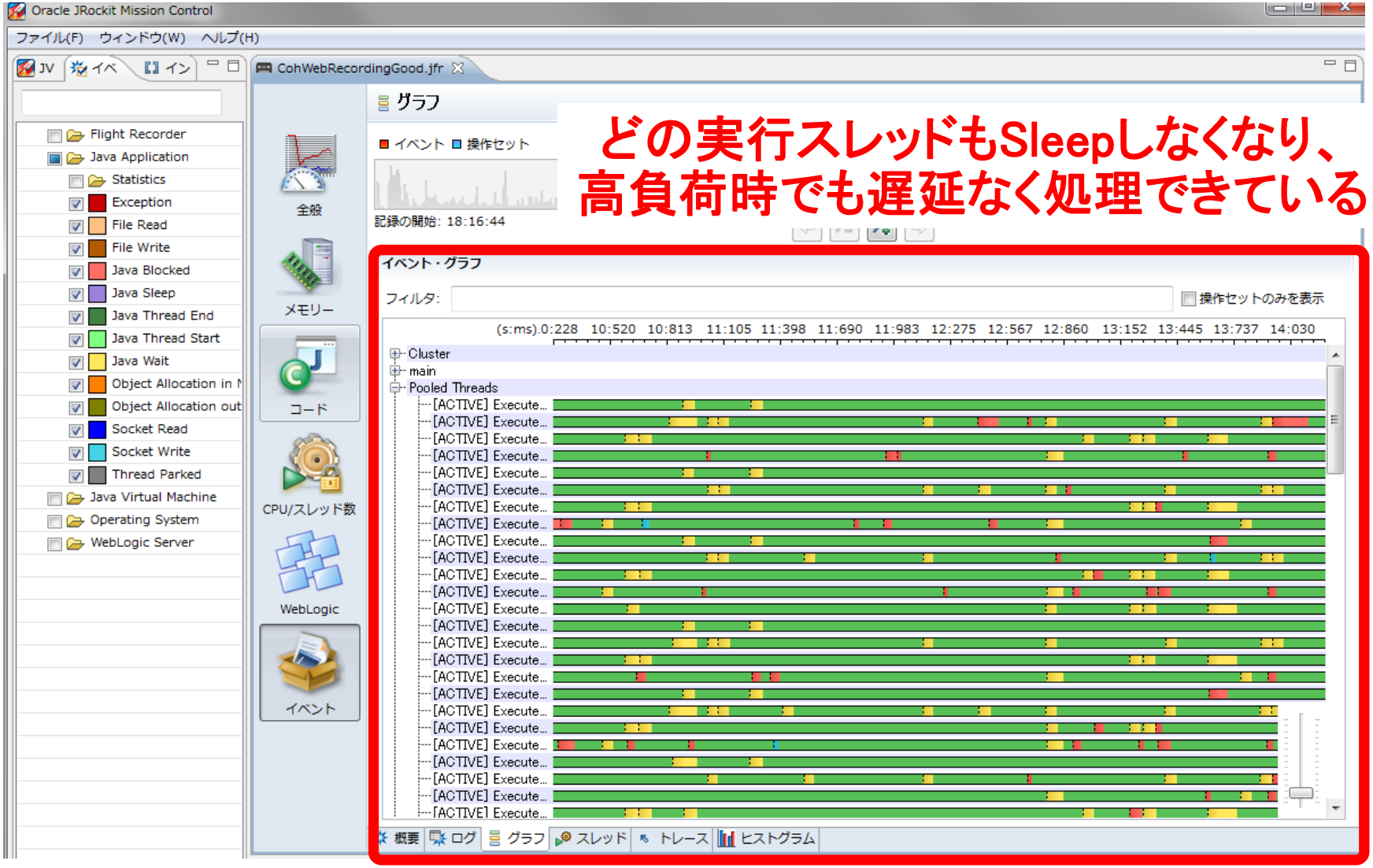


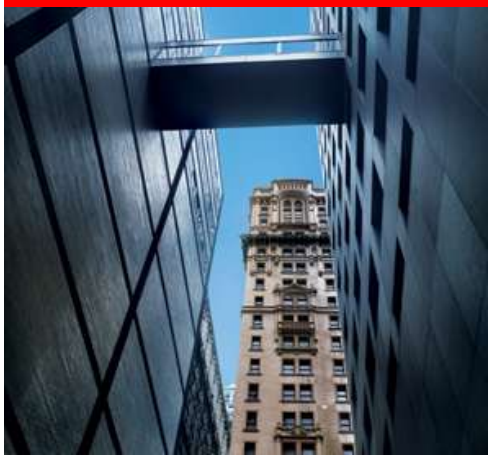
実施した対策(JVMの多重化)

MapListenerが1JVMあたり
1スレッドでしか動けないため、
JVMのプロセス数を増やす
ことで滞留を解消した



対策実施後のFlight Recorderの確認





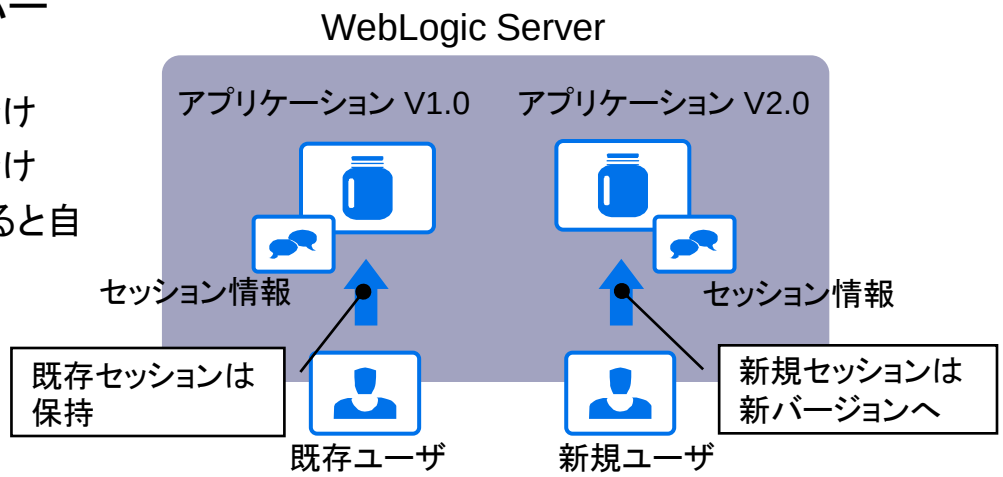
性能トラブル事例2(B社)

B社システムで発覚した性能問題

システム概要	•WebLogic Serverを利用して24時間オンラインサービスを提供中 •サービスの改善のためにアプリケーションのバージョンアップを頻繁に実施(WebLogicのプロダクション再デプロイメント機能を利用)
トラブル状況	•2～3日おきにOutOfMemoryError(OOME)が発生し、サービスが停止してしまう •暫定回避策として毎日トラフィックの少ない時間にWebLogic Serverを再起動している

【参考】WebLogic Serverのプロダクション再デプロイメント
～アプリケーションの更新を緩やかに実施する機能～

- プロダクション再デプロイメント　WLS 9.0 ～
  - アプリケーションを更新の際に緩やかなバージョン移行を実現
    - 新規セッションは新バージョンアプリに振分け
    - 既存セッションは旧バージョンアプリに振分け
    - 既存セッションが無くなるかタイムアウトすると自動的に旧バージョンアプリをリタイア
  - 対応モジュールタイプ
    - WAR
    - EAR(HTTP経由で利用するもの)
    - Web Services
    - RMI Client
  - バージョンコントロールフレームワーク
    - マニフェスト内のバージョン情報からアプリケーションのバージョン情報を把握



Manifestファイルのバージョン情報

```
Manifest-Version: 1.0
Weblogic-Application-Version: 1.0.1
```

アプリケーションの更新によるダウンタイムを極小化

お客様が事前に実施したこと

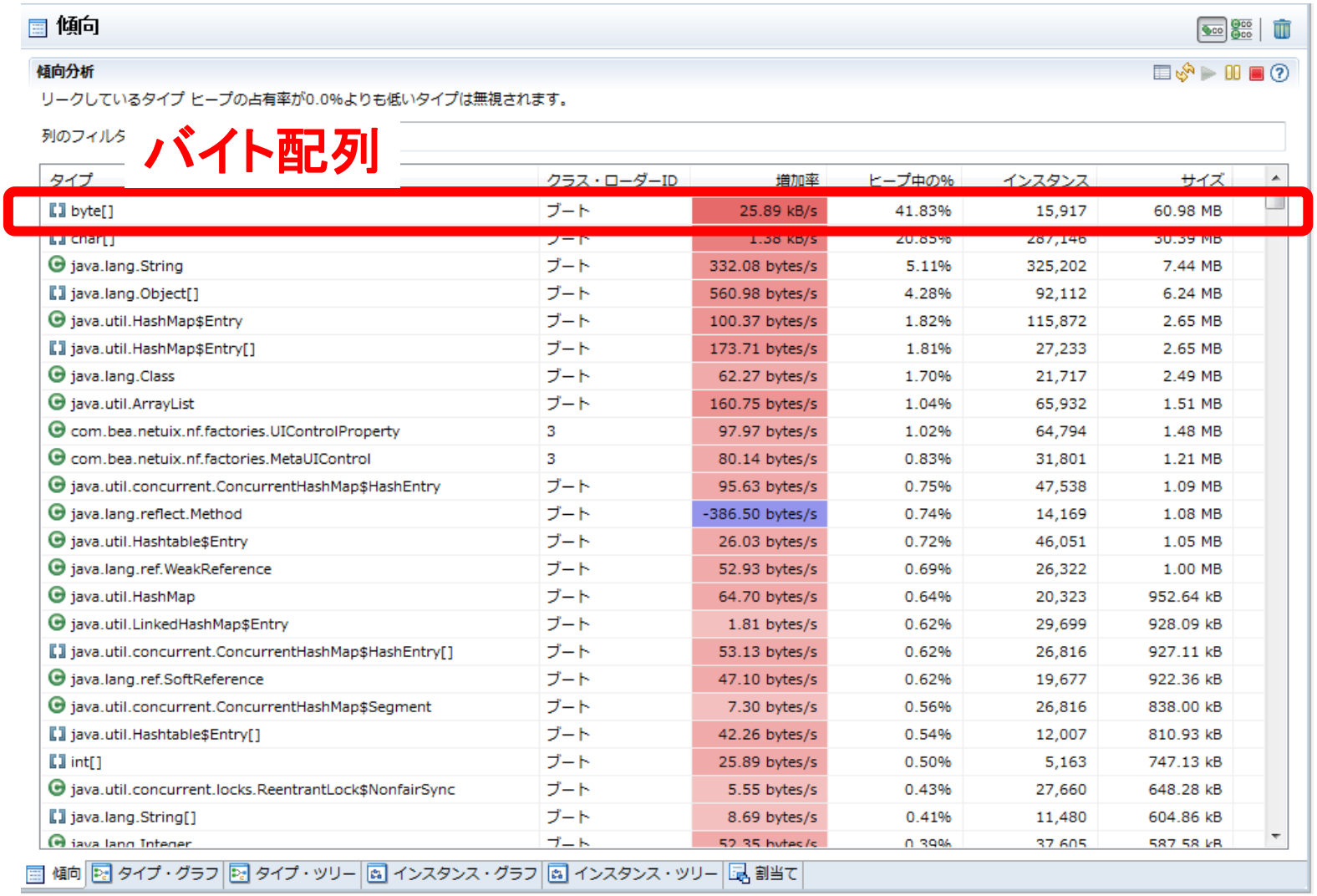
- JVMのヒープサイズを大きくしてみる

⇒しかし、OutOfMemoryErrorが発生するまでの猶予期間が長くなるだけであり、根本的な解決にはならず・・・

その後の解決までの流れ

- システム構成概要のヒアリング
- JRockit Memory Leak Detectorによる監視
  JRockit Flight Recorderの取得と解析
- 改善策の提案
- 改善策の実施

Memory Leak Detector 傾向

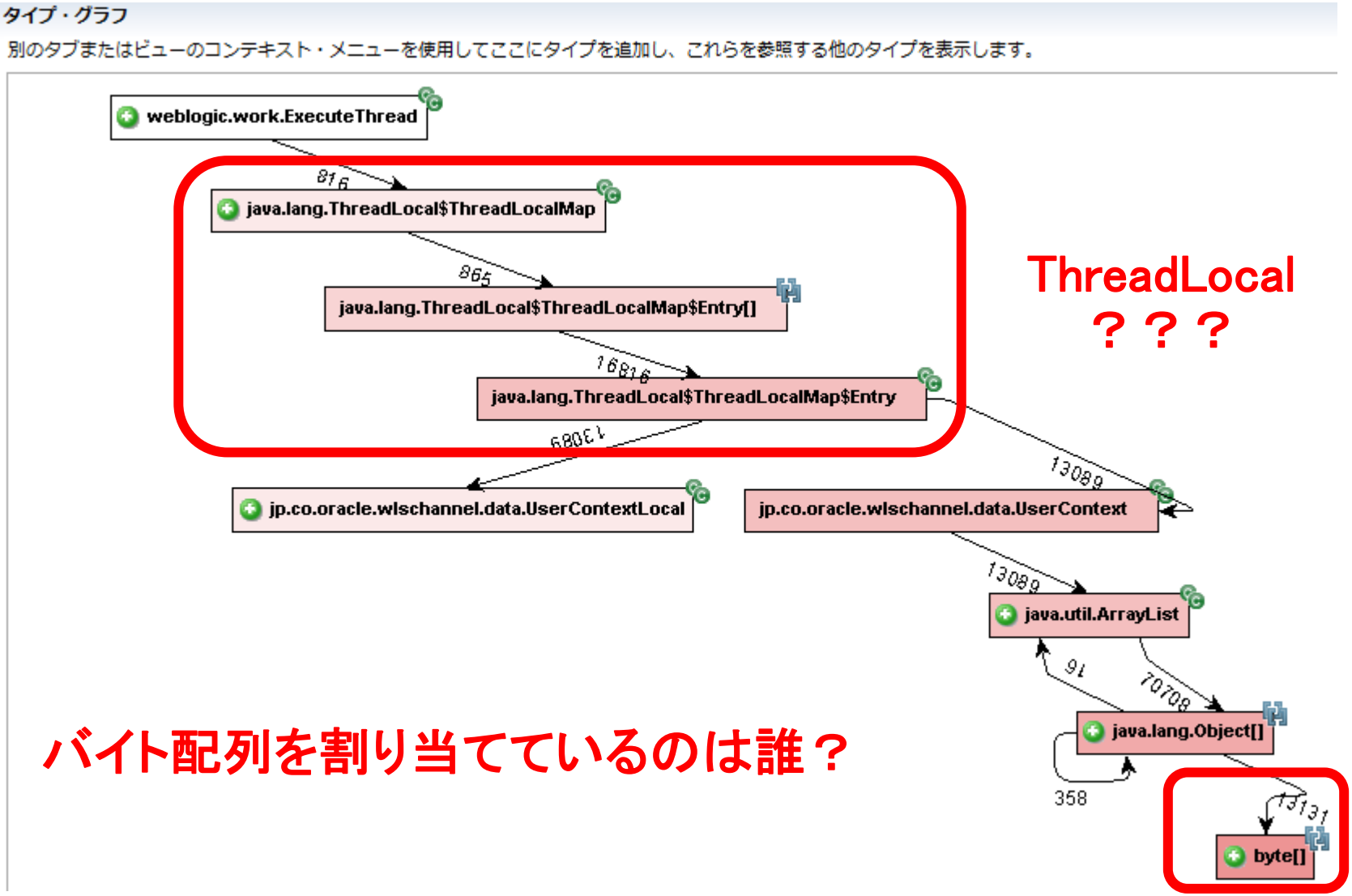


Memory Leak Detectorで分かったこと

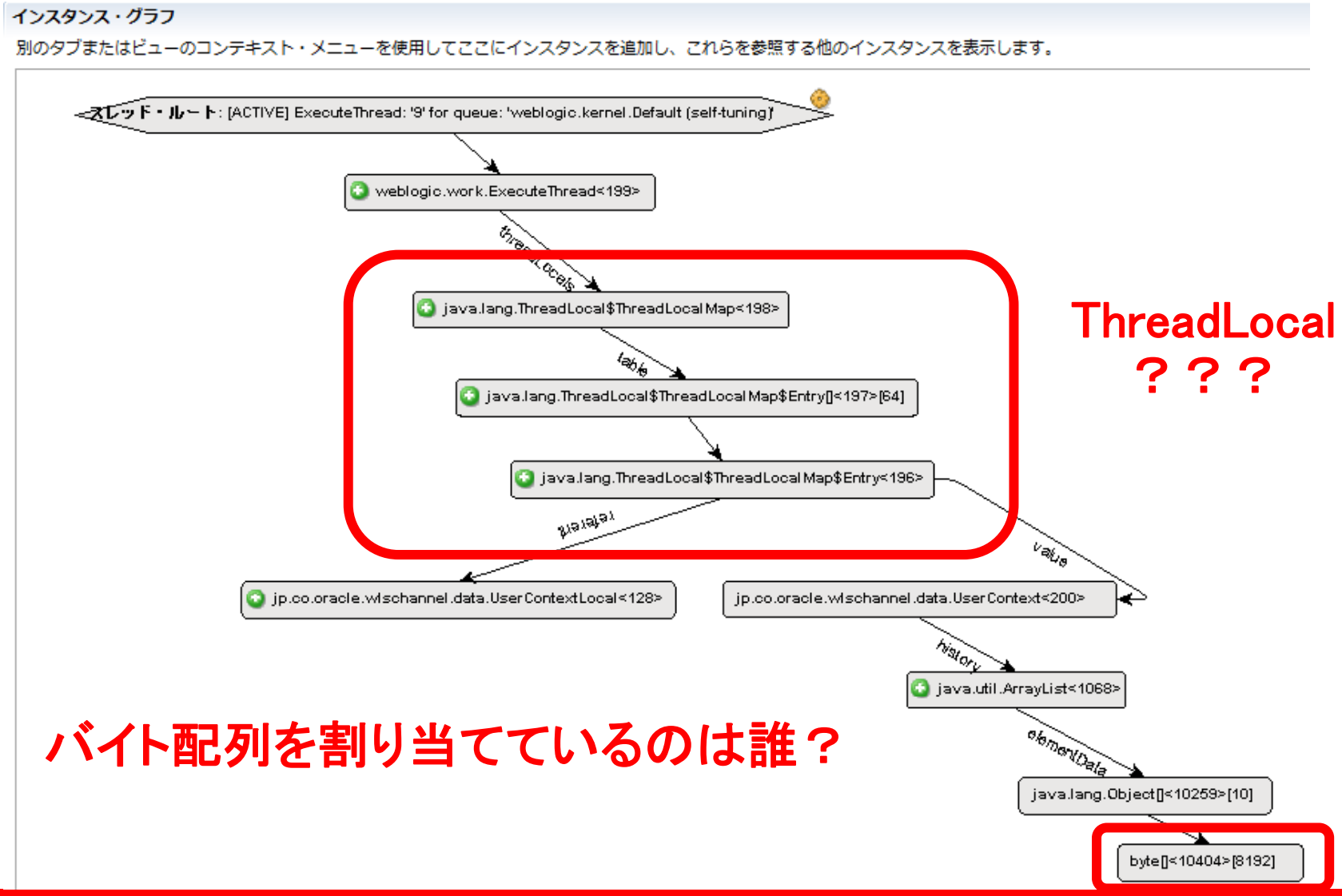
- byte[](バイト配列のクラス)のインスタンスがヒープの大部分を占めていることが分かる
  - しかし、バイト配列といった一般に多用されるクラスのインスタンスが多いという情報だけでは、原因の絞り込みは困難…

- そこで、バイト配列の割当元をさぐるために以下に注目
  - MemoryLeak Detectorのタイプグラフとインスタンスグラフ
  - JRockit Flight Recorderのメモリの割り当てトレース

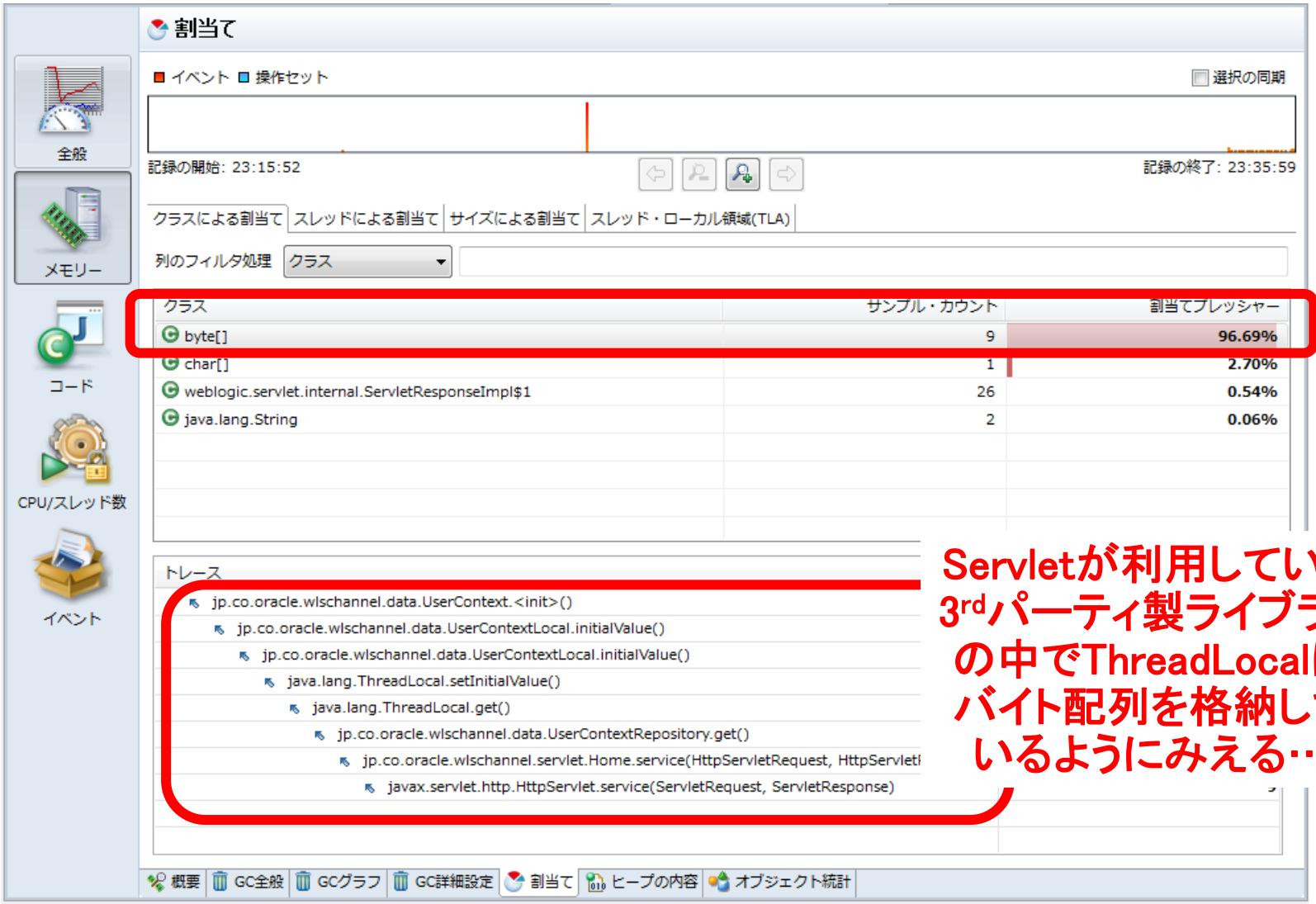
Memory Leak Detector タイプグラフ



Memory Leak Detector インスタンスグラフ



Flight Recorder メモリ割当てトレース



Servletが利用している3rdパーティ製ライブラリの中でThreadLocalにバイト配列を格納しているようにみえる…

【参考】java.lang.ThreadLocal

- スレッド毎に固有の値を保持するための入れ物
  - 同一スレッド実行される複数コンポーネント間で使い回すデータを格納するのに用いる
  - イメージ的にはスレッドIDをキー、任意オブジェクトを値にとるMapのようなもの
    - ThreadLocal#set(T value)で呼出元スレッド用の値を格納
    - ThreadLocal#get() で呼出元スレッド用の値を取得
      - 初回呼出時はinitialValue() メソッドで指定した値で初期化される

ThreadLocal –X

ThreadID	Value
Thread-1	X-1
Thread-2	X-2
Thread-3	X-3

ThreadLocal –Y

ThreadID	Value
Thread-1	Y-1
Thread-2	Y-2
Thread-3	Y-3

ThreadLocal– Z

ThreadID	Value
Thread-1	Z-1
Thread-2	Z-2
Thread-3	Z-3

- ただし、実際の実装ではスレッド毎にThreadLocalをキー、任意オブジェクトを値にとるMap

ThreadLocalMap for Thread–1

ThreadLocal	Value
ThreadLocal-X	X-1
ThreadLocal-Y	Y-1
ThreadLocal-Z	Z-1

ThreadLocalMap for Thread–2

ThreadLocal	Value
ThreadLocal-X	X-2
ThreadLocal-Y	Y-2
ThreadLocal-Z	Z-2

ThreadLocalMap for Thread–3

ThreadLocal	Value
ThreadLoca-lX	X-3
ThreadLocal-Y	Y-3
ThreadLocal-Z	Z-3

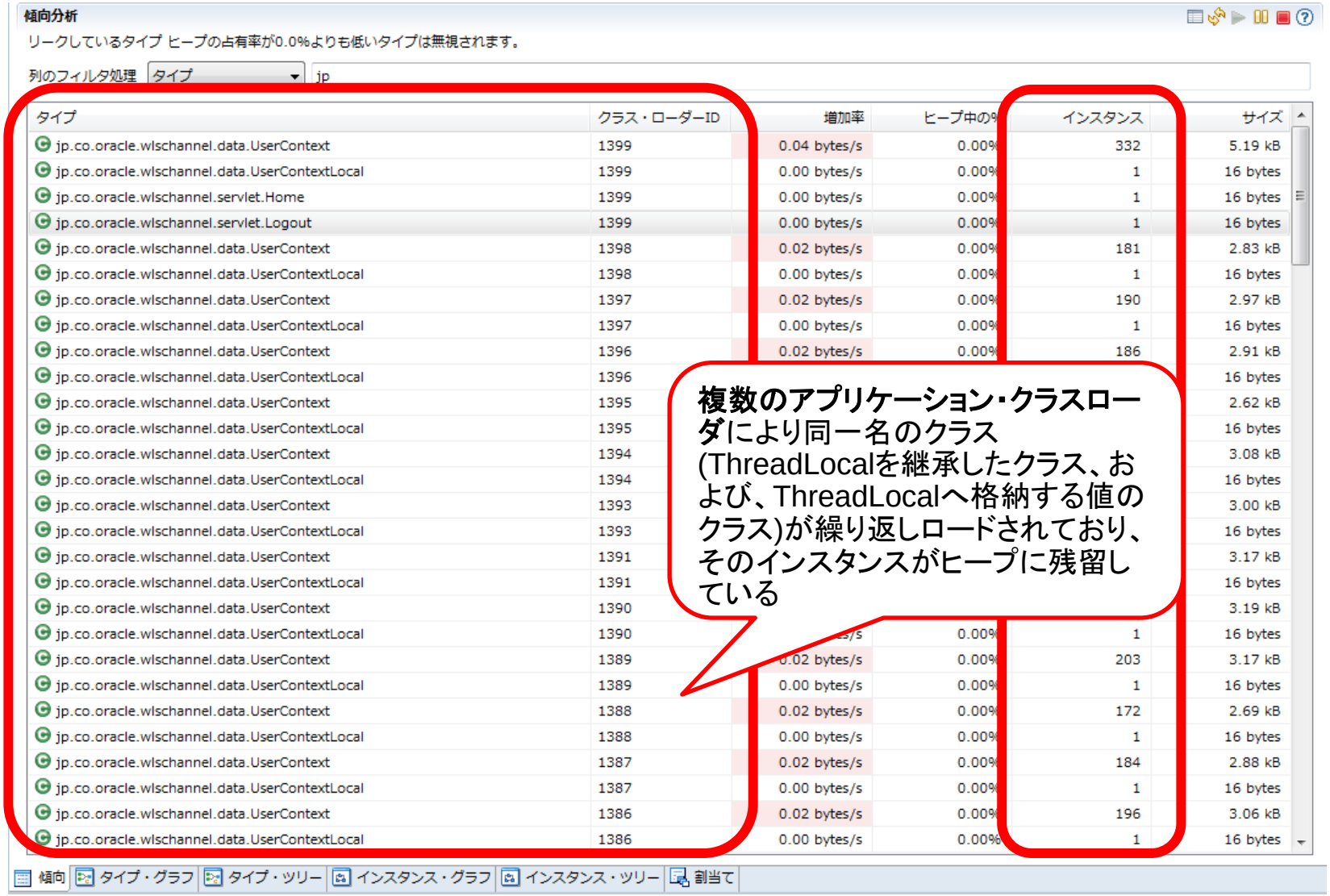
ここまでで分かったことと、さらなる追及

- サーブレットの中で呼ばれるサードパーティのライブラリの中でThreadLocal（を継承したクラス）にデータ(バイト配列)を格納しており、その容量がヒープの大部分を占めているようだ

- ただ、ThreadLocalへのデータの格納は、通常アプリがそのスレッドを初めて利用したときに行われるので、データ数はせいぜいWLSのワーカースレッド数程度にしか増えないはずである

- なぜOOMEになるほどにメモリリークしてしまうのか？

MemoryLeak Detector傾向（クラスローダ別表示）



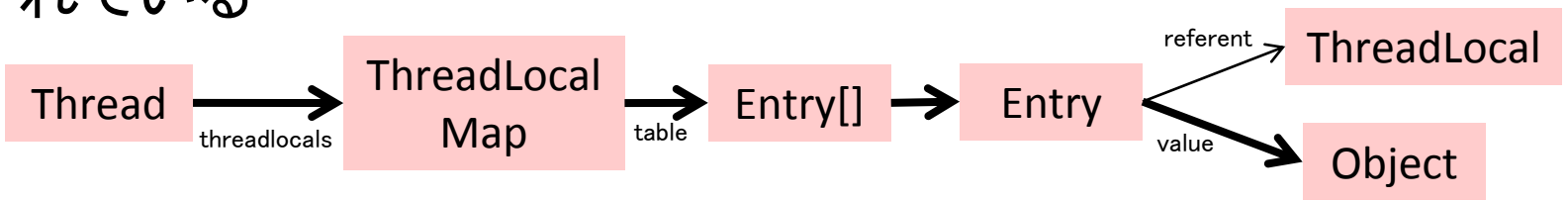
複数のアプリケーション・クラスローダにより同一名のクラス(ThreadLocalを継承したクラス、および、ThreadLocalへ格納する値のクラス)が繰り返しロードされており、そのインスタンスがヒープに残留している

ここまでで分かったことと、さらなる追及

- Memory Leak Detectorによると複数のアプリケーション・クラスローダから同一名のThreadLocalのサブクラス、および、格納値クラスが繰り返しロードされており、そのインスタンスがヒープに居続けている
- このシステムではプロダクション再デプロイメントを用いて、頻繁にアプリケーションのバージョンアップをしているので、アプリケーションのクラスローダが複数出現するのは当然だが、WebLogic Serverのプロダクション再デプロイメントでは最大2バージョンしか同時稼働できないはず
- ヒープ上にはアンデプロイ(自動リタイア)しているはずの2バージョン以上古いアプリケーションのオブジェクトが残留している・・・、どうしてなのか？

アンデプロイされてもThreadLocal/格納値が開放されない理由

- ThreadLocalに格納された値は設定元のThreadから強参照されている

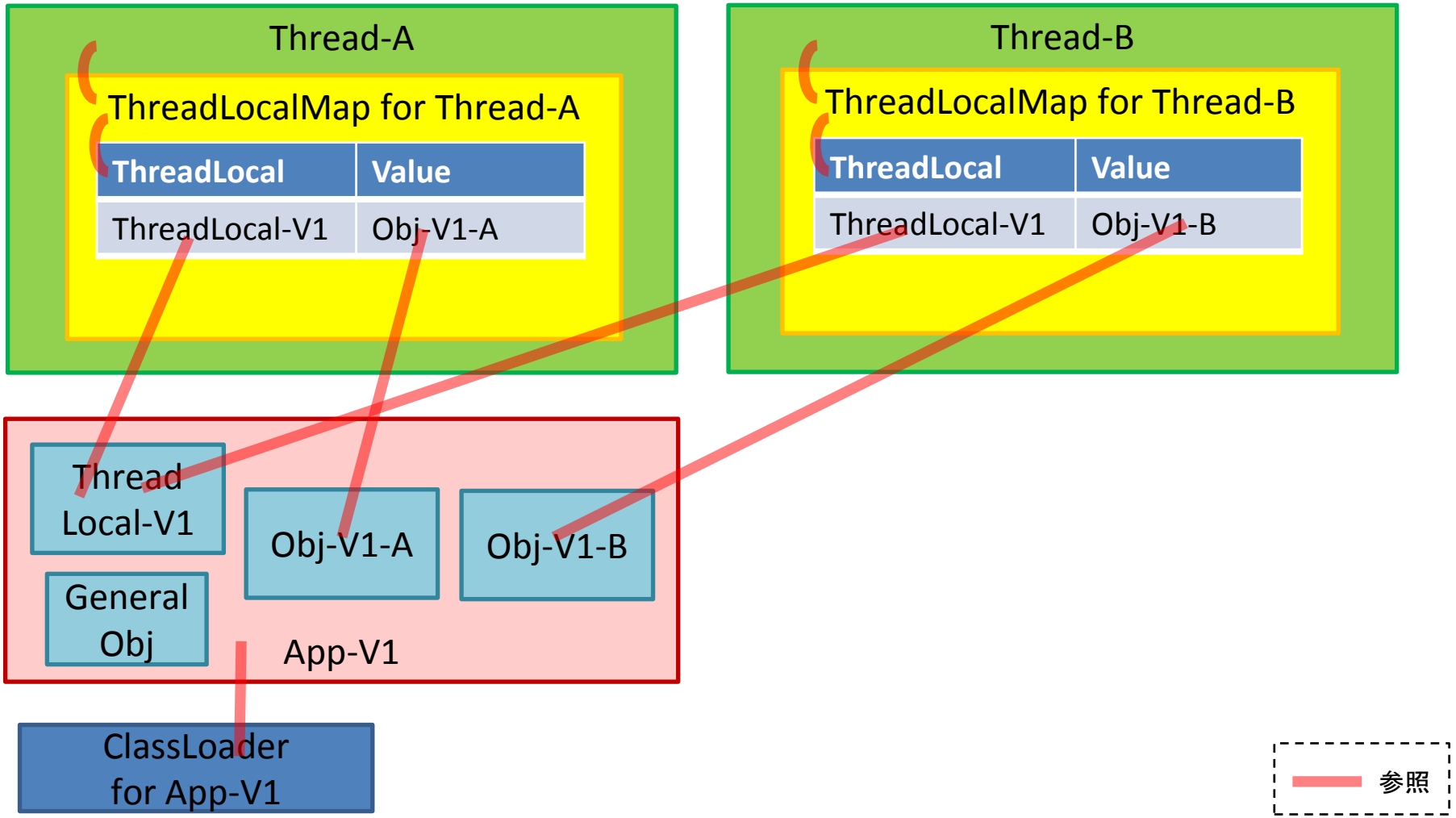


- ThreadLocalへ格納された値は、明示的にその値またはThreadLocal自身を破棄しない限りは、格納元のスレッドが消滅するまでは、アプリをアンデプロイしてもGC対象にならない（WebLogicはスレッドプールを利用するため、通常、一度生成された実行スレッドはWebLogic停止まで無くならない）
- 格納値を破棄できなければ、その値クラスをロードしたクラスローダ(アプリケーション・クラスローダ)も破棄できず、アンデプロイは見掛け上は成功していても、内部的にはゴミが残る

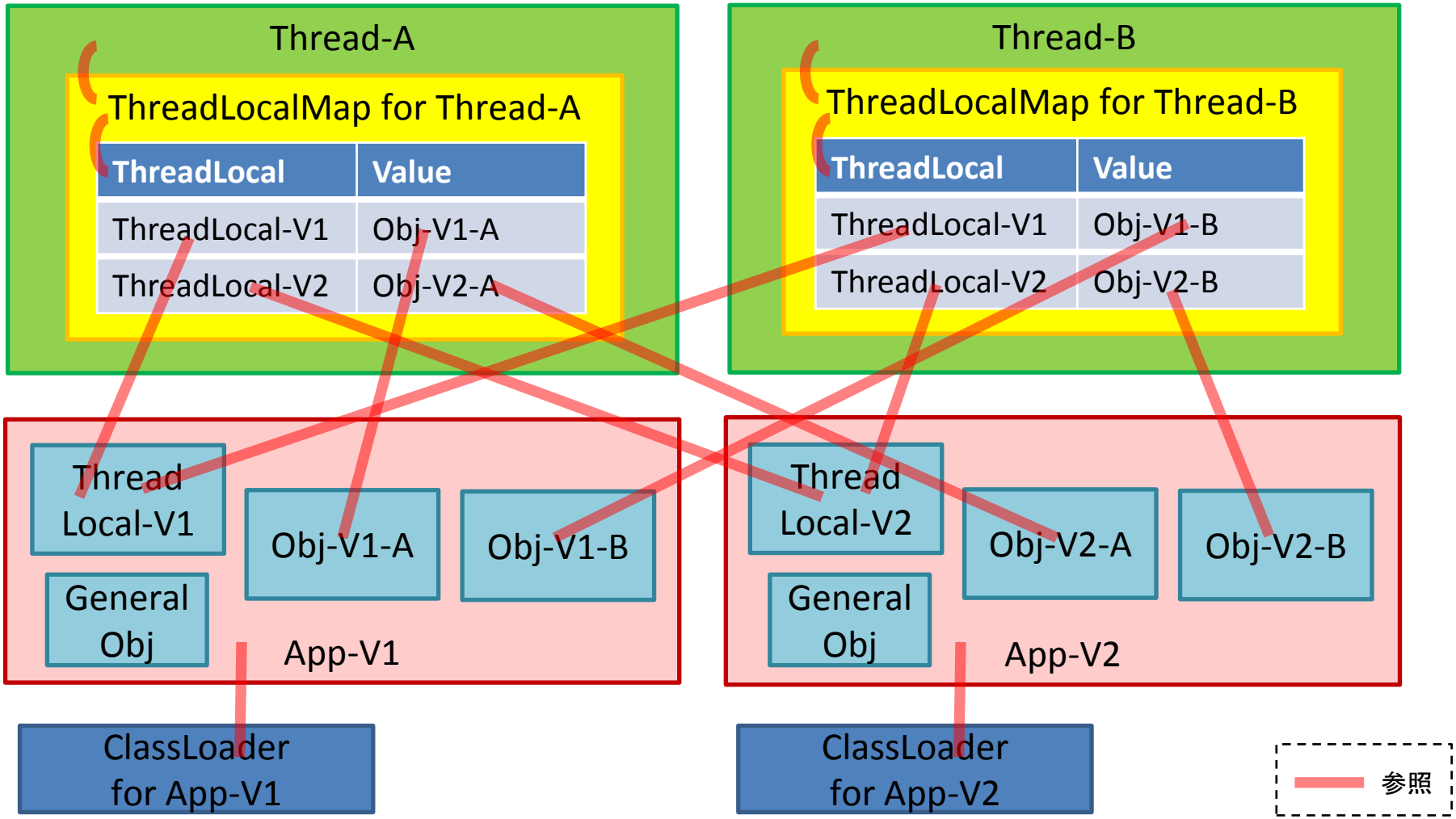
メモリリーク原因まとめ

- プロダクション再デプロイメントでは同時には最大2バージョンのアプリしか稼働できないので、通常ならばリタイアしたバージョンのアプリから生成されたオブジェクトはヒープから次々と消えてゆく
- 今回のアプリではサードパーティのライブラリを経由してThreadLocalを利用していた
- WebLogicのように永続的なスレッドプールを用いるアプリケーションサーバ上でThreadLocalを利用していて、かつ、ThreadLocal/格納値を明示的に破棄していない場合は、旧バージョンアプリのリタイア後もThreadLocalに旧バージョンアプリで生成されたオブジェクトが残ってしまう（通常のアンデプロイでも同様）

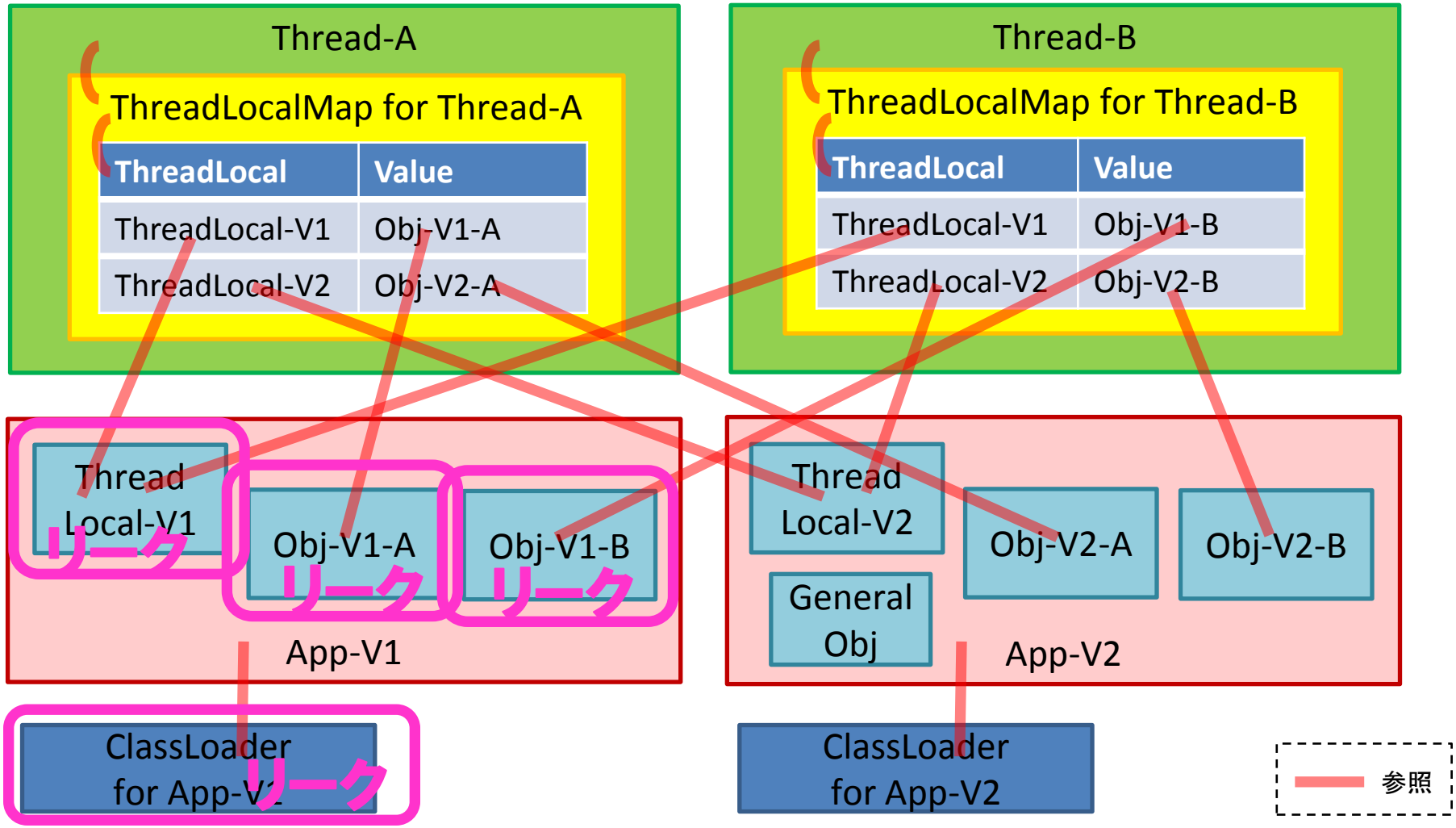
プロダクション再デプロイ前



プロダクション再デプロイ後、新旧両バージョン同時稼働時

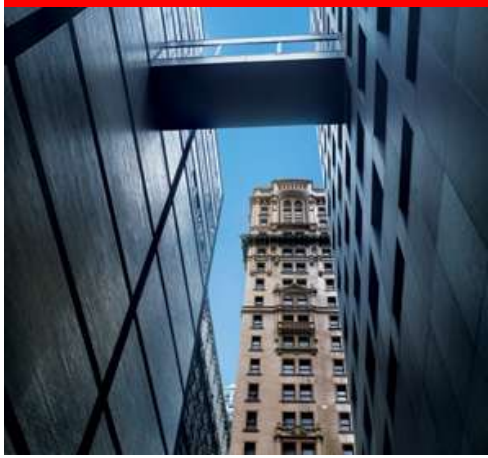


旧バージョンリタイア後



対策

- アプリケーションサーバへデプロイするアプリケーションではThreadLocalを利用しない
  - WebLogicでもThreadLocalの利用は非推奨（KROWN 134678）
- ThreadLocalを使わざるを得ない場合は、アプリケーション側で責任を持って削除
  - スレッドがスレッドプールに戻る前にThreadLocal#remove() でThreadLocalに格納されている値を削除
  - さらにThreadLocalのインスタンス自身がGC対象になるように、ThreadLocalへの全ての参照変数にnullをセット
- 3rdPartyのライブラリの内部でThreadLocalが利用されている場合は、リフレクション等を用いて削除するしかない
  - 一部、自動削除に対応しているアプリケーションサーバもある



性能トラブル事例3(C社)

C社システムで発覚した性能問題

システム概要	•WebLogic Serverを利用して24時間オンラインサービスを提供中 •将来のトラフィック増に備えて、マシンを4コアマシン(1ソケット×4コア)から12コア(2ソケット×6コア)マシンへアップグレードした
トラブル状況	•性能がむしろ悪化してしまった •4コアマシンでは秒間3500件処理できていた •12コアマシンにしたところ秒間2000件しか処理できず

お客様が事前に実施したこと

- WebLogic Serverの実行スレッド数を変えてみる
- JVMのヒープサイズを変えてみる
- GCのモードを変えてみる

⇒しかし、いずれも効果なし・・・

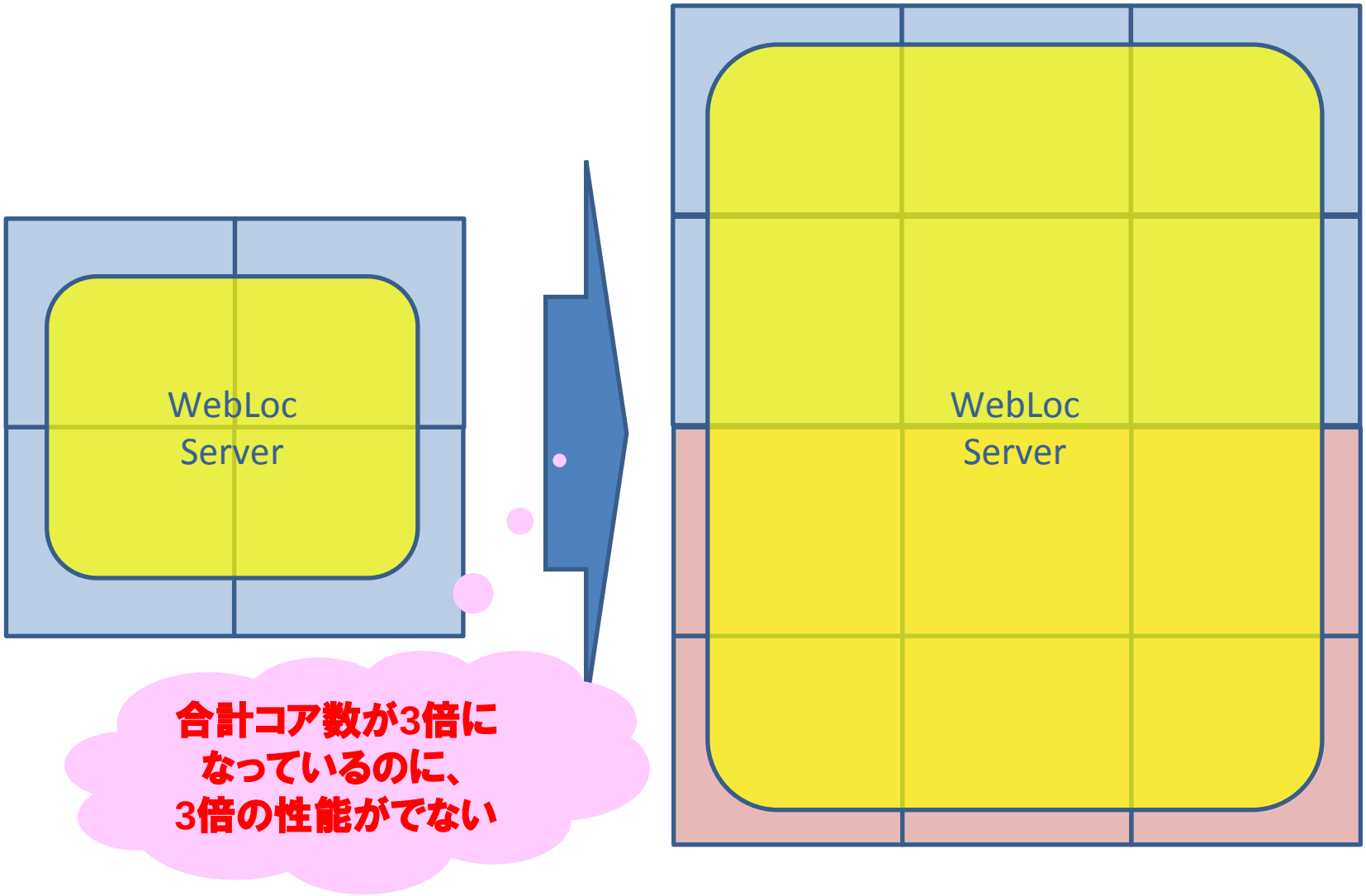
- OSベンダーへの問い合わせ
- HWベンダーへの問い合わせ

⇒有効な回答は得られず・・・

その後の解決までの流れ

- システム構成概要のヒアリング
- JRockit Flight Recorderの取得と解析
- 改善策の提案
- 改善策の実施

新・旧構成比較



JFRによるJavaロック・プロファイリング

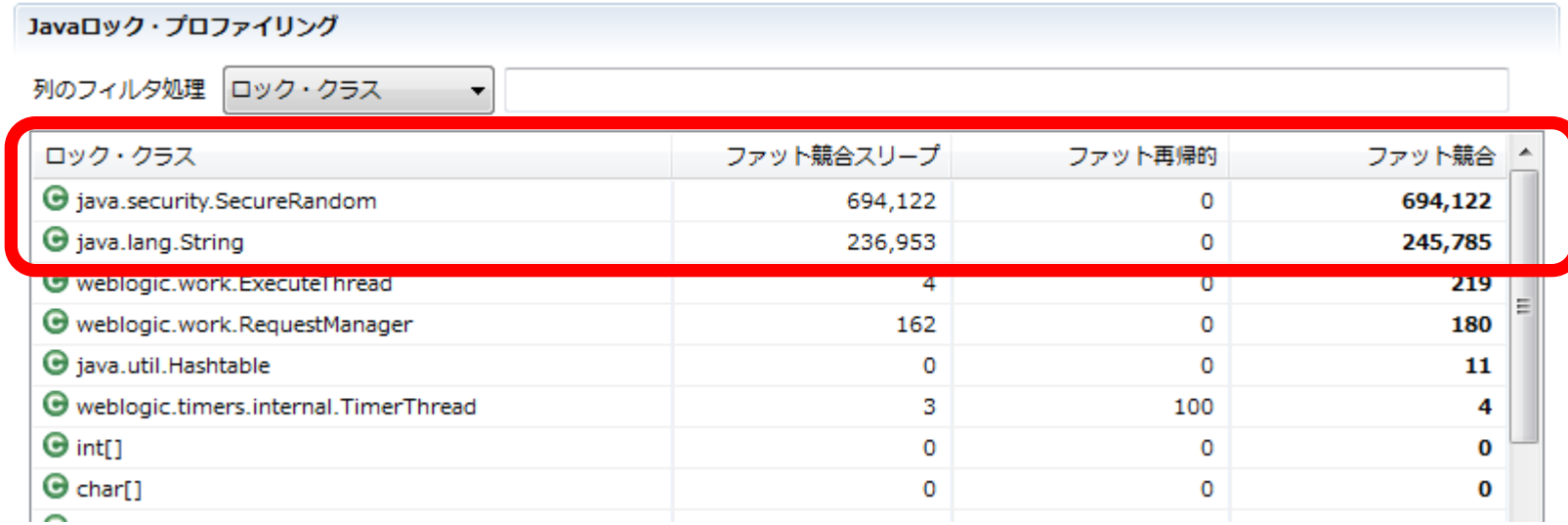
4コア
1JVM

Javaロック・プロファイリング

列のフィルタ処理 ロック・クラス

ロック・クラス	ファット競合スリープ	ファット再帰的	ファット競合
java.lang.String	17,513	0	**17,912**
java.security.SecureRandom	11,798	0	**11,799**
weblogic.work.ExecuteThread	51	0	**369**
weblogic.timers.internal.TimerThread	7	90	**8**
int[]	0	0	**0**
char[]	0	0	**0**

12コア
1JVM



Javaロック・プロファイリング

列のフィルタ処理 ロック・クラス

ロック・クラス	ファット競合スリープ	ファット再帰的	ファット競合
java.security.SecureRandom	694,122	0	**694,122**
java.lang.String	236,953	0	**245,785**
weblogic.work.ExecuteThread	4	0	**219**
weblogic.work.RequestManager	162	0	**180**
java.util.Hashtable	0	0	**11**
weblogic.timers.internal.TimerThread	3	100	**4**
int[]	0	0	**0**
char[]	0	0	**0**

Flight Recorderから分かったこと

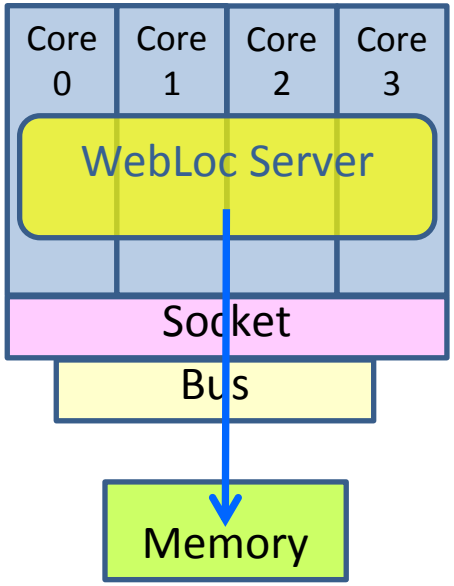
- 4コアマシン利用時に比べて、12コアマシン利用時に**ファット・ロックの発生頻度が約30倍に上昇**している

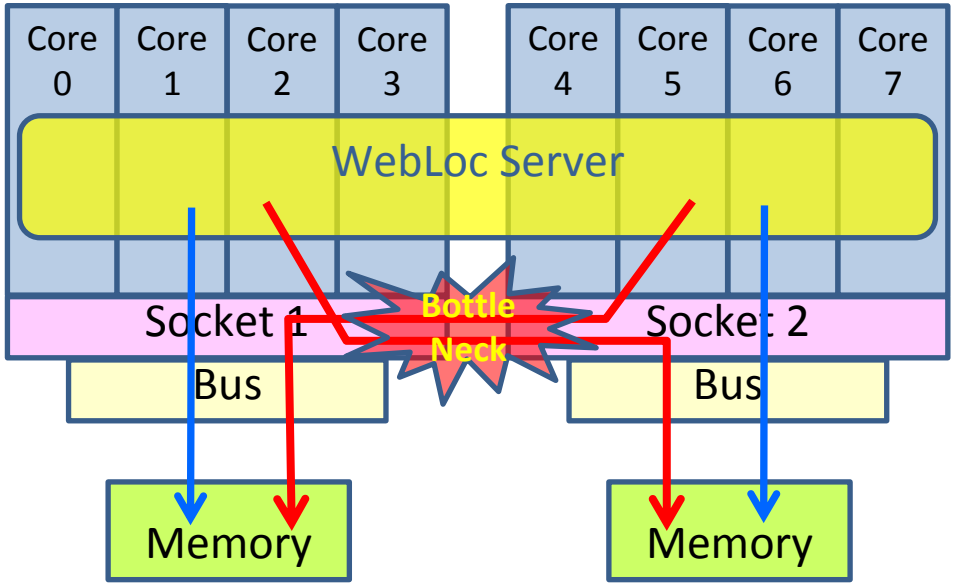
ロックについて

- 複数スレッドを用いてパラレルに実行するタスク内に、同時には1スレッドでしか実行してはいけないセクション(クリティカルセクション)がある場合は、排他と同期が必要になる。
- 排他と同期は通常ロックを用いて実装される。ロックを獲得したスレッドのみがクリティカルセクションを実行できる。
- ロックを獲得する際に、既に別のスレッドが獲得している場合は、解放されるまで待つ。
  - 競合が少ない場合は、Spinして解放を待つ(シン・ロック)
  - 競合が多く、しばらくSpinしても解放されない場合は、Sleepして解放を待つ(ファット・ロック)

マルチソケットマシンでの非対象メモリアクセス（NUMAアーキテクチャ）

- 従来のシングルソケットのマルチコアマシンとは異なり、最近のマルチソケットのマルチコアマシンでは、ソケットに閉じたメモリアクセスの速度に比べて、ソケットを跨るメモリアクセスの速度が遅い。ソケットを跨るメモリアクセスはボトルネックになりやすい。





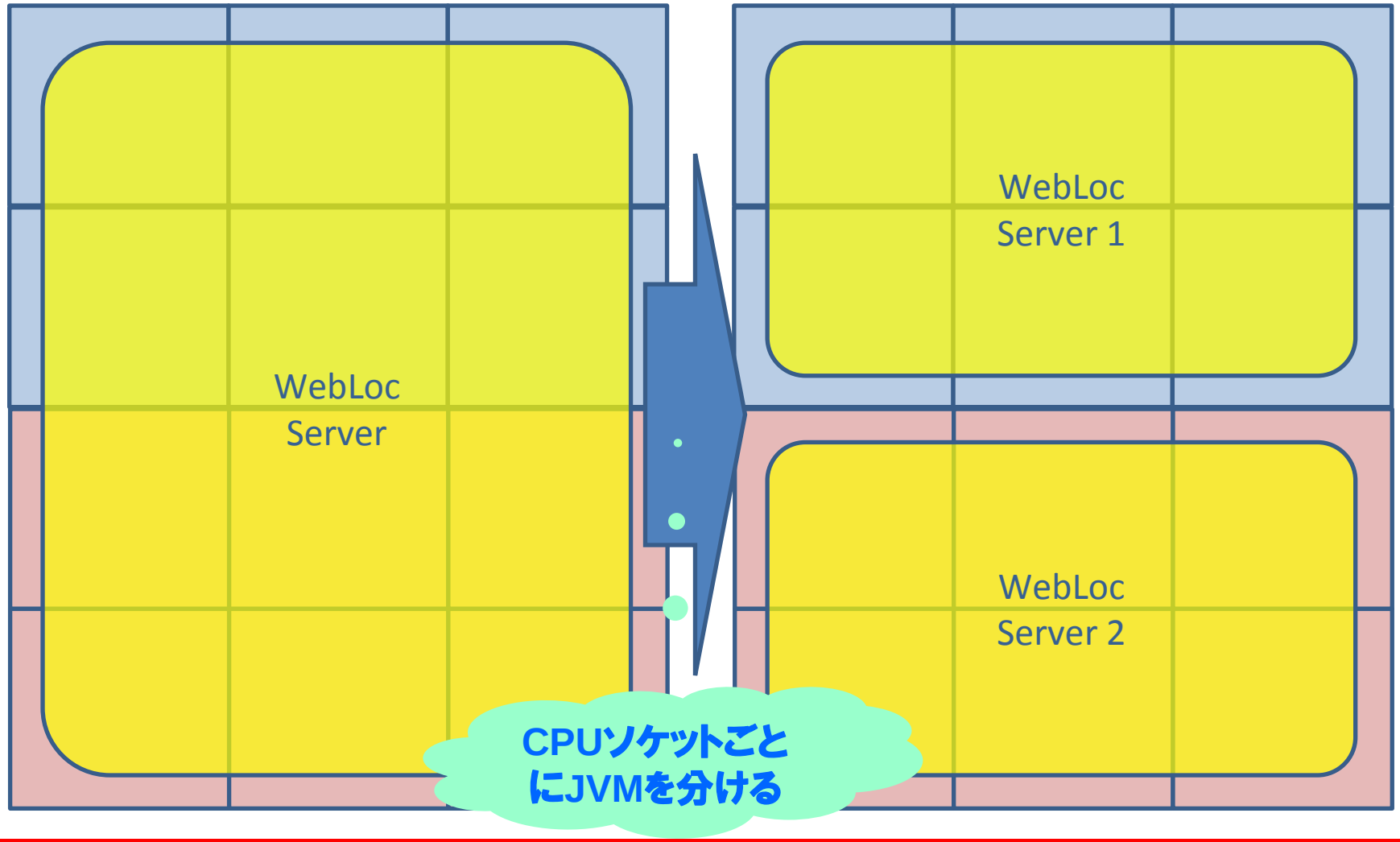
マルチソケットマシンでのロックの競合

- ロック・オブジェクトはいずれかのソケット下のメインメモリに属する。
- 異なるソケットに属する複数のスレッドからロック獲得（ロック・オブジェクトへのロックフラグを立てること）要求が同時に発生すると、ソケット間でのCPUキャッシュの転送が頻繁に生じてしまう。
  - Oracle RACで、異なるインスタンスから同一データブロックへの更新要求が多発すると、インスタンス間のデータブロック転送（キャッシュフュージョン）が多発して性能劣化するのと似ている
- それゆえ、ソケットをまたぐロックの獲得には時間がかかるため、Thin LockからFat Lockにエスカレーションしてしまうことが多くなる

実施した対策（JVMの多重化）

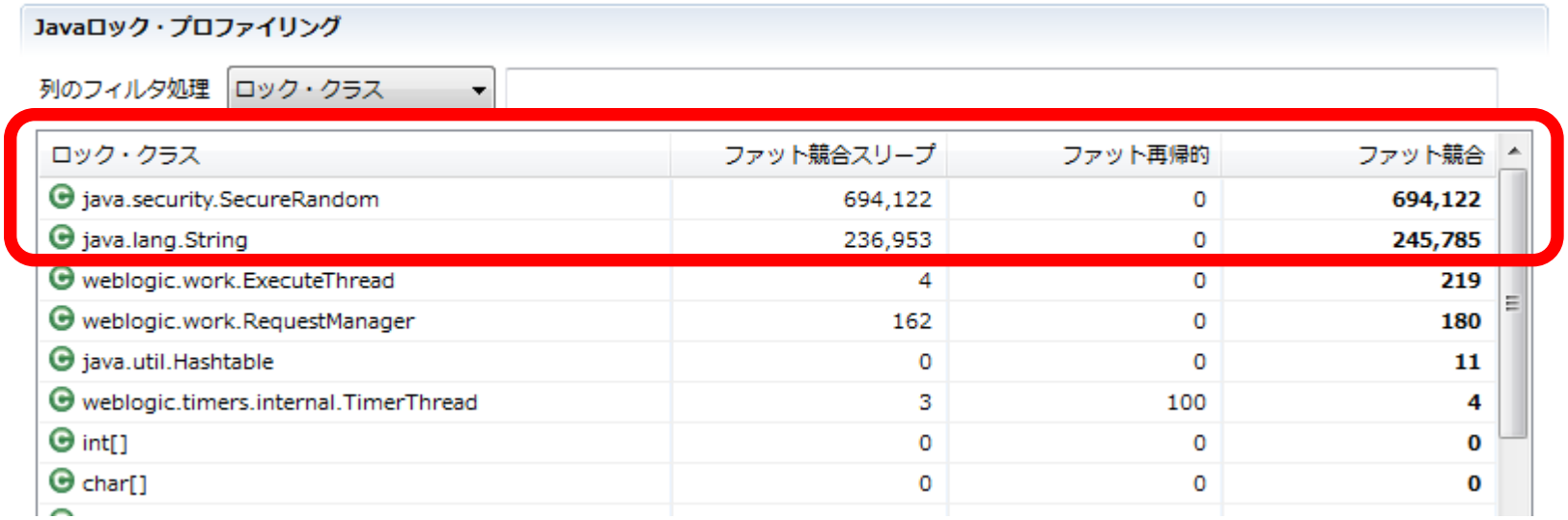
- 同一マシン内で複数JVM化
- 各JVMが使用するリソースを特定ソケットに括りつける
  JRockitのパラメータを設定
  - For WLS1
    - -XX:BindToCPUs=0,1,2,3,4,5
    - -XX:NumaMemoryPolicy=strictlocal
    - -XXgcThreads:6
  - For WLS2
    - -XX:BindToCPUs=6,7,8,9,10,11
    - -XX:NumaMemoryPolicy=strictlocal
    - -XXgcThreads:6

実施した対策（JVMの多重化）



対策実施前後のJFR

12コア
1JVM



Javaロック・プロファイリング

列のフィルタ処理 ロック・クラス

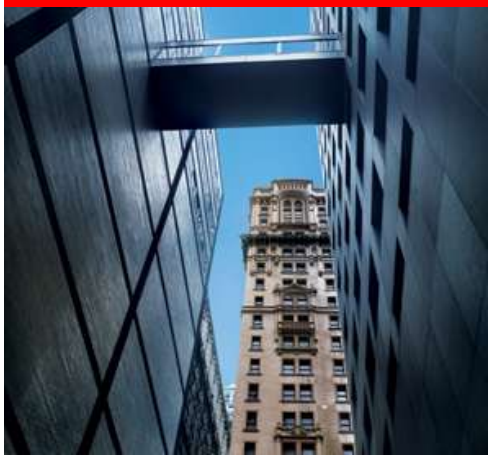
ロック・クラス	ファット競合スリープ	ファット再帰的	ファット競合
java.security.SecureRandom	694,122	0	**694,122**
java.lang.String	236,953	0	**245,785**
weblogic.work.ExecuteThread	4	0	**219**
weblogic.work.RequestManager	162	0	**180**
java.util.Hashtable	0	0	**11**
weblogic.timers.internal.TimerThread	3	100	**4**
int[]	0	0	**0**
char[]	0	0	**0**

12コア
2JVM

Javaロック・プロファイリング

列のフィルタ処理 ロック・クラス

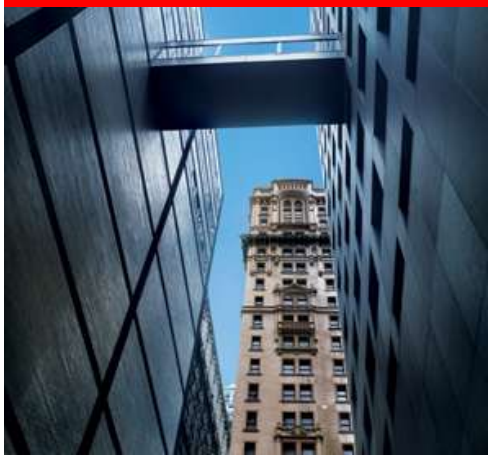
ロック・クラス	ファット競合スリープ	ファット再帰的	ファット競合
java.security.SecureRandom	148,118	0	**148,159**
java.lang.String	63,448	0	**64,981**
weblogic.work.ExecuteThread	8	0	**127**
int[]	0	0	**0**
char[]	0	0	**0**



まとめ

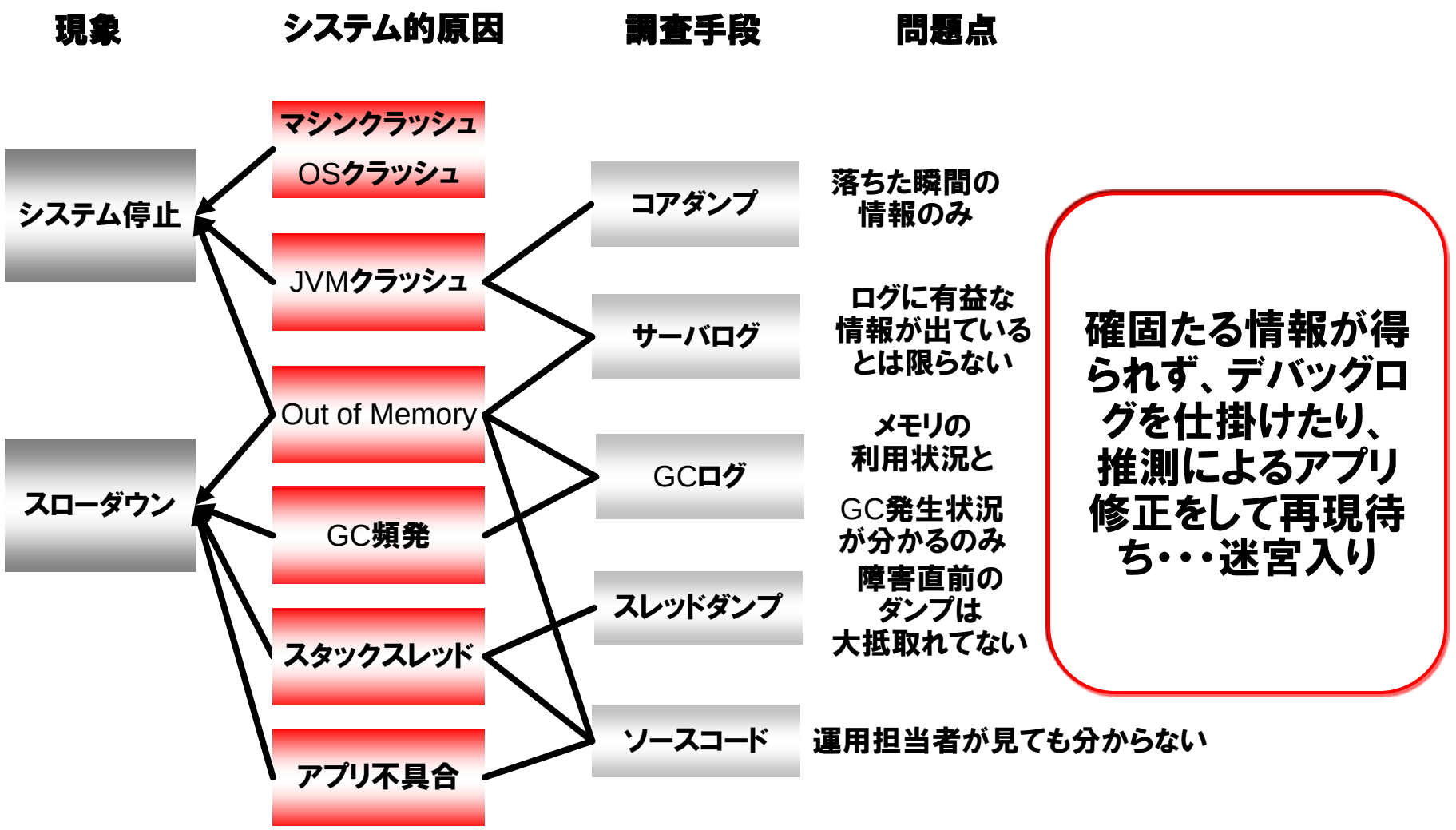
性能トラブルへの備え

- Javaの性能トラブルへの対処方法をパターン化できればいいが、なかなか難しい
- 過去に遡って稼働状況を正確に収集するための汎用的な道具を用意しておくことが大事
- JRockit Flight RecorderではJVM内の挙動を低オーバーヘッドで常時記録し、過去に遡って調査ができるので、さまざまな性能トラブルの原因の早期発見に役立つ

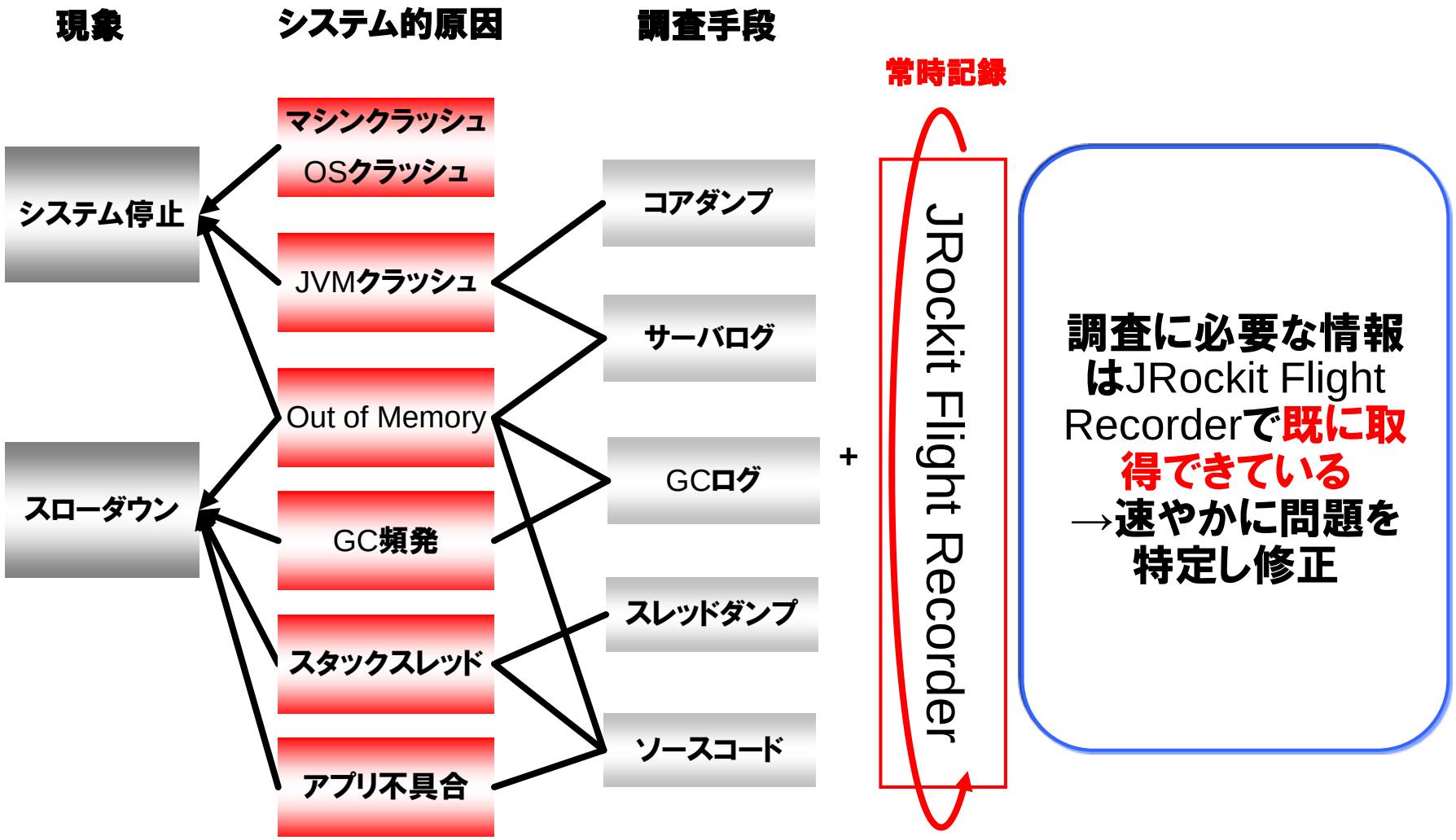


Appendix
JRockit Flight Recorder 概要

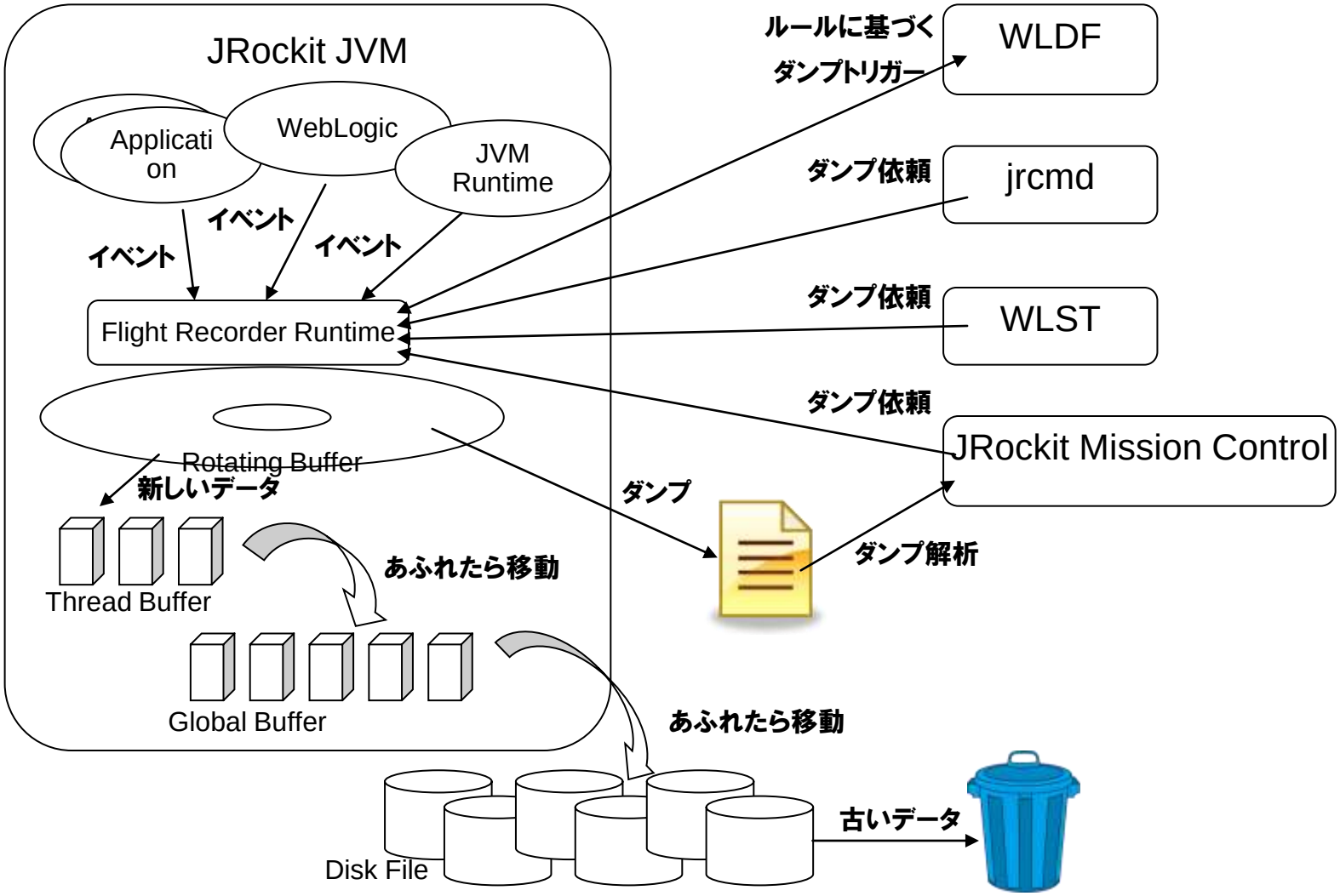
一般的な障害パターンと調査方法



障害パターンと調査手段（JRockit Flight Recorderを使った場合）



JRockit Flight Recorder のアーキテクチャ



Flight Recorderを仕掛ける方法

- JRockit起動時のオプションで
  - -XX:FlightRecorderOptions=disk=true
    -XX:StartFlightRecording=filename=<出力ファイル名>
    ,settings=<テンプレートファイル名>
    ,maxage=<ディスク上に保持する期間>
- JRockit上でWebLogic（10.3.3以降）を利用する場合は、WebLogicの「診断ボリューム」の設定でも可能
  - 「診断ボリューム」をOff/Low/Medium/Highのいずれかに変更
- JRockit起動後に外からコマンドで
  - jrcmd <jrockit_pid> start_flight_recording settings=<テンプレートファイル名>
- JRockit起動後にJRockit Mission ControlのGUIから
  - JVMブラウザ→「フライト記録の開始…」→テンプレート選択

ダンプを取得する方法

- JRockit上でWebLogic（10.3.3以降）を利用する場合は、WLDF（WebLogic診断フレームワーク）で定義した監視ルールが満たされた際に自動的にダンプを取得することが可能
  - エラー発生時
  - レスポンスタイム超過時

- コマンドで手動でダンプ
  - jrcmd <jrockit pid> dump_flightrecording
    id=<レコーディングid>
    copy_to_file=<出力ファイル

- JRockit Mission ControlのGUIからダンプ
  - フライト・レコーダ・コントロール→「ダンプ…」→ダンプする時間範囲を指定
  - JRMCのトリガー機能でルールを設定して自動でダンプすることも可能

OTNセミナーオンデマンド

コンテンツに対する
ご意見・ご感想を是非お寄せください。

OTNオンデマンド　感想

http://blogs.oracle.com/oracle4engineer/entry/otn_ondemand_questionnaire

上記に簡単なアンケート入力フォームをご用意しております。

セミナー講師/資料作成者にフィードバックし、
コンテンツのより一層の改善に役立てさせていただきます。

是非ご協力をよろしくお願いいたします。

OTNセミナーオンデマンド

日本オラクルのエンジニアが作成したセミナー資料・動画ダウンロードサイト

ORACLE TECHNOLOGY NETWORK
OTNセミナー オンデマンド

掲載コンテンツカテゴリ(一部抜粋)
Database 基礎
Database 現場テクニック
Database スペシャリストが語る
Java
WebLogic Server/アプリケーション・グリッド
EPM/BI 技術情報
サーバー
ストレージ

100以上のコンテンツをログイン不要でダウンロードし放題

データベースからハードウェアまで充実のラインナップ

毎月、旬なトピックの新作コンテンツが続々登場

例えばこんな使い方

- 製品概要を効率的につかむ
- 基礎を体系的に学ぶ/学ばせる
- 時間や場所を選ばず（オンデマンド）に受講
- スマートフォンで通勤中にも受講可能

毎月チェック！

コンテンツ一覧 はこちら
http://www.oracle.com/technetwork/jp/ondemand/index.html

新作＆おすすめコンテンツ情報 はこちら
http://oracletech.jp/seminar/recommended/000073.html

OTNオンデマンド

ORACLE

オラクルエンジニア通信

オラクル製品に関わるエンジニアの方のための技術情報サイト

技術資料 — インストールガイド・設定チュートリアルetc. 欲しい資料への最短ルート

アクセスランキング — 他のエンジニアは何を見ているのか？人気資料のランキングは毎月更新

特集テーマ Pick UP — 性能管理やチューニングなど月間テーマを掘り下げて詳細にご説明

技術コラム — SQLスクリプト、索引メンテナンスetc. 当たり前の運用/機能が見違える!?

http://blogs.oracle.com/oracle4engineer/

オラクルエンジニア通信

oracletech.jp

ITエンジニアの皆様に向けて旬な情報を楽しくお届け

製品/技術情報
Oracle Databaseっていくら？オプション機能も見積れる簡単ツールが大活躍

セミナー
基礎から最新技術までお勧めセミナーで自分にあった学習方法が見つかる

スキルアップ
ORACLE MASTER！試験頻出分野の模擬問題と解説を好評連載中

Viva! Developer
全国で活躍しているエンジニアにスポットライト。きらりと輝くスキルと視点を盗もう

http://oracletech.jp/

oracletech

あなたにいちばん近いオラクル

Oracle Direct

まずはお問合せください

Oracle Direct

システムの検討・構築から運用まで、ITプロジェクト全般の相談窓口としてご支援いたします。
ステム構成やライセンス/購入方法などお気軽にお問い合わせ下さい。

Web問い合わせフォーム

専用お問い合わせフォームにてご相談内容を承ります。
http://www.oracle.co.jp/inq_pl/INQUIRY/quest?rid=28

※フォームの入力にはログインが必要となります。
※こちらから詳細確認のお電話を差し上げる場合がありますので
ご登録の連絡先が最新のものになっているかご確認下さい。

フリーダイヤル

0120－155－096

※月曜～金曜
9:00～12:00、13:00～18:00
（祝日および年末年始除く）

ORACLE

Hardware and Software
Engineered to Work Together


ORACLE®