HITACHI
Inspire the Next

SOA運用で差をつけよう!
Webシステムのトラブル解決ノウハウ

2008年11月18日

SAMURAIZ
株式会社サムライズ
技術グループ 部長　プロダクトコンサルタント
富田　誠

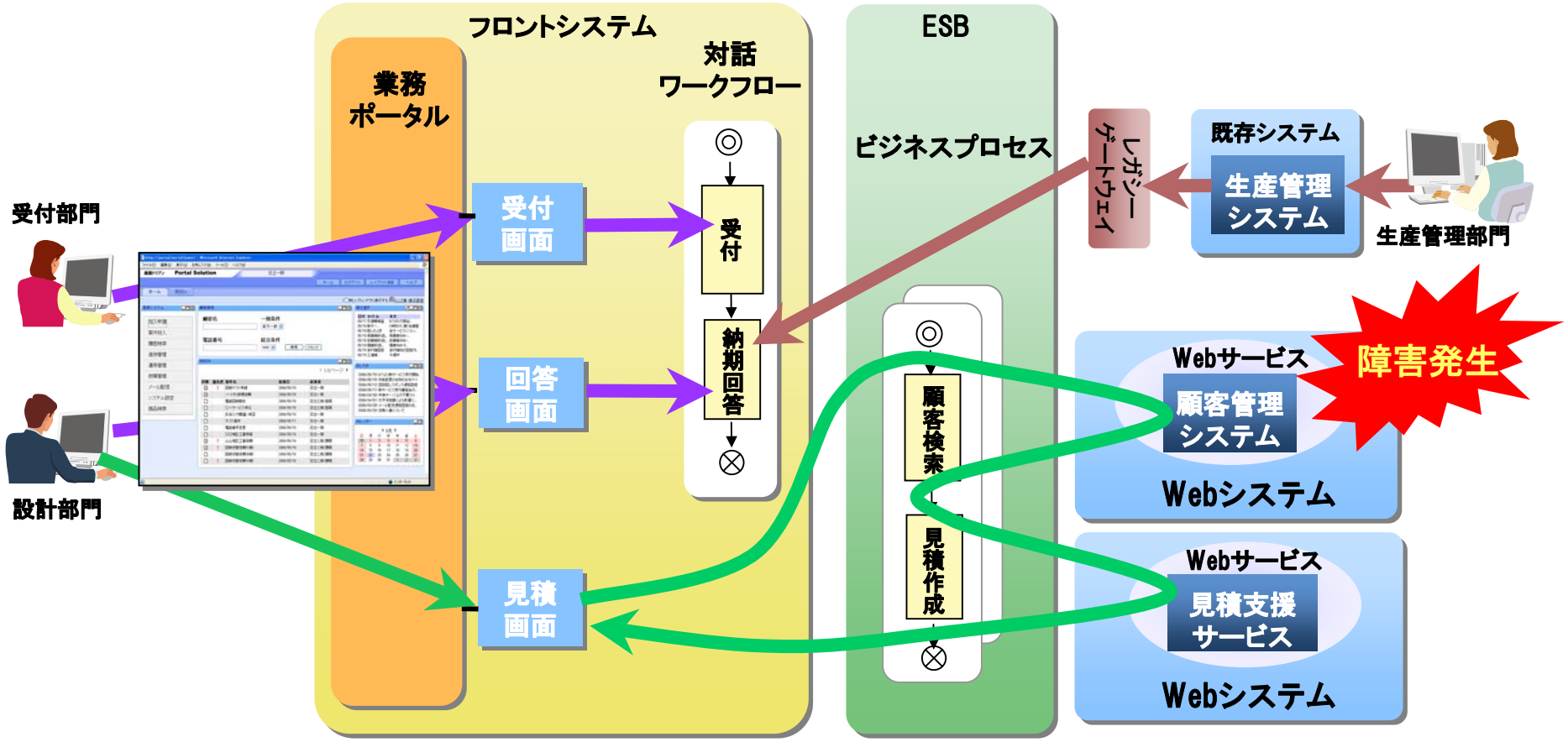
HITACHI
株式会社日立製作所　ソフトウェア事業部
第2AP基盤ソフト設計部　部長
齋場　正弘

uVALUE

HITACHI
Inspire the Next

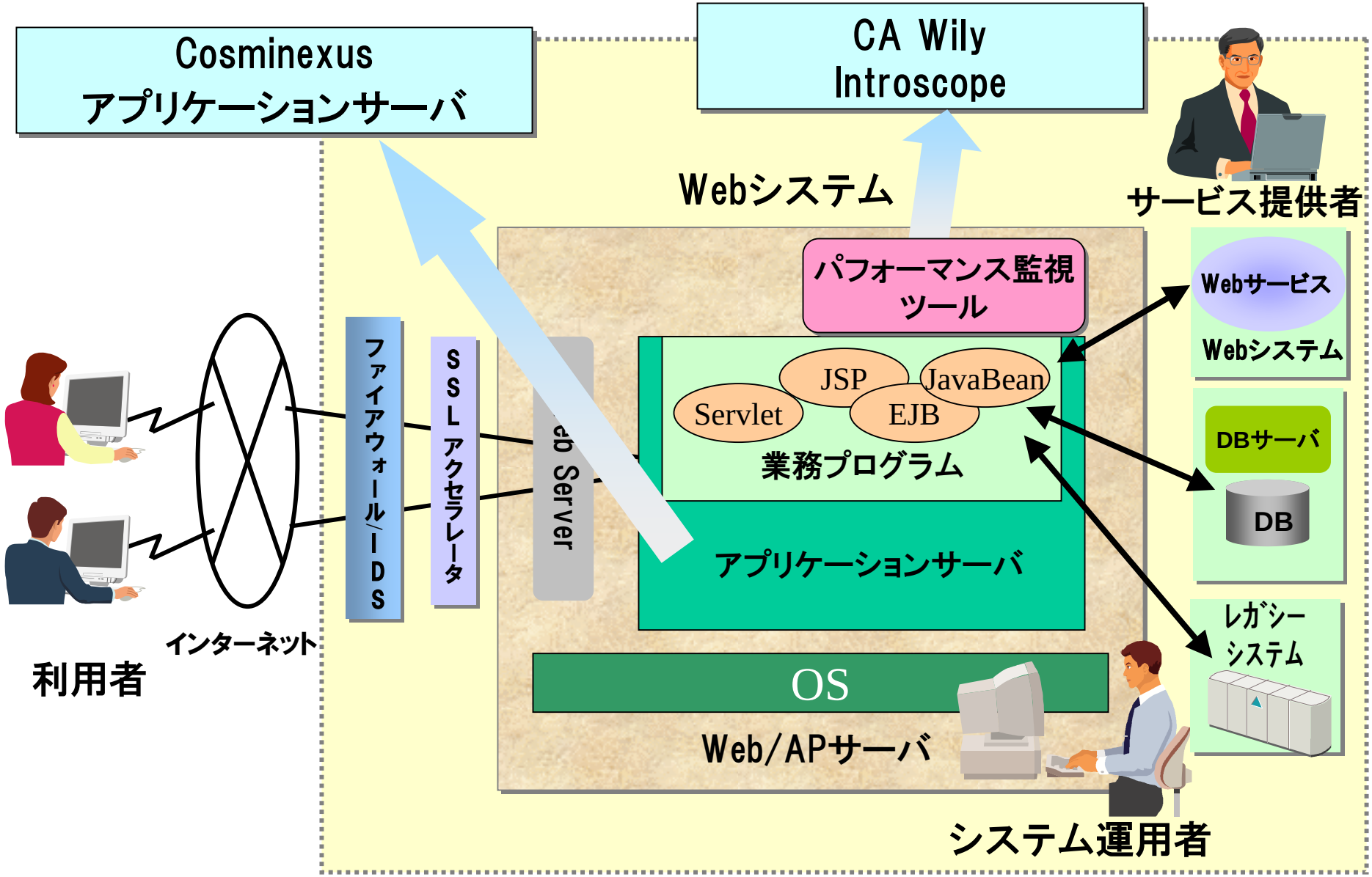
SOAによるシステム統合で求められるもの

Webシステムでの障害検知をはじめとして
障害を誘発する要因の排除、障害回復の迅速化が求められる。



HITACHI
Inspire the Next

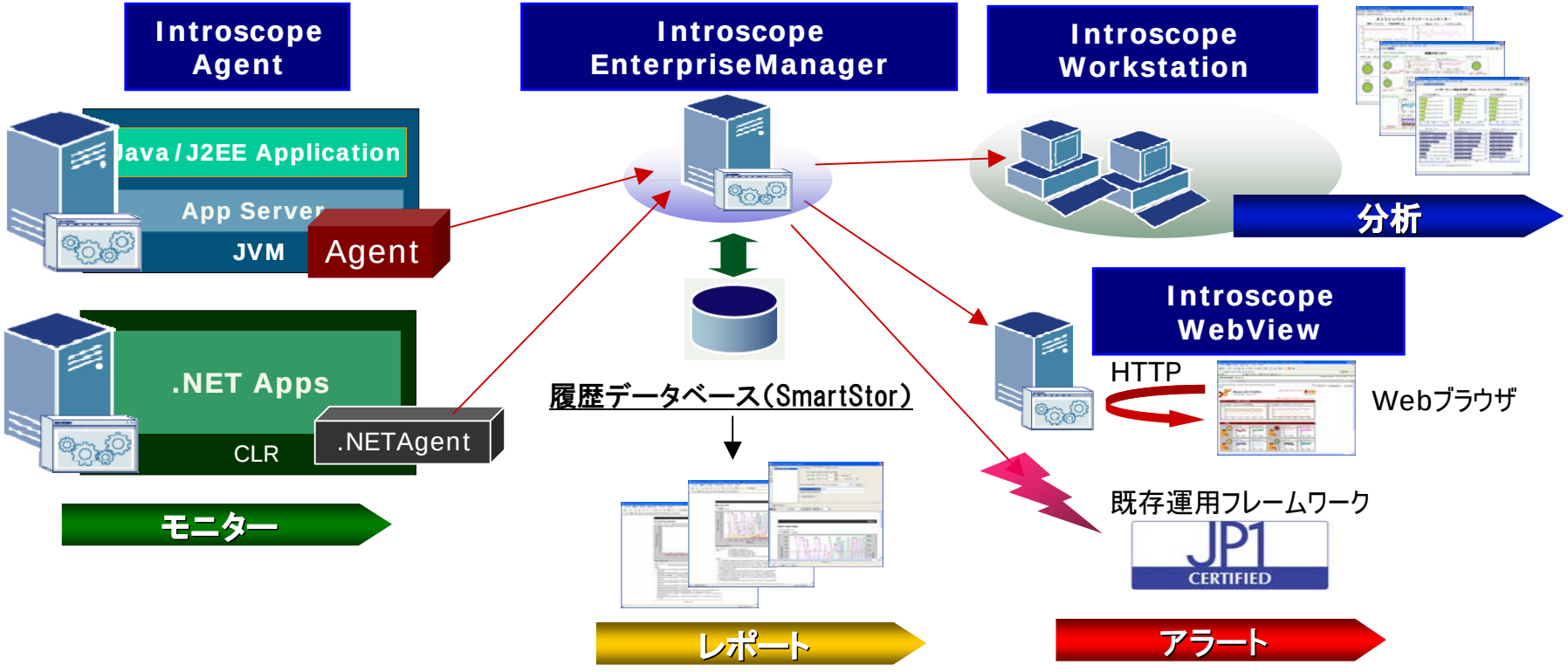
Webシステムの構成



HITACHI
Inspire the Next

CA Wily Introscopeとは

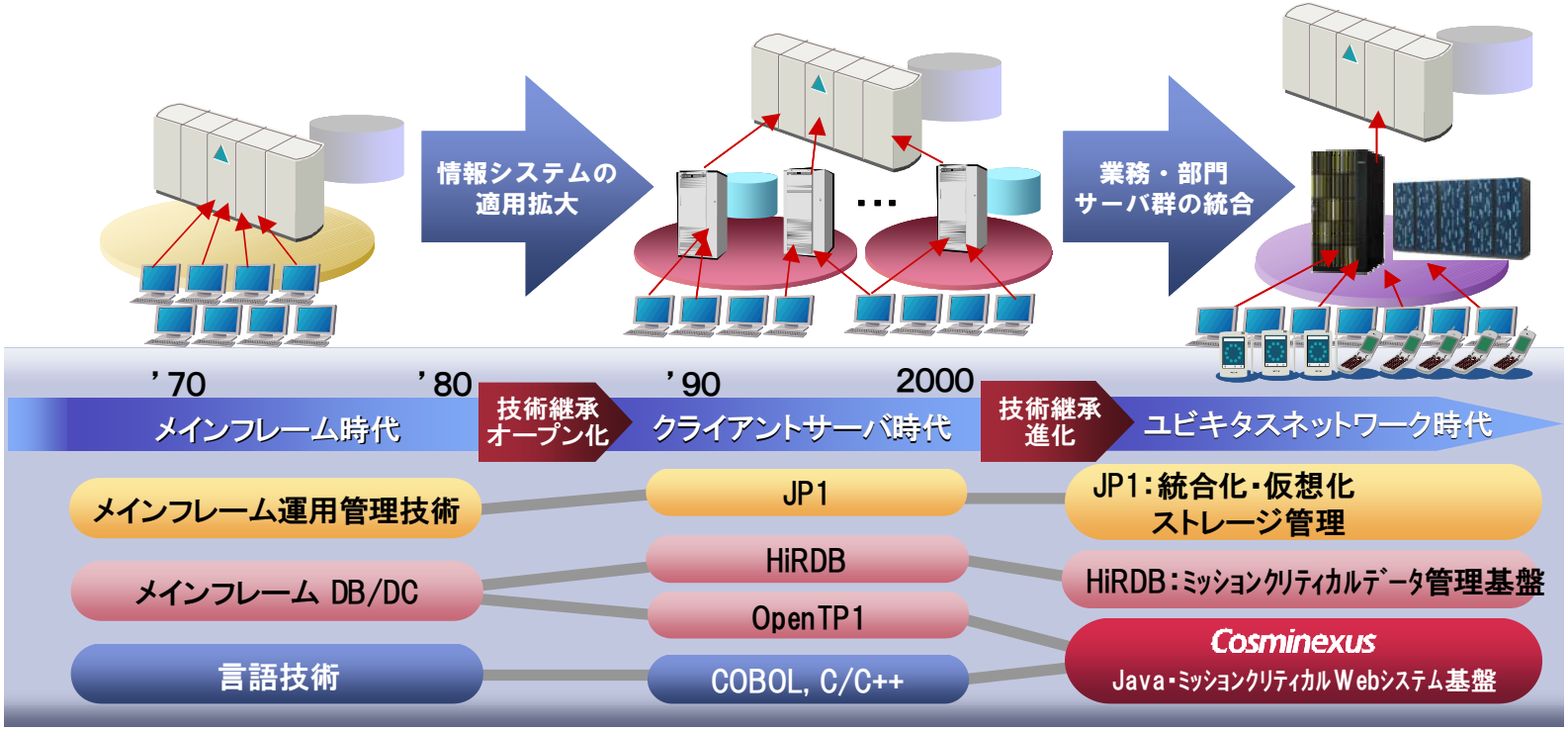
JavaおよびNETによる、Webアプリケーションのパフォーマンスを監視するソリューションです。
ソースコードを変更することなく、24時間365日にわたる低負荷な性能情報の取得を実現します。
ニーズに応じて監視画面を簡単に作成できるとともに、分析、レポート作成の迅速な対応が可能です。



HITACHI
Inspire the Next

Cosminexusアプリケーションサーバとは

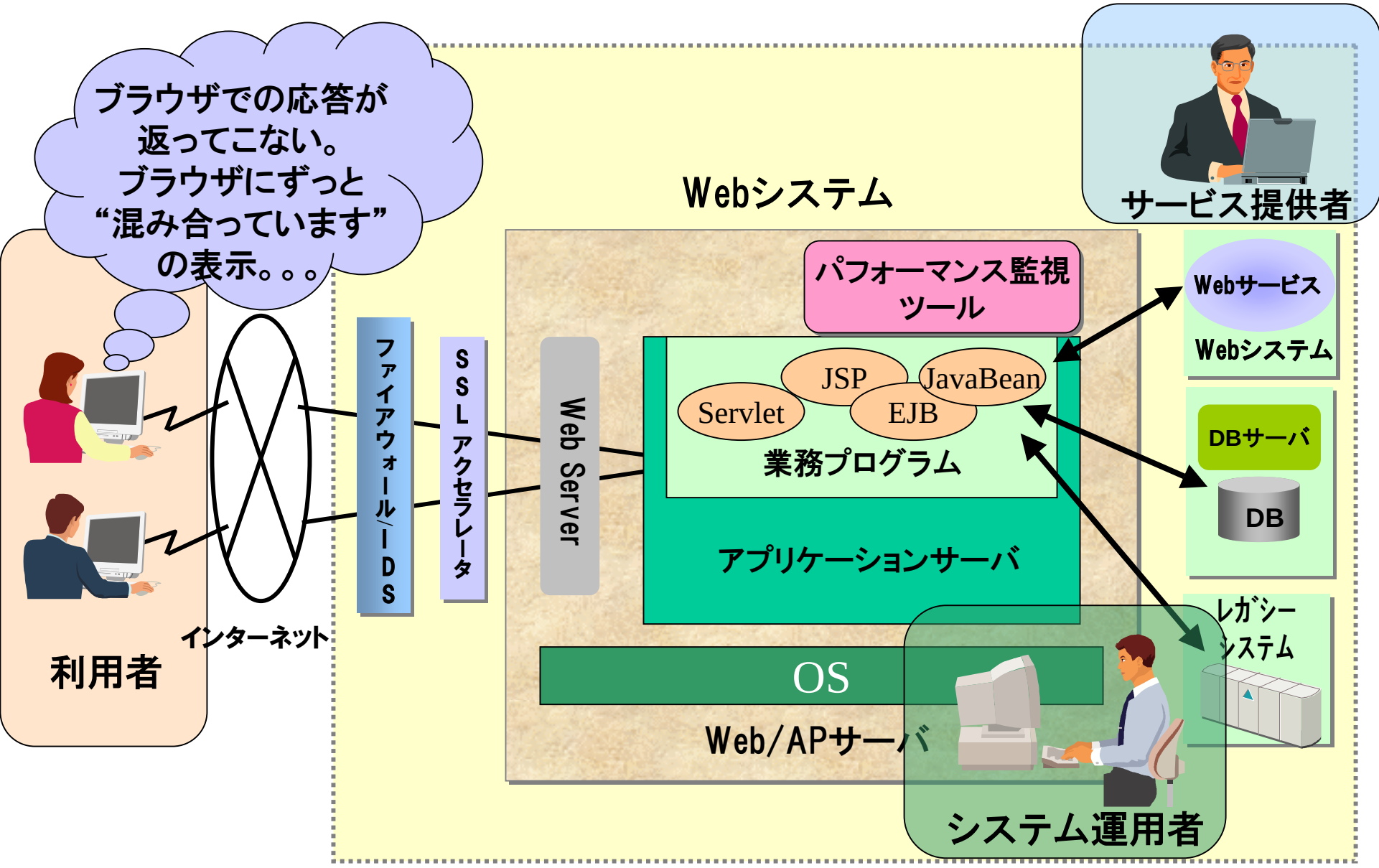
Java SE 5.0、Java EE 5 に対応したアプリケーションサーバです。
国内にソースを保有し、Java標準対応だけでなく、性能、信頼性、運用容易性、保守性を向上するための機能エンハンスを行なっています。
特に、Java VMについては、弊社基準の品質の確保、トラブルシュート機能の強化、性能強化を行っています。



「メインフレームで培ったミッションクリティカル技術の継承とたゆまざる進化」

uVALUE

Webシステムのトラブルとは？



HITACHI
Inspire the Next

トラブルでの対処

本番運用環境における障害発生時の典型的アプローチ

- 全員集合
  - 影響度により、すでに解散したプロジェクト要員を召集
  - すでに他のシステムに取り組んでいるエース級の人員を投入
- ソースコードレベルの調整
  - メソッドレベルにログ出力コードを付加
    - 修正工数と修正人員
    - 実行オーバヘッド増大
  - ログ集計とレポート
    - 複数サーバのログ集計の複雑性と工数の増大
    - レポート作業工数の増大
- 問題事象のステージングでの再現
  - すでにテスト中の次期システムがステージングを占有
    - 環境構築，テスト環境の利用スケジュール調整に翻弄される
  - なかなか再現しない、あるいはまったく再現しない
    - 本番での再発リスクと対応工数の増大
    - 次期システムのスケジュール遅延と人員キープのコスト増大
- 原因不明のまま時間が経過
  - 本番でまた再発！！！

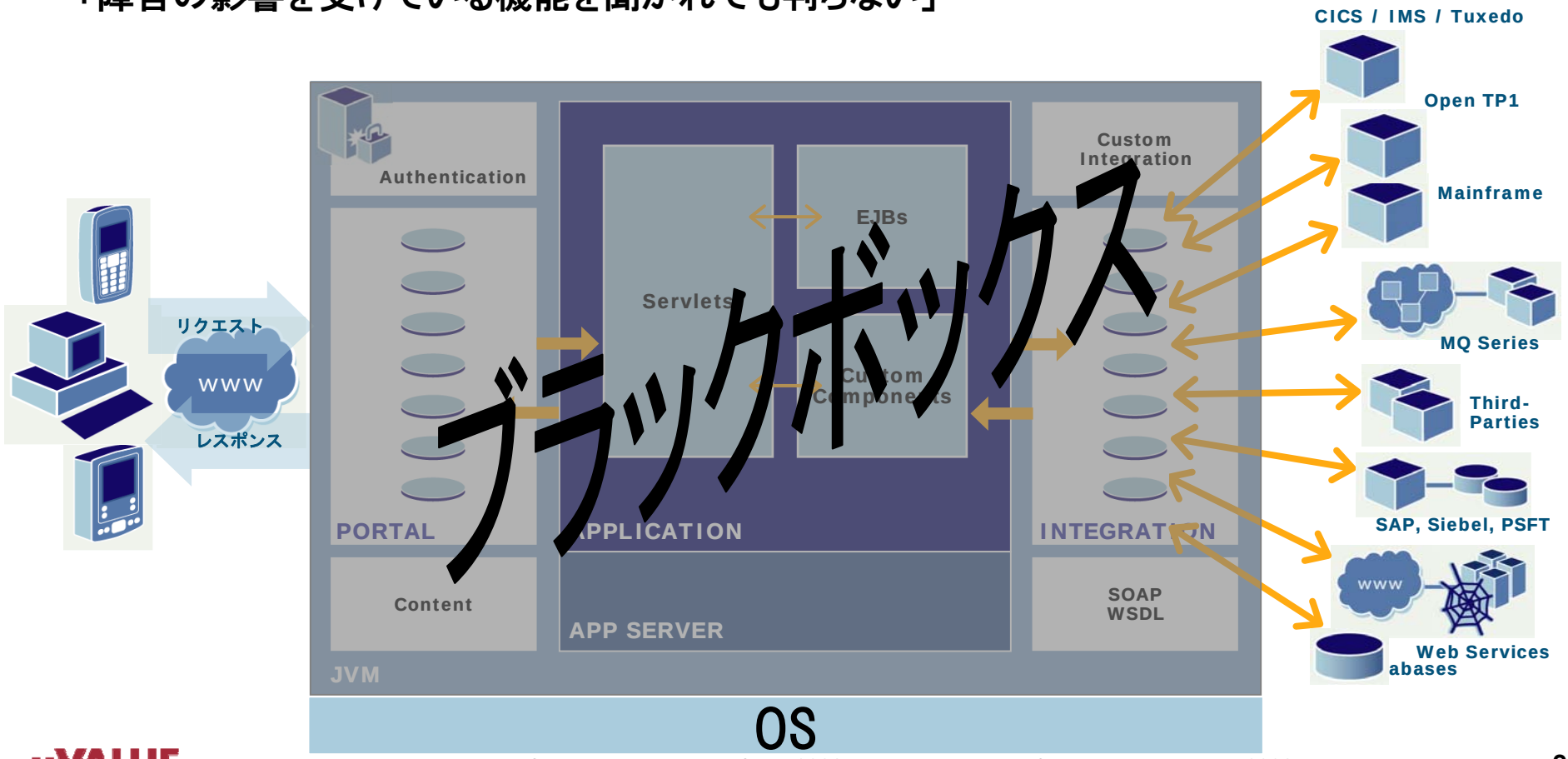
uVALUE

HITACHI
Inspire the Next

トラブルでの対処

J2EEアプリケーション上の運用　管理者の証言

「ブラックボックス。メインフレームなどに比べ、機能単位・業務単位の管理がしにくい」
「様々なバックエンドシステムの影響を常に受けているので、問題の特定が難しい」
「一時切り分けが出来ない為、障害発生時には、関係者全員集合」
「障害の影響を受けている機能を聞かれても判らない」



HITACHI
Inspire the Next

トラブルでの対処のポイント

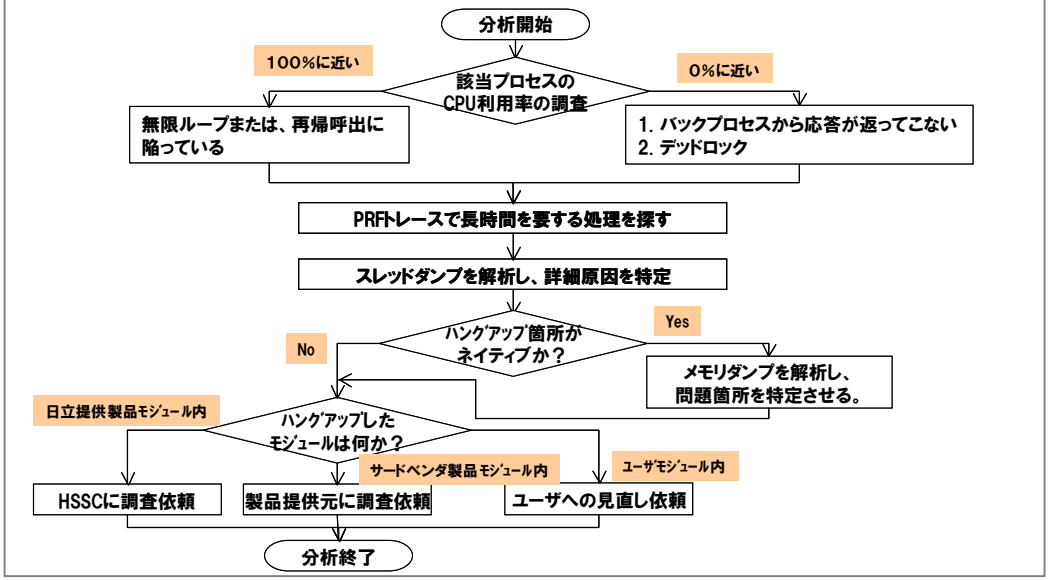
利用者の不安を取り除くことが第一

利用者の不安を取り除くためには、サービス提供者が、
利用者に状況を適切かつ定期的に伝えることが必要。

サービス提供者とサービス運用者での問題の共有が重要。
どこまで原因が判明していてどこが分らないのか。分るために何が課題で、何が必要なのかを認識しあうことが重要。

1. 原因調査のフローを作成
2. 調査状況は、フローに基づき定期的に報告

【調査ポイント】該当プロセスのCPU使用率とのつきあわせから推定原因を挙げ、PRFトレースで問題箇所を探し、詳細をスレッドダンプで特定
【有効な資料】PRFトレース/スレッドダンプ/ OSの統計情報



原因調査フローの例(Cosminexus認定資格講座 教育テキストより抜粋)

HITACHI
Inspire the Next

トラブルの原因は

非機能要件による障害が多い

- ユーザーの利用状況が想定できていない
  - リソースのアンバランス(データベース、ホストなどの外部接続のリソースが要件と合っていない）で、スレッドが詰まってしまう。

- J2EEのマルチスレッドに慣れていない
  - セマフォによるロックが長すぎる為に、実行スレッド数が多くなると急激に遅くなる。

- クラスタリングした事で、安心している
  - 一台が、障害が起きて、無反応やプロセスダウンになった時に、残されたサーバでリクエスト数をこなせないと、すべてのサーバが次々にダウンすると言う現象が発生します。(共倒れ)
    この場合の問題点は2種類存在していて、最初に障害を起こしたサーバの障害の解決と、1台辺りの想定リクエストを超えた時にどの様にリクエストをハンドリングするかを決める必要があります。
    良く聞くのは、根本の原因を追究せずに、残されたサーバーが倒れないように、ソーリーサーバにリクエストを振ることだけで終わってしまっています。

uVALUE

HITACHI
Inspire the Next

トラブルの種類

トラブルで特に多いのは？
迷宮入りしてしまうのはどんなトラブル？

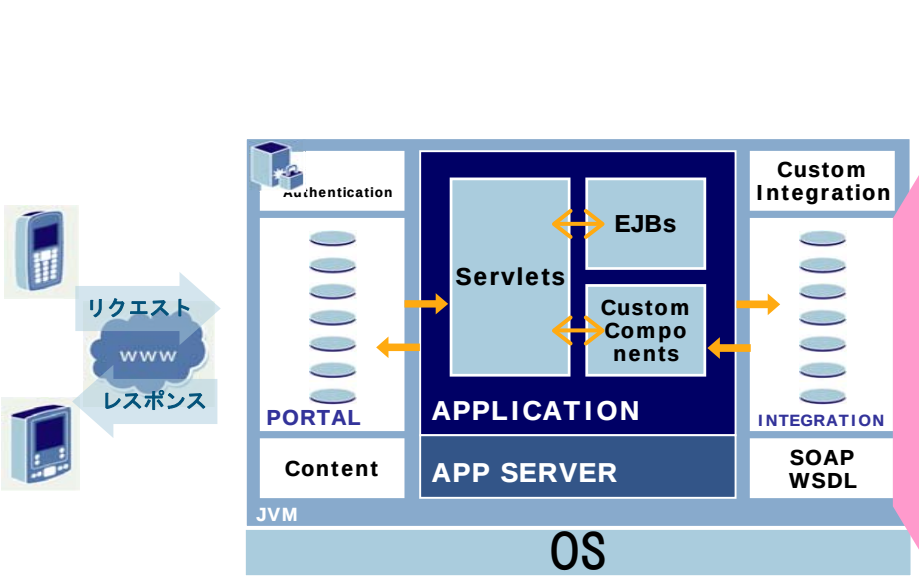
- プログラムのエラー終了
- 実行結果誤り
- スローダウン　どこをどう調べていいのかわからない。情報取得も不足し長期化
- セキュリティ(他の人の情報が見える)
- プロセスダウン
- ハングアップ
- メモリリーク
- ・・・

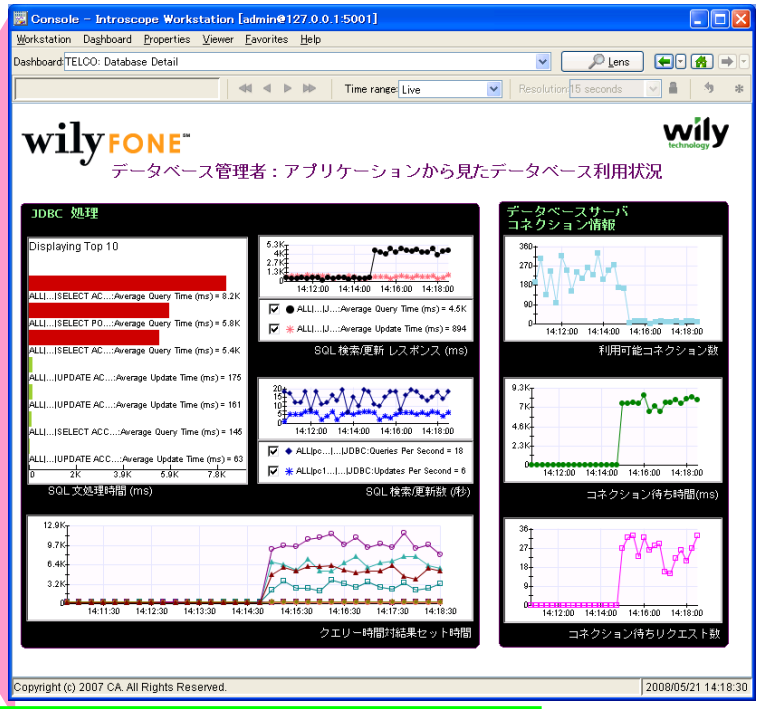
HITACHI
Inspire the Next

スローダウンの原因は？

今までは、データベースへのアクセス処理で遅延が発生しているケースが多い

- SQL Agentにより、実現
  Java,J2EE,.NETから見たデータベースの性能を監視
  -データベースに起因する様々な性能障害を特定





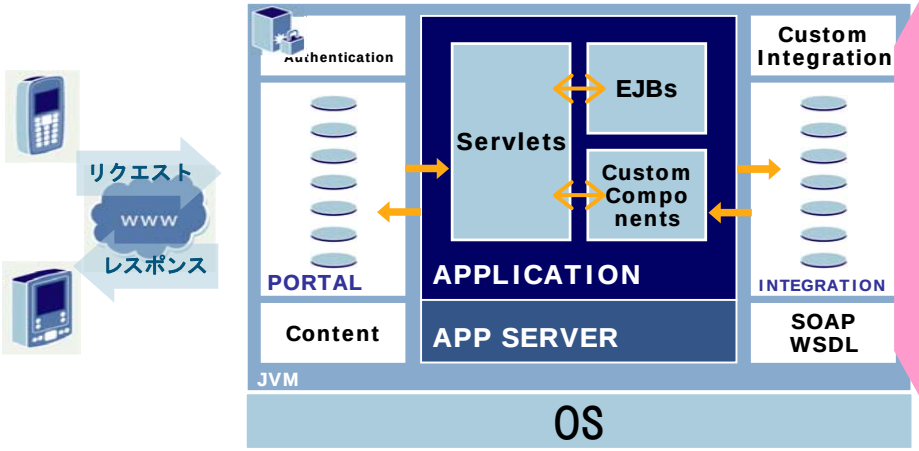
- SQL文単位に、応答時間、実行回数が取得可能
  SELECT ? FROM USERPROFILE WHERE USER = ? AND STATUS = ?
- コネクションプールのリソース情報が取得可能

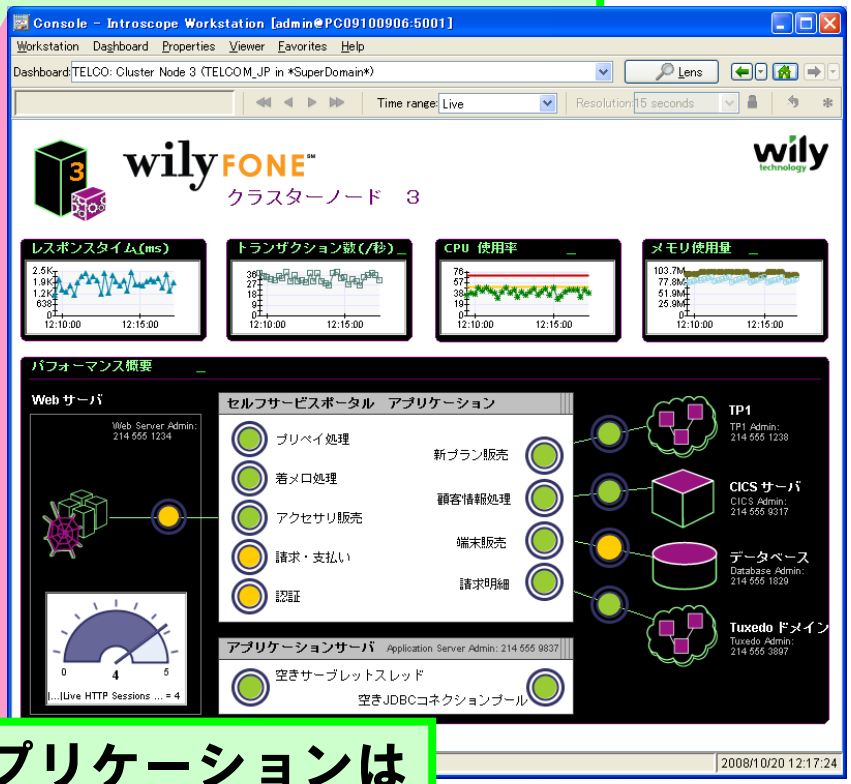
HITACHI
Inspire the Next

スローダウンの原因は？

レガシーシステムなどの外部接続処理での遅延が発生するケースが増えてる

- 外部接続の為のクラス(メソッド)を計測することにより実現
  応答時間、実行数、ストール状況を監視
  障害時の切り分けが明確





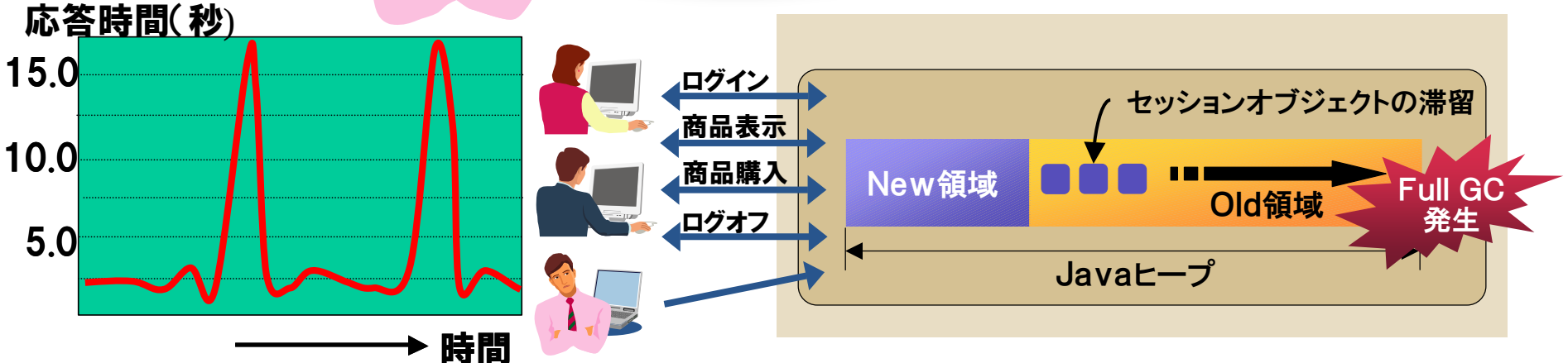
SOA化が進むことでフロントのWebアプリケーションは
システム全体のハブとなる為に、今以上に必要となる

HITACHI
Inspire the Next

スローダウンの原因は？

最近ではFull GCが原因となるスローダウンが多い。

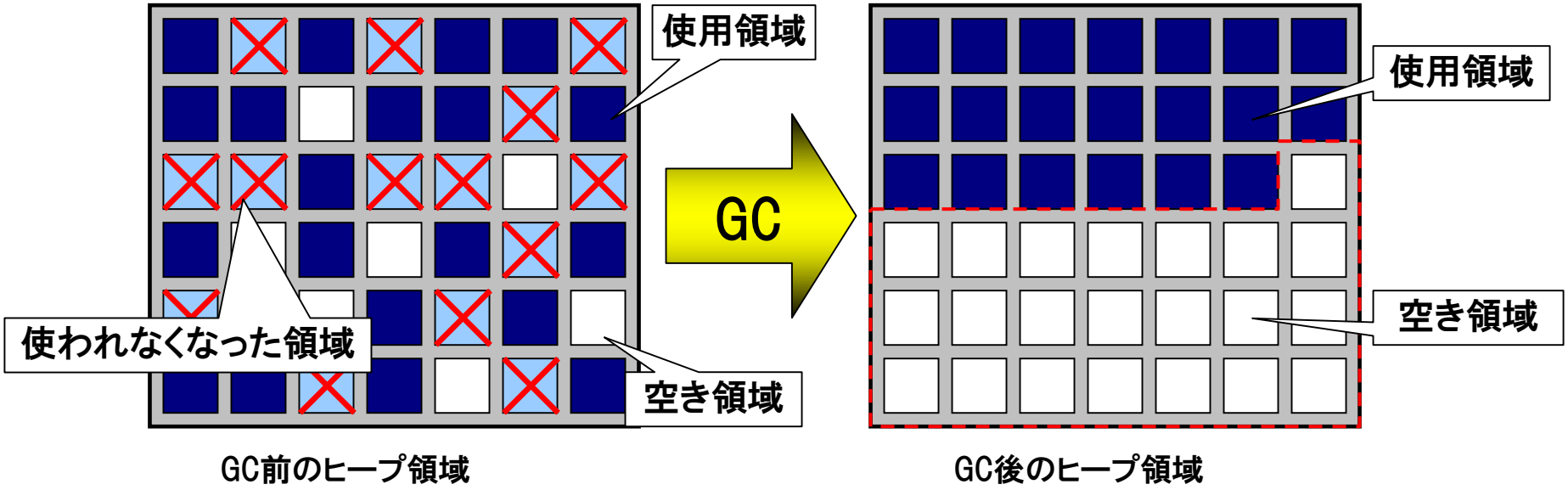
メモリも64ビット化になり一旦、Full GCが起きると数十秒間停まってしまう。
64ビット化のマシンを導入していいのか迷っている人も多い。



HITACHI
Inspire the Next

Full GCとは？

GCとは、Java VMが管理するメモリ領域中の使われなくなった領域を破棄し、空き領域を作ること。GCの処理は、実行中のプログラムを停止して行われる。



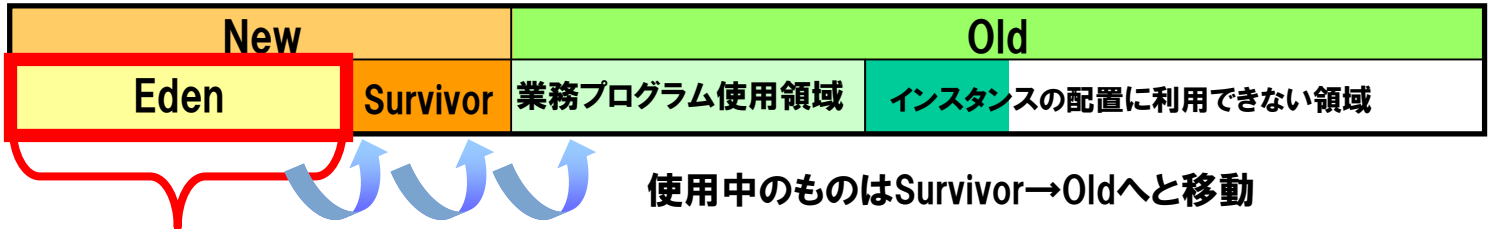
対象となるメモリ領域サイズが増加するとGC処理にかかる時間も増加する。

HITACHI
Inspire the Next

Full GCとは？

●GCには2種類ある

Copy GC…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



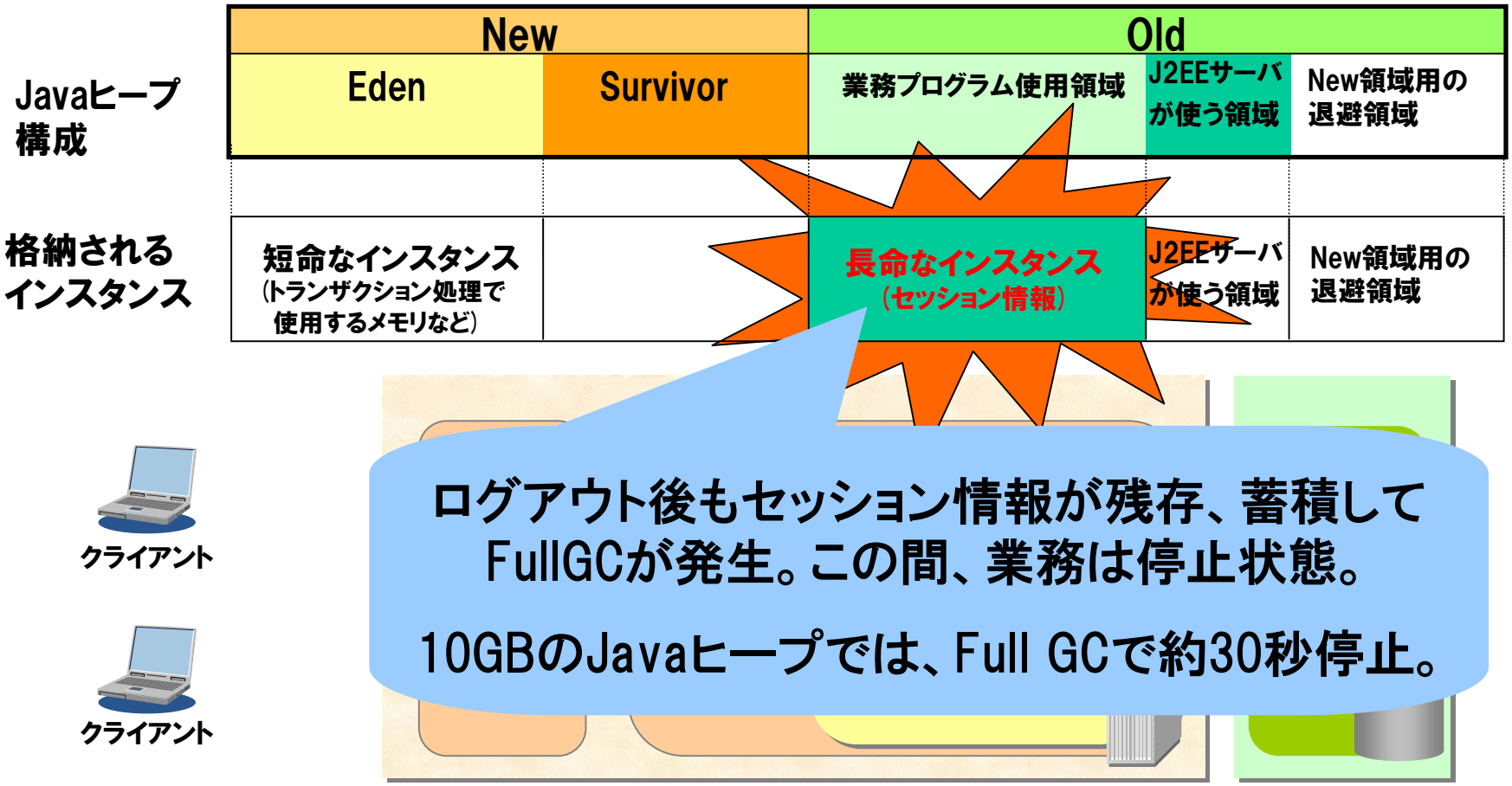
Full GC…全ての領域を対象に、使用済みのインスタンスを全て削除する

GC種別	範囲	発生タイミング	GC処理時間
CopyGC	New領域	Eden領域に空きがなくなった時	0.01～0.7秒
FullGC	全領域	Old領域の業務プログラム使用領域が一杯になった時	1秒～数十秒

HITACHI
Inspire the Next

Full GC発生の仕組み

Webアプリケーションでは、セッションオブジェクトが
Javaヒープ中のOld領域に滞留するためFull GCが発生

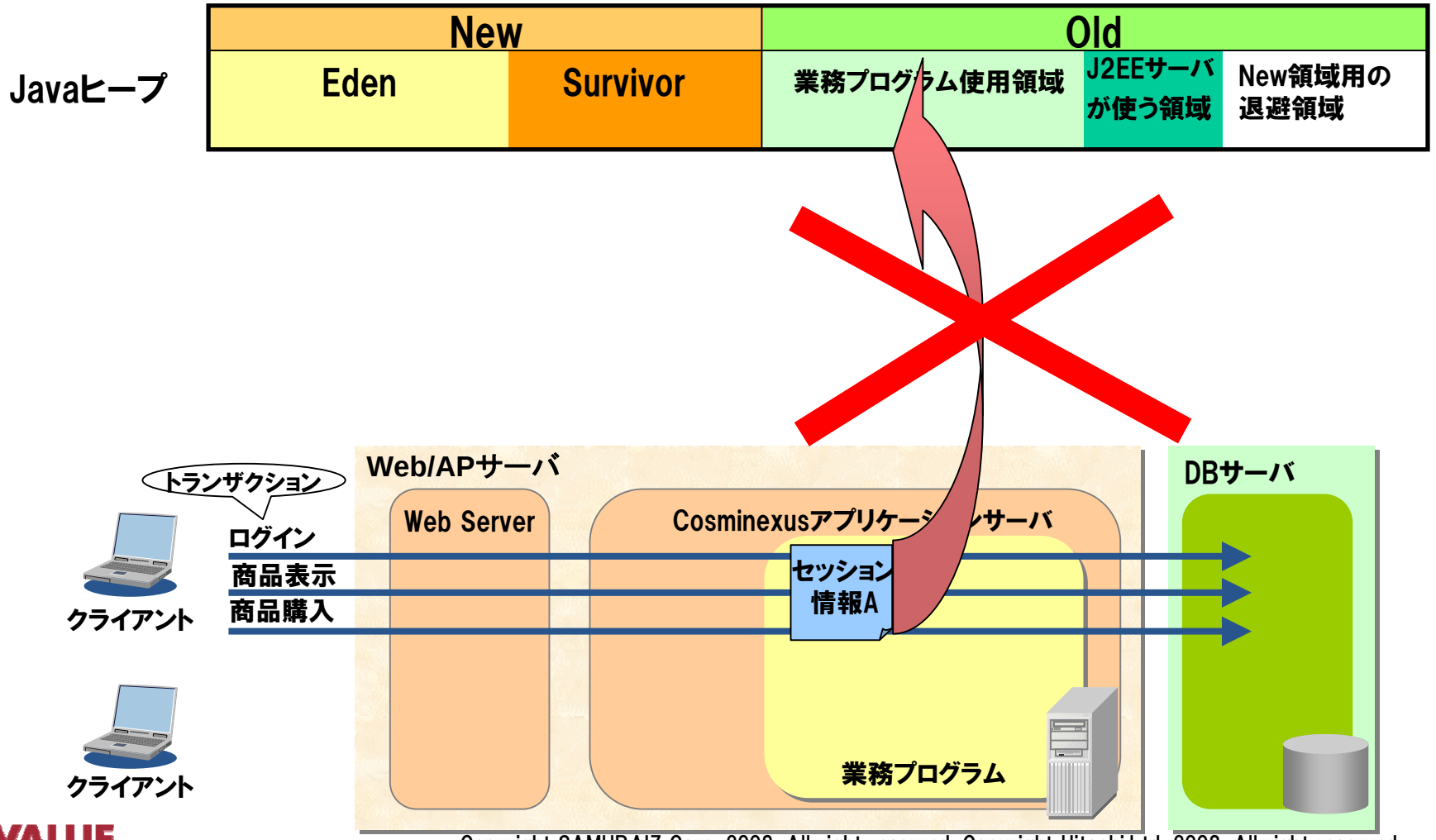


uVALUE

HITACHI
Inspire the Next

Full GCレス機能とは

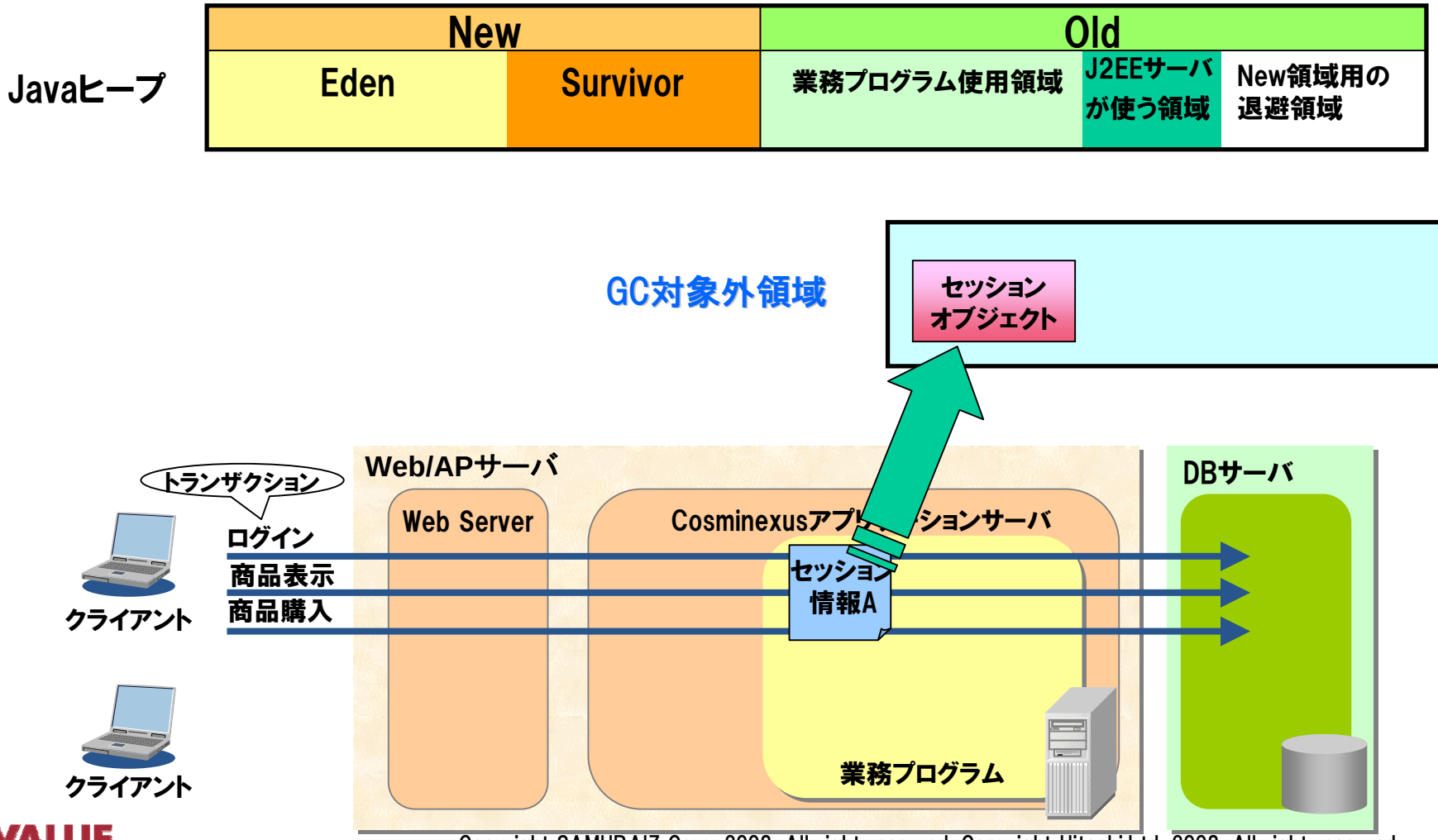
Webアプリケーションのセッションオブジェクトを
GC対象外領域に配置、ログアウト時に開放することでFull GCを抑止



HITACHI
Inspire the Next

Full GCレス機能とは

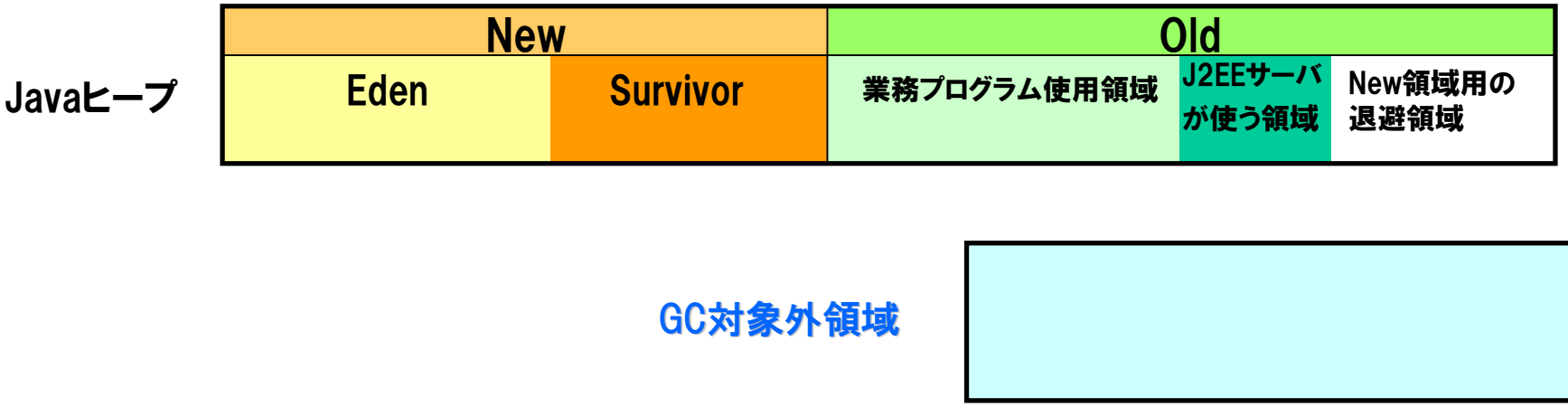
Webアプリケーションのセッションオブジェクトを GC対象外領域に配置、ログアウト時に開放することでFull GCを抑止

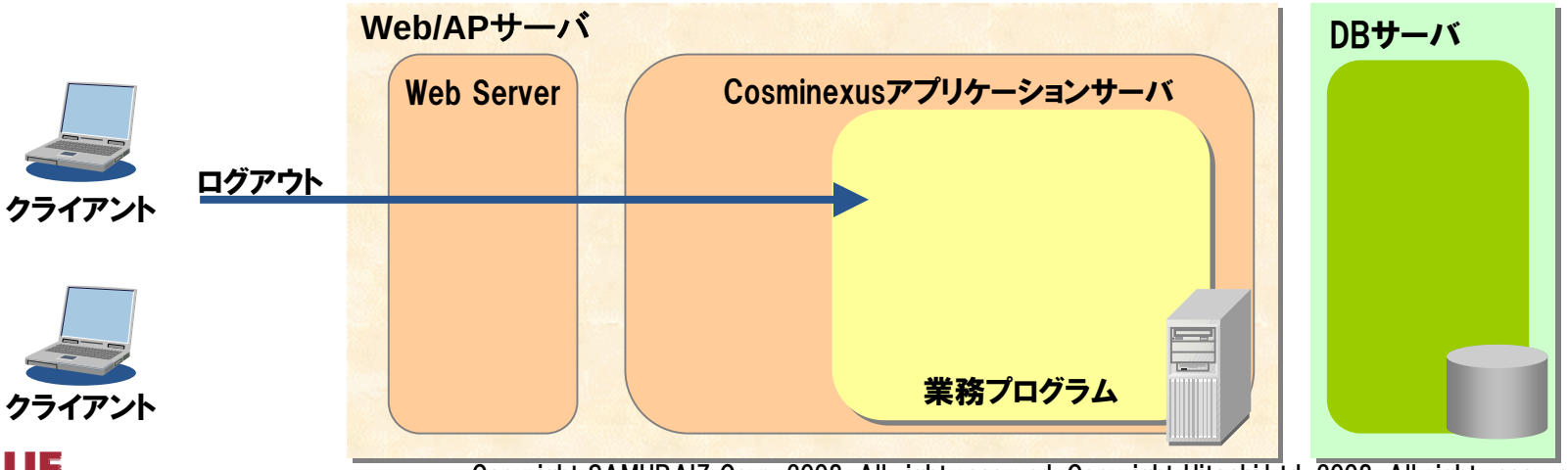


HITACHI
Inspire the Next

Full GCレス機能とは

Webアプリケーションのセッションオブジェクトを
GC対象外領域に配置、ログアウト時に開放することでFull GCを抑止



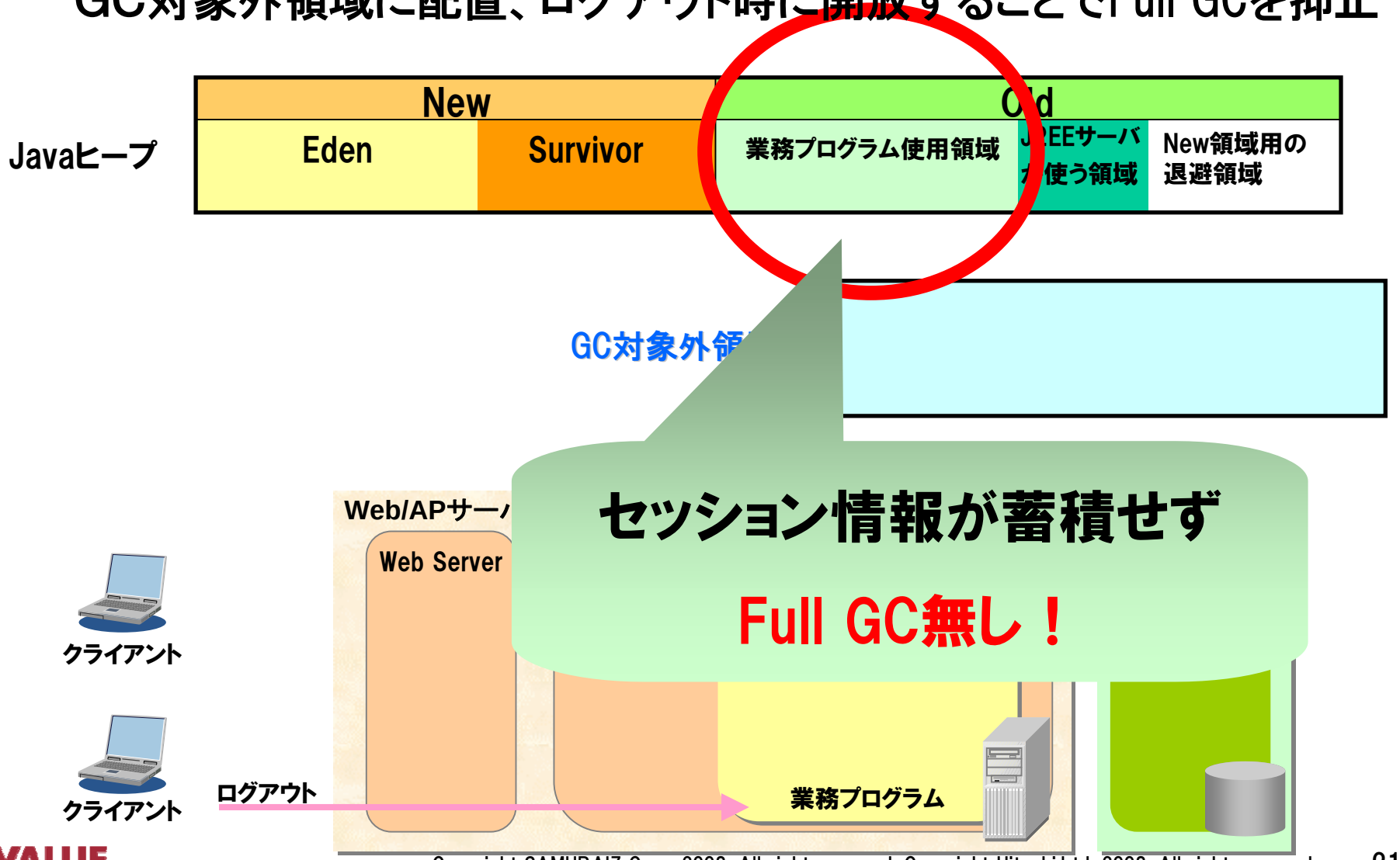


uVALUE

HITACHI
Inspire the Next

Full GCレス機能とは

Webアプリケーションのセッションオブジェクトを
GC対象外領域に配置、ログアウト時に開放することでFull GCを抑止



トラブルを未然に防ぐためには

適切なシステム設計がなされているのだろうか？

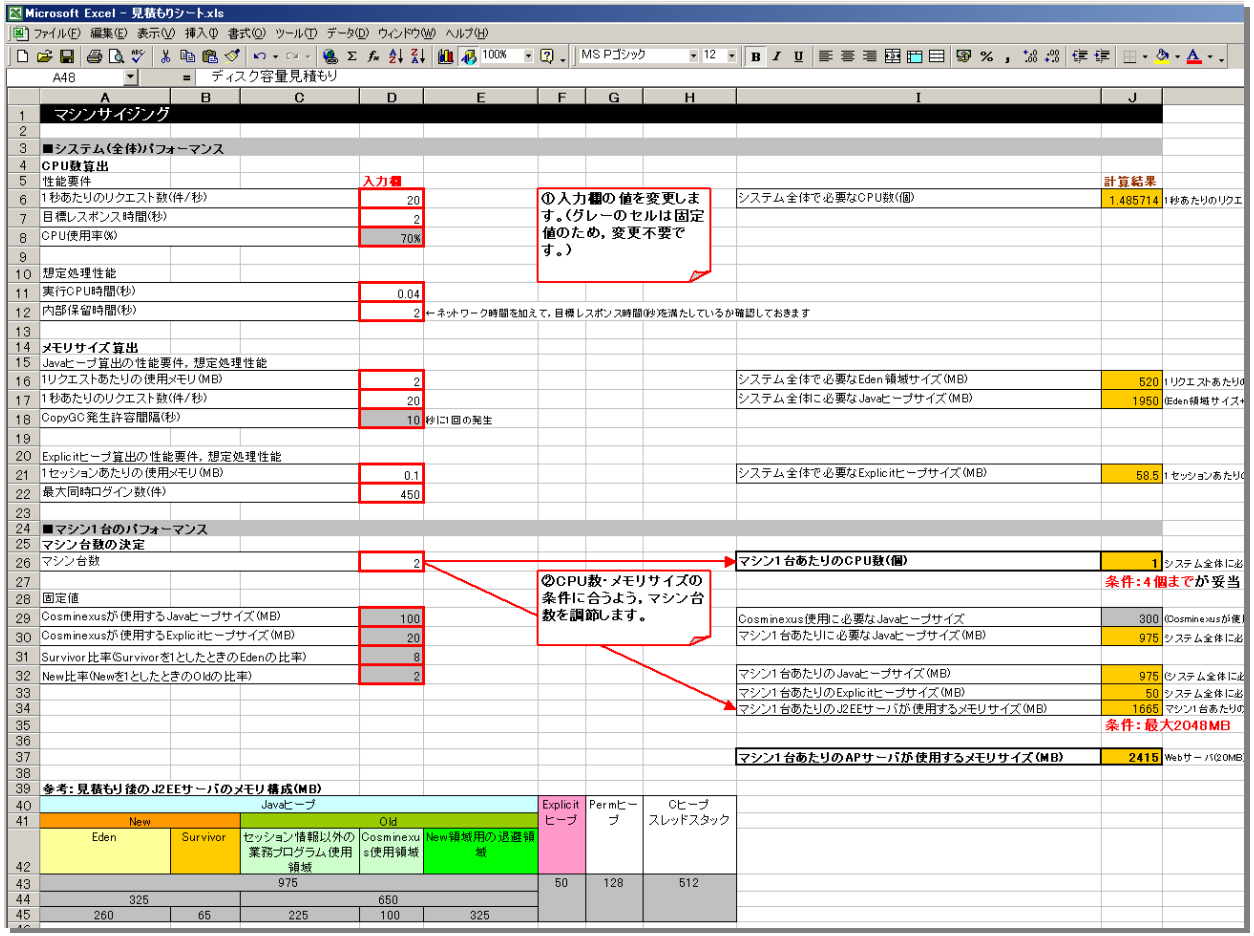
- 見積り/サイジング
- 性能設計
- 信頼性設計
- 運用設計

HITACHI
Inspire the Next

見積り/サイジング:Webシステムのマシンサイジング

Javaのシステムで見積もり/サイジングは出来るのか？

性能要件、プログラム属性情報から、マシン台数(CPU)，実メモリサイズは、見積り可能。



Microsoft Excel - 見積もりシート.xls

	A	B	C	D	E	F	G	H	I	J
1	マシンサイジング									
2										
3	■システム(全体)パフォーマンス									
4	CPU数算出									
5	性能要件			入力欄						計算結果
6	1秒あたりのリクエスト数(件/秒)			20					システム全体で必要なCPU数(個)	1.485714 1秒あたりのリクエ
7	目標レスポンス時間(秒)			2						
8	CPU使用率(%)			70%						
9										
10	想定処理性能									
11	実行CPU時間(秒)			0.04						
12	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					
13										
14	メモリサイズ算出									
15	Javaヒープ算出の性能要件，想定処理性能									
16	1リクエストあたりの使用メモリ(MB)			2					システム全体で必要なEden領域サイズ(MB)	520 1リクエストあたり
17	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
18	CopyGC発生許容間隔(秒)			10	秒に1回の発生					
19										
20	Explicitヒープ算出の性能要件，想定処理性能									
21	1セッションあたりの使用メモリ(MB)			0.1					システム全体で必要なExplicitヒープサイズ(MB)	58.5 1セッションあたり
22	最大同時ログイン数(件)			450						
23										
24	■マシン1台のパフォーマンス									
25	マシン台数の決定									
26	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
27										条件：4個までが妥当
28	固定値									
29	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
30	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
31	Survivor比率(Survivorを1としたときのEdenの比率)			8						
32	New比率(Newを1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
33									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
34									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
35										条件：最大2048MB
36										
37									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)
38										
39	参考：見積もり後のJ2EEサーバのメモリ構成(MB)									
40	Javaヒープ					Explicitヒープ	Permヒープ	Cヒープ スレッドスタック		
41	New		Old							
42	Eden	Survivor	セッション情報以外の業務プログラム使用領域	Cosminexusが使用領域	New領域用の退避領域					
43	975					50	128	512		
44	325		650							
45	260	65	225	100	325					

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。

パフォーマンス見積りシート(Cosminexus認定資格講座 教育テキストより抜粋)

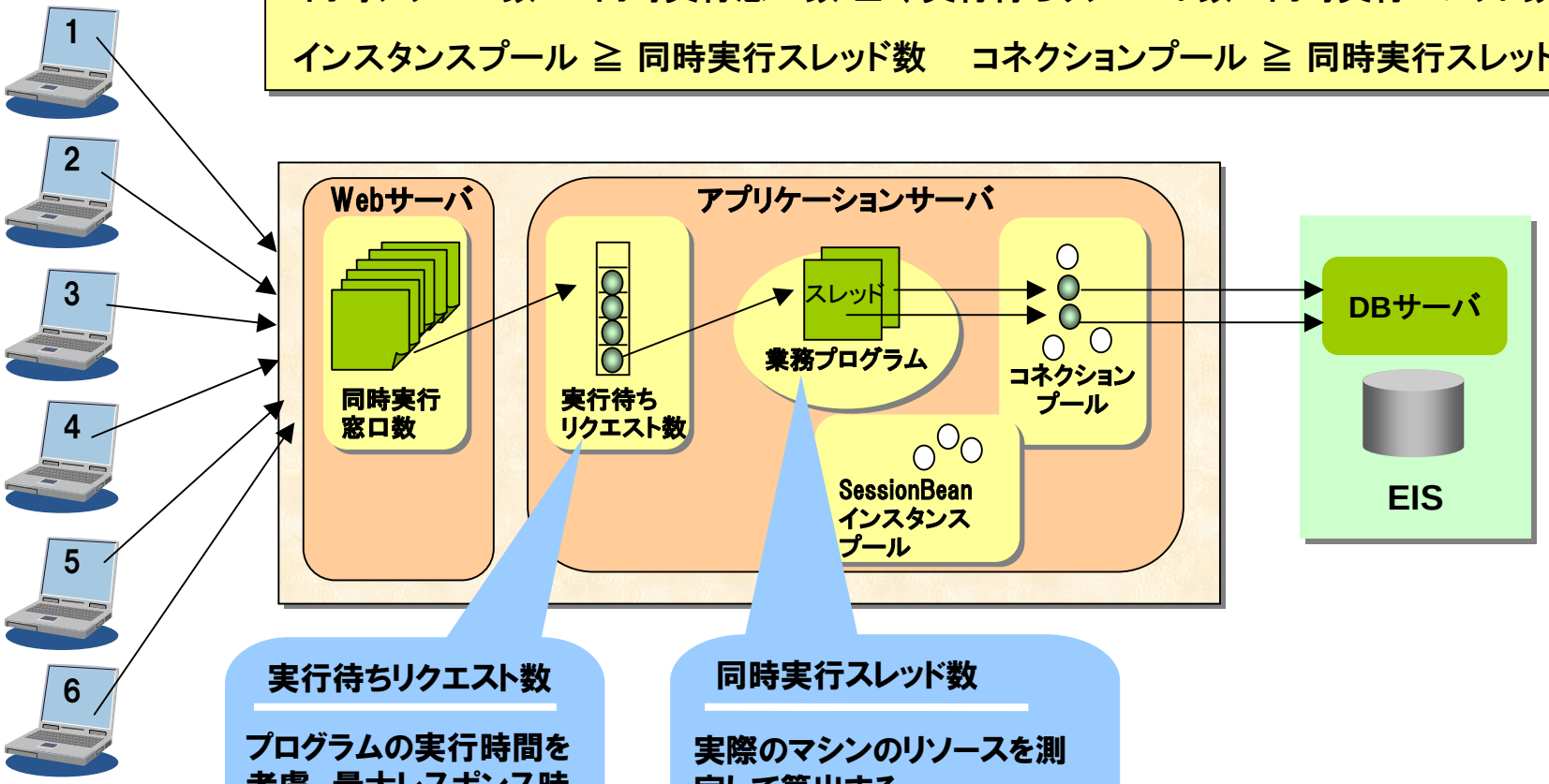
性能設計:Webシステムの流量制御設計

- 設計の指針

同時実行スレッド数 ＝（スループット×内部保留時間)/マシン台数

同時アクセス数 ＝ 同時実行窓口数 ≧（ 実行待ちリクエスト数＋同時実行スレッド数 ）

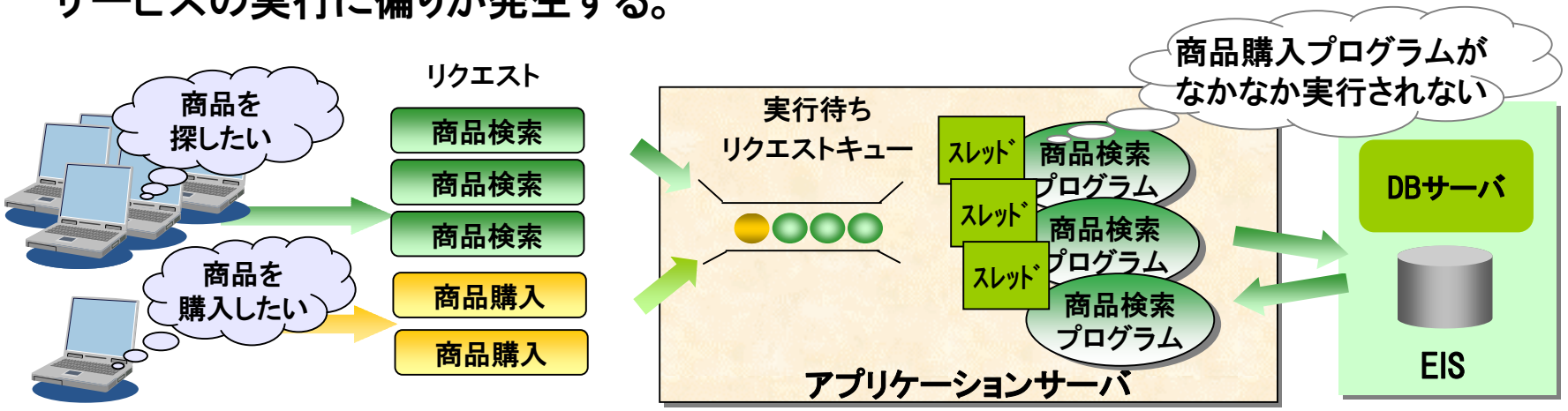
インスタンスプール ≧ 同時実行スレッド数　コネクションプール ≧ 同時実行スレッド数



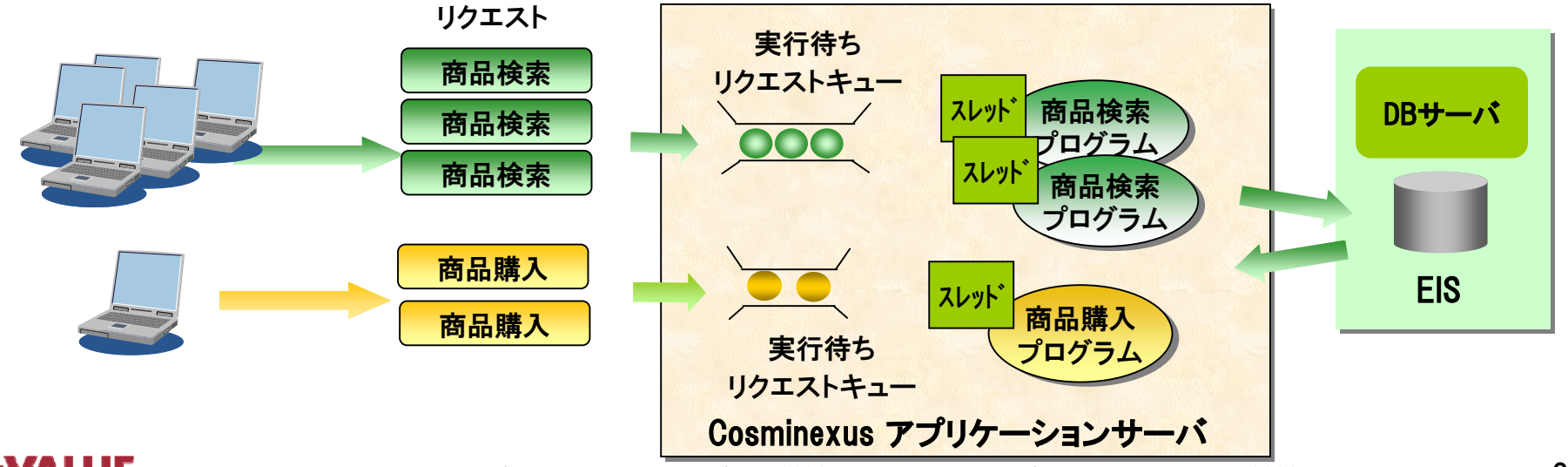
Webシステムの流量制御設計(Cosminexus認定資格講座 教育テキストより抜粋)

性能設計:Webシステムの流量制御設計

- 実行待ちリクエスト/同時実行スレッド数をJ2EEサーバ単位で管理する場合、サービスの実行に偏りが発生する。



- Cosminexusでは、業務プログラム単位(URLグループ単位)に実行待ちリクエスト数/同時実行スレッド数が設定可能。



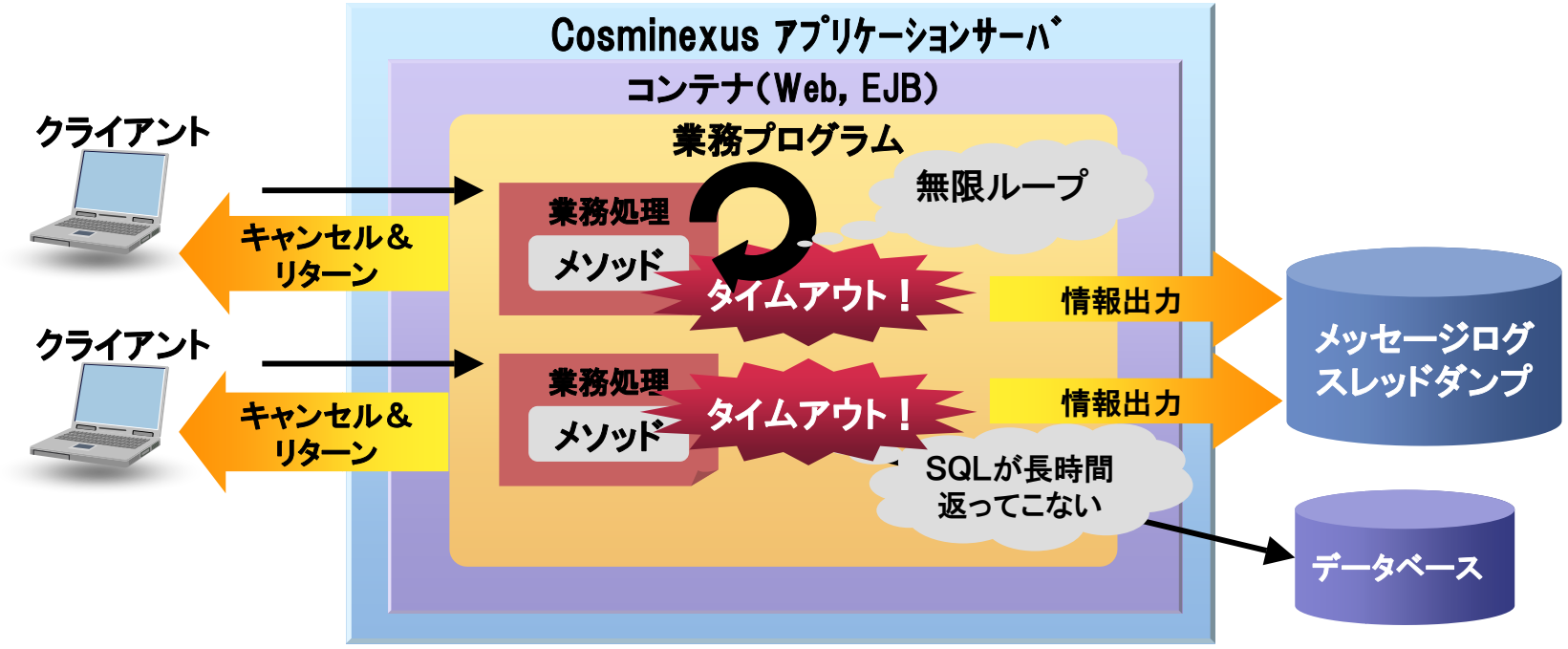
HITACHI
Inspire the Next

信頼性設計:障害の検知と情報収集

- 設計の指針

障害発生から復旧までの時間を短縮する。
障害をいち早く検知し、原因究明のための情報を収集する。

- Cosminexusでは、JSP/Servlet,EJBのメソッド単位に実行時間を監視し、タイムアウトを検知可能。
  タイムアウト発生時には、メッセージの出力およびスレッドダンプを自動出力。
  問題となったプログラム処理もキャンセル可能。



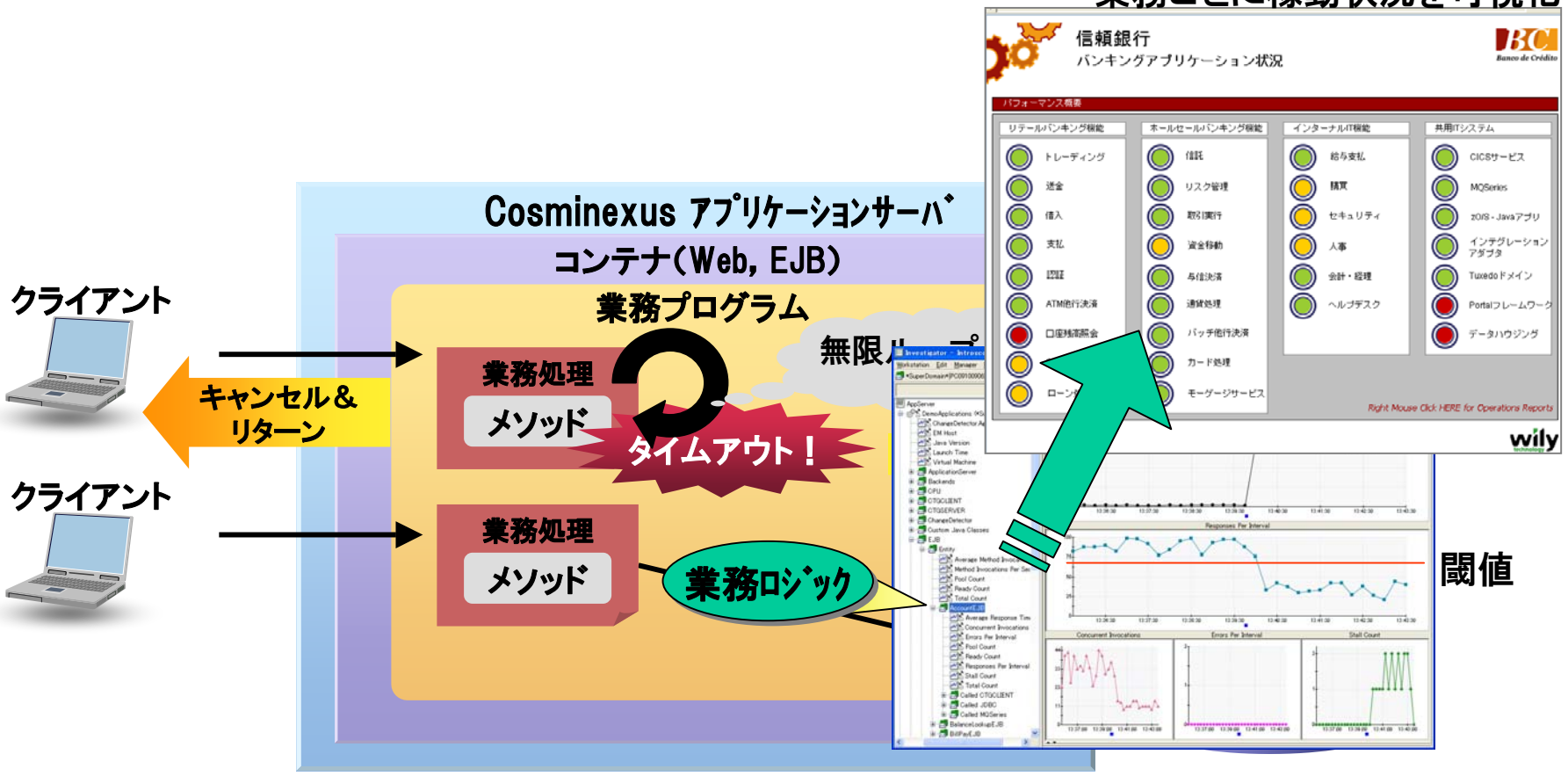
HITACHI
Inspire the Next

信頼性設計:障害の検知と情報収集

業務単位での性能管理が必要

業務ロジックのクラス(メソッド) を、監視ポイントに設定。
アラート(閾値)を設定することで、業務の稼働状況を可視化。

業務ごとに稼動状況を可視化



HITACHI
Inspire the Next

運用設計：稼動情報の収集

- 設計の指針

現在の利用者に快適に使用頂くためのトレンド分析、将来のキャパシティ・プランニングに役立てるため、稼動状況、問題点を時系列的に収集、管理する。

対象	取得ポイント		キャパシティプランニング	トレンド分析
Web Server	同時実行窓口数		○	
	アクセス情報（アクセスページ、キャリア種別など）			○
J2EEサーバ（Webコンテナ）	Webコンテナ単位	実行待ちリクエスト数	○	
		リクエスト処理中のスレッド数	○	
	業務アプリケーション単位	実行待ちキュー	○	○
		リクエスト処理中のスレッド数	○	○
	URL単位	実行待ちキュー	○	○
		リクエスト処理中のスレッド数	○	○
J2EEサーバ（JavaVM）	メモリ情報		○	
	GC情報		○	
リソース	使用中のコネクション数		○	
OS	メモリ情報（ファイルシステムキャッシュが利用しているバイト数、１秒間あたりのページフォールト数など）		○	
	プロセス情報（現在オープンしているハンドルの総数、スレッド数）		○	

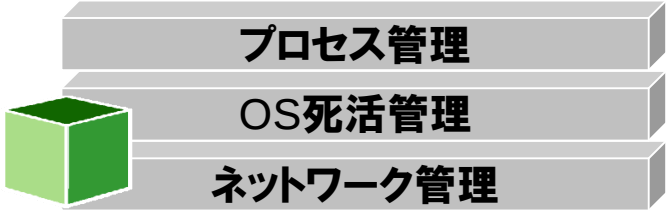
稼動情報一覧(Cosminexus認定資格講座 教育テキストより抜粋)※網掛け部分はCosminexusで自動取得される項目

uVALUE

HITACHI
Inspire the Next

運用設計：稼動情報の収集

OSやプロセス単位の稼動情報だけでは不十分。
同一時間軸で、業務ロジックとリソースの状況を把握しておくことが重要。



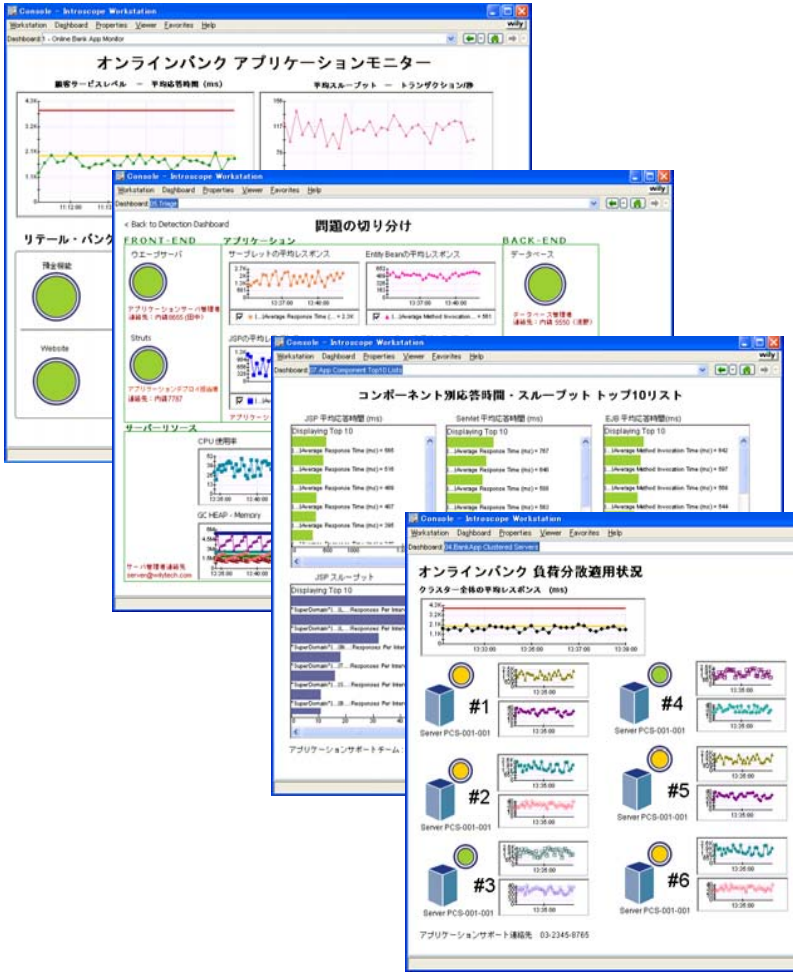
JP1による統合監視

- OSリソース
- ネットワークなど

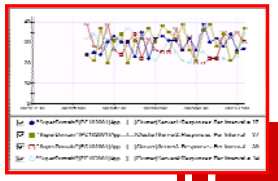
HITACHI
Inspire the Next

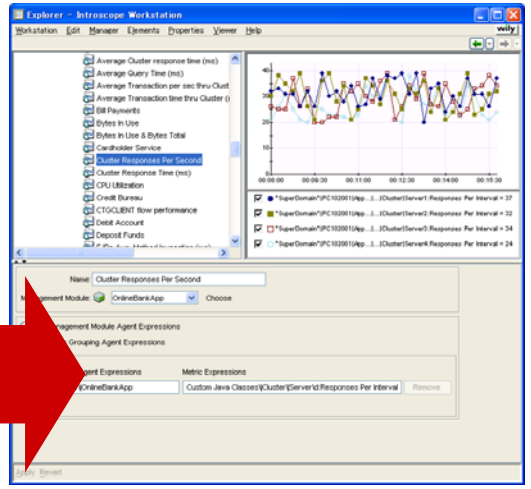
運用設計：稼動情報の収集

24x7のオーバヘッドの少ない情報蓄積



- ソースコードの変更が無い情報収集
- Agentは約5%という非常に小さな実行負荷で正常な状態を含めて、24時間定常モニタリング
- 管理プロセスや担当する視点別に自由にレイアウトし、リンクできる管理ダッシュボード





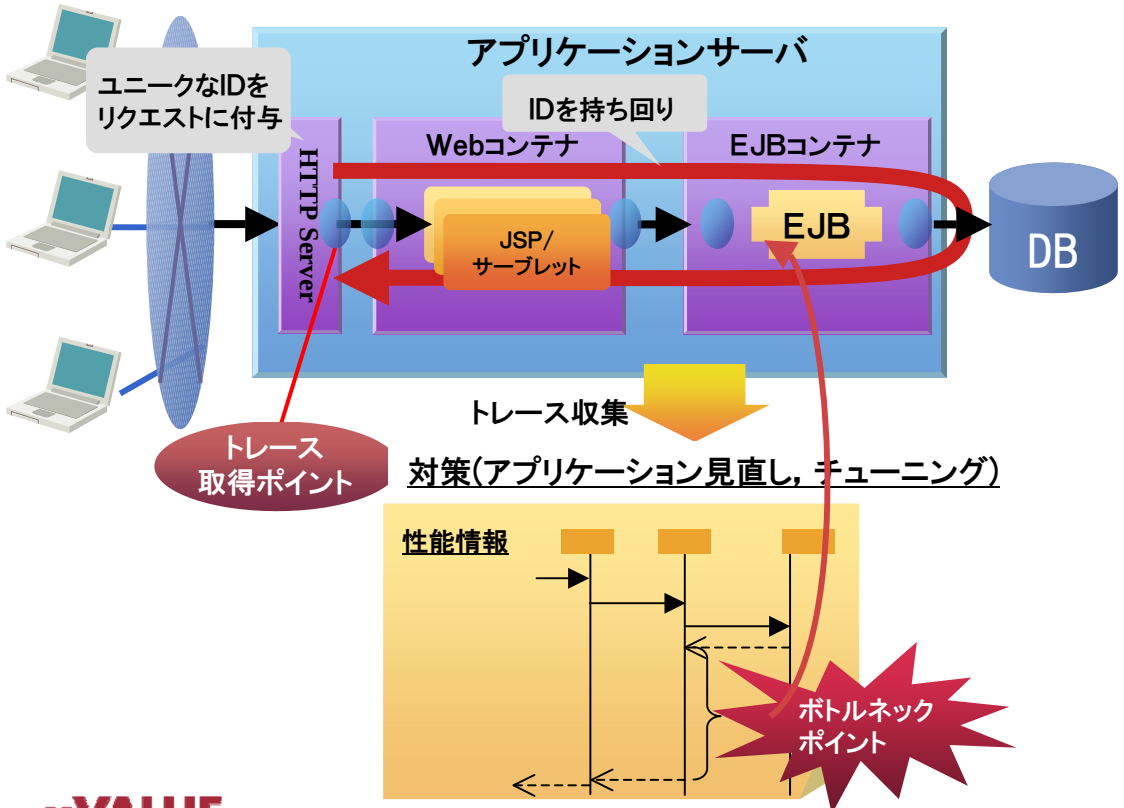
HITACHI
Inspire the Next

トラブルシュート：サーバ内の可視化

- 障害の対処では、どう動いたか、問題となった時点のデータ内容はどうだったかを明確にする必要がある。

Cosminexusでは、通常稼動中にリクエスト毎にトレースを取得。性能のボトルネック、障害発生のトリガーとなったリクエストを簡単に特定可能。

Cosminexusのスレッドダンプでは、異常の部位に加えデータ内容、消費CPU時間、待ち回数が出力。



Cosminexus JavaVMスレッドダンプ出力例

```
"Sample" daemon prio=5 jid=0x003fc47c tid=0x02f89900 nid=0x23b8 runnable
[user cpu time=43406ms, kernel cpu time=5234ms] [blocked count=7788, waited count=1]

  at Example3.method(Example3.java:18)
    locals:
      (Example3) this = <0xaa07db58> "I am an Example3 instance." (Example3)
      (java.lang.String) l1 = <0xae173a28> "local 1" (java.lang.String)
      (java.lang.StringBuffer) l2 = <0xaa07dca0> "local 1 + local 2" (java.lang.StringBuffer)
      (java.lang.Boolean) l3 = <0xaa07de18> "false" (java.lang.Boolean)
      (java.lang.Character) l4 = <0xaa07df68> "X" (java.lang.Character)
      (java.lang.Long) l5 = <0xaa07e078> "-922337203685[illegible]5808" (java.lang.Long)
      (java.lang.Object) l6 = <0xaa07e1a8> "Thread[Thread[illegible].main]" (java.lang.Thread)
      (java.lang.Object[]) l7 = <0xaa07e298> "[L.java.lang.[illegible]ead;@26e431"
(java.lang.Thread[])
  at Example3.main(Example3.java:4)
    locals:
  ...
```

メソッドの引数、内部変数データを出力

スレッドの消費CPU時間、ロック待ち回数を出力

Sun版オリジナルJavaVMスレッドダンプ出力例(ご参考)

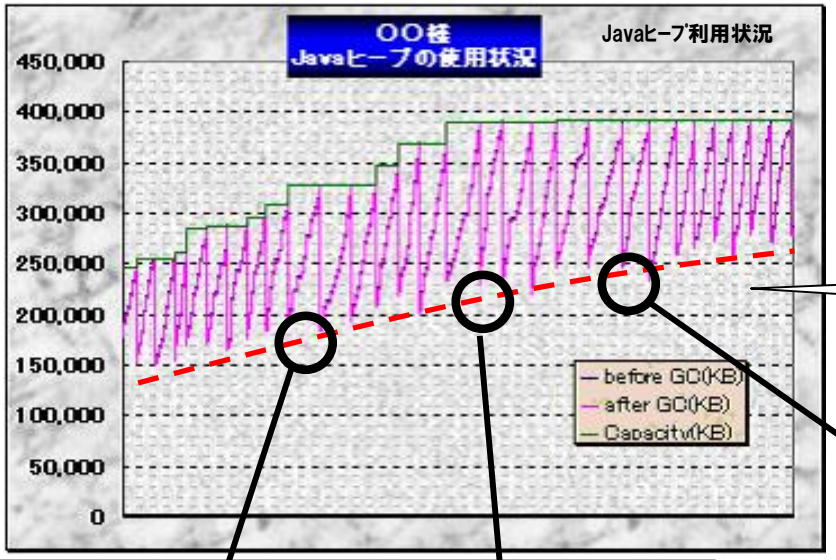
```
"Sample" daemon prio=5 jid=0x003fc47c tid=0x02f89900 nid=0x23b8 runnable

at Example3.method(Example3.java:18)
at Example3.main(Example3.java:4)
...
```

uVALUE

HITACHI
Inspire the Next

トラブルシュート：メモリリークの調査

- メモリリークは、再現および原因解析が困難。本番稼動中にオブジェクトの詳細な情報が取得できる必要がある。
- Cosminexusでは、本番稼動中にオブジェクトの詳細情報を取得可能。



①メモリリークの兆候を発見。

メモリ使用量が単調増加しており
メモリリークしている可能性がある。

③所要量が大きいクラス名を引数にコマンドを投入し、
クラス参照情報を出力。

②クラス別統計情報をコマンドを使って出力。

```
Java Heap Profile
------------------------
_____Size_Instances_Class_____
          859336                    5
Ljava.io.ObjectStreamField:
          856632       314 java.lang.Clas
          303848       373 [[I
          147192       701 [C
          131352       743 java.lang.Stri
          115272         2 java.lang.Thr
                （以下省略）
```

所要量が大きいクラスを抽出

```
Java Heap Profile
------------------------
_____Size_Instances_Class_____
          860872       314 java.lang.Class
          859336                    5
[Ljava.io.ObjectStreamField:
          303656       373 [[I
          161400      1056 [C
          154080      1098 java.lang.String
          108400         2 java.io.PrintStream
                （以下省略）
```

```
Reference of class java.lang.String
------------------------------------
java.lang.ThreadGroup (0x104f00c8)
  java.lang.ThreadGroup (0x104f00f0)
    java.lang.Thread (0x104f0168)

java.lang.ThreadLocal$ThreadLocalMap (0x104f1840)
------
（中略）
------
java.io.WinNTFileSystem (0x10503ba0)
  java.io.ExpiringCache (0x10503bb8)
    java.util.HashMap (0x10503bd8)
      java.util.HashMap$Entry (0x10504e88)
        java.io.ExpiringCache$Entry (0x10504e70)
          java.lang.String (0x10504da8)
------
```

④問題のクラスの
検証・対策を実施。

uVALUE

トラブルシュート(デモンストレーション)

オブジェクト・リンク
障害予兆検知時や緊急の際の対応プロセスなどをリンクとして自由に定義ができます。
単純な監視ではなく、そこから何をするべきかというプロセス指向の管理システムを設計できます。

問題を検知・特定

問題の切り分け

さらに詳細な問題判別画面へ

管理情報を掲載したウェブページ

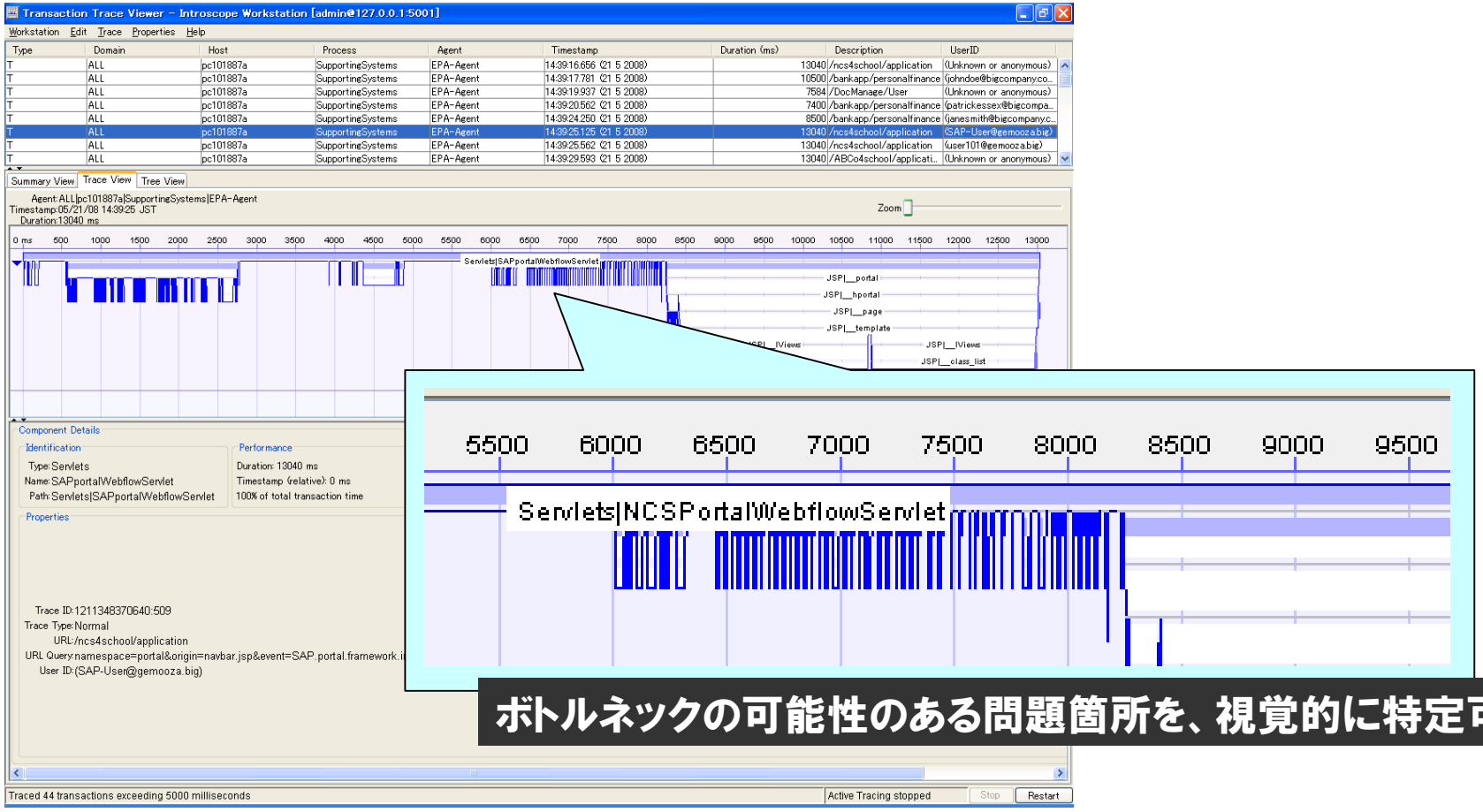
HITACHI
Inspire the Next

トラブルシュート(デモンストレーション)

トランザクションデータを丸裸に！

ボトルネックの可能性のある問題箇所を瞬時に分析可能

トランザクションの処理時間推移にて呼出関係を確認可能



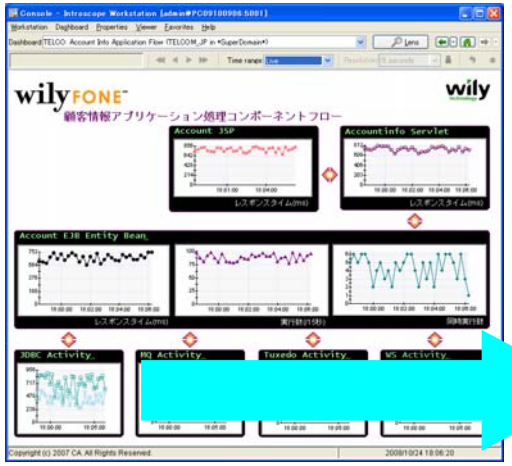
ボトルネックの可能性のある問題箇所を、視覚的に特定可能

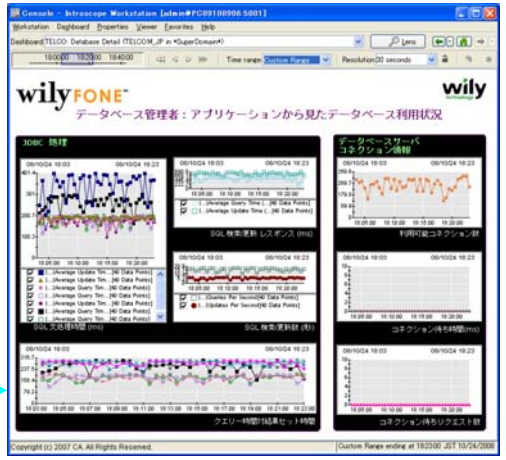
HITACHI
Inspire the Next

予兆検知(デモンストレーション)

- 正常時と異常時の比較　→　予兆の洗い出し

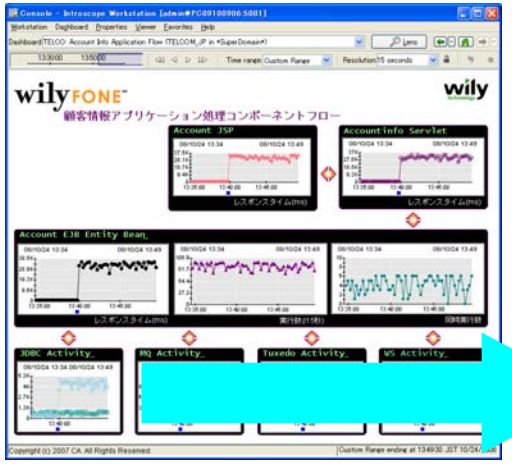
正常時

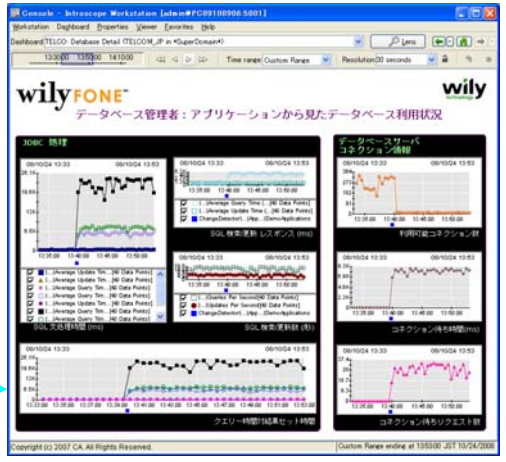




- リソースからの予兆
  - コネクション待ちの発生
  - フレームワークのリソース
  - アプリケーション内のリソース
- メソッドの実行からの予兆
  - メソッドの実行時間
  - メソッドの実行回数

異常時





予兆としてのアラートを設定して、ユーザーが障害を感じる前に対応

HITACHI
Inspire the Next

まとめ

- アプリケーションの性能情報を取得することが重要。
  → リアルタイムの計測と集計はツールが実施。
  集計作業は不要。

- アプリケーションの動きを、共有できることが重要。
  → ダッシュボードで、必要な情報を共有可能。
  担当者の視点で、瞬時に状況を把握可能。

- 正常時と異常時の比較をすることが重要。
  → データは蓄積されていて、容易に履歴が参照可能。
  履歴を比較することで、問題箇所を検出可能。

- 予兆としての検知をすることが重要。
  → 正常時と異常時の比較、解析により予兆ポイントが検出可能。
  利用者が影響を受ける前に対応開始が可能。

まとめ

Webシステムのトラブルでは、

利用者の不安を取り除くことが第一

利用者に快適かつ安心してサービスを使って頂くためには、以下の設計、対処が必要。

快適なWebシステムには、応答性能の良さが求められる

・見積り/サイジング → 要求性能に対し、適切なハードウェアを準備する。
・性能設計 → 見積られたハードウェア上で、要求性能を定常的に確保する。

障害が発生してもサービスを継続、迅速かつ確実に回復

・信頼性設計 → 障害をいち早く検知し、原因究明のための情報収集を行う。

利用者に快適なサービスを継続し続ける

・運用設計 → トレンド分析、キャパシティ・プランニングのため稼動情報を収集。

障害の原因究明の迅速化

・トラブルシュート → サーバ内を可視化し、問題箇所、原因を明確化。

他社所有名称に対する表示

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

サムライズ ホームページ
http://www.samuraiz.co.jp/product/wily
CA Wily Introscopeに関するお問合せ先
intorscope_info@samuraiz.co.jp

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。