重新考虑在 zEnterprise 上运行Java

z/OS 上的WebSphere
zBX上的WebSphere
跨zEnterprise的Java

Mike Cox

IBM 杰出工程师
美国高级技术技能
华盛顿系统中心
coxm@us.ibm.com

主题

- Java – J9 和 zSeries - z/OS
- 为什么采用面向 z/OS 的 WebSphere
- 为什么采用在 zBX 上发布的 WebSphere
- Java 机会
- 参考信息
- 问题

关于 z/OS 上的Java，你应当知道些什么

IBM 和 Java

Java 对于 IBM 极其重要

IBM 软件产品的根本基础架构

WebSphere, Lotus, Tivoli, Rational, Information Management (IM)

IBM 正在对虚拟机中的 Java 进行战略性投资

在 Java 5.0 时，实现单一 JVM 支持

JME, JSE, JEE

新的技术基础 (J9/TR 编译器)，以此为基础实现更好的性能、可靠性和可维护性

IBM 也投资并支持Java 公共创新

Eclipse, Apache (XML, Aries, Derby, Geronimo, Harmony, Tuscany, Hadoop ...)

广泛参与相关的开放标准 (JCP, OSGi)

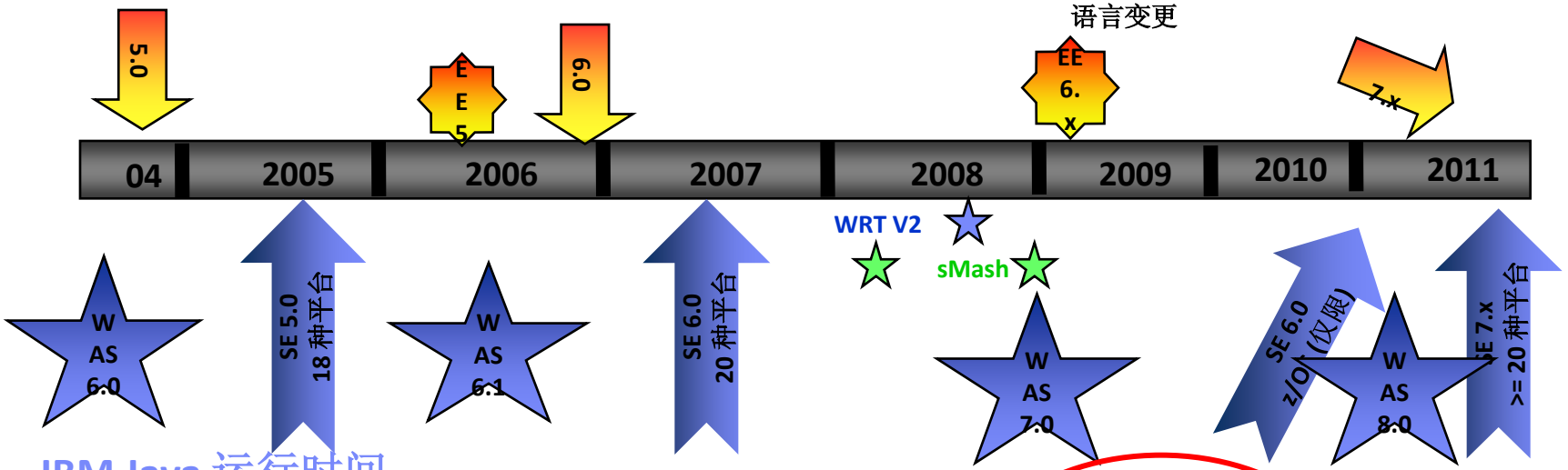
Java 路线图

Oracle Java 运行时间

Java 5.0
新语言功能：
自动类型包装
枚举类型
通用类型
元数据

Java 6.0
性能改进
客户端 WebServices 支持

Java 7.x
支持动态语言
改进SWING的易用性
新的 IO API (NIO2)
Java 存续 API
JMX 2.x 和 WS 连接，用于 JMX 代理
语言变更



IBM Java 运行时间

IBM Java 5.0
提高了性能
世代型垃圾回收器
共享的类支持
新型 J9 虚拟机
新型 Testarossa JIT 技术
第一套故障数据采集技术
全速调试
热代码替代
通用运行时间技术
ME, SE, EE

IBM Java 6.0
改进体现在
性能
可维护性工具
类型共享
XML 分析器改进
z10™ 开发
面向BigDecimal的DFP 开发
大页面
新的 ISA 功能

IBM Java 6.0.1
改进体现在
性能
GC 技术
z196™ 开发
OOO（乱序）流水线
70 多条新指令
JZOS/安全性强化

IBM Java 7.x
改进的可维护性和可消费性
改进的性能
虚拟机更新，支持语言变更

时间线和交付件可能会有变化。

IBM Java 6.0.1 on z/OS – 最新、最大

z196 和 Java6.0.1: 一同设计

使 Java 吞吐量提高了2.1 倍

减少了占地空间

与 z/OS 设施紧密集成

提高了应用行为的响应性

J9 R2.6 虚拟机

对 JIT 优化技术进行了大力增强

z196 可以利用指令及新的流水线

新的均衡的GC 策略，以缩短最大暂停时间

默认的GC 策略变更为gencon

z/OS 独特的强化

JZOS 2.4.0

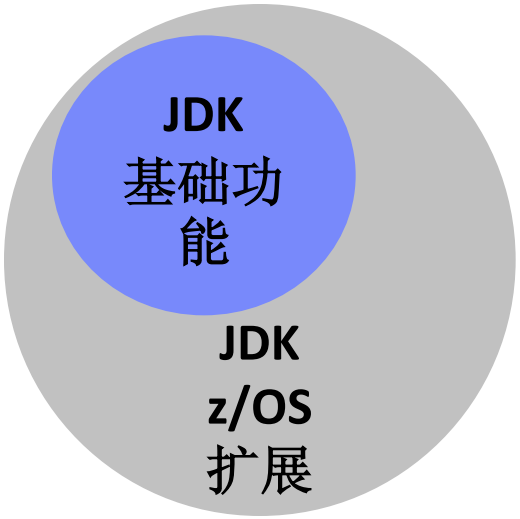
z/OS Java 独特的安全性强化

性能

多线程负载性能提高 2.1 倍

CPU 密集型负载性能提高1.93 倍

z/OS – zSeries 扩展



JZOS 扩展能够支持对 z/OS 功能的访问

- 传统的OS 数据集访问、VSAM 访问
- z/OS 控制台（修改命令和消息支持）
- z/OS 系统记录器访问
- Apache log4j 输出源(appender)
- JES 作业提交
- DFSORT
- SMF
- Java 流程按照批处理作业或者启动的任务来运行
- WLM 飞地
- ENQ
- PDS 和目录访问

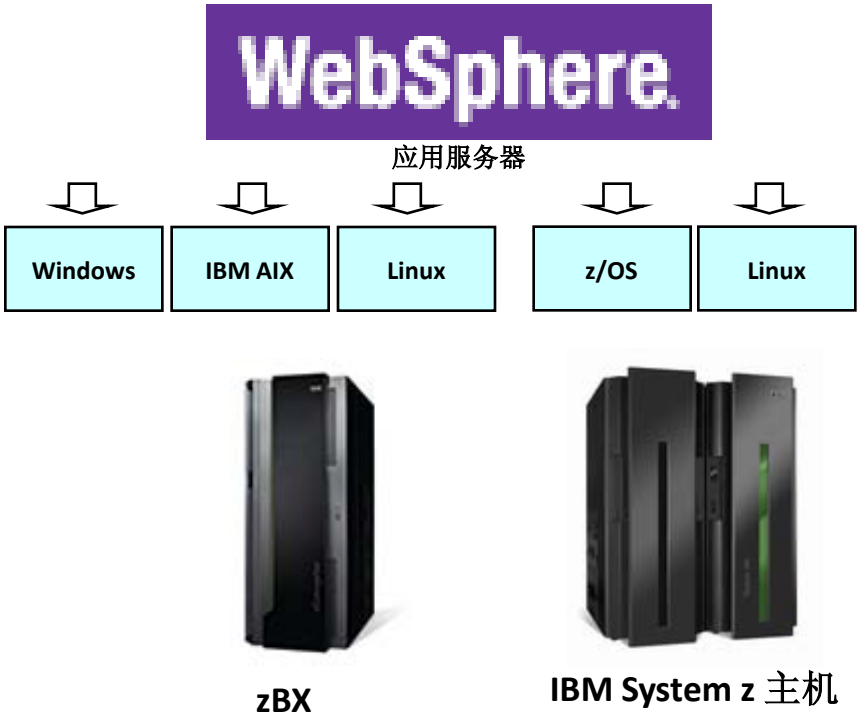
与系统相关的扩展能够让你编写健壮的、能够与传统的 z/OS 操作环境相集成的中间件和应用。

- 允许进行独立于平台的设计开发。
- 在需要时进行与平台相关的实施
- 允许进行操作及资源优化

http://www.ibm.com/developerworks/java/zos/javadoc/jzos/index.html

WAS 和平台设计

WAS 是一套跨平台产品

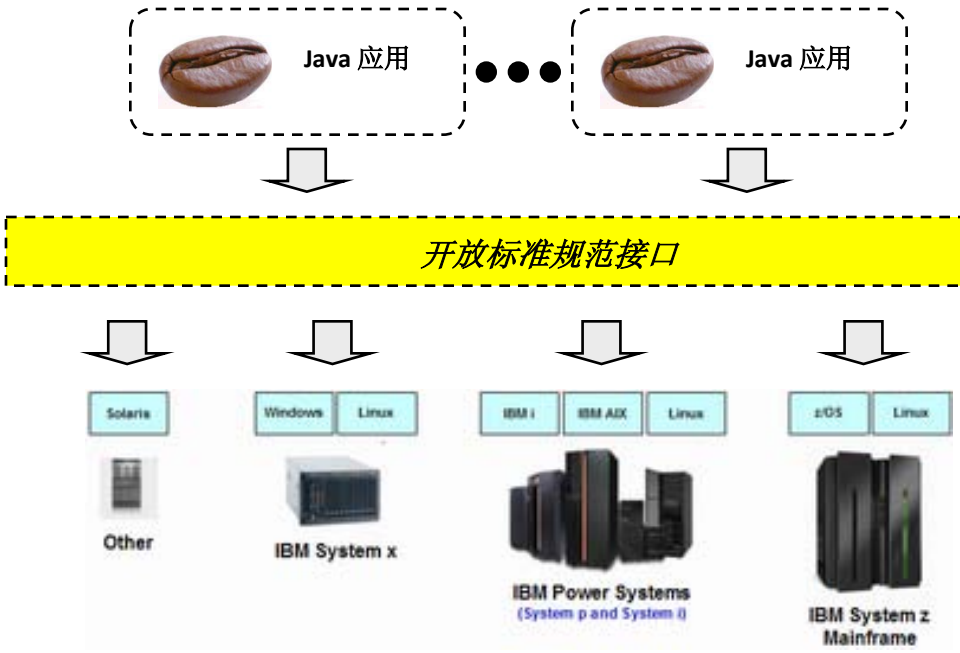


做出 zEnterprise 托管决策……

非常重要的起始概念

这一点怎么强调都不过分 -- 区别并*不*在于所提供的开放标准规范支持。

这在跨平台系统中很常见！



在规范接口线以上，“WebSphere 就是WebSphere”

Java EE, Java SE, EJB, Servlet, JSP, JDBC, JCA, JMS, Web 服务，等等

它是如何*实施*的取决于平台 … 其特性、功能、属性及服务质量

只有一套 WebSphere 代码库

代码能够检测到平台并根据情况调用与平台相关的开发。

*** 起始前提 ***

在规范线及以上，**WebSphere** 就是 **WebSphere**

因此，在*该层次上*不存在平台差异

差异出现在规范线*以下*

在这个层次上，在该平台上实施 **WAS** 时将利用平台的属性

我们希望集中于以下主题：

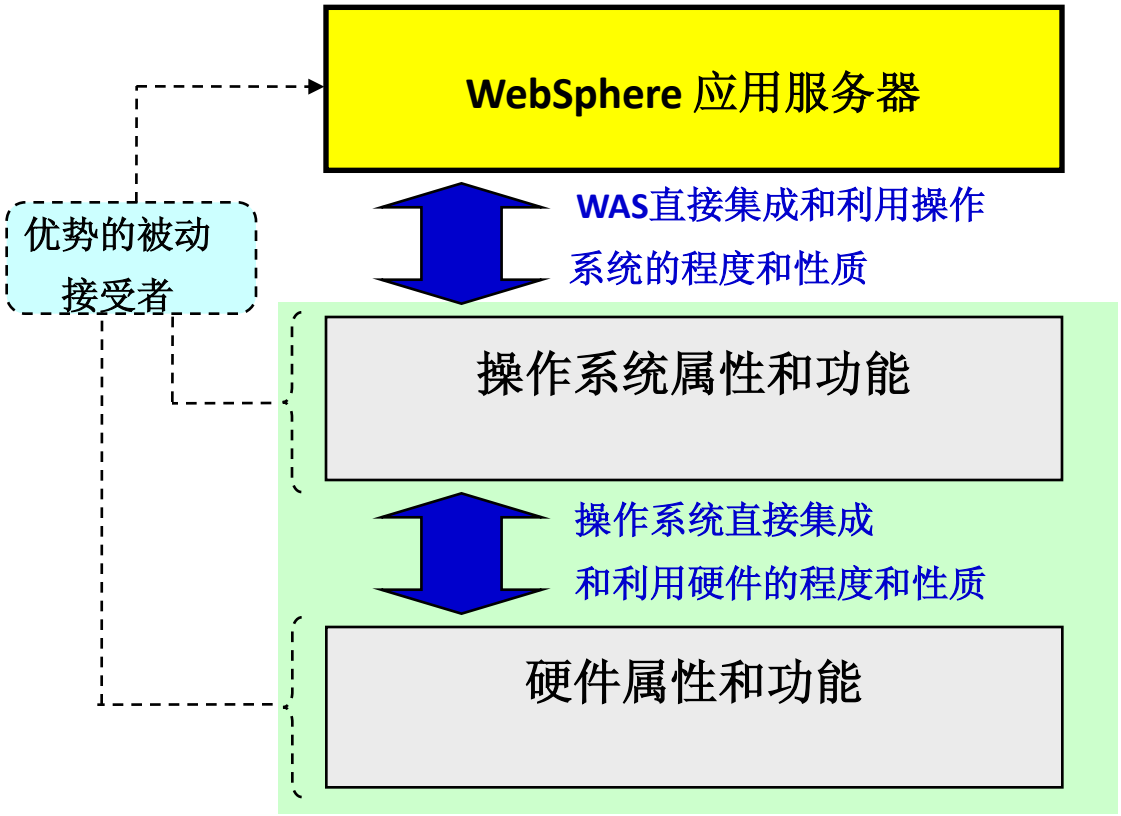
1. **System z** 和 **z/OS** 平台的质量和属性是什么？
2. **WebSphere** 应用服务器在什么程度上利用这些质量和属性？
3. 这些质量和属性如何有助于满足你的关键业务目标？

为什么采用 WebSphere on z/OS

进行了多层次的开发

我们需要了解，有来自硬件设计的优势，

有来自操作系统设计的优势，*还有来自它们之间的集成的优势*



并非所有的硬件设计都是一样的

并非所有的操作系统都是一样的

并非所有的操作系统都与

硬件有相同程度的集成

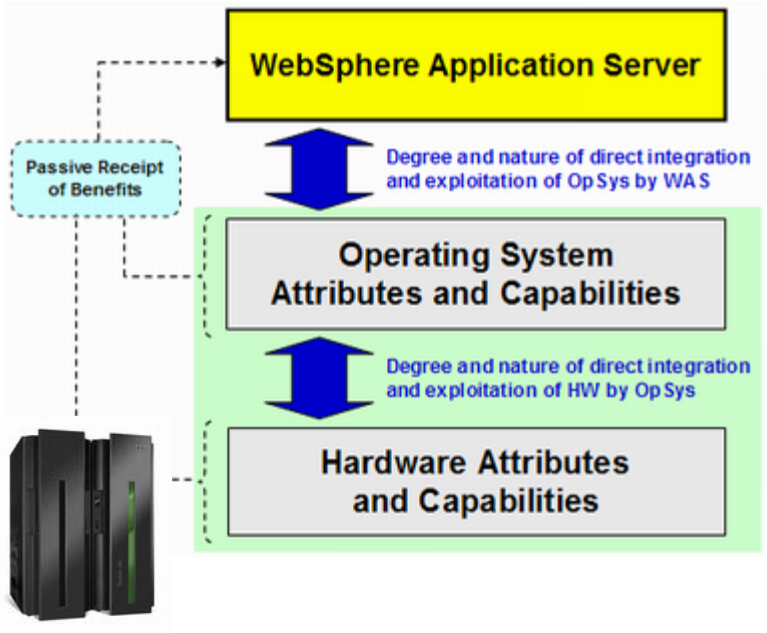
例如：z/OS 仅运行在 System z 上… …没有什么权衡来支持多平台灵活性。
操作系统针对硬件进行了优化；两者是一同开发的

这并不意味着System z 和 z/OS 适合于所有的情况

也不意味着其他平台和操作系统能够完成 System z 和 z/OS 所完成的工作

被动优势包含多个类别

运行在 System z 和 z/OS 上的程序在如下几个不同方面获得被动优势：



硬件

内在的设计成熟度和稳定性

冗余和灵活的更新

均衡的设计提供了非常高的吞吐量

通过 LPAR 实现成熟的、久经考验的虚拟化

操作系统

与服务器硬件设计紧密集成

极其成熟的体系结构

存储保护

负载管理器 (WLM)

Intelligent Resource Director (IRD)（智能资源目录）

本地 TCP 优化

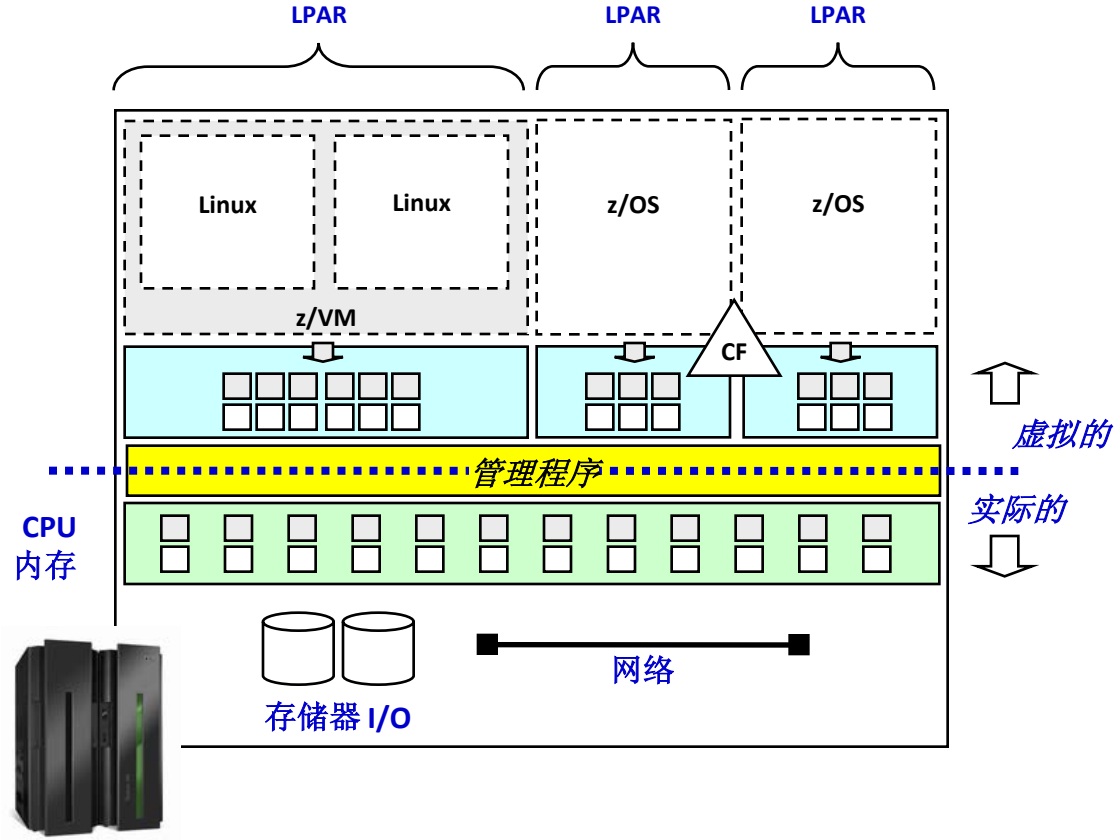
成熟的系统管理工具

久经考验的灾难恢复能力

我们在较高级的层次上考察其中的部分内容

硬件虚拟化 – 逻辑分区 (LPAR)

一种极其成熟的虚拟化技术，它能够使实际的资源在多个逻辑分区之间得到共享，各个分区之间完全独立



久经考验的虚拟化技术

多年久经考验的可靠性

把硬件划分为多个逻辑分区

利用 z/VM 以及客户机进一步实现虚拟化

各个 LPAR 相互之间完全独立

管理程序防止 LPAR 在为其分配的资源之外独占资源

彻底的内存隔离，因此无需担心相互覆盖

彻底的操作系统隔离，因此，所有的操作系统实例的元素都得到隔离

彻底的网络隔离，因此无需再担心安全性入侵

整合的优势以及隔离的好处

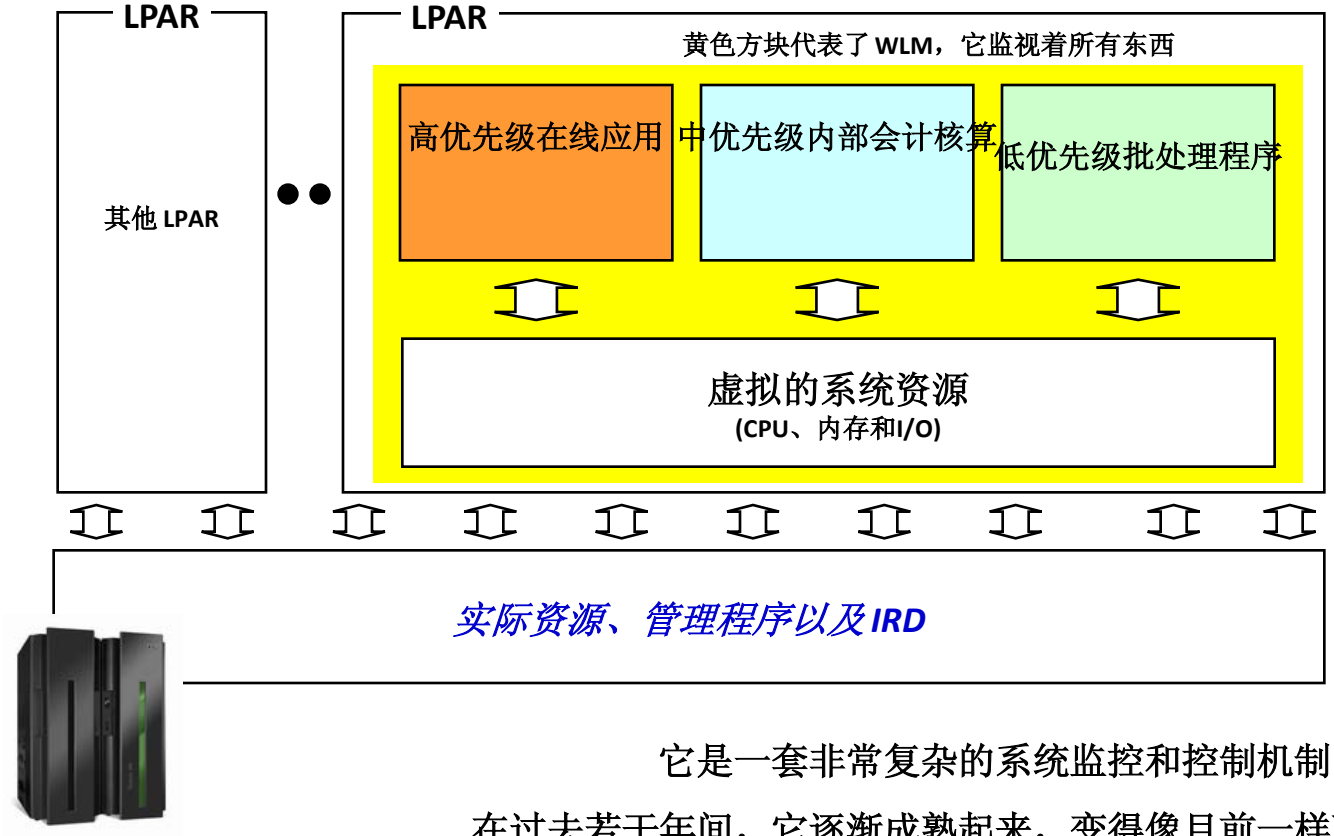
这是虚拟化涉及的全部东西。差异在于成熟度和能力。各种虚拟化方法之间的技术差异可能很快会成为一个复杂的问题。

这里的要点是，System z LPAR 具有久经验证生产记录

负载管理器 (WLM)

WLM是z/OS 操作系统的一个组件，它能够密切观察关键系统指标，

并管理各种资源，以实现你所确定的目标



这部分包含五项：

1.WLM 对整体系统资源利用率进行实时监

你确定的WLM服务等级目标，它决定了

2.WLM将如何管理资源

3.WLM 对每一个程序把你的服务等级

目标与实际的系统性能相比较

4.WLM重新分配LPAR中的资

源，以确保目标得以实现

5.WLM 通知 IRD 是否需要

在 LPAR 之间进行资源分配

它是一套非常复杂的系统监控和控制机制

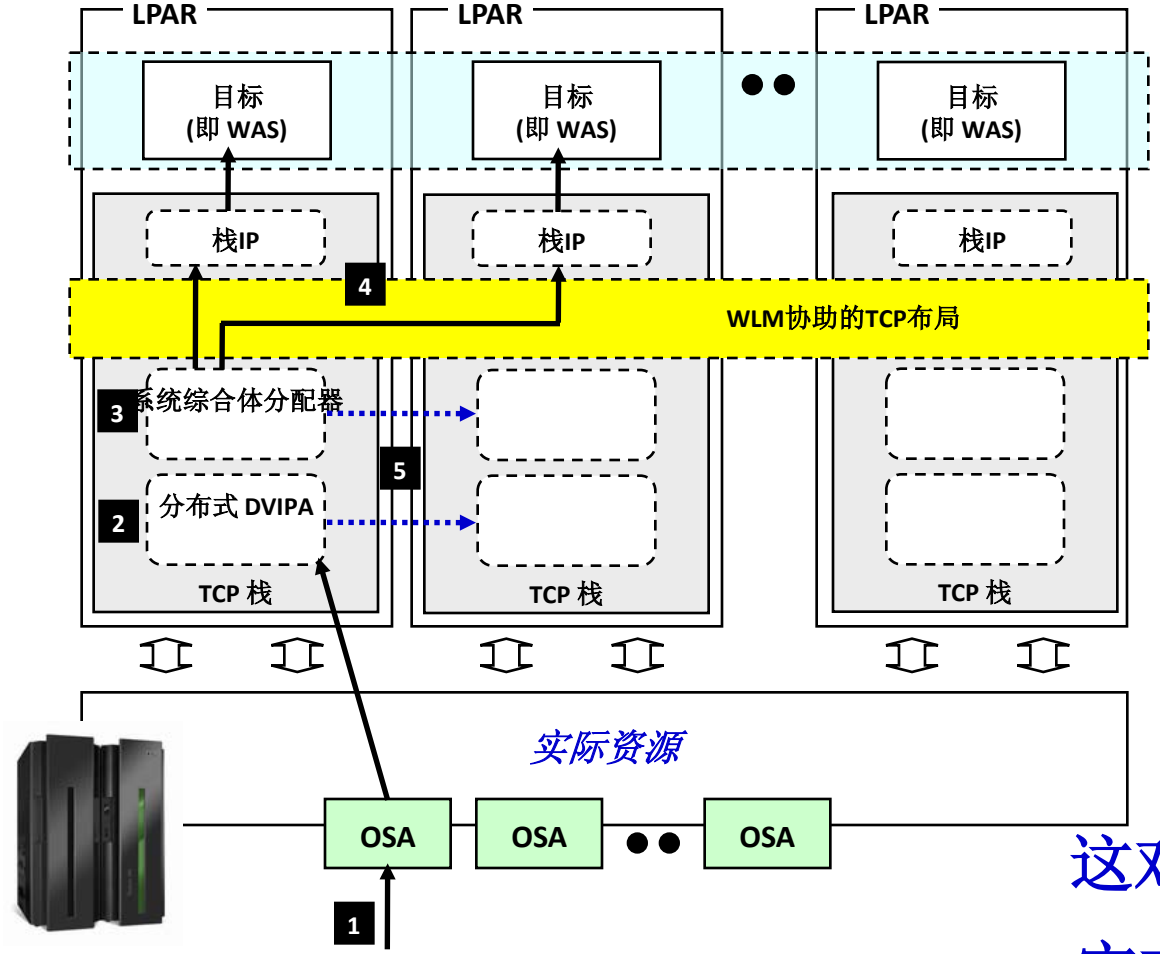
在过去若干年间，它逐渐成熟起来，变得像目前一样可靠、高效

其他的解决方案声称提供了“负载管理”能力，但与z/OS WLM 相比较，它们的功能通常太弱

虚拟IP 和系统综合体分配器

这是TCP 在z/OS上的一项功能，能够让你利用WLM协助的TCP

连接布局把重复的资源“隐藏”到单一的 IP 地址之后



本图中的内容：

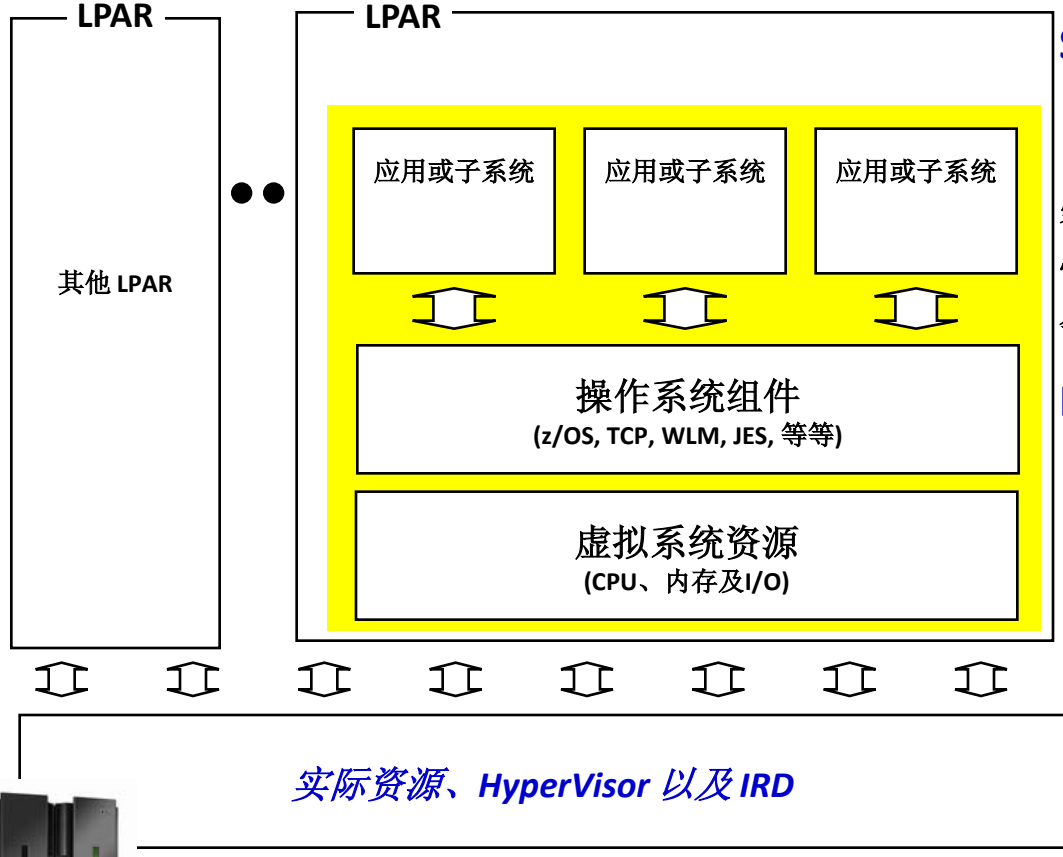
1. 实际的客户机把它们自己指向“类属”的 IP 主机名。路由器把它解析到机器上的某个适配器。
   注：有多种方法来获得冗余的 OSA 适配器，以实现可用性
   注：它未在本图中示出，但WLM 也可以通知某些非板载的(off-board) Cisco 路由器。
2. 请求被映射到系统综合体中的TCP 栈，它托管分布式虚拟 IP (DVIPA) 类属主机。
3. 系统综合体分配器功能能够决定哪个潜在的目标 LPAR 是在该时点接受新工作的最佳候选人。
4. TCP 连接在客户端与目标之间进行。
5. 如果托管的LPAR 或 TCP 栈发生故障，DVIPA 和系统综合体分配器的功能将自动移动到事先定义的“下一个”栈中。

这对于应用是透明的 …

它在该过程中是被动的

资源和系统监控 -- RMF 和 SMF

为了有效、高效地管理你的服务器环境，你需要了解谁在什么时候使用什么东西



SMF

系统用于把记录写入系统数据库的一种功能。

这些记录随后可能被用于分析系统使用情况，以便用于：

容量规划

性能规划

会计核算和计费

RMF

把 SMF 写入关于关键系统活动报告的另一种功能。

对于制定规划和调查报告非常重要

如果该平台更容易得到管理，

那么该平台的用户就可以

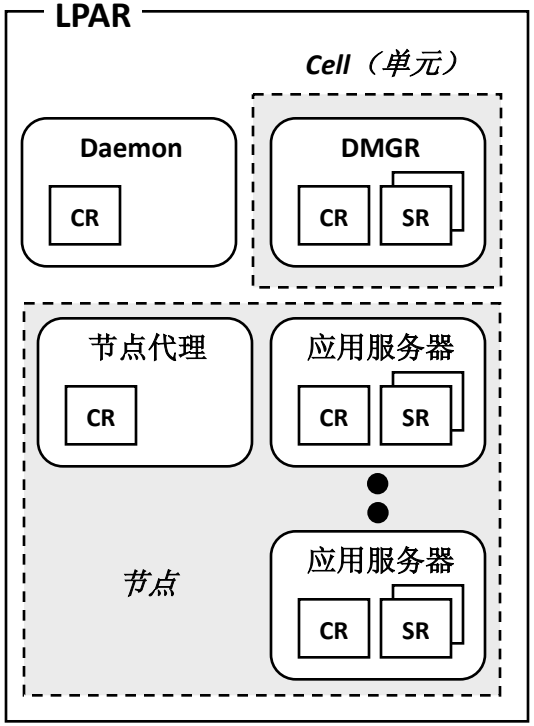
由此得到间接好处

注：WAS z/OS 自动利用SMF。

利用 JES 和共同的 z/OS 功能

这意味着，现有的z/OS 系统编程人员将可以很舒服地

进行关键的WAS z/OS操作 … 它符合他们现有的技能



WAS z/OS 运行时间实施为一系列启动的任务

使用标准的JCL 和 START 命令

JCL START 进程在PROCLIB中进行维护

输出结果默认地写入JES

JES 管理输出和存储

启动的任务和地址空间可以像任何其他程序一样得到显示

像任何其他程序一样启动和停止

能够使用 MODIFY 命令进行动态操作

配置保存在HFS 或 ZFS 文件系统中

能够让系统自动化任务控制操作

这完全是标准的东西。关键是，实施WAS z/OS

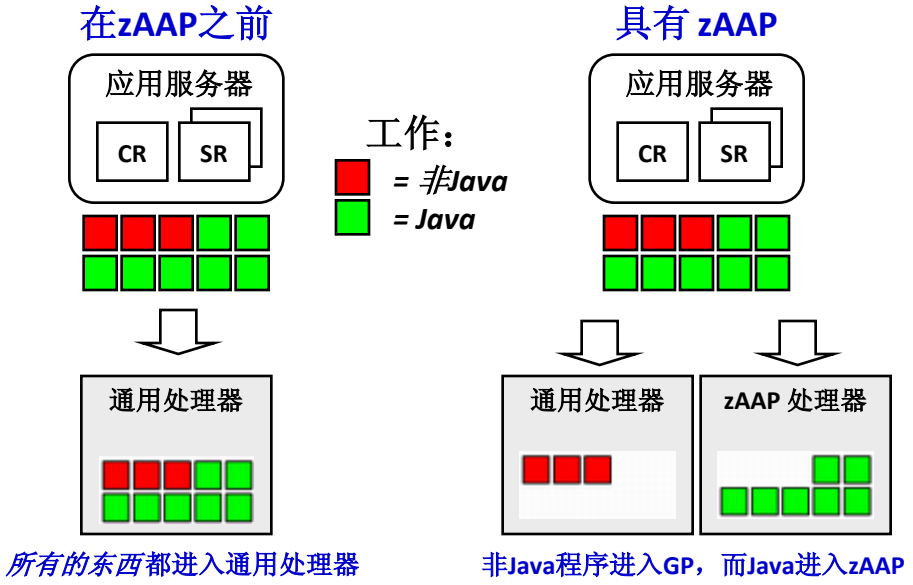
以便与现有的z/OS技能相兼容，并利用现有的z/OS功能。

WAS z/OS 不仅仅是运行在USS中的UNIX流程。

利用zAAP 专业引擎

zAAP 引擎是Java 卸载引擎。它们增强了z/OS 平台的财务状况，

而且它们释放了GP用于其他关键子系统处理



理解 zAAP 价值的关键：

- 与GP相比较，zAAP 处理器具有非常低的采购成本

把Java负载转移到zAAP 通常可以使越来越多的非Java

- 工作存续于现有的GP中，这样就可以避免为购买而支出资
- 以系统容量为基础的月度许可证费可能会受到存在 zAAP的影响，zAAP不计入收费中

这里还有许多技术细节未得到叙述,涉及到如何配置它们、调度规则如何以及 Java 在什么时候可以进入GP，等等。

这里的目的在于关键点，而不是细节。

这确实是Java SDK以及z/OS调度器的一项功能。

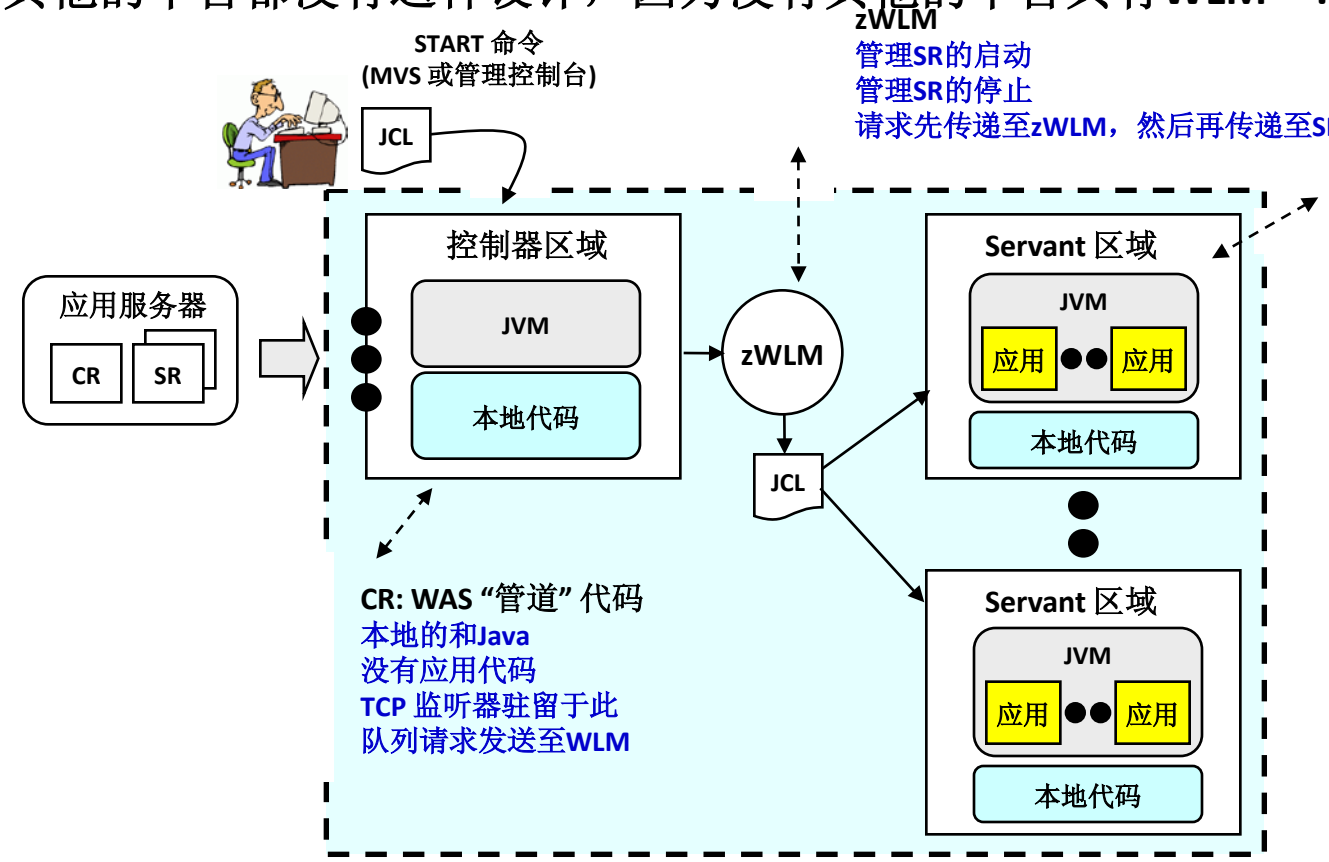
支持zAAP的Java SDK与 WAS z/OS包装在一起，因此WAS可以

自动利用zAAP，如果后者存在并得到配置的话

控制器 / 伺服器(Servant) 体系结构

这是WAS z/OS 设计中的一项独特的体系结构元素。

其他的平台都没有这种设计，因为没有其他的平台具有WLM:**

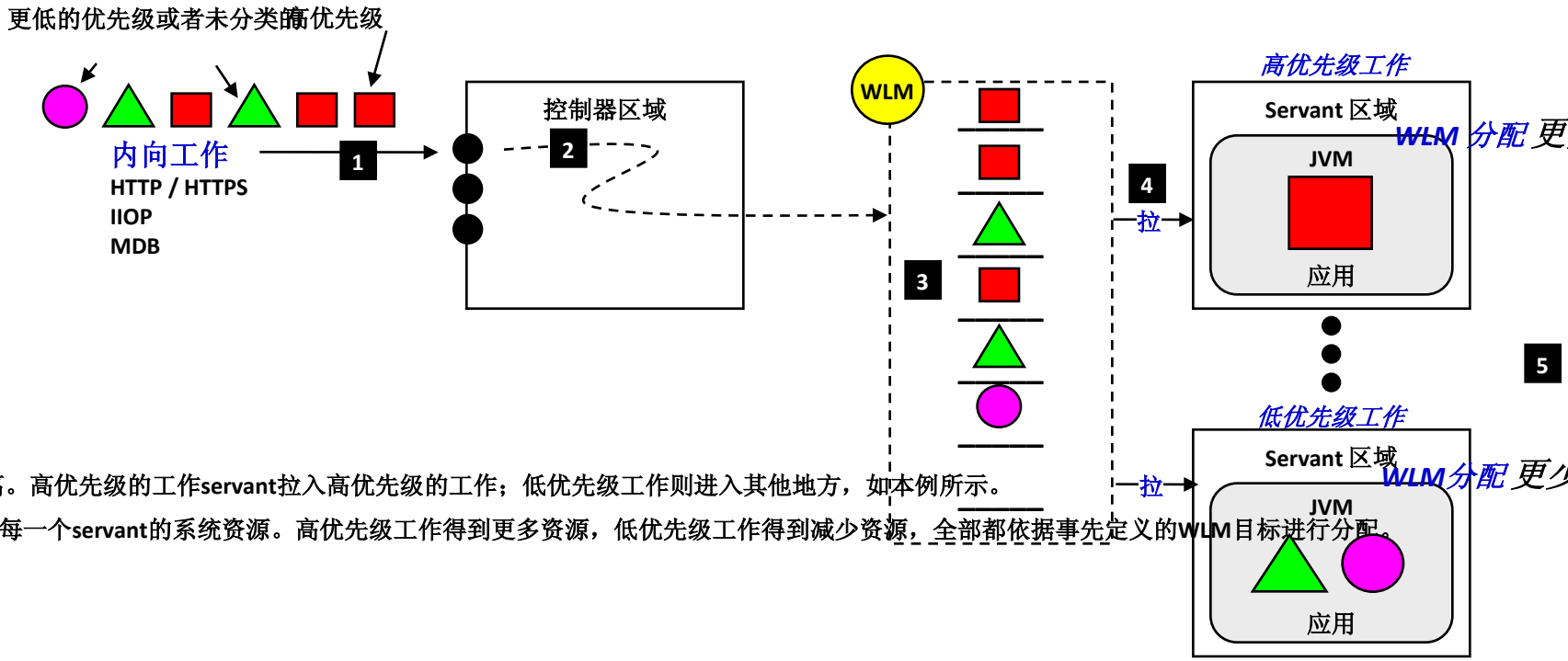


我们来看看这是如何实现的 ...

** WebSphere 在分布式系统上时使用 “负载管理”这样的说法，但它与zWLM不同

对服务器中混合工作的智能管理

工作提供一个“事务分类”。利用该功能，CR可以把工作引导至伺服器(servant)，而且WLM 可



隔离。高优先级的工作servant拉入高优先级的工作；低优先级工作则进入其他地方，如本例所示。

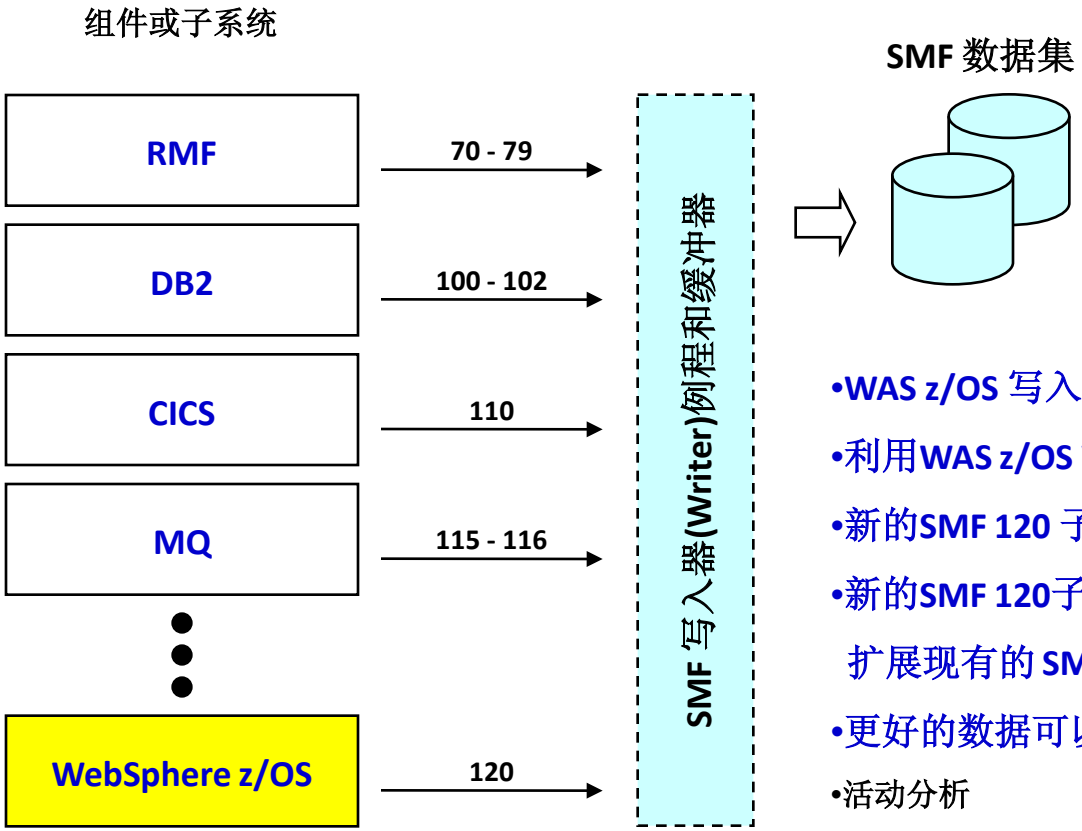
配给每一个servant的系统资源。高优先级工作得到更多资源，低优先级工作得到减少资源，全部都依据事先定义的WLM目标进行分配。

成熟的工作优先级安排

在其他平台上，这只能通过把工作分配到单独的服务器上来实现。它们没有WLM在这个层

利用 SMF

SMF 是z/OS 的一项活动记录功能，能够让子系统记录关键活动，以便进行分析、管理及会计计费



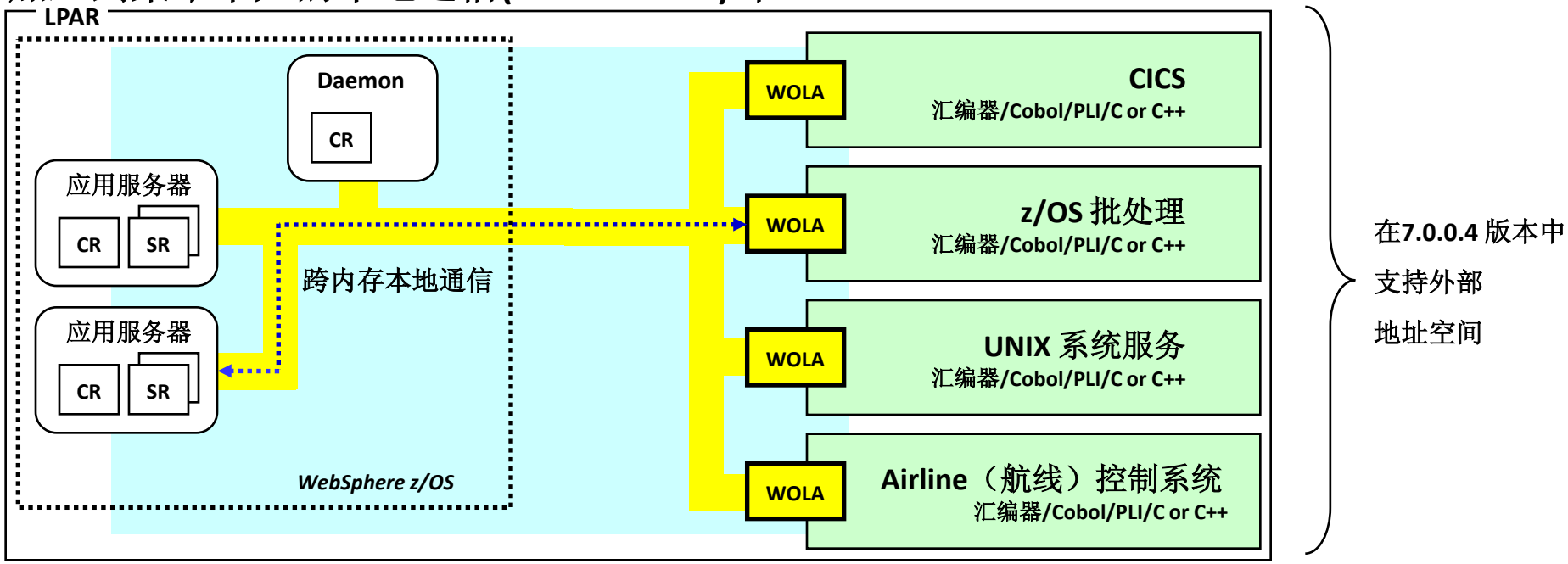
- **WAS z/OS 写入 SMF 120 条记录**
- **利用WAS z/OS V7，创建了一种新的子类型：9**
- **新的SMF 120 子类型 9 以更低的开销成本提供了更好的数据**
- **新的SMF 120子类型9 记录了补充内容，并从其他子系统中扩展现有的 SMF，从而能够更好地理解正在发生的事情**
- **更好的数据可以用于 …**
- 活动分析
- 使用量统计
- 会计计费

http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101342

跨内存：新型优化的本地适配器 (WOLA)

在V7.0.0.4 中提供的这一新功能能够让外部地址空间

加入到某个单元的本地通信(Local Comm) 中。



优势：

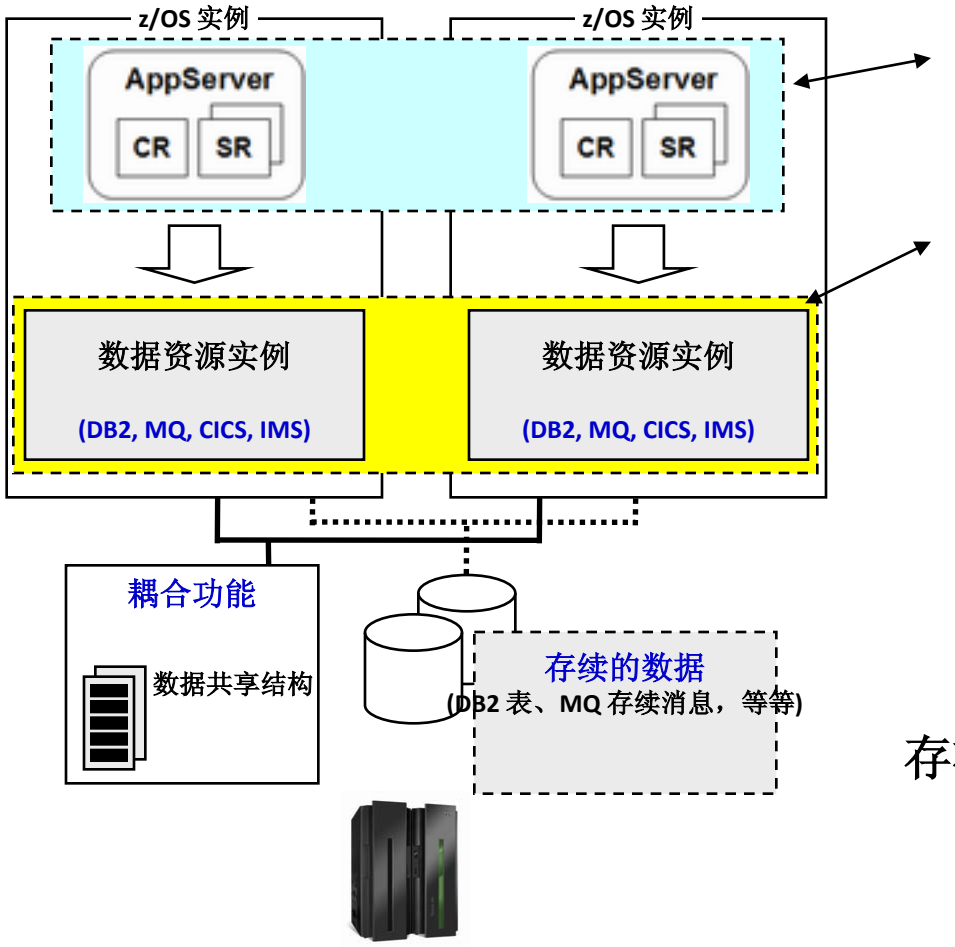
- 基于本地通信(Comm) (z/OS 除外)
- 双向 … WAS 外向或内向进入WAS (WOLA 除外)
- CICS 安全性和事务传播 (适用某些限制)
- 比其他本地解决方案速度更快

WP101490 在 ibm.com/support/techdocs 上可以找到详细信息

核心 -- 系统综合体数据共享

并行系统综合体和数据共享提供了对相同数据的重复访问能力。

数据访问和锁定问题由耦合功能和子系统提供。



WebSphere "集群" 由多个物理应用服务器构成

它们在多数情况下在物理上是分离的。

它们共同代表了一个逻辑服务器。

共享群组

是物理上分离的实例，

但在组织上可以使他们以群组方式理解参与者，

并具有定义的共享关系

AppServer（应用服务器）和资源实例

AppServer中的应用与数据资源*实例*进行交互。

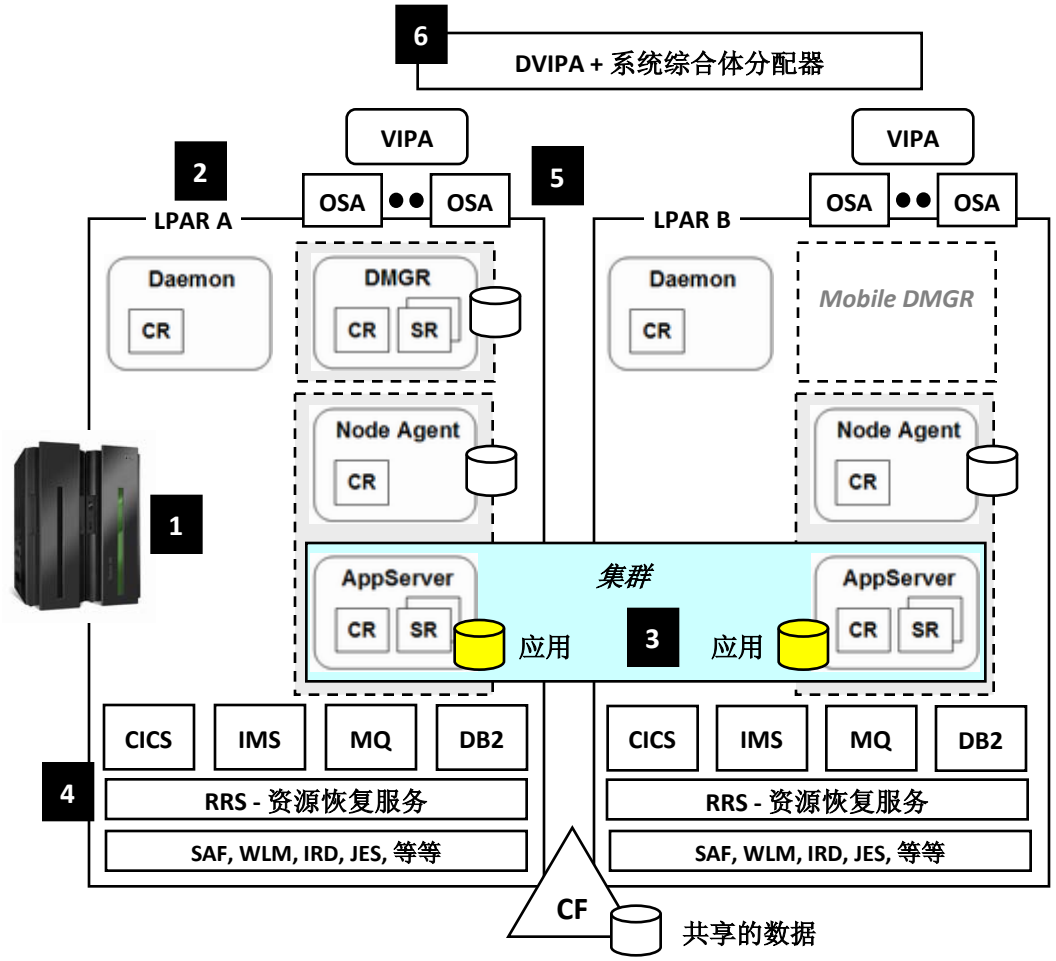
共享冲突由数据资源实例相互配合，

在z/OS和耦合功能的帮助下得到解决。

并行系统综合体和数据共享已经存在十多年了。该技术很成熟并得到考验，并由世界各地的大型客户使用。

WAS z/OS 和并行系统综合体高可用性总体情况

这涉及到冗余*以及*与平台高可用性功能之间的整合



- ■冗余、容错的硬件
  - System z 硬件设计具有多层容错及冗余能力。
- ■冗余的z/OS实例
  - 或者通过逻辑分区 (LPAR)，或者通过单独的物理机器。
- ■群集的 WebSphere z/OS 服务器
  - 多台应用服务器组合为一套逻辑单元，用于应用部署和管理
  - z/OS 除外：动态SR扩展（更多的即将提供）
- ■冗余数据资源管理器以及系统综合体共享数据
  - 在CF中以及一个全局同步点管理器(RRS)中多重资源管理器实例以及共享的数据
- ■隐藏在虚拟 IP 地址之后的冗余网络适配器
  - 在前台，具有可移动的虚拟 IP 地址的多种网络接口能够防止中断
- ■隐藏在分布式虚拟 IP 及系统综合体分配器之后
- ■的负载分布
  - 对隐藏在虚拟IP之后的实际IP地址的进一步抽象，这样的虚拟IP地址可以在系统综合体中的影像之间进行调换，而且系统综合体分配器提供了基于 WLM 的TCP 连接分配

我们展示了两种操作系统实例。这可以进一步提高性能，

以便实现更大的可用性以及更易于管理的故障隔离能力。

高可用性战略图景

它*不仅仅*是技术 ...

技术

网络

应用

中间件 / 数据

并行系统综合体

存储器

物理的 / 环境

监控和管理

容量管理

性能管理

系统组件监控

应用监控

端到端监控

流程控制和治理

资产库存管理

变更管理

故障查找和解决管理

反应性流程/ 预测性流程

高可用性是一个大问题

它完全是一个系统问题

... 关注某一项设计元素并

不能保证

实现高可用性

它是个技术问题，

但同时也是个流程问题

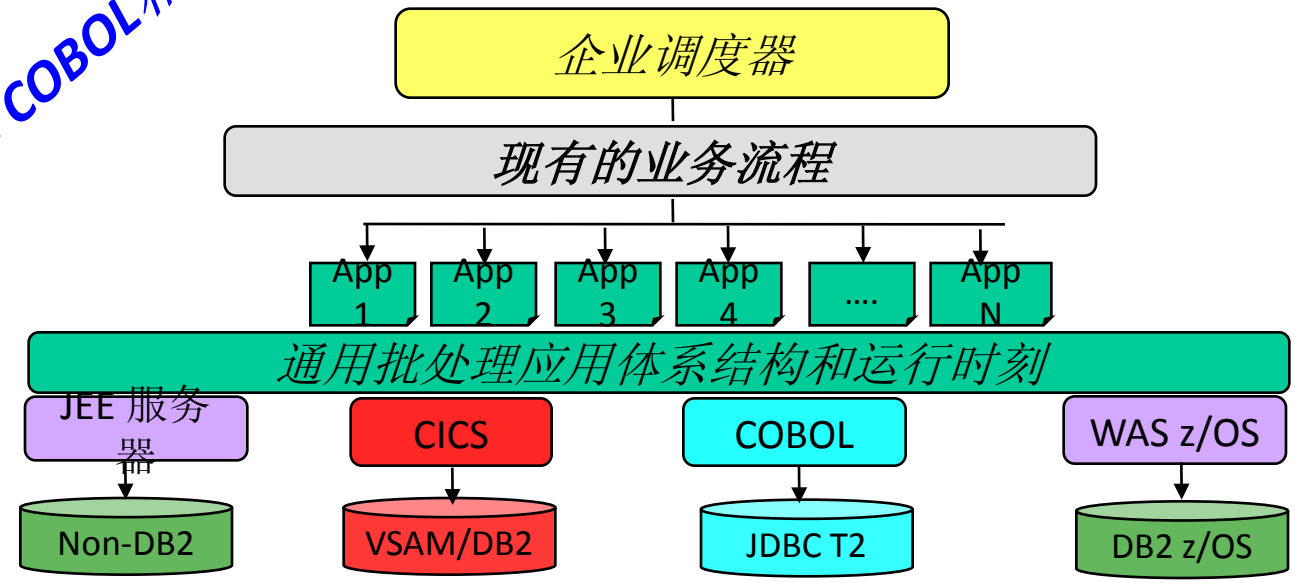
重要的第一步是确定企业

真正需要什么程度

的高可用性

与CICS 和 COBOL相集成

批处理愿景



批处理容器应当运行 *于各处*

跨越各种平台及 J2EE 供应商的 ***可移植的批处理应用***

数据的位置决定了批处理应用的布局

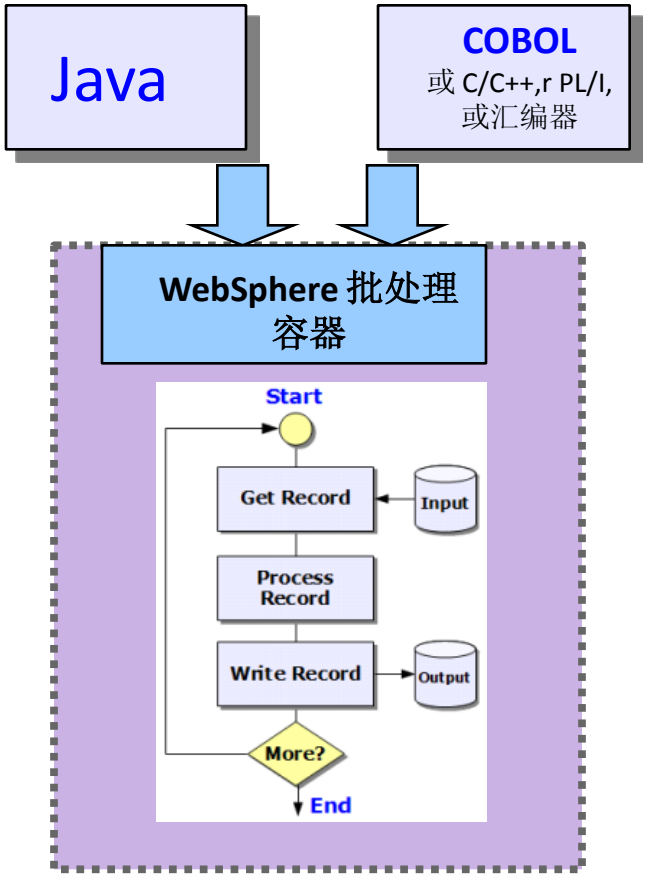
由我们的企业调度器集中管理

与现有的解决方案相集成：灾难恢复、审计、日志记录、归档

WebSphere 批处理功能包和XD 计算网格

面向现代批处理的功能包（现在是WAS V.8的一部分）

- 批处理容器环境
- 编程框架
- 作业调度器和分配器(dispatcher)功能
- 声明性作业控制文件(xJCL)
- 开发类库
- 批处理数据流 (BDS)
- 有条件的多步骤作业支持
- Ckpt 处理利用WAS 事务管理器



WebSphere XD 计算网格

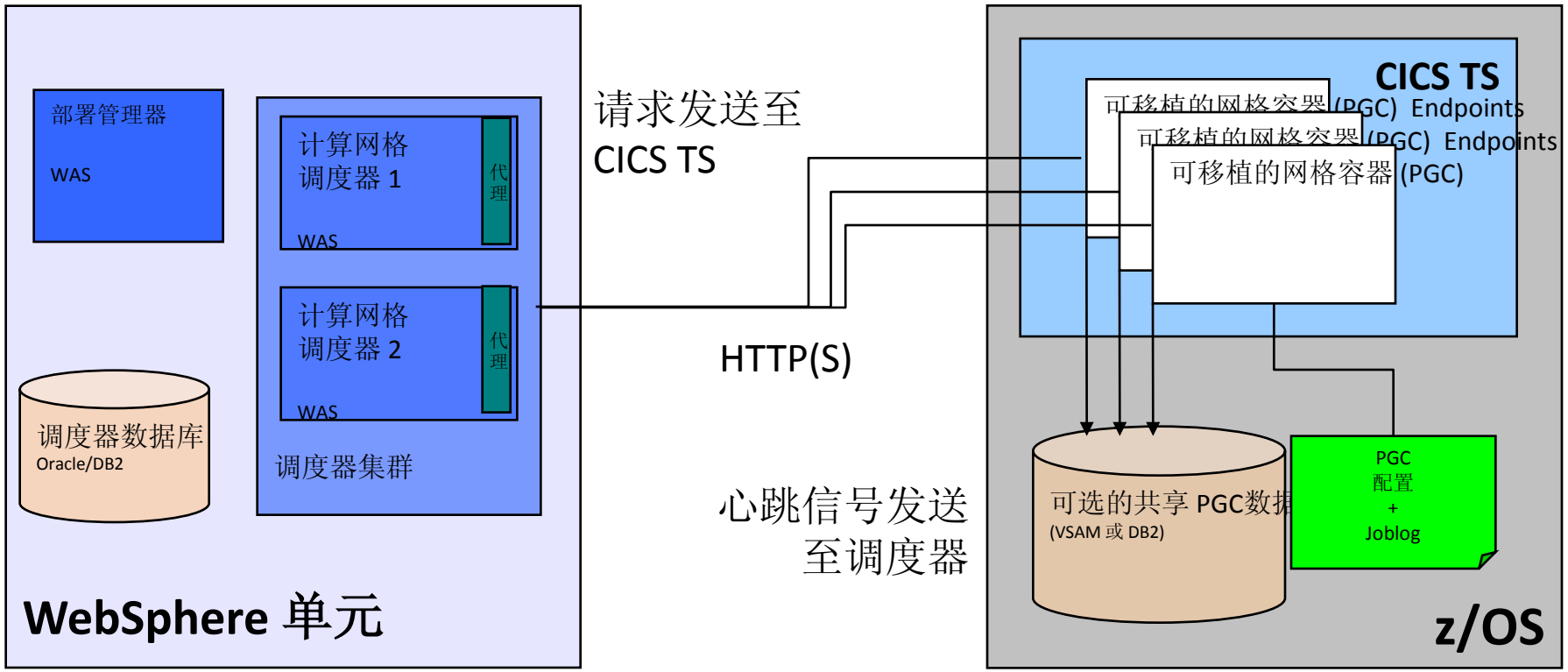
你在"面向现代批处理功能包"中看到的所有东西以及 ...

- 作业的日历及时钟调度
- 与扩展的调度器产品相集成
- SMF 120.20 & 9 使用报告
- *根据作业*对WLM 事务进行分类
- 应用静默及更新
- 作业提交节奏(pacing) 及限制(throttling)
- 并行作业管理和调度
- 与COBOL 和 CICS相集成

计算网格和CICS以及SupportPac CN11

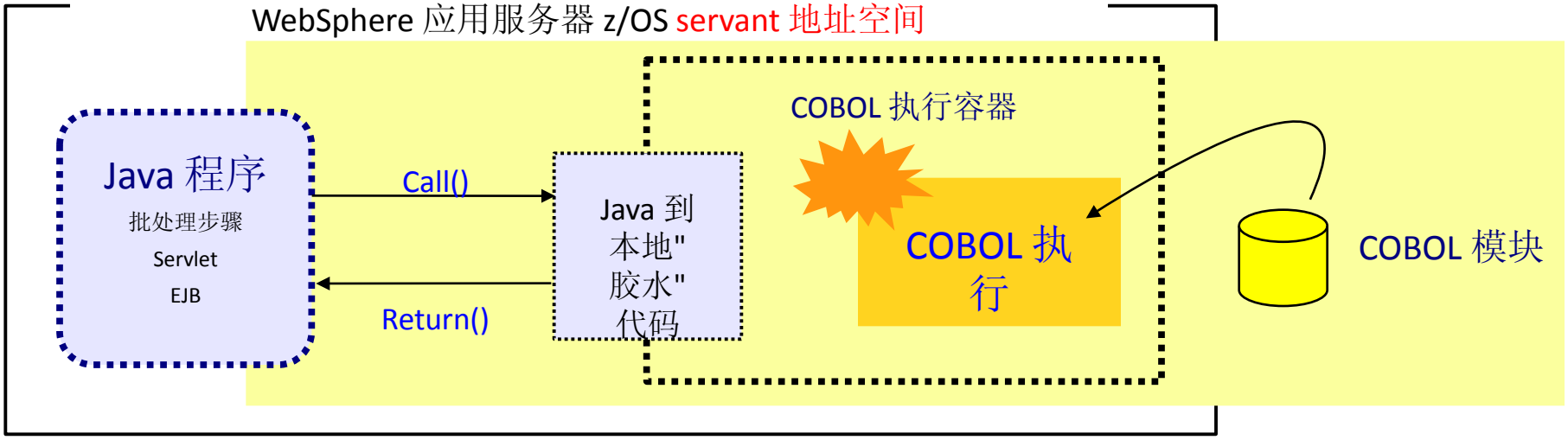
计算网格中的批处理作业发送HTTP 请求至CICS以启动事务程序。

CICS 根据CN11URIresource 中的路径匹配它，以处理该请求。



计算网格COBOL容器

“COBOL 容器" 提供了JNI 服务：



要点：

在servant 地址空间中反复创建和销毁COBOL容器

COBOL 容器的LE飞地与地址空间的LE飞地相隔离　(干净环境）

JDBC T2 连接可以在Java与COBOL程序直接得到共享　(利用 RRSAF维护事务处理背景)

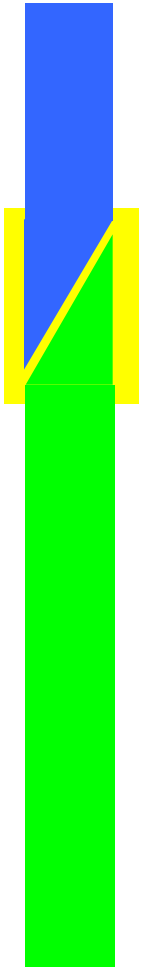
我们看到有两群人 -- WebSphere管理员和z/OS管理员 … 这两群人会合于何处？

与应用相关的
- 设计
- 开发
- 部署
- 与应用相关的管理控制台
- 与应用相关的WSADMIN

运行时间*操作*
- 启动及停止服务器
- 服务器踪迹及日志的默认位置
- 配置 z/OS 规范

示例：为本地模式配置数据连接器

运行时间*创建和系统管理*
- 高级概念都是人们所熟悉的
- WebSphere z/OS 需要创建和提交定制的JCL 批处理作业
- 备份/恢复，保证适当的系统资源、容量规划 … 所有的传统z/OS系统程序员任务



WebSphere管理员

有关WebSphere管理的常见任务在各个平台之间都是共同的。

这是重叠地带…

z/OS管理员

在创建及运行层次上，WebSphere z/OS

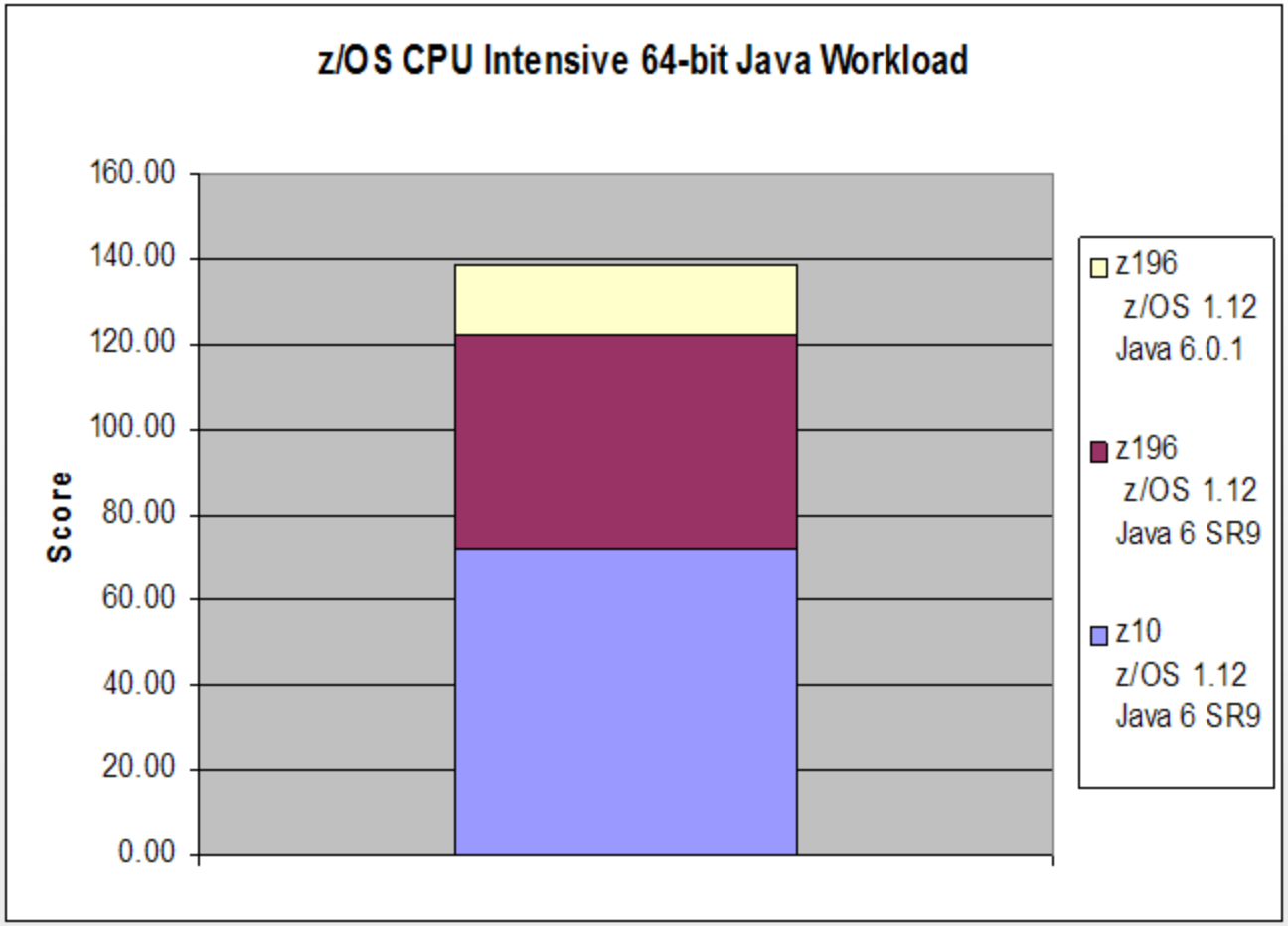
只是一系列启动的任务，因此应当为z/OS管理员所熟悉

任何社区都无需成为另一个社区的专家

一般而言 … z/OS系统程序员经常会转变为运行时间设置管理和应用部署的角色

关于WebSphere性能需要讨论什么?

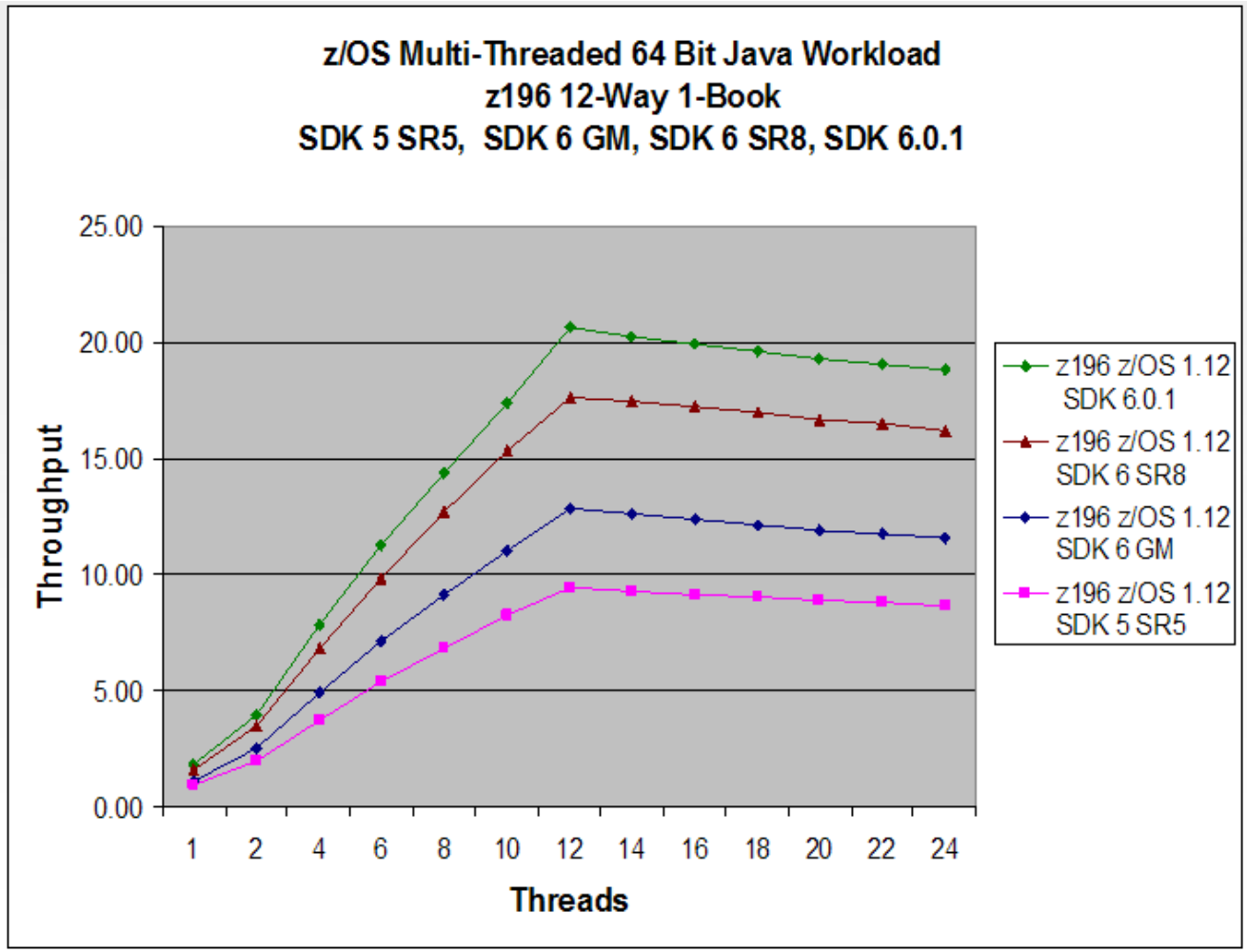
在z/OS上的性能: CPU密集型基准测试



93%的累积改进

- 14%的Java 6.0.1改进
- 70%的硬件改进

z/OS Java SDK 6.0.1 性能: 64位多线程基准测试



2.17倍的累积软件改进

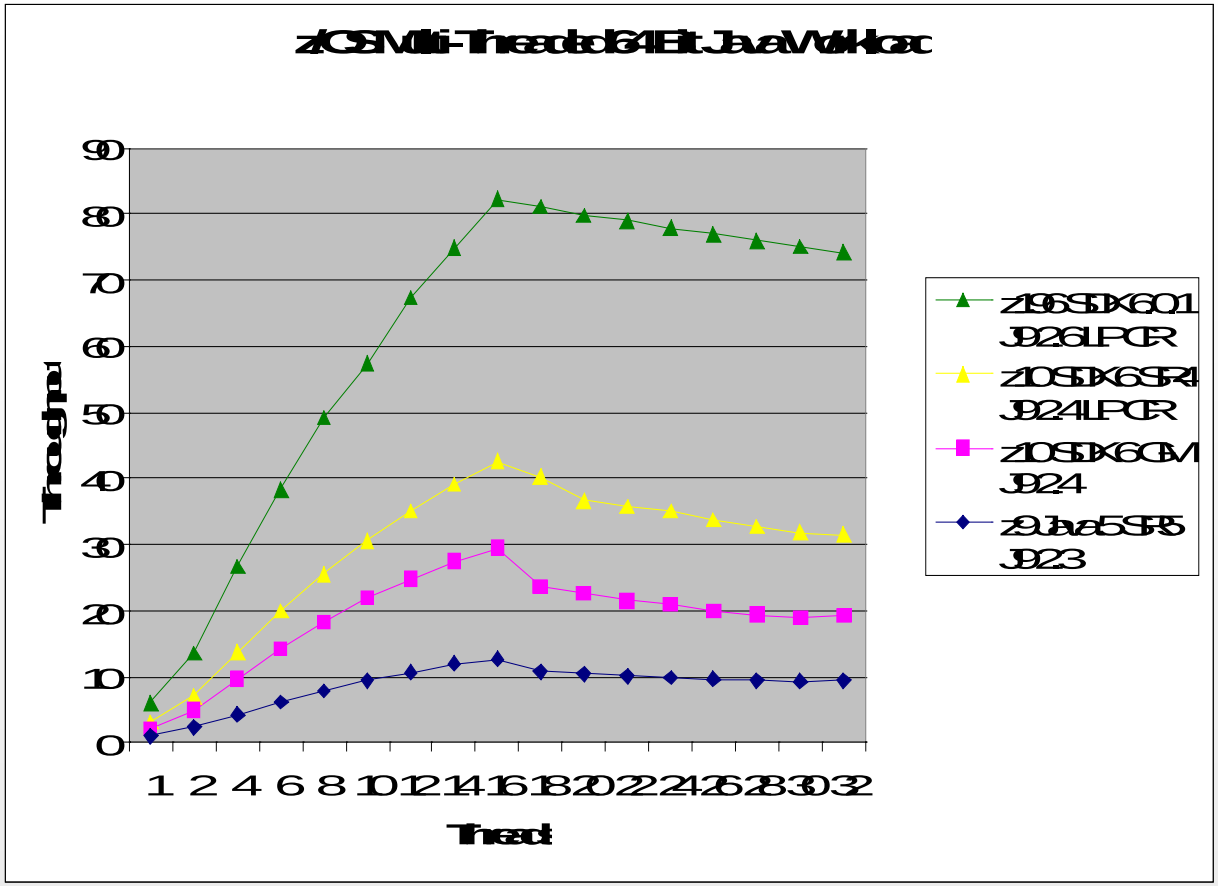
16%的Java 6.0.1改进

39%的 Java 6 SR8 改进（相对于Java 6 GM）

35%的Java 6 GM改进（相对于Java 5 SR5）

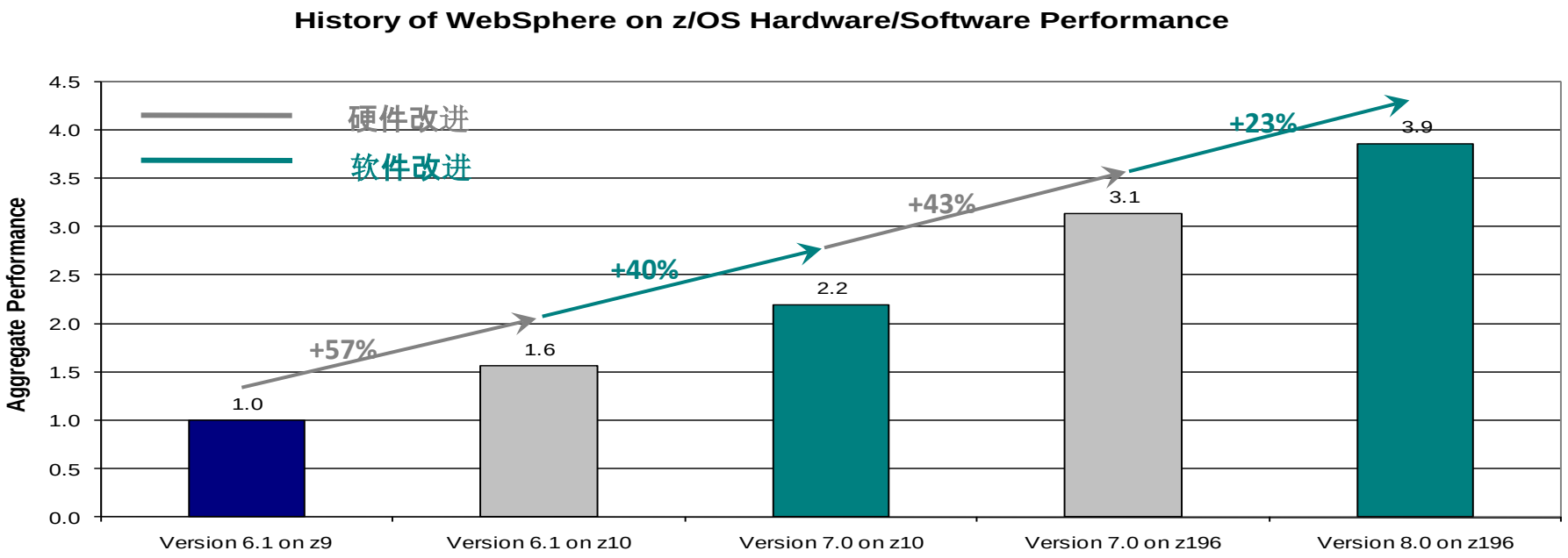
z/OS Java SDK 6.0.1 性能:

累积硬件及SDK改进，从z10、z196、Java6到Java6.0.1



大约7倍改进，从z10、z196、Java6到Java6.0.1

z/OS上的性能: WAS 在 z/OS上



本图示出了从zSeries硬件（从z9到z196）和软件（从V6.1到V8.0）获得的改进历史。这些数据源于利用DayTrader EJB负载获得的评测结果。

本图显示，从z9上的WAS V6.1到z196上的 WAS V8.0，累积性能改进几乎达到4倍。

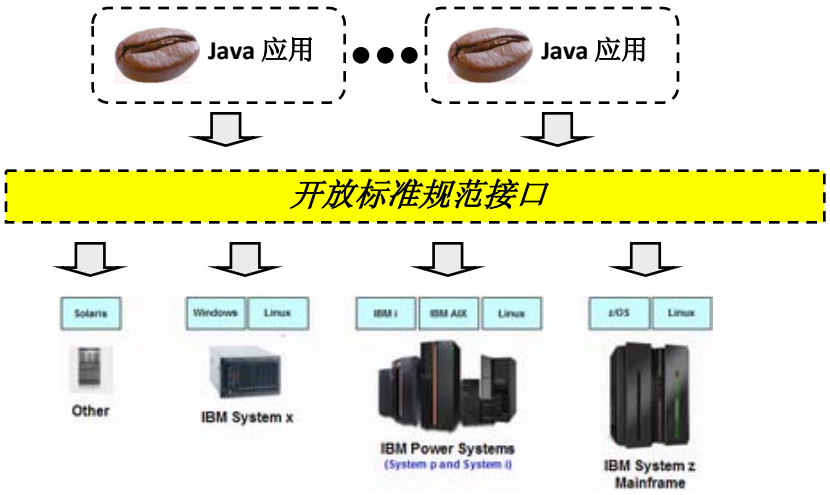
硬件组件的提高大约是2.25倍(1.57 x 1.43)

软件组件提高是1.72倍(1.40 x .123)

回答最初问题 – 在zEnterprise内部什么地方

在规范接口线以上，

“WebSphere 就是WebSphere”



它是如何*实施*的取决于平台 ...

其特性、功能、属性及服务质量

*** 现实 ***

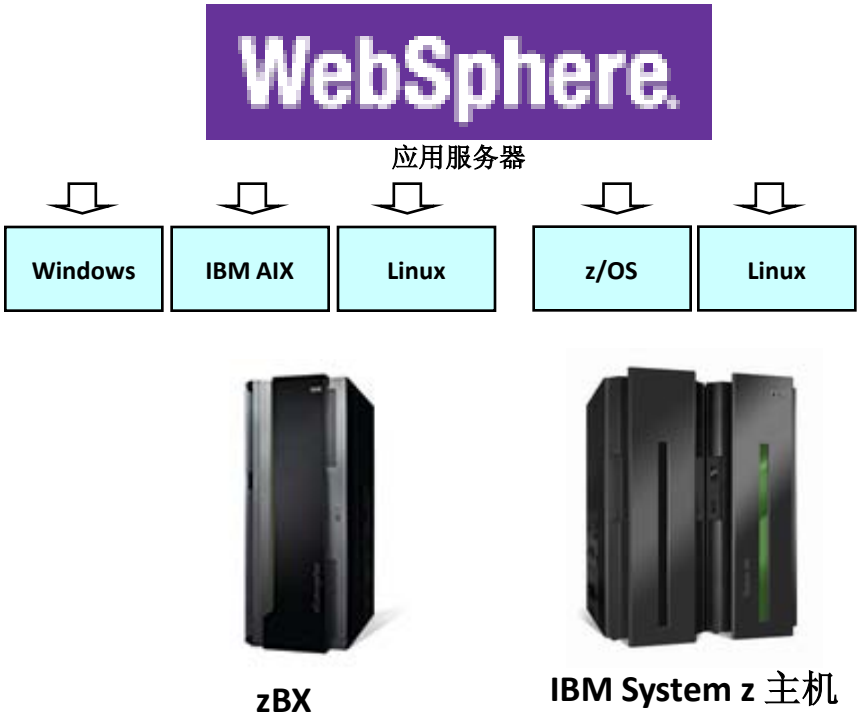
并不是每一套应用的

每一部分都将运行在z/OS上，原因如下：

- **WebSphere**实施不影响某些负载类型的性能和容量。
- **AJAX** 输入预测**(type-ahead)**
- 应用含有一些仅在特定平台上得到支持的组件。
- **eXstreme Scale**（巨大规模的）服务器
- 有些部署体系结构使用了无状态的、可处置的前端流程。
- 没有业务集成的负载，或者与**z/OS**数据没有密切关系的负载。
- 协议转换器 / **DP**
- 政策法令

为什么在zBX使用WebSphere

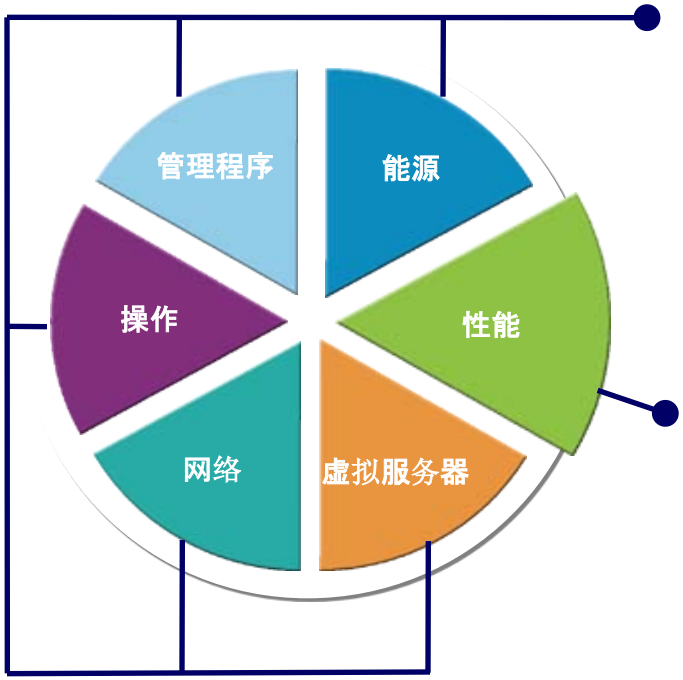
为什么在 zBX / zEnterprise上部署WebSphere服务器?



因为zEnterprise管理功能提供了操作结构和工具！

zManager组件

为WebSphere负载提供操作协作



“管理”套件

- 提供了此饼状图所示出的大部分功能
- 利用“管理”套件，你可以在管理程序及虚拟服务器的层次上*监控*资源利用率，但你无法通过目标导向型的管理使其自动化。

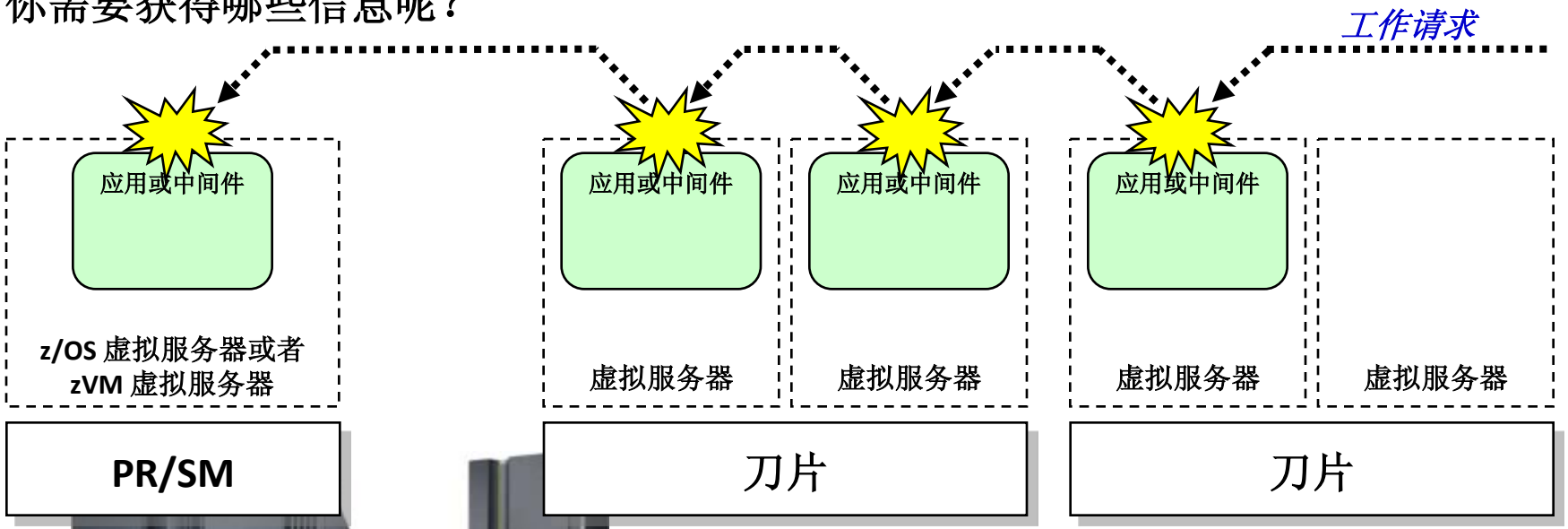
“自动化”或者“高级管理”

- 可选的功能套件，它提供了我们在本单元中所讨论的目标导向型性能管理
- 高级管理涵盖了System x刀片，使所有其他东西实现自动化。

负载管理与网络管理及存储管理相结合，能够实现更好的端到端管理，协作实现服务等级协议。

起始... 总览

设想你是zManager，你希望管理和控制在平台之间跳来跳去的负载。
你需要获得哪些信息呢？

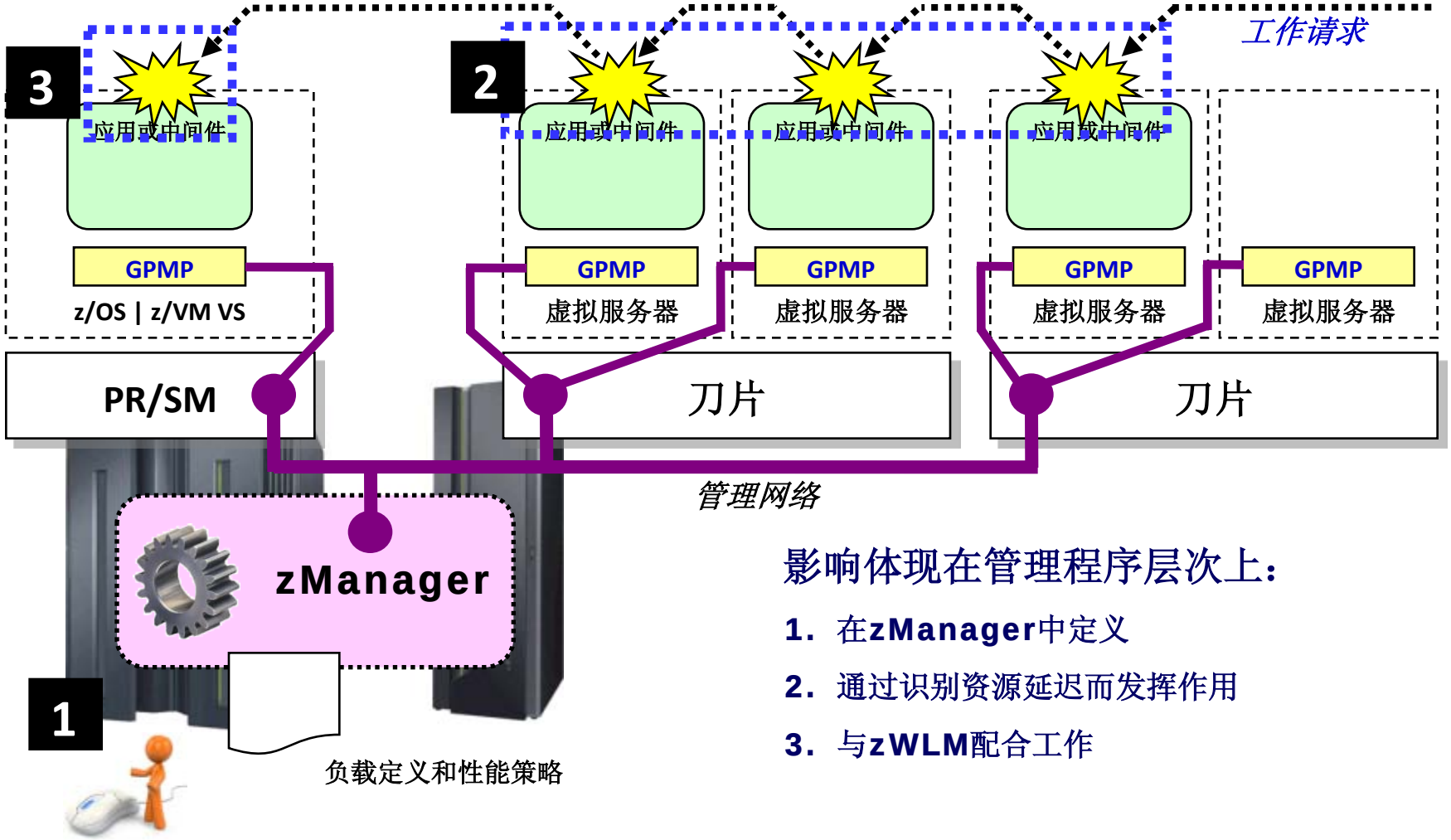


- 对 *管理程序* 的洞察可以看到不同虚拟服务器上的物理资源利用率状态
- 与虚拟服务器操作系统之间的某些接口能够用来观察及管理它们
- 是一种 *发现工作并把它们分组*到一个集合中的途经，这对于人工管理员很有意义
- 是一种量化已经被分组到负载中的工作之间 *相对优先级*的途经
- 是一种 *调整资源*以实现所确定的相对优先级目标的途经

你设想正在扮演 zManager的角色

对工作进行识别、分类和管理

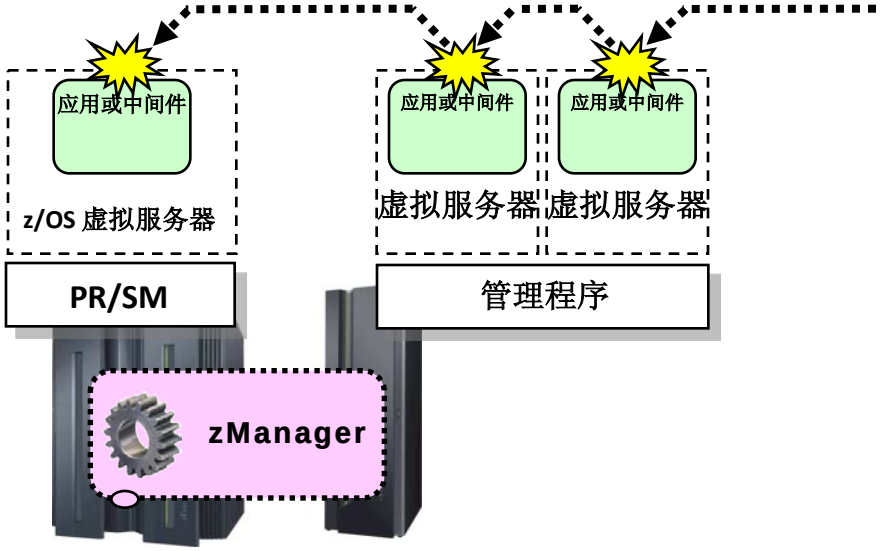
不同的工作经常具有不同的性能优先级。为了区分不同的工作，首先要识别它们 … 然后才可以指定性能目标



影响体现在管理程序层次上：

1. 在zManager中定义
2. 通过识别资源延迟而发挥作用
3. 与zWLM配合工作

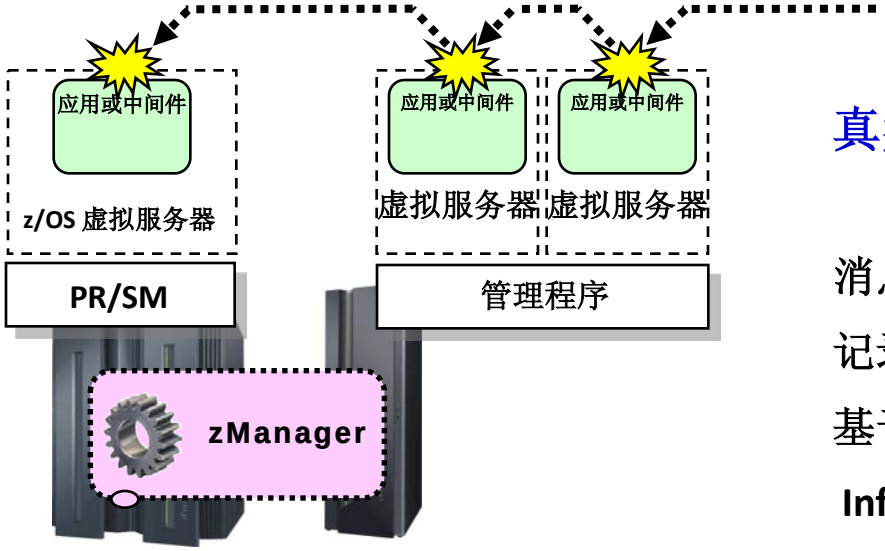
请记住 – 仅仅通过展示就可能实现价值



- 分布式的WebSphere并不'知道' 它在zBX之中
- 它不会做任何事情来利用此环境
- 价值是通过共同的z 基础架构使用及zManager来被动地获得的
  - 性能管理
  - 共同的存储子系统使用
  - 专有网络
  - 受控制的管理程序
  - 电源管理
  - 利用率统计
  - ...

非 WebSphere Java 选择

WebSphere之外的Java机会 – z/OS



真实的Java客户z/OS机会

消息路由和转换服务器

记录的压缩 - CMPSC

基于Tomcat/Jetty 的应用服务器

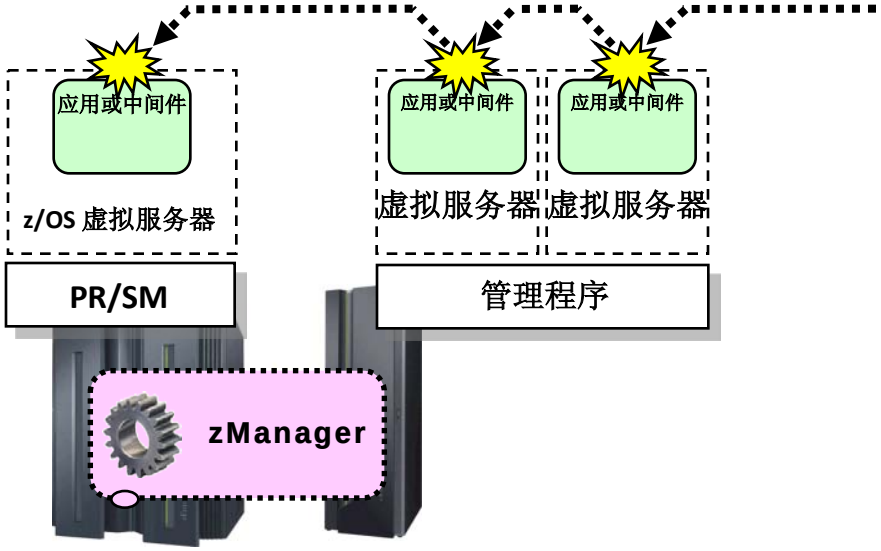
Infocenter

.pdf 文件生成 /转换

COBOL – Java 容器以及DB2 连接共享

通过z/OS R13 和 JZOS 的Java扩展而实现

WebSphere 之外的Java机会 – zBX



真实的客户Java zBX 机会

BigInsights 或 Hadoop

gzip/zlib 记录压缩

Tomcat/Jetty/Glassfish 服务器

.pdf / 图片生成

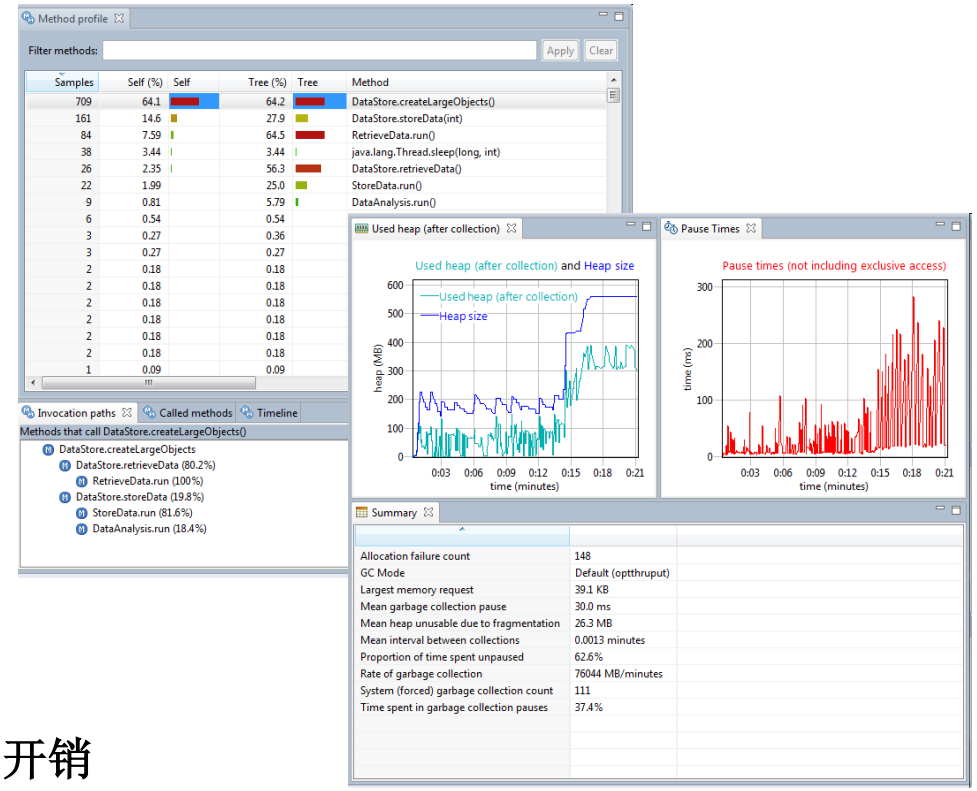
Java 对象缓存

利用 co:Z 工具控制控制远程 Java 流程

常用Java 工具

IBM 面向Java的监控和诊断工具 - Health Center（健康中心）

我在解决什么问题
文档JVM在干什么？一切都正常吗？
为什么我的应用运行缓慢？
为什么无法伸缩？
我在使用正确的选择吗？

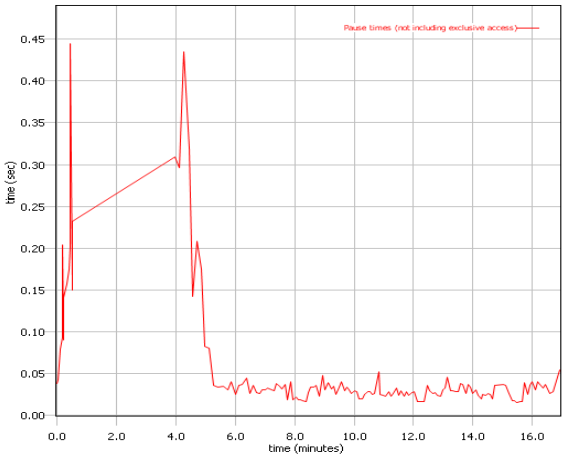


概述
轻量级实时监控工具只需要非常低的开销
了解你的应用的行为方式，诊断潜在问题并提供建议
使垃圾回收、方法建档、类加载、锁分析、文件 I/O 以及本地内存利用实现虚拟化
适合于运行在IBM JVM 上的所有Java 应用 – z/OS, AIX, xLinux, zLinux

IBM 面向Java的监控和诊断工具 - GCMV

我在解决什么问题
垃圾回收器(GC) 的表现如何？能够做得更好吗？
GC 占用了多少时间？
我的 JVM 还有多少剩余内存？

概述
分析Java 词汇(verbose) GC 日志，提供对应用行为的洞察
可视化多种垃圾回收数据以及Java堆在不同时间的统计信息
提供了检测内存泄露的能力并优化了垃圾回收
提供建议时采用启发式方法，指导你采用GC性能调优



Tuning recommendation

The garbage collector seems to be compacting excessively. On average 45% of each pause was spent compacting the heap. Compaction occurred on 40% of collections. Possible causes of excessive compaction include the heap size being too small or the application allocating objects that are larger than any contiguous block of free space on the heap.

The garbage collector is performing system (forced) GCs. 5 out of 145 collections (3.448%) were triggered by System.gc() calls. The use of System.gc() is generally not recommended since they can cause long pauses and do not allow the garbage collection algorithms to optimise themselves. Consider inspecting your code for occurrences of System.gc().

The mean occupancy in the nursery is 7%. This is low, so the gencon policy is probably an optimal policy for this workload.

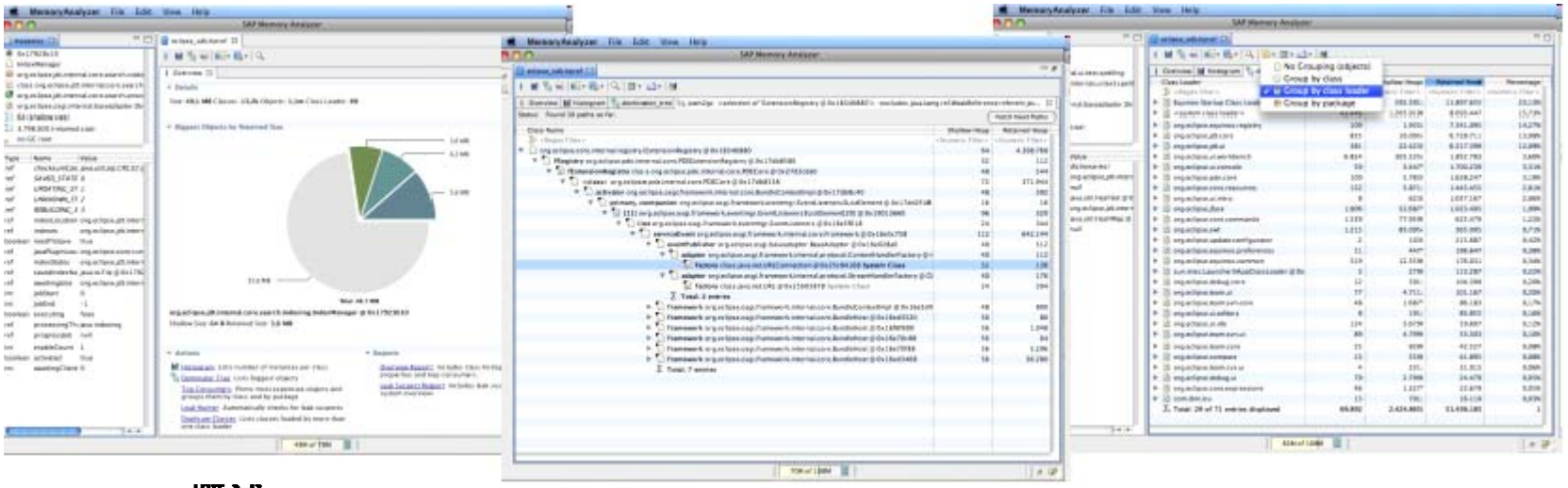
The mean occupancy in the tenured area is 14%. This is low, so you have some room to shrink the heap if required.

Summary

Allocation failure count	140
Concurrent collection count	0
Forced collection count	5
GC Mode	gencon
Global collections - Mean garbage collection pause (ms)	185
Global collections - Mean interval between collections (minutes)	0.13
Global collections - Number of collections	5
Global collections - Total amount tenured (MB)	93.1
Largest memory request (bytes)	127784
Minor collections - Mean garbage collection pause (ms)	48.2
Minor collections - Mean interval between collections (ms)	7193
Minor collections - Number of collections	140
Minor collections - Total amount flipped (MB)	668
Minor collections - Total amount tenured (MB)	38.8
Proportion of time spent in garbage collection pauses (%)	0.76
Proportion of time spent unpaused (%)	99.24
Rate of garbage collection (MB/minutes)	874

IBM 面向Java的监控和诊断工具 - 内存分析器

我在解决什么问题
为什么我耗尽了Java内存？
我的Java堆里有什么东西？我如何浏览它并获得新的洞察？



概述
用于分析堆转储并寻找JVM内存泄漏的工具
与IBM系统转储、堆转储以及Sun HPROF二进制转储协同工作
提供内存泄漏检测、足迹分析以及对浪费的空间的洞察
按类划分的对象、支配树分析、GC 根路径、按类加载器划分的支配树
提供类似 SQL 的对象查询语言 (OQL)

提 问

结束

感谢您的关注