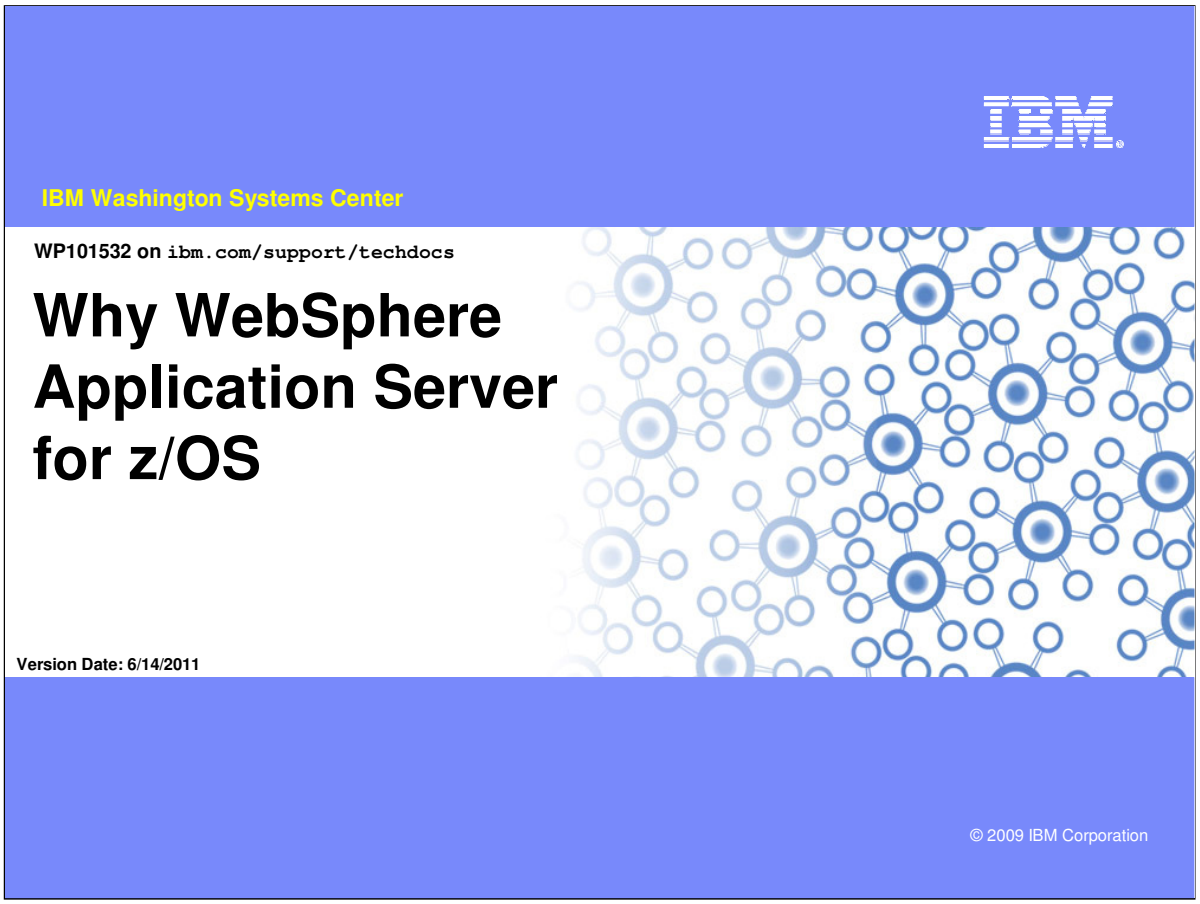

IBM Washington Systems Center
WP101532 on ibm.com/support/techdocs
Why WebSphere Application Server for z/OS
Version Date: 6/14/2011
© 2009 IBM Corporation

Document Change History

July 30, 2009	Original Edition
July 31, 2009	WP101532 document number applied and republished
August 3, 2009	Various updates based on review by Dave Follis
June 14, 2011	Various updates based on WAS z/OS V8

The WebSphere Application Server for z/OS support team at the Washington Systems Center consists of: **John Hutchinson**, **Mike Kearney**, **Louis Wilen**, **Lee-Win Tai**, **Steve Matulevich**, **Mike Loos** and **Don Bagwell**.

Mike Cox, Distinguished Engineer, serves as technical consultant and advisor for all of our activities.

Particular thanks to **Dave Follis** of the WAS z/OS development staff for his review and guidance on this presentation.

For questions or comments regarding this document, send e-mail to Don Bagwell at dbagwell@us.ibm.com

Introduction

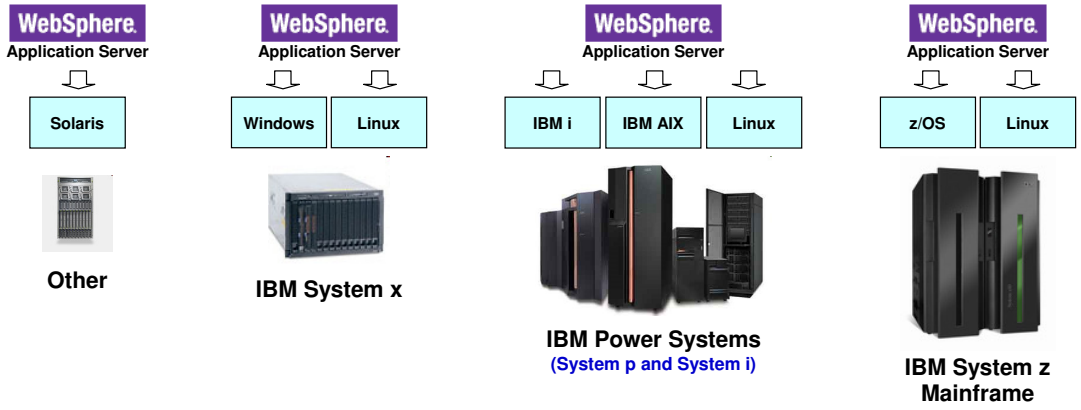
The objective of this presentation is to illustrate the key technical differentiators of WebSphere Application Server running on System z and z/OS platform.

We will also show how those technical attributes contribute to business value.

WAS and the Platform Decision

Assume the decision for WebSphere Application Server has been made.

WAS is a cross-platform product offering



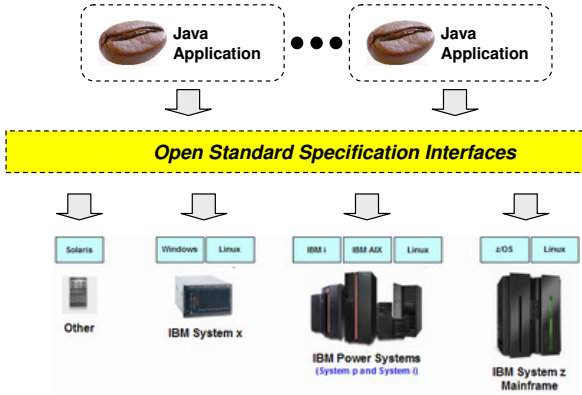
Making the hosting decision …

This chart illustrates one of the key basic facts about WebSphere Application Server -- it is a cross-platform *family*. It runs on several different operating systems on several different hardware platforms.

Assuming that the decision has been made to use WebSphere Application Server as the Java runtime environment, how does one approach the question of where to host it?

Very Important Starting Concept

This point can't be stressed enough -- the differentiation is *not* in the open standard specification support offered. *That is common across platforms!*



"WebSphere is WebSphere" above the specification interface line

How it's *implemented* is dependent on the platform … its features, functions, attributes and qualities of service

Starting with V6.0 the code base merged into one with a single source

Problems with code divergence solved. Code has ability to detect platform and invoke platform-specific exploitation as appropriate.

A question often asked is this: "WebSphere Application Server is based on accepted open standards. 'WebSphere is WebSphere' people say. How can the platform itself make a difference?"

It's true that "WebSphere is WebSphere" -- by that we mean that the open standard specification support for WebSphere Application Server is the same across all the operating systems. So, for example, Version 7.0 on Windows has the exact same specification support as Version 7.0 on z/OS.

Note: such was not always the case. But starting with V6.0 it became true.

The important thing to understand is that the open standards are about *interfaces*, not implementation. Vendors are free to implement the function in any way they choose provided the various interface specifications are adhered to. And so it's at the level *below* the specification line that platform differentiation takes place.

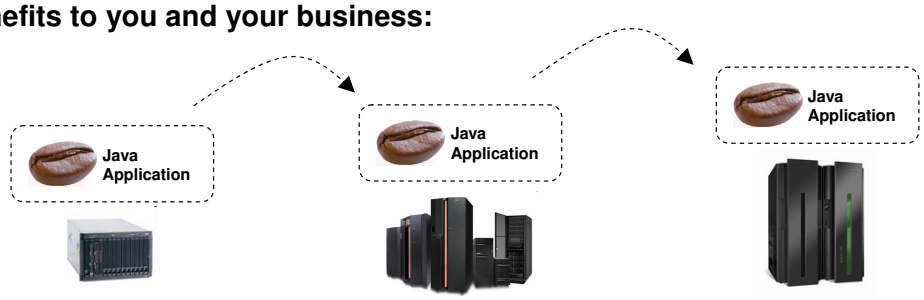
That will be the focus of this presentation. The very short answer to "Why WAS on z/OS" is this: "Because of z/OS."

It's all about the way in which WAS on z/OS is written to *exploit* the features and functions of System z and z/OS platform.

By the way, that exploitation is not exclusive to just z/OS … the other platforms have their own platform-specific exploitation code. But those platforms don't have the same features to exploit that z/OS does.

Key Benefits of Alignment Across Platforms

The alignment of specifications across all the IBM platforms brings several key benefits to you and your business:



Development and test | Mid-tier application hosting | Mainframe exploitation application hosting

- **Avoids costly rewrite of applications to change platforms; reduces the testing effort**
- **Ability to promote applications "up the ladder" without concern about loss of interface function**
- **Ability to architect application designs that span multiple platforms without having to make sacrifices based on the platform**
- **Ability to settle on a common set of application tooling across all platforms**
- **Ability to have an essentially common management interface across all the platforms**
  Minor **differences exist in areas related to platform specifics such as starting servers. More in a bit.**

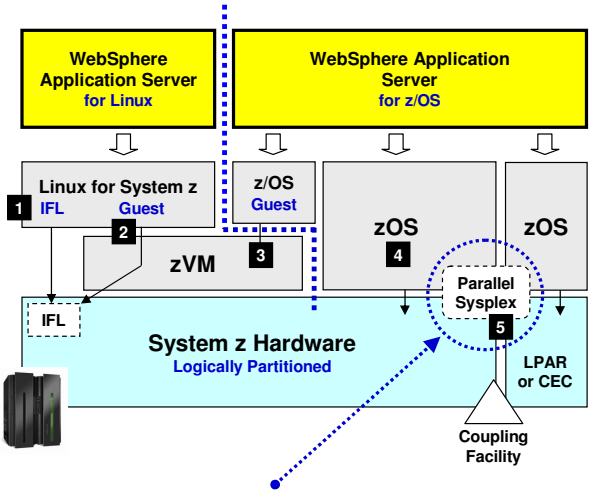
These benefits are intended … this is why "WebSphere is WebSphere" across the platforms

This chart is illustrating some of the reasons behind IBM's strategy of aligning the specification support across all the platforms. This is why "WebSphere is WebSphere."

As we mentioned earlier, this was not always the case. In the very early releases of WebSphere Application Server there was functional disparity. But starting with V6.0 the different platforms "came together" -- not just in specification support but also in terms of how the code base is managed within IBM. It all makes for a more stable and cohesive product family.

System z, z/OS, Linux, and WebSphere Application Server

Here's a mapping of how the two flavors of WebSphere Application Server can be hosted on System z hardware:



1. **Linux for System z directly on IFL**
   Possible, but not very common. Solution where no zVM skills exist
2. **Linux for System z as guest on zVM**
   Very common. This provides excellent virtualization with zVM with Linux running as a guest. Runs on the IFL.
3. **z/OS as guest on zVM**
   Another example of zVM's virtualization capabilities. WAS z/OS as guest typically in a development or test environment.
4. **z/OS in a non-Sysplex environment**
   WAS runs directly on z/OS with no zVM virtualization. No Sysplex more common in test environments or small production.
5. **z/OS in a Parallel Sysplex environment**
   This is the flagship environment. This is where high availability, scalability and maximum platform exploitation takes place.

WebSphere z/OS design and implementation capitalizes on the Sysplex environment

Much more to follow

7 IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD

WP101532 on ibm.com/support/techdocs

Here we're providing a kind of schematic diagram of how WebSphere Application Server is capable of running on the System z hardware. The System z hardware has three operating systems it can run, and this chart is designed to try to clear up the options.

The chart's text provides the information intended. Our focus in this presentation will really be on numbered blocks 4 and 5, as those are the z/OS options where platform exploitation is the most apparent.

Note: Numbered block 3 also has z/OS exploitation going on, but for the purposes of this presentation we'll assume 3 is the same as 4 and 5. Running z/OS as a guest under zVM makes a wonderful test/development environment. Not as common for production environments.

We will briefly talk about Linux for System z as a positioning of that option against z/OS. But in general this presentation is about WAS on z/OS.

Preliminary Conclusion

WebSphere is WebSphere at the specification line and above

Therefore, there is no platform differentiation *at that level*

Differentiation occurs *below* the specification line

That's where the attributes of the platform are exploited by the implementation of WAS on that platform

Three Big Questions:

1. **What are the qualities and attributes of the System z and z/OS platform?**
2. **To what degree does WebSphere Application Server exploit those qualities and attributes?**
3. **How do those qualities and attributes contribute to meeting your key business objectives?**

Addressing those three questions is the objective of this presentation

We've established that "WebSphere is WebSphere" across the platforms at the specification line and above, and that the platform exploitation that takes place is *below* that line. So ultimately the question is: what are the attributes of the platform and how does WAS exploit them? As mentioned, our focus will be on z/OS.

We'll also strive to map those attributes to business value as at the end of the day that's what this is about -- providing value to the core business to meet the objectives of the business.

Outline of Presentation

Going forward from here we'll touch on the following topics:

- **Platform exploitation and its two forms: passive and active**
  - **Both are important; active is where WAS for z/OS shines. We'll have a sub-focus in this section on the IBM SDK for z/OS and see how it too exploits the platform**
- **Take a look at WAS on Linux for System z**
  - **To understand the key differences and position the two System z offerings**
- **What makes Java code "z ready"**
  - **Hint: this is going to be a very short section!**
- **Availability and scalability**
  - **We explore how WAS z/OS uniquely exploits platform attributes to provide both**
- **Quick Look at Other WAS-Based Solutions**
  - **One chart to make the point that these solutions can ride on the benefits derived by WAS z/OS**
- **Administration and skills**
  - **A common question is how current skills map to the WAS z/OS environment**
- **The issue of cost**
  - **It's on everyone's mind … so we try to tackle it head-on**

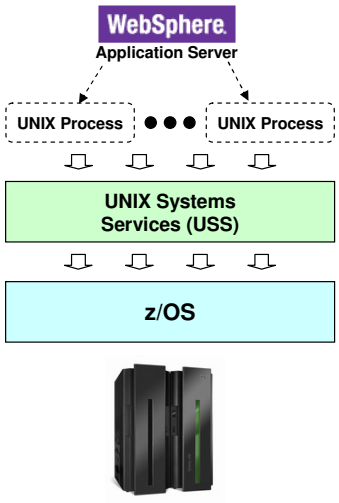
Here is the agenda and flow of this presentation.

The Issue of Platform Exploitation

Establishing key concepts related to platform exploitation

What *Could* Have Been … But Thankfully Was Not

WebSphere Application Server "for z/OS" could have been implemented as a pure UNIX application, with no direct exploitation at all:



Question: would there be any platform exploitation?

Answer: yes, but it would be very *passive*.

Examples: redundant design of the hardware platform; efficient and scalable I/O subsystem; storage protection architecture; virtualization at the LPAR level; etc.

This positions the concept of *active* and *passive* exploitation

Active	direct exploitation of platform qualities and attributes by the code under the specification interfaces
Passive	benefits that derive simply by running on the platform … or as some say, "just showing up"

Let's expand on that a bit …

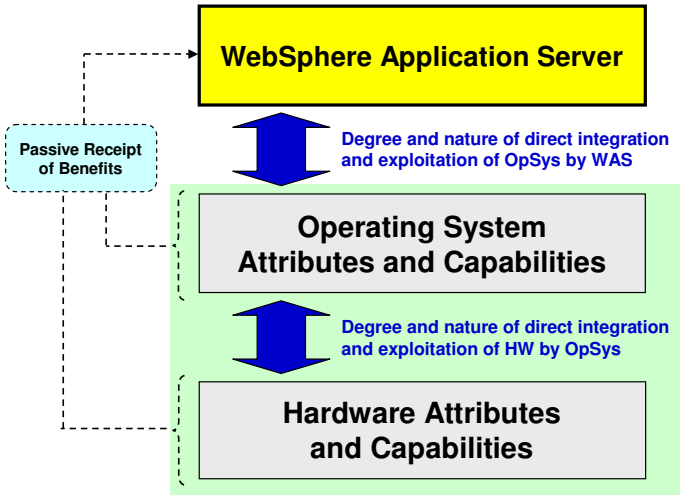
IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD

The origins of WebSphere Application Server come from the distributed platform, Unix in particular. That solution could have been carried to the z/OS platform and run as a series of Unix processes without any real integration with the z/OS operating system beyond running there under Unix Systems Services.

That would have resulted in "platform exploitation," but we hope you can see how that exploitation would have been very *passive*. That helps position the two forms of exploitation we wish to show -- passive and active.

Multiple Levels of Exploitation Taking Place

We need to understand that there are benefits from the hardware design, benefits from the operating system design, *and benefits from the integration between the two*



Not all hardware designs are equal

Not all operating systems are equal

Not all operating systems have the same degree of integration with the hardware

Example: z/OS only runs on System z … there are no tradeoffs to enable multi-platform flexibility. The OS is optimized for the hardware; the two are developed jointly

That doesn't mean System z and z/OS are appropriate for all cases
Nor does it mean other platforms and operating systems can do what System z and z/OS can do

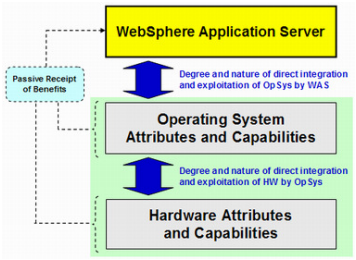
Here we're drawing into focus the relationships that exist between a solution (WAS in this case) and the underlying hardware and operating systems:

- The first is to what degree the solution directly and actively exploits the functions of the operating system. As we stated, had WAS z/OS been delivered as a packaged set of Unix processes that ran under USS, there would have been very little active exploitation of z/OS functions such as WLM.
- The second is the degree of direct and active exploitation of the hardware platform by the operating system. Here we wish to point out that z/OS is *not* designed to be a semi-portable operating system that runs on many different hardware platforms. z/OS is designed to be run on System z hardware. That means the operating system is tightly integrated with the hardware. The hardware and operating systems are designed to work closely together. There's a rich set of low-level exploitation that get taken advantage of by, for example, the JDK for z/OS.
- Finally, there are the benefits that bubble up to the solution passively. That is, just by running there (by "just showing up") a solution receives some benefits. An example of this would be the systems management of z/OS in areas such as backup and recovery, or capacity planning, which ultimately benefits the solution but the solution is not actively reaching out and making use of anything to receive the benefits.

The yellow box at the bottom of the chart is important to note. There is a difference in operating systems and hardware platforms. z/OS may very well be suitable for the business needs; or other solutions may suit the business. The key is to understand the merits of the platform and understand how they might map to key business objectives. If the choice is for WAS z/OS then we're happy. But if a careful analysis yields a different conclusion that is better for your business, then that is good as well.

Three Responses to "Value Statements"

When the features and functions of the System z and z/OS environment are discussed, we often see people respond in three ways:



Understand and Agree

People with a background in System z. Focus then turns to the way WAS z/OS exploits those features.

News to Me … Want Better Understanding

Becomes a matter of going into greater detail on each of the functions. It can be a broad topic area and often requires time and a whiteboard.

We welcome these discussions. Please ask for more details if you're unfamiliar with the feature.

Have Heard Before and Have Different Opinion

Addressing this involves understanding the nature of the differences between our view and yours.

We welcome this discussion as well. Different perspectives are always useful. We'll take the discussion offline and see where we can agree and where differences in opinion still remain.

We need to pause and understand that any time "value statements" about something are offered -- doesn't matter if it's a computer solution like WAS or a new washing machine -- there are three essential responses. The chart explains those three responses.

We are not trying to suggest which of these anyone *should* have. We are trying to get to a point where everyone is in agreement on what they agree on and what they might disagree on.

Different opinions on some of the attributes of a platform are inevitable. Differences of opinion on the degree of benefits one gets from something are even more likely.

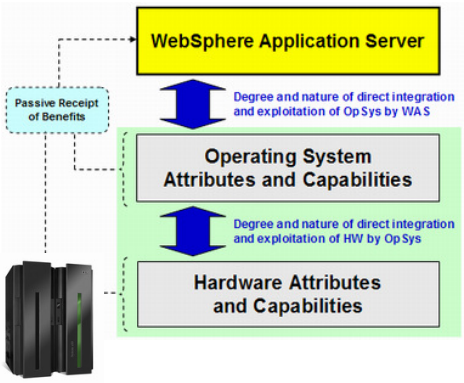
All perfectly okay … the point here is that we should be clear on all this. A good spirited discussion -- civil in tone, of course -- is a good thing.

Passive Exploitation Benefits

Benefits derived by WebSphere Application Server by virtue of running on the platform

Passive Benefits Fall Into Several Categories

Programs that run on System z and z/OS receive passive benefits in a couple of different areas:



Hardware

- **Inherent maturity and stability of design**
- **Redundancy and flexible updates**
- **Balanced design offers very high throughput**
- **Mature and proven virtualization through LPAR**

Operating System

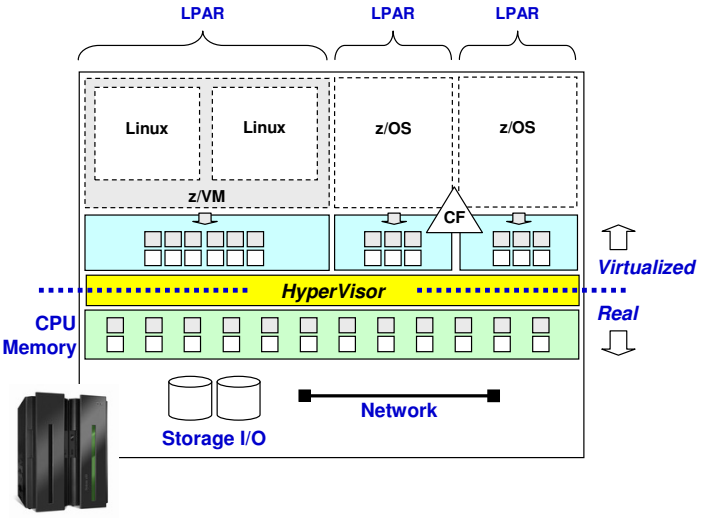
- **Tight integration with server hardware design**
- **Extremely mature architecture**
- **Storage protection**
- **Workload Manager (WLM)**
- **Intelligent Resource Director (IRD)**
- **Local TCP optimization**
- **Mature systems management tools**
- **Proven disaster recovery capabilities**

Let's explore some of these at a high level

Passive benefits are those that are associated with the platform and operating system, but are ones that are not *directly* taken advantage of by the solution. For WAS those can be categorized in two ways: the hardware benefits and the operating system benefits. The chart offers some bullets that give you a sense of the case we make on behalf of System z and z/OS.

Hardware Virtualization -- Logical Partitions (LPAR)

An extremely mature virtualization technology that allows real resources to be shared across multiple logical partitions, each entirely separate



Proven virtualization technology

- Years of proven reliability
- Partitioning of HW into logical partitions
- Further virtualization using z/VM with guest machines

Each LPAR entirely separate from the other

- Hypervisor protects one LPAR from monopolizing resources above what its allocated
- Complete memory isolation, so no overlay concerns
- Complete operating system isolation, so all elements of OS instances separated
- Complete network isolation, so no concerns about security breaches

Benefits of consolidation with the advantages of isolation

That's what virtualization is all about. The difference is one of maturity and capability. The technical differences between virtualization approaches can become a complex topic quickly. Point here is that System z LPAR has a proven production track record

We start out with a high-level description of the virtualization capabilities of the platform. The topic of virtualization can get very detailed very quickly, and there are several forms of virtualization technologies available in the world.

The key point here is that logical partitioning on System z is done very low -- just above the hardware layer -- and has been around for a long time. This is not "new" technology. That means it's *stable* technology.

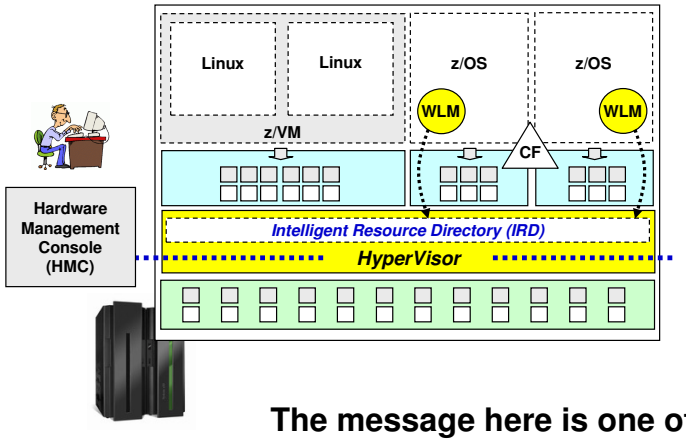
The LPAR technology on System z has matured to the point where the hardware "sharing" that takes place is carefully regulated so no partition can monopolize more than its granted. There is *complete* isolation between the LPARs -- operating system, network, and memory.

One of the key things that System z virtualization provides is a virtualization of the I/O subsystem capacity. One LPAR can't consume more of that than allowed, which means other LPARs that have higher I/O needs can get what they need. Again, no monopolization.

We readily grant that there are other virtualization technologies out there. There are even some that do a kind of hardware layer virtualization. But none are as mature as System z virtualization.

Dynamic Modification of LPAR CPU Resource Allocations

Manual and automatic …



- Non-disruptively add CPU to the machine and assign to LPAR
- Allow IRD to dynamically move CPU between LPARs
- Dynamically vary I/O capacity across LPARs to solve bottlenecks
- With z/OS you may have WLM advise IRD to reallocate CPU and I/O between LPARs in the Sysplex

The message here is one of dynamic flexibility.

The pace of change is increasing … rigid designs hinder rapid exploitation of opportunities

System z LPAR technology coupled with z/OS WLM provides a proven flexible and dynamic environment

17 IBM Americas Advanced Technical Support, Washington Systems Center, Gaithersburg, MD WP101532 on ibm.com/support/techdocs

When we move to the way the operating systems make use of the hardware virtualization, we see an even more complete picture.

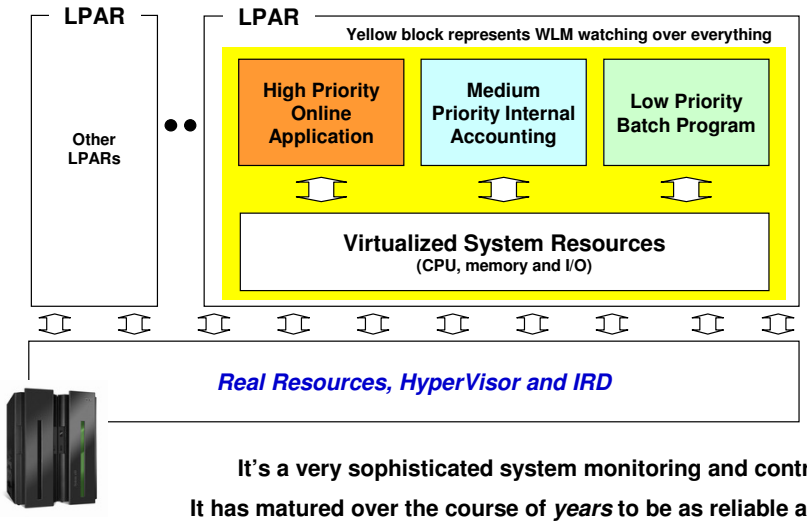
First, the virtualization capability of System z allows for a *dynamic* re-allocation of CPU between LPARs, and a *dynamic* re-allocation of I/O capacity to overcome unanticipated bottlenecks.

Above that, we have IRD -- Intelligent Resource Director -- which is a function that intelligently shifts resources based on priorities established by you. And higher still, WLM (Workload Manager, a function of the z/OS operating system) monitors activity within the LPARs against established goals and can work with IRD to move system resources in or out based on those goals.

Exactly how all this works and how its configured and defined is a topic for a separate discussion. The point here is that System z has evolved these capabilities over time. Early releases had their issues, of course, but over time those were worked out. Today we have a very mature, very proven virtualization platform. Not the *only* one in the market, we agree; but the most mature.

Workload Manager (WLM)

A component of the z/OS operating system, WLM keeps close watch on key system metrics and manages resources towards meeting your defined goals



Five pieces to this:

1. WLM's real time monitoring of the overall system resource utilization
2. The WLM service level goals you've defined that determine how WLM will manage resources
3. WLM's comparing your service level goals against the actual system performance on a program by program basis
4. WLM's reallocating resources within the LPAR to make sure goals are met
5. WLM advising IRD if resource allocation across LPARs is needed

It's a very sophisticated system monitoring and control mechanism

It has matured over the course of *years* to be as reliable and effective as it is

Other solutions claim to offer "workload management" but are often rather weak in function compared to how z/OS WLM operates

Workload Manager (WLM) is a function of the z/OS operating system and is a key part of the management of resources within z/OS, and is a key component that's actively exploited by WAS.

Here we're looking at passive exploitation. We'll explore the active exploitation in a bit.

WLM monitors a wide range of system metrics and compares those values -- using all sorts of complex heuristics -- against performance goals you've established for the system. The result is that WLM manages the resources of the system to achieve your goals. Lower priority work items can not "steal the system" -- WLM will not allow it. If higher priority work isn't meeting its goal, WLM will reduce resources to lower priority work and give it to the higher priority work so the goals can be met.

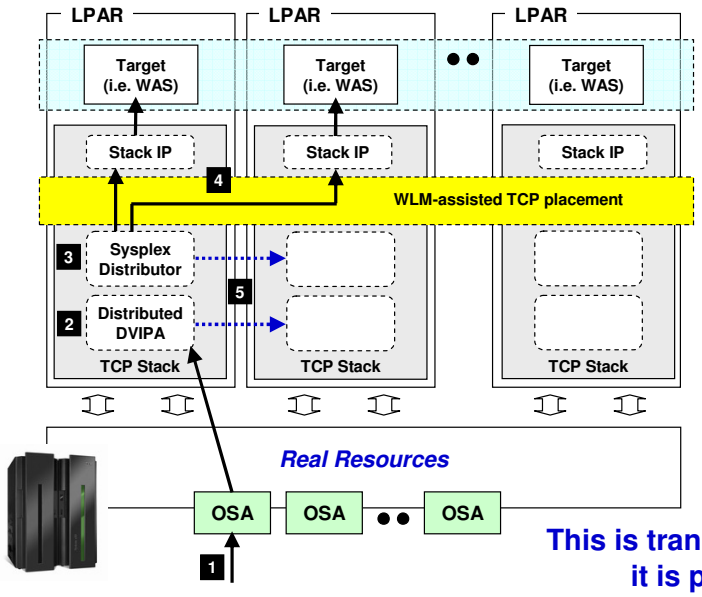
This is how z/OS manages a system to such high utilization numbers. It's by intelligently allocating system resources to work based on your goals.

And don't forget that WLM can advise IRD to move resources from LPAR to LPAR.

The picture starts to come through -- intelligent monitoring of work and goals and the re-allocation of resources … all dynamically done.

Virtual IP and Sysplex Distributor

Is a function of TCP on z/OS which allows you to "hide" duplicated resources behind a single IP address with WLM-assisted TCP connection placement



What's going on in this picture:

1. Clients in the world point themselves at a "generic" IP host name. Routers resolve that to one of the OSA adapters on the machine.
   - Note: there are ways to have redundant OSA adapters for availability
   - Note: it's not shown on this picture, but WLM can also advise some off-board Cisco routers.
2. Request is mapped to the TCP stack in the Sysplex that's hosting the Distributed Virtual IP (DVIPA) generic host.
3. Sysplex Distributor function determines which of the potential target LPARs is the best candidate to receive new work at that point in time.
4. TCP connection is made between client and the target
5. In the event of an outage of the hosting LPAR or TCP stack, the DVIPA and Sysplex Distributor functions automatically move to a defined "next in line" stack.

This is transparent to the application … it is passive in this process

19 IBM Americas Advanced Technical Support Washington Systems Center, Gaithersburg, MD WP101532 on ibm.com/support/techdocs

There another element of virtualization and dynamic operations and it comes in at the networking level. Down at the hardware level there are physical adapters, and they can service one or many LPARs on the system.

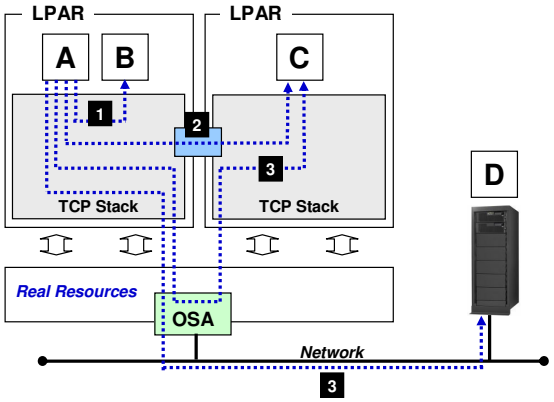
TCP on the z/OS system has the facility to virtualize IP addresses. Some may be *dynamic* virtual IPs that have the ability to move to another LPAR in the Sysplex in the event the initial hosting TCP stack is lost. That's a way to provide availability of a listening IP by virtualizing that IP behind the physical adapters and making it dynamically movable across LPARs if configured that way.

Between the DVIPA and the target WAS z/OS (or any TCP socket application for that matter) there can be another layer called "Sysplex Distributor." That is a TCP connection placement mechanism that gets placement advice from WLM. WLM determines what target is best able to take the connection at that point in time.

Numbered block #5 shows the case where there's an outage of some kind on the first LPAR and the DVIPA is automatically moved to a configured backup LPAR. The Sysplex Distributor function is also moved non-disruptively, all of this transparent to the outside world, and transparent to the applications.

Local TCP Optimization

z/OS is smart … it knows when client and target are on the same TCP and it optimizes the request with minimum code path employed



1. **Same LPAR**
   Request resolved within the TCP stack. Never gets to wire. Doesn't even get to the OSA adapter. (Also known as "Fast Local Sockets")
2. **Different LPAR, HiperSockets**
   Request flows memory-to-memory via HiperSocket network, which is a virtual network implemented by Hipervisor.
3. **Different LPAR, *not* HiperSockets**
   Request flows to OSA, but does not touch the wire. No short loop cables. Request stays in OSA microcode and then up to other LPAR.
4. **Off System z**
   Here System z has no choice but to go to the wire.

This can make a measureable difference

Network latency adds up as workloads scale

Some think "TCP is TCP," but in reality there are degrees of optimization on the z/OS platform depending on where the client and target are in the TCP network. The picture above shows four scenarios, from "same TCP stack" to "out on the wire and to another server box."

For numbered block #1 z/OS automatically detects this and the processing gets resolved within the TCP stack with very little code path. It's very fast and very efficient.

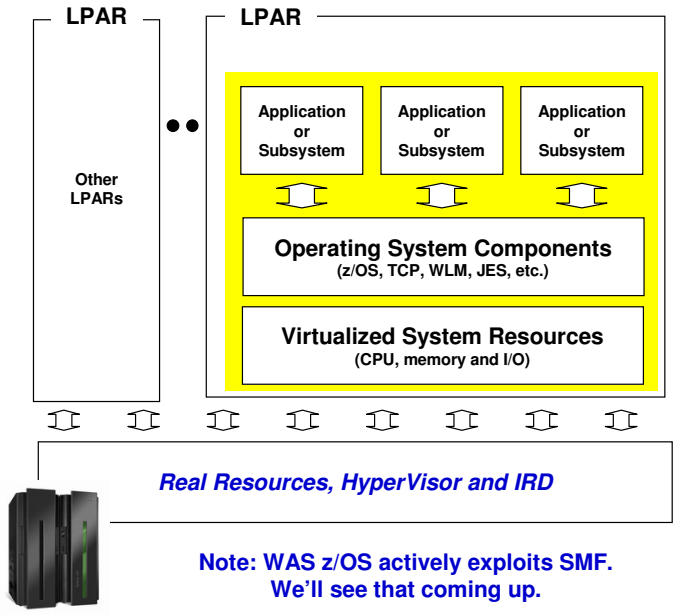
If going to another LPAR there is a function on System z called "HiperSockets." Recall that the memory in an LPAR environment is virtualized; there's the physical memory on the machine and then the virtualized memory in each LPAR. HiperSockets is a cross-memory transfer mechanism between LPARs. To TCP and the applications it simply looks like a network; but it gets mapped to the HiperSockets function near the hardware and mapped across memory.

If, due to the way things are configured, the TCP flow does not go across the HiperSockets network, then it goes down to the OSA adapter as if it was going out on the wire. But if going to another LPAR in the CEC ("Central Electronic Complex" … the physical server box), OSA will prevent the flow from going out on the wire. The communication remains up in the OSA microcode and then back up into the other LPAR.

Network latency is can be a significant issue, particularly in high volume applications. What's shown on this chart is an example of how System z and z/OS tries to squeeze as much network latency out of the equation as possible.

Resource and System Monitoring -- RMF and SMF

In order to manage your server environment effectively and efficiently, you'll need to understand who's using what and when



SMF

A facility that components may use to write records to a system database.

Those records may then be used to analyze system usage for:

- **Capacity planning**
- **Performance planning**
- **Accounting and chargeback**

RMF

Another facility that writes SMF to report on key system activities.

Invaluable for planning and investigation of issues

Note: WAS z/OS actively exploits SMF. We'll see that coming up.

If the platform is more manageable, then users of the platform derive indirect benefit from that

Two other functions of z/OS are SMF and RMF. Both are system monitoring and reporting functions. Both help administrators have a very clear view of what's going on in the system. WAS z/OS actively exploits SMF as we'll see later. WAS z/OS receives passive benefits from RMF because that facility helps you manage the platform better than can be done with other platforms.

Summary of the Business Value Benefits of Passive Exploitation

Broad Guiding Principles of Business Value:

- **Available and Reliable**
  The System z hardware and the key operating systems (z/OS and z/VM with Linux) are mature and proven.

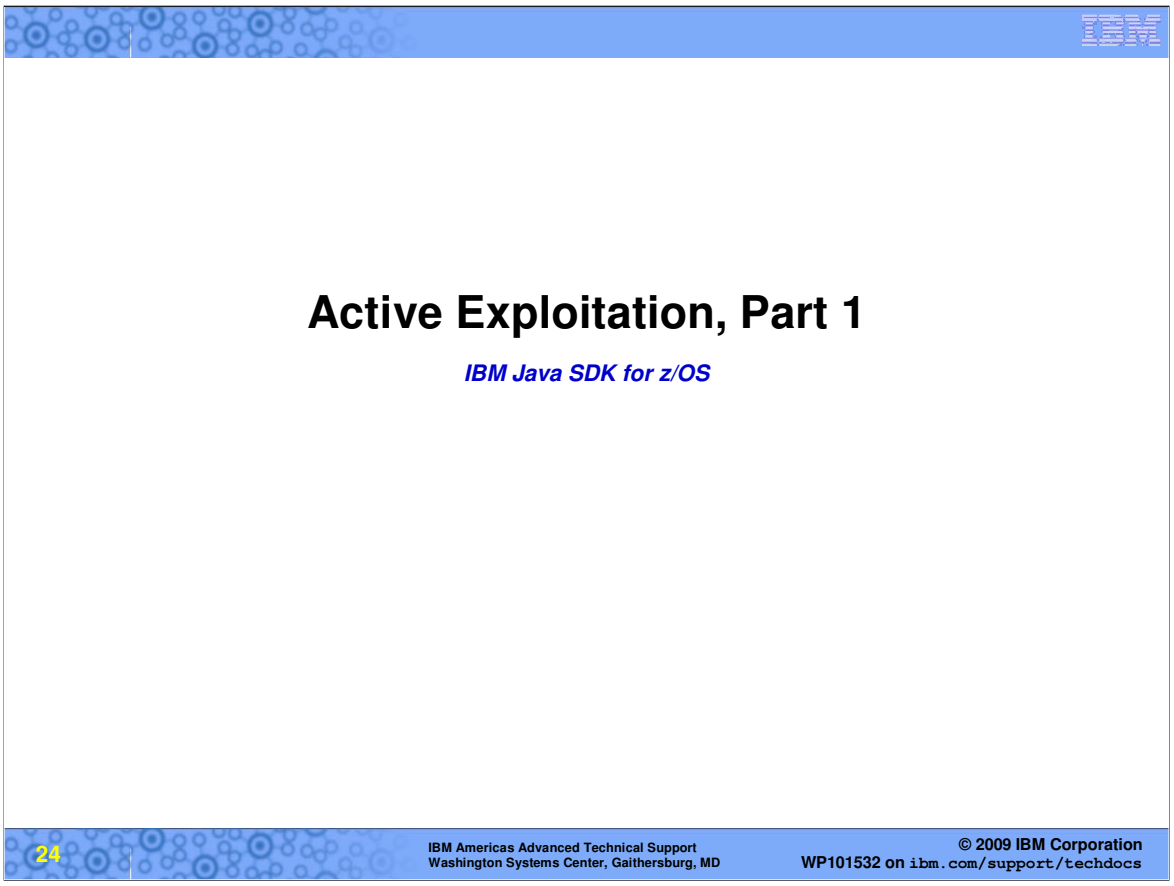
  More reliability and availability is achieved when we get to active exploitation of the platform by WAS z/OS.
- **Manageable**
  The platform has a rich set of systems management tools that help maintain the platform and keep it running.
- **Flexible**
  For z/OS the shared-resource design is mature and allows for co-location with key isolation attributes. z/VM and Linux provides extensive virtualization capabilities.
- **Affordable**
  The value proposition of System z and z/OS is centered around efficient sharing of resources. We cover the cost discussion at the end of the presentation.

This chart sets down four very broad business value statements and maps some of the System z and z/OS attributes to the business values.

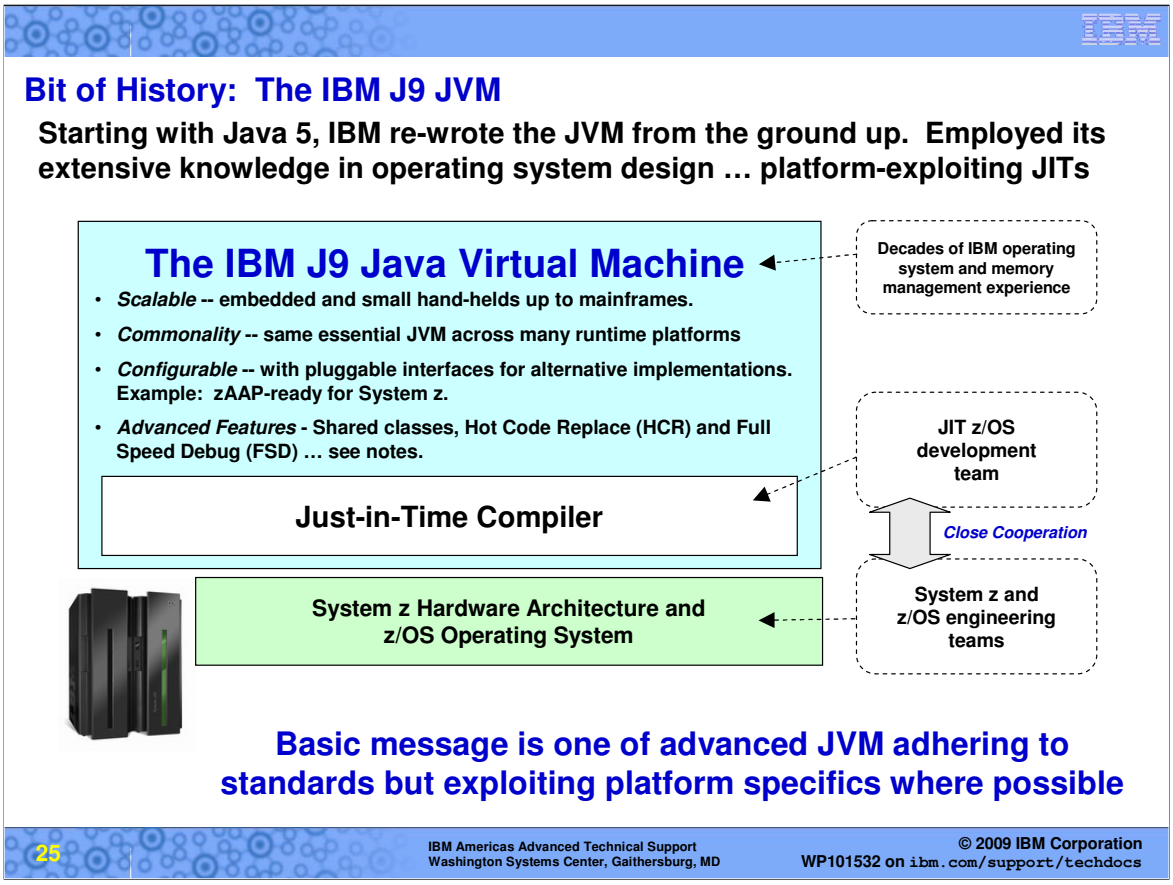
This chart is intended to provide a framework for discussion. Recall our earlier chart about different interpretations of value statements. There may be differences of opinion on some of these, but the key is to engage the conversation meaningfully to see what's best for the business.

Active Exploitation Benefits

In two parts: Java SDK for z/OS and WAS for z/OS



We'll start the active exploitation discussion with a focus on Java for z/OS. Then we'll move to a discussion of the active exploitation WAS z/OS itself does.



A little while back IBM set out to completely rewrite its JVM from the ground up. It was done in what's called a "clean room" so that no Sun intellectual property is part of the IBM JVM. This re-write also allowed IBM to use its very extensive knowledge of Java and general operating system design techniques in the creation of this new JVM.

The JVM -- called "J9" -- is designed to be scalable across a wide range of products, from small hand-held and embedded devices all the way up to the mainframe. All would run the same core JVM with extensible plugins to provide features needed for the platform.

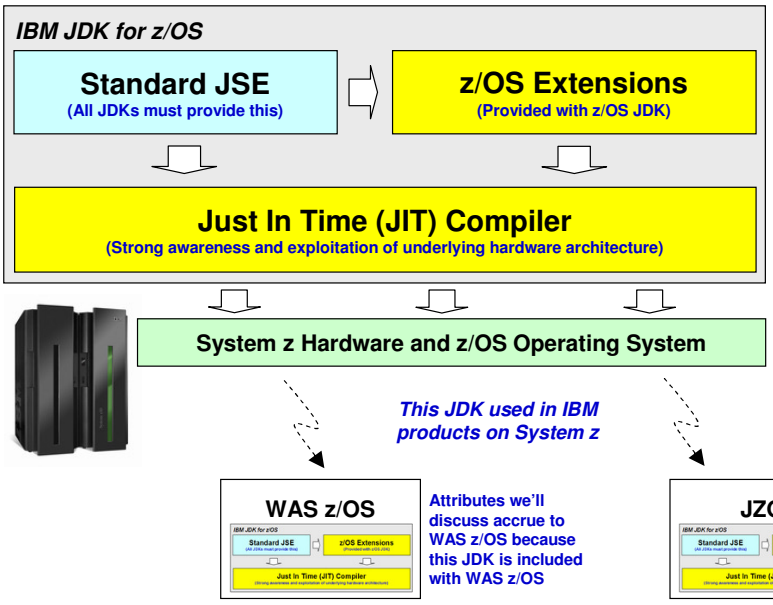
Some features:

- *Shared Classes* -- ability for multiple JVM environments to use the same classes held in a shared memory pool. This reduces duplicated memory consumption and makes for faster JVM startup as classes may simply be fetched from the shared class pool.
- *Hot Code Replace* -- ability to swap in a new class file and have the JVM swap it with the existing class file.
- *Full Speed Debug* -- ability to run the JIT in debug mode at full speed. Use in conjunction with Hot Code Replace to do on-the-fly testing and debugging of issues.

Within the JVM is the Just In Time compiler (the JIT) that turns Java bytecode into platform readable code. Here we need to make sure a key point is understood -- the JVM on each platform has a unique platform JIT engine. The JIT for z/OS is well aware of the z/OS operating system. In fact, it makes extensive use of it. The developers of the JIT for z/OS worked extensively with the hardware and software engineers for System z and z/OS to make sure the integration was tight and fully exploitive.

High-Level Overview of Things We'll Cover

"Java is Java" but not all JDKs are the same, and not all JDKs are fully aware of and exploit the underlying platform



Functionality *beyond* the standard JSE
Particularly in the area of security, but a few other things as well as we'll see

Taking advantage of System z hardware
Which manifests itself in efficiency and performance

Attributes we'll discuss accrue to WAS z/OS because this JDK is included with WAS z/OS

JZOS is a batch Java launching tool.

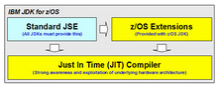
JZOS extends the z/OS-specific functionality even further

This picture is illustrating another key point about IBM's Java … while it is fully compliant with the standards defined for a Java Standard Edition (Java SE, or JSE), it goes beyond that. Going beyond the standard is perfectly acceptable. IBM extends Java to provide access to key z/OS functions, and the JIT, as we just explained, makes extensive use of the lower level hardware.

This J9 JVM then gets used in IBM products. WAS z/OS is our focus for this presentation. But we'll also make brief mention of JZOS because it provides *even more* Java-extensions for the mainframe.

Examples of System z Exploitation by JIT

System z hardware has evolved over the years to have some very sophisticated underlying features. JIT in JDK for z/OS is written to exploit them:



Superscalar Dual-Pipeline

The z890, z990, z9 and z10 are superscalar dual pipeline designs. It permits the dispatching of three instructions per cycle. The JIT understand this and generates code that exploits this where possible.

Register Allocation

13 of the 16 64-bit processor registers are available and the JIT makes extensive and efficient use of the registers to enhance throughput and performance.

Compare-and-Swap

Compare-and-Swap is a feature that allows programs to use comparison of register values to provide a form of code access lock management. It's very efficient. The JIT uses this for Java synchronization.

Private Linkage

Program linkage at the lowest possible level, allowing one method to call another with a minimum of overhead. Includes direct JNI dispatch from JIT.

CISC Exploitation

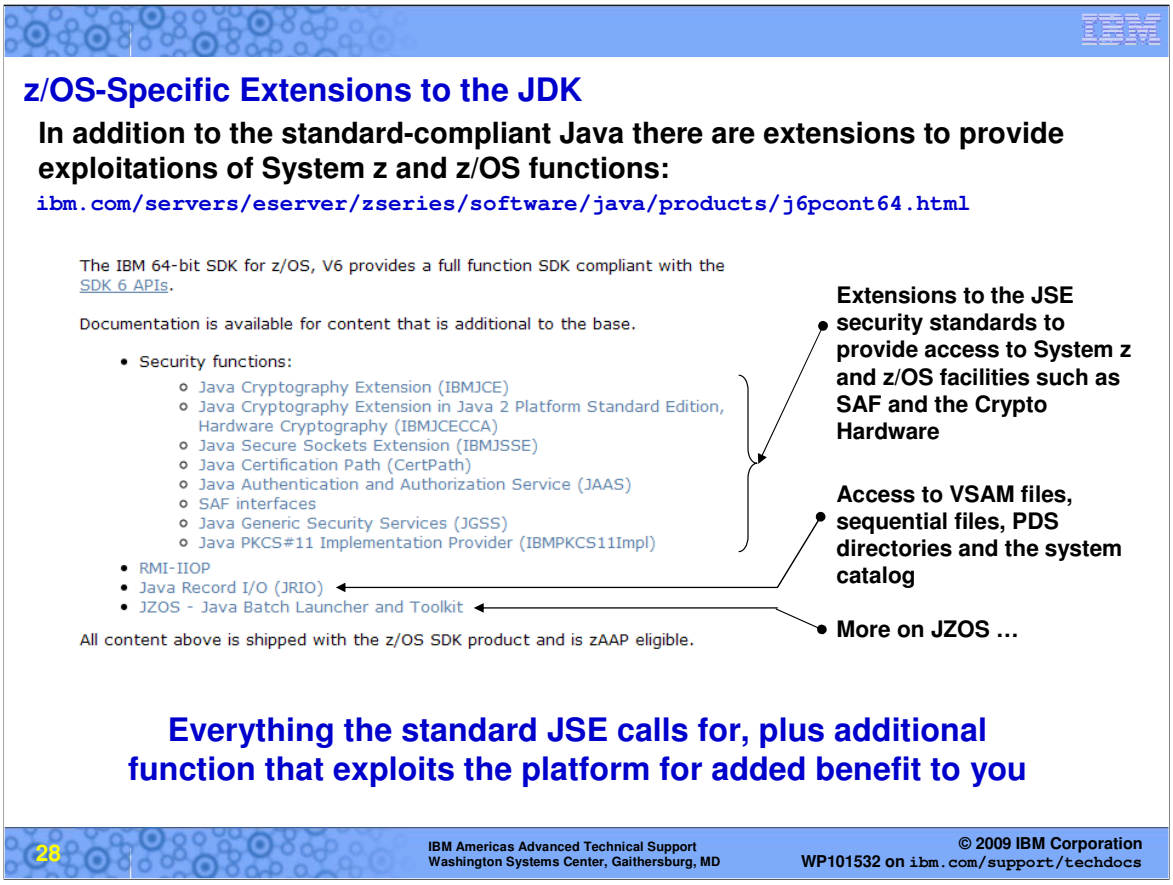
System z hardware is a "Complex Instruction Set Computer" (CISC) design, with many very elaborate instruction features. The JIT knows about these and exploits them; for example, Java loop structures reduced to a very small set of CISC instructions.

64-Bit Instructions and 31-Bit JVMs

The System z hardware is a 64-bit design. If the JVM is running in 31-bit mode the JIT is capable of exploiting the 64-bit hardware for long arithmetic operations.

The JIT is very much aware of the System z hardware features and exploits them directly for greater throughput and efficiency

Let's talk about direct exploitation of the underlying hardware by the System z aware JIT. As mentioned, the developers of the J9 JVM for z/OS JIT worked with the System z hardware engineers to make sure the JIT compiled Java into code that took advantage of the hardware. The chart shows several examples of this.



The URL on the chart above points you to the Java 6 64-bit page … change the 64 to 31 and you'll get the 31-bit page. Same essential information in either case.

There is a set of links on that page that offer information on extensions to the J9 for z/OS JVM to make use of System z specifics, many of which center around the topic of security … the exploitation of SAF and the Crypto facilities of the System z platform.

This is *in addition* to the standard JSE. Applications do not *need* to be written to this, but they *may* if the functionality is desired.

JZOS and Even More Access to z/OS Functionality

JZOS is acquired technology that allows Java to be run from batch or a started task with a more "z-Like" set of features and behavior

```
//JZOSBAT JOB (999,XXX),'JAVA JZOS',CLASS=A,MSGLEVEL=(1,1)
// MSGCLASS=X,REGION=0M,NOTIFY=&SYSUID
//JAVAJVM EXEC PGM=JZOSVM14,
// PARM='HelloWorld'
//STEPLIB DD DSN=JZOS.LIBRARY,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//STDENV DD *
. /etc/profile
 :
//
```

Benefits over BPXBATCH

Flexible configuration of environment variables	Yes
Route output directory to SYSOUT datasets	Yes
Control output encoding separately from default JVM encoding	Yes
Condition-code passing between Java and non-Java steps	Yes
Use MVS datasets and DD statements	Yes
JVM runs in same address space	Yes
Communication with MVS console	Yes

Provides an environment where all sorts of System z and z/OS unique Java class libraries can be built and provided:

Packages	
com.ibm.jzos	Provides the primary classes for the JZOS toolkit.
com.ibm.jzos.fields	Provides the field converter classes for mapping byte array fields into Java data types.
com.ibm.jzos.sample	Provides sample classes for using the JZOS Toolkit API.
com.ibm.jzos.sample.dfsort	Provides sample classes for using the JZOS toolkit DfSort class.
com.ibm.jzos.sample.fields	Provides sample record mapping classes for using the JZOS field converter classes.

JVM launcher tailored to z/OS environment

Launch a batch job … issue a WTO … DFSort … other stuff

29 IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD

WP101532 on ibm.com/support/techdocs

JZOS is technology acquired by IBM a few years back from Dovetail Technologies. The good folks at Dovetail looked at launching JVMs from USS using BPXBATCH and saw several things that weren't quite "z/OS-like." So they developed code that brought up a JVM inside the started task address space and provided all manner of usable extensions to make access to z/OS facilities easier.

In a nutshell -- JZOS is a JVM launcher with extra functionality wrappered around it.

The chart on the upper right shows some of the benefits over BPXBATCH.

In addition, JZOS represents an environment where additional functionality can be fairly easily included. So JZOS comes with a set of class libraries to do things like access DFSORT, use JES facilities, map to DD cards in the JZOS JCL.

In short, JZOS launches the IBM JDK for z/OS (with all its exploitation of the platform) and then goes even further with additional extensions and features.

64 Bit Performance

64-bit JVMs provide relief from heap crowding, but initially came with a performance cost. Two new features match the 31-bit performance profile:

ibm.com/partnerworld/wps/whitepaper/systemz/java_websphere/performance

PartnerWorld > Products > Systems, servers, and storage > Technical >

Match 31-bit WebSphere Application Server performance with new features in 64-bit Java on System z

by Kishor Patil, Marcel Mitran, Jim Cunningham

Last updated: 2009-05-20

Download the white paper (285 KB)

Compressed References

Function added to the z/OS Real Storage Manager (RSM) with APAR OA26294 provides a direct assembler interface which allow memory allocations in the 2GB (2^{31}) to 32GB (2^{35}) virtual address range. The JVM uses this API to allocate the heap in this virtual address range.

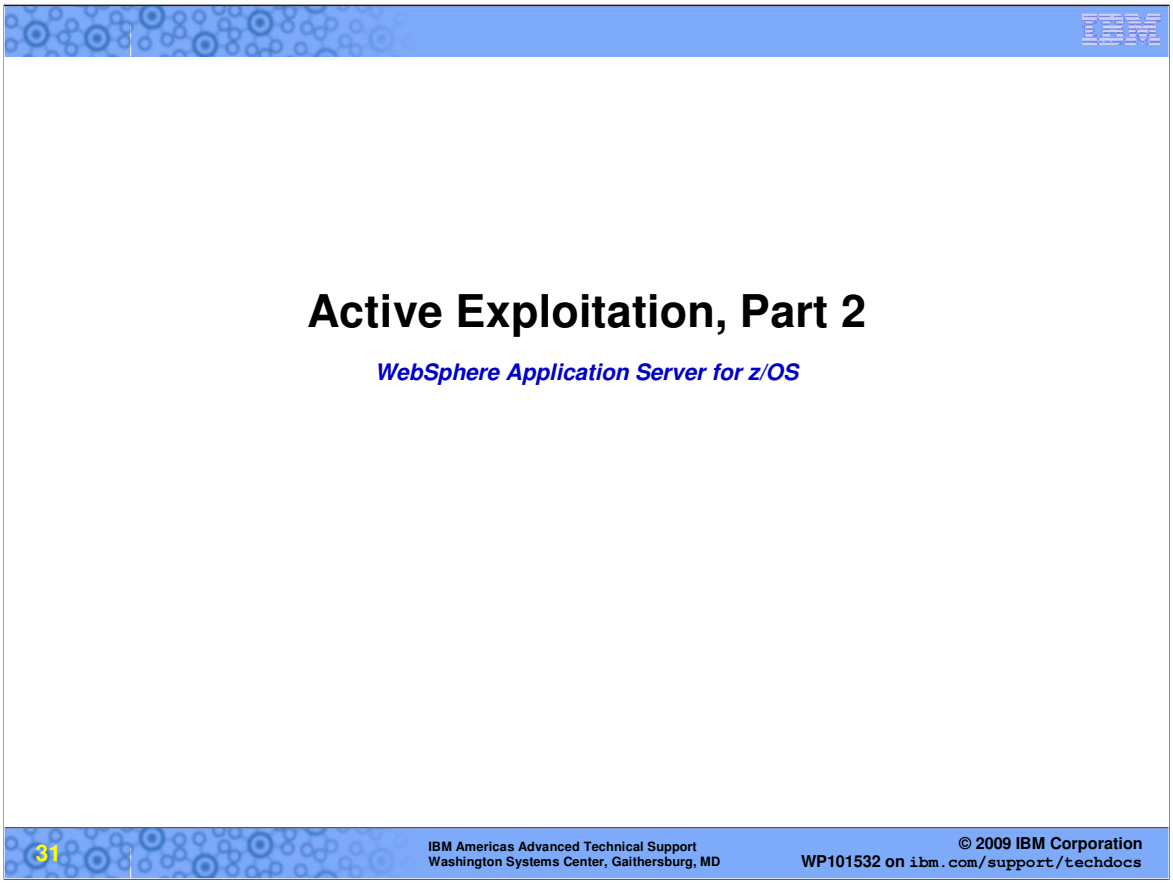
Large Page Support

System z10 processors introduced support for 1MB pages (APAR OA20902 and OA25485 for z/OS 1.9). The 64-bit JVM can achieve performance gains by using large pages, which results in fewer Translation Look-aside Buffer (TLB) entries needed. Fewer TLBs needed for the data footprint of the JVM means more TLBs are available for the executable code. Fewer TLB misses in instruction fetches occurs, which enhances performance.

System z/10 with these two features allows a 64-bit JVM to operate with a larger heap and match the performance seen with the smaller heap 31-bit JVMs.

30 IBM Americas Advanced Technical Support Washington Systems Center, Gaithersburg, MD WP101532 on ibm.com/support/techdocs

64-bit addressable Java came out as a response to the heap crowding that was taking place in 31-bit JVMs. 64-bit addressable Java can make use of enormous quantities of heap storage. One of the early findings was that the extra heap came at a performance cost. But two new features have provided a way to improve the performance of 64-bit Java so it essentially matches the performance profile of the 31-bit. This provides the large heap of 64-bit without the additional performance overhead.



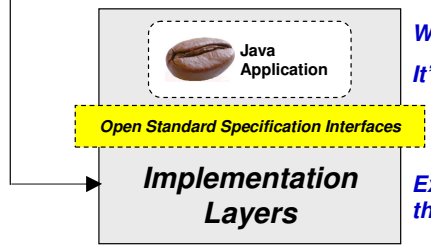
Now we turn our attention to the WAS z/OS itself and how it actively exploits the platform.

Eight Things We'll Explore in This Section

1. Exploitation of JES and common z/OS facilities
2. Exploitation of zAAP specialty engines
3. Exploitation of WLM
4. Exploitation of RRS
5. Exploitation of SAF and Crypto
6. Exploitation of SMF
7. Exploitation of z/OS exclusive Cross Memory Communications

We'll focus heavily on WLM exploitation because that's at the heart of the "Why WAS z/OS" question

These are all z/OS value attributes



WebSphere is WebSphere up here

It's all based on accepted standards

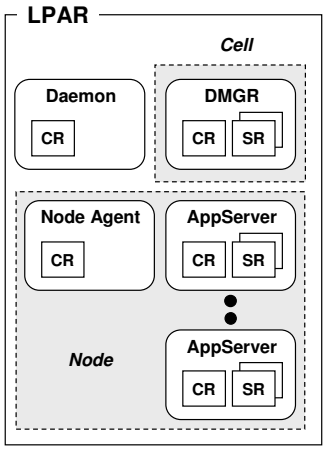
Exploitation taking place below the open standard interface line

We'll focus on the eight areas of exploitation outlined on the chart. We'll pay particular attention to WLM as that is at the very heart of the WAS z/OS exploitation of the z/OS operating system.

Remember -- WAS is WAS at the specification layer and above; what we are going to talk to happens *below* that line.

Exploitation of JES and Common z/OS Facilities

And that means that existing z/OS system programmers will be comfortable with the essential operations of WAS z/OS … it maps to their present skills



- WAS z/OS runtime implemented as a series of started tasks
- Standard JCL and START commands employed
- JCL START procedures maintained in PROCLIB
- Output written by default to JES
- JES manages output and storage
- Started tasks and address spaces displayable like any other
- Started and stopped like any other
- Able to use MODIFY commands for dynamic operations
- Configuration held in HFS or ZFS file systems
- Allows system automation tasks to control operations

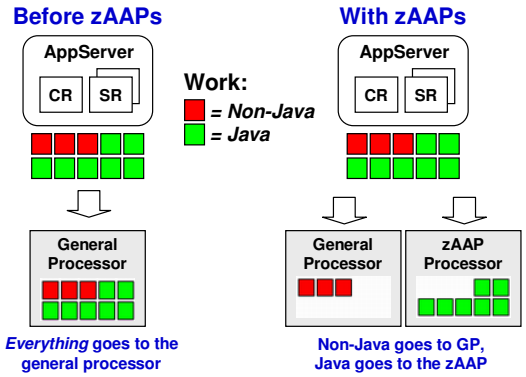
This is all standard stuff. The key is that WAS z/OS was implemented to be compatible with existing z/OS skills, and to take advantage of existing z/OS facilities. WAS z/OS is *not* merely UNIX processes running in USS.

WAS z/OS is implemented as a series of started tasks, which means that managing the environment -- starting, stopping, etc. -- should be very familiar to your current z/OS staff.

We draw your attention to the highlighted block. System automation tools on z/OS are capable of performing a great many tasks without operator intervention. Because WAS z/OS is implemented as started tasks, and it operates as such, system automation tools can be used to monitor and manage the WAS z/OS environment.

Exploitation of zAAP Specialty Engines

zAAP engines are Java offload engines. They enhance the financial picture of the z/OS platform, and they free up GP for other key subsystem processing



Keys to understanding value of zAAPs:

- **zAAP processors have a considerably lower acquisition cost compared to GPs**
- **Offloading Java to zAAP frequently allows growing non-Java work to live within existing GPs, thus avoiding capital acquisition**
- **Monthly license charges based on capacity of the system can be influenced by the presence of zAAPs, which do not count towards charges**

There are many technical details left unsaid here with respect to how they're configured, the rules for dispatching, when Java might go to GP, etc. Objective here was key points, not details.

This is really a function of the Java SDK and the dispatcher of z/OS.

The zAAP-enabled Java SDK is packaged with WAS z/OS, so WAS automatically takes advantage of zAAPs if they're present and configured

zAAP stands for "zSeries Application Assist Processor" … they are specially-designated processor engines on the System z machine that the JDK can use to offload Java work. By doing that you gain two benefits -- potentially lower software license charges; and capacity on your GPs that would otherwise have been used for Java.

WAS itself really knows nothing about this. This is a function of the Java z/OS JDK, which has code to interact with the system dispatcher so Java work gets switched to the zAAP.

Exploitation of WLM

Many view WLM exploitation as the heart of the platform exploitation model for WAS z/OS. There are four main elements of this exploitation …

Intelligent Dynamic Capacity Expansion

The ability to increase the number of JVM instances based on WLM goals and configuration settings.

This is the "Controller / Servant" structure you may have heard about

Intelligent Workload Flow Control

An element of the Controller/Servant structure. Inbound work is queued and held, waiting for a thread to select it, based on importance and arrival. It's a pull model rather than a push. Applications in JVMs take only what they can handle.

Intelligent Management of Mixed Work in Server

Multiple servants allows differently classified work to be placed in different servant regions. This allows WAS/WLM to understand what kind of work is in each and to manage system resources accordingly.

Intelligent Workload Routing Advice

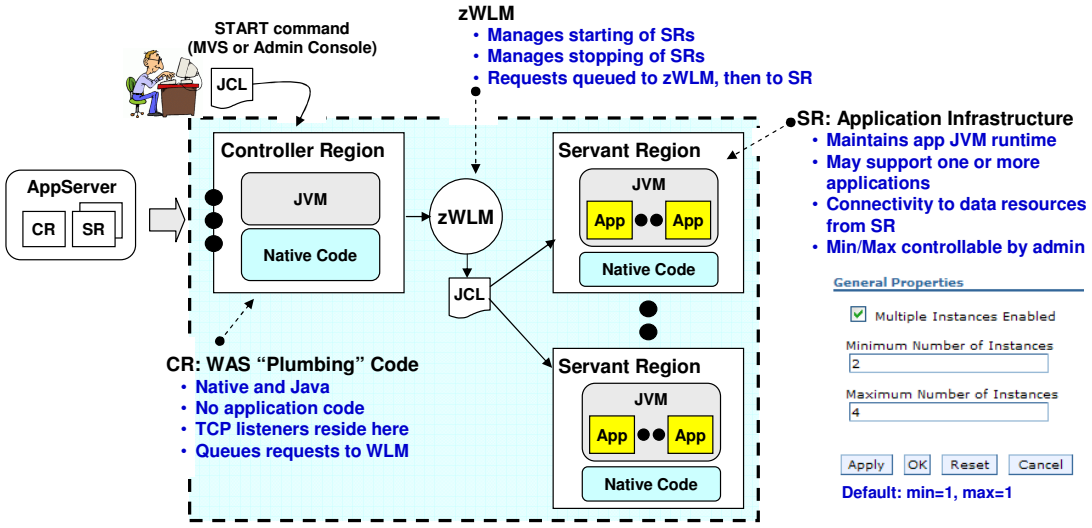
WAS z/OS using WLM to determine where best to route certain kinds of work

The key is the controller / servant architecture …

Exploitation of Workload Manager (WLM) is at the heart of WebSphere z/OS. We offer four categories of exploitation here. We'll cover some of them in the upcoming charts.

The Controller / Servant Architecture

This is a unique architectural element to the WAS z/OS design. No other platform has WLM:**



Let's now explore how this is accomplished …

** WebSphere on distributed uses the phrase "Workload Management" but it's not the same as zWLM

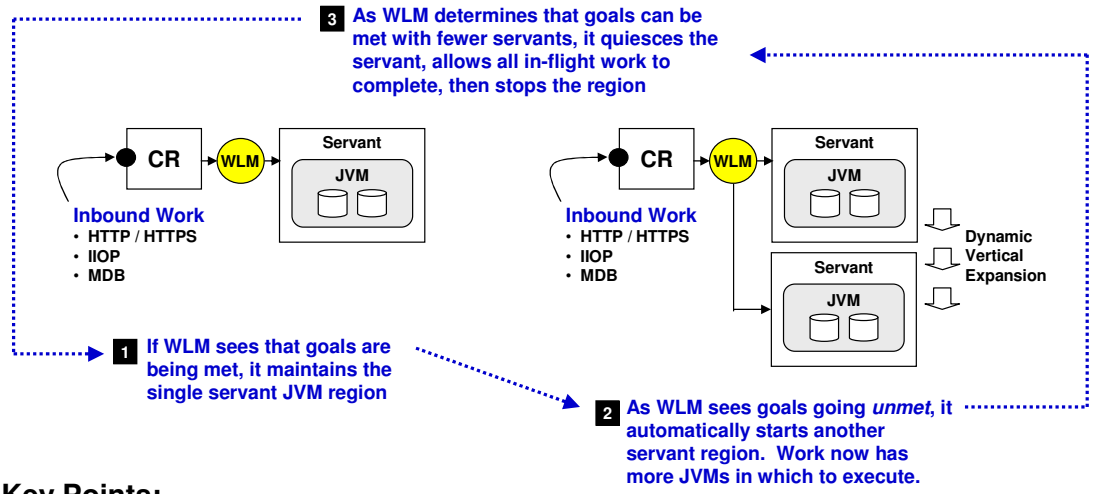
WAS z/OS has an application server architecture that is unique to the WAS family. Rather than consisting of a single JVM, it consists of multiple, each in its own address space: a controller region (CR) and one to many servant regions (SRs). This is configurable by you; the default behavior being one CR and one SR.

One of the key advantages of this structure is that it separates pure "plumbing" code from user application code. That allows the plumbing code to run in a more protected address space. And it protects the plumbing code from JVM outages caused by misbehaving user applications.

The chart offers a description of what function is contained in which region. WLM sits in the middle of this and performs several key functions, which we'll explore now.

Intelligent Dynamic Capacity Expansion

This is the "vertical scaling" capability of the multi-Servant structure. If allowed, WLM will start additional servant regions if it sees unmet goals:



Key Points:

- **The minimum and maximum number of servants is configurable. Default: Min=1, Max=1**
- **We see distributed WAS users trying to do something similar by configuring a "vertical cluster" to provide duplicate JVMs on a server box. Not quite the same -- no WLM assist of that**

One of the things this CR/SR structure allows is the ability to dynamically expand the capacity for a given application server by WLM starting a second (or more) servants. Again, the minimum and maximums are configurable by you, so it operates within the boundaries you set.

This is all based on WLM goals and WLM's monitoring of how well work is meeting the goals. If WLM sees that goals are not being met, and you've configured the server to allow for SR expansion, WLM will start up a second servant region.

Servants contain the JVM that runs the application code. When WLM starts another servant region, it creates an entirely new address space with a physically separate JVM from the first SR. Identical in every way but physically separate.

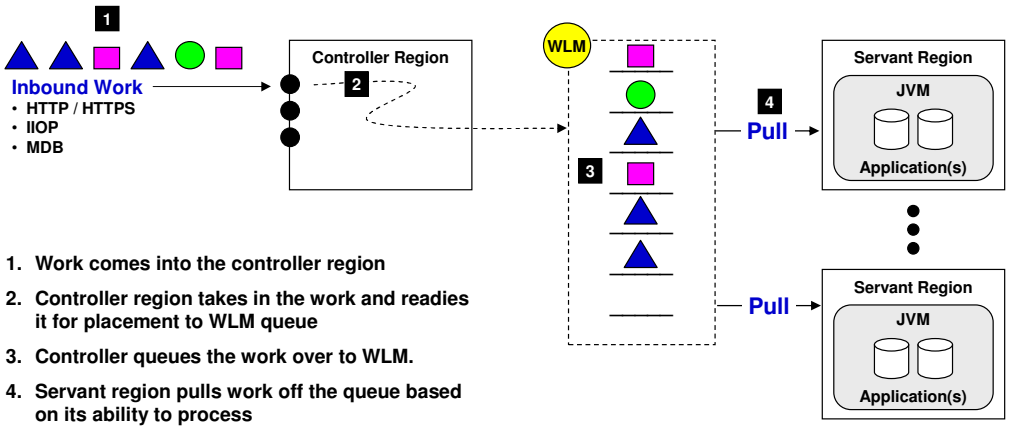
The inbound work comes to the CR, which then queues the work to WLM. The SRs pull the work from the WLM queue based on its ability to process work. This is a pull model, not a push model.

As WLM continues to monitor the goal attainment it may come to a point where it determines the additional SR is no longer needed. It will then stop work from flowing to that servant region, allow the in-flight work to flush out, and then the SR will be brought down.

Note: WebSphere on other platforms try to do something similar to this CR/SR structure by creating a WAS cluster "vertically" on the same server box. It's not really the same thing though because the other platforms don't have WLM management at the heart of it. The vertical cluster approach provides duplicated JVMs, but not a dynamic structure.

Intelligent Workload Flow Control

This is the WLM queueing mechanism that exists between the CR and the SR. It creates a "pull" model that prevents overwhelming an application JVM:



1. Work comes into the controller region
2. Controller region takes in the work and readies it for placement to WLM queue
3. Controller queues the work over to WLM.
4. Servant region pulls work off the queue based on its ability to process

Servant can't be overwhelmed

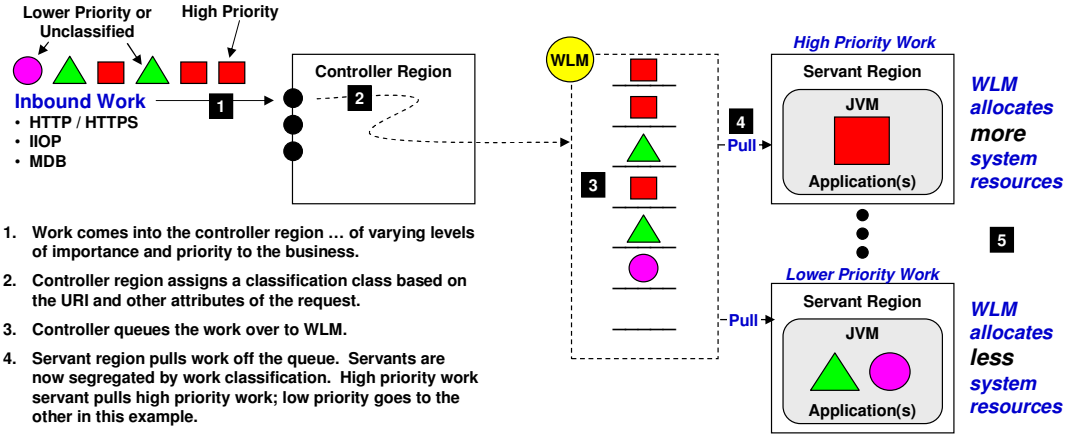
Servant only takes what it can. Controller will take in and queue up what can't be handled immediately.

We mentioned this on the previous chart -- the model between the CR and the SR is a pull model, not a push. That means that inbound work that hits the CR goes first to a queueing mechanism, then it gets pulled by the SR based on the SR's ability to do the work at that point in time. If a single SR is getting behind and WLM sees goals going unmet, then it'll start up a second servant region (provided it's configured to allow multiple servants).

This queueing mechanism also provides the ability to route work based on priorities to specific servant regions, where WLM can manage system resources servant by servant. That's next.

Intelligent Management of Mixed Work in Server

This involves inbound work being given a "Transaction Classification." With that, the CR can direct work to servants and WLM can manage:



1. Work comes into the controller region … of varying levels of importance and priority to the business.
2. Controller region assigns a classification class based on the URI and other attributes of the request.
3. Controller queues the work over to WLM.
4. Servant region pulls work off the queue. Servants are now segregated by work classification. High priority work servant pulls high priority work; low priority goes to the other in this example.
5. With work segregated by servant region, WLM can now manage the system resources given to each servant. High priority work gets more, lower priority less, all according to defined WLM goals.

Sophisticated Work Prioritization

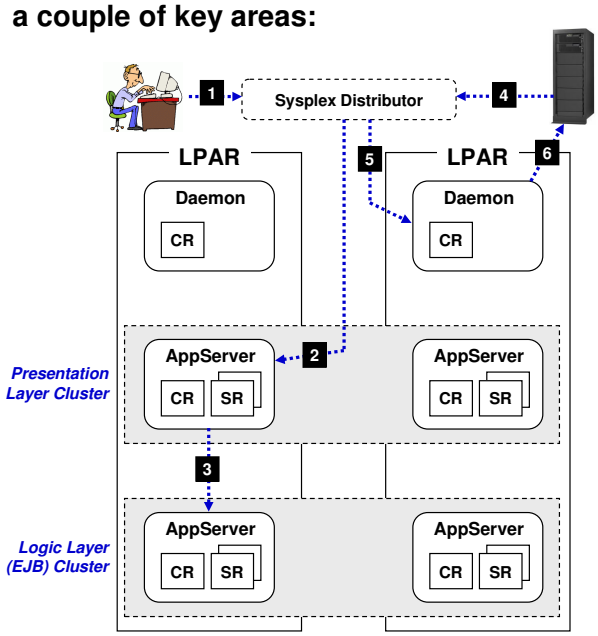
On other platforms this can only be done by allocating work to separate servers. No WLM there to manage at this level.

Work coming into a controller region may take many forms. Some of the work may be of higher priority than other work. The WAS z/OS CR/SR structure provides a way to segregate the work to individual servant regions so WLM can prioritize the allocation of system resources to achieve the goals. In this picture higher priority work goes to one SR and gets more system resources; lower priority work goes to the other SR and WLM grants it relatively less system resources based on the defined goals.

A single JVM appserver structure would not provide this ability. It would require creating multiple application servers and routing to the server based on priority.

Intelligent Workload Routing Advice

WAS z/OS relies on WLM to make routing decisions. We see this exercised in a couple of key areas:



Inbound HTTP or IIOP Work

1. Work comes in over network aimed at DVIPA and Sysplex Distributor
2. WLM advises Sysplex Distributor of presentation cluster member to place TCP request to
3. For servlet-to-EJB flow (IIOP) WLM advises server, which then places the IIOP request to one of the members in the EJB cluster

IOR Resolution

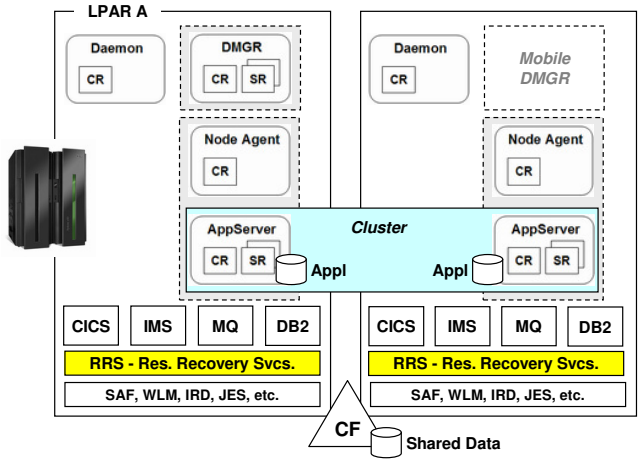
4. External client seeks to use EJB deployed in WAS. It addresses itself to DVIPA/Sysplex Distributor for bootstrap (not shown on picture), then gets DVIPA host for Daemon location service.
5. External client then come back to DVIPA/SD for Daemon location service. WLM advises Sysplex Distributor for best Daemon to place request.
6. Daemon consults WLM for object location best able to service external client at that time. External client provided with host:port for EJB in the cluster.

The point here is that WLM plays a key role in the routing decisions made by clients and WAS itself for real-time routing

Finally, WLM offers WebSphere z/OS advice on where to route certain work flows. This is based on a real-time awareness on the part of WLM for the system resources across the Sysplex (not just within a given LPAR) and the goals defined.

Exploitation of Resource Recovery Services (RRS)

Two-phase commit processing involves coordination of participants to make sure all are ready to commit. RRS plays that role in Parallel Sysplex:



- We'll see this picture later when we discuss high availability
- WebSphere Application Server is a transaction manager … it is able to initiate a transaction and have other resource managers (DB2, CICS, IMS) participate in the unit of work
- For two phase commit processing, *someone* has to play the role of syncpoint coordinator
- On z/OS and Parallel Sysplex that someone is RRS, which uses Coupling Facility data structures and patented recovery algorithms to provide very efficient failed transaction recovery
- WAS z/OS registers with RRS, as do resource managers. RRS handles the two-phase commit coordination

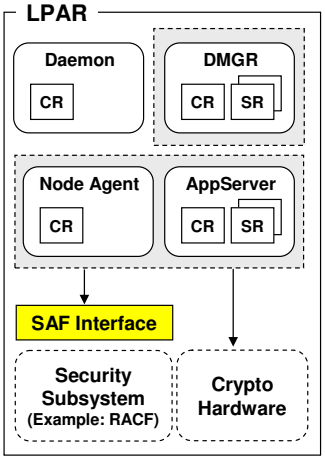
Another case of "below the specification line" exploitation of existing z/OS and Parallel Sysplex technology to perform a task in an optimized manner for the platform

Shifting away from WLM, let's now look at RRS. RRS is a Sysplex-wide syncpoint coordinator for global transaction management. All the major middleware subsystems on z/OS can take advantage of RRS for this service.

There are other technologies to do this … XA for example. But RRS has the benefit of being local (which implies a cross-memory connection) and algorithms that provide for very fast transaction recovery.

Exploitation of SAF and Crypto

SAF is a security interface; Crypto is a hardware-assist processor for encryption and key storage on the System z and z/OS platform



SAF Security Subsystem

- Sysplex-wide integrated security repository
- Single location for security artifacts rather than scattered model
- IDs, groups, keyrings, certificates, EJB role enforcement
- Local access … unlike LDAP, do not need to traverse network
- Extremely robust security model

Crypto Hardware

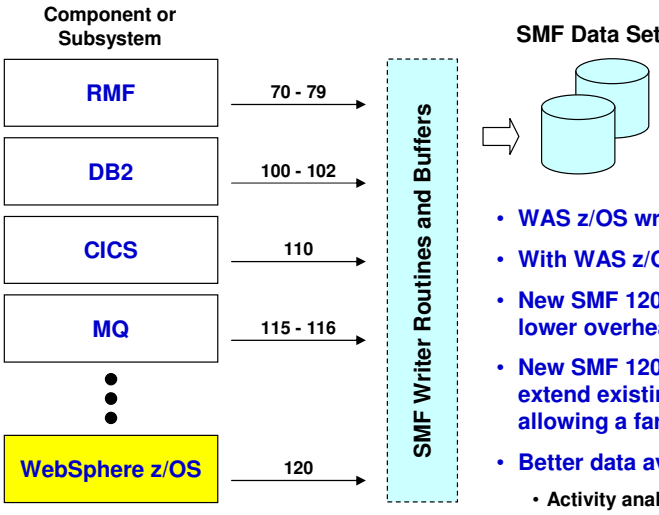
- Hardware-assisted cryptographic encryption and de-encryption
- Extremely secure private key store management

Properly configured, z/OS provides an extremely secure environment … many say the most secure available

Continuing … WAS z/OS exploits the SAF interface of z/OS, which provides a consolidated security repository which is Sysplex-wide. SAF, and the products under the interface such as RACF, provide an extremely secure mechanism for securing not just userids and passwords, but access to servers, system resources and application roles. The access is cross-memory, not cross-network, so latencies are reduced.

Exploitation of SMF

SMF is an activity recording facility of z/OS that allows subsystems to record key activity for analysis, management and accounting chargeback



- WAS z/OS writes SMF 120 records
- With WAS z/OS V7, a new subtype was created: 9
- New SMF 120 subtype 9 provides better data with lower overhead cost
- New SMF 120 subtype 9 records complement and extend existing SMF from other subsystems, allowing a far better picture of what's going on
- Better data available for …
  - Activity analysis
  - Usage statistics
  - Accounting chargeback

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

SMF is a mechanism that allows subsystems on the platform to write activity records into a repository that may then be used for analysis. The chart shows the SMF record types written by the various subsystems, including WAS z/OS.

With WAS z/OS V7 the SMF support has been enhanced to provide better information at a lower cost of overhead. There's an excellent white paper at the URL shown on the chart that gives all the details of this new record type.

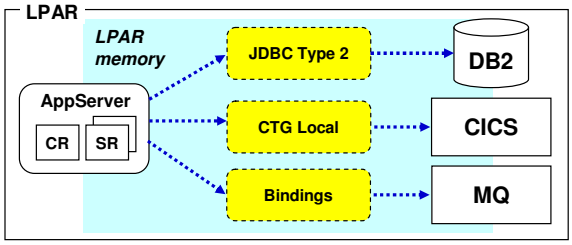
The key value of SMF is this -- it provides a much better view of what's going on within the system than might otherwise be available. That information can be used for understanding usage patterns, planning for additional capacity, and for accounting chargeback purposes.

Knowing who's using what resources is an important element of running the I/T operations. SMF allows you to do just that.

Exploitation of Cross-Memory Communications

Any time client and target are in the same LPAR, there's an opportunity for cross-memory exploitation. Let's look at a few examples:

Data Access

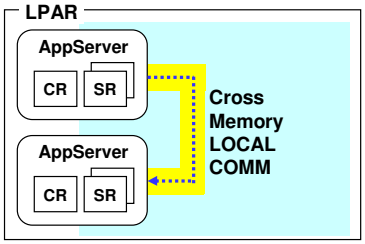


Benefits:

- Cross memory speed
- Security ID propagation (no alias)
- Exploitation of RRS
- Avoid serialization of parameters
- Avoids SSL overhead
- Single thread of execution

LOCAL COMM

Used for IIOP flows between servers on the same LPAR.



Benefits:

- Avoids IP stack entirely
- Avoids SSL overhead
- Very fast, very secure

Extension to Local Comm: new Optimized Local Adapters ...

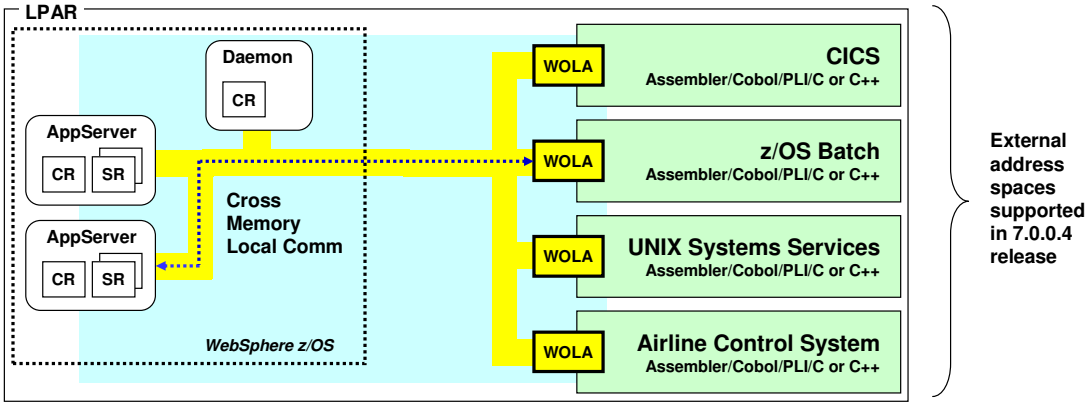
When application servers and data subsystems reside on the same operating system image, it opens up the possibility of using cross-memory communication services. This helps avoid the cummulative latency typically associated with network protocols. But it also provides some other advantages you may not be aware of. The chart offers a list of bullets that briefly summarizes those benefits.

In addition to local communication outbound from WAS, communications within a WAS cell between servers uses something called Local Communications, or Local Comm for short. Local Comm is a high-memory transfer mechanism that avoids TCP entirely, which means less code-path to do the communications. There's no need to encrypt when going memory-to-memory so the overhead of SSL is avoided. And it's very fast.

This Local Comm facility was exploited in WAS z/OS V7.0.0.4 to provide a new kind of cross-memory connector called WOLA, which we describe next.

Cross-Memory: New Optimized Local Adapters (WOLA)

New in V7.0.0.4, this new function allows external address spaces to participate in a cell's Local Comm communications.



Benefits:

- Based on Local Comm (z/OS exclusive)
- Bi-directional … WAS outbound or inbound to WAS (WOLA exclusive)
- CICS Security and Transaction propagation (some restrictions apply)
- Faster than other local solutions

WP101490 on ibm.com/support/techdocs for more

WOLA is a new cross-memory communication mechanism that is really an extension of the Local Comm structure we talked about in the previous slide. By extending the Local Comm structure it allows WAS to talk out to external address spaces, and for them to talk into WAS.

Focus on the "z/OS Batch" block … imagine batch programs being able to drive into WAS and re-use WAS EJB assets as part of their batch processing. That's just one example of a usage pattern that before WOLA would have had to be done with less efficient technologies.

Performance measurements show this technology capable of sustaining many times the comparable throughput for other mechanisms such as web services. Good performance improvements are seen even for other local communication techniques.

There's a white paper on Techdocs -- WP101490 -- that provides more on this function. If this strikes you as a compelling opportunity for co-location exploitation, please pull that paper and read through it.

Summary of the Business Value Benefits of Active Exploitation

Broad Guiding Principles of Business Value:

- **Available and Reliable**
  The active exploitation elements take direct advantage of Parallel Sysplex and the shared data clustering capabilities of the technology. Properly configured, no other platform offers the same degree of availability and reliability.
- **Manageable**
  The active exploitation of system components such as WLM, SAF, RRS and SMF allow for management using proven and mature technologies.
  Key point: WAS z/OS uses and exploits well known and understood system services. There's no need to develop new tools for operations, capacity planning, performance management, etc. This is building on existing knowledge.
- **Flexible**
  The active exploitation of z/OS interfaces such as MODIFY provide dynamic operations at the OS level against the operating WAS servers.
- **Affordable**
  One element of contribution to the expense question is the zAAP specialty engine for Java offload.

  Again, the value proposition of System z and z/OS is centered around efficient sharing of resources. We cover the cost discussion at end of presentation.

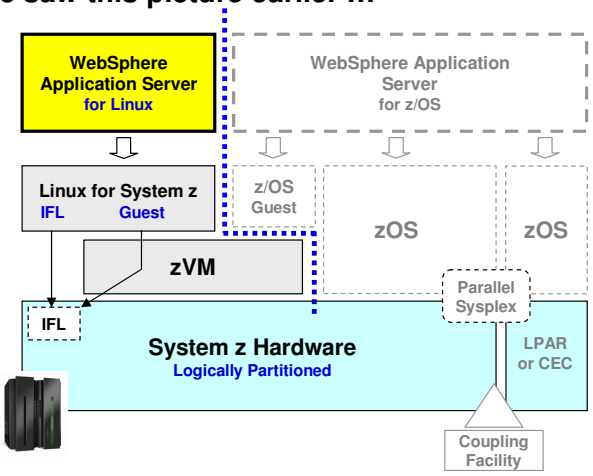
As we did earlier, here we map the direct exploitation capabilities to broad categories of business value.

WAS and Linux for System z

To understand the key differences and position the two System z offerings

Our Earlier Picture

We saw this picture earlier …



All the WAS z/OS pieces of that picture have been grayed-out

So too has the Parallel Sysplex and Coupling Facility

WAS for Linux on System z can not *directly* participate in Parallel Sysplex or the attributes of the z/OS operating system.

***Active* exploitation of hardware virtualization through zVM is realized**

***Passive* benefit from the System z hardware is realized.**

***Indirect* participate in Parallel Sysplex is possible …**

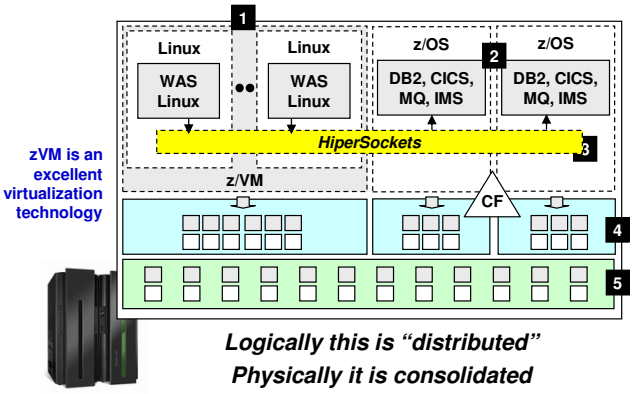
WebSphere Application Server is a cross-platform solution, as we stated earlier. The question often comes up about WAS on Linux for System z. The picture above shows how that can play in a System z environment.

Linux for System z is *not* z/OS, which means it does *not* have the facilities of z/OS such as WLM, RMF, SMF, SAF. But Linux for System z does run on the System z hardware within a logical partition, so it does receive the passive benefits of the hardware platform and the virtualization technologies we discussed earlier.

And while Linux for System z can not participate in a Parallel Sysplex (that's z/OS only), it can receive indirect benefit from being on the same server platform as a z/OS Parallel Sysplex. That's next.

Common Use of WAS Linux System z and Parallel Sysplex

Many access data on z/OS Parallel Sysplex from Linux LPARs in the same CEC, using HiperSockets for TCP access:



1. **Linux LPARs running WAS**
   Exploiting virtualization of z/VM to realize server consolidation to single HW footprint
2. **Data subsystems in Parallel Sysplex**
   Exploiting data sharing
3. **HiperSockets**
   Optimized cross-LPAR virtual network
4. **Virtualized resources**
   Exploiting LPAR technology of System z
5. **Real resources**
   Efficient sharing of real resources through HiperVisor allocation of real to virtual

This is primarily used for server consolidation. Advantages over server farms include reduced power, cooling, square footage, administration and potentially software

Very good, but it does have drawbacks compared to co-location on z/OS …

This is a very common solution topology we see -- a multi-LPAR Parallel Sysplex providing highly available data services, with WAS on Linux for System z in another LPAR using HiperSockets to access the data.

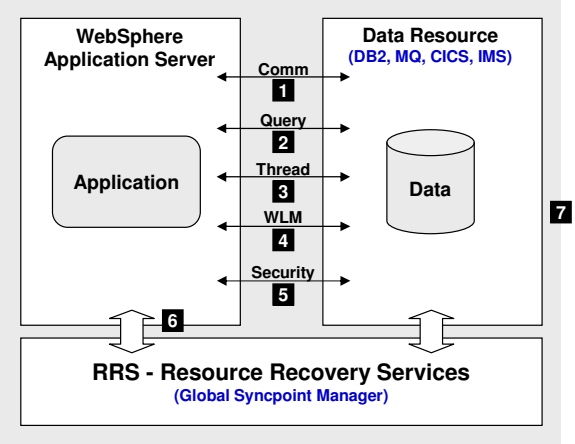
The virtualization capabilities of the System z platform are exploited. The dynamic movement of system resources between LPARs is possible between z/OS and an LPAR running zVM.

The motivation for this kind of design is commonly driven by a consolidation desire … a server farm that has grown out of control is consolidated back to multiple Linux images running under zVM on the System z hardware.

But this is not the same thing as co-location with data using z/OS. We look at that next.

Linux vs. Co-Location on z/OS

This gets to the "Value of Co-Location" as outlined in WP101476 paper on `ibm.com/support/techdocs`:



1. **Cross-memory data transfer**
   Better than even HiperSockets
2. **Avoid data and parameter serialization**
   Since not passing across network, do not need to serialize. Avoids SSL as well.
3. **Single thread of execution**
   Avoid switching threads, which means even greater efficiency
4. **Manage to a single WLM goal**
   Easier goal definition and management
5. **Passing security context**
   More options for security identity propagation: servant ID, client ID, application role vs. alias for remote T4
6. **Sysplex-wide RRS**
   Extremely efficient transaction recovery processing
7. **Reduced complexity**
   Single OS, more focused problem determination

Important point here … solution architectures that span multiple operating systems environments implies different monitoring and management capabilities. Correlating information between those different tools can be challenging. Co-location helps reduce that complexity by bringing it all under a single operating system environment.

Linux on another LPAR on the System z platform coming across HiperSockets still uses TCP. It's very fast because HiperSockets is a cross-memory type of TCP implementation. But it's still TCP, which means that certain things are lost when compared to co-locating with the data.

The message here is one of performance, simplicity and control. Co-location affords the opportunity to exercise the capabilities of the z/OS environment to form a simpler overall topology, which greatly reduces problem determination efforts. The components are within a single operating system environment, there is (commonly) a cohesive team involved with problem determination efforts on the mainframe, and there's a rich set of system facilities to aid in the problem determination.

Summary of the Business Value Benefits of Linux on System z

Broad Guiding Principles of Business Value:

- **Available and Reliable**
  z/VM and IBM's Linux kernel are proven and reliable. No direct exploitation of Parallel Sysplex, but access to data elements on z/OS provide for a degree of that at the data layer.
- **Manageable**
  Linux is Linux, but the z/VM tools at the virtualization layer are extensive and mature.
- **Flexible**
  This is one of the key value statements of Linux on System z. The virtualization capabilities translate directly into rapid flexibility.

  Software vendors often limit their support to a few operating systems, Linux often being one. Linux on System z allows you to take advantage of the platform and maintain ISV support.
- **Affordable**
  Consolidation of servers from distributed to virtualized Linux serves on z/VM provides reduced environmental costs, potentially reduced administrative costs, and potentially reduced software licensing costs.

  Every situation is different. An analysis of cost savings is recommended.

Mapping to broad business value …

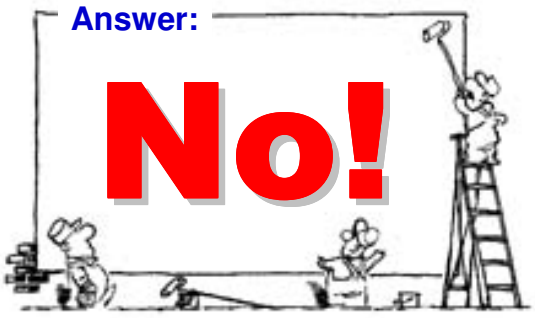
Application Design Considerations

Exploring what makes a Java application "z ready"

Is Java on System z Different?

Is there anything special that needs to be considered when designing and writing Java code to run on System z or z/OS?

Answer:



Clear enough? ☺

Java is Java

The point of an open standard application platform is to eliminate platform dependencies

Two points:

1. **That wasn't always the case … earlier we had ASCII / EBCDIC issues. No more.**
2. **There are poor coding practices that make bringing applications to z/OS problematic …**

One of the questions that comes up time and again -- less so recently, but still some -- is whether writing Java code for WAS on z/OS is "different" in any way.

The answer? Well … the chart makes it clear.

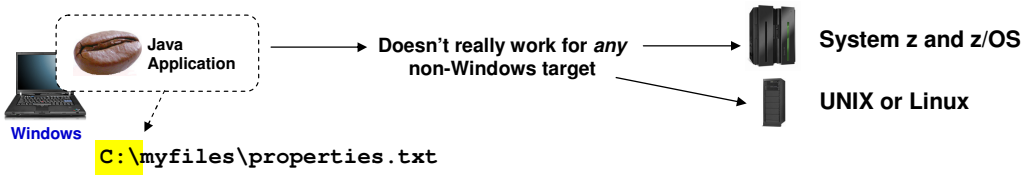
It used to be different many years ago, but that's changed and now there's a common Java environment -- remember our earlier discussion about the specification line and above where "WebSphere is WebSphere?" -- across the platforms. Exploitation is below the line. We've covered a great deal of that exploitation already.

But that's not to say that all Java code is equally well written. We have seen cases where poorly designed code "breaks" in some fashion when it comes to System z. It's not because Java on System z is "different" … but because there are issues with the application.

Compatibility -- Two Common Problems

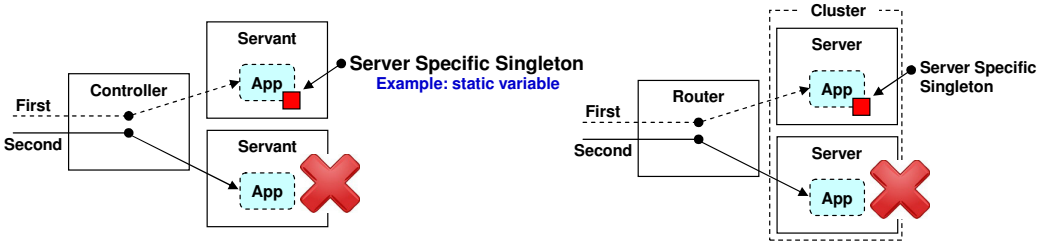
And neither is a "problem" with the platform, but rather a problem with Java code that makes improper assumptions

Hard-coded references to properties or output files



Assuming a single JVM instance

This is different from the use of HTTP session objects. That is a container-managed mechanism and can be handled in multi-JVM cases



Both not recommended since the underlying assumption is incorrect

In our experience with Java code we've seen a couple of common problems that cause a breakage when the code is brought to System z.

The first is simple enough -- hard-coded references to pathing that is Windows-specific and not UNIX oriented. That problem would surface if the code was brought to any non-Windows platform.

The second problem has to do with WAS z/OS's CR/SR structure. If multiple SR instances are running and the code employs a server-specific singleton (something that resides only within that JVM and is not available externally or copied to other JVMs), then it's possible that another instances of the application will not function properly when a request flows to the other instance, such as in the case of a second SR.

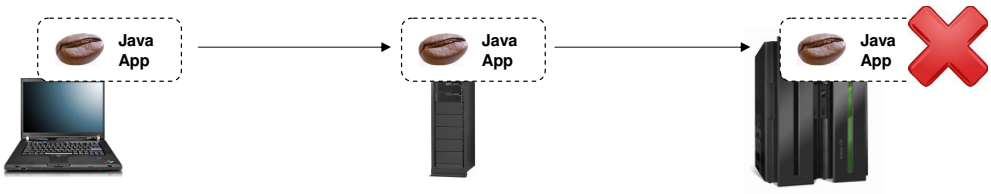
But truth is, this would happen in any environment where multiple JVM instances are maintained, such as a WAS cluster on any platform.

Both of these are examples of improper practices because there are incorrect assumptions made about the portability of the code.

Efficiency -- Write Smart, Tight Server Side Code

An application expected to scale up to heavy user rates must be sure to use very efficient server side code

We've seen this often in the benchmark center …



Development
Simple component testing

Test
Functional validation and perhaps some limited system integration

Production
Subjected to heavy user traffic. Application under-performs or crashes

In every case an investigation revealed inefficient coding practices on heavily used components. Once fixed, the application scaled to the desired levels.

There are many different examples of inefficient coding practices. Going into all of them is outside the scope of this presentation. Key message is: code expected to scale must be efficient.

The second major issue we see is one of performance. We've seen cases where code is brought up to z/OS where the performance is seen as disappointing. But in each case the real problem was one of inefficient code that didn't scale well when brought to a high-volume platform like System z.

This is not an issue with the platform but an issue with the application design. Code destined for high scaling must be written efficiently to scale properly.

Efficient Code Advice

Some broad pieces of advice …

- **Have a performance expectation documented**
  - Called a "performance budget" … this guides development efforts and helps set proper expectations
- **Understand where time is spent**
  - Tools like JINSight help you analyze -- "profile" -- code
- **Write heavily used code as tight as possible**
  - Applications typically have some components more used than others -- optimize those
- **Consider JVM co-location and local methods**
  - This is a way to avoid "expensive" remote EJB calls
- **Take advantage of caching where possible**
  - Caching within application, or use WebSphere dynamic caching
- **Use statefulness wisely**
  - Creating statefulness creates affinities, which hinder scalability and availability
- **Trust But Verify**
  - Popularity in open source communities does not guarantee the package is robust, efficient and scalable.

This chart provides some key bullets that provide for good code that scales well.

Summary of the Business Value Benefits of Good Java Code

Broad Guiding Principles of Business Value:

- **Available and Reliable**
  Unexpected outages due to inefficient code designs are avoided
- **Manageable**
  Properly architected Java solutions are more easily managed. Consistent design practices allow for consistent management approaches.
- **Flexible**
  Properly architected Java solutions allow for easier integration of new features and functions. Poorly designed solutions require more time and effort to incorporate changes or expose as services.
- **Affordable**
  Classic case of "pay me now or pay me later" … spending time up front to design and craft good code pays dividends down the road in terms of reduced adminstrative costs, reduced resource consumption, and reduced cost to the business from outages or insufficient service to the customers.

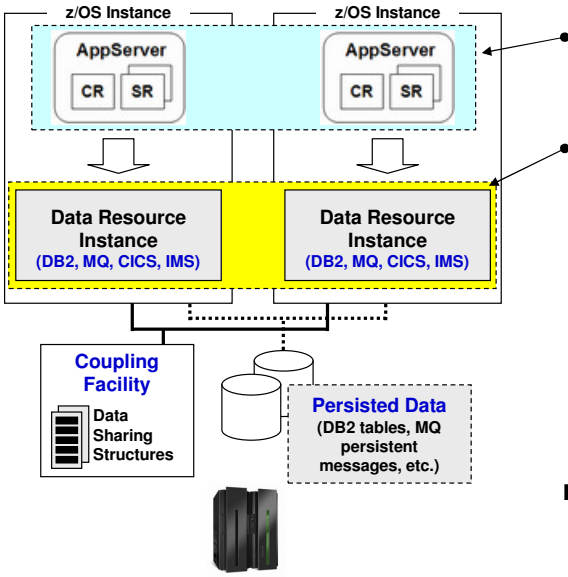
Business value …

Availability and Scalability

Two of the more common business drivers for System z and z/OS

At The Heart -- Sysplex Data Sharing

Parallel Sysplex data sharing provides duplicated access to the same data. Data access and locking issues provided by Coupling Facility and Subsystems



- **WebSphere "Cluster" consists of multiple physical application servers**
  They are physically separate in most ways. Together it represents a logical one.
- **Sharing Group**
  Physically separate instances but organized so they understand participants in group and have defined sharing relationship
- **AppServers and Resource Instances**
  An application in an AppServer interacts with a data resource *instance*. Sharing conflicts are resolved by the data resource instances working in concert with each other with the help of z/OS and the Coupling Facility.

Parallel Sysplex and Data Sharing has been around for a decade and more. The technology is mature and proven and in use by large customers the world around.

We turn our attention to the question of availability and scalability. On the System z platform the heart of this is Parallel Sysplex and the data sharing capability in the middle of that.

Parallel Sysplex is a clustering technology that has a shared data facility in the heart of it. That's done with the Coupling Facility, which maintains shared data structures, cache instances and locking information.

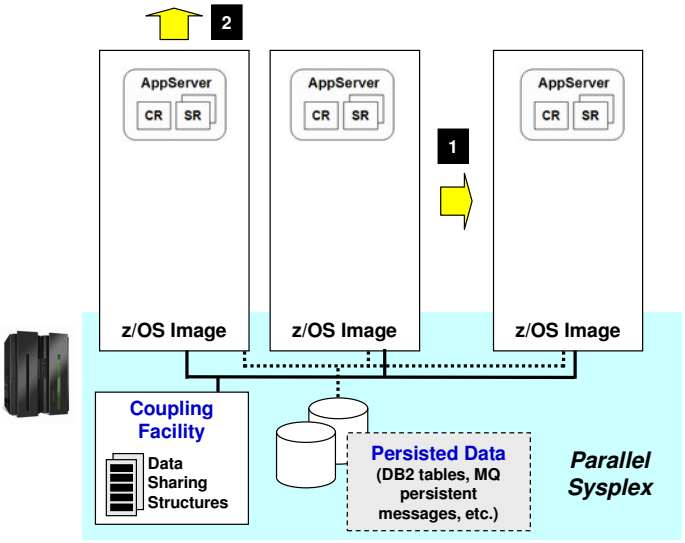
What this means is that data subsystems such as DB2, MQ, CICS and IMS can participate in a data sharing arrangement across the Sysplex. This creates a single *logical* data model even though multiple data subsystem instances are present. The data is accessible from any one of the participating data instances. Issues of data locking are managed within the coupling facility.

With that structure in place then WAS z/OS itself can cluster on top of that. The applications are now duplicated across a logical cluster, with local access to data. Within the WAS z/OS cluster there are high availability mechanisms to failover key services if one part of the cluster is lost.

HA is very broad topic and we can't go into every one of the details here. The key is that Parallel Sysplex and data sharing isn't some new flash-in-the-pan technology. It's been around for a while and it's proven and reliable.

Scalability

Two kinds of scalability -- Horizontal and Vertical



1. Horizontal Scaling

This is what people most often think about when they think of scalability.

It can work, but it gets increasingly difficult unless you have an effective shared resource (data) clustering mechanism.

Parallel Sysplex is just such a mechanism.

Shared disk storage systems, proven locking mechanisms, in-memory data structures and caching (CF) all make for effective horizontal scaling.

2. Vertical Scaling

Vertical scaling is often overlooked. The result is massive horizontal scaling with all the attendant issues of manageability.

z/OS is designed for high degrees of utilization and has the capability to scale very high per system image. The balanced architecture (CPU, memory, cache, I/O) allow for this.

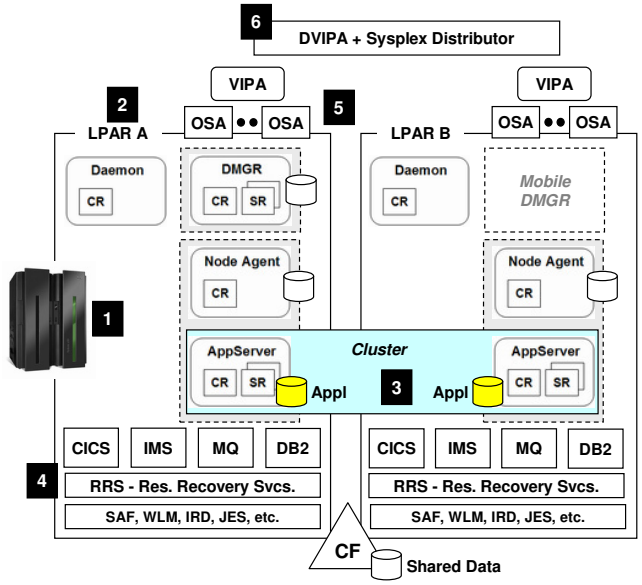
System z and Parallel Sysplex provides both. That's the design point of the platform. That's how it's used in many large customer installations.

When it comes to scalability there are two kinds -- horizontal and vertical. We spoke of the data sharing in the middle of the Parallel Sysplex, and that plays a key role in the ability of the Parallel Sysplex to scale horizontally.

Vertical scaling is one that's most often overlooked. The key to z/OS participating in this is not just the ability to load a System z LPAR with CPU and memory (which can be done, of course), but also the way z/OS can manage multiple workloads within the same operating system instance. This allows workload to scale up in a server instance to handle spikes, then as work scales back other work can come back in and continue to utilize the server.

The Big Picture of WAS z/OS and Parallel Sysplex HA

It's all about redundancy *and* integration with platform HA function

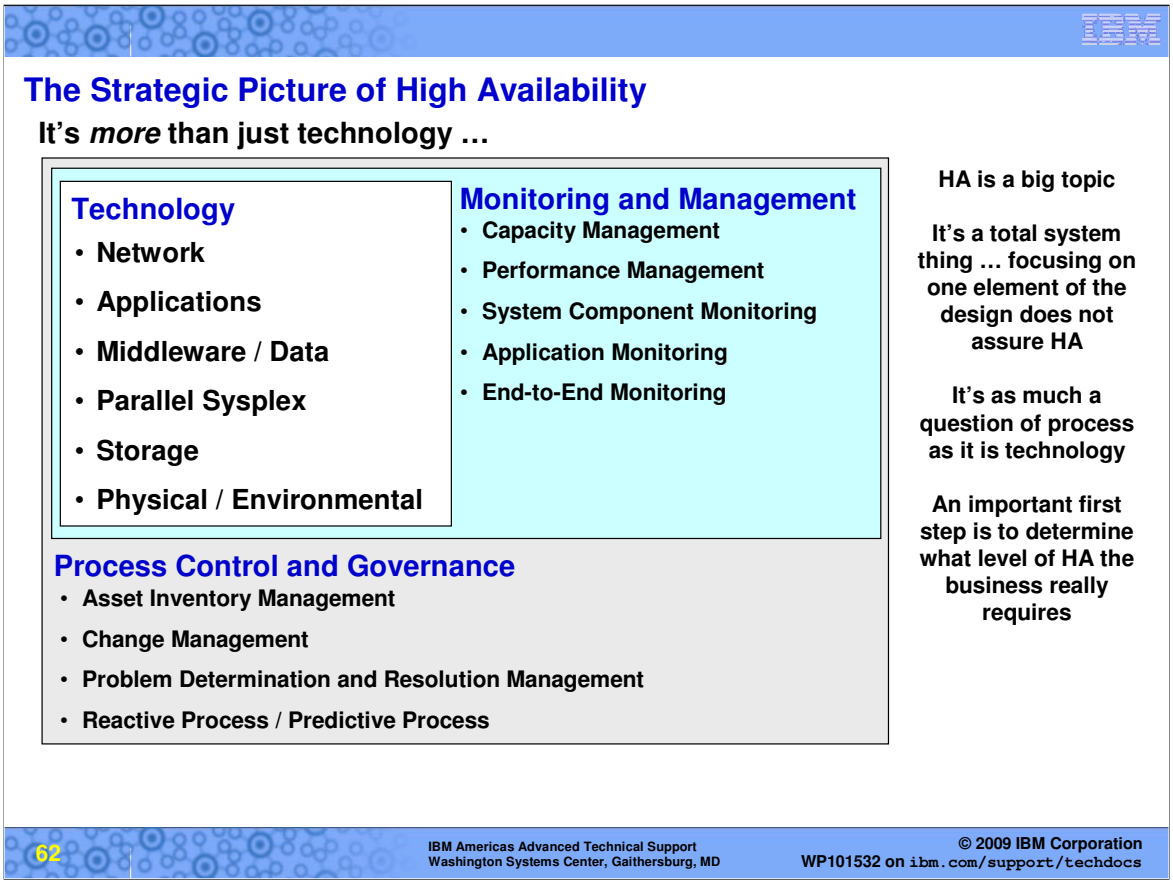


We show two operating system instances. That can be higher for greater availability and more manageable failover

1. **Redundant and fault-tolerant hardware**
   - System z hardware design has many layers of fault tolerance and redundancy.
2. **Redundant z/OS instances**
   - Either through logical partitioning (LPAR) or separate physical machines.
3. **Clustered WebSphere z/OS servers**
   - Multiple application servers grouped into a logical unit for application deployment and management
     - z/OS exclusive: dynamic SR expansion (more coming up)
4. **Redundant data resource managers with Sysplex shared data**
   - Multiple resource managers instances with shared data in CF and a global syncpoint manager (RRS)
5. **Redundant network adapters hidden behind Virtual IP address**
   - On the front end, multiple network interfaces with a moveable virtual IP address protecting against outage
6. **Workload distribution hidden behind distributed virtual IP and Sysplex Distributor**
   - Further abstraction of real IP addresses behind a virtual IP that can be swapped across images in a Sysplex, with Sysplex Distributor providing TCP connection distribution based on WLM

61 IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD
© 2009 IBM Corporation
WP101532 on ibm.com/support/techdocs

Here is the high-level view of the high availability story for z/OS and Parallel Sysplex, and the way WAS z/OS exploits those capabilities.



It's worth reminding the reader that high availability is more than just technology. There are really three categories of disciplines that must work together in concert to provide HA.

The message is that System z and z/OS provides significant contributions in the technology space and the monitoring and management space. And System z staffs traditionally have well-defined process controls already in place.

Bottom line … System z is widely considered to be *the* platform for high availability.

Summary of the Business Value Benefits of Scaling and HA

Broad Guiding Principles of Business Value:

- **Available and Reliable**
  This is the heart of this topic. The System z hardware design is inherently stable and mature (passive benefit). The z/OS operating system is inherently stable and mature (passive and active benefit). The design of Parallel Sysplex provides for both scalability and availability. The key value attribute of the platform.
- **Manageable**
  Mature systems management tools make managing a Sysplex effective.
- **Flexible**
  WLM and IRD represent extraordinary flexibility of resource allocation across a Sysplex within a CEC. Many elements of the hardware and system software design provide hot-pluggability and dynamic vary online and off.
- **Affordable**
  Higher initial acquisition with lower overall, depending on a number of costing factors. We cover this at the end of the presentation.

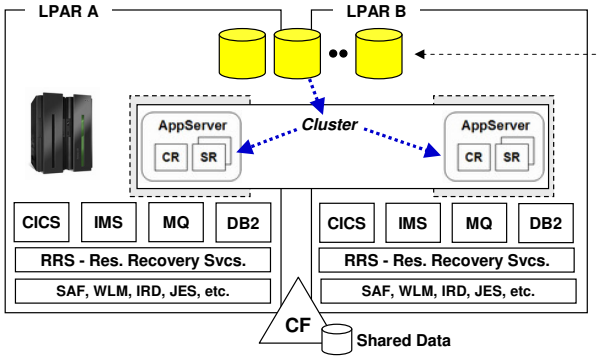
Business values.

Other WAS-Based Solutions

A quick note about IBM's strategy to leverage the WAS z/OS solution

Other WAS-Based IBM Solutions

We see many IBM z/OS solutions being built as Java EE applications that leverage the existing foundation of WAS z/OS:



Examples:

- **WebSphere Process Server (WPS)**
- **WebSphere Enterprise Service Bus (WESB)**
- **WebSphere Service Registry and Repository (WSRR)**
- **WebSphere Extended Deployment**
  - Virtual Enterprise
  - Compute Grid
  - Extreme Scale

Why reinvent the wheel? IBM has a robust, platform-exploiting Java EE runtime environment in WAS z/OS. It's proven and it works well.

Why not just piggyback on all the key services it provides (security, transaction, containers, data access) that WAS z/OS provides? That's exactly what's done.

All the benefits that accrue to WAS z/OS accrue to these solutions as well

IBM provides many higher level solutions that are based on the WebSphere Application Server as the Java runtime framework. Examples of these are listed on the chart. We don't intend to go into the details of each of those.

The main point is that all the benefits that accrue to WAS by running on z/OS also bubble up to the solutions that run on WAS z/OS.

Summary of the Business Value Benefits of Leveraging WAS

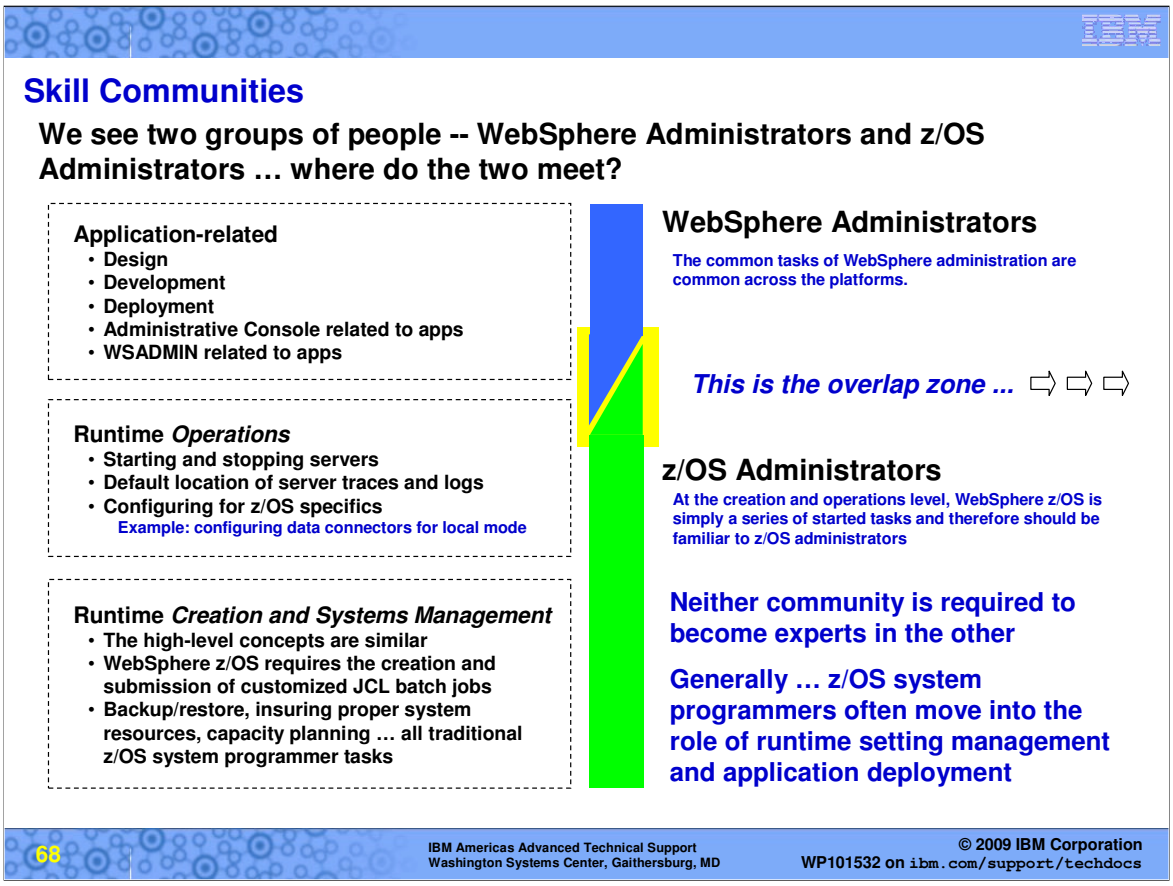
Broad Guiding Principles of Business Value:

- **Available and Reliable**
  All the availability and reliability statements made earlier apply to the solutions built on the Java EE base of WAS z/OS
- **Manageable**
  All the manageability statements made earlier apply to the solutions built on the Java EE base of WAS z/OS
- **Flexible**
  All the flexibility statements made earlier apply to the solutions built on the Java EE base of WAS z/OS
- **Affordable**
  Higher initial acquisition with lower overall, depending on a number of costing factors. We cover this at the end of the presentation.

Business values.

Administration and Skills

Where the z/OS skills intersect with WAS skills and how to manage the overlap



A very common question that comes up is how WAS on z/OS can be managed from a staff perspective. Often we see cases where WAS on another platform already exists and there's a set of trained administrators familiar with WAS, but are not z/OS administrators. Similarly, there are z/OS system programmers who are deeply skilled in that area, but not familiar with WAS. So where's the line to be drawn? This chart helps explain a working approach this separation of duties.

Common Patterns of Skills Development

Some observations from other customers:

WebSphere Administrators Seeking Some Knowledge of z/OS

Not required … WAS administrator can be effective with no z/OS knowledge if z/OS environment "hidden" from them

Generally speaking, the objective is to provide knowledge of common activities:

- Logging into TSO, a little on using ISPF
- Becoming familiar with concept of started tasks
- Understanding how to browse held output in JES
- Knowing how to check started task status
- Starting/stopping servers (maybe)

Objective is rarely to make them z/OS administrators. *No need* to learn in-depth:

- WLM classifications
- RACF or other SAF product
- JCL, z/OS commands, etc.

We often see access to z/OS using familiar tools:

- Telnet / FTP for UNIX access
- WebSphere Admin Console (essentially common)
- WSADMIN scripting (common across platforms)

z/OS Administrators Seeking Some Knowledge of WebSphere

Some knowledge is helpful

Generally speaking, the objective is to provide conceptual framework of WebSphere:

- Understanding the runtime architecture
- Construction of runtime
- Key systems management of runtime

Objective is rarely to make them deep WAS administrators or Java programmers. *No need* to learn in-depth:

- Java programming
- Developing tooling

Frequently see z/OS personnel desire to expand skill set and learn things like:

- JVM monitoring and tuning
- Application deployment techniques
- WebSphere runtime customization details

The key point of this chart is to send the clear message that each skill community can focus on what it does best without having to learn everything about what the other community does. *Some*, perhaps, but not a complete overlap. This is an important point because we often see some initial apprehension to WAS on z/OS *because* of a misunderstanding about this.

Summary of the Business Value Benefits of Skill Communities

Broad Guiding Principles of Business Value:

- **Available and Reliable**
  The platform provides the ability to provide this; your System z and z/OS staff is what makes it work. WAS z/OS is implemented in a way where existing z/OS skills can be effectively leveraged without requiring massive re-learning of new technologies.
- **Manageable**
  WAS z/OS directly exploits many standard z/OS management functions: MODIFY, SAF, SMF, RMF, WLM.
- **Flexible**
  WAS z/OS is designed to directly exploit the z/OS flexibility attributes.
- **Affordable**
  Existing z/OS staffs have effectively managed the addition of z/OS without additional staffing. Existing distributed WAS administrators can use WAS z/OS with little or no awareness of the different platform.

  The key is effective and efficient use of *existing* staffing resources.

Business values.

Addressing the Issue of Cost

Acquisition cost versus Total Cost of Ownership

Three Costs Go Into Total Cost

It is an often complex undertaking to calculate the true total cost

Acquisition Cost

+ Recurring Operating Cost

+ Cost of Lost Service

= Total Cost over Time

- Software license charges
  Remember: zAAP, zIIP help with this
- Power consumption charges
- Cooling charges
- Square footage charges
- Administrative personnel

- Cost of outages and insufficient service
- Cost of security breaches
- Cost of underutilized assets

We are *not* minimizing the pressures you face to keep acquisition costs low

We are *not* suggesting determining some of these costs is easy

We are asking that give consideration to the total cost picture

Cost is on everyone's mind. Cost involves many things … not just acquisition costs, though.

What this chart is trying to do is set the stage for a discussion of cost and cost justification by looking at the bigger picture. In doing so, please do consider the three lines at the bottom of the chart. We truly are not dismissing the pressures you face with acquisition costs. We are simply asking that the analysis consider other costs as well.

Really Quick Primer on Fixed Cost Allocation

This is at the heart of the "total cost of ownership" (TCO) issue. Let's imagine a hypothetical household:

Mortgage	Property Taxes	Insurance	Utilities	College Tuition	All Other Costs
$36,000 / year	$10,000 / year	$5,000 / year	$5,000 / year	$30,000 / year	$14,000 / year

$100,000 per year

The objective is to allocate that across some "marginal" (incremental) unit

(Reason is because businesses wish to charge operating units to cover overhead, and because they want to estimate profit per unit)

Per Person?

$100K / 4 = $25K / person

Per Square Foot?

$100K / 3000 = $33 / sq. foot

Per Pet?

$100K / 2 = $50K / pet

Per Kilowatt Hour?

$100,00 / 14kWh = $7,000 / kWh

Per Meal Consumed?

$100K / (4 * 365 * 3) = $23 / meal

This "cost" per unit can affect future purchase or activity decisions …

"Having your mother move in would cost us $25K extra!"

"That addition is going to cost more than I expected!"

"No, Billy can't get a goldfish!"

"Turn off the lights!"

"You just ate! You want something else?"

The allocated fixed cost does not reflect the *true* marginal cost … but it can still influence perception

IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD

This chart is meant to be a somewhat humorous chart to illustrate the key concept of fixed cost allocation. Fixed costs are those that are not readily associated with variable (or "marginal," or "incremental") units of output. For example, a restaurant has the cost of the food used in a meal as well as a host of other costs such as wages, rent, utilities, etc.

If the restaurant charged just for the marginal cost of the food that goes into a meal, then they'd fairly quickly go out of business because those "other" costs -- the fixed costs -- would go uncovered. They need to charge enough per meal to get enough revenue to cover the cost of the food and the other costs.

Allocating fixed costs across some designated variable unit is a key part of business, and how it's done can influence perceptions and behavior. That's where this chart comes in.

Imagine a household with $100,000 in fixed costs per year. Imagine trying to allocate that $100K over some marginal unit. What marginal unit? The chart shows some potential ideas, and some potential behavior effects of that allocation.

Is this chart reality? Well, not really … not many households really try to allocate their fixed costs over some marginal unit. They just pay the bills. But business are different.

The Chargeback Conundrum

Chargeback can be a complex topic. But understanding how the "mainframe" costs are charged back is essential to the TCO question.

Mainframe support staff

Monthly License Charges

Environmental Costs
(Power and Cooling)

Building Overhead Costs
Such as rent or debt service, security, non-technical administrative staff, general lighting and HVAC, landscaping, property taxes, etc.

Often assessed per square foot

"Other" Overhead Costs
Corporate overhead burden that needs to be allocated somewhere. Typically flows out to operating units based on something like square footage or headcount or some other divisor.

==========

"Total Cost" of Mainframe

"Other" I/T Costs
Due to lack of good accounting tools on early distributed platforms, the cost of non-mainframe I/T -- including support staff -- is put into mainframe bucket. Practice never revisited … most of distributed server farm costs carried by mainframe

Is this fair? No.

Does it happen? Yes.

Why? Because it's an easy and convenient cost accounting bucket

==========

"Cost of the Mainframe" Number

÷ **Mainframe CPU Cycles Used**

==========

Cost per CPU Cycle To Use MF

The impression formed is …

"Cheap acquisition, free otherwise"

"High acquisition, high usage charge"

Neither is accurate. True TCO will account for costs attributable to the distributed server farms. Mainframe would bear only its fair share.

Challenge is that some costs are not clearly associated with one or other. How to fairly allocate? Cost accounting is not easy.

Here's the conundrum we sometimes face in the mainframe space. Mainframes are resources shared across many users. It costs money to acquire and maintain a mainframe so all those users can use it. To recover those costs, there is very often a practice of charging for the usage based on something like CPU time consumed. The ability to track how much CPU time has been used by what process has been around for a long time and it was a simple and convenient metric to use when chargeback policies were established originally.

So what costs should go into the "bucket" that is spread across the CPU time? The chart shows some of the common costs -- going from costs clearly associated with the mainframe itself (green boxes), to costs somewhat related to the mainframe (pale yellow), to other overhead costs that have to go somewhere (orange).

And then often we see something interesting happen. Distributed server box costs get lumped into the mainframe bucket as well. Why? Because it's an established chargeback mechanism and it's easier to put the distributed server farm costs there than to establish new monitoring and chargeback mechanisms. "It all comes from the same budget, right?" is the common refrain.

But such a practice influences perceptions quite a bit. With the "cost / CPU" of the mainframe pushed up, and users of mainframe services receiving charges for that usage, the response is often to move workload off the mainframe to avoid the charges.

This kind of chargeback practice does not accurately reflect the true cost of the mainframe operations. It unfairly penalizes the mainframe and unfairly benefits the distributed solutions

Does this happen everywhere? No. Some places? Yes.

The message here is that an accurate accounting of where fixed cost allocations actually go will more accurately reflect the true cost of computing solutions. That's at the heart of a good Total Cost of Ownership (TCO) study … what are the true costs?

Summary of the Business Value Benefits for Costing

Broad Guiding Principles of Business Value:

- **Available and Reliable**
  Addressed earlier in the presentation
- **Manageable**
  Addressed earlier in the presentation
- **Flexible**
  Operational flexibility addressed earlier in the presentation

  SMF allows for usage details down to the request, as opposed to costing based on server box
- **Affordable**
  TCO studies have shown the mainframe to be an affordable solution when costs are properly taken into account

Business values.

Overall Summary

Overall Presentation Summary

- The System z and z/OS platform has technical qualities that are part of its design
- Those technical qualities can map to your business objectives
- Two ways to derive the benefits:
  1. Passively -- simply by running on the platform
  2. Actively -- by directly making use of the features
- WebSphere Application Server for z/OS is designed to actively exploit the value features of the System z and z/OS platform
- WebSphere Application Server for z/OS is capable of extremely high degrees of scalability and availability if that's a business objective
- The administrative skills to run WAS z/OS are in many ways common with WAS on other platforms.
- The cost of WAS z/OS is often cited as an inhibitor. We ask that the question of cost allocation be looked at objectively.

Our concluding points.

We hope this presentation has provided you with a sense of the overall value statements of the platform, and thus the values that accrue up to the applications running in the JVMs of WAS z/OS.

End of Document