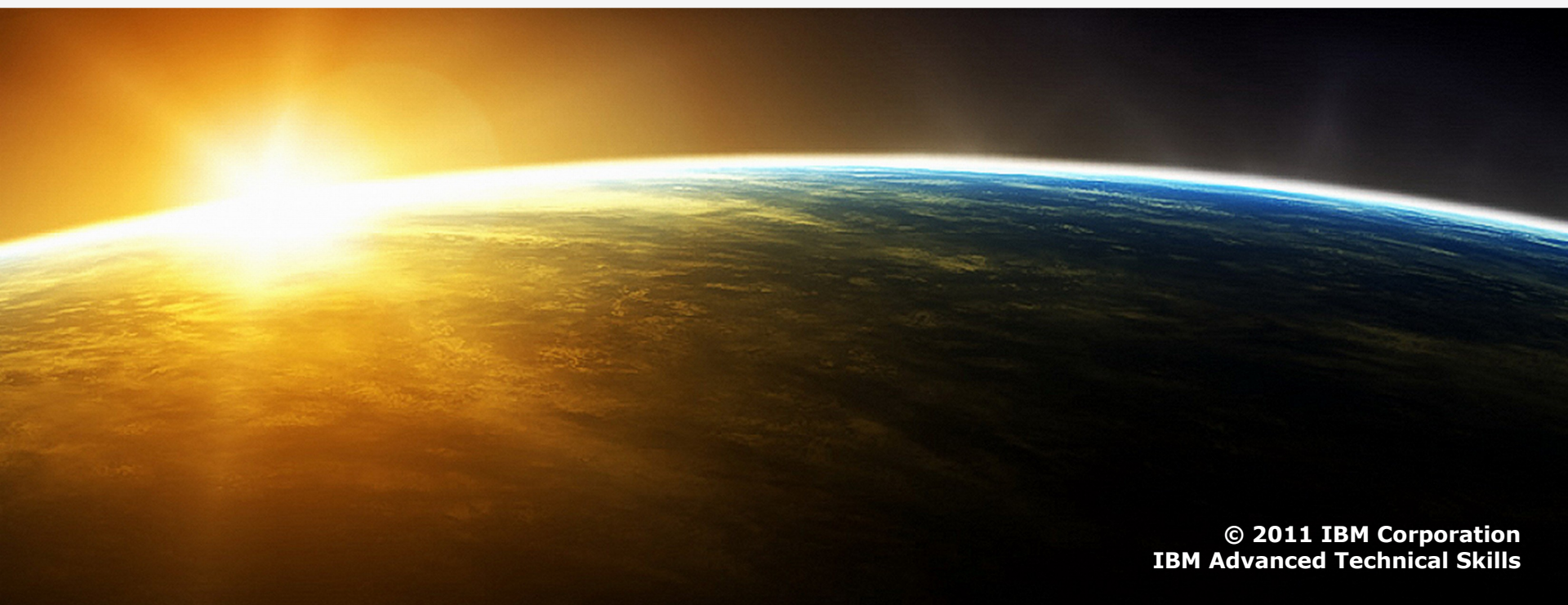






WebSphere Application Server z/OS Version 8

# Technical Overview

A companion piece to video series on YouTube



# Companion Videos on YouTube

<input type="checkbox"/>		<p> <a href="#">WASZOSV8 - Intro to WAS V8 - Part 1 of 4</a> <span>HD</span>            April 2, 2011 02:15 PM   This sets the stage for WebSphere Application Server V8 by str...  <a href="#">Edit</a> <a href="#">Insight</a> ▼         </p>	<p><b>4m:50s</b></p>
<input type="checkbox"/>		<p> <a href="#">WASZOSV8 - Intro to WAS V8 - Part 2 of 4</a> <span>HD</span>            April 2, 2011 02:16 PM   In this part we review the updated open standards and the roll...  <a href="#">Edit</a> <a href="#">Insight</a> ▼         </p>	<p><b>6m:30s</b></p>
<input type="checkbox"/>		<p> <a href="#">WASZOSV8 - Intro to WAS V8 - Part 3 of 4</a> <span>HD</span>            April 2, 2011 02:18 PM   In part 3 we look at two functions added to WAS V8 z/OS that a...  <a href="#">Edit</a> <a href="#">Insight</a> ▼         </p>	<p><b>8m:54s</b></p>
<input type="checkbox"/>		<p> <a href="#">WASZOSV8 - Intro to WAS V8 - Part 4 of 4</a> <span>HD</span>            April 2, 2011 02:13 PM   In this final section we look at z/OS-specific functionality -...  <a href="#">Edit</a> <a href="#">Insight</a> ▼         </p>	<p><b>18m:00s</b></p>

**Search on key string WASZOSV8 to find these videos**

# Outline of Topics

- **Update of Open Standards**

A list of open standards that have been updated by the standards community and incorporated into WebSphere Application Server V8

- **Updated JVM inside Java**

Still “Java 6” but the JVM has been updated to provide better performance and be aware of the z196 processor

- **Use of IBM Installation Manager**

Changes the initial install of base code but the construction of the runtime remains the same

- **High Performance Extensible Logging (HPEL)**

Optional binary logging function

- **Data Resource Routing**

Provides ability to detect loss of local DB2 (or WOLA address space) and route to alternative connection factory definition

- **More Granular Control of RAS Functions**

Provides ability to dynamically specify timeout and tracing functions down to the individual request level

# Updated Standards



# New Version = Updated/New Specs

Best place to understand the new specs is to go to the InfoCenter and search on `rovr_specs`

[Network Deployment \(z/OS\), Version 8.0](#) > [Reference](#) > [Supported configurations and limitations](#)

## Specifications and API documentation

The WebSphere® Application Server product supports various ir specifications and application programming interface (API) docu previous product releases.

### Components

- [Any application type](#)
- [Web applications](#)
- [Portlet applications](#)
- [SIP applications](#)
- [EJB applications](#)
- [Client applications](#)
- [Web services](#)
- [Service Component Architecture](#)
- [Service integration](#)
- [Data access resources](#)
- [Messaging resources](#)
- [Mail, URLs, and other Java EE resources](#)
- [Security](#)
- [Web Services Security](#)
- [Naming and directory](#)
- [Object Request Broker](#)
- [Transactions](#)
- [WebSphere extensions](#)
- [Administration](#)

#### Any application type

Table 1. Supported specifications for any application type. The product supports the specifications or APIs in this table.

Specification or API	Version 8.0	Version 7.0	Version 6.1	Version 6.0
Java™ Platform, Enterprise Edition (Java EE) specification	<a href="#">Java EE 6 (JSR 316)</a> New	<a href="#">Java EE 5</a> New	<a href="#">J2EE 1.4</a>	<a href="#">J2EE 1.4</a> New
Prior to Java EE 5, the specification name was Java 2 Platform, Enterprise Edition (J2EE).	<a href="#">Java EE 5</a>	<a href="#">J2EE 1.4</a>	<a href="#">J2EE 1.3</a>	<a href="#">J2EE 1.3</a>
	<a href="#">J2EE 1.4</a>	<a href="#">J2EE 1.3</a>	<a href="#">J2EE 1.2</a>	<a href="#">J2EE 1.2</a>
	<a href="#">J2EE 1.3</a>			
Java Platform, Standard Edition (Java SE) specification	<a href="#">Java SE 6</a>	<a href="#">Java SE 6</a> New	<a href="#">J2SE 5</a>	<a href="#">J2SE 1.4.2</a>
Prior to Java SE 6, the specification name was Java 2 Platform, Standard Edition (J2SE).				
ISO 8859 specifications	<a href="#">ISO 8859</a> applies to these versions.			

## Java EE 6 (JSR 316)

Continues trend towards increased function and a simpler development model

## Java Servlet 3.0 (JSR 315)

Enhancements to support modern web development

## EJB 3.1 (JSR 318)

Further simplifies development of EJBs

## JCA 1.6 (JSR 322)

Update specification architecture based on feedback from experts and users

# New JVM Inside Java

# Java, JVM, and Platform-Awareness

Version 8 still provides Java 6 (though called "6.0.1"). What's new is the JVM inside the supplied Java, and the z196-awareness:

**Application**

Application still "sees" Java 6, but benefits from the performance enhancements below

## Java 6 Specification Definition

**Java Virtual Machine**

**z-Aware Native Layer**

- Significant enhancements to JIT optimization technology
- New Balanced GC policy to reduce max pause times
- z196 exploitation of instructions and new Out-of-Order-Execution pipeline



z196 processor has faster chips, larger and better cache, an Out-of-Order-Execution pipeline and additional instructions

With WebSphere Application Server for z/OS V8 comes a new Java Virtual Machine inside the delivered Java runtime. It is still considered "Java 6" -- *there is no change in the application layer interfaces* -- but the JVM under the covers has been enhanced. That brings better performance to the applications running above the Java 6 specification line.

With WAS z/OS V8 comes a mixture of things that contribute to performance benefits:

- The benefits associated with the new z196 processor, which as a faster clock speed and an improved cache structure. Those benefits appear regardless of the level of WAS running above.
- The benefits associated with the new JVM. Most of those benefits do *not* require a new z196; the benefits appear even on a z10. They are simply benefits of more efficient execution within the JVM's Just-In-Time (JIT) compiler and the new garbage collection (GC) mechanisms.
- The specific benefits seen by the new JVM exploiting the new instructions in the z196 processor chip. For this category of benefits you'd need both WAS V8 and a z196.
- The benefits associated with general improvements in WAS z/OS V8 itself. These are improvements independent of the JVM or the processor.

For this document we are deliberately avoiding offering specific performance numbers. In time publications will appear that provide the specifics.

The message here is this -- there *are* performance benefits to be had, and how *much* you realize is a function of the things outlined above.



# IBM Installation Manager

# Background on IBM Installation Manager



**IBM Installation Manager is a product intended to install software, update software, and keep track of levels installed.**



**Introduced in 2006, it now has over 120 IBM products using it**

## **Things it can do:**

- **Install software**
- **Update with fix packs**
- **Modify features and functions**
- **Rollback features and functions**
- **Uninstall software**

**Most commonly thought of as a workstation tool, it *does* have a z/OS command line component**

**For WebSphere Application Server z/OS V8, IM is used for two things:**

1. **Creating/maintaining the hlq.SBBOHFS file system on z/OS**
2. **Installing/maintaining the WCT V8 tool on your workstation**

The next topic we will cover is that of the IBM Installation Manager, or "IM" for short. Starting with WAS z/OS V8 it will be part of the installation story. We have several charts coming up that explain how IM is used. First, let's step back and give a bit of background on what Installation Manager is.

Many of us are familiar with installing software on our workstations. Often the installation mechanism is something as simple as an executable file that we double-click on and it launches a wizard which installs the software. That works fine for relatively simple software packages. Complications arise when you go to upgrade that software, or if the software has extensions or features that can be installed or uninstalled, upgraded or not, as time goes on.

In 2006 IBM released Installation Manager as a way of better managing the installation of software. On the surface some may *think* it's just an installation shell. And while it does perform that function, it also keeps track of the software it has installed (one package, many packages), and is used to update, upgrade and modify the features of installed packages. One workstations the Installation Manager is provided as a graphical user interface (GUI) function. The bitmap on the chart is an actual clip of an IM panel from a Windows machine. There is also a command-line interface which allows IM to be used on systems where a GUI is no present (z/OS) or may not be configured.

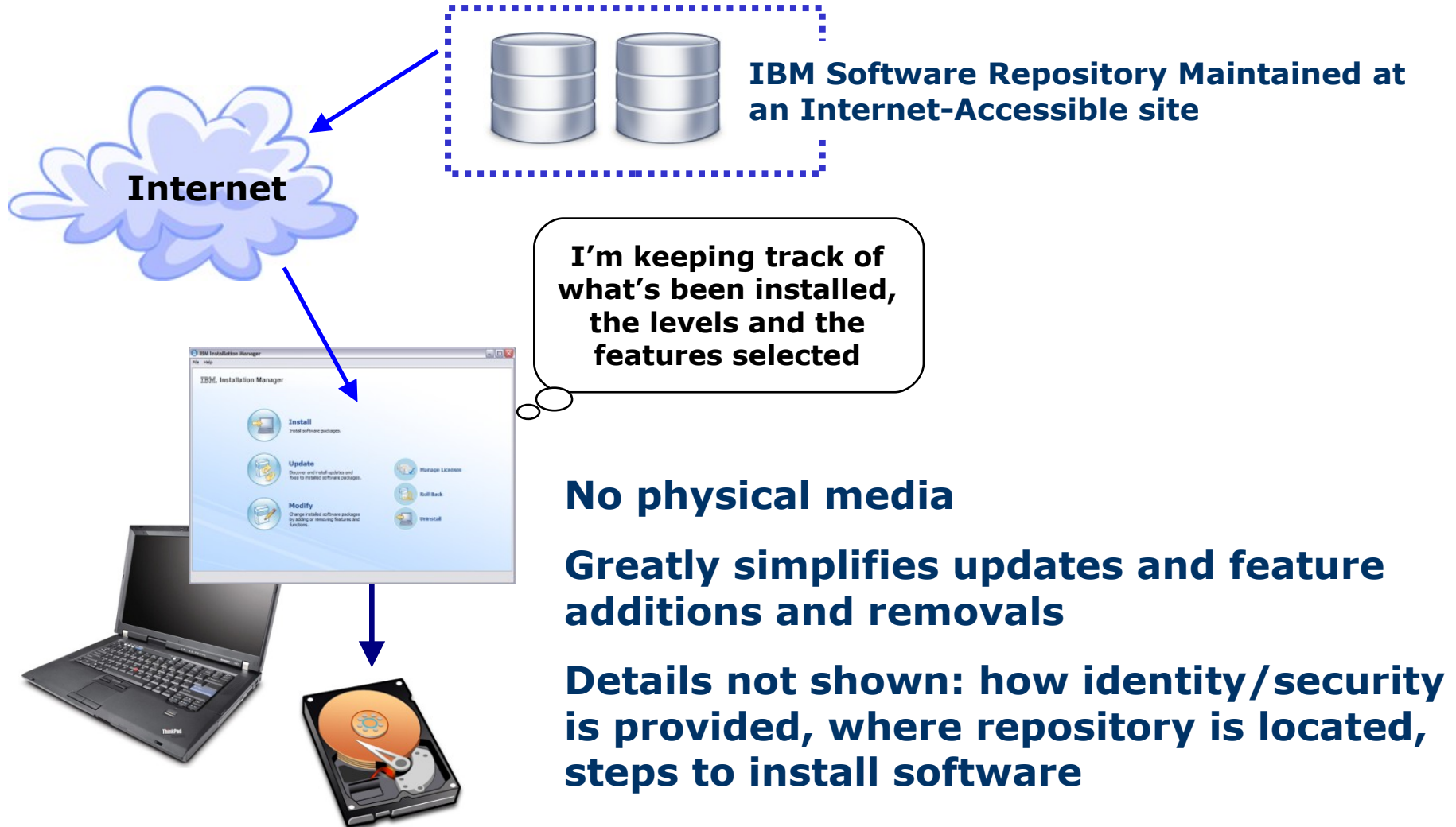
Starting with WAS z/OS V8 the IBM Installation Manager will become part of the installation process for the product. It is used in two ways --

1. Installation Manager on the **workstation** to install and manage the WebSphere Customization Tools (WCT) Version 8 product. In the past this was installed as a simple install shell executable, but starting with V8 it's packaged for IM, and IM manages the initial install, subsequent updates, as well as managing the installation of the various "extensions" needed when you use WCT to configure Feature Packs and migrations.
2. Installation Manager on the **z/OS system** to create the product UNIX System Services (USS) file system data set that holds the product files. This is the `hlq.SBBOHFS` data set for those familiar with the physical infrastructure of WAS on z/OS.

We have much to discuss and explain. The next few charts set about to do just that.

# Key Concept: IM “Repository”

Think of a repository as the place where Installation Manager goes to get the software files and any updates to installed software:



We start by introducing a central concept of Installation Manager -- the "Repository." You may think of a repository as the source for product files when IM goes to install a piece of software. It's really just a file tree -- directories and files with lots of XML to describe everything.

Repositories may reside "locally" or "remotely," and if remotely then they are accessible over the Internet as a URL. That URL points to a hosting site at IBM (Boulder, CO) where IBM maintains the repository and makes sure all the latest updates are available in the repository. Then when an individual user installs a piece of software on their workstation, their local copy of Installation Manager reaches out across the network to the IBM-hosted repository and pulls the files necessary to install the requested product and its features.

All the while the local copy of the IM is keeping track of what's being request, what's being installed, and what has been installed. That's an important element of the IM story because when you go to update a piece of software using IM, IM already understands what's presently in place. And it calculates the delta files needed. It pulls those delta files from the repository. It installs those files into the copy of the software. It manages all the pre-req and co-req relationships. In short, it does all the hard thinking and heavy lifting. It makes thing *much* easier.

As the chart indicates, there's a great deal of detail we're omitting at this time ... in particular details about how IM provides its identity to the repository. It's an interesting topic, just beyond the scope of *this* document.

That's the quick story of the repository. Keep this in mind as we unfold the other elements of this story.



# IM and Workstation

The “WebSphere Customization Tool” (WCT) is what’s used to create the customized jobs that build your runtime. IM installs that for WCT V8:



**WebSphere  
Customization Tools V8**

## Process:

- **Install IM on your workstation (no-charge software; download, double-click and take most defaults)**
- **Point IM to IBM repository where WCT V8 is available (WCT is no-charge as well)**
- **Use WCT much like you did for Version 7:**
  - **Plan runtime with spreadsheet (updated for V8)**
  - **Generate customized jobs and upload to z/OS system**
  - **Submit execute jobs**
  - **Start the runtime**
- **Use IM to updated WCT with any extensions for feature packs or stack products**

We'll first discuss the role of IM on your workstation to manage the installation of WCT V8. This is a relatively simple thing to understand and do, so by discussing this first we get it off the table so we can focus more carefully on the topic of Installation Manager on z/OS, which comes next. For now, let's focus on IM on the workstation.

WebSphere Application Server for z/OS V8 requires a set of customized jobs to build the runtime environment. Those customized jobs are created by the WebSphere Customization Tools product, or WCT for short. It's been that way since WAS z/OS V6. It continues to be the case into WAS z/OS V8.

WAS z/OS V8 requires a new level of the WCT to produce the customized jobs used to build a V8 runtime. That new level of the WCT is, not coincidentally, WCT V8. So to create the customized jobs used to build a WAS z/OS V8 runtime you need WCT V8.

To install WCT V8 on your workstation you'll now also need Installation Manager. You may already have IM on your workstation if you've installed some other product that used it ... for example, Rational Application Developer. But if you don't have IM, then you must first install it and then using that install the WCT. Details on how to acquire the installation package for IM and where the WCT V8 repository resides is provided in the InfoCenter.

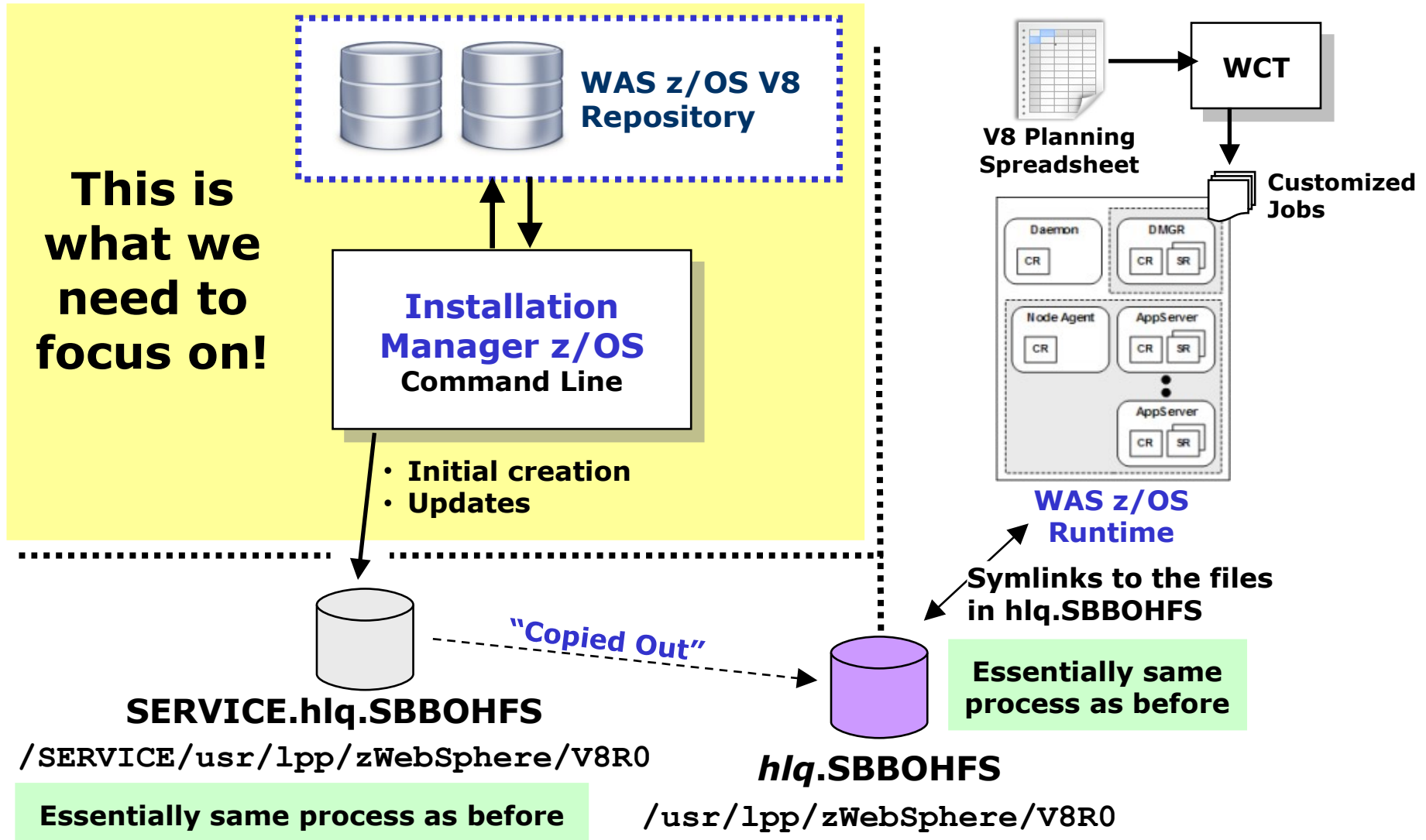
Installing IM is relatively simple -- a few mouse clicks and with defaults taken IM is installed on your workstation. Using IM to install WCT is already relatively simple -- point the copy of IM to the IBM repository, select the WCT V8 product and IM does the rest. Once WCT V8 is installed then using it to create the customized jobs is very similar to how it's been in V7.

Okay ... that's IM and WCT on the workstation. A necessary process to get WCT V8 in place so you may create the customized jobs to build the WAS z/OS V8 runtime.

Clear your mind of the workstation. Let's now focus on z/OS.

# IM and z/OS

This is entirely new for WAS z/OS V8 ... the use of command line IM on z/OS to create and maintain the hlq.SBBOHFS file product file system:



When it comes to Installation Manager on z/OS we start the story by focusing on the purple disk icon labeled "hlq.SBBOHFS" ... for that is the object of this discussion: the creation of that file system with all the piece-parts that make up the WAS z/OS V8 product. The WAS z/OS V8 runtime you build and operate will point to the files in the hlq.SBBOHFS file system with symlinks coming out of the configuration file systems.

Pause and consider that ... see the vertical line on the right side of this picture? Everything to the right of that line is pretty much the same in V8 as it was in V7 in terms of structure and concepts. Sure, the WAS product itself is newer. But the basic concept of having the runtime components access a read-only file system to get the product binaries is the same in V8 as it was in V7.

Let's now move to the left side of the picture and see how Installation Manager works into it.

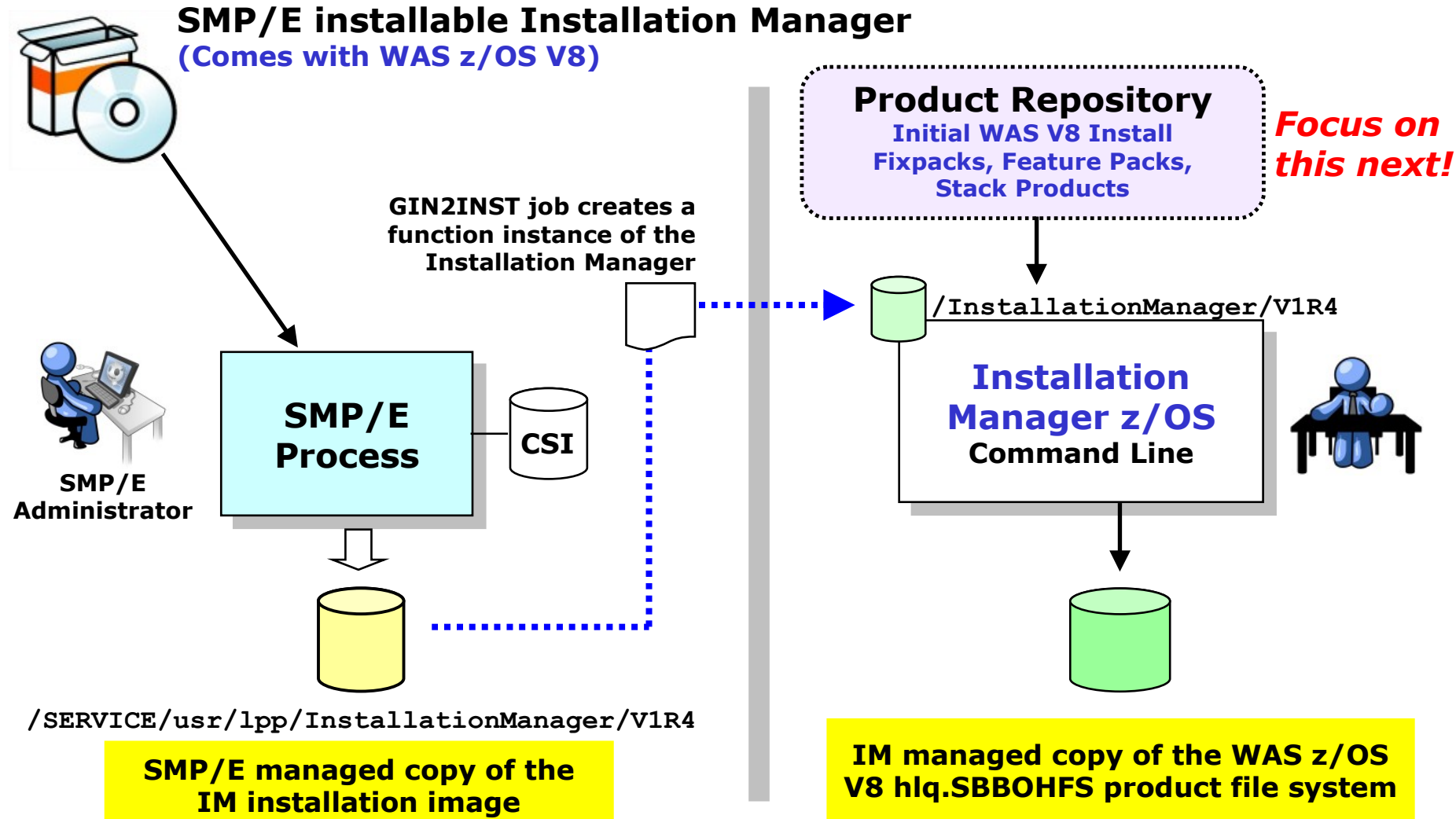
For operations on z/OS, Installation Manager uses a command line interface. IM accesses the WAS V8 product repository and builds the file system that contains all the WAS z/OS V8 files. Now in most shops that is first built to a "service" location such as what's shown in the picture, then copied out as illustrated. But the main point remains -- Installation Manager is what creates that file system based on your commands as input to IM.

And later, when you apply fixpacks or feature packs, IM is used again to update the "service" copy of the SBBOHFS file system with whatever files and directories make up the updates. When IM is done updating the "service" copy, it's again "copied out" to the runtime environments.

So Installation Manager plays a key role in the *creation* of the SBBOHFS file system which contains the WAS z/OS V8 binaries. The upper-left portion of the picture is what we need to focus on because there are many questions we need to answer about that part of the picture.

# Installing the Installation Manager Itself

It's delivered as an SMP/E package. The result is a file system that contains the IM code. You then run simple job to create runtime instance:





# Notes

The first part of the story involves the installation of Installation Manager itself. IM on z/OS doesn't just magically appear ... it has to get there by some action on your part.

When you order WAS z/OS V8 you will get an SMP/E installable that is the Installation Manager function. You go through the normal SMP/E process to install this, and what results is a mounted file system that is not a function IM (yet), but rather an "image" of all the files needed to install an operational copy of IM on your system.

(In a sense this is like WAS z/OS itself -- the hlq.SBBOHFS file system is not *itself* an operational copy of WAS z/OS. It doesn't become a Deployment Manager or AppServer until you customize and submit the jobs to create the nodes and servers.)

The SMP/E jobs that come with the IM for z/OS include a job to install an operational copy of the Installation Manager. When that process is done, you have a mounted file system which has within it the "imcl" program -- and that stands for "Installation Manager Command Line." That's what you use to invoke IM itself to create the WAS z/OS V8 product file system. That's what you use to update the WAS z/OS V8 product file system. That's what you use to add feature packs to the WAS z/OS V8 product file system.

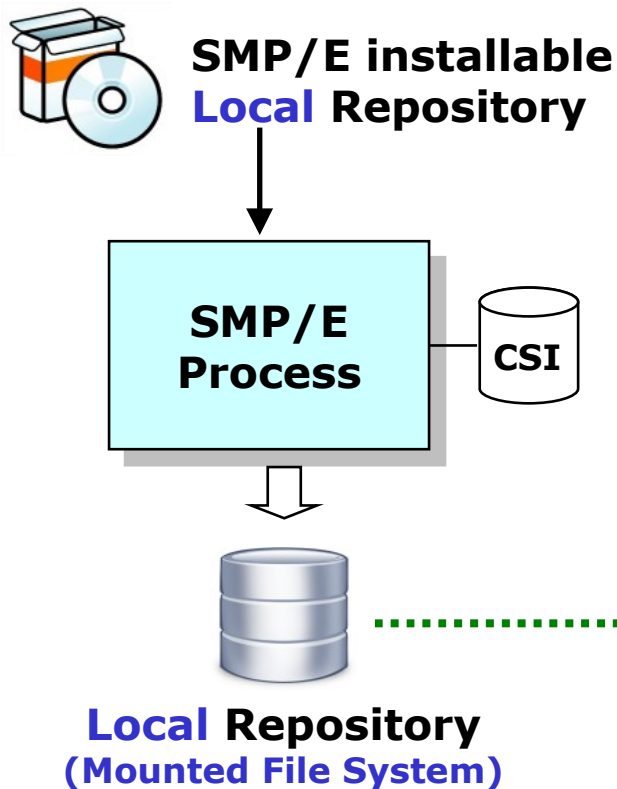
The installed copy of Installation Manager and the "imcl" command become what you use to create, manage and update the WAS z/OS V8 product on z/OS ... just like is done for products on the workstation.

The picture shows IM on z/OS accessing a "repository" as the source for the files needed to create and update the WAS z/OS V8 product file system. That becomes the next area of focus for us ... where is that repository?

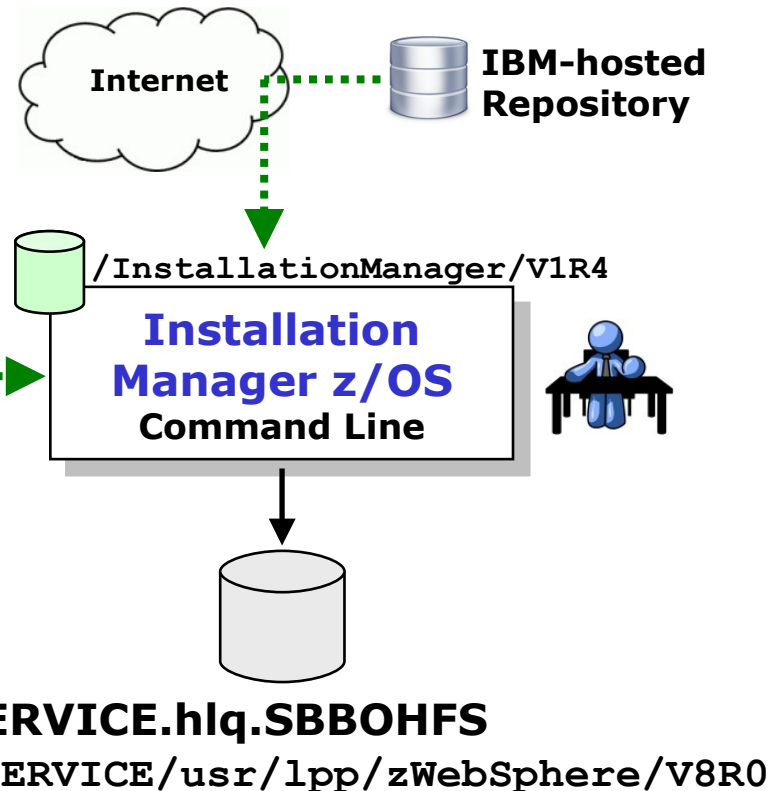
# The WAS V8 Product Repository

We have a two-part story here: one for the initial product delivery, then the opportunity to use IBM's hosted repository for fixpacks/feature packs

## Initial Acquisition of WAS z/OS V8 from IBM



## Fixpack and Feature Pack Updates



# Notes

Installation Manager allows the repository to be either a hosted "remote" repository (either hosted by IBM or hosted by your own company) or a "local" repository, which is really nothing more than a mounted file system. For WAS z/OS V8 the **initial** acquisition comes in the form of an SMP/E installable that lays down a local repository. That local repository has the WAS z/OS V8 files that IM uses to build the hlq.SBBOHFS read-only WAS product file system.

Thereafter you may use a remote file system for updates and feature packs, which in most cases will be the IBM-hosted repository. Your copy of the Installation Manager is keeping track of what levels of the code are in the target, which in this picture is the SERVICE.hlq.SBBOHFS file system. If you add a fixpack then IM will draw the binaries from the remote file system, include them in the mounted target file system, and update its own XML to keep track of what's in that managed copy of the code.

What about SMP/E ... does it just drop out of the picture? No, it may still play a role if you wish. Fixpacks and feature packs will still be delivered as SMP/E installables. Your SMP/E then becomes manager of the repository, with your copy of IM being the manager of the target WAS z/OS V8. Or you may simply keep the data in your SMP/E CSI up to date with UCLIN commands supplied by IBM. This will allow you to maintain SMP/E as a point for auditing and accounting while IM becomes the manager of the product itself.

# Summary of Installation Manager

- IBM's key software installation and update management product
- Used extensively by other IBM products (i.e. Rational tools)
- Now used for WAS z/OS V8
- SMP/E still plays a role for now:
  - Installation of IM itself
  - Installation of the initial local repository
- Fixpacks and updates may be then drawn from IBM hosted repository
  - IM generates UCLIN you may then apply to keep SMP/E information current
- Feature Packs binaries are added to WAS z/OS product file system, not hung off to side and symlinked to as it's done today
- Stack products may be installed through IM
- Add hot updates without need for ++APAR
- Back-out updates easily using IM

**IM is a powerful product and is the direction IBM is headed. Some learning curve involved but we are confident you will become proficient quickly and will come to understand the value of IM into the future**

# High Performance Extensible Logging (HPEL)



# Very High Level Overview of HPEL

This is an **optional** mechanism to format traces and logs into a WAS binary format. A utility is then used to offload to a viewable text file:

**Configurable on a  
Server-by-Server Basis**

**WebSphere  
Application Server  
Version 8**  
All Supported Platforms

Java Trace  
Java Logs  
System.out  
System.err

**More efficient use of space,  
faster write operations**



**WAS-specific  
binary format  
log file**

Write to memory buffer,  
then file

Controls to dictate size  
limits, what to do when  
limit reached, how to  
trim files, start new  
files, etc.

**Log Viewer  
Utility**



**Common across  
all platforms**

**Use whatever  
view/edit tool  
you prefer**



**ASCII  
readable  
file**

# Notes

One of the new functions in WAS V8 -- all platforms, actually -- is something called "High Performance Extensible Logging," or HPEL for short.

The first point to make about this is that it's an optional function -- you turn it on if you wish, and it's controllable down to the individual appserver level.

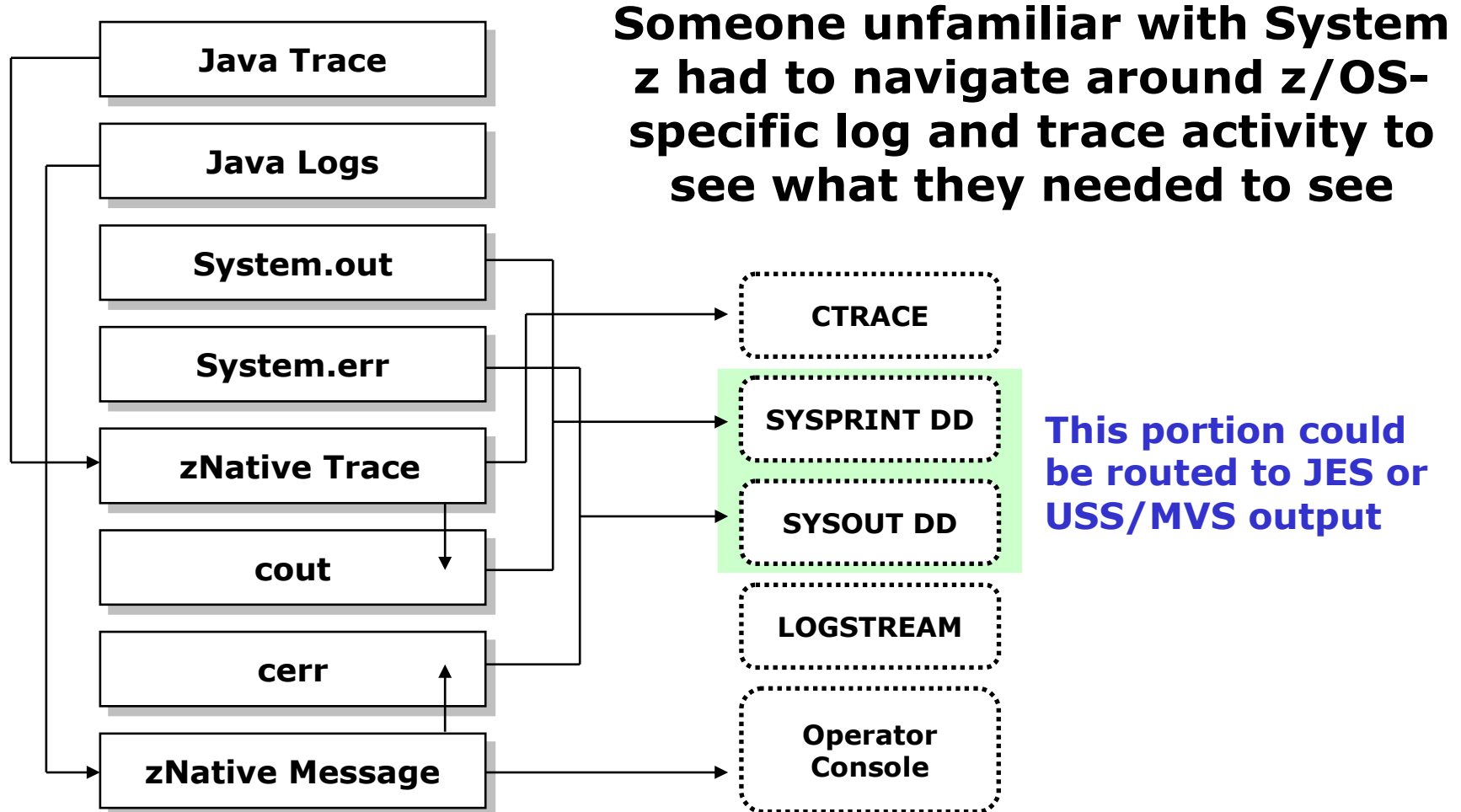
What's behind this new function? The logging and tracing function of WAS is a text-readable format that takes up a fair amount of disk space and incurs some overhead in writing. HPEL is written out in a binary format, which has far less "white space" and is more efficient to write. If you wish to view the contents of the file you use a supplied "Log Viewer" utility that copies the binary format out to a human-readable ASCII file.

Again ... this is *optional*, and it's configurable on a server by server basis.

In addition to being more efficient to write, it also has additional controls built into it which allows you to write to a memory buffer first, then roll that to disk; to control the maximum size of the file and what to do when that limit is reached.

# For z/OS: Output We Have Today

This is known as "Basic" in Version 8. On the next chart we'll see where HPEL affects the picture



# Notes

Let's focus on where traces and logs go using the existing logging function, which is now referred to as "Basic" as compared to "HPEL". The chart above is for z/OS.

So for example, by default pretty much all that eventually ends up in JES held spool.

Is that your desired outcome? JES spool? Then take no action in WAS V8 -- HPEL is optional.

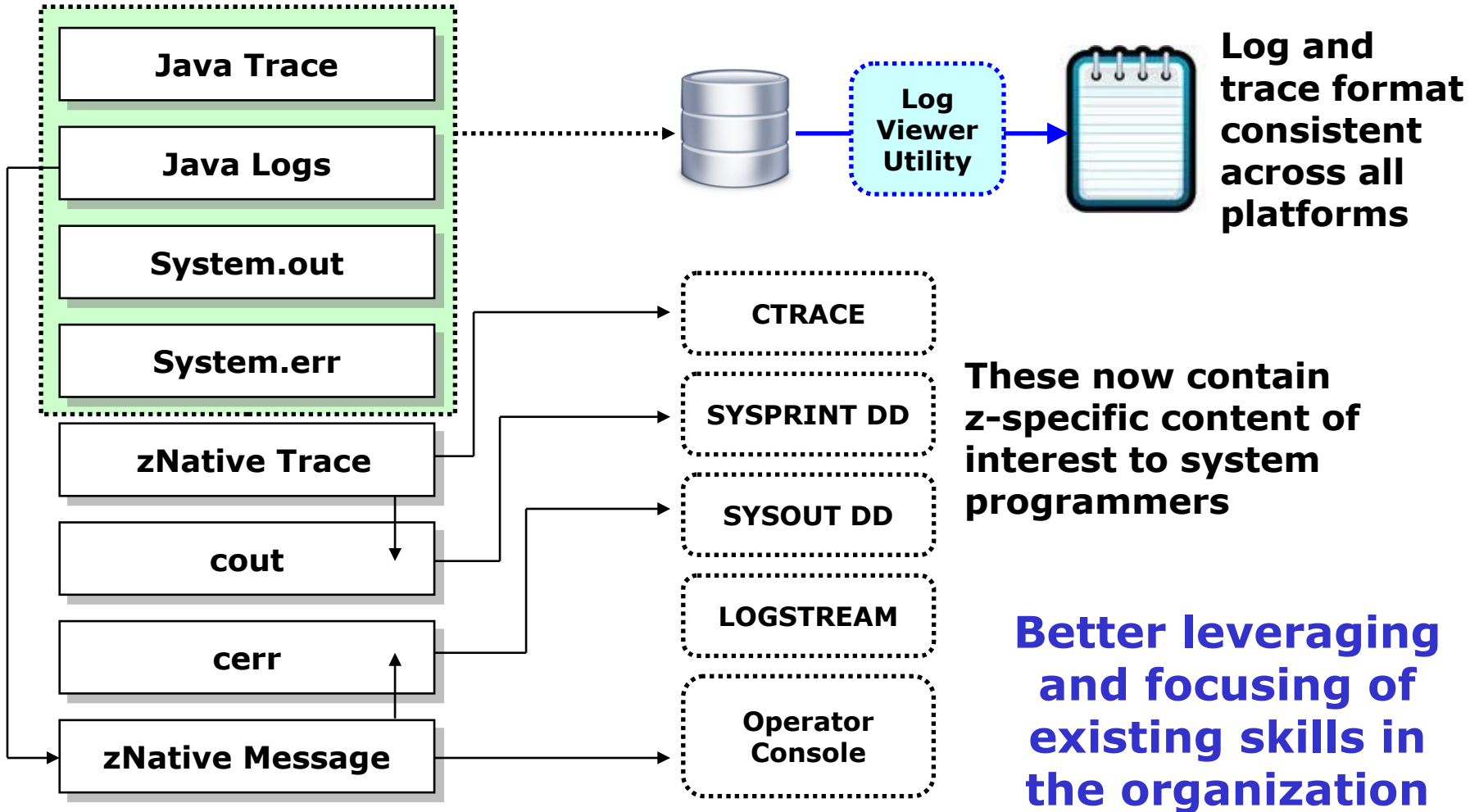
It is possible to redirect the SYSPRINT and SYSOUT DD to a USS or MVS file rather than JES.

In any event, a skilled WAS administrator on distributed who is unfamiliar with z/OS will have some difficulty navigating TSO, or navigating around the z/OS-specific things in the output file. One of the advantages of HPEL is that it provides a common and consistent log and trace format across all the platforms.

So that's how it is today. Let's now look at what happens with HPEL.

# For z/OS: With HPEL in Effect for Server

And here's what it looks like when you put HPEL into effect:





# Notes

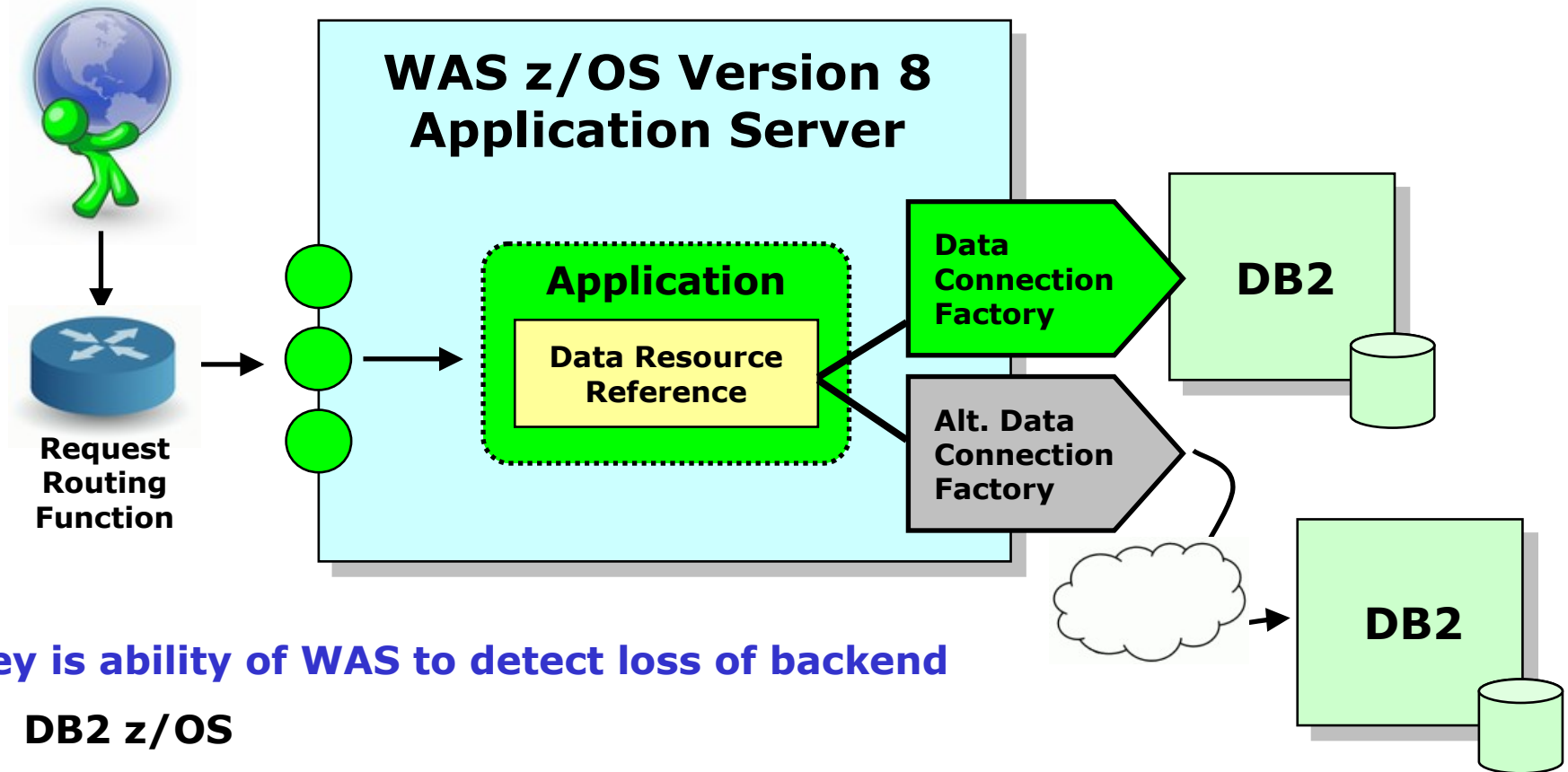
Here's the same picture, but with HPEL introduced. We see that the green box highlights the output that now would go to the HPEL log. z/OS-specific information would continue to flow where we expect it -- JES spool, WTO, etc.

This results in a logging format that is, as we mentioned, consistent across all platforms. A skilled distributed WAS administrator could be given Telnet access to the z/OS platform, and from there they could use the Log Viewer utility to output a file and review it. To them z/OS is no longer that "different" box to be avoided, but rather "just another UNIX server" that they will already be familiar with.

# Data Resource Routing

# High Level Overview of this Function

This function allows you to configure an alternative connection factory so in the event of a data resource loss the defined backup can be used:



**Key is ability of WAS to detect loss of backend**

- DB2 z/OS
- IMS if you're at a relatively recent level of maintenance
- Not yet CICS if using CTG functionality

# Notes

Let's now turn our attention to another function that is common to all platforms, yet as we'll soon see WAS z/OS adds an additional layer to it.

The background on this new function is this -- in the past applications with resource references would bind to a connection factory and that would be the path to the actual data resource. As long as that data resource is there, then data could be served. But if the data resource is lost there was no mechanism to route the application over to a different connection factory. This was a particular issue for "local" connectors ... lose the local data resource and the application was essentially without data until the resource came back.

With this new function we have the opportunity to define a secondary, or alternative, connection factory. If WAS detects the loss of the primary data resource, it automatically routes new connection requests from the application over the alternative connection factory.

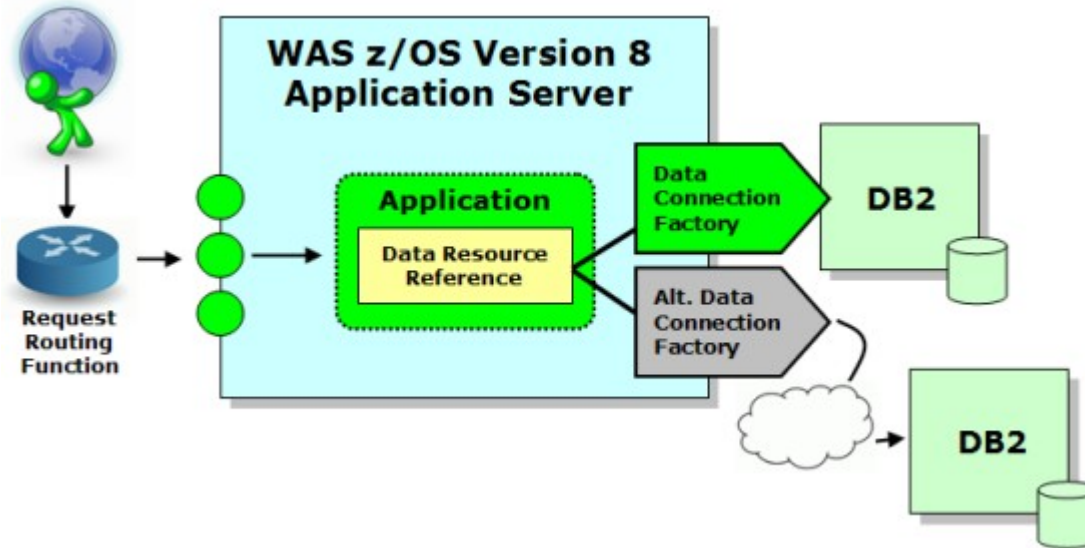
**Note:** the loss of a data resource behind a connection factory necessarily means the connections are lost. So this function does *not* mean connections are somehow auto-magically moved to the secondary data resource. Existing connections are lost and new ones have a place to go.

For this to work WAS must be able to detect the loss of the connected data resource. On z/OS it can do that today with DB2. It will do that with IMS provided you're at a relatively recent level of maintenance. For CICS the function is not yet available using the CTG function (even CTG in local mode using EXCI).

This function is simply yet another layer of availability.

# Further z/OS Exploitation

Some of what was shown on prior page is common across platforms. WAS z/OS V8 adds an additional layer:



**Imagine the backup DB2 is also lost, or you've not configured for an alternative connection factory. What then?**

## Three further configurable options:

1. Have WAS issue a Write to Operator (WTO) and use that to take further system automation action
2. Have WAS issue PAUSELISTENERS, thus turning off input ports to server, allowing front-end routers to detect and go elsewhere
3. Have WAS determine which applications are affected and stop just those applications, leaving other applications up and running



# Notes

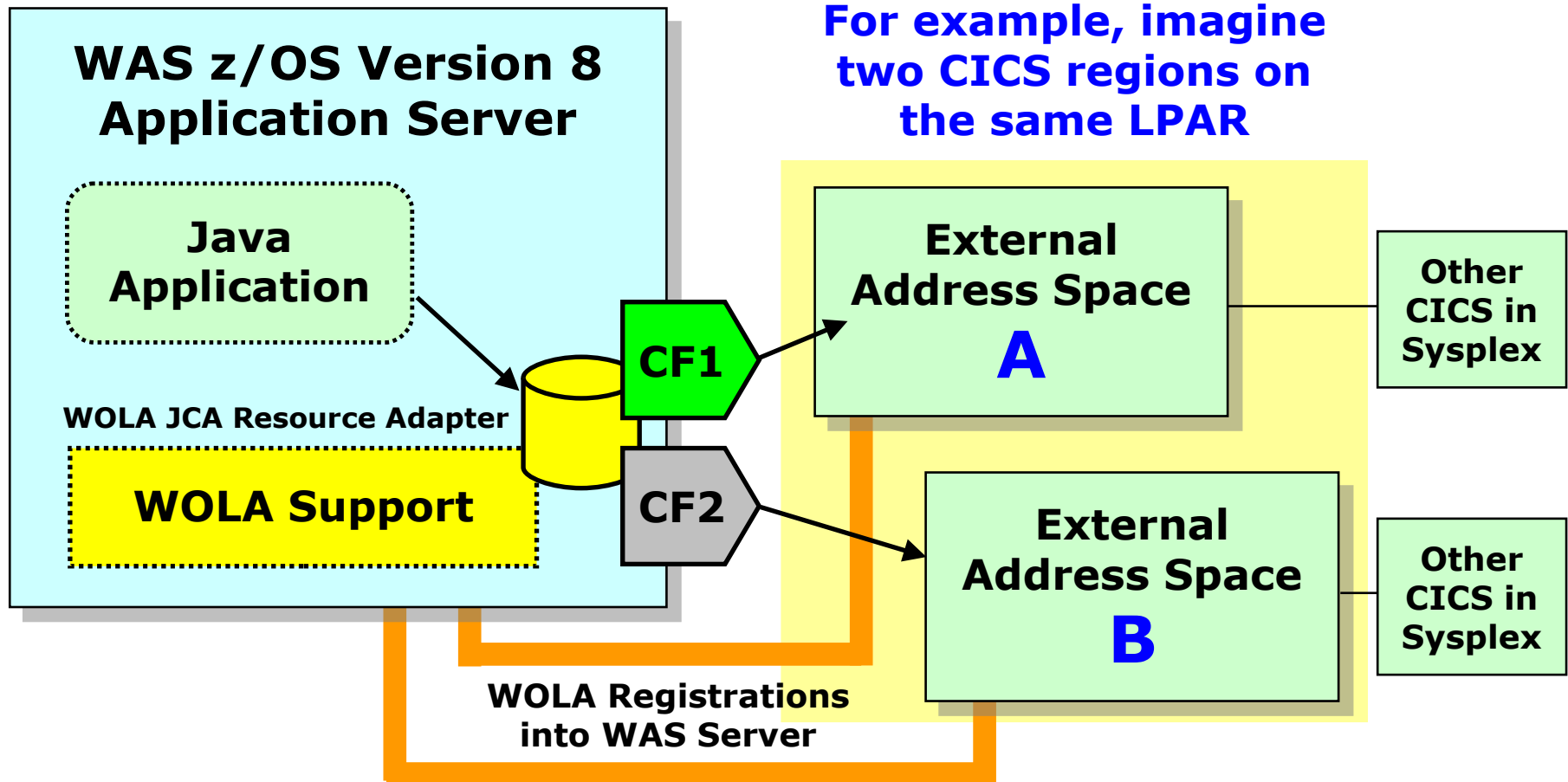
WAS z/OS V8 takes this one step further. The function we'll outline here is for the case where you don't have a secondary (or "alternative") data resource, or worse ... both primary and secondary are gone.

WAS z/OS V8 then allows you to configure one of the three things shown on the chart to take place.

The distinction between #2 and #3 is that some front-end routing mechanisms are capable of seeing the "server up but application down" case and routing around it. For example, the WAS Proxy server is capable of detecting whether an application is present or not. The WAS HTTP Server Plugin is not. Nor is Sysplex Distributor. So if you have as a front-end device a mechanism that can detect an application outage, then stopping just that application gives a more granular control over the reaction to the loss of a data resource.

# WOLA Variation on This New Function

WOLA participates in this as well in that a backup registered external address space now be used in the event the primary is lost:



**WOLA is by definition "same LPAR," and this gives you a degree of availability by allowing routing to secondary registered external address sapce**

# Notes

WOLA participates in this as well. The scenario on the chart shows the customer requirement that came up that led to WOLA's change in design.

Imagine you have two CICS regions on the same LPAR, and both are CICS routing regions in a CICSplex environment.

What this new function allows you to do is have both CICS regions register into the WAS server using WOLA, and provide WOLA with two connection factory definitions -- a primary and a backup. If the primary CICS region goes down, WAS detects this and routes new connection requests over the registration to the secondary CICS region. WOLA works with CICS because the loss of the CICS region implies the loss of the registration, and that's what tells WAS to switch over to the backup. (The CTG function operates differently, and at present there's a challenge in detecting whether a backing CICS region has really gone away.)

If the objective is to use WOLA to get from WAS into a CICS region where some of the processing is done cross-Plex to other CICS regions, then this new function solves an availability problem. Before the loss of the single registered CICS region mean the entire CICSplex was lost to the WOLA processing in WAS. No longer ... now it has a way to get to a secondary CICS region and back into the CICSplex.

# Granular RAS

# RAS Function Control Down to Request

This new function leverages existing classification file to extend scope of certain RAS functions down to the request level:

**Not just HTTP ... any classifiable input type (IIOP, MDB, etc.)**

```
<http_classification_info transaction_class="_____"
```

```
dispatch_timeout="_____"
queue_timeout_percent="_____"
request_timeout="_____"
stalled_thread_dump_action="_____"
cputimeused_limit="_____"
cputimeused_dump_action="_____"
dpm_interval="_____"
dpm_dump_action="_____"
SMF_request_activity_enabled="_____"
SMF_request_activity_timestamps="_____"
SMF_request_activity_security="_____"
SMF_request_activity_CPU_detail="_____"
classification_only_trace="_____"
message_tag="_____"
timeout_recovery="_____">
```

**Additional XML values permitted in the classification file**

**These control behavior when classification identification is made by this function**

**MODIFY command allows you to dynamically enable a new file, or dynamically revert back to previous**

**Enhances granularity. Previously down to server level; now to request.**



# Notes

The final function we'll discuss is a way to increase the granularity of behavior based on requests. This takes the control from at best server level in V7 to specific request level in V8.

This function piggybacks on the XML classification file mechanism that's in WAS z/OS V7 today. That XML file is what's used to identify inbound requests for WLM classification. This new function extends the values allowable in the XML to influence RAS behavior once a request has been identified.

So for example we're showing a snippet of the classification XML in the chart above. What the yellow highlighting indicates is the additional XML now permitted to control behavior for requests that match the request type (HTTP, IIOP, MDB, etc.) and the filter for the specific request.

For example, you may now set timeouts to the individual application request level.

Or set the maximum CPU allowed based on an individual request type.

Or set SMF 120.9 recording activity based on the request.

You may also specified a character tag to be appended to messages so your system automation routines may act upon those tags.

Further, this classification XML file may be dynamically enabled with a MODIFY command. So if you with to change the behavior of something you would change the XML, issue the MODIFY command and that XML is then in effect. You may issue a different MODIFY command to revert back to the previous XML file if your new XML file introduces behavior you didn't intend.

# Summary

# **WAS z/OS Version 8 ...**

- **Continued improvement and refinement of the proven base of WAS**
- **Skills honed with WAS V6.1 or V7 apply equally well to V8**
- **Enhanced commonality with other platforms – use of IM and HPEL**
- **Enhanced availability with data resource routing**
- **Enhanced control with granular RAS**