WBSR85

WebSphere Application Server z/OS V8.5

Unit 3 - Server Models

TechDocs WP101740, WP102110

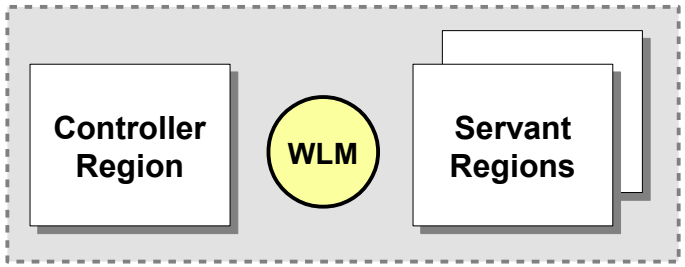
This page intentionally left blank

Overview

With WAS z/OS V8.5 we now have two server models to choose from:

Traditional Multi-JVM Model

"Application Server"



- Two or more JVMs make up an application server instance
- CR does the request handling, SR hosts the applications
- Full Java EE server runtime
- Administration through DMGR and Admin Console as seen in Unit 2
- Includes "Granular RAS" function which we'll explore in this unit

Liberty Profile Model

"Application Server"

- One JVM makes up an application server instance
- Lightweight, composable and dynamic updates
- Web applications at this time
- Simple configuration and administrative model
- Not part of the traditional WAS cell or administrative model

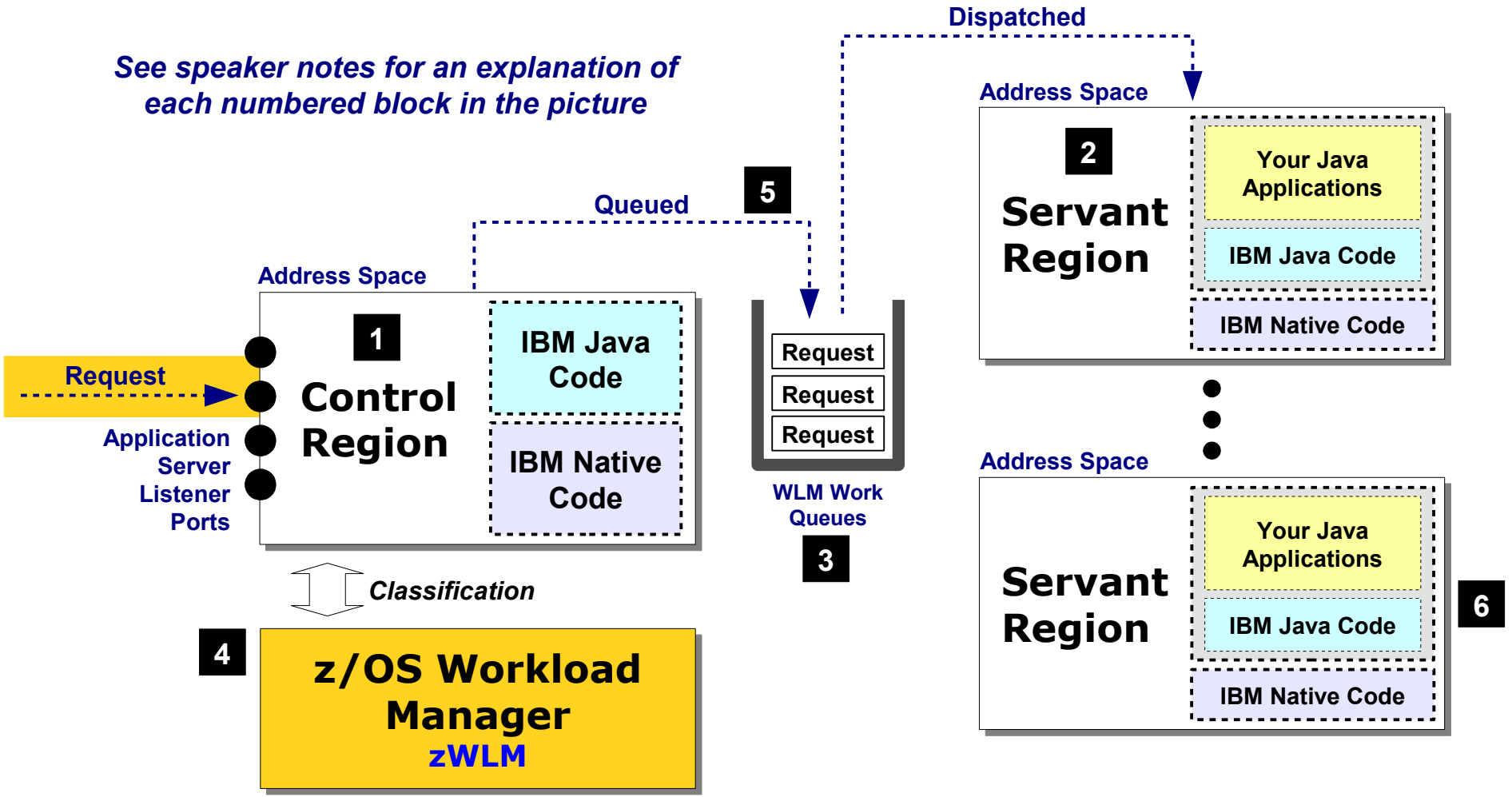
Traditional WAS z/OS model ...

Traditional WAS z/OS

The full Java EE, multiple-JVM model

The Essential Structure of WAS z/OS "AppServer"

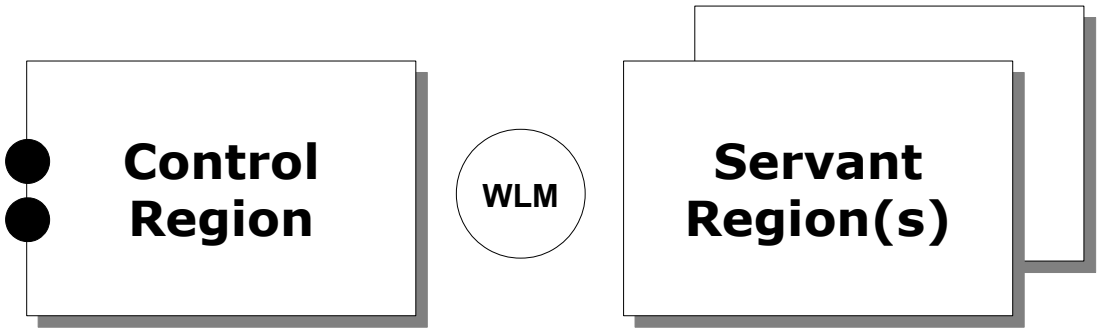
Earlier we spoke of an application server consisting of a "Control Region" and one or more "Servant Regions"



Control Region Adjunct …

Detour: the "Control Region Adjunct"

People familiar with WAS z/OS often ask: "What's the Adjunct Region used for?"



Control Region Adjunct

Like servants, this is started automatically by WLM

Unlike servants, there is either 0 or 1 of these

Region is related to messaging:

- **If "Service Integration Bus" (SIBus) configured and server is hosting a "messaging engine" on the bus**
- **If server has "Activation Spec" defined and used by deployed application**
  Which may listen on MQ or SIBus, so CRA is not related to *just* SIBus work

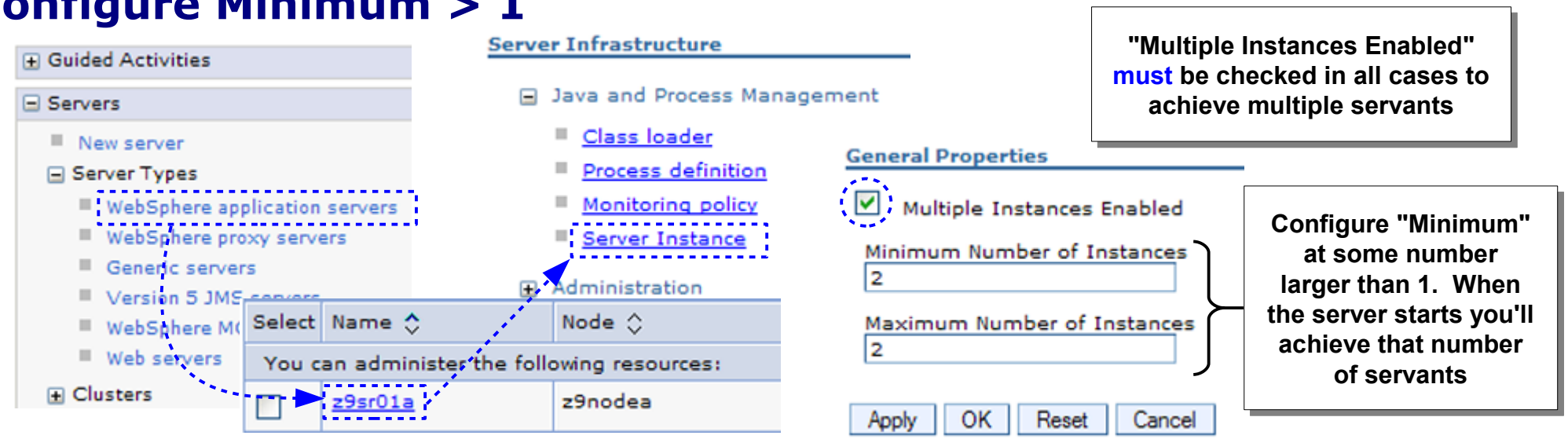
We won't focus on the CRA in this workshop. This chart is included just to let you know what it's used for and under what circumstances you would see it start

Three ways to get multiple servants …

Three Ways to Achieve Multiple Servant Regions

There are three ways to achieve more than one servant region for a given appserver:

Configure Minimum > 1



Use MODIFY to dynamically change minimum

```
F Z9SR01A,WLM_MIN_MAX=3,3
```

Controller JOBNAME

MIN,MAX

- "Multiple Instances Enabled" must already be checked
- If MODIFY minimum > configured minimum then servant is started
- If MODIFY minimum < configured minimum then excess servants will be stopped when all work flushed

Let WLM dynamically start more servants

Based on volume of work

Based on WLM Service Class assignments

z/OS and starting servant …

Servant Started by WLM Based on WAS Request

This chart summarizes the relationship between the `START` command for the CR and how the SR is started automatically:

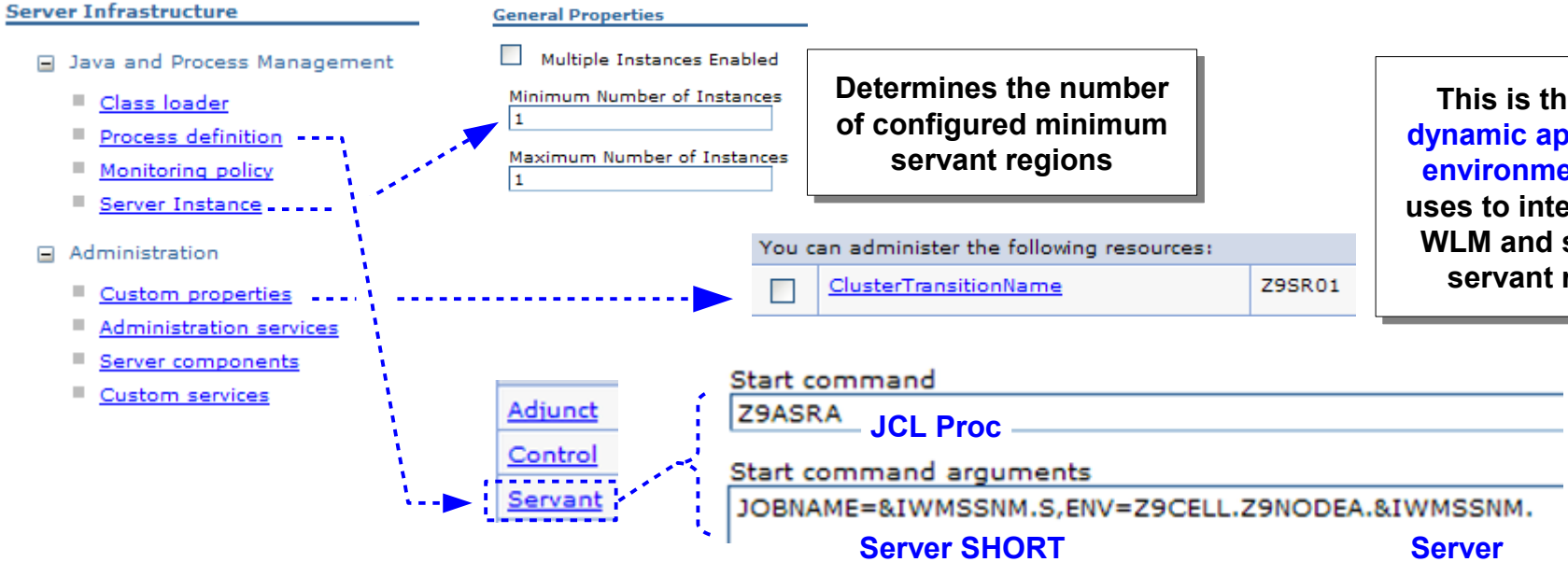
You issue control region `START` from MVS console

Or Start Server from Admin Console

JCL Proc | We recommend this be equal to server SHORT | Cell SHORT | Node SHORT | Server SHORT

```
S Z9ACRA,JOBNAME=Z9SR01A,ENV=Z9CELL.Z9NODEA.Z9SR01A
```

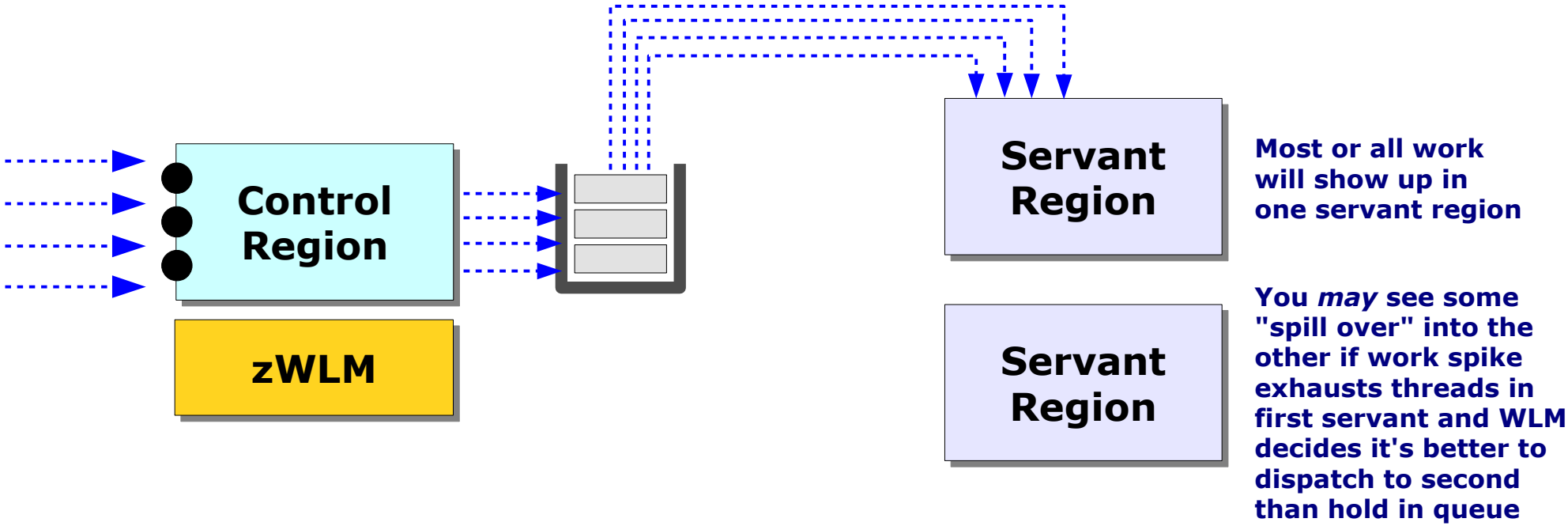
WAS z/OS then works with WLM to start configured servants



Work distribution …

In General, WLM will Favor One Servant

This is behavior many *don't* expect ... when multiple servants are present the work tends to end up in one but not the other:



WLM's default is to take "first available" and once chosen, stick with it

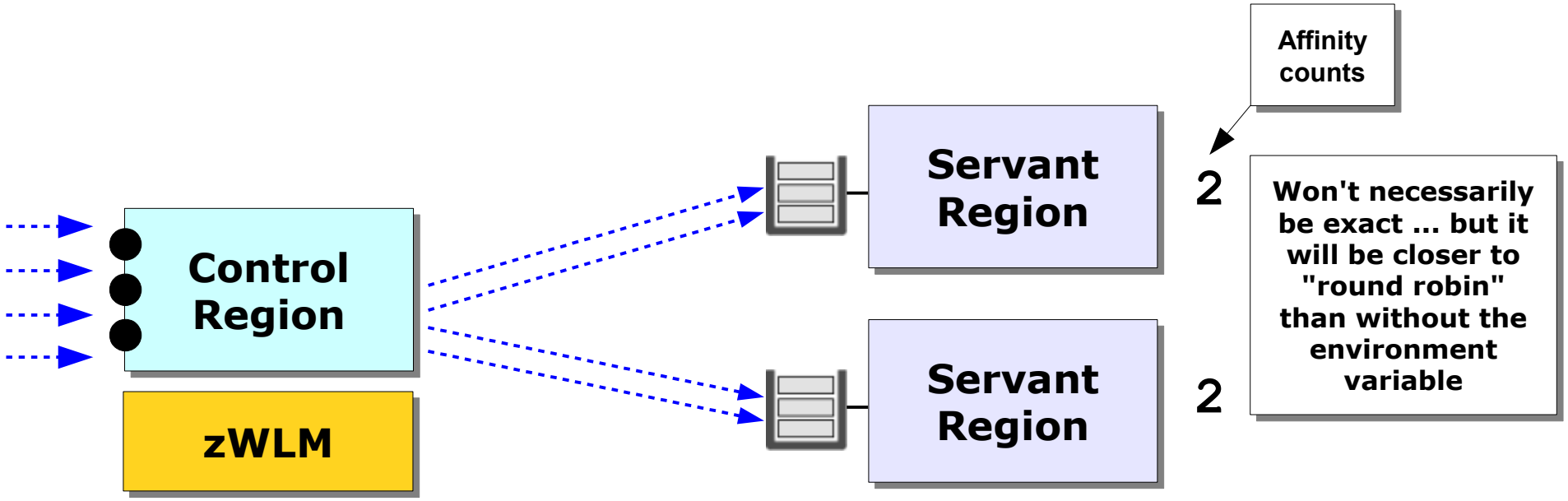
This behavior makes some sense. If all work can be handled by one servant, why not keep two swapped in and active when one will do?

There's an environment variable to *approximate* "round-robin" ...

Stateful "round-robin" ...

Environment: wlm_stateful_session_placement_on

This environment variable, when set to 1, will tell WLM to try to *balance affinities* across the available servants. It does not round-robin stateless work



Try this with stateless work and you'll find it tends to look like previous chart … work into first servant and occasional spill to others

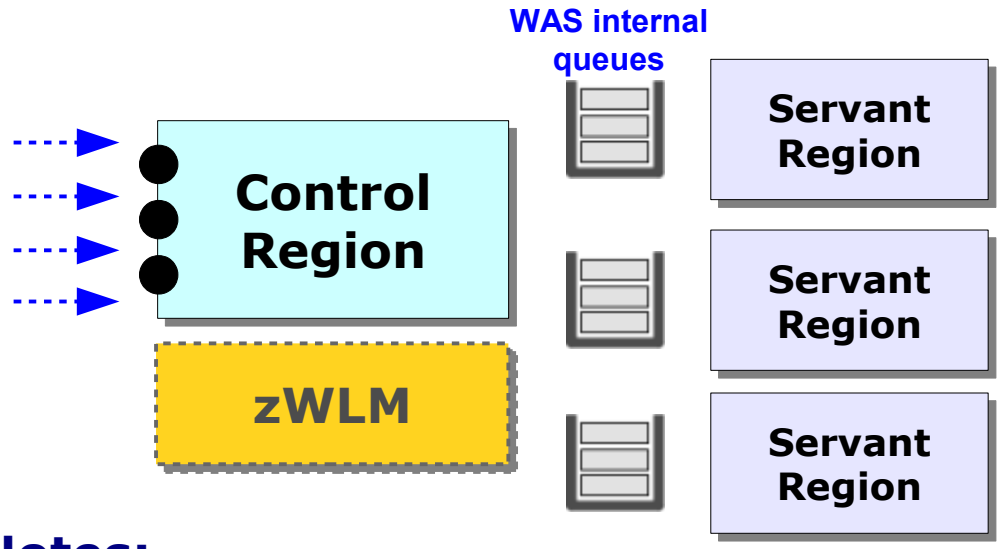
InfoCenter urun_rproperty_custproperties

Round Robin and WLM-less Queuing

For cases where you want to control work distribution to multiple servants, consider WLM-less queueing ... uses WAS queues, WLM plays minimal role

Two environment variable controls:

`server_use_wlm_to_queue_work`	0
`server_work_distribution_algorithm`	0 | 1 | 2



0 - Hot Thread

WAS looks for first available thread starting with first servant. First servant utilized the most, then second, etc.

1 - Round Robin

True round robin. If thread not available in targeted servant then work gets put into servant queue and waits for thread in that servant to free up.

2 - Hot Robin

Round Robin with a twist: if thread not available in targeted servant, then work put in queue available to all servants. First servant to free a thread gets work.

Notes:

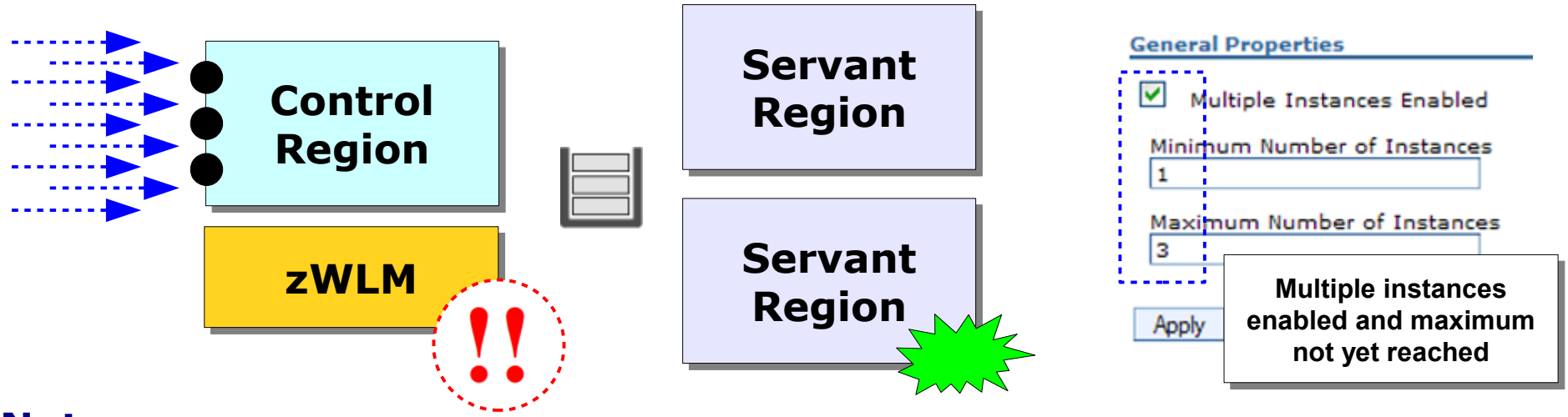
- Stateless or stateful applications ... both work with this
- WLM will still maintain minimum and restart failed
- WLM will not start additional servants
- WLM will still classify work, but it will not get involved in placement
- This works best when all the work in the server has the same Service Class

InfoCenter `urun_rproperty_custproperties`

WLM start servants? ...

Will WLM Start Servant for Spike in Work?

Yes ... again provided the MAX value has not yet been met. If incoming work can't meet goals with one servant, another started:

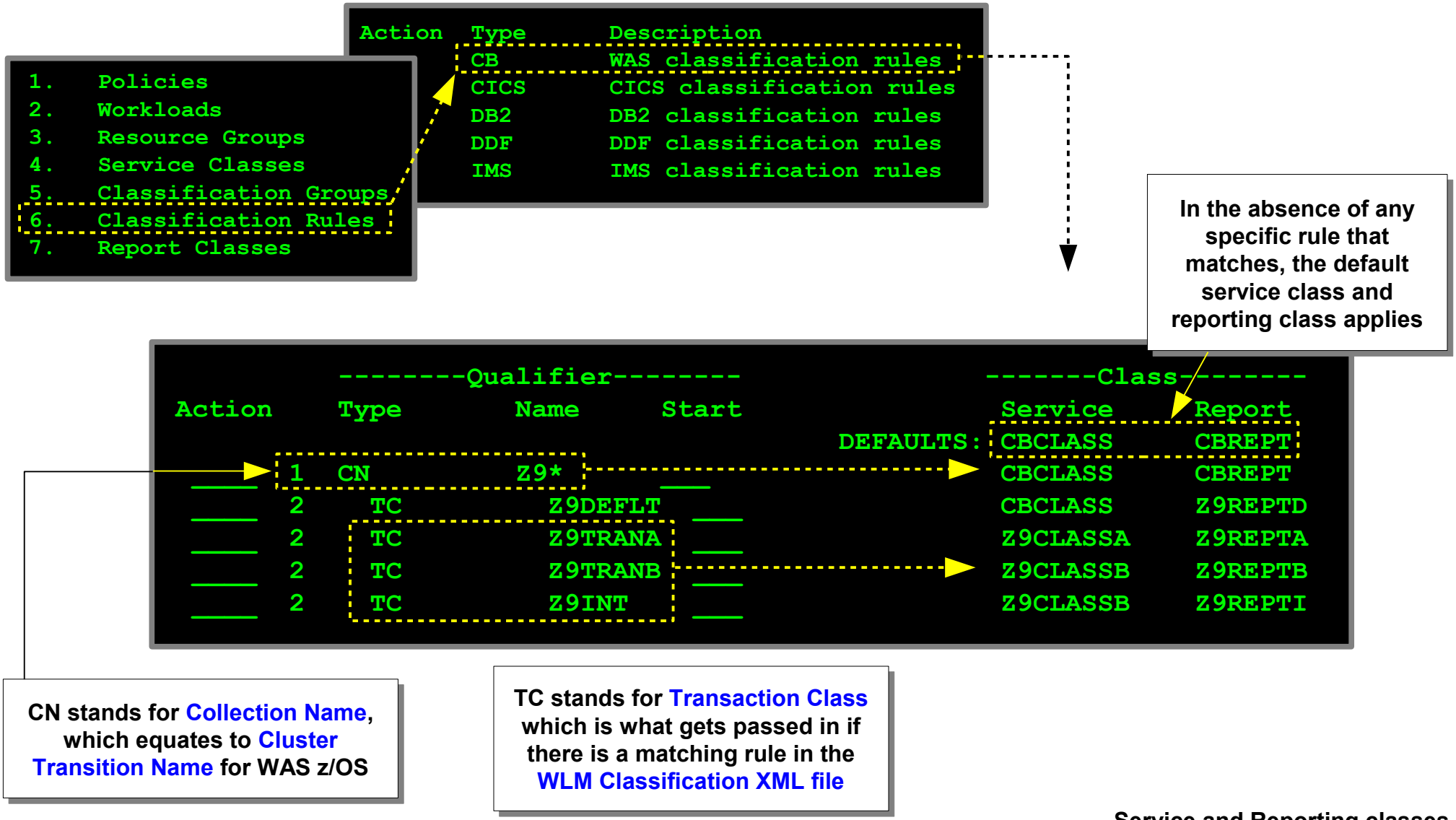


Notes:

- **WLM uses very sophisticated algorithms ... exactly when it will start the next servant region is not always easy to predict. *Not* based on simple rule of work queue depth.**
- **WLM will shut down extra servant if not needed, but it's very conservative about eliminating resources once created. You may MODIFY the MAX down to close unneeded servant regions if you wish.**
- **When second servant is up the work is *not necessarily* distributed evenly as we just saw**
- **If you're using WLM-less queueing then servant will *not* be automatically started**

The Essentials of WLM Classification

All WAS z/OS requests get classified. How that's done depends on how the "CB Rules" are coded in WLM:



Service and Reporting classes ...

WLM Service Classes and Reporting Classes

Service Classes are what WLM uses to assign priorities and manage work; Reporting Classes allow WLM to report on resource usage for specific work:

Service Classes

- A grouping of work WLM uses to understand relative priority, one group vs. others
- Service Classes carry priority goals:

  Response Time -- X% of work completes within Y amount of time

  Velocity -- a relative measure of how long work may wait for resources

  Discretionary -- work that is of lower priority and may be serviced when system has resources to do so
- It is possible to have work within a server be assigned separate Service Classes and have WLM manage that work with different priorities
- Assigning separate service classes requires the XML classification file

Reporting Classes

- A grouping of work WLM uses collect and report system resource (GP, zIIP, zAAP, etc.) usage statistics
- This is perhaps the most common reason to use the XML classification file ... to assign separate reporting classes for different work so usage statistics can be collected and reported

  XML classification file used

  Unique TC values for work to be reported separately

  One Service Class for all classified work ...

  ... But different reporting classes assigned to each

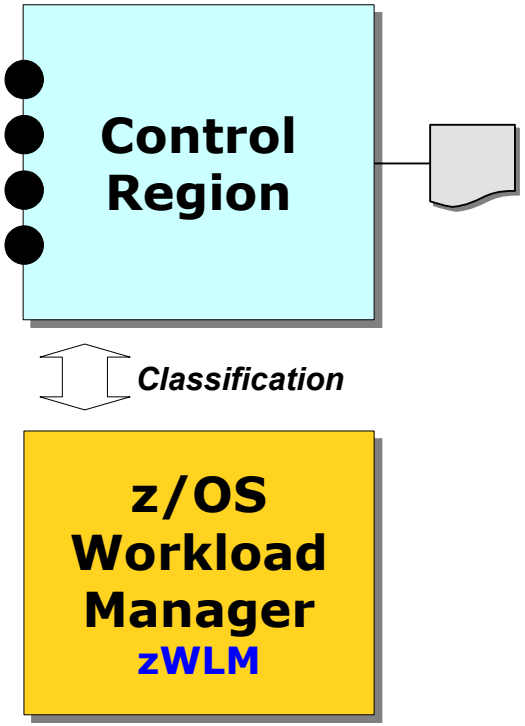
XML classification file ...

XML Classification File

This provides controller a way to assign Transaction Classes (TCs) which then get mapped to Service Classes. This is how multiple Service Classes possible in server:

Environment Variable:

```
wlm_classification_file = /u/myfiles/classification.xml
```




```
<Classification schema_version="1.0">                          1
   <InboundClassification type="http" schema_version="1.0"
      default_transaction_class="Z9DEFLT" >                    4
      <http_classification_info
            uri="/SuperSnoopWeb/*"
            transaction_class="Z9TRANA"                        2
            description="Snoop" />
      <http_classification_info
            uri="/MyIVT/*"
            transaction_class="Z9TRANB"                        3
            description="MyIVT" />
   </InboundClassification>
</Classification>
```

Transaction Classes mapped to Service Classes in WLM ... see that next

1. If the inbound work is of type "http", then ...
2. If the URI = `/SuperSnoopWeb/*` then assign TC= `Z9TRANA`
3. Or if URI = `/MyIVT/*` then assign TC = `Z9TRANB`
4. Or if no matches on rules then assign TC = `Z9DEFLT`

InfoCenter `rrun_wlm_tclass_dtd`

One SC per servant region …

One Service Class Per Servant Region

There's a subtle "gotcha" with respect to WAS internal work. This provides us a good opportunity to review planning for the number of servant regions you need:

```
<Classification schema_version="1.0">
   <InboundClassification type="http" schema_vers
      default_transaction_class="Z9DEFLT" >
      <http_classification_info
            uri="/SuperSnoopWeb/*"
            transaction_class="Z9TRANA"
            description="Snoop" />
      <http_classification_info
            uri="/MyIVT/*"
            transaction_class="Z9TRANB"
            description="MyIVT" />
   </InboundClassification>
</Classification>
```

CB Classification Rules

Action	Type	Name	Start	--Class-- Service
			DEFAULTS:	CBCLASS
1	CN	Z9*		CBCLASS
2	TC	Z9DEFLT		CBCLASS
2	TC	Z9TRANA		Z9CLASSA
2	TC	Z9TRANB		Z9CLASSB
2	TC	Z9INT		Z9CLASSB
1	TC	MY*		MYCLASS

The URI classification rules suggest at least two servant regions

Servant Region Z9CLASSA

Servant Region Z9CLASSB

What happens if another application with a different URI is in the server?

It gets the default TC and in this example gets mapped to CBCLASS

Servant Region CBCLASS

Third servant needed

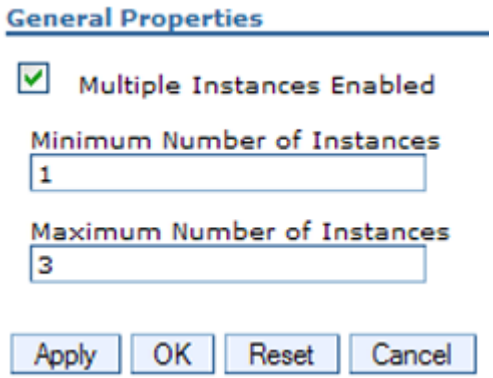
WAS internal work will be classified, and in the absence of a rule it gets the CN default ... CBCLASS

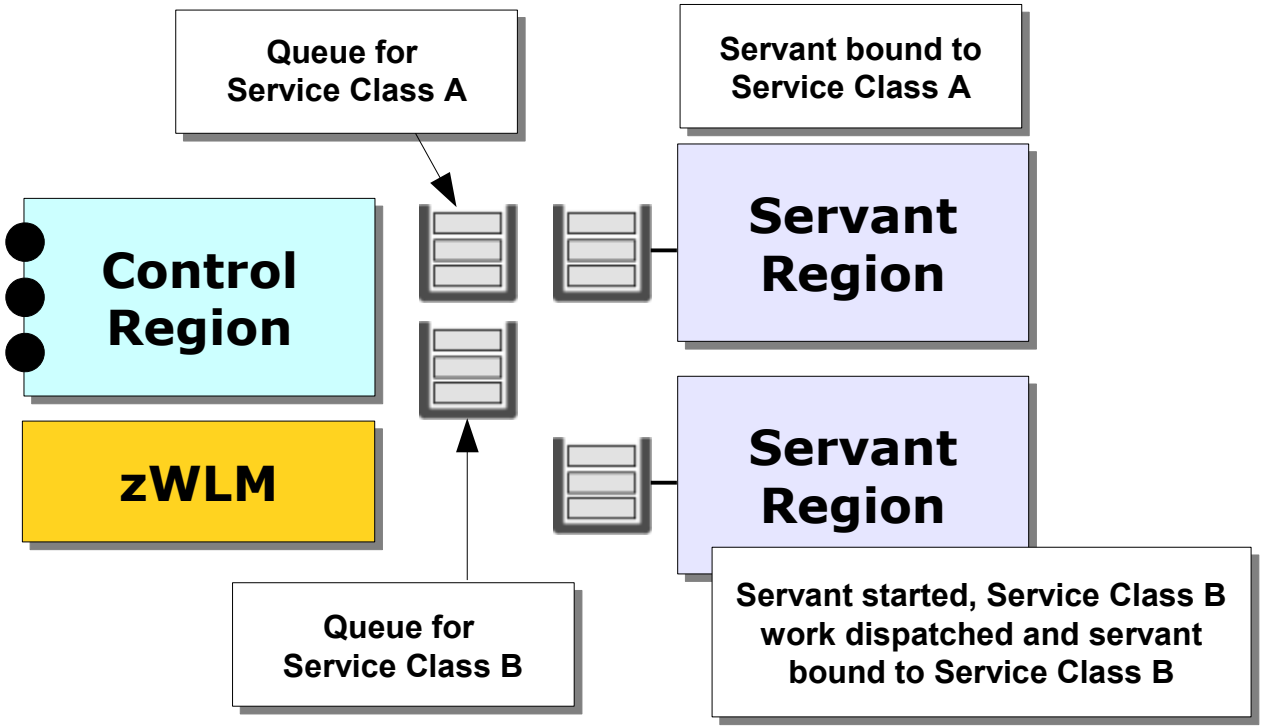
Thus it might be a good idea to take internal work into account in the XML classification rules

First ... Will WLM start servant for service class? ...

Will WLM Start Servant for a New Service Class?

Yes ... provided the MAX value has not yet been met, a Service Class that comes in without a place to go will result in the dynamic start of an additional servant:





Notes:

- **Server most likely won't have multiple Service Classes unless XML Classification File used. If you're not using that then most likely only one Service Class will be present.**
- **While servant is starting work waits in queue. Possibility exists it could timeout waiting in that queue**
- **Probably don't want to use dynamic expansion for this purpose. Understand your Service Classes and plan for MIN value accordingly.**

How to account for internal work ...

How to Account for Internal Work

There *will* be internal work classified. How can you account for it without it simply falling under the CN default Service Class?

```
<Classification schema_version="1.0">
   <InboundClassification type="http" schema_version="1.0"
      default_transaction_class="Z9DEFLT" >
      <http_classification_info
            uri="/SuperSnoopWeb/*" transaction_class="Z9TRANA"
            description="Snoop" />
      <http_classification_info
            uri="/MyIVT/*" transaction_class="Z9TRANB"
            description="MyIVT" />
   </InboundClassification>
   <InboundClassification type="internal" schema_version="1.0"
      default_transaction_class="Z9INT" >
   </InboundClassification>
</Classification>
```

"http" is one of several inbound work types:

http	internal
iiop	mdb
sip	ola

Account for internal work as shown. Then map to a TC you know will be used by one of your other rules.

Do same for the default TC and the CN default and you then have all cases covered.

		--Qualifier--		--Class--
Action	Type	Name	Start	Service
			DEFAULTS:	CBCLASS
____ 1	CN	Z9*	___	Z9CLASSB
____ 2	TC	Z9DEFLT	___	Z9CLASSB
____ 2	TC	Z9TRANA	___	Z9CLASSA
____ 2	TC	Z9TRANB	___	Z9CLASSB
____ 2	TC	Z9INT	___	Z9CLASSB

You can assign separate reporting classes to isolate out the internal work and get numbers on each service class

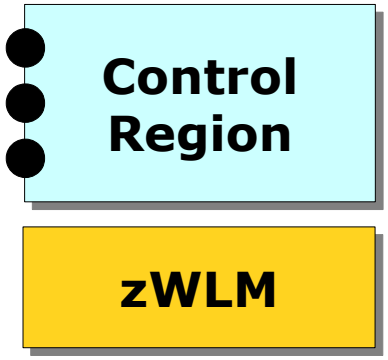
AE_SPREADMIN …

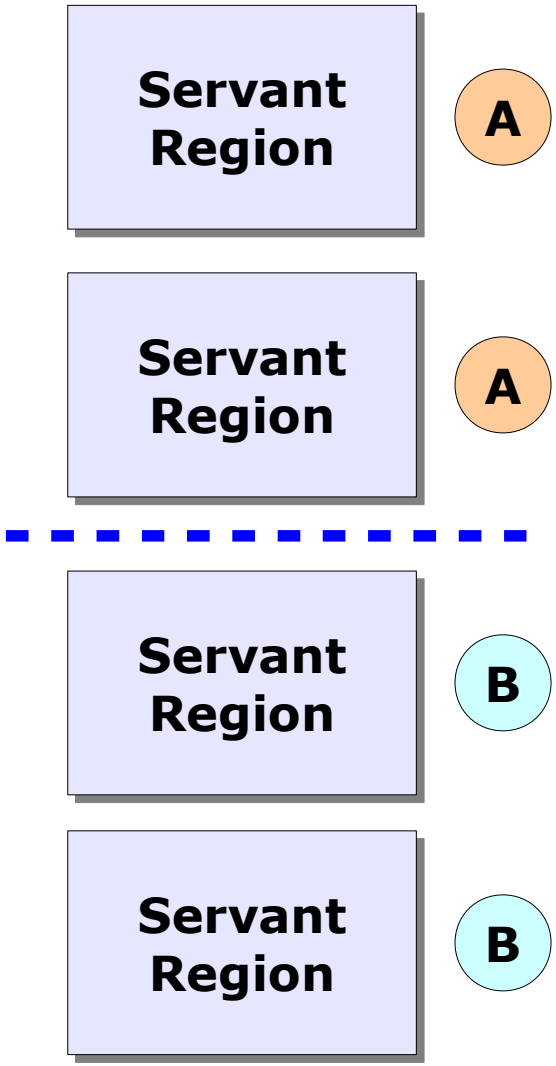
wlm_ae_spreadmin and Re-Balancing of Service Classes

This is the next level of nuance in this ... one final control that determines the behavior you see in this. Assume for example MIN=4 and two Service Classes seen:

`wlm_ae_spreadmin = 1`

Default, prior to V8 fixed at this value





With value = 1 WLM will attempt to balance service classes across the minimum servants

Servants that hosted SC=A may get rebalanced to start hosting SC=B

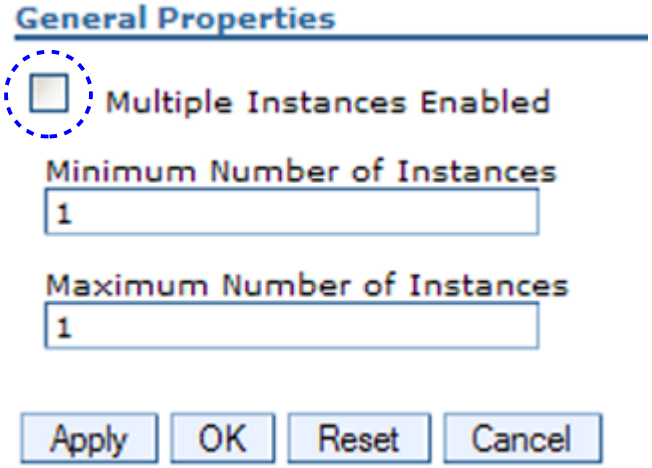
Can start new servant for SC if max not met

If #SC > max servants then nowhere to go

Loose ends …

Tying up Loose End -- Multiple Servant Instances

There's a very subtle configuration scenario you should be aware of ...

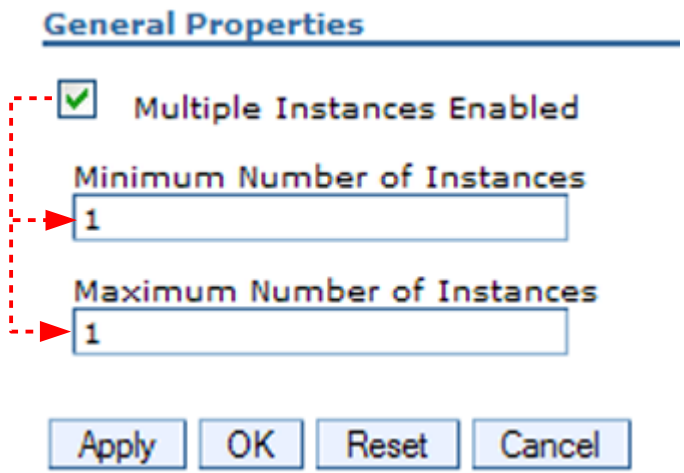


This is a true single servant environment

This will allow multiple Service Classes to co-mingle in the same servant

If you really want just one servant, this is the way to configure it.

However, can't use MODIFY to expand.



This is a multi-servant environment with only one servant

This restricts servant to one Service Class

Generally not recommended unless you're very certain about the Service Classes in use. Better to specify MAX greater than MIN to give WLM ability to process other work if needed.

You do have opportunity to MODIFY MIN and MAX higher, however.

Setting stage for Granular RAS …

XML File Extended -- Control Driven to Request Level

As we saw, the XML file identifies requests ... this new function then picks up and drives various WAS behavior controls from server level down to the request level:

```
<Classification schema_version="1.0">
   <InboundClassification type="http" schema_version="1.0"
      default_transaction_class="Z9DEFLT" >
      <http_classification_info
            uri="/SuperSnoopWeb/*" transaction_class="Z9TRANA"
            description="Snoop"  Granular Control to Request Level  />
      <http_classification_info
            uri="/MyIVT/*" transaction_class="Z9TRANB"
            description="MyIVT"  Granular Control to Request Level  />
   </InboundClassification>
</Classification>
```

- **Various Timeouts**
- **Stalled Thread Dump Actions**
- **CPU Time Used Limit**
- **DPM Interval and Dump Action**
- **SMF Recording**
- **Tracing**
- **Message Tagging**
- **Timeout Recovery Actions**

Topics to Cover in this Section:

- What those functions are and how they work
- How to dynamically reload a new or updated XML file
- How to dynamically revert to previous XML file

InfoCenter `rrun_wlm_tclass_dtd`

TechDocs `WP102023`

Preliminary notes ...

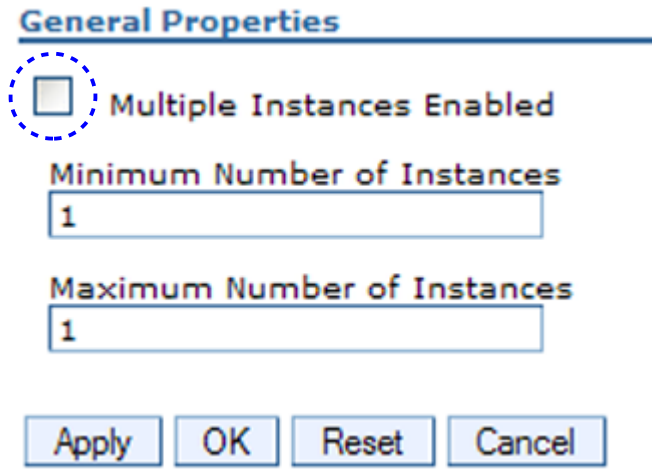
A Few Preliminary Notes

To use the granular control features implies classifying work with transaction classes as well ...

```
<Classification schema_version="1.0">
   <InboundClassification type="http"
      schema_version="1.0"
      default_transaction_class="AAA" >
      <http_classification_info
            uri="/SuperSnoopWeb/*"
            transaction_class="AAA"
            description="Snoop"
            New Function   />
      <http_classification_info
            uri="/MyIVT/*"
            transaction_class="AAA"
            description="MyIVT"
            New Function   />
   </InboundClassification>
</Classification>
```

If you don't wish to use multiple transaction classes then code all the TCs in the XML with the same value

If the CB rules in WLM don't have TC rules then other defaults will apply



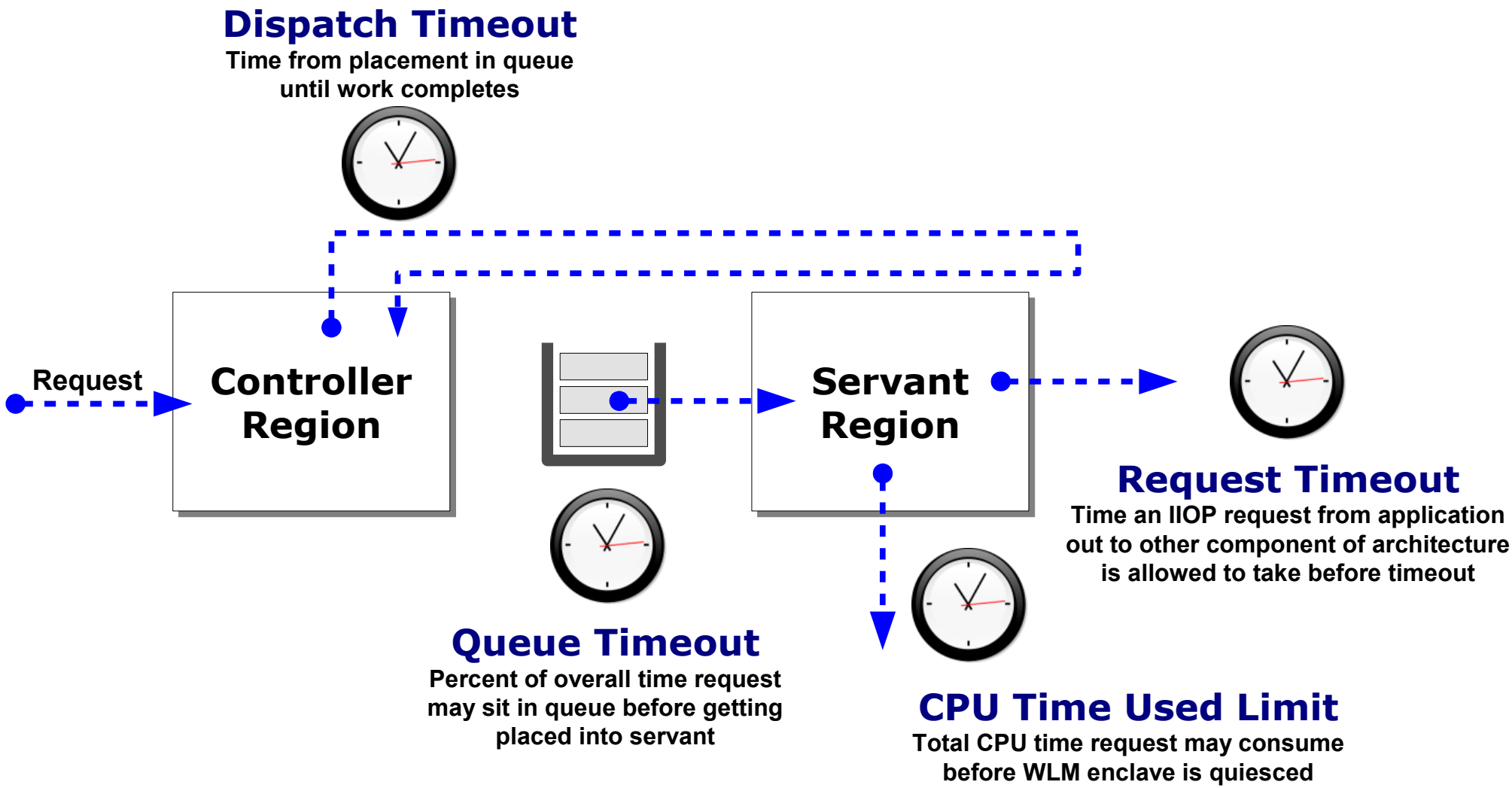
This new function does not require multiple servants, even if two or more Service Classes at work

WLM will place different Service Classes in servant if the server is true single server

Request cycle ...

High-Level of Request Cycle and Timeouts

We'll be talking about a few timeout settings ... the following picture sets context:



Dispatch timeout ...

First Example - Dispatch Timeout

Work dispatched from queue to servant starts a timer to control timeout of that work.
Before: environment variable, server level at best. Now: request level:

Server Level -- Environment Variable

```
IIOP     control_region_wlm_dispatch_timeout
HTTP     protocol_http_timeout_output
HTTPS    protocol_https_timeout_output
MDB      control_region_mdb_request_timeout
WOLA     control_region_wlm_dispatch_timeout
```

The current environment variables for HTTP dispatch timeouts. Granular down to server.

Other protocol dispatch timeouts

Granular RAS - XML Classification File

```
<Classification schema_version="1.0">
   <InboundClassification type="http" schema_version="1.0"
      default_transaction_class="TRANCL" >
      <http_classification_info
            uri="/SuperSnoopWeb/*" transaction_class="TRANCL"
            description="Snoop" dispatch_timeout="60" />
      <http_classification_info
            uri="/MyIVT/*" transaction_class="TRANCL"
            description="MyIVT" dispatch_timeout="15" />
   </InboundClassification>
</Classification>
```

XML Classification file section for HTTP. File supports `http`, `iiop`, `mdb`, `ola`, `sip`, `internal`

Requests matching this get 60 second timeout

Requests matching this get 15 second timeout

If timeout in XML and it applicable then it takes precedence over configured environment variable

InfoCenter `rtrb_controllingtimeout` Timeouts

InfoCenter `rrun_wlm_tclass_dtd` XML file

XML nesting …

XML Nesting and Effect on Precedence

The XML Classification File supports nesting, which means you may configure higher level values as well as lower level, more specific values:

```
<InboundClassification type="http"
    schema_version="1.0" default_transaction_class="TRANCL" >
  <http_classification_info transaction_class="TRANCL"
      host="host.company.com" dispatch_timeout="300" >          Open
    Open <http_classification_info transaction_class="TRANA"
        uri="/SuperSnoop/*" dispatch_timeout="60" />            Close
    Open <http_classification_info transaction_class="TRANB"
        uri="/MyIVT/*" dispatch_timeout="15"/>                  Close
  </http_classification_info>                                   Close
</InboundClassification>
```

If request received and …

… matches the `host=` *and* a `uri=`, then that timeout applies

… matches the `host=` but none of the `uri=`, then the `host=` timeout applies

… does not match `host=` then environment variable (or default) timeout applies

Granular RAS options …

The Available Granular Control Options

Here's a complete list of the options available with this new function:

Previous chart

```
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="_____">
```

Timeout for time spent in queue prior to dispatching to servant

Expressed as a percent of the dispatch timeout

Example:

- Dispatch = 300 seconds
- Queue = 10 percent
- Request must be dispatched to servant within 30 seconds or request times out
- Set this too high and request sits in queue and if dispatched has very little time to complete

Multiple keywords in XML okay …

Multiple Keywords in XML Acceptable

At this point you may be wondering whether multiple keywords can be coded in the XML, and the answer is yes ...

```
<InboundClassification type="http"
   schema_version="1.0" default_transaction_class="TRANCL" >
   <http_classification_info transaction_class="TRANCL"
      host="host.company.com"
      dispatch_timeout="300"
      stalled_thread_dump_action="traceback" >
      <http_classification_info transaction_class="TRANA"
         uri="/SuperSnoop/*"
         dispatch_timeout="60"
         queue_timeout_percent="10"
         cputimeused_limit="500" />
      <http_classification_info transaction_class="TRANB"
         uri="/MyIVT/*" dispatch_timeout="15"/>
   </http_classification_info>
</InboundClassification>
```

These will apply to lower nodes in the nested XML unless overridden at lower level

Example of three keywords used for the SuperSnoop classification node on the XML tree

Request timeout and CPU used …

Request Timeout and CPU Time Used Limit

```
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="_____">
```

Timeout for *outbound* requests issued by Java programs in servant

It is a request from the perspective of the servlet or EJB

Expressed in seconds

Maximum CPU this request may consume before having the WLM enclave quiesced

Expressed in milliseconds

Dump action …

Dump Action When Timeout Occurs

```
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="_____">
```

This controls what happens when two other controls expire:

- `dispatch_timeout`
- `cputimeused_limit`

Options are:

- `svcdump`
- `javacore`
- `heapdump`
- `traceback`
- `javatdump`
- `none`

Dispatch Progress Monitor …

Dispatch Progress Monitor (DPM) Settings

```
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="_____">
```

DPM stands for Dispatch Progress Monitor. It is a function that will process a dump action every *n* seconds.

`dpm_interval` is the interval period expressed in seconds

`dpm_dump_action` is the same as we just saw for the other dump action: `svcdump`, `javacore`, `heapdump`, `traceback`, `javatdump` and `none`

This function has a set of `MODIFY` commands that may be used to clear DPM settings or reset to XML settings

See WP102023 for the details on these `MODIFY` actions for DPM

SMF 120.9 …

SMF 120.9 Recording

```
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="_____">
```

WAS z/OS Version 7 introduced a new SMF record format -- the SMF 120 subtype 9 records.

With WAS z/OS V8 the recording of SMF 120.9 records now down to identified requests

This includes the base records as well as the optional additional information records.

Value is 0 (off) or 1 (on)

`F <server>,SMF,REQUEST,OFF` will override XML

`F <server>,SMF,REQUEST,RESET` will go back to XML settings

Tracing for Identified Requests Only

```
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="_____">
```

Prior to V8 tracing was granular to server only. All activity in the server traced. That often resulted in a great deal of trace output.

This allows you to set a trace level for the server, but trace only identified requests.

Value is 0 (off) or 1 (on)

If WAS z/OS sees this value set to 1 in the XML file, then tracing is done only for matching records.

Custom message tagging …

Custom Message Tagging

```
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="_____">
```

This allows you to place a custom string on all log, trace and system messages output for requests that match the classification.

Up to 8 characters

Output shows up as:

```
tag=MYTAG
```

within the log, trace or message.

This may affect system automation. Either correct system automation, or not use in XML, or specify environment variable:

```
ras_tag_wto_messages = 0
```

That tells WAS to ignore XML settings for message tags written to the operator console.

Message tagging goes to JES but not to HPEL

Timeout recovery …

Timeout Recovery Option

```
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="_____">
```

We are accustomed to a timeout resulting in an EC3 abend of the servant region.

The V7 feature to delay timeout abends, particularly with the hung thread threshhold setting, could delay loss of the servant.

This new function in V8 allows you to set the recovery action:

SERVANT - normal EC3 abend (or delay if hung thread threshhold in play)

SESSION - sends error message to client, then closes the TCP socket and the HTTP session. Servant stays up. Thread either completes or ends up hung.

XML file and MODIFY …

How XML File Can Be Read and Made Active

There's a few ways to bring an XML file or changes to an XML file into the server:

Environment Variable

```
wlm_classification_file = /<path>/<file>
```

Then start or restart the server

MODIFY to load initial or replace existing

```
F <server>,RECLASSIFY,FILE='/<path>/<file>'
```

WAS will load the specified file. This will replace a file named on the configured environment variable or it will load the file initially.

MODIFY to turn off classification completely

```
F <server>,RECLASSIFY,FILE=
```

WAS will cease using any classification file

MODIFY to revert to defined environment variable

```
F <server>,RECLASSIFY
```

WAS will re-read whatever XML file you were most recently using

This is a way to update the current XML and have WAS read it in to have the changes take effect

Checking for state of XML …

Checking The State of the Classification File

Here's a quick summary of what to check for to make certain what file was loaded and whether any XML parsing errors occurred:

In the Control Region output -- Positive Sign

```
BBOJ0129I: The /wasetc/was8lab/other/classification.xml workload
classification file was loaded at 2011/11/25 12:22:22.710 (EST)
```

In the Control Region output -- Sign of Problems

```
BBOJ0085E: PROBLEMS ENCOUNTERED PARSING WLM CLASSIFICATION XML FILE
```

It then offers fairly good details on what the problem is

MODIFY to see the state of the XML file

```
F Z9SR01A,DISPLAY,WORK,CLINFO
  :
BBOJ0129I: The /wasetc/was8lab/other/classification.xml
workload classification file was loaded at 2011/11/26
14:58:28.586 (EST)
```

The Liberty Profile

Single JVM, composable, dynamic


WebSphere Application Server for z/OS Version 8.5

Liberty Profile

Quick Start Guide

Version Date: July 4, 2012

See "Document Change History" on page 35 for a description of the changes in this version of the document

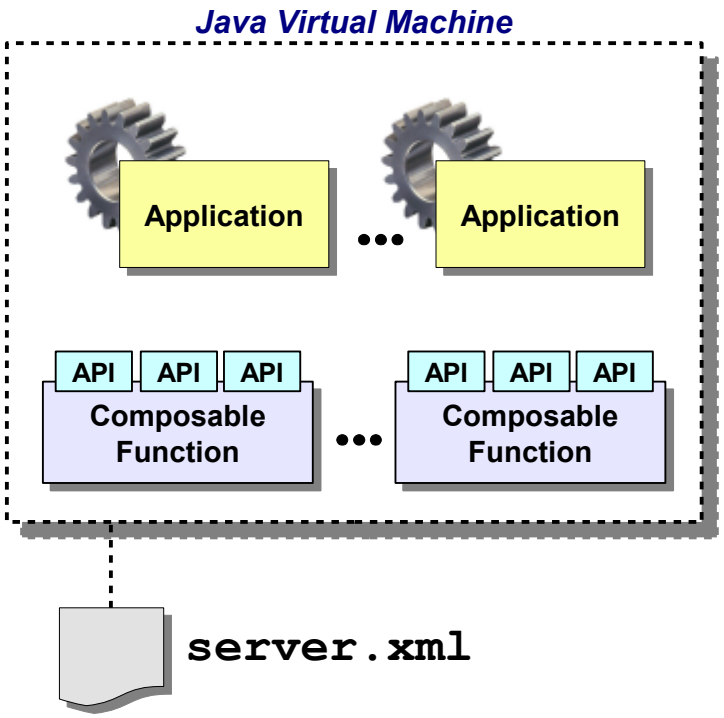
IBM Advanced Technical Skills
Gaithersburg, MD

WP102110 at

ibm.com/support/techdocs

Overview of the Liberty Profile

The Liberty Profile is designed to be a single-JVM server model that is lightweight, composable and dynamic:



- **Composable** -- you configure the function the application needs; you don't need to load up everything
- **Dynamic** -- changes to configuration or changes to applications detected and dynamically enabled
- **Subset of traditional WAS function**
  Liberty is not full Java EE, traditional WAS is
- **Upwards application compatibility**
  Apps that run in Liberty will run in traditional WAS ... but not necessarily the other way around since Liberty is subset of traditional WAS
- **Each server is one JVM**
- **Run from UNIX shell or as started task**
- **One required configuration file: `server.xml`**
- ***Not* part of traditional WAS administrative DMGR, federated node model**
  But there is an ability to manage via the "Job Manager" function of traditional WAS (advanced topic, we won't get into that here)

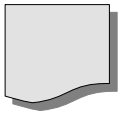
Composable "features" …

Server "Features" -- Composable Functionality

The InfoCenter lists the features that may be configured into the server.xml, which provides those functions to the Liberty Profile server instance:

```
beanvalidation-1.0
blueprint-1.0
jaxrs-1.1
jdbc-4.0
jndi-1.0
jpa-2.0
jsf-2.0
jsp-2.2
json-1.0
localConnector-1.0
monitor-1.0
osgi.jpa-1.0
restConnector-1.0
ssl-1.0
appSecurity-1.0
serverStatus-1.0
servlet-3.0
sessionDatabase-1.0
wab-1.0
zosSecurity-1.0
zosTransaction-1.0
zosWlm-1.0
```

InfoCenter `rwlp_feat`



`server.xml`

```
<server description="myServer">
<featureManager>
   <feature>servlet-3.0</feature>
   <feature>jdbc-4.0</feature>
   <feature>zosTransaction-1.0</feature>
</featureManager>
  :
```

Web applications only in Version 8.5 of Liberty

Update `server.xml` with new feature and feature dynamically loaded (server restart not needed)

If you specify a feature and that implies another is also needed, Liberty will automatically load the other as well

z/OS extensions:

- Use of SAF as identity store and trust/keystore
- Use of RRS for Type 2 transaction management
- Ability to classify work (remember Report Class discussion)
- Ability to use MODIFY commands if run as started task

Liberty 8.5.5 …

Server "Features" -- Version 8.5.5 update

V8.5.5 saw significant updates to the Liberty Profile features:

- **ejblite** - session (stateful, stateless), JPA, container TX
- **managedBeans** - JMX and mBean support
- **oauth** - open standard for authorization
- **cdi** - contexts and dependency injection
- **webCache** - Dynacache or use WXS or DataPower caching
- **concurrent** - asynchronous work with context of calling thread
- **wasJmsClient** - JMS client to Liberty engine or full WAS SIBus
- **wmqJMSClient** - JMS client to MQ
- **jmsMdb** - host JMS MDB application
- **wasJmsServer** - hosts messaging engine and queue
- **wasJmsSecurity** - for messaging engine
- **jaxb** - Java Architecture for XML Binding 2.2
- **jaxws** - Java API for XML Web Services 2.2
- **wsSecurity** - web services security
- **mongodb** - open standard noSQL database
- **ldapRegistry** - use LDAP for security registry

The JMS support was a known limitation of Liberty 8.5 and with 8.5.5 that function is provided

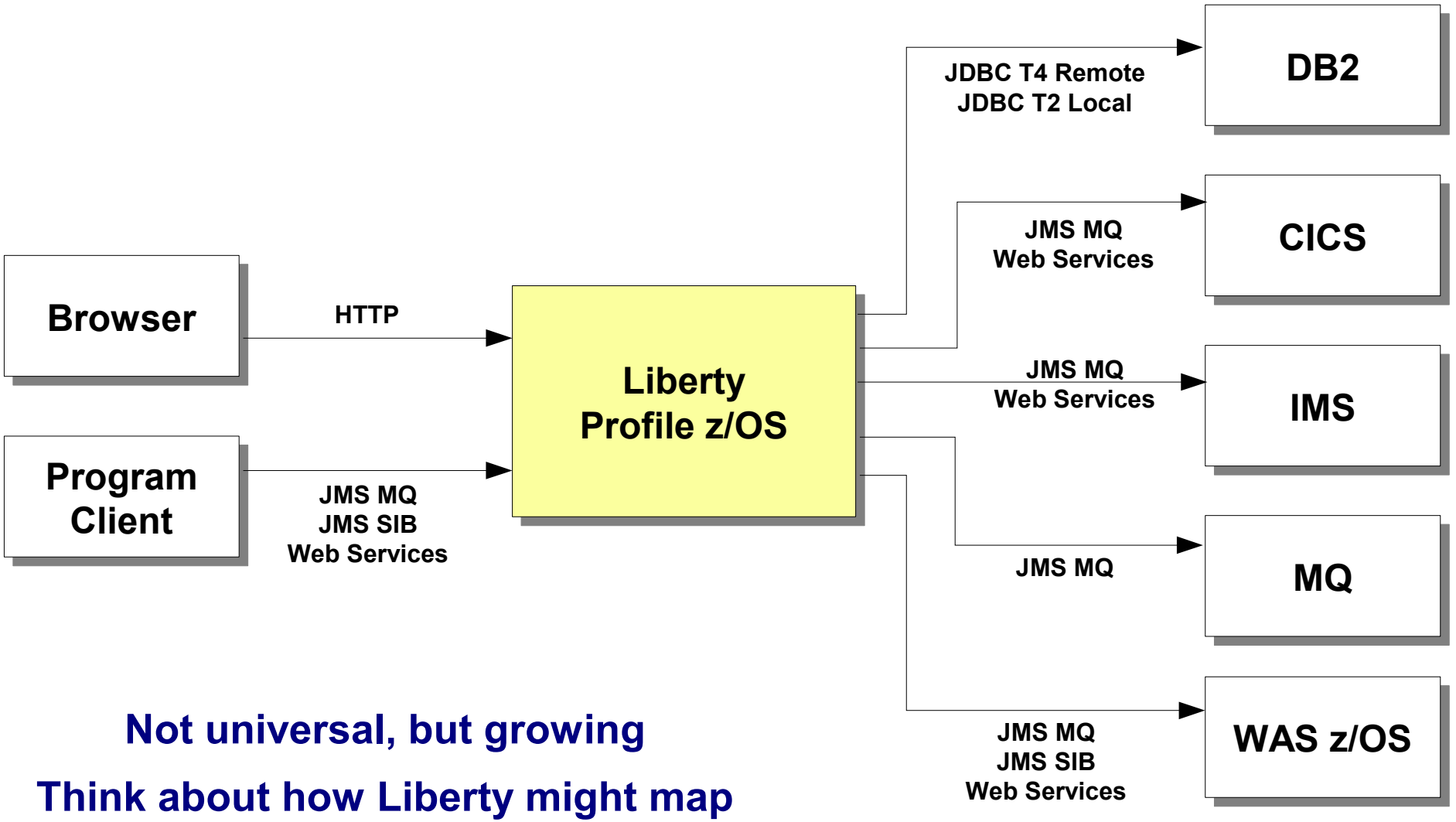
Building the capabilities of Liberty Profile

InfoCenter `rwlp_feat` 8.5 InfoCenter updated with 8.5.5 "What's New" tags

Connectivity options …

Connectivity Options with 8.5.5

A summary chart of connectivity options with Liberty z/OS:



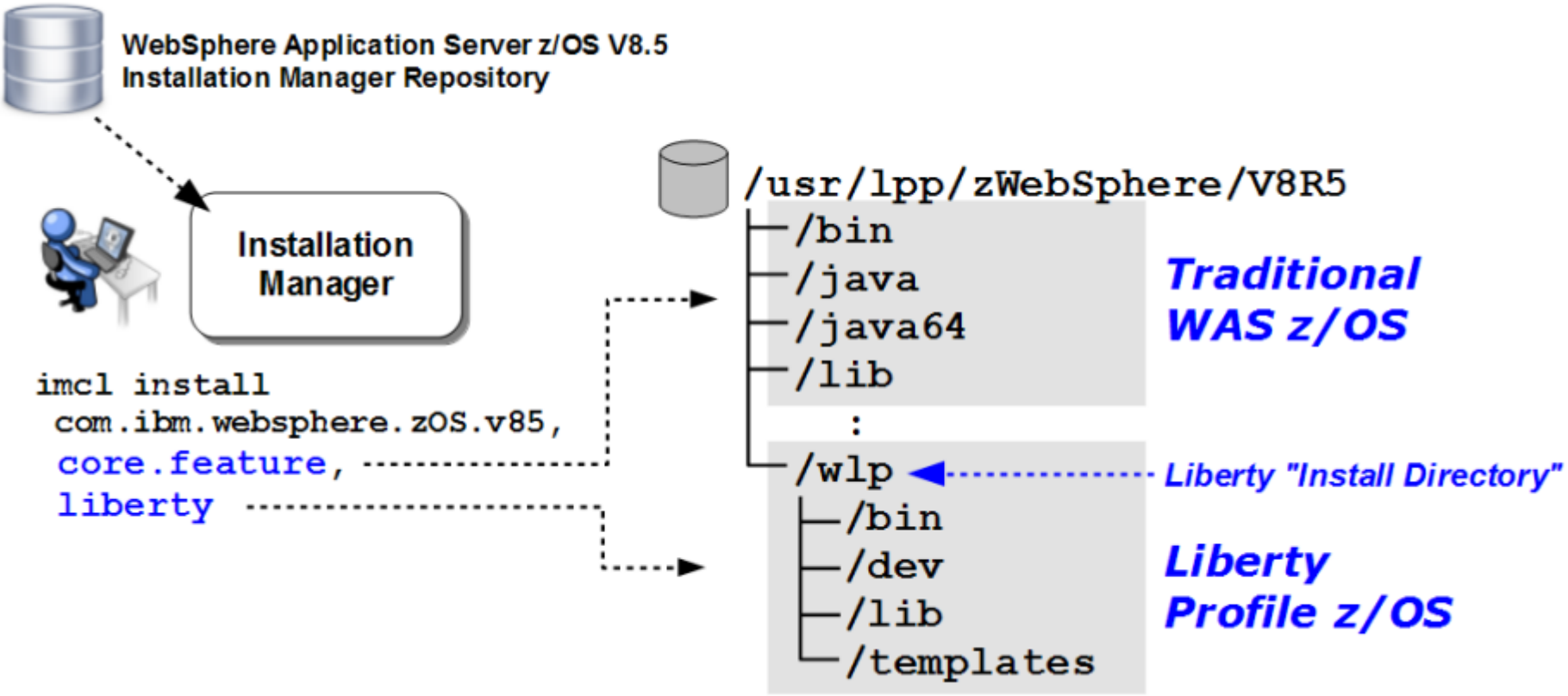
Not universal, but growing

Think about how Liberty might map into the lower end of the architecture

Liberty in the file system …

What 8.5.0 Liberty Looks Like in File System

When you specify option `liberty` to IM it will install the following directory and file structure into the target location:



Those files are relatively small ... around 60MB. They represent the product files of Liberty. You will likely have this as a read-only file system. So where do the configuration files go? In a "user directory" ...

V8.5.5 ...

What 8.5.5 Liberty Looks Like in File System

With 8.5.5 the installation of Liberty changes a bit:

Separate package name

```
imcl install com.ibm.websphere.liberty.zOS.v85,
liberty,embeddablecontainer,extprogmodels +
-installationDirectory /usr/lpp/zWebSphere/Liberty
```

Separate install directory from WAS itself

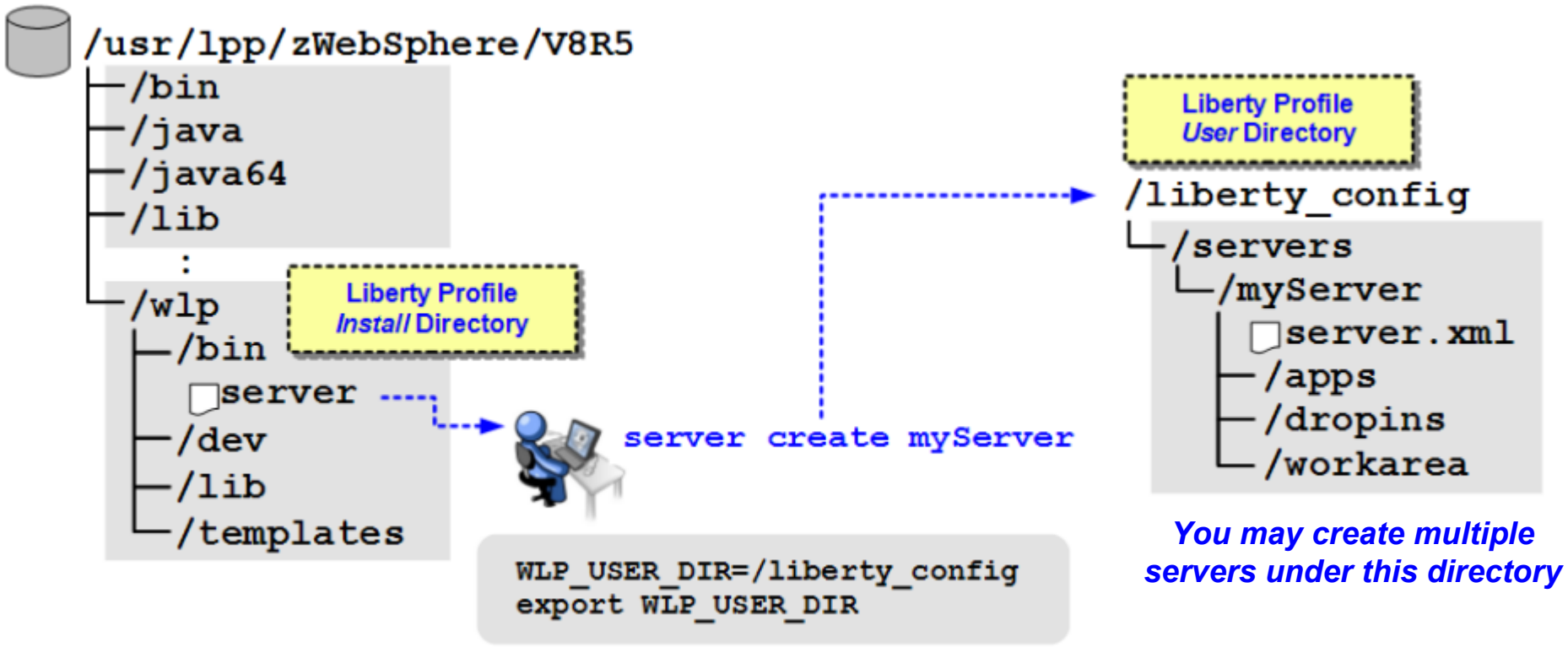
```
<dir>
├─/bin
├─/clients
├─/dev
├─/lib
└─/templates
```

Then the structure is essentially the same as the prior chart, with 8.5.5 delivering additional function

Creating a server …

Creating a Server ... and the "User Directory"

The server configuration files and directory structure may be created at a separate location ... called the "user directory":

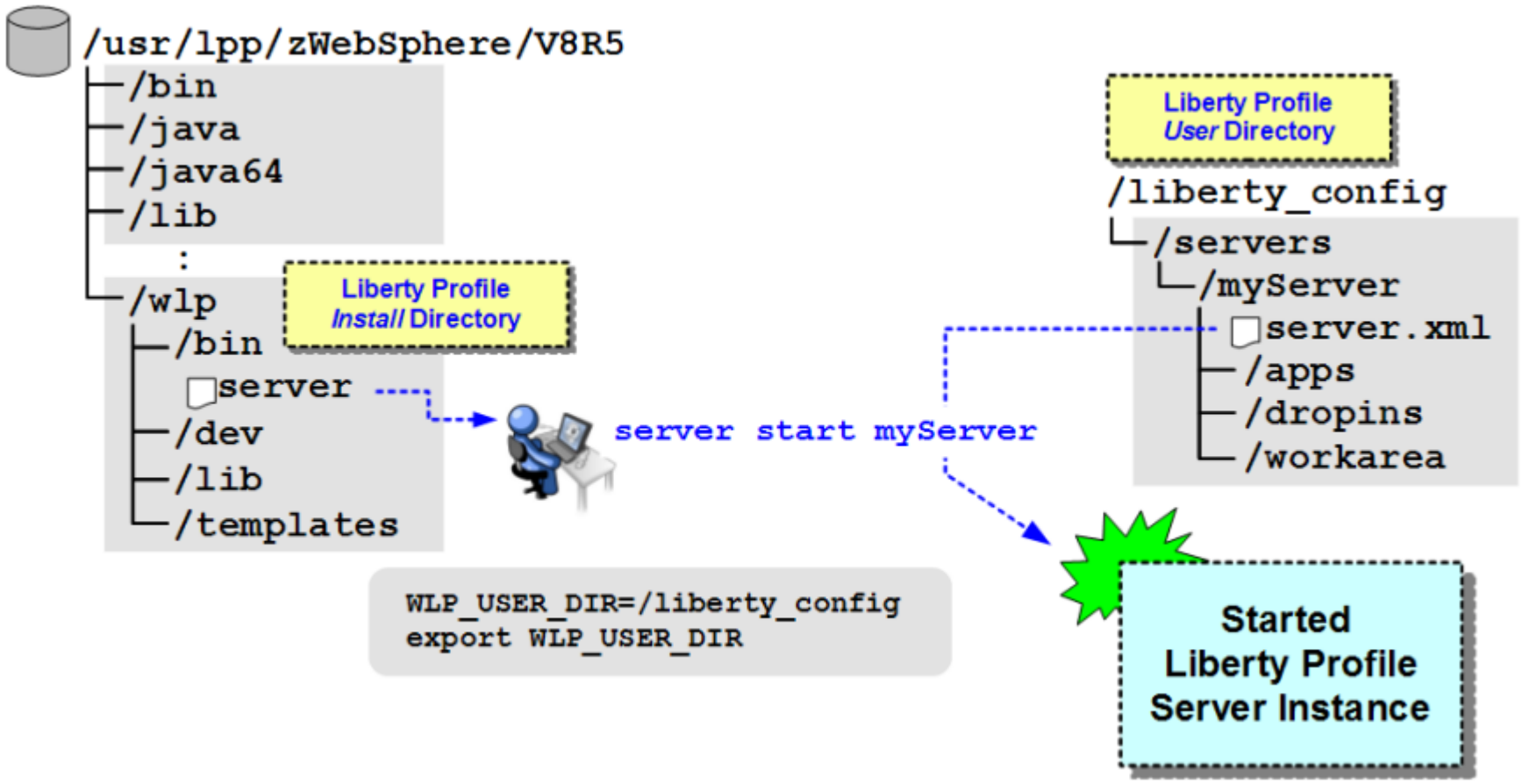


That "user directory" may be located anywhere, and Liberty may operate under any ID you wish. The key is the `WLP_USER_DIR` shell environment variable ... that tells Liberty where the server configurations reside

Starting the server in UNIX shell ...

Starting a Server from the UNIX Shell

Liberty may be started from the UNIX shell or as a z/OS started task. Here we show how it is started from the UNIX shell:

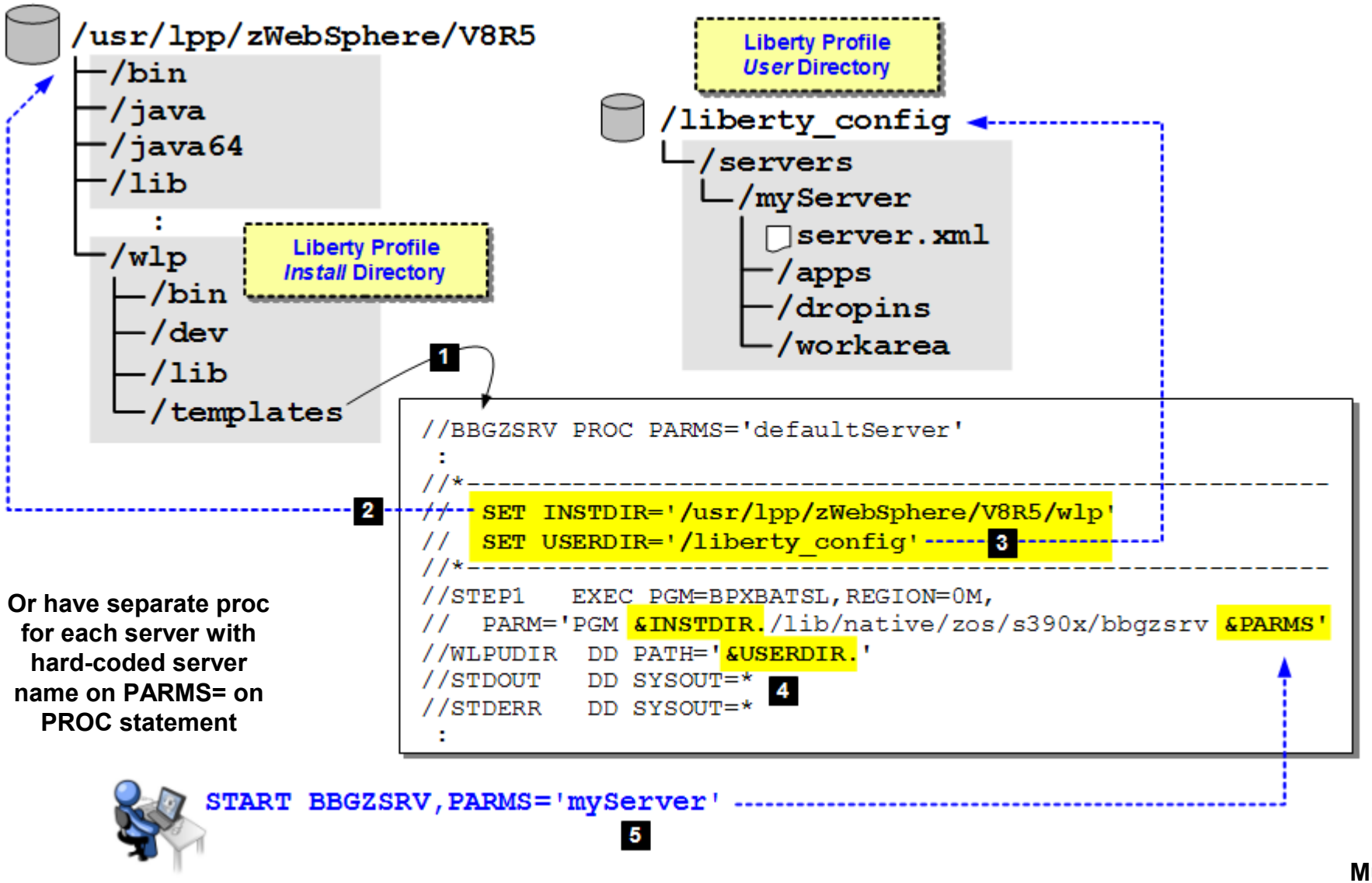


The server shell script may also be used to check the status of a server or stop the server (along with several other administrative actions)

Starting server as z/OS started task …

Starting a Server as a z/OS Started Task

Liberty may be started from the UNIX shell or as a z/OS started task. Here we show how it is started from the UNIX shell:



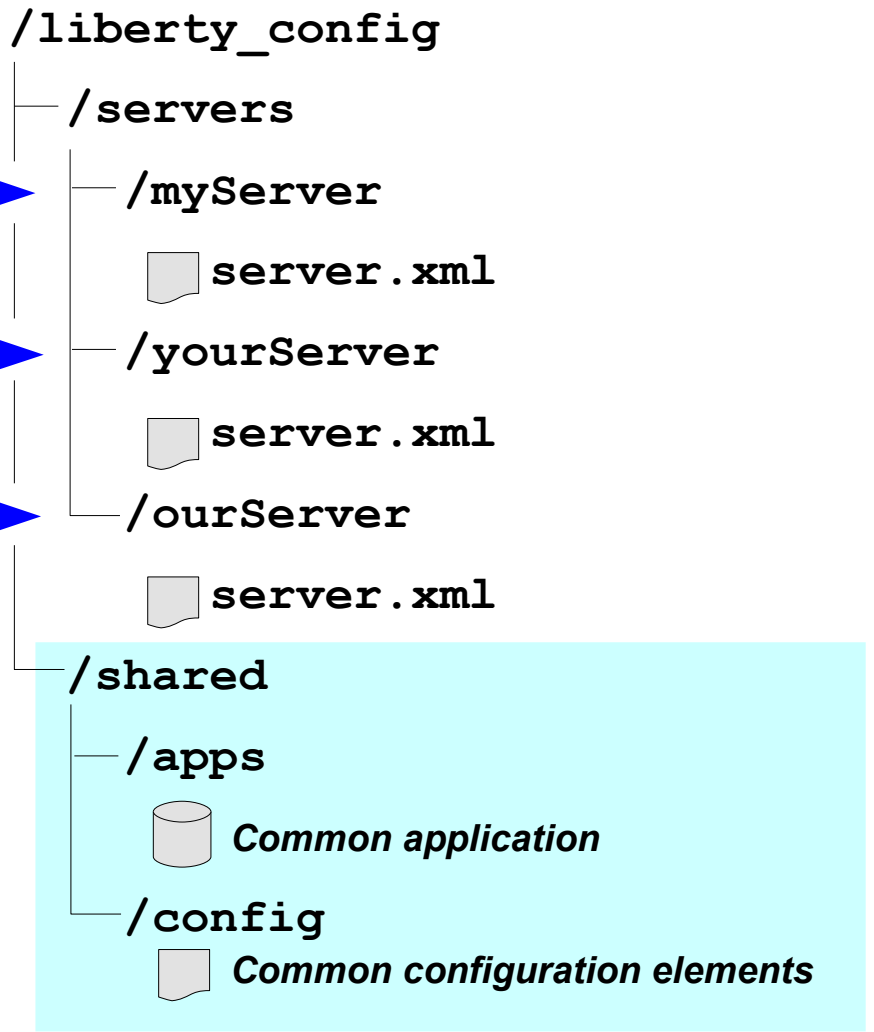
Or have separate proc for each server with hard-coded server name on PARMS= on PROC statement

Multiple servers …

Multiple Servers Under Same User Directory

You may create multiple servers under the same user directory, and those servers may then share a set of common directories:

```
WLP_USER_DIR=/liberty_config
server create myServer
server create yourServer
server create ourServer
```



`${shared.app.dir}`

Use this variable in configuration XML to refer to the `/shared/apps` directory under the user directory where the server operates

`${shared.config.dir}`

Use this variable in configuration XML to refer to the `/shared/config` directory, and use `<include>` tag to bring in common XML

The server.xml file …

The `server.xml` File ... the Central Configuration File

The server.xml file provides the Liberty Profile server instance information about what features to load and other information needed to perform the needed functions:

```
<server description="myServer">

    <featureManager>
        <feature>servlet-3.0</feature>
        <feature>jdbc-4.0</feature>
        <feature>zosTransaction-1.0</feature>
    </featureManager>

    <jdbcDriver id="DB2T2" libraryRef="DB2T2LibRef" />

    <library id="DB2T2LibRef">
        <fileset dir="/shared/db2a10/jdbc/classes/" />
        <fileset dir="/shared/db2a10/jdbc/lib/" />
    </library>

    <dataSource id="ds1"
        jndiName="jdbc/exampleDS"
        jdbcDriverRef="DB2T2">
      <properties.db2.jcc driverType="2" databaseName="WSCDBP0"/>
    </dataSource>

     <httpEndpoint id="defaultHttpEndpoint"
      host="*"
      httpPort="19123" />

</server>
```

This example shows how to configure JDBC T2 using z/OS RRS

No application definition in this example ... apps are placed in `/dropins` directory and dynamically picked up

There is an `<application>` tag that may be used to explicitely define application and WAR file location

Remember: this file may be updated and changes dynamically detected and incorporated

Possible, but now shown:

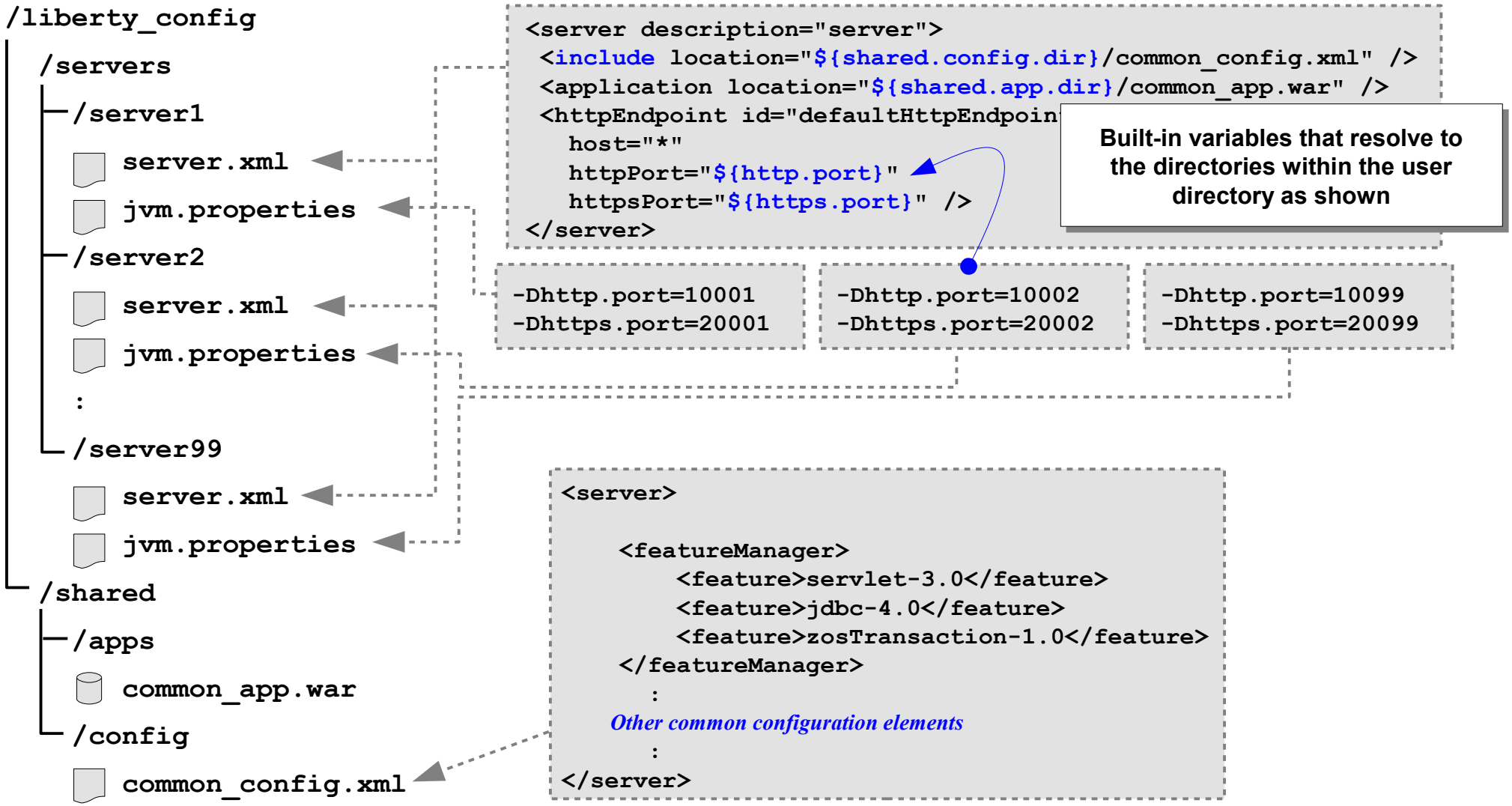
- ***Substitution variables***
- ***Includes from other files***
- ***Many other features***

InfoCenter `rwlp_metatype_4ic`

Multiple servers sharing configuration …

Multiple Servers, Common `server.xml`

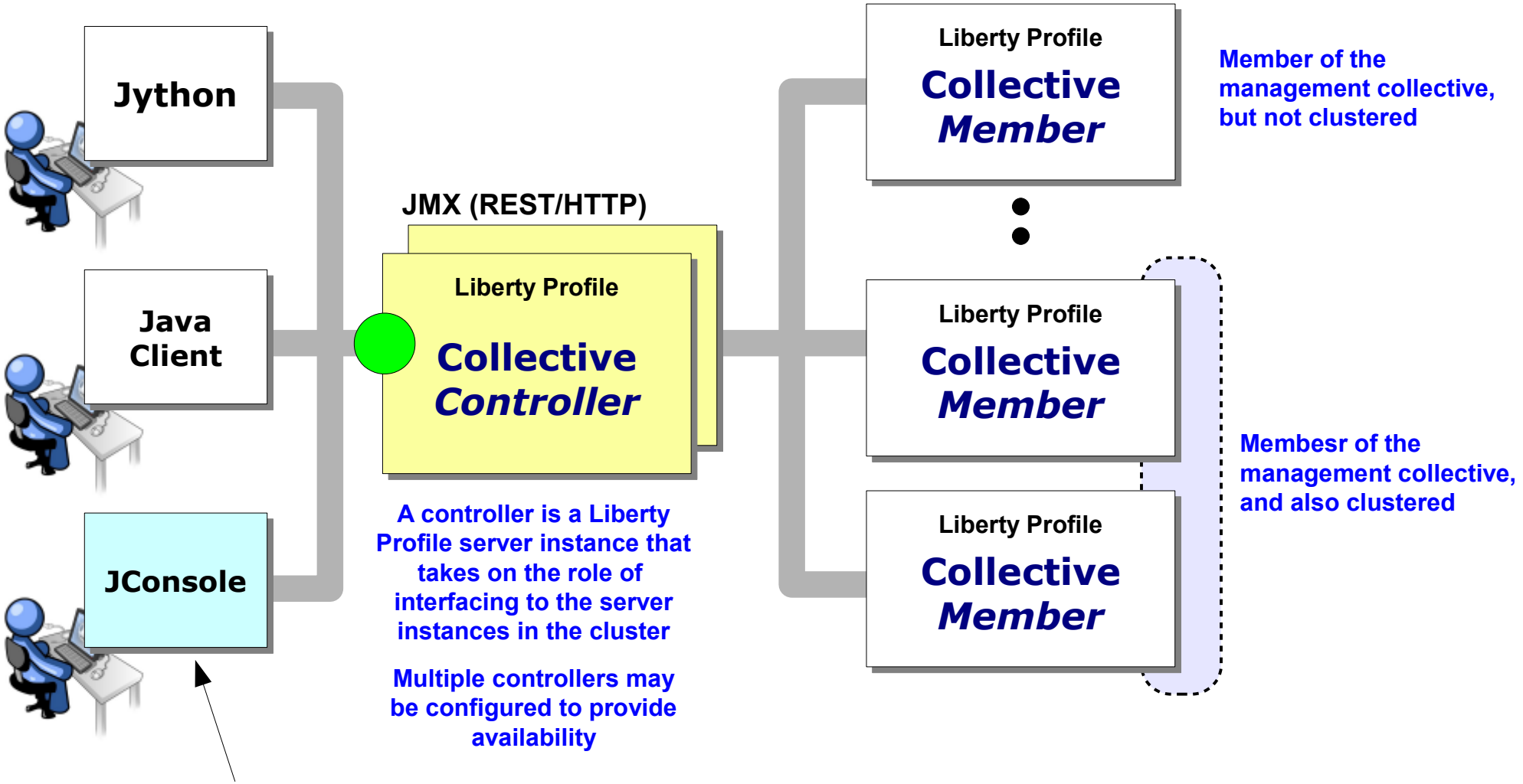
Previous chart mentioned substitution variables and file includes. This makes possible multiple servers having a common `server.xml`, but having unique values:



V8.5.5 Collectives …

Version 8.5.5 Collectives

Collectives are groupings of Liberty Profile server instances for the purposes of management and monitoring. Collectives may be accessed through a controller:

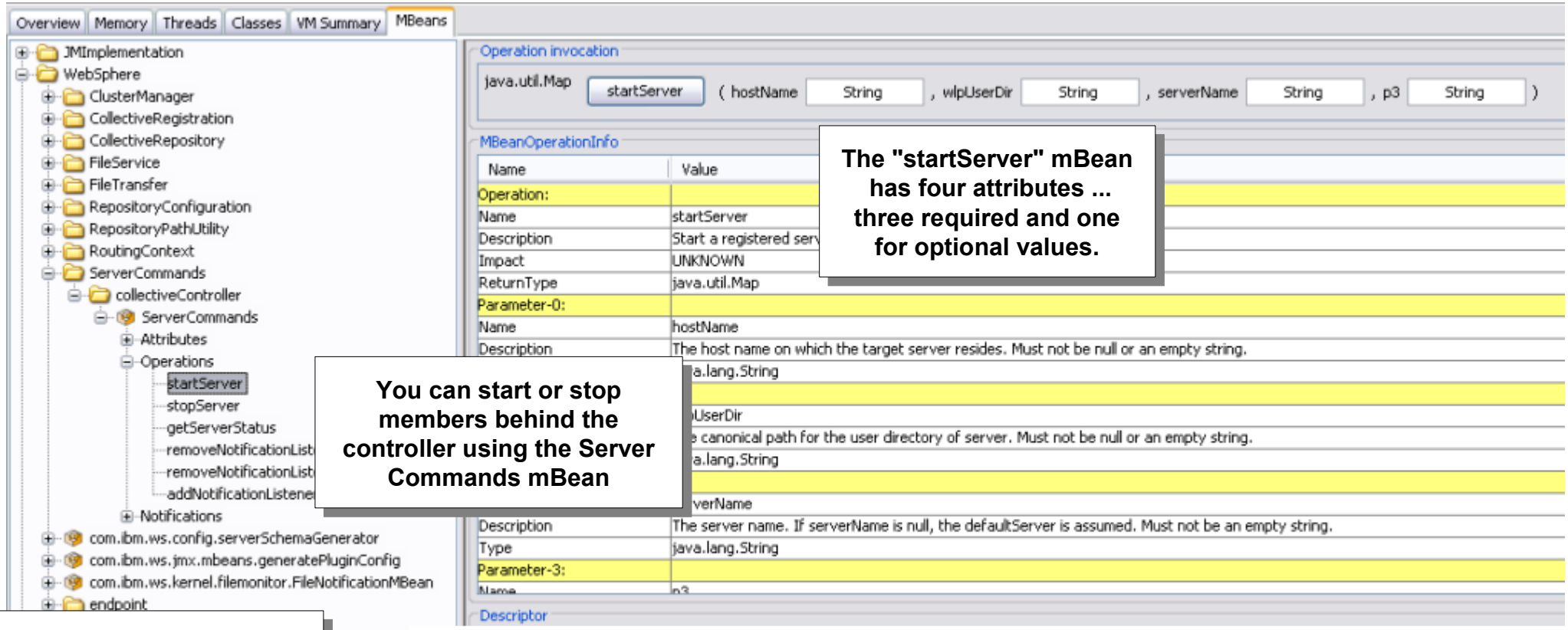



```
service:jmx:rest://<host>:<port>/IBMJMXConnectorREST
```

JConsole ...

Version 8.5.5 Collectives - JConsole Example

JConsole provides a GUI interface to issue JMX commands to the controller, which then routes to the target member or cluster:



The "startServer" mBean has four attributes ... three required and one for optional values.

You can start or stop members behind the controller using the Server Commands mBean

JConsole showing the mBeans explosed by the JMX RESTful interface of the controller

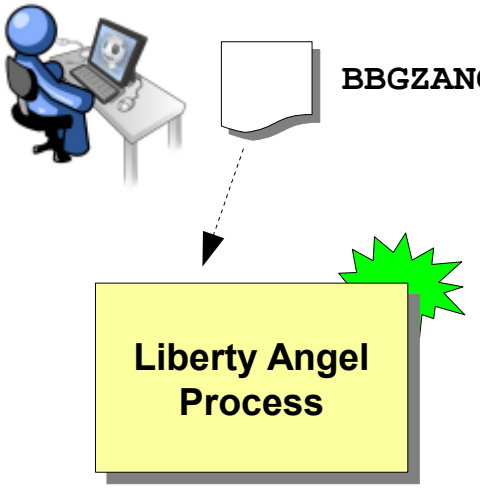
Your Java client or Jython scripts would do essentially the same thing -- connect to the Controller JMX interface and invoke the mBeans to perform the operations supplied by the mBeans

InfoCenter `rwlp_mbeans_list`

Angel process ...

The "Angel" Process and its Role with Liberty

The Angel process provides an anchor point for access to z/OS authorized services. There are several important things to note about the Angel process:



- **Not strictly required**
  - Only required if there's a Liberty server instance on the LPAR that requires access to z/OS authorized services
- **If needed, then only one per LPAR**
  - Whether one Liberty server instance or a thousand
- **Very lightweight**
  - Very little memory, almost no CPU once started, no TCP ports, no configuration files
- **Access to authorized through `SERVER` profiles**
  - Small handful of `SERVER` profiles to set up ... you grant `READ` to server ID
- **Services: SAF, WLM, RRS, z/OS DUMP**
  - Of those, only RRS and z/OS DUMP *require* Angel process; SAF and WLM will work without but not as efficient as authority check then done for every call rather than once

z/OS extensions …

z/OS Extensions to the Liberty Profile

A brief summary of the specific exploitation of z/OS functions provided by Liberty when run on the z/OS platform:

SAF

- Use SAF for authentication repository (userid and passwords)
- Use SAF for trust and key store (digital certificates)
- If Angel, then SERVER profile: `BBG.AUTHMOD.BBGZSAFM.SAFCRED`

WLM

- Provide transaction classification (TC) to work requests
- Elements in `server.xml` provide classification rules *(not separate XML file like trad. WAS z/OS)*
- Common use-case: provide separate reporting classes for work
- If Angel, then SERVER profile: `BBG.AUTHMOD.BBGZSAFM.ZOSWLM`

RRS

- Use for JDBC Type 2 with RRS for transaction management
- Angel process required for this
- SERVER profile: `BBG.AUTHMOD.BBGZSAFM.TXRRS`

DUMP

- Provides ability MODIFY request for SVCDUMP or Java Transaction (TDUMP)
- Angel process required for this
- SERVER profile: `BBG.AUTHMOD.BBGZSAFM.ZOSDUMP`

Summary …

Summary of Unit

Two fundamental server models:

- "Traditional" WAS z/OS -- the multi-JVM, CR/SR model
- New "Liberty Profile" ... enhanced in V8.5.5

Traditional WAS z/OS:

- CR does request handling, SR hosts applications and data access
- WLM work queueing betweeen CR and SRs
- Classification file enables multiple service classes and reporting classes
- Classification file extension to support Granular RAS function

Liberty Profile

- Packaged / delivered with WAS V8.5 ... operationally different from trad. WAS
- Lightweight, composable function, dynamic updates
- Web applications in V8.5, enhanced in V8.5.5 with EJB Lite and much more
- z/OS extensions to exploit SAF, WLM, RRS and z/OS DUMP