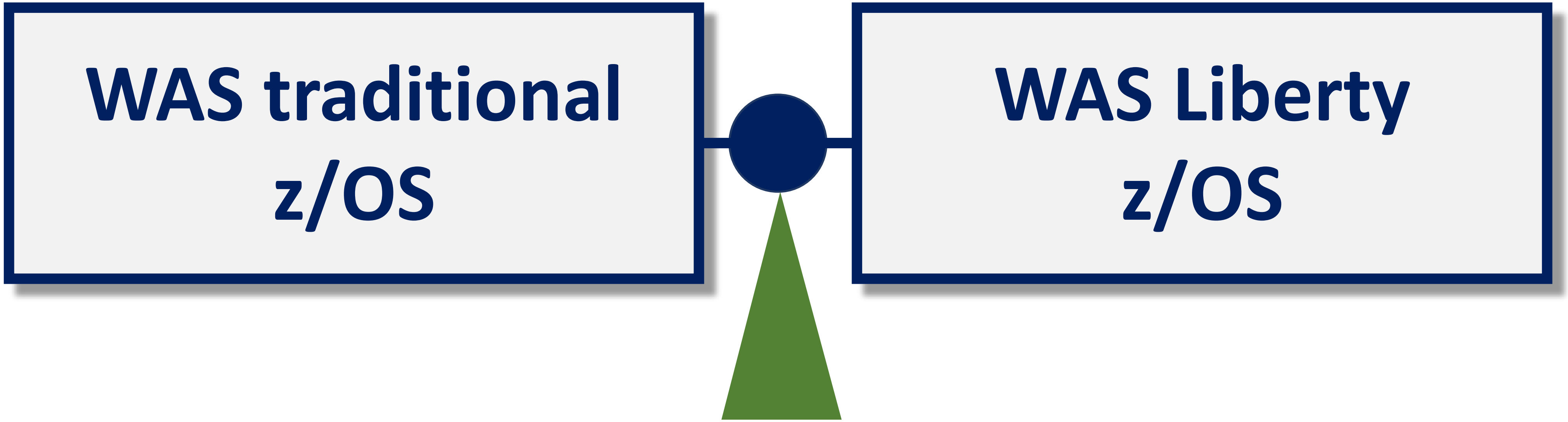
WebSphere Application Server z/OS



Deciding Which to Use

✓ The answer may well be "both" ... the intent of this material is to help you understand and weigh the considerations of both against the needs of your application serving architecture. Utilizing *both* WAS traditional and Liberty is a pattern that may fit your needs.

Hyperlink

➡ **Executive Overview**

A one-chart summary of the usage considerations presented in the document

➡ **Setting Context**

Establishing terminology and providing background on the evolution over time of each runtime models

➡ **Application Considerations**

Exploring the application interface considerations of each runtime model

➡ **Operational Considerations**

Exploring the runtime operational considerations of each runtime model

➡ **Performance Considerations**

Exploring the performance profile of each runtime model

➡ **Other Information for Consideration**

A collection of other information you may find useful when making this decision

Executive Overview

Executive Summary

Liberty is the newer runtime model and has considerable IBM focus and investment

WAS traditional z/OS continues to be a viable platform with IBM support into future

Liberty z/OS benefits include: smaller memory footprint, greater zIIP offload, more flexible configuration and application deployment

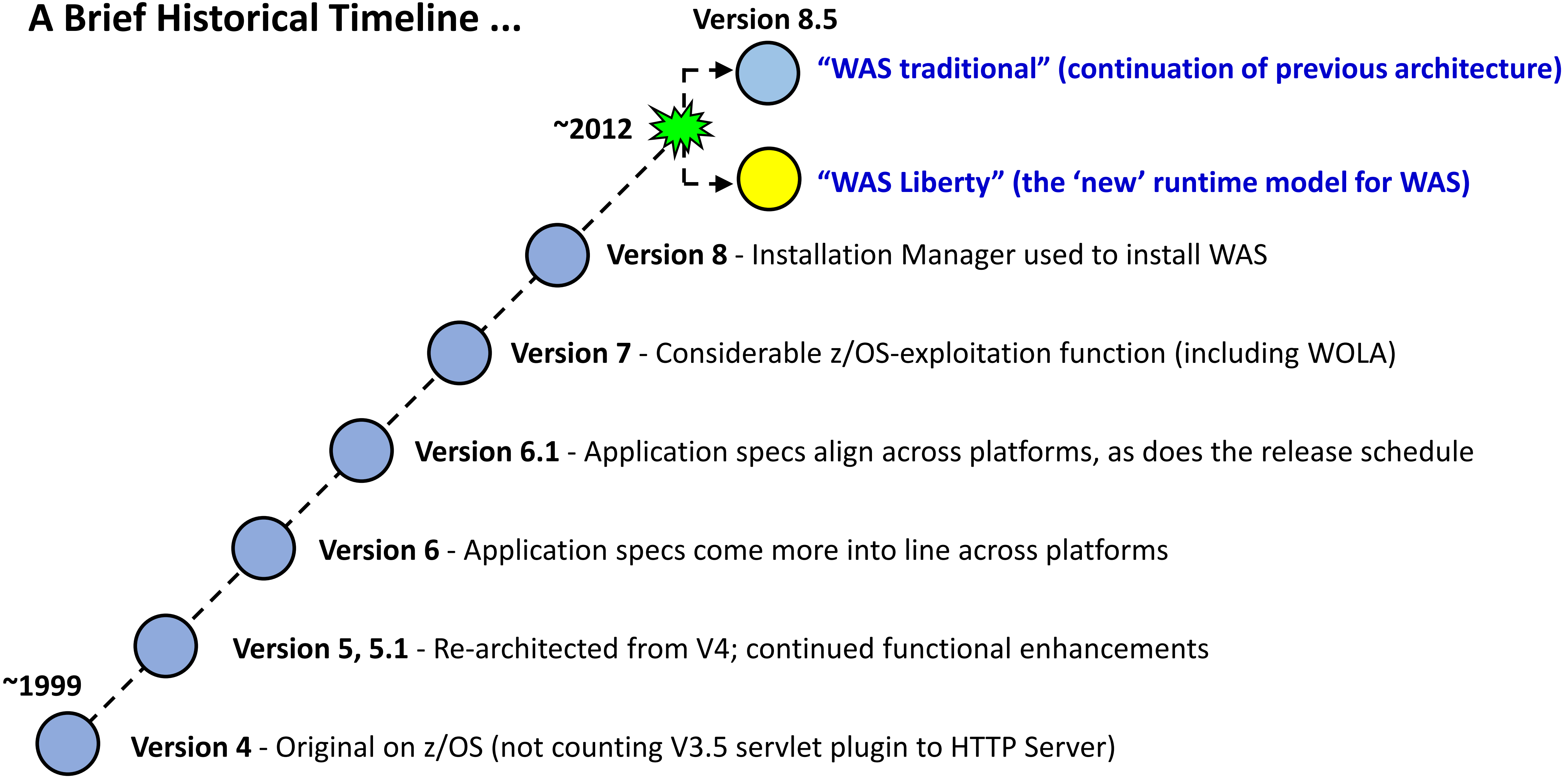
If there is a business driver to consider moving to Liberty, then:

- Determine the viability of moving the applications to Liberty
- Assess the operational differences and determine if any value is diminished by moving
- If value exceeds cost, then it’s a net benefit to the business and a move should be considered
- If cost exceeds value, then maintain WAS traditional for those applications
- Maintaining *both* environments is possible and would provide a “best of both worlds” environment

Setting Context

© 2016, IBM Corporation

A Brief Historical Timeline ...



Version 8.5

“WAS traditional” (continuation of previous architecture)

~2012

“WAS Liberty” (the ‘new’ runtime model for WAS)

Version 8 - Installation Manager used to install WAS

Version 7 - Considerable z/OS-exploitation function (including WOLA)

Version 6.1 - Application specs align across platforms, as does the release schedule

Version 6 - Application specs come more into line across platforms

Version 5, 5.1 - Re-architected from V4; continued functional enhancements

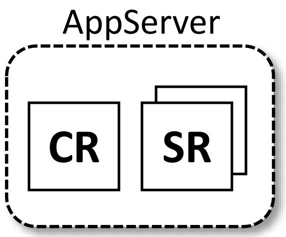
~1999

Version 4 - Original on z/OS (not counting V3.5 servlet plugin to HTTP Server)

© 2016, IBM Corporation

“WAS traditional” and “Liberty”

WAS traditional z/OS


AppServer

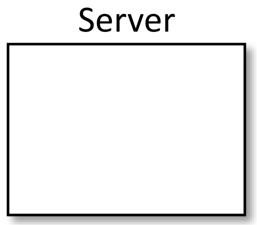
CR

SR

“Multi-JVM model”

- The original WAS, going back 15 years to Version 4
- On z/OS it consisted of “controllers” and “servants”
- It was organized into “nodes” and “cells”
- Specific function to exploit z/OS capabilities
- Considerable production-hardened investment here

Liberty z/OS


Server

- First introduced in V8.5.0.0 (2012) all platforms
- Single JVM server model (no CR/SR)
- Key attributes: lightweight, composable, dynamic
- Has z/OS exploitation functions
- Under “continuous development” = frequent updates

Both fall under “WebSphere Application Server” umbrella, but are not the same thing

(Which is why this positioning discussion is needed)

What was Behind Creation of Liberty?

WAS traditional ...	WAS Liberty ...
... loaded most functions even if applications did not require them	... is composable, allowing for customized function enablement
... required application and server restarts for most changes	... is dynamic, allowing for application and configuration changes without restarts
... Has a mature, but somewhat inflexible management model	... has a management model* that is, by design, flexible and highly scalable

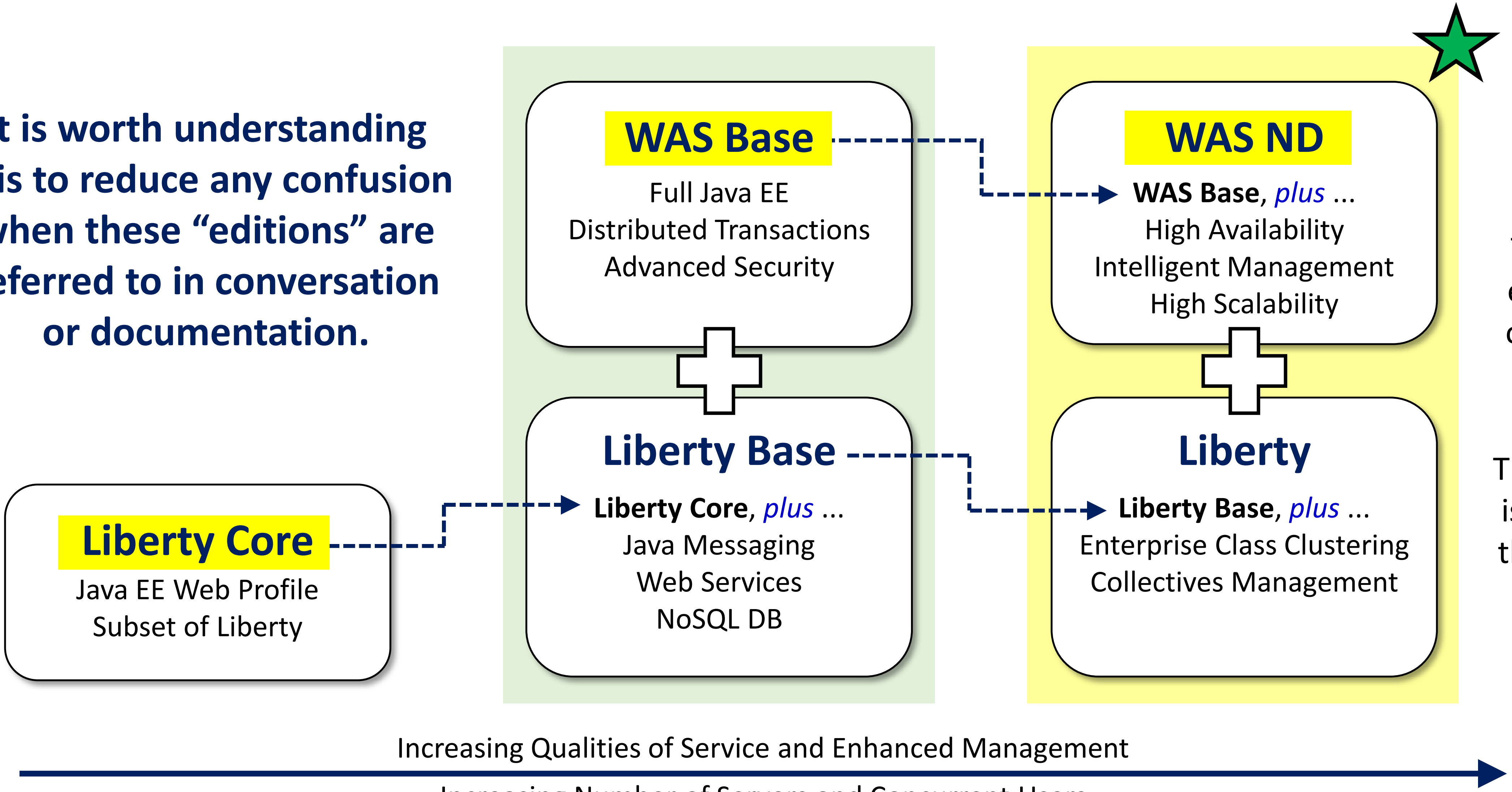
WAS traditional has its architectural roots going back 15 years. Times change, and a more flexible and dynamic server model was needed. That is Liberty.

* Called "Collectives". More on that later in the presentation.

© 2016, IBM Corporation

Understanding WAS Product Terminology

It is worth understanding this to reduce any confusion when these “editions” are referred to in conversation or documentation.



WAS Base
Full Java EE
Distributed Transactions
Advanced Security

WAS ND
WAS Base, *plus* ...
High Availability
Intelligent Management
High Scalability

Liberty Core
Java EE Web Profile
Subset of Liberty

Liberty Base
Liberty Core, *plus* ...
Java Messaging
Web Services
NoSQL DB

Liberty
Liberty Base, *plus* ...
Enterprise Class Clustering
Collectives Management

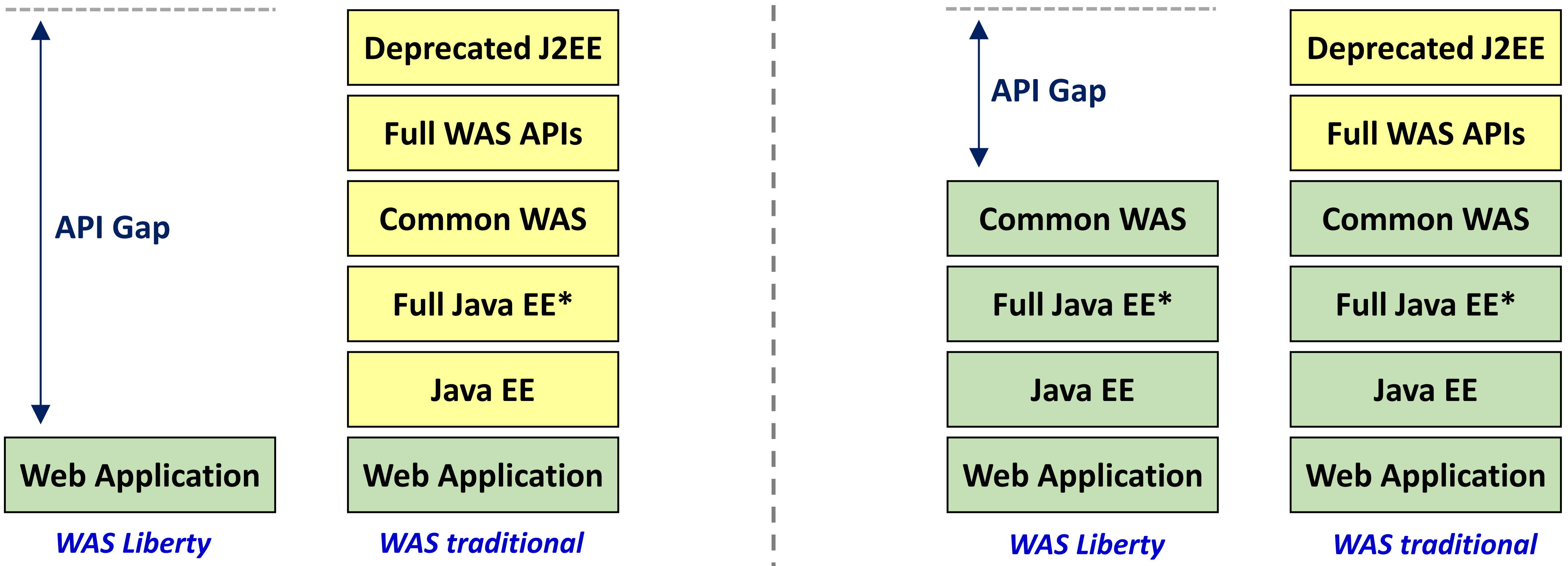
Increasing Qualities of Service and Enhanced Management

Increasing Number of Servers and Concurrent Users

The WAS ND edition is the only one that is supported on z/OS

Therefore, that is the focus of this document

Differences in the Application Programming Interface (APIs)



8.5.0.0

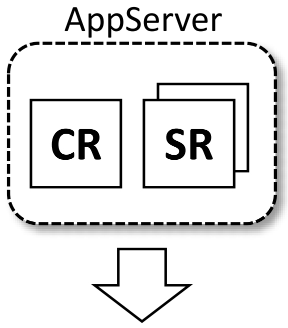
8.5.5.9

Initially the gap was large, and some existing WAS traditional applications could not run on Liberty. Now, many (if not most) can run on Liberty with little or no changes.

* For Liberty: partial Java EE 6, full Java EE 7. For WAS traditional: Full Java EE 6, Full Java EE 7 in beta

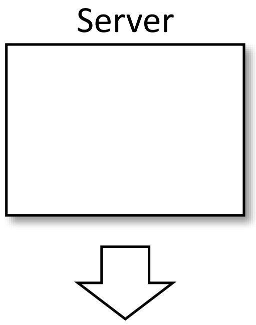
Greater zIIP Offload and Lower Cost

WAS traditional z/OS



~ 80% or perhaps higher offload

Liberty z/OS



~ 90% or perhaps higher offload

Many “it depends” qualifiers around these numbers

In general: WAS traditional has a greater degree of native code (not eligible for zIIP offload) supporting the Java runtime than does Liberty

Best way to determine offload difference is to benchmark specific application

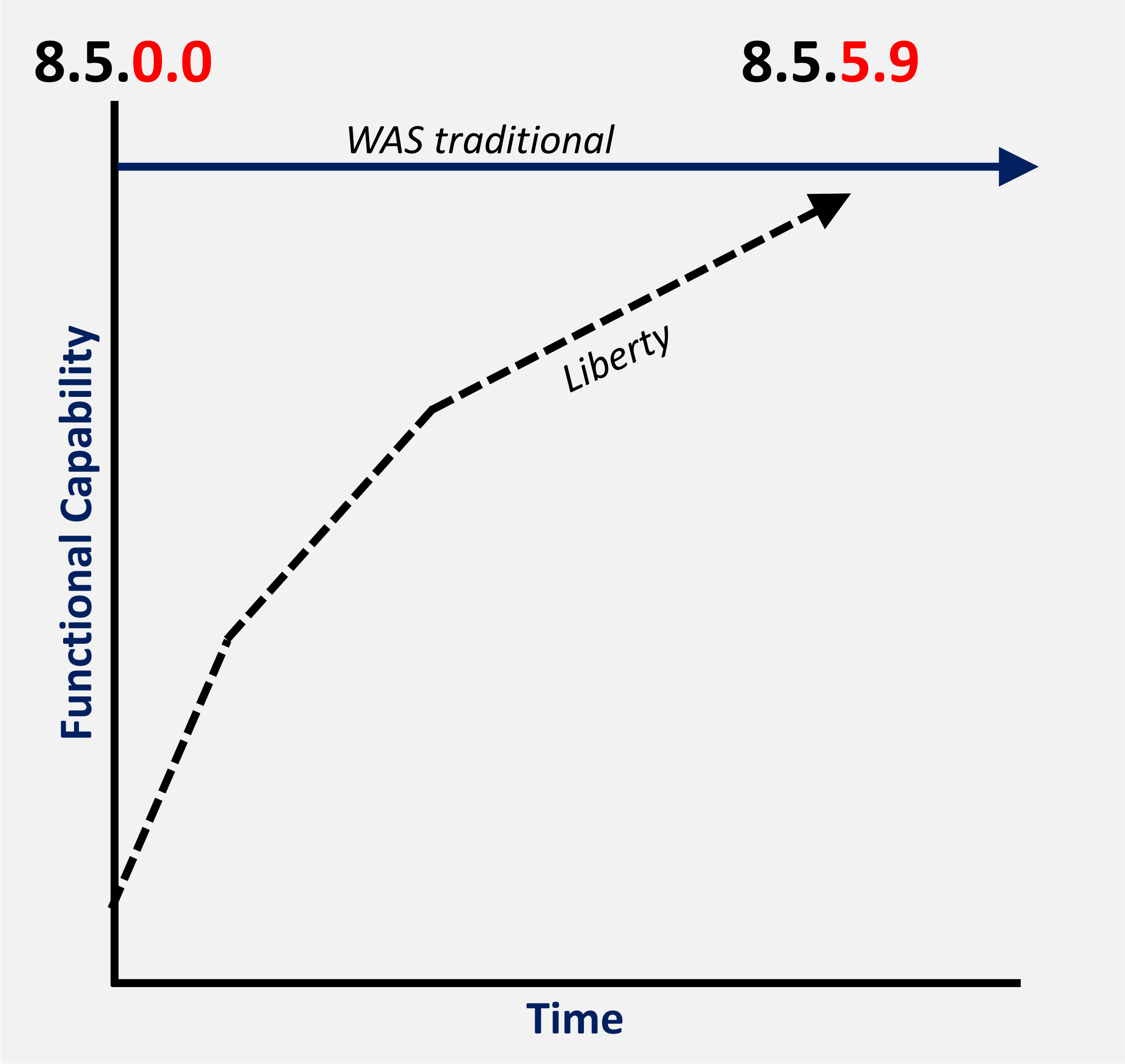
In additional to the greater zIIP offload potential, it is possible the same workload running in Liberty would require fewer Value Unit Entitlements (VUEs) and thus imply a lower One Time Charge (OTC) cost.

Using Liberty z/OS with zCAP pricing could provide a very cost-effective solution for new Java workloads on z/OS -- even when compared across all platforms.

Potential exists for very attractive cost model for Java on z/OS

Consult with your IBM sales representative for specific details about pricing

Differences in the Management Models



- **WAS traditional management model is mature and functionally stable**
- **Initial Liberty management model was lacking in functional capabilities**
- **Investment focus has shifted to Liberty and its management model**
- **Investment also being made in dev/ops flows for Liberty**

The models are different, so a direct comparison is difficult. Key point: Liberty has advanced considerably since 8.5.0.0 and management model is far more feature-rich than it was at first.

When we Speak of “Operational Considerations,” we Refer to the Following ...

- Product installation
- Product maintenance updates
- Runtime creation
- Runtime provisioning (dev/ops, cloud, containers)
- Runtime configuration changes
- Runtime updates to new versions
- Application deployments / updates
- Backup and restore
- Capacity and performance monitoring
- Troubleshooting and problem tracking
- Usage monitoring and chargeback
- System automation routines

... and other activities

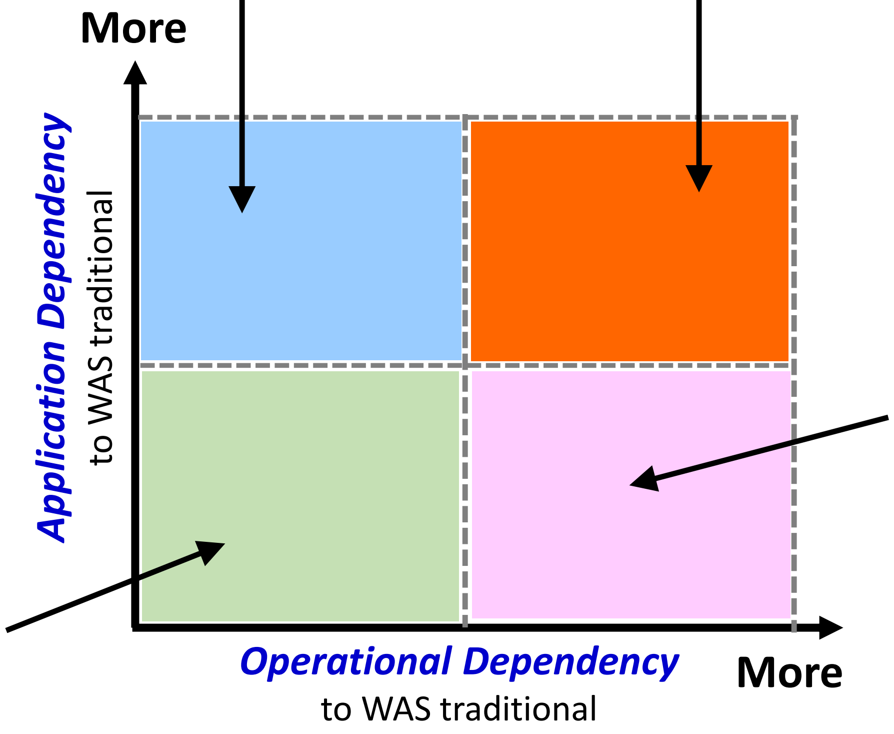
These activities are, to varying degrees, important to the business

The discussion here is how deeply invested you are in tools and processes for these activities *today*, and how easily can you move to a Liberty runtime platform *tomorrow*

A High-Level Framework for Evaluating Existing Workload for Move to Liberty

High Application / Low Operational

- Applications have dependencies
- Little or no script investment
- Investment in Liberty skills in plan
- **Consider Liberty for new workloads**
- **Investigate application re-engineering for cases where move to Liberty is justified**



High Application / High Operational

- Applications not easily moved
- Vendor application dependencies
- Investment in WADMIN scripts
- Deep skills in WAS traditional Admin
- **Maintain WAS traditional**
- **Consider Liberty for new workloads**

Low Application / High Operational

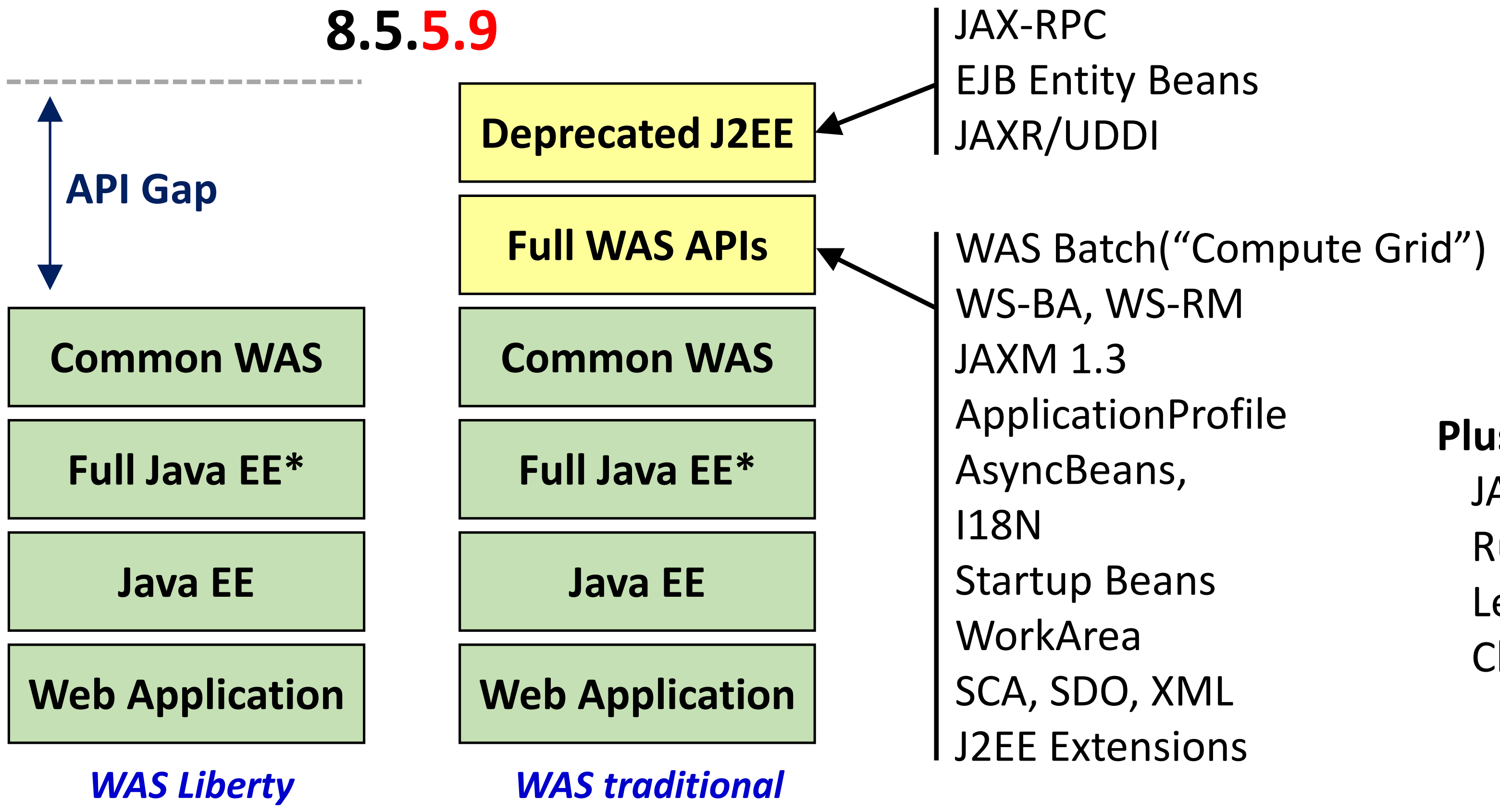
- Little or no application dependencies
- Investment in WSADMIN scripts
- Deep skills in WAS traditional Admin
- **Maintain WAS traditional for existing**
- **Consider Liberty for new workloads**

Low Application / Low Operational

- Little or no application dependencies
- Little or no script or skill investment
- **Consider Liberty for existing and new workloads**

Application Considerations

More on the API Gap between Liberty and WAS traditional



8.5.5.9

API Gap

WAS Liberty	WAS traditional
	Deprecated J2EE
	Full WAS APIs
Common WAS	Common WAS
Full Java EE*	Full Java EE*
Java EE	Java EE
Web Application	Web Application

Deprecated J2EE:

JAX-RPC
EJB Entity Beans
JAXR/UDDI

Full WAS APIs:

WAS Batch(“Compute Grid”)
WS-BA, WS-RM
JAXM 1.3
ApplicationProfile
AsyncBeans,
I18N
Startup Beans
WorkArea
SCA, SDO, XML
J2EE Extensions

Plus….

JAX-WS stacks are different
Runtime class visibility is different
Less EJB/IIOP QOS in Liberty
Client code may be different

An application that makes use of the APIs in the “API Gap” list may need re-engineering to move to Liberty. If the application uses APIs that are common across WAS traditional and Liberty, then it may move easily.

* For Liberty: partial Java EE 6, full Java EE 7. For WAS traditional: Full Java EE 6, Full Java EE 7 in beta

Considerations Beyond the APIs

Time horizon for application ------------

An application with a relatively short life horizon may not be worth moving. Better to leave it where it is and focus energy on higher-value applications

Value of application investment ------------

An application with a longer expected life span may require re-engineering investment to run properly on Liberty. Does the proposed investment yield positive return for the business?

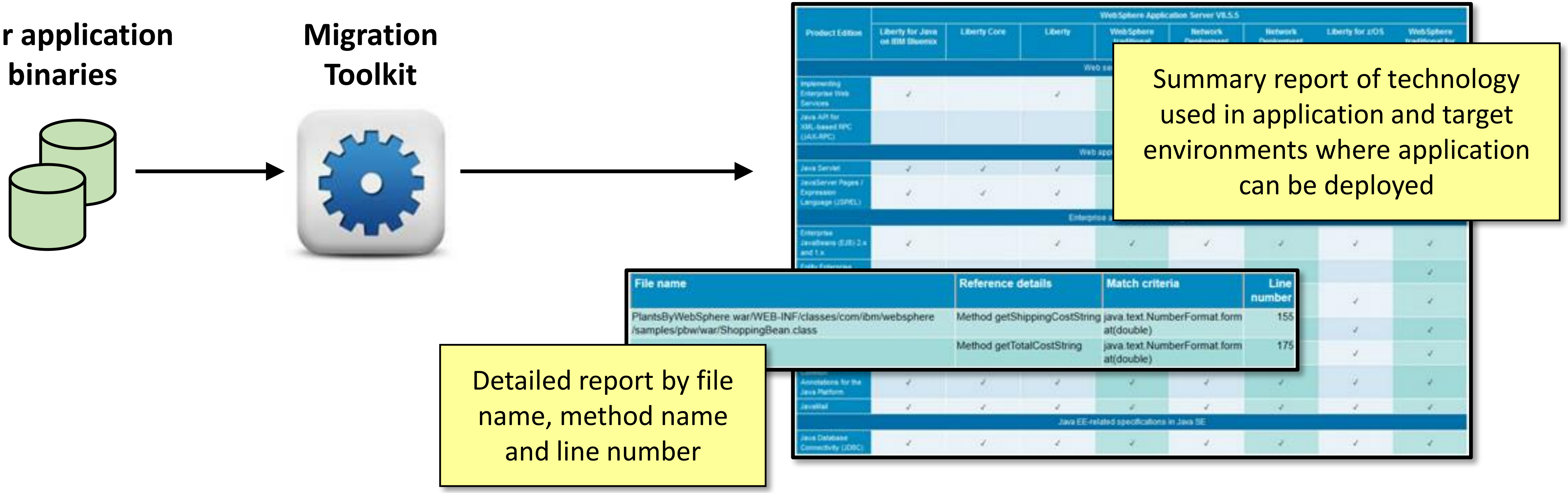
Potential deployment environments --

For new applications, do you expect to deploy the application into environments such as IaaS cloud, or Bluemix, or container environments such as Docker? That may imply targeting Liberty as that runtime is better prepared for operations in those environments.

Migration Toolkit for Application Binaries

Your application binaries

Migration Toolkit


WebSphere Application Server V8.5.5
Product Edition
Liberty for Java on IBM Bluemix
Liberty Core
Liberty
WebSphere traditional
Network Deployment
Network Deployment
Liberty for z/OS
WebSphere traditional for
Web se
Implementing Enterprise Web Services
Java API for XML-based RPC (JAX-RPC)
Web app
Java Servlet
JavaServer Pages / Expression Language (JSP/EL)
Enterprise
Enterprise JavaBeans (EJB) 2.x and 1.x
Entity Enterprise
Annotations for the Java Platform
JavaMail
Java EE-related specifications in Java SE
Java Database Connectivity (JDBC)
✓

File name	Reference details	Match criteria	Line number
PlantsByWebSphere.war/WEB-INF/classes/com/ibm/websphere/samples/pbw/war/ShoppingBean.class	Method getShippingCostString	java.text.NumberFormat.format(double)	155
	Method getTotalCostString	java.text.NumberFormat.format(double)	175

Summary report of technology used in application and target environments where application can be deployed

Detailed report by file name, method name and line number

Main wasDev page:

https://developer.ibm.com/wasdev/downloads/#asset/tools-Migration_Toolkit_for_Application_Binaries

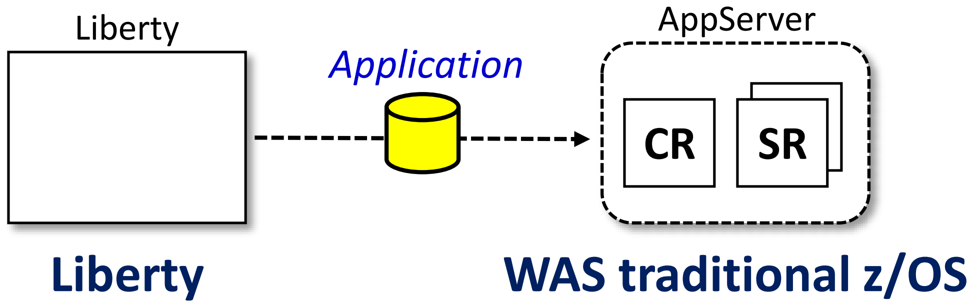
Technical Overview:

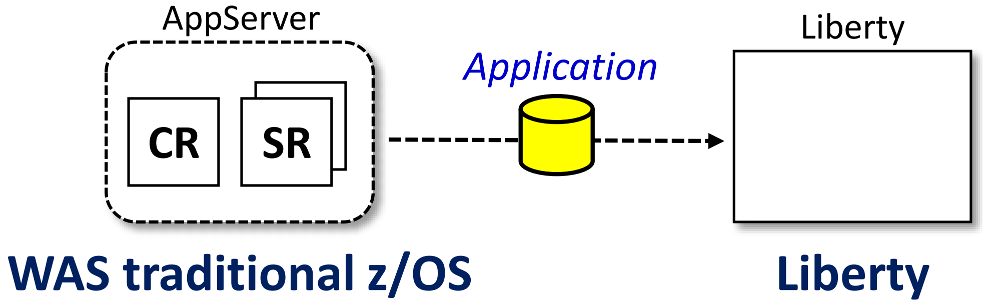
https://developer.ibm.com/wasdev/docs/migration-toolkit-application-binaries-tech/

Updates page:

https://developer.ibm.com/wasdev/blog/2015/03/13/announcing-websphere-liberty-migration-tools-updates/

Final Points on Application Considerations





This application path is relatively seamless

Notes:

- Liberty has Java EE 7, WAS traditional is in beta with that technology. An application that makes specific use of Java EE 7 (ex: JSR 352 Java Batch) would not work on WAS traditional if Java EE 7 not present.
- Liberty is a single JVM environment, where WAS traditional on z/OS has the potential for multiple application JVMs (SRs). Applications that create singletons *may* experience issues.

This path can work, but a bit more care needed

Notes:

- If application uses APIs in the "API Gap" illustrated earlier, the application would require updating.
- If the application is relying on session replication between SRs, that aspect of the application would need inspection and persistence (if needed) configured in Liberty using a database or caching layer.

Operational Considerations

Broad Topic with Many Disciplines

Install and Maintain

- Product installations
- Maintenance updates
- Create runtimes
- Migrate to new versions
- Backup and restore

Plan, Monitor, Troubleshoot

- Capacity planning
- Performance planning
- Monitoring usage, resources, performance
- Analyze problems, track resolution

Change Management

- Identify change requirements
- Implement and test
- Promote up to production
- Track progress, effect back-outs

Develop, Deploy, and Test

- Application design and develop
- Deployment automation
- Deployment target provisioning
- Test planning and automation
- Other Dev/Ops activities

Other?

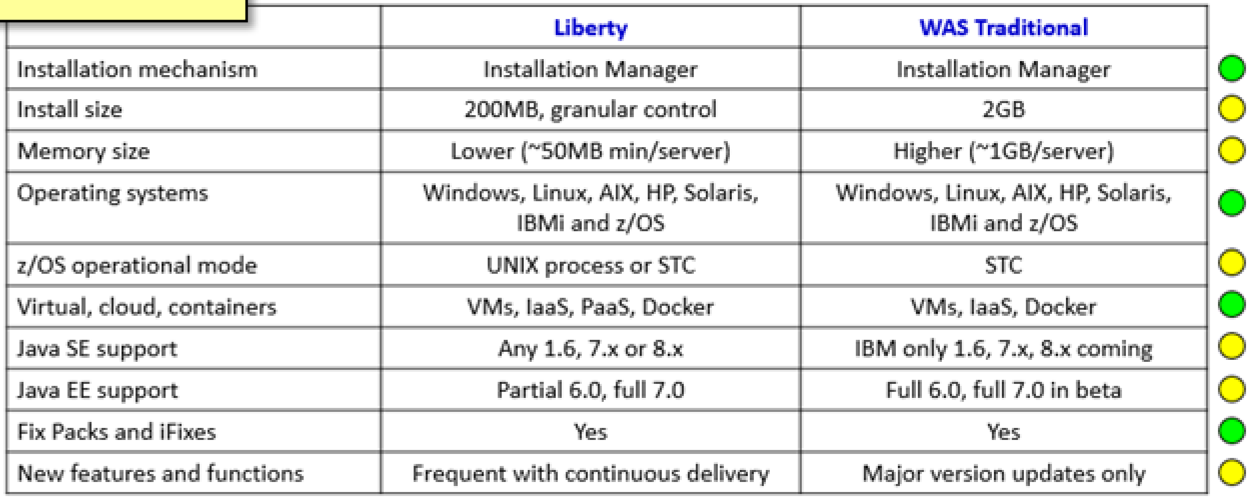
- Any other operational activities not on the lists above

© 2016, IBM Corporation

Comparison Grids to Follow

Operational attribute or task

Runtime, Liberty or WAS traditional



	Liberty	WAS Traditional	
Installation mechanism	Installation Manager	Installation Manager	🟢
Install size	200MB, granular control	2GB	🟡
Memory size	Lower (~50MB min/server)	Higher (~1GB/server)	🟡
Operating systems	Windows, Linux, AIX, HP, Solaris, IBMi and z/OS	Windows, Linux, AIX, HP, Solaris, IBMi and z/OS	🟢
z/OS operational mode	UNIX process or STC	STC	🟡
Virtual, cloud, containers	VMs, IaaS, PaaS, Docker	VMs, IaaS, Docker	🟢
Java SE support	Any 1.6, 7.x or 8.x	IBM only 1.6, 7.x, 8.x coming	🟡
Java EE support	Partial 6.0, full 7.0	Full 6.0, full 7.0 in beta	🟡
Fix Packs and iFixes	Yes	Yes	🟢
New features and functions	Frequent with continuous delivery	Major version updates only	🟡

Green = same
Yellow = delta

By walking through the operational attributes it has the potential to stimulate thinking and discussion about your current environment compared to Liberty. We encourage the discussion. The objective is a clear understanding of the similarities and differences.

© 2016, IBM Corporation

General Product Considerations

	Liberty	WAS traditional	
Installation mechanism	Installation Manager	Installation Manager	🟢
Install size	200MB, granular control	2GB	🟡
Memory size	Lower (~50MB min/server)	Higher (~1GB/server)	🟡
Operating systems	Windows, Linux, AIX, HP, Solaris, IBMi and z/OS	Windows, Linux, AIX, HP, Solaris, IBMi and z/OS	🟢
z/OS operational mode	UNIX process or STC	STC	🟡
Virtual, cloud, containers	VMs, IaaS, PaaS, Docker	VMs, IaaS, Docker	🟢
Java SE support	Any 1.6, 7.x or 8.x	IBM only 1.6, 7.x, 8.x coming	🟡
Java EE support	Partial 6.0, full 7.0	Full 6.0, full 7.0 in beta	🟡
Fix Packs and iFixes	Yes	Yes	🟢
New features and functions	Frequent with continuous delivery	Major version updates only	🟡

Configuration and Deployment

	Liberty	WAS traditional	
Composable runtime	Yes (via Features)	No	🟡
Dynamic configuration	Yes	Partial	🟡
Configuration structure	Relatively simple, flexible location	More complex, defined location	🟡
Configuration editing	Simple XML updates; admin tools	Admin console; WSADMIN scripting	🟡
Configuration updates	Simple file-based	XML file deltas via tools	🟡
Central management	Collectives (no agents)	Cell (with node agents)	🟡
Central management scale	Very small to 10,000+	Very small to ~700 maximum	🟡
Central management failover	Yes (controller replica)	No (restart DMGR on other LPAR)	🟡
Configuration ownership	Each server (no synchronization)	DMGR (central with synchronization)	🟡
Application deployment	Manual, script, with server package	Admin Console, WSADMIN script	🟡
Application update	Replace application file	Redeploy through Admin	🟡
Product update	No migration	Migration tools	🟡

Operational Capabilities

	Liberty	WAS traditional	
HTTP load balancing	Plugin, ODRLIB, any HTTP proxy	Same as Liberty, plus Java ODR	🟡
HTTP session replication	DB persistence or WXS caching	Same as Liberty, plus DRS	🟡
Scripting support	Any	WSADMIN (JACL or Jython)	🟡
Dynamic clusters / auto-scale	Yes	Yes	🟢
JMX client	Java, REST	WAS Admin Client	🟡
Monitoring	mBeans, PMI	PMI	🟡
Fine-grained admin authority	No (single admin role)	Yes	🟡
JMS providers	Internal, WMQ, 3rd Party	Internal, WMQ, 3rd Party	🟢
Clustered JMS provider	No (use WMQ)	Yes	🟡
2PC transaction recovery	Yes	Yes	🟢
Remote EJB calls	Yes	Yes	🟢
Runtime class visibility	Defined API	Internals are accessible	🟡
Docker support	Yes (collective support in beta)	Yes	🟢

© 2016, IBM Corporation

Security Options (1 of 2)

	Liberty	WAS traditional	
Default passwords	No	No	🟢
Minimal ports opened	Yes	No	🟡
Secured remote admin	Yes (mandatory)	Yes (but can be turned off)	🟢
File user registry	Yes (server.xml)	Yes (file based)	🟢
Federated LDAP or SAF	Yes	Yes	🟢
OAuth, OpenID, OIDC client	Yes	Yes	🟢
OIDC server/provider	Yes	No	🟡
LTPA, SPNEGO tokens	Yes	Yes	🟢
SAML Web SSO	Yes	Yes	🟢
SAML Web Services	Yes	Yes	🟢
User and Group API	Yes	Yes	🟢
Federated File registry w/ LDAP	Yes	Yes	🟢

© 2016, IBM Corporation

Security Options (2 of 2)

	Liberty	WAS traditional	
Auditing	No	Yes	🟡
Advanced key/cert management	Yes	Yes	🟢
Local OS registry	No (yes if z/OS = SAF)	Yes	🟡
JAX-WS support for LTPA	No	Yes	🟡
JSEEHelper API	No	Yes	🟡

© 2016, IBM Corporation

z/OS Integration and Platform Exploitation

	Liberty	WAS traditional	
Multi-JVM (CR/SR)	No	Yes	🟡
z/OS Connect	Yes	No	🟡
zWLM	Yes (Service and Report classification)	Same, and work placement by SC	🟡
WOLA local adapters	Yes (no 2PC yet)	Yes	🟢
RRS TX coordination	Yes (JDBC only)	Yes	🟡
SMF request tracking	Yes (HTTP only)	Yes	🟢
Messages to server job log	Yes	Yes	🟢
Messages redirect to console	Yes	Yes	🟢
Hung thread stop and recover	No	Yes	🟡
Pause/Resume Listeners	No	Yes	🟡
Dispatch Progress Monitor	Yes (with Health Manager feature)	Yes	🟢
MODIFY interface	Yes, but limited	Yes	🟡

Summary of z/OS Operational Considerations

Install and backup/restore are somewhat similar for both

Liberty requires no migration tools to move to new version, WAS traditional does, and the effort to migrate is not trivial

Administrative interfaces are different; scripting interfaces are different

Both are operated as started tasks, so:

- Can use system automation routines
- Can monitor with SMF Type 30

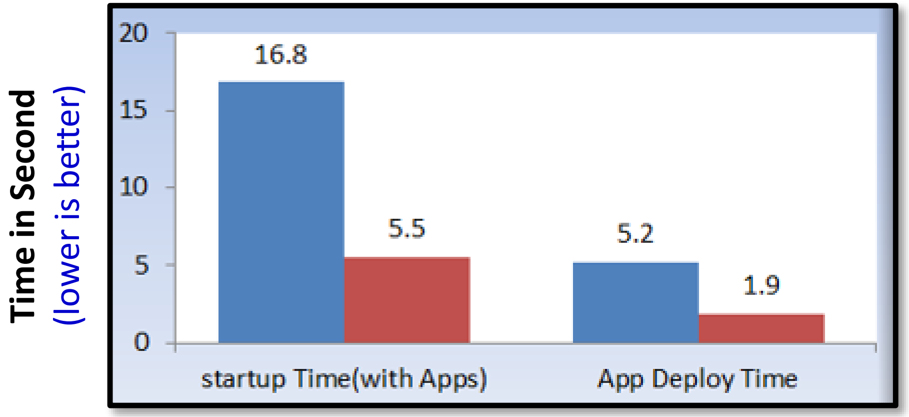
Both are capable of WLM service class and report classification based on matching request URI patterns

WAS traditional has deeper z/OS integration functions, but if that’s not something you’re making use of, then it’s less a factor

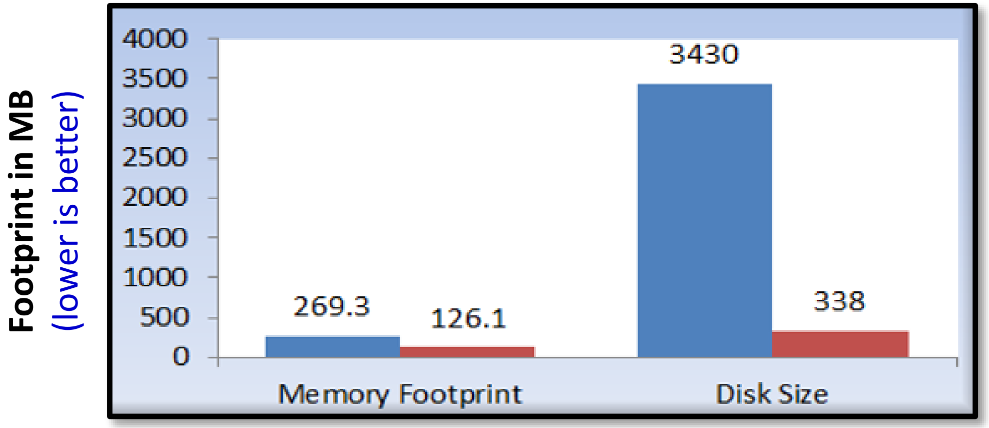
Performance Considerations

Startup Time, App Deploy Time, and Memory/Disk Footprint

🟦 **WAS traditional** 🟥 **Liberty**



Startup Time, App Deploy Time



Memory Footprint, Disk Size

Startup time for Liberty 32% the time of WAS traditional

Application deployment time 36% the time of WAS traditional

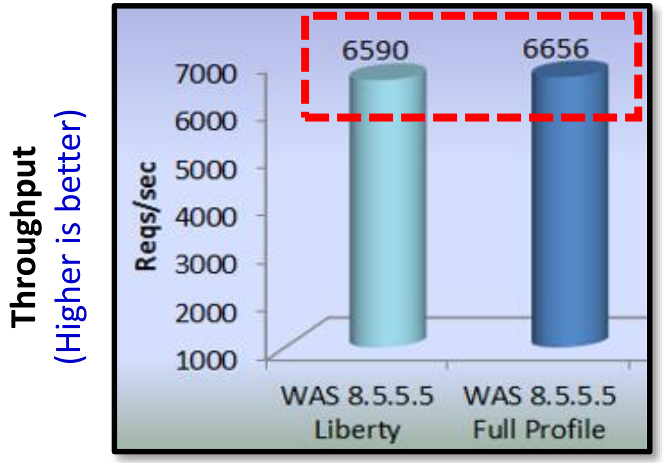
Memory footprint for Liberty 47% that of WAS traditional

Disk size for Liberty 10% that of WAS traditional

Performance results derived in a controlled environment under specific conditions. Your results may vary depending on a number of factors.

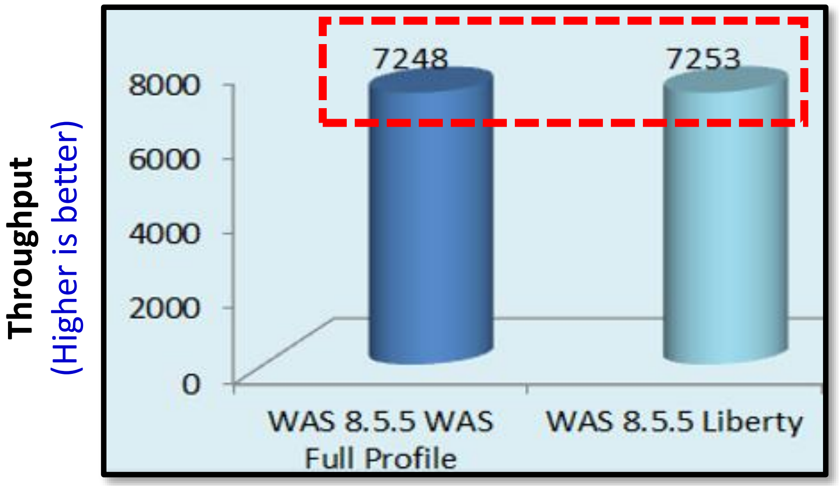
Throughput on Distributed Platforms … z/OS on Next Chart

DayTrader 3 EJB, Hotspot JDK 8_31



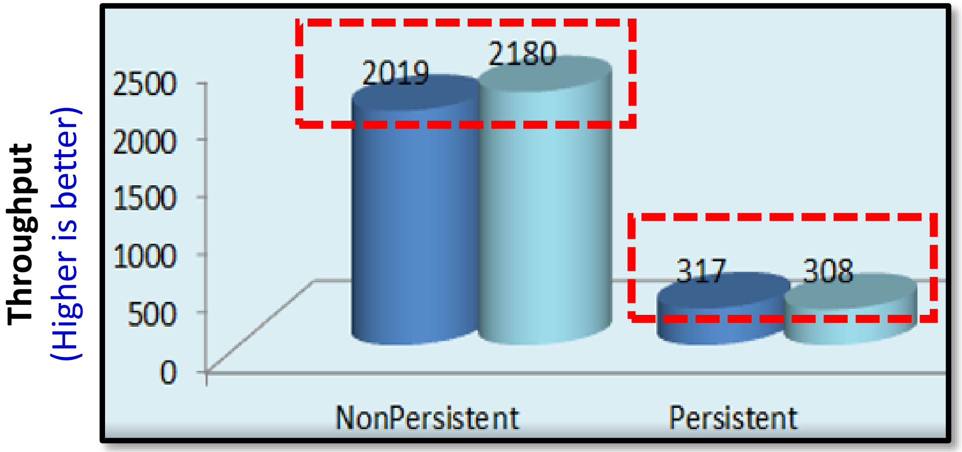
Liberty 99% of WAS traditional

Web Services SOABench



Liberty 100% of WAS traditional

Messaging, JMS Prims 10k/10k



Liberty 108% of WAS traditional

Liberty 97% of WAS traditional

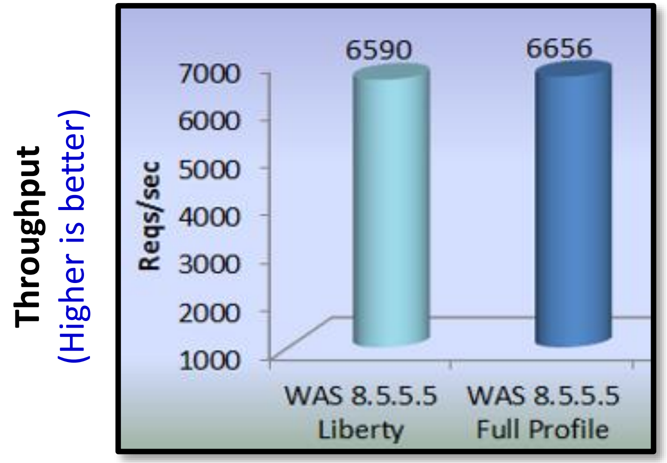
Effectively the same throughput for WAS traditional and Liberty on the distributed platforms for DayTrader (EJB), SOABench (SOAP/WSDL), and Messaging (JMS)

No loss of throughput moving from WAS traditional to Liberty on distributed

Performance results derived in a controlled environment under specific conditions. Your results may vary depending on a number of factors.

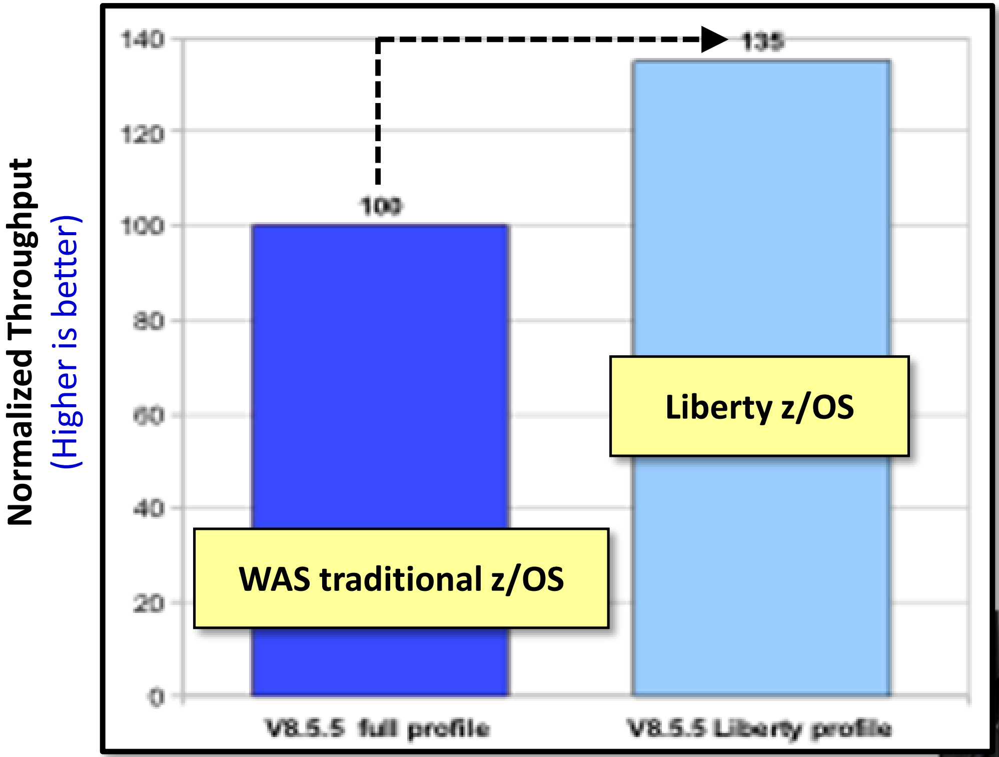
DayTrader 3 on z/OS Shows Liberty Outperforming WAS traditional

DayTrader 3 EJB



Distributed
(from previous chart)

DayTrader 3 EJB



Note: the throughput axis for z/OS shows results normalized ... that is, the WAS traditional throughput achieved was set to "100" and the Liberty throughput achieved was proportional to the baseline 100 value.

Actual throughput is a function of many factors, including processor speed, memory, cache size, and I/O.

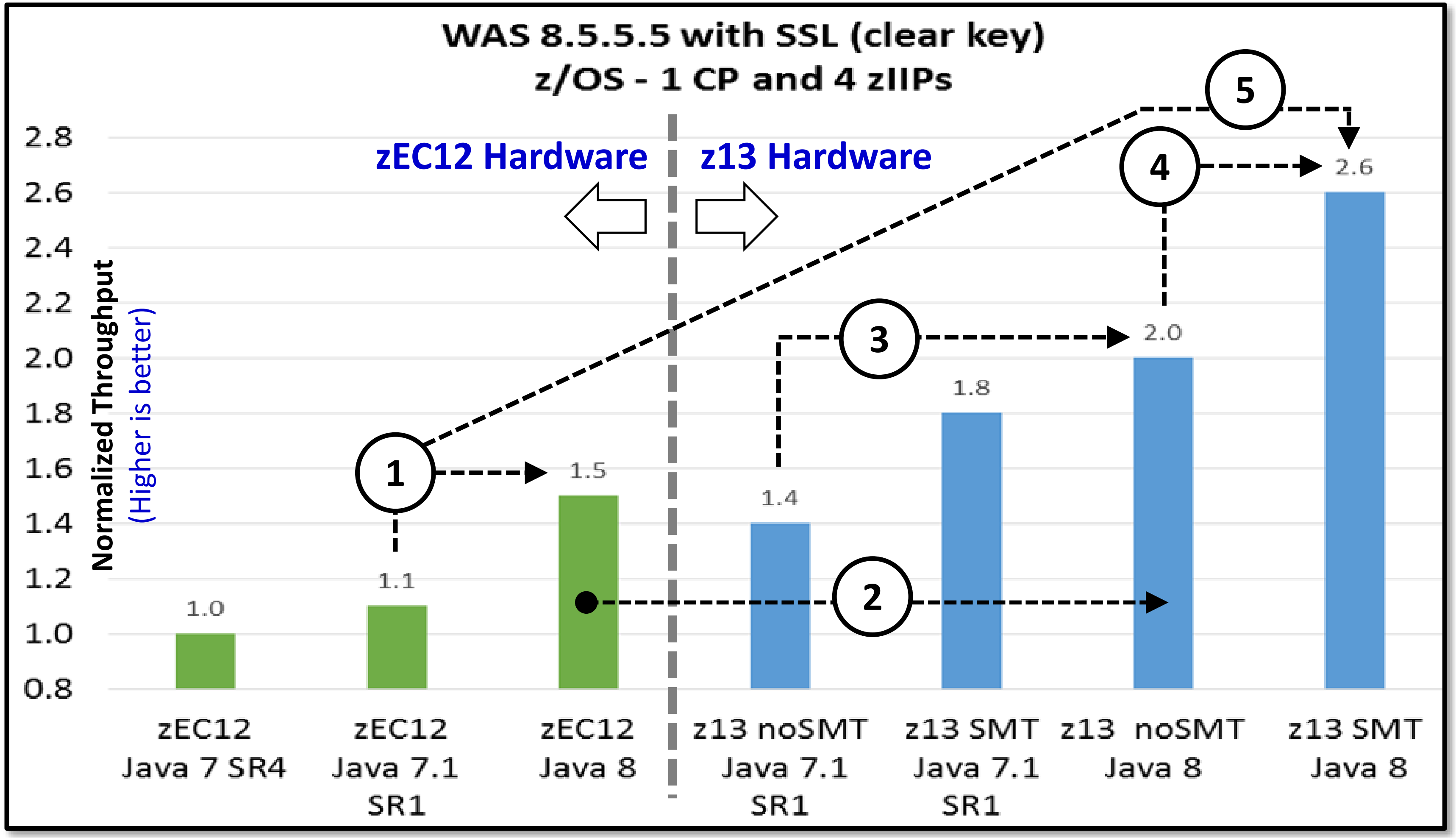
The tests performed here were not meant to compare distributed directly with z/OS. Rather, the point here is that on z/OS, Liberty outperformed WAS traditional. On distributed, the two were roughly equivalent.

This is because Liberty's single-JVM model is more efficient than WAS traditional's multi-JVM model with controller and servant regions

Performance results derived in a controlled environment under specific conditions. Your results may vary depending on a number of factors.

The Value of z13 Hardware, Java 8 and SMT Exploitation

SSL-Enabled DayTrader 3.0 with Liberty z/OS measured



1. **Java 8 on zEC12**
   36% improvement -- improved JVM/JIT
   (1.5/1.1 = 1.36)

2. **Value of z13**
   33% improvement -- faster HW, greater instruction exploitation by SDK
   (2.0/1.5 = 1.33)

3. **Java 8 on z13**
   43% improvement -- improved JVM/JIT, greater instruction exploitation by SDK
   (2.0/1.4 = 1.43)

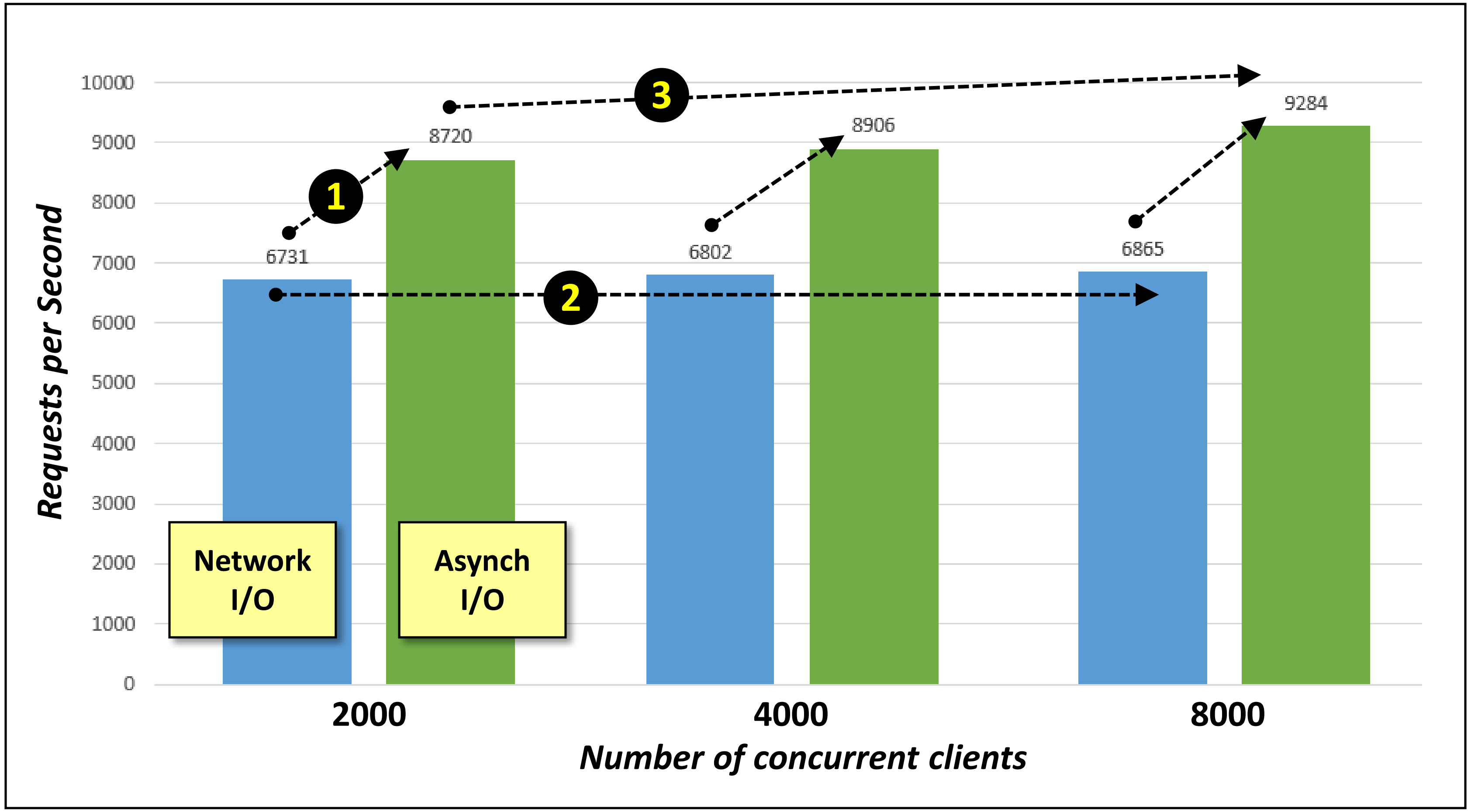
4. **Value of SMT**
   30% improvement -- exploitation of SMT by Java 8 SDK
   (2.6/2.0 = 1.30)

5. **Overall**
   Java level, HW level, and SMT. We see a 136% improvement
   (2.6/1.1 = 2.36)

Performance results derived in a controlled environment under specific conditions. Your results may vary depending on a number of factors.

Asynchronous v. Network I/O in Liberty z/OS 16.0.0.3

Asynchronous I/O performance benefits are most significant with larger numbers of concurrent clients:

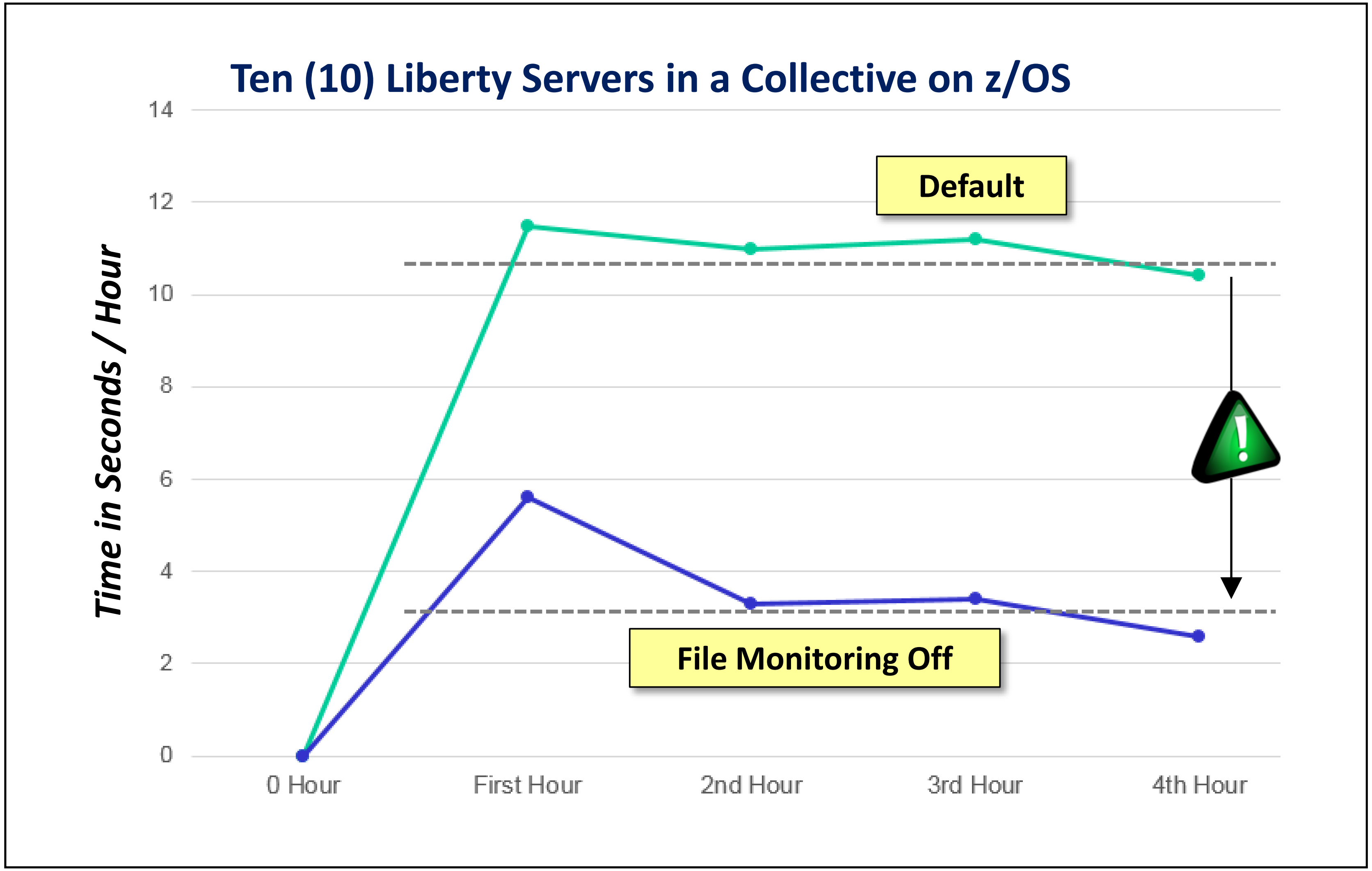


Three key points:

1. **Asynch I/O > Network I/O**
   In all three concurrent user scenarios, Asynch I/O was 30% or more greater throughput
   2000 concurrent = +30%
   4000 concurrent = +31%
   8000 concurrent = +35%
2. **Network I/O mostly flat**
   As concurrent users scale up, we see a relatively flat line for Network I/O (~1.9% improvement 2K to 8K)
3. **Asynch I/O trends up**
   As concurrent users scale up, we see a trend upwards with Asynch I/O (~6.5 improvement 2K to 8K)

Performance results derived in a controlled environment under specific conditions. Your results may vary depending on a number of factors.

Idle CPU time in Liberty on z/OS



This chart is showing the CPU time for 10 Liberty z/OS servers in a Collective as they idle

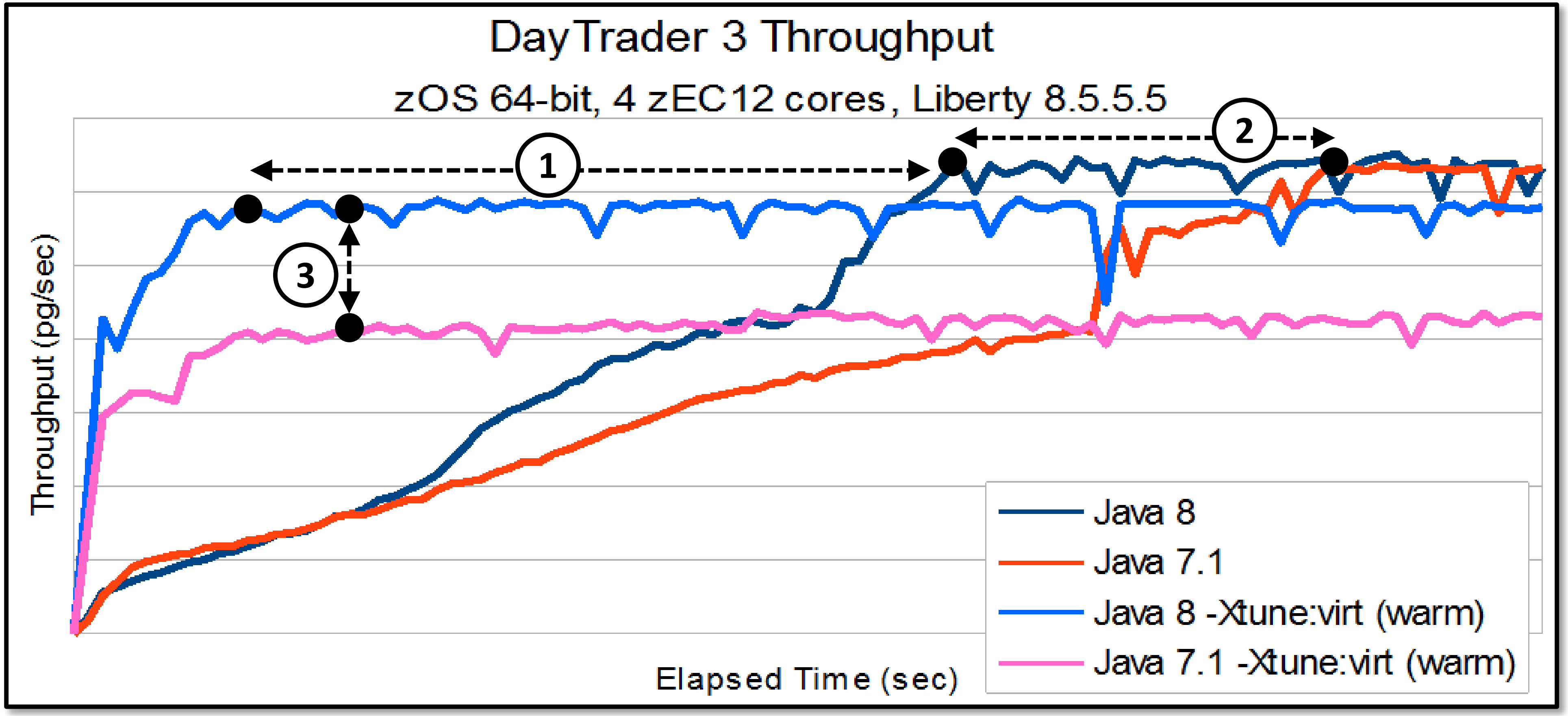
The Y Axis shows the CPU time in seconds for all 10 servers at each hour mark (the X Axis).

When configured with the default file monitoring setting, the environment averaged about 11 CPU seconds per hour for the 10 servers.

When file monitoring is turned off, the CPU time dropped to about 3 seconds total per hour for the 10 servers.

Performance results derived in a controlled environment under specific conditions. Your results may vary depending on a number of factors.

z/OS Liberty Ramp-up with IBM Java 8



1

Ramp-up improvement due to -Xtune:virt

Less elapsed time to steady state when -Xtune:virt used

2

Ramp-up improvement Java 8 vs. Java 7

Java 8 achieved steady state in less elapsed time than Java 7

3 **Steady-state throughput improvement Java 8 over Java 7 with -Xtune:virt**

Once steady state is achieved, Java 8 results in better throughput

Performance results derived in a controlled environment under specific conditions. Your results may vary depending on a number of factors.

Startup footprint : WAS traditional ND on z/OS vs. Liberty z/OS

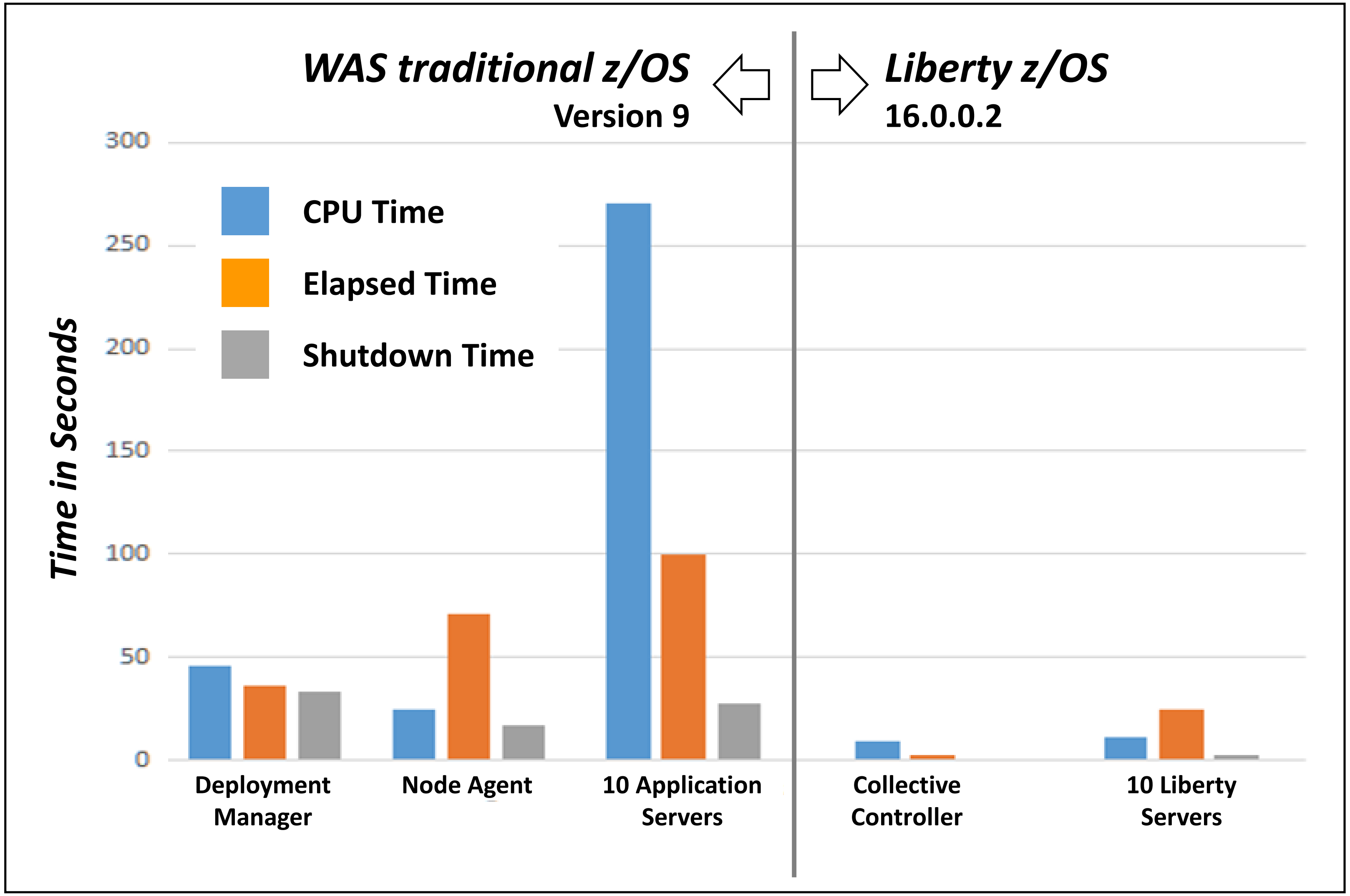
WAS traditional Network Deployment on zEC12			
Process Name	CPU Time (seconds)	Elapsed Time (seconds)	Memory (MB)
DMGR CR	15.96	32	306.4
DMGR SR	20.01	13	398.0
Node Agent	11.39	72	224.0
Member1 CR	10.30	19	239.2
Member1 SR	7.58	7	256.4
Member2 CR	10.20	19	241.6
Member2 SR	7.56	7	259.6
Total	**83**	**169**	**1925.2**

Liberty Collectives on zEC12			
Process Name	CPU Time (seconds)	Elapsed Time (seconds)	Memory (MB)
Controller	9.62	2.3	153
Member1	5.96	1.7	138
Member2	5.14	1.9	141
Total	**20.72**	**5.9**	**432**

Liberty involves fewer processes to create a two-member cluster, and the design of Liberty provides a smaller footprint and faster startup. The results bear this out.

Performance results derived in a controlled environment under specific conditions. Your results may vary depending on a number of factors.

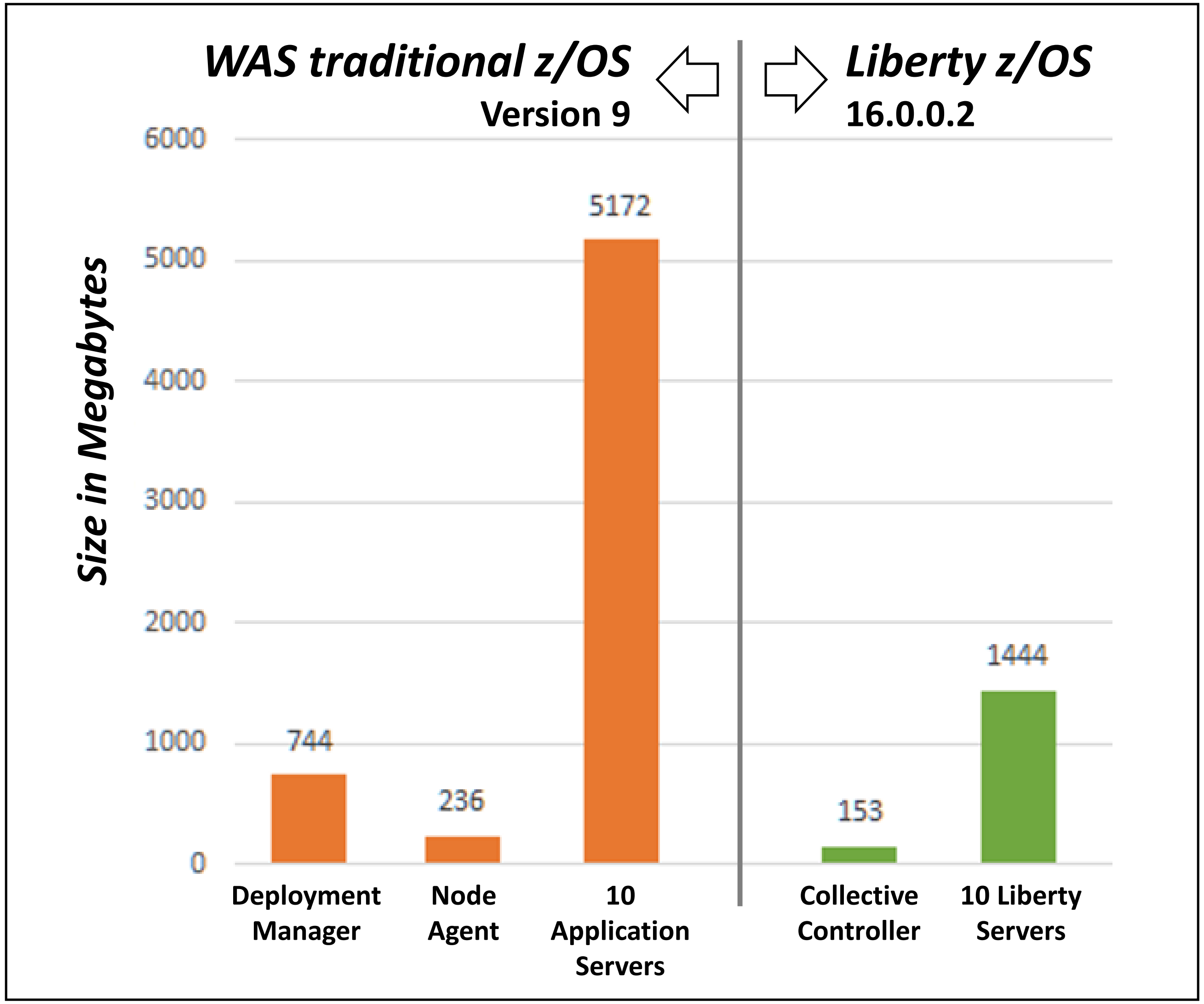
Startup and Shutdown Times: WAS traditional vs. Liberty



Start-up and shutdown of 10 servers in a Liberty Collective is significantly faster and more efficient.

Performance results derived in a controlled environment under specific conditions. Your results may vary depending on a number of factors.

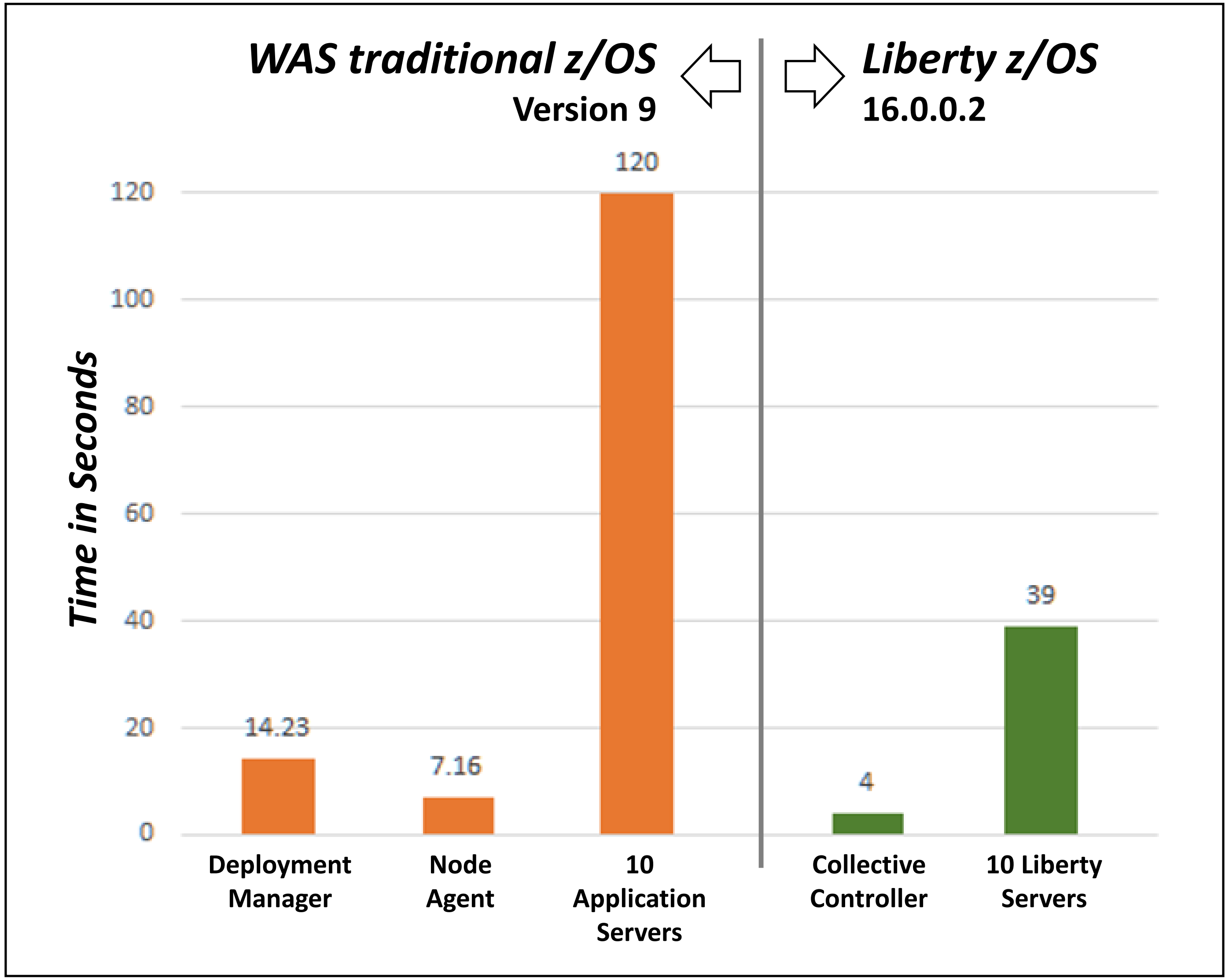
Memory Footprint: WAS traditional vs. Liberty



Memory footprint for 10 Liberty servers is almost 5 times less compare to 10 WAS traditional servers.

Performance results derived in a controlled environment under specific conditions. Your results may vary depending on a number of factors.

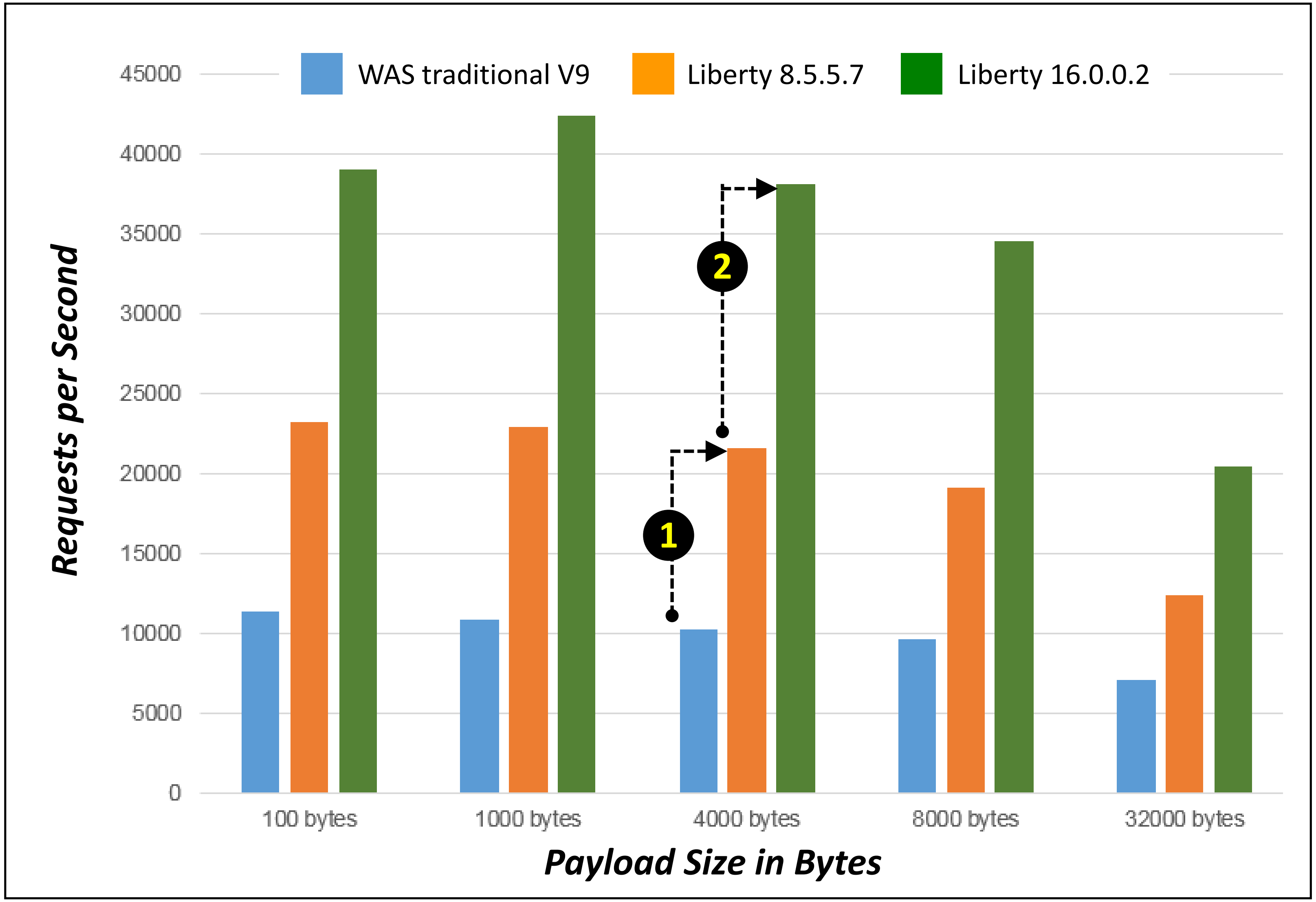
Idle CPU Time: WAS traditional vs. Liberty



Idle CPU time with 10 Liberty servers is approximately 3 times less than WAS traditional servers. The time shown is average per hour.

Performance results derived in a controlled environment under specific conditions. Your results may vary depending on a number of factors.

WOLA - WAS traditional v. Liberty on z/OS



Scenario is COBOL batch calling in to Java in WAS traditional and Liberty

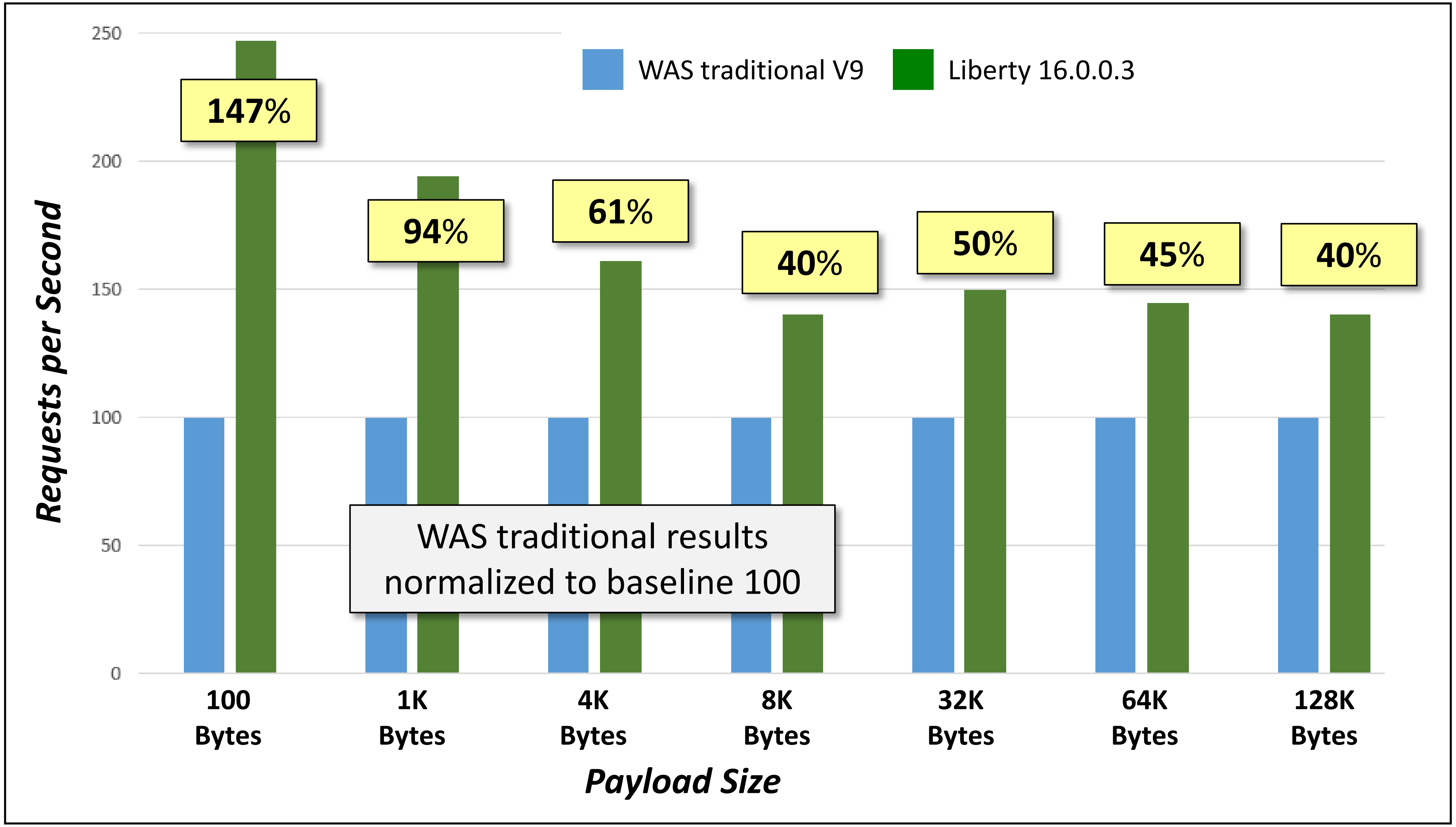
Liberty's WOLA support is in general more efficient than WAS . We see greater throughput comparing WAS traditional V9 vs. Liberty 8.5.5.7 (highlight ❶)

In 16.0.0.2 further enhancements were made the Liberty WOLA support providing even greater throughput (highlight ❷)

Performance results derived in a controlled environment under specific conditions. Your results may vary depending on a number of factors.

WOLA and IMS Inbound - WAS traditional v. Liberty on z/OS

The Liberty z/OS support for WOLA and IMS came available in the **16.0.0.3** release

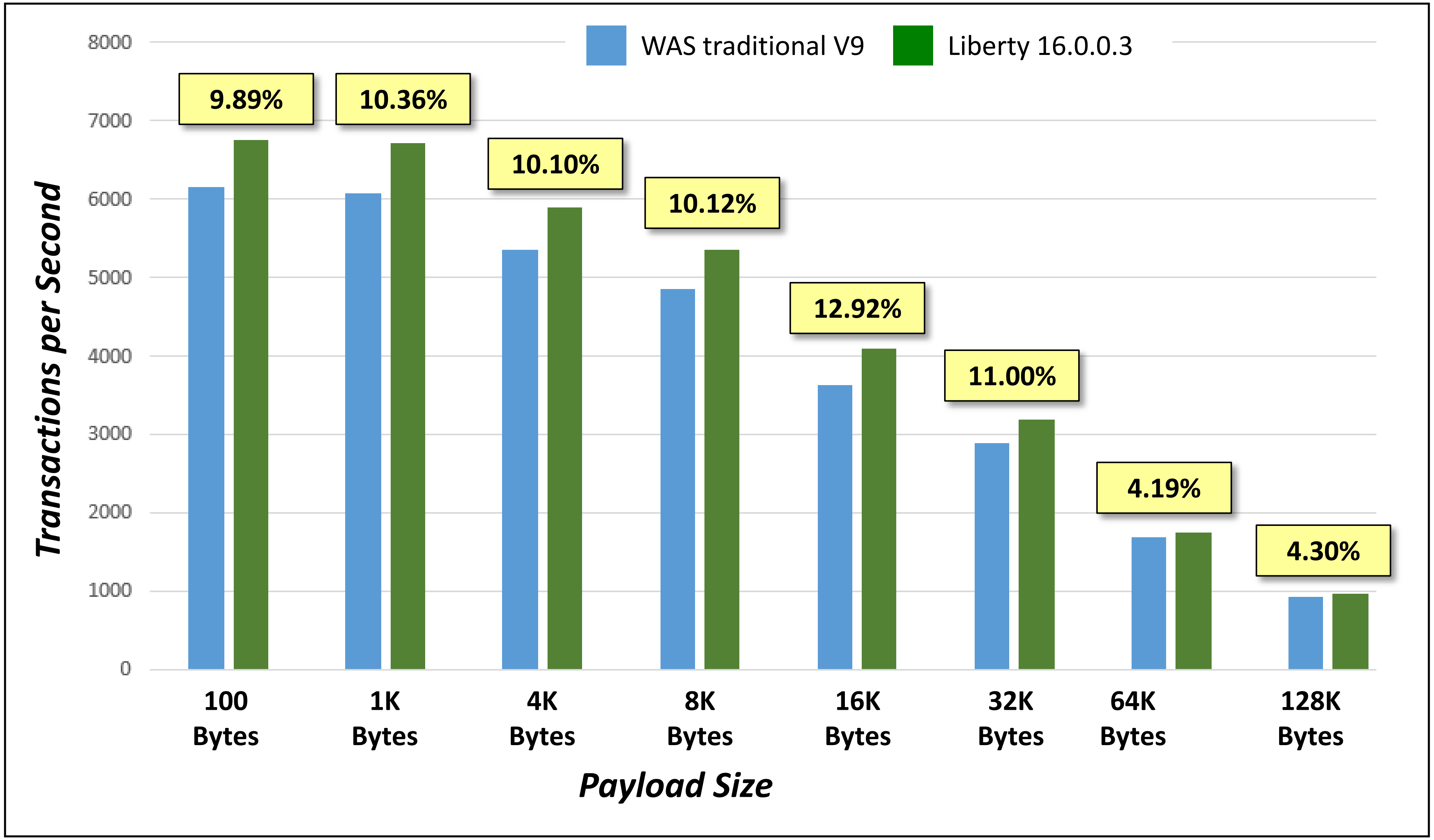


Liberty outperforms traditional WAS in all the payload sizes

Performance results derived in a controlled environment under specific conditions. Your results may vary depending on a number of factors.

WOLA and IMS Outbound - WAS traditional v. Liberty on z/OS

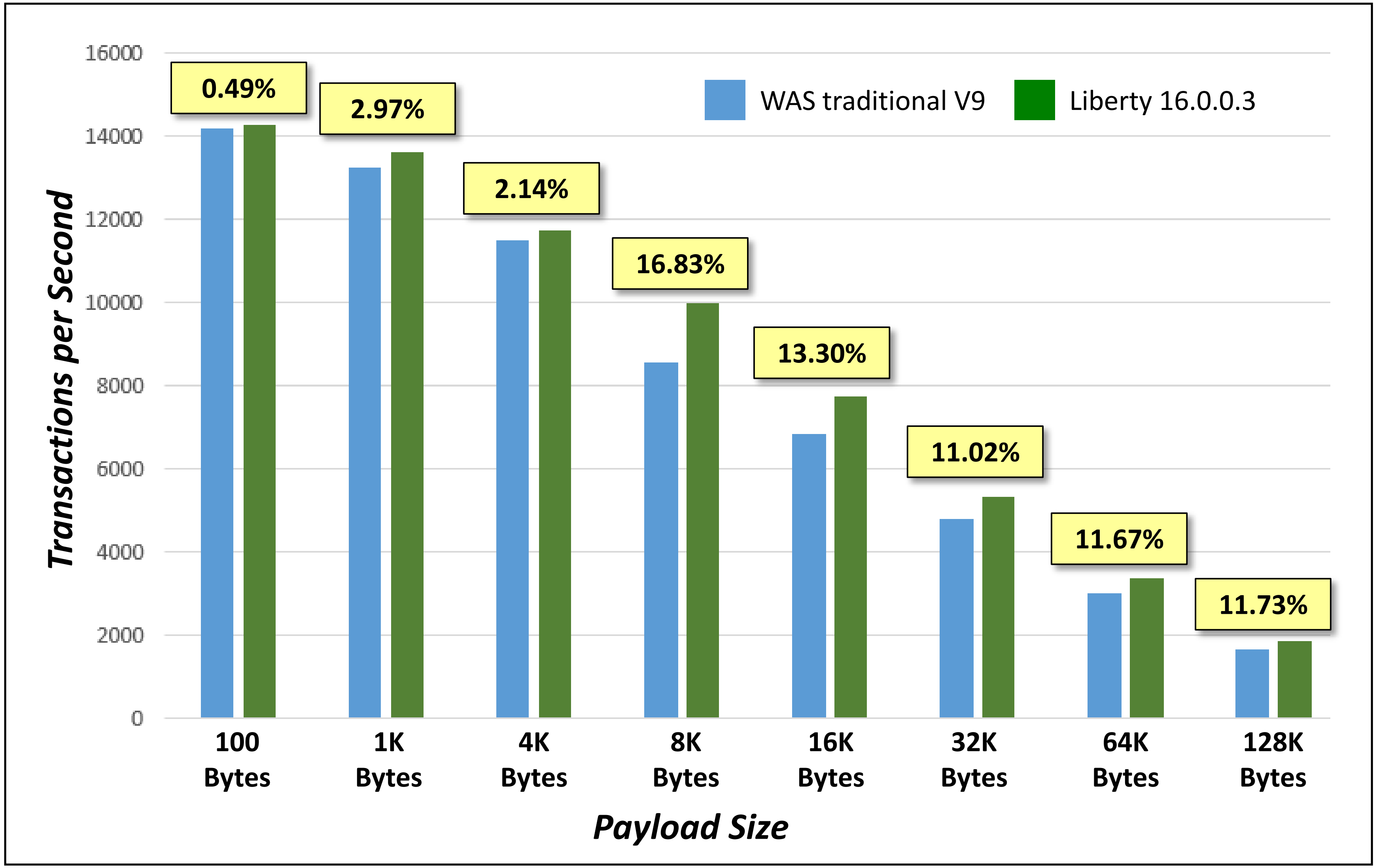
The Liberty z/OS support for WOLA and IMS came available in the **16.0.0.3** release



Liberty outperforms traditional WAS in all the payload sizes ranging from ~10% up to 32K payloads and ~4% in 64k and 128k payloads size.

Performance results derived in a controlled environment under specific conditions. Your results may vary depending on a number of factors.

WOLA and CICS Outbound - WAS traditional v. Liberty on z/OS



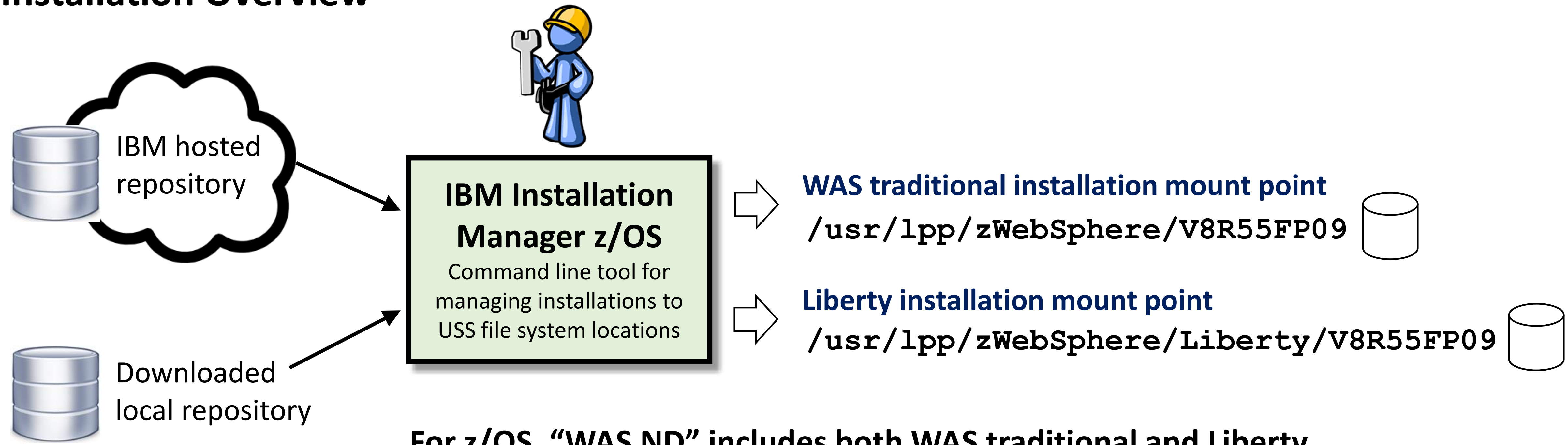
Liberty outperforms traditional WAS in all payload sizes.

The difference is less in smaller payload size and is more in larger payload size.

Performance results derived in a controlled environment under specific conditions. Your results may vary depending on a number of factors.

Other Information for Consideration

Installation Overview



IBM hosted repository

Downloaded local repository

IBM Installation Manager z/OS

Command line tool for managing installations to USS file system locations

WAS traditional installation mount point

`/usr/lpp/zWebSphere/V8R55FP09`

Liberty installation mount point

`/usr/lpp/zWebSphere/Liberty/V8R55FP09`

For z/OS, “WAS ND” includes both WAS traditional and Liberty

They are installed separately, and may be installed in different locations

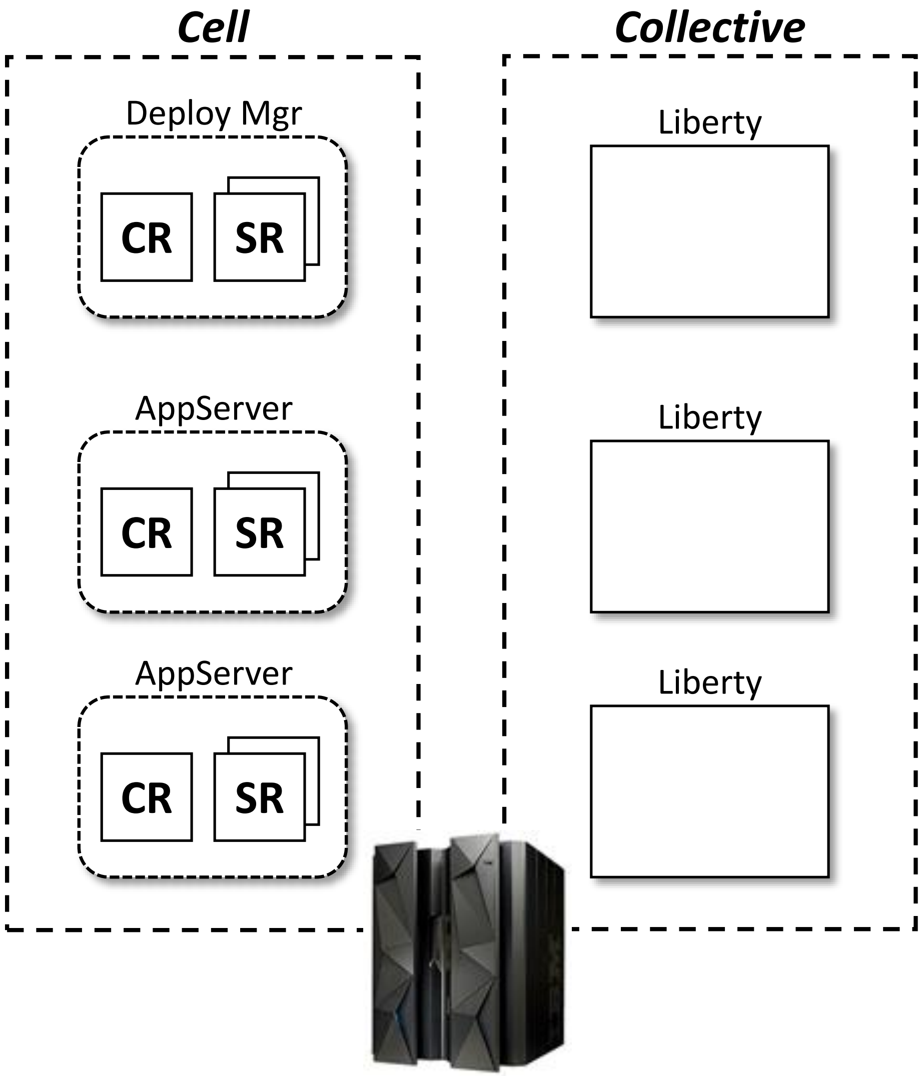
Maintenance is applied separately, so you may control when updates occur

You may maintain multiple levels of each in separate file systems

- WAS traditional is less flexible when it comes to moving up and down levels
- Liberty is by design flexible so you can easily change level of code used by servers

Installation Manager z/OS Techdoc
http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102554

Concurrent WAS traditional and Liberty



This is possible and can be accomplished

Same LPAR or same Sysplex.

They are separate installations, separate configurations, and separate started tasks. Normal z/OS considerations apply: avoid port conflicts, avoid naming conflicts, etc.

Purpose: dual environments during runtime cutover

Avoids “big bang” cutover; allows applications to be moved one at a time.

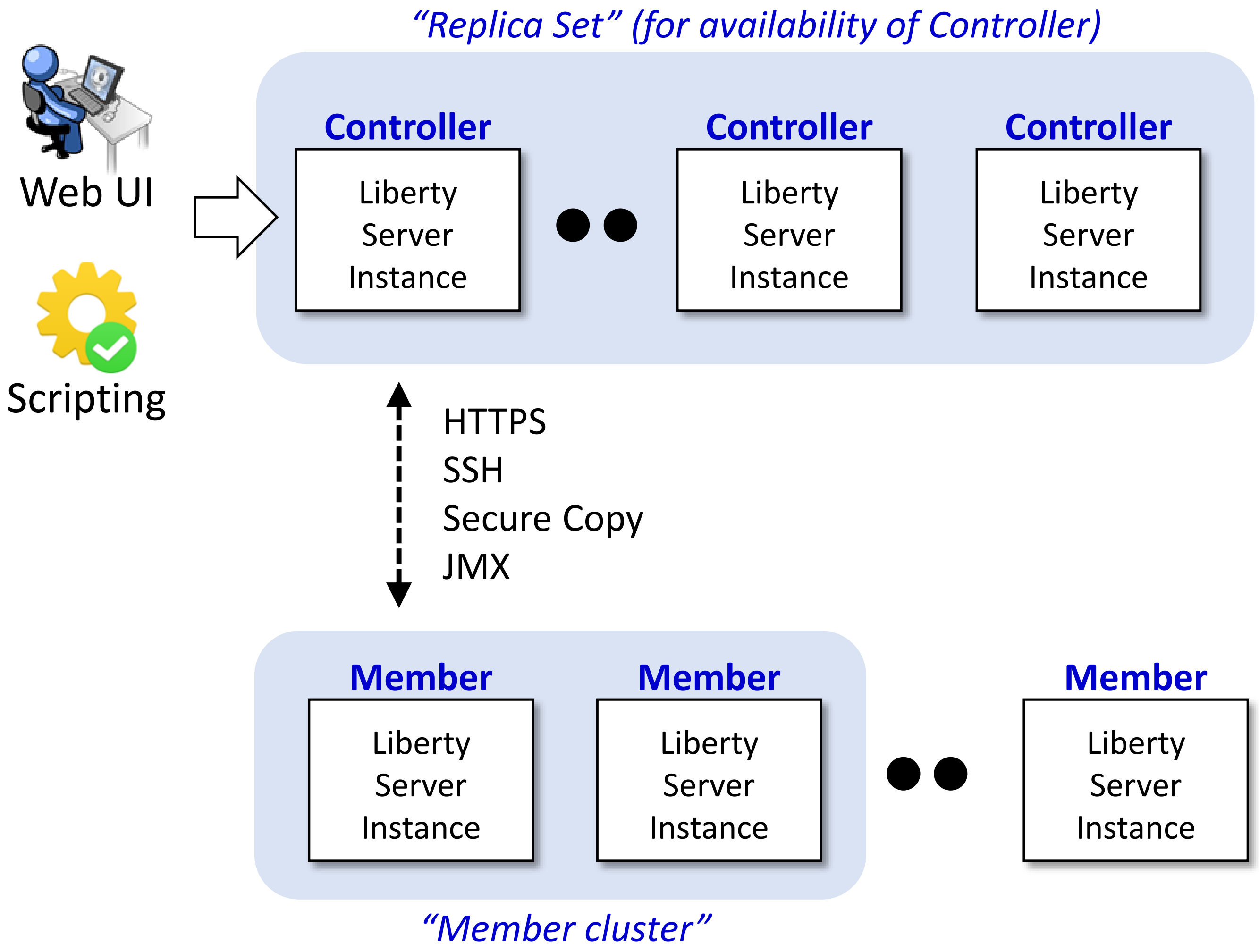
They would be managed separately

WAS traditional management model would be unaware of Liberty collective, and Liberty collective controller would be unaware of WAS traditional cell.

Application integration between environments is possible; complexity a function of pattern:

MQ (or JMS messaging) = relatively easy
REST = relatively easy
IIOP = more complex

Liberty Collectives Overview



"Collective"

A collection of Liberty servers with some servers designated as "controllers" and others as "members" of the collective.

Flexible: Join, Leave

Simple XML definitions specify the collective to which a server will be a member. Relatively easy to join a collective; easy to leave and join another.

Server clustering

Members can arrange into a cluster for purposes of application availability and intelligent workload placement.

Rich set of management beans

For monitoring and managing the environment

AdminCenter interface

For web interface to collective

Available, scalable

Controllers can be arranged into a highly available "replica set". Designed to scale to large topology.

Document Change History

Date	*Description*
May 17, 2016	Original document
Feb 8, 2017	Updated to reflect new function in Liberty z/OS (SMF, WOLA and IMS, SAF keyring for collectives), as well as the additional of a number of new performance charts.