

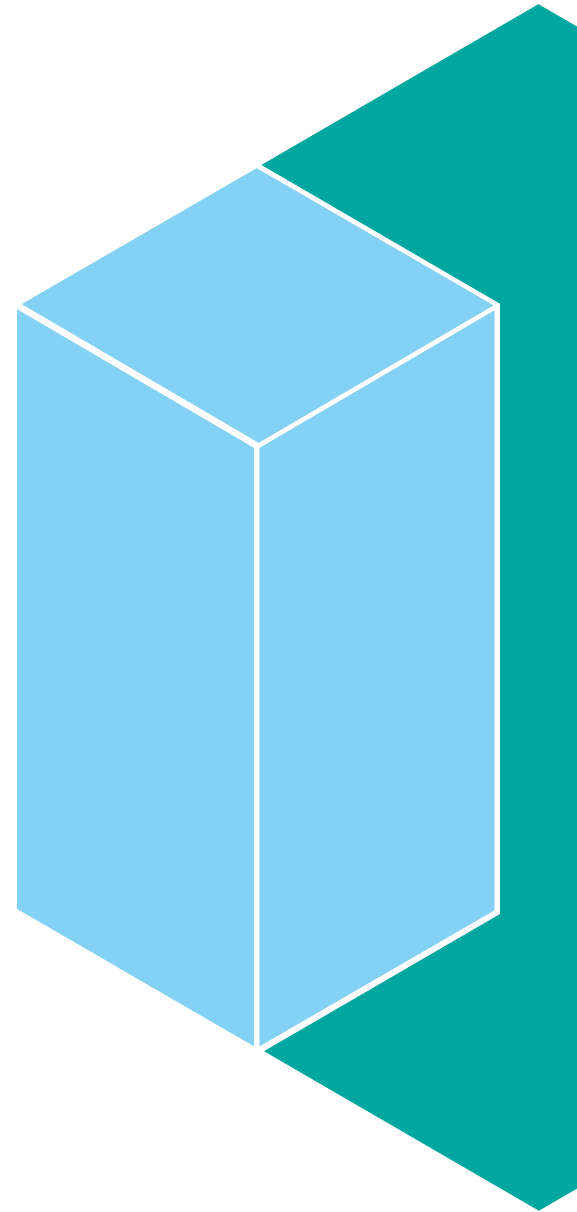
WAS z/OS with WOLA: a game changer

Carl Farkas

IBM Europe zWebSphere consultant

farkas@fr.ibm.com

(with thanks to Don Bagwell and others for much of the content)



Sharpen your competitive edge

2016 IMS Technical Symposium

March 7 – 10, 2016

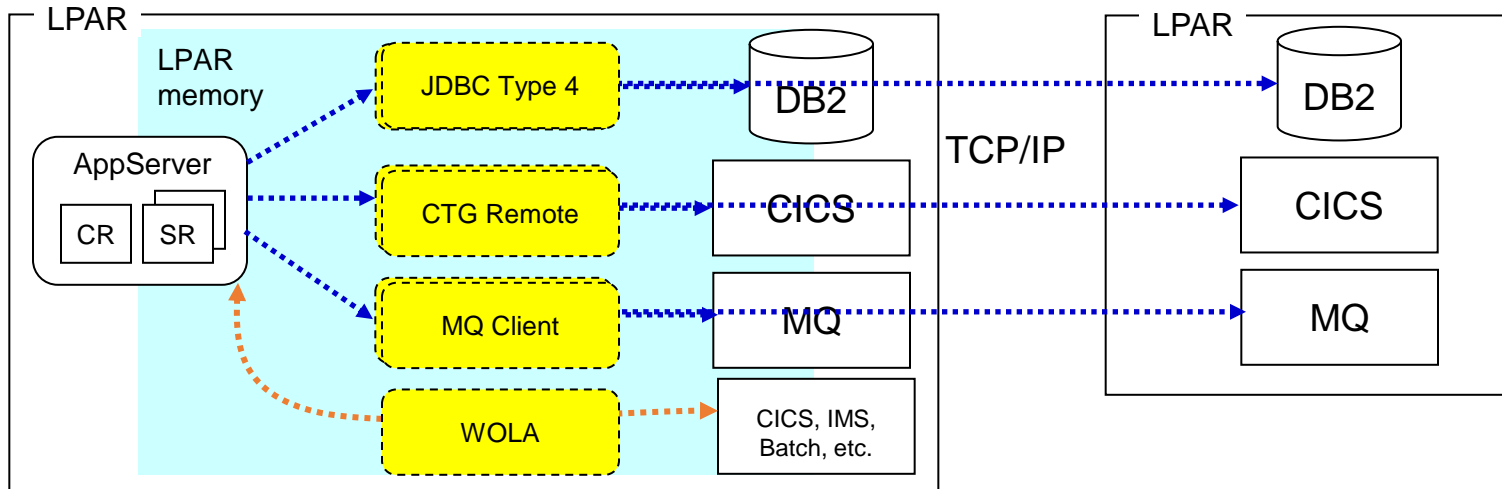
Wiesbaden, Germany

www.ims-symposium.com

Agenda

- **Introduction**
- **Enablement**
- **Development**
- **Example scenarios**
- **Positioning**
- **Recent enhancements**

Co-Location - Cross-Memory Communications



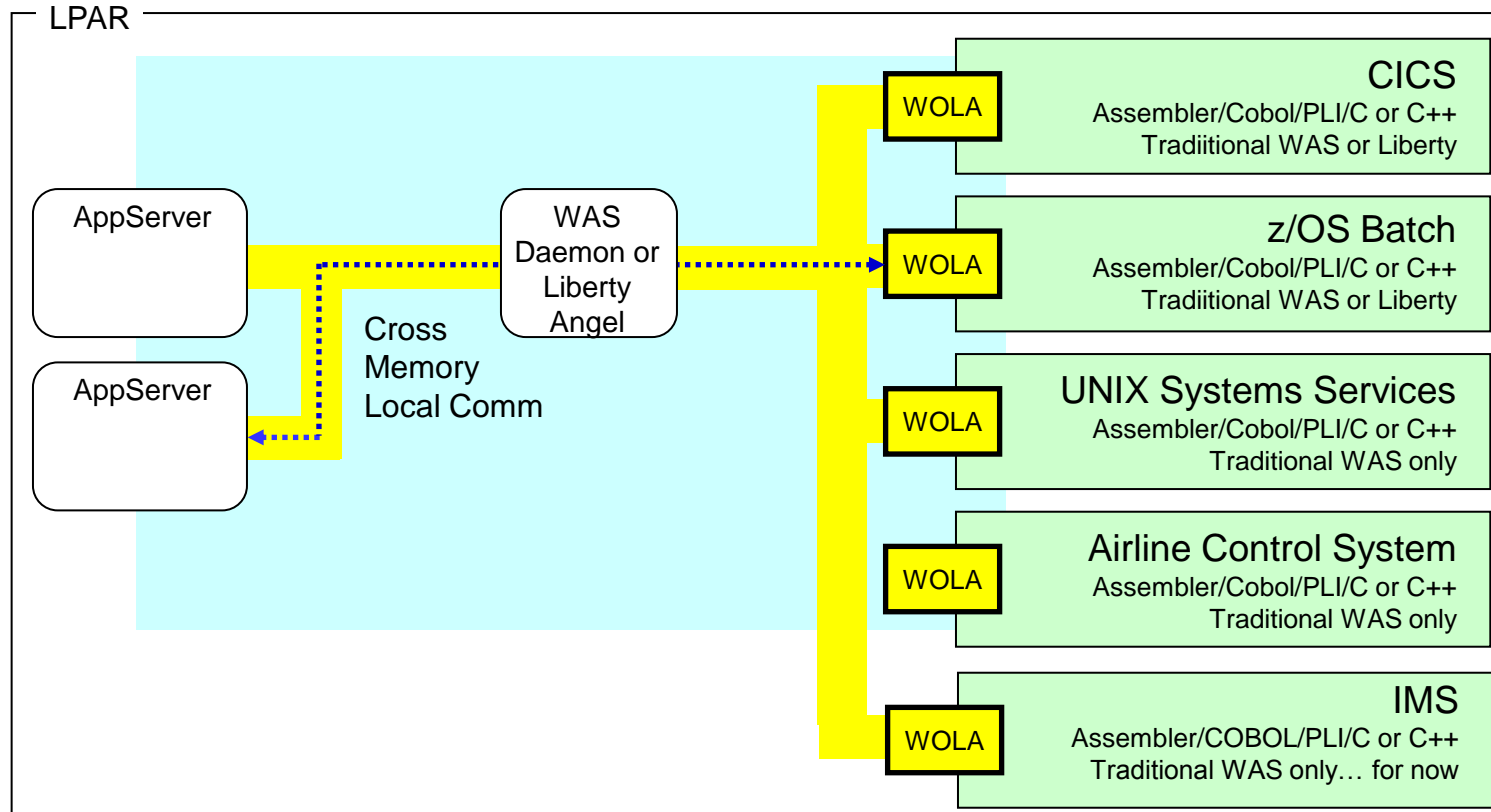
- ✓ Cross memory speed
- ✓ Avoids encryption overhead
- ✓ Security ID propagation
- ✓ Exploitation of z/OS transaction management (RRS)
- ✓ Avoid serialization of parameters
- ✓ Single thread of execution

Benefits: Save mips, increase robustness, augment security

For details, see: [WAS z/OS – the value of Co-Location](http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101476), <http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101476>

So what is WOLA?

WebSphere Optimized Local Adapter



- Not really new.... based on Local Comm cross-memory access (z/OS exclusive)
- Introduced (made available as a customer accessible API) beginning with WAS v7.0.0.4
- **Bi-directional ... WAS outbound or inbound to WAS.... simple data exchanging API**
- Supports security and transaction propagation (2PC only in traditional WAS...today)
- Very, very fast. 2x – 6x faster than other comparable solutions
- Efficiently leverage your other co-located z/OS assets

WOLA Interface -- Perspective from Eight Angles

Programs that initiate a call to WOLA do so through a supplied J2C adapter. Several WOLA-specific methods used to invoke services over WOLA

WebSphere Environment

Enterprise Java Bean
(Or Servlet)

Enterprise Java Bean

WOLA J2C Adapter

WOLA Execute()
ExecuteHome()

EJBs that will be the target of inbound calls need to implement the WOLA-supplied Execute() and ExecuteHome() classes.

A Batch program that wishes to initiate an outbound connection must write to the WOLA APIs

Calls into CICS come across WOLA-supplied BBO\$/BBO# task and transaction. Target CICS program unchanged if able to be invoked over COMMAREA or Channel/Container

Batch Environment

Batch Program

Batch Program

WOLA Modules/APIs

WOLA Modules/APIs

WOLA

WOLA BBO\$/BBO#

WOLA Modules/APIs

CICS Environment

CICS Program

CICS Program

A CICS program that wishes to initiate a connection must write to the WOLA APIs

A Batch program can be called into also, but this would be rare

(Traditional WAS only)

WOLA IMS
ESAF

WOLA OTMA

BMP/MPP/
IFP

IMS Dependent regions

A WAS application can call an existing unchanged IMS transaction using OLA over OTMA. This is "implicit" WOLA invocation.

See KC Liberty
"twlp_dat_useolar"

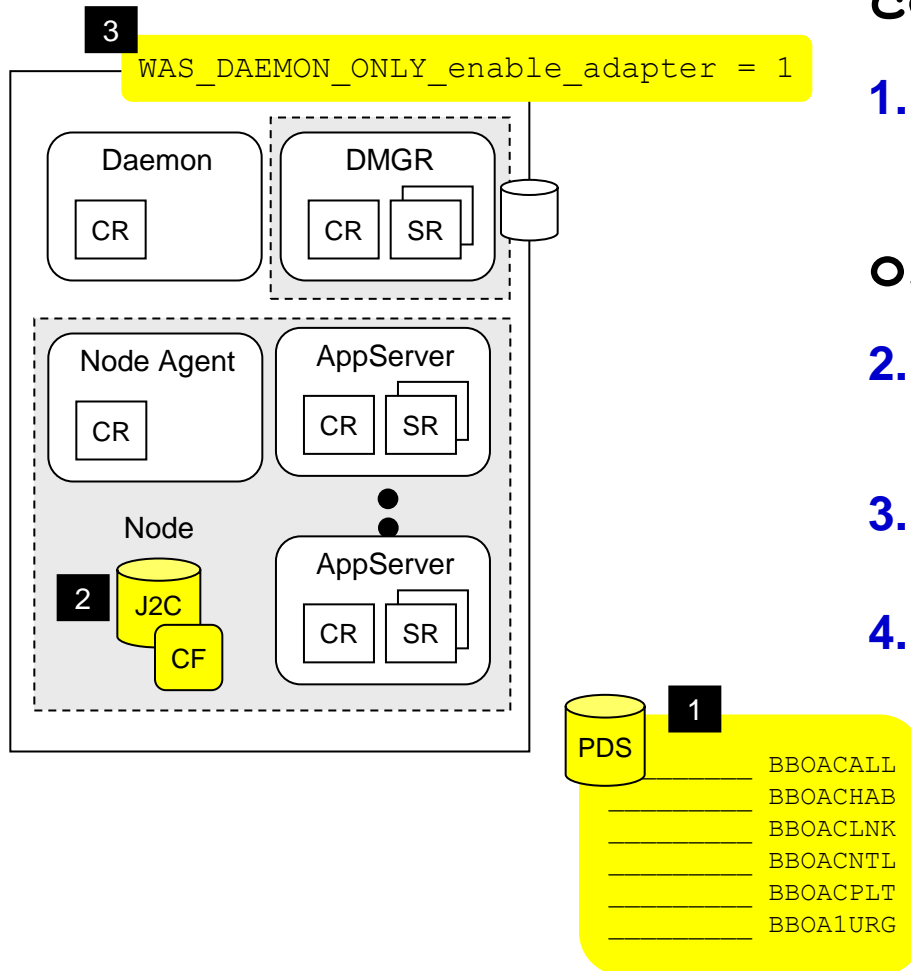
WAS can exchange with IMS symmetrically via ESAF, but explicit WOLA coding required on IMS

See KC traditional
WAS "cdat_ola"

Agenda

- Introduction
- Enablement
- Development
- Example scenarios
- Positioning
- Recent enhancements

The Essentials of Enabling WOLA for Use with traditional WAS*



copyZOS.sh

1. Modules copied out to PDS

So external address spaces (batch, CICS, etc.) can access modules and APIs

olaRar.py (or do manually)

2. J2C adapter installed with ConnFactory

This is what makes the WOLA modules available to the node (this is only necessary for “outbound” WOLA... into WAS)

3. WAS environment variable

Simple switch to enable function in Daemon

4. Configure resource adapter

The ola.rar is required for outbound calls

See KC traditional WAS
["tdat_enableconnector"](#)

For IMS side, see KC
traditional WAS
["tdat_enableconnectorims"](#)

* Note: these are the enablement steps for WAS v8; it's slightly different if you're using WAS v7

The Essentials of Enabling WOLA for Use with Liberty WAS

1. Install WOLA support into Liberty binary zFS

Use Liberty featureManager install zosLocalAdapters-1.0 command. This is done once per site typically.

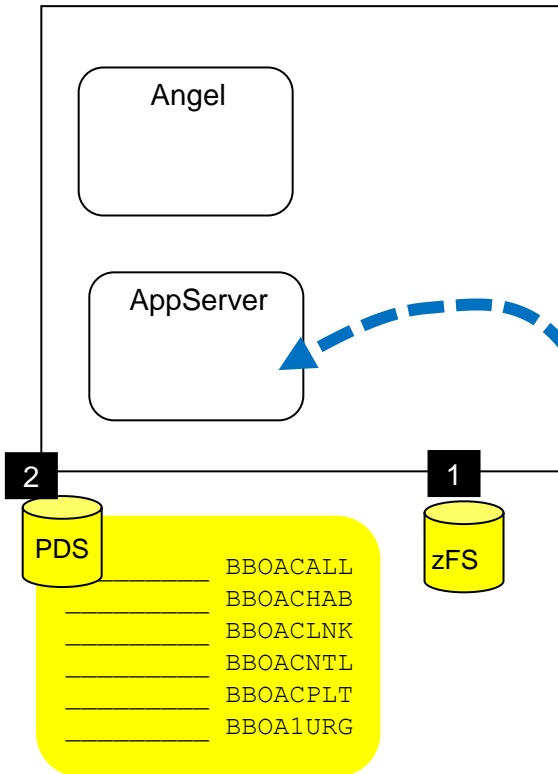
2. Copy modules copied out to PDS

So external address spaces (batch, CICS, etc.) can access modules and APIs. Only once per PAR.

```
cp -Xv wlp/clients/zos/* "///'$DSNAME'"
```

3. Enable WOLA in every server.xml

This is what makes the WOLA modules available to the App Server.



See KC
"twlp_dat_enableconnecto"

```
<featureManager>
  <feature>zosLocalAdapters-1.0</feature>
</featureManager>

<zosLocalAdapters
  wolaGroup="FARKAS" wolaName2="ZT01"wolaName3="ZOSCONN"/>

<connectionFactory id="wolaCF" jndiName="eis/ola">
  <properties.ola RegisterName="OLASERVER" />
</connectionFactory>
```

server.xml

SAF security definitions

■ Traditional WAS

- The basic installation of WAS includes the appropriate SAF (eg.RACF) definitions.
- You do, however, need to give READ access to the region ids (eg. IMS) for the CLASS(CBIND) for the target WAS, eg.

```
PERMIT CB.BIND.MYCELL.** CLASS(CBIND) ACCESS(READ)
```
- Nicely documented in WP101490 Quick Start.

See KC traditional WAS
“cdat_enableconnectorims”

■ Liberty WAS

- Using WOLA will require that the Liberty Angel be running, which entails numerous SAF definitions. See the WP101490.
- The BBG.AUTHMOD.BBGZSAFM.WOLA and .LOCALCOM profiles need to be defined in SERVER class and the Liberty server needs ACCESS(READ)
- The IMS region’s user id also needs ACCESS(READ) to the BBG.WOLA.group.name2.name3 profile in CBIND class.
- Nicely documented in WP101490 Liberty Quick Start

See KC Liberty
“twlp_config_security_zosr”

Agenda

- Introduction
- Enablement
- Development
- Example scenarios
- Positioning
- Recent enhancements

Developing with WOLA – “Legacy” (non-WAS) side

Legacy as “client” -> WAS

BBOA1REG – Register

BBOA1URG – Unregister

BBOA1INV – Invoke an EJB method

BBOA1CNG – Get Connection

BBOA1CNR – Release Connection

BBOA1SRQ – Send Request (async)

**BBOA1RCL – Receive response
length**

BBOA1GET – Get Data (async)

WAS -> Legacy as “server”

BBOA1REG – Register

BBOA1URG – Unregister

BBOA1SRV – Host a service

BBOA1SRP – Send a response

**BBOA1RCA – Receive any
request**

**BBOA1RCS – Receive a specific
request**

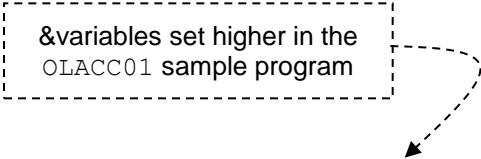
**BBOA1SRX – Send Response
Exception**

- **Colored verbs** above are for more advanced asynchronous support
- WAS provides an excellent set of source samples; see the WAS InfoCenter for complete documentation and search for “cdat_olaapis” or “twlp_dat_useoutboundconnection”.
- Also, excellent TechDoc at <http://www-03.ibm.com/support/techdocs/atsmastr.nsf/Web/Techdocs> and search “WP101490”

Example of API -- BBOA1REG (from OLACC01 Batch Sample)

BBOA1REG is one of thirteen APIs that come with WOLA. It's used by external address spaces to register into the WAS Daemon (traditional WAS) or Angel (Liberty):

&variables set higher in the
OLACC01 sample program



```
BBOA1REG (  &daemonGroupName /* Cell name normally, eg. "CFCELL"; maybe WAS SAF domain for Liberty */
            , &nodeName        /* Node name, eg. "CFNODE"; perhaps LPAR id for Liberty */
            , &serverName      /* AppServer name, eg. "CFSR011"; same suggested for Liberty */
            , &registerName    /* A name for my connection, eg. "CarlApp"; used by WAS Outbound */
            , &minConn
            , &maxConn
            , &regFlags
            , &rc
            , &rsn
        );
```

The KnowledgeCenter has very nice write-ups
of the APIs and the parameters

```
if (rc != 0)
{
    printf("Register error! rc: %d rsn: %d\n",rc,rsn);
    return(-1);
}
```

Some error? Returns beautiful return and reason codes. Same InfoCenter page spells it all out. Wonderful granularity of error reporting.

- Like any new API it takes a bit of time to learn the operations and syntax. But it's relatively easy, and the samples provide some nice examples.
- Note that the 3 part name seen above does not refer to the Cell & Node & AppServer with WOLA and WAS Liberty, but the syntax remains the same. Any unique 3 values can be used, but both parties (WAS & "other") must of course use the same values.

Example of API -- BBOA1INV (from OLACC01 Batch Sample)

Once registered to Daemon, how does batch invoke EJB? With the BBOA1INV API and naming the EJB's home interface JNDI:

```
BBOA1INV (  &registerName
            , &requestType
            , &requestServiceName
            , &requestServiceNameLen
            , &requestDataPtr
            , &requestDataLen
            , &responseDataPtr
            , &responseLen
            , &waittime
            , &rc
            , &rsn
            , &rv
            );
```

- The name of the registration connection to use to access the EJB. This is the pool of connections over to the WAS server address space, eg. "CarlApp"
- 1=for local EJB (typical case); 2 for remote EJB (for development mode)
- The JNDI name of the home interface for the target EJB in WAS

Earlier in OLACC01 sample this was set to:

```
ejb/com/ibm/ola/olasample1_echoHome
```

Which is the default JNDI name on the OLASample01.ear file

Point here is not to drill deep into programming specifics

Point is to illustrate key concept -- register into Daemon and make connection to target WAS AppServer, then invoke the target EJB using the registration pool and the EJB's JNDI home interface name

Developing with WOLA – WAS side

■ EJB Definition into WAS (“inbound WOLA”)

- Only stateless session beans are supported
- Remote home interface must be `com.ibm.websphere.ola.ExecuteHome`
- Remote interface must be `com.ibm.websphere.ola.Execute`
- Business logic is contained in the `byte[] execute(byte[] input)` method provided on the remote interface
- Remote/Home interfaces in `ola_apis.jar`
- See “tdat_useola_in_step2” for EJB details

■ OLA J2C Interface “outbound WOLA” from WAS

- WebSphere caller can be an EJB or Web component (Servlet/JSP)
- J2C supports CCI interfaces (ConnectionFactory, Connection, etc)
- ConnectionFactory obtained via JNDI

Create an Enterprise Bean (1.x-2.x)

Enterprise Bean Details

Select the session type, transaction type, and the classes necessary for this session bean.

Session type:

Transaction type:

Bean supertype:

Bean class:

☒ Remote client view

Remote home interface:

Remote interface:

[Enterprise Applications](#) > [OLASample2](#) > [EJB JNDI names](#)

EJB JNDI names

Each non-message-driven enterprise bean in your application or module must be bound to a Java Naming and Directory Interface (JNDI) name. For beans in a pre-EJB 3.0 module, you have to use JNDI name for the bean to provide the binding. For beans in an EJB 3.0 module, you can optionally provide binding through JNDI name for the bean or local/remote home JNDI names. If JNDI name for the bean is specified, you cannot specify binding for its local/remote home and any business interface. If no JNDI name is specified for beans in an EJB 3.0 module, runtime will provide a container default.

EJB module	EJB	URI	Target Resource JNDI Name
OLA Sample2	olasample1_echo	OLA_Sample2.jar,META-INF/ejb-jar.xml	Target Resource JNDI Name ejb/com/ibm/ola/olasam
OLA Sample2	Was2Cics	OLA_Sample2.jar,META-INF/ejb-jar.xml	Target Resource JNDI Name ejb/com/ibm/ola/Was2Ci
OLA Sample2	olasample1_roundtrip	OLA_Sample2.jar,META-INF/ejb-jar.xml	Target Resource JNDI Name ejb/com/ibm/ola/olasam

Resource adapters

[Resource adapters](#) > [OptimizedLocalAdapter](#) > [J2C connection factories](#)

Use this page to create a connection factory for use with the resource adapter. The connection factory is a collection of configuration values that define a WebSphere(R) Application Server connection to your Enterprise Information System (EIS). The connection pool manager uses these properties as directions for allocating connections during runtime. You can configure multiple connection factories for each resource adapter.

☐ Preferences

☒ ☐ ☐ ☐ ☐

Select	Name	JNDI name	Scope	Provider	Description	Connection factory interface	Category
<input type="checkbox"/>	wola	eis/ola	Node=cfnod1	OptimizedLocalAdapter	For WOLA - created by Carl	javax.resource.cci.ConnectionFactory	

Total 1

Testing WOLA with IMS

- Strongly recommend starting with IBM supplied samples found in `<zWAS_install_root>/util/zos/OLASamples`
 - For WAS side, use the IMS OTMA tester OLA_Sample2.ear
 - For IMS implicit side, see the OLAPL* sample PL/I programs for OTMA
 - For IMS explicit side, CICS samples can work
- Work thru the Techdocs found in WP101490

IBM WAS on z/OS Optimized Local Adapters Test Utility Version 2 - Mozilla Firefox: IBM Edition

File Edit View History Bookmarks Tools Help

http://zt01.pssc.mop.fr.ibm.com:20667/OLA_Sample2_Web/ OTMA_OPEN RSN

Most Visited IBM

Integrated Solutions Con... x IBM WAS on z/OS Opti... x IBM WebSphere Application ... x IBM Information Manag... x mail Orange

IBM WAS z/OS Optimized Local Adapters Test Utility Panel Version 3

Data to send to external address space : ☐ Convert to ebcdic
Test data to external a/s

Response back from external address space : ☐ Convert to ascii

OLA Register Name (max 12 chars)
CICSTEST

OLA Service Name (max 256 chars)
OLAP02

CICS Link Server-specific data : ☐ Use Containers
CICS-Link Request Container Id (max 16 chars) ☐ BIT Container
WAS_BBOA_REQ
CICS-Link Response Container Id (max 16 chars) ☐ BIT Container
WAS_BBOA_REQ
CICS-Link Transaction id (max 4 chars) BBO#

OLA over IMS OTMA data: ☒ Use OLA over OTMA ☒ Print debug data?
IMS OTMA XCF Group Name IMSCGRP IMS OTMA Server Name IMC1 IMS OTMA size of message to echo 100 Byte

Run WAS->External address space test!

Done

See KC "cdat_olasamples"

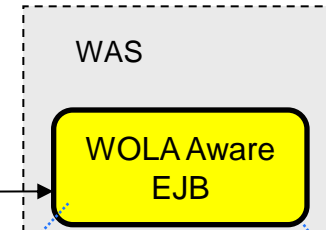
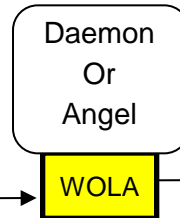
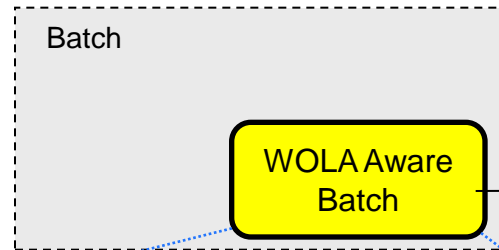
Agenda

- Introduction
- Enablement
- Development
- Example scenarios
- Positioning
- Recent enhancements

Invoking WAS Program, Batch, IMS, CICS → WAS

CFCELL

AppServer
CFSR011



```

BBOA1REG ( "CFCELL", "CFNODE1", "CFSR011",
"CarlBat",
&minConn, &maxConn, &regFlags, &rc, &rsn);
:
BBOA1INV ( "CarlBat", 1
, "ejb/com/ibm/ola/olasample1_echoHome"
, &requestServiceNameLen
, "Knock, knock.... anyone home in WAS?"
, &requestDataLen
, &responseDataPtr, &responseLen
, &waittime, &rc, &rsn, &rv);
:
BBOA1URG ( "CarlBat", &unregFlags, &rc, &rsn);
    
```

Java EE - IBM Rational
Enterprise Explorer

Cell=cfcell, Profile=default

Enterprise Applications

Enterprise Applications > OLASample1 > EJB JNDI names

EJB JNDI names

Each non-message-driven enterprise bean in your application or module must be bound to a Java Naming and Directory Interface (JNDI) name. For beans in a pre-EJB 3.0 module, you have to use JNDI name for the bean to provide the binding. For beans in a EJB 3.0 module, you can optionally provide binding through JNDI name for the bean or local/remote home JNDI names. If JNDI name for the bean is specified, you cannot specify binding for its local/remote home and any business interface. If no JNDI name is specified for beans in a EJB 3.0 module, runtime will provide a container default.

EJB module	EJB	URI	Target Resource JNDI Name
OLA Sample1	olasample1_echo	OLA_Sample1.jar:META-INF/ejb-jar.xml	Target Resource JNDI Name [olasample1_echoHome]
OLA Sample1	Was2Cics	OLA_Sample1.jar:META-INF/ejb-jar.xml	Target Resource JNDI Name [ejb/com/ibm/ola/Was2C]
OLA Sample1	olasample1_roundtrip	OLA_Sample1.jar:META-INF/ejb-jar.xml	Target Resource JNDI Name [ejb/com/ibm/ola/olasam]

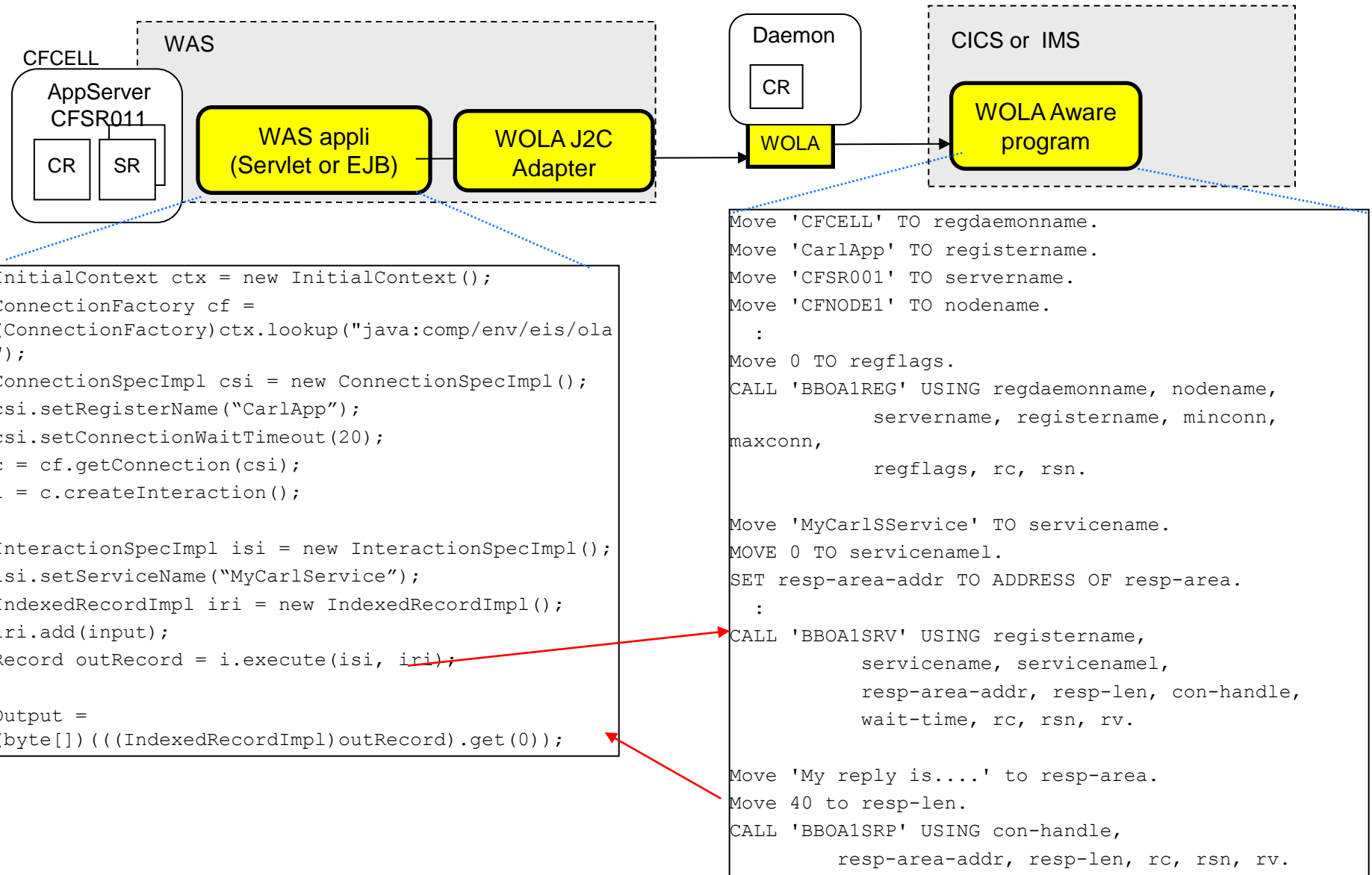
OK Cancel

```

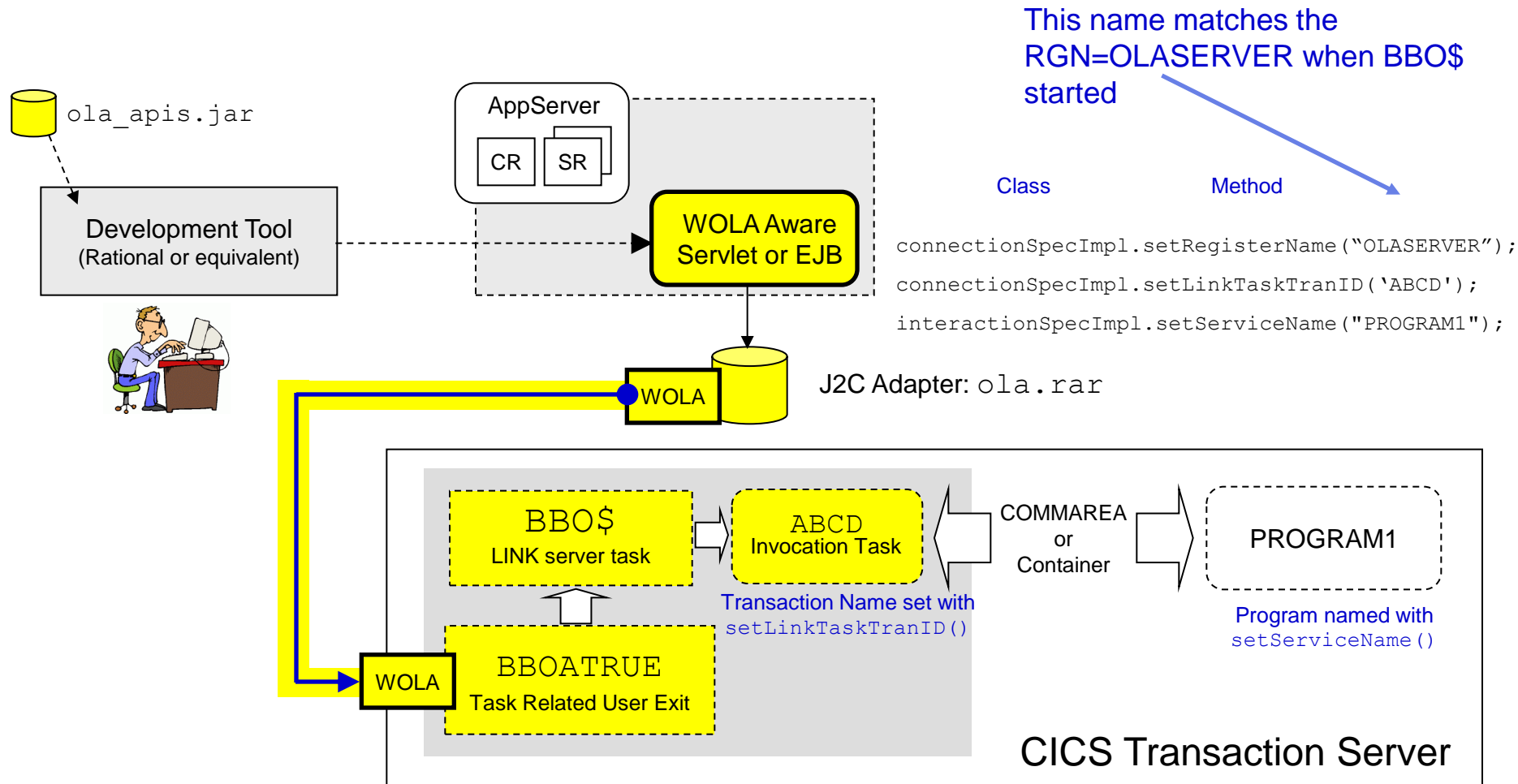
package com.ibm.ola;

public class olasample1_echoBean implements
javax.ejb.SessionBean {
    private javax.ejb.SessionContext mySessionCtx;
    :
    public byte[] execute(byte[] arg0) {
        String list = new String(arg0);;;
        System.out.println("Ola amigo. I received your data:
"+list);
        return arg0;
    }
}
    
```

Invoking **explicit** CICS or IMS Program, WAS → CICS or IMS

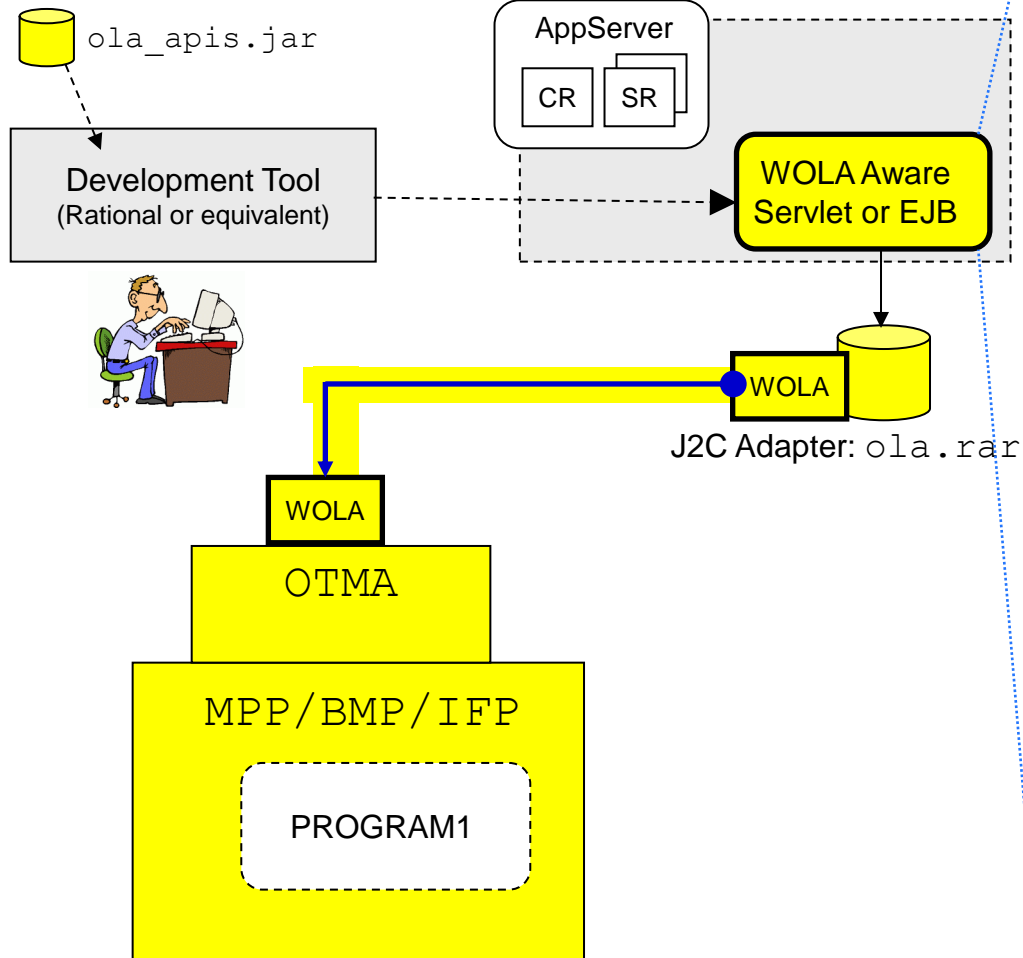


Invoking **implicit** CICS Program, WAS → CICS



This illustrates use of WOLA-supplied BBO\$ and BBO# to invoke CICS program without any modification to the CICS program code.

Invoking **implicit** IMS Program via OTMA, WAS → IMS



```
:
InitialContext ctx = new InitialContext();
ConnectionFactory cf =
    (ConnectionFactory) ctx.lookup("java:comp/env/eis/ola");
ConnectionSpecImpl csi = new ConnectionSpecImpl();
csi.setConnectionWaitTimeout(20);
csi.setOTMAServerName("IMC1");
csi.setOTMAGroupID("IMSCGRP");
csi.setOTMASyncLevel("1");
csi.setUseOTMA(boolean true);

c = cf.getConnection(csi);
i = c.createInteraction();
InteractionSpecImpl isi = new InteractionSpecImpl();
IndexedRecordImpl iri = new IndexedRecordImpl();

java.nio.ByteBuffer bb =
    java.nio.ByteBuffer.allocate(BYTEBUF_LEN);
bb.putShort(BYTEBUF_LEN); /* Set LL */
bb.putShort((short)0); /* Set ZZ */

bufx = "OLAP02 " + "My data here"; /* Set the trans and data */
byte[] bufz = bufx.getBytes("cp1047");
bb.put(bufz); /* Write translated bytes to ByteBuffer */

bb.rewind(); /* Reset ByteBuffer for reading from start */
input = new byte[bb.limit()];
bb.get(input); /* Read ByteBuffer into byte array */
iri.add(input); /* Set up full call structure */

Record outRecord = i.execute(isi, iri); /* Make the call */

Output = (byte[])((IndexedRecordImpl)outRecord).get(0);
String outputstr = new String(output,"cp1047"); /* Translate it */
:
```

This illustrates use of WOLA-supplied to invoke IMS program without any modification to the IMS program code.

See KC
"tdat_useoutboundconnection"
and "tdat_connect2wasapp"

Agenda

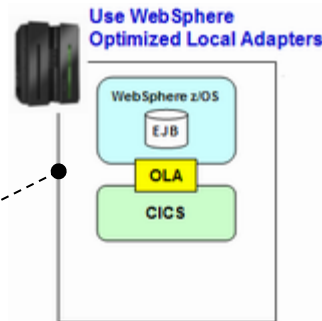
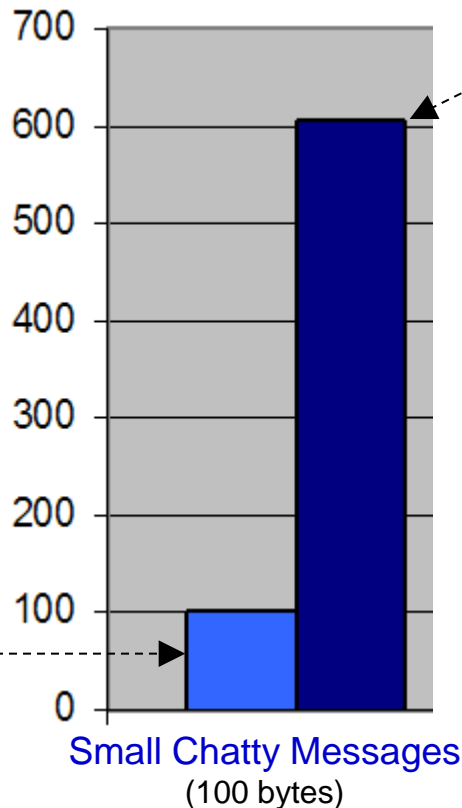
- Introduction
- Enablement
- Development
- Example scenarios
- Positioning
- Recent enhancements

WAS → CICS, Using SOAP vs. WOLA on the Same LPAR

We ran a test ... WAS and CICS on the same LPAR. CICS driving into WAS with 100 byte SOAP messages over HTTP.

Relative Throughput

Based on the specific CPU and memory of this benchmark system



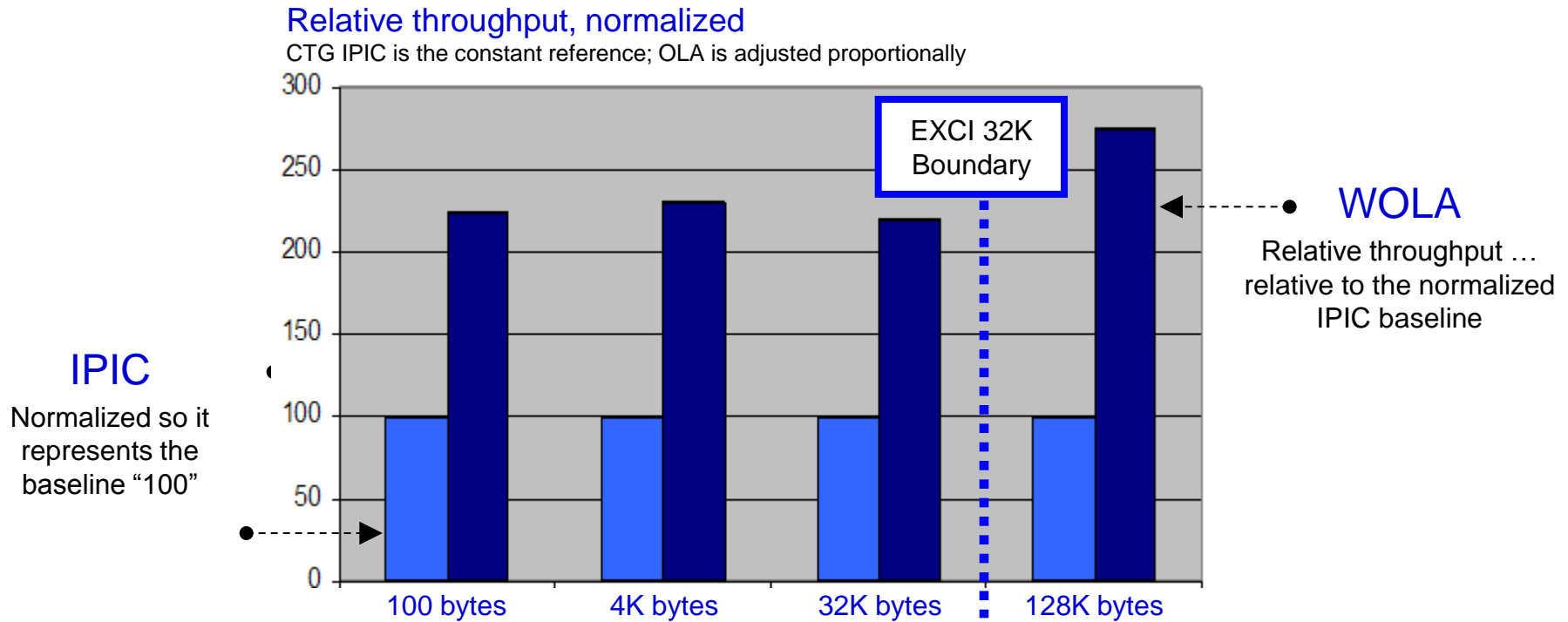
That's a big difference
How is that possible?

Web Services is an open and effective exchange mechanism, but it is not optimized

- CICS has to transform COMMAREA to SOAP XML
- It has to transmit over network
In this case an optimized local TCP network at that!
- WAS has to take in the XML, parse it, and turn it into the format expected by the receiving EJB

By comparison, WOLA is optimized for cross-memory exchanges. No transforming COMMAREA to XML, no initiating a TCP exchange









WAS → CICS, Using IPIC of CICS TS3.2



WOLA is a very good large message local transfer mechanism












WOLA and CTG positioning

WOLA is a complementary technology with CTG. Both have their place within an enterprise architecture.

	Relative Advantage Favors ...	
	WOLA	CTG
Bi-directional ... WAS→CICS and CICS→WAS WOLA is bi-directional, CTG is only WAS→CICS		
Part of the WebSphere Application Server z/OS Product WOLA shipped with 7.0.0.4, CTG is a separate FMID		
Able to be used for local or remote access to CICS WOLA is a local technology only, CTG supports both local EXCI as well as TCP-based remote access		
Two-Phase Commit WAS→CICS ** WOLA 7.0.0.12 added support for 2-phase Commit for WAS to CICS **		
Two-Phase Commit CICS→WAS CTG can not be used for CICS→WAS. WOLA able to propagate TX CICS→WAS with full 2-phase commit support using RRS for syncpoint coordination.		
Flexible use of CICS channels and containers WOLA restricts container usage to one named channel only: IBM-WAS-ADAPTER. CTG supports multiple channels. WOLA uses indexedrecord while CTG uses mappedrecords. That means CTG supports the passing of multiple named containers on a channel while WOLA can not.		
** WOLA 8.0.0.5/8.5.0.2 added support for naming channels and multiple containers **		

WOLA and IMS-JCA Connect positioning (traditional WAS only)

WOLA is a complementary technology with IMS-JCA Connector.
Both have their place within an enterprise architecture.

	Relative Advantage Favors ...	
	WOLA	IMSC
Bi-directional and able to call existing unchanged IMS transactions		
Part of the WebSphere Application Server z/OS Product <small>WOLA II shipped with 7.0.0.12, IMS Connect is a separate FMID that ships with IMS</small>		
Able to be used for local access to IMS <small>WOLA is a local technology only, IMS supports TCP-based access, which can be used remotely or locally</small>		
Able to be used for remote access to IMS <small>WOLA is a local technology only, IMS supports TCP-based access, which can be used remotely or locally</small>		
Propagation/assertion of User Identity <small>WOLA can propagate the thread-level ID over a call into the WAS EJB container and assert it. WOLA can propagate the thread-level ID over an OTMA call into IMS MPP & IFP</small>		
Global Transactions WAS → IMS <small>Available since Nov 2012!</small>		
Global Transactions IMS → WAS <small>Added to WOLA with 8.0.0.4</small>		
Speed (throughput rate) <small>Customer tests have suggested performance gain is over 6x</small>		

Traditional WAS WOLA vs. Liberty WAS WOLA

The two are similar at the programming interface, but different in some other respects:

Similarities

- Outbound JCA programming interfaces are the same
- Inbound native API programming model is the same
- CICS Link Server Task function very similar in design and operation
- Security assertion with CICS both directions supported
- Supplied samples nearly identical

Differences

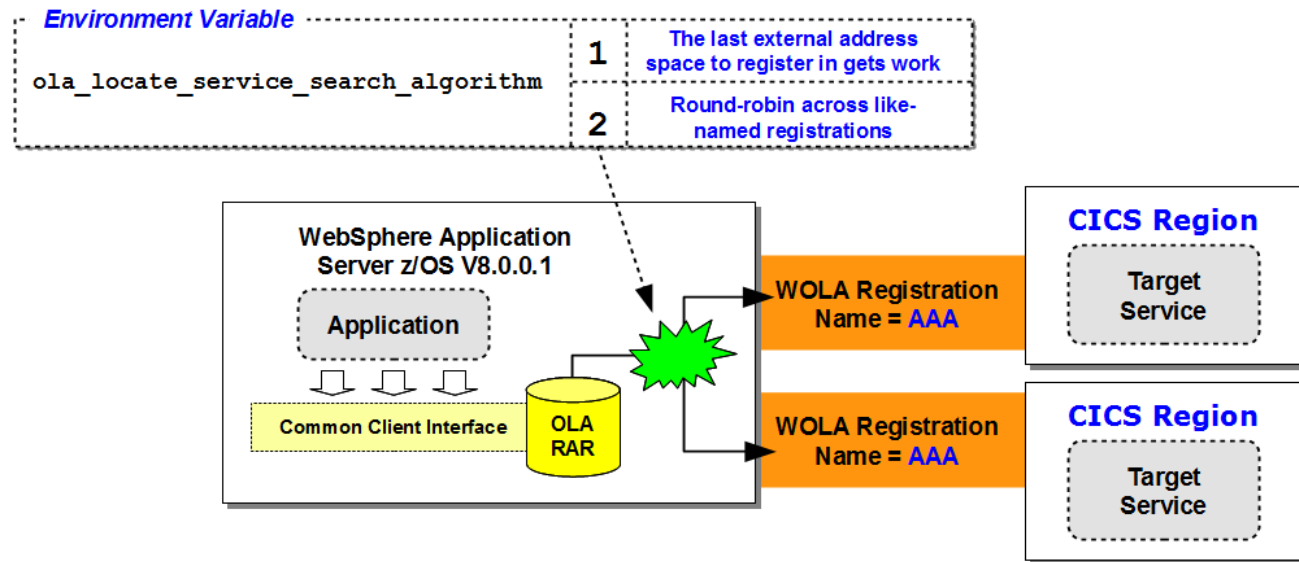
- Global transaction *not* (yet) supported with Liberty WOLA
Applications that start global transactions will need to be modified before using with Liberty WOLA
- General programming APIs of Liberty not as complete as full-function WAS
Depending on what application is doing, it may or may not operate with Liberty (this is more a Liberty statement than a WOLA statement)
- Target EJB for inbound must be EJB 3.x and has Liberty-specific design requirements
- Round-Robin and Alternate JNDI not supported
Relied on function of traditional WAS not present in Liberty
- Liberty WOLA and IMS not supported (yet)
- No WOLA MODIFY commands or SMF 120.10 for WOLA

Agenda

- **Introduction**
- **Enablement**
- **Development**
- **Example scenarios**
- **Positioning**
- **Recent enhancements**

V8.0.0.1 and WOLA Round-Robin (traditional WAS only)

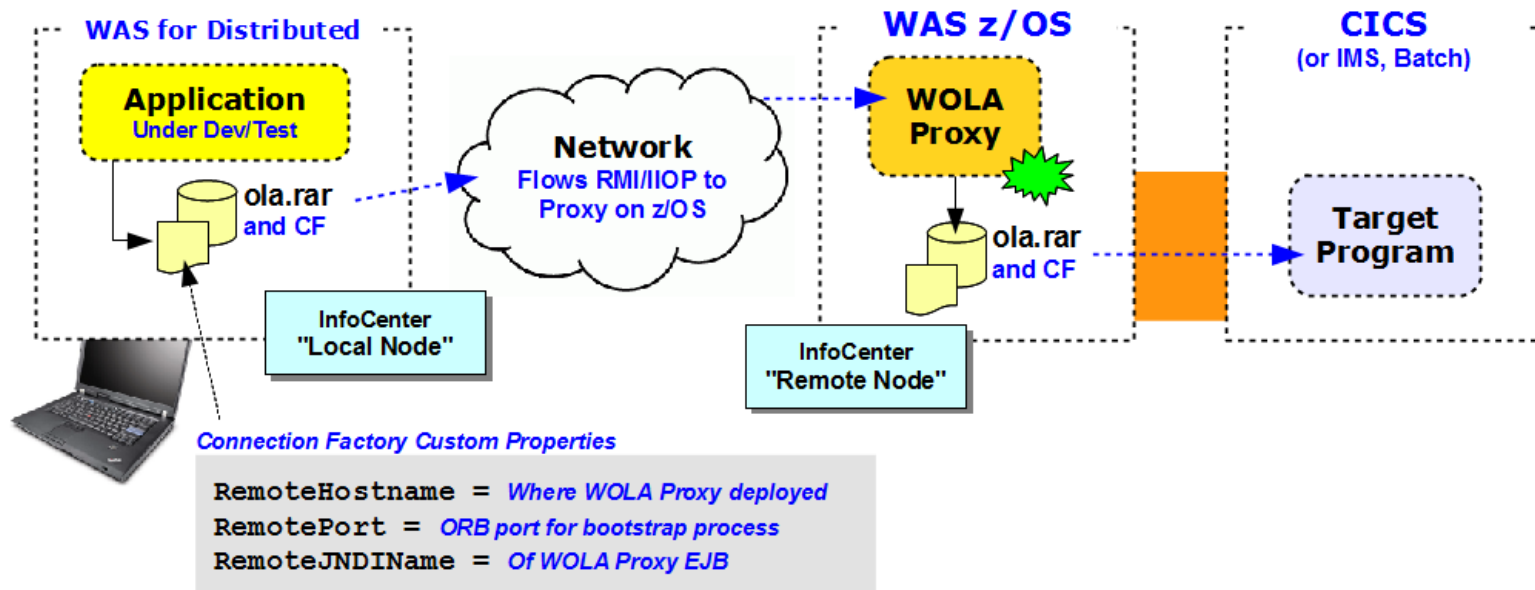
The 8.0.0.1 fixpack brought new WOLA function, including ability to round-robin between multiple instances of the partner (address space, eg. IMS or CICS region) registered into the server with the same name:



- For calls outbound from WAS to external address space, Registration names must be identical
- Targeted service must be present in multiple address spaces participating in the work distribution
- Any supported external address space, not just CICS

V8.0.0.1 Development Mode - Outbound Applications (traditional WAS only)

The focus here is on developing and testing WOLA outbound applications without the developer needing direct access to a z/OS system



Limitations:

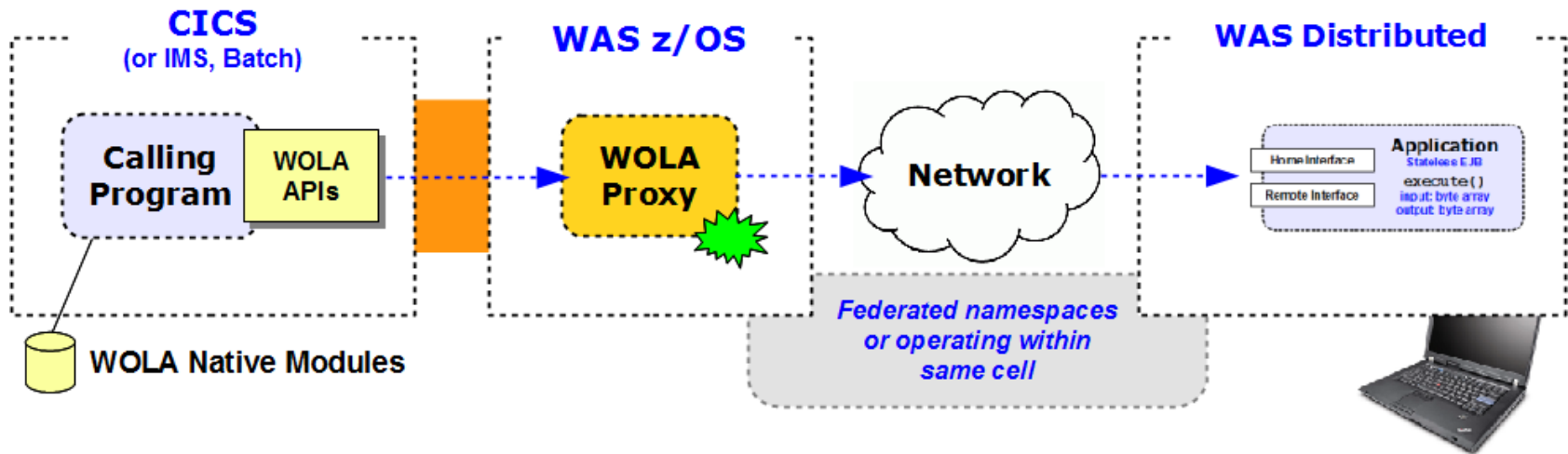
- Can not participate in global transaction 2PC
- Can not assert distributed WAS thread ID up to z/OS.

For additional info, check the InfoCenter, and search for "cdat_devmode_overview"

V8.0.0.1 Development Mode - Inbound Applications (traditional WAS only)

Let's take the reverse ... the case where you wish a native z/OS program to make an inbound call to a target EJB running in WAS. Can EJB be on WAS distributed?

Yes ...



- WOLA API developer writes as if target EJB is in the WOLA-attached WAS z/OS server
- One parameter difference - requesttype on BBOA1INV or BBOA1SRQ - set to "2" (for remote EJB request) rather than "1"
- EJB Developer develops stateless EJB with WOLA class libraries as if deployed on z/OS

For additional info, check the InfoCenter, and search for "cdat_ola_remotequest"

Customer examples...

IMS Technical Symposium

W&W Informatik GmbH

**Using WOLA and OTMA for Efficient
Communication between IMS/COBOL and
WebSphere z/OS at W&W**

Session B10

Daniel Schoeman – Senior Middleware Specialist

daniel.schoeman@ww-informatik.de

W&W Informatik GmbH, Germany



ww wüstenrot
württembergische

“Planned Production Dates:

- ☐ IBAN Converter: June 2015, with 100.000 daily COBOL to EJB Requests
- ☐ KAP Client Lookup: October 2015, with 50.000+ daily Java to IMS Requests
- ☐ Tax Software (Cortax): 2015, with 10.000 daily COBOL to EJB Requests”

2016 IBM Systems Technical Universities



IBM z Systems Technical University – München, Germany June 13-17

Enterprise IT infrastructure for cognitive business.

Reinventing IT for digital business, the new IBM z13 mainframe and z Systems are built for mobile, ready for and trusted for cloud. IBM z Systems provide the computing infrastructure for the mobile generation and the new app economy. Designed to exploit the mobile transaction explosion, z Systems apply in-transaction analytics and offer the most secure, trusted service delivery — all while transforming the efficiency and economics of IT.

IBM z Systems lectures and labs will focus on following topics:

- The new IBM z13 and its technology innovations
- IBM z Systems Enterprise Data Compression (zEDC) and Flash Express z13 update and lessons learned, z13 and z/OS dispatching update, and SMT and SIMD
- z/OS Version 2.1 and 2.2 latest updates, migration and advanced functions
- z/OSMF Version 2.1 and 2.2 implementation and configuration
- What's new in Linux on z Systems
- z/VM new features, advanced functions and implementation updates
- What's new in z Systems software pricing on the z13
- How cloud, analytics, mobile, social (CAMS) are remaking the mainframe
- Using Hadoop to analyze z Systems data
- IBM CICS Version 5 planning and implementation ...

Questions: stg_conferences@be.ibm.com

Website: bit.ly/IBMTechU2016Munich

Watch. Listen. Learn more about IBM technologies.
ibm.com/training/events

Bibliography

- WAS z/OS Home page :
http://www.ibm.com/software/webserver/appserv/zos_os390/
- Knowledge Center for traditional WAS z/OS and WOLA: [“tdat_useola”](#)
- Knowledge Center for WAS Liberty z/OS and WOLA:
[“twlp_dat_useola”](#)
- WebSphere on z/OS - Optimized Local Adapters (WOLA) (REDP4550)
- IBM Washington Systems Center (WSC) White Papers WP101490
<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101490>
See in particular the WOLA Executive Overview, Overview and Usage and WOLA Quick Start Guide.
- Still confused about WOLA? See
<http://www.youtube.com/user/WASOLA1>