WOLA Architectural Considerations

IBM Advanced Technical Skills (ATS)

A true partnership:

- WAS z/OS Support Team
- CICS Support Team
- IBM Software Group, WebSphere Application Server z/OS Development

Don Bagwell
dbagwell@us.ibm.com

Other Sessions

Room	Day	Time	Title	Speaker
312	Monday	12:15	Lab	Stephen
203	Monday	4:30	WebSphere: What's New?	Follis
203	Wednesday	9:30	WebSphere 101	Houde / Stephen
201	Wednesday	1:30	Introduction to IBM Support Assistant (ISA)	Hutchinson
200	Wednesday	3:00	WebSphere Process Manager and Business Process Manager Configuration	Hutchinson
200	Wednesday	4:30	OSGi/JPA/Batch Feature Packs	Follis / Bagwell
203	Wednesday	6:00	WebSphere for z/OS: I'm no longer a dummy but..	Bagwell
310	Thursday	8:00	WOLA Application Designs	Bagwell
310	Thursday	9:30	Security Architecture: How Does WebSphere Play?	O'Donnell
310	Thursday	11:00	WAS on z/OS High Availability Considerations	Bagwell
200	Thursday	12:15	Staged Application Development in a WebSphere ND Cluster	Loos
310	Thursday	1:30	WAS on z/OS and WLM Interactions	Follis

Agenda

- Overview of WOLA
- The "Inbound" vs. "Outbound" Concept
- CICS
  - Outbound
  - Inbound
- Non-CICS ... Batch, USS
  - Outbound
  - Inbound

Considerations we'll cover:

- Programming
- Security
- Transaction
- Performance

Techdoc ...

WOLA Techdoc Page

`ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101490`

Design and Planning Guide ← The source for much of the information you'll see in today's presentation

Native APIs COBOL Primer ← Many of the coding principles are spelled out in this "Primer"

As well as other presentations and white papers on WOLA

And don't overlook the InfoCenter ... very good information as well

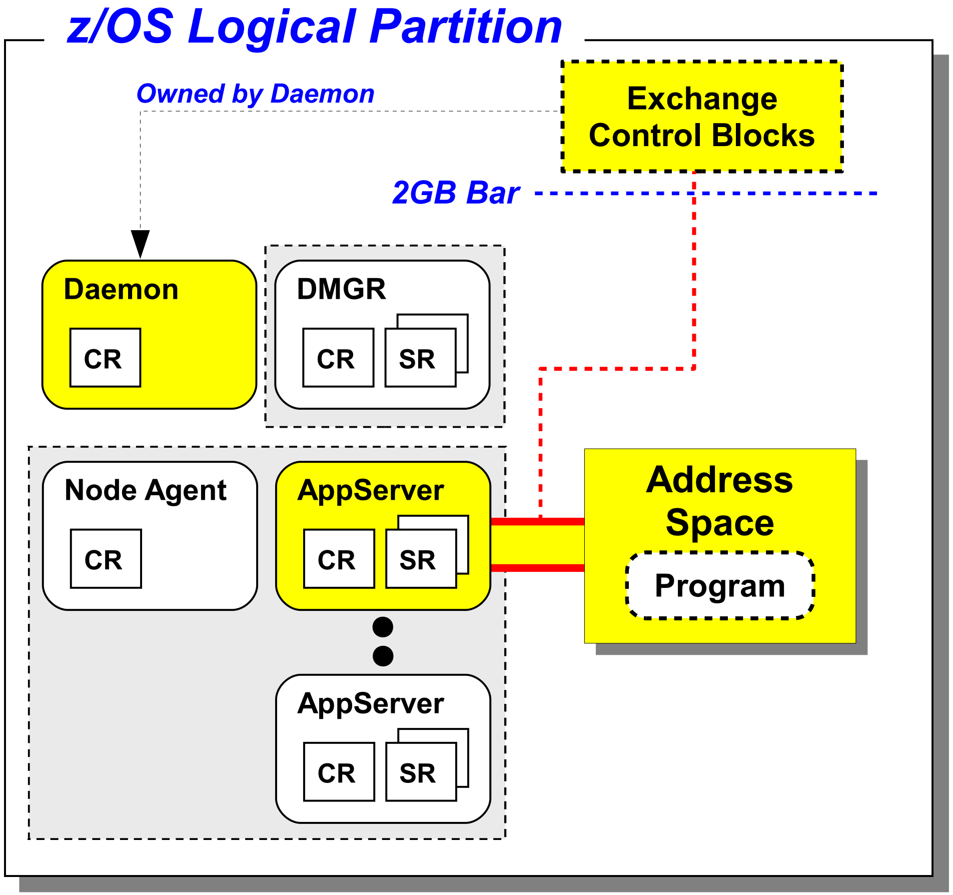
Overview ...

Overview of WOLA

Establishing a baseline of key terminology and concepts

Basic Framework of WOLA

WOLA is at its heart a cross-memory byte array exchange mechanism:



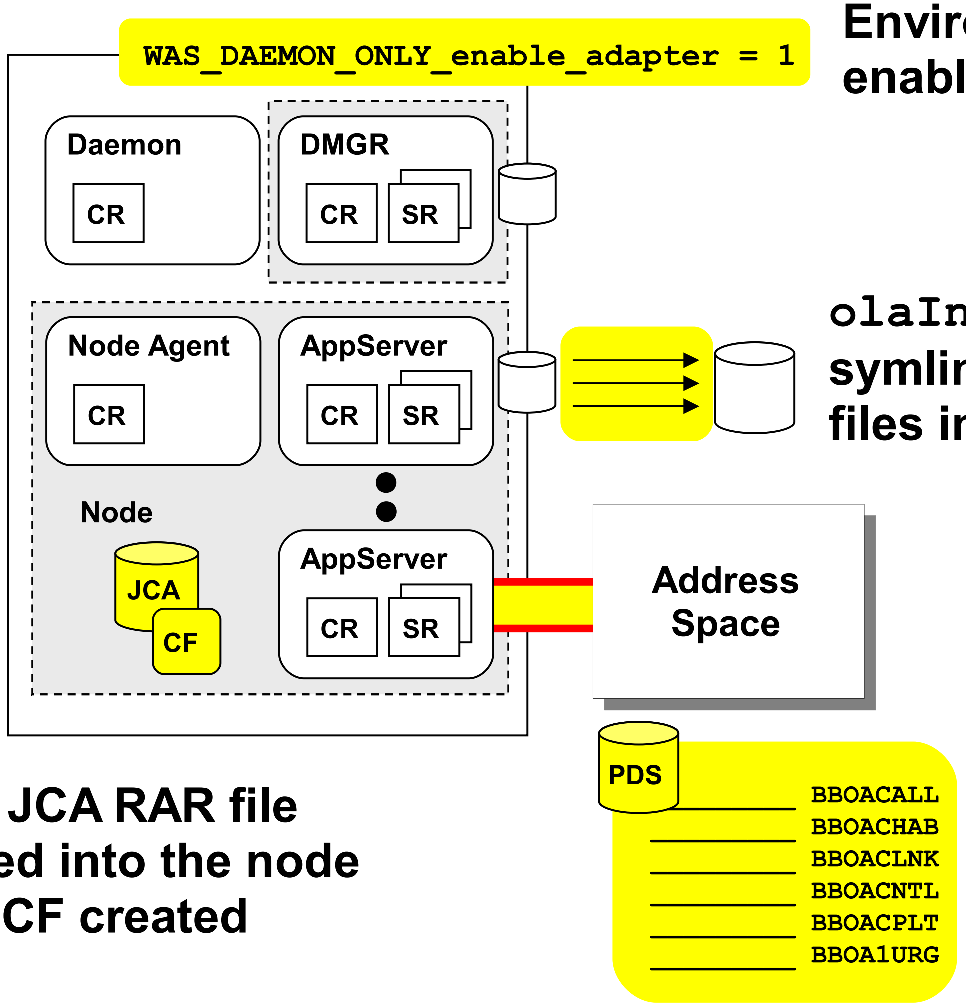
- **Address space to address space**
- **Same LPAR only**
- **CICS, Batch, USS and ALCS**
- **Bi-directional**
- **The Daemon plays a key role in this**
- **Not "transparent" to application ... but there are ways to minimize as we'll see**
- **WOLA *itself* does not care about the layout, format or contents of the exchange ... it's a byte array**
- **The parties at either end of the "pipe" *do* care about layout, format and contents**

Much more to discuss ...

Key enablers ...

Key Pieces that Need to be in Place

Some basic environment setup work needs to be in place for things to work. This chart summarizes ... Techdoc provides details.



Environment variable to enable function for the cell

`olaInstall.sh` run to create symlinks from node to WOLA files in SMP/E file system

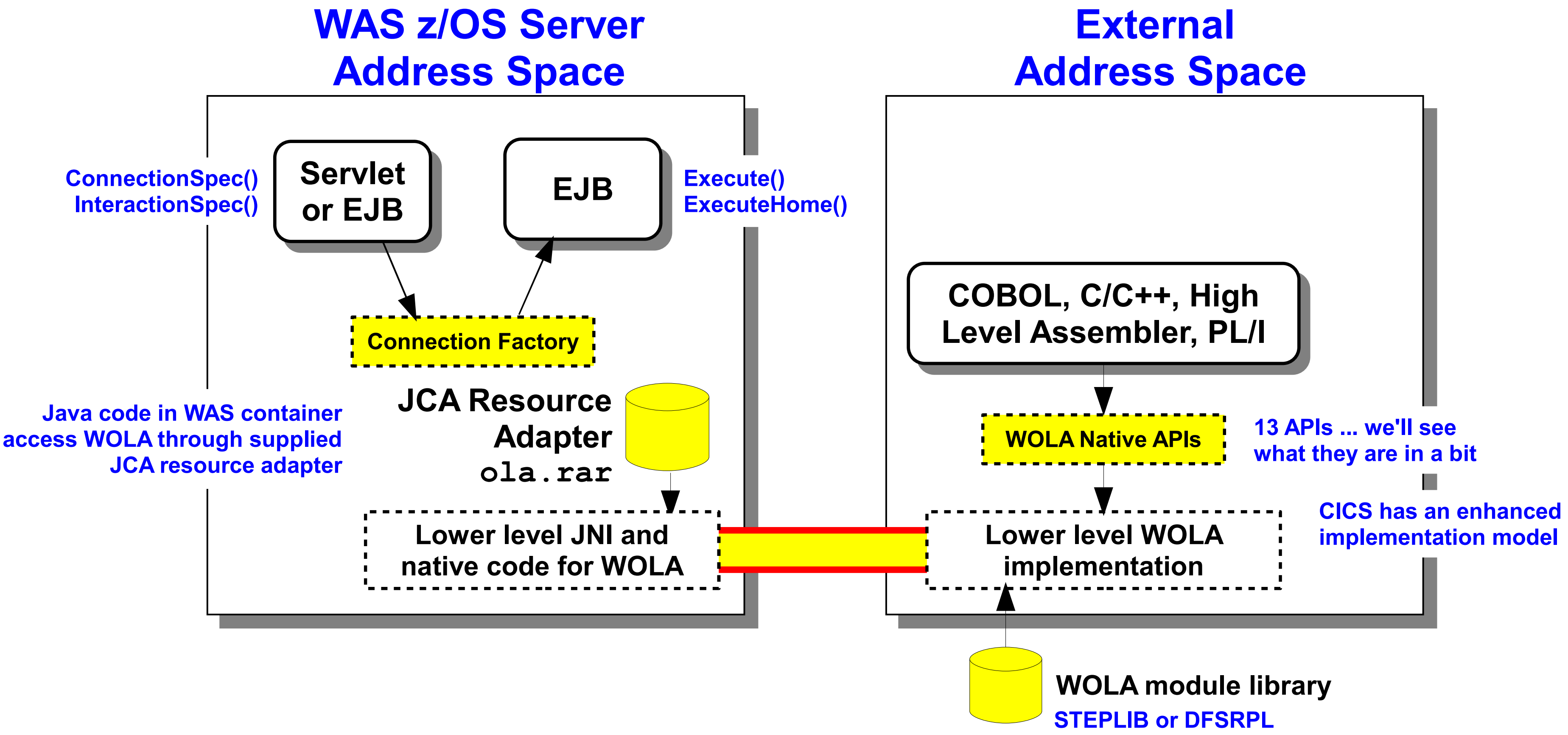
WOLA JCA RAR file installed into the node with a CF created

API modules copied out to PDS load library

Programming overview ...

Programming Considerations *Overview*

More details coming later in presentation


WAS z/OS Server Address Space
External Address Space
ConnectionSpec()
InteractionSpec()
Servlet or EJB
EJB
Execute()
ExecuteHome()
Connection Factory
JCA Resource Adapter
ola.rar
Java code in WAS container access WOLA through supplied JCA resource adapter
Lower level JNI and native code for WOLA
COBOL, C/C++, High Level Assembler, PL/I
WOLA Native APIs
13 APIs ... we'll see what they are in a bit
Lower level WOLA implementation
CICS has an enhanced implementation model
WOLA module library
STEPLIB or DFSRPL

The programming is not difficult ... but it may be unfamiliar

Java interfaces with standard JCA methods

WOLA not transparent ... but that does not mean all programs are affected

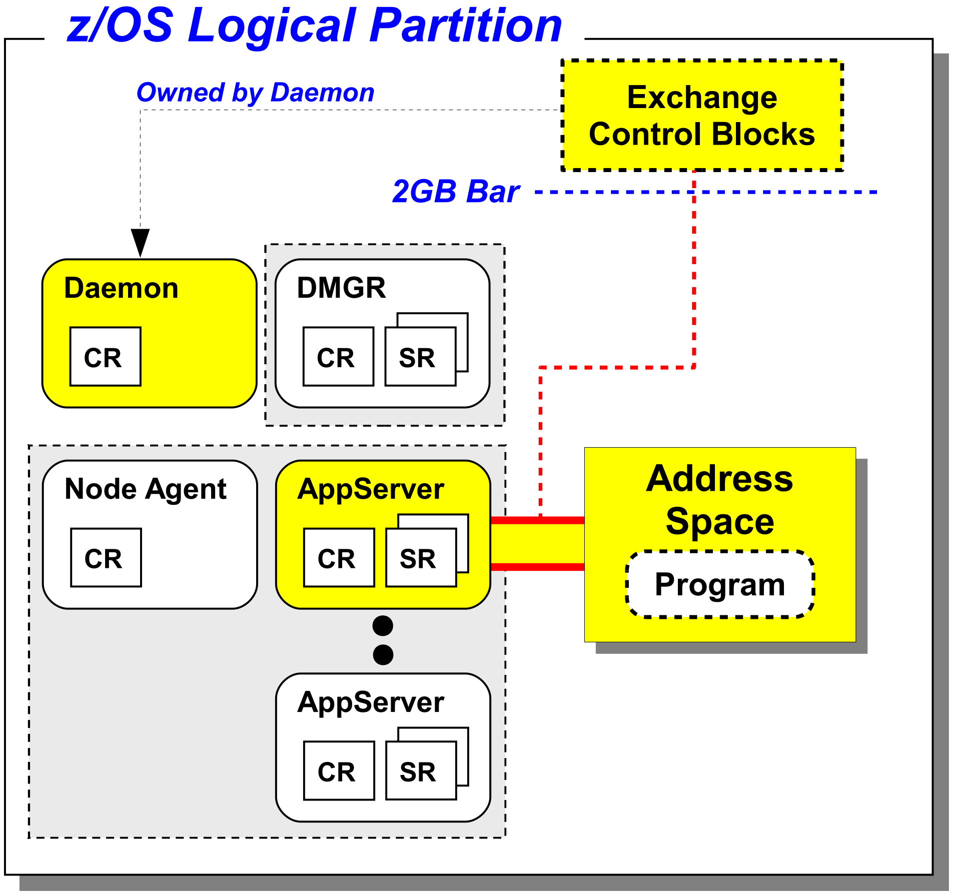
Inbound / Outbound ...

Inbound vs. Outbound

The starting point for any discussion of specifics

A Registration into WAS Must Be in Place

Before any exchange across the WOLA "pipe" can be made, the WOLA pipe has to be *established*. That's called "registering" ... and it's *always* done by the external program:



Starting State

- No WOLA connection exists
- WAS application server is up and running
- WAS Daemon server stands ready to accept registration request

Registration Phase

- The external address space program initiates an action that results in the BBOA1REG API being executed
- That API names the cell, node and server short names.
- That API also provides information about the number of connections to create in the connection pool
- That API also provides information on security and transactionality
- The registration carries a *name.*

Result

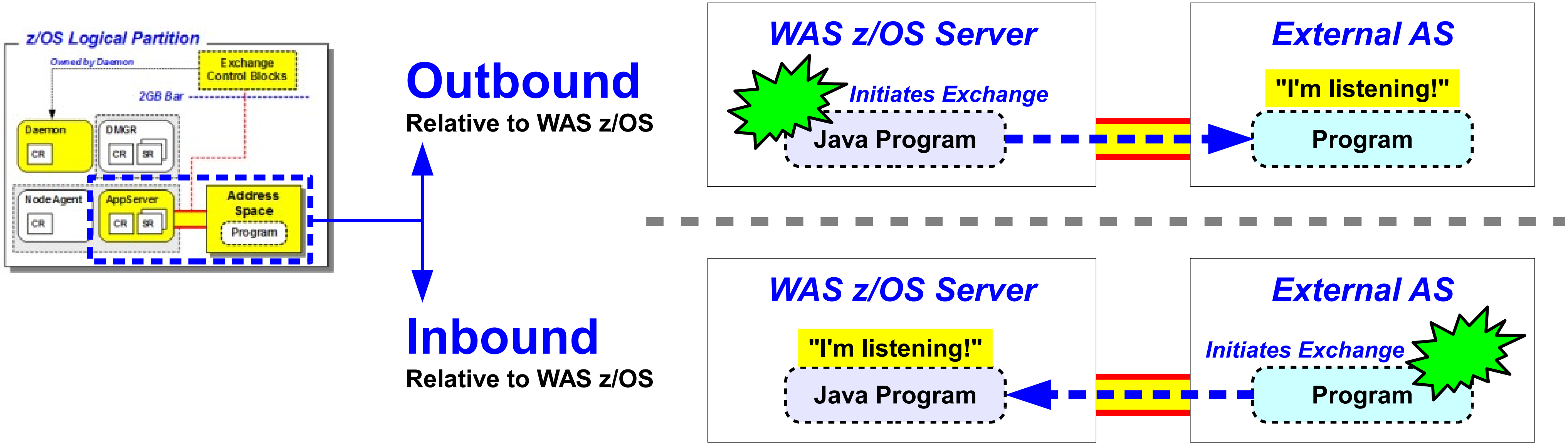
- Daemon establishes control block structure above the 2GB line
- External address space connects into WAS "local comm" structure
- WOLA pipe built between external AS and WAS application server controller region

Now programs are ready to communicate across the WOLA registration

Who initiates? ...

The Next Question is: Who *Initiates* the Exchange?

This is what differentiates "Inbound" vs. "Outbound" ... which side of the WOLA connection *initiates the exchange*:



Outbound
Relative to WAS z/OS

WAS z/OS Server
Initiates Exchange
Java Program

External AS
"I'm listening!"
Program

Inbound
Relative to WAS z/OS

WAS z/OS Server
"I'm listening!"
Java Program

External AS
Initiates Exchange
Program

Drawing this distinction is important because it helps us focus on the APIs that get used. There are 13 APIs ... not all need to be used.

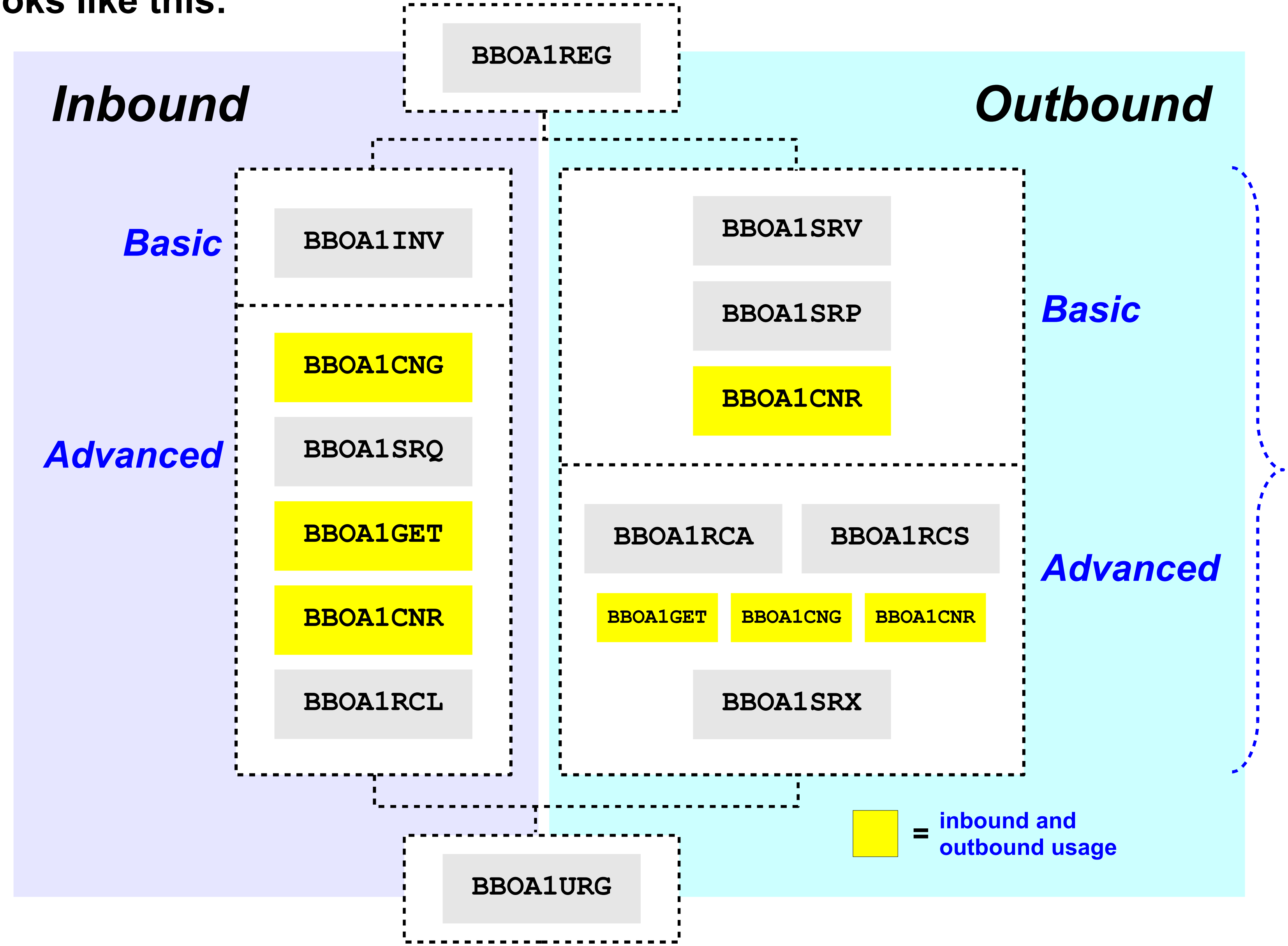
It's also important because something has to be ready to receive the initiation request coming over WOLA. Different ways to accomplish that.

When CICS ... transactionality and security are determined by this.

API picture ...

The APIs Organized Around Inbound / Outbound

Looks like this:



🟨 = inbound and outbound usage

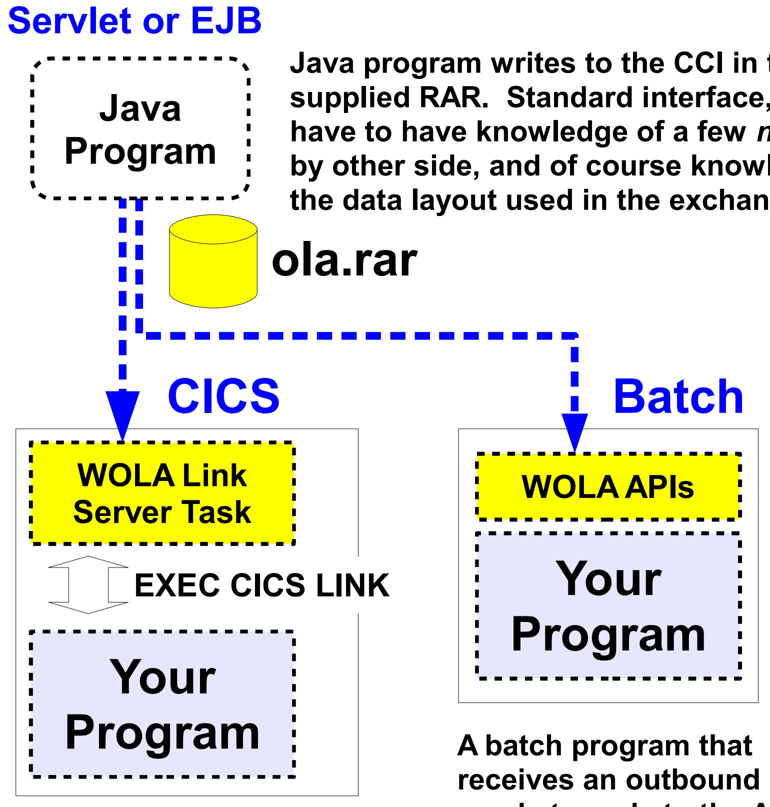
And perhaps no coding of APIs at all for outbound if to CICS

Shielding ...

Shielding Programs from WOLA-specific Coding

Here's a few preliminary comments, with details to come later in session ...

Outbound from WAS

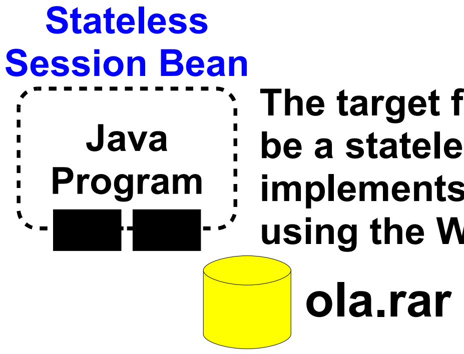


Java program writes to the CCI in the supplied RAR. Standard interface, but it does have to have knowledge of a few *names* used by other side, and of course knowledge of the data layout used in the exchange

Supplied Link Server task shields your CICS programs provided they can be invoked with a LINK. Details coming.

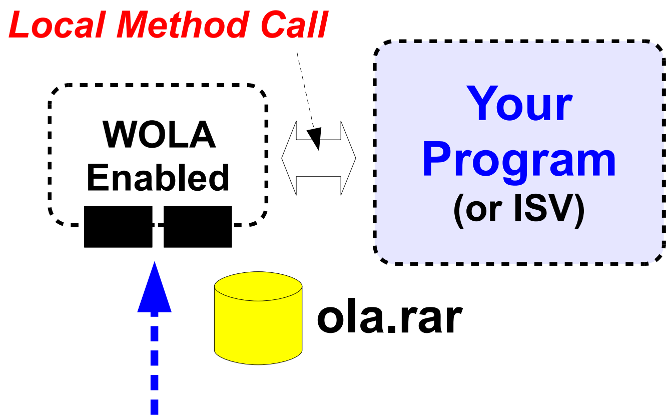
A batch program that receives an outbound call needs to code to the APIs.

Inbound to WAS



The target for an inbound WOLA call must be a stateless session bean that implements Execute() and ExecuteHome() using the WOLA classes

This may not be what you want to do or can do. Solution is to build a "bridge" (or "shim") EJB that simply turns and invokes the target EJB:



CICS ...

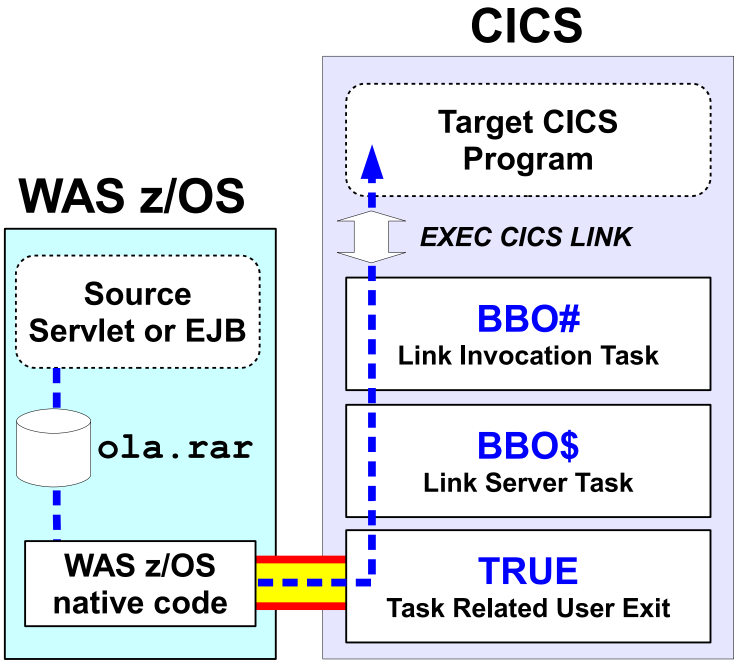
CICS

Evaluating the key architectural considerations for WOLA and CICS

The CICS Inbound and Outbound Model, Summarized

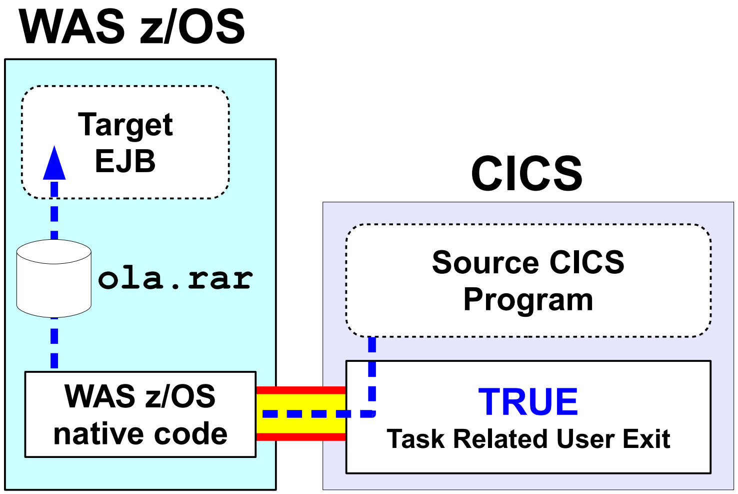
Details will follow:

Outbound



- TRUE provides the essential WOLA infrastructure inside the CICS region.
- The BBO$/BBO# link server function implements the APIs "under the covers" -- makes things simple to use
- No coding to the APIs needed
- The BBO# invocation task performs an EXEC CICS LINK against the named target CICS program
- As long as target CICS program can be invoked with a LINK there's no changes needed to it.

Inbound



- TRUE provides the essential WOLA infrastructure inside the CICS region.
- No BBO$/BBO# needed ... those are functions to receive a call outbound from WAS
- Instead, the source CICS program writes to the WOLA APIs
- This is really just like batch inbound to WAS
- The target program in WAS must be a stateless session bean that implements execute() and executeHome() using the WOLA classes.

Link Server Task ...

Focus in on the BBO$/BBO# Link Server Task (Outbound)

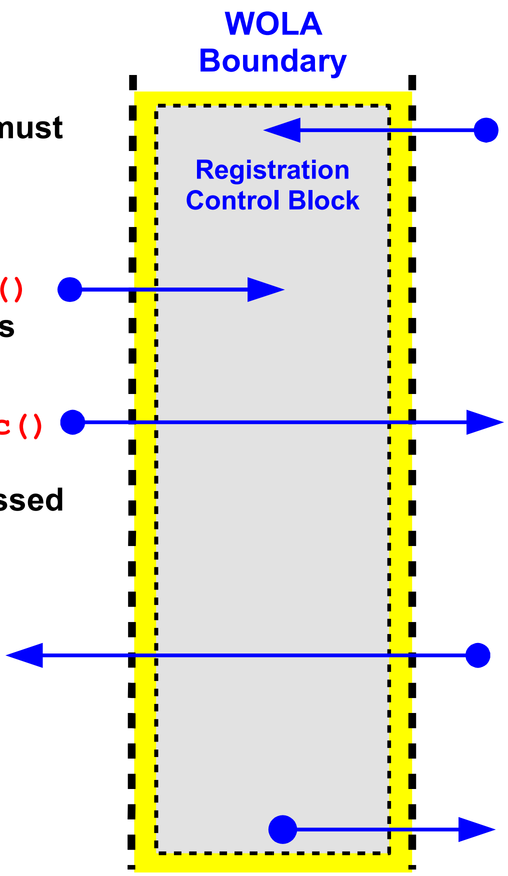
Here's the exchange flow and some of the details behind it:

WAS Daemon and application server must be up and running

Java program uses ConnectionSpec() and names the registration name as its connection point

Java program uses InteractionSpec() and names the CICS program as the "service name" of the target. Data passed as byte array representation of COMMAREA or Channel/Container

Java program receives the response provided it from the underlying WOLA JCA resource adapter



`BBOC START_TRUE`

"BBOC" is a WOLA control program. The TRUE may be started manually or at CICS initialization with PLTPI

`BBOC START_SRVR`

This starts the BBO$ link server task, as well as having that task perform a registration. Parameters include the cell, node and server short along with a registration name

BBO$ link server task receives request and launches an instance of the BBO# invocation task

BBO# invocation task issues an EXEC CICS LINK against the CICS program named as the "service"

CICS program provides return

●
●
●

`BBOC STOP_SRVR`

This unregisters and stops the BBO$ link server task

`BBOC STOP_TRUE`

Stops the WOLA TRUE

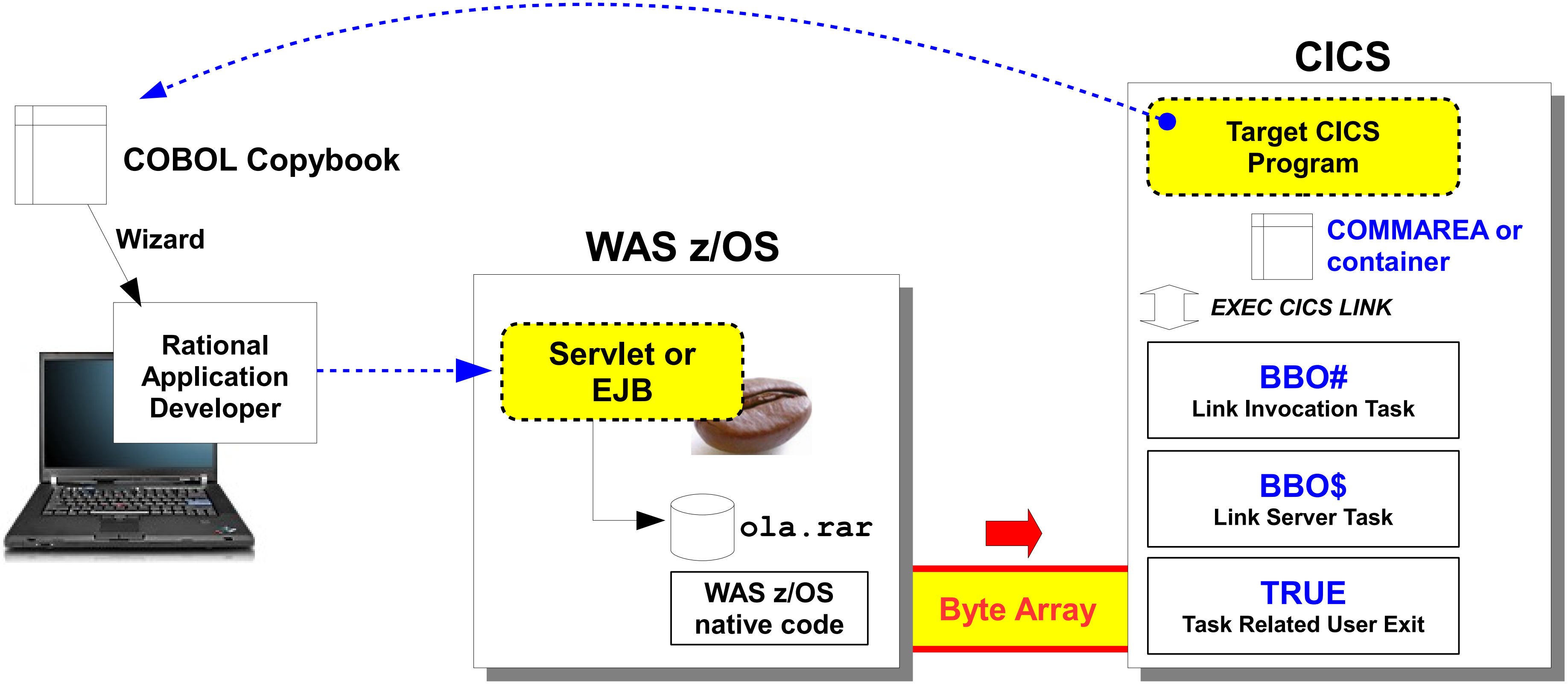
No coding to the native APIs, no modifications to the target CICS program

(provided it can be invoked with an EXEC CICS LINK)

Data layout ...

The Java Program needs to understand the data layout for CICS

WOLA itself sees the data as a byte array. It has no awareness of the data format.



But the two application partners in the exchange do have to know the data format

There's a COPYBOOK import WIZARD in RAD that assists with this

YouTube demonstration -- search on `WASOLA1`
IBM Redbook RedPiece -- `redp4550`

BBOC START_SRVR ...

BBOC START_SRVR and the Parameter Flags

This starts the link server task and initiates a registration into the named Daemon space. The parameters supplied influence things like security and performance:

```
BBOC START_SRVR <parameters>
```

RGN=<name> The registration name. Java-side needs to know this for `ConnectionSpec()`

DGN=<name> The cell short name

NDN=<name> The node short name

SVC=<name> The service name(s) supported ... asterisk (*) means any

SVN=<name> The server short name

MNC=<minimum_number_of_connections> The minimum connections in the connection pool

MXC=<maximum_number_of_connections> The maximum connections in the connection pool

SEC=<yes|no> Determines whether CICS will consider the asserted ID coming from WAS

TXN=<yes|no> For inbound to WAS this determines if transaction propagation takes place

STX=<CICS_link_server_transaction_ID> Overrides default value of BBO$

LTX=<CICS_link_server_invocation_ID> Overrides default value of BBO#

TRC=0|1|2 Trace level

TDQ=<tdqname> Transient data queue for trace data

REU=<yes|no> If SEC=NO, then REU=YES means BBO# invocation tasks re-used

InfoCenter search string: `rdat_cics`

SEC and TXN ...

Outbound -- Security and Transactionality

Influenced by the BBOC parameters SEC and TXN

Transaction

`TXN=<yes|no>`

At the present time WOLA supports "sync on return" only for WAS ⇨ CICS outbound initiated flows.

That limitation is imposed on WOLA by the design of the CICS Task Related User Exit.

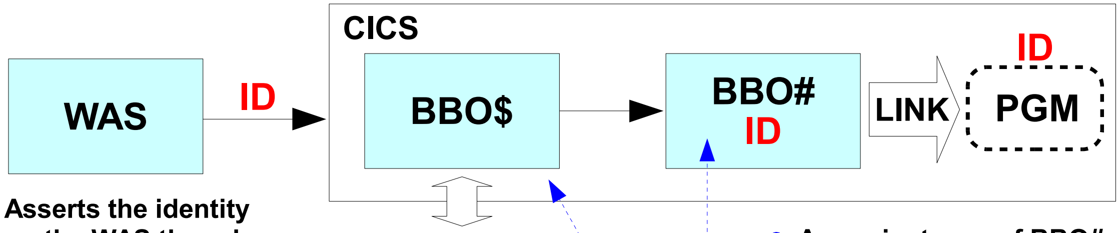
* * *

For CICS ⇨ WAS *inbound* initiated flows `TXN=YES` provides propagation of TX. WAS then participates in the CICS global transaction 2PC processing.

Security

`SEC=<yes|no>`

`SEC=YES`



Asserts the identity on the WAS thread.

Either the servant ID or user identity depending how you have WAS configured

BBO$ will check with SAF every invocation

- A new instance of BBO# invocation task started for every invocation. REU=Y not permitted

- The ID BBO$ runs under needs SURROGATE to start the BBO# task and pass the ID

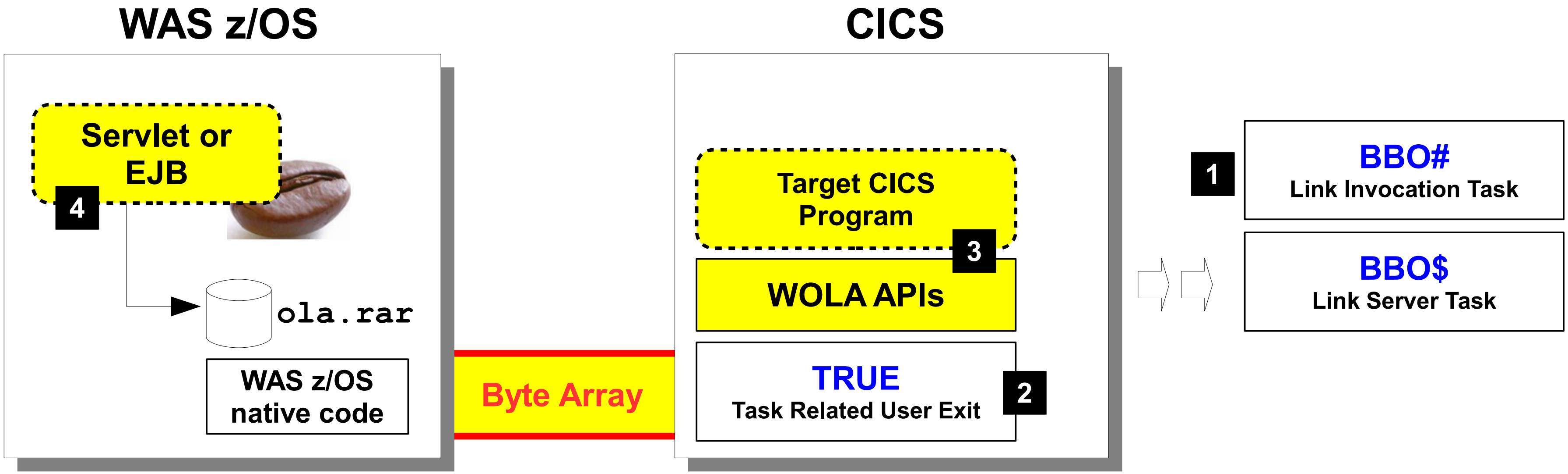
`SEC=NO`

- **No identity asserted**
  For same-LPAR high-performance applications that may be acceptable
- **No SAF checking**
  Acceptable if same-LPAR a trusted domain. Aids performance.
- **Allows REU=Y ... instances of BBO# link invocation tasks maintained and re-used**
  Performance

Bypassing link server ...

Outbound -- Maximum Performance

If you're looking to squeeze every drop of throughput ...



1. Do *not* use BBO$/BBO#

BBO$/BBO# provide ease-of-use and flexibility, but at the cost of some overhead. If maximum throughput is needed, do not start the link server tasks

2. Still need TRUE

This is what provides the essential WOLA infrasturcture support inside of CICS. Need this in any event.

3. Code program directly to the WOLA APIs

- Register using BBOA1REG API
- SEC=N to minimize SAF checking
- Provide a "service name" on the registration
- "Host a Service" using BBOA1SRV or primitive (more in a bit)
- Multi-thread and async operatons (more in a bit)

4. Java program similar to before:

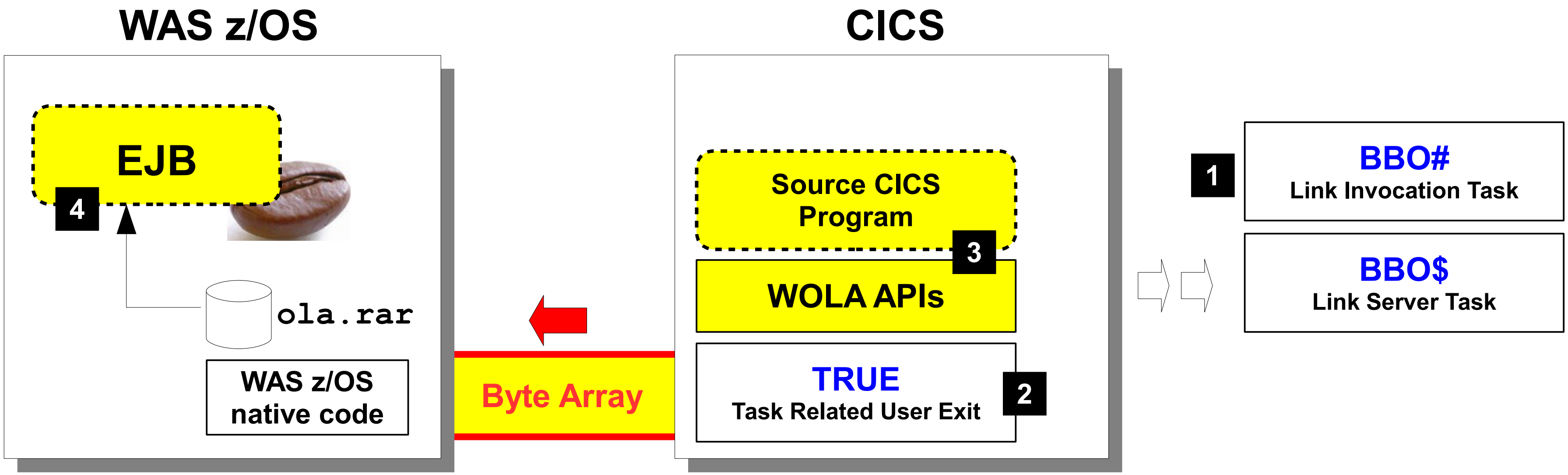
- Still use ConnectionSpec() and InteractionSpec()
- Name the "service" the CICS program used on registration
- Multi-thread and use concurrent connections (more in a bit)

API coding considerations just like "batch" ... we'll cover details in that section

Inbound ...

Inbound -- Need to Code to the APIs; BBO$/BBO# Not Used

Inbound implies CICS program is initiating the exchange:



1. Do *not* use BBO$/BBO#

The link server task is an outbound construct. For inbound to WAS the program initiates using one of the WOLA APIs.

2. Still need TRUE

This is what provides the essential WOLA infrasturcture support inside of CICS. Need this in any event.

3. Code program directly to the WOLA APIs

- Register using BBOA1REG API
  - SEC=Y ... CICS region ID or application user ID
  - Set `ola_cicsuser_identity_propagate=1` WAS variable
  - TXN=Y ... WAS participates in CICS global tran, 2PC with RRS
- Using BBOA1INV or one the primitives (more in a bit)
- "Service name" is the EJB home interface JDNI

4. Java program requirements

- Must be a stateless session bean
- Execute() and ExecuteHome() implemented with WOLA classes

API coding like "batch" ... we'll cover details in that section

CICS summary ...

CICS Support -- Summary

	Outbound	Inbound
WOLA TRUE installed/enabled	Required	Required
BBO$/BBO# Link Server task used	Optional (ease of use vs. performance)	Not Applicable
Java Programming	Servlet or EJB Code to JCA methods of WOLA adapter	Stateless session bean. `Execute()` and `ExecuteHome()` implemented with WOLA classes
Registration	Required. Use `BBOC` or use `BBOA1REG`	Required. Use `BBOC` or use `BBOA1REG`
Native API Programming	If using link server task, then none. Otherwise, program must "host a service" (`BBOA1SRV` or primitive combination)	If using link server task, then none. Otherwise, program must "host a service" (`BBOA1SRV` or primitive combination)
Security	If SEC=Y, then WAS asserts ID of execution thread	If SEC=Y, then CICS asserts region ID or application user
Transaction	Sync-on-return only	If TXN=Y then 2PC

Non-CICS ...

Non-CICS ... Batch/USS

With a particular focus on the APIs and key coding constructs

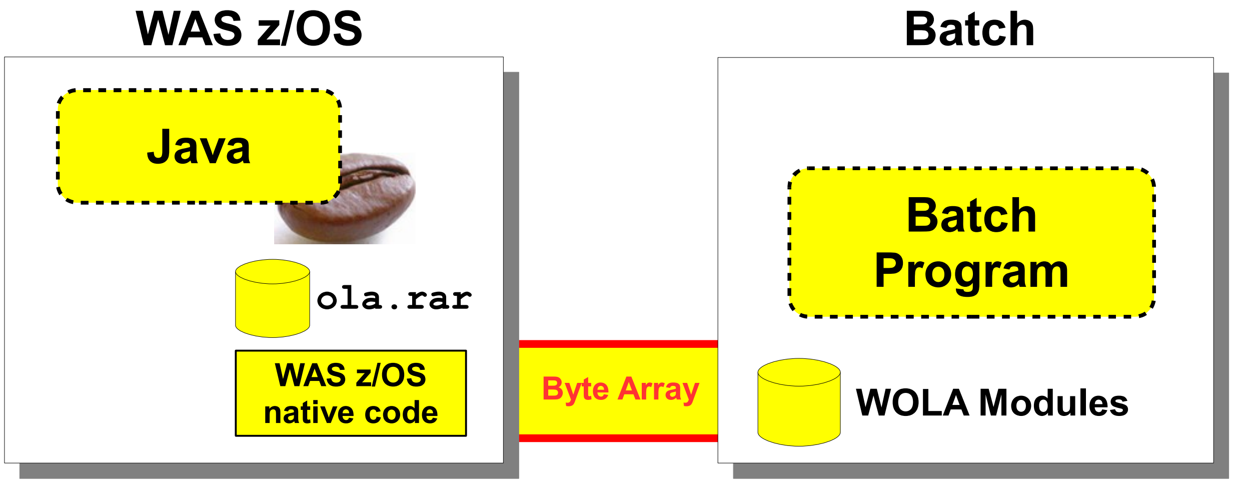
Overview of the Key Considerations

There's a handful of things to keep in mind:

Same essential concept of "inbound" and "outbound" -- who initiates the exchange

Java issues comparable to CICS; that is, servlet or EJB outbound, stateless EJB inbound.

The WAS/WOLA infrastructure pieces need to be in place



The batch program must perform the registration -- inbound or outbound

The WOLA modules must be in a PDS accessible by the batch program

If the programs can multi-thread then potential exists for parallel connections in use across WOLA

Do you want the program to operate synchronously or asynchronously

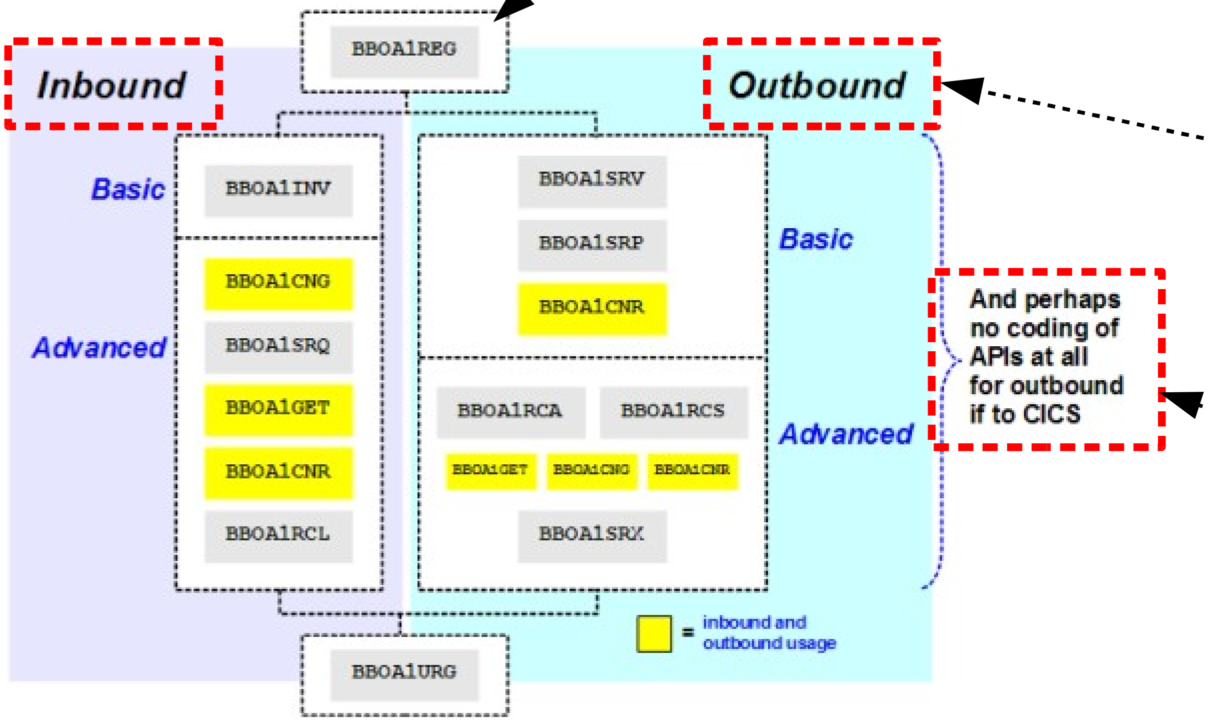
These two influence which APIs will be used.
The key objective is performance.

API categorization ...

Reminder of Native API Categorization

We saw this earlier in the presentation:

We saw that registering is a key first step in all cases. And it's always done by the external address space. Perhaps "hidden" with a `BBOC START_SRVR` command, but it still must take place.



We emphasized the key concept of "inbound" vs. "outbound" processing. The APIs you use in the external address space are determined by this.

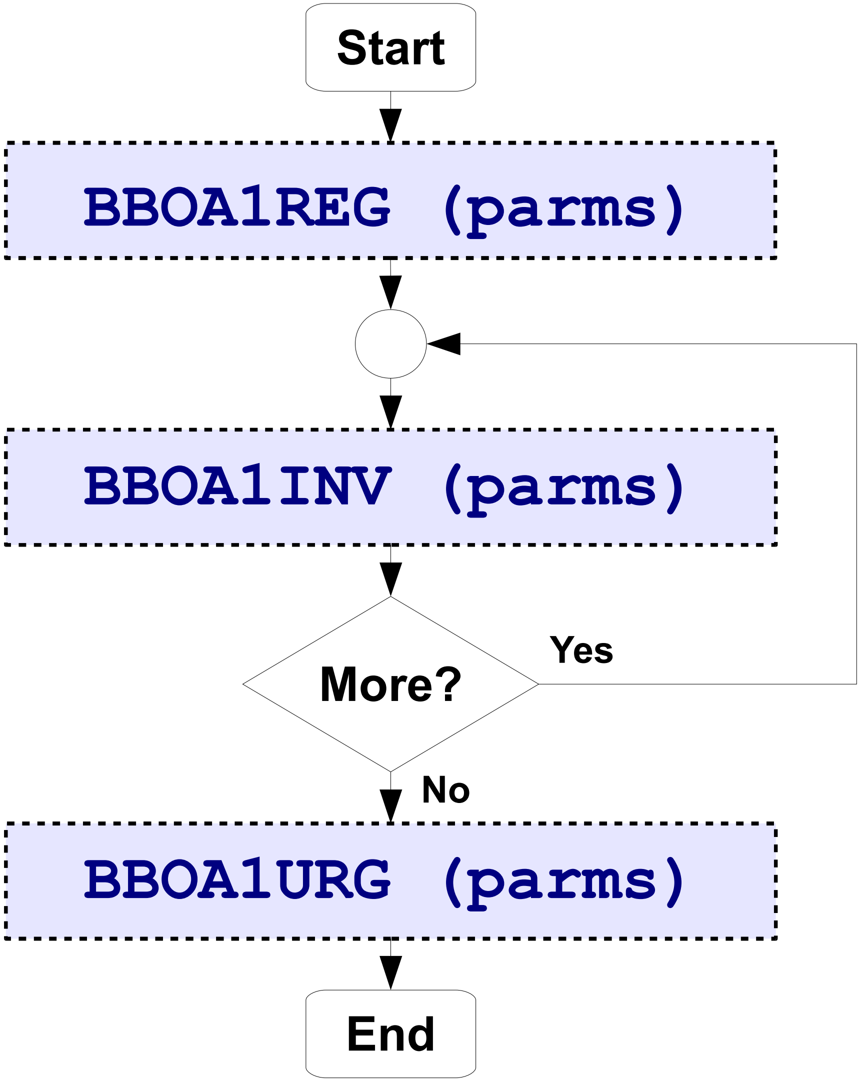
We saw how with outbound CICS it's possible to avoid coding APIs altogether. That's because the supplied link server task hides all that from you.

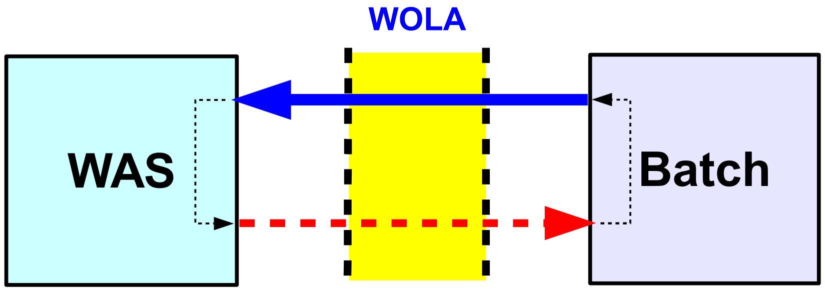
Now we'll explore the APIs in a bit deeper detail, and see about this "basic" vs. "advanced" concept inherent in the APIs.

Simplest model ...

A Starting "Comfort" Chart

An **inbound** program might be as simple as this:





Simple and Easy

Effective ... very fast

But ...

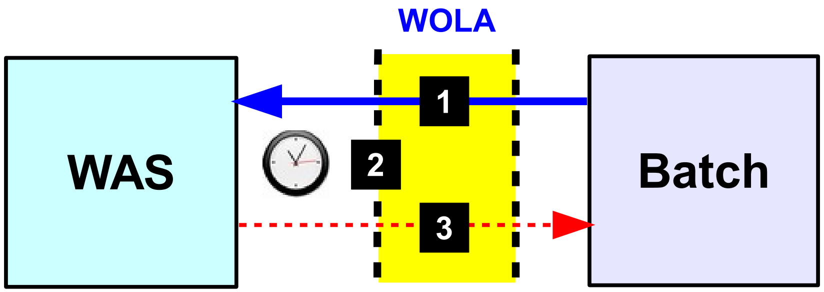
- **Inbound ... outbound is a bit more involved**
  - Some of the connection management is the batch program's responsibility
- **Synchronous ... which means batch thread waits for WAS to return**
  - Very easy to under-utilize the WOLA mechanism if there's a lot of synchronous waiting on in-server processing to complete

Synchronous / Asynchronous ...

Synchronous vs. Asynchronous

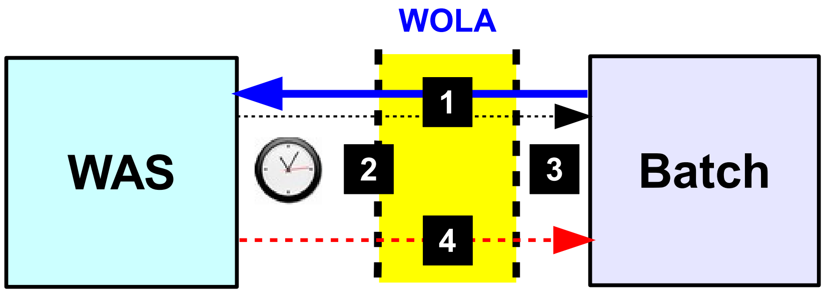
The APIs allow both. In general, synchronous is simpler. But asynchronous allows for potentially greater throughput:

Synchronous



1. Batch program calls WAS program
2. WAS program processes request. Program control is held from batch processing thread until request returns.
3. WAS program responds

Asynchronous



1. Batch program calls WAS program. Program control is returned to batch thread immediately.
2. WAS program processes request.
3. Batch program free to do other work or employ other WOLA connections (more on connections next chart)
4. WAS program responds at some future point.

The "basic" APIs operate synchronously. It's a simpler model.

The "advanced" APIs (sometimes called "primitives") are finer-grained subsets of the basics which allow asynchronous activity. *But that implies your program goes back at some point and checks to see if a response has been received.*

Connections ...

Connections within the Registration Pool

Two of the parameters on the `BBOA1REG` registration API determine the minimum and maximum connections provided in the registration:

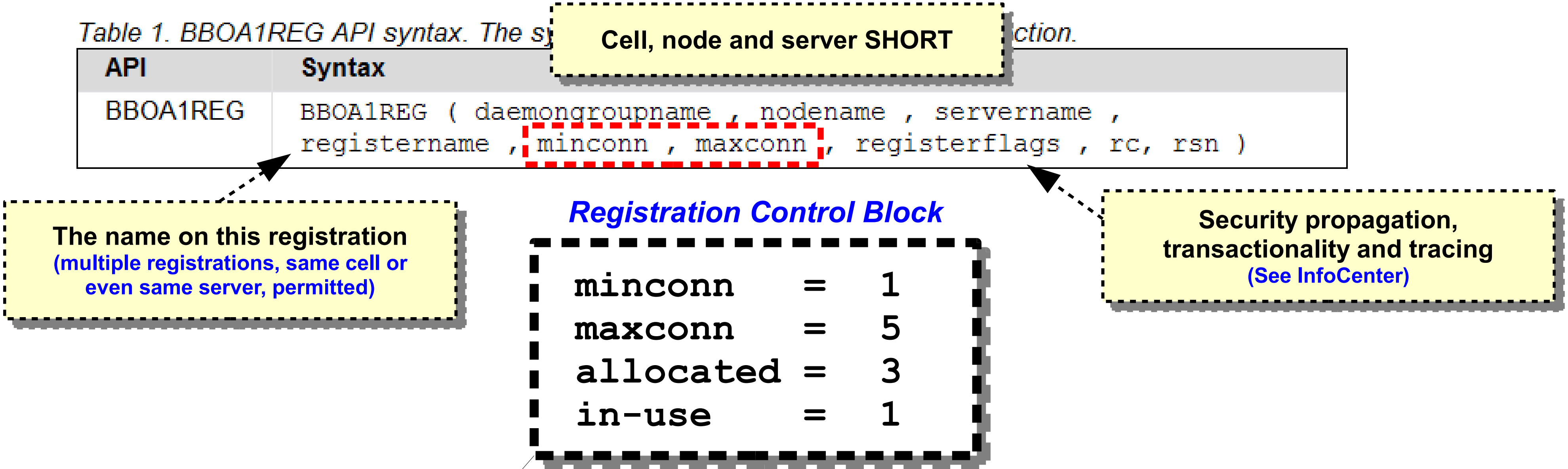


Table 1. BBOA1REG API syntax. The sy ction.

API	Syntax
BBOA1REG	`BBOA1REG ( daemongroupname , nodename , servername , registername , minconn , maxconn , registerflags , rc, rsn )`

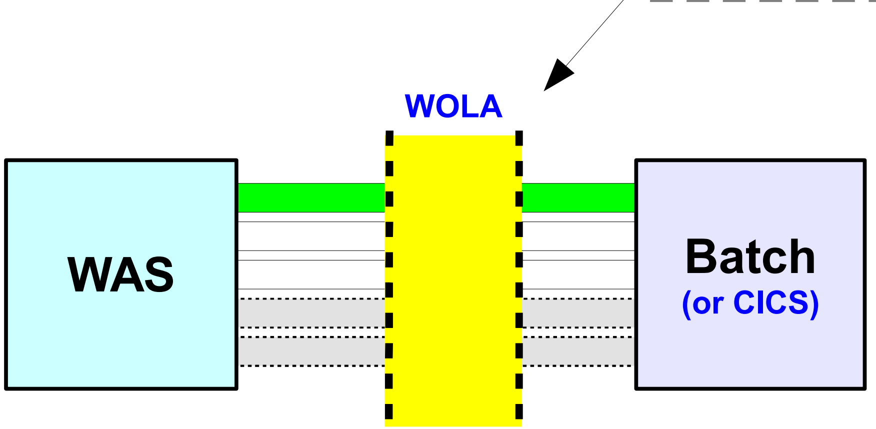
Cell, node and server SHORT

The name on this registration (multiple registrations, same cell or even same server, permitted)

Security propagation, transactionality and tracing (See InfoCenter)

Registration Control Block

```
minconn   =  1
maxconn   =  5
allocated =  3
in-use    =  1
```



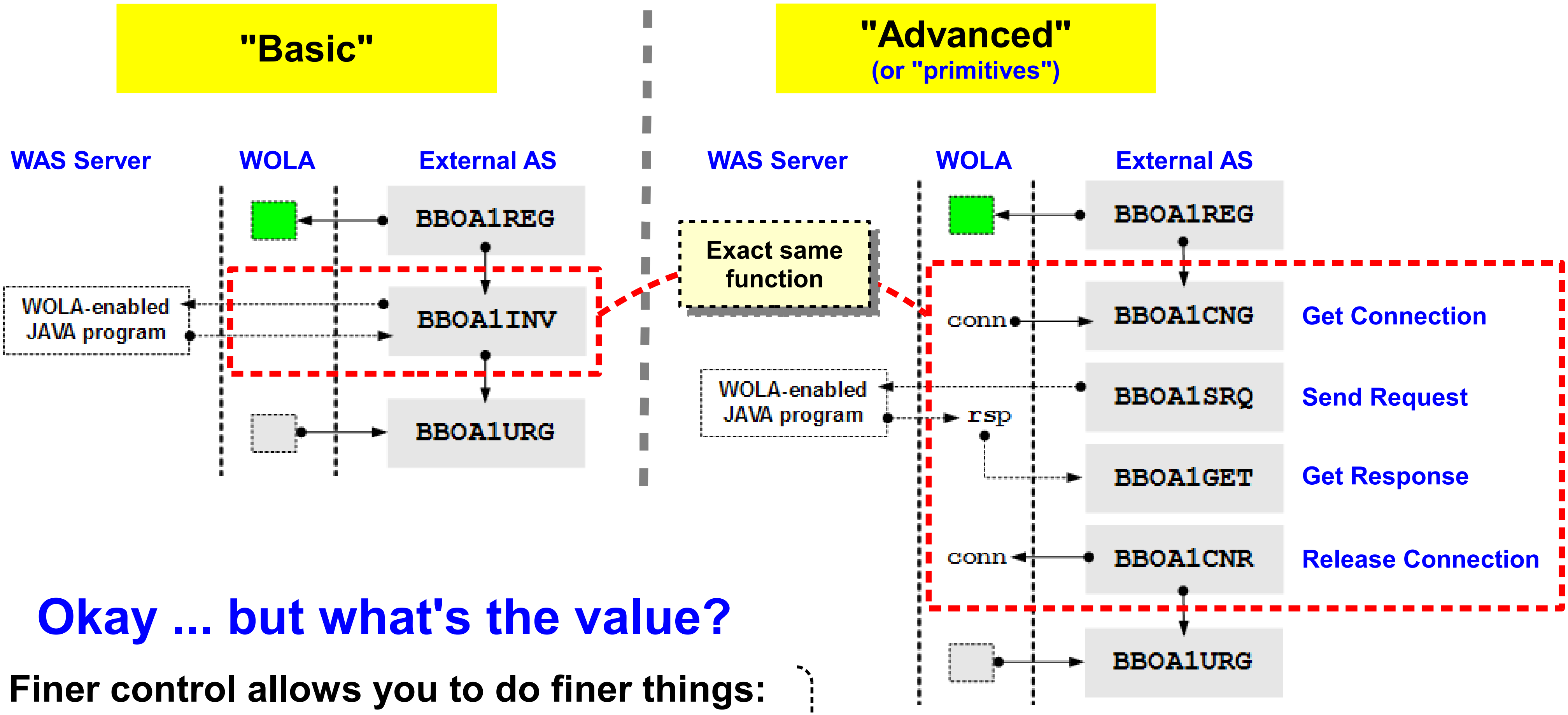
- minconn is the number of connections allocated at registration
- maxconn is the limit of allocations on this registration
- in this example 3 connections have been allocated
- one connection is currently in use
- two connections are allocated and available
- two more could be allocated if needed
- RC=8, RSN=10 if maximum connections occupied

InfoCenter search string: `cdat_olaapis`

Primitives ...

Explore BBOA1INV vs. Primitives to do Same Function

This information is from the documents in the WP101490 Techdoc:



"Basic"

WAS Server

WOLA

External AS

WOLA-enabled JAVA program

BBOA1REG

BBOA1INV

BBOA1URG

Exact same function

"Advanced"
(or "primitives")

WAS Server

WOLA

External AS

WOLA-enabled JAVA program

conn

rsp

conn

BBOA1REG

BBOA1CNG

Get Connection

BBOA1SRQ

Send Request

BBOA1GET

Get Response

BBOA1CNR

Release Connection

BBOA1URG

Okay ... but what's the value?

Finer control allows you to do finer things:

- BBOA1SRQ allows for synchronous or asynchronous
- Get a connection and re-use it many times
- Get a pool of connections and multi-thread over it

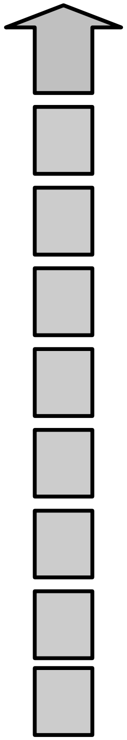
These sorts of things get to the question of performance ...

Performance ...

WOLA Performance ... Heavily Generalized

Two key conceptual points to be made:

Finer Control = Performance
(if done properly)



Greater Performance:

- Multi-threaded
- Concurrent multi-connections
- Tune user threads to connections
- Hold and re-use connections
- Asynchronous
- Large messages
- No security propagation
- No transactional propagation
- If outbound CICS, bypass CICS link server task

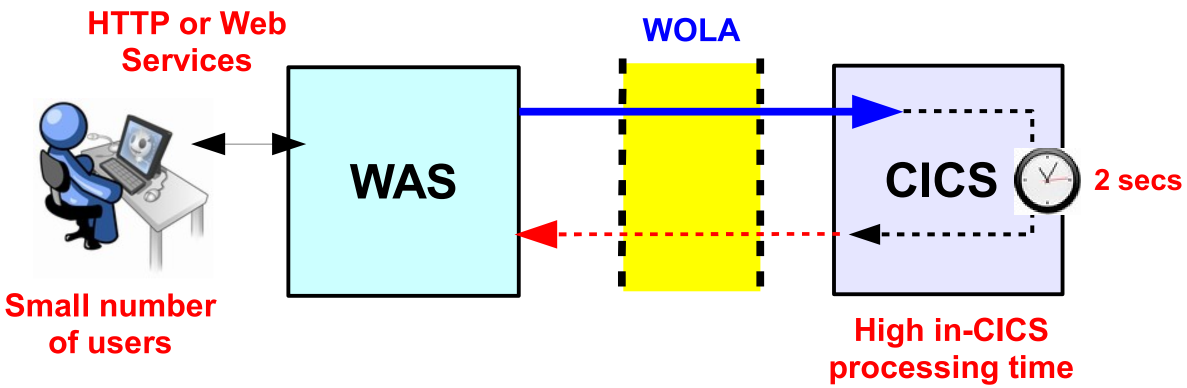
Lesser Performance:

- Single thread
- Synchronous
- Small, chatty messages
- Security checking
- Transactional
- CICS Link Server Task

Trade-off between simplicity and ease of use and performance through more sophisticated usage of programming

Utilize Full Capacity

Here's an example of under-utilizing WOLA:



A user in this example may not see much benefit from WOLA vs. another connector technology.

But that's because the WOLA-time is such a very small percentage of total time.

The greater the utilization of WOLA capacity, the greater the *relative benefit* you'll see.

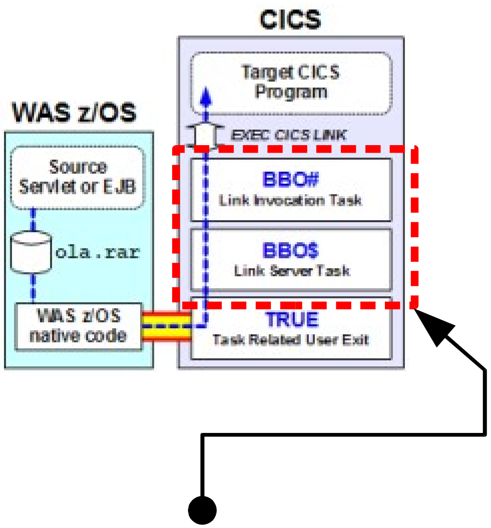
InfoCenter search string: `cdat_perfconsid`
Also WP101490 Techdoc

Outbound ...

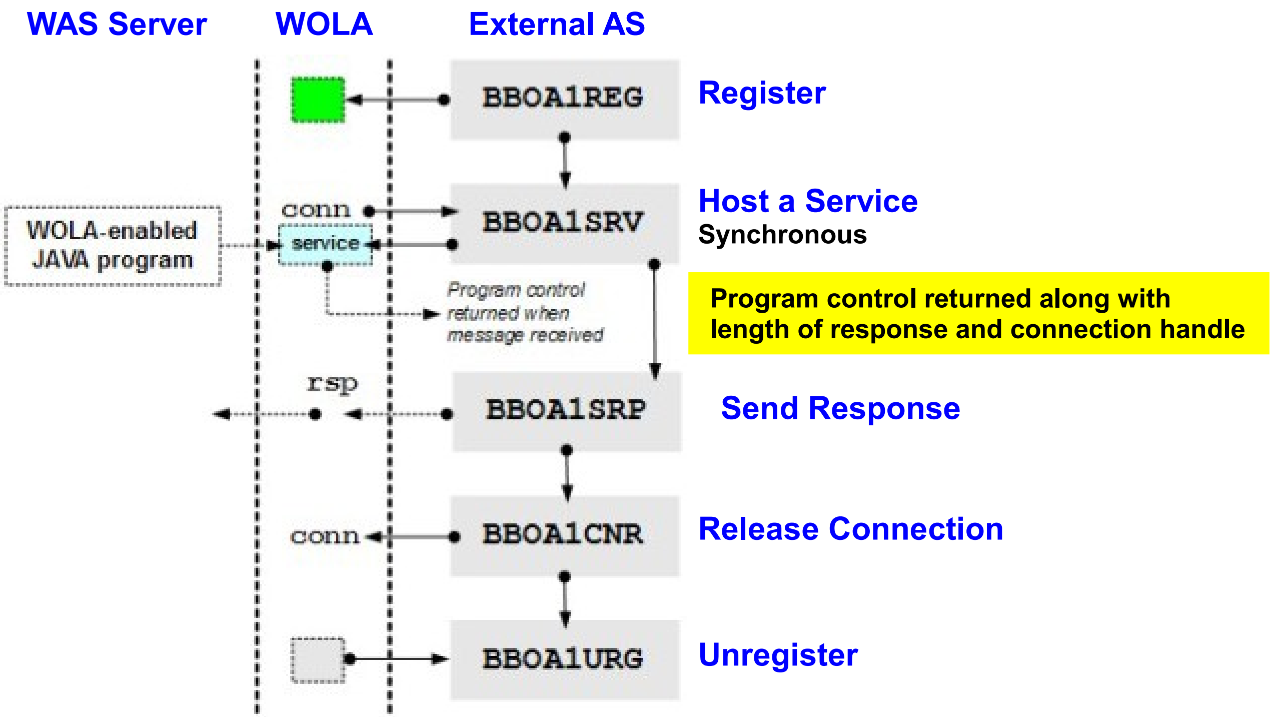
Outbound to Batch (or CICS with bypass of Link Server Task)

The issue here is that you must have the external program in a *listen state*. The `BBOA1SRV` API is for that purpose -- "hosts a service":

Recall the CICS outbound scenario



The BBO$/BBO# link server task function was serving as the "listener" for calls coming from WAS. It used the API function "under the covers"



Program control returned along with length of response and connection handle

If asynchronous is desired, then `BBOA1RCA` with `BBOA1GET` ... allows your batch program to go do other work while WAS processes the inbound request.

The WP101490 "Primer" illustrates all of this in detail

Batch summary ...

Batch Summary

	Outbound	Inbound
Registration	Required. Use `BBOC` or use `BBOA1REG`	Required. Use `BBOC` or use `BBOA1REG`
Native API Programming	Need to "host a service" using `BBOA1SRV` or the primitives.	`BBOA1INV` or the primitives
Java Programming	Servlet or EJB Code to JCA methods of WOLA adapter	Stateless session bean. `Execute()` and `ExecuteHome()` implemented with WOLA classes
Security	No security assertion outbound from WAS to batch; with CICS yes: the same model as with link the BBO$/BBO# link server task	If batch then the ID of the job; if CICS, then region or application thread userid.
Transaction	None for batch; with CICS then same as before: sync-on-return	None for batch; for CICS then same as before: 2PC

Reminder: WP101490 Techdoc!
That has quick-search tags for InfoCenter

Overall summary ...

Overall Summary

Functionality

- Cross-memory single-LPAR byte area low-overhead exchange mechanism
- Inbound and outbound; CICS, Batch, USS and ALCS (watch this space for future cool stuff ☺)

Applicability

- Very well suited for inbound to WAS where other solutions may impose unacceptable overhead
- Excellent solution for high-speed batch interchanges
- Outbound to CICS for very large message sizes and where particular attributes of CTG not indicated

Programming

- Non-Java side: C/C++, COBOL, High-Level Assembler, PL/I
- Native APIs used as illustrated earlier and in WP101490 Techdoc
- Java side: code to CCI methods of supplied JCA adapter

Security

- Security propagation inbound and outbound is possible, depending on the case (see summaries)
- Region ID or Thread ID, inbound/outbound with CICS

Transaction

- Two-phase commit inbound to WAS from CICS using RRS as syncpoint coordinator
- One-phase (sync-on-return) outbound WAS to CICS due to present limitation in TRUE architecture

Performance

- "Out of the box" basics provides very good performance
- Potential exists to tune even further using programming primitives as illustrated earlier
- WOLA will show greater and greater relative performance to other technologies the more you utilize the capacity of the WOLA connections