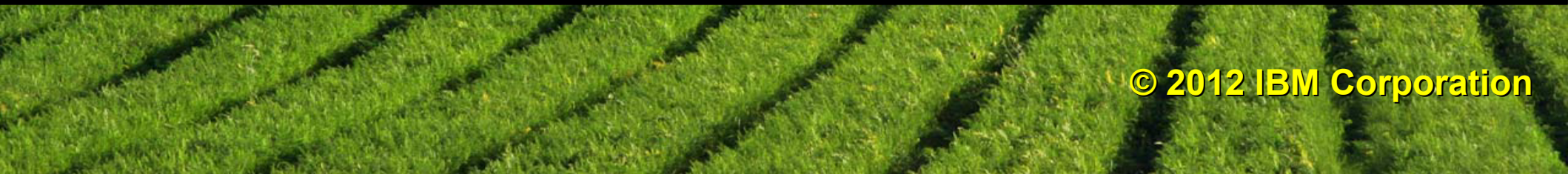
WP101490

WebSphere Optimized Local Adapters for WAS z/OS

Overview and Usage

ibm.com/support/techdocs



Contents

The flow of this document is organized around the following main topics:

Other WP101490 documents ...

Other Sources of Information

From the WP101490 Techdoc:

Technical Brochure

A "glossy brochure" format that gives a high-level of WOLA and answers a few anticipated questions.

Planning Guide

Providing specific information and InfoCenter pointers based on the type of WOLA usage you intend.

Native APIs Primer

A guided instructional exploration of the WOLA native APIs with COBOL

On Wikipedia:

`http://en.wikipedia.org/wiki/WebSphere_Optimized_Local_Adapters`

Or search WOLA and "disambiguate"

On YouTube:

`http://www.youtube.com/results?search_query=WASOLA1&aq=f`

The InfoCenter ...

The WAS V7 InfoCenter Has Wonderful Detail

The specifics of implementation and usage is offered in the InfoCenter:

`http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp`

z/OS

Optimized local adapters on WebSphere Application Server for z/OS

Optimized local adapters support on WebSphere® Application Server for z/OS® consists of a set of callable serv
together provi iness logic in a

The optimize d on WebSphe
from WebSph ddress space or

With this support, existing z/OS applications that are written in Cobol, PL/I, C, C++, and assembler can achieve
deployed under WebSphere Application Server on the same z/OS system.

Optimized local adapters also provide close integration of qualities of service (QoS), including support for fast thr
external address spaces andWebSphere Application Server for z/OS. Support is provided for using the adapters
(CICS®), UNIX® System Services (USS), and batch processing.

`cdat_ola` **Search string yields a good starting page**

Very good reference material on implementation and programming specifics

Table 1. BBOA1REG API syntax. The syntax is explained in the Parameters section.

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

`ola` **Including this in search tends to yield specifics on WOLA**

43 result(s) found for **ola**

com.ibm.websphere.ola (IBM WebSphere Application Server, Release 7)
com.ibm.websphere.ola Interfaces Execute ExecuteHome Classes ConnectionSpecImpl IndexedRecordImpl InteractionSpecImpl

com.ibm.websphere.ola (IBM WebSphere Application Server, Release 7)
Overview Package Class Tree Serialized Deprecated Index Help IBM WebSphere Application ServerTM Release 7 PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES All Classes Package com...

com.ibm.websphere.ola Class Hierarchy (IBM WebSphere Application Server, Release 7)
Overview Package Class Tree Serialized

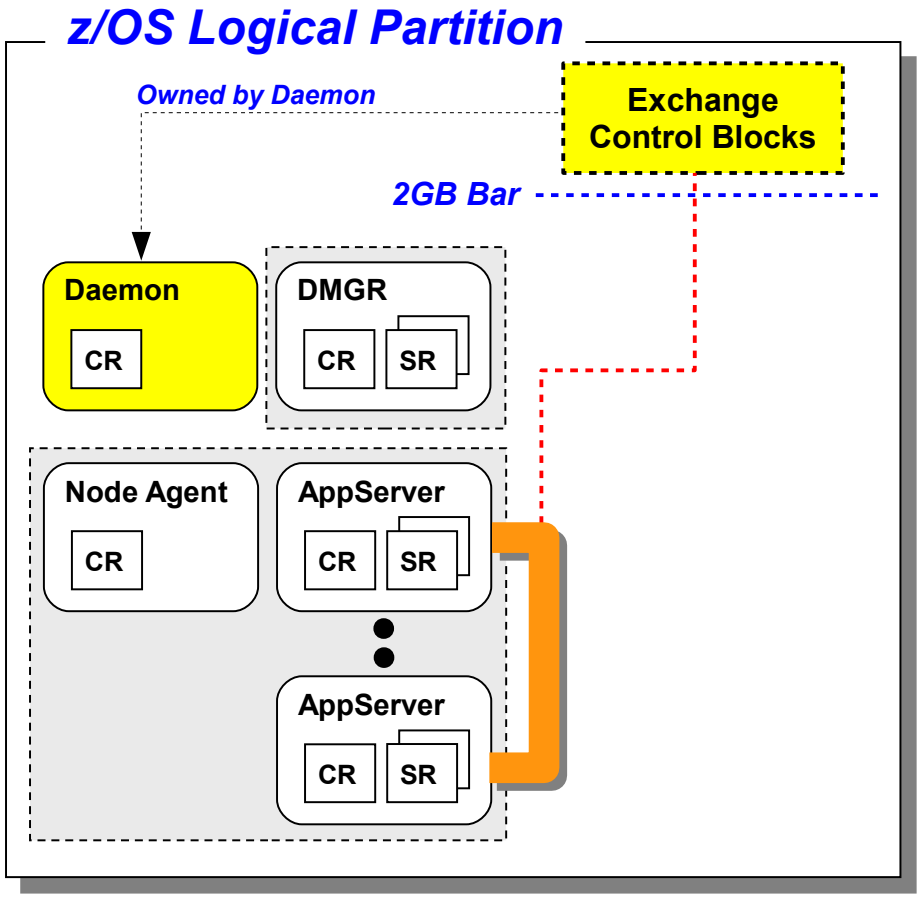
Setting stage ...

Setting the Stage

A little bit of background to the story of WOLA

In The Beginning Was "Local Comm"

This has been around since the early days of WAS on z/OS. It's a way to bypass the TCP/IP stack for internal IIOP calls between servers on the same LPAR:



If an IIOP call is made and WAS z/OS sees it's on the same LPAR, then this is automatically done

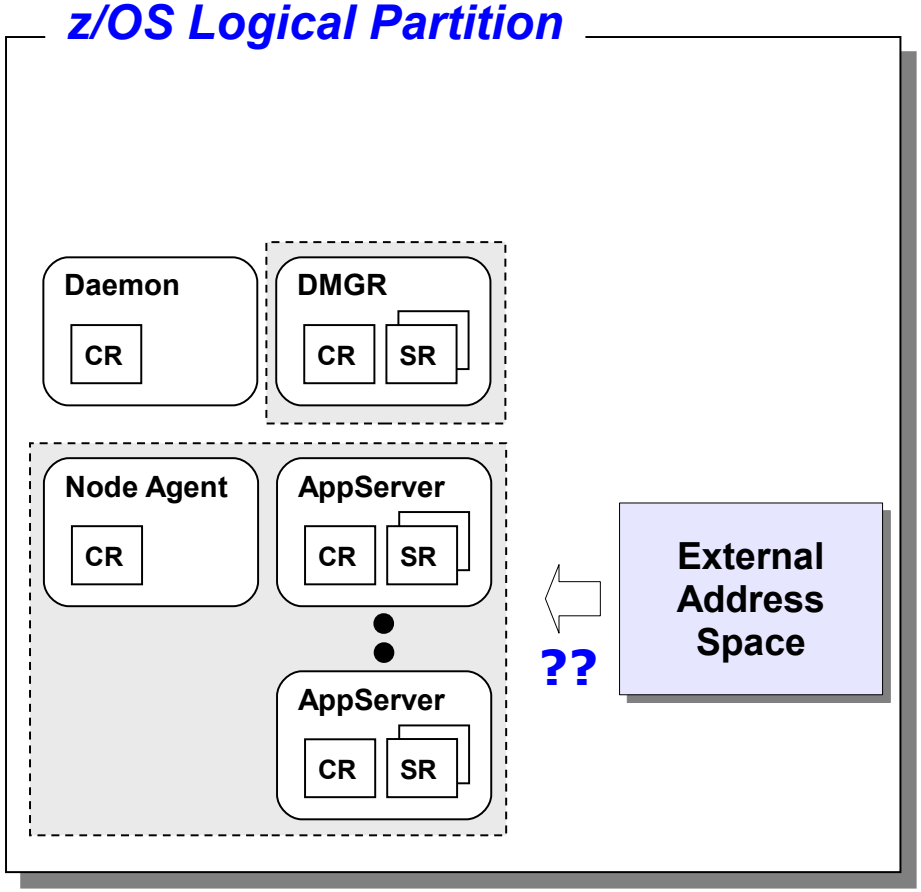
The Daemon server plays a key role in this; it owns the above-the-bar space used shared space and does the inter-address switching

It's very fast with very low overhead

The initial motivation ...

The Motivation Behind WOLA

It *started out* as a way to allow program access *into* WAS for high transaction rate batch programs. Other solutions existed, but they all had limitations:



Inbound to WAS?

As more and more solutions are built based on Java EE, there is a growing desire to access them by batch, CICS and IMS programs

MQ or Web Services?

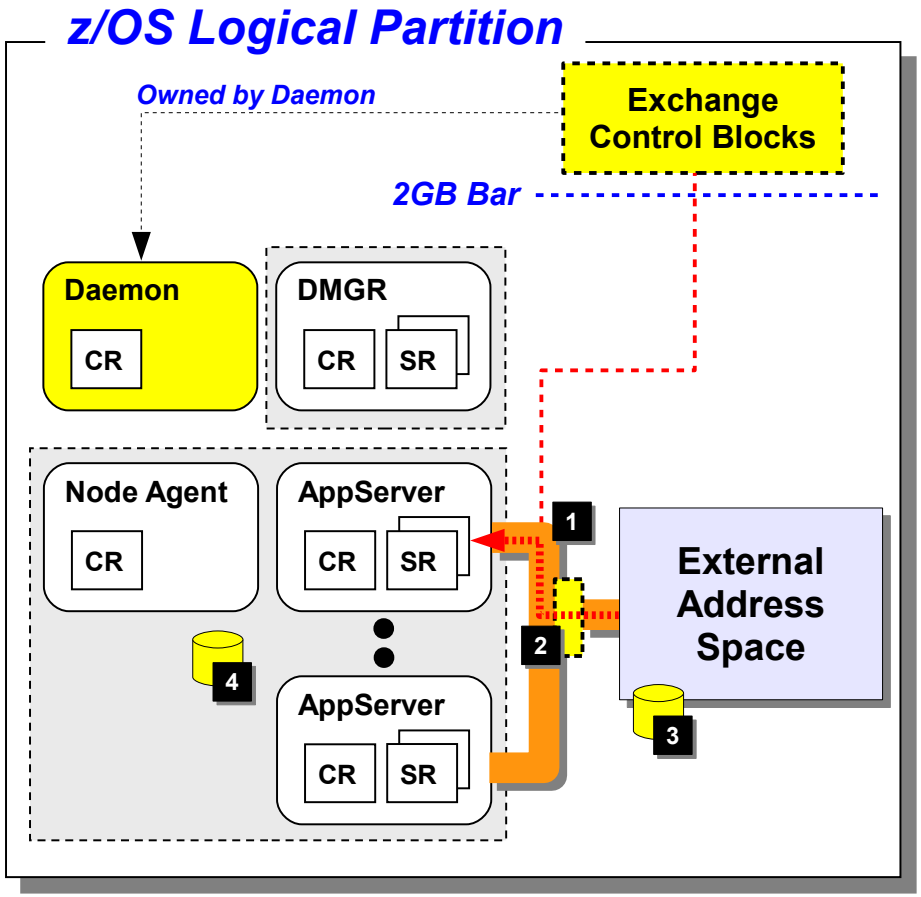
Both are very good technologies and have their role. But for very high throughput and low overhead, each has their drawbacks.

Something else was needed ... something *very fast* with as *little overhead* per exchange as possible

The birth of WOLA ...

Answer: Externalize the Local Comm Function

The Local Comm function was there. It just needed interface modules so external address spaces could access it:



WOLA was born

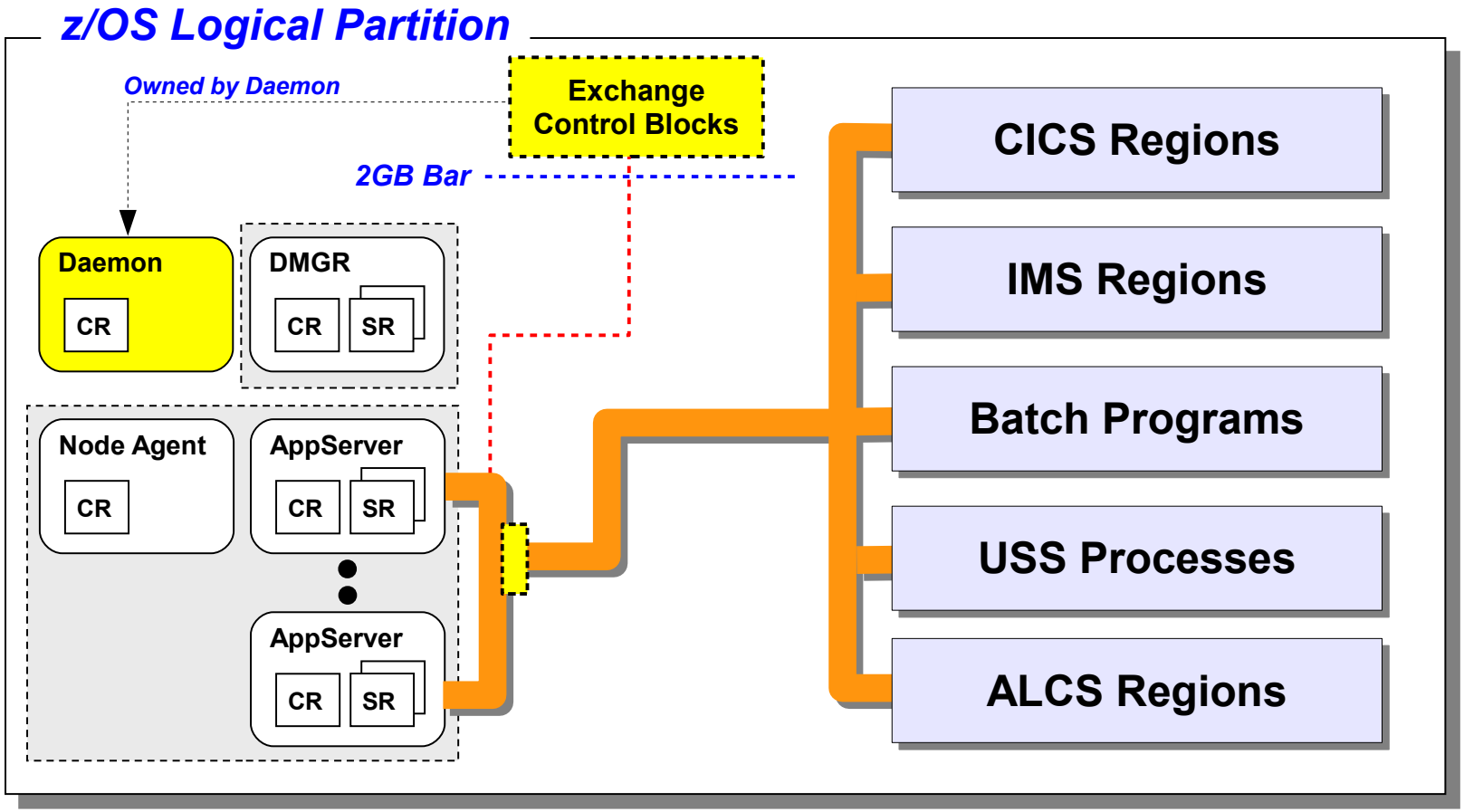
1. Existing Local Comm exploited
2. Externalization routines written
3. Programming APIs for external address spaces provided
4. Standard JCA adapter for the WAS server provided

Just Inbound? No! This is a bi-directional technology. Outside into WAS, and WAS out to external address space

What external address spaces? ...

What External Address Spaces Supported?

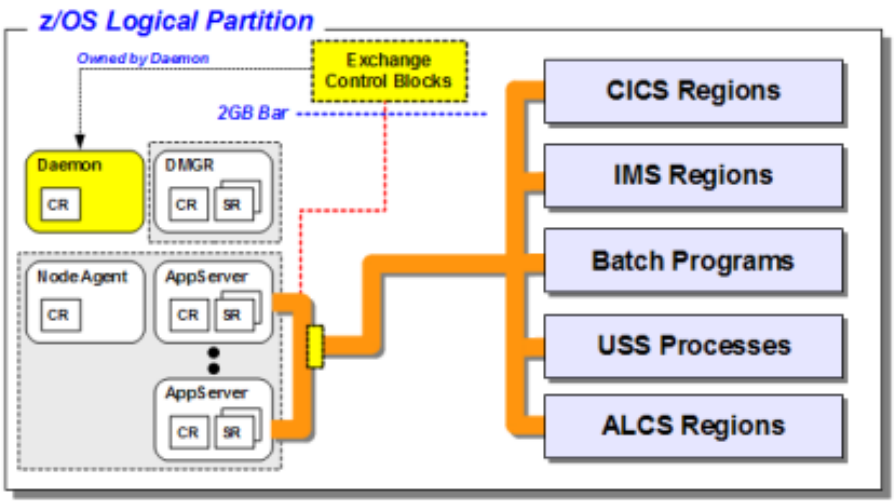
The picture looks like this ... recently enhanced with IMS in the 7.0.0.12 fixpack:



Let's slow down ...

We've Only Just Begun ...

The previous pictures set the basic framework in place, but there are many details left to explore



This is a somewhat abstract picture.

Key concepts are correct but many details have been omitted

It's important to keep in mind that the different external address spaces have different characteristics, and thus different WOLA considerations

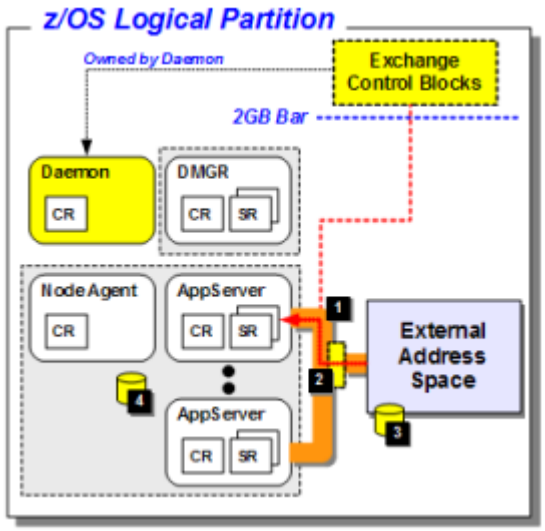
With that, let's start digging under the covers ...

The Basic Components

The pieces that make up WOLA and how they fit into the overall picture

Key Components

The earlier picture gave a hint to some of the components. We'll start to review them a bit more closely here



WOLA was born

1. Existing Local Comm exploited
2. Externalization routines written
3. Programming APIs for external address spaces provided
4. Standard JCA adapter for the WAS server provided

Proper level of WAS z/OS

- Function made available first in 7.0.0.4
- Recently enhanced to include IMS and other functionality in 7.0.0.12

WOLA support enabled in the WAS z/OS cell

- Shell script to create symlinks to the WOLA files
- Single WAS environment variable to turn function on

WOLA JCA adapter installed in the WAS z/OS nodes

- Simple JCA adapter installation like any other

External WOLA modules copied out to library

- A supplied shell script will do this for you
- This is how you make function available to others

Enablement work done in external address space

- For batch program it's as simple as STEPLIB
- For CICS and IMS there's a few easy Sysprog tasks to do

Sample code used to validate the environment

- Sample EAR file provides the WAS-side code
- Sample COBOL and C/C++ code provides the external

There's very little here that's complex for the Sysprog. Fairly straightforward stuff.

The WAS z/OS InfoCenter has very good documentation on this

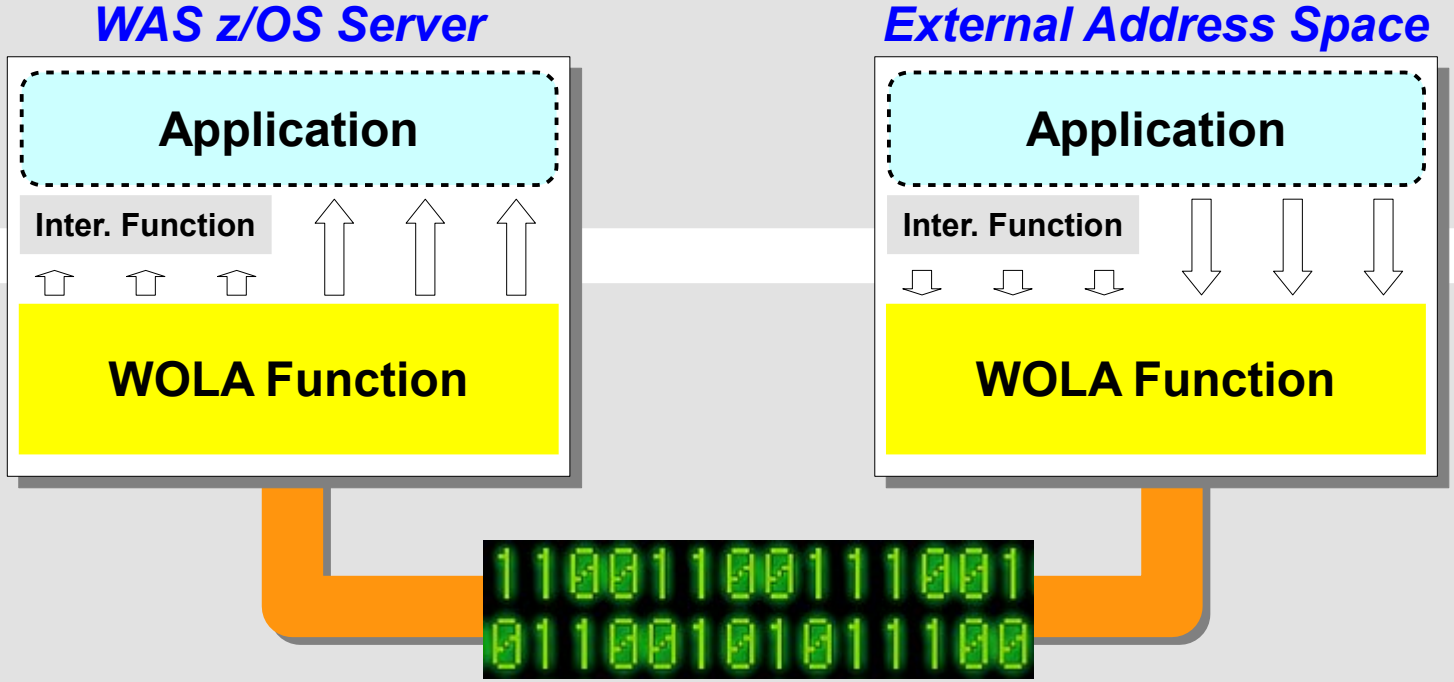
Key concepts ...

A Few Fundamental Concepts

These are absolutely essential to understanding the details that follow

Concept #1 - WOLA is a Byte Array Pipe

As such it pays no attention to format or code page. That's why it's so fast.



The two sides of the exchange must have some awareness of each other so that the data can be in the proper format, layout and codepage

There are Eclipse based wizards to help with COBOL COPYBOOKS for CICS programs

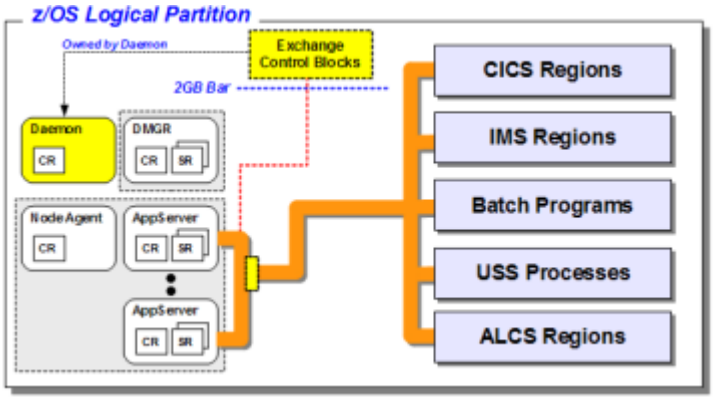
(See the YouTube videos referenced earlier)

Address space to address space ...

Concept #2 - WOLA is Address Space to Address Space

This is a very low-level mechanism between address spaces:

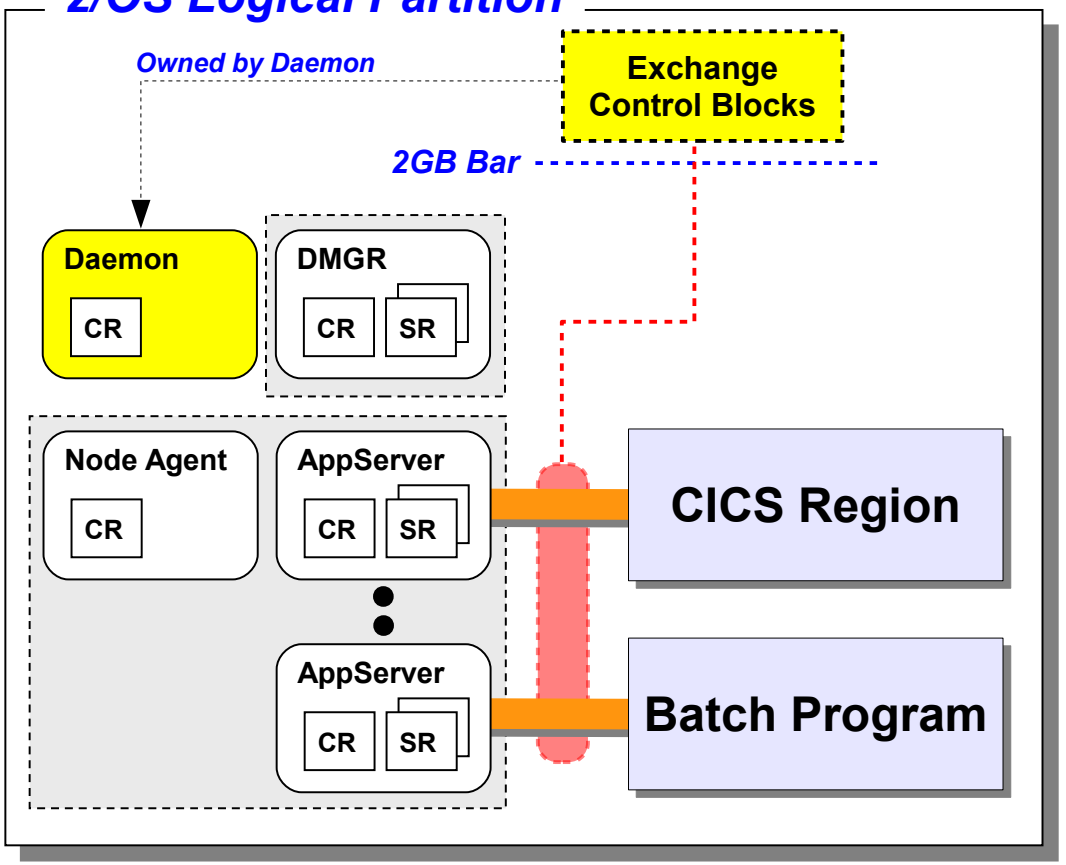
This picture from earlier was a bit misleading



To exchange with an application in a server, it is required to register to *that specific server*

WOLA does not provide general access to the WAS cell, it provides very low-level access to the specific server and applications in that server

This is more technically correct*



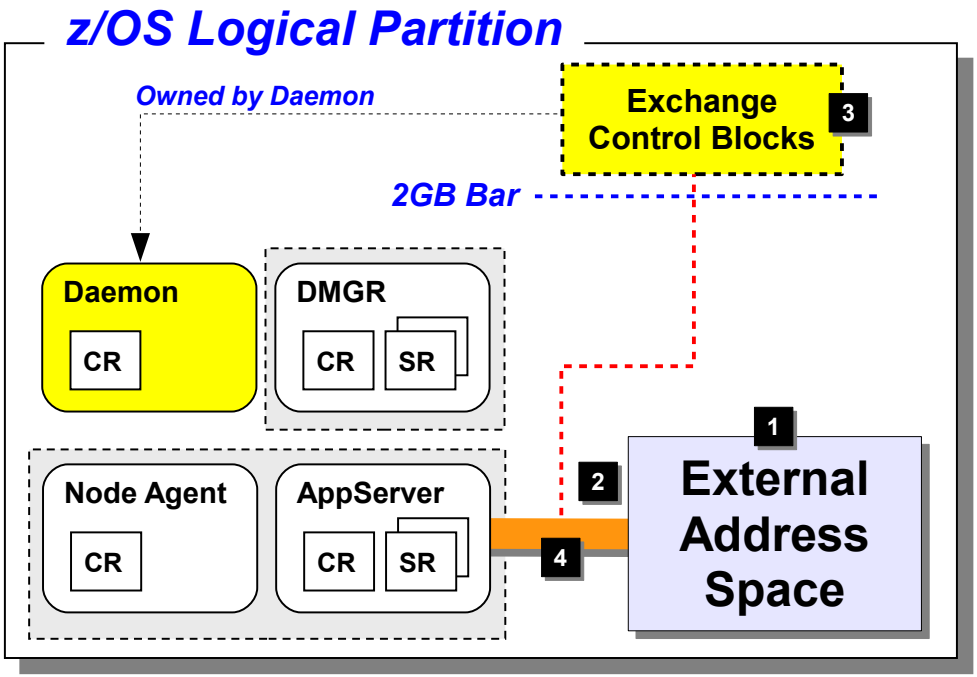
Multiple registrations permitted, to the same server or different servers

* Strictly speaking the connection is to the WAS server controller region

"Registration" ...

Concept #3 - Registration

Before any external address space may use WOLA, it must first *register in:*



Registration ...

1. Is *always* initiated by the external address space into the specific WAS server
   There are different ways to accomplish this as we'll see
2. The cell, node and server *short* name is given, along with a *registration name*
   The registration name is what identifies the WOLA transfer pipe. Since multiple registrations is permitted, this is how applications designate which pipe to use.
3. The registration control block structure is built in the Daemon shared space above the 2GB bar
   This is not in the Daemon address space. It is owned by the Daemon.
4. Connection built and ready to use
   Which brings up the inbound/outbound discussion

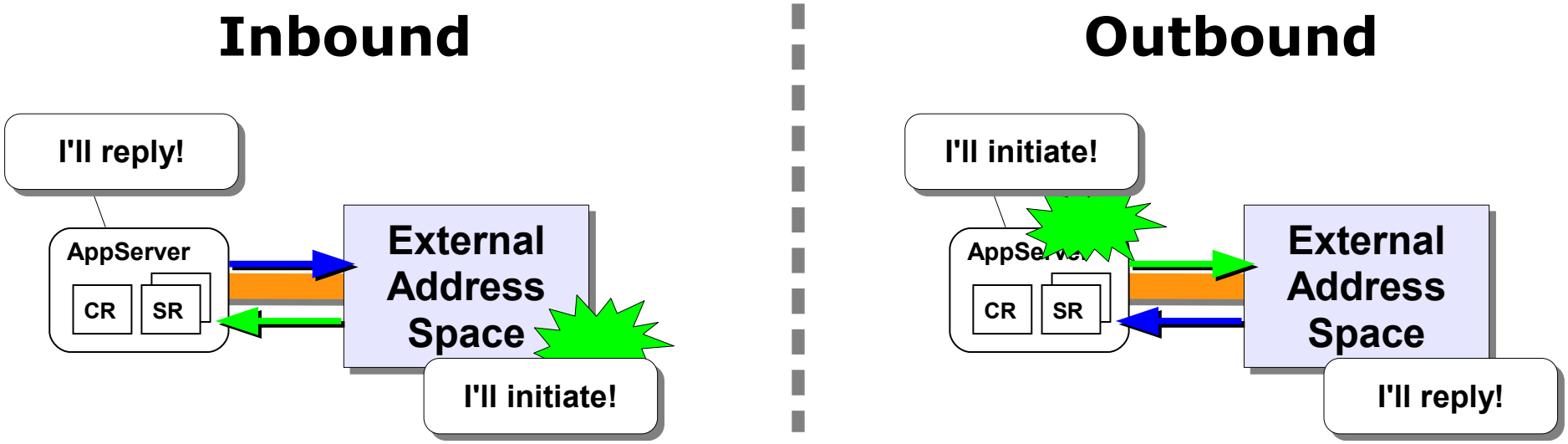
The key here is that the external address space has to request of WAS z/OS that it allow the access and build the connectivity infrastructure.

Inbound vs. Outbound ...

Concept #4 - Inbound vs. Outbound

Assuming registration is in place, **then this is all about which side *initiates* the conversation between programs:**

We think of this from the perspective of WAS z/OS:



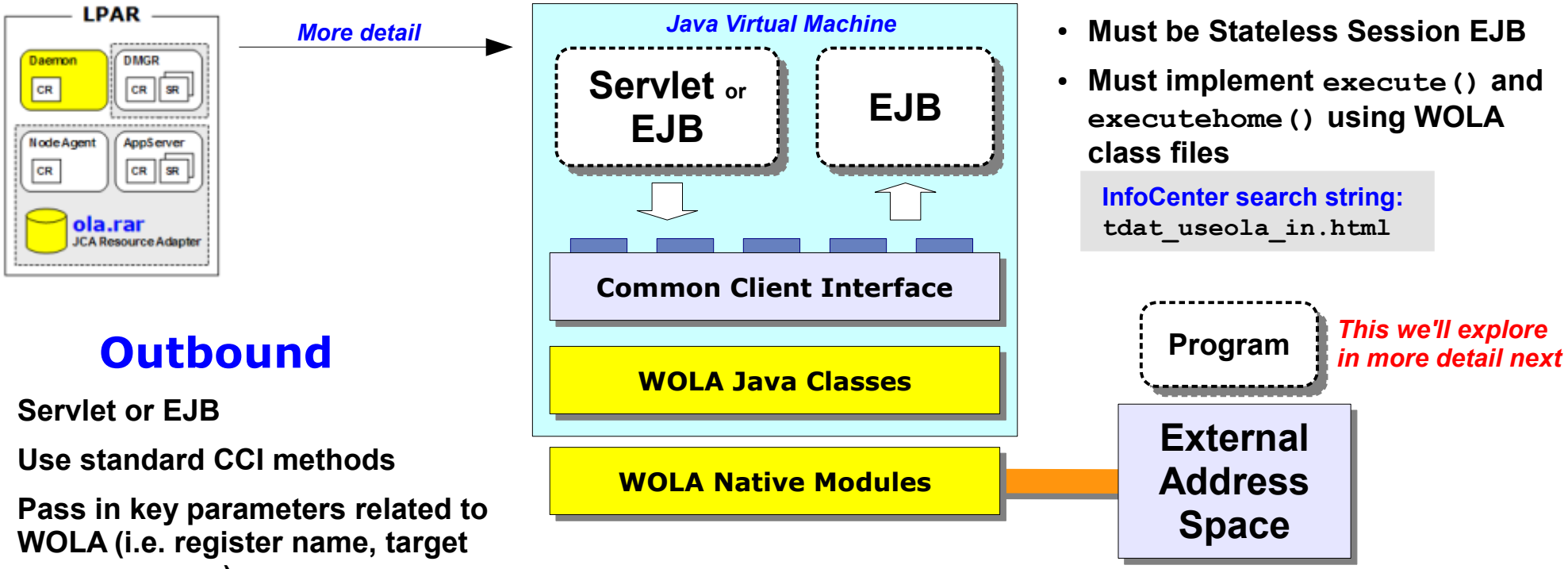
This is important because many application considerations are a function of this

What APIs, what kind of Java program, security propagation ...

Java side ...

Concept #5 - The Java Side of the Application Question

Much of WOLA is hidden by a standard JCA resource adapter. But a few things do come into play:



Inbound

- Must be Stateless Session EJB
- Must implement `execute()` and `executehome()` using WOLA class files

InfoCenter search string:
`tdat_useola_in.html`

Outbound

- Servlet or EJB
- Use standard CCI methods
- Pass in key parameters related to WOLA (i.e. register name, target program name)
- WOLA specifics are generally well hidden from the application

InfoCenter search string:
`tdat_useoutboundconnection.html`

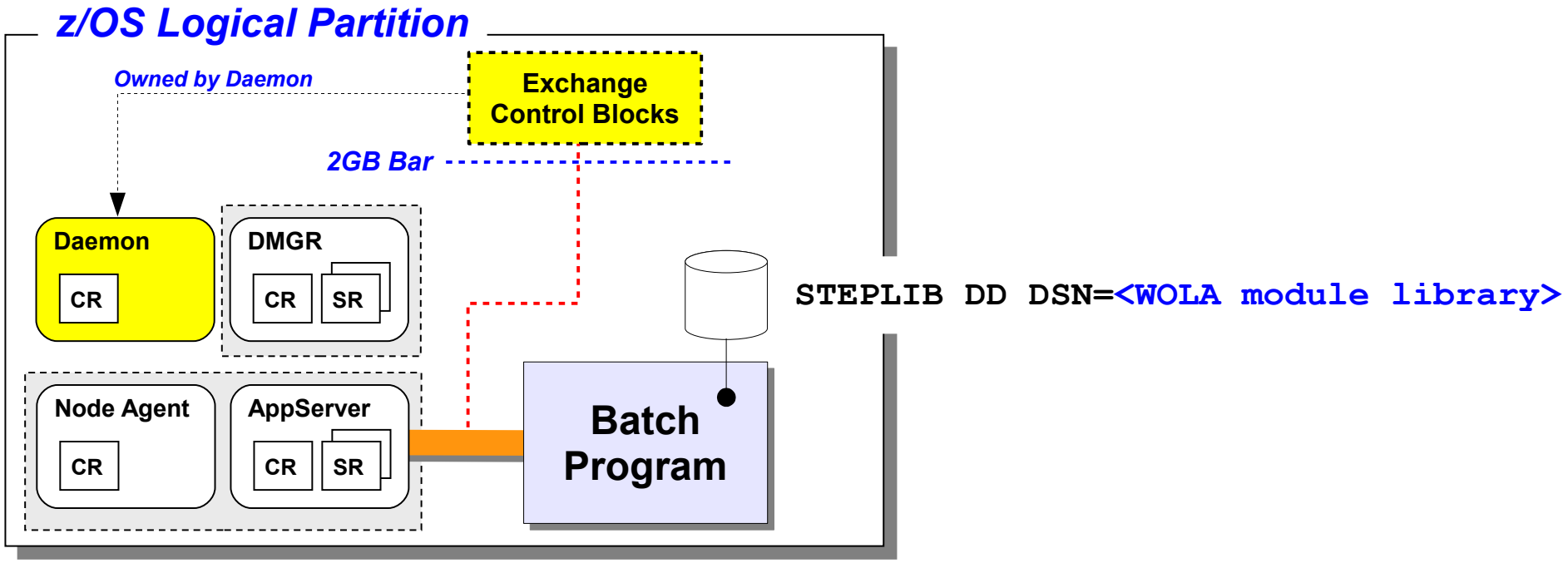
Not complete transparency, but a considerable degree of shielding Java from this very low level interchange

WOLA and Batch

How it's implemented and how it works

Batch Program Access to WOLA Modules

This is easy ... copy the modules out to a module library and make it available to the batch program -- STEPLIB or LINKLIST



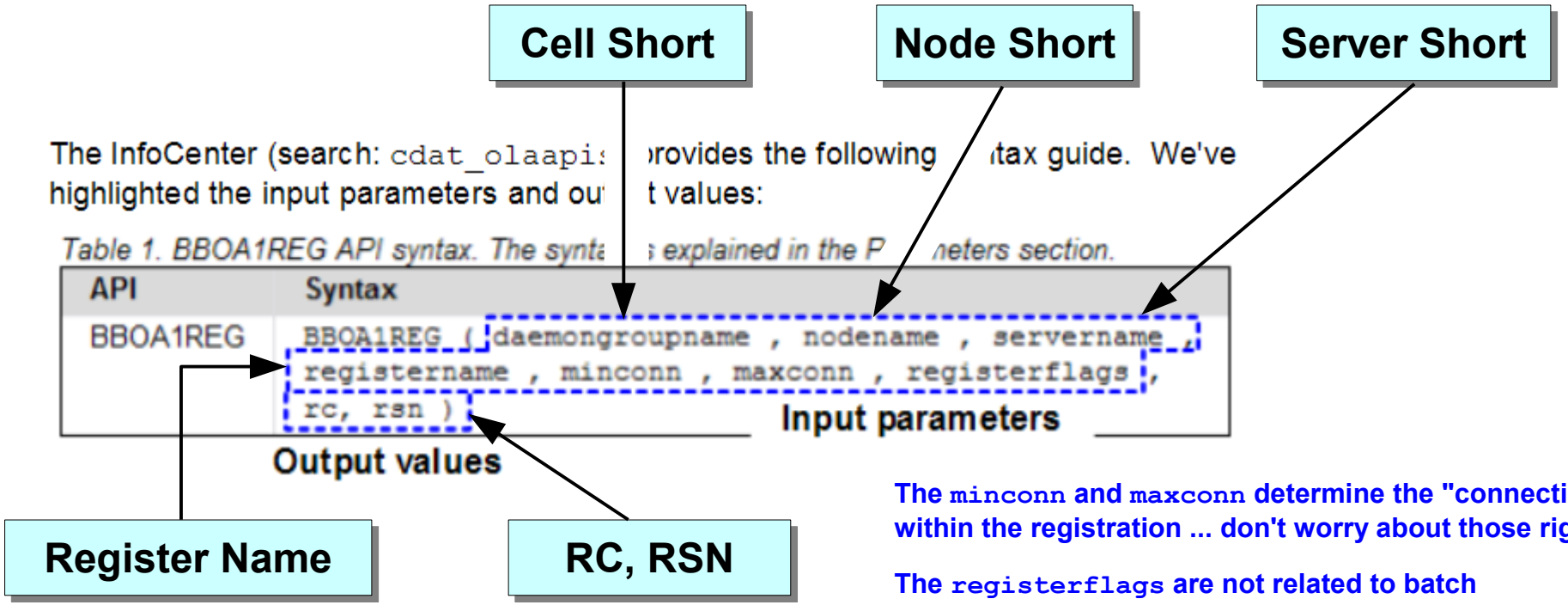
This merely gives your batch program *access* to the APIs

The next step is to actually *use* them ...

First step, registration ...

Registering Into the WAS Server

The first step in any processing ... the external AS *always* registers *in*, never the other way around. From the "Primer" document at WP101490:



The `minconn` and `maxconn` determine the "connections" within the registration ... don't worry about those right now

The `registerflags` are not related to batch

Registration is one of the easier APIs to work with. It either works or it does not work, and if not the RC, RSN in the InfoCenter is really clear about the problem

The author of this presentation *loves* the InfoCenter's RC, RSN code write-ups ... perfectly clear and easy to understand

BBOA1INV ...

Inbound: Invoking the Target EJB in the WAS Server

The `BBOA1INV` API is a "basic" API ... very simple to use ... it invokes the named EJB in the target WAS server:

Registration Name
(set on `BBOA1REG` API)

Table 23. BBOA1INV API syntax. The s ax is explained in the Parameters section.

API	Syntax
BBOA1INV	`BBOA1INV ( registername, requesttype, requestservicename, requestservicenamel, requestdata, requestdatalen, responsedata, responsedatalen, waittime, rc, rsn, rv )`

You need to specify the input and output pointers and length.

The Primer walks you through that. It's not difficult

Request Service Name
This is the JNDI name of the home interface of the target EJB in WAS

Return, Reason
Again, InfoCenter has excellent write-up on these values

Value
This is the response back from the program in WAS z/OS

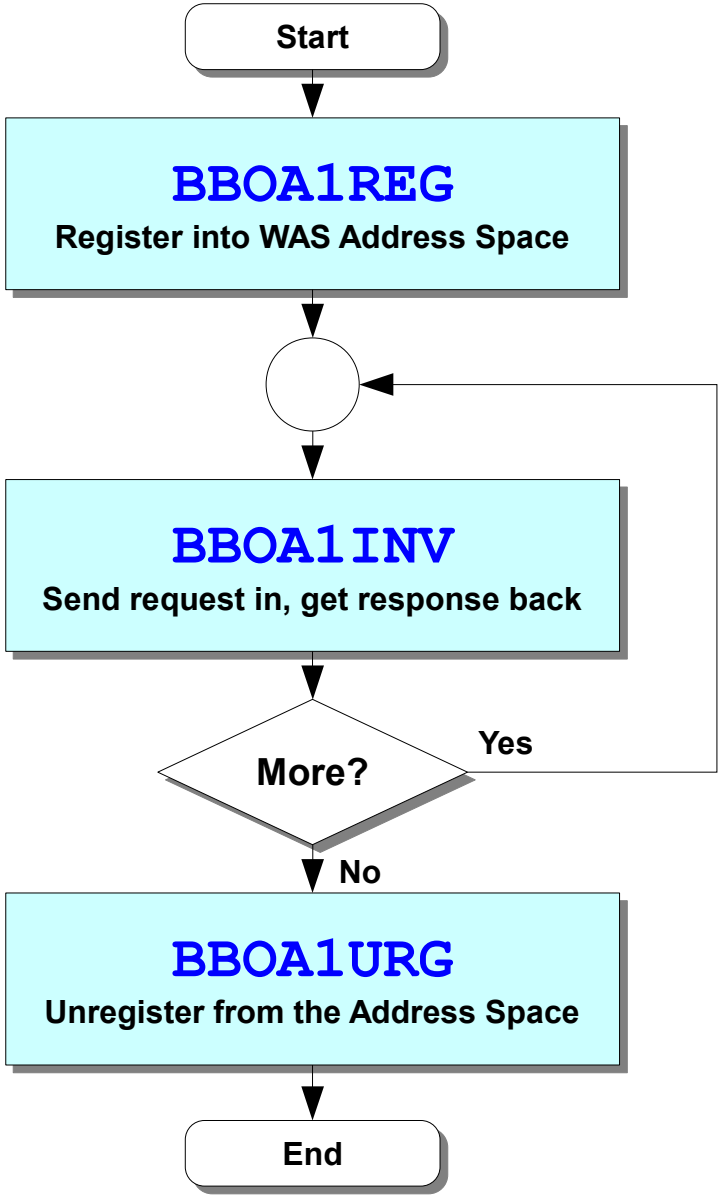
Keep an eye on this notion of the response being the return on the call. Later we'll show how you can operate asynchronously.

This API can be looped over and over again within the same registration ...

Looping ...

Inbound: BBOA1REG, Loop BBOA1INV, BBOA1URG

That's the most simple programming structure:



A registration may be used over and over again before tearing it down

Loop as many times as you need for the program requirements

The BBOA1INV is making some assumptions to keep things simple

- Synchronous ... wait until response received
- Get connection, use, tear down connection
- (This is where "advanced" APIs come in)

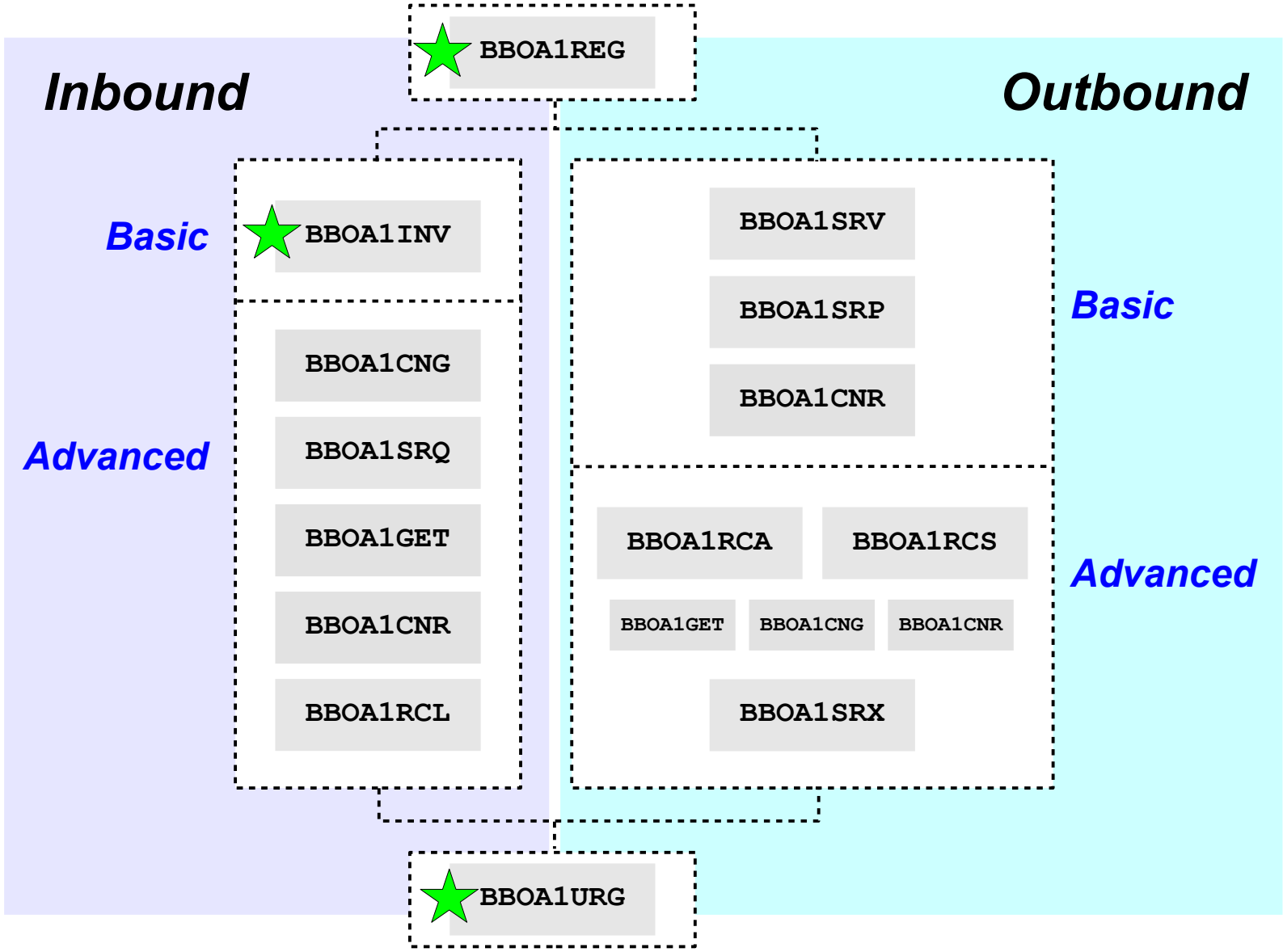
When done, tear down registration

Simple, easy, effective for inbound exchanges

API map ...

A Map of the APIs

This sets the stage for the discussion of the APIs



Simple or more complex, depending on needs

Green stars show what we illustrated on previous chart

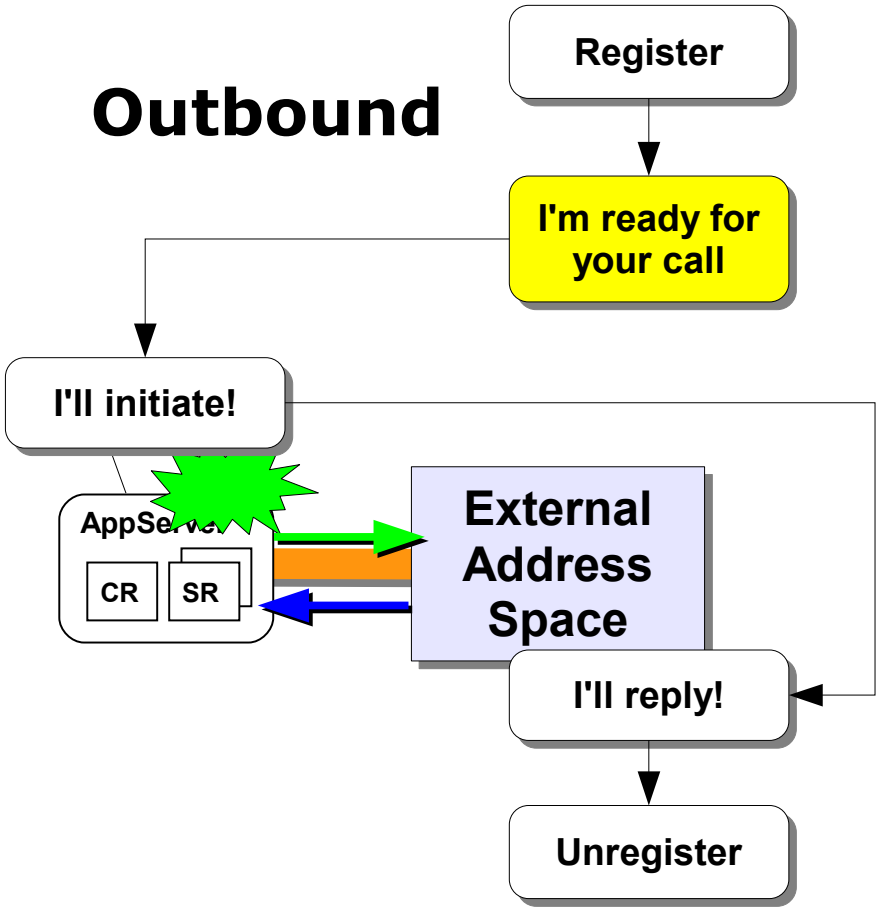
Possible to shield CICS and IMS programs from all of this

Primer document walks you through simple exercises of all these APIs

Outbound from WAS ...

Outbound Gets a Bit More Complex with Batch

The reason why is the batch program has to set in a "listen state" for the call from WAS:



- **Host a Service**
  - This is what puts the batch program in a "listen state"
  - `BBOA1SRV`
- **Send a Response**
  - When request comes from Java side this API is used to send a response back
  - `BBOA1SRP`
- **Release Connection**
  - Releases the connection handle secured by `BBOA1SRV`
  - `BBOA1CNR`

Outbound to batch not that common a thing for initial usage of WOLA. We showed you this to help understand the division of APIs around "inbound" and "outbound"

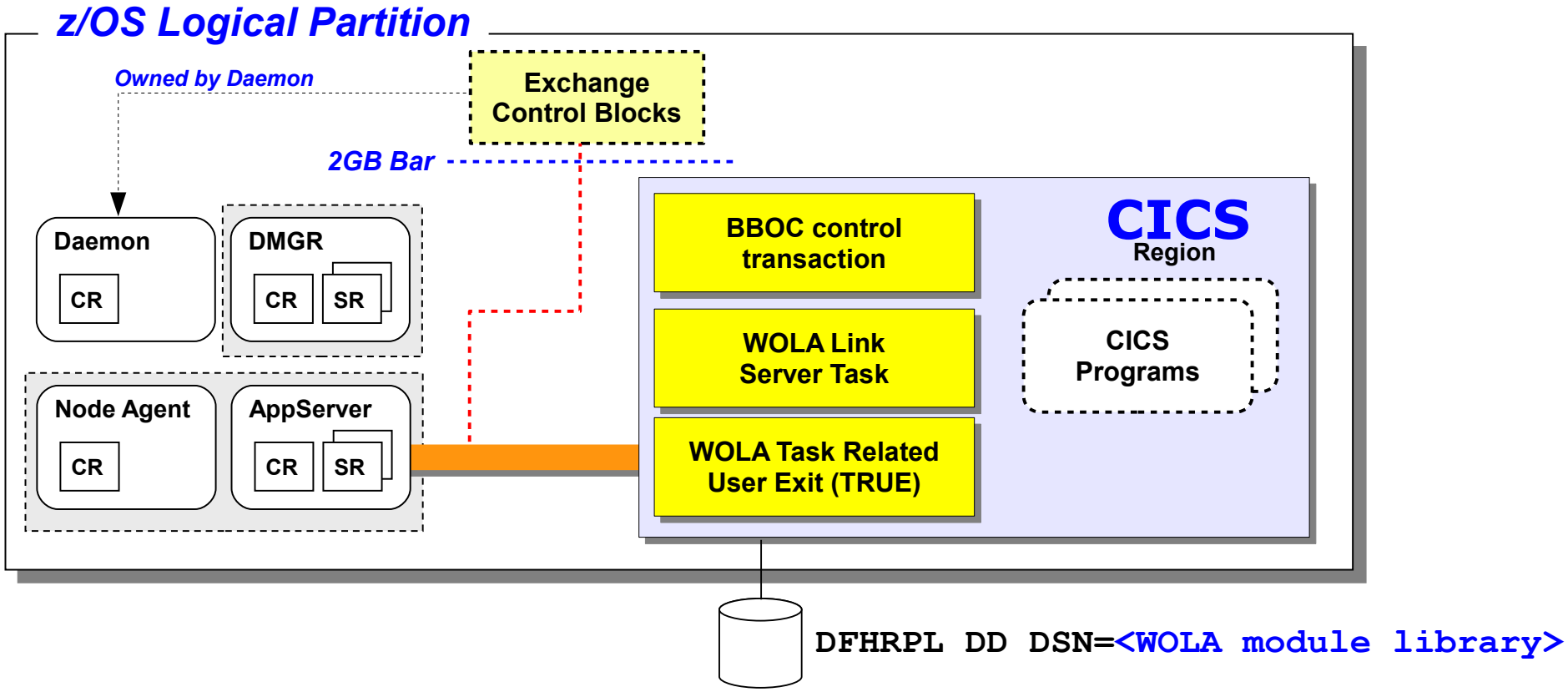
CICS support ...

WOLA and CICS

How it's implemented and how it works

High Level Overview of CICS Support

The following picture illustrates the pieces:



TRUE -- the low-level heart of the WOLA support in CICS

Link Server Task -- a way of shielding CICS programs from the APIs and knowledge of WOLA

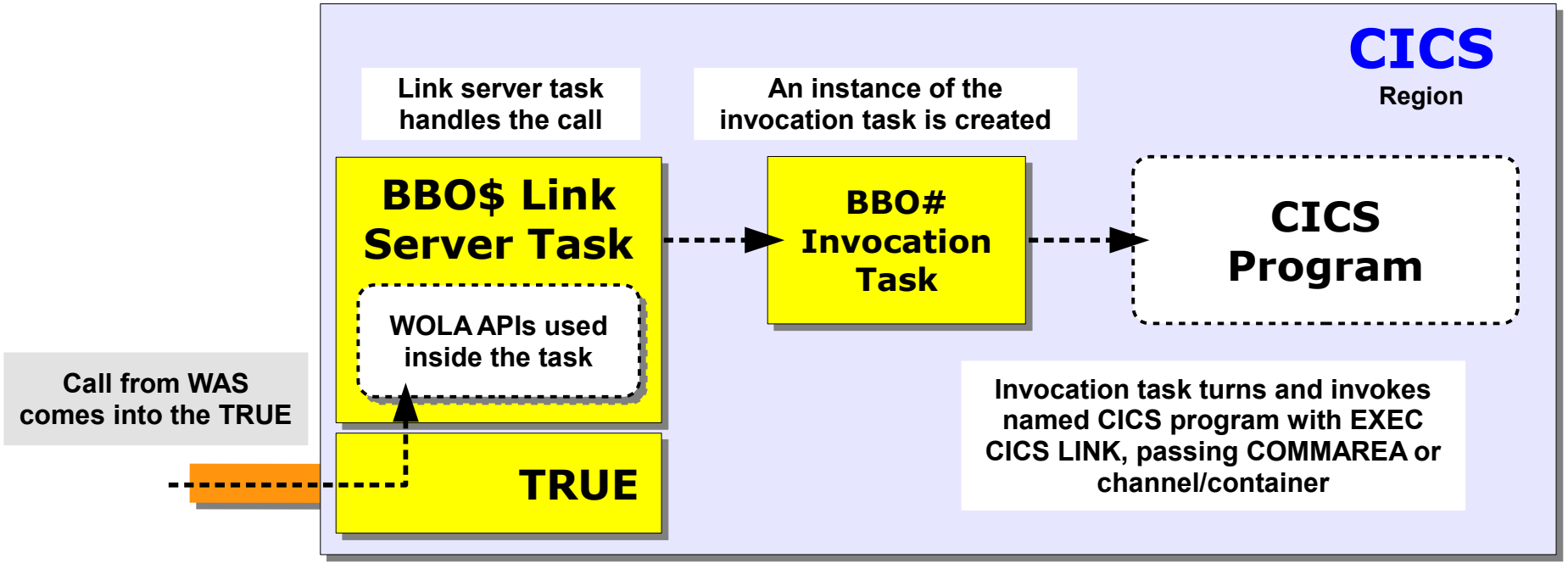
BBOC -- a useful command line transaction to start and stop various pieces of WOLA

Installation of each is standard CICS system programmer stuff ... very easy

Link Server Task ...

Link Server Task

The piece that allows you to shield CICS programs from knowldge of WOLA. It applies to outbound calls ...



The target CICS program has no awareness of WOLA. It sees that another program invoked it using standard CICS routines. The BBO$ / BBO# activity is automatic.

The program's response is returned to WAS automatically as well.

There are details and optional changes to behavior we're not showing you here. The key concept is that there is a way to shield CICS programs and the Link Server Task provides it for *outbound* calls.

The BBOC transaction ...

The 3270 BBOC Transaction

This provides the tools to do many things with the WOLA support in CICS:

InfoCenter search string: `rdat_cics`

TRUE Related

```
BBOC START_TRUE <parms>
BBOC STOP_TRUE <parms>
```

You may automate this by including the TRUE in the Program List Table Post-Initialization (PLTPI)

Link Server Related

```
BBOC START_SRVR <parms>
BBOC STOP_SRVR <parms>
```

Parameters include ability to process registration on link server task start, set security and transaction flags and other options

If you don't use the link server task you do not need to start the link server

Registration Related

```
BBOC REGISTER <parms>
BBOC UNREGISTER <parms>
```

Provides a way to process a registration into WAS z/OS outside the program itself.

Or use `BBOA1REG` in the program

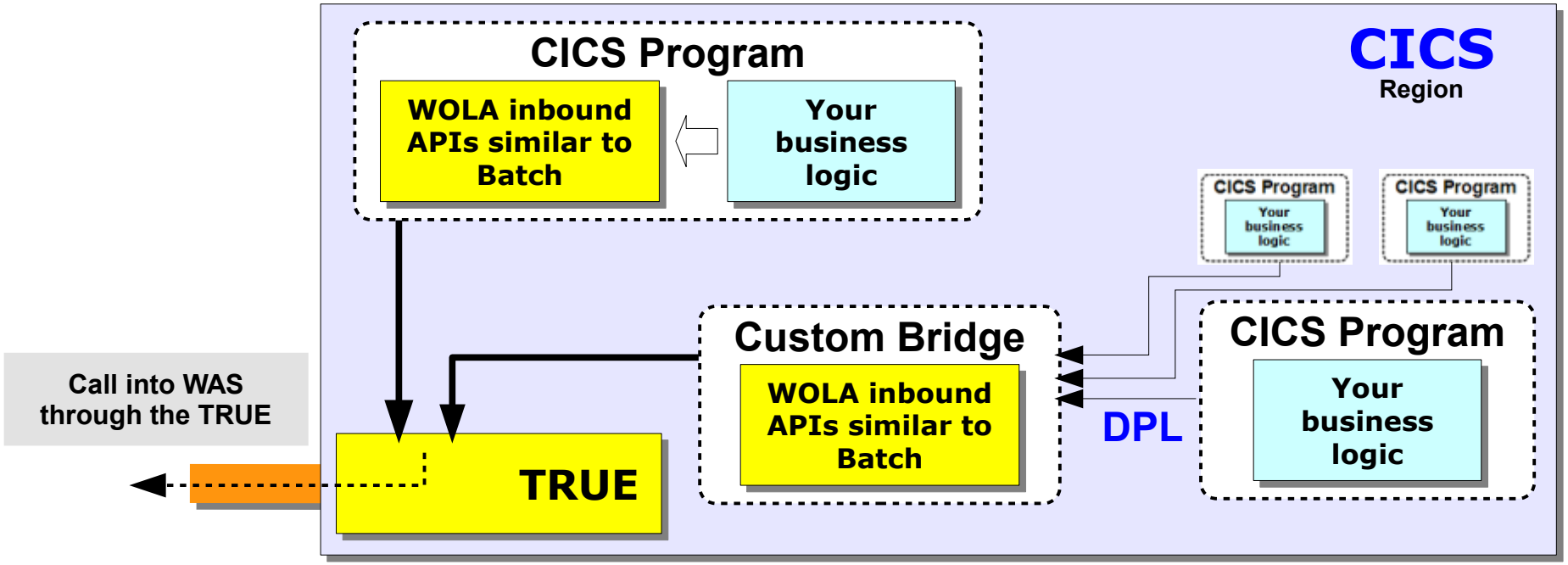
Or use the link server task which can also process registration

Point here is not the details of this, but rather what BBOC offers

CICS inbound ...

When the CICS Program Initiates an Inbound Call

The Link Server Task does not come into play. Therefore, at least some coding to the WOLA APIs is necessary. But here *you* can provide some shielding ...



Embedded API Usage

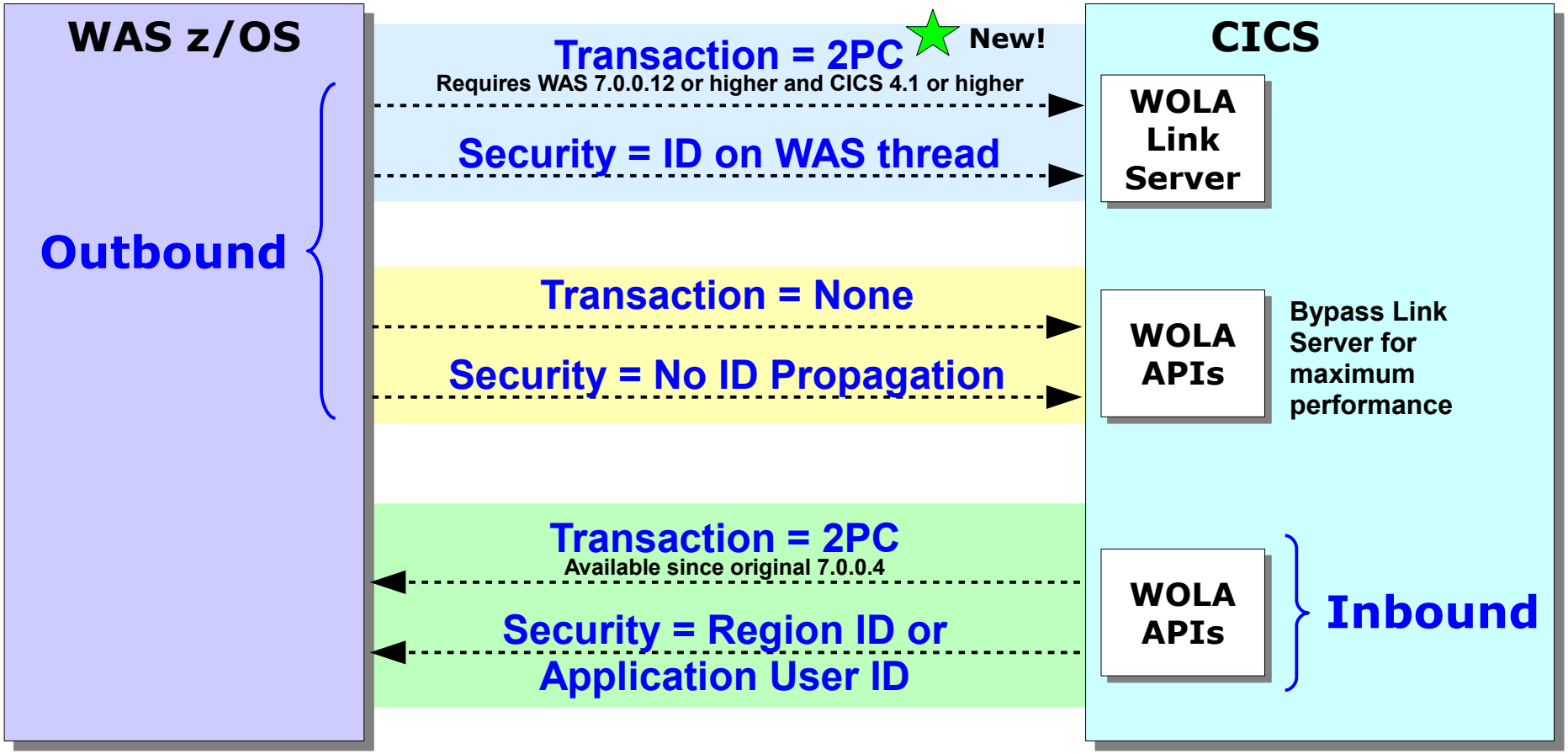
- API's in the compiled program
- Normal WOLA Inbound APIs
- Can offload register/unregister to manual BBOC if you wish

Custom Bridge Program

- Modular design ... isolate WOLA from business logic
- Multiple programs could use DPL to drive WOLA
- Bridge code then uses APIs

WOLA and CICS, Transaction and Security

A summary picture:



See the WP101490 "Design and Planning Guide or the InfoCenter for the specific details of this

IMS ...

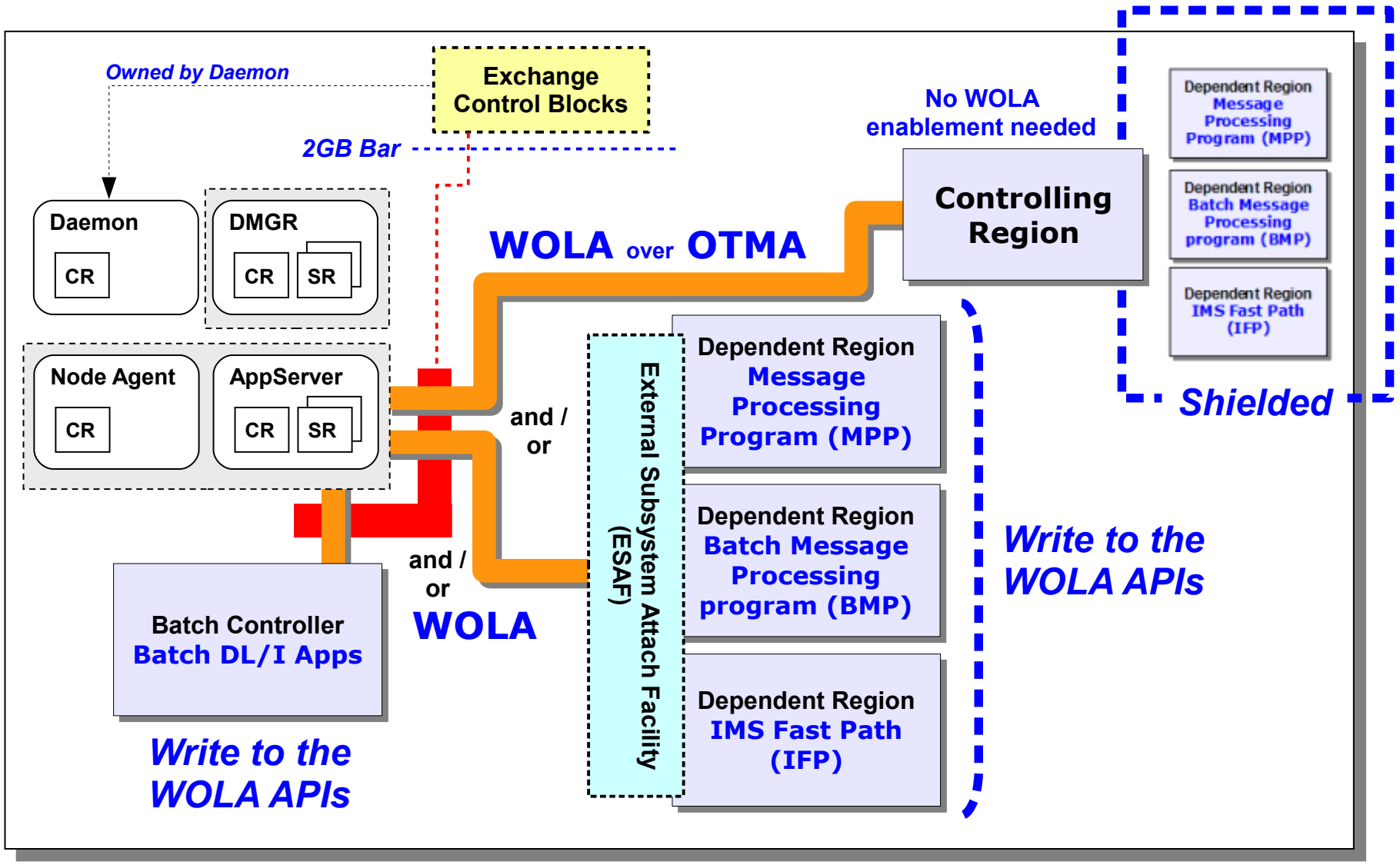
WOLA and IMS

New!
7.0.0.12

How it's implemented and how it works

High Level Overview of IMS Support

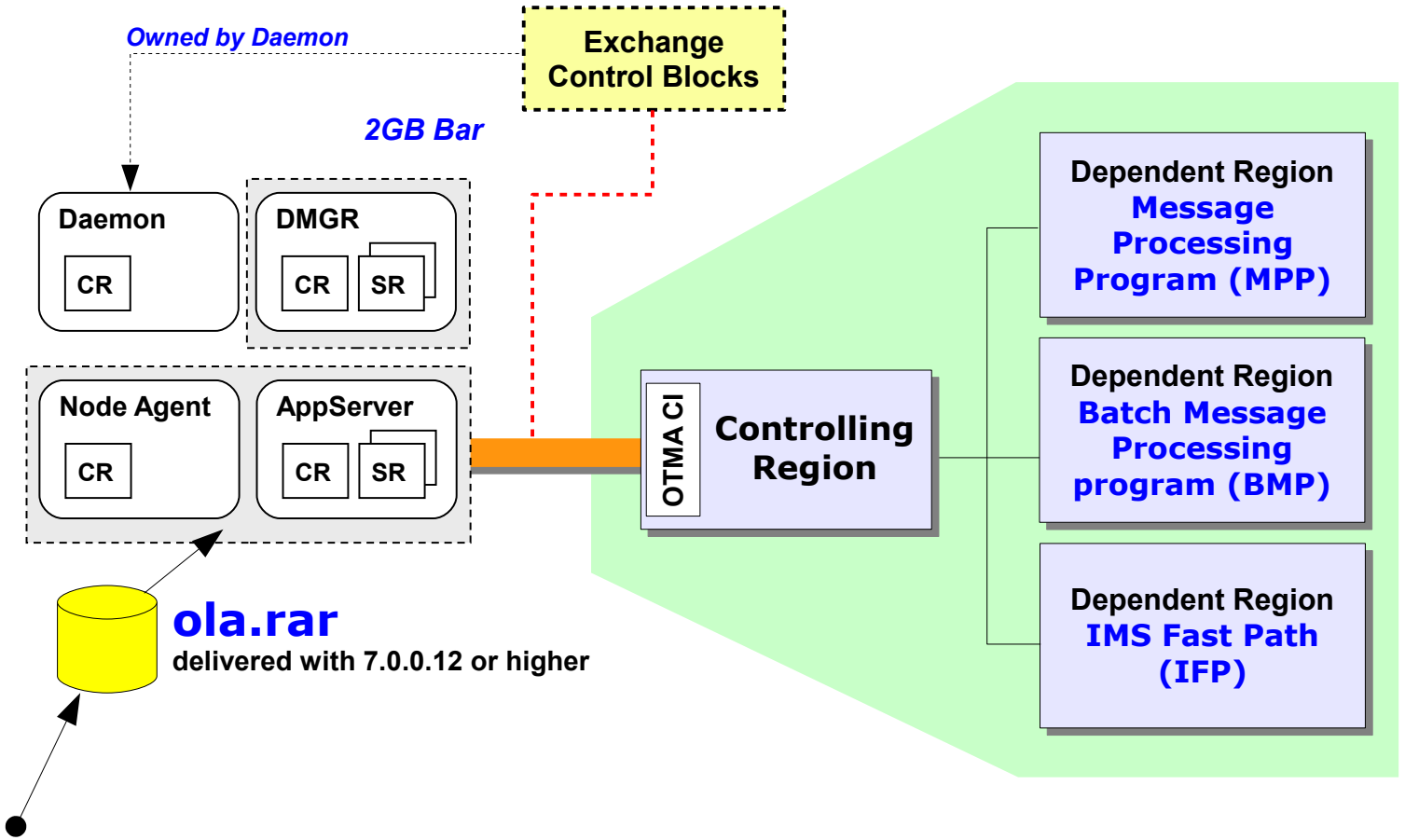
New with 7.0.0.12, the following picture illustrates the pieces:



OTMA ...

The OTMA Support

To use the OTMA support it's important to be using the latest ola.rar file:



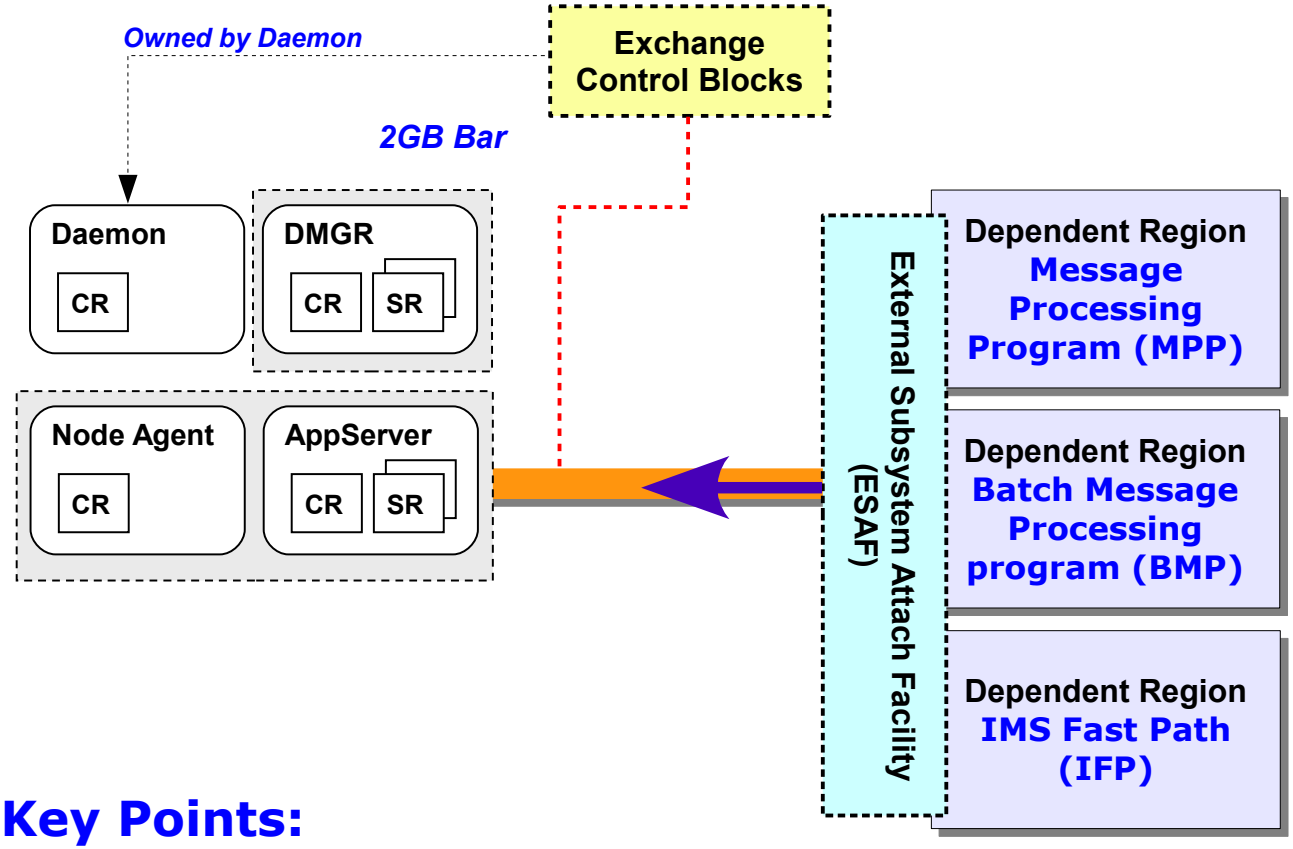
- **No changes needed to IMS**
- **No changes to applications in IMS**
- **Provides a way to propagate minimum sync level in**
- **Provides a way to propagate WAS thread ID into IMS**
- ***Not a replacement for IMS Connect***
  It's complementary

`setOTMAServerName()`	XCF Server name for IMS control region
`setOTMAGroupID()`	XCF Group name for IMS control region
`setOTMASyncLevel()`	To set SyncOnReturn or SyncLevel1 (CM0 or CM1)

Inbound ESAF ...

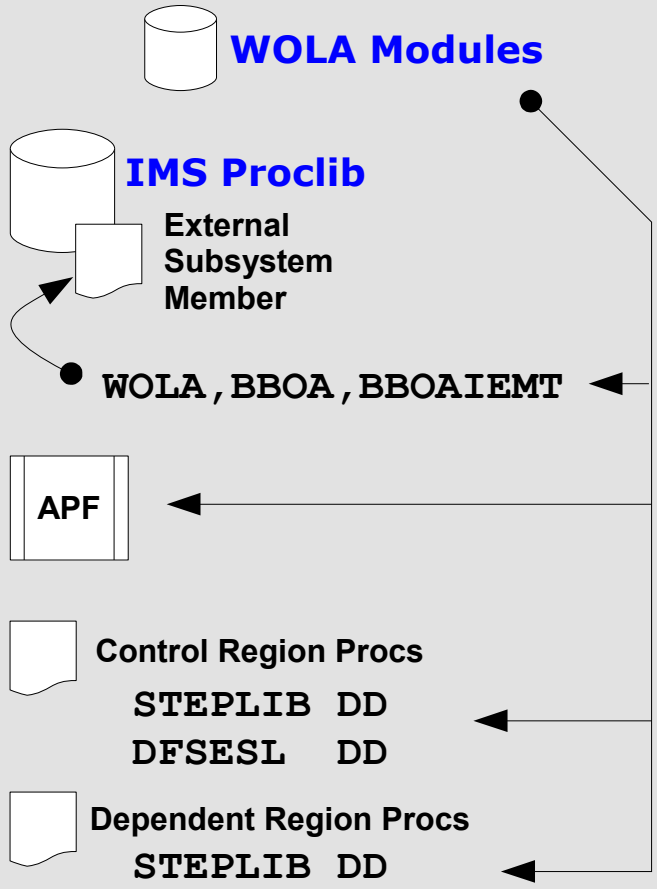
Inbound Using WOLA APIs and ESAF

They're the same API names and syntax, but they've been updated to recognize they're operating in IMS and use ESAF:



Sysprog Work

InfoCenter: `tdat_enableconnectorims`



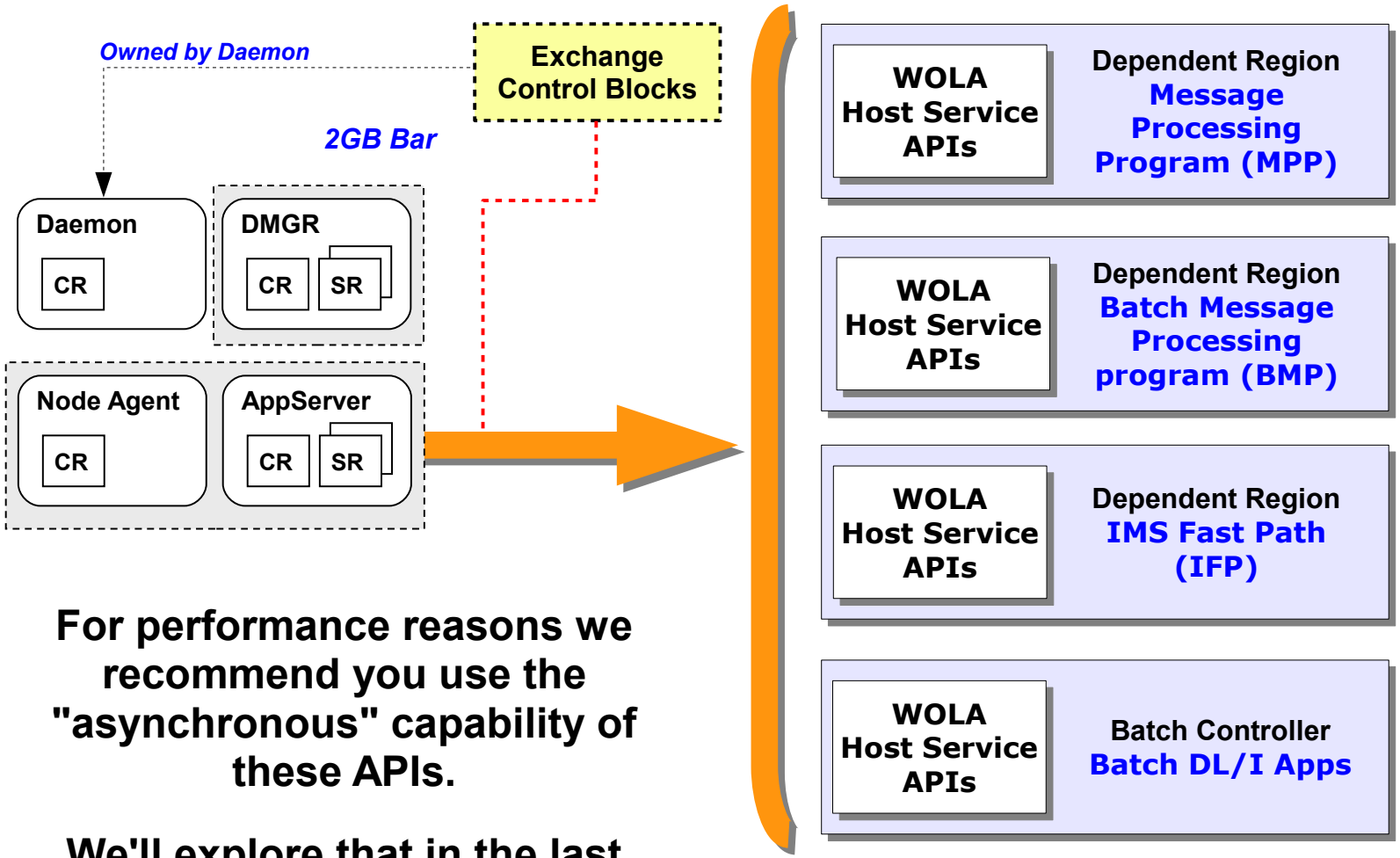
Key Points:

- **No new APIs "just for IMS" … same APIs as before**
- **Same usage and syntax**
- **If IMS, make sure 7.0.0.12 or higher used so modules will know they're in IMS and use ESAF**

Outbound WOLA …

Outbound Using WOLA APIs (No OTMA ... "WOLA both Sides")

This provides the maximum efficiency, but there are some considerations:



For performance reasons we recommend you use the "asynchronous" capability of these APIs.

We'll explore that in the last section of this presentation

We recommend asynch option on APIs to prevent the execution thread from being blocked.

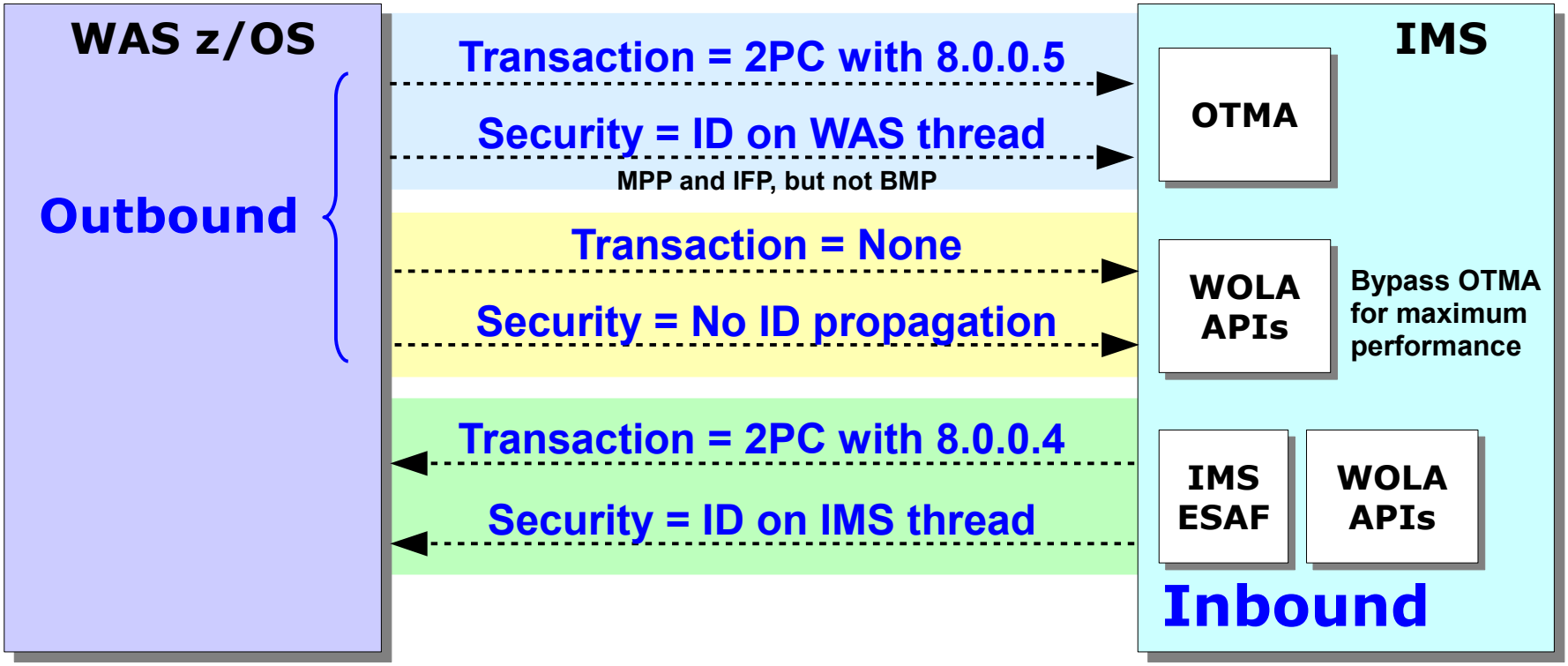
For IFP/BMP this may be intentional and acceptable since these are single purpose dependent regions.

For MPPs, running in message processing regions, using the blocking versions of these APIs will tie up a critical thread in the MPR. That may prevent other transactions from being serviced.

Transaction and security ...

WOLA and IMS, Transaction and Security

A summary picture:



The transactional propagation issue is on the list of enhancements for the future.

See InfoCenter for details about security assertion requirements

Setup very similar to those required for traditional batch

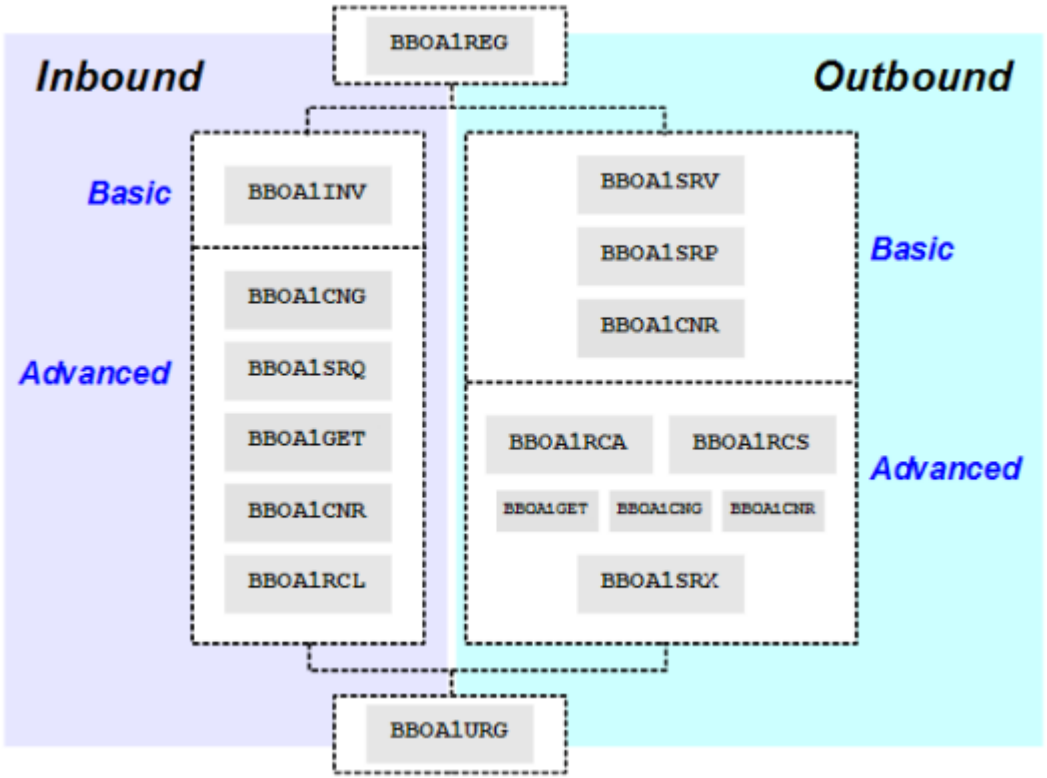
Advanced considerations ...

Finer Details

Drilling down into the next level of detail

Reminder of the API Map Shown Earlier

We'll now discuss the Inbound/Outbound and Basic/Primitive issue a bit more ...



Inbound / Outbound

- Key is who initiates exchange
- Various things available to shield external from outbound APIs
  - CICS link server task
  - IMS OTMA
- Those shielding mechanisms imply *some* overhead; using the APIs can reduce that

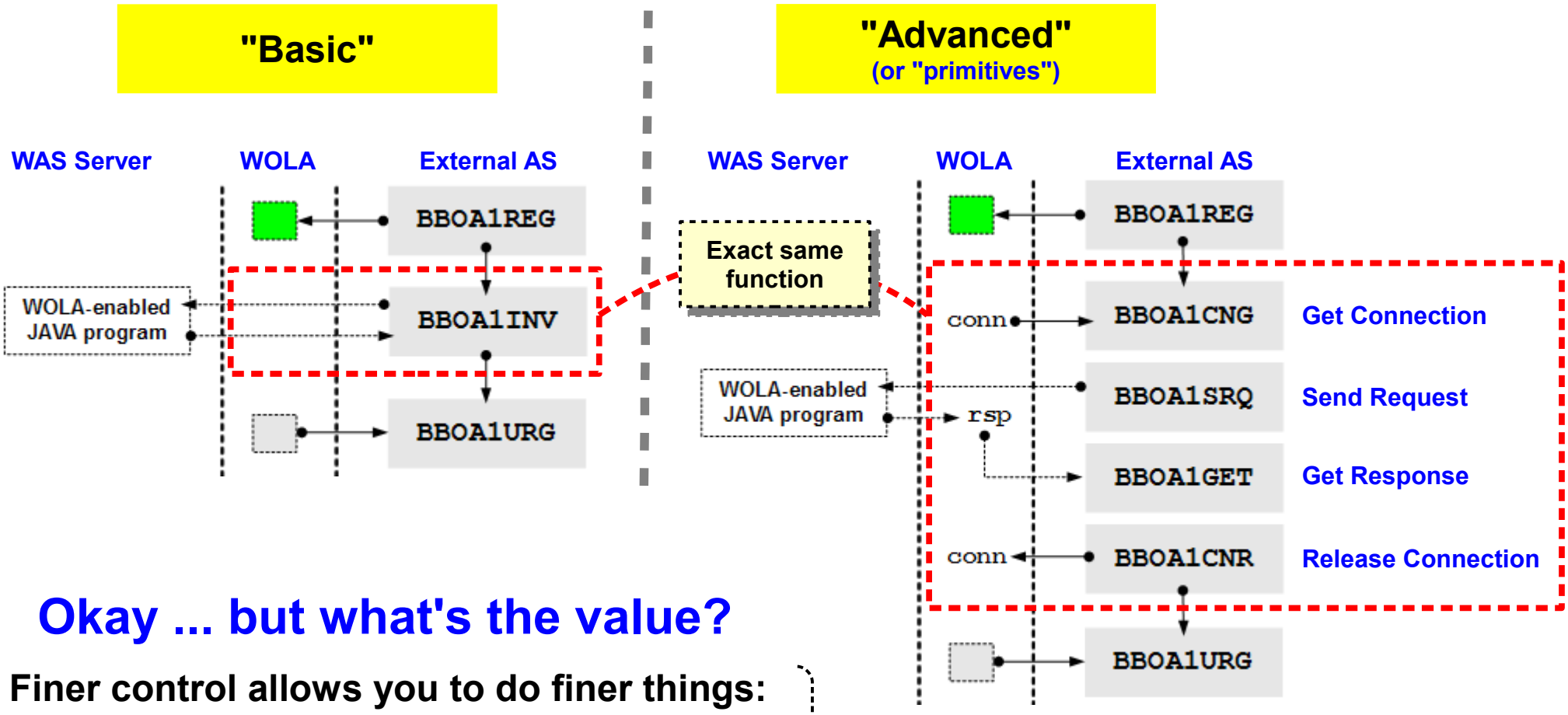
Basic / Advanced

- One set is simple to use but it makes assumptions to keep things simple
- The other set is a bit more complex but it gives you far greater control, which can translate to greater performance

Basic vs. Primitive ...

"Basic" vs. "Primitive" APIs

Let's explore the BBOA1INV API and see that it's really comprised of primitives:



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

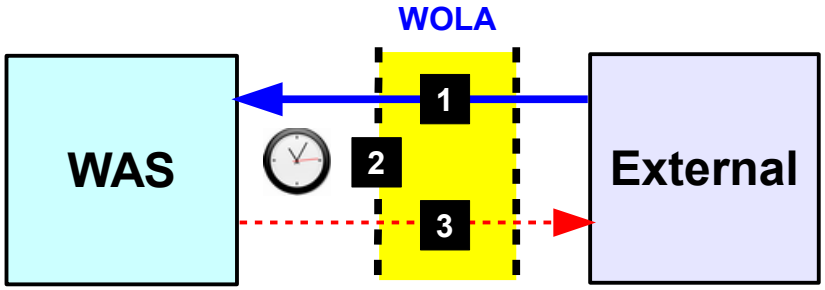
These sorts of things get to the question of performance ...

Synchronous vs. Asynchronous ...

Synchronous vs. Asynchronous

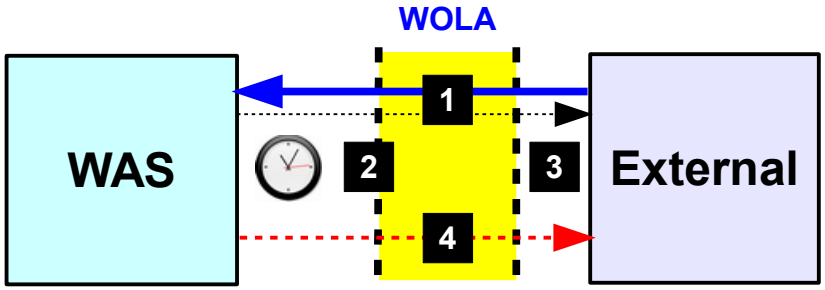
The APIs allow both. In general, synchronous is simpler. But asynchronous allows for potentially greater throughput:

Synchronous



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

Asynchronous



1. External program calls WAS program. Program control is returned to external thread immediately.
2. WAS program processes request.
3. External 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 )`

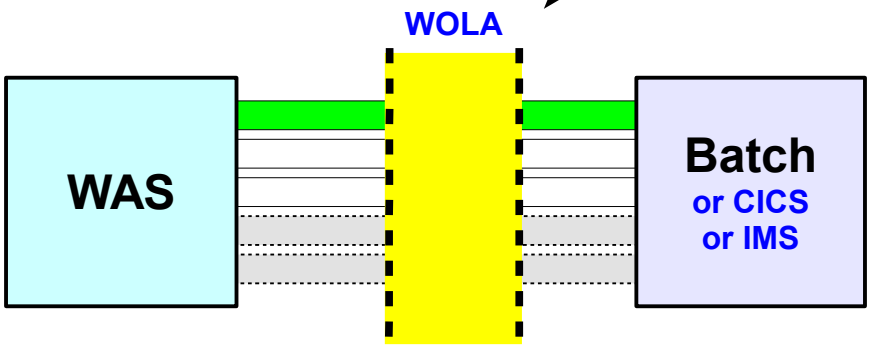
Cell, node and server SHORT

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
```



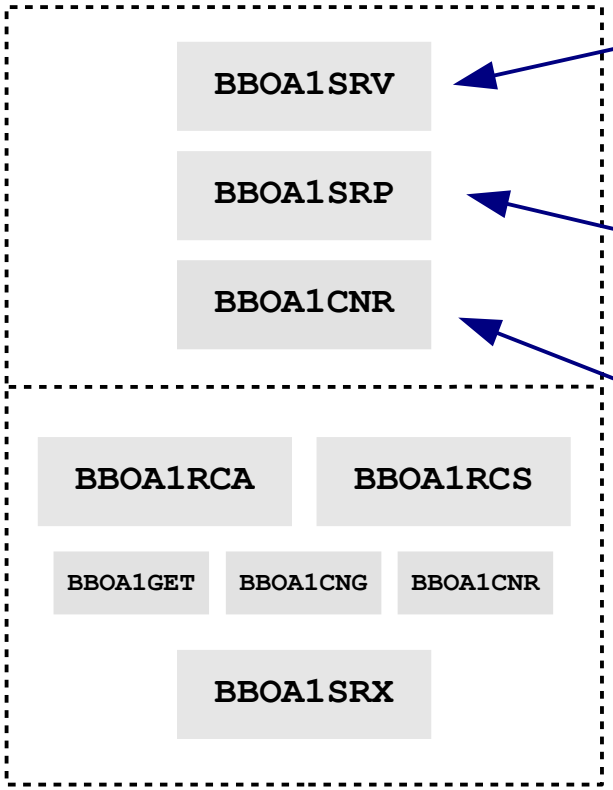
- minconn is the number of connections allocated at registration
- maxconn is the limit of allocations on this registration
- in this example 3 connections have been allocated
- one connection is currently in use
- two connections are allocated and available
- two more could be allocated if needed
-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

InfoCenter search string: `cdat_olaapis`

Host a Service APIs ...

Host a Service ... Establishing "I'm Listening" State

In order for an outbound call to work, the external address space has to put itself in a state where it can accept the call. That's called "hosting a service."



"Host a Service"

BBOA1SRV is a "basic" API. It ...

- Gets a connection from connection pool
- Puts program in listen state
- Holds program control until request received

"Send Response"

BBOA1SRP is used to respond back when the call comes from WAS

"Release Connection"

BBOA1CNR is used to release a connection. Since the basic API BBOA1SRV gets a connection each time, this is how you release it

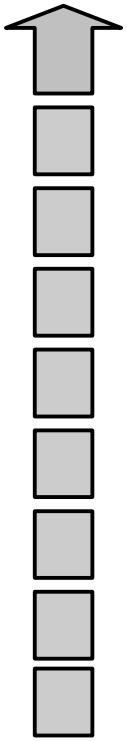
CICS Link Server Task was doing this under the covers, which is how it shields your CICS programs from the APIs

Advanced APIs allow you operate asynchronous, maintain multiple connections, and multi-thread over them as needed

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

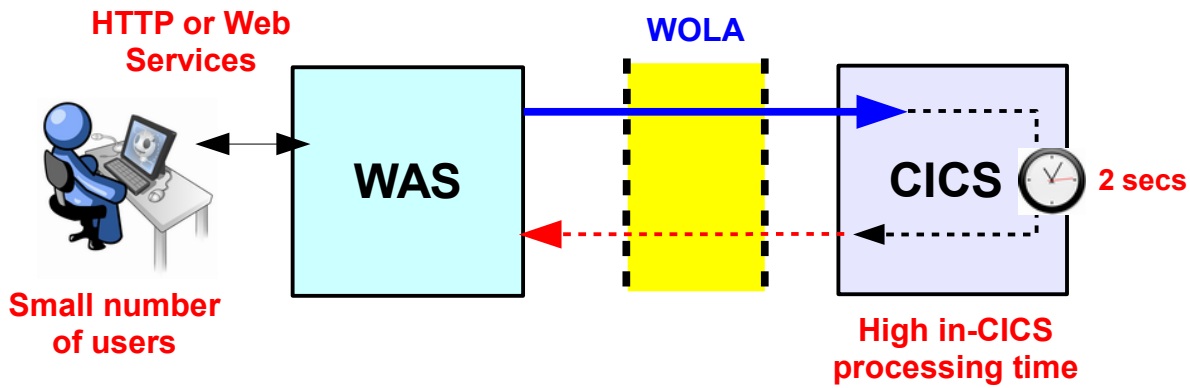
Lesser Performance:

- 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

Summary ...

Overall Summary

Wrapping up

Overall Summary

Functionality

- Cross-memory single-LPAR byte area low-overhead exchange mechanism
- Inbound and outbound; CICS, IMS, Batch, USS and ALCS

Applicability

- Very well suited for inbound to WAS where other solutions may impose unacceptable overhead
- Excellent solution for high-speed batch interchanges

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
- Thread ID in and out for IMS

Transaction

- Two-phase commit inbound and outbound WAS/CICS using RRS as syncpoint coordinator

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

Document Change History

September 12, 2010	Original document
September 11, 2012	Updated the IMS section to reflect TX assertion IMS into WAS
November 12, 2012	Updated the IMS section to reflect TX assertion WAS into IMS over OTMA.