Web Services in CICS - Beyond the basics

Ian J Mitchell, IBM Distinguished Engineer
ianj_mitchell@uk.ibm.com

3rd August 2010

SHARE in Boston

Service Enablement in CICS - Beyond the obvious

Ian J Mitchell, IBM Distinguished Engineer
ianj_mitchell@uk.ibm.com

3rd August 2010

SHARE in Boston

Copyright and Trademarks

Agenda

- Web Services and SOA in CICS TS v4
  - A broader view
- The Pipeline
  - Pipeline Overview
    - Why you might like to exploit the CICS Pipeline
  - The Configuration Files
    - How you incorporate your code into the pipeline processing
  - Custom Handlers
    - How to write your own SOAP header handlers
- Web Services and Exploiters
  - WLM, DataPower, CICS Internal Transports
  - Dynamic Scripting FeaturePack
  - New options with DB2 PureXML
  - Going it alone

Agenda

Web Services and SOA in CICS TS v4

- **A broader view**

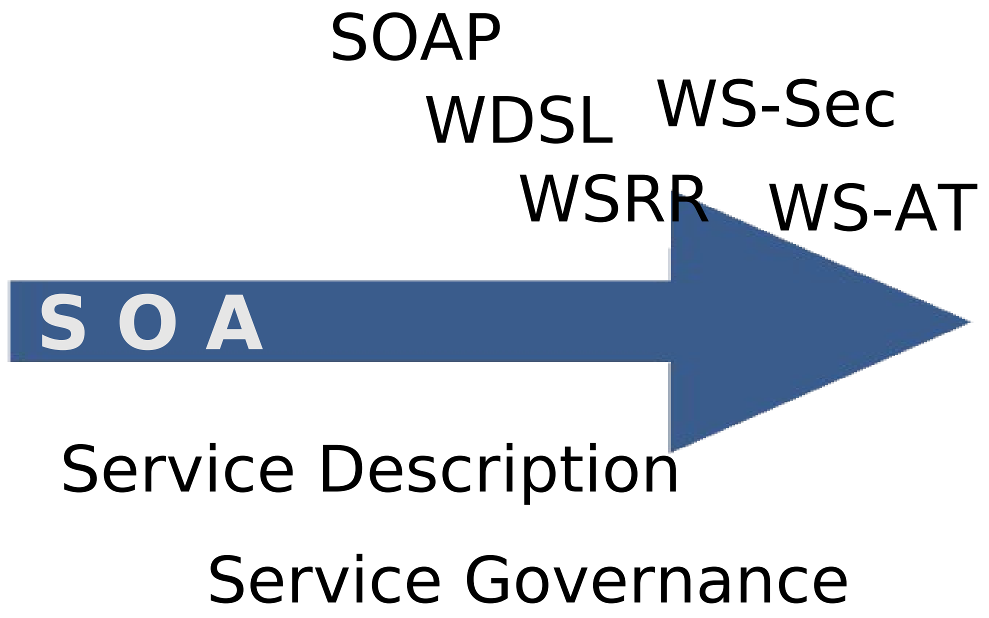
The Pipeline

- Pipeline Overview
  - Why you might like to exploit the CICS Pipeline
- The Configuration Files
  - How you incorporate your code into the pipeline processing
- Custom Handlers
  - How to write your own SOAP header handlers

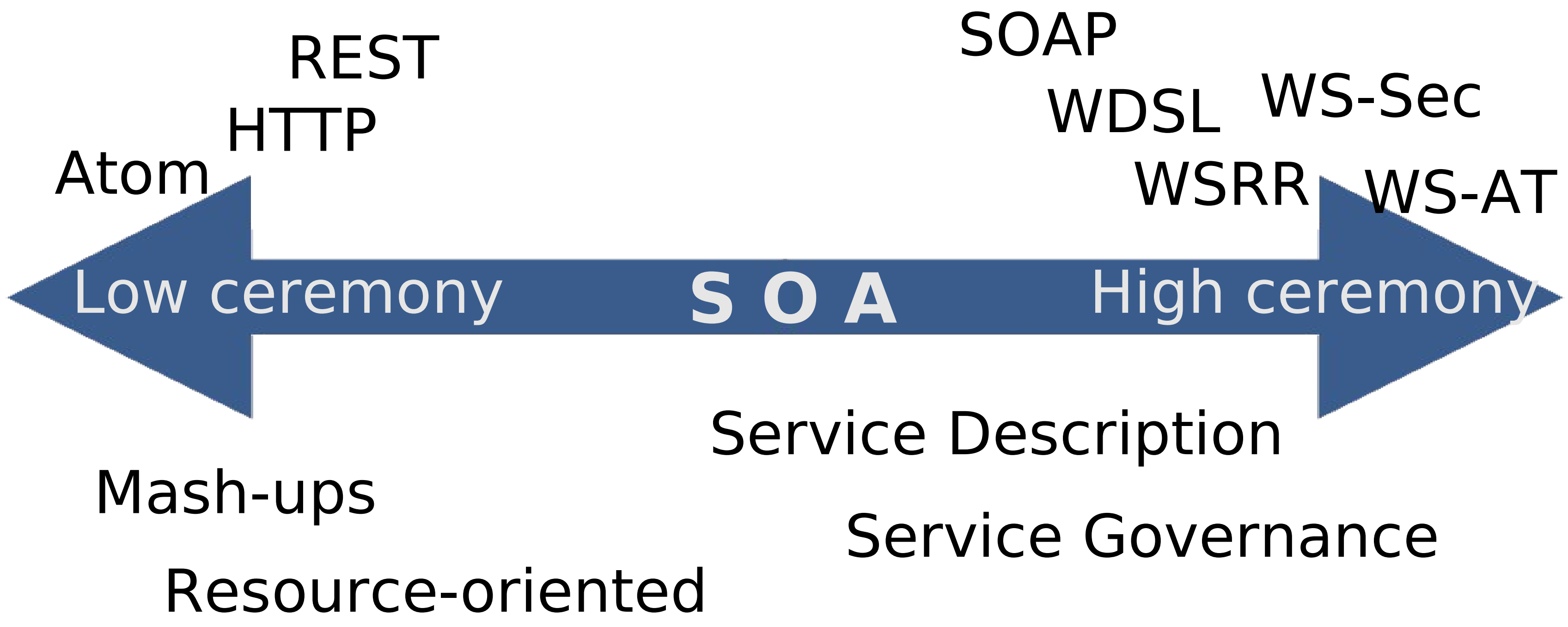
Web Services and Exploiters

- WLM, DataPower, CICS Internal Transports
- Dynamic Scripting FeaturePack
- New options with DB2 PureXML
- Going it alone

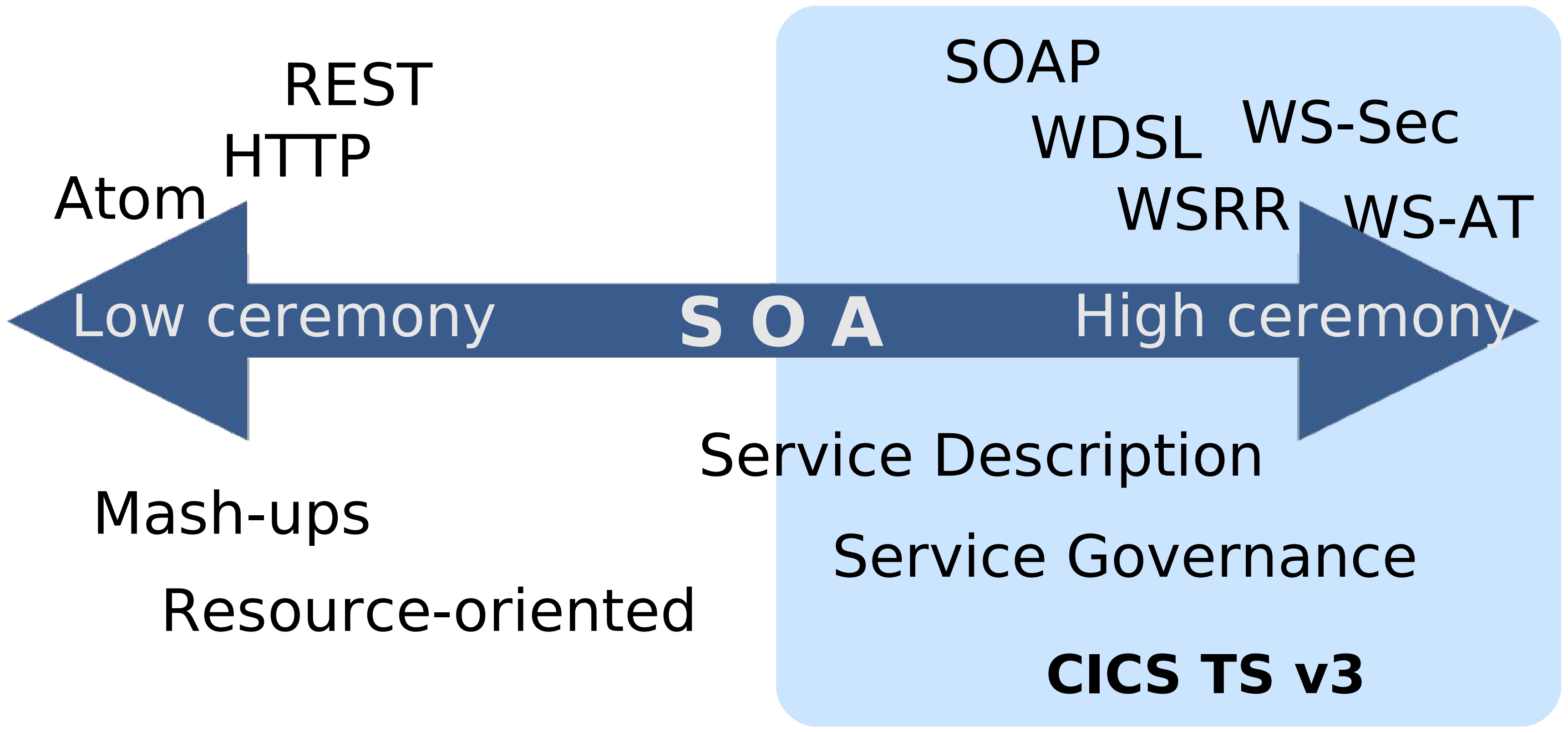
A Spectrum of Service Enablement



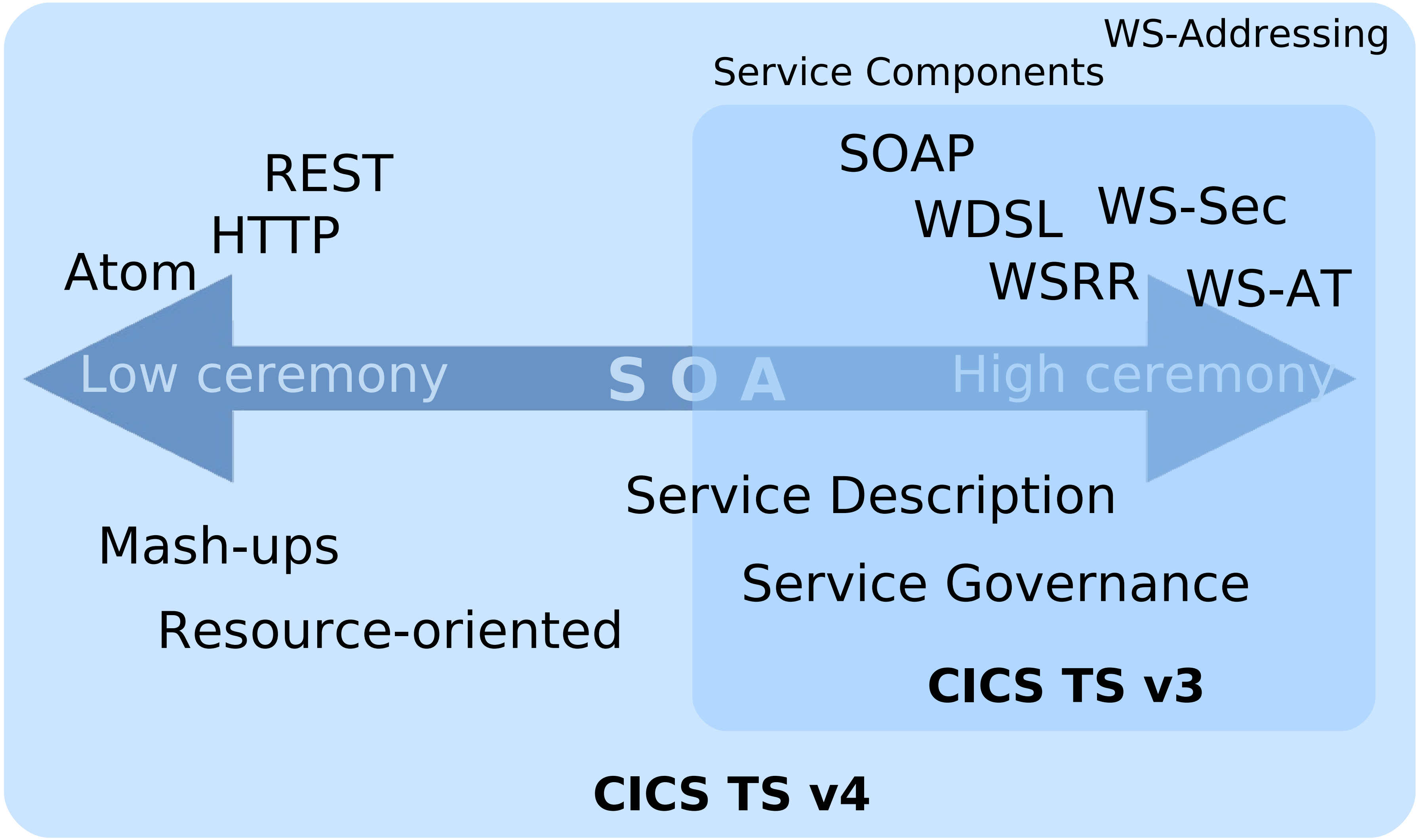
A Spectrum of Service Enablement



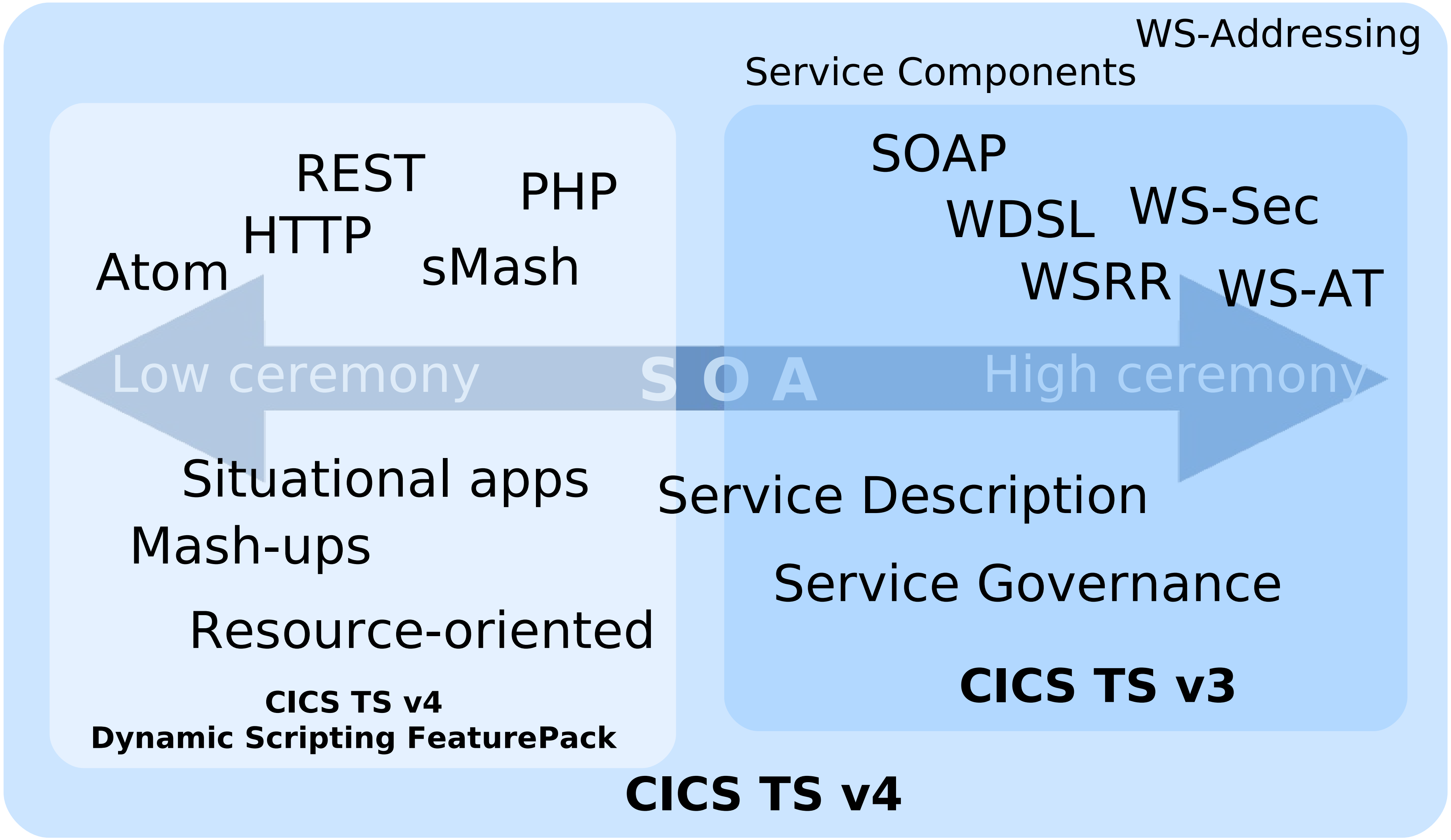
A Spectrum of Service Enablement



A Spectrum of Service Enablement

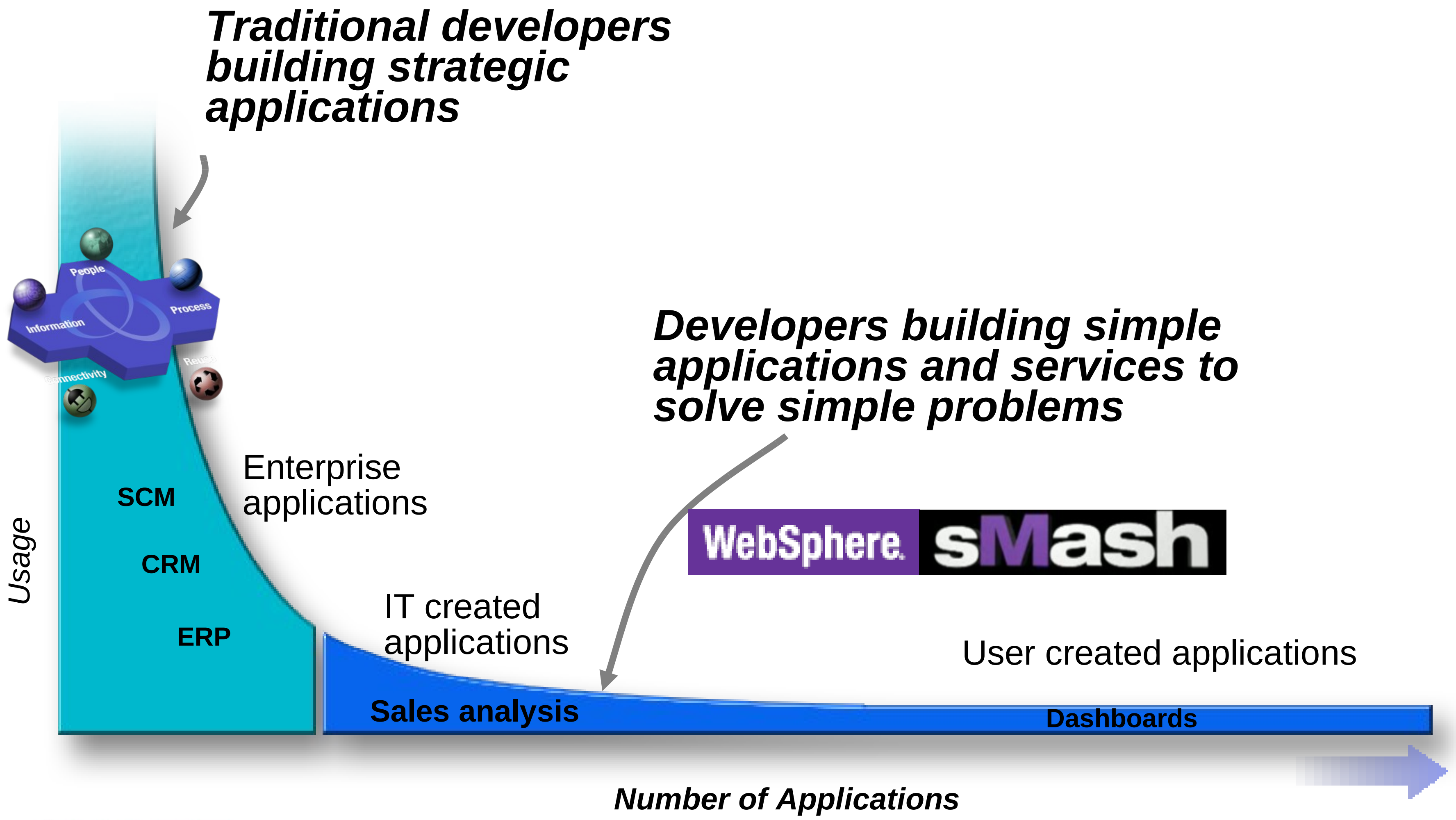


A Spectrum of Service Enablement



SHARE in Boston

The Application Landscape



Agenda

Web Services and SOA in CICS TS v4

- A broader view

The Pipeline

- **Pipeline Overview**
  - **Why you might like to exploit the CICS Pipeline**
- **The Configuration Files**
  - **How you incorporate your code into the pipeline processing**
- **Custom Handlers**
  - **How to write your own SOAP header handlers**

Web Services and Exploiters

- WLM, DataPower, CICS Internal Transports
- Dynamic Scripting FeaturePack
- New options with DB2 PureXML
- Going it alone

Why?... What?...How?...

Why?

Having invented your own message formats and protocols, you've been struggling to manage HTTP or MQ messages with your own infrastructure.

You want to add some function to track, log or monitor all requests to some set of Web service providers.

What?

Using the CICS pipeline can manage the transport, and provide standard system management capabilities built-in to v3.1.

Adding to the CICS-supplied pipeline plugins can augment the standards-based processing

How?

Write some relatively simple components that can be plugged in to the infrastructure provided by CICS.

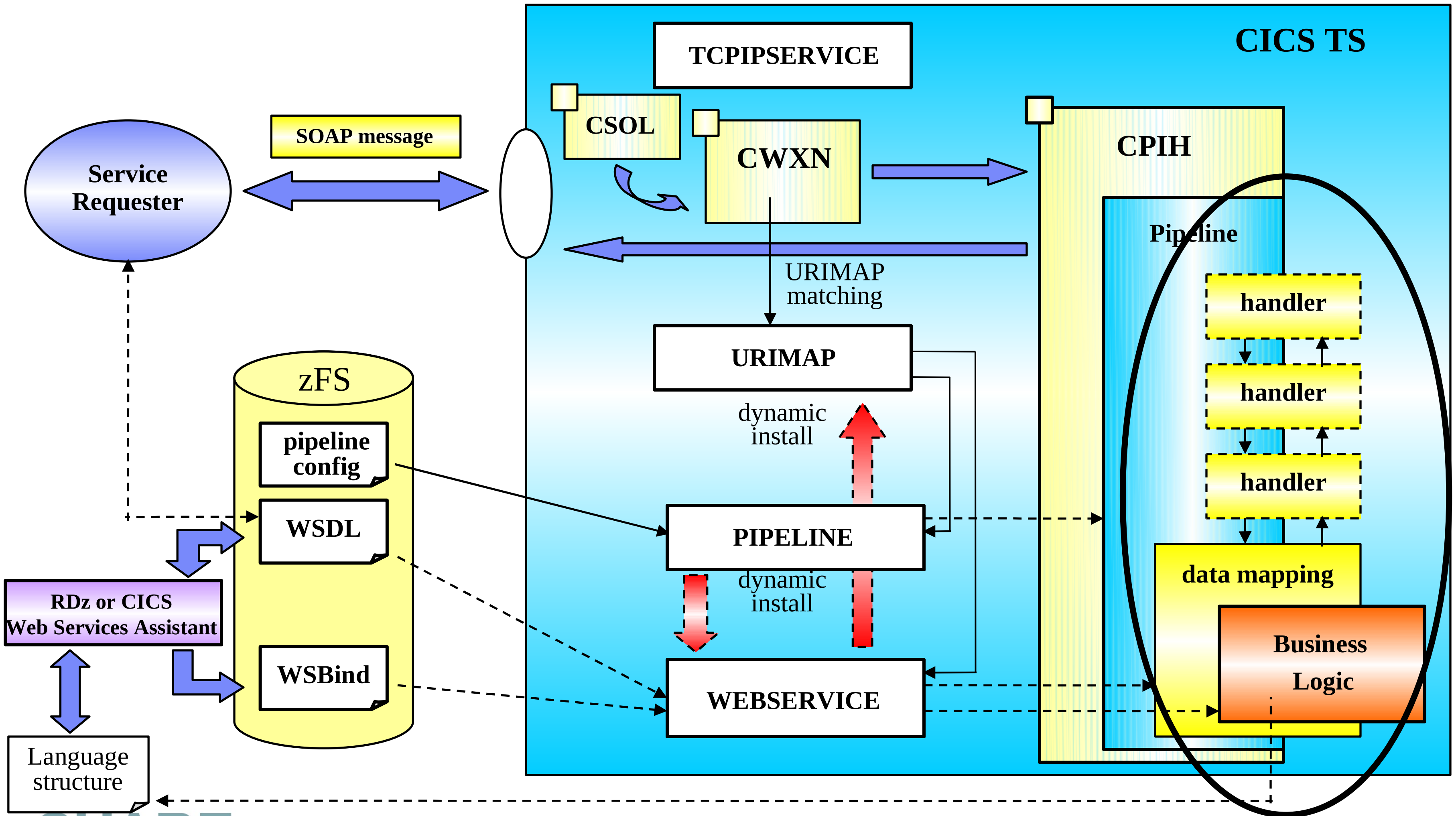
Pipelines

- One request : One pipeline instance : One task
  - Unless context (tranid, userid, ...) changed
- Utilise Channel / Containers for communication
  - One logical channel passed along
- Consist of a sequence of CICS programs invoked in a defined order
  - Transport ➜ message ➜ header
- Designed to deal with request / response type processing
  - But can handle “one-way” messages too
- Can use HTTP & WMQ transports
  - Message and header handlers are transport agnostic

Pipelines continued

- Configured via a XML file
  - Stored in HFS, referenced from a resource definition
- CICS supplied handlers provide SOAP and Web Services functionality
- Customisation could allow for processing of any message format
  - Your own XML grammar, for example
- Used for both CICS as a Provider and Requester
  - Role is an attribute of the pipeline

Resources for CICS as a service provider



Pipeline Message Processing

Pipeline Security

- Transport Level Security
  - SSL with Client Cert can be used to establish a user id for the pipeline transaction.
  - URIMAP can be used to alter the Tran ID (from the default CPIH)
  - WMQ Trigger definitions can be used to similar effect
- The CICS supplied SOAP Handler can switch the Tran Id and User Id of the target application

Pipeline Defintion – CSD example

```
OBJECT CHARACTERISTICS
 Pipeline        : PRODPIPE
 Group           : AGROUP
 Description     :
 STatus          : Enabled            Enabled | Disabled
 Configfile      : /u/ijmitch/pipeprof/prov_w_serv.xml

 (Mixed Case)    :
                 :
                 :
                 :
 SHelf           : /u/ijmitch/pipeshelf
 (Mixed Case)    :
                 :
                 :
                 :
 Wsdir           : /u/ijmitch/pipeline/autoinstall
 (Mixed Case)    :
```

Provider Pipeline Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline

  xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/provider.xsd">

  <transport>
  </transport>

  <service>
    <service_handler_list>
    </service_handler_list>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>


  <apphandler>DFHPITP</apphandler>

  <service_parameter_list />

</provider_pipeline>
```

Provider Pipeline Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline

  xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/provider.xsd ">
```

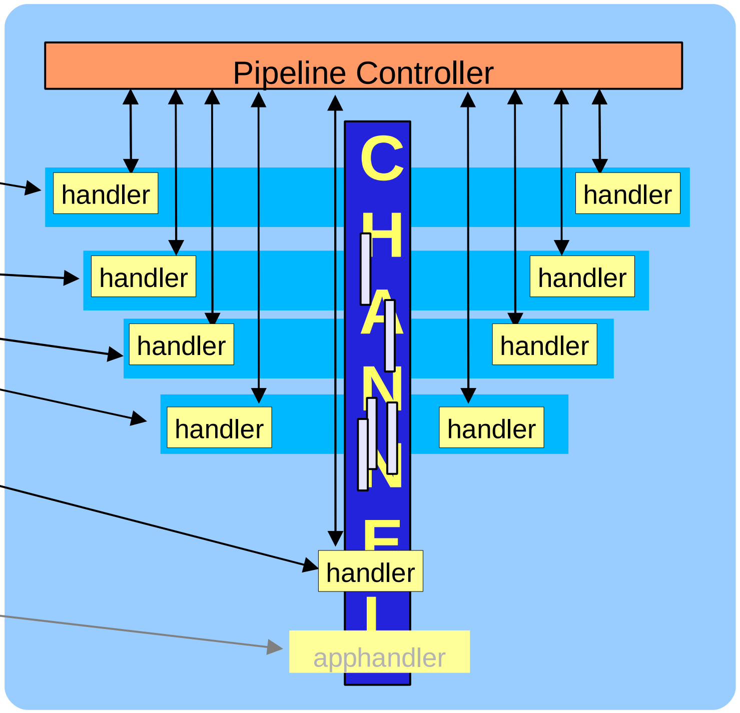
```
<transport>
   :
</transport>
```

```
<service>
  <service_handler_list>
  :
  </service_handler_list>
  <terminal_handler>
    <cics_soap_1.1_handler/>
  </terminal_handler>
</service>
```

```
<apphandler>DFHPITP</apphandler>
```

```
<service_parameter_list />
```

```
</provider_pipeline>
```



SHARE in Boston

Provider Pipeline configuration, detail 1

<transport>

- Optional Element
- Used to add handlers on a per transport basis. These are run between the Transport stage and Service Handlers
- More later…

<service>

- Mandatory Element
- Used to add handlers that are always part of the service.
  - Optional <service_handler_list> contains 0+ handlers
  - Mandatory <terminal_handler> contains the ‘core’ handler for the service

<apphandler>

- Used to populate the DFHWS-APPHANDLER container in the pipeline
- The CICS supplied SOAP handlers use this as the message adaptor program

<service_parameter_list>

- The contents of this element are used to populate the DFH-SERVICEPLIST container in the pipeline. This container is available to all the handlers in the pipeline (global pipeline configuration).

Provider Pipeline Transport element

```
<transport>
  <default_transport_handler_list>
    :
  </default_transport_handler_list>

  <default_http_transport_handler_list>
    :
  </default_http_transport_handler_list>

  <default_mq_handler_list>
    :
  </default_mq_handler_list>

  <named_transport_entry type="http“|”mq”>
    <name>SOAPSERV</name>
    <transport_handler_list>
      :
    </transport_handler_list>
  </named_transport_entry>
</transport>
```

Most specific matching list is picked at runtime.

SHARE in Boston

Provider Pipeline Transport element

- All the transport handler elements are optional
- Each element contains a possible list of handlers that could be used in a pipeline.
- The most specific list applicable is selected
  - Only ONE of the lists is ever used
- <named_transport_entry>
  - Can name a specific TCPIP Service or WMQ Queue.

TIP: Can be used to block access to a pipeline over specific transports

Pipeline Handler Elements

```
<handler>
  <program>PASSTHRU</program>
  <handler_parameter_list>
    [Any XML can go here]
  </handler_parameter_list>
</handler>
```

```
<cics_soap_1.1_handler/>
```

```
<cics_soap_1.2_handler/>
```

```
<wsse_handler/> (For WS-Security)
```

Handler Detail

<handler>

- For user written handlers
- <program> contains the CICS program name of the handler implementation
- <handler_parameter_list> contains xml used populate the DFH-HANDLERPLIST container. This container is passed on the channel to this handler only (Handler specific configuration).

The CICS SOAP Handlers have there own set of configuration elements that we will look at later.

Pipeline Customisation

- Two main ways to customise the processing in a pipeline
  - Custom Message Handlers
  - SOAP Header Processing Programs
- Which to use?
  - Depends on whether the message needs to be transformed
  - Depends on whether the message is SOAP

Custom Message Handlers

- EXEC CICS LINKedto by the Pipeline Manager with a Channel and architected set of Containers
  - Same channel is passed to all handlers in a pipeline so state can be shared
- Have access to the entire message in containers named DFHREQUEST or DFHRESPONSE and can alter either

The Handler Channel

DFHFUNCTION container
(is a 16 character string value)

“RECEIVE-REQUEST”

“SEND-RESPONSE”

“PROCESS-REQUEST”

“SEND-REQUEST”

“RECEIVE-RESPONSE”

“NO-RESPONSE”

“HANDLER-ERROR”

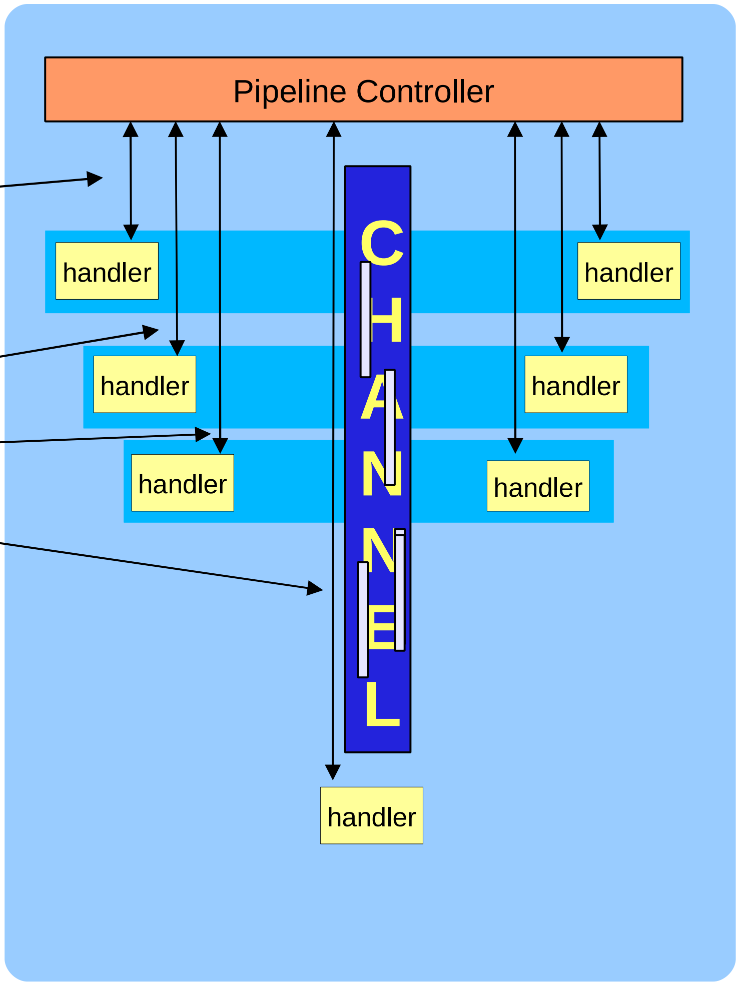
Provider Handlers

Standard Link Order

Transport Invoked

(HANDLER)	***(DFHFUNCTION)***
Handler 1	**Receive-Request**
Handler 2	**Receive-Request**
Handler 3	**Process-Request**
Handler 2	**Send-Response**
Handler 1	**Send-Response**

Transport Invoked



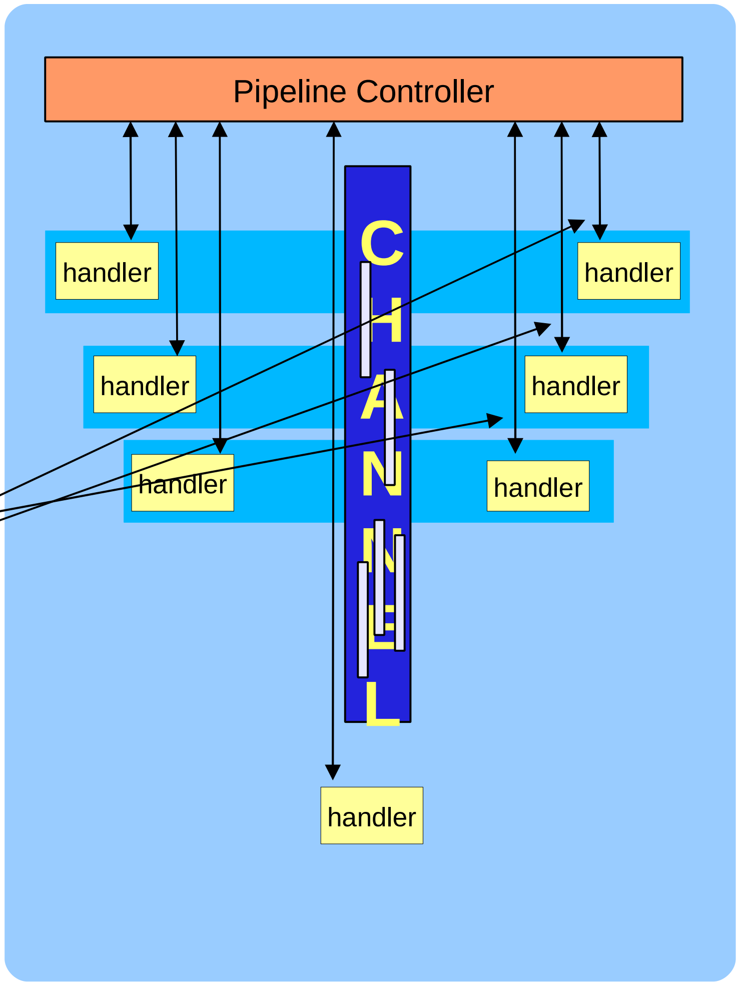
Provider Handlers

Standard Link Order

Transport Invoked

(HANDLER)	***(DFHFUNCTION)***
Handler 1	**Receive-Request**
Handler 2	**Receive-Request**
Handler 3	**Process-Request**
Handler 2	**Send-Response**
Handler 1	**Send-Response**

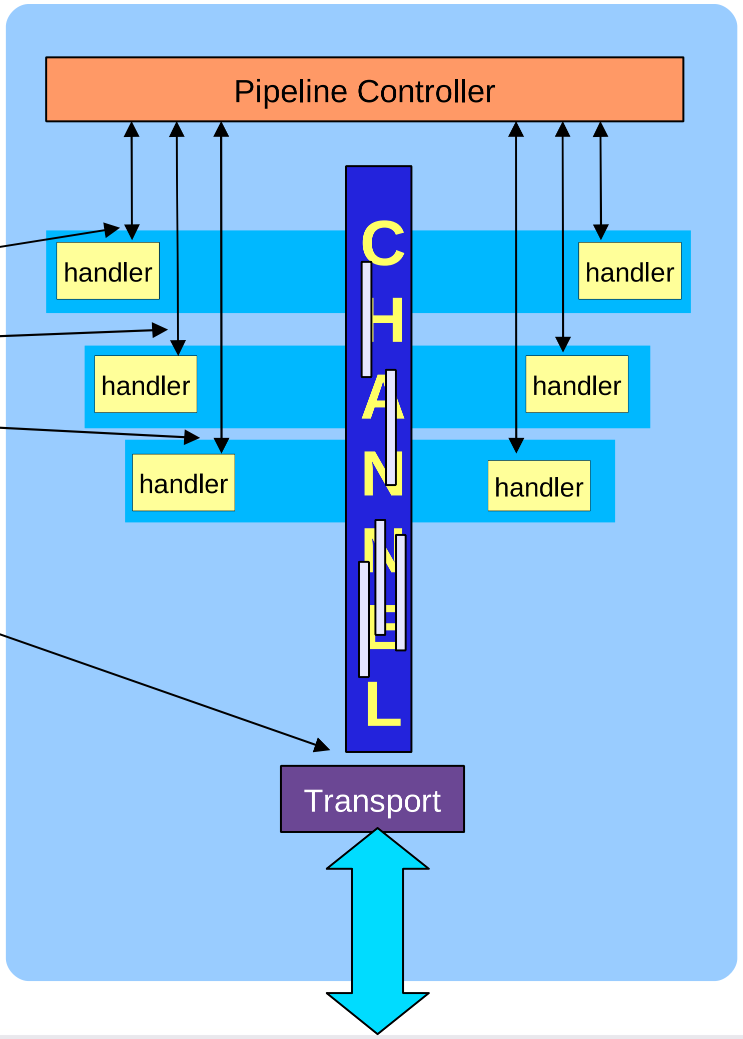
Transport Invoked



Requester Handlers

Standard Link Order

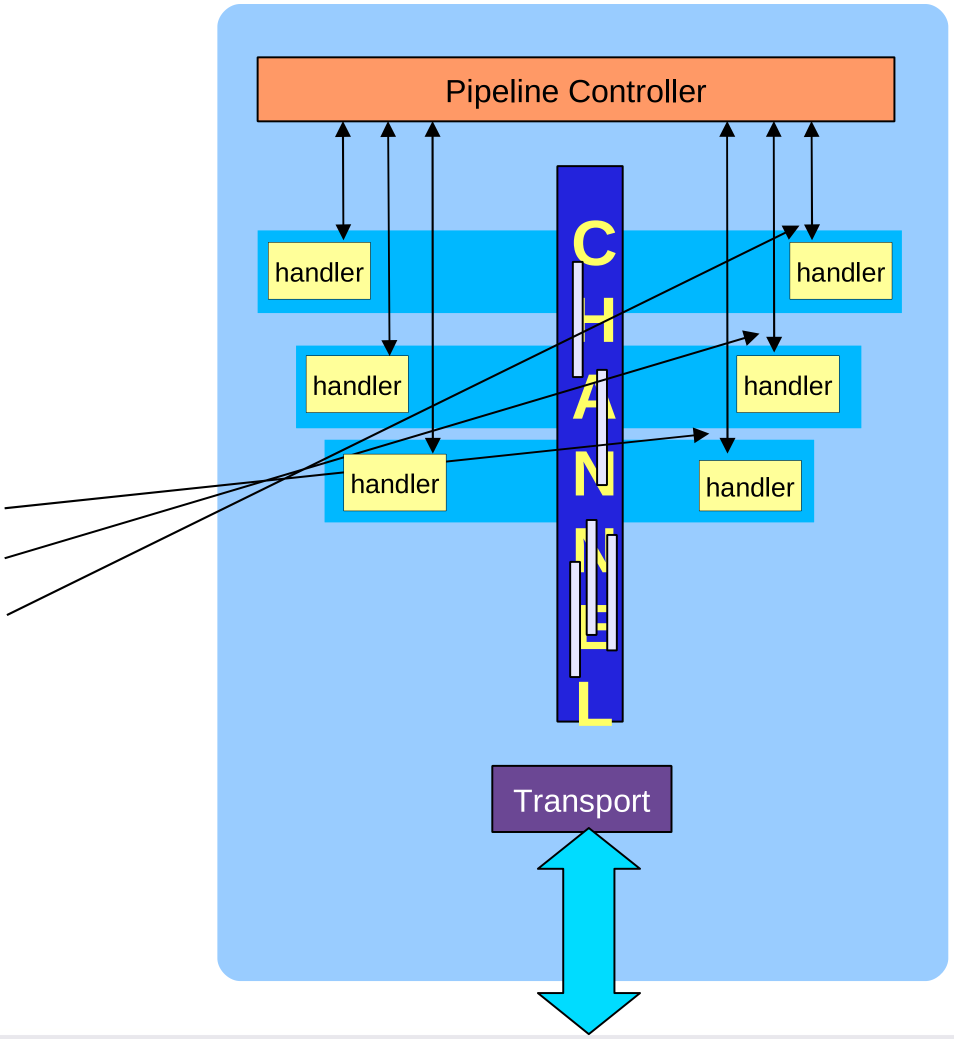
(HANDLER)	***(DFHFUNCTION)***
Handler 1	**Send-Request**
Handler 2	**Send-Request**
Handler 3	**Send-Request**
Transport invoked	
Handler 3	**Receive-Response**
Handler 2	**Receive-Response**
Handler 1	**Receive-Response**



Requester Handlers

Standard Link Order

(HANDLER)	***(DFHFUNCTION)***
Handler 1	**Send-Request**
Handler 2	**Send-Request**
Handler 3	**Send-Request**
Transport invoked	
Handler 3	**Receive-Response**
Handler 2	**Receive-Response**
Handler 1	**Receive-Response**



Additional Functions

PROCESS-REQUEST

Combines RECEIVE-REQUEST & SEND-RESPONSE for the Terminal Handler in a Provider Pipeline

HANDLER-ERROR

Another handler has failed (possibly you), most commonly abended or broke the container contract.

NO-RESPONSE

The handler is being called after processing a request, when there is no response to be processed.

DFHERROR Structure

```
01 PIISNEB.
    02 PIISNEB-MAJOR-VERSION PIC X(1).
    02 PIISNEB-MINOR-VERSION PIC X(1).
    02 PIISNEB-ERROR-TYPE PIC X(1).
    02 PIISNEB-ERROR-MODE PIC X(1).
    02 PIISNEB-ABCODE PIC X(4).
    02 PIISNEB-ERROR-CONTAINER1 PIC X(16).
    02 PIISNEB-ERROR-CONTAINER2 PIC X(16).
    02 PIISNEB-ERROR-NODE PIC X(8).

77 PIIS-NODE-ABEND PIC X(1) VALUE X'01'.
77 PIIS-NULL-CNT PIC X(1) VALUE X'02'.
77 PIIS-NO-CNT PIC X(1) VALUE X'03'.
77 PIIS-EXTRA-CNT PIC X(1) VALUE X'04'.
77 PIIS-NODE-LINKFAIL PIC X(1) VALUE X'05'.
77 PIIS-TRANS-FAILED PIC X(1) VALUE X'06'.
```

Handler Contract – Request Leg

When a handler is called, the channel has:

- DFHREQUEST & DFHRESPONSE are on the channel.
- DFHREQUEST has the request in
- DFHRESPONSE has 0 length

When the handler finishes:

- It is an error for both DFHREQUEST & DFHRESPONSE still to be on the channel

Handler Contract – Response Leg

When a handler is called, the channel has:

- DFHRESPONSE with the response in
- DFHREQUEST is ignored during response processing

When the handler finishes:

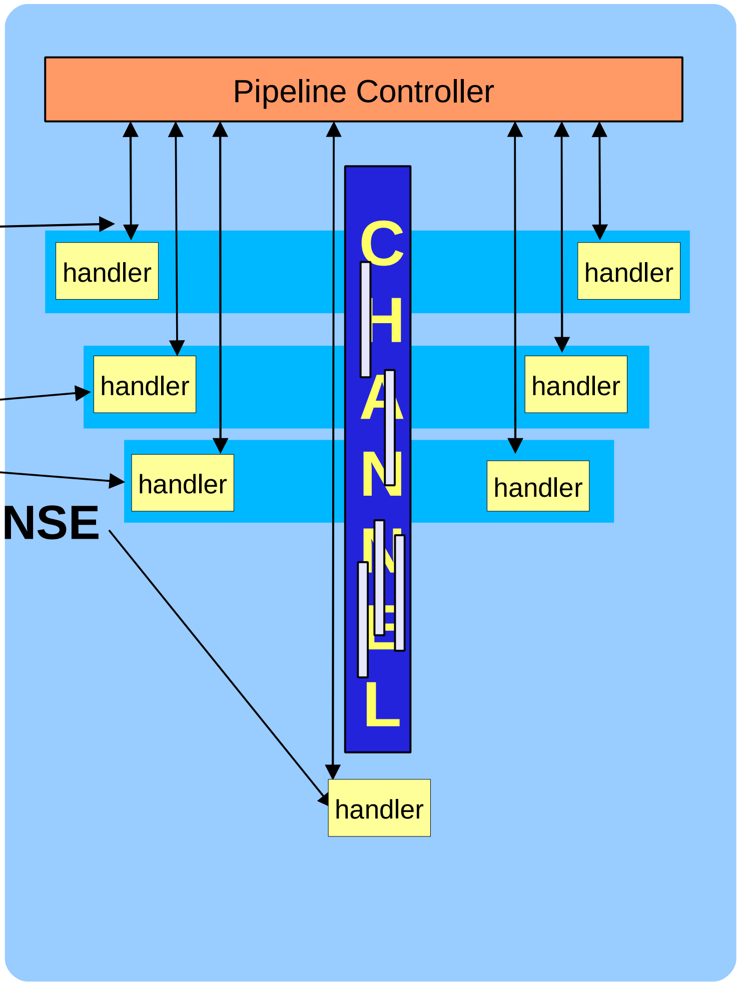
- DFHRESPONSE still present means send-response
- DFHRESPONSE deleted means no-response

Normal Response Example

Transport Invoked

(HANDLER)	*(Action)*
Handler 1	Deletes DFHRESPONSE
Handler 2	Deletes DFHRESPONSE
Handler 3	EXEC CICS PUT DFHRESPONSE
Handler 2	Send-Response
Handler 1	Send-Response

Transport Invoked

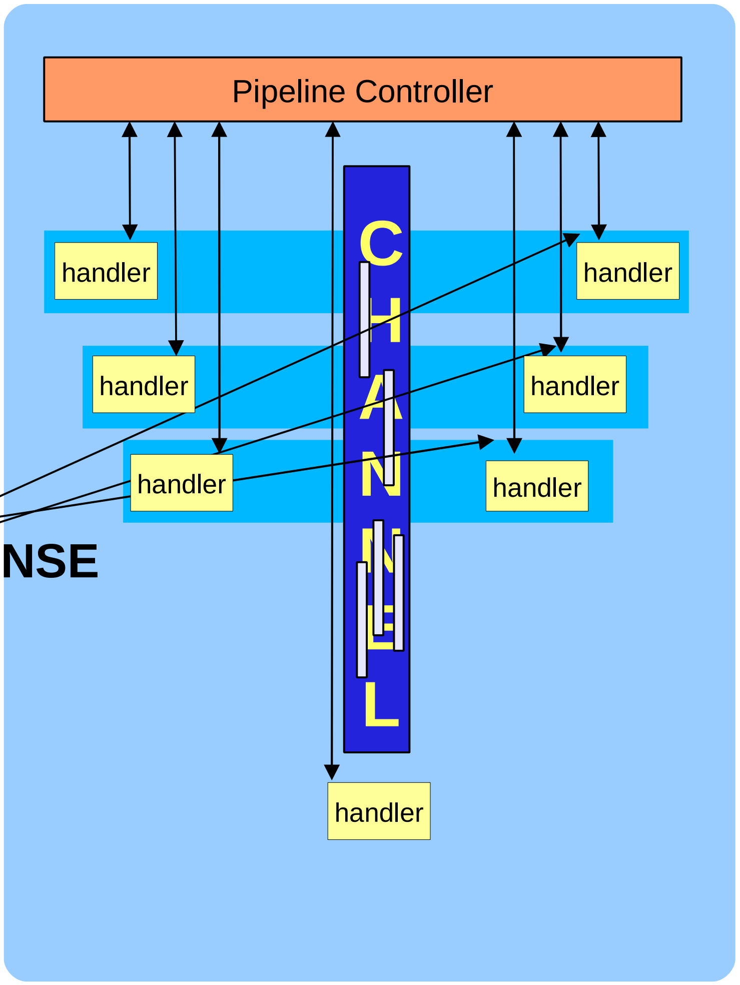


Normal Response Example

Transport Invoked

(HANDLER)	*(Action)*
Handler 1	Deletes DFHRESPONSE
Handler 2	Deletes DFHRESPONSE
Handler 3	EXEC CICS PUT DFHRESPONSE
Handler 2	Send-Response
Handler 1	Send-Response

Transport Invoked



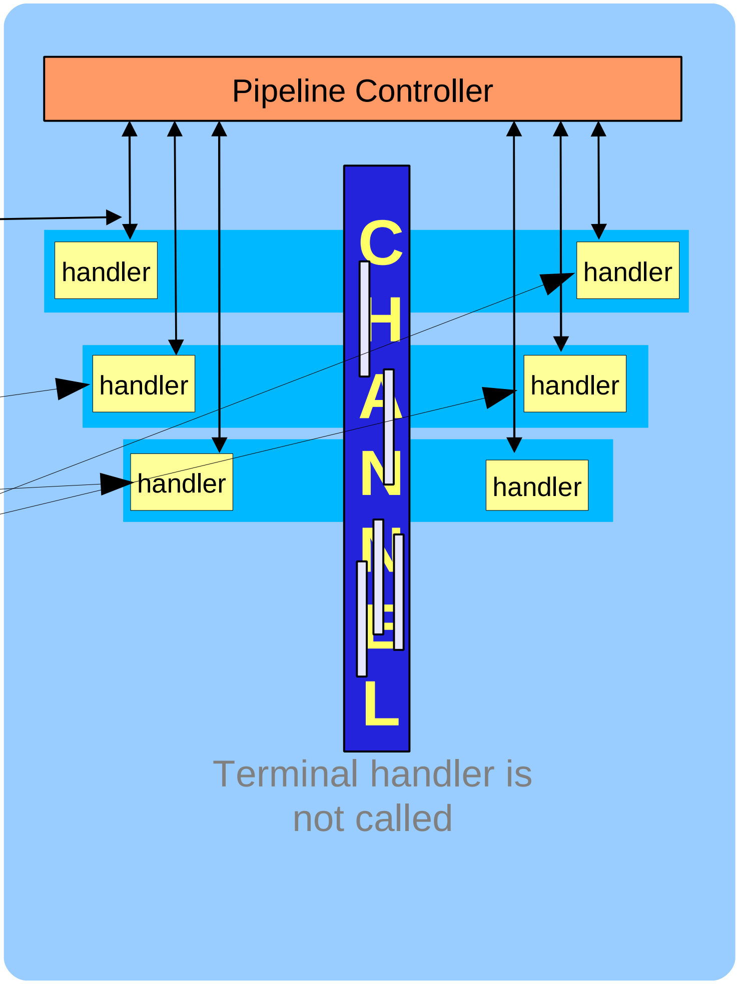
Early Response Example

Transport Invoked

(HANDLER) (Action)

Handler 1	**Deletes DFHRESPONSE**
Handler 2	**Deletes DFHREQUEST**
	PUT DFHRESPONSE
Handler 1	**Send-Response**

Transport Invoked



Pipeline Containers – Control 1

Container DFHERROR

A container of DATATYPE(BIT) that is used to convey information about pipeline errors to other message handlers.

Container DFHFUNCTION

A container of DATATYPE(CHAR) that contains a 16-character string that indicates where in a pipeline a program is being invoked.

Container DFHNORESPONSE

A container of DATATYPE(CHAR) that, in the request phase of a service requester pipeline, indicates that the service provider is not expected to return a response.

Container DFHREQUEST

A container of DATATYPE(CHAR) that contains the request message that is processed in the request phase of a pipeline. If the message was constructed by a CICS-supplied SOAP message handler, and has not been changed subsequently, DFHREQUEST contains a complete SOAP envelope and all its contents in UTF-8 code page.

Container DFHRESPONSE

A container of DATATYPE(CHAR) that contains the response message that is processed in the response phase of a pipeline. If the message was constructed by a CICS-supplied SOAP message handler, and has not been changed subsequently, DFHRESPONSE contains a complete SOAP envelope and all its contents in UTF-8 code page.

Pipeline Containers – Context 1

Container DFH-HANDLERPLIST

A container of DATATYPE(CHAR) that is initialized whenever a CICS-provided SOAP handler is invoked, with the contents of the appropriate <handler_parameter_list> element of the pipeline configuration file.

Container DFH-SERVICEPLIST

A container of DATATYPE(CHAR) that contains the contents of the <service_parameter_list> element of the pipeline configuration file.

Container DFHWS-APPHANDLER

A container of DATATYPE(CHAR) that, in a service provider pipeline, is initialized with the contents of the <apphandler> element of the pipeline configuration file.

Container DFHWS-DATA

A container of DATATYPE(BIT) that is used in a service requester application deployed with the CICS Web services assistant. It holds the top level data structure that is mapped to and from a SOAP request.

Pipeline Containers – Context 2

Container DFHWS-OPERATION

A container of DATATYPE(CHAR) that is normally used in a service provider application deployed with the CICS Web services assistant. It holds the name of the operation that is specified in a SOAP request.

Container DFHWS-PIPELINE

A container of DATATYPE(CHAR) that contains the name of the PIPELINE in which the program is being run.

Container DFHWS-SOAPLEVEL

DFHWS-SOAPLEVEL is a container of DATATYPE(BIT) that holds information about the level of SOAP used in the message that you are processing.

Container DFHWS-TRANID

DFHWS-TRANID is a container of DATATYPE(CHAR) that is initialized with the transaction ID of the task in which the pipeline is running.

Pipeline Containers – Context 3

Container DFHWS-URI

A container of DATATYPE(CHAR) that, in a service provider, is initialized with the URI of the service. CICS extracts the URI from the incoming message.

Container DFHWS-USERID

A container of DATATYPE(CHAR) that is initialized with the user ID of the task in which the pipeline is running.

Container DFHWS-WEBSERVICE

A container of DATATYPE(CHAR) that is used in a service provider pipeline only. It holds the name of the WEBSERVICE that specifies the execution environment when the target application has been deployed using the Web services assistant.

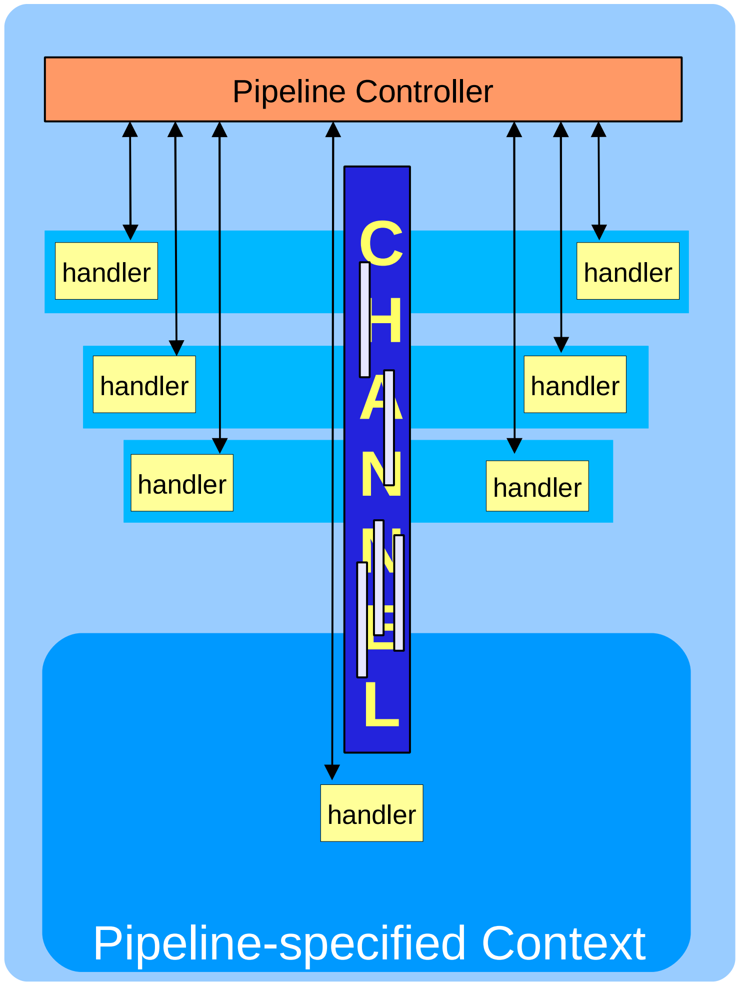
Context Switching

When the terminal handler is a CICS SOAP Handler it is possible to context switch the Application Handler.

If DFHWS-TRANID or DFHWS-USERID have been changed from the default then the new values will be used for a new task to run the business logic.

If the values are unchanged the logic would run in the pipeline task as normal.

EXPERIENCE LESSON: IF YOU CHANGE THE USERID, YOU SHOULD CHANGE THE TRANID AS WELL! *(Avoid TCLASS deadlocks)*



Pipeline Containers – User

- Custom Handlers can add any containers they wish to the channel
- Should avoid names starting ‘DFH’ as CICS may architect containers with these names in future support.

Handler Pros & Cons

Pros (benefits)

Can transform whole message

Not just for SOAP

Full flexibility to reply early or not reply at all

Always invoked

Invocation order fixed by config file

Cons (disadvantages)

If processing SOAP must be aware of protocol issues (SOAP level)

Cannot use the EXEC SOAPFAULT API

Always invoked

Processing SOAP:
SOAP Header Handlers and the Application Handler

SOAP Header Handling Programs

- Can be attached via the config file to a CICS SOAP Message Handler
- Program is called if header is in SOAP message
- More limited than Custom Message Handlers (cannot transform the whole message, can only add and remove headers and transform the body).
- Are given the header as well as the message to save parsing time

Adding a Header Processing Program

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline>
  xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline/provider.xsd">
  <service>
    <service_handler_list/>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

Adding Header Processing

A header handler specified thus...

```
<cics_soap_1.1_handler>
<headerprogram>
  <program_name>WSSEHEAD</program_name>
  <namespace>http://schemas.xmlsoap.org/ws/2003/07/secext</namespace>
    <localname>Security</localname>
    <mandatory>false</mandatory>
</headerprogram>
</cics_soap_1.1_handler>
```

.... would be called to process a header encountered of the following form....

```
 <wsse:Security
        xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/07/secext">
        <wsse:UsernameToken wsu:ID="myToken">
           <wsse:Username>fraser</wsse:Username>
           <wsse:Password>passw0rd</wsse:Password>
        </wsse:UsernameToken>
 </wsse:Security>
```

DFHHEADER

- The main container for Header Processing
- For RECV functions
    - On Input has the matched header
    - On Output has the headers to be added
- For SEND Functions
    - On Input is normally empty (unless you are using multiple Handlers)
    - On Output has the headers to be added

The APPHANDLER program

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline

  xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/provider.xsd ">

  <transport>
  </transport>

  <service>
    <service_handler_list>
    </service_handler_list>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>


  <apphandler>DFHPITP</apphandler>

  <service_parameter_list />

</provider_pipeline>
```

Application Handler – Going it your own

- Called from a CICS SOAP Handler to invoke application logic
- CICS supplies one (DFHPITP) which uses the Web Service resource to do this
- Can be user written and is passed these containers
    - DFHWS-BODY
        - DFHNORESPONSE to signal no response
        - Or update - process the request and put response SOAP body into the container
    - DFHWS-XMLNS
        - Same as for header handlers
    - Context containers

DFHWS-XMLNS

When the header processing program is invoked, DFHWS-XMLNS contains information about XML namespaces that are declared in the SOAP envelope. The header program can use this information:

to resolve qualified names that it encounters in the header block

to construct qualified names in new or modified header blocks.

The namespace information consists of a list of namespace declarations, which use the standard XML notation for declaring namespaces. The namespace declarations in DFHWS-XMLNS are separated by spaces.

For example:

```
 xmlns:na='http://abc.example.org/schema'
    xmlns:nx='http://xyz.example.org/schema'
```

You can add further namespace declarations to the SOAP envelope by appending them to the contents of DFHWS-XMLNS. However, namespaces whose scope is a SOAP header block or a SOAP body are best declared in the header block or the body respectively.

DFHWS-BODY

- Contains the body section of the SOAP envelope. The header handler may modify the contents. When the header processing program is invoked, DFHWS-BODY contains the SOAP <Body> element.
- When the header program returns, container DFHWS-BODY must again contain a valid SOAP <Body>, which CICS inserts in the SOAP message in place of the original:
  - You can return the original body unchanged.
  - You can modify the contents of the body.
- You must not delete the SOAP body completely, as every SOAP message must contain a <Body> element.

Header Handler Pros & Cons

Pros (benefits)

Targeted invocation

Less need to worry about SOAP Protocol issues

Can use the SOAPFAULT API

Cons (disadvantages)

Cannot Transform the whole message, only body and targeted header.

Can only fault or continue, cannot switch to no-reply

Agenda

Web Services and SOA in CICS TS v4

- A broader view

The Pipeline

- Pipeline Overview
  - Why you might like to exploit the CICS Pipeline
- The Configuration Files
  - How you incorporate your code into the pipeline processing
- Custom Handlers
  - How to write your own SOAP header handlers

Web Services and Exploiters

- **WLM, DataPower, CICS Internal Transports**
- **Dynamic Scripting FeaturePack**
- **New options with DB2 PureXML**
- **Going it alone**

Workload Management and Context Switching

WLM and Context Switching

•The context switch can be to a remote region (MRO only).

•It uses the distributed routing program.

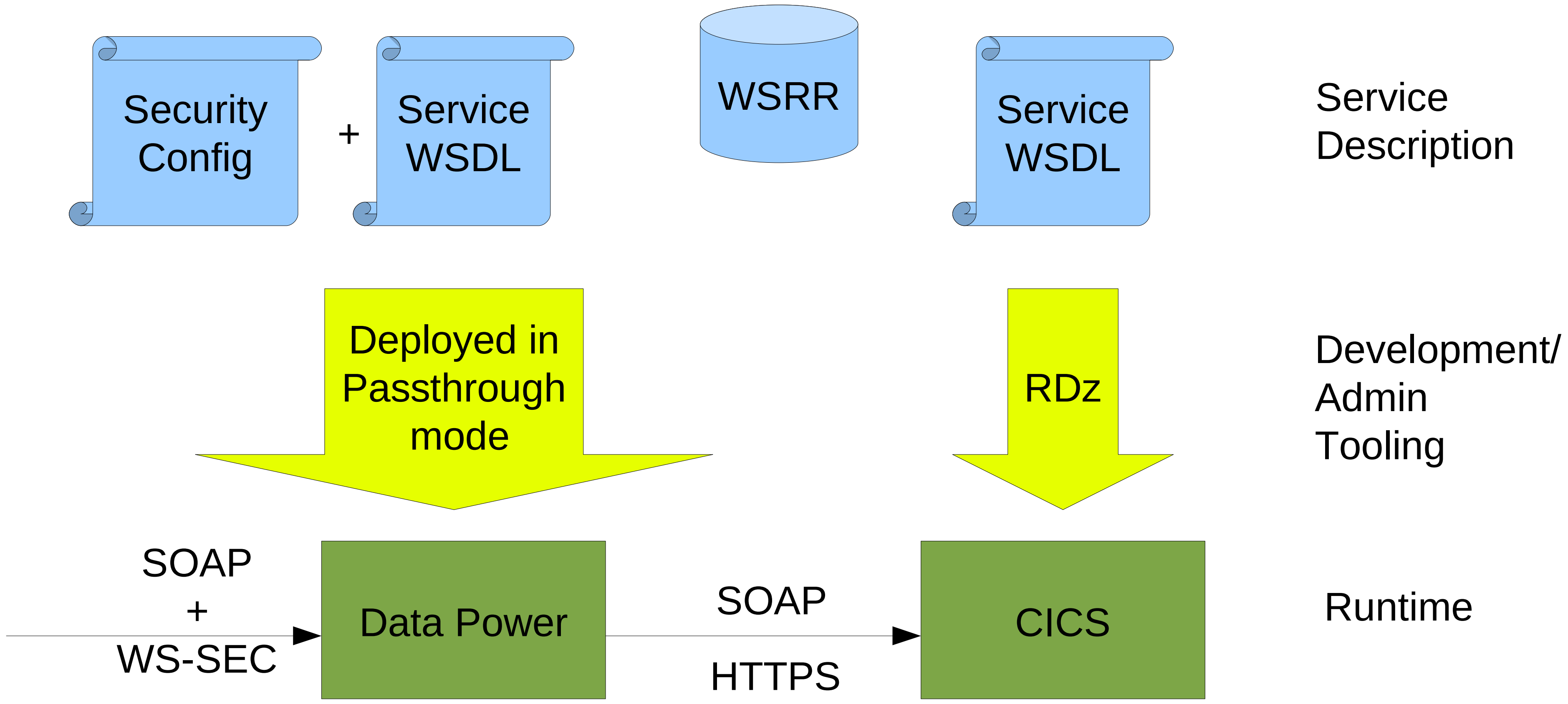
•Can only be routed to a CICS TS v3 or later region as a channel is being passed.

•Any EXEC CICS LINK (DPL) from within the pipeline can be subject to WLM

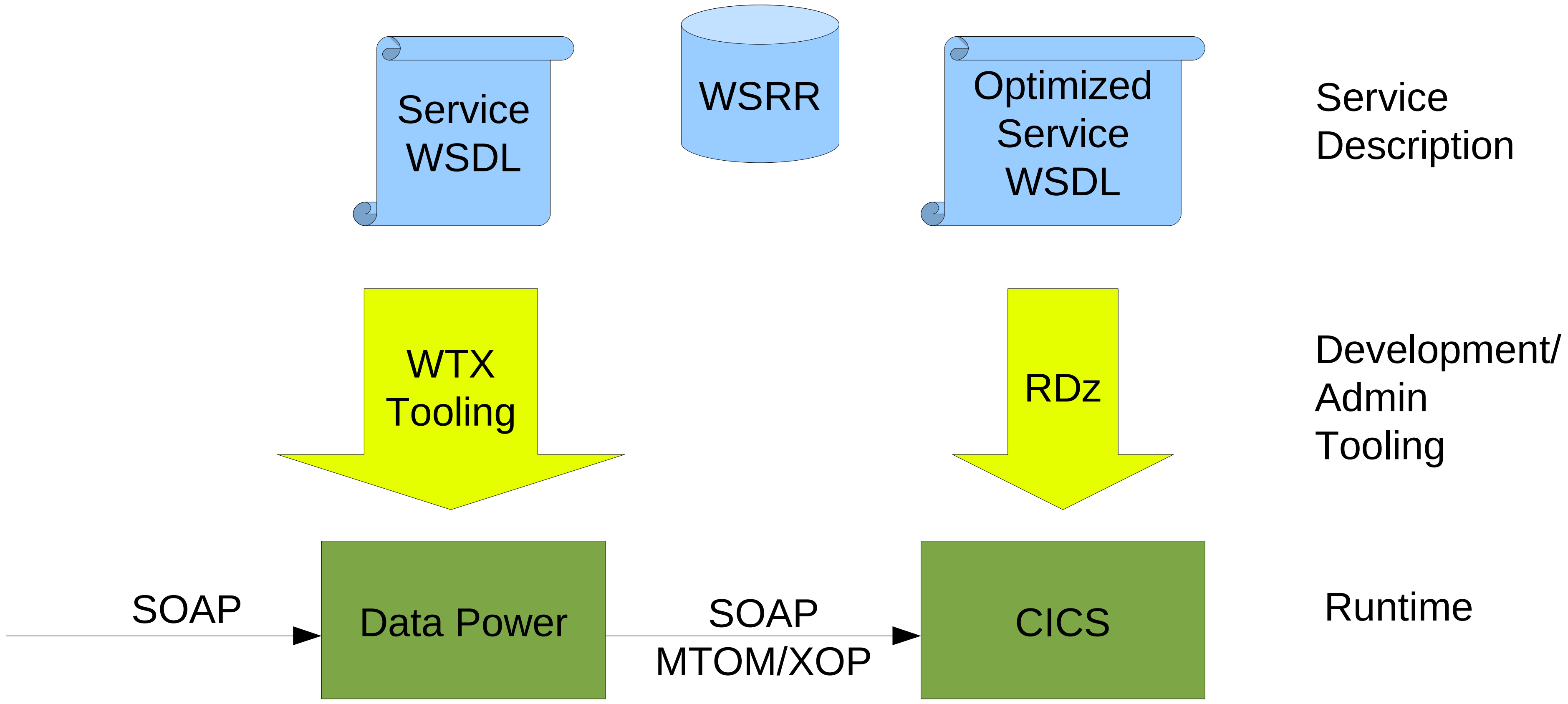
•This includes the link made by the CICS provided Application handler to the application logic

CICS and WebSphere DataPower

Datapower for Security - Provider



Datapower for Body Transformation - Provider



Service views without MTOM/XOP

Datapower – Full WSDL
<Service>
<operation>
<schema = full xml message>

Example message

```
<s11:Body>
<myRequest>
<myData1>...</myData1>
<myData2>...</myData2>
...
</myRequest>
</s11:Body>
```

CICS - Optimized WSDL
<Service>
<operation>
<schema = single base64 encoded element which maps commarea>

Example message

```
<s11:Body>
<myRequest>Z6FD2KJ...
</myRequest>
</s11:Body>
```

CICS Message with MTOM XOP

```
--MIME Boundary

<s11:Envelope>
<s11:Body>
<myRequest><xop:include id=”attachement1”/>
</myRequest>
</s11:Body>
</s11:Envelope>

--MIME Boundary, id=”attachement1”
############# <-real binary

--MIME Boundary
```

RD/z Support for MTOM/XOP

RD/z 7.6 includes support for generating this style web service and bindfile

Will extract data from the MTOM/XOP container and Link to the application program with a commarea

On return it repopulates the MTOM/XOP container with the response commarea

CICS Internal Transport

CICS PIPELINE Internal Transport

The V3 PIPELINE has two transport types based on URI

- HTTP / HTTPS use HTTP Transport
- WMQ / JMS use the WMQ Transport

Adding a CICS Transport that uses internal services rather than the network for CICS <-> CICS service calls

- New CICS URI Format

A more flexible version of the local optimization that already exists for Web Services

CICS Transport URI format

- Have the request target a program
  - `cics://program/MYPROG`
- Have the request target a Service (Provider pipeline)
  - `cics://service/myservice?targetService=/myProviderApp/ServiceUri`
- Can be used to create your own Provider transport types
- Have the request run a second Requester Pipeline
  - `cics://pipeline/MYPIPE`

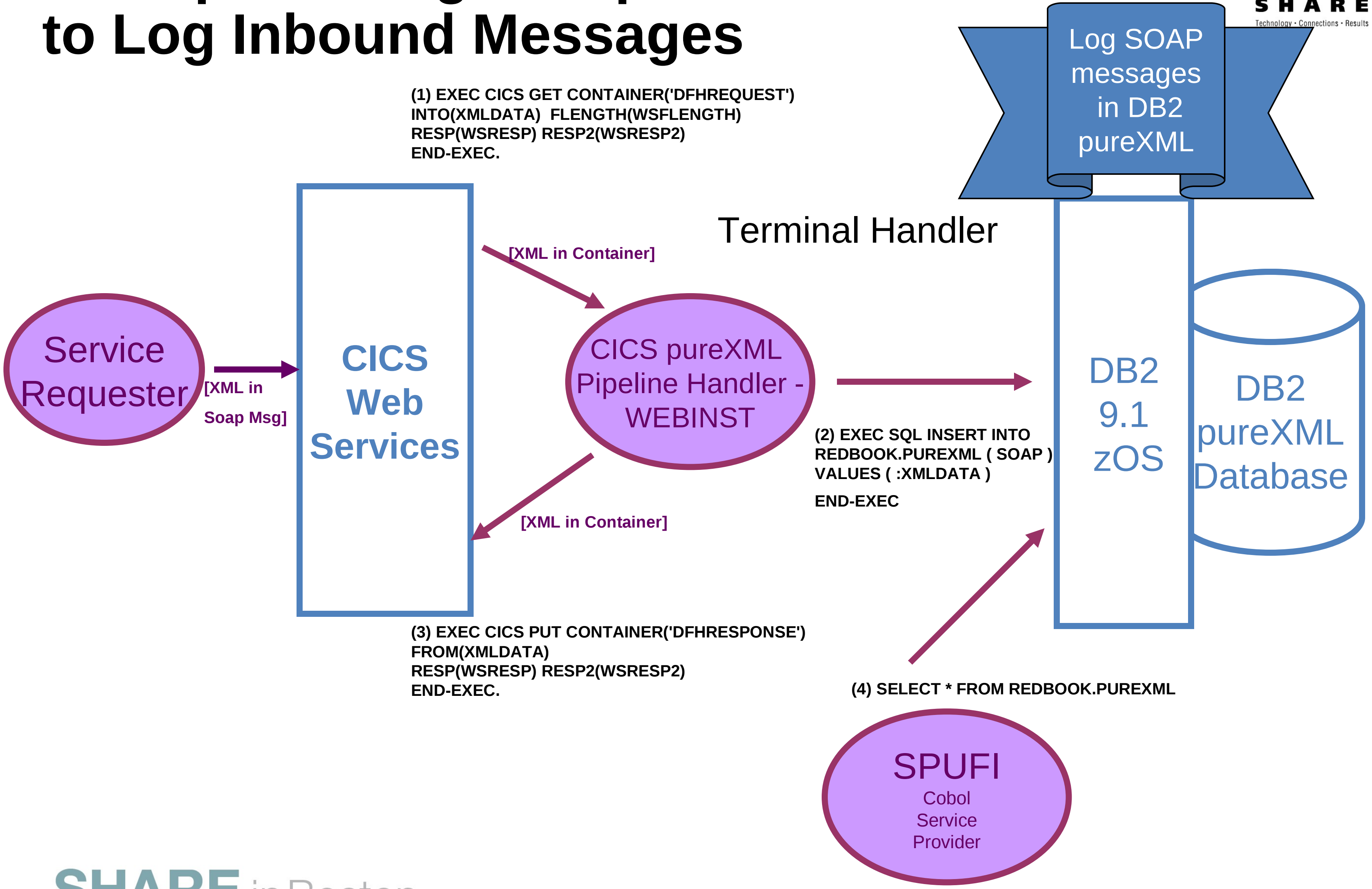
New options with DB2 PureXML

DB2 pureXML – Features

With DB2 9 you can:

- Store well-formed XML messages in the database as collections of XML (in an XML column of type XML)
  - Query stored XML messages
    - Using XQuery including update – UNIX and Windows only
    - Using SQL/XML
    - Using some combination of XQuery, XPath, and SQL/XML
  - Optionally index the XML messages
  - Optionally validate the XML messages with respect to an XML schema
  - Store documents with different XML structures in the same column
    - Helps with schema evolution
  - Get started with industry formats such as FpML through pureXML industry bundles
  - In summary, you can process many XML messages with one declarative request – [contrast with XML storage in files]
- Store XML messages as relational data
  - Shredding technology makes it possible to transform XML into relational data by using either the XMLTABLE function or the annotated schema shred

Example: Using DB2 pureXML to Log Inbound Messages



```
(1) EXEC CICS GET CONTAINER('DFHREQUEST')
INTO(XMLDATA)  FLENGTH(WSFLENGTH)
RESP(WSRESP) RESP2(WSRESP2)
END-EXEC.
```

```
(2) EXEC SQL INSERT INTO
REDBOOK.PUREXML ( SOAP )
VALUES ( :XMLDATA )
END-EXEC
```

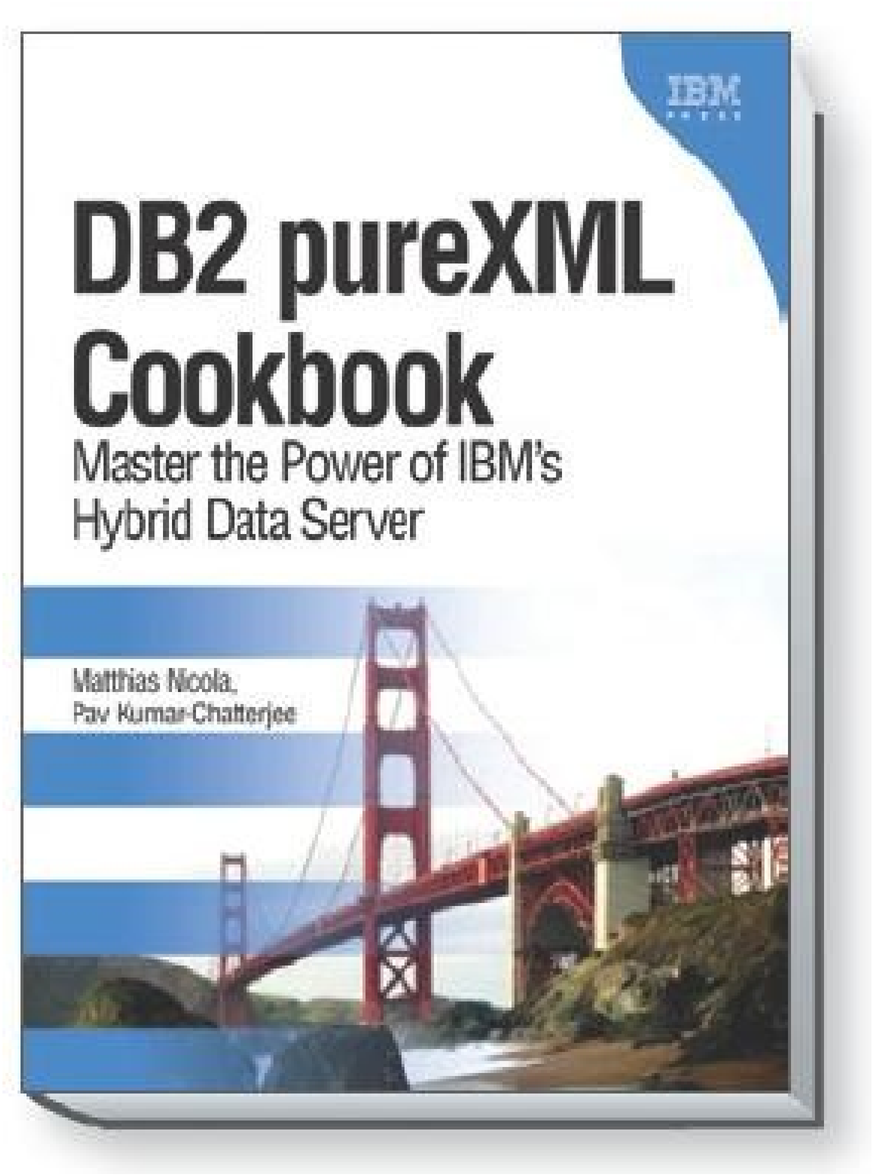
```
(3) EXEC CICS PUT CONTAINER('DFHRESPONSE')
FROM(XMLDATA)
RESP(WSRESP) RESP2(WSRESP2)
END-EXEC.
```

```
(4) SELECT * FROM REDBOOK.PUREXML
```

Using CICS with DB2 pureXML Scenarios and use cases

DeveloperWork Article

- **Summary:** This article provides an introduction to using DB2® pureXML® with CICS® applications written in Common Business Oriented Language (COBOL). XML is playing an increasingly important role in CICS applications. Therefore, the need to store and query XML in CICS applications is growing. This article describes two scenarios for using CICS with DB2 pureXML. The first scenario shows how to store inbound XML Web service messages in DB2 pureXML without first parsing the messages in CICS. The second shows how a CICS application can retrieve XML data from DB2 and transmit it through a Web service. The article provides sample source code that you can download.
- http://www.ibm.com/developerworks/data/library/techarticle/dm-1004cicsdb2purexml/index.html
- Download includes the set-up and source code for WEBINST


IBM Press
DB2 pureXML Cookbook
Master the Power of IBM's Hybrid Data Server
Matthias Nicola, Pav Kumar-Chatterjee

- Comprehensive coverage of pureXML in DB2 for Linux, UNIX, Windows and DB2 for z/OS
- **http://tinyurl.com/pureXML**

Dynamic Scripting Featurepack

What is Project Zero?

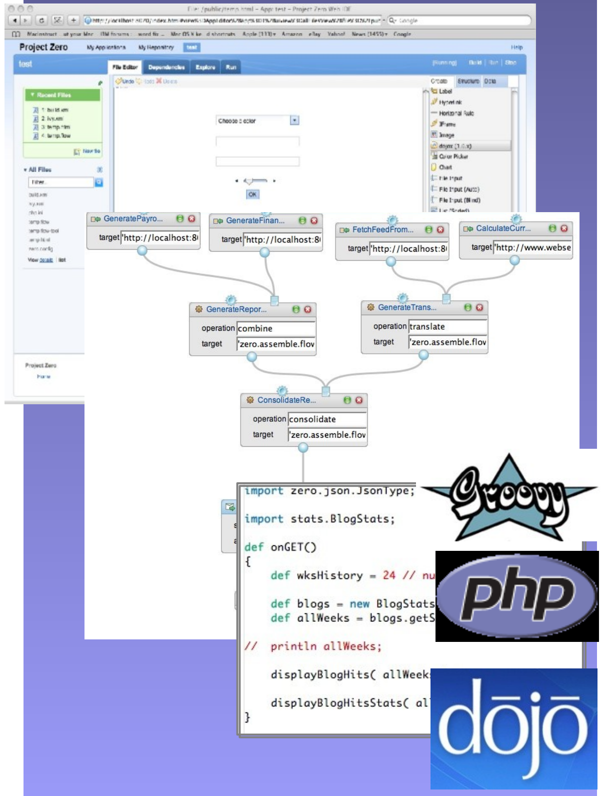
- Project Zero is a new Agile Web Application Platform
  - Leveraging Dynamic Scripting Languages
  - Optimized for Producing...
    - REST-based Services
    - Integration Applications
    - Mash-ups
    - Rich Web Interfaces
  - Architected for...
    - Speed
    - Simplicity
    - Agility

- **The Dynamic Scripting Featurepack hosts the Project Zero runtime in a CICS JVMServer**
  - Successor to SupportPac CA1S
  - (Equivalent Featurepack for WAS - except on z/OS)

Project Zero

- Speed
  - Dynamic scripting languages (PHP and Groovy – with Java as system language)
  - Core application constructs: templates, pre-built services
- Simplicity
  - Built-in browser-based composition tools: Visual tools (for web page construction or scripting activities into a flow) and scripting tools (server-side dynamic scripting)
  - REST-style architecture – simple ways to expose and consume services
- Agility
  - Simple deployment (application "is" the server)
  - Runtime Characteristics (clean, cost effective, short-lived)

Lightweight platform with browser-based composition Tools



```
import zero.json.JsonType;
import stats.BlogStats;
def onGET()
{
    def wksHistory = 24 // nu
    def blogs = new BlogStats
    def allWeeks = blogs.getS
//  println allWeeks;
    displayBlogHits( allWeek
    displayBlogHitsStats( al
}
```

CICS Dynamic Scripting FeaturePack

- Newly available for CICS TS 4.1
- Embed Zero programming model into CICS on z/OS
  - Script CICS assets using Groovy or PHP.
  - Build a web 2.0 AJAX presentation layer onto CICS Programs.
  - Expose CICS assets as RESTful services.
- Tight integration with existing CICS application assets and data
  - Easy and efficient access to COBOL assets
  - Inheriting CICS run time QoS

CICS Dynamic Scripting FeaturePack - Delivers

- Agility
  - Simple programming model, light-weight or zero tooling
- Access to CICS application assets and data
  - Efficient, affordable, high QoS
- Safe architecture
  - Inherit CICS and z/OS strengths (access control, auditability, instrumentation, ...)

What is REST?

- REST is the acronym for “Representational State Transfer”
  - It is the architectural model on which the World Wide Web is based

- Principles of REST
  - Resource centric approach
  - All relevant resources are addressable via URIs
  - Uniform access via HTTP – GET, POST, PUT, DELETE
  - Content type negotiation allows retrieving alternative representations from same URI

- REST style services
  - are easy to access from code running in web browsers, any other client or servers

- More info: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Restful Application Resources

RESTful Design

- Collection Model
- Action can be taken on the entire collection or a specified member of the collection
- URI and HTTP method define the resource request

HTTP Verb	URI	Action
GET	/people	List all members
POST	/people	Create new member
GET	/people/1	Retrieve single member
PUT	/people/1	Update member
DELETE	/people/1	Delete Member

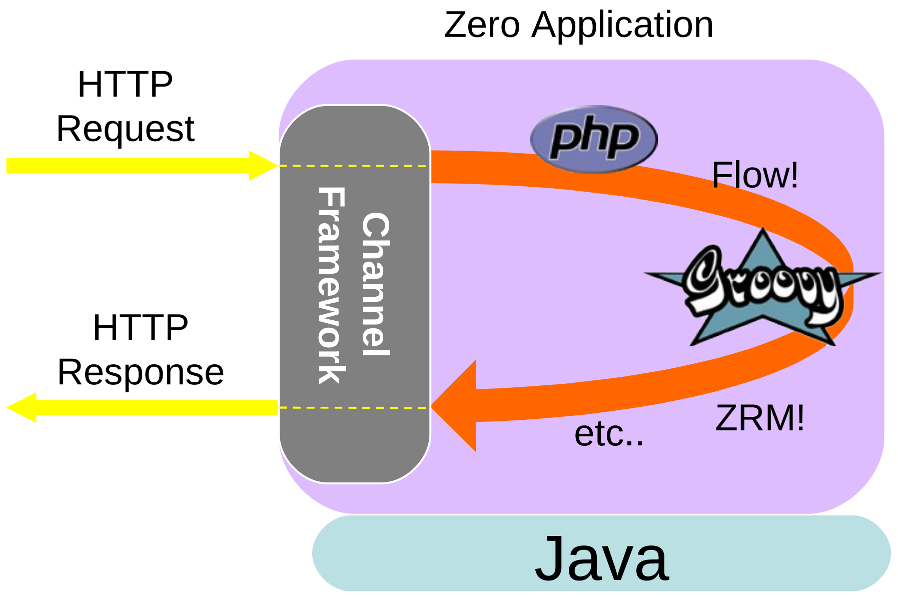
REST and Project Zero

- Project Zero supports
  - URI and HTTP method define the collection resource model
  - Each script in the <apphome>/app/resources directory represents a resource handler
  - URL convention for interacting with resources based on
  - /resources/<collectionName>[/<memberID>[/<pathInfo>]]
  - where the actions are defined as follows:

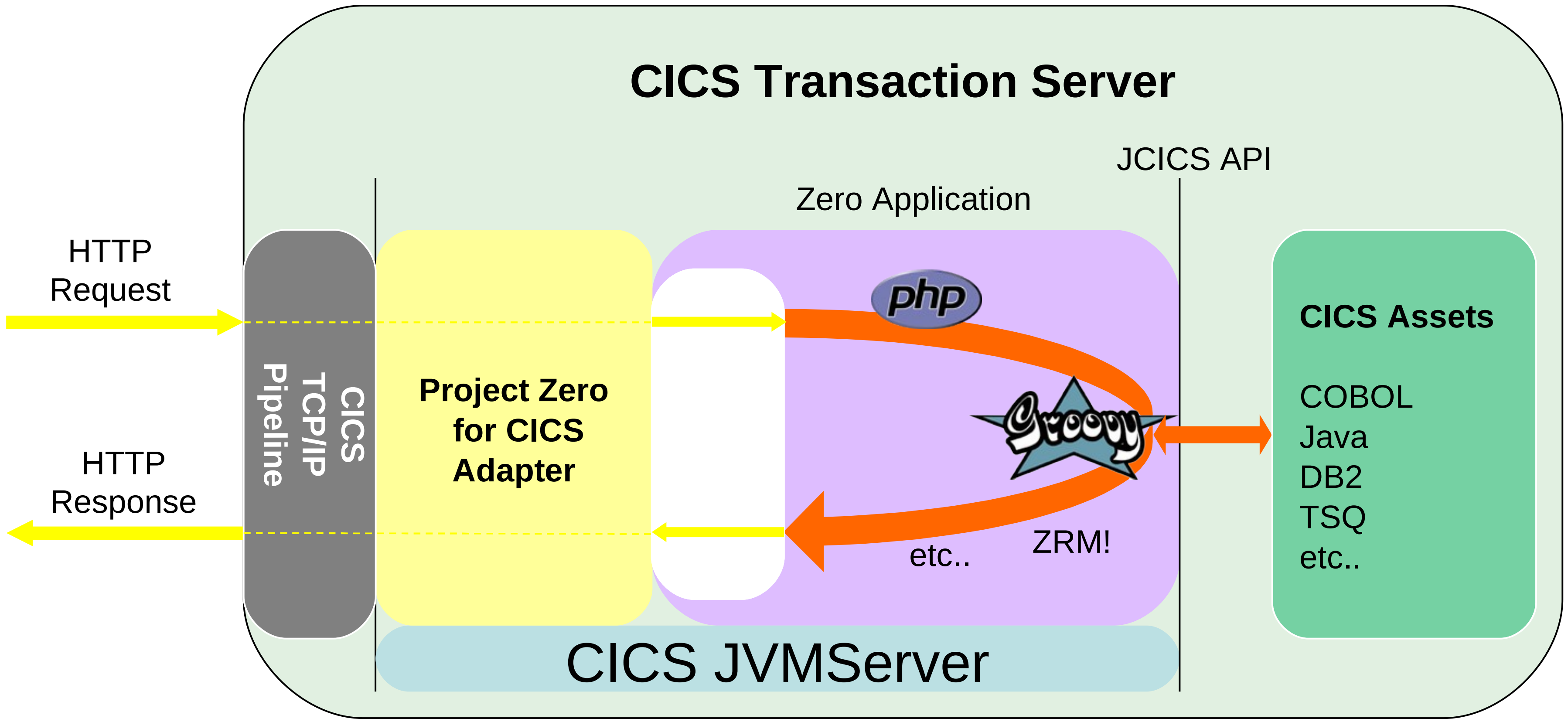
Resource	GET	PUT	POST	DELETE
Collection	List	putCollection	create	deleteCollection
Member	Retrieve	update	postMember	delete

Project Zero Architecture

Project Zero today.

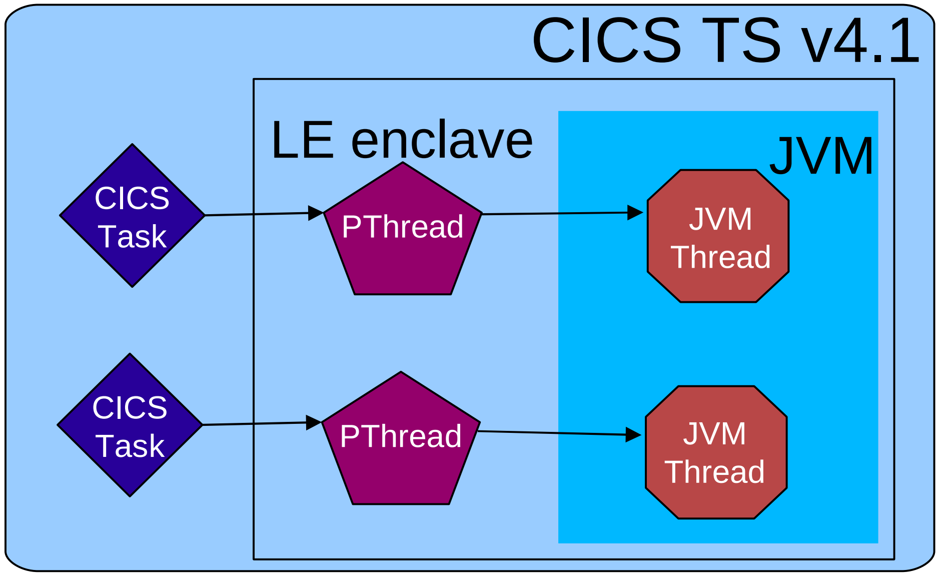


Project Zero on CICS Architecture



CICS TS v4 JVMServer Architecture

- New CICS OTE TCB mode
  - Known as T8 TCBs
  - Both a CICS TCB and an LE "*pthread*".
- JVM allows a pthread to be attached to an existing JVM
  - Can attach multiple pthread/T8/CICS tasks to the JVM at the same time.
- Single JVM can serve many requests in parallel.
  - Saves memory
  - Achieving **hundreds** of concurrent Java tasks in a region



Going it alone -
DataMapping and Transports

Going it alone – Data Mapping - Provider

- For Provider this means writing your own Application Handler but still using the CICS supplied SOAP Handler
  - DFHWS-BODY & DFHWS-XMLNS are the key containers
  - In CICS TS 4.1 the XMLTRANSFORM API (and resource) can be used to allow CICS to do some of the work for you and only do custom processing where required

- For the very brave,
  - You can use the Vendor Bindfile format to create bindfiles (and thus Web Services) that call programs of your choice for encode/decode.
  - This is how RD/z creates compiled converters

Going it alone – Data Mapping - Requester

For Requester this means writing your own program to replace the EXEC CICS Invoke Service command. Again still using the CICS supplied SOAP Handler.

- Containers and options are the same as for the Provider case

You can use a LINK to program DFHPIRT to start a requester pipeline from your replacement code

- DFHWS-PIPELINE has the pipeline name in it

If you are going the vendor bindfile route then you don't need a replacement and can still use 'Invoke Service'

Going it alone – User Transports

Why?

- CICS default pipeline transport behaviour doesn't meet your needs...
- Custom transport security processing

But...

- We ideally don't want you to have to do this!
- So please raise requirements

Requester would likely be more common than Provider

- Transport Handlers have access to the transport in Provider
- In Requester the transport is opened after the the transport handlers are run...

Going it alone – User Transports - Requester

How?

- Create a handler that sits in the pipeline at the end of the service handler list (if it is later CICS will have validated the uri before you are called)
- Create a uri scheme that describes your transport and use the URI container (i.e. cics:// was one we created for the cics transport) or use an existing one
- Check the uri in your handler and eitehr passthru to CICS transport processing or handle it yourself and 'reply early'
  - Either create DFHREQUEST for pass though or
  - Create DFHRESPONSE for reply early

Going it alone – User Transports - Provider

How?

- From your transport listener / message source use the CICS internal transport to start the required provider pipeline
  - cics://service/myService?TargetService=*theuripath*

- The Requester pipeline you use to do this can be 'empty'
  - No handlers, so it just starts the real provider pipeline

- The transport listener needs to build a channel (DFHREQUEST etc) that can be passed in and extract the response from the one passed back (before making the reply in a transport specific manner).

Summary

- Web Services and SOA in CICS TS v4
  - A broader view – **it's more than WS-* now**
- The Pipeline
  - Pipeline Overview
    - **It's very flexible and CICS is using it more and more**
  - The Configuration Files
    - **Not too scary**
  - Custom Handlers
    - **If you <u>really</u> need to**
- Web Services and Exploiters
  - WLM, DataPower, CICS Internal Transports
  - **Dynamic Scripting FeaturePack**
  - New options with DB2 PureXML
  - Going it alone – **if you <u>really</u>, <u>really</u> need to**

SHARE
Technology • Connections • Results

Any Questions?

SHARE in Boston

Thank You for your Attention

Please fill out a session evaluation form