Closing the Gap
between products
and where they need to go

Introduction to Creating DLAs

with Tivoli Directory Integrator (TDI),
a multidirectional integration service

BlueGlue

"The issue is not about how much is addressed with out-of-the-box integrations,
but how much time is spent on the rest"

- IBM Product Manager

Acknowledgement: the Tivoli Enablement Team

- Mel, Mads and the gang made this event possible. Thanks!
- You can get the updated slides here:

  ...the step-by-step guide here:

  ...and my finished Config here:

  Download now to help you work through the exercises, but please pay attention!

  Ask questions! And tell me to repeat myself more slowly :)

Agenda

- Short introductions: you, us and tdi
- Setting up your TDI environment
- TDI 101 – Hello, World (and Hello, Debugger)
- My First DLA
  - Reading Input Data
  - Creating an IdML Discovery Book
  - Adding ConfigurationItems (CIs)
  - Validating your work
  - Adding Relationships between CIs (like "InstalledOn")
  - Transferring your output to the TADDM Server for import*
  - Importing the data into TADDM*

* These points will be covered in theory, and may be carried out by those with adequate connectivity to a TADDM server.

Agenda - continued

- If all goes well and we get through the agenda early, we can look at real-world scenarios attendees may be facing.

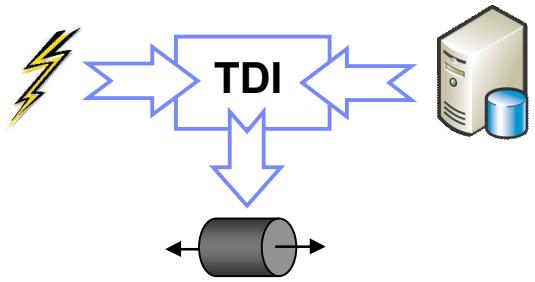
  Bring along text files (any format) with data that you want to look at. You can even use this yourself while following the online course.

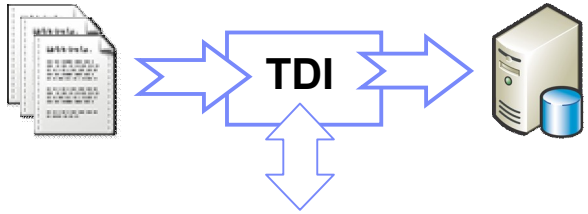
Agenda

- Short introductions: you, us and tdi
- Setting up your TDI environment
- TDI 101 – Hello, World (and Hello, Debugger)
- My First DLA
  - Reading Input Data
  - Creating an IdML Discovery Book
  - Adding ConfigurationItems (CIs)
  - Validating your work
  - Adding Relationships between CIs (like "InstalledOn")
  - Transferring your output to the TADDM Server for import*
  - Importing the data into TADDM*

* These points will be covered in theory, and may be carried out by those with adequate connectivity to a TADDM server.

TDI is a multi-purpose, multi-directional, integration, synchronization, and transformation service

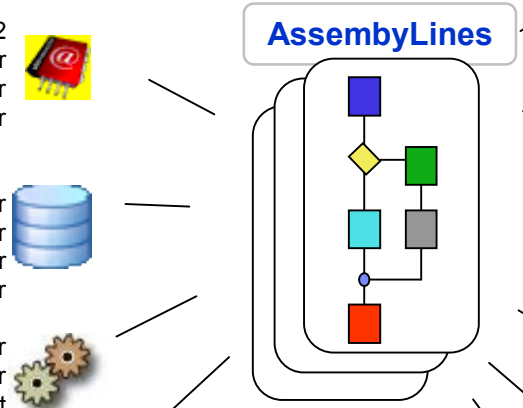




Delta service detects changes in originating source

- A unique approach to data integration that equally well synchronizes and transforms data between widely different systems (such as files, databases, directories, message queues, web services, and many more), as responding to events (such as email, HTTP, web services, SNMP, TCP, JMX, and more)
- The combination of the above capabilities allows TDI to be applied to a broad set of usage scenarios
- TDI integrates practically anything, and - despite its name - is not in any way limited to directories. It's a truly generic data integration tool that's suitable for a wide range of problems that usually require custom coding and significantly more resources to address with traditional integration tools
- TDI is a lightweight Java based application (no app server necessary) that consists of a lightweight server run-time environment and a graphical tool to build, test and maintain the rules that the server executes.
- Supported platforms: Windows, Linux, AIX, Sun, HP, i5/OS and zOS.

TDI 6.1.1 Server



Axis Easy Web Service Server Connector
Axis Easy Web Service Invoke
Axis Java-to-Soap
Invoke Soap Web Service
Axis Soap-to-Java

LDAP Connector
LDAP Server Connector
Tivoli Access Manager Connector
Windows Users and Groups Connector

Active Directory Changelog Connector v2
IBM Directory Server Changelog Connector
Netscape/iPlanet Changelog Connector
zOS LDAP Changelog Connector

JDBC Connector
Properties Connector
SystemStore Connector
RDBMS Changelog Connector

AssemblyLine Connector
Server Notifications Connector
AssemblyLine Function Component

Domino Change Detection Connector
Domino Users Connector
Lotus Notes Connector

Exchange Changelog Connector
Mailbox Connector
SendEMail Function Component

TIM DSMLv2 Connector
DSMLv2 SOAP Connector
DSML v2 SOAP Server Connector
Generic JNDI Connector
ITIM Agent Connector

EMF SDOToXML Function Component
EMF XMLToSDO Function Component

Timer Connector

Generic Log Adapter Connector
RAC Connector
Entry to CommonBaseEvent Function

JMX Connector
SNMP Connector
SNMP Server Connector
TCP Connector
TCP Server Connector

Remedy/Peregrine /CCMDB tickets
Maximo MEA
Many custom Components on
OPAL, tdi-users.org or on request

PeopleSoft Connector
Siebel Connector
SAP ALE IDoc Connector
SAP R/3 Business Object Repository
SAP R/3 User Registry
SAP R/3 RFC Component

Script Connector
Generic Java Method
Parser FC

Remote secure command Line
z/OS TSO/E Command Line
Command Line Connector

MemQueue Connector
Memory Stream Connector

HTTP/REST Connector
File System Connector
FTP Client Connector
URL Connector
HTTP Client
HTTP Server Connector

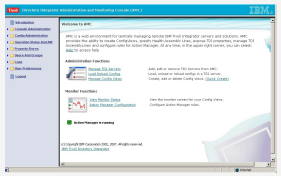
IBM MQ Series Connector
JMS Pub/Sub Connector
MQe Password Store Connector

CCMDB
Netcool
RSS
Google

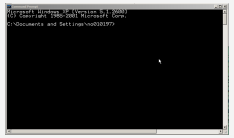
Parsers
CSV
DSML v1 & v2
Fixed Record
HTTP
LDIF
Line reader/writer
SOAP
Script
XML
XML Sax
XSL

TDI architecture

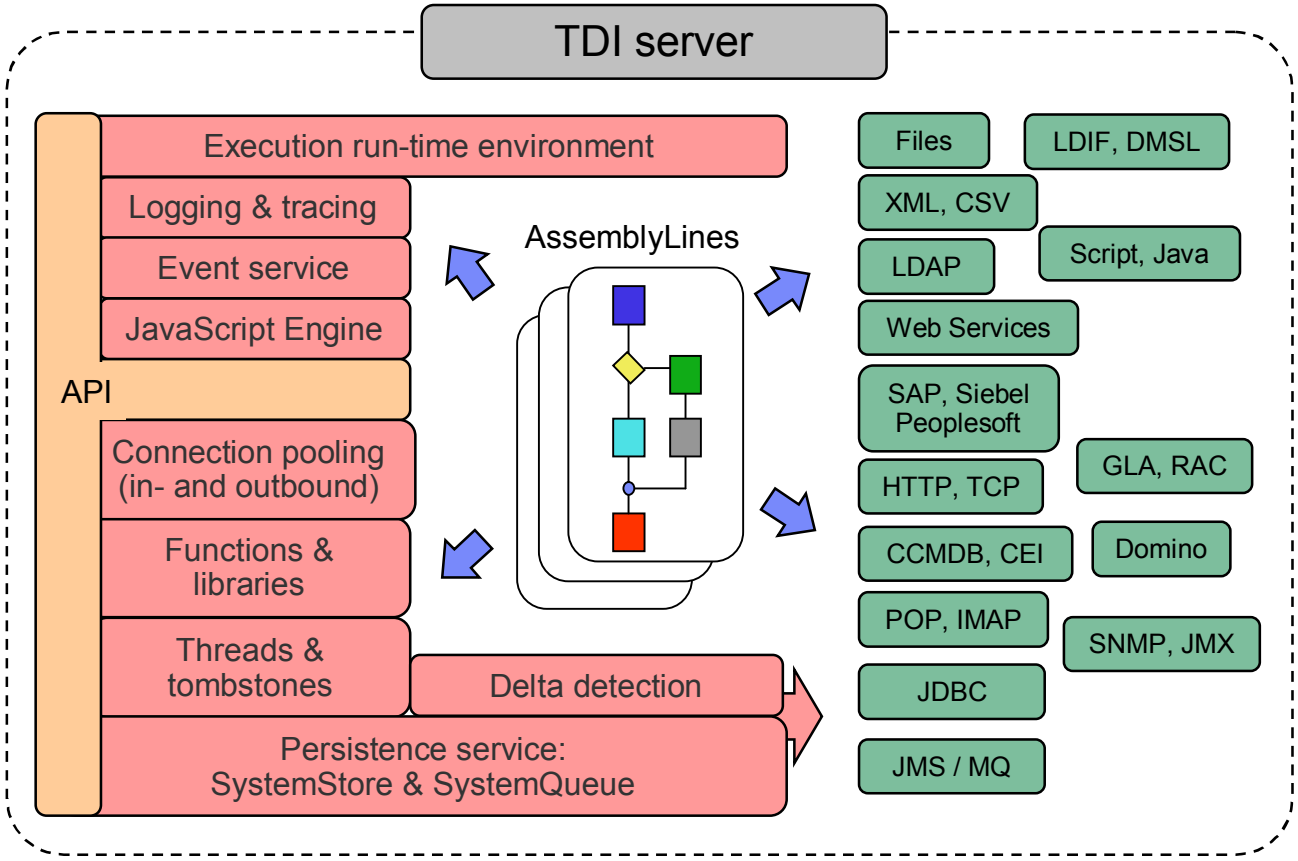
Integrated development environment



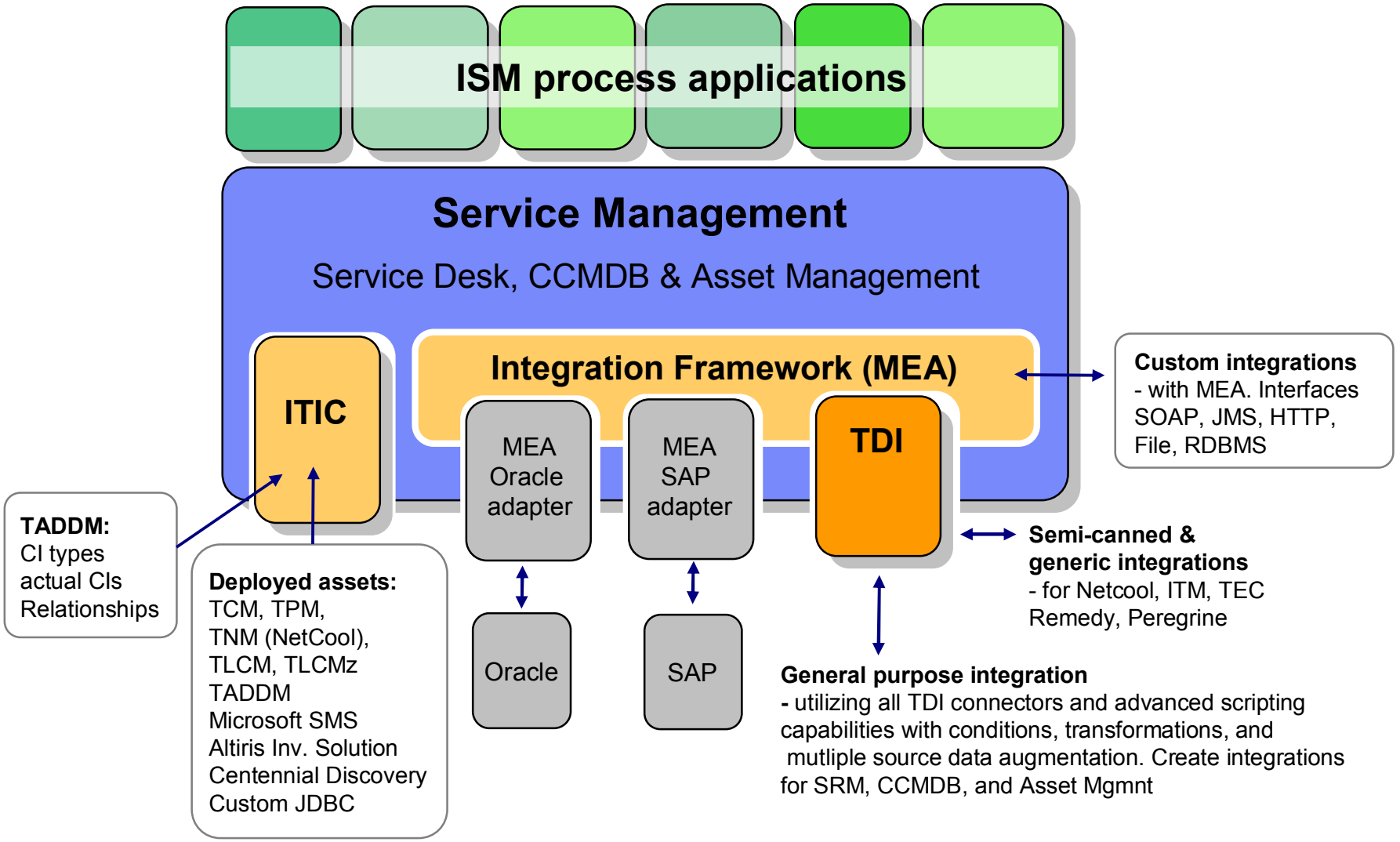
Web administration and monitoring



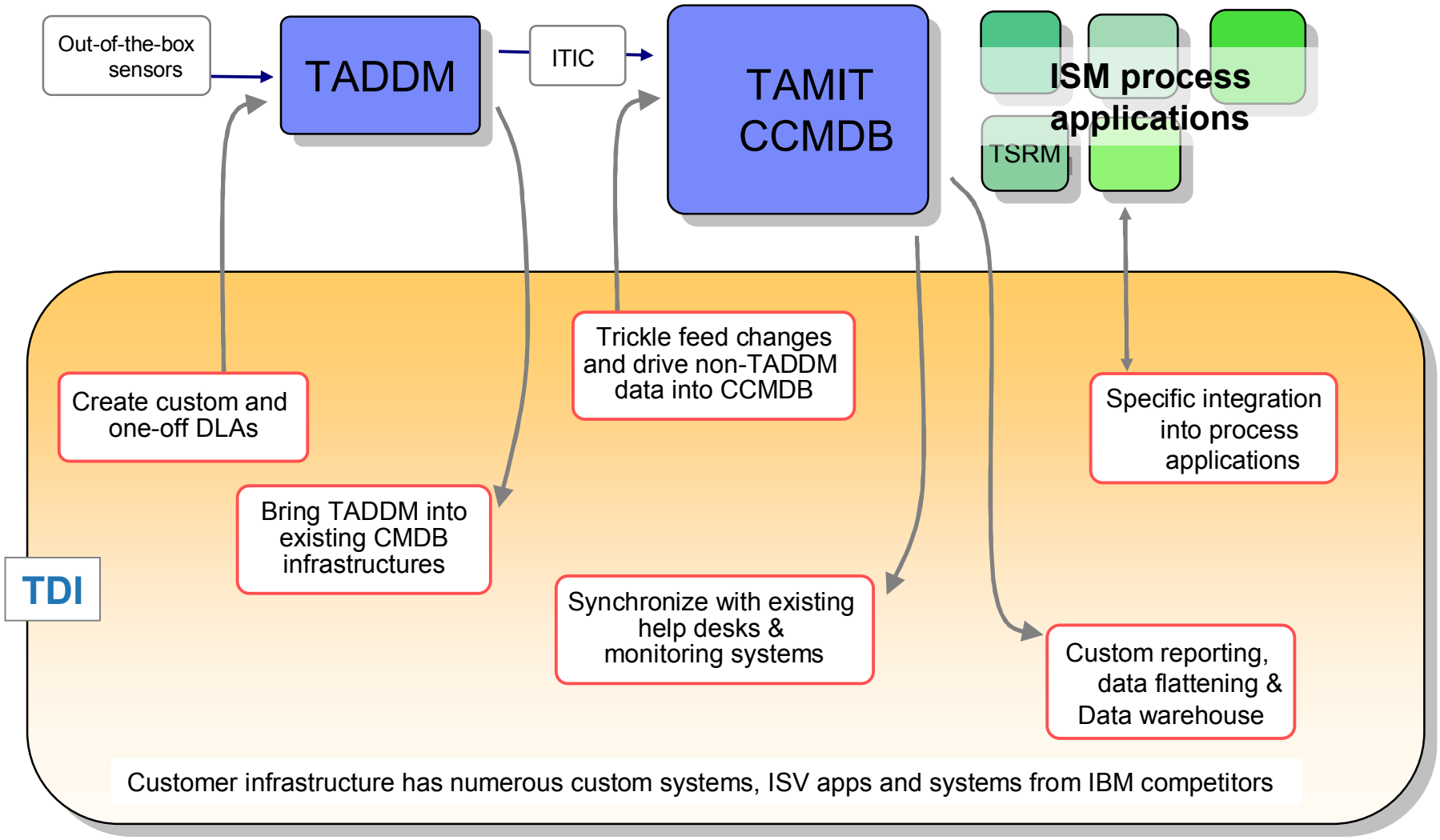
Commandline interface



TDI in the ISM integration architecture



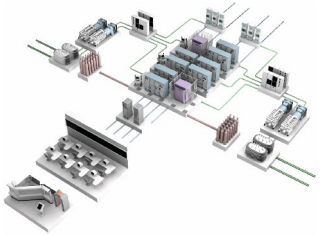
One product integration across the ISM portfolio



Available Stand-alone or Bundled

Websphere RFID Information Center

Access Manager

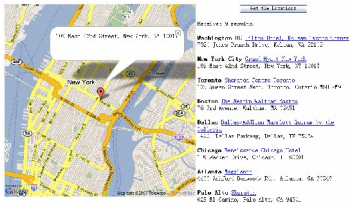


CCMDB asset discovery

Service Desk

Lotus Domino

Lotus Connections



IM MashupHub

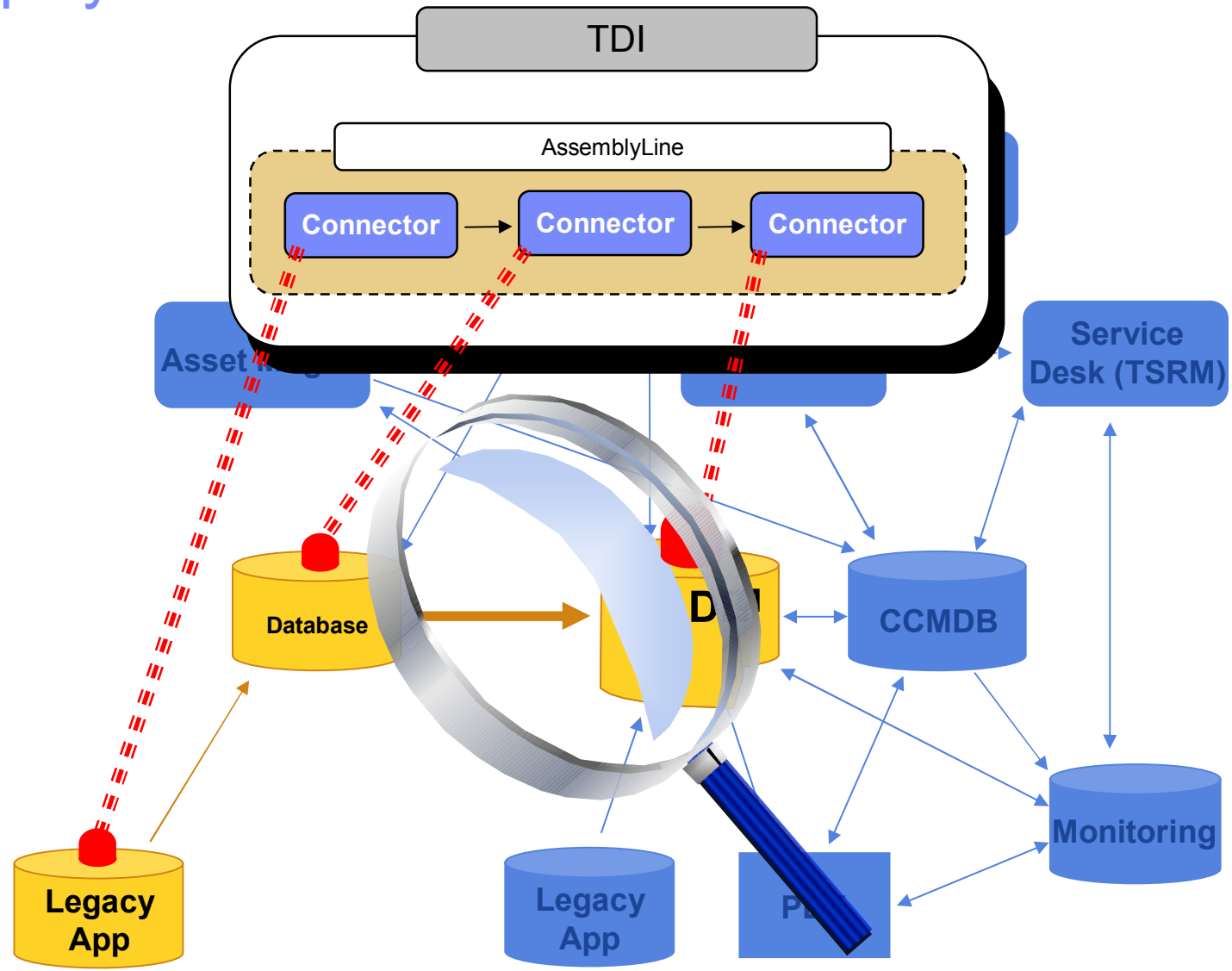
Identity Manager

Integration Army Knife

Designed for the front lines:
PoCs, PoTs, migrations,
conversions/win-backs,
synchronizations,
...

Trench Tool and enterprise-strength middleware

Simplify and Solve



Agenda

- Short introductions: you, us and tdi
- Setting up your TDI environment
- TDI 101 – Hello, World (and Hello, Debugger)
- My First DLA
  - Reading Input Data
  - Creating an IdML Discovery Book
  - Adding ConfigurationItems (CIs)
  - Validating your work
  - Adding Relationships between CIs (like "InstalledOn")
  - Transferring your output to the TADDM Server for import*
  - Importing the data into TADDM*

* These points will be covered in theory, and may be carried out by those with adequate connectivity to a TADDM server.

Setting Up TDI: Your mission should decide to accept it...

- Get TDI up and running at the latest patch level
- Explore your setup: Solution Directory and properties
- Extend TDI by adding new IdML components
- Prepare the test input data for this class

Setting Up TDI: No TDI installed yet?

- **download and unzip this file to** C:\Program Files\IBM\TDI:
  http://www.tdi-users.org/twiki/pub/Integrator/GlossaryEntry/Full_TDI6.1.1FP4_Just_unizp_to_ProgramFiles_IBM_TDI.zip
  Yes, you have to create these directories if they don't exist
  Your TDI install directory is then C:\Program Files\IBM\TDI\V6.1.1
- **download and unzip this file to** C:\
  http://www.tdi-users.org/twiki/pub/Integrator/GlossaryEntry/TDI_SolDir_Unzip_to_root_of_C_drive.zip
  Your Solution Directory is then C:\TDI Solution Directory

Now you should be able to launch the TDI Dev Tool, called the *"Config Editor"*, or just *"CE"* short:

C:\Program Files\IBM\TDI\V6.1.1\ibmditk.bat

Setting Up TDI: Already Have TDI Installed?

- **If you still need to apply FixPack #4 (recommended!!) download and unzip this file to** C:\Program Files\IBM\TDI\V6.1.1*:
  http://www.tdi-users.org/twiki/pub/Integrator/GlossaryEntry/TDI6.1.1_FP4_Unzip_To_Install_Folder.zip
  *...or whatever folder you installed the TDI binaries to.
- **download and unzip this file wherever you choose**
  http://www.tdi-users.org/twiki/pub/Integrator/GlossaryEntry/TDI_SolDir_Unzip_to_root_of_C_drive.zip
  It contains the Custom Jar files needed for this exercise (Custom Jars sub-directory), as well as the _My First TDI DLA folder and its contents.

  You also have to make the custom .jar files available to TDI. The technique for doing this is described here:
  http://tdiingoutloud.blogspot.com/2008/11/new-component-and-library-jar-files.html

  Now you should be able to launch the TDI Dev Tool, called the *"Config Editor"*, or just *"CE"* short:

  C:\Program Files\IBM\TDI\V6.1.1\ibmditk.bat

Setting Up TDI: You And Your Solution Directory

- **Installation Directory**
  Where the TDI binaries are found.
  Default on Windows is C:\Program Files\IBM\TDI\V6.1.1
  The global.properties file is found in the etc sub-folder.
  Two important batch-files/scripts: ibmditk and ibmdisrv
- **Solution Directory**
  Where your TDI project files are kept. *Back this one up!*
  TDI project files are called *Configs* and are XML documents.
  Default on Windows is C:\My Documents\<userid>\TDI
  The solution.properties file is found in this folder, and it overrides settings in global.properties.

The Solution Directory is the *root* for all *relative file paths* in TDI

Setting Up TDI: Adding the IdML components

- Either copy new .jar files to the jars folder in the TDI installation directory (or some sub-folder therein)
- Or, *preferrably*, edit the com.ibm.di.loader.userjars property to point to .jar and .zip files, or folders that contain these, or any combination of the above (;-separated on Windows, : on unix).
  - Edit solution.properties if you have a Solution Directory
  - Otherwise change global.properties in <TDI InstallDir>/etc

e.g. com.ibm.di.loader.userjars=C:\TDI Solution Directory\Custom Jars

Setting Up TDI: Start your engines...

- Launch the TDI Config Editor
- Select File > New
- Open the folder called "_My First TDI DLA"
- Create (or open) TDI Config called "1_HelloWorld.xml"
- Right-click Connectors folder and choose *New Connector...*
- Make sure these components appear in the list (at bottom): idml.IDMLConfigurationItem and idml.IDMLReln
- Right-click on Functions and check for these: idml.OpenIDML and idml.CloseIDML

Setting Up TDI: Download the test data

- Open this document and copy contents to copy buffer
  http://docs.google.com/Doc?id=dsrxm8p_15w985sfj&invite=cgpnf4m
- Create a new text file in <TDI SolDir>/_My First TDI DLA
  Call it MachineAndOS.csv and paste in the data you copied.
  Close and save this file.

Setting Up TDI: Lessons Learned?

- Data processed by the AL is carried in the Work Entry.
- Attribute Maps "gate" data into and out from the AL.
  Input Maps move data from *conn* to *work*
  Output Maps move data from *work* to *conn*
- AssemblyLines handle one Entry per cycle.
- TDI Components are interchangeable.
- Feeds is a built-in Loop that drives the Flow section.
  The AL will cycle as long as Feeds produces new Entries.
- The Debugger is Your Friend :)
  Don't be afraid to *touch* the data.

Agenda

- Short introductions: you, us and tdi
- Setting up your TDI environment
- TDI 101 – Hello, World (and Hello, Debugger)
- My First DLA
  - Reading Input Data
  - Creating an IdML Discovery Book
  - Adding ConfigurationItems (CIs)
  - Validating your work
  - Adding Relationships between CIs (like "InstalledOn")
  - Transferring your output to the TADDM Server for import*
  - Importing the data into TADDM*

* These points will be covered in theory, and may be carried out by those with adequate connectivity to a TADDM server.

TDI 101: Hello, World

- Create a new AssemblyLine called "1_HelloWorld"
- In the AL Hooks tab, enter code in "On Start Of Cycle" Hook
  task.logmsg("Hello, World");
- Press the **Run** button to test

- Add a Script Component named "SayHello" with this code:
  task.logmsg(thisConnector.getName() + " says: Hello, World");
- Press **Run** again. *Did you get two messages this time?*

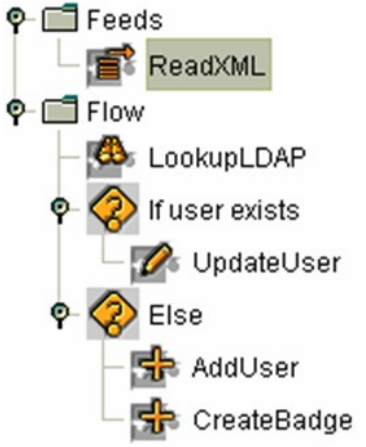
- Disable the "On Start Of Cycle" Hook and try again.
- Now set the Run mode to **Step (Paused)** and *step* through your AL.

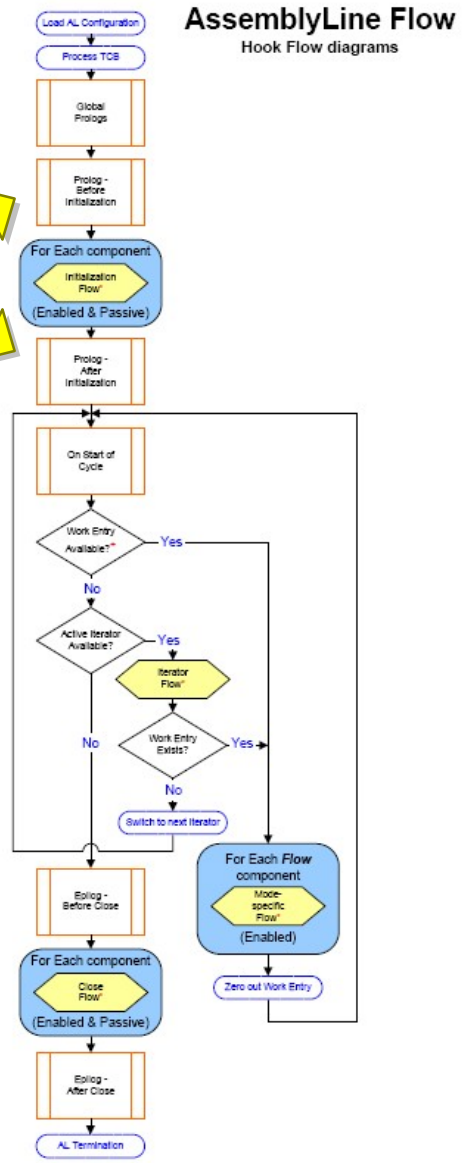
TDI 101: AssemblyLine Flow

There is a built-in *microflow* that drives the AssemblyLine. This default AL behavior can be augmented or even overridden as needed by writing snippets of JavaScript into the appropriate *Hooks*.

Hook coding is how errors are handled in TDI.

In addition to this built-in pipeline, you can visually implement business and data processing logic in your AssemblyLines using the development tool: the CE.



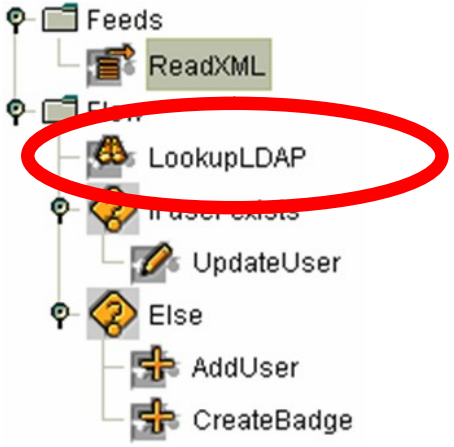


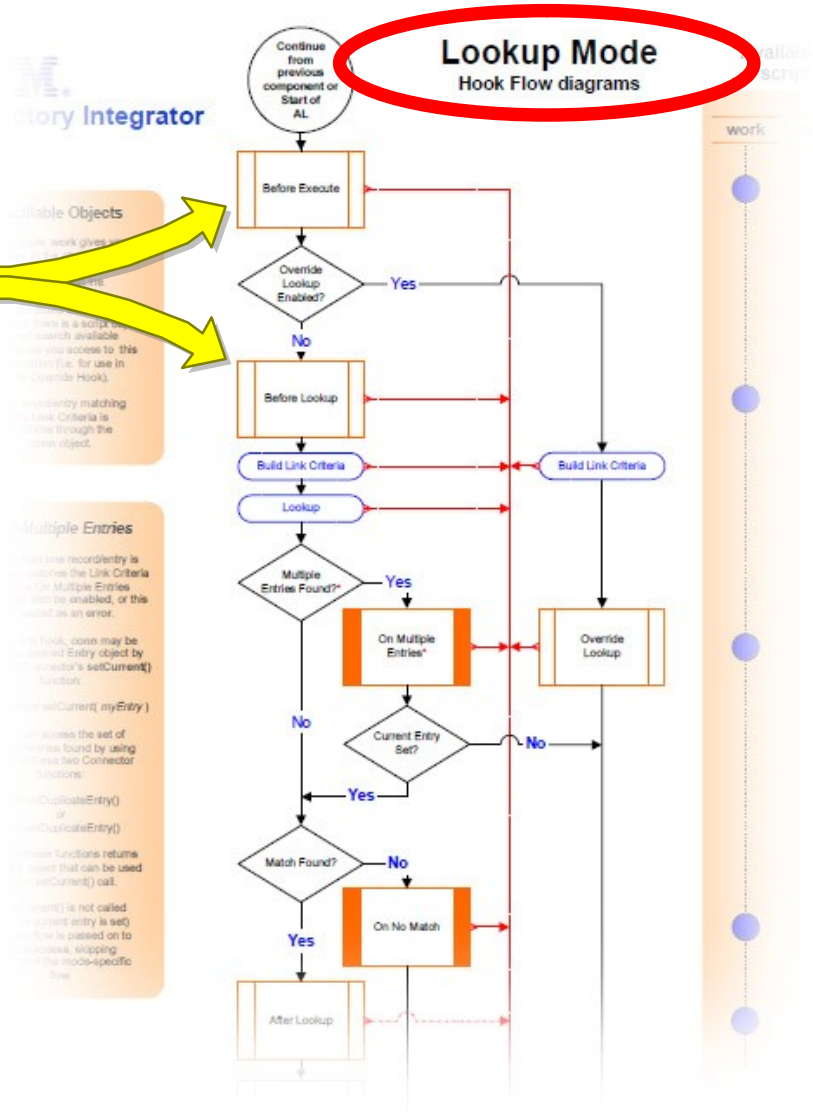
TDI 101: Component Flow

There is also a *microflow* associated with each type of component (and each Connector mode).

Just like the AL flow, these are also configurable and can be enhanced or overridden by scripting Hooks.

Error handling can be done by scripting component Error Hooks, those of the AL itself, or both.





TDI 101: The AL Debugger

- Step through AssemblyLine execution
  - from component to component
  - from Hook to Hook in the built-in logic flows
- Set breakpoints to pause execution as needed
- Execute JavaScript interactively inside the running AL
  - manipulate data values and script variables
  - call external systems and libraries
- Explore the integration problem piece by piece

The "Hello, World" exercise requires no actual development.

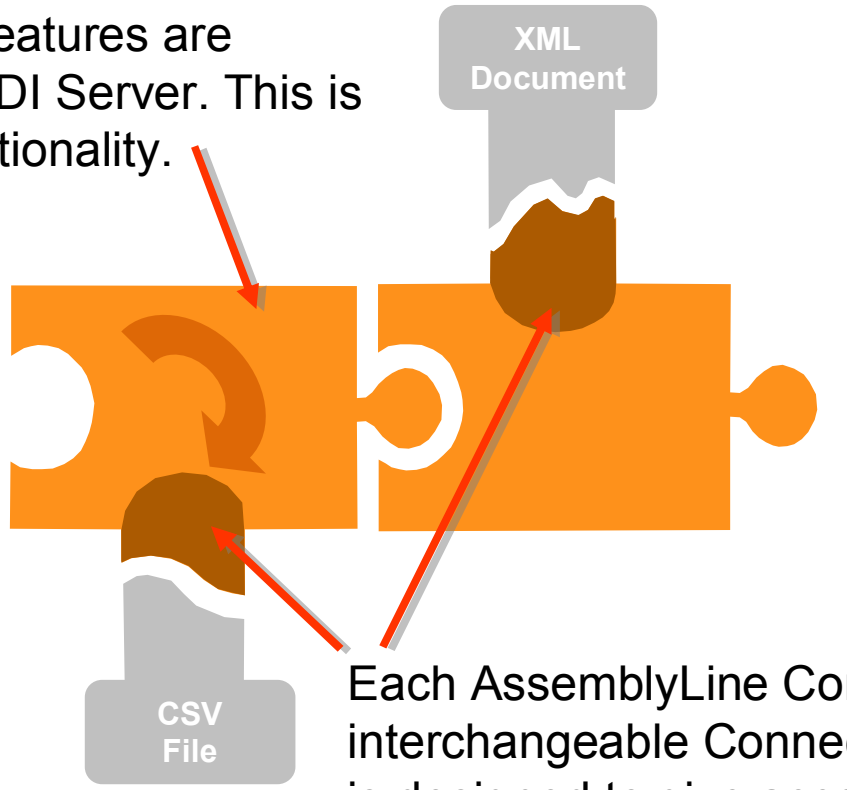
TDI 101: Simple AL – Reading and Writing Data

- New AL: "2_CSV2XML"
- Add Connector "ReadCSV" to Feeds section
  - FileSystem Connector (Iterator mode)
    File Path: **_My First TDI DLA/MachineAndOS.csv**
    Attach the ibmdi.CSV Parser (note: separator may not be ";")
- Add Connector "Output" to Flow section
  - FileSystem Connector (AddOnly mode)
    File Path: **_My First TDI DLA/TestOutput.xml**
    Attach the ibmdi.XML Parser
- Select "Step (Paused)" and press **Run** to step through

TDI 101: AssemblyLine Connector vs. Connector Interface

Most Connector features are provided by the TDI Server. This is called *kernel* functionality.

Kernal functionality includes Attribute Maps, search criteria for Lookup, Delete and Update operations, Hooks, Auto-Reconnect and Change Detection.



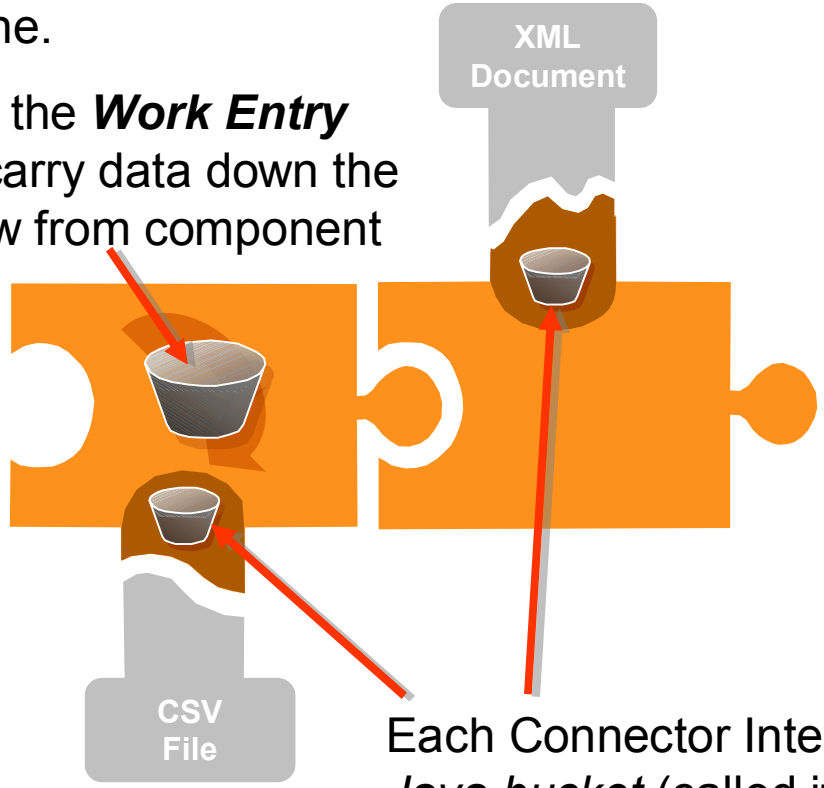
Each AssemblyLine Connector drives an interchangeable Connector Interface (CI) which is designed to give access to some protocol, API, transport or format. This is called *component* functionality.

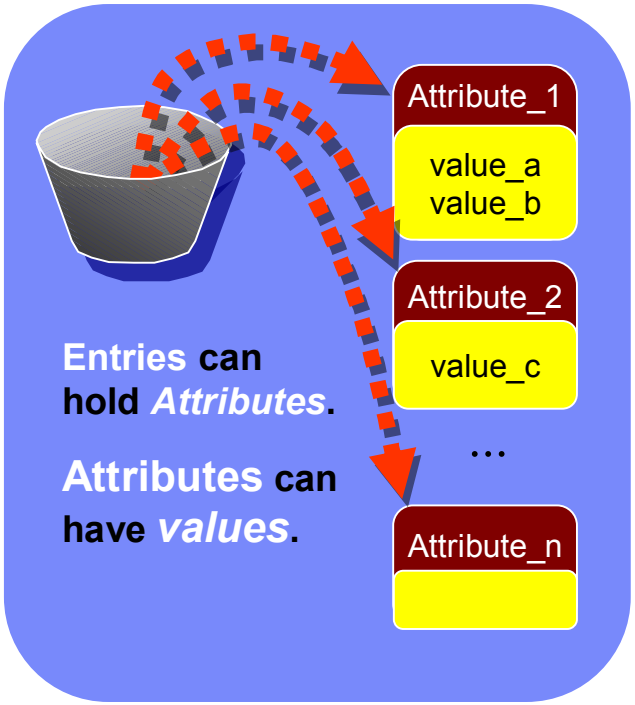
TDI 101: The Entry object - TDI data model

The ***Entry*** object is the "data carrier" in an AssemblyLine.

The main Entry is the ***Work Entry*** which is used to carry data down the AssemblyLine flow from component to component.

The Work Entry is available to your scripts as the pre-registered variable ***work***.

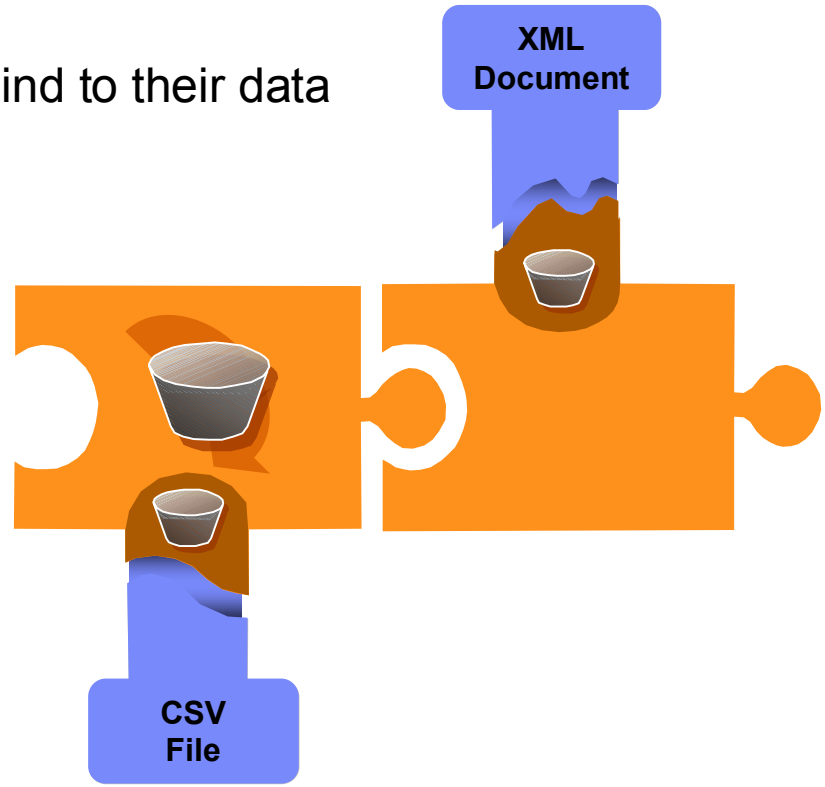




Each Connector Interface has its own local *Java bucket* (called its ***Conn Entry***) which is used as a local cache for reads & writes. This is accessed via the ***conn*** variable.

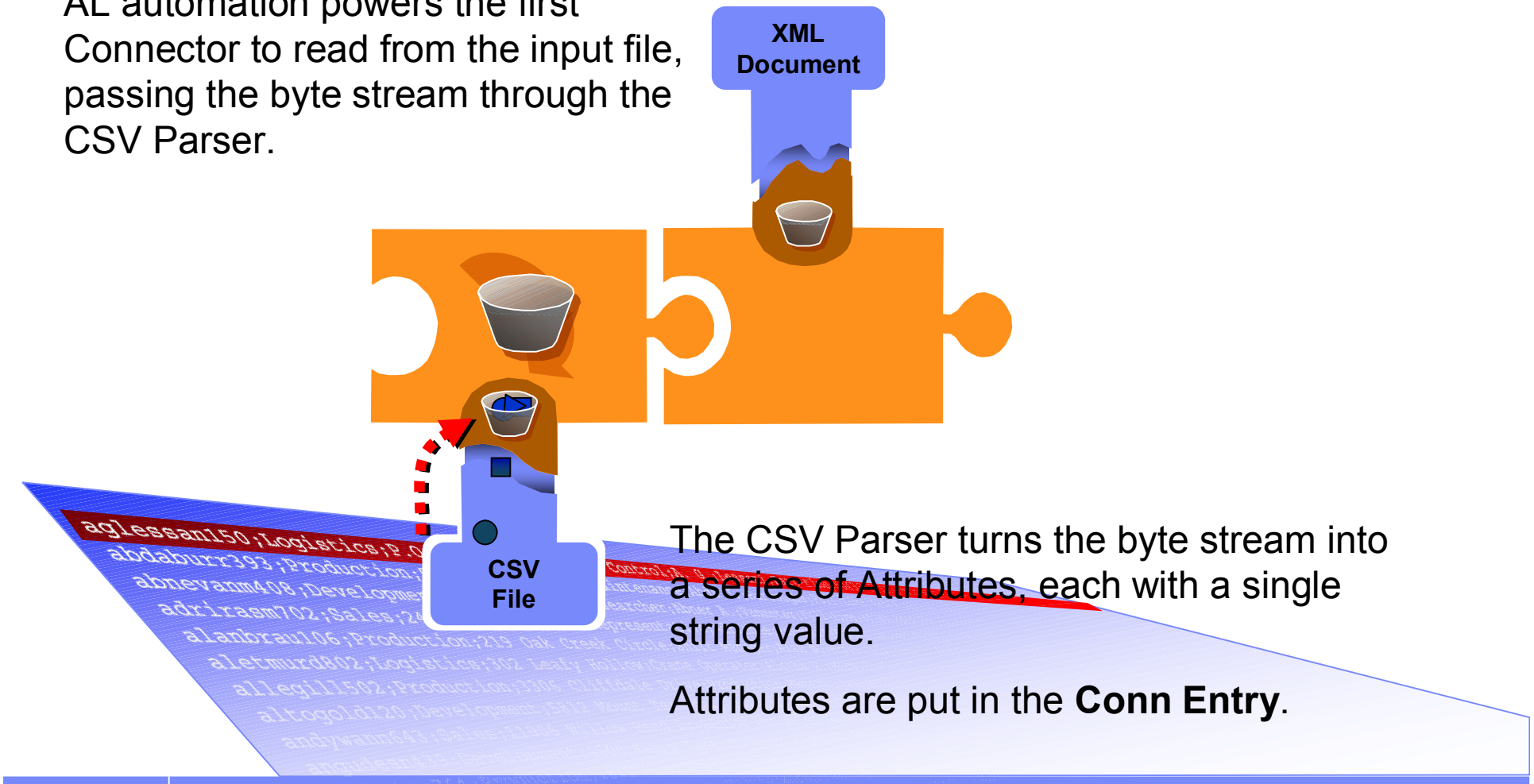
TDI 101: AL Lifecycle - Phase One: Initialization

All Connectors bind to their data sources.



TDI 101: AL Lifecycle - Phase Two: Cycling (Read)

AL automation powers the first Connector to read from the input file, passing the byte stream through the CSV Parser.

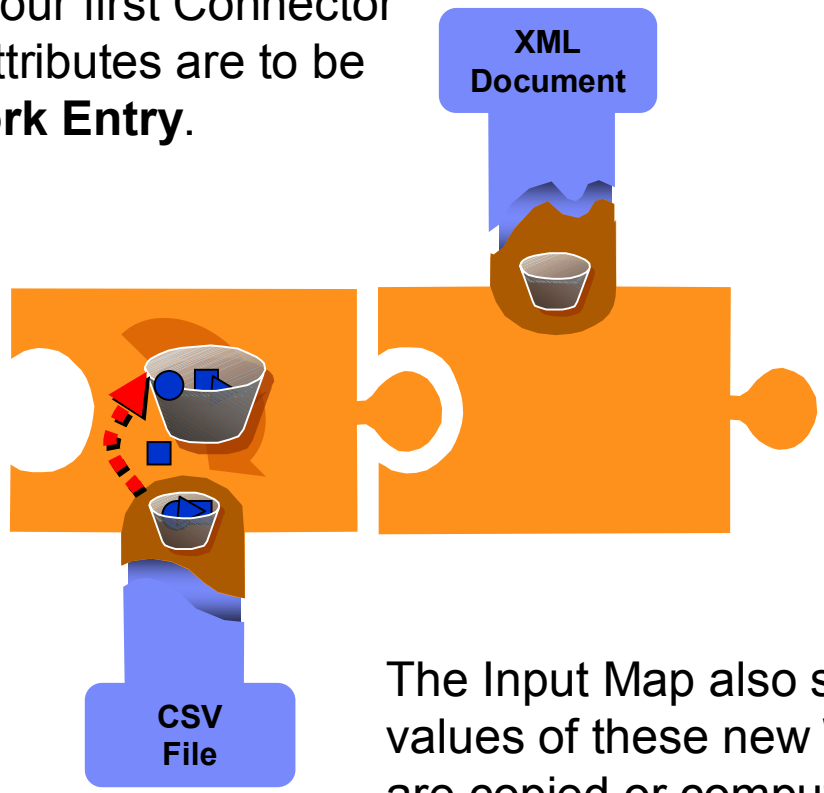


The CSV Parser turns the byte stream into a series of Attributes, each with a single string value.

Attributes are put in the **Conn Entry**.

TDI 101: AL Lifecycle - Phase Two: Cycling (Input Map)

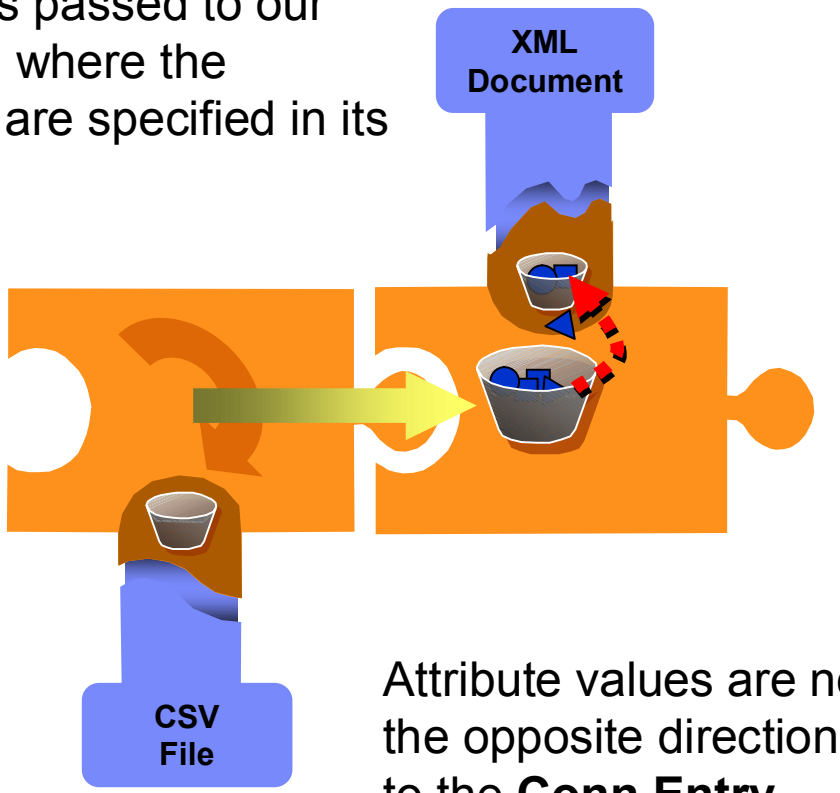
The Input Map of our first Connector specifies which Attributes are to be created in the **Work Entry**.



The Input Map also specifies how the values of these new **Work Entry** Attributes are copied or computed based on those stored in the **Conn Entry**.

TDI 101: AL Lifecycle - Phase Two: Cycling (Output Map)

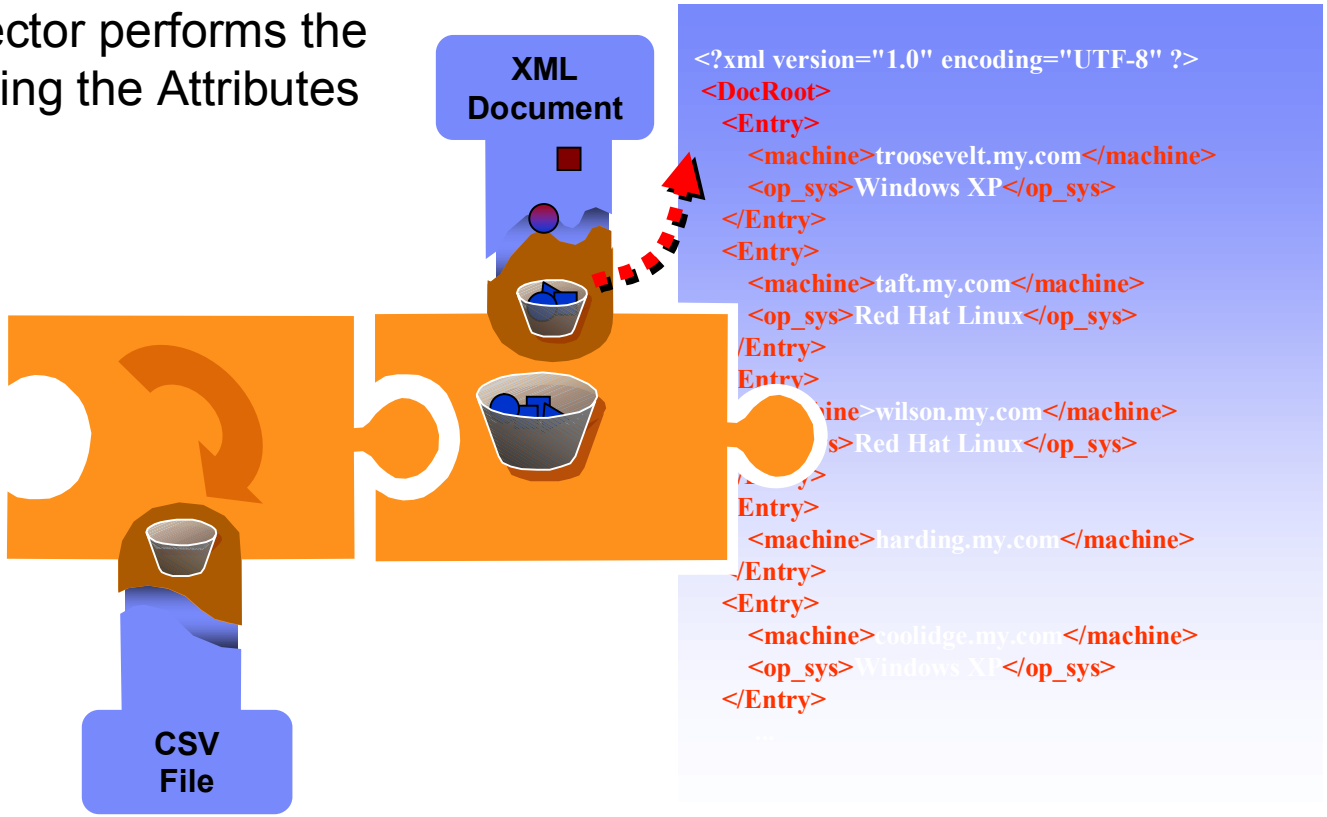
The **Work Entry** is passed to our output Connector, where the Attributes to write are specified in its Output Map.



Attribute values are now copied/computed the opposite direction: from the **Work Entry** to the **Conn Entry**.

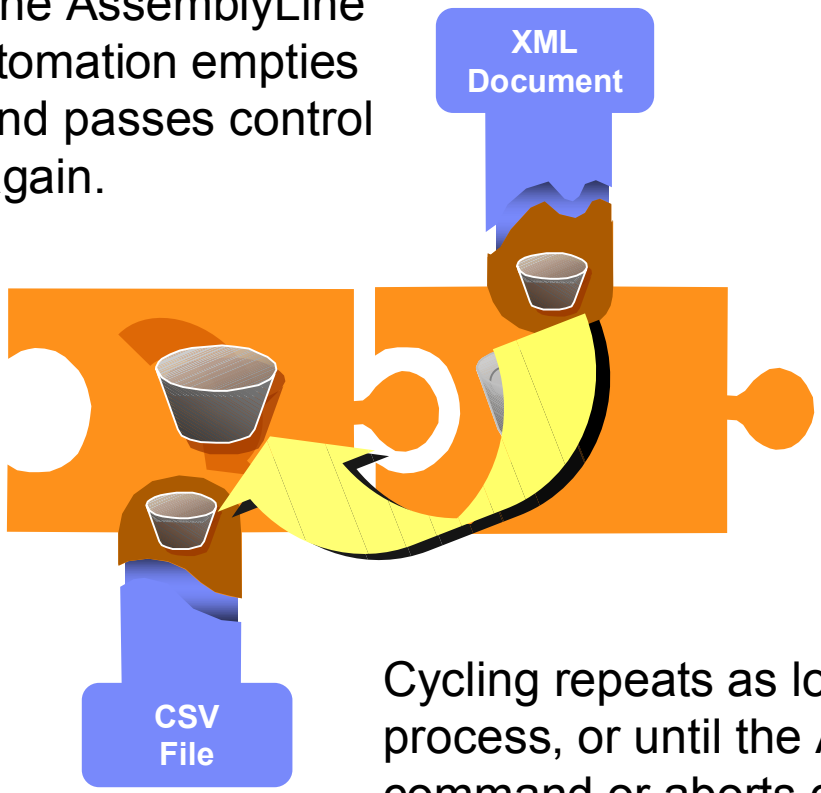
TDI 101: AL Lifecycle - Phase Two: Cycling (Write)

The output Connector performs the write operation using the Attributes in its **Conn Entry**.



TDI 101: AL Lifecycle - Phase Two: Cycling (Repeat...)

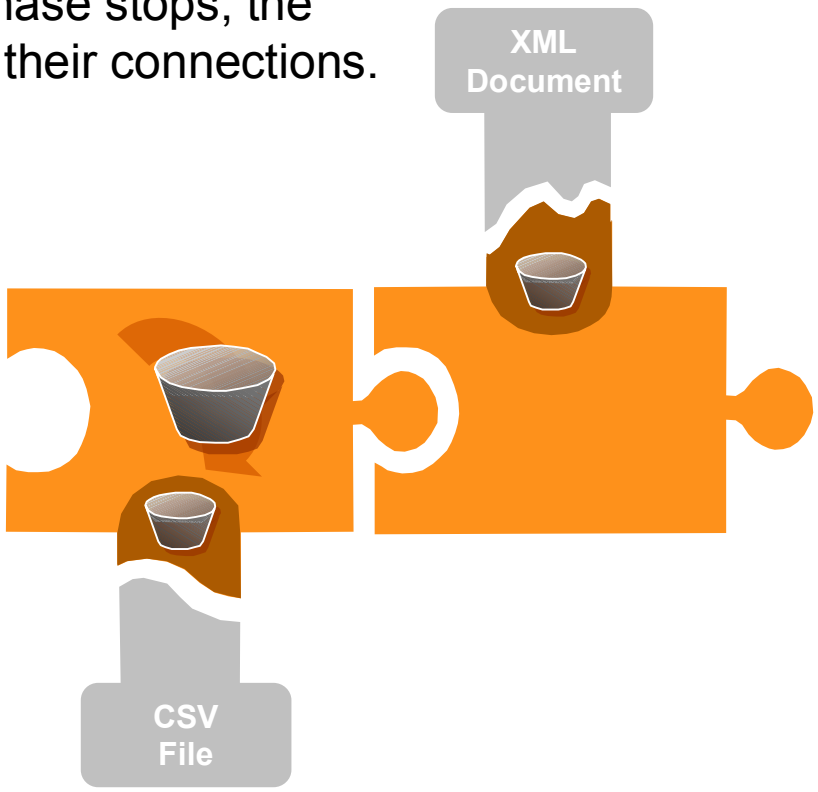
When the end of the AssemblyLine is reached, AL automation empties the **Work Entry** and passes control back to the start again.



Cycling repeats as long as there is data to process, or until the AL is terminated by command or aborts due to unhandled errors.

TDI 101: AL Lifecycle - Phase Three: Shutdown

When the cycle phase stops, the Connectors close their connections.



TDI 101: Clone and Swap Parser

- Clone the "2_CSV2XML" AL and call it "3_CSV2Fixed"
- Edit "Output" Connector in Flow section
  - Swap out XML Parser with ibmdi.Fixed Parser (fixed format)
    Column Description: **op_sys,1,40**
    machine,41,20
- Select "Step (Paused)" and press **Run** to test again

Your AL now converts from CSV to fixed format.

TDI 101: Clone again and Swap Connector

- Clone the "3_CSV2Fixed" AL and call it "4_CSV2DB"
- Change type of "Output" Connector in Flow section
  - Swap out FileSystem with SystemStore Connector
    Key Attribute Name: **machine**
    Table Name: **Machines**
- Select "Step (Paused)" and press **Run** to test again

TDI 101: Project Library

- Copy "ReadCSV" Connector to the Project Library.
  Right-click or use the button 
- Library Components can be reused in multiple ALs.
  - configuration, schema/maps, error handling and business logic.
- Modify Library Components, and ALs inherit these changes.
  - go from *lab* to *live* in a few minutes
- Drag from Library to Resources to keep a personal library.

Inheritance is based on the component name and therefore easily broken!

TDI 101: Lessons Learned?

- TDI components (CIs) are interchangeable.
  Use FileSystem Connector for visual control of output.
- Hooks are used to handle errors and modify default behavior.
- Data is carried in the AL by Entry objects,
  the primary of which is Work Entry.
- Attribute Maps "gate" data into and out from the AL.
  Input Maps move data from *conn* to *work*
  Output Maps move data from *work* to *conn*
- AssemblyLines handle one Entry per cycle.
- Feeds is a built-in Loop that drives the Flow section.
  The AL will cycle as long as Feeds produces new Entries.
- The Debugger is Your Friend :)
  Don't be afraid to *touch* the data.

Agenda

- Short introductions: you, us and tdi
- Setting up your TDI environment
- TDI 101 – Hello, World (and Hello, Debugger)
- My First DLA
  - Reading Input Data
  - Creating an IdML Discovery Book
  - Adding ConfigurationItems (CIs)
  - Validating your work
  - Adding Relationships between CIs (like "InstalledOn")
  - Transferring your output to the TADDM Server for import*
  - Importing the data into TADDM*

* These points will be covered in theory, and may be carried out by those with adequate connectivity to a TADDM server.

My First DLA: Opening the IdML Book

- Create new AL: "MyFirstDLA"
- Add an FC (name it "OpenBook") in the Flow section
  Choose the idml.OpenIDML Function component:
  Configuration parameters:

  Application Code: **App 1.0**
  Directory Name: **C:\temp** *(make sure you have this folder!)*
  Book Name: **MyBook**
  Manufacturer Name: **IBM**
  Product Name: **MyProduct**
  Hostname: **host.ibm.com**

- Press **Run** button to test.

My First DLA: Creating your own Iterator Loop

- Add a Connector Loop named "FOR EACH machine read"
  - Add a Loop component to the Flow section.
  - Keep the default type: *Connector Loop.*
  - Drag your ReadCSV Library Connector onto Inherit From button Leave in Iterator mode, and Initialize And Select option.
  - Map in all Attributes in the Input Map

You have now added your own Feeds-like Iterator loop.

My First DLA: Machine ConfigurationItem (CI)

- Under the Loop add the idml.IDMLConfigurationItem FC.
  - Call it "AddMachineCI"
    ClassType: **cdm:sys:ComputerSystem**
    Book Name: **MyBook**
  - Drag the Attribute "machine" from Work Entry into Output Map
    Rename to "id".
  - Drag it in again, this time renaming it to "cdm:Signature"
  - Once more, this time renaming to "cdm:Fqdn".
- Enable the Validate option in the "Open Book" FC.
- Run and view the validation report.

My First DLA: OS Configuration Item (CI)

- Add another idml.IDMLConfigurationItem Connector.
  - Call it "AddOS"
    ClassType: **cdm:sys:OperatingSystem**
    Book Name: **MyBook**
  - Drag the Attribute "machine" from Work Entry into Output Map
    Rename to "id", change to Expression map and enter this:
    {work.machine}_os
  - Drag "op_sys" in from Work Entry, rename to "cdm:OSName"

My First DLA: Relationship (OS *installed-on* machine)

- Add the idml.IDMLReln Connector.
  - Call it "AddRelationship"
    Relationship Class Type: **cdm:installedOn**
    Book Name: **MyBook**
  - Drag the Attribute "machine" from Work Entry into Output Map
    Rename to "target".
  - Drag "machine" once more from Work Entry into Output Map
    Rename to "source", change to Expression map and enter this:
    {work.machine}_os
- Select **Step (Paused)** mode and start the Debugger.

My First DLA: Transferring IdML Book to TADDM

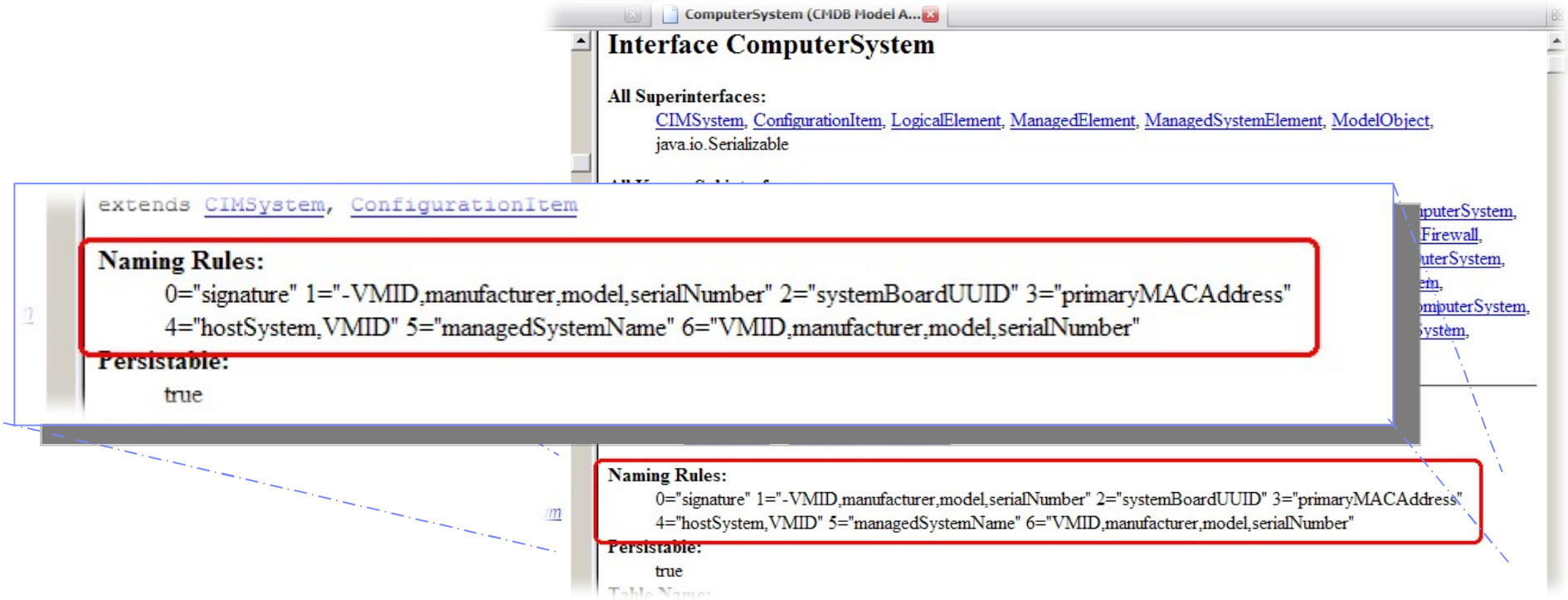
- Close the Book with the idml.IDMLCloseBook FC.
- Add Secure File Transfer FC:
  http://www-01.ibm.com/software/brandcatalog/portal/opal/details?catalog.label=1TW10DI0C
- Configure to send IdML Discovery Book to TADDM machine

My First DLA: Loading the IdML Book into TADDM

- Add Remote Commandline FC:
- Set up command to load IdML file into TADDM
  e.g. c:\ibm\cmdb\dist\bin\loadidml.bat -f c:\ibm\cmdb\dla\

My First DLA: Common Data Model

- JavaDocs here:
  http://www.tdi-users.org/twiki/pub/Integrator/GlossaryEntry/model-javadoc.tar.gz
- Defines the various CDM CIs and Relationships including their *naming rules*.



Backup Slides

Community Resources

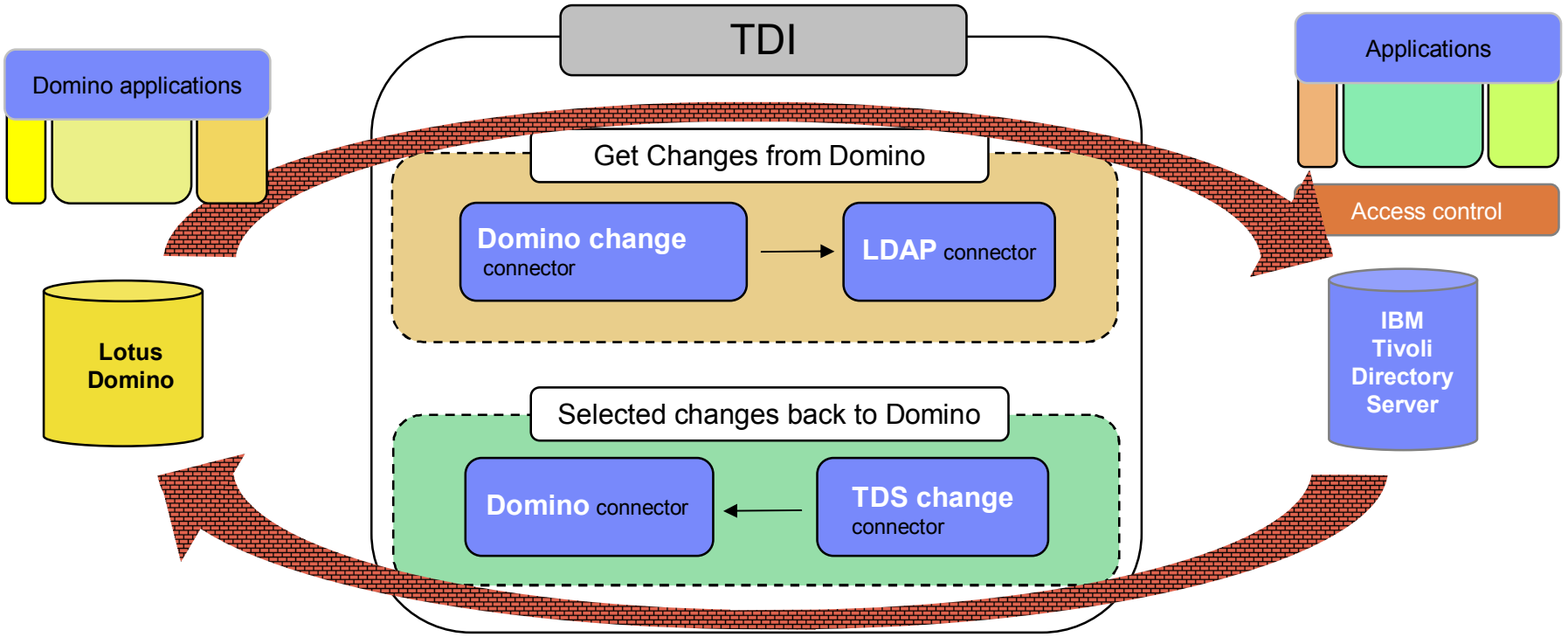
- **Video tutorials, examples, components, documentation+++**
  http://www.tdi-users.org

- **IBM internal site:**
  http://w3.tap.ibm.com/w3ki05/display/TDI/1.+TDI+Wiki+Home

- **TDI Newsgroup**
  http://groups.google.com/group/ibm.software.network.directory-integrator

!! Participate and Share !!

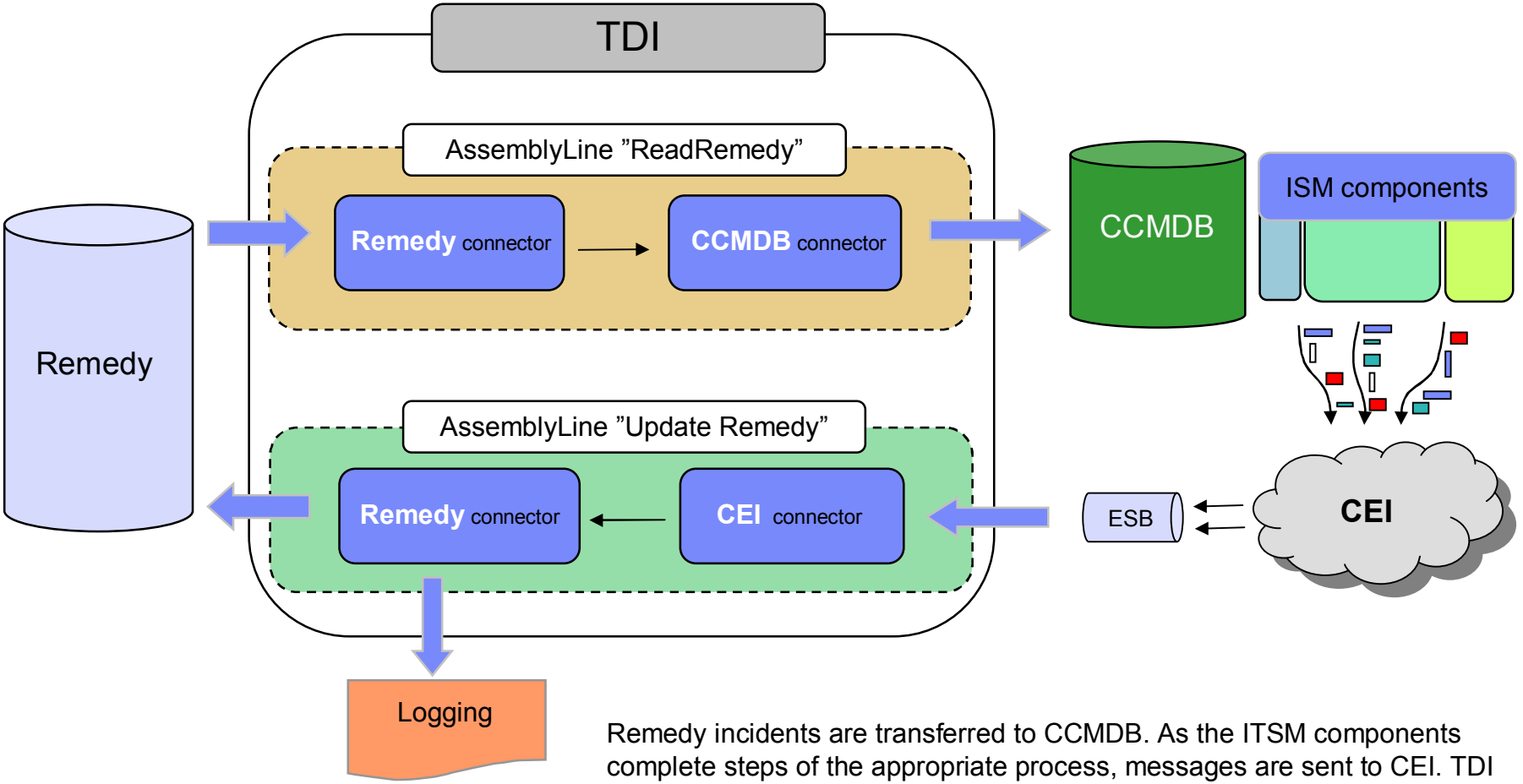
Change propagation



In this scenario, users are managed in Domino and need to be synchronized with TDS for multiple reasons

1. Portal/WAS security is implemented in WAS or with Access Manager into TDS LDAP.
2. The WAS applications need information about users that is maintained in Domino
3. Domain names (dn) must match so that WAS can seemlessly access data in Domino
4. WAS applications might modify user data that needs to be propagated back to Domino

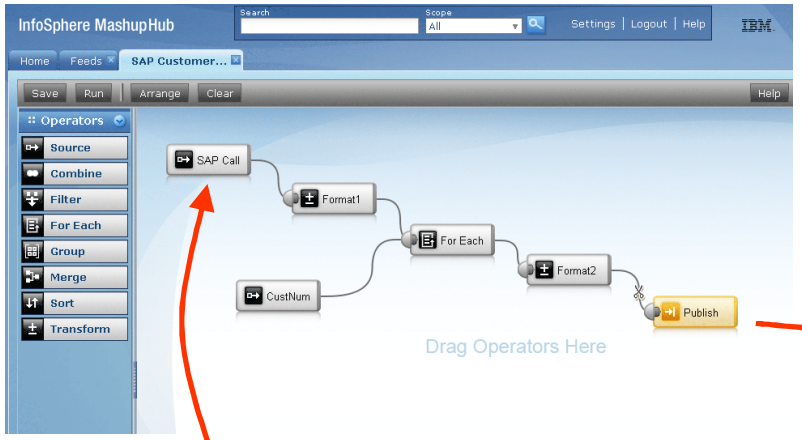
Remedy helpdesk integrated with CCMDB



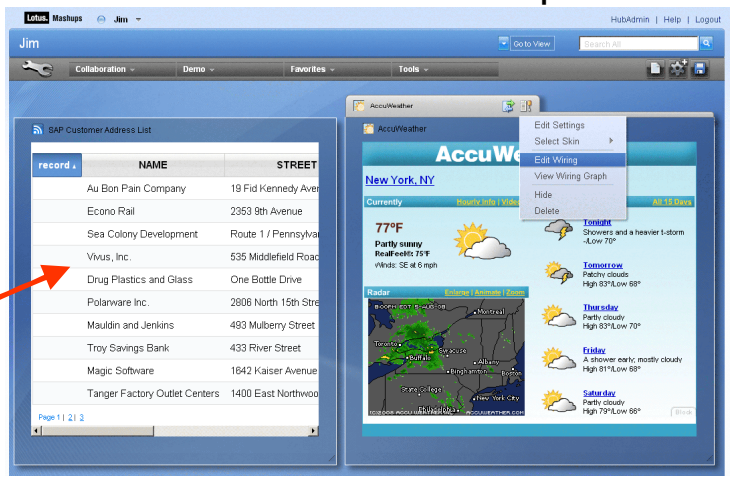
Remedy incidents are transferred to CCMDB. As the ITSM components complete steps of the appropriate process, messages are sent to CEI. TDI subscribes to these messages and update Remedy accordingly so that users can follow the progress of their tickets.

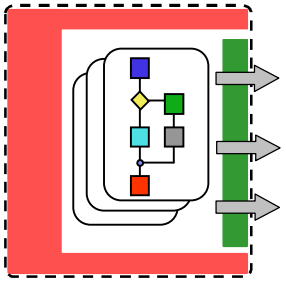
TDI as Feed server in a Mashup environment

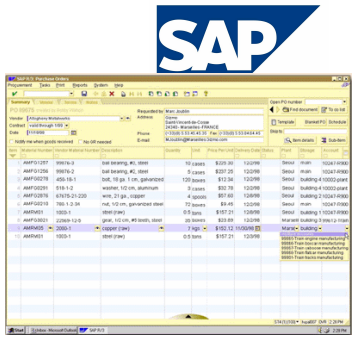
Design and store feed in MashupHub

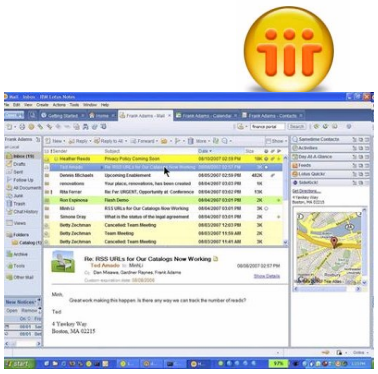


Use feed in Lotus Mashups



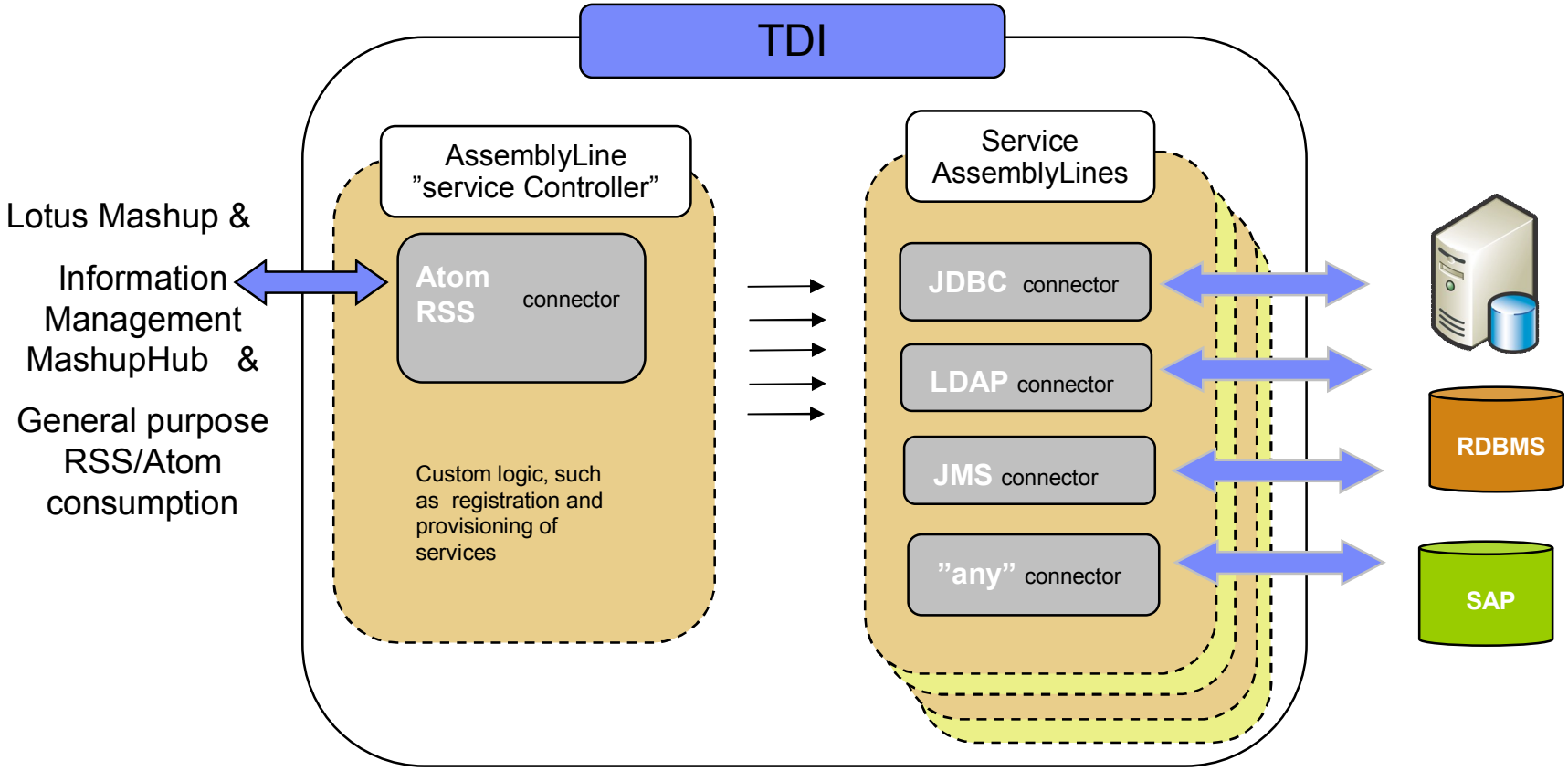






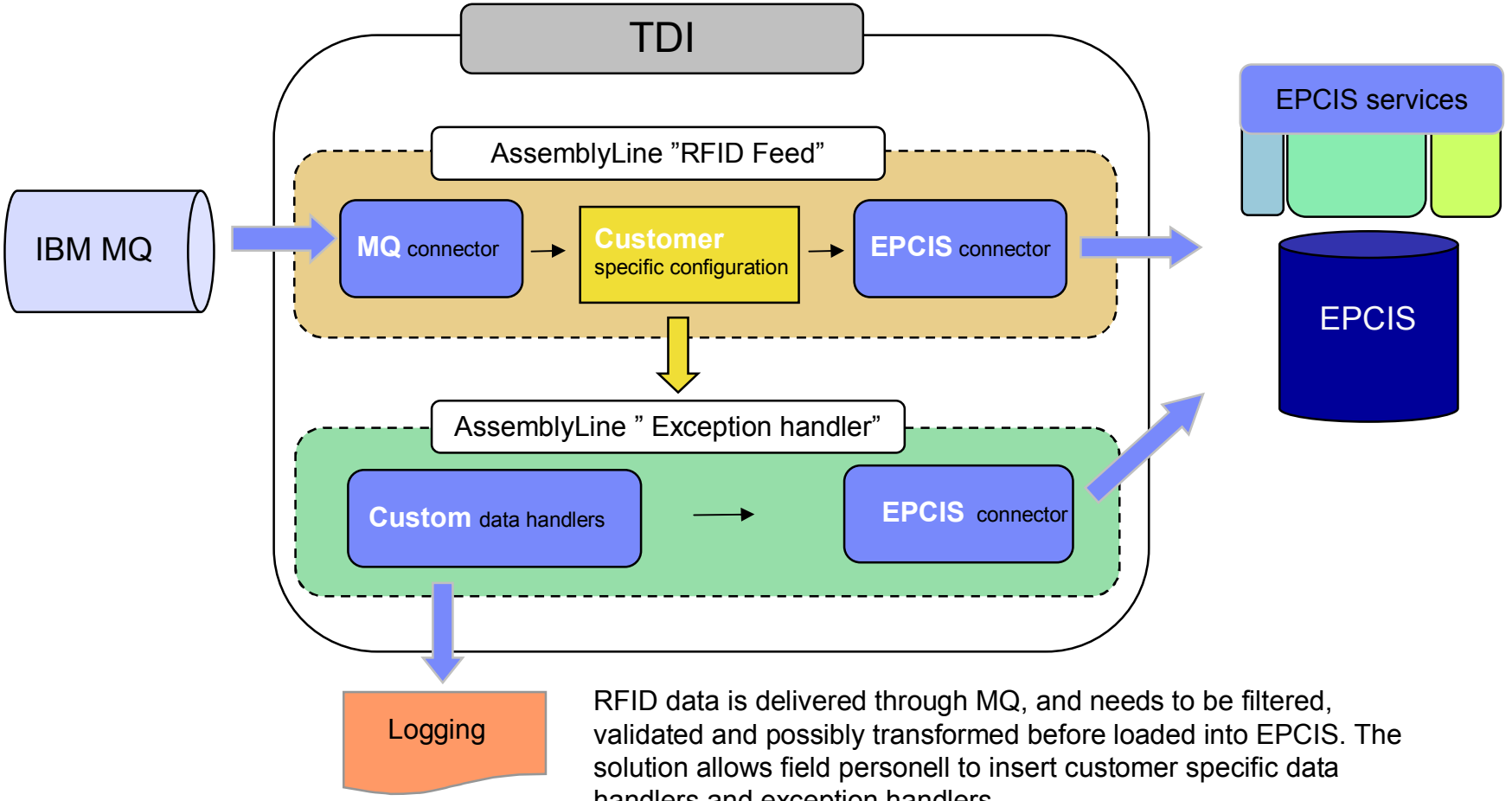
Deliver data from TDI – MIS,
for example SAP and Lotus Domino.
Supports add, modify, and delete operations
with simple or complex integrations

TDI – MIS: Mashup & syndication Integration Server



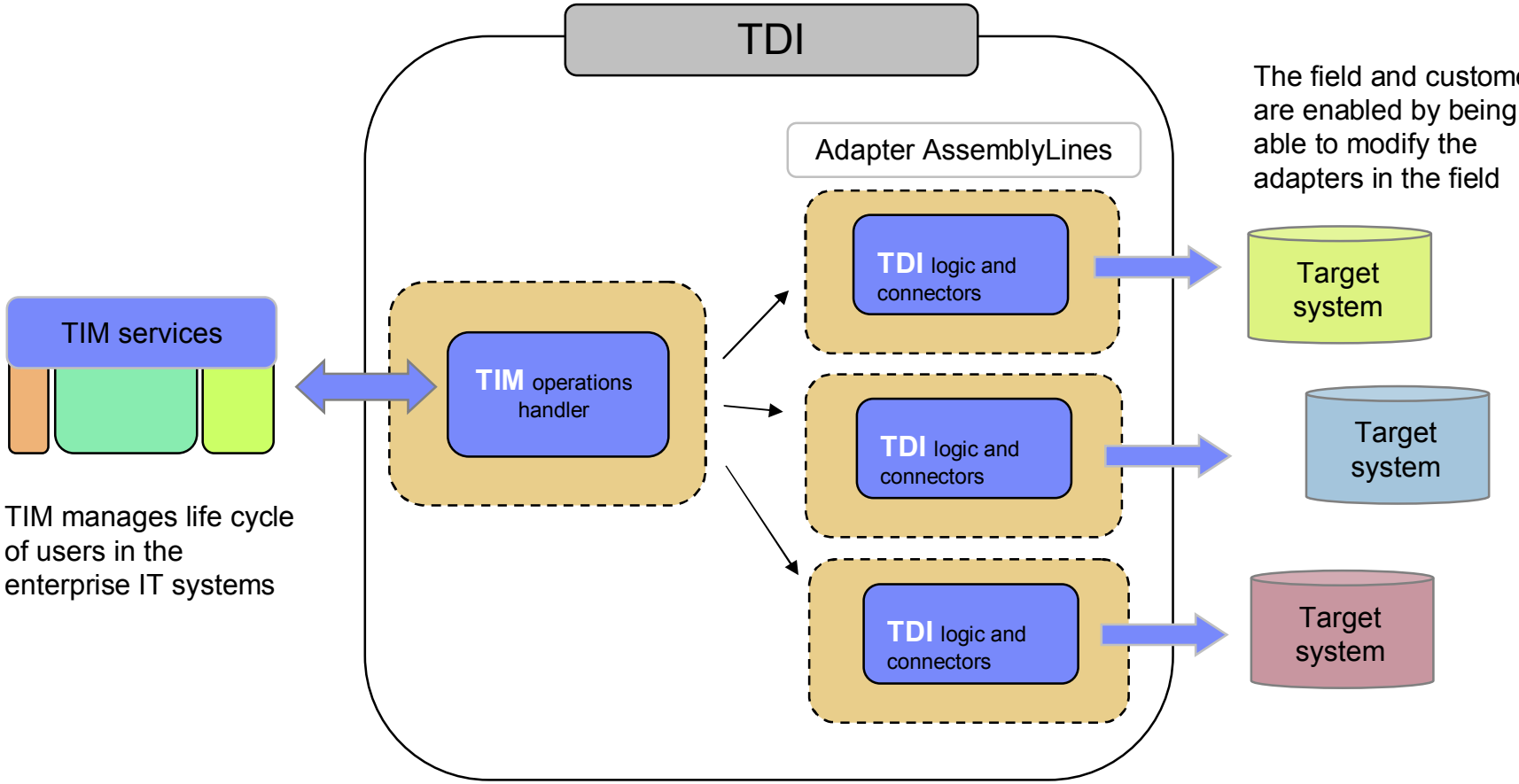
Any number of ”service AssemblyLines” (sAL) are accessed through the multi-threaded ”Service controller”. Each sAL can utilize all of the connectors and capabilitites of TDI to create advanced feeds and services that provide transformation, augmentation, enrichment of the data from any number of connected sources. The sAL can contain as little as a single connector, basically turning TDI into a connector service for the upstream system calling into TDI through REST calls. This could be as simple as CRUD or extended in the ”service Controller” to address specifc requirements.

RFID data into Websphere EPCIS/RFID

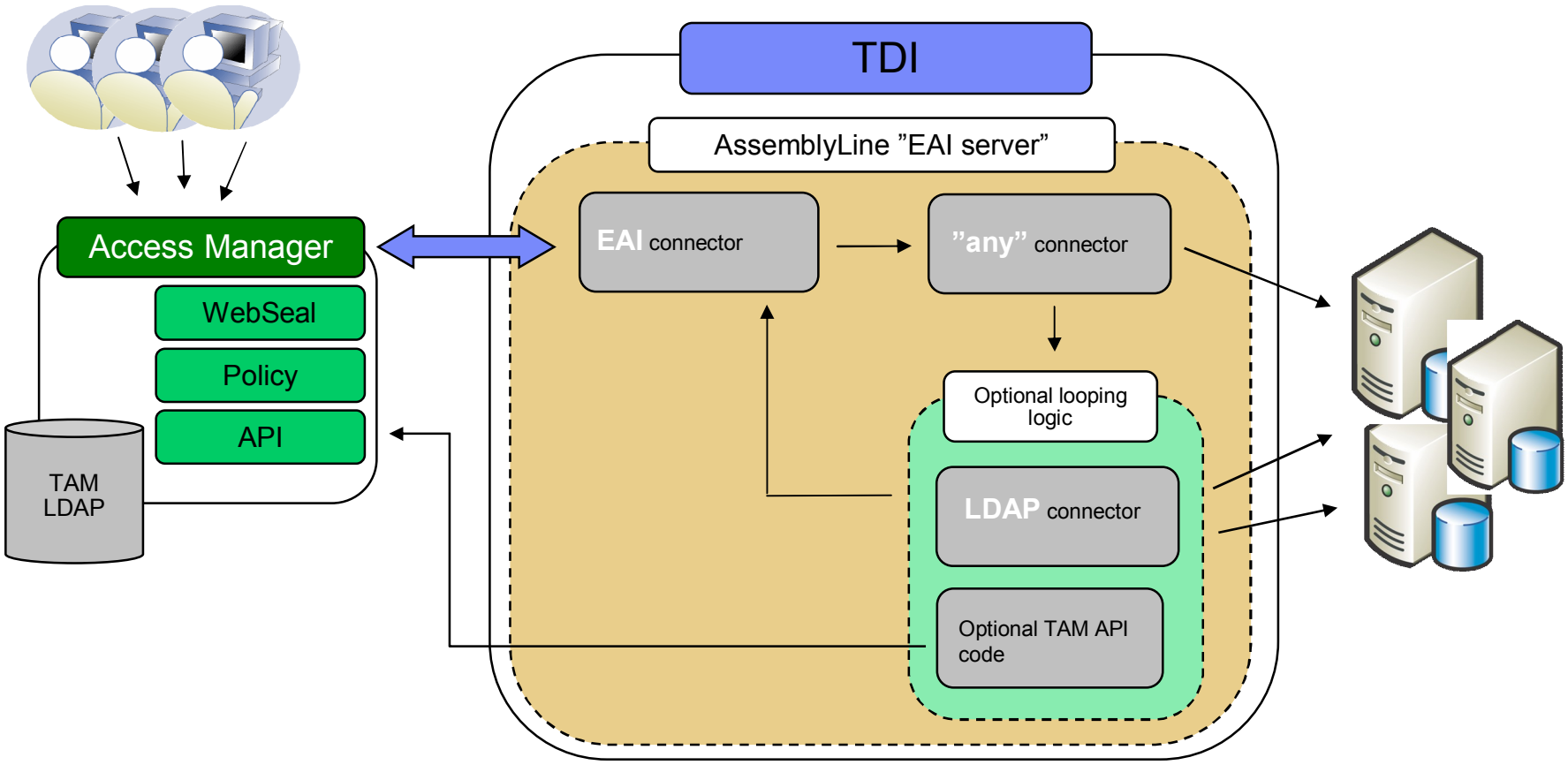


RFID data is delivered through MQ, and needs to be filtered, validated and possibly transformed before loaded into EPCIS. The solution allows field personell to insert customer specific data handlers and exception handlers.

Adapter framework for ITIM (Tivoli Identity Manager)

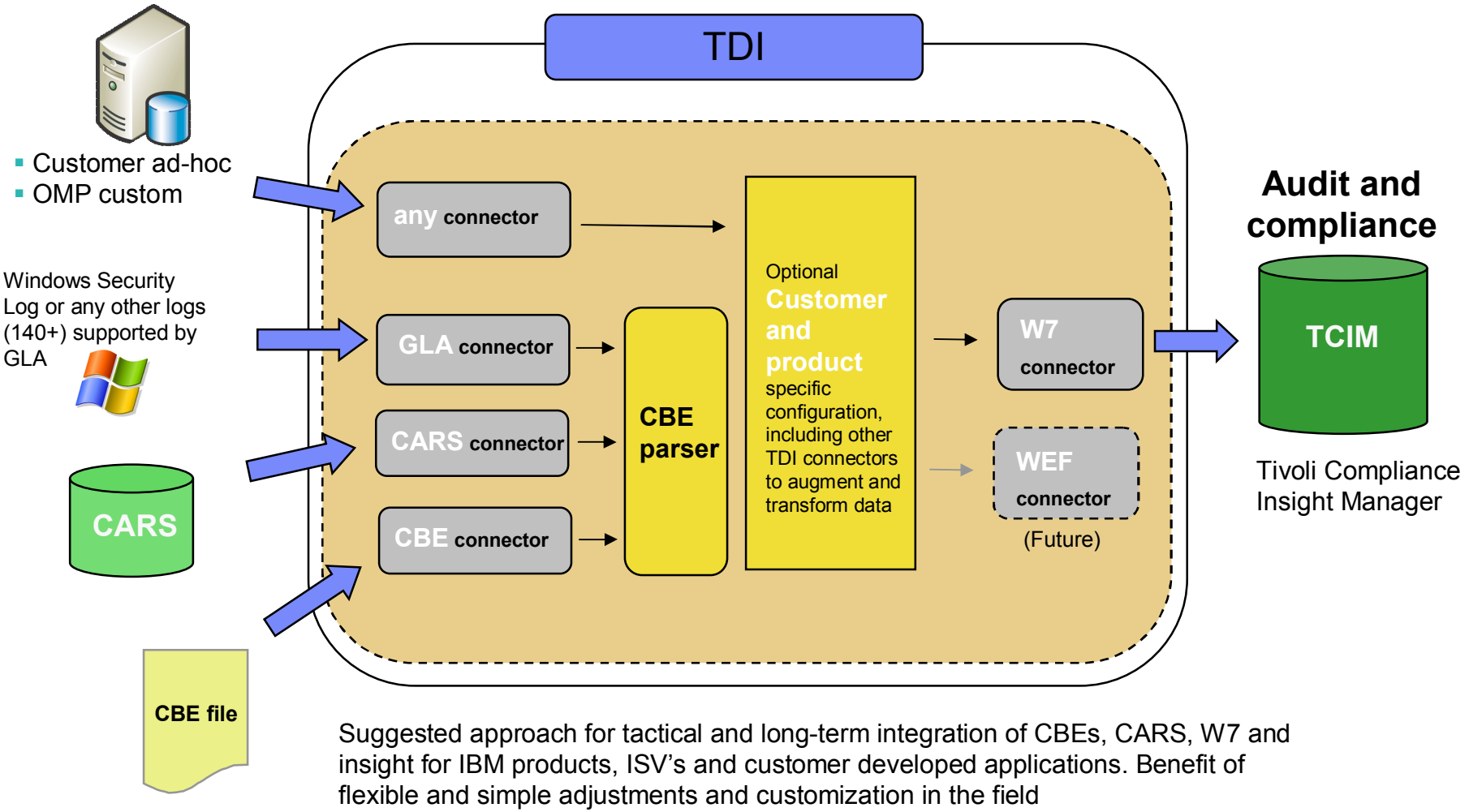


TAM EAI authentication service



TDI provides a generic, run-time authentication server for TAM, where the AssemblyLine can lookup multiple sources, as well as format and transform data. The AssemblyLine above does not illustrate any specific scenario, but illustrates how the integration with the TAM EAI (External Authentication Interface) service works.

Audit integration



Suggested approach for tactical and long-term integration of CBEs, CARS, W7 and insight for IBM products, ISV's and customer developed applications. Benefit of flexible and simple adjustments and customization in the field