interskill learning

Transaction Processing Systems

Objectives

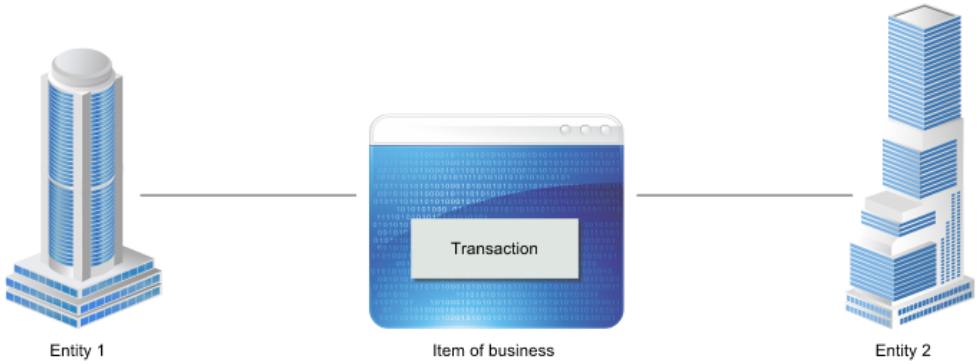
Transaction Processing Systems

In this module, you will examine transaction processing systems and discover their importance in enterprise environments.

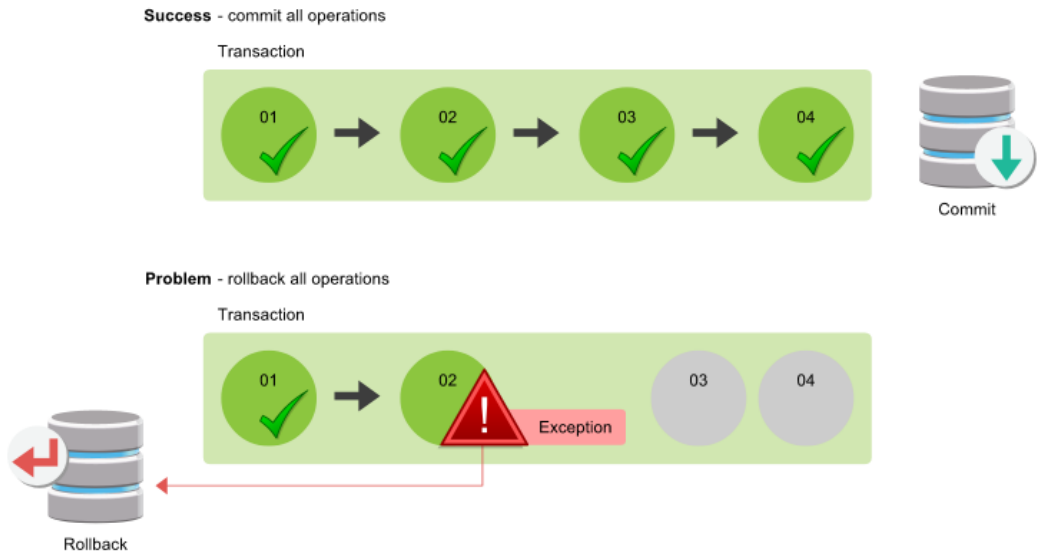
You will focus on the transaction processing systems provided in the IBM enterprise environment.

After completing this module, you will be able to:

- Identify a Transaction
- Define IBM IMS TM
- Define CICS
- Define IBM WebSphere Application Server



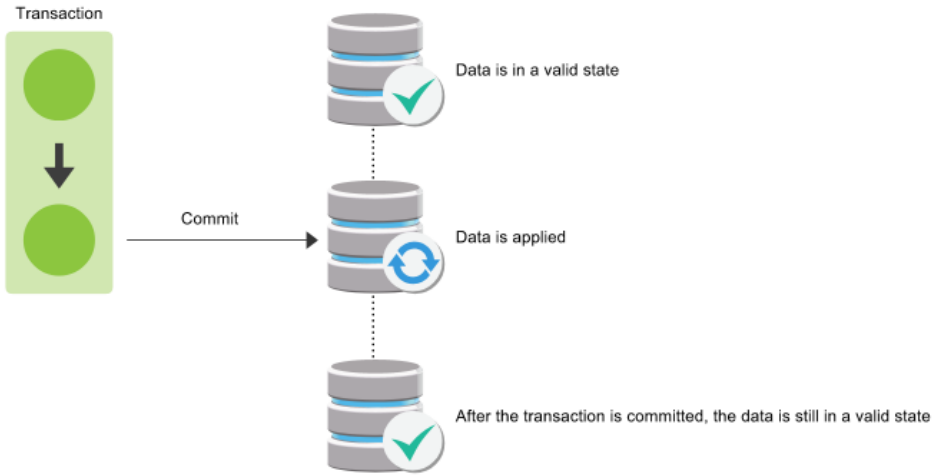
In technical terms, the meaning of "transaction" is similar to its everyday meaning as a single event or item of business between two parties. However, a computer system needs a more exact definition.



To obtain a more exact definition of "transaction", you apply an ACID test where A is for atomicity.

To be atomic, a transaction must execute completely or not at all. If any part of the transaction fails, the whole transaction fails and all data changes must be undone. Only if all operations complete successfully are the data changes made permanent.

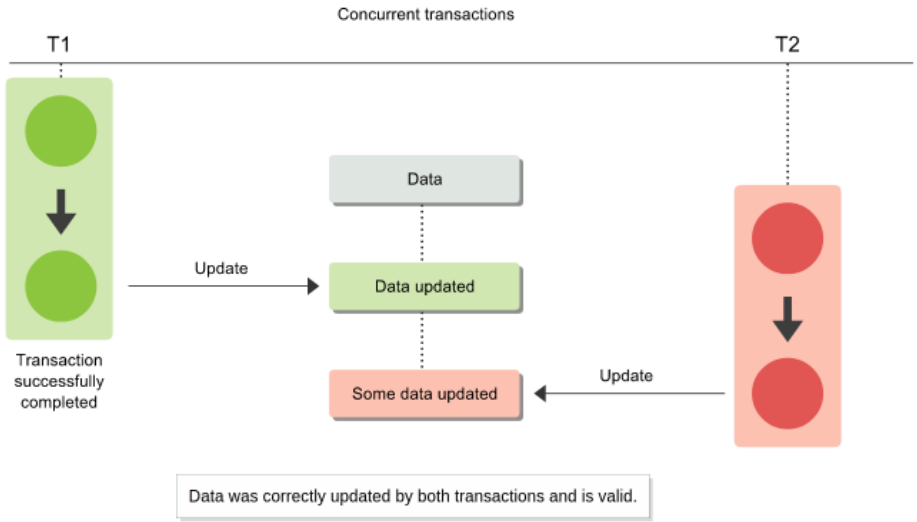
For example, an account transfer consists of both a debit and a credit, but neither will be recorded if one fails.



In the ACID test, C is for consistency.

To be consistent, a transaction taken as a whole cannot violate any integrity constraints associated with resources.

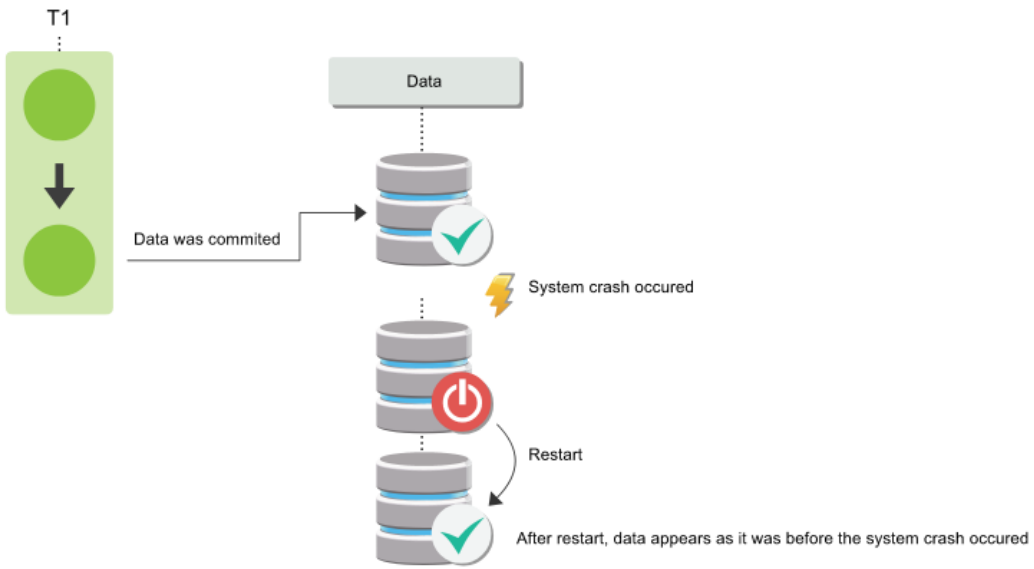
Using the account transfer example, the total of all accounts will remain the same after the transfer. No money will be created or lost from the system as a whole.



In the ACID test, I is for isolation.

To be isolated means that each transaction appears to occur before or after any other transaction, not simultaneously, regardless of whether the transactions executed concurrently; therefore, the data that a transaction accesses cannot affect, or be affected by, any other part of the system until the transaction is complete.

If this holds true, the system as a whole will remain consistent whether the transaction completes or backs out.



In the ACID test, D is for durability.

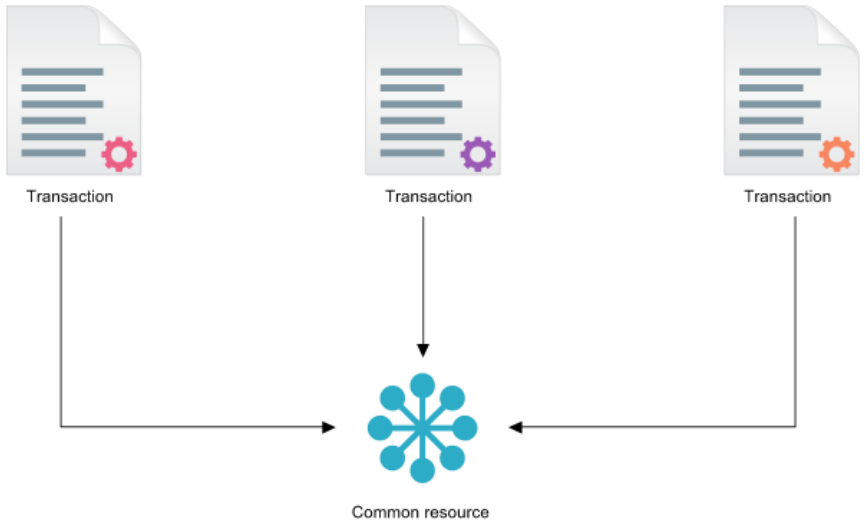
If a transaction completes, the completed state will survive any subsequent failures.

ACID transactions are ideal when dealing with the real world of very high volumes, transaction rates, and concurrency, but some restrictions must be applied.

ACID transactions should be short in time span because they lock resources and prevent other transactions from running for that period of time.

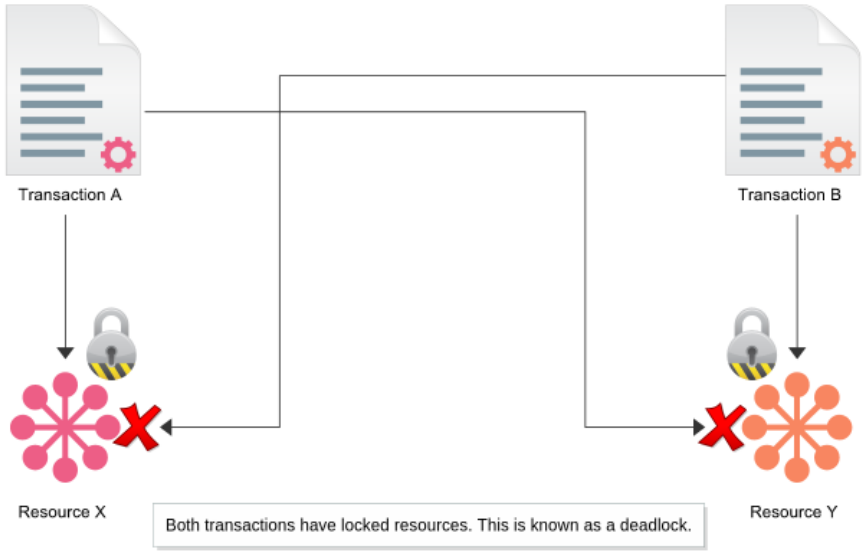
ACID transactions should not be too big in scope. They should not attempt to update a whole database when a row at a time is possible.

Click Play for a demonstration of this concept.



In real systems, compromises must be made to enable acceptable throughput.

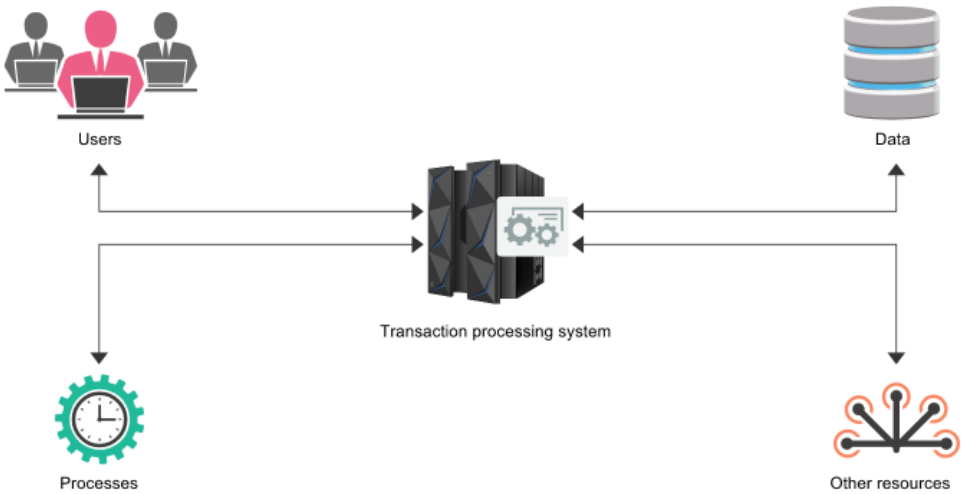
Sometimes two or more transactions require access to the same set of resources. Isolating that resource would severely slow transaction throughput so it is necessary to relax isolation. This must be done with care.



A transaction may require access to a resource that is locked by another transaction, but this other transaction is waiting to access a resource that is locked by the first transaction. This situation is known as a deadlock.

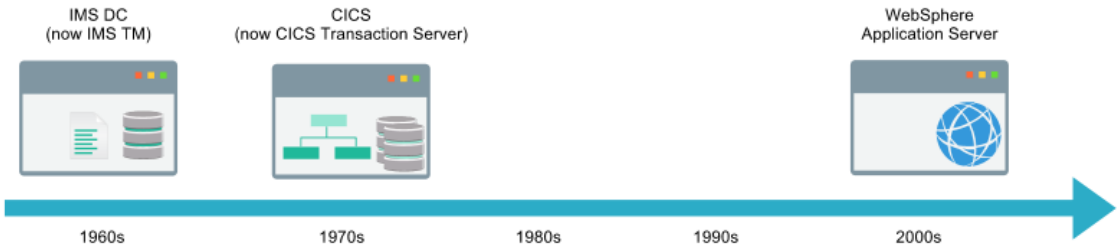
To resolve a deadlock, the transaction processing system must arbitrarily cancel one of the transactions and enable the other to complete.

The canceled transaction can then be attempted again.



Now that we have defined what a transaction is, you will examine the systems that provide them and see how they are implemented on IBM enterprise systems.

These systems are usually referred to as transaction processing systems or transaction monitoring systems.



IBM provides three systems that can be considered to be transaction processing systems. These systems have been developed over time as the technical environment evolved.

The three systems are IMS TM, CICS Transaction Server, and WebSphere Application Server. You will look at each of these in turn.

Note that IBM also provides an integrated transaction processing operating system called z/TPF. As this is very specialized, we will not cover it here.

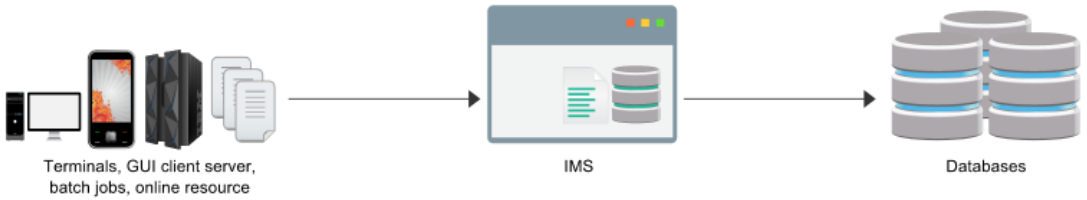
IMS, which was developed in the 1960s, consisted of two components, the database (IMS DB) and data communications (IMS DC). Over time, these components evolved into separate products.

IMS DB has developed as a self-contained database system that is able to work with other transaction processing systems.

IMS DC has become the transaction manager (IMS TM), a transaction processing system that can manage all types of enterprise databases, including IMS DB.

IMS was designed to provide online access and to update the data of IBM's biggest customers at that time. As these early adopters of online transaction systems were such large organizations, the systems running under IMS have become some of the largest systems in existence.

IMS has been updated to handle these requirements. Today, it can be found controlling systems with tens of thousands of online users accessing terabytes of data.

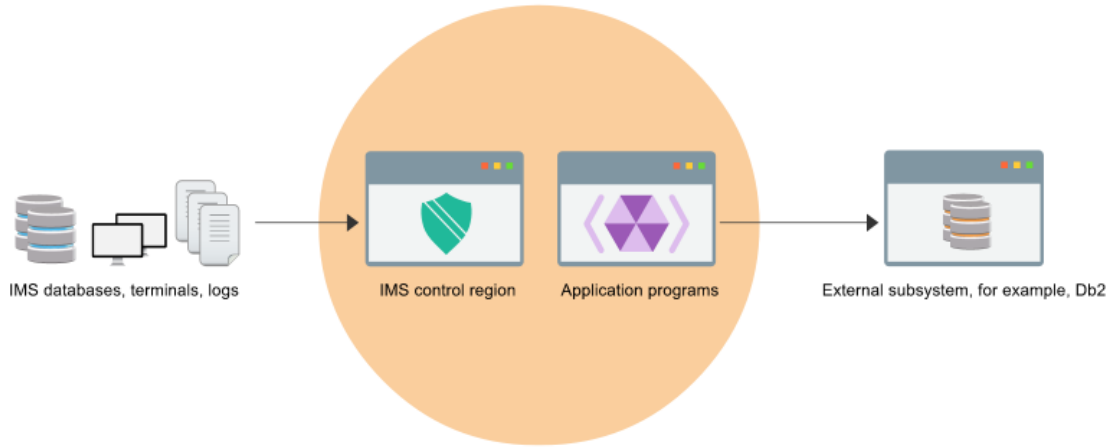


Originally, IMS TM only provided green-screen terminal access and concurrent batch access.

Batch access through an online system like IMS may seem strange, but it is often necessary to enable a concurrent update in batch to resources that are also available online.

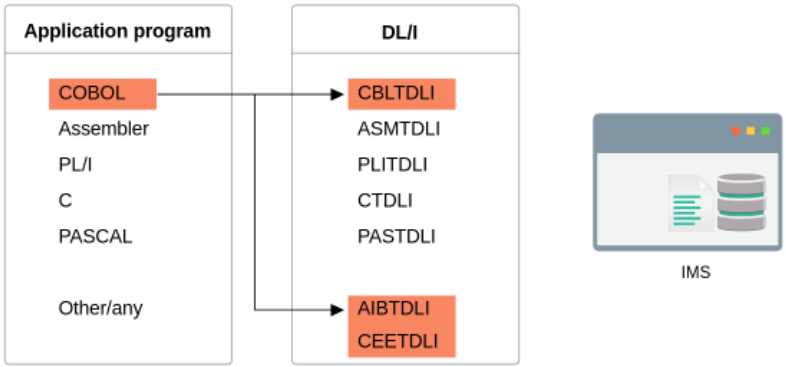
In this situation, IMS TM controls the databases and data sets, manages concurrency and recovery, and provides interfaces that may be used by batch programs.

Modern IMS provides interfaces for Web and client servers, enabling these types of processes to concurrently and securely update the resources that it controls.



IMS would be of little use if it could not be used to produce business value. Programs can be written to run under the control of IMS and by communicating with IMS, they can access the data, terminals, and other resources under IMS control.

These programs can also access other external subsystems, such as Db2.



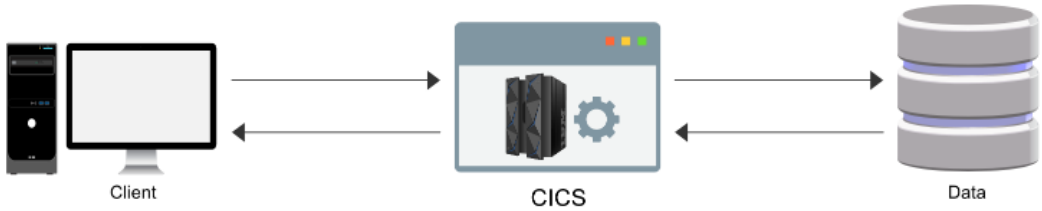
IMS programs can be written in most of the mainstream enterprise programming languages. Most programs communicate with IMS by using an interface language or API called DL/I.

DL/I has a complex syntax of functions and arguments that enable a program to control both an IMS online environment and a database.

Specific DL/I interfaces are provided for some languages, as shown above, or the generic CEE or AIB interfaces can be used. For example, a COBOL IMS program would contain CBLTDLI calls to access and control IMS.

CICS is an alternate online transaction system to IMS TM, which was developed soon after IMS. Unlike IMS, however, it has never had an integrated database.

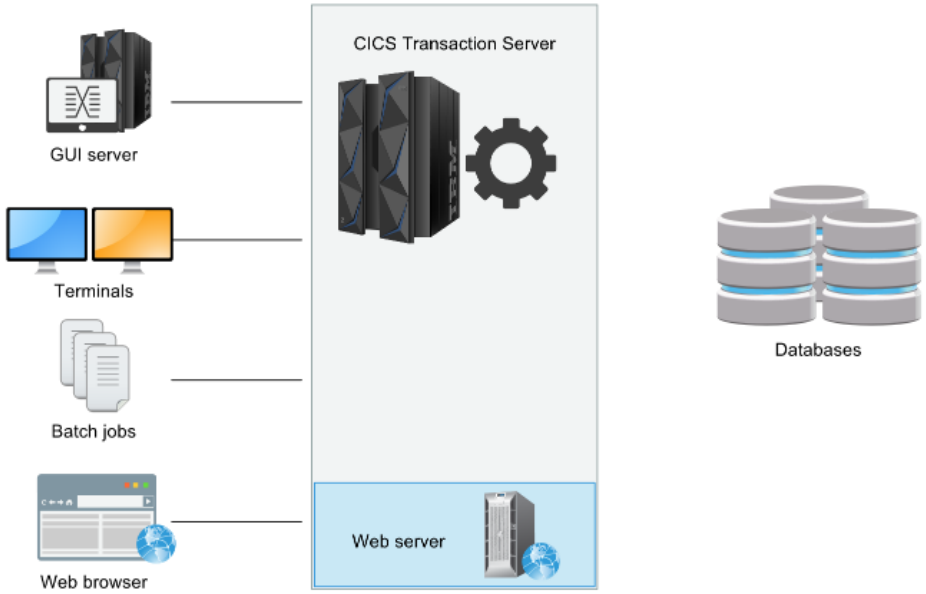
CICS has been seen as a simpler development environment. Initially, many smaller systems were developed in CICS, but CICS quickly grew to match IMS for capacity and, in many ways, to surpass it in functionality.



CICS can be used to manage online access to data contained in IMS DB, Db2, and native data sets like VSAM.

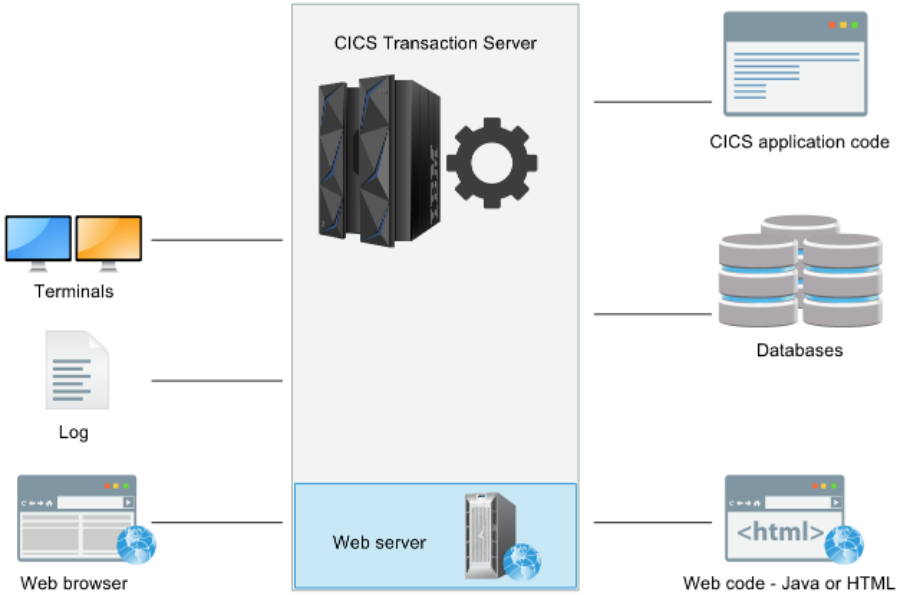
In fact, the combination of VSAM and CICS provides VSAM data with much of the protection and functionality of IMS DB. This combination was used in many early implementations.

CICS has also been used by many third-party vendors as the transaction system of choice for their databases and products.



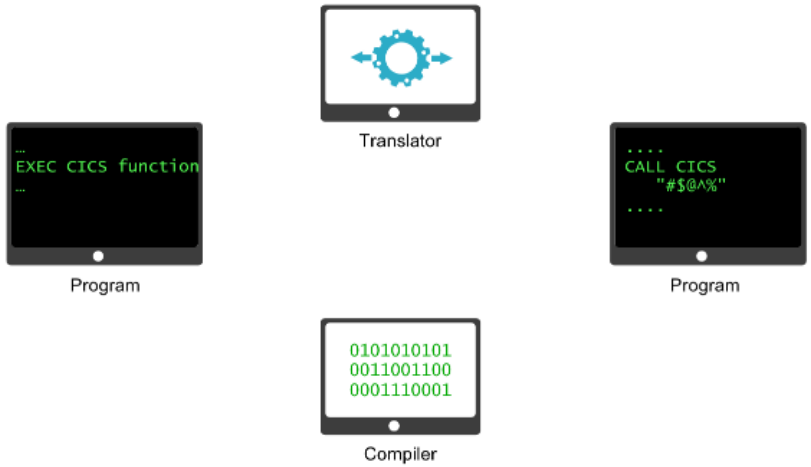
CICS has been positioned as the premier IBM transaction system, both for legacy and Web-based systems. To reflect this, it is now called CICS Transaction Server.

The CICS Transaction Server continues to be a fully functional enterprise transaction system that can run the applications built up over many years. It has also been extended to include Web server capabilities, both for modern Web transactions and to enable Web-based systems access to legacy transactions.



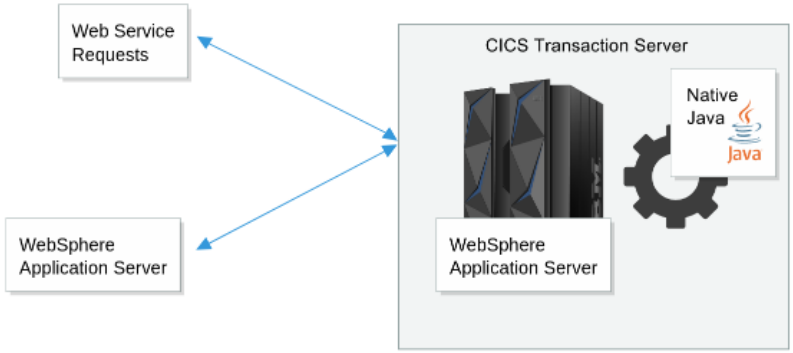
Like IMS, CICS would be of little use without the ability to write business applications to run under its control. CICS enables business applications to be coded in all the IBM enterprise languages, as well as Java.

These programs are usually launched by CICS and they communicate with CICS to receive input and access data.



Unlike IMS where the programmer calls an IMS program, CICS allows most programs to use a simpler interface - the CICS command level interface.

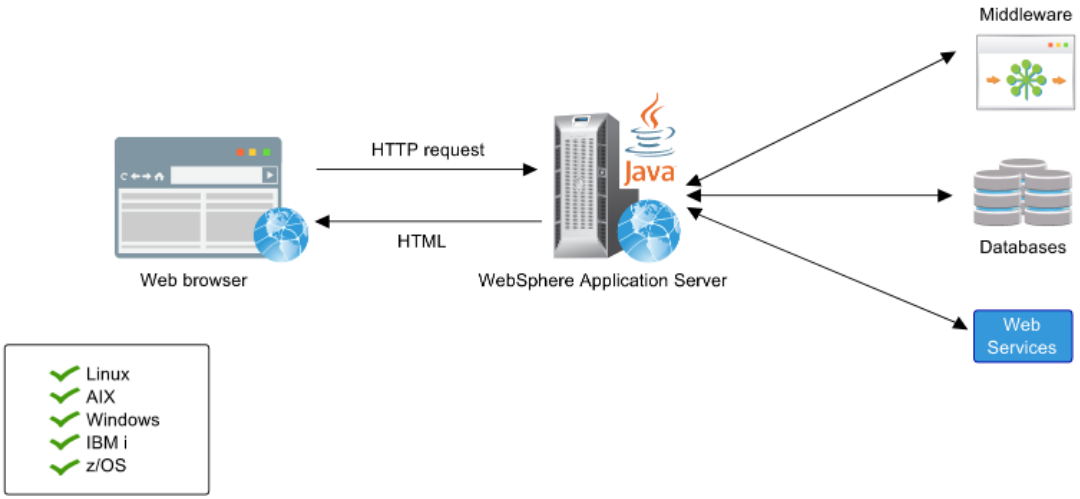
CICS command level allows applications to code straightforward commands beginning with EXEC CICS. These commands are passed through a translator before the program is compiled to convert them to CICS calls. These calls are then compiled by the normal programming language compiler.



CICS is not just for traditional programming languages and interfaces. CICS applications can be coded in Java. WebSphere Application Server can also run within CICS to execute Java applications.

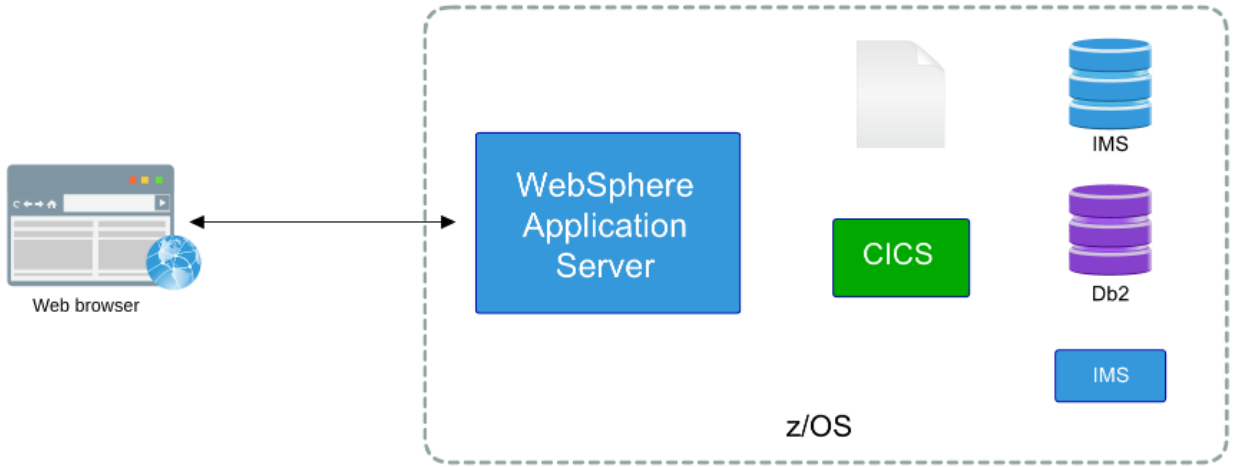
CICS provides features to process incoming web services requests, including JSON and SOAP requests. CICS programs can also send outgoing web services requests, and external WebSphere Application Server programs can also interact with CICS.

CICS features are covered in more detail in the Interskill CICS courses.



The previous screen talked about IBM WebSphere Application Server (WAS). WAS is a J2EE platform. This provides a platform for Java code servicing incoming requests to execute, and features to allow this code to access resources including middleware such as IBM MQ, databases, web services, and more.

WAS runs on many platforms, including Linux, AIX, Windows, and IBM i.



WAS on z/OS provides access to legacy resources including IMS and Db2 databases, IMS and CICS transactions, z/OS data sets, and more. WAS applications can provide a modern, web-based interface to these legacy resources.