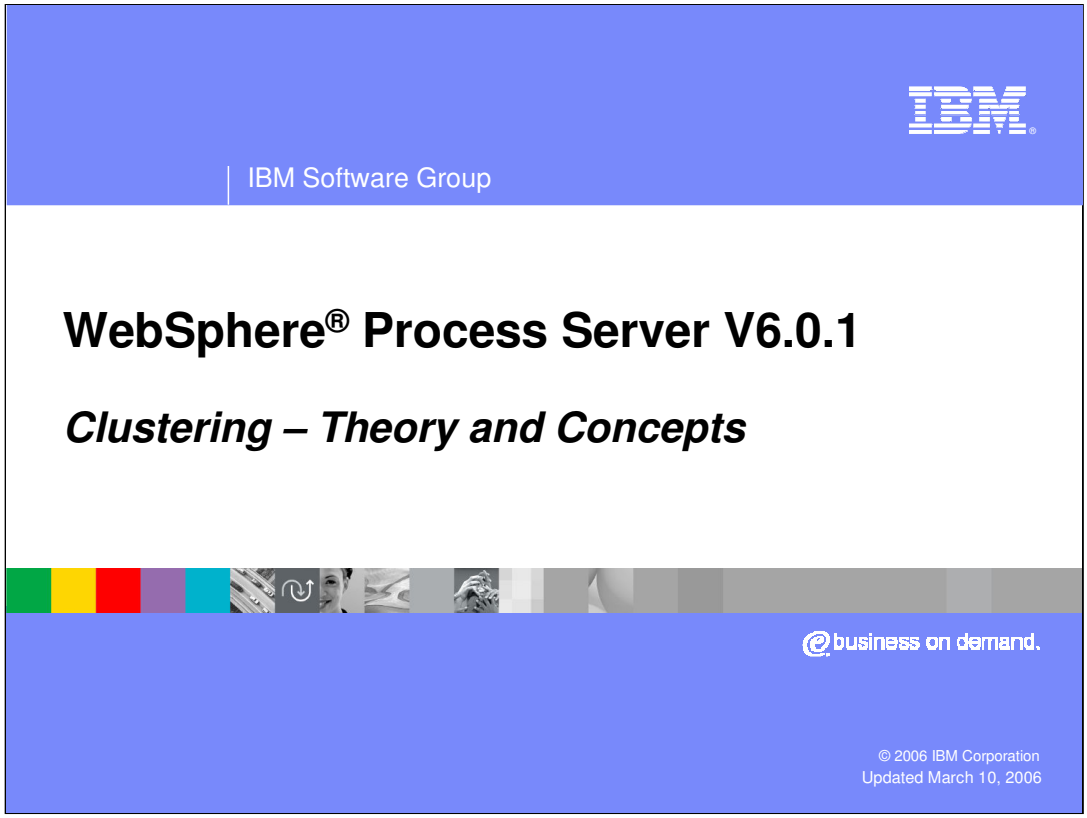
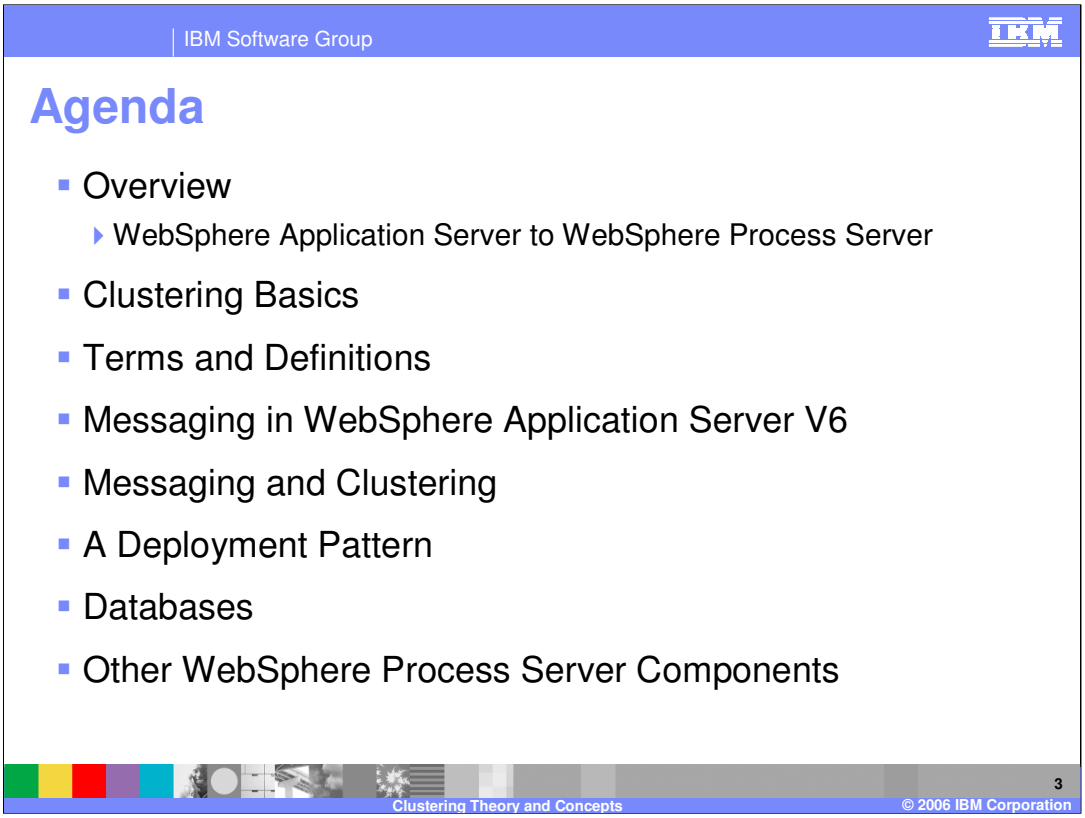


This presentation will cover the theory and concepts of clustering for WebSphere Process Server V6.0.1

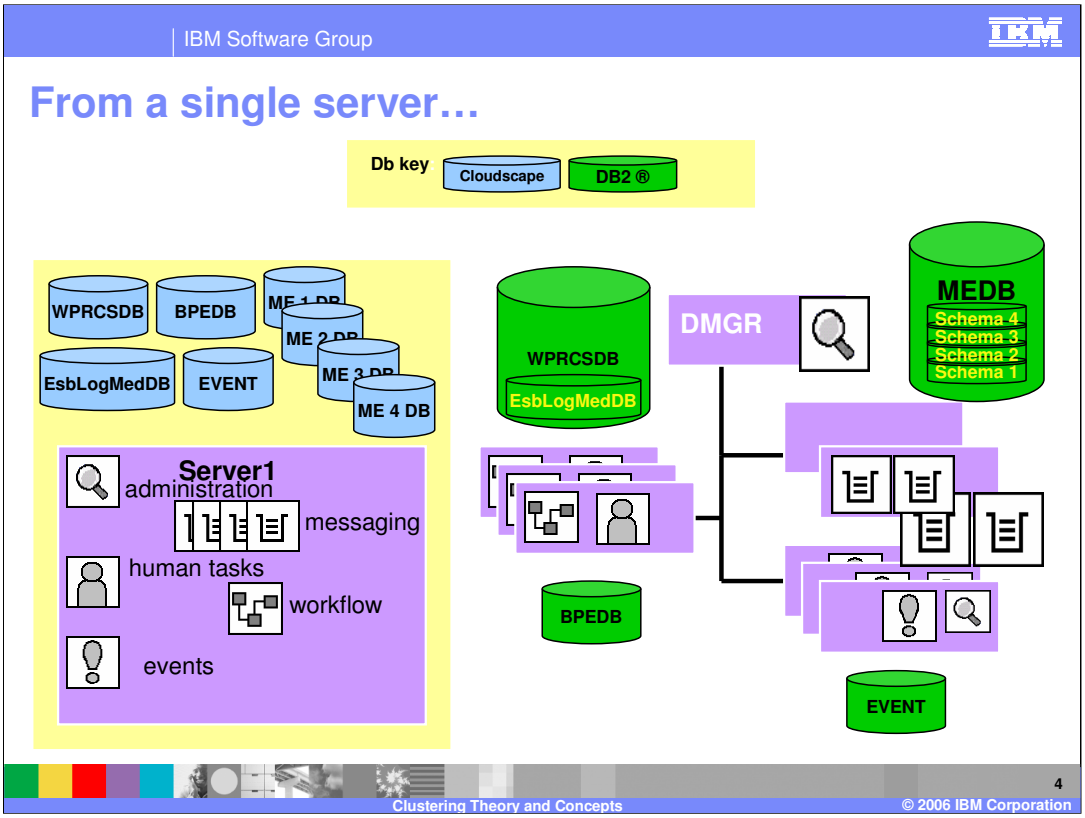
Assumptions and goals

- Familiarity with clustering in WebSphere Application Server Network Deployment V6
- Explain clustering and messaging
- Discuss *high availability* and *scalability* in terms of Message Engines
- Introduce the configuration to be used for
  - the tutorial exercise
  - presentation on administering a clustered environment

This presentation assumes that you are familiar with clustering in WebSphere Application Server Network Deployment V6.0. The goal is to present the underlying and motivating issues involved with clustering and messaging in a WebSphere Process Server environment. The next goal is to discuss high availability and scalability in terms of Message Engines. Next, you will be introduced to the configuration for *high availability* that will be used for the tutorial and the presentation on administering a clustered environment.



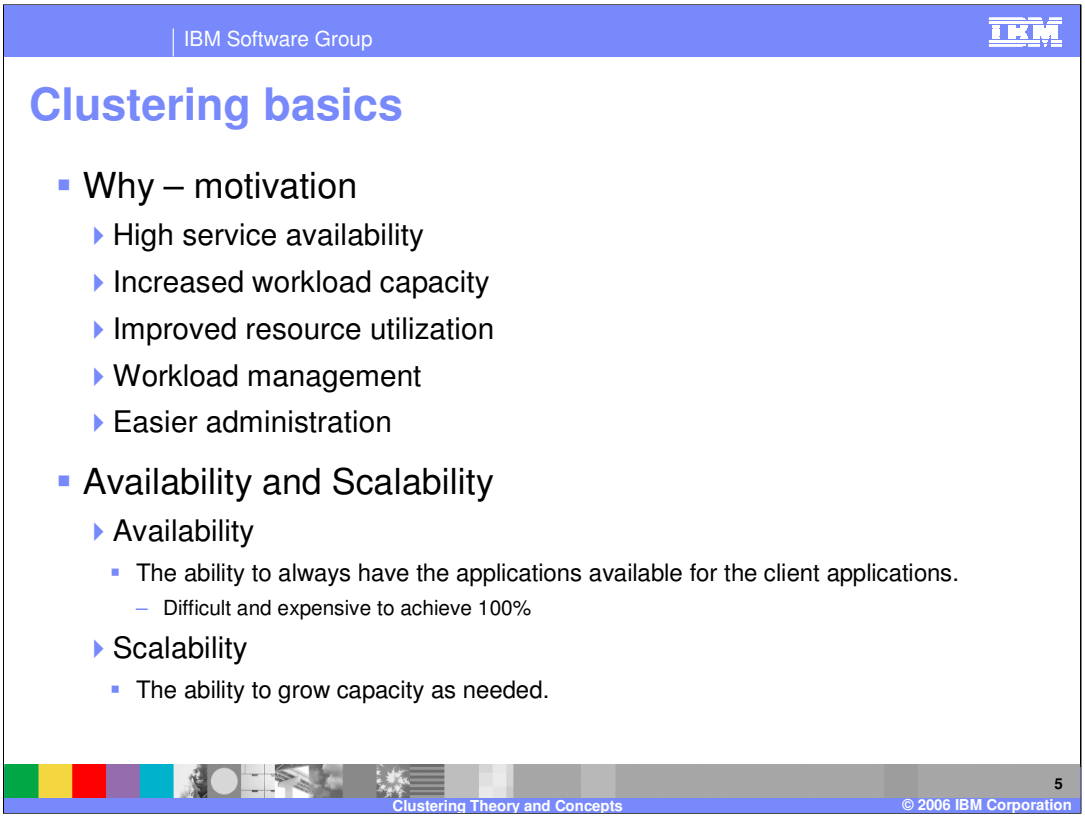
The agenda for this presentation begins with an overview of the differences between WebSphere Application Server and WebSphere Process Server. Then it will cover clustering basics, including some terms and definitions, before moving into Messaging topics. Finally, deployment options, database options, and other components will be discussed.



This demonstrates the mapping of the artifacts as they are created in the WebSphere Integration Developer environment to the WebSphere Process Server Network Deployment V6 runtime environment.

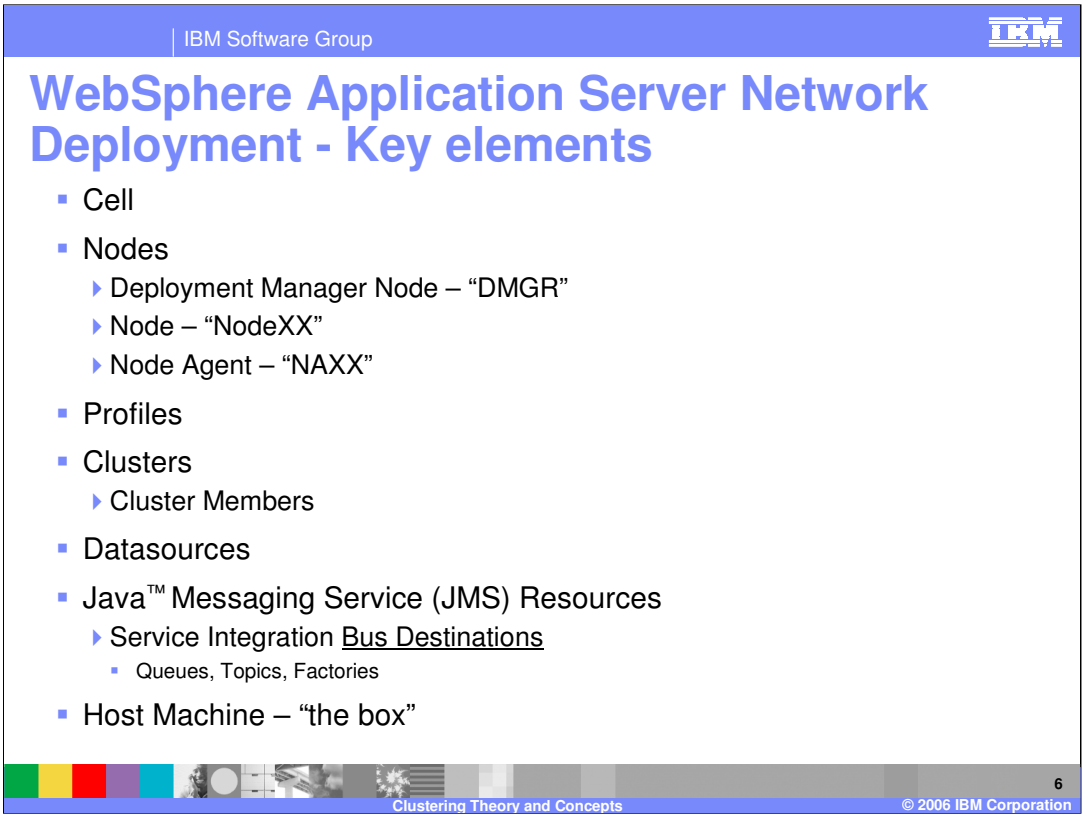
Initially everything is in the WebSphere Integration Developer, using cloudscape for the databases and everything all in one server.

1. Move the WPRCSDB and incorporate database required for the Enterprise Service Bus mediations.
2. As the messaging component moves to the runtime environment, the datastores for the Message Engines are consolidated to a centrally located, remote database and the JMS resources are established on numerous servers.
3. As the business processes move to the runtime, a new database is created and the applications are deployed on numerous servers.
4. Moving the Common Event Infrastructure component to the runtime cause the creation of yet another database specifically for the Common Event Infrastructure events, which can be generated from any of the servers, some from the Business Process Choreographer infrastructure and some from applications.



As an architect or system administrator using WebSphere Process Server to implement your business processes, you expect to have a system that will tolerate failure and allow for maintenance without the loss of data or loss of service. You also need to be able to add processing capacity, to grow your systems to meet increased user demand.

WebSphere Network Deployment provides the ability to create logical groups of servers, with the servers being distributed across one or more machines. This capability provides a mechanism to tolerate failover, apply maintenance to some servers while others keep running, grow or shrink capacity by adding and removing servers from the group, all with a single point of administration.



Shown here are the key elements of a WebSphere Network Deployment Cell which are necessary for a discussion on clustering.

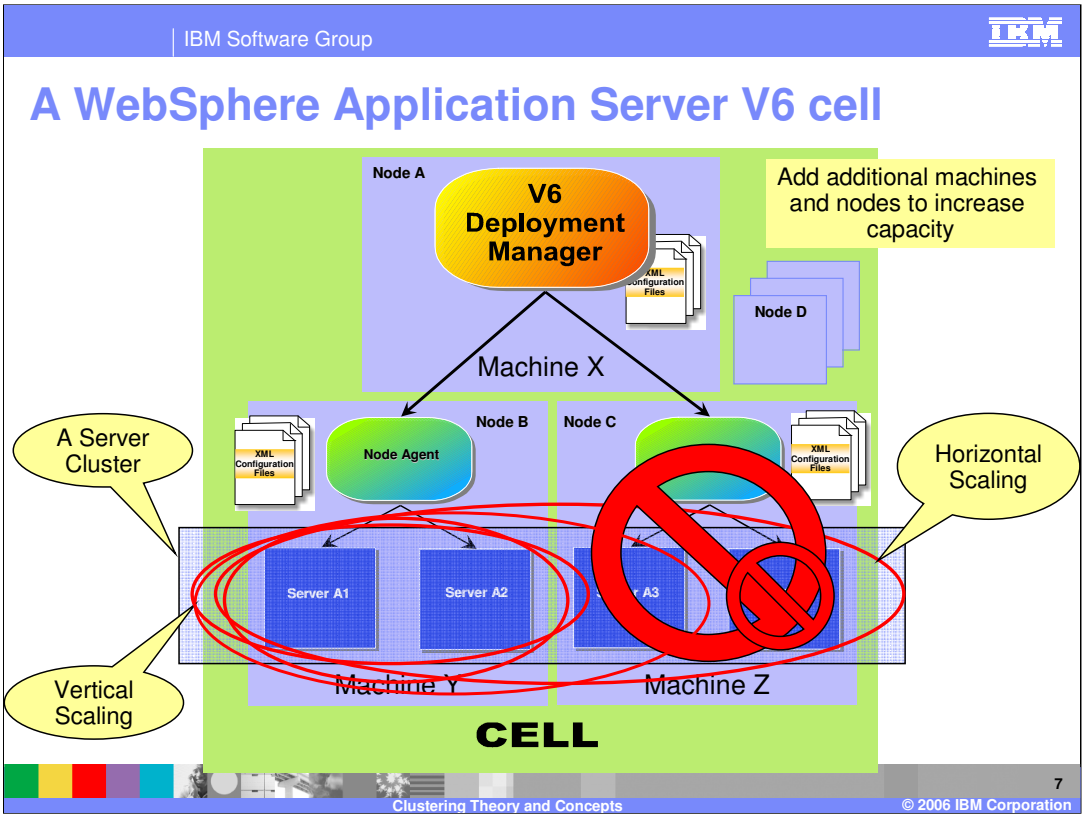
A **node** is a logical grouping of managed servers. A node usually corresponds to a logical or physical computer system with a distinct IP host address. Nodes cannot span multiple computers.

The WebSphere Application Server **profile** defines the runtime environment for a node. To create a node, that is to say, the runtime environment to host the servers on a given host machine, you run the Profile Wizard. One of the primary purposes of the profile is to provide separation between the WebSphere runtime artifacts specific for a given configuration, and the WebSphere Application Server Binaries, which are common to all configurations.

Clusters are sets of identical servers that are managed together and participate in workload management. The servers that are members of a cluster can be on different host machines.

A **Bus destination** is a virtual location within a service integration bus, to which applications attach as producers, consumers, or both to exchange messages.

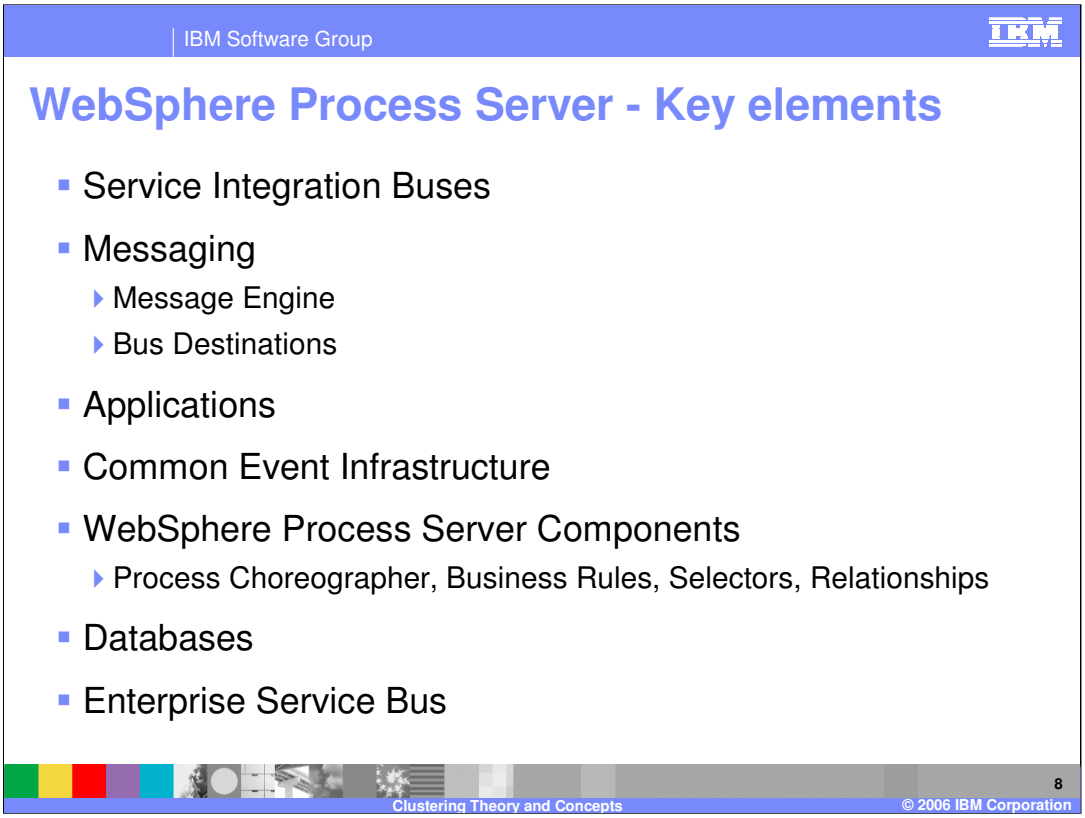
Another term is “the box” or “the host machine” which often gets confused with the server. A server is the Application Server, which is a software entity that hosts applications, whereas



The nodes may be on separate boxes or they may be on the same box. They are shown here on separate boxes and the deployment manager is on a separate machine as well. To host more than one node, the host machine must have ample memory, disk space, and processor capacity.

All four servers in the **server cluster** are identical.

1. If Machine Z needs to be taken off-line for maintenance then Machine Y will still have 2 servers available to service requests.
2. If any one of the servers experiences a problem and depending on how the system is configured, it will be possible for one of the remaining 3 to recover the work in progress, that is, **failover** to one of the remaining servers in the server cluster.
3. Adding additional application servers to a server cluster on the same machine is called ***vertical*** scaling.
4. Adding additional application servers to the server cluster on a different machine is called ***horizontal*** scaling.
5. To **scale up** the capacity, one or more machines can be brought online and the new nodes can be federated into the Cell and more servers can be added to the cluster.



Although Service Integration Buses and Messaging are part of WebSphere Application Server Network Deployment V6, they are listed here because they are key elements for WebSphere Process Server solutions.

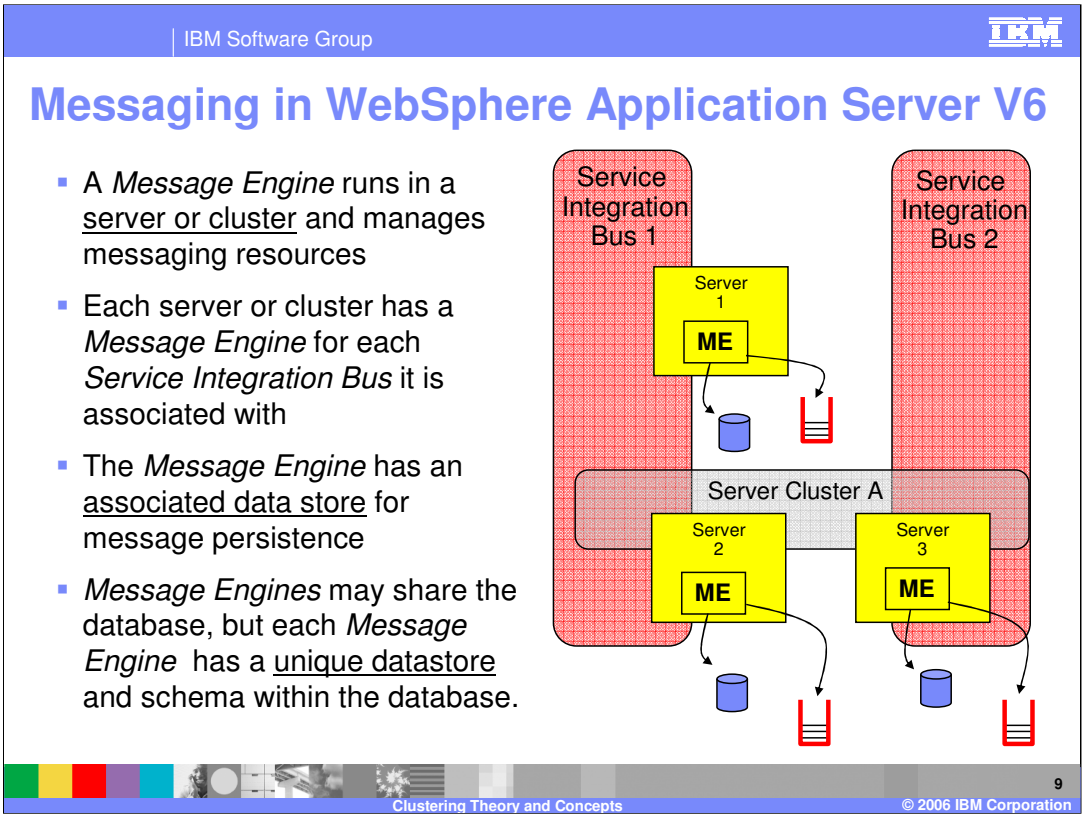
A ***service integration bus*** supports applications using message-based and service-oriented architectures. A bus is a group of one or more interconnected servers or server clusters that have been added as members of the bus. Applications connect to a bus at one of the messaging engines associated with its bus members.

A ***messaging engine*** is a server component that provides the core messaging functionality of a service integration bus. A messaging engine manages bus resources and provides a connection point for applications.

JMS destination is the queue

The ***Enterprise Service Bus*** is a new feature provided with WebSphere Process Server V6.0.1 that adds additional messaging and service oriented features such as complex transformations and mediations.

Messaging is at the heart of the WebSphere Process Server functionality. Its used for the Common Event Infrastructure, Business Process Choreography and for the asynchronous applications. This is why you need a thorough understanding of WebSphere Application Server Network Deployment V6 messaging and clustering in order to



Server 1 is a member of the Service Integration Bus **Service Integration Bus 1**

Server 2 and 3 are both members Server Cluster A which is also a member of the **Service Integration Bus 1** and **Services Integration Bus 2**.

A ***service integration bus*** supports applications using message-based and service-oriented architectures. A bus is a group of one or more interconnected servers or server clusters that have been added as members of the bus. Applications connect to a bus at one of the messaging engines associated with its bus members.

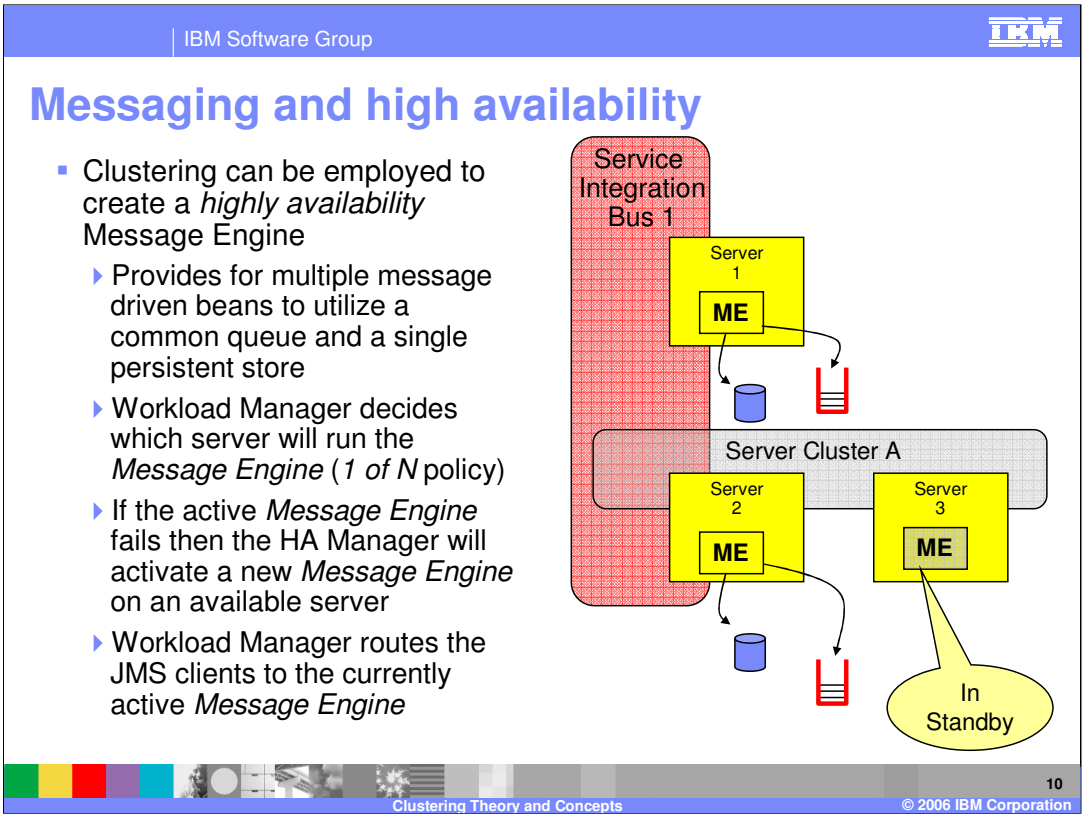
Each ***messaging engine*** is associated with a server or a server cluster that has been added as a member of a bus. When you add an application server or a server cluster as a bus member, a messaging engine is automatically created for this new member. If you add the same server as a member of multiple buses, the server is associated with multiple messaging engines (one messaging engine for each bus).

Bus members

The members of a service integration bus are the application servers and server clusters within which messaging engines for that bus can run.

Note:

The are two different kind of aggregations, the collection of servers (and server-clusters) that are part of (members) the Service Integration Bus and then there is another collection of servers that comprise the server cluster. A ***cluster member*** will refer to the relationship between a server and the server cluster and a ***bus member*** will refer to the relationship between the server or server-cluster and the service integration bus.



When a application server cluster is added as a member of the Service Integration Bus the Message Engine is created for the cluster using the active/standby pattern by default.

This configuration is appropriate when the goal is *high availability*, and when there must always be a messaging engine available. This capability is also referred to as *failover*.

The drawback with this configuration is that there is only one queue and database table for the cluster. This could become a bottleneck. If this happens, the only way to get more throughput is to add a server on a faster computer with more memory.

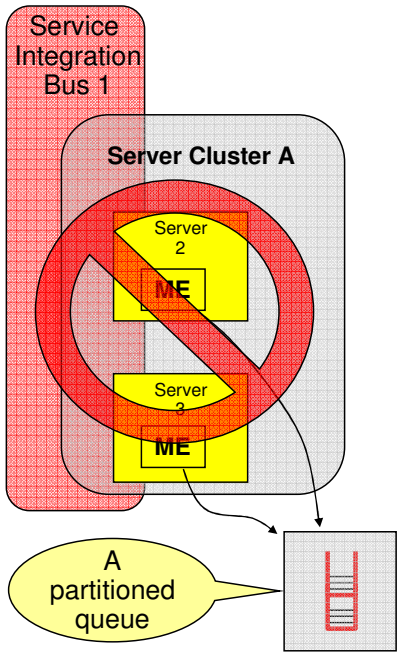
It is possible to create a server cluster that has computers with different levels of service, some may be faster than others. In this situation the server with the highest capacity can be designated as the primary and the other servers can be the standbys.

Sidebar:

When a cluster has servers that have different capacities, this is referred to as a mixed configuration. The applications running on the servers must all be the same but the configuration information can be unique for each server.

Messaging and scalability

- The logical approach to increasing throughput through the Message Engine would be to add more Message Engines to the server cluster
- Multiple Message Engines can be configured for a given bus using an HA Manager *N of M* policy
- JMS destination is *partitioned* across the Message Engines within the cluster
  - Constrains the kind of applications that can be run
    - Message order is random
    - Application cannot predict which partition of the queue a message will come from
    - Orphan messages in a failover situation.
    - No affinity to cluster members
      - Impacts stateful messages



When you consider how to increase messaging throughput, it might seem that you should add more Message Engines to the server cluster.

The drawback with this approach is that it places constraints on the kind of applications that can be run.

•When a user of the queue reconnects to the queue it may be routed to a different partition, so messages may not be processed in the order that they were placed on the queue.

•The message consumer will not be able to determine which partition a message is coming from, therefore it will need to retrieve the messages from all of the servers.

•The order of the messages is indeterminant.

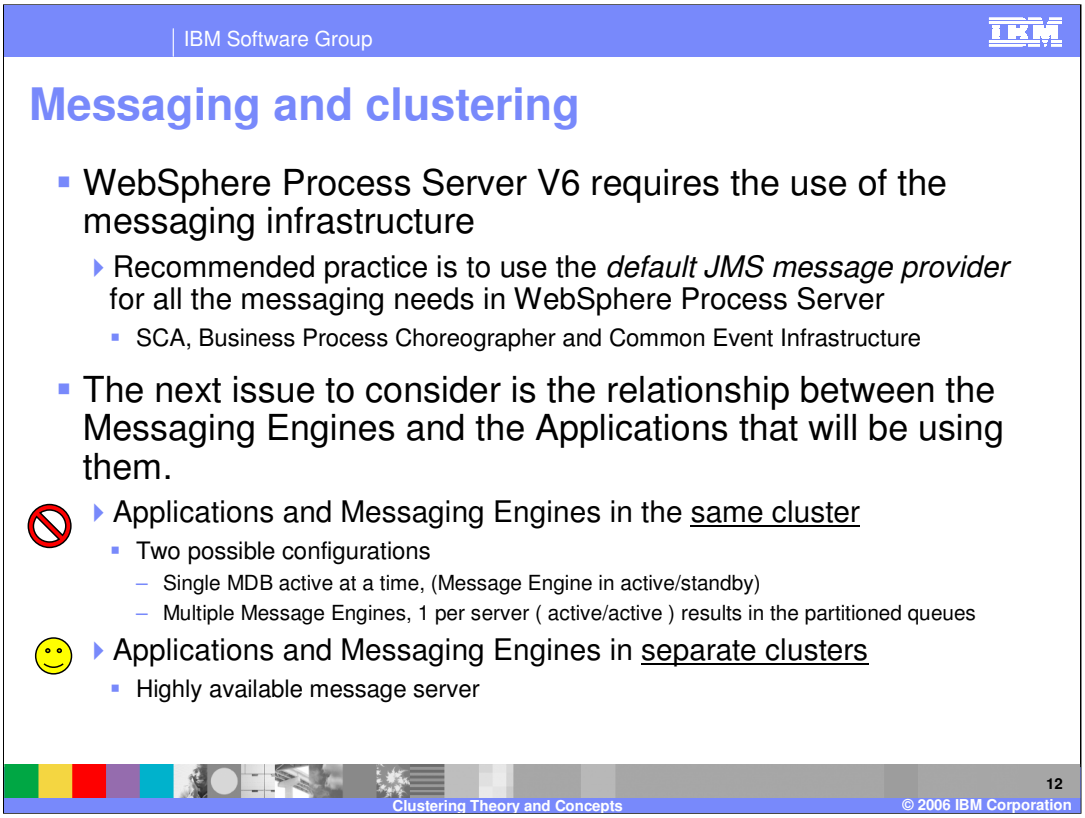
•An application may not rely on the assumption that a message that has been put to the queue will still be there on the next connection.

•When a cluster member fails, the messages in the associated queue will not be available until that server comes back online; they are orphaned.

<click animation here>

Because of the constraints imposed on the application, this topology is not recommended for general use.

Note: the pub/sub configuration is a special case where affinity to the queue can be obtained, but this is not applicable in the more general case used by the WebSphere



The default JMS message provider is the one that comes with WebSphere Application Server V6.

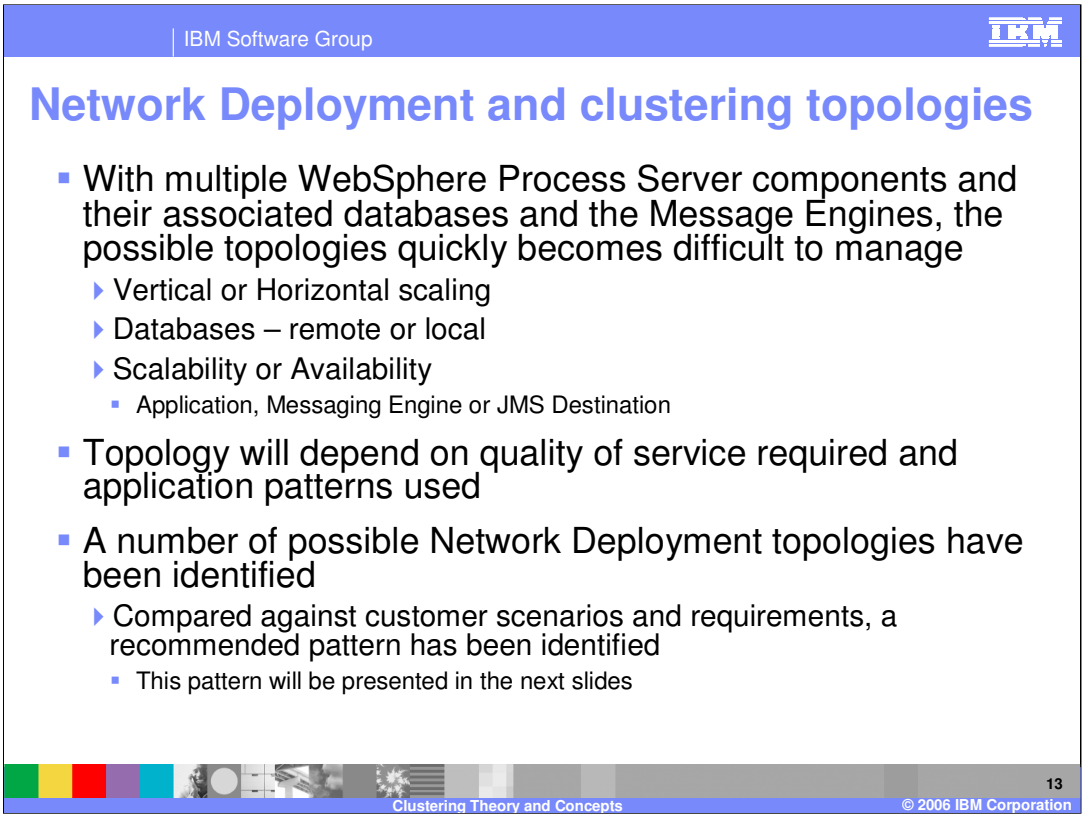
It is possible to use other JMS providers for Business Process Choreographer and the Common Event Infrastructure. Since the Service Component Architecture requires the default JMS message provider it is recommended that the default JMS message provider be used for the Service Component Architecture and the Common Event Infrastructure requirements as well.

Assuming that the applications are to be clustered.

Grouping the applications and the Message Engine into the **separate clusters** allows for greater flexibility in configuration and tuning and is the **recommended approach**.

When you consider the case with the applications and the Message Engine in the same cluster there are 2 alternatives to consider.

1. Active/Standby
    - Is the default provided when a cluster is added to an Service Integration Bus as a bus member.
    - There is one message driven bean per Message Engine and only one Message Engine is active at a time therefore this will limit the throughput of the applications.
2. Active/Active



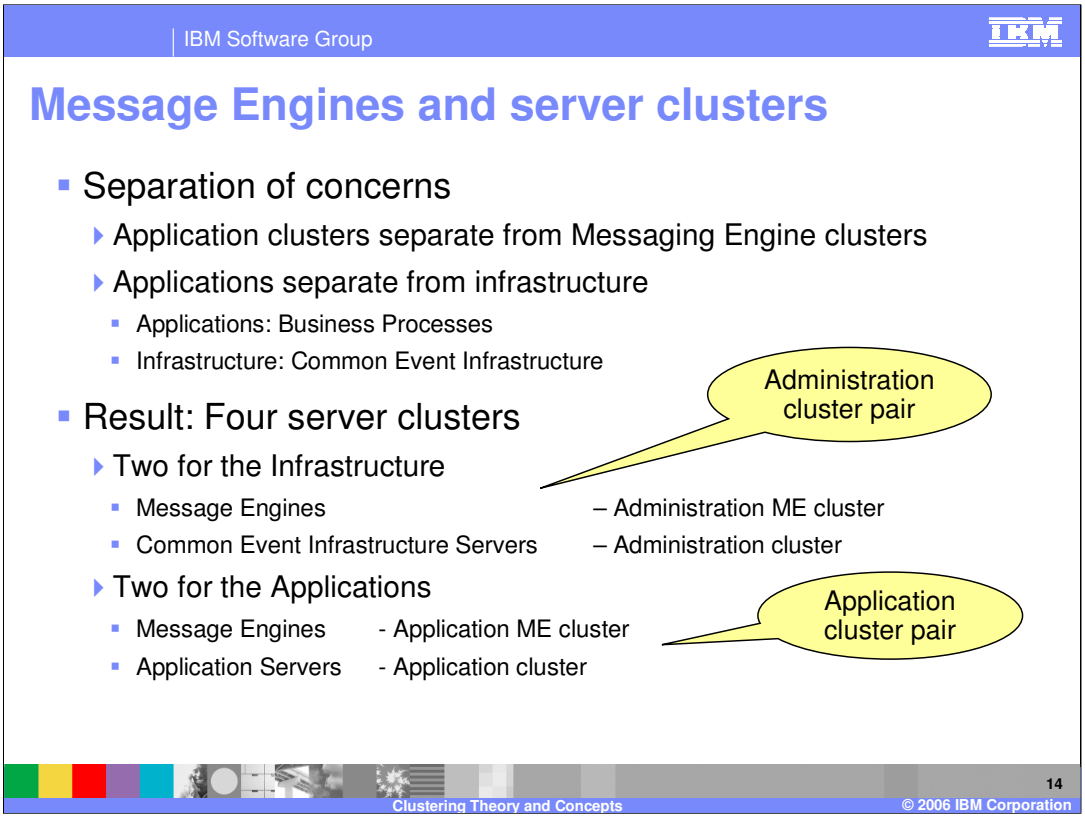
There are many facets to consider when deciding how to configure the Cell topology. Is the goal high availability or scalability or some combination of both? If so, which one is most important based on the requirements of the application?

Will the databases be remote or local?

Will there be several databases or will tables be combined into one or two databases, where it is feasible?

Ultimately it will depend on the administrative processes already in place and the quality of service required by the applications. There are many ways to configure the cell topology and WebSphere provides the flexibility to do what is needed.

Based on typical customer requirements for scalability and high availability, a ‘suggested’ topology will be presented as a vehicle for discussing how to configure a complex WebSphere Process Server topology.



The topology being presented here is recommended as the most likely topology based on the quality of services available to the applications. It will be used to demonstrate the steps for setup and configuration of a clustered topology. Once you understand the principles and the steps, you can easily develop alternative solutions.

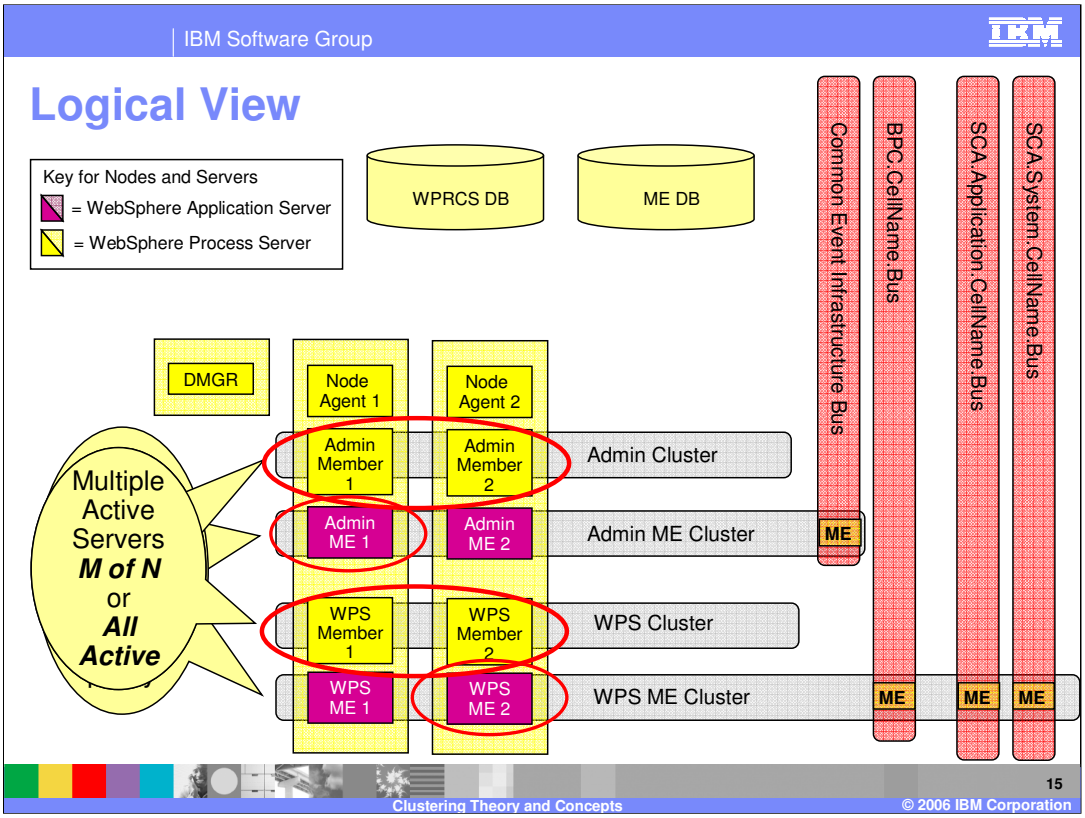
The Common Event Infrastructure will likely be an integral part of all WebSphere Process Server applications and it is therefore considered to be an infrastructural component of the system.

The principle of “Separation of Concerns” is used as a guide for deciding how to partition the system and distribute the function.

Based on the previous discussion regarding the message engine and the applications, there are message engine clusters and application clusters.

Considering functional boundaries, you can partition the system based on whether it is an administrative function or functionality associated with the end-user application. An example of a administrative function would be the Common Event Infrastructure (CEI) or the Business Rules Manager. Both of these are WebSphere Process Server components and therefore part of the overall infrastructure.

Using this criteria, the resulting topology starts with four server clusters. As the system grows there may be additional application cluster pairs, but only one administration cluster pair would be necessary.



Notice the key in the upper left. There are two different kinds of servers in each node. The three nodes, deployment manager, Node 1, and Node 2, are represented by the yellow rectangles.

The service integration buses are represented by the pink vertical columns on the right and the server clusters are the gray horizontal bars, enclosing the application servers and sometimes intersecting with the Service Integration Buses.

The Common Event Infrastructure Servers in the AdminCluster are associated to the Message Engine in the AdminMECluster through the JMS destination configurations.

Likewise the Application Servers in the WPSCluster are associated to their messaging engine in the WPSMECluster through their JMS destination configurations.

The **ME** (Message Engine) shown on the Service Integration Buses represent the association between the server cluster and the Service Integration Bus.

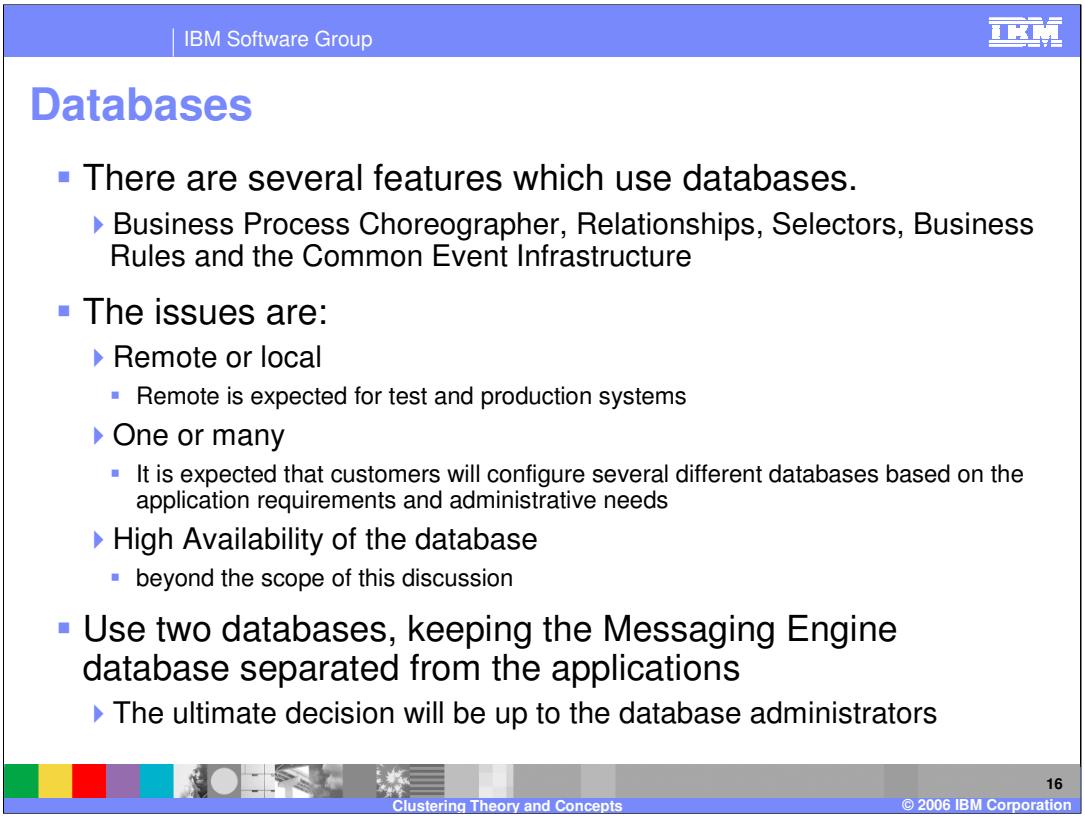
<click for animation>

Only one Message Engine is active in a server cluster at a given time. The WebSphere Workload Manager decides which Message Engine is active based on the HA Manager policy which is configured as 1 to N.

<click for animation>

Whereas in the application clusters the servers are configured so that they can all be active at the same time.

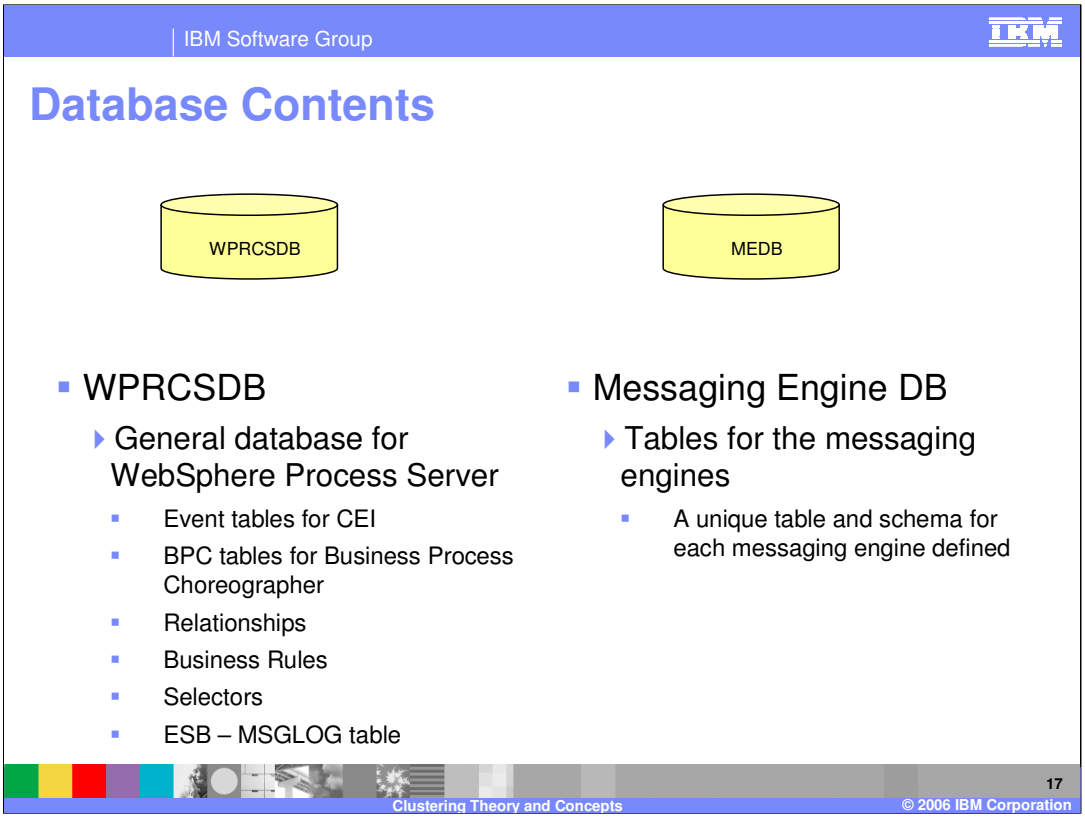
The SCA System and Application Buses are required for the basic SCA functionality.



With WebSphere Process Server there are quite a few databases introduced.

In a typical production environment there will be several databases, the ESBLogMedDB, the EVENTDB, the BPEDB, the WPRSCDB, the MEDB, along with the end-user application databases. It is expected that they will be remote, residing on database server, managed by database administrators.

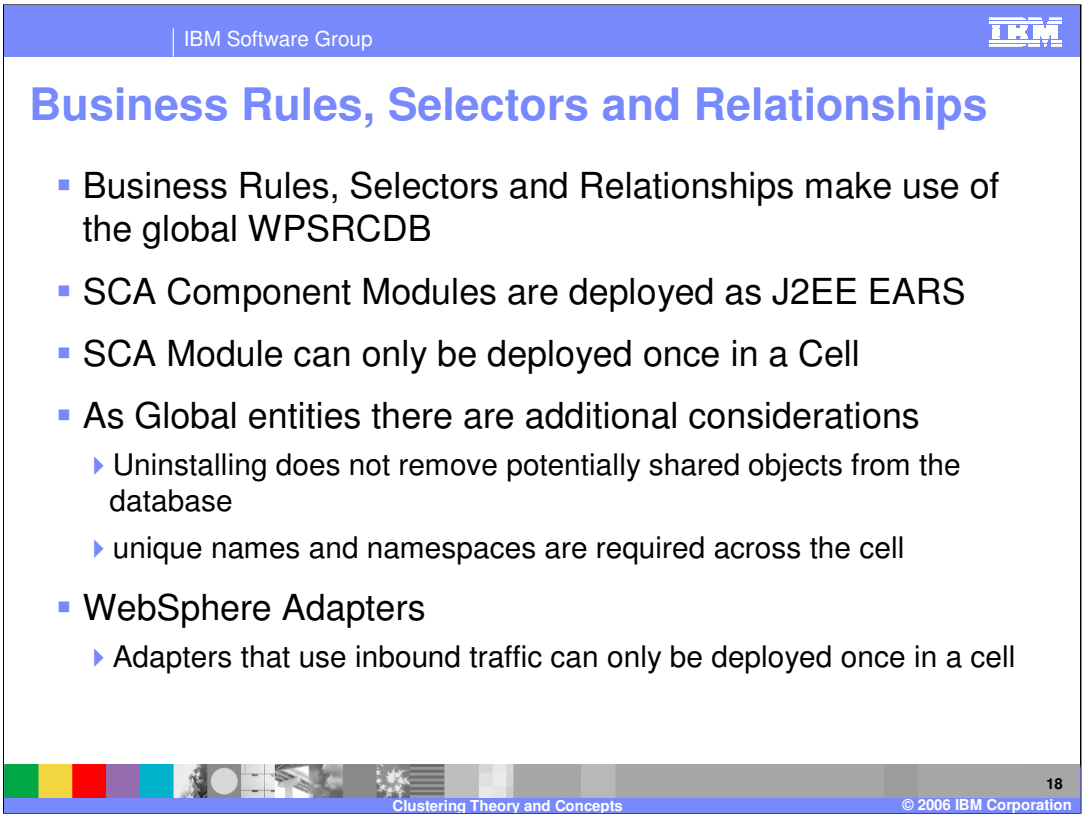
For the configuration being presented there will be 2 databases. The tables for the EVENTDB, ESBLogMedDB and BPEDB are created in the WPRCSDB, keeping the Message Engine datastores isolated in the MEDB. This decision is made for convenience only.



Note: create the WPRCSDB before creating the Deployment Manger node (DMGR)

The decision to use two databases rather than many simplifies things for demonstration purposes.

Creation of the databases and tables will be covered in detail as part of the tutorial.

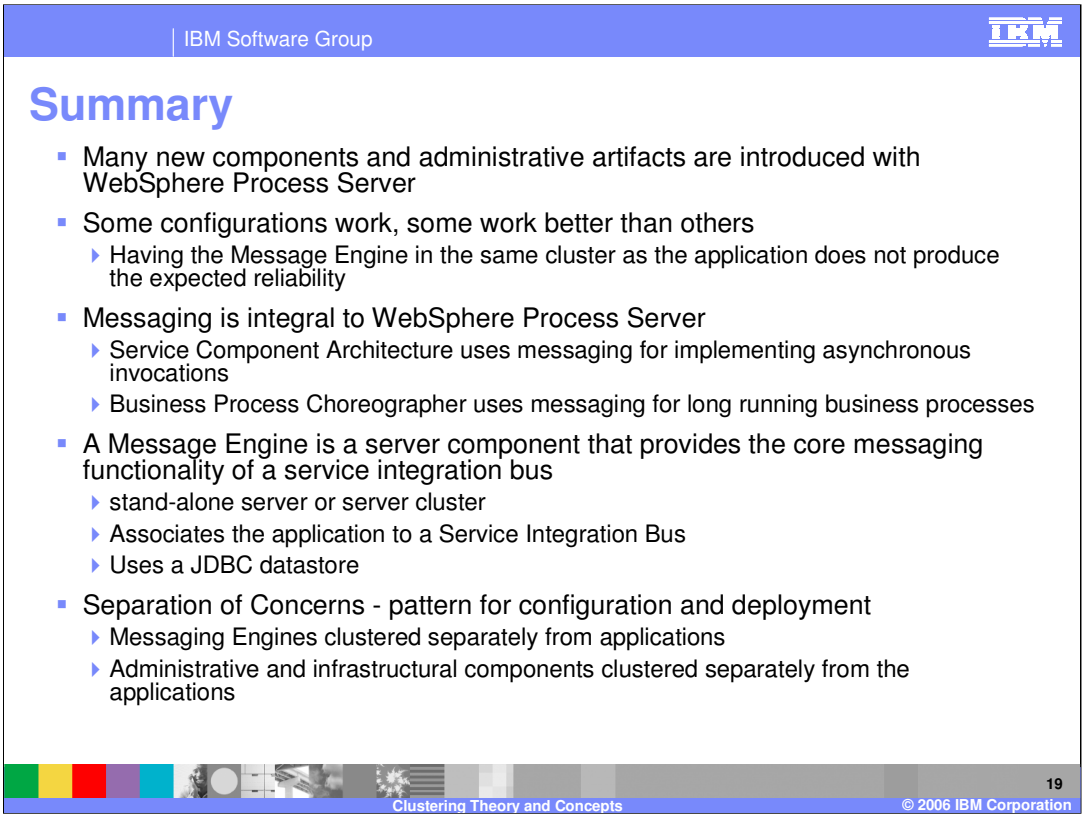


A few considerations regarding the use of other Service Component Architecture components which may impact the design of the cell, the databases and application server clusters.

The global nature of the WebSphere Process Server components presents challenges when deploying and redeploying applications.

The potential for name clashes prohibits duplicate names in many situations and the use of common components means that an entity may not be removed upon uninstall as expected, causing a deployment failure on the next deployment cycle.

WebSphere Adapters that are used for inbound traffic can only be deployed once in a cell.



Moving to WebSphere Process Server V6 introduces a lot of new parts and pieces to consider and work with.

There are many combinations for deployment and configuration, some work, some work better than others and some just don't work in a realistic manner.

The key element in business process applications and all WebSphere Process Server applications is messaging. Messaging lies at the heart of everything and when it comes to creating applications that leverage the WebSphere clustering capabilities the Message Engine is the component that must be considered first.

The deployment pattern presented here has 2 features. First separating the Message Engine cluster from the application cluster. This is imperative for reliable messaging. The second feature presented as part of this pattern is the separation of the administrative or infrastructural components of the system from the user application components. This second feature is optional and may not be necessary in all cases. For any enterprise deployment, it is the approach to use.

Trademarks, Copyrights, and Disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	CICS	IMS	MQSeries	Tivoli
IBM(logo)	Cloudscape	Informix	OS/390	WebSphere
e(logo)business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product and service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.