Online Upgrades Become Standard

Louise Moser

Eternal Systems, Inc.

BUILDING SYSTEMS THAT RUN FOREVER

Introduction

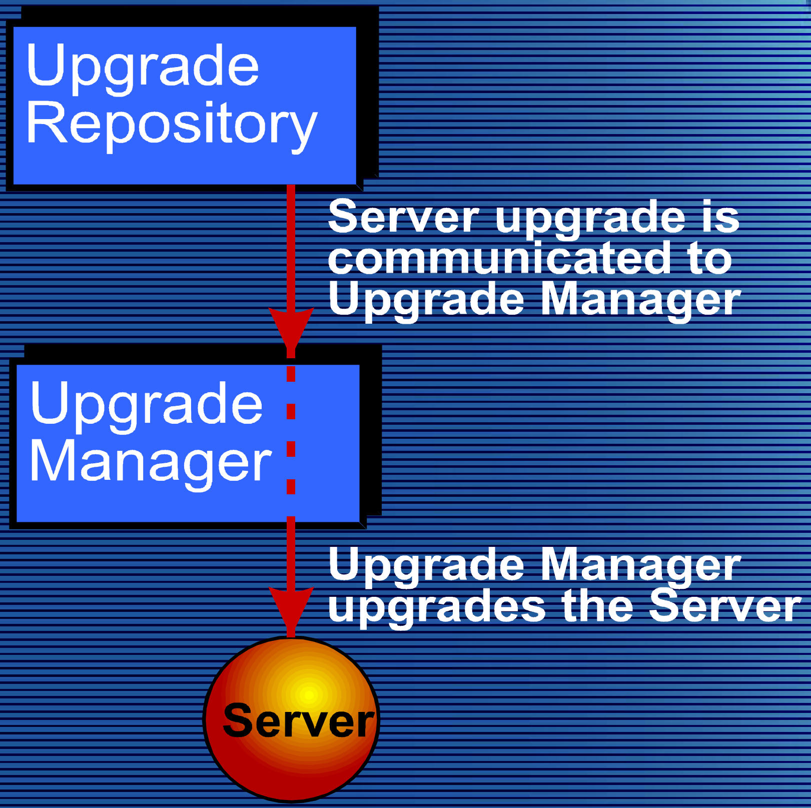
- Many computer systems must provide **continuous service**, without interruption or suspension of service, over a long lifetime
- Such systems include large complex systems and small embedded systems
  - Financial
  - Supply chain
  - Telecommunications
  - Industrial control
  - Transportation
  - Defense

Introduction

- Application deployers must be able to upgrade their systems by replacing individual software and hardware components
- However, many systems cannot be taken out of service to perform an upgrade
- Often, it is difficult to take part of a system out of service for upgrading, while other parts of the system continue to operate
- Therefore, **online upgrades** are needed

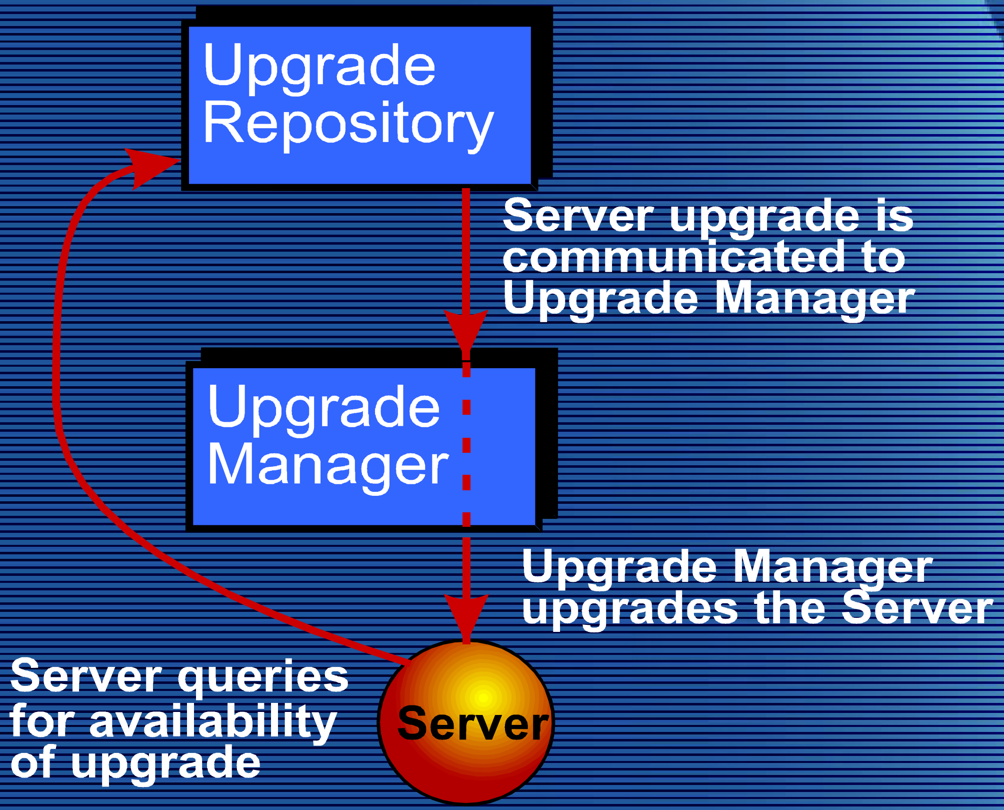
Online Upgrade Scenarios

Pushed Upgrade


Upgrade Repository
Server upgrade is communicated to Upgrade Manager
Upgrade Manager
Upgrade Manager upgrades the Server
Server

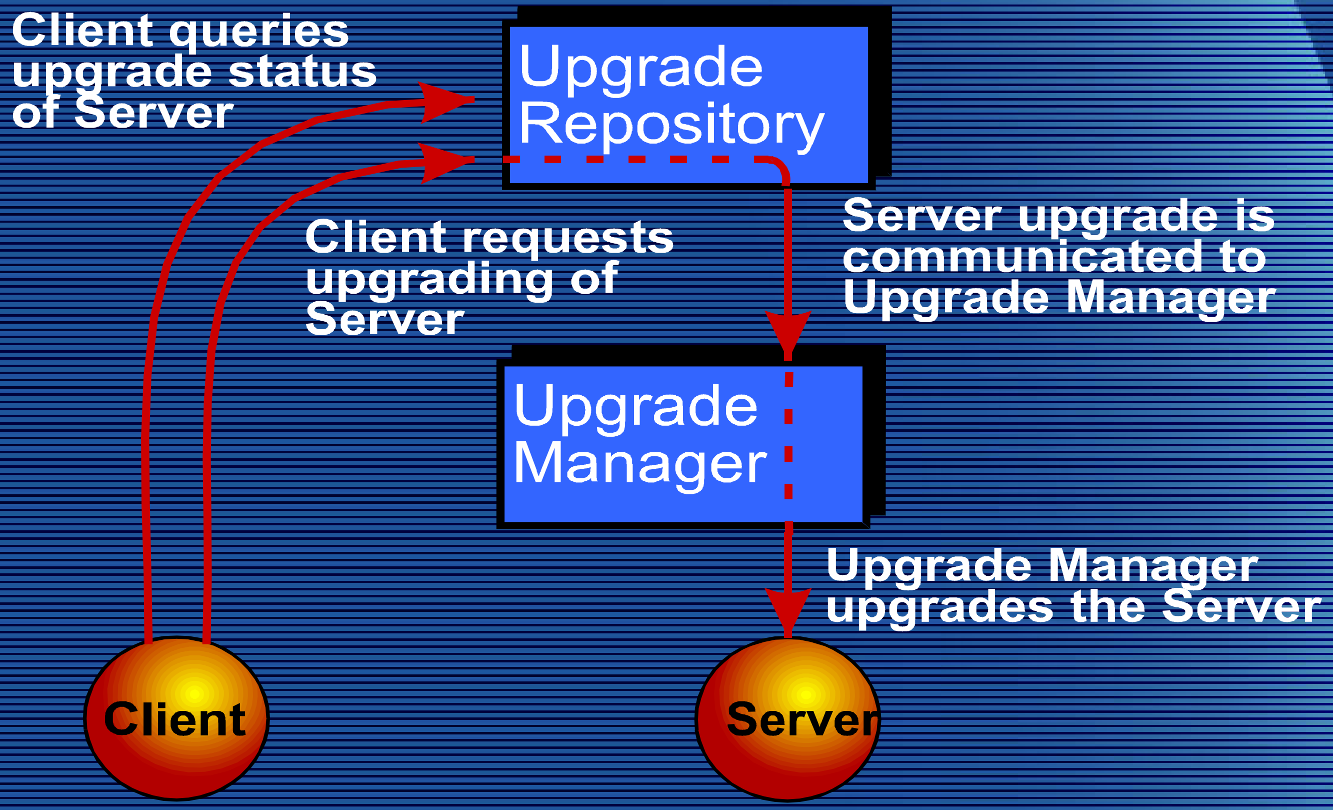
Online Upgrade Scenarios

Pulled Upgrade


Upgrade Repository
Server upgrade is communicated to Upgrade Manager
Upgrade Manager
Upgrade Manager upgrades the Server
Server queries for availability of upgrade
Server

Online Upgrade Scenarios

Smart Clients


Client queries upgrade status of Server
Upgrade Repository
Client requests upgrading of Server
Server upgrade is communicated to Upgrade Manager
Upgrade Manager
Upgrade Manager upgrades the Server
Client
Server

The Need for Standards

- In the past, online upgrade technology was
  - Proprietary
  - Difficult to use
  - Prone to fiascos
  - Not portable
  - Not interoperable
- Today, industrial standards for online upgrades are being adopted and implemented

Industrial Standards

- Object Management Group
  - Online Upgrades mars-2002-06-11
  - CORBA distributed object applications
- Java Community Process
  - JSR 117 Continuous Availability
  - EJB/J2EE application servers
  - Enterprise applications
- Service Availability Forum
  - Embedded applications, data/telecom

Intent of CORBA Standard

- Initial step
  - Provides basic functionality for interoperable and portable online upgrades
- Building block
  - Basic online upgrade service
  - More sophisticated online upgrade services can be built on top of this basic service

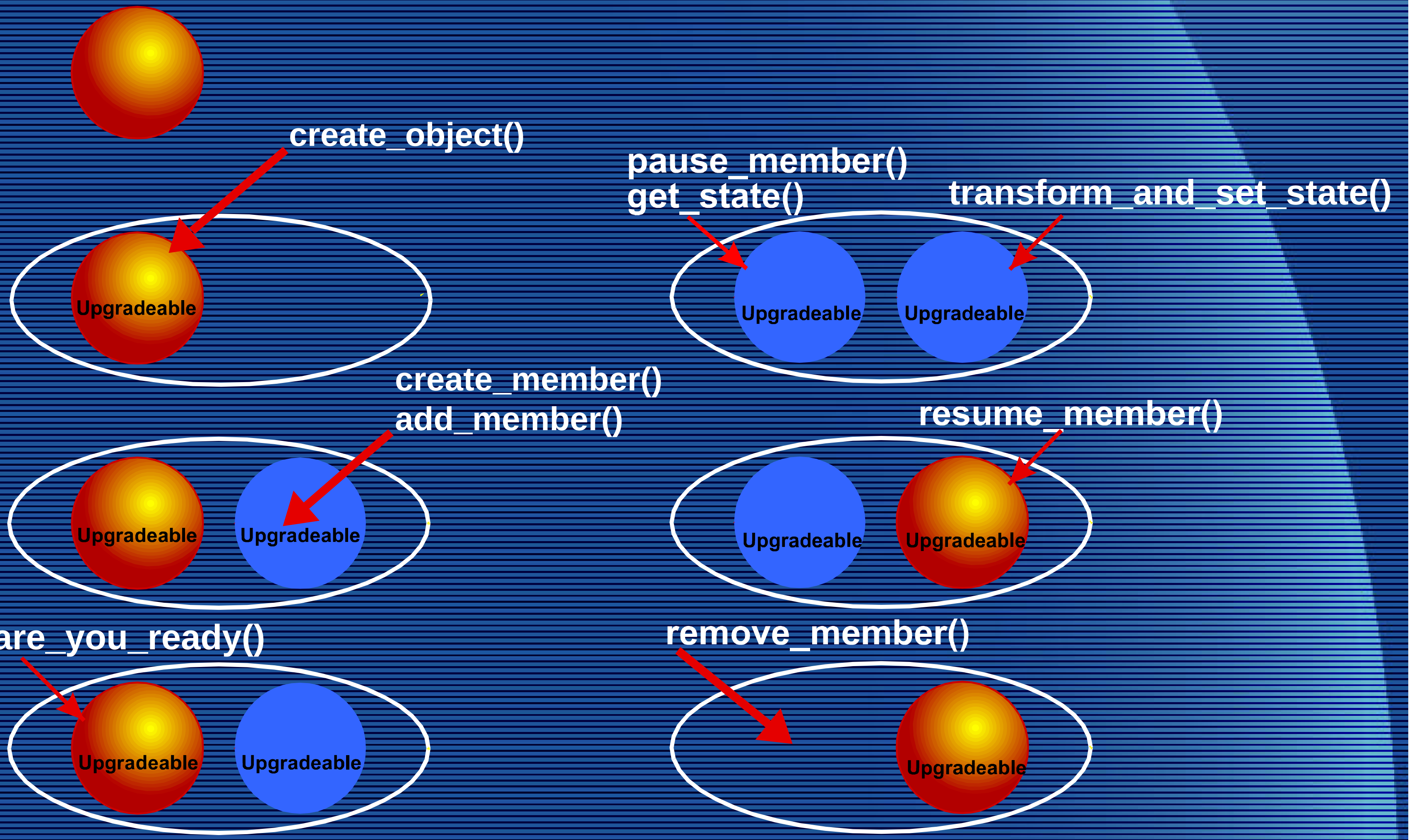
Objectives

- Upgrade individual objects by changing their implementations, but not their interfaces
- Pause an object, so that it can be upgraded
  – Allow it to reach a safe and quiescent state
  – Transfer state from the old instance to the new instance
- Continue service using the new instance without risk that messages are lost, incorrectly ordered, or processed twice

Objectives

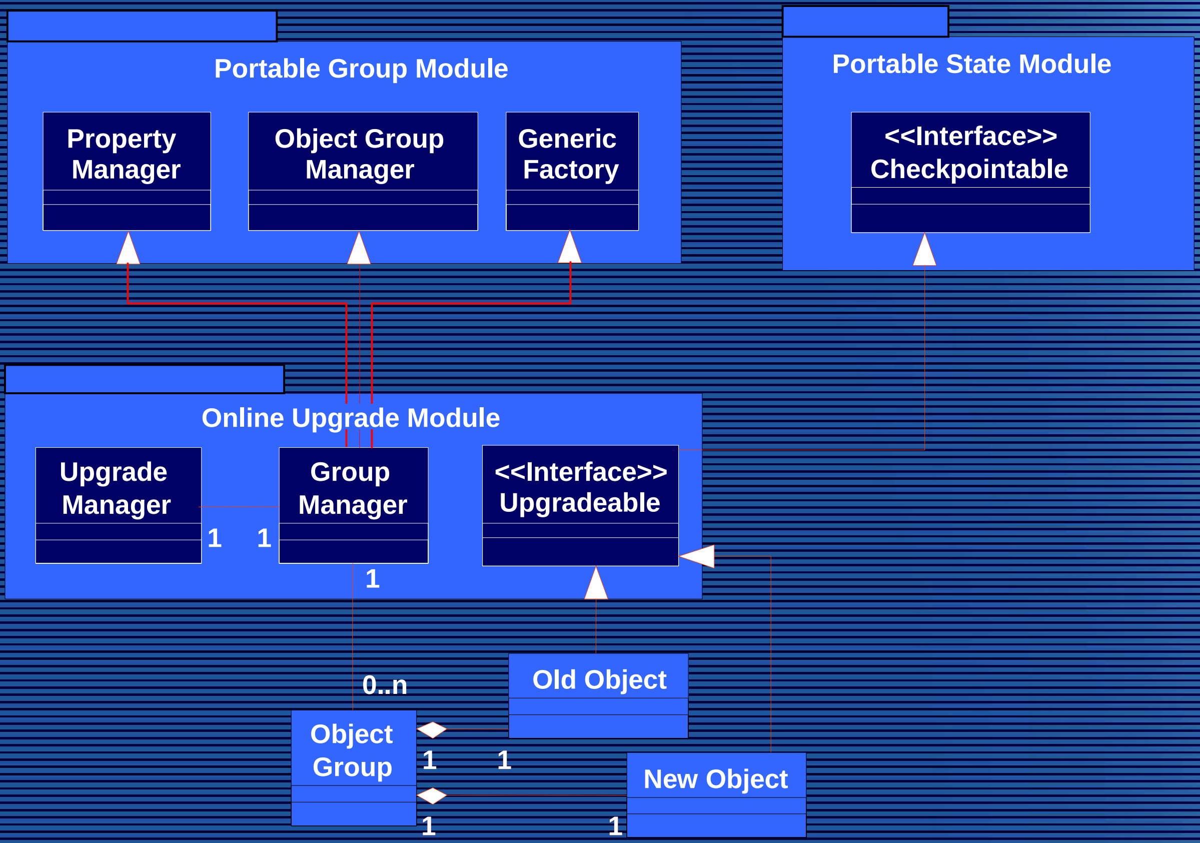
- Undo an upgrade
  - Rollback an upgrade, before the new instance becomes operational, if part of the upgrade fails
  - Revert from a new instance to an old instance, if the new instance does not operate correctly
- Upgrade collections of objects by allowing the application to commit and rollback the upgrades explicitly

Stages of an Upgrade

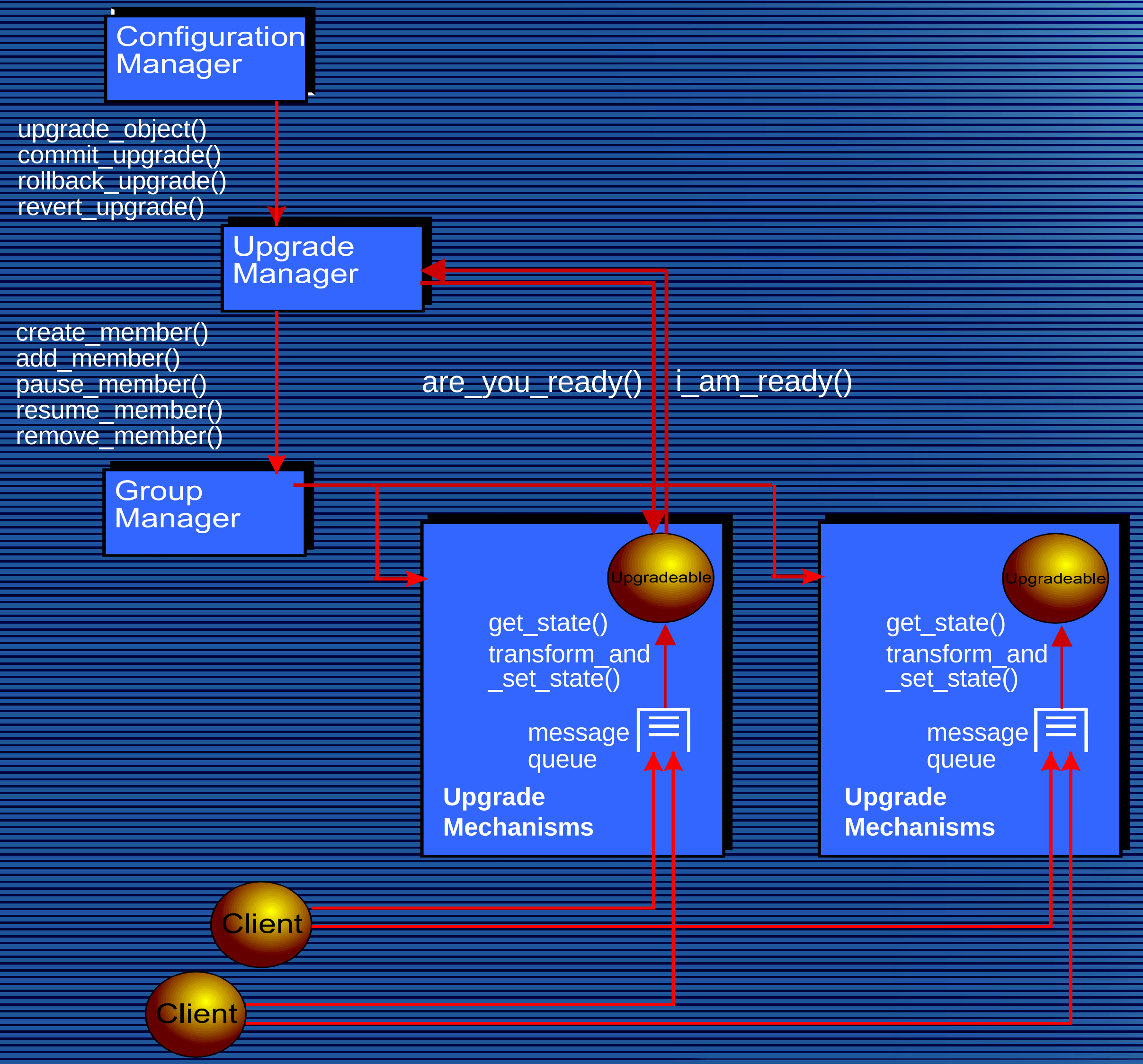


BUILDING SYSTEMS THAT RUN FOREVER

Architectural Overview



Architectural Overview



Upgrade Manager

- Provides methods to
  - Prepare an object for upgrading
  - Perform the upgrades of one or more objects
  - Rollback upgrades of objects
  - Revert an object from its new implementation to its old implementation
- Invokes methods of the Group Manager
- Invokes methods of the Upgradeable interface that the application objects inherit

Upgrade Manager Methods

- upgrade_object(): Upgrades the object defined in its parameter list
  - object_group: Reference of the object-to-be-upgraded
  - type_id: Type of the object-to-be-upgraded
  - the_location: Location at which the upgraded implementation is to be instantiated
  - the_factory: Factory that is to be used to create the instance of the upgraded implementation
  - app_ctrl_commit: Boolean that allows the application to commit the upgrade explicitly or to delegate that responsibility to the Upgrade Manager

Upgrade Manager Methods

eternal SYSTEMS

- commit_upgrade(): Allows the application to commit the upgrade explicitly
- rollback_upgrade(): Allows the application to rollback the upgrade before it is committed
- revert_upgrade(): Allows the application to revert the upgraded implementation to the old implementation after it is committed

Upgradeable Objects

- An upgradeable object must inherit the Upgradeable interface
- Upgradeable interface extends the PortableState module
- PortableState module defines the Checkpointable interface
- Checkpointable interface defines get_state() and set_state() methods

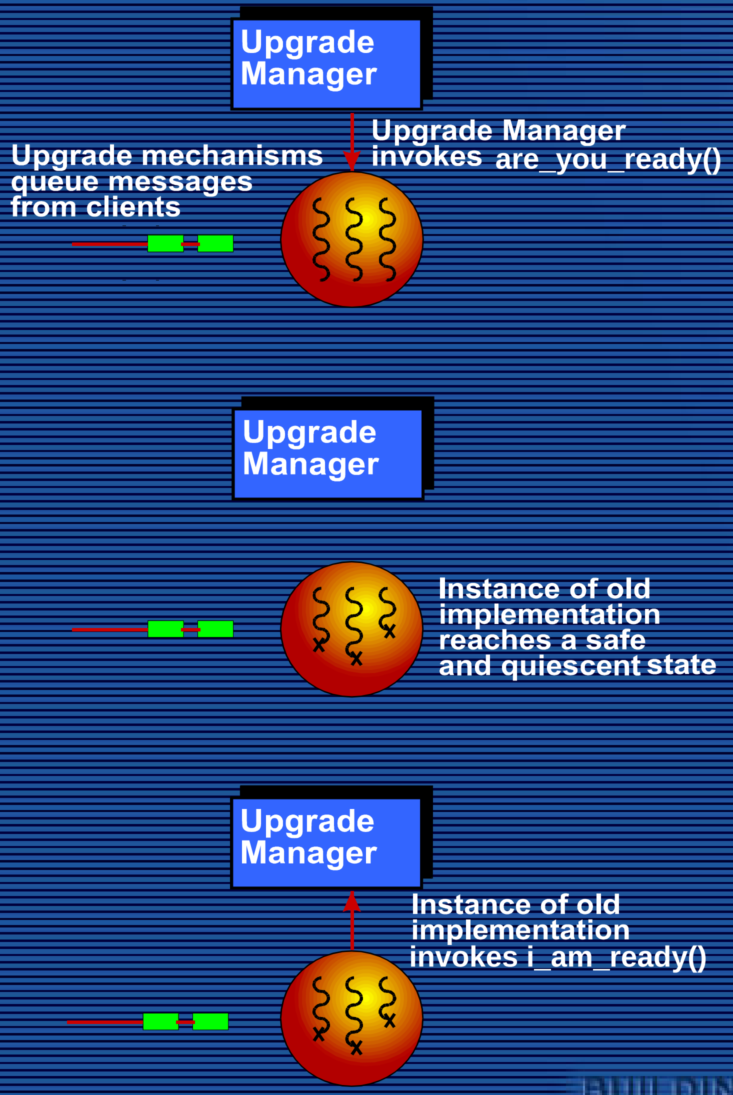
BUILDING SYSTEMS THAT RUN FOREVER

Upgradeable Methods

In addition to the Checkpointable methods, the Upgradeable interface defines the following method:

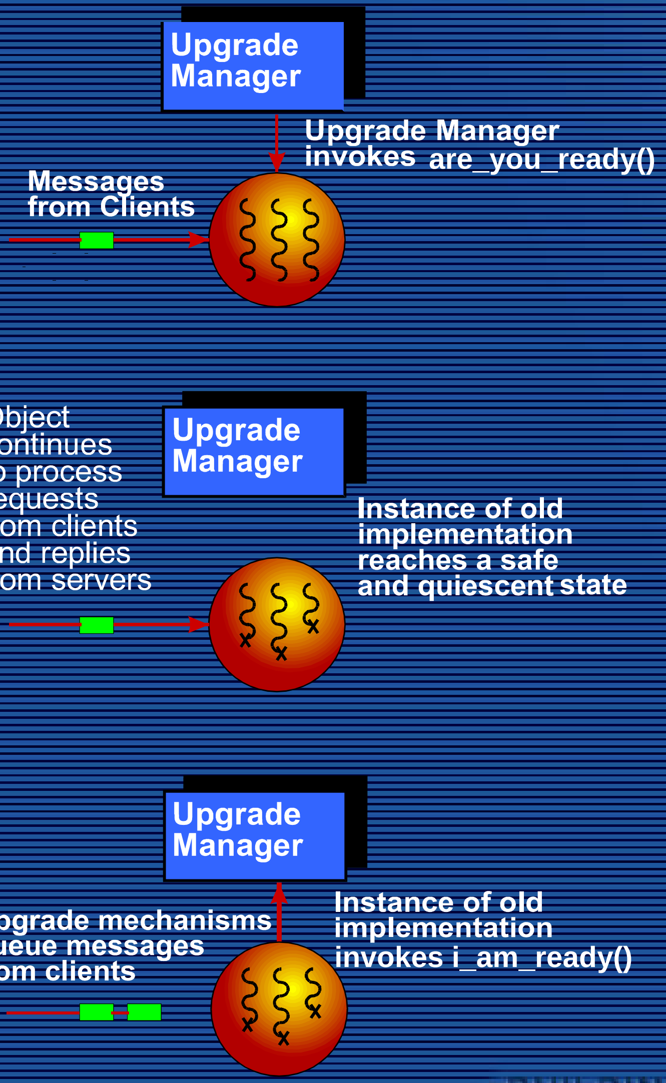
- are_you_ready()
  - Invoked by the Upgrade Manager on an upgradeable object to query the object whether it is ready to be upgraded
  - The object must be in a safe and quiescent state to be upgraded
  - If it is in a safe and quiescent state, the object invokes i_am_ready() with the ready parameter equal to true

are_you_ready() no callbacks


Upgrade Manager
Upgrade Manager invokes are_you_ready()
Upgrade mechanisms queue messages from clients
Upgrade Manager
Instance of old implementation reaches a safe and quiescent state
Upgrade Manager
Instance of old implementation invokes i_am_ready()

BUILDING SYSTEMS THAT RUN FOREVER

are_you_ready() with callbacks

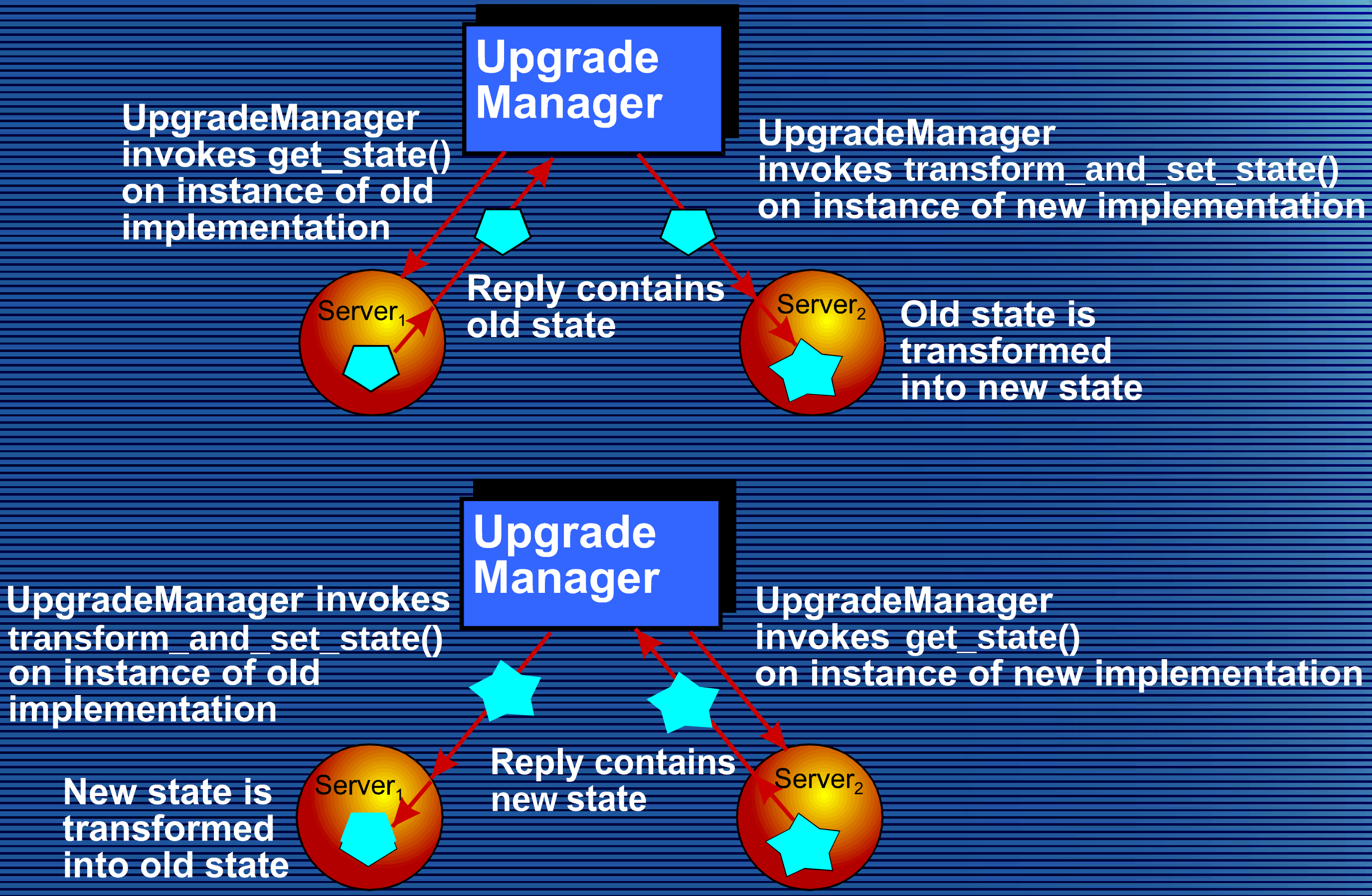

Upgrade Manager
Upgrade Manager invokes are_you_ready()
Messages from Clients
Object continues to process requests from clients and replies from servers
Upgrade Manager
Instance of old implementation reaches a safe and quiescent state
Upgrade Manager
Upgrade mechanisms queue messages from clients
Instance of old implementation invokes i_am_ready()

BUILDING SYSTEMS THAT RUN FOREVER

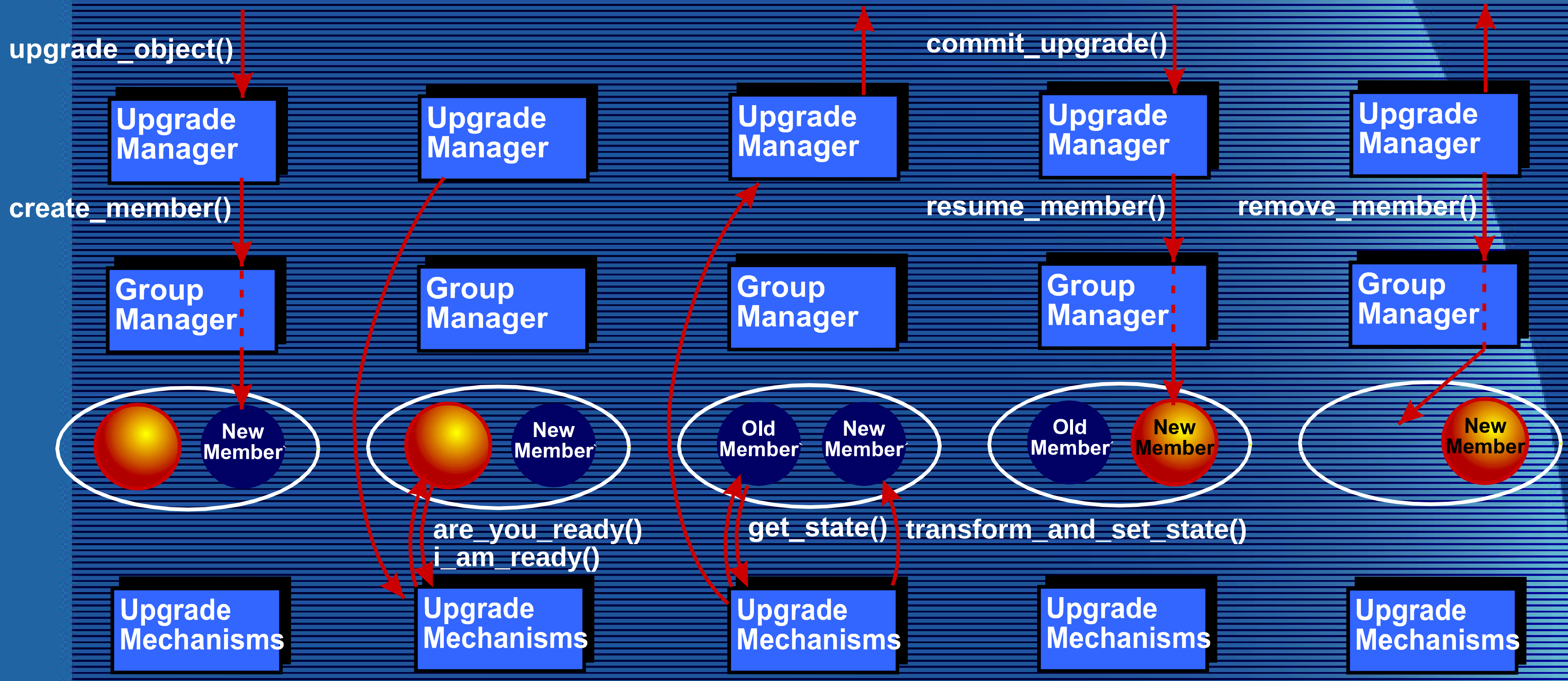
Upgradeable Methods

- transform_and_set_state()
  - Invoked by the Upgrade mechanisms on an instance of the new implementation
  - Transforms the state of the instance of the old implementation into the state of the instance of the new implementation, providing values for new attributes of the new implementation
  - Assigns the state to the new implementation

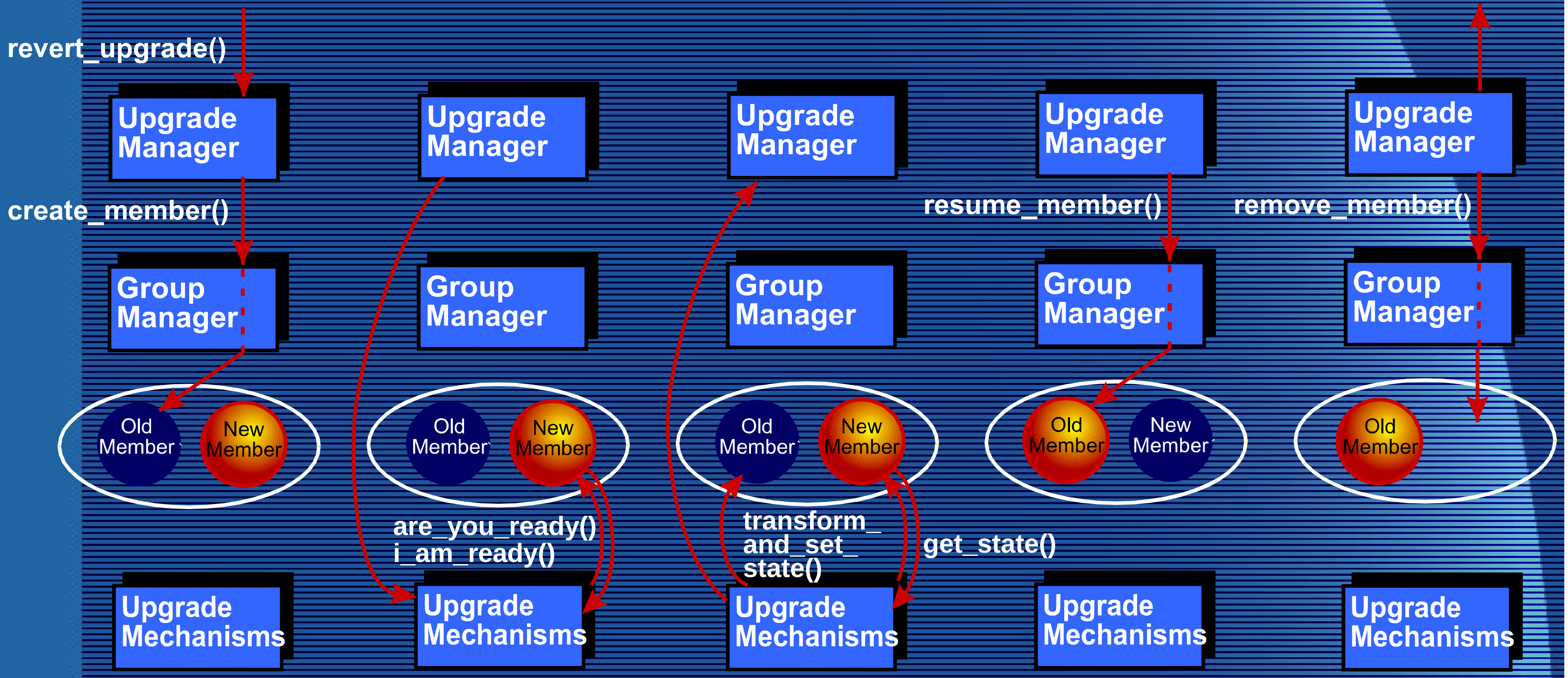
State Transfer


Upgrade Manager
UpgradeManager invokes get_state() on instance of old implementation
UpgradeManager invokes transform_and_set_state() on instance of new implementation
Server$_1$
Reply contains old state
Server$_2$
Old state is transformed into new state
Upgrade Manager
UpgradeManager invokes transform_and_set_state() on instance of old implementation
UpgradeManager invokes get_state() on instance of new implementation
New state is transformed into old state
Server$_1$
Reply contains new state
Server$_2$

Use Case upgrade_object()



Use Case revert_upgrade()



Extensions

The CORBA standard can be easily extended with capabilities to

- Upgrade the interfaces of an object
- Allow an object to initiate its own upgrading
- Operate instances of the old implementation and the new implementation concurrently
- Revert to an instance of a prior implementation other than the immediately prior implementation

Extensions

The CORBA standard can be further extended with capabilities to

- Test a new implementation
- Define version numbers for implementations
- Determine when an upgrade is available and when it should be applied
- Determine the security or validity of an upgrade

Conclusion

eternal SYSTEMS

With the adoption of industrial standards for online upgrades, commercial implementations are becoming available

But, the work is not yet finished and includes

- Extensions to provide more functionality
- Features that have not yet been considered
- Uses that have not yet been addressed

Contact Information

Louise Moser

Eternal Systems, Inc.

5290 Overpass Road, Building D

Santa Barbara, CA 93111

805-696-9051 X223

moser@eternal-systems.com

www.eternal-systems.com

BUILDING SYSTEMS THAT RUN FOREVER