Gold Rush : Mobile Transaction Middleware with JAVA Object Replication

Presented By Goutham Rao

University Of Pennsylvania

What we will be talking about

An approach taken to equip mobile clients with access to a central database, even in a disconnected environment.

"Gold Rush is a middleware supporting the writing of JAVA applications that resides on an intermittently connected mobile client and access an enterprise database on a central server"

Flow of the presentation

- Motivation. Scenarios in mobile computing.
- Off-line transactions : what they mean, the off-line transaction model and problems encountered.
- Introduction of Gold Rush as an off-line transaction based system.
- Related approaches taken in providing mobile clients with access to enterprise data.

Flow of the presentation

- The Gold Rush Approach and architecture.
  - General Overview.
  - The Gold Rush view of off -line transactions
    - how Gold Rush deals with relational databases and database objects.
  - The Gold Rush three tier architecture and its basic components.

Flow of the presentation

- Detailed discussion of the Gold Rush approach.
  - Correspondence between relational databases and database objects.
  - Persistent client store.
  - Off-line transactional semantics
  - Reducing data traffic between client and server.

Motivation

- We want to be able to provide mobile users with access to computing resources at the location where they can be used most effectively.
- We cannot always expect applications to have a connected access to a central database.
- Communication links are characterized by :
  - *Low bandwidth.*
  - *High cost of connections : any time - any where wireless links are expensive.*
  - *Frequent disconnection*
  - *radio devices drain batteries quickly.*

Motivation

- Application connectivity requirements to the data source may vary.
  - *Some applications require high degrees of connectivity even while mobile. Example : applications used by stock traders.*
  - *In contrast some applications mainly work in a disconnected (off line) mode connecting to the data source occasionally for updates (re-integration).*
- When connection requirements are less stringent, it is possible to perform the transactions on a local copy of the data in a disconnected mode.
  - *Such a technique is called an off-line transaction.*
  - *We later need to update the data source with the changes made my the mobile computer.*

Scenario

- *Financial planner downloads part of an enterprise database on his local computer.*
- *Performs transaction while disconnected from the data source at the clients location.*
- *Reconnects at the end of the day to upload changes he made to the clients profile.*

Motivation

- Such disconnected transactions may not be used in every application
  - Applications where the central database changes rapidly and where the latest information is always needed for a transaction.
    - *Applications used by stock traders who need continual real time access to the stock market data is an example.*
  - Applications where there is a high degree (or potential) of conflict in the transactions performed by different mobile clients.
    - *An example is an application which updates the value of a shared counter at every transition will conflict at every transaction with other clients.*

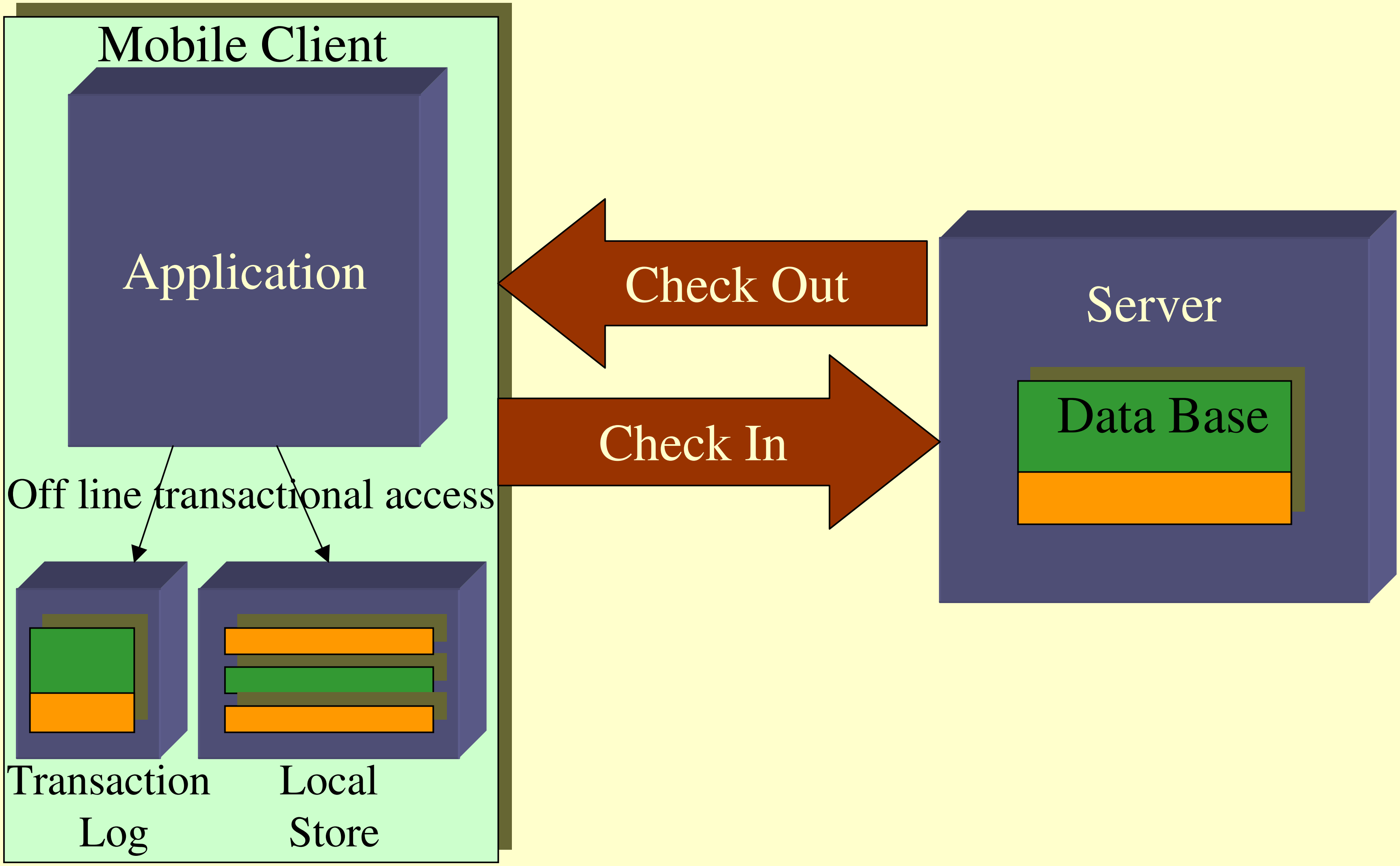
Motivation

- Some applications may be redesigned to avoid these problems.
  - *The shared counter problem may be solved if the applications specify a delta in the counter rather than the actual value.*

Off-line transaction model

- A transaction is a set of read/write operations on data.
- An off-line transaction is a set of read/write operations by a mobile client on a local copy of some portion of the database.

Off-line transaction model



Off-line transaction model

- Recap of the flow of this model
  - Check out: Partial replication of the data along with integrity constraints.
  - Access: Off-line transactional access with read/write information logged.
  - Check in: Re-integration of off-line transactional data with the main database.
  - Conflict handling: Detection and resolution of conflicts.

Off-line transaction model

- Off-line transactions have to be integrated with the central database at a later time.
  - This is called a transaction replay.
  - This is handled by keeping a log of off-line transactions.
  - A “commit” is the submittal of a transaction to a server.
  - Since the replay happens at a later stage, it is called a “lazy commit approach”.

Off-line transaction model

- Why do we need such a model ?
  - We have seen where we may work in a disconnected (off-line) mode. We must see how to write such applications.
  - The off-line transaction model helps us understand (or specifies a model for) the writing of such applications.
  - As we will see, transactions help guarantee concurrency control, data integrity.
    - Transactions enforce integrity constraints and Non compliant transactions are aborted.
  - Such a model enables us to understand the scope of conflict and to re-integrate off-line data with the central database.

Off-line transaction model

- Problems are encountered with off-line transactions.
  - How do you write applications that run seamlessly both in connected and disconnected modes ? Can the same application be used ?
  - Which data should be replicated on the mobile client ?
  - How do you synchronize replicated data back to the server at a later stage ?
  - How might conflicts arise and how will they be resolved ?

Related approaches to allow mobile workers with access to central databases

- IBM's VisualAge for JAVA and Symantec's dbANYWHERE.
  - Designed for permanently connected clients.

Related approaches to allow mobile workers with access to central databases

- Other approaches for disconnected clients
  - Download a portion of the central database on the mobile client.
  - Mobile client may access the local database by traditional methods.
    - *example through JDBC/ODBC .*
  - Changes made to the local database are reconciled with the central database at a later stage through a replicator.

Related approaches to allow mobile workers with access to central databases

- Conflicts are handled during reconciliation.
  - Conflicting changes may be merged, tuples with an old time stamp may be dropped, etc.
- These replicators are often tightly coupled with the implementation of the database system in the client and server.
- Such an approach requires the client to host a small server to host the partial database.
- Various replication methods have been studied and reconciliation algorithms proposed.

Related approaches to allow mobile workers with access to central databases

- Comments on these approaches
  - A replicated database burdens the light weight client with a database server.
  - It is desirable to place queried data on the client and provide tools to work with this data.
  - It is desirable to bring the data base implementation to the server side.
    - *The interface between the application and the database is moved to the server side.*

Gold Rush Overview

- Gold Rush’s objective is to make enterprise data available to mobile clients.
- Gold Rush is a mobile transaction middleware providing mobile connectivity and data management.

“It is not sufficient to simply extend database query facility to the mobile client. There must be services for mobile data management as well”.

Gold Rush Overview

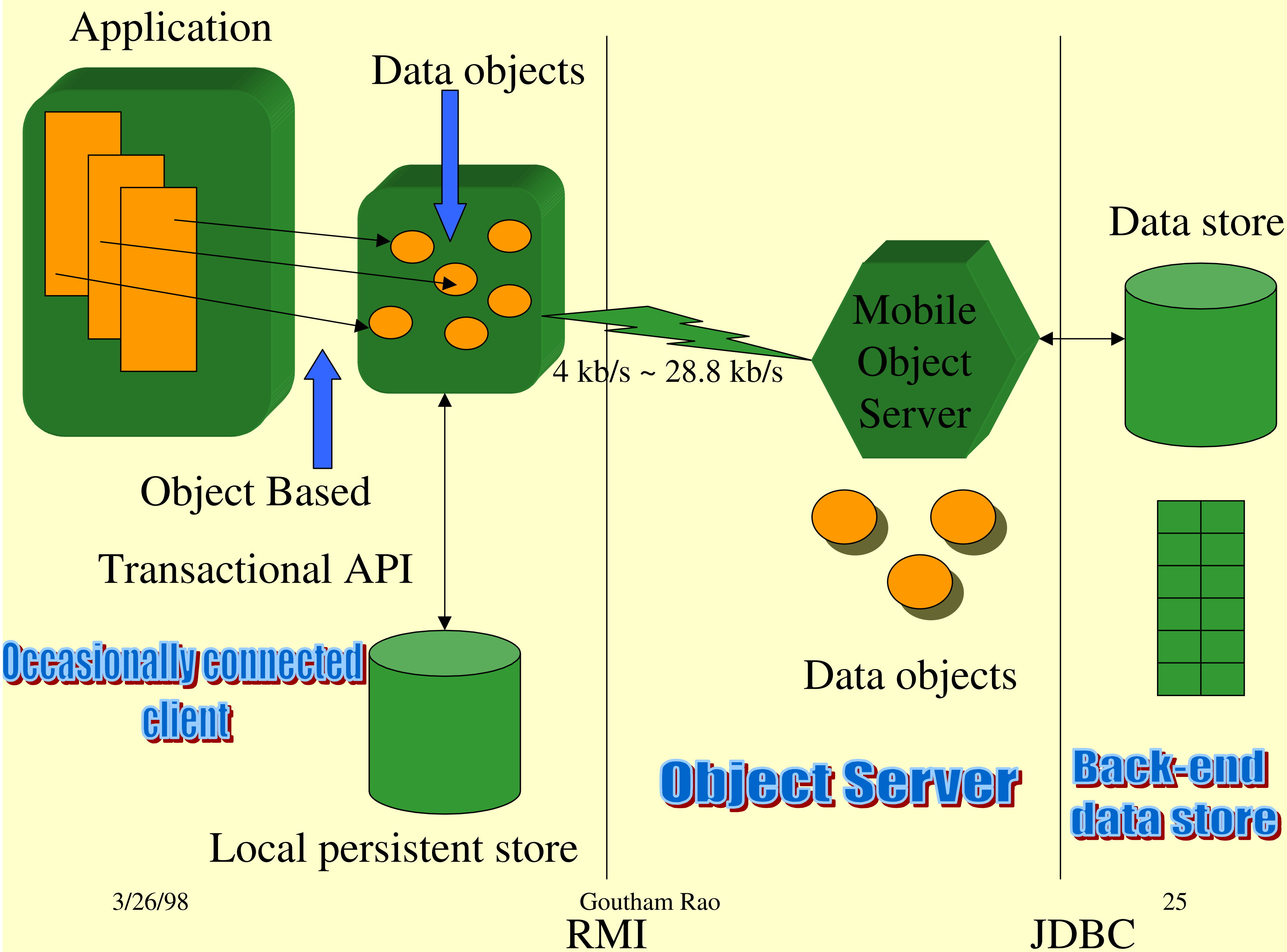
- Gold Rush provides a three tier approach moving the database implementation to the server side.
- Gold Rush provides support for:
  - A wire efficient access protocol.
  - Object caching and replication.
  - Logging of deferred transactions.

Gold Rush Overview

- Gold Rush mobile data management is based on JAVA objects.
- Important to note that enterprise data is most likely in relational databases: a very small portion is in object databases.

Gold Rush Architecture

- The three tier architecture consists of:
  - A JAVA client.
  - An intermediate mobile object server.
  - Back end data store.
- The middleware provides the client application with the same transaction API regardless of the connected mode.
  - Some features are not available when client is disconnected.


Application
Data objects
Data store
Mobile Object Server
4 kb/s ~ 28.8 kb/s
Object Based
Transactional API
Occasionally connected client
Data objects
Object Server
Back-end data store
Local persistent store
RMI
JDBC

Gold Rush Architecture

- The Gold Rush middleware has the following basic components:
  - Database objects:
    - *A data base object is a JAVA object that represents an instance of a database entity.*
  - Object caching:
    - *To support disconnected transactions on database objects, there is a persistent local object store on the client.*

Gold Rush Architecture

- Transactions with optimistic concurrency control:
  - *The client works primarily off-line.*
  - *Transactions are logged on the client and replayed to the server at a later time.*
  - *Since multiple clients exist, we need concurrency control.*
  - *Concurrency control is optimistic.*
    - *An object read may be lock with an optimistic read lock or left unlocked.*
    - *An object written is locked with an optimistic write lock.*
    - *Optimistic because the client does not actually have the lock, but hopes it will during replay.*

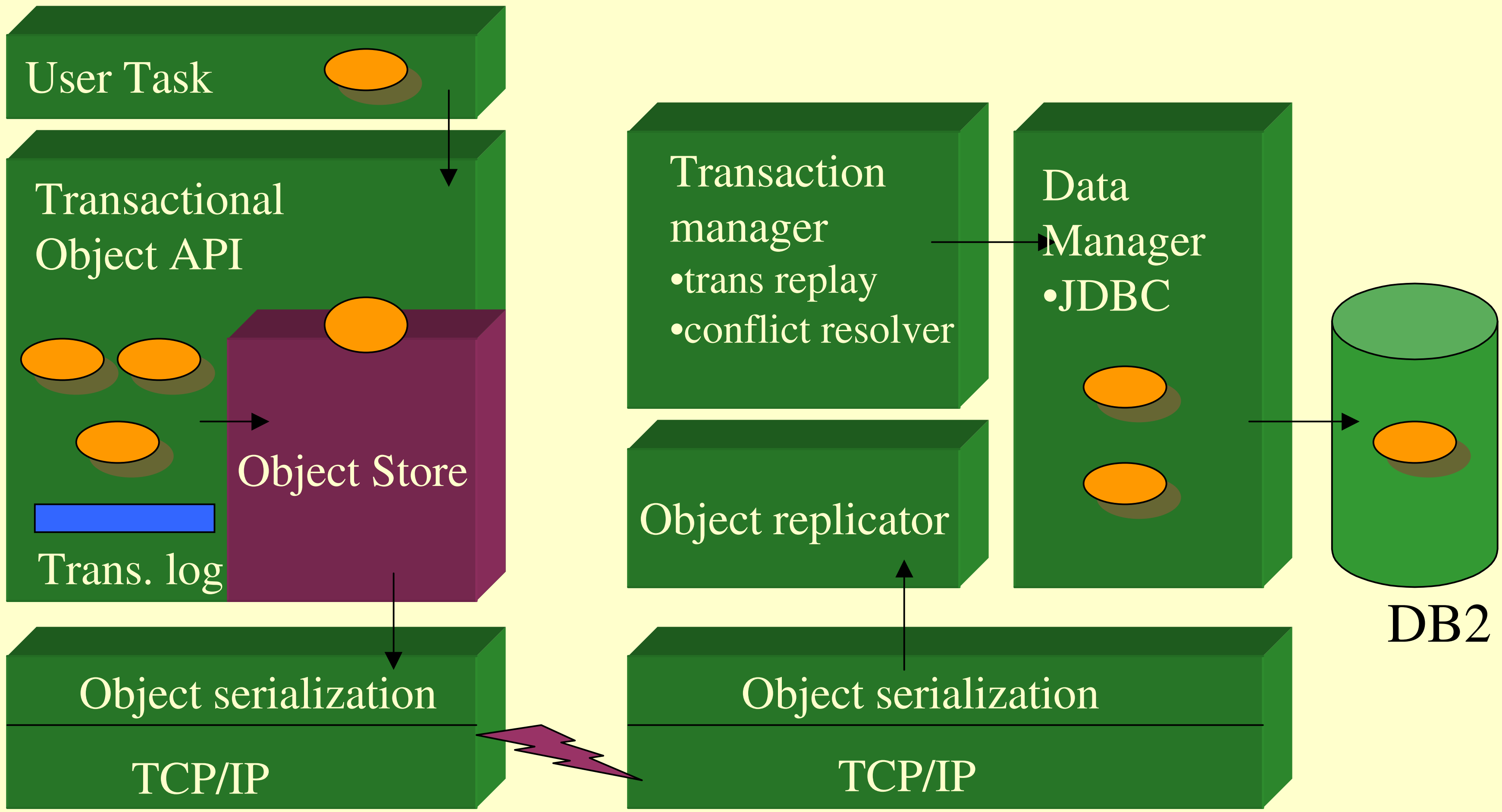
Gold Rush Architecture

– Communication:
  - Database objects are transferred between the client and the server using RMI and object serialization.
  - Transfer is optimized... Only differences in the object versions are transmitted.

Gold Rush Architecture

NOTE

- Some parts of the applications that are specifically mobile are exposed to the programming interface:
  - Handling the modes of connectivity.
  - Pre-fetching and downloading data objects.
  - Controlling the replay of transactions.
  - Resolving conflicts during reconnection.

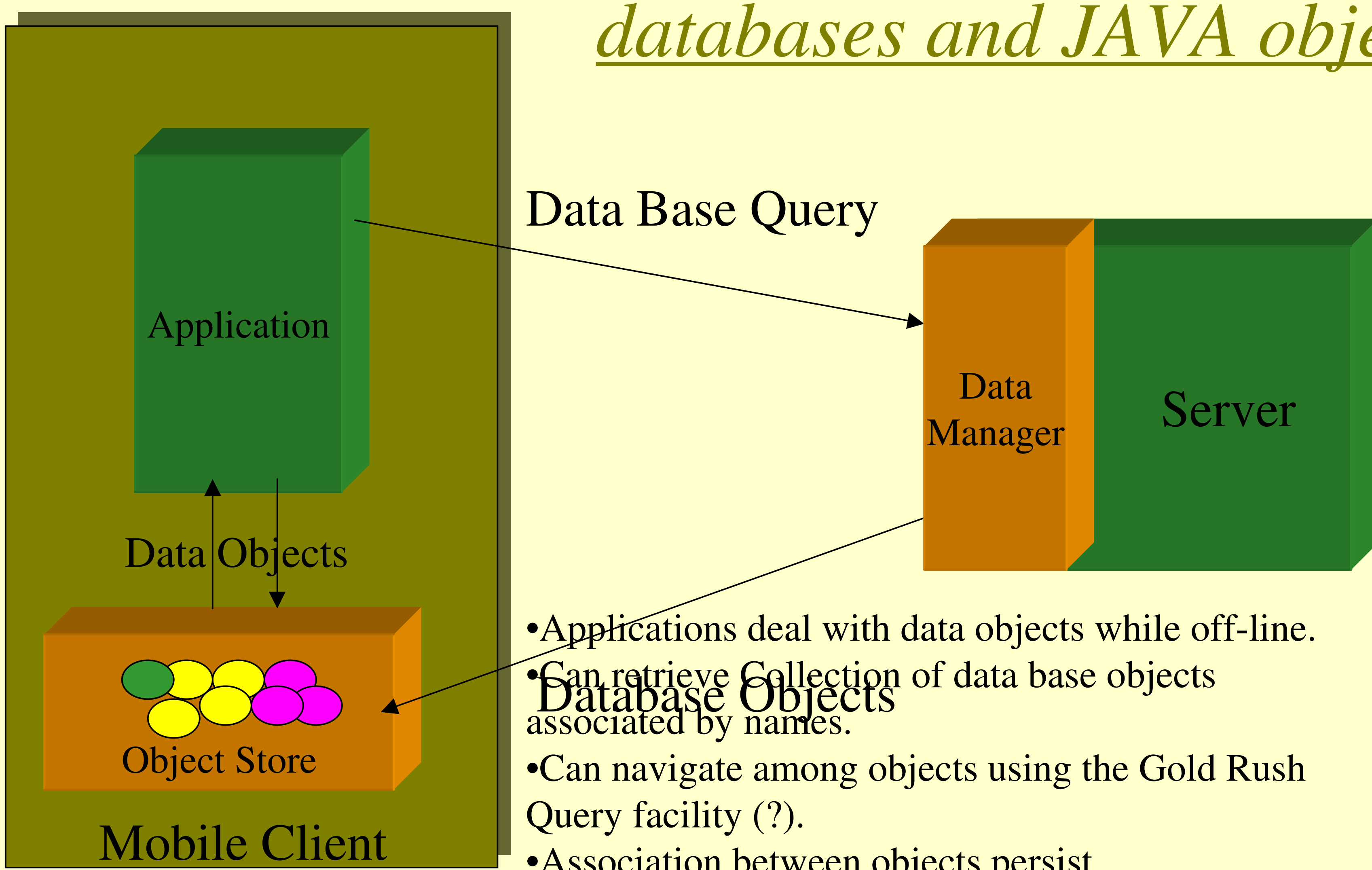


Mobile middle ware components and the off line transaction process

Correspondence between relational databases and JAVA objects

- On the client, Gold Rush manages data objects, not any portion of the data base
- A database object corresponds to a row in a relational database table (entity).
- Relationships that exist between tuples of an *rdbms* are brought about in the object.
    - 1-n and m-n relationships through foreign keys

Correspondence between relational databases and JAVA objects



•Applications deal with data objects while off-line.
•Can retrieve Collection of data base objects associated by names.
•Can navigate among objects using the Gold Rush Query facility (?).
•Association between objects persist.

Correspondence between relational databases and JAVA objects

- Gold Rush provides a tool to map relational data to object classes.
  - Lotus Notes tool uses relational data base tuples to generate class definitions for the data base objects.
  - Generates Code needed to instantiate objects from tuples.
  - Generates code that navigates between related object instances.

Correspondence between relational databases and JAVA objects

- Gold Rush provides for a data manager.
  - Resides on the server.
  - JDBC based data manager class.
  - Provides SQL statements for retrieval, insertion and update.
  - While client is connected, data manager can retrieve records from the data base and return them as a collection of JAVA data objects.
    - *One JAVA data object per row that satisfied the query*.
  - Can take in a JAVA data base object and insert a new row in the data base checking for constraint violations and for conflicts.
  - Can take a JAVA object and replace an existing row, checking for constraint violations and for conflicts.

Client Object Store

- Data is cached in the object store on the client
- Object store provides
  - Object lookup
  - Object store
  - Object update
  - Object retrieval functions

Client Object Store

- *Caching:* Data is cached during connected transaction processing. Latest version is stored.
- *Retrieval:* Applications retrieve data from the store while disconnected.
  - Can access a particular data object
  - Can access a collection of objects of a particular class
- *Committal:* All mutated objects of a committed transaction are updated. A log is maintained.
- *Replay:* On reconnection with server, transactions are replayed. Object store cleans up stale versions.

Client Object Store

- Files maintained by object store
  - Class Files
    - All versions of all objects of the same class are stored in a single file
  - Class Index Files
    - A separate index file per class.
    - Provides fast look up.
  - Transaction File
    - A file per committed transaction. *(local Vs remote commit).*
    - Identifies exact version of each object involved in transaction.
  - Transaction Log
    - Record of all transactions.

Off-line transactional semantics

- When client is connected to server, locks are held in the traditional way.
- While disconnected, locks are granted optimistically. *"Lazy mode"*.
- When disconnected, transactions occur with local data objects.
- *Local Commit :* Local execution of the transaction.
- *Remote Commit :* Replayed transactions.
- Locks are used to detect conflict during Remote Commit.

Off-line transactional semantics

- Locks are of usual semantics
  - Shared read, exclusive write.
- On client locks are granted without contention.
- Possible to read an object without locking it.
  - This reduces contention during replay.
  - Good when it is known that an object will not change.
  - Good when approximate answers suffice
- During replay locks are always checked.
  - Other clients may have used the data object.

Off-line transactional semantics

BeginTransaction

Register particular objects *(locks are always granted)*

use/modify objects

Call commit *(This writes to the transaction log, class file etc.)*

Remote commit and conflict handling

- Locks are checked during remote commit
- On the server, every object has
  - a unique object ID
  - timestamp of the last update. *last-modified time*
- On the client, every object has
  - same object ID as above
  - *last-modified time*
    - *unique user number* + *local clock*
  - *local-commit time*
- Initially last-modified timestamp of client object is set to that of the server.

Remote commit and conflict handling

- Subsequently, every time the object is read
  - the last-modified timestamp is set to the last local-commit timestamp.
  - When the transaction completes, the local-commit timestamp is updated with the local commit time.

Remote commit and conflict handling

<u>Transaction Replay</u>

- Compare client object's *last-modified* timestamp and server object's *last-modified* timestamp
- if timestamps are the same then
    - if object has changed then replace object
    - change server objects *last-modified* timestamp equal to the client objects *local-commit* timestamp
    - read next object in transaction and repeat step 1
- else the timestamps do not match, some other client may have used the object, so abort current transaction.

Reducing data traffic

- Check object ID before transmitting.
- Transmit differences if possible.
- If object does not exist, transfer whole object

Discussion