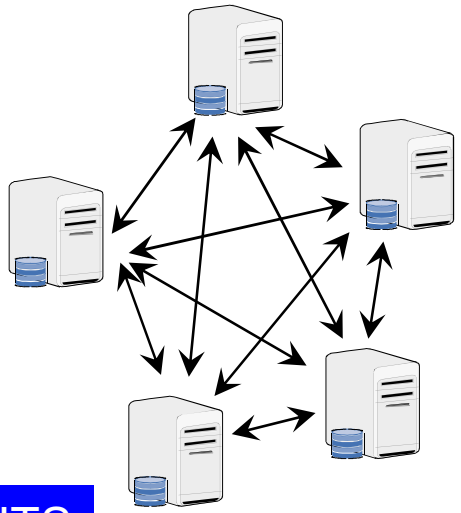
# IBM Informix 12.1 – Simply Powerful

## Informix Enterprise Replication & Flexible Grid

# Agenda – The Informix Flexible Grid

- Enterprise Replication – the Foundation
- What is the Informix Flexible Grid?
- Grid - Technical Overview & Requirements
- Grid Setup
- Grid Operations and Monitoring
- Other Grid Features
- Grid Benefits
- Grid and the Open Admin Tool
- Grid new features in 12.1
- Summary

# Enterprise Replication (ER) a.k.a CDR



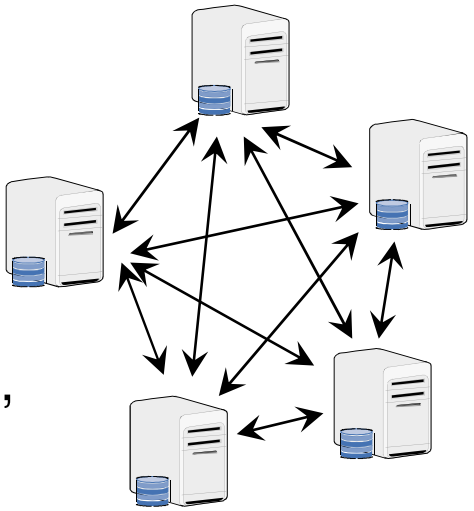
- The entire group of servers is the replication domain
  - Any node within the domain can replicate data with any other node in the domain
  - Servers in domain can be configured to be *root, non-root* and *Leaf*
- Supports
  - Heterogeneous OS, Informix versions, and H/W
  - Secure data communication
  - Update anywhere (Bi-directional replication)
    - Conflicting updates resolved by Timestamp, stored procedure, or always apply
- Based on log snooping

**BENEFITS**

- Low data transfer latency
- Already integrated in the server!
- Flexible
  - Choose what to replicate – column level!
  - Choose where to replicate – all nodes or select
- Scalable
  - Add or remove servers/nodes easily

# ER & Informix Flexible Grid

- Informix Flexible Grid takes Enterprise Replication (ER) to the next level
- Flexible scalability
  - Subset of data
  - Source and target can have different database, table and column names
- Integrated
  - Compatible with other availability solutions
  - Can coexist with MACH clusters
  - Any server can also have secondary
    - Shared disk secondary (SDS)
    - Remote secondary (RSS)
- Secure data communications



# Informix Enterprise Replication (ER)

- Log based, Transaction oriented replication
- Asynchronous, Homogeneous (IDS 7.22+ only)
- Primary/Target + Update anywhere
- Consolidation, Dissemination, Workload partitioning, Failover
- Tightly coupled with the server
- Web and command line administration

# ER History

- Initial Release: 7.22 in 12/1996
  - Version I - 7.22 - 7.30 releases, peer to peer
- Version II (7.31 & 9.2x)
  - Queue and NIF redesign, Hierarchical Routing
- Version III (9.3)
  - Extensibility, Increased parallelism, Smart blob queuing, In-place alter to add/drop CRCOLS, Serial Col Primary Key Support, …
- Version IV (9.4)
  - ER/HDR support, Large transaction support, Complex Type support, Performance enhancements, Network Encryption
- Version V (10.x)
  - Templates, Alter Support, Resync Support, Mastered Replicates, Shadow Replicates
- Version VI (11.x)
  - SSL, Performance, Monitoring, Role Separation, Dynamic Features, New event alarms, SQL Admin API usage, and Work with compressed data

# High Availability Data Replication (HDR) & ER differences

## HDR

- Provides single primary and single secondary
- Primary and secondary must run the same executables and have similar disk layout
- Secondary restricted to report processing
- Simple to set up and administer
- Primary and secondary are mirror images
- Does not support blobspace blobs
- Replication can be synchronous
- Primary purpose is for high availability

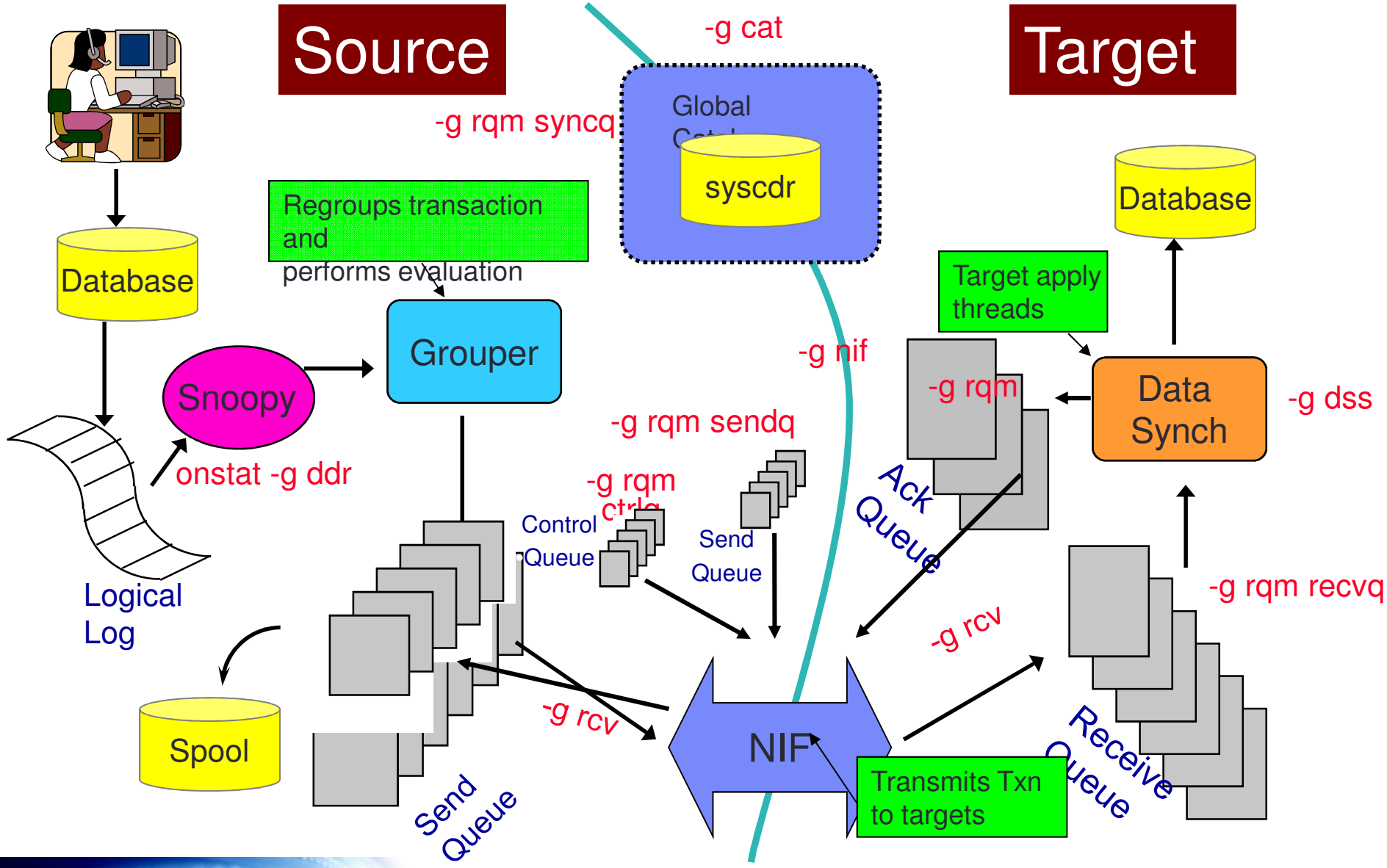
## ER

- Allows configurable source(s) / target(s)
- Source/target do not have to be the same
- Allows full usage of both source / target
- Setup and administration more complex
- Source and target can be different
- Supports blobspace blobs
- Replication is asynchronous
- Primary purpose is for data distribution

# How ER replicates a Transaction

1. A client application performs a transaction in a database table that has a defined *replicate*
2. The transaction is put into the logical log
3. The log capture component (snoopy) reads the logical log and passes the log records onto the grouper component
4. The grouper thread evaluates the log records for replication and groups them into a message (original transaction)
5. The grouper thread places the message in the *send queue* - under certain situations, the send queue spools messages to disk for temporary storage
6. The send queue transports the replication message to the target server
7. The replication message is placed in the receive queue at the target server
8. The data sync thread applies the transaction in the target database - if necessary, the data sync component performs conflict resolution
9. An acknowledgment that the message was successfully applied is placed in the *acknowledgment queue*
10. The acknowledgment message is sent back to the source server

# ER – how it works



# Send and Receive Data Queues

- Queues receive or deliver replication data to and from servers that participate in a replicate
- Send queue
  - Replication data stored in memory on source for delivery to participants
  - If the send queue fills, ER spools the send-queue transaction records to a *dbspace* and the send-queue row data to an *sbspace* (stable queue)
- Receive queue
  - Replication data stored in memory at the target database server until it acknowledges receipt of the data
  - If the receive queue fills as a result of a large transaction, ER spools the receive queue transaction header and replicate records to a *dbspace* and the receive queue row data to an *sbspace* (stable queue)
- Managed by the Reliable Queue Manager (RQM)
- Configurable using CDR_QUEUEMEM configuration parameter

# General ER Requirements

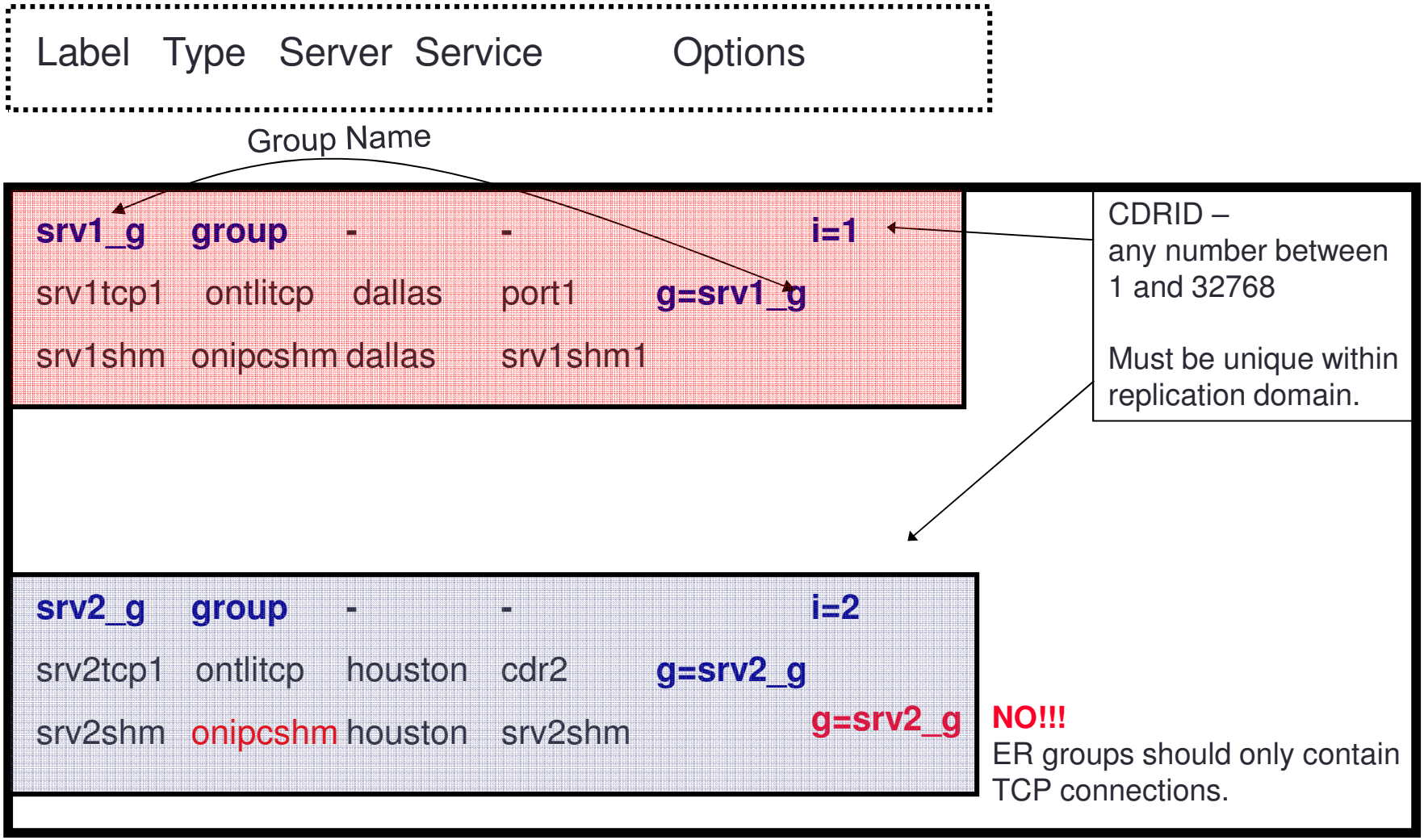
- Table must have a primary key (Not Anymore!!!)
- SQLHOSTS group entries
- Logged databases
- Logical Logs
- Conflict Resolution
- Topology
- Scope (Row/Transaction)
  - Rollback affects only Target
- Stable Queue
- Time synchronization

- Server to Server communications configured for all servers involved in ER
  - /etc/hosts
  - /etc/services
  - Trusted environment
    - hosts.equiv
    - .rhosts
  - Test: *rlogin* or dbaccess → connection → connect
- Logical logs (dbspace & archive)
- Extra database space for CRCOLS and delete tables
- Dbspaces for send and receive queues
- Dbspace for grouper paging file
- Disk space/directories for ATS and RIS files

# Requirements - Global Catalog

- Global inventory of ER configuration information and state
  - Found on all *root* and *nonroot* replication servers
  - Stored in the **syscdr** database
    - **syscdr** created when the first server is defined for replication
- The global catalog includes the following:
  - Enterprise Replication server definitions and state
  - Routing and connectivity information
  - Replicate definitions and state
  - Participant definitions and state
  - Replicate set definitions and state
  - Conflict detection and resolution rules and any associated SPL routines
- Tables in one global catalog instance are automatically replicated to the global catalogs of all other replication servers (except leaf servers)
  - Allows for management of complete ER domain from one non-leaf replication server
- *Leaf* replication servers have limited catalogs

# Setup - Sqlhosts



# Setup - Servers

If timeout is 0 (default),
The connection does not time out.

- Define first Server

```
cdr define server --connect=stan \
--idle=500 --ats=/cdr/ats --ris=/cdr/ris \
--atsrisformat=text --init g_stan
```

Adds server *group* to the replication domain

- Add another server

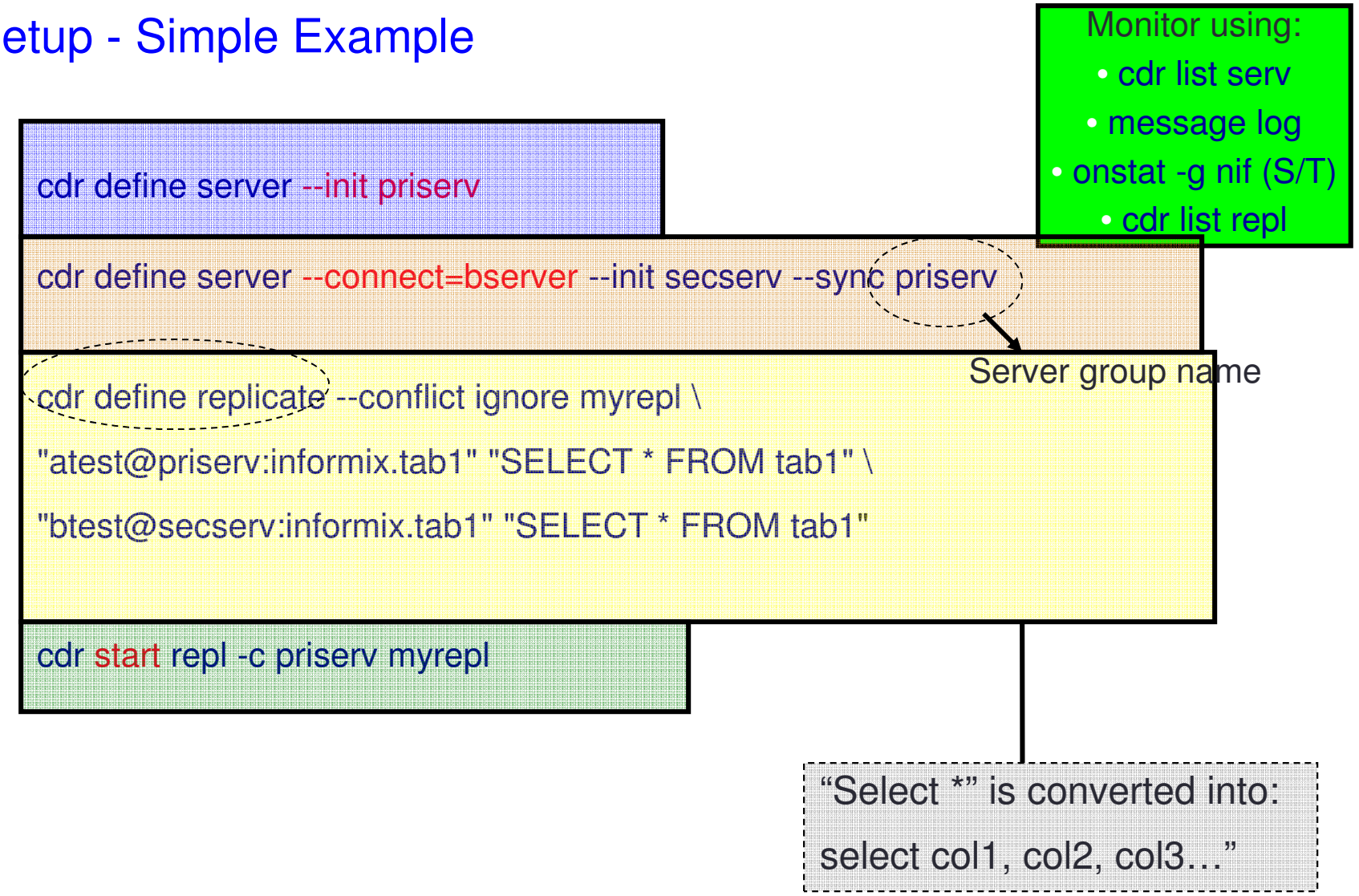
```
cdr define server -c oliver -i 600 \
-A /cdr/ats -R /dev/null -X xml \
-S g_stan -I g_oliver
```

Synchronizes catalogs with the catalogs on the existing database server *stan*

# Setup - Replicates

- Replicates
  - Defines Participants
  - Defines what data to transmit
  - Defines conflict resolution rules and scope
- Replicate set
  - Grouping replicates so they have the same characteristics
- Templates
  - Easier set up of replication with large numbers of tables to replicate
  - Defines a group of master replicates and a replicate set

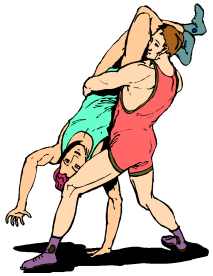
# Setup - Simple Example

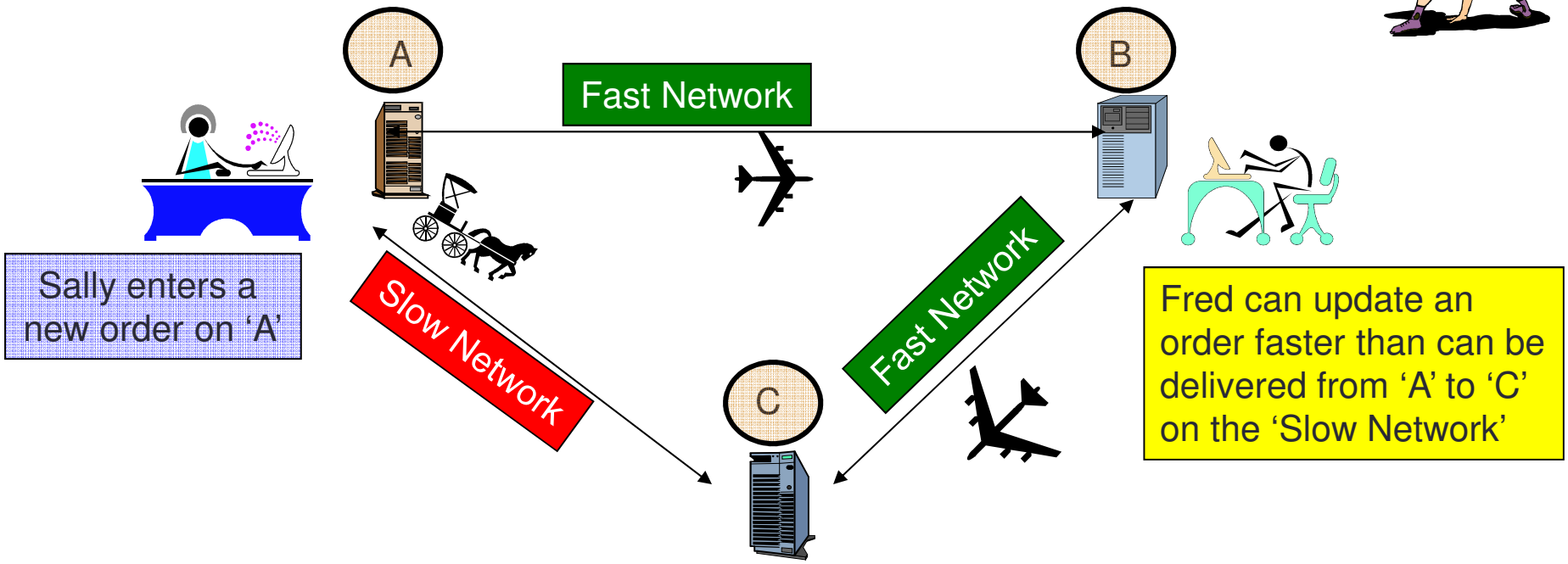


# Setup – Configuration Parameters

- CDR_EVALTHREADS - The number of grouper eval threads
- CDR_DSLOCKWAIT - Number of secs to wait to release DB lock
- CDR_QUEUEMEM - The maximum amount of memory (KB)
- CDR_NIFCOMPRESS - Controls the network interface
- CDR_SERIAL - Serial Column Sequence
- CDR_DBSPACE - The dbspace name for the syscdr
- CDR_QHDR_DBSPACE - The name of the transaction record
- CDR_QDATA_SBSPACE - The names of sbspaces for spooled
- CDR_MAX_DYNAMIC_LOGS - Maximum number of dynamic logs
- CDR_SUPPRESS_ATSRISWARN - Suppress DSync error & warning code
- CDR_ATSRISNAME_DELIM - Delimiter used in the time portion of ATS and RIS text files
- CDR_DISABLE_SPOOL - Controls generation of ATS/RIS files

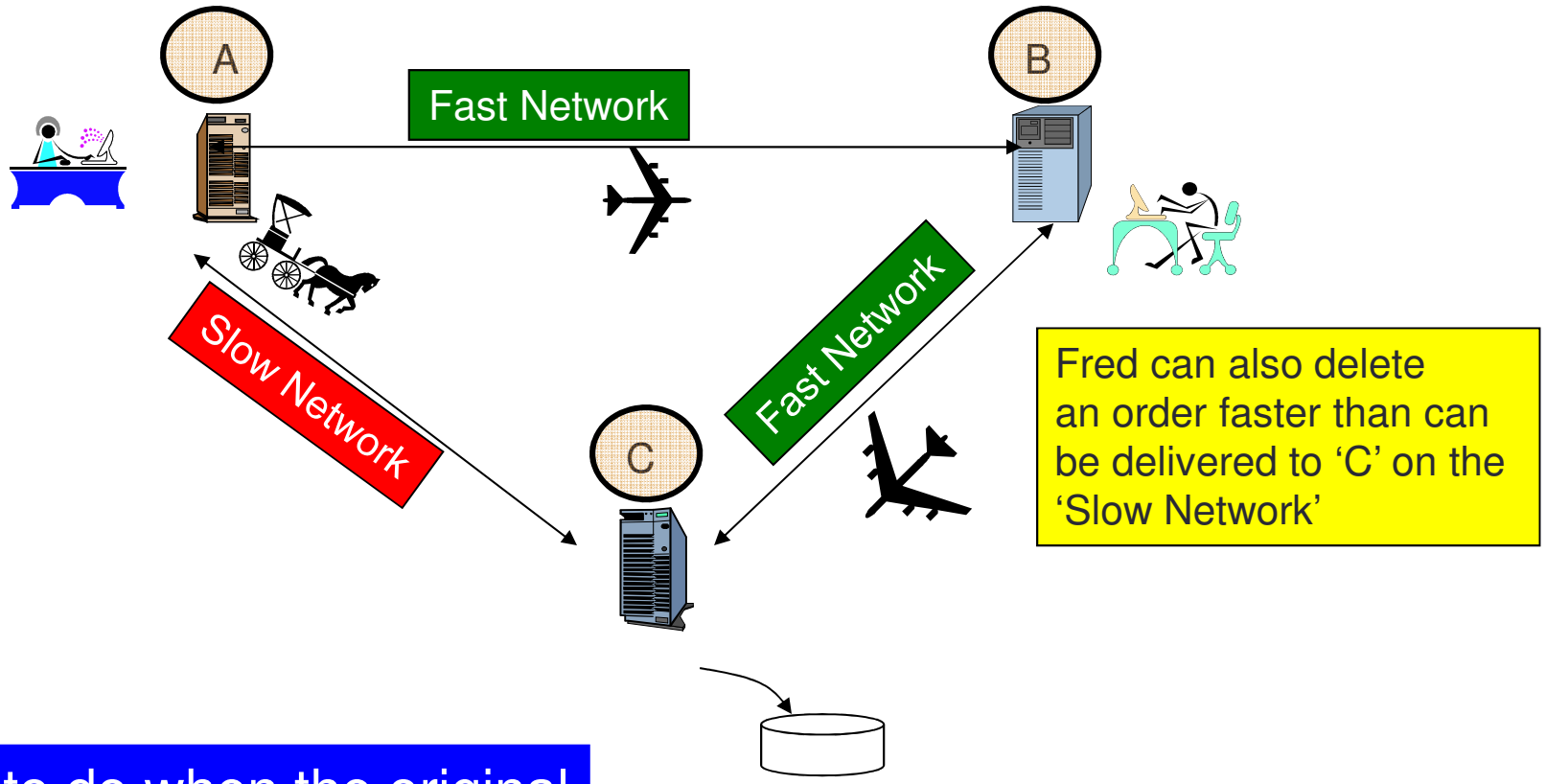
# What is Conflict Resolution?






- What to do when the original order finally arrives from A after Fred's update has been applied?
- Who Wins???

# What if Fred Deleted the order?




What to do when the original order finally arrives from A?

# Conflict Resolution

- Method to determine if the current version or a just received version of the row should 'win'

| CR Type | Description |
|---|---|
| Ignore | Row must be applied as is |
| *Timestamp | Most recent update wins<br>Upsert processing |
| *Timestamp with SPL | If time stamps are identical, call stored procedure |
| Delete Wins | DELETEs and INSERTs win over UPDATEs; else row/transaction with the most recent time stamp applied |
| Always Apply (10.0) | Like Ignore but performs upserts |

# Defining Shadow (Hidden) Columns

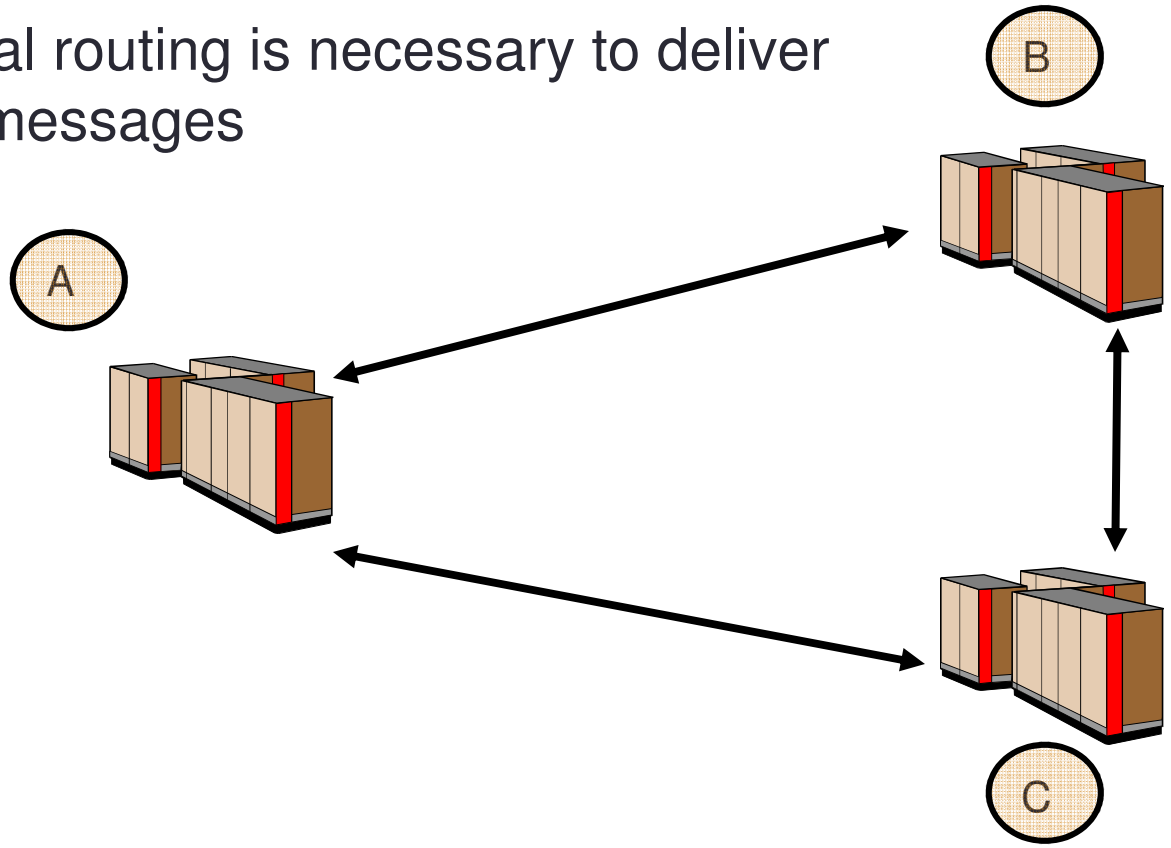
| Table Specification | Description |
|---|---|
| WITH CRCOLS<br>`ALTER table tabname add CRCOLS;`<br>`CREATE table tabname (...) with CRCOLS` | Required for time stamps and TS UDR conflict resolution<br>Creates two hidden shadow columns used for conflict resolution<br>• cdrserver: contains the identity of the server where the last modification occurred<br>• cdrtime: contains the time stamp of the last modification |
| WITH ERKEY | Used for tables that do not have a primary key<br>3 columns added: ifx_erkey_1, ifx_erkey_2, and ifx_erkey_3<br>Visible columns – can be indexed and viewed in catalogs<br>Columns used to create a unique index (PK) and a unique constraint |
| WITH REPLCHECK | Creates the visible ifx_replcheck shadow column that is used for consistency checking<br>Must create a new unique index on the primary key and the ifx_replcheck column<br>Allows the server to determine whether rows in different tables have different values without comparing the values in those rows |

# ATS and RIS files

- Aborted Transaction Spooling
  - *Transactions* that fail to be applied to the target database
  - Entire transaction is aborted
  - Transactions defined with row scope that have aborted rows but are successfully committed on the target tables are not logged
  - All rows that fail conflict resolution for a transaction that has row scope defined are also written to the RIS file
- Row Information Spooling
  - Replicate *row* data that fails conflict resolution or encounters replication order problems
  - Replication exceptions (such as upserts, insert→ update)
  - SPLs return codes called to resolve a conflict

# Routing - Fully Connected

- Each server connects directly to every other database server in the replication environment
- No additional routing is necessary to deliver replication messages

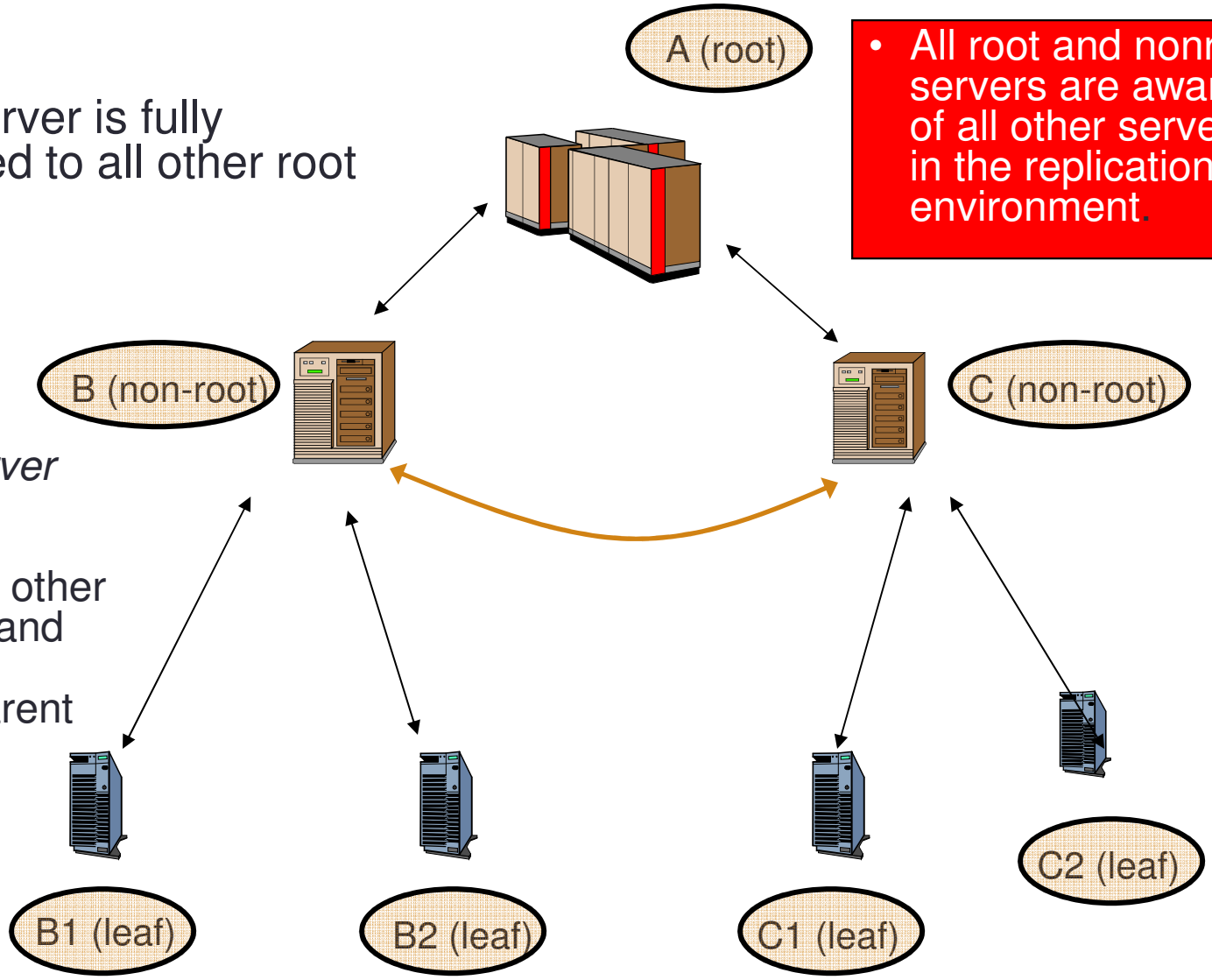


# Routing - Hierarchical Tree

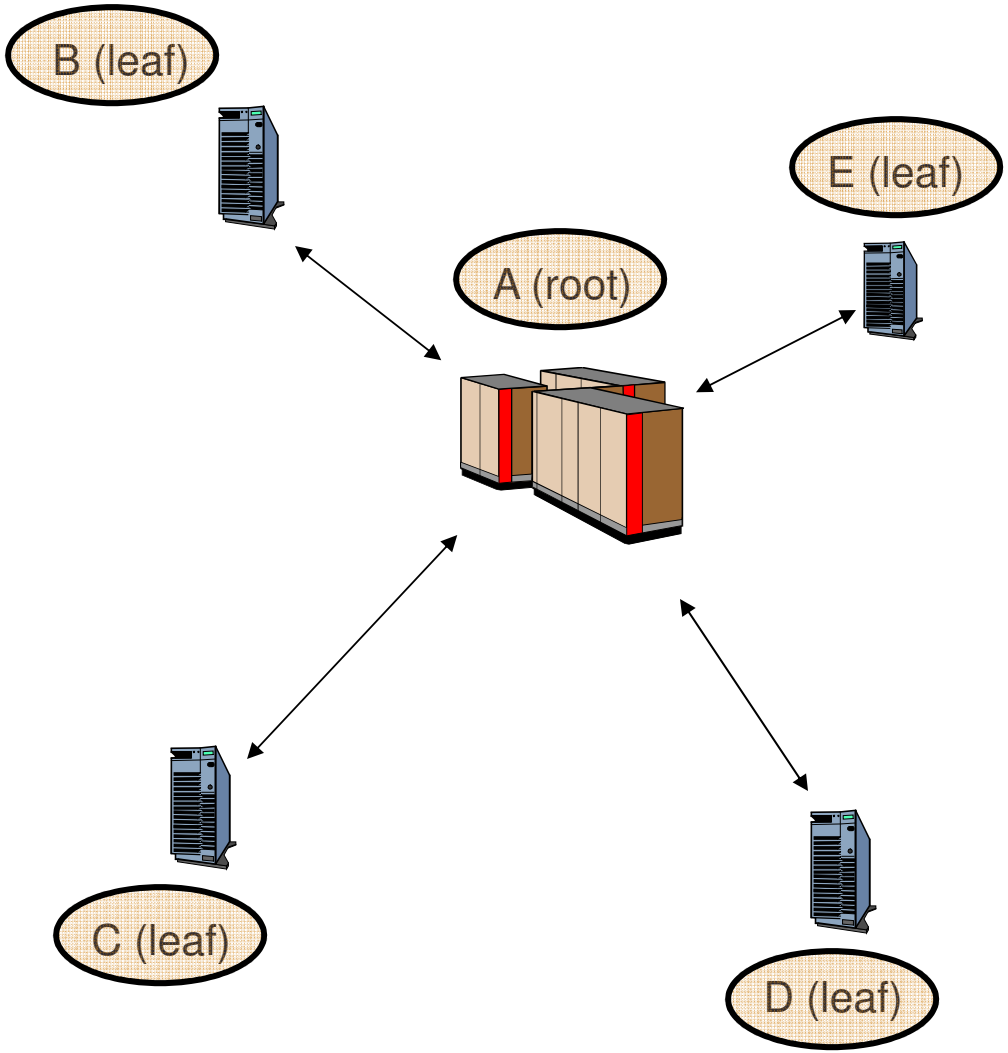
- A *root* server is fully connected to all other root servers

- All root and nonroot servers are aware of all other servers in the replication environment.

- A *nonroot server* forwards all replicated messages for other root servers (and their children) through its parent



# Routing - Hub/Spoke

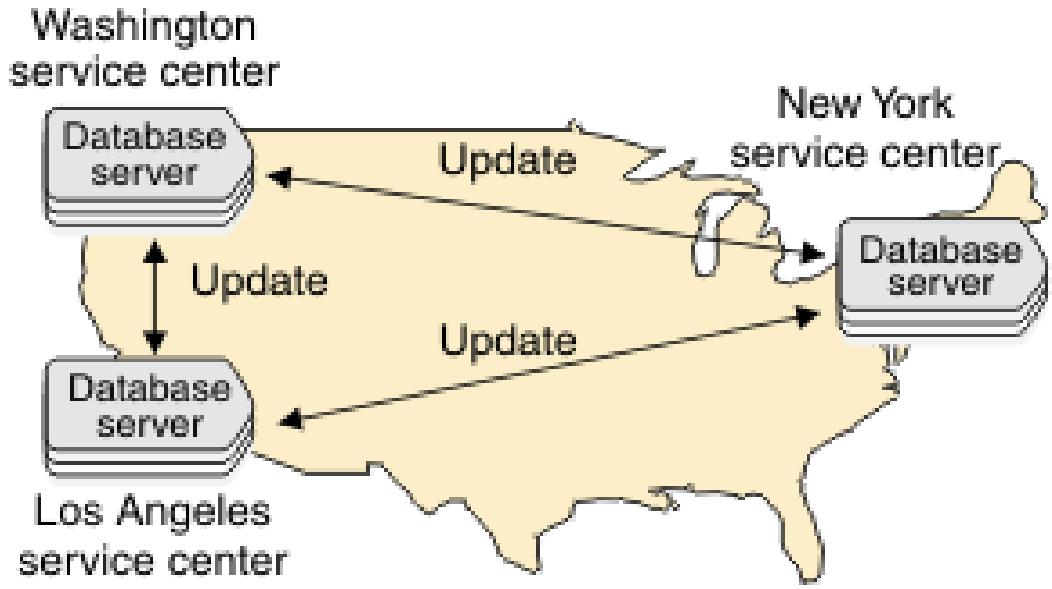


# Primary-Target Replication

- Flow of information is in one direction
- Changes at many target servers are not replicated to the primary
- One-to-many replication (*data distribution*)
  - All changes to a primary database server are replicated to many target database servers
- Many-to-one replication (*data consolidation*)
  - Many primary servers send information to a single target server

# Update-anywhere replication

- Changes made on any participating server are replicated to all other participating servers
- Allows users to function autonomously even when other systems or networks in the replication system are not available



# Things to think about Scope

- Transaction scope is really “All or Nothing”
  - If one row fails within the transaction and you are in transaction scope, then all of the rows fail
  - The transaction is always applied as a transaction
  - The transaction is NOT rolled-back on the source
- Triggers are normally not fired on the target
  - Firing triggers can be a way to replicate a procedure rather than replicate a table change
- Timed Based replication is not a good thing

# Informix Flexible Grid – Questions Addressed …

- I want to create a grid with a `mixture` of hardware, software, and Informix versions
- I want to set up my grid quickly and easily
- I want to *`easily`* administer ALL of servers in my grid
- I want my grid to support from 2 to 1000s of servers
- I want to `synchronize` my schema and data across the grid
- I want to be able to perform rolling upgrades and planned maintenance with no down time
- I want to scale capacity when and where needed within minutes?

# Informix Flexible Grid

- Comprehensive suite of capabilities that allows you to optimize utilization of your existing environment and maintain 24x7 operations

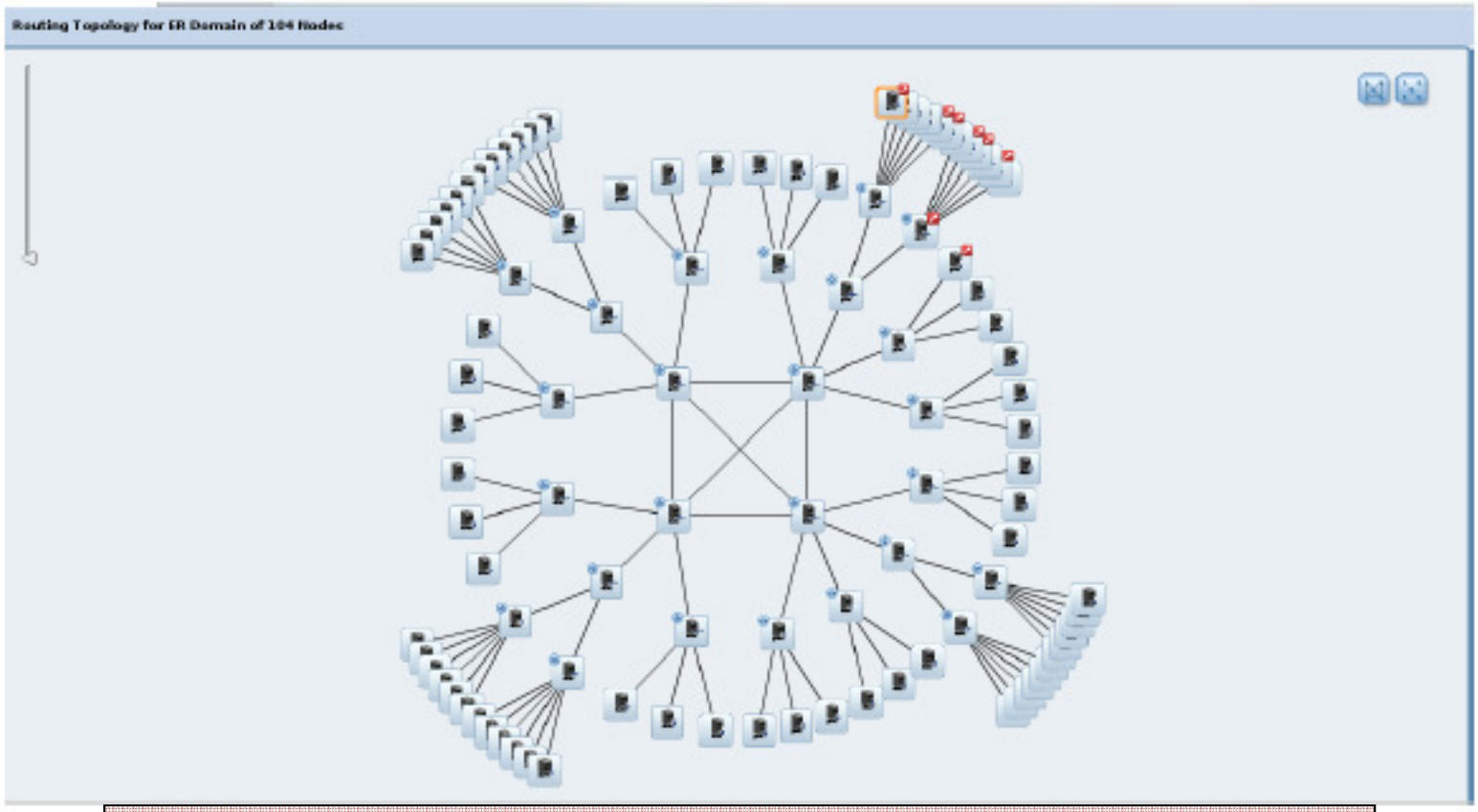
- Administer all servers remotely with SQL or the OpenAdmin Tool (OAT)
  - Define which servers are allowed to administer the grid
  - Attach to that server and administer all servers in your grid
  - Administer remotely with no on-site requirements

- Set Up Initial Grid in minutes or hours, rather than in days or weeks

# Informix Flexible Grid – What does it provide?

- The ability to create small to massively sized grids easily
- The ability to mix hardware, software, and versions of Informix in the Grid
- Centralized, simultaneous administration of servers and databases in the Grid
- Workload balancing across nodes in the grid
- Rolling upgrades (0 downtime during upgrades of O/S or Informix)
- Informix server/node cloning made easy
- Selective data replication if desired (just like in ER, but more powerful)

# Just how scalable and easy to manage is Flexible Grid?




OAT representation of a BETA customer who tested the Informix Flexible Grid

# Informix Flexible Grid – Technical Features

- Provides a means of replicating DDL across multiple nodes
  - **`CREATE TABLE, CREATE INDEX, CREATE PROCEDURE…`**
- Ability to replicate the execution of a statement rather than just the results of the execution
- Supports the connection manager on top of ER
- Can now replicate data using ER without a primary key
- Create ER replication as part of a `create table` DDL statement
- Make instance changes across all members of the Grid
  - Add / Drop logical logs, chunks, dbspaces, update **`$ONCONFIG`**, etc.
- Turn on/off ER replication within the transaction and not just at the start of the transaction

# Informix Flexible Grid – Requirements

- Informix Flexible Grid (Grid) builds upon an ER foundation
  - Enterprise Replication must be initialized
    - Create required db / smart spaces
    - Set the **$ONCONFIG** parameters
    - Create the **$SQLHOSTS** instance "group" definitions for Grid nodes like ER nodes
  - Specific syntax for Grid vs "regular" ER configuration and administration
- Grid requires all instances to be on Informix 11.7 or higher
    - Grid is still heterogeneous from a H/W perspective though

# Informix Flexible Grid - Primary Keys

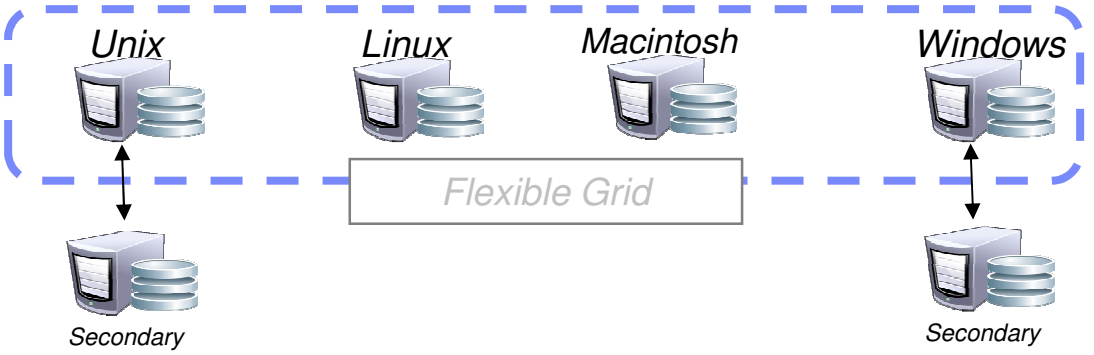
- No longer require a Primary Keys for tables replicated by Enterprise Replication (ER)
- Use the WITH ERKEY keyword when defining tables or `--erkey` when defining a replicate
  - Creates shadow columns (ifx_erkey_1, ifx_erkey_2, and ifx_erkey_3)
  - Creates a new unique index and a unique constraint using these columns that ER uses for a primary key
- For most database operations, the ERKEY columns are hidden
  - Not visible to statement like **`SELECT * FROM tablename;`**
- Example

```
CREATE TABLE customer (id INT) WITH ERKEY;
ALTER TABLE customer ADD ERKEY;
```

# Informix Flexible Grid – Easy to set up and use

- Install Informix on your server(s)
  - Grid servers (like the ER servers) may have secondary servers attached such as HDR, RSS, or SDS servers
- Define a grid to contain your servers
  - Give your grid a *name* and associate a list of servers with it
  - Use graphical interface or a command line tool for grid operations
  - Configure whether you want to replicate just schema changes or schema and data changes

The grid is ready to use!




Spread your workload across different HW, operating systems, and versions of Informix

# Defining the Grid

- The GRID is defined by using the `cdr` utility
  - Defines the nodes within the grid

```
cdr define grid <grid_name> --all
```

```
cdr define grid <grid_name> <node1 node2 …>
```

- Can create a grid based on an existing replication domain
  - Use the **`--all`** to include all replication servers in the domain in the grid
    - Do not use when running in a mixed Informix version environment

# Enabling the Grid

- Controls who can perform grid operations and from which server in the grid

```
cdr enable grid -grid=<grid_name> --user=<user>
                                  --node=<node>
```

  - At least one user and one server must be authorized
  - User *informix* does not have permission to perform grid operations unless explicitly authorized
  - Authorizing more than one server from which to run grid commands can lead to conflicts between grid commands
  - The users must have Connect privilege for all databases on which they run grid routines on all the servers in the grid

# Informix Grid - Initialization

- When a grid is enabled, the grid's name is used to generate a replicate set of the same name
- Any table that is created as a grid operation and is replicated becomes part of that replicate set

```
Pan_1: cdr list replset
Ex T REPLSET                    PARTICIPANTS
-------------------------------------------------
N  N ifx_internal_set           _ifx_check_timestamp,
                                _ifx_grid_cm_er_serv,
                                _ifx_grid_cm_nodes,
                                _ifx_grid_cm_sla, _ifx_grid_cmd,
                                _ifx_grid_cmd_ddl_part,
                                _ifx_grid_def, _ifx_grid_node,
                                _ifx_grid_part, _ifx_grid_users,
                                _ifx_qod_clock_differences

N  N mytest_grid
Pan_1:
```

The replicate names are numeric and don't include the table name

# Disable Grid Operations

- Used to remove a *node* or *user* from being able to perform grid operations

```
cdr disable grid -grid=<grid_name> --node=<node_name>
```

```
cdr disable grid -grid=<gird_name> --user=<user_name>
```

```
cdr disable grid -g <grid_name> -n <node_name>
                 -u <user_name>
```

- Revokes the permission to run routines on the specified grid from the specified user or server that were earlier granted

# Performing GRID DDL operations

- DDL operations will be performed on the target nodes within the Grid
  - Within the same database as on the source
  - By the same user as was on the source
  - Using the same locale as on the source
- DDL operations can be executed from any database, including system databases for which a user ID has connect permissions
    - Implies an end-user database is not required for Grid operations
- DDL operations create database/table/index/etc
  - But master replicates are NOT created *by default*

# Performing GRID DDL operations

- To perform DDL operations at a grid level
  - Must first connect to the Grid on an *authorized server* and as an *authorized user*
  - Execute the built-in procedure

```
ifx_grid_connect(<gridName>, <tag>, <er_enable>);
```

  - The *tag* and *er_enable* flags are optional
  - *er_enable* (0/1) enables or disables the creation of a replicate and replicate set AND starting replication for any tables created while the connection to the grid is open
  - The *tag* can be used to make it easier to monitor the success/failure of grid operations

# ifx_grid_connect(*grid_name*, *tag*, *er_enable*)

- Propagates DDL SQL statements and routines following the call to all the servers in the grid
  - Statement is simultaneously run on each server
- Does NOT propagate DML statements through the grid
  - The ER replication system propagates the results of statements to the other replication servers
- You must connect to a database before running this function
- Auto Registration of ER (*er_enable = 1*)
  - All tables created through the grid will have a replicate created that contains the newly created table with all the servers in the grid as participants
  - The replicate belongs to a replicate set that has the same name as the grid
  - The ERKEY shadow columns are added automatically

# Example of DDL propagation

Replicate will be created for table *tab1*

```
execute procedure ifx_grid_connect('grid1', 'tag1'. 1);

create database tstdb with log;
create table tab1 (
 col1   int primary key,
 col2   int,
 col3   char(20)) lock mode row;
create index idx1 on tab1 (col2);
create procedure loadtab1(maxnum int)
define tnum int;
for tnum = 1 to maxnum
 insert into tab1 values
    (tnum, tnum * tnum, 'mydata');
end for:
end procedure;

execute procedure ifx_grid_disconnect();
```

Will be executed on all nodes within the 'grid1' GRID

# Performing Other Grid Operations

- Three routines for easier execution of Grid operations

```
ifx_grid_execute()
```

  - Provides execution of a SQL-based DDL or administrative command as a Grid operation

```
ifx_grid_function() & ifx_grid_procedure()
```

  - Provide the ability to execute a function or procedure as a Grid operation

- Do NOT have to explicitly connect to / disconnect from the Grid using these operations

- Each operation can take three arguments

```
('gridname', 'command', 'tag')
```

# Performing Grid Procedure Execution

- In addition to DDL propagation, can perform the execution of a procedure, function, or statement as a grid operation.

```
execute procedure ifx_grid_procedure(
            'grid1', 'loadtab1(20000)','tag2');
```

  - This would cause the execution of *loadtab1(20000)* on all of the nodes within the *grid1* Grid
  - The command would be 'tagged' with "tag2"
  - By default, the results of the procedure would not be replicated by ER

# Performing Grid Function Execution

- The only difference between a function an a procedure is that a function will have a return value
  - The return is saved in the syscdr database and can be viewed from `cdr list grid`

```
database sysadmin;
execute function ifx_grid_function('grid1',
  'task("create dbspace","dbsp3",
"/db/chk/chk3",    "8G","0")');
```

  - The above would create a new 8GB dbspace called “dbsp3” on all nodes within the grid
  - By default the results of the function execution would not be replicated by ER

# Performing Grid Statement Execution

- An individual statement can also be executed as a grid statement

```
execute procedure ifx_grid_execute('grid1',
  'delete from tab1 where mod(col1,2) = 1');
```

  - The execution is replicated, not the results of the execution
  - The table *tab1* could be a raw table, or even contained within a non-logging database
  - By default, the results would not be replicated by ER

# Replicating Captured Transactions

- You can enable replication within a transaction that is run in the context of the grid
  - Changes the *snoop* status of ER from *within* a transaction
  - By default, the results of transactions run in the context of the grid are not also replicated by ER
- In certain situations you might want to both propagate a transaction to the servers in the grid and replicate the results of the transaction
- Use the built-in procedure `ifx_set_erstate()` to change the replication state from within a transaction
- Important: Must reset the replication state back to the default at the end of the transaction or replication loops indefinitely

# Example of enabling ER for the execution of a Procedure

- Retail chain wants to run a procedure to create a report that populates a summary table of each store's current inventory
- The summary information then needs to be sent (replicated) to a central server from each store

```
execute procedure ifx_grid_connect('grid1');
 create procedure myproc()
   execute procedure ifx_set_erstate('on');
   execute procedure create_summary_report();
 end procedure;
execute procedure ifx_grid_disconnect();
```

```
execute procedure ifx_grid_procedure('grid1','myproc()');
```

# Grid Operation Functions

- Operations can be run by any database in any node on the Grid

| | |
|---|---|
| `ifx_grid_connect()` | Opens a connection and any command run is applied to the Grid |
| `ifx_grid_disconnect()` | Closes a connection with the Grid |
| `ifx_grid_execute()` | Executes a single command across the Grid |
| `ifx_grid_function()` | Executes a routine across the Grid |
| `ifx_grid_procedure()` | Executes a procedure across the Grid |
| `ifx_set_erstate()` | Controls replication of DML across the Grid for all tables that participate in a replicate |
| `ifx_get_erstate()` | Reports whether replication is enabled on a transaction that is propagated across the Grid |
| `Ifx_grid_purge()` | Purges metadata about operations that have been executed on the Grid |
| `ifx_grid_redo()` | re-executes a failed *and tagged* grid operation |

# Monitoring a Grid

- `cdr list grid`
  - View information about server in the grid
    - View the commands that were run on servers in the grid
    - Without any options or a grid name, the output shows the list of grids
- Servers in the grid on which users are authorized to run grid commands are marked with an asterisk (*)
- When you add a server to the grid, any commands that were previously run through the grid have a status of PENDING for that server
- Options include:

  `--source=<source_node>`

  `--summary`

  **`--verbose`**

  **`--nacks`**

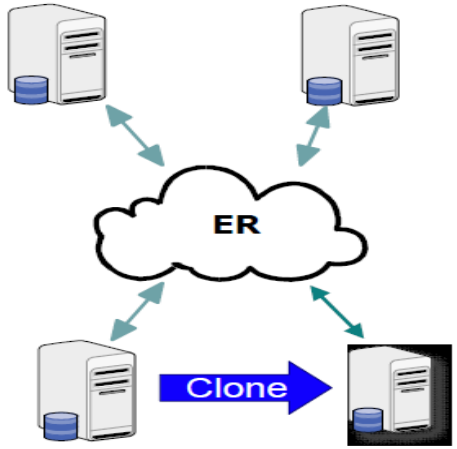
  **`--acks`**

  **`--commands`**

`cdr list grid grid1`

```
Grid                 Node                 User
-------------------- -------------------- ----------------
grid1                cdr1*                bill
                     cdr2
                     cdr3
```

# One Step Instantiation on *new* Nodes




- Previously, to clone the Primary
  1. Create a level-0 backup
  2. Transfer the backup to the new system
  3. Restore the image
  4. Initialize the instance
- ifxclone utility
  - Clones an instance from a single command
    - Starts the backup and restore processes simultaneously
    - No need to read or write data to disk or tape
  - Creates a standalone server, ER node or a remote standalone secondary (RSS) server
  - If creating a new ER node, ER registration is cloned as well
    - No Sync/Check is necessary

```
ifxclone -T -S machine2 -I 111.222.333.555 -P 456 -t machine1
                -i 111.222.333.444 -p 123
```

# Easily Convert Cluster Servers to ER nodes

- RSS → ER
  - Use the **`rss2er()`** stored procedure is located in the syscdr database
  - Converts the RSS secondary server into an ER server
  - Secondary will inherit the replication rules that the primary had
  - Does not require a '**`cdr check`**' or '**`cdr sync`**'

- HDR/RSS pair → ER pair (`cdr start sec2er`) (11.50)
  - Converts an HDR/RSS pair into an ER pair
  - Automatically creates ER replication between primary and secondary server
  - Splits HDR/RSS pair into independent standard servers that use ER

# Upgrading a Cluster while it is Online

- Use 'cdr start sec2er' and 'ifxclone' to perform a rolling upgrade of an HDR/RSS pair
- No planned down time required during a server migration!!!

- Basic Steps
  1. Execute 'cdr start sec2er' to convert HDR/RSS pair to ER
  2. Restrict application to only one of the nodes
  3. Migrate server on which the apps are not running
  4. Move apps to the migrated server
  5. Use ifxclone to switch back to RSS/HDR

# Informix Flexible Grid - Benefits

- Supports a wide range of platforms
  - Enables organizations to leverage inexpensive commodity hardware to accomplish their scalability objectives
- Given its replication strategy, can seamlessly add and remove servers in a Grid with little impact on the client applications
- Benefit of using Informix Flexible Grid with the Informix Continuous Availability Feature
  - Organizations enjoy similar benefits in addition to the incremental flexibility of grid

# Informix Flexible Grid - Benefits

- Demonstrates exceptional reliability and autonomics
  - Virtually eliminating maintenance activates on one or many members
- Provides unique ability to simultaneously propagate DDL and DML globally across a large group of interconnected members
- Provides cloning capabilities to speed the creation of member nodes

# Informix Flexible Grid - Easy to Setup and Manage in OAT


## Easy to Manage

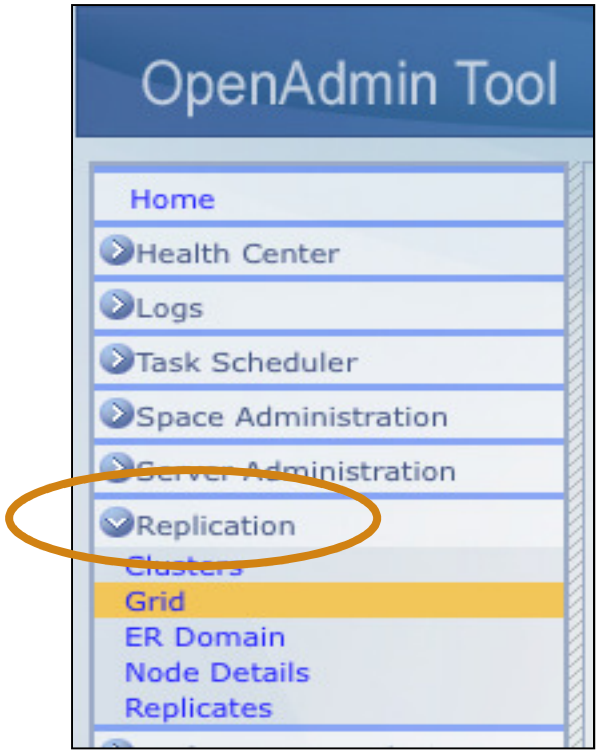
- Manage ANYTHING from ANYWHERE with a single ‘click’
- Set-up takes minutes not days or weeks
- Propagate changes to all servers on the Grid

## Reliable

- Enterprise Replication (ER) must be setup and running
- Grid servers must be on 11.70 (Panther)
- OAT Replication plug-in must be installed

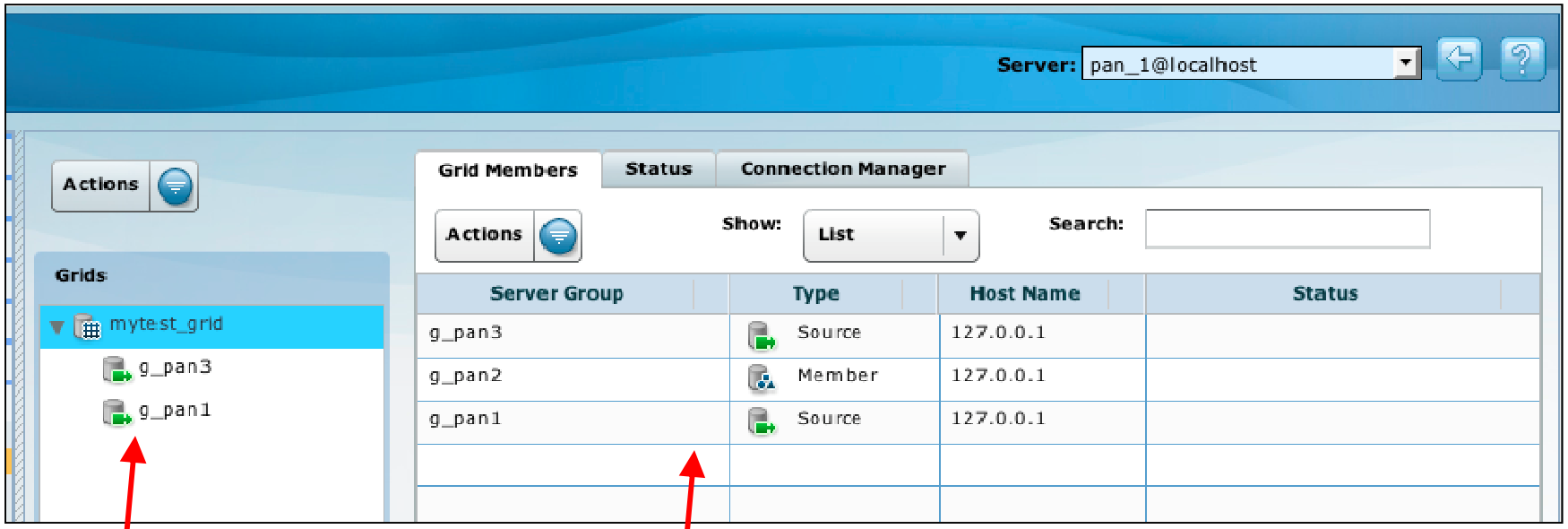
# OAT and the Grid

- The OpenAdmin Tool for Informix (OAT) contains monitoring and administrative options for Grid installations
- Available under the new Replication menu option



# OAT and the Grid

- Overview page

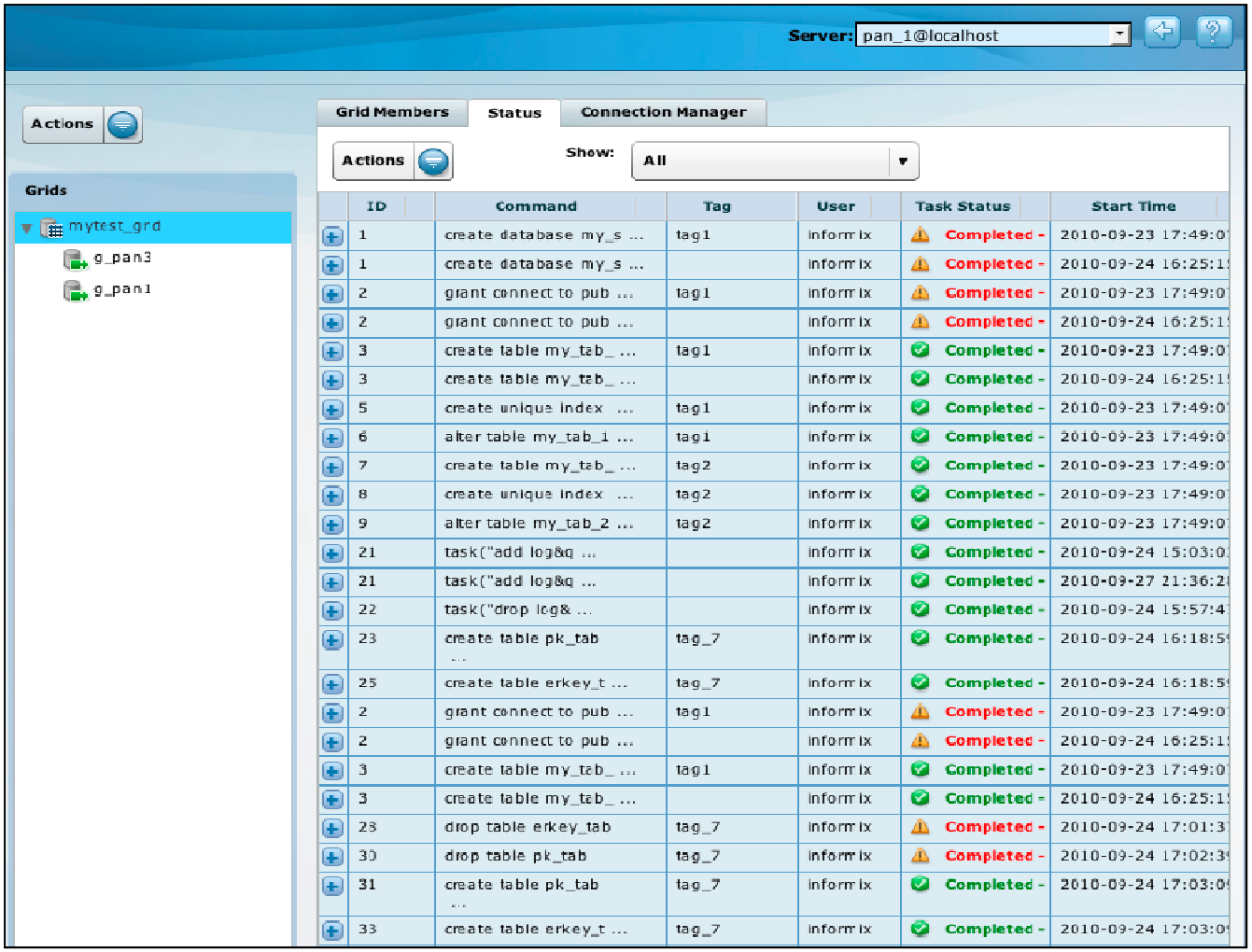



Grid name and administrative nodes

All nodes in the Grid, their location, and type `source` (aka administrative) or `member` (aka regular)

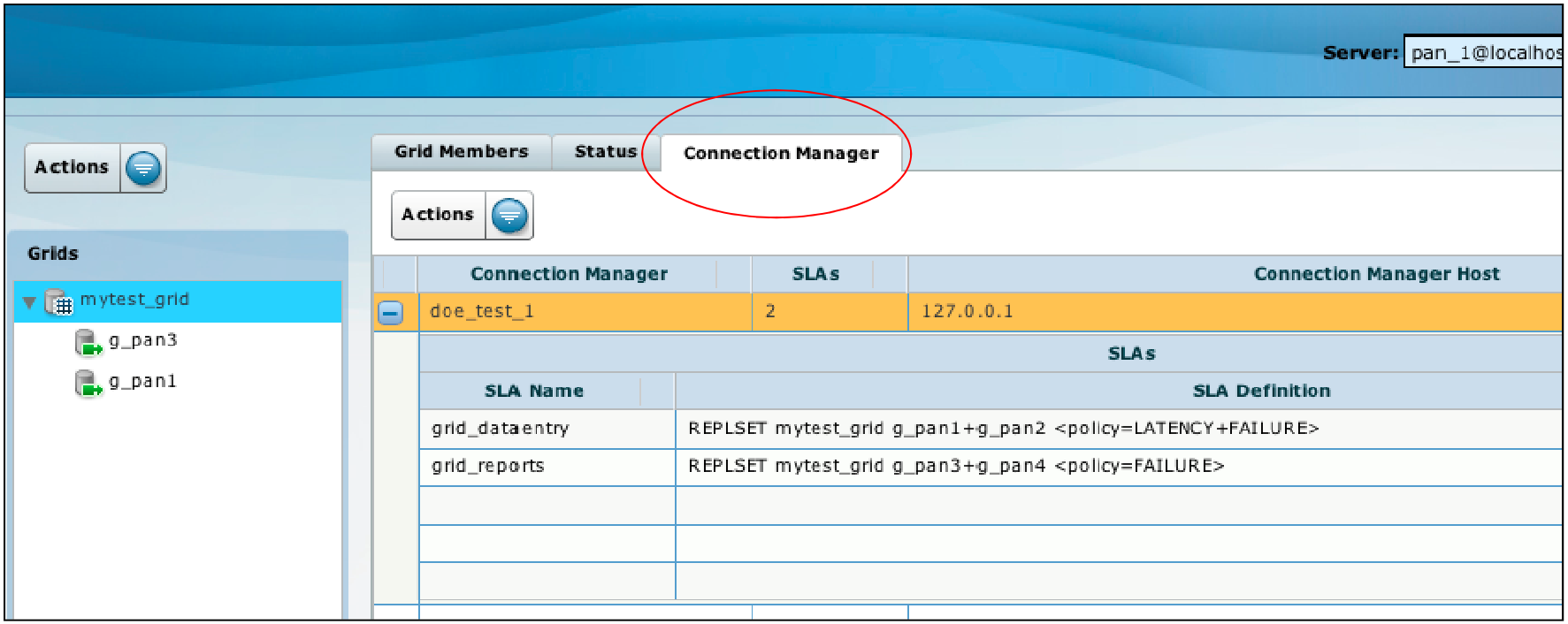
# OAT and the Grid

- Review the results of tagged Grid operations



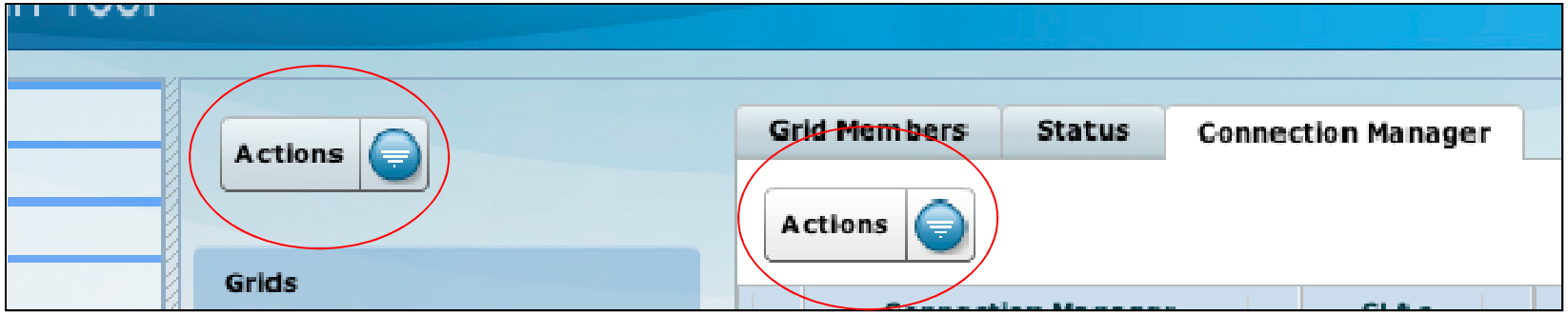
# OAT and the Grid

- View `oncmsm` agents for the Grid



# OAT and the Grid

- Depending on what you have selected, the two Actions buttons will present different options including:
  - Create or drop a Grid
  - Add a node
  - Add or remove an SLA
  - Change node enablement
  - More



# Grid Queries

- New GRID / GRID ALL clause to be used along with SELECT statements
- Table to be defined as “grid table” prior to using it in SELECT statements.
- Query against all the nodes in a grid or a sub-set of nodes defined as “regions”
- Regions can overlap
- Use ALL keyword to query all rows. (like union all)
- New built in functions ifx_node_id() and ifx_node_name() can be used to identify data origin. Can be used to group results.
- Use GRID_NODE_SKIP environment statement to skip grid servers that are unavailable.

# Grid Queries Examples

- Defining tables for grid queries
  - Adding tables orders / items are grid tables

```
cdr change gridtable –grid=grid1 –database=stores –add items orders
```

- Defining Regions

```
cdr define region –grid=grid1 region1 gs_north gs_south
```

- Grid Query

```
set environment select_grid_all region1;
select fname, lname, ifx_node_id() as node, sum(tot_quantity) as tot_cnt
        sum(tot_price) as tot_amt
  from items i, orders 0, customer c
where i.order_num = o.order_num
    and o.customer_num = c.customer_num
 group by 2,1,3
  order by 2,1,3
```

# Defer propagation of DDL statements

- Run DDL statements on the local server but defer propagation to other nodes in the grid
  - Use the ifx_grid_connect() procedure

```
execute procedure ifx_grid_connect("grid1", "tag1", 4)
```

- Third parameter to ifx_grid_connect with a value of 4 or 5 facilitates deferred propagation of DDL statements
- Use ifx_grid_release() procedure to start propagation of deferred DDL statements

```
execute procedure ifx_grid_release("grid1", "tag1");
```

# Copy external files from one grid server to another

- Copy non-database related, external data files from one grid server to another
  - Use the ifx_grid_copy() procedure

```
execute procedure ifx_grid_copy("grid1", "bin/new_cust_data.txt")
```

Copies the file new_cust_data.txt from the current node to all the nodes of the grid grid1.

- Path name is relative and is decided based on the ONCONFIG parameter GRIDCOPY_DIR
- Copy files to different relative locations is possible
- Copy files and rename them to something different than the source file name

# Summary

- A Grid is a named set of interconnected replication servers
  - Useful if you have multiple replication servers and you often need to perform the same tasks on every replication server
- Guaranteed to save lots of effort in synchronizing servers in a multi node ER environment
  - Ability to execute one DDL statement automatically multiple times
- Grid offers a lot of flexibility to programmers and DBA's
  - However, the flexibility of ER to replicate tables selectively forces programmers and DBA's to know beforehand where their data is supposed to be and where is it going to go in larger multi node environments
- Data consistency at the table level has implications for table design, application logic, etc

# Informix Flexible Grid – Data Consistency

- Informix Grid changes the concept of data consistency and accessibility within an Informix cluster
  - With Informix HA replication clusters, data consistency and accessibility has been consistent and at a cluster level
  - With ER clusters, Global data consistency is across all nodes for replicated data
    - Through Conflict Resolution
    - Individual rows can be easily found
  - With Informix Flexible Grid, data consistency and accessibility is only at a node level
    - Enforced at a local table level
    - Depends on how tables are created and how DML statements are executed

# Informix 12.1: Simply Powerful

# Thank you