Lotusphere Europe

# AD113
# Replication: What It Is And How It Works

George Langlais: Iris Developer
Programmability Team

Lotus.

WORK AS ONE

# Replication Is ...

- A process for distributing/resynchronising databases
  - Step 1 - Make a replica copy of a Notes/Domino database
  - Step 2 - Modify the replica copy and the original replica
  - Step 3 - Replicate (resynchronise) the two replicas
- A replica retains info needed to resynch with other replicas
- An essential groupware feature
  - Work offline for convenience and/or performance
  - Balance server loads
  - Reduce disruptions due to maintenance/problems

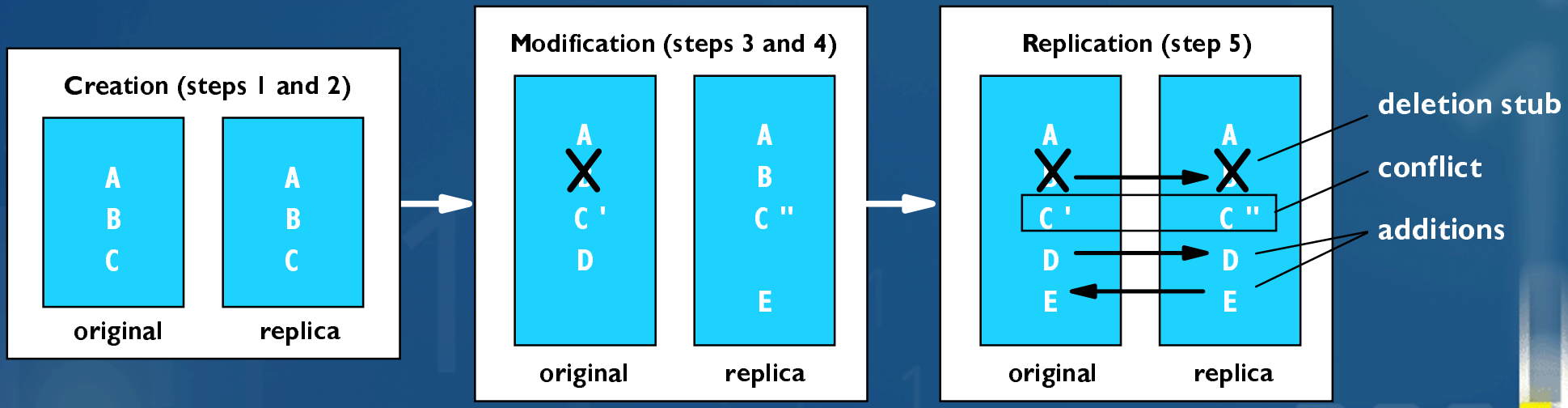
# Replication - A Solution That Fits How We Work

- Organizations typically have distinct operating groups that are separated by geographical and/or political boundaries
  - Constant online access to central data is often impractical

- Conflicts are infrequent in practice
  - Typically, only the author of a document modifies it later
    - Others add comments as "response" documents
  - Conflicts can be further minimised with a little planning

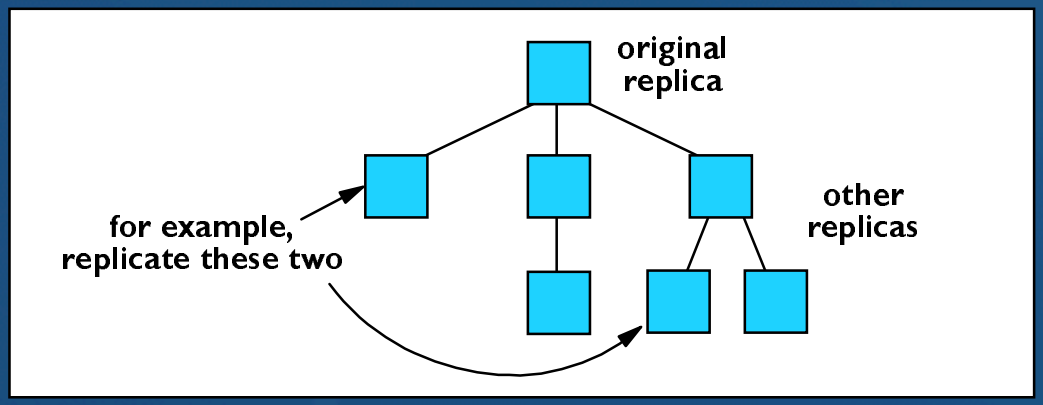
# Simple Replication Example

1. Create a new database and add documents A, B, and C
2. Make a replica of the database
3. Remove B, modify C, and add D to original database
4. Modify C and add E to the replica
5. Replicate the original database and the replica



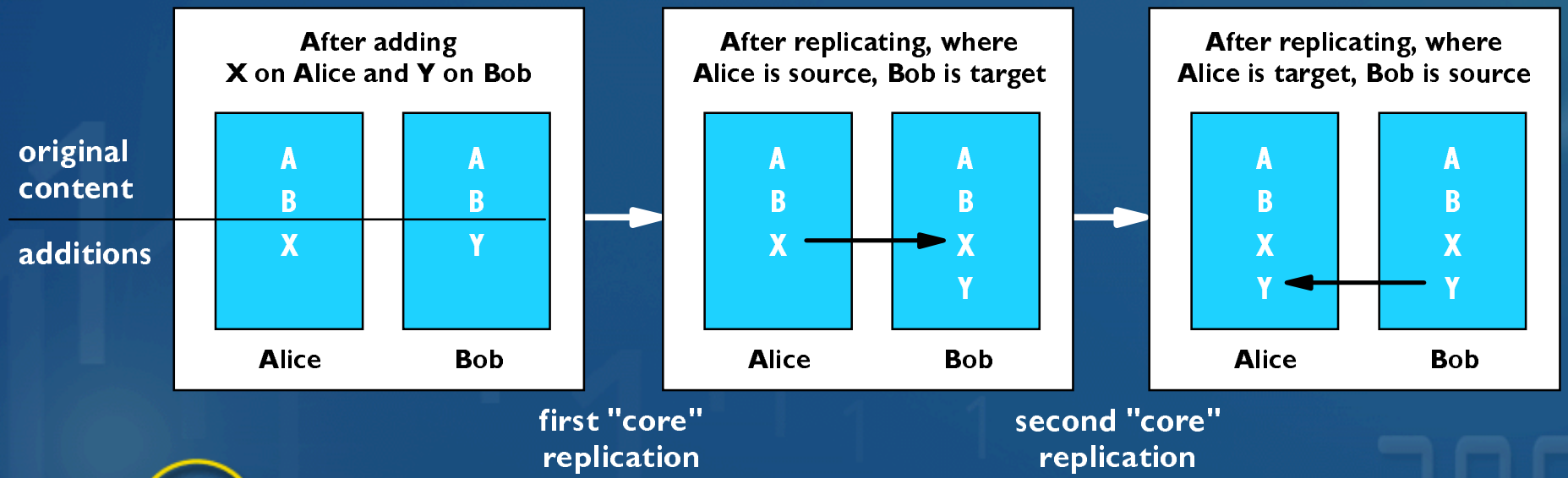
# Notes/Domino Replication Is Versatile

- Others can modify a database while it is being replicated
- Replication can cope with network failures, etc.
- Replication enforces the Notes/Domino security model
- You can fine-tune replication parameters to your needs
  - Specify a selection of notes to be replicated
  - Specify the types of notes to be replicated
  - Specify replication direction(s) and schedules
- Any two replicas of a DB can replicate with each other



# The Core Replication Code Works In One Direction

- It synchronises changes from a source DB to a target DB
- Use it twice to fully synchronise two replicas
- Example: Fully synchronise additions made to replicas on servers Alice and Bob



# Replication Is Incremental At Heart

- Default operation is to efficiently replicates all data that has changed in the source database since the last replication of the source to the target database
  - Uses replication histories to manage this
- Use selective replication to replicate a just a subset of all data that has changed
  - And to cause non-selected notes to be removed from target
  - To save space and/or to speed the replication process
    - A mobile worker can take just the documents needed
    - Headquarters can send different data to different offices

# Replication Histories

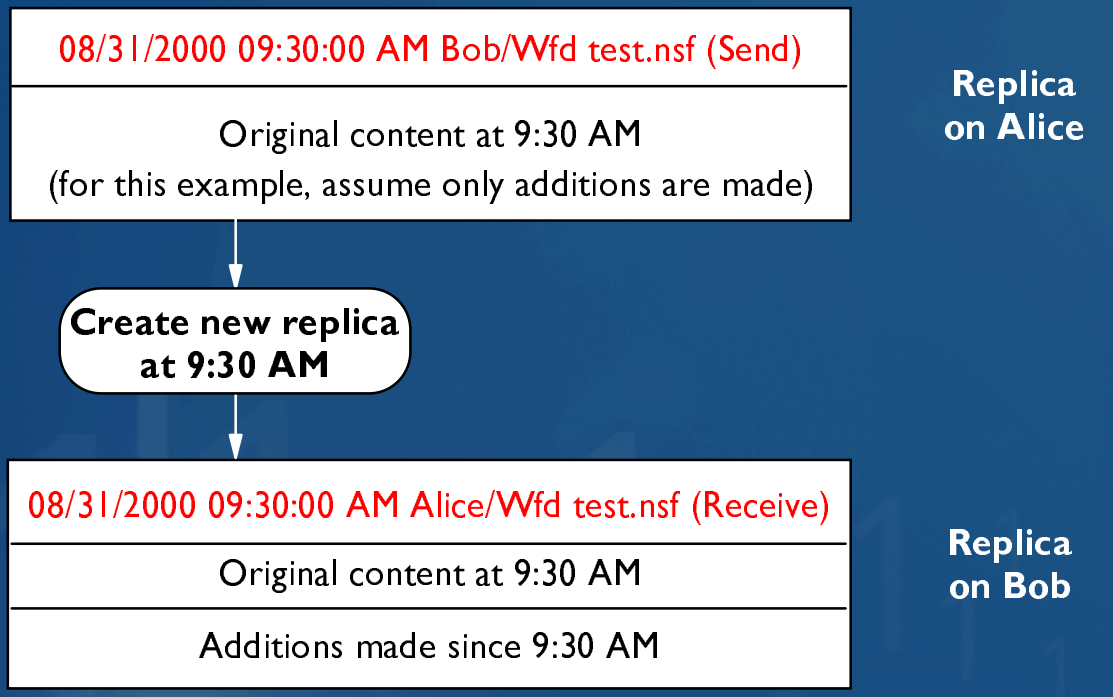
- Used to determine what sources, if any, have changed
- Holds a list of a database's replication activities
  - One entry per server with which the DB has replicated
- Example: Start with a database on server Alice

Original content at 9:30 AM
(for this example, assume only additions are made)

**Replica on Alice**

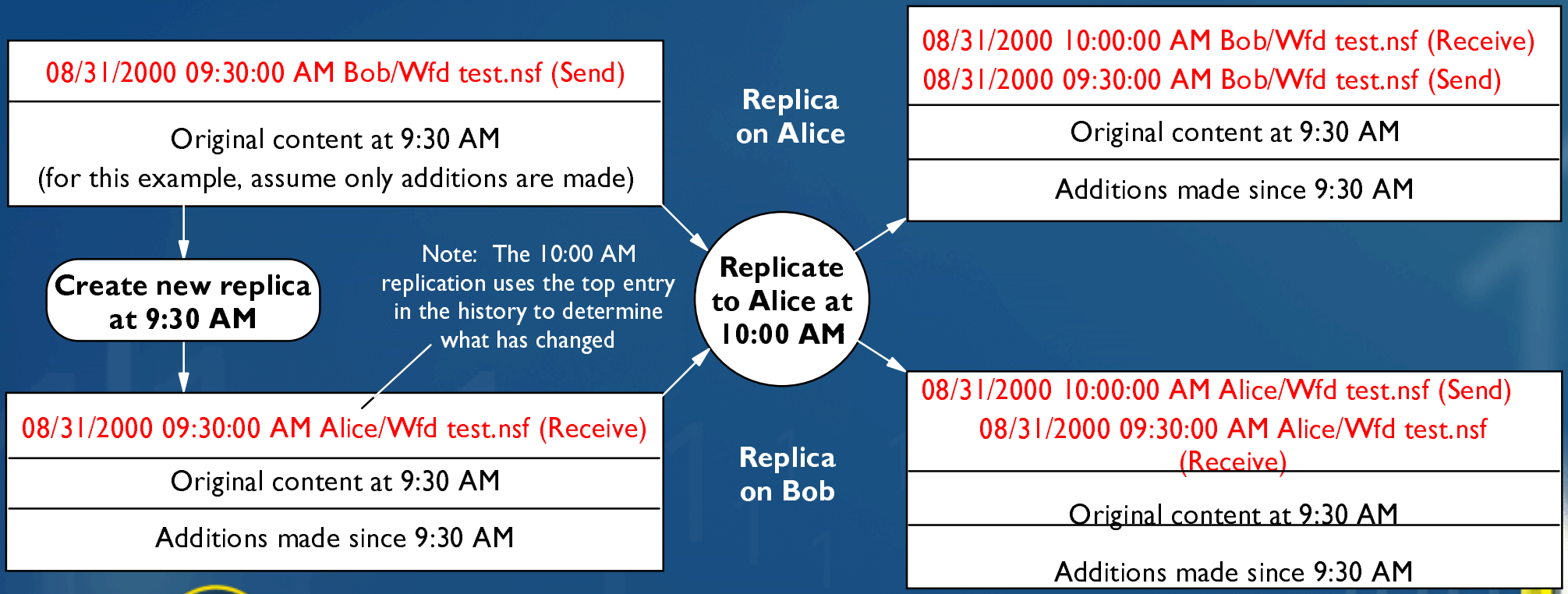
# Example Of Basic Incremental Replication

- Example continued ...
  - Replicate the DB to server Bob at 9:30 AM
  - Modify the replica on Bob



# Example Of Basic Incremental Replication

- Example continued ...
  - Replicate additions back to server Alice at 10:00 AM
    - Use histories to determine what to resync
    - Update histories at the end of the replication



# Be Careful If You Change The System Clock

- Time-based (incremental) replication may appear to fail
- Consider this scenario ...
  - Replicate a database on server A to one on server B
  - Change the system clock backwards on server A
  - Modify data in A's replica
- The next time the database on A is replicated to B ...
  - The modifications will be ignored !!!
    - Their modification times predate the last replication

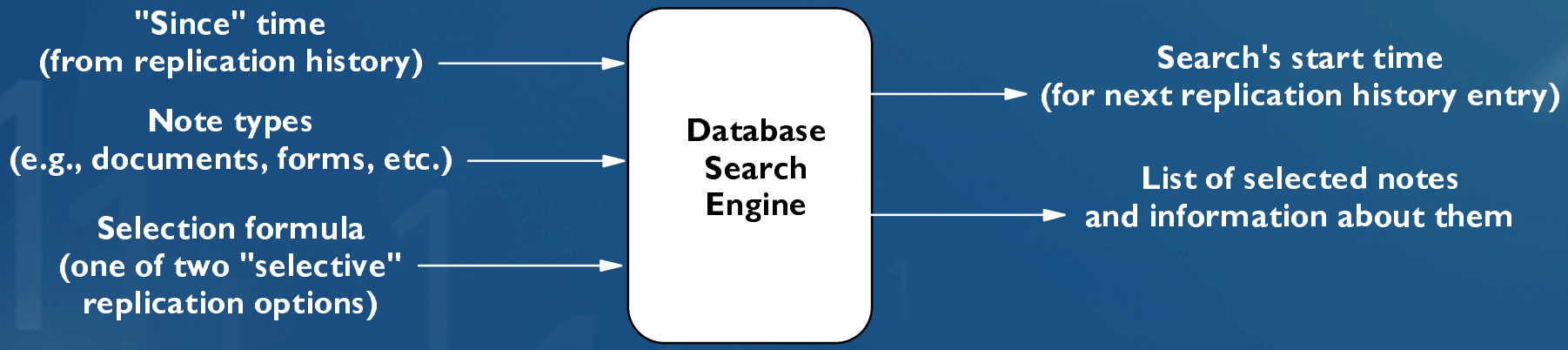
# How To Force A Full (Non-Incremental) Replication

- Clear the replication history in a database
  - For each source DB that then replicates with the database, replication will think that the two DBs have never replicated before and will resync ALL source notes
  - You must have "manager" access to do this

- "Fixup" a database
  - Actually, you don't do this, NSF does it for you
  - Used by NSF to to get replication to restore notes lost during a fixup

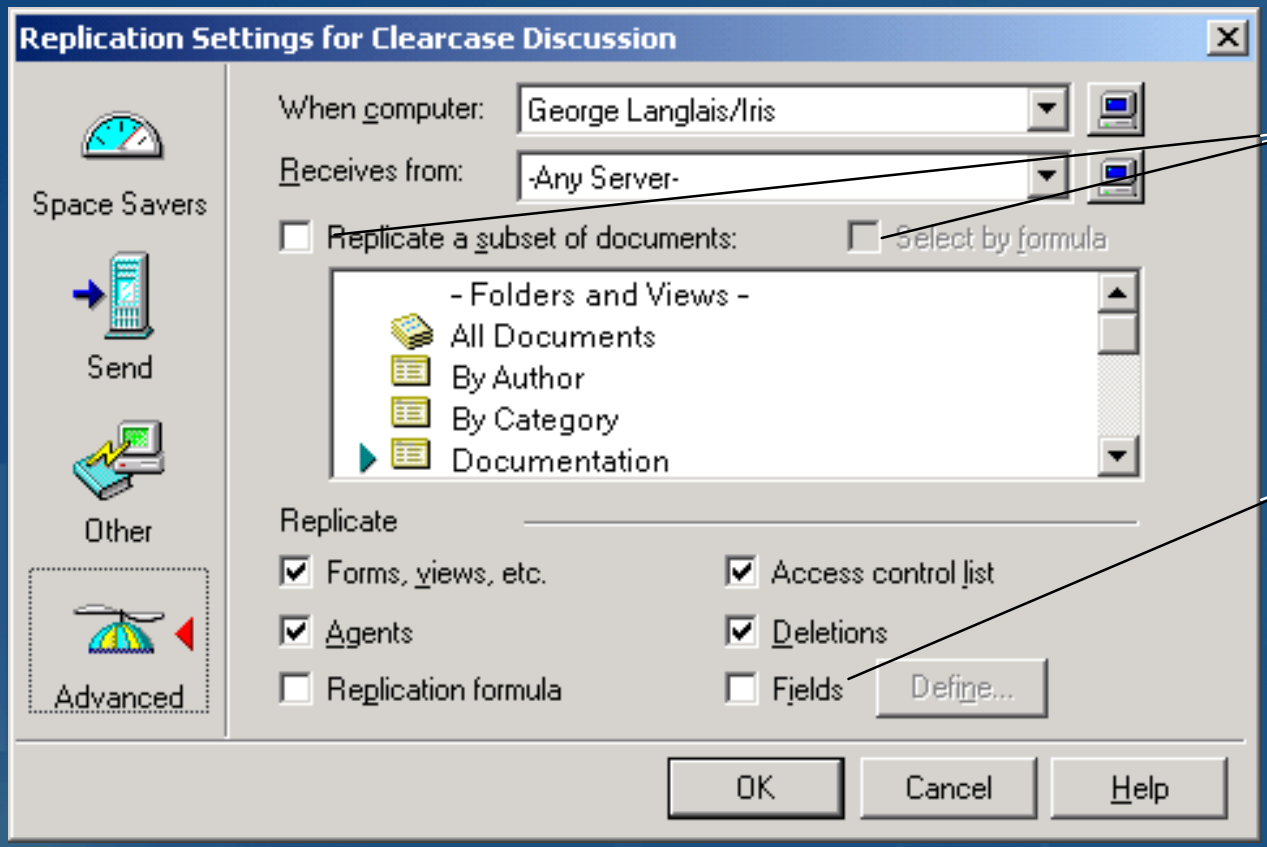
# Database Search Engine Is Used When Selecting Source Notes

- A core service in Notes Object Services (NOS) used to find a specific subset of notes in a database
- Search engine inputs several selection criteria
  - All are optional and are defaulted if not specifically set
- Search engine returns a list of notes and the time it started



# Use Replication Settings To Specify Selection Criteria

- The target DB gets to control what it receives
- Specifies search-engine parameters and other selection factors




Use these to specify a "selective" replication

Use this to limit the fields that replication synchronises

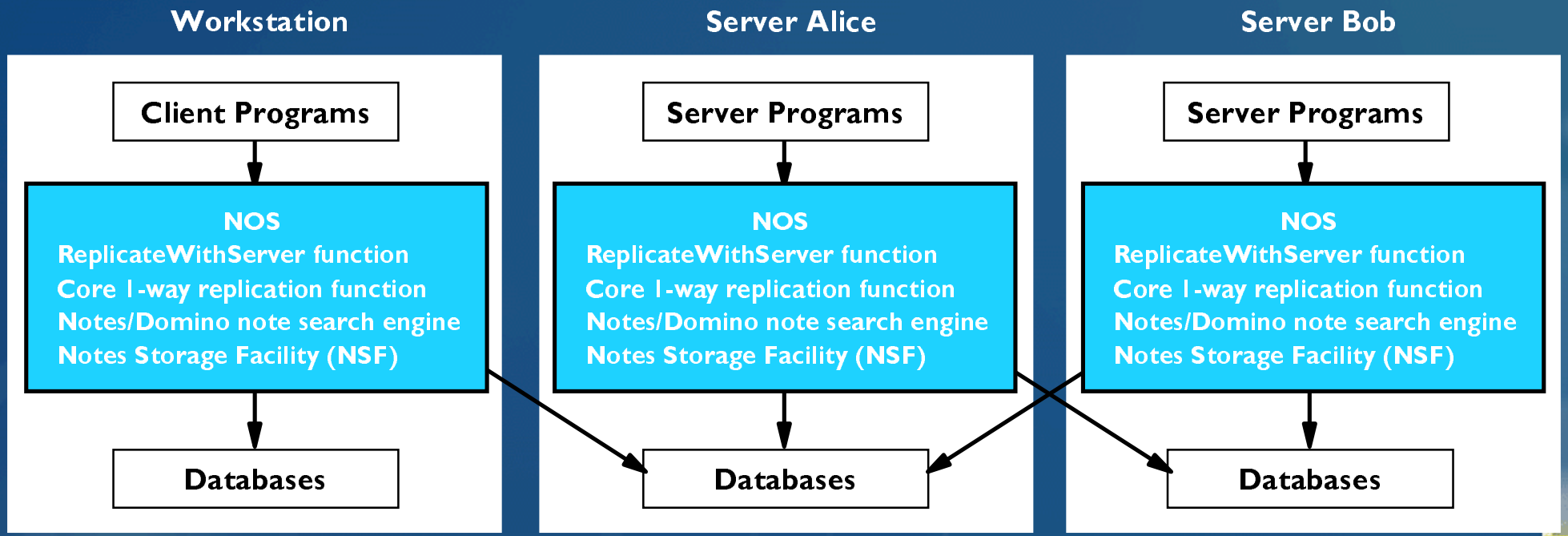
# How A Source Note Is Selected

1. Initial filtering based on replication history and note type
   - Notes added/modified/deleted since the last replication AND note is a document or of type specified in the settings

2. If a "selective" replication is specified, then ...
   - Documents selected in the first step are retained if ...
     - EITHER referenced by specified folders and/or views
     - OR if selected by a search formula
   - Other documents are removed from the source selection

# Replication Programs And Where They Run

- Replicating database between a workstation and a server must be done by programs running on the workstation
  - Because servers cannot access databases on workstations
- Server/server replication can be run on workstations or servers



# Full Replication Options: Pull-Push And Pull-Pull

- Some definitions
  - Pull means that the one-way replication program is running on the computer holding the target database
  - Push means that the one-way replication program is running on the computer holding the source database
- Full replication options available in Notes/Domino
  - Workstations-server: pull-push (all done by workstation)
  - Server-server: pull-push (all done by one server)
    pull-pull (1/2 done by each server)
- Pull-pull replication - one server asks the other to pull data before it itself pulls data the other way

# Pull-Pull Replication Versus Pull-Push Replication

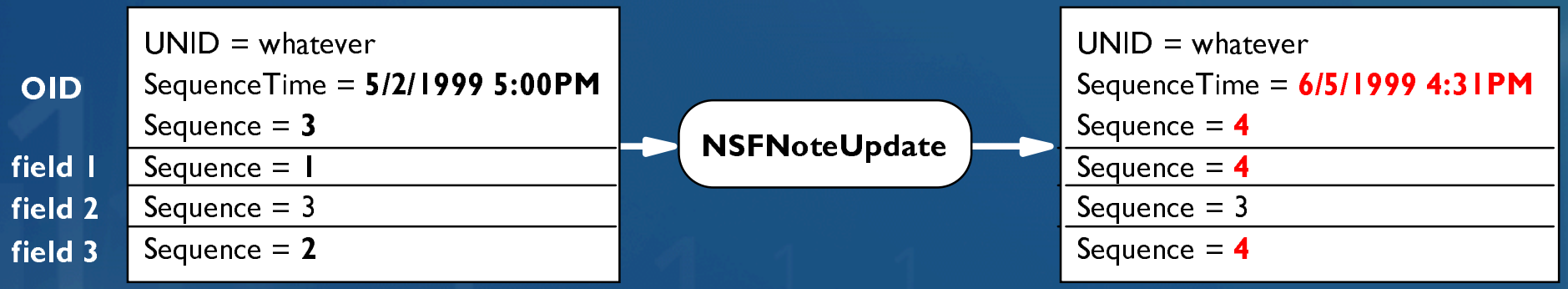
- Pull-Pull pros and cons
  - (Pro) Can optimize network throughput to the extent that the two operations do indeed overlap
  - (Con) Creates two separate logs, one on each server
  - (Con) Some internal setup operations are done twice
  - (Con) Uses two network pipes - may be firewall problems
- Pull-push pros and cons
  - The inverse of the above
- Recommendation: Use pull-push except where pull-pull yields big savings on network utilization

# Primary Data Used By Replication Programs

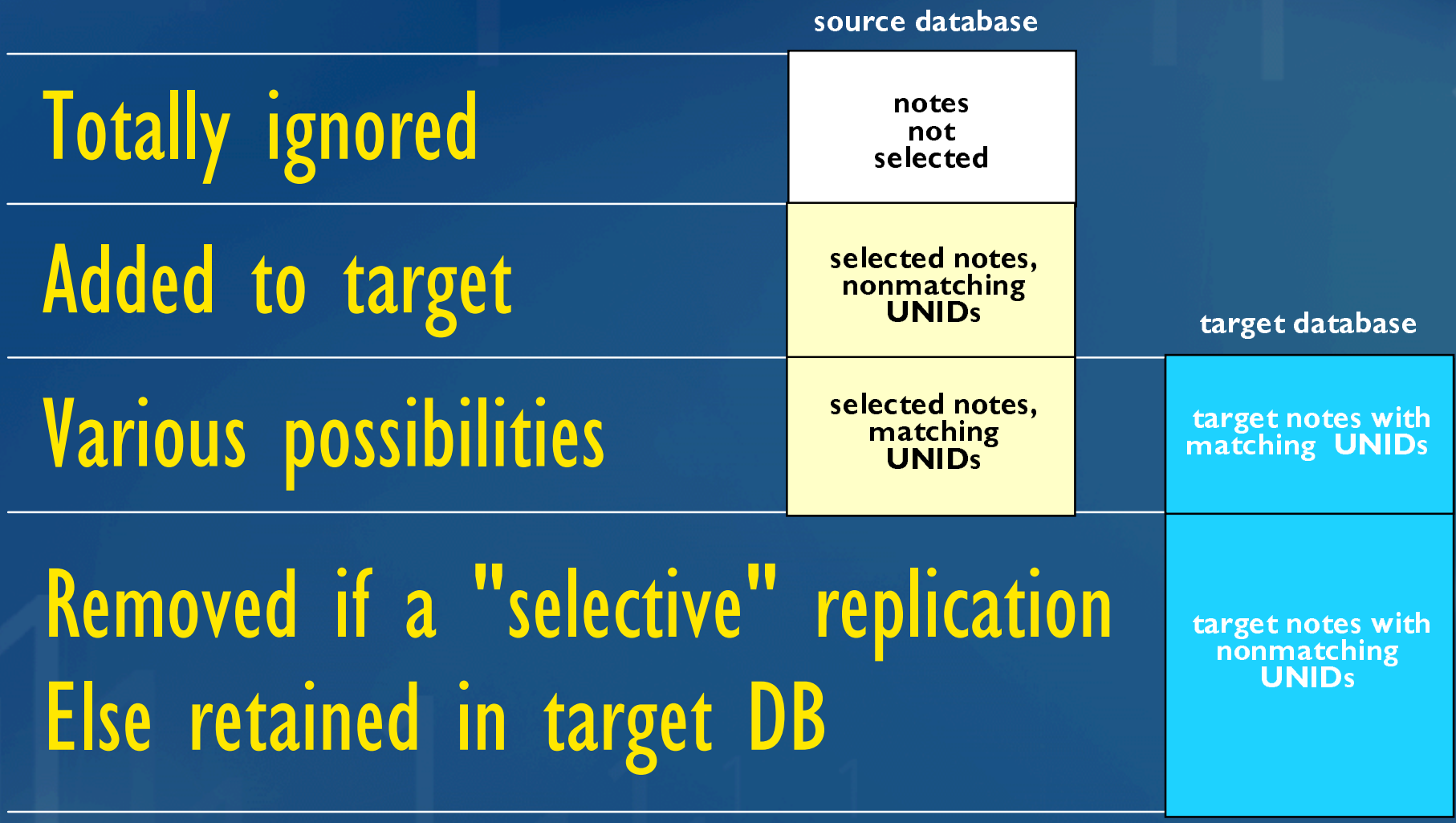
| Location | | Specific Data | How Used |
|---|---|---|---|
| Server | Domino directory | Connection documents | Replication schedules |
| | Database cache | Filenames, replica ID's, etc. | Readily available info about all databases on the server |
| Database | Database header | Replica information structure<br>ReplicaInfo.ReplicaID<br>ReplicaInfo.Flags, etc. | Indicates if one database is a replica of another<br>Miscellaneous replication-setting options |
| Note | Note header | Originator ID (OID) structure<br>OID.UNID (Universal Note ID)<br>OID.Sequence<br>OID.SequenceTime | Indicates if one note is a replica of another<br>Indicates how many times the note has been modified<br>Timestamp indicates when note was last modified |
| | Note fields | Sequence value | Copy of note's OID.Sequence when field was last modified |
| | Special note fields | $Revisions | Ordered list of times when the note was updated |
| | | $ConflictAction (optional) | Indicates field-merging option is desired |
| | | $Conflict | Indicates that the note was a replication-conflict loser |

# NSF Manages Data Needed For Replication To Work

- When a program updates (writes) a modified note, then NSF
  - Sets the note's SequenceTime (in OID) to the current time
  - Increments the note's Sequence value (in OID)
  - Tags each modified field with note's new Sequence value
- Example - write a note having updates to fields 1 and 3



# Overview of The Core One-Way Replication Logic

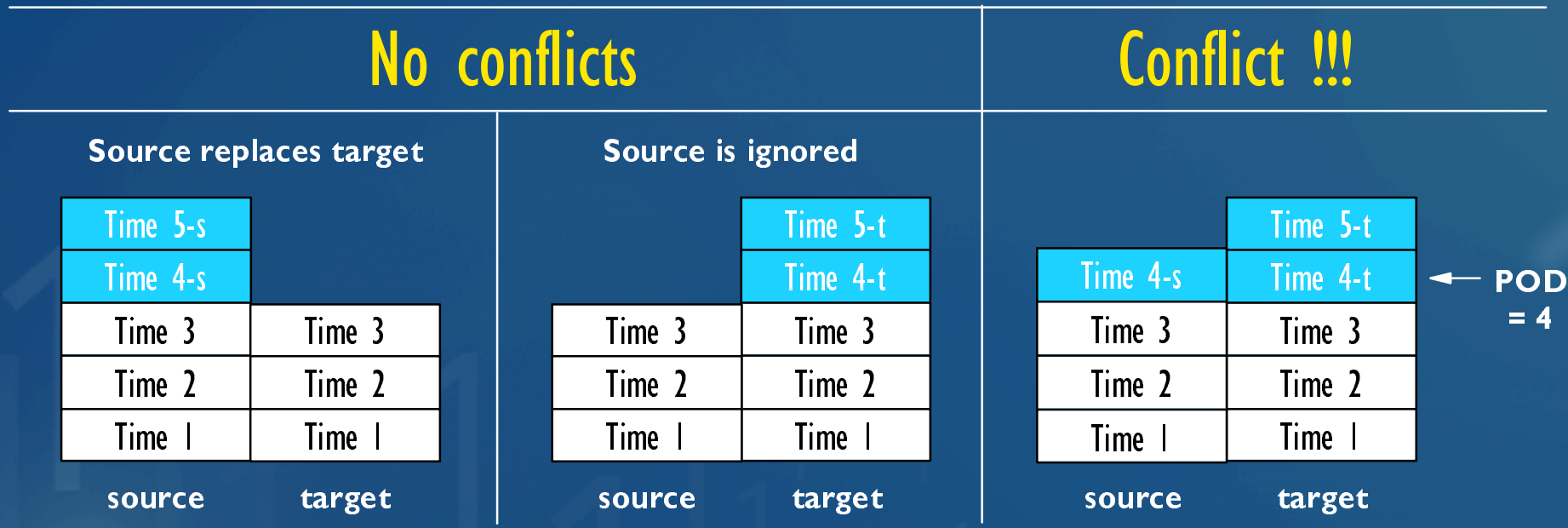


# Overview of The Core One-Way Replication Logic

1. Make two lists: selected source notes and all target notes
2. For each source note ...
   a. If source does not have a matching target note
      - Add new source note to target database
   b. Else if source note has changed and target has not
      - Replace target-note fields with modified source-note fields
   c. Else if both the source note and target note have changed
      - Handle conflict according to target's $ConflictAction field
3. If a "selective" replication ...
   ... then delete nonmatching notes
4. Update replication histories

# Steps 2b/c - Checking For A Source/Target Conflict

- There is a conflict if a "point of divergence" (POD) exists between the $Revisions items of source and target notes
  - A simple test is used - there is a conflict if the top entry in the target's list is not in the source's list




$Revisions lists are compared

# Step 2c - Handling Source/Target Conflicts

- If field merging is enabled and fields can be merged ...
  - Replace target fields with source fields that have changed
- Else determine the winner note
  - If the source note is the winner ...
    - Add the source into the target DB
    - Make the target note a response to the source note and give the target note a new UNID
    - Add a $Conflict field to the target note
  - Else do nothing if the target note is the winner

# Step 2c - Resolving Note Conflicts By Merging Fields

- Target note must have a $ConflictAction field set to 1 AND cannot have a field whose source and target Sequence values are both >= POD (i.e., both have changed)
  - If can merge, replace target fields with source fields whose Sequence values are >= POD
- Examples - assume POD = 4 (see earlier slide)

Cannot merge - conflict on field 2 !!!

| Field | Source Sequence | Target Sequence |
|---|---|---|
| 1 | 4 | 2 |
| 2 | 5 | 4 |
| 3 | 2 | 4 |

Can merge fields - no conflicts

| Field | Source Sequence | Target Sequence |
|---|---|---|
| 1 | 4 | 2 |
| 2 | 5 | 3 |
| 3 | 2 | 5 |

Will replace target fields 1 and 2 with corresponding source fields

# Step 2c - Determining Winner Versus Loser Note

- The note with the bigger OID Sequence value wins
  - That is, the note that has been changed more often
- Uses OID SequenceTime as a tie-breaker
  - The more-recent update is the winner
- Examples

| Conflict Note | Source's OID<br>UNID / Sequence / SequenceTime | Target's OID<br>UNID / Sequence / SequenceTime | |
|---|---|---|---|
| 1 | whatever / 5 / 8-31-00 11:42:22 AM | whatever / 4 / 7-15-00 09:55:41 AM | ← source wins |
| 2 | whatever / 4 / 6-12-00 04:19:10 PM | whatever / 4 / 8-14-00 11:03:22 AM | ← target wins |
| 3 | whatever / 4 / 8-30-00 02:14:12 PM | whatever / 5 / 4-18-00 11:11:14 AM | ← target wins |

# AD113
# Replication - What It Is And How It Works

## Questions
## And
## Answers