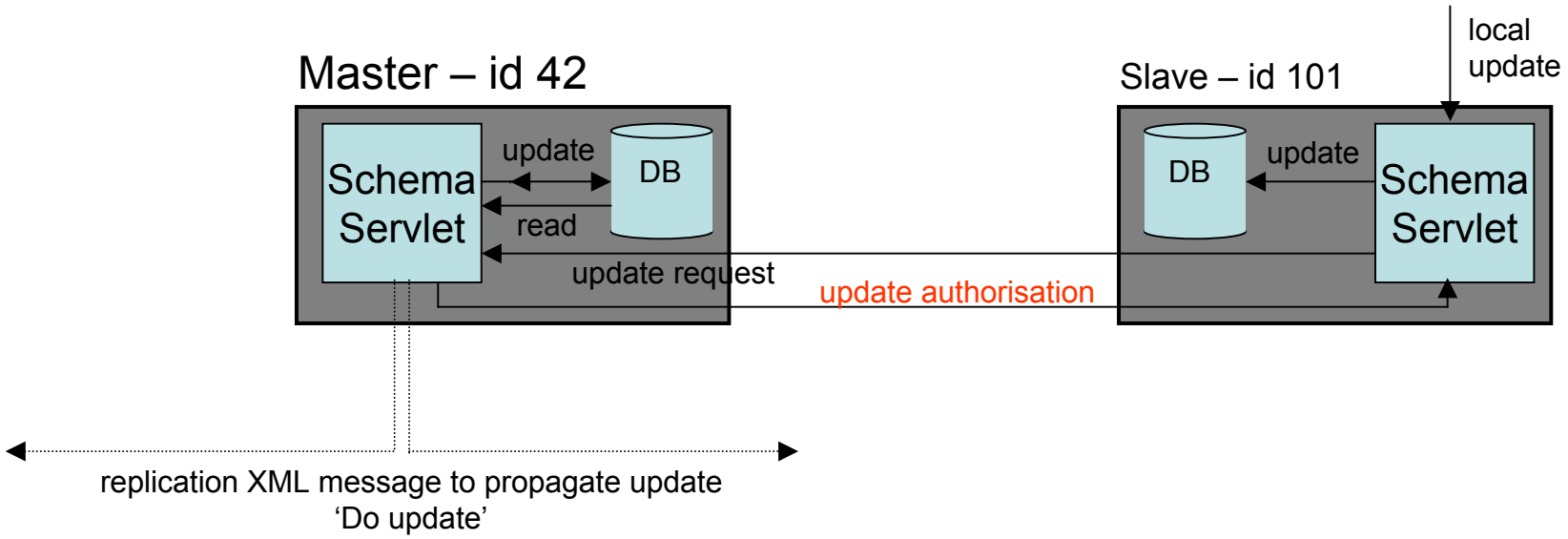
Schema Replication

Schema Replication

- Schema replication to use Bully algorithm
  - Why?
    - On create, must ensure unique table names across all schemas
    - Need a recognised way of maintaining a master schema
    - Can afford the time to validate each new entry in schema
      - Due to low frequency of updates on Schema tables
- One Master schema per VO
  - Bully algorithm votes on master schema, if none exists
  - Master propagates ‘authorised’ updates to remaining schemas
- Queries aimed at ‘local’ schemas
  - Queries handled locally, if possible, to reduce response time
  - Write-thru caching to reduce write times when update authorised

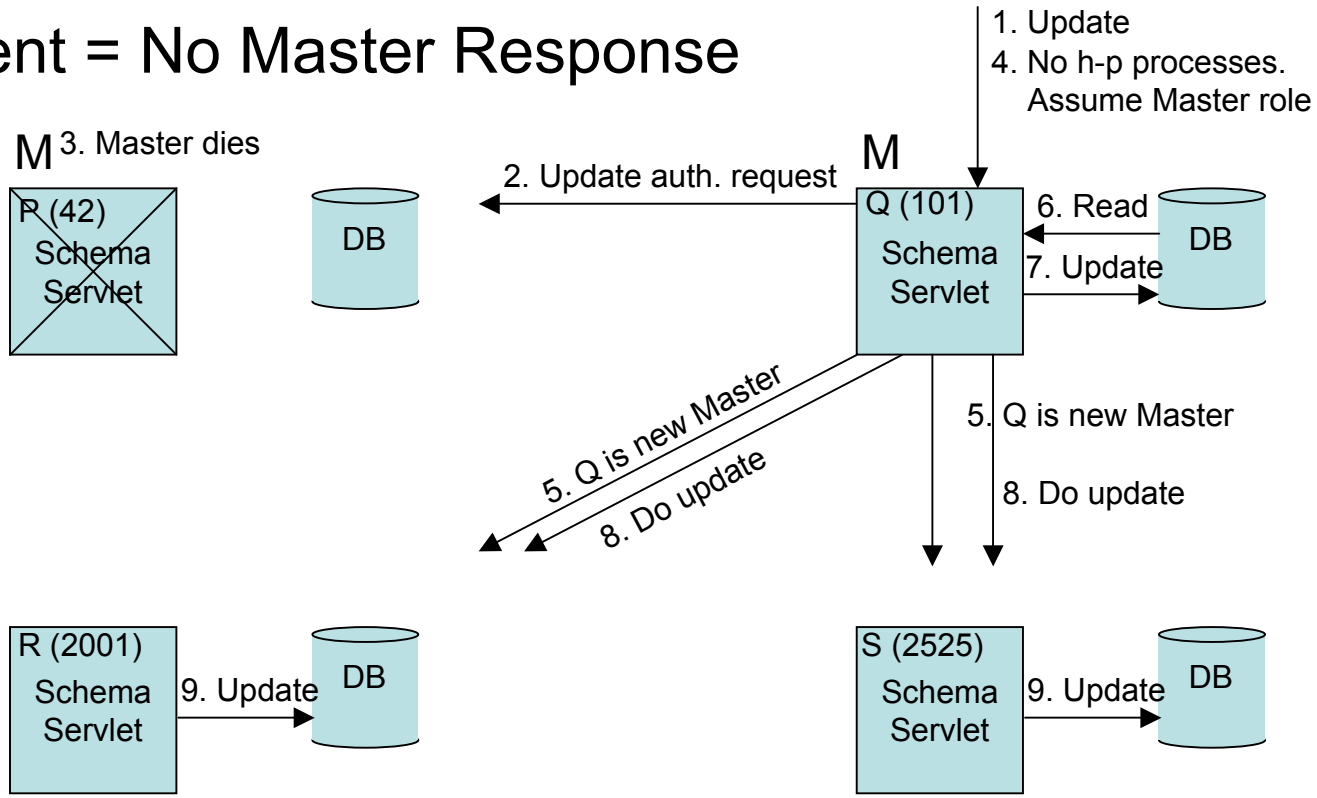
Schema Update and Replication

* Lowest id == highest priority



Bully Algorithm Animated

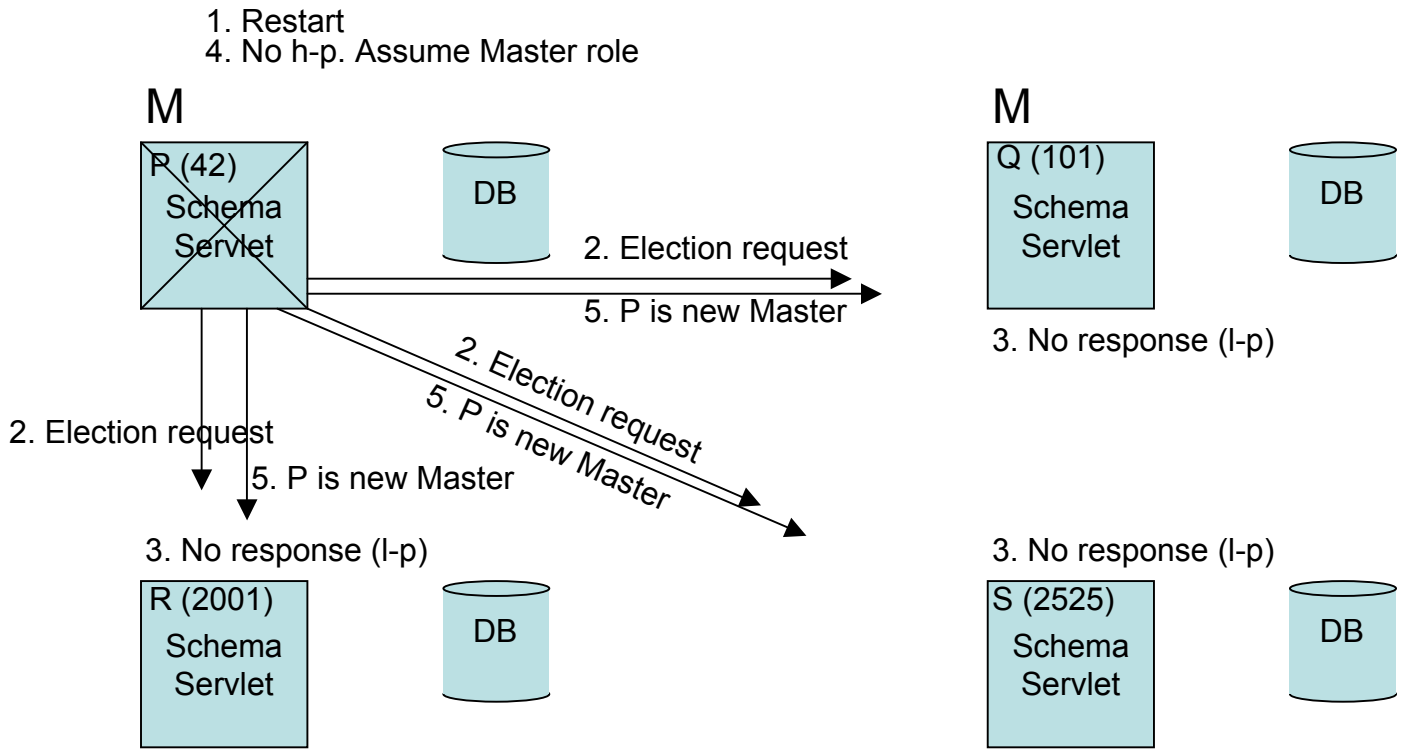
- Event = No Master Response



*h-p == higher-priority process

Bully Algorithm Animated

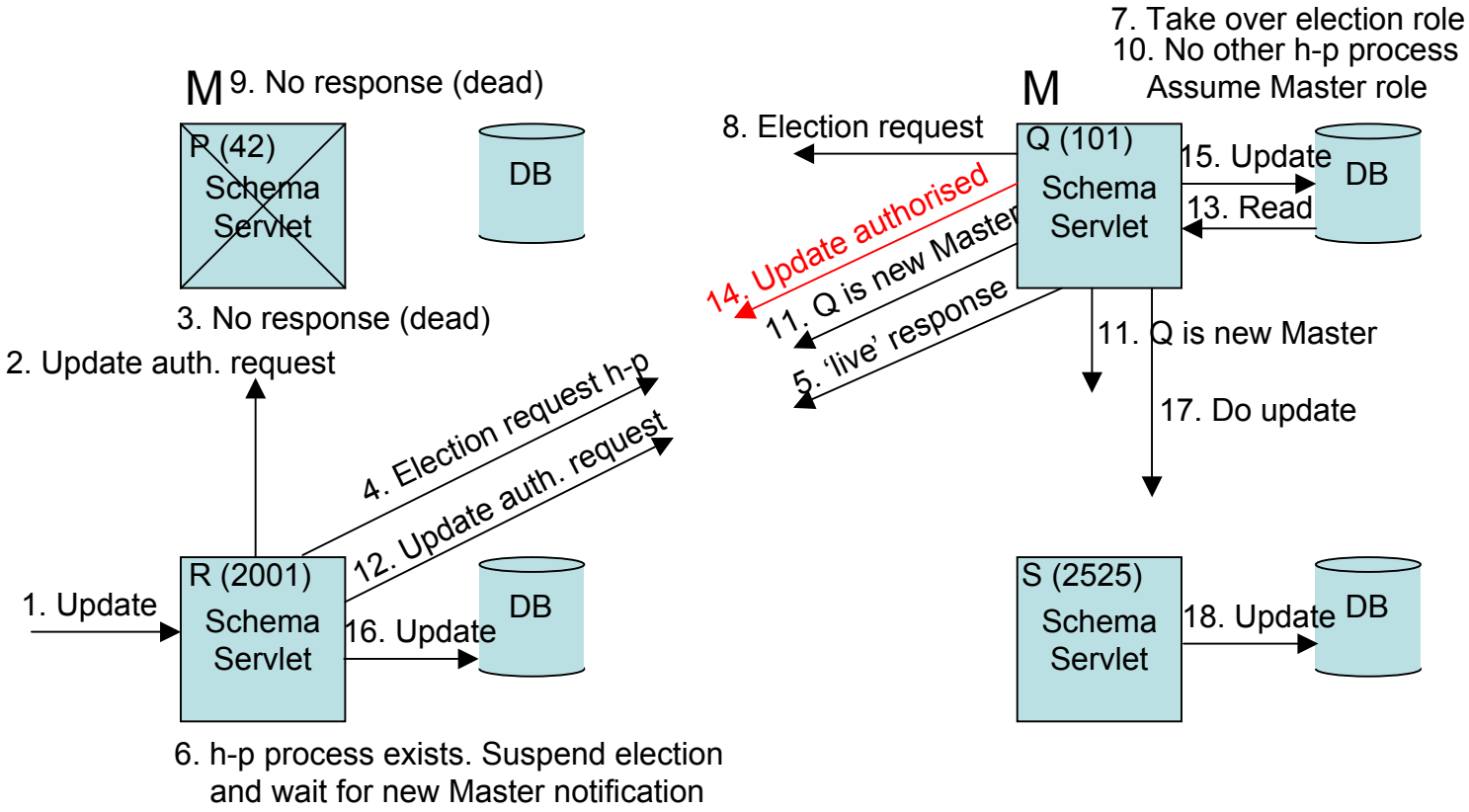
- Event = Restart



*l-p == lower-priority process

Bully Algorithm Animated

- Event = Election Request Received



Bully Algorithm

(Event = No Master Response)
Process P attempts to do an update to the schema DB and notices that the Master is no longer responding to requests, so P initiates an election...

1. P sends an election message to all higher priority processes.
2. If no one responds, then P wins the election and becomes the new Master.
3. If one of the high-priority processes responds, then that h-p process takes over the election and P is done.

(Event = Restart)
If a process P that was previously down comes back up...

1. P holds an election. If P happens to be the highest-priority, then P wins the election and takes over the coordinator's job.

(Event = Election Request Received)
When a process receives an election message from a lower-priority process...

1. The receiver sends an 'ok' message back to the l-p process, indicating that it is alive and will take over the election process.
2. The receiver holds an election, unless it is already holding one.
3. Eventually all processes give up except one, the one with the highest-priority.
4. The new Master announces its victory by sending all other processes a message telling them that it is the new Master.

Affected Classes

- SchemaServlet*
  - Changes to doGet() and add Bully algorithm logic on createTable and synchronisation between schemas
- SchemaReplicaCreator+
  - Extract schema table data to replicate, based on certain rules
- SchemaReplicaEncoder+
  - Translate replication data to XML ready to post
- SchemaReplicaSender+
  - Send HTTP-post XML replication message to remote schemas
- SchemaReplicaReceiver+
  - Receive HTTP-post XML message of replication data
- SchemaReplicaStorer+
  - Decode XML data to internal structure to record replication data via RDBMS

*Changes to existing class
+Add new class

Proposed Code Changes

- SchemaServlet
  - In SchemaServlet() constructor
    - Initialise schemaElectionId, masterSchemaLocation, isMasterSchema, electionInProgress
    - Each time servlet starts…
      - Call electMasterSchema() to force an election
      - Call SynchroniseSchemaDB()
      - If isMasterSchema is false, then contact Master and synchronise DB state.
      - If isMasterSchema is true, then contact all Slaves and merge DB state.
        - » This could be tricky!
        - » What to do with inconsistencies? Vote on latest entry to win? Don't delete if unsure
  - In doGet()
    - Add masterNotification(location)
      - Receives location of who is the new master and record this
    - Add electionRequest()
      - Respond to a lower-priority process and take over election process
  - In createTable(…)
    - If isMasterSchema is true, then attempt table create
      - May fail on duplicate table – handle error gracefully
      - Read current schema list and apply update to each one
        - » Authorisation given if isMasterSchema is true
    - Else isMasterSchema is false, so get permission for table create
    - If remote Master is 'dead' force an election to get a new master appointed

Proposed Code Changes

- SchemaServlet
  - electSchemaMaster()
    - Initiate an election process
    - electionInProgress = true
    - Build a list of known higher-priority schemas
      - Each schema keeps full list of know schemas
      - Build ‘live’ list and record each entry’s priority id
        - » A schema’s (unique and consistent) priority can be derived from the hashcode of its location string
    - Poll each h-p process for an ‘alive’ response
      - No response means that this process is the highest priority process
        - » This process is now the master (masterSchemaLocation = this.schemaLocation)
        - » Synchronise state with all other known schemas
        - » Tell all the other processes the location of the new master
      - A response means that another process has a higher priority
        - » electionInProgress = false (another h-p process initiates an election)
        - » Eventually the h-p process will assume the role of Master

Some Food for thought

- Propagation of table updates
  - Need to ensure that all updates are carried out, even for postponed updates for schemas that are *down* when the 'Do update' message was sent. (See next bullet point).
- Consistent state on start/restart
  - If a schema starts/restarts, then it must synchronise its internal state with that of the other schemas
    - Same consistency rule applies for potential Master, as well as Slaves.
      - If Master, then contact all Slaves and synchronise with them
        - » Voting scheme required in case of inconsistency between slave's state
      - If Slave, then contact Master and synchronise with it
    - Only then can the restarted schema be 'alive and ready'
- Security
  - Validation of registries and schemas during replication
    - Do we know and/or trust who we're sending to?
  - Adding a new registry or schema to the VO
    - Is this new registry or schema known and/or trusted?

To do list

- Sort out registry replication first!
  - Decide on which version to use
    - Replicate only new/changed data
    - Replicate all data regardless
  - Decide on which checksum mechanism to use
- Check-in draft new schema replication classes
- Change schema replication classes in-line with recent changes to registry replication classes and schema table changes
- Integrate draft Java bully-algorithm logic into SchemaServlet code
- Plug remaining gaps in bully logic for schema table updates
  - And propagating those updates (even if schema misses an update)
- Generic table readers for registry and schema to build replication data
- Security considerations on adding new registrys/schemas
- Testing, testing, testing
  - Can make use of existing Junit unit tests for building registry replication messages, as a basis for testing schema replication class methods
  - Need new system testing for different Schema (Master/slave) replication model

The End