

## **Z-Merge – Keep Zarafa connected to the world**

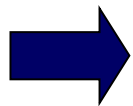
Sebastian Kummer; [s.kummer@zarafa.com](mailto:s.kummer@zarafa.com)



## Why synchronization is necessary?

*Multiple databases containing same information*

- **Zarafa:** e-mail, calendar, contacts, tasks
- **CRM/ERP-Systems:** customer contacts, archived e-mail, calendar (appointments, meetings), tasks
- **DMS/Project management tools:** files(attachements,tasks, meetings)



*Customers and Users use functionality of all systems trying to keep all data synchronized*

## How it works today

### *Using Outlook as mediator*

- Outlook is running on the client's machine
- 3rd party application plugin installed inside Outlook
- Synchronization is done by the plugin depending on the correct use by the end-user

### *Conceptual problems*

- Blocks user of using other clients (Webaccess or Thunderbird)
- Doesn't work correctly with server-side profiles (Outlook on different machines)
- Doesn't work if user not logged in

### Using Outlook + OL-Plugin



## **The idea behind it**

**Give the user an intuitive and easy way to synchronize and to archive emails and attachments from Zarafa.**

**→ From the Webaccess today!**

## Approach via Z-merge

### Combine or select one of both

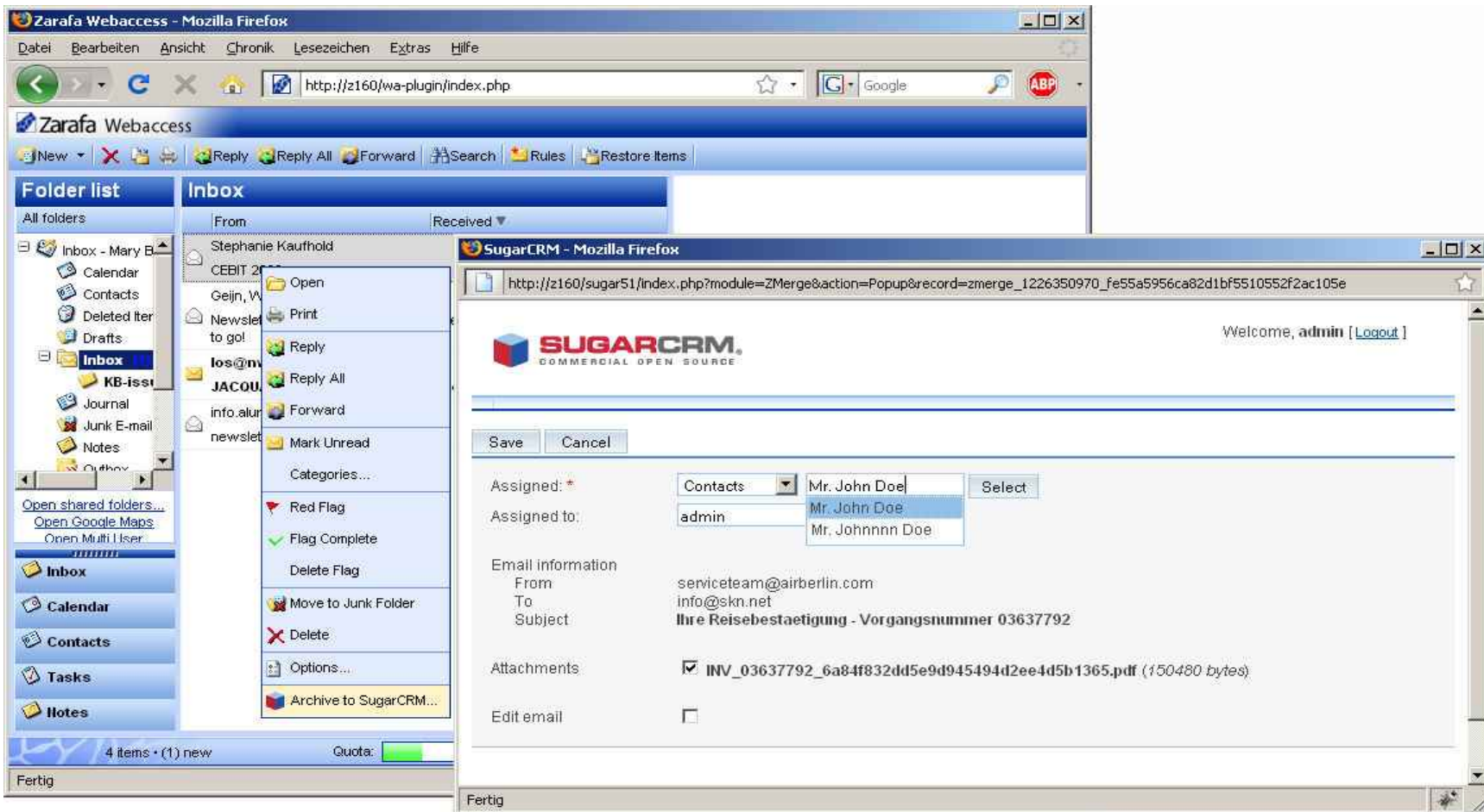
- A One time-one way synchronization which depends on user interaction (*Z-Merge-Archive*)
  - *Archiving email and/or attachments to other applications*
  - *Using Webaccess Plugin System – ZMA plugin*
  
- Services in the background with continuous synchronization (*beta4*)
  - *Contacts*
  - *Appointments*
  - *Tasks*
  - *Notes*

## Archiving Email or Attachments

- Right click on the email
  - Archive to .....
  - Send it to a external interface
- Works in the **Webaccess**



# How it looks...



The image displays two overlapping browser windows. The background window is titled "Zarafa Webaccess - Mozilla Firefox" and shows an Outlook-style interface. The address bar contains "http://z160/wa-plugin/index.php". The main content area shows an "Inbox" with a list of emails and a context menu open over one of them, offering actions like "Open", "Print", "Reply", "Reply All", "Forward", "Mark Unread", "Categories...", "Red Flag", "Flag Complete", "Delete Flag", "Move to Junk Folder", "Delete", "Options...", and "Archive to SugarCRM...". The status bar at the bottom indicates "4 items · (1) new" and "Quota: [progress bar]".

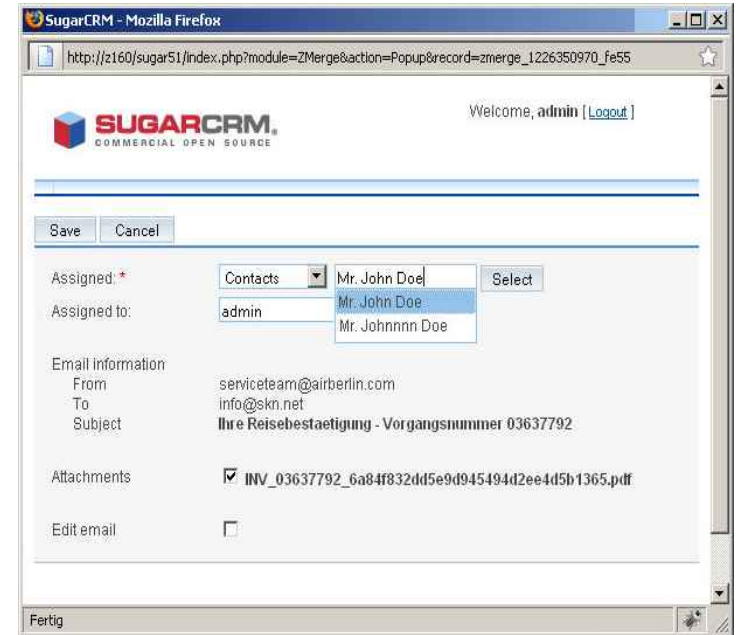
The foreground window is titled "SugarCRM - Mozilla Firefox" and shows the SugarCRM interface. The address bar contains "http://z160/sugar51/index.php?module=ZMerge&action=Popup&record=zmerge\_1226350970\_fe55a5956ca82d1bf5510552f2ac105e". The page header includes the SugarCRM logo and "Welcome, admin [Logout]". The main content area shows a form with "Save" and "Cancel" buttons. The "Assigned:" field is set to "Contacts" and "Mr. John Doe". The "Assigned to:" field is set to "admin". The "Email information" section shows "From: serviceteam@airberlin.com", "To: info@skn.net", and "Subject: Ihre Reisebestaetigung - Vorgangsnummer 03637792". The "Attachments" section shows a checked checkbox next to "INV\_03637792\_6a84f832dd5e9d945494d2ee4d5b1365.pdf (150480 bytes)". The "Edit email" checkbox is unchecked. The status bar at the bottom indicates "Fertig".

# Components commitment

- Uses Webaccess Plugin System
- Handover of user
- E-Mail data
- Open pop-up



Configured on the Zarafa System



This has to be done by the (integrator of) a 3rd party software

## Integration on the 3rd party site

- Interface that takes the user into the system
  - Possibly reuse existing interfaces
  - Receive the data (RFC822 message)
  - Allow user to select
    - Projects
    - Contacts / Accounts
    - Etc.
- Archive the data

**Interface integrated in the Z-Merge SugarCRM connector!**

## Approach via Z-merge

### Combine or select one of both

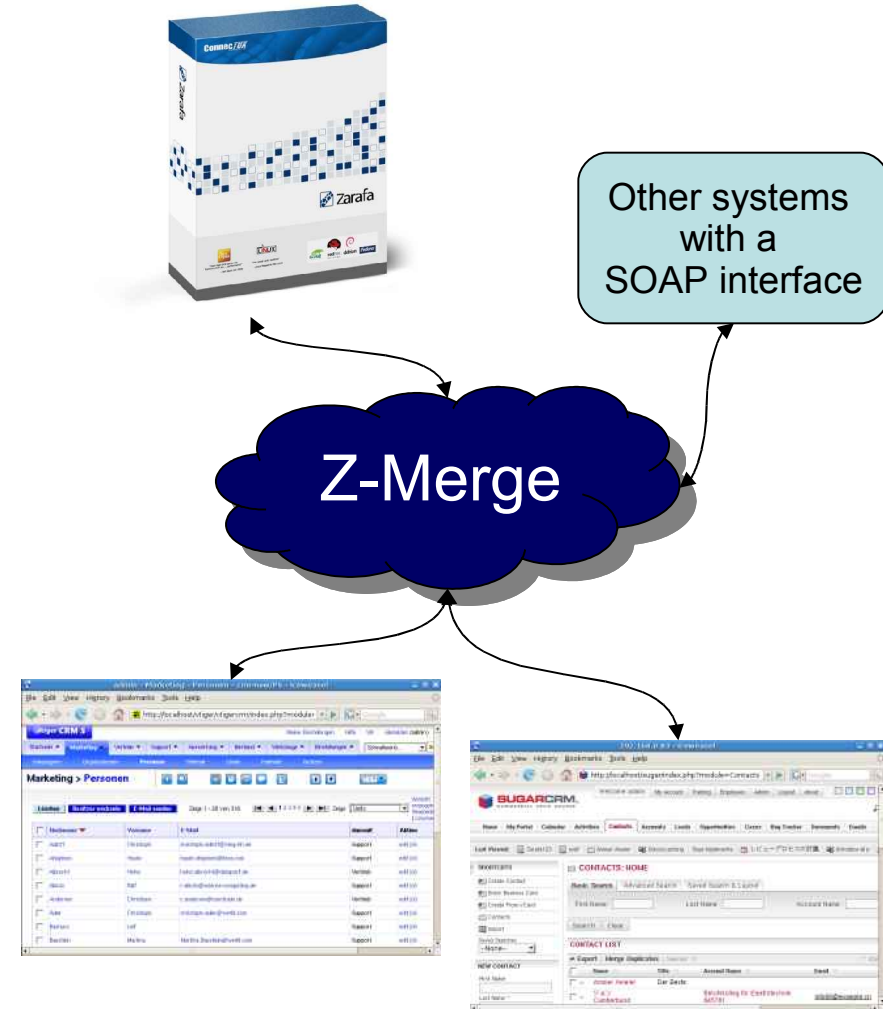
- One time-one way synchronization which depends on user interaction (*Z-Merge-Archive*)
  - *Archiving email and/or attachments to other applications*
  - *Using Webaccess Plugin System – ZMA plugin*
  
- B** Services in the background with continuous synchronization (*beta4*)
  - *Contacts*
  - *Appointments*
  - *Tasks*
  - *Notes*

## Goals for the background synchronization

- Reliable synchronization
- No user knowledge
- Without interaction
- Synchronize more than 2 servers/databases
- Manage big databases (>1000 contacts)
- Avoid instability problems with “connector-applications” in Outlook

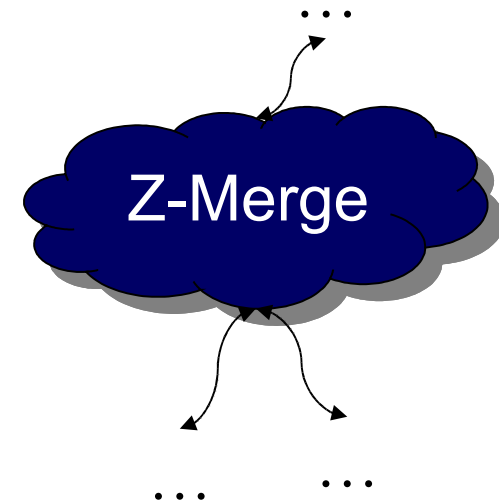
## Why Server-to-Server synchronization is better

- Outlook as connector is not longer necessary
- Synchronization done without user interaction
- Changes propagate immediately to all connected servers (<1 sec)
- All applications are working on the same data
- Compatible with all systems with a customizable SOAP interface



## How does it work?

- Changes propagate individually to each connected server
  - If server goes down, changes aren't lost and will be applied when the server is back online again
  - Server availability "watch-dog" → Notifies administrator when server goes down with status messages
- ObjectMapping is done by the agent
- User/Group mapping could be implemented for individual solutions



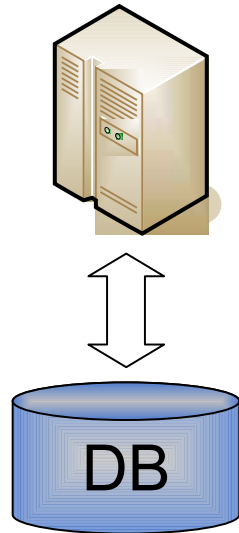
## How does it work (2) ?

- Constant communication with all servers
  - Gets, transforms and sent changes
- When one server goes down, the others stay in sync
  - Checks the status of the offline server continuously
    - If the server is back online, so called “offline-sync” is performed

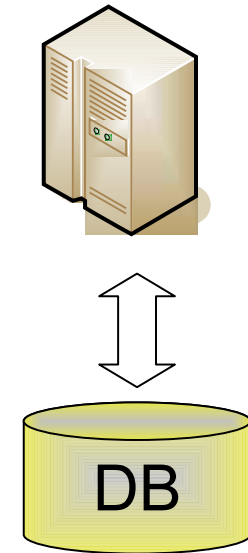
## Data storage

Each application works with its own data

 Zarafa



**Z-Merge saves only references and state informations**



## Status and supported versions

- Available in beta4
- Several testdrives on customers with SugarCRM
  
- Zarafa version 5.20 and later
- SugarCRM version 4.5.1, 5.0 and 5.1

Available as debian packages and tarballs

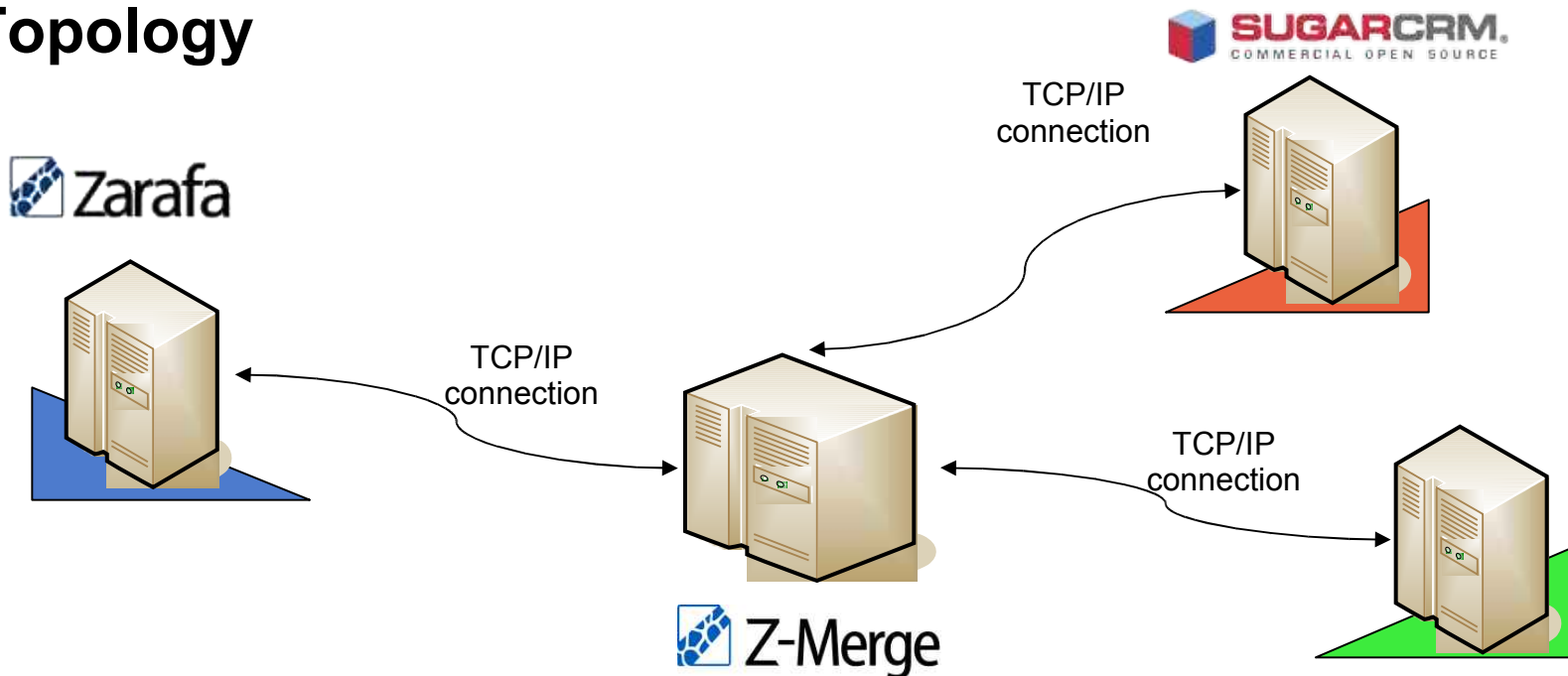
## Completely without user interaction?

*In special cases user interaction is necessary:*

- Conflict resolution
  - When one (or more systems) are not in livesync, they could present changes in conflict with other changes.
  - User should decide which data to keep.
    - As there is no interaction, objects get **duplicated!**

## Technical stuff

## Topology



Could be located on one system or on one system itself, depending on your necessities

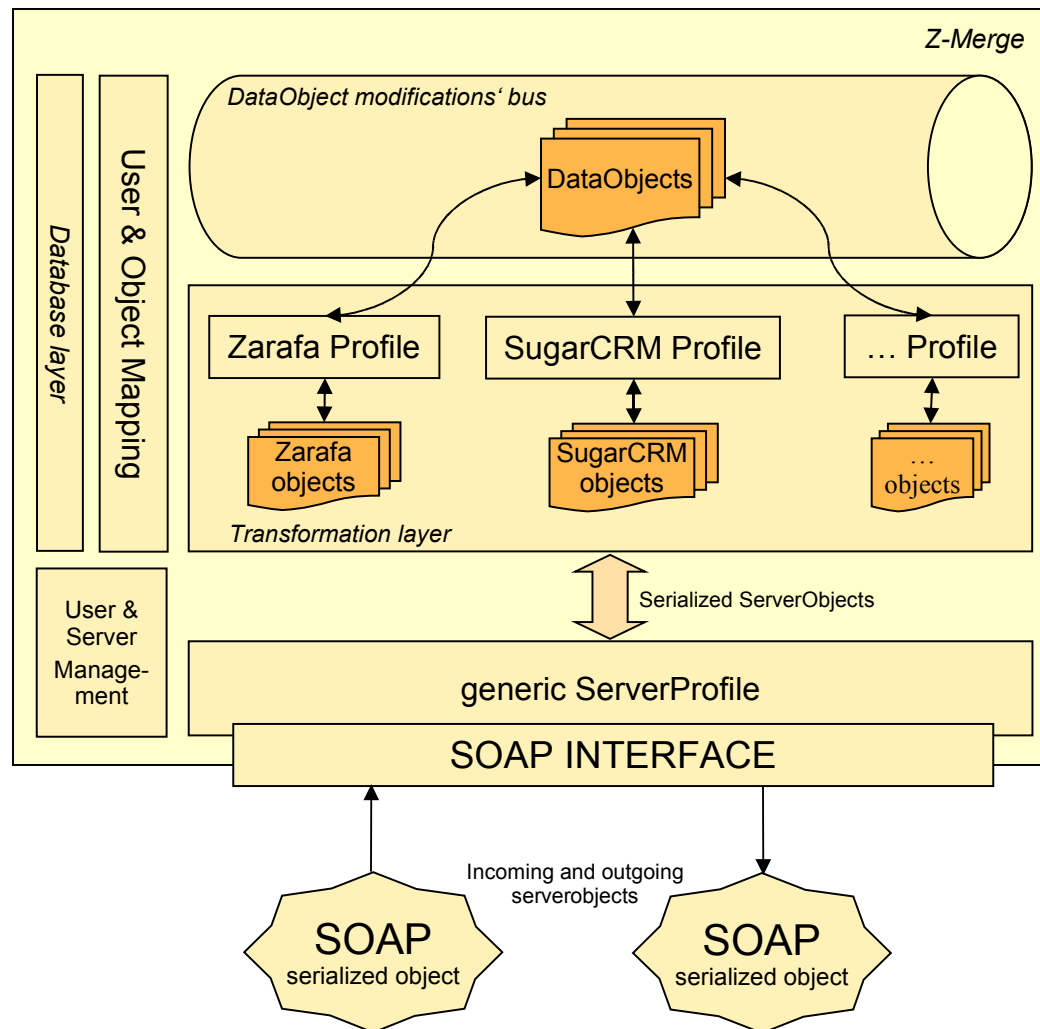
Just needs a stable TCP/IP connection between the Agent and the Connectors

## Z-Merge internals

- Written in PHP-CLI
  - *easy development of your own conversion classes*
- Communication, SOAP, conflict-resolution, mappings, distribution running in the background
- Integrate a new applications by implementing only 4 methods

## Z-Merge AGENT

- Constant connection to all servers
- Receives and sends serialized 3<sup>rd</sup> party objects
- Abstracted data structure in so called DataObjects
- Bi-directional transforming of 3rd-party and Z-Merge DataObjects
- DataObjects have common user, group and object definitions
- 3rd-party components support group to multiple user object mapping



## Non-invasive integration on the application side

- No data transformation necessary
- Very lightweight
- Only object (un)serialization and receiving/sending
- Own SOAP interface recommended

## **Connect applicationX to an Z-Merge infrastructure**

### **Implement a Z-Merge connector that supports**

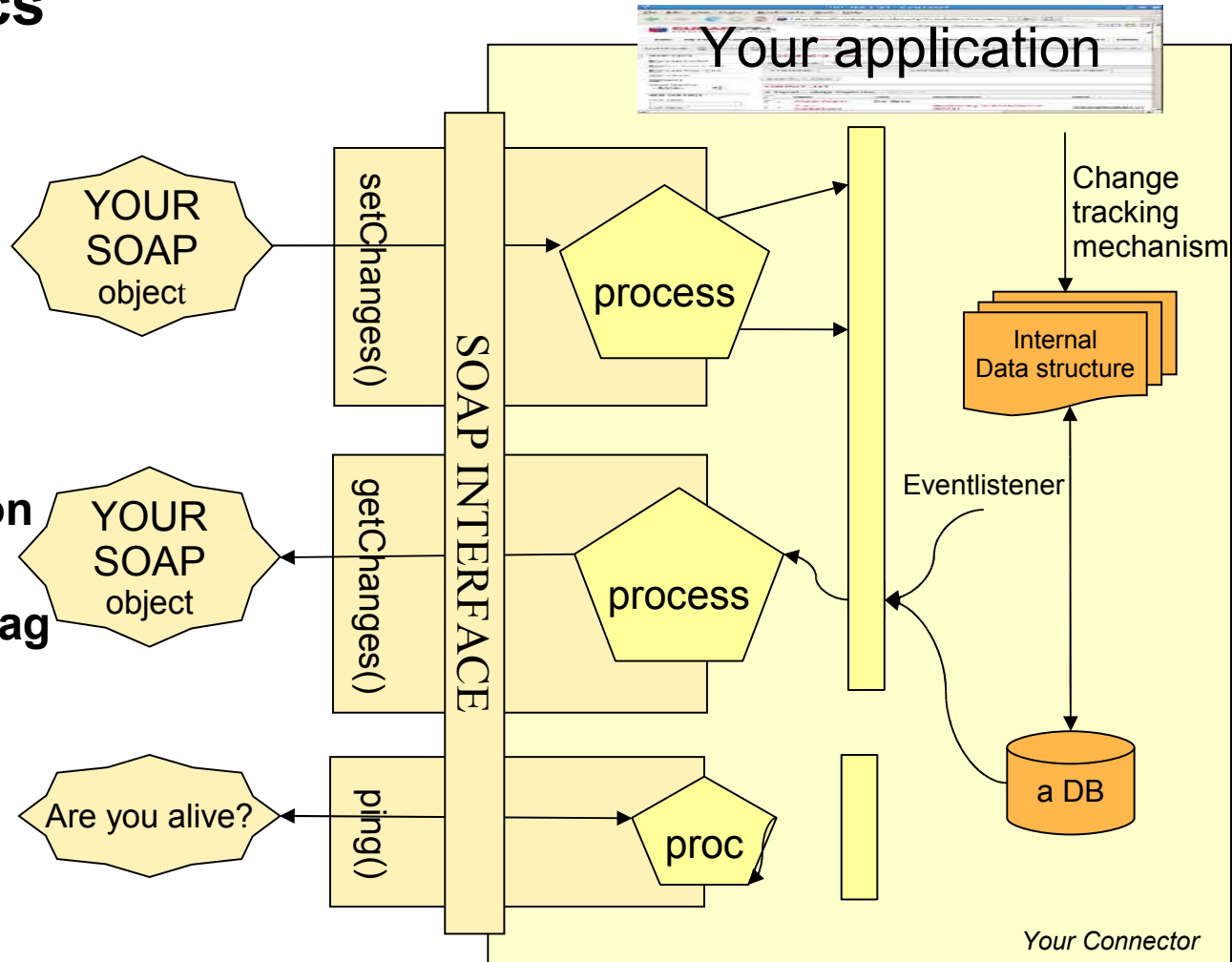
- e-mail, calendar, contacts, tasks, notes

### **Connector must/should:**

- provide a simple SOAP interface
- support long executed HTTP calls

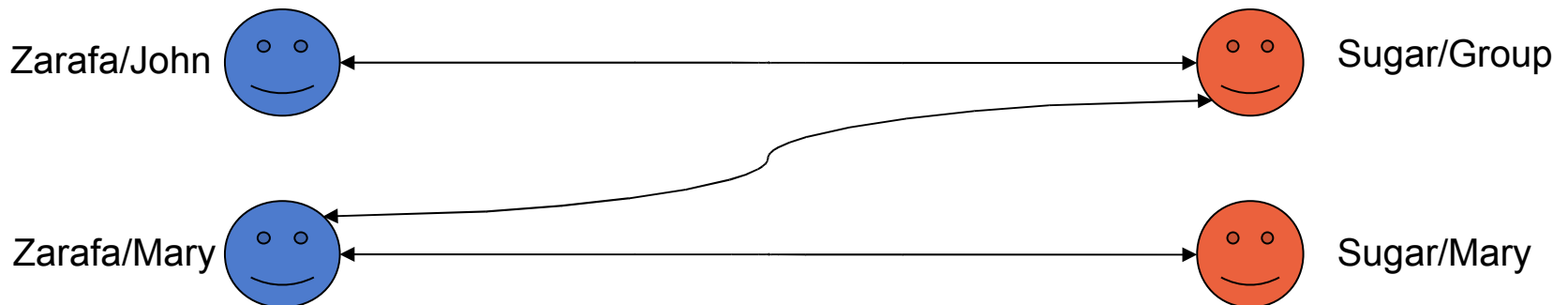
## Connector basics

- Three methods to implement
  - `getChanges()`
  - `setChanges()`
  - `ping()`
- Standard protocol
  - Last modification number
  - MoreChanges flag
  - User id/name
  - Object id
- Data as own structure/protocol

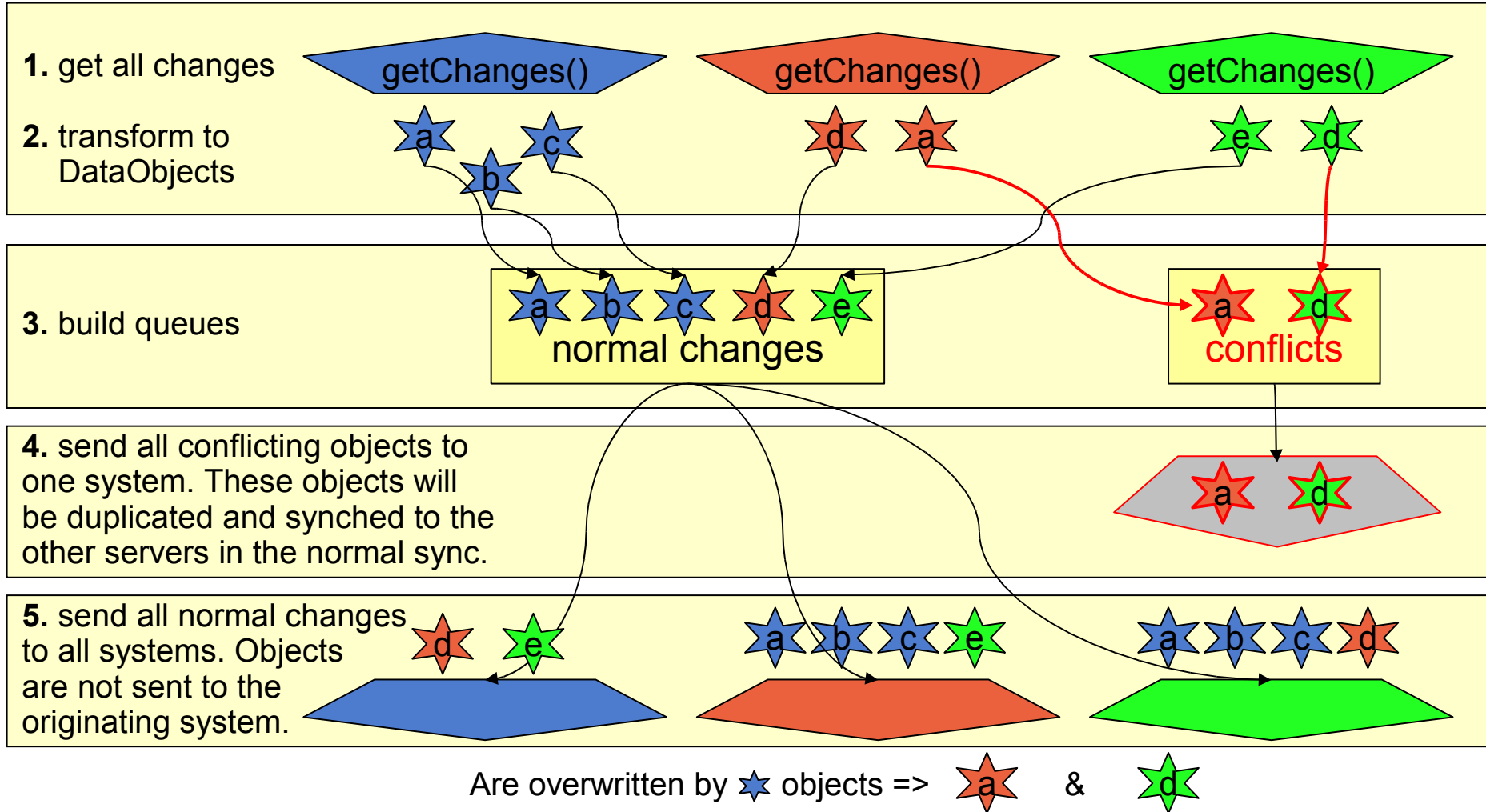


## User Mapping

- Simple auto-mapping using 'z-merge check'
  - same usernames on all systems.
- Advanced mapping in DB possible.
  - GROUP from A points to UserA and UserA on B
    - Relation is saved only in the Z-Merge UserMapping!



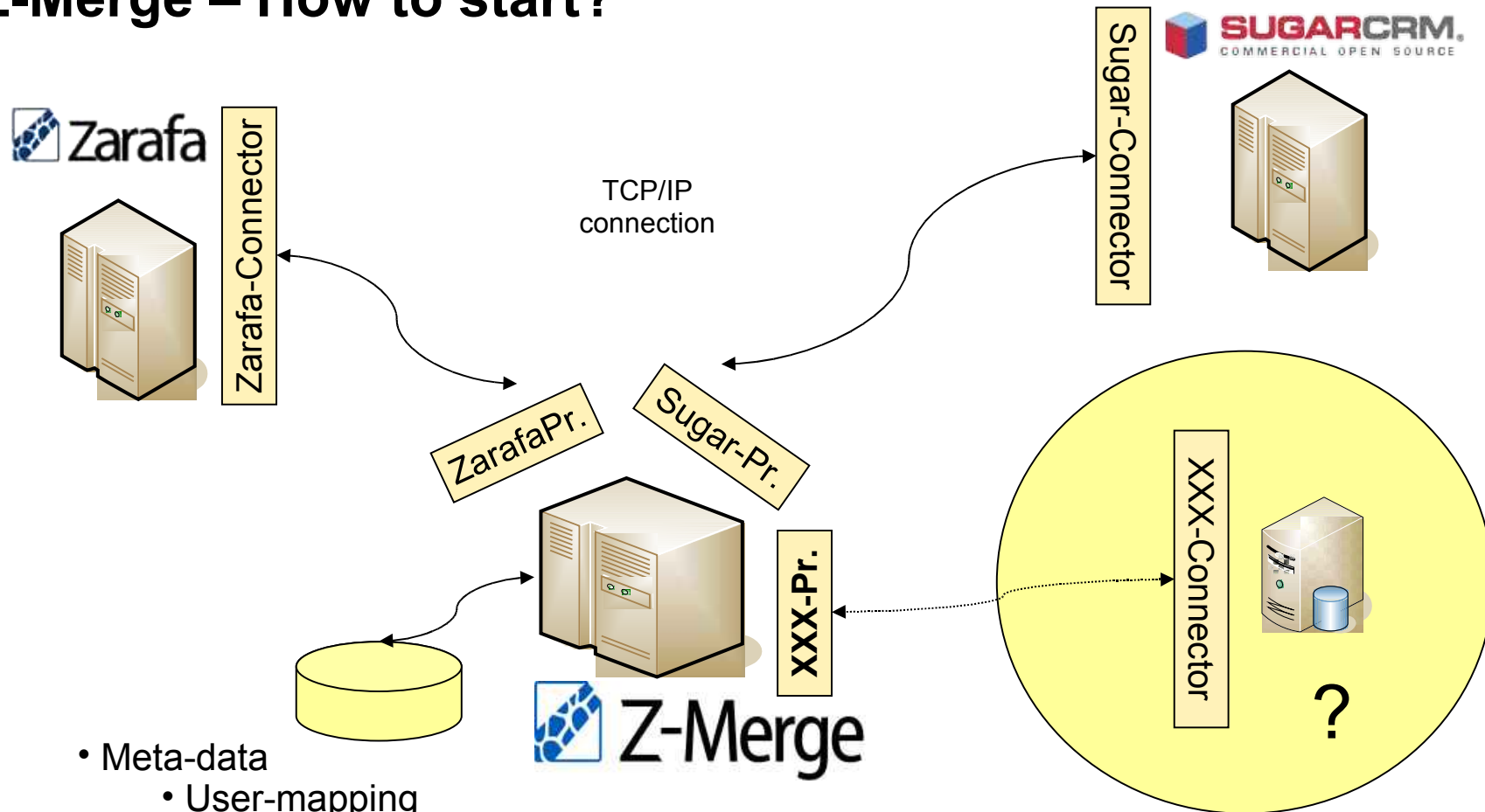
# Conflict resolution algorithm



## Offline-Synchronisation (incl. Conflict resolution)

1. Requests all latest changes (since the last known synchronous state) from all other servers
2. Transforms all modifications to comparable *DataObjects*
3. Compares objects and builds outgoing queues for
  - Normal changes (not conflicting)
  - Conflicting changes
4. Sends normal changes to all servers (unless the originating server)
5. Sends conflicting changes to one (configureable) server.
  - Objects get duplicated
6. Duplicated conflicts are later synced as „new“ to the other servers

## Z-Merge – How to start?



- Meta-data
  - User-mapping
  - Object Mapping between the systems

## Standard protocol

- **User name/id**

Is the value mapped internally in the UserMapping. Can be a username or an id, depending on what you want to use. **Requirement:** Has to be the same in the mapping and can not change (not implemented at the moment)

- **Object id**

Can be anything, will be inserted into the ObjectMapping automatically.

**Requirement:** has to be the same for reading and writing. The ObjectID may **never change** for the lifetime of the object.

- **Modification id (modid)**

Used to identify a change/modification. Has to be continuously increasing, an integer value. It must be possible to compare modids from the same connector to each other (Comparator can be overwritten)

## Exchanged data

Each change consists of

- Standard data (modid, user, object ID)
- Unspecified data containing the structured (?) data of the changed object

When writing data (setChanges() => write data) the connector executes the create/update/delete operation and returns the status containing

- The standard data
- Status information (ok, fail etc. and additional debug information)

## getChanges()

*Returns data since the given modification id. If `completeChanges` is set to true, doesn't wait for new changes. If set to true the method should return all changes, including the changes made by the connector.*

*Method should block if no changes are available. If there are many changes (for e.g. `getChanges(0)`) the script should only return a certain amount (to not exceed memory limitations).*

## **setChanges() (1)**

*Writes changes to the database. For objects with an object id an update or a delete is invoked. Objects without an object id should be created.*

*Method modifies objects and returns status information. For newly created objects it returns the newly created object id.*

## setChanges() (2)

*If the `isConflict` flag is set, that means the the transmitted objects should be  **duplicated** . In these cases, the transmitted object id is the id of the original object. The connector should “deepcopy” all data of the old object (inlc. relations etc) to a new one.*

*The object id of the new object is returned to the agent.*

## ping()

*Called by the agent to check if the connector is working.*

*The method should perform some internal tests (e.g. connect to the database) and then just return a boolean (true → ok, false → problems).*

## YourProfile extends ServerProfile

### *Methods to implement*

- *array(DataObjects)* **getDataObjects**( *yourSOAPresp()* )
- *yourSOAPresp()* **getServerObjects**( *array(DataObjects)* )
- boolean **processReturnObjects**  
( *SOAPResponse, array(DataObjects), isConflict\_flag* )
- boolean **compareModIDs**( *oldID, newID* )

## TestConnector example since Z-Merge beta2

Start developing Z-Merge using the *TestConnector*.

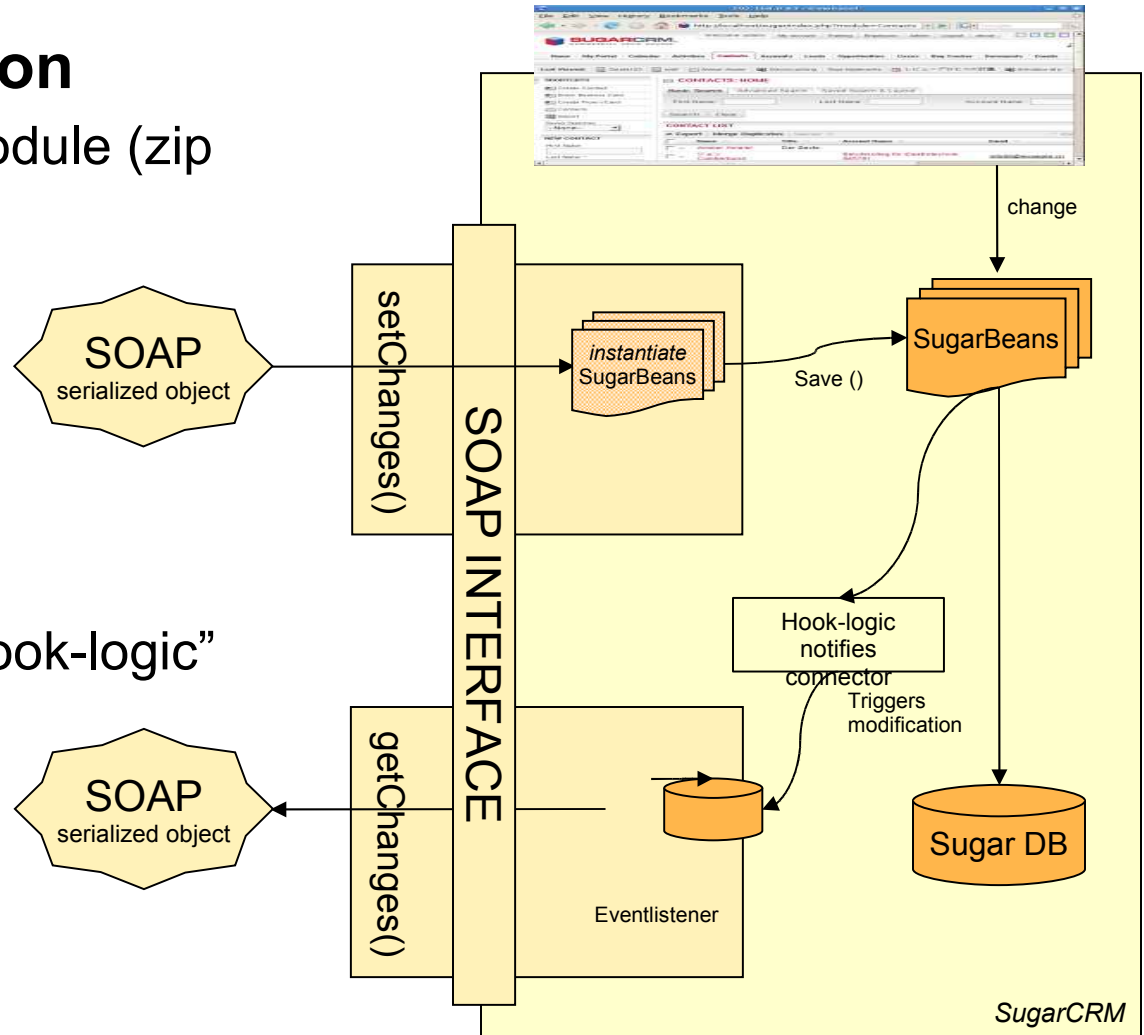
- Fully working connector
- Based on a simple database table for contacts
- Event catching using MySQL-Triggers (could/should be done using your API)
- Contains Z-Merge agent converting classes

The *TestConnector* is available as tarball.

## SugarCRM integration

## SugarCRM integration

- Loadable as Sugar-Module (zip file)
- Update safe!
- Changes are captured after max. 0,25s by event-driven push mechanism
- Implemented using “hook-logic”
- Writing instantiating SugarBeans



SugarCRM