### Sync with a Master, who does the merging; simple view.

Local  
DB

Merging Master, e.g Taskwarrior

**Send changes since last sync**

**Retrieve updated data and apply.**

Sync with Master, local changes while synchronizing.

Merging Master, e.g Taskwarrior

**Send changes since last sync**

Local  
DB

DB

DB

Previously synced

New local changes

User changes while sync in background

Updated server data

**Retrieve updated data**

DB

DB

DB

**Current sync point**

DB

DB

DB

DB

**New sync point**

DB

DB

DB

DB

DB

**Current sync point**

**Reapply changes**

### Merge a non-versioning remote slave with a conflict merging master.

Non-Ver­sioning Remote

Interme­diate DB

DB

DB

(1) Drop everything into intermediate  
DB, enabling change tracking

Previous sync point

Merging Master, e.g Taskwarrior

(2) Send versioned changes

(3) Retrieve updated data

DB

DB

DB

DB

(4) Drop again, possibly recording intermediate changes

DB

DB

DB

DB

DB

New sync point

(5) Save updated data, reapplying intermediate changes, if any.

Non-Ver­sioning Remote

(6) Save changes

DB

DB

Note that between steps (4) and (6) there is a tiny window where conflicting modifications to the remote will not properly be merged but unconditionally overwritten.

To prevent this, changes to the remote would have to be locked out between (4) and (6).

### Be the Merging Master

New virtual sync point

Remote client

Local  
DB

DB

other   
DB

Previously synced

New changes from other sources

Remote changes

**(1) Client sends his changes from previous sync point.**

**Sync point after merging remote.**

(3) Subtract the accepted “remote changes” from “new changes from other sources”. Create a virtual sync point.

(2) Apply the changes, possibly rejecting con­flict­ing changes in “other” on a per-field basis, by modifi­cation date.

(4) Upload these changes

### P2P Synchronizing

DB

DB

DB

Common sync point

Previously synced

Participant A

Participant B

Participant C

DB

DB

DB

DB

DB

DB

Invariant: Each unique sync point (CommitId) stands for a logical timestamp representing the same state of the synchronized data on all participants. This does not mean that this state has to be reconstructable.

Independent changes,

Possibly conflicting

DB

DB

Common sync point

Sync between A and B,

A takes server role.

DB

DB

DB

Case (a):  
Sync between B and C, C takes server role.

DB

DB

DB

Common sync point

„TWO-SYNC“

After the sync, we are basically in the same situation as “TWO-SYNC”.

Case (b):  
Sync between B and C, B takes server role.

Common sync point

Should work, though on a first look it seems that it would have been simpler if C, who is closer to the common sync point, would have taken the server role.

DB