How to migrate to sharding with Spider

Kentoku SHIBA

Agenda

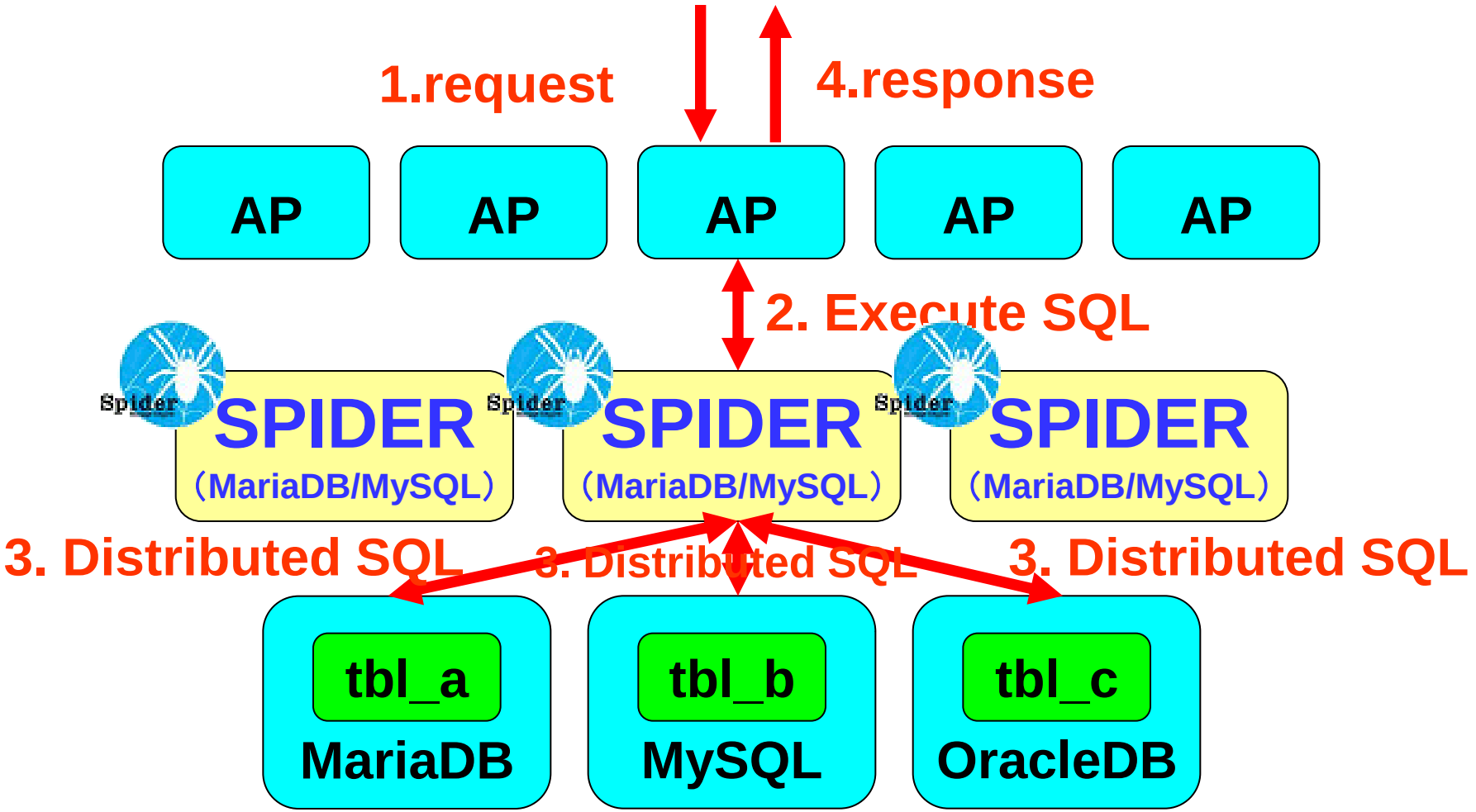
1. What is SPIDER?
2. Why SPIDER? what SPIDER can do for you?
3. How to migrate to sharding using Replication
4. How to migrate to sharding using Trigger
5. How to migrate to sharding using Spider function
6. How to migrate to sharding
   using Vertical Partitioning Storage Engine

What is Spider

What is the Spider Storage Engine?

Spider is a sharding solution and proxying solution. **Spider Storage Engine is a plugin of MariaDB/MySQL.** Spider tables can be used to federate from other servers MariaDB/MySQL/OracleDB tables as if they stand on local server. And Spider can create database sharding by using table partitioning feature.

What is the Spider Storage Engine?



All databases can be used as ONE database through Spider.

What is the Spider Storage Engine?

Spider is bundled in MariaDB

from 10.0 and all patches for MariaDB is applied in 10.3

Why SPIDER?
What SPIDER can do for you?

Why Spider? What Spider can do for you?

For federation

You can attach tables from other servers or from local server by using Spider.

For sharding

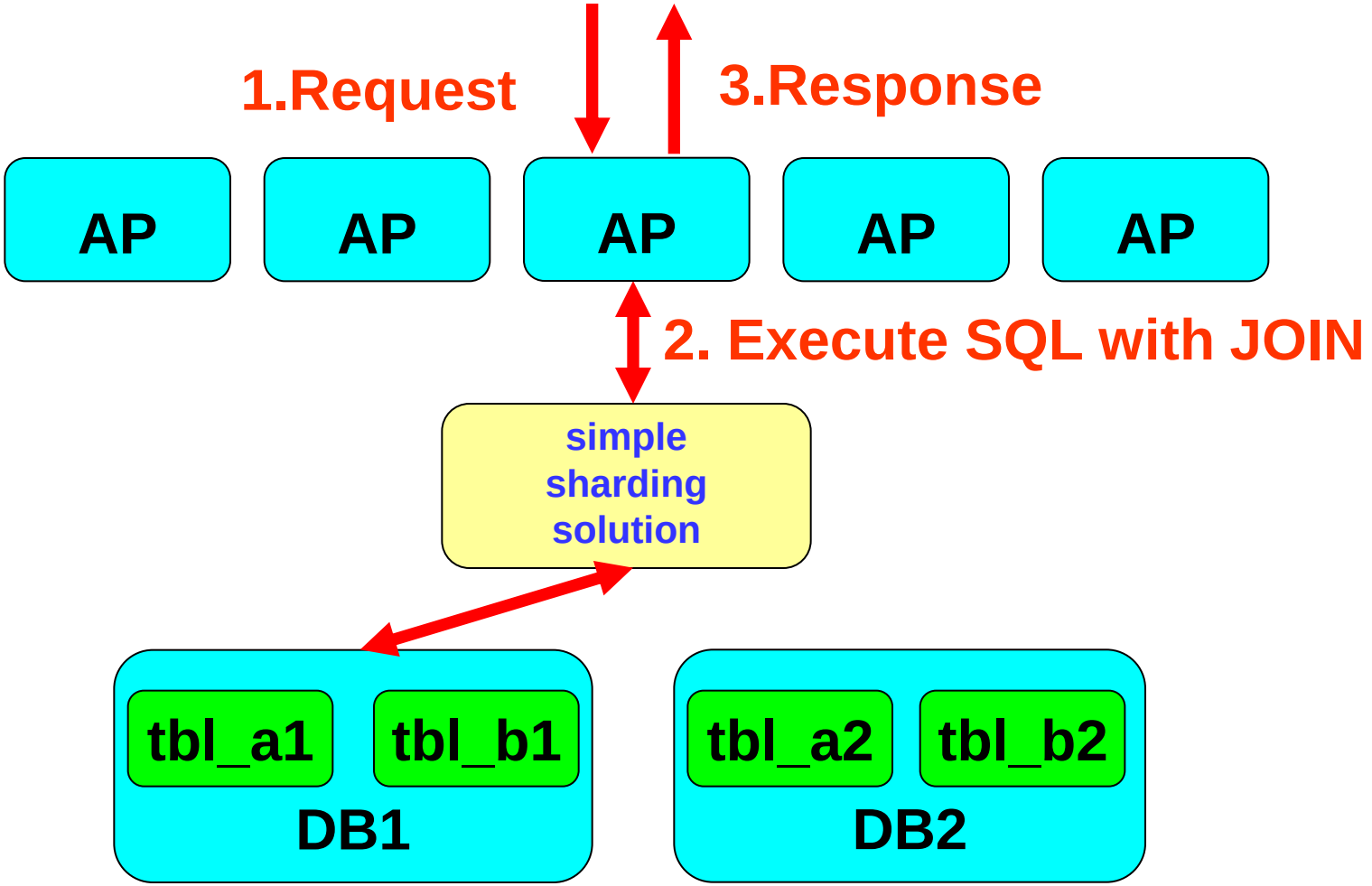
You can divide huge tables and huge traffics to multiple servers by using Spider.

Why Spider? What Spider can do for you?

Cross shard join

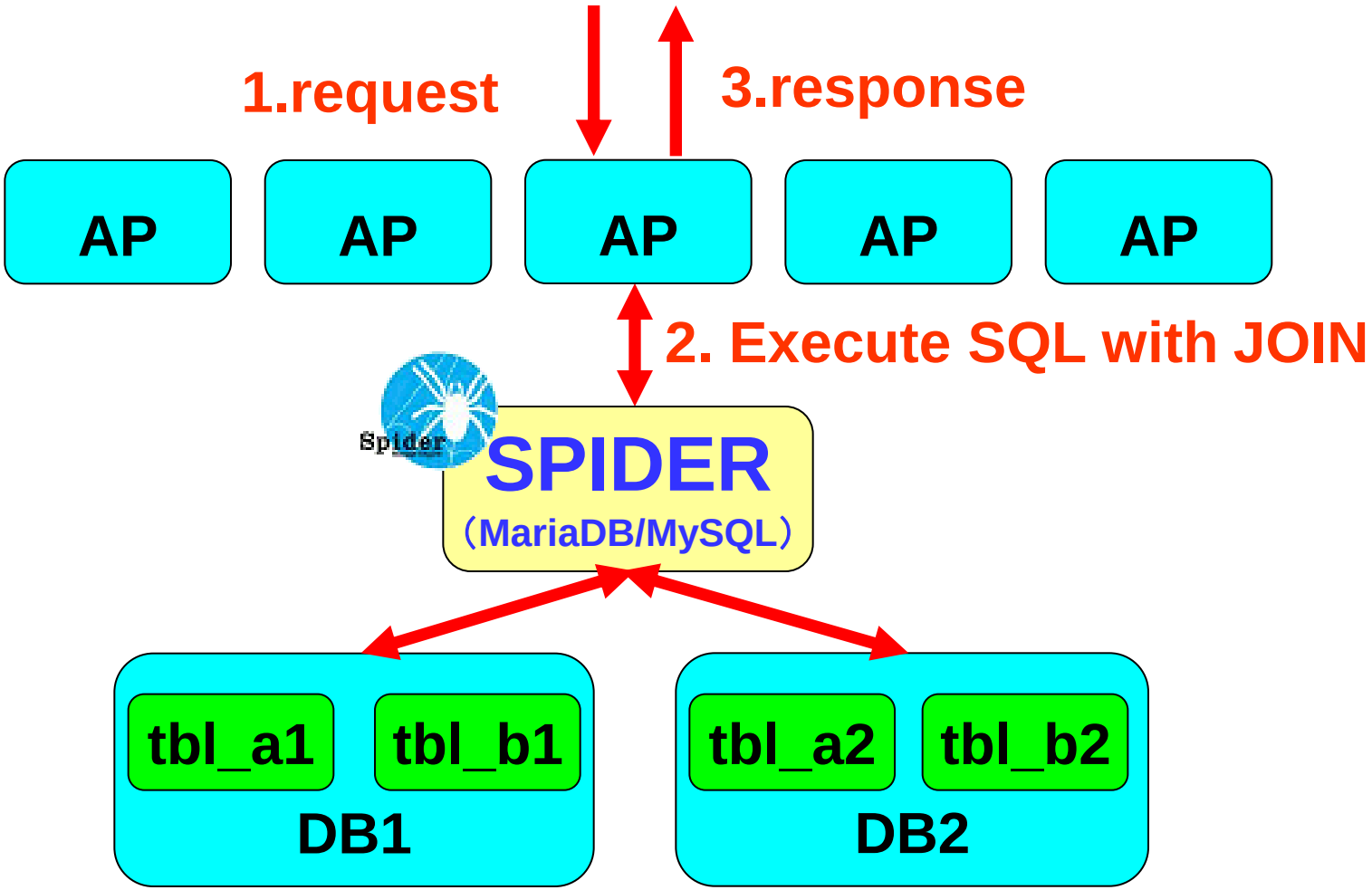
You can join all tables by using Spider,
even if tables are on different servers.

Join operation with simple sharding solution **(without Spider)**



Join operation requires that all joined tables are on same server.

Join operation with Spider



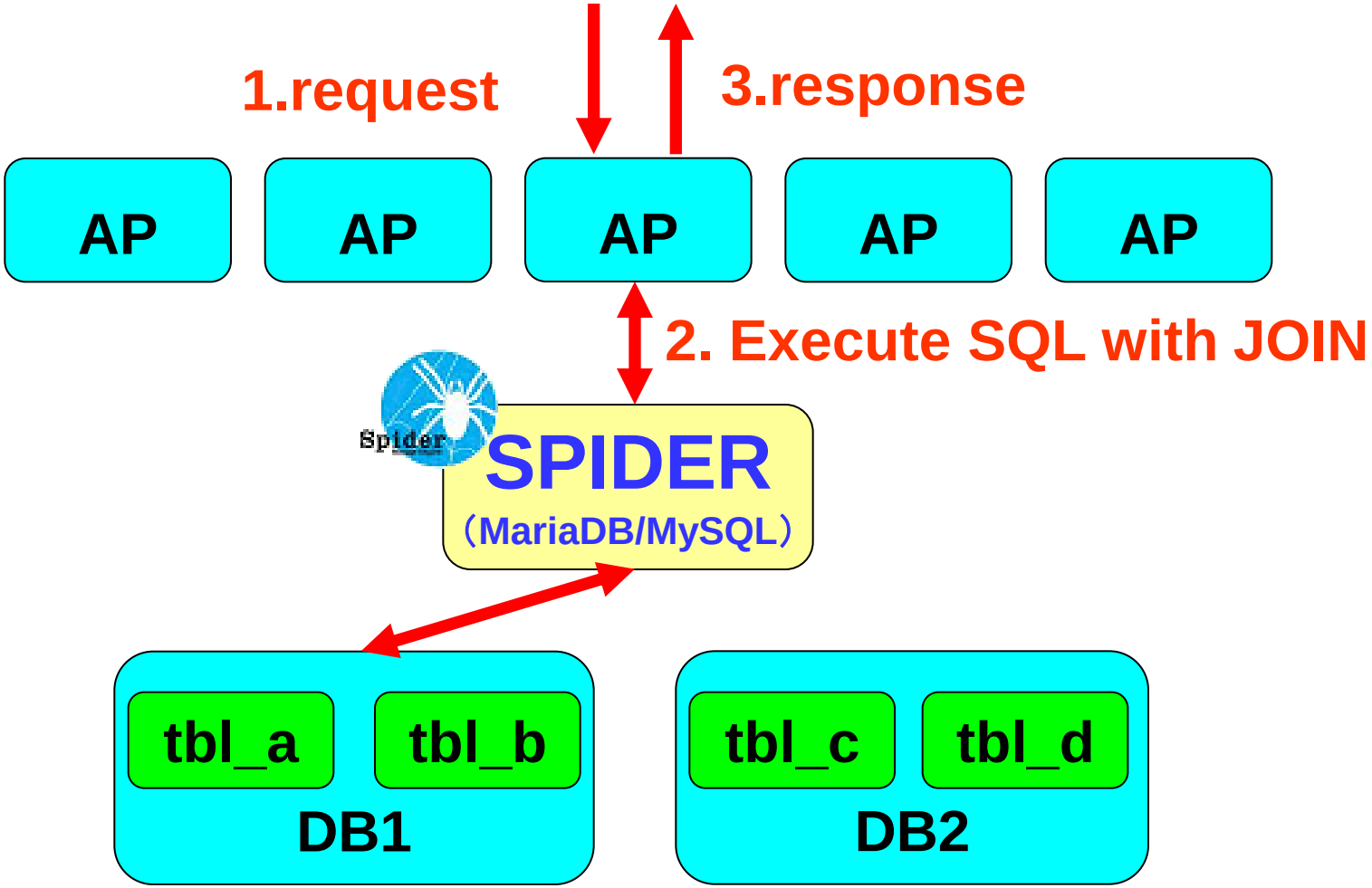
You can JOIN all tables, even if tables are on different servers.

Why Spider? What Spider can do for you?

Join push down

If it is possible, Spider executes JOIN operation at data node directly.

JOIN push down



If all tables are on same data node, Spider executes JOIN operation on data node directly.

JOIN push down

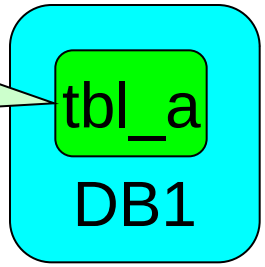
Simple join operation are two times faster on simple JOIN pushdown test.

Also, in this pushdown of JOIN, when aggregate functions are included in the query, since the aggregation processing is also executed at the data node, the amount of data transfer is greatly reduced and it becomes super high speed.

How to migrate to sharding with Spider using Replication

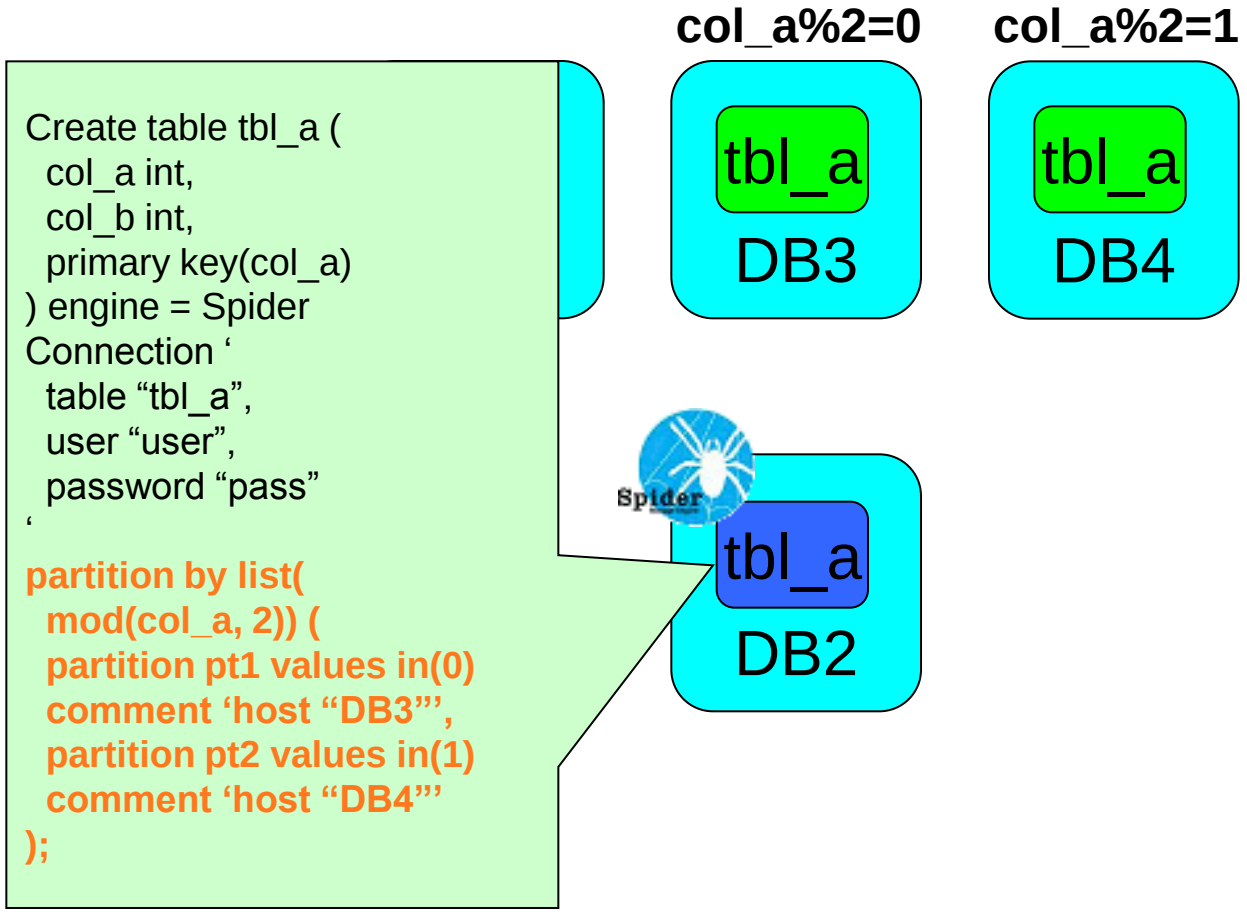
Initial Structure

```
Create table tbl_a (
  col_a int,
  col_b int,
  primary key(col_a)
) engine = InnoDB;
```



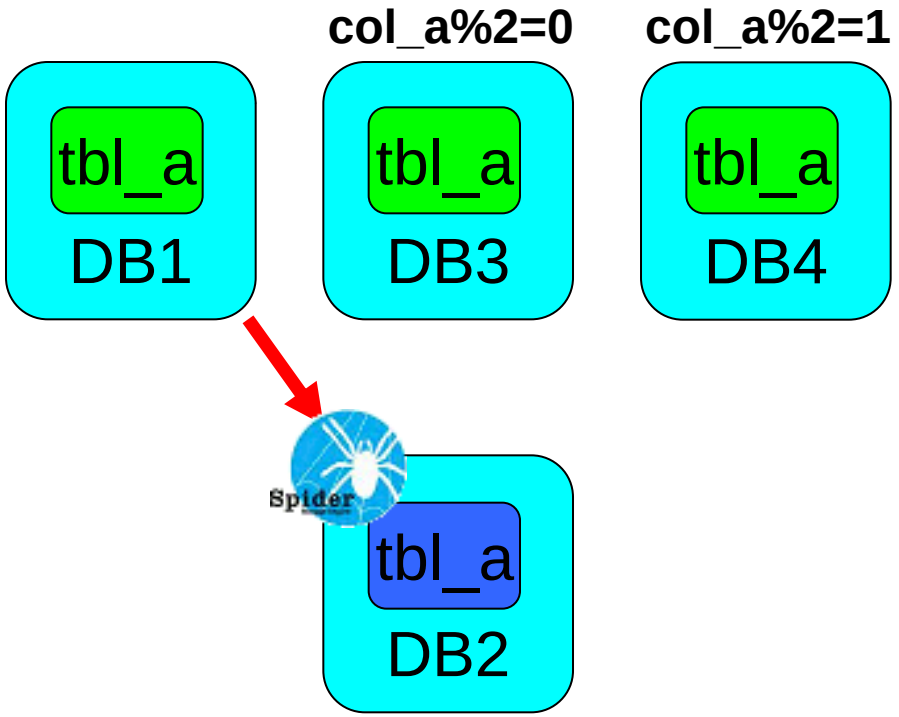
There is 1 MariaDB server without Spider.

Step 1 (for migrating)



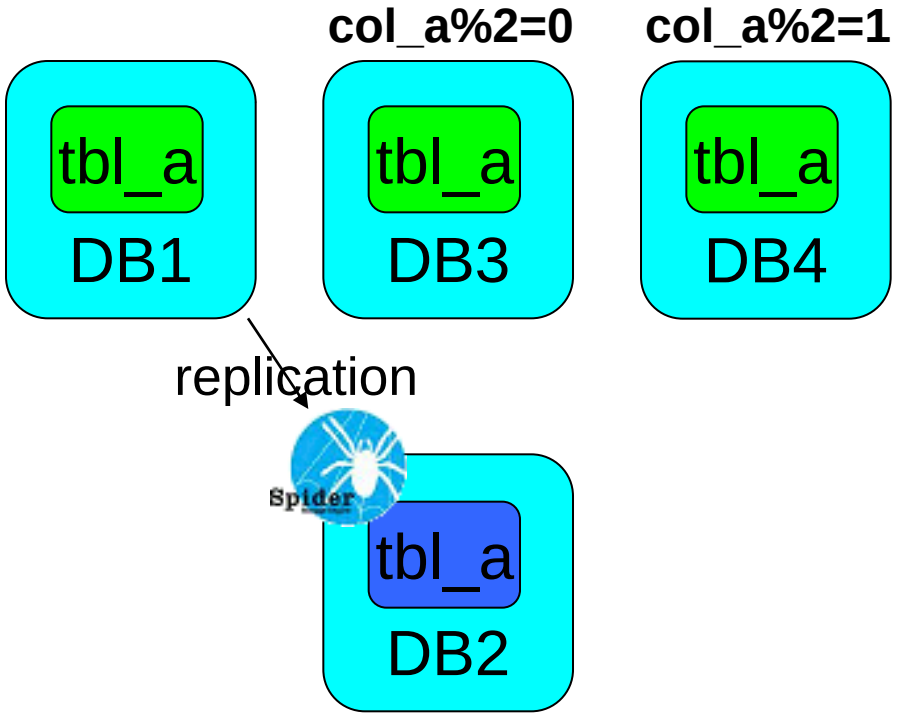
Create table on DB3 and DB4.
Then create Spider table on DB2.

Step 2



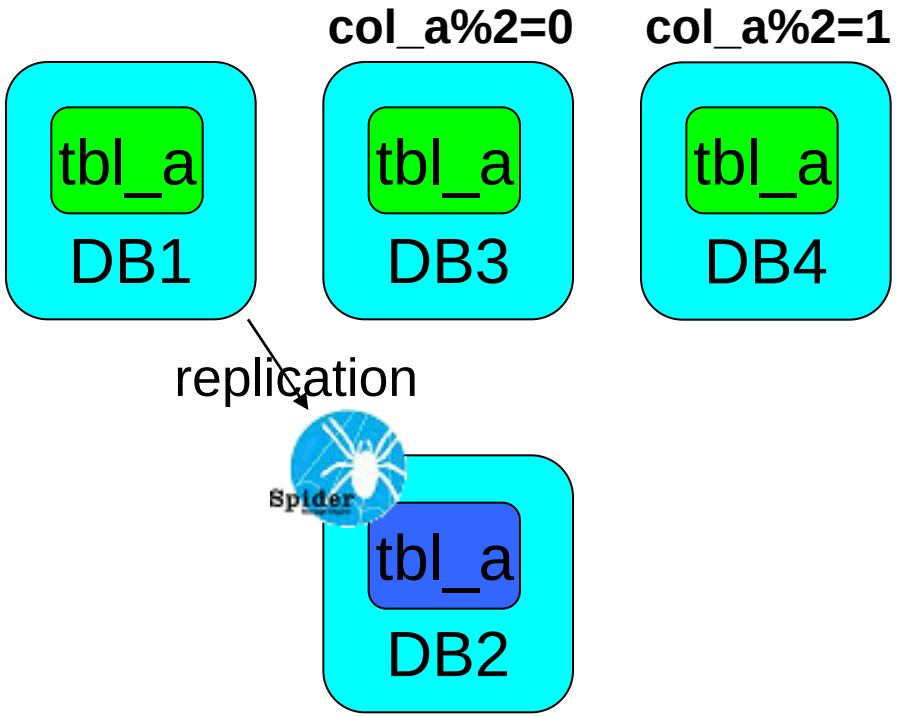
Copy table data from DB1 to DB2.
(Use mysqldump with “--master-data = 1 or 2” option)

Step 3



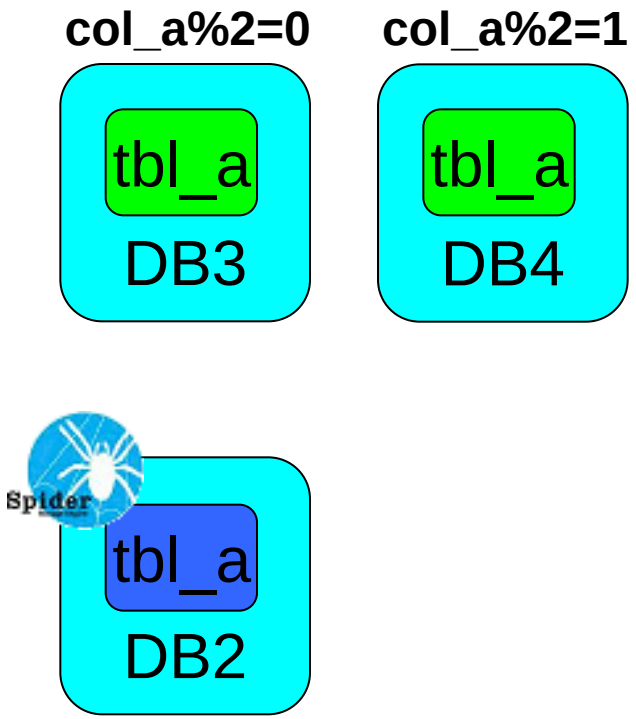
Start replication from DB1 to DB2.
Wait for resolving replication delay.

Step 4



Stop client access for DB1.
Wait for resolving replication delay.
Switch client access from DB1 to DB2.

Finish



Stop replication on DB2.
Remove DB1.

Pros and Cons of Replication way

Pros

1. No need to manage lock size for coping.
2. Support non primary key table.

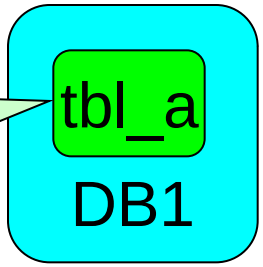
Cons

1. Need to stop writing.

How to migrate to sharding with Spider using Trigger

Initial Structure

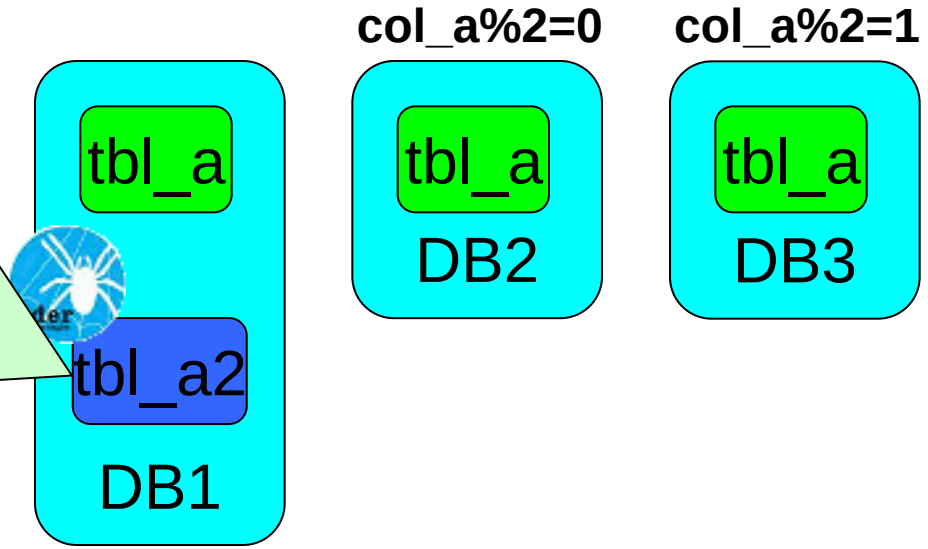
```
Create table tbl_a (
  col_a int,
  col_b int,
  primary key(col_a)
) engine = InnoDB;
```



There is 1 MariaDB server without Spider.

Step 1 (for migrating)

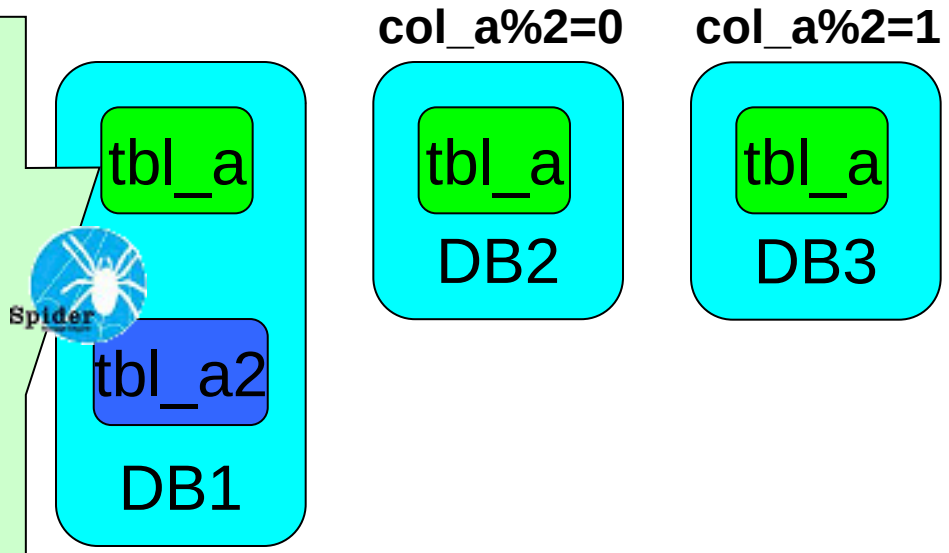
```
Create table tbl_a2 (
  col_a int,
  col_b int,
  primary key(col_a)
) engine = Spider
Connection ‘
  table “tbl_a”,
  user “user”,
  password “pass”
‘
partition by list(
  mod(col_a, 2)) (
  partition pt1 values in(0)
  comment ‘host “DB2”’,
  partition pt2 values in(1)
  comment ‘host “DB3”’
);
```



Create table on DB2 and DB3.
Then create Spider table on DB1.

Step 2

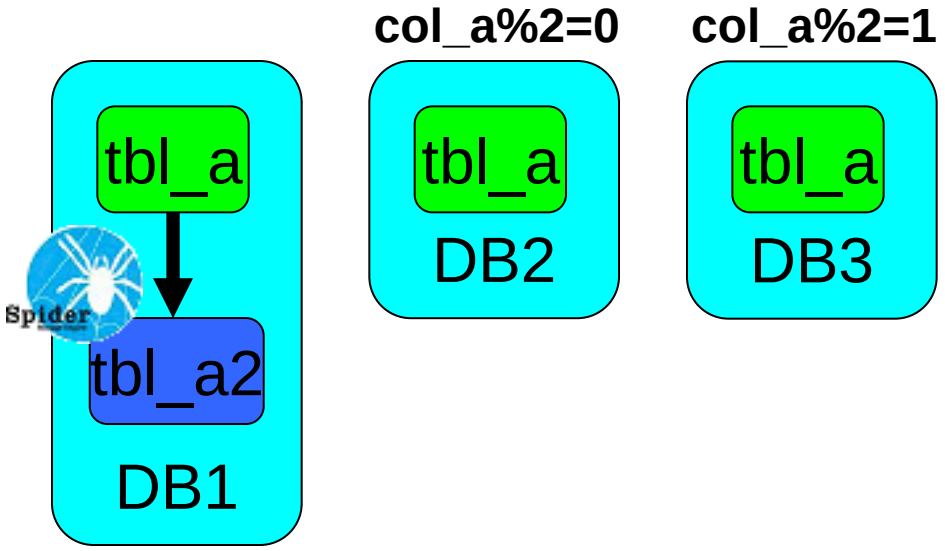
```
delimiter |
create trigger tbl_a_i after insert
  on tbl_a for each row
  insert into tbl_a2 (a,b) values
    (new.a, new.b);
|
create trigger tbl_a_u after update
  on tbl_a for each row
  update tbl_a2 set a = new.a,
    b = new.b
    where a = old.a;
|
create trigger tbl_a_d after delete
  on tbl_a for each row
  delete from tbl_a2 where a = old.a;
|
delimiter ;
```



Create triggers on DB1.

(For copying insert, update and delete. If you use “truncate” for tbl_a, you should better to use other way)

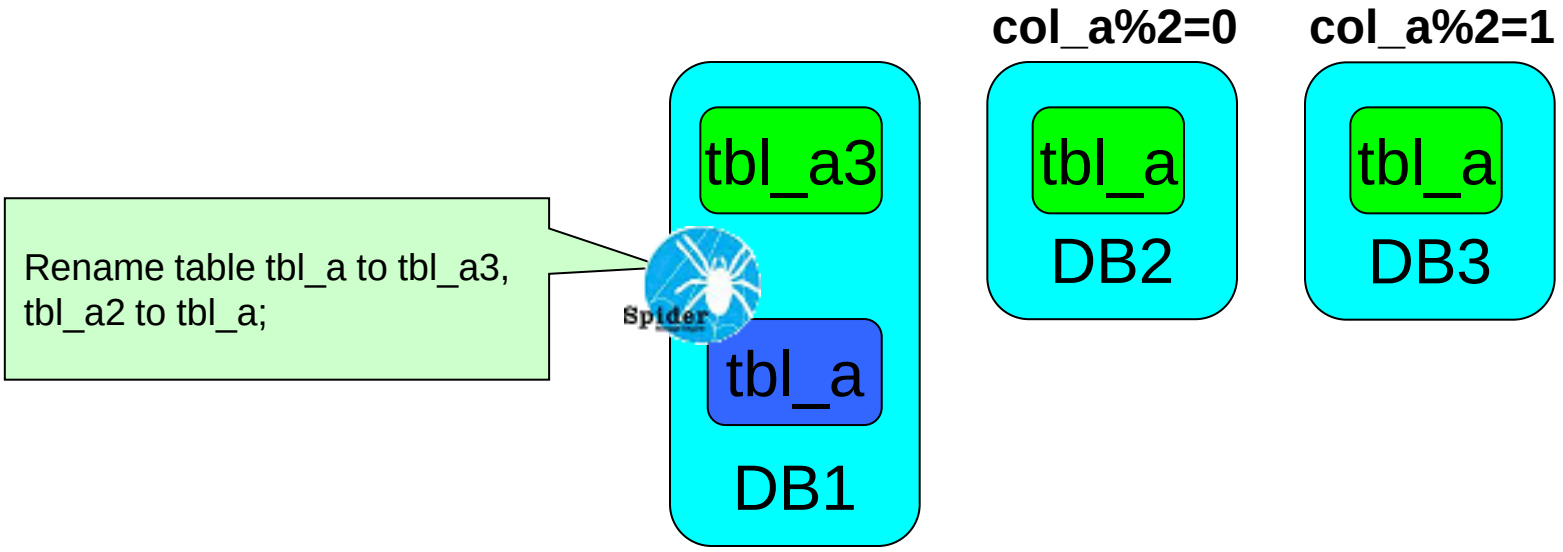
Step 3



Insert select from tbl_a to tbl_a2.

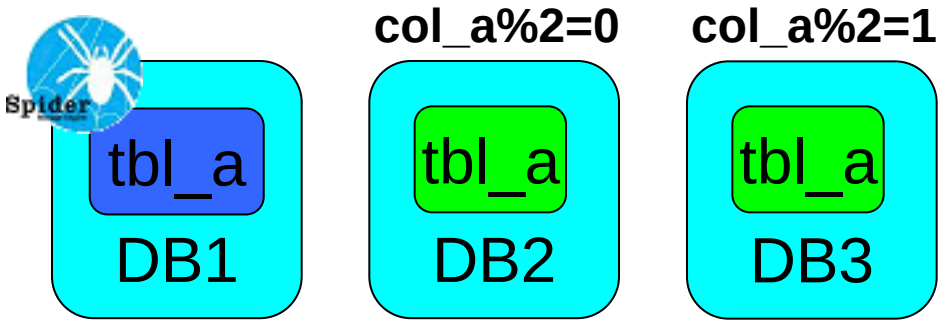
(Please take care of locking time for tbl_a and tbl_a2.)

Step 4



Rename table from tbl_a2 to tbl_a.

Finish



Drop table tbl_a3.

Pros and Cons of Trigger way

Pros

1. No need to stop services.
2. Easy to copy.(Simple command)

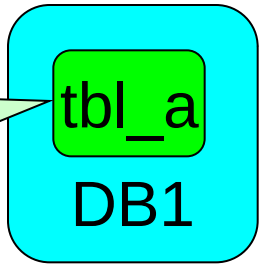
Cons

1. Impossible to support truncate.
2. Need to manage lock size at coping.
3. Impossible to support non primary key.

How to migrate to sharding with Spider using Spider function

Initial Structure

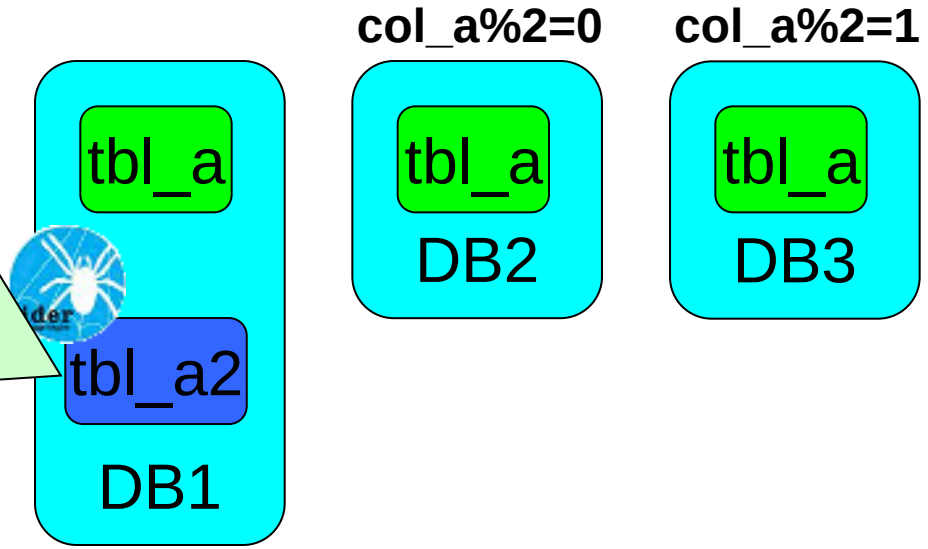
```
Create table tbl_a (
  col_a int,
  col_b int,
  primary key(col_a)
) engine = InnoDB;
```



There is 1 MariaDB server without Spider.

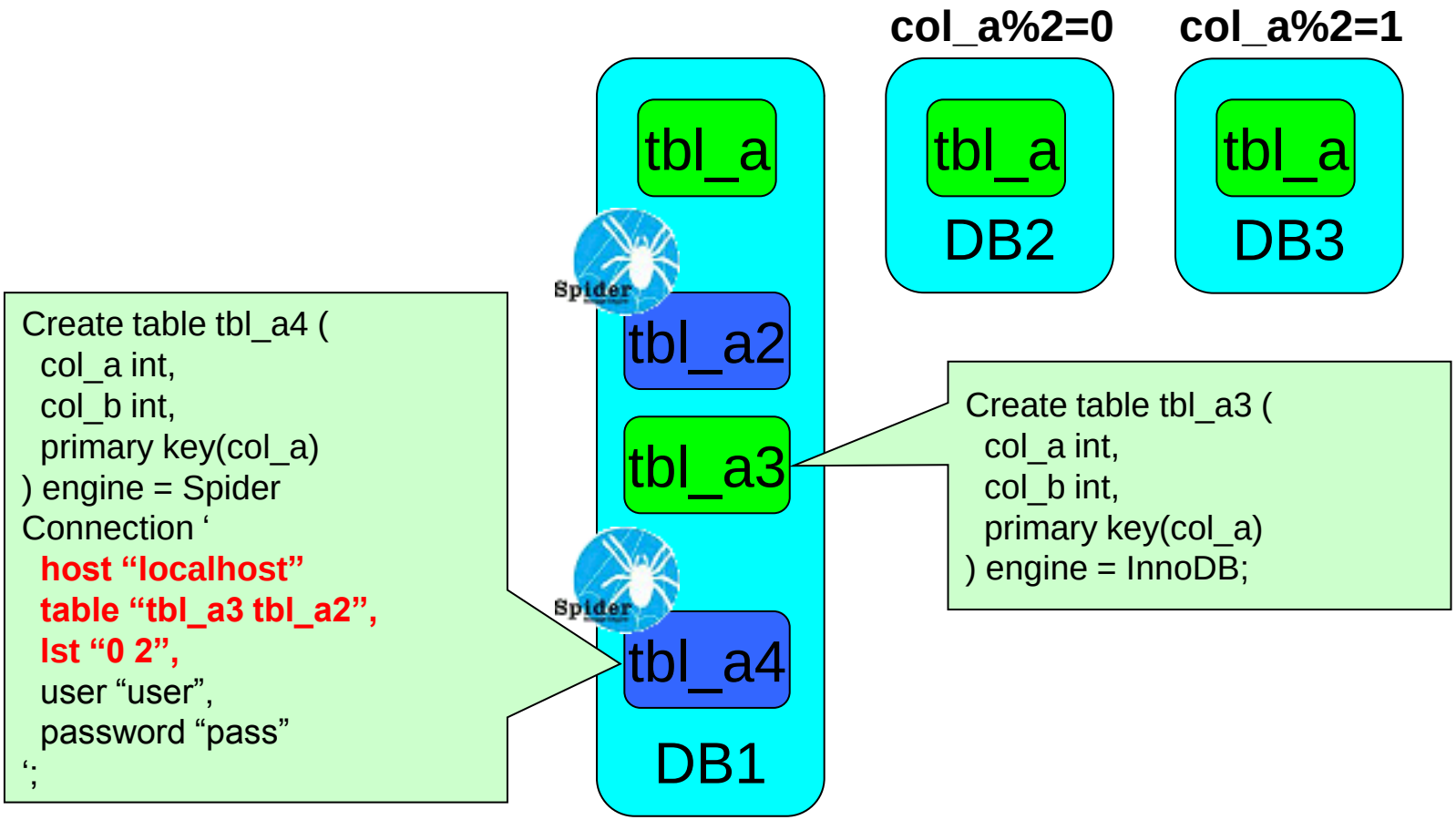
Step 1 (for migrating)

```
Create table tbl_a2 (
  col_a int,
  col_b int,
  primary key(col_a)
) engine = Spider
Connection '
  table "tbl_a",
  user "user",
  password "pass"
'
partition by list(
  mod(col_a, 2)) (
  partition pt1 values in(0)
  comment 'host "DB2"',
  partition pt2 values in(1)
  comment 'host "DB3"'
);
```



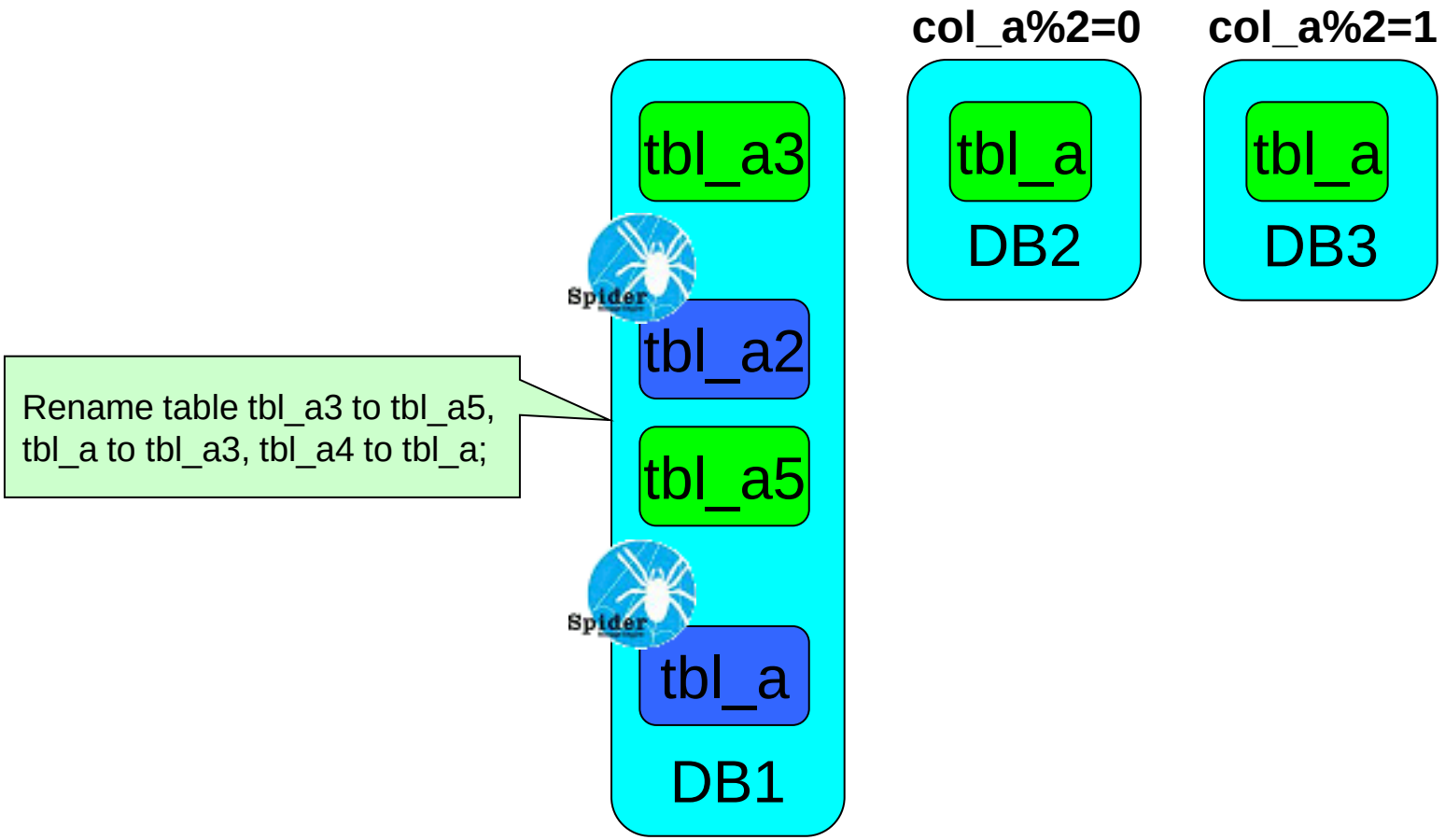
Create table on DB2 and DB3.
Then create Spider table on DB1.

Step 2



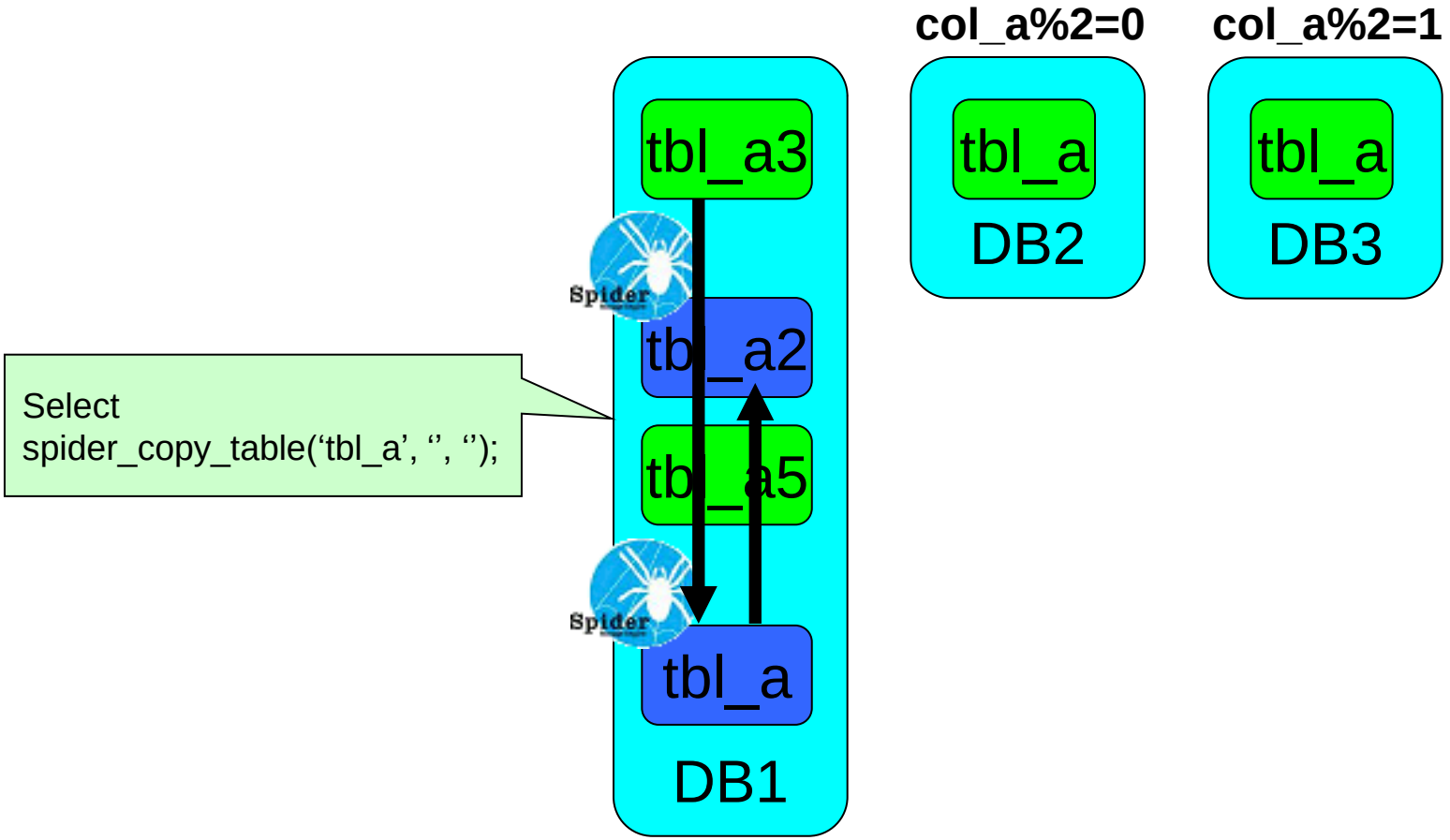
Create tables on DB1.

Step 3



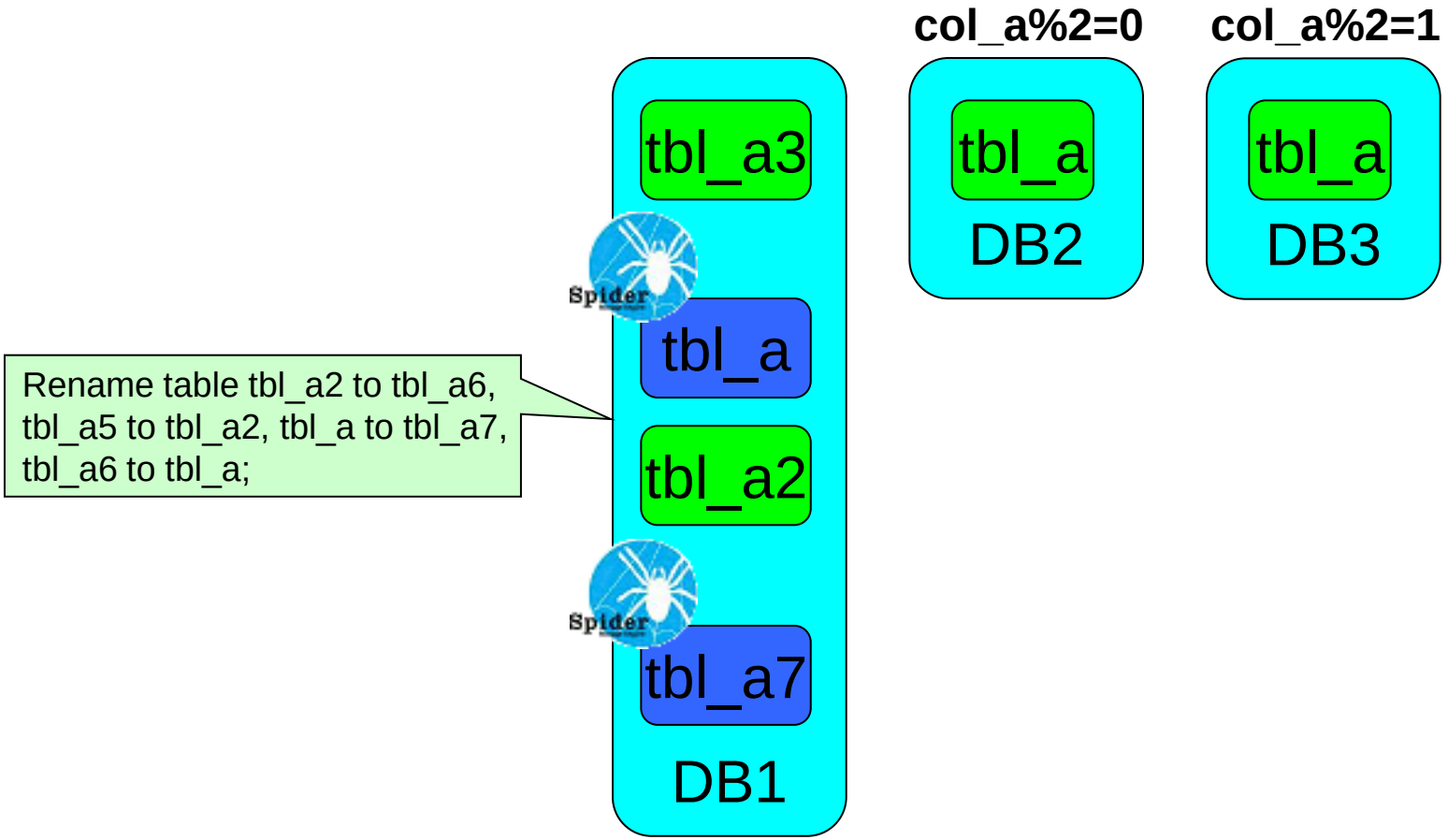
Rename table on DB1.

Step 4



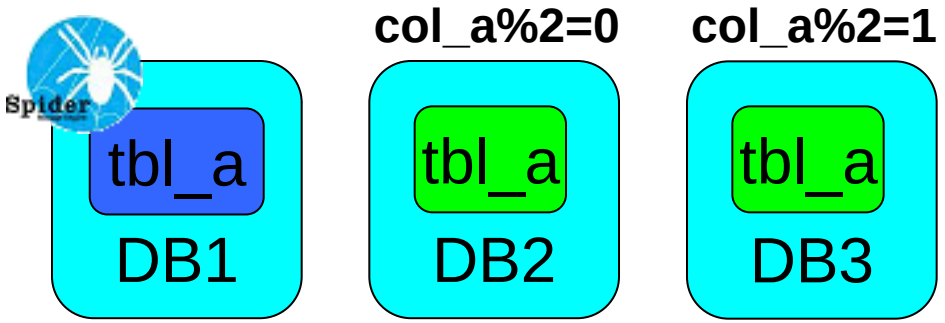
Copy data on DB1.

Step 5



Rename table on DB1.

Finish



Drop table tbl_a2, tbl_a3 and tbl_a7.

Pros and Cons of Spider function way

Pros

1. No need to stop services.
2. Easy to copy.(Simple command. Lock size is managed by Spider)

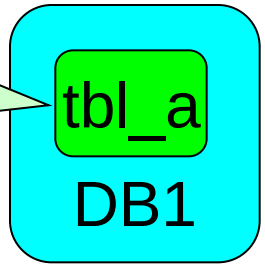
Cons

1. Impossible to support non primary key.

How to migrate to sharding with Spider using Vertical Partitioning Storage Engine

Initial Structure

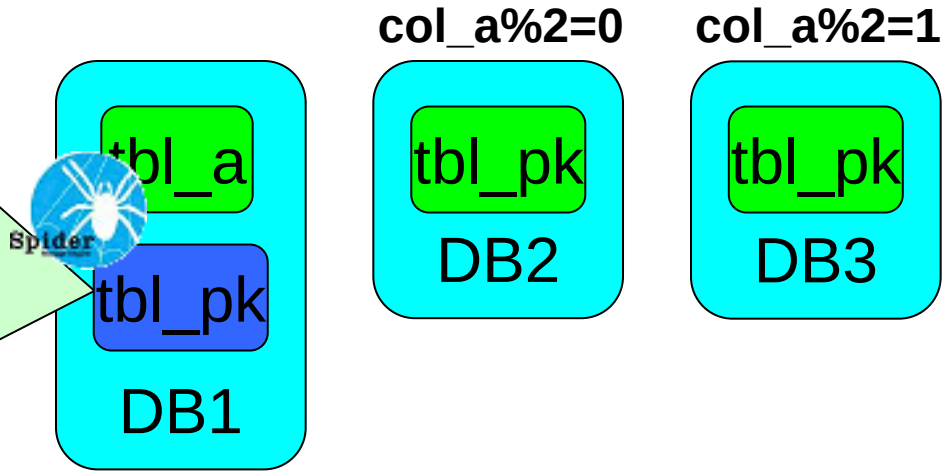
```
Create table tbl_a (
  col_a int,
  col_b int,
  primary key(col_a),
  key idx2(col_b)
) engine = InnoDB;
```



There is 1 MariaDB server without Spider.

Step 1 (for migrating)

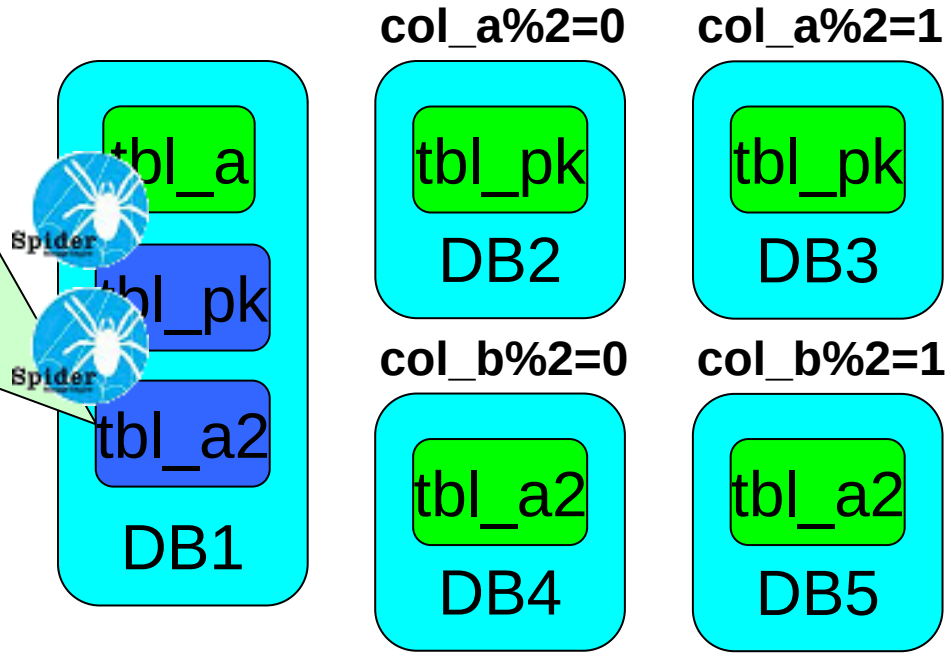
```
Create table tbl_pk (
  col_a int,
  primary key(col_a)
) engine = Spider
Connection '
  table "tbl_pk",
  user "user",
  password "pass"
'
partition by list(
  mod(col_a, 2)) (
  partition pt1 values in(0)
  comment 'host "DB2"',
  partition pt2 values in(1)
  comment 'host "DB3"'
);
```



Create table on DB2 and DB3.
Then create tables on DB1.

Step 2

```
Create table tbl_a3 (
  col_a int,
  col_b int,
  key idx1(col_a),
  key idx2(col_b)
) engine = Spider
Connection '
  table "tbl_a2",
  user "user",
  password "pass"
'
partition by list(
  mod(col_b, 2)) (
  partition pt1 values in(0)
  comment 'host "DB4"',
  partition pt2 values in(1)
  comment 'host "DB5"'
);
```

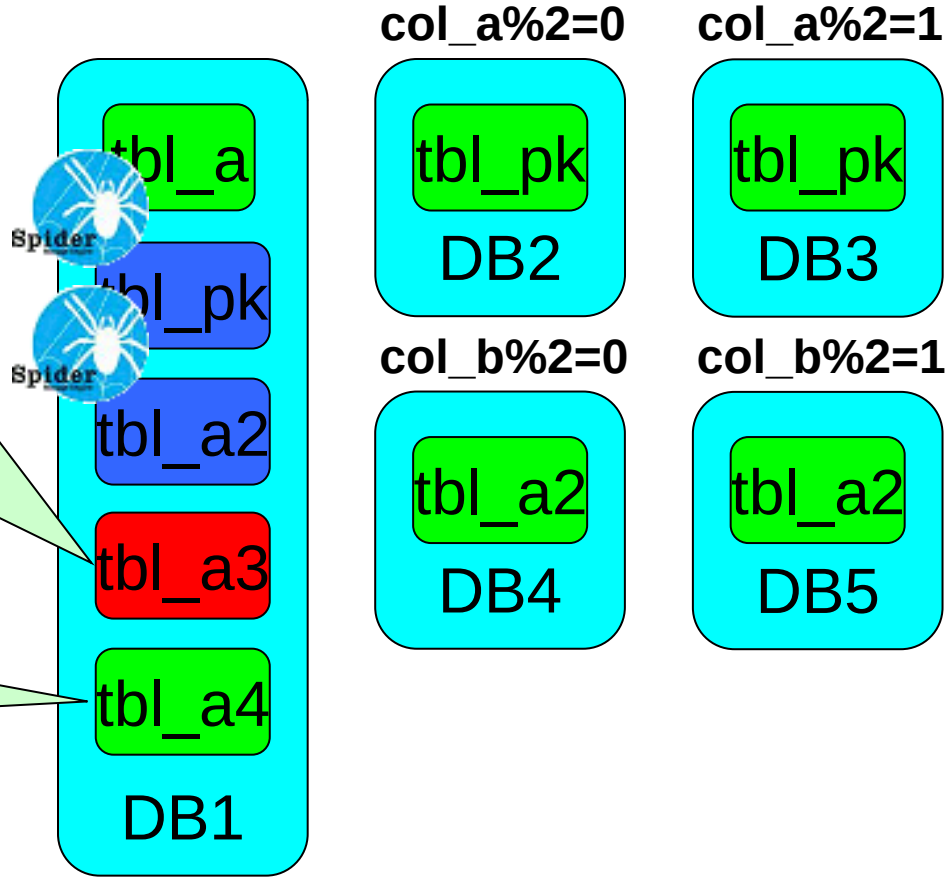


Create table on DB4 and DB5.
Then create tables on DB1.

Step 3

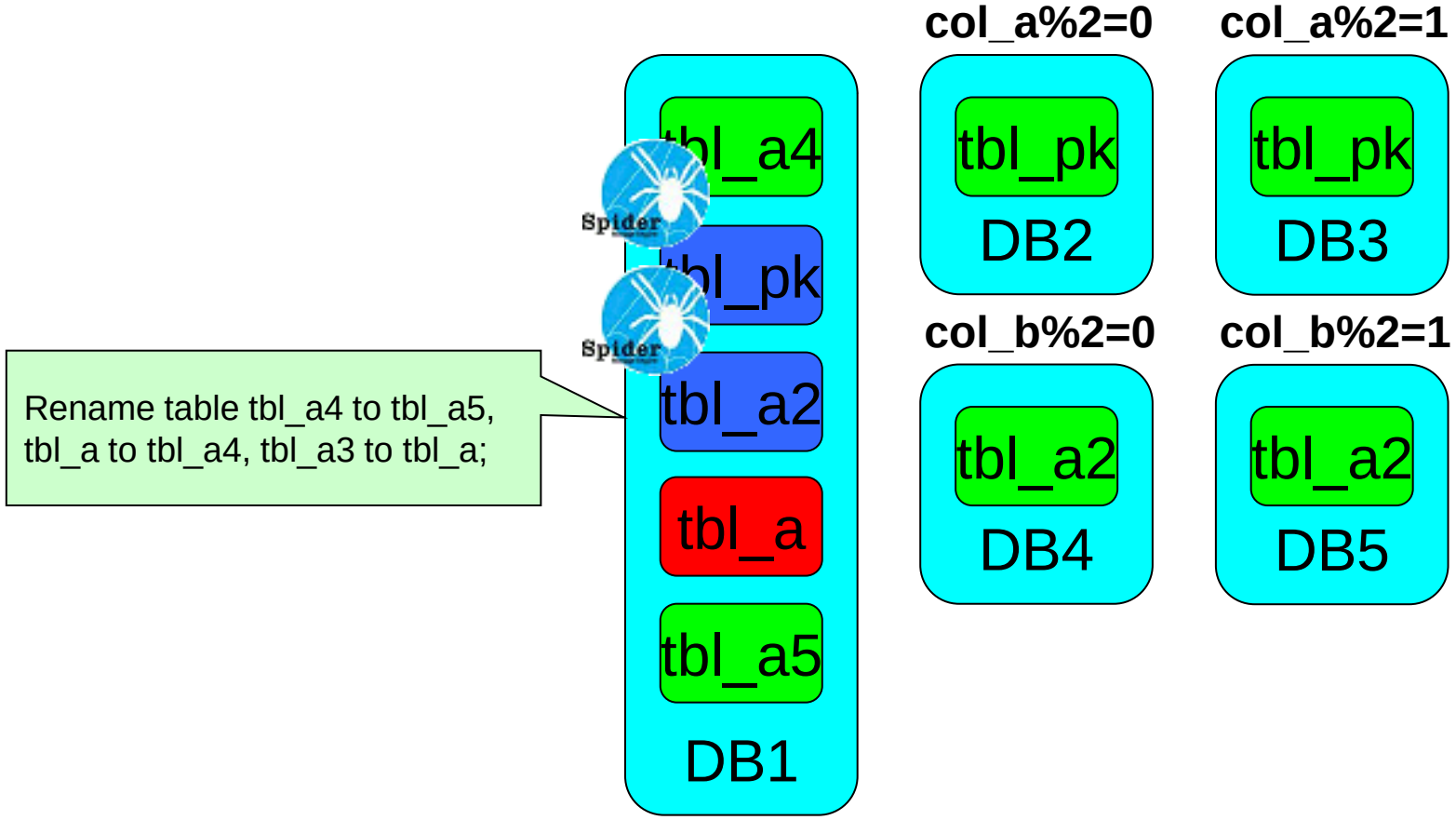
```
Create table tbl_a3 (
  col_a int,
  col_b int,
  primary key(col_a),
  key idx2(col_b)
) engine = VP
Comment '
  ctm "1",
  ist "1",
  pcm "1",
  tnl "tbl_a4 tbl_pk tbl_a2"
';
```

```
Create table tbl_a4 (
  col_a int,
  col_b int,
  primary key(col_a)
) engine = InnoDB;
```



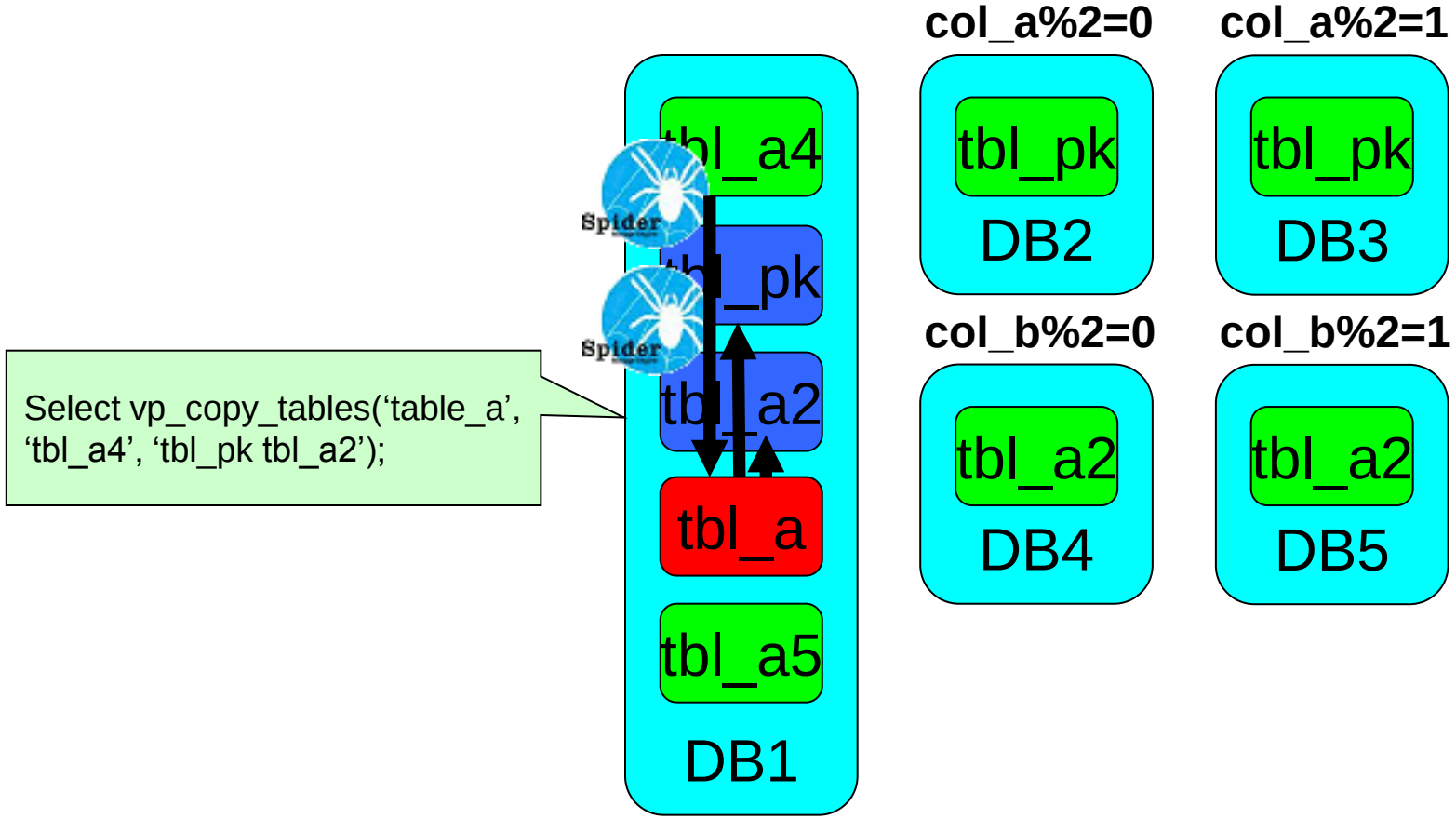
Create tables on DB1.

Step 4



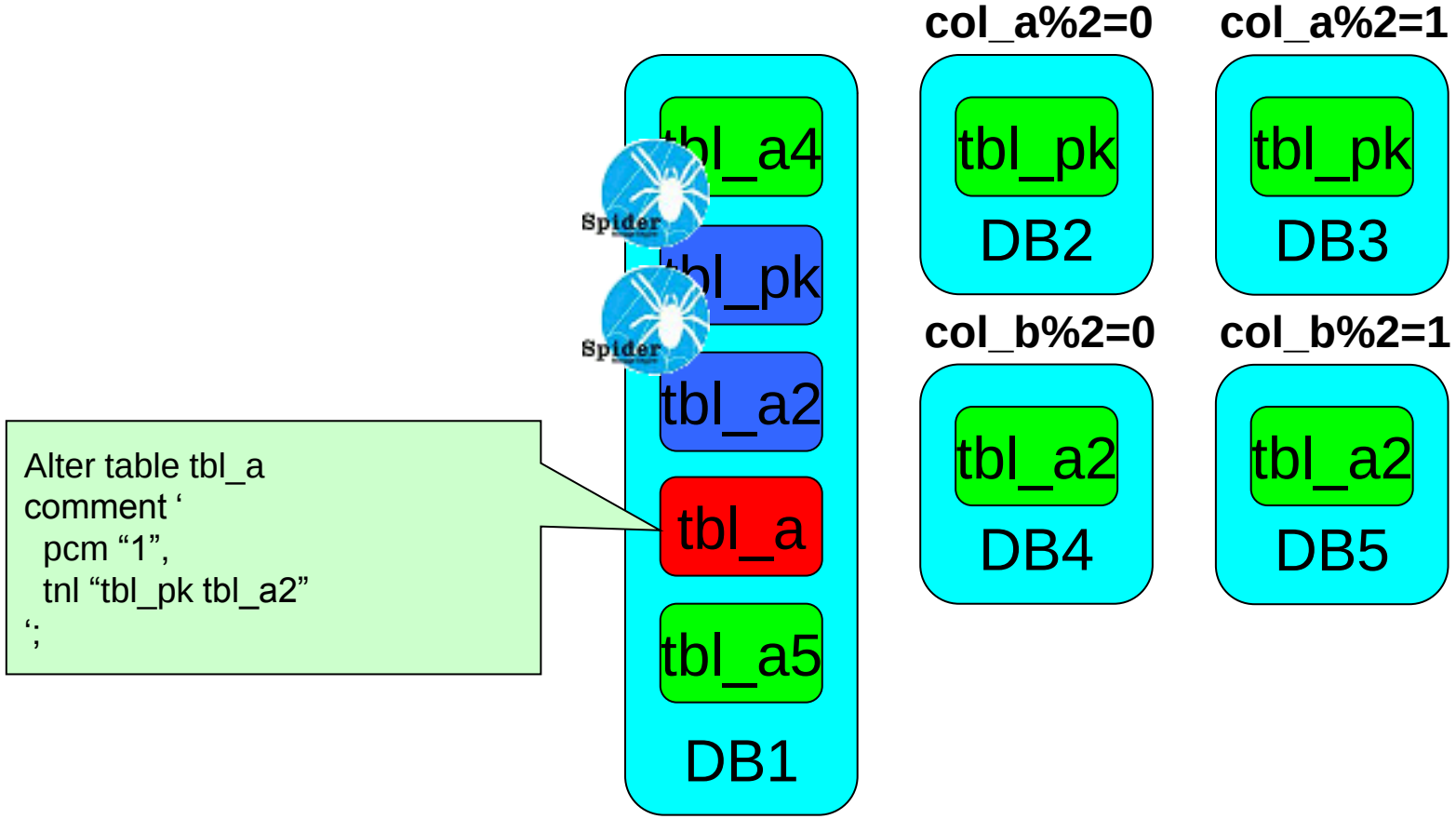
Rename tables on DB1.

Step 5



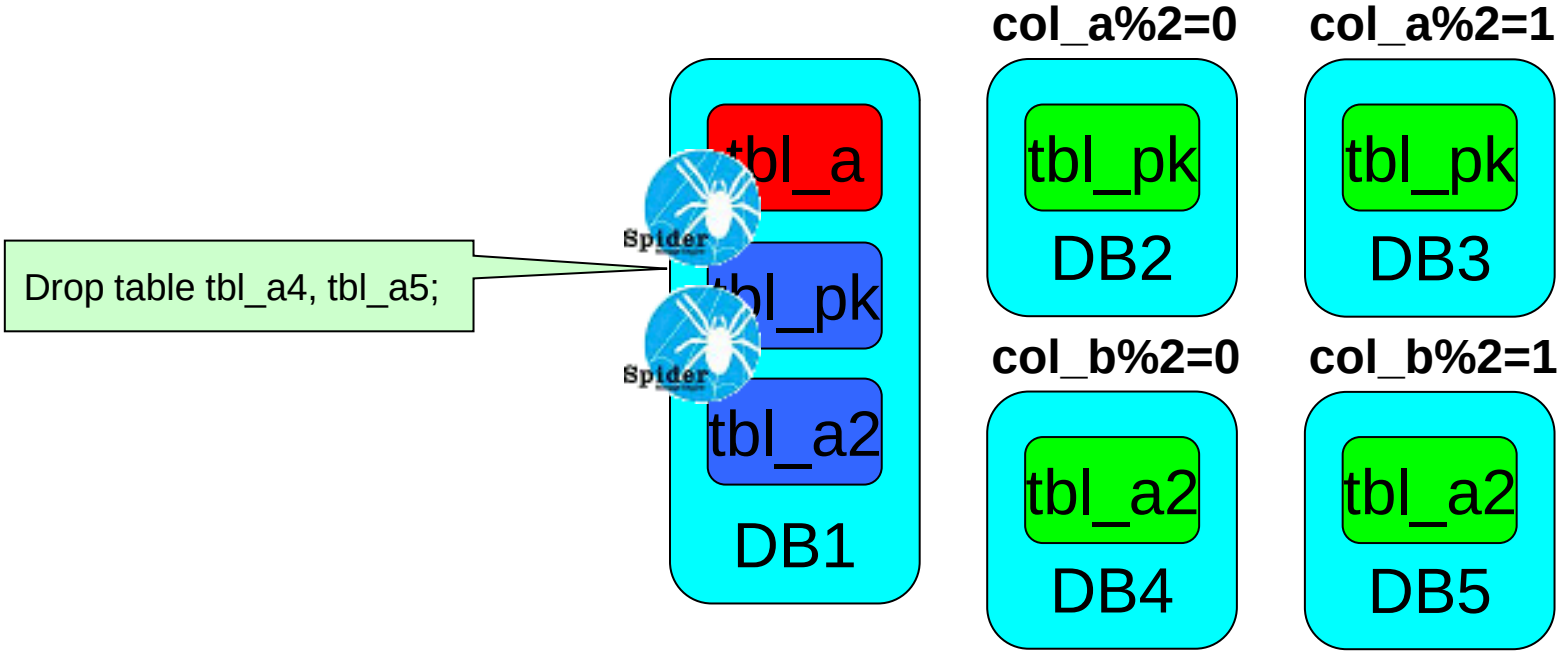
Copy data from tbl_a4 to tbl_pk and tbl_a2 on DB1.

Step 6



Alter table tbl_a on DB1.

Finish



Drop table tbl_a on DB1.

Pros and Cons of VP way

Pros

1. No need to stop services.
2. Support spiltting by non unique columns.
3. Easy to copy.(Simple command. Lock size is managed by VP)

Cons

1. VP storage engine is required.
2. Impossible to support non primary key.

Thank you for taking your time!!