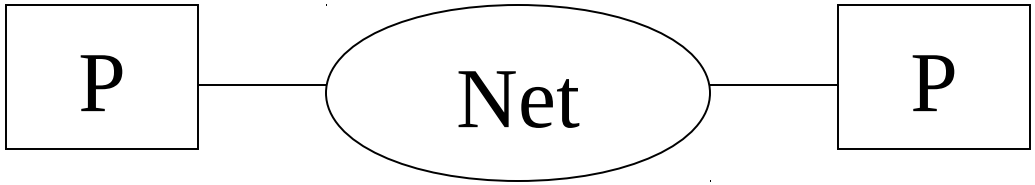
CS 347:
Distributed Databases and Transaction Processing
Notes 09: Network Partitions

Hector Garcia-Molina

Network partitions

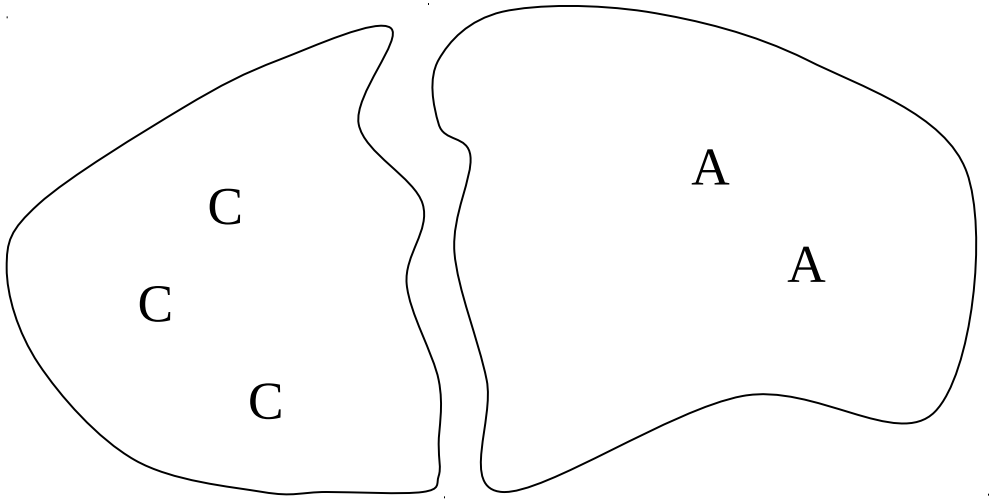
- Groups of nodes may be isolated or nodes may be slow in responding



- Where are partitions of interest?
  - True network partitions (disaster)
  - Single node failure cannot be distinguished from partition (e.g., ethernet board fails)
  - Phone-in, wireless networks
  - Autonomous nodes

No data replication

- If data unavailable, we are stuck...
- A problem: partition during commit protocol



Quorums

$C_1 = \{\{a,b,c\}, \{a,b,d\}, \{a,c,d\}, \{b,c,d\}\}$
$A_1 = \{\{a,b\}, \{a,c\}, \{a,d\}, \{b,c\}, \{b,d\}, \{c,d\}\}$

Important property: $X \in C_1 \Rightarrow \forall Y \in A_1 \;\; X \cap Y \neq \emptyset$
$Y \in A_1 \Rightarrow \forall X \in C_1 \;\; X \cap Y \neq \emptyset$

Additional Properties for Quorums?

added

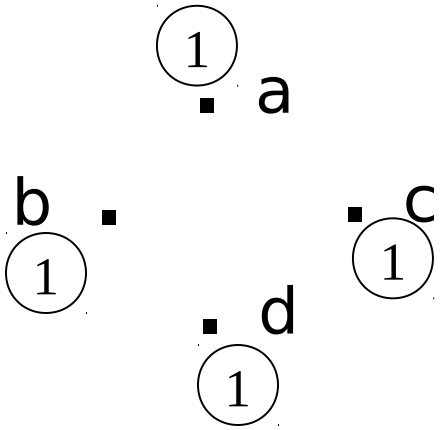
C_1 = {{a,b,c,d}, {a,b,d}, {a,c,d}, {b,c,d}}

A_1 = {{a,b}, {a,c}, {a,d}, {b,c}, {b,d}, {c,d}}

Is there a “better” quorum?

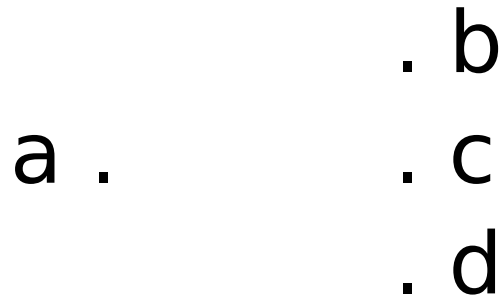
- Quorums can be implemented with vote assignments

To commit ≥ 3
To abort ≥ 2



Votes to commit + votes to abort > total votes

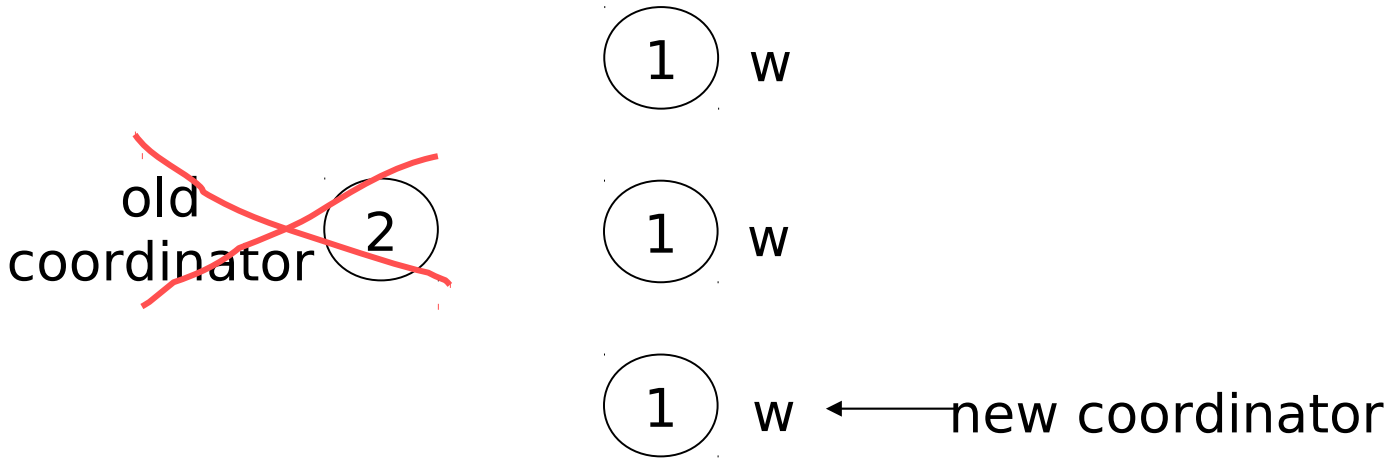
- Commit protocol must enforce quorum

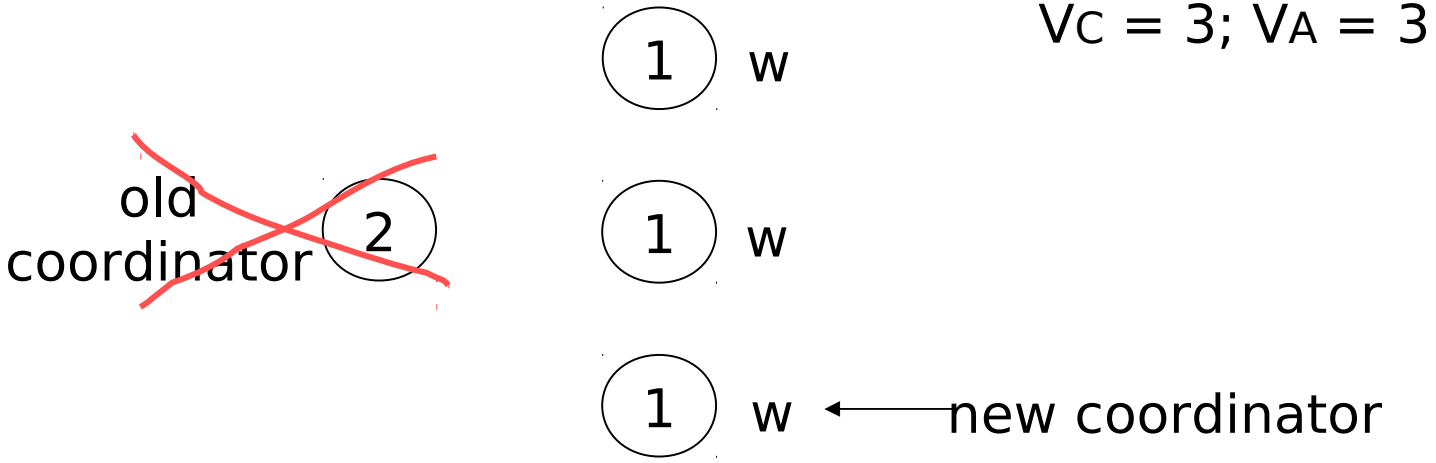


- If node knows transaction could have committed (aborted), if cannot abort (commit) even if abort (commit) quorum available
- All commit protocols are blocking (with network partitions)
    - can we do anything about it?

3PC Example

- To make commit decision: commit quorum
- To make abort decision: abort quorum
- Example:
  - votes for commit $V_C = 3$
  - votes for abort $V_A = 3$

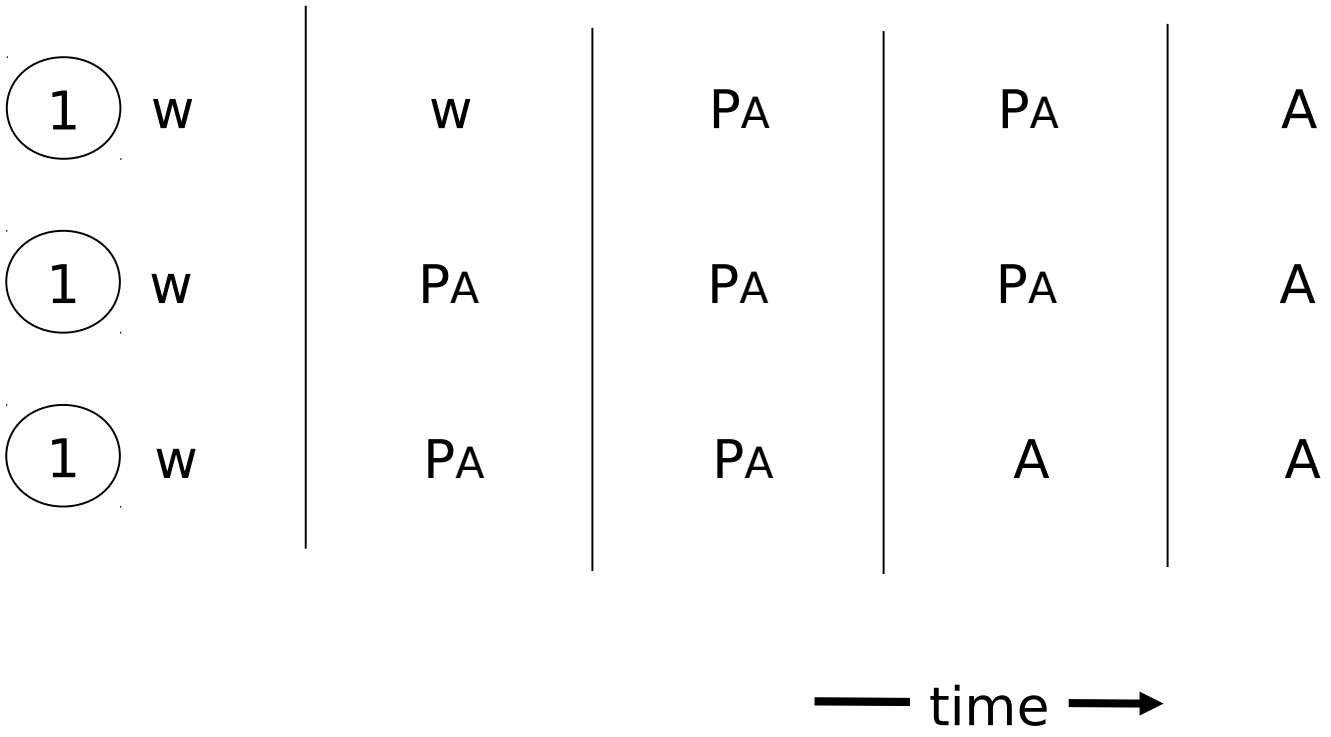




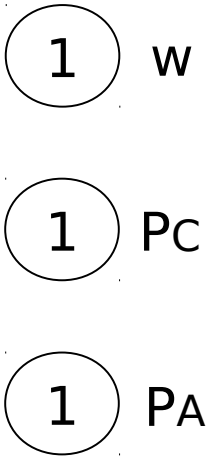
- Coordinator could NOT have committed
- Have abort quorum

$\Rightarrow$ Try to abort!

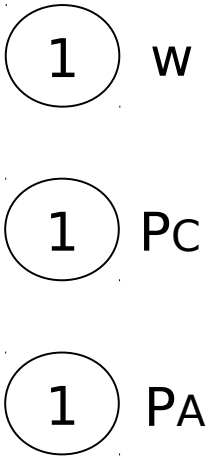
- Note: need to go to Prepare to Abort state (PA), analogous to Prepare to Commit state (PC)



- What if new group has following states?

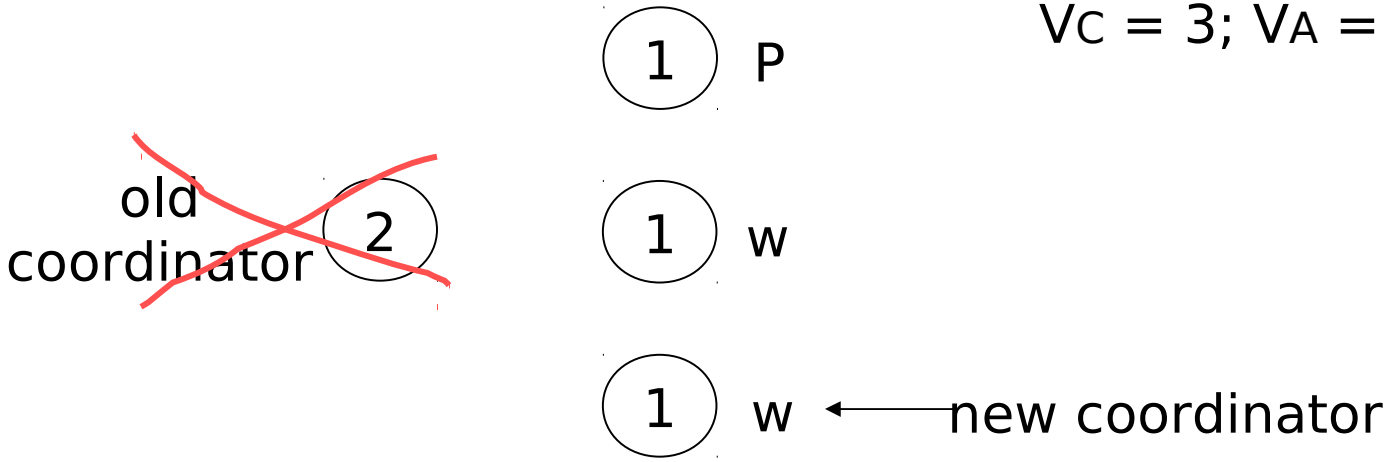


- What if new group has following states?

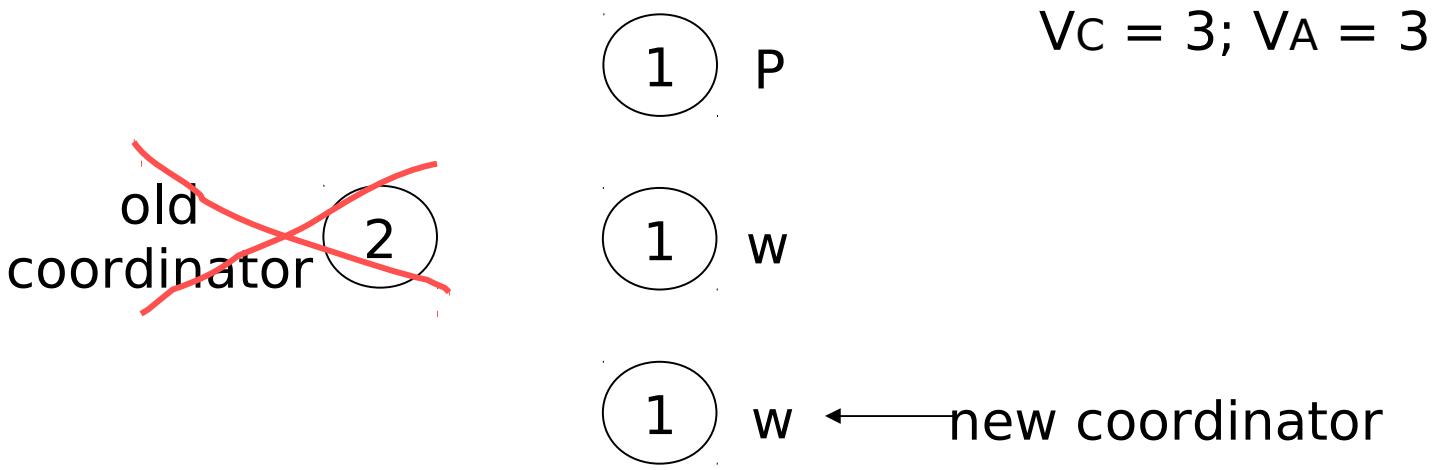


- Can this happen? How?
- Could transaction have aborted?
- Could transaction have committed?
- What do we do? Block?

Another 3PC Example



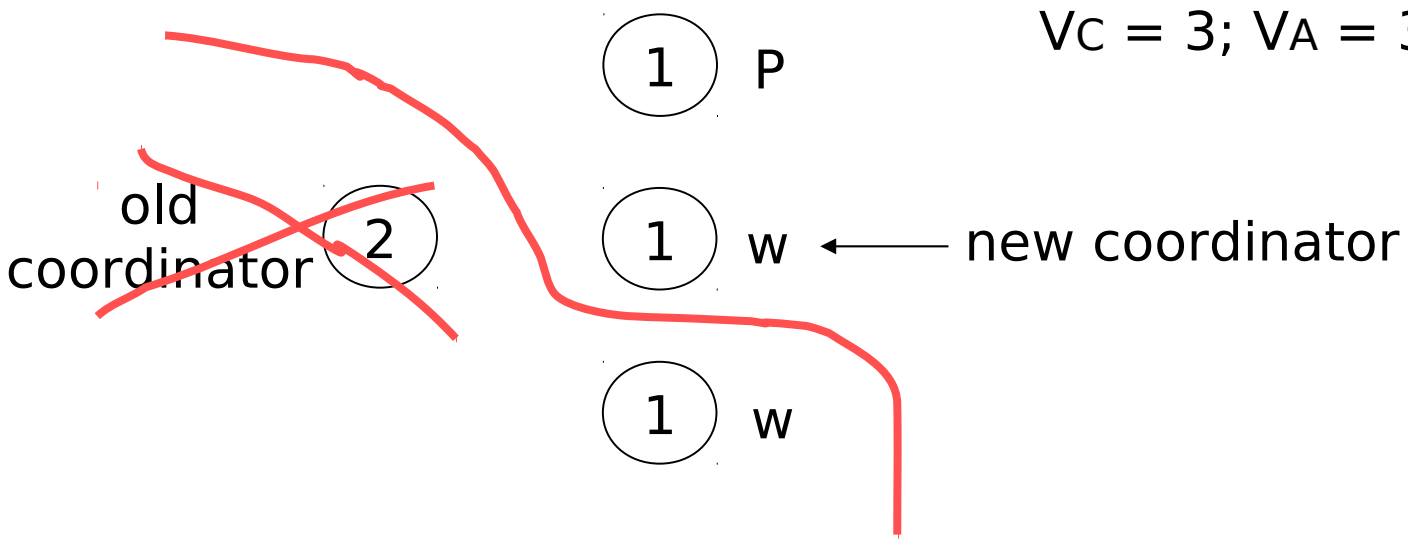
Another 3PC Example



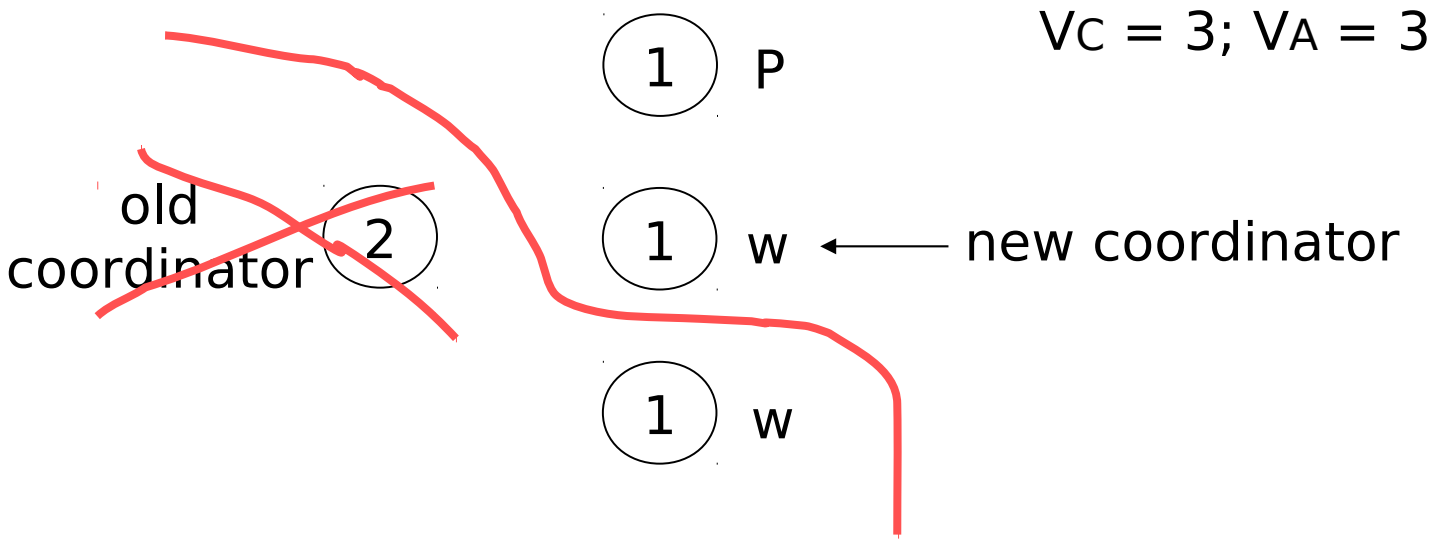
- Coordinator could not have aborted
- Have commit quorum

$\Rightarrow$ Try to commit!

Yet Another 3PC Example



Yet Another 3PC Example

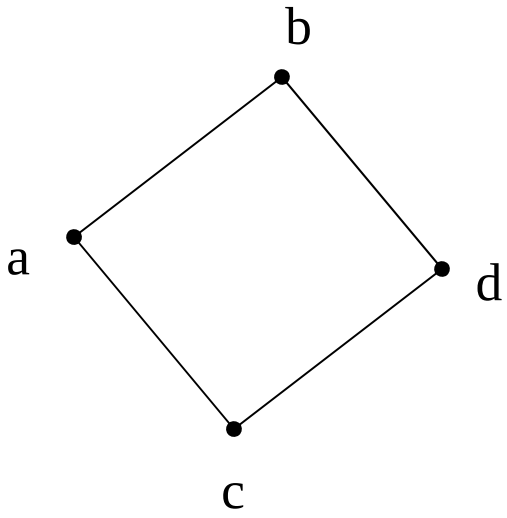


- Not enough votes!

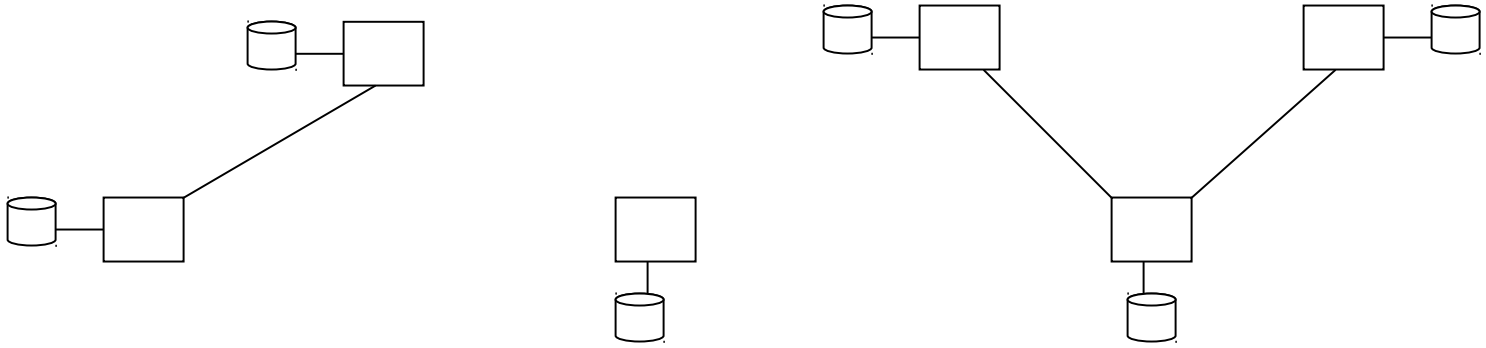
$\Rightarrow$ Block!

Not all quorums can be implemented via votes

C_2 = {{a,b}, {c,d}}
A_2 = {{a,c}, {a,d}, {b,c}, {b,d}}



Partitions and data replication



Options:

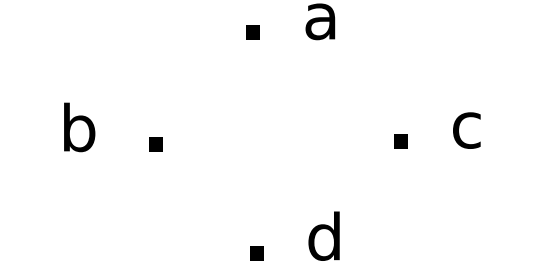
(1) all copies required for updates

(2) Group may update, but at most one (at a time)

(3) Any group may update

Updates by at most one group

Coterie



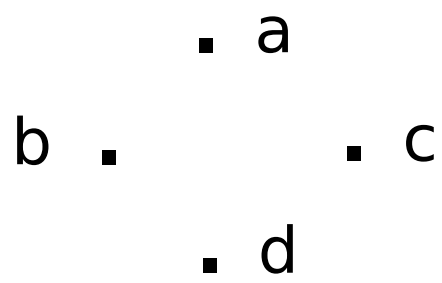
$C_1 = \{\{a,b,c\}, \{a,b,d\}, \{a,c,d\}, \{b,c,d\}\}$
$C_2 = \{\{a,b\}, \{a,c\}, \{a,d\}, \{b,c,d\}\}$

$X_1 = \{\{a,b\}, \{c,d\}\}$ not valid

Important property:

$$S \in C \Rightarrow \forall G \in C,\ S \cap G \neq \emptyset$$

Reading replicated data



$C_1 = \{\{a,b,c\}, \{a,b,d\}, \{a,c,d\}, \{b,c,d\}\}$

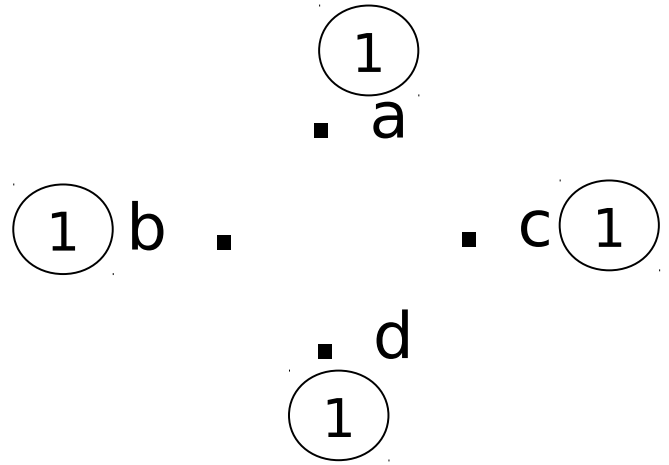
$R_1 = \{\{a,b\}, \{a,c\}, \{a,d\},\{b,c\}, \{b,d\}, \{c,d\}\}$

$C_2 = R_2 = \{\{a,b\}, \{a,c\}, \{a,d\}, \{b,c,d\}\}$

Reading replicated data - Votes

C_1 = {{a,b,c}, {a,b,d}, {a,c,d}, {b,c,d}}
R_1 = {{a,b}, {a,c}, {a,d},{b,c}, {b,d}, {c,d}}

to write get 3 votes (V_w)
to read get 2 votes (V_R)
($2 V_w > T$, $V_w + V_R > T$)



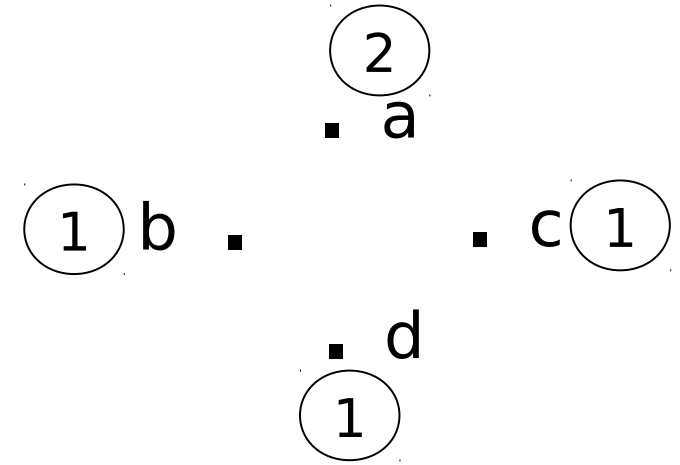
Reading replicated data - Votes

$C_2 = R_2 = \{\{a,b\}, \{a,c\}, \{a,d\}, \{b,c,d\}\}$

to write get 3 votes (V_w)

to read get 3 votes (V_R)

($2 V_w > T$, $V_w + V_R > T$)



Incidentally, which one is "better"?

C_1 = {{a,b,c}, {a,b,d}, {a,c,d}, {b,c,d}}

R_1 = {{a,b}, {a,c}, {a,d},{b,c}, {b,d}, {c,d}}

C_2 = R_2 = {{a,b}, {a,c}, {a,d}, {b,c,d}}

Note

- Note: not all coteries have vote assignments

Note

- Note: not all coteries have vote assignments

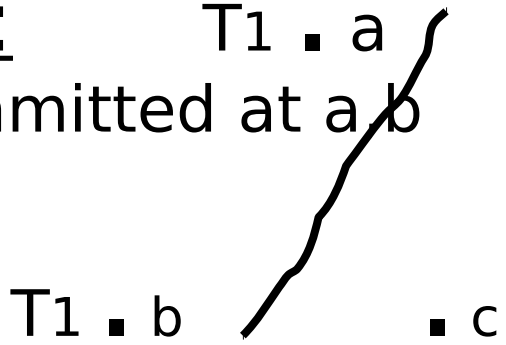
Nodes = {a,b,c,d,e,f}

C = { {a,b}, {a,c,d}, {a,c,e}, {a,d,f}, {a,e,f}, {b,c,f}, {b,d,e} }

A Problem

Example: a 3 node system, 1 vote each node, replicated data

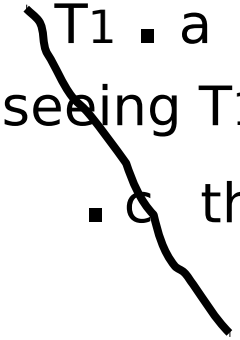
Now: T1 . a T1 is committed at a, b



Later: T1 ▪ a T2 reads at c

(not seeing T1);

▪ b ▪ c then writes and commits at a, c

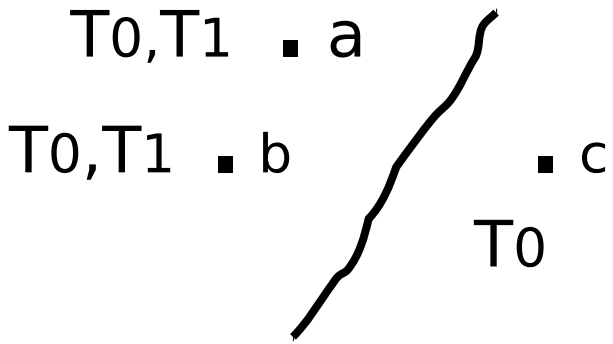


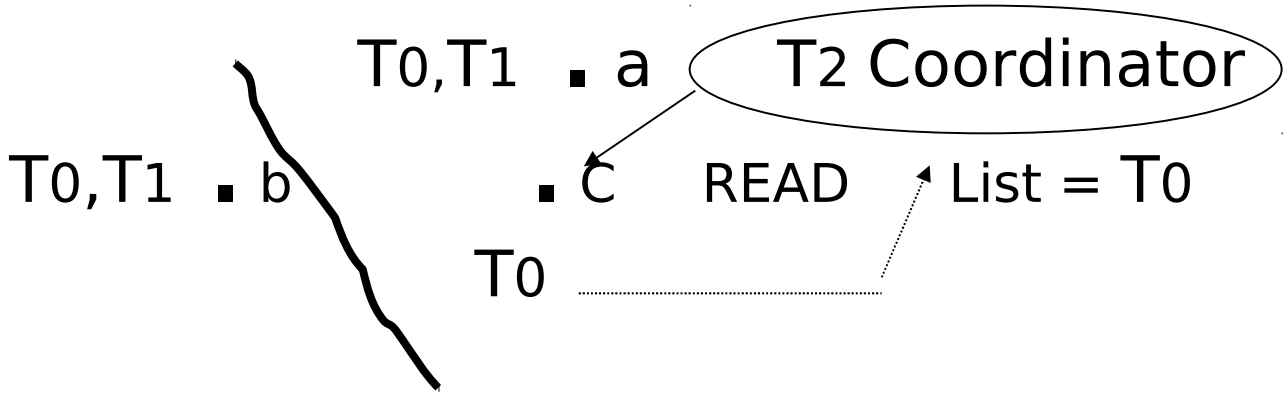
Solution

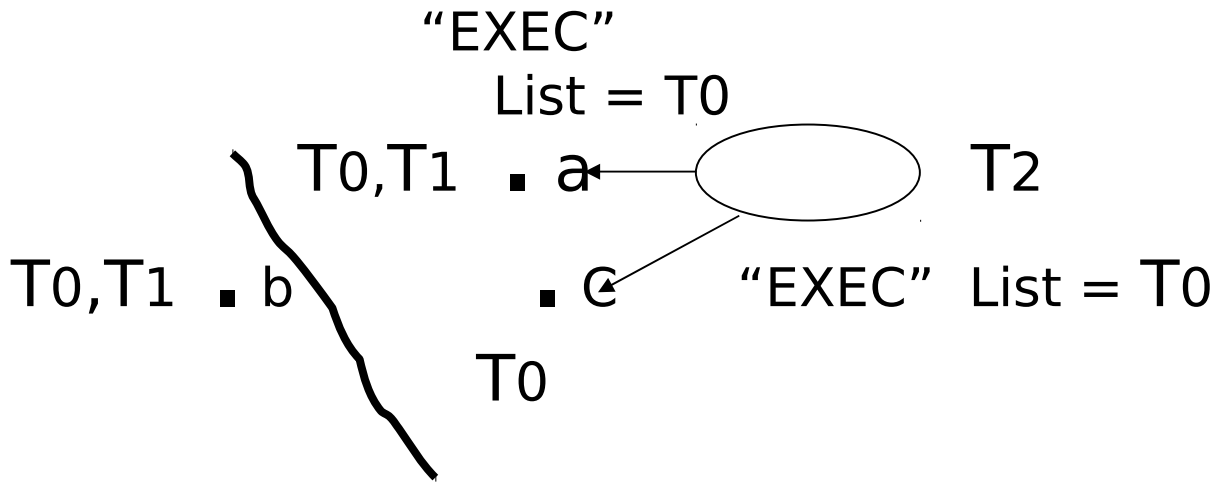
- each node keeps list of committed transactions
- compare list at read site with those at write sites
- update sites that missed transactions

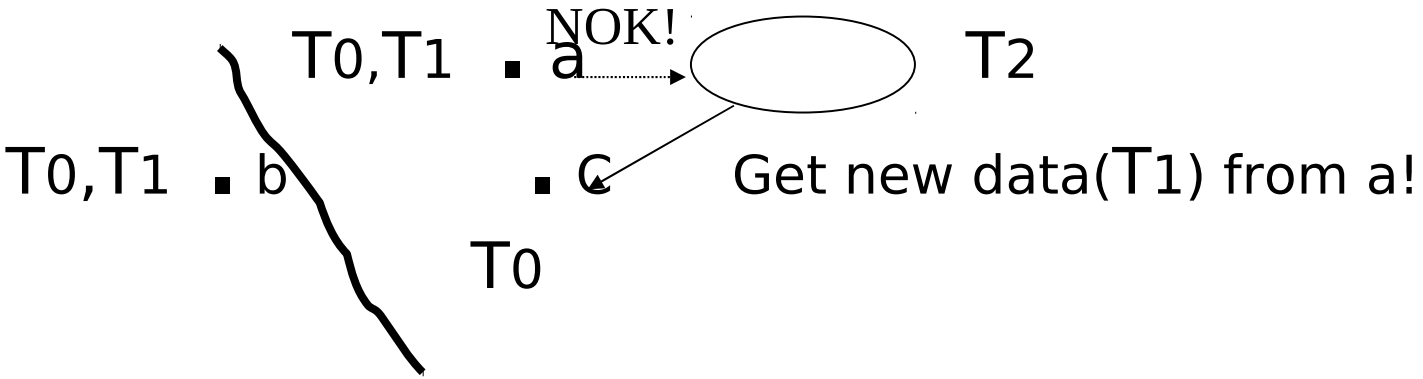
Example Revisited

Initially




T0,T1
a
T2 Coordinator
T0,T1
b
C
READ
List = T0
T0

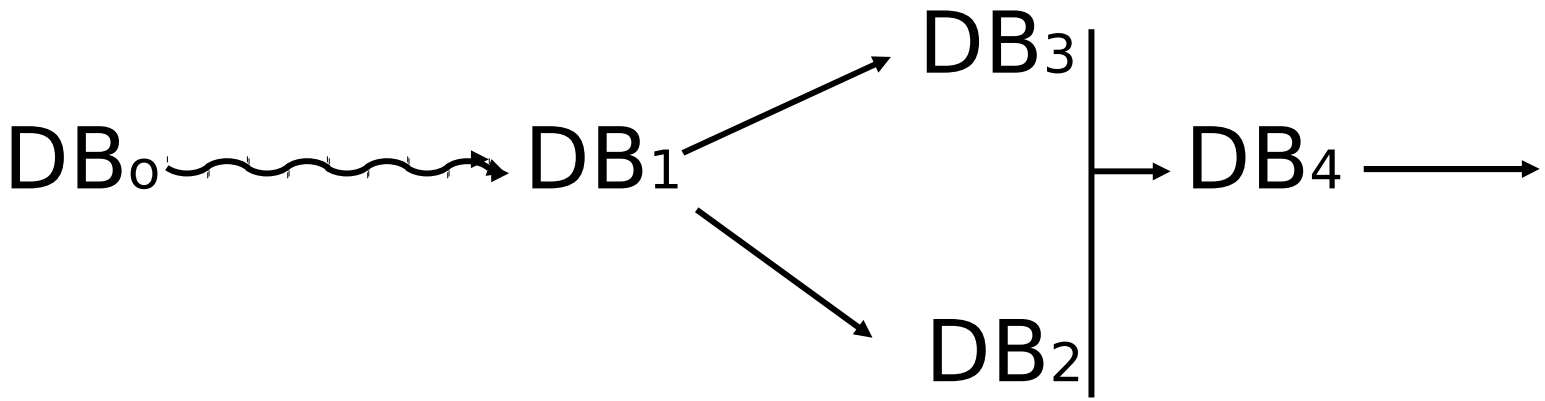

“EXEC”
List = T0
T0,T1
a
T2
T0,T1
b
c
“EXEC”
List = T0
T0


NOK!
T0,T1
a
T2
T0,T1
b
c
Get new data(T1) from a!
T0

- Each node must keep updates for transactions until all nodes have seen them

  - interesting problem...

Separate Operational Groups

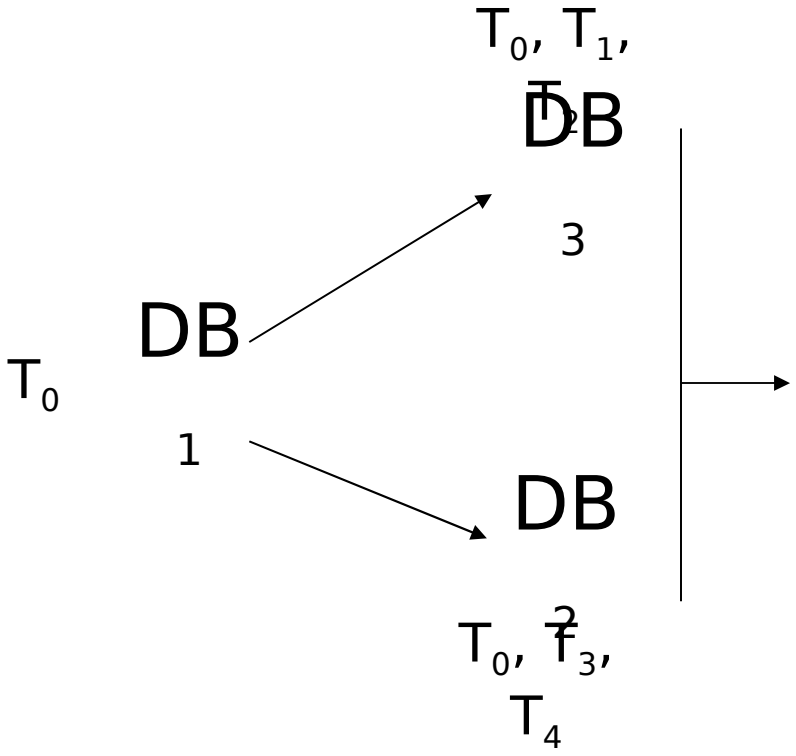


For integration

(1) Compensate transactions to make schedules match

(2) Data-patch: semantic fix

Example: compensation



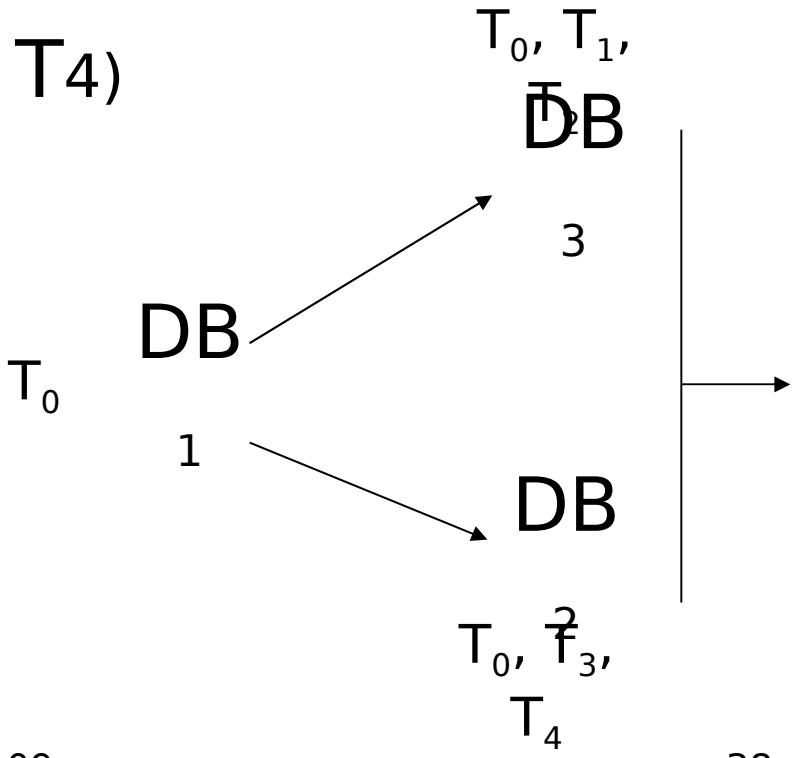
- Say T_1 commutes with T_3 and T_4

  E.g.: no conflicting operations

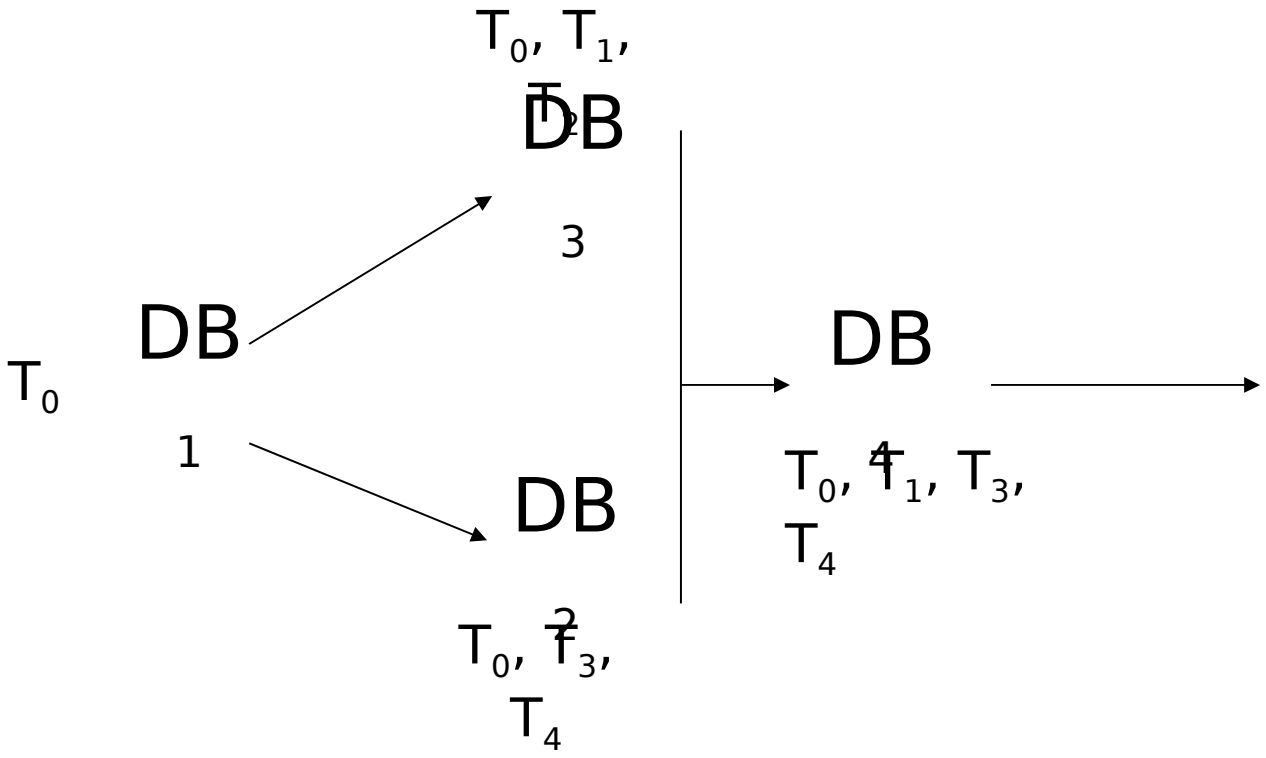
- At DB_2:

  Schedule = T_0, T_3, T_4, T_1

  (equivalent to T_0, T_1, T_3, T_4)



- At DB_3:
  Schedule = $T_0, T_1, T_2, T_2^{-1}, T_3, T_4$
  (equivalent to T_0, T_1, T_3, T_4)

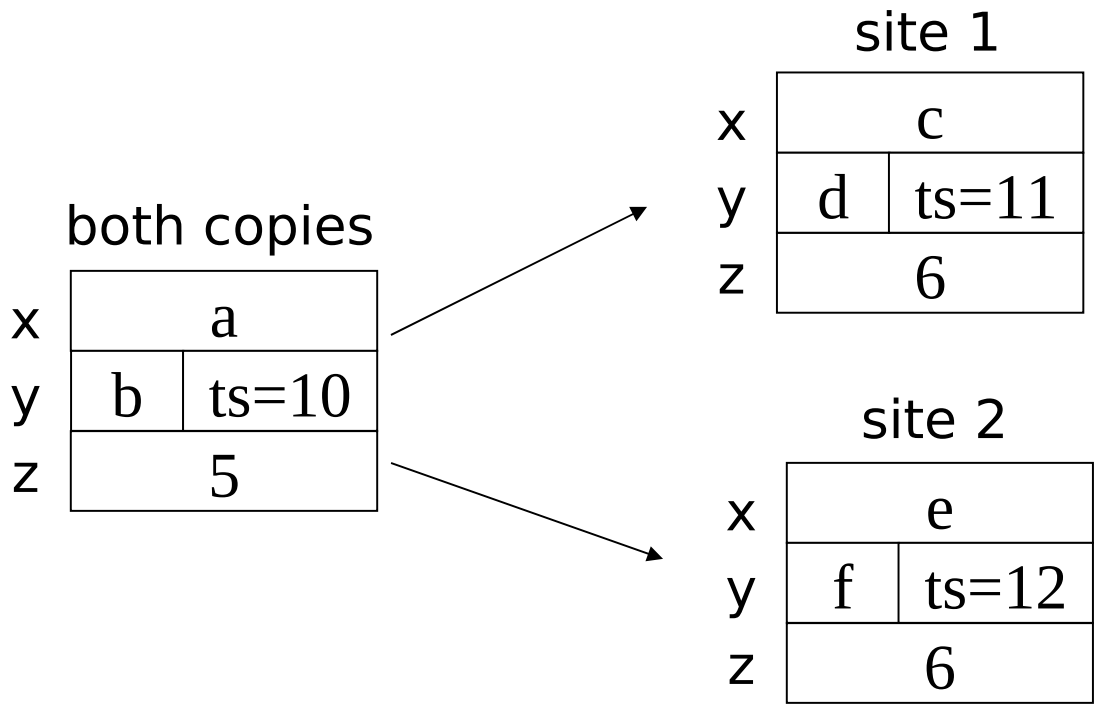


- In general:

  Based on characteristics of transactions, can “merge” schedules

Example: Data Patch

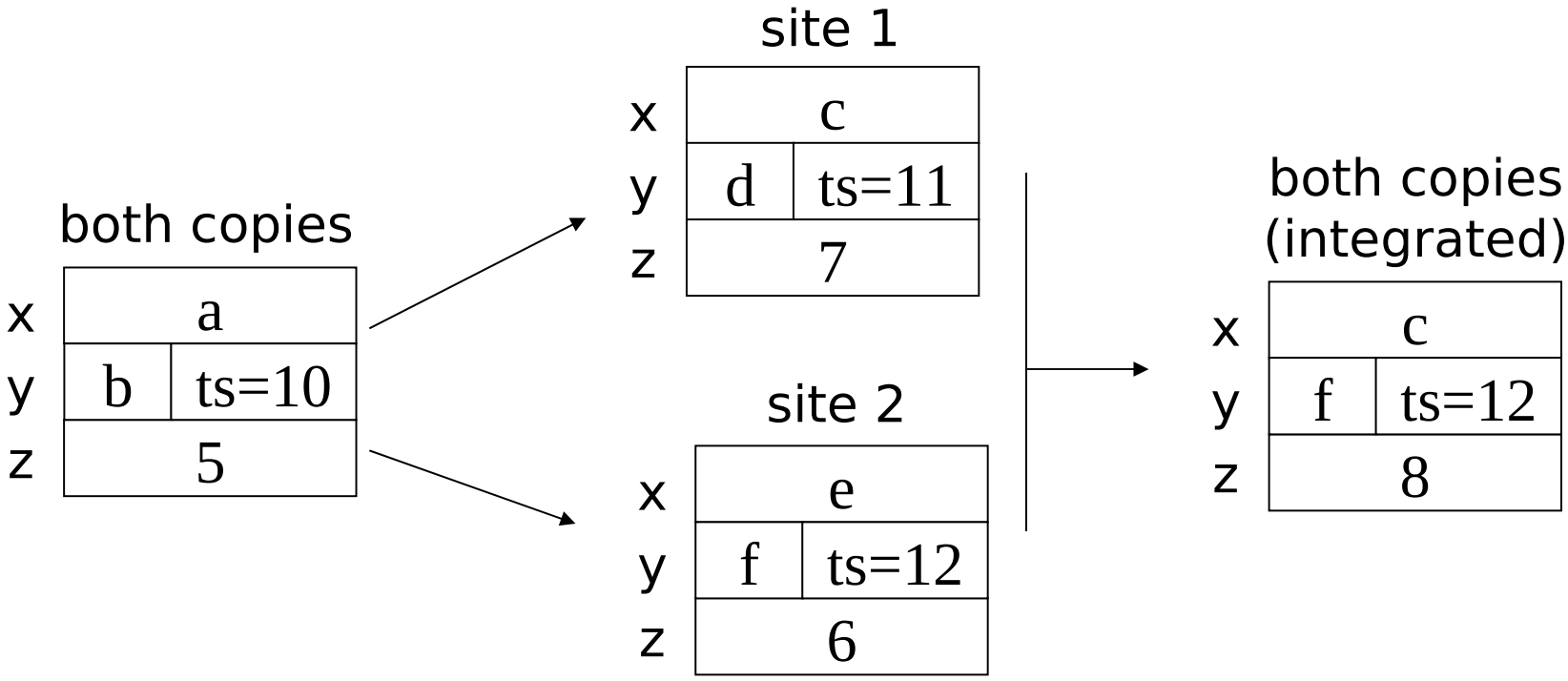
- forget schedules!
- integrate differing values via “rules”



Simple rules

- For X: site 1 wins
- For Y: latest timestamp wins
- For Z: add increments

For X: site 1 wins
For Y: latest timestamp wins
For Z: add increments



for Z:
8 = 5 + site 1 increment
+ site 2 increment = 5 + 2 + 1

Network Partitions: Summary

- No replicated data
  - abort, commit quorums
  - commit protocols
- Replicated data
  - at most one operational group
    - coterie
    - propagation of updates
  - multiple operational groups