Outline

- Introduction
- Background
- Distributed DBMS Architecture
- Distributed Database Design
- Distributed Query Processing
- Distributed Transaction Management
  - Transaction Concepts and Models
  - Distributed Concurrency Control
  - Distributed Reliability
- Building Distributed Database Systems (RAID)
- Mobile Database Systems
- Privacy, Trust, and Authentication
- Peer to Peer Systems

Useful References (1)

- S. B. Davidson, *Optimism and consistency in partitioned distributed database systems*, ACM Transactions on Database Systems 9(3): 456-481, 1984.
- S. B. Davidson, H. Garcia-Molina, and D. Skeen, *Consistency in Partitioned Networks,* ACM Computer Survey, 17(3): 341-370, 1985.
- B. Bhargava, *Resilient Concurrency Control in Distributed Database Systems*, IEEE Trans. on Reliability, R-31(5): 437-443, 1984.
- Jr. D. Parker, et al., *Detection of Mutual Inconsistency in Distributed Systems*, IEEE Trans. on Software Engineering, SE-9, 1983.

Useful References (2)

- Bharat Bhargava, *Site Recovery in Replicated Distributed Database Systems*, Computer Science Technical Reports, report number: 85-564

- B. Bhargava, *Transaction Processing and Consistency Control of Replicated Copies during Failures in Distributed Databases*, Proceedings of Conference on Current Issues in Database Systems, Newark, May 1986.

- B. Bhargava, P.L. Ng, *A dynamic majority determination algorithm for reconfiguration of network partitions*, Information Sciences, Vol 46 No 1-2, 1988

Site Failure and Recovery

- Maintain consistency of replicated copies during site failure.
- Announce failure and restart of a site.
- Identify out-of-date data items.
- Update stale data items.

Main Ideas and Concepts

- Read one Write all available protocol.
- Fail locks and copier transactions.
- Session vectors.
- Control transactions.

Measurements of Behavior of Site Failure/Network Partitioning

- How did partition occur?
- How many objects become out of date?
- How many transactions are blocked or aborted?
- Availability.
- Effects of creating more copies when failures occur.

Replication Control

Replication Control enables a distributed database system to operate effectively despite periods of failures and communication breakdowns.

At Purdue University, we have developed the Mini-RAID system for conducting experiments in replication control.

In the following slides, an example based on the system is presented. The configuration for our example is:

- 3 database sites (S0, S1, S2)
- a fully replicated database of 4 objects (a, b, c, d)

Dynamic Majority for Network Partitioning

- Majority of majority continues processing.
- Declare a tie when too few sites in majority follow optimistic approach and continue processing without commit.
- Full availability to read-only transactions (view serializability).*
- Arbitrary merges of partition to form majority once again.

* Journal of Information Science, Feb. 1988

Example of Replicated Copy Control Problem

1. $T_a = R_a[X]\ W_a[Y]$, $T_b = R_b[Y]\ W_b[X]$
   $X = \{x_1, x_2\}$, $Y = \{y_1, y_2\}$
   x_1 and y_1 are written by copier transactions T_c and T_d
   $H = r_a[x_1]\ r_b[y_1]$ (site 1 crashes) $w_a[y_2]\ w_b[x_2]$
   (site 1 recovers) . . . $r_c[x_2]\ w_c[x_1]\ r_d[y_2]\ w_d[y_1]$
   H is serializable, but has the same effect as an incorrect history
   $r_a[x_1]\ r_b[y_1]\ w_a[y_2]\ w_b[x_2]\ w_b[x_2]\ w_b[x_1]\ w_a[y_1]$

2. $T_a = R_a[X]\ W_a[Y]$, $T_b = R_b[Y]\ W_b[X]$
   $X = \{x_1, x_2\}$, $Y = \{y_1, y_2\}$
   Copier transaction T_c reads x_1 and writes to x_2
   $H =$ (site 2 crashes) $r_a[x_1]\ r_b[y_1]\ w_a[x_1]$
   (site 2 recovers) . . . $r_c[x_1]\ w_c[x_2]\ w_b\ [x_1]$
   Missing update – x_2 reflects $W_a[X]$ but not $W_b[X]$

Logical and Physical Copies of Data

X: Logical data item

x_k: A copy of item X on site k

Strict read-one write all (ROWA) requires reading at Least at one site and writing at all sites.

$$\text{Read}(X) = \vee \{\text{read}(x_k),\ x_k \in X\}$$

$$\text{Write}(X) = \wedge \{\text{write}(x_k),\ x_k \in X\}$$

New Protocols and Algorithm

- Multiple Site Failure and Recovery*
  - Session numbers.
  - Read one write all available (ROWAA).
  - Fail-locks.
  - Copier Transactions.
  - Database available as long as a single copy is up.
  - Operation site do little work for failed site(s).
  - Failed site recovers on demand or automatically via updates on open sites.

---

* Journal of Management Information Systems, Vol. 4, No. 2, 1987.

Session Numbers and Nominal Session Numbers

- Each operational session of a site is designated with an integer, session number.
- Failed site has session number = 0.
- as[k] is actual session number of site k.
- ns_i[k] is nominal session number of site k at site i.
- NS[k] is nominal session number of site k.

A nominal session vector consisting of nominal session numbers of all sites is stored at each site.

ns_i is the nominal session vector at site i.

Read one Write all Available (ROWAA)

Transaction initiated at site i, reads and writes as follows:

$$\text{Read}(X) = \vee \{\text{read}(x_k),\ x_k \in X \text{ and } ns_i[k] \neq 0\}$$

$$\text{Write}(X) = \wedge \{\text{write}(x_k),\ x_k \in X \text{ and } ns_i[k] \neq 0\}$$

At site k, the $ns_i(k)$ is checked against as as[k]. If they are not equal, the transaction is rejected.

Transaction is not sent to a failed site for whom $ns_i(k) = 0$.

Control Transactions for Announcing Recovery

Type 1: Claims that a site is nominally up.

Updates the session vector of all operational sites with the recovering site's new session number.

New session number is one more than the last session number (like an incarnation).

Example:

as[k] = 1 initially

as[k] = 0 after site failure

as[k] = 2 after site recovers

as[k] = 0 after site failure

as[k] = 3 after site recovers second time

Control Transactions for Announcing Failure

Type 2: Claims that one or more sites are down.

Claim is made when a site attempts and fails to access a data item on another site.

Control transaction type 2 sets a value 0 for a failed site in the nominal session vectors at all operational sites.

This allows operational sites to avoid sending read and write requests to failed sites.

Fail Locks

- A fail lock is set at an operational site on behalf of a failed site if a data item is updated.
- Fail lock can be set per site or per data item.
- Fail lock used to identify out-of-date items (or missed updates) when a site recovers.
- All fail locks are released when all sites are up and all data copies are consistent.

Copier Transaction

- Copier transaction reads current values (for failed lock items) on operational sites and writes on out of data items on the recover site.

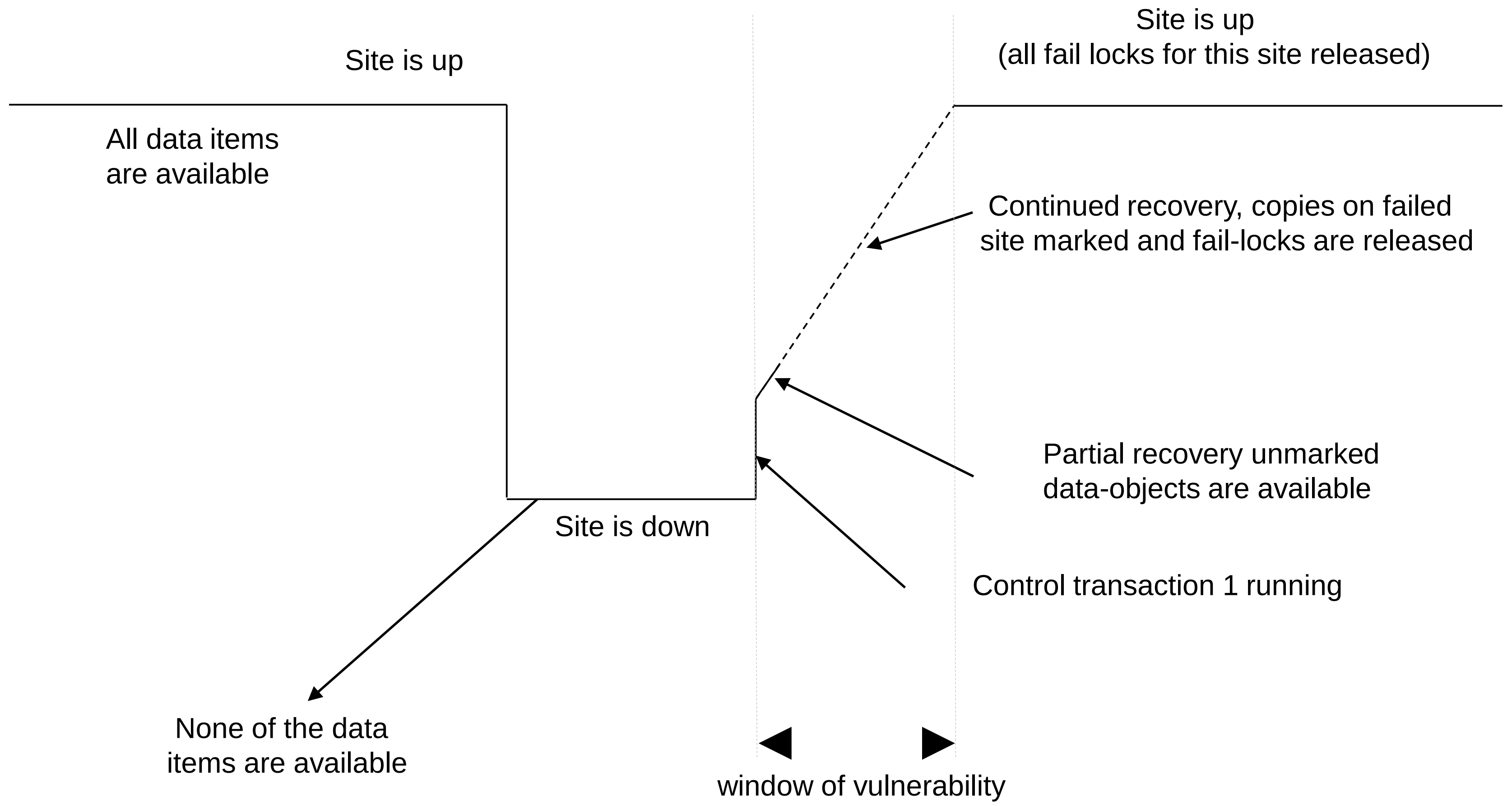
Site Recovery Procedure

1. When a site k starts, it loads its actual session number as[k] with 0, meaning that the site is ready to process control transactions but not user transactions.
2. Next, the site initiates a control transaction of type 1. It reads an available copy of the nominal session vector and refreshes its own copy. Next this control transaction writes a newly chosen session number into $ns_i[k]$ for all operational sites I including itself, but not as[k] as yet.

Site Recovery Procedure

3. Using the fail locks on the operational site, the recovering site marks the data copies that have missed updates since the site failed. Note that steps 2 and 3 can be combined.
4. If the control transaction in step 2 commits, the site is nominally up. The site converts its state from recovering to operational by loading the new session number into as[k]. If step 2 fails due to a crash of another site, the recovering site must initiate a control transaction of type 2 to exclude the newly crashed site, and then must try step 2 and 3 again. Note that the recovery procedure is delayed by the failure of another site, but the algorithm is robust as long as there is at least one operational site coordinating the transaction in the system.

Status in site recovery and Availability of Data Items for Transaction Processing



Transaction Processing when Network Partitioning Occurs

Three Alternatives after Partition

A. Allow each group of nodes to process new transactions
B. Allow at most one group to process new transactions
C. Halt all transaction processing

Alternative A

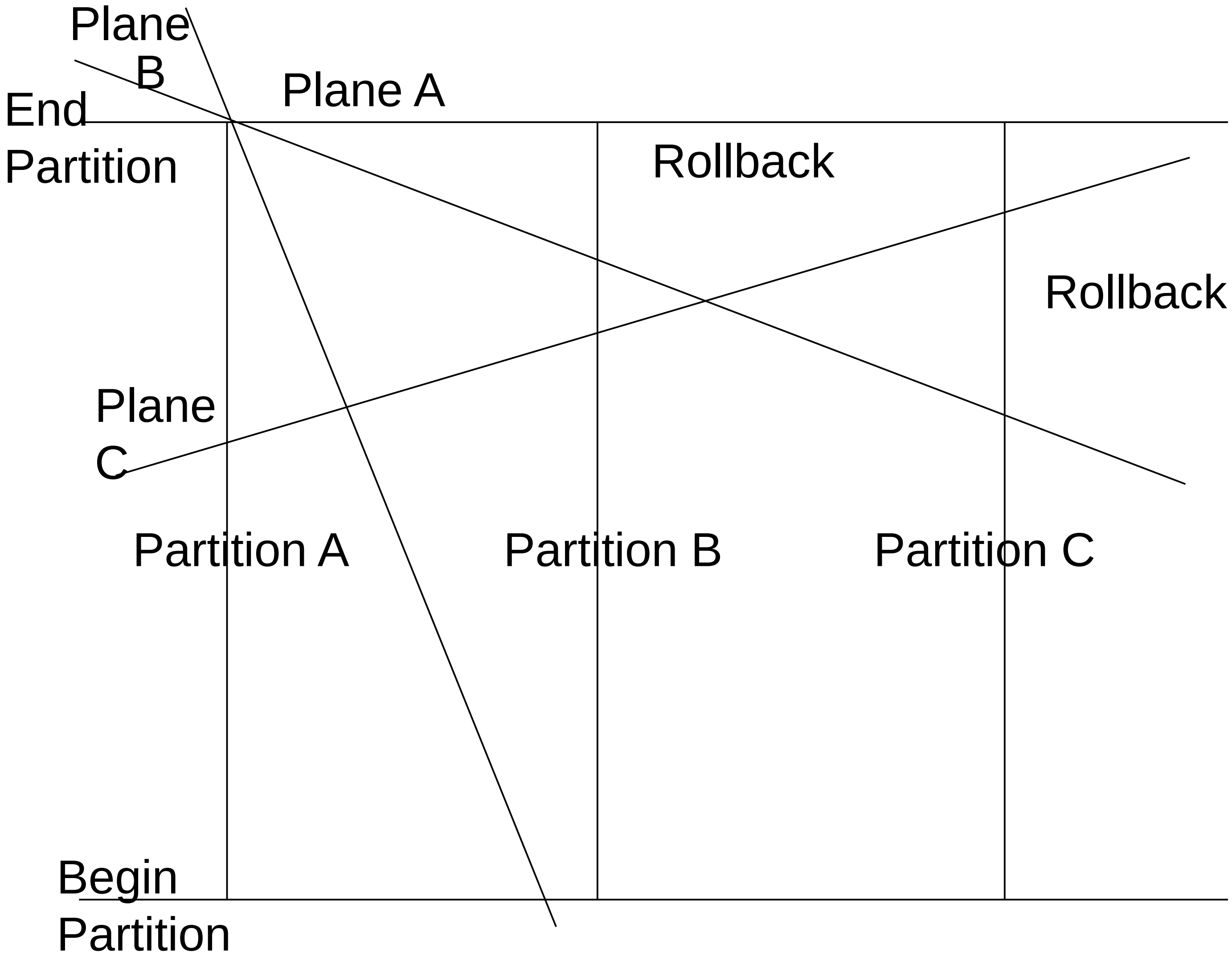
- Database values will diverge database inconsistent when partition is eliminated
    - Undo some transactions
        - detailed log
        - expensive
    - Integrate the inconsistent values
        - item I has values v_1, v_2
        - new value = $v_1 + v_2$ – value of i at partition

Network Partition Alternatives

Alternative B

- How to guarantee only one group processes transactions
    - assign a number of points to each node   partition with majority of points proceeds
- Both partition and node failure cases are equivalent in the sense in both situations we have a group of nodes which know that no other node outside the group may process transactions
- What if ∃ no group with a majority?
    - should we allow transactions to proceed?
    - commit point?
    - delay the commit decision?
    - force transaction to commit or cancel?

Planes of Serializability



Merging Semi-Committed Transactions

Merger of Semi-Committed Transactions From Several Partitions

- Combine DCG, DCG_2, --- DCG_N
  - (DCG is Dynamic Cyclic Graph)
  - (minimize rollback if cycle exists)
- NP-complete
  - (minimum feedback vertex set problem)
- Consider each DCG as a single transaction
- Check acyclicity of this N node graph
  - (too optimistic!)
- Assign a weight to transactions in each partition
- Consider DCG_1 with maximum weight
- Select transactions from other DCG's that do not create cycles

Breaking Cycle by Aborting Transactions

Two Choices

- Abort transactions who create cycles
- Consider each transaction that creates cycle one at a time.

Abort transactions which optimize rollback

(complexity $O(n^3)$)

Minimization not necessarily optimal globally

Commutative Actions and Semantics

Semantics of Transaction Computation

- Commutative
  - Give $5000 bonus to every employee
- Commutativity can be predetermined or recognized dynamically
- Maintain log (REDO/UNDO) of commutative and noncommutative actions
- Partially rollback transactions to their first noncommutative action

Compensating Actions

- Compensating Transactions
  - Commit transactions in all partitions
  - Break cycle by removing semi-committed transactions
  - Otherwise abort transactions that are invisible to the environment

    (no incident edges)
  - Pay the price of commiting such transactions and issue compensating transactions
- Recomputing Cost
  - Size of readset/writeset
  - Computation complexity

Network Partitioning

- Simple partitioning
  - Only two partitions
- Multiple partitioning
  - More than two partitions
- Formal bounds:
  - There exists no non-blocking protocol that is resilient to a network partition if messages are lost when partition occurs.
  - There exist non-blocking protocols which are resilient to a single network partition if all undeliverable messages are returned to sender.
  - There exists no non-blocking protocol which is resilient to a multiple partition.

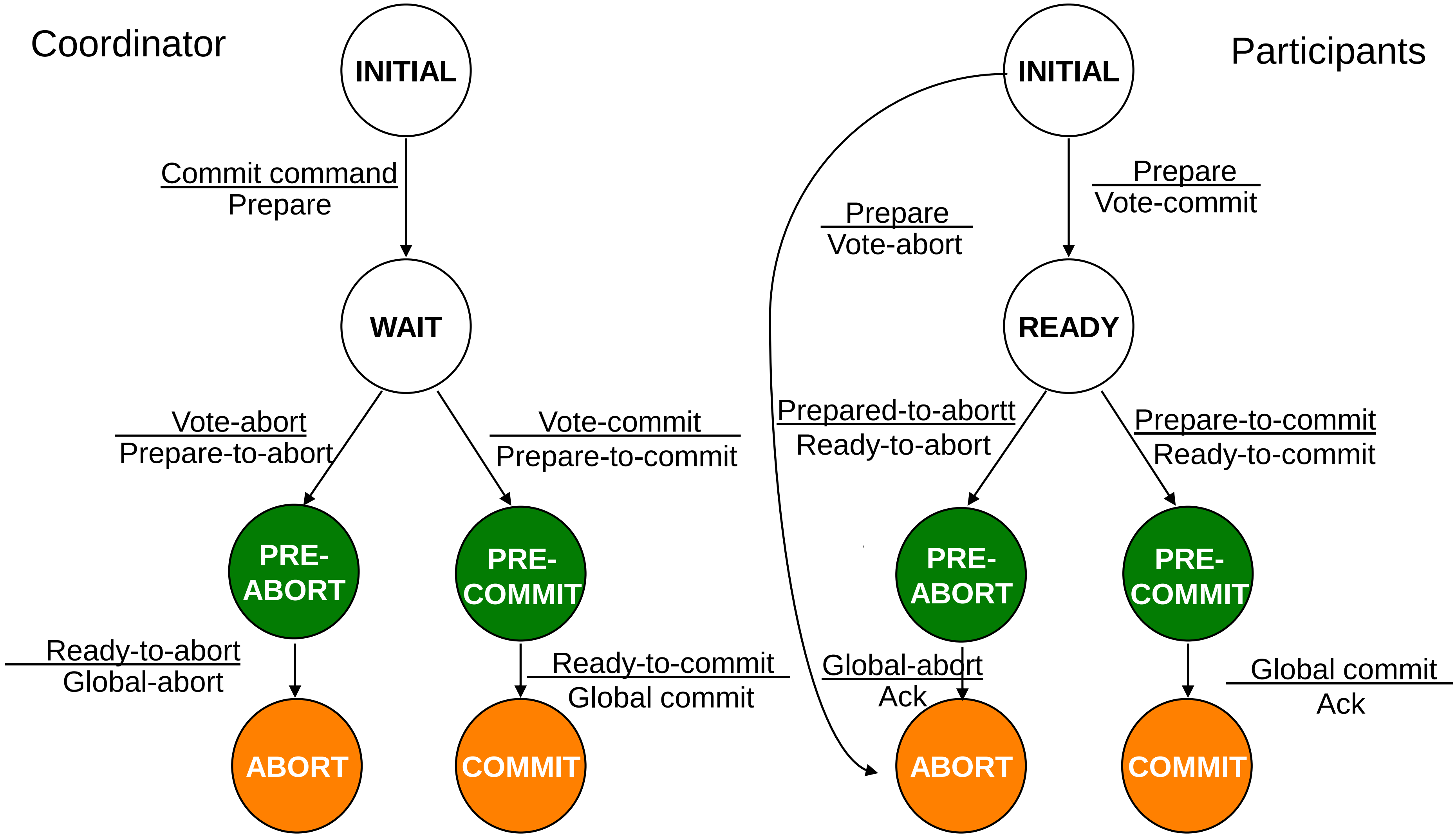
Independent Recovery Protocols for Network Partitioning

- No general solution possible
  - allow one group to terminate while the other is blocked
  - improve availability
- How to determine which group to proceed?
  - The group with a majority
- How does a group know if it has majority?
  - centralized
    - whichever partitions contains the central site should terminate the transaction
  - voting-based (quorum)
    - different for replicated vs non-replicated databases

Quorum Protocols for Non-Replicated Databases

- The network partitioning problem is handled by the commit protocol.
- Every site is assigned a vote Vi.
- Total number of votes in the system V
- Abort quorum V_a, commit quorum V_c
  - $V_a + V_c > V$ where $0 \leq V_a, V_c \leq V$
  - Before a transaction commits, it must obtain a commit quorum V_c
  - Before a transaction aborts, it must obtain an abort quorum V_a

State Transitions in Quorum Protocols



Quorum Protocols for Replicated Databases

- Network partitioning is handled by the replica control protocol.
- One implementation:
  - Assign a vote to each *copy* of a replicated data item (say V_i) such that $\Sigma_i V_i = V$
  - Each operation has to obtain a *read quorum* (V_r) to read and a *write quorum* (V_w) to write a data item
  - Then the following rules have to be obeyed in determining the quorums:
    - $V_r + V_w > V$ a data item is not read and written by two transactions concurrently
    - $V_w > V/2$ two write operations from two transactions cannot occur concurrently on the same data item

Use for Network Partitioning

- Simple modification of the ROWA rule:
  - When the replica control protocol attempts to read or write a data item, it first checks if a majority of the sites are in the same partition as the site that the protocol is running on (by checking its votes). If so, execute the ROWA rule within that partition.
- Assumes that failures are "clean" which means:
  - failures that change the network's topology are detected by all sites instantaneously
  - each site has a view of the network consisting of all the sites it can communicate with

Open Problems

- Replication protocols
  - experimental validation
  - replication of computation and communication
- Transaction models
  - changing requirements
    - cooperative sharing vs. competitive sharing
    - interactive transactions
    - longer duration
    - complex operations on complex data
  - relaxed semantics
    - non-serializable correctness criteria

Other Issues

- Detection of mutual inconsistency in distributed systems
- Distributed system with replication for
  - reliability (availability)
  - efficient access
- Maintaining consistency of all copies
  - hard to do efficiently
- Handling discovered inconsistencies
  - not always possible
  - semantics-dependent

Replication and Consistency

- Tradeoffs between

  degree of replication of objects access time of object

  availability of object (during partition)

  synchronization of updates

  (overhead of consistency)
- All objects should always be available.
- All objects should always be consistent.

  "Partitioning can destroy mutual consistency in the worst case".

Basic Design Issue:

  Single failure must not affect entire system (robust, reliable).

Availability and Consistency

- Previous work
  - Maintain consistency by:
    - Voting (majority consent)
    - Tokens (unique/resource)
    - Primary site (LOCUS)
    - Reliable networks (SDD-1)

  Prevent inconsistency at a cost does not address detection or resolution issues.

  Want to provide availability and correct propagation of updates.

View-serializability

	x	y
T_1	x'	y
T_2	x''	y
T_3	x''	y'

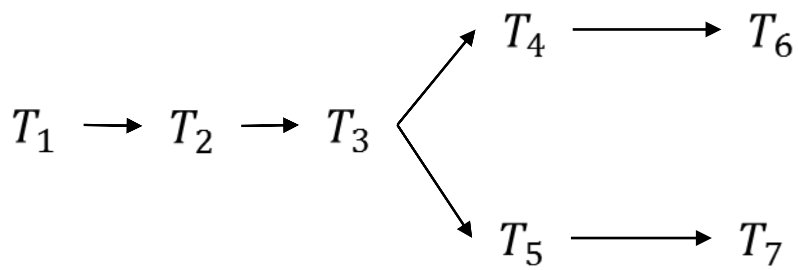
Partitioning

P_1(x token) P_2(y token)

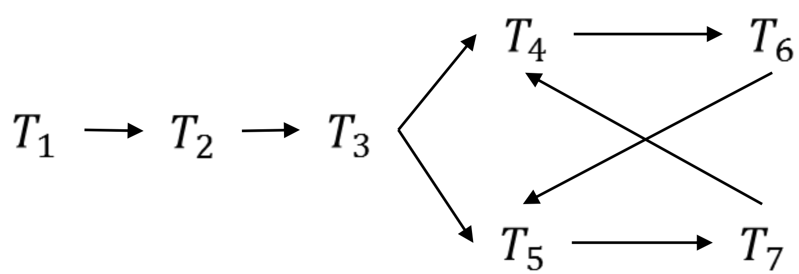
	$x''y'$	$x''y'$
T_4	$x'''y'$	$x''y'$
T_5	$x'''y'$	$x''y''$
T_6	...	...

Global conflict graph:

1)



2) Partitions merge.



Detecting Inconsistency

- Detecting Inconsistency

  Network may continue to partition or partially merge for an unbounded time.

  Semantics also different with replication:

  - naming, creation, deletion…
  - names in on partition do not relate to entities in another partition

  Need globally unique system name, and user name(s).

  Must be able to use in partitions.

Types of Conflicting Consistency

- System name consists of a
  - < Origin, Version > pair
  - Origin – globally unique creation name
  - Version – vector of modification history
- Two types of conflicts:
  - Name – two files have same user-name
  - Version – two incompatible versions of the same file.
- Conflicting files may be identical…
  - Semantics of update determine action
- Detection of version conflicts
  - Timestamp – overkill
  - Version vector – "necessary + sufficient"
  - Update log – need global synchronization

Version Vector

Version vector approach

  each file has a version vector

  (S_i : u_i) pairs

    S_i – Site on which the file is stored

    u_i – Number of updates on that site

Example: < A:4, B:2; C:0; D:1 >

Compatible vectors:

  one is at least as large as the other over all sites in vector

< A:1; B:2; C:4; D:3 > ← < A:0; B:2; C:2; D:3 >

< A:1; B:2; C:4; D:3 > ≠ < A:1; B:2; C:3; D:4 > (Not Compatible)

  (< A:1; B:2; C:4; D:4 >)

Additional Comments

Committed updates on site S_i will update u_i by one

Deletion/Renaming are updates

Resolution on site S_i increments u_i to maintain consistency later.

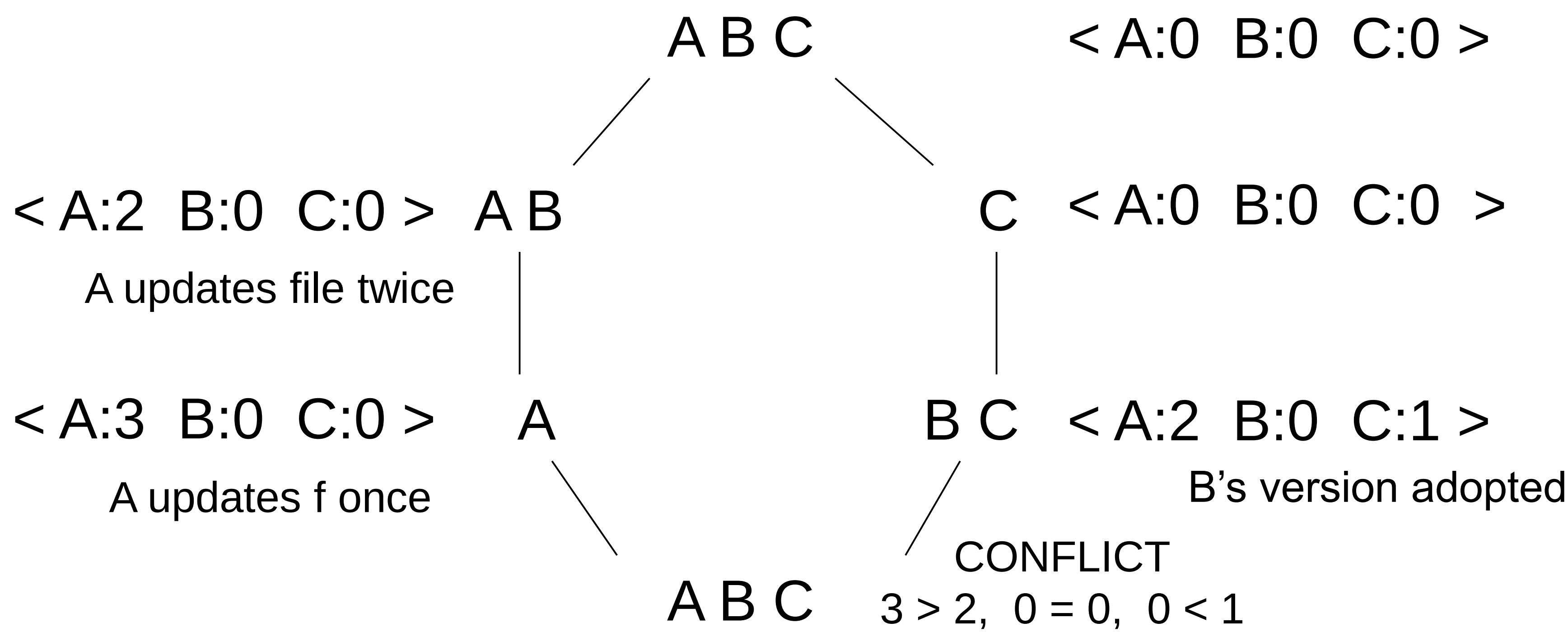
to Max S_i

Storing a file at new site makes vector longer by one site.

Inconsistency determined as early as possible.

Only works for single file consistency, and not transactions…

Example of Conflicting Operation in Different Partitions

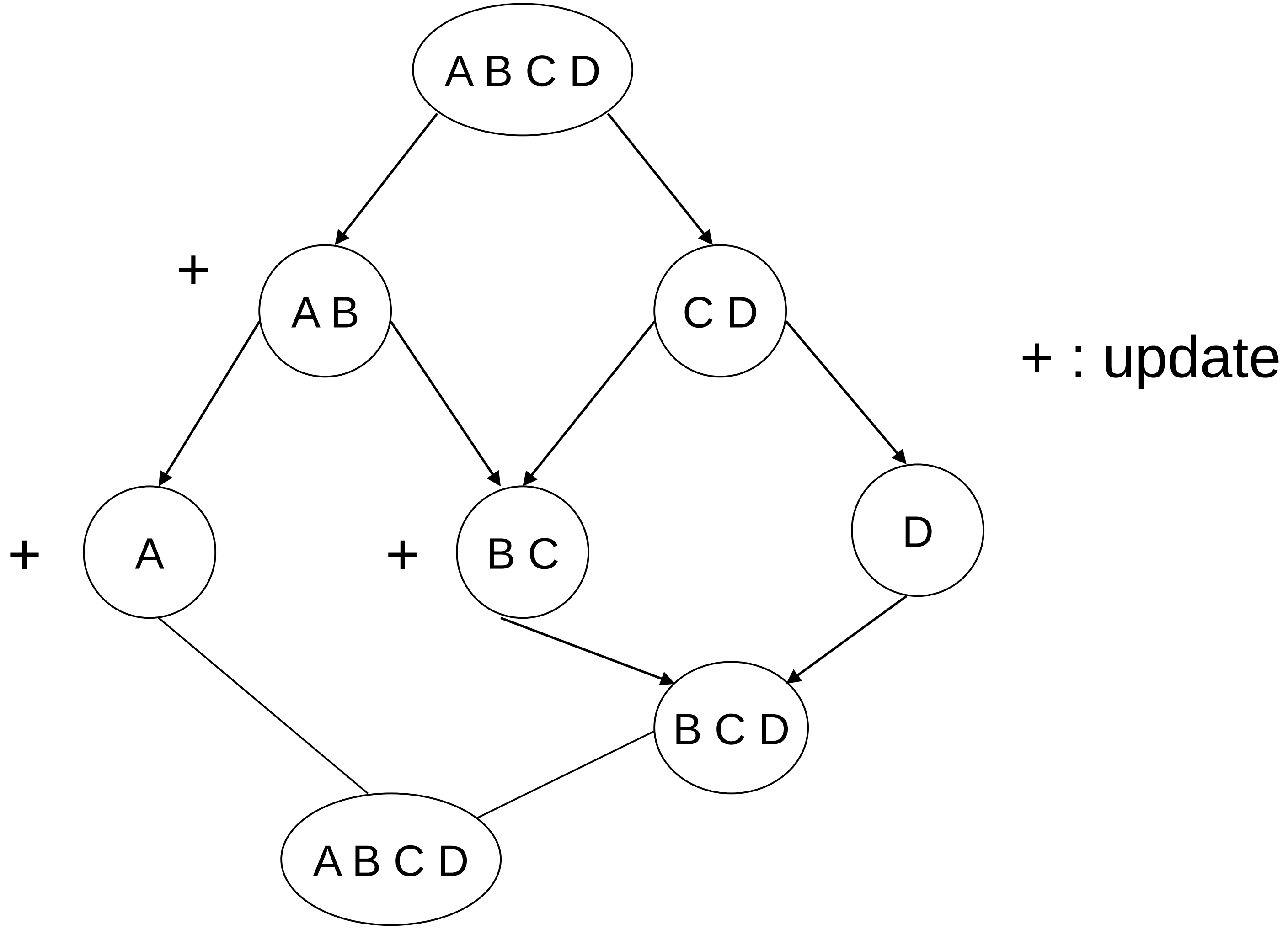


Version vector

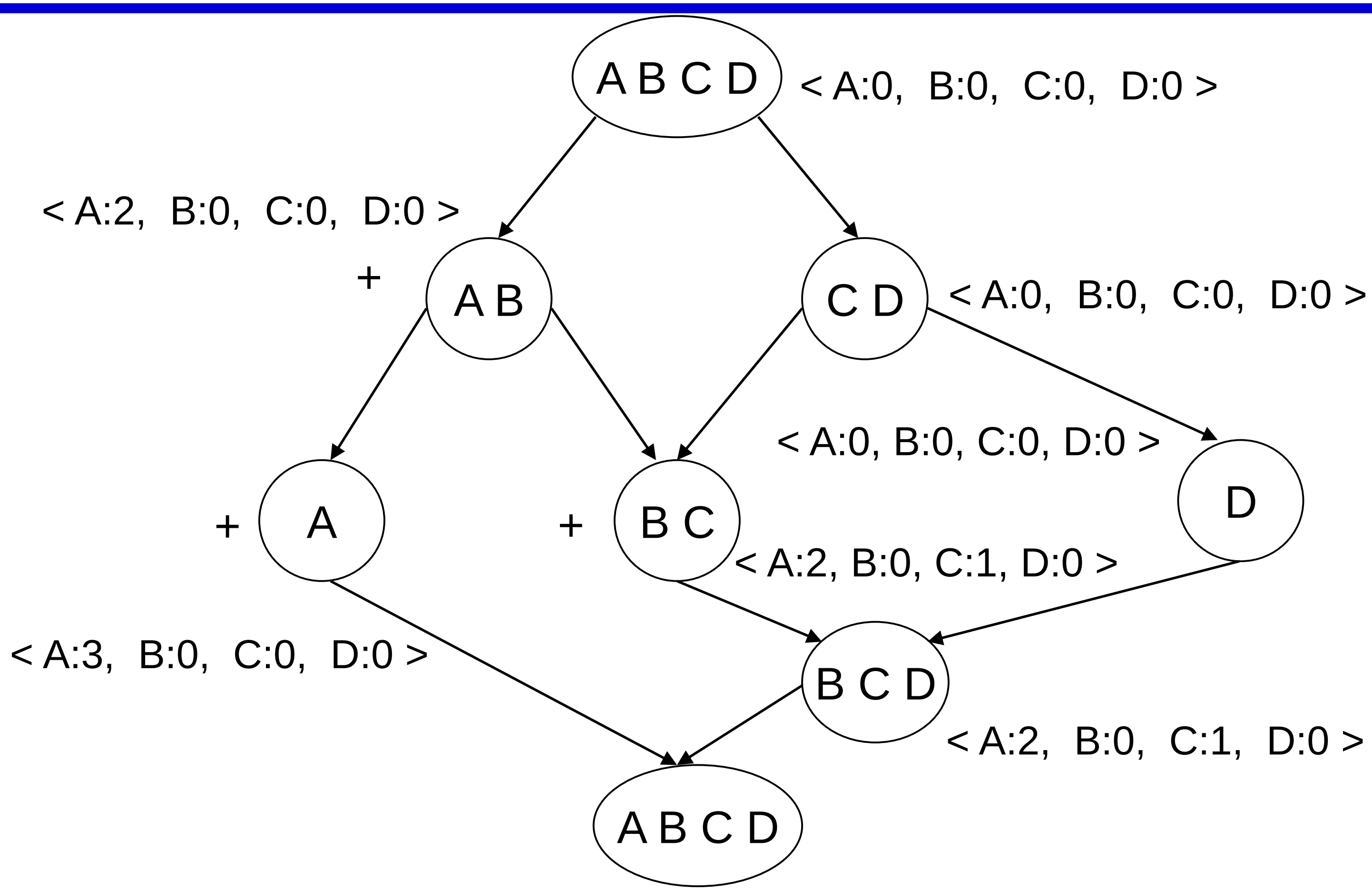
$VV_i = (S_i ; v_i)$

v_i update to file f at site S_i

Example of Partition and Merge



Create Conflict



CONFLICT!
After reconcilation at site B
< A:3, B:1, C:1, D:0 >

General resolution rules not possible.

External (irrevocable) actions prevent reconciliation, rollback, etc.

Resolution should be inexpensive.

System must address:

- detection of conflicts (when, how)
- meaning of a conflict (accesses)
- resolution of conflicts
  - automatic
  - user-assisted

Conclusions

Effective detection procedure

providing access without mutual

exclusion (consent).

Robust during partitions (no loss).

Occasional inconsistency tolerated for the sake of availability.

Reconciliation semantics…

Recognize dependence upon semantics.