DISTRIBUTED DATABASES

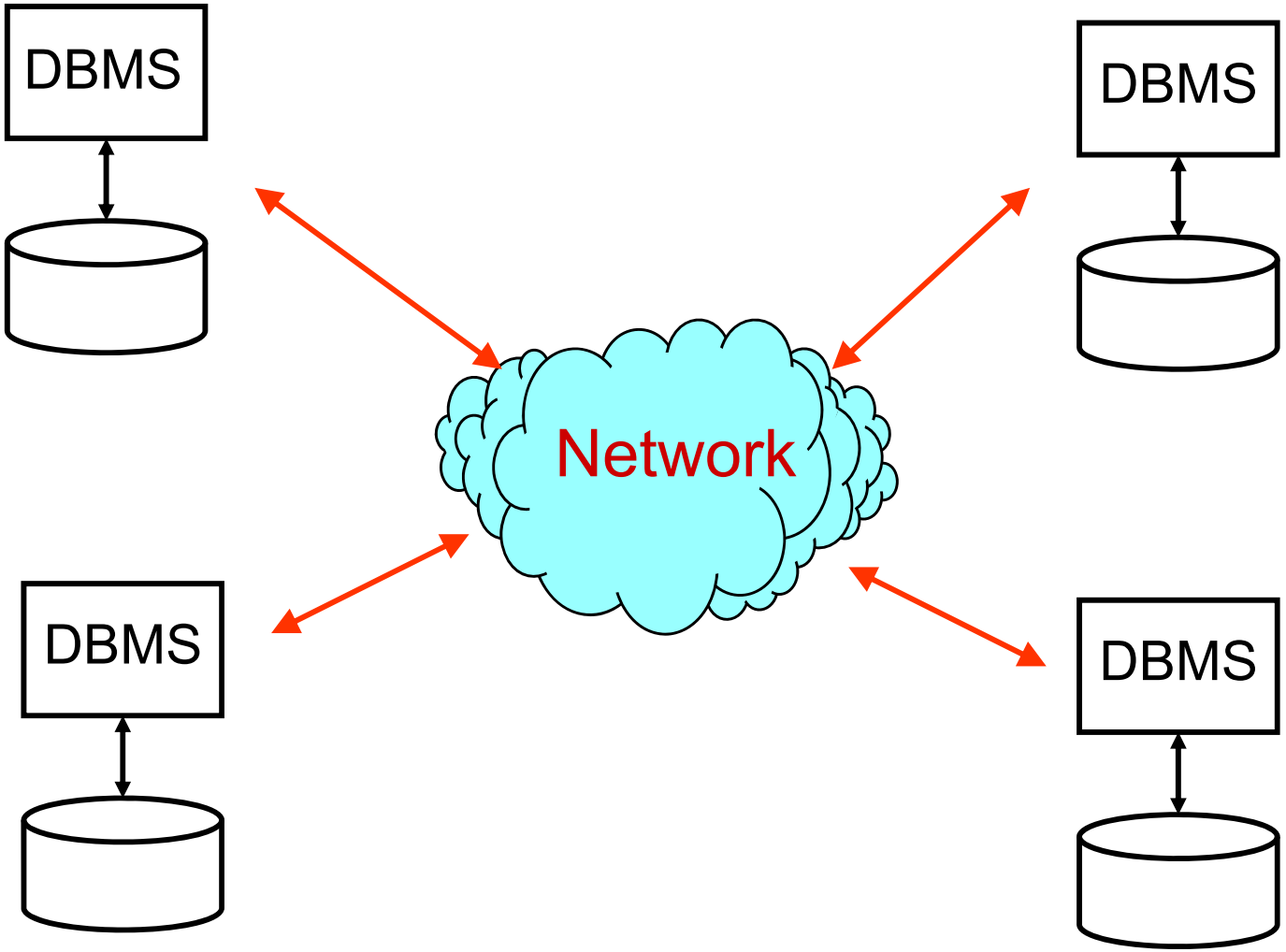
E0 261

Jayant Haritsa

Computer Science and Automation

Indian Institute of Science

Distributed Architecture



ADVANTAGES

- Availability
- Locality
- Reliability
- Parallelism
- Load Balancing
- Scalability
- Autonomy

DISADVANTAGES

- Significantly more complex to program and to maintain
- Communication costs need to be taken into account
- Communication failures need to be handled

DESIGN GOAL

- From user's perspective, the distributed system should appear exactly as if it were a centralized system !

- Implications on
  - Query Processing
  - Transaction Processing

Distributed Query Processing

Problem Complexity

- In centralized, performance metric was number of disk accesses
- In distributed, disk accesses and communication cost have to be taken into account
- Plan space explodes since location also becomes a parameter

Simple Case: R ▷◁ S

- R is at Site A, S is at Site B, user is at Site C
- Options:
  - Ship R and S to Site C, choose local strategy
  - Ship R to Site B, compute the join, ship result to Site C
  - Ship S to Site A, compute the join, ship result to Site C
  - Any others? Yes, “semi-join” strategy !

- Now, what about R ▷◁ S ▷◁ T, with each at different site ?

Semi-join Strategy

- Compute $\Pi_J(R)$ at Site A and ship to Site B
- Compute S $\triangleright\!\triangleleft$ $\Pi_J(R)$ at Site B and ship to Site A
- Compute R $\triangleright\!\triangleleft$ (S $\triangleright\!\triangleleft$ $\Pi_J(R)$) at Site A and ship to Site C
- This is called the semijoin approach, and the second operation is referred to as S $\triangleright\!<$ R
  - basic idea is "ship only those tuples that participate in the join"

Proof of Correctness

$R \triangleright\!\triangleleft (S \triangleright\!\triangleleft \Pi_J(R))$

$= (R \triangleright\!\triangleleft \Pi_J(R)) \triangleright\!\triangleleft S$

– because of associativity/commutativity properties

$= R \triangleright\!\triangleleft S$

Cost Computation Factors

- Disk Cost
- Communication Cost
- Meta-data shipping or re-creation Cost !!!

Cost Estimates

- Assume communication cost = $C_1 + C_2 d$ where d is the amount of data transmitted
- Assume disk IO cost to be linear in amount of data = $C_3 d$
- Assume stats are available on each relation
  - N_R = number of tuples in relation R
  - S_R = number of bytes in a tuple of relation R
  - $V_{x,R}$ = number of distinct values of attribute x in relation R

R ▷◁ S

- Relation R (x,y) at Site A
- Relation S (y,z) at Site B
- Stats:
  - $N_R = 128$, $N_S = 256$
  - $V_{y,R} = 32$, $V_{y,S} = 128$
  - $S_R = 10$, $S_S = 10$
- Results required at Site A

Strategy 1

- Ship S to Site A and compute join with R.
- Cost:
  - Retrieve S from disk: $C_3 (N_S S_s) = 2560\ C_3$
  - Ship S to Site A: $C_1 + C_2 (N_S S_S) = C_1 + 2560\ C_2$
  - Store S at Site A: $C_3 (N_S S_s) = 2560\ C_3$
  - nested-loop tuple join:

    $C_3 (N_R S_R + N_R (N_S S_S)) = 328960\ C_3$
    - (note that smaller relation, R, is outer)

Strategy 2

- Ship R to Site B, compute join with S, ship results back to Site A
- Cost:
  - Retrieve R from disk: $C_3 (N_R S_R) = 1280\ C_3$
  - Ship R to Site B: $C_1 + C_2 (N_R S_R) = C_1 + 1280\ C_2$
  - Store R at Site B: $C_3 (N_R S_R) = 1280\ C_3$
  - Join: $C_3 (N_R S_R + N_R N_S S_S) = 328960\ C_3$
  - Ship result to site A: need to compute result size
    - Number of tuples = $N_R N_S / \max (V_{y,R}, V_{y,S})$
      $= 128 * 256 / \max (32, 128) = 256$
    - Size of tuple (max) = $S_R + S_S$ – join attribute = 16 (if join on int)
    - Total size = 256 * 16 = 4096
    - $C_1 + 4096\ C_2$

Strategy 3: Semijoin - S $\triangleright\!\!<$ R

- Project Cost of N = $\Pi_y(R)$ = C_3 (N_R S_R) = 1280 C_3
- Ship N to Site B
  - C_1 + C_2 ($N_N S_N$) = C_1 + C_2 ($V_{y,R} S_N$) = C_1 + 32*4 C_2
- Store N at Site B = C_3 ($N_N S_N$) = 128 C_3
- Compute (semijoin) M = S $\triangleright\!\!\triangleleft$ N
  - C_3 ($N_N S_N$ + N_N ($N_S S_S$)) = C_3 (128 + 32 * 256 * 10) = 82048 C_3
- Ship M to Site A
  - number of tuples = $V_{y,R}$ N_S / max ($V_{y,R}$, $V_{y,S}$) = 32 * 256 / 128 = 64
  - C_1 + 64 * 10 C_2 = C_1 + 640 C_2
- Store M in site A = C_3 (N_M S_M) = 640 C_3
- Compute R $\triangleright\!\!\triangleleft$ M at Site A
  - C_3 (640 + 64 * 128 * 10) = 82560 C_3
  - (note, M is the outer relation since it is smaller)

Strategy 4: Semijoin - $R \triangleright\!\!<\!\!S$

- Project Cost of $N = \Pi_y(S) = C_3 (N_S S_S) = 2560\ C_3$
- Ship N to Site A
  - $C_1 + C_2 (N_N S_N) = C_1 + C_2 (V_{y,S} S_N) = C_1 + 512\ C_2$
- Store N at Site A = $C_3 (N_N S_N) = 512\ C_3$
- Compute (semijoin) $M = R \triangleright\!\!\triangleleft N$
  - $C_3 (N_N S_N + N_N (N_R S_R)) = C_3 (512 + 128 * 128 * 10) = 165120\ C_3$
- Ship M to Site B
  - number of tuples = $V_{y,S} N_R / \max (V_{y,R}, V_{y,S}) = 128 * 128 / 128 = 128$
  - $C_1 + 128 * 10\ C_2 = C_1 + 1280\ C_2$
- Store M in Site B = $C_3 (N_M S_M) = 1280\ C_3$

Strategy 4: Semijoin - R ▷<S (contd)

- Compute S ▷◁ M at Site B
  - C_3 (1280 + 128 * 256 * 10) = 328960 C_3
  - (note, M is the outer relation since it is smaller)
- Ship results to Site A
  - C_1 + C_2 (256 * 16) = C_1 + 4096 C_2

Cost Comparison

- Strategy 1: $C_1 + 2560\ C_2 + 334080\ C_3$
- Strategy 2: $2C_1 + 5376\ C_2 + 331520\ C_3$
- Strategy 3: $2C_1 + 768\ C_2 + 166656\ C_3$
- Strategy 4: $3C_1 + 5888\ C_2 + 498432\ C_3$

- Advantages of Semijoin Strategy 3:
  - Less communication: reflected in lower C_2 term
  - Smaller relations: reflected in lower C_3 term

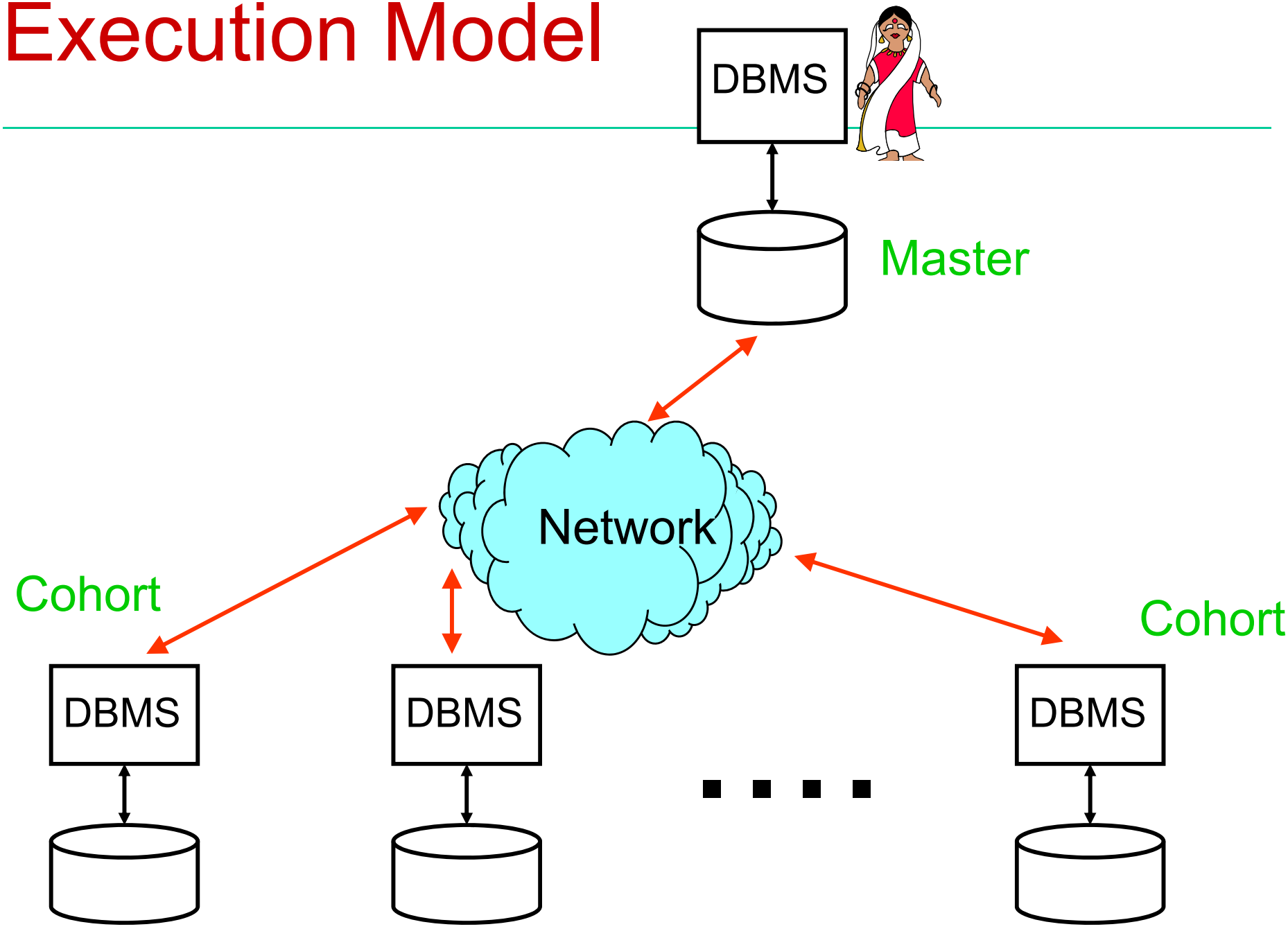
Centralized Semi-Join

- Strategy 1: $C_1 + 2560\ C_2 + 331520\ C_3$
- Strategy 2: $2C_1 + 5376\ C_2 + 330240\ C_3$
- Strategy 3: $2C_1 + 768\ C_2 + 166656\ C_3$
- Strategy 4: $3C_1 + 5888\ C_2 + 498432\ C_3$

- Semi-join can be useful even in a centralized system!
  - $R \triangleright\!\triangleleft S$: Cost = $328960\ C_3$
  - $R \triangleright\!\triangleleft (S \triangleright\!\triangleleft \Pi_J(R))$: Cost = $166656\ C_3$

Distributed Transaction Processing

Execution Model



Guaranteeing ACID

- Each cohort site can guarantee local ACID using the standard 2PL + WAL combination
- But, does "local ACID $\Rightarrow$ global ACID" ?
- No! For example, all sites may not implement the same decision (commit/abort)
- Therefore, need a commit protocol to ensure uniform decision across all sites participating in the distributed execution of the transaction

Two-Phase Commit Protocol

- 2PC guarantees global atomicity
- Note, 2PC different from 2PL !

Transaction Execution

- Master sends STARTWORK messages to the cohorts along with the associated work (this can be either sequential or in parallel)
- A cohort after completing its work sends a WORKDONE message to the master.
- The 2PC protocol is initiated by the master after the “data processing” part of the transaction is completed

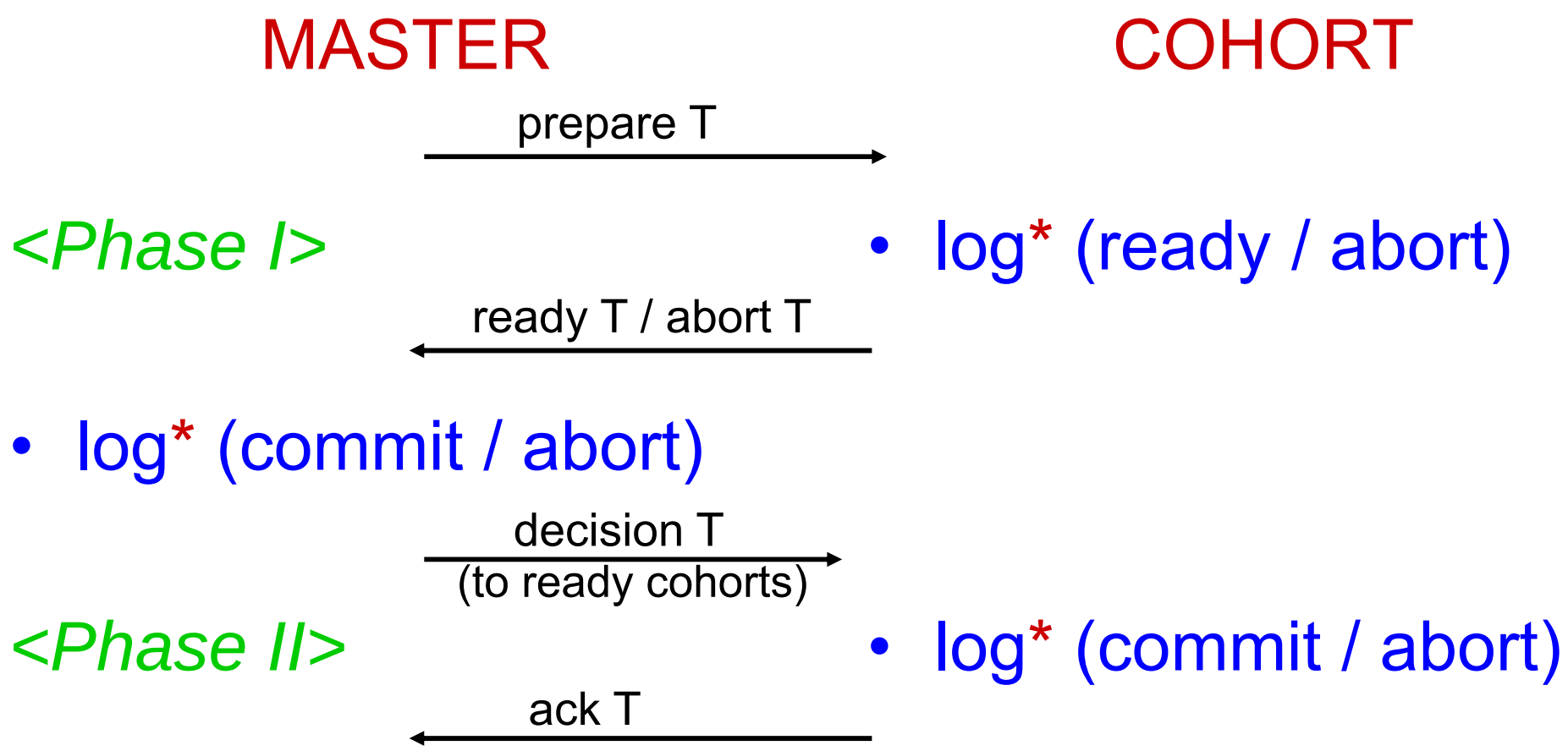
Phase I: Obtaining a decision

- Master asks all cohorts to *prepare* to commit transaction *T*.
  - sends **prepare** *T* messages to all sites at which *T* executed
- Upon receiving message, transaction manager at each site determines if it can commit the transaction
  - if not, add a record <**abort** *T*> to the log and send **abort** *T* message to master
  - if the transaction can be committed, then:
    - add the record <**ready** *T*> to the log
    - force *all log records* for *T* to stable storage
    - send **ready** *T* message to master

Phase II: Recording the Decision

- *T* can be committed if master received a **ready** *T* message from all cohorts; otherwise *T* must be aborted.
- Master adds a decision record, **<commit** *T*> or **<abort** *T*>, to the log and forces record onto stable storage.
- Master sends a message to each cohort informing it of the decision (commit or abort)
- The cohorts take appropriate action locally by writing the log records and send an <ack T> message to master.
- After receiving all ACK messages, the master writes an <end T> log record and then "forgets" the transaction.

2PC Protocol



TYPES OF FAILURES

- Cohort Site Failure
- Master Site Failure
- Network Partition

1. Handling Cohort Failure

When the site recovers, it examines its log to determine the fate of transactions active at the time of the failure.

- Log contains <**commit** *T*> record: site executes **redo** (*T*)
- Log contains <**abort** *T*> record: site executes **undo** (*T*)
- Log contains <**ready** *T*> record: site must consult the master to determine the fate of *T*
  - If *T* committed, **redo** (*T*)
  - If *T* aborted, **undo** (*T*)
- Log contains no control records for *T* :
  site executes **undo** (*T*)

2. Handling Master Failure

Cohort does not hear from master within a timeout period:

- If log does not contain a <**ready** *T*> record:

  abort *T*.

- If log contains <**ready** *T*> record:

  WAIT until master recovers!

- Could ask sibling cohorts: If any of them have an <**abort** *T*> or <**commit** *T*> log record, implement the same decision.

Blocking Problem

- What if *all* cohorts are in <**ready** *T*> state ?
  - All cohorts must wait for the master to recover in order to find out the final decision (called the *Blocking* problem)
    - Particularly problematic since cohorts block while holding locks, resulting in entire database processing transitively coming to a halt due to a convoy phenomenon.

3.Handling Network Partition Failure

- If master and all cohorts remain in one partition, the failure has no effect on the commit protocol.
- If master and cohorts are spread across partitions:
  - Sites not in the partition containing the master think master has failed, and execute the protocol to deal with master failure.
    - No harm results, but sites may still have to wait for decision from master.
  - Master and the sites in its partition think that the sites in the other partition have failed, and follow the usual commit protocol.
    - Again, no harm results

Three Phase Commit

- Designed to avoid Blocking
- Assumptions:
  - No network partitioning
  - At most K sites (cohorts as well as master) can fail simultaneously

Concurrency Control

Concurrency Control

- If all sites individually follow basic 2PL, is there any problem ?
- Yes, could have different serial orders at different sites!

Example Scenario

- Site A
  - R_{1A} (P)
  - W_{1A} (P)
  - R_{2A} (P)
  - W_{2A} (P)
- $T_1 \rightarrow T_2$

- Site B
  - R_{2B} (Q)
  - W_{2B} (Q)
  - R_{1B} (Q)
  - W_{1B} (Q)
- $T_2 \rightarrow T_1$

Problem will not occur with Strict 2PL !

Concurrency Control (contd)

- So, simple solution is to use strict 2PL at all the sites hosting the distributed DBMS.
- Note that above is assuming single copy of all data items, i.e. no replicas.
- If there are replicas, numerous techniques for handling concurrency control ("read any, write all", "primary copy", "majority vote", ...)
  - Full book on this topic:
    Concurrency Control and Recovery in Database Systems
    P. Bernstein, V. Hadzilacos, N. Goodman
    Addison-Wesley, 1987

END DISTRIBUTED DATABASES

E0 261