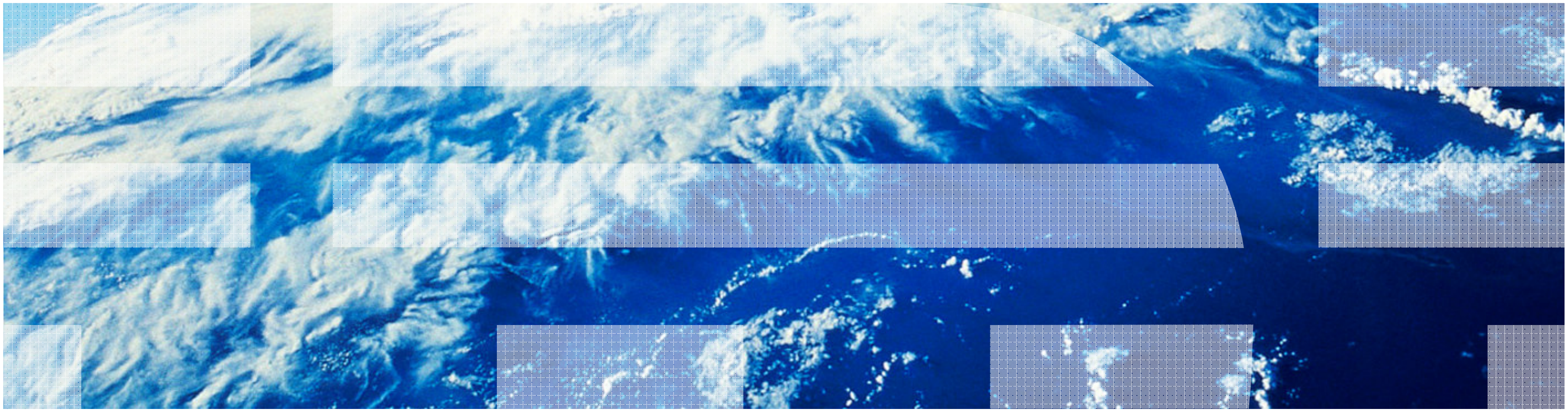
# Distributed and Fault-Tolerant Execution Framework for Transaction Processing

May 30, 2011

Toshio Suganuma, Akira Koseki, Kazuaki Ishizaki, Yohei Ueda,
Ken Mizuno, Daniel Silva[*], Hideaki Komatsu, Toshio Nakatani



IBM Research – Tokyo, [*]Amazon.com, Inc

# Motivation

- Transaction-volume explosions are increasingly common in many commercial businesses
  - Online shopping, online auction services
  - Algorithmic trading
  - Banking services
  - more…

- It is difficult (if not impossible) to create systems that satisfy transaction, scalable performance, and high availability

- Can we improve performance without significant loss of availability?

# Contributions

Study of performance-availability trade-off in a distributed cluster environment by proposing a new replication protocol

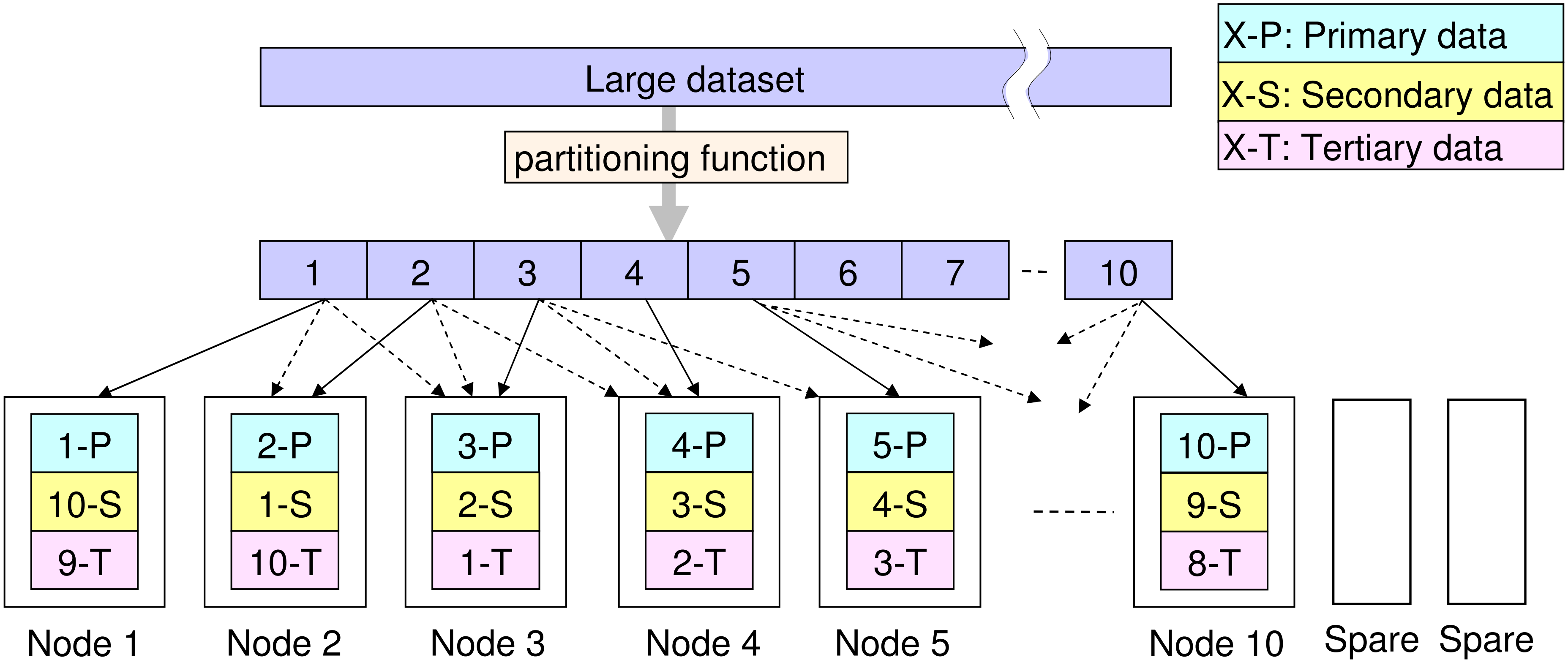
- Our replication protocol
  - Has a feature of continuous adjustment between performance and availability
  - Keeps global data consistency at transaction boundaries
  - Enables scalable performance with a slight compromise of availability

# Agenda

- Motivation and Contributions
- Replication Scheme
  - Data replication model
  - Existing replication strategies
  - Our approach
  - Replication protocol detail
  - Failure recovery process
  - Failover example
- Availability
- Experiment
- Summary

# Data Replication Model

- Data tables are partitioned and distributed over a cluster of nodes.
- Each partition is replicated on 3 different nodes (as Primary, Secondary, and Tertiary data), and each node serves for 3 different partitions



X-P: Primary data
X-S: Secondary data
X-T: Tertiary data

Large dataset

partitioning function

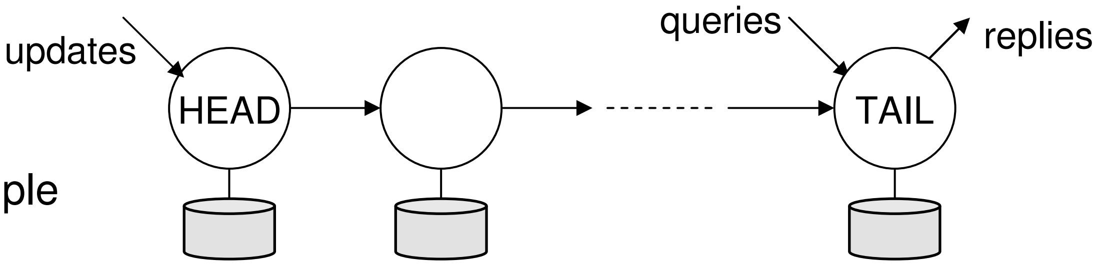
1 | 2 | 3 | 4 | 5 | 6 | 7 | -- | 10

| Node 1 | Node 2 | Node 3 | Node 4 | Node 5 | ------ | Node 10 | Spare | Spare |
|---|---|---|---|---|---|---|---|---|
| 1-P | 2-P | 3-P | 4-P | 5-P | | 10-P | | |
| 10-S | 1-S | 2-S | 3-S | 4-S | | 9-S | | |
| 9-T | 10-T | 1-T | 2-T | 3-T | | 8-T | | |

# Replication Scheme – Existing Approach

1. Synchronous replication
   - Primary waits for changes to be mirrored in Backup nodes
   - Allows failover without data loss
   - Limited performance: “Danger of replication…” paper [SIGMOD, 1996]
   - Example: Traditional RDB systems, e.g. DB2 parallel edition

2. Asynchronous replication
   - Primary proceeds without waiting acknowledgement from Backup
   - Risk data loss upon failover to Backup nodes
   - Better performance by passing synchronization delay to read transaction
   - Example: Chain replication [OSDI, 2004], Ganymed [Middleware, 2004]



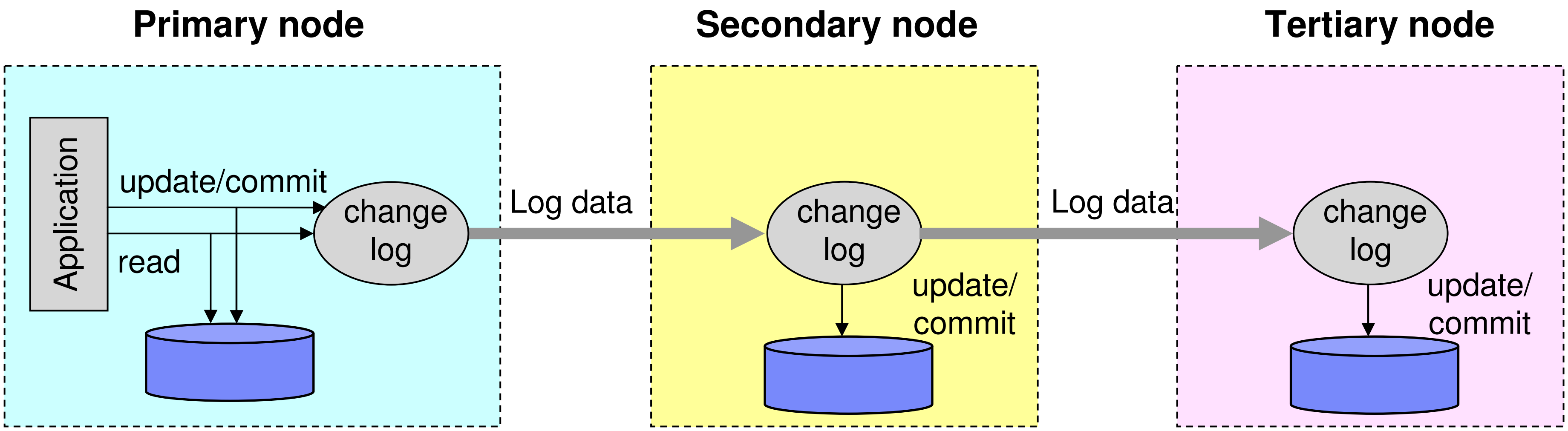
A Chain example

# Replication Scheme – Our Approach

We employ different replication policy for 2 backup nodes

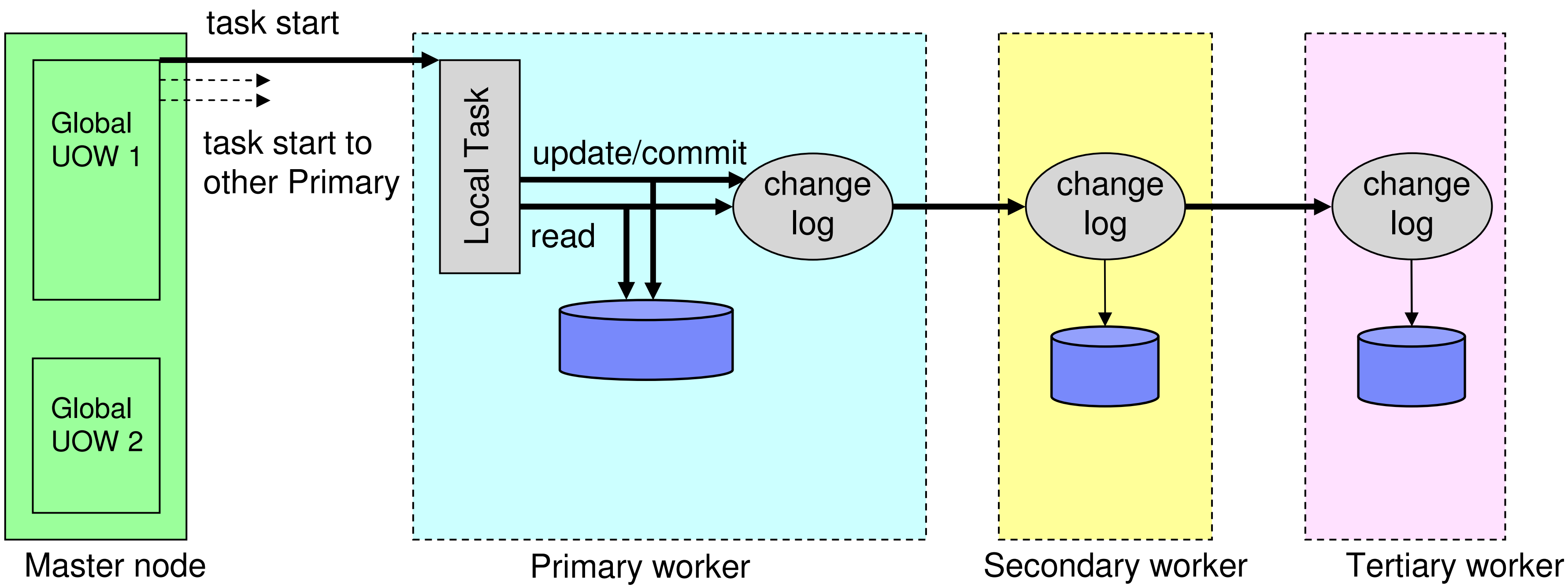
- Primary: Active computation node
- Secondary: Synchronous replication node
- Tertiary: Asynchronous replication node

➔This allows performance improvement with relaxed synchronization, while Tertiary can contribute for increasing availability



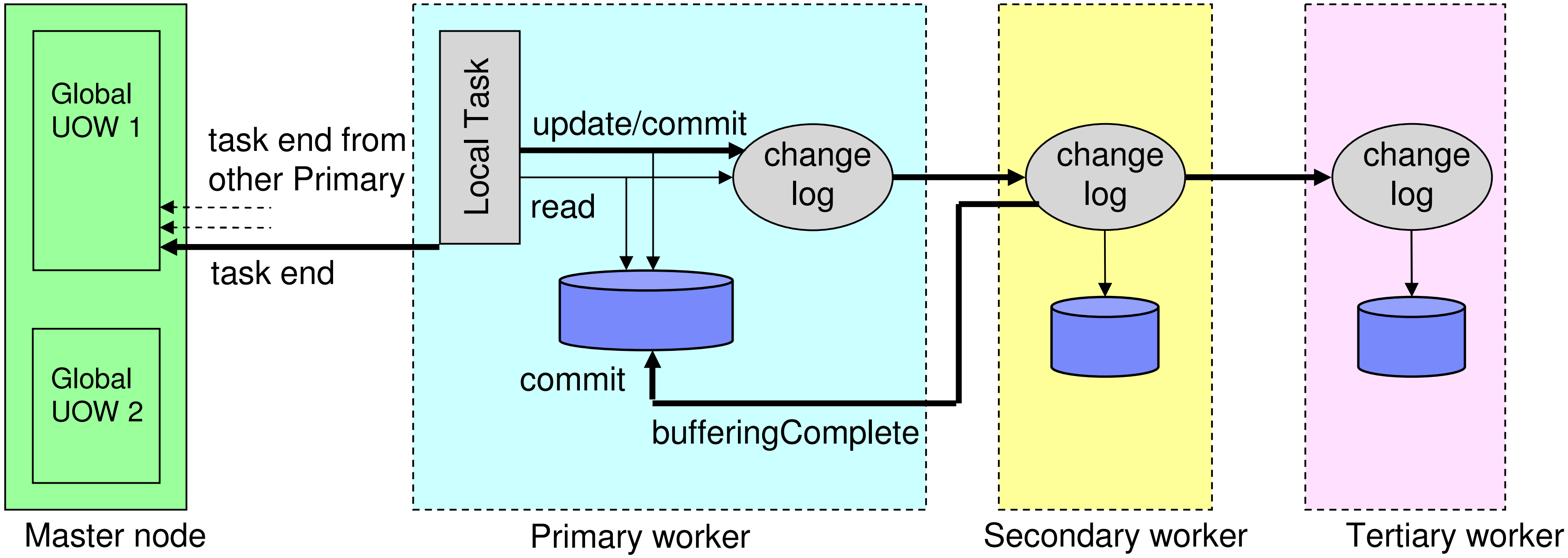
# Protocol Detail – 1. Execute a Task

- Master sends messages to all Primary to start their local tasks
- Primary accumulates all data updates from application to logs and sends them to Secondary
- Secondary passes the change logs to Tertiary



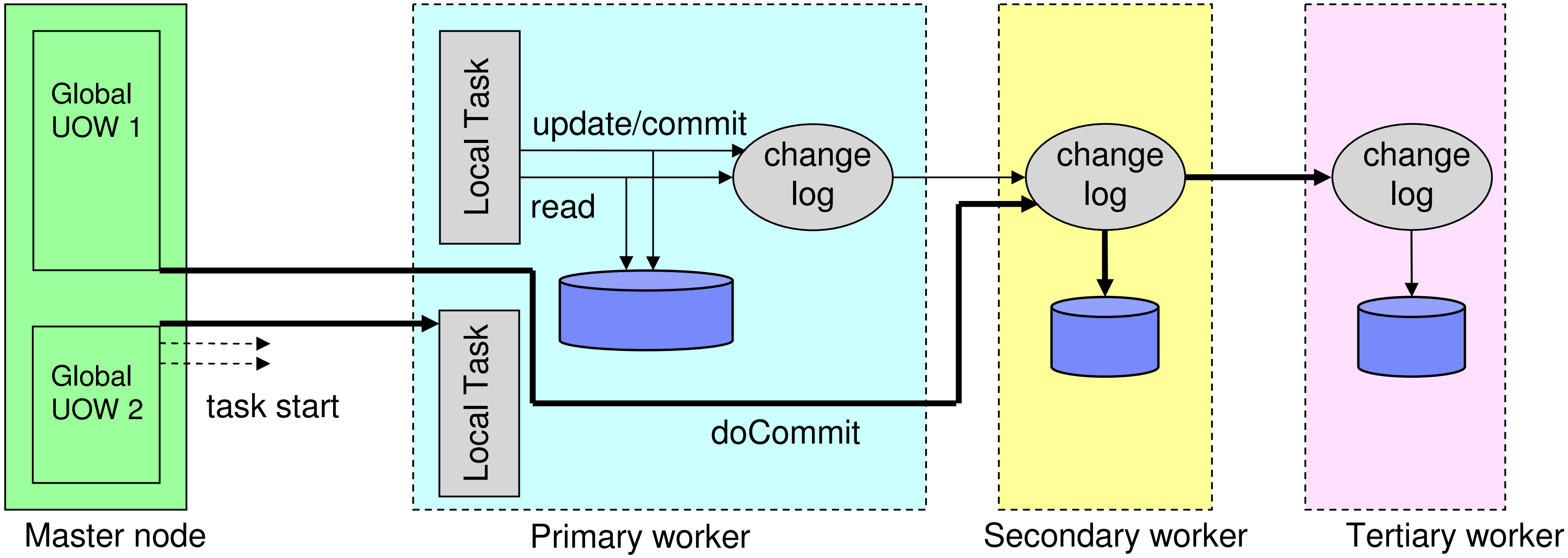
# Protocol Detail – 2. Commit in Primary

- Secondary notifies Primary when log buffering is completed
- Primary commits the local transaction when log buffering completion message arrived from Secondary
- Primary then sends the task end message to Master



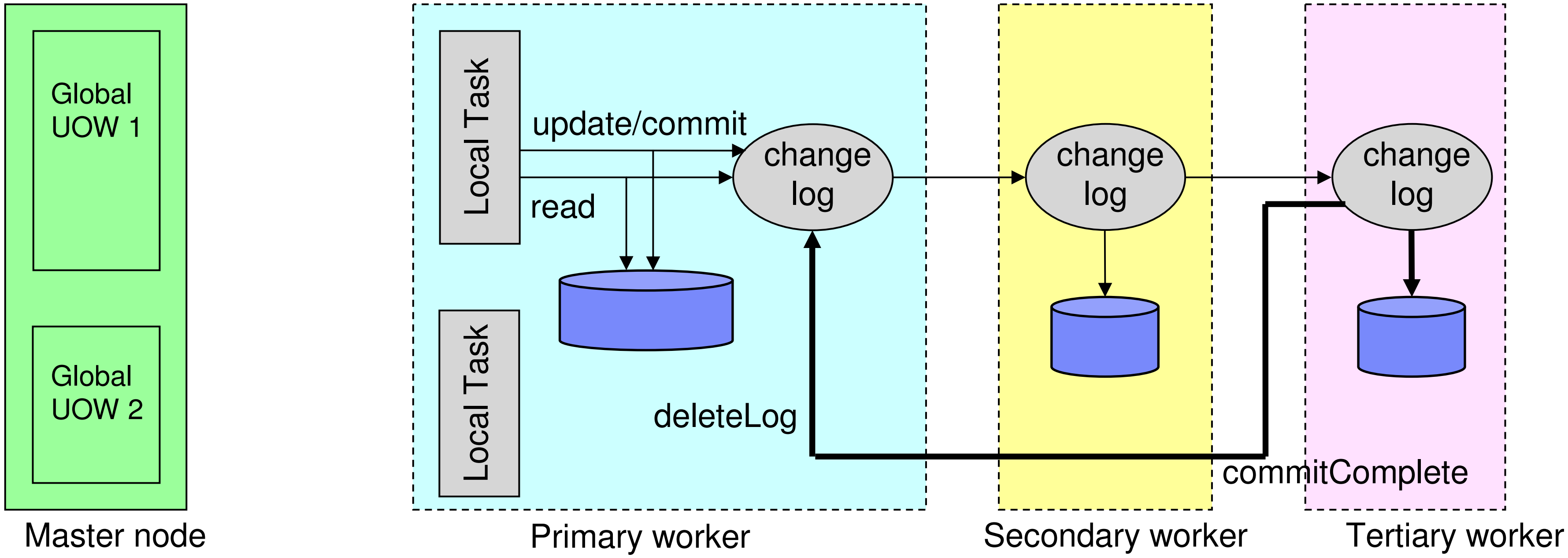
# Protocol Detail – 3. Commit in Secondary

- Master receives task end messages from all Primary
- Master sends all Secondary to commit
- Primary start the next local task after receiving the message from Master



# Protocol Detail – 4. Commit in Tertiary

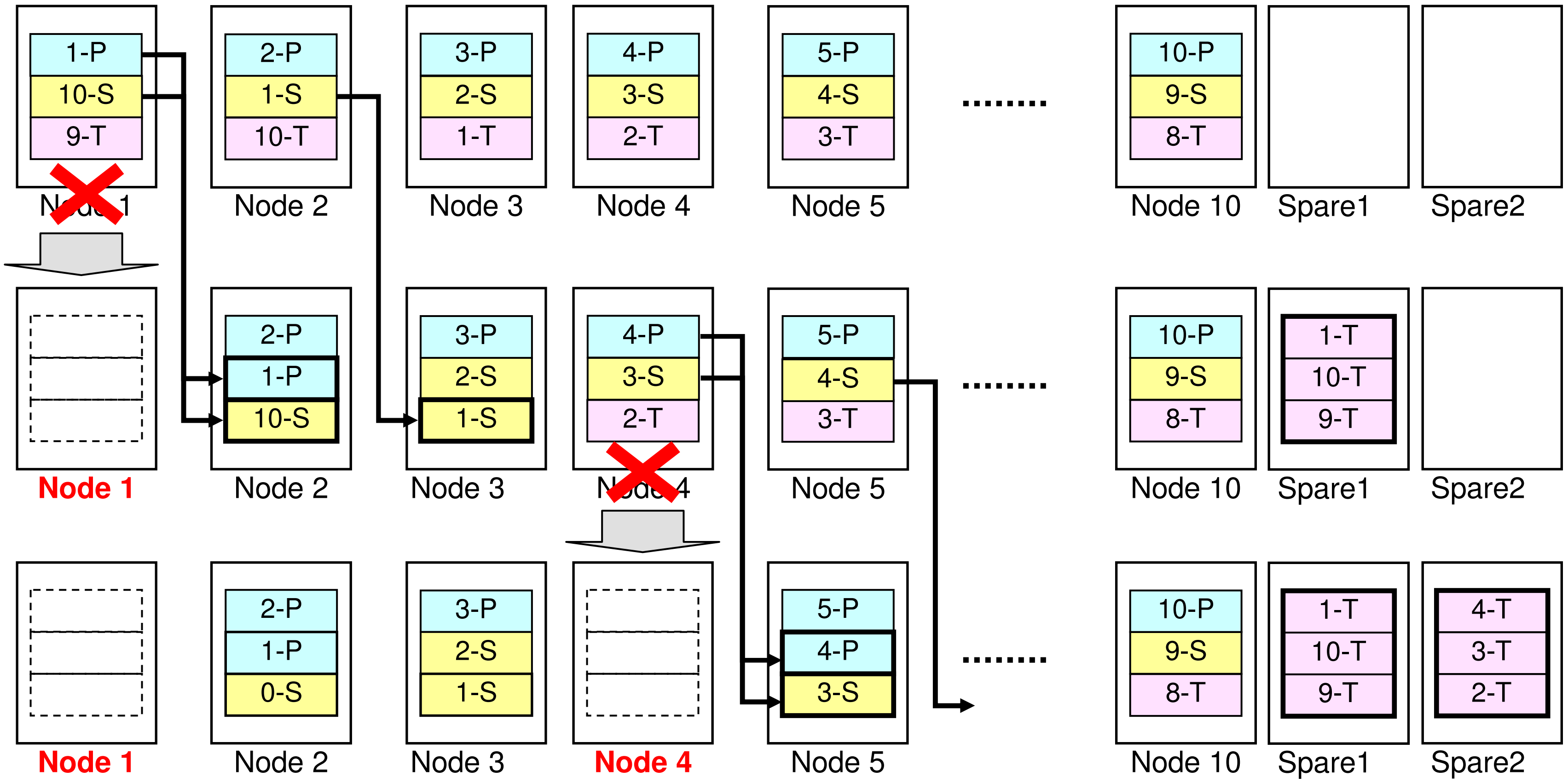
- Tertiary notifies Primary when the change logs are committed
- Primary deletes the corresponding logs



# Node Failover Example

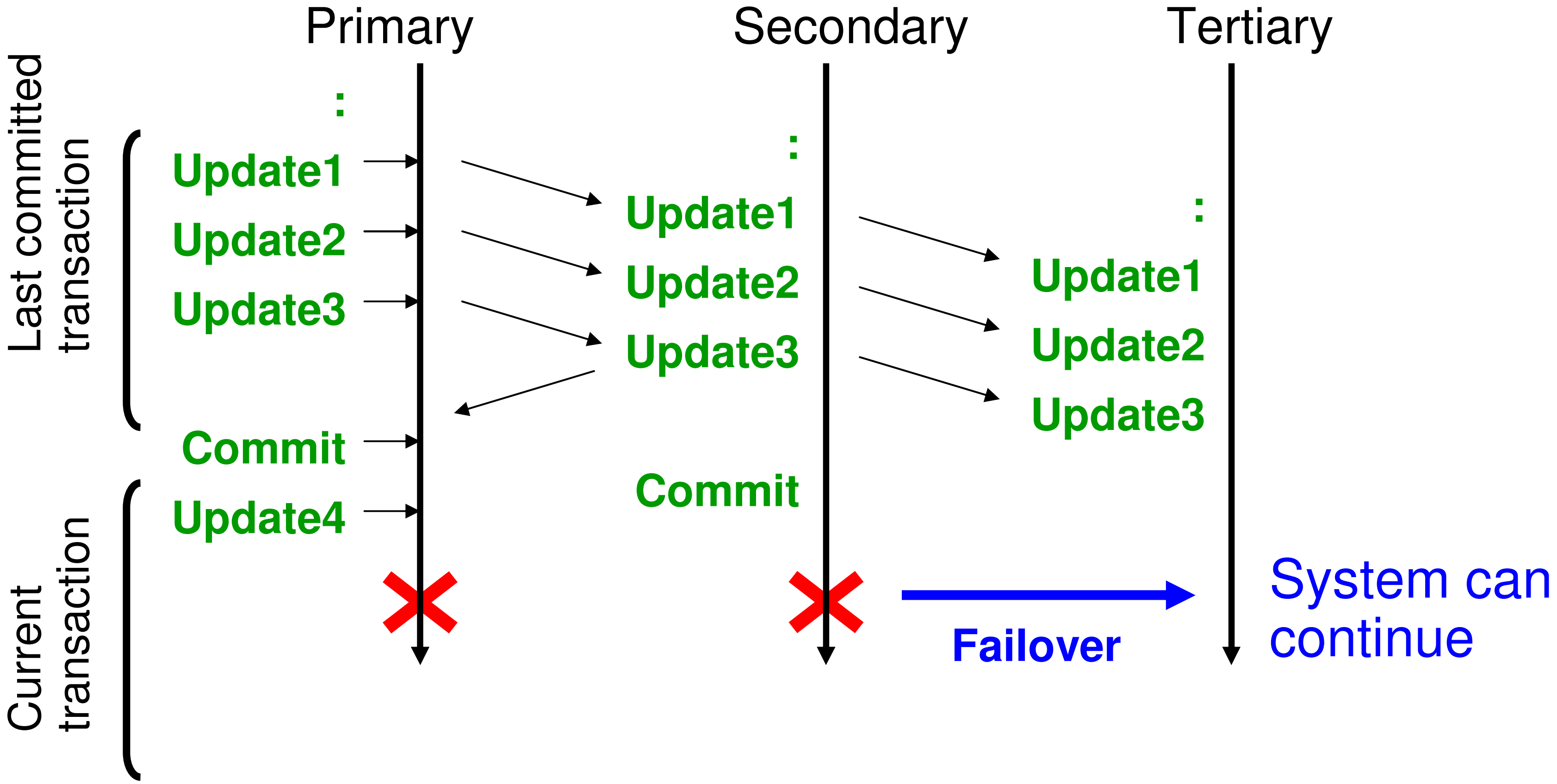
Primary
Secondary
Tertiary

- A spare node is activated upon failure of any single node
  - Secondary and Tertiary are promoted to Primary and Secondary
  - Spare node gets copies from the new Secondary, and acts as Tertiary



# Improved Availability by Tertiary

- Suppose both Primary and Secondary fail at the same time
- If Tertiary has the log records made in the last committed transaction, the system can continue without data loss



Primary

Secondary

Tertiary

Last commited transaction

Current transaction

:

Update1

Update2

Update3

Commit

Update4

:

Update1

Update2

Update3

Commit

:

Update1

Update2

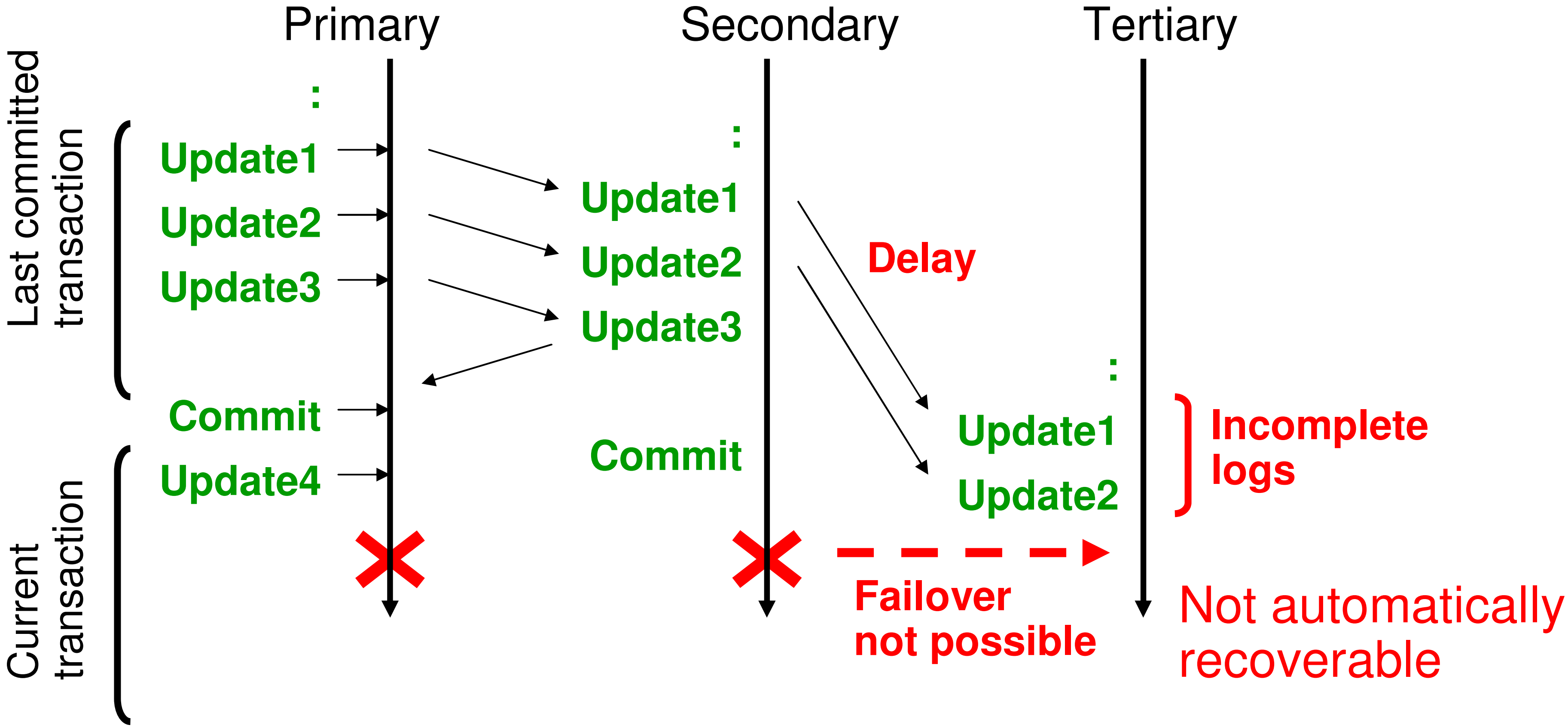
Update3

Failover

System can continue

# Data Loss Case

- If both Primary and Secondary fail at the same time, and
- If Tertiary has not received all the logs of the last committed transaction, some data is lost and the system is not automatically recoverable



Primary

Secondary

Tertiary

Last commited transaction

Current transaction

:

Update1

Update2

Update3

Commit

Update4

:

Update1

Update2

Update3

Commit

Delay

:

Update1

Update2

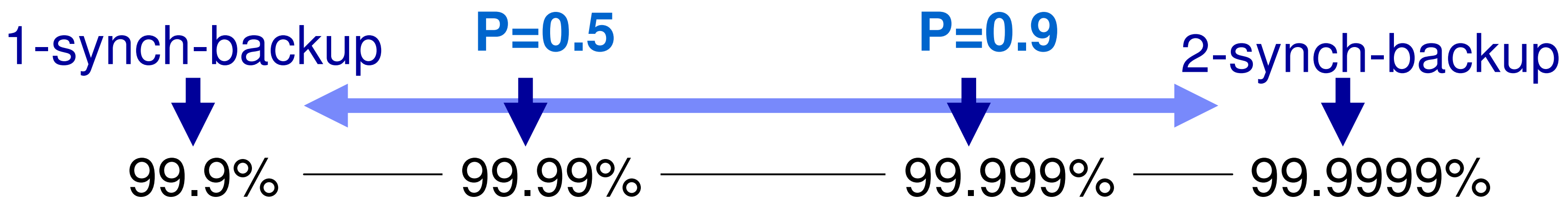
Incomplete logs

Failover not possible

Not automatically recoverable

# Availability with Our Replication Scheme

- Availability of our system is affected by the delay of transferring the log to Tertiary.
- The delay is significantly affected by data transfer efficiency from Secondary to Tertiary
  - Disk accesses due to insufficient memory can be a bottleneck
- By removing I/O bottlenecks on the nodes, we can minimize the delay and maximize **P**, the probability of availability of the log records of the last committed transaction.



1-synch-backup

**P=0.5**

**P=0.9**

2-synch-backup

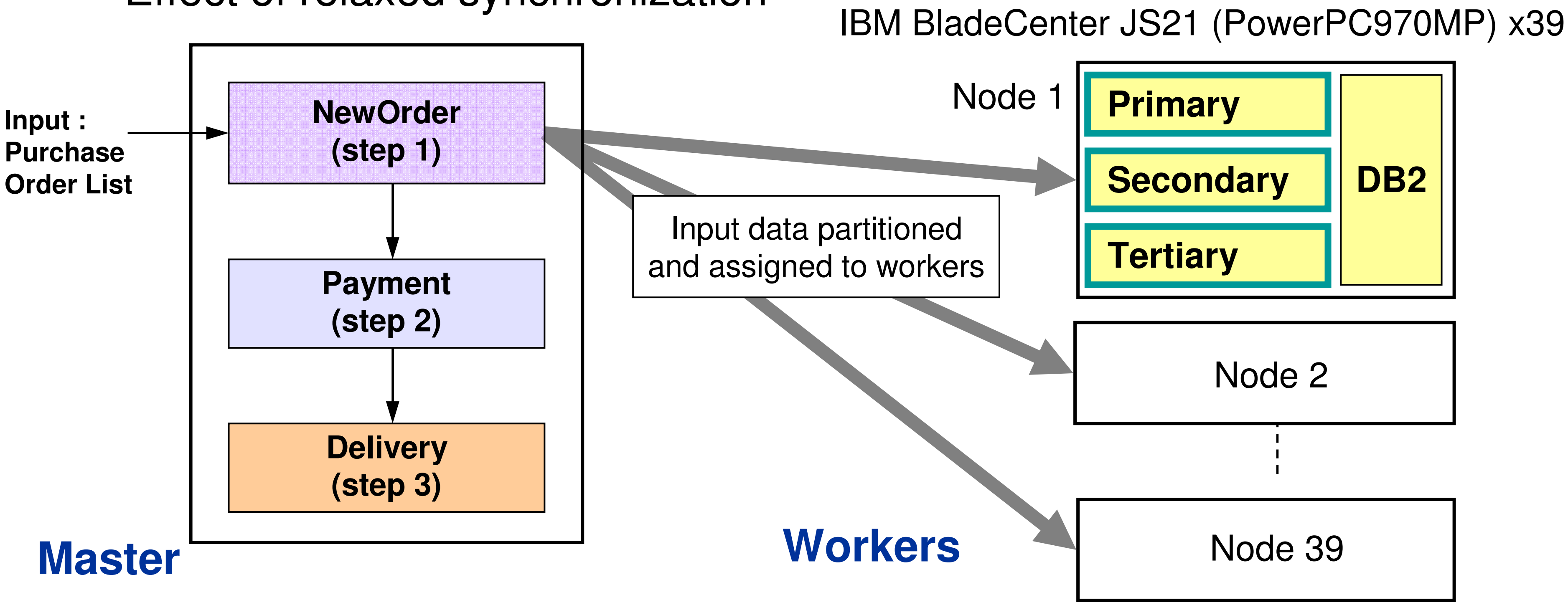
99.9% — 99.99% — 99.999% — 99.9999%

**Case:** A cluster of 1,000 nodes, each has 0.001 failure probability (corresponding 3-year (= 1,000-day) MTBF and 1-day MTTR

# Evaluation with TPC-C Workloads

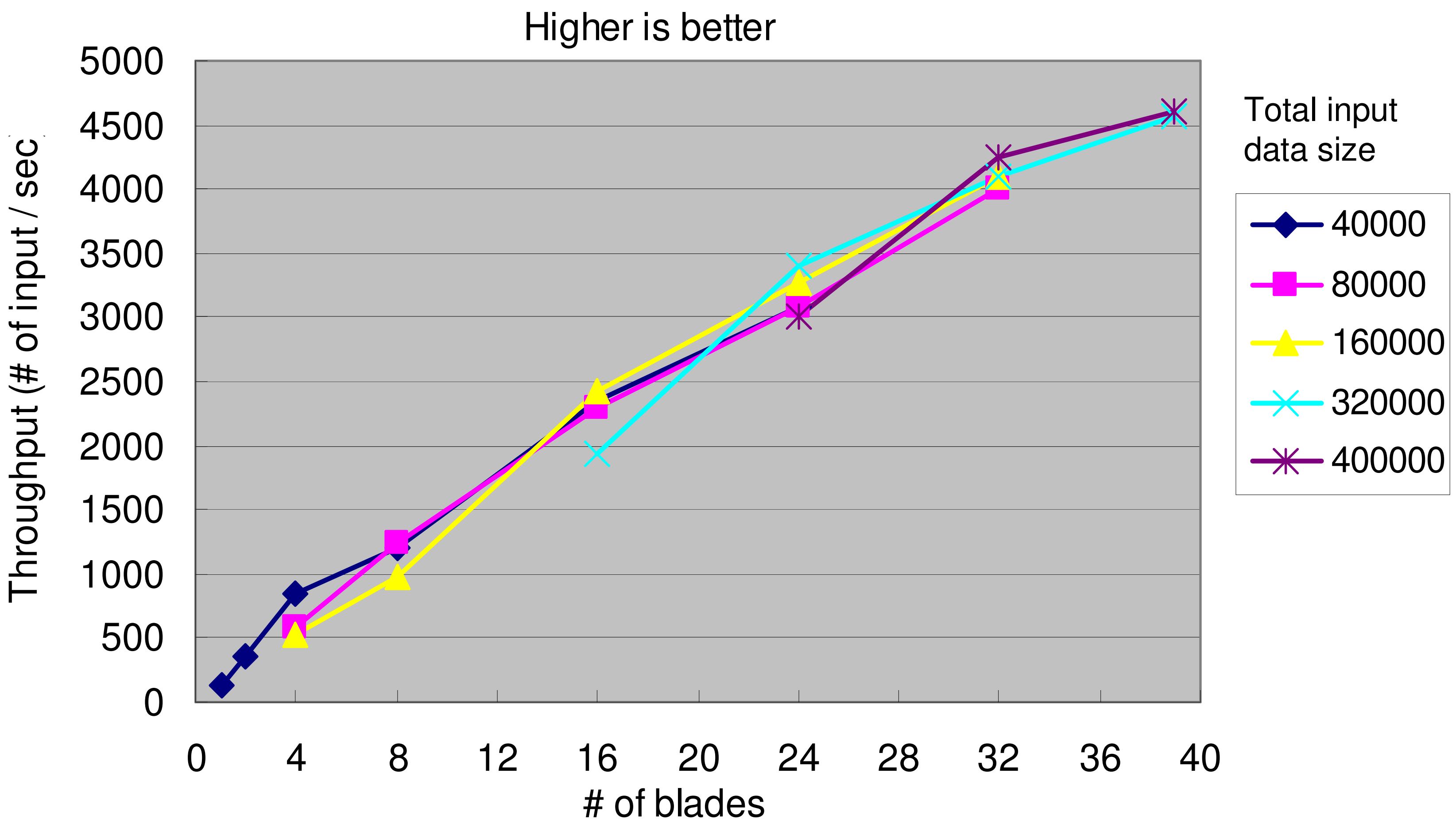
- We created a batch job by combining three different scenarios in TPC-C; NewOrder, Payment, and Delivery

- We evaluate our replication protocol from the following aspects:
  - Scaling efficiency (strong scaling and weak scaling)
  - Replication overhead (with and without replication)
  - Effect of relaxed synchronization



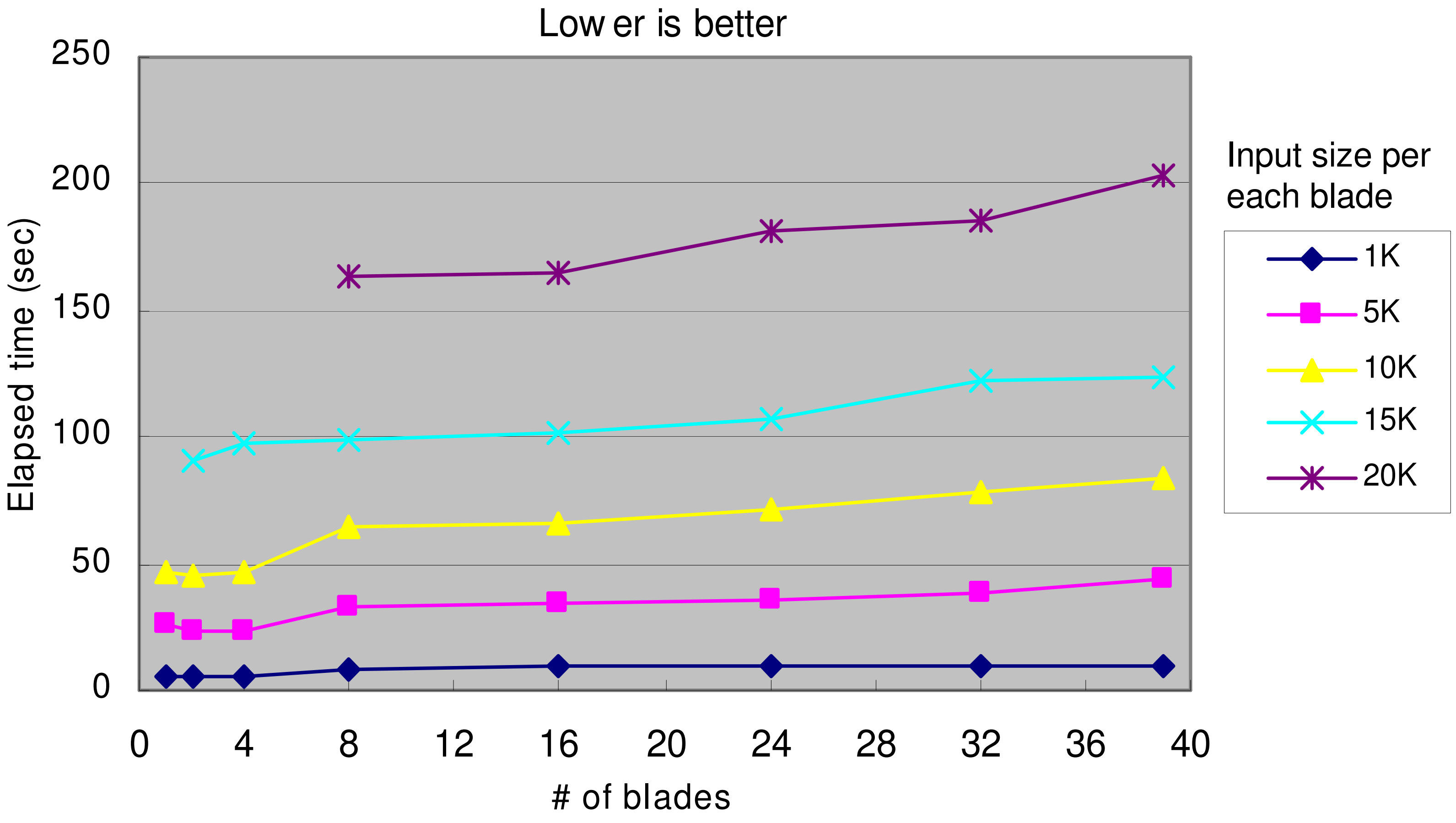
# Strong Scaling Efficiency

- The throughput is increased almost linearly as nodes are added.



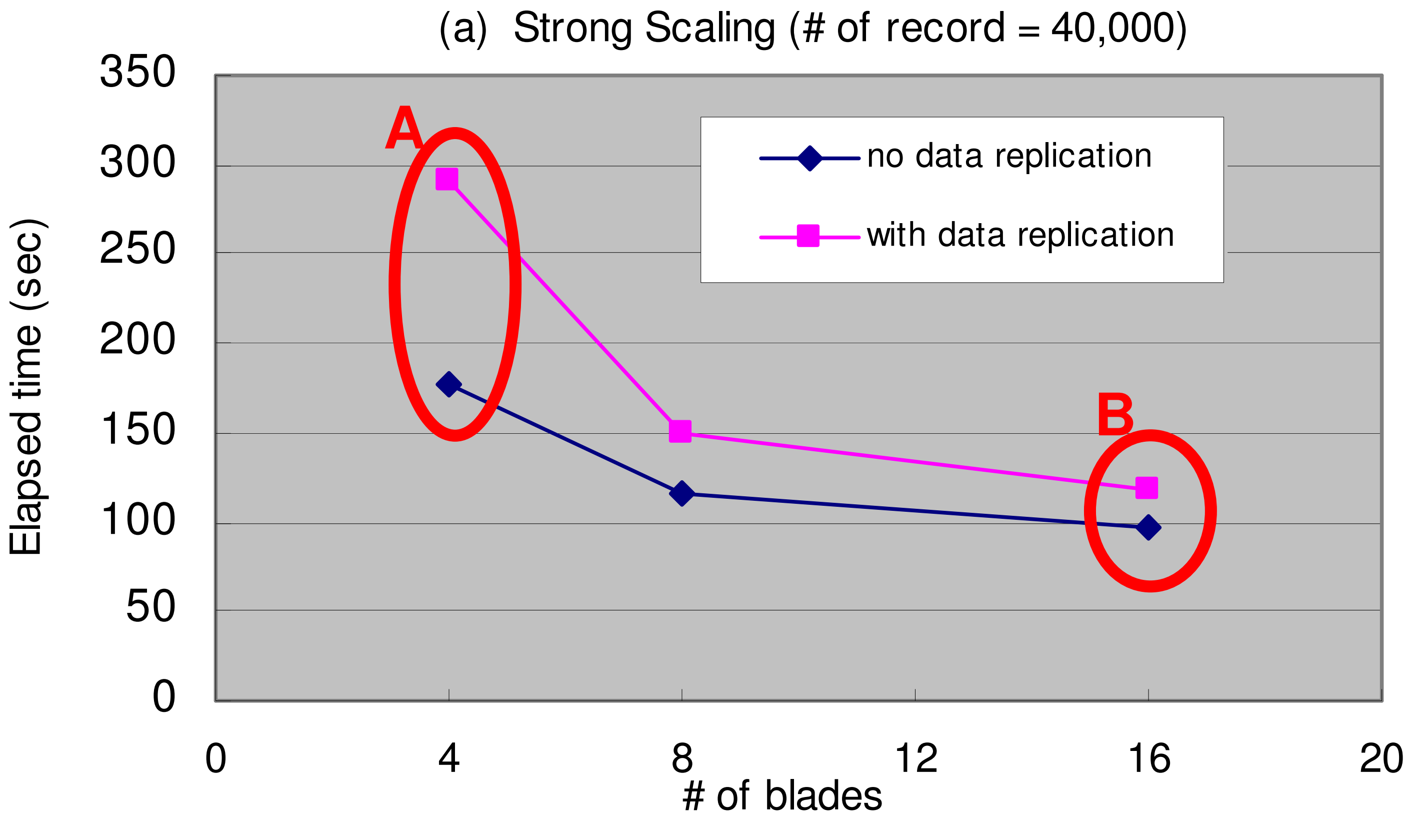
# Weak Scaling Efficiency

- The execution time is almost flat as nodes are added if sufficient memory is available for the node (e.g. buffer pools of DB).
  - Otherwise, the increase of disk accesses causes the delay of synchronization



# Replication Overhead

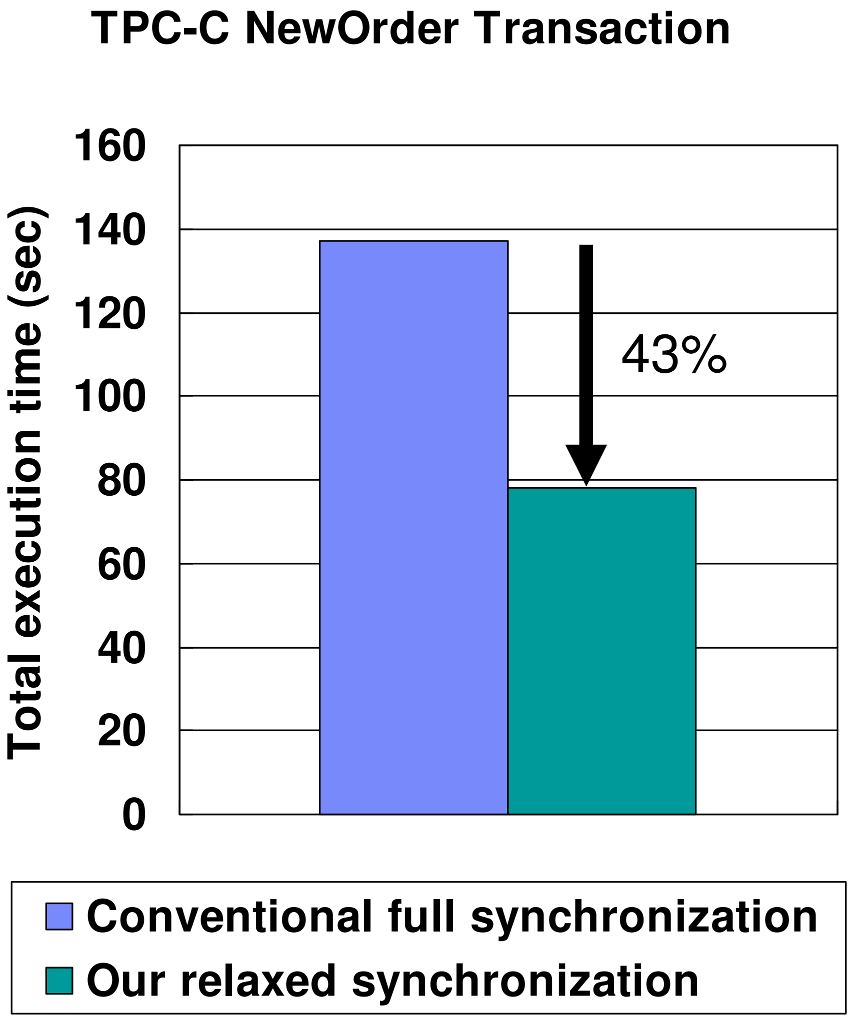
- The replication overhead varies with the input data size per blade
  - **A** → heavy disk accesses causes fairly high overhead
  - **B** → with sufficient memory resource, the overhead is 20%



# Data Synchronization Effect

- We compared the total execution time of TPC-C NewOrder transactions between conventional (full synchronization) model and our relaxed synchronization model

- The 43% reduction of the execution time is due to our approach of low synchronization overhead



# Summary

- We proposed a new replication protocol that combines two different replication policies
  - Synchronous replication for Secondary and asynchronous replication for Tertiary.

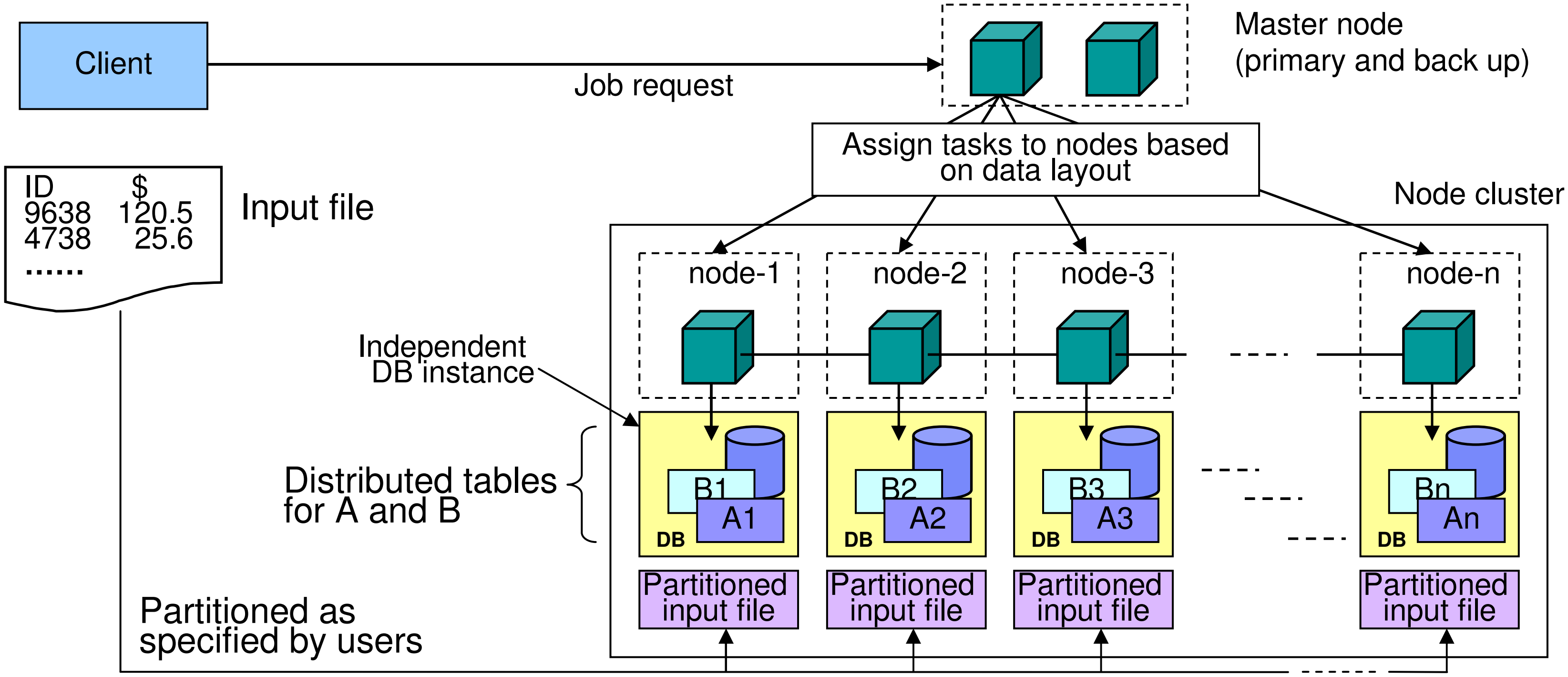
- Using our replication scheme:
  - We can achieve scalable performance
  - System tolerates up to 2 simultaneous node failure among triple redundant nodes most of the time
  - Overhead of data replication is 20% with sufficient memory

- We showed performance-availability trade-off that we can obtain performance improvements by slightly compromising availability
  - E.g. 99.9999% → 99.999%

# Backup

# System Overview [Ishizaki et al, SYSTOR 2010]

1. Data (both DB tables and input files) are partitioned and distributed over a cluster of nodes, as specified by users
2. Master partitions the job into tasks based on data layout, and assigns them to nodes based on owner-compute-rule
3. Each node executes a task (which only requires local data accesses)



# Replication Scheme in KVS

- Optimistic replication
  - Rely on eventual consistency model
  - Conflict resolution mechanism is necessary
  - Transaction cannot be supported (e.g., read-modify-write is not possible)
  - Superior in performance and availability
  - Example: Gossip protocol in Cassandra

# Availability Calculation

**Example**

- A cluster of 1,000 nodes, each has the probability of 0.001 failure
  - E.g. 3-year (= 1,000-day) MTBF (Mean Time Between Failure) and 1-day MTTR (Mean Time To Repair)

- Conventional full synchronization approach:
  - System becomes unavailable only when all nodes holding a copy of a particular partition fail at the same time

  ➔99.9999% availability for 2-backup-node replication

  ➔99.9% availability for 1-backup-node replication

- Our relaxed synchronization approach:
  - System availability depends on the probability of log availability in tertiary on a simultaneous failure of primary and secondary nodes

  ➔99.999% if we assume this probability of log-availability is 0.9

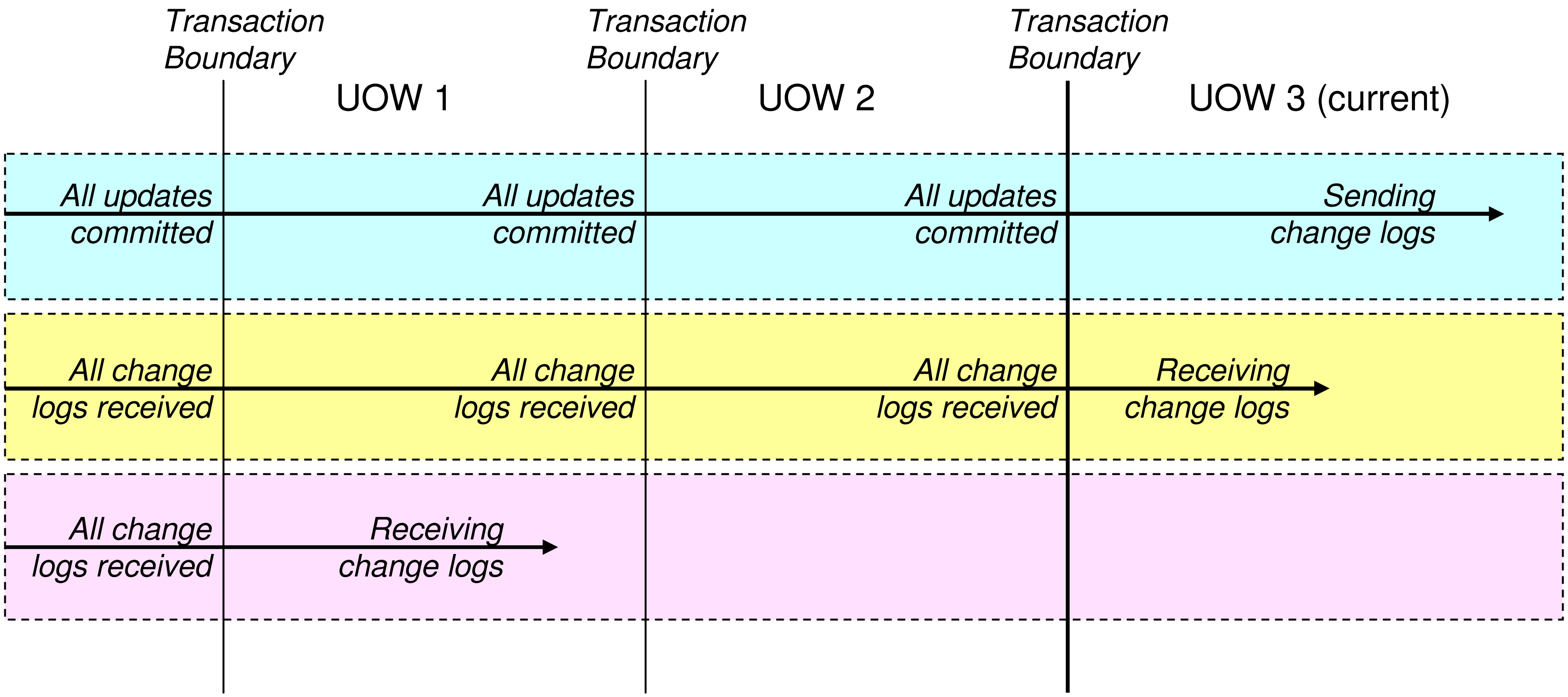
  ➔99.99% if we assume this probability of log-availability is 0.5

# What is "Sustainable State"?

- Primary has committed all updates in the last UOW and sent their logs to Secondary.
- Secondary has received the logs for the last UOW.
- Tertiary is alive and ready to receive logs

➔Our protocol proceeds by keeping the Sustainable State among all triplets

**Example**



*Transaction Boundary*

*Transaction Boundary*

*Transaction Boundary*

UOW 1

UOW 2

UOW 3 (current)

**Primary**

*All updates committed*

*All updates committed*

*All updates committed*

*Sending change logs*

**Secondary**

*All change logs received*

*All change logs received*

*All change logs received*

*Receiving change logs*

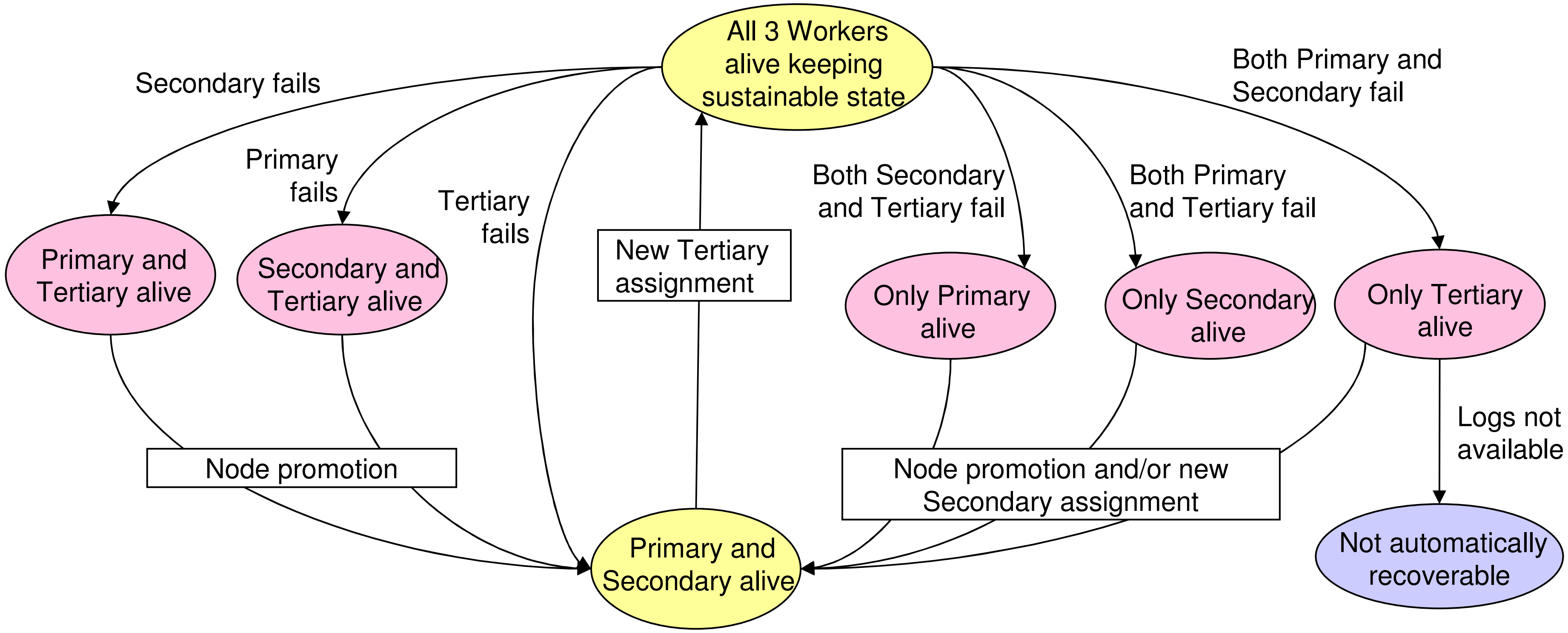
**Tertiary**

*All change logs received*

*Receiving change logs*

# Failure Recovery Process

1. Find a transaction recovery point and determine new Primary and Secondary
2. Select a node to join the triplet as new Tertiary
3. Have the new Secondary send a snapshot and logs to the new Tertiary
4. Resume application on new Primary



All 3 Workers alive keeping sustainable state

Secondary fails

Primary fails

Tertiary fails

Both Secondary and Tertiary fail

Both Primary and Tertiary fail

Both Primary and Secondary fail

Primary and Tertiary alive

Secondary and Tertiary alive

New Tertiary assignment

Only Primary alive

Only Secondary alive

Only Tertiary alive

Logs not available

Node promotion

Node promotion and/or new Secondary assignment

Primary and Secondary alive

Not automatically recoverable