# Chapter 26: Advanced Transaction Processing

## Database System Concepts, 6th Ed.

# Chapter 26: Advanced Transaction Processing

- 26.1 Transaction-Processing Monitors
- 26.2 Transactional Workflows
- 26.3 Transactions in E-Commerce
- 26.4 Main Memory Databases
- 26.5 Real-Time Transaction Systems
- 26.6 Long-Duration Transactions

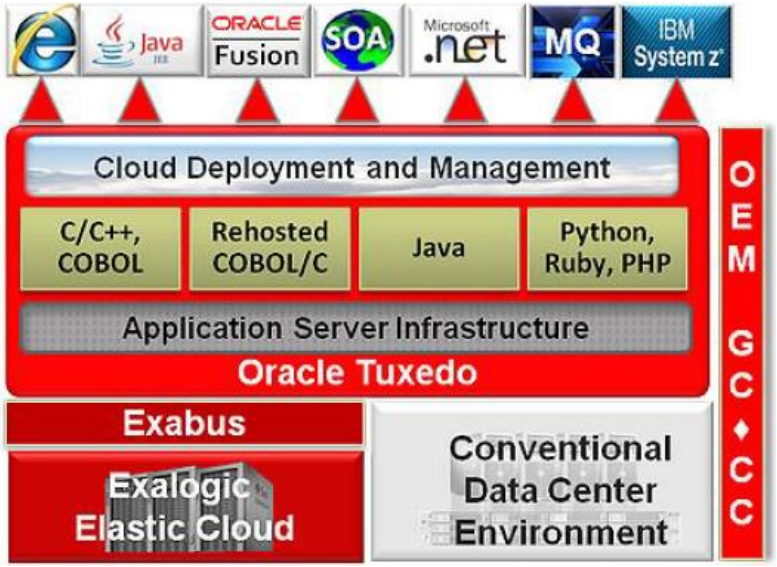
# Transaction Processing Monitors

- **TP monitors** initially developed as multithreaded servers to support large numbers of terminals from a single process.
- Provide infrastructure for building and administering complex transaction processing systems with a large number of clients and multiple servers.
- Provide services such as:
  - Presentation facilities to simplify creating user interfaces
  - Persistent queuing of client requests and server responses
  - Routing of client messages to servers
  - Coordination of two-phase commit when transactions access multiple servers.
- Some commercial TP monitors:
  - CICS from IBM
  - Pathway from Tandem
  - Top End from NCR
  - Encina from Transarc

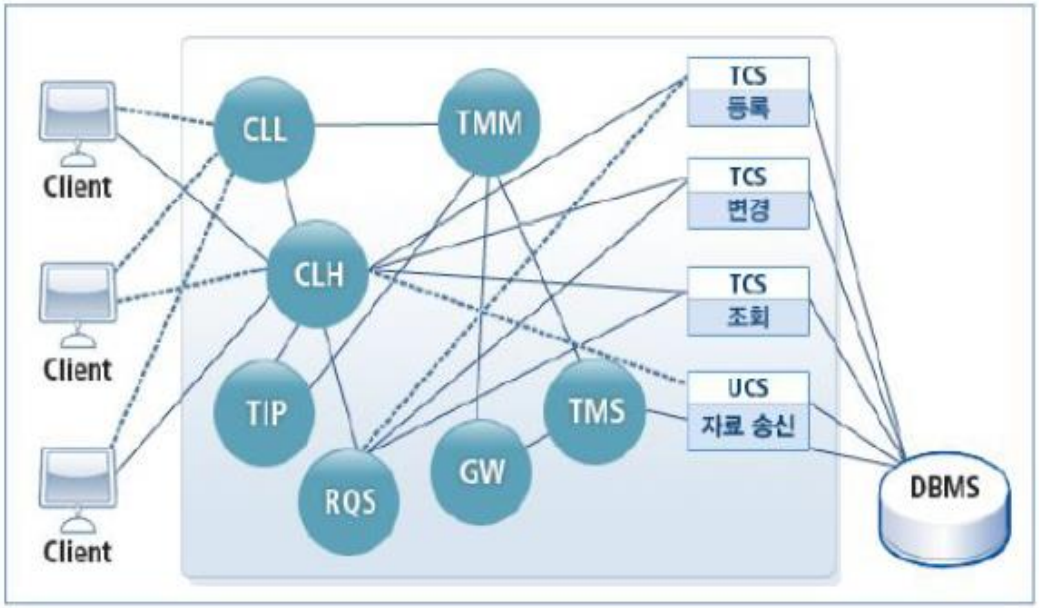
참고자료

# Commercial Products

- Oracle Tuxedo (First developed by AT&T in 1980s)

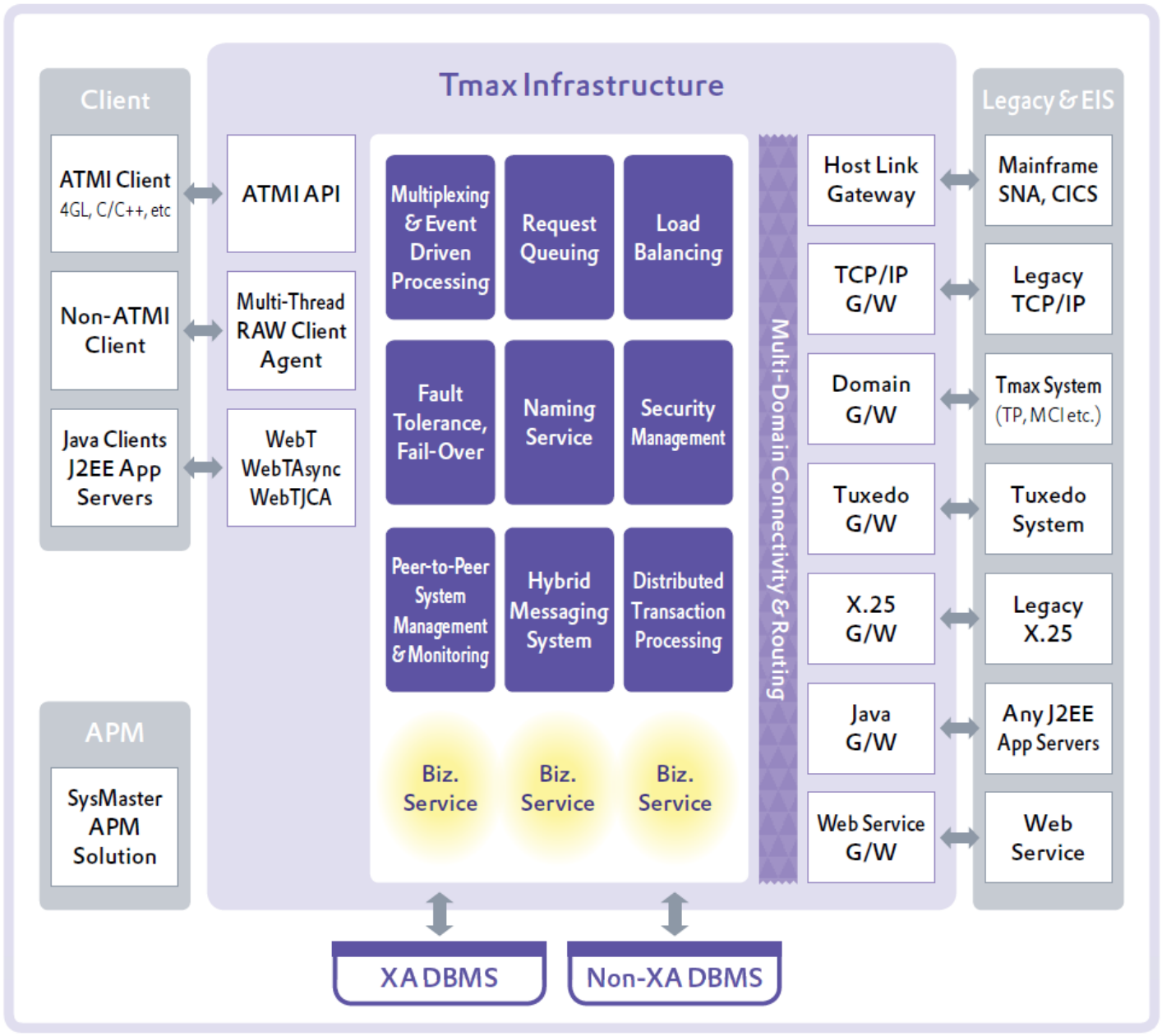



- Tmax (Developed by TmaxSoft)

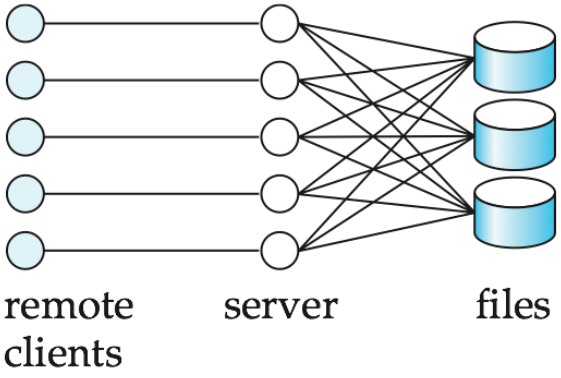



TMM: Tmax 시스템 운용 프로세스
CLL : Client Listener 프로세스
CLH: Client Handler 프로세스
RQS: 디스크 큐 관리 프로세스
TMS: DB 관리 및 분산 트랜잭션 처리 담당 프로세스

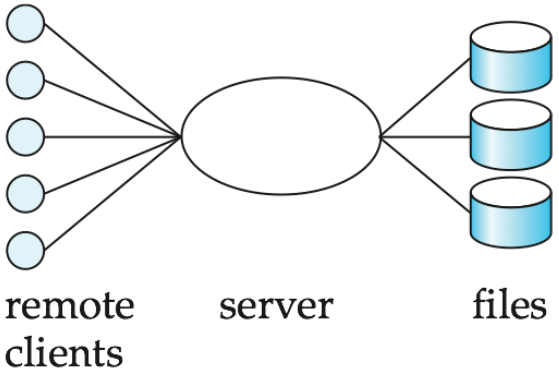
# Commercial Products



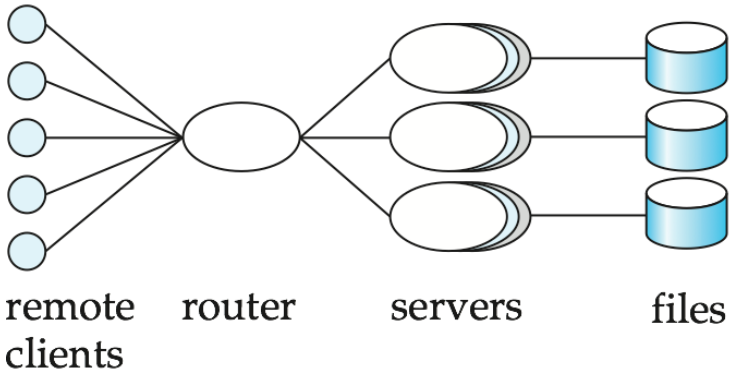
# TP Monitor Architectures [1/3]



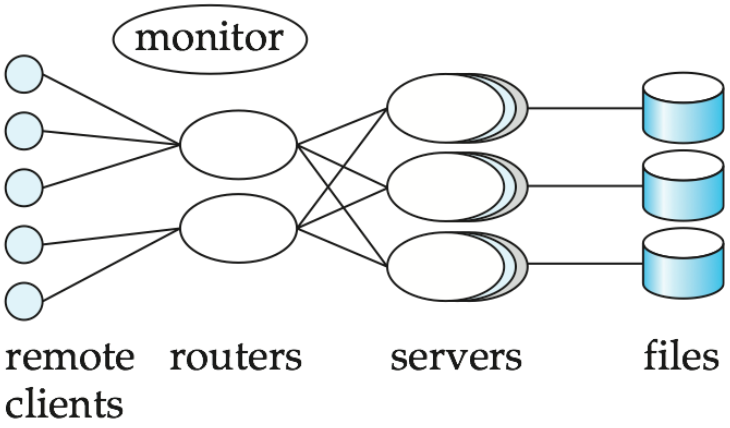

(a) Process-per-client model




(b) Single-server model




(c) Many-server, single-router model

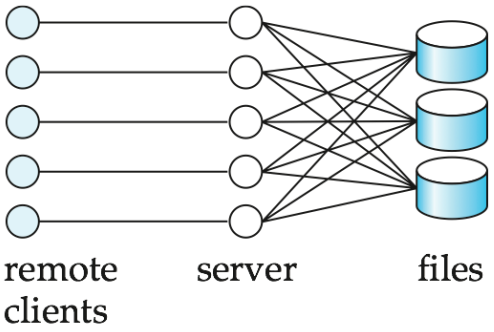



(d) Many-server, many-router model

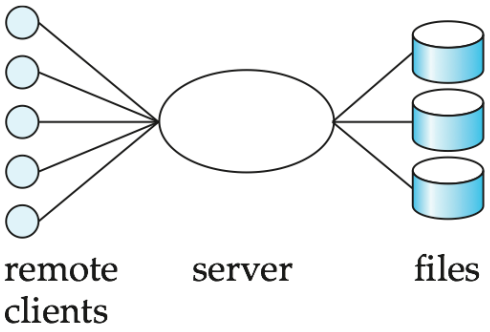
Fig 26.02

# TP Monitor Architectures [2/3]

- **Process per client model** - instead of individual login session per terminal, server process communicates with the terminal, handles authentication, and executes actions.
  - Memory requirements are high
  - Multitasking- high CPU overhead for context switching between processes
- **Single process model** - all remote terminals connect to a single server process.
  - Used in client-server environments
  - Server process is multi-threaded; low cost for thread switching
  - No protection between applications
  - Not suited for parallel or distributed databases



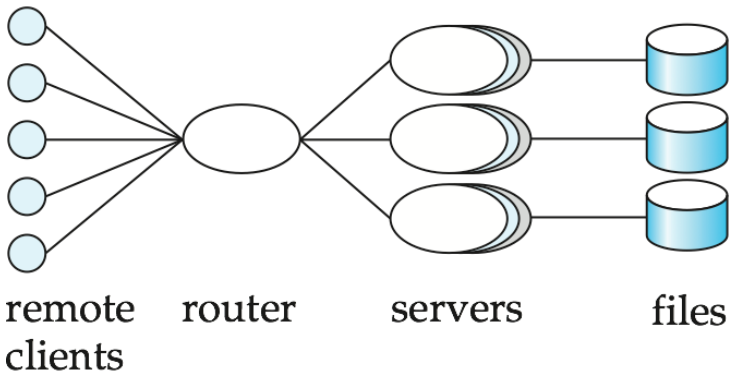

(a) Process-per-client model



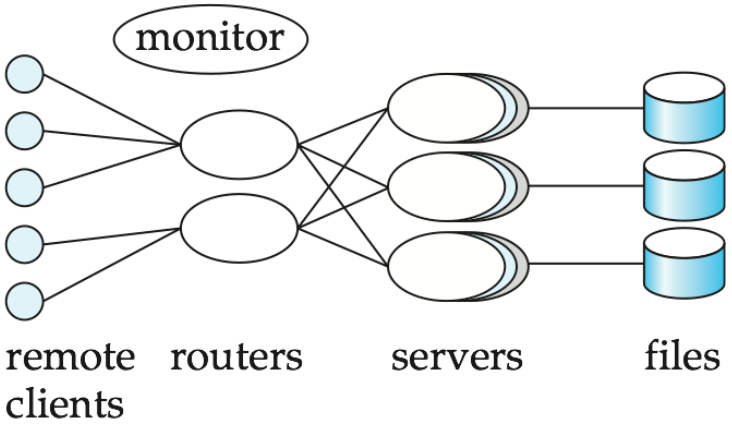

(b) Single-server model

# TP Monitor Architectures [3/3]

- **Many-server single-router model** - multiple application server processes access a common database; clients communicate with the application through a single communication process that routes requests.
  - Independent server processes for multiple applications
  - Multithread server process
  - Run on parallel or distributed database
- **Many server many-router model** - multiple processes communicate with clients.
  - Client communication processes interact with router processes that route their requests to the appropriate server.
  - Controller process starts up and supervises other processes.

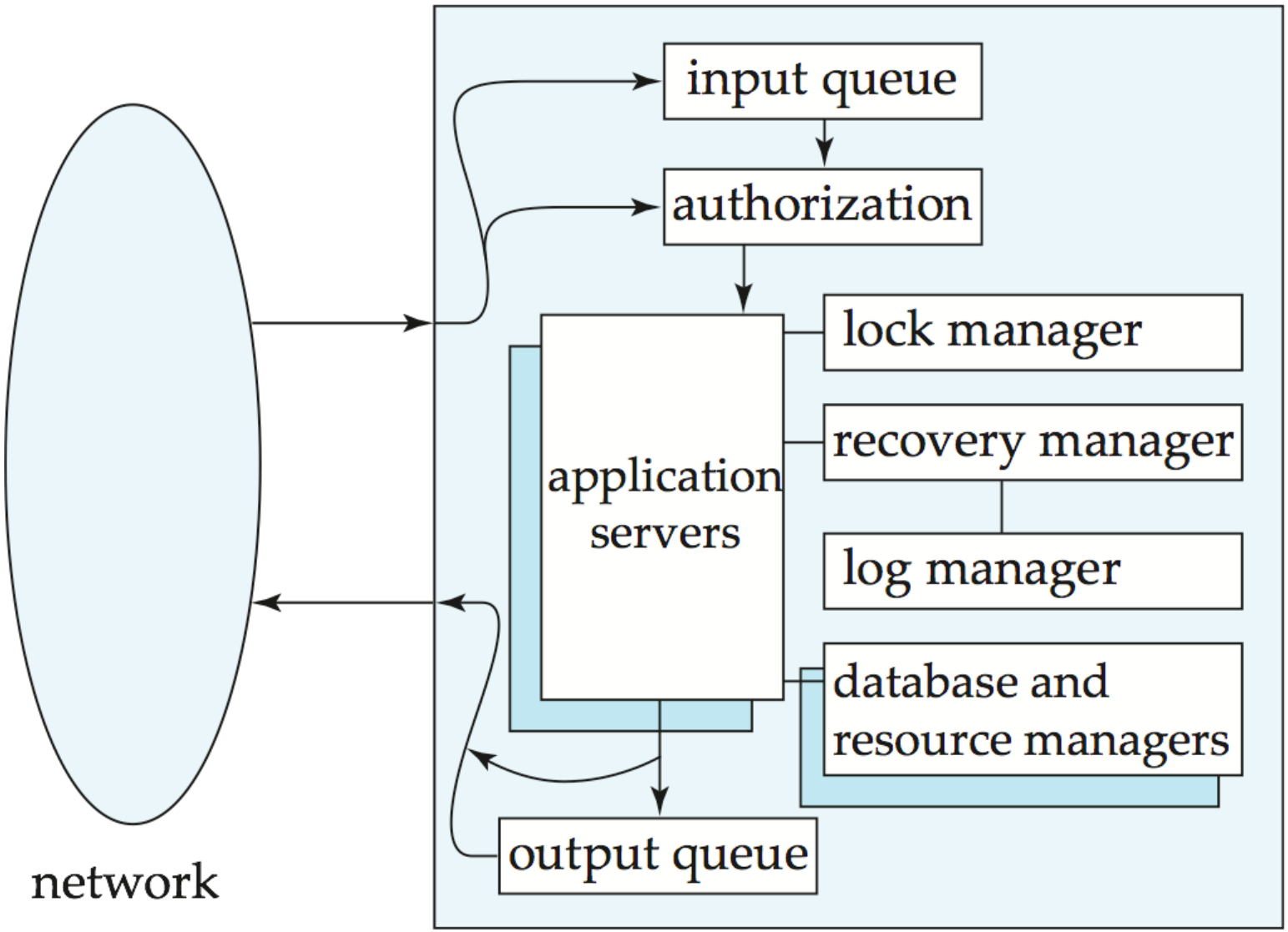



(c) Many-server, single-router model




(d) Many-server, many-router model

# Detailed Structure of a TP Monitor [1/2]



# Detailed Structure of a TP Monitor [2/2]

- Queue manager handles incoming messages
- Some queue managers provide **persistent or durable message queueing** contents of queue are safe even if systems fails.
- Durable queueing of outgoing messages is important
  - application server writes message to durable queue as part of a transaction
  - once the transaction commits, the TP monitor guarantees message is eventually delivered, regardless of crashes.
  - ACID properties are thus provided even for messages sent outside the database
- Many TP monitors provide locking, logging and recovery services, to enable application servers to implement ACID properties by themselves.

# Application Coordination using TP Monitors

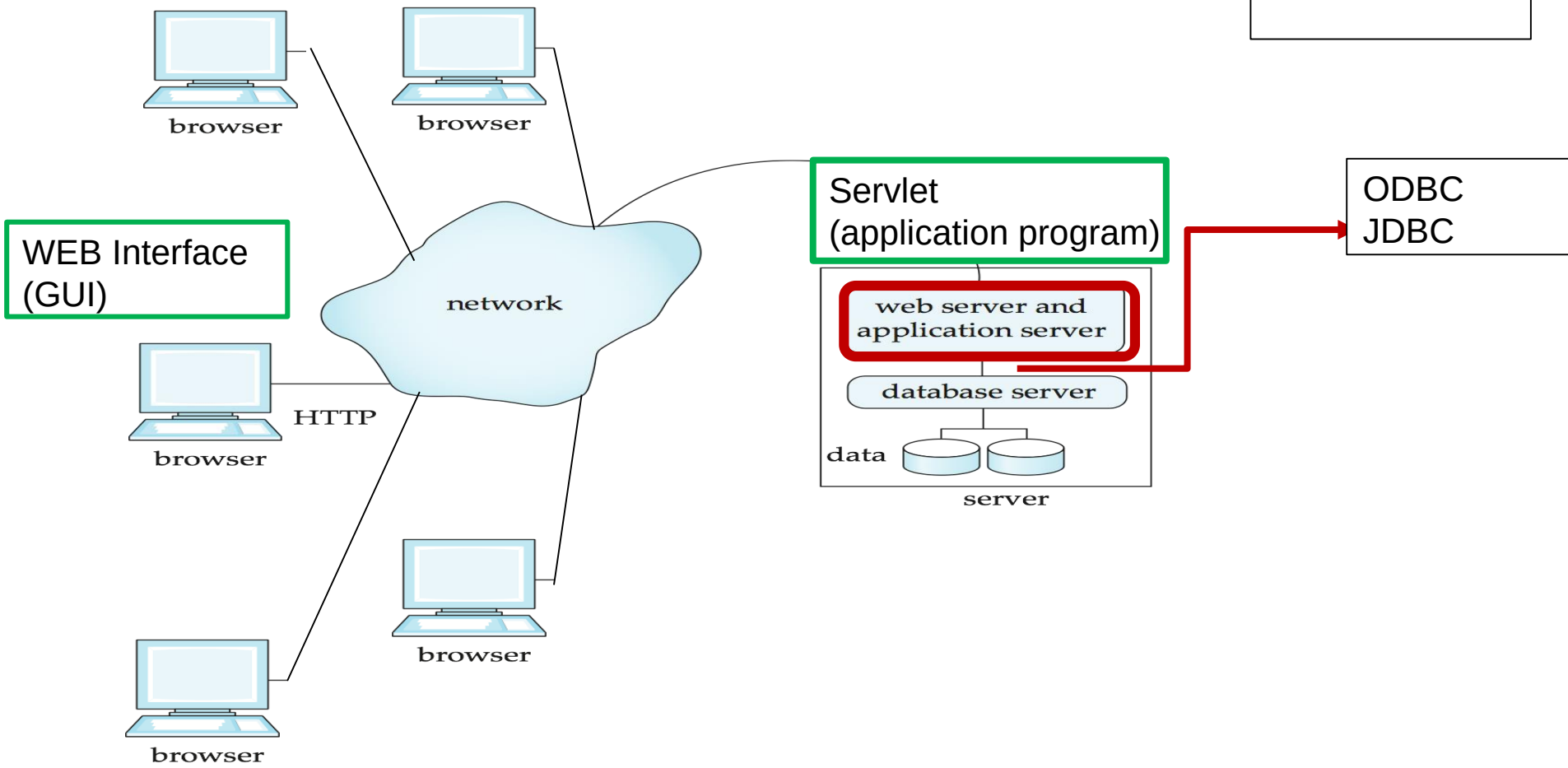
- A TP monitor treats each subsystem as a **resource manager** that provides transactional access to some set of resources.
- The interface between the TP monitor and the resource manager is defined by a set of transaction primitives
- The resource manager interface is defined by the X/Open Distributed Transaction Processing standard.
- TP monitor systems provide a **transactional remote procedure call (transactional RPC)** interface to their service
  - Transactional RPC provides calls to enclose a series of RPC calls within a transaction.
  - Updates performed by an RPC are carried out within the scope of the transaction, and can be rolled back if there is any failure.

- Interface between TP monitor and resource manager
  - begin_transaction
  - commit_transaction
  - abort_transaction
  - prepare_to_commit_transaction

참고자료

# Web Application Server

WAS:
Web logic
Jeus (Tmax)
JBoss



# Chapter 26: Advanced Transaction Processing

- 26.1 Transaction-Processing Monitors
- 26.2 Transactional Workflows
- 26.3 Transactions in E-Commerce
- 26.4 Main Memory Databases
- 26.5 Real-Time Transaction Systems
- 26.6 Long-Duration Transactions

# Transactional Workflows

- **Workflows** are activities that involve the coordinated execution of multiple tasks performed by different processing entities.
- With the growth of networks, and the existence of multiple autonomous database systems, workflows provide a convenient way of carrying out tasks that involve multiple systems.
- Example of a workflow delivery of an email message, which goes through several mails systems to reach destination.
  - Each mailer performs a tasks: forwarding of the mail to the next mailer.
  - If a mailer cannot deliver mail, failure must be handled semantically (delivery failure message).
- Workflows usually involve humans: e.g. loan processing, or purchase order processing.

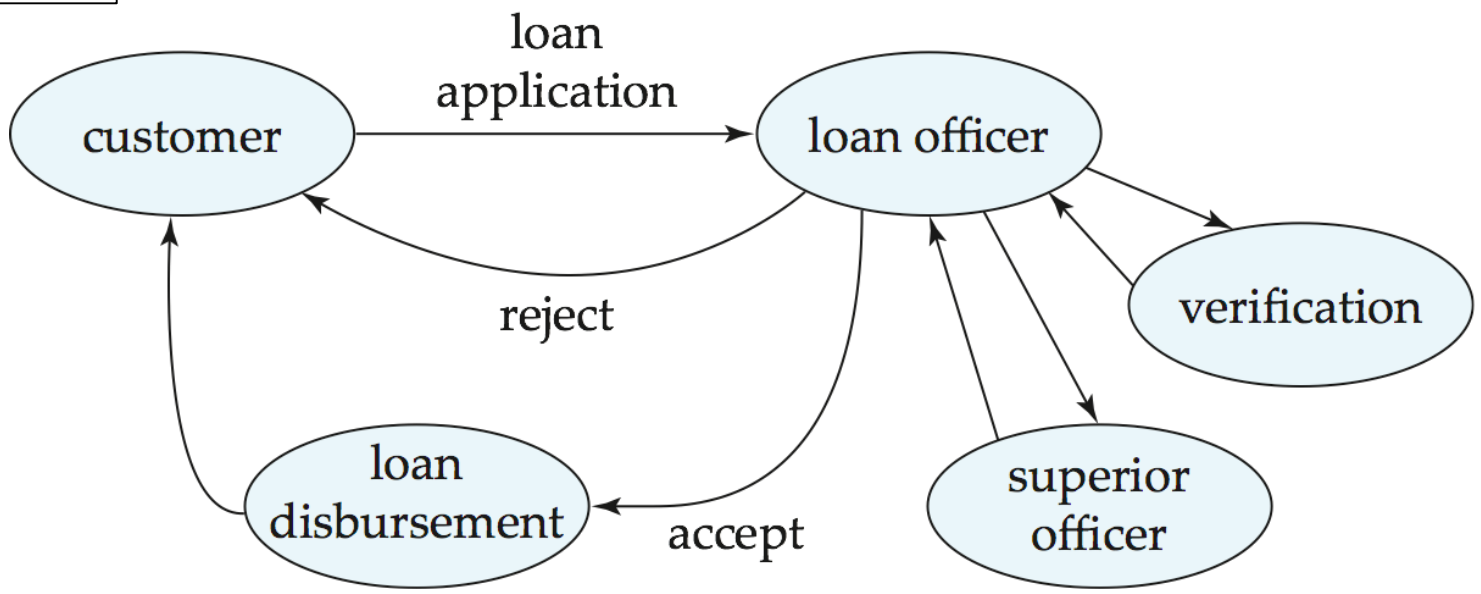
# Examples of Workflows

Fig 26.03

| Workflow application | Typical task | Typical processing entity |
|---|---|---|
| electronic-mail routing | electronic-mail message | mailers |
| loan processing | form processing | humans, application software |
| purchase-order processing | form processing | humans, application software, DBMSs |

# Loan Processing Workflow

Fig 26.04




- In the past, workflows were handled by creating and forwarding paper forms
- Computerized workflows aim to automate many of the tasks. But the humans still play role e.g., in approving loans

# Transactional Workflows

- Must address following issues to computerize a workflow
  - Specification of workflows - detailing the tasks that must be carried out and defining the execution requirements
  - Execution of workflows - execute transactions specified in the workflow while also providing traditional database safeguards related to the correctness of computations, data integrity, and durability
  - E.g.: Loan application should not get lost even if system fails
- Extend transaction concepts to the context of workflows
- State of a workflow - consists of the collection of states of its constituent tasks, and the states (i.e., values) of all variables in the execution plan

# Workflow Specification

- **Static specification** of task coordination:
  - Tasks and dependencies among them are defined before the execution of the workflow starts.
  - Can establish preconditions for execution of each task: tasks are executed only when their preconditions are satisfied.
  - Defined preconditions through dependencies:
    - Execution states of other tasks.

      "task $t_i$ cannot start until task $t_j$ has ended"
    - Output values of other tasks.

      "task $t_i$ can start if task $t_j$ returns a value greater than 25"
    - External variables, that are modified by external events.

      "task $t_i$ must be started within 24 hours of the completion of task $t_j$"

- **Dynamic task coordination**

  E.g., Electronic mail routing system in which the text to be schedule for a given mail message depends on the destination address and on which intermediate routers are functioning.

# Failure-Automicity Requirements

- Usual ACID transactional requirements are too strong/ unimplementable for workflow applications.
- However, workflows must satisfy some limited transactional properties that guarantee a process is not left in an inconsistent state.
- **Acceptable termination states** - every execution of a workflow will terminate in a state that satisfies the failure-atomicity requirements defined by the designer.
  - Committed - objectives of a workflow have been achieved.
  - Aborted - valid termination state in which a workflow has failed to achieve its objectives.
- A workflow must reach an acceptable termination state even in the presence of system failures.

# Execution of Workflows

Workflow management systems include:

- **Scheduler** - program that process workflows by submitting various tasks for execution, monitoring various events, and evaluation conditions related to intertask dependencies
- **Task agents** - control the execution of a task by a processing entity
- Mechanism to query to state of the workflow system

# Workflow Management System Architectures

- **Centralized** - a single scheduler schedules the tasks for all concurrently executing workflows.
  - used in workflow systems where the data is stored in a central database.
  - easier to keep track of the state of a workflow.
- **Partially distributed** - has one (instance of a) scheduler for each workflow.
- **Fully distributed** - has no scheduler, but the task agents coordinate their execution by communicating with each other to satisfy task dependencies and other workflow execution requirements.
  - used in simplest workflow execution systems
  - based on electronic mail

# Workflow Scheduler

- Ideally scheduler should execute a workflow only after ensuring that it will terminate in an acceptable state
- Consider a workflow consisting of two tasks $S_1$ and $S_2$. Let the failure-atomicity requirement be that either both or neither of the subtransactions should be committed
  - Suppose systems executing $S_1$ and $S_2$ do not provide prepared-to-commit states and $S_1$ or $S_2$ do not have compensating transactions
  - It is then possible to reach a state where one subtransaction is committed and the other aborted. Both cannot then be brought to the same state
  - Workflow specification is unsafe, and should be rejected
- Determination of safety by the scheduler is not possible in general, and is usually left to the designer of the workflow

# Recovery of a Workflow [1/2]

- Ensure that is a failure occurs in any of the workflow-processing components, the workflow eventually reaches an acceptable termination state
- Failure-recovery routines need to restore the state information of the scheduler at the time of failure, including the information about the execution states of each task. Log status information on stable storage
- Handoff of tasks between agents should occur exactly once in spite of failure
- Problem: Repeating handoff on recovery may lead to duplicate execution of task; not repeating handoff may lead to task not being executed
  - Solution: Persistent messaging systems

# Recovery of a Workflow [2/2]

- Persistent messages: messages are stored in permanent message queue and therefore not lost in case of failure.
  - Described in detail in Chapter 19 (Distributed Databases)
- Before an agent commits, it writes to the persistent message queue whatever messages need to be sent out.
- The persistent message system must make sure the messages get delivered eventually if and only if the transaction commits.
- The message system needs to resend a message when the site recovers, if the message is not known to have reached its destination.
- Messages must be logged in stable storage at the receiving end to detect multiple receipts of a message.

# Workflow Management Systems

- In many cases, Workflows are often hand coded as part of applications
  - For example, ERP have numerous workflows built into them

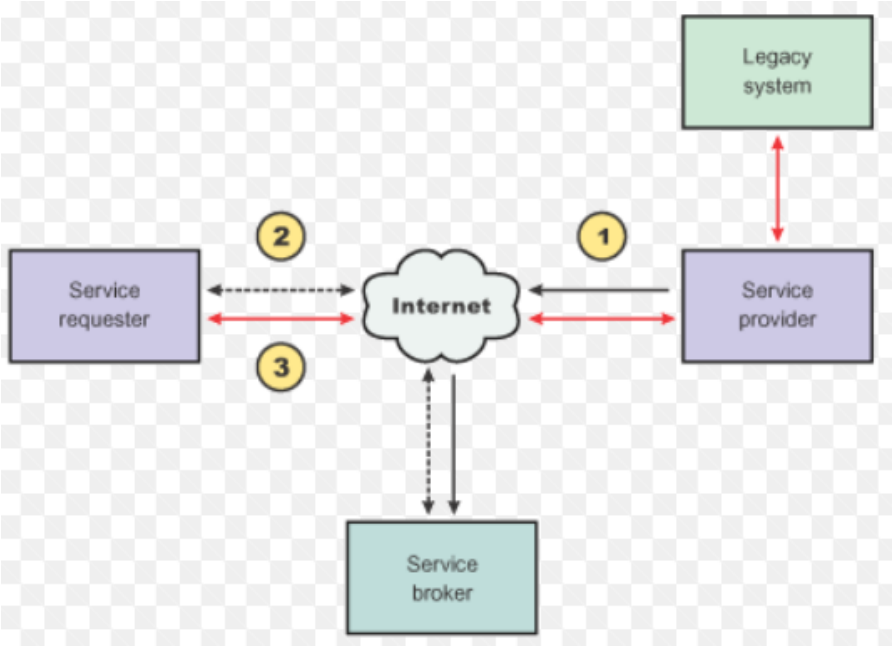
- Automated SW for Business Process Management
- SOA (service oriented architecture)
  - SW system architecture based on invoking services by multiple applications
  - Orchestration: The process logic that controls the workflow by invoking the services
- Business Process Modeling Notation (BPMN)
  - Standard for graphical modeling of workflow
  - XML Process Definition Language (XPDL) for BPMN

- Commercial Workflow Management Systems
  - MicroSoft BizTalk Server
  - IBM WebSphere Business Integration Server Foundation
  - BEA WebLogic Process Edition

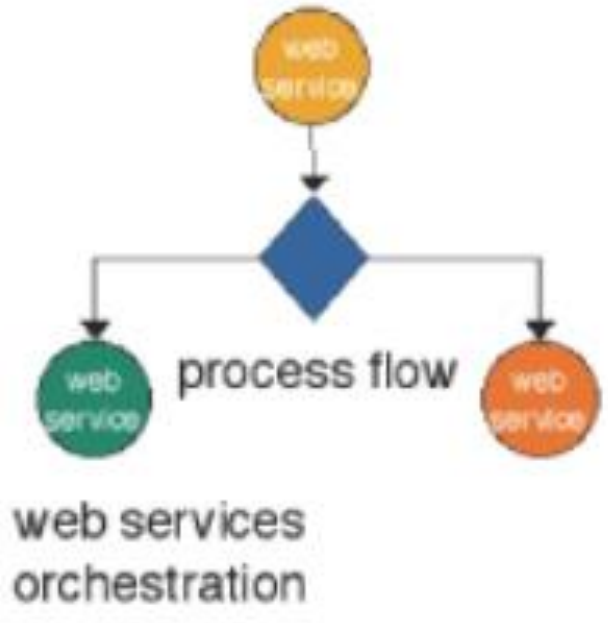
# Workflow Management Systems

참고자료

- Web services approach to SOA (Service-Oriented Architecture)



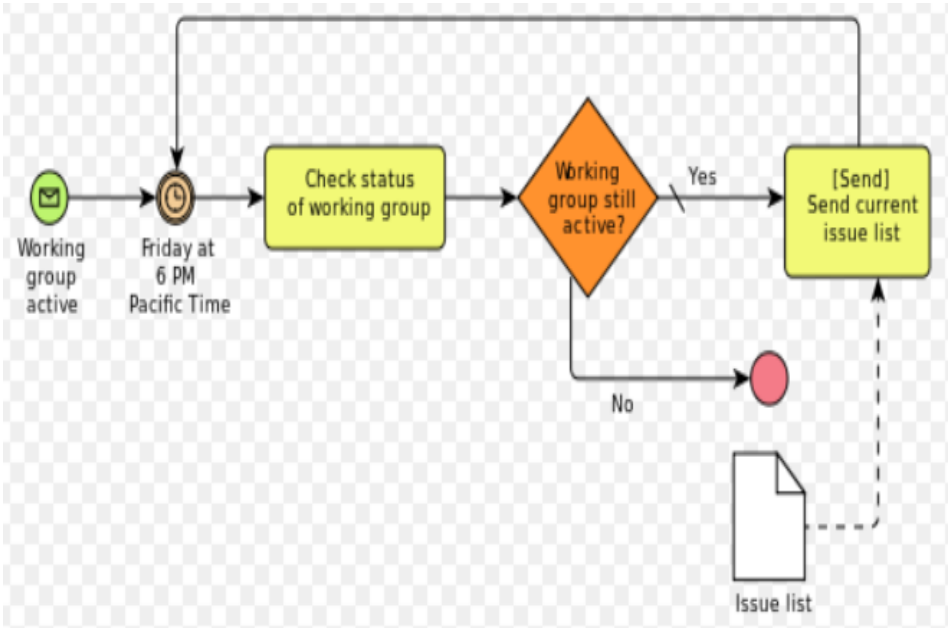

- Orchestration



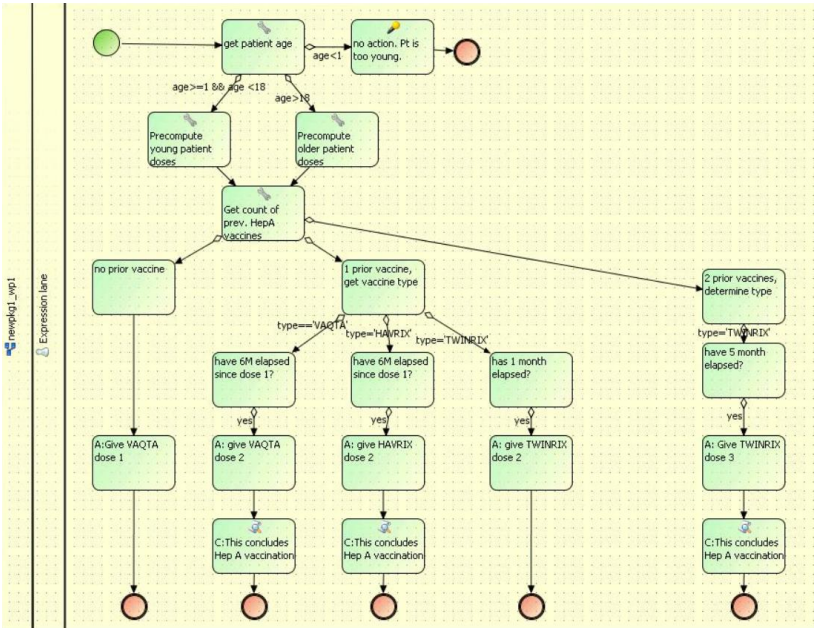
# Workflow Management Systems

참고자료

- Example of a **BPMN** for a process with a normal flow




- Process represented in **XPDL** (healthcare domain example)




- Graphical representation for specifying business processes in a business process model

- Designed to exchange the process definition, both the graphics and the semantics of a workflow business process

# Chapter 26: Advanced Transaction Processing

- 26.1 Transaction-Processing Monitors
- 26.2 Transactional Workflows
- 26.3 Transactions in E-Commerce
- 26.4 Main Memory Databases
- 26.5 Real-Time Transaction Systems
- 26.6 Long-Duration Transactions

# E-Commerce

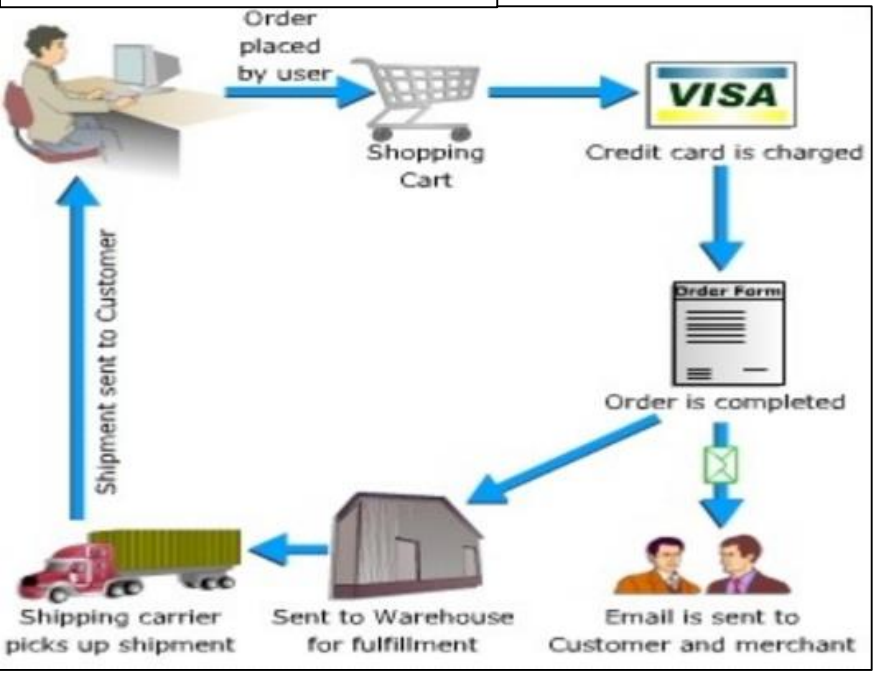
- E-commerce is the process of carrying out various activities related to commerce through electronic means
- Activities include:
  - Presale activities: catalogs, advertisements, etc.
  - Sale process: negotiations on price/quality of service
  - Marketplace: e.g., stock exchange, auctions, reverse auctions
  - Payment for sale
  - Delivery related activities: electronic shipping, or electronic tracking of order processing/shipping
  - Customer support and post-sale service

- Database Issues in E-commerce
- Transactional Issues in E-commerce

참고자료

# E-Commerce

## Process of E-Commerce




## Types of E-Commerce

| | Government | Business | Consumer |
|---|---|---|---|
| Government | **G2G** e.g. Central & State | **G2B** e-Tenders | **G2C** Information to Citizens, online forms |
| Business | **B2G** e.g. procurement | **B2B** Covisint.com EDI, EFT | **B2C** Flipkart.com Rediff.com |
| Consumer | **C2G** Online filing of tax returns | **C2B** Job portals like naukri.com | **C2C** Facebook.com, Ebay.in,flikr.com |

## Word's Leading E-Commerce Companies

Amazon JD.com Wal-Mart eBay Otto Group Alibaba

# E-Catalogs

- Product catalogs must provide searching and browsing facilities
  - Organize products into intuitive hierarchy
  - Keyword search
  - Help customer with comparison of products
- Customization of catalog
  - Negotiated pricing for specific organizations
  - Special discounts for customers based on past history
    - E.g., loyalty discount
  - Legal restrictions on sales
    - Certain items not exposed to under-age customers
- Customization requires extensive customer-specific information

# Marketplaces

- Marketplaces help in negotiating the price of a product when there are multiple sellers and buyers
- Several types of marketplaces
  - Reverse auction
  - Auction
  - Exchange
- Real world marketplaces can be quite complicated due to product differentiation
- Database issues:
  - Authenticate bidders
  - Record buy/sell bids securely
  - Communicate bids quickly to participants
    - Delays can lead to financial loss to some participants
  - Need to handle very large volumes of trade at times
    - E.g., at the end of an auction

# Types of Marketplace

- **Reverse auction system:** single buyer, multiple sellers.
  - Buyer states requirements, sellers bid for supplying items. Lowest bidder wins. (also known as tender system)
  - **Open bidding** vs. **closed bidding**

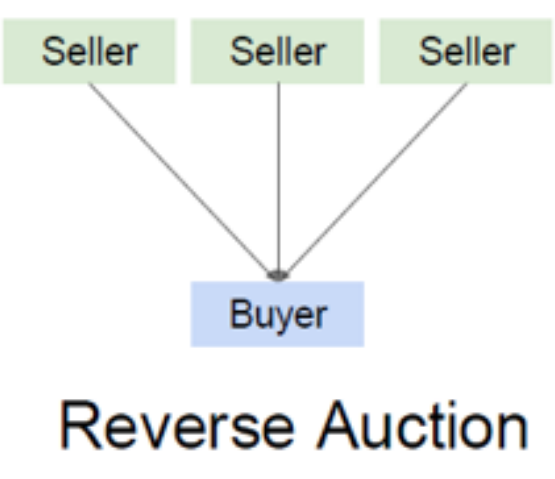
- **Auction:** Multiple buyers, single seller
  - Simplest case: only one instance of each item is being sold
  - Highest bidder for an item wins
  - More complicated with multiple copies, and buyers bid for specific number of copies

- **Exchange:** multiple buyers, multiple sellers
  - E.g., stock exchange
  - Buyers specify maximum price, sellers specify minimum price
  - exchange matches buy and sell bids, deciding on price for the trade
    - e.g., average of buy/sell bids

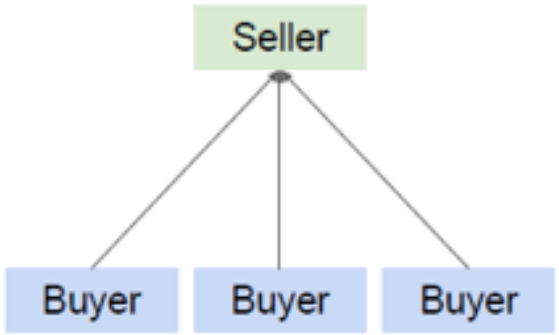
참고자료

# Marketplaces

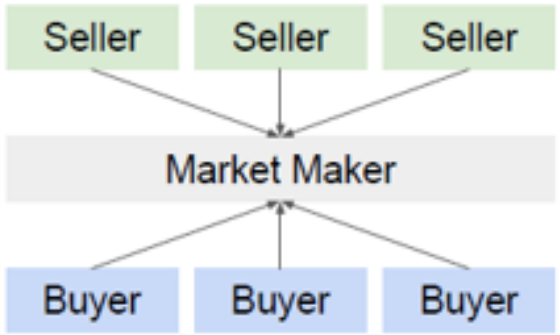
- Types of marketplaces




Reverse Auction




Auction




Exchange

- Database issues
  - Authentication before buying or selling
  - Secure records of buying or selling activities in a database
  - No delays in broadcasting buying/selling.
  - Extremely large volumes of trades

# Order Settlement

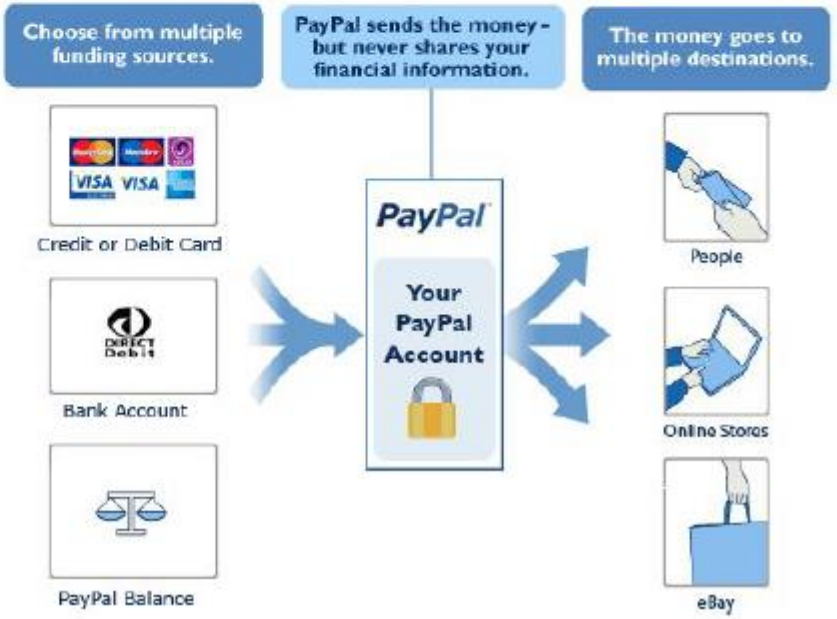
- Order settlement: payment for goods and delivery
- Insecure means for electronic payment: send credit card number
  - Buyers may present some one else's credit card numbers
  - Seller has to be trusted to bill only for agreed-on item
  - Seller has to be trusted not to pass on the credit card number to unauthorized people
- Need secure payment systems
  - Avoid above-mentioned problems
  - Provide greater degree of privacy
    - E.g., not reveal buyers identity to seller
  - Ensure that anyone monitoring the electronic transmissions cannot access critical information

참고자료

# Order Settlement

- Secure Payment Gateways


- Paypal



# Secure Payment Systems [1/2]

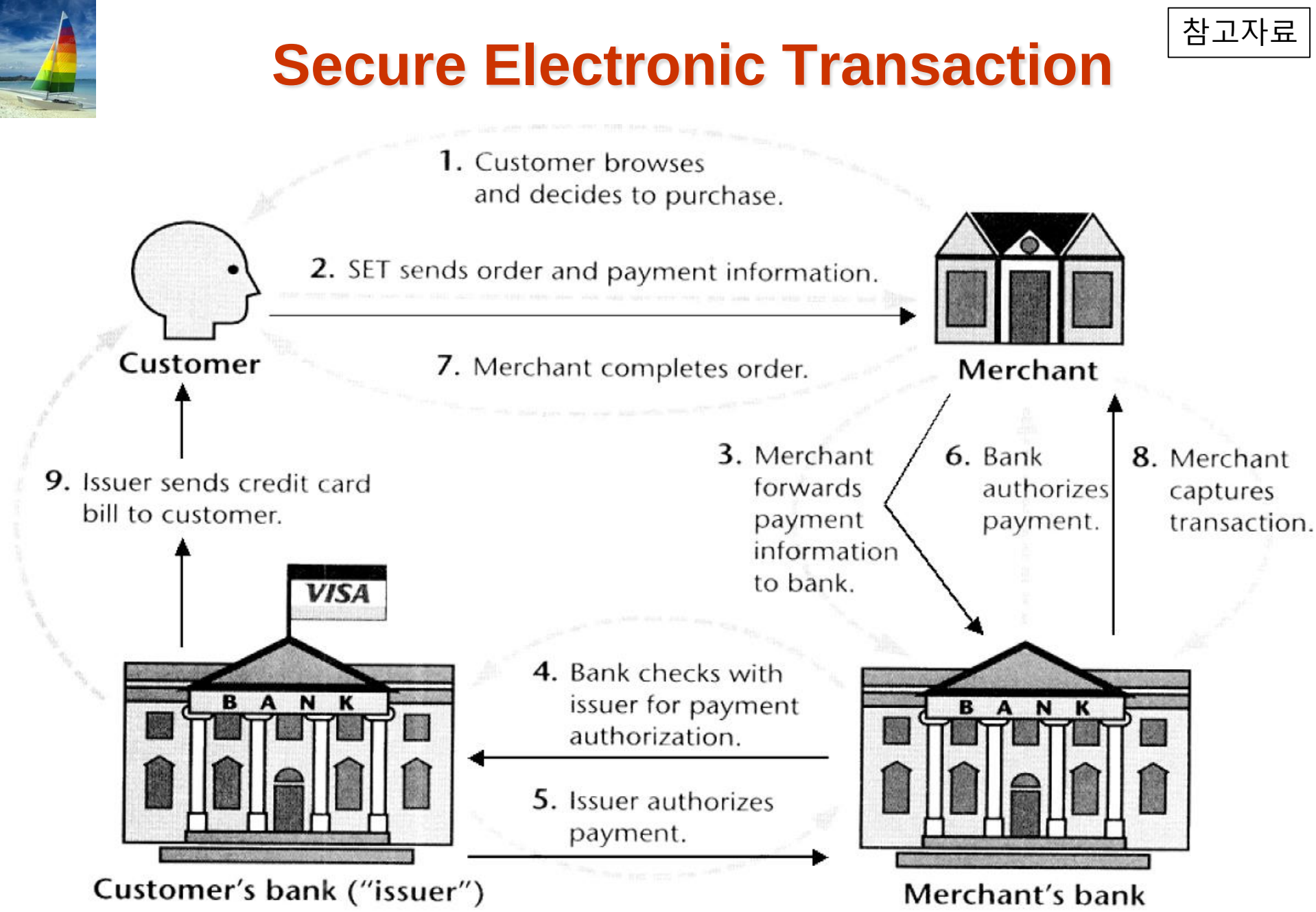
- All information must be encrypted to prevent eavesdropping
  - Public/private key encryption widely used
- Must prevent **person-in-the-middle attacks**
  - E.g., someone impersonates seller or bank/credit card company and fools buyer into revealing information
    - Encrypting messages alone doesn't solve this problem
    - More on this in next slide
- Three-way communication between seller, buyer and credit-card company to make payment
  - Credit card company credits amount to seller
  - Credit card company consolidates all payments from a buyer and collects them together
    - E.g., via buyer's bank through physical/electronic check payment

# Secure Payment Systems [2/2]

- **Digital certificates** are used to prevent impersonation/man-in-the middle attack
  - Certification agency creates digital certificate by encrypting, e.g., seller's public key using its own private key
    - Verifies sellers identity by external means first!
  - Seller sends certificate to buyer
  - Customer uses public key of certification agency to decrypt certificate and find sellers public key
    - Man-in-the-middle cannot send fake public key
  - Sellers public key used for setting up secure communication
- Several secure payment protocols
  - E.g., **Secure Electronic Transaction (SET)**

참고자료

# Secure Electronic Transaction



# Digital Cash

- Credit-card payment does not provide anonymity
  - The SET protocol hides buyers identity from seller
  - But even with SET, buyer can be traced with help of credit card company
- Digital cash systems provide anonymity similar to that provided by physical cash
  - E.g., **Dig Cash**
  - Based on encryption techniques that make it impossible to find out who purchased digital cash from the bank
  - Digital cash can be spent by purchaser in parts
    - much like writing a check on an account whose owner is anonymous

# Chapter 26: Advanced Transaction Processing

- 26.1 Transaction-Processing Monitors
- 26.2 Transactional Workflows
- 26.3 Transactions in E-Commerce
- 26.4 Main Memory Databases
- 26.5 Real-Time Transaction Systems
- 26.6 Long-Duration Transactions

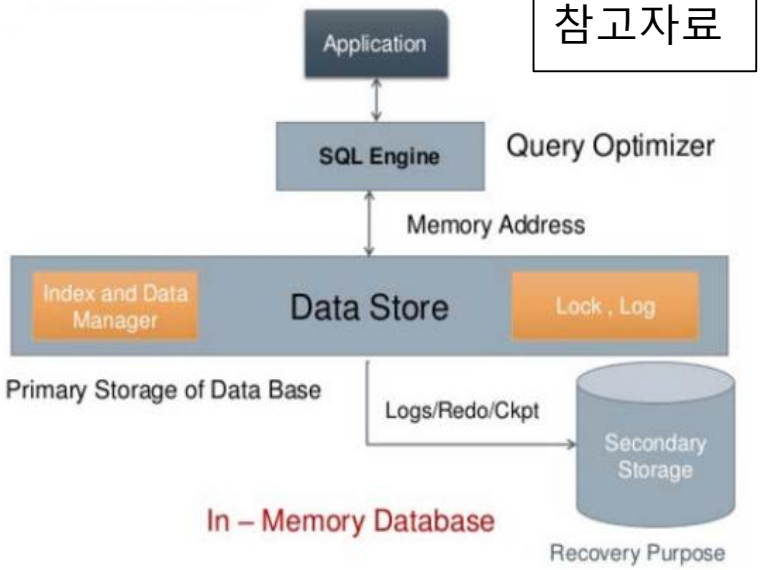
# High-Performance Transaction Systems

- High-performance hardware and parallelism help improve the rate of transaction processing, but are insufficient to obtain high performance:
  - Disk I/O is a bottleneck — I/O time (10 milliseconds) has no decreased at a rate comparable to the increase in processor speeds.
  - Parallel transactions may attempt to read or write the same data item, resulting in data conflicts that reduce effective parallelism
- We can reduce the degree to which a database system is disk bound by increasing the size of the database buffer

# Main-Memory Database

- Commercial 64-bit systems can support main memories of tens of gigabytes
- Memory resident data allows faster processing of transactions
- Disk-related limitations:
  - Logging is a bottleneck when transaction rate is high
  - Use group-commit to reduce number of output operations (Will study two slides ahead.)
  - If the update rate for modified buffer blocks is high, the disk data-transfer rate could become a bottleneck
  - If the system crashes, all of main memory is lost

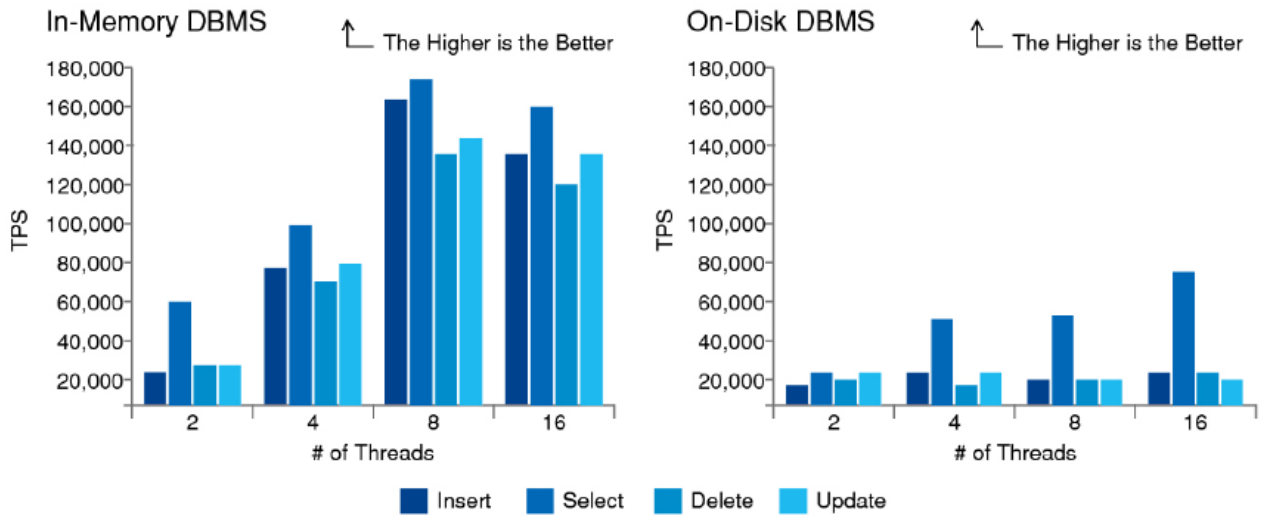
참고자료



참고자료

# In-Memory vs. Disk Resident

| | In-Memory | Disk Resident |
|---|---|---|
| CPU Usage | Fewer CPU cycles with simple search algorithms | Heavy CPU usage with complex search algorithms |
| Disk I/O | Minimal I/O for recovery | I/O for all CRUD operations (including I/O for cache copies) |
| Storage Usage | Efficient use of storage | Less efficient due to cheaper and abundant storage |
| Persistence | Volatile (recovery with disk) | Non-Volatile |
| Locking | Coarse-grained | Fine-grained |
| **Pros** | **Fast with simple optimizations** | **Easily scalable and stable** |
| **Cons** | **Difficult to scale data, must manage a separate disk for persistency** | **Slow, due to complex structures** |

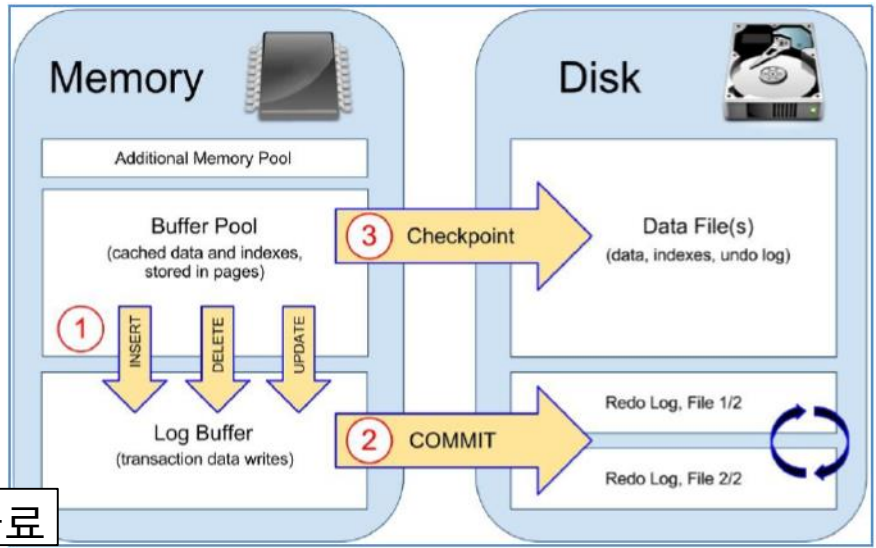


# Main-Memory Database Optimizations

- To reduce space overheads, main-memory databases can use structures with pointers crossing multiple pages. In disk databases, the I/O cost to traverse multiple pages would be excessively high
- No need to pin buffer pages in memory before data are accessed, since buffer pages will never be replaced
- Design query-processing techniques to minimize space overhead - avoid exceeding main memory limits during query evaluation
- Improve implementation of operations such as locking and latching, so they do not become bottlenecks
- Optimize recovery algorithms, since pages rarely need to be written out to make space for other pages

# Group Commit

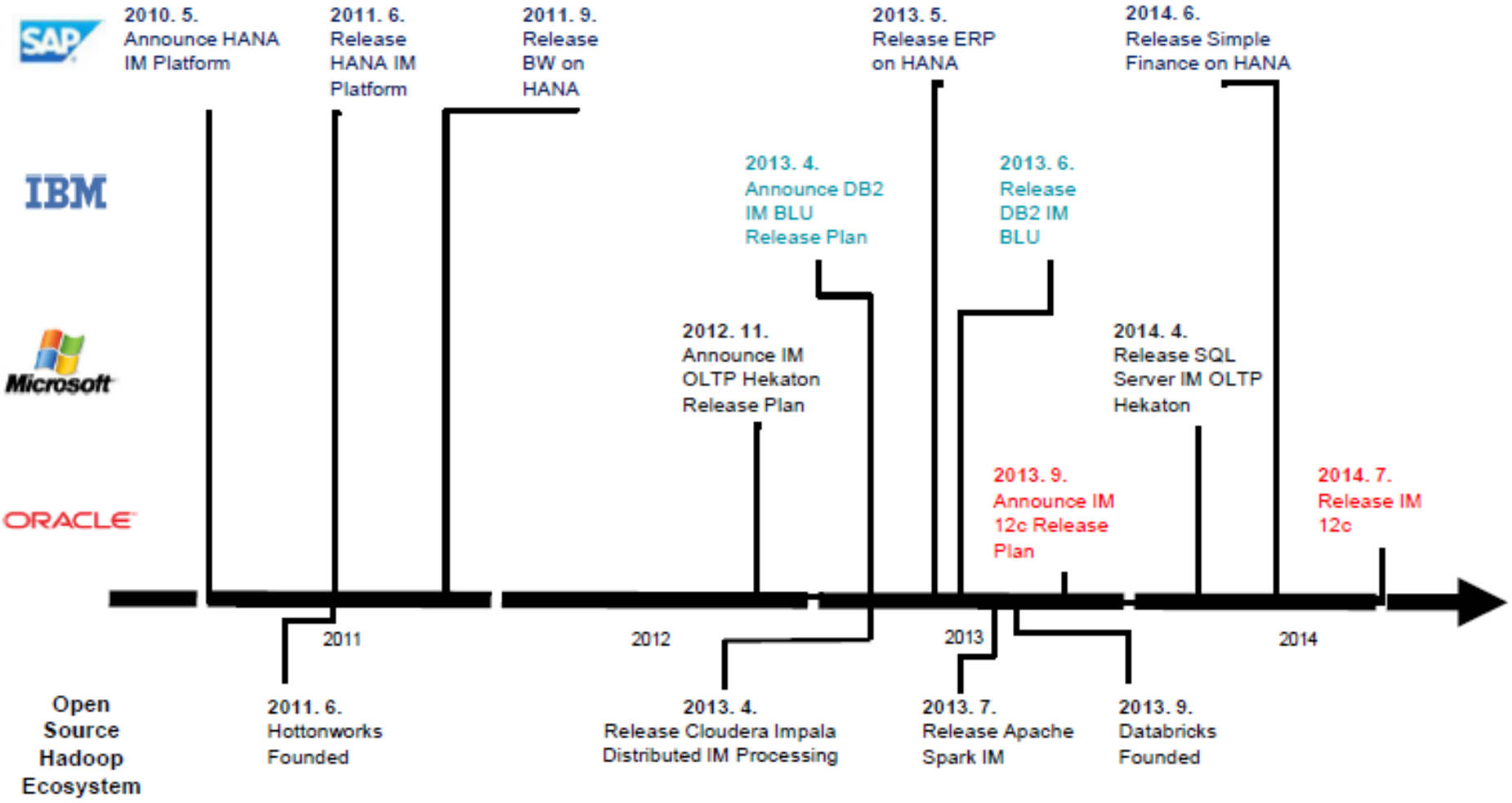
- Idea: Instead of performing output of log records to stable storage as soon as a transaction is ready to commit, wait until
  - log buffer block is full, or
  - a transaction has been waiting sufficiently long after being ready to commit
- Results in fewer output operations per committed transaction, and correspondingly a higher throughput
- However, commits are delayed until a sufficiently large group of transactions are ready to commit, or a transaction has been waiting long enough-leads to slightly increased response time
- Above delay acceptable in high-performance transaction systems since log buffer blocks will fill up quickly




참고자료

참고자료

# In-Memory DB Products



# Chapter 26: Advanced Transaction Processing

- 26.1 Transaction-Processing Monitors
- 26.2 Transactional Workflows
- 26.3 Transactions in E-Commerce
- 26.4 Main Memory Databases
- 26.5 Real-Time Transaction Systems
- 26.6 Long-Duration Transactions

# Real-Time Transaction Systems

- In systems with real-time constraints, correctness of execution involves both database consistency and the satisfaction of deadlines
  - **Hard deadline** – Serious problems may occur if task is not completed within deadline
  - **Firm deadline** - The task has zero value if it completed after the deadline.
  - **Soft deadline** - The task has diminishing value if it is completed after the deadline
- The wide variance of execution times for read and write operations on disks complicates the transaction management problem for time-constrained systems
  - main-memory databases are thus often used
  - Waits for locks, transaction aborts, contention for resources remain as problems even if data is in main memory
- Design of a real-time system involves ensuring that enough processing power exists to meet deadline without requiring excessive hardware resources

# Chapter 26: Advanced Transaction Processing

- 26.1 Transaction-Processing Monitors
- 26.2 Transactional Workflows
- 26.3 Transactions in E-Commerce
- 26.4 Main Memory Databases
- 26.5 Real-Time Transaction Systems
- 26.6 Long-Duration Transactions

# Long Duration Transactions

Traditional concurrency control techniques do not work well when user interaction is required:

- **Long duration**: Design edit sessions are very long
- **Exposure of uncommitted data**: E.g., partial update to a design
- **Subtasks**: support partial rollback
- **Recoverability**: on crash state should be restored even for yet-to-be committed data, so user work is not lost
- **Performance**: fast response time is essential so user time is not wasted
- Represent as a nested transaction
  - atomic database operations (read/write) at a lowest level
- If transaction fails, only active short-duration transactions abort.
- Active long-duration transactions resume once any short duration transactions have recovered
- The efficient management of long-duration waits, and the possibility of aborts
- Need alternatives to waits and aborts; alternative techniques must ensure correctness without requiring serializability

# Concurrency Control [1/2]

- Correctness without serializability:
  - Correctness depends on the specific consistency constraints for the databases.
  - Correctness depends on the properties of operations performed by each transaction.
- Use database consistency constraints as to split the database into subdatabases on which concurrency can be managed separately.
- Treat some operations besides read and write as fundamental low-level operations and extend concurrency control to deal with them.

# Concurrency Control [2/2]

A non-conflict-serializable schedule that preserves the sum of A + B

Fig 26.05

| $T_1$ | $T_2$ |
|---|---|
| read($A$) | |
| $A : A - 50$ | |
| write($A$) | |
| | read($B$) |
| | $B : B - 10$ |
| | write($B$) |
| read($B$) | |
| $B : B + 50$ | |
| write($B$) | |
| | read($A$) |
| | $A : A + 10$ |
| | write($A$) |

# Nested and Multilevel Transactions [1/2]

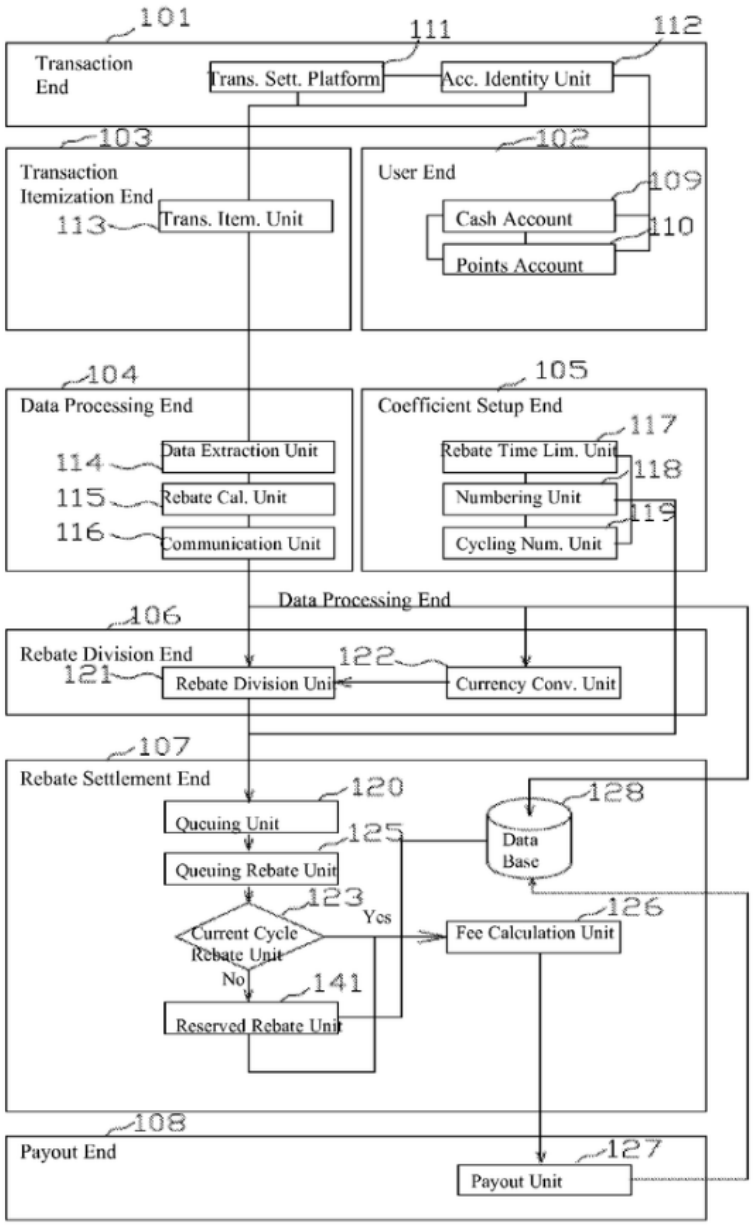
- A **nested or multilevel transaction** $T$ is represented by a set $T = \{t_1, t_2, ..., t_n\}$ of subtransactions and a partial order $P$ on $T$.
- A subtransaction $t_i$ in $T$ may abort without forcing $T$ to abort.
- Instead, $T$ may either restart $t_i$, or simply choose not to run $t_i$.
- If $t_i$ commits, this action does not make $t_i$, permanent (unlike the situation in Chapter 15). Instead, $t_i$, commits to $T$, and may still abort (or require compensation) if $T$ aborts.
- An execution of $T$ must not violate the partial order $P$, i.e., if an edge $t_i \rightarrow t_j$ appears in the precedence graph, then $t_j \rightarrow t_i$ must not be in the transitive closure of $P$.

# Nested and Multilevel Transactions [2/2]

- Subtransactions can themselves be nested/multilevel transactions.
  - Lowest level of nesting: standard read and write operations.
- Nesting can create higher-level operations that may enhance concurrency.
- Types of nested/ multilevel transactions:
  - **Multilevel transaction**: subtransaction of *T* is permitted to release locks on completion.
  - **Saga**: multilevel long-duration transaction.
  - **Nested transaction**: locks held by a subtransaction $t_i$ of *T* are automatically assign to *T* on completion of $t_i$.

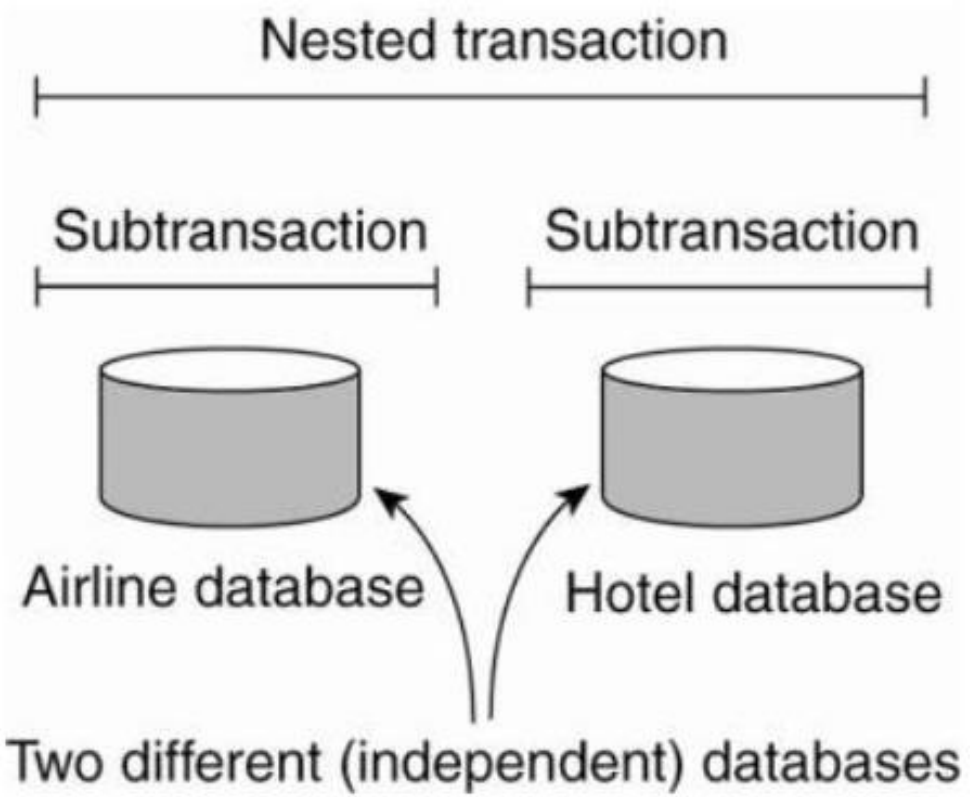
참고자료

# Nested and Multilevel Transactions

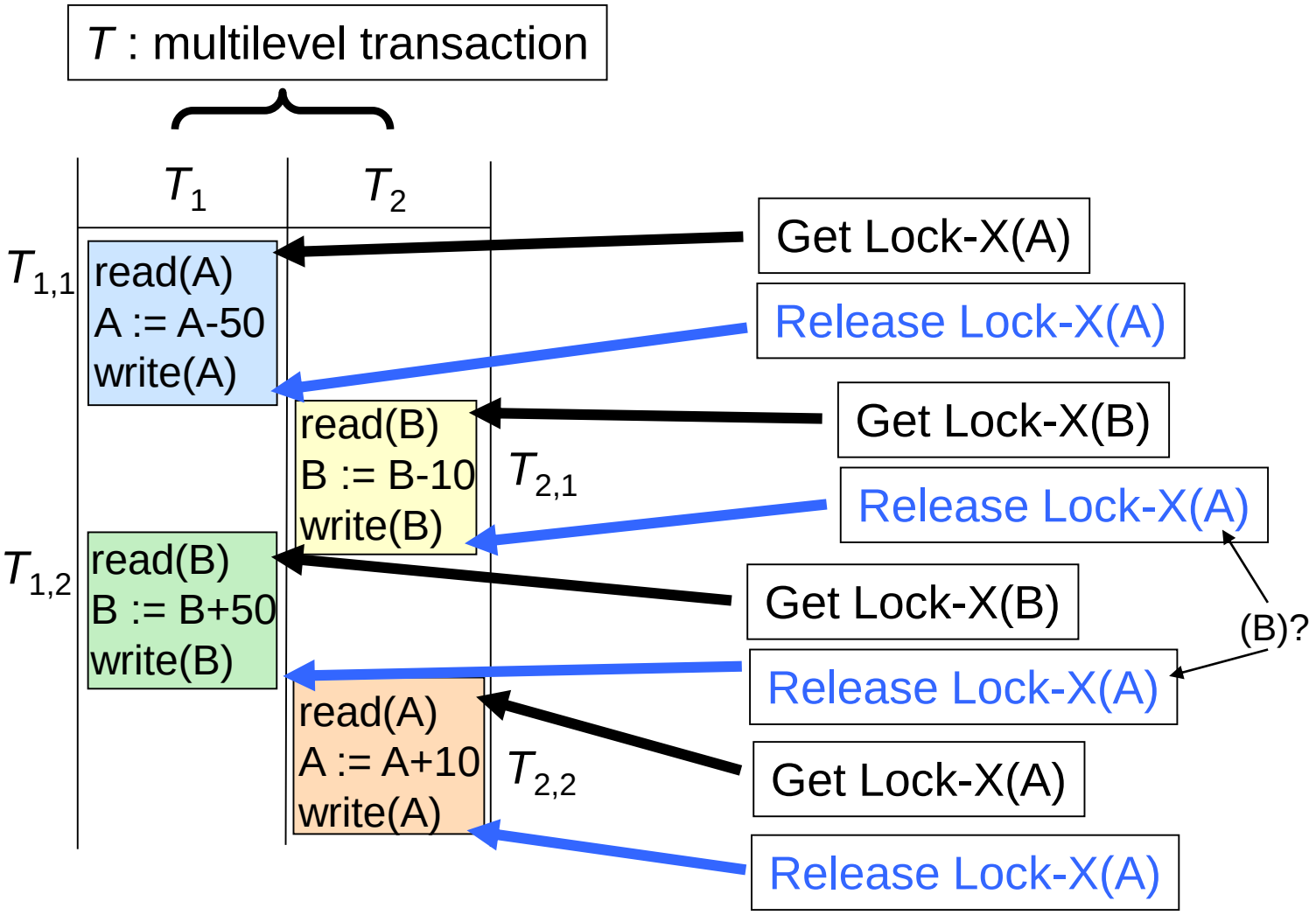



- Multi-level Transaction
  (Rebate computation example)

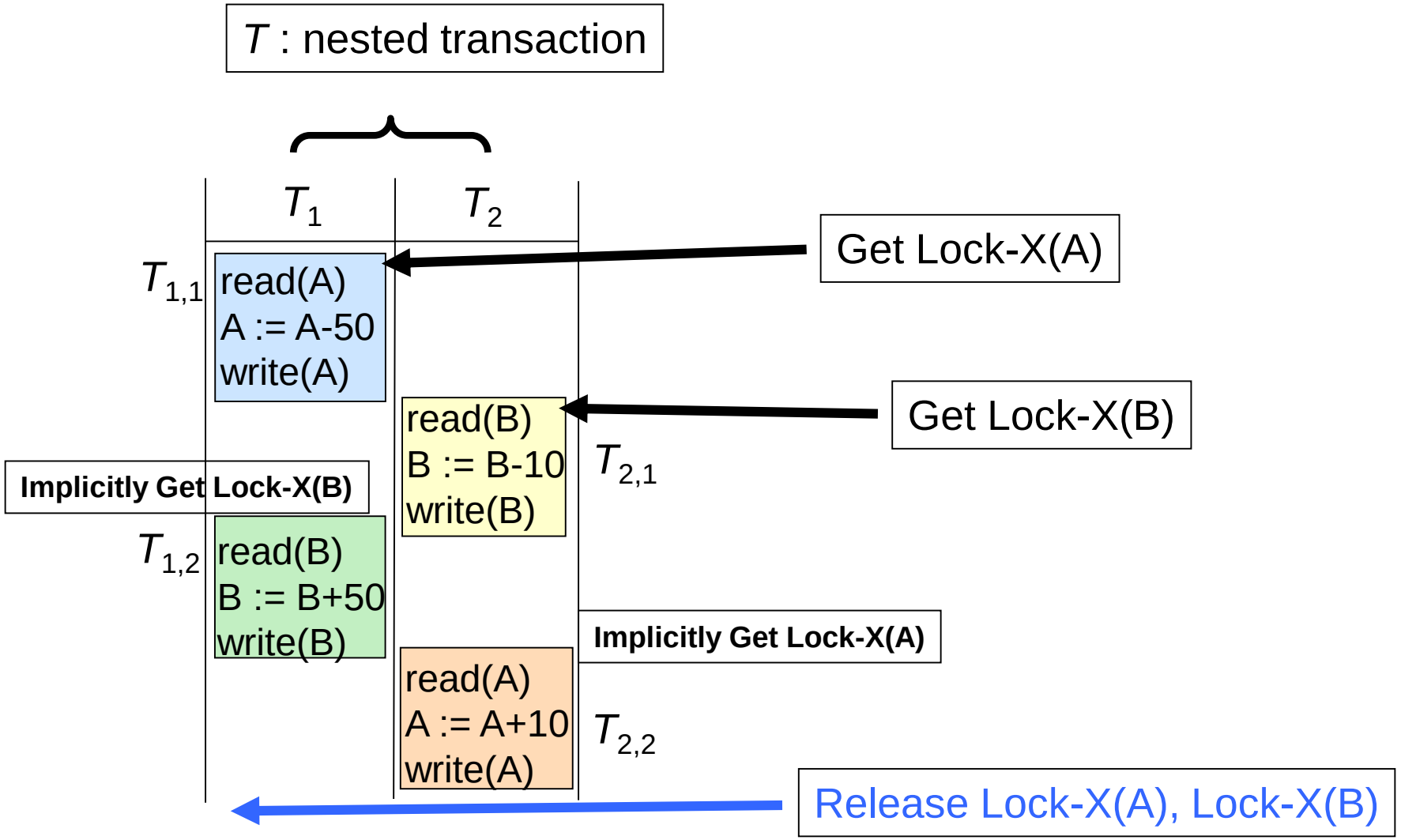
- Nested Transaction



# Multilevel Transaction by Linch: Example

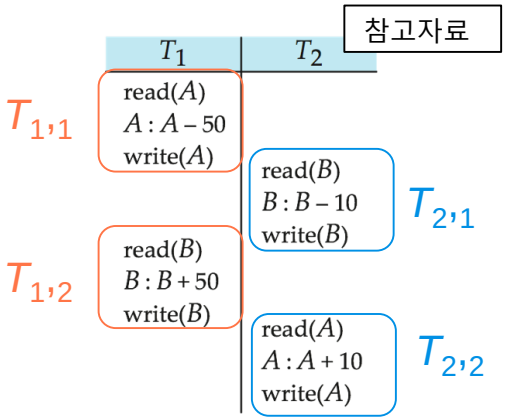


# Nested Transaction by Moss: Example



# Example of Nesting

- Rewrite transaction $T_1$ using subtransactions $T_a$ and $T_b$ that perform increment or decrement operations:
  - $T_1$ consists of
    - $T_{1,1}$, which subtracts 50 from $A$
    - $T_{1,2}$, which adds 50 to $B$
- Rewrite transaction $T_2$ using subtransactions $T_c$ and $T_d$ that perform increment or decrement operations:
  - $T_2$ consists of
    - $T_{2,1}$, which subtracts 10 from $B$
    - $T_{2,2}$, which adds 10 to $A$
- No ordering is specified on subtransactions; any execution generates a correct result.



# Compensating Transactions

- Alternative to undo operation; compensating transactions deal with the problem of cascading rollbacks.
- Instead of undoing all changes made by the failed transaction, action is taken to "**compensate**" for the failure.
- Consider a long-duration transaction $T_i$ representing a travel reservation, with subtransactions $T_{i,1}$, which makes airline reservations, $T_{i,2}$ which reserves rental cars, and $T_{i,3}$ which reserves a hotel room.
  - Hotel cancels the reservation.
  - Instead of undoing all of $T_i$, the failure of $T_{i,3}$ is compensated for by deleting the old hotel reservation and making a new one.
  - Requires use of semantics of the failed transaction.

# Implementation Issues

- For long-duration transactions to survive system crashes, we must log not only changes to the database, but also changes to internal system data pertaining to these transactions
- Logging of updates is made more complex by physically large data items (CAD design, document text); undesirable to store both old and new values
- Two approaches to reducing the overhead of ensuring the recoverability of large data items:
  - Operation logging: Only the operation performed on the data item and the data-item name are stored in the log
  - Logging and shadow paging: Use logging from small data items; use shadow paging for large data items. Only modified pages need to be stored in duplicate