ADVANCED TRANSACTION PROCESSING

COMPILED BY
M ABDUL JAWAD

TRANSACTION – PROCESSING MONITORS

- **Transaction-Processing monitors (TP monitors)** are the systems that were developed in the 1970s and 1980s, initially in response to a need to support a large number of remote terminals – **{ *Airline-Reservation Terminals* }** from a single computer.

- **TP Monitor – *Teleprocessing Monitor.***

- Provide infrastructure for building and administering complex transaction processing systems with a large number of clients and multiple servers.

TRANSACTION – PROCESSING MONITORS

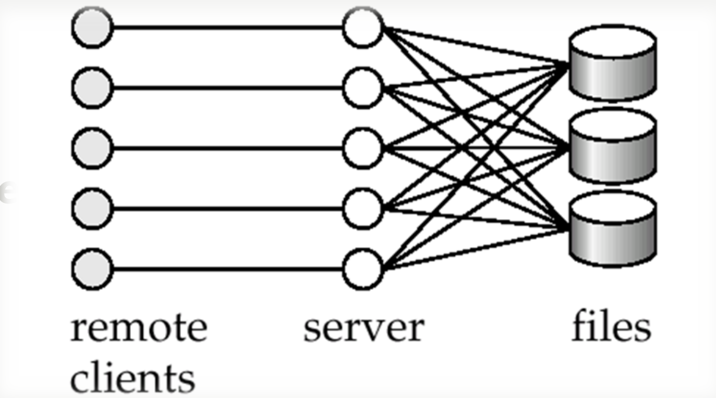
- ***Transaction-Processing Monitors provide services*** such as:
  - Presentation facilities to simplify creating user interfaces.
  - Persistent queuing of client requests and server responses.
  - Routing of client messages to servers.
  - Coordination of two-phase commit when transactions access multiple servers.

- Some commercial TP monitors: CICS from IBM, Pathway from Tandem, Top End from NCR, and Encina from Transarc.

TP-MONITOR ARCHITECTURES

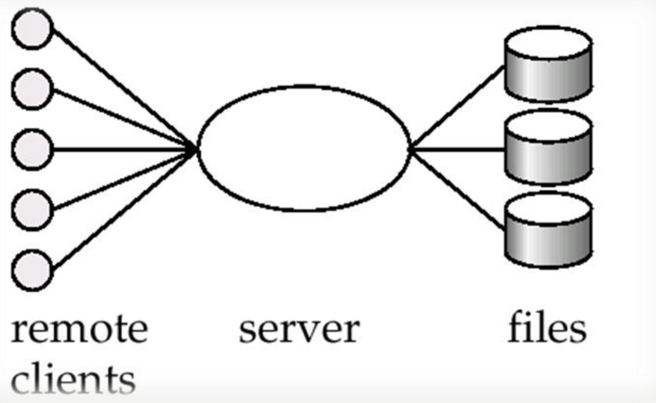
- Large-scale transaction processing systems are built around a client-server architecture and a number of different implementations exist in a client-server architecture.

- ***Process-per-Client Model***
  - *We have a Server Process for each client; the server performs the authentication and then executes actions requested by the client.*
  - *Per-Process memory requirements are high.*
  - *Operating system divides up available CPU time among processes by switching among them – Overhead of Context Switching.*



TP-MONITOR ARCHITECTURES

- ***Single Server Model***
  - ***We have a single-server process to which all remote clients connect.***
  - ***Remote Clients send requests to the server process which then executes those requests.***
  - ***The Server Process handles tasks – Authentication etc.***
  - ***To avoid blocking other clients when processing a long request for one client, the server process is multithreaded – Lesser Overhead while switching between threads.***
  - No protection between threads.
  - Not suited for parallel or distributed databases.

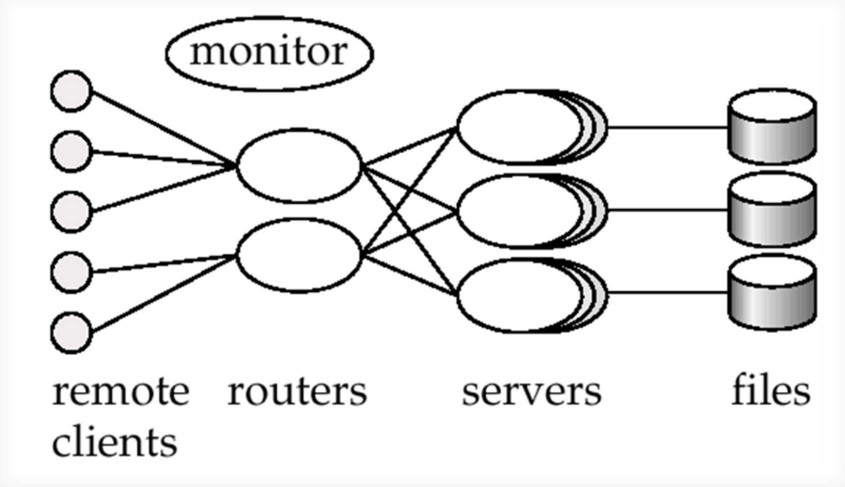


TP-MONITOR ARCHITECTURES

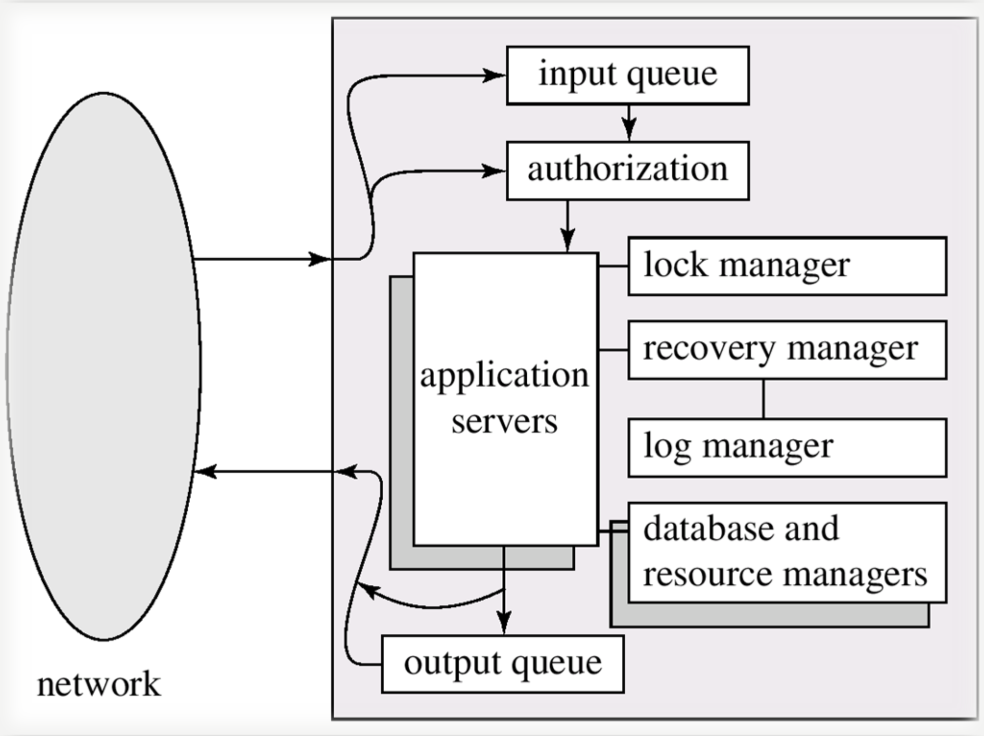
- **Many-Server, Single-Router Model**
  - We have multiple application-server processes that access a common database and let the clients communicate with the application through a single communication process that routes requests.
  - Independent server processes for multiple applications
  - Multithread server process
  - Application servers can run on different sites of a distributed database and communication process can handle the coordination among the processes.

TP-MONITOR ARCHITECTURES

- **Many-Server, Many Router Model**
  - The client communication processes interact with one or more router processes, which route their requests to appropriate server.
  - Controller process starts up and supervises other processes.
  - Implemented in High Performance Web Servers.



TP-MONITOR COMPONENTS



TP-MONITOR COMPONENTS

- 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.
- Many TP monitors provide locking, logging and recovery services, to enable application servers to implement ACID properties by themselves.

APPLICATION COORDINATION USING TP MONITORS

- Applications need to communicate
  - **Multiple Databases**.
  - **Legacy Systems** – Special Data Storage systems built directly on file systems.
  - Finally the **users** or **other applications at remote sites**.
  - In addition to this, they still need to interact **network** that facilitates the communication among various subsystems.
- It is very important to coordinate data accesses and implement ACID Properties.
- Modern TP managers provide such support and facilities for construction and administration of such large applications.

APPLICATION COORDINATION USING TP MONITORS

- TP monitor treats each subsystem as a **Resource Manager** – providing transactional access to some set of resources.
- Interface between TP monitor and Resource Manager is defined by set of transaction primitives:
  - ***Begin_transaction***
  - ***Commit_transaction***
  - ***Abort_transaction***
  - ***Prepare_to_commit_transaction***
- ***The resource manager must also provide other services – Supplying data to the application.***

APPLICATION COORDINATION USING TP MONITORS

- 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

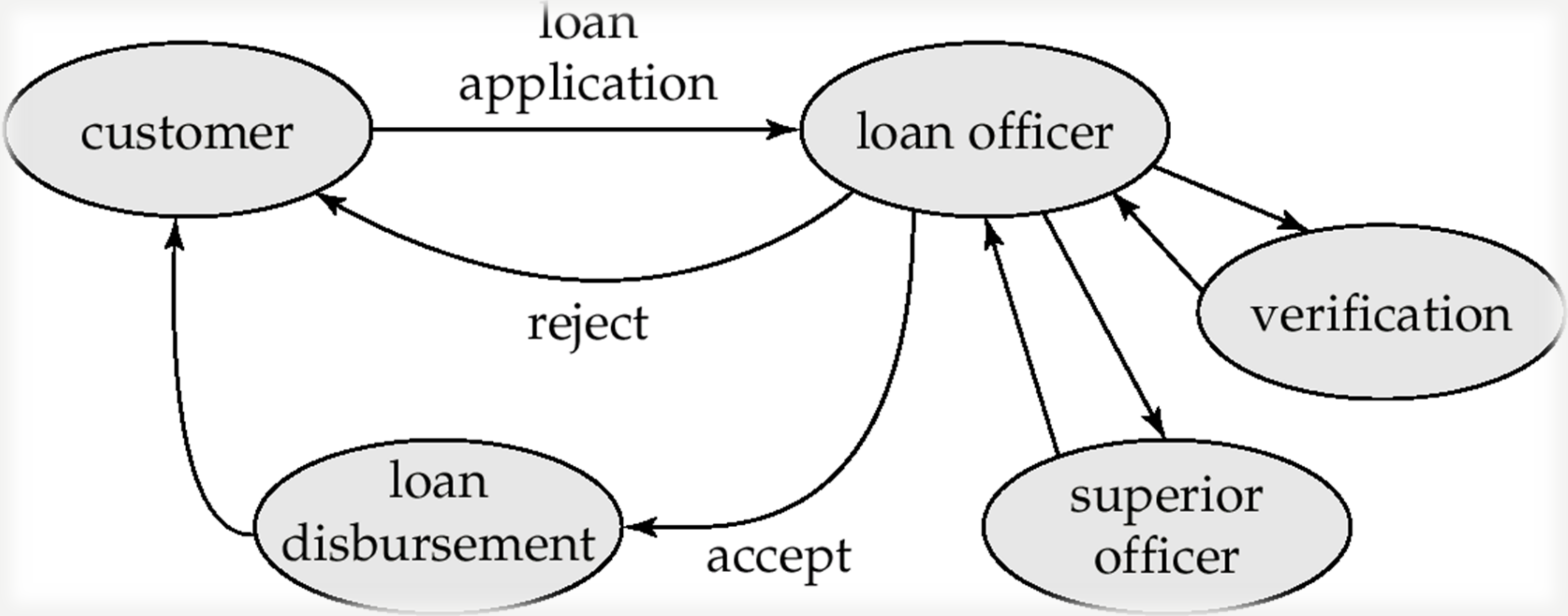
TRANSACTIONAL WORKFLOWS

- **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

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



- 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

- To automate the tasks involved in Loan Processing – We can store the Loan application and associated information in a database.
- Workflow itself then involves handling responsibility from one human to next.
- Workflows are very important in organizations and organizations today have multiple software systems that need to work together.
  - Employee joins an organization and information about the employee is provided to
    - Payroll System.
    - Library System.
    - Authentication Systems.

TRANSACTIONAL WORKFLOWS

- We need to address two activities in general to automate a workflow:
  - **Workflow Specification**.
  - **Workflow Execution.**

- Both activities are complicated by the fact that many organizations use several independently managed information processing systems, that in most cases were developed separately to automate different functions.
- Workflow activities may require interactions among several such systems each performing a task as well as interactions with humans.

WORKFLOW SPECIFICATION

- A **task** may use parameters stored in its variables, may retrieve and update data in the local system, may store its results in its output variables.
- At any time during the execution, the **workflow state** consists of the collection of states of the workflows constituent tasks and the states of all variables in the workflow specification.
- The coordination of tasks can be specified either statically or dynamically.
  – A static specification defines the tasks and dependencies among them, before the execution of workflow begins.
  – A dynamic specification defines the dependencies and execution of tasks on demand and along the route of execution itself.

STATIC SPECIFICATION

- In static specification, the dependencies among the tasks may be simple and everything is known well in advance before any execution of any task starts – Each task may be completed before the next begins.
- The transactions whose preconditions are satisfied are executed. The preconditions can be defined through dependencies like:

  - **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"**

FAILURE-ATOMICITY 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 inter-task dependencies
- **Task agents** - control the execution of a task by a processing entity.
- Mechanism to query to state of the workflow system.
- We have three architectural approaches for development of a workflow management system:
  - **Centralized.**
  - **Partially distributed.**
  - **Fully distributed**.

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 sub-transactions 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 sub-transaction 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.