

# Unit – II

## TP Monitors

# Transaction Processing Monitors (TP Monitors)

- specialize in **managing transactions**
- from their **point of origin—typically on the client**—across one or more servers, and then **back to the originating client**.
- **TP Monitor ensures** that all the systems involved in the transaction are left in a **consistent state**, at the end of a transaction

- TP Monitors know
  - **how to run transactions,**
  - **route them across systems,**
  - **load-balance their execution, and**
  - **restart them after failures.**
- overseer of all aspects of a distributed transaction, regardless of the systems or resource managers used

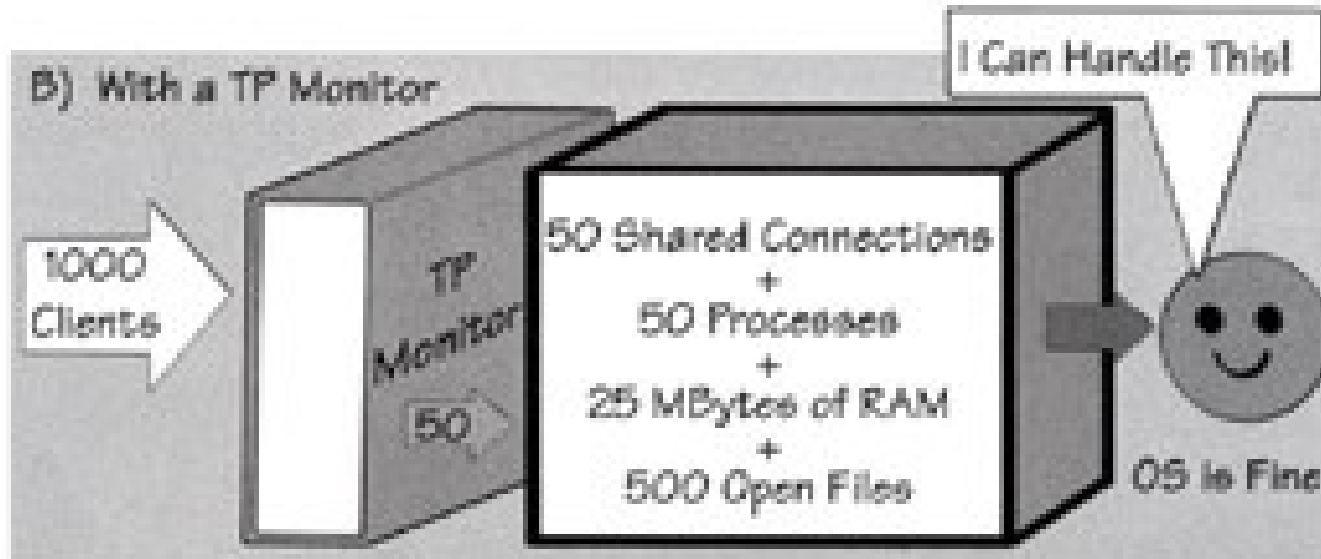
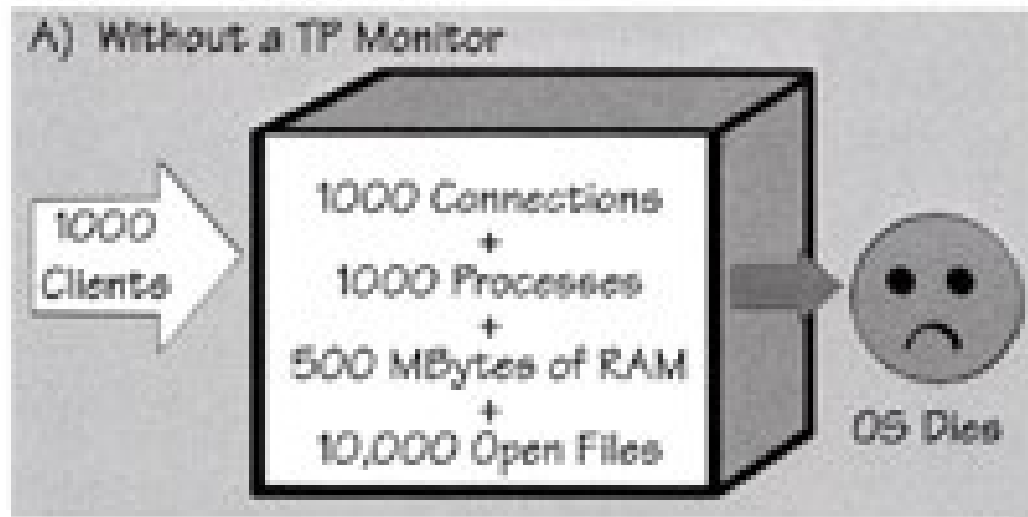
- A TP Monitor can manage
  - resources on a single server
  - or multiple servers,
  - and it can cooperate with other TP Monitors in federated arrangements
- TP Monitor is **"an operating system for transaction processing"**.
- TP Monitor also provides a framework for running middle-tier server applications and components

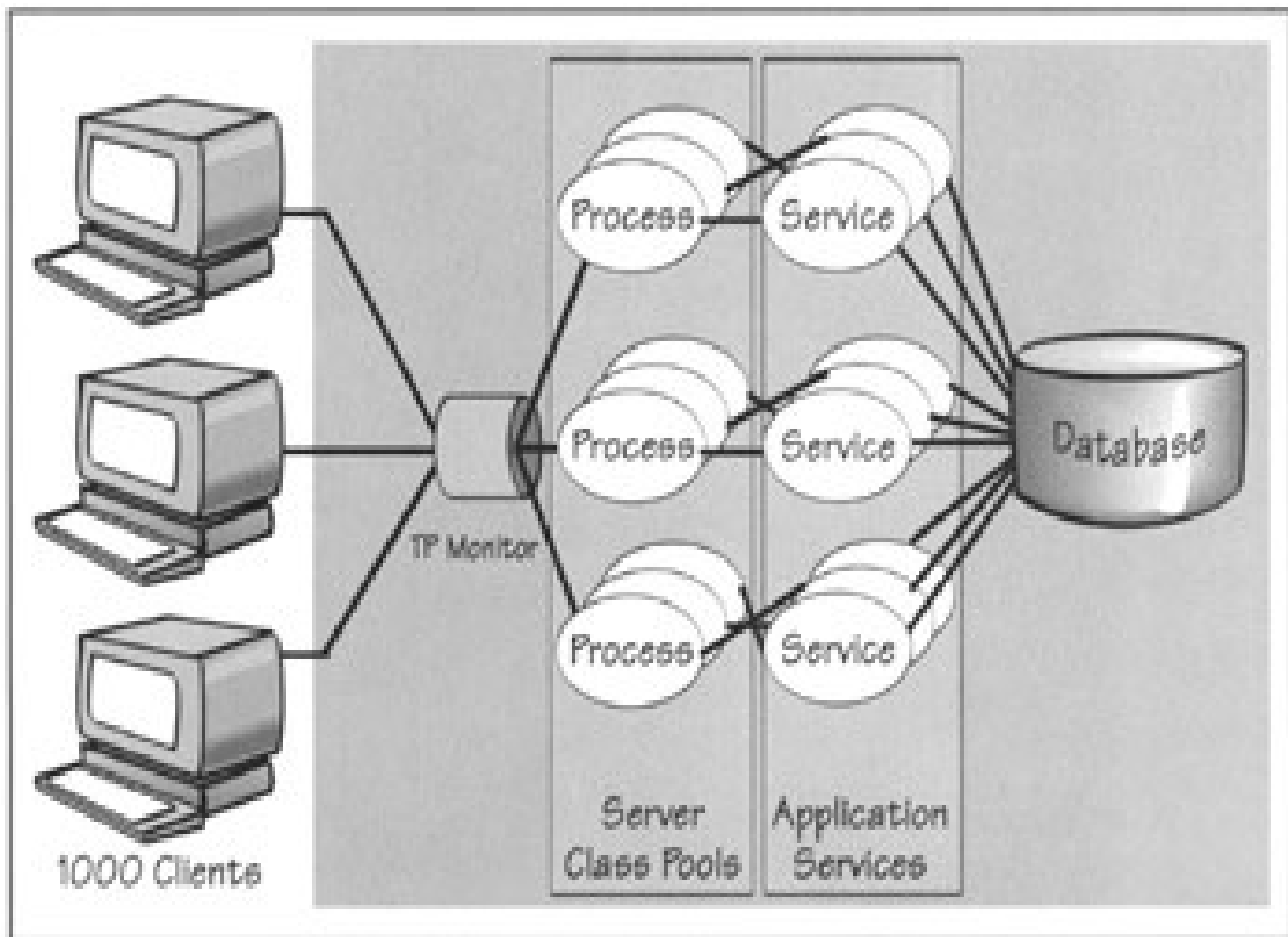
- The TP Monitor acts as **middleware** between:
- **Clients (presentation tier)** and
- **Back-end systems (databases, legacy systems)**
- Within this **middle tier**, it offers a runtime environment for server-side components.

## **TP Monitor does three things extremely well** (Services provided to middle-tier applications):

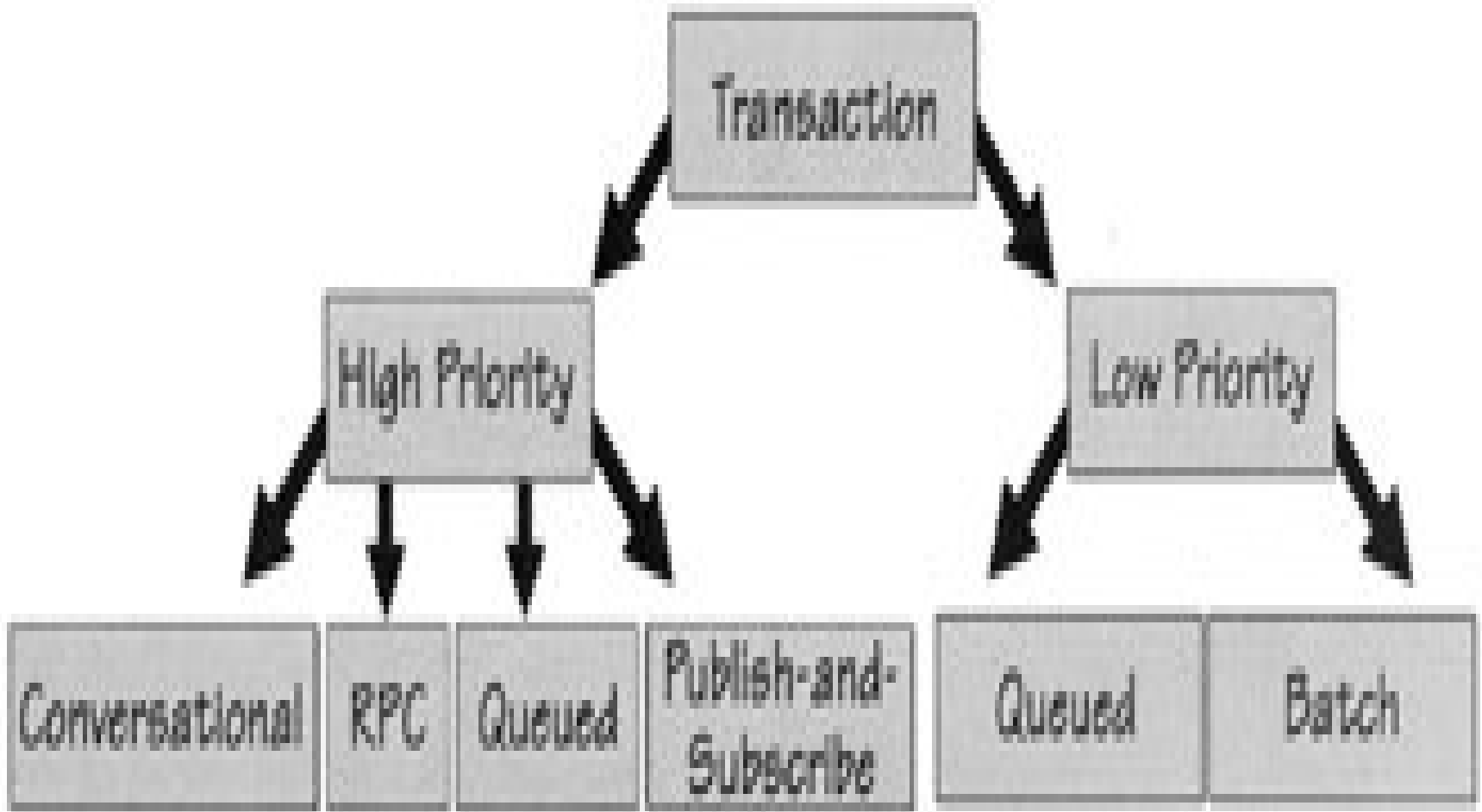
- **Process management** includes starting server processes, funneling work to them, monitoring their execution, and balancing their workloads.
- **Transaction management** means that it guarantees the ACID properties to all programs that run under its protection.
- **Client/server communications management** allows clients (and services) to invoke an application component in a variety of ways—including request-response, conversations, queuing, publish-and-subscribe, or broadcast.

# Why OS needs TP monitor and how is the relationship managed?





# TP Monitor Client/Server Interaction Types



# Transactional Versus Non-Transactional Communications

Feature	Ordinary Queues, RPCs, ORBs, Publish-and-Subscribe, and Conversational Communications	Transactional Queues, RPCs, ORBs, Publish-and-Subscribe, and Conversational Communications
Participants	Loosely-coupled client/server programs.	Transactionally bound client/server and server/server programs. The message invocation causes the recipient program to join the transaction.
Commit synchronization	No	Yes
Only-once semantics	No	Yes
Server management on the recipient node	No. It's just a delivery mechanism. (ORBs provide minimal server management.)	Yes. The process that receives the message is started, load balanced, monitored, and tracked as part of the transaction.
Load balancing	Using the directory services. The first server to register becomes a hotspot. No dynamic load balancing is provided.	Uses the TP Monitor's sophisticated load-balancing algorithms. Can spread work across multiple SMP machines and dynamically add more processes to cover hotspots of activity. Several servers can read from the same queue.
Supervised exchanges	No. Exchanges are simply between the client and the server. The exchanges are transient. No crash recovery or error management is provided. You're on your own.	The TP Monitor supervises the entire exchange, restarts communication links, redirects messages to an alternate server process if the first one gets hung, performs retries, and provides persistent queues and crash recovery.

# TP Monitor Benefits

- Client/Server application development framework
- Firewalls of protection
- High availability
- Load balancing
- MOM integration
- Scalability of function
- Reduced system cost.

# TP monitors and ORBs

ORBs bring TP Monitors to the client/server mainstream

- they will become the orchestrators of the Object Web
  - the next-generation infrastructure that combines Web and object technologies to support Internet, intranet, and extranet application
- ORBs provide TP Monitors with a myriad of standard services
  - including metadata, dynamic invocations, persistence, relationships, events, naming, component factories, versioning, licensing, security, change management, and collections.
- And objects also make it easier for TP Monitors to create and managed rich transaction models
  - such as nested transactions and long-lived transactions (or workflow).
- TP Monitors will become frameworks for managing objects packaged as transaction-savvy components
  - for example, CORBA/EJBs or server-side ActiveXs.

ORBs brought TP Monitor capabilities into mainstream client/server systems by **integrating object-oriented middleware with distributed transaction management.**

Traditional TP Monitor	ORB-based systems
Procedural, centralized	Procedural, centralized
Specialized environments	Specialized environments
Explicit transaction control	Explicit transaction control

# TP-Heavy vs TP-Lite

## TP-Lite.

- SQL database managers are also in the business of managing transactions across their own resources.
- Database transactions with stored procedures is all that's needed in the area of transaction management.
- TP-Lite is simply the integration of TP Monitor functions in the database engines.

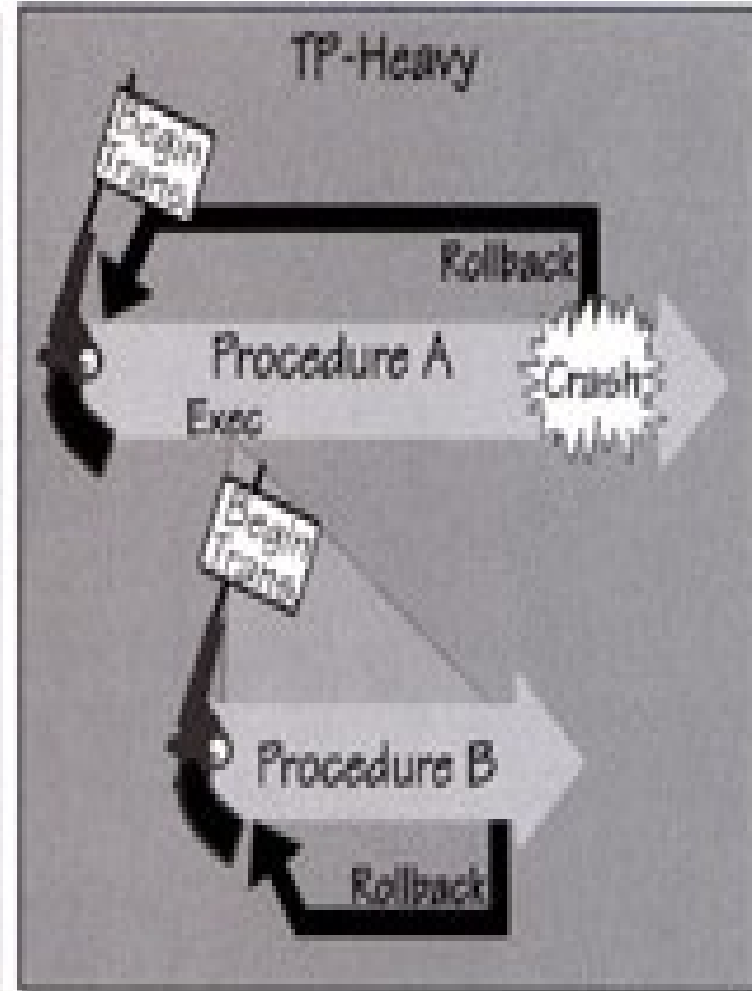
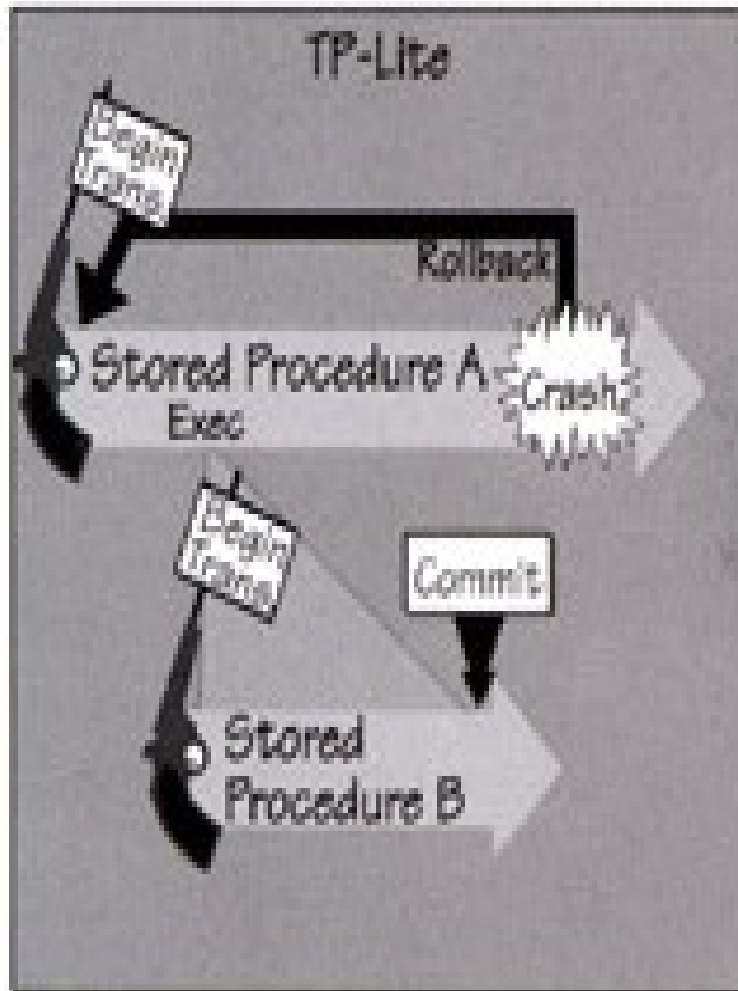
## TP-Heavy

- In other cases, TP Monitors extend the notion of transactions to all resources, not just data-centric ones
- TP Monitors track the execution of functions on a single server or across servers on the network

Majority of the client/server world is TP-Less

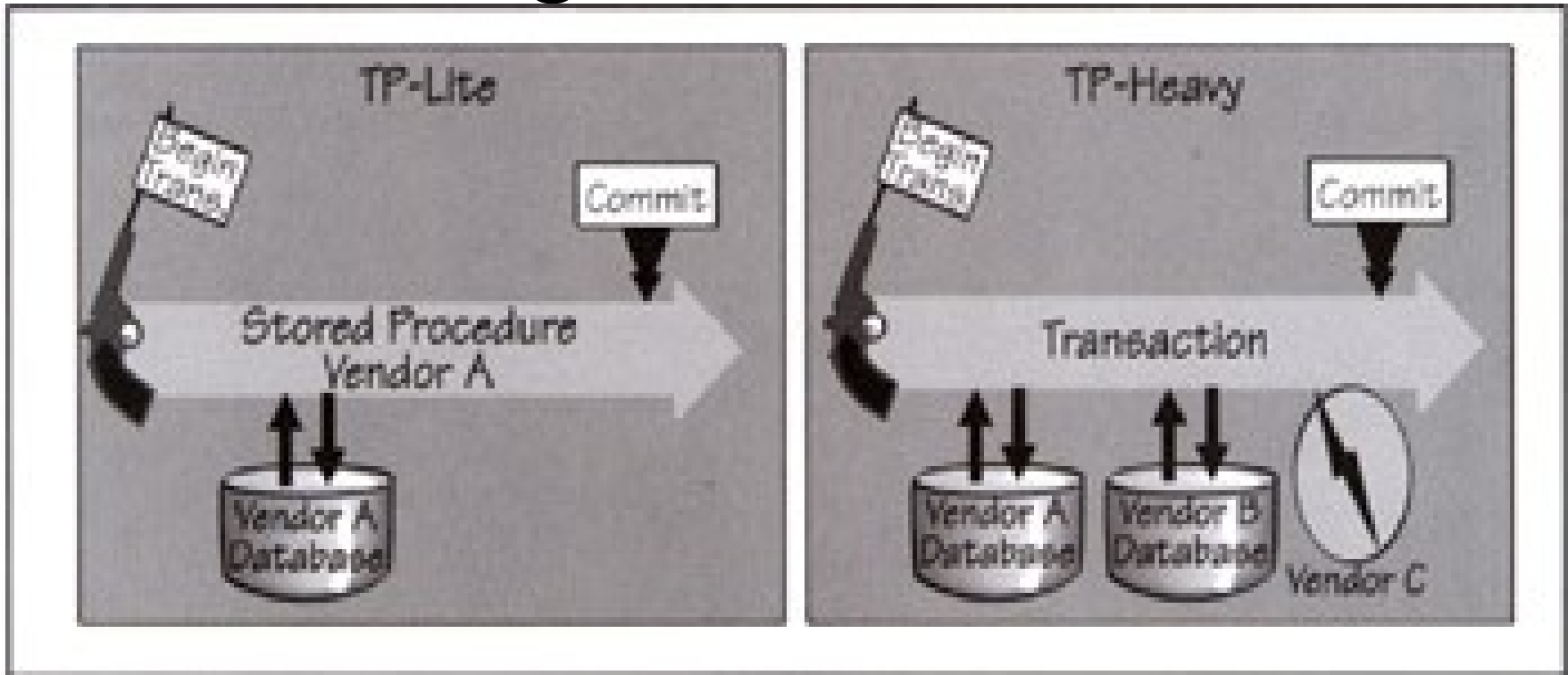
- All these TP Monitors support the client/server architecture and allow PCs to initiate some very complex multiserver transactions from the desktop.
- TP-Heavy includes process management, load balancing, global transaction synchronization, interfaces to multiple resource managers, and error recovery.

# TP-Lite Versus TP-Heavy: Scope of the Commit



- A TP-Lite stored procedure is written in a database-vendor proprietary procedural language—PL/SQL, Transact SQL, and so on—and is stored in the database
- If stored procedure A dies after invoking stored procedure B, A's work will automatically get rolled back while B's work is committed for posterity.
- This is a violation of the ACID all-or-nothing proposition.

# TP-Lite Versus TP-Heavy: Managing Heterogeneous Resources

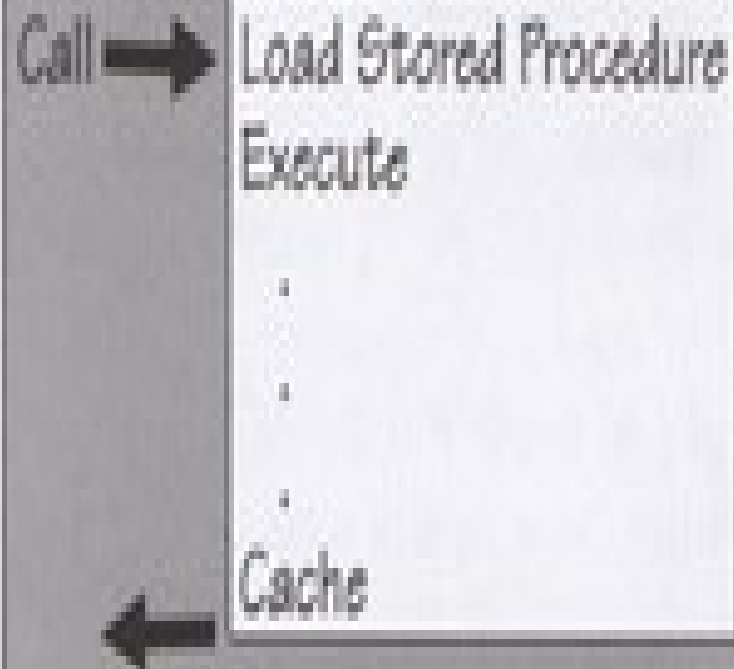


- A TP-Lite stored procedure can only commit transaction resources that are on the vendor's database or resource manager
- cannot synchronize or commit work that is on a foreign database or resource manager—whether local or remote

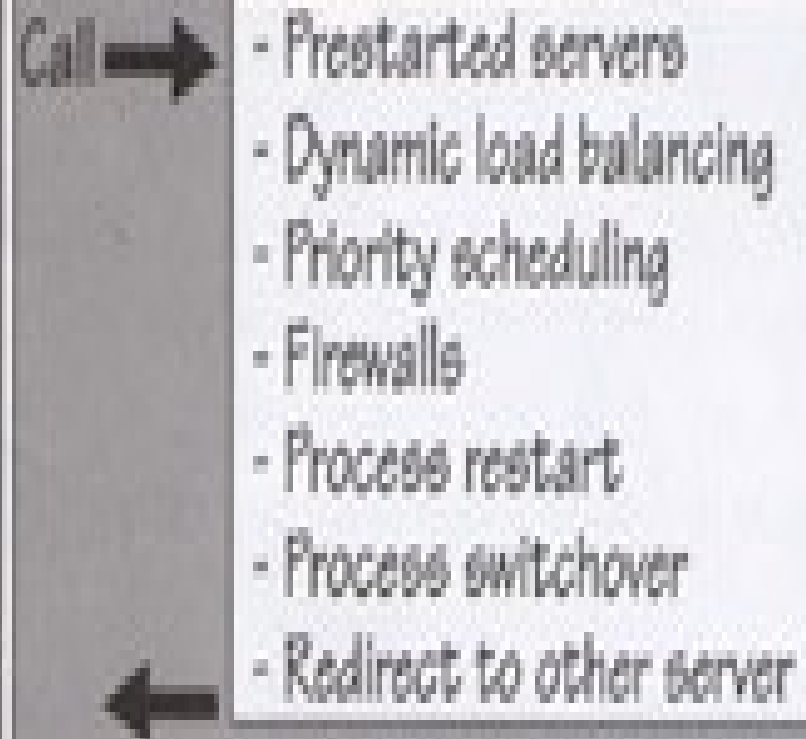
# TP-Lite Versus TP-Heavy: Process Management

- A TP-Lite stored procedure gets invoked, executed under ACID protection (within a single-phase commit), and may then be cached in memory for future reuse
- In contrast, TP-Heavy processes are prestarted and managed as server classes

### TP-Lite

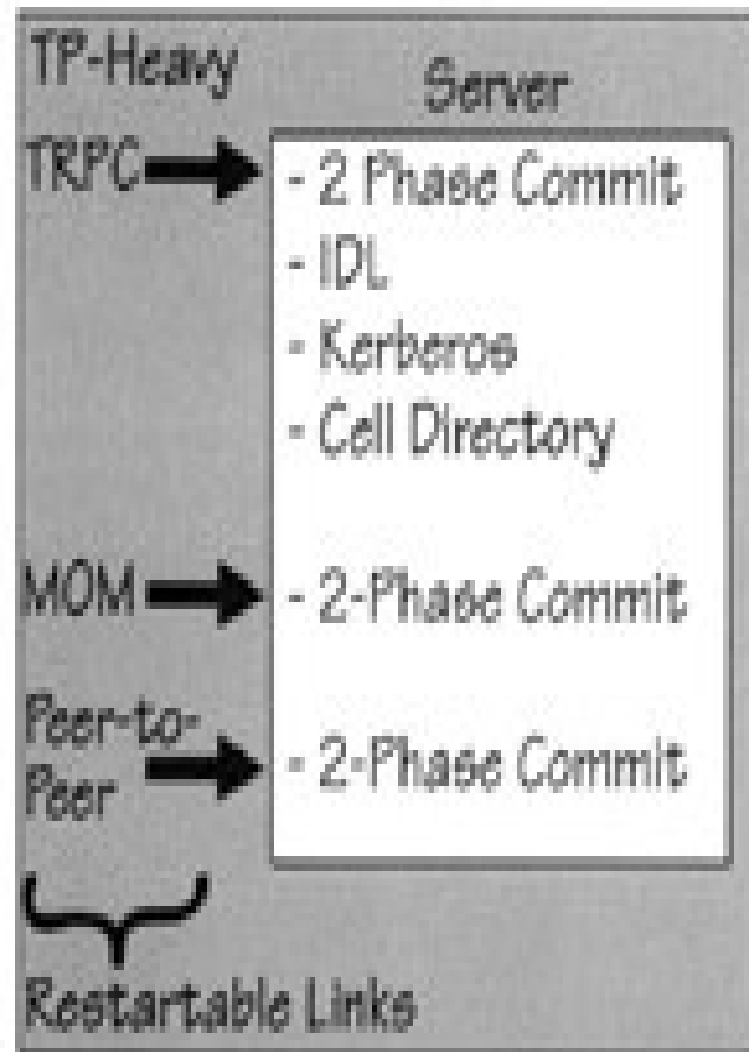
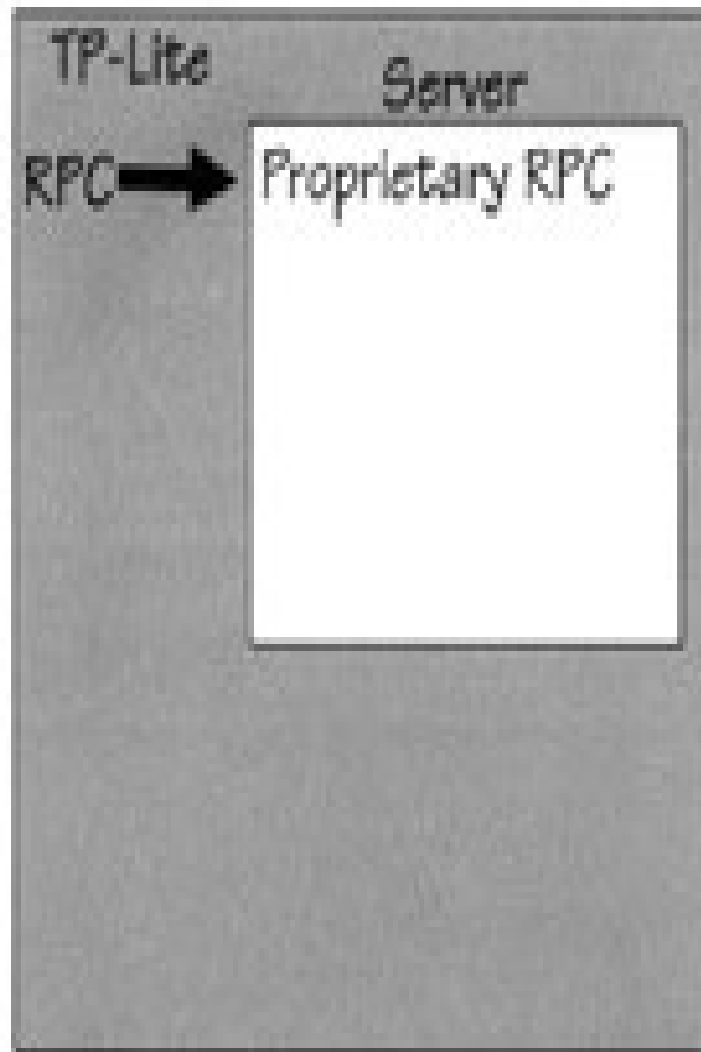


### TP-Heavy



# TP-Lite Versus TP-Heavy: Client/Server Invocations

- The TP-Lite stored procedure invocation is **extremely non-standard**. Vendors provide their own proprietary RPC invocation mechanism.
- TP-Lite does not support communications alternatives like conversations, queues, or publish-and-subscribe
- **TP-Heavy environment is very open to different communication styles**

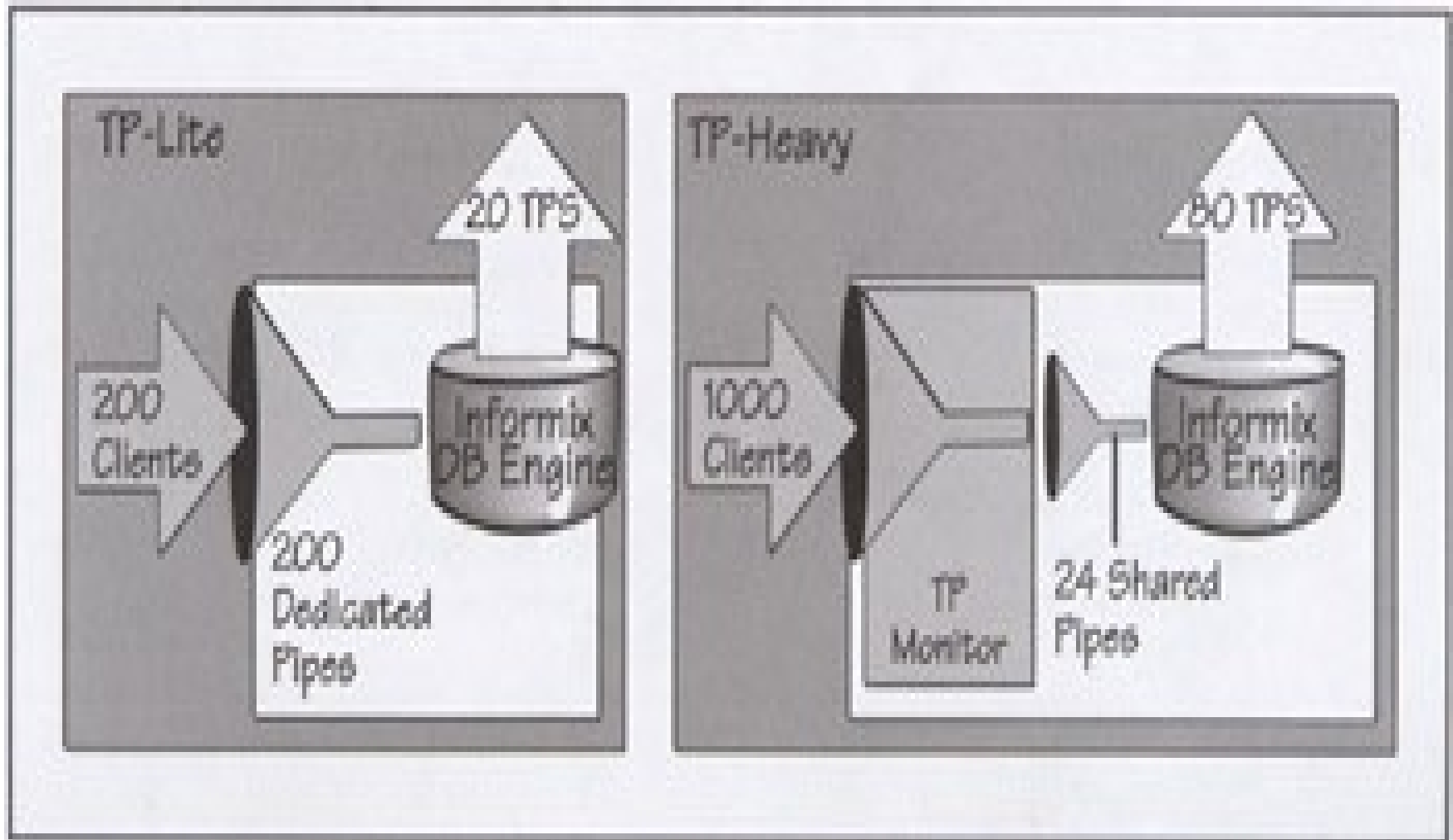


# **TP-Lite Versus TP-Heavy: Performance**

- **TP-Lite stored procedures are much faster than networked static or dynamic SQL**
- **They don't perform as well as TP-Heavy managed procedures, especially under heavy loads**
- **Most stored procedures dynamically interpret each SQL statement for each transaction and then recreate the access plan.**
- **In addition, most stored procedures are written using interpreted 4GLs, which are slow**

# TP-Lite Versus TP-Heavy:

## Resources & Cost








- **200 clients**
- Each client has a **dedicated pipe/connection** to the database
- **No TP Monitor**
- Clients communicate **directly with the Informix DB Engine**
- Throughput: **20 TPS (Transactions Per Second)**
- **Problems in TP-Lite:**
  - One connection per client → **resource-heavy**
  - Large number of open pipes → **high overhead**
  - Poor scalability as clients increase
  - Database becomes a **bottleneck**
  - TP-Lite works **for small systems but does not scale well.**

## TP-Heavy (Right Side)

### What happens here:


- **1000 clients**
- Clients connect to a **TP Monitor**
- TP Monitor uses only **24 shared pipes** to the database
- Throughput: **80 TPS**
- Database access is **controlled and optimized**

### Advantages of TP-Heavy:

-  Connection pooling (shared pipes)
-  Reduced database load
-  Better CPU and memory utilization
-  High scalability
-  Better performance even with more clients

### In short:

- TP-Heavy supports large-scale enterprise systems.



# THANK YOU