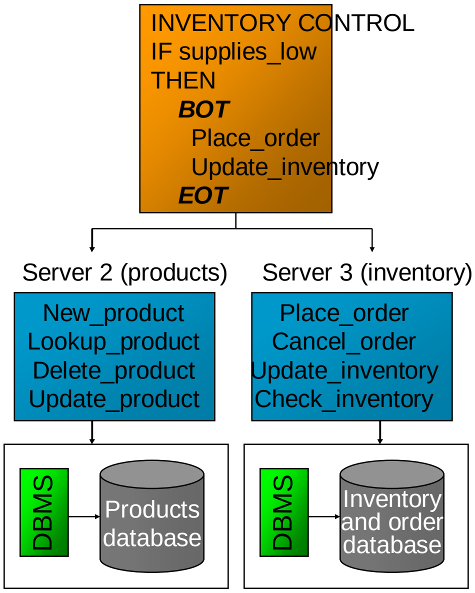
Chapter 2:
Transaction Processing Monitors (TP-monitors)

Gustavo Alonso
Computer Science Department
Swiss Federal Institute of Technology (ETHZ)
alonso@inf.ethz.ch
http://www.iks.inf.ethz.ch/

Outline

- Historical perspective:
  - The problem: synchronization and atomic interaction
  - The solution: transactional RPC and additional support
- TP Monitors
  - Example and Functionality
  - Architectures
  - Structure
  - Components
- TP Monitor functionality in CORBA

Client, server, and databases



```
INVENTORY CONTROL
IF supplies_low
THEN
    BOT
      Place_order
      Update_inventory
    EOT
```

Server 2 (products)

New_product
Lookup_product
Delete_product
Update_product

DBMS → Products database

Server 3 (inventory)

Place_order
Cancel_order
Update_inventory
Check_inventory

DBMS → Inventory and order database

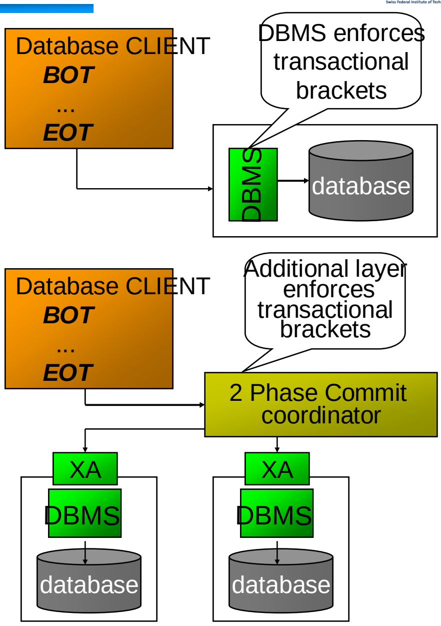
- Processing, storing, accessing and retrieving data has always been one of the key aspects of enterprise computing. Most of this data resides in relational database management systems, which have well defined interfaces and provided very clear guarantees to the operations performed over the data.
- However:
  - not all the data can reside in the same database
  - the application is built on top of the database. The guarantees provided by the database need to be understood by the application running on top

The nice thing about databases ...

- … is that they take care of all aspects related to data management, from physical storage to concurrency control and recovery
- Using a database can reduce the amount of code necessary in a large application by about 40 %
- From a client/server perspective, the databases help in:
  - concurrency control: many servers can be connected in parallel to the same database and the database will still have correct data
  - recovery: if a server fails in the middle of an operation, the database makes sure this does not affect the data or other servers
- Unfortunately, these properties are provided only to operations performed within the database. In principle, they do not apply when:
  - An operation spawns several databases
  - the operations access data not in the database (e.g., in the server)
- To help with this problem, the Distributed Transaction processing Model was created by X/Open (a standard's body). The heart of this model is the XA interface for 2 Phase Commit, which can be used to ensure that an operation spawning several databases enjoy the same atomicity properties as if it were executed in one database.

One at a time interaction

- Databases follow a single thread execution model where a client can only have one outstanding call to one and only one server at any time. The basic idea is one call per process (thread).
- Databases provide no mechanism to bundle together several requests into a single work unit
- The XA interface solves this problem for databases by providing an interface that supports a 2 Phase Commit protocol. However, without any further support, the client becomes the one responsible for running the protocol which is highly impractical
- An intermediate layer is needed to run the 2PC protocol


Database CLIENT
BOT
...
EOT
DBMS enforces transactional brackets
DBMS
database
Database CLIENT
BOT
...
EOT
Additional layer enforces transactional brackets
2 Phase Commit coordinator
XA
DBMS
database
XA
DBMS
database

2 Phase Commit

BASIC 2PC

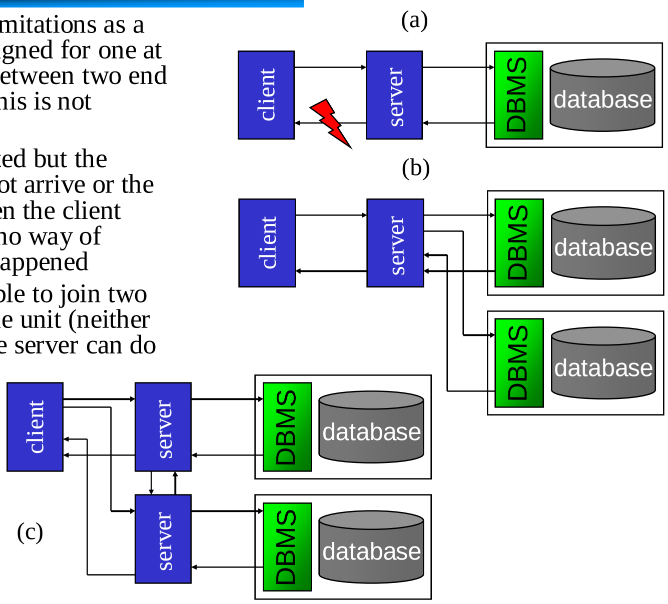
- Coordinator send PREPARE to all participants.
- Upon receiving a PREPARE message, a participant sends a message with YES or NO (if the vote is NO, the participant aborts the transaction and stops).
- Coordinator collects all votes:
  - All YES = Commit and send COMMIT to all others.
  - Some NO = Abort and send ABORT to all which voted YES.
- A participant receiving COMMIT or ABORT messages from the coordinator decides accordingly and stops.

What is needed to run 2PC?

- Control of Participants: A transaction may involve many resource managers, somebody has to keep track of which ones have participated in the execution
- Preserving Transactional Context: During a transaction, a participant may be invoked several times on behalf of the same transaction. The resource manager must keep track of calls and be able to identify which ones belong to the same transaction by using a transaction identifier in all invocations
- Transactional Protocols: somebody acting as the coordinator in the 2PC protocol
- Make sure the participants understand the protocol (this is what the XA interface is for)

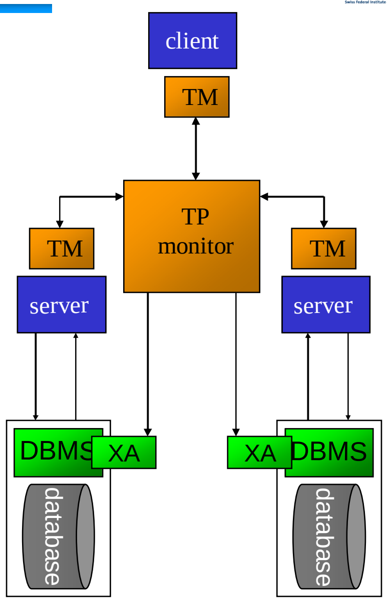
Interactions through RPC

- RPC has the same limitations as a database: it was designed for one at a time interactions between two end points. In practice, this is not enough:
  - a) the call is executed but the response does not arrive or the client fails. When the client recovers, it has no way of knowing what happened
  - b) c) it is not possible to join two calls into a single unit (neither the client nor the server can do this)

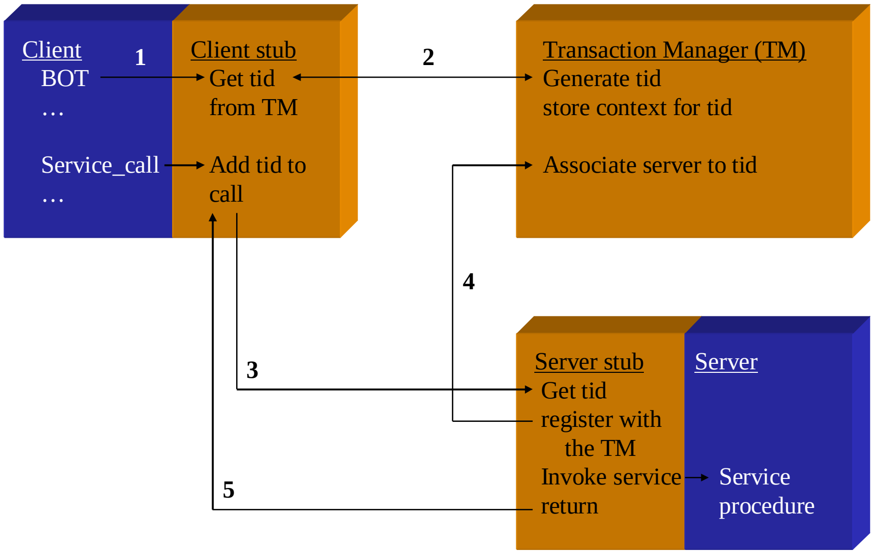


Transactional RPC

- The limitations of RPC can be resolved by making RPC calls transactional. In practice, this means that they are controlled by a 2PC protocol
- As before, an intermediate entity is needed to run 2PC (the client and server could do this themselves but it is neither practical nor generic enough)
- This intermediate entity is usually called a transaction manager (TM) and acts as intermediary in all interactions between clients, servers, and resource managers
- When all the services needed to support RPC, transactional RPC, and additional features are added to the intermediate layer, the result is a TP-Monitor



Basic TRPC (making calls)



Client
BOT
…
Service_call
…

1

Client stub
Get tid from TM
Add tid to call

2

Transaction Manager (TM)
Generate tid
store context for tid
Associate server to tid

4

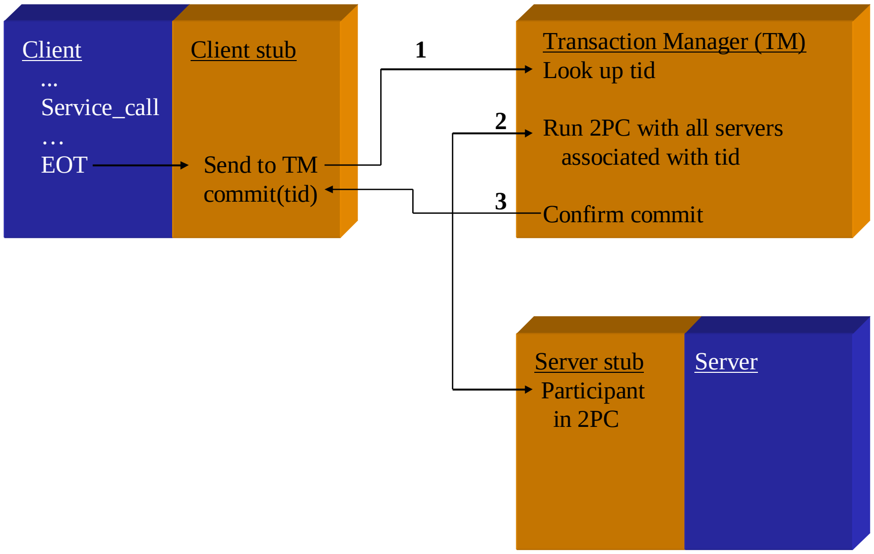
3

5

Server stub
Get tid
register with the TM
Invoke service
return

Server
Service procedure

Basic TRPC (committing calls)



Client
…
Service_call
…
EOT

Client stub
Send to TM commit(tid)

1

2

3

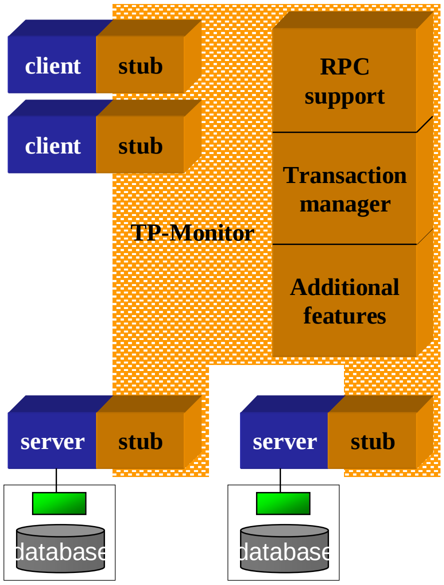
Transaction Manager (TM)
Look up tid
Run 2PC with all servers associated with tid
Confirm commit

Server stub
Participant in 2PC

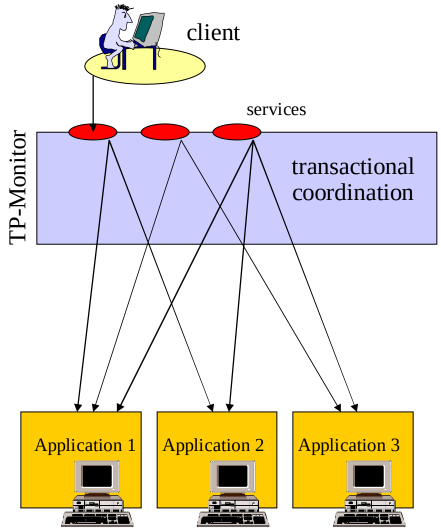
Server

One step beyond ...

- The previous example assumes the server is transactional and can run 2PC. This could be, for instance, a stored procedure interface within a database. However, this is not the usual model
- Typically, the server invokes a resource manager (e.g., a database) that is the one actually running the transaction
- This makes the interaction more complicated as it adds more participants but the basic concept is the same:
    - the server registers the resource manager(s) it uses
    - the TM runs 2PC with those resources managers instead of with the server (see OTS at the end)



TP-Monitors = transactional RPC

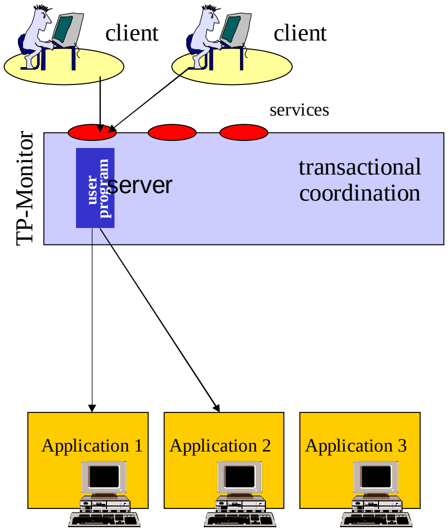


- A TP-Monitor allows building a common interface to several applications while maintaining or adding transactional properties. Examples: CICS, Tuxedo, Encina.
- A TP-Monitor extends the transactional capabilities of a database beyond the database domain. It provides the mechanisms and tools necessary to build applications in which transactional guarantees are provided.
- TP-Monitors are, perhaps, the best, oldest, and most complex example of middleware. Some even try to act as distributed operating systems providing file systems, communications, security controls, etc.
- TP-Monitors have traditionally been associated to the mainframe world. Their functionality, however, has long since migrated to other environments and has been incorporated into most middleware tools.

TP-Monitor functionality

- TP-Monitors appeared because operating systems are not suited for transactional processing. TP-Monitors are built as operating systems on top of operating systems.
- As a result, TP-Monitor functionality is not well defined and very much system dependent.
- A TP-Monitor tries to cover the deficiencies of existing “all purpose” systems. What it does is determined by the systems it tries to ”improve”.
- A TP-Monitor is basically an integration tool. It allows system designers to tie together heterogeneous system components using a number of utilities that can be mixed and matched depending on the particular characteristics of each case.
- Using the tools provided by the TP-Monitor, the integration effort becomes more straightforward as most of the needed functionality is directly supported by the TP-Monitor.
- A TP-Monitor addresses the problems of sharing data from heterogeneous, distributed sources, providing clean interfaces and ensuring ACID properties.
- A TP-Monitor extrapolates the functions of a transaction manager (locking, scheduling, logging, recovery) and controls the distributed execution. As such, TP-Monitor functionality is at the core of the integration efforts of many software producers (databases, workflow systems, CORBA providers, …).
- A TP-Monitor also controls and manages distributed computations. It performs load balancing, monitoring of components, starting and finishing components as needed, routing of requests, recovery of components, logging of all operations, assignment of priorities, scheduling, etc. In many cases it has its own transactional file system, becoming almost indistinguishable from a distributed operating system.

Transactional properties



- The TP-monitor tries to encapsulate the services provided within transactional brackets. This implies RPC augmented with:
  - atomicity: a service that produces modifications in several components should be executed entirely and correctly in each component or should not be executed at all (in any of the components).
  - isolation: if several clients request the same service at the same time and access the same data, the overall result will be as if they were alone in the system.
  - consistency: a service is correct when executed in its entirety (it does not introduce false or incorrect data into the component databases)
  - durability: the system keeps track of what has been done and is capable of redoing and undoing changes in case of failures.

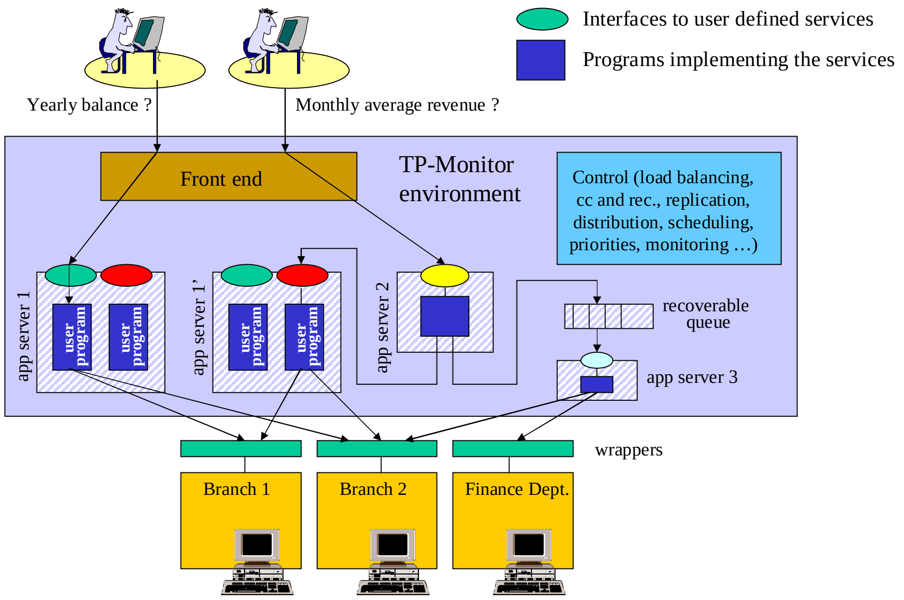
TRAN-C (Encina)

```
# include <tc/tc.h>
inModule(“helloWorld”);

void Main () {
   int i;
   inFunction(“main”);
   initTC();                    /* initializes transaction manager */

   transaction {                /* starts a transaction */
            printf(“Hello World - transaction %d\n”, getTid());
            if (I % 2) abort (“Odd transactions are aborted”);
            }
   onCommit
            printf(“Transaction Comitted”);
   onAbort
            printf(“Abort in module: %s\n \t %s\n”, abortModuleNAme(), abortReason());
}
```

TP-Monitor, generic architecture



Tasks of a TP Monitor

Core services

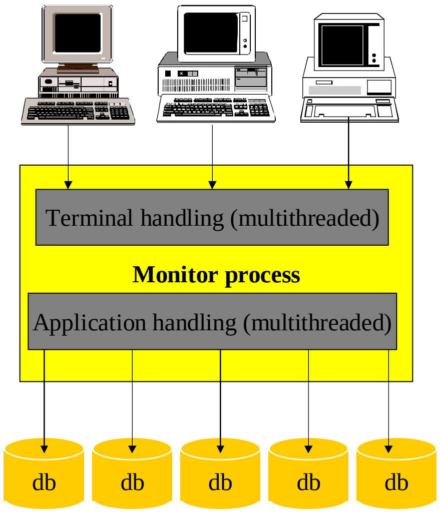
- Transactional RPC: Implements RPC and enforces transactional semantics, scheduling operations accordingly
- Transaction manager: runs 2PC and takes care of recovery operations
- Log manager: records all changes done by transactions so that a consistent version of the system can be reconstructed in case of failures
- Lock manager: a generic mechanism to regulate access to shared data outside the resource managers

Additional services

- Server monitoring and administration: starting, stopping and monitoring servers; load balancing
- Authentication and authorization: checking that a user can invoke a given service from a given terminal, at a given time, on a given object and with a given set of parameters (the OS does not do this)
- Data storage: in the form of a transactional file system
- Transactional queues: for asynchronous interaction between components
- Booting, system recovery, and other administrative chores

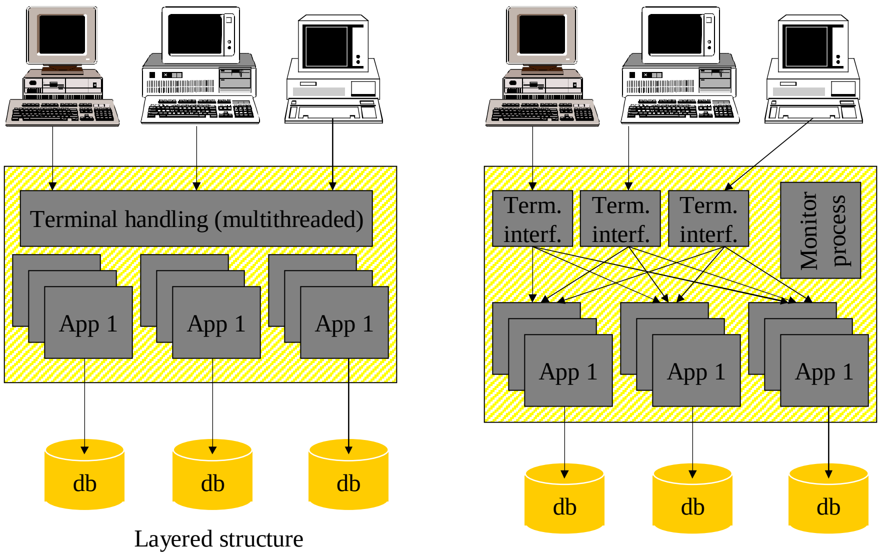
Structure of TP-Monitors (I)

- TP-Monitors try in many aspects to replace the operating system so as to provide more efficient transactional properties. Depending what type of operating system they try to replace, they have a different structure:
  - Monolithic: all the functionality of the TP-Monitor is implemented within one single process. The design is simpler (the process can control everything) but restrictive (bottleneck, single point of failure, must support all possible protocols in one single place).
  - Layered: the functionality is divided in two layers. One for terminal handling and several processes for interaction with the resource managers. The design is still simple but provides better performance and resilience.
  - Multiprocessor: the functionality is divided among many independent, distributed processes.


Terminal handling (multithreaded)
Monitor process
Application handling (multithreaded)
db
db
db
db
db

Monolithic structure

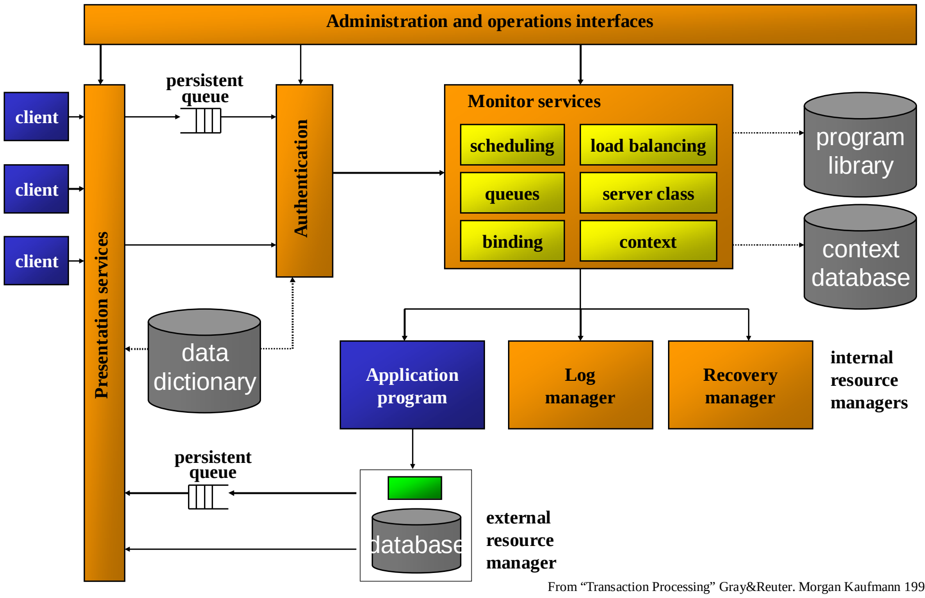
Structure of TP-Monitors (II)



Layered structure

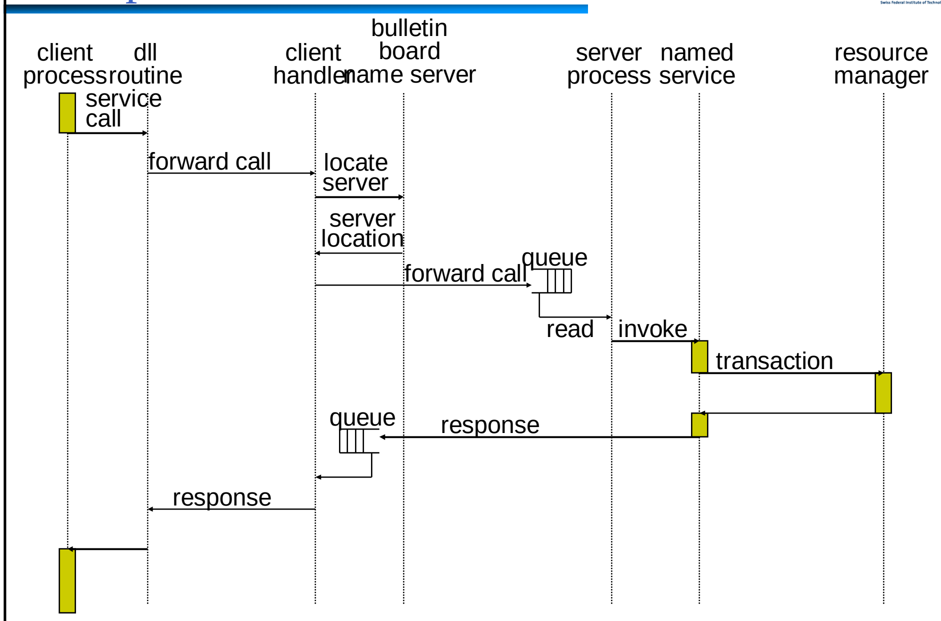
Multiprocessor structure

TP-Monitor components (generic)



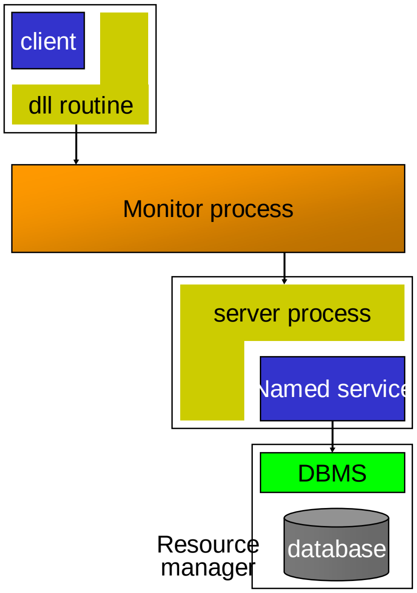
From “Transaction Processing” Gray&Reuter. Morgan Kaufmann 1993

Example: BEA Tuxedo



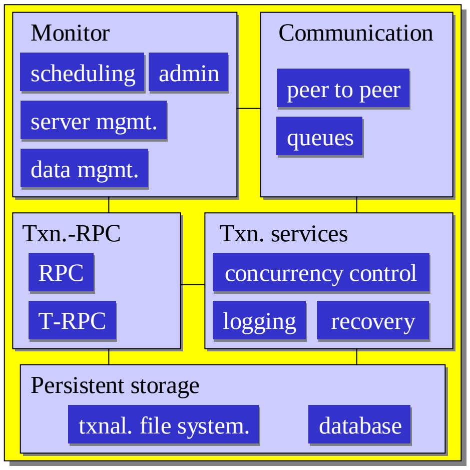
Example: BEA Tuxedo

- The client uses DLL (Dynamic Link Libraries) routines to interact with the TP-Monitor
- The Monitor Process or Tuxedo server implements all system services (name services, transaction management, load balancing, etc) and acts as the control point for all interactions
- Application services are known as named services. These named services interact with the system through a local server process
- Interaction across components is through message queues rather than direct calls (although clients and servers may interact synchronously)



TP-Monitor components (Encina)

- □ The current trend is towards a “family of products” instead of a single system. Each element can be used by itself (reduced footprint) and, in some cases, can be used completely independent of the TP-Monitor.
- □ Monitor: execution environment providing integrity, availability, security, fast response time and high throughput. It includes tools for administration and installation of components and the development environment.
- □ Communication services: protocols and mechanisms for persistent messages and peer to peer communication.
- □ Transactional RPC: basic interaction mechanism
- □ Transactional services: supporting concurrency control, recovery, logging and transactional programming. Behavior of the system can be tailored (advances transaction models, selective logging, ad-hoc recovery …)
- □ Persistent storage



External interfaces

With clients

- □ The main interface is through the presentation services. In old systems, presentation services included terminal handling and format control for presentation on a screen. Today, the presentation services are mostly interfaces to other systems that take care of data presentation (mainly web servers)
- □ The most important part of the presentation services still in use today is the RPC (TRPC) stubs and libraries used on the client side for invoking services implemented within the TP-Monitor

With administrators

- □ The TP-Monitor needs to be maintained and administered like any other system. Today there are a wide variety of tools for doing so. They include:
  - node monitoring
  - service monitoring
  - load monitoring
  - configuration tools
  - programming support
  - …
- □ Another important part of the interfaces to the system are the development environments which tend to be similar in nature to that of RPC systems

Monitor services

- Monitor services are those facilities that provide the basic functionality of the TP-Monitor. They can be implemented as part of the TP-Monitor process or as external resource managers
- Server class: each application program implementing services has a server class in the monitor. The server class starts and stops the application, creates message queues, monitors the load, etc. In general, it manages one application program
- Binding: acts as the name and directory services and offers similar functionality as the binder in RPC. It might be coupled with the load balancing service for better distribution
- Load balancing: tries to optimize the resources of the system by providing an accurate picture of the ongoing and scheduled work
- Context management: a key service in TRPC that is also used in keeping context across transaction boundaries or to store and forward data between different resource managers and servers
- Communication services (queue management and networking) are usually implemented as external resource managers. They take care of transactional queuing and any other aspect of message passing

Resource managers

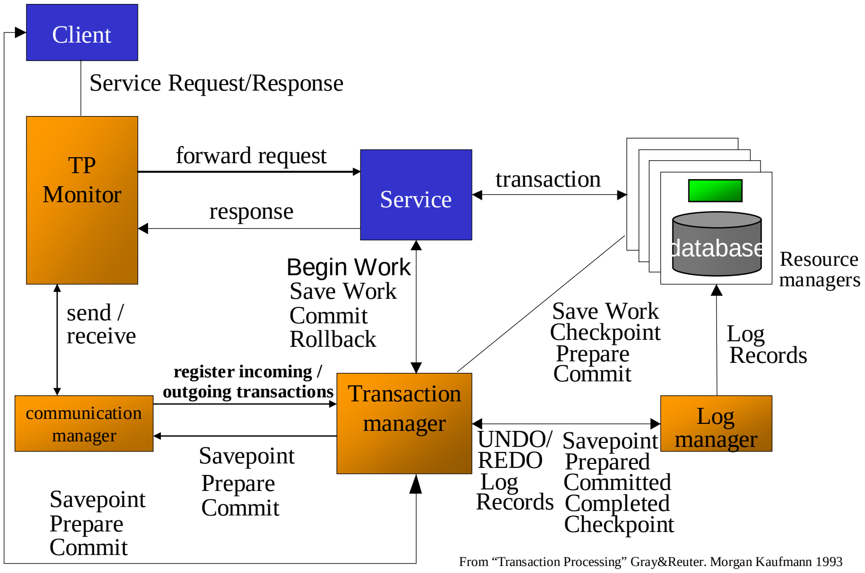
Internal Resource Managers

- These are modules that implement a particular service in the TP-Monitor. There are two kinds:
- Application programs: programs that implement a collection of services that can be invoked by the clients of the TP-Monitor. They define the application built upon the TP-Monitor
- Internal services: like logging, locking, recovery, or queuing. Implementing these services as resource managers gives more modularity to the system and even allows to use other systems for this purpose (like queue management systems)

External Resource Managers

- These are the systems the TP-Monitor has to integrate
- The typical resource manager is a database management system with an SQL/XA interface. It can also be a legacy application, in which case wrappers are needed to bridge the interface gap. A typical example are screen scraping modules that interact with mainframe based applications by posing as dumb terminals
- The number and type of external resource managers keeps growing and a resource manager can be another TP monitor.
- The WWW is slowly also becoming a resource manager

Transaction processing components



From "Transaction Processing" Gray&Reuter. Morgan Kaufmann 1993

TP-Monitors vs. OS

	processing	data	communication
TP Services	Admin interface Configuration tools Load balancing Programming tools	Databases Disaster recovery Resource managers Flow of control	Name server Server invocation Protected user interface
TP internal system services	Txn identifiers Server class Scheduling Authentication	Transaction manager Logs and context Durable queues Transactional files	Transactional RPC Transactional Sessions RPC
OS	Process – Threads Address space Scheduling Local naming protection	Repository File System Blocks, paging File security	IPC Simple sessions Naming Authentication
Hardware	CPU	Memory	Wires, switches

TP Monitor

From "Transaction Processing" Gray&Reuter. Morgan Kaufmann 1993

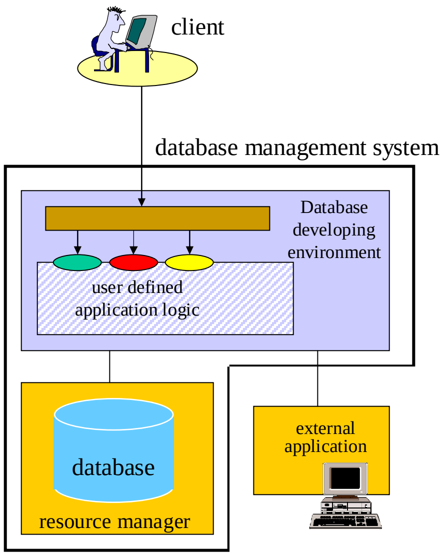
Advantages of TP-Monitors

- TP-Monitors are a development and run-time platform for distributed applications
- The separation between the monitor and the transaction manager was a practical consideration but turned out to be a significant advantage as many of the features provided by the monitor are as valuable as transactions
- The move towards more modular architectures prepared TP-Monitors for changes that had not been foreseen but turned be quite advantageous:
  - the web as the main interface to applications: the presentation services included an interface so that requests could be channeled through a web server
  - queuing as a form of middleware in itself (Message Oriented Middleware, MOM): once the queuing service was an internal resource manager, it was not too difficult to adapt the interface so that the TP-Monitor could talk with other queuing systems
  - Distributed object systems (e.g., CORBA) required only a small syntactic layer in the development tools and the presentation services so that services will appear as objects and TRPC would be come a method invocation to those objects.

TP-Heavy vs. TP-Light = 2 tier vs. 3 tier

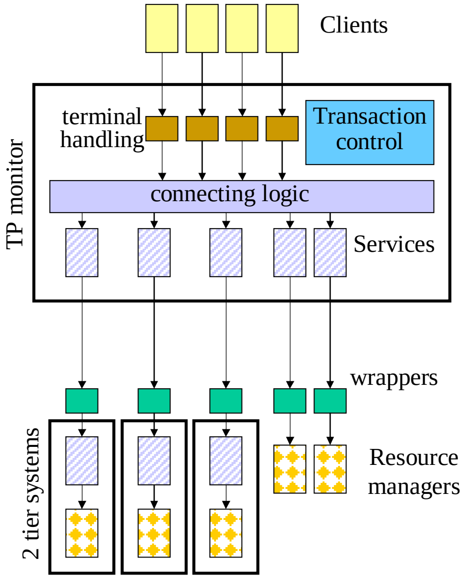
- A TP-heavy monitor provides:
  - a full development environment (programming tools, services, libraries, etc.),
  - additional services (persistent queues, communication tools, transactional services, priority scheduling, buffering),
  - support for authentication (of users and access rights to different services),
  - its own solutions for communication, replication, load balancing, storage management ... (most of the functionality of an operating system).
- Its main purpose is to provide an execution environment for resource managers (applications), and do all this with guaranteed reasonable performance (e.g., > 1000 txns. per second).
- This is the traditional monitor: CICS, Encina, Tuxedo.
- A TP-Light is an extension to a database:
  - it is implemented as threads, instead of processes,
  - it is based on stored procedures ("methods" stored in the database that perform an specific set of operations) and triggers,
  - it does not provide a development environment.
- Light Monitors are appearing as databases become more sophisticated and provide more services, such as integrating part of the functionality of a TP-Monitor within the database.
- Instead of writing a complex query, the query is implemented as a stored procedure. A client, instead of running the query, invokes the stored procedure.
- Stored procedure languages: Sybase's Transact-SQL, Oracle's PL/SQL.

TP-light: databases and the 2 tier approach


client
database management system
Database developing environment
user defined application logic
database
resource manager
external application

- Databases are traditionally used to manage data.
- However, simply managing data is not an end in itself. One manages data because it has some concrete application logic in mind. This is often forgotten when considering databases (specially benchmarking) and has allowed SAP to take over a significant market share before any other vendors reacted.
- But if the application logic is what matters, why not move the application logic into the database? These is what many vendors are advocating. By doing this, they propose a 2 tier model with the database providing the tools necessary to implement complex application logic.
- These tools include triggers, replication, stored procedures, queuing systems, standard access interfaces (ODBC, JDBC) .. which are already in place in many databases.

TP-Heavy: 3-tier middleware

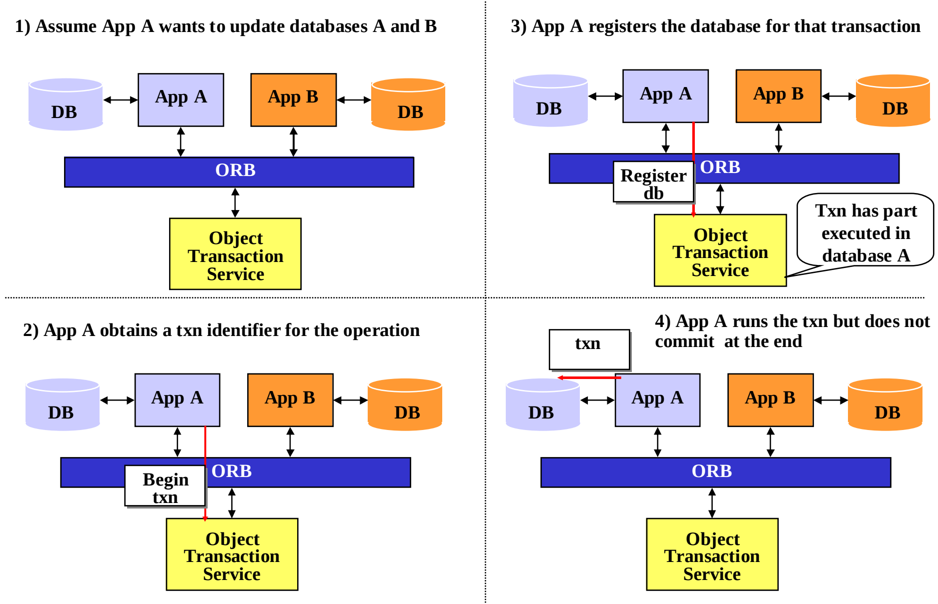


- TP-heavy are middleware platforms for developing 3-tier architectures. They provide all the functionality necessary for such an architecture to work.
- A system designer only need to program the services (which will run within the scope of the TP-Monitor; the services are linked to a number of TP libraries providing the needed functionality), the wrappers (if they are not already provided), and the clients. The TP-Monitors takes these components and embeds them within the overall system as interconnected components.
- The TP-Monitor provides the infrastructure for the components to work and the tools necessary to build services, wrappers and clients. In some cases, it provides even its own programming language (e.g., Transational-C of Encina).

Object Transaction Service

- An OTS provides transactional guarantees to the execution of invocations between different components of a distributed application built on top of an ORB. It is part of the CORBA standard It is identical to a basic TP-Monitor
- There are two ways to trace calls:
  - Explicit (manual): the invocation itself contains the transaction identifier. Then, when the application registers the resource manager, it uses this transaction identifier to say to which transaction it is "subscribing"
  - Implicit (automatic): the call is made through the OTS, which will forward the transaction identifier along with the invocation. This requires to link with the OTS library and to make all methods involved transactional
- ... and two ways to register resources (necessary in order to tell the OTS who will participate in the 2PC protocol and what type of interface is supported)
- Manual registration implies the the user provides an implementation of the resource. This implementation acts as an intermediary between the OTS and the actual resource manager (useful for legacy applications that need to be wrapped)
- Automatic registration is used when the resource manager understands transactions (i.e., it is a database), in which case it will support the XA interface for 2PC directly. A resource are registered only once, and implicit propagation is used to check which transactions go there

Running a distributed transaction (1)



1) Assume App A wants to update databases A and B

DB
App A
App B
DB
ORB
Object Transaction Service

2) App A obtains a txn identifier for the operation

DB
App A
App B
DB
Begin txn
ORB
Object Transaction Service

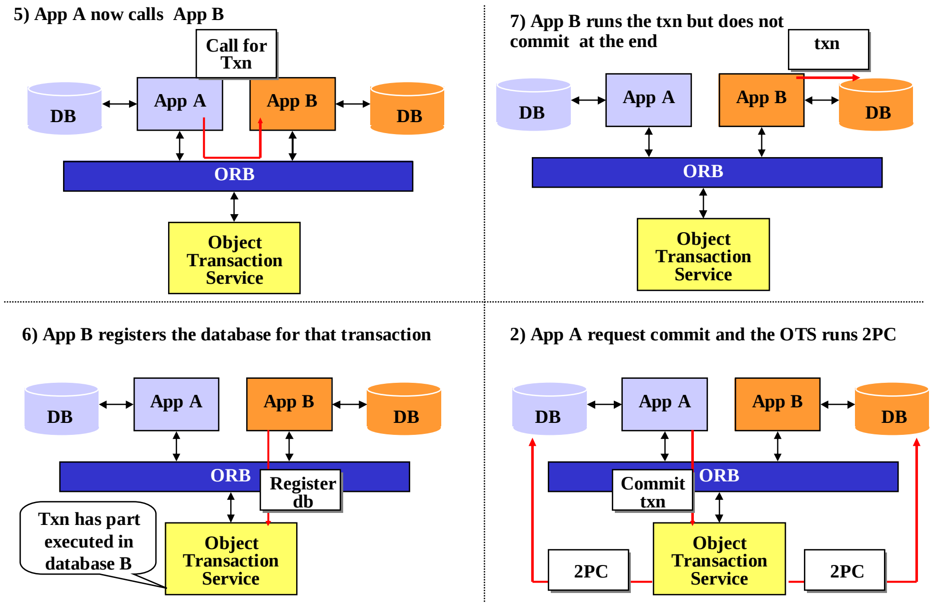
3) App A registers the database for that transaction

DB
App A
App B
DB
Register db
ORB
Object Transaction Service
Txn has part executed in database A

4) App A runs the txn but does not commit at the end

txn
DB
App A
App B
DB
ORB
Object Transaction Service

Running a distributed transaction (2)


5) App A now calls App B
Call for Txn
DB
App A
App B
DB
ORB
Object Transaction Service
7) App B runs the txn but does not commit at the end
txn
DB
App A
App B
DB
ORB
Object Transaction Service
6) App B registers the database for that transaction
DB
App A
App B
DB
ORB
Register db
Txn has part executed in database B
Object Transaction Service
2) App A request commit and the OTS runs 2PC
DB
App A
App B
DB
Commit txn
ORB
2PC
Object Transaction Service
2PC

The future of TP-Monitors

- TP-Monitors are the best example of middleware and the most successful implementation both in terms of performance and functionality.
- Together with object brokers, TP-Monitors form the foundation of today’s distributed data management products. Enterprise Application Integration is still largely based on TP-Monitor technology.
- TP-Monitors are the main reference for implementing middleware:
  - in terms of performance, TP-Monitors are orders of magnitude ahead of other middleware systems
  - in terms of functionality, TP-Monitors offer a quite complete, well integrated platform that can be extended to provide the functionality needed in other middleware systems
- Unlike other forms of middleware, TP-Monitors have proven to be quite resilient in time: some product lines are almost 30 years old already. Although the technology changes, the answer to fundamental design problems is well understood in TP-Monitors. These expertise will still have a significant impact on any emerging form of middleware.