Distributed Transaction Management – 2003

Jyrki Nummenmaa
http://www.cs.uta.fi/~dtm
jyrki@cs.uta.fi

Adding new subtxns to a txn

Growing the txn

- Centralised commit protocols use a coordinator.
- The coordinator can be initiated when the txn starts.
- After that the sub-txns on each site are joined in the txn as the txn needs to access data on different servers.
- The coordinator knows all sub-txns.
- Before commit, the sub-txns can be locally aborted, so they do not need to know of the other txns.
- Remember that in a commit protocol the coordinator is to send a participant list to all participants (which can be used in termination and recovery).

How does the coordinator know where to grow the txn?

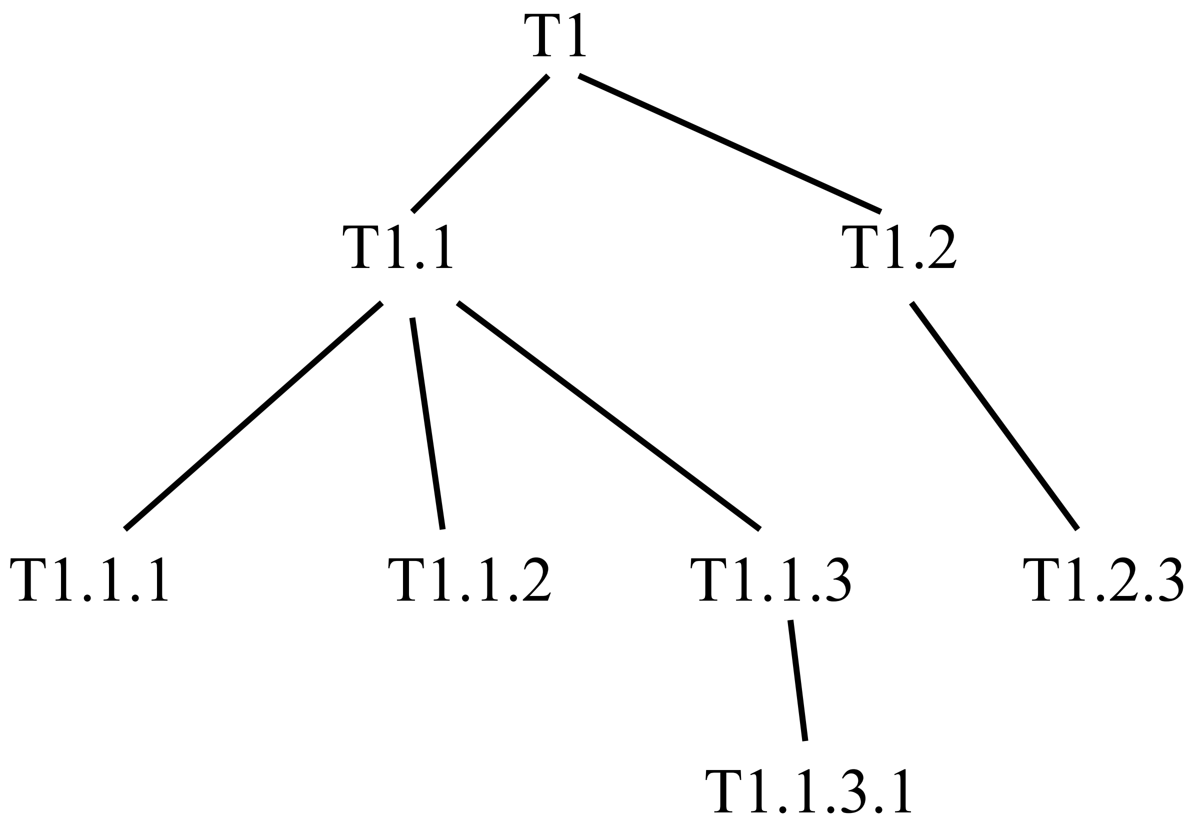
- There could be a dictionary, which tells, where data is replicated/situated.
- This could also be based on a hashing rule (Books from A to C are on Server 1, from D to E on Server 2, …)
- In a performance-critical system, there could be a dedicated server for this (with a continuously updated backup server).
- Sometimes it is also possible to replicate the distribution information to all sites (but this gives more distributed data to manage…)

Nested (Hierarchical) Transactions

Nested txns – what?

- In a nested txn, first a top-level txn is created.
- The top-level txn can create child txns.
- The child txns can create their child txns up to any level of nesting.
- The child txns are called subtxns.
- The child txn starts after its parent and finishes before its parent. (Its lifetime is completely included in the parent's lifetime.)
- Siblings can run concurrently.

Example

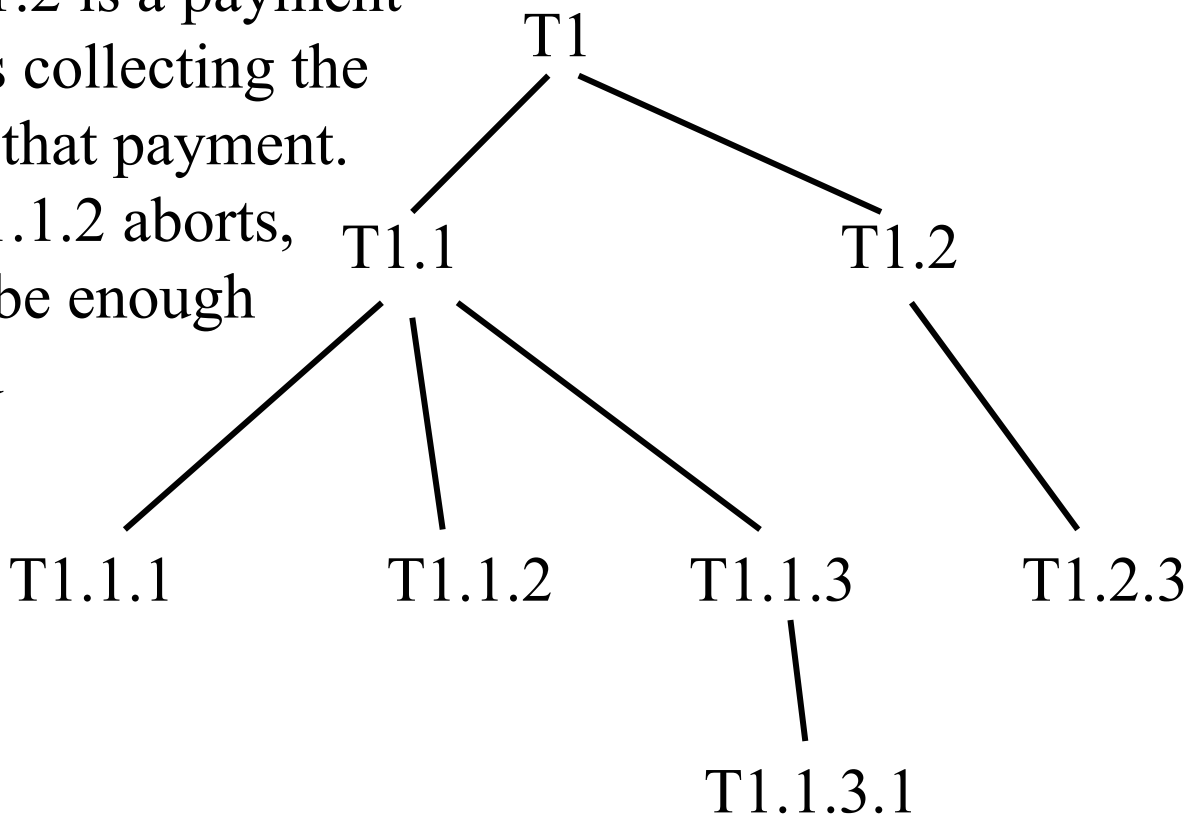


Provisional commit

- A child txn makes an independent commit decision: The txn may either decide to abort, in which case it is just rolled back, or it may decide to *provisionally* commit.
- Provisional commit means, that the subtxn is ready to do a final commit, if it is requested by the parent txn.
- If a subtxn informs its parent that it decides to abort, the parent does not necessarily need to abort – it adjusts its actions depending on the situation.

Example

Assume T1.2 is a payment and T1.1 is collecting the money for that payment. Even if T.1.1.2 aborts, there may be enough money and T1 may decide to commit.



Nested txn commit

- The parent tells the provisionally committed subtxns to commit, if it decides to commit.
- This should be done with 2PC, as the subtxns may have got into trouble and aborted.
- One way to do it is to use nested 2PC – the top txns coordinator sends VoteReq-commands to its child txns, they run a vote on their subtree and return the result.
- The Commit/Abort decisions are sent recursively down the tree.

Flat txn commit

- As an alternative, the subtxns may tell their parents their provisional commit decision with a list of the subtxn ids and their servers.
- Of course, they would not include the subtxns that are aborted.
- This way, the root txn will eventually get all provisionally committed txns and can perform a flat (normal) 2PC.

Locking in nested txns

- A basic solution would be to let all txns do their locking as separate txns.
- This simplifies correct serialisation.
- Another possibility is to treat the sub-txns as the same txn
  - E.g. give them access to same new written values.
  - Note that if the subtxn which wrote a new value aborts, then so should the other subtxns, which have read the value. -> Complications, and we omit further discussion.

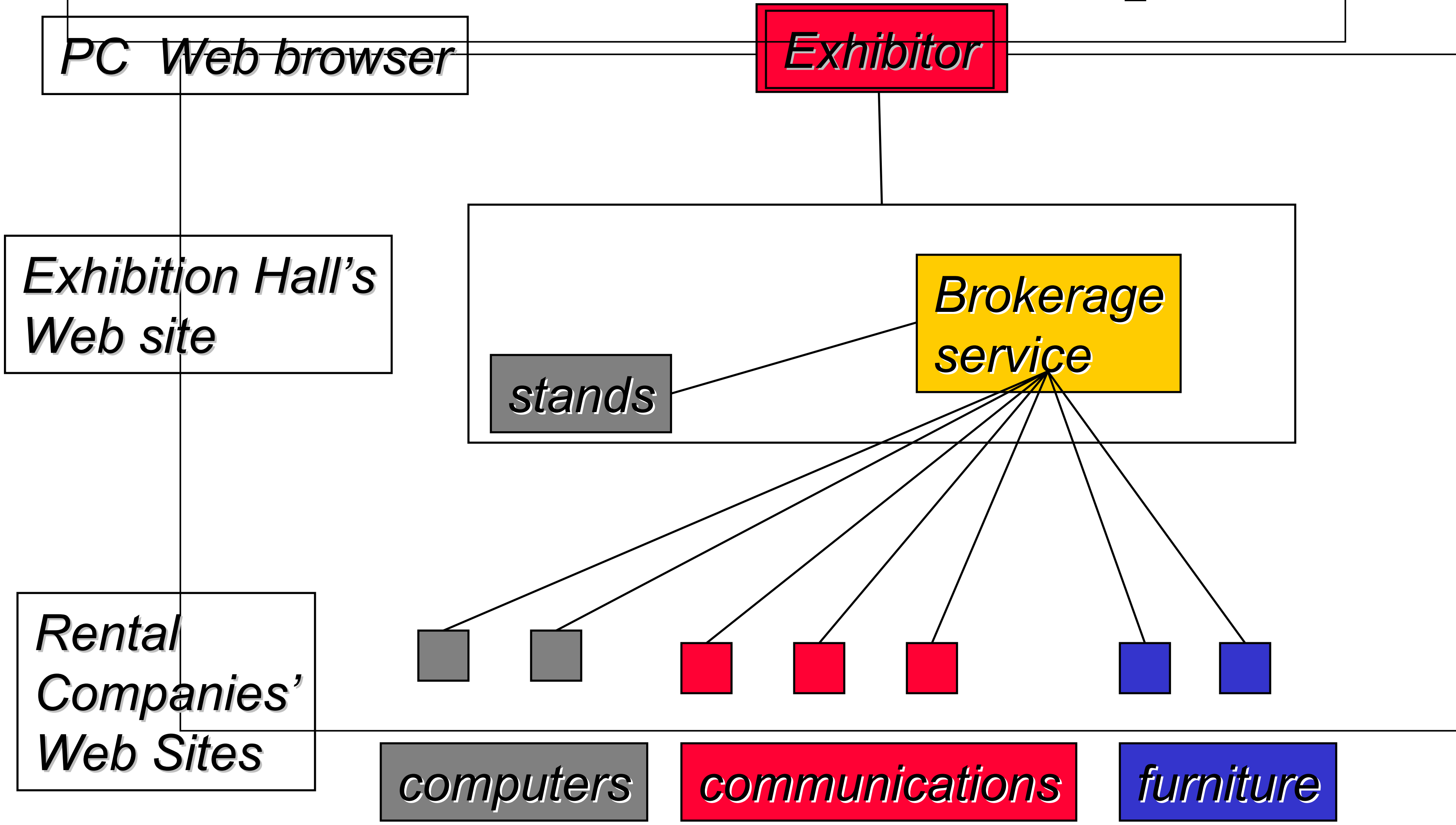
TIP (Transaction Internet Protocol)

Internet Commerce - Distributed Application Example Area

To exemplify the potential risks in safety and credibility of distributed systems, we will discuss an example application area.

Internet commerce is a good example area, because it deals with money and there is a lot of interest in application development.

Internet Commerce Example



So what has changed?

- Electronic commerce
  - Fixed set of participating companies
  - Proprietary, special-purpose protocols.
  - Specialist agent drives the dialogue, with special-purpose software
- Internet commerce
  - Transient sets of companies, maybe with brokers.
  - Protocols are Internet standards
  - The customer drives the dialogue from a general-purpose Web browser.

Internet Commerce

- A person, running a web browser on a desktop computer, electronically purchases a set of goods or services from several vendors at different web sites.
  - This person wants either the *complete set* of purchases to go through, or *none* of them.

Technical Problems with Internet Commerce

- Security
- Failure
- Multiple sites
- Protocol problems
- Server product limitations
- Response time

Security: some solutions

- Confidentiality: *Encryption*.
- Authentication: *Certification*.
- Integrity: *Digitally signed message digest codes*.
- Non-repudiation: *Receipts containing a digital signature*.
- You can do these through SSL/TLS or using the Java APIs.

TIP: Transaction Internet Protocol

- Proposed as an Internet Standard.
  - Backed by Microsoft and Tandem.
- Heterogeneous Transaction Managers can implement TIP to communicate with each other.

‘Conventional’ vs. Internet Transaction Processing

Conventional:

OSI TP

One-pipe:

- the application may only use the communications services supported by the transaction protocol.

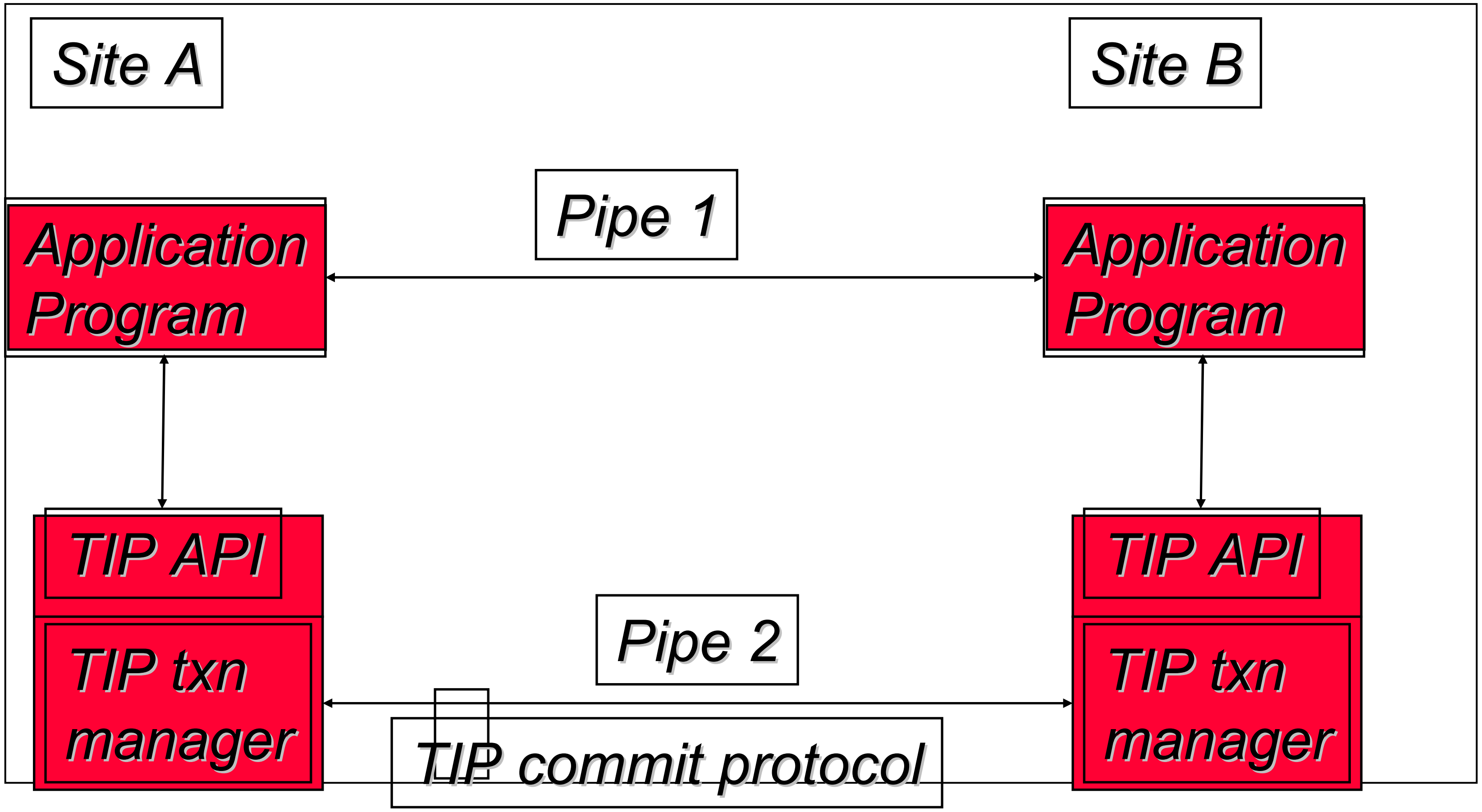
Internet:

‘Open’: TIP?

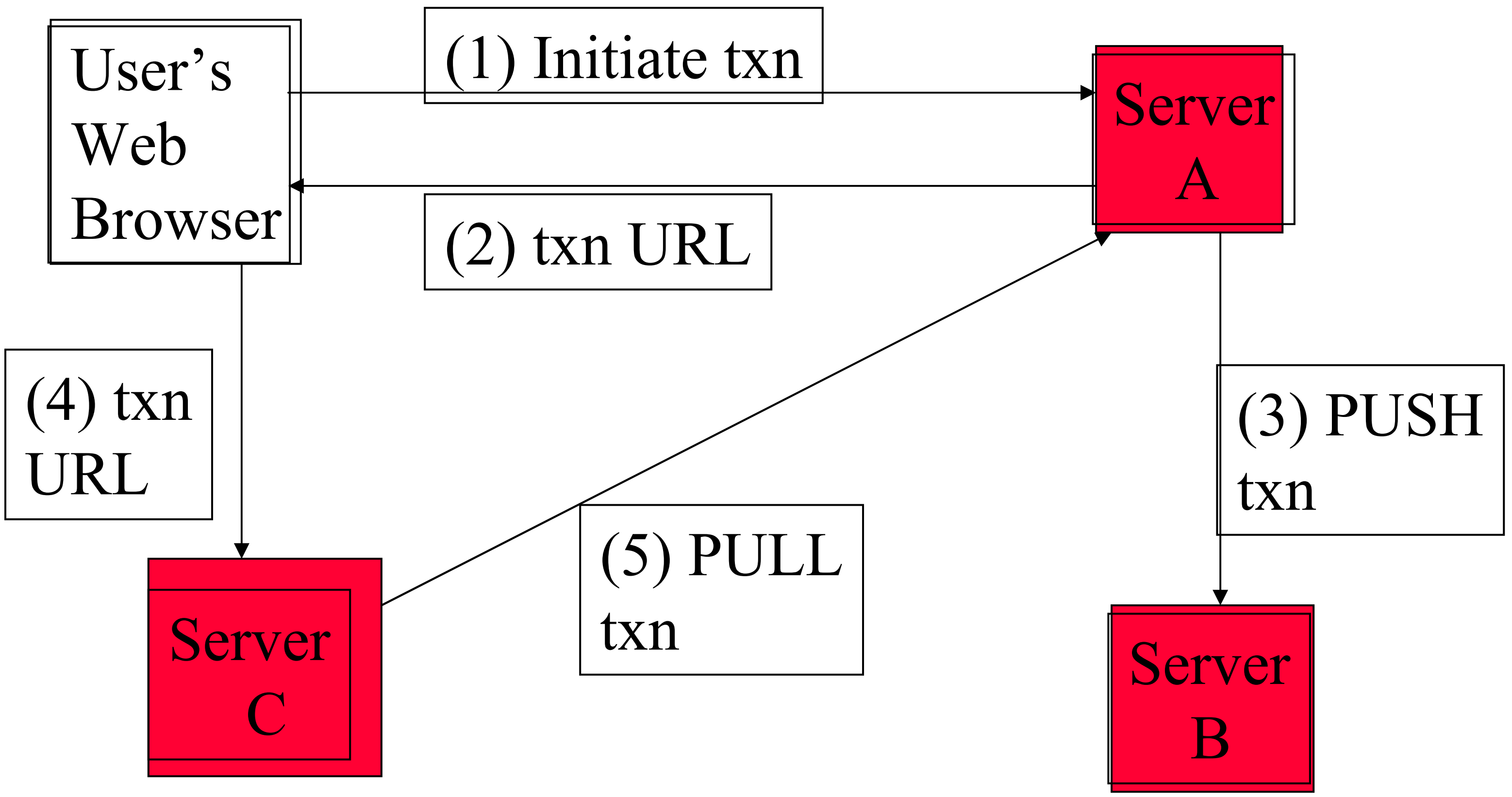
Two-pipe?:

- inter-application communication via some other protocol.

TIP: Two-pipe model



A Browsing Transaction



TIP Security

- Requires Secure-HTTP/SSL/TLS with
  - encryption and
  - end-to-end authentication.
- Operator intervention is needed when the commit protocol fouls up.
  - How will this work on the Internet?

TIP URLs and Txn Ids

- TIP URL:
  tip://<transaction manager address>?<transaction string>
- E.g. a TIP Txn Id
  tip://123.123.123.123/?transid1

TIP Connection

- Only one end (primary) can send commands to the other one (secondary)
- The primary and the secondary may swap roles in some special cases.
- We will go through the participant states and commands.

TIP Commands

- All commands and responses consist of one line of ASCII text.
- Each line can be split up into one or more "words”.
- Extra white space (spaces etc) is ignored.
- The first word of each line indicates the type of command or response.
- All defined commands and responses consist of upper-case letters only.
- For some commands and responses, subsequent words convey parameters for the command or response; each command and response takes a fixed number of parameters.

ERROR Command

- This command is valid in any state.
- It informs the secondary that a previous response was not recognized or was badly formed.
- A secondary should not respond to this command.
- The connection enters Error state.

MULTIPLEX Command

- MULTIPLEX <protocol-identifier> This command is only valid in the Idle state. The command seeks agreement to use the connection for a multiplexing protocol that will supply a large number of connections on the existing connection. We will skip furter details.
- Possible responses to the MULTIPLEX command are:
  - MULTIPLEXING The secondary party agrees to use the specified multiplexing protocol. The connection enters the Multiplexing state.
  - CANTMULTIPLEX The secondary party cannot support (or refuses to use) the specified multiplexing protocol. The connection remains in the Idle state.
  - ERROR The command was issued in the wrong state, or was malformed. The connection enters the Error state.

TLS Command

- TLS This command is valid only in the Initial state. A primary uses this command to attempt to establish a secured connection using TLS (a secure protocol for Internet communication). We skip further details of this.
- Possible responses to the TLS command are:
  - TLSING The secondary party agrees to use the TLS protocol. The connection enters the Tls state, and all subsequent communication is as defined by the TLS protocol.
  - CANTTLS The secondary party cannot support (or refuses to use) the TLS protocol. The connection remains in the Initial state.
  - ERROR The command was issued in the wrong state, or was malformed. The connection enters the Error state.

IDENTIFY Command

- IDENTIFY <lowest protocol version> <highest protocol version> <primary transaction manager address> | "-" <secondary transaction manager address>
- This command is valid only in the Initial state. The primary party informs the secondary party of:
  (1) the lowest and highest protocol version supported (all versions between the lowest and highest must be supported;
  (2) optionally, an identifier for the primary party at which the secondary party can re-establish a connection if ever needed (the primary transaction manager address); and
  (3) an identifier which may be used by intermediate proxy servers to connect to the required TIP transaction manager (the secondary transaction manager address).
- If a primary transaction manager address is not supplied, the secondary party will respond with ABORTED or READONLY to any PREPARE commands.

IDENTIFY Command

- Possible responses are:
  - IDENTIFIED <protocol version> The secondary party has been successfully contacted and has saved the primary transaction manager address. The response contains the highest protocol version supported by the secondary party. All future communication is assumed to take place using the smaller of the protocol versions in the IDENTIFY command and the IDENTIFIED response. The connection enters the Idle state.
  - NEEDTLS The secondary party is only willing to communicate over TLS secured connections. The connection enters the Tls state, and all subsequent communication is as defined by the TLS protocol (and not discussed here). If the primary party cannot support (or refuses to use) the TLS protocol, it closes the connection.
  - ERROR The command was issued in the wrong state, or was malformed. This response also occurs if the secondary party does not support any version of the protocol in the range supported by the primary party. The connection enters the Error state. The primary party should close the connection.

PUSH Command

- PUSH <superior's transaction identifier> This command is valid only in the Idle state. It seeks to establish a superior/subordinate relationship in a transaction with the primary as the superior.
- Possible responses are:
  - PUSHED <subordinate's transaction identifier> The relationship has been established, and the identifier by which the subordinate knows the transaction is returned. The transaction becomes the current transaction for the connection, and the connection enters Enlisted state.
  - ALREADYPUSHED <subordinate's transaction identifier> The relationship has been established, and the identifier by which the subordinate knows the transaction is returned. However, the subordinate already knows about the transaction, and is expecting the two-phase commit protocol to arrive via a different connection. In this case, the connection remains in the Idle state.
  - NOTPUSHED The relationship could not be established. The connection remains in the Idle state.
  - ERROR The command was issued in the wrong state, or was malformed. The connection enters Error state.

PULL Command

- PULL <superior's transaction identifier> <subordinate's transaction identifier>
- This command is only valid in Idle state. This command seeks to establish a superior/subordinate relationship in a transaction, with the primary party of the connection as the subordinate.
- Possible responses are:
  - PULLED The relationship has been established. Upon receipt of this response, the specified transaction becomes the current transaction of the connection, and the connection enters Enlisted state. Additionally, the roles of primary and secondary become reversed. (That is, the superior becomes the primary for the connection.)
  - NOTPULLED The relationship has not been established (possibly, because the secondary party no longer has the requested transaction). The connection remains in Idle state.
  - ERROR The command was issued in the wrong state, or was malformed. The connection enters the Error state.

BEGIN Command

- This command is valid only in the Idle state. It asks the secondary to create a new transaction and associate it with the connection. The newly created transaction will be completed with a one-phase protocol.
- Possible responses are:
  - BEGUN <transaction identifier> A new transaction has been successfully begun, and that transaction is now the current transaction of the connection. The connection enters Begun state.
  - NOTBEGUN A new transaction could not be begun; the connection remains in Idle state.
  - ERROR The command was issued in the wrong state, or was malformed. The connection enters the Error state.

PREPARE Command

- Valid only in the Enlisted state; Requests the secondary to prepare the transaction for commitment (2PC).
- Possible responses are:
  - PREPARED The subordinate has prepared the transaction; the connection enters PREPARED state.
  - ABORTED The subordinate has vetoed committing the transaction. The connection enters the Idle state. After this response, the superior has no responsibilities to the subordinate with respect to the transaction.
  - READONLY The subordinate no longer cares whether the transaction commits or aborts. The connection enters the Idle state. After this response, the superior has no responsibilities to the subordinate with respect to the transaction.
  - ERROR The command was issued in the wrong state, or was malformed. The connection enters the Error state.

ABORT Command

- This command is valid in the Begun, Enlisted, and Prepared states.
- It informs the secondary that the current transaction of the connection will abort.
- Possible responses are:
  - ABORTED The transaction has aborted; the connection enters Idle state.
  - ERROR The command was issued in the wrong state, or was malformed. The connection enters the Error state.

COMMIT Command

- Valid in the Begun, Enlisted or Prepared states.
- In the Begun or Enlisted state, it asks the secondary to attempt to commit the transaction.
- In the Prepared state, it informs the secondary that the transaction has committed.
    - Note that in the Enlisted state this command represents a one-phase protocol, and should only be done when the sender has 1) no local recoverable resources involved in the transaction, and 2) only one subordinate (the sender will not be involved in any transaction recovery process).

COMMIT Command /2

- Possible responses are:
  - ABORTED This response is possible only from the Begun and Enlisted states. It indicates that some party has vetoed the commitment of the transaction, so it has been aborted instead of committing. The connection enters the Idle state.
  - COMMITTED This response indicates that the transaction has been committed, and that the primary no longer has any responsibilities to the secondary with respect to the transaction. The connection enters the Idle state.
  - ERROR The command was issued in the wrong state, or was malformed. The connection enters the Error state.

QUERY Command

- QUERY <superior's transaction identifier> This command is valid only in the Idle state.
- A subordinate uses this command to determine whether a specific transaction still exists at the superior.
- Possible responses are:
  - QUERIEDEXISTS The transaction still exists. The connection remains in the Idle state.
  - QUERIEDNOTFOUND The transaction no longer exists. The connection remains in the Idle state.
  - ERROR The command was issued in the wrong state, or was malformed. The connection enters Error state.

RECONNECT Command

- RECONNECT <subordinate's transaction identifier>
- This command is valid only in the Idle state.
- A superior uses the command to re-establish a connection for a transaction, when the previous connection was lost during Prepared state.
- Possible responses are:
  - RECONNECTED The subordinate accepts the reconnection. The connection enters Prepared state.
  - NOTRECONNECTED The subordinate no longer knows about the transaction. The connection remains in Idle state.
  - ERROR The command was issued in the wrong state, or was malformed. The connection enters Error state.

Initial state

- Start state. When a connection is initiated, the initiating party becomes primary and the other secondary.
- The primary can send Identify or TLS commands. (Both are related to the connection, not to a txn).
    - Commands will be explained later.

Idle state

- The primary and the secondary have agreed on the protocol version.
- There is no txn yet associated with the connection.
- From this state, the primary can send any of the following commands: BEGIN, MULTIPLEX, PUSH, PULL, QUERY and RECONNECT.

Begun state

- The connection is associated with an active transaction, which can only be completed by a one-phase protocol (coordinator simply saying ABORT or COMMIT).
- A BEGUN response to a BEGIN command places a connection into this state.
- Failure of a connection in Begun state implies that the transaction will be aborted.
- From this state, the primary can send an ABORT, or COMMIT command.

Enlisted state

- The connection is associated with an active transaction, which can be completed by a one-phase or, two-phase protocol.
- A PUSHED response to a PUSH command, or a PULLED response to a PULL command, places the connection into this state.
- Failure of the connection in Enlisted state implies that the transaction will be aborted.
- From this state, the primary can send an ABORT, COMMIT, or PREPARE command.

Prepared state

- The connection is associated with a transaction that has been prepared (for commit).
- A PREPARED response to a PREPARE command, or a RECONNECTED response to a RECONNECT command places a connection into this state.
- Unlike other states, failure of a connection in this state does not cause the transaction to automatically abort.
- From this state, the primary can send an ABORT, or a COMMIT command.

Multiplexing state

- The connection is being used by a multiplexing protocol, which provides its own set of connections.
- No TIP commands are possible on the connection. (Of course, TIP commands are possible on the connections supplied by the multiplexing protocol.)
- The connection can never leave this state.

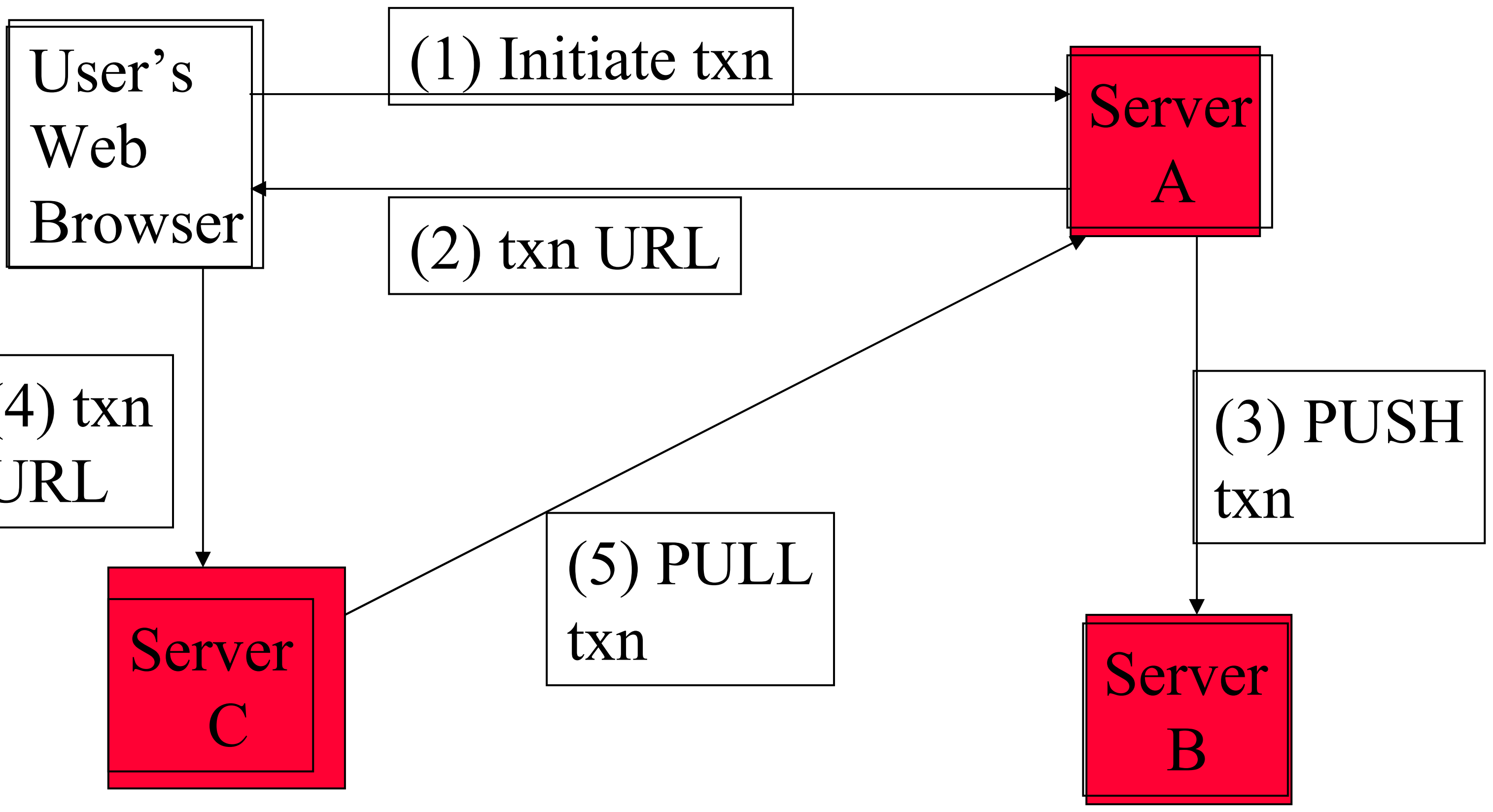
Tls state

- The connection is being used by the TLS protocol, which provides its own secured connection.
- No TIP commands are possible on the connection. (Of course, TIP commands are possible on the connection supplied by the TLS protocol.) The connection can never leave this state.

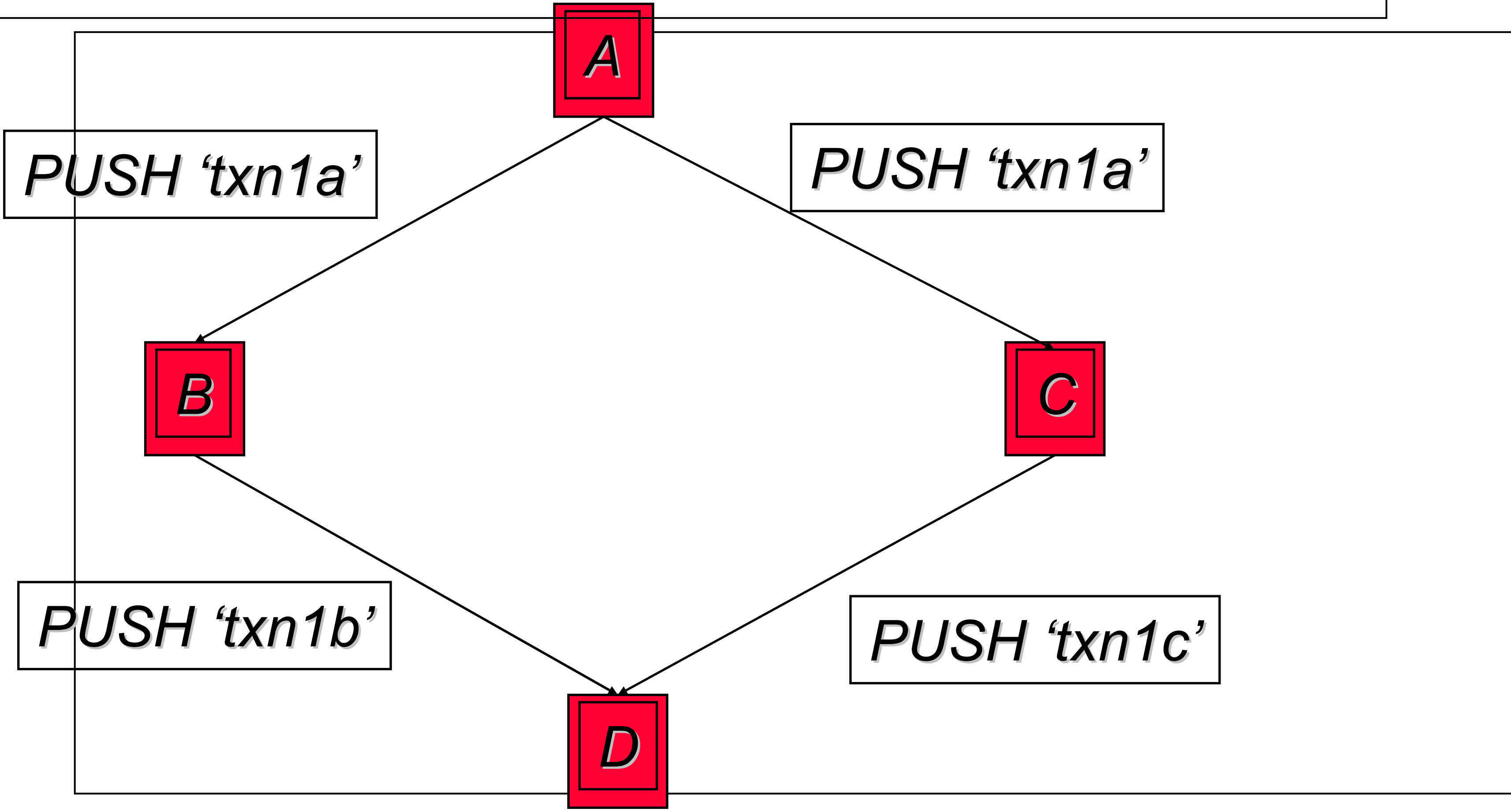
Error state

- A protocol error has occurred, and the connection is no longer useful.
- The connection can never leave this state.

A Browsing Transaction



Multiple inclusions of a site



TIP vulnerability

- Communication is pairwise point-to-point.
- Vulnerable to single link failures.

TIP Security

- Requires Secure-HTTP/SSL/TLS with
  - encryption and
  - end-to-end authentication.
- Operator intervention is needed when the commit protocol fouls up.
  - How will this work on the Internet?

SSL/TLS does NOT solve all of the problems

- TIP with TLS does not ensure non-repudiation.
- Various Denial-of-Service attacks are possible.
- A rogue participant could block progress by refusing to commit.

Denial-of-Service

- PULL-based:
  - A rogue company that knows the transaction ID sends a PULL to a site then closes the connection.
- PUSH-based
  - Flood a sites with PUSHes so that it cannot service legitimate requests.

Broken connection

- If a site loses its connection to its superior, the rogue sites sends it a RECONNECT command and tells it the wrong result of the commit.

Repudiation

- General point about how to repudiate:
- The site that wants to repudiate a transaction can always cause itself to crash and then recover, meanwhile losing all information that was in vulnerable storage.

Repudiation

- *Interaction of 2PC and authenticated protocol messages*
  - *The semantics of the authenticated messages only apply if the txn is committed.*

Repudiation

- *If a message from A to B is part of a 2PC protocol, then B's possession of the digital signature proves nothing.*
  - *A can claim: Yes, that was sent, but the action was rolled back.*
  - *B must prove that the action was committed. B must also prove that the message was part of that txn.*

Implications for Internet Commerce

- Existing protocols are inappropriate for the way people expect to be able to do business on the Internet.
- The TIP approach looked promising, but was not really accepted.
- For particular business sectors, a detailed analysis of likely transaction behavior will be needed.
- Market opportunities for brokerage companies.

When should one use distributed txn processing

When to user transactional services

- We have primarily talked about traditional data management, but our opening lecture slides have also other examples.
- There is a number of useful criteria for when transactional services should really be used.

Commit Protocols - when? Answer: <u>MT</u>BAP properties.

- [M] *Multiple* Participants,
  - at least one of which makes some durable data changes.
- [T] *Tentativeness*
  - The Participants can tentatively establish of feasibility, but external circumstances can render the changes infeasible.

Commit Protocols - when? Answer: MTBAP properties.

- [B1 B2] *Before* the Participant votes,
  - (1) external circumstances can render the changes infeasible at any time and
  - (2) the computation context or Coordinator can force the Participant to abort at any time.

Commit Protocols - when? Answer: MTBAP properties.

- [A1 A2] *After* the Participant votes feasible,
  - (1) only a catastrophic failure can render the changes infeasible and
  - (2) the Coordinator's decision determines whether the Participant commits or aborts.
- [P] *Permanence*
  - A Participant that is required to make changes cannot undo its changes.

MTBAP Example 1

- A single client is purchasing CDs from a single business.
- No A2 (No coordinator needed)
  - Once the customer has ordered, the business site can unilaterally decide the fate of the transaction.
- This type of transactions is clearly dominant at present.

MTBAP Example 2

- A flight booking system
  - Reservation decrements the number of seats available.
  - At transaction rollback, a compensating transaction increments the number of seats available.
- No P (permanence) because of the compensating transactions
  - No distributed commit protocol is required.

Barrier Synchronisation - When? Answer: MFAG properties

- [M] *Multiple* independent Participants are involved in the computation.
- [F] *Fluid* state of the world
  - The Participant does not have the ability to freeze the state of a part of the outside world to obtain a consistent snapshot.

Barrier Synchronisation - When? Answer: MF<u>AG</u> properties

- [A] *Asynchronicity*
  - There is no global time and the computations have no direct control over the precise moment at which a Participant takes a reading from the outside world.
- [G] *Globally*-consistent state
  - The computation's Participants need to see snapshots of the world that are consistent with each other.

MFAG Example

- Consider again customers buying books from a bookstore with all server objects executing on the same machine
- The servers' view of the availability of books is synchronised via the local operations.
- No A (asynchronicity) property, thus no barrier synchronisation is required.

Practical distributed txn management issues

How to implement the transactional services?

- Traditionally, the distributed DBMS provided transactional services.
- Implementing the services yourself is difficult.
- Java 2 Enterprise Edition (J2EE) and Microsoft Transaction Server offer the application programmer an API for implementing the participant while relying on the conventional system services for coordination and communication.

Heterogeneous systems and commit protocols

- If a business transaction consists of a set of distributed transactions on heterogeneous systems / platforms, these must be coordinated.
- Implementing 2PC or using e.g the Transaction Internet Protocol is not very complicated.

Heterogeneous systems and commit protocols / Coordination

- Implementation of a 2PC protocol is easy, but local/global coordination may be complicated.
- Suppose we use a database - we cannot commit first locally, since the global outcome may be abort.
- If we vote Yes, then we must be sure that our transaction is still alive, if we get a Commit decision our transaction locally.

Heterogeneous systems and commit / Sub-txn management

- 1: Write your own local txn processing (with data mangement etc.)
  - This is quite complicated.
- 2: The txn-managent (e.g. database) should provide a provisional-commit stage, where only a catastrophic failure aborts the txn.
  - The txn can not participate in deadlock but some approximate deadlock management schemes may abort it.

Coursework

Coursework

- Implement distributed data management.
  - This far, we have just implemented a lock manager, which of course contains much of the required functionality.
  - You may manage, e.g. an integer vector, but you may also choose a more complicated data structure, if you feel like it.

Functionalities

- Client takes read, write, rollback and commit commands, and calls the services from a txn coordinator.
- The txn coordinator (like LockClient in the example implementation) takes the locks from the servers and, having got them, performs the read/update operation.
- The resource manager (like LockManager) includes locking, deadlock management, 2PC, and commit/rollback of txns.

More coursework instrucions

- More coursework instructions are going to be available through course web pages.
- Deadlines will be set separately by each department.

Exam

Exam

- No material will be allowed in the exam.
- Some technical details may be given as a part of examination material.