8. Application Servers

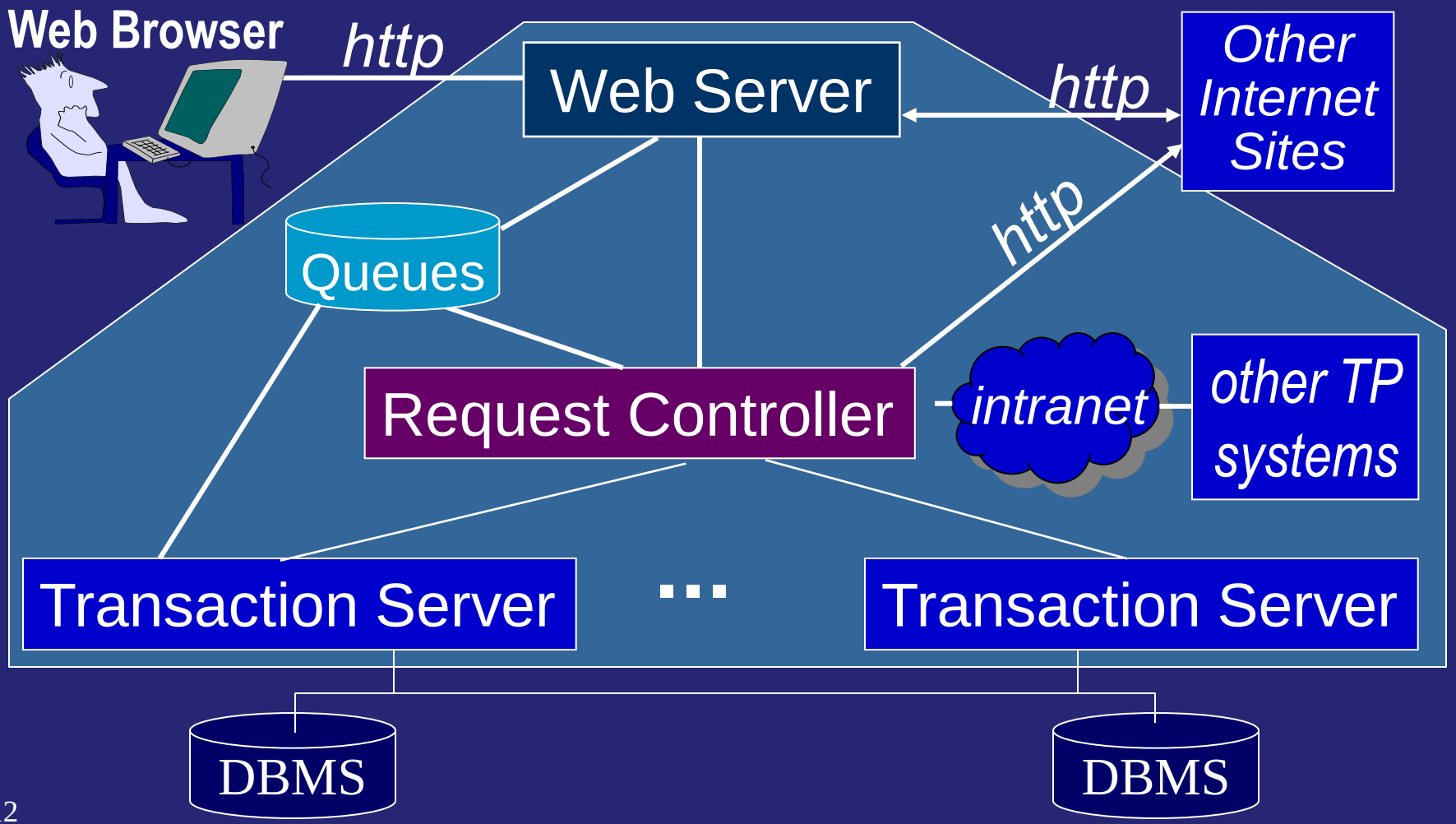
CSEP 545 Transaction Processing
Philip A. Bernstein

Outline

1. Introduction
2. Two-Tier vs. Three-Tier
3. Web Servers
4. Transaction Bracketing
5. Processes and Threads
6. Remote Procedure Call

8.1 Introduction

- An application server coordinates the flow of requests between message sources (displays, applications, etc.) and application programs that run requests as transactions.



Application Server Components

- Web Browser
  - A smart device, with forms, menus, input validation
- Web server
  - Performs front-end work, e.g., security, data caching, ….
  - "Calls" the web page associated with the URL, which in turn calls a request controller
- Request controller
  - Calls Start, Commit, and Abort
  - App logic that transforms request (automatic loan payment, money transfer) into calls on basic objects (loan, account). Sometimes called *business rules*.
- Transaction server
  - Business objects (customer, account, loan, teller)
- DBMS – Database Management System

Application Server Functions

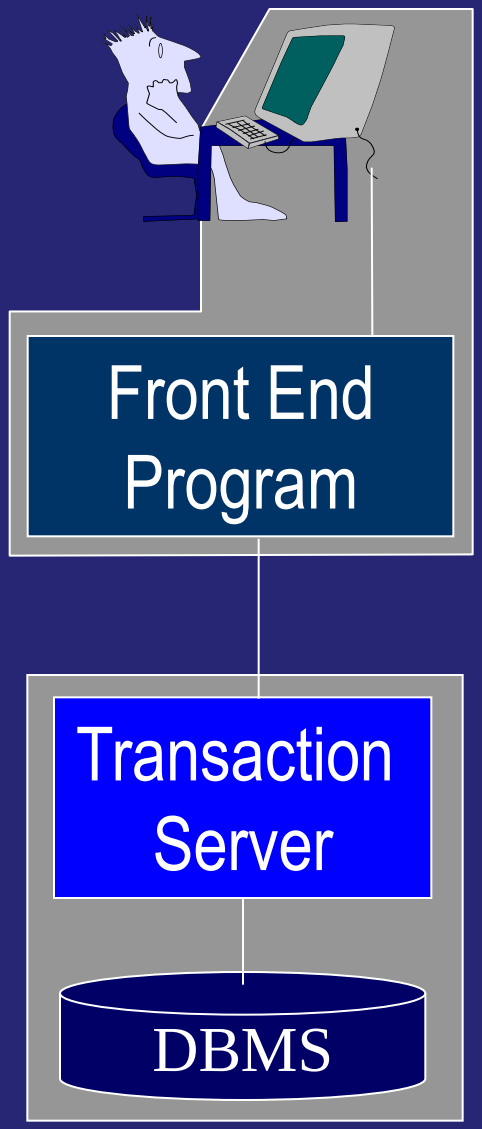
- Glue and veneer for TP applications.
  - Glue fills in gaps in system functionality.
  - Covers the interface with a seamless veneer.
- Mostly, it provides run-time functions for applications (request control and transaction servers).
  - OS functions: threading and inter-process communication, often passed through from the underlying OS.
  - Dist'd system functions: transactions, security, queuing, name service, object pools, load balancing, …
  - Portal functions: shopping cart, catalog mgmt, personalization ...
- Provides some application development tools.
- Provides system mgmt for the running application.

Application Server Products

- Adobe (Macromedia) ColdFusion
- Apple WebObjects
- HP (Tandem) Pathway
- HP (DEC) ACMS
- IBM CICS
- IBM IMS/DC
- IBM Websphere
- Iona iPortal App Server
- Microsoft .NET Enterprise Services (formerly COM+, MS Transaction Server (MTS))
- Oracle (BEA) Tuxedo
- Oracle (BEA) WebLogic
- Oracle Application Server
- RedHat JBoss
- Sybase EAServer
- Also see serverwatch.com

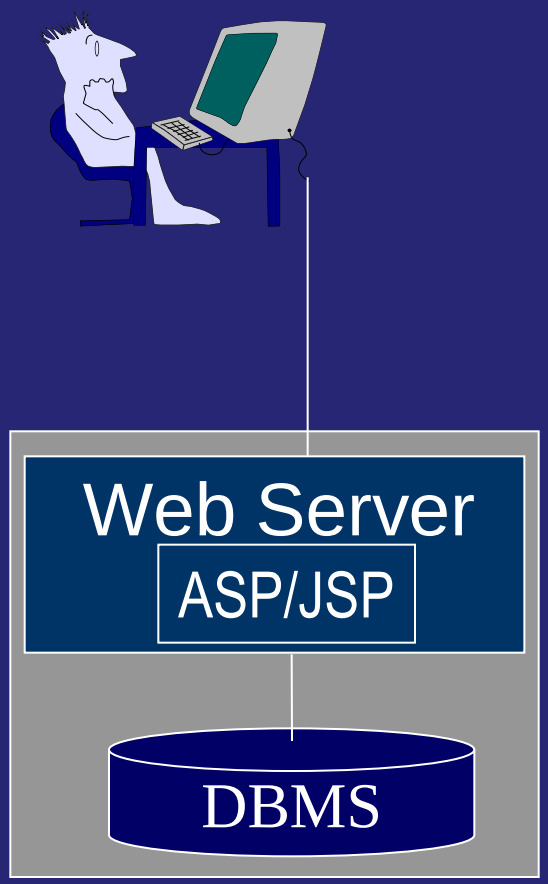
8.2 Two-Tier vs. Three-Tier

- Before the web, most small-to-medium scale apps were implemented in 2 tiers on a LAN
  - PC runs a 4GL, such as Sybase PowerBuilder, Microsoft Visual Basic, or Embarcadero Delphi
  - Server system includes transaction server application and DBMS



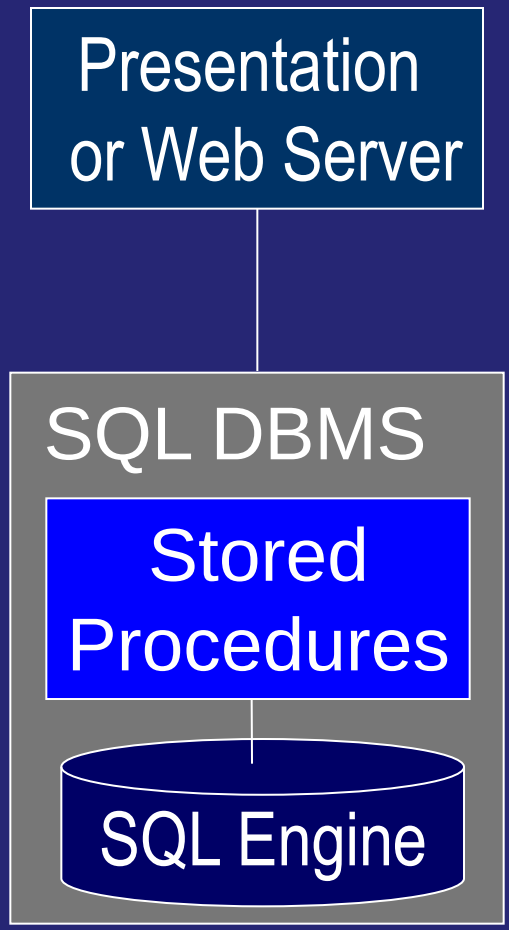
Two-Tier for the Web

- Front end program ⇒ Web server
  - In essence, the web browser is a device
- Web server invokes a web page that has embedded script
  - Active Server Page (ASP .NET) or Java Server Page (JSP)
  - Page (file) extension tells the web server to run the ASP/JSP interpreter
  - Script can include DBMS calls and can run as a transaction



Two-Tier is Enabled by DBMS Stored Procedures

- Stored procedure – An application procedure that runs inside the DBMS
  - Often in a proprietary language, such as PL/SQL (Oracle), T-SQL (MS, Sybase)
  - Moving toward standard languages, such as Java and C#
- Implement transaction servers as stored procedures
- Use DBMS client-server protocol
- No application server needed
  - Hence, sometimes called “TP lite”



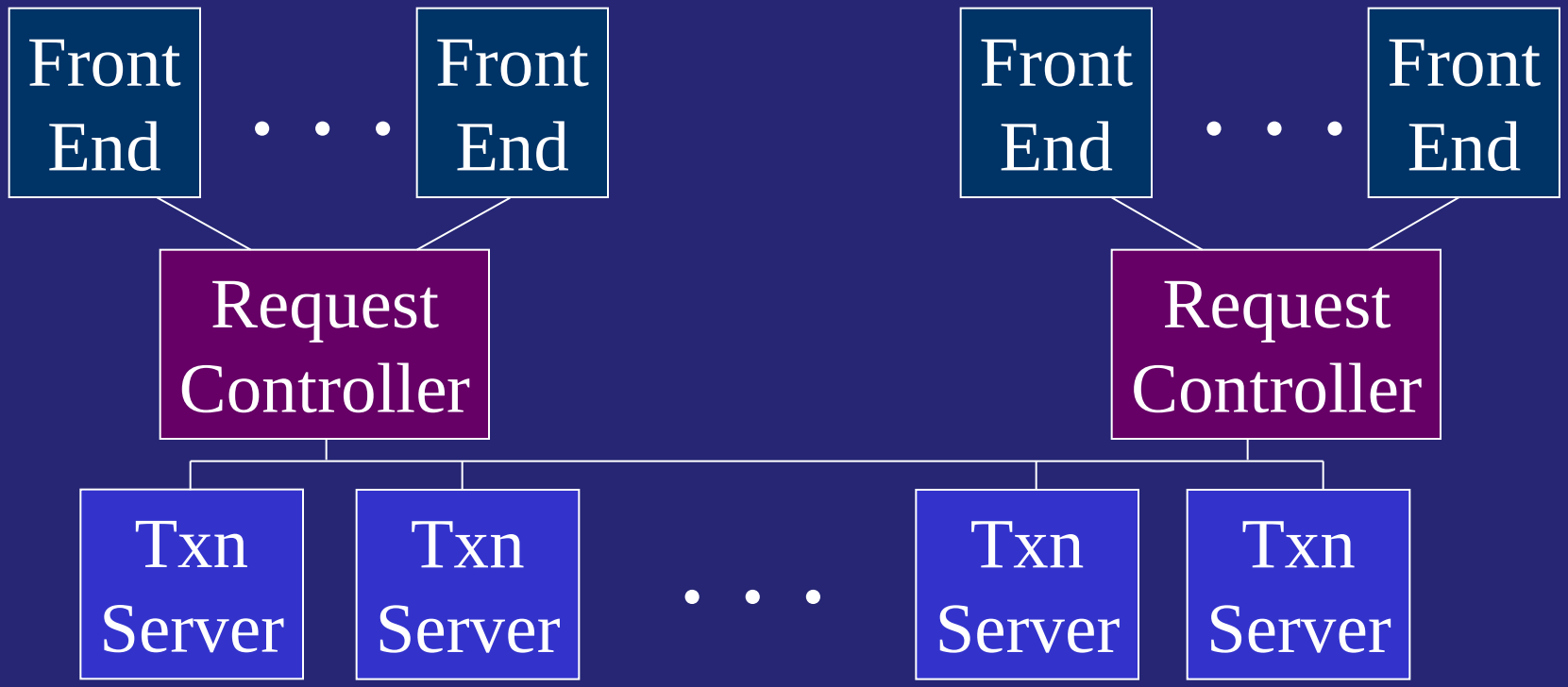
An Aside: DBMS Interfaces

- Most apps are object-oriented
- Most database interfaces are relational
- So the object-relational mapping layer is an important part of TP applications
  - Often custom for an app suite
  - Some generic: Microsoft Entity Framework, Oracle TopLink, Open Source Hibernate
- Language Integrated Query (LINQ)
  - Strongly-typed DB interface to .NET languages

Scalability Problem of Two-Tier

- 2-tier is feasible, but does not scale as well as 3-tier due to session management
- Session - shared state between communicating parties
  - Entails memory cost and a setup cost (3-way handshake)
- Sessions reduce amount of per-request context passing (comm. addresses, authenticated user/device)
  - Standard DB APIs (e.g., ODBC) work this way
  - Hence, in 2-tier, N clients and M servers $\Rightarrow$ N$\times$ M sessions
  - E.g. 10^5 presentation servers and 100 servers $\Rightarrow$ 10^7 sessions
- Partition presentation servers across request controllers
  - Each request controller still connects to all txn servers but there are many fewer request controllers than presentation servers

3-Tier Reduces the Number of Sessions



- Partition the set of front end devices (e.g., 10^3 devices per RC)
- 100 RC × (10^3 devices/RC + 10^2 TS/RC) = 110,000 sessions

Partitioning Txn Servers

- If DB server is a bottleneck, then partition it.
  - By value ranges or hashing
    - E.g., partition Accounts by account range
  - Range partitioning is susceptible to overload. It benefits from auto-reconfiguration by splitting ranges.
  - Table-lookup partitioning, per key-value.
    - Enables upgrading a user to a new service or new release
- Request control is needed to direct a call to the right DB partition (parameter-based routing)
  - RC sends a Debit request for Account *x* to the TS connected to the DB partition containing Account *x*

2-Tier vs. 3 Tier — Other Issues

- In early 90's people argued whether 2-Tier was enough
  - Scalability was the decisive factor, but there were other issues
- Database Servers
  - Nonstandard stored procedure language, usually less expressive with weaker development tools and it's another language to learn
  - Limited interoperability of cross-server calls
  - Limited interoperability of distributed transactions
  - Poor fit with OO design, which are inherently 3-tier (client, business rules, business objects)
- Application Servers
  - More system complexity

How the Web Changed Things

- Front End Program ⇒ Web server
- All requests have to pass through a Web server
  - In 2-tier, each Web server needs sessions to all DB servers
  - Session reduction by request control is less critical but still useful
  - DB partitioning may be implemented by the DB server
- Request control is still useful for request mgmt
  - Calling Start, Commit, and Abort
  - Encapsulating business rules that transform each request into calls on basic objects

8.3 Web Servers

- Presentation independence - application is independent of the display device used
  - Today, this is via http and html
  - In the past, it was via a display controller or middle-tier minicomputer whose presentation functions insulated the rest of the back-end system from different device types
- Web server performs presentation functions:
  - Gathering input
  - Validating input
  - DB caching
  - Authentication
- They also do some basic request routing
  - Constructing requests
  - Invoking applications
- Examples - IIS (MS), Apache, Netscape Server

Gathering Input

- Gathering input - Select transaction type (menu item, etc.), and fill in a form (request's parameters)
  - Today, Web forms, moving to XML (XForms, XSLT, …)

- 40-year evolution of presentation devices
  - Teletype, character-at-a-time terminal (async), block-mode terminal (IBM 3270)
  - Specialized devices - ATMs, bar code readers, gas pumps, robots, credit card authorization, cash registers, ticket printers, etc.
  - 4GL on a PC - ActiveX controls accessed from Visual Basic (VB), PowerBuilder, Delphi, etc.
  - HTML 5 in a web browser.

Caching

- Every process-to-process call has a cost
  - Adds to response time and consumes resources
- Use a cache in Web server to avoid calling request controller or DB system
  - Cache popular read-only data that need not be refreshed frequently
  - E.g., catalog items, sale items, cover page at an auction site, recent news, etc.
  - Also, data required for input validation info
- Or use a cache server, such as memcached, Oracle Coherence, or Windows Server AppFabric Caching

Input Validation

- Validate input against locally cached tables
  - E.g., product types, department numbers
- Avoids wasting communications and server resources for obvious input errors
  - Fewer round-trips to the DBMS
  - And faster feedback to the end user
- "Cache" is part of the web page
  - List boxes, script
  - Cache size is a factor (it affects page access time)

Authentication

- Authentication - determining the identity of a user and/or display device
  - Client system (e.g., PC) may do authentication, but the server usually does it too (doesn't trust clients)
  - Encrypt the wire to avoid wiretapping and spoofing
- On the Web, Transport Layer Security (successor to SSL))
  - Client gets a certificate with server's public key from the server, signed by trusted authority's private key
  - Client validates certificate using the authority's public key
  - Client and server exchange encryption keys
  - Then all messages are encrypted

Authentication (cont'd)

- Geographical entitlement - check that a particular *device* is allowed access (e.g., security trading room)
- Need system mgmt functions to create accounts, initialize passwords, bracket hours of access (simplify it using a role abstraction)

Constructing Requests

- A request includes
  - User id – for authorization and personalization
  - Device id – where to send a reply
  - Device type - what message types can it understand?
  - ObjectID – in a OO setting
  - RequestID – to ask later about request status & to link a reply
  - Request type – name of transaction type requested
  - Request-specific parameters

- Can be combined with protocol header (e.g., http header)

Application Invocation

- Request arrives as an http message.
  - Need to call a program (i.e. a WFC), to perform the request
- Common Gateway Interface
  - Write a script, store it as a file in cgi-bin
  - Web server creates a process to execute the request (Slow!!)
- ISAPI (Microsoft) and NSAPI (Netscape)
  - Web server calls an in-proc .dll instead of creating a process
  - Web server can cache the .dll
  - More complex programming model, but much faster
- Active Server Pages and Java Server Pages
  - Offers the performance of ISAPI with programmability of CGI

Load Balancing

- Web servers enable scale out, so you can just add more server boxes to handle more load.
- To simplify this problem
  - Ensure all web servers are stateless. I.e., no server-specific state and don't retain client state on web servers (hard to avoid …)
  - Statelessness implies any web server can process any request.
  - It also makes web server recovery is easy.
  - Randomly assign requests to web servers (e.g., an IP sprayer)
  - Avoid sending requests to a failed web server
  - Downside: Have to pass all state with every request
- This is the philosophy behind REST/HTTP, using Get and Post operations

8.4 Transaction Bracketing

- For the most part, Request Controllers (RC) and Transaction Servers are just plain old server programs

- The main RC differentiating features
  - Brackets transactions (issues Start, Commit, and Abort)
  - Handles Aborts (returns cause of the Abort)
  - Does not access the DBMS

Nested Transaction Calls

- What does Start do, when executed within a txn?
  1. it starts an independent transaction, or
  2. it does nothing, or
  3. it increments a nested transaction count (which is decremented by each commit and abort), or
  4. it starts a sub-transaction.

- (2) and (3) are common.
  - Enables a transaction-bracketed program to be called by another transaction

- (1) implies Be Careful!

Transaction Bracketing

- Request controller brackets the transaction with Start, Commit, Abort.
- Chained - All programs execute in a transaction.
  A program can commit/abort a transaction,
  after which another transaction immediately starts
  - E.g., CICS syncpoint = Commit&Start
  - Prevents programmer from accidentally issuing resource manager operations outside a transaction
- Unchained - Explicit Start operation, so some statements can execute outside a transaction
  - No advantages, unless transactions have overhead even if they don't access resources.

Transparent Transaction Bracketing

- Transaction-hood is a property of the app component.
- In COM+, a class is declared:
  - *requires new* - callee always starts a new transaction
  - *required* - if caller is in a transaction, then run callee in caller's transaction, else start a new transaction
  - *supported* - if caller is in a transaction, then run callee in caller's transaction, else run outside of any transaction
  - *not supported* - don't run in a transaction
- Caller can create a transaction context, which supports Commit and Abort (chained model).
  - Callee issues SetComplete when it's done and willing to commit, or SetAbort to abort.

Transparent Txn Bracketing (cont'd)

- Java Enterprise Edition
  - Implements COM+ technology in Java: RequiresNew, Required, Supported, NotSupported
  - It came later, so there are two additions.
  - Mandatory – If caller is in a transaction, then run the callee in that transaction, else raise an exception
  - Never – If caller is in a transaction, then raise an exception

Runtime Library Support

- TP services require runtime library support
  - May or may not be language-specific
- Language-specific
  - Java 2 Enterprise Edition (J2EE, formerly Enterprise Java Beans)
    - Encapsulates runtime library as a *container* object.
    - BEA Weblogic, IBM Websphere, ….
  - Older examples are Tandem Pathway (Screen COBOL) and Digital's ACMSxp (Structured Txn Defn Lang)
- Language-independent runtime library
  - MS COM+, IBM's CICS, Oracle App Server, …

Exception Handling

- Request control brackets the transaction, so it must say what to do if the transaction aborts
- An exception handler must know what state information is available
  - Cause of the abort, e.g., a status variable
  - Possibly program exception separate from abort reason
  - For system failures, application must save state in stable storage; note that none of the aborted txn's state will be available
- Chained model - exception handler starts a new txn
- COM+ - component returns a failure hresult

Integrity of Request after Abort

- To permit request retries, it's useful if **`get-request`** runs inside the request's transaction:

```
Start;
  get-request;
     . . .
Commit;
```

- If the transaction aborts, then **`get-request`** is undone, so the request becomes available for the next **`get-request`**.
- In the RPC or "push model," make the "catch-the-call" operation explicit, so it can be undone. Possibly hidden in the dispatch mechanism. Often requires a queue manager.

Savepoints

- Savepoint - a point in a program where an application saves all its recoverable state
- Can restore a savepoint within the transaction that issued the savepoint. (It's a partial rollback.)
- SQL DBMSs use them to support atomic SQL statements.

```
Start;
get-request;
Savepoint("B"); . . .;
if (error) {Restore("B"); …; Commit;}
. . .;
Commit;
```

- Savepoints are not recoverable. If the system fails or the transaction aborts, the txn is completely undone.

8.5 Processes and Threads

- Application Server architecture is greatly affected by
  - which components share an address space
  - how many control threads per address space
- TP grew up in the days of batch processing, and reached maturity in the days of timesharing.
- TP users learned early that a process-per-user fails:
  - Too much context switching
  - Too much fixed memory overhead per process
  - Process per user per machine, when distributed
  - Some OS functions scan the list of processes
  - Load control is hard

Multithreading

- Have multiple threads of control in an address space
- Used to be a major Application Server feature
  - Application Server switches threads when app calls a Application Server function that blocks
- Now, most OS's support it natively
  - Can run a process's threads on different processors (SMP)
- Whether at the user or OS level,
  - multithreading has fewer processes and less context switching
  - but little protection between threads and a server failure affects many transactions

Mapping Servers to Processes

- Presentation/Web servers, request controllers, and transaction servers are multithreaded servers

- Costs 1500 - 25,000 instructions per process call, vs. 50 instructions per local procedure call …
  - but it scales, with flexible configuration and control

8.6 Remote Procedure Call

- Program calls remote procedure the same way it would call a local procedure
- Hides certain underlying complexities
  - communications and message ordering errors
  - data representation differences between programs
- Transactional RPC
  - Ideally, Start returns a transaction ID that's hidden from the caller
  - Procedures don't need to explicitly pass transaction id's.
  - Easier and avoids errors

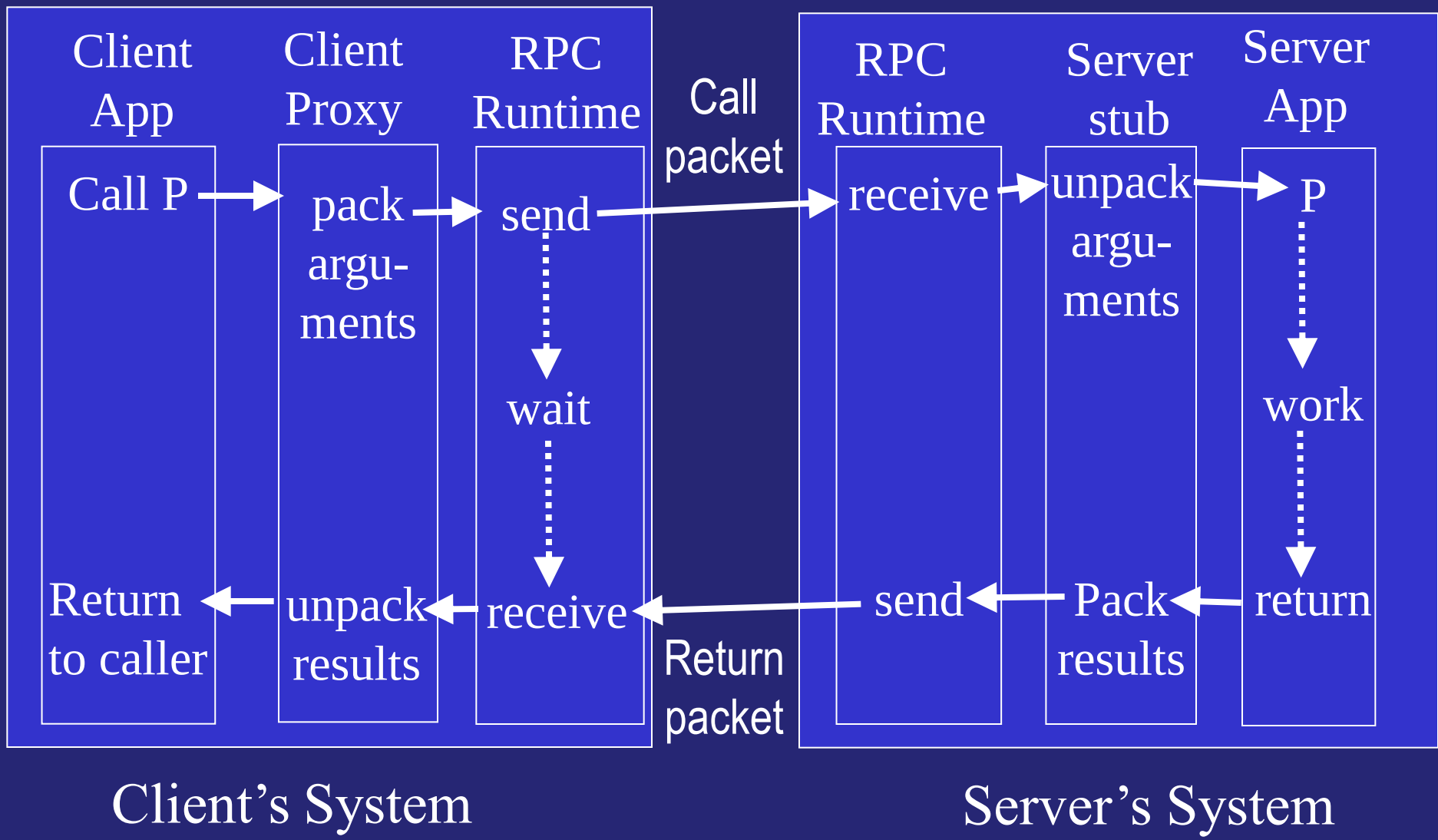
Binding

- Interface definitions
  - From app or written in an interface definition language (IDL)
  - compiles into Proxy and Stub programs
  - Client calls the Proxy (representing the server)
  - Stub calls the Server (represents the client on the server)
- Marshaling
  - proxy marshals (sequentially lays out) calling parameters in a packet and decodes marshaled return values
  - stub decodes marshaled calling params and marshals return params
- Communications binding
  - Client finds the server location via a directory service, based on server name and possibly a parameter value
  - To load balance across identical servers, randomly choose a server

Binding (cont'd)

- The binding process has security guarantees
  - The client must have privileges to bind to the server
  - The client must know it's binding to an appropriate server to avoid being spoofed
  - E.g. client and server authenticate each other during session creation, and maybe per-access too

RPC Walkthrough



Performance

- There are basically 3 costs
  - marshaling and unmarshaling
  - RPC runtime and network protocol
  - physical wire transfer
- In a LAN, these are typically about equal
- Typical commercial numbers are 10-25K machine instructions
- Can do much better in the local case by avoiding a full context switch

Stateful Applications

- Sometimes an application maintains state on client's behalf, possibly across transactions. E.g.,
  - Server scans a file. Each time it hits a relevant record it returns it. Next call picks up the scan where it left off.
  - Web server maintains a shopping basket or itinerary, etc.
  - Server caches client's authenticated identity or authorizations
  - Server caches user's profile for personalization

Approach 1: client passes state to server on each call, and server returns it on each reply. Server retains no state.

- Doesn't work well for TP, because there's too much state
- Note that transaction id context is handled this way.

Stateful Servers Using Sessions

Approach 2: Shared client & server state via a session

- Server maintains state, indexed by client id (txn id or cookie). Client's later RPCs must go to same server.
- If the client fails, server must be notified to release client's state or deallocate based on timeout
- For transaction RPC, encapsulate context as a (volatile) resource. Delete the state at commit/abort. Or possibly, maintain state across transaction boundaries, but reconstruct it after system failure.

- E.g., COM+: Client can call a server object many times
  - Client creates server object, which retains state across RPCs
  - SetComplete (or SetAbort) by server app says that transaction can be committed (or aborted) *and* state can be deleted
  - EnableCommit (or DisableCommit) by server app says transaction can (or cannot) be committed by client and *don't* delete server state

Stateful Servers Using Sessions (cont'd)

- Session state can be stored persistently
  - In a database system
    - Possibly saved within a transaction
  - Requires explicit deletion when the session fails
    - E.g., via a lease that times out
  - Could be tied to a long-lived business process

Fault Tolerance

- If a client doesn't receive a reply within its timeout period
  - RPC runtime can send a "ping" for non-idempotent calls
  - After multiple pings, it return an error.
  - For idempotent calls, RPC runtime can retry the call (server interface definition can say whether it's idempotent)

Web Services

- Distributed computing standards to enable interoperation on the Internet
- SOAP - RPC with XML as marshalling format and WSDL as interface definition
- UDDI - directory for finding Web Service descriptions
- WS-Transaction - 2PC
- WS-Security, WS-Coordination, WS-Routing, …
- www.ws-i.org

Summary

- Scalability – 2 vs. 3 tier, sessions, stored procedures
- Web Server – gathering input, validating input, caching, authentication, constructing requests, invoking applications, load balancing
- Transaction bracketing – transparency, nesting, exceptions, request integrity, savepoints
- Server processes – threads
- RPC – binding, stateful servers