1. Introduction

CSEP 545 Transaction Processing

Philip A. Bernstein

Outline

1. The Basics
2. ACID Properties
3. Atomicity and Two-Phase Commit
4. Performance
5. Styles of System

1.1 The Basics - What's a Transaction?

- The *execution* of a program that performs an administrative function by accessing a *shared database*, usually on behalf of an *on-line* user.

Examples

- Reserve an airline seat. Buy an airline ticket
- Withdraw money from an ATM.
- Verify a credit card sale.
- Order an item from an Internet retailer
- Place a bid at an on-line auction
- Submit a corporate purchase order

The "ities" are What Makes Transaction Processing (TP) Hard

- Reliability - system should rarely fail
- Availability - system must be up all the time
- Response time - within 1-2 seconds
- Throughput - thousands of transactions/second
- Scalability - start small, ramp up to Internet-scale
- Security – for confidentiality and high finance
- Configurability - for above requirements + low cost
- Atomicity - no partial results
- Durability - a transaction is a legal contract
- Distribution - of users and data

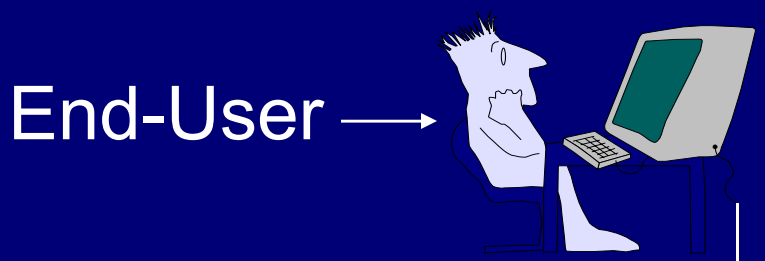
What Makes TP Important?

- It's at the core of electronic commerce
- Most medium-to-large businesses use TP for their production systems. The business can't operate without it.
- It's a *huge* slice of the computer system market. One of the largest applications of computers.

TP System Infrastructure

- User's viewpoint
  - Enter a request from a browser or other display device
  - The system performs some application-specific work, which includes database accesses
  - Receive a reply (usually, but not always)
- The TP system ensures that each transaction
  - is an independent unit of work
  - executes exactly once, and
  - produces permanent results.
- TP system makes it easy to program transactions
- TP system has tools to make it easy to manage

TP System Infrastructure …
Defines System and Application Structure



System Characteristics

- Typically < 100 transaction types per application
- Transaction size has high variance. Typically,
  - 0-30 disk accesses
  - 10K - 1M instructions executed
  - 2-20 messages
- A large-scale example: airline reservations
  - hundreds of thousands of active display devices
  - plus indirect access via Internet
  - tens of thousands of transactions per second, peak

Availability

- Fraction of time system is able to do useful work
- Some systems are *very* sensitive to downtime
  - airline reservation, stock exchange, telephone switching
  - downtime is front page news

Downtime	Availability
1 hour/day	95.8%
1 hour/week	99.41%
1 hour/month	99.86%
1 hour/year	99.9886%
1 hour/20years	99.99942%

- Contributing factors
  - failures due to environment, system mgmt, h/w, s/w
  - recovery time

Application Servers

- A software product to create, execute and manage TP applications
- Formerly called *TP monitors*. Some people say App Server = TP monitor + web functionality.
- Programmer writes an app to process a single request. App Server scales it up to a large, distributed system
  - E.g. application developer writes programs to debit a checking account and verify a credit card purchase.
  - App Server helps system engineer deploy it to 10s/100s of servers and 10Ks of displays
  - App Server helps system engineer deploy it on the Internet, accessible from web browsers

Application Servers (cont'd)

- Components include
  - an application programming interface (API) (e.g., Enterprise Java Beans)
  - tools for program development
  - tools for system management (app deployment, fault & performance monitoring, user mgmt, etc.)
- Enterprise Java Beans, IBM Websphere, Microsoft .NET (COM+), BEA Weblogic, Oracle Application Server

App Server Architecture, pre-Web

- Boxes below are distributed on an intranet

Front End Program

Message Inputs

Queues

Requests

Network

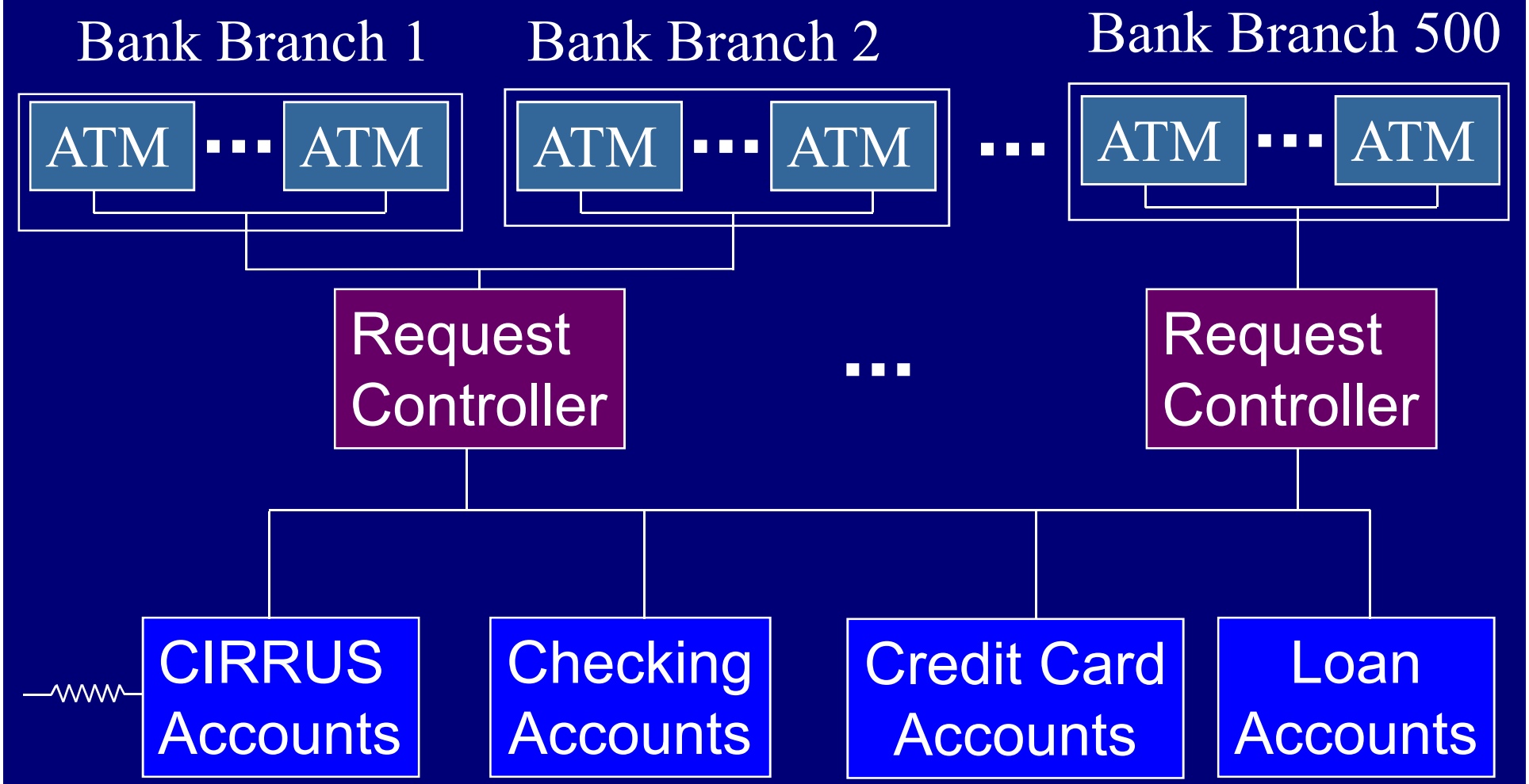
Request Controller

Transaction Server

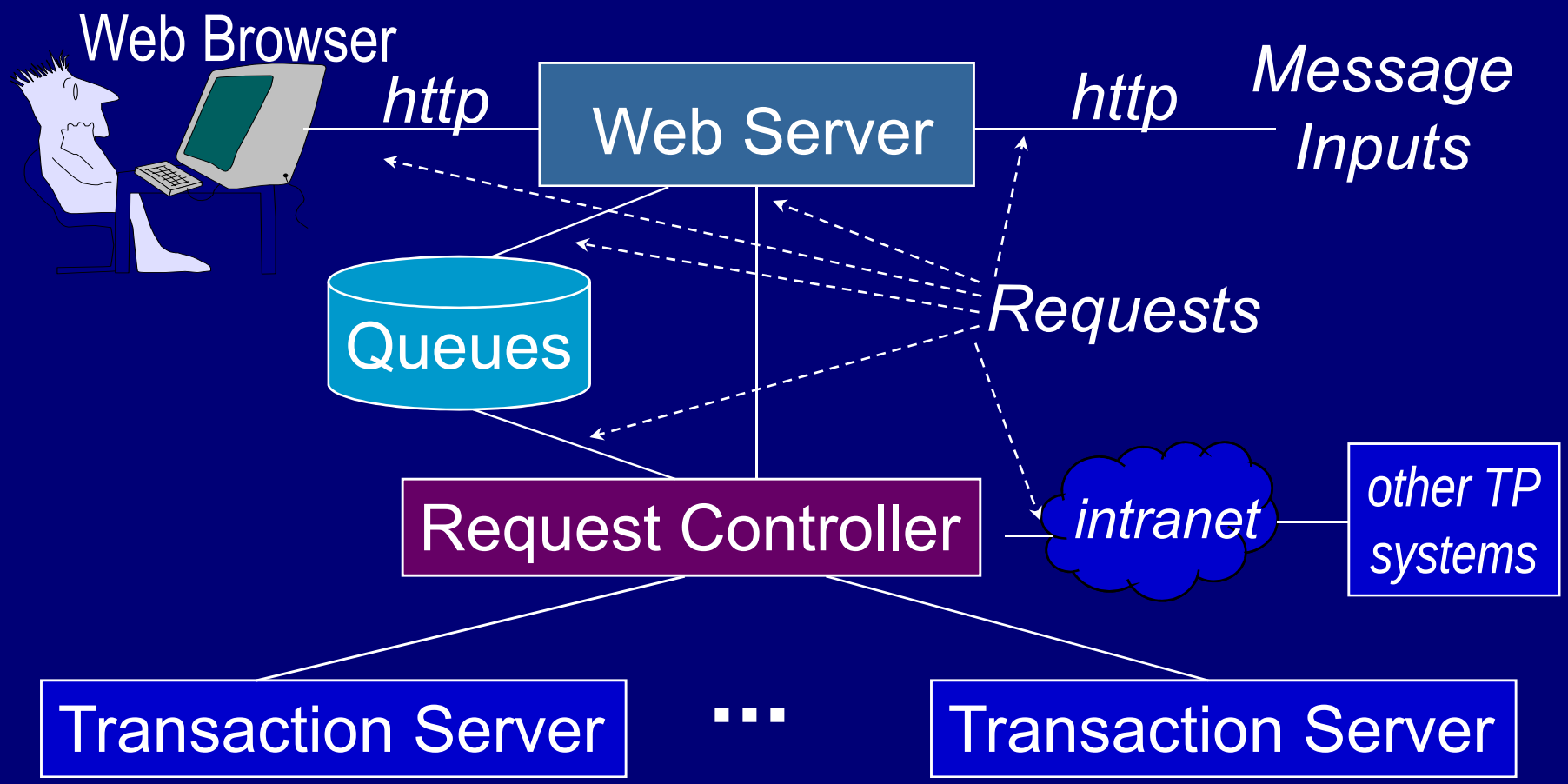
...

Transaction Server

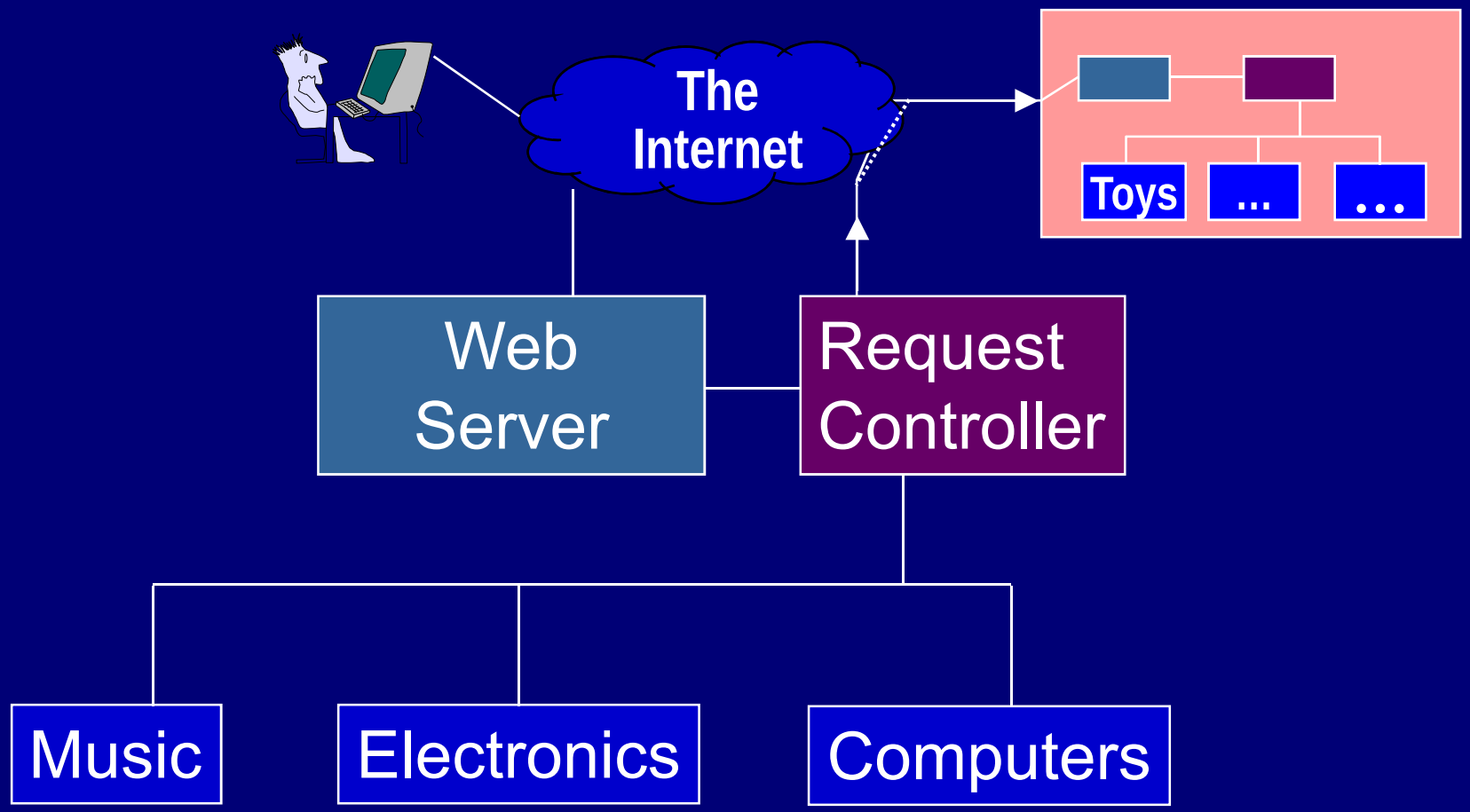
Automated Teller Machine (ATM) Application Example



Application Server Architecture

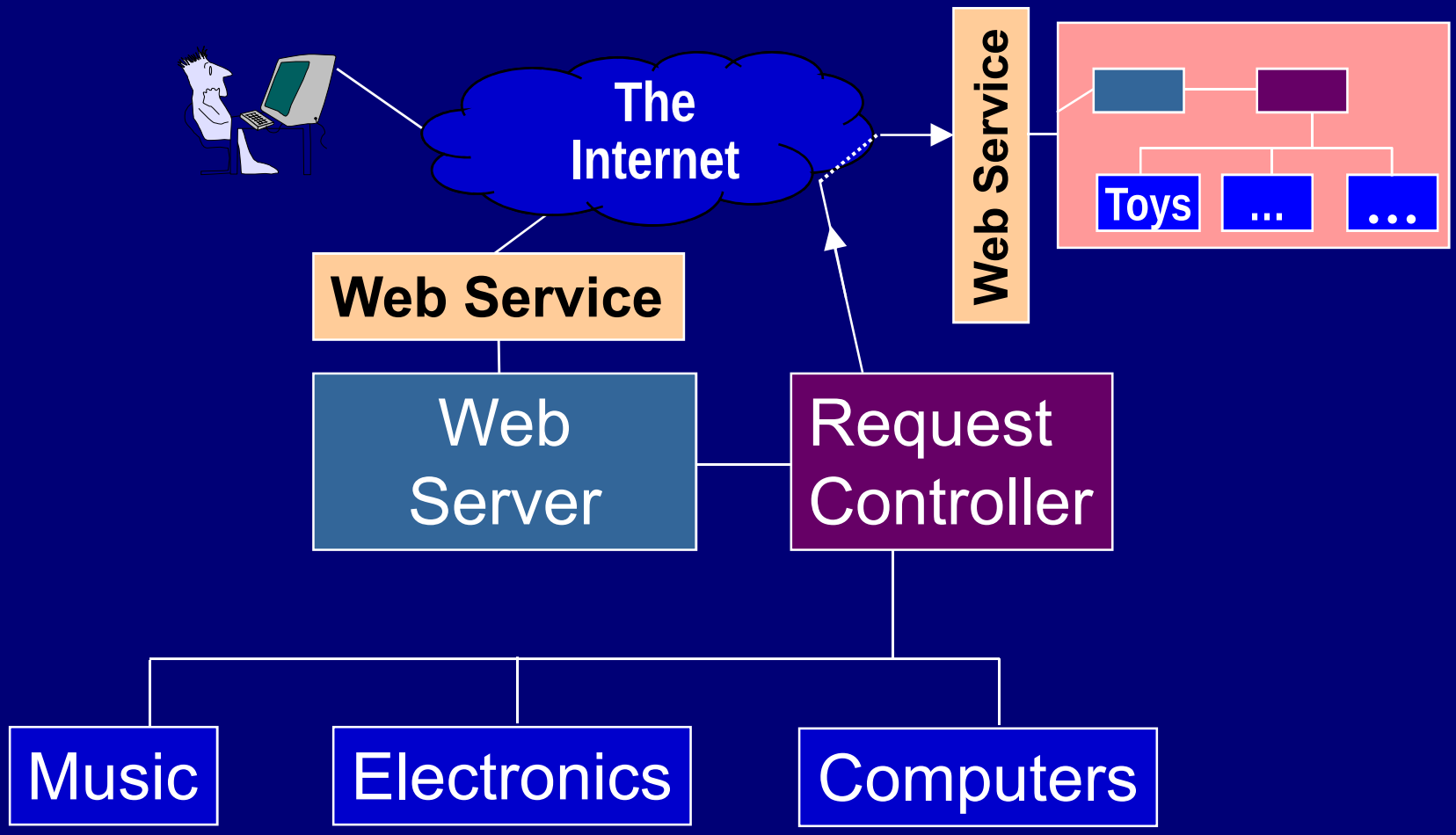


Internet Retailer



Service Oriented Architecture (SOA)

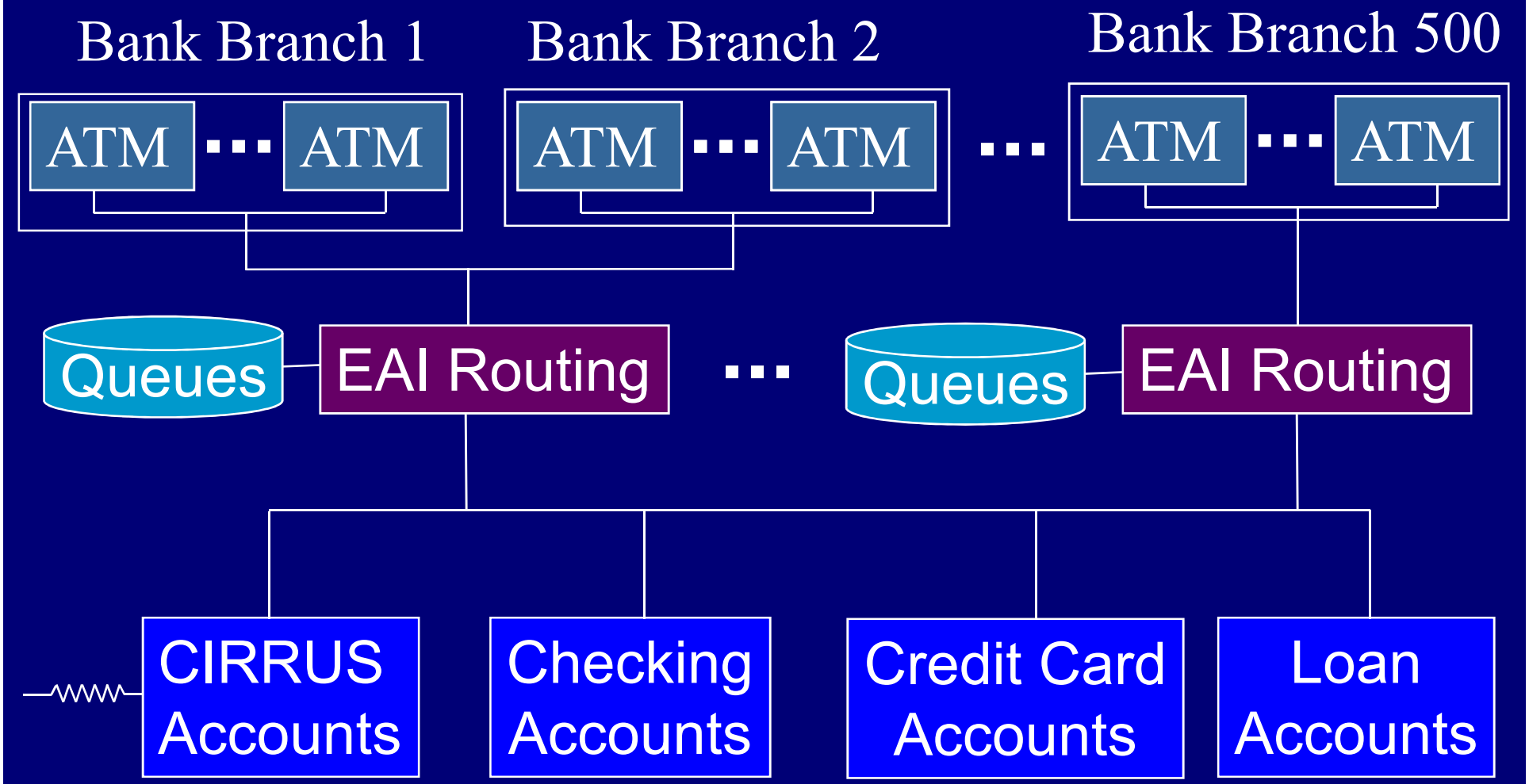
- Web services - interface and protocol standards to do app server functions over the internet.



Enterprise Application Integration (EAI)

- A software product to route requests between independent application systems. Often include
  - A queuing system
  - A message mapping system
  - Application adaptors (SAP, PeopleSoft, etc.)
- EAI and Application Servers address a similar problem, with different emphasis
- IBM Websphere MQ, TIBCO, Vitria, SeeBeyond

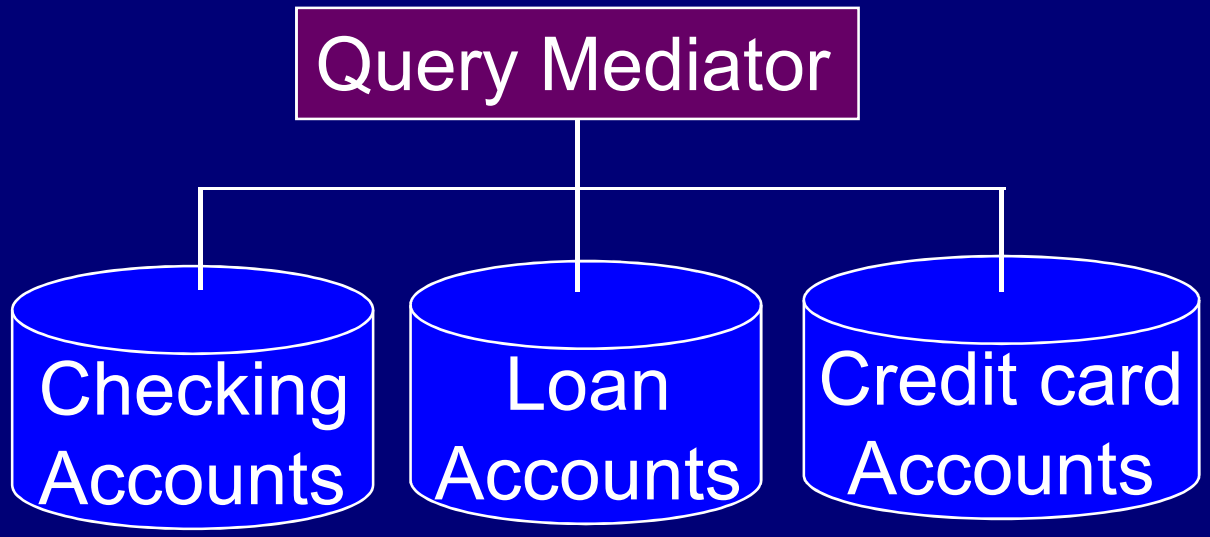
ATM Example with an EAI System



Workflow, or Business Process Mgmt

- A software product that executes multi-transaction long-running scripts (e.g. process an order)
- Product components
  - A workflow script language
  - Workflow script interpreter and scheduler
  - Workflow tracking
  - Message translation
  - Application and queue system adaptors
- Transaction-centric vs. document-centric
- Structured processes vs. case management
- IBM Websphere MQ Workflow, Microsoft BizTalk, SAP, Vitria, Oracle Workflow, FileNET, Documentum, ….

Data Integration Systems (Enterprise Information Integration)



- Heterogeneous query systems (mediators). It's database system software, but …
- It's similar to EAI with more focus on data transformations than on message mgmt
- There are hybrids, e.g., BEA AquaLogic

Transactional Middleware

- In summary, there are *many* variations that package different combinations of middleware features.
  - Application Server
  - Enterprise Application Integration
  - Business process management (aka Workflow)
  - Enterprise Server Bus
- New ones all the time, that defy categorization.

System Software Vendor's View

- TP is partly a component product problem
  - Hardware
  - Operating system
  - Database system
  - Application Server
- TP is partly a system engineering problem
  - Getting all those components to work together to produce a system with all those "ilities".
- This course focuses primarily on the Database System and Application Server

Outline

- ✓1. The Basics
  2. ACID Properties
  3. Atomicity and Two-Phase Commit
  4. Performance
  5. Styles of System

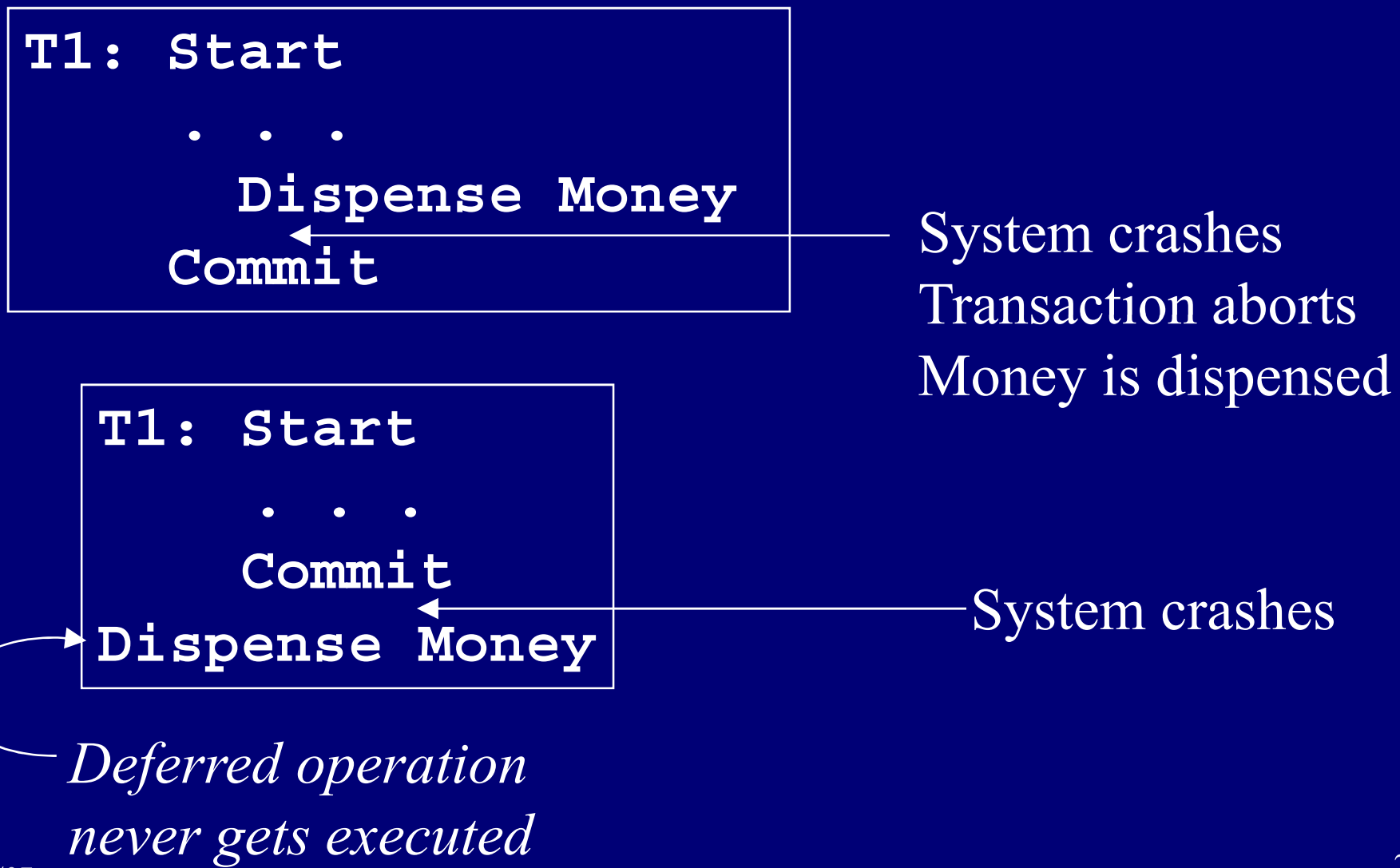
1.2 The ACID Properties

- Transactions have 4 main properties
  - Atomicity - all or nothing
  - Consistency - preserve database integrity
  - Isolation - execute as if they were run alone
  - Durability - results aren't lost by a failure

Atomicity

- All-or-nothing, no partial results.
  - E.g. in a money transfer, debit one account, credit the other. Either debit and credit both run, or neither runs.
  - Successful completion is called *Commit*.
  - Transaction failure is called *Abort.*
- Commit and abort are irrevocable actions.
- An Abort *undoes* operations that already executed
  - For database operations, restore the data's previous value from before the transaction
  - But some real world operations are not undoable. Examples - transfer money, print ticket, fire missile

Example - ATM Dispenses Money (a non-undoable operation)



Reading Uncommitted Output Isn't Undoable

```
T1: Start
      ...
    Display output
      ...
    If error, Abort
```

User reads output

…

User enters input

Brain transport

```
T2: Start
    Get input from display
      ...
    Commit
```

Compensating Transactions

- A transaction that reverses the effect of another transaction (that committed). For example,
  - "Adjustment" in a financial system
  - Annul a marriage
- Not all transactions have complete compensations
  - E.g. Certain money transfers
  - E.g. Fire missile, cancel contract
  - Contract law talks a lot about appropriate compensations
- ☝ A well-designed TP application should have a compensation for every transaction type

Consistency

Every transaction should maintain DB consistency

- Referential integrity - E.g. each order references an existing customer number and existing part numbers
- The books balance (debits = credits, assets = liabilities)

☝ *Consistency preservation is a property of a transaction, not of the TP system*
(unlike the A, I, and D of ACID)

- If each transaction maintains consistency, then serial executions of transactions do too.

Some Notation

- $r_i[x]$ = Read(x) by transaction T_i
- $w_i[x]$ = Write(x) by transaction T_i
- c_i = Commit by transaction T_i
- a_i = Abort by transaction T_i
- A *history* is a sequence of such operations, in the order that the database system processed them.

Consistency Preservation Example

T_1: Start;
A = Read(x);
A = A - 1;
Write(y, A);
Commit;

T_2: Start;
B = Read(x);
C = Read(y);
If (B > C+1) then B = B - 1;
Write(x, B);
Commit;

- Consistency predicate is x > y.
- Serial executions preserve consistency. Interleaved executions may not.
- H = $r_1[x]$ $r_2[x]$ $r_2[y]$ $w_2[x]$ $w_1[y]$
  - e.g. try it with x=4 and y=2 initially

Isolation

- Intuitively, the effect of a set of transactions should be the same as if they ran independently
- Formally, an interleaved execution of transactions is *serializable* if its effect is equivalent to a serial one.
- Implies a user view where the system runs each user's transaction stand-alone.
- Of course, transactions in fact run with lots of concurrency, to use device parallelism.

A Serializability Example

T_1: Start;
A = Read(x);
A = A + 1;
Write(x, A);
Commit;

T_2: Start;
B = Read(x);
B = B + 1;
Write(y, B);
Commit;

- H = $r_1[x]$ $r_2[x]$ $w_1[x]$ c_1 $w_2[y]$ c_2
- H is equivalent to executing T_2 followed by T_1
- Note, H is *not* equivalent to T_1 followed by T_2
- Also, note that T_1 started before T_2 and finished before T_2, yet the effect is that T_2 ran first.

Serializability Examples (cont'd)

- Client must control the relative order of transactions, using handshakes
  (wait for T_1 to commit before submitting T_2).
- Some more serializable executions:
  $r_1[x]\ r_2[y]\ w_2[y]\ w_1[x] \equiv T_1\ T_2 \equiv T_2\ T_1$

  $r_1[y]\ r_2[y]\ w_2[y]\ w_1[x] \equiv T_1\ T_2 \not\equiv T_2\ T_1$

  $r_1[x]\ r_2[y]\ w_2[y]\ w_1[y] \equiv T_2\ T_1 \not\equiv T_1\ T_2$
- Serializability says the execution is equivalent to *some* serial order, not necessarily to *all* serial orders

Non-Serializable Examples

- $r_1[x]$ $r_2[x]$ $w_2[x]$ $w_1[x]$ (*race condition*)
  - e.g. T_1 and T_2 are each adding 100 to x
- $r_1[x]$ $r_2[y]$ $w_2[x]$ $w_1[y]$
  - e.g. each transaction is trying to make x = y, but the interleaved effect is a swap
- $r_1[x]$ $r_1[y]$ $w_1[x]$ $r_2[x]$ $r_2[y]$ c_2 $w_1[y]$ c_1 (*inconsistent retrieval*)
  - e.g. T_1 is moving $100 from x to y.
  - T_2 sees only half of the result of T_1
- Compare to the OS view of synchronization

Durability

- When a transaction commits, its results will survive failures (e.g. of the application, OS, DB system … even of the disk).
- Makes it possible for a transaction to be a legal contract.
- Implementation is usually via a log
  - DB system writes all transaction updates to its log
  - to commit, it adds a record "commit(T_i)" to the log
  - when the commit record is on disk, the transaction is committed.
  - system waits for disk ack before acking to user

Outline

- ✓1. The Basics
- ✓2. ACID Properties
- 3. Atomicity and Two-Phase Commit
- 4. Performance
- 5. Styles of System

1.3 Atomicity and Two-Phase Commit

- Distributed systems make atomicity harder
- Suppose a transaction updates data managed by two DB systems.
- One DB system could commit the transaction, but a failure could prevent the other system from committing.
- The solution is the two-phase commit protocol.
- Abstract "DB system" by *resource manager* (could be a SQL DBMS, message mgr, queue mgr, OO DBMS, etc.)

Two-Phase Commit

- Main idea - all resource managers (RMs) save a durable copy of the transaction's updates before any of them commit.
- If one RM fails after another commits, the failed RM can still commit after it recovers.
- The protocol to commit transaction T
  - Phase 1 - T's coordinator asks all participant RMs to "prepare the transaction". Each participant RM replies "prepared" after T's updates are durable.
  - Phase 2 - After receiving "prepared" from *all* participant RMs, the coordinator tells all participant RMs to commit.

Two-Phase Commit System Architecture

1. Start transaction returns a unique *transaction identifier*
2. Resource accesses include the transaction identifier. For each transaction, RM registers with TM
3. When application asks TM to commit, the TM runs two-phase commit.

Outline

- ✓1. The Basics
- ✓2. ACID Properties
- ✓3. Atomicity and Two-Phase Commit
- 4. Performance
- 5. Styles of System

1.4 Performance Requirements

- Measured in max transaction per second (tps) or per minute (tpm), and dollars per tps or tpm.
- Dollars measured by list purchase price plus 5 year vendor maintenance ("cost of ownership")
- Workload typically has this profile:
  - 10% application server plus application
  - 30% communications system (not counting presentation)
  - 50% DB system
- TP Performance Council (*TPC*) sets standards
  - http://www.tpc.org.
- TPC A & B ('89-'95), now TPC C &W

TPC-A/B — Bank Tellers

- Obsolete (a retired standard), but interesting
- Input is 100 byte message requesting deposit/withdrawal
- Database tables = {Accounts, Tellers, Branches, History}

```
Start
   Read message from terminal (100 bytes)
   Read+write account record (random access)
   Write history record (sequential access)
   Read+write teller record (random access)
   Read+write branch record (random access)
   Write message to terminal (200 bytes)
Commit
```

- End of history and branch records are bottlenecks

The TPC-C Order-Entry Benchmark

Table	Rows/Whse	Bytes/row
Warehouse	1	89
District	10	95
Customer	30K	655
History	30K	46
Order	30K	24
New-Order	9K	8
OrderLine	300K	54
Stock	100K	306
Item	100K	82

- TPC-C uses heavier weight transactions

TPC-C Transactions

- New-Order
  - Get records describing a warehouse, customer, & district
  - Update the district
  - Increment next available order number
  - Insert record into Order and New-Order tables
  - For 5-15 items, get Item record, get/update Stock record
  - Insert Order-Line Record
- Payment, Order-Status, Delivery, Stock-Level have similar complexity, with different frequencies
- tpmC = number of New-Order transaction per min.

Comments on TPC-C

- Enables apples-to-apples comparison of TP systems
- Does not predict how *your* application will run, or how much hardware you will need, or which system will work best on your workload
- Not all vendors optimize for TPC-C.
  - Some high-end system sales require custom benchmarks.

Typical TPC-C Numbers

- All numbers are highly sensitive to date submitted.
- $1 - $6 / tpmC for results released in 2006-2007.
  - Low end numbers are almost all MS SQL Server & Windows.
  - High end is mostly Oracle and IBM, Linux, BEA Tuxedo
- System cost $27K (HP) - $12M (IBM)
- Examples of high throughput (32 dual-core processors)
  - IBM, 4.0M tpmC, $12.0M, $2.97/tpmC
    (1/22/07 IBM AIX/DB2, MS Windows/COM+)
- Examples of low cost (MS SQL Server, Windows, COM+)
  - HP ProLiant, 18K tpmC, $28K, $1.57/tpmC, 10/19/04
  - Dell, 70K tpmC, $66K, $0.96/tpmC, 3/9/07

Coming Soon, TPC-E

- Approved March 07
- Replaces TPC-C, it's database-centric
- A brokerage application
- More realistic disk configuration (smaller % of price)

Outline

- ✓1. The Basics
- ✓2. ACID Properties
- ✓3. Atomicity and Two-Phase Commit
- ✓4. Performance
- 5. Styles of System

1.5 Styles of Systems

- TP is System Engineering
- Compare TP to other kinds of system engineering …
- Batch processing - S*ubmit* a job and receive file output.
- Real time - *Submit requests* that have a deadline
- Data warehouse - *Submit queries* to a shared database, populated from TP data sources
- TP - *Submit a request* to run a transaction

TP vs. Batch Processing (BP)

- A BP application is usually uniprogrammed so serializability is trivial. TP is multiprogrammed.
- BP performance is measured by throughput. TP is also measured by response time.
- BP can optimize by sorting transactions by the file key. TP must handle random transaction arrivals.
- BP produces new output file. To recover, re-run the app.
- BP has fixed and predictable load, unlike TP.
- But, where there is TP, there is almost always BP too.
  - TP gathers the input. BP post-processes work that has weak response time requirements
  - So, TP systems must also do BP well.

TP vs. Real Time (RT)

- RT has more stringent response time requirements. It may control a physical process.
- RT deals with more specialized devices.
- RT doesn't need or use a transaction abstraction
  - usually loose about atomicity and serializability
- In RT, response time goals are usually more important than completeness or correctness. In TP, correctness is paramount.

TP and Data Warehouse

- Two usage scenarios
  - Populate the warehouse (extract, transform, load (ETL))
  - Run queries against the data warehouse
- Often long-running queries, usually with lower data integrity requirements than TP.
- TP systems provide the raw data for DSSs.

Outline

- ✓1. The Basics
- ✓2. ACID Properties
- ✓3. Atomicity and Two-Phase Commit
- ✓4. Performance
- ✓5. Styles of System

What's Next?

- This chapter covered TP system structure and properties of transactions and TP systems
- The rest of the course drills deeply into each of these areas, one by one.