Introduction: What is a Transaction

- **Transactions:**
  - Units of work upon a data (persistent data, transient state)
  - Bernstein: 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 and buy a ticket
    - Withdraw money from ATM
    - Verify a credit card sale
    - Carry out an order using on the Internet
    - Fire a missile
    - Download a video clip

What Makes Transactions Hard ?

- Reliability – system should rarely fail
- Availability – system must be up all the time
- Response time – system should not be slower
- Throughput – thousands of transactions per second
- Scalability – both small and Internet scale
- Configurability – above requirements and low cost
- Atomicity – no partial results
- Durability – a transaction is a legal contract
- Distribution – of users and data

What Makes Transactions Important ?

- **Well-defined and working abstraction at the level of databases.**
- **Most medium and large businesses use transactions for their production systems. The business cannot operate without it.**
- **A huge slice of the computer system market, over $50B a year. Probably the single largest application of computers.**
- **Enables Internet commerce. In 2002, Intel did 10% or $5B of contracting over electronic commerce.**

System Characteristics

- **Typically < 100 transaction types per application**

- **Transaction size has high variance**
  - 0-30 disk accesses
  - 10K - 1M instructions executed
  - 2-20 messages

- **A large scale example: airline reservations**
  - 150,000 active display devices
  - thousands of disk drives
  - 3000 transactions per second, peaks

Fault Tolerance: System Availability

- **Fault → Error → Failure**
- **Error latency**
- **Errors:**
  - Latent
  - Effective
- **Failures**
  - Soft (recoverable)
  - Hard
- **Mean time to failure (MTTF):**
  - Average amount of time till the next failure
- **Mean time to repair (MTTR):**
  - Average amount of time necessary for recovery

Fault Tolerance: System Availability

- **Availability of the system:**

  MTTF/(MTTF+MTTR)

- **Some systems are *very* sensitive to downtime**
  - Airline reservation, stock exchange, telephone switching

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%

Fault Tolerance: Error Avoidance and Correction

- **Error correction**
  - Latent error processing
  - Effective error processing
    - Failure correction or masking

- **Avoidance: Valid construction**
  - ISO 9001, …
  - Duplexing, N-plexing (HW methods)
  - Defensive programming
  - Well-constructed software based on well-defined abstractions
    - …
    - Transactions

Introduction: What Is a Transaction

- **Transactions = ACID operations**
  - **A**tomicity:
    - All or nothing property
    - Includes all messages, operations, …
  - **C**onsistency:
    - A correct transformation of the state
  - **I**solation:
    - A transaction is not aware of other transactions
  - **D**urability:
    - Once a transaction completes successfully (commits), its results survive some sort of failures

Atomicity

- **All-or-nothing, no partial results**
  - Classical debit/credit example:
    - A money transfer, debit one account, credit another
    - 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 (Durability)**
- **An Abort *undoes* operations that already executed**
  - For database operations, restore data's previous value from before the transaction
  - But some real world operations are not undoable
    - Examples: transfer money, print ticket, destroy house, fire missile
    - Do such **real actions** at the end of transaction (as a part of commit)

Transactions: An Example

- **Example 1: A very generic one**

```
begin_work;
        operation(object1);
        operation(object2);
        …
commit_work;

begin_work;
        operation(object3);
        operation(object4);
        …
abort_work;
```

- **Notes:**
  - Abort vs. Rollback
  - Dependency on execution environment

Transactions: Another Example

- **Example 2: SQL**

```
SELECT id FROM accounts WHERE balance < 200;

INSERT INTO accounts VALUES ((67890, 1000));

UPDATE accounts SET balance=balance+1000 WHERE id=12345;

COMMIT_WORK;
```

Transactions: Yet Another Example

- **Example 3: Enterprise JavaBeans (EJB)**

```
try {
    Account acc1 = homeInterface.create(…);
    Account acc2 = homeInterface.create(…);
    UserTransaction tx = JNDIService.lookup(…);
    tx.begin();                 // Thread becomes associated with tx
    acc1.add(1000);             //
    acc2.add(-1000);            // Operations performed on behalf of  tx
    tx.commit();
} catch (TransactionAbortedException e) {
    …
}
```

Example - ATM Dispenses Money
a non-undoable operation

```
TX: Start
     . . .
       Transfer Money
    Commit
```

System crashes
Transaction aborts
Money is transferred

```
TX: Start
     . . .
    Commit
Transfer Money
```

System crashes

Deferred operation
never gets executed

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 (which was committed)**
  - “Adjustment” in a financial system
  - Annul a marriage
- **Not all transactions have complete compensations**
  - Certain money transfers
  - Fire missile

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)
- ***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**
  - To prevent other transactions to see inconsistent temporary states, transactions are isolated
  - Serializability theory

Isolation

- **Concurrency control**
  - 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
  - Serializability theory
- **Locking-based schedulers**
- **Other schedulers**
- **Isolation levels**
- **Multiversion concurrency control**
- **…**

Durability

- **Externalization of transaction's effects**
  - Storing modified data
  - Sending messages
  - Commit or Abort cannot be revoked
- **Mostly implemented by resource managers**
  - Persistent stores
  - Databases

Transaction Processing Monitors

- A software product to create, execute and manage TP applications

- Takes an application written to process a single request and 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.
  - TP monitor helps system engineer deploy it to 10s/100s of servers and 10Ks of displays

- Includes an application programming interface and tools for program development and system management