COMP90050:Advanced Database Systems

Professor Rao Kotagiri

Lecture Set3

Transaction models

ACID (Atomicity, Consistency, Isolation, Durability) properties:

- Atomicity - all or none. This means the application would not be able to find reasons for failure directly. One may be able to find reasons of failures through other means - e.g., by looking at the system logs, or making a fake success.
- Consistency- this is in general a very hard problem:

  It is not computable in general

  Even if it is computable for a restricted language, it may not be practical -- meaning , huge computational requirement.

  Only restricted type of consistency can be guaranteed, e.g. serializable transactions which will be discussed later.

Transaction models ...

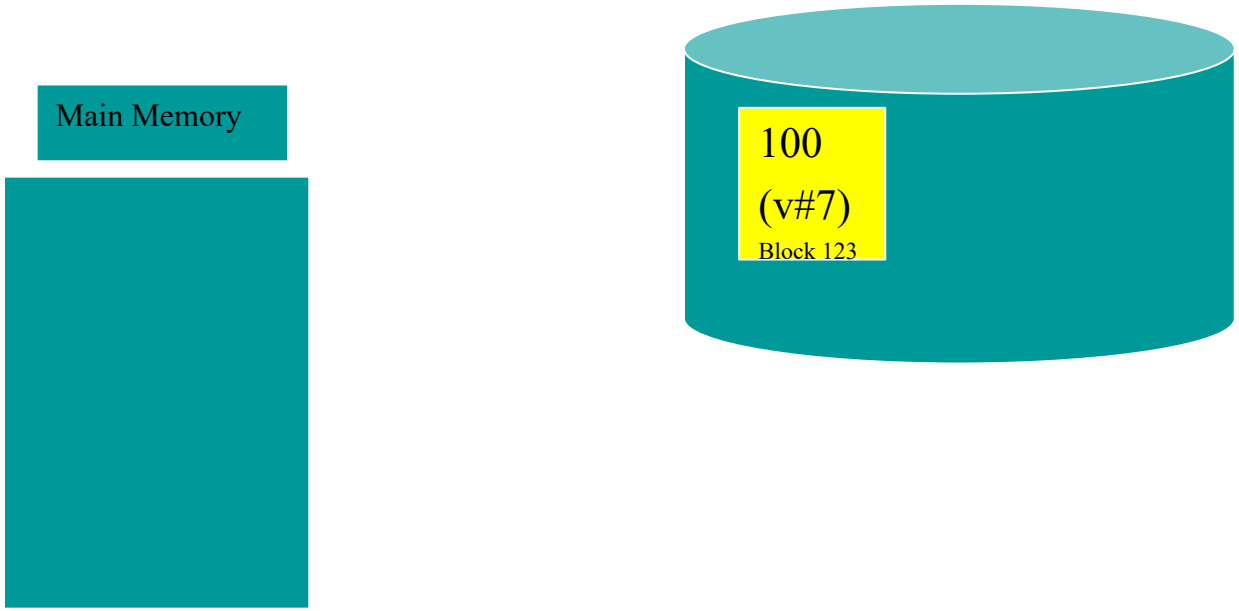
- Isolation- transaction are executed as if it was only the one on the system.
- Durability- the system should tolerate system failures and any committed updates should not be lost.
- Atomic Disk writes - either entire block is written correctly on disk or the contents of the block is unchanged. To achieve atomic disk writes we require duplex write.
  - a block of data is written two disk blocks *sequentially*
  - we can determine whether the contents of a disk block has an error or not by means of its CRC.
  - each block is associated with a version number.

Transaction models ...

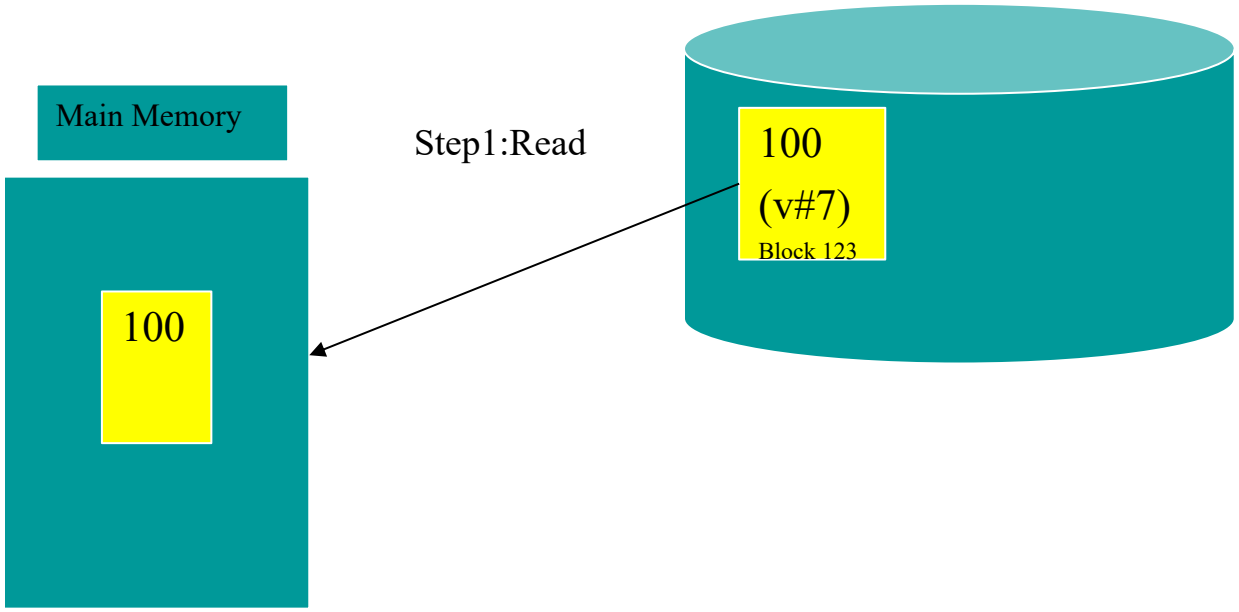
- the block which contains the latest version number is the one which contains most recent data.
- if one of the writes fail system can issue another write to the disk block that failed.
- it always guarantees at least one block has consistent data.

- Logged write- it is similar to the duplex write, except one of the writes goes to a log. This method is very efficient if the changes to a block are small. We will discuss an efficient method later in the subject.

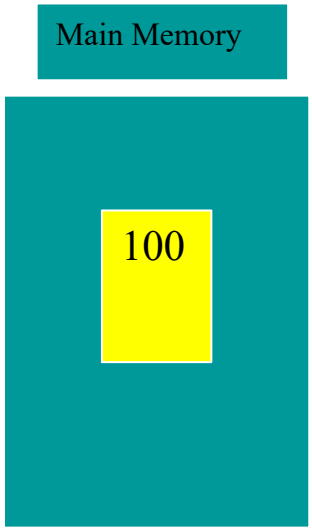
Updating disk block



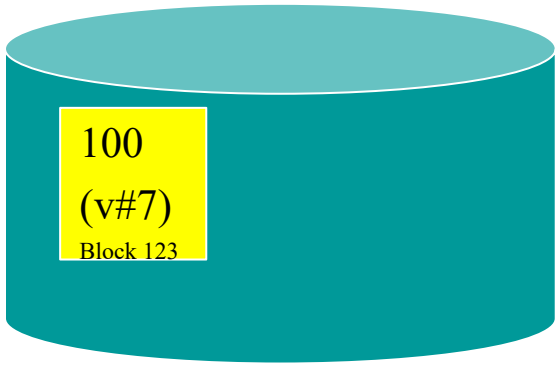
Updating disk block



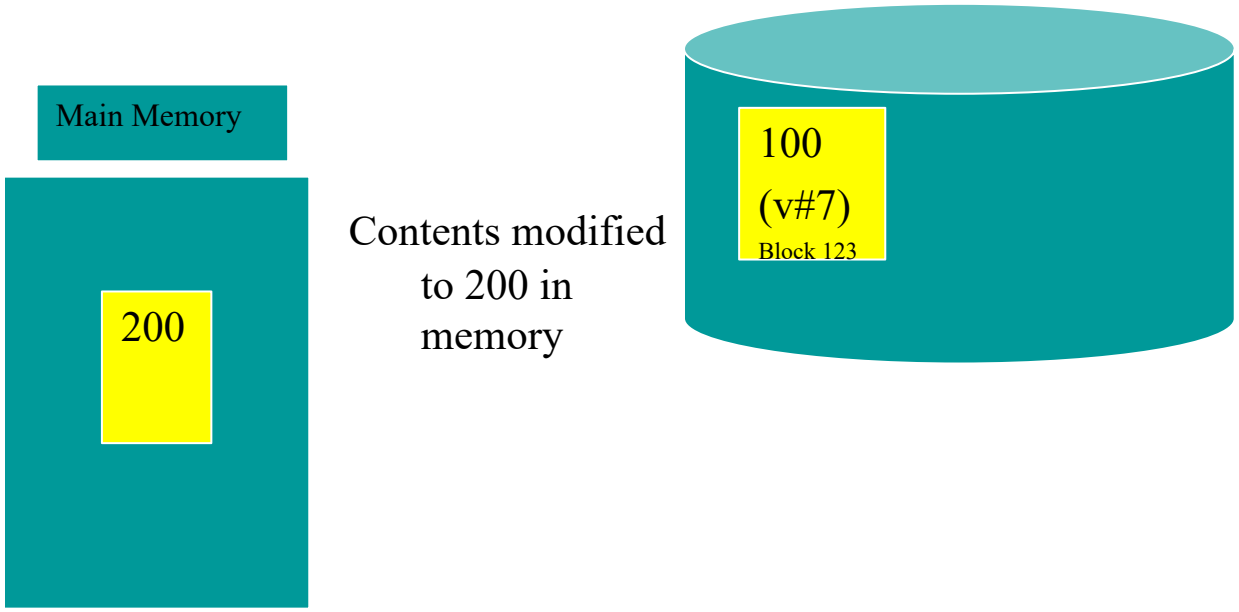
Updating disk block



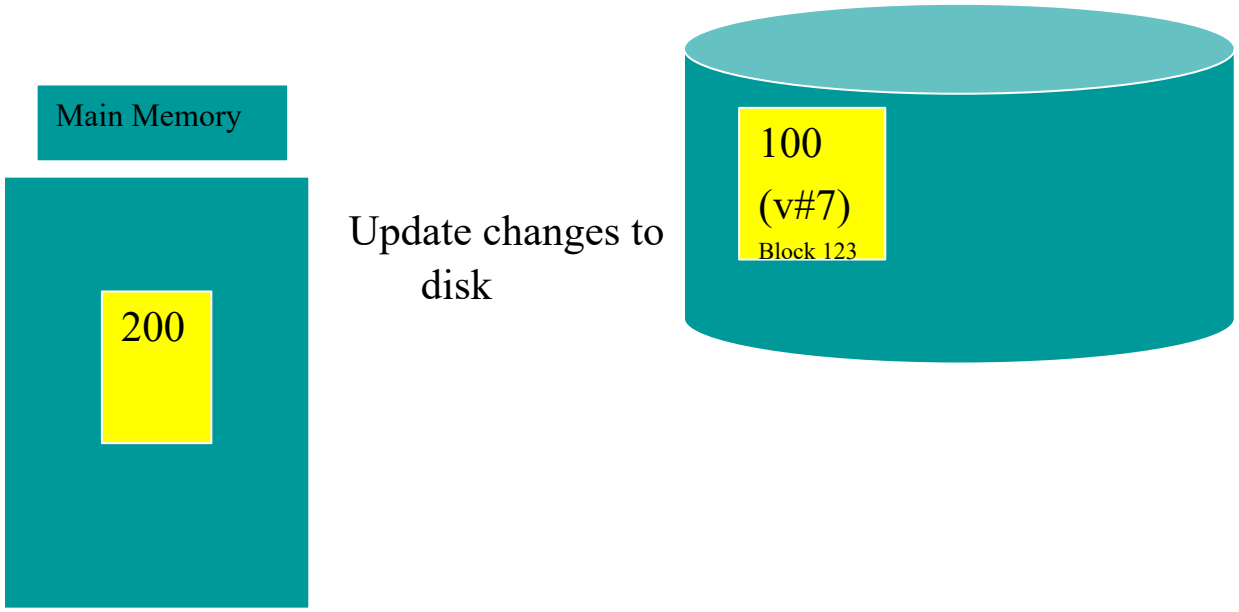
Step2:
Modify contents in memory to say 200



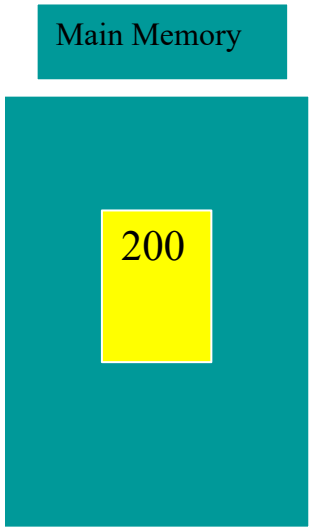
Updating disk block



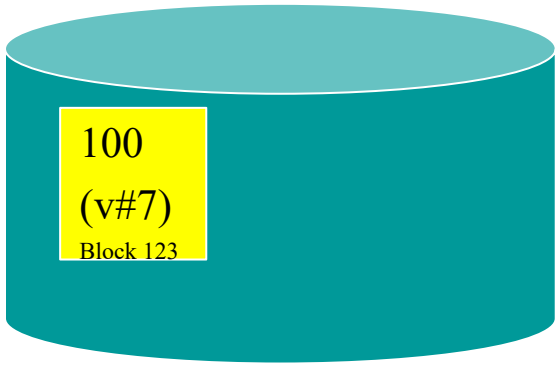
Updating disk block



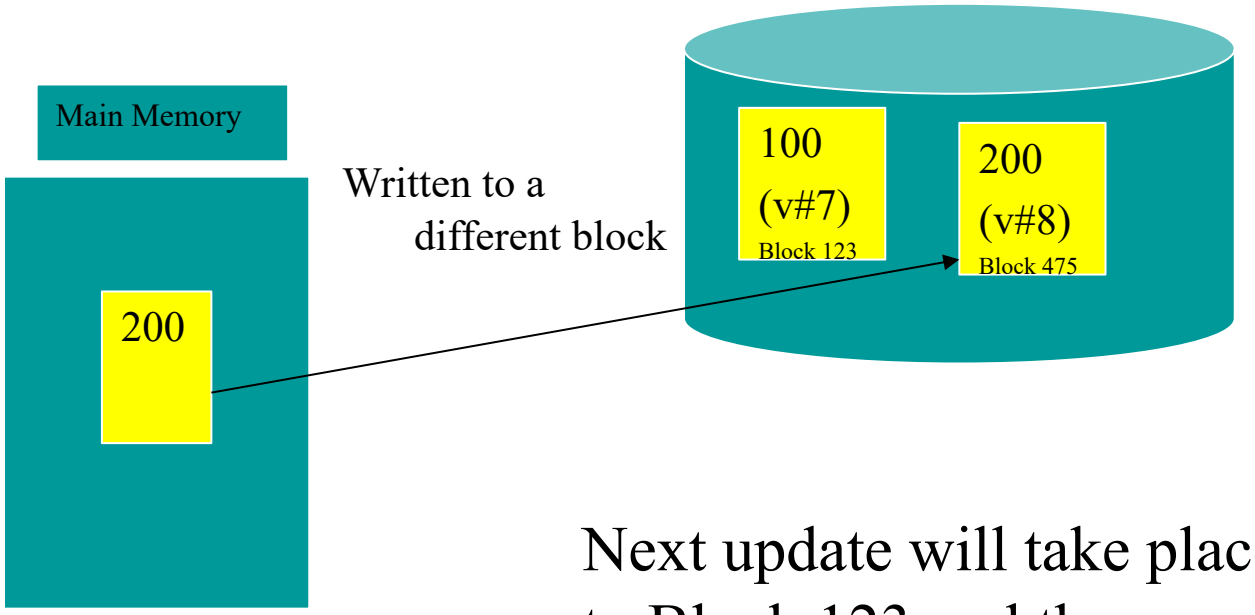
Updating disk block



Step3:
Write to disk in
a different block



Updating disk block



Next update will take place to Block 123 and the version number V#7 will be changed to v#9.

Cyclic Redundancy Check (CRC) generation

```
11010011101100 000 <--- input left shifted by 3 bits
1011           <--- divisor
01100011101100 000 <--- result
 1011          <--- divisor ...
00111011101100 000
  1011
00010111101100 000
   1011
00000011101100 000
      1011
00000000110100 000
        1011
00000000011000 000
          1011
00000000001110 000
            1011
00000000000101 000
             101 1
----------------
00000000000000 100 <---remainder (3 bits)
```

```
11010011101100 100 <--- input with CRC
1011           <--- divisor
01100011101100 100 <--- result
 1011          <--- divisor ...
00111011101100 100
……
00000000001110 100
            1011
00000000000101 100
             101 1
-----------------
             0 <--- remainder
```

$1011 = x^3 + x + 1$

Cyclic Redundancy Check (CRC) generation

CRC polynomial $x^{32} + x^{23} + x^7 + 1$

Most errors in communications or on disk happen contiguously, that is in burst in nature. The above CRC generator can detect all burst errors with a length less than or equal to 32 bits; 5 out of 10 billion burst errors with length 33 will be undetected; 3 out of 10 billion burst errors of length 34 or more will be undetected.

Example CRC polynomials

$x^5 + x^3 + 1$

$x^{15} + x^{14} + x^{11} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$

Transaction models ...

- Types of Actions

  Unprotected actions - no ACID property

  Protected actions- these actions are not externalised before they are completely done. These actions are controlled and can be rolled back if required. These have ACID property.

  Real actions - these are real physical actions once performed cannot be undone. In many situations, atomicity is not possible with real actions.

  E.g. firing two rockets as a single atomic action.

Embedded SQL example in C

(Open Database Connectivity)

```
int main()
{   exec sql INCLUDE SQLCA; /*SQL Communication Area*/
    exec sql BEGIN DECLARE SECTION;
     /* The following variables are used for communicating
    between SQL and C */
       int OrderID; /* Employee ID (from user) */
      int CustID; /* Retrieved customer ID */
      char SalesPerson[10] /* Retrieved salesperson name */
      char Status[6] /* Retrieved order status */
    exec sql  END DECLARE SECTION;
/* Set up error processing */
    exec sql WHENEVER SQLERROR GOTO query_error;
    exec sql WHENEVER NOT FOUND GOTO bad_number;
```

```
 /* Prompt the user for order number */
        printf ("Enter order number: ");
        scanf_s("%d", &OrderID);
/* Execute the SQL query */
        exec sql SELECT CustID, SalesPerson, Status
             FROM Orders
          WHERE OrderID = :OrderID // ":" indicates to refer to  C variable
          INTO :CustID, :SalesPerson, :Status;
/* Display the results */
        printf ("Customer number: %d\n", CustID);
        printf ("Salesperson: %s\n", SalesPerson);
        printf ("Status: %s\n", Status);
       exit();
 query_error:
         printf ("SQL error: %ld\n", sqlca->sqlcode); exit();
 bad_number:
         printf ("Invalid order number.\n"); exit(); }
```

- ## Host Variables

  These are the variables declared in a section enclosed by the BEGIN DECLARE SECTION and END DECLARE SECTION keywords. When sql needs access to these variable they are prefixed by a colon ":". The colon is essential to distinguish between host variables and database objects, for example tables and columns, that have the same name by the sql pre-processor.

- ## Data Types

  The data types supported by a DBMS and a host language can be quite different. This affects host variables because they play a dual role. On one hand, host variables are program variables, declared and manipulated by host language statements. On the other hand, they are used in embedded SQL statements to retrieve database data. If there is no host language type that corresponds to a DBMS data type, the DBMS automatically converts the data. However, because each DBMS has its own rules and idiosyncrasies associated with the conversion process, the host variable types must be chosen carefully.

- Error Handling

  The DBMS reports run-time errors to the applications program through an SQL Communications Area, or SQLCA (SQL Communication Area). In the example, the first embedded SQL statement is INCLUDE SQLCA. This tells the pre-processor to include the SQLCA structure in the program. This is required whenever the program will process errors returned by the DBMS. The WHENEVER...GOTO statement tells the pre-processor to generate error-handling code that branches to a specific label when an error occurs.

- Singleton SELECT

  The statement used to return the data is a singleton SELECT statement; that is, it returns only a single row of data. Therefore, the code example does not declare or use cursors.

Reference:

http://msdn.microsoft.com/enus/library/ms714570(VS.85).aspx

Flat Transaction

```
exec sql CREATE Table accounts (
        AccId           NUMERIC(9),
        BranchId        NUMERIC(9), FOREIGN KEY
          REFERENCES branches,
        AccBalance      NUMERIC(10),
        PRIMARY KEY(AccId));
**********************************
```

Flat Transaction ...

```
exec sql BEGIN DECLARE SECTION;
    long AccId, BranchId, TellerId, delta, AccBalance;
exec sql END DELCLARATION;

/* Debit/Credit Transaction*/
DCApplication()
{read input msg;
     exec sql BEGIN WORK;
     AccBalance = DodebitCredit(BranchId, TellerId,
        AccId, delta);
     send output msg;
     exec sql COMMIT WORK;
}
```

Flat Transaction ...

```
/* Withdraw money -- bank debits;  Deposit money – bank credits */
Long DoDebitCredit( long BranchId,
    long TellerId, long AccId, long AccBalance, long delta)
{
      exec sql UPDATE accounts
      SET AccBalance =AccBalance + :delta
      WHERE AccId = :AccId;

      exec sql SELECT AccBalance INTO  :AccBalance
      FROM accounts WHERE AccId = :AccId;

      exec sql UPDATE tellers
      SET TellerBalance = TellerBalance + :delta
      WHERE TellerId = :TellerId;

      exec sql UPDATE branches
      SET BranchBalance = BranchBalance + :delta
      WHERE BranchId = :BranchId;
```

Flat Transaction ...

```
    Exec sql INSERT INTO history(TellerId, BranchId,
       AccId, delta, time)
    VALUES( :TellerId, :BranchId, :AccId, :delta,
       CURRENT);
    return(AccBalance);
}
```

Flat Transaction ...

```
exec sql BEGIN DECLARE SECTION
long AccId, BranchId, TellerId,delta, AccBalance;
exec sql END DELCLARATION;
/* Debit/Credit Transaction */
DCApplication()
{read input msg;
      exec sql BEGIN WORK;
      AccBalance = DodebitCredit(BranchId, TellerId, AccId, delta);
      if (AccBalance < 0 && delta < 0)
         { exec sql ROLLBACK WORK;}
      else
         { send output msg;
         exec sql COMMIT WORK;
         }
}
```

Limitations of Flat Transactions

- Flat transactions do not model many real applications.

  E.g. airline booking

  BEGIN WORK

  S1: book flight from Melbourne to Singapore

  S2: book flight from Singapore to London

  S3: book flight from London to Dublin

  END WORK

  Problem: from Dublin if we cannot reach our final destination instead we wish to fly to Paris from Singapore and then reach our final destination.

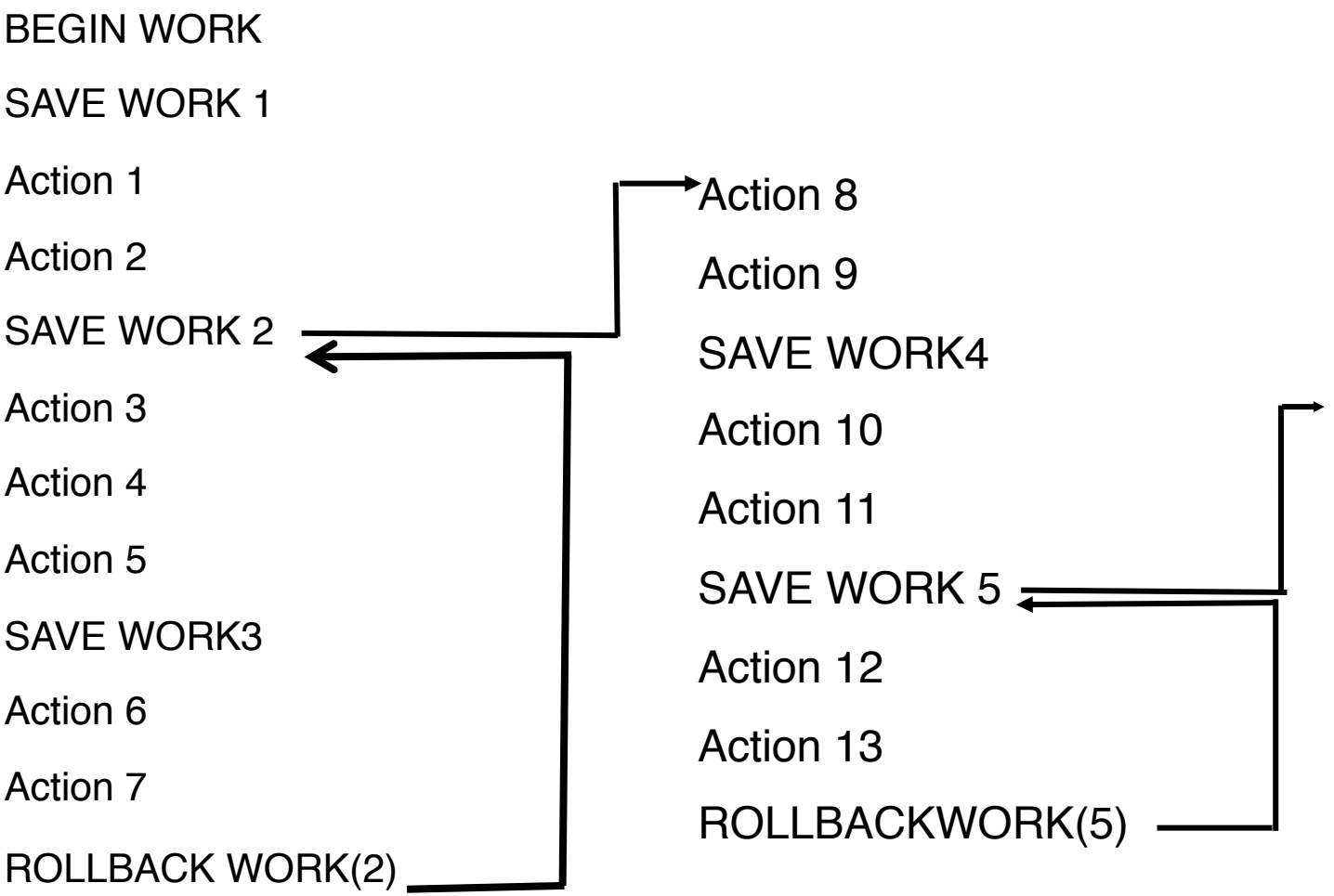
  If we roll back we need to re do the booking from Melbourne to Singapore which is a waste.

Limitations of Flat Transactions ...

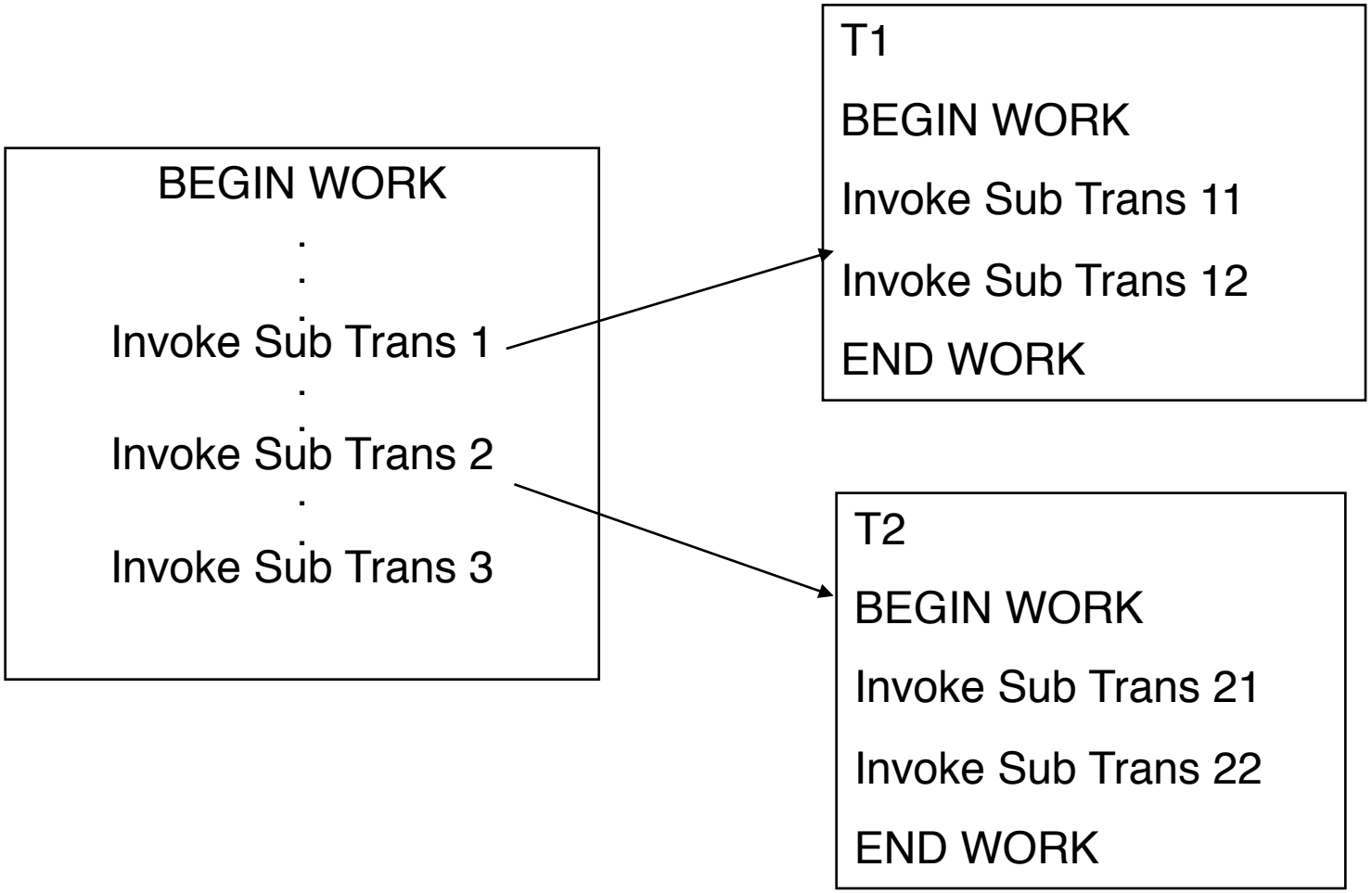
```
IncreaseSalary()
{   real percentRaise;
    receive(percentRaise);
    exec SQL BEGIN WORK;
        exec SQL UPDATE employee
        set salary = salary*(1+ :percentRaise)
        send(done);
    exec sql COMMIT WORK;
    return
}
```

This can be a very long running transaction. Any failure of transaction requires lot of unnecessary computation.

Transaction with save points



Nested Transactions



Nested Transaction Rules

Commit rule

- A subtransaction can either commit or abort, however, commit cannot take place unless the parent itself commits.
- Subtransactions have A, C, and I properties but not have D property unless all its ancestors commit.
- Commit of a sub transaction makes its results available only to its parents.

Roll back Rules

- If a subtransaction rolls back all its children are forced to roll back.

Visibility Rules

- Changes made by a sub transaction are visible to the parent only when the sub transaction commits. Whereas all objects of parent are visible to its children. Implication of this is that the parent should not modify objects while children are accessing them. This is not a problem as parent is not run in parallel with its children.

TP monitor

Computing styles

- Batch Processing:
  - Large Units of work;
  - Coarse grained resource allocation;
  - Sequential access patterns;
  - Less concurrent jobs;
  - Isolated execution;
  - Application is itself responsible for recovery -- e.g. by means of frequent program check pointing; running some system tools to recover lost data, etc.

- Time-sharing:
  - One process per terminal;
  - Coarse grained resource allocation;
  - Unpredictable resource requests,;
  - Sequential execution;
  - Hundreds of concurrent users;
  - Application/user is responsible for recovery -- e.g., frequent check pointing, running system tools to recover lost information, requesting the operator to restore the files lost.

TP monitor ...

Computing styles..

- Real-Time processing:
  - Event-driven operation;
  - Repetitive work load -- e.g. monitoring real devices and responding to exceptions, etc.;
  - Dynamic binding of devices to tasks -- e.g. each task may be directly responsible for monitoring or controlling a device and transferring information to those task that require the information;
  - Isolated execution;
  - High availability but does not necessarily have formal consistency;
  - High performance -- system should provide excellent response time which may involve guaranteeing deadline scheduling;

TP monitor ...

Computing styles..

- Client-Server Processing: this is very similar to time-sharing except for services are processed by sending messages to the server process. Client-server model implementation of a time-sharing system can increase the number of concurrent users that can be supported.
- Transaction -Oriented Processing:
  - Computations *share* data;
  - Variable requests;
  - Repetitive workload;
  - Usually simple computations (short transactions);
  - May support batch transaction;
  - Large concurrent users (OLTP);
  - Intelligent client processes;
  - High availability;
  - System based automatic recovery;
  - Auto load balancing

TP services

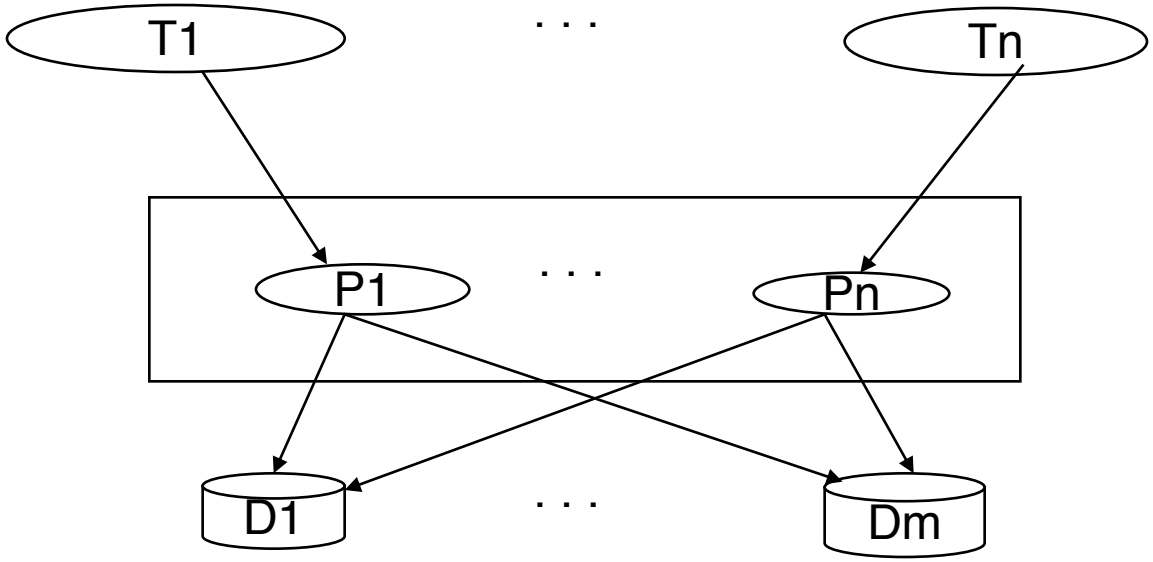
- Heterogeneity: If the application needs access to different database systems local ACID properties of individual database systems is not sufficient. Local TP monitor needs to interact with other TP monitors to ensure ACID property. A form of 2 phase commit protocol has to be employed for this purpose (this will be discussed later).
- Control communication: If the application communicates with other remote processes, the local TP monitor should maintain the communication status among the processes for it be able to recover from a crash. This was discussed earlier.

TP Services ...

- Terminal management: Since many terminals run client software the TP monitor should provide appropriate ACID property between the client and the server processes.
- Presentation service: this is similar to terminal management in the sense it has to deal with different presentation (user interface) software -- e.g. X-windows
- Context management: E.g. maintaining the sessions etc.
- Start/Restart : There is no difference between start and restart in TP based system.

TP process structure

One process per terminal performing all possible requests.



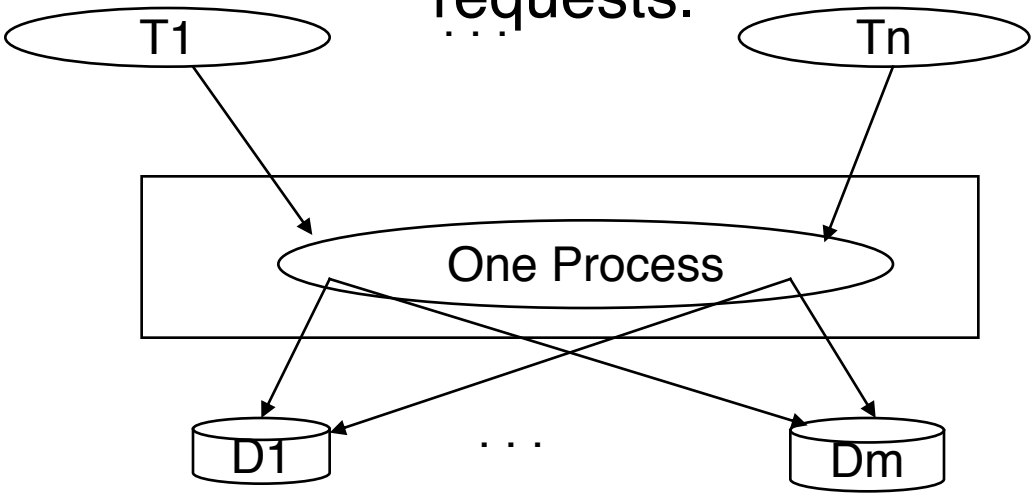
Very expensive if we have 20 000 terminals, 1000 files (relations) and

each process needing 50 blocks of data requires approximately:

20000 * 1000 * 50 = 10^9 blocks of data

TP process structure ...

One process for all terminals performing all possible requests.

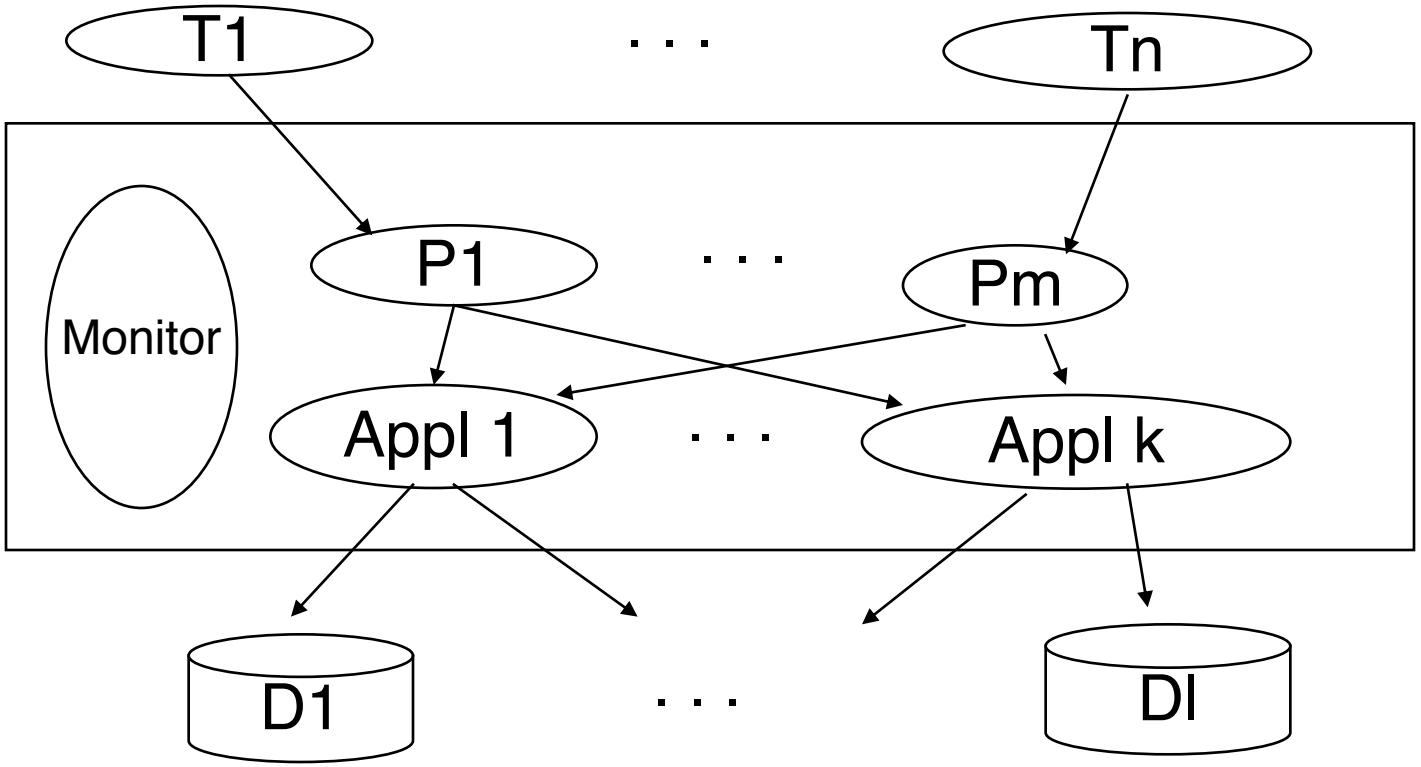


This model of implementation can be problematic for the following:

- Context switching due to scheduling can cause poor response;
- Single large program has to deal with all kinds of terminals.

TP Process structure ...

Multiple communication processes and servers



Components of A TP Monitor

- Presentation service : defines interface between the application and the devices it has to interact to.
- Queue management: performs the queuing of transactions.