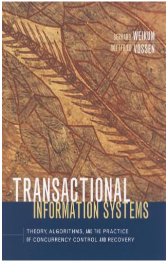
Transactional Information Systems:

Theory, Algorithms, and the Practice of Concurrency Control and Recovery



Gerhard Weikum and Gottfried Vossen

© 2002 Morgan Kaufmann
ISBN 1-55860-508-8

"Teamwork is essential. It allows you to blame someone else."(Anonymous)

Part I: Background and Motivation

Chapter 1: What Is It All About?

- **1.2 Application Examples**
- 1.3 System Paradigms
- 1.4 Virtues of Transactions
- 1.5 Architecture of Database Servers
- 1.6 Lessons Learned

"If I had had more time, I could written you a shorter letter"
(Blaise Pascal)

Application Examples

- OLTP, e.g., funds transfer
- E-commerce, e.g., Internet book store
- Workflow, e.g., travel planning & booking

OLTP Example: Debit/Credit

```
void main ( ) {
  EXEC SQL BEGIN DECLARE SECTION
    int b /*balance*/, a /*accountid*/, amount;
  EXEC SQL END DECLARE SECTION;
  /* read user input */
  scanf ("%d %d", &a, &amount);
  /* read account balance */
  EXEC SQL Select Balance into :b From Account
    Where Account_Id = :a;
/* add amount (positive for debit, negative for credit) */
  b = b + amount;
/* write account balance back into database */
  EXEC SQL Update Account
    Set Balance = :b Where Account_Id = :a;
  EXEC SQL Commit Work;
}
```

OLTP Example 1.1: Concurrent Executions

P1	Time	P2
Select Balance Into :b_1 From Account Where Account_Id = :a	1	
	/* b_1=0, a.Balance=100, b_2=0 */	
	2	**Select Balance Into :b_2 From Account Where Account_Id = :a**
	/* b_1=100, a.Balance=100, b_2=100 */	
b1 = b1-50	**3**	
	/* b_1=50, a.Balance=100, b_2=100 */	
	4	**$b_2 = b_2$ +100**
	/* b_1=50, a.Balance=100, b_2=200 */	
Update Account Set Balance = :b_1 Where Account_Id = :a	**5**	
	/* b_1=50, a.Balance=50, b_2=200 */	
	6	**Update Account Set Balance = :b_2 Where Account_Id = :a**
	/* b_1=50, a.Balance=200, b_2=200 */	

OLTP Example 1.1: Concurrent Executions

P1	Time	P2
Select Balance Into :b_1 From Account Where Account_Id = :a	1	
	/* b_1=0, a.Balance=100, b_2=0 */	
	2	**Select Balance Into :b_2 From Account Where Account_Id = :a**
	/* b_1=100, a.Balance=100, b_2=100 */	
b1 = b1-50	**3**	
	/* b_1=50, a.Balance=100, b_2=100 */	
	4	**b_2 = b_2 +100**
	/* b_1=50, a.Balance=100, b_2=200 */	
Update Account Set Balance = :b_1 Where Account_Id = :a	**5**	
	/* b_1=50, a.Balance=50, b_2=200 */	
	6	**Update Account Set Balance = :b_2 Where Account_Id = :a**
	/* b_1=50, a.Balance=200, b_2=200 */	

Observation: *concurrency or parallelism may cause inconsistencies, requires concurrency control for "isolation"*

OLTP Example 1.2: Funds Transfer

```
void main ( ) {
  /* read user input */
  scanf ("%d %d %d", &sourceid, &targetid, &amount);
  /* subtract amount from source account */
  EXEC SQL Update Account
    Set Balance = Balance - :amount Where Account_Id = :sourceid;
  /* add amount to target account */
  EXEC SQL Update Account
    Set Balance = Balance + :amount Where Account_Id = :targetid;
  EXEC SQL Commit Work; }
```

OLTP Example 1.2: Funds Transfer

```
void main ( ) {
  /* read user input */
  scanf ("%d %d %d", &sourceid, &targetid, &amount);
  /* subtract amount from source account */
  EXEC SQL Update Account
    Set Balance = Balance - :amount Where Account_Id = :sourceid;
  /* add amount to target account */
  EXEC SQL Update Account
    Set Balance = Balance + :amount Where Account_Id = :targetid;
  EXEC SQL Commit Work; }
```

Observation: *failures may cause inconsistencies,*
require recovery for "atomicity" and "durability"

E-Commerce Example

Shopping at Internet book store:

- client connects to the book store's server and starts browsing and querying the store's catalog
- client fills electronic shopping cart
- upon check-out client makes decision on items to purchase
- client provides information for definitive order (including credit card or cyber cash info)
- merchant's server forwards payment info to customer's bank credit or card company or cyber cash clearinghouse
- when payment is accepted, shipping of ordered items is initiated by the merchant's server and client is notified

E-Commerce Example

Shopping at Internet book store:

- client connects to the book store's server and
  starts browsing and querying the store's catalog
- client fills electronic shopping cart
- upon check-out client makes decision on items to purchase
- client provides information for definitive order
  (including credit card or cyber cash info)
- merchant's server forwards payment info to customer's bank
  credit or card company or cyber cash clearinghouse
- when payment is accepted,
  shipping of ordered items is initiated by the merchant's server
  and client is notified

*Observations: distributed, heterogeneous system
with general information/document/mail servers
and transactional effects on persistent data and messages*

Workflow Example

Workflows are (the computerized part of) **business processes**, consisting of a set of (automated or intellectual) **activities** with specified control and data flow between them (e.g., specified as a state chart or Petri net)

Conference travel planning:

- Select a conference, based on subject, program, time, and place. If no suitable conference is found, then the process is terminated.
- Check out the cost of the trip to this conference.
- Check out the registration fee for the conference.
- Compare total cost of attending the conference to allowed budget, and decide to attend only if the cost is within the budget.

Workflow Example

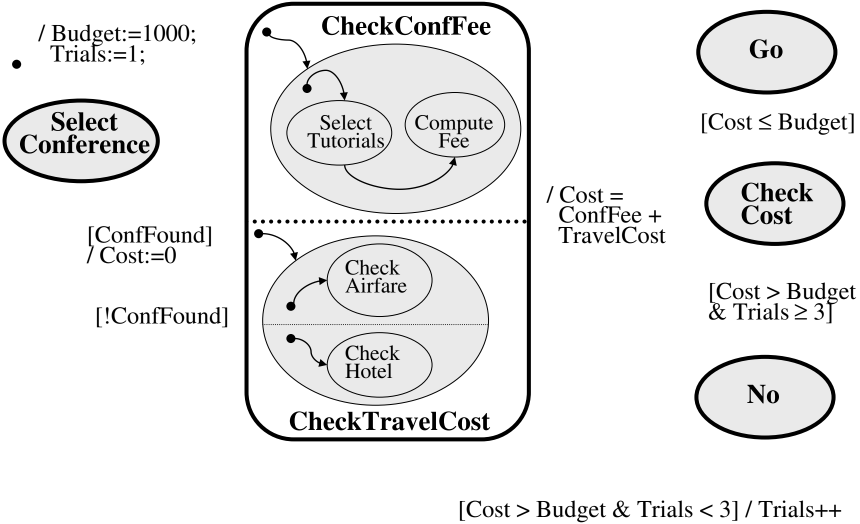
Workflows are (the computerized part of) **business processes**, consisting of a set of (automated or intellectual) **activities** with specified control and data flow between them (e.g., specified as a state chart or Petri net)

Conference travel planning:

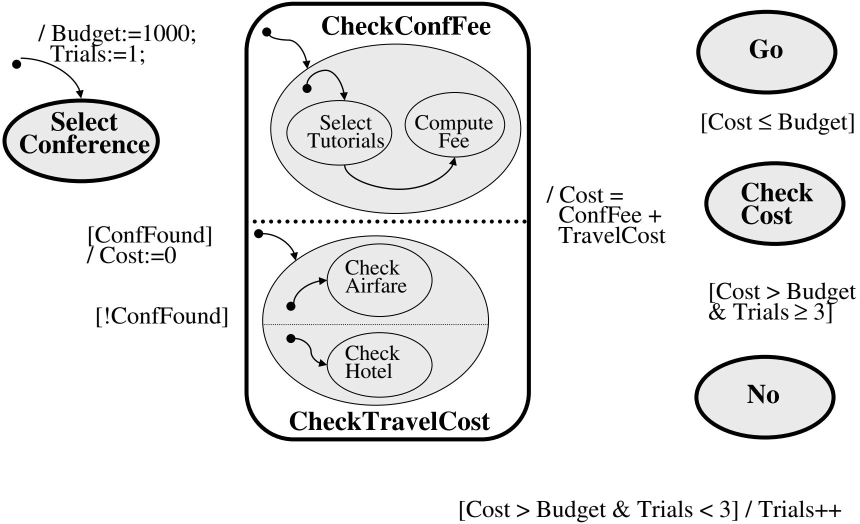
- Select a conference, based on subject, program, time, and place. If no suitable conference is found, then the process is terminated.
- Check out the cost of the trip to this conference.
- Check out the registration fee for the conference.
- Compare total cost of attending the conference to allowed budget, and decide to attend only if the cost is within the budget.

Observations: activities spawn transactions on information servers,
workflow state must be failure-resilient,
long-lived workflows are not isolated

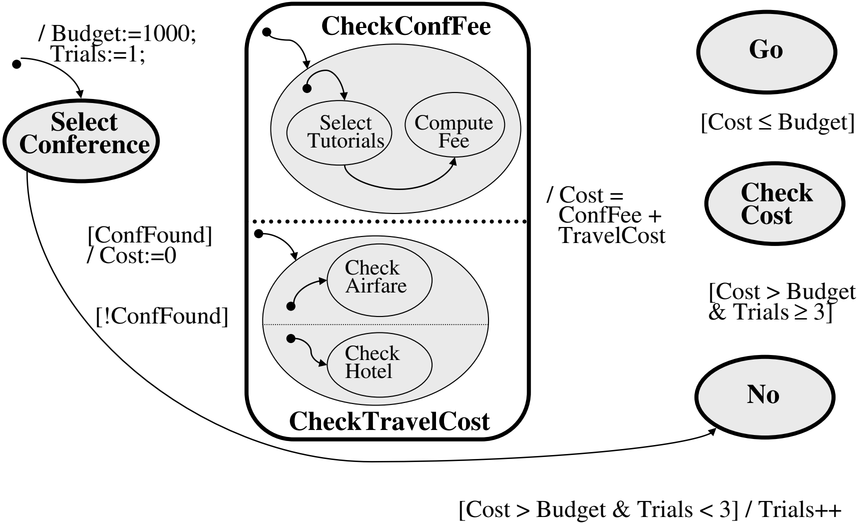
Example: Travel Planning Workflow



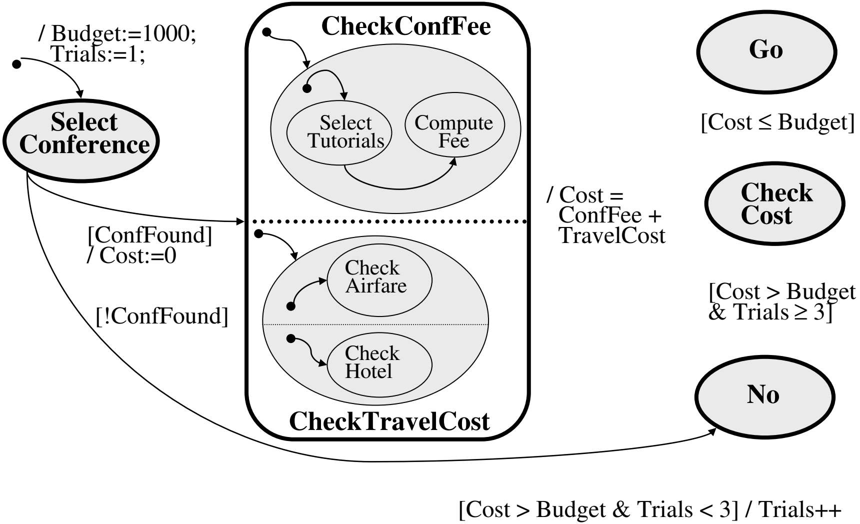
Example: Travel Planning Workflow



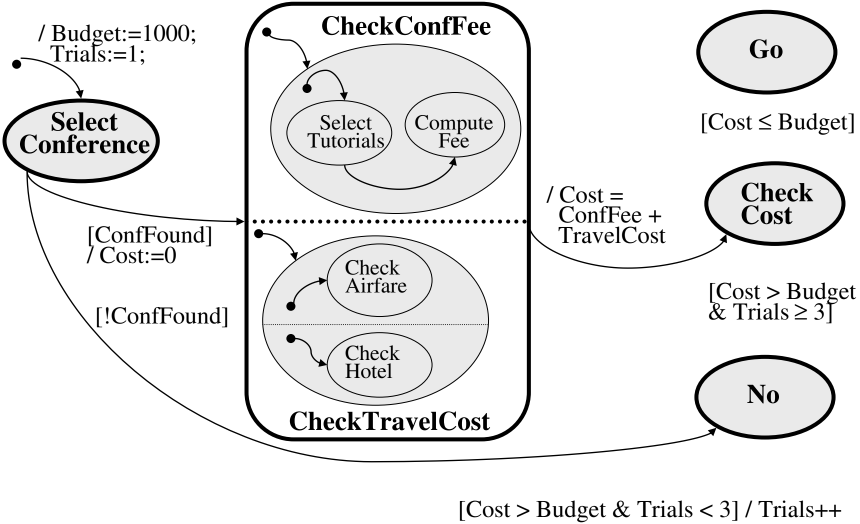
Example: Travel Planning Workflow



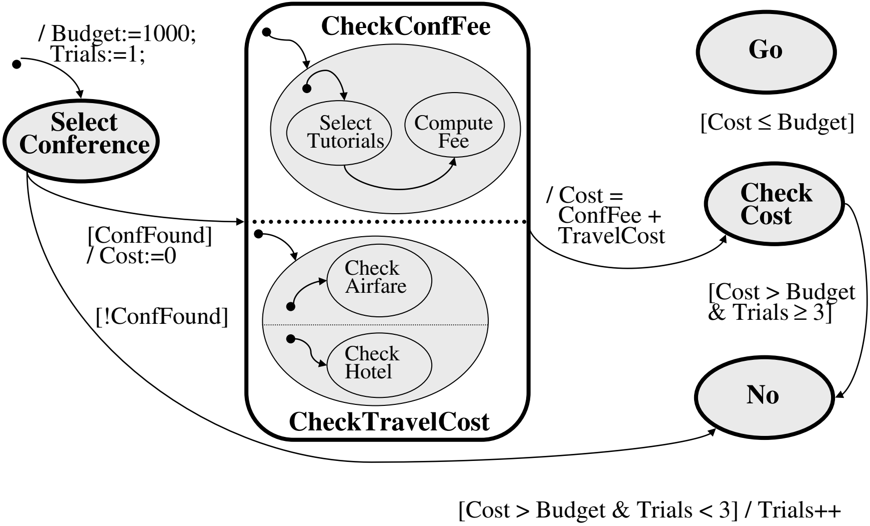
Example: Travel Planning Workflow



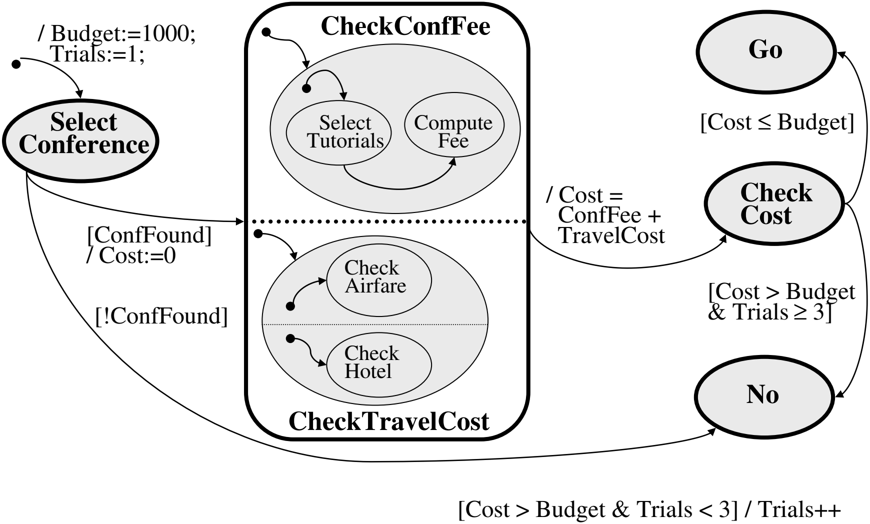
Example: Travel Planning Workflow



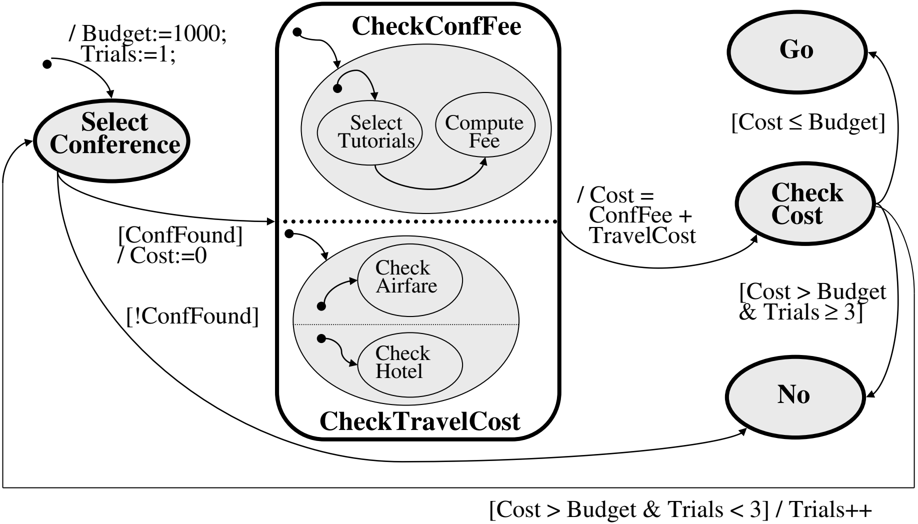
Example: Travel Planning Workflow



Example: Travel Planning Workflow



Example: Travel Planning Workflow



Introduction

- Application Examples
- **System Paradigms**
- Virtues of Transactions
- Architecture of Database Servers
- Lessons Learned

“If I had had more time, I could written you a shorter letter” (Blaise Pascal)

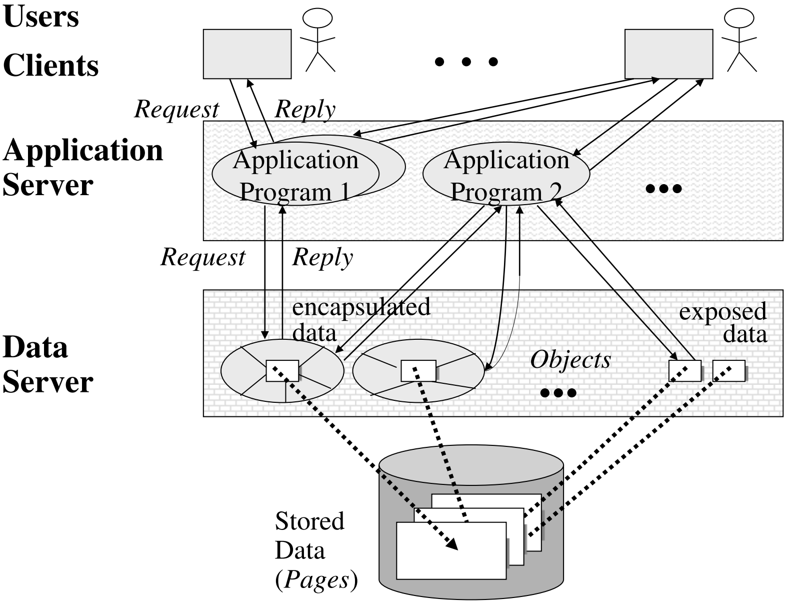
3-Tier System Architectures

- **Clients:**
  presentation (GUI, Internet browser)
- **Application server:**
  - application programs (business objects, servlets)
  - request brokering (TP monitor, ORB, Web server)
    based on **middleware** (CORBA, DCOM, EJB, SOAP, etc.)
- **Data server:**
  database / (ADT) object / document / mail / etc. servers

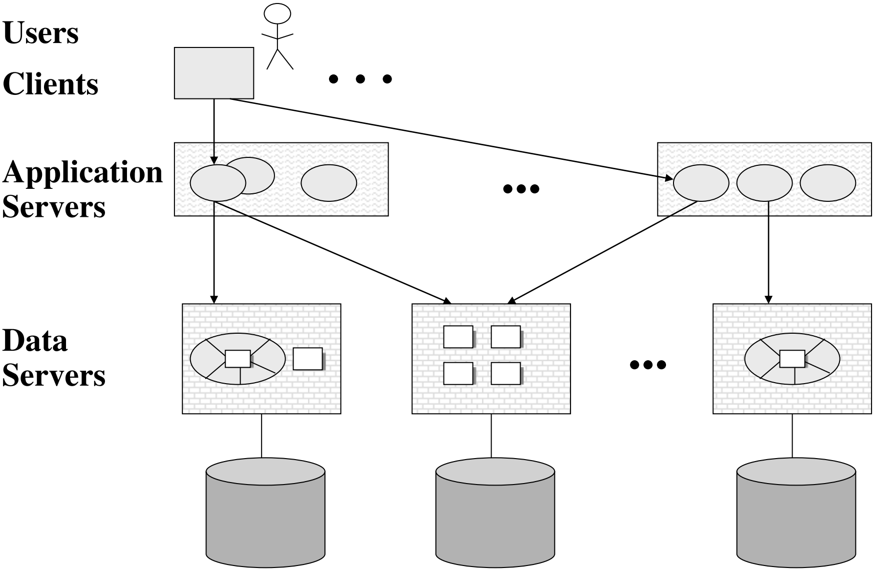
Specialization to 2-Tier Client-Server Architecture:

- Client-server with "fat" clients (app on client + ODBC)
- Client-server with "thin" clients (app on server, e.g., stored proc)

3-Tier Reference Architecture



System Federations



Introduction

- Application Examples
- System Paradigms
- **Virtues of Transactions**
- Architecture of Database Servers
- Lessons Learned

"If I had had more time, I could written you a shorter letter"
(Blaise Pascal)

ACID Properties of Transactions

- **Atomicity:**
  all-or-nothing effect,
  simple (but not completely transparent) failure handling
- **Consistency-preservation:**
  transaction abort upon consistency violation
- **Isolation:**
  only consistent data visible as if single-user mode,
  concurrency is masked to app developers
- **Durability (persistence):**
  committed effects are failure-resilient

Transaction programming interface ("ACID contract")

- begin transaction
- commit transaction ("commit work" in SQL)
- rollback transaction ("rollback work" in SQL)

Requirements on Transactional Servers

Server components:

- **Concurrency Control**
  guarantees isolation
- **Recovery:**
  guarantees atomicity and durability

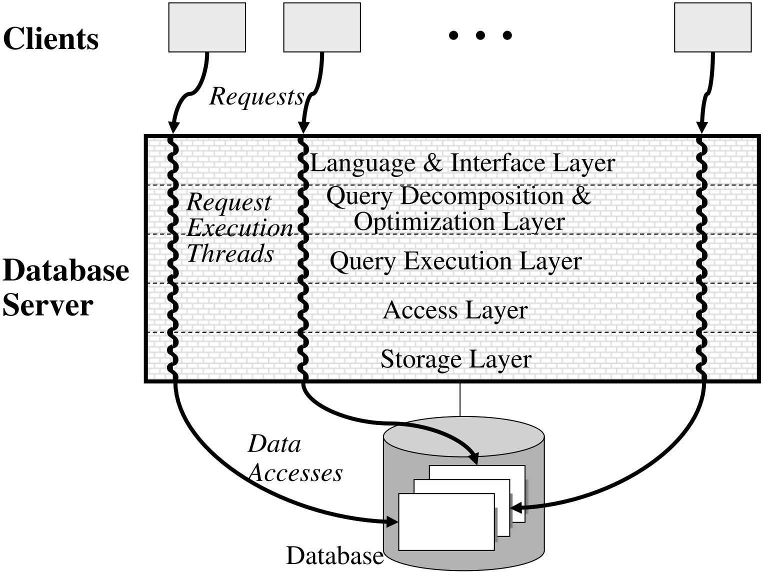
- **Performance:**
  high throughput (committed transactions per second)
  short response time
- **Reliability:**
  (almost) never lose data despite failures
- **Availability:**
  very short downtime
  almost continuous, 24x7, service

Introduction

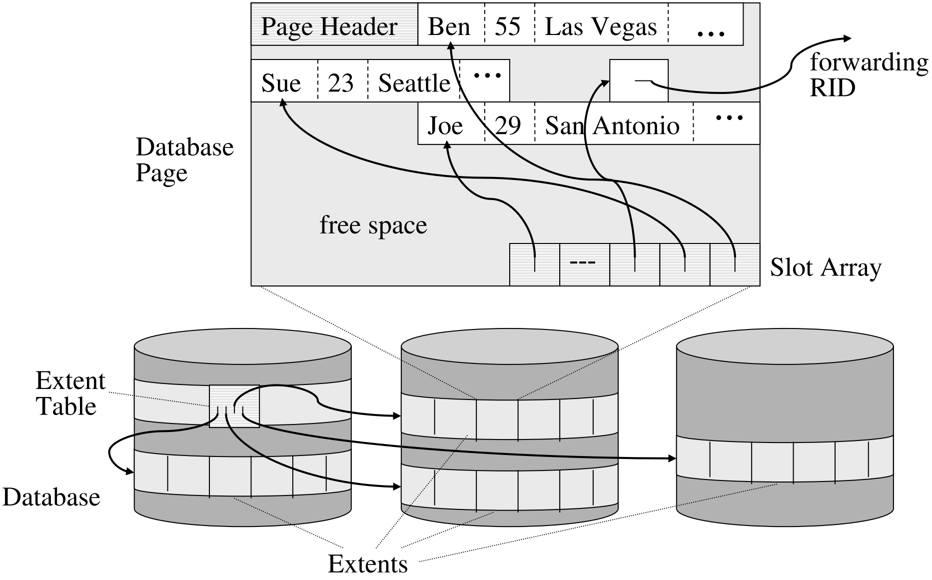
- Application Examples
- System Paradigms
- Virtues of Transactions
- **Architecture of Database Servers**
- Lessons Learned

“If I had had more time, I could written you a shorter letter” (Blaise Pascal)

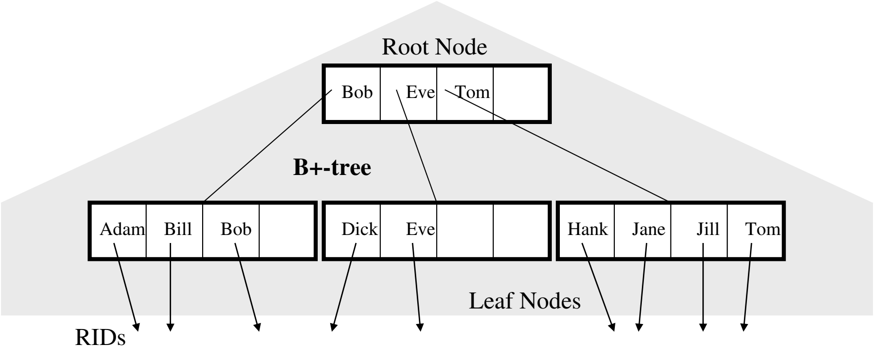
Database System Layers



Storage Structures



Access Structures

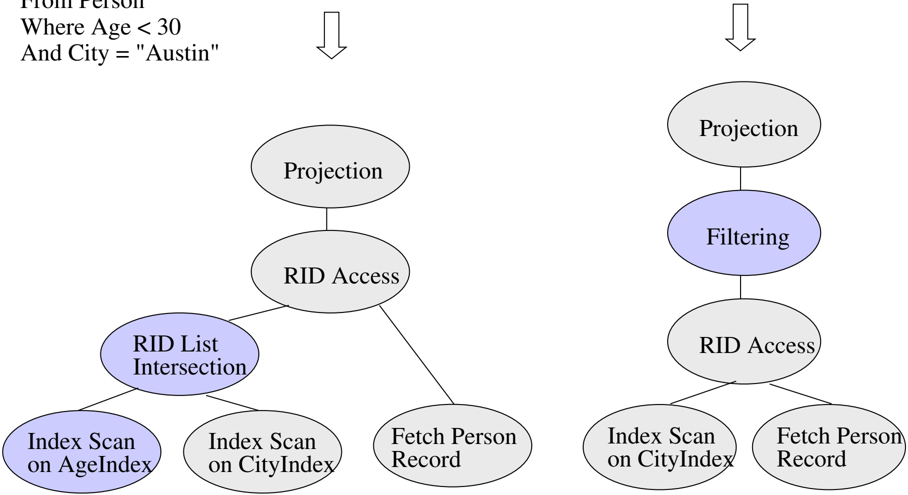


Search tree interface:

- lookup <index> where <indexed field> = <search key>
- lookup <index> where <indexed field>
  between <lower bound> and <higher bound>

Query Execution Plans

Select Name, City, Zipcode, Street
From Person
Where Age < 30
And City = "Austin"



Introduction

- Application Examples
- System Paradigms
- Virtues of Transactions
- Architecture of Database Servers
- **Lessons Learned**

"If I had had more time, I could written you a shorter letter"
(Blaise Pascal)

Lessons Learned

- **Benefits of ACID contract:**
  - For users: federation-wide data consistency
  - For application developers: ease of programming
- **Server obligations:**
  - Concurrency control
  - Recovery