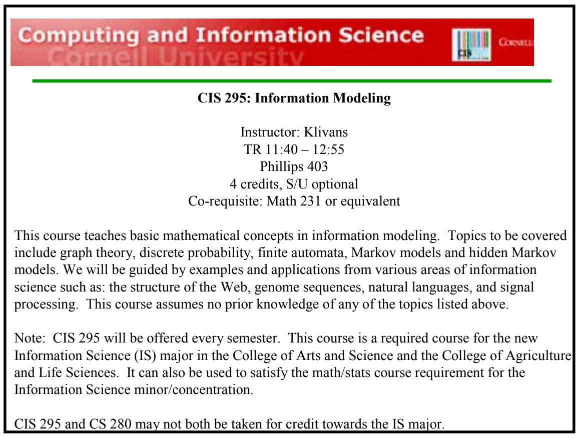
CIS 330: Applied Database Systems

Lecture 2: Technologies at the Three Tiers

Johannes Gehrke
johannes@cs.cornell.edu
http://www.cs.cornell.edu/johannes

Announcements

- Laptop handout starting today at 5:30pm in Upson B17.
  - Time for imaging the laptop
- Thursdays class starts at 2:55pm
- First two assignments will be online later today.
  - Exception: They are not handed out via the CMS; watch the course homepage.
- Slides from the first lectures will be online tonight
- New Information Science courses



Computing and Information Science
Cornell University

CIS 295: Information Modeling

Instructor: Klivans
TR 11:40 – 12:55
Phillips 403
4 credits, S/U optional
Co-requisite: Math 231 or equivalent

This course teaches basic mathematical concepts in information modeling. Topics to be covered include graph theory, discrete probability, finite automata, Markov models and hidden Markov models. We will be guided by examples and applications from various areas of information science such as: the structure of the Web, genome sequences, natural languages, and signal processing. This course assumes no prior knowledge of any of the topics listed above.

Note: CIS 295 will be offered every semester. This course is a required course for the new Information Science (IS) major in the College of Arts and Science and the College of Agriculture and Life Sciences. It can also be used to satisfy the math/stats course requirement for the Information Science minor/concentration.

CIS 295 and CS 280 may not both be taken for credit towards the IS major.

Computing and Information Science
Cornell University

CIS 435/635: Seminar on Applications of Information Science
Instructor: Paul Ginsparg
TR 11:40-12:55pm
Information Science seminar room, 301 College Avenue.
Grade options: Letter or S/U. 3 credits

This seminar course examines the technological, sociological, legal, financial and political aspects of information systems in the context of innovative applications. The course is designed as a series of case studies in information science, with presentations given by the people involved in designing or maintaining those systems. Examples will include arXiv, NSDL, NuPrl, the Legal Information Institute, Protomap, Dspace, and others created or maintained within Cornell, as well as some representative exterior resources.

The case studies will be augmented by readings and discussions of recent articles on technical components of the information systems, including machine learning tools, link and network analysis, metadata standards, document formats and clustering, data integrity, and natural language processing. Aspects of human and social interactions with these information systems to be considered will include copyright issues, privacy issues, public/private partnerships, and publishing models. The course prerequisites include background in computing, data structures, and programming at the level of CS 211 or equivalent, and experience in using information systems.

Computing and Information Science
Cornell University

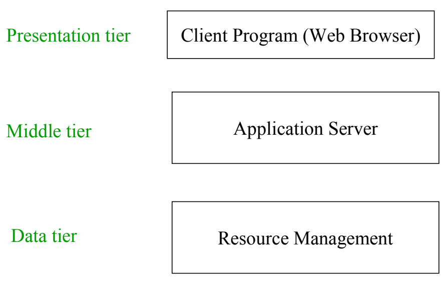
CIS 440: Social and Economic Data (also ILRLE 447 and ILRLE 740)

Instructor: John Abowd
MW 10:10-11:00am
F (lab) 10:10-11:00am OR 12:20-1:10pm
HO 162
4 credits

Prerequisites: one semester of calculus, the IS statistics requirement, at least one upper level social science course or permission of the instructor.

The course is designed to teach the student all the basics required to acquire and transform raw information into social and economic data. Legal, statistical, computing, and social science aspects of the data "manufacturing" process will all be treated. The formal US, Eurostat, OECD, and UN statistical infrastructure will be covered. Major private data sources will also be covered. Topics include: basic statistical principles of populations and sampling frames; acquiring data via samples, censuses, administrative records, and transaction logging; the law, economics and statistics of data privacy and confidentiality protection; data linking and integration techniques (probabilistic record linking; multivariate statistical matching); analytic methods in the social sciences. Grading will be based on a group term project.

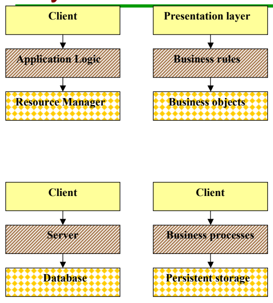
Last Class: Three-Tier Architectures



Technologies

Client Program *(Web Browser)*	*HTML Javascript XSLT*
Application Server *(Tomcat, Apache)*	*SQL, JSP, Servlets Cookies, EJB, XPath, web services*
Database System *(Microsoft SQL Server)*	*XML, Stored Procedures*

Layers and Tiers

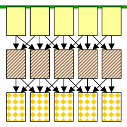


Client is any user or program that wants to perform an operation over the system. Clients interact with the system through a presentation layer

The application logic determines what the system actually does. It takes care of enforcing the business rules and establish the business processes. The application logic can take many forms: programs, constraints, business processes, etc.

The resource manager deals with the organization (storage, indexing, and retrieval) of the data necessary to support the application logic. This is typically a database but it can also be a text retrieval system or any other data management system providing querying capabilities and persistence.

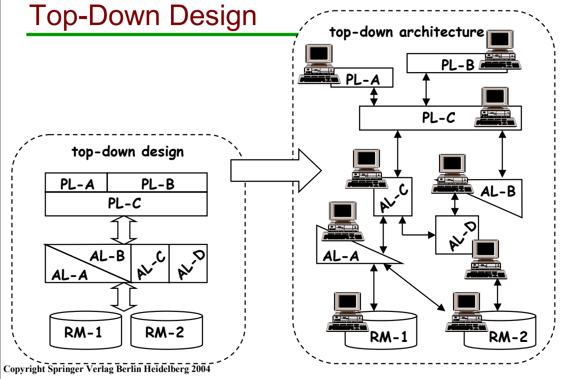
A Game of Boxes and Arrows



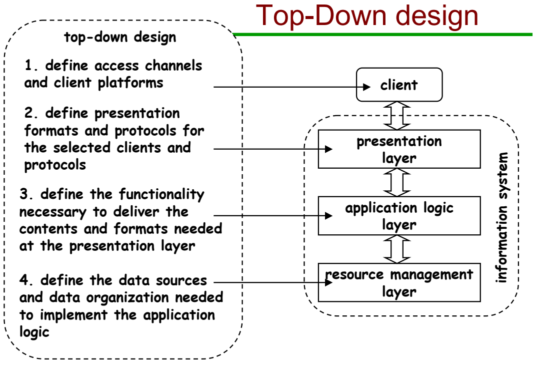
There is no problem in system design that cannot be solved by adding a level of indirection. There is no performance problem that cannot be solved by removing a level of indirection.

- Each box represents a part of the system.
- Each arrow represents a connection between two parts of the system.
- The more boxes, the more modular the system: more opportunities for distribution and parallelism. This allows encapsulation, component based design, reuse.
- The more boxes, the more arrows: more sessions (connections) need to be maintained, more coordination is necessary. The system becomes more complex to monitor and manage.
- The more boxes, the greater the number of context switches and intermediate steps to go through before one gets to the data. Performance suffers considerably.
- System designers try to balance the flexibility of modular design with the performance demands of real applications. Once a layer is established, it tends to migrate down and merge with lower layers.

Top-Down Design

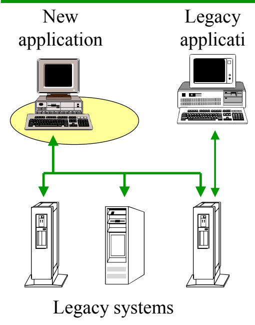


Top-Down design



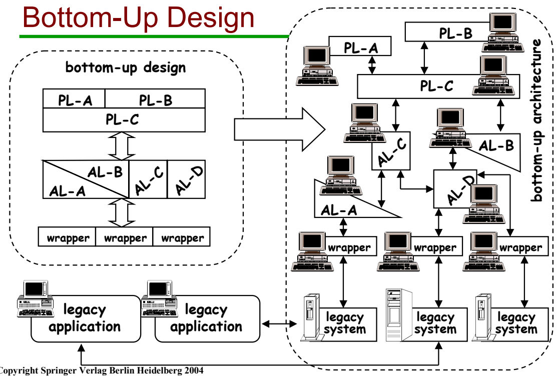
1. define access channels and client platforms
2. define presentation formats and protocols for the selected clients and protocols
3. define the functionality necessary to deliver the contents and formats needed at the presentation layer
4. define the data sources and data organization needed to implement the application logic

Bottom-Up Design

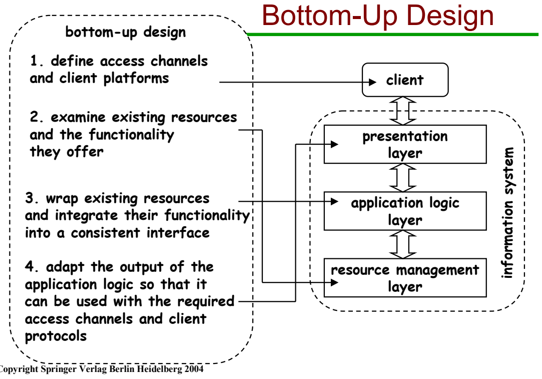


- In a bottom up design, many of the basic components already exist. These are stand alone systems which need to be integrated into new systems.
- The components do not necessarily cease to work as stand alone components. Often old applications continue running at the same time as new applications.
- This approach has a wide application because the underlying systems already exist and cannot be easily replaced.
- Much of the work and products in this area are related to middleware, the intermediate layer used to provide a common interface, bridge heterogeneity, and cope with distribution.

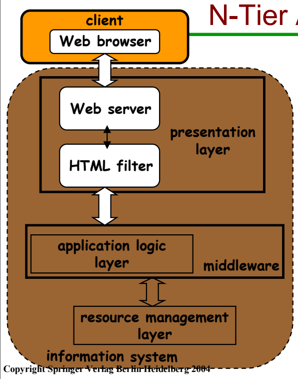
Bottom-Up Design



Bottom-Up Design

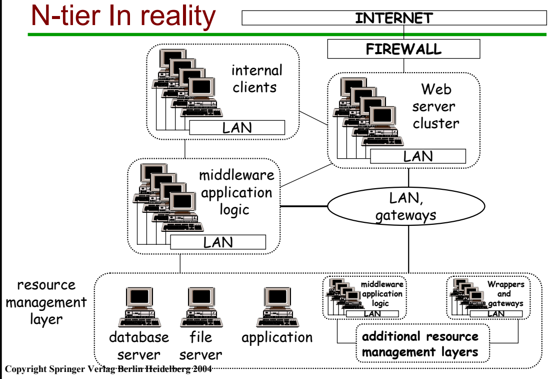

bottom-up design
1. define access channels and client platforms
2. examine existing resources and the functionality they offer
3. wrap existing resources and integrate their functionality into a consistent interface
4. adapt the output of the application logic so that it can be used with the required access channels and client protocols
client
presentation layer
application logic layer
resource management layer
information system

N-Tier Architectures



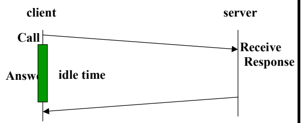
- N-tier architectures result from connecting several three tier systems to each other
- The addition of the Web layer led to the notion of "application servers", which was used to refer to middleware platforms supporting access through the Web

N-tier In reality



Blocking or Synchronous Interaction

- Traditionally, information systems use blocking calls Synchronous interaction requires both parties to be "on-line": the caller makes a request, the receiver gets the request, processes the request, sends a response, the caller receives the response.
- The caller must wait until the response comes back. but the interaction requires both client and server to be "alive" at the same time

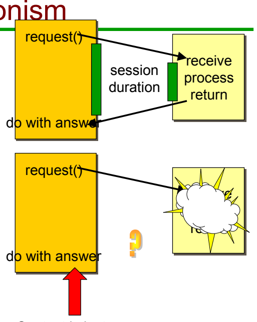


Disadvantages due to synchronization:

- Connection overhead
- Higher probability of failures
- Difficult to identify and react to failures
- It is not really practical for complex interactions

Overhead of Synchronism

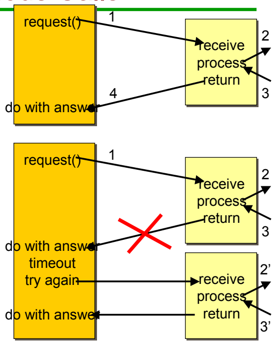
- Need to maintain a session between the caller and the receiver.
- Maintaining sessions is expensive. There is also a limit on how many sessions can be active at the same time
- For this reason, client/server systems often resort to connection pooling to optimize resource utilization
  - Have a pool of open connections
  - Allocate connections as needed
- Synchronous interaction requires a context for each call and a context management system for all incoming calls.



Context is lost
Needs to be restarted!!

Failures In Synchronous Calls

- If the client or the server fail, the context is lost.
  - If the failure occurred before 1, nothing has happened
  - If the failure occurs after 1 but before 2 (receiver crashes), then the request is lost
  - If the failure happens after 2 but before 3, side effects may cause inconsistencies
  - If the failure occurs after 3 but before 4, the response is lost but the action has been performed (do it again?)
- Who is responsible for finding out what happened?
- Finding out when the failure took place may not be easy. If there is a chain of invocations the failure can occur anywhere along the chain.



Two Solutions

ENHANCED SUPPORT

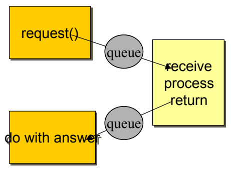
- Client/Server systems and middleware platforms provide a number of mechanisms to deal with the problems created by synchronous interaction:
  - Transactional interaction
  - Service replication and load balancing

ASYNCHRONOUS INTERACTION

- Using asynchronous interaction, the caller sends a message that gets stored somewhere until the receiver reads it and sends a response. The response is sent in a similar manner
- Asynchronous interaction can take place in two forms:
  - Non-blocking invocation
  - Persistent queues

Message Queuing

- Reliable queuing is an excellent complement to synchronous interactions:
  - Suitable to modular design: the code for making a request can be in a different module (even a different machine!) than the code for dealing with the response
  - Easier to design sophisticated distribution modes and it also helps to handle communication sessions in a more abstract way
  - More natural way to implement complex interactions between heterogeneous systems



This Lecture

- DBMS overview (continued)
- HTTP
- Cookies

Transactions

- A transaction is an atomic sequence of actions
- Each transaction must leave the system in a consistent state (if system is consistent when the transaction starts).
- The ACID Properties:
  - Atomicity
  - Consistency
  - Isolation
  - Durability

Concurrency Control for Isolation

(Start: A=$100; B=$100)

Consider two transactions:

- T1: START, A=A+100, B=B-100, COMMIT
- T2: START, A=1.06*A, B=1.06*B, COMMIT

The first transaction is transferring $100 from B's account to A's account. The second transaction is crediting both accounts with a 6% interest payment.

Database systems try to do as many operations concurrently as possible, to increase performance.

Example (Contd.)

(Start: A=$100; B=$100)

- Consider a possible interleaving (schedule):

```
T1: A=A+$100, B=B-$100 COMMIT
T2:                                  A=1.06*A, B=1.06*B COMMIT
End result: A=$106; B=$0
```

- Another possible interleaving:

```
T1: A=A+100,                                    B=B-100 COMMIT
T2:                 A=1.06*A, B=1.06*B COMMIT
End result: A=$112; B=$6
```

The second interleaving is incorrect! Concurrency control of a database system makes sure that the second schedule does not happen.

Ensuring Atomicity

- DBMS ensures atomicity (all-or-nothing property) even if the system crashes in the middle of a transaction.
- Idea: Keep a log (history) of all actions carried out by the DBMS while executing :
  - Before a change is made to the database, the corresponding log entry is forced to a safe location.
  - After a crash, the effects of partially executed transactions are undone using the log.

Recovery

- A DBMS logs all elementary events on stable storage. This data is called the log.
- The log contains everything that changes data: Inserts, updates, and deletes.
- Reasons for logging:
  - Need to UNDO transactions
  - Recover from a systems crash

Recovery: Example

(Simplified process)

- Insert customer data into the database
- Check order availability
- Insert order data into the database
- Write recovery data (the log) to stable storage
- Return order confirmation number to the customer

Why Store Data in a DBMS?

- Benefits
  - Transactions (concurrent data access, recovery from system crashes)
  - High-level abstractions for data access, manipulation, and administration
  - Data integrity and security
  - Performance and scalability

Data Model

- A data model is a collection of concepts for describing data.
- Examples:
  - ER model (used for conceptual modeling)
  - Relational model, object-oriented model, object-relational model (actually implemented in current DBMS)

The Relational Data Model

A relational database is a set of relations. Turing Award (Nobel Price in CS) for Codd in 1980 for his work on the relational model

- Example relation:
  Customers(cid: integer, name: string, byear: integer, state: string)

cid	name	byear	state
1	Jones	1960	NY
2	Smith	1974	CA
3	Smith	1950	NY

The Relational Model: Terminology

- Relation instance and schema
- Field (column)
- Record or tuple (row)
- Cardinality

cid	name	byear	state
1	Jones	1960	NY
2	Smith	1974	CA
3	Smith	1950	NY

Customer Relation (Contd.)

- In your enterprise, you are more likely to have a schema similar to the following:

  Customers(cid, identifier, nameType, salutation, firstName, middleNames, lastName, culturalGreetingStyle, gender, customerType, degrees, ethnicity, companyName, departmentName, jobTitle, primaryPhone, primaryFax, email, website, building, floor, mailstop, addressType, streetNumber, streetName, streetDirection, POBox, city, state, zipCode, region, country, assembledAddressBlock, currency, maritalStatus, bYear, profession)

Product Relation

- Relation schema:
  Products(pid: integer, pname: string, price: float, category: string)
- Relation instance:

pid	pname	price	category
1	Intel PIII-700	300.00	hardware
2	MS Office Pro	500.00	software
3	IBM DB2	5000.00	software
4	Thinkpad 600E	5000.00	hardware

Transaction Relation

- Relation schema:
  Transactions(
  tid: integer,
  tdate: date,
  cid: integer,
  pid: integer)
- Relation instance:

tid	tdate	cid	pid
1	1/1/2000	1	1
1	1/1/2000	1	2
2	1/1/2000	1	4
3	2/1/2000	2	3
3	2/1/2000	2	4

The Object-Oriented Data Model

- Richer data model. Goal: Bridge impedance mismatch between programming languages and the database system.
- Example components of the data model: Relationships between objects directly as pointers.
- Result: Can store abstract data types directly in the DBMS
  - Pictures
  - Geographic coordinates
  - Movies
  - CAD objects

Object-Oriented DBMS

- Advantages: Engineering applications (CAD and CAM and CASE computer aided software engineering), multimedia applications.
- Disadvantages:
  - Technology not as mature as relational DMBS
  - Not suitable for decision support, weak security
  - Vendors are much smaller companies and their financial stability is questionable.

Object-Oriented DBMS (Contd.)

Vendors:

- Gemstone (www.gemstone.com)
- Objectivity (www.objy.com)
- ObjectStore (www.objectstore.net)
- POET (www.poet.com)
- Versant (www.versant.com, merged with POET)

Organizations:

- OMG: Object Management Group (www.omg.org)

Object-Relational DBMS

- Mixture between the object-oriented and the object-relational data model
  - Combines ease of querying with ability to store abstract data types
  - Conceptually, the relational model, but every field
- All major relational vendors are currently extending their relational DBMS to the object-relational model

Query Languages

We need a high-level language to describe and manipulate the data

Requirements:

- Precise semantics
- Easy integration into applications written in C++/Java/Visual Basic/etc.
- Easy to learn
- DBMS needs to be able to efficiently evaluate queries written in the language

Relational Query Languages

- The relational model supports simple, powerful querying of data.
  - Precise semantics for relational queries
  - Efficient execution of queries by the DBMS
  - Independent of physical storage

SQL: Structured Query Language

- Developed by IBM (System R) in the 1970s
- ANSI standard since 1986:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision, current standard)
  - SQL-99 (major extensions)
- More about SQL in the next lecture

Example Query

- Example Schema:
  Customers(
  cid: integer,
  name: string,
  byear: integer,
  state: string)

cid	name	byear	state
1	Jones	1960	NY
2	Smith	1974	CA
3	Smith	1950	NY

- Query:

```
SELECT
    Customers.cid,
    Customers.name,
    Customers.byear,
    Customers.state
FROM Customers
WHERE Customers.cid = 3
```

cid	name	byear	state
3	Smith	1950	NY

Example Query

```
SELECT
  Customers.cid,
  Customers.name,
  Customers.byear,
  Customers.state
FROM Customers
WHERE
  Customers.cid = 1
```

cid	name	byear	state
1	Jones	1960	NY
2	Smith	1974	CA
3	Smith	1950	NY

cid	name	byear	state
1	Jones	1960	NY

Why Store Data in a DBMS?

- Benefits
  - Transactions (concurrent data access, recovery from system crashes)
  - High-level abstractions for data access, manipulation, and administration
  - Data integrity and security
  - Performance and scalability

Integrity Constraints

- Integrity Constraints (ICs): Condition that must be true for any instance of the database.
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
  - A legal instance of a relation is one that satisfies all specified ICs.
  - DBMS should only allow legal instances.
- Example: Domain constraints.

Primary Key Constraints

- A set of fields is a superkey for a relation if no two distinct tuples can have same values in all key fields.
- A set of fields is a key if the set is a superkey, and none of its subsets is a superkey.
- Example:
  - {cid, name} is a superkey for Customers
  - {cid} is a key for Customers
- Where do primary key constraints come from?

Primary Key Constraints (Contd.)

- Can there be more than one key for a relation?
- What is the maximum number of superkeys for a relation with k fields?

Where do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.
- We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about all possible instances!
  - From example, we know state cannot be a key, but the assertion that cid is a key is given to us.
- Key and foreign key ICs are very common; a DBMS supports more general ICs.

Security

- Secrecy: Users should not be able to see things they are not supposed to.
  - E.g., A student can't see other students' grades.
- Integrity: Users should not be able to modify things they are not supposed to.
  - E.g., Only instructors can assign grades.
- Availability: Users should be able to see and modify things they are allowed to.

Why Store Data in a DBMS?

- Benefits
  - Transactions (concurrent data access, recovery from system crashes)
  - High-level abstractions for data access, manipulation, and administration
  - Data integrity and security
  - Performance and scalability

DBMS and Performance

- Efficient implementation of all database operations
- Indexes: Auxiliary structures that allow fast access to the portion of data that a query is about
- Smart buffer management
- Query optimization: Finds the best way to execute a query
- Automatic high-performance concurrent query execution, query parallelization

Summary Of DBMS Benefits

- Transactions
  - ACID properties, concurrency control, recovery
- High-level abstractions for data access
  - Data models
- Data integrity and security
  - Key constraints, foreign key constraints, access control
- Performance and scalability
  - Parallel DBMS, distributed DBMS, performance tuning

Technologies at the Three Tiers

- Internet concepts
  - URIs
  - The HTTP Protocol
- The presentation layer
  - HTML, HTML Forms
  - JavaScript
  - Style Sheets
  - Cookies

Uniform Resource Identifiers

- Uniform naming schema to identify *resources* on the Internet
- A resource can be anything:
  - Index.html
  - mysong.mp3
  - picture.jpg
- Example URIs:
  http://www.cs.wisc.edu/~dbbook/index.html
  mailto:webmaster@bookstore.com

Structure of URIs

http://www.cs.wisc.edu/~dbbook/index.html

- URI has three parts:
  - Naming schema (http)
  - Name of the host computer (www.cs.wisc.edu)
  - Name of the resource (~dbbook/index.html)
- URLs are a subset of URIs

HTTP Overview

- HTTP: HyperText Transfer Protocol
- Developed by Tim Berners Lee, 1990
- Client/Server Architecture:
  - Client requests a document
    - Example clients: IE, Netscape, etc.
  - Server returns the document
    - Example servers: Apache, IIS

Watch HTTP

- Telnet:
  - telnet www.yahoo.com 80
  - GET /
- See your requests:
  - http://www.schroepl.net/cgi-bin/http_trace.pl

Example HTTP Session

- Client sends request → Server sends response
- Client requests the following URL: http://hypothetical.ora.com:80/
- Anatomy of the Request:
  - http:// HyperText Transfer Protocol; other options: ftp, mailto.
  - hypothetical.ora.com: host name
  - :80: Port Number. 80 is reserved for HTTP. Ports can range from: 1-65,535
  - / Root document

The Client Request

- Actual Browser Request:

```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap,
  image/ jpeg, image/pjpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible;
  MSIE 5.01; Windows NT)
Host: hypothetical.ora.com
Connection: Keep-Alive
```

Anatomy of the Client Request

- GET / HTTP/1.1
  - Requests the root / document.
  - Specifies HTTP version 1.1.
  - HTTP Versions: 1.0 and 1.1 (more on this later…)
- Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
  - Indicates what type of media the browser will accept.
- Accept-Language: en-us
  - Browser's preferred language
- Accept-Encoding: gzip, deflate
  - Accepts compressed data (speeds download times.)

Anatomy of the Client Request

- User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT)
  - Indicates the browser type.
- Host: hypothetical.ora.com
  - Required for HTTP 1.1
  - Optional for HTTP 1.0
  - A Server may host multiple hostnames. Hence, the browser indicates the host name here.
- Connection: Keep-Alive
  - Enables "persistent connections". Faster performance (more later…)

Server Response

```
HTTP/1.1 200 OK
Date: Mon, 24 Sept 2001 20:54:26 GMT
Server: Apache/1.3.6 (Unix)
Last-Modified: Mon, 24 Sept 2001 14:06:11 GMT
Content-length: 327
Connection: close
Content-type: text/html
<title>Sample Homepage</title>
<img src="/images/oreilly_mast.gif">
<h1>Welcome</h2>Hi there, this is a simple web page.
  Granted, it may not be as elegant as some other
  web pages you've seen on the net, but there are
  some common qualities...
```

Anatomy of Server Response

- HTTP/1.1 200 OK
  - Server Status Code
  - Code 200: Document was found
  - We will examine other status codes shortly.
- Date: Mon, 24 Sept 2001 20:54:26 GMT
  - Date on the server.
  - GMT (Greenwich Mean Time)
- Last-Modified: Mon, 24 Sept 2001 14:06:11 GMT
  - Indicates the time when the document was last modified.
  - Very useful for browser caching.
  - If a browser already has the page in its cache, it may not need to request the whole document again (more later…)

Anatomy of Server Response

- Content-length: 327
  - Number of bytes in the document response.
- Connection: close
  - Indicates that the server will close the connection.
  - If the client wants to send another request, it will need to open another connection to the server.
- Content-type: text/html
  - Indicates the MIME Type of the return document.
  - Multi-Purpose Internet Mail Extensions
  - Enables web servers to return binary or text files.
  - Other MIME Categories:
    - audio, video, images, xml

Anatomy of Server Response

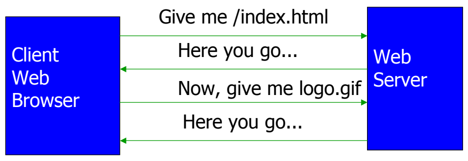
The actual HTML document:

```
<HTML>
<HEAD>
    <TITLE>Cornell University - CS/CIS 330</TITLE>
    <meta http-equiv="Content-Type" content="text/html;
    charset=utf-8" />
    <meta NAME="Author" CONTENT="Scott Selikoff">
    <meta NAME="robots" content="index,follow">
</HEAD>

<NOFRAMES>
<h1 align=center>
```

Getting Objects

- Once a browser receives an HTML page, it makes separate connections to retrieve different objects within the page.



HTTP 1.0 v. 1.1

- HTTP 1.0:
  - For each request, you must open a new connection with the server.
- HTTP 1.1
  - For each request, the default action is to maintain an open connection with the server.
  - Faster, Persistent Connections
  - Supported by most browsers and servers.

Example: HTTP 1.0 v. 1.1

- HTTP 1.0: Get HTML Page plus Images
  - Open Connection: GET /index.html
  - Open Connection: GET /logo.gif
  - Open Connection: GET /button.gif
- HTTP 1.1: Get HTML Page plus Images
  - Open Persistent Connection: GET /index.html
  - GET /logo.gif
  - GET /button.gif

Client Requests

- Every client request includes three parts:
  - Method: Used to indicate type of request, HTTP Version and name of requested document.
  - Header Information: Used to specify browser version, language, etc.
  - Entity Body: Used to specify form data for POST requests.

Client Methods

- GET and POST: We will see them later when we discuss HTML forms.
- HEAD:
  - Similar to GET, except that the method requests only the header information.
  - Server will return date-modified, but will not return the data portion of the requested document.
  - Useful for browser caching.
  - For example:
    - If browser contains a cached version of a page, it issues a head request.
    - If document has not been modified recently, use cached version.

Server Responses

- Every server response includes three parts:
  - Response line: HTTP version number, three digit status code, and status message.
  - Header: Information about the server
  - Entity Body: The actual data.

Server Status Codes

100-199	Informational
200-299	Client Request Successful
300-399	Client Request Redirected
400-499	Client Request Incomplete
500-599	Server Errors

Some Important Status Codes

- 200: OK
  - Request was successful.
- 301: Moved Permanently
  - Server redirects client to a new URL.
- 404 Not Found
  - Document does not exist
- 500 Internal Server Error
  - Error within the Web Server

HTTP Is Stateless

- What does this mean:
  - No "sessions"
  - Every message is completely self-contained
  - No previous interaction is "remembered" by the protocol
  - Tradeoff between ease of implementation and ease of application development: Other functionality has to be built on top
- Implications for applications:
  - Any state information (shopping carts, user login-information) need to be encoded in every HTTP request and response!
  - Popular methods on how to maintain state:
    - Cookies
    - Dynamically generate unique URL's at the server level