CS 194: Distributed Systems
Final Review

Scott Shenker and Ion Stoica
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720-1776

During this Class…

- Learn the basics of distributed systems
- Two parts:
- First part: traditional distributed system theory
  - Algorithms and protocols to implement basic services
- Second part: examples of distributed systems
  - Illustrate the context in which classic algorithms/protocols are used
  - Discuss the non-traditional distributed systems

Traditional Distributed Systems Theory

- Deal with implementing services required to build such systems
  - Usually strong assumptions and semantics

Clock Synchronization

- Real clocks:
  - Cristian's algorithm (UTC time-server based)
  - Berkeley algorithm (no UTC signal, but master)
- Logical clocks: capture the causality between events in a distributed system
  - E.g., Lamport timestamps

Elections

- Need to select a special node, that all other nodes agree on
- Assume all nodes have unique ID
- Example methods for picking node with highest ID
  - Bully algorithm
  - Gossip method

Exclusion

- Ensuring that a critical resource is accessed by no more than one process at the same time
- Methods:
  - Centralized coordinator: ask, get permission, release
  - Distributed coordinator: treat all nodes as coordinator
    - If two nodes are competing, timestamps resolve conflict
  - Interlocking permission sets: Every node I asks permission from set P[I], where P[I] and P[J] always have nonempty intersections

Concurrency Control

- Want to allow several transactions to be in progress
- But the result must be the same as some sequential order of transactions
- Use locking policies:
  - Grab and hold
  - Grab and unlock when not needed
  - Lock when first needed, unlock when done
  - Two-phase locking
- Which policies can have deadlock?

7

Agreement

- How do two or more processes reach agreement?
- **Two-Army Problem**
  - Assumptions
    - Processes are correct
    - Adversary can intercept messages
  - No solution
- **Byzantine agreement**
  - Assumptions:
    - Processes subject to arbitrary failures
    - Messages delivery is correct, and bounded
  - Solution: In a system with m faulty processes agreement can be achieved only if there are $2m+1$ functioning correctly

8

Distributed Commit

- **Goal**: Either **all** members of a group decide to perform an operation, or **none** of them perform the operation
- Assumptions:
  - Crash failures that can be recovered
  - Communication failures detectable by timeouts
- Solution: two phase commit (2PC)
- Notes:
  - Commit requires a set of processes to agree…
  - …similar to the two-army problem…
  - … but solvable because simpler because stronger assumptions

9

Group Communication

- **Reliable multicast**: all non-faulty processes which do not join/leave during communication receive the message
  - Example: SRM
- **Atomic multicast**: all messages are delivered in the same order to all processes
  - Birman et al. algorithm

Multicast	Basic Message Ordering	Total-ordered Delivery?
Reliable multicast	None	No
FIFO multicast	FIFO-ordered delivery	No
Causal multicast	Causal-ordered delivery	No
Atomic multicast	None	Yes
FIFO atomic multicast	FIFO-ordered delivery	Yes
Causal atomic multicast	Causal-ordered delivery	Yes

10

Data Replication and Consistency

- Scalability requires replicated data
- Application correctness requires some form of consistency
  - Here we focus on individual operations, not transactions
- Consistency models:
  - Strict consistency (in your dreams…)
  - Linearizable (in your proofs….)
  - Sequential consistency: same order of operations
  - Causal consistency: all causal operations ordered
  - FIFO consistency: operations within process ordered
- Mechanisms
  - Local cache replicas: pull, push, lease (produce sequential consistency)
  - Replicated-write protocols: quorum techniques

11

Security (Requirements)

- **Authentication**: ensures that sender and receiver are who they are claiming to be
- **Data integrity**: ensure that data is not changed from source to destination
- **Confidentiality**: ensures that data is red only by authorized users
- **Non-repudiation**: ensures that the sender has strong evidence that the receiver has received the message, and the receiver has strong evidence of the sender identity (not discussed in this class)

12

Security (Solutions)

- Security foundation: cryptographic algorithms
  - Secret key cryptography, Data Encryption Standard (DES)
  - Public key cryptography, RSA algorithm
  - Message digest, MD5
- Confidentiality → data encryption
- Integrity → digital signature
- Authentication
  - Shared secrete key based authentication
  - Key Distribution Center (KDC) based authentication
    - E.g., Needham-Schroeder Protocol
  - Public key cryptography authentication
- Key management → Public Key Infrastructure (PKI), Kerberos

13

Final Exam Information

- Date, time: May 19, 12:30-3:30pm
- Final will include midterm material (1/3), but emphasize on second part (2/3)
  - This lecture reviews mainly the second part material
  - See midterm review for first part material!
- We'll post a practice exam by the end of this week
- Closed books; 8,5"x11" crib sheet (both sides)
- No calculators, PDAs, cell phones with cameras, etc
- Please use PENCIL and ERASER

14

Outline

- Distributed File Systems
- Distributed Object-based Systems
- Coordination Systems
- Web

15

Distributed File Systems

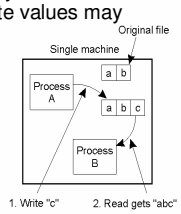
- Provide a client **transparent** access to files stored at a **remote** server
- Why would you want to store files remotely?
  - Sharing files
  - Reliability
  - Manageability

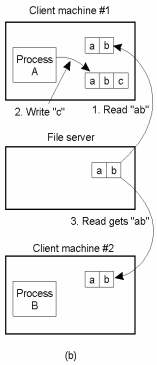
16

Semantics of File Sharing

a) On a single processor, when a *read* follows a *write*, the value returned by the *read* is the value just written

b) In a distributed system with caching, obsolete values may be returned





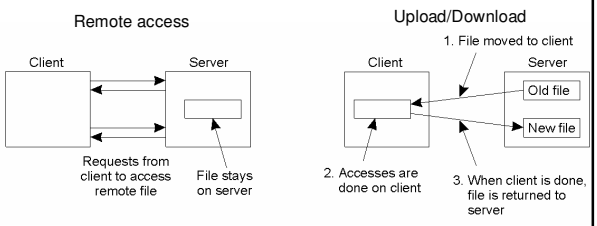
Semantics of File Sharing

Method	Comment
UNIX semantics	Every operation on a file is instantly visible to all processes
Session semantics	No changes are visible to other processes until the file is closed
Immutable files	No updates are possible; simplifies sharing and replication
Transaction	All changes occur atomically

18

Access Model

- Two access models :
  - Remote access
  - Upload/Download



19

Stateful vs. Stateless

- Stateless model: each call contains complete information to execute operation
- Stateful model: server maintain context (info) shared by consecutive operations
- Discussion: compare stateless and stateful design

20

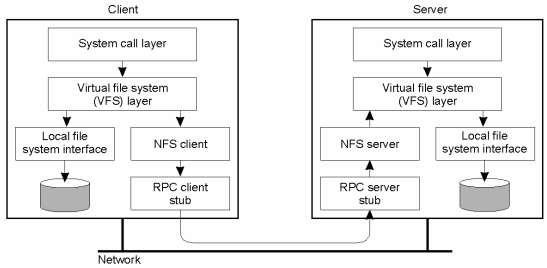
Network File System (NSF)

- A specification for a distributed file system (by Sun, 1984)
  - Implemented on various OS's
  - De facto standard in the UNIX community
  - Latest version is 4 (2000)
- NSF v1-v3
  - File sharing semantics: UNIX (if no caching)
  - Access model: remote access
  - Stateless design
- NSF v4:
  - File sharing semantics: session
  - Access model: upload/download
  - Stateful design

21

NFS Architecture

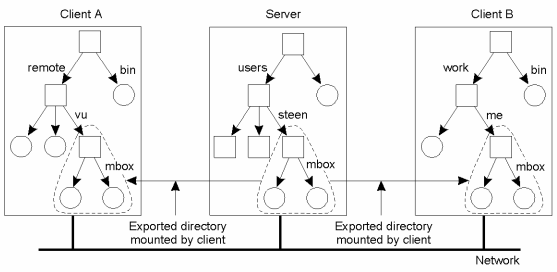
- Virtual File System (VFS) provide a uniform access to local and remote files



22

Naming

- Allow a client to mount a remote file system into its own local file system
- Pathnames are **not** globally unique; what's the implication?



CODA

- Developed at CMU
  - Based on Andrew File System (AFS), another distributed system developed at CMU
- Goals
  - **Scalability:** system should grow without major problems
  - **Fault-Tolerance:** system should remain usable in the presence of server failures, communication failures and voluntary disconnections
  - **Disconnected mode** for **portable computers**
- **Design philosophy: Scalability and Accessibility more important than consistency**

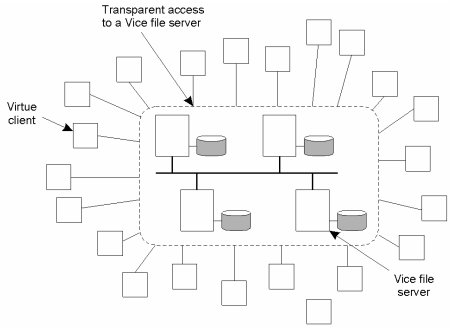
24

CODA

- Sharing model: transaction
  - Session viewed as a transaction
- Access model: upload/download

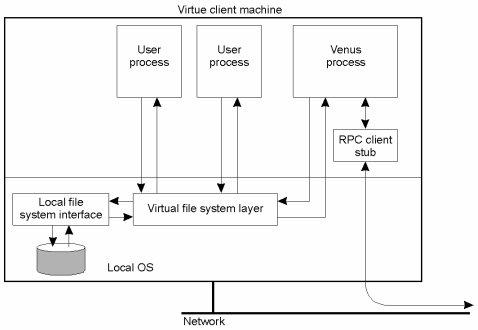
25

Overall Organization of AFS & Coda



26

Internal Organization of Virtue



27

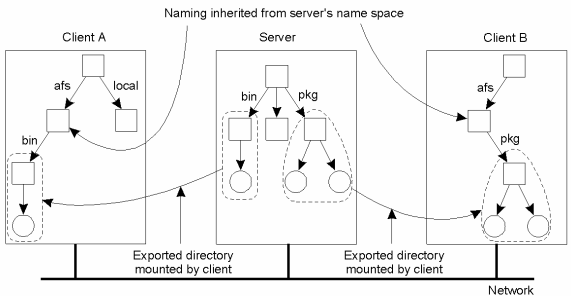
Communication

- Based on RPC2: provides reliable transmission on top of UDP
- RPC2 supports side-effects, i.e., user defined protocols
- RPC2 provides support for multicast
  - Transparent for the client

28

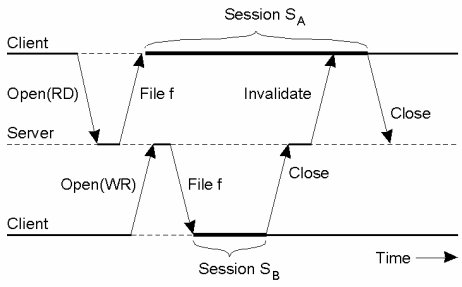
Naming

- Single shared naming space (vs. client-based in NFS)



Sharing Files in Coda

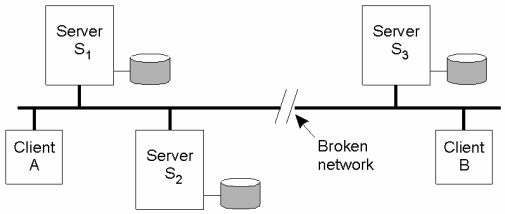
- Transaction semantics: session is treated like a transaction



30

Handling Network Partition

- Versioning vector when partition happens: [1,1,1]
- Client A updates file → versioning vector in its partition: [2,2,1]
- Client B updates file → versioning vector in its partition: [1,1,2]
- Partition repaired → compare versioning vectors: **conflict**!



31

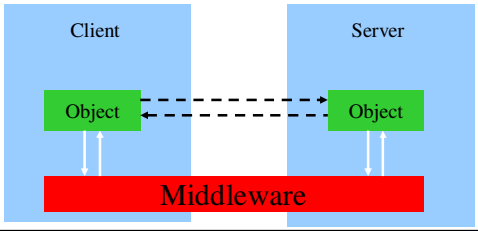
Outline

- Distributed File Systems
- Distributed Object-based Systems
- Coordination Systems
- Web

32

Distributed Object-based Systems

- Goal: **transparently** access remote objects in a distributed system
- Challenges:
  - Ensure semantics of invoking a **local** object
  - Accommodate heterogeneity, e.g., multiple languages, OSes, …



33

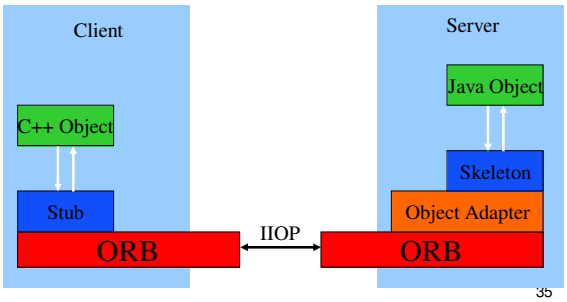
Two Examples

- Common Object Request Broker Architecture (CORBA)
  - International standard: multiple languages, OSes, vendors
- Distributed Common Object Model (DCOM)
  - MS standard: only MS OSes

34

CORBA Architecture

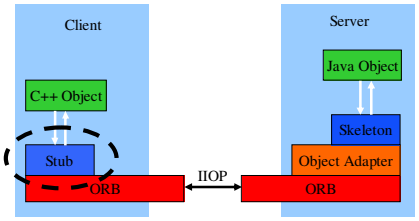
- Remote-object: object implementation resides in server's address space



35

Stub

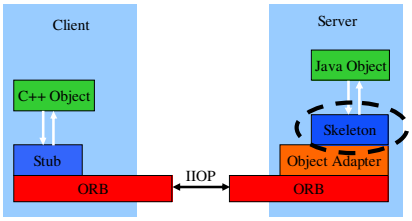
- Provides interface between client object and ORB
- Marshalling: client invocation
- Unmarshalling: server response



36

Skeleton

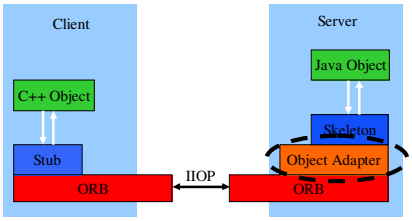
- Provides iterface between server object and ORB
- Unmarshaling: client invocation
- Marshaling: server response



37

(Portable) Object Adapter (POA)

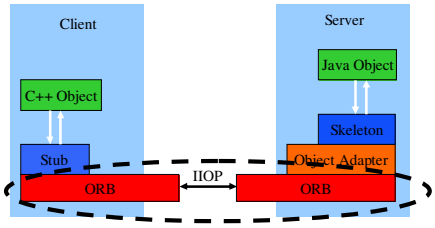
- Register class implementations
- Creates and destroys objects
- Handles method invokation
- Handles client authentication and access control



38

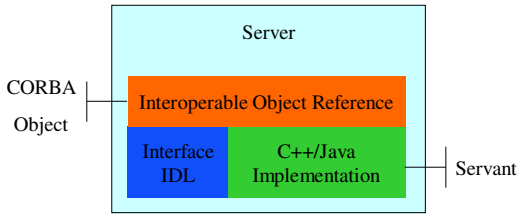
Object Request Broker (ORB)

- Communication infrastructure sending messages between objects
- Communication type:
  - GIOP (General Inter-ORB Protocol)
  - IIOP (Internet Inter-ORB Protocol) (GIOP on TCP/IP)



39

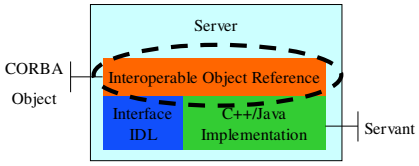
CORBA Object



40

Interoperable Object Reference (IOR)

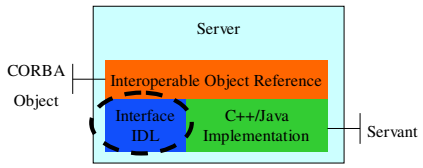
- Uniquely identifies an object (see object references)



41

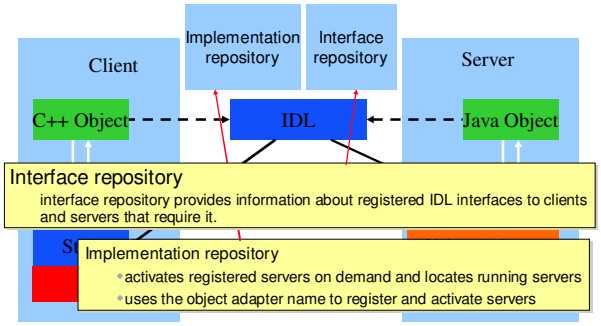
Interface Definition Language (IDL)

- Describes interface
- Language independent
- Client and server platform independent



42

Overall CORBA Architecture



Example of CORBA Services

- Naming: Keeps track of association between object names and their reference. Allows ORB to locate referenced objects
- Life Cycle: Handles the creation, copying, moving, and deletion objects
- Trader: A "yellow pages" for objects. Lets you find them by the services they provide
- Event: Facilitates asynchronous communications through events
- Concurrency: Manages locks so objects can share resources
- Query: Locates objects by specified search criteria
- …

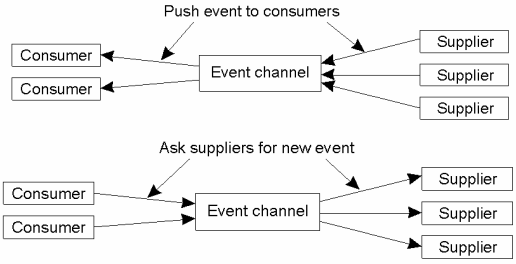
Object Invocation Models

- Invocation models supported in CORBA

Request type	Failure semantics	Description
Synchronous	At-most-once	Caller blocks until a response is returned or an exception is raised
One-way	Best effort delivery	Caller continues immediately without waiting for any response from the server
Deferred synchronous	At-most-once	Caller continues immediately and can later block until response is delivered

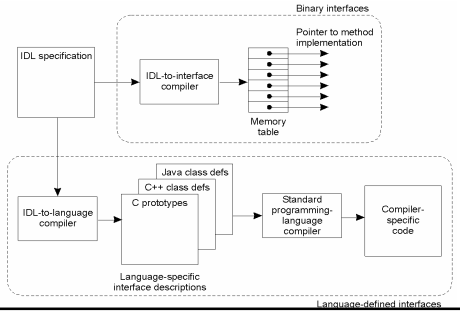
Event and Notification Services

- Models:
  - Push: consumers need to register to a channel
  - Pull: consumers explicitly ask for events

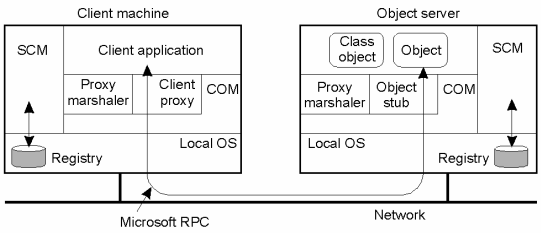


CORBA vs DCOM

- CORBA: language-defined interface
- DCOM: binary interface



DCOM Architecture



- SCM: Service Control Manager

Creating objects

- Classes of objects have globally unique identifiers (GUIDs)
  - 128 bit numbers
  - Also called class ids (CLSID)
- DCOM provides functions to create objects given a server name and a class id
  - The SCM on the client connects to the SCM of the server and requests creation of the object

49

Outline

- Distributed File Systems
- Distributed Object-based Systems
- Coordination Systems
- Web

50

Coordination Systems

- Handle all communication and cooperation between processes/objects in a distributed system
  - Emphasize **not** on transparency
  - Object distribution is **explicit**
- Can be classified along two dimensions:
  - **Temporal**: do sender and receiver need to be active simultaneously?
  - **Referential**: do sender need to know the identifier of the receiver?

51

Taxonomy of Coordination Models

Referential \ Temporal	Coupled	Uncoupled
Coupled	Direct	Mailbox
Uncoupled	Meeting oriented	Generative communication

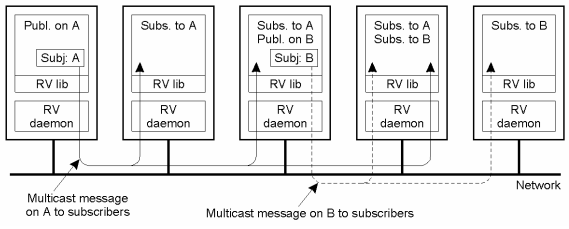
52

TIB/Rendezvous System

- Meeting oriented model (a.k.a. publish/subscriber)
- Build around concept of **information bus**
- Messages are **subject-based** addressed
  - A message doesn't specify destination, but a **subject name**
- A message is delivered to all objects interested in message's subject

53

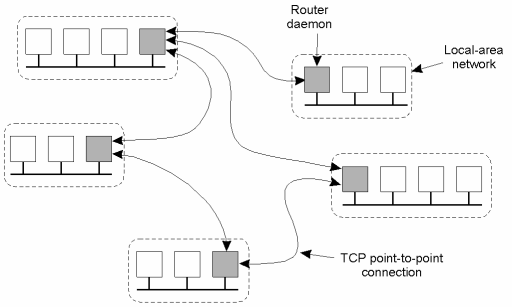
TIB/Rendezvous Architecture



54

Wide-area Architecture

- Use IP multicast on LANs
- Overlay multicast in wide-area



55

Communication Primitives

- **send()**: send message; non-blocking operation
- **sendreply()**: send a reply upon receiving a message; non-blocking operation
- **sendrequest()**: send message; blocks until a reply is received
- No receive operation; received messages are handled via events

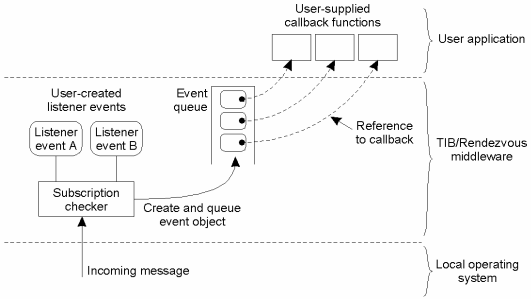
56

Events

- To subscribe to a subject, create a **listener event** object
- **Listener event** contains reference to a callback function
- When a message arrives, create an **event object** and enque it in an **event queue**
- Each **event queue** is associated a **dispatcher thread**
- **Dispatcher thread** removes object at the head of the queue and invokes callback function

57

Handling Events



58

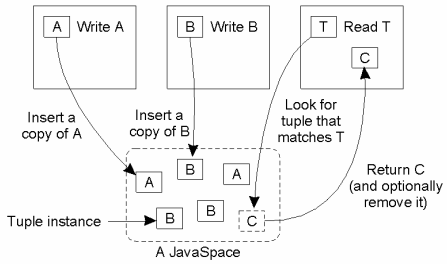
Jini

- Generative communication model
- Built around the concept of tuple space
  - First proposed by Linda
- Tuple space
  - Distributed associative memory
  - Instantiated as a JavaSpace in Jini
- In addition, Jini
  - Provide distributed event and notification system
  - Allow clients discover services when become available

59

JavaSpace

- **write()**: create an object copy and store it in JavaSpace
- **read()**: return tuples from JavaSpace that match a template
- **take()**: like read, but removes tuple from JavaSpace



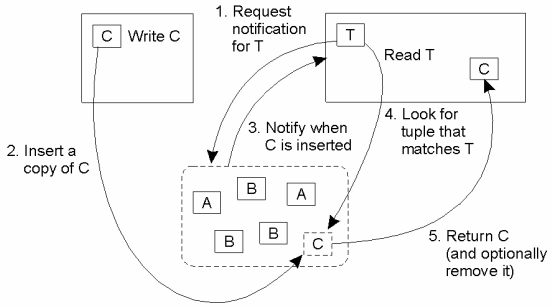
60

Events

- A client can register with an object that has events of interest
- A client can tell object to pass event to another process
- Notification implemented by remote call

61

Using Events with JavaSpaces



62

Outline

- Distributed File Systems
- Distributed Object-based Systems
- Coordination Systems
- Web

63

The Web

Tim Berners-Lee

- World Wide Web (WWW): a distributed database of "pages" linked through Hypertext Transport Protocol (HTTP)
  - First HTTP implementation - 1990
    - Tim Berners-Lee at CERN
  - HTTP/0.9 – 1991
    - Simple GET command for the Web
  - HTTP/1.0 –1992
    - Client/Server information, simple caching
  - HTTP/1.1 - 1996

64

Web Architecture

- Core components:
  - Servers: store files and execute remote commands
  - Browsers: retrieve and display "pages"
  - Uniform Resource Locators (URLs): way to refer to pages
- A protocol to transfer information between clients and servers
  - HTTP

65

Uniform Record Locator (URL)

```
protocol://host-name:port/directory-path/resource
```

- Extend the idea of hierarchical namespaces to include anything in a file system
  - ftp://www.cs.berkeley.edu/~istoica/cs194/05/lecture.ppt
- Extend to program executions as well…
  - http://us.f413.mail.yahoo.com/ym/ShowLetter?box=%40B%40Bulk&MsgId=2604_1744106_29699_1123_1261_0_28917_3552_1289957100&Search=&Nhead=f&YY=31454&order=down&sort=date&pos=0&view=a&head=b
  - Server side processing can be incorporated in the name

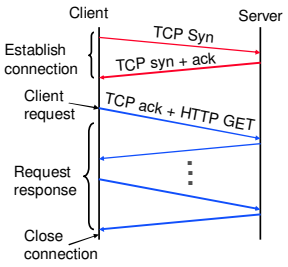
66

Hyper Text Transfer Protocol (HTTP)

- Client-server architecture
- Synchronous request/reply protocol
  - Runs over TCP, Port 80
- Stateless

67

Big Picture



68

Hyper Text Transfer Protocol Commands

- GET – transfer resource from given URL
- HEAD – GET resource metadata (headers) only
- PUT – store/modify resource under given URL
- DELETE – remove resource
- POST – provide input for a process identified by the given URL (usually used to post CGI parameters)

69

Client Request

- Steps to get the resource:

  http://www.eecs.berkeley.edu/index.html

  1. Use DNS to obtain the IP address of www.eecs.berkeley.edu

  2. Send to an HTTP request:

     ```
     GET /index.html HTTP/1.0
     ```

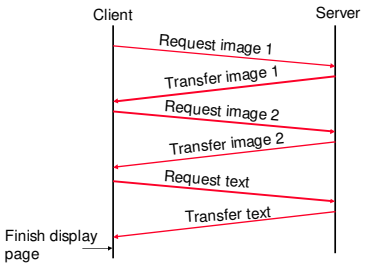
70

Server Response

```
HTTP/1.0 200 OK
Content-Type: text/html
Content-Length: 1234
Last-Modified: Mon, 19 Nov 2001 15:31:20 GMT
<HTML>
<HEAD>
<TITLE>EECS Home Page</TITLE>
</HEAD>
…
</BODY>
</HTML>
```

71

HTTP/1.0 Example



72

HHTP/1.0 Performance

- Create a new TCP connection for each resource
  - Large number of embedded objects in a web page
  - Many short lived connections
- TCP transfer
  - Too slow for small object
  - May never exit slow-start phase
- Connections may be set up in parallel (5 is default in most browsers)

73

HTTP/1.1 (1996)

- Performance:
  - Persistent connections
  - Pipelined requests/responses
  - …
- Efficient caching support
  - Network Cache assumed more explicitly in the design
  - Gives more control to the server on how it wants data cached
- Support for virtual hosting
  - Allows to run multiple web servers on the same machine

74

Final Exam Information

- Date, time: May 19, 12:30-3:30pm
- Final will include midterm material (1/3), but emphasize on second part (2/3)
  - This lecture reviews mainly the second part material
  - See midterm review for first part material!
- We'll post a practice exam by the end of this week
- Closed books; 8,5"x11" crib sheet (both sides)
- No calculators, PDAs, cell phones with cameras, etc
- Please use PENCIL and ERASER

75