What is "Cloud Computing"?
A programming paradigm for distributed applications
A business paradigm for reassigning business risk
An infrastructure paradigm for greening the data center
Clouds as a programming paradigm

No useful concept of a file or a database.
Throw data instances into (or perhaps "at") the cloud.
Query the cloud whenever you need something back.
Cloud is an **opaque data container** that does all of the work of persistence and integrity for you.
Clouds provide a way to reassign (or outsource) business risks.

Running an Information Technology infrastructure is very expensive in human labor.

Main expenses: assuring availability and data integrity.

One can "pay someone else" to insure these.
Fact: power needs for computing are growing without bound.

One way to reduce power consumption is to **exploit economy of scale**.

Running one large datacenter and sharing resources takes much less power than several small datacenters for specific purposes.
Kinds of clouds:

**Software-as-a-service (SaaS):** a high-availability application is provided for ubiquitous access by the consumer. (gmail)

**Platform-as-a-service (PaaS):** cloud provider makes a programming platform available to a client, for running the client's programs in serving the client's customers. (Google Apps)

**Infrastructure-as-a-service (IaaS):** bare-metal or virtual machines are provided to a company for their own use, in a remote datacenter. (Amazon)
Example of SaaS: Google Apps
  Calendar, mail, tasks.
  Google writes the apps.

Simpler example: "Remember the milk"
  Customer pays small access fee
  Ubiquitous access to task lists from all kinds of devices.
Example of PaaS: Google AppEngine
   Customer provides a "cloud-aware application".
   Program runs on Google's servers.

Other examples:
   Microsoft Azure
   Yahoo Hadoop
Example of IaaS: Amazon Web Services
  Customer pays Amazon to host an application.
  Customer provides images of linux servers.
  Amazon charges customer only when servers are
  needed and running.

Another example: eucalyptus
  "Build your own" infrastructure management.
  You're responsible for "maintaining the cloud".
Cloud computing is about serial processing
   Not a parallel computing paradigm.
   Instead, about enabling reliable serial processes.
   Reliability is attained through scale and parallelism!
Public versus private clouds

In a public cloud, you potentially share resources with competitors. (Amazon EC2)

In a private cloud, your machines and infrastructure are dedicated to your use. (Eucalyptus)
Cloud challenges

Writing cloud-enabled applications
Cloudsourcing: adapting non-cloud and legacy applications to a cloud.
Security and privacy: understanding risks and guarantees in public and private settings.
Availability and elasticity: making a cloud respond to changes in demand.
Power awareness: making the cloud "green".
Rough sequence of the course

Monday, January 25, 2010
12:41 PM

Rough sequence of this course
  PaaS
  IaaS
  Power awareness
PaaS has rapidly evolved through several distinct steps:

First stage: **web applications**.
Second stage: **applet/servlet architecture**.
Third stage: **service-oriented architectures (SOAs)**.
Fourth stage: **cloud applications** -- what we will study.

Good news: no SOAP or REST
Bad news: other difficulties arise instead!
Web applications

Monday, January 25, 2010
12:26 PM

First stage: web applications

Web "pages" communicate with "web servers" via the
Common Gateway Interface (CGI)

   CGI request: what is wanted.
   CGI response: response to the request.

Requests are generated by user, responses are
returned directly to the user.
Second stage: Applet/Servlet architecture

Pages contain "Applets" that are little Java programs that implement, e.g., innovative interfaces. Applets communicate with servlets via some protocol. "Servlets" are little Java Programs that answer one kind of request. Applets display results of servlet calls.
Third stage: service-oriented architecture (SOA)

Outsource certain functions to third parties.
Example: let google Maps draw a map.
Web page uses JavaScript to invoke services, and communicates with services via

SOAP: Simple Object Access Protocol
REST: Representative State Transfer protocols.

This is the basic pattern of Web 2.0.
Fourth stage: cloud applications

No more browser or html or javascript involved! An application on the client host communicates "directly with the cloud" via some protocol.

Usually two parts to each application:

A client part that mediates between the user and a server part.

A server part that mediates between the client part and the cloud, via some request/response protocol.

A mysterious ritual:

You (as a programmer) write one thing.
It is translated (automagically) into two sides.
You don't need to understand SOAP.
You don't need to understand storage.
It just happens.
The ritual of cloud programming

You (as a programmer) write a serial program.
It is translated (automagically) into two sides.
   A client side.
   A cloud side.
Some boxes are closed.
   You don't need to understand SOAP.
   You don't need to understand storage.
   It just happens.
Example: android "applications"

Client: a program "on the phone" that displays data.
Server: a program "in the google cloud" that interacts with Google and makes data available.
Target cloud: google AppEngine.
Thinking in the cloud

One challenge: thinking appropriately for a cloud

Data storage occurs at a very high level of abstraction. Access data mostly via **object interfaces**. Must conceive of **data storage without structure**. Cloud handles all structural optimizations. A lot of programming the cloud lies in trusting the optimizations of the cloud, and not interfering with them!
Datastore service

Makes crucial data persistent.
Later, query storage for stored things.
Storage versus persistence

Usually, we might think of storing data in a file. In a cloud, we think instead of making an object persistent, which means that its value stays the same between program invocations. How persistence is accomplished doesn't matter; we can presume that it will be made persistent, and not worry about how.
The AppEngine Datastore

**Annotate** objects with **Java DataStore Object** annotations (JDO).

Make persistent object store with a **factory**.

Control persistence with **Java Persistence API** (JPA).

Source:

[http://code.google.com/appengine/docs/java/datastore/overview.html](http://code.google.com/appengine/docs/java/datastore/overview.html)
import com.google.appengine.api.datastore.Key;

import java.util.Date;
import javax.jdo.annotations.IdGeneratorStrategy;
import javax.jdo.annotations.IdentityType;
import javax.jdo.annotations.PersistenceCapable;
import javax.jdo.annotations.Persistent;
import javax.jdo.annotations.PrimaryKey;

@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class Employee {
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key key;

    @Persistent
    private String firstName;

    @Persistent
    private String lastName;

    @Persistent
    private Date hireDate;

    public Employee(String firstName, String lastName, Date hireDate) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.hireDate = hireDate;
    }

    // Accessors for the fields. JDO doesn't use these, but your application does.
    public Key getKey() {
        return key;
    }
}
public String getFirstName() {
    return firstName;
}

// ... other accessors...

Pasted from
<http://code.google.com/appengine/docs/java/datastore/overview.html>
Declares an object as possibly persistent. IdentityType.APPLICATION describes scope of persistence (until application is decommissioned, i.e., permanently!).

@PersistenceCapable
   
   Declares an object as possibly persistent. IdentityType.APPLICATION describes scope of persistence (until application is decommissioned, i.e., permanently!).

@PrimaryKey
   Declares an object as unique, indexed, and searchable.

@Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
   Declares an object as being stored between calls to the server.
   valueStrategy: how to determine the value of the thing. IdGeneratorStrategy.IDENTITY: automatically make the value unique.

@Persistent
   Just declares an object as being stored between calls to the server.
One makes persistent objects via a factory

```java
import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManagerFactory;

public final class PMF {
    private static final PersistenceManagerFactory pmfInstance = 
        JDOHelper.getPersistenceManagerFactory("transactions-optional");

    private PMF() {}

    public static PersistenceManagerFactory get() {
        return pmfInstance;
    }
}
```

Pasted from
<http://code.google.com/appengine/docs/java/datastore/overview.html>
Taking the example apart

private static final PersistenceManagerFactory pmfInstance
    = JDOHelper.getPersistenceManagerFactory(
        "transactions-optional"
    );

Generates a factory that deals with persistence for the Employee class. "transactions-optional" means that we don't "need" a transaction store but that it's ok to create one!

public static PersistenceManagerFactory get() {
    return pmfInstance;
}

Just an accessor for the factory, that makes sure that it is only created once!
Assuring persistence:

```java
import java.util.Date;
import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManager;
import javax.jdo.PersistenceManagerFactory;

import Employee; // your persistent class
import PMF;       // your persistence manager factory

// ...
Employee employee = new Employee("Alfred", "Smith", new Date());

PersistenceManager pm = PMF.get().getPersistenceManager();

try {
    pm.makePersistent(employee);
} finally {
    pm.close();
}
```

Pasted from
<http://code.google.com/appengine/docs/java/datastore/overview.html>
Taking the example apart

PersistenceManager pm = PMF.get().getPersistenceManager();

Just gets a reference to the (unique) instance created by the factory.

try {
    pm.makePersistent(employee);
} finally {
    pm.close();
}

Store one employee in the datastore, ignoring errors, and closes the persistence mechanism.
import java.util.List;
import Employee;

// ...
    String query = "select from " + Employee.class.getName()
    + " where lastName == 'Smith'";
    List<Employee> employees = (List<Employee>)
    pm.newQuery(query).execute();

Pasted from
<http://code.google.com/appengine/docs/java/datastore/overview.html>
Taking the query apart

"select from " + Employee.class.getName() + " where lastName == 'Smith'";

A query specification in **PDO query language (PDOQL). This is not Structured Query Language (SQL)!**

(List<Employee>) pm.newQuery(query).execute();

Get employee records matching the query. Note that the query is to the **persistence manager** (pm), **not the object**.