Announcements

- I will be out of town Monday and Tuesday to present at Multimedia Computing and Networking (MMCN '01)
- Prof. David Lowenthal would provide a guest lecture on Tuesday

- You are expected to review papers. Paper reviews are graded. You are supposed to send me your name, email address and research interests for Project Milestone 1.

- Remember, late <anything> is not accepted

Outline for today

- Oceanstore: An architecture for Global-Scale Persistent Storage – University of California, Berkeley. ASPLOS 2000

- Feasibility of a Serverless Distributed File System deployed on an Existing set of Desktop PCs – Microsoft research. ACM SIGMETRICS 2000

Persistent store

E.g. files (traditional operating systems), persistent objects (in a object based system)

- Applications operate on objects in persistent store
  - Powerpoint operates on a persistent .ppt file, mutating its contents
  - Palm calendar operates on my calendar which is replicated in myYahoo, Palm Desktop and the Pilot itself
- Storage is cheap but maintenance is not
  - ~ 4 $/GB

Global Persistent Store

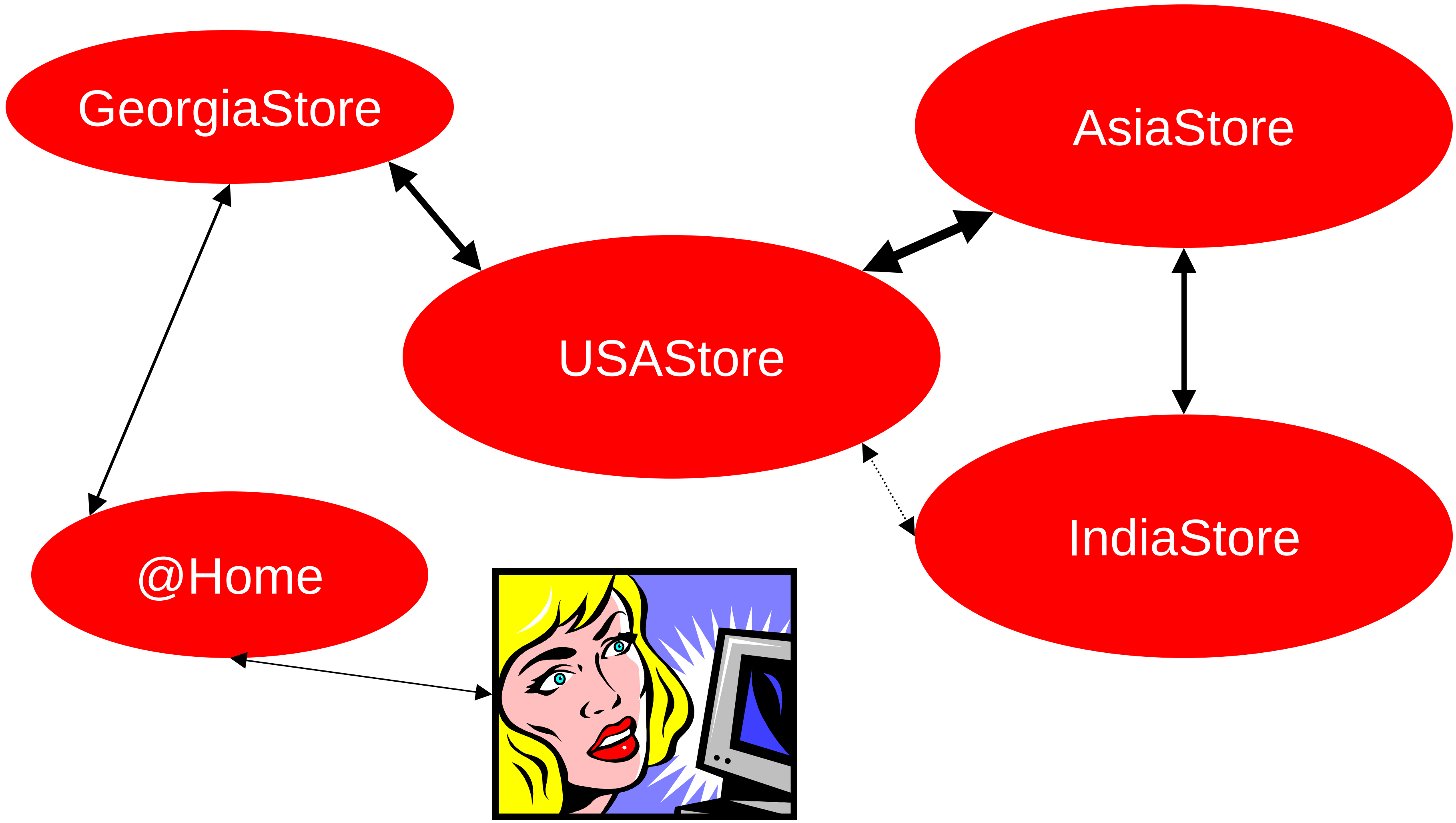
- Persistent store is fundamental for Ubiquitous Computing because it allows "devices" to operate transparently, consistently and reliably on data.

- Transparent: Permits behavior to be independent of the device themselves
- Consistently: Allows users to safely access the same information from many different devices simultaneously.
- Reliably: Devices can be rebooted or replaced without losing vital configuration information

Persistent store on a wide-scale

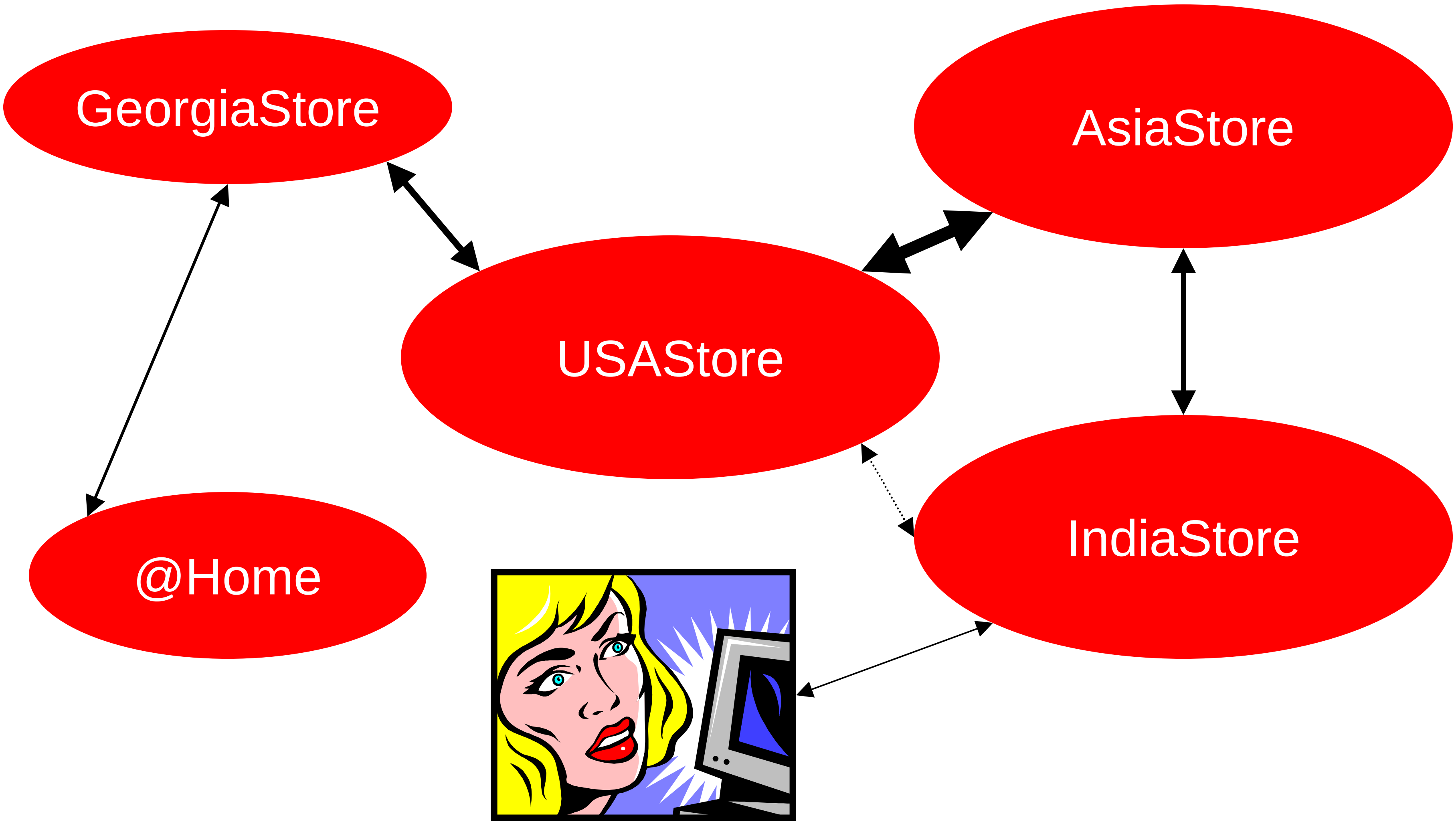
- 10 billion users, 10,000 files per user = 100 trillion files!!
- Information:
  - should be separated from location. To achieve uniform and highly-available access to information, servers must be geographically distributed, but exploit caching close to clients for performance
  - must be secure
  - must be durable
  - must be consistent

Oceanstore system model: Data Utility



End User with roaming access

Oceanstore system model: Data Utility



End User with roaming access

Oceanstore Goals

- Untrusted infrastructure (utility model – telephone)
  - Only clients can be trusted
  - Servers can crash, or leak information to third parties
  - Most of the servers are working correctly most of the time
  - Class of trusted servers that can carry out protocols on the clients behalf (financially liable for integrity of data)
- Nomadic Data Access
  - Data can be cached anywhere, anytime (promiscuous caching)
  - Continuous introspective monitoring to locate data close to the user

Oceanstore Persistent Object

- Named by a globally unique id (GUID)
- Such GUIDs are hard to use. If you are expecting 10 trillion files, your GUID will have to be a long (say 128 bit) ID rather than a simple name
  - passwd vs 12agfs237dfdfhj459uxzozfk459ldfnhgga
- self-certifying names
  1. secureHash(/id=surendar,ou=uga,key=<SecureKey>/etc/passwd) -> uniqueId
  2. Map uniqueId->GUID
  - Users would use symbolic links for easy usage
    - /etc/passwd -> uniqueId

SecureHash

- Pros:
  - The self-certifying name specifies my access rights
- Cons:
  - If I lose the key, the data is lost
    - Key management issues
      - Keys can be upgraded
      - Keys can be revoked
  - How do we share data?

Access Control

- All read-shared-users share an encryption key
  - Revocation:
    - Data should be deleted from all replicas
    - Data should be re-encrypted
    - New keys should be distributed
    - Clients can still access old data till it is deleted in all replicas
- All writes are signed
  - Validity checked by Access Control Lists (ACLs)
  - If A says trust B, B says trust C, C says trust D,

    what can you infer about A ? D

Oceanstore Persistent Object

- Objects are replicated on multiple servers. Replicated objects are not tied to particular servers i.e. floating replicas
- Replicas located by a probabilistic algorithm first before using a deterministic algorithm
- Data can be active or archival.
  - Archival data is read-only and spread over multiple servers – deep archival storage

Updates

- Objects are modified through updates (data is never overwritten) i.e. versioning system
- Application level conflict resolution
- Updates consist of a predicate and value pair. If a predicate evaluates to true, the corresponding value is applied.
  1. <room 453 free?>, <reserve room>
  2. <room 527 free?>, <reserve room>
  3. <else> <go to Jittery Joes>
- This is similar to Bayou which we will explore later in the semester

Introspection

- Oceanstore uses introspection to monitor system behavior
- Use this information for cluster recognition
- Use this information for replica management

MSR Serverless Distributed File System

- They've actually implemented this system within Microsoft and hence have real results

- Assumption 1: <u>not-fully-trusted</u> environment
- Assumption 2: Disk space is not that free

- Each disk is partitioned into three areas:
  - Scratch area – for local computations
  - Global storage area
  - Local cache for global storage

Efficiency consideration

- Compress data in storage
- Coalesce distinct files that have identical contents
  - Probably an artifact of Windows environment that stores files in specific locations e.g. c:\windows\system\
- File are replicated
  - Machines that are topologically close
  - Machines that are lightly loaded
  - Non-cache reads and writes to prevent buffer cache pollution

Replica management

- Files in a directory are replicated together
- When new machines join, its data is replicated to other machines
- Replicas of other files are moved into the new machine
- When machine leaves, the data in that machine is replicated in other machines from other replicas

Security

- File updates are digitally signed
- File contents are encrypted before replication
- Convergent encryption to coalesce encrypted file
  - Encryption:
    1. Hash(file contents) -> uniqueHash
       1. Encrypt(unencrypterfile, uniqueHash)->encryptedfile
          1. User1: encrypt(UserKey1, uniqueHash) -> Key1
          2. User2: encrypt(UserKey2, uniqueHash) -> Key2
  - Decryption
  - User1: decrypt(UserKey1, Key1) -> uniqueHash
  - Decrypt(encryptedfile, uniqueHash) -> unencryptedfile

Application API

- Related read, write operations to objects form a session (defined by the application developer)
- Users specify the session guarantees required for each session
- Applications can register call back functions for exceptions

Transactions (Database technology)

- A **transaction** is a program unit that must be executed atomically; either the entire unit is executed or none at all. The transaction either completes in its entirety, or it does not (or at least, nothing appears to have happened).
- A transaction can generally be thought of as a sequence of reads and writes, which is either **committed** or **aborted**. A committed transaction is one that has been completed entirely and successfully, whereas an aborted transaction is one that has not. If a transaction is aborted, then the state of the system must be **rolled-back** to the state it had before the aborted transaction began.

ACID semantics

- Atomicity – each transaction is atomic, every operation succeeds or none at all
- Consistency – maintaining correct invariants across the data before and after the transaction
- Isolation - either has the value before the atomic action or after it, but never intermediate
- Durability – persistent on stable storage (backups, transaction logging, checkpoints)

Relaxed semantics

- Relax the ACID constraints

- We could relax consistency for better performance (ala Bayou) where you are willing to tolerate inconsistent data for better performance. For example, you are willing to work with partial calendar update and are willing to work with partial information rather than wait for confirmed data. More on this later on in the course.

Discussion