14-736: DISTRIBUTED SYSTEMS

LECTURE 13 * SPRING 2019 * KESDEN

WHAT ARE DISTRIBUTED FILE SYSTEMS?

- Maybe we'd better start with what are *file systems*?
- Maybe we'd better start with what are *files*?

- A *file* is a unit of data organized by the user. The data within a file isn't necessarily meaningful to the operating system. Instead, a file is created by the user and meaningful to the user. It is the job of the file system to maintain this unit, without understanding or caring why.
  - Contrast this with a record in a database, which is described by a schema and upon which the database enforces constraints

WHAT ARE DISTRIBUTED FILE SYSTEMS?

- Okay. We got *files*.
- So, what are *file systems*?
- A *file system* is a service responsible for managing files. File systems typically implement persistent storage, although volatile file systems are also possible (/proc is such an example).

WHAT DOES IT MEAN TO "MANAGE FILES"?

- Name files in meaningful ways. The file system should allow a user to locate a file using a human-friendly name. In many cases, this is done using a hierarchical naming scheme like those we know and love in Windows, UNIX, Linux, &c.
- Access files. Create, destroy, read, write, append, truncate, keep track of position within, &c
- Physical allocation. Decide where to store things. Reduce fragmentation. Keep related things close together, &c.
- Security and protection. Ensure privacy, prevent accidental (or malicious) damage, &c.
- Resource administration. Enforce quotas, implement priorities, &c.

WHAT ARE DISTRIBUTED FILE SYSTEMS?

- Got *files*.
- Got *file systems*.

- Well, it is a type of *file system*, which means that it is a *file system*.
- The distinction isn't *what it is or what it does* but the *environment in which it does it*.
- A traditional file system typically has all of the users and all of the storage resident on the same machine. A distributed file system typically operates in an environment where the data may be spread out across many, many hosts on a network -- and the users of the system may be equally distributed.

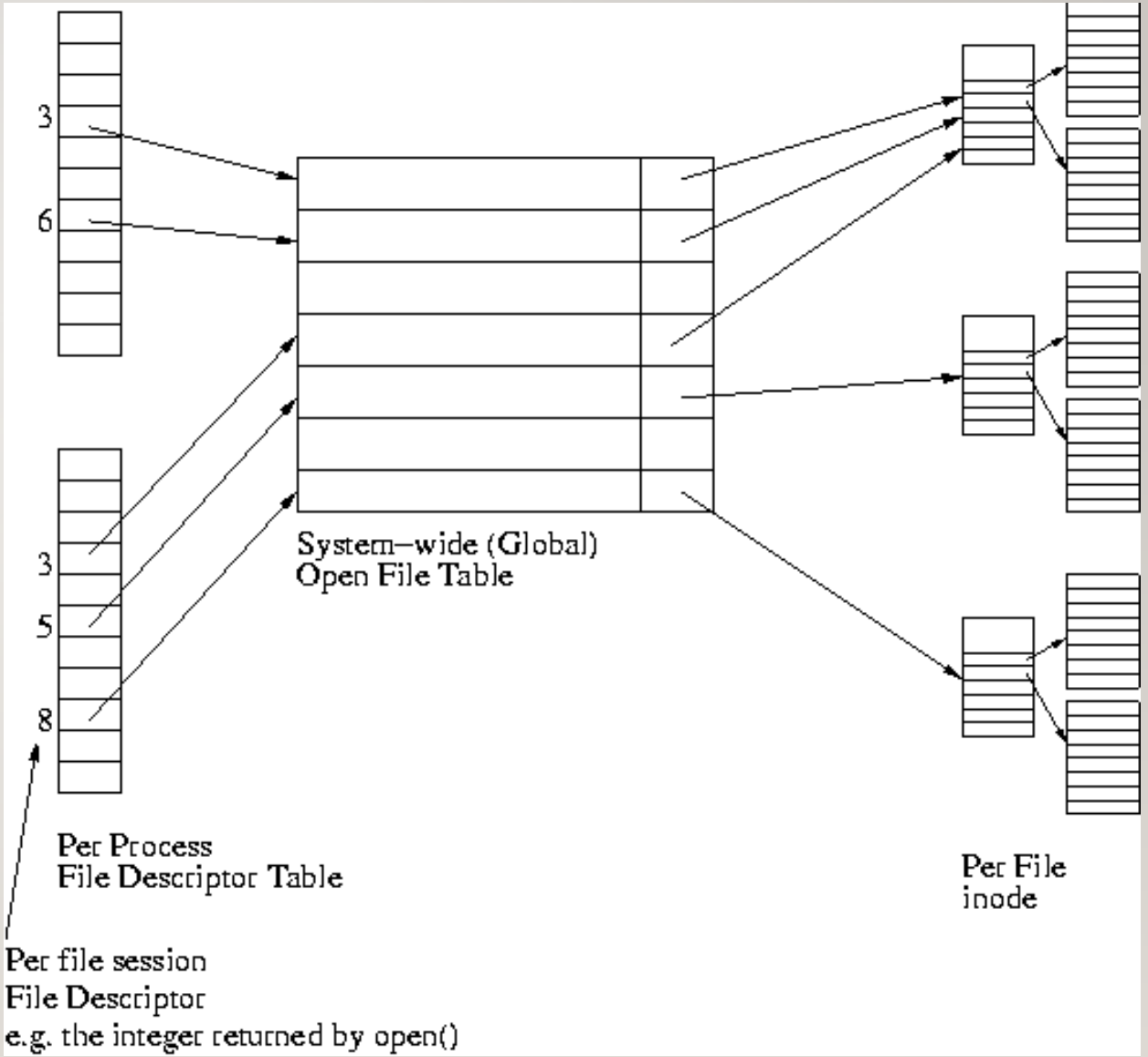
GOAL OF A GENERAL PURPOSE DISTRIBUTED FILE SYSTEM (DFS)

- A general purpose distributed file system should appear to the user as if a local file system
- Users shouldn't have to worry about details such as
  - Where files are located
  - How to identify the locations
  - How many replicas there are
  - If the replicas are up to date
  - What happens if part of the system fails, e.g. partitioning, host failure, etc.

WHY MIGHT WE WANT A GENERAL PURPOSE DISTRIBUTED FILE SYSTEM?

- More storage
- More throughput
- More robustness from host and storage failure
- More robust from environmental hazards
- Faster access, e.g. closer to users
- Etc.

RECALL, FOR EXAMPLE, UNIX-STYLE FILE SYSTEMS



UNIX-LIKE FILE SYSTEMS, CONTINUED

- "Virtual File System (VFS)"
- File system, inode are base classes
  - Known as "Virtual file system (VFS)" and vnode
  - Specific implementation is derived type
  - Implementation may be legit OO, e.g. C++
  - Or, implementation may, for example, ugly C with unions, etc.
- Implementation abstracted from interface
- Distributed File Systems can be just an alternative implementation for storage

GENERAL PURPOSE DISTRIBUTED FILE SYSTEM

- Maintain same naming system
  - "Mount" in UNIX
  - "Map" in Windows, etc
- Just change storage layer to use network and manage errors

NETWORK FILE SYSTEM (NFS) "EARLY VERSIONS"

- Not a DFS
- Simply central storage accessed via network
- Client made requests on per-block basis
- "Stateless server"
- No client caching
  - Turned out to be too painful
  - Clients cached anyway
  - Just accepted staleness for a while since no way to validate
- Evolved over time into DFS with support for caching

ANDREW FILE SYSTEM (AFS) "OUR FRIEND"

- Stateful server
  - Callback mechanism
  - Invalidates client caches upon change, but in use copies unaffected
- Whole-file semantics
  - Modified to move blocks in version 2.x (when it left CMU and went to IBM)

CODA

- Extension of AFS Project
- Added replication
- Added weakly connected mode: One server replicates on behalf of client
- Added disconnected mode: Uploaded and resolved by client later
- Hoard demon: Pull whole files needed for later
- Spy: Figure out which files to hoard.
- Client-side resolution
- “Insert CVV discussion again here”