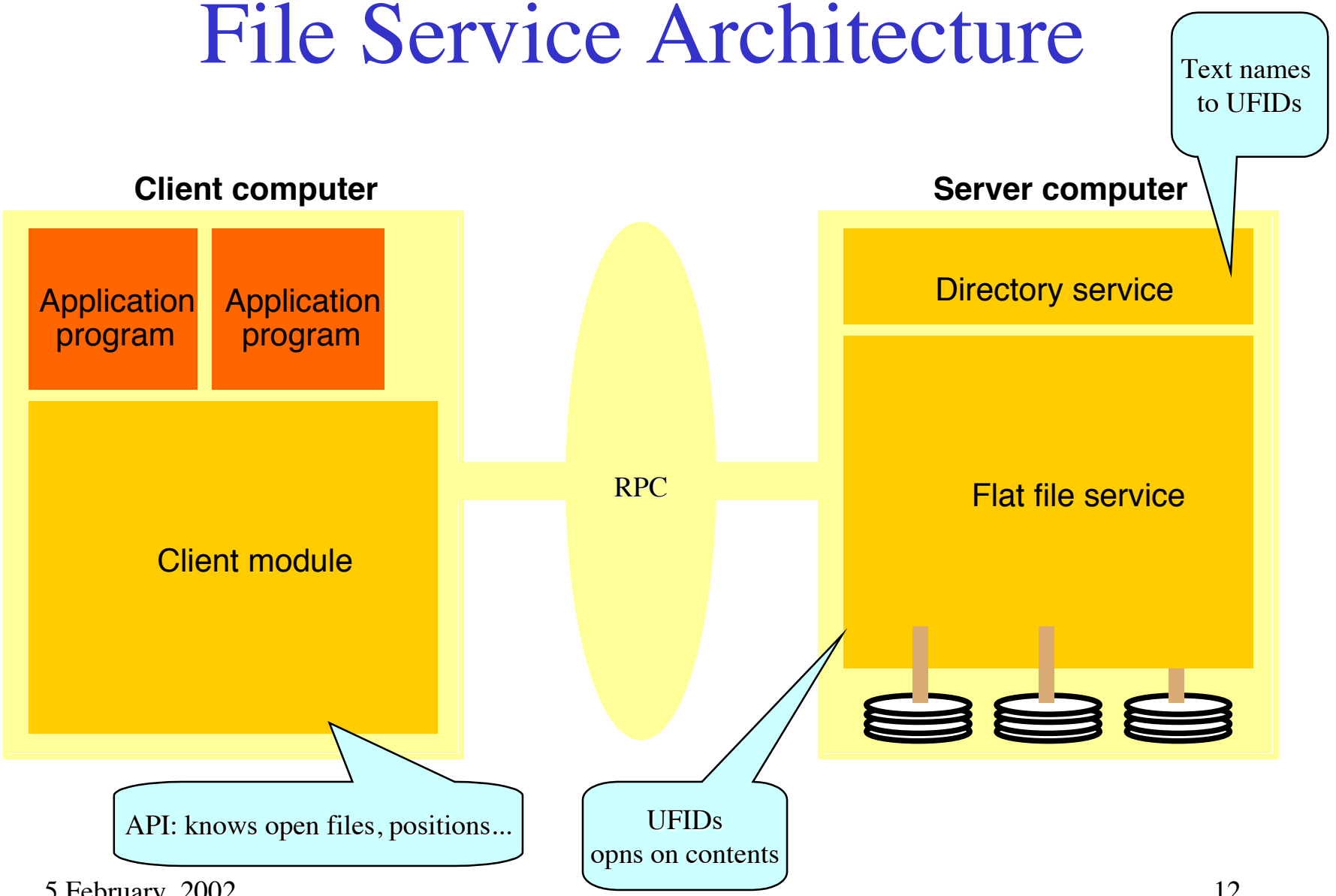
File Service Architecture



File server architecture

Components (for openness):

- Flat file service
  - operations on file contents
  - unique file identifiers (UFIDs)
  - translates UFIDs to file locations
- Directory service
  - mapping between text names to UFIDs
- Client module
  - API for file access, one per client computer
  - holds state: open files, positions
  - knows network location of flat file & directory server

Flat file service RPC interface

- Used by client modules, not user programs
  - *FileId* (UFID) uniquely identifies file
  - invalid if file not present or inappropriate access
  - *Read*/*Write*; *Create*/*Delete*; *Get*/*SetAttributes*
- No open/close! (unlike UNIX)
  - access immediate with *FileId*
  - *Read*/*Write* identify starting point
- Improved fault-tolerance
  - operations idempotent except Create, can be repeated (at-least-once RPC semantics)
  - stateless service

Flat file service operations

Read(FileId, i, n) -> Data — throws *BadPosition*	If $1 \leq i \leq Length(File)$: Reads a sequence of up to *n* items from a file starting at item *i* and returns it in *Data*.
Write(FileId, i, Data) — throws *BadPosition*	If $1 \leq i \leq Length(File)+1$: Writes a sequence of *Data* to a file, starting at item *i*, extending the file if necessary.
Create() -> FileId	Creates a new file of length 0 and delivers a UFID for it.
Delete(FileId)	Removes the file from the file store.
GetAttributes(FileId) -> Attr	Returns the file attributes for the file.
SetAttributes(FileId, Attr)	Sets the file attributes (only owner, file type and access control list; rest maintained by flat file service).

Access control

- In UNIX file system
  - access rights are checked against the access mode (read, write, execute) in open
  - user identity checked at login time, cannot be tampered with
- In distributed systems
  - access rights must be checked at server
    - RPC unprotected
    - forging identity possible, a security risk
  - user id typically passed with every request (e.g. Sun NFS)
  - stateless

Directory structure

- Hierarchical
  - tree-like, pathnames from root
  - (in UNIX) several names per file (*link* operation)
- Naming system
  - implemented by client module, using directory service
  - root has well-known UFID
  - locate file following path from root