Distributed Systems

ID2201

distributed file systems

Johan Montelius

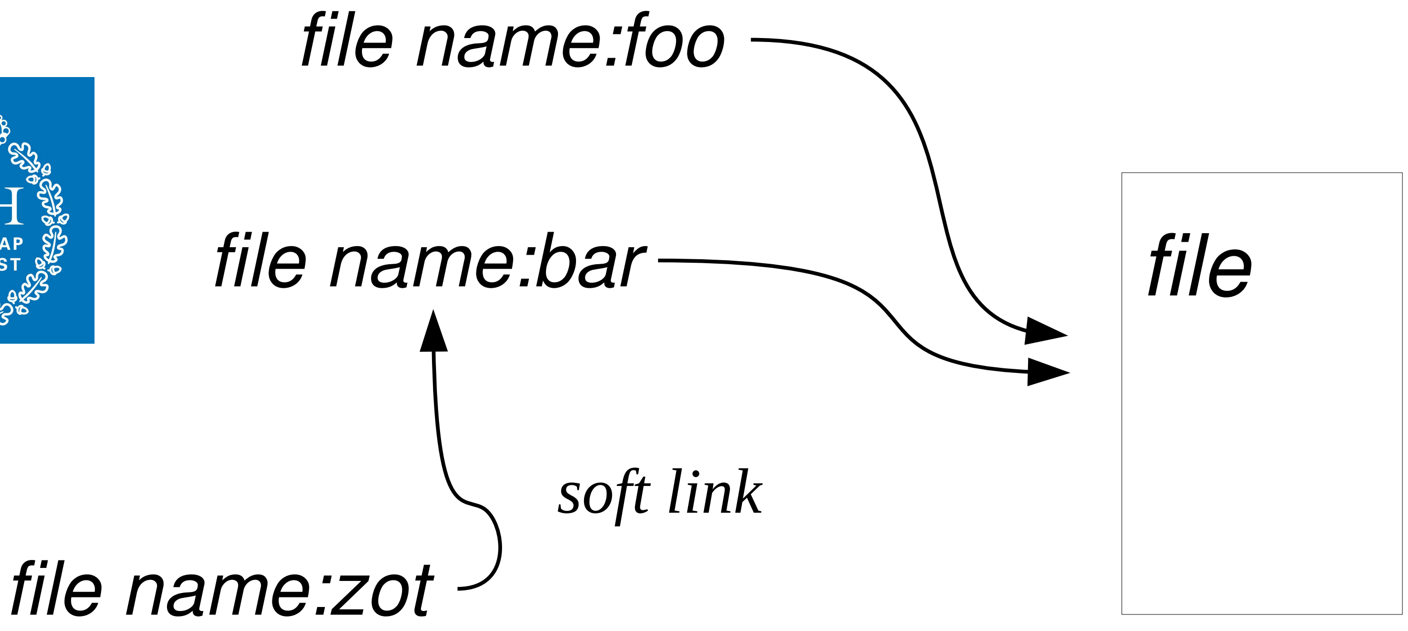
so what's a file

- a sequence of bytes
  - does it have to be of finite length?
- associated *meta-data*
  - size and type
  - owner and permissions
  - author
  - created, last written
  - icons, fonts, presentation....

so what's a file system

- Procedures for:
  - creating and deleting a file
  - manipulating the content of a file
  - associating a file with a name
  - organizing files
  - checking authentication and authorization

file names and files



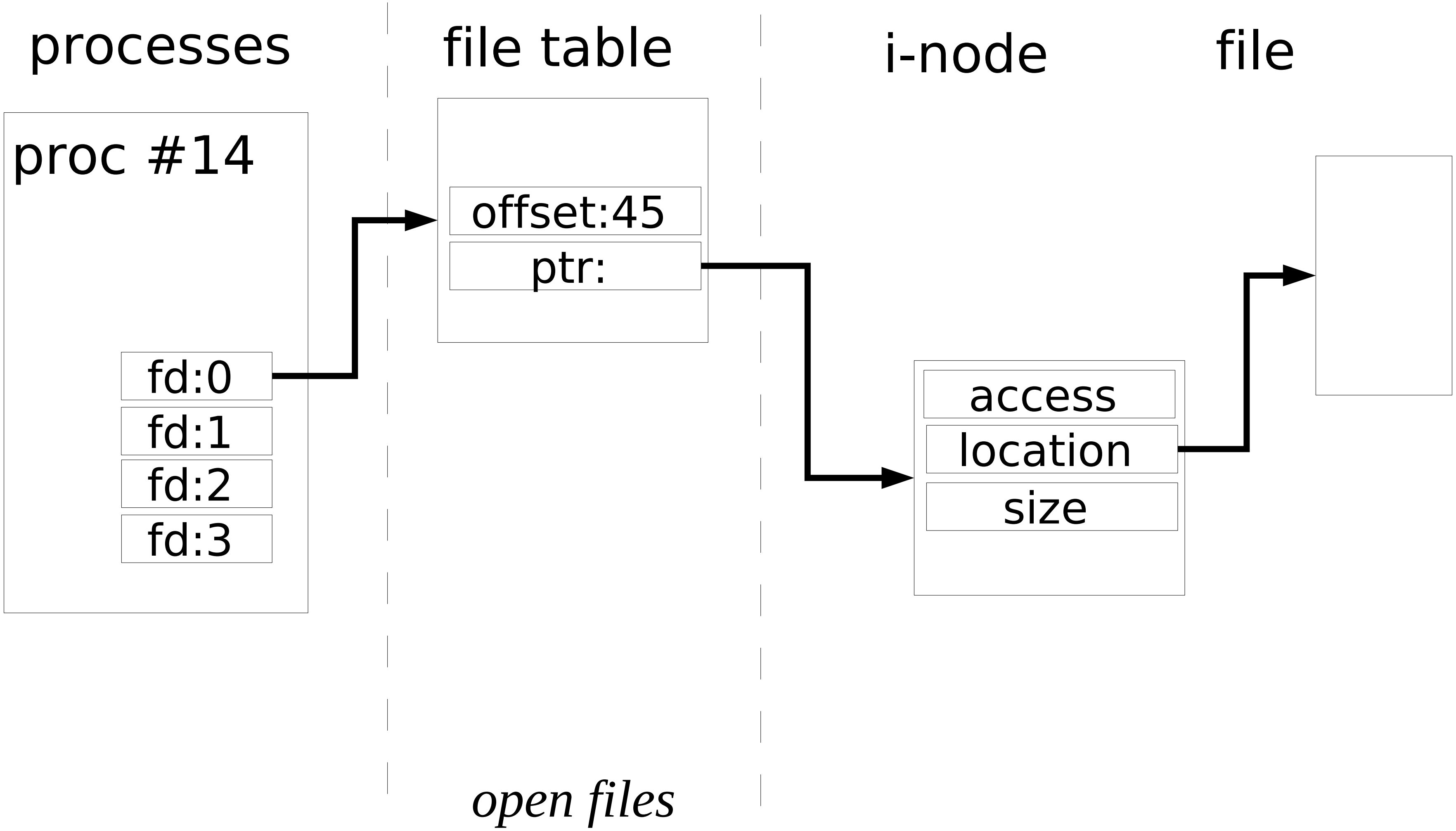
the API in UNIX

- *fd* = create(name, mode)
- *fd* = open(name, mode)
- *status* = close(fd)
- *count* = read(fd, buffer, n)
- *count* = write(fd, buffer, n)
- *offset* = lseek(fd, offset, set/cur/end)
- *status* = unlink(name)
- *status* = link(name, name)
- *status* = stat(name, buffer)
- ... locking?

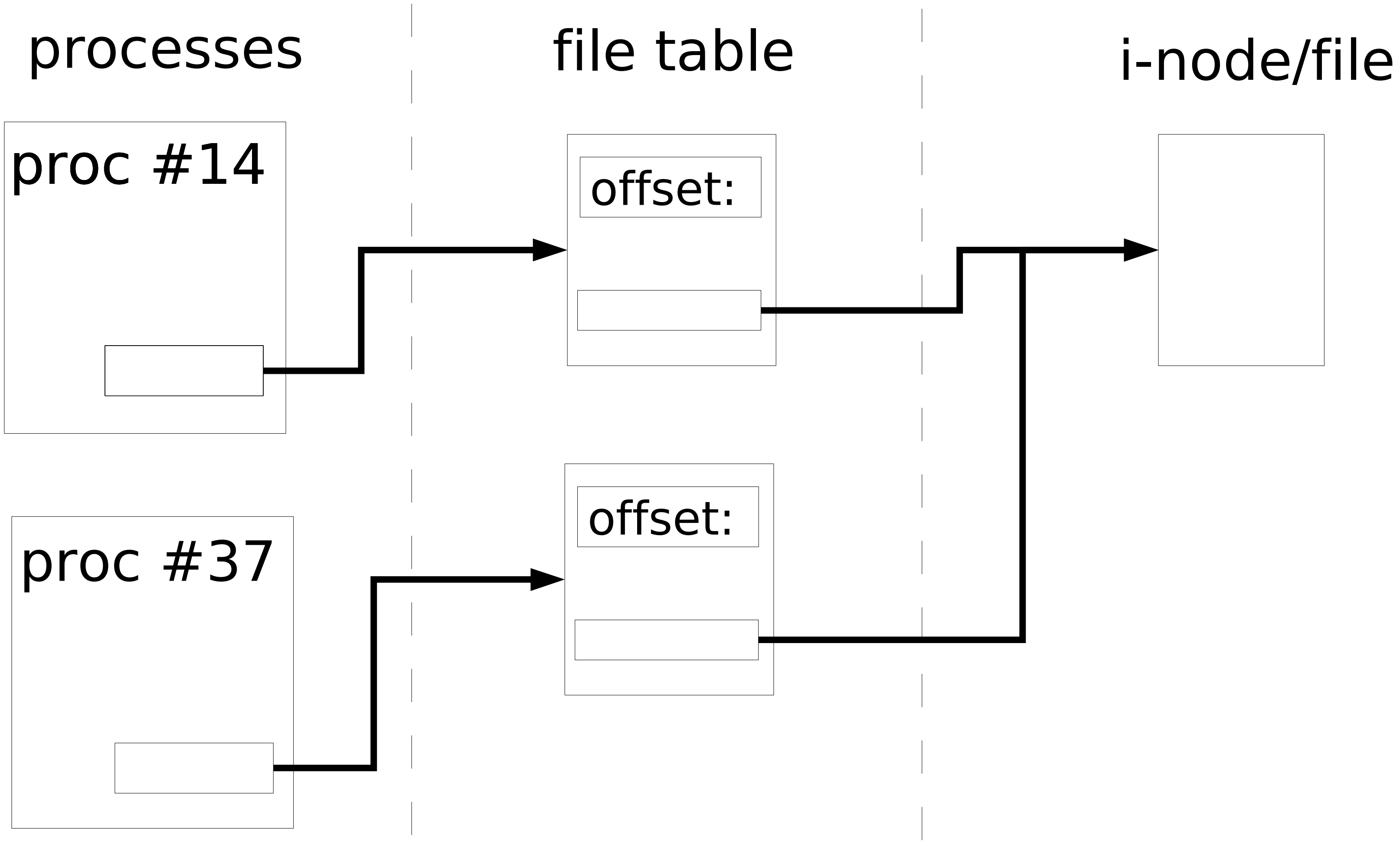
programing language API

- Programing languages often provide an API that improves file operations.
- Buffering of write operations to reduce the number of system calls.
- Sometimes you must do things by hand, for example flushing of buffers.
- Language API could be limited in giving you full access to operating system API.

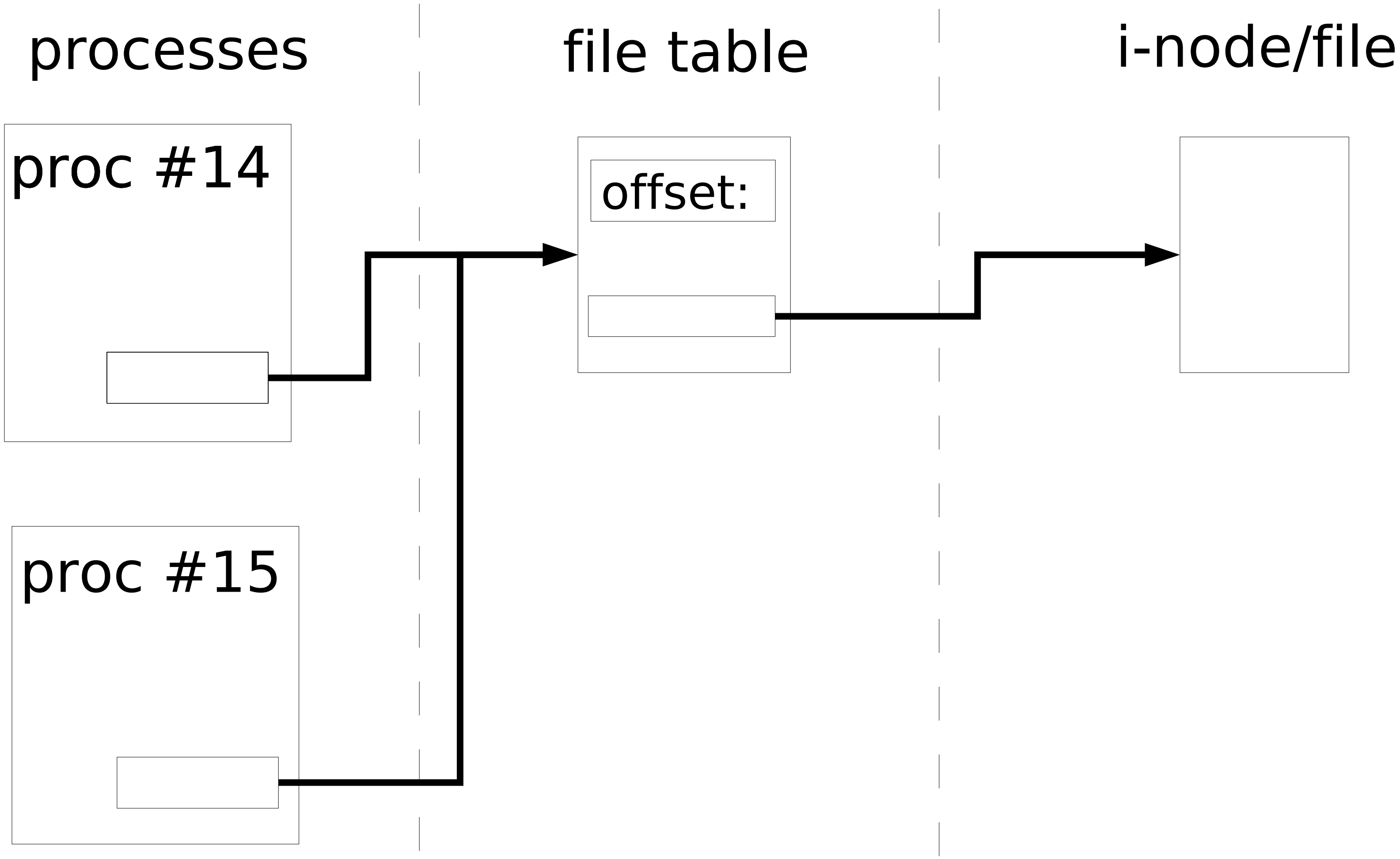
Operating systems and files



sharing files: open twice



sharing files: fork



one-copy semantics

- Most file systems give us a *one-copy semantics*
  - *we expect operations to be visible by everyone and that everyone see the same file*
  - *if I tell you that the file has been modified the modification should be visible*
- We might be surprised when other processes access the file but this does not violate the one-copy view.

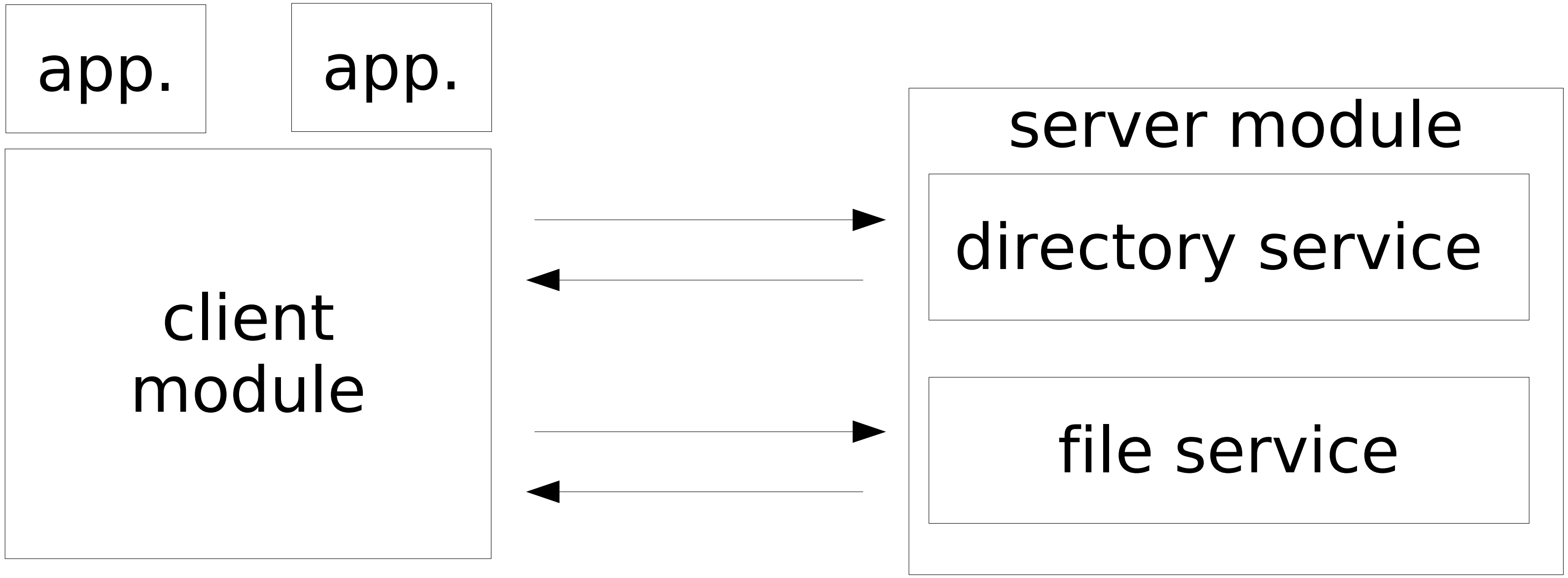
consistency

- How do we provide a consistent interface to shared files?
- What does it mean to set the offset to the last position in the file?
- We need atomic operations that perform several operations in a unbroken sequence.
  - open, or if it does not exists then create
  - append to the end of the file

Distributed File System

- Challenge:
  - make a file system available to several distributed clients
- Semantics:
  - keep the one-copy view
- Performance:
  - comparable to non-distributed

Distributed architecture



directory service: maps file names to *unique file descriptors*

file service: performs operations on files given file descriptors

the directory service

- What operations do we need?
  - Lookup(Dir, Name) -> FileId
  - AddName(Dir, Name, FileId)
  - UnName(Dir, Name)
  - GetNames(Dir) -> NameSeq
- Note – the directory service only handles how names are mapped to file identifiers.

the file service

- What operations should be provided
  - create and delete
  - read and write
- Do we need a *open* operation?
  - What does open do in Unix?
  - What do we need if we don't have an open operation?
  - What would the benefit be?

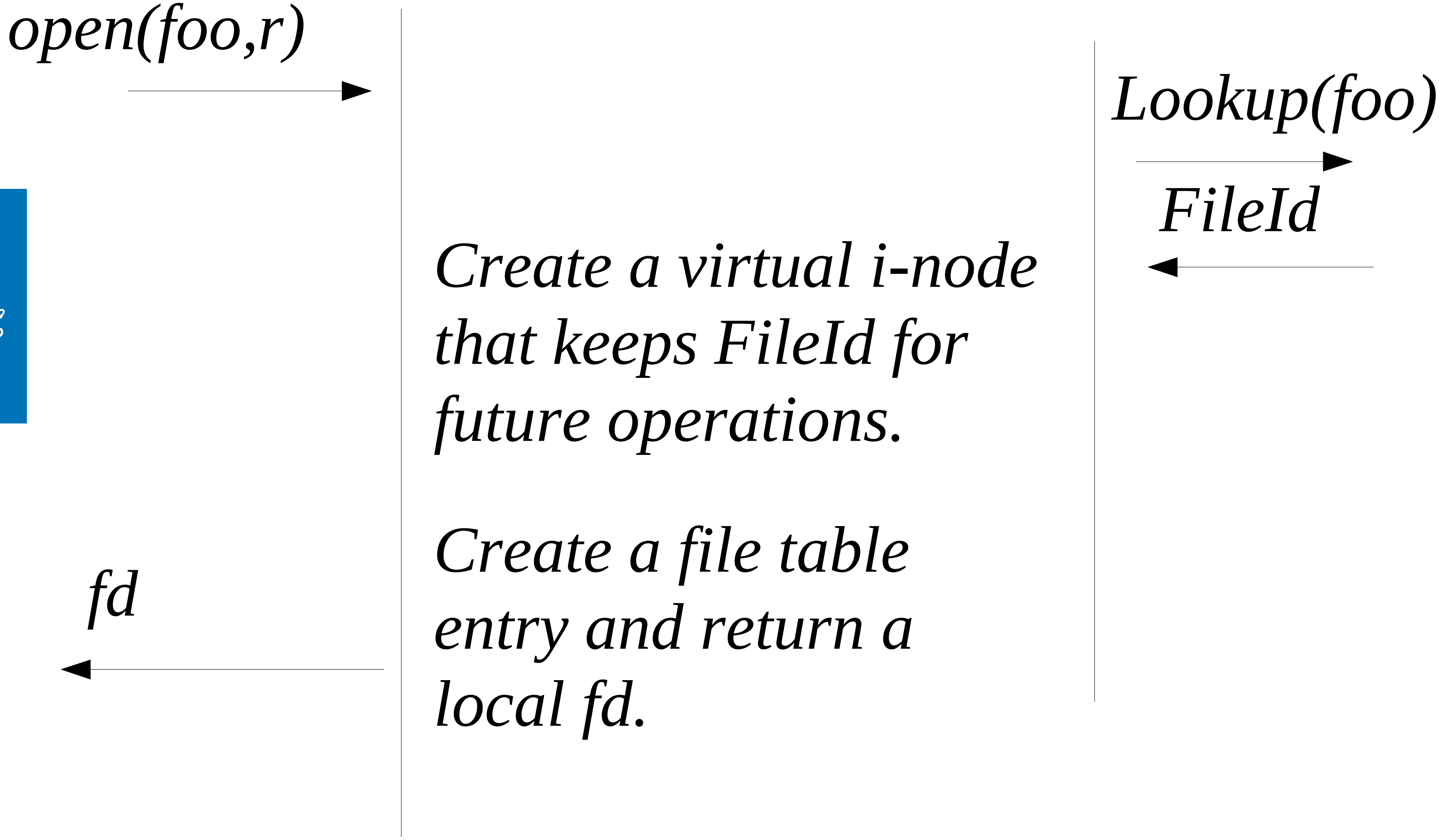
stateless server

- What are the benefits of a stateless server?
  - In what sense is it stateless?
- How can we maintain a *session state* while keeping the server stateless?
- Give me examples of other systems where this is used.

How to handle security?

- In Unix, permissions are checked when a file is opened.
- Access to the file can then be done without security control.
- If we do not have a open operation how can we perform authentication and authorization control?

Open



open(foo,r)

Lookup(foo)

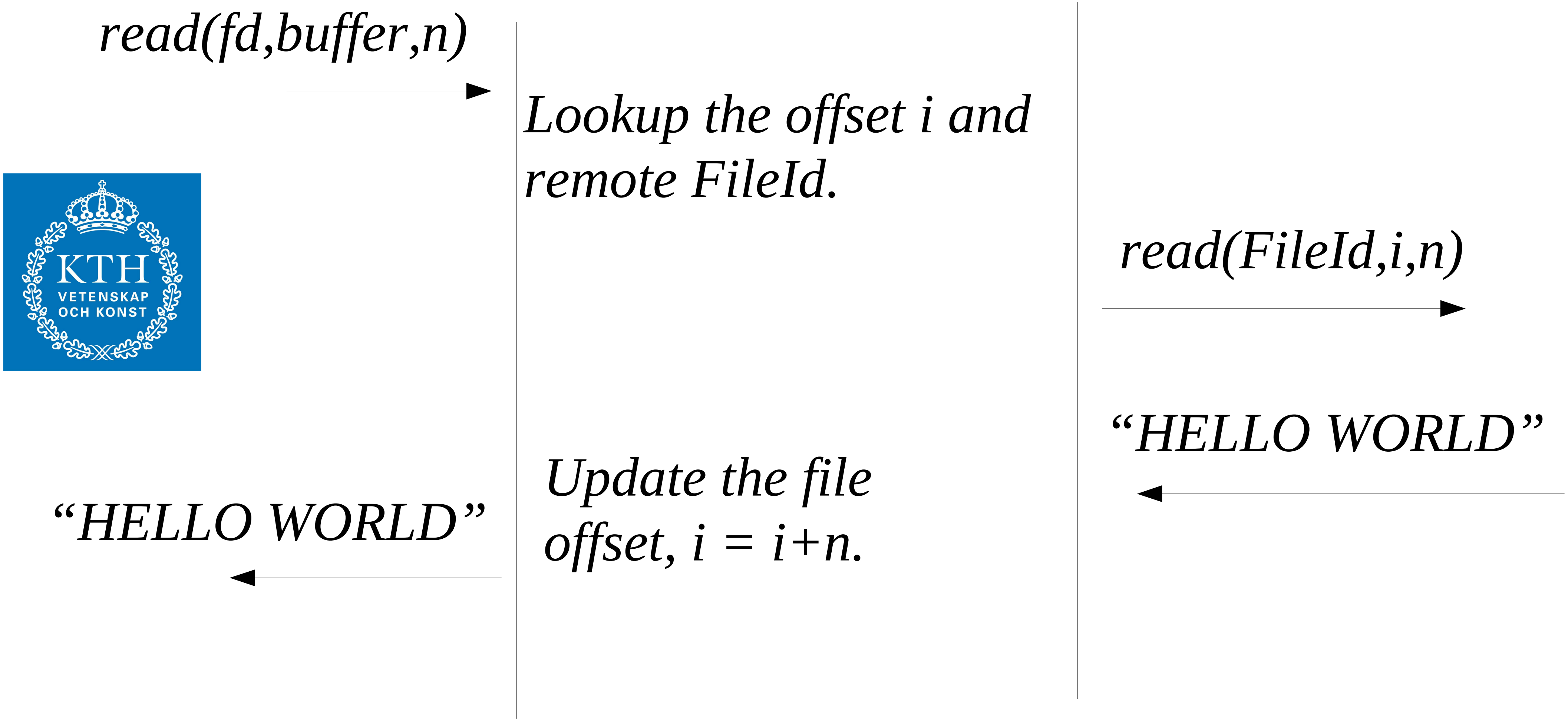
FileId

Create a virtual i-node that keeps FileId for future operations.

Create a file table entry and return a local fd.

fd

Read



read(fd,buffer,n)

Lookup the offset i and remote FileId.

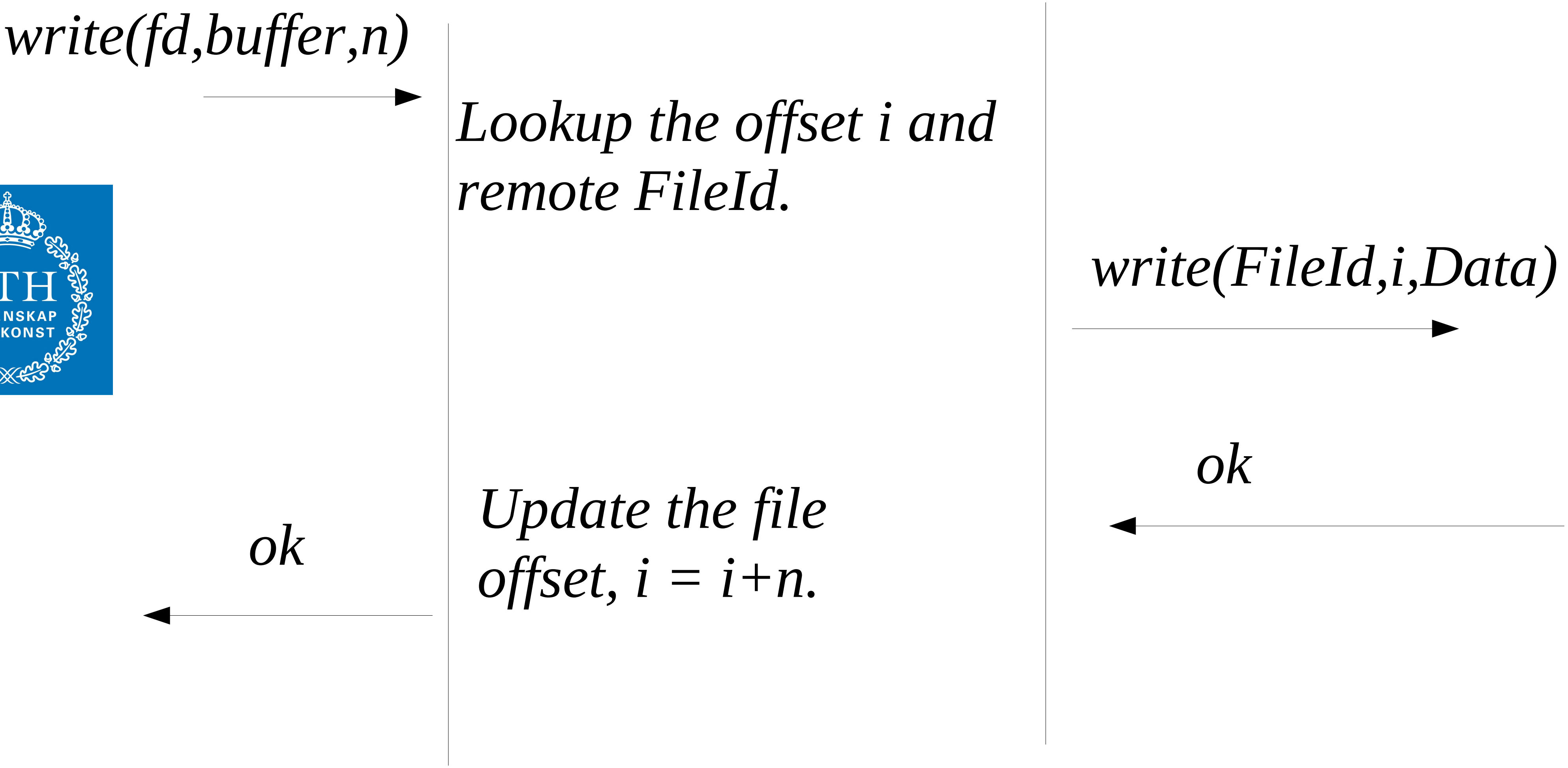
read(FileId,i,n)

“HELLO WORLD”

Update the file offset, i = i+n.

“HELLO WORLD”

Write



write(fd,buffer,n)

Lookup the offset i and remote FileId.

write(FileId,i,Data)

ok

Update the file offset, i = i+n.

ok

Problems

- Network
  - What happens if a *write message* is lost?
- Authorization
  - How do we know that a client has authorization?
- Performance
  - Every read and write operation will now require a message round trip.

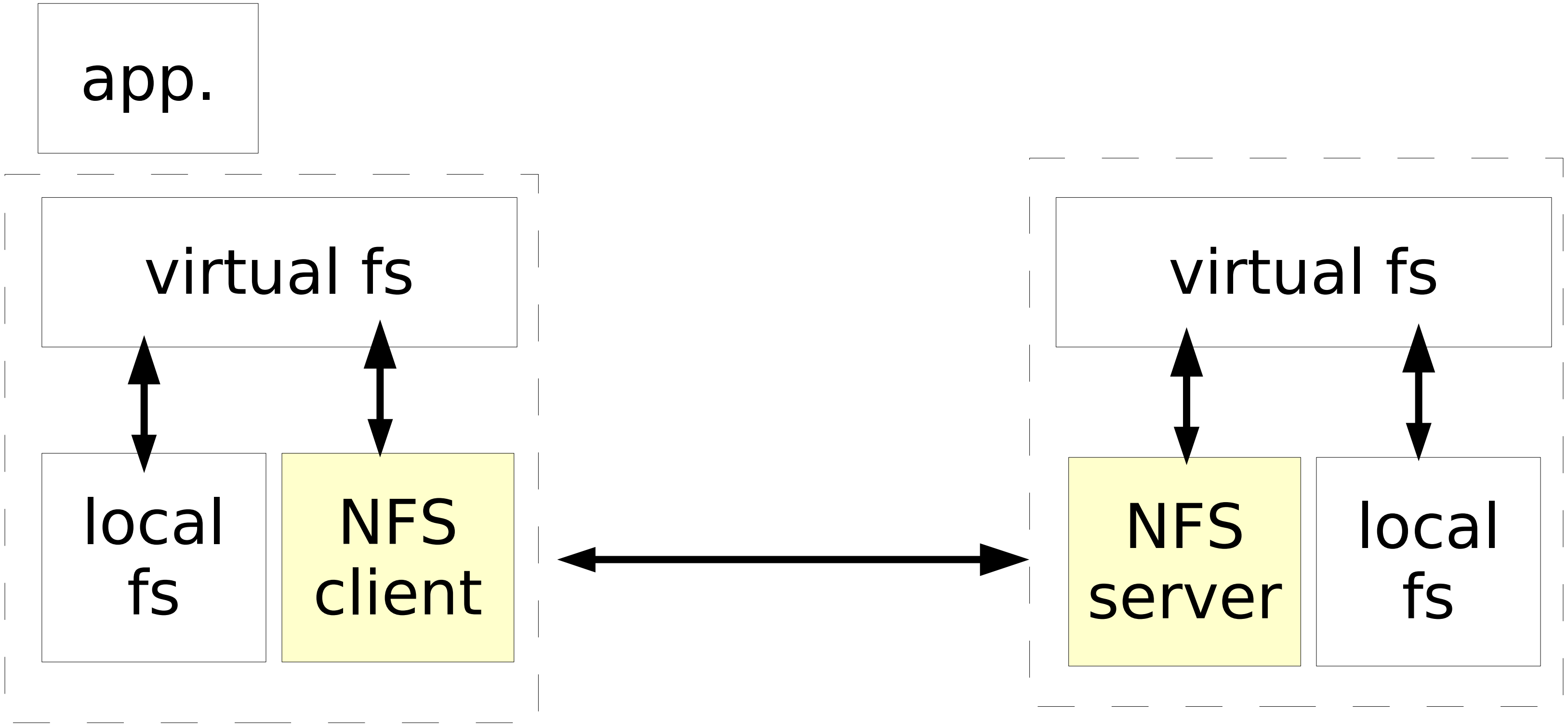
Caching

- Keep read sections of a file in a client cache.
- Read and write operations can now be done locally if segment is in the cache.
- Consistency
  - How do we know that the file has not been changed on the server?
  - If we write to a file in cache only, no one will see it.
  - Can we have two copies of a file?

NFS – network file system

- Developed by Sun, 1984
- Implemented using RPC (*Open Network Computing*)
- Public API: RFC 1094, 1813, 3530
- Originally used UDP, later versions have support for TCP to improve performance over WAN
- Mostly used with UNIX systems but client on all platforms available.

NFS


app.
virtual fs
local fs
NFS client
virtual fs
NFS server
local fs

NFS server caching

- Server will open files on request and keep a copy in memory.
- When are write operations performed?
  - for each client write?
  - when a file is closed? (is there a close?)
  - *write-through* or *commit* when closed
- What happens if the server crashes?

NFS client caching

- In a read operation a segment (8Kbyte) is cached by the client.
- Read operations can read from the cache if the entry is still *valid* (more on this).
- Write operations are done locally and the segment is scheduled to be sent to the server.

NFS entry validation

- Each cache entry have two time-stamps.
  - Tc : time the entry was validated
  - Tm: time file modified by server
- An entry is valid at time T if either:
  - T – Tc < some value *t* *(3-30s)*
  - *Tm(at server) = Tm(at client)*
- If Tm is checked then Tc is set to T.
- If segment has been modified a new copy is read.

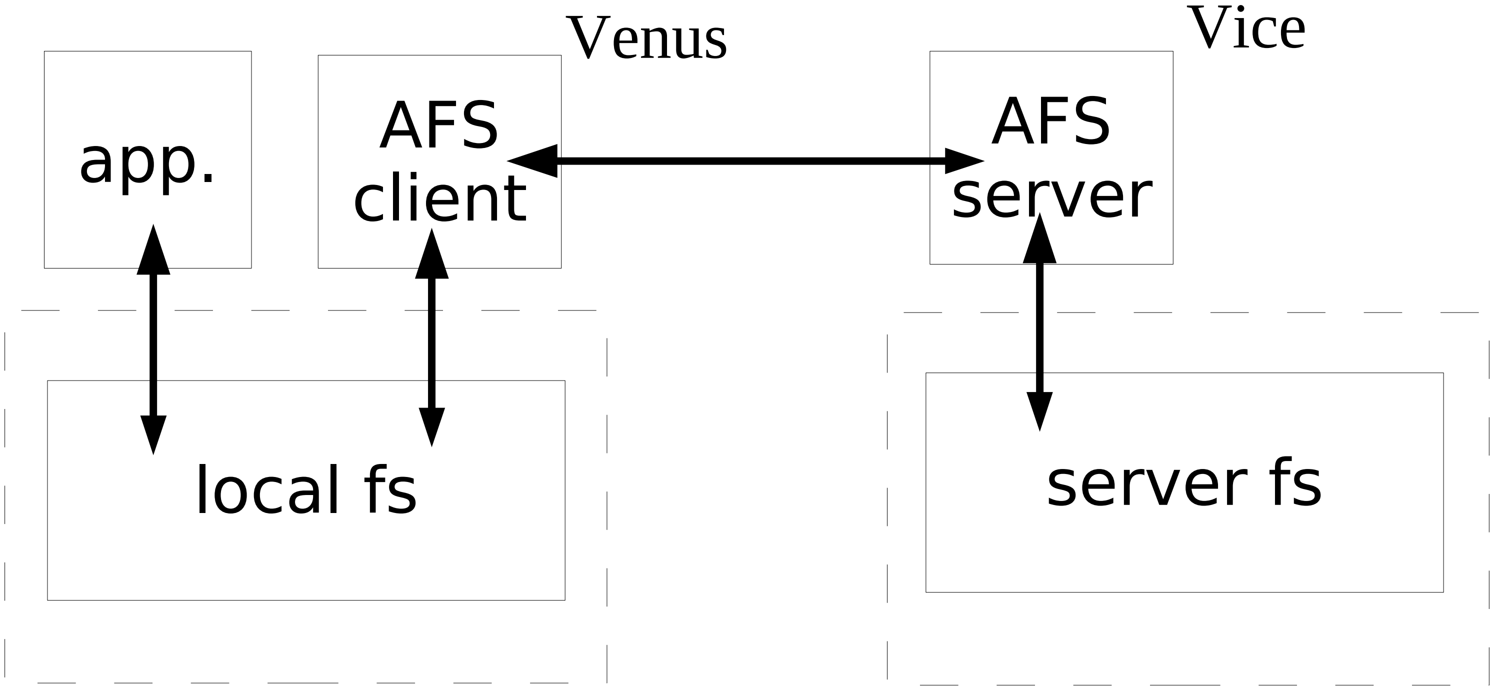
NFS – caching and consistency

- How could we have an inconsistency?
- How much do we dare to gamble?
- How are write operations handled:
  - When should we update the server?
  - Can we loose data?
  - When are we sure that our write operation succeeded?
- NFS tries to provide read and write operations that give us a one-copy semantics.

AFS – Andrew File System

- Developed by Carnegie Mellon University
- Clients for most platforms, OpenAFS (from IBM), Arla (a KTH implementation).
- Used mainly in WAN (Internet) where the overhead of NFS would be prohibitive.
- Relies on caching of whole files and infrequent sharing of writable files.

AFS



AFS - caching

- Opening a file will create a copy in the local file system.
- All read/write operations directed to the local copy.
- Server will notify a client if another client modifies the same file, a *callback promise*.
- A closed modified file is copied back to the server.
- A client can flush a file and thereby force a copy to the server.

AFS – callback promise

- Each cached copy is tagged with a *local modified time* and a *valid* or *canceled promise*.
- A cached file can be re-opened if the promise is valid.
- Promises can be canceled by the server or by a time out (*few minutes*).
- Canceled promises can be made valid after asking the server if the local modified time is the most resent copy.

AFS -consistency

- How much is a promise worth?
  - Does a promise prevent other clients from modifying the file?
  - What happens if two clients update the same file?
  - What if a call-back is lost?
  - In which situations will AFS not work?
  - Only close() or also fsync()?

SMB

- Service Message Block (SMB) was originally developed by IBM but then modified by Microsoft, now also under the name Common Internet File System (CIFS).
- Not only file sharing but also name servers, printer sharing etc.
- Samba is an open source reimplementation of SMB by Andrew Tridgell.

SMB semantics

- SMB uses client locks to solve the one-copy view problem.
  - A client can open a file an lock it; all read and write operations in client cache.
  - A read only lock will allow multiple clients to cache and read a file.
  - Locks can be revoked by the server forcing the client to flush any changes.
- In a unreliable or high latency network, locking can be dangerous and counter productive.

More distributed file systems

- Reliability
  - Google File System
- Mobility
  - Unison
- Web servers and proxies
  - Squid
- Version control
  - CVS, Subversion

Summary

- We would like to provide a transparent file system
  - one-copy view
  - performance
- Caching is key to performance but makes a one-copy view hard to maintain.
- Different usage pattern and network properties could require different solution.