IF3230 – Sistem Terdistribusi

Distributed File Systems

Achmad Imam Kistijantoro (imam@informatika.org)

Judhi Santoso (judhi@informatika.org)

Anggrahita Bayu Sasmita (bayu.anggrahita@informatika.org)

Topik

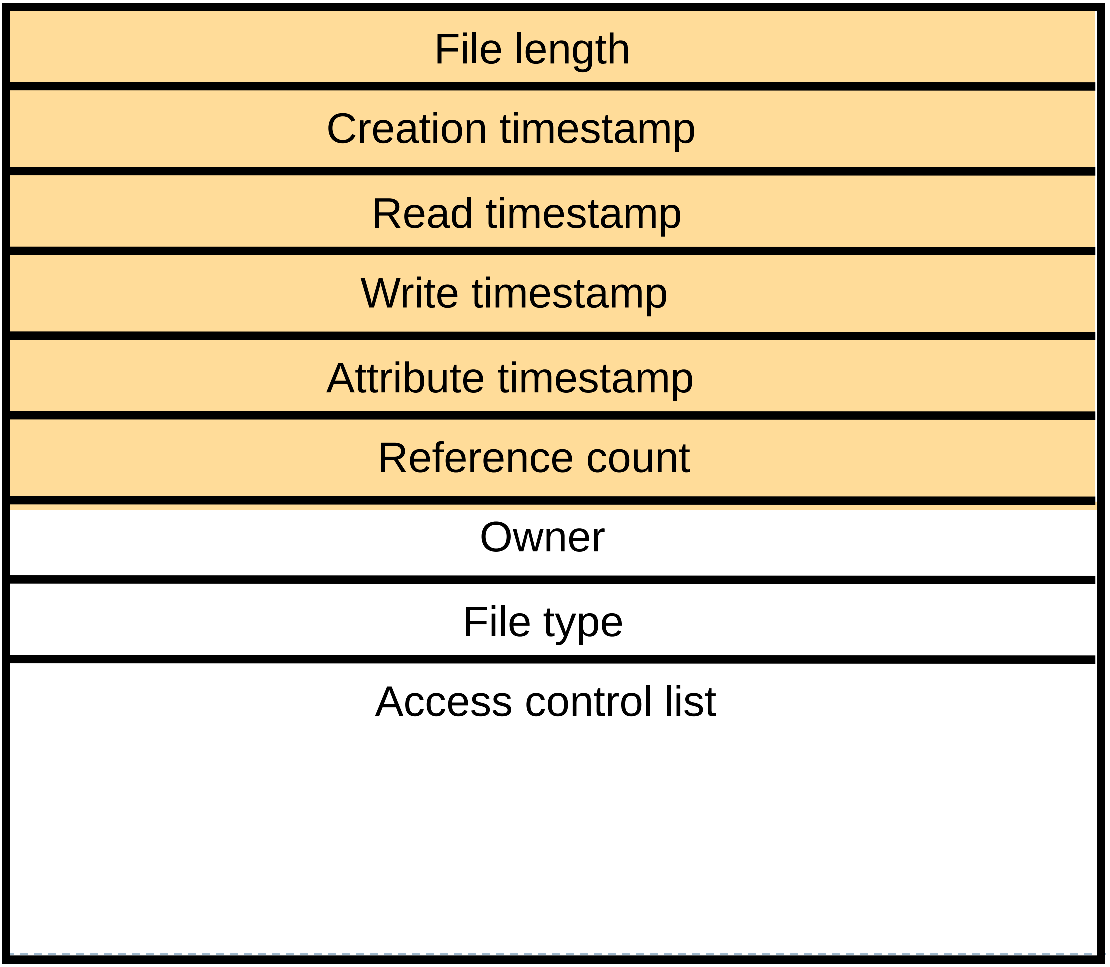
- Generic Distributed File System
- NFS Network File System
- AFS Andrew File System
- Google File Systems
- Hadoop Distributed File System (HDFS)

Typical File System Abstractions

- Implementasi file systems umumnya dilakukan dengan model layer
  - Modul direktori: mapping nama file ke file id
  - Modul file: mapping id file ke data file tertentu
  - Modul Access control: memeriksa ijin operasi yang akan dilakukan
  - Modul file access: baca/tulis data/atribut file
  - Modul blok: akses dan alokasi blok disk
  - Modul device: disk I/O dan buffering

Characteristics File Systems

- File systems mengelola data dan atribut pada file



File length

Creation timestamp

Read timestamp

Write timestamp

Attribute timestamp

Reference count

Owner

File type

Access control list

Operasi pada File Systems Linux

- Diimplementasikan pada kernel, menyediakan layanan lokal

filedes = open(name, mode) *filedes = creat(name, mode)*	Opens an existing file with the given *name*. Creates a new file with the given *name*. Both operations deliver a file descriptor referencing the open file. The *mode* is *read*, *write* or both.
status = close(filedes)	Closes the open file *filedes*.
count = read(filedes, buffer, n) *count = write(filedes, buffer, n)*	Transfers *n* bytes from the file referenced by *filedes* to *buffer*. Transfers *n* bytes to the file referenced by *filedes* from buffer. Both operations deliver the number of bytes actually transferred and advance the read-write pointer.
pos = lseek(filedes, offset, whence)	Moves the read-write pointer to offset (relative or absolute, depending on *whence*).
status = unlink(name)	Removes the file *name* from the directory structure. If the file has no other names, it is deleted.
status = link(name1, name2)	Adds a new name (*name2*) for a file (*name1*).
status = stat(name, buffer)	Gets the file attributes for file *name* into *buffer*.

Distributed File Systems

- Why?

- Sharing: file dapat diakses oleh banyak orang
  - File dapat dikirim via email, whatsapp … update problem

- Mobility: bekerja di rumah, kantor, berbeda alat

- Scalability: ukuran, akses banyak

Distributed File Systems

- Challenges
  - Performance
  - Availability
  - Consistency
  - Fault Tolerance
  - Reliability
  - Simplicity

Distributed File Systems

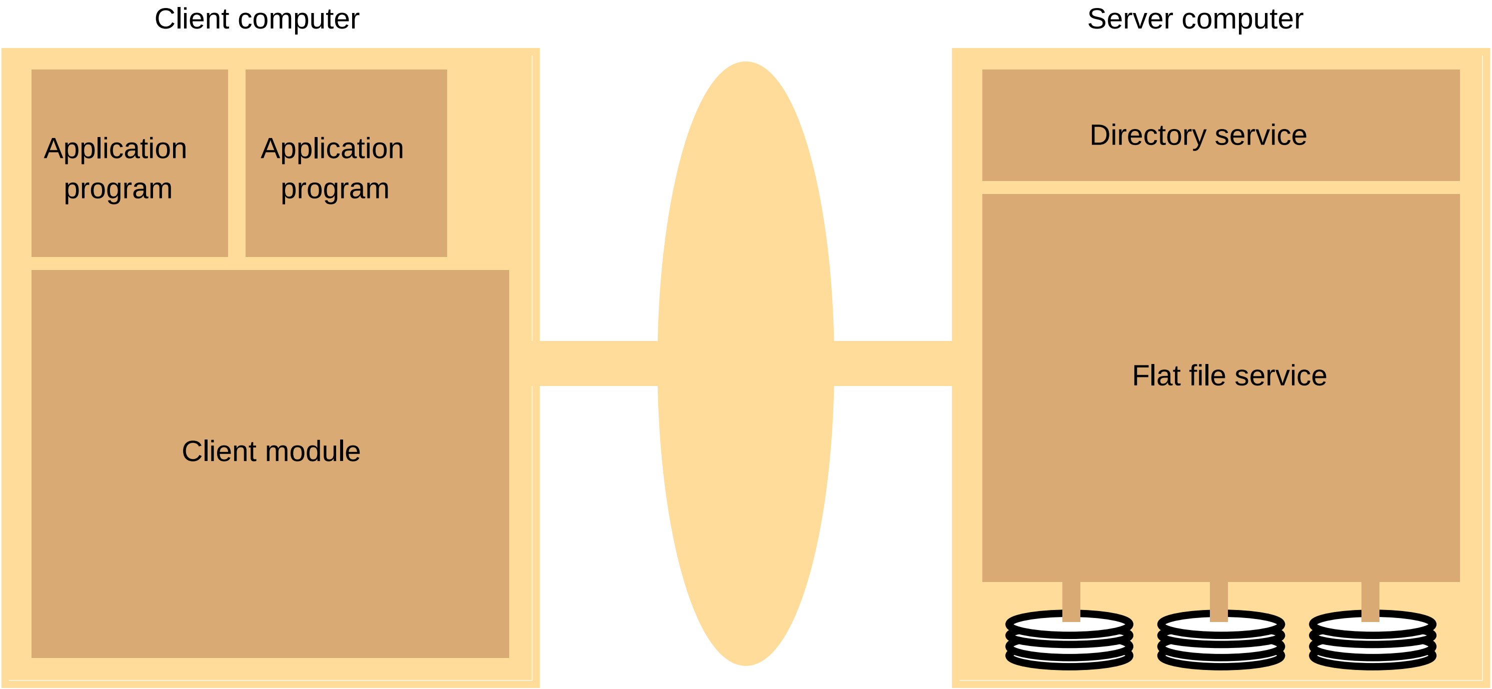
- Simplicity
- Implement berbasis client server
- Menggunakan RPC
- Akses ke file system dipecah menjadi client yang memanggil RPC services ke server

- NFS v2

- File system implementation: Flat file services dan Directory services

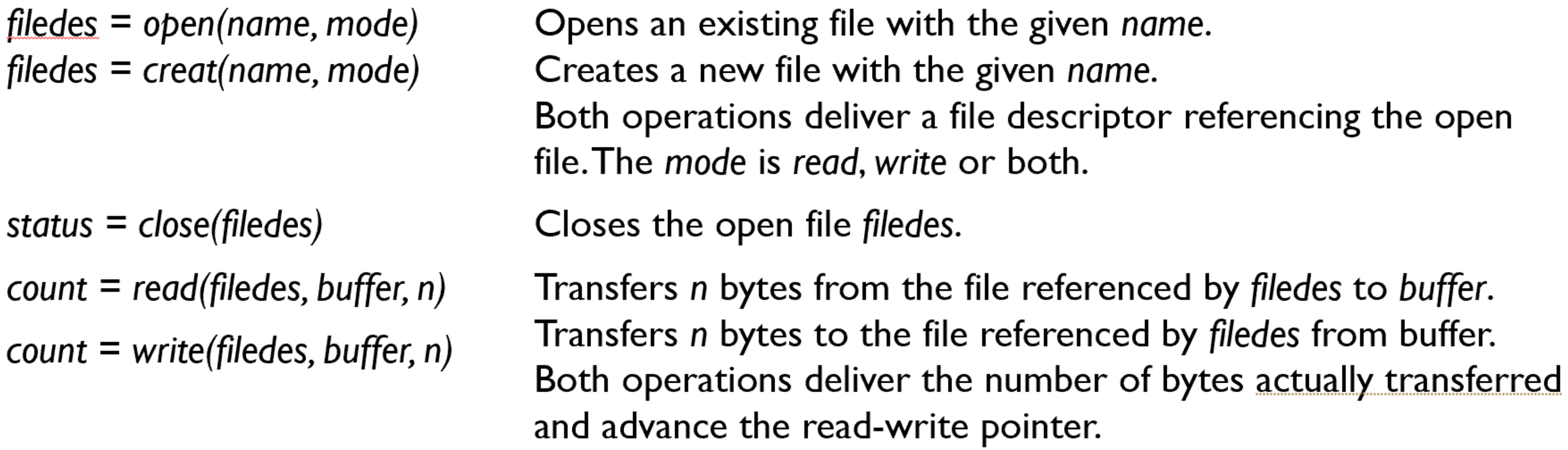
File Service Architecture

- File service dapat distrukturkan menjadi 3 komponen
  - Flat file service
  - Directory service
  - Client module



Distributed File Systems

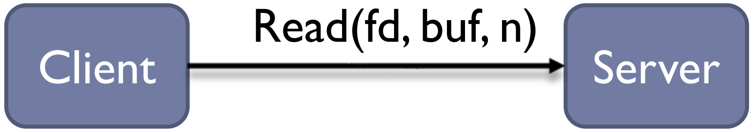
- Fault Tolerance
- Bagaimana menangani jika ada error?
- Layanan file system local menggunakan file descriptor
- Saat file dibuka (open), OS menyimpan struktur data ttg file tsb, dan file descriptor digunakan aplikasi sebagai pointer/reference untuk mengakses file



filedes = open(name, mode) *filedes = creat(name, mode)*	Opens an existing file with the given *name*. Creates a new file with the given *name*. Both operations deliver a file descriptor referencing the open file. The *mode* is *read*, *write* or both.
status = close(filedes)	Closes the open file *filedes*.
count = read(filedes, buffer, n) *count = write(filedes, buffer, n)*	Transfers *n* bytes from the file referenced by *filedes* to *buffer*. Transfers *n* bytes to the file referenced by *filedes* from buffer. Both operations deliver the number of bytes actually transferred and advance the read-write pointer.

Distributed File Systems – Fault Tolerance

- Apa yang terjadi jika jaringan terputus saat pembacaan/penulisan file?
  - Retry?

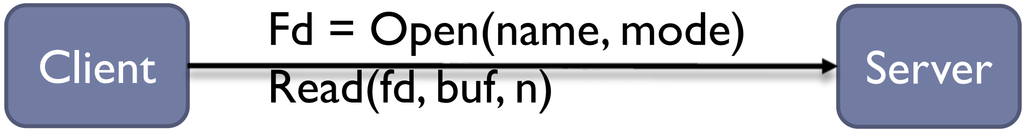


Client

Read(fd, buf, n)

Server

- Apa yang terjadi jika server down saat pembacaan/penulisan file?



Client

Fd = Open(name, mode)

Read(fd, buf, n)

Server

Layanan flat file service

- idempotent, stateless server, access control

Read(FileId, i, n) -> Data — throws *BadPosition*	If *1* ≤ *i* ≤ *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* ≤ *i* ≤ *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 those attributes that are directly manimulated by users)

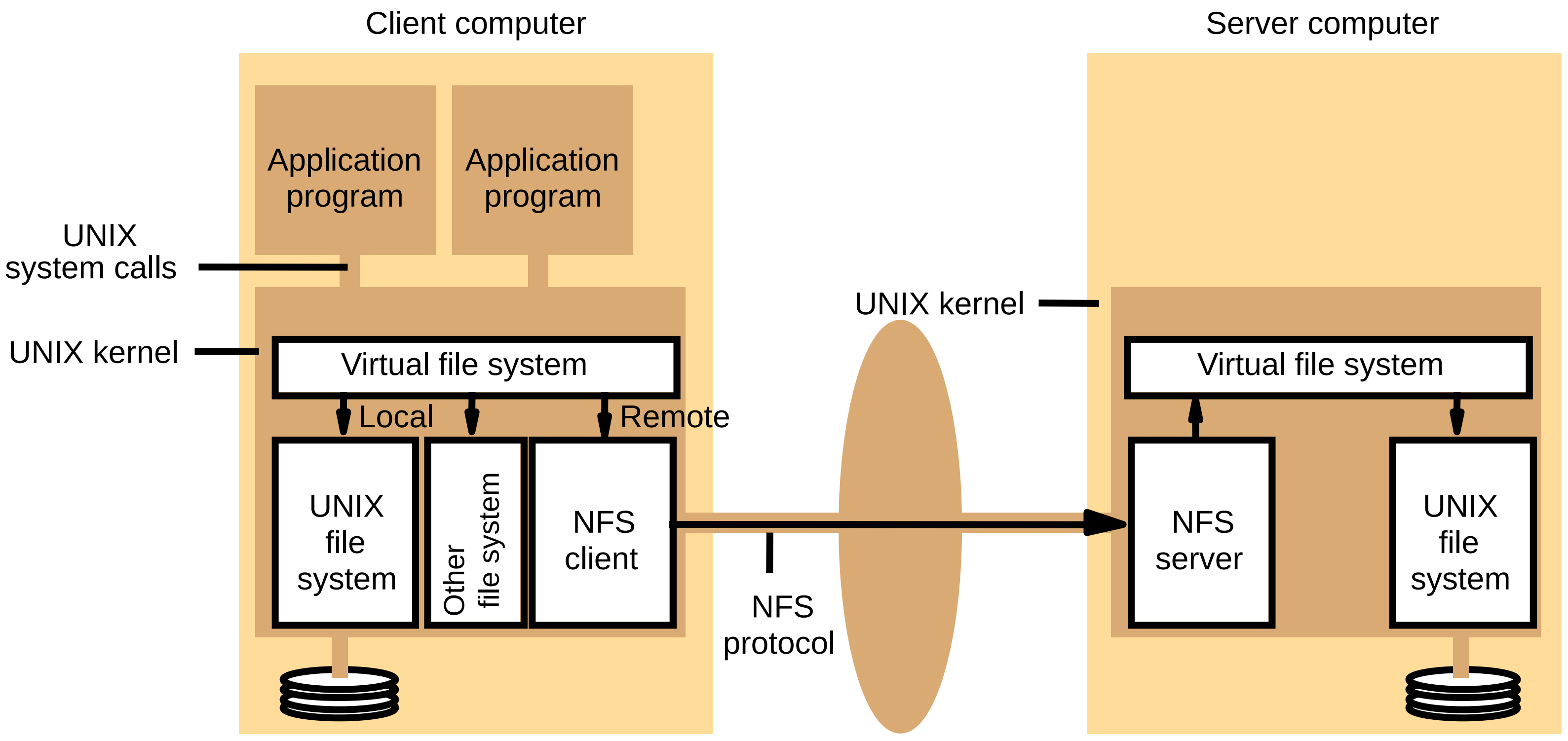
layanan directory service

- tugas utama: mapping nama ke UFID (Unique File ID)
- support untuk hierarchical file systems
- file groups

Lookup(Dir, Name) -> FileId —throws *NotFound*	Locates the text name in the directory and returns the relevant UFID. If *Name* is not in the directory, throws an exception.
AddName(Dir, Name, FileId) —throws *NameDuplicate*	If *Name* is not in the directory, adds (*Name*, *File*) to the directory and updates the file’ s attribute record. If *Name* is already in the directory: throws an exception.
UnName(Dir, Name) —throws *NotFound*	If *Name* is in the directory: the entry containing *Name* is removed from the directory. If *Name* is not in the directory: throws an exception.
GetNames(Dir, Pattern) -> NameSeq	Returns all the text names in the directory that match the regular expression *Pattern*.

NFS architecture

- NFS protocol menggunakan RPC
- VFS menyediakan transparansi



NFS server operations (simplified) – 1

lookup(dirfh, name) -> fh, attr

Returns file handle and attributes for the file *name* in the directory *dirfh*.

create(dirfh, name, attr) -> newfh, attr

Creates a new file name in directory *dirfh* with attributes *attr* and returns the new file handle and attributes.

remove(dirfh, name) status

Removes file name from directory *dirfh*.

getattr(fh) -> attr

Returns file attributes of file *fh*. (Similar to the UNIX *stat* system call.)

setattr(fh, attr) -> attr

Sets the attributes (mode, user id, group id, size, access time and modify time of a file). Setting the size to 0 truncates the file.

read(fh, offset, count) -> attr, data

Returns up to *count* bytes of data from a file starting at *offset*. Also returns the latest attributes of the file.

write(fh, offset, count, data) -> attr

Writes *count* bytes of data to a file starting at *offset*. Returns the attributes of the file after the write has taken place.

rename(dirfh, name, todirfh, toname) -> status

Changes the name of file *name* in directory *dirfh* to *toname* in directory to *todirfh*

link(newdirfh, newname, dirfh, name) -> status

Creates an entry *newname* in the directory *newdirfh* which refers to file *name* in the directory *dirfh*.

NFS server operations (simplified) – 2

symlink(newdirfh, newname, string) *-> status*	Creates an entry *newname* in the directory *newdirfh* of type symbolic link with the value *string*. The server does not interpret the *string* but makes a symbolic link file to hold it.
readlink(fh) -> string	Returns the string that is associated with the symbolic link file identified by *fh*.
mkdir(dirfh, name, attr) -> *newfh, attr*	Creates a new directory *name* with attributes *attr* and returns the new file handle and attributes.
rmdir(dirfh, name) -> status	Removes the empty directory *name* from the parent directory *dirfh*. Fails if the directory is not empty.
readdir(dirfh, cookie, count) -> *entries*	Returns up to *count* bytes of directory entries from the directory *dirfh*. Each entry contains a file name, a file handle, and an opaque pointer to the next directory entry, called a *cookie*. The *cookie* is used in subsequent *readdir* calls to start reading from the following entry. If the value of *cookie* is 0, reads from the first entry in the directory.
statfs(fh) -> fsstats	Returns file system information (such as block size, number of free blocks and so on) for the file system containing a file *fh*.

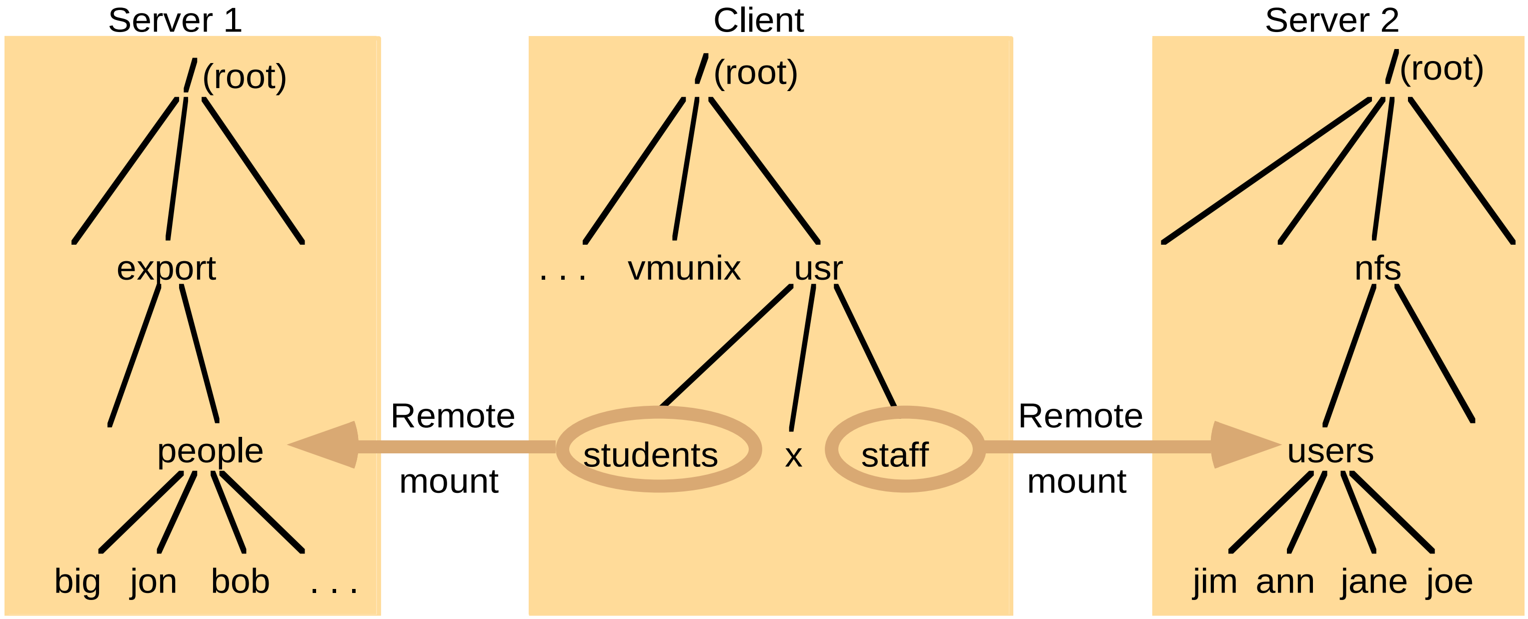
NFS

- access control & authentication
    - NFS server stateless
    - server harus mencek identitas client setiap request
    - NFS protocol mengirimkan user authentication information (e.g., user Id dan group Id) pada setiap request
    - potensi untuk pemalsuan user ID -> enkripsi dan kerberos

NFS

- mount service
- server menyediakan mount service yang berjalan pada user level
- /etc/exports berisi daftar direktori yang dapat di-mount remote
- client menggunakan mount untuk meminta mount pada remote server, dengan memberikan hostname, remote directory dan local mount point

Local and remote file systems accessible on an NFS client



Note: The file system mounted at */usr/students* in the client is actually the sub-tree located at */export/people* in Server 1; the file system mounted at */usr/staff* in the client is actually the sub-tree located at */nfs/users* in Server 2.

NFS

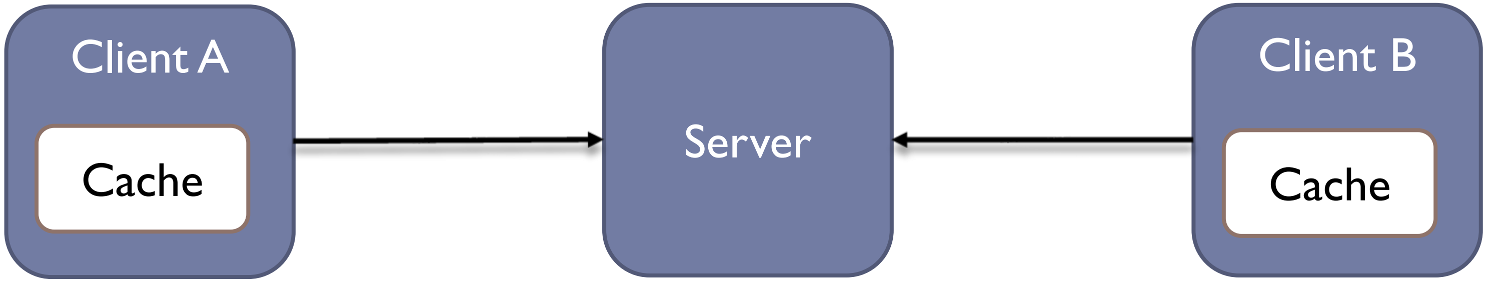
- soft mount vs hard mount:
  - hard-mount: jika terjadi kegagalan, proses akan terblok sampai request berhasil
  - soft-mount: client module akan mengembalikan kode error setelah beberapa retry. proses aplikasi pengguna harus menangani kode error tersebut.

NFS

- automounter
- autofs (diimplementasikan pada kernel) memonitor sejumlah mount point, dan akan otomatis melakukan mounting jika ada user yang mengakses direktori tersebut
- sebuah mount point dapat dilayani oleh beberapa NFS server, untuk fault tolerance

Distributed File Systems - Performance

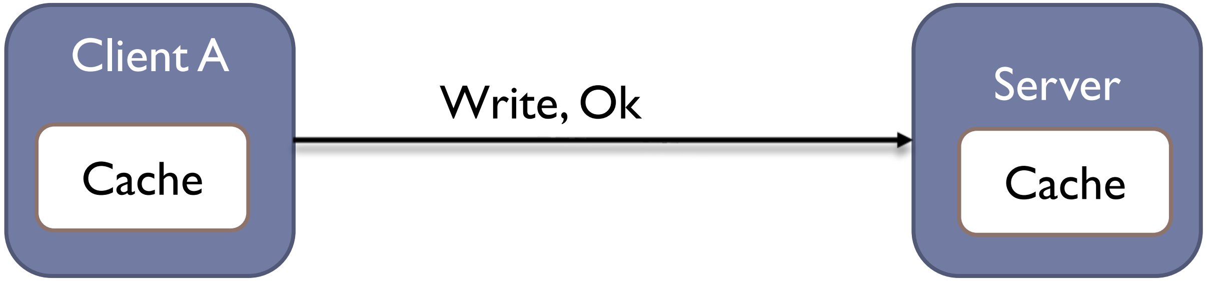
- Bagaimana meningkatkan performance pada NFS?
- Implementasi Cache: copy file disimpan di sisi client



- Problem: update file. Jika Client A menyimpan cache file x, dan client B mengupdate file x. Apa yang terjadi saat Client A membaca file x dari cache

Distributed File Systems - Performance

- Implementasi Cache di sisi server



- Di sisi server, cache mempercepat akses ke file systems, karena copy sudah tersedia di memori
- Problem: Jika server down, data akan hilang, padahal client sudah mendapat konfirmasi write sukses

- Pada NFS, server cache tidak dibolehkan

NFS

- Caching berfungsi untuk meningkatkan kinerja
- file caching: read ahead buffer dan delayed write
- server caching: delayed write bermasalah, karena client sudah mendapat konfirmasi, namun file bisa hilang jika server crash sebelum sempat menulis ke disk
  - write through: performance problem
  - menggunakan commit operation (NFS 3)
- client caching:
  - data pada client, disimpan timestamp saat cache entry divalidasi, dan data asli pada server
  - (T-Tc <t) V (Tm client = Tm server)

Sumber

- Slide buku Coulouris et.al., chapter 12, 2012