Sistem Terdistribusi

Distributed File System

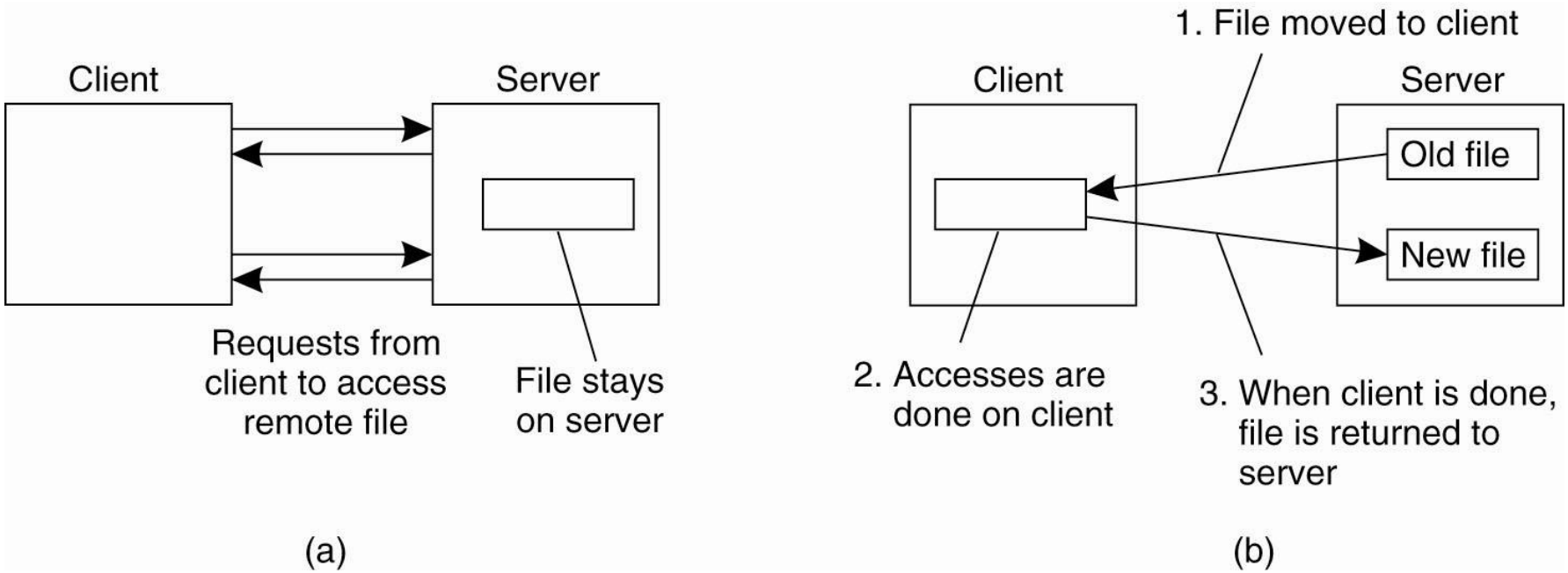
Latar Belakang

- Perlu adanya **sharing** informasi/resource
- Informasi/resource mungkin berada pada komputer lain dan dalam format lain (file system)
- Seharusnya pengguna **tidak perlu tahu** file system yang dipakai pada sistem lain
- Sehingga diperlukan **service** yang memungkinkan akses file remote seperti layaknya pengaksesan pada harddisk lokal walaupun melalui jaringan intranet/Internet

Model Arsitektur

- **Client Server**
  - Server provide files/directories
  - Client access files/directories
  - Operation: Add/Remove, Read/Write
- **Distributed file system**
  - File System shared to many distributed clients
  - Communication through shared files and storages resources

Client-Server Architectures



- (a) The remote access model.
  (b) The upload/download model.

Background

- Distributed file system (DFS) adalah sebuah sistem di mana banyak pengguna dapat berbagi berkas dan sumber daya penyimpanan.
- A DFS manages **set of distributed storage** devices
- Overall storage space managed by a DFS is composed of *different, remotely located, smaller storage spaces*
- implementation of the **classical time sharing model** of a file system

DFS Structure

- Service – software yang bisa berjalan pada satu atau banyak mesin dan meyediakan sebuah fungsi tertentu yang berfungsi untuk memprioritaskan layanan pada client
- Server – service software that running on a single machine
- Client – process that can invoke a service using a set of operations that forms its client interface

- A client interface for a file service is formed by a set of **primitive file operations** (create, delete, read, write)
- Client interface of a DFS should be **transparent**, i.e., not distinguish between local and remote files

Tantangan

- Transparency
  - **Access**: client can acess remote files as local files
  - **Location**: client can't tell the file location
  - **Migration**: file can move to other server
  - **Replication**: multiple copies of file exists
  - **Concurrency**: multiple client access
  - **Failure**: client & program have to run even when server fail
- Flexibility
  - support multiple FS types
  - server can be added/removed
- Consistency
  - File tetap konsisten diakses dari manapun
- Security
  - User access

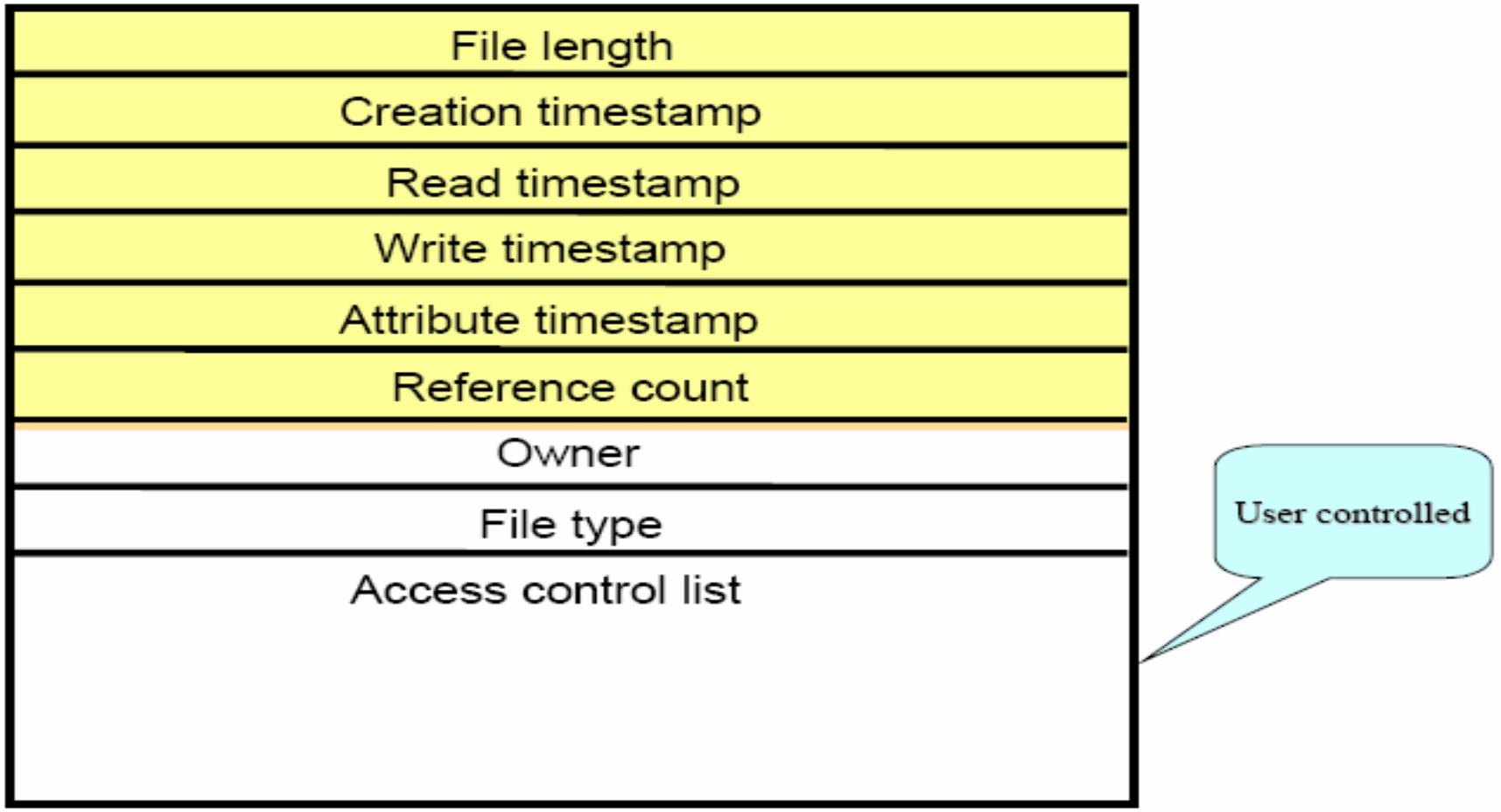
Tantangan (2)

- Fault tolerance
  - Jika terjadi kegagalan dapat diatasi dengan cepat
- Performance
  - Diatasi cara load balancing
- Naming
  - mapping between logical and physical objects.
- Scalability
  - File and users can be grown exponentially
- Heterogeneity
  - OS, Programming language, software
- Efficiency
  - Working on network introduce latency and other overhead, be careful

File & Direktori

- Setiap file punya data dan attribute (meta data)
  - Nama file
  - Ukuran file
  - Read/Write/Creation timestamp
  - Pemilik/ACL
- File diatur dan diorganisasi dalam direktori
- Direktori adalah sebuah file dengan jenis khusus yang melakukan mapping dari sebuah nama ke sebuah identifier file internal

Atribut file



Layanan file terdistribusi

- **Layanan Dasar**
  - Menyediakan tempat penyimpanan yang tetap untuk data dan program
  - Menyediakan operasi terhadap file (create, open, read,…)
  - Memiliki multiple remote clients
  - Mendukung file sharing pada jaringan
  - menggunakan semantic **one-copy update** umum, melalui RPC
- Perkembangan baru
  - persistent object stores (storage of objects)
  - multimedia terdistribusi

Operasi File

- Operasi terhadap files (=data + attributes)
  - create/delete
  - attribute query/modify
  - open/close
  - read/write
  - copy
  - access controls

Organisasi Storage

- directory structure (hierarchical, menggunakan pathnames)
- metadata (file management information)
  - Misalnya cylinder, track, sector
  - Hidden dari user
- file attributes
  - Read only, writeable, hidden
- informasi struktur directory
  - Root, child, leaf

Storage systems and their properties

	Sharing	*Persis-tence*	*Distributed cache/replicas*	*Consistency maintenance*	*Example*
Main memory	×	×	×	1	RAM
File system	×	✓	×	1	UNIX file system
Distributed file system	✓	✓	✓	✓	Sun NFS
Web	✓	✓	✓	×	Web server
Distributed shared memory	✓	×	✓	✓	Ivy
Remote objects (RMI/ORB)	✓	×	×	1	CORBA
Persistent object store	✓	✓	×	1	CORBA Persistent Object Service
Peer-to-peer storage system	✓	✓	✓	2	OceanStore

Types of consistency:
1: strict one-copy. 3: slightly weaker guarantees. 2: considerably weaker guarantees.

UNIX file system operations

filedes = open(name, mode)	Opens an existing file with the given *name*.
filedes = creat(name, mode)	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)	Transfers *n* bytes from the file referenced by *filedes* to *buffer*.
count = write(filedes, buffer, n)	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*.

Bentuk layanan file

- **Stateful**
  - server **menyimpan** informasi tentang file yang diopen, posisi sekarang (current position) dari pembacaan file dan file locks
  - open (dibuka) sebelum diakses dan kemudian setelah selesai ditutup
  - performa yang baik, dimungkinkan untuk **read-ahead**
  - Tidak memerlukan semua informasi dikirim (*shorter message*) ke client
  - Jika server failure - kehilangan state
  - Jika client failure – state server masih ada
  - Mampu menyediakan mekanisme file locks

Opsi Layanan File

- Stateless
  - server **tidak menyimpan** state informasi
  - file operations harus mengandung semua yang diperlukan (longer message) untuk dikirim ke client
  - perancangan file server yang lebih **simpel**
  - locking membutuhkan **extra lock server** untuk mempertahankan state
  - State tidak perlu dimaintenance sehingga recovery cepat
    - unnoticable

File sharing

- Multiple clients berbagi (share) file yang sama untuk akses **read/write** secara concurrent
- **One-copy update** semantics
  - setiap read melihat dampak semua writes sebelumnya
  - suatu proses write akan segera visible ke client yang sudah siap membuka file untuk reading
- Problems:
  - **Dibutuhkan caching**: untuk memelihara konsistensi antara beberapa copies file
  - **Locking:** akses serialisasi dengan menggunakan file locks (sehingga mempengaruhi performance)
  - Ada pengaruh antara consistency dan performance

Flat file service operations

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 not shaded in Figure 8.3).

Directory service operations

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 (Network file system)

- Protokol yang dikembangkan oleh **Sun Microsystem (1984)**
- Dipakai secara luas pada industri dan pendidikan pada tahun **1985**
- Sekarang menjadi public source
  - RFC 1094, http://tools.ietf.org/html/rfc1094
  - RFC 1813, http://tools.ietf.org/html/rfc1813
  - RFC 3530, http://tools.ietf.org/html/rfc3530

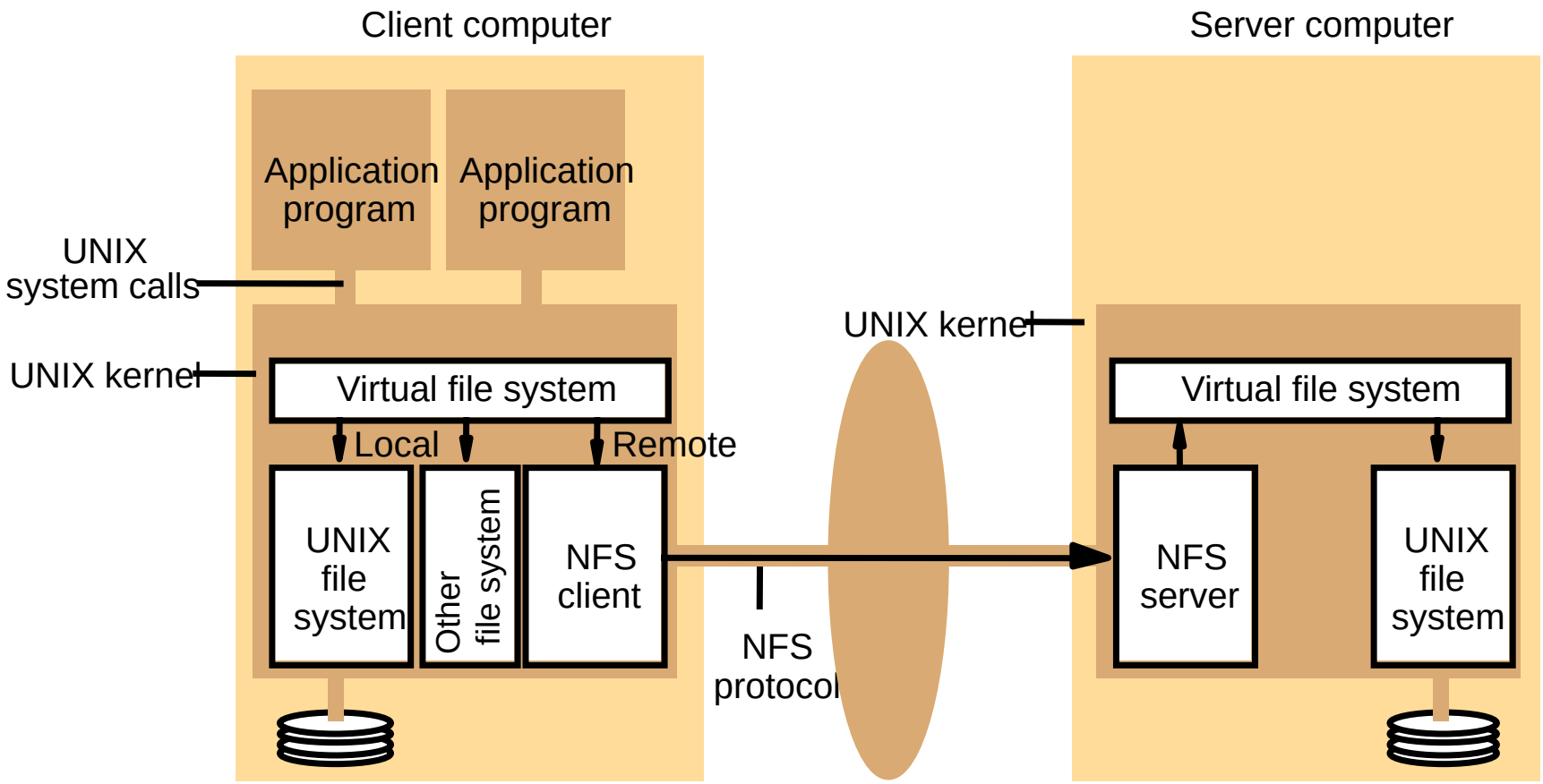
NFS

- Diimplementasikan menggunakan **Open Network Computing Remote Procedure Call**
- support for **64-bit** file sizes and offsets, to handle files larger than **2** gigabytes (GB);
- support for **asynchronous** writes on the server, to improve write performance;

NFS

- **Server side**
  - NFS protocol is **independent** from file system
  - NFS Server run as a **daemon**
  - ***/etc/exports*** (Linux) specify what directory are exported to whom under which policy
  - Transparent caching (read ahead)
- **Client side**
  - Mounting (explicit/auto)
  - Support diskless workstations (thin clients)
  - ***/etc/fstab*** (Linux)

NFS architecture



Operasi-operasi NFS

Operation	v3	v4	Description
Create	Yes	No	Create a regular file
Create	No	Yes	Create a nonregular file
Link	Yes	Yes	Create a hard link to a file
Symlink	Yes	No	Create a symbolic link to a file
Mkdir	Yes	No	Create a subdirectory in a given directory
Mknod	Yes	No	Create a special file
Rename	Yes	Yes	Change the name of a file
Remove	Yes	Yes	Remove a file from a file system
Rmdir	Yes	No	Remove an empty subdirectory from a directory
Open	No	Yes	Open a file
Close	No	Yes	Close a file
Lookup	Yes	Yes	Look up a file by means of a file name
Readdir	Yes	Yes	Read the entries in a directory
Readlink	Yes	Yes	Read the path name stored in a symbolic link
Getattr	Yes	Yes	Get the attribute values for a file
Setattr	Yes	Yes	Set one or more attribute values for a file
Read	Yes	Yes	Read the data contained in a file
Write	Yes	Yes	Write data to a file

NFS Daemon

- **nfsd** : NFS daemon melayani permintaan dari NFS client.
- **mountd** :NFS daemon yang membawa permintaan mounting yang telah melewati nfsd.
- **portmap** : portmapper daemon yang membolehkan NFS client untuk menemukan port mana yang digunakan oleh NFS Server.

NFS di Linux

- Linux support **NFS Client/Server**
- Require **portmap** to handle RPC connection
  - server that converts RPC program numbers into DARPA protocol port numbers
- Configuration file on server: ***/etc/exports***
  - /home -alldirs 10.0.0.2 10.0.0.3 10.0.0.4
  - /a -maproot=root antonie.com coba.com
- Client only mount with option **nfs** on ***/etc/fstab***

Cara-cara

- **Cara jalankan NFS server :**
  - #portmap
  - #nfsd
  - #mountd -r
- **Cara jalankan NFS client :**
  - #nfsiod
- Mounting di server (auto boot):
  - #mount server:/home /mnt nfs rw 0 0

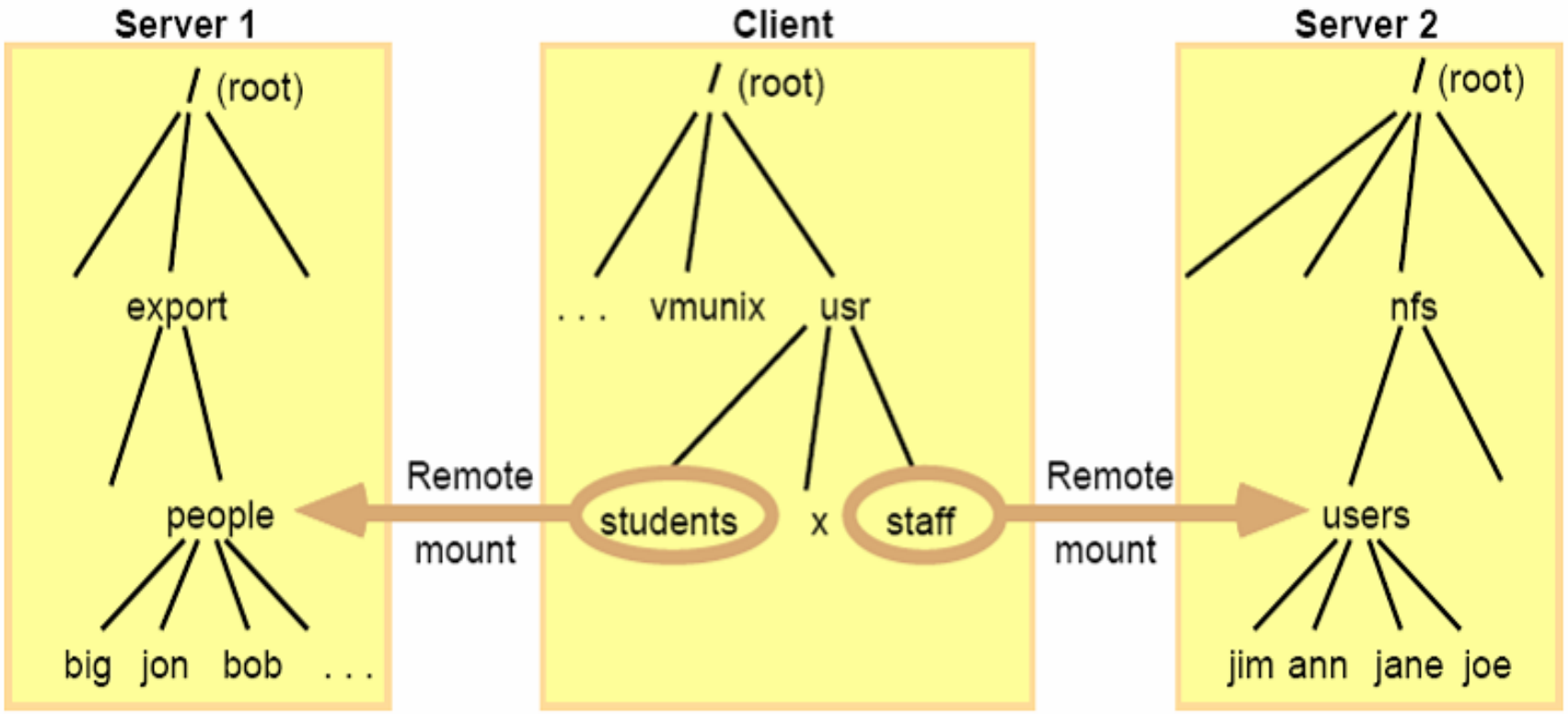
Keuntungan & Kerugian NFS

- Keuntungan:
  - Penggunaan bersama resources:
    - CDROM, Flashdisk, Harddisk
  - Disk yang dibutuhkan lebih sedikit
- Kerugian:
  - Terjadi kelambatan akses ketika terjadi konkurensi akses
  - Protocol is ***stateless***
    - Each procedure call contains *all the information necessary to complete the call*
    - Server maintains no “between call” information

Sun NFS

- Implementasi NFS oleh **Sun Microsystem**
- Sun RPC (Sun Remote Procedure Call)
- NFS Client **'call'** fungsi pada NFS Server
- Spesifikasi NFS mendefinisikan remote interface yang bisa dipakai oleh client
- Pengiriman data bisa menggunakan TCP/ UDP

Remote mount



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.

Caching

- Akses data yang terus menerus bisa membebani server dan berpengaruh pada performa
- Solusi: **Caching** (menyimpan sebagian data pada **RAM**)
- Caching sebaiknya dilakukan pada **server/client**
- Server Caching menyebabkan:
  - Read ahead
  - Delayed write

Caching

- **Read-Ahead**
  - Asumsi file dibaca secara sequential
  - Kernel membaca blok berikutnya dari file sebelum aplikasi memintanya
- Bisa menghasilkan I/O yg useless dan memakan resource memory jika ternyata tidak sesuai yang diharapkan
- **Delayed Write**
  - Perubahan akan disimpan pada disk jika buffer hendak dipakai oleh request lain
  - Pada UNIX, operasi sync akan dilakukan per 30s

Mekanisme Caching pada Server

- **Write-through**
  - Data disimpan pada **memory** cache server
    dan ditulis pada disk saat *mengirimkan reply ke client*
  - Client yakin bahwa data sudah tersimpan permanen ketika menerima jawaban dari server
- **Write-commit**
  - Penyimpanan pada disk dilakukan jika
    server menerima perintah **commit**
  - Banyak dipakai oleh NFS Client standar
    - Client mengirimkan perintah commit

Client Caching

- **Client**: Melakukan cache terhadap operasi read, write, getattr, lookup, dan readdir
- Bisa menimbulkan masalah **konsistensi** karena seringkali data sudah diupdate oleh proses lain
- **Solusi**: client **poll** server secara interval apakah data masih up-to-date

Cache Location – Disk vs. Main Memory

- Advantages of **disk caches**
  - More **reliable**
  - Cached data **kept on disk** are still there during recovery and don't need to be fetched again
- Advantages of **main-memory caches**:
  - Permit workstations to be **diskless**
  - Data can be accessed more **quickly**
  - Performance **speedup** in bigger memories
  - Server caches (used to speed up disk I/O) are in main memory regardless of where user caches are located;
    - using main-memory caches on the user machine permits a **single caching** mechanism for servers and client

Andrew File System

- Dikembangkan oleh **Carnegie Mellon University**.
- Mendukung sharing informasi untuk **skala besar** (100 – 10000+ pengguna).
- Asumsi penggunaan file pada AFS :
  - Kebanyakan dari file-file merupakan file yang kecil.
  - Lebih sering membaca daripada menulis file.
  - Kebanyakan file dibaca/tulis oleh satu pengguna.
- AFS menggunakan file serving keseluruhan berada di server dan keseluruhan file caching berada di client.

AFS

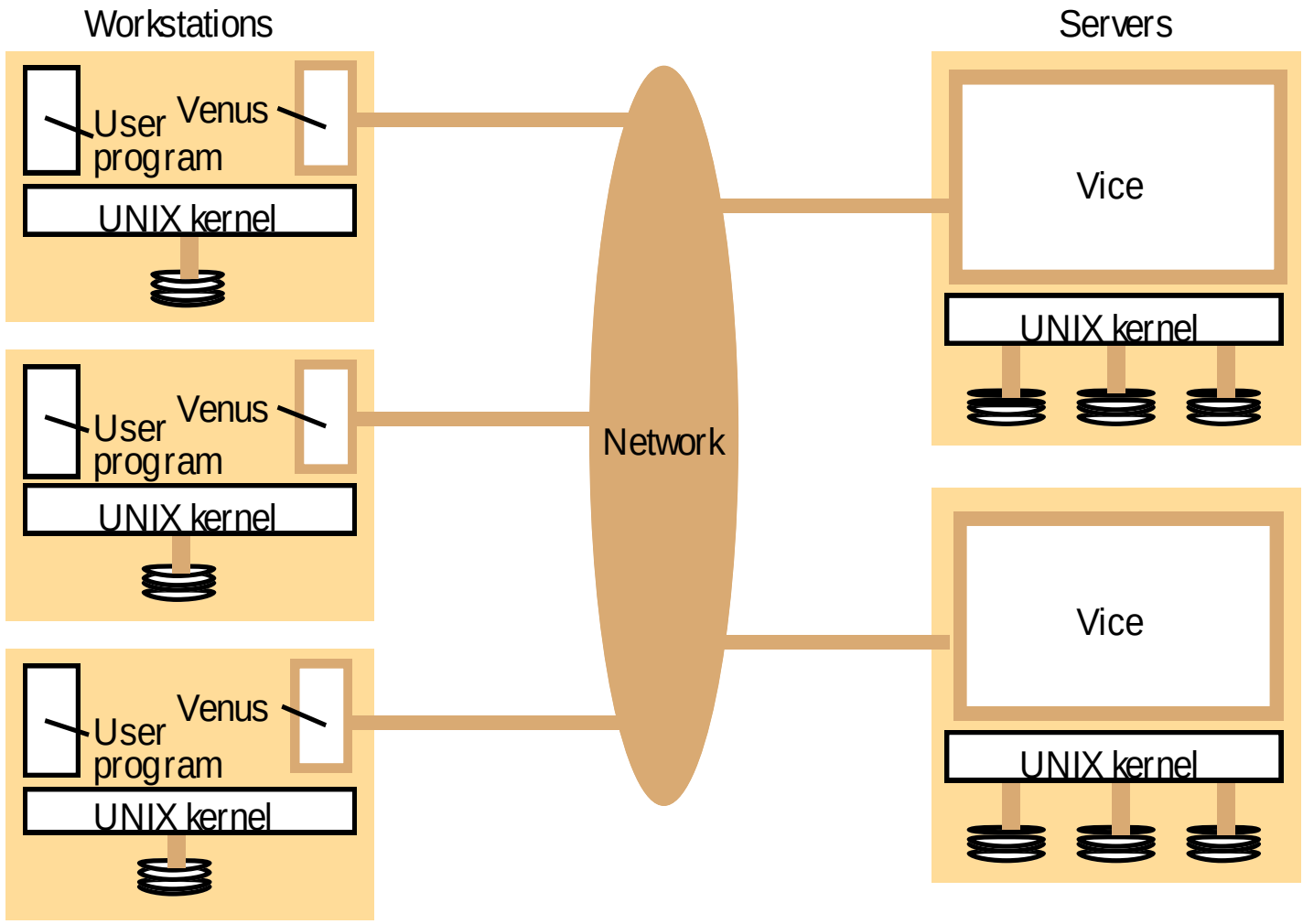
- AFFS uses **a set of trusted servers** to present a homogeneous, location-transparent file name space to all the client workstations
- AFS uses **Kerberos** for authentication, and implements **access control lists** on directories for users and groups
- Each client caches files on the local file system
  - Speedup access

Tujuan AFS

Design goals of AFS (Andrew File System)

- Maximum performance
- Ability to handle large number of users
- Scalability
- Reliability to ensure maximum uptime and availability
- To ensure computers are available to handle queries

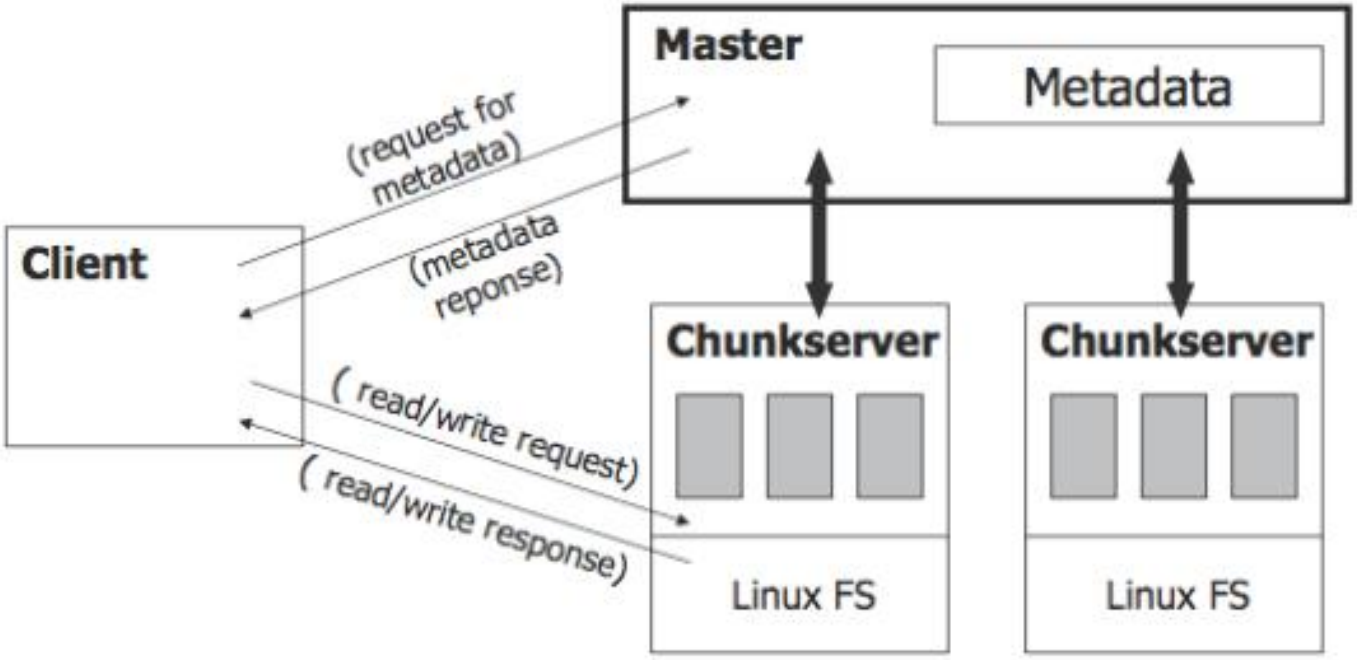
Distribution of processes in the Andrew File System



Google file system

- Made by Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung (2003)
  - http://labs.google.com/papers/gfs.html
- Successor of **BigFiles**, created by Google Founders (Larry Page and Sergey Brin)
- Idea: “Store data reliably even in the presence of unreliable machines"
- Optimized to run on Google clusters which consists of thousands of nodes per cluster
- Ensure availability by replicating files at least on three different computers in a given server cluster

Google File System Architecture



- A master process maintains the metadata
- A lower layer (i.e. a set of chunkservers) stores the data in unit called *chunks*

Chunk

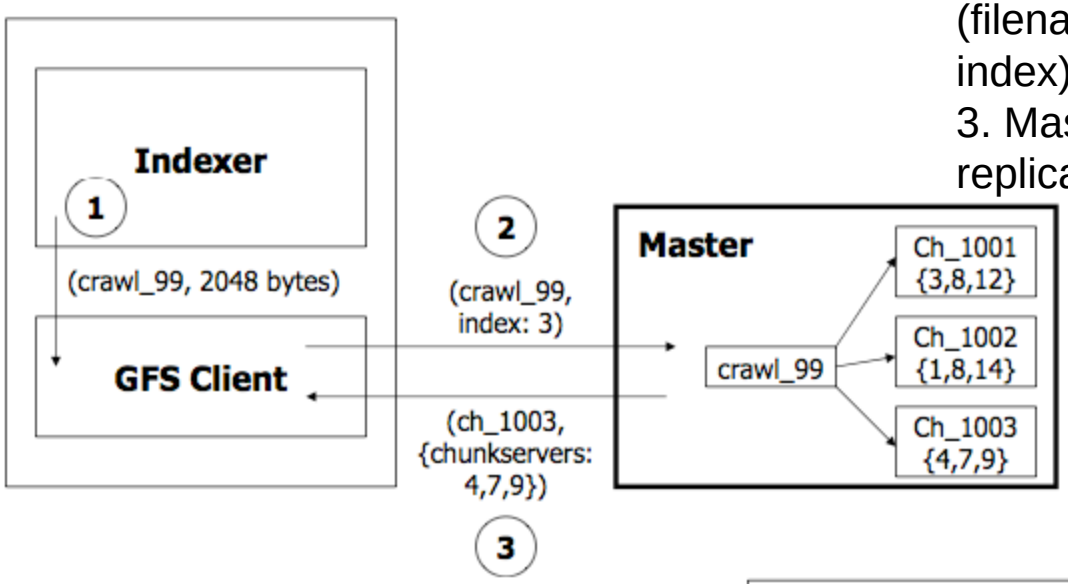
- Similar to block, much larger than typical file system block size
- Size: 64 MB!
- Chunk replicas across multiple chunkservers
- Stored in chunkserver as file
- Why so big?
  - Reduces client's need to contact with the master
  - On a large chunk a client can perform many operations

Master

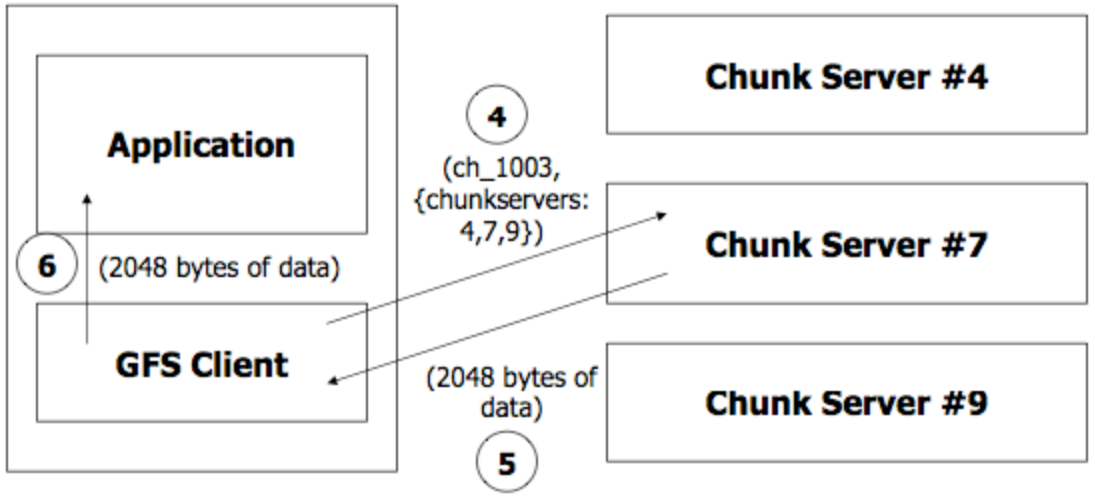
- Stores all metadata
  - File and chunk namespace
  - File to chunk mappings
  - Chunk location information
  - Access control information
  - Chunk version numbers
- Master and chunkserver communicate regularly to obtain state
- Master sends instructions to chunkserver

GFS Read algorithm

1. Application originates the read request
2. GFS client translates the request form (filename, byte range) -> (filename, chunk index), and sends it to master
3. Master responds with chunk handle and replica locations

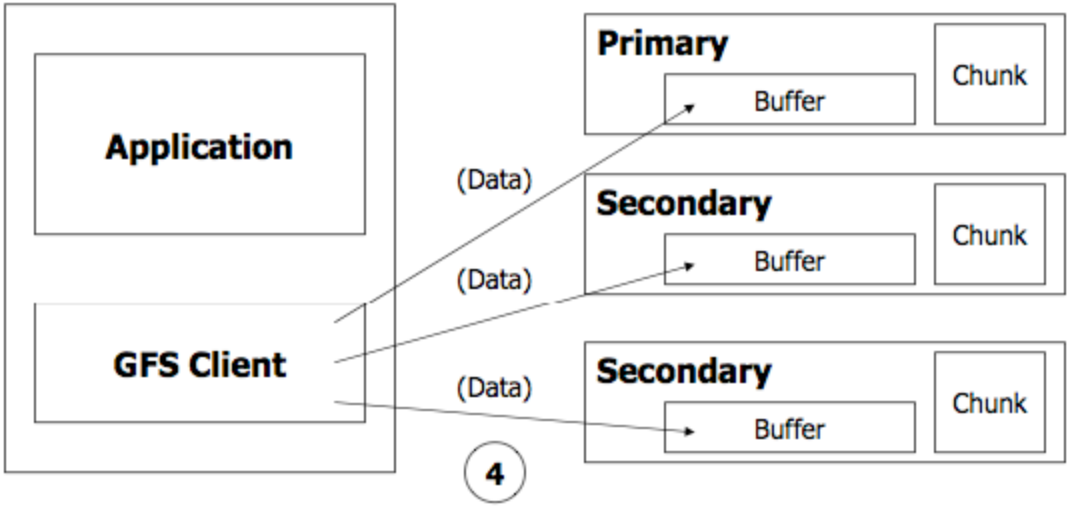


4. Client picks a location and sends the (chunk handle, byte range) request to the location
5. Chunk server sends requested data to the client
6. Client forwards the data to the application



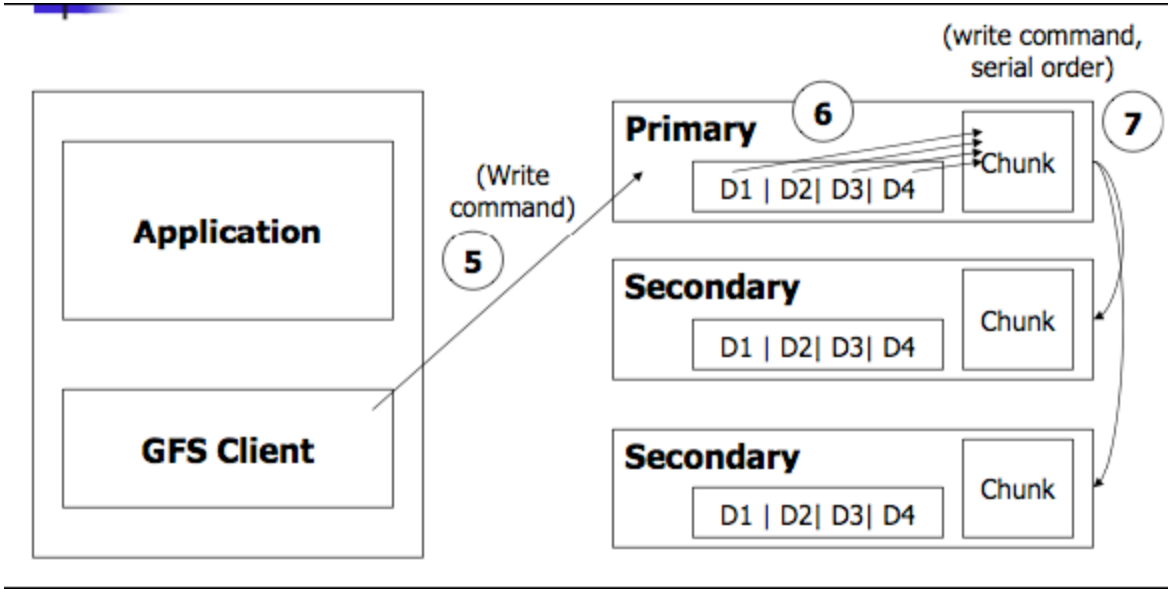
GFS write algorithm

- 1. Application originates the read request
- 2. GFS client translates request from (filename, data) -> (filename, chunk index), and sends it to master
- 3. Master responds with chunk handle and (primary+secondary) replica locations
- 4. Client pushes write data to all locations. Store in buffer



Write algoritm

- 5. Client sends write command to primary
- 6. Primary determines serial order for data instances stored in its buffer and writes the instances in that order to the chunk
- 7. Primary send the serial order to the secondary and tell them to perform the write



DFS lainnya

- Amazon S3
- Apple Filing Protocol
- File Access Listener
- Microsoft Office Groove
- Self Certifying File System
- Coda
- WebDFS

NEXT

- Naming Services