File system distribuiti

- Introduzione
- Architettura di un File System distribuito (FSD) tradizionale:
  - *File Service*
  - *Directory service*
- FSD tradizionali
  - Sun Network File System (NFS)
  - Andrew File System (AFS)
  - CODA
  - Plan9
  - XFS
  - SFS
- Nuovi FSD per cluster

  HDFS

  Ceph

  GlusterFS

Introduzione

- Il termine *Permanent Storage* è una astrazione fondamentale in informatica, ed indica un insieme di "oggetti"...
  - dotati di nome…
  - ...i quali vengono creati esplicitamente…
  - ...ed esistono fino a che non sono esplicitamente distrutti

- I *File System* sono una possibile implementazione di questo livello di astrazione; le *Basi di dati* sono un'altra possibilità
- L'implementazione di un File System dipende dal livello di sofisticazione dei requisiti che si richiedono.

Livelli dei requisiti

- Livello 1: Sistema **monoutente**, **monoprogrammato**
  - Esempio: MS-DOS
  - Un unico utente esegue un singolo processo alla volta
  - Requisiti del File System:
    - Definire una struttura dei nomi consistente
    - Definire una interfaccia per le applicazioni
    - Mappare la struttura dei nomi nel dispositivo fisico di memorizzazione
    - Garantire l'integrità dei dati
- Livello 2: Sistema **monoutente**, **multiprogrammato**
  - Esempio: OS/2
  - Un unico utente esegue più processi alla volta
  - Requisiti del File System (oltre a quelli di Livello 1):
    - Fornire un controllo della concorrenza, per garantire che processi diversi non interferiscano tra di loro.

Livelli dei requisiti (cont.)

- Livello 3: Sistema **multiutente**, **multiprogrammato**
  - Esempio: UNIX
  - Utenti diversi eseguono processi diversi sulla stessa macchina
  - Requisiti del File System (oltre a quelli di Livello 2)
    - Fornire dei meccanismi per garantire la sicurezza (proteggere i dati da accessi da parte di utenti non autorizzati)
- Livello 4: Sistema **distribuito**
  - Esempio: Cluster di Workstation
  - È composto da insiemi, anche eterogenei, di unità collegate in rete
  - Requisiti del File System (oltre a quelli di Livello 3)
    - Fornire una interfaccia per gestiore l'allocazione dei file sui nodi della rete
    - Garantire la disponibilità del servizio di file sharing anche in presenza di reti di comunicazione non affidabili

File system distribuiti

- Il servizio dei file è la specifica di ciò che il file system offre ai propri clienti: descrive le primitive disponibili, i loro parametri e le loro azioni.
- Il file server è un processo che gira su una qualche macchina e aiuta ad implementare il servizio dei file.
- In alcuni sistemi distribuiti ci sono solo le operazioni di CREATE e READ. Una volta che è stato creato, non può essere modificato (immutabile).
- Per quanto riguarda la protezione nei sistemi distribuiti si usa la stessa tecnica dei sistemi operativi uniprocessori: capability e liste di controllo degli accessi.
  - Capability è un biglietto dato a ciascun utente che per ogni oggetto specifica l'operazione permessa (per es. lettura ma non scrittura, ecc.)
  - Le liste di controllo degli accessi associano a ciascun file una lista di utenti che possono accedere al file e quali tipi di accesso sono permessi (lo schema di Unix con i bit di controllo r/w/x separati per l'utente, il gruppo e tutti è un esempio semplificato di liste di controllo).

File system distribuiti

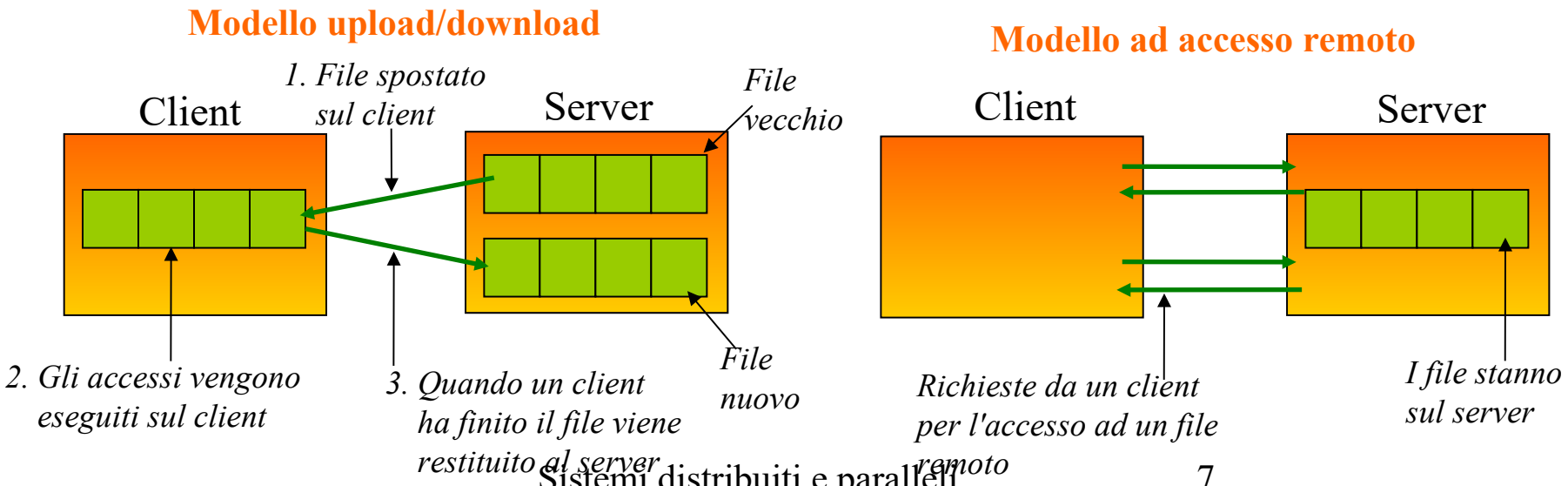
- I servizi dei file si possono dividere in due tipi a seconda che mettano a disposizione uno dei modelli seguenti:
  - modello upload/download
  - modello di accesso remoto.
- Nel modello upload/download il servizio dei file mette a disposizione due operazioni principali: lettura da un file e scrittura su un file.
  - La lettura trasferisce un intero file da uno dei file server al client che ne ha fatto richiesta
  - La scrittura trasferisce un intero file dal client al server.

In tale modello i file vengono spostati per intero in entrambe le direzioni e possono essere memorizzati o in memoria o nel disco locale. Il vantaggio di tale modello è la sua semplicità concettuale. Ogni file modificato o creato deve essere riscritto quando il programma termina senza appesantimenti di interfaccia. Lo svantaggio principale sta nell'occupazione eccessiva di spazi di memoria e nel pesante trasferimento in rete dell'intero file.

File system distribuiti

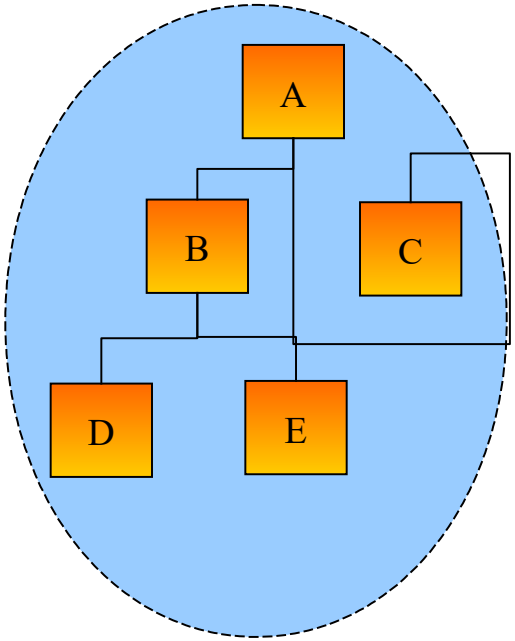
- Nel modello ad accesso remoto il servizio dei file mette a disposizione un gran numero di operazioni per l'apertura e chiusura dei file, per la lettura e la scrittura di parti di file, per muoversi all'interno dei file (LSEEK), per esaminarne gli attributi, ecc.

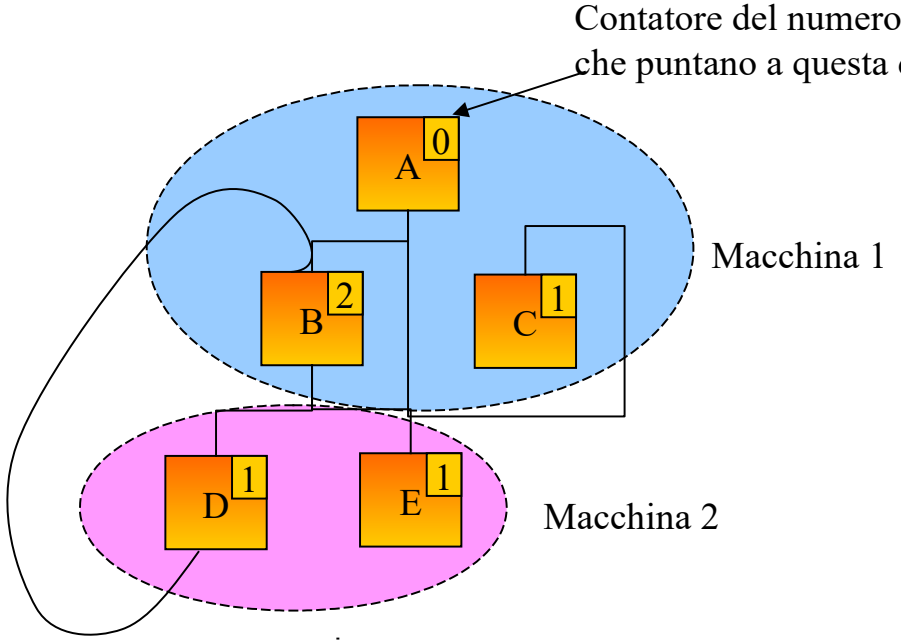
  In tale modello il servizio dei file gira sui server e non sui client, con il vantaggio di non richiedere grandi spazi sul client e piccole porzioni da trasferire.



Interfaccia di servizio delle directory

- Un file system gerarchico può essere rappresentato da un albero di directory al cui interno possono esserci file o altre directory.
- In alcuni sistemi è possibile creare link o puntatori a directory i quali possono essere messi in qualunque directory, rendendo possibile costruire non solo alberi, ma anche grafi di directory, che risultano essere molto più potenti in un sistema distribuito.





Tecniche frequentemente usate

- Per migliorare le prestazioni dei File System distribuiti, molte implementazioni fanno uso di una serie di tecniche ormai collaudate:
  - Caching dei dati
  - Trasferimenti dei dati in blocchi (*Bulk Transfers*)
  - *Hints*
  - Crittografia

Caching

- Caching dei file nel client
  - Chiaramente è l'idea che maggiormente contribuisce a migliorare le prestazioni nell'accesso ai dati in un FS distribuito
  - La cache in questo contesto sfrutta sia la località *temporale*, che la località *spaziale* nell'accesso ai file
    - **Località temporale**: lo stesso file verrà acceduto più volte a breve distanza di tempo
    - **Località spaziale**: blocchi vicini alla posizione corrente di un file verranno acceduti con maggiore probabilità rispetto a blocchi lontani
- Ci sono molte domande alle quali gli sviluppatori del FS devono rispondere, a proposito delle cache:
  - La cache deve operare su blocchi dei file oppure su interi file? Nel primo caso, quale deve essere la dimensione del blocco?
  - Dove risiede la cache? In memoria centrale o su disco?
  - Fino a che livello è possibile garantire la coerenza dei dati nelle varie cache?

Bulk Transfer

- L'overhead necessario per ogni trasmissione o ricezione di dati su una rete locale normalmente è la principale fonte di latenza.
- Conviene raggruppare, quando possibile, la trasmissione di tanti pacchetti "piccoli" con un unico trasferimento (*bulk transfer*)
  - Questo solitamente riduce il numero di *acknowledgement* che devono viaggiare sulla rete, oltre a ridurre il numero di context-switch necessari per trasferire ogni singolo pacchetto

Hints

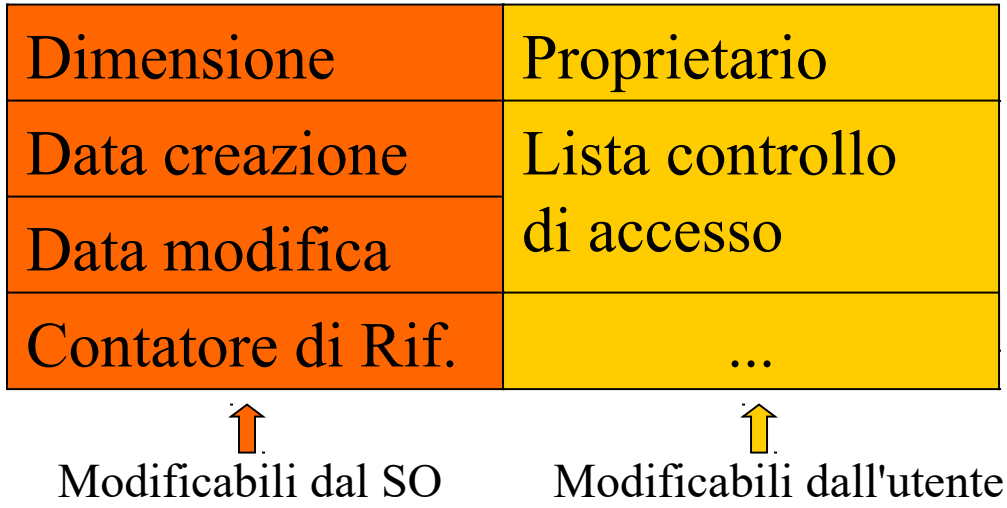
- Un *hint* (suggerimento) è una informazione che:
  - *Se corretta*, può migliorare le prestazioni del sistema;
  - Se *non corretta*, non ha conseguenze semanticamente negative
- Solo informazioni che sono "auto-validanti" in seguito al loro uso possono essere classificate come *hints*.
- La maggior parte degli *hint* riguardano informazioni sulla locazione delle informazioni nei sistemi distribuiti
  - Ad esempio, l'associazione tra il percorso di una directory e l'indirizzo IP del server che mantiene i file;
    - Se un hint del genere non è valido, quando accedo al server mi rendo subito conto che non contiene i dati che cerco.
    - "auto-validazione" = Riuscire a capire quando un hint è invalido, in seguito al suo utilizzo

Crittografia

- L'uso della crittografia in questo contesto è evidente:
  - Consentire solo agli utenti autorizzati di accedere alle informazioni
  - Proteggere il contenuto dei file mentre vengono trasferiti dal client al server
- Due approcci alla crittografia:
  - AFS usa un meccanismo a chiave privata, che si basa su un A*uthentication Server* che mantiene la lista delle chiavi, e che deve essere fisicamente sicuro.
    - Lo svantaggio è che la sicurezza di tutto il sistema si basa sulla sicurezza fisica del server che mantiene le chiavi
  - Versioni recenti di NFS utilizzano un meccanismo di crittografia a chiave pubblica, memorizzano le chiavi di autenticazione sul server, codificate con la password degli utenti.
    - Lo svantaggio è che i sistemi di crittografia a chiave pubblica sono sostanzialmente più lenti di quelli a chiave privata.

Caratteristiche dei File System

- Il File System è responsabile dell'organizzazione, memorizzazione, recupero, gestione dei nomi, condivisione e protezione dei file.
- Ciascun file contiene dati e attributi (detti anche *metadati*)
  - I dati sono le informazioni memorizzate dall'utente, e sono accessibili mediante primitive di lettura e scrittura
  - I metadati sono solitamente raggruppati in un record di controllo, e mantengono informazioni "amministrative"



Operazioni UNIX sui file

filedes=open(name,mode)
pos=lseek(filedes,offset,whence)

Apre un file esistente

filedes=creat(name,mode)

Crea un nuovo file

status=close(filedes)

Chiude un file aperto

count=read(filedes,buffer,n)

Legge *n* byte dal file *filedes*, a partire dalla posizione corrente

count=write(filedes,buffer,n)

Scrive *n* byte sul file *filedes*

Sposta il puntatore di lettura/scrittura di una quantità *offset* (relativa o assoluta, dipende dal parametro *whence*)

status=unlink(name)

Rimuove una occorrenza del file *name* dalla directory

status=link(name1,name2)

Aggiunge il nuovo nome *name2* al file *name1*

status=stat(name,buffer)

Legge gli attributi del file *name*

Nota sull'interfaccia UNIX al File System

- Le funzioni appena viste non sono di solito utilizzate direttamente, ma attraverso funzioni della libreria Standard C per l'Input/Output.
- Le operazioni UNIX si basano sull'assunzione che il Sistema Operativo mantenga delle informazioni sullo stato dei file durante il loro uso
  - Il Sistema Operativo mantiene la lista dei file aperti, con associato il puntatore di lettura/scrittura per ciascun file.

Requisiti dei File System Distribuiti

- **Trasparenza**
  - *Access Transparency*
    - Gli utenti devono poter ignorare se stanno accedendo ad un file locale o uno remoto. L'interfaccia ai diversi file system deve essere omogenea
  - *Location Transparency*
    - Gli utenti devono disporre di uno spazio dei nomi uniforme. File e directory remoti devono poter essere rilocati su altri nodi senza dover cambiare il loro percorso
  - *Mobility Transparency*
    - I programmi devono funzionare inalterati anche se i file sono spostati da un server ad uno diverso.
  - *Performance Transparency*
    - Gli utenti devono continuare a ricevere prestazioni soddisfacenti anche se il carico sul file system varia in un intervallo specificato
  - *Scaling Transparency*
    - Le prestazioni del file system devono poter essere aumentate in modo incrementale per far fronte ad aumenti del carico di lavoro.

Requisiti (cont.)

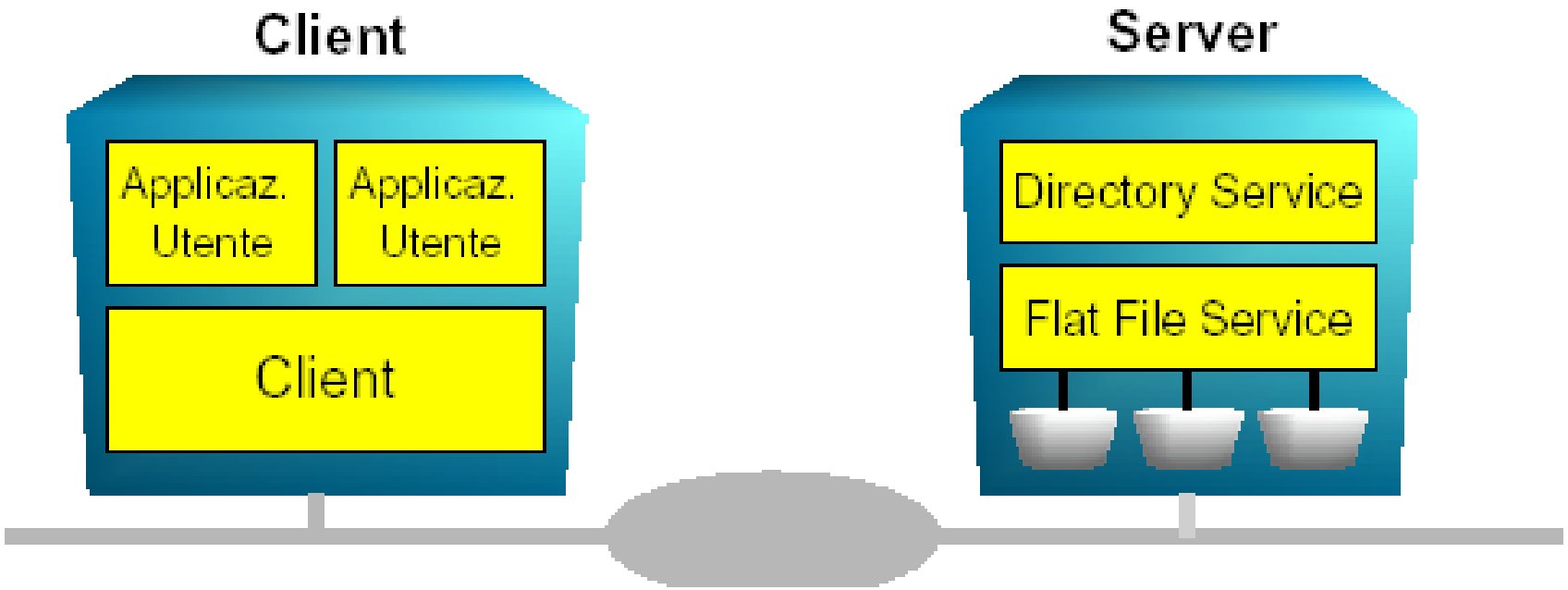
- **Operazioni di aggiornamento concorrenti**
  - Utenti diversi devono poter accedere concorrentemente agli stessi dati senza interferire tra di loro
- **Replica dei file**
  - In un file system che supporta questa funzionalità, lo stesso file può essere replicato in più copie in nodi diversi. Ciò consente di distribuire equamente il carico di lavoro tra più server nel caso in cui molti utenti accedano agli stessi file.
  - In più, questo può migliorare la tolleranza ai guasti del file system; nel caso in cui il server che contiene una replica si guasti, le altre repliche continuano ad essere disponibili.
- **Supporto di hardware e Sistemi Operativi eterogenei**
  - I file system distribuiti consentendo a utenti di condividere file tra piattaforme hardware/software diverse.
- **Consistenza**
  - Se i file sono replicati o mantenuti in cache in siti diversi, la semantica di aggiornamento dei dati deve garantire che il risultato delle operazioni sia lo stesso che nel caso in cui esista una sola copia dei dati.

Requisiti (cont.)

- **Tolleranza ai guasti**
  - Qui si intende che il protocollo di accesso ai dati deve permettere una moderata tolleranza agli errori, sia da parte del server che dei clienti:
    - La semantica di accesso ai dati deve essere tale per cui le operazioni siano *idempotenti*: la stessa operazione eseguita due o più volte deve essere equivalente ad una unica applicazione dell'operazione;
    - Il server dovrebbe essere di tipo *stateless*, in modo che il servizio possa essere ripristinato dopo un fallimento senza bisogno di ricaricare lo stato salvato in precedenza.
- **Sicurezza**
  - Chiaramente il file system, in combinazione col Sistema Operativo, deve garantire che l'accesso ai dati sia consentito solo agli utenti autorizzati
- **Efficienza**
  - I servizi offerti da un file system distribuito devono essere comparabili, in termini di prestazioni e generalità di utilizzo, a quelli forniti da un file system centralizzato tradizionale

Architettura di un File Service

- I casi di studio che prenderemo in esame (NFS e AFS) hanno una architettura simile, con tre componenti fondamentali:
  - Un modulo *client* su ciascuna Workstation che vuole accedere ai File System distribuiti
  - Due moduli *Directory Service* e *Flat File Service* sul server



Architettura di un File Service (cont.)

- **Flat File Service**
  - Implementa le operazioni sul contenuto dei file. Ciascun file è indicato con un identificatore univoco (UFID, *Unique File Identifier*), e il *Flat File Service* utilizza solo questi identificatori per operare sui file
- **Directory Service**
  - Effettua la mappatura tra il nome "testuale" di un file e il corrispondente UFID. Inoltre, provvede ad allocare nuove directory e a creare/spostare file all'interno di queste. Il Directory Service si basa sui servizi forniti dal Flat File Service.
- **Client**
  - Integra le funzionalità del File System locale. Fornisce all'utente la stessa interfaccia ai file remoti rispetto a quelli locali; mantiene le informazioni circa l'ubicazione sulla rete dei vari file server; migliora le prestazioni nell'accesso ai dati remoti con meccanismi di caching.

Operazioni implementate dal Flat File Service

- ***Data=Read(FileID, i,n)***
  - Legge al più *n* bytes consecutivi dal file identificato da *FileId*, a partire dalla posizione *i* .
- ***Write(FileID, i, Data)***
  - Scrive la sequenza di bayte contenuti in *Data* nel file indicato da *FileID* a partire dalla posizione *i*.
- ***Id=Create( )***
  - Crea un nuovo file vuoto, e ne restituisce l'identificatore.
- ***Delete(FileID)***
  - Cancella il file dal disco.
- ***Attr=GetAttributes(FileID)***
  - Restituisce gli attributi del file.
- ***SetAttributes(FileID, Attr)***
  - Imposta gli attributi del file (si applica ai soli attributi modificabili dall'utente).

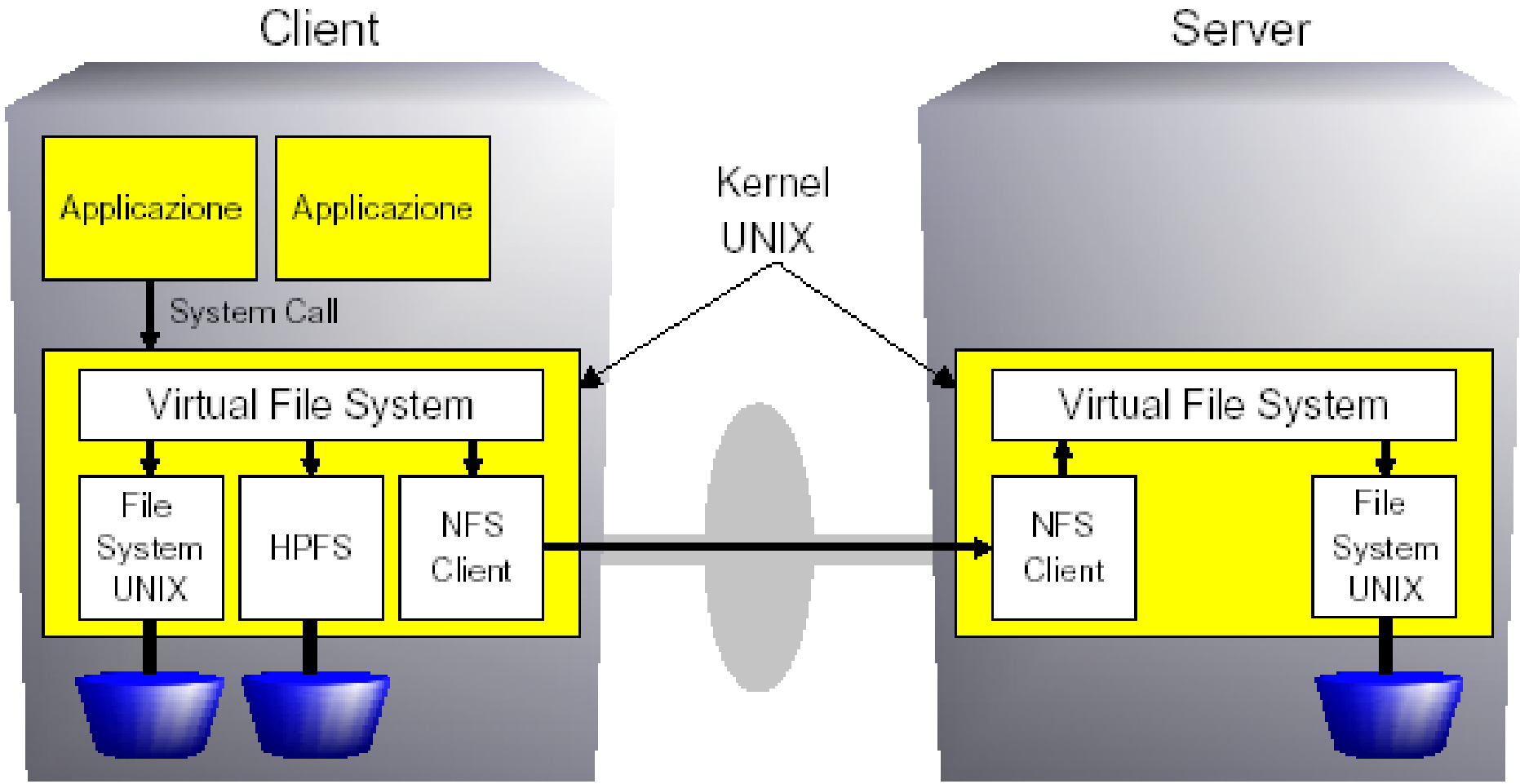
Nota sulle operazioni del File Service

- Si può constatare come le operazioni messe a disposizione dal File Service siano funzionalmente equivalenti alle primitive UNIX per l'accesso ai file.
  - Il File Service descritto non richiede l'apertura e la chiusura di un file.
- Ci sono però differenze con le primitive UNIX:
  - Le operazioni (tranne *Create( )*) sono **Ripetibili** (*Idempotenti*)
    - Cioè se si ripete la stessa operazione più volte di seguito con gli stessi parametri, il risultato sul file sarà lo stesso
    - Le primitive UNIX **non** sono idempotenti (a causa dell'avanzamento implicito del puntatore di letture/scrittura).
  - L'interfaccia del File Service è adatta ad essere implementata da un server privo di stato (*stateless*). Questa è una caratteristica desiderabile, in quanto un server *stateless* può essere fatto ripartire dopo un crash senza che né lui né i client debbano ripristinare alcuno stato.

Operazioni implementate dal Directory Service

- ***FileID = LookUp(Dir, Name)***
  - Ritorna l'identificatore del file di nome *Name* all'interno della directory *Dir*
- ***AddName(Dir, Name, FileID)***
  - Aggiungi all'interno della directory *Dir* il file di identificatore *FileID* con il nome *Name*
- ***UnName(Dir, Name)***
  - Rimuove il file *Name* dalla directory *Dir*
- ***NameSeq = GetNames(Dir, Pattern)***
  - Ritorna la sequenza dei nomi dei file contenuti nella directory *Dir* che corrispondono all'espressione regolare *Pattern.*

Sun Network File System (NFS)



Virtual File System

- Questo modulo consente a UNIX di fornire una interfaccia uguale a file system differenti, i quali possono essere sia locali che remoti, e possono essere implementati a basso livello in modo differente.
  - Ad esempio, sotto Linux è possibile utilizzare in modo del tutto trasparente file system diversi da ext2fs (vfat, Minix, NFS, ...)
- Gli identificatori usati da NFS per individuare ciasun file sono chiamati *File Handle*, e vengono derivati dall'*i-node* UNIX del file:

Filesystem ID	I-node number	I-node generation number

- Si noti che dal punto di vista del client, il contenuto di un File Handle è del tutto ininfluente

```
/*
 * This is the kernel NFS client file handle representation
 */
#define NFS_MAXFHSIZE      64
struct nfs_fh {
        unsigned short          size;
        unsigned char           data[NFS_MAXFHSIZE];
};
```

Esempio di VFS: Linux

- Ad esempio, questa è l'interfaccia che il layer VFS di Linux presenta all'utente per le operazioni sui file
  - si veda */usr/src/linux/Documentation/filesystems/vfs.txt*

```
struct file_operations {
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
        int (*readdir) (struct file *, void *, filldir_t);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, struct dentry *);
        int (*fasync) (struct file *, int);
        int (*check_media_change) (kdev_t dev);
        int (*revalidate) (kdev_t dev);
        int (*lock) (struct file *, int, struct file_lock *);
 };
```

Integrazione del Client

- Il client NFS è approssimativamente equivalente al client astratto del File Service descritto in precedenza.
- Il client NFS è integrato all'interno del kernel UNIX.
- Questo comporta una serie di vantaggi:
  - I programmi utente possono accedere ai file tramite NFS senza bisogno di ricompilazione;
  - Un unico client NFS serve tutti i processi utente sullo stesso processore, potendo così gestire una unica cache per ottimizzare gli accessi.
  - Se si fa uso di una chiave per autenticare gli utenti nel server NFS, questa chiave può essere trattenuta all'interno del client (nel kernel), migliorando la sicurezza.

Autenticazione e controllo degli accessi

- Il server NFS (fino alla versione 3) è di tipo *stateless*, quindi ad ogni accesso il server deve verificare le credenziali del client, per verificare che esso sia abilitato ad accedere il file
- Il meccanismo di autenticazione originale non era particolarmente robusto
  - Le comunicazioni tra client e server si basano su Sun RPC; questo protocollo impone di inviare le informazioni di autenticazione dell'utente (ad es, lo User ID e Group ID)
  - Poiché il server NFS sta in ascolto su una porta pubblica, disponibile ad ogni processo esterno, sarebbe possibile per un utente inviare direttamente delle richieste RPC appositamente costruite con dati fasulli
- Versioni recenti di RPC implementano protocolli crittografici più robusti (DES e Kerberos) per codificare le informazioni degli utenti

Operazioni del Server NFS

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*.

Operazioni del server NFS (cont.)

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*.

Mount Service

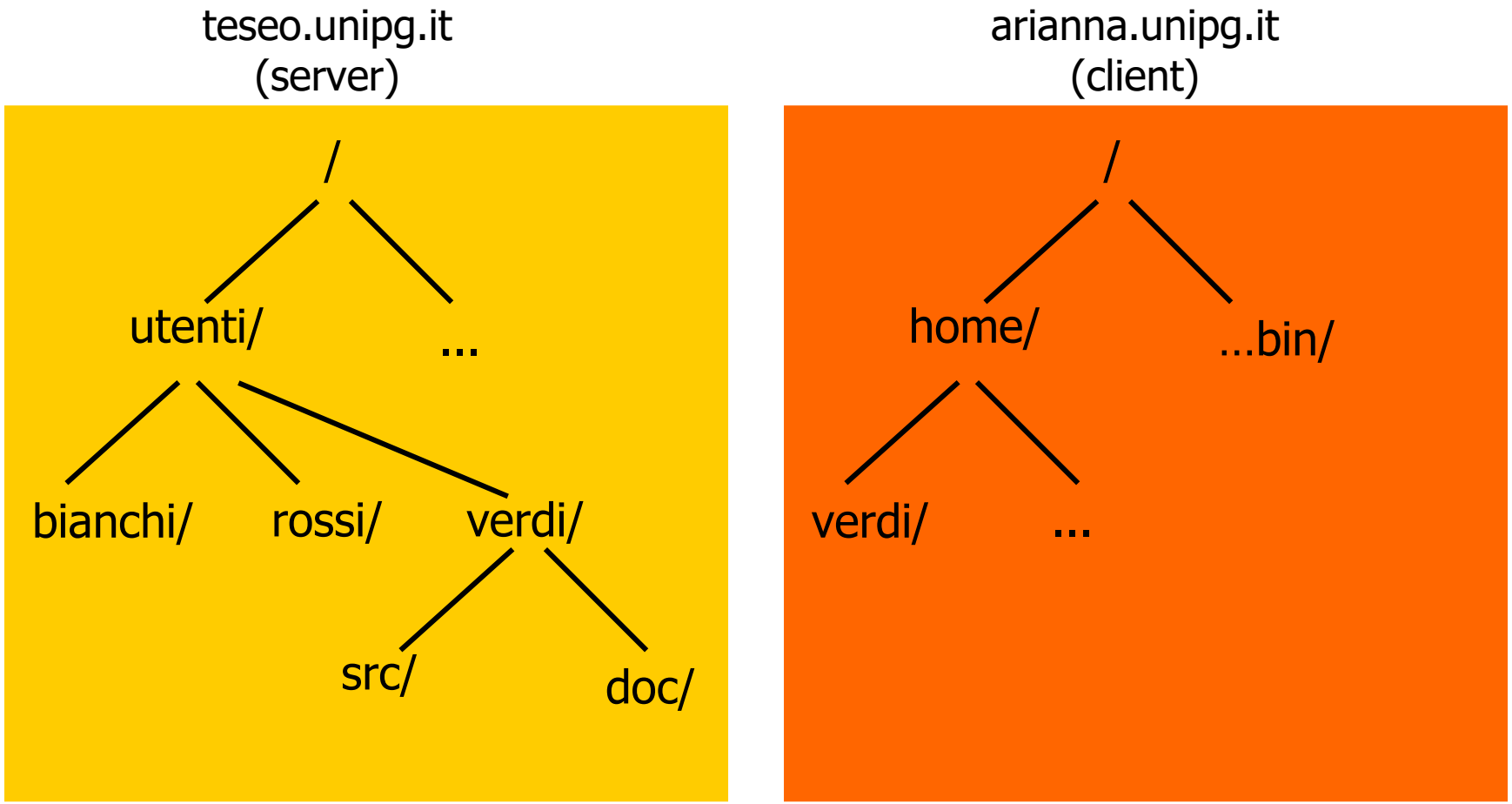
- Il *mount* di un filesystem remoto via NFS si effettua con un apposito *mount service*, che è un processo utente che gira su ogni server NFS
- Ogni server ha un file di configurazione (*/etc/exports*) che contiene la lista dei filesystem che può esportare agli utenti
- Gli utenti possono effettuare il mount di un filesystem remoto utilizzando una versione apposita del comando *mount*, indicando
  - Nome del server
  - Percorso di una directory nel filesystem remoto
  - Percorso locale in cui montare il filesystem

Nota sul comando *mount* per NFS

Dalla man page di mount(8) su Linux:

For most types all the mount program has to do is issue a simple mount(2) system call, and no detailed knowledge of the filesystem type is required. For a few types however (like nfs, smbfs, ncpfs) ad hoc code is necessary. The nfs ad hoc code is built in, but smbfs and ncpfs have a separate mount program.

Esempio


teseo.unipg.it
(server)
/
utenti/
...
bianchi/
rossi/
verdi/
src/
doc/
arianna.unipg.it
(client)
/
home/
...bin/
verdi/
...

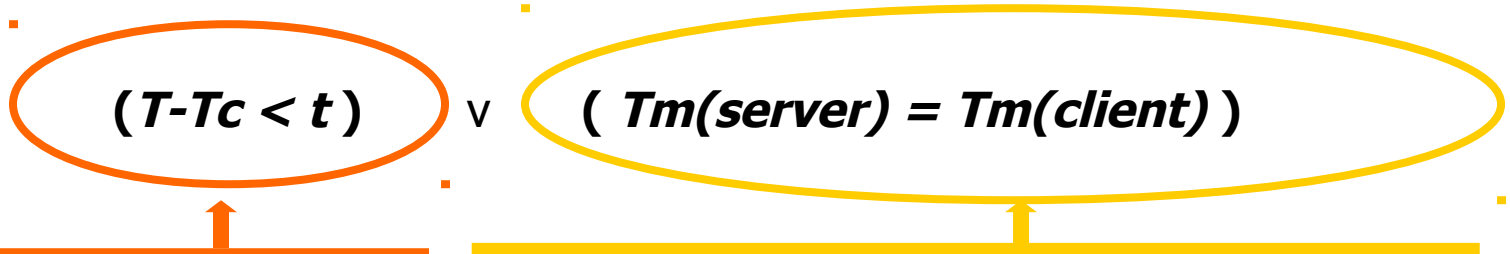
Caching nel server

- Le implementazioni di NFS sfruttano una cache sia nel client che nel server;
- Per il server, la cache funziona come ci si aspetta: blocchi dei files appartenenti ai filesystem esportati sono trattenuti in memoria per poter rispondere più velocemente ai client in caso di richieste consecutive dello stesso blocco;
- Due meccanismi per garantire la consistenza in scrittura:
  - Write Through
    - I blocchi vengono scritti su disco non appena il server riceve la richiesta di scrittura da un client
  - Write Back
    - I blocchi non vengono subito scritti su disco dopo la ricezione della richiesta di scrittura. NFS versione 3, che usa questo tipo di cache, implementa una nuova primitiva *commit( )* con la quale il client può chiedere la scrittura di tutti i blocchi in cache.

Caching nel client

- Il client trattiene in cache i risultati delle operazioni di lettura e scrittura per diminuire il numero di richieste spedite al server.
- Il meccanismo di caching nel client introduce un problema potenziale: l'esistenza di versioni differenti dello stesso file in nodi diversi della rete
  - Questo perché le operazioni di scrittura da parte di un client possono non venire propagate immediatamente nel server e alle copie trattenute in cache da altri client
- Il problema viene risolto assegnando al client la responsabilità di verificare che i dati nella sua cache siano consistenti:
  - Ogni entry nella cache del client contiene il tempo dell'ultima validazione, *Tc* , ed il tempo dell'ultima modifica *Tm*

Condizione di validità della Cache



Il tempo trascorso dall'ultima validazione non deve superare un certo valore *t*

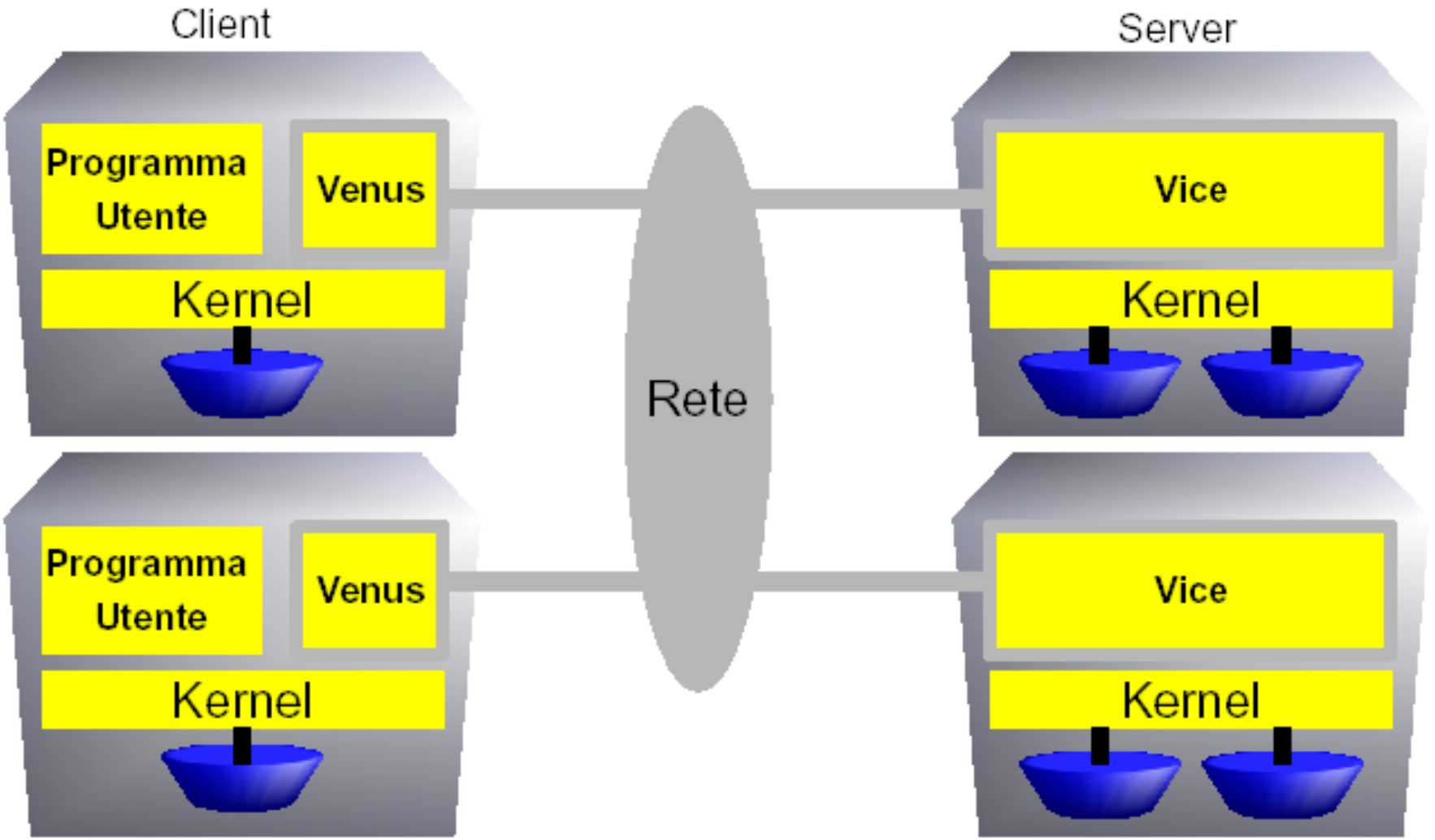
Il tempo dell'ultima modifica registrata dal server deve coincidere con il tempo dell'ultima modifica registrata dal client

NB: **Questa condizione deve essere verificata *ad ogni accesso a un file condiviso***

Andrew File System (AFS)

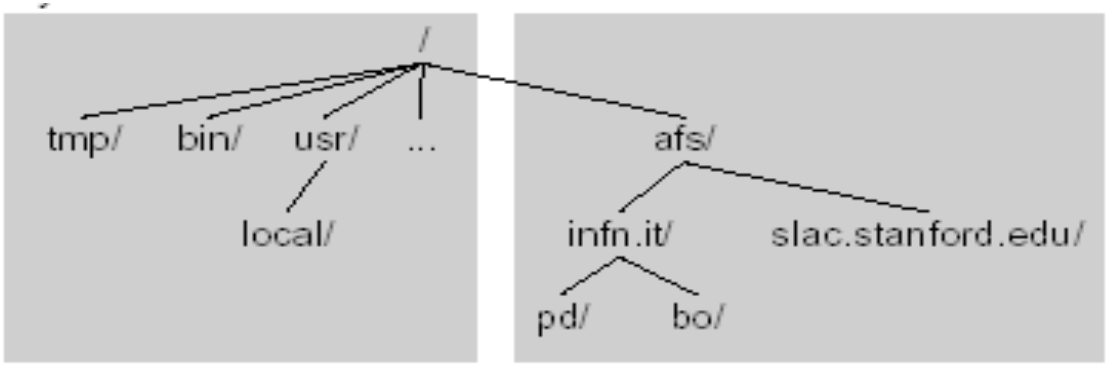
- Lo sviluppo di AFS inizia nel 1983 da uno sforzo congiunto tra la Carnegie Mellon University e IBM
- AFS, come NFS, consente alle applicazioni l'accesso a file system remoti, senza bisogno di ricompilazioni
- AFS differisce da NFS in quanto pone maggiormente l'accento sulla *scalabilità*. Questo ha portato a scelte implementative diverse da quelle adottate per NFS.
- Caratteristiche salienti:
  - I file vengono trasferiti interamente dai server ai client in un unica operazione (in AFS-3, file più grandi di 64K vengono trasferiti in blocchi di 64K)
  - I client tengono nella loro cache locale le copie dei file (interi) ricevuti dai server

AFS: Architettura



Implementazione

- AFS è implementato con due componenti software:
  - **Venus** è un processo **client** che opera a livello utente su ogni workstation
  - **Vice** è un processo **server**, anche lui operante a livello utente.
- Il filesystem di ciascuna workstation è suddiviso in due parti
  - la parte locale è visibile solamente alla workstation, e non viene condivisa
  - la parte condivisa è memorizzata su un server, e può esistere in varie copie parziali sparse nelle cache dei client. I file condivisi si trovano sempre nella sottodirectory */afs*



Scenario

- Funzionamento di AFS:
  - Quando un processo su una workstation client apre un file remoto, e non ne esiste una copia nella cache locale, il file viene localizzato e ne viene richiesta una copia
  - La copia viene memorizzata nel file system locale alla workstation.
  - Successive operazioni di lettura/scrittura da parte del processo vengono compiute sulla copia locale
  - Quando il processo chiude il file, se la copia locale è stata modificata, questa viene rispedita al server. La copia locale alla workstation viene mantenuta, nel caso venga successivamente richiesta.

Osservazioni

- File che vengono aggiornati raramente, oppure che vengono utilizzati in lettura/scrittura da un solo utente, possono rimanere validi a lungo nella cache dei client.
- La cache locale di ciascuna workstation può essere molto capiente, in quanto risiede sul disco locale.
- L'implementazione di AFS si basa su alcune considerazioni circa il presunto utilizzo dei file in ambiente UNIX:
  - La maggior parte dei file sono piccoli
  - Operazioni di lettura sono molto più frequenti di operazioni di scrittura
  - L'accesso sequenziale è quello comunemente usato
  - La maggior parte dei file vengono modificati da un solo utente
  - I file vengono acceduti in *burst*
- I database tipicamente non hanno queste caratteristiche, quindi non si prestano bene ad essere condivisi tramite AFS

Consistenza della Cache AFS

- Quando un modulo Vice (server) fornisce un file a Venus (client), insieme al file viene fornito un *callback promise*
  - Cioè la *promessa* da parte del server a ricontattare il client nel caso il file venga modificato da parte di altri client
  - A ciascun file nella cache dei client viene associato il callback. Questo può avere due stati:
    - **Valid**: Il server non ha ancora comunicato modifiche al file, che quindi è da ritenersi valido
    - **Cancelled**: Il server ha richiamato, comunicando di invalidare la copia locale del file perché è stata modificata.
- Quando un server riceve la richiesta di modificare un file, manda un avviso a tutti i client ai quali ha promesso un *callback*
  - Quindi, è chiaro che a differenza di NFS, i server AFS non sono stateless
- Una volta contattato dal server, il client provvede a porre il callback associato a quel file come *cancelled*.

Consistenza della Cache AFS (cont.)

- Quando una workstation riparte dopo un crash, deve cercare di mantenere la maggior parte del contenuto della propria cache.
- Purtroppo, durante il crash potrebbe aver perso alcuni dei callback da parte dei server, quindi la cache va *rivalidata*
- Per (ri)validare la cache, Venus manda una *cache validation request* contenente il file da validare e la data di ultima modifica, come risulta dal client
- Se il server verifica che non ci sono state modifiche a partire dalla data indicata dal client, il file (e relativo callback) viene validato
- Altrimenti, il callback viene posto a *cancelled*.
- In ogni caso, ciascun client deve rivalidare un callback una volta trascorso un certo intervallo di tempo senza ricevere notifiche dal server.

Consistenza della Cache AFS (cont.)

- Il meccanismo di consistenza appena descritto entra in azione solo durante le operazioni di *open( )* e *close( )*.
- Ciò significa che c'è potenzialmente la possibilità che clienti diversi tentino di modificare lo stesso file, con risultati inconsistenti.
- Per evitare ciò, è responsabilità dei programmi utente adottare apposite strategie per garantire la consistenza delle operazioni.
  - Questo è in linea con la semantica delle operazioni standard UNIX sui file, che di default non forza nessuna verifica della concorrenza.

Coda

- È discendente di AFS-2, e mette maggiormente l'accento sulla robustezza del sistema anche a fronte di fallimenti dei server e/o della rete.
- Si basa su processi chiamati Vice e Venus, che hanno funzionalità simili a quelle già viste per AFS-2
- Coda fornisce agli utenti tutti i vantaggi di un filesystem condiviso, ma permette a ciascuno di lavorare con risorse locali in caso i dati condivisi siano parzialmente o del tutto inaccessibili
- Coda si presta bene a supportare utenti *mobili*, che alternano fasi di lavoro in cui sono connessi alla rete (e al filesystem condiviso) a fasi in cui sono sconnessi.

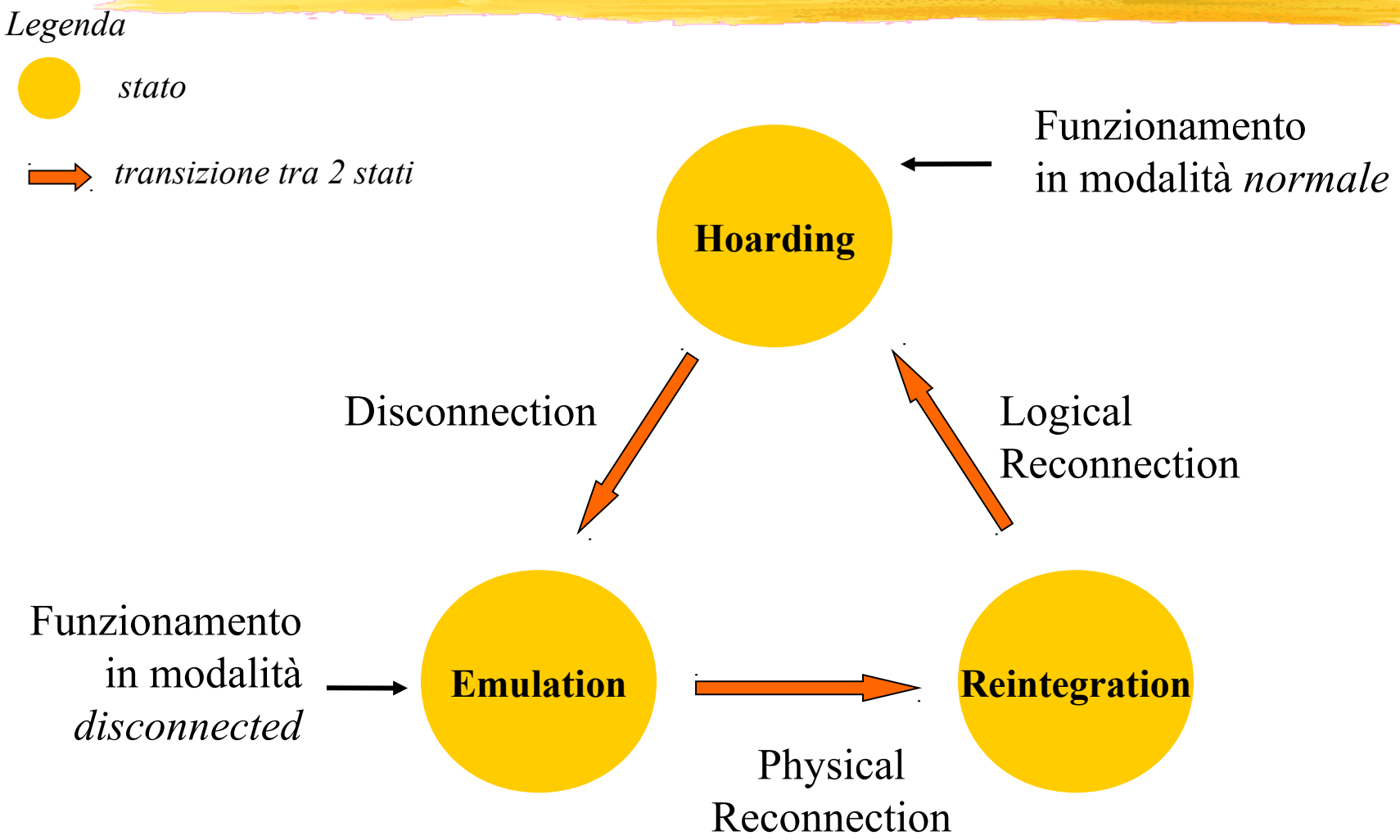
Architettura di Coda

- L'architettura di Coda è molto simile a quella di AFS-2:
  - I client mantengono nella cache interi file. Questo offre intrinsecamente una garanzia di robustezza contro i fallimenti, poiché la copia locale dei file rimane disponibile anche dopo la sconnessione dal server
  - La coerenza della cache viene mantenuta usando i *callback*
  - I client usano dei suggerimenti (*hints*) per mantenere informazioni sulla localizzazione dei server
  - Meccanismi di sicurezza integrata permettono lo scambio sicuro di informazioni tra client e server.
- Per migliorare la resistenza contro i fallimenti:
  - Replicazione dei server
  - Passaggio automatico alla modalità operativa *disconnected* nel caso in cui nessun server sia raggiungibile.

Operatività in modalità *disconnected*

- Normalmente ciascun client opera circa come in AFS-2; in più, anticipando eventuali sconnessioni improvvise, è costantemente in modalità *hoarding*
  - In pratica, il client cerca di prelevare dal server tutti i file di cui prevede l'utente avrà bisogno per operare in modalità disconnessa.
  - L'utente può intervenire manualmente specificando i files che più gli interessano; altrimenti, il client si basa sulla storia degli accessi recenti per capire quali file sono stati più frequentemente utilizzati
- A seguito della sconnessione fisica, ciascun client passa in modalità *emulation*, in cui *simula* la presenza del server
  - Sostanzialmente facendo uso solo delle informazioni nella cache locale.
  - Inoltre, deve registrare localmente ogni informazione che risulti necessaria per ripetere, una volta riconnesso, tutte le modifiche effettuate.
- Quando l'utente si riconnette, il client entra in modalità *reintegration*
  - le modifiche locali vengono propagate al server ed eventuali conflitti vengono risolti

Transizione tra gli stati di Venus



Alcune considerazioni e suggerimenti

- **Distinguere i client dai server**
  - Sembra banale, ma è una decisione fondamentale. Facendo questa distinzione è possibile dimensionare adeguatamente le caratteristiche dei client e dei server per garantire un servizio ottimale contenendo i costi
  - Si possono gestire meglio le politiche di accesso e di sicurezza
- **Fare il più possibile ricorso alle risorse dei client**
  - Tenendo presente le caratteristiche di un File System distribuito, quando si deve decidere se far compiere una operazione al client o al server, conviene quasi sempre assegnarla al client.
  - Questo diminuisce la necessità di dover aumentare le risorse sui server nel caso in cui il sistema debba crescere oltre le previsioni
- **Usare le cache ove possibile**
- **Sfruttare le statistiche di accesso ai dati**
  - La conoscenza dei *pattern* di utilizzo dei file consente di costruire soluzioni efficienti che traggano il maggior beneficio possibile da questa informazione

Alcune considerazioni e suggerimenti

- **Minimizzare la necessità di avere conoscenze globali**
  - In un sistema distribuito di larga scala, è spesso difficile se non impossibile essere a conoscenza dello stato globale. La scalabilità di un File System distribuito sarà certamente migliore se non è necessario fare affidamento o aggiornare informazioni globali
- **Fidarsi del minor numero di entità possibile**
  - Un sistema la cui sicurezza si basa sul minor numero di entità possibili è più probabile rimanga sicuro man mano che viene espanso.
- **Raggruppare le operazioni in batch**
  - Raggruppare tante piccole operazioni in poche operazioni più grosse spesso migliora l'utilizzazione della rete.

Nuove implementazioni di NFS

- Sebbene NFS in principio usa il modello remote access, molte implementazioni usano cache locali, che in pratica corrisponde ad usare il modello upload/download.

- NSF implementa la *session semantics*:
  - Le modifiche ad un file aperto sono inizialmente visibili solo al processo che ha modificato il file. Quando il file viene chiuso tutte le modifiche sono visibili agli altri processi (computer).

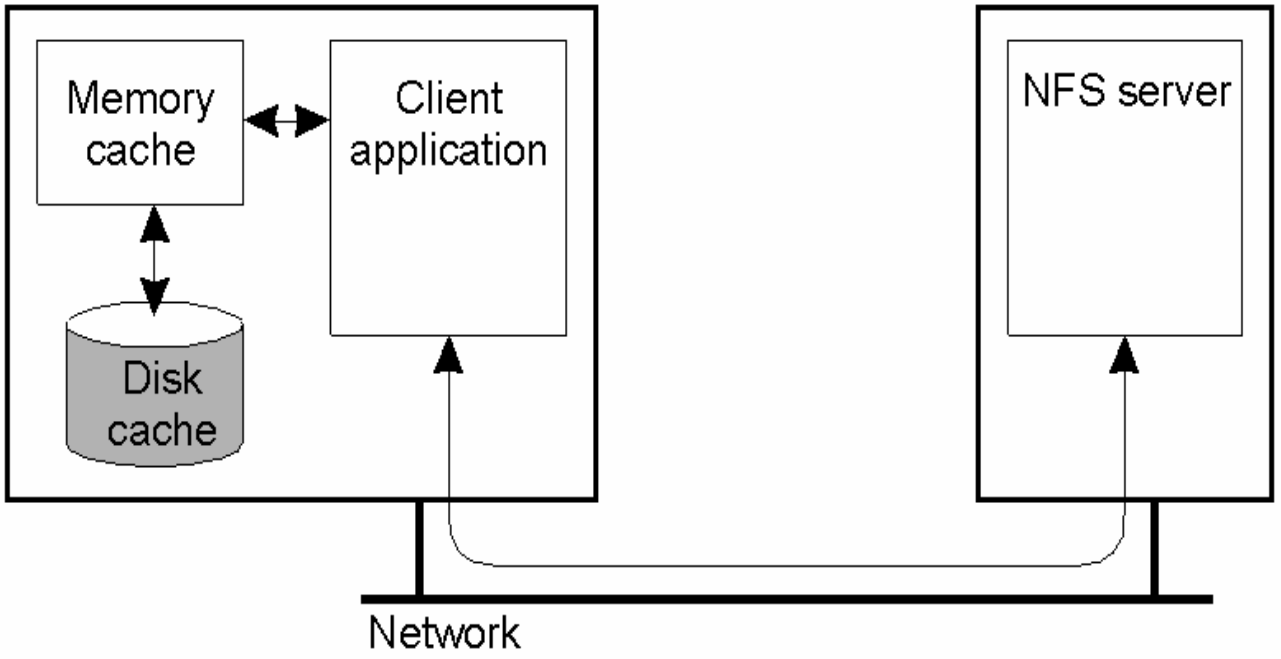
Nuove implementazioni di NFS

- Cosa accade quando due processi memorizzano localmente un file e lo modificano?
- Quattro differenti modelli per gestire file condivisi in un sistema distribuito:

Metodo	Commento
UNIX semantics	Ogni modifica su un file è instantaneamente visibile a tutti i processi
Session semantics	Nessuna modifica è visibile ad altri processi prima che il file venga chiuso
Immutable files	Non sono permesse modifiche; una modifica crea un nuovo file
Transaction	Tutte le modifiche sono atomiche

NFS v. 4 lato client

- Mentre la versione 3 non usa una cache, dalla versione 4 NFS implementa un sistema di caching sul lato client che include una Memory cache e una Disk cache.
- Per un file: i dati, gli attributi, gli handle, e le directory possono essere memorizzati nella cache.

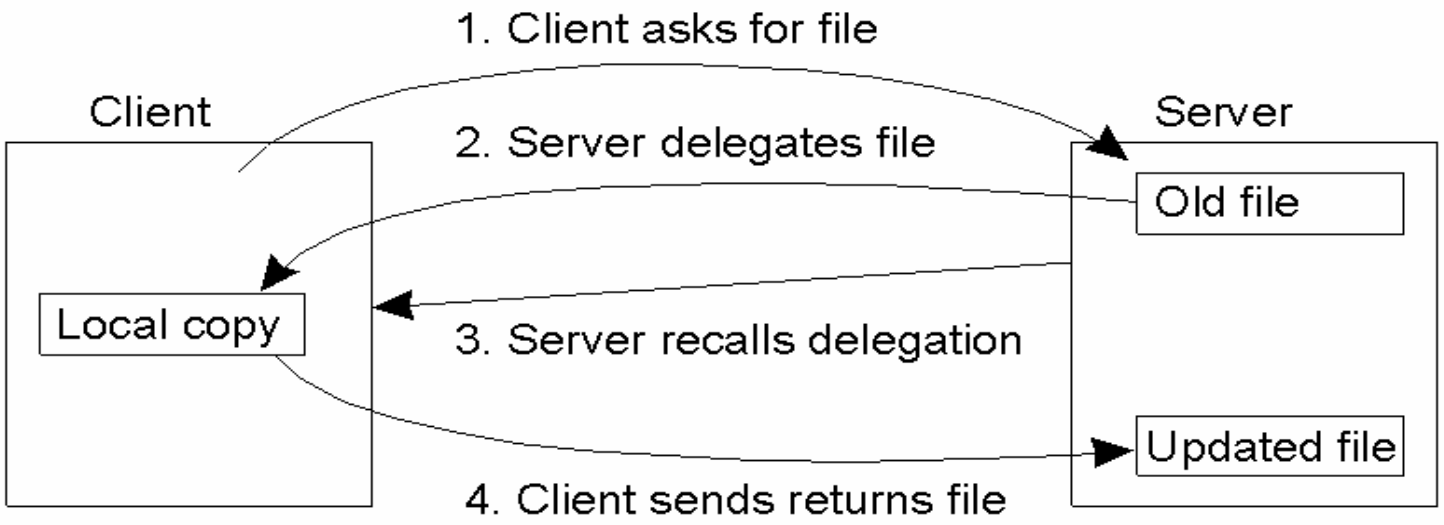


NFS v. 4 lato client

- Il sistema di caching dei dati di un file usa la session semantics : le modifiche dei dati nella cache vengono spostate sul server quando un client chiude il file.
- I dati possono essere mantenuti nella cache, ma se il file sarà riaperto, dovranno essere rivalidati.
- NFS usa anche la open delegation per delegare alcuni diritti al client che ha aperto il file.
- Il client può prendere decisioni relative al proprio nodo senza chiedere al server. Altre decisioni rimangono al server.

NFS v. 4 lato client

- Un server può aver bisogno di ritirare una delega quando un altro client su una macchina differente chiede i diritti di accesso per un file.
- Il meccanismo di callback è usato che ritirare una file delegation.



NFS v. 4 lato client

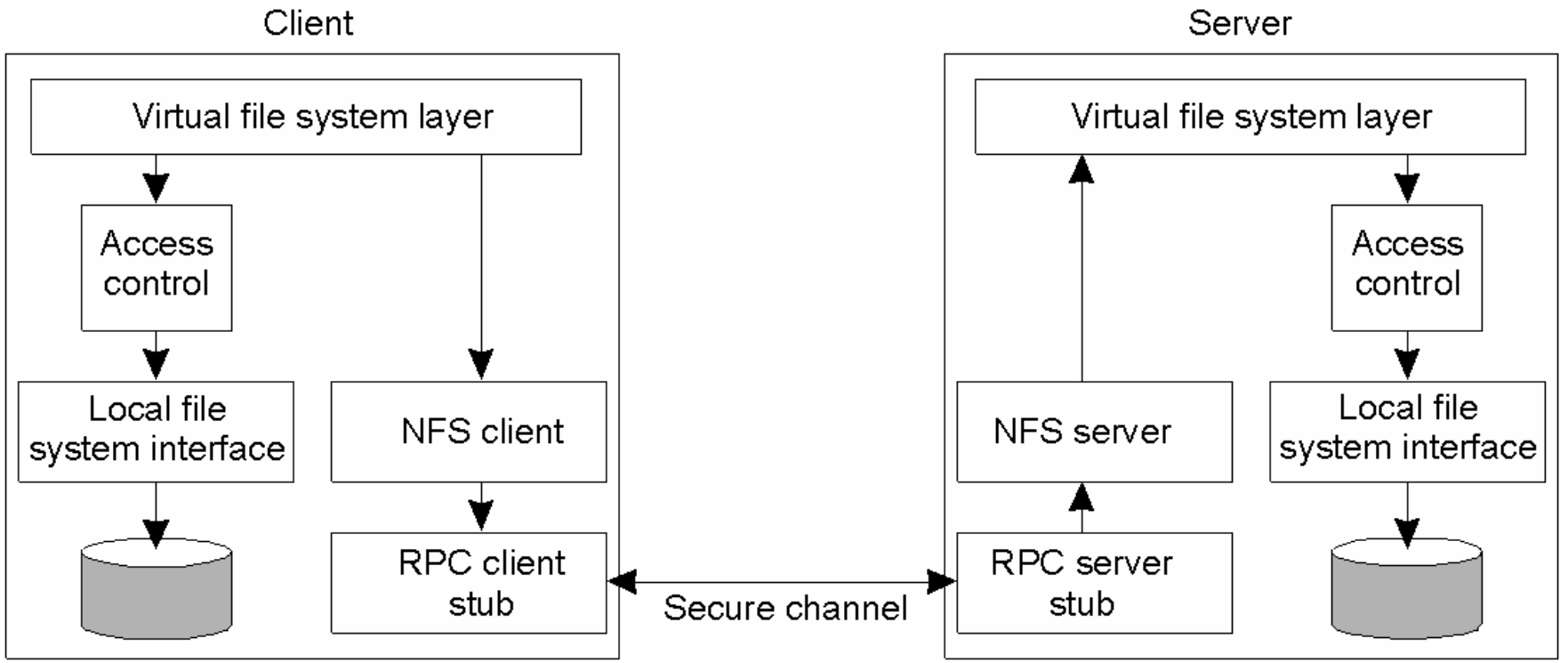
- Attributi, file handle, e directory possono essere memorizzati nella cache, ma le modifiche di questi valori devono essere inviate al server.
- Le informazioni nella cache sono automaticamente invalidate dopo un dato intervallo di tempo. Questo obbliga i clienti di ri-validarli prima di poterle riusare.
- NFS v4 offre un supporto per la replicazione di un file system tramite una lista di locazioni dove il file system può essere memorizzato (su diverse macchine del sistema distribuito).

NFS v. 4 Fault tolerance

- Poichè NFS v4 implementa server stateful (e gestisce file locking, open delegation, ecc.), sono necessari meccanismi di fault tolerance e di recovery per gestire eventuali fallimenti delle RPC.
- Le RPC di NFS usano protocolli TCP e UDP che hanno diversi livelli di affidabilità.
- Ad esempio: RPC può generare richieste duplicate quando si ha la perdita di una chiamata di procedura remota. Questo può portare il server ad effettuare più volte una richiesta.
- E' necessario gestire la duplicazione di chiamate.

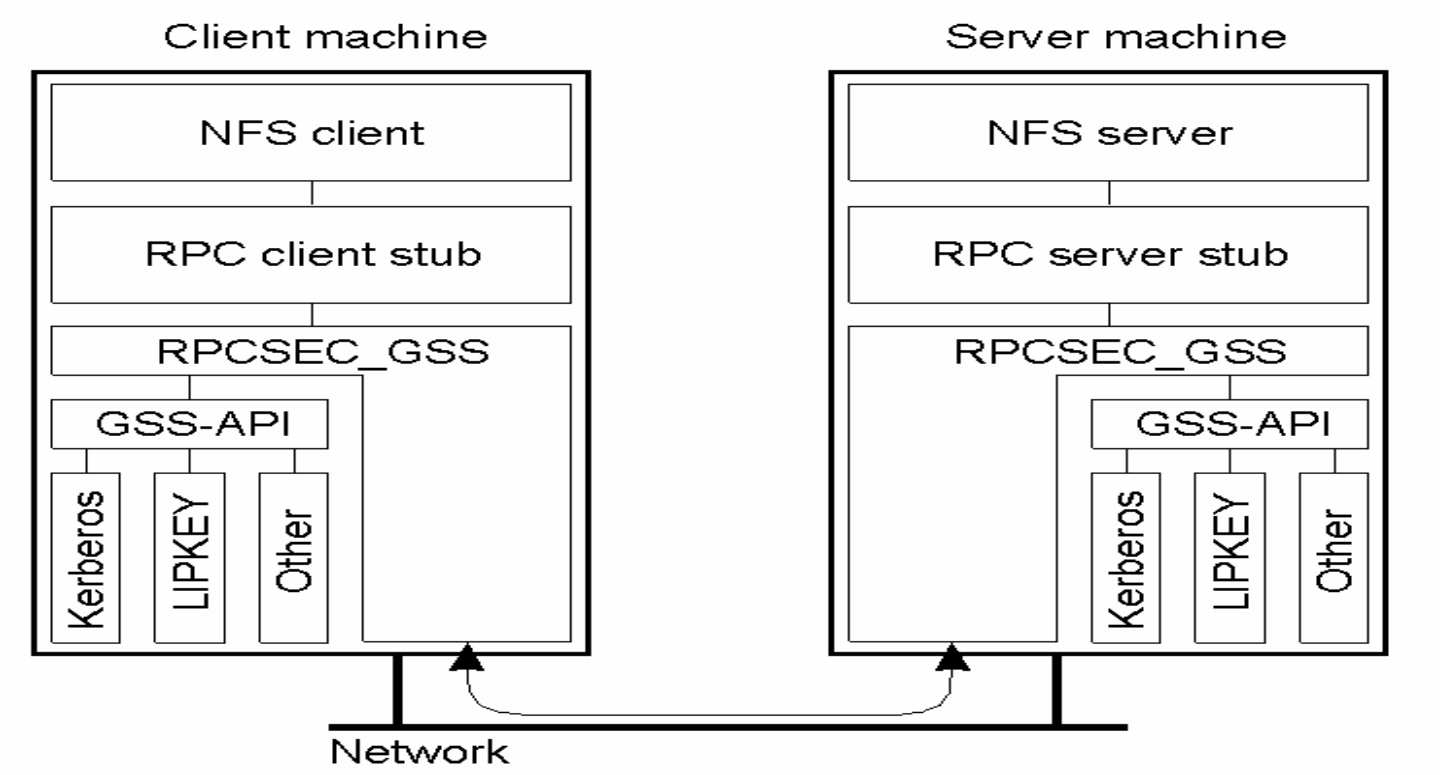
NFS v. 4 Sicurezza

- La sicurezza in NFS è basata su canali di comunicazione sicuri (secure channels) e meccanismi di controllo degli accessi (file access control).



RPC Sicuro

- RPC Sicuro in NFS v4 è basato su RPCSEC_GSS.
- RPCSEC_GSS - Generic Security Service authentication protocol per ONC RPC basato su Generic Security Services API (GSS API)



Controllo accessi

- Operazioni riconosciute da NFS rispetto al controllo degli accessi

Attributi ACL	Descrizione
Write_owner	Permission to to change the owner
Write_named_attrs	Permission to to write the named attributes of a file
Read_named_attrs	Permission to to read the named attributes of a file
Write_attributes	Permission to to change the other basic attributes of a file
Read_attributes	The ability to read the other basic attributes of a file
Write_acl	Permission to to write the ACL
Read_acl	Permission to to read the ACL
Delete_child	Permission to to delete a file or directory within a directory
Delete	Permission to to delete a file
Add_subdirectory	Permission to to create a subdirectory to a directory
Add_file	Permission to to add a new file to a directory
List_directory	Permission to to list the contents of a directory
Execute	Permission to to execute a file
Append_data	Permission to to append data to a file
Write_data	Permission to to modify a file's data
Read_data	Permission to read the data contained in a file
Synchronize	Permission to to access a file locally at the server with synchronous reads and writes

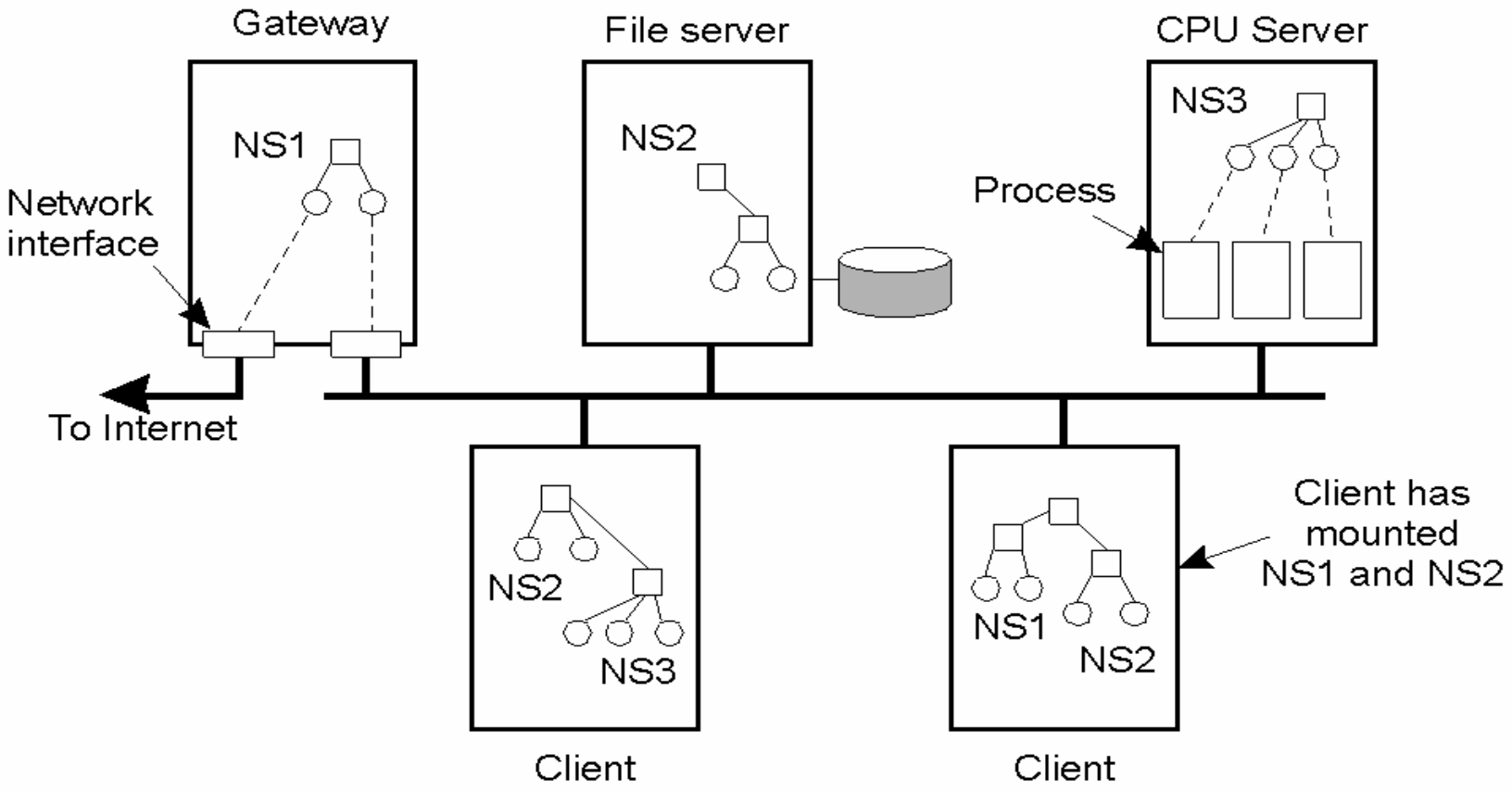
Tipi di utenti NFS

- Operazioni riconosciute da NFS rispetto al controllo degli accessi

Tipo d'utente	Descrizione
Service	Qualsiasi processo di servizio del sistema che accede il file
Authenticated	Qualsiasi utente autenticato
Anonymous	Qualsiasi utente che accede il file senza autenticazione
Batch	Qualsiasi processo che accede il file come parte di un job batch
Dialup	Qualsiasi processo che accede il file tramite una connessione lenta
Network	Qualsiasi processo che accede il file dalla rete
Interactive	Qualsiasi processo che accede il file in modalità interattiva
Everyone	Qualsiasi utente di un processo
Group	Gruppo degli utenti associato al file
Owner	Proprietario del file

PLAN 9
Risorse accedute come file

- Tutte le risorse sono accedute usando una sintassi simile a quella usata per i file su un pool di server.



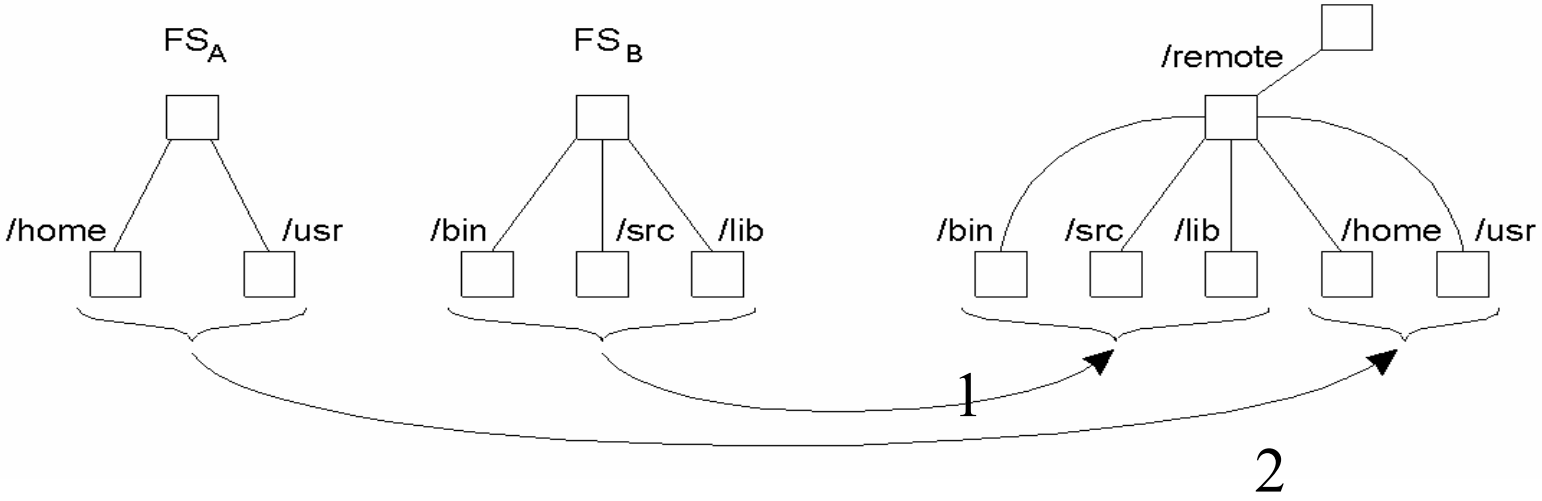
PLAN 9
Comunicazioni

- Per le comunicazioni, Plan 9 usa il protocollo 9P e le interfacce di rete rappresentate come directory.
- I file associati ad una singola connessione TCP in Plan 9 sono:

FILE	DESCRIZIONE
Status	Fornisce informazioni sullo stato della connessione
remote	Fornisce informazioni sul lato remoto (chiamato) della connessione
local	Fornisce informazioni sul lato chiamante della connessione
listen	Usato per accettare richieste di connessione in arrivo
data	Usato per leggere e scrivere dati
Ctl	Usato per comandi di controllo specifici del protocollo

PLAN 9
Naming

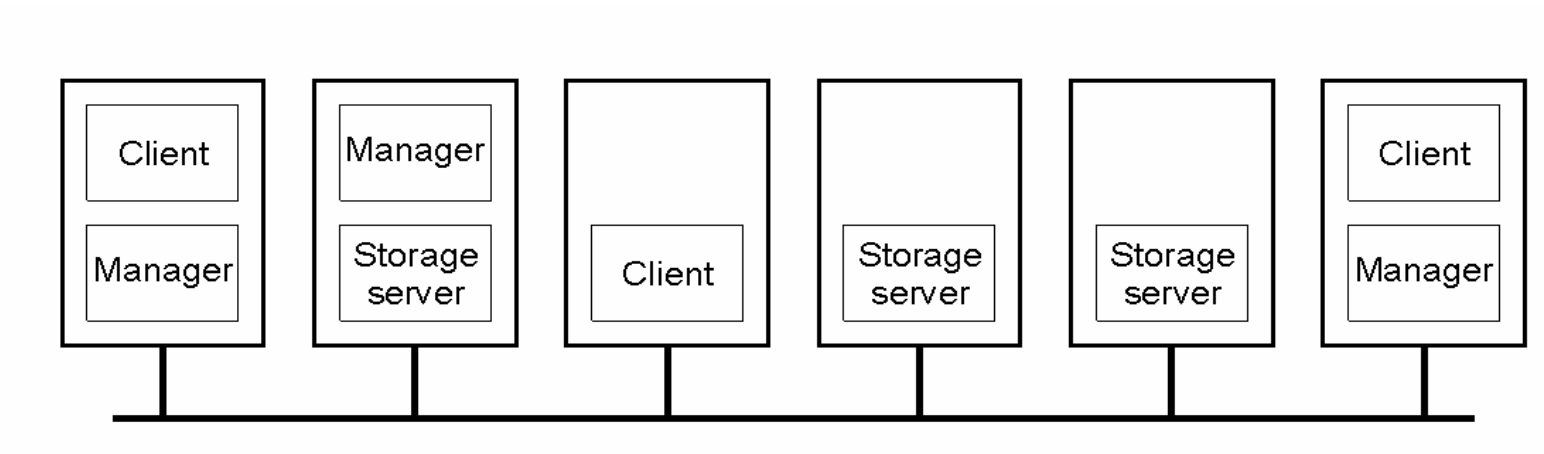
- Un client può montare più di un name space nello stesso mount point componendo una union directory.
- L'ordine di mounting è mantenuto durante la ricerca dei file.



Una union directory in Plan 9 dove è stato prima montato FS_B e dopo FS_A.

xFS
Principi

- Il file system xFS è basato su un modello serverless.
- L'intero file system è distribuito sulle diverse macchine inclusi i clienti.
- Ogni macchina può eseguire uno storage server, un metadata server e un processo client.



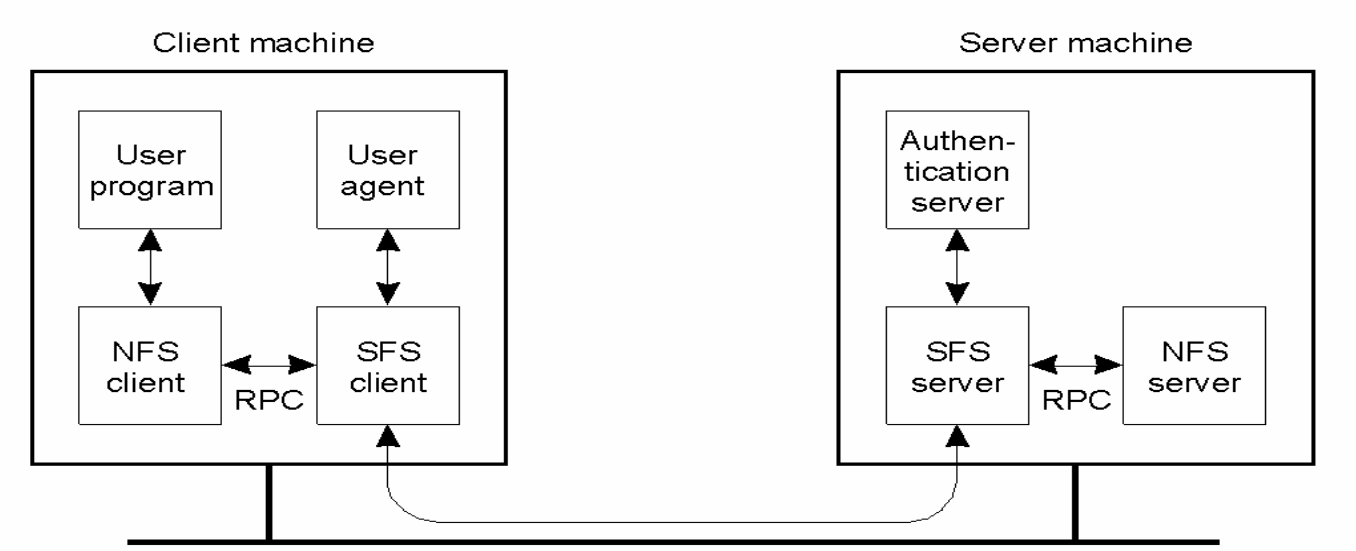
Una tipica distribuzione di processi xFS su più macchine

xFS
Comunicazioni

- Le prestazioni delle RPC in un ambiente altamente decentralizzato non sono ottimali a causa dei problemi di elevata comunicazione.
- In xFS le RPC sono state sostituite con gli *active messages*. I messaggi attivi sono realizzati tramite processi che gestiscono il loro arrivo.
- Nel modello degli active message, quando un messaggio arriva, un handler viene automaticamente eseguito per gestire il messaggio ed effettuare le opportune operazioni.

SFS
Principi

- Il Secure File System (SFS) usa delle chiavi segrete per implementare meccanismi di security nel file system.
- Un client non può accedere un file senza disporre della chiave segreta associata che deve essere prelevata in precedenza.
- La gestione delle chiavi è separata dalla gestione dei file.
- SFS usa NFS v3 per il controllo degli accessi degli utenti e per le comunicazioni.



Confronto tra FSD classici

Issue	NFS	Coda	Plan 9	xFS	SFS
Obiettivi	Trasparenza negli accessi	Alta disponibilità	Uniformità	Sistema serverless	Sicurezza scalabile
Modello di accesso	Remoto	Up/Download	Remoto	Basato su Log	Remoto
Comunicazioni	RPC	RPC	Speciale (9P)	Active msgs	RPC
Processo Client	Leggero/pesante	Pesante	Leggero	Pesante	Medio
Gruppi di Server	No	Si	No	Si	No
Granularità di mount	Directory	File system	File system	File system	Directory
Name space	Per client	Globale	Per process	Globale	Globale
Validità File ID	File server	Globale	Server	Globale	File system
Semantica Condiv.	Session	Transazionale	UNIX	UNIX	N/S
Consistenza cache	write-back	write-back	write-through	write-back	write-back
Replicazione	Minima	ROWA	Nessuna	Striping	Nessuna
Tolleranza ai guasti	Comunicazioni affidabili	Replicazione e caching	Comunicazioni affidabili	Striping	Comunicazioni affidabili
Recovery	Client-based	Reintegrazione	N/S	Checkpoint & write logs	N/S
Canali sicuri	Meccanismi esistenti	Needham-Schroeder	Needham-Schroeder	No pathnames	Self-certificate
Controllo degli accessi	Molte operazioni	Directory operations	UNIX based	UNIX based	NFS BASED

HDFS

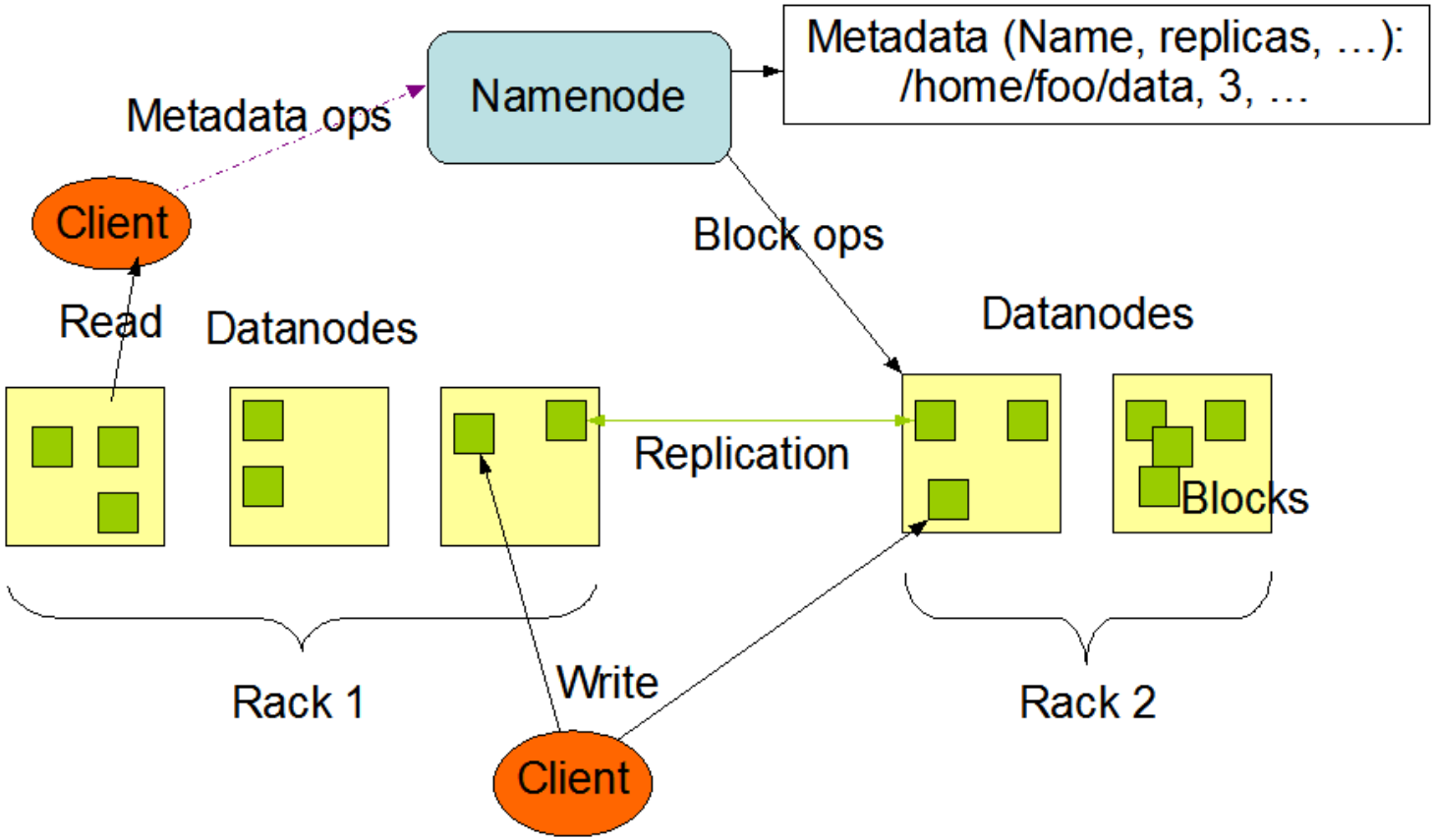
HDFS è il FSD Hadoop sotto licenza di Apache 2.0 sviluppato da Apache Software Foundation.

HDFS è un FSD semi-centralizzato, infatti i metadati sono gestiti da un singolo Server chiamato *Namenode*, mentre i dati sono divisi in blocchi, distribuiti e replicati in diversi *Datanode*.

Un *Secondary Namenode* permette ad HDFS di resettarsi con una nuova configurazione nel caso non sia disponibile il Namenode principale.

HDFS

HDFS Architecture



Ceph

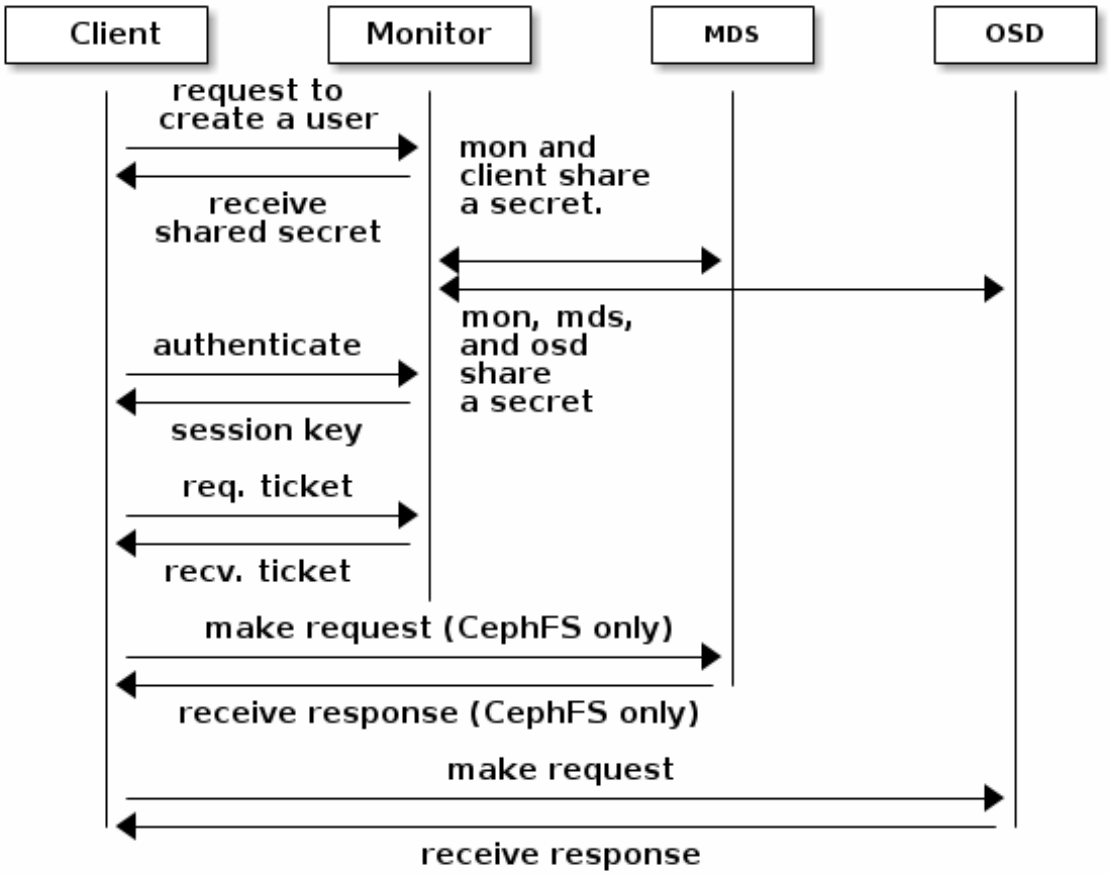
Ceph è un FSD Open Source sviluppato da Sage Weil.

La sua architettura garantisce una gestione dinamica e distribuita dei metadati tramite i *Metadata Server* (MDS) che costituiscono un *cluster*, i quali si occupano, oltre che del namespace, della sicurezza e della consistenza del sistema, anche di eseguire query sui metadati.

Dati e metadati vengono memorizzati tramite gli *Object Storage Daemon* (OSD), che invece hanno la funzione di svolgere operazioni di Input/Output.

Un Monitor mantiene le mappe dello stato del cluster. I monitor sono anche responsabili della gestione dell'autenticazione tra demoni e client.

Ceph

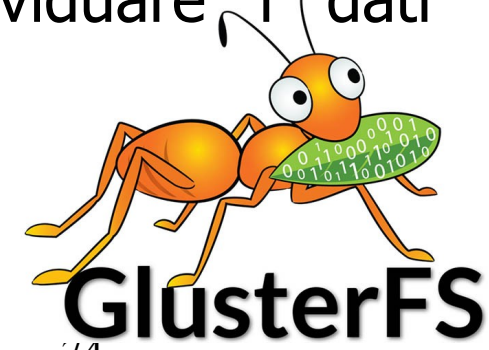

Client
Monitor
MDS
OSD
request to create a user
receive shared secret
mon and client share a secret.
mon, mds, and osd share a secret
authenticate
session key
req. ticket
recv. ticket
make request (CephFS only)
receive response (CephFS only)
make request
receive response

GlusterFS

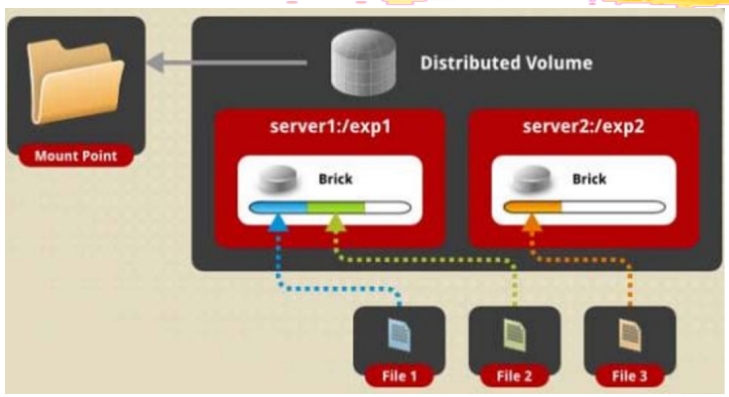
GlusterFS è un FSD Open Source sviluppato da Gluster Inc. che permette, a partire da *Volumes* hostati in molteplici Server, la costruzione di un FSD.

Ogni *Volume* può essere composto da più *Subvolume*, i quali, a loro volta, sono composti da *Brick*, che identificano i FS su cui opera GlusterFS.

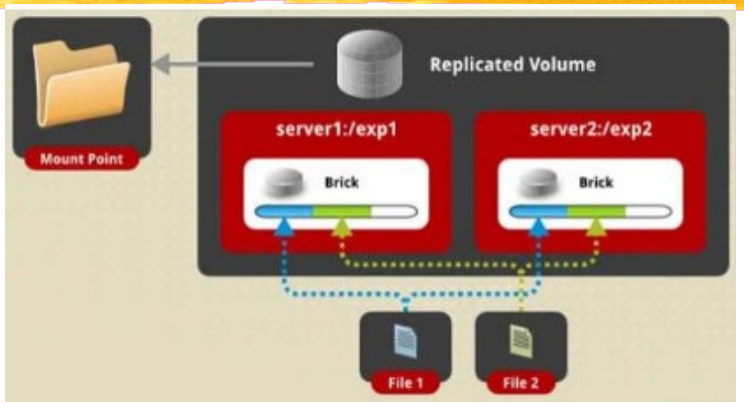
A differenza degli altri FSD, in GlusterFS non esiste alcun Server per i metadati, poiché vengono utilizzate Distribuited Hash Table (DHT) per individuare i dati all'interno dei *Brick*.



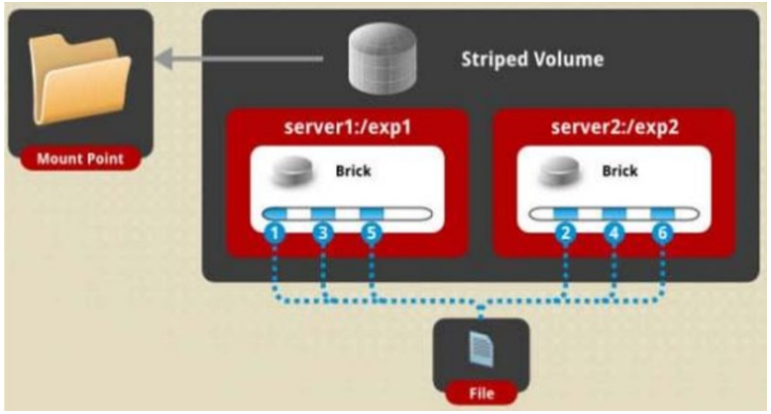
Tipi di Volume in GlusterFS



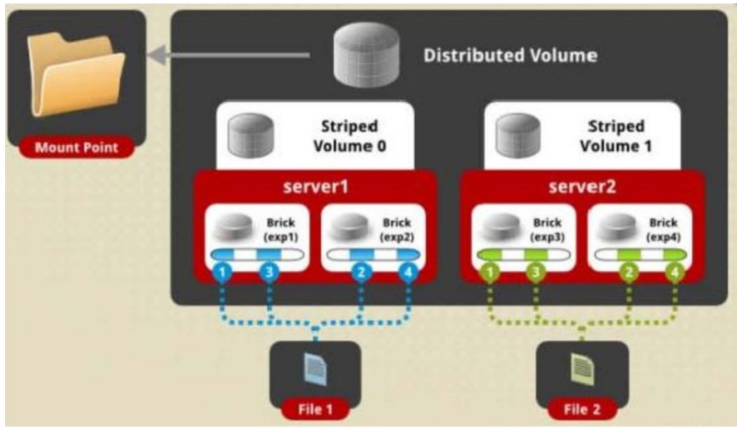
Distributed



Replicated

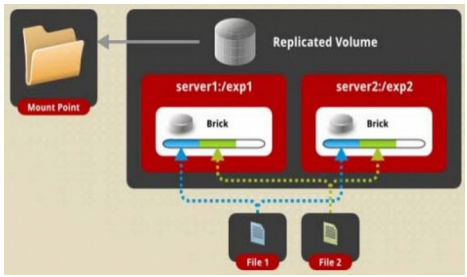


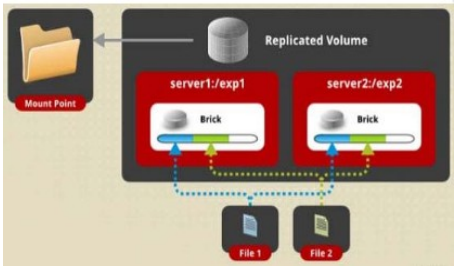
Stripped

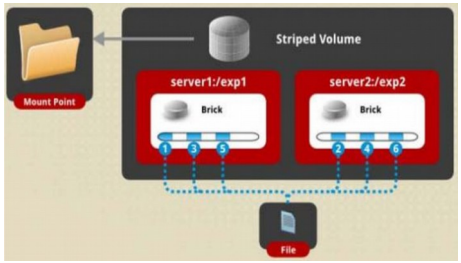


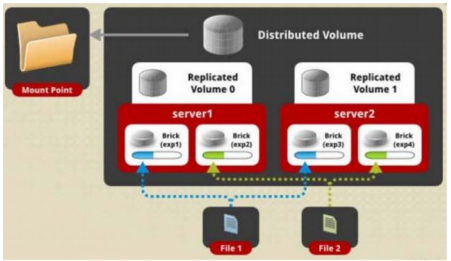
Distributed stripped

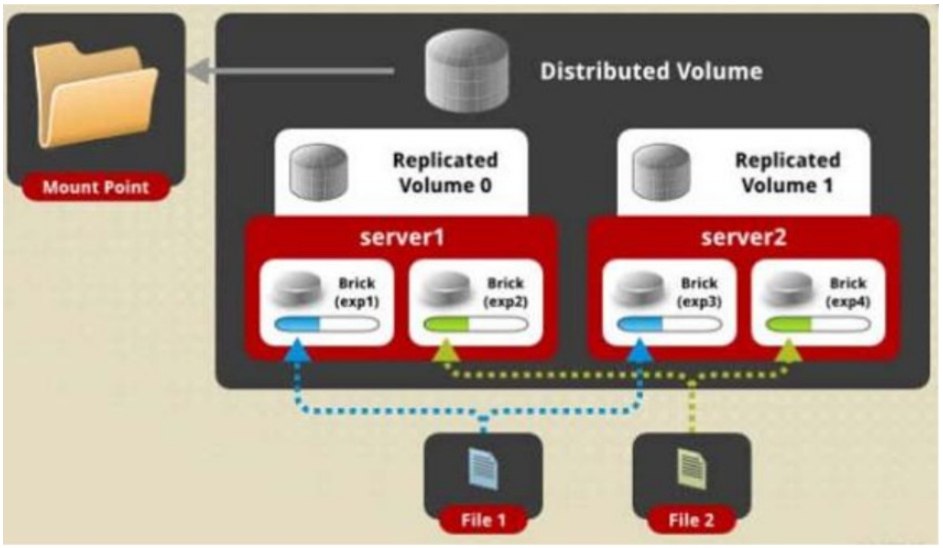
Tipi di Volume in GlusterFS

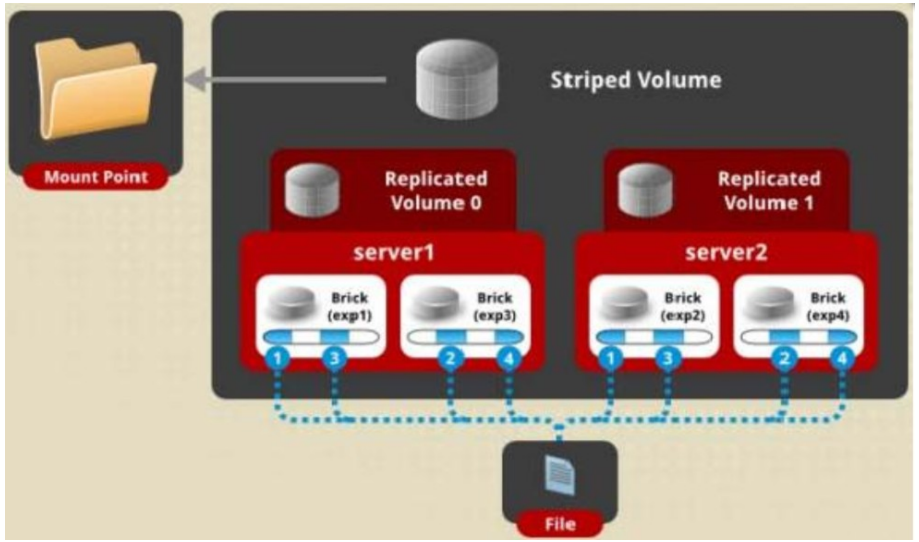












Distributed replicated

Stripped replicated

Mobile Communication Networks

- Una rete (network) è formata da nodi e connessioni (link). I nodi sono di due tipi:
  - endpoints o (hosts/ terminals) che operano come sorgente/ destinazione di traffico
  - switch (router/ access point/ base station) che hanno la funzione di smistare il traffico
- Un network è detto mobile se cui alcuni nodi (endpoint e switch) cambiano ubicazione relativa nel tempo durante il funzionamento

Mobile Communication Networks

- Un routing system è un insieme di componenti atte a
  - monitorare la topologia della rete (network) e i servizi
  - distribuire l'informazione necessaria per costruire i percorsi (route)
  - localizzare gli end-point di una sessione
  - smistare il traffico in base ai percorsi (route) stabiliti

Mobile Communication Networks

- In funzione del grado di mobilità dei suoi componenti una rete si divide in
  - ***Cablata (Wireline)*** - tutte le componenti sono fisse (rete wired tradizionale) Es. la LAN del Dipartimento
  - ***Cellulare*** - solo gli end point sono mobili, mentre le switch sono fisse
  - ***Satellitare (Satellite network)*** quando le switch sono satelliti orbitanti intorno alla terra
  - ***Pocket radio (ad hoc)*** - Sia gli end point che le switch sono mobili

Mobile Communication Networks

- Network cellulari
  - ***Le switch sono fisse e chiamate anche base station*** alcune sono collegate alla rete in modo wired altre in modo wireless
  - ***Gli end point sono mobili*** - solo ripartiti in ***celle*** ciascuna associata ad una base station (una base station può gestire una o più celle). Questa organizzazione favorisce il riuso delle frequenze di trasmissione in celle geograficamente distanti
  - Appartengono a questa categoria le ***reti telefoniche cellulari (AMPS, PCS, GSM)*** gli standard di telefonia cordless, le WLAN IEEE 802.11

Mobile Communication Networks

- Network cellulari - problemi
  - ***Le dimensione delle celle -***
    - i sistemi ***microcellulari*** hanno celle con raggio da 50m a 500m e costituiscono sistemi di massima capacità ma con maggior costo di handoff
    - handoff (o handover) - l'operazione di aggiornamento della configurazione dovuta al cambiamento di posizione (cella) di un endpoint. Il cambio di cella è provocato dal cambio posizione, l'appartenenza a una cella, detto *affiliazione,* viene formalizzato mediante un processo di *registrazione*.
  - ***Assegnamento delle celle*** ai ***Mobile Switching Centers (MSCs)*** - un MCS è un'entità associata ad una porzione fissata del network

Mobile Communication Networks

- Satellite Network
  - *Le switch units* sono imbarcate su satelliti orbitanti. I link sono terra-satellite o inter-satellitari. Hanno problemi simili ai sistemi cellulari. Sono di due tipi:
    - un piccolo numero di satelliti in orbita geostazionaria offre una buona copertura del terreno tali da raggiungere quasi ogni utente con un singolo ***hop***. Utenti non coperti dallo stesso satellite iteragiscono su più hop terra-cielo o usando link intersatellitari
    - ***Low hearth Orbit*** (LEO) network composti da molti satelliti (molto costosi es. ***Iridium***) Benché dotati di topologia dinamica non richiedono meccanismi di *traking* a topologia adattativa.

Mobile Communication Networks

- Pocket Radio Network
  - I sistemi di comunicazione militari richiedono tecniche di networking di tipo adattivo in modo da sopravvivere in ambienti ostili. Esempi: *DARPA PRNET SURAN.*
  - Solo di recente si è visto un interesse per l'uso anche in applicazioni civili. Questi network vengono chiamati *ad hoc* dal comitato dello standard IEEE 802.11 - si chiamano anche i *packet radio network*.
  - La mobilità delle switch porta alcuni problemi di organizzazione diversi da quelli dei network cellulari

Mobile Communication Networks

- Pocket Radio Network cnt.
  - La necessità di risposte rapide al movimento degli switch richiede meccanismi organizzativi automatici operanti con interventi manuali minimi.
  - Il problema di progetto principale consiste nell'organizzare le switch in gruppi: problema motivato da
    - esigenza di controllare il *riuso spaziale* dei canali (in termini di frequenza, tempo, *spreading code...*)
    - ridurre il tempo di overhead del routing

Mobile Communication Networks

- Pocket Radio Network - Tecniche di clustering delle switch per realizzare il controllo dei canali di comunicazione
  - ***Algoritmi principali***
    - Un primo algoritmo, di tipo distribuito,fa usa di ***clusterheads.*** Ogni nodo ha un proprio ***ID*** numerico. Il nodo con ID minimo in un certo intorno viene scelto come clusterhead locale e delegato a controllare l'accesso al canale
    - Un secondo algoritmo costruisce cluster senza scegliere un clusterhead: ogni cluster ha un diametro di al più due hop. I cluster vengono costruiti iterativamente partendo dal nodo di massimo grado per formare il primo cluster e iterando il metodo nel parte restante del network

Mobile Communication Networks

- Pocket Radio Network - Tecniche di clustering gerarchico
  - Si applica a grandi network (DARPA) per ridurre lo spazio occupato infatti la quantità di informazione richiesta è dell'ordine O(n^2) mentre la capacità del network cresce solo linearmente con n.

Mobile Communication Networks

- Location tracking
  - Ogni network deve avere informazioni sulla posizione corrente di ogni endpoint. Lo scopo è quello di smistare il traffico di comunicazione ai destinatari interessati: il problema è detto location tracking o mobility tracking o mobility management
  - Di solito si mantengono due informazioni distinte l'ID di un endpoint e il suo indirizzo (che specifica dove è situato).
  - Molti meccanismi per il tracking della locazione si basano su un mapping, dipendente dal tempo, dell'ID sull'indirizzo attuale di ogni endpoint

Mobile Communication Networks

- Location tracking cont.
  - Molti metodi sono simili ad aggiornamenti/ interrogazioni applicate a un database distribuito (detto location database). Il problema si divide in due sotto-problemi:
    - *stabilire quando e come cambiare una entry nel database*
    - *organizzare e mantenere il database stesso*

Mobile Communication Networks

- Updating the location database
  - Essendo la mobilità all'interno di di una cella trasparente al network - l'operazione è necessaria solo quando un endpoint cambia cella. Si effettuano due operazioni
    - *updating (detta **registration**) l'azione con cui un endpoint mobile inizia un cambio di locazione;*
    - *finding (detta **paging**) l'azione di attivazione di una ricerca di un endpoint*
    - *molti metodi usano una combinazione di entrambi - ad esempio gli update non vengono attivati ad ogni cambio di cella ma secondo una strategia predefinita*

Mobile Communication Networks

- Updating the location database: static strategies
  - In questo caso vi è un insieme di celle predefinito in cui avviene l'aggiornamento. Quando un endpoint entra in una di queste celle si può attivare un aggiornamento (non necessariamente ogni volta). Vi sono due metodi:
    - *Location areas (o paging o registration areas)* L'area di servizio è ripartita in gruppi di celle, ciascuno gruppo forma una a location area: la posizione di un endpoint è aggiornata solo quando esso cambia location area; usato da sistemi cellulari di seconda generazione (GSM, IS-41);
    - *Reporting cells (o reporting centers);* in questo metodo un sotto-insieme di celle da cui poter aggiornare la posizione di un endpoint viene definito staticamente. La ricerca di un endpoint si effettua nelle vicinanze dalla *reporting cell* da cui è stato attivato l'aggiornamento più recente (ultimo indirizzo conosciuto)

Mobile Communication Networks

- Updating the location database: dynamic strategies
  - In questo caso è un endpoint a decidere quando aggiornare il database in base al proprio stato di movimento
  - tale aggiornamento è eseguibile da una cella arbitraria.
  - In prima approssimazione si tende ad estendere le strategie statiche aggiungendo informazioni "che rappresentano il movimento".

Mobile Communication Networks

- Organizzazione del *location database*
  - Si vogliono ottimizzare:
    - la quantità di memoria usata
    - il numero di messaggi necessari.
  - Sono due criteri contrastanti per cui si segue un compromesso tra essi e si tiene conto anche della semplicità di implementazione

Mobile Communication Networks

Organizzazione del *location database* cont.

- Il sistema più pratico usa un approccio centralizzato memorizzando tutti gli abbinamenti ID-to-address in un unico nodo centrale:
- Il metodo diviene rapidamente inutilizzabile al crescere del numero di endpoint. Il metodo non è fault-tolerant per la mancanza di replicazione
- Si preferisce quindi distribuire il database nel network

Mobile Communication Networks

- Organizzazione del *location database cont.*
  - Il metodo consiste nel partizionare il network e allocare una parte del database in ciascuna partizione
  - Il metodo è particolarmente adatto ai casi in cui ogni utente è registrato in un'area specifica detta home. Quando un endpoint si sposta si aggiorna il relativo home database per riflettere il cambiamento

Mobile Communication Networks

- Organizzazione del *location database* - esempi
  - Nei network PCS (cellulari) si usano due registri:
    - Home Location Register (HLR)
    - Visitor Location Register (VLR)
  - Un metodo analogo viene usato nel Mobile IP scheme e nel Cellular Digital Packet Data (CDPD)
  - I Metodi gerarchici adatti a problemi di grande dimensione hanno successo anche in questo caso

Mobile Communication Networks

- Location tracking nei PCS
  - Nei network PCS si hanno due standard:
    - Nord-americano IS-41
    - Europeo standard GSM
  - Entrambi usano partizionare l'area di servizio in location area e si basano su una gerarchia a due livelli
    - ogni utente ha una entry nel HLR
    - quando si muove ottiene un record temporaneo nel VLR che invia un messaggio di registrazione al HLR

Mobile Communication Networks

- Location tracking in Internet
  - E' stato introdotto lo standard Mobile IP
    - Ogni endpoint mobile ha un indirizzo IP permanente su un home network su cui ha anche un home agent che opera come gestore di mobilità e che gestisce l'indirizzo corrente (care-of address) del nodo mobile
    - Quando l' endpoint si muove usa un un indirizzo temporaneo di smistamento (care-of address) nel network "straniero" abbinato ad un foreign agent nel network visitato presso cui l' endpoint si registra. Il care-of address non è altro che l' IP del foreign agent.

Mobile Communication Networks

- Location tracking nei Pocket Radio Network
  - Se il network è flat (piatto) tutti i nodi sono visibili tra loro e non occorrono meccanismi di location tracking. Essendo non scalabili essi richiedono forme di clusterizzazione
    - ogni nodo ha un indirizzo gerarchico formato dagli ID dei cluster incapsulati che lo contengono
    - Vi sono Address Server (AS) in ogni cluster di livello zero che gestiscono gli indirizzi gerarchici usati per il routing.
    - Quando un nodo vuole comunicare con un altro nodo interroga AS del proprio cluster che può a sua volta rivolgersi ad altri AS

Mobile Communication Networks

Route selection and forwarding

- Sono tecniche che richiedono informazioni dettagliate su
  - Stato corrente del network (interconnettività dei nodi e qualità dei link)
  - la sessione (cioè rate di traffico, locazione degli endpoint..)
- Questo allo scopo di di dirigere il traffico lungo cammini consistenti con il servizio richiesto e le restrizioni imposte dal network
  - si possono avere re-routing di traffico molto frequenti

Mobile Communication Networks

- Route selection and forwarding cont.
  - Come nei network wired, il tipo di route selection e le procedure di smistamento dipendono dalla tecnologia di switching sottostante
    - Circuit-based
    - Packet based
    - In parte dipende anche dalla mobilità delle switch

Mobile Communication Networks

Route selection and forwarding cont.

- In molti network cellulari le route vengono calcolate off-line e le chiamate vengono smistate tramite circuiti impostati lungo tali route
- le procedure di handoff permettono ad una call di continuare anche quando un endpoint si sposta da una cella all'altra
- In molti packet radio network le route vengono calcolate dalle switch stesse e il traffico viene smistato hop-by-hop ad ogni switch lungo una route. Le switch modificano le route in base a mutamenti percepiti nel network

Mobile Communication Networks

- Route selection: infrastruttura stazionaria
  - Il network può essere wireline o wireless - esso connette un insieme di base station con interfacce radio tramite le quali gli endpoint accedono al network
    - network cellulari i telecomunicazione
    - network ATM
    - inter-network con endpoint mobili
  - Il moto degli endpoint attiva modifiche nel routing, ma le modifiche vengono calcolate nella infrastruttura statica

Mobile Communication Networks

- Route selection: network cellulari (CTN)
  - IS-41 e GSM hanno un comportamento simili: quando un endpoint migra (roams) in un'area di servizio esterna alla propria home, la sua posizione corrente nell' HLR deve essere aggiornata affinché esso continui a ricevere chiamate
    - Un MSC viene informato di ciò mediante una richiesta di registrazione
    - lo MSC notifica il VLR che a sua volta notifica l' HLR della nuova locazione dell' endpoint

Mobile Communication Networks

- Route selection: network cellulari (CTN)
  - quando un endpoint x vuole comunicare con un endpoint y la call e' smistata tramite il network fisso a un MSC nell'area home di y. L' MSC consulta l' HLR di y e scopre la locazione del VLR di y estendendo quindi la call setup ad esso
  - nel caso che la base station corrente di y sia sconosciuta il VLR inizia l'operazione di paging tramite un MSC locale che effettua un broadcast della pagina a tutte la base station dell'area. La base station interessata risponde e l'MSC può completare l'instaurazione della call

Mobile Communication Networks

- Route selection: network cellulari - handoff
  - ***quando y si muove cambiando base station deve affiliarsi rapidamente con una nuova base station (handoff). Vi sono 4 tipi di handoff***
  - ***Mobile-controlled***
  - ***Network-controlled***
  - ***Mobile-assisted***
  - ***Soft***

Mobile Communication Networks

- Mobile-controlled handoff
  - Gli endpoint mobili monitorizzano costantemente la qualità del segnale della base station e di quelle vicine e scegliendo come nuova base station quella che offre il segnale migliore (usando isteresi)
- Network-controlled handoff
  - è la base station a controllare la qualità del segnale degli endpoint. Quando esso scende sotto una certa soglia la base station invia una richiesta di handoff al MSC che chiede alle base station vicine di monitorare il segnale dell'endpoint e scegliendone una nuova tra quelle con buon livello di segnale

Mobile Communication Networks

- Mobile-assisted handoff (GSM, IS-41)
  - Sono le base station a chiedere agli endpoint mobili di monitorare costantemente la qualità del segnale ricevuto da un insieme specifico di base station vicine.
  - Le misure sono inviate alla base station che le smista al MSC. L'MSC stabilisce quando iniziare un handoff e quale base station diventerà la nuova base station di un endpoint.

Mobile Communication Networks

- Soft handoff (IS-95 digital CDMA)
  - Gli endpoint mobili possono affiliarsi simultaneamente a più base station dotate segnale di equivalente qualità.
  - In ricezione gli endpoint possono combinare i segnali ricevuti aumentando la probabilità di una decodifica positiva.
  - In trasmissione sono le base station che gestiscono una call di un endpoint che smistano il traffico da esso ricevuto estesa con un'informazione di qualità al MSC che seleziona lo stream di informazione di qualità migliore.

Implicazioni della Mobilità

- Riguardano aspetti generali e nuove prospettive di utilizzo di cui tener conto nello sviluppo di software. Tra essi:
  - Ubiquitous Computing
  - Location Awareness
  - Application Awareness
  - Radio Awareness

Ubiquitous Computing

- Prospettato come "terzo paradigma" dopo i mainframe ed i personal computers.
  - Può essere visto come l'opposto della ***realtà virtuale*** che mette l'utente a contatto con un mondo virtuale generato a computer
  - L' ***Ubiquitous Computing*** mette il computer in grado di operare nel mondo reale, in ogni situazione, all'aperto, in macchina, in fabbrica ecc.

Location Awareness

- Un sistema location-aware è un sistema che conosce la locazione di ogni unità mobile e che usa questa informazione per migliorare il proprio comportamento
- Livelli di astrazione
  - Location Transparent: l'impatto che la mobilità ha sulle applicazioni è completamente nascosto
  - Location-Tolerant: astrazione intermedia, che permette alle applicazioni di reagire a conseguenze della mobilità che non possono essere filtrate
  - Location Aware: il livello applicativo stesso conosce la propria ubicazione ed usa direttamente questa informazione

Radio Awareness

- Si basa sul fatto che il *Broadcasting* non ha costi extra in una WLAN ridotta, o ha costi ridotti in network radio di maggiori dimensioni. Questo fatto offre la possibilità ad ogni utente di essere sempre informato su eventi rilevanti del network, indipendentemente dal posto in cui sta operando.

Client-server in Mobile Environments

- Mobile Client-server Computing: Paradigmi
  - **Mobile-aware adaptation**
  - **Modello client-server esteso**
  - **Accesso ai dati in ambiente mobile**

Client-server in Mobile Environments

- Mobile-aware adaptation
  - il sistema client-server mobile deve essere in grado di reagire a cambiamenti rapidi nel network e nelle risorse locali durante l'accesso a dati remoti, onde permettere all'applicazione un funzionamento corretto
  - Il rango di adattabilità adottabili va da
    - laisse-faire adaptation : responsabilità dell'applicazione
    - application-transparent adaptation (del sistema)
    - application-aware adaptation (intermedia)

Client-server in Mobile Environments

- Application transparent adaptation
  - Usata da molte applicazioni client-server che sono state sviluppate assumendo che l'ambiente di una applicazione non cambi (mobility unaware)
  - Il metodo permette a queste applicazioni di operare anche in ambienti mobili. Il sistema nasconde le differenze tra ambiente stazionario ed ambiente mobile

Application Transparent Adaptation

- *File system Proxy il file system proxy nasconde la mobilità ed emula sui computer mobili i servizi del file server.*
  - *CODA file system: il proxy gestisce il funzionamento in modo disconnesso mediante logging delle operazioni eseguite durante la disconnessione.*

Application Transparent Adaptation

- File system Proxy (cnt)
  - Disconnected Operations:
    - il proxy gestisce il funzionamento in modo disconnesso operando un logging delle operazioni effettuate durante la disconnessione
    - Esse verranno rieseguite al momento della riconnessione.
  - Weakly Connected Operations:
    - il proxy pre-carica (pre-fetches) i dati dal server nella cache del client ed usa object or volume call-back per validarne il contenuto. Volume call-back è pessimistico

Application Transparent Adaptation

- File system Proxy (cnt)
  - Isolation-only Transaction:
    - Le operazioni eseguite in modo disconnesso possono portare ad inconsistenze sui dati dovute ad azioni di più computer operanti sugli stessi dati in modo disconnesso
    - la loro esecuzione è realizzata dal file system proxy che gestisce la consistenza in dipendenza dalle condizioni di connessione.
    - Al contrario delle transazioni tradizionali la failure atomicity non è garantita.

Application Transparent Adaptation

- Web Proxy (WebExpress) Vi sono due componenti tra un Web client e un Web server
  - The Client Side Interceptor (CSI)
  - The Server Side Interceptor (SSI)
  - Insieme eseguono ottimizzazioni per ridurre l'uso di banda e tempi di latenza
  - Caching:
    - sia CSI che SSI operano caching di grafica e oggetti HTML. Se una URL si riferisce ad un oggetto in cache esso viene smistato all'istante
    - la cache di SSI contiene le risposte del web server usato.

Application Transparent Adaptation

- Web Proxy (cnt)
  - Differencing:
    - Il concetto consiste nel caching di oggetti base comuni sia sul CSI che sul SSI. Al ricevimento di una risposta il SSI calcola le differenze tra l'oggetto base e la risposta e manda le differenze al CSI che ricostruisce per immersione il risultato
  - Protocol Reduction:
    - ogni CSI si connette al rispettivo SSI con una singola connessione TCP/IP eliminando costi di riconnessione e "multiplexando" richieste/risposte
  - Header Reduction:
    - HTTP è stateless: quindi ogni richiesta deve specificare le capability del browser.

Application Aware Adaptation

- permette alle applicazioni di reagire a cambiamenti di risorse mobili. Si realizza tramite collaborazione tra il sistema ed ogni applicazione
- Si divide in
  - Client-based Application Adaptation
  - Client-Server Application Adaptation
  - Proxy-based Application Adaptation

Application Aware Adaptation

- Client-based Application Adaptation
  - Solo le applicazioni in esecuzione sui clienti mobili reagiscono a cambiamenti nell'ambiente
- Client-Server Application Adaptation
  - sia le applicazioni sui clienti mobili che quelle sui server reagiscono a cambiamenti nell'ambiente
- Proxy-based Application Adaptation
  - Supporta adattamenti application-specific che avvengono solo sul server proxy del network fisso

Extended Client Server Model

- Si basa sull'esame dell'impatto che la mobilità ha sul modello client-server (modello client-server esteso)
  - La limitazione di risorse computazionali del client può richiedere di eseguire sul server operazioni usualmente eseguite sul client
  - Alternativamente, problemi di connessione possono richiedere al client di eseguire operazioni usualmente eseguite dal server
    - Un caso estremo viene chiamato thin client architecture
    - l'altro estremo full client architecture
    - Flexible client architecture

Thin Client Architecture

- Il sistema della Citrix corporation
  - Permette a diverse piattaforme remote di connettersi ad un terminal server Windows NT.
  - Un server chiamato Metaframe gira sotto WNT su un desktop e comunica con i thin client mediante un protocollo proprietario detto Independent Computing Architecture protocol (ICA)
  - Metaframe esegue tutte le operazioni in modo remoto i client hanno solo la funzione di user interface

Thin Client Architecture

- Il sistema della Citrix corporation è stato sperimentato in ambiente wireless dalla Motorola

- Il risultato di questa ricerca ha mostrato che i limiti di banda non riducono le performance dell'architettura poiché i clienti fanno un uso limitato della banda

Full Client Architecture

- I clienti mobili spesso si trovano in stato di "weak connectivity " a causa di
  - bassa bandwidth
  - intermittenza delle comunicazioni e alta latenza
  - nei casi estremi i clienti devono operare in disconnected mode
- La capacità di operare in modo disconnesso offre vantaggi anche quando la connessione è disponibile

Full Client Architecture

- Il modo disconnesso ha i seguenti vantaggi
  - risparmio della carica delle batterie
  - riduzione del carico sulla rete
  - in applicazioni militari permette il silenzio readio
- L' architettura full client permette di lavorare in modo disconnesso o debolmente connesso
- L'emulazione è eseguibile tramite un lightweight server risiedente sul client

Flexible Client Architecture

- Generalizza entrambe le architetture thin e full client rilocando dinamicamente i ruoli di client e server eseguendo l'applicazione su uno o sull'altro
  - Mobile objects (o mobile agents) sono codice che viaggia liberamente sul network . Essi permettono di eseguire le funzionalità richieste sia su host mobili che stazionari.
  - Collaborative Groups: formati da un gruppo di user disconnessi dal resto del network

Flexible Client Architecture

- Application specific proxy: intermediario tra clienti e server per eseguire operazioni compute intensive.
- Virtual mobility of servers: si replicano servizi sul network per migliorare l' handoff di clienti mobili

Mobile Data Access

- Supporta lo smistamento di dati dal server e il mantenimento della consistenza dei dati in ambienti mobili e wireless.
  - L'accesso ai dati mobili è caratterizzabile dal data delivery mode che può essere di tre tipi:
    - Server-push delivery
    - Client-pull delivery
    - Hybrid delivery

Mobile Data Access

- Server data dissemination:
  - Broadcast disks
  - Indexing on Air
- Client Cache management
  - Automated hoarding
  - varied granularity of cache coherence

Mobile Data Access

- Server data dissemination (Broadcast disk)
  - In molte applicazioni (Web) il volume di scambio dati dal server al client è molto maggiore del flusso contrario.
  - Inoltre il numero di clienti è molto elevato
  - La soluzione consiste nell' uso di broadcasting
    - Broadcast disk: quando il server smista i dati in continuazione ai clienti, il canale di broadcast viene assimilato ad un disco da cui i clienti ottendono i dati.

Mobile Data Access

- Server data dissemination (Indexing on Air)
  - Nel metodo push-based i server periodicamente smistano i dati più richiesti, in broadcast ai clienti.
  - Il server può modificare dinamicamente il contenuto dello hot spot.
  - Il cliente è lazy (trasmette solo quando necessario). Gli indici sono strutturati per minimizzare:
  - Il query time: tempo tra issue-of-a-query e ricevimento della risposta
  - Listening time: tempo di ascolto del canale

Mobile Data Access

- Client cache management: si basa sul caching di dati acceduti frequentemente. Permette di gestire
  - disconnected mode e
  - intermitted connected mode
- la connettività debole rende costoso il mantenimeno coerente del contenuto della cache
- Il termine Hoarding indica tecniche di pre-fetching dei dati prima della disconnessione.

Mobile Data Access

- Automated Hoarding.
  - Caching sul client prima di disconnessione
- Varied granularity of Cache coherence
  - estensione di metodi client/server tradizionali ad ambienti mobile
- Cache invalidation reports
  - approccio all'invalidazione della cache basato sulla dissemination (Barbara & Imielinski). Periodicamente il server invia un report di invalidazione

Mobile Data Access

- Automated Hoarding. I files non locali vengono 'cached' sulla cache del client prima della disconnessione. Vi sono tre tecniche basate sulla misura della distanza semantica tra i files.

- Varied granularity of Cache coherence: estensione di metodi di consistenza in applicazioni client-server tradizionali:
  - callback approach: i server inviano messagi di invalidation
  - detection approach: i client inviano query di validazione

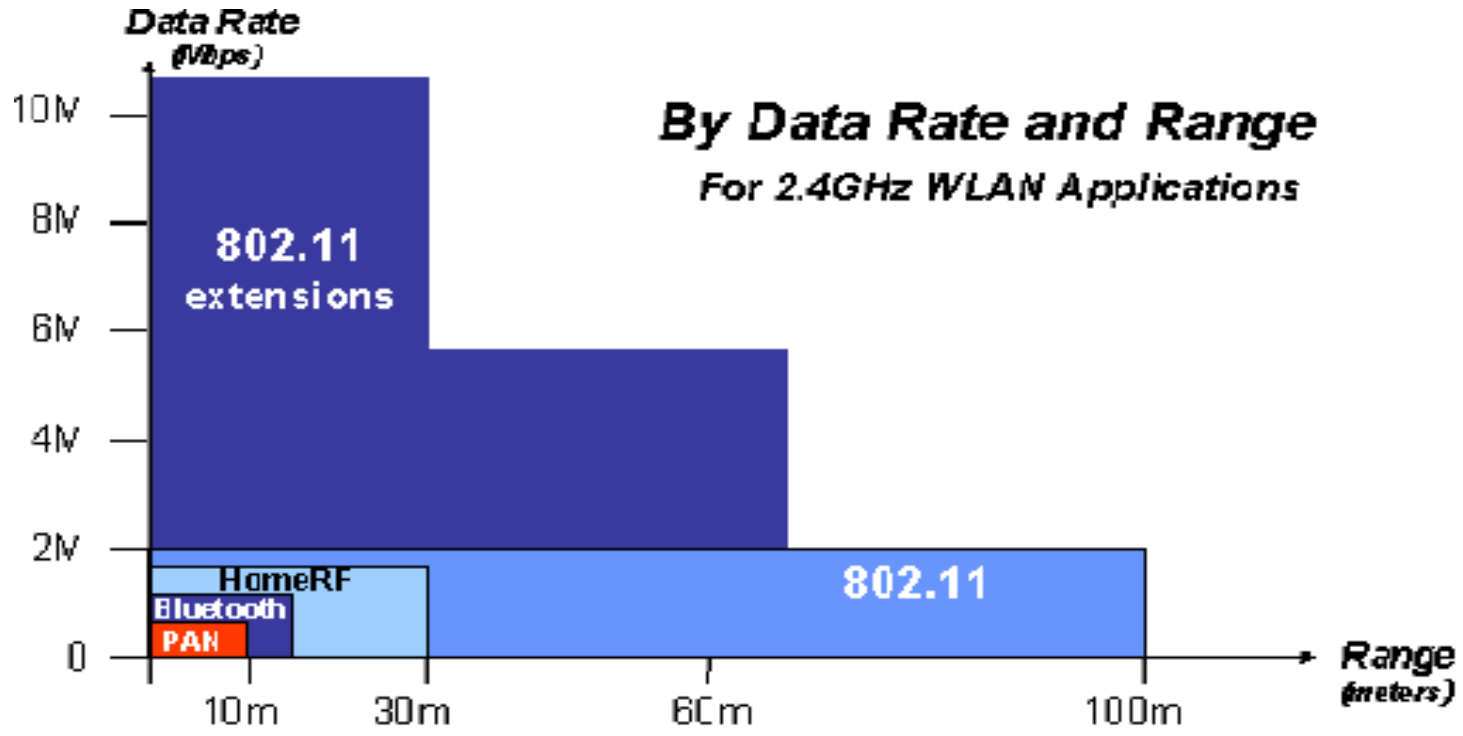
Mobile Data Access

- Cache invalidation reports:
  - Invece di richiedere al server la validazione di copie cached i clienti si pongono in ascolto di report di invalidazione in arrivo dai server. Tre metodi
    - TS - broadcasting Time Stamps
    - AT - Amnesic Terminals
    - SIG - SIGnatures

Wireless LAN

- Vi sono diverse proposte Standard. Le principali sono
  - IEEE-802.11
  - Bluetooth
  - Home RF
  - PAN

Wireless LAN



Lo Standard IEEE 802.11

- Lo Spettro Radio
  - Lo spettro delle radio-frequenze (RF) e` diviso in due tipi di bande: assegnate e libere.
    - le prime sono dedicate a grandi organizzazioni,
    - le seconde sono soggette a regole, ma prive di restrizioni e tasse

Lo Standard IEEE 802.11

- Basato su un metodo che permette un buon impiego di ampiezze di banda limitata detto *Spread Spectrum* (SS)
  - esso garantisce comunicazioni affidabili anche in ambienti a *rumore* elevato

Lo Standard IEEE 802.11

- Molti network wireless attuali seguono le specifiche dello standard IEEE 802.11 per reti WLAN multi-hopping operanti nella banda di 2.4GHz.

- Lo standard europeo più vicino è lo ETSI (European Telecomm. Standard Institute)

- I metodi di uso uniforme del canale classici:
  - Time Division Multiple Access (TDMA)
  - Frequency division Multiple Access (FDMA)
  - Code Division Multiple Access (CDMA)

  non fanno un uso efficiente della bandwidth di canale poiché richiedono allocazioni on demand

Lo Standard IEEE 802.11

- la tecnologia SS permette la massima data rate su un canale a banda limitata.
- Vi sono due tecniche implementative
  - **F**requency **H**opping **S**pread **S**pectrum (**FHSS**)
  - **D**irect **S**equence **S**pread **S**pectrum (**DSSS**) (chiamata anche *Direct-Sequence Modulation* DSM)

IEEE 802.11 - FHSS

- Il metodo FHSS usa diversi canali di frequenza
  - Il trasmettitore invia un segnale su un canale per un breve periodo, *saltando (hopping)* poi in modo pseudo-random, su un altro canale di frequenza.
  - Il ricevitore deve seguire la stessa sequenza pseudo-random ripetendo gli stessi salti del trasmettitore.

IEEE 802.11 - FHSS

- Vantaggi:
  - non richiede una banda contigua
  - semplice ed economico da realizzare

- Svantaggi:
  - rende l' *handower* tra celle contigue difficile da coordinare

IEEE 802.11 - DSSS

- Nel metodo DSSS, il trasmettitore sostituisce gli zeri e gli uno di un segnale digitale in banda base da trasmettere, con codici di lunghezza fissa, producendo un allargamento (spread) di banda di una opportuna quantita'
- Il ricevitore usa lo stesso codice del trasmettitore e mediante correlazione ricostruisce il segnale trasmesso.

IEEE 802.11 - DSSS

- Vantaggi :
  - miglior copertura a parita` di potenza irradiata (piu' del 50% di FHSS)
  - miglior qualita` del segnale
  - handower piu' facile
- Svantaggi:
  - richiede una banda contigua

IEEE 802.11 - DSSS

- Network Multihop
  - La configurazione più semplice di un network wirless è composta da una unità mobile connessa ad una stazione di supporto a sua volta connessa ad una rete wired. Questa semplice configurazione è detta **single hop**.
  - La tecnologia **multi-hop** permette ad una radio di smistare pacchetti ad altre radio situate sul percorso verso la loro destinazione.
  - La tecnologia multi-hopping permette di limitare la potenza di emissione delle radio limitando il consumo delle batterie.

IEEE 802.11 - DSSS

- Architettura di un Wireless LAN multi hop
  - Le componenti generiche di una LAN radio sono:
    - Radio Modules
    - Access Points
    - Bridges
    - Network Adapters
  - Un network tipico è composto da
    - access point / repeater
    - radio node / radio end node

IEEE 802.11 - DSSS