

# *Fundamentos de Sistemas de Operação*

*Unix*   *Windows NT*   *Netware*   *MacOS*   *DOS/VIS*   *Vax/VMS*  
*Linux*   *Solaris*   *HP/UX*   *AIX*   *Mach*  
Chorus

*Sistemas distribuídos:*  
*NFS - um SF cliente/servidor*

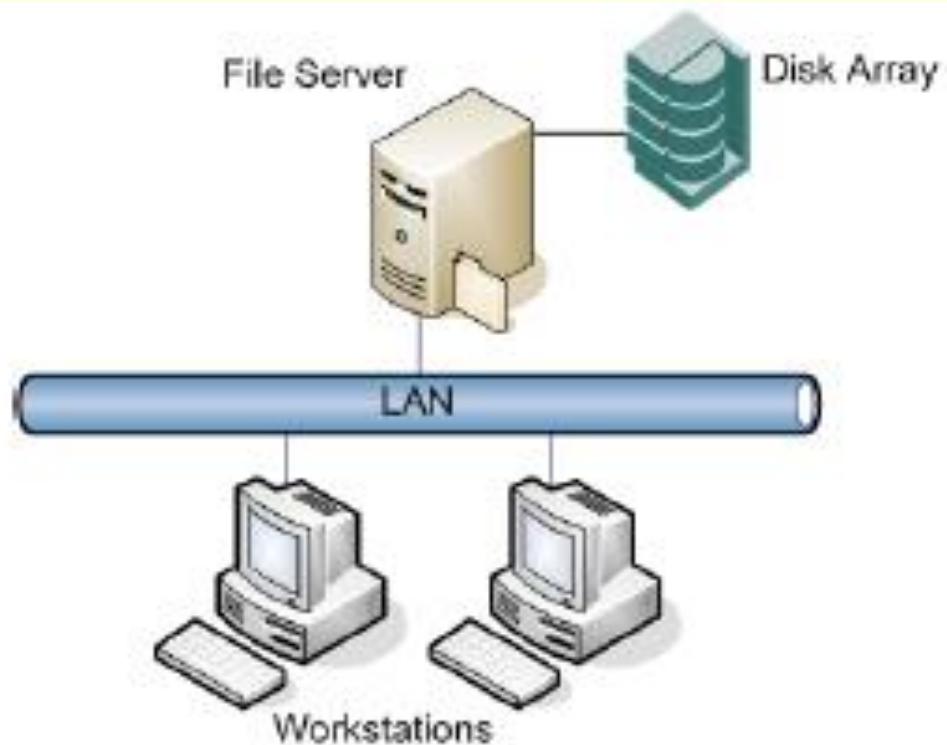
# *Sistemas de Ficheiros Distribuídos*

Porquê? – e.g. aqui, em que os utilizadores querem partilhar ficheiros

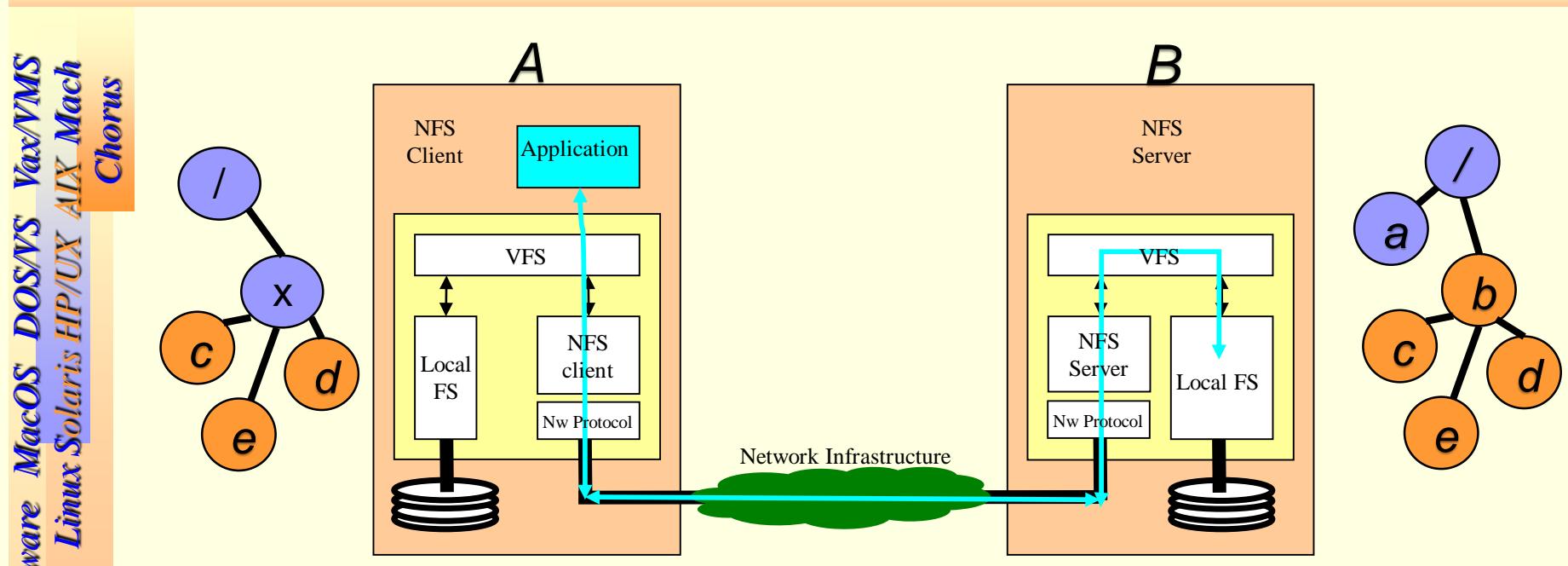
Um SF cliente/servidor é um tipo de SF distribuído que tem:

- um componente do lado do (SF do) servidor, e
- outro no lado do (SF do) cliente

Para um utilizador, a porção cliente do SF é similar a qualquer outro SF local, mas há aspectos a considerar, particularmente na semântica de partilha de ficheiros...



# Um SF cliente/servidor: NFS (1)



Os dois computadores correm Linux. O **A** tem um disco local do qual vemos as directorias / e x, existentes no disco; o **B** tem um disco local do qual vemos as directorias /,a,b,c,d,e, existentes no disco local. O sistema **B** **exporta** a árvore (b,c,d,e) que é montada pelo **A** “abaixo” da directoria x: assim, A vê a árvore (b,c,d,e) como se fossem parte do “seu disco”...

# *Um SF cliente/servidor: NFS (2)*

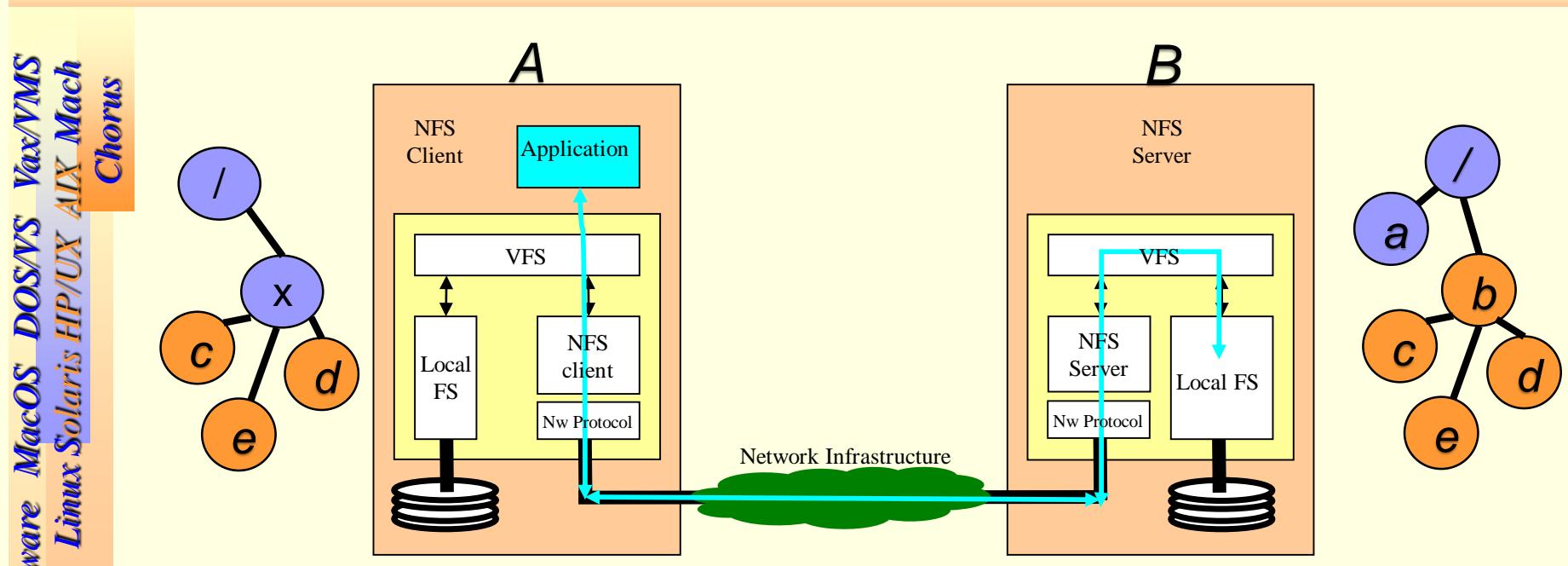
□ *Como é que um utilizador (root) faz “acontecer”?*

- Assume-se que:
  - A ordem para exportar a árvore “abaixo” de **b** já foi dada em **B**
  - Em **A** existe uma directória **x** vazia (de preferência)
  - O IP de **A** é 10.10.10.1 e o de **B** 10.10.10.2

1. *Fazer login em **A** (como root)*

```
# mount -t nfs 10.10.10.2:/b /x
```

# Um SF cliente/servidor: NFS (3)



Imagine-se agora que um programa em **A** faz `open("/x/c/ficheiro", ...)`. Que acontece? 1) Enquanto percorre o caminho, o SO descobre que abaixo de x é um SF remote, pelo que chama o modulo “NFS client” para continuar a percorrer o caminho abaixo de x. 2) O “NFS client” chama uma função de “NFS server” para fazer esse trabalho em **B**; 3) ...

# *O protocolo NFSv2*

- Para a versão 2 do Network File System (NFSv2), a empresa que o “inventou”, Sun Microsystems, definiu um protocolo.
- O protocolo define:
  - Que funções existem para serem chamadas pelo(s) clientes, e o que fazem
  - Para cada uma, quais são os parâmetros e resultados (incluindo erros)
- Funções: (acrescentar **NFSPROC\_** antes do nome abaixo indicado)
  - LOOKUP, REaddir, MKDIR, RMDIR
  - CREATE, REMOVE, READ, WRITE
  - GETATTR, SETATTR
  - Note-se: **NÃO existe OPEN ou CLOSE**

# *O protocolo NFSv2: open*

- Imagine-se o cenário (slide 5) em que a aplicação em **A** faz `open("/x/c/ficheiro", ...)`. Que acontece?
  - Quando o Linux em **A** “descobre” que “/x” é um mountpoint NFS, chama uma função do módulo “NFS cliente” para que encontre o “resto do caminho”, i.e., “/c/ficheiro”
  - Essa função faz: `fh= NFSPROC_LOOKUP("/c/ficheiro");`
  - A string “/c/ficheiro” e a informação para se chamar a função `NFSPROC_LOOKUP` é passada via rede para **B**
- Em **B** a função é executada e,
  - as instruções equivalentes a um `stat("c/ficheiro", ...)`, agora executado em **B**, são efectuadas. Se correr bem, é devolvido a **A** um “file handle”, objecto que tem as informações para acesso ao ficheiro (`/c/ficheiro`); o handle aparece à aplicação como um descriptor de ficheiro (`fd`) aberto

# *O protocolo NFSv2: read*

- Imagine-se a continuação do cenário em que agora a aplicação em **A** faz `read(fd,...)` ao ficheiro. Que acontece?
  - Quando o Linux vai executar o `read()` “descobre” que tem de chamar `rc= NFSPROC_READ(fh,buf,offset,len);`
  - De novo a informação para se chamar a função `NFSPROC_READ` é passada via rede para **B**
- Em **B** a função é executada e,
  - as instruções equivalentes a um `read(fd,...)`, agora executado em **B**, são efectuadas. Se correr bem, são devolvidos a **A** um “file handle” com o offset actualizado e um buffer com os dados lidos do ficheiro.

# O protocolo NFSv2: exemplo (1)

Client	Server
<code>fd = open("/foo", ...);</code> Send LOOKUP (rootdir FH, "foo")	Receive LOOKUP request look for "foo" in root dir return foo's FH + attributes
Receive LOOKUP reply allocate file desc in open file table store foo's FH in table store current file position (0) return file descriptor to application	
<code>read(fd, buffer, MAX);</code> Index into open file table with fd get NFS file handle (FH) use current file position as offset Send READ (FH, offset=0, count=MAX)	Receive READ request use FH to get volume/inode num read inode from disk (or cache) compute block location (using offset) read data from disk (or cache) return data to client
Receive READ reply update file position (+bytes read) set current file position = MAX return data/error code to app	

# O protocolo NFSv2: exemplo (2)

Client	Server
<b>read(fd, buffer, MAX);</b> Same except offset=MAX and set current file position = 2*MAX	
<b>read(fd, buffer, MAX);</b> Same except offset=2*MAX and set current file position = 3*MAX	
<b>close(fd);</b> Just need to clean up local structures Free descriptor "fd" in open file table (No need to talk to server)	

Figure 49.5: Reading A File: Client-side And File Server Actions

# O protocolo NFSv2: falhas

- *Read em cliente:*
  1. É possível saber qual das 3 falhas aconteceu?
  2. Como recuperar?

R1: Não

R2: Sim, basta repetir o read

3. Porquê tão simples?

R: o servidor não guarda estado, este está no cliente

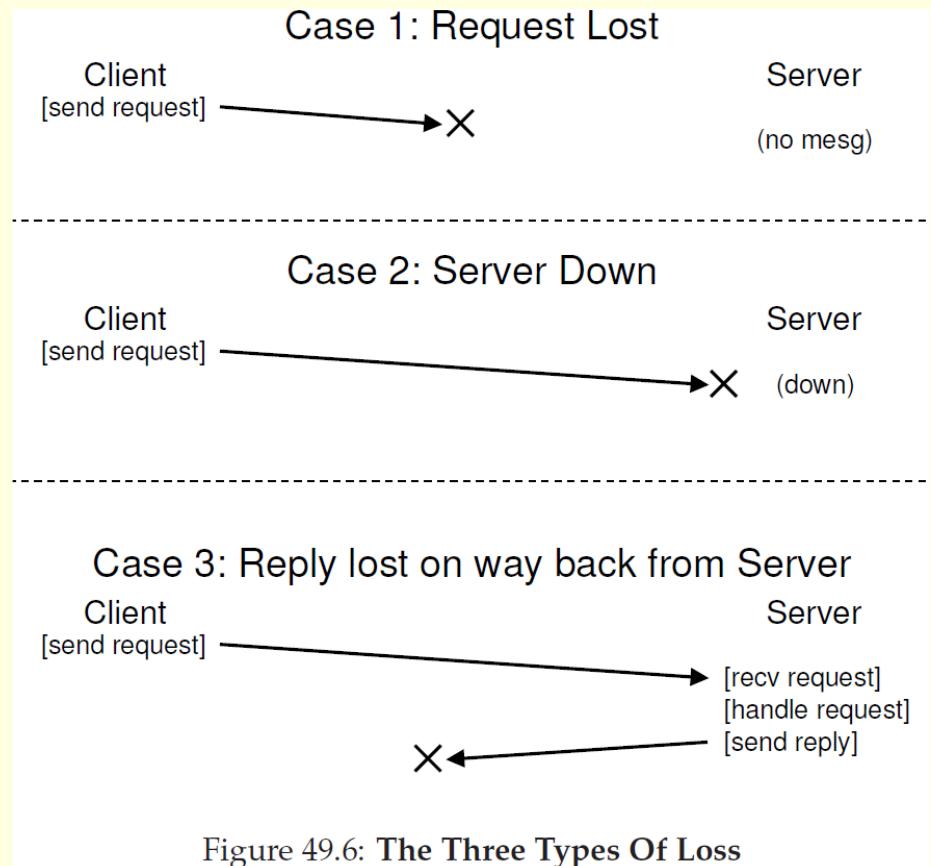
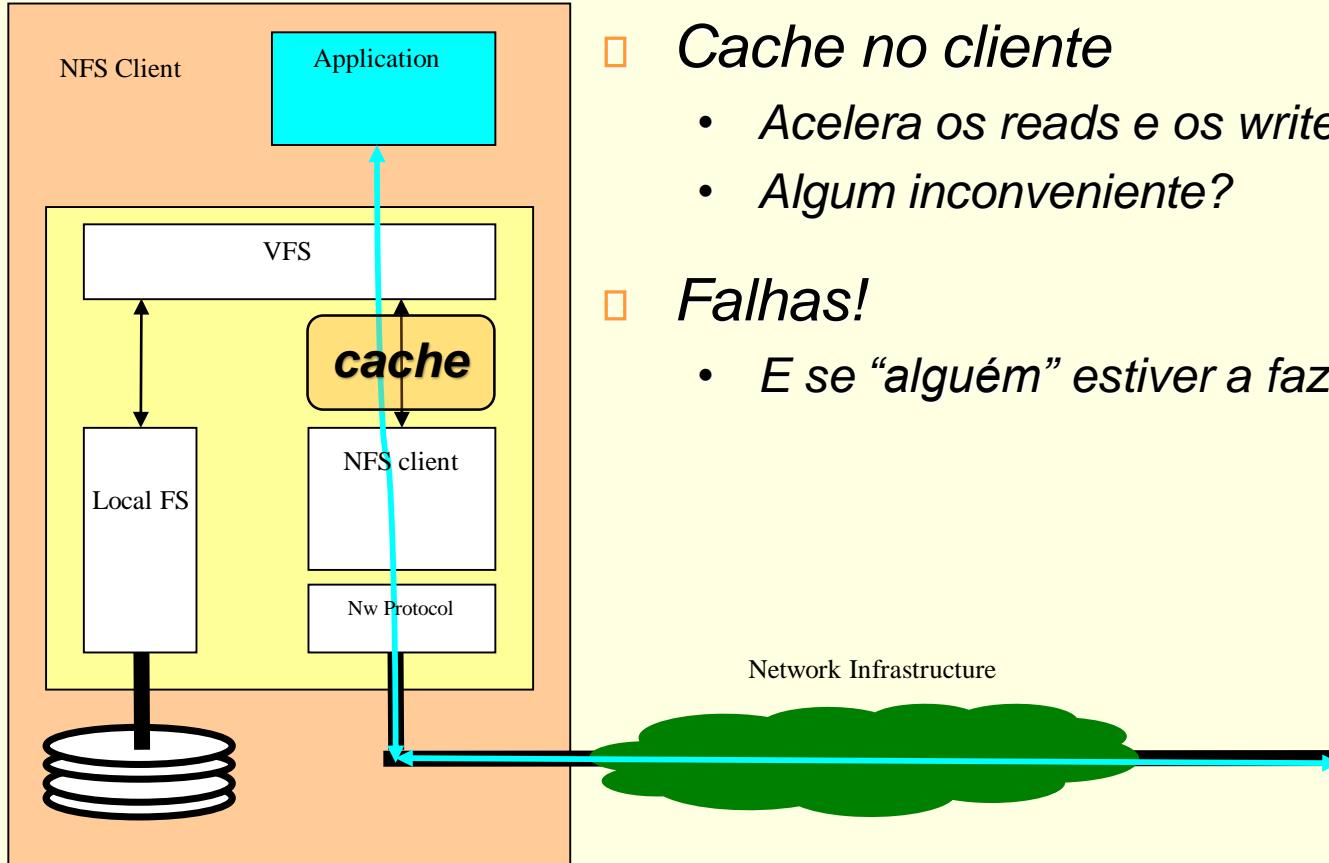


Figure 49.6: The Three Types Of Loss

# O protocolo NFSv2: caching (1)



## □ Cache no cliente

- Acelera os *reads* e os *writes*
- Algum *inconveniente*?

## □ Falhas!

- E se “*alguém*” estiver a fazer *writes*?

# *O protocolo NFSv2: caching (1)*

## □ Cache nos clientes

- C1 tem a “versão original” de F; C2 já mudou F, mas as modificações ainda estão em cache; C3 ainda não acedeu a F...

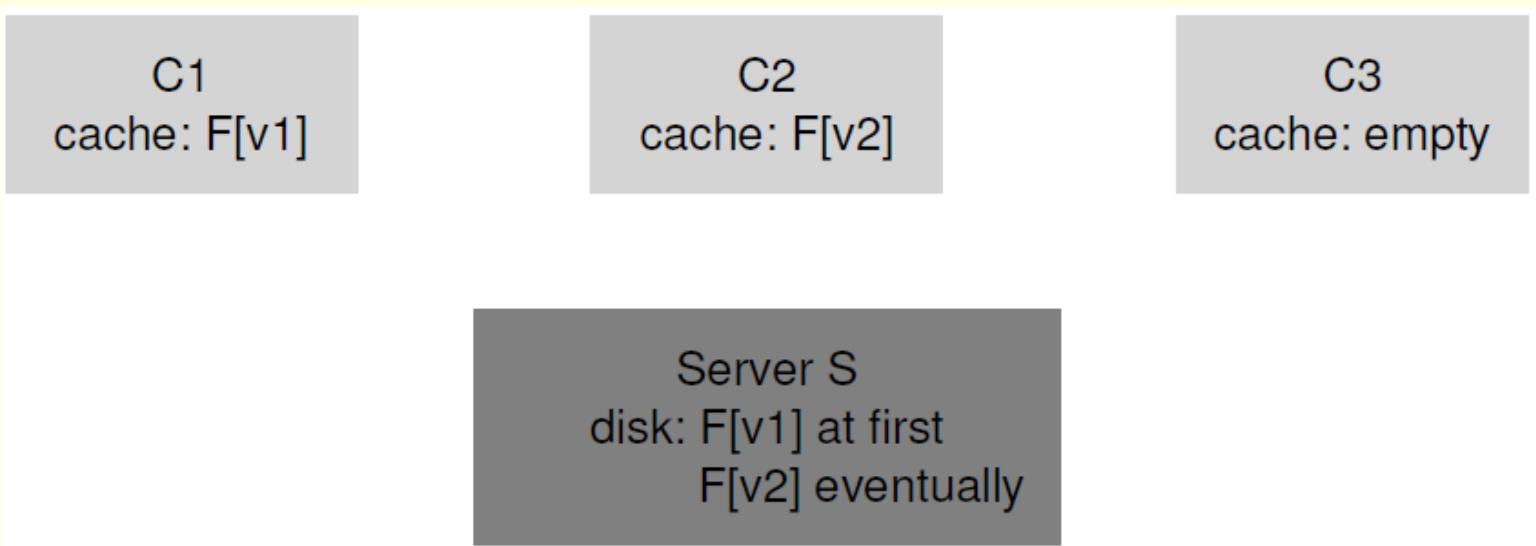


Figure 49.7: The Cache Consistency Problem

# *O protocolo NFSv2: caching (2)*

## □ As várias facetas do problema:

- *Visibilidade das actualizações: quando é que um “bloco” cached deve ser actualizado no servidor?*
- *Consistência: que fazer quando se actualiza no servidor mas há clientes com cópias (versões) antigas?*

## □ As soluções do NFSv2:

- *Visibilidade das actualizações: no close (flush-on-close)*
- *Consistência: antes de usar informação cached, verificar se ela está válida: como?*
  - Solução 1 (abandonada): executar GETATTR – problema demasiados GETATTR.
  - Solução 2: “Atribute cache” - de x em x seg. fazer GETATTR