UNIVERSIDADE DE ÉVORA

Sistemas de Ficheiros Distribuídos

Introdução: Armazenamento

	Sharing	*Persistence*	*Distributed cache/replicas*	*Consistency maintenance*	*Example*
Main memory	✗	✗	✗	1	RAM
File system	✗	✓	✗	1	UNIX file system
Distributed file system	✓	✓	✓	✓	Sun NFS
Web	✓	✓	✓	✗	Web server
Distributed shared memory	✓	✗	✓	✓	Ivy (DSM, Ch. 6)
Remote objects (RMI/ORB)	✓	✗	✗	1	CORBA
Persistent object store	✓	✓	✗	1	CORBA Persistent State Service
Peer-to-peer storage system	✓	✓	✓	2	OceanStore (Ch. 10)

ypes of consistency:
1: strict one-copy ✓: slightly weaker guarantees 2: considerably weaker guarantees

Introdução: SFD

partilha de informação previamente armazenada

- é um dos aspetos mais importantes da partilha de recursos a nível distribuído

Sistema de Ficheiros Distribuídos

- permite às aplicações gravar e aceder a ficheiros remotos exatamente do mesmo modo que utilizam para os ficheiros locais
- permite o acesso dos utilizadores aos seus ficheiros desde qualquer computador
-

Propósito (nos SFD elementares/básicos)

- emular a funcionalidade de um sistema de ficheiros (não distribuído) para aplicações cliente que podem estar em execução em vários computadores remotos

Características dos Sistemas de Ficheiros (SF)

um Sistema de Ficheiros (distribuído **ou não**) é responsável por:

- organização
- armazenamento
- acesso (*retrieval*)
- gestão de nomes
- partilha
- proteção/segurança dos ficheiros

SF fornecem uma interface de programação: a abstração ficheiro

- liberta os programadores de detalhes de armazenamento

Características dos Sistemas de Ficheiros

Módulos de um Sistema de Ficheiros (*não distribuído*)

- funciona por camadas, cada uma presta serviços à de cima

Directory module:	relates file names to file IDs
File module:	relates file IDs to particular files
Access control module:	checks permission for operation requested
File access module:	reads or writes file data or attributes
Block module:	accesses and allocates disk blocks
Device module:	disk I/O and buffering

Um SFD requer componentes adicionais para lidar com a comunicação entre cliente/servidor, nomes a nível distribuído e localização dos ficheiros

Sistemas de Ficheiros

- Armazenamento centralizado
  - Risco de congestionamento
  - Risco de Falhas
  - Tamanho de um ficheiro limitado à capacidade de um servidor
    - É o usual, com sistemas de ficheiros convencionais

- Armazenamento **paralelo**
  - Um ficheiro pode ser distribuído por diversas máquinas
  - Metadados e dados podem ser alojados em servidores diferentes
  - Ex: GFS

Ficheiros e Diretorias

Ficheiros

- podem ser armazenados de forma persistente em discos ou outros suportes não voláteis
- contém dados (sequências de bytes, 8 bits) e atributos (mantidos num registo, contém informação sobre o ficheiro – metadados)
- manipulados através da interface do serviço de ficheiros
  - OPERAÇÕES: criar, apagar, abrir, fechar, ler, escrever, append, seek, obter atributos, definir um atributo, renomear

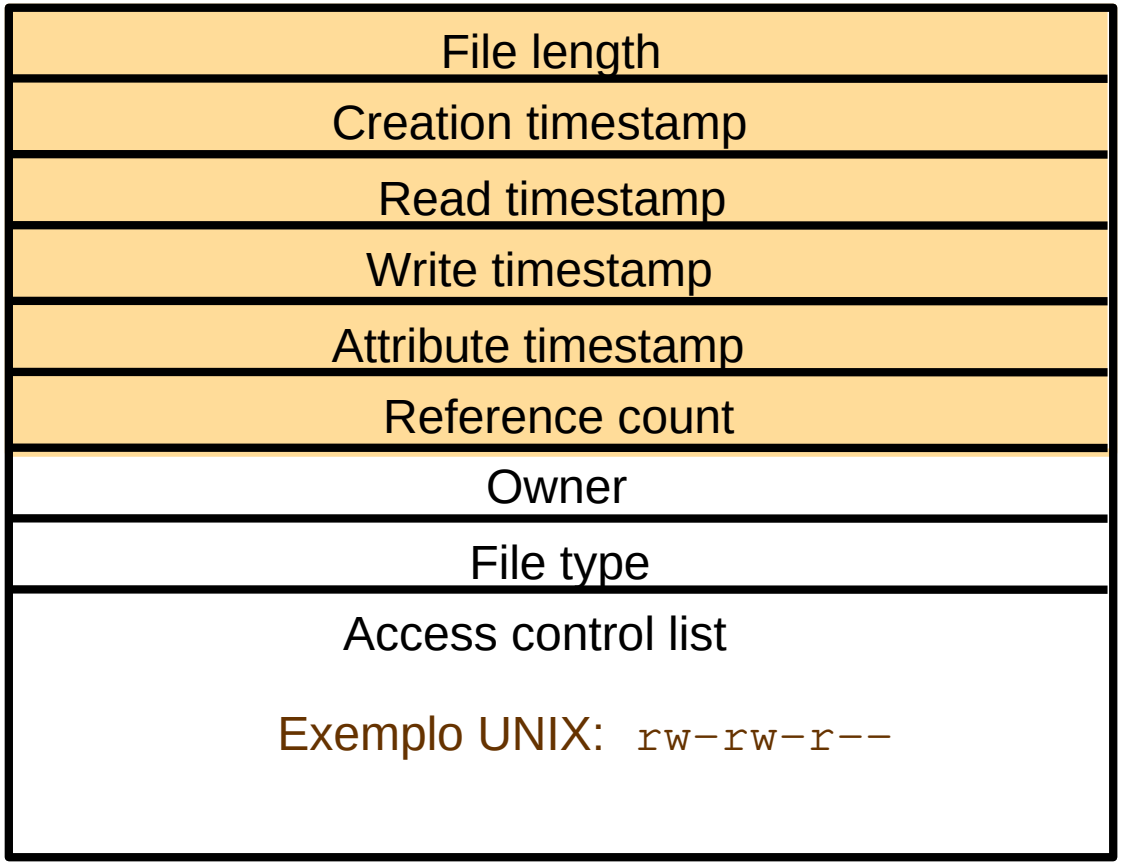
Um FS armazena um grande nº de ficheiros

A gestão dos nomes é facilitada com o uso de Diretorias

- uma diretoria é um ficheiro especial que mapeia um nome (texto) em identificadores internos para os ficheiros
- uma diretoria pode incluir o nome de outra diretoria, formando um esquema de nomes hierárquico

Atributos de um Ficheiro

Estrutura do registo dos atributos de um ficheiro:



Operações sobre o Sistema de Ficheiros UNIX

operações disponibilizadas pelo Kernel (system calls)

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

O Sistema de Ficheiros faz o controlo do acesso aos ficheiros

Requisitos de um Sistema de Ficheiros Distribuído

- Transparência
  - (acesso, localização, mobilidade, desempenho, escala)
- Controlo de concorrência
  - se duas aplicações escreverem no mesmo ficheiro em simultâneo?
- Replicação de ficheiros
- Abertura e independência face a diferenças de hardware ou SOs
- Tolerância a falhas
- Consistência
  - Se há réplicas, e uma recebe uma escrita, o que fazer à outra?
- Segurança
  - controlo de acessos, autenticação do cliente
- Eficiência

Sistemas de Ficheiros Distribuídos de referência (pré 2000)

Sun Network File System (NFS)

- 1985
- sucesso técnico e comercial
- o primeiro SFD disponível como um “produto acabado” (existiam outros com utilização limitada a alguns laboratórios)
- norma definida no RFC 1813
- a relação cliente/servidor é simétrica (cada computador ligado ao serviço pode atuar como cliente ou como servidor)
- independente do SO

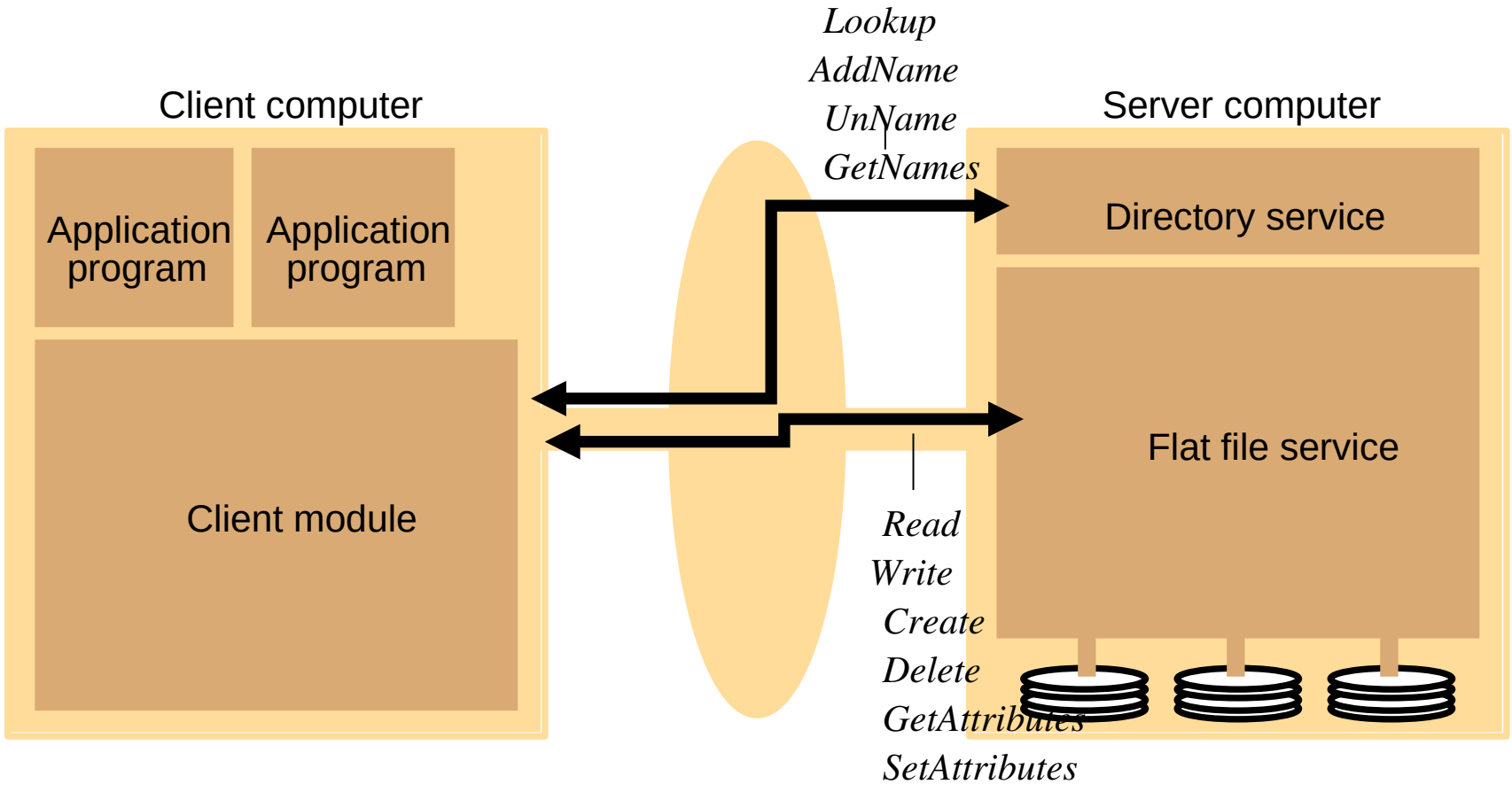
Andrew File System (AFS)

- final dos anos 80
- SF para o ambiente de computação distribuída Andrew, Carnegie Mellon U.
- suporte para partilha de larga escala – minimiza comunicação client/server
- em 1991 suportava 800 workstations, servidas por 40 servidores

Arquitetura de um Serviço de Ficheiros genérico

3 componentes

- serviço *flat file*, serviço de diretorias e módulo cliente



Componentes do Serviço de Ficheiros

Serviço *Flat File*

- operações sobre o conteúdo dos ficheiros
- pedidos referem ficheiros com identificadores únicos no SD (UFIDs)
- ao criar um ficheiro, é gerado um novo UFID

Serviço de Diretorias

- mapeia nomes (texto) de ficheiros em UFIDs
- criar diretorias, adicionar ficheiros a diretorias
- **cliente** do serviço *flat file*
- num esquema hierárquico (unix) as diretorias têm referências para outras diretorias

Módulo Cliente

- executado em cada computador cliente
- integra e estende as operações dos componentes anteriores, disponibiliza uma interface de programação aos programas (*user-level*)
- guarda informação sobre a localização na rede dos serviços de diretorias e *flat file*

Operações do Serviço *Flat File*

Read(FileId, i, n) -> Data — throws *BadPosition*	If $1 \leq i \leq Length(File)$: Reads a sequence of up to *n* items from a file starting at item *i* and returns it in *Data*.
Write(FileId, i, Data) — throws *BadPosition*	If $1 \leq i \leq Length(File)+1$: Writes a sequence of *Data* to a file, starting at item *i*, extending the file if necessary.
Create() -> FileId	Creates a new file of length 0 and delivers a UFID for it.
Delete(FileId)	Removes the file from the file store.
GetAttributes(FileId) -> Attr	Returns the file attributes for the file.
SetAttributes(FileId, Attr)	Sets the file attributes (only those attributes that are not shaded in).

Controlo de Acessos:

- Sistema de Ficheiros UNIX: os direitos do utilizador são validados com o modo de acesso especificado na operação *open,* com base no uid local
- SFD: verificação dos direitos de acesso efetua-se **no servidor**
  - cada pedido inclui a identificação do utilizador

Operações do Serviço de Diretorias

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

Sun NFS

protocolo NFS

- RPCs que permitem aos clientes trabalhar com ficheiros remotos
- independente de S.Op., mas inicialmente pensado para sistemas com UNIX

Servidor NFS

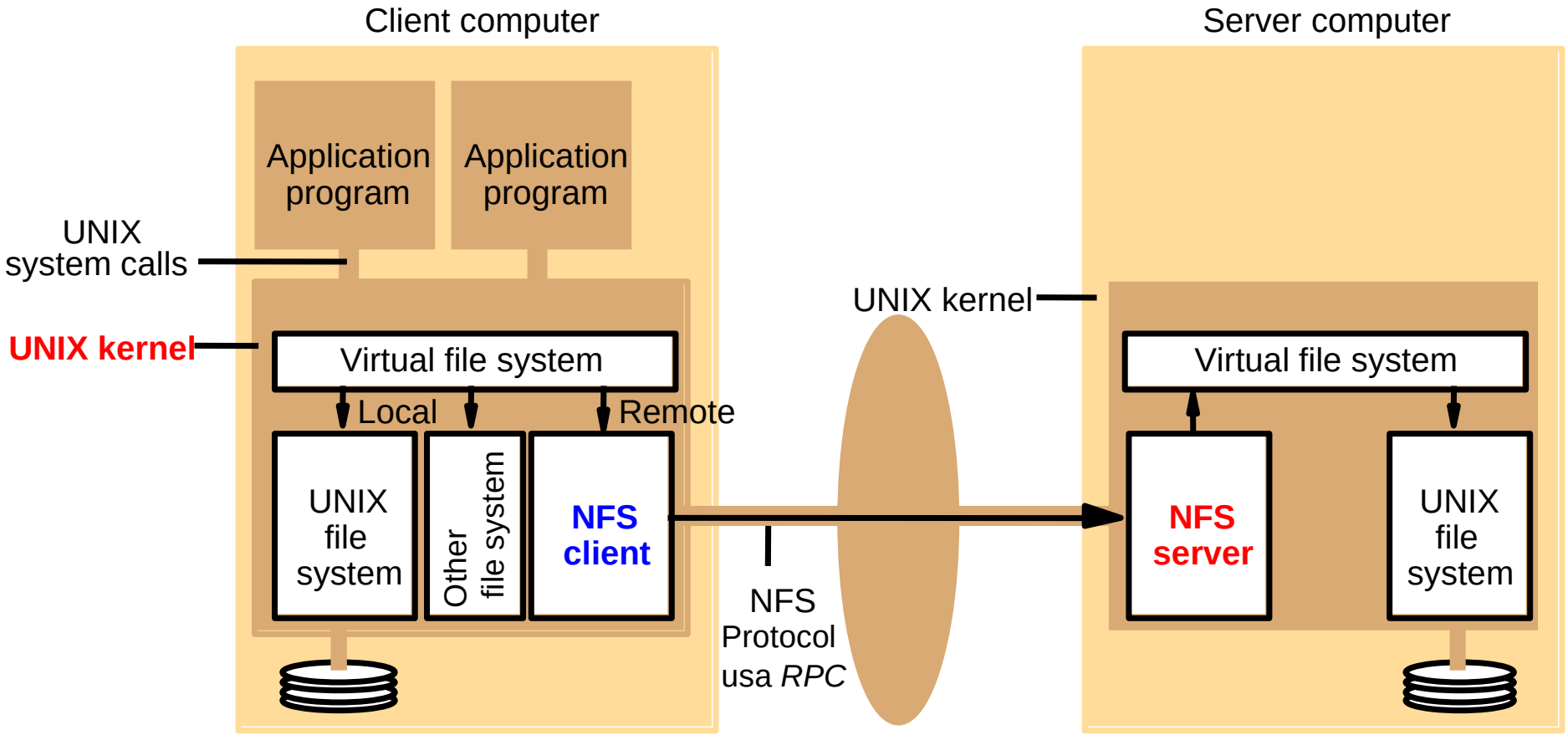
- ao **nível do Kernel** de um computador que age como servidor NFS

Os pedidos sobre ficheiros remotos são traduzidos pelo módulo cliente em operações do protocolo NFS e passados ao servidor que detém esses ficheiros

Podem usar-se/exigir-se credenciais de identificação

- medida opcional de segurança

Arquitetura NFS



Sun NFS: VFS

Sistema de Ficheiros Virtual (VFS)

Faz a integração entre o sistema de ficheiros local e o remoto

- encaminha cada pedido para o destino apropriado (NFS Client ou SF local)

Em NFS os identificadores dos ficheiros são *file handles*

- contém a informação que servidor precisa para distinguir os ficheiros

VFS contem um *v-node* por ficheiro aberto

- permite saber se o mesmo é local ou remoto

Cliente NFS é integrado no Kernel *(não tem de ser carregado como biblioteca)*

- permite o acesso aos ficheiros através de system calls
  - Coopera com o VFS passando os dados remotos "tal como" o UNIX-FS faz para os dados/ficheiros locais
- um único módulo cliente serve todos os processos user-level, com uma cache dos blocos em uso
- a chave usada na encritação do *uid* fica protegida a nível do kernel, a salvo das aplicações *user-level*

NFS: controlo de acesso e autenticação

- o servidor é ***stateless*** (vantagem para a consistência)
- servidor tem de validar a identidade do utilizador junto dos atributos de acesso do ficheiro a cada pedido
- os clientes enviam informação sobre a autenticação do utilizador, a cada pedido, em campos próprios nas RPCs
  - em UNIX: 16 bits com uid e gid
- Tem um problema de segurança na versão básica
  - um utilizador podia alterar o uid passado por RPC
- **Solução**: Utilização de encriptação DES da informação de autenticação do utilizador (chave secreta)
  - atualmente: também se pode utilizar o serviço de autenticação Kerberos

Interface do servidor 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*.

continua no próximo slide

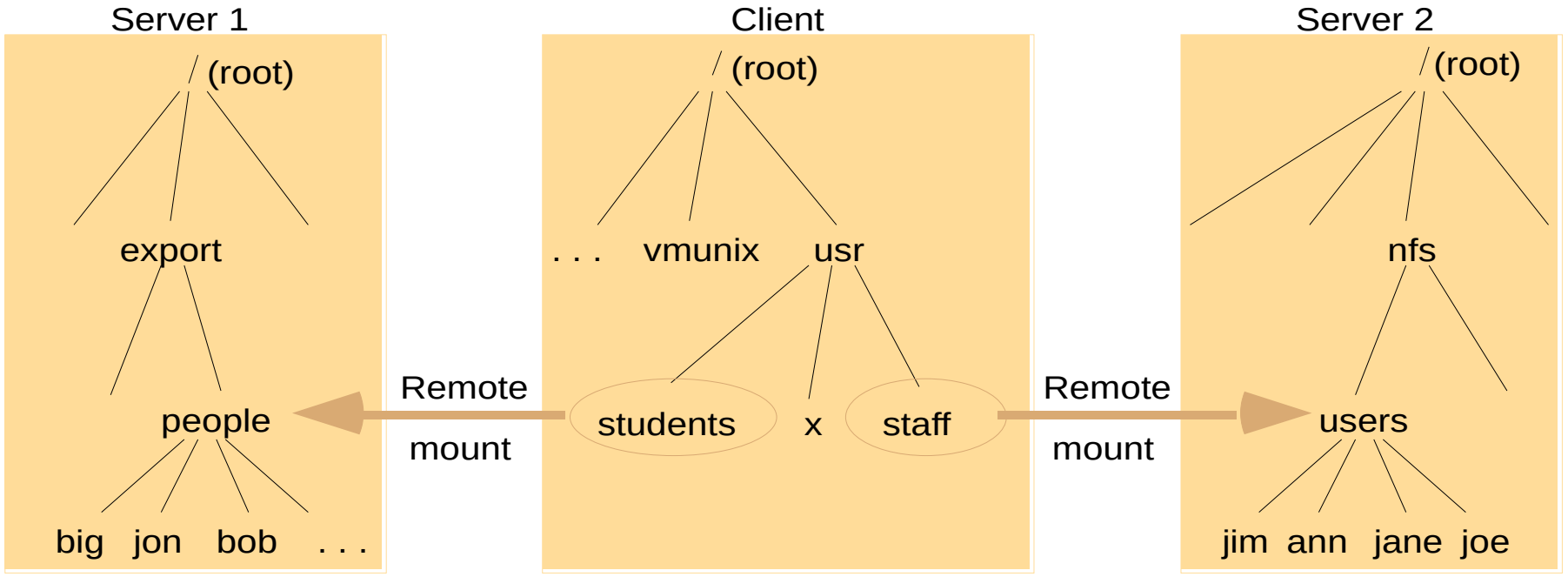
Interface do servidor NFS (continuação)

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

NFS: Sistema de Ficheiros Local e Remoto acessíveis

Mount Service: permite montar parte de um file system remoto por um cliente

- serviço fornecido por um servidor específico (user level no NFS server)



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

NFS: Cache

no servidor

- blocos de ficheiros e atributos mantidos em memória
- **read ahead** (pelo princípio de localidade espacial e temporal)
- persistência/consistência: refletir escrita imediatamente no disco?
  - 1- ***write-through caching***: atualizar memória e disco antes de responder ao cliente
  - 2- ***delayed commit*** *ou* ***delayed write:*** escrever apenas na cache em memória. A sincronização com o disco será efetuada com a operação *commit* quando o ficheiro é fechado, ou sempre que o cliente o pede, ou após um timeout (30s).

no cliente

- cache sobre o resultado de read, write, getattr, lookup e readdir para reduzir os pedidos ao servidor e o tráfego na rede
- o estado de um ficheiro ativo é atualizado com o servidor a cada 3s
  - para evitar o volume do getattr(), pode assumir-se que cada operação sobre ficheiros ou diretorias tem um getattr() implícito

NFS: considerações sobre desempenho

transparência de acesso e localização, tolerância a falhas*

*o facto do servidor ser *stateless* permite retomar o funcionamento (em cliente ou servidor) após crash sem necessidade de procedimento especial de recuperação

- perder-se-ia só a cache no cliente

normalmente não introduz penalizações muito relevantes comparativamente ao acesso a ficheiros locais

Problemas indicados:

- escala limitada pela capacidade do servidor
- sensível à latência da rede
- desempenho relativamente inferior na operação write se usar write-through no servidor
  - mas writes são “pouco” frequentes

Andrew File System (AFS)

acesso transparente a ficheiros partilhados remotos (como NFS) para programas UNIX

o acesso aos ficheiros faz-se através das primitivas UNIX

Servidor AFS

- armazena ficheiros UNIX, referenciados de modo semelhante ao NFS (não pelo i-node do sistema UNIX)

O principal objectivo do AFS é a escalabilidade, alcançada via cache

- Whole-file Serving: o servidor transmite o conteúdo inteiro de ficheiros e diretorias (em blocos de 64K)
- Whole-file Caching: ao receber um ficheiro (inteiro), o cliente faz uma cache... uma cache persistente (permanece após um crash do cliente)

AFS: funcionamento resumido

quando uma aplicação do utilizador faz *open* de um ficheiro remoto que não tem uma réplica local, o servidor respetivo é localizado e envia uma cópia

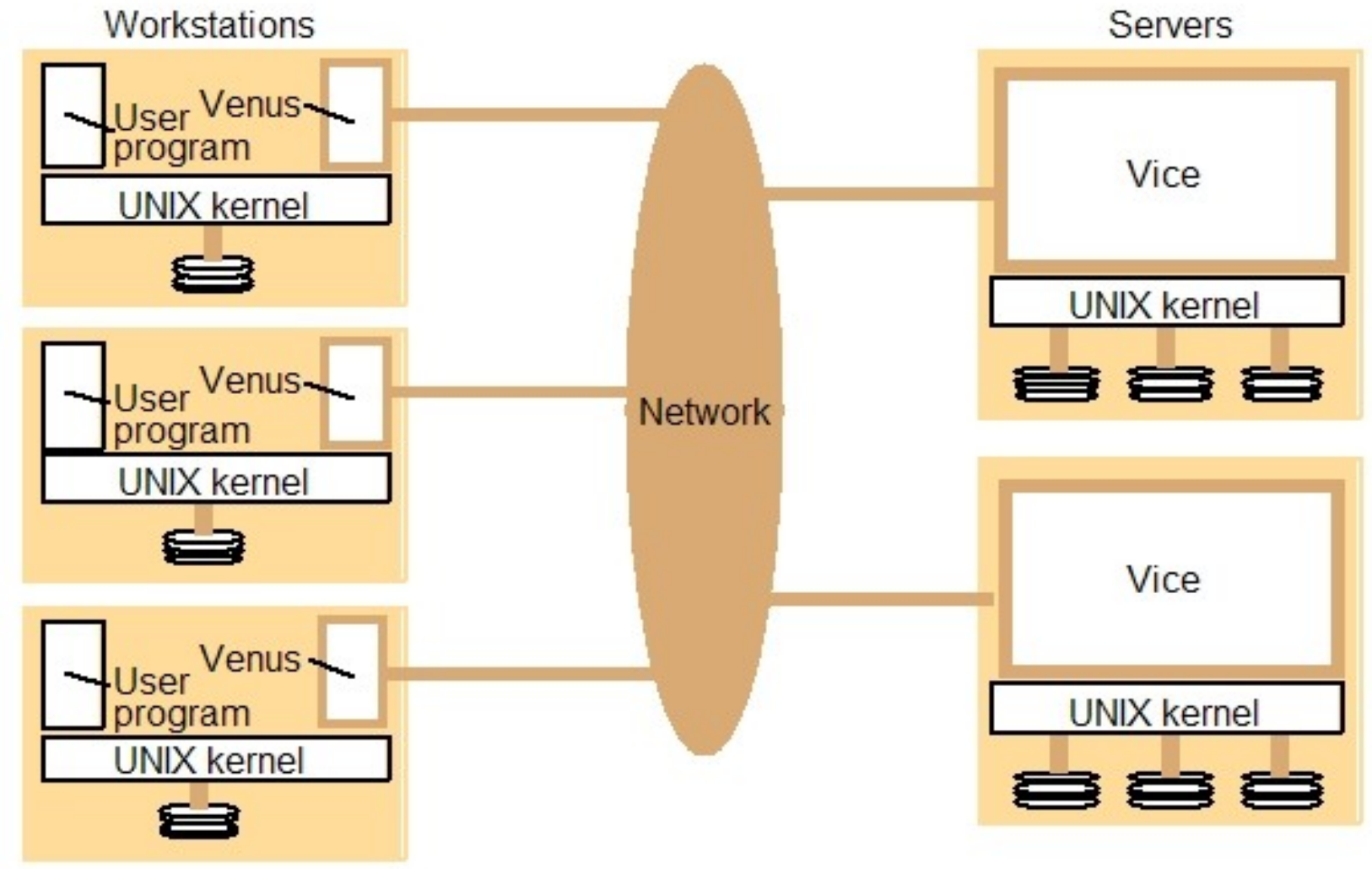
a cópia é armazenada no file system do cliente e aberta, sendo devolvido ao utilizador o respectivo *file descriptor* unix

as operações de leitura e escrita ocorrem sobre essa réplica

ao efetuar um *close* se a réplica foi alterada o conteúdo é enviado ao servidor AFS que atualiza a sua. A réplica permanece junto do cliente para eventual uso por outro utilizador da *workstation*

...e a consistência??

Distribuição de processos no Sistema de F. Andrew



AFS: processos (módulos)

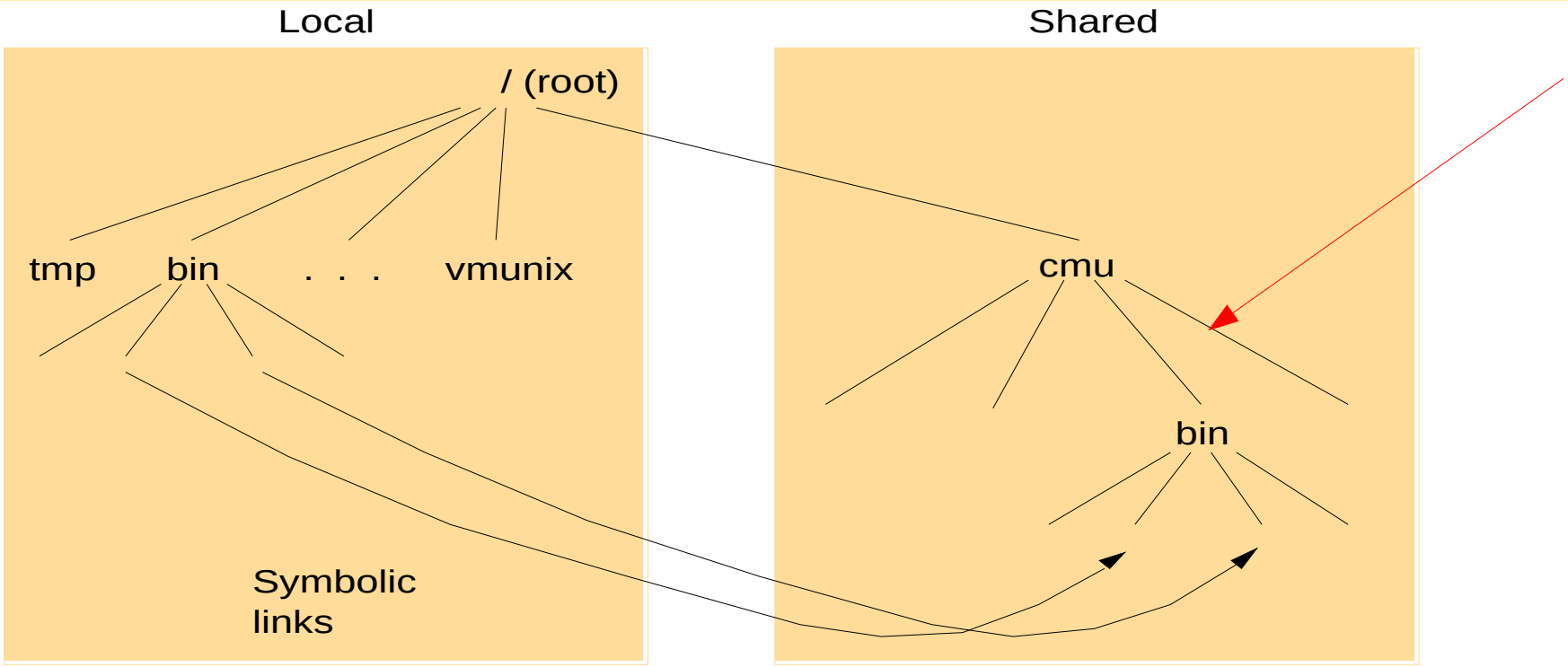
Vice

- software, processo UNIX *user-level*, corre do lado do servidor

Venus

- software, processo UNIX *user-level* do lado do cliente – corresponde ao Módulo Cliente no modelo abstrato

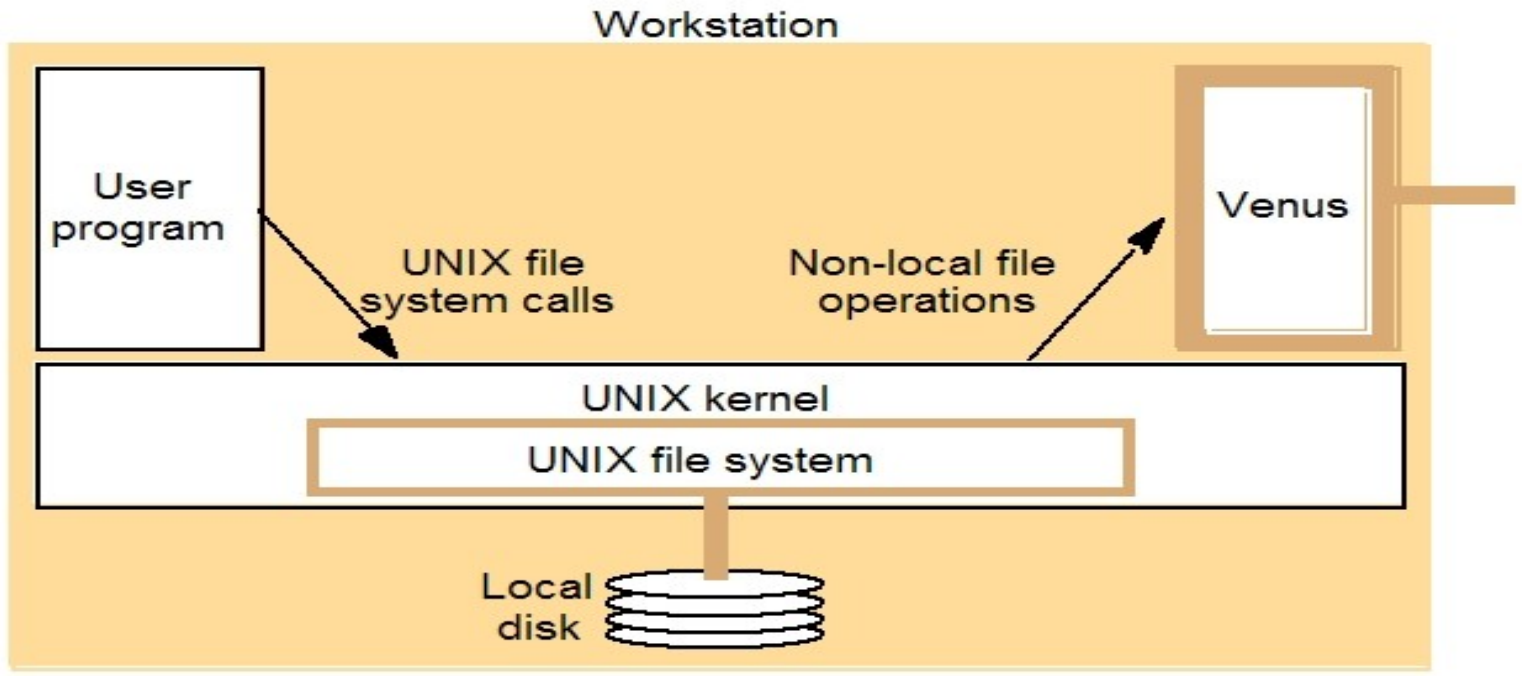
Espaço de Nomes visível pelos clientes SF Andrew



/cmu vai conter todos os ficheiros partilhados

separação local/partilhado vai contra a transparência de localização

Identificação de System Calls sobre ficheiros remotos



kernel UNIX intercepta *system calls* sobre partilhados e passa-as ao processo Venus

Implementação de *file systems* no AFS

User process	*UNIX kernel*	*Venus*	*Net*	*Vice*
open(FileName, mode)	If *FileName* refers to a file in shared file space, pass the request to Venus.	Check list of files in local cache. If not present or there is no valid *callback promise*, send a request for the file to the Vice server that is custodian of the volume containing the file.	→	Transfer a copy of the file and a *callback promise* to the workstation. Log the callback promise.
	Open the local file and return the file descriptor to the application.	Place the copy of the file in the local file system, enter its local name in the local cache list and return the local name to UNIX.	←	
read(FileDescriptor, Buffer, length)	Perform a normal UNIX read operation on the local copy.			
write(FileDescriptor, Buffer, length)	Perform a normal UNIX write operation on the local copy.			
close(FileDescriptor)	Close the local copy and notify Venus that the file has been closed.	If the local copy has been changed, send a copy to the Vice server that is the custodian of the file.	→	Replace the file contents and send a *callback* to all other clients holding *callback promises* on the file.

AFS: Consistência

callback promise: garantia de Vice para Venus de que notificará a atualização do ficheiro sempre outro cliente o alterar

- armazenadas do lado do cliente, junto à réplica do ficheiro, 2 estados:
  - válida (usar a cópia em cache)
  - cancelada (é necessário obter uma cópia de Vice)
- são revistas quando o servidor recebe uma operação *close* de um ficheiro atualizado

Semântica de Atualização

- manter a consistência da cache com uma aproximação da semântica *one-copy file* que seja praticável sem degradação do desempenho

AFS: Componentes da *service interface* Vice

Fetch(fid) -> attr, data	Returns the attributes (status) and, optionally, the contents of file identified by the *fid* and records a callback promise on it.
Store(fid, attr, data)	Updates the attributes and (optionally) the contents of a specified file.
Create() -> fid	Creates a new file and records a callback promise on it.
Remove(fid)	Deletes the specified file.
SetLock(fid, mode)	Sets a lock on the specified file or directory. The mode of the lock may be shared or exclusive. Locks that are not removed expire after 30 minutes.
ReleaseLock(fid)	Unlocks the specified file or directory.
RemoveCallback(fid)	Informs server that a Venus process has flushed a file from its cache.
BreakCallback(fid)	This call is made by a Vice server to a Venus process. It cancels the callback promise on the relevant file.

AFS: outros aspetos

requer alterações no Kernel UNIX

- manipular os ficheiros a partir de *file handles* e não com os usuais UNIX *file descriptors*

o host do servidor é dedicado ao serviço AFS

réplicas *read-only* (guardadas em servidores separados, maior desempenho)

- cada atualização (rara) efetua-se diretamente no servidor

Desempenho (*performance*)

- principal objectivo do AFS é a escalabilidade
- boa performance para um elevado nº de utilizadores
- reduz a carga no servidor (pode ser até metade do NFS)
-

SFDs pré-GFS: Desenvolvimentos Posteriores

NFS

- Spritely NFS, NQNFS
  - tentam diferentes formas de manter a consistência da cache que resultem em melhoria de desempenho
- WebNFS – RFC 2055 e 2056
  - programa cliente de um servidor NFS algures na internet, utilizando diretamente o protocolo NFS – em vez de o usar indiretamente com um módulo do kernel

AFS

- DFS
  - gerar callbacks imediatamente a cada atualização do ficheiro (e não apenas no fecho)

Mecanismos de Armazenamento (em Discos)

- Redundant Arrays of Inexpensive Disks (RAID)
- Log-structured File Storage (LFS) (dados sofrem *updates* em memória, havendo lugar a commits periódicos para disco)

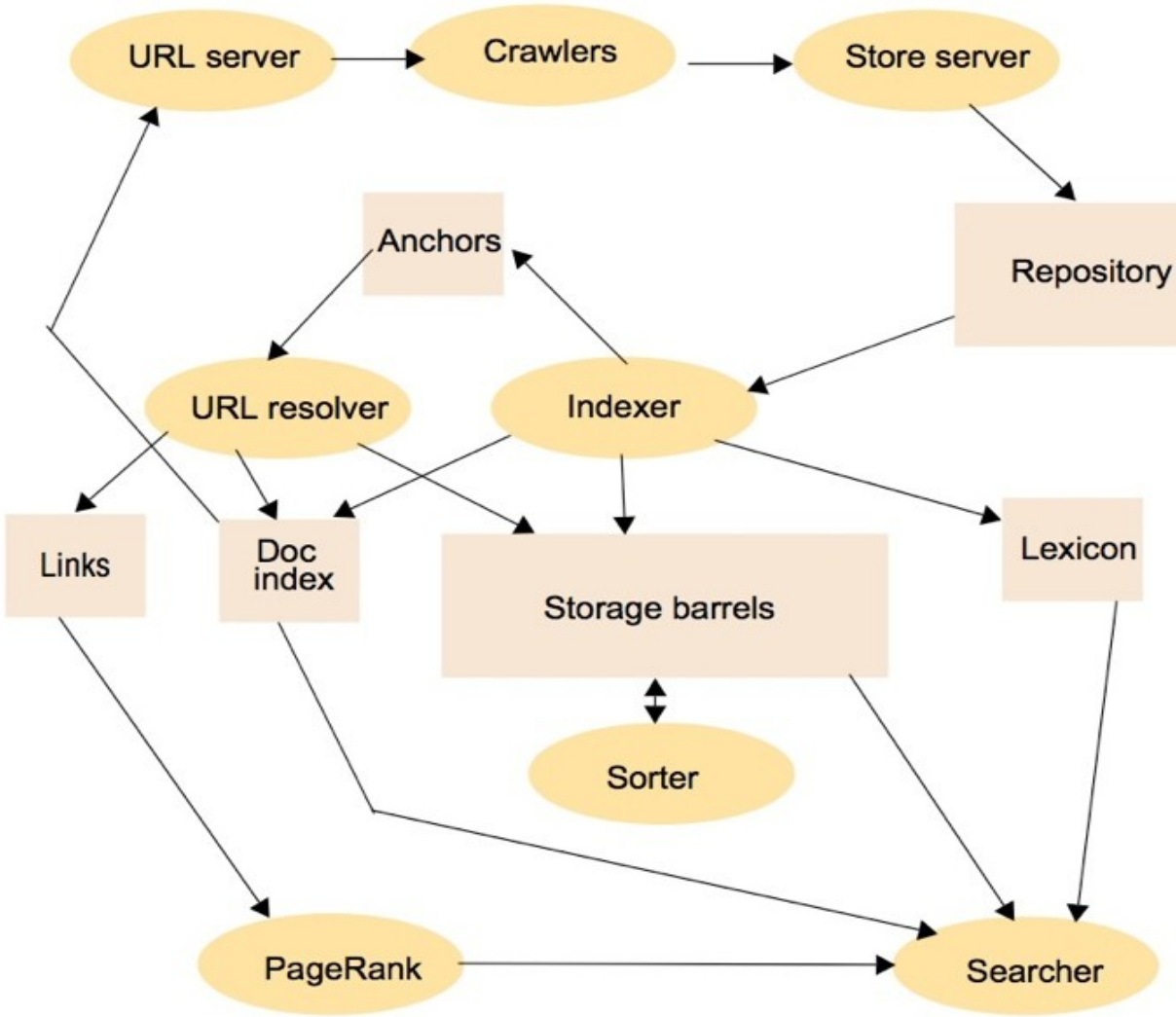
Google

- Nome deriva de *googol*, expressão que significa 10^100
  - a propósito de grandes quantidades de informação
- 1998
  - Resulta de um projeto de investigação na Stanford University
- Sucesso
  - Associado ao algoritmo de ranking de páginas Web no **motor de pesquisa**
- Hoje
  - Alargaram o leque de serviços
  - Uma referência em **cloud computing**

Motor de Pesquisa

- A partir dos termos da consulta, obter lista ordenada de docs Web
  - Crawling
    - Localizar e obter conteúdo de docs Web, passado-o ao módulo de indexação
    - Googlebot: agente que lê docs, recolhe URL que referem e avança recursivamente para os documentos apontados por esses endereços... Ajusta-se dinamicamente à frequência de atualização dos sites. Ex: sites de notícias são processados com mais frequência que outros menos dinâmicos.
  - Indexação
    - Indexar o conteúdo dos documentos (html, pdf, doc...). Para cada expressão ou palavra, manter um apontador para o local em que surge e como surge (maiúsculas, bold...)
  - Ranking
    - Ordenação do resultado pela importância relativamente à consulta/query
    - Algoritmo **PageRank** – critérios para apurar a importância para a ordenação
      - Uma página é mais importante se tem mais referências
      - A importância dos sites que referem a página é considerada
      - A proximidade das palavras na query, o facto desses termos surgirem destacados (bold, num título ou em maiúsculas)
      - ...

Figure 21.1
Outline architecture of the original Google search engine [Brin and Page 1998]



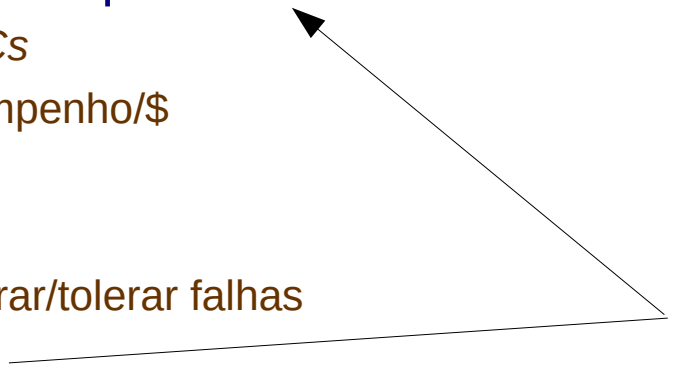
Google: fornecedor de serviços *cloud based*

- *Software as a Service* (SaaS)
  - Paradigma onde as aplicações são alojadas na internet, ao contrário do convencional software local. Tipicamente, são usadas através de uma interface web.
  - Exemplo: Gmail, Google Calendar...

- *Platform as a Service*
  - Disponibilização de APIs para serviços (distribuídos) que permitem a execução de aplicações (Web)
  - virtualização da plataforma

Infraestrutura Google: Modelo Físico

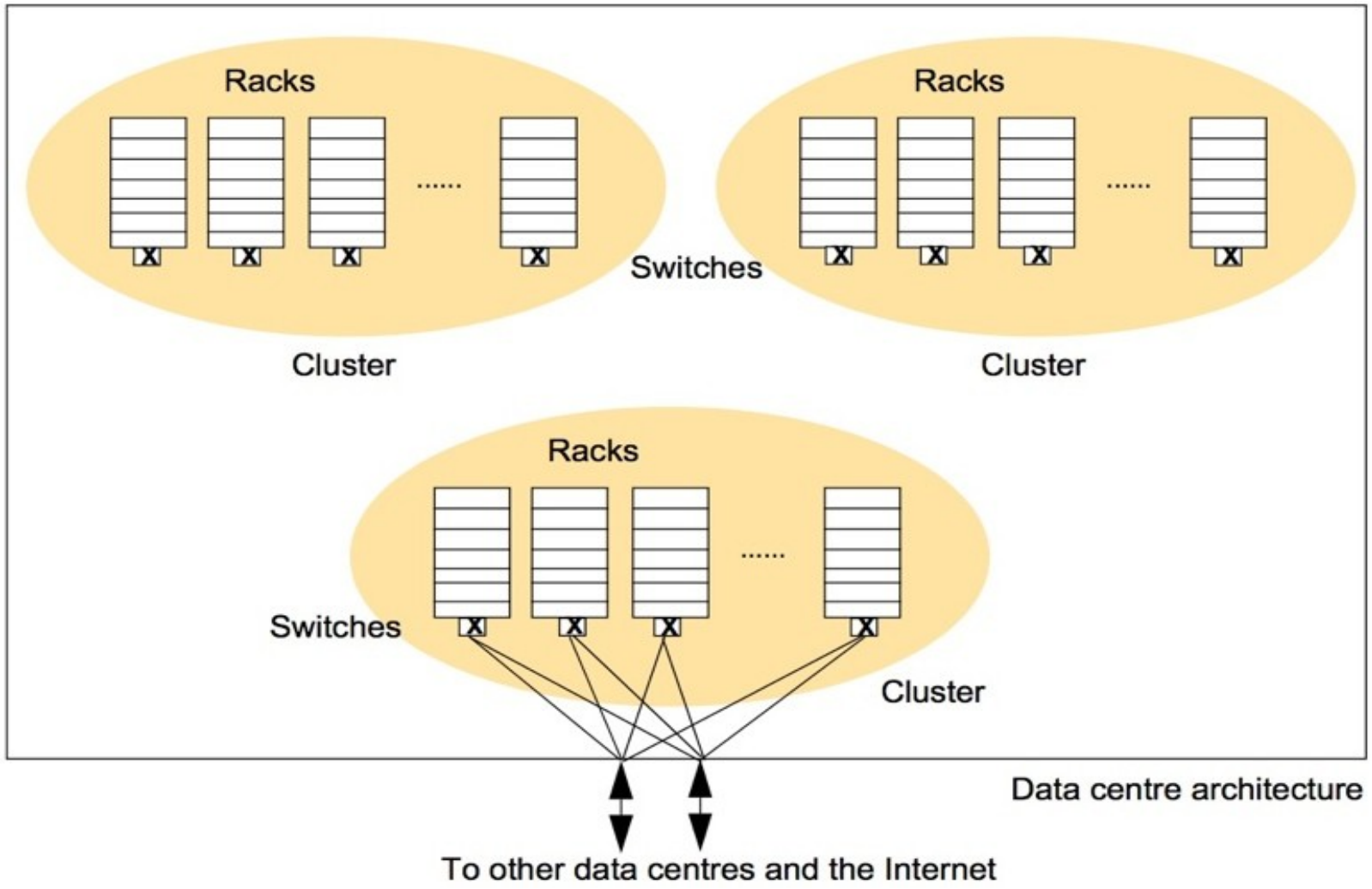
- Não recorrer a hardware caro e especialmente poderoso
  - Procurar o uso de quantidades de *commodity PCs*
    - A prioridade é obter a melhor relação desempenho/$
      - ~ 1k$, 2TB, 16G RAM, Linux
  - As falhas são inevitáveis em hardware comum
    - O sistema tem várias estratégias para superar/tolerar falhas
    - Maioria das falhas tem a ver com software
    - Falhas de HW são 1/10 das SW, e a maioria relacionada com disco ou RAM



- Racks
  - Conjuntos de 40 a 80 PCs
  - Divididas ao meio com metade das máquinas de cada lado
  - Ethernet switch com ligações redundantes

- Cluster
  - Grupo de 30 ou mais racks, com 2 switches de ALTA largura de banda
  - Cada rack é ligado a ambos os switches, por redundância

Figure 21.3
Organization of the Google physical infrastructure



(To avoid clutter the Ethernet connections are shown from only one of the clusters to the external links)

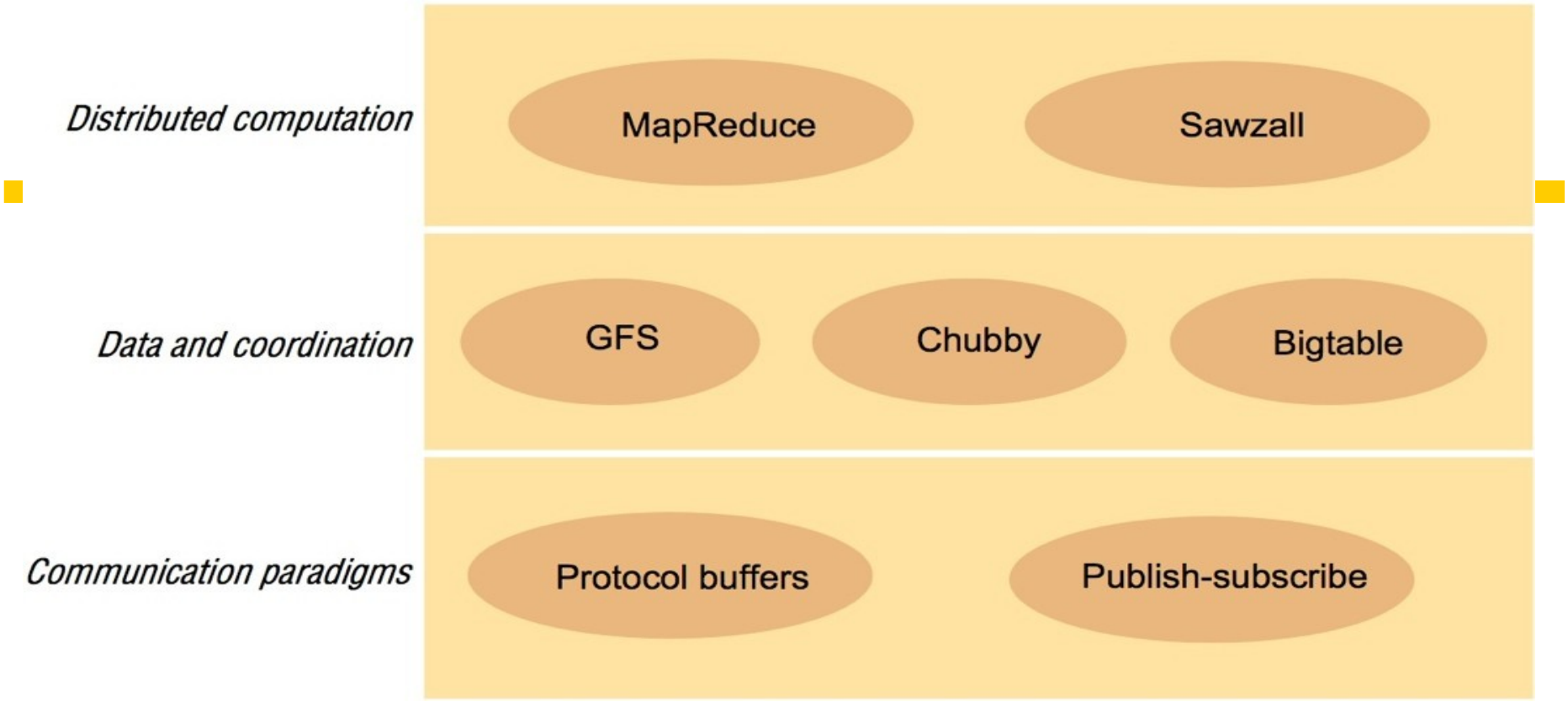
Requisitos da Arquitetura de suporte

- Escalabilidade
  - Aspetos:
    - Capacidade de tratar mais e mais dados, acomodar o crescimento da Web
    - Capacidade de processar mais consultas no search engine
    - Oferecer melhores resultados (chave para o sucesso)
  - Requer estratégia distribuída eficaz para crawl, indexar, ordenar
- Fiabilidade
  - Disponibilidade dos serviços (Ex: a pesquisa não pode ficar em baixo)
  - 1 de Setembro de 2009: GMail falhou... durante 100 minutos por colapso por sobrecarga
- Desempenho
  - Baixa latência na resposta ao utilizador (Ex: pesquisa Web em <=0.2s)
- Abertura
  - APIs, modularidade, retrocompatibilidade... abstração...

Infraestrutura Google: elementos por camadas

- Comunicação:
  - Direta / invocação remota / serialização: **Protocol Buffers**
  - Assíncrona, grande nº de subscritores: Google **publish-subscribe**
- Armazenamento, coordenação e acesso a dados
  - GFS: SFD específico do universo google
  - Chubby: coordenação e algum armazenamento
  - Bigtable: repositório de larga escala para dados semi-estruturados
- Computação com carácter distribuído
  - MapReduce: paradigma de computação sobre grandes volumes de dados, nomeadamente sobre Bigtable
  - Sawzall: linguagem de alto nível para descrever computação distribuída

Figure 21.6
Google infrastructure



Protocol Buffers

- Mecanismo independente da linguagem e plataforma para serialização de dados de modo simples e eficiente
  - Invocação do tipo RPC
- Serialização de alto desempenho
  - Dados comprimidos, para transmissão e/ou armazenamento
  - Reduzir custos de transmissão pela rede (3 a 10x menos espaço que XML)
- Mensagens
  - Campos: nome e tipo
  - Tipos: primitivos, enumerations, nested
  - O nº do campo (que começa em 1 para cada novo âmbito) permite identificar facilmente o campo pela posição do respetivo valor na representação binária
- Métodos gerados automaticamente: *setters, getters, test, clear, toString*...

Google File System (GFS)

Google File System

- descrito num paper de 2003
- SFD escalável pensado para sistemas que fazem uso intensivo de um grande volume de dados
- Tolerância a falhas
- Implementável com hardware comum (*inexpensive commodity hardware*)
- Alto desempenho para um grande nº de clientes

Design do GFS

- Baseado na observação do comportamento dos sistemas existentes
  - Eventos, carga, fluxos, padrões...
- Pensado para suportar as exigências específicas do universo Google (ex GoogleApps)

http://research.google.com/archive/gfs.html

GFS: Princípios

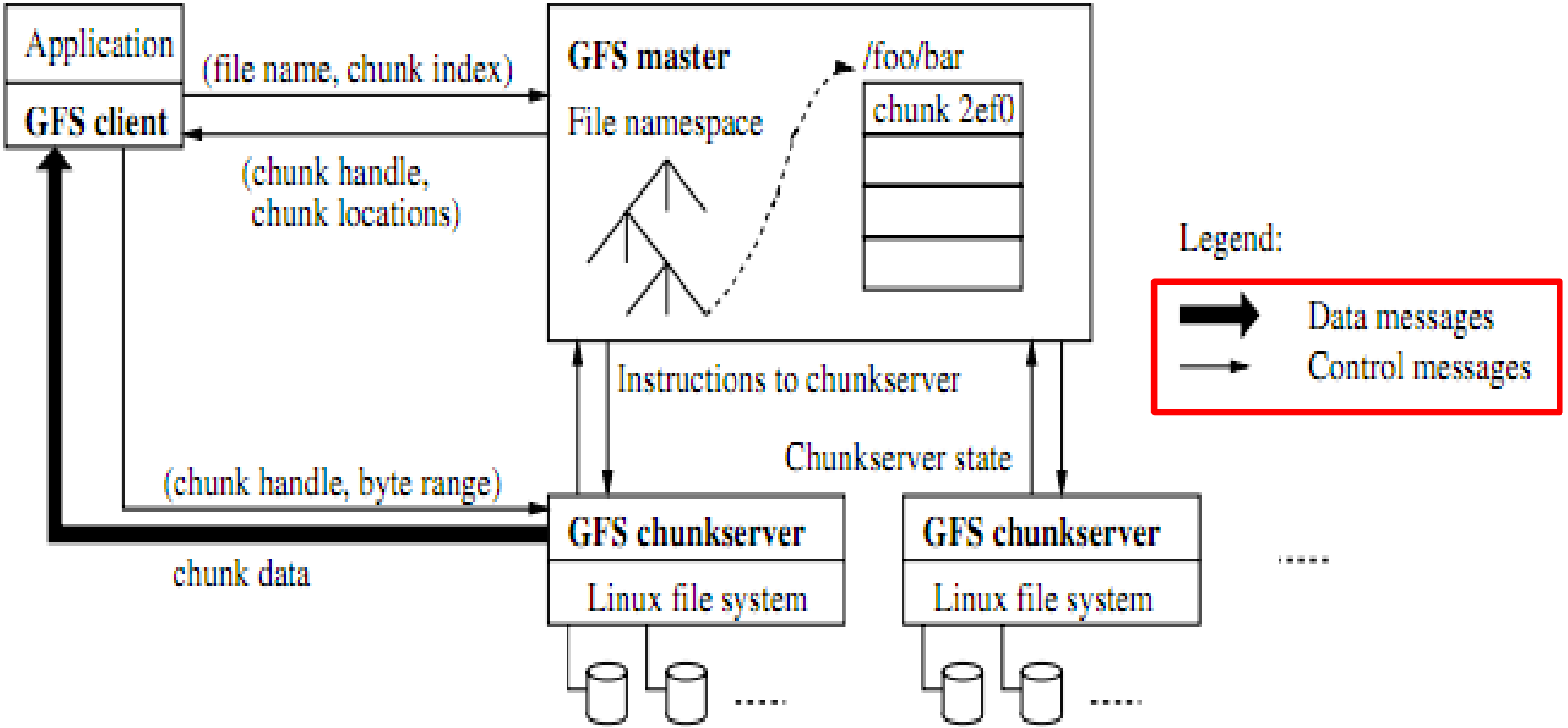
- Erros com os componentes são comuns (hardware simples falha)
- Ficheiros MUITO grandes (multi-giga)
- Muitas operações de leitura sequencial longa (*para streaming*)
- Operações de escrita
  - Poucas, pontuais
  - Maioria: grandes operações de substituição ou adição de ficheiros
- Padrão: ficheiro escrito uma vez e apenas lido desde então
- Alta largura de banda tem prioridade sobre a baixa latência

- Chunks: blocos de 64 MB* (divididos em blocos de 64KB+32 bits checksum)
  - Cada chunk é replicado por >=3 *chunkservers (*réplicas*)*
  - ** - bem maior que o usual em SFD convencionais*
- Mutação: escrita que ocorre em todas as réplicas do chunk, de acordo com um esquemas de *leases*.
- Gestão centralizada: *Master* - coordena acessos e controla metainformação
- Sem cache (pouco útil com grandes blocos de dados)

http://labs.google.com/papers/gfs-sosp2003.pdf

GFS

GFS – Arquitetura



http://labs.google.com/papers/gfs-sosp2003.pdf

GFS

GFS – Fluxo (controlo / dados) numa operação de escrita

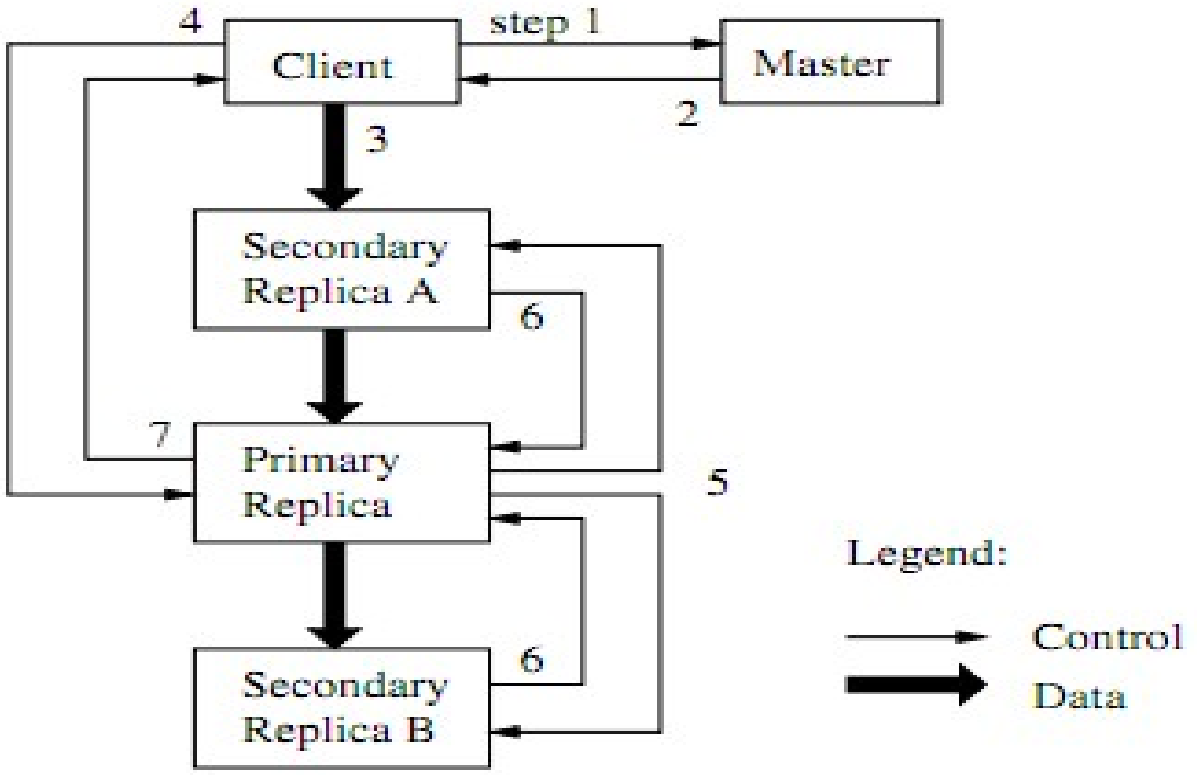


Figure 2: Write Control and Data Flow

http://research.google.com/archive/gfs.html

GFS: análise

- Gestão da consistência numa escrita
  - Em *write, append* ou *delete*, o master atribui o lease do chunck a uma das réplicas (primária). Esta ficará responsável pela aplicação ordenada de operações de escrita sobre o chunk, operações indicadas também aos chunkservers de backup
  - O cliente começa por contactar o master, que lhe identifica os chunckservers (primário atual - com lease - e outros)
  - O cliente envia* dados aos chunkservers, que registam mas não aplicam
    - * - cliente envia a um servidor, que propaga em cadeia aos restantes (otim. LgBanda)
      - *pipelining* para minimizar latência do processo
  - Chunkservers confirmam receção e o primário determina ordem de aplicação das operações. Se todos os chunkservers confirmam ao primário, este responde com sucesso ao cliente
- Elementos chave:
  - Redundância e gestão ágil das réplicas
  - Desempenho: especialmente em situações de acesso integral e sequencial
- Possíveis desvantagens
  - Desempenho em **pequenas** operações de escrita ou leitura
  - O modo de escrita em "*append mode*" é pensado para o contexto Google mas pode não ser o mais adequado às necessidades de uma empresa comum

Chubby

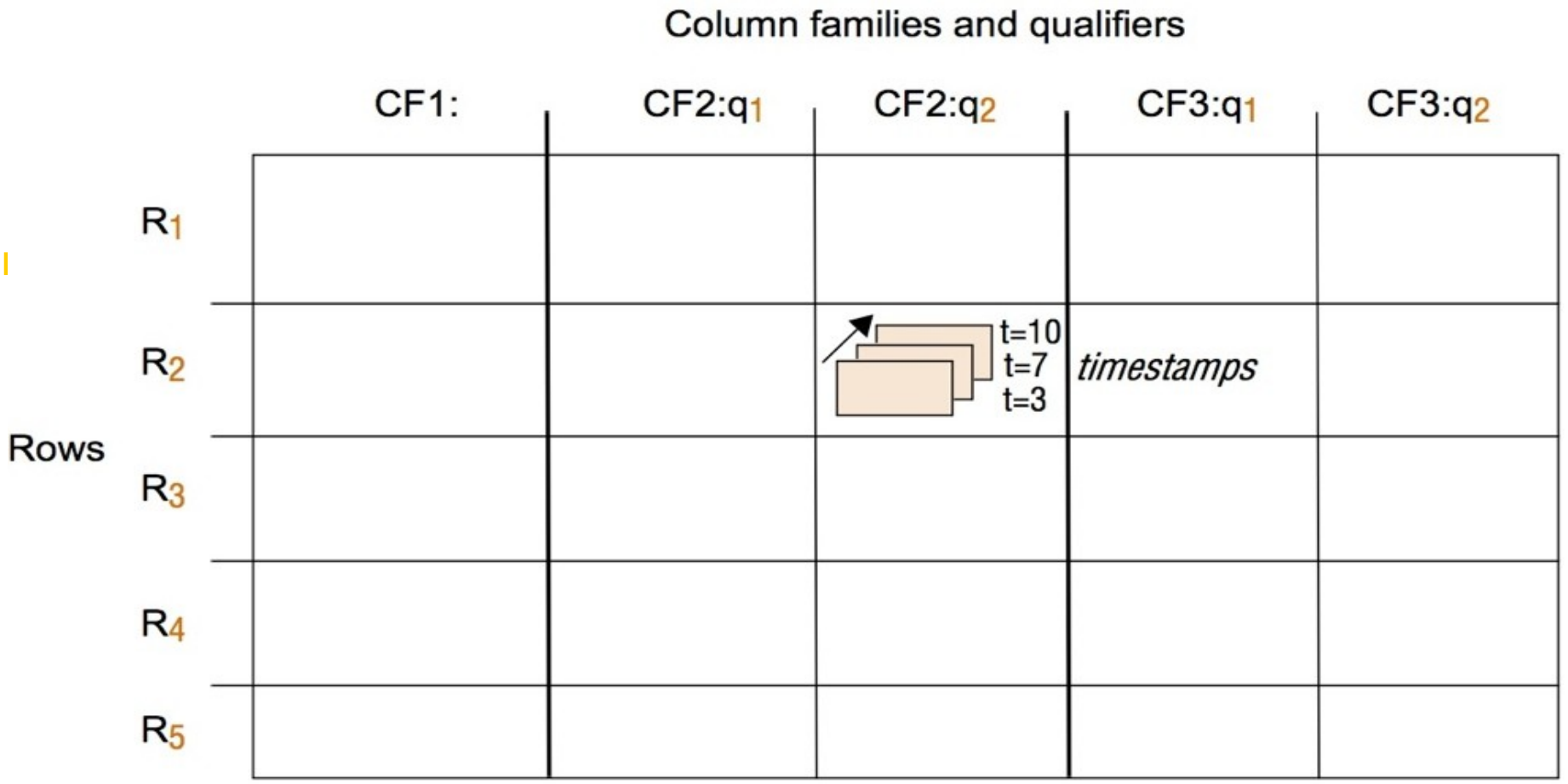
- Serviço para armazenamento de ficheiros comuns/pequenos
  - Complementar o GFS
- Serviço de sincronização distribuída em ambiente assíncrono
- Serviço de eleição (por exemplo do chunkserver primário)
  - Paxos: consensus algorithm
- Serviço de nomes

Bigtable

- Armazenamento de dados estruturados e semi-estruturados
  - Possivelmente ficheiros
  - Acesso pelo offset do conteúdo a partir do início do ficheiro
  - Read/write executados com grande otimização
  - Distribuição, grandes volumes
  - Dados organizados em **tabelas** muito grandes
    - Células indexadas por:
      - linha (string)
      - coluna (estrutura *família:qualifier*)
      - timestamp

- Google Analytics usa:
  - Tabela com dados de cliques sobre páginas visitadas (~200 TB)
  - Tabela com o sumário/análise desses dados (~20 TB, análise com MapReduce)
  - Optar por uma BD relacional distribuída teria desempenho inferior
  - Bigtable

Figure 21.13
The table abstraction in Bigtable



Arquitetura Bigtable

- Dados divididos em *Tablets* (100 a 200 MB)
  - Armazenadas em GFS
- Chubby: coordenação do acesso e distribuição
- Tablets são representadas em SSTables - *(key, value) pair maps*

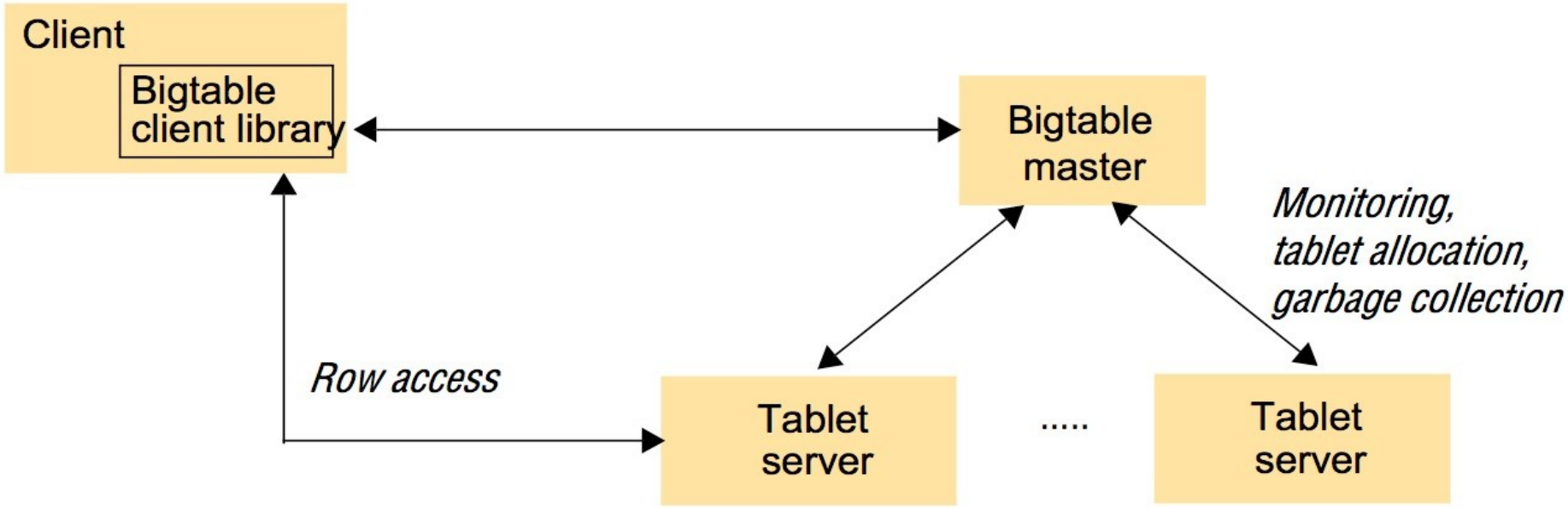


Figure 21.17

Summary of design choices related to data storage and coordination

Element	*Design choice*	*Rationale*	*Trade-offs*
GFS	The use of a large chunk size (64 megabytes)	Suited to the size of files in GFS; efficient for large sequential reads and appends; minimizes the amount of metadata	Would be very inefficient for random access to small parts of files
	The use of a centralized master	The master maintains a global view that informs management decisions; simpler to implement	Single point of failure (mitigated by maintaining replicas of operations logs)
	Separation of control and data flows	High-performance file access with minimal master involvement	Complicates the client library as it must deal with both the master and chunkservers
	Relaxed consistency model	High performance, exploiting semantics of the GFS operations	Data may be inconsistent, in particular duplicated
Chubby	Combined lock and file abstraction	Multipurpose, for example supporting elections	Need to understand and differentiate between different facets
	Whole-file reading and writing	Very efficient for small files	Inappropriate for large files
	Client caching with strict consistency	Deterministic semantics	Overhead of maintaining strict consistency
Bigtable	The use of a table abstraction	Supports structured data efficiently	Less expressive than a relational database
	The use of a centralized master	As above, master has a global view; simpler to implement	Single point of failure; possible bottleneck
	Separation of control and data flows	High-performance data access with minimal master involvement	-
	Emphasis on monitoring and load balancing	Ability to support very large numbers of parallel clients	Overhead associated with maintaining global states