Confiabilidade de Sistemas Distribuídos
Dependable Distributed Systems

DI-FCT-UNL, Henrique Domingos, Nuno Preguiça

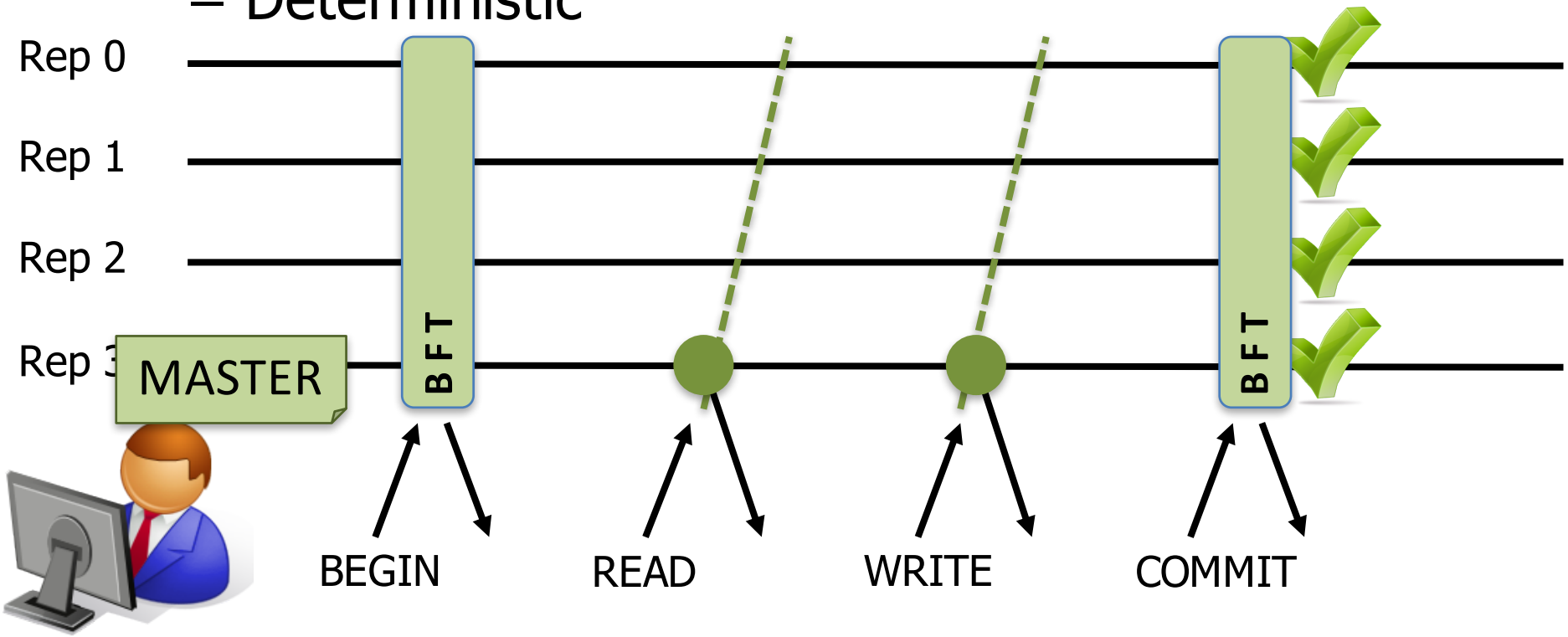
Lect. 8
Use cases: DepSky

2015/2016, 2nd SEM

MIEI
Mestrado Integrado em Engenharia Informática

Last-lecture: Byzantine fault-tolerant database replication

- Correct replicas compute the same results
  - Execute in the same snapshot
    - BEGIN & COMMIT are totally ordered
  - Deterministic



Comparing solutions

- Single master
  - All transactions proceed in all replicas with one-operation lag
  - Faster commits
- Multiple masters
  - Non-master replicas execute transaction operations in a burst

Today

- DepSky

DepSky – Dependable and Secure Storage in a Cloud-of-Clouds

Context

- Data is moving to the cloud
  - Enterprise: Amazon S3, Azure Storage, RackSpace, …
  - Personal: Dropbox, Google Drive, …
- Availability, fault-tolerance, elasticity/cost model

Potential problems of using a single cloud

- Loss of availability
  - Services are sometimes unavailable
- Loss and corruption of data
  - ... sometimes wrong things happen
- Loss of privacy
  - Even if provider is trusted, administrator sometimes are not
- Vendor lock-in
  - Hard to change service

NEWS ANALYSIS

OOPS: Google "loses" your cloud data (sky falling; film at 11)

"Google takes availability very seriously, and the durability of storage is our highest priority. We apologise to all our customers who were affected by this exceptional incident. We have conducted a thorough analysis of

MORE LIKE THIS

Mother Nature teaches Google a lesson

A Warehouse of Carphones? What is this, Back To The Future?

Meet Google Fi: The new meh, so-so, OK carrier

on IDG **Answers**

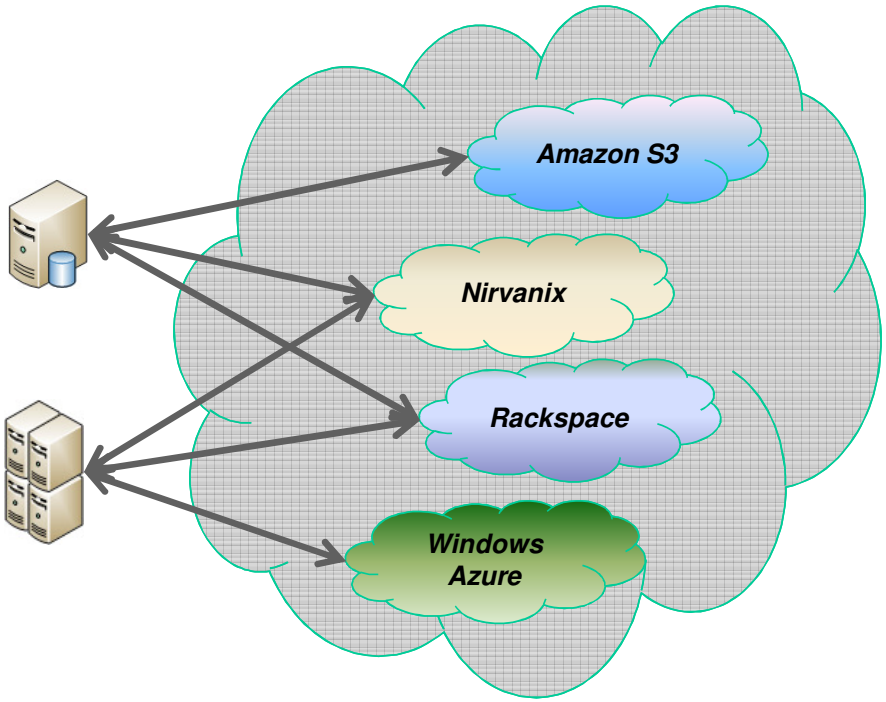
How is data stored and accessed in the cloud?

AT&T DATA THIEF FIRED AFTER ACCESSING 1,600 PRIVATE CUSTOMER ACCOUNTS

By **Andy Boxall** — October 7, 2014

Cloud of cloud storage

- "Replicate" data at multiple clouds
- Benefits:
  - Datacenter and cloud outages
  - Vendor lock-in
  - Data corruption
    - Bugs
    - Malicious insiders
    - Attacks and intrusions



DepSky Design Principles

1. **No trust on individual cloud providers**
   – *Distributed trust is built by using multiple clouds*
2. **Use storage clouds as they are**
   – *No server-side code on the replication protocols*
3. **Data is updatable**
   – *Quorum replication protocols for consistency*

Availability despite cloud failures

- Goal: continue to be able to access data even if some server is down
- Possible solution
  - Replicate at multiple replicas
- Drawbacks
  - Too much space used
  - If a single replica is compromised, data may get compromised

- Is it possible to create n fragments of which it is only necessary to have m to reconstruct data?

Erasure codes / Códigos de Apagamento

- Problemática de enquadramento:
  - Fragmentação com garantias de disponibilidade e integridade
  - Garantias adicionais: confidencialidade e autenticidade
- Fragmentação com redundância e dispersão
- Códigos de apagamento
- Códigos de apagamento com garantias de confidencialidade
- Esquemas existentes

Ideia interessante: fragmentar, disseminar e cifrar os fragmentos

- Fragmentar os dados
- Disseminar os fragmentos pelas réplicas
  - de modo a haver sempre possibilidade de reconstruir os objetos (dados) integrais, sempre que se queira
  - Podemos também mitigar a explosão em complexidade linear do armazenamento ? Aspecto interessante !
- Cifrar os fragmentos (para garantir a confidencialidade)
- Podemos ainda garantir provas de integridade ou autenticidade dos fragmentos
  - MACs, HMACs, CMACs .... Ou mesmo Assinaturas Digitais

Como aproximar soluções para os efeitos desejados ?

- Podemos ter que garantir que a reconstrução possa apenas ser feita por uma entidade (principal) com autorização para o fazer !
  - Se o atacante comprometer um servidor apenas terá acesso a um pedaço dos dados, sem significância

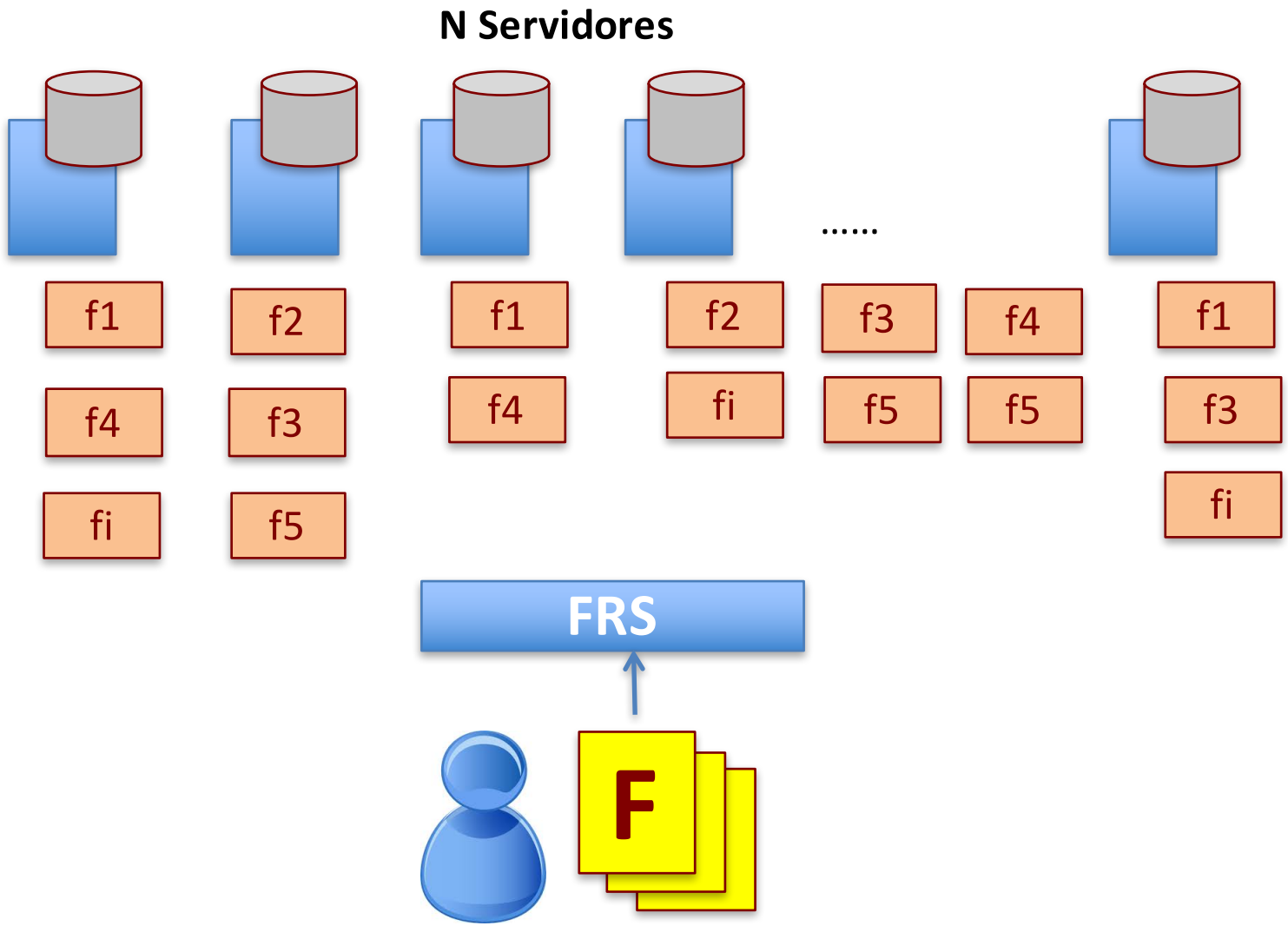
- Abordagem de referência aplicada à problemática da tolerância a intrusões:
  - **Fragmentação para redundância e dispersão !**

Códigos de Apagamento

- Problemática de enquadramento:
  - Fragmentação com garantias de disponibilidade e integridade
  - Garantias adicionais: confidencialidade e autenticidade
- Fragmentação com redundância e dispersão
- Códigos de apagamento
- Códigos de apagamento com garantias de confidencialidade
- Esquemas existentes

FRS – *Fragmentation-Redundancy-Scattering*

- Fraga, Powel "A Fault and Intrusion Tolerant File System", Proc. of the 3rd Intl Conference on Computer Security, pp 203-218, Aug 1985

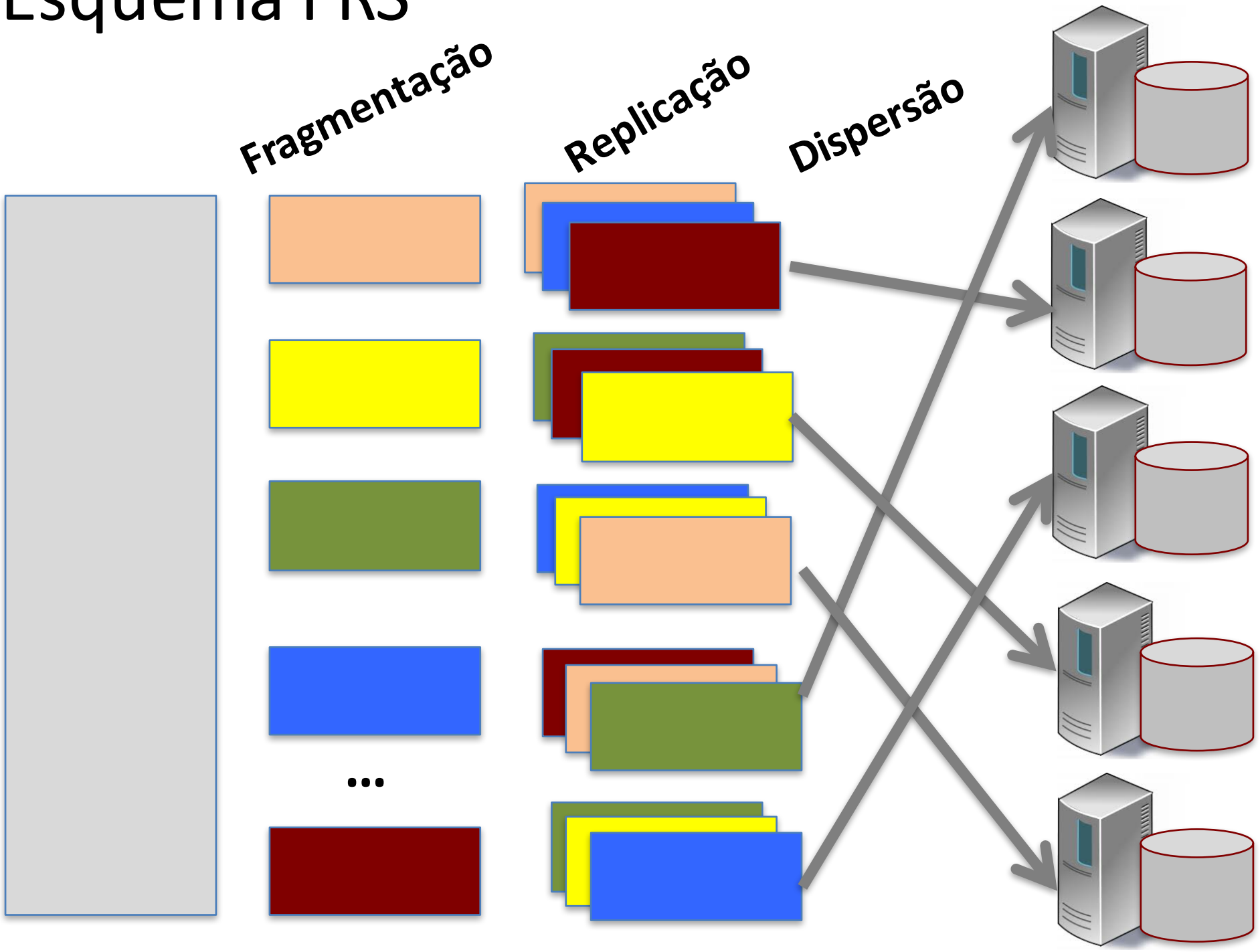


FRS – *Fragmentation-Redundancy-Scattering*

- Fraga, Powel "A Fault and Intrusion Tolerant File System", Proc. of the 3rd Intl Conference on Computer Security, pp 203-218, Aug 1985

- Abordagem inicial, com repercussão em muitas aproximações seguintes na investigação
- **Ideia base:**
  - **Sistema de ficheiros distribuído num conjunto de servidores**
    - Cada Ficheiro **F**, antes de armazenado, é fragmentado em m fragmentos
    - Os fragmentos são espalhados por **N** servidores
    - Cada fragmento é garantido estar em vários servidores
      - **Garante disponibilidade**
    - Nenhum servidor tem fragmentos suficientes que permitam reconstruir um objeto integral por parte de um intruso
      - **Garante confidencialidade**

Esquema FRS



Notas a reter no esquema FRS

- Não se garante confidencialidade (em sentido estrito)
  - Embora mitigue o problema forçando o adversário a obter m fragmentos, sendo m o factor de resiliência do processo de fragmentação
  - De acordo com os autores: a ideia é (sic) “reduzir o significado da informação acessível ao intruso”
- Integridade: duas soluções alternativas:
    - Na leitura lêem-se diversas cópias de cada fragmento e faz-se uma votação
    - Ou junta-se um MAC a cada fragmento

Outros aspetos no sistema FRS

- A localização dos fragmentos está num serviço também ele distribuído para tolerar intrusões
- Aspeto importante deste serviço é implementar um sistema de autorizações (controlo de acesso) discricionário (DAC)
- Solução teórica avançada:
  - Um esquema de partilha de segredos (secret splitting): para construir uma chave necessária para acesso ao ficheiro é necessário obter K partes desse segredo.
  - Os clientes que fazem o acesso são confiáveis

Códigos de Apagamento

- Problemática de enquadramento:
  - Fragmentação com garantias de disponibilidade e integridade
  - Garantias adicionais: confidencialidade e autenticidade
- Fragmentação com redundância e dispersão
- Códigos de apagamento
- Códigos de apagamento com garantias de confidencialidade
- Esquemas existentes

Códigos de Apagamento

- Já depois do trabalho de Fraga e Powell, surgiu o interesse neste tipo de técnicas
  - Ex., Quantos fragmentos ? Que optimização do espaço de dispersão ?
- Ex., Rabin – Algoritmo de Dispersão de Informação
  - Optimização do espaço, com códigos de apagamento (ERASURE CODES)

M. Rabin, Efficient Dispersal of Information for Security, Load Balancing and Fault Tolerance, Journal of the ACM 36(2), 335-348, 1989

Códigos de Apagamento: origem e inspiração

- Códigos de correção de erros usados no domínio das telecomunicações
  - Mas no caso a ideia é que a informação pode apenas ser "apagada" (não modificada)

- Ideia base:
  - **Dividir um objeto (ex. ficheiro F) em N fragmentos**
  - **Bastando ser suficiente ter *k* fragmentos para reconstruir F**
  - **Sem que quaisquer *k-1* fragmentos consigam reconstruir F**

Dizemos então que um tal esquema é um código de apagamento-(k, N)

Códigos de Apagamento de Shamir

- A ideia base foi apenas expressa inicialmente em formalismos e propriedades teóricas da fragmentação
- Implementações concretas de esquemas (em protocolos ou algoritmos) só vieram depois e que derivaram daquele trabalho
- Exemplos:
  - H. Krawczyk, "Distributed Fingerprints and Secure Information Dispersal", ACM Symp. on Principles of Distributed Computing, 1993
  - M. Alon, H. Kaplan, M. Krivelevich, D. Malkhi, J. Stern, "Scalable Secure Storage when Half the System is Faulty", Proc. Of 27th Intl Colloquium on Automata, Languages and Programming, LNCS 1853, 2000
  - J. Garay, R. Gennaro, C. Jutla, T. Rabin, "Secure Distributed Storage and Retrieval", Theoretical Computer Science, 2000

Códigos de Apagamento

- Problemática de enquadramento:
  - Fragmentação com garantias de disponibilidade e integridade
  - Garantias adicionais: confidencialidade e autenticidade
- Fragmentação com redundância e dispersão
- Códigos de apagamento
- Códigos de apagamento com garantias de confidencialidade
- Esquemas existentes

Como controlar acesso à informação

- Exige controlo de acesso ao ficheiro
- Junto com o ficheiro mantem-se uma lista de controlo de acessos ***L*** contendo identificadores dos clientes que podem aceder
- Cada F é cifrado usando cripto simétrica antes de ser armazenado
- Usa-se um esquema de criptografia de limiar (ou ***threshold scheme***)

cAVID com criptografia de limiar

- Evita que a chave criptográfica de confidencialidade fique com o cliente
  - Pois só ele poderia recuperar o ficheiro
- Criptografia de limiar:
  - É um esquema que permite cifrar dados com uma chave pública ***Pk*** com um algoritmo criptográfico assimétrico, com a decifa assegurada com a respetiva chave privada do par que é desdobrada em várias partes: ***SKi***
  - **Uma chave *Ski* permite obter um fragmento f_{ci}**

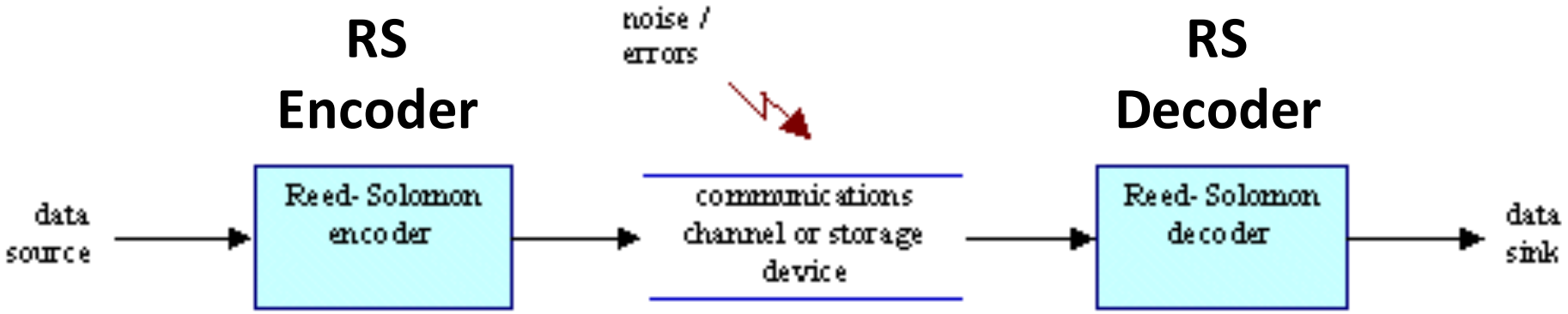
Arquitetura da solução

- Cada servidor tem a sua chave privada $\boldsymbol{Sk_i}$
- Todos os cliente têm a chave pública $\boldsymbol{Pk}$
- Para armazenar um ficheiro F:
  - Cliente gera chave simétrica $\boldsymbol{K}$ e calcula $\boldsymbol{\{F\}_k}$
  - Constrói um envelope de chave pública $\boldsymbol{\{K\}_{Pk}}$
  - Usa o alg. de dispersão AVID para armazenar o Ficheiro **F**, $\boldsymbol{\{K\}_{Pk}}$ e a lista de controlo de acessos $\boldsymbol{L}$
- Para ler:
  - Obtêm-se os fragmentos $\boldsymbol{f_{ci}}$ de $\boldsymbol{k}$ servidores para reconstruir $\boldsymbol{K}$

Códigos de Apagamento

- Problemática de enquadramento:
  - Fragmentação com garantias de disponibilidade e integridade
  - Garantias adicionais: confidencialidade e autenticidade
- Fragmentação com redundância e dispersão
- Códigos de apagamento
- Códigos de apagamento com garantias de confidencialidade
- Esquemas existentes

Reed-Solomon: utilização típica



Bloco de Dados

\+

Adição de Bits Redundantes

Ruido, Interferência na Comunicação "Scratches" num CD, etc

Recepção + Recuperação de Erros

Eficiência e o número e tipo de erros recuperáveis dependem do código RS utilizado

Reed Solomon codes

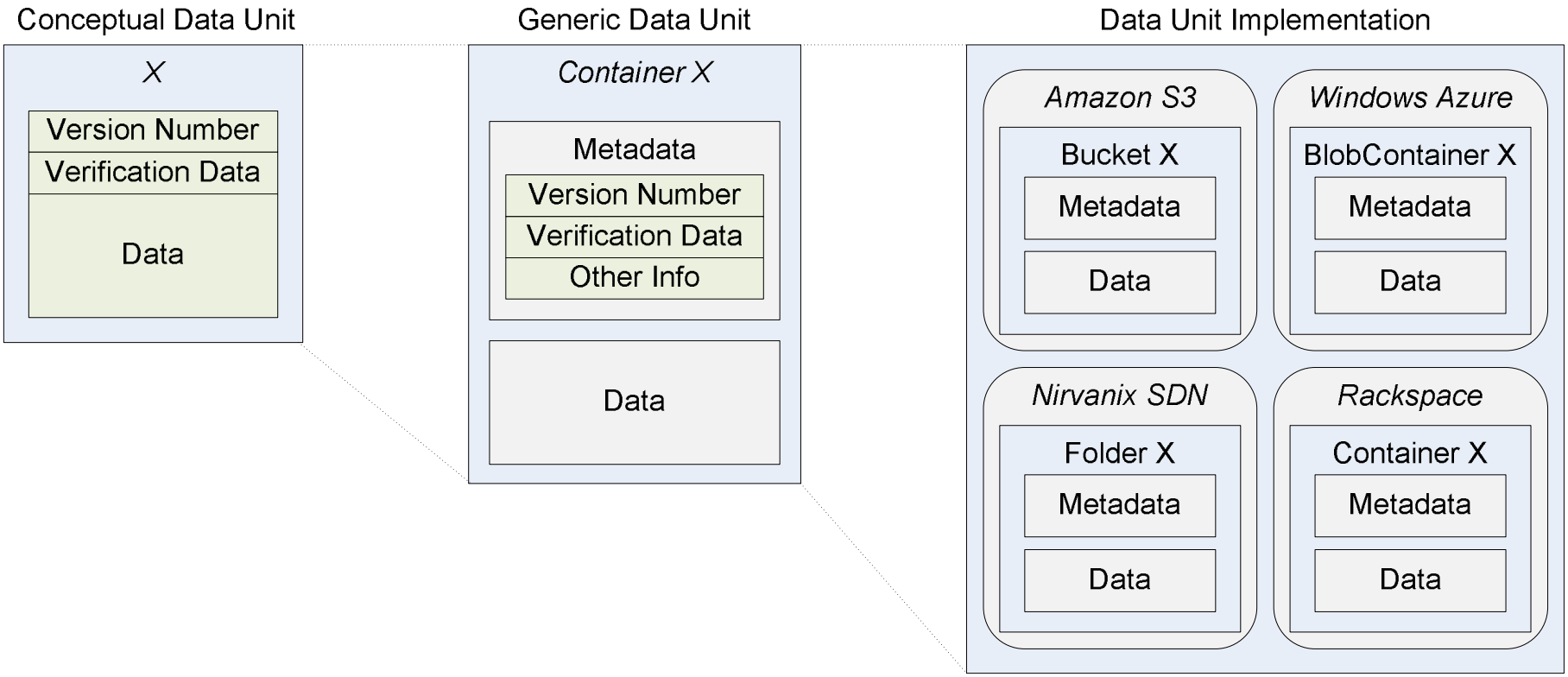
- k data symbols, $n - k$ parity checks.
- Field size $O(n)$.
- Any k symbols suffice for full data recovery.

How many parity checks do you need?

o(k) redundancy seems to be sufficient.

Should be getting overheads close to 0!

DepSky data model



DepSky key principles

- Metadata and data stored independently
- Metadata
  - Name
  - Version
  - Hash
- One writer / multiple readers
  - Multiple writers supported using locks

DepSky – confidentiality and availability

Function	Description
$queryMetadata(du)$	obtains the correctly signed file metadata stored in the container du of $n-f$ out-of the n clouds used to store the data unit and returns it in an array.
$writeQuorum(du, name, value)$	for every cloud $i \in \{0,...,n-1\}$, writes the $value[i]$ on a file named $name$ on the container du in that cloud. Blocks until it receives write confirmations from $n-f$ clouds.
$\mathsf{H}(value)$	returns the cryptographic hash of $value$.

DepSky – confidentiality and availability

Function	Description
$generateSecretKey()$	generates a random secret key
$\mathsf{E}(v,k)/\mathsf{D}(e,k)$	encrypts v and decrypts e with key k
$encode(d,n,t)$	encodes d on n blocks in such a way that t are required to recover it
$decode(db,n,t)$	decodes array db of n blocks, with at least t valid, to recover d
$share(s,n,t)$	generates n shares in such a way that at least t of them are required to obtain any information about s
$combine(ss,n,t)$	combines shares on array ss of size n containing at least t correct shares to obtain the secret s

DepSky – confidentiality and availability

```
1  procedure DepSkyCAWrite(du,value)
2  begin
3      if max_ver_du = 0 then
4          m ⟵ queryMetadata(du)
5          max_ver_du ⟵ max({m[i].version : 0 ≤ i ≤ n−1})
6      new_ver ⟵ max_ver_du + 1
7      k ⟵ generateSecretKey()
8      e ⟵ E(value, k)
9      s[0 .. n−1] ⟵ share(k, n, f+1)
10     v[0 .. n−1] ⟵ encode(e, n, f+1)
11     for 0 ≤ i < n−1 do
12         d[i] ⟵ ⟨s[i], e[i]⟩
13         h[i] ⟵ H(d[i])
14     writeQuorum(du,"value"+new_ver, d)
15     new_meta ⟵ ⟨new_ver, h⟩
16     sign(new_meta, K_rw)
17     v[0 .. n−1] ⟵ new_meta
18     writeQuorum(du,"metadata", v)
19     max_ver_du ⟵ new_ver
```

DepSky – confidentiality and availability

20 **function** DepSkyCARead(du)
21 **begin**
22 $m \longleftarrow queryMetadata(du)$
23 $max\_id \longleftarrow i : m[i].ver = \max(\{m[i].ver : 0 \leq i \leq n-1\})$
24 $d[0 .. n-1] \longleftarrow \perp$
25 **parallel for** $0 \leq i \leq n-1$ **do**
26 $tmp_i \longleftarrow cloud_i.get(du, \text{“value”} + m[max\_id].ver)$
27 **if** $\mathsf{H}(tmp_i) = m[max\_id].digest[i]$ **then** $d[i] \longleftarrow tmp_i$
28 **wait until** $|\{i : d[i] \neq \perp\}| > f$
29 **for** $0 \leq i \leq n-1$ **do** $cloud_i.cancel\_pending()$
30 $e \longleftarrow decode(d.e, n, f+1)$
31 $k \longleftarrow combine(d.s, n, f+1)$
32 **return** $\mathsf{D}(e,k)$

Questions

- How does DepSky tolerates DC fault?
- How does DepSky preserves confidentiality without the need for key distribution?