TECHNISCHE UNIVERSITÄT DRESDEN

Netzwerkkodierung in Theorie und Praxis

Praktische Anwendungen der Netzwerkkodierung

Professor Dr.-Ing. Dr. h.c. Frank H.P. Fitzek

M.Sc. Juan Cabrera

Deutsche Telekom Chair of Communication Networks (ComNets)

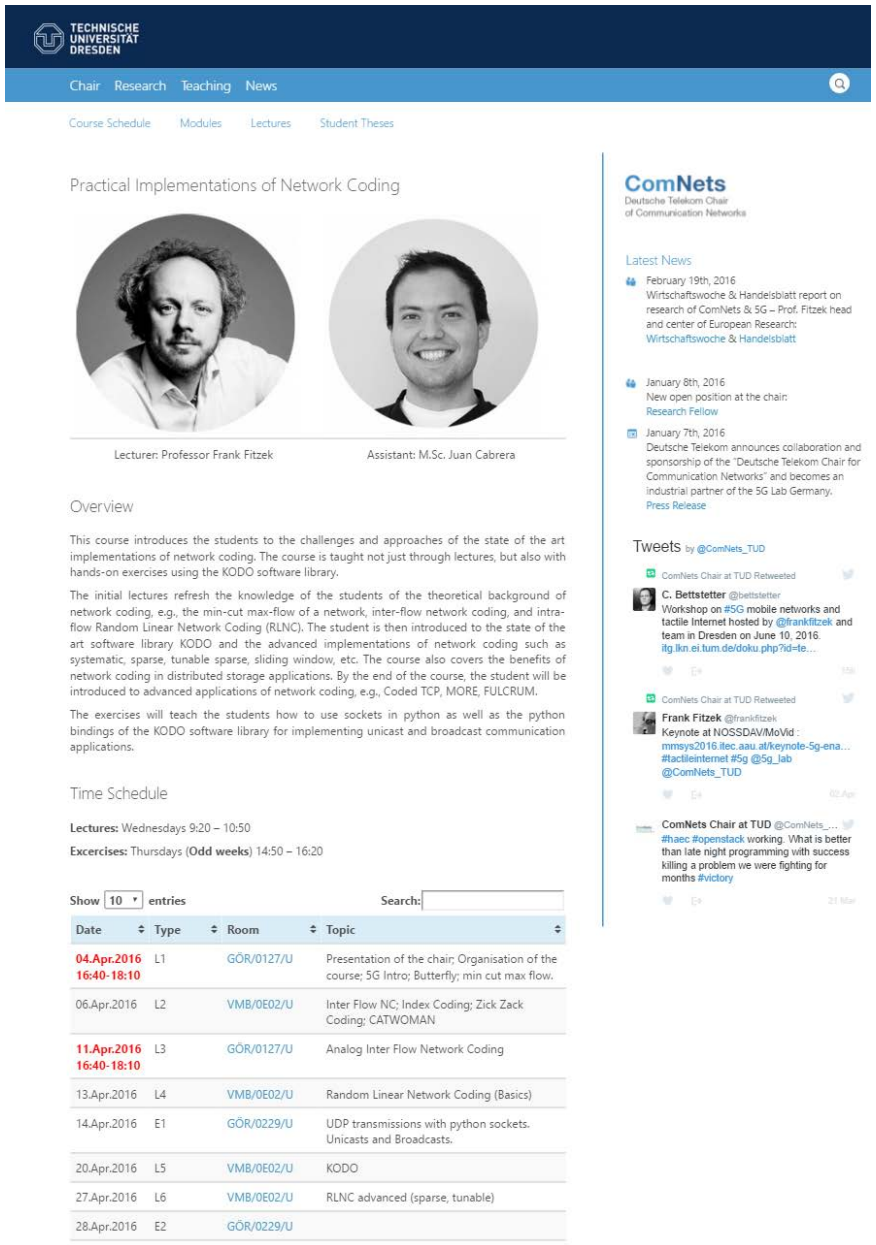
Netzwerkkodierungstheorie

Professor Dr.-Ing. Eduard Jorswieck

Dipl.-Ing. Johannes Richter

Theoretische Nachrichtentechnik

Lecture / Exercise Dates - tinyurl.com/zooafld



- Here all information for the lecture and the exercise can be found.
- Slides
- Links
  - Steinwurf
  - Python
  - KODOMARK (google play)

Please check every week!

TECHNISCHE UNIVERSITÄT DRESDEN

Network Coding Storage

Introduction and Motivation

- Cloud storage
  - Fast growth in the past years (both enterprise and home users)
  - Free storage space: Dropbox, Google Drive, SkyDrive, Box, iCloud, and countless more
  - Public APIs, usually REST-based
- Issues:
  - Reliability issues
  - Privacy and security issues
  - Limited free storage space

- Distributed storage on clouds – in use today on the provider side
- Our approach:
  - Distribute data to multiple cloud providers on the user side
  - Use network coding to create a robust, fast and secure distributed storage solution

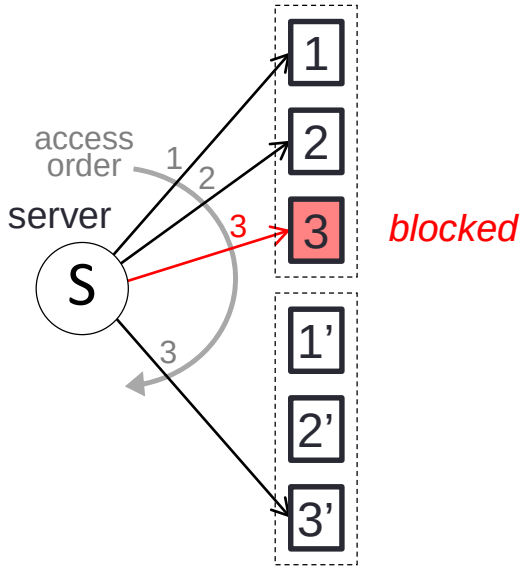
Distributed Storage

Data Centre 1G

User Cloud Storage 2G/3G

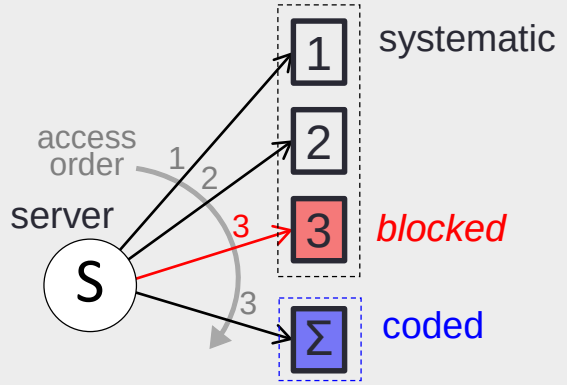
Distributed Storage

Conventional SAN



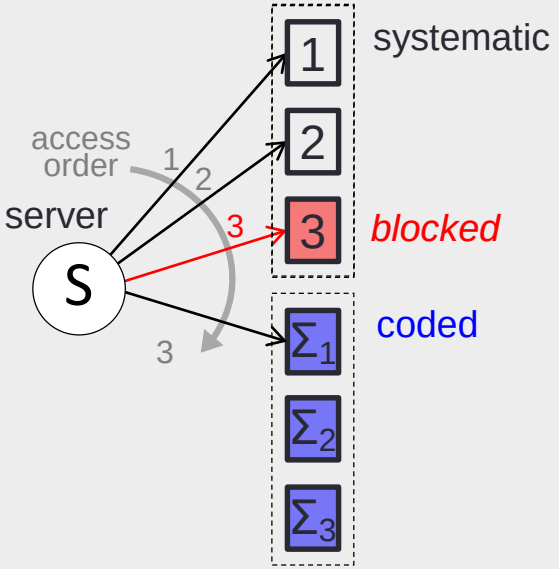
- 2 file copies
- 3 file chunks
- One chunk per drive
- Round-robin server access
- Chunk “3” blocked

RLNC-encoded SAN (two options)

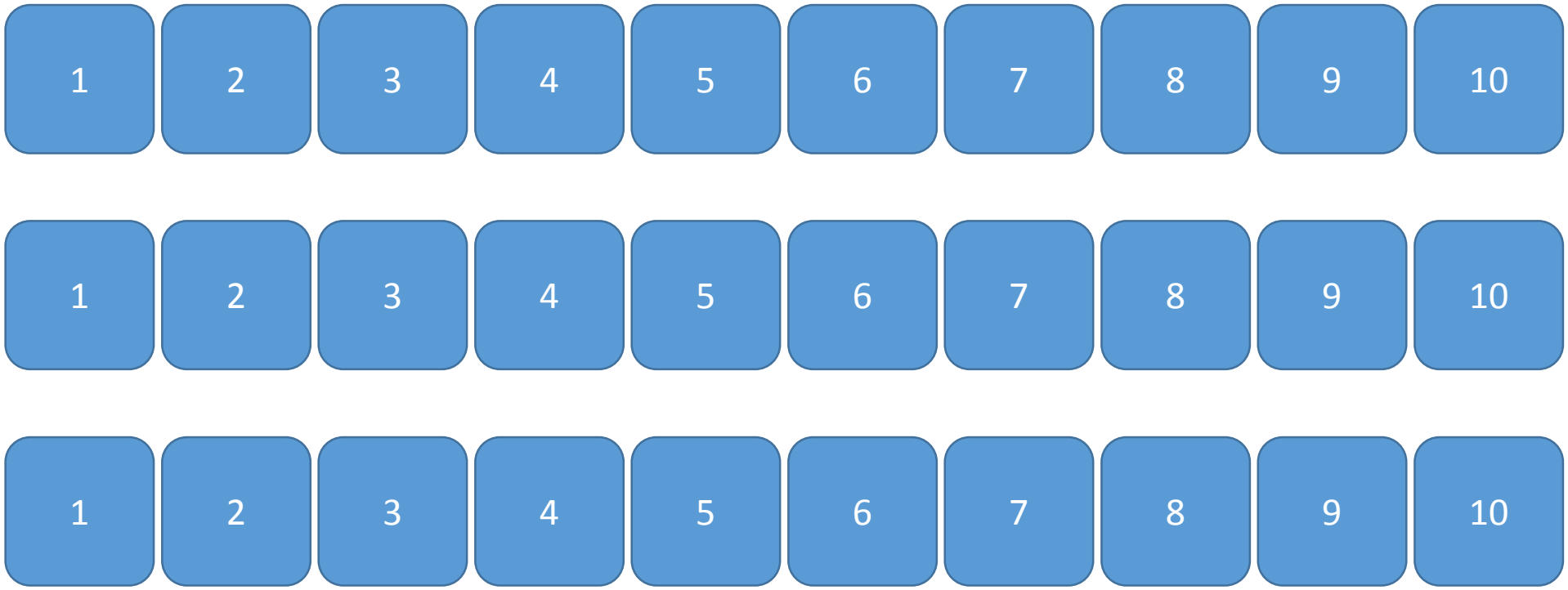


Decoding

$3 = \Sigma - 1 - 2$

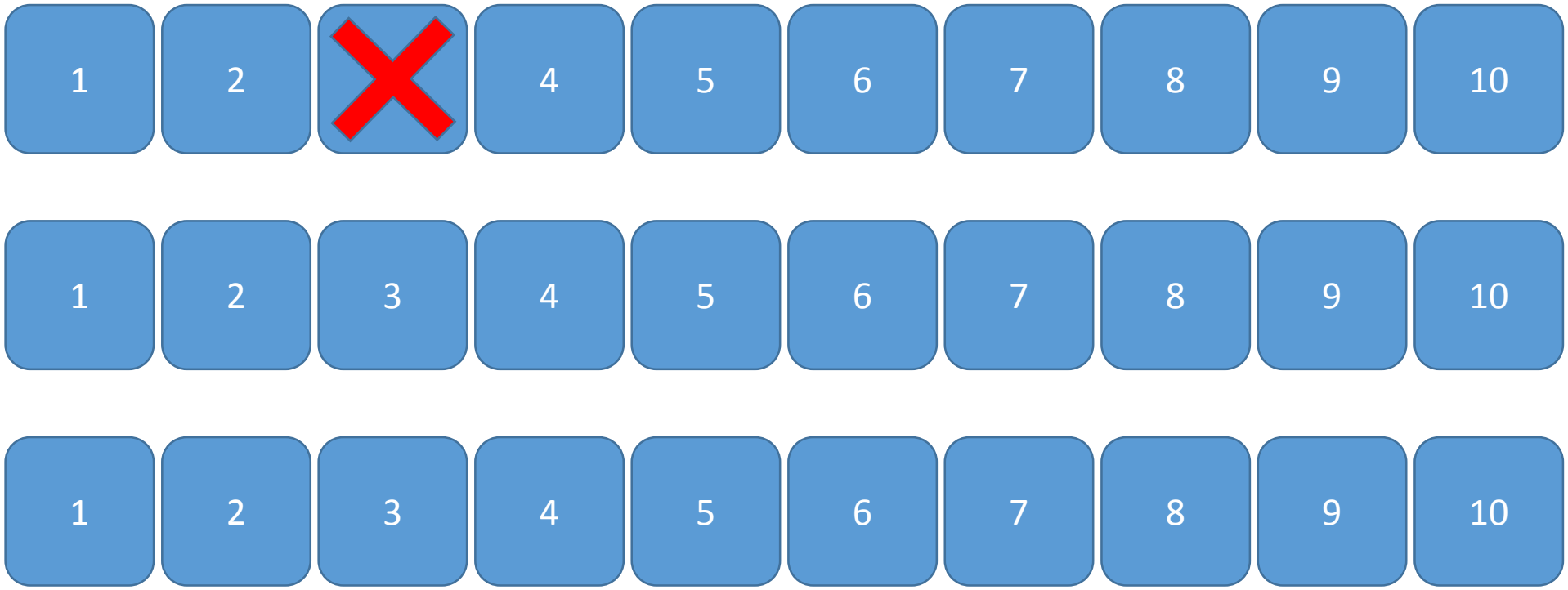


State of the Art



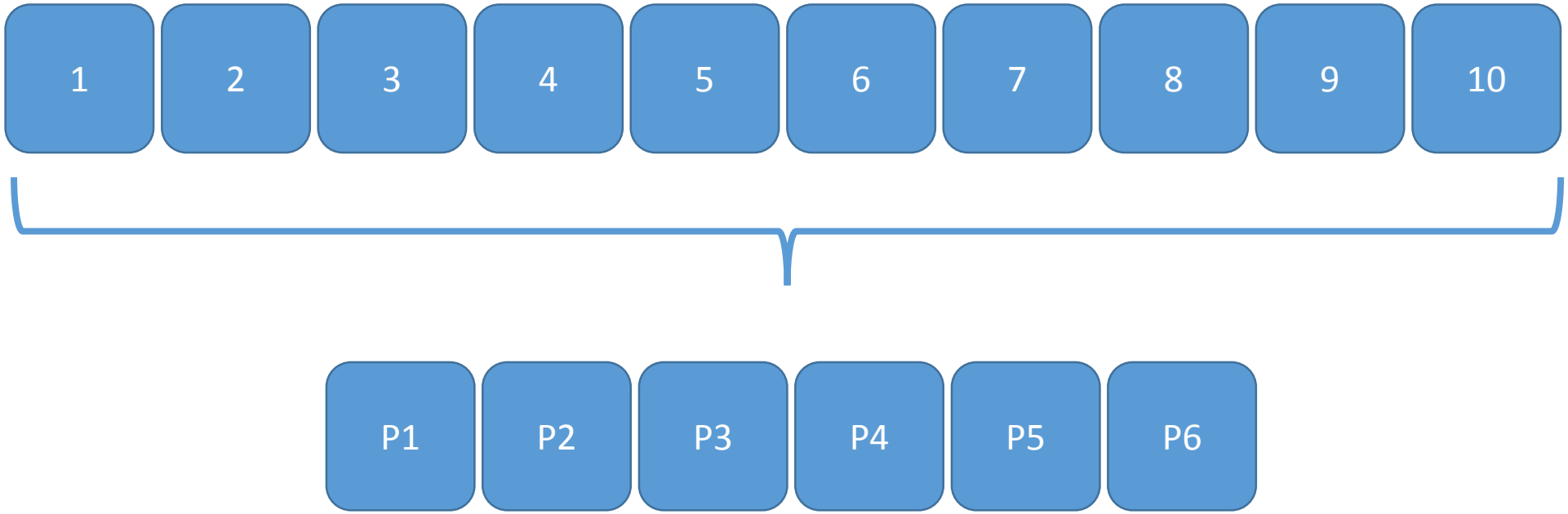
Overhead: 200%

State of the Art



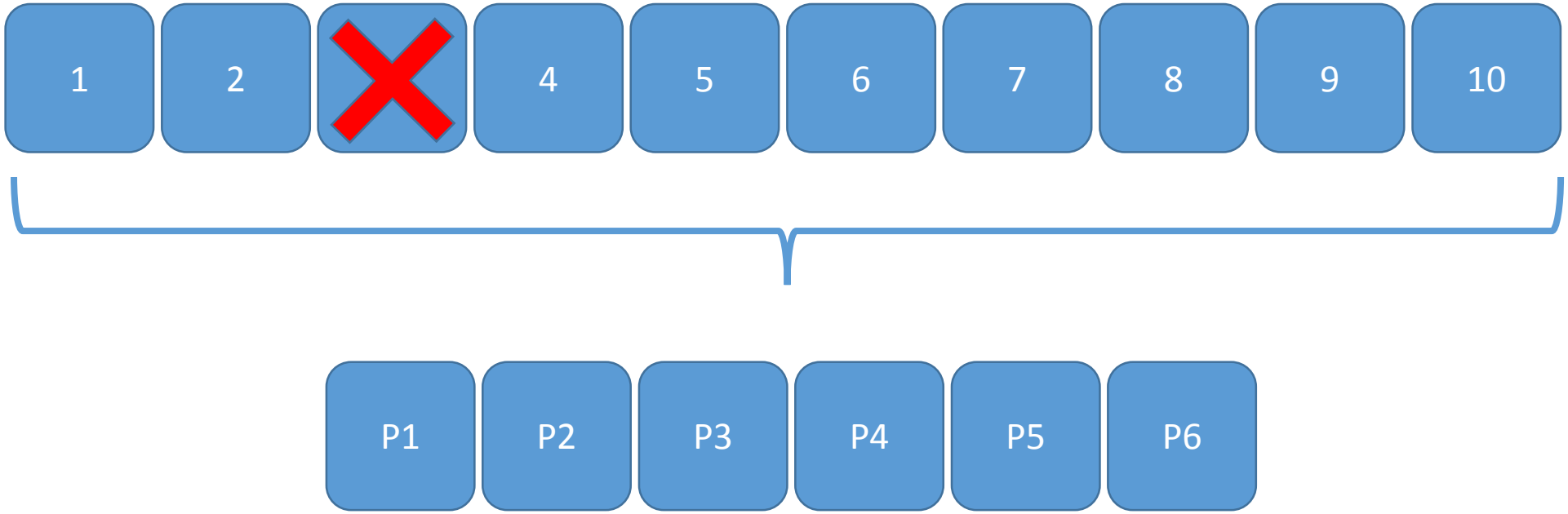
One failure results in one traffic unit

10:6 Code (Facebook)



Overhead: 60%

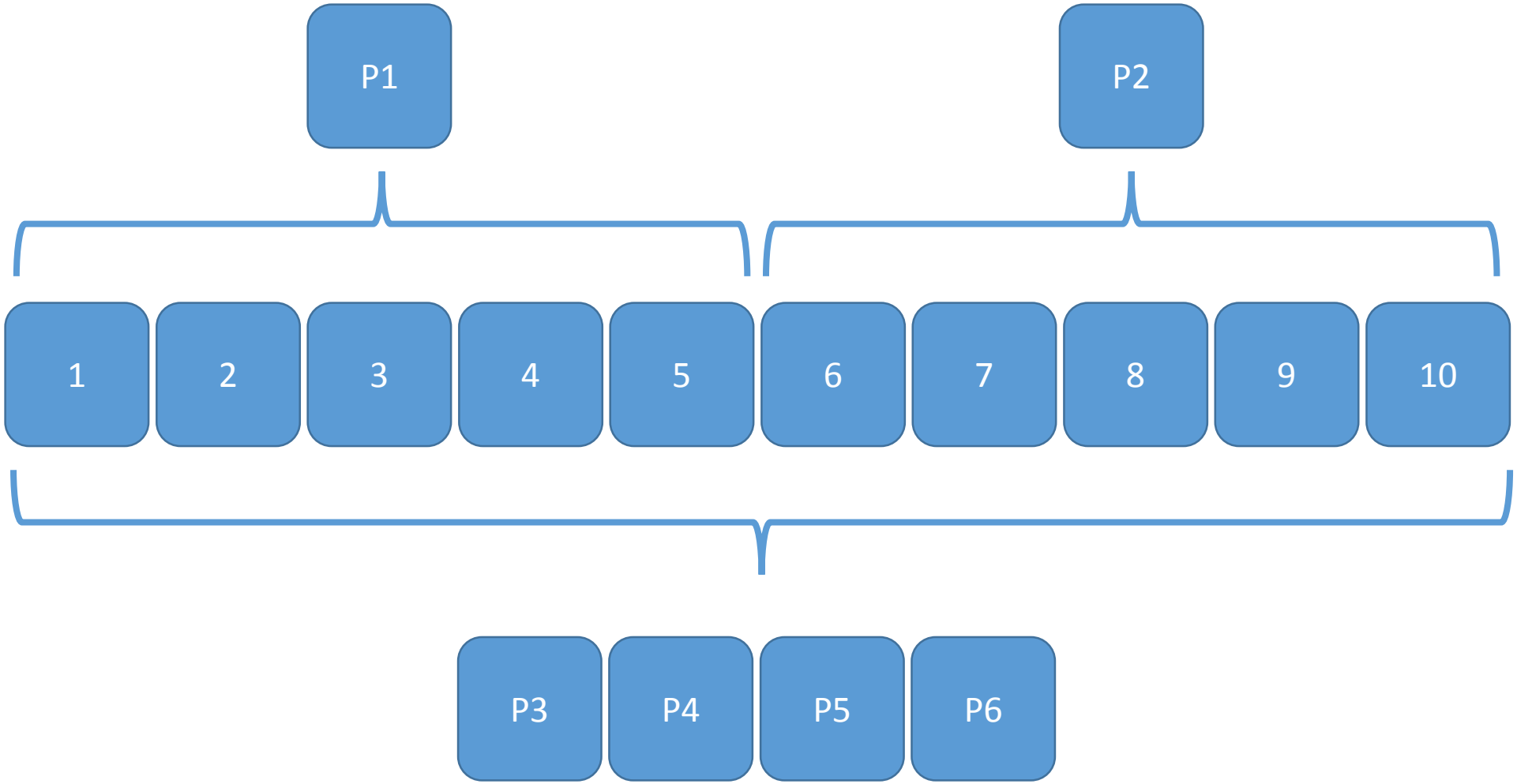
10:6 Code (Facebook)



One failure results in 10 traffic units

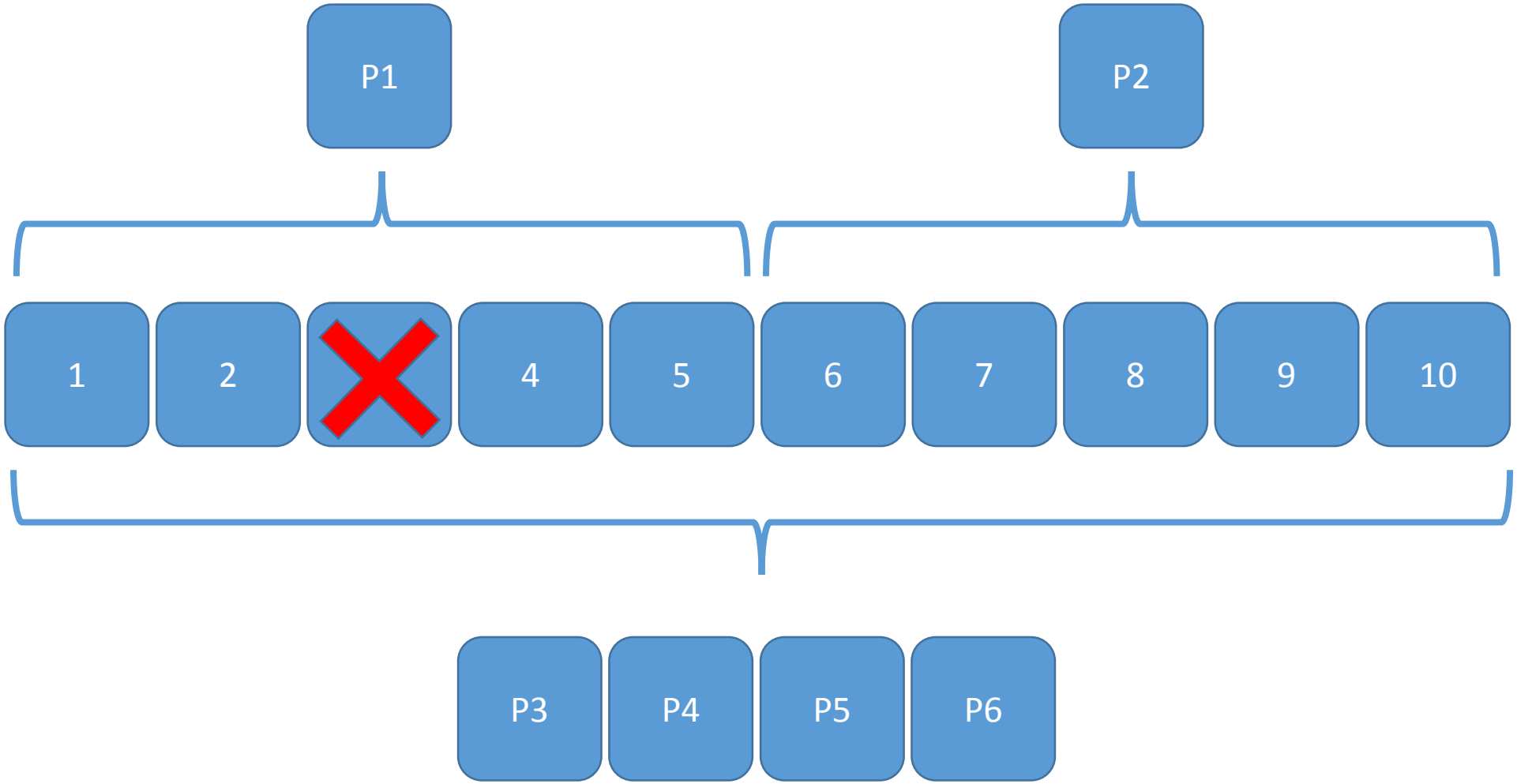
TECHNISCHE UNIVERSITÄT DRESDEN

Xorbas



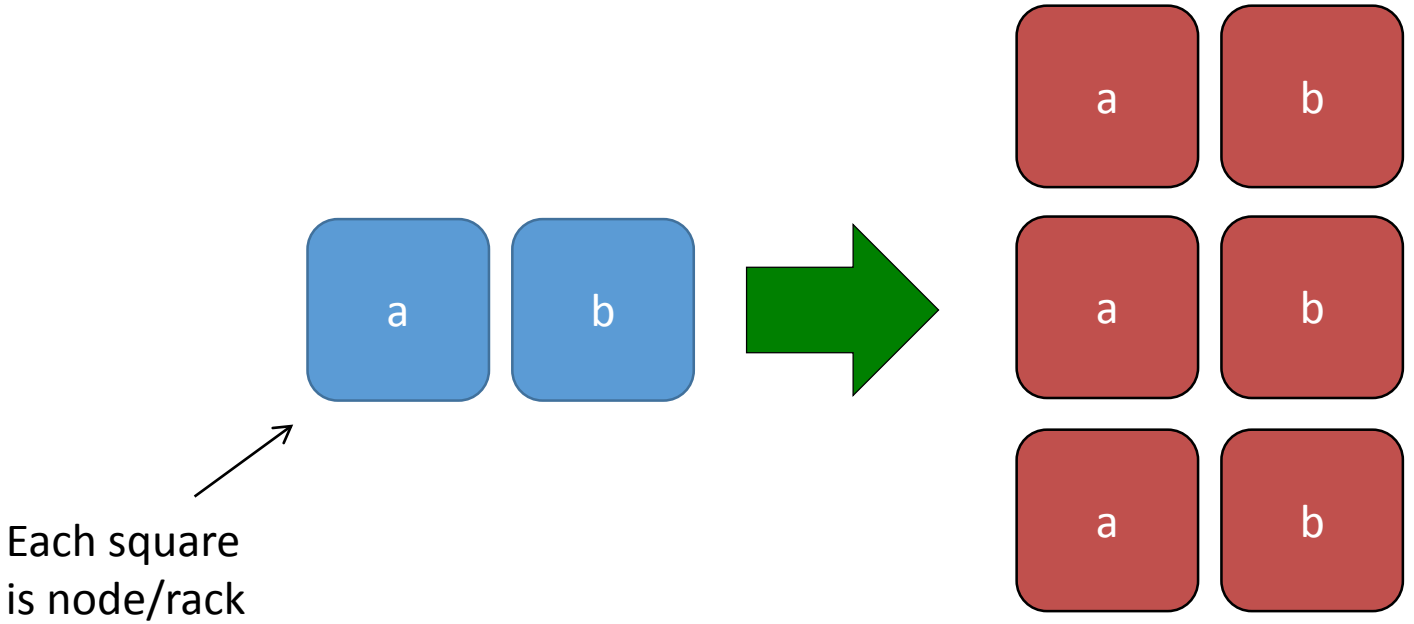
Overhead: 60%

Xorbas



One failure results in 5 traffic units

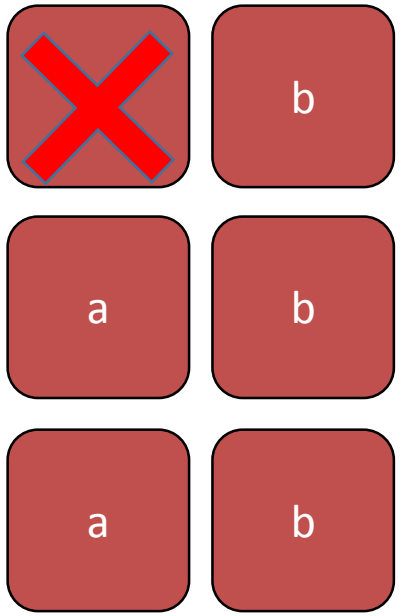
State of the Art: Replication



Overhead: 200%

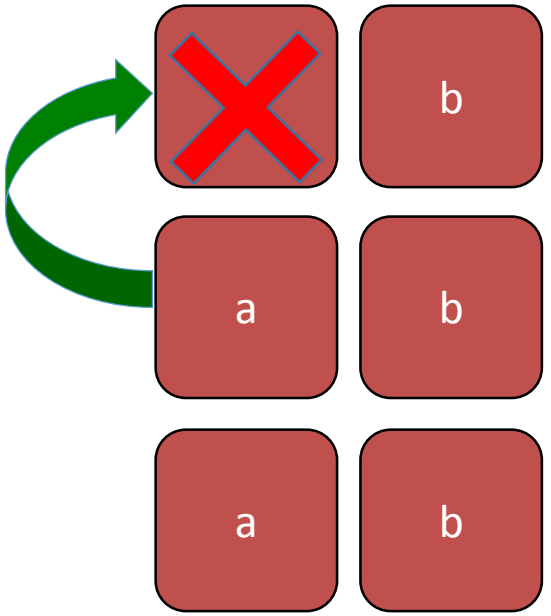
State of the Art: Replication

- Overhead: 200%
  - High storage cost
- Traffic to repair loss:
  - 1 unit
- Protection:
  - 4 losses (best case)
  - 2 losses (worst case)

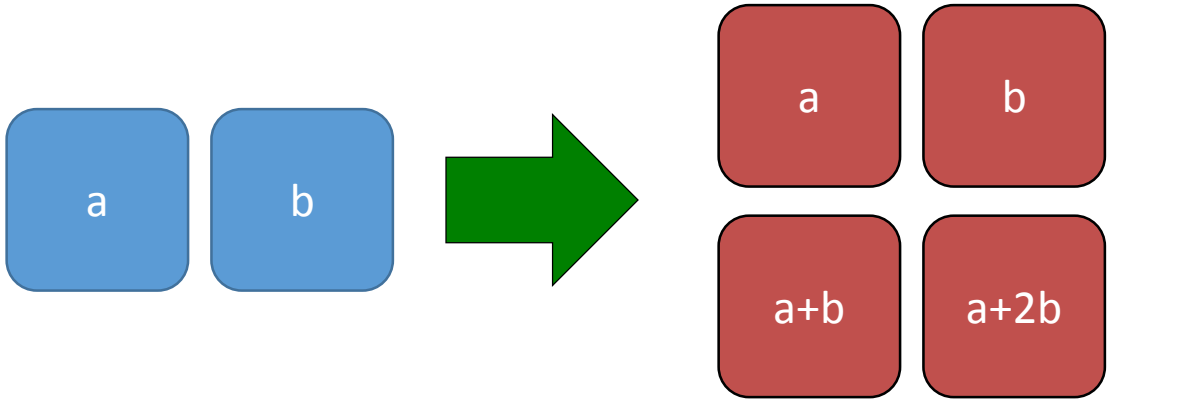


State of the Art: Replication

- Overhead: 200%
  - High storage cost
- Traffic to repair loss:
  - 1 unit
- Protection:
  - 4 losses (best case)
  - 2 losses (worst case)



State of the Art: RAID6

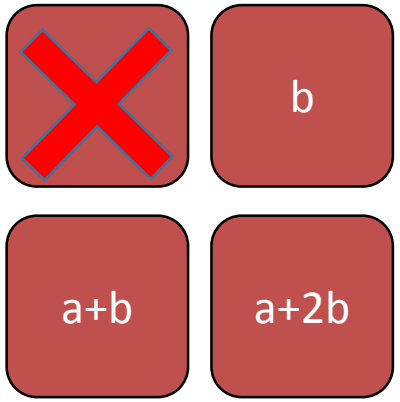


e.g. Reed-Solomon code

Overhead: 100%

State of the Art: RAID6

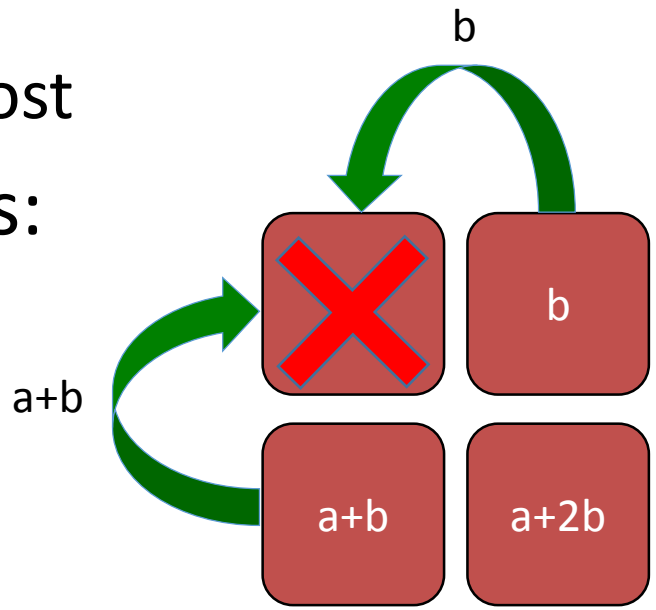
- Overhead: 100%
  - Medium storage cost
- Traffic to repair loss:
  - 2 units
- Protection:
  - 2 losses (any case)



e.g. Reed-Solomon code

State of the Art: RAID6

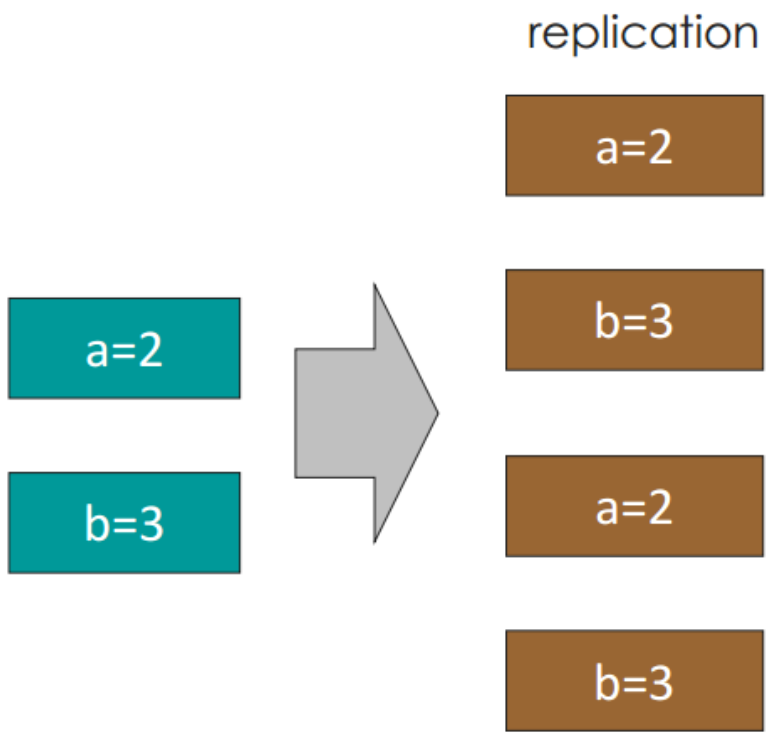
- Overhead: 100%
  - Medium storage cost
- Traffic to repair loss:
  - 2 units
- Protection:
  - 2 losses (any case)



e.g. Reed-Solomon code

$$\begin{array}{rr} + & a + b \\ & b \\ \hline & a \end{array}$$

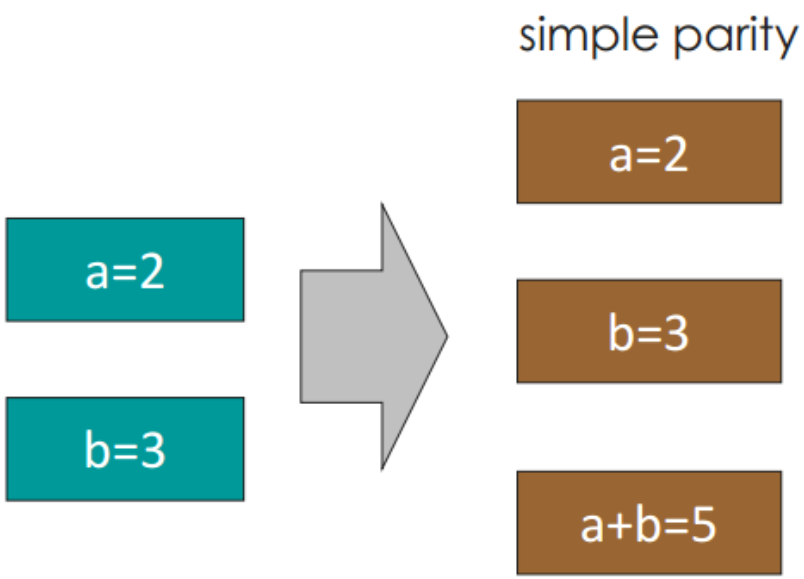
Simplest Erasure Coding – Replication



- # of data units k = 2
- # of code units n = 4
- # of tolerable unit losses
  - worst case $r_w = 1$
  - best case $r_b = 2$
- overhead: n/k = 2x

TECHNISCHE UNIVERSITÄT DRESDEN

Trivial Erasure Coding – Simple Parity



- # of data units k = 2
- # of code units n = 3
- # of tolerable unit losses
  - worst case $r_w = 1$
  - best case $r_b = 1$
- overhead: n/k = 1.5x

TECHNISCHE UNIVERSITÄT DRESDEN

MDS Codes and Systematic Codes

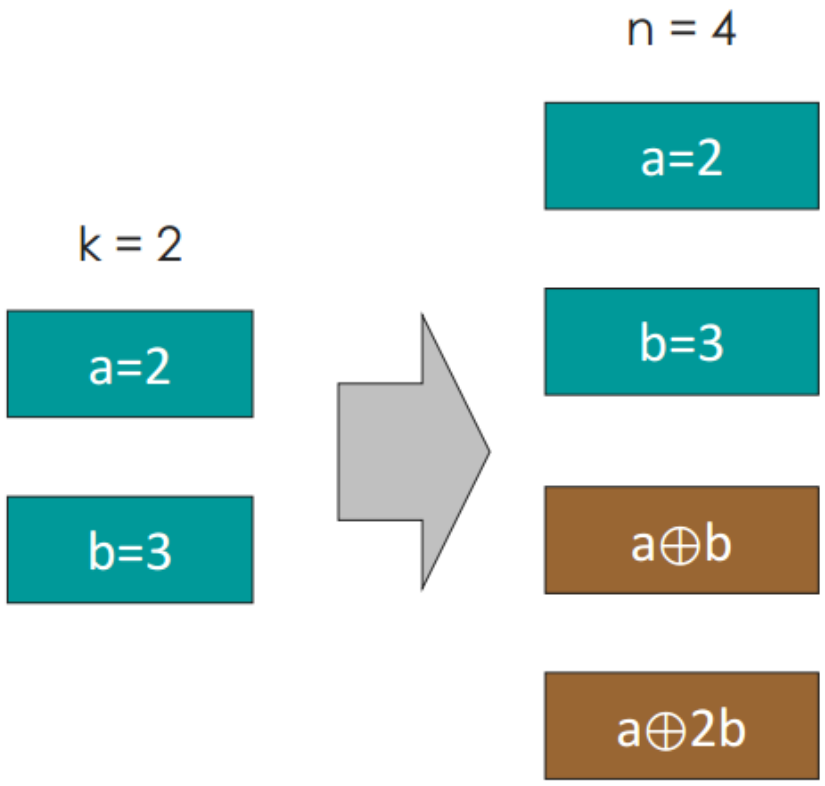
- MDS codes
  - Same number of tolerable losses (best vs. worst)
  - Optimal overhead
  - Extra computational complexity to perform coding

- Systematic codes
  - Direct access to the content without the need for coding if no losses
  - Coded information only needed with losses and then also increased traffic

TECHNISCHE UNIVERSITÄT DRESDEN

Reed-Solomon Example

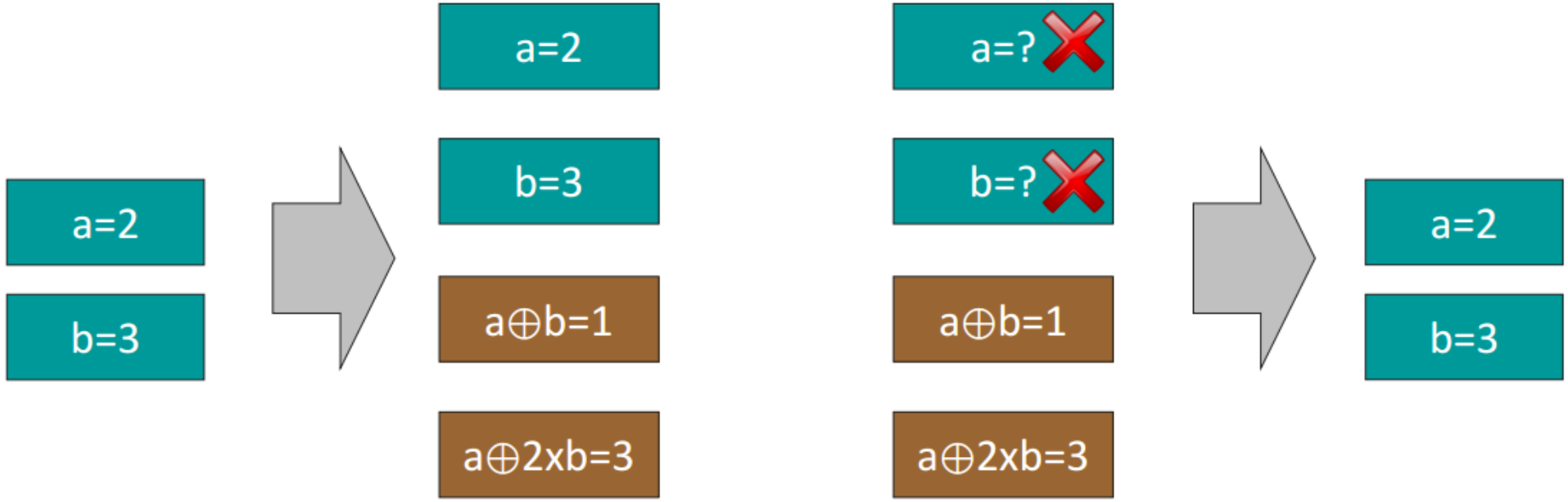
- The most important class of MDS and systematic codes
- Example



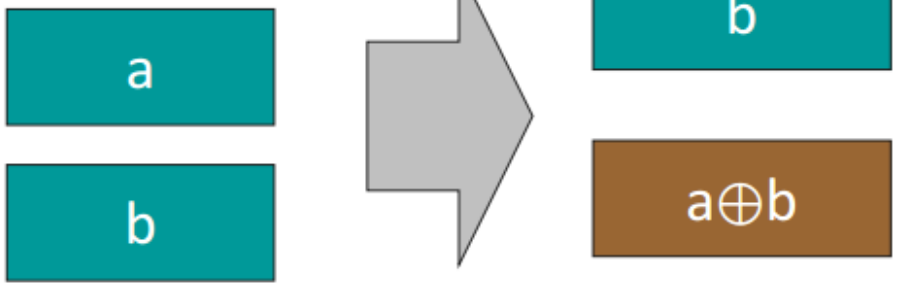
Reed-Solomon Example: Operations in FF

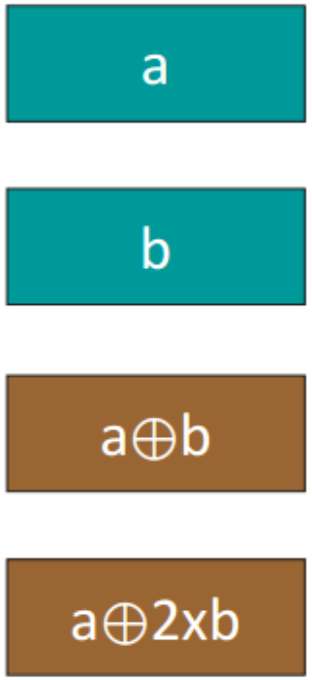
⊕	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

X	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2



Reed-Solomon Example: Operations in FF





- a & b can be of any size
  - Network packet (1KB)
  - Storage disk (1TB)
- ⊕ is very fast on CPU
- What about multiplication?

TECHNISCHE UNIVERSITÄT DRESDEN

Repair Problem

TECHNISCHE UNIVERSITÄT DRESDEN

More General Reed-Solomon

- Can split data into k fragments and generate n fragments
- Any k fragments would be enough to recover

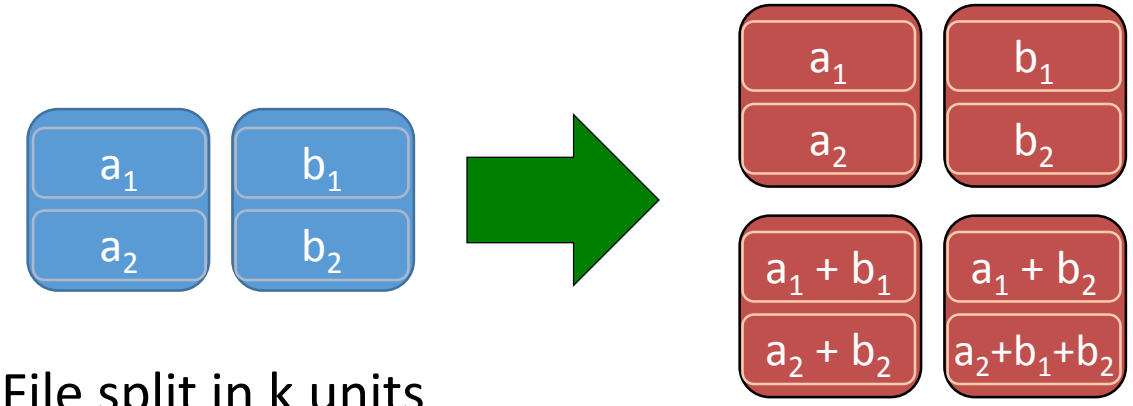
- Overhead: (n-k)/k
- Traffic for repair: k
- Protection: can recover from n – k losses

TECHNISCHE UNIVERSITÄT DRESDEN

Regenerating Codes

- Goal: Minimize repair traffic with same fault tolerance as RS
  - Faster repair
  - Inherent trade-off of storage and traffic cost
  - Network coding is key: recoding needed at the nodes/racks before sending

Regenerating Codes

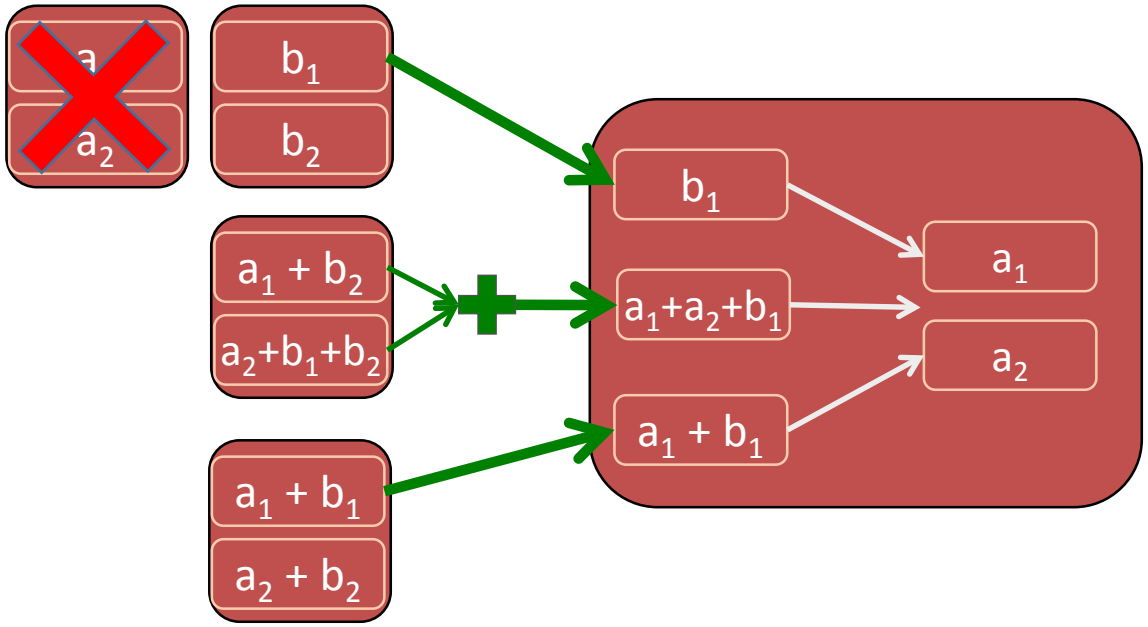


File split in k units

Overhead: 100%

Regenerating Codes

- Overhead: 100%
- Protection: 2 losses of nodes/racks
- Repair traffic: 1.5 units → ¾ of file size



Regenerating Codes

- For a file of size *B* bits, split into *k* pieces and encoded into *n* pieces
- Each piece is stored in a disk with *B/k* bits per disk
- **Theorem:** it is possible to **functionally** repair a code by communicating

$$\frac{n-1}{n-k}\frac{B}{k} \quad \text{bits}$$

- As opposed to transmitting *B* bits (naive cost)

- **Previous Example:** For $n = 4$, $k = 2$, then at least ¾ *B* bits are needed (and we used exactly that)

New Challenges in Storing

- So far the structure has been static
- But new use cases require dynamic / versatile codes
  - Edge caching
  - Storage in mobile nodes

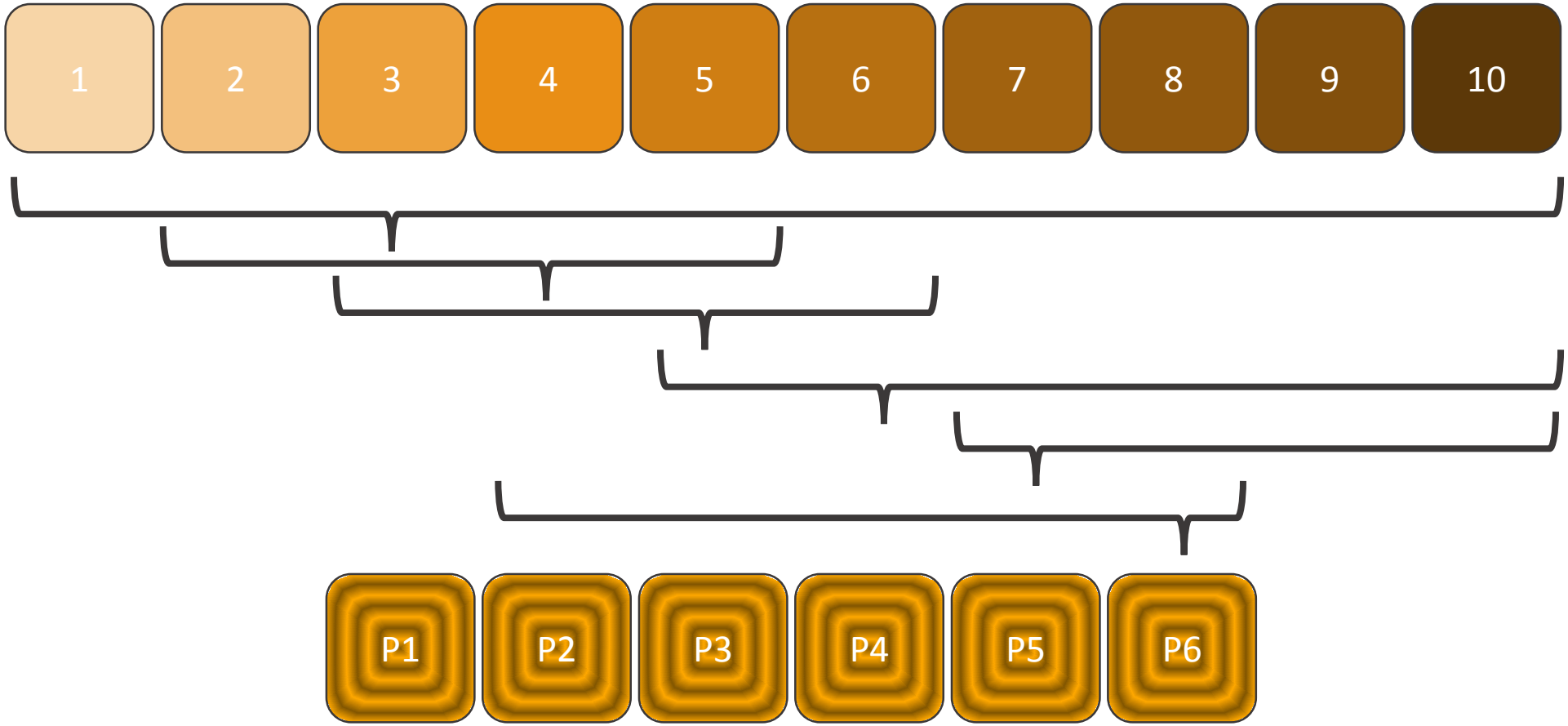
TECHNISCHE UNIVERSITÄT DRESDEN

What RLNC can do

- New cloud storages can be filled with available information. Some information might be lost or not available.
- Filling storage on arrival without storing data in memory before decoding.

TECHNISCHE UNIVERSITÄT DRESDEN

RLNC



Overhead: scalable

TECHNISCHE UNIVERSITÄT DRESDEN

What we looked at

- Reliability
- Privacy and security
- Storage space
- Costs
- Download speed
- Amount of transferred data
- Computational overhead of network coding

TECHNISCHE UNIVERSITÄT DRESDEN

Reliability

- Failures are not uncommon in the world of cloud computing:
  - Google (partial): 16th of August 2013 , 18th-19th of March 2013, 11th of August 2008
  - Microsoft Azure: 22nd of February 2013, 13th of March 2009
  - Amazon S3: 31st of January 2013, 20th of July 2008
  - Dropbox: 10 th of January 2013
  - …
- Permanent data loss is possible too:
  - Microsoft & T-Mobile Sidekick: 11th of October 2009
- However, the chance that more than one provider is down at a given time is very low.
- Solution: distribute the data to multiple providers with some added redundancy.

Reliability

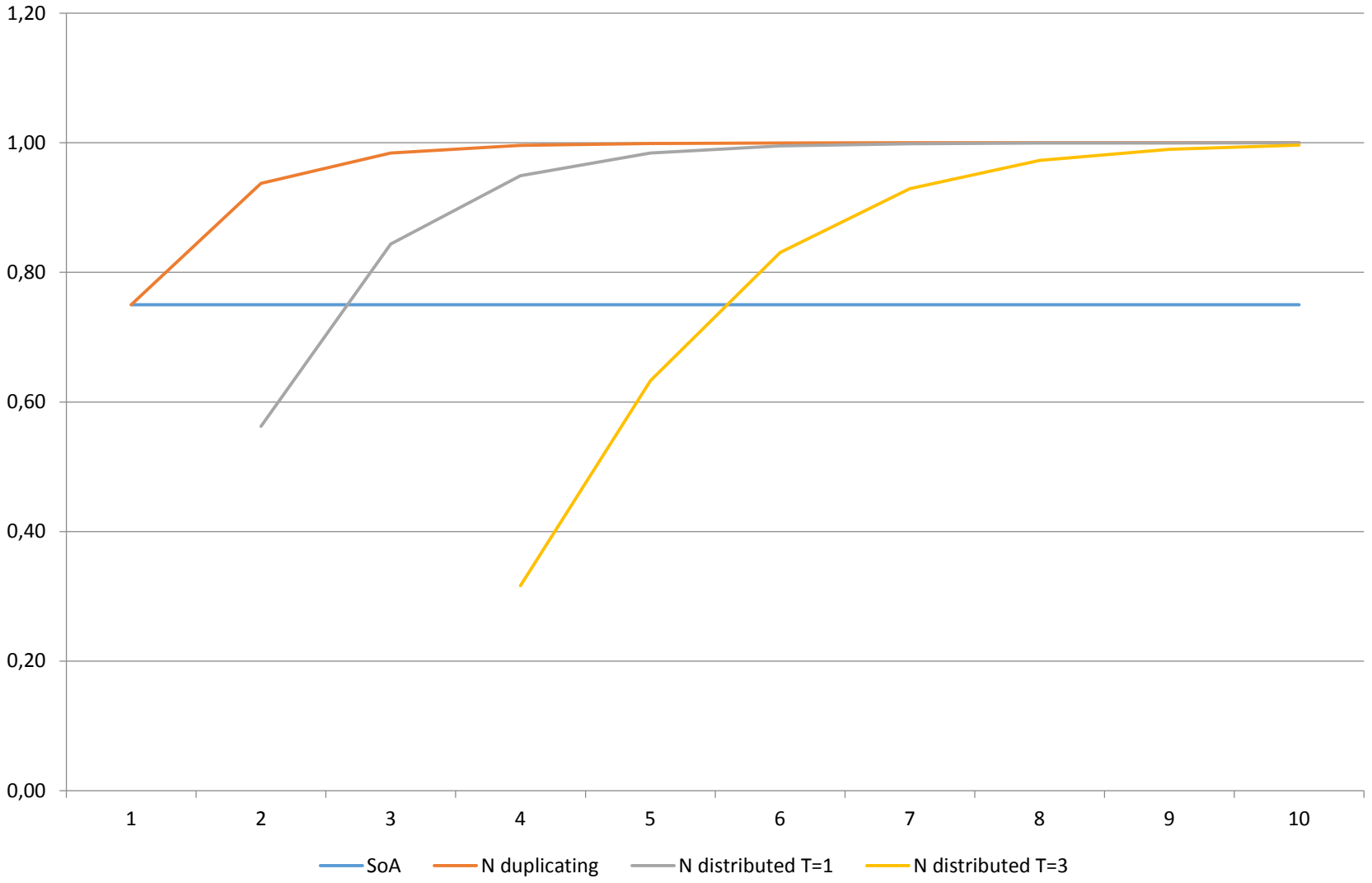
- It is relatively simple to calculate the probability that the data is not available at a given time:

$$\mathcal{P}(T, N, p) = 1 - \sum_{t=0}^{t=T} \binom{N}{t} \cdot (1-p)^{N-t} \cdot p^t$$

  - N – number of clouds
  - T – number of simultaneous cloud failure to be able to sustain
  - p – the probability that a provider is unavailable

TECHNISCHE UNIVERSITÄT DRESDEN

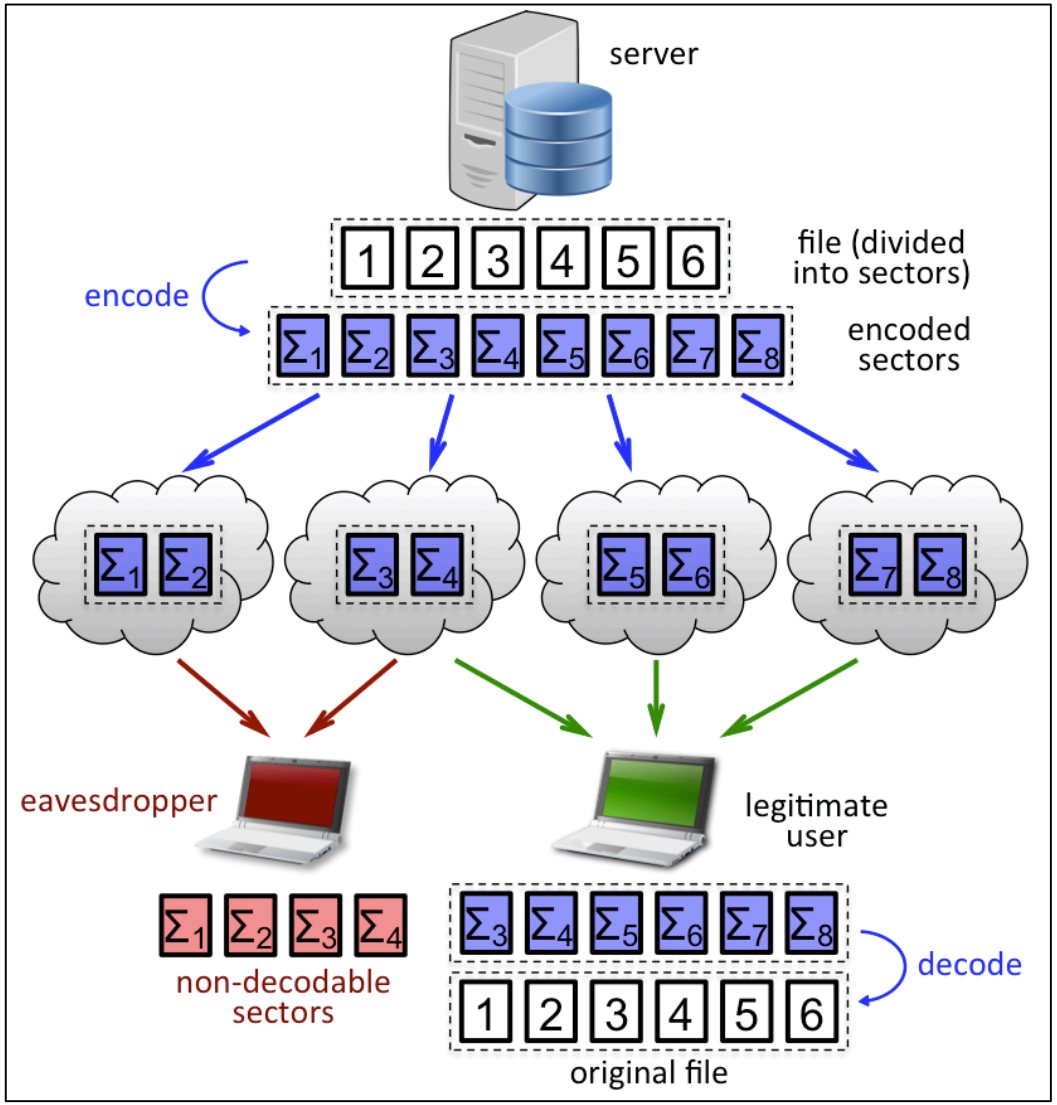
Reliability



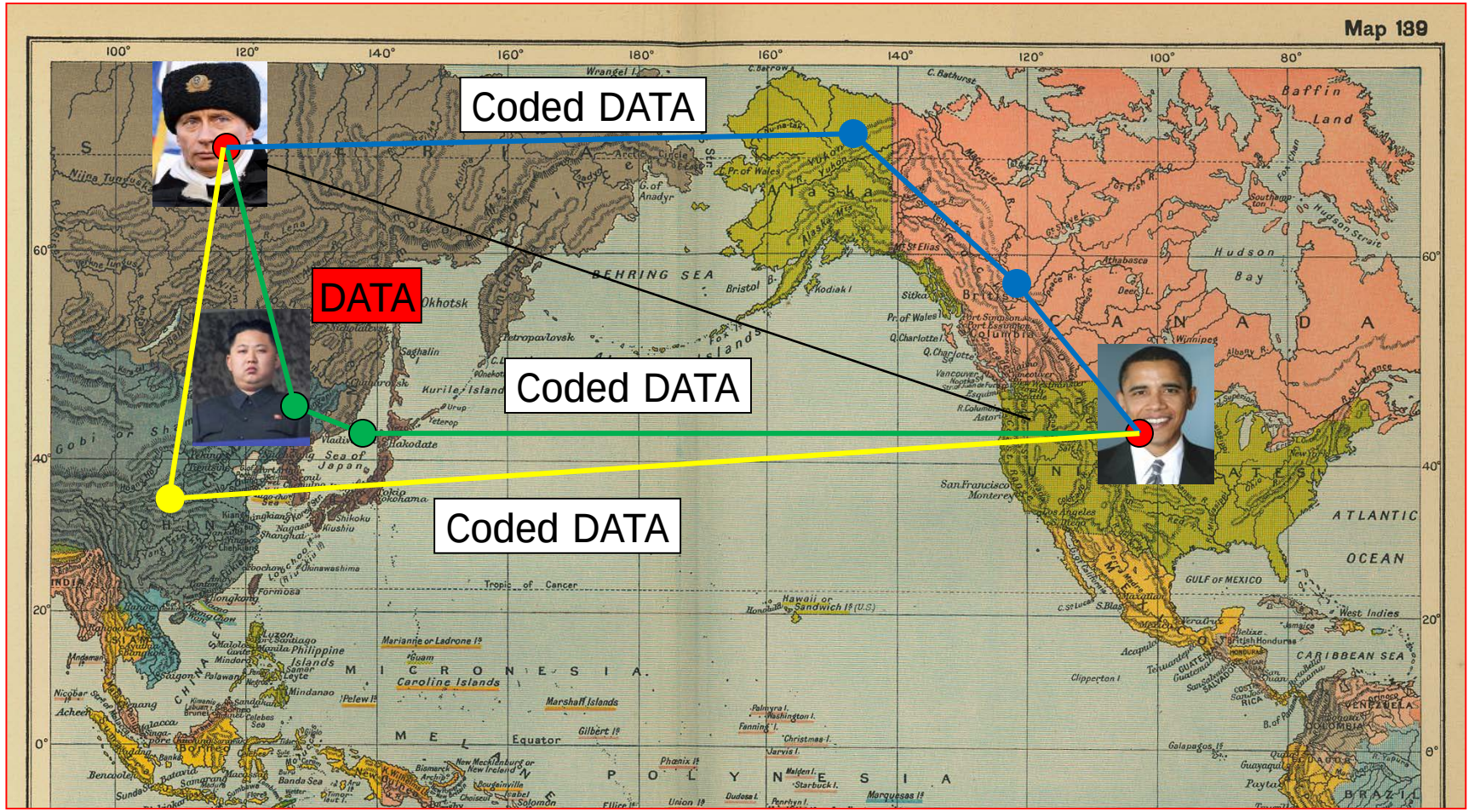
Privacy and Security

- One of the biggest hurdles holding back cloud computing is lack of trust in the providers.
- Security breaches:
  - Sony PlayStation Network: 17th and 19th of April 2011 – 77 million accounts stolen
  - Smaller breaches have occurred with most cloud computing providers
- Government agencies have access to data stored by cloud services in some countries:
  - PRISM
- By using network coding, it is possible to force the attacker to need to compromise several of the providers to access any of the data.
  - The number of providers that need to be hacked can be set arbitrarily.

Coding as a Measure Content of Protection



Some insights for security



Storage costs

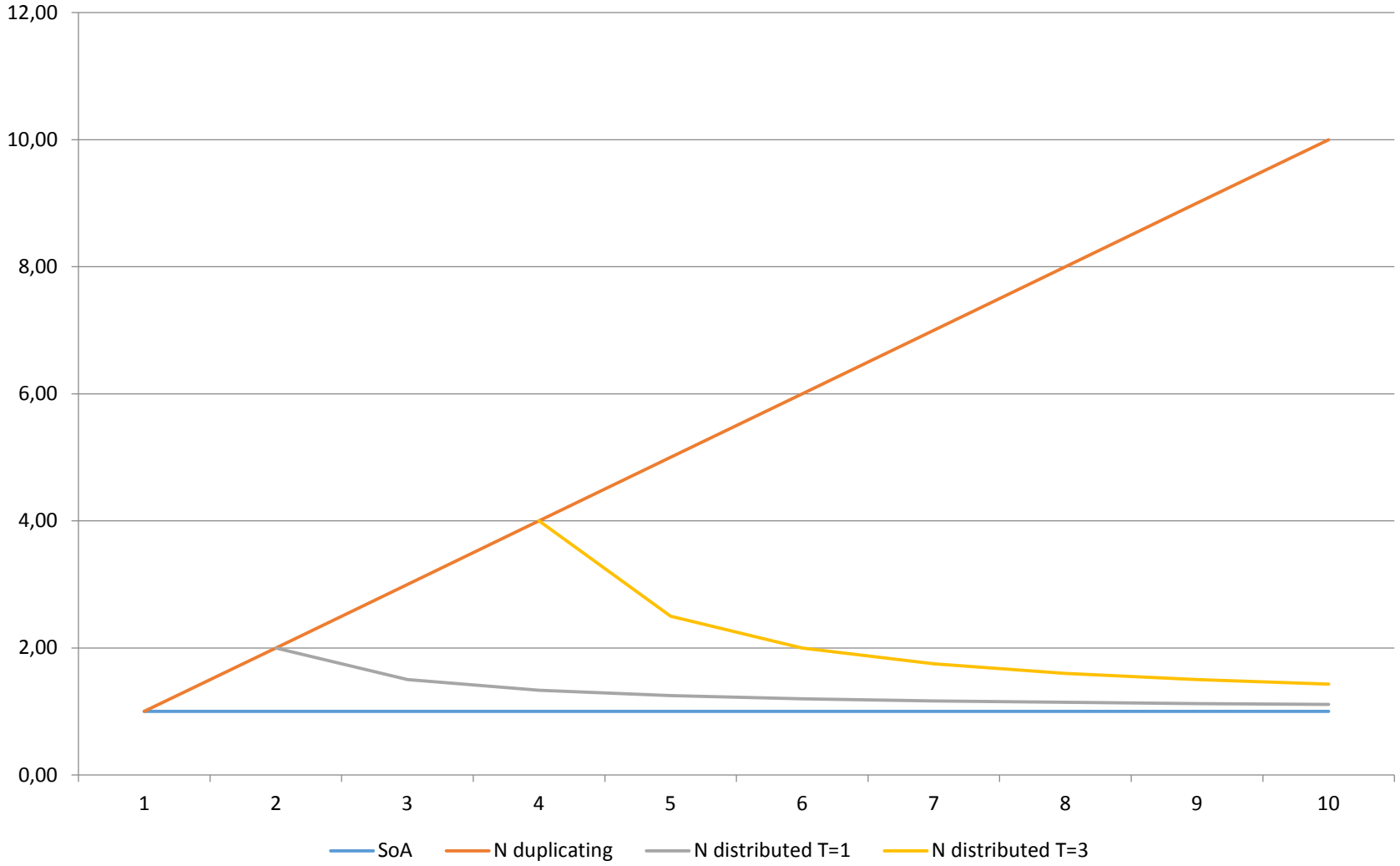
- Each provider grants some free storage space. With our approach you can add up the capacities.
- Reliability can be easily quantified in terms of costs. The required storage space is:

$$H(N,T,F,P) = \frac{N}{N-T} \cdot F = F + F \cdot \frac{T}{N-T}$$

  - F – file size in bytes
- Clearly if $N >> T$, the storage cost becomes negligible.

TECHNISCHE UNIVERSITÄT DRESDEN

Storage/Cost



Total costs

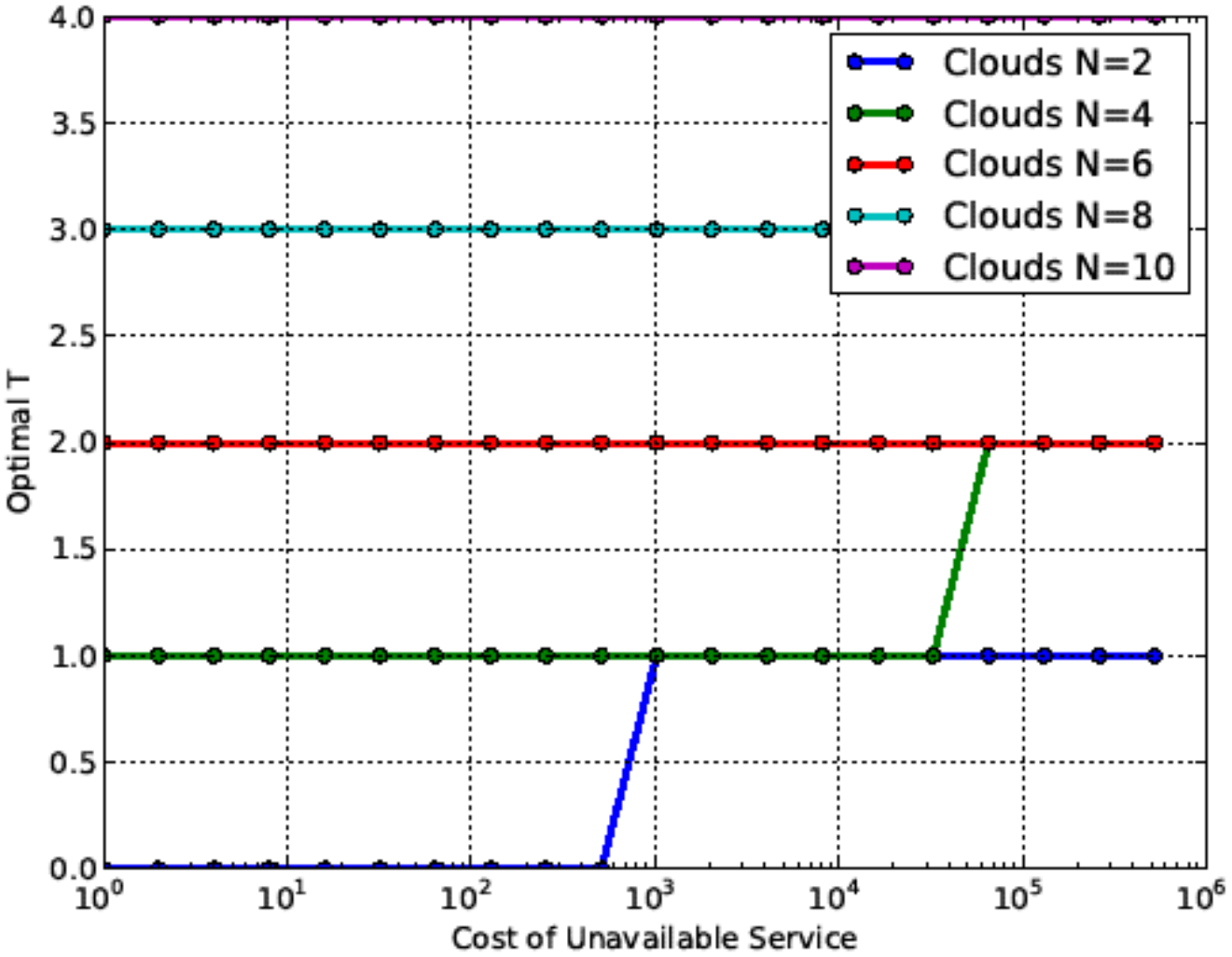
- The goal is to find the number of redundant clouds (T) for which the total cost is minimal:

$$\min_{T} H(N, T, F, P) + U_c \cdot \mathcal{P}(T, N, \{p_i\})$$

  - Uc – cost of data being unavailable
  - H(N,T,F,P) –storage costs

TECHNISCHE UNIVERSITÄT DRESDEN

Finding the optimal T



Download speed

- Considering clouds with various download rates (R), the fraction of file requested from cloud c_i is:

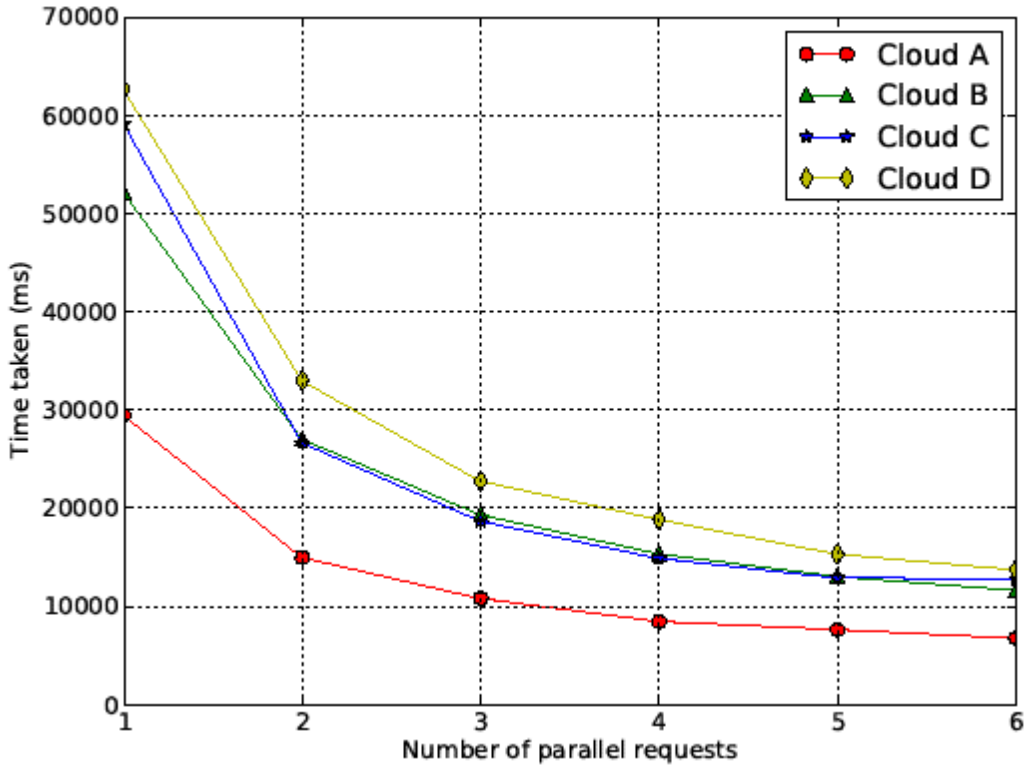
$$\alpha_i = \frac{R_{(c_i,u)}}{\sum_{i=1}^{N} R_{(c_i,u)}}$$

$$P_i = \max\left(\frac{1}{N-T}, \alpha_i\right) = \max\left(\frac{1}{N-T}, \frac{R_{(c_i,u)}}{\sum_{i=1}^{N} R_{(c_i,u)}}\right)$$

- If we chose to optimize for download speed whilst still ensure a give level of reliability, the fraction of data stored in cloud should c_i be at least:

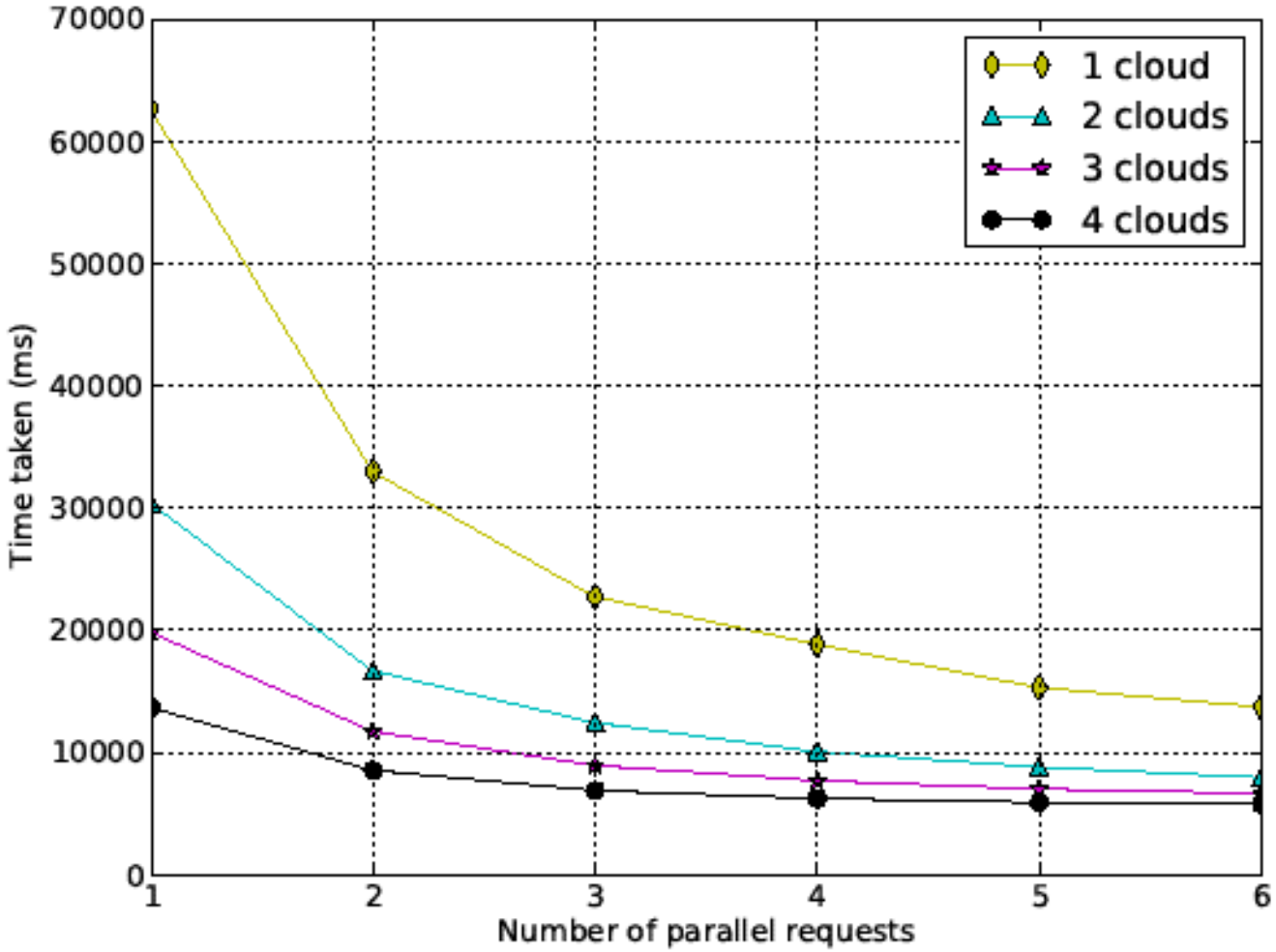
Download speed – individual clouds

- We looked at how each cloud performs on its own. 16 MB file, packets of 512k.



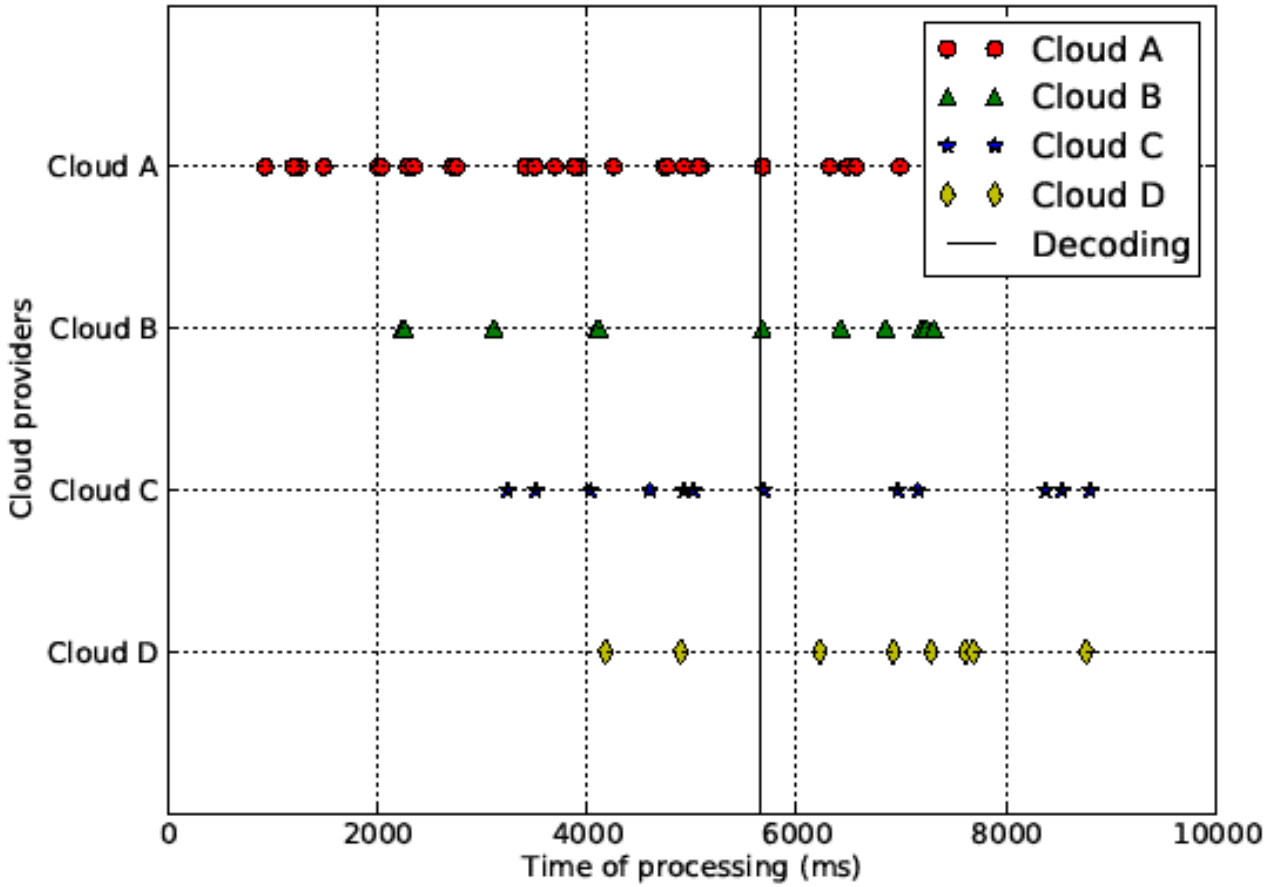
Why use multiple clouds?

- Collaboration.



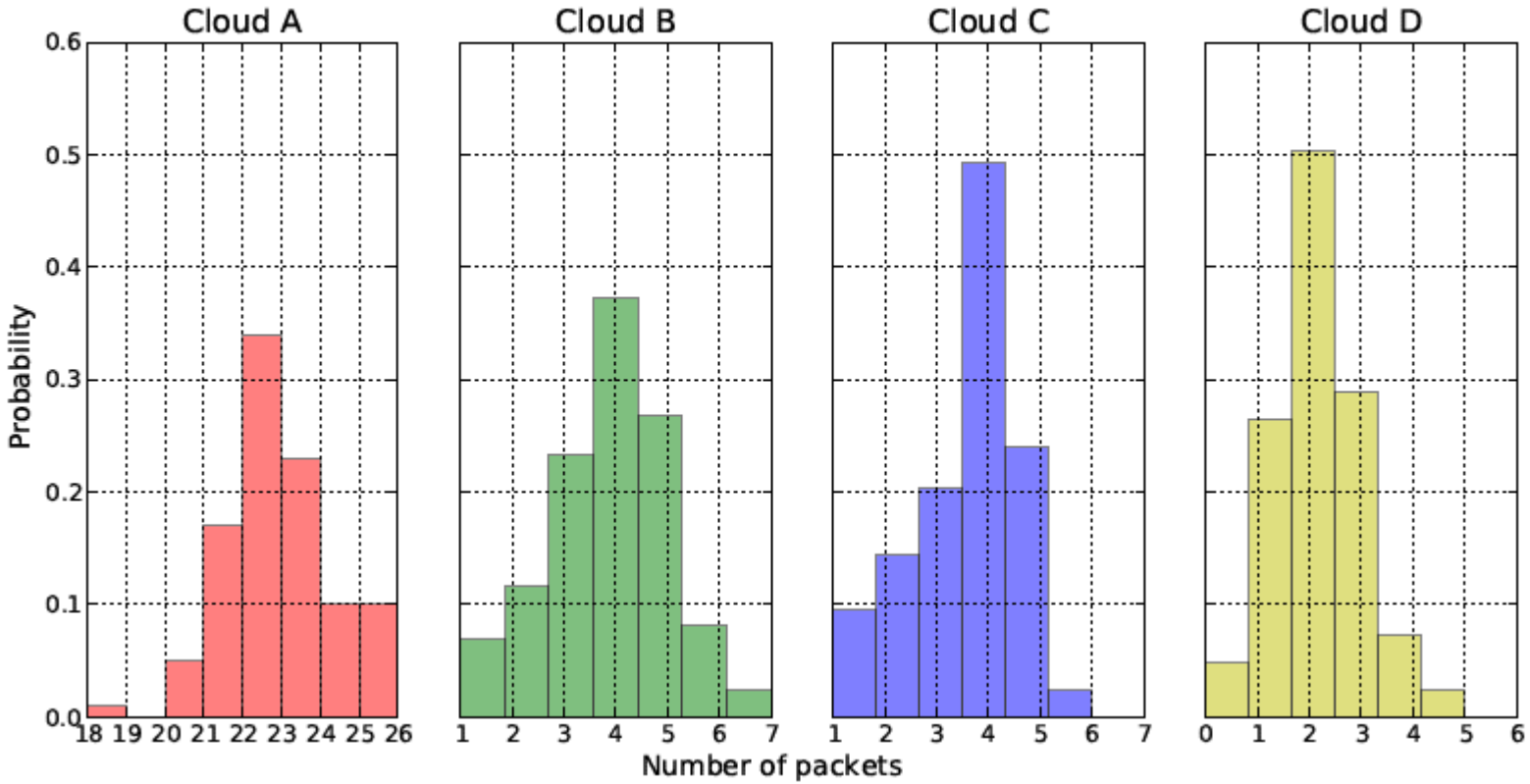
Why use network coding?

- Collaboration is usually not simple.

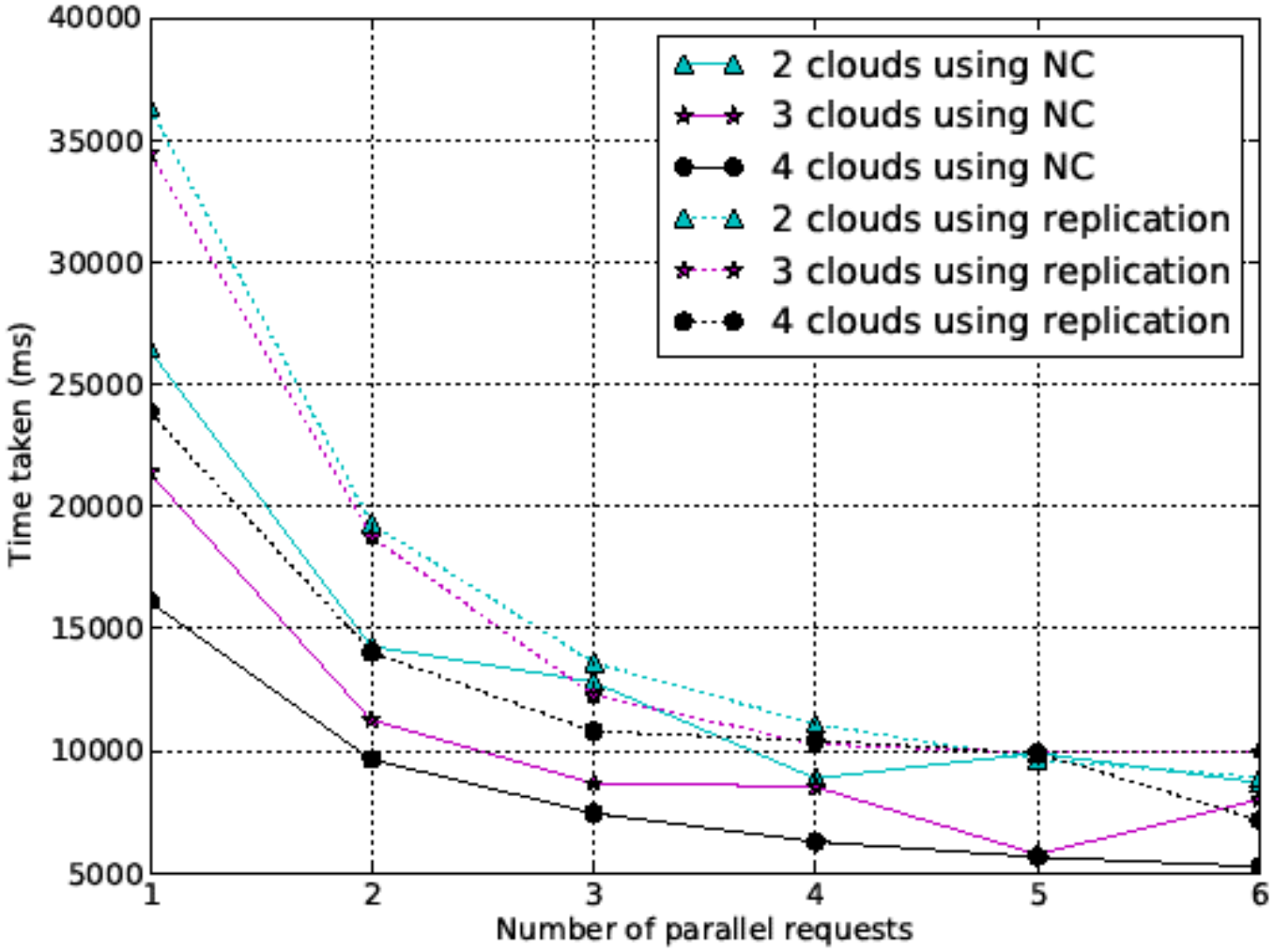


Why use network coding?

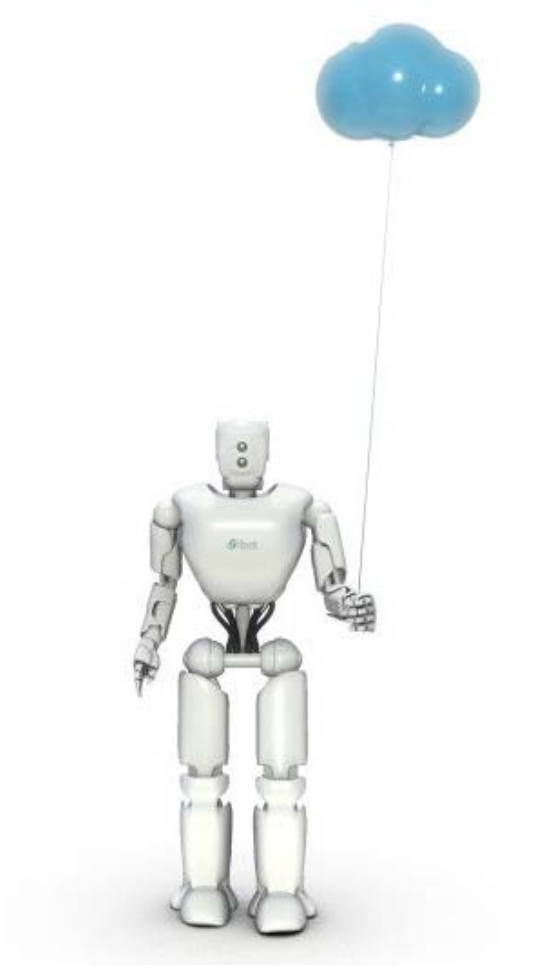
- Conditions change.



Comparing NC to replication redundancy

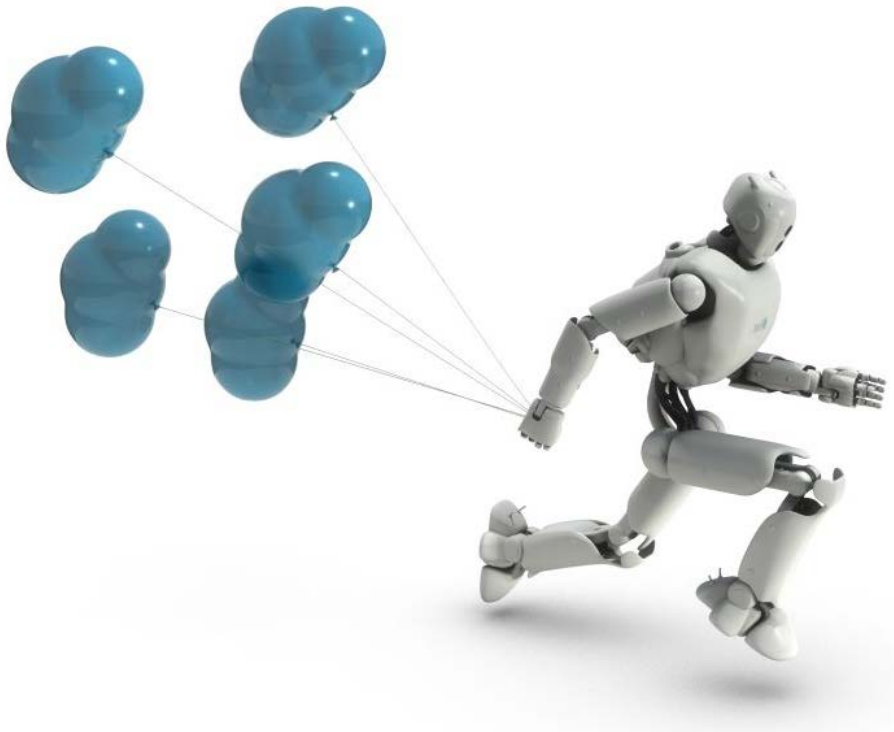


Cloud Evolution



Single/Static

Distributed/Static

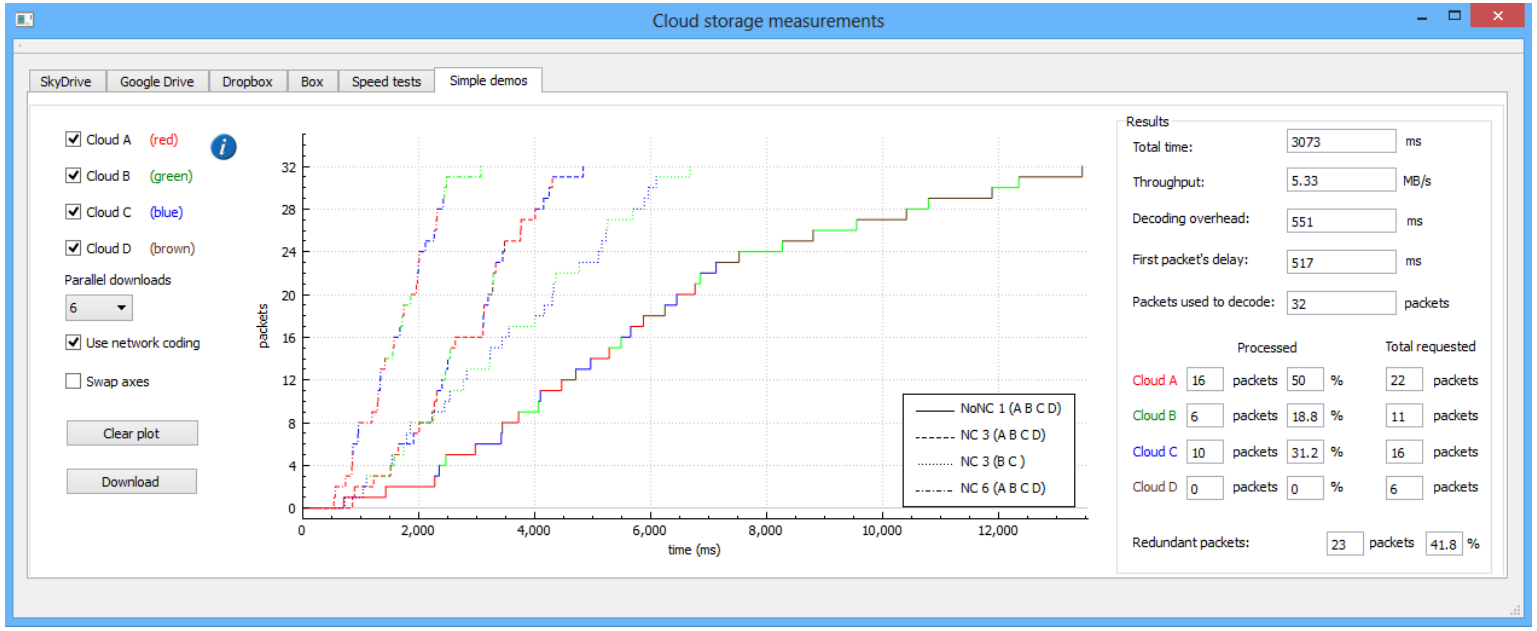


Distributed/Agile

Example: Distributed Cloud

Measurement campaign

- Implementation using commercial cloud solution (Dropbox, Box, Skydrive, and Google Drive) to:
  - showcase theoretical results
  - do practical measurements



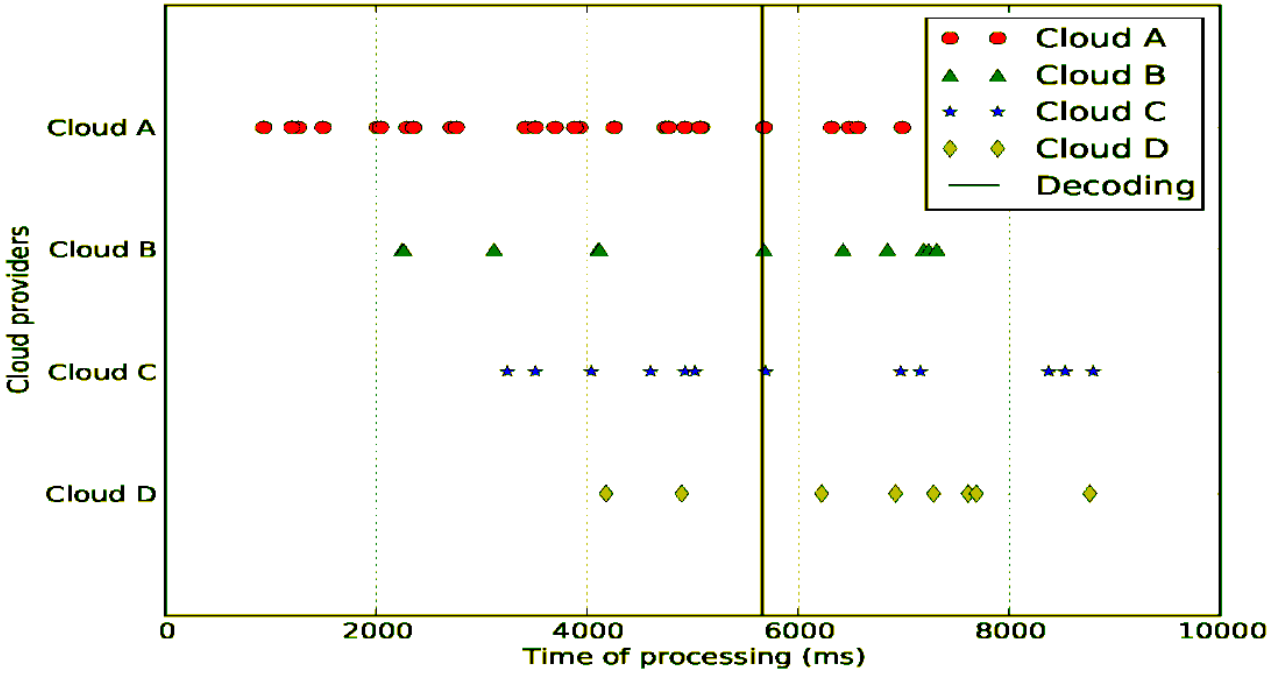
- Simple UI to be able to quickly explore different scenarios.

TECHNISCHE UNIVERSITÄT DRESDEN

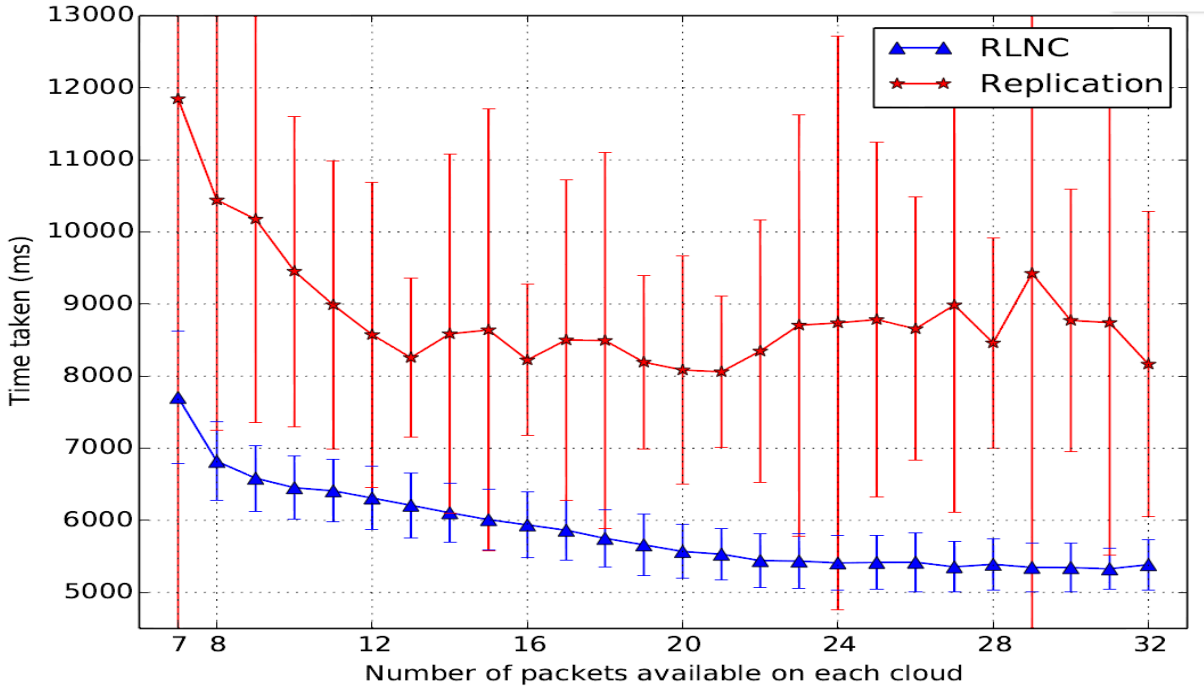
DEMO

Example: Distributed Cloud

- Heterogenity (4 clouds)
- Clouds behave differently



- Speed-Up (5 clouds)
- RLNC does not need full degree of freedom



- M. Sipos, F.H.P. Fitzek, D. Lucani, and M.V. Pedersen, “Dynamic Allocation and Efficient Distribution of Data Among Multiple Clouds Using Network Coding,” in IEEE International Conference on Cloud Networking (IEEE CloudNet’14), Oct. 2014.
- M. Sipos, F.H.P. Fitzek, D. Lucani, and M.V. Pedersen, “Distributed Cloud Storage Using Network Coding,” in IEEE Consumer Communication and Networking Conference, Jan. 2014.

TECHNISCHE UNIVERSITÄT DRESDEN

Network Coding Storage

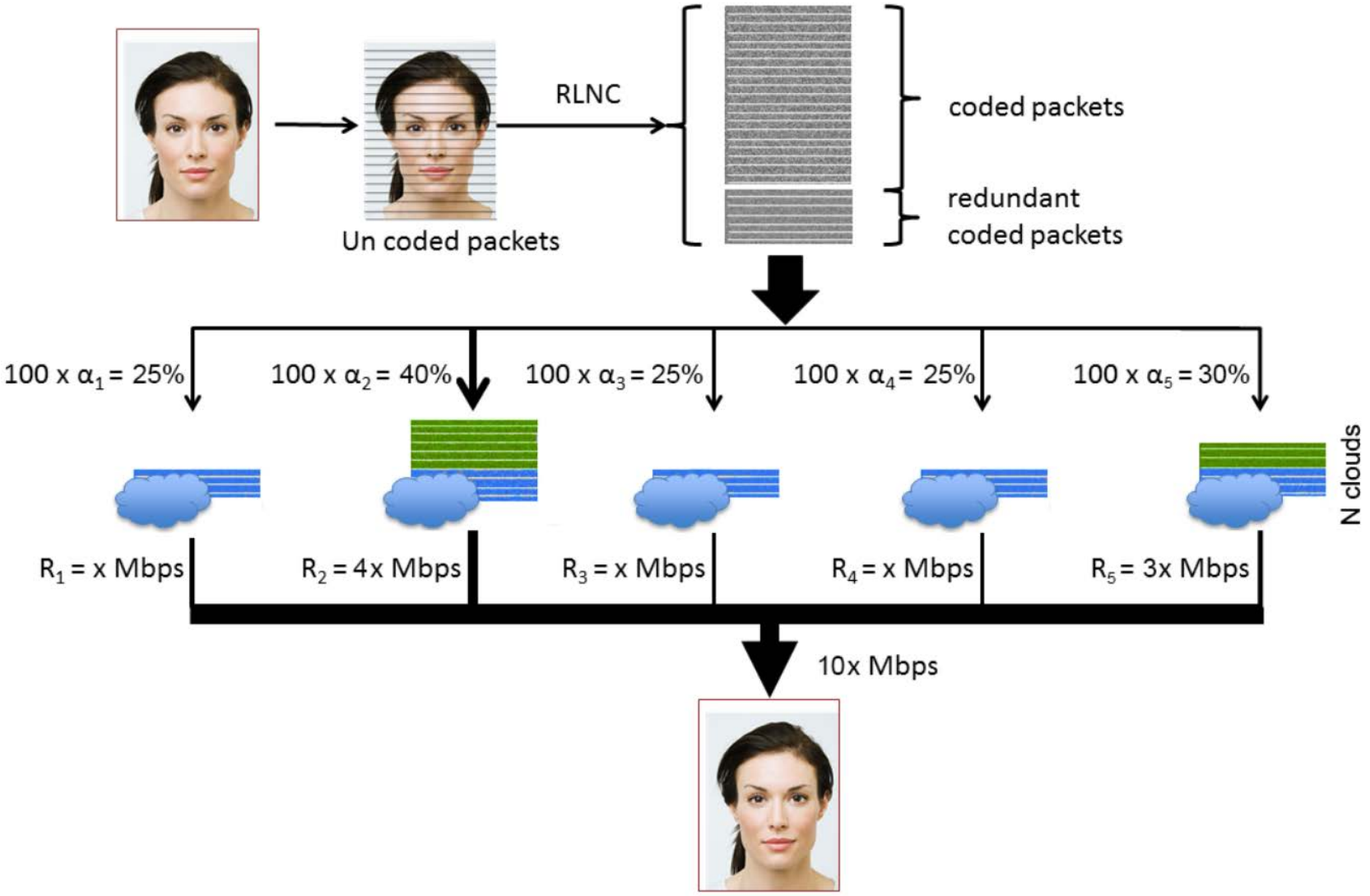
TECHNISCHE UNIVERSITÄT DRESDEN

Storage Benefits in Dynamic Settings

Comparing a FULLY decentralized approach (no coding, RS, RLNC) versus a centralized version (no coding, RS).

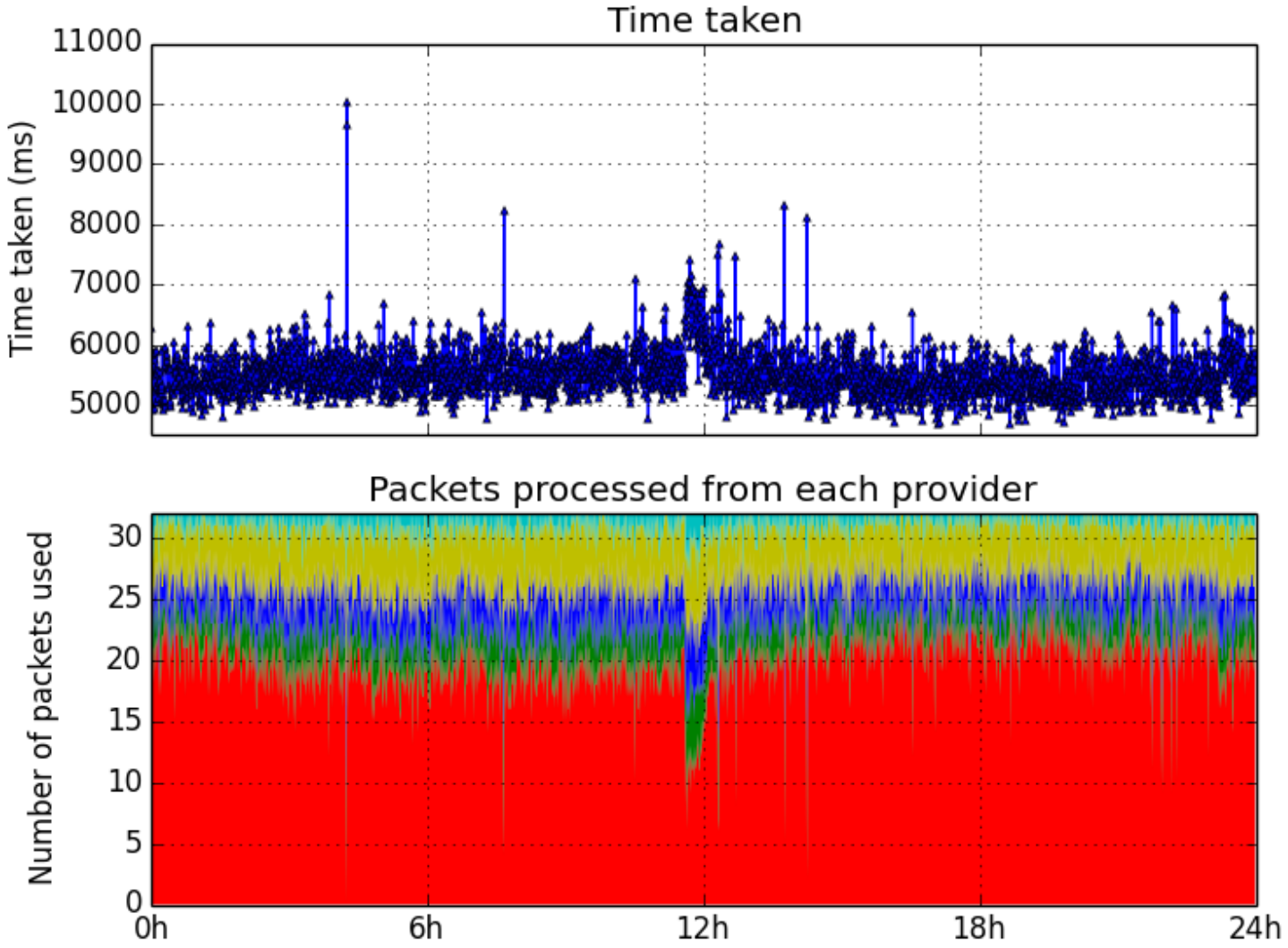
Dynamic: Not all racks are available temporarily (load, traffic, power)

Dynamics in Distributed Clouds

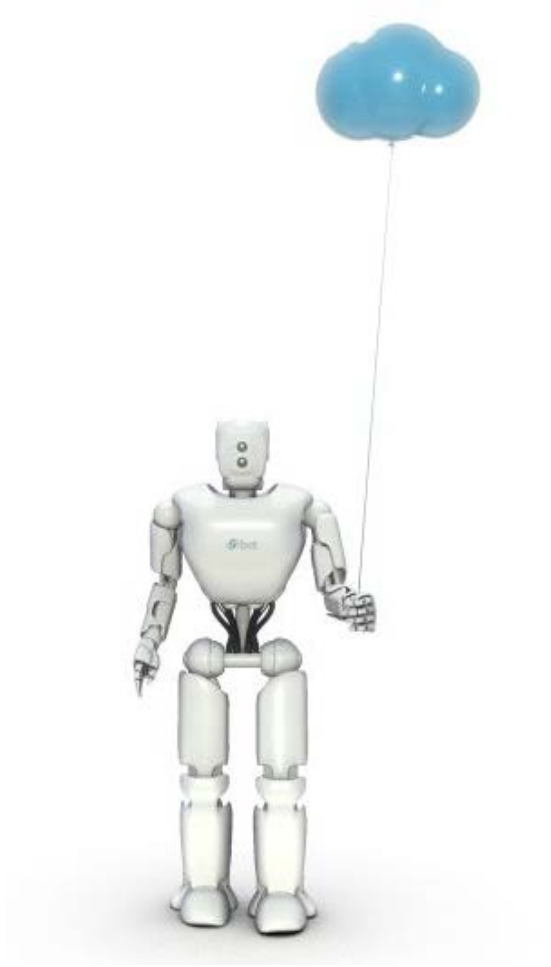


TECHNISCHE UNIVERSITÄT DRESDEN

Dynamics in Distributed Clouds

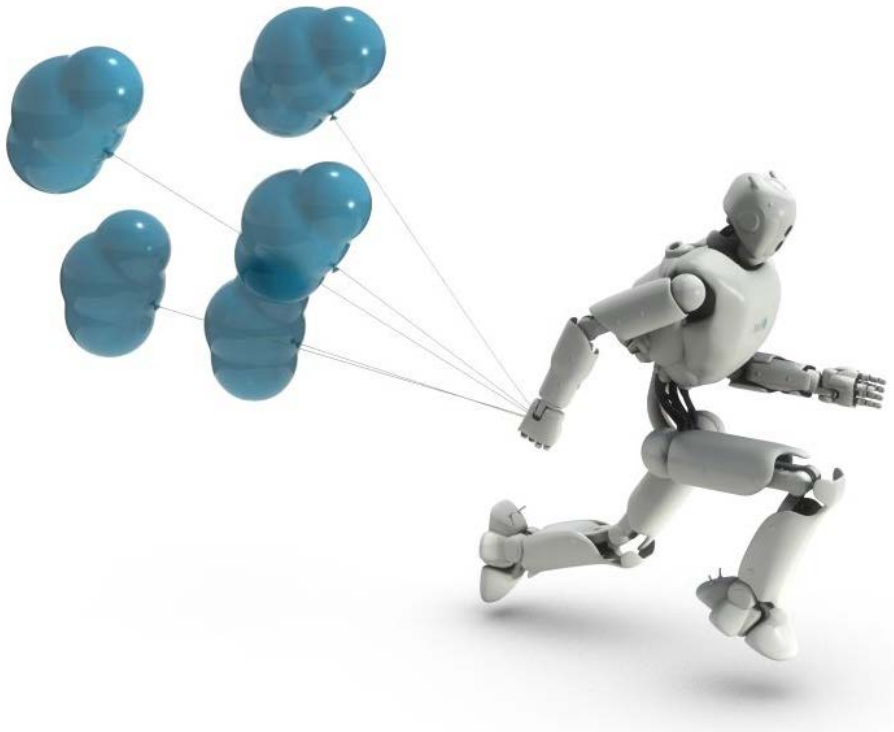


Cloud Evolution



Single/Static

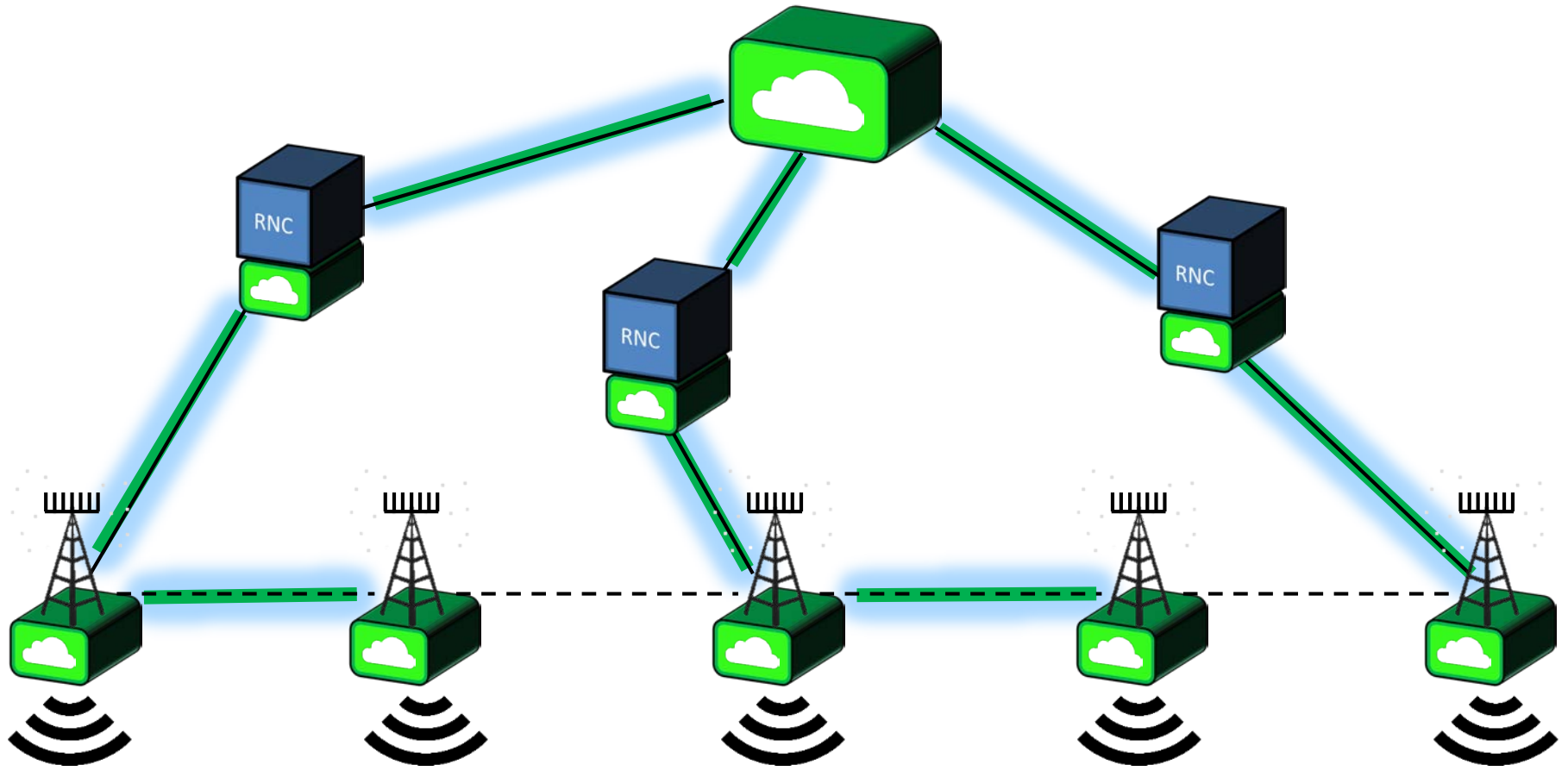
Distributed/Static



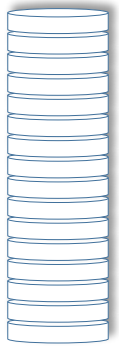
Distributed/Agile

TECHNISCHE UNIVERSITÄT DRESDEN

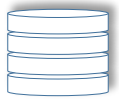
Mobile Edge Cloud / Micro Cloud / Cloud

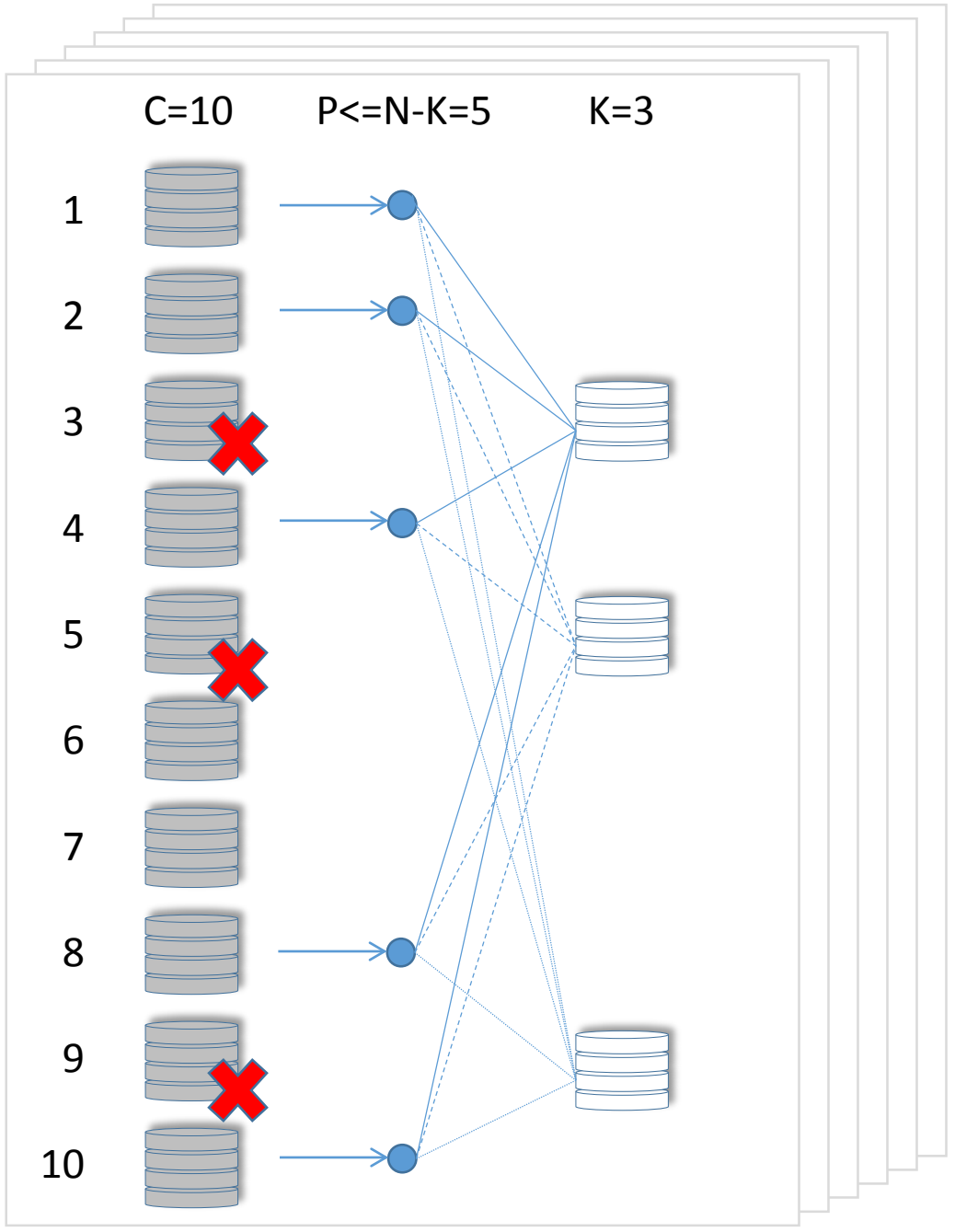


G=20



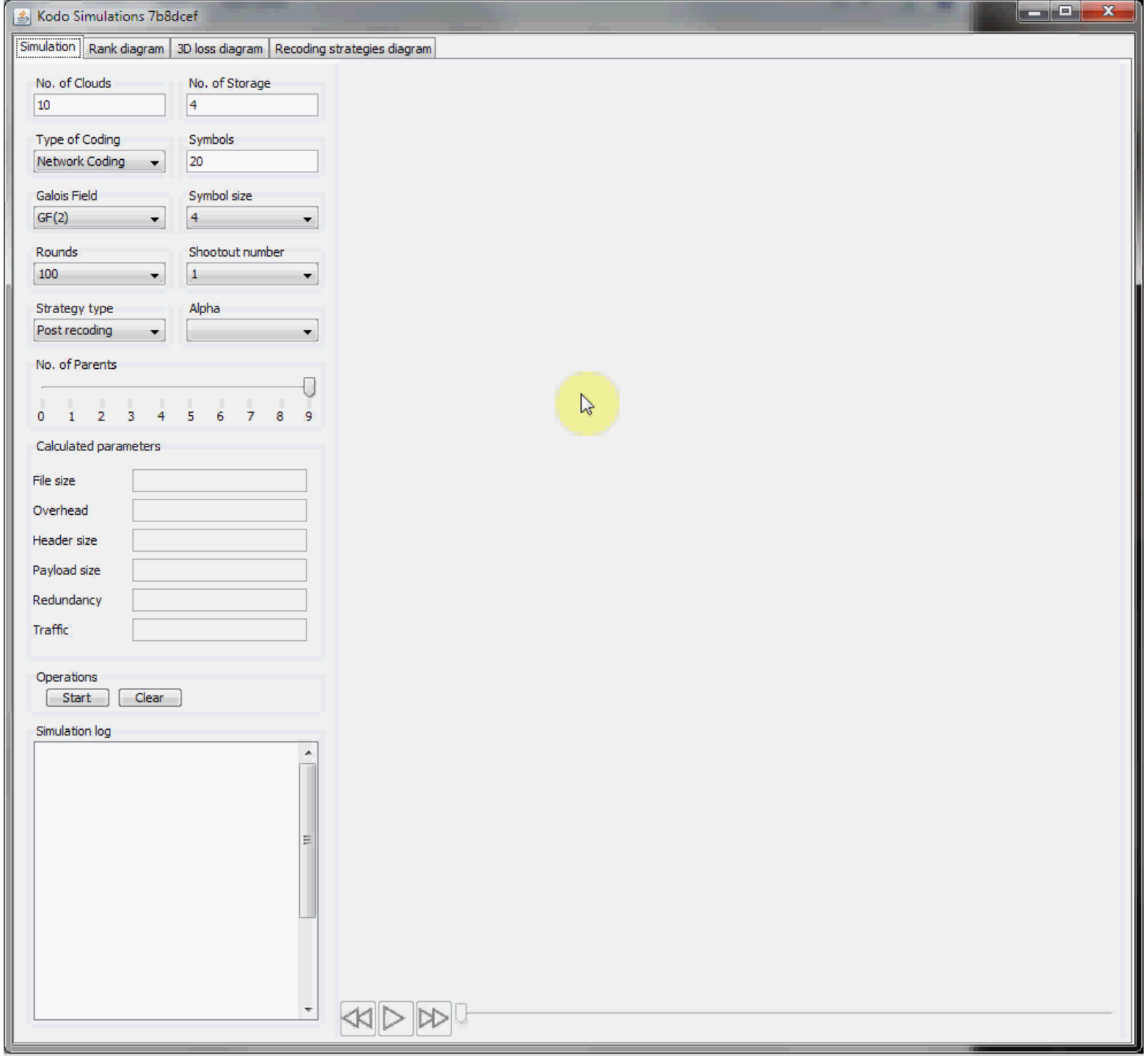
Q=4



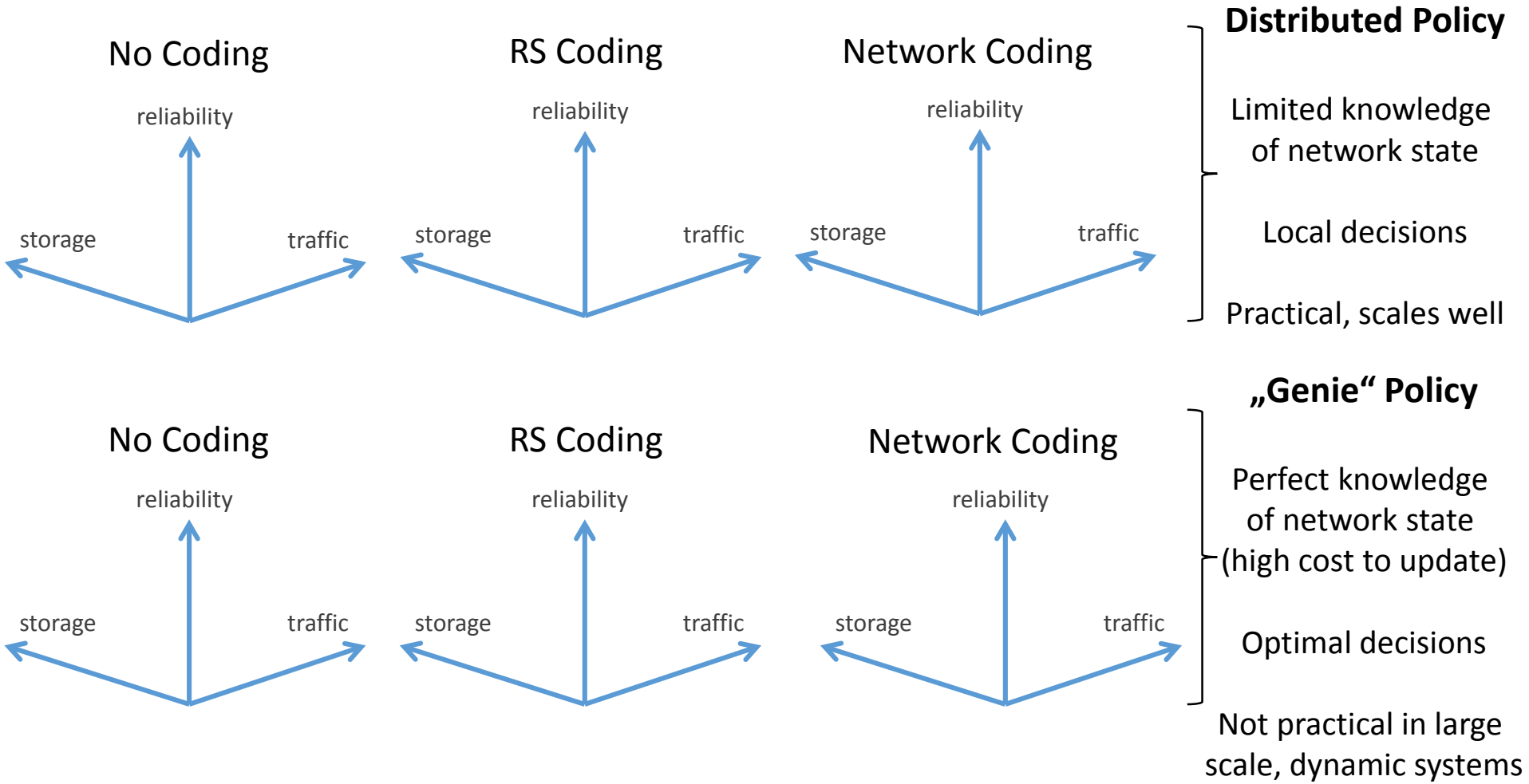


Rounds

TECHNISCHE UNIVERSITÄT DRESDEN



Data Survival over Time

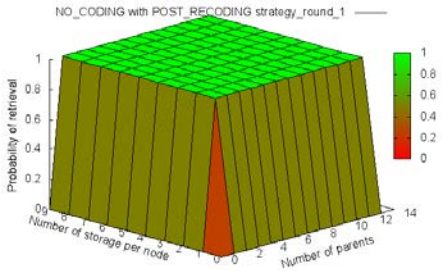


Current systems are somewhere in between these two policies

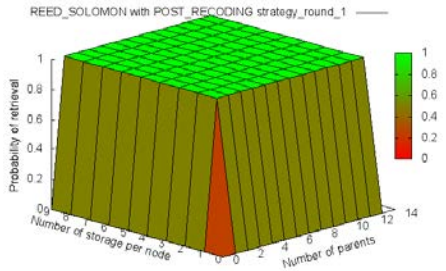
Data Survival over Time

Distributed Policy

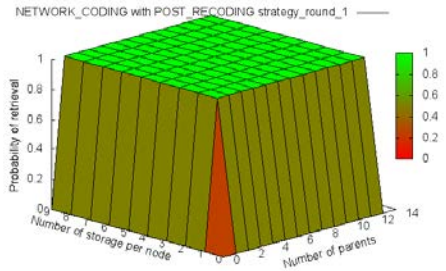
No Coding



RS Coding

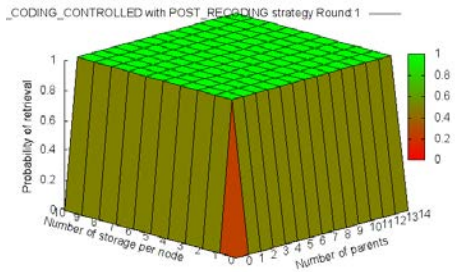


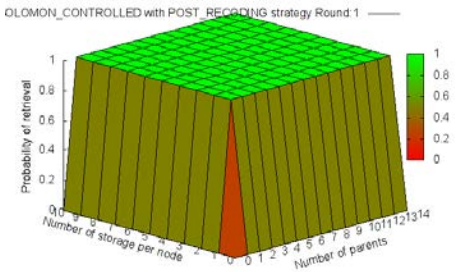
Network Coding

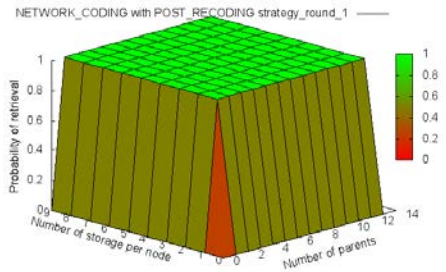


state-less

„Genie“ Policy







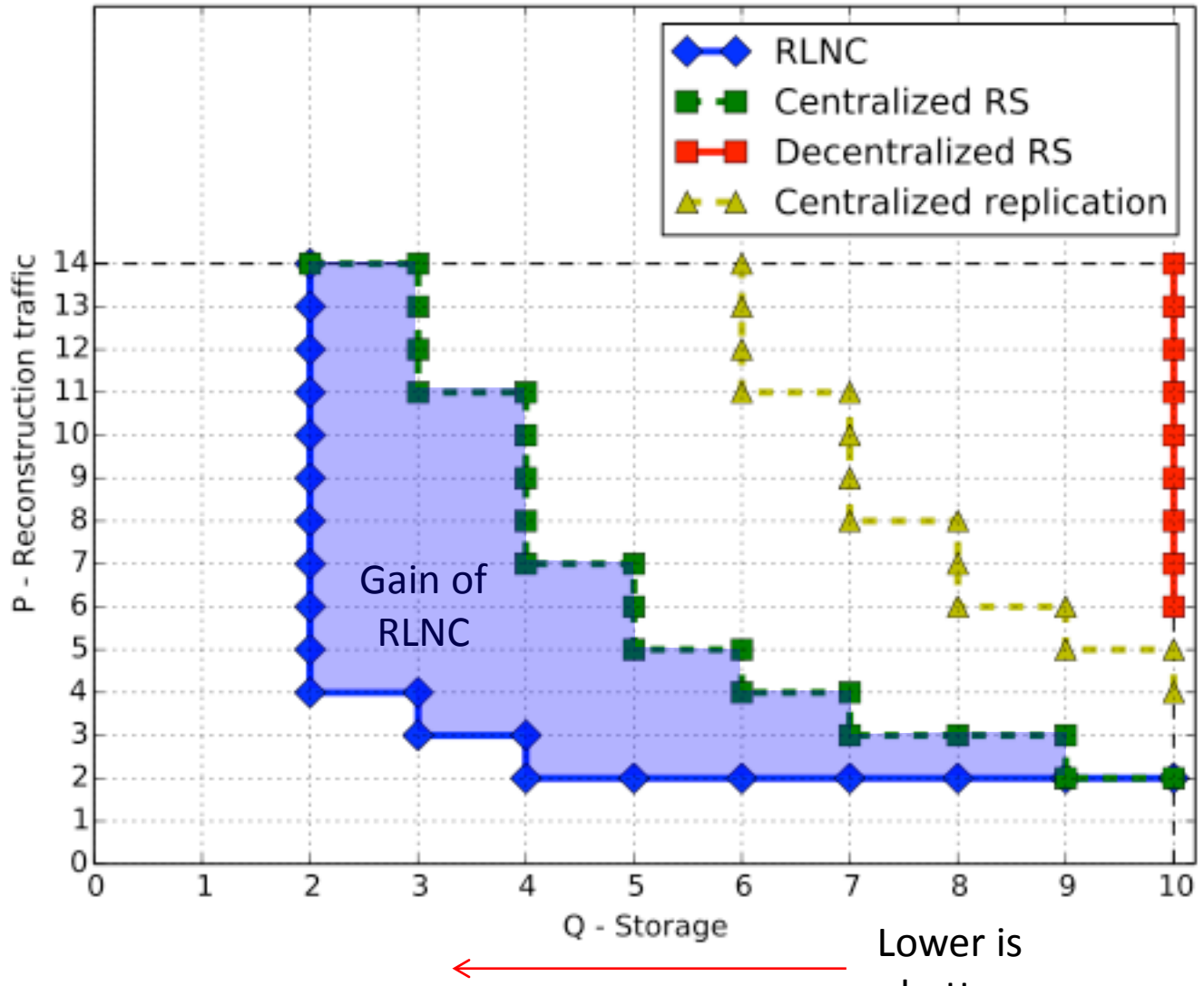
Over best alternative, NC buys you:

- Up to 65% reduction in storage
- Up to 71% reduction in network use

No costly methods to maintain a perfect knowledge of network state

Data Survival (large # of runs)

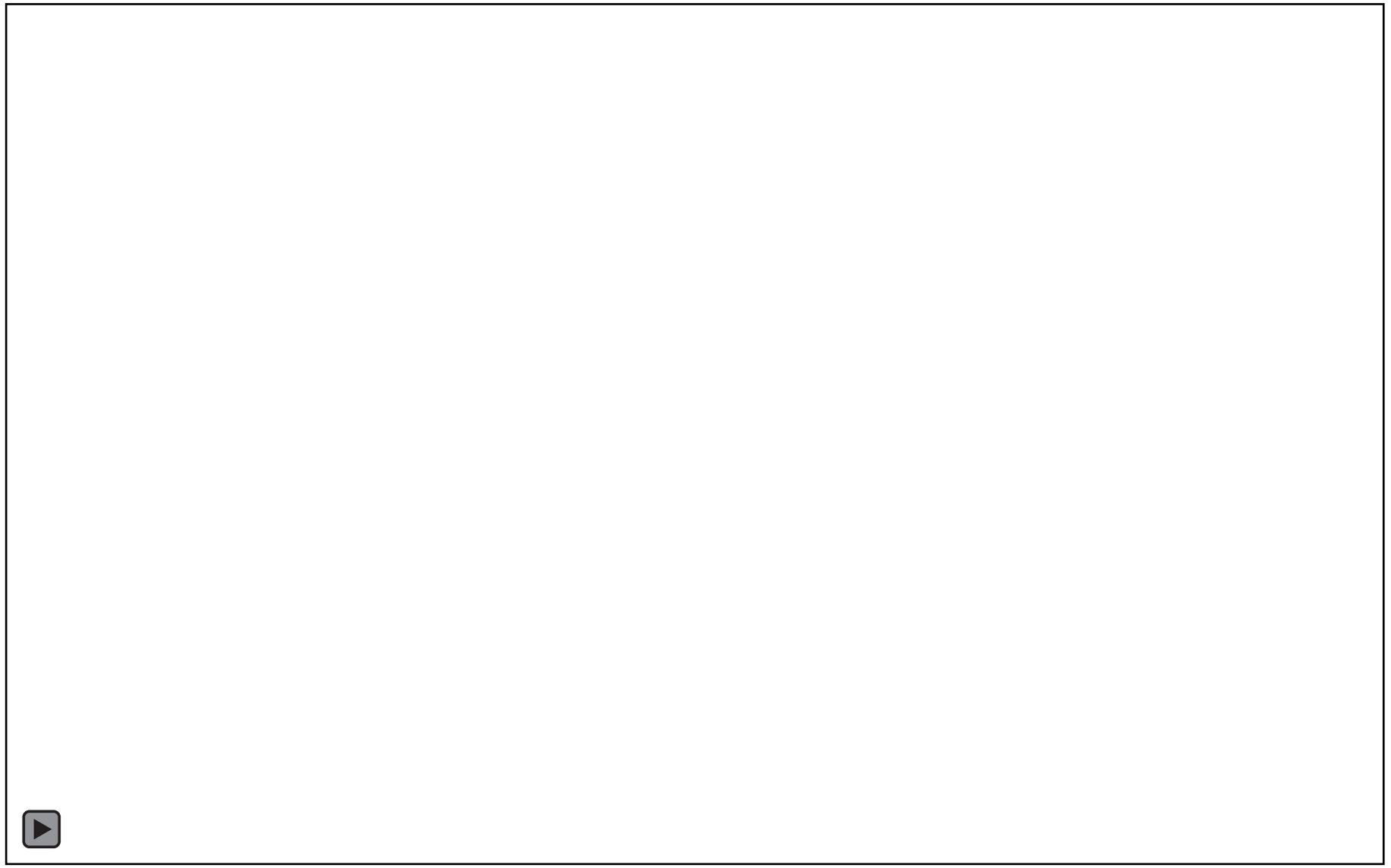
Lower is better



Lower is better

Top view of 3D plots

Ongoing Research

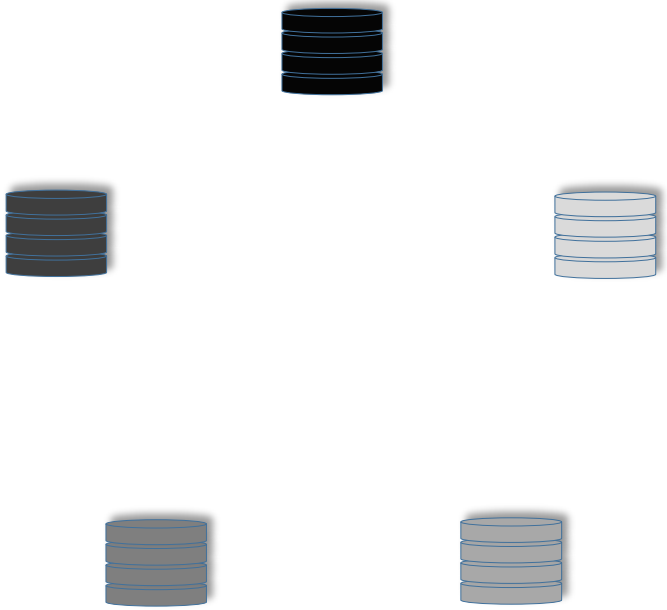


TECHNISCHE UNIVERSITÄT DRESDEN

INTER RACK / INTER CENTER

More on Regenerating Codes

File made up of 15 chunks

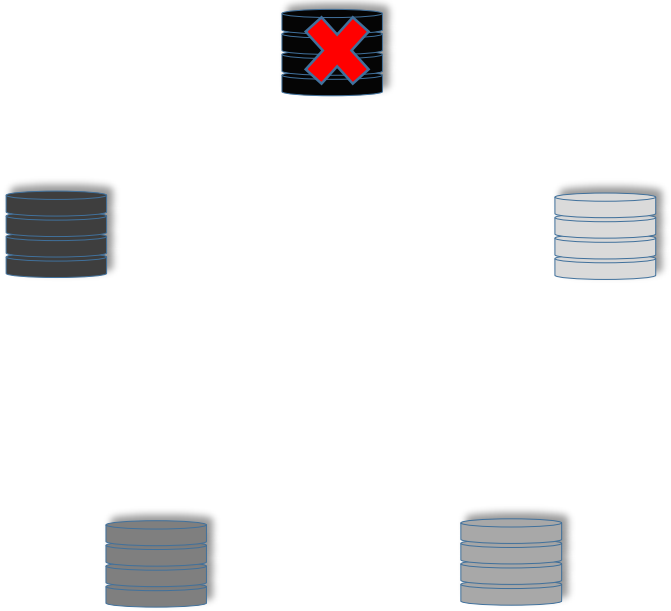


Stored in 5 racks,
4 chunks each
Redundancy 33%

TECHNISCHE UNIVERSITÄT DRESDEN

Example 1

File made up of 15 chunks

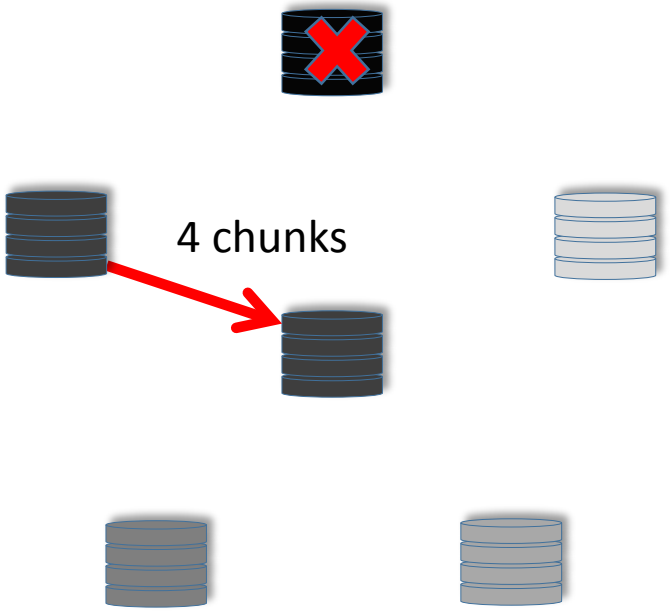


Stored in 5 racks,
4 chunks each
Redundancy 33%

Example 1: Reed-Solomon

File made up of 15 chunks

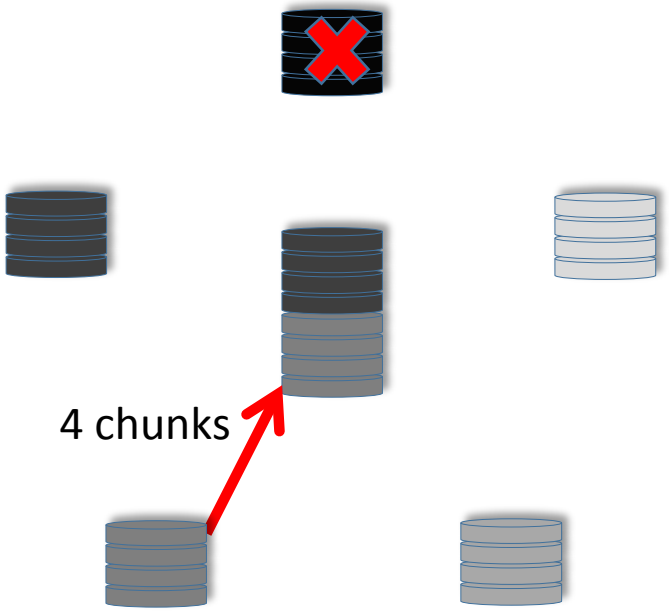
Stored in 5 racks,
4 chunks each
Redundancy 33%



Example 1: Reed-Solomon

File made up of 15 chunks

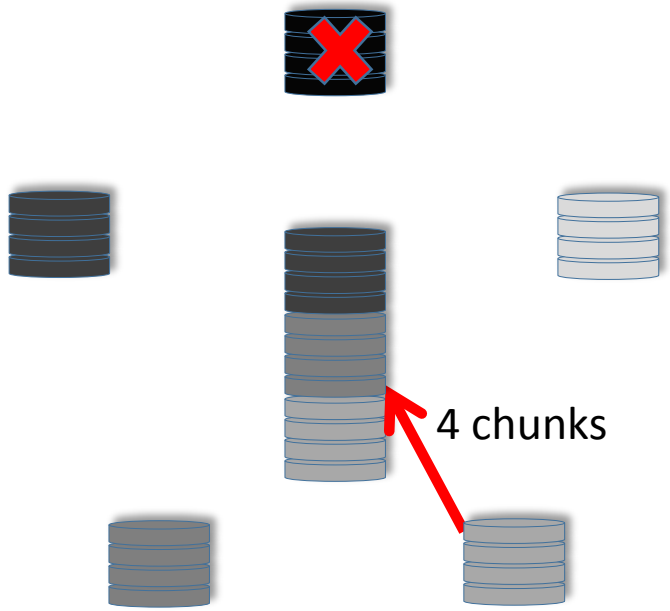
Stored in 5 racks,
4 chunks each
Redundancy 33%



Example 1: Reed-Solomon

File made up of 15 chunks

Stored in 5 racks,
4 chunks each
Redundancy 33%

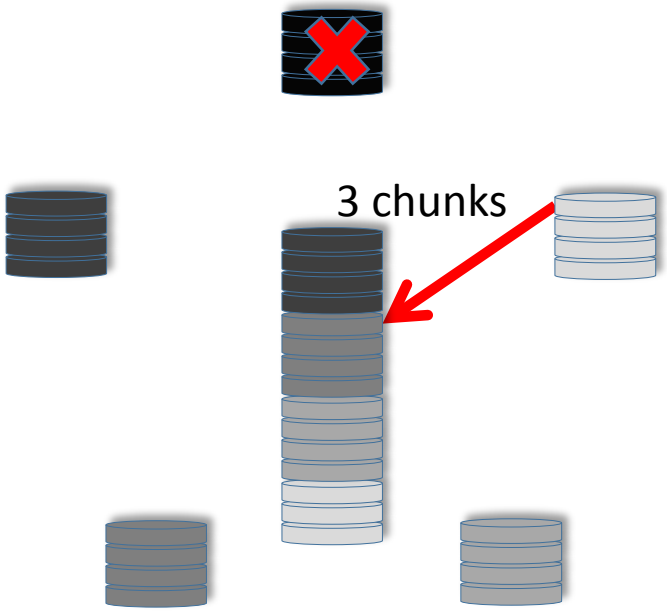


TECHNISCHE UNIVERSITÄT DRESDEN

Example 1: Reed-Solomon

File made up of 15 chunks

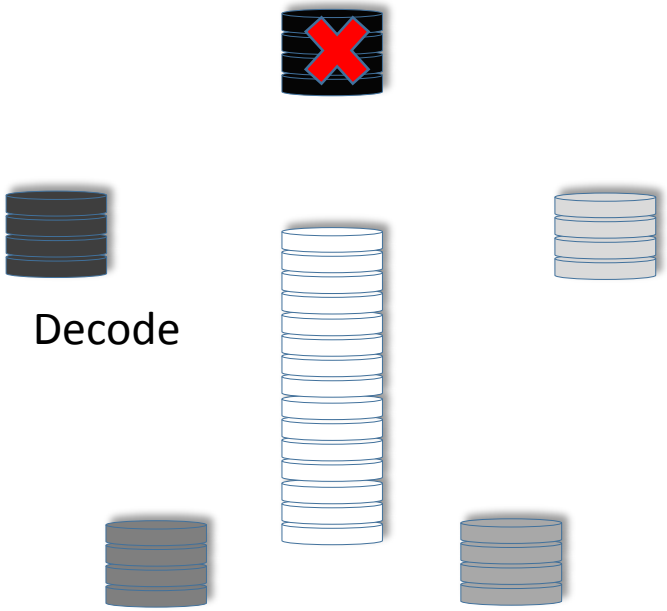
Stored in 5 racks,
4 chunks each
Redundancy 33%



Example 1: Reed-Solomon

File made up of 15 chunks

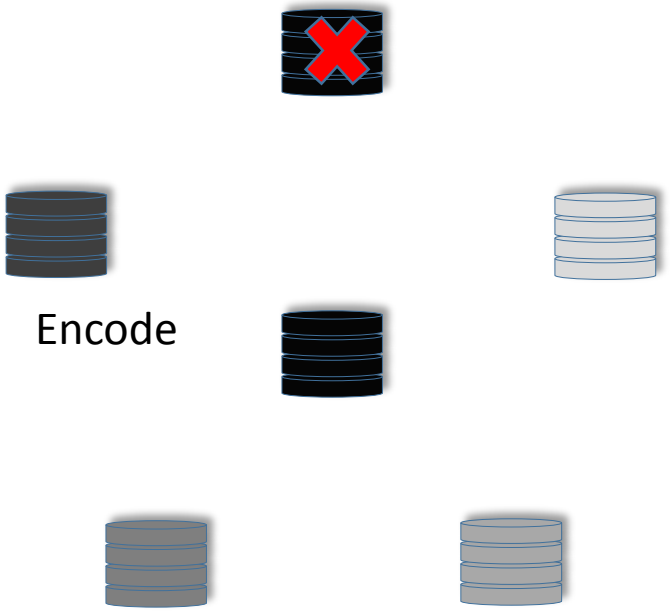
Stored in 5 racks,
4 chunks each
Redundancy 33%



Example 1: Reed-Solomon

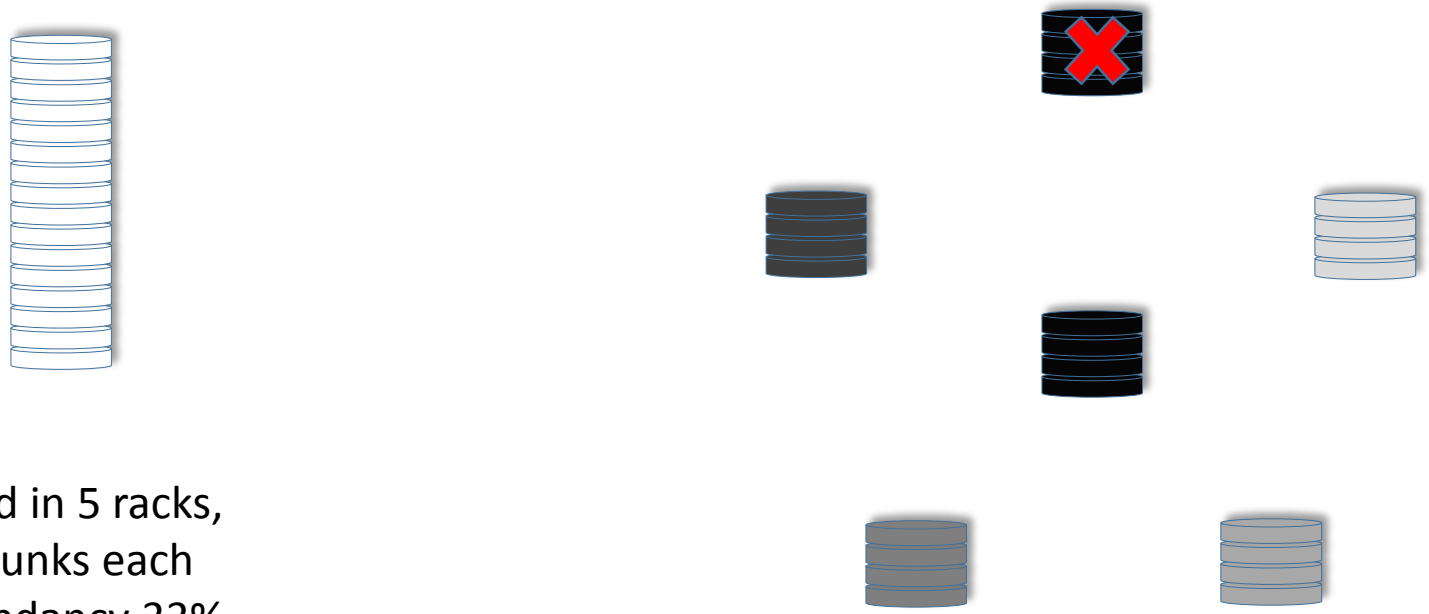
File made up of 15 chunks

Stored in 5 racks,
4 chunks each
Redundancy 33%



Example 1: Reed-Solomon

File made up of 15 chunks



Stored in 5 racks,
4 chunks each
Redundancy 33%

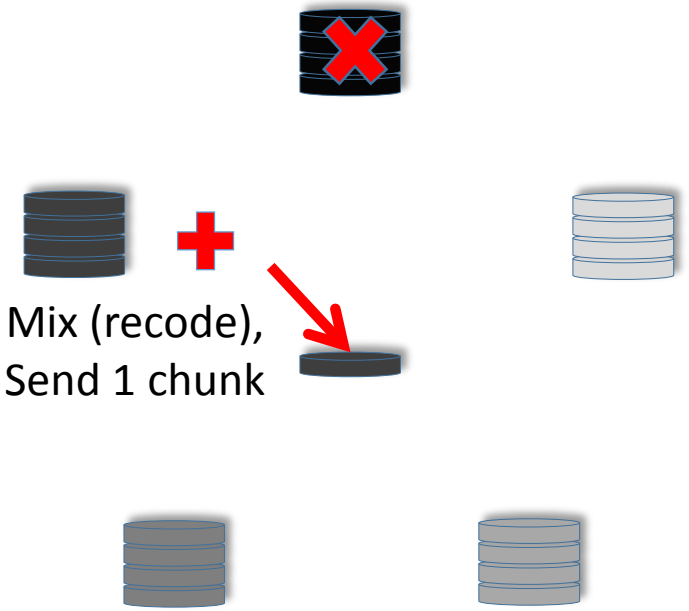
	I/O	Network: Intra-Rack	Inter-Rack	Processing
RS:	15	0*	15	Decode + Encode 15x15 matrix (new rack)
RLNC:				

* May require some intra-rack transfer depending on structure

Example 1: RLNC

File made up of 15 chunks

Stored in 5 racks,
4 chunks each
Redundancy 33%

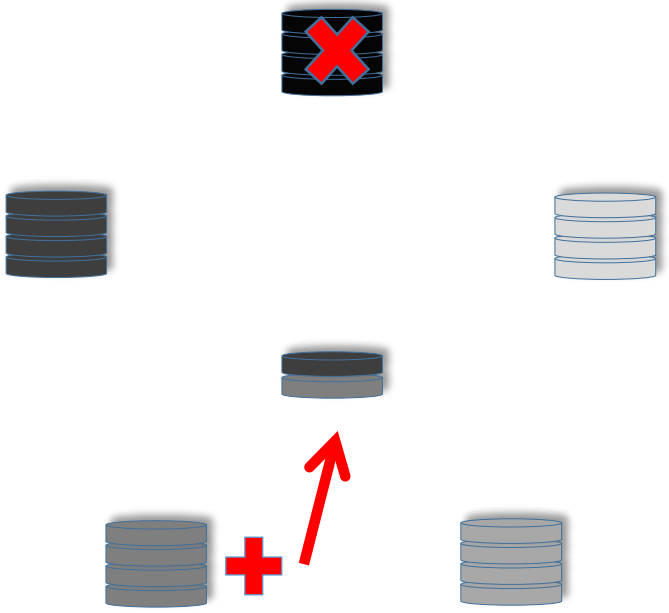


TECHNISCHE UNIVERSITÄT DRESDEN

Example 1: RLNC

File made up of 15 chunks

Stored in 5 racks,
4 chunks each
Redundancy 33%



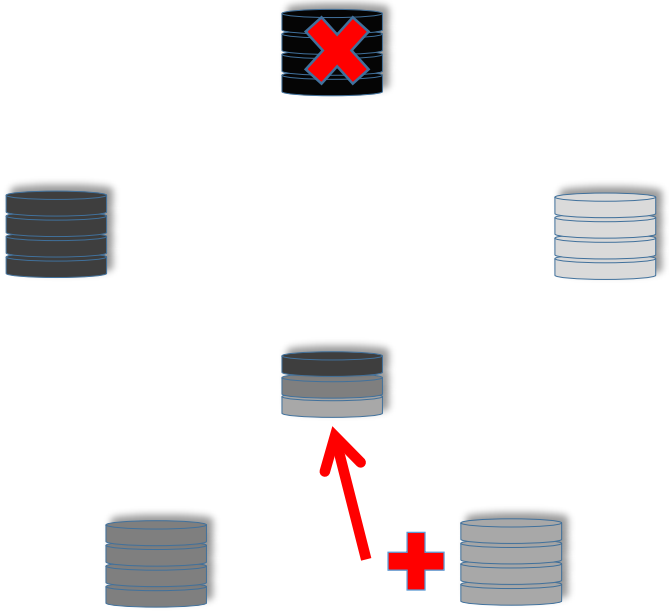
Mix (recode),
Send 1 chunk

TECHNISCHE UNIVERSITÄT DRESDEN

Example 1: RLNC

File made up of 15 chunks

Stored in 5 racks,
4 chunks each
Redundancy 33%

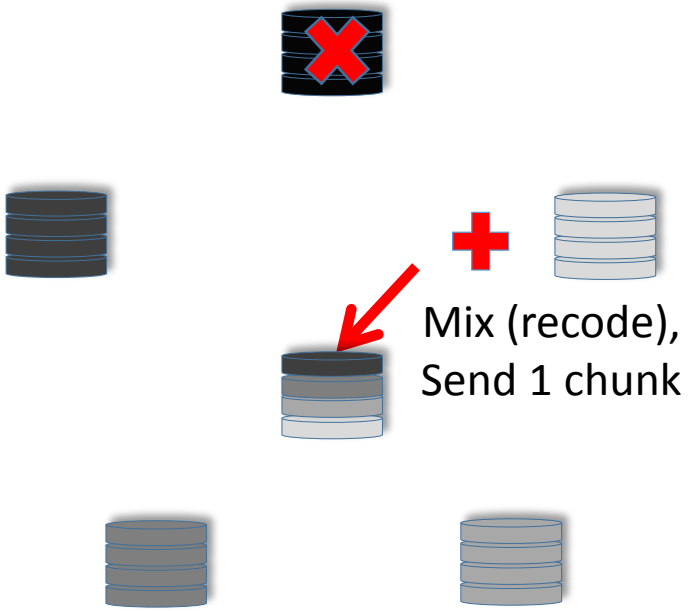


Mix (recode),
Send 1 chunk

Example 1: RLNC

File made up of 15 chunks

Stored in 5 racks,
4 chunks each
Redundancy 33%

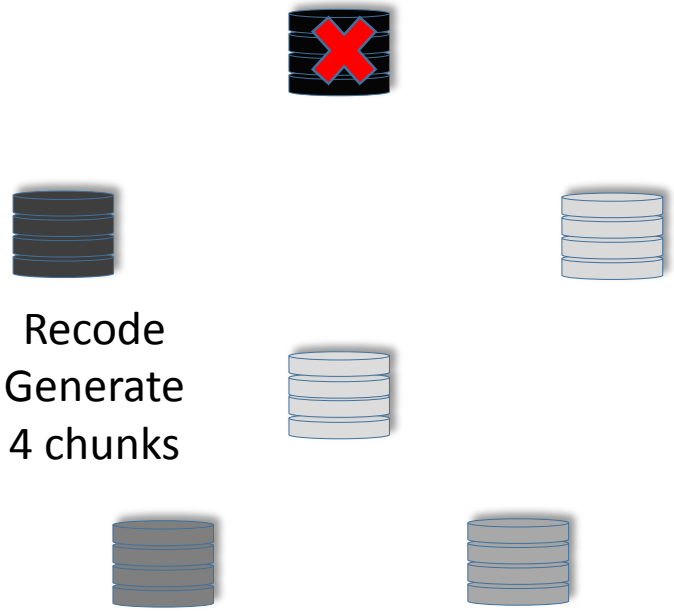


TECHNISCHE UNIVERSITÄT DRESDEN

Example 1: RLNC

File made up of 15 chunks

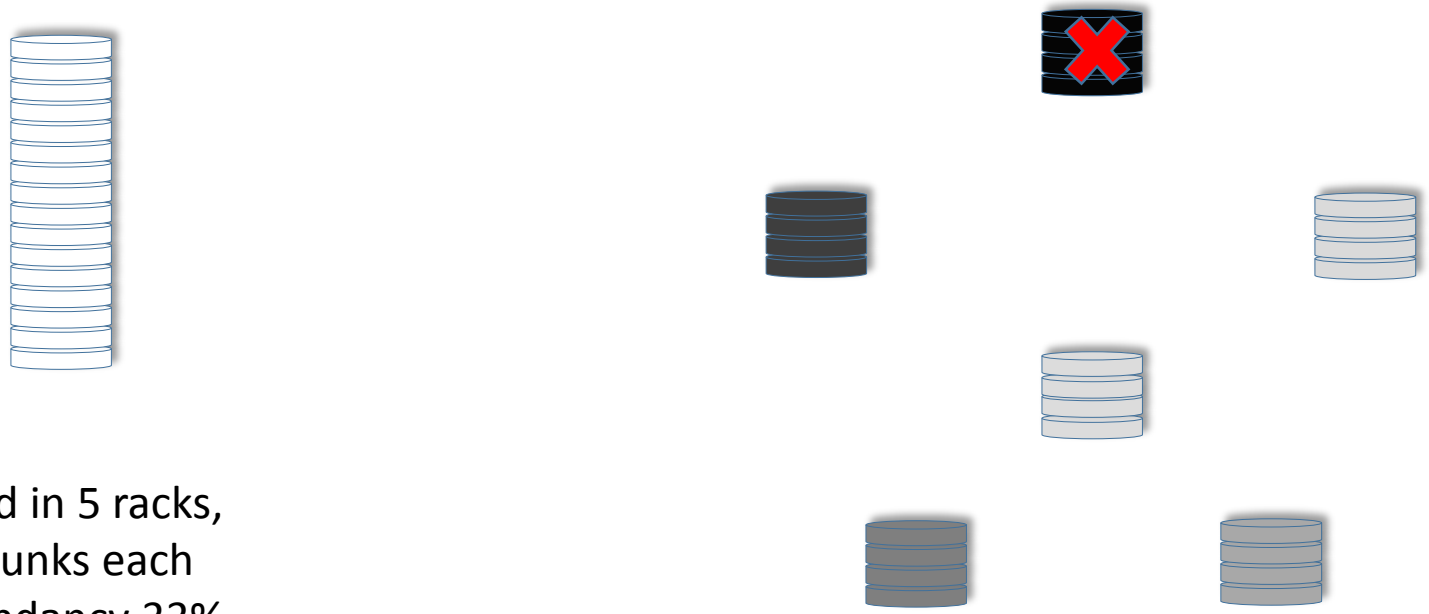
Stored in 5 racks,
4 chunks each
Redundancy 33%



Recode
Generate
4 chunks

Example 1: RLNC

File made up of 15 chunks



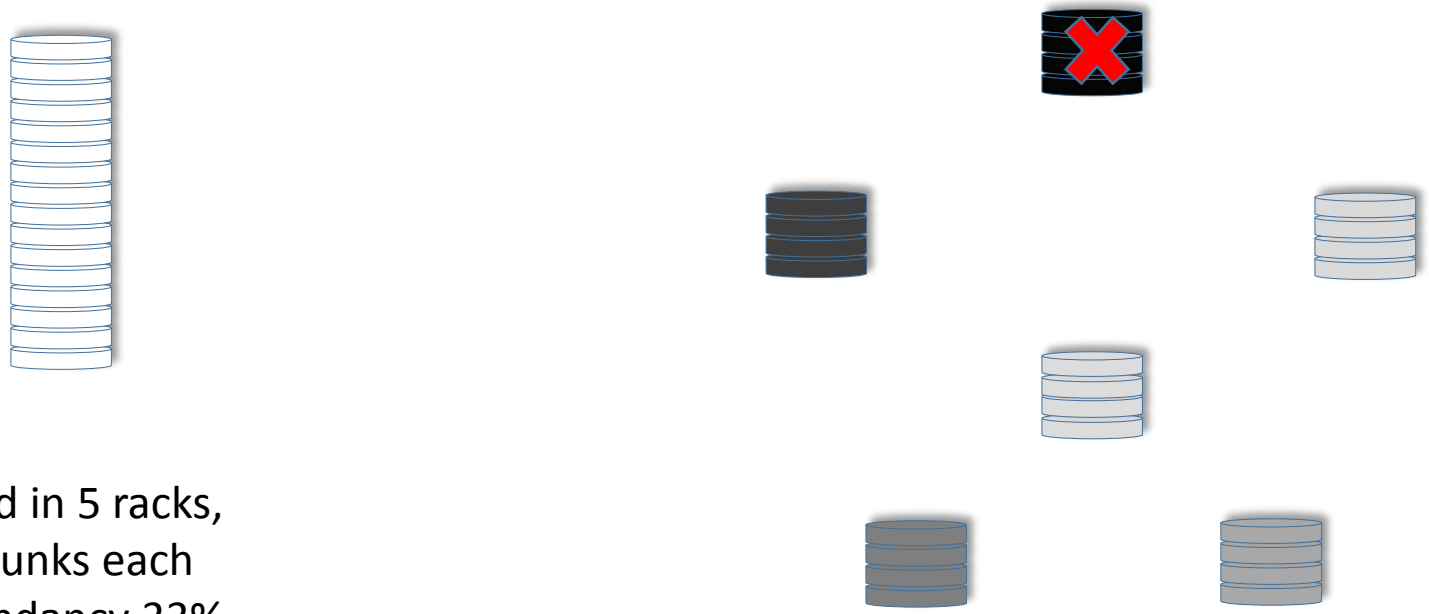
Stored in 5 racks,
4 chunks each
Redundancy 33%

	I/O	Network: Intra-Rack	Inter-Rack	Processing
RS:	15	0*	15	Decode + Encode 15x15 matrix (new rack)
RLNC:	15	11	4	Encode 4x4 matrices (4 times), and one 3x3 matrix

* May require some intra-rack transfer depending on structure

Example 1: RLNC

File made up of 15 chunks



Stored in 5 racks,
4 chunks each
Redundancy 33%

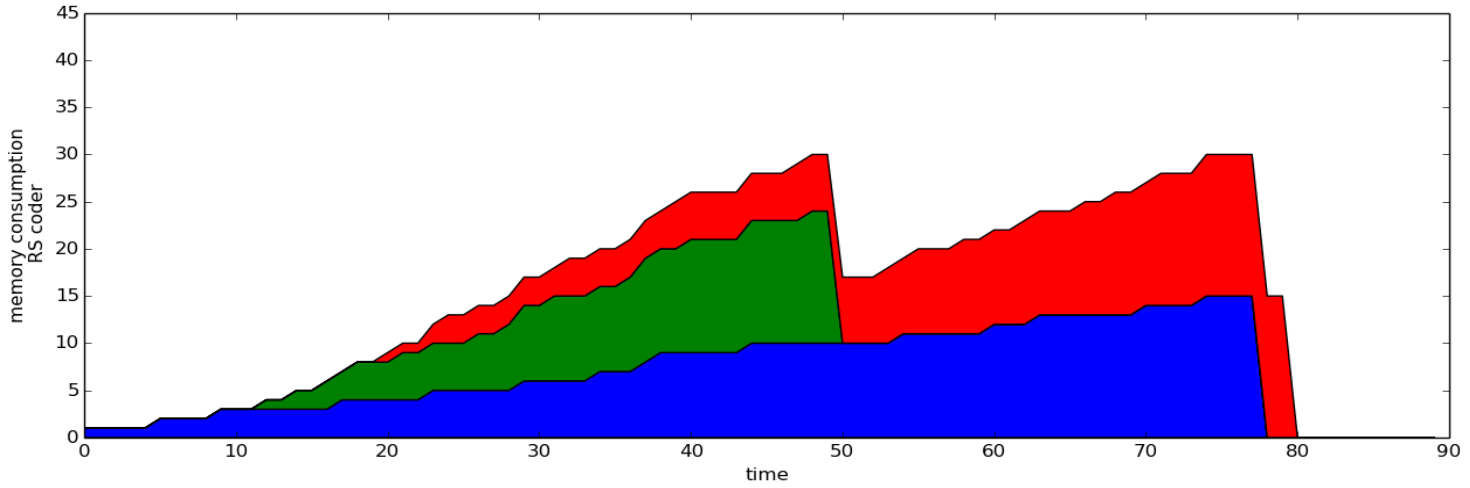
	I/O	Network: Intra-Rack	Inter-Rack	Processing
RS:	15	0*	15	Centralized in new rack
RLNC:	15	11	4	Distributed in old and new racks

* May require some intra-rack transfer depending on structure

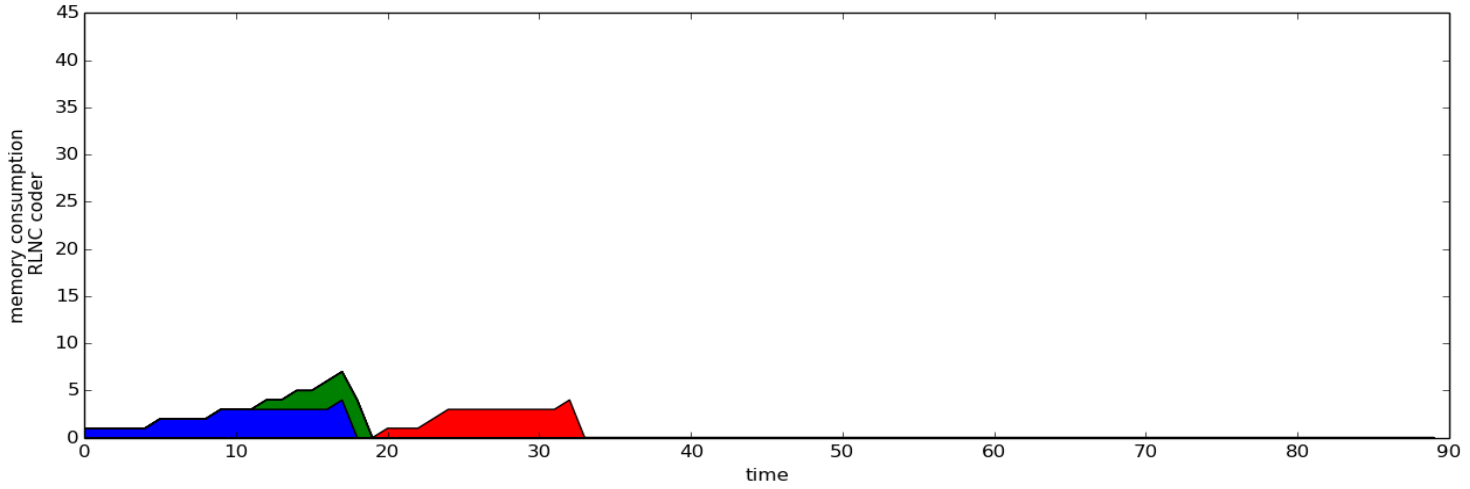
TECHNISCHE UNIVERSITÄT DRESDEN

Memory Consumption RS vs RLNC

Reed-Solomon



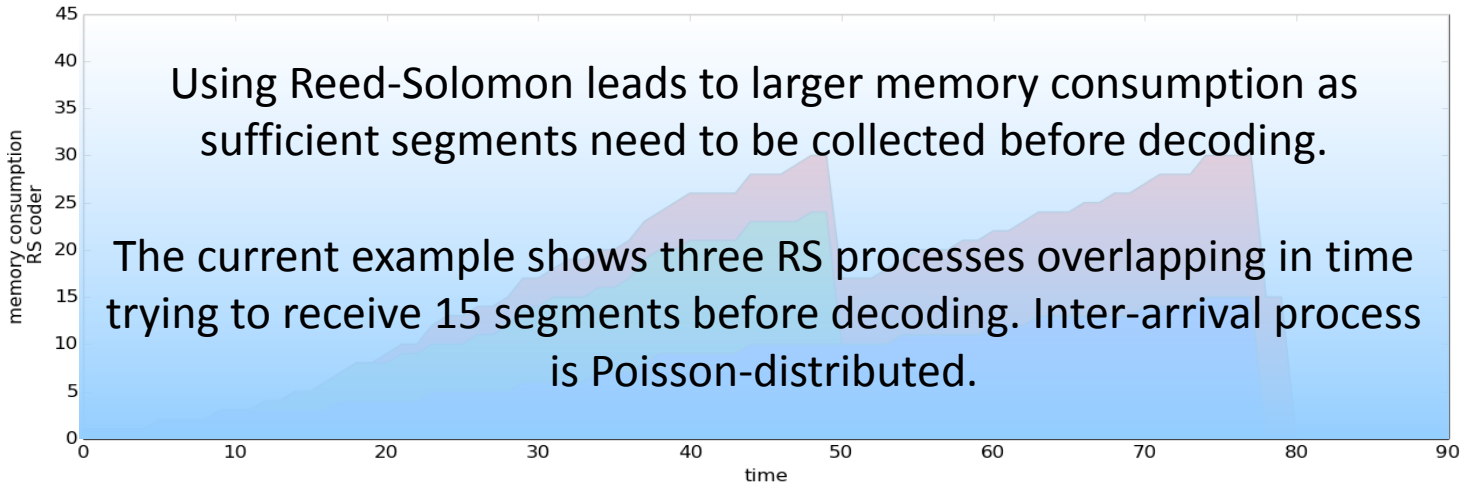
RLNC



TECHNISCHE UNIVERSITÄT DRESDEN

Memory Consumption RS vs RLNC

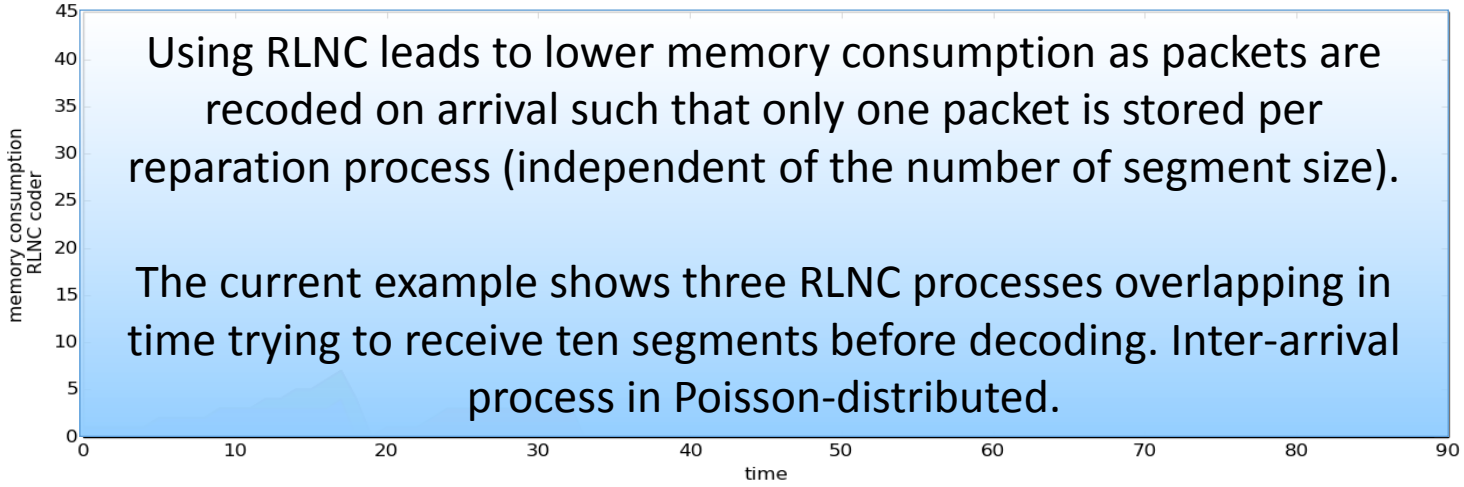
Reed-Solomon



Using Reed-Solomon leads to larger memory consumption as sufficient segments need to be collected before decoding.

The current example shows three RS processes overlapping in time trying to receive 15 segments before decoding. Inter-arrival process is Poisson-distributed.

RLNC



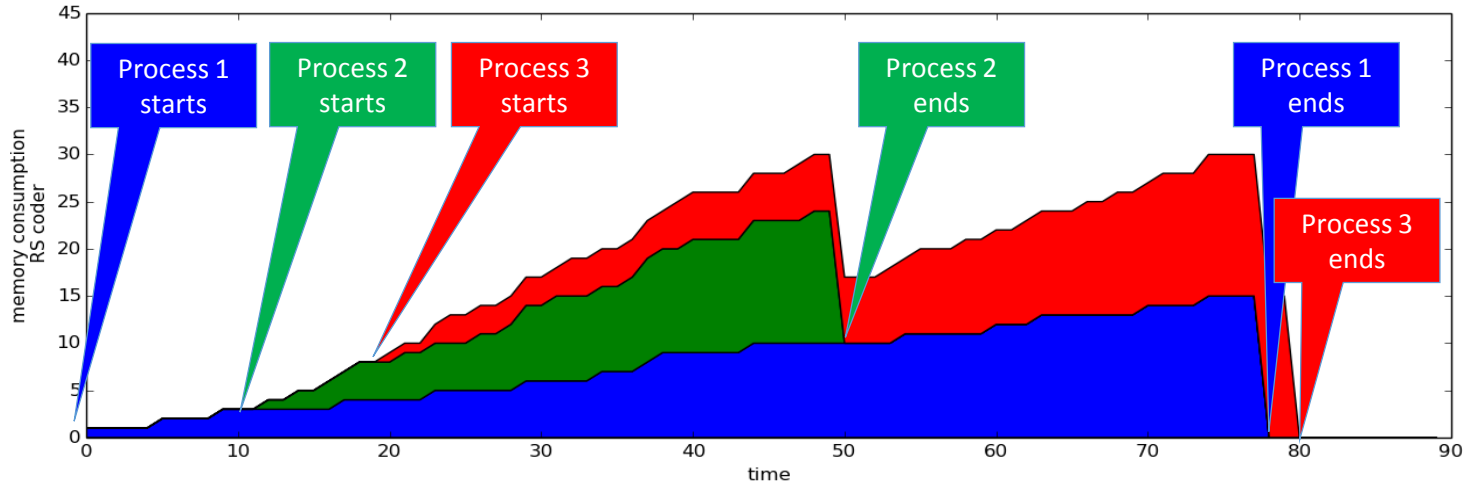
Using RLNC leads to lower memory consumption as packets are recoded on arrival such that only one packet is stored per reparation process (independent of the number of segment size).

The current example shows three RLNC processes overlapping in time trying to receive ten segments before decoding. Inter-arrival process in Poisson-distributed.

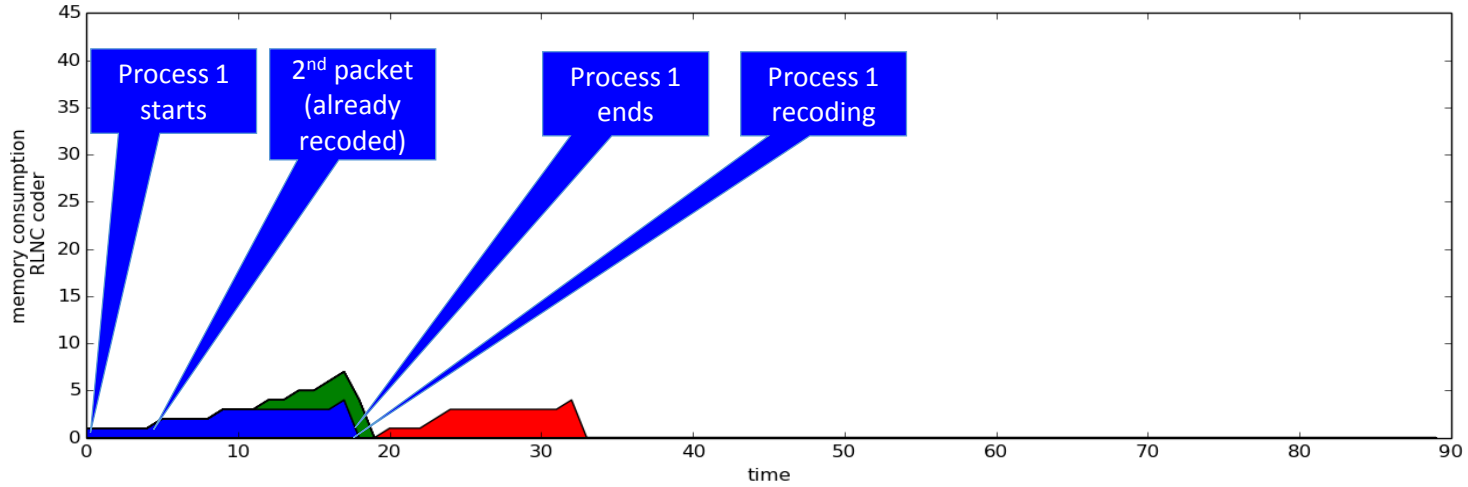
TECHNISCHE UNIVERSITÄT DRESDEN

Memory Consumption RS vs RLNC

Reed-Solomon



RLNC

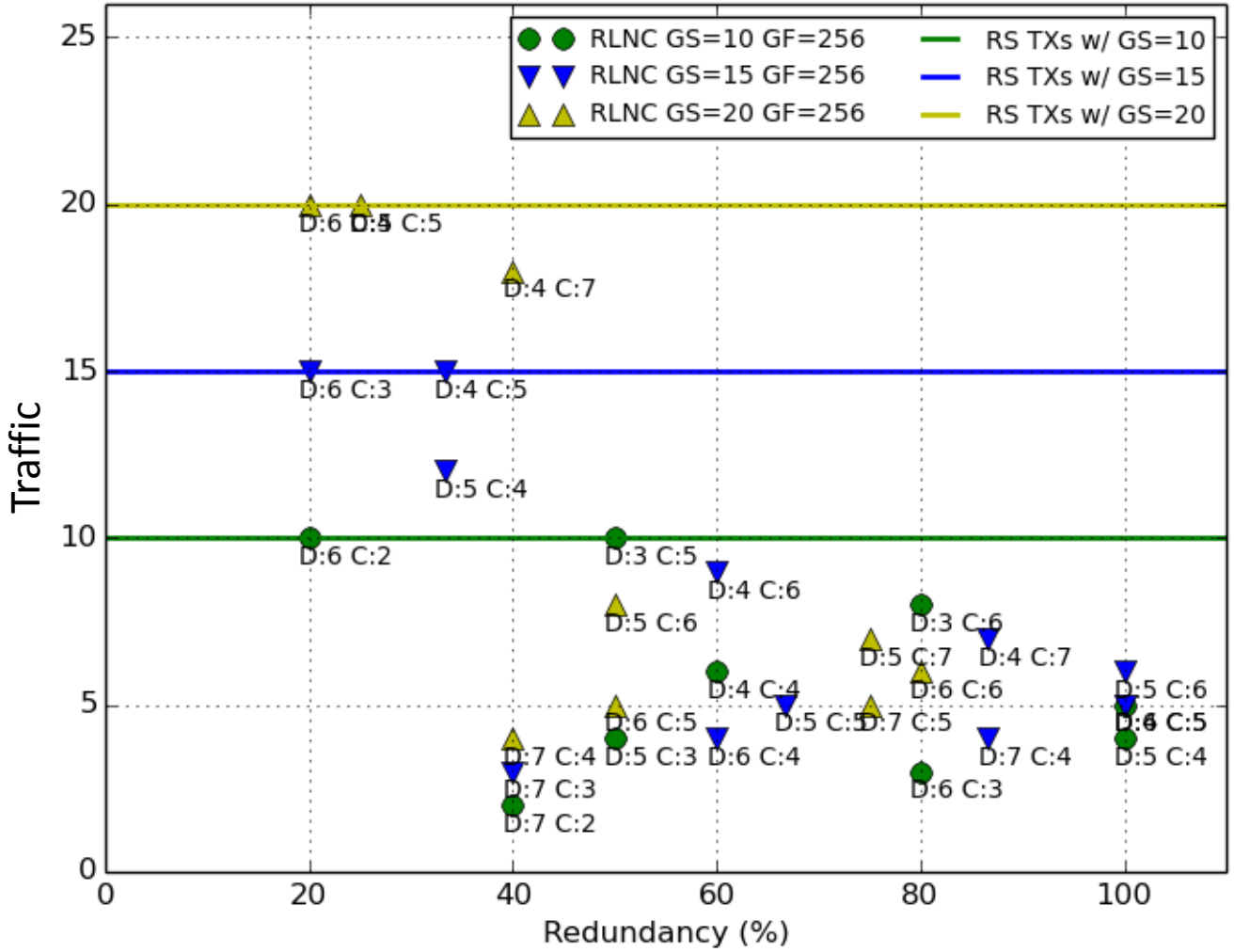


TECHNISCHE UNIVERSITÄT DRESDEN

Memory Consumption RS vs RLNC

- Advantages for recoding on the fly
  - Less memory consumption during recovery process
  - Workload can be distributed during recovery process (across racks), while RS will have large workload in the end
  - Less delay after last packet arrives for RLNC as only one packet need to be coded, while RS need to code over multiple packets.

Gains Beyond the Example

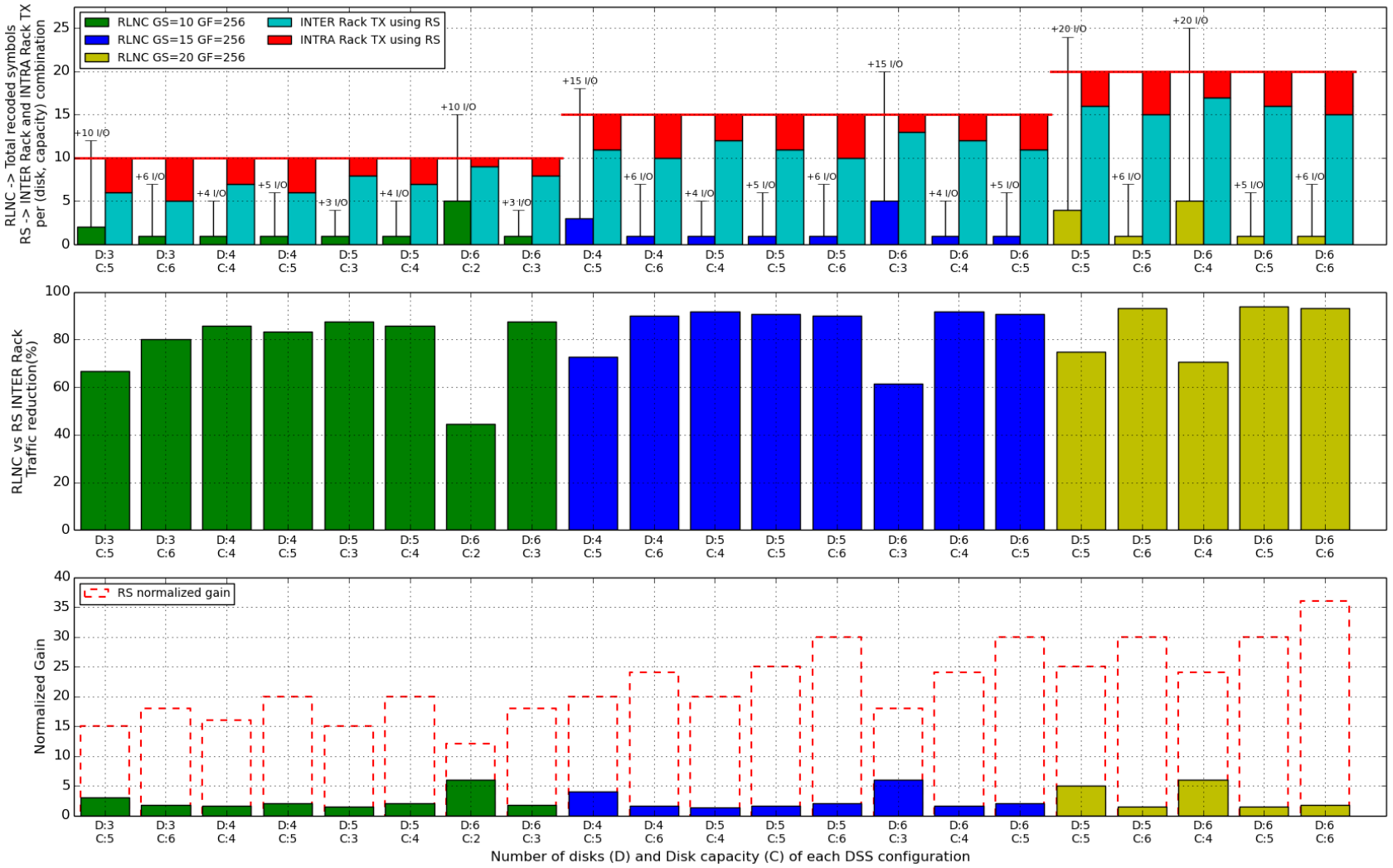


D: # of racks

C: # fragments of the file

TECHNISCHE UNIVERSITÄT DRESDEN

Gains Beyond the Example



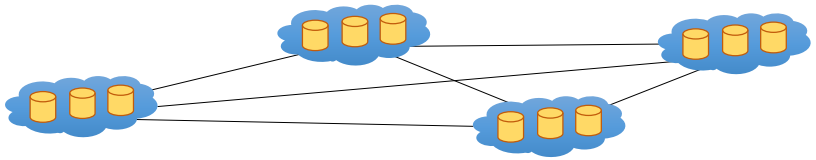
TECHNISCHE UNIVERSITÄT DRESDEN

Another Example showing advantage of RLNC over RS/XORBAS

Example 2

8 segments (plus redundancy) in 4 clouds

Example: 4 clouds with 3 disks (12 disk storage).



Coding Scheme	Disk Storage (less is better)	Inter (Intra) Cloud Bandwidth (less is better)	
		Cloud failure	Disk failure
RS 8:4	12	8	6
XORBAS 8:4:2	16	8	0(1)
RLNC v1a 8:4 systematic	12	6	2
RLNC v1b 8:8 systematic	16	4	1
RLNC v2 dense	12	3	1

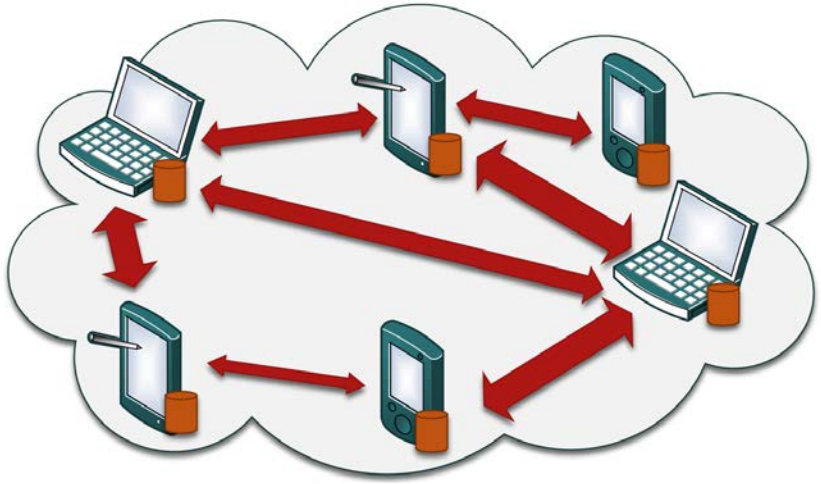
- Conclusion: RLNC approaches will reduce the traffic at comparable storage situations.
- Staircase/LDPC need significant storage - **unable even to reach 16 in storage**

Lessons learned

- Network Coding is the only solution for dynamic storage places …
- So how far can we go with this technology?

The mobile storage cloud

- Loosely coupled mobile nodes – P2P network
- Nodes are free to join and leave – highly dynamic
- Fully distributed



- One goal: keep the data alive

Scenarios

- Fans at a football game
  - > Statistics & info on the teams
  - > Shared media: photos & video

- Participants in traffic
  - > Traffic & road conditions
  - > Accident warning

Some assumptions so far

- The number of concurrent leaving nodes is constant and is in balance with joining nodes.
- The system has enough bandwidth to fill joining nodes.
- Motivation for this paper:
- How safe are these assumptions in reality?
- Do we need any additional constraints for a real system to function?

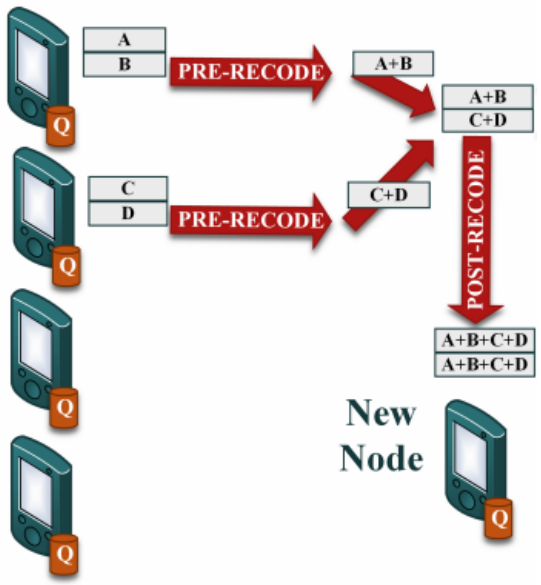
Dr. Torrent

- Android BitTorrent client
- > 50 000 downloads
- Community-based mobile P2P storage

- Collects anonymous usage information
- Data used with permission from the users and the app's creators

Self-sustainability

- For a given period of time, there should be enough bandwidth supplied by surviving nodes to fill new nodes.



- This gives an upper bound for the amount of data that can be stored in each node Tmax, Δt .

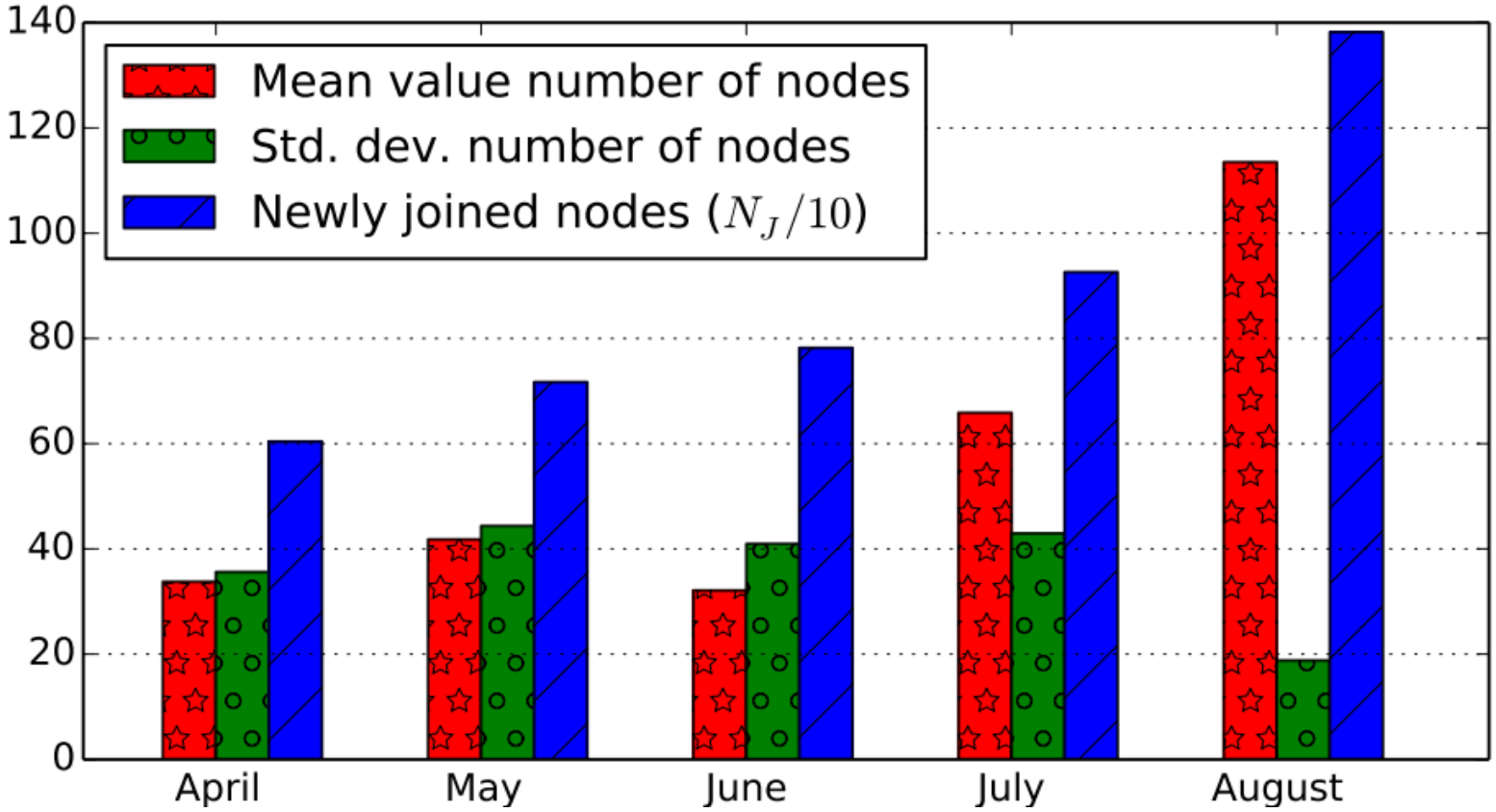
Node variance

- Redundancy in storage provides the slack to even out temporary imbalances between the number of leaving and joining nodes

- How much redundancy is needed?
  - > Enough to survive the worst-case scenario.

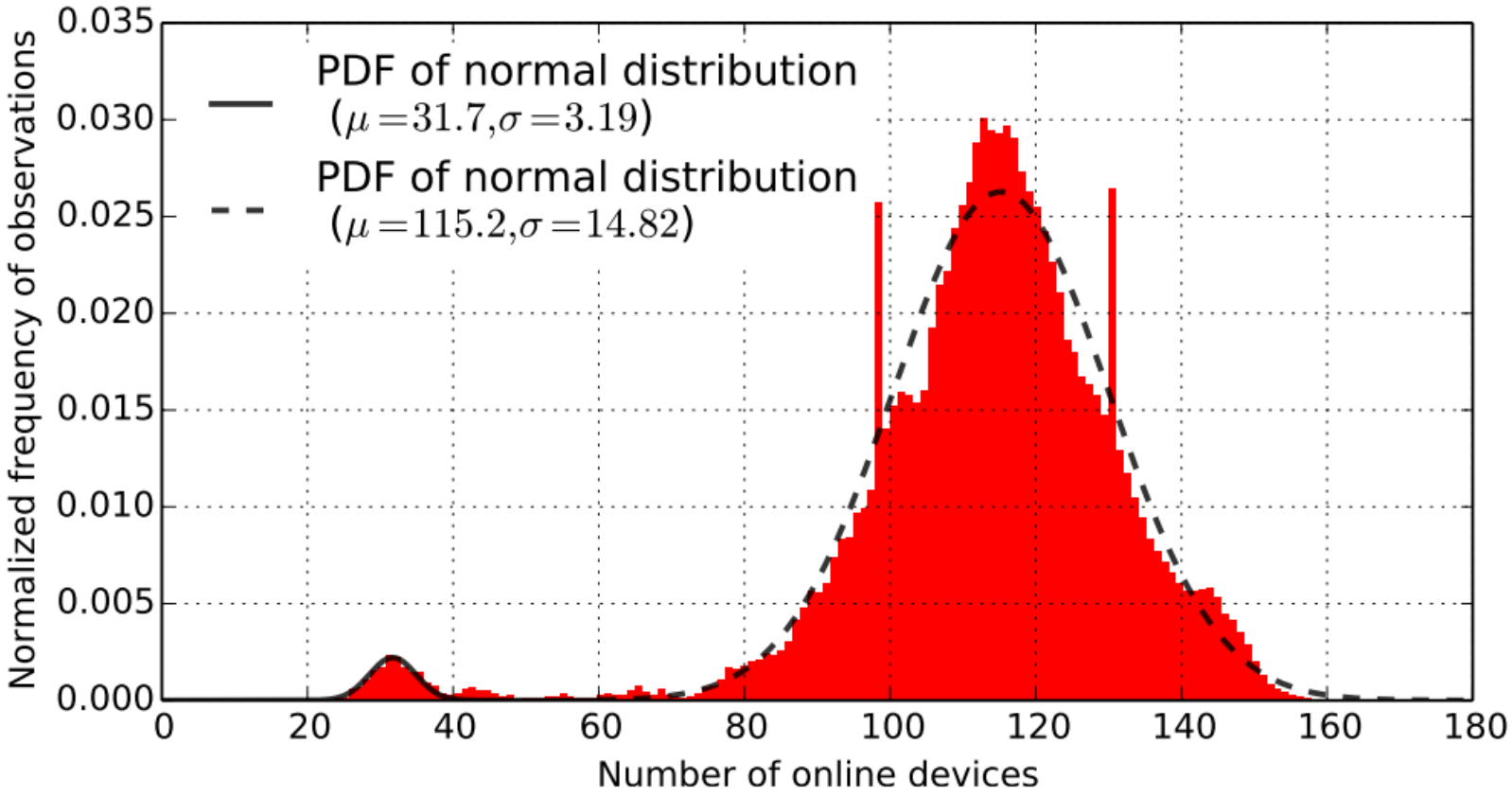
Long-term trends

- As the network grew, variance in the number of nodes decreased



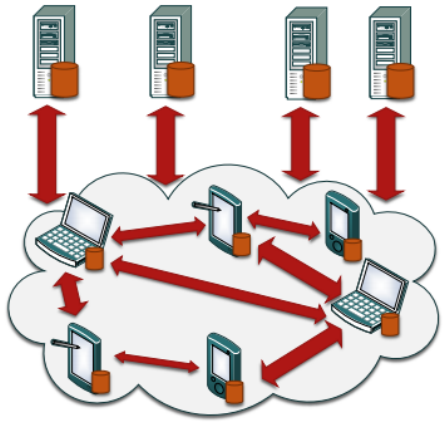
Concurrent users

- Unfortunately, there are occasional big drops in the number of online devices at a given time

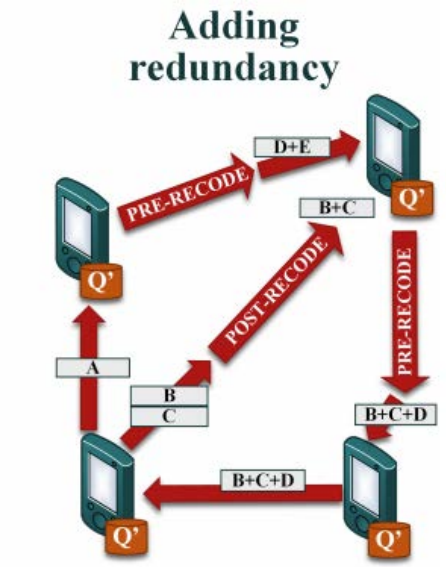


Increasing data survival

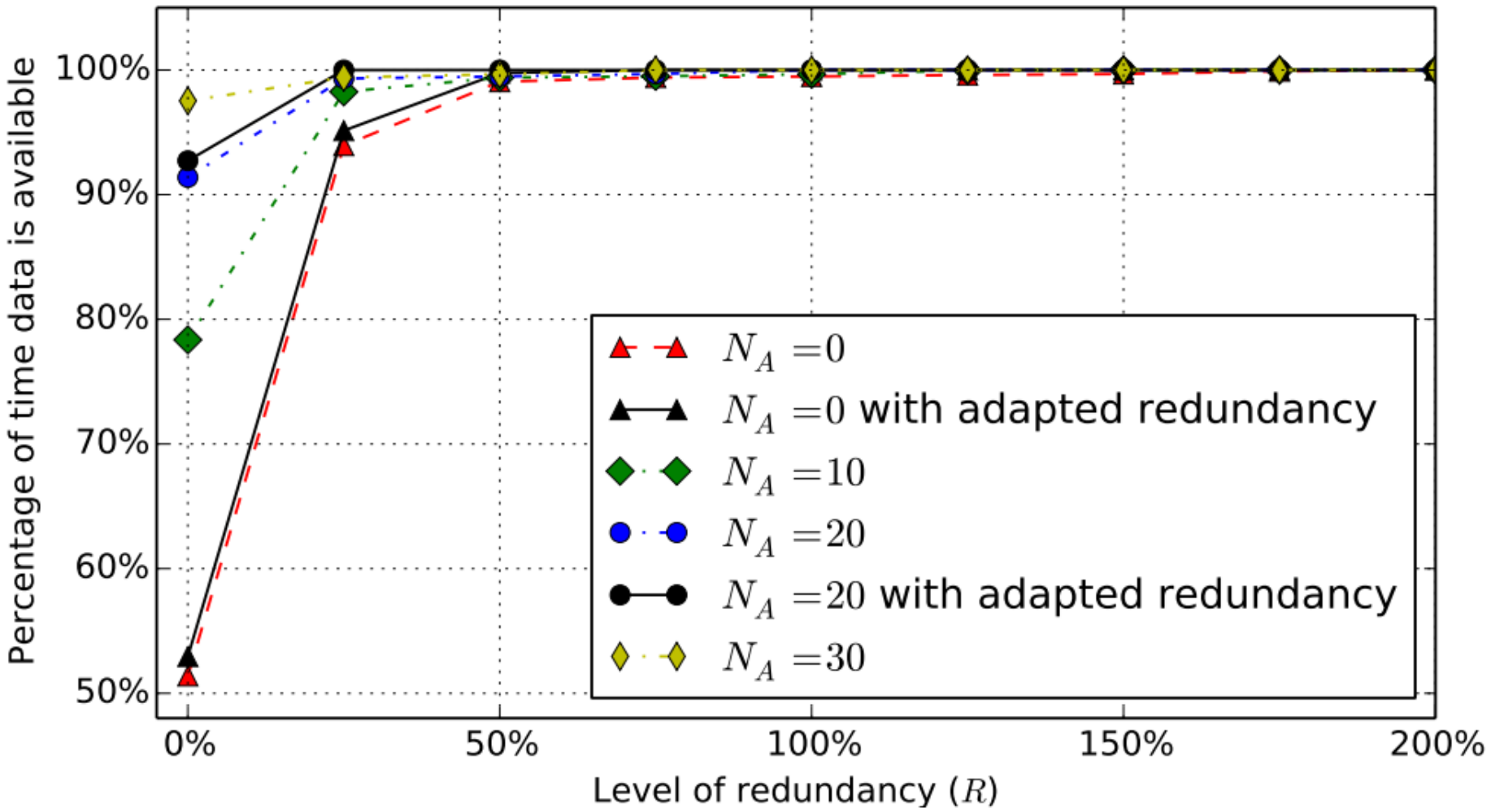
- Introducing high availability nodes



- Adding extra redundancy on the fly using recoding



Results


Percentage of time data is available
100%
90%
80%
70%
60%
50%
0%
50%
100%
150%
200%
Level of redundancy (R)
$N_A = 0$
$N_A = 0$ with adapted redundancy
$N_A = 10$
$N_A = 20$
$N_A = 20$ with adapted redundancy
$N_A = 30$

Conclusion and future work

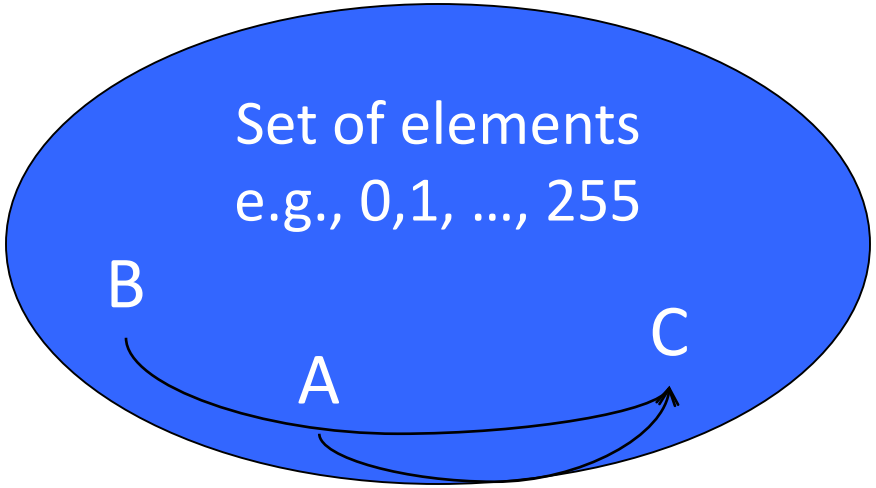
- We have shown using traces from a popular P2P mobile app, that most networks should have sufficient bandwidth for data to survive.
- We have measured how the amount of redundant storage affects data survival and proposed two solutions to lower costs.

- Future goals:
  - > Formalize data adaptation as a control theory problem
  - > Look at the performance of mobile nodes

TECHNISCHE UNIVERSITÄT DRESDEN

Finite Fields

Finite Fields – What are they?



Operations:
Addition, Multiplication

Property: closure

+ , x

Finite Fields GF(p)

Can write fields of the form GF(pn), where p is prime

Addition and multiplication over GF(p) are mod p

Focus on p = 2

Example:

GF(2) addition: equivalent to XOR

multiplication: equivalent to AND

How to divide? Multiply by multiplicative inverse

Finding the multiplicative inverse

1.- Can look for a^{-1} such that (a-1 * a) ≡ 1

2.- Can use the extended Euclidean algorithm

Finite Fields - Applying GF(2) to NC

- Example:
- GF(2) addition: XOR
- multiplication: AND
- Given 2 data packets
- P1: 01011001 P2: 10001001
- calculate the content of the coded packet P1+P2.

- What are the coefficients? P1 + P2 = 01011001 (XOR bit by bit)
  10001001
  11010000

What happens when “p” is not a prime?

Will modular arithmetic still work?

Example 1:

If gcd(a,n) ≠ 1, the last equation does not hold

e.g. $6 \times 3 = 18 = 2 \bmod 8$

and $6 \times 7 = 42 = 2 \bmod 8$ but

$3 \bmod 8 \neq 7 \bmod 8$

What happens when “p” is not a prime?

- Will modular arithmetic still work?

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

x	0	1	2	3
0	0	0	0	0
1	0	**1**	**2**	**3**
2	0	**2**	**0**	**2**
3	0	**3**	**2**	**1**

What about GF(2^n)?

- Since 2^n is not a prime, operations are defined in a different way
  - polynomial arithmetic
- Ordinary polynomial arithmetic:
  - A polynomial of degree n
  - $F(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0 x^0 = \Sigma$ ai xi
  - a_i are the coefficients, chosen from a set

- Operations:
- Addition $f(x) + g(x) = \Sigma (a_i + b_i) x_i$

- Multiplication $f(x) \times g(x) = \Sigma C_i x^i$
- with $c_k = a_0 b_k + a_1 b_{k-1} + \ldots + a_k b_0$

What about GF(2^n)?

- Polynomial arithmetic in GF(2^n):
- Arithmetic follows rules of polynomial arithmetic
- Arithmetic of coefficients is performed modulo 2
  - i.e.,using GF(2) addition/multiplication for coefficients of the same order
  - e.g., $(a_i + b_i) \bmod 2$
- If multiplication results in a polynomial greater than n-1, then the polynomial is reduced modulo an irreducible polynomial p(x)
  - Think of it as a mod p(x) operation: divide by p(x), keep the remainder
-

Example GF(2^2)

- Irreducible polynomial p(x) = x2 + x + 1 (111)b

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

How about 2 +3?

$2 = (10)_b$ and $3 = (11)_b$

As polynomials:

$2 \equiv x$ and $3 \equiv x + 1$

Thus, 2 + 3 becomes

$x + (x + 1) = 1$

Example GF(2^2)

Irreducible polynomial $p(x) = x^2 + x + 1$ $(111)_b$

Useable field -> primitive polynomial

x	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

How about 2 x 3?

$x \cdot (x + 1) = (x^2 + x) \bmod p(x)$

$$\begin{array}{r|l} & 1 \\ \hline x^2 + x + 1 & x^2 + x \\ & x^2 + x + 1 \\ \hline & 1 \end{array}$$

Equal number of each element: RLNC's properties are maintained when recoding

Multiplicative inverses: easy to spot in table

Can we compute without generating table?

TECHNISCHE UNIVERSITÄT DRESDEN

fifi-python

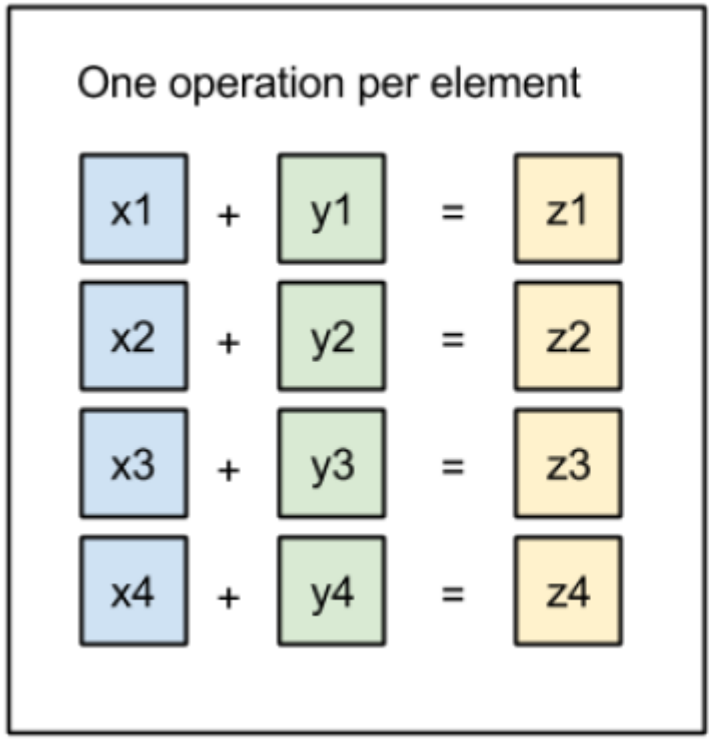
Input by morten v. pedersen

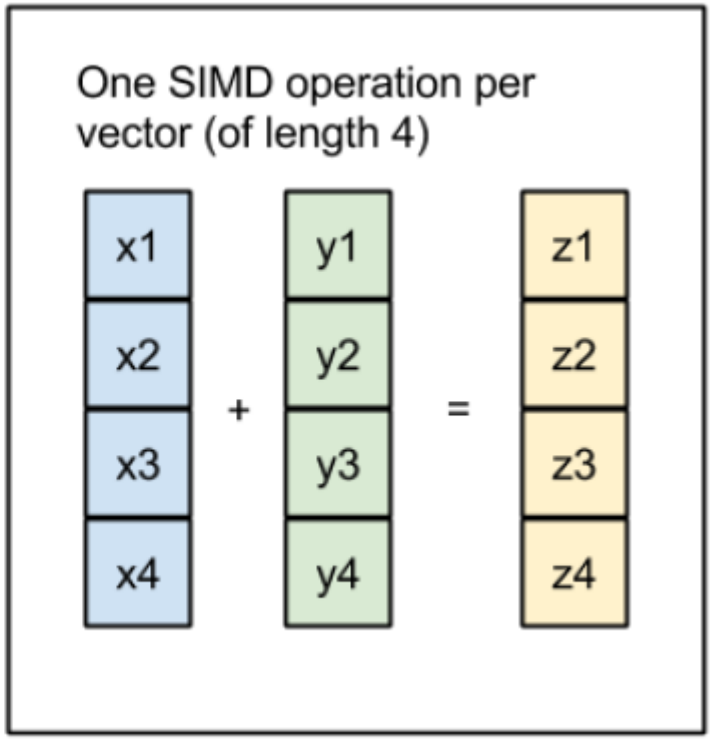
Finite field implementation

- Finite fields are a key component in many important applications:
- Cryptography.
- Error/Erasure correcting codes.
- Error detection e.g. CRC (Cyclic Redundancy Checks).
- Having an efficient implementation is important to evaluate system critical parameters:
- Throughput, latency, energy consumption, memory consumption etc.
- Today we will use the fifi-python library to perform finite field computations.

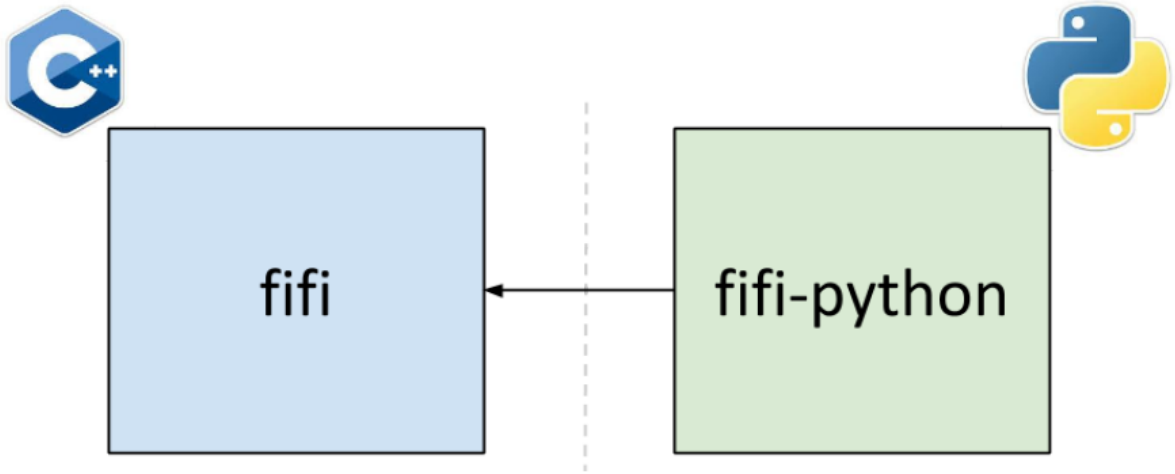
A note on performance

- In the past few year performance has increase significantly
- This has happened because we understand how to utilize specialized instructions on
- modern CPUs called SIMD (Single Instruction Multiple Data).
- fifi and fifi-python uses this extensively (on both x86 and ARM CPUs)





About fifi and fifi-python




```
fifi-python/
├── config.py
├── examples
│   ├── fifi_simple_api.py
│   ├── fifi_simple_api_table_example.py
│   ├── hello_fifi.py
│   └── hello_simple_api.py
├── LICENSE.rst
├── NEWS.rst
├── README.rst
├── src
│   └── fifi_python
├── test
│   ├── data.json
│   └── fifi_python_tests.py
├── waf
└── wscript
```

TECHNISCHE UNIVERSITÄT DRESDEN

Exploring the API

- To use the library `import` it into your scripts.

In []:
```
import fifi
```

- You can explore the API using the built-in `help()` function

In [2]:
```
help(fifi)
```

```
Help on module fifi:

NAME
    fifi

FILE
    /home/mvp/Dropbox/work_code/notebooks/slides_fifi_python/fifi.so

CLASSES
    Boost.Python.instance(__builtin__.object)
        extended_log_table_binary16
        extended_log_table_binary4
        extended_log_table_binary8
        full_table_binary4
        full_table_binary8
        log_table_binary16
        log_table_binary4
        log_table_binary8
        optimal_prime_prime2325
        simple_online_binary
        simple_online_binary16
        simple_online_binary4
        simple_online_binary8

    class extended_log_table_binary16(Boost.Python.instance)
     |  A finite field implementation
     |
```

Using **help()**

- You can also get more specific help, by passing a specific class or function.

In [4]:
```
import fifi
help(fifi.simple_online_binary4.add)
```

```
Help on method add:

add(...) unbound fifi.simple_online_binary4 method
    Returns the sum of two field elements.

            :param a: The augend.

            :param b: The addend.
            :returns: The sum of a and b.
```

A small example

In [8]:
```
import fifi

field = fifi.simple_online_binary4()
order = 2**4

table = ''


for i in range(order):
    for j in range(order):
        table += '{:02d} '.format(field.multiply(i,j))
    table += '\n'

print(table)
```

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
00 02 04 06 08 10 12 14 03 01 07 05 11 09 15 13
00 03 06 05 12 15 10 09 11 08 13 14 07 04 01 02
00 04 08 12 03 07 11 15 06 02 14 10 05 01 13 09
00 05 10 15 07 02 13 08 14 11 04 01 09 12 03 06
00 06 12 10 11 13 07 01 05 03 09 15 14 08 02 04
00 07 14 09 15 08 01 06 13 10 03 04 02 05 12 11
00 08 03 11 06 14 05 13 12 04 15 07 10 02 09 01
00 09 01 08 02 11 03 10 04 13 05 12 06 15 07 14
00 10 07 13 14 04 09 03 15 05 08 02 01 11 06 12
00 11 05 14 10 01 15 04 07 12 02 09 13 06 08 03
00 12 11 07 05 09 14 02 10 06 01 13 15 03 04 08
00 13 09 04 01 12 08 05 02 15 11 06 03 14 10 07
00 14 15 01 13 03 02 12 09 07 06 08 04 10 11 05
00 15 13 02 09 06 04 11 01 14 12 03 08 07 05 10
```

Hands-On

- Make sure fifi-python is installed (lecture slides)
- Create a script (e.g. print_table.py) and write the code to print both the multiplication and division tables.
- Run the script python print_table.py.

Using the Simple API

- The previous example use the field object directly.
- It is also possible to standard arithmetic operations (+,-,*,/,~) with field elements.
- To use it copy the fifi-python/examples/fifi_simple_api.py to your scripts
- folder.

In [3]:
```
#! /usr/bin/env python

# Copyright Steinwurf ApS 2011-2013.
# Distributed under the "STEINWURF RESEARCH LICENSE 1.0".
# See accompanying file LICENSE.rst or
# http://www.steinwurf.com/licensing

from fifi_simple_api import B4


def main():
    a = B4(13)
    b = B4(7)

    print("{a} + {b} = {result}".format(a=a, b=b, result=a + b))
    print("{a} - {b} = {result}".format(a=a, b=b, result=a - b))
    print("{a} * {b} = {result}".format(a=a, b=b, result=a * b))
    print("{a} / {b} = {result}".format(a=a, b=b, result=a / b))
    print("~{a} = {result}".format(a=a, result=~a))

if __name__ == '__main__':
    main()
```

```
13 + 7 = 10
13 - 7 = 10
13 * 7 = 5
13 / 7 = 8
~13 = 4
```

TECHNISCHE UNIVERSITÄT DRESDEN

Discussion Look-up Table vs. Online