DSS

Data & Storage Services

CERN IT Department

Introduction to data management

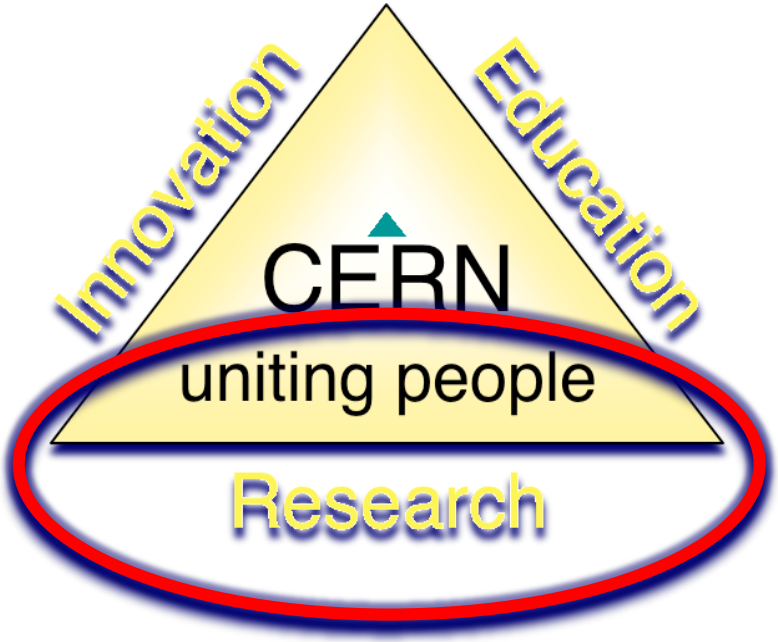
Alberto Pace

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

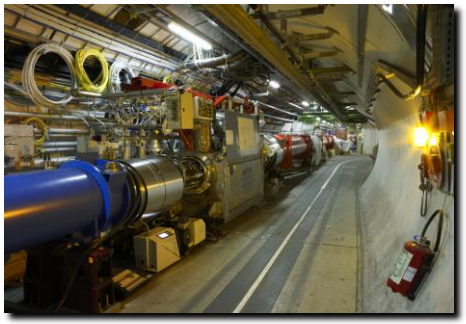
CERN

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

The mission of CERN



We are here



Accelerating particle beams

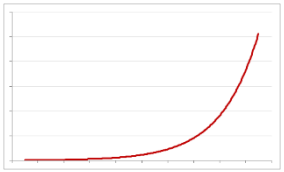
Detecting particles (experiments)

Large-scale computing (Analysis)

Discovery

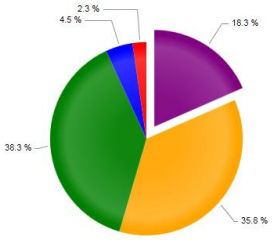
The need for computing in research

- **Scientific research in recent years has exploded the computing requirements**
  - Computing has been the strategy to reduce the cost of traditional research



At constant cost, exponential growth of performances

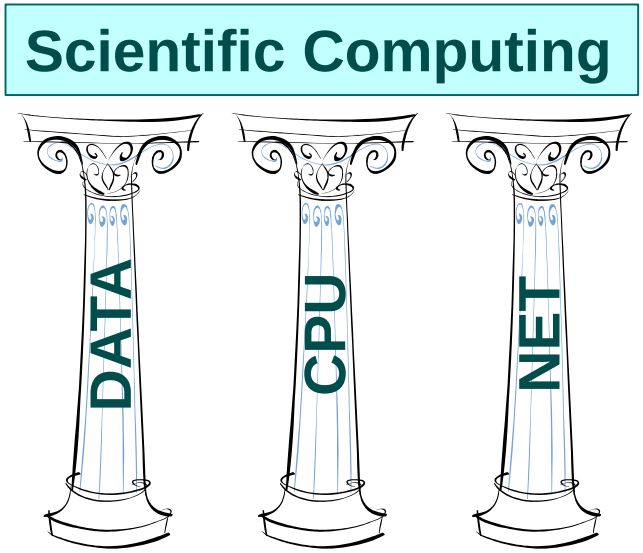
  - Computing has opened new horizons of research not only in High Energy Physics



Return in computing investment higher than other fields: Budget available for computing increased, growth is more than exponential

The need for storage in computing

- Scientific computing for large experiments is typically based on a distributed infrastructure
- Storage is one of the main pillars
- Storage requires Data Management…

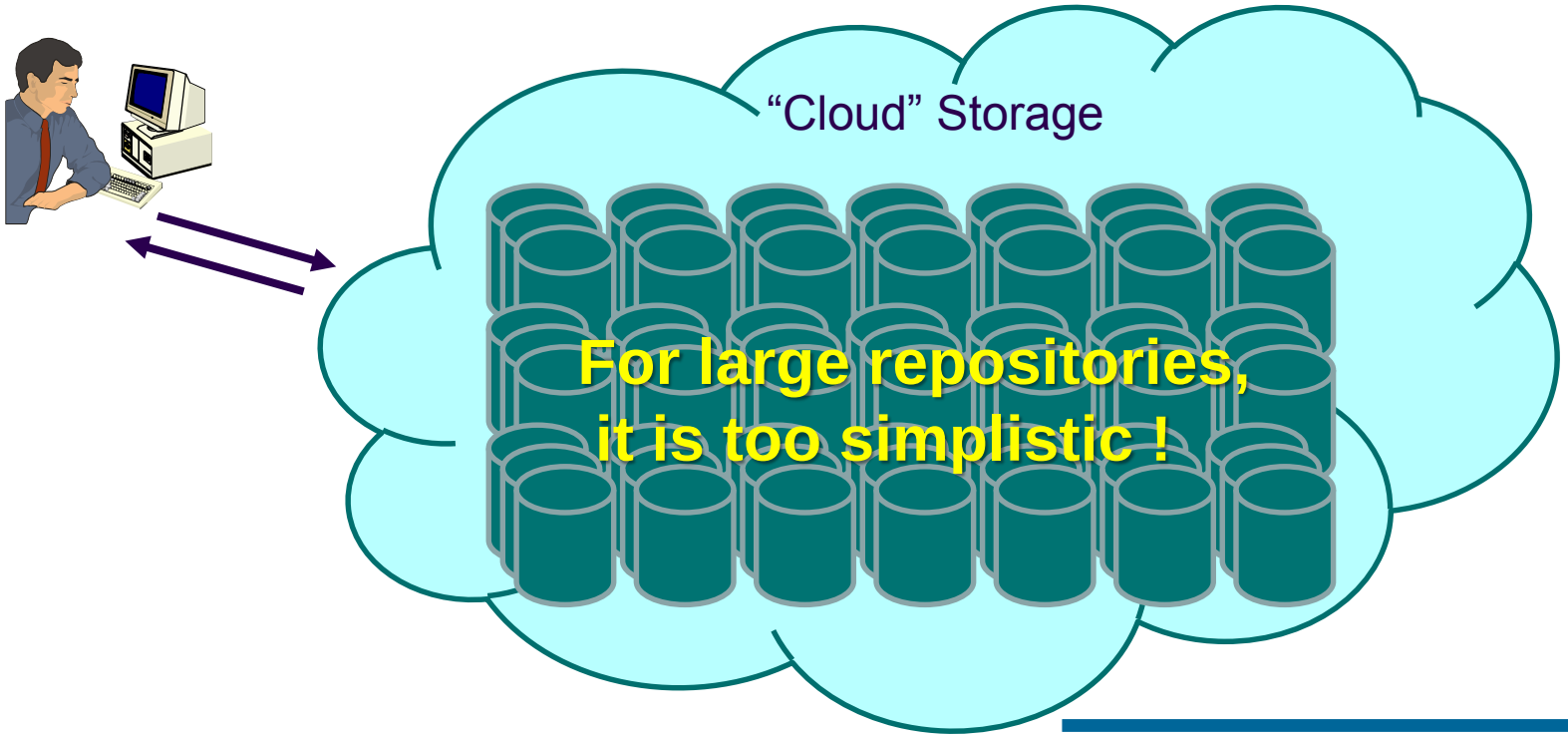


“Why” data management ?

- **Data Management solves the following problems**
  - Data reliability
  - Access control
  - Data distribution
  - Data archives, history, long term preservation
  - In general:
    - Empower the implementation of a workflow for data processing

Can we make it simple ?

- **A simple storage model: all data into the same container**
  - Uniform, simple, easy to manage, no need to move data
  - Can provide sufficient level of performance and reliability

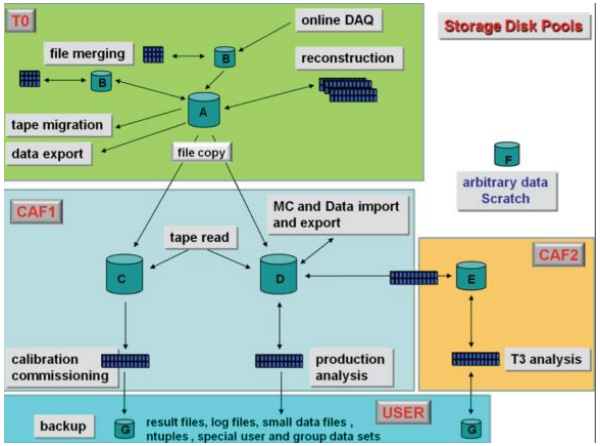


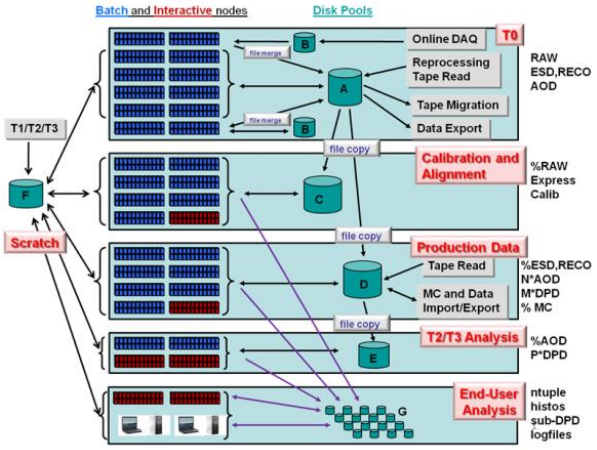
Why multiple pools and quality ?

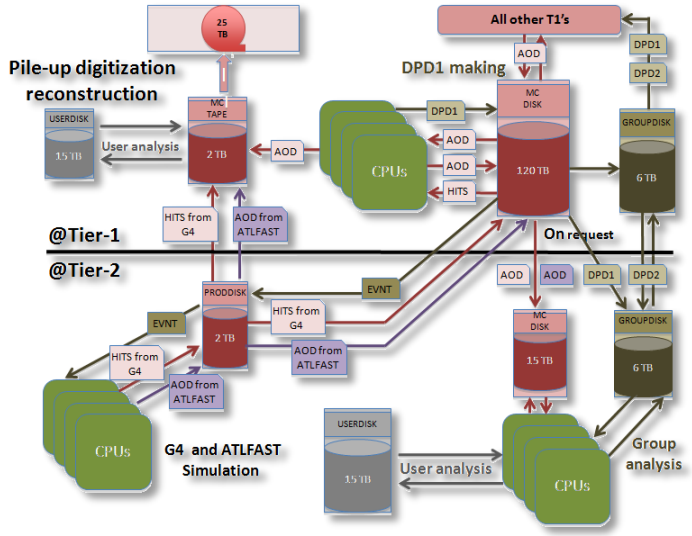
- **Derived data used for analysis and accessed by thousands of nodes**
  - Need high performance, Low cost, minimal reliability (derived data can be recalculated)
- **Raw data that need to be analyzed**
  - Need high performance, High reliability, can be expensive (small sizes)
- **Raw data that has been analyzed and archived**
  - Must be low cost (huge volumes), High reliability (must be preserved), performance not necessary

So, … what is data management ?

- **Examples from LHC experiment data models**



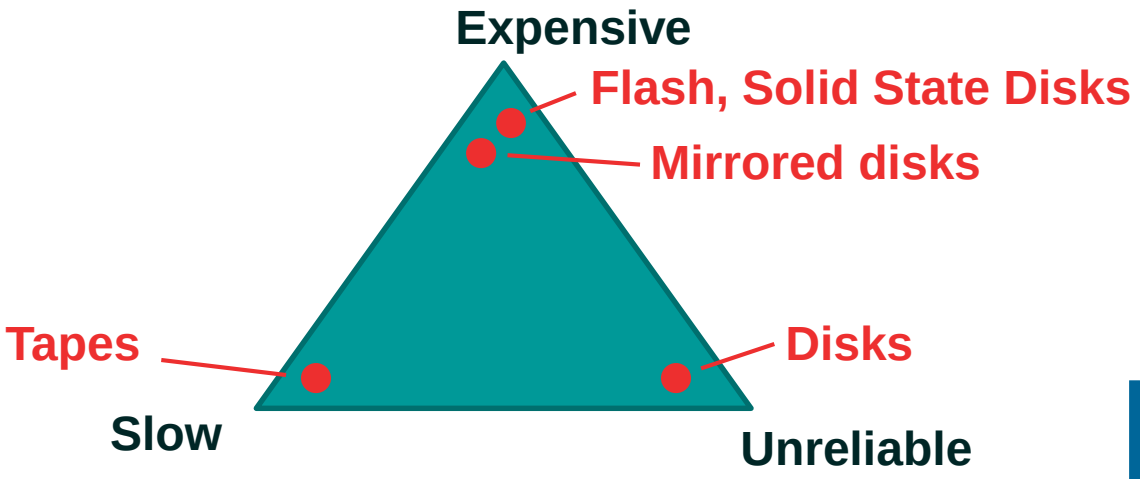




- **Two building blocks to empower data processing**
  - Data pools with different quality of services
  - Tools for data transfer between pools

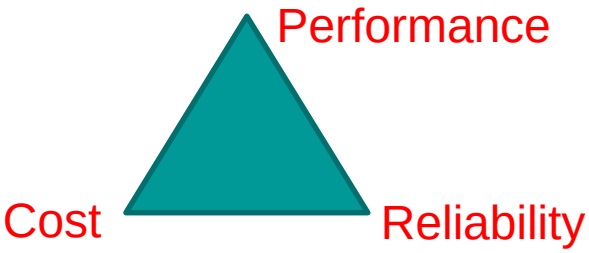
Data pools

- **Different quality of services**
  - Three parameters: (Performance, Reliability, Cost)
  - You can have two but not three

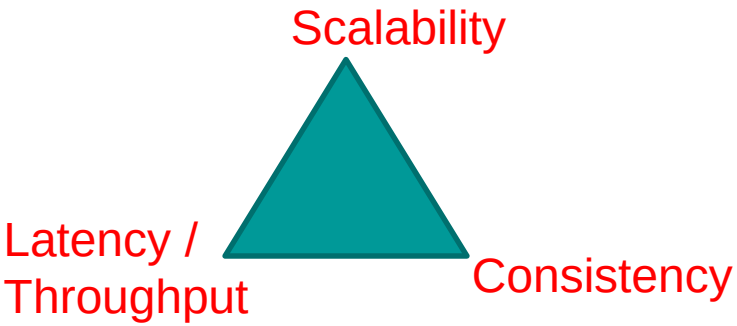


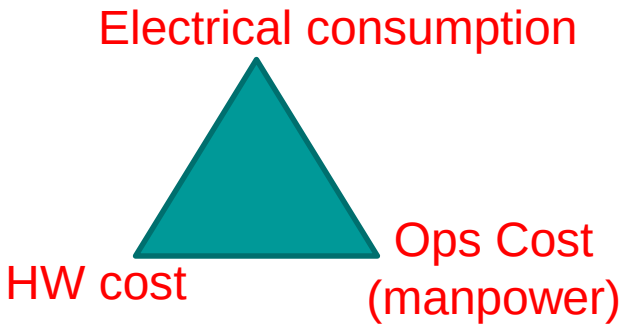
But the balance is not as simple

- Many ways to split (performance, reliability, cost)



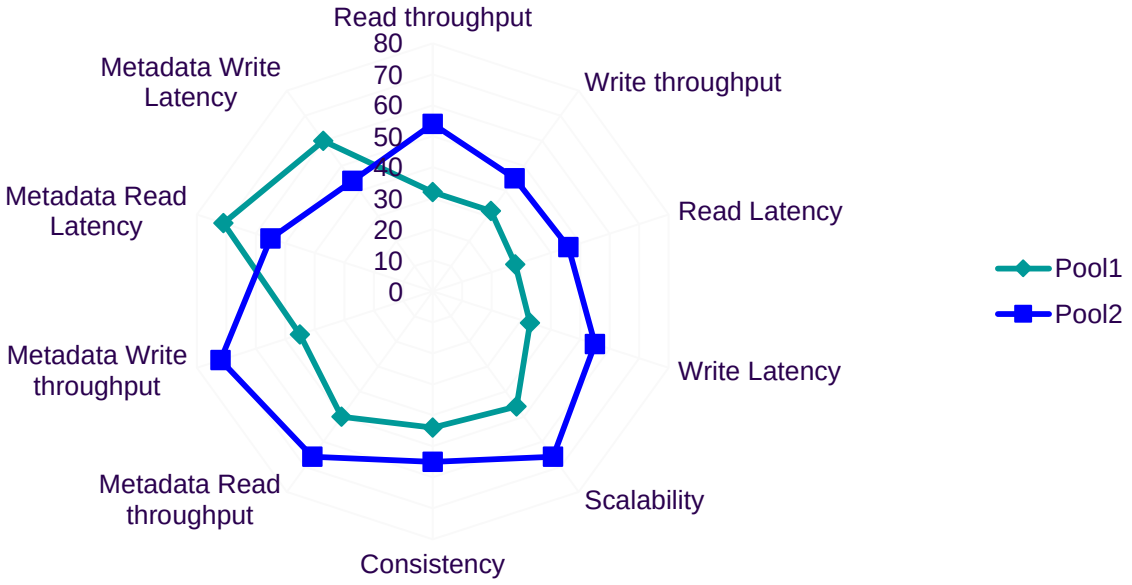
- Performance has many sub-parameters
- Cost has many sub-parameters
- Reliability has many sub-parameters





And reality is complicated

- Key requirements: Simple, Scalable, Consistent, Reliable, Available, Manageable, Flexible, Performing, Cheap, Secure.
- Aiming for “à la carte” services (storage pools) with on-demand “quality of service”
- And where is scalability ?



Areas of research in Data Management

Reliability, Scalability, Security, Manageability

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

Storage Reliability

- **Reliability is related to the probability to lose data**
  - Def: “the probability that a storage device will perform an arbitrarily large number of I/O operations without data loss during a specified period of time”
- **Reliability of the “service” depends on the environment (energy, cooling, people, ...)**
  - Will not discuss this further
- **Reliability of the “service” starts from the reliability of the underlying hardware**
  - Example of disk servers with simple disks: reliability of service = reliability of disks
- **But data management solutions can increase the reliability of the hardware at the expenses of performance and/or additional hardware / software**
  - Disk Mirroring
  - Redundant Array of Inexpensive Disks (RAID)

Hardware reliability

- **Do we need tapes ?**
- **Tapes have a bad reputation in some use cases**
  - Slow in random access mode
    - high latency in mounting process and when seeking data (F-FWD, REW)
  - Inefficient for small files (in some cases)
  - Comparable cost per (peta)byte as hard disks
- **Tapes have also some advantages**
  - Fast in sequential access mode
    - > 2x faster than disk, with physical read after write verification
  - Several orders of magnitude more reliable than disks
    - Few hundreds GB loss per year on 80 PB tape repository
    - Few hundreds TB loss per year on 50 PB disk repository
  - No power required to preserve the data
  - Less physical volume required per (peta)byte
  - Inefficiency for small files issue resolved by recent developments
  - Nobody can delete hundreds of PB in minutes
- **Bottom line: if not used for random access, tapes have a clear role in the architecture**

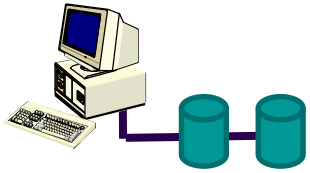
Reminder: types of RAID

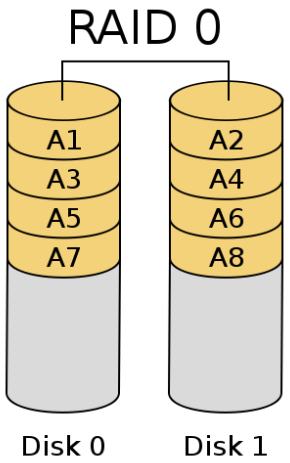
- **RAID0**
  - Disk striping
- **RAID1**
  - Disk mirroring
- **RAID5**
  - Parity information is distributed across all disks
- **RAID6**
  - Uses Reed–Solomon error correction, allowing the loss of 2 disks in the array without data loss

http://en.wikipedia.org/wiki/RAID

Reminder: types of RAID

- **RAID0**
  - Disk striping
- **RAID1**
  - Disk mirroring
- **RAID5**
  - Parity information is distributed across all disks
- **RAID6**
  - Uses Reed–Solomon error correction, allowing the loss of 2 disks in the array without data loss

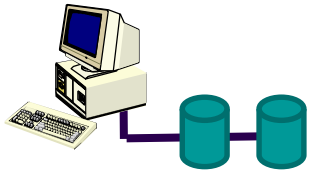


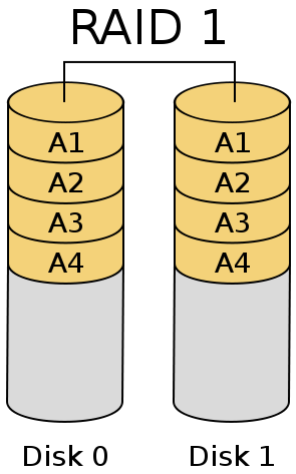


http://en.wikipedia.org/wiki/RAID

Reminder: types of RAID

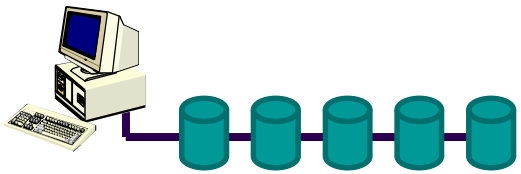
- **RAID0**
  - Disk striping
- **RAID1**
  - Disk mirroring
- **RAID5**
  - Parity information is distributed across all disks
- **RAID6**
  - Uses Reed–Solomon error correction, allowing the loss of 2 disks in the array without data loss

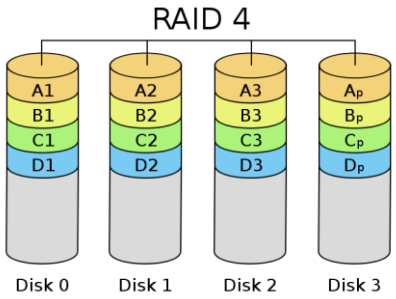


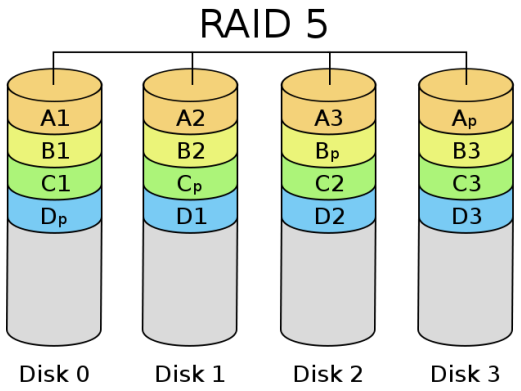


http://en.wikipedia.org/wiki/RAID

Reminder: types of RAID





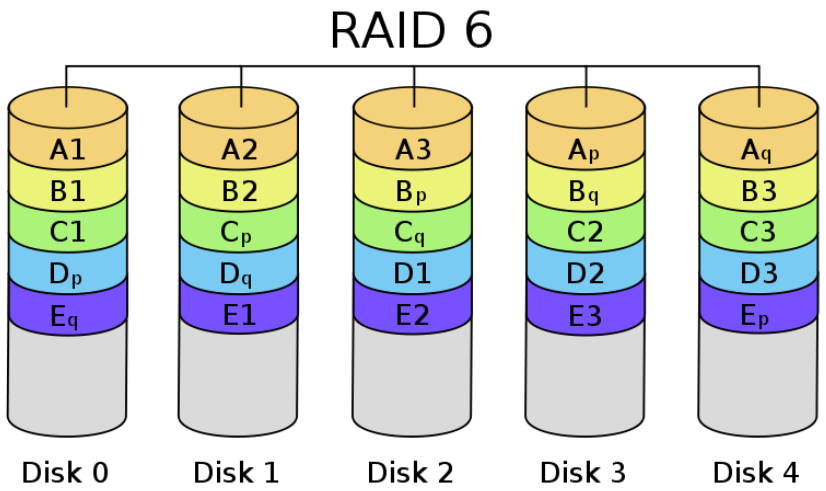


- **RAID0**
  - Disk striping
- **RAID1**
  - Disk mirroring
- **RAID5**
  - Parity information is distributed across all disks
- **RAID6**
  - Uses Reed–Solomon error correction, allowing the loss of 2 disks in the array without data loss

http://en.wikipedia.org/wiki/RAID

Reminder: types of RAID

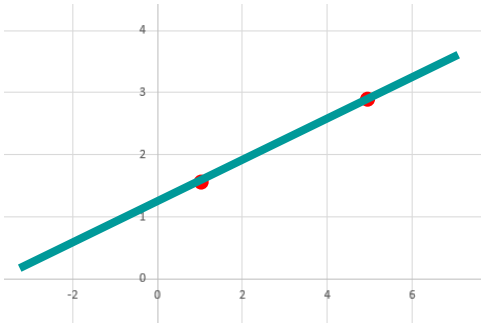
- **RAID0**
  - Disk striping
- **RAID1**
  - Disk mirroring
- **RAID5**
  - Parity information is distributed across all disks
- **RAID6**
  - Uses Reed–Solomon error correction, allowing the loss of 2 disks in the array without data loss



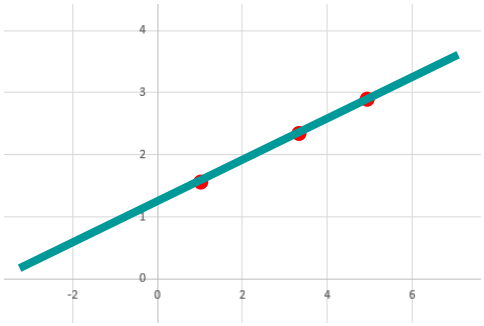
http://en.wikipedia.org/wiki/RAID

Understanding error correction

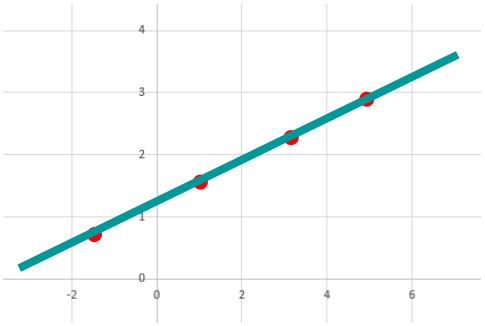
- **A line is defined by 2 numbers: a, b**
  - (a, b) is the information
  - y = ax + b
- **Instead of transmitting a and b, transmit some points on the line at known abscissa. 2 points define a line. If I transmit more points, these should be aligned.**



2 points



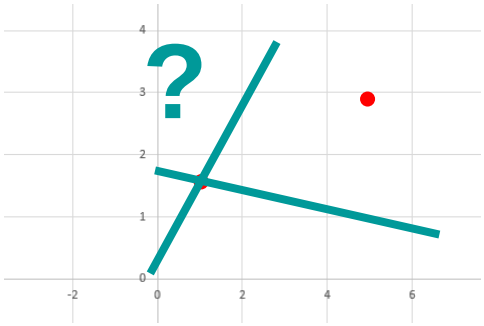
3 points



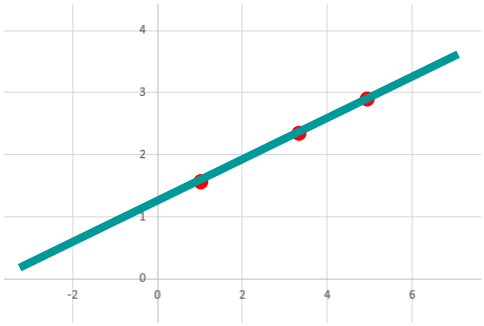
4 points

If we lose some information …

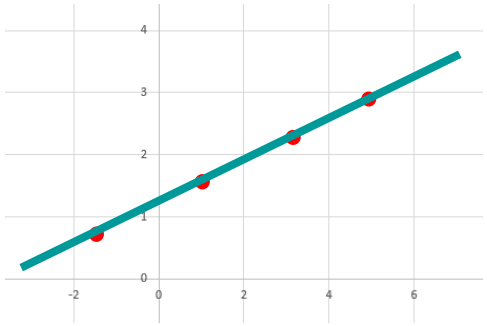
- If we transmit more than 2 points, we can lose any point, provided the total number of point left is >= 2



1 point instead of 2 information lost



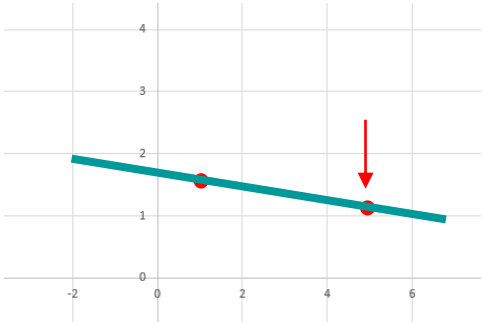
2 points instead of 3



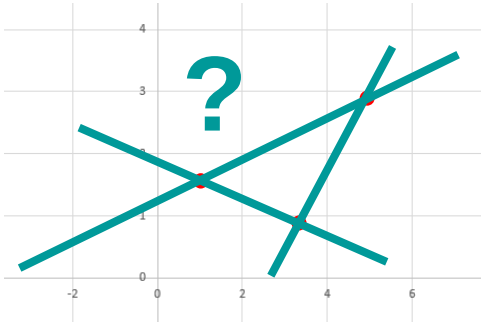
2 or 3 points instead of 4

If we have an error …

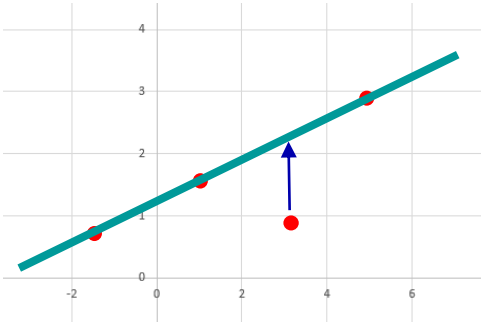
- **If there is an error, I can detect it if I have transmitted more than 2 points, and correct it if have transmitted more than 3 points**



Information lost
(and you do not notice)



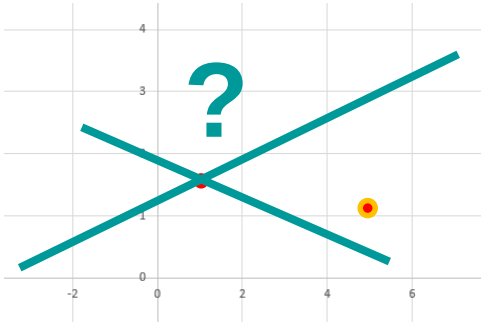
Error detection
Information is lost
(and you notice)



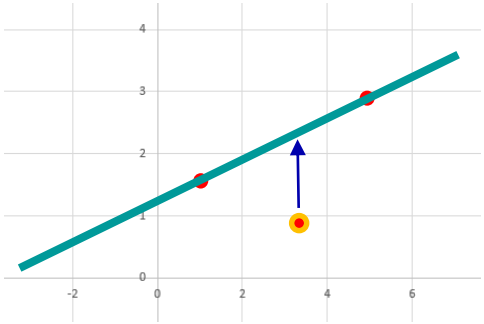
Error correction
Information is recovered

If you have checksumming on data …

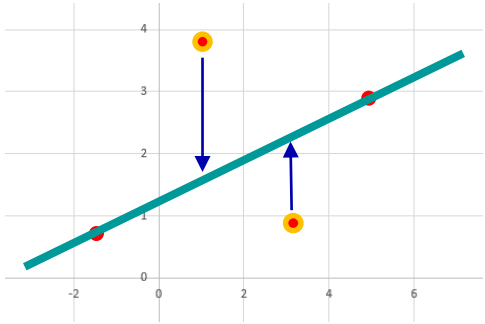
- You can detect errors by verifying the consistency of the data with the respective checksums. So you can detect errors independently.
- … and use all redundancy for error correction



Information lost
(and you notice)



Error correction
Information is recovered



2 Error corrections possible
Information is recovered

Reed–Solomon error correction …

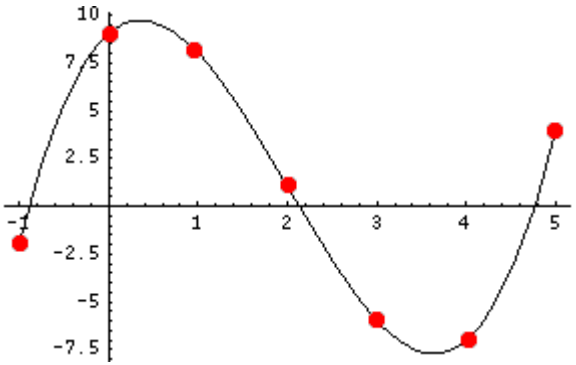
- .. is an error-correcting code that works by oversampling (by n points) a polynomial constructed from the data
- Any m distinct points uniquely determine a polynomial of degree, at most, m − 1
- The sender determines the polynomial (of degree m − 1), that represents the m data points. The polynomial is "encoded" by its evaluation at n+m points. If during transmission, the number of corrupted values is < n the receiver can recover the original polynomial.
- Implementation examples:
  - n = 0 no redundancy
  - n = 1 is Raid 5 (parity)
  - n = 2 is Raid 6 (Reed Solomon, double / diagonal parity)
  - n = 3 is … (Triple parity)

http://en.wikipedia.org/wiki/Reed-Solomon

Reed–Solomon (simplified) Example

- 4 Numbers to encode: { 1, -6, 4, 9 } (m=4)
- polynomial of degree 3 (m − 1):

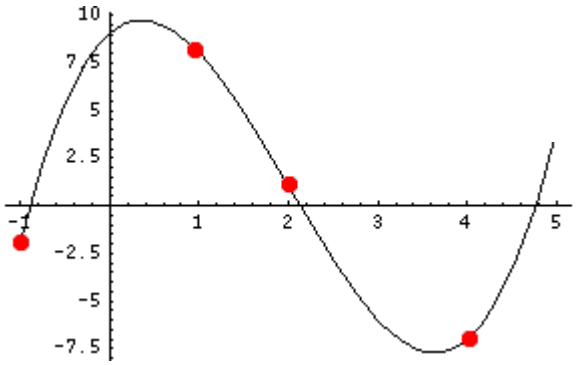
$$y = x^3 - 6x^2 + 4x + 9$$



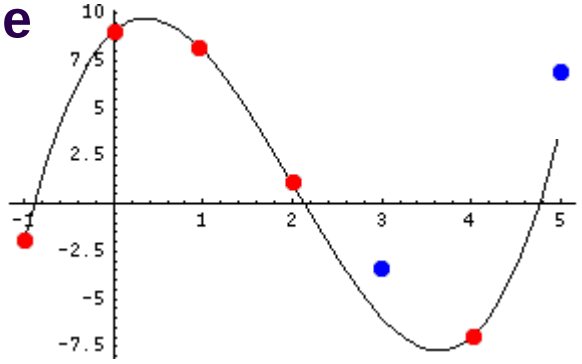
- We encode the polynomial with n + m = 7 points
  { -2, 9, 8, 1, -6, -7, 4 }

Reed–Solomon (simplified) Example

- To reconstruct the polynomial, any 4 points are enough: we can lose any 3 points.



- We can have an error on any 2 points that can be corrected: We need to identify the 5 points "aligned" on the only one polynomial of degree 3 possible



http://kernel.org/pub/linux/kernel/people/hpa/raid6.pdf

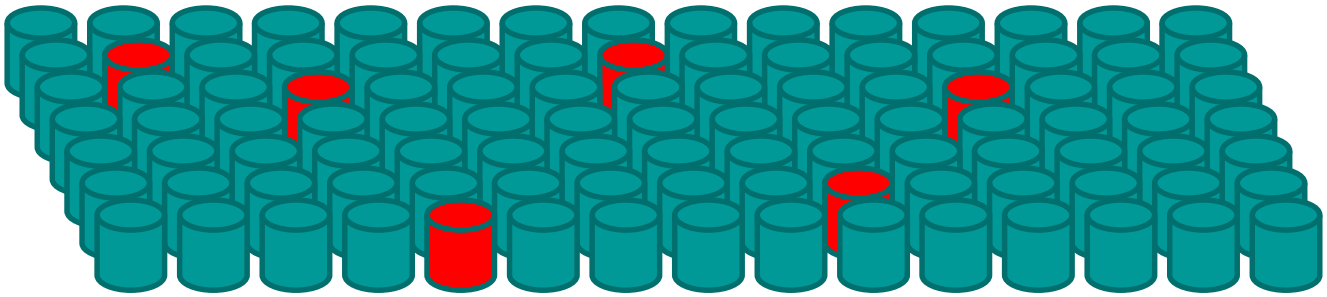
Error detection vs Error correction

- **With Reed-Solomon:**
  - If the number of corrupted values is **=** n we can **only detect** the error
  - If the number of corrupted values is **<** n we can **correct** the error
- **However, by adding a checksum or hash on each point, we can individually identify the corrupted values**
  - If checksum has been added, Reed-Solomon can **correct** corrupted values **≤** n

Reliability calculations

- **With RAID, the final reliability depends on several parameters**
  - The reliability of the hardware
  - The type of RAID
  - The number of disks in the set
- **Already this gives lot of flexibility in implementing arbitrary reliability**

Raid 5 reliability



- Disk are regrouped in sets of equal size. If c is the capacity of the disk and n is the number of disks, the sets will have a capacity of

$$c\ (n-1)$$

example: 6 disks of 1TB can be aggregated to a “reliable” set of 5TB

- The set is immune to the loss of 1 disk in the set. The loss of 2 disks implies the loss of the entire set content.

Some calculations for Raid 5

- Disks MTBF is between 3 x 10^5 and 1.2 x 10^6 hours
- Replacement time of a failed disk is < 4 hours
- Probability of 1 disk to fail within the next 4 hours

$$P_f = \frac{Hours}{MTBF} = \frac{4}{3 \times 10^5} = 1.3 \times 10^{-5}$$

Some calculations for Raid 5

- Disks MTBF is between 3 x 10^5 and 1.2 x 10^6 hours
- Replacement time of a failed disk is < 4 hours
- Probability of 1 disk to fail within the next 4 hours

$$P_f = \frac{Hours}{MTBF} = \frac{4}{3\times10^5} = 1.3\times10^{-5}$$

- Probability to have a failing disk in the next 4 hours in a 15 PB computer centre (15'000 disks)

$$P_{f15000} = 1-(1-P_f)^{15000} = 0.18$$

Some calculations for Raid 5

- Disks MTBF is between 3 x 10^5 and 1.2 x 10^6 hours
- Replacement time of a failed disk is < 4 hours
- Probability of 1 disk to fail within the next 4 hours

$$P_f = \frac{Hours}{MTBF} = \frac{4}{3\times10^5} = 1.3\times10^{-5}$$

p(A and B) = p(A) * p(B/A)

if A,B independent : p(A) * p(B)

- Probability to have a failing disk in the next 4 hours in a 15 PB computer centre (15'000 disks)

$$P_{f15000} = 1-(1-P_f)^{15000} = 0.18$$

- Imagine a Raid set of 10 disks. Probability to have one of the remaining disk failing within 4 hours

$$P_{f9} = 1-(1-P_f)^{9} = 1.2\times10^{-4}$$

Some calculations for Raid 5

- Disks MTBF is between 3 x 10^5 and 1.2 x 10^6 hours
- Replacement time of a failed disk is < 4 hours
- Probability of 1 disk to fail within the next 4 hours

$$P_f = \frac{Hours}{MTBF} = \frac{4}{3\times10^5} = 1.3\times10^{-5}$$

p(A and B) = p(A) * p(B/A)

if A,B independent : p(A) * p(B)

- Probability to have a failing disk in the next 4 hours in a 15 PB computer centre (15'000 disks)

$$P_{f15000} = 1-(1-P_f)^{15000} = 0.18$$

- Imagine a Raid set of 10 disks. Probability to have one of the remaining disk failing within 4 hours

$$P_{f9} = 1-(1-P_f)^{9} = 1.2\times10^{-4}$$

- However the second failure may not be independent from the first one. There is no way to calculate this probability ! We can arbitrarily increase it by two orders of magnitude to account the dependencies (over temperature, high noise, EMP, high voltage, faulty common controller, ...)

$$P_{f9corrected} = 1-(1-P_f)^{900} = 0.0119$$

Some calculations for Raid 5

- Disks MTBF is between 3 x 10^5 and 1.2 x 10^6 hours
- Replacement time of a failed disk is < 4 hours
- Probability of 1 disk to fail within the next 4 hours

p(A and B) = p(A) * p(B/A)

if A,B independent : p(A) * p(B)

$$P_f = \frac{Hours}{MTBF} = \frac{4}{3\times10^5} = 1.3\times10^{-5}$$

- Probability to have a failing disk in the next 4 hours in a 15 PB computer centre (15'000 disks)

$$P_{f15000} = 1-(1-P_f)^{15000} = 0.18$$

- Imagine a Raid set of 10 disks. Probability to have one of the remaining disk failing within 4 hours

$$P_{f9} = 1-(1-P_f)^{9} = 1.2\times10^{-4}$$

- However the second failure may not be independent from the first one. There is no way to calculate this probability ! We can arbitrarily increase it by two orders of magnitude to account the dependencies (over temperature, high noise, EMP, high voltage, faulty common controller, ...)

$$P_{f9corrected} = 1-(1-P_f)^{900} = 0.0119$$

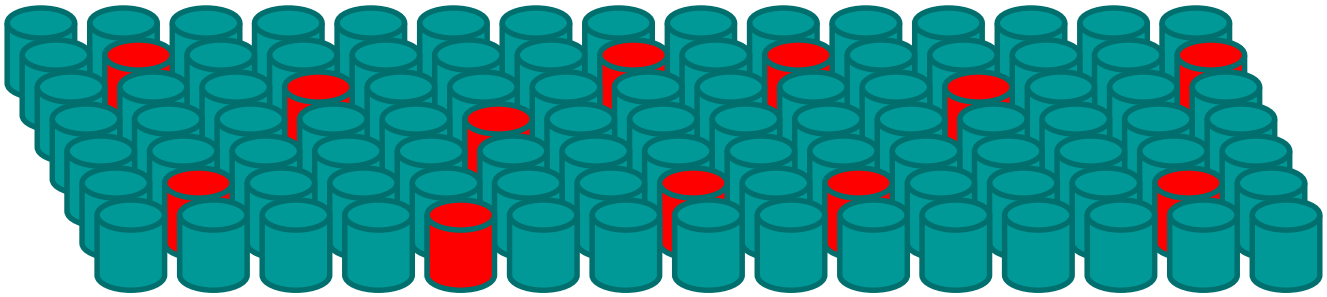
- Probability to lose computer centre data in the next 4 hours

$$P_{loss} = P_{f15000}\times P_{f9corrected} = 6.16\times10^{-4}$$

- Probability to lose data in the next 10 years

$$P_{loss10yrs} = 1-(1-P_{loss})^{10\times365\times6} = 1\text{-}10^{-21} \cong 1$$

Raid 6 reliability



- Disk are regrouped in sets of arbitrary size. If c is the capacity of the disk and n is the number of disks, the sets will have a capacity of

$$c\ (n-2)$$

example: 12 disks of 1TB can be aggregated to a “reliable” set of 10TB

- The set is immune to the loss of 2 disks in the set. The loss of 3 disks implies the loss of the entire set content.

Same calculations for Raid 6

- Probability of 1 disk to fail within the next 4 hours

$$P_f = \frac{Hours}{MTBF} = \frac{4}{3\times10^5} = 1.3\times10^{-5}$$

- Imagine a raid set of 10 disks. Probability to have one of the remaining 9 disks failing within 4 hours (increased by two orders of magnitudes)

$$P_{f9} = 1-(1-P_f)^{900} = 1.19\times10^{-2}$$

- Probability to have another of the remaining 8 disks failing within 4 hours (also increased by two orders of magnitudes)

$$P_{f8} = 1-(1-P_f)^{800} = 1.06\times10^{-2}$$

- Probability to lose data in the next 4 hours

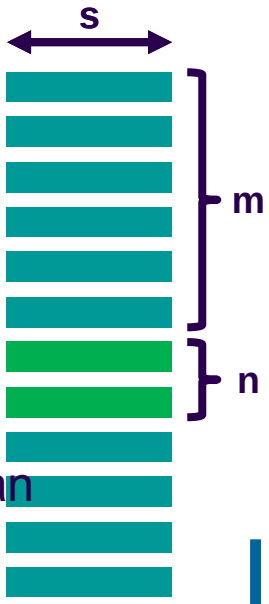
$$P_{loss} = P_{f15000}\times P_{f99}\times P_{f98} = 2.29\times10^{-5}$$

- Probability to lose data in the next 10 years

$$P_{loss10yrs} = 1-(1-P_{loss})^{10\times365\times6} = 0.394$$

Arbitrary reliability

- **RAID is “disks” based. This lacks of granularity**
- **For increased flexibility, an alternative would be to use files ... but files do not have constant size**
- **File “chunks” (or “blocks”) is the solution**
  - Split files in chunks of size “s”
  - Group them in sets of “m” chunks
  - For each group of “m” chunks, generate “n” additional chunks so that
    - For any set of “m” chunks chosen among the “m+n” you can reconstruct the missing “n” chunks
  - Scatter the “m+n” chunks on independent storage



Arbitrary reliability with the "chunk" based solution

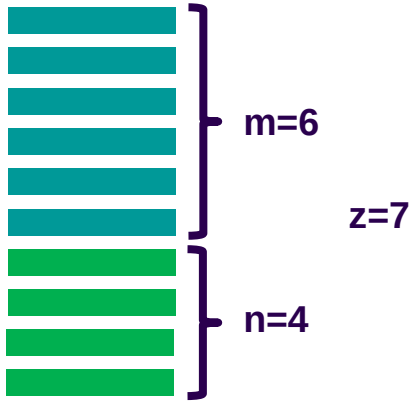
- **The reliability is independent form the size "s" which is arbitrary.**
  - Note: both large and small "s" impact performance
- **Whatever the reliability of the hardware is, the system is immune to the loss of "n" simultaneous failures from pools of "m+n" storage chunks**
  - Both "m" and "n" are arbitrary. Therefore arbitrary reliability can be achieved
- **The fraction of raw storage space loss is n / (n + m)**
- **Note that space loss can also be reduced arbitrarily by increasing m**
  - At the cost of increasing the amount of data loss if this would ever happen

Analogy with the gambling world

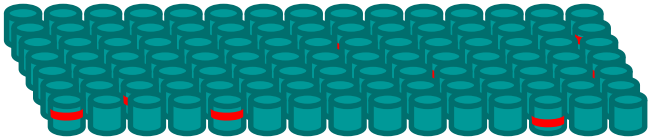
- We just demonstrated that you can achieve “arbitrary reliability” at the cost of an “arbitrary low” amount of disk space. This is possible because you increase the amount of data you accept loosing when this rare event happens.
- In the gambling world there are several playing schemes that allows you to win an arbitrary amount of money with an arbitrary probability.
- Example: you can easily win 100 Euros at > 99 % probability ...
  - By playing up to 7 times on the “Red” of a French Roulette and doubling the bet until you win.
  - The probability of not having a “Red” for 7 times is $(19/37)^7 = 0.0094$)
  - You just need to take the risk of loosing 12’700 euros with a 0.94 % probability

Amount		Win		Lost	
Bet	Cumulated	Probability	Amount	Probability	Amount
100	100	48.65%	100	51.35%	100
200	300	73.63%	100	26.37%	300
400	700	86.46%	100	13.54%	700
800	1500	93.05%	100	6.95%	1500
1600	3100	96.43%	100	3.57%	3100
3200	6300	98.17%	100	1.83%	6300
6400	12700	99.06%	100	0.94%	12700

Practical comments



- **n can be …**
  - 1 = Parity
  - 2 = Parity + Reed-Solomon, double parity
  - 3 = Reed Solomon, ZFS triple parity
- **m chunks of any (m + n) sets are enough to obtain the information. Must be saved on independent media**
  - Performance can depend on m (and thus on s, the size of the chunks): The larger m is, the more the reading can be parallelized
  - Until the client bandwidth is reached
- **For n > 2 Reed Solomon has a computational impact affecting performances**
  - Alternate encoding algorithms are available requiring z chunks to reconstruct the data, being $m < z < m+n$ (see example later on with LDPC).
  - These guarantees high performance at the expenses of additional storage. When m=z we fall back in the “optimal” storage scenario



http://blogs.sun.com/ahl/entry/triple_parity_raid_z

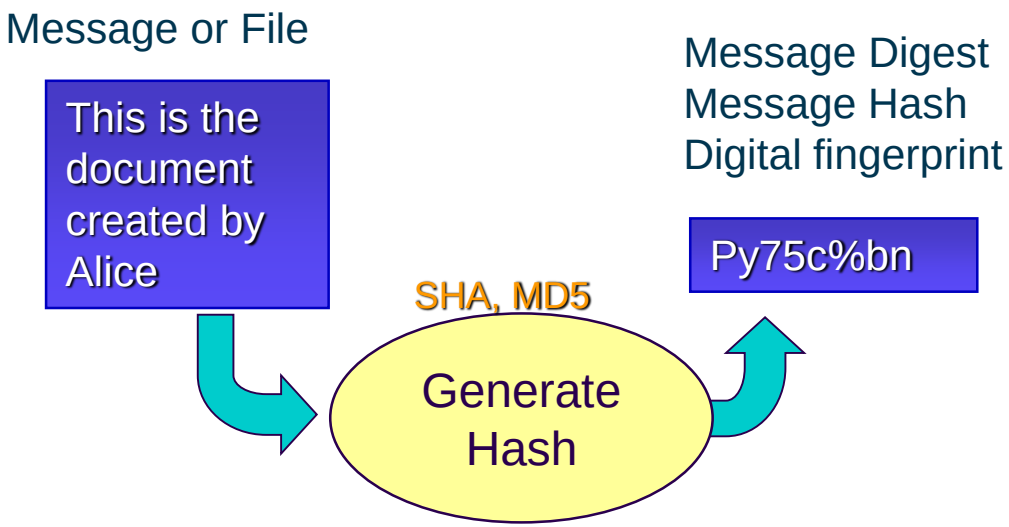
Chunk transfers

- **Among many protocols, Bittorrent is the most popular**
- **An SHA1 hash (160 bit digest) is created for each chunk**
- **All digests are assembled in a "torrent file" with all relevant metadata information**
- **Torrent files are published and registered with a tracker which maintains lists of the clients currently sharing the torrent's chunks**
- **In particular, torrent files have:**
  - an "announce" section, which specifies the URL of the tracker
  - an "info" section, containing (suggested) names for the files, their lengths, the list of SHA-1 digests
- **Reminder: it is the client's duty to reassemble the initial file and therefore it is the client that always verifies the integrity of the data received**

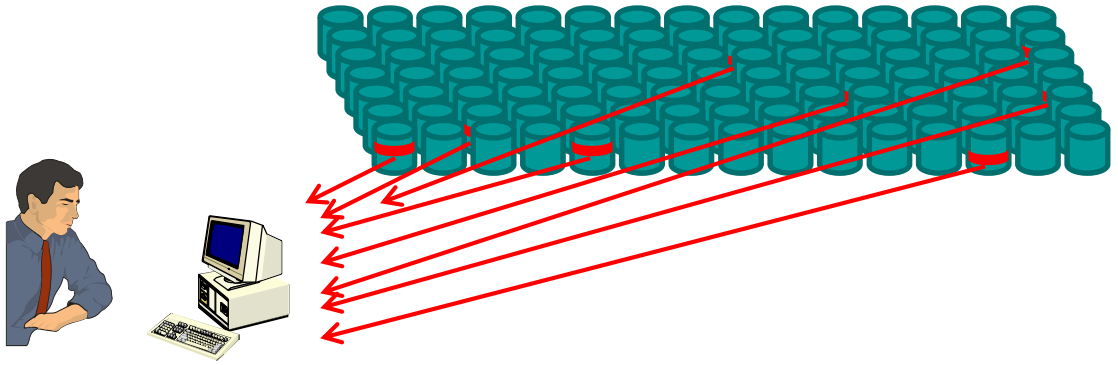
http://en.wikipedia.org/wiki/BitTorrent_(protocol)

Cryptographic Hash Functions

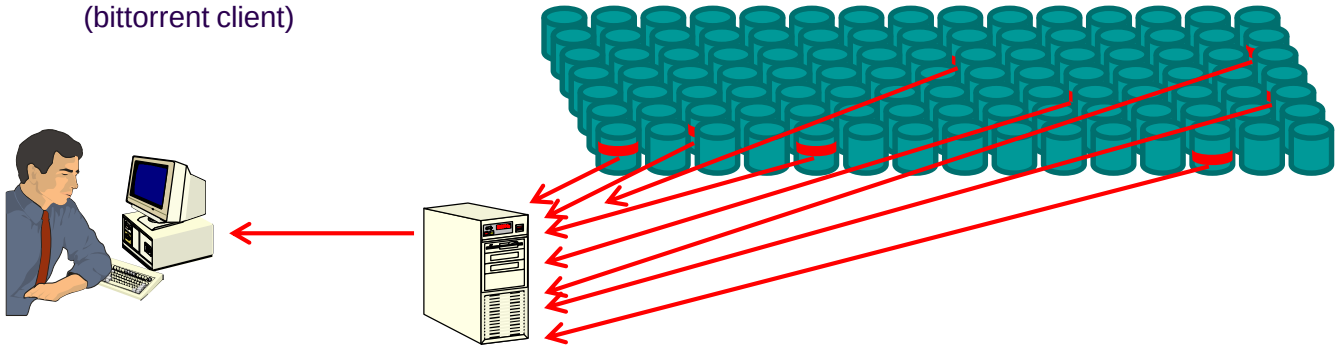
- **A transformation that returns a fixed-size string, which is a short representation of the message from which it was computed**
  - Any (small) modification in the message generates a modification in the digest
- **Should be efficiently computable and impossible to:**
  - find a (previously unseen) message that matches a given digest
  - find "collisions", wherein two different messages have the same message digest



Reassembling the chunks



Data reassembled directly on the client (bittorrent client)

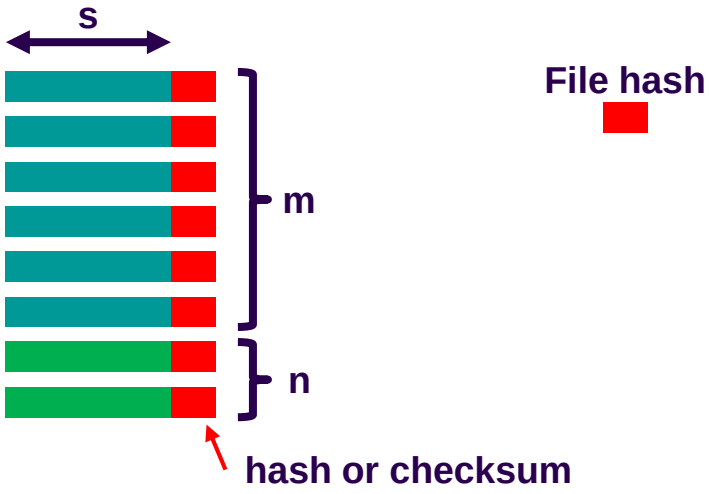


Reassembly done by the data management infrastructure

Middleware

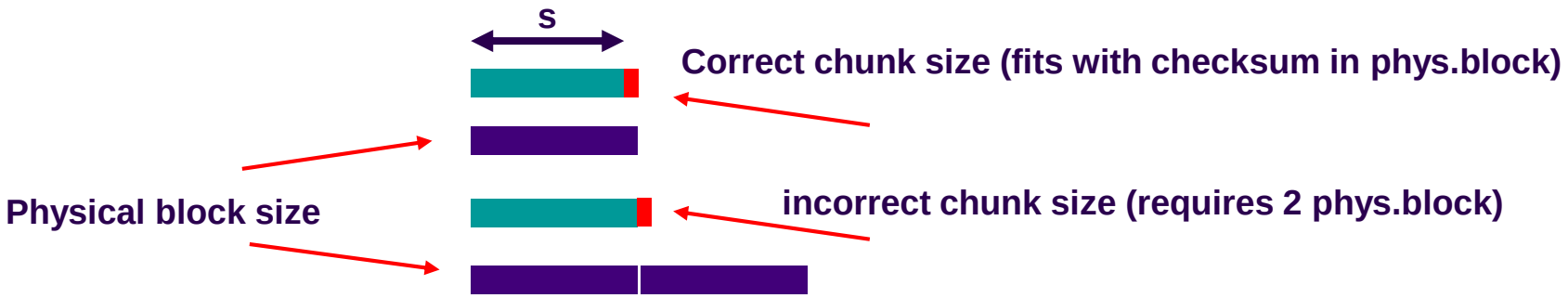
Ensure integrity, identify corruptions

- **You must be able to identify broken files**
  - A hash is required for every file.
- **You must be able to identify broken chunks**
  - A hash for every chunk (example SHA1 160 bit digest) guarantees chunks integrity.
- **It tells you the corrupted chunks and allows you to correct n errors (instead of n-1 if you would not know which chunks are corrupted)**



Chunk size and physical blocks

- The storage overhead of the checksum is typically of few hundred bytes and can be easily neglected compared to the chunk size that is of few megabytes.
- To guarantee a high efficiency in transferring the chunks is essential that the sum of the chunk size with its checksum is an exact multiple or divisor of the physical block size of the storage
- Avoid at all cost is to choose a chunk size equal to the physical disk block size leaving no space to save the checksum in the same physical block.

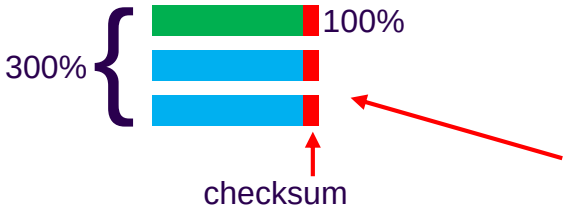


Types of arbitrary reliability (summary)

- **Plain (reliability of the service = reliability of the hardware)**

Types of arbitrary reliability (summary)

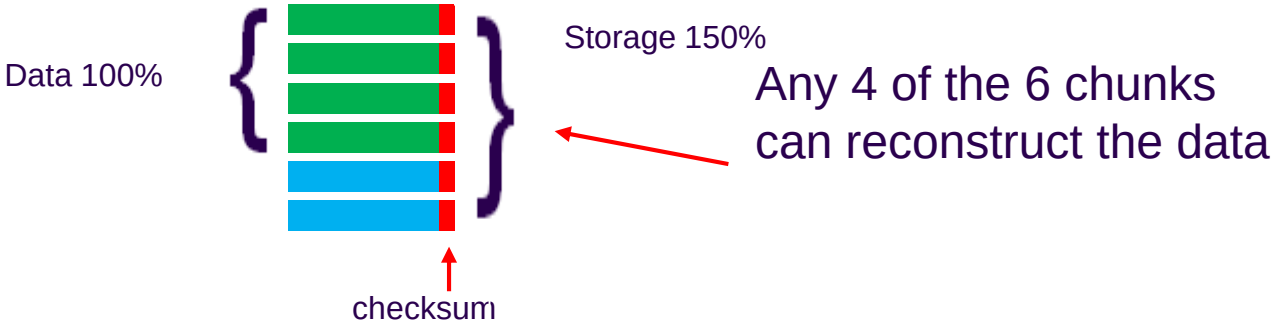
- **Plain (reliability of the service = reliability of the hardware)**
- **Replication**
  - Reliable, maximum performance, but heavy storage overhead
  - Example: 3 copies, 200% overhead



Any of the 3 copies is enough to reconstruct the data

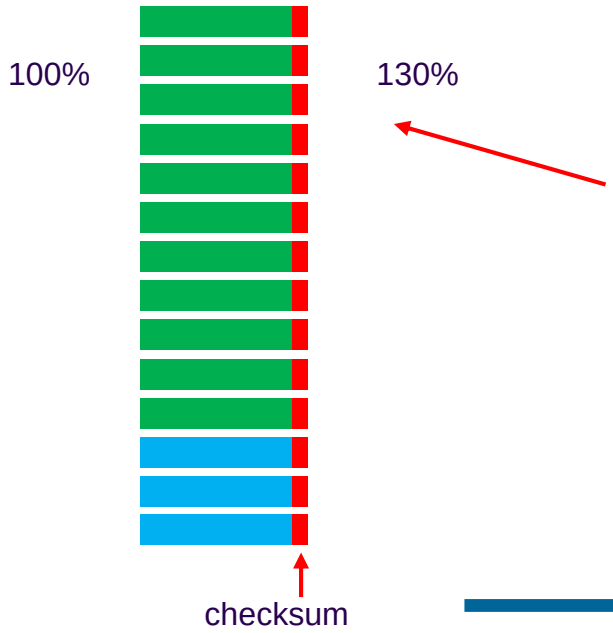
Types of arbitrary reliability (summary)

- **Double parity / Diagonal parity**
  - Example 4+2, can lose any 2, remaining 4 are enough to reconstruct, only 50 % storage overhead



Types of arbitrary reliability (summary)

- **Plain (reliability of the service = reliability of the hardware)**
- **Replication**
  - Reliable, maximum performance, but heavy storage overhead
  - Example: 3 copies, 200% overhead
- **Reed-Solomon, double, triple parity, NetRaid5, NetRaid6**
  - Maximum reliability, minimum storage overhead
  - Example 10+3, can lose any 3, remaining 10 are enough to reconstruct, only 30 % storage overhead



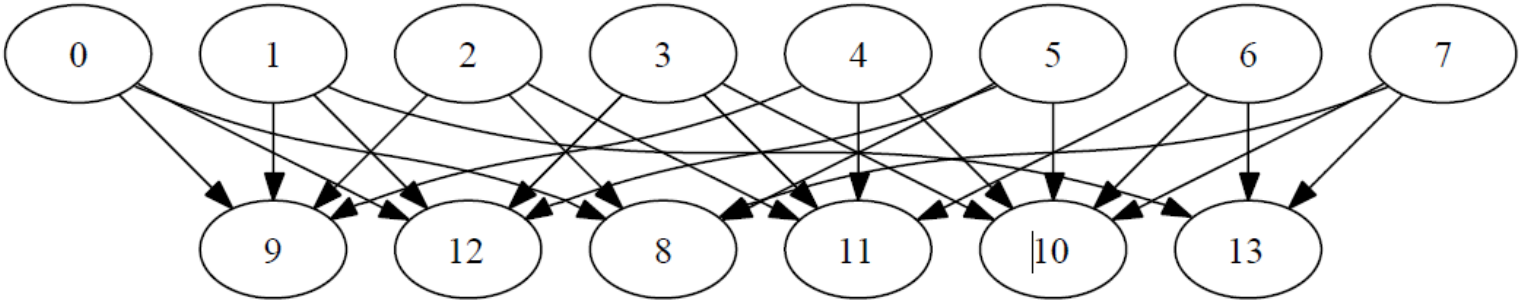
Any 10 of the 13 chunks are enough to reconstruct the data

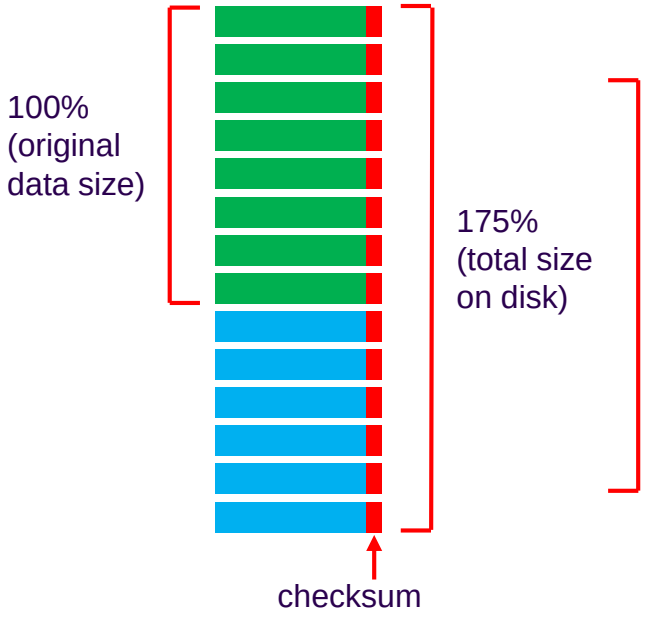
Types of arbitrary reliability (summary)

- **Plain (reliability of the service = reliability of the hardware)**
- **Replication**
  - Reliable, maximum performance, but heavy storage overhead
  - Example: 3 copies, 200% overhead
- **Reed-Solomon, double, triple parity, NetRaid5, NetRaid6**
  - Maximum reliability, minimum storage overhead
  - Example 10+3, can lose any 3, remaining 10 are enough to reconstruct, only 30 % storage overhead
- **Low Density Parity Check (LDPC) / Fountain Codes / Raptor Codes**
  - Excellent performance, more storage overhead
  - Example: 8+6, can lose any 3, remaining 11 are enough to reconstruct, 75 % storage overhead (See next slide)

Example: 8+6 LDPC

0 .. 7: original data
8 .. 13: data xor-ed following the arrows in the graph





138%
(min size required
to reconstruct)

You are allowed to
lose any 3 chunks (21 %)

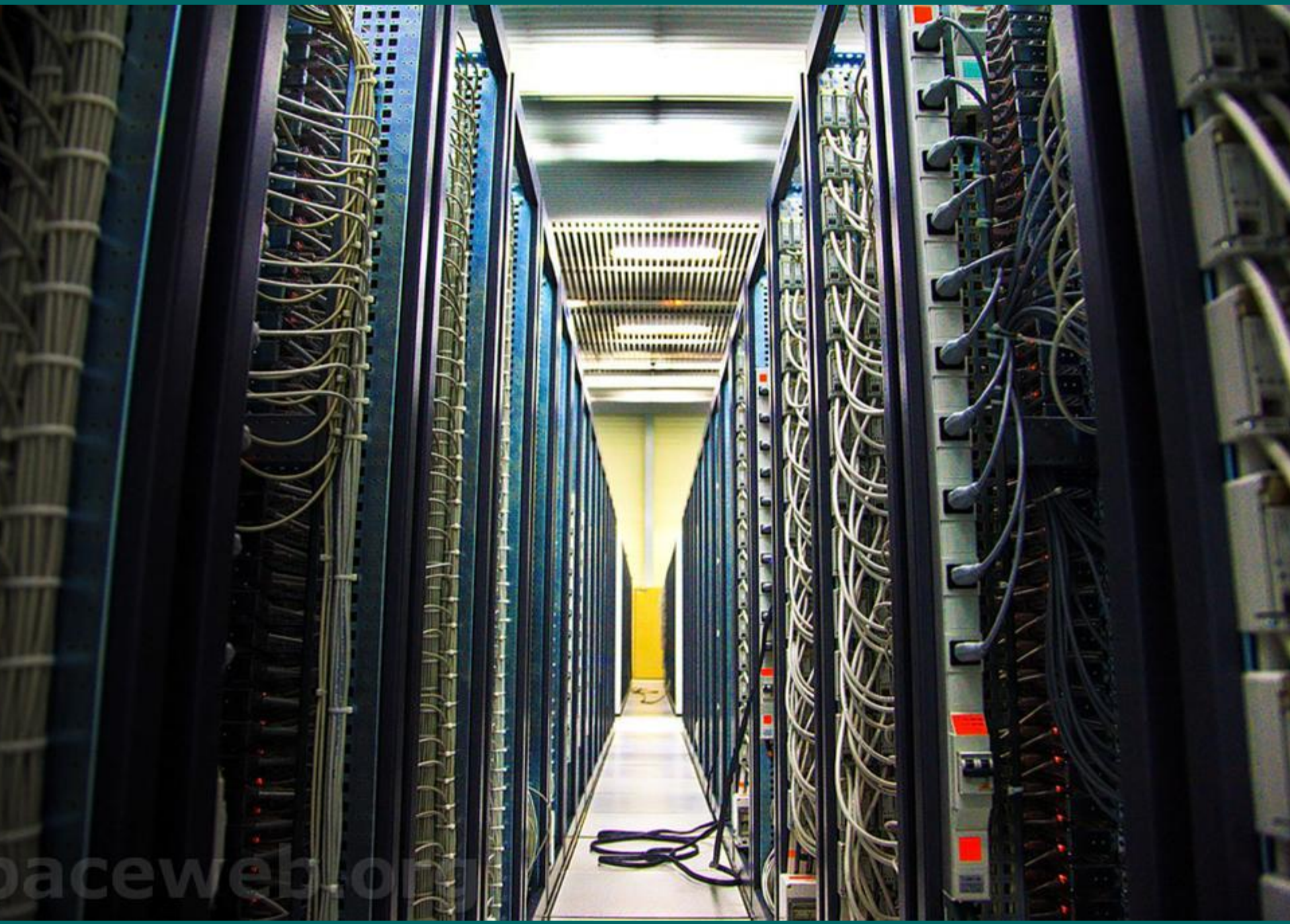
Any 11 of the 14 chunks are enough to reconstruct the data using only XOR operations (very fast)

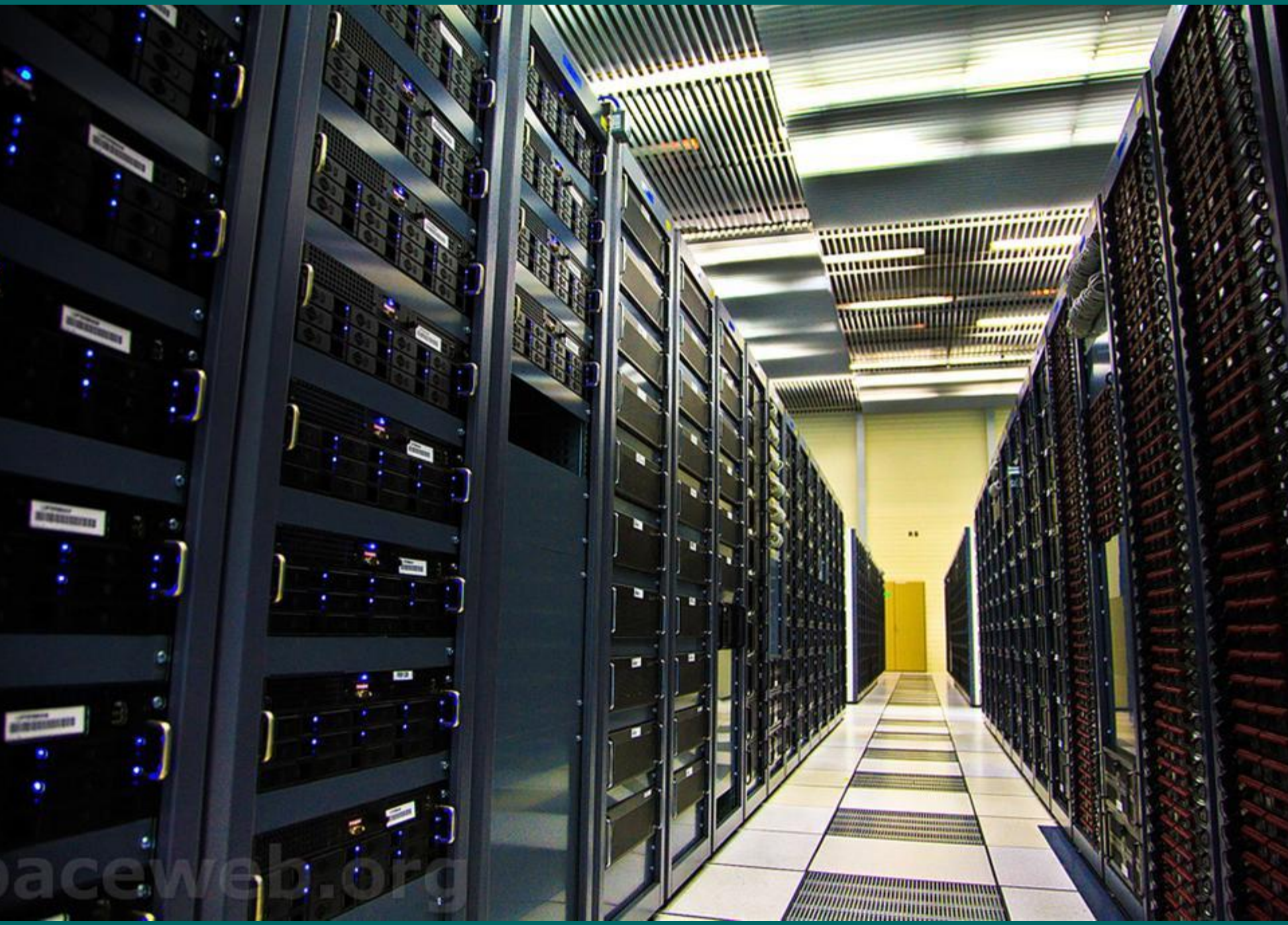
Types of arbitrary reliability (summary)

- **Plain (reliability of the service = reliability of the hardware)**
- **Replication**
  - Reliable, maximum performance, but heavy storage overhead
  - Example: 3 copies, 200% overhead
- **Reed-Solomon, double, triple parity, NetRaid5, NetRaid6**
  - Maximum reliability, minimum storage overhead
  - Example 4+2, can lose any 2, remaining 4 are enough to reconstruct, 50 % storage overhead
  - Example 10+3, can lose any 3, remaining 10 are enough to reconstruct, 30 % storage overhead
- **Low Density Parity Check (LDPC) / Fountain Codes**
  - Excellent performance, more storage overhead
  - Example: 8+6, can lose any 3, remaining 11 are enough to reconstruct, 75 % storage overhead
- **In addition to**
  - File checksums (available today)
  - Block-level checksums (available today)

Virtual visit to the computer Centre

Photo credit to Alexey Tselishchev


spaceweb.org


aceweb.org


StorageTek
StorageTek
StorageTek


aceweb.org

And the antimatter ...

Conclusion

Be Ambitious
Be Brave

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it