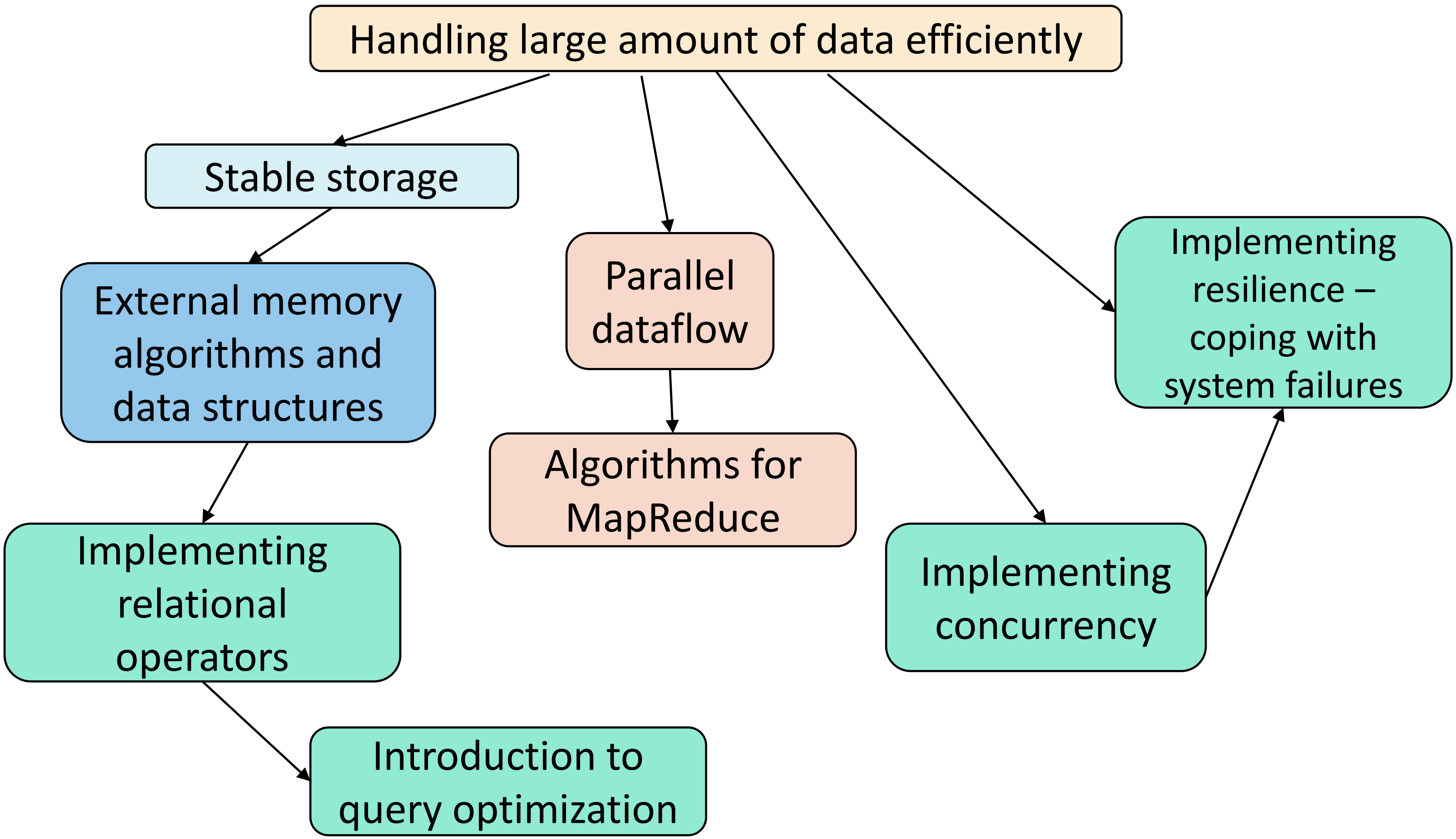
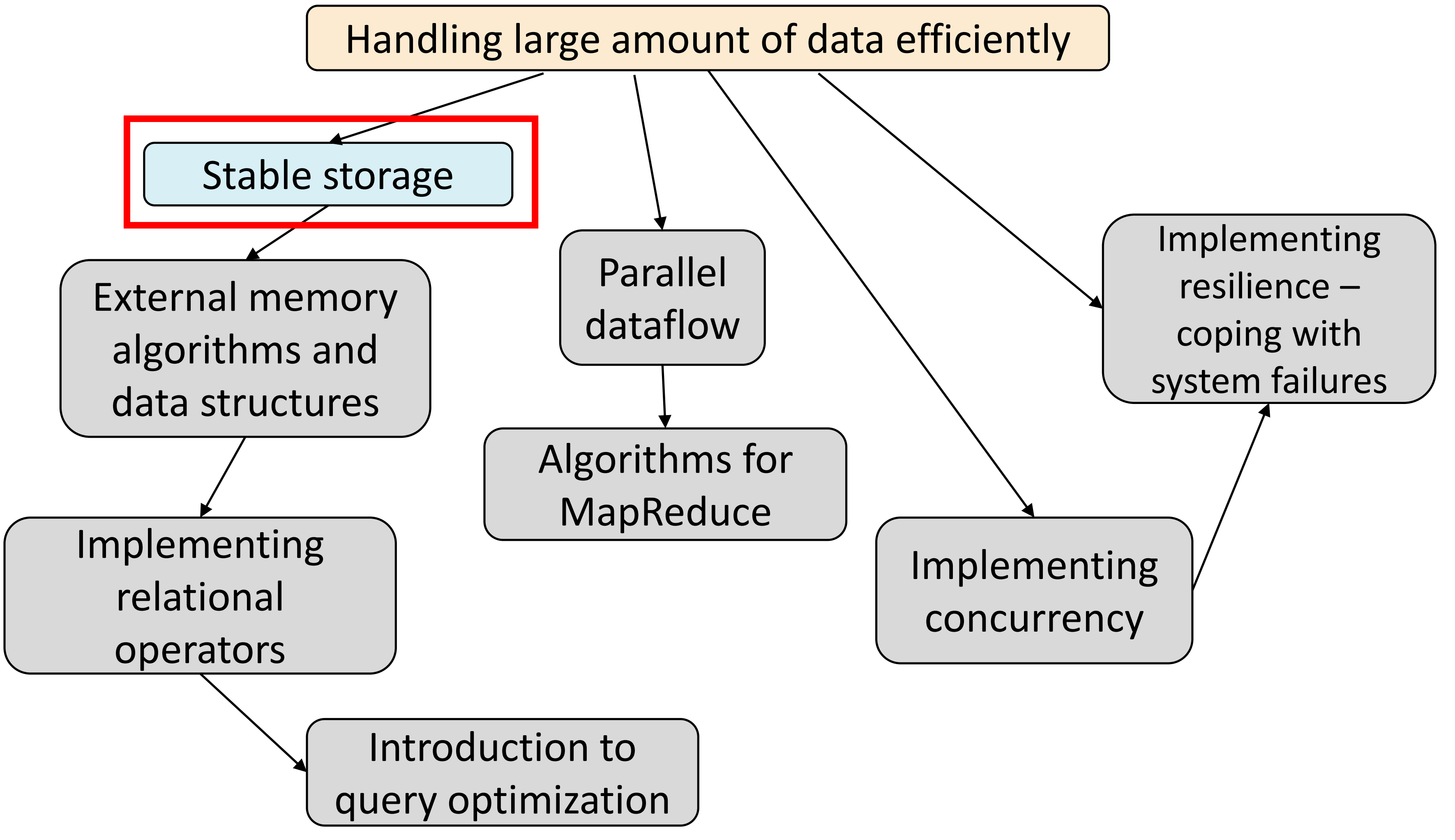
Roadmap


Handling large amount of data efficiently
Stable storage
External memory algorithms and data structures
Implementing relational operators
Introduction to query optimization
Parallel dataflow
Algorithms for MapReduce
Implementing concurrency
Implementing resilience – coping with system failures

Stable storage: how stable?



Lecture 01.02

Coping with disk failures

By Marina Barsky
Winter 2017, University of Toronto

Disks fail in different ways

- *Intermittent failure* – the data transfer failed, but the disk data are not corrupted
- *Disk crash* – the entire disk becomes unreadable, suddenly and permanently

Intermittent Failures

- How do we know that the read/write failed?

- Disk sectors store some redundant bits that can be used to tell us if an I/O operation was successful

- For writes, we simply re-read the sector and check the status bits

Checksums for failure detection

- Status validation is performed with ***checksum***
  - One or more bits that, with high probability, verify the correctness of the operation
  - The checksum is written by the disk controller

Parity bit

- A simple form of checksum is the *parity bit*:
  - Add one bit per sector so that the number of 1's in the sector data + the parity bit is **even**
  - A disk read (per sector) would return status "good" if the bit string has an even number of 1's; otherwise, status = bad

Odd parity – 1bit error

11101110 | 0

Good

11101010 | 0

Bad

If the total sequence of bits, including the parity bit, contains an odd number of 1s – disk controller reports an error

2-bit errors

11101110 | 0

Good

10101010 | 0

Good?

- If more than 1 bit is corrupted, the probability that even parity will be preserved is 50%.

Why?

- For example, if two bits were changed, say, the first erroneous bit was 1 and became 0, the probability that the second erroneous bit was also 1 and become 0 is 50%.

- An error will go undetected in 50% of cases!

Using several parity bits

- Let’s have 8 parity bits – one for each corresponding bit of data bytes

```
01110110
11001101
00001111
10110100
```

Data bytes

8 parity bits

Several parity bits solve the problem

```
01110110
11001101
00001111
10110100
```

Data bytes

8 parity bits

- The probability that a single parity bit will not detect an error is 1/2. The chance that none of 8 bits will detect an error is $\mathbf{1/2^8}$ = **1/256**
- With ***n*** parity bits, the probability of undetected error = $\mathbf{1/2^{\mathit{n}}}$
- If we devote 4 bytes (32 bits) to a checksum of a disk block, the probability of undetected error is ~1/4,000,000,000.

Disk failure types

- Intermittent failure
- Disk crash – the entire disk becomes unreadable, suddenly and permanently

Disk failure and data loss

- *Mean time to failure (MTTF)* = when 50% of the disks have crashed, typically 10 years
- Simplified (assuming this happens linearly) computation
  - In the 1st year = 5% disks fail,
  - In the 2nd year = 5%,
  - …
  - In the 20th year = 5%
- However the mean time to a **disk crash** **doesn't** have to be the same as the mean time to **data loss**; *there are solutions.*

Redundant Array of Independent Disks, RAID

- Mirror each disk (*data disk*/*redundant* disk)
- If data disk fails, restore using the mirror

RAID 1 solution

- Mirror each one data disk with one redundant disk

Assume:

- 5% failure per year; MTTF = 10 years (for disks).
- 3 hours to replace and restore failed disk.

If a failure to one disk occurs, then the other better not fail in the next three hours

- Probability of failure during replacement = 5% $\times$3/(24 $\times$ 365) = 1/58,400.
- If half disks fail every 10 years, then one of two will fail every 5 years
- One in 58,400 of those failures results in data loss; **MTTF = 5*58,400 = 292,000 years**.

RAID 1

- Mirror each data disk with one redundant disk
- Drawback: We need one redundant disk for each data disk.

RAID 4 solution

- ***n*** data disks & **1** redundant disk (for any *n*)

Modulo-2 sum

- We'll refer to the expression ***x*⊕*y*** as **modulo-2 sum** of *x* and *y* **(XOR)**

  E.g. 11110000 ⊕ 10101010 = 01011010

Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Output is 1 when A and B differ

Properties of XOR: $\oplus$

- *Commutativity*: $\mathbf{x \oplus y = y \oplus x}$
- *Associativity*: $\mathbf{x \oplus (y \oplus z) = (x \oplus y) \oplus z}$
- *Identity*: $\mathbf{x \oplus 0 = 0 \oplus x = x}$ **(0 is vector 0000…)**
- *Self-inverse*: $\mathbf{x \oplus x = 0}$

- As a useful consequence, if $x \oplus y = z$, then we can "add" x to both sides and get $y = x \oplus z$
- More generally, if

  $\mathbf{0} = x_1 \oplus \ldots \oplus x_n$

  Then "adding" x_i to both sides, we get:

  $x_i = x_1 \oplus \ldots x_{i-1} \oplus x_{i+1} \oplus \ldots \oplus x_n$

RAID 4 solution

- ***n*** **data** disks & **1** redundant disk (for any *n*)

- Each block in the redundant disk has the **modulo-2 sum** for the corresponding blocks in the other disks.

***i* th Block of Disk 1:**	11110000
***i* th Block of Disk 2:**	10101010
***i* th Block of Disk 3:**	00111000
***i* th Block of red. disk:**	01100010
	00000000

The redundant disk adjusts modulo-2 sum of all corresponding bits to 0

Failure recovery in RAID 4

We must be able to restore whatever disk crashes.

- Just compute the modulo2 sum of corresponding blocks of all the other disks (including redundant)

- Use equation to restore each block of failed disk

$$x_j = x_1 \oplus \ldots x_{j-1} \oplus x_{j+1} \oplus \ldots \oplus x_n \oplus x_{red}$$

RAID 4 recovery example

- Disk 1 crashes – recover it

```
i th Block of Disk 1:        --------
i th Block of Disk 2:        10101010
i th Block of Disk 3:        00111000
i th Block of red. disk:     01100010
                           ____________
                             00000000
```

RAID 4 recovery example

- Recovered disk 1

***i* th Block of Disk 1:**	**11110000**
***i* th Block of Disk 2:**	**10101010**
***i* th Block of Disk 3:**	**00111000**
***i* th Block of red. disk:**	**01100010**
	00000000

RAID 4: reading opportunity

- **Interesting possibility**: If we want to read from disk *i*, but it is busy and all other disks are free, then instead we can read the corresponding blocks from all other disks and modulo2 sum them.

RAID 4: writing challenge

- **Writing**:
  - Write data block
  - Update redundant block

- **Naively**: Read all ***n*** corresponding blocks

  n+1 disk I/O's:

  n-1 blocks read,

  1 data block write,

  1 redundant block write.

- **Better**: How?

RAID 4: writing

- **Better Writing**: To write block *i* of data disk 1 (new value **v**):
  - Read old value of that block **o**.
  - Read the ***i***th block of the redundant disk with value **r**.
  - Compute **w** = **v** $\oplus$ **o** $\oplus$ **r**.
  - Write **v** in block *i* of disk 1.
  - Write **w** in block *i* of the redundant disk.

- Total: 4 disk I/O; (true for any number of data disks)
- **Why does this work?**
  - Intuition: **v** $\oplus$ **o** is the "change" to the overall parity
  - *Redundant disk* must change accordingly to compensate.

RAID 4 writing example

i th Block of Disk1: `11110000`
i th Block of Disk 2: *10101010*
i th Block of Disk 3: `00111000`
i th Block of *red* disk: `01100010`

Suppose we change ***10101010*** into ***01101110***

```
10101010
01101110
01100010
----------------
10100110
```

Re-computing by using all 3 disks:

```
11110000
01101110
00111000
-------------
10100110
```

RAID 5: solves writing bottleneck

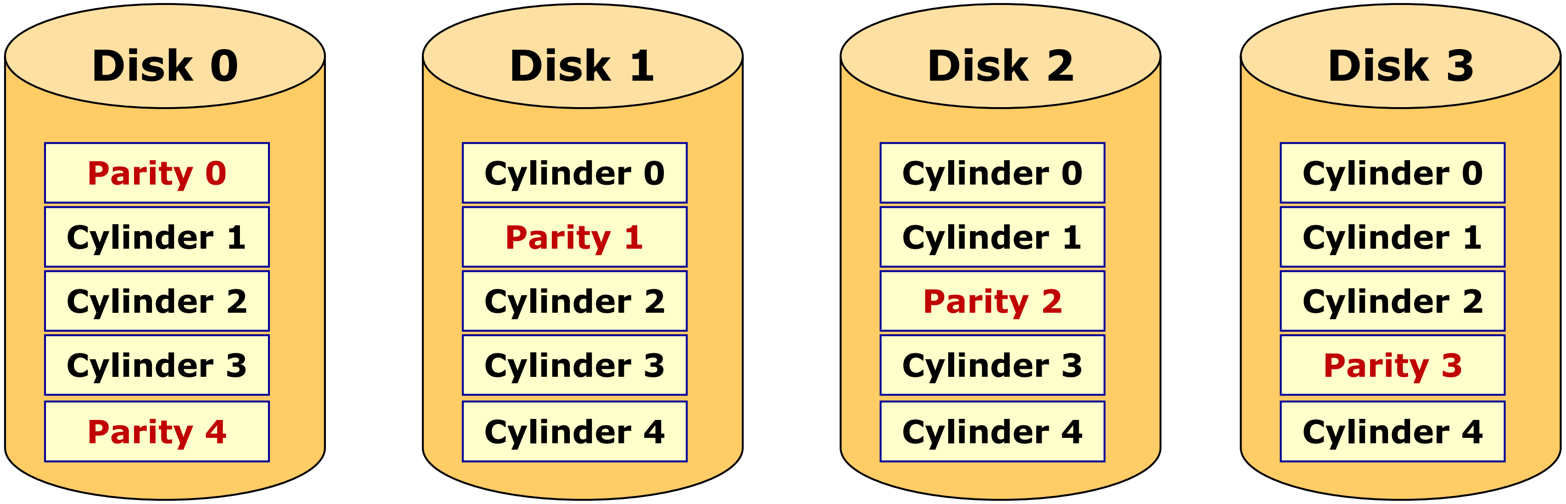
- In RAID 4: the redundant disk is involved in every write → Bottleneck!

- Solution: RAID 5 - vary the redundant disk for different blocks.
  - If we have ***n*** disks, then block ***j*** of disk ***i*** serves as redundant if ***i = j%n***

- In this way, all blocks of each disk are used for data, except some that are used for parity bits of the rest of the disks
- For example, in disk 2 in RAID of 10 disks, the blocks 2, 12, 22 etc. are used for storing parity bits for all the other disks

RAID 5 example

- In practice, not blocks but entire cylinders are used for redundancy
- Example: n=4. So, there are 4 disks.
  - First disk numbered 0, would serve as “redundant” when considering cylinders numbered: 0, 4, 8, 12 etc. (because they leave reminder 0 when divided by 4).
  - Disk numbered 1, would be “redundant” for cylinders numbered: 1, 5, 9, etc.



RAID 6: Coping with multiple disk crashes

- There is a theory of error-correcting codes that allows us to deal with any number of disk crashes – if we use enough redundant disks

- We look how two simultaneous crashes can be recoverable based on the simplest error-correcting code, known as a *Hamming code*

RAID 6 - for multiple disk crashes

- 7 disks, numbered 1 through 7
- The first 4 are data disks, and disks 5 through 7 are redundant.
- The relationship between data and redundant disks is summarized by a 3 x 7 matrix of 0's and 1's

1	***2***	***3***	***4***	***5***	***6***	***7***
1	1	1	0	1	0	0
1	1	0	1	0	1	0
1	0	1	1	0	0	1

5 – first redundant,
6 – second redundant,
7 – third redundant

The 1s in row *i* of data disks tell that the parity for these disks is in a redundant disk *i*

Each data disk has at least 2 associated redundant disks

There are no two equal participation columns for two different data disks

RAID 6 - example

1) **1111**0000
2) **1**0**1**0**1**0**1**0
3) 00**111**000
4) 01000001
5) 0**11**000**1**0
6) 00011011
7) 10001001

disk **5** is modulo 2 sum of disks 1,2,3

disk 6 is modulo 2 sum of disks 1,2,4

disk 7 is modulo 2 sum of disks 1,3,4

1	*2*	*3*	*4*	*5*	*6*	*7*
1	**1**	**1**	0	1	0	0
1	1	0	1	0	1	0
1	0	1	1	0	0	1

RAID 6 - example

1) **1111**0000
2) **1**0**1**0**1**0**1**0
3) 00111000
4) 0**1**00000**1**
5) 01100010
6) 000**11**0**11**
7) 10001001

disk 5 is modulo 2 sum of disks 1,2,3

disk **6** is modulo 2 sum of disks 1,2,4

disk 7 is modulo 2 sum of disks 1,3,4

1	*2*	*3*	*4*	*5*	*6*	*7*
1	1	1	0	1	0	0
1	**1**	0	**1**	0	1	0
1	0	1	1	0	0	1

RAID 6 - example

1) **1111**0000
2) 10101010
3) 00**111**000
4) 0**1**00000**1**
5) 01100010
6) 00011011
7) **1**000**1**00**1**

disk 5 is modulo 2 sum of disks 1,2,3

disk 6 is modulo 2 sum of disks 1,2,4

disk 7 is modulo 2 sum of disks 1,3,4

1	*2*	*3*	*4*	*5*	*6*	*7*
1	1	1	0	1	0	0
1	1	0	1	0	1	0
1	0	1	1	0	0	1

RAID 6 Recovery

1	2	3	4	5	6	7
1	1	1	0	1	0	0
1	1	0	1	0	1	0
1	0	1	1	0	0	1

Why is it possible to recover from two disk crashes?

- Let the failed disks be *a* and *b.*
- Since all columns of the redundancy matrix are different, we must be able to find some row *r* in which the columns for *a* and *b* are different.
  - Suppose that *a* has 0 in row *r,* while *b* has 1 there.
- Then we can compute the correct *b* by taking the modulo-2 sum of corresponding bits from all the disks other than *b* that have 1 in row *r.*
  - Note that *a* is not among these, so none of them have failed.
- Having done so, we can recompute *a,* with all other disks available.

RAID 6 – How many redundant disks?

- The total number of disks can be one less than any power of 2, say $2^k - 1$.

- Of these disks, k are redundant, and the remaining $2^k - 1 - k$ are data disks, so the redundancy grows roughly as the logarithm of the number of data disks.

- For any *k,* we can construct the redundancy matrix by writing all possible columns of k 0's and 1's, except the all-0's column.
  - The columns with a single 1 correspond to the redundant disks, and the columns with more than one 1 are the data disks.

Note finally that we can combine RAID 6 with RAID 5 to reduce the performance bottleneck on the redundant disks

Exercises

RAID 4

***i* th Block of Disk 1:** **11110000**

***i* th Block of Disk 2:** **10101010**

***i* th Block of Disk 3:** **00111000**

***i* th Block of Disk 3:** **11111011**

***i* th Block of red. disk:**

RAID 4

i th Block of Disk 1: 11110000

i th Block of Disk 2: 10101010

i th Block of Disk 3: 00111000

i th Block of Disk 3: 11111011

i th Block of red. disk: 10011001

RAID 4

i **th Block of Disk 1:** --------

i **th Block of Disk 2:** 10101010

i **th Block of Disk 3:** 00111000

i **th Block of Disk 3:** 11111011

i **th Block of red. disk:** 10011001

Now suppose that Disk 1 crashed. Recover it.

RAID 4

i **th Block of Disk 1:** 11110000

i **th Block of Disk 2:** 10101010

i **th Block of Disk 3:** 00111000

i **th Block of Disk 3:** 11111011

i **th Block of red. disk:** 10011001

Now suppose that Disk 1 crashed. Recover it.

RAID 5

Disk 1: 1111000001

Disk 2: 1010101011

Disk 3: 0011100000

Disk 4: 1111101101

Disk 5: 1001100111

The red bits are used for redundancy

(This is toy example. In practice we talk in terms of cylinders)

RAID 5

Disk 1: ----------

Disk 2: 1010101011

Disk 3: 0011100000

Disk 4: 1111101101

Disk 5: 1001100111

The red bits are used for redundancy

(This is toy example. In practice we talk in terms of cylinders)

Now suppose that Disk 1 crashed. Recover it.

RAID 5

Disk 1: --11000001

Disk 2: 1010101011

Disk 3: 0011100000

Disk 4: 1111101101

Disk 5: 1001100111

The red bits are used for redundancy

(This is toy example. In practice we talk in terms of cylinders)

Now suppose that Disk 1 crashed. Recover it.

RAID 5

Disk 1: 1111000001

Disk 2: 1010101011

Disk 3: 0011100000

Disk 4: 1111101101

Disk 5: 1001100111

The red bits are used for redundancy

(This is toy example. In practice we talk in terms of cylinders)

Now suppose that Disk 1 crashed. Recover it.

RAID 6

1) 11110000
2) 10101010
3) 00111000
4) 01000001
5) 01100010
6) 00011011
7) 10001001

1	2	3	4	5	6	7
1	1	1	0	1	0	0
1	1	0	1	0	1	0
1	0	1	1	0	0	1

RAID 6

1) 11110000

2) --------

3) 00111000

4) 01000001

5) --------

6) 00011011

7) 10001001

1	2	3	4	5	6	7
1	1	1	0	1	0	0
1	1	0	1	0	1	0
1	0	1	1	0	0	1

Now suppose that Disk 2 and Disk 5 crash. Recover them.

RAID 6

1) **1111**0000
2) **10101010**
3) 00111000
4) 0**1**00000**1**
5) --------
6) 000**11**0**11**
7) 10001001

1	2	3	4	5	6	7
1	1	1	0	1	0	0
1	1	0	1	0	1	0
1	0	1	1	0	0	1

Now suppose that Disk 2 and Disk 5 crash. Recover them.

We find the row with 1 for disk 2 and 0 for disk 5
We can recover disk 2 using redundant disk 6 which is the parity for disks 1,2,4

RAID 6

1) **1111**0000

2) **1**0**1**0**1**0**1**0

3) 00**111**000

4) 01000001

5) **00100010**

6) 00011011

7) 10001001

1	2	3	4	5	6	7
1	1	1	0	1	0	0
1	1	0	1	0	1	0
1	0	1	1	0	0	1

Now suppose that Disk 2 and Disk 5 crash. Recover them.

We know that disk 5 is a parity disk for data disks 1,2,3. All their values are known, so we recover disk 5

RAID 6

1) 11110000

2) --------

3) 00111000

4) --------

5) 01100010

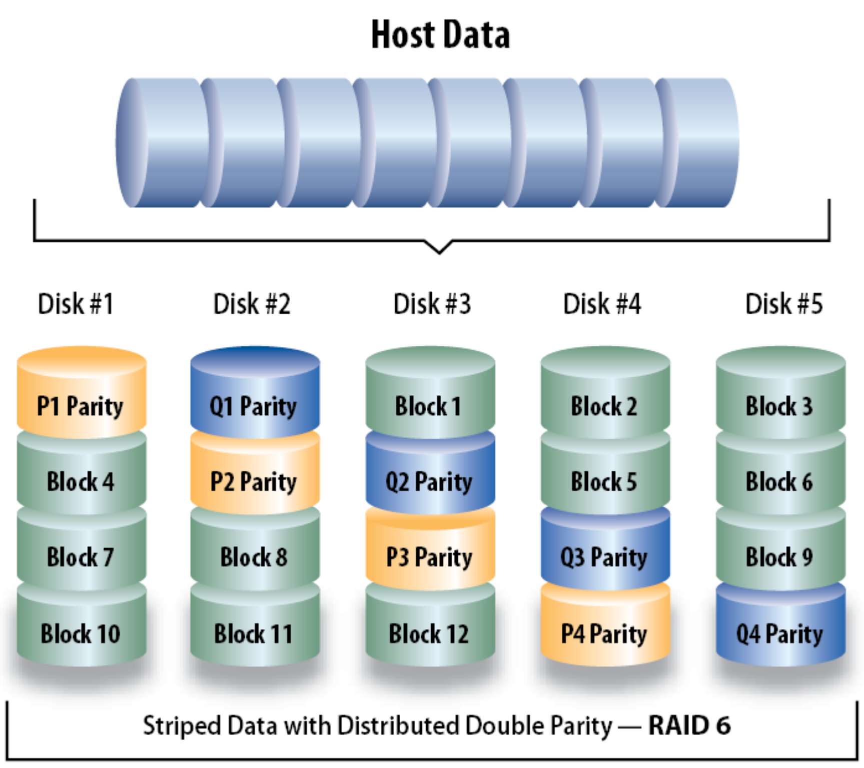
6) 00011011

7) 10001001

1	2	3	4	5	6	7
1	1	1	0	1	0	0
1	1	0	1	0	1	0
1	0	1	1	0	0	1

Now suppose that Disk 2 and Disk 4 crash. Recover them.

Another Version of RAID 6



- RAID 6 based on Reed-Solomon codes (1997).
- The damage protection method can be briefly explained via these two mathematical expressions:

P = D1 + D2 + D3 + D4

Q = 1*D1 + 2*D2 + 3*D3 + 4*D4

- If any two of P, Q, D1, D2, D3 and D4 become unknown (or lost), then solve the system of equations for 2 unknowns.
- In fact, we don't really multiply by 1,2,3,4 but by *g*, *g*^2, *g*^3, *g*^4, where *g* is a Galois field generator.