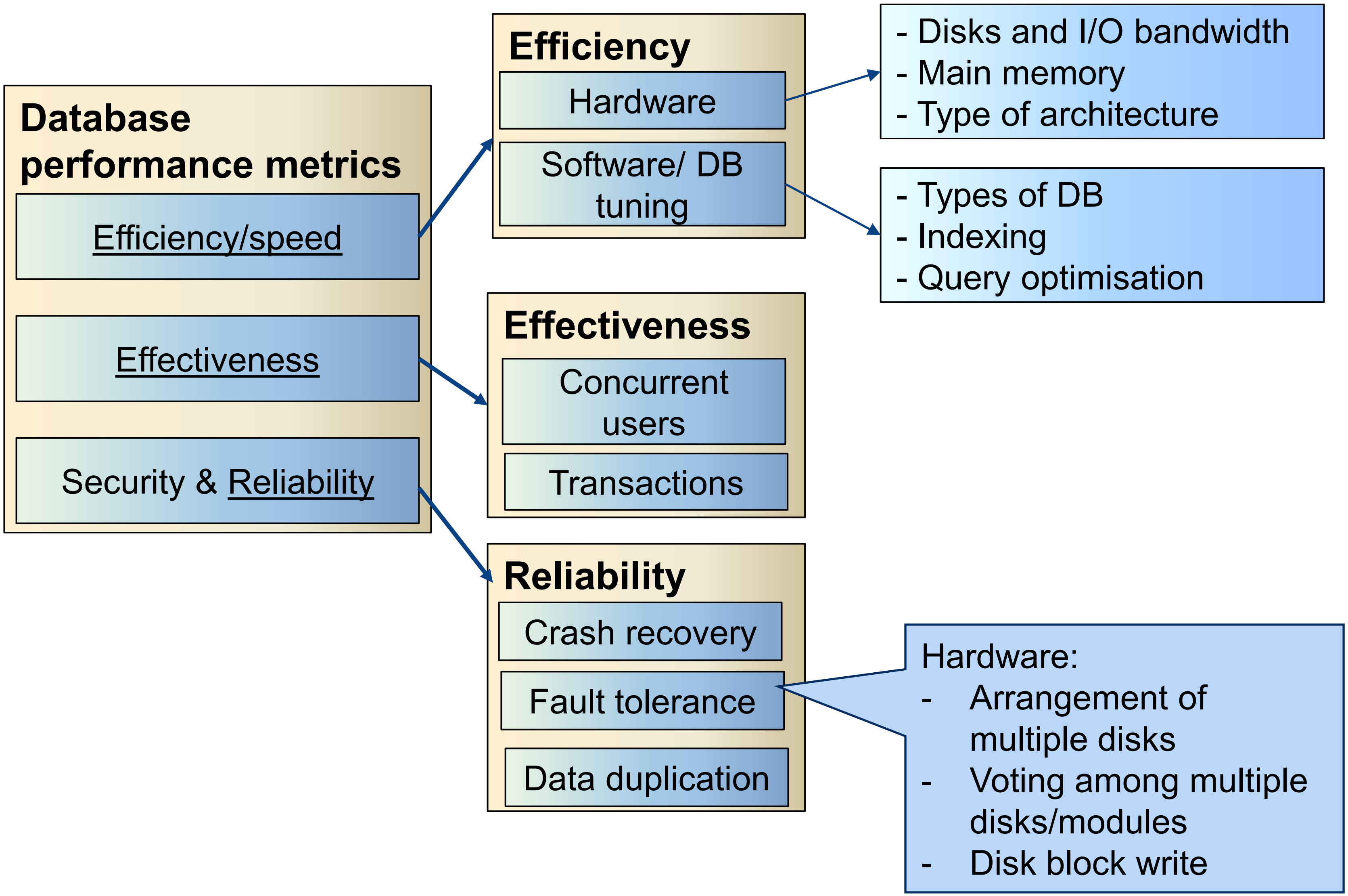
COMP90050
Advanced Database Systems

Winter Semester, 2023

Lecturer: Farhana Choudhury (PhD)

Week 1 part 4

Core Concepts of Database management system



Disk writes for consistency:

Either entire block is written **correctly** on disk or the contents of the block is unchanged. To achieve disk write consistency we can do –

- ***Duplex*** write
- **Logged** write

Transaction models…

Disk writes for consistency:

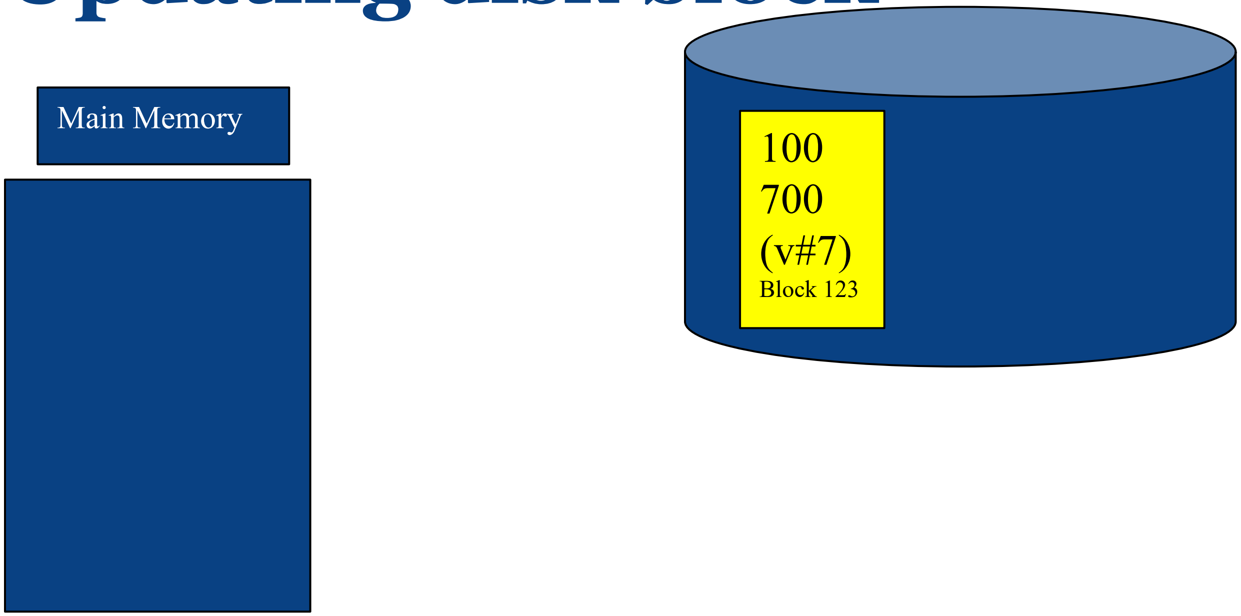
Either entire block is written **correctly** on disk or the contents of the block is unchanged. To achieve disk write consistency we can do –

Duplex write:

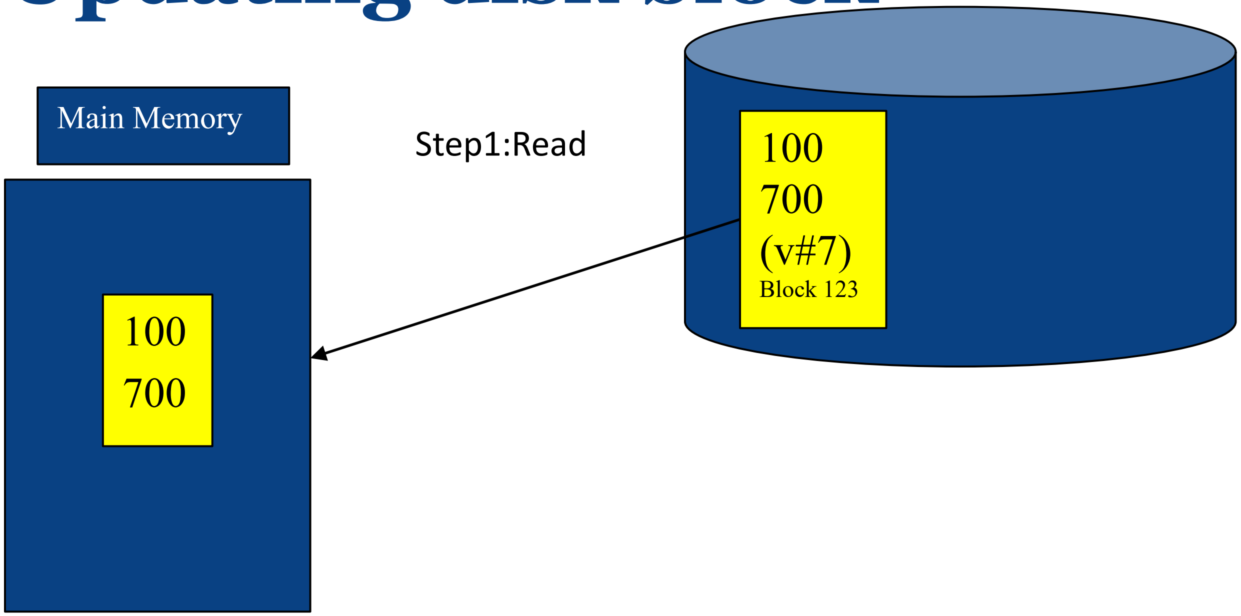
- Each block of data is written in two places *sequentially*
- If one of the writes fail, system can issue another write
- Each block is associated with a version number. The block with the latest version number contains the most recent data.
- While reading - we can determine error of a disk block by its ***CRC***.
- It always guarantees at least one block has consistent data.

Logged write- similar to duplex write, except one of the writes goes to a log. This method is very efficient if the changes to a block are small. We will discuss an efficient method later in the subject.

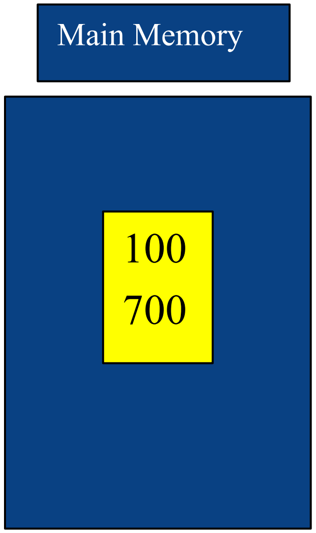
Updating disk block



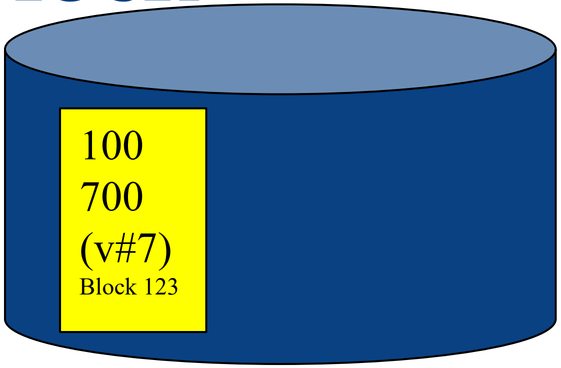
Updating disk block



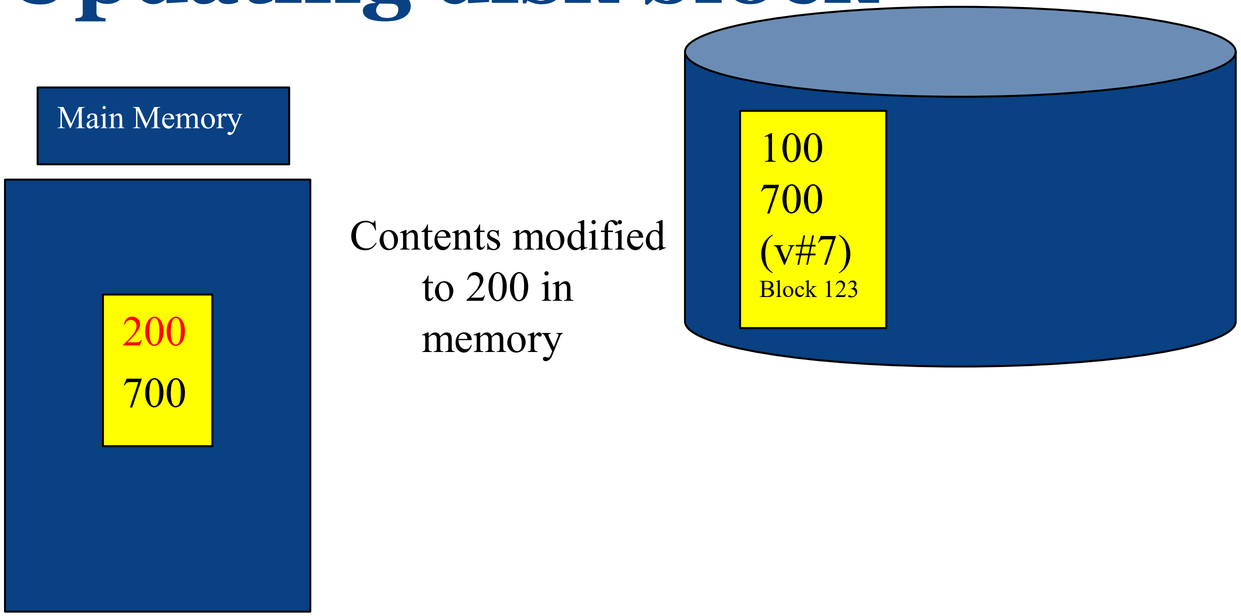
Updating disk block



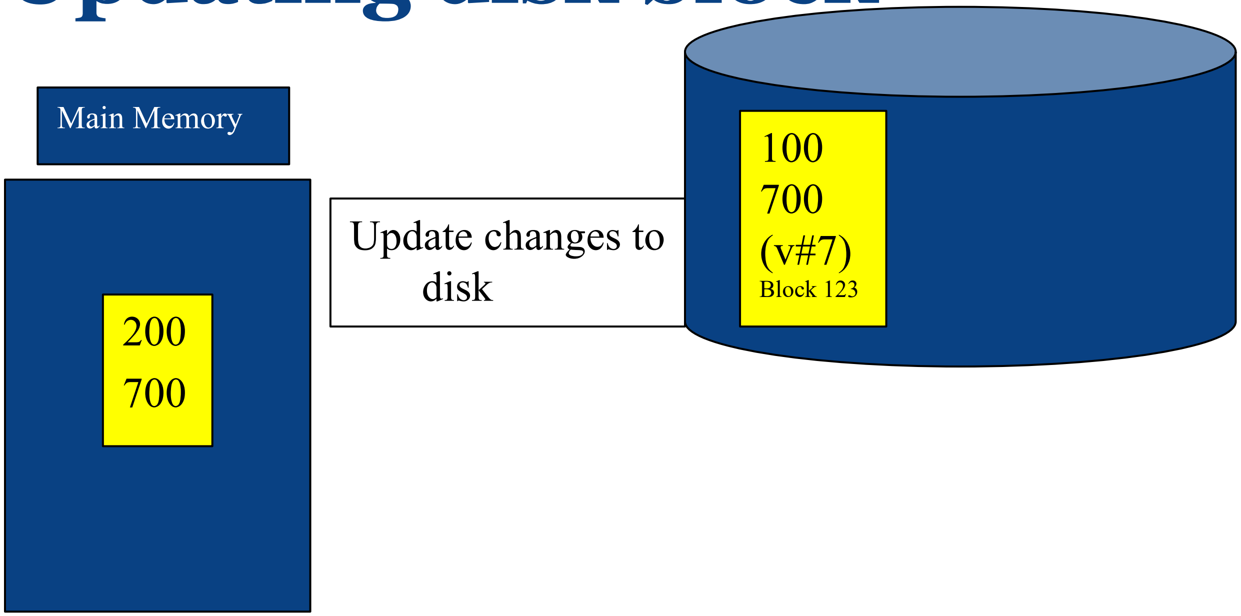
Step2:
Modify contents in memory to say 200



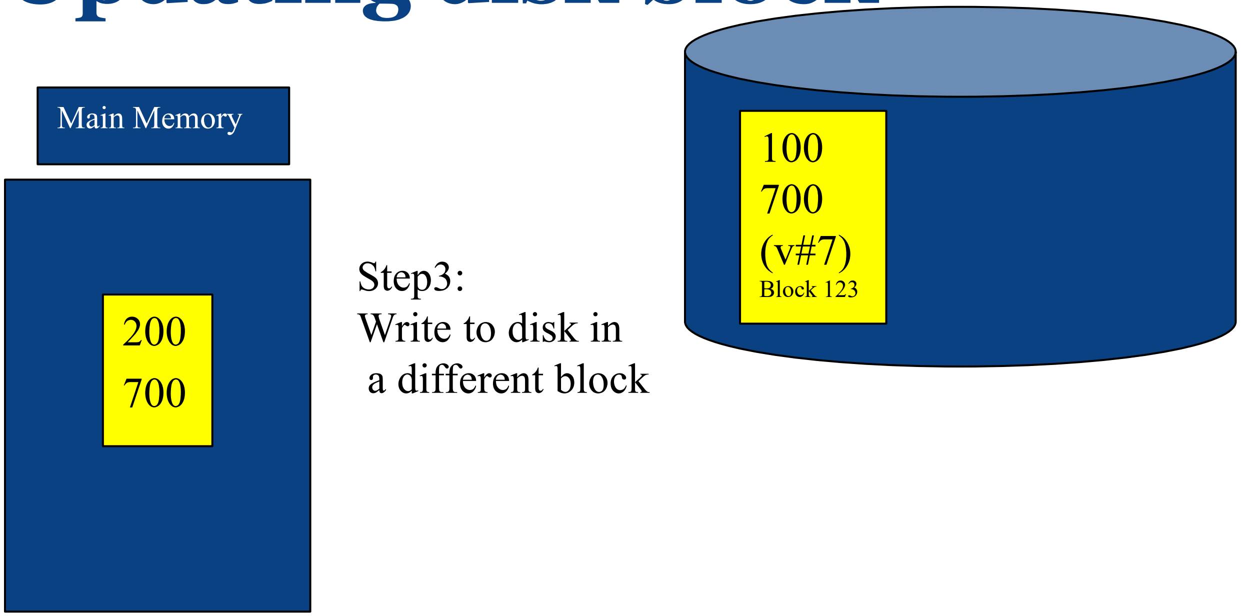
Updating disk block



Updating disk block

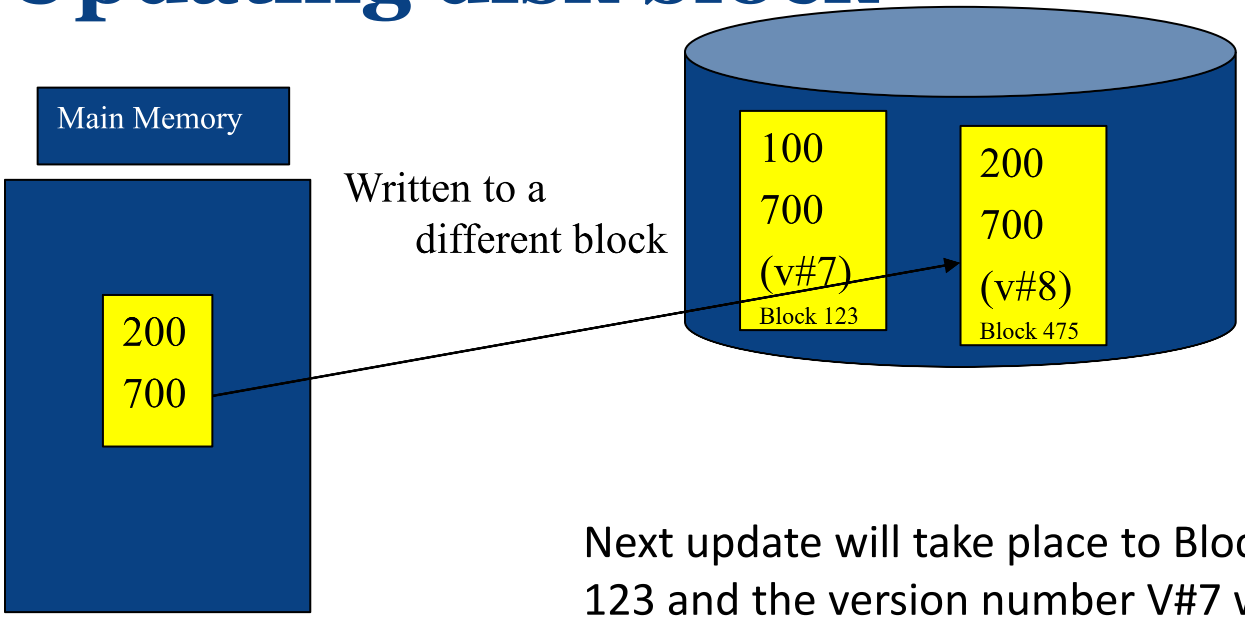


Updating disk block



Step3:
Write to disk in
a different block

Updating disk block



Next update will take place to Block 123 and the version number V#7 will be changed to v#9.

(Two different physical disks can be used for duplex writes as well)

Transaction models…

Disk writes for consistency

Either entire block is written **correctly** on disk or the contents of the block is unchanged. To achieve atomic disk writes we can do –

Duplex write:

- Each block of data is written in two places *sequentially*
- If one of the writes fail, system can issue another write
- Each block is associated with a version number. The block with the latest version number contains the most recent data.
- While reading - we can determine error of a disk block by its ***CRC***.
- It always guarantees at least one block has consistent data.

Cyclic Redundancy Check (CRC) generation

CRC polynomial $x^{32} + x^{23} + x^{7} + 1$

Most errors in communications or on disk happen contiguously, that is in burst in nature. The above CRC generator can detect all burst errors with a length less than or equal to 32 bits; 5 out of 10 billion burst errors with length 33 will be undetected; 3 out of 10 billion burst errors of length 34 or more will be undetected.

Example CRC polynomials

$x^{5} + x^{3} + 1$

$x^{15} + x^{14} + x^{11} + x^{10} + x^{8} + x^{7} + x^{4} + x^{3} + 1$

Cyclic Redundancy Check (CRC) generation

To compute an *n*-bit binary CRC:

1. Add n zero bits as ‘padding’ to the right of the input bits.

Input: 11010011101100

This is first padded with zeros corresponding to the bit length n of the CRC:

11010011101100 000 <--- input left shifted by 3 bits of padding

2. Compute the (n + 1)-bit pattern representing the CRC's divisor (called a "polynomial")

In the following example, we shall encode 14 bits of message with a 3-bit CRC, with a polynomial $x^3 + x + 1$*. The polynomial is written in binary as the coefficients; a 3rd-degree polynomial has 4 coefficients (*$1x^3 + 0x^2 + 1x + 1$*). In this case, the coefficients are 1, 0, 1 and 1.*

3. Position the (n + 1)-bit pattern representing the CRC's divisor underneath the left-hand end of the input bits.

11010011101100 000 <--- input right padded by 3 bits
1011 <--- divisor (4 bits) = $x^3 + x + 1$

4. The algorithm acts on the bits directly above the divisor in each step.

- *The result for each iteration is the bitwise* ***XOR*** *of the polynomial divisor with the bits above it.*

-*The bits not above the divisor are simply copied directly below for that step.*

-*The divisor is then shifted one bit to the right (or moves over to align with the next 1 in the dividend), and the process is repeated until the bits of the input message becomes zero. Here is the entire calculation:*

Cyclic Redundancy Check (CRC) generation

1011 = $x^3 + x + 1$

11010011101100 000 <--- input left shifted by 3 bits
1011 <--- divisor
01100011101100 000 <--- result
1011 <--- divisor ...
00111011101100 000
1011
00010111101100 000
1011
00000001101100 000
1011
00000000110100 000
1011
00000000011000 000
1011
00000000001110 000
1011
00000000000101 000
101 1

00000000000000 100 <---remainder (3 bits)

moves over to align with the next 1 in the dividend

(Division algorithm stops here as dividend is equal to zero. The remainder 100 will be the value of the CRC function

Checking validity with CRC

The validity of a received message can easily be verified by performing the above calculation again, this time with the check value added instead of zeroes. The remainder should equal zero if there are no detectable errors.

```
11010011101100 100 <--- input with CRC
1011               <--- divisor
01100011101100 100 <--- result
 1011              <--- divisor ...
00111011101100 100
......
00000000001110 100
          1011
00000000000101 100
           101 1
------------------
            0 <--- remainder
```