Now let's think a bit about what happens if there's an error and one or more of the bits in our encoded data gets corrupted.

We'll focus on single-bit errors, but much of what we discuss can be generalized to multi-bit errors.

For example, consider encoding the results of some unpredictable event, e.g., flipping a fair coin.

There are two outcomes: "heads", encoded as, say, 0, and "tails" encoded as 1.

Now suppose some error occurs during processing - for example, the data is corrupted while being transmitted from Bob to Alice: Bob intended to send the message "heads", but the 0 was corrupted and become a 1 during transmission, so Alice receives 1, which she interprets as "tails".

So this simple encoding doesn't work very well if there's the possibility of single-bit errors.

To help with our discussion, we'll introduce the notion of "Hamming distance", defined as the number of positions in which the corresponding digits differ in two encodings of the same length.

For example, here are two 7-bit encodings, which differ in their third and fifth positions, so the Hamming distance between the encodings is 2.

If someone tells us the Hamming distance of two encodings is 0, then the two encodings are identical.

Hamming distance is a handy tool for measuring how encodings differ.

How does this help us think about single-bit errors?

A single-bit error changes exactly one of the bits of an encoding, so the Hamming distance between a valid binary code word and the same code word with a single-bit error is 1.

The difficulty with our simple encoding is that the two valid code words ("0" and "1") also have a Hamming distance of 1.

So a single-bit error changes one valid code word into another valid code word.

We'll show this graphically, using an arrow to indicate that two encodings differ by a single bit, i.e., that the Hamming distance between the encodings is 1.

The real issue here is that when Alice receives a 1, she can't distinguish between an uncorrupted encoding of tails and a corrupted encoding of heads - she can't detect that an error occurred.

Let's figure how to solve her problem!

The insight is to come up with a set of valid code words such that a single-bit error does NOT produce another valid code word.

What we need are code words that differ by at least two bits, i.e., we want the minimum Hamming distance between any two code words to be at least 2.

If we have a set of code words where the minimum Hamming distance is 1, we can generate the set we want by adding a parity bit to each of the original code words.

There's "even parity" and "odd parity." Using even parity, the additional parity bit is chosen so that the total number of 1 bits in the new code word are even.

For example, our original encoding for "heads" was 0, adding an even parity bit gives us 00.

Adding an even parity bit to our original encoding for "tails" gives us 11.

The minimum Hamming distance between code words has increased from 1 to 2.

How does this help?

Consider what happens when there's a single-bit error: 00 would be corrupted to 01 or 10, neither of which is a valid code word.

Aha!

We can detect that a single-bit error has occurred.

Similarly, single-bit errors for 11 would also be detected.

Note that the valid code words 00 and 11 both have an even number of 1-bits, but that the corrupted code words 01 or 10 have an odd number of 1-bits.

We say that corrupted code words have a "parity error".

It's easy to perform a parity check: simply count the number of 1s in the code word.

If it's even, a single-bit error has NOT occurred; if it's odd, a single-bit error HAS occurred.

We'll see in a couple of chapters that the Boolean function exclusive-or can be used to perform parity checks.

Note that parity won't help us if there's an even number of bit errors, where a corrupted code word would have an even number of 1-bits and hence appear to be okay.

Parity is useful for detecting single-bit errors; we'll need a more sophisticated encoding to detect more errors.

In general, to detect some number E of errors, we need a minimum Hamming distance of E+1 between code words.

We can see this graphically below which shows how errors can corrupt the valid code words 000 and 111, which have a Hamming distance of 3.

In theory this means we should be able to detect up to 2-bit errors.

Each arrow represents a single-bit error and we can see from the diagram that following any path of length 2 from either 000 or 111 doesn't get us to the other valid code word.

In other words, assuming we start with either 000 or 111, we can detect the occurrence of up to 2 errors.

Basically our error detection scheme relies on choosing code words far enough apart, as measured by Hamming distance, so that E errors can't corrupt one valid code word so that it looks like another valid code word.