# Computer Systems and Professional Practice

Professor Matthew Leeke
School of Computer Science
University of Birmingham

Topic 4 - Memory Systems

# Session Outline

The memory hierarchy, including motivation and structure

Organising memory cells for addressing

Detecting and correcting errors in unreliable memory and network communications

The design and implementation of common memory components, including hard disks and optical discs

# The Memory Hierarchy

# The Choice of Memory Technology

To construct a memory system, we must think carefully about the properties of each storage approach

Only condition is that a memory system must facilitate data modification (writing) and data detection (reading)

Many factors influence the choice of memory technology

- Frequency of access
- Access time
- Capacity required
- Cost, e.g., cost per bit

# The Designer’s Dilemma for Memory Systems

The designer’s dilemma

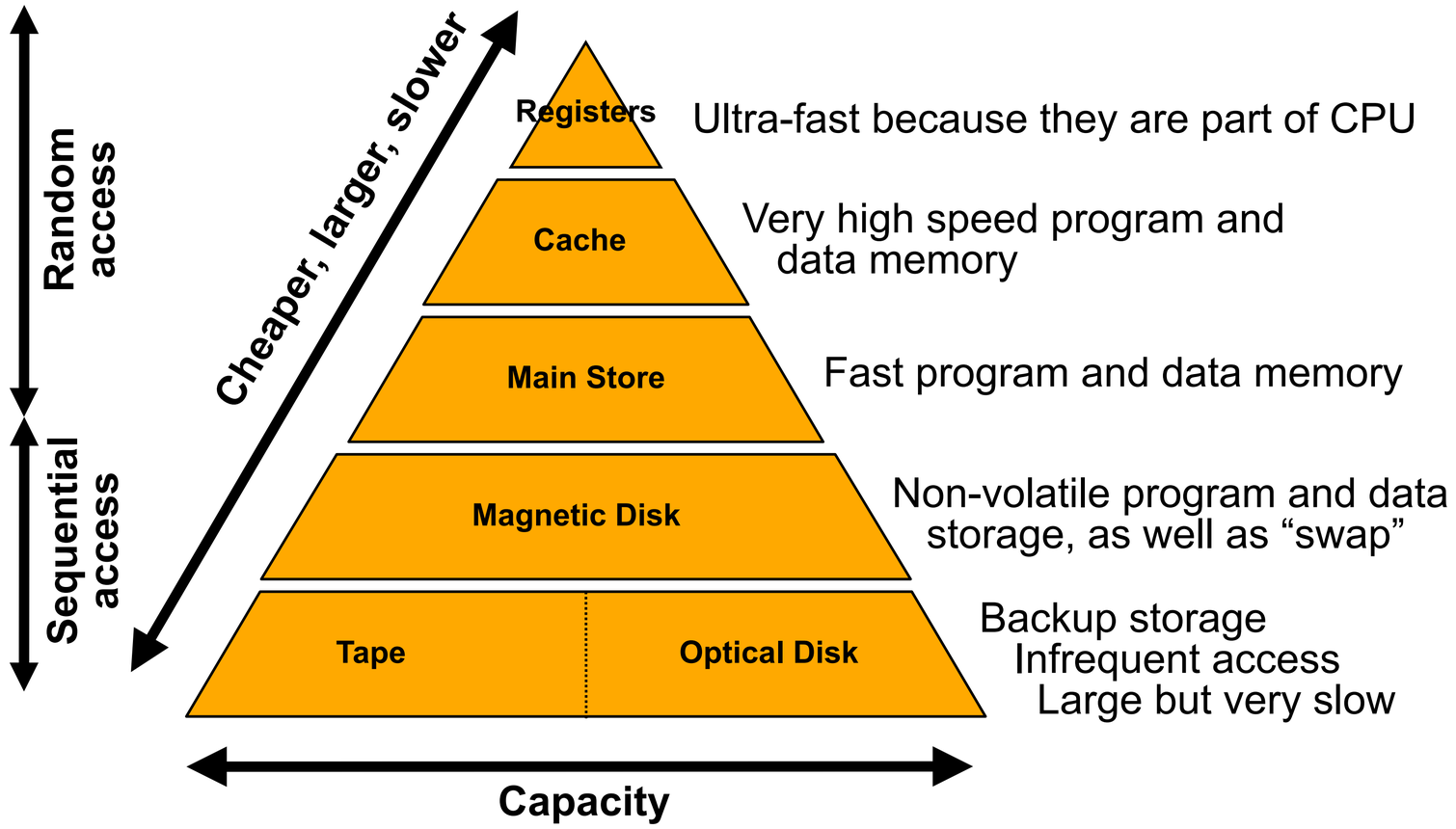
Desire low cost and high capacity - leads to choice of cheap per bit, high capacity and slow access time

Desire high performance - leads to choice of expensive per bit, low capacity and fast access time

**To solve the problem we choose both by organising memory into a hierarchy**

# The Memory Hierarchy

A general view of the memory system that underpin modern computer systems



# Why Have a Memory Hierarchy?

Economics!

Small sequential loops, subroutines, operations on arrays, tables and other structures mean that the locations in memory which will be accessed is somewhat predictable

Temporal locality - If a particular memory location is referenced, it is likely that the same location will be referenced again in the near future

Spatial locality - If a particular memory location is referenced, it is likely that nearby memory locations will be referenced in the near future

Reasonable assumption - It has been shown that 90% of memory accesses are within ± 2 kilobytes of previous PC position

# Semiconductor Memory Types

Particularly interested in memory that provides random access

RAM is most common type of semiconductor memory, though the term is misleading since all types shown above allow random access

| Memory Type | Category | Erasure | Write Mechanism | Volatility |
|---|---|---|---|---|
| Random Access Memory (RAM) | Read-write | Electrically at byte-level | Electrically written | Volatile |
| Read-only Memory (ROM) | Read-only | Not possible | Mask written | Non-volatile |
| Programmable ROM (PROM) | | Not possible | Electrically written | |
| Erasable PROM (EPROM) | Read-mostly | UV-light at chip-level | | |
| Electrically Erasable PROM (EEPROM) | | Electrically at byte-level | | |
| Flash Memory | | Electrically at block-level | | |

# Cache Memory

# What is a Cache?

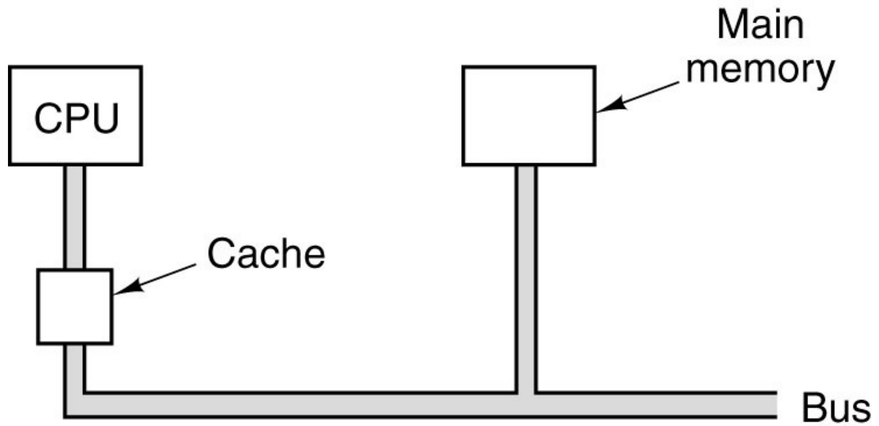
"90% of memory accesses within only 2 Kbytes"

So store those 2 Kbytes in a small, fast "cache" memory

If data required by CPU is in the cache (a "cache hit"), big speed improvement

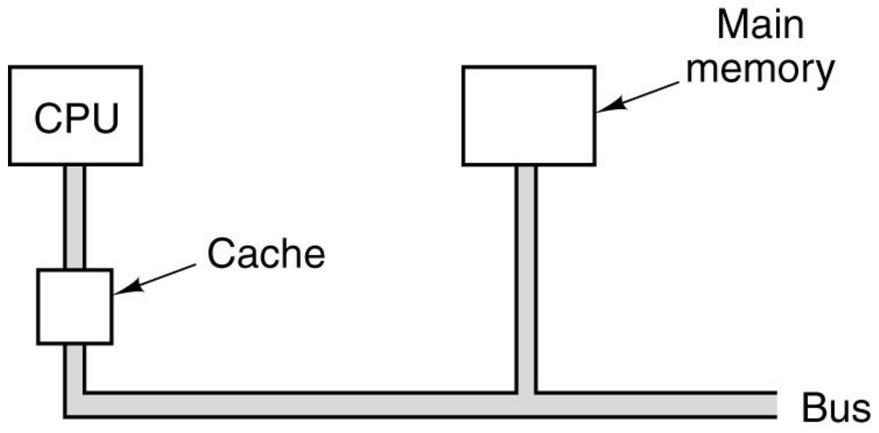
Cache is small to limit cost

Note that "cache" means "hiding place" - transparent to programmer



# What is a Cache?

| | |
|---|---|
| execute typical instruction | 1/1,000,000,000 sec = 1 nanosec |
| fetch from L1 cache memory | 0.5 nanosec |
| branch misprediction | 5 nanosec |
| fetch from L2 cache memory | 7 nanosec |
| Mutex lock/unlock | 25 nanosec |
| fetch from main memory | 100 nanosec |
| send 2K bytes over 1Gbps network | 20,000 nanosec |
| read 1MB sequentially from memory | 250,000 nanosec |
| fetch from new disk location (seek) | 8,000,000 nanosec |
| read 1MB sequentially from disk | 20,000,000 nanosec |
| send packet US to Europe and back | 150 milliseconds = 150,000,000 nanosec |

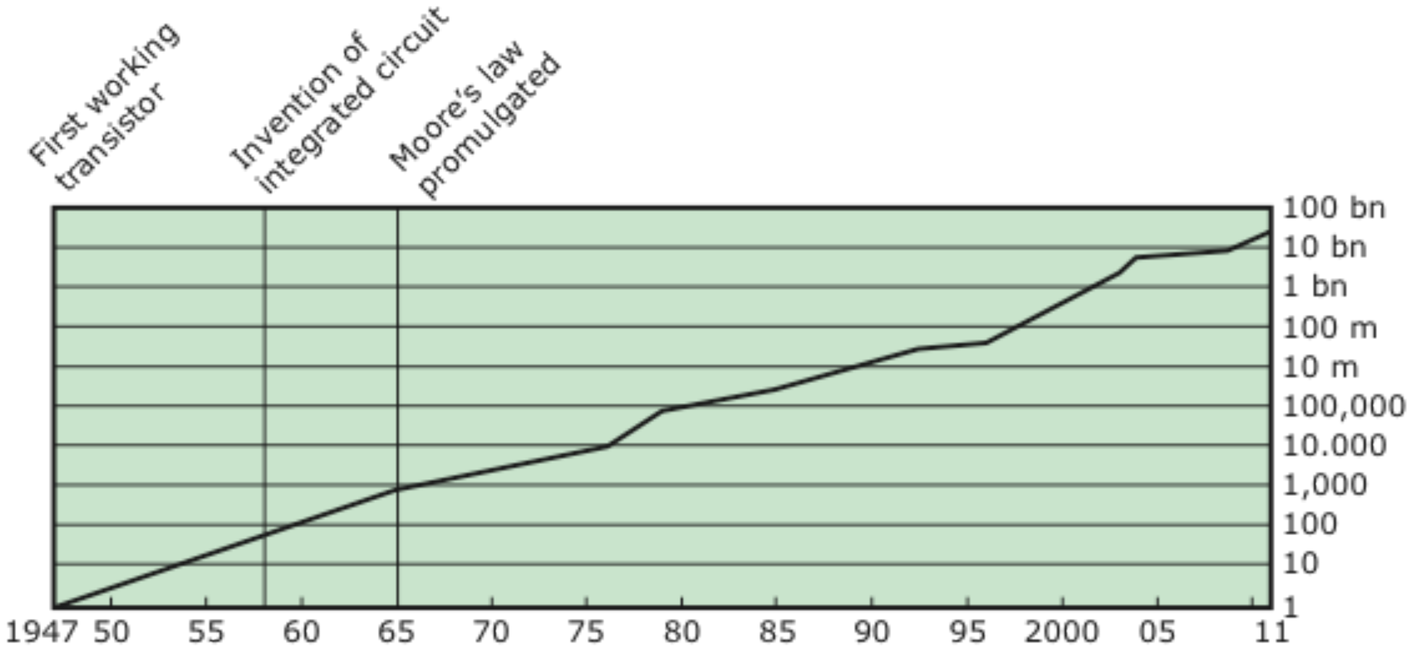


# Moore’s Law

“The complexity for minimum component costs has increased at a rate of roughly a factor of two per year.  Certainly over the short term this rate can be expected to continue, if not to increase.  Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years.” ***- Gordon Moore (co-founder of Intel), Electronics, Volume 38, No. 8, April 1965***

Now more commonly expressed as “The number of transistors on a memory chip doubles every 18 months”

# Moore’s Law




Reproduced from: W.Stallings, “Computer Organization and Architecture”, 9th Edition, Pearson, 2013

# Moore’s Law

The cost of computer logic and memory circuitry has fallen at a dramatic rate

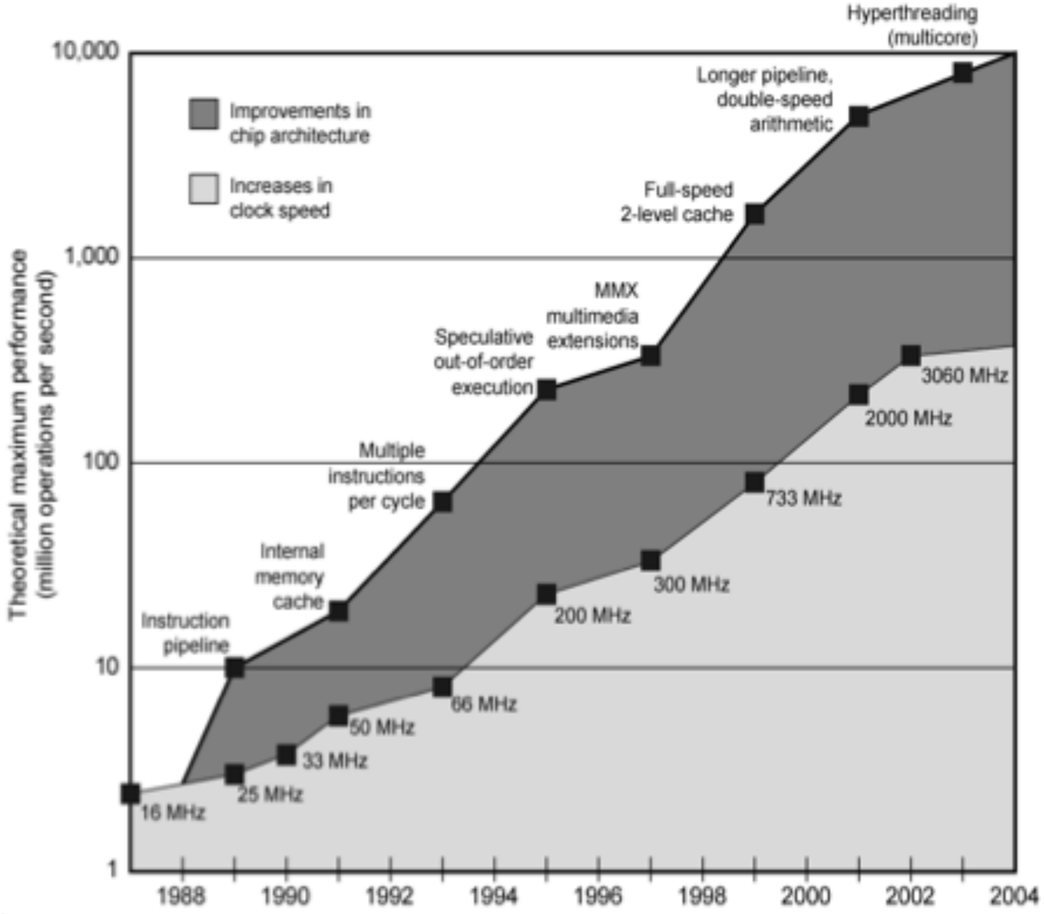
As logic and memory is placed closer together on more densely packed chips, the electrical path is shortened, increasing operating speed

Computers have become smaller, making them more convenient for use in a variety of environments, especially embedded systems

Reduction in power and cooling requirements

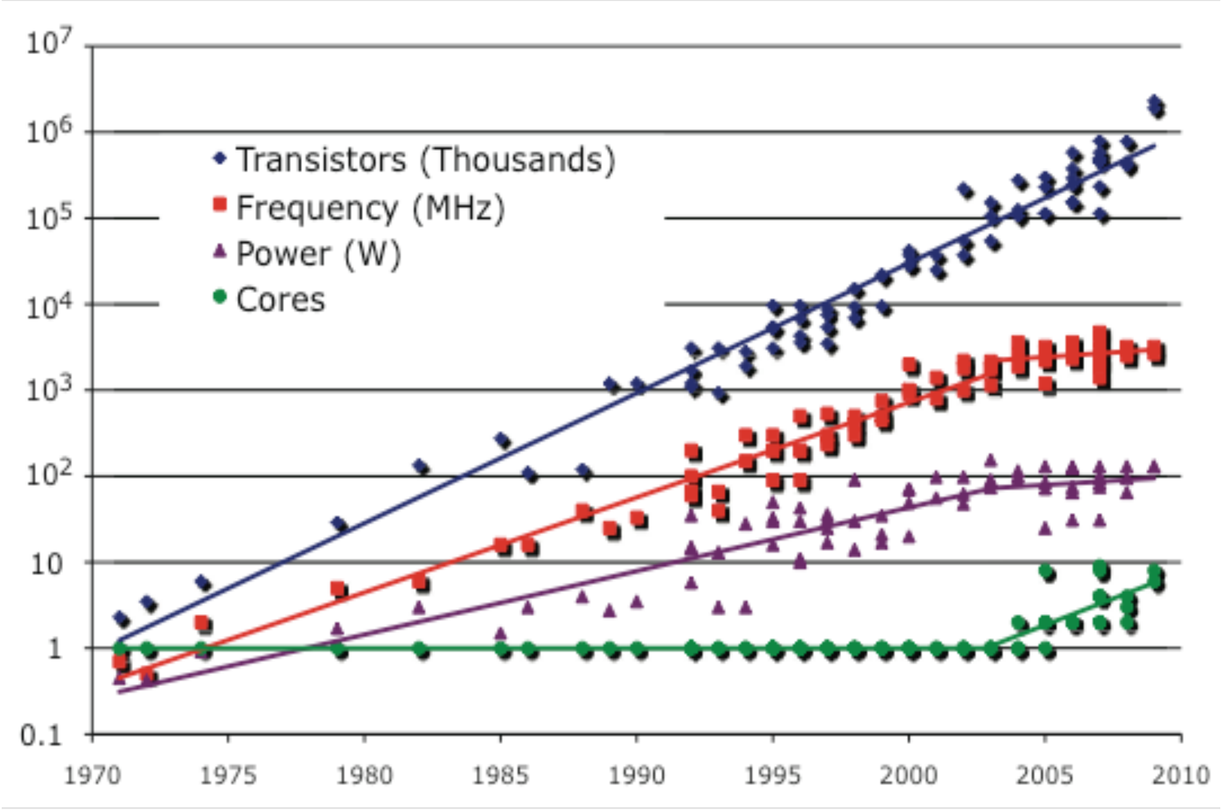
Interconnections on an IC are more reliable than solder connections, thus having fewer off-chip connections increases reliability

# Intel Microprocessor Performance




Reproduced from: W.Stallings, “Computer Organization and Architecture”, 8th Edition, Pearson, 2010

# Current Outlook - Hardware

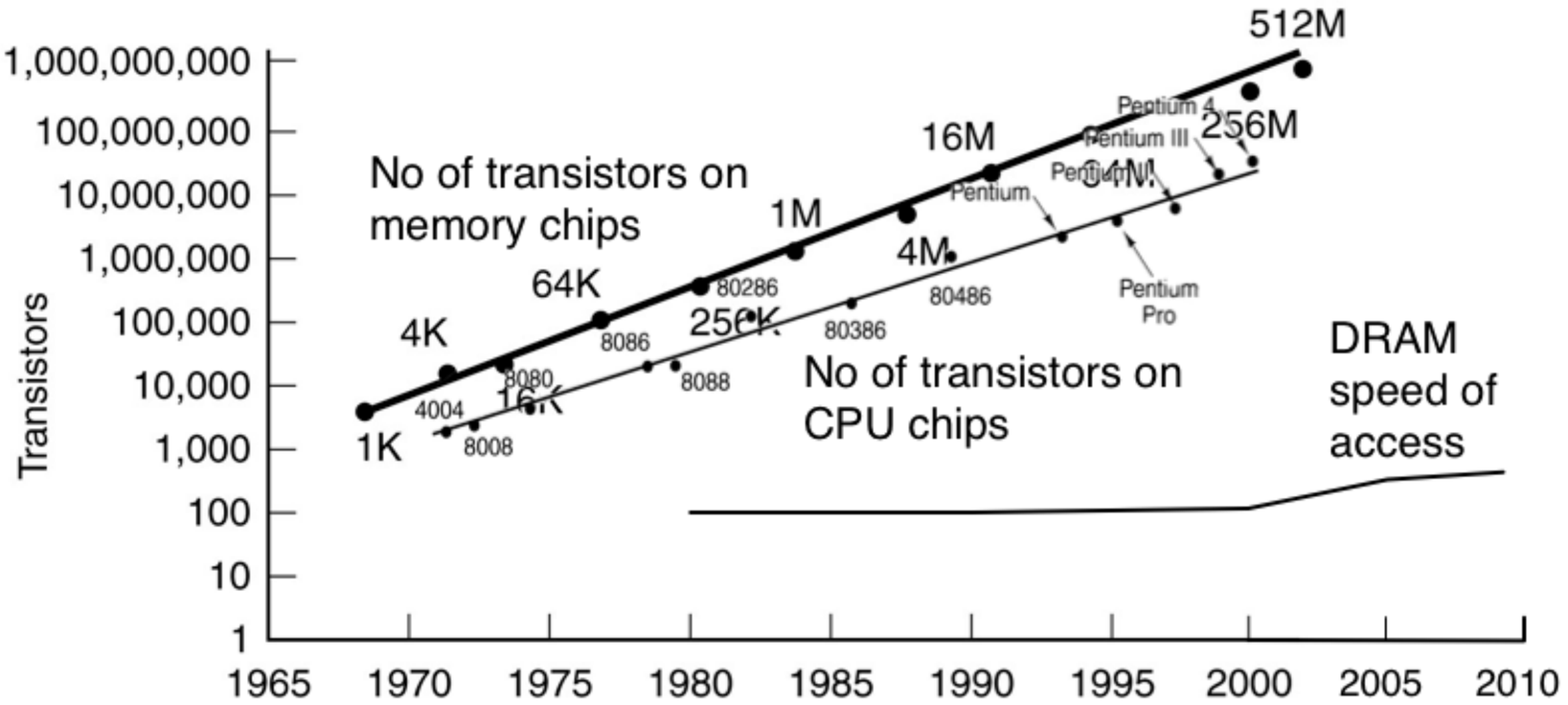



Reproduced from: W.Stallings, "Computer Organization and Architecture", 9th Edition, Pearson, 2013

# Why Have a Cache?

Memory size is increasing according to Moore's Law

Memory access speed is improving much more slowly



# Cache Concepts

Caching read-only data is relatively straightforward

> We don’t need to consider the possibility that items will change, hence copies across the memory hierarchy will remain consistent

Two general strategies can be adopted when caching writes, both of which can employ a write buffer to enhance performance

> **Write Through** - Updates the item in cache and writes through to update lower levels of the memory hierarchy
>
> **Write Back** - Only updates copy in cache, ensuring that blocks are copied back to memory when when they are to be replaced

# Types of Cache Miss

Measures of cache performance include hit rate (h) and miss rate (1-h)

$$h = \frac{\text{Number of times the words are in cache}}{\text{Total number of memory references}}$$

We can understand cache misses by sorting all misses into categories

**Compulsory** - Misses that would occur regardless of cache size, e.g., the first access to a block can not be in cache (regardless of cache size) so the block must be retrieved

**Capacity** - Misses occurring because a cache is not large enough to contain all blocks needed during the execution of a program

**Conflict** - Misses occurring as a result of the placement strategy for blocks not being fully associative, which means that a block may be discarded and retrieved

**Coherency** - Misses occurring due to cache flushes in multiprocessor systems

# Measuring Cache Performance

Performance measures based solely on hit / miss rates don't factor in the cost of a cache miss - the real performance issue!

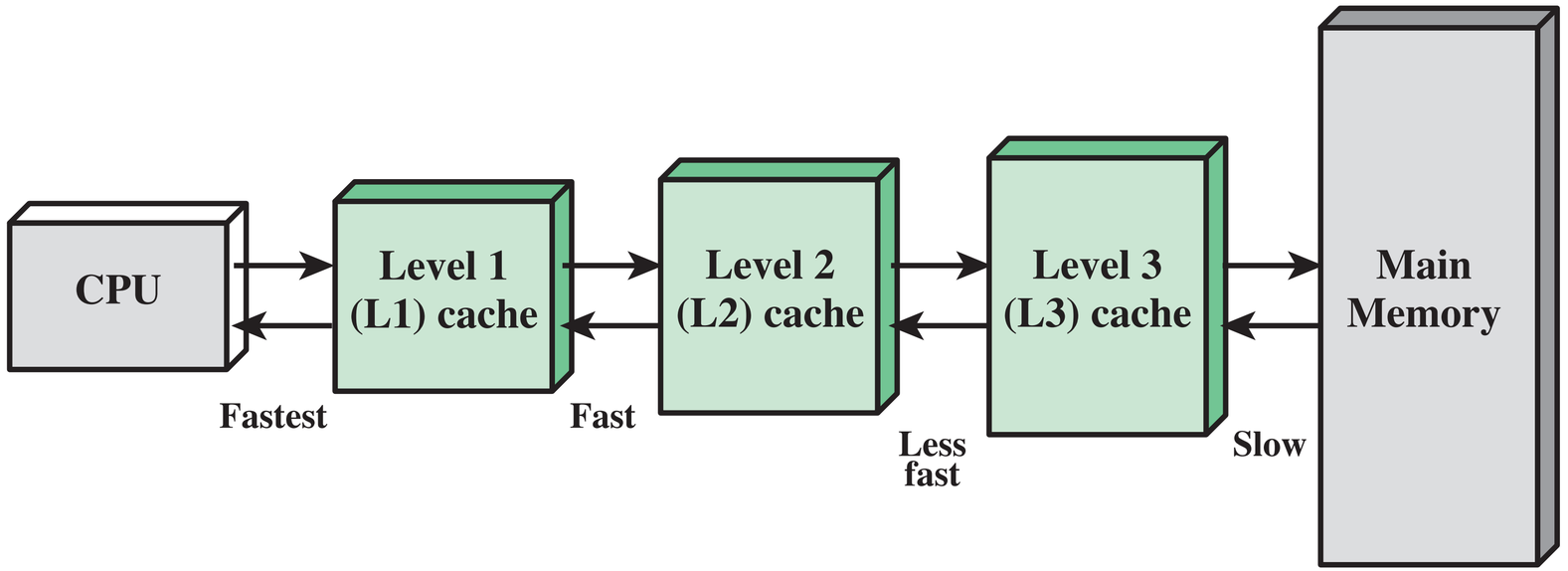
Average memory access time is an alternative measure that accounts for the cost of a cache miss

$$\text{Average memory access time} = \text{Hit time} + (\text{Miss rate} \times \text{Miss Penalty})$$

This is where hit time is the time to hit in the cache and the miss penalty is the time taken to replace the block from memory

Even average access time is still an indirect measure of performance - execution time can often be a better measure in real-world situations

# Multilevel Cache




Reproduced from: W.Stallings, “Computer Organization and Architecture”, 9th Edition, Pearson, 2013

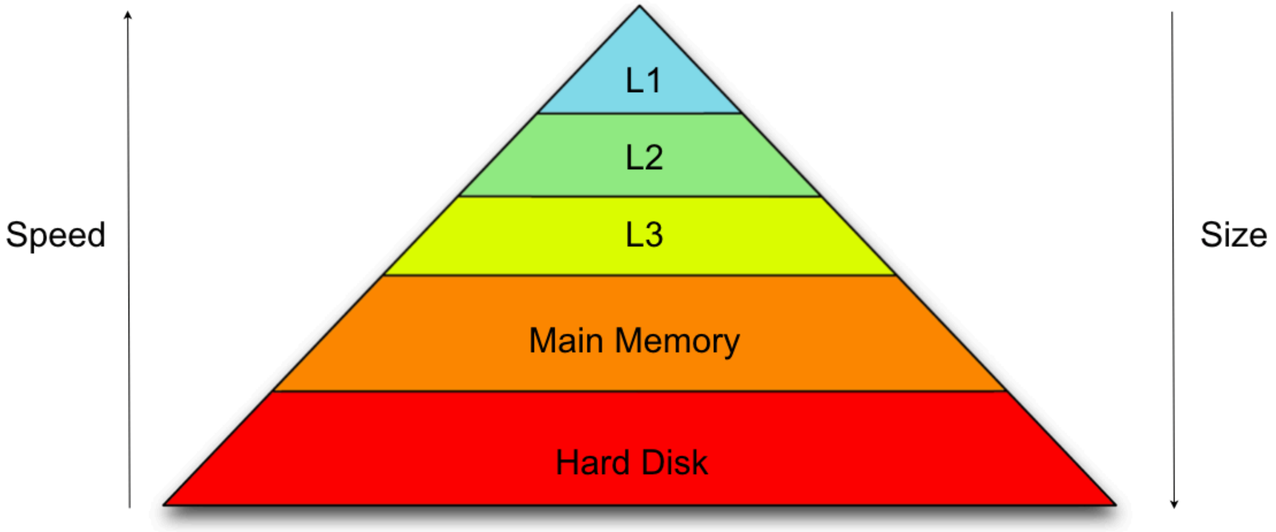
# Relating the Memory Hierarchy to Cache

Data storage is a balance of size and speed:

Memory closer to the CPU (Cache - L1, L2, L3) is typically faster but has a far smaller capacity.

Main memory is further and slower but has a far greater capacity

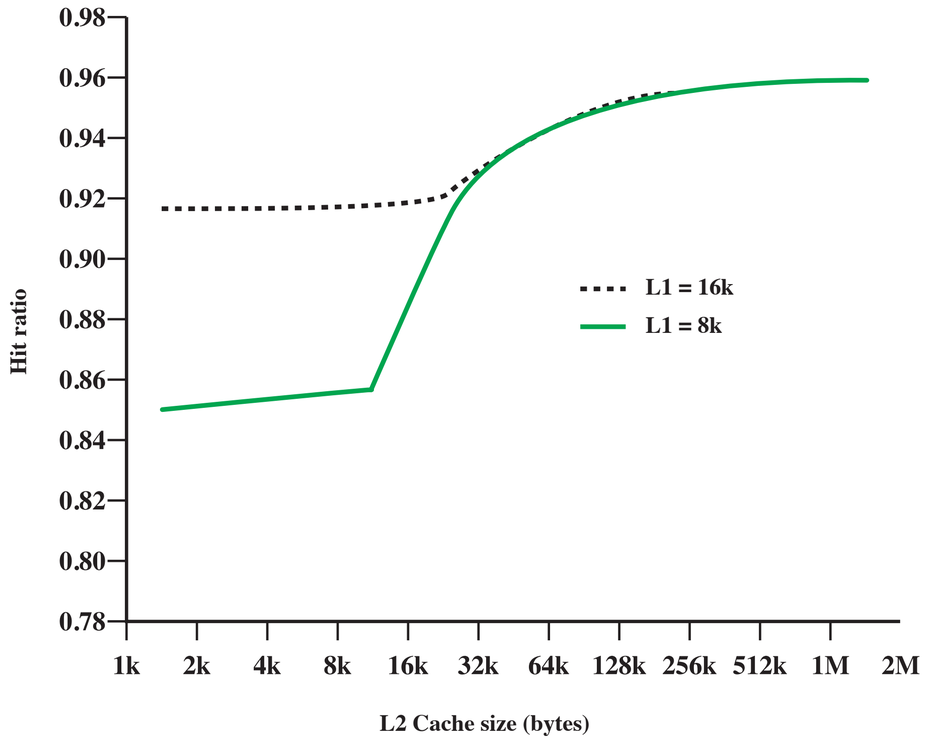
Hard disks have the greatest capacity, and do not need continuous power, but have the slowest access times.




**Optimisation:** We want to use fastest memory possible for high performance, but problems may large datasets.

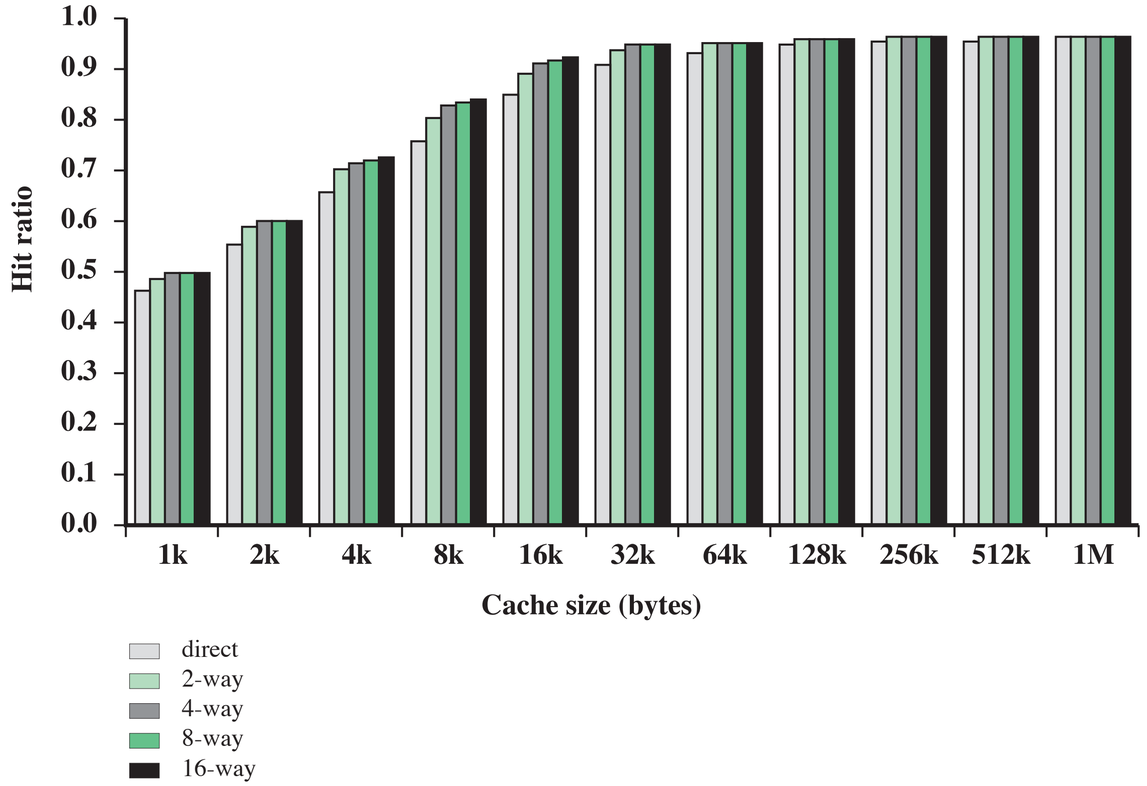
**Solution:** Reuse data already in cache as much as possible and prefetch data from main memory into cache before it is needed, masking load times

# L1 Cache Size and Complexity




Reproduced from: W.Stallings, “Computer Organization and Architecture”, 9th Edition, Pearson, 2013

# L1 Cache Size and Complexity




Reproduced from: W.Stallings, “Computer Organization and Architecture”, 9th Edition, Pearson, 2013

# Memory Cell Organisation

# Semiconductor Memory

Most common form of main store

RAM (Random Access Memory)

Two main technologies

- Static RAM (SRAM)
- Dynamic RAM (DRAM)

SRAM use a flip-flop as storage element for each bit

DRAM for each bit, use the presence or absence of charge in a capacitor to denote a 1 or 0

- Capacitor charge leaks away over time - requires periodic refreshing (more complex) but is cheaper than SRAM so is more commonly used

# Static RAM (SRAM)

SRAM stores data using configurations of flip-flops and logic gates

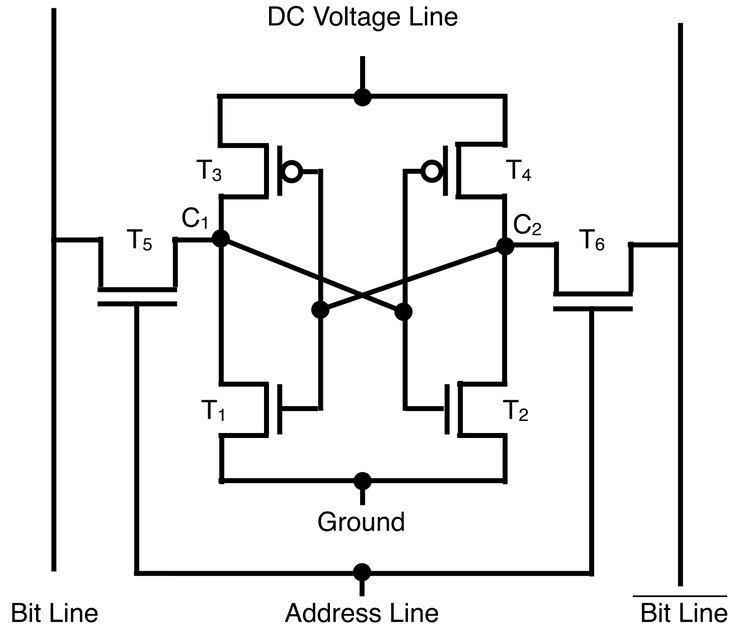
Memory cells hold data as long as power is supplied

Typical SRAM cell uses four cross-coupled transistors to establish a stable logic state

When storing logical 1, C1 is high and C2 is low (T1 and T4 off, T2 and T3 on)

When storing logical 0, C1 is low and C2 is high (T1 and T4 on, T2 and T3 off)

Address line controls two transistors (T5 and T6), where both transistors being on permits read and write




Write operation by applying bit value to Bit Line, which sets T1, T2, T3 and T4 to the appropriate bit value

Read operation by reading Bit Line

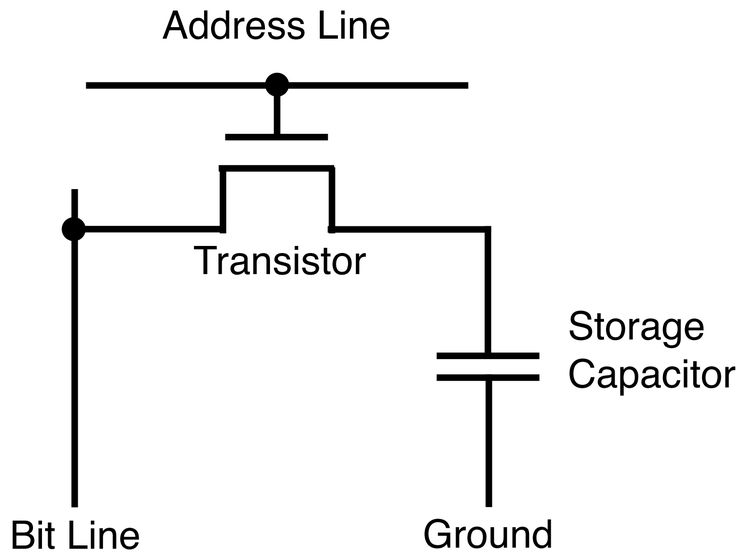
# Dynamic RAM (DRAM)

DRAM memory cells store data as charge on capacitors

- Capacitors naturally discharge, hence DRAM cells require periodic charge to maintain data storage
- The term ‘dynamic’ refers to the natural leakage of charge that takes place even when power is applied
- Presence or absence of charge is interpreted as a logical 0 or 1

We can appreciated the operation of DRAM by considering a 1-bit DRAM memory cell

- Address line is activated when the bit value from the cell is to be read or written
- A transistor acts as a switch that is closed (allowing current flow) if a voltage is applied to the address line and open if a voltage is not applied to the address line (no flow)



# DRAM Operation

**Write operation:**

Voltage applied to the bit line (high for logical 1, low for logical 0)

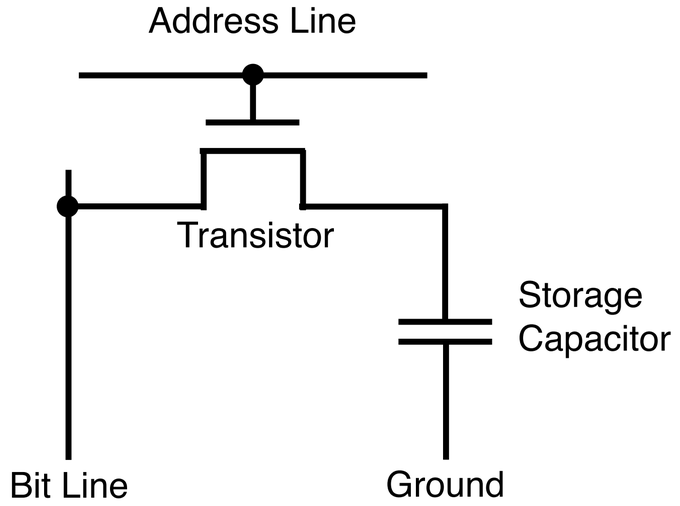
A signal is then applied to the address line, allow a charge to be transferred to the capacitor (data store)

**Read operation:**

Transistor turns on when address line is selected, allowing the stored charge to be fed out to the bit line and a sense amplifier

Sense amplifier compared the capacitor voltage to a predetermined threshold value to determine if the cell contains a logic 0 or 1

The read process discharges the capacitor, which means that it must be restored (as well as refreshed) following the read operation



# Comparing SRAM and DRAM

Both are volatile - power must be continuously applied

Dynamic memory cells are generally simpler and more compact

- Allows greater memory cell density (more cells per unit area)
- Cheaper to produce than equivalent SRAM memory

Refresh circuitry incurs one-off cost that is only compensated for by larger memory capacities

SRAM are typically provide better read and write times than DRAM

- Cache memory (on and off chip) is often implemented as SRAM, whilst DRAM is commonly used for main memory

# Advanced DRAM Organisations

The interface to main memory is a critical performance bottleneck

Motivation for multiple high-speed SRAM caches between a processor and DRAM main memory

To achieve better system performance we can also consider advanced DRAM organisations

Synchronous DRAM (SDRAM)

Rambus DRAM (RDRAM)

DDR SDRAM

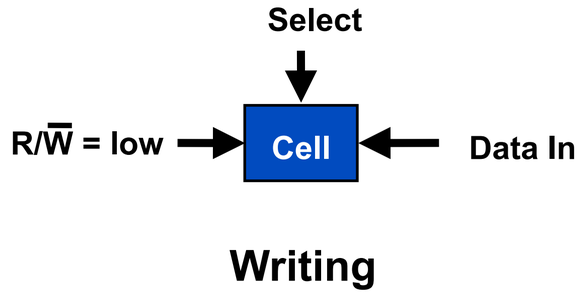
Cache DRAM (CDRAM)

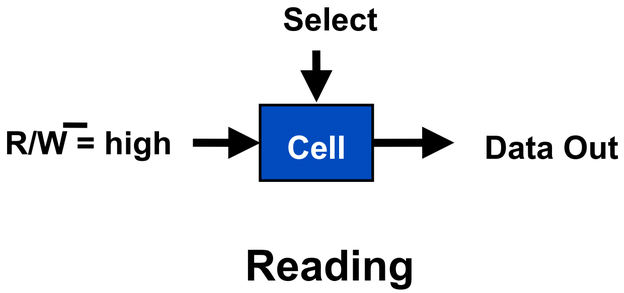
# Memory Organisation

Basic element is a memory cell

Two states 1, 0

Capable of being written and read




**Select** – Enable a particular memory cell

**R/W** – Indicate read or write

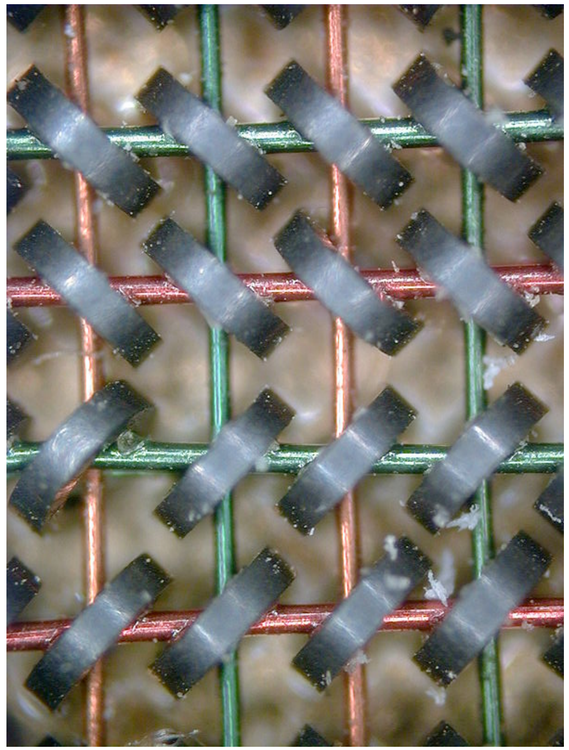
**Data** – Input when writing or output when reading

# Core Memory

An early form of RAM where data is stored through the magnetisation of tiny metal rings – one bit per ring

Rings are organised into a grid array

Individual bits are accessed by energising row and column wires

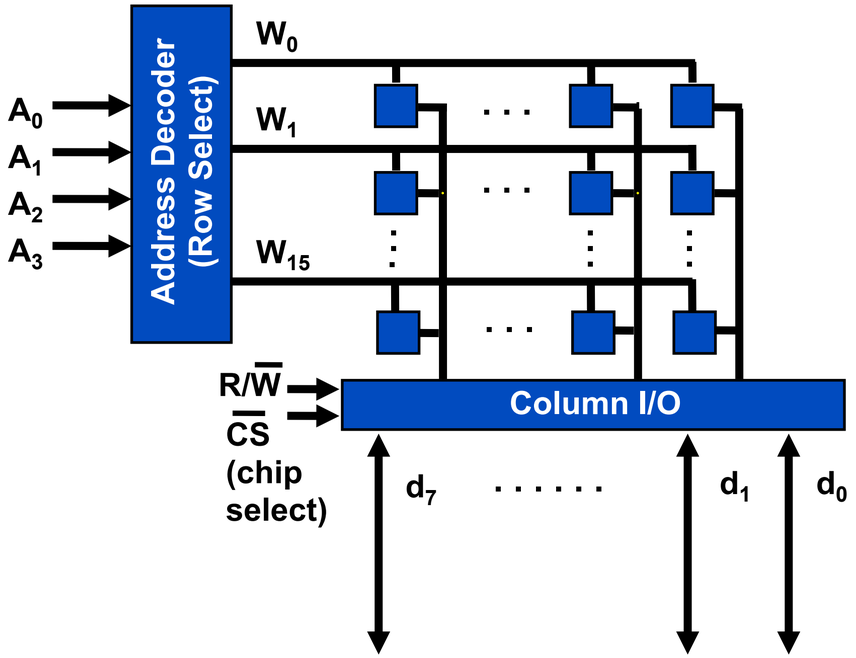


# Organisation of a Memory Chip

As you might expect, it is critical to maintain a structured approach to the organisation of on-chip memory

Example of a 16x8 memory integrated circuit

16x8 denotes
W x B, i.e., words x bits
16 words, each of 8 bits



# Choices in Memory Chip Organisation

IC design for 16x8 = 128 cells required 4 address inputs (= $\log_2$ W, where W = no of words) and 8 data lines (=B, where B = word size)

Consider a 1Kbit =1024 cells device

Possible to organise as a 128 x 8 memory cell array

Requires 7 address pins (log2 128) and 8 data pins

A total of 15 I/O pins, plus power, etc...

# Poor Choices in Memory Chip Organisation

Alternative, it is possible to organise as a 1024 x 1 array

Requires 10 address pins ($\log_2 1024$) and 1 data pin

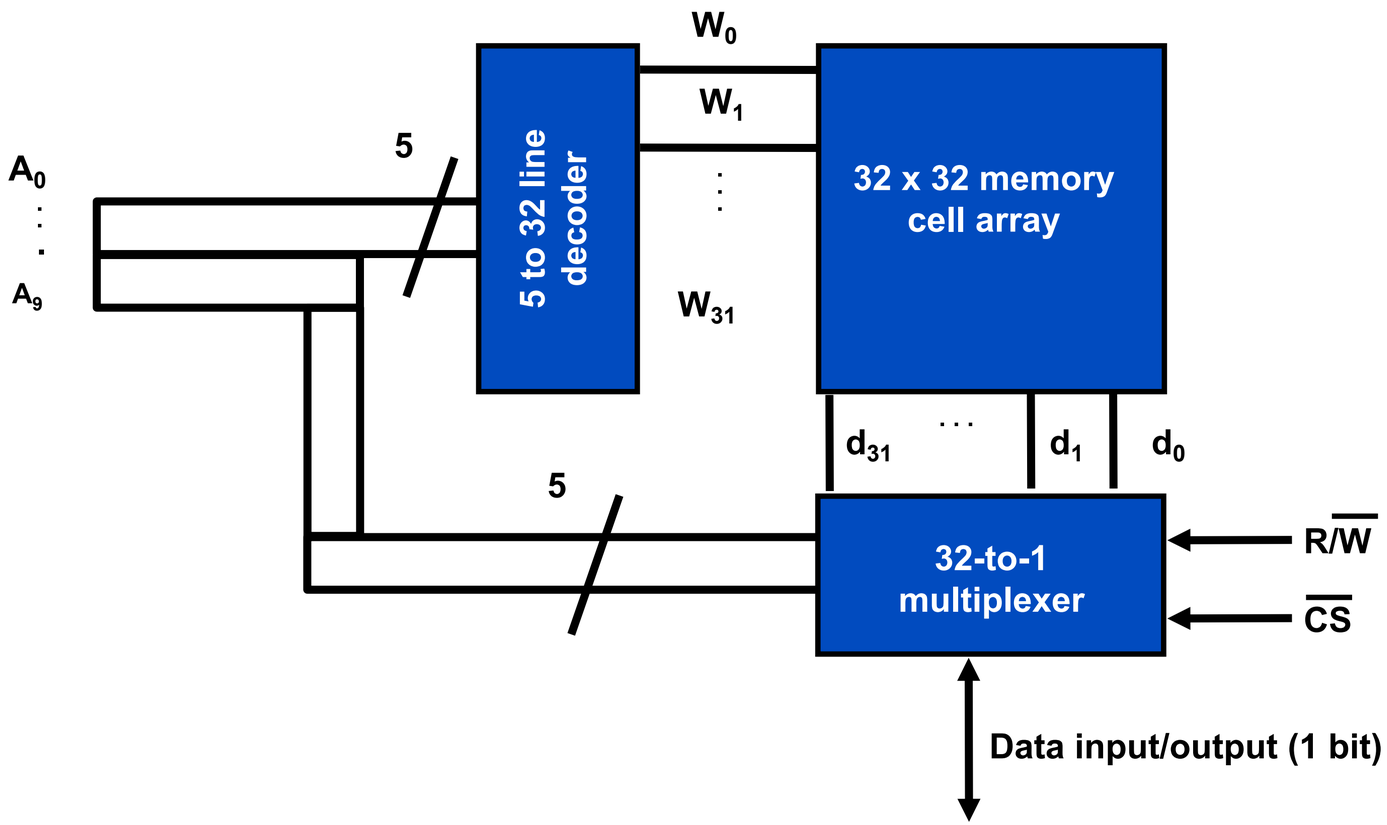
A total of only 11 I/O pins, plus power, etc...

**Results in a long, narrow cell array with a large address decoder, which is an inefficient use of space**

# Minimising Address Decoding Space

Divide the address inputs into 2 parts - row address and column address

This minimises the space required for address decoding and hence maximises space used for memory cells



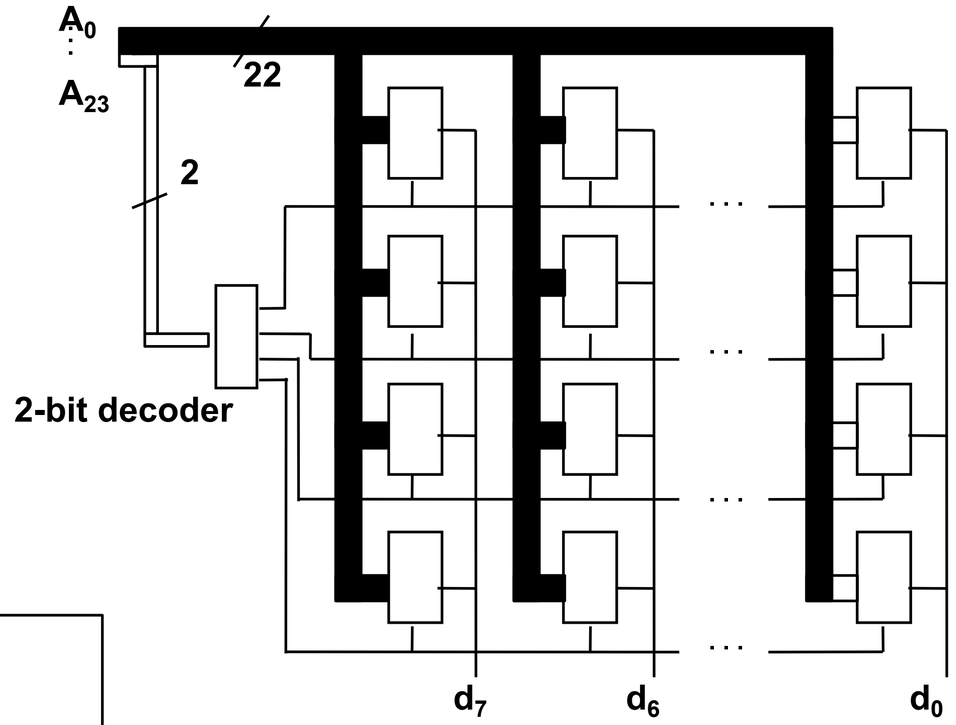
# System Memory Organisation

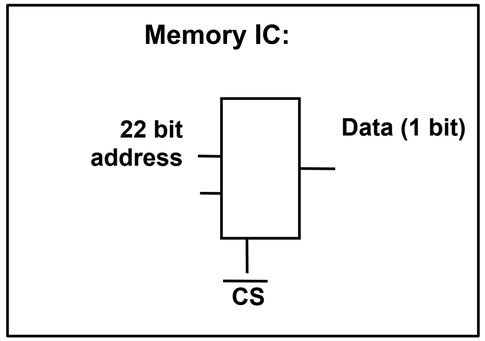
We can use the same idea to organise multiple memory ICs

Here, $A_{22}$ and $A_{23}$ select 1 row of 8 chips, each providing one bit of the 8-bit word

Each chip is 4M x 1 (22 address pins, 1 data pin)

System has 4 rows, 8 chips per row, making the system total a 16M x 8 memory

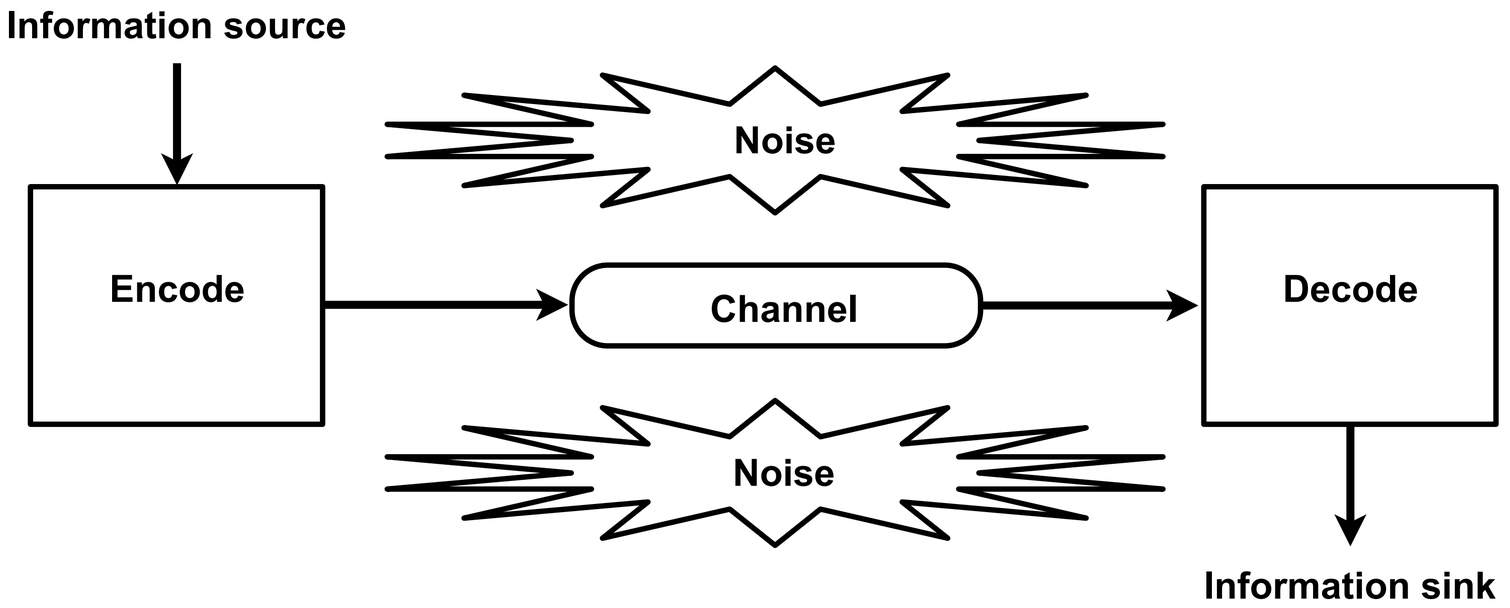


# Detecting and Correcting Errors

# Error Detection and Correction

Errors occur within a computer system, e.g., in system memory, and in the communication between systems, e.g., in the transmission of messages



# Noise

Noise is unwanted information

It comes in various forms, but is always present, and is one of the limiting factors in computer systems

Noise arises from the physical properties of devices

- Thermal noise
- Noise of electronic components
- Noise of transmission circuits

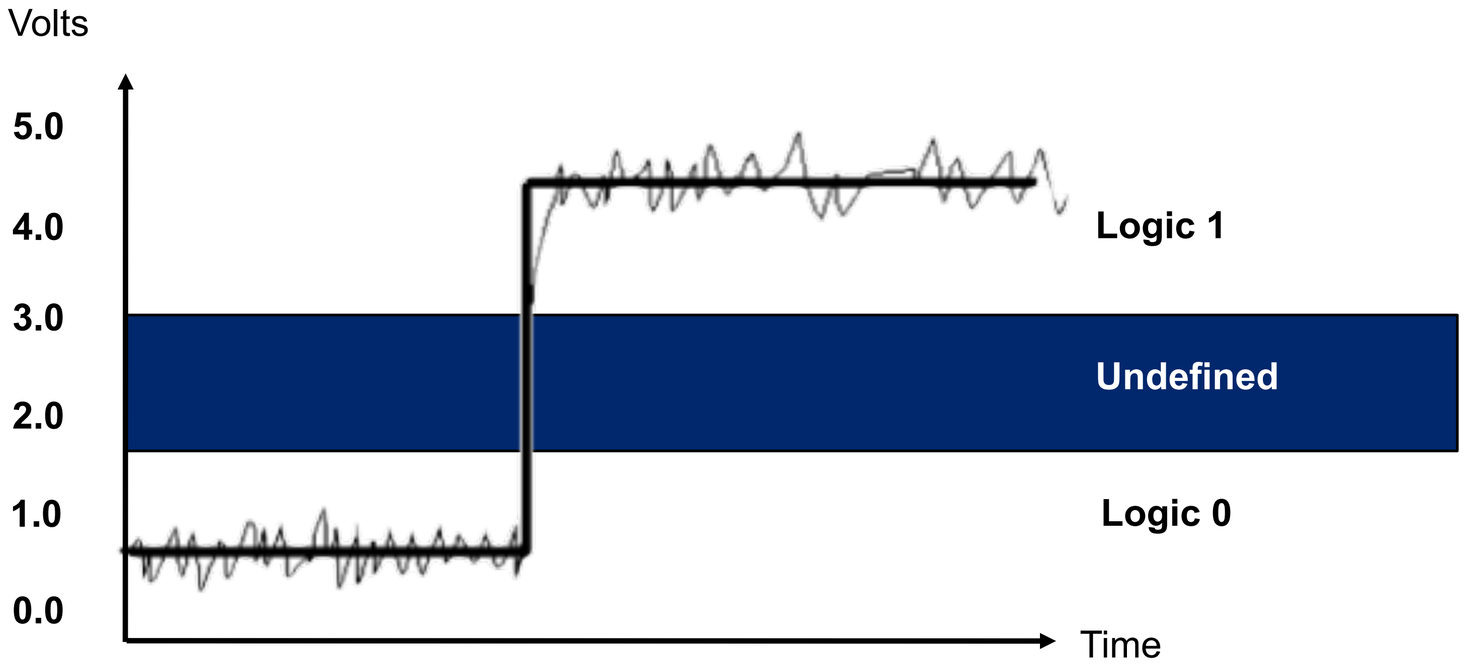
Magnetic media also have a classic form of noise due to random alignment of magnetic fields.

- Decreased area to store a bit means noise gets worse, making errors likely

# Noise and Digital Logic Devices

Digital logic gives us a high degree of noise immunity

Immunity collapses once the noise reaches a certain magnitude - our aim is to not let this happen



# Detecting Single, Isolated Errors

These are considered to occur at random, usually due to noise

We could send the message three times and take a vote

- Very expensive

Observation that, if the probability of an error in the channel is low, the probability of two errors close together is even lower

Thus, we can add a “parity” bit to the message every so often, which “summarises” a property of the message that we can check to determine whether the message has been altered

- Much cheaper and adequate in many situations

# Parity

Parity adds an extra bit

There are two types of parity system.

**Even parity system** - The value of the extra bit is chosen to make the total number of logic 1s an even number

**Odd parity system** - Make the total number of logic 1s odd

Parity bit value can be computed in software or hardware

| | |
|---|---|
| **7-bit ASCII for 'A':** | **A = 0100 0001** |
| **Using even parity system:** | **A = 0100 0001** |
| **Using odd parity system:** | **A = 1100 0001** |

# Parity in Software

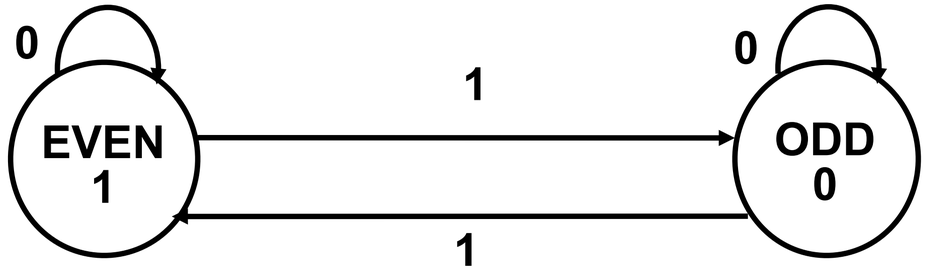
Odd parity system - Start at EVEN

Even parity system - Start at ODD

Each 1 in the message causes a transition of state

Final state at the end of the message is the resulting parity bit

**Parity may be computed using a two state Finite Automaton**



# Parity in Software - Example

For example, try sending 110 using an even parity system

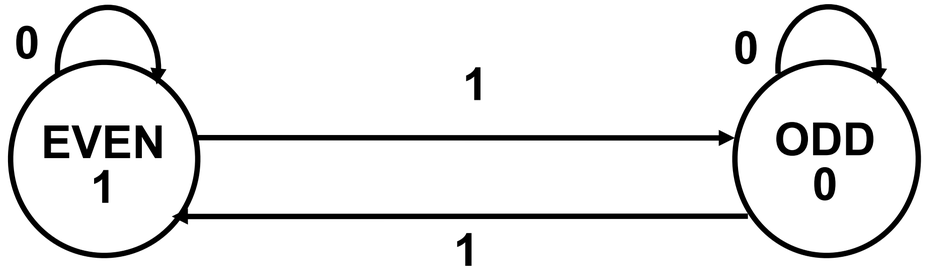
Start with the ODD state, then

Bit 0 = 0, so go from ODD to ODD

Bit 1 = 1, so go from ODD to EVEN

Bit 2 = 1, so go from EVEN to ODD

**Parity may be computed using a two state Finite Automaton**

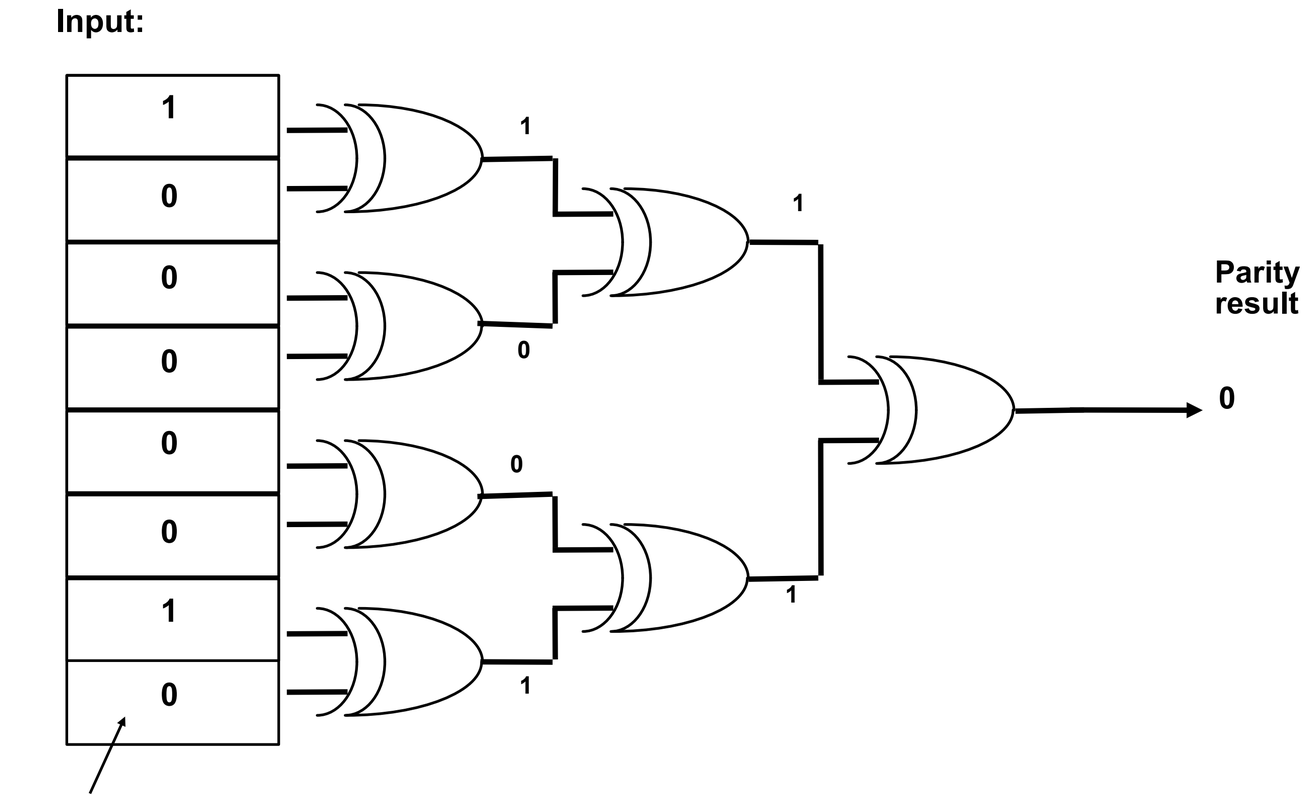



Therefore the parity bit is 0 and the message to be sent is 0110

# Parity in Hardware

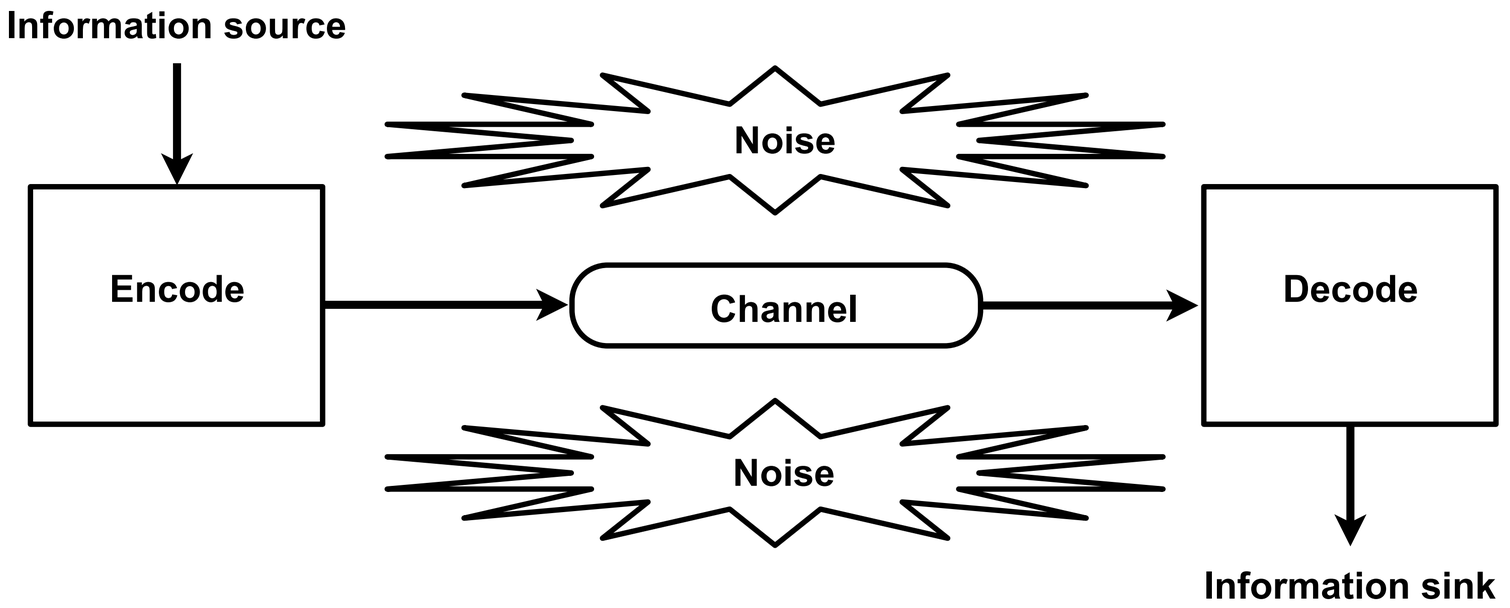
Computing parity in hardware is popular

Hardware implementations generally perform better than software implementations, despite being a relatively simple digital logic circuit



# Overview of Parity

Parity provides error detection without the need to significantly change our model for communication



# Error Detection Using Parity

**At our transmitter a message, e.g., 7 bits, is encoded**

Compute parity (P) and append parity to message

Send modified message across channel

**At our receiver the message, now 8-bits including parity, is decoded**

Compute parity (Q) from data bits of received message

Compare computed parity Q with received parity P in message and flag ERROR if these are not consistent

# Error Detection Using Parity - Example

Transmitter and receiver have been designed to use even parity

**Message: 100 0101**

Transmitter computes P=1

Transmits: 1100 0101

**An error in the channel changes one bit**

Received: 1101 0101. Receiver notes received P=1

**Receiver uses the data bits (101 0101) to compute Q=0**

Receiver compares received P and computed Q.

**An error has been detected, thus receiver will ask for retransmission**

# Burst Errors

Remember that, if the probability of an error in the channel is low, then the probability of two errors close together is even lower

In practice, multiple errors often occur in bursts, e.g., a 1ms dropout on a 56k baud modem link can error up to ~58 bits

Worse on broadband, e.g, ADSL

Parity can detect single isolated errors but an even number of errors close together leaves parity unchanged, e.g., 0100 0001 => 0100 1101 => parity unchanged!

The form of parity we have seen is inadequate for burst errors

A possible solution is to compute “checksum” values, i.e., the parity of each bit-column over multiple bytes

# Detecting Burst Errors - Example

Checksums vastly improve our error detection capabilities and allow for the detection of burst errors

1. To send the following message over a channel:

(NB 7-bit ASCII without parity)

| | |
|---|---|
| **M** | 100 1101 |
| **e** | 110 0101 |
| **s** | 111 0011 |
| **s** | 111 0011 |
| **a** | 110 0001 |
| **g** | 110 0111 |
| **e** | 110 0101 |

2. Calculate bit-column parity "checksum" values, e.g., using even parity

**100 1011** = ASCII '**K**'

3. Send "Message**K**" instead of "Message"

**This system detects all burst errors < 14 bits**

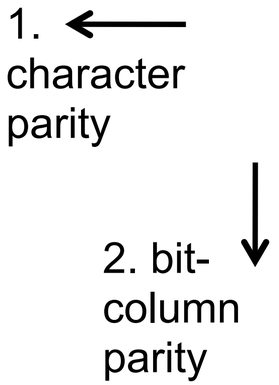
**Fails if an even number of errors occurs in a bit-column**

# Error Correcting Codes

The checksum codes we have seen (those with bit-column parity) could detect some burst errors

We can extend to an Error Correcting Code (ECC) by calculating both character and bit-column parity

| | | |
|---|---|---|
| M | **0** | 100 1101 |
| e | **0** | 110 0101 |
| s | **1** | 111 0011 |
| s | **1** | 111 0011 |
| a | **1** | 110 0001 |
| g | **1** | 110 0111 |
| e | **0** | 110 0101 |
| **K** | **0** | **100 1011** |



# ECCs - Example

**At transmitter we computer and transmit**

Compute character parity bits, then bit-column parity value, then transmit message with these added bits

**At receiver we re-compute, compare and correct if needed**

Compute character parity bits, ignoring received parity bits

For each character, compare re-computed character parity with received parity

If not equal, ERROR somewhere in row

Re-compute bit-column parity value

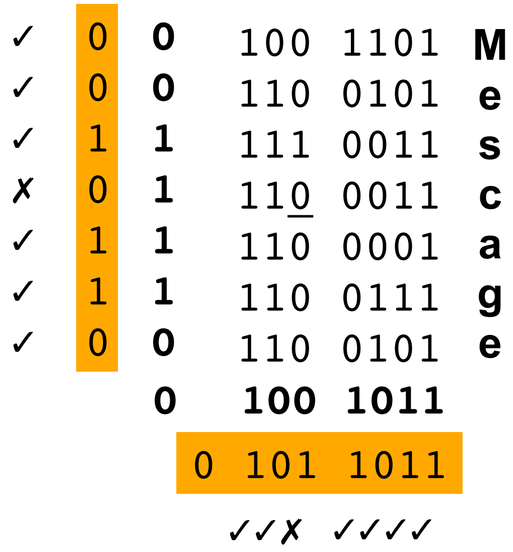
Compare re-computed value with received value

If not equal, ERROR somewhere in column

If a single error detected, correct by inverting bit in message at (row, column)

# ECCs - Graphically

A graphical representation allows us to think clearly about the potential issues and capabilities of a particular ECC




Key:

| | |
|---|---|
| 1101 | – received message bits |
| 0 | – bit changed by noise |
| **1011** | – received parity bits |
| 1011 | – re-computed parity bits |
| ✓✓✗ | – comparisons |

This ECC can detect and correct single errors

It can also detect, but not correct, certain combinations of multiple error

# Common Memory Components

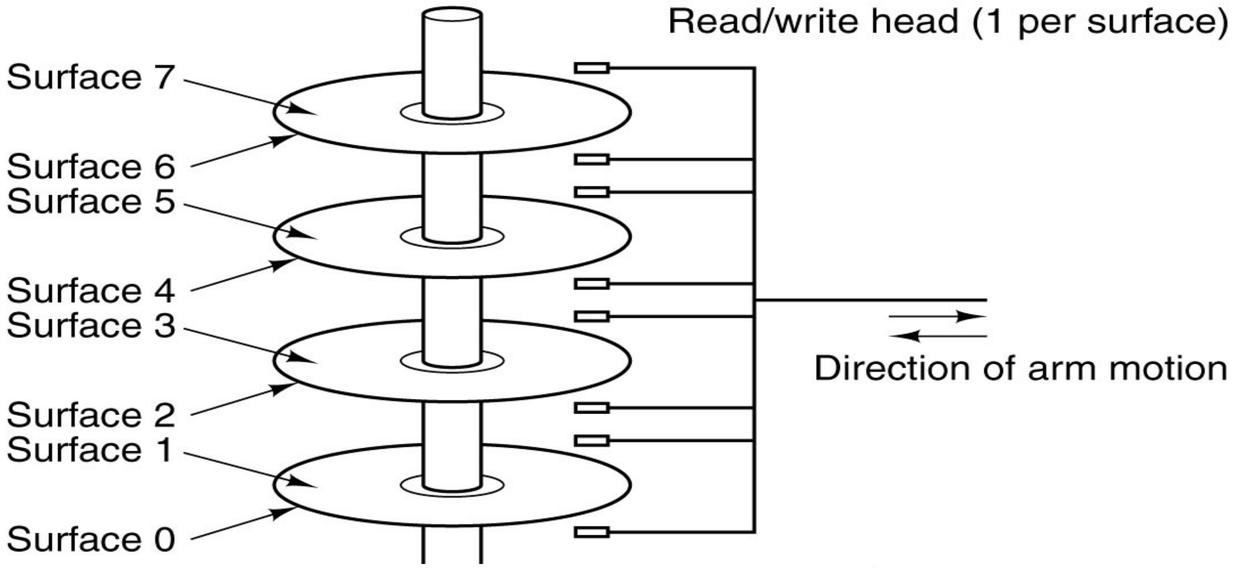
# Magnetic Hard Disks - Platters, Surfaces and Heads

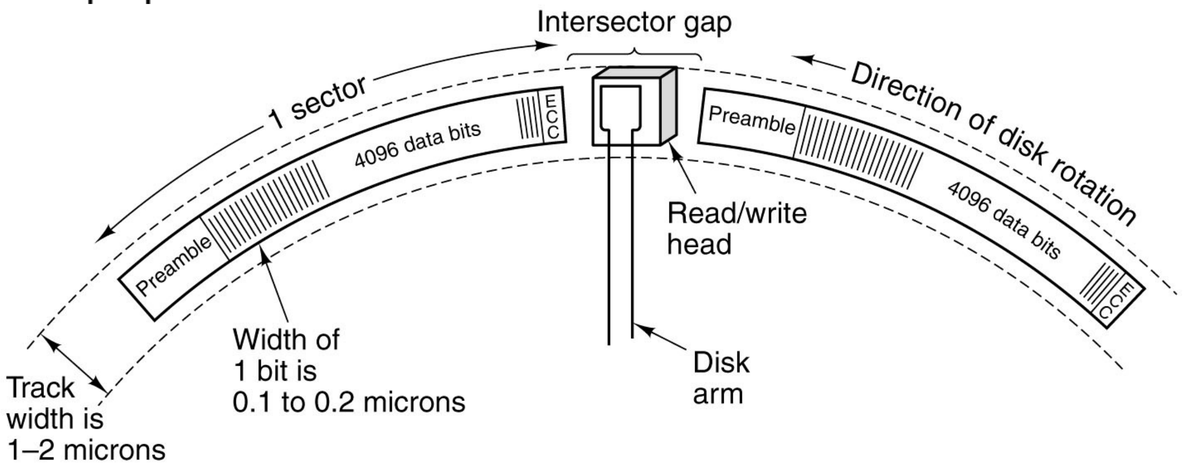
Record data by magnetising a thin film of ferromagnetic material on a disk

One track contains many sectors

Each sector is separated by an inter-sector gap.

Sector contains preamble to allow head to be synchronised before read/ write, data and ECC



# Magnetic Hard Disks - Performance - What Can We Learn?

Seek times - time to move arm to correct track

- Typically about 5ms between different tracks but can be below 1ms for consecutive tracks

Rotational latency

- Constant angular velocity, i.e., motor spins at constant speed
- Drives available with rotational speeds of 5400 RPM, 7200 RPM or 10800 RPM
- Average delay (half a rotation) = 3 to 6ms

Sector read time is about 0.013ms, which means that seek time and rotational latency dominate access time.

Formatting (preambles, ECC etc...) reduces capacity by about 15%

# Hard Disks - Increasing Capacity with Zones - More To Learn?

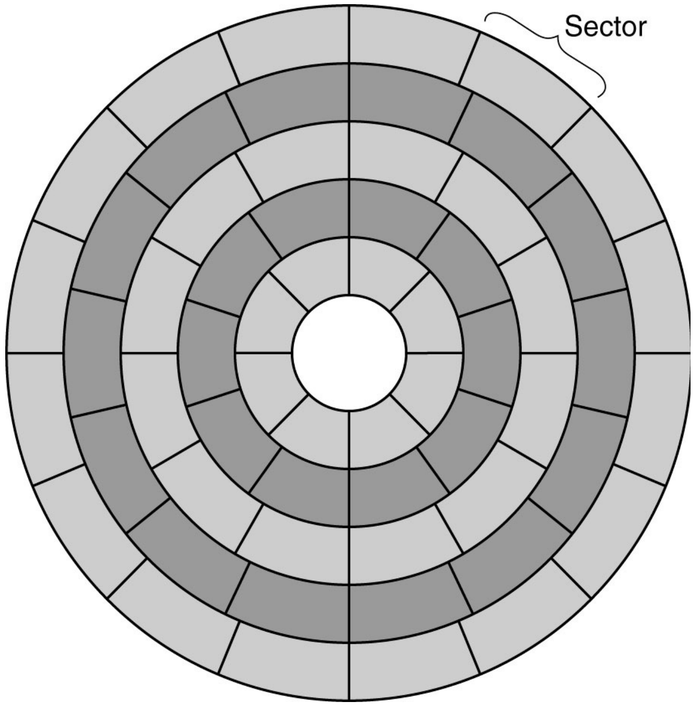
Older drives have a constant number of sectors per track, which meant that, due to constant angular velocity, bit density is lower on outer tracks

Most Recent disks are zoned (10-30 zones), with more sectors in the longer (outer) tracks

Many tracks per zone

More sectors gives higher capacity

Graphic shows a disk with 5 zones



# Optical Disc - Spiral Track

Originally designed to hold music

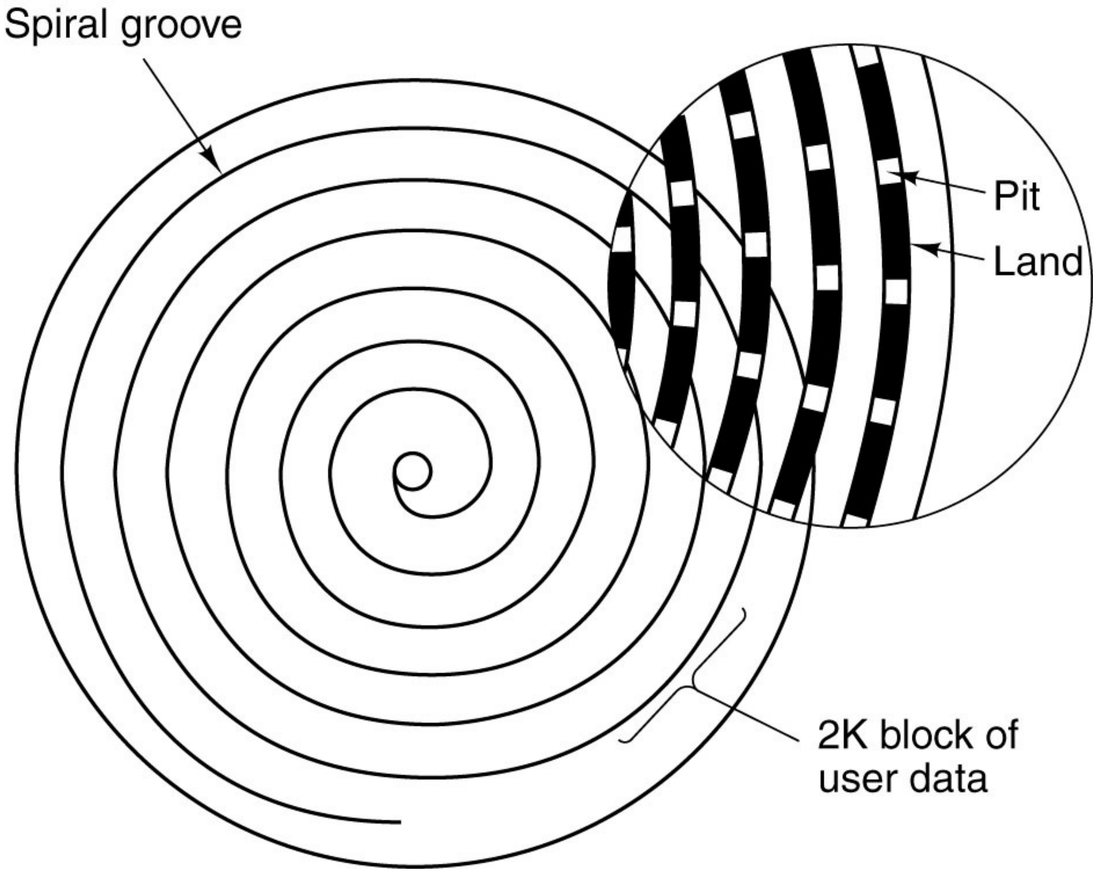
Max 74 mins - Beethoven's Ninth Symphony!

Spiral is 3.5 miles long if unwound

6,000 tracks/cm

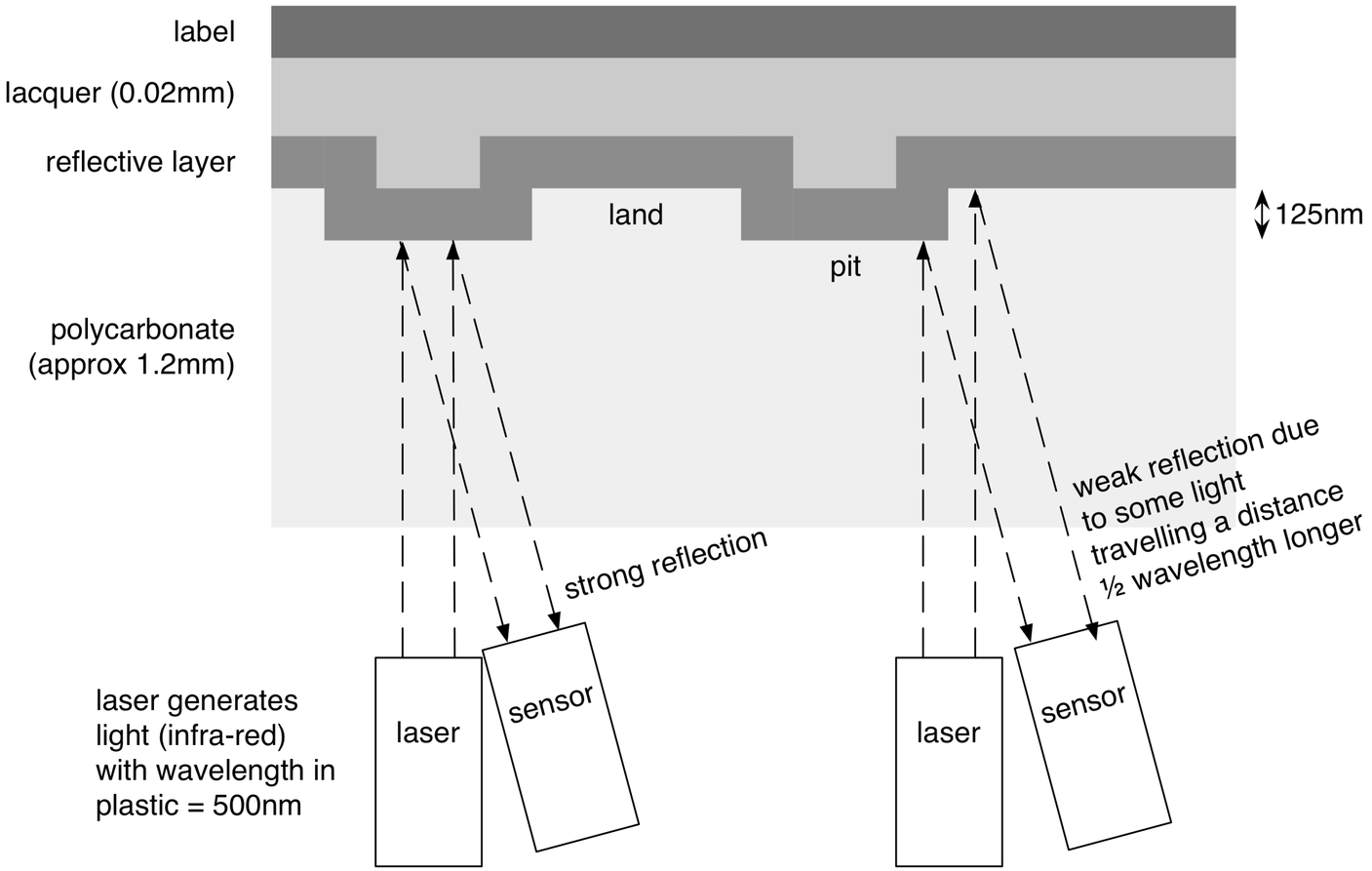
Data encoded as "pits" and "lands"

15,000 pits and lands/cm



# Optical Disc - Spiral Tack Detail

Consider a much closer look at how op



# Optical Disc - Data Encoding

A change in state (pit/land, land/pit) signifies data is a 1

Data represented as bytes, but each 8 bit byte is mapped to a 14 bit symbol (first level of CD-ROM ECC)

Symbols are chosen such that there are always two 0s between 1s and a max of ten 0s in succession

Of the 214 possibilities, only 267 values fit the rules

This is more than sufficient for the 256 values (for the 8 bits) that are to be represented

42 symbols make a frame (containing 24 bytes of data)

98 frames make a sector (containing 2048 bytes)

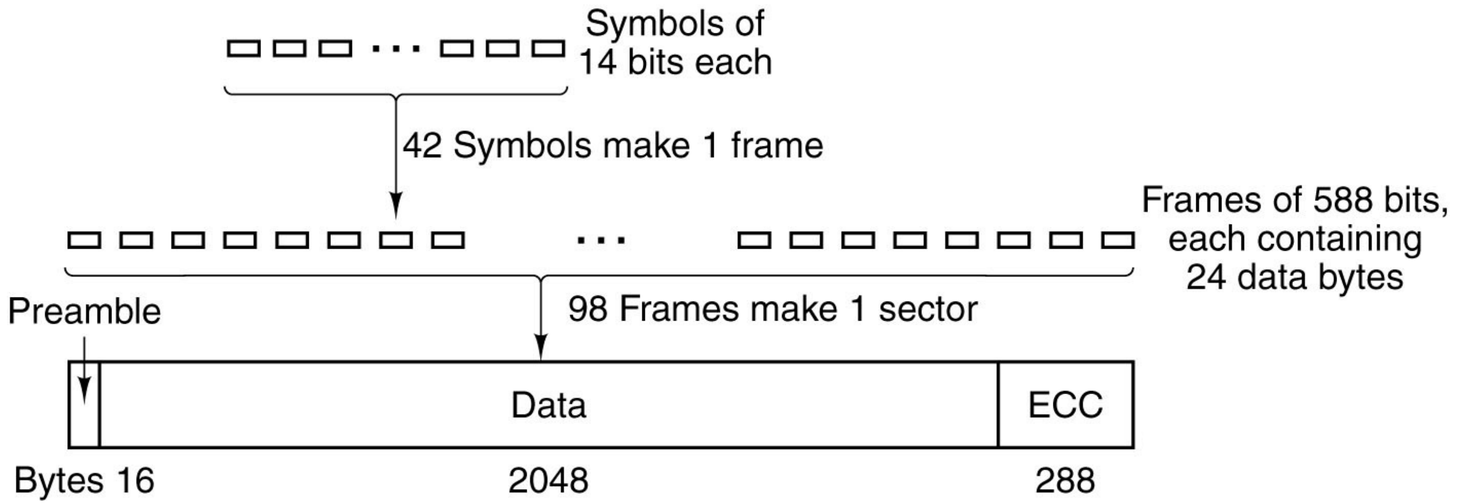
# Optical Disc - Data Encoding Graphically

Multiple separate error correcting schemes

Single bit errors corrected at symbol level

Short burst errors corrected at frame level

Any residual errors corrected at sector level



# Optical Disc - Error Prevention

Most errors are burst and local, e.g., those caused by scratches

Cross Interleaved Reed-Solomon (CIRC) code is used to spread data out over the surface of the disk.

Burst errors of up to 2mm track can be corrected

Data accounts for only 28% of bits stored, thus we pay a heavy price for great reliability

# Optical Disc - Performance

Constant linear velocity, i.e., motor spins at variable speed, to keep the pits and lands moving over the laser at a constant speed

For a “1x” drive, i.e., audio CD speed we have 120 cm/sec (data rate of 150kbytes / sec) and 200 RPM (outer) – 530 RPM (inner part of spiral)

An “8x” drive spins 8 times faster (1600-4000 RPM)

Above 12x speed, CAV is usually used, and the “x” number denotes data transfer rate at the outer edge of the disc, i.e., the max rate

Seek times averages are typically 80ms, which is at least an order of magnitude worse than hard disks