



See the Resources page for more historical links. Note that this tutorial is aimed at getting your thinking: exam questions will be more specific.

Learning objectives for this week:

- Understand the major concepts we've covered, including:
 - types of computer system
 - types of operating system
 - major services an operating system offers
- Apply these concepts to reasoning about OS, including making choices between types of computer and OS.
- Understand the effect of long-term trends, and how they effect what is considered mainstream.

If anyone can do all of these in the first week, it will be great. If you can do the last question, and make a reasonable attempt at the rest, you are off to a good start. You need to aim to be able to answer all of these questions by the end of the course. I will post some notes on how to go about the questions after all the tutorial sessions are over.

1. Moore's Law is a well-known example of a *learning curve law* – a measure of a steady rate of improvement, resulting in exponential improvement over time. A typical rate of such a change in normal life is compound interest. Generally, a learning curve looks like this:

$$P = kR^t$$

where P is final performance after t time units, with a rate of improvement, R . The k is an initial measure of performance. For example, to use Moore's Law to predict the number of transistors you could afford in 3 years, k represents the number you can buy today, $t = 3$ (time is in years) and R is 2 (since the number you can buy doubles in a year):

$$P = k2^3 = 8k$$

A client has observed this trend, and has also noted that an algorithm for continuous speech recognition was only viable on high-end mainframes in 1990, and wants you to predict when it will be viable for cheap PDAs (under \$100). You need to invert the usual sense of Moore's Law (how much more can you buy for the same money), to predict instead how long it will be before a given technology meets a price point. A consultant has plotted trends in CPU cost reduction since 1990 (a CPU drops in price by a factor of 2 every year) and DRAM (the technology for main memory – a factor of 4 every 3 years until 1996, and 2 every 3 years since then), but the client needs someone to interpret the data, in Figure 1.

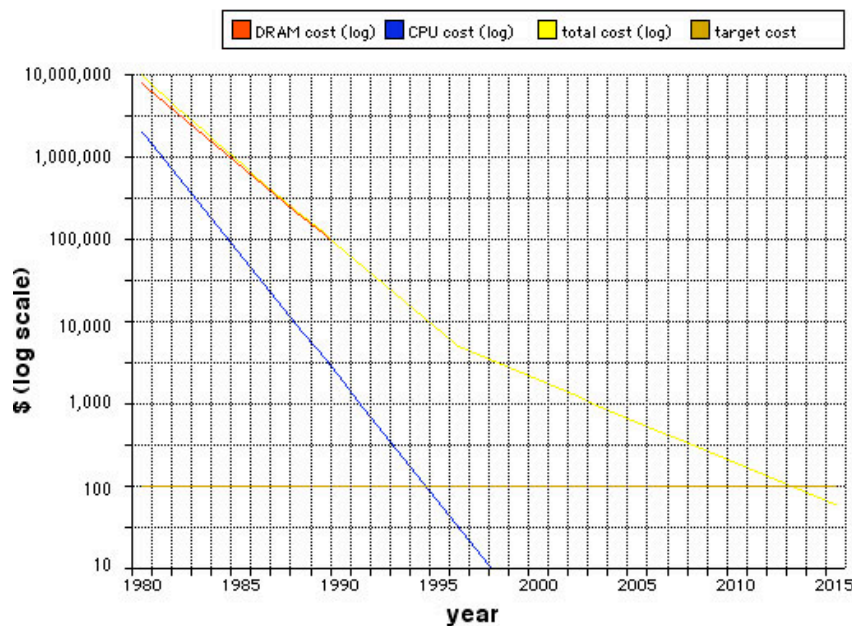


Figure 1. CPU, DRAM cost trends. Roughly, total cost = CPU cost + DRAM cost.

- a. How significant is it that the total cost and DRAM cost lines are almost indistinguishable? What does that mean?
 - b. In what year would the total device cost drop below the target cost of \$100?
 - c. In what year would the CPU cost drop below \$100?
 - d. If you could halve the memory requirement, how much sooner would the target cost be met? How much sooner if you could reduce the memory required to 10% of the original design?
 - e. Given these trends, how would you predict the operating system for a PDA might differ in terms of its resource usage, as compared with a 1980 mainframe operating system?
 - f. If some people have said, "Memory is so cheap, we can treat it as if it's free – efficient memory management strategies of the past are no longer relevant." – are they talking nonsense? Discuss this question, considering the growing market for PDAs, smart devices and embedded computers in the light of the trends in Figure 1.
 - g. By only looking at CPU and DRAM cost trends, have we neglected anything important? What components of a traditional large-scale system are likely to be found in a hand-held device?
2. Many portable devices have real-time requirements, such as multimedia processing, text-to-speech, handwriting recognition and speech recognition. A developer of such devices proposes to use Linux because it's free, and the licensing costs of using a proprietary operating system would be a significant fraction of the cost.
 - a. How many of the following would you classify as *hard* versus *soft* real-time requirements? Explain your answer in each case.
 - i. Continuous speech recognition. Up to 1% of words may be incorrect, requiring user intervention.
 - ii. Handwriting recognition. Up to 5% of characters may be incorrect, requiring user intervention.
 - iii. Text-to-speech. 100% accuracy is required.
 - b. Which of the requirements do you think would be hard to achieve with an operating system like Linux? Explain in each case.
 3. Here are a few ball park numbers, relating to speeds of components of a computer system:
 - a processor completes an instruction, on average, every 0.5ns: call this 1 time unit
 - DRAM (the stuff main memory is made of) takes 50ns for an operation: this is 100 time units
 - a disk takes 5ms to complete an operation: this is 10,000,000 time units
 - a. Every time an I/O operation takes place, at least one disk access takes place. Assume that a minimum disk operation moves 512 bytes, and memory has to be

accessed by the disk to store all the information. Memory is accessed in units of 4 bytes at a time, so this will be 128 memory operations. Then, to recover the information from the disk (now in DRAM), the processor also has to do 128 memory operations.

- i. Without doing a detailed computation, think about the amount of time the processor will wait for disk and memory, if a program does a disk operation only in 1% of the instructions it executes.
 - ii. How can an operating system hide the time you are spending waiting for the disk?
 - iii. How comparable is the time you would spend waiting for DRAM? Is it a comparable problem to waiting for a disk?
 - b. A computer has a hierarchy of memory components. The fastest, smallest, most expensive parts are closest to the processor. Given the numbers in this question, explain why the interface between DRAM and disk is usually managed in software, whereas the interface between DRAM and higher layers is usually managed in hardware.
4. You are planning a large-scale computer system with the following requirements:
- You don't mind if performance is reduced for up to 24 hours, as long as the system is able to continue to run if a part of it is broken
 - You want to maximize performance (in the case where nothing is broken)
 - Running costs are significant as a factor in your overall budget, and you are willing to buy a more expensive solution if it cuts running costs.
- a. Are you looking for a *high availability* or *high reliability* solution? Explain.
 - b. Would a single-processor system be a good fit to the requirements? Explain.
 - c. How would a network of cheap systems compare with a single, large multiprocessor system? Discuss the trade-offs, and how you would apply the given requirements to making a decision.
5. The kernel, ideally, is the smallest part of the operating system which runs without memory protection, and other limitations of applications. The kernel provides a hardware abstraction layer, which hides the machine-specific detail and makes the rest of the operating system (ideally) relatively portable. Which of the following do you think should be part of the kernel? Explain in each case:
- a. A disk driver, which needs to know how the disk is organized internally, and its low-level command set.
 - b. A file system, which organizes information, without knowing how the disk or other devices drivers control each device type.
 - c. A low-level graphics driver, which knows how to address the graphics hardware, and convert programmer-level graphics primitives into hardware-specific commands.
 - d. A window manager, which doesn't know anything about the underlying hardware, but relies on the graphics driver to provide hardware-specific functionality
6. Can you think of any major attribute of traditional large-scale computers which is not today found on personal computers (excluding features not relevant to functionality, like size, and the ability to dim the lights of the nearest city)? What is the difference between the "personal" and "server" editions of mass-market systems like Windows (XP, etc.) and Mac OS X?
7. How do you think a systems programming language might differ from a language only designed for applications? Consider the following features, and comment on whether Java meets each requirement:
- a. Ability to access data at a specific memory location – the kernel usually runs without address translation, and some operating system details require that a programmer address a specific memory location, e.g., to check the status of a device.
 - b. Ability to access code at a specific memory location – an operating system may sometimes place device drivers or interrupt handlers at specific memory locations: you need to be able to treat that memory location as a starting point for code you invoke.
 - c. Ability to control memory allocation and deallocation.
 - d. Ability to ensure that a particular piece of code will execute with low (ideally zero) probability of some high variation in its usual running time.
 - e. Ability to pass a pointer to a specific piece of code so it can be accessed by an operating system routine.
 - f. Ability to express numbers in other bases such as hexadecimal, for convenience of encoding operating system information.