Distributed systems

On this lecture, we will discuss an important area of technology which was not yet covered in this course, that is, networking and **distributed systems**. These allow delegating computing tasks or storing data to external machines.

Networking technology allows combining multiple computers into a single **distributed system** in which all the hardware resources can be used for more complex tasks. The word *distributed* here is intended to distinguish this case from *centralized* systems, in which all exchange of information happens via a single central computer. Yet another alternative to the distributed model is a single *supercomputer*, equipped with extremely powerful components. These approaches are presented on the following diagrams:

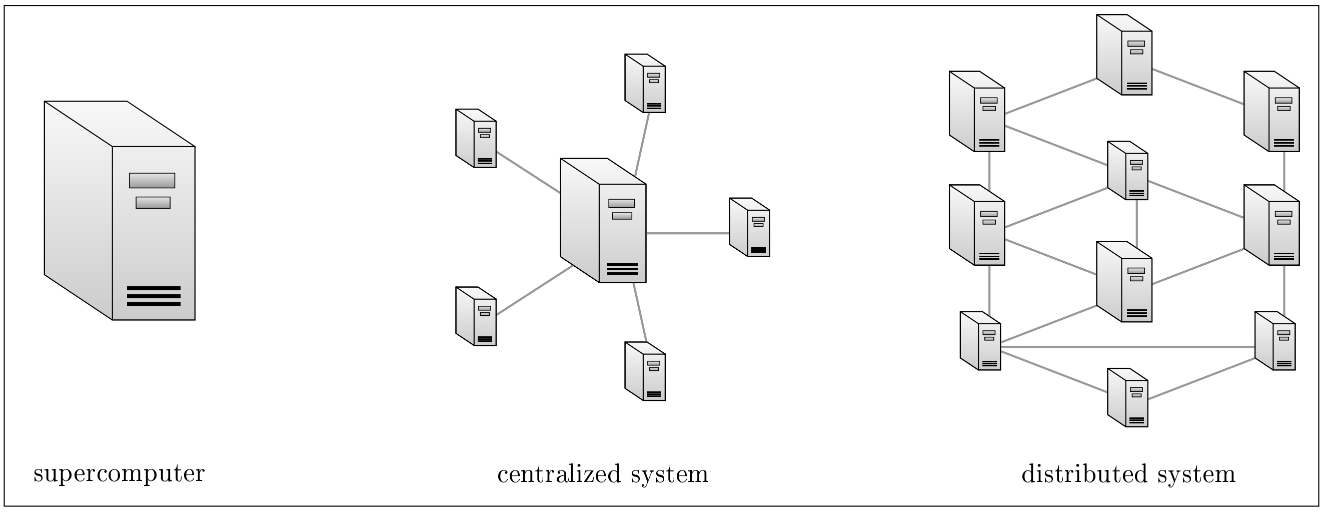

Figure 1. Types of large-scale computer systems.

Pros and cons

The advantage of distributed systems over centralized ones is typically better resilience to a failure of single node, or to network overload. In comparison to supercomputers, the advantages are following:

- increasing computing power at relatively lower costs;
- better resilience to hardware failures;
- quite easy scalability of the system
  (when more computing power is needed, just add new nodes).

On the other hand, a weakness of distrbuted systems compared to supercomputers is limited computing power of a single node, which matters in case of most complex tasks, in which parallelism may be hard to introduce, and longer latencies of transmitting data between threads may be not acceptable.

For these reasons, supercomputers are typically used for specific, most demanding tasks (e.g. complex metheorological models, intensitve training of artificial intelligence), while distributed systems are more often used for parallel execution of multiple, relatively simpler and often mutually independent, tasks.

Another disadvantage of distributed systems is the necessity for managing its complexity; moreover, in a decentralized way. This brings a whole palette of challenges, including:

- handling **asynchronous communication**, in which delivering messages takes a significant and unpredictable time (or can fail altogether);
- **replicating** information across different nodes (to make it available even in case of local hardware failures), and on the other hand, ensuring **consistency** of that information across different replicas;
- administering the system on the software level (e.g. ensuring compatibility between various operating systems, or between various versions of the same program — and/or introducing an appropriate strategy of updates).

These are all very interesting topics, though unfortunately they are outside the scope of this course.

Finally, let us note that many networks used in practice have an "intermediate" structure, that is, they are distributed systems built from *powerful* node machines (which allows efficient execution of more demanding tasks), though of course lying still far behind the top supercomputers. In such cases, when we want to expand such a system, we can choose between two strategies: **horizontal scaling** (i.e. adding new nodes, which will make the system "more distributed") or **vertical scaling** (i.e. improving the efficiency of individual nodes, which goes more in the direction of centralization).

Organzation at software level

We will now discuss a few main types of logical architecture of computer system. Depending on the ownership landscape of nodes and their role, these are:

- **Peer-to-peer architecture** (*p2p*): network nodes have diverse owners (e.g. often they will be simply personal computers of unrelated people) and have all the same role; each node may begin to be served by another node, upon establishing an appropriate bilateral conection between them.

  This model is used by popular *torrent* systems — each node in the network can send a request and start downloading data from any other node; similarly, each node can be sending files. Another example are instant messengers, like Skype.

- **Client-server architecture**: it distinguishes a central *server* (being either a single computer or — increasingly often — a whole computer system), exposing a specific functionality to other (typically numerous) nodes, which are called *clients.*

  The server is responsible for (almost) whole logic related to its functionality; in practice, the cient side may contain an additional user interface.

  For example, although the essential operation while browsing internet is sending requests and receiving responses from HTTP servers, it is also an important functionality to display the received web page in a legible way, and that is the task for a web browser (i.e. *user interface*) installed on the user's computer (i.e. on the *client side*).

In case when the server itself is a whole computer system, there are more approaches to choose from:

- **Cluster architecture**: the server consists of many machines, each of which takes its share of tasks which are similar to each other. In this model, the server itself becomes a computer network, though when speaking of a *cluster*, we typically mean a network of machines located close to each other. (In case of servers distributed to large distances, e.g. to diverse continents, we would typically describe them as a *network of local clusters*). Anyway, in all cases, it becomes important to effectively (which usually means: evenly) handle spread the work among the nodes, which is the responsibility of a *load balancer*.
- **Multi-layer architecture**: here, the functionality of the server is split into separate *modules* (or *layers*) assigned to different machines. In the simplest case, the server is split into a database and a web server communicating with the client. However, one can also distinguish a larger number of specialized layers (for example: authorization layer, cache, task scheduler).

In practice, naturally, we can meet mixed architectures, e.g. a multi-layer server in which the layer dealing with the main computations (and sometimes also other layers) will be a distributed system of a cluster architecture.

Let us also mention the **grid architecture** in which a distributed system is built of diverse machines, often belonging to various owners, which agree to dedicate some computing resources to a shared goal. Modern examples include volunteering projects for specific intensive computations (e.g. *Folding@home*, a network performing bioinformatical simulations for the purpose of fighting certain class of diseases); as another example, we can consider the block-chain networks for verifying cryptocurrencies (e.g. Bitcoin).

In the context of our classifications above, grid architectures do not allow simple labeling as a whole class; their classification depends on the particular case. Verifying cryptocurencies happens typically in a decentralized peer-to-peer network. In contrast, a typical *citizen science* project may be regarded on one hand as a single distributed *server*, whose service (computation) is used e.g. by the scientists at the root of the project; on the other hand, the individual machines involved in those computations are commonly described as *clients* which connect to a *server* (or *servers*) in order to receive subsequent shares from the central pool of tasks. (In this way — quite unusually — it is the *server* who requests work from the *client*).

Services in the cloud

In the modern distributed systems, special focus is usually put on:

- a specific **service** delivered by the system (for example: storing user's files, providing videos on demand, providing an environment for the customer's web app operation);
- ease of use, understood as hiding from the customers the whole technical layer and leaving them just with a well-defined **user interface**, through which the service is used;
- high **availability**, understood as minimizing the time of the service being inactive (due to any failures), but also minimizing the waiting time for connecting to the server and having the task done (which typically is achieved by distributing the servers over the world regions).

Such philosophy of building computer systems has a common name of **service-oriented architecture**. In the context of distributed servers of a global scope (i.e. possibly close to all users), and an implementation hidden behind an abstract interface, this technology is commonly described as the **cloud**.

Below, you can see a graph illustrating the main providers of cloud solutions. As it shows, in the case of cloud, we typically observe a single owner controlling a whole ecosystem of services and managing that whole system in a uniform way. The service delivered by the cloud provider is usually also much more extensive than in the case of grid systems.

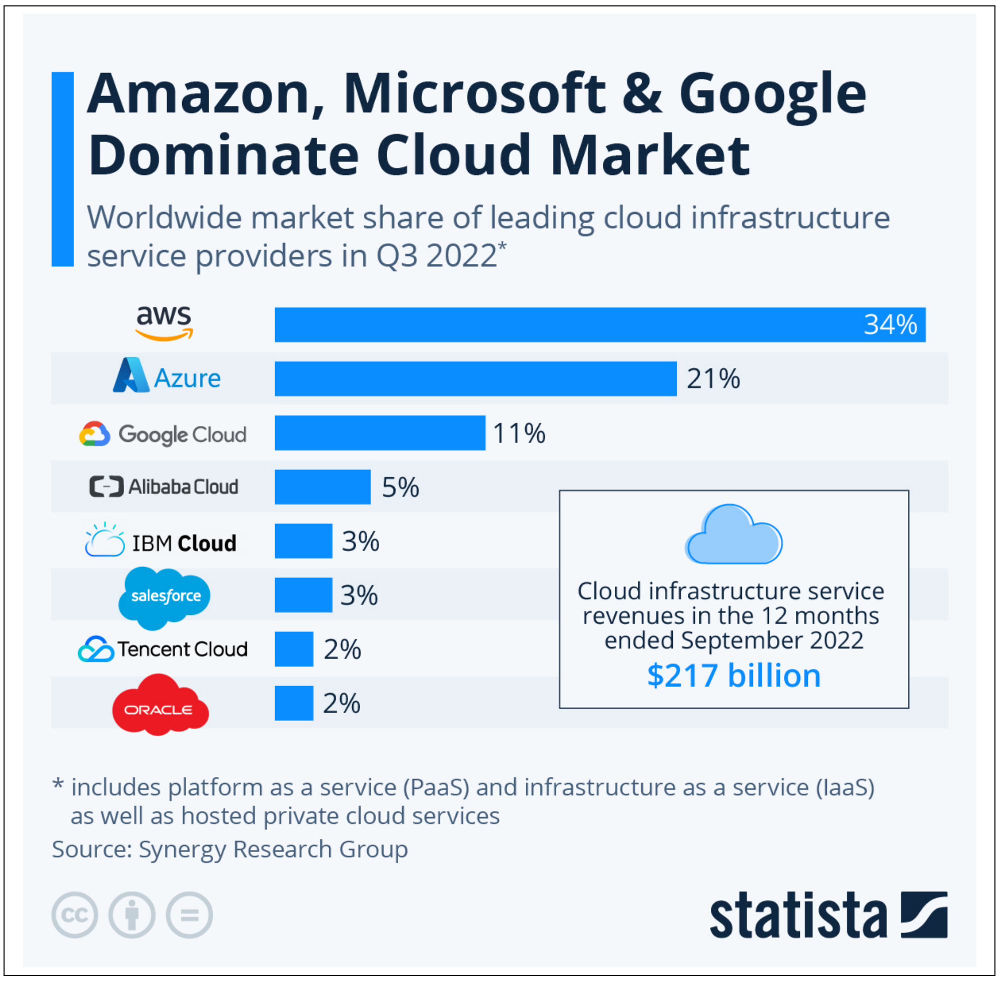

Figure 2. The leading providers of cloud services.

On the level of technical network structure, as a cloud system typically involves lots of users with independent computing tasks, cloud servers are almost always distributed systems; more specifically, clusters or distributed networks of clusters.

Levels of service

Considering the type of delivered services, we can distinguish 3 main models of cloud systems:

- **IaaS** (*infrastructure as a service*): here, the provider offers infrastructure, that is, access to servers together with maintaining them in an operational state, but no accompanying software.

  Physically, the computers could be located at the customer's headquarters, or even be customer's property. Typically, however, the customer remotely uses hardware belonging to the provider. In the latter case, it's a frequent practice to place customer's processes on *virtual* *machines* (special programs emulating a given computer model and an operating system) — which allows e.g. enforcing **isolation** between processes from different customers running on a single physical machine (as it will now suffice to run those processes on separate virtual machines).

- **PaaS** (*platform as a service*): here, in addition, the provider offers an *environment*, meaning mostly software related to the servers, network, databases. That is typically used by developers for building web applications, abstracting from technical details of the underlying hardware.

  This layer is also traditionally considered to contain utils oriented at software engineering: frameworks for automated testing, deployment, monitoring the quality of service as perceived by end user, etc.

- **SaaS** (*software as a service*): here, the provider offers specific applications used by the users for particular purposes. An example of this is Dropbox, allowing for uploading, viewing and downloading files via a web browser (thus creating a disk space "in the cloud").

As individual private users, we often use cloud solutions even without realizing that. This happens whenever we send an e-mail, listen to music from the internet, play internet games, or watch movies.

The picture becomes somewhat different when a cloud system is used by a company. Here, delegating some tasks to the cloud provider helps in areas such as:

- data analysis (regarding e.g. customer choices, application performance, etc.);
- delivering the product to end users;
- ensuring scalability of hardware
  (upwards, but also downwards — to avoid being overcharged for unused hardware resources);
- creating and maintaining data backups;
- networking security issues;
- testing the product and its new versions;
- applying the *dev-ops* culture — that is, a culture of cooperation between the developer teams and the product teams reponsible for product health and maintenance;
- applying the *agile programming* concept, which assumes e.g. frequently confronting the product with the actual users' needs, and frequently adjusting the project.

The concept of cloud is also related to the notion of **virtualization**, though we're now considering this word in a much broader sense than just *RAM virtualization* which we saw in an earlier lecture. Generally, *virtualizing* means emulating some resources by using other ones; often, by uing appropriate software. In the case of cloud solutions, this is typically implemented by the abovementioned *virtual operating system*, which ensures isolation of a single customer's processes, and may also provide a *remote desktop*, which lets the user connect to the remote server ("in the cloud") and use the emulated desktop just as if it was a part of a standard operating system. *Network virtualization* may also take place; this means splitting the capacity of physically existing network connections into independent virtual channels.

Graphics

On this lecture, we will look at the most important input/output devices which have not been discussed so far: graphics cards, screens, and printers. (Of course, there are also many other kinds of input/output devices which we frequently use, but we will restrict our discussion to these kinds).

Graphics cards

A **graphics card**, as the name suggests, is responsible for processing the images displayed on the screen. That is not an easy task. On the lecture "Encoding data", we described how a computer stores and operates on integers, floats and texts. Working with images also requires encoding them in a binary form. The easiest way would be to encode the color of each pixel separately (as it happens e.g. in BMP files); however, this method would be very inefficient computationally. (The importance of efficiency here is well known by every computer gamer — it is the graphics card computing power which often decides if the game will operate smoothly or *lag*). Therefore, the standard data representations used in graphics cards are more sophisticated.

Types of graphics cards

While a graphics card can be found in almost every computer, it can take one of the following two forms:

- an **integrated** graphics card: the word *integrated* represents the fact that it's a part of the processor;
- a **dedicated** graphics card: a separate module, usually exchanging data via a PCI Express bus (discussed on the previous lectures).

The advantages of an integrated card are:

- lower price;
- lower power consumption, which in case of laptops leads to:
  - smaller weight;
  - longer battery running time.

On the other hand, a dedicated graphics card offers greater computing power. It also leaves an option to be later replaced by a better model.

Hardware properties

The most important component of a graphics card is the **graphics processing unit** (**GPU**). In colloquial speech, the phrases *GPU* and *graphics card* happen to be used interchangeably. A GPU shows many resemblances to the computer CPU which we have already met, as both units play a similar role: just like CPU, GPU is responsible for performing computations. However, CPU is a general-purpose unit, fit for a variety of tasks, including those of serial nature. Every single core of a CPU can execute tasks on its own, serving as a stand-alone processing unit. On the other hand, the GPU architecture is optimized for highly **parallel** and **vector** computations, performed simultaneously on many cores. As a result, a GPU will typically contain many more cores than a CPU, often going into hundreds or even thousands.

Just like CPU, a graphics card needs to use memory. Dedicated cards are equipped with their own (separate) hardware die, whose size may currently reach even 24 GB, though typically is below 10 GB. For moderate applications, 2-4 GB should suffice. Graphics cards integrated with the CPU do not have such dedicated memory chips; instead, they are given some area of the main RAM memory by the processor, of size depending on hardware settings, and sometimes also on user preferences. For some cards, an option to control their memory size will be available through their drivers; such parameter can be also managed in the UEFI/BIOS settings.

Applications

Although the main task of graphics cards is graphics imaging, they have also found applications in other areas requiring heavy computations which admit intensive parallelization. These include implmentations of so-called *deep learning* or artificial intelligence — in particular when the data being handled are very large.

In the recent years, graphics cards have found a new important application in the so-called *mining* of crypto-currencies. Sadly, this has led to a substantial increase in graphics cards prices, and had a role in increasing the global consumption of electric power. Also, we observe an increased number of security attacks on computers, after which hackers use the overtaken machines as crypto-currency "mines".

Dedicated graphics cards also allow working with more connected monitors than 2, which is a frequent limit for integrated cards.

Screens

We will now consider the devices for displaying images, which may be called **screens**, **displays** or **monitors**. (All these terms largely overlap, though a *monitor* sounds more like a stand-alone device, not including e.g. the display/screen of a smartphone). Before describing the main technologies of manufacturing them, let us discuss some aspects common for all of them.

Pixels and the RGB space

In nearly all modern displays, the image is built from **pixels** — "points" (or, more strictly, tiny dots) of color — typically laid out in a rectangular array.

A single pixel is usually capable of showing any color of the device-supported color space — which is enabled by building it from three **sub-pixels**: a red, green and blue one. These sub-pixels are *monochromatic* (i.e. they shine with varying intensity, but in a fixed color).

That choice of sub-pixel colors results from law of optics regarding mixing colors of light (which, in turn, are based on the anatomy of human eye — which, in most individuals, employs a color recognition system based on 3 types of *cone cells*), as the following diagram shows:

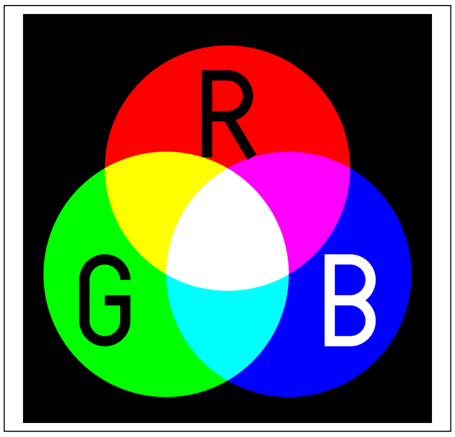

Figure 1. The diagram of mixing colors of light. (Source: `http://en.wikipedia.org`)
The above image can be obtained by placing a white screen in a fully dark room and then using three colored torches (a red, green and blue one) to illuminate three partially overlapping areas. As we see, all three primary colors combined together give the white light; combininig pairs produces yet other colors.

Generally, it turns out that *almost every* color recognizable by human eye (at least, for most people) can be obtained as a combination of lights in the 3 primary colors, just by appropriate adjustment of their intensities. This fact is the basis of the **RGB** (*Red-Green-Blue*) color system, used in screens to display various colors, and more generally, in computers to describe colors. In the most popular version, the intensity of each of the primary lights is described by an integer between 0 (none) and 255 (maximal intensity), producing a space of $256^3 \simeq 16.7$ million of possible colors. For example, the triple `(150, 30, 50)` will denote the color 🟥 . (Also, hexadecimal notation is commonly used here: for our example, that would be `961E32`).

One of the main screen properties from the user viewpoint is its **resolution**, which means the number of pixels displayed in the current operating mode. For example, *Full HD* resolution means a rectangular array of 1920×1080 pixels. Modern computer screens can typically operate either in their **native resolution** (the one they have been designed for) or in a *lower* one (i.e. using less pixels) — which require scaling the image and may negatively affect its quality.

Manufacturing technologies

CRT

The generation of **CRT screens** has essentially gone out of fashion. In those models, the key role is played by three electron guns emitting beams of electrons which hit a layer of *phosphor* (which means: some material which is *luminescent*, i.e. can emit light when activated by another kind of energy; not to be confused by the chemical element of *phosphorus*). The exact place where the electron beam will hit the screen is controlled by a deflecting element (using a magnetic field). Each of the electron beams periodically "runs through" all the pixels on the screen; the displayed image is stable thanks to the fact that the phosphor continues to emit light for a short time after it has been activated.

Using a **shadow mask** — a metal barrier with a grid of round holes (corresponding to pixels), placed at some distance from the screen — allows a better control of the target of the electron beams. This is even more important, given that different points of the screen are covered with phosphor shining in different RGB colors; the placement of electron guns and shadow mask holes is chosen so that e.g. the “red-light” gun (the one responsible for red sub-pixels) can hit only those points of the screen which are covered with the red-glowing phosphor.

The CRT screens were replaced by a newer technology (LCD) due to their following disadvantages:

- high power consumption;
- large size;
- eye fatigue (due to radiation);
- poor brightness and contrast;
- issues with image geometry.

LCD

The **liquid-crystal displays** (**LCD**) have no electron gun, which enabled substantially reducing their mass and changing their shape to nearly flat. Light emitted by a flat source (e.g. a system of gas-discharge CCFL lamps) falls on a polarizing filter, which lets through only light waves of specific orientation. Then, there comes a layer of liquid-crystal molecules, in which by using transistors we can turn on or off the individual pixels. (Technically, this happens as follows: by applying different voltage levels, we manipulate the pattern formed by these molecules, which impacts the polarization of light after traversing the liquid-crystal layer; that in turn determines whether this light will be blocked by yet another polarizing layer). Finally, single sub-pixels contain color filters which gives the light its desired color (red, green or blue).

There are a few technologies of managing the patterns of crystals; they influence e.g. suitable viewing angles for the screens.

There is a newer generation of liquid-crystal displays, in which the illuminating CCFL lamps are replaced with light-emitting diodes (LED). (However, their light is still processed by the liquid-crystal layer, polarizers, and often by color filters). Such devices are often called **LED displays** (though it would be more precise to call them *LED-backlit LCD displays*, to avoid confusion with “proper LED displays” in which the LED diodes directly constitute the subpixels). The advantages of the LED backlight, comparing to CCFL, are:

- better contrast (enabled by smaller size of diodes, which allows dimming them selectively in the darker areas of the image);
- better viewing angles.

The disadvantage, on the other hand, is the price, higher than for the traditional LCD displays.

Although the LCD/LED displays are usually considered better than CRT — thanks to removing the CRT disadvantages mentioned above — it’s worth knowing that they also have some disadvantages on their own:

- worse representation of colors;
- worse image quality when viewing from sharp angles;

- longer response time (except best LED models);
- worse quality for non-native resolutions.

Other technologies

Let us now describe some other technologies for producing screens, which are (at least currently) less frequently used due to their price.

In the **plasma screens**, light is generated by ionizing gas, which — when brought to a plasma state — begins to emit ultraviolet light. Every subpixel is a chamber for such gas, filled with an appropriate phosphor, which upon activation by UV light begins to shine in the desired primary color.

The **OLED** screens and their improved versions (AMOLED, SUPER AMOLOED) are examples of "true LED displays", i.e. the role of subpixels is played directly by diodes of various colors. This means that we no longer need an additional "backlight" which needs to be polarized. The differences between these technologies come from different kinds of diodes. The AMOLED and SUPER AMOLED displays, due to their price, are so far used almost exclusively in small mobile devices (smartphones, tablets).

The table below shows disadvantages and advantages of plasma and OLED technologies in comparison with LCD screens.

Technology	Advantages	Disadvantages
	(in comparison with LCD)	
Plasma	• thinner • better viewing angles • better contrast • longer lifetime • better mechanical resistance	• heavier • higher power consumption • uneven usage of phosphor • image blinking • noise — noticeable on higher altitudes
OLED	• better viewing angles • better contrast (and color space) • shorter refresh time • no mercury used	• higher power consumption • more fragile • shorter lifetime • development restricted by patent rights

Ports

A screen is typically connected to a modern computer by one of the following ports:

- USB — a standard already discussed on the lecture "Non-volatile storage".
- **HDMI** — a standard for transmitting audio & video signal (also in high quality, without compression). It has throughput up to 48 Gb/s. It also allows using longer cables (which does not just mean using more metal — but also increased chance of data corruption which needs to be detected and fixed).

- **DisplayPort** — mainly for connecting the computer to a screen or a home theater system. It allows bidirectional transfer up to 80 Gb/s. The transmitted signal may be protected from copying (with the DPCP technology) or from usage unintended by the copyright owner (with the DRM technology).

Printers

Printer colors: the CMYK space

Similarly as for screens, the printing technology is based on colored **pixels**, whose color is obtained by combining various intensities of selected primary colors. The difference is in the choice of those primary colors: since the printing inks behave more like a filter than a source of light, mixing their colors acts somewhat conversely to mixing lights, leading to *darker* results. Due to this fact, it is the standard in modern poligraphy to use the **CMY** (*Cyan-Magenta-Yellow*)) palette, which differs from the RGB set by a swap of roles between the primary and secondary colors (compare Figure 1 with Figure 2).[1]

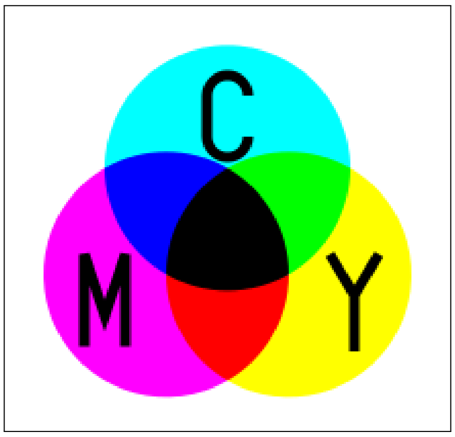

Figure 2. The diagram of mixing colors of printing ink. (Source: `http://en.wikipedia.org`)
The above image can be obtained by putting on a white sheet of paper simultaneously three circles of printing ink (a cyan, magenta and yellow one). As we see, the primary CMY colors coincide with the secondary light colors in the RGB model (see Figure 1), and conversely: combining pairs of CMY inks leads to RGB primary colors. In practice, however, this correspondence will not be perfect, due to technological differences between ink and diode producers.

In a perfect world, a just combination of cyan, magenta and yellow would be black; however, in practice, such "blackness" turns into a bit dirty brown. Moreover, a more pure black ink can be produced much cheaper than by mixing 3 colored ones. Therefore, the system is expanded by introducing a black inkc component, denoted by **K** (*key*), producing altogether the **CMYK** model. In it, each color is described with 4 numbers, specifying the intensity of the component colors, in percent (from 0 to 100). Traditionally, printing happens first with the black ink, then with magenta, cyan, and finally yellow. The important aspect here is that every ink layer should have high transparency.

[1]In the classical art theory, it is popular to consider a color system with a different set of primary colors, described now in short as RYB (*Red-Yellow-Blue*). That one has the advantage of being more intuitive (more tied to the culturally "important" palette). Technically speaking, the "quality" of such system (i.e. how well it covers the space of colors recognizable by human eye) can depend on the subtleties of choice of the primary colors, as well as on the physical properties of the inks/paints which are mixed. The CMY model is now considered optimal in this regard, and its additional advantage is its natural connection to the RGB system of light colors.

Printer types

We distinguish three main types of printers:

- **Dot matrix** printers: in these, paper moves under an ink ribbon which is pushed down by a needle to contact the paper in specific selected places. This somewhat resembles the mode of operation of all printing machines. This type of printers has fallen out of broad use, though it still has some usage niches. For example, it enables printing a few copies at once (through carbon paper), which together with its low price makes it an attractive choice for printing invoices.
- **Ink** printers: these operate by placing on the paper drops of ink, typically in primary CMYK colors. While the devices themselves are relatively cheap, their maintenance costs (i.e. the cost of ink) is high.
- **Laser** printers: these paint the paper by transferring to it a colorizing powder (**toner**) from an electrified roller. A laser beam is used to discharge specific locations of the roller, which makes them no longer attract the toner. These devices are more expensive than ink printers; on the other hand, their running costs are lower.

Connectivity

Printers may communicate with the computer e.g. via a USB cable (see the lecture "Non-volatile storage"); however, increasingly more models also support **wireless** communication. In the latter case, the following protocols may be used:

- **Bluetooth**[2] uses radio waves for transmitting data between two devices which have been *paired* together. The transmission requires little electric power, which leads to a disadvantage of relatively short communication range (up to 10 meters for the most common class of devices). The newest versions of Bluetooth standard achieve the transmission rate of up to 50 Mb/s.
- **WiFi** also uses radio waves. The main differences comparing to Bluetooth are:
  - larger range (up to ca. 90 metres);
  - higher power consumption;
  - faster transmission (though still significantly lower than for the fastest wired solutions);
  - no concept of "pairing": once a device is connected to a *local network*, it can essentially communicate with any other device in that network, though this requires an intermediate device called a **router**.

Computers and the electric power

To close the lecture, we will mention two issues related to using electrict power in computers.

[2]Fun fact: the name *Bluetooth* refers to the king Harald Bluetooth which has (for a short time) unified the Scandinavian Viking lands under his reign. Analogously, it was the intent of the Bluetooth creator (who was, by the way, Swedish) to create a protocol "unifying" various kinds of input/output devices. Even the Bluetooth symbol is a combination of Nordic runes corresponding to letters H and B (king Harald's "initials").

Failover supply

The first such issue is a threat of a **power outage**. Interrupting operation is unacceptable in case of critical infrastructure (e.g. in hospitals) but also in home conditions it may be strongly undesired or even pose a risk of damaging the hardware (e.g. a sudden power outage during operation of an SSD drive). The solution is to use a **failover** (or *uninterruptible*) **power supply**, known in short as **UPS**.

Besides keeping electricty on during an external power outage, UPS devices can also protect the hardware connected to them from other disruptions like sudden changes of network voltage. This leads to 3 main kinds of such devices:

- *Online*: separates the protected device completely from the external network. This means that the receiver is no longer threatened either by outages or voltage disruptions. However, this variant involves greater energy losses, more (acoustic) noise, and a shorter lifetime and higher price of the UPS device.
- *Offline*: the receiver device consumes electricity directly from the outer network, and UPS batteries are only charged when necessary. This means minimal energy losses and lack of noise in normal conditions. A switch to failover electricity occurs at the moment of external power outage; however, this transition has a duration of about 10 milliseconds. Also, this type does not protect against unexpected changes in voltage levels.
- *Line-interactive* and *line-interactive AVR*: improved variants of the offline solution, involving (to some extend) stabilizing the input voltage level. The reaction time in case of outage is also shorter than in offline UPS, being about 5 ms.

Cooling

Another issue related to electricity is the heat emitted during computer operation, threatening with overheating its components and damaging them. This means that stable operation of a computer requires an efficient **cooling** mechanism.

The components with largest power consumption are processors and graphics cards; that's why they are equipped with cooling systems. Typically these contain:

- a **radiator** — a comb-shaped system of metal plates which efficiently transfer heat from the processor or graphics card to the air located around these components;
- a **fan** which, by pushing the air into circulation, facilitates transmitting heat further away.

The rotation speed of the fan depends on the current temperature. In case when the fan is already rotating at its full speed, and the temperature nevertheless keeps growing, the computer (at least assuming a valid configuration) will automatically reduce its efficiency, or in extreme case switch to suspend mode or even to a full shutdown. The exact temperature thresholds can be changed in UEFI/BIOS settings — though one should remember that raising them too high brings a threat of damaging the hardware. This scheme is called **active cooling**, as opposed to **passive cooling** which involves just a radiator (without a fan). Passive cooling is practically never used in modern computers.

Yet another solution is **liquid cooling**, which involves setting up a circulation of some liquid (e.g. purified water) in a close loop, alternatingly close to the hottest computer components and much cooler masses of air. In this scheme, water takes away the heat from the computer components, and then gives it back to the air. The advantages of this approach are reduction of noise and better cooling efficiency. The disadvantages are the increased complexity and volume of the cooling system.