Orchestration: The need for System and Language Abstractions

Stuart Clayman,
Dept. of Electronic Engineering
University College London
`s.clayman@ucl.ac.uk`

26th IEEE SIGNAL PROCESSING AND COMMUNICATIONS APPLICATIONS CONFERENCE

MAY 2-5, 2018 / Çeşme - İzmir - Turkey

Introduction

- This talk is focussed towards people in the arenas of orchestration and higher level management, where they are attempting the role out of SDN, NFV, and SFC.
- It is a talk with no measurements and no experiments !
- It is not really about right / wrong, more that there are opportunities to use working and well tested concepts.
- This is about a perspective from the viewpoint of operating systems and programming languages.
- We need to encourage people to design / built / utilize more in the area of abstractions, layering, and separation of concerns, by showing the successes in other areas.

Background

- This work came about from discussions with networking people, telecoms operators, DevOps, in recent EU projects, in networking conferences, and the IETF, who mentioned how difficult it was to interact with the complex system they had, and how difficult it was to deploy a new service.
- There is a 5G goal to reduce deployment from 90 days to 90 minutes. Recent techniques being targeted are: *programmability*, *machine learning / machine intelligence*, and *intent*.
- My observation, mentioned to them, was that there were not enough composable abstractions, nor programable elements to support the run-run dynamics.

Background

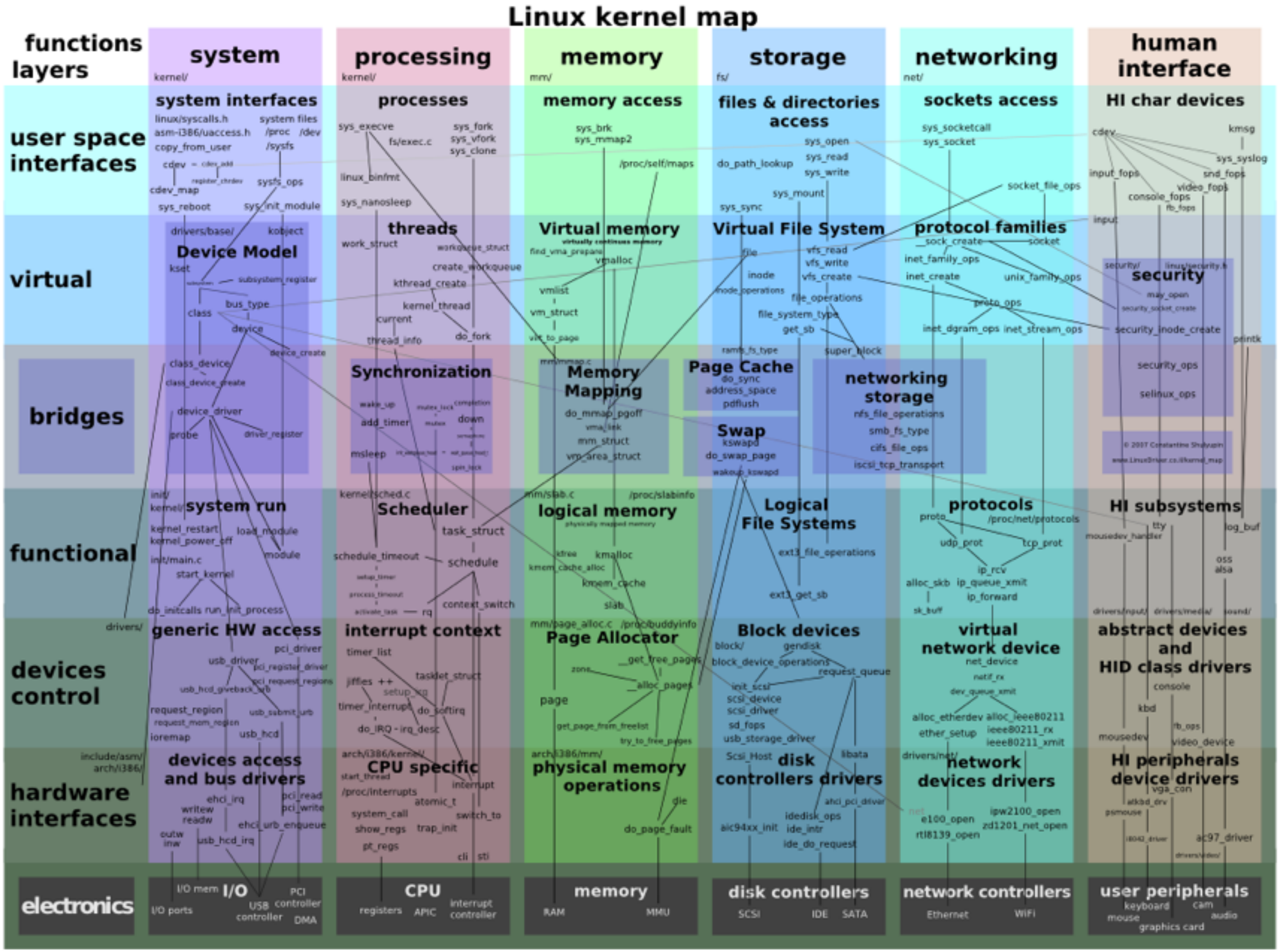
- From these discussions, there is a general feeling that abstractions at the management level will hinder the operation of the network, by hiding the relevant details, or will slow down the interactions with the devices that need to be managed.
- At the same time there is an acceptance that managing a network, especially with VNFs is a very complex task due to the number of resources / devices and their different operational behaviours, and the number of services that are running over the network.
- The question often arises:

  "*How is it possible to manage all of these diverse elements and functions in a better way*? "

High-level Abstractions

- The abstractions that appear in operating systems hide the underlying features, operations, and interfaces of the hardware, presenting elements to users, programmers and system managers that are easier to understand and easier to interact with.
- The operations on the abstracted elements are ***queued***, ***mapped***, ***processed*** and ***multiplexed***, through various layers, into control and data requests for the devices.
- All of these techniques are becoming far more important for Net Man as the environments of the cloud and the network are becoming more coupled, using virtualization.
- All the elements need to be managed as one, so we need to gain insight into such mechanisms.

Layering and Abstractions



Layering and Abstractions

- We see far too many one-layer SDN controllers !!
- Operating Systems have many abstractions over the devices in the machine and the controller for the devices.
- Many of these were originally devised in the 1960s and 1970s, so there is a lot of experience as to what function to put in what location.
- These we consider in a bit more detail:
  - system / processing / memory → Processes
  - storage → File System
  - networking → Networking

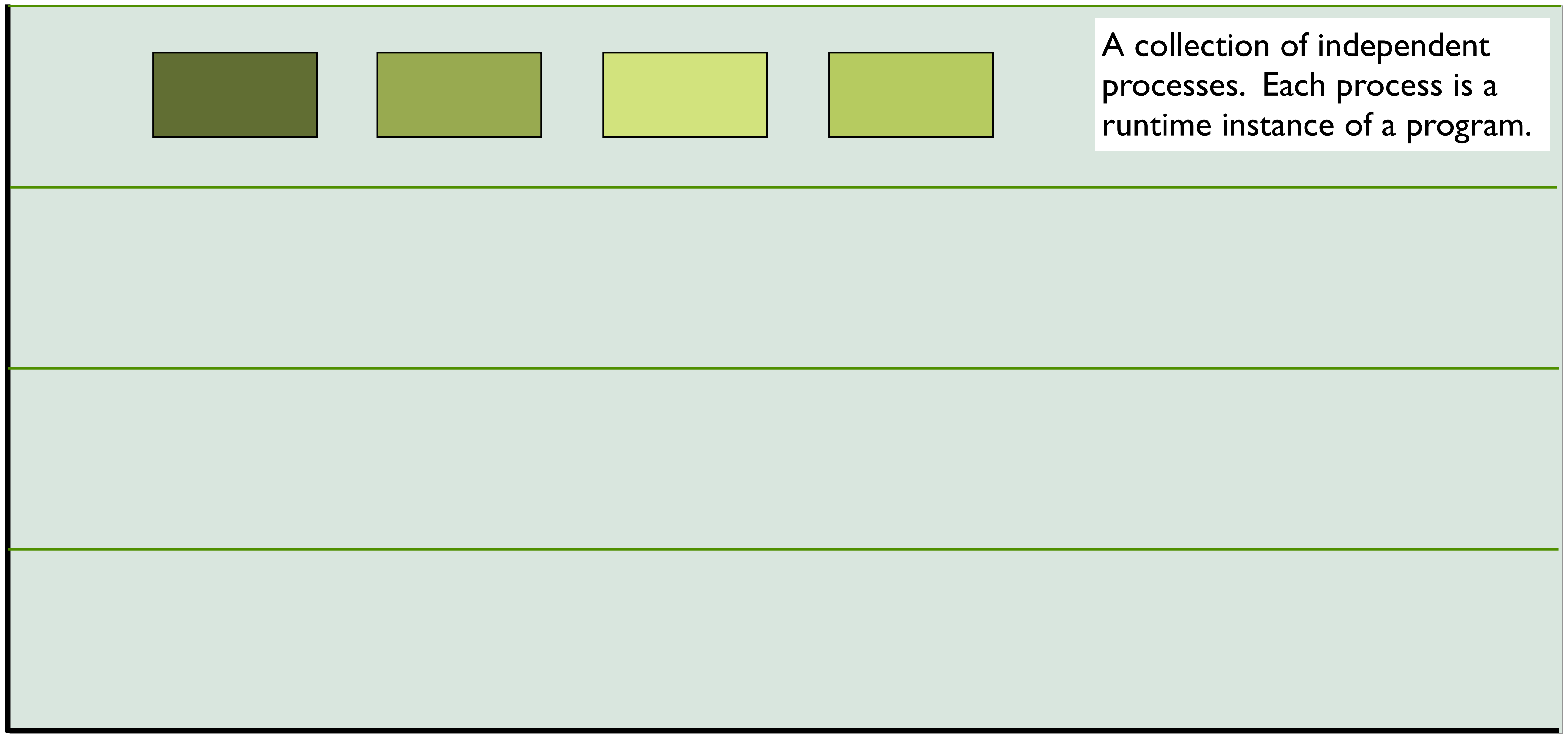
Processes

- A process is a manifestation of a program that executes on the computer. It is independent of other processes, but can interact with other processes.
- The operating system allocates resources to the process, such as memory and cpu time.
- The process is an abstraction, independent of the hardware. The process can be considered without knowing anything about the physical resources of the computer, how many other processes there are, or what state the OS thinks the process is in.
- All of this in handled *automatically* by the *process scheduler.*

Processes

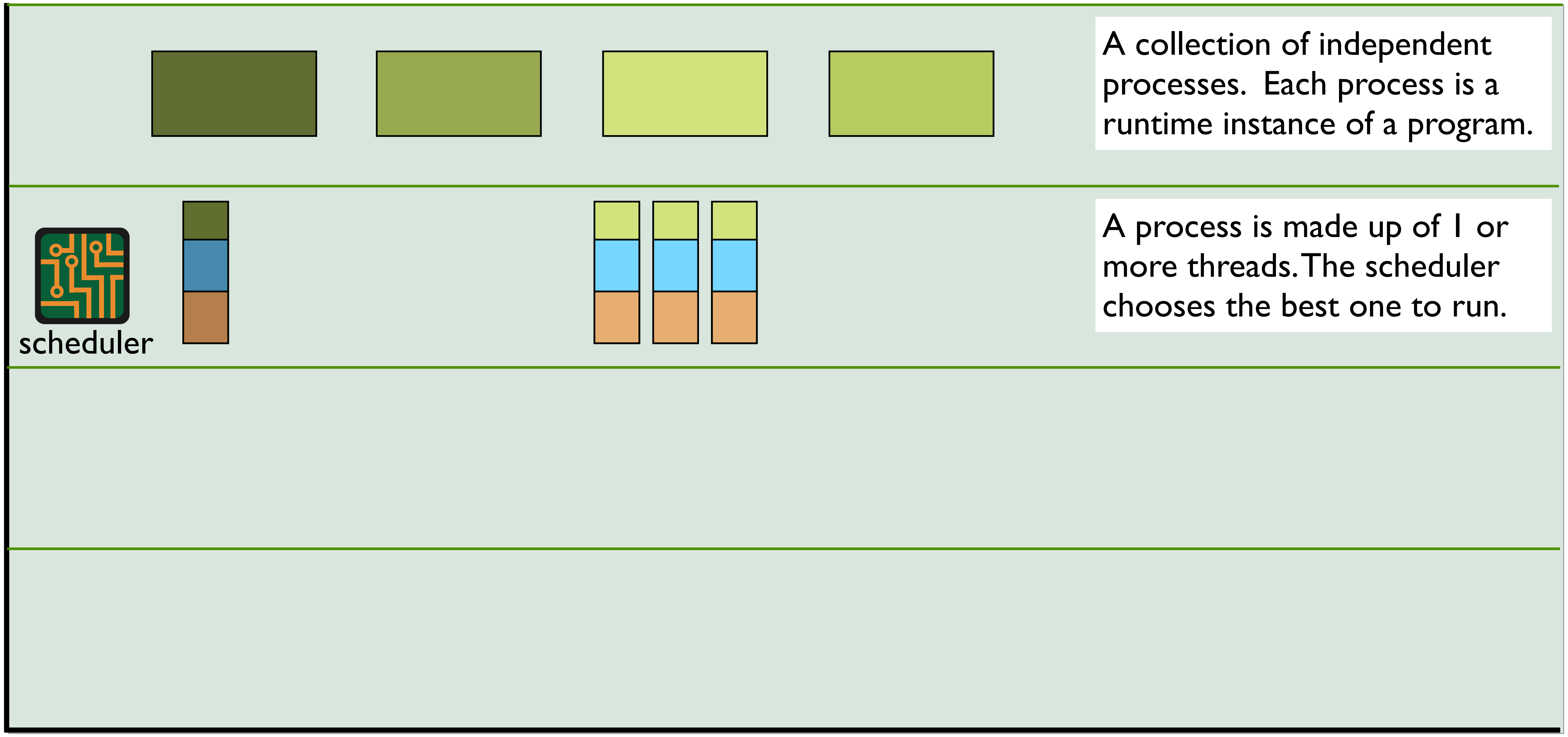
- This scheduler decides which process to execute next. The operating system schedules each process depending on whether it is suitable to execute and whether is should be allocated some CPU time.
- Using this method, the OS can reliably execute thousands of processes concurrently. From the human perspective they execute at the same time → ***time sharing***.
- Memory allocation per process, and for all of the processes in the system, is not fixed to the maximum size of physical memory. It is done dynamically, using virtual memory, using an over-provisioning strategy.
- Virtual memory and memory management was solved in 1960s.

Processes



A collection of independent processes. Each process is a runtime instance of a program.

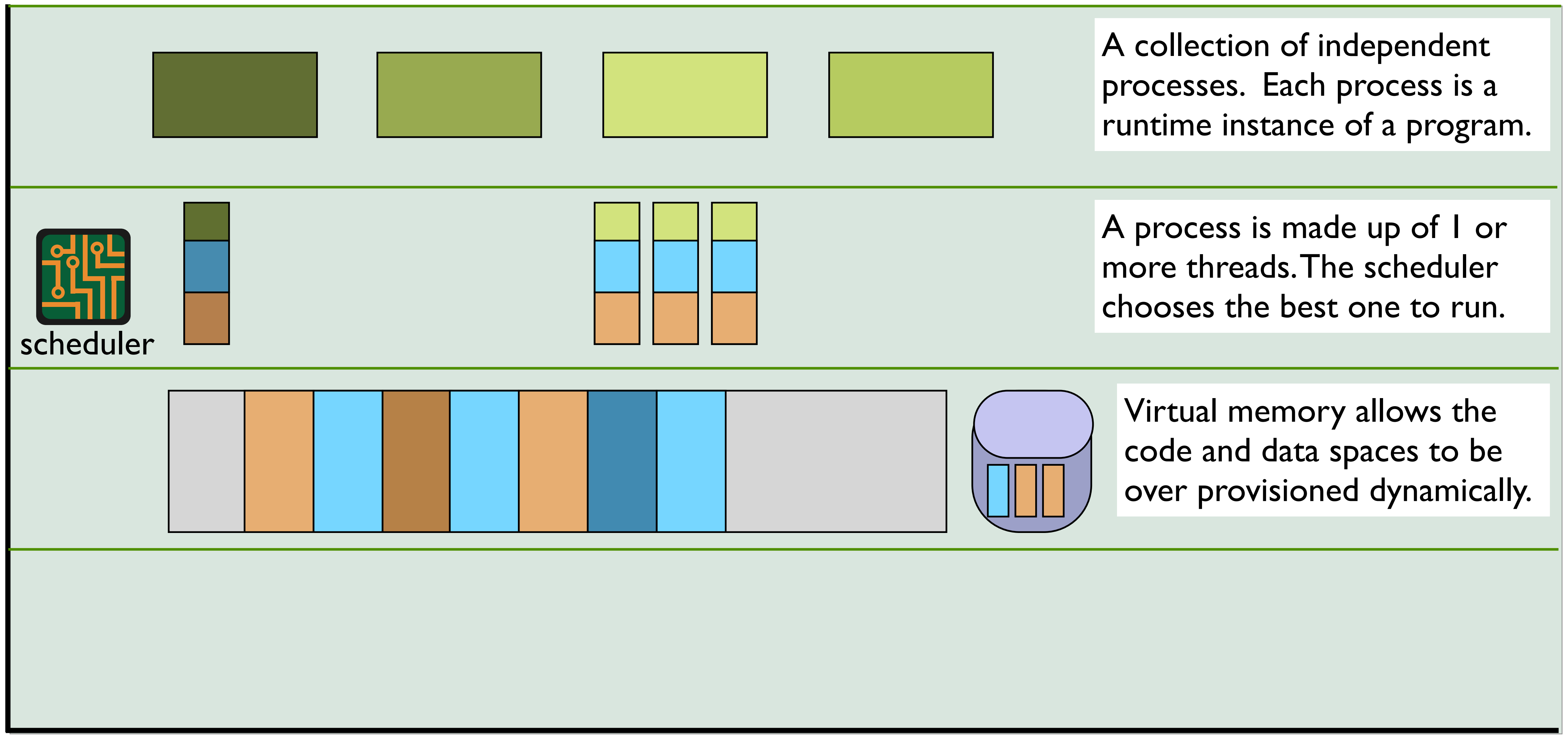
Processes



A collection of independent processes. Each process is a runtime instance of a program.

A process is made up of 1 or more threads. The scheduler chooses the best one to run.

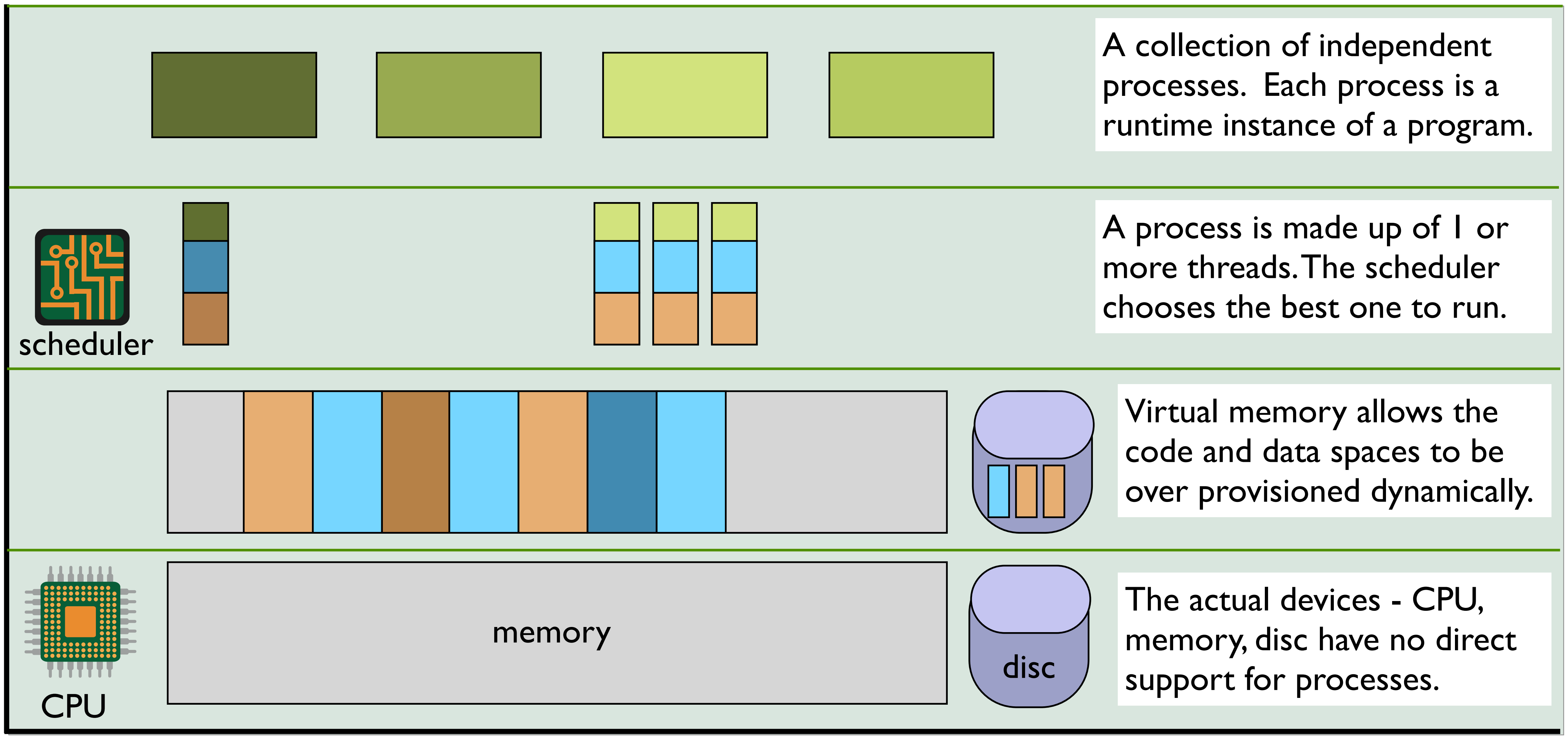
Processes


A collection of independent processes. Each process is a runtime instance of a program.

A process is made up of 1 or more threads. The scheduler chooses the best one to run.

Virtual memory allows the code and data spaces to be over provisioned dynamically.

Processes



A collection of independent processes. Each process is a runtime instance of a program.

A process is made up of 1 or more threads. The scheduler chooses the best one to run.

Virtual memory allows the code and data spaces to be over provisioned dynamically.

The actual devices - CPU, memory, disc have no direct support for processes.

File System

- The abstraction that is exposed by the OS and the end-user sees is the file and the directory. Each of these files & directories is mapped to specific underlying file systems. The file system will then be mapped to blocks, and the blocks will be placed on the disc.
- The disc drive device has no concept of files or directories or file systems.
- There are mainly 3 layers of abstraction:

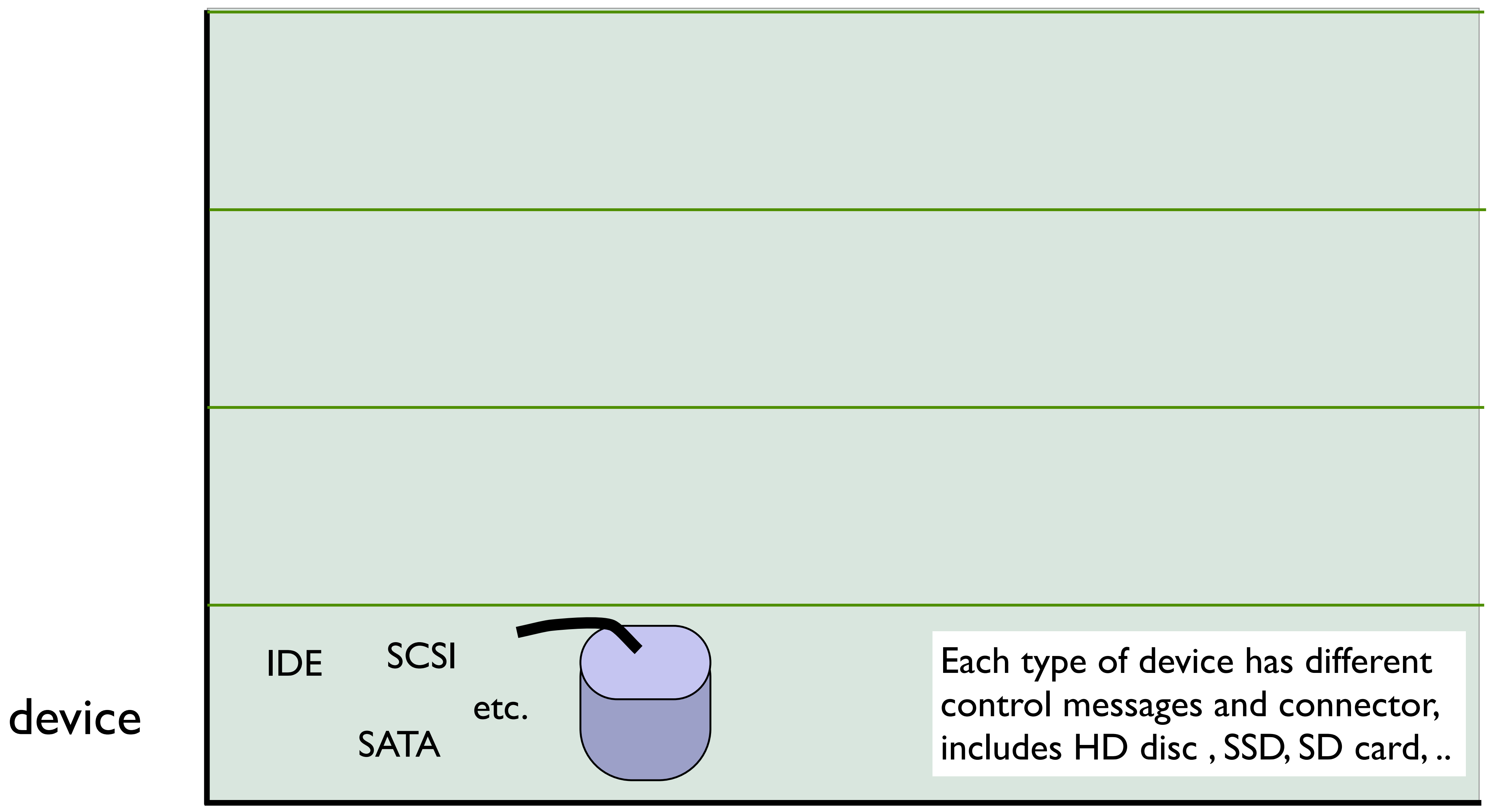
  (i) device → disc block

  (ii) disc block → FS format (logical file system)

  (iii) FS format → generic tree (virtual file system)

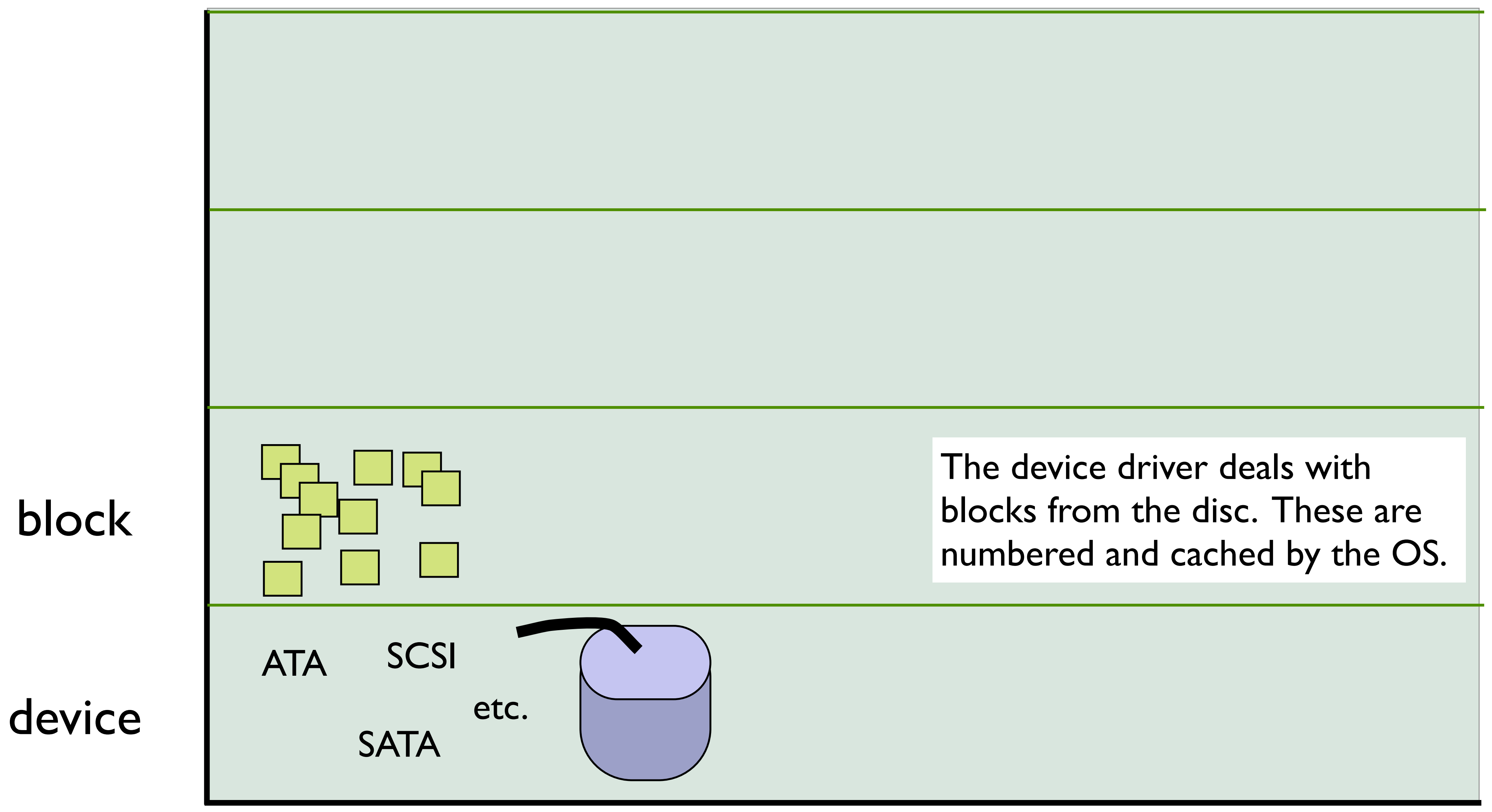
File System

- Device level


IDE
SCSI
SATA
etc.
device
Each type of device has different control messages and connector, includes HD disc , SSD, SD card, ..

File System

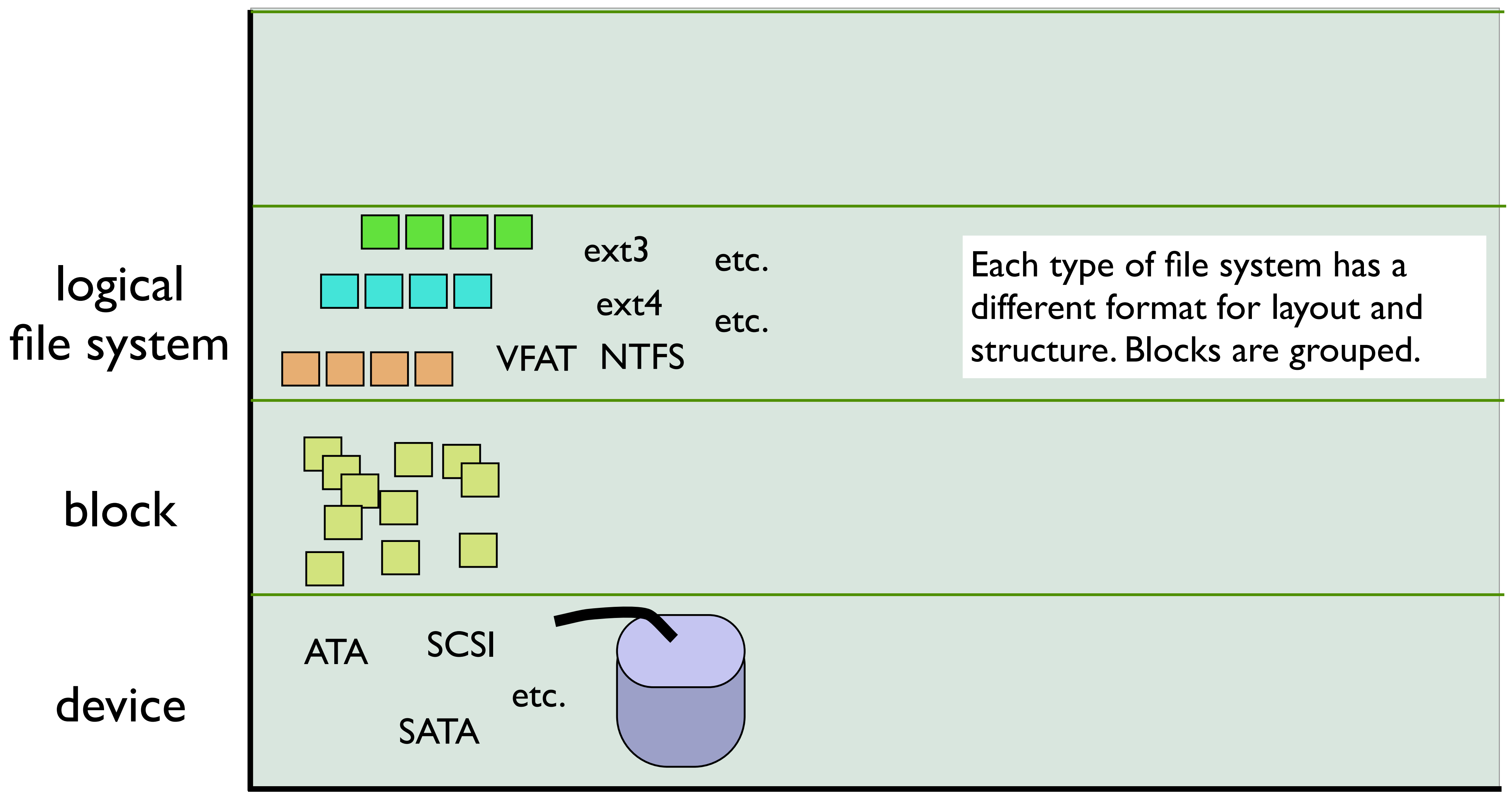
- Abstraction Layer 1: device → disc block



The device driver deals with blocks from the disc. These are numbered and cached by the OS.

File System

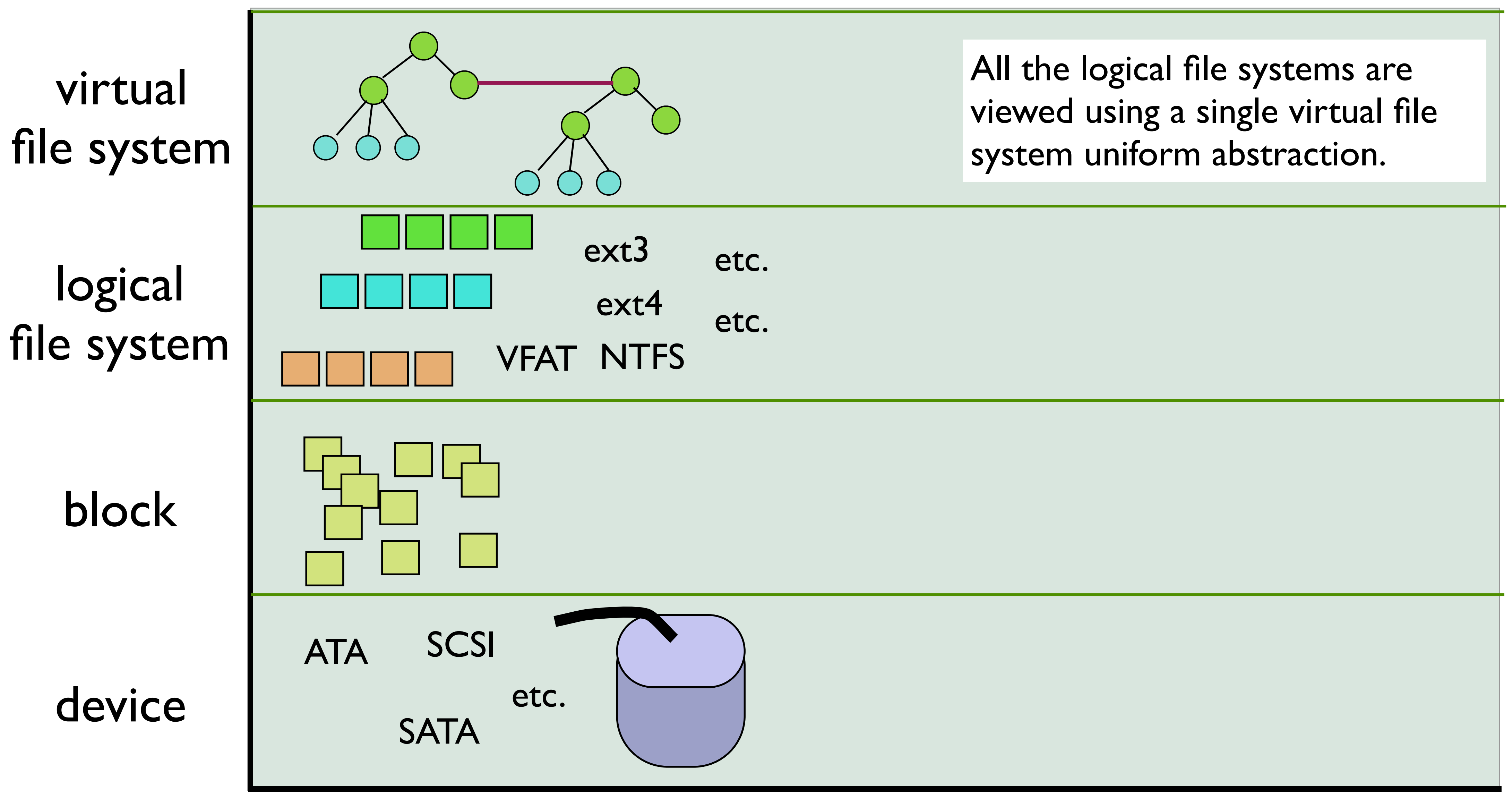
- Abstraction Layer 2: disc block → FS format (logical file system)



Each type of file system has a different format for layout and structure. Blocks are grouped.

File System

- Abstraction Layer 3: FS format → generic tree (virtual file system)


virtual file system
All the logical file systems are viewed using a single virtual file system uniform abstraction.
logical file system
ext3
etc.
ext4
etc.
VFAT
NTFS
block
device
ATA
SCSI
etc.
SATA

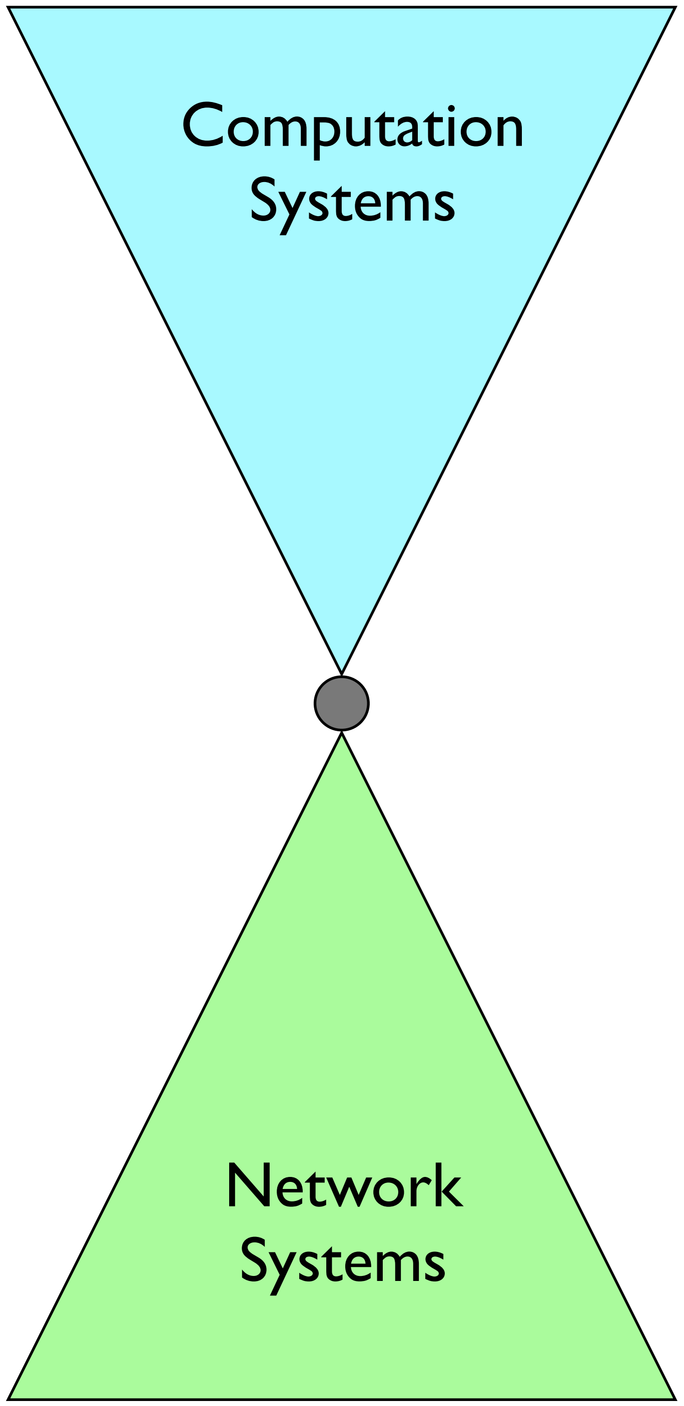
File System

- The graphic presents a view of file systems on local discs.
- File systems can be extended to include:
  - storage area networks on Fibre Channel
  - networked file systems such as NFS, CIFS
  - iSCSI: block level SCSI over LAN / WAN
  - Cluster file systems - a single view across a fully distributed set of discs
  - user space file systems: to mount zip files, gmailFS, ...
    - for more see FUSE

File System

- The value of all of this is that there are multiple layers to bind into, depending on the functionality in the solution.
- The whole mechanism allows for:
  - more files, bigger files, more reliability, more flexibility, more scalability, uniformity, ease of use, concurrent access, more users, seamless distributed access, different technologies working together, ...
  - different layers having a separation of concern, but also have useful techniques for mapping blocks to files, caching, block re-ordering, read-ahead, ...
- What did we lose ? The aggravation of low-level fiddling !
- All from one well designed abstraction !

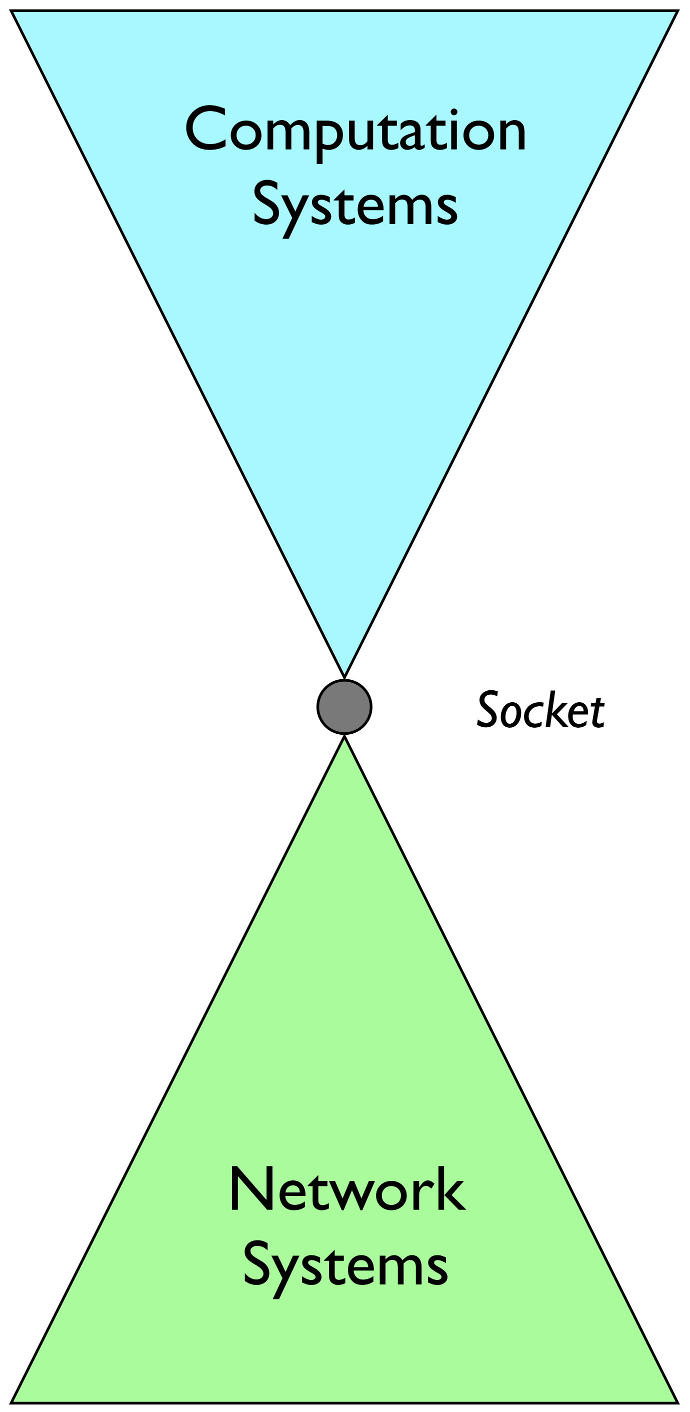
Networking



The Internet is made up of two great infrastructures:
- the computational systems
- the networked systems

They are intrinsically linked, however there is a tension regarding the exposed set of functions.

Networking

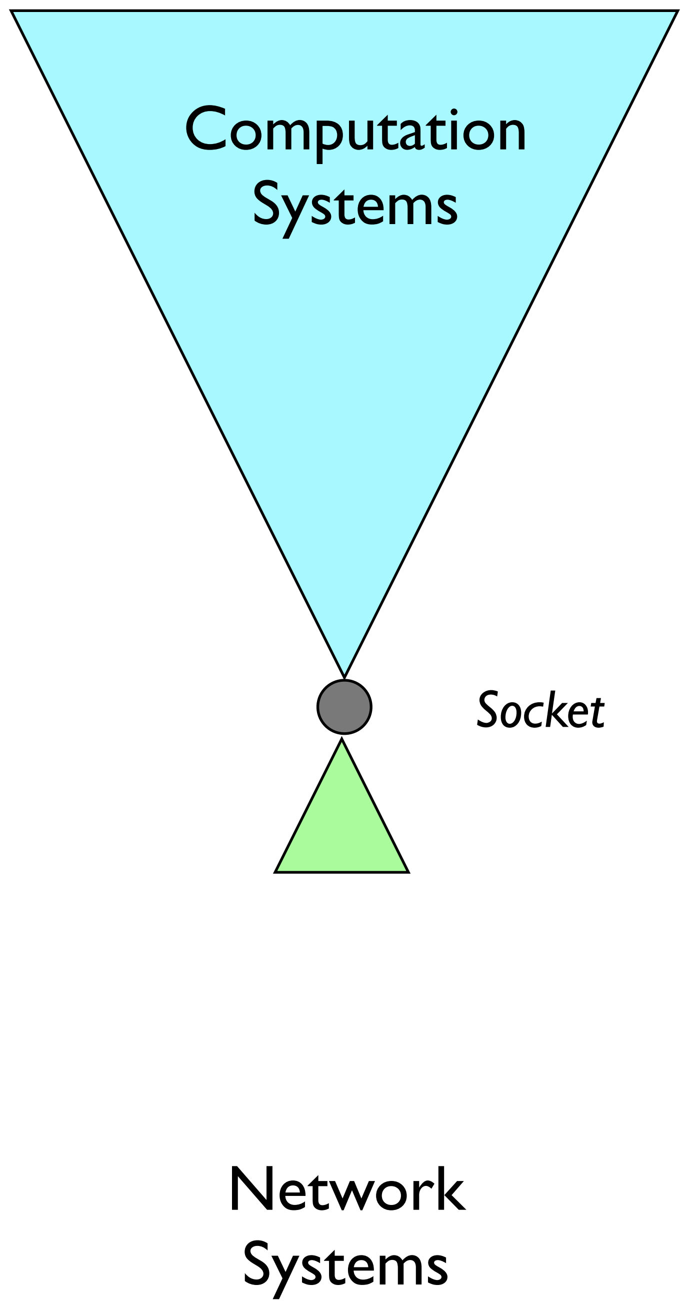


Both systems are joined using the Socket.

The Socket is an abstraction that allows data to travel from one point to another.

All applications use Sockets to communicate.

Networking



As far as the computation systems are concerned, the network could be simplistic and not very featureful.

The network is abstracted away as a delivery mechanism. Also the network operators hide the network features and attributes. So users cannot always get the benefit of both systems.

Networking

- The underlying networking hardware on many machines supports link layer transmission mechanisms, and can operate using very different schemes, including: ethernet, optical, wireless, WiFi, bluetooth, and so on.
- The networking layer of most current operating systems is presented using TCP/IP. In essence, this gives the user / programmer two kinds of network interaction:
  - UDP – an unreliable datagram delivery mechanism, and
  - TCP – a reliable stream delivery mechanism
- A Socket is a uniform abstraction as a network access point, that supports operations for sending and receiving data. Both UDP and TCP are accessed via a Socket API.

Networking

- The use of the Socket abstraction and TCP/IP hides all of these different networking interfaces, and they can all co-exist in the same machine.
- One might consider that TCP itself is another layer of abstraction over the network transport. To the user it presents a reliable stream, and to the network it sends packets. Each piece of data presented by the user to a TCP socket stream will become many packets at the network level, all of which are intrinsically managed.
- This differs from UDP, whereby each piece of data presented to a UDP socket will become one network packet.

Networking

- TCP actually has 3 abstraction mechanisms:

  (i) two byte streams – an input stream and an output stream which can be accessed from either end of the TCP connection, and is used by applications and programmers.

  (ii) a reliable transport mechanism – such that any data loss between the end-points is overcome through re-sending lost data packets

  (iii) a congestion control mechanism – such that TCP can adapt its sending rate, both up and down, depending on how it perceives any congestion in the network.

Languages

- There is a need to express operations on these abstractions in order to make them function.
- The expression of these operations is done through the use of high-level languages, of which there are many.
- A mapping is done, via a compiler or an interpreter, that converts expressions / statements in the high-level language into assembly language - this is a sequence of instructions for the machine.
- They have different syntax structures and different semantics, to specify high-level operations, but what they have in common is that *eventually* they map to the underlying machine.

Machine Instructions

- Assembly languages are a 1-to-1 mapping of a text representation of an instruction.
- The machine operates, but it take considerable expertise to elaborate and understand how a sequence of machine instructions represents a higher level concept.
- Expressing operations instruction by instruction is very low level, but this is how network management is done.
- In networking, the operations are instructions for a router or a switch. Although these router / switch instructions undertake more work than a machine instruction, in essence the situation is the same.
- *Machine instructions are hard to reason about.*

Languages

- Programming languages have been an area of great interest for many years.

  “... today... 1700 special programming languages used to 'communicate' in over 700 application areas.” -- Computer Software Issues, American Mathematical Association Prospectus, ???????????

What year was this published ?

Languages

- Programming languages have been an area of great interest for many years.

  "... today... 1700 special programming languages used to 'communicate' in over 700 application areas." -- Computer Software Issues, American Mathematical Association Prospectus, July 1965

  from "The Next 700 Programming Languages", Peter Landin, Communications of the ACM, Volume 9 / Number 3 / March 1966

Languages

- **procedural** – FORTRAN, COBOL, Algol, Pascal, C ...
- **list** – LISP, Scheme
- **vector** – APL
- **pattern matching** – Snobol, awk
- **object oriented** – Simula, Smalltalk, Java
- **logic** – Prolog
- **stack based** – Forth, Postscript
- **rule based** – OPS5
- **functional** – ISWIM, SASL, Haskell

Languages

- **procedural** – FORTRAN, COBOL, Algol, Pascal, C ...
  1957 1959 1960 1970 1972
- **list** – LISP, Scheme
  1958 1970
- **vector** – APL
  1964
- **pattern matching** – Snobol, awk
  1962 1977
- **object oriented** – Simula, Smalltalk, Java
  1965 1972 1995
- **logic** – Prolog
  1972
- **stack based** – Forth, Postscript
  1970 1982
- **rule based** – OPS5
  1977
- **functional** – ISWIM, SASL, Haskell
  1966 1975 1990

Languages

- Domain specific languages are currently of interest in the world of networking.
- These are languages where the main abstractions and symbols are specific and focussed on the domain, but not always generic and computationally complete.
- Examples in the domain of networking include:
  - frenetic / pyretic – provide a domain specific sub-language for specifying data plane packet processing
  - P4 – a language for expressing how packets are processed by the pipeline of a network forwarding element
- This is a good start, but there is a long way to go.

Observations

- It is still common for network operators to write scripts that interact directly with specific devices.
- However, these scripts are written to send instructions to a machine - a router or an SDN switch.
- If an operator has routers from Cisco and Juniper, there might be 2 versions of the script. Any changes will have to be made to both of the scripts.
- Note: the manual for Cisco IOS alone is over 1200 pages.
- We need to express *what* to do, not *how* to do it. With a declarative language run-time, the what can be dynamically translated into the how.

Observations

- The lesson from the operating systems world is that using high-level programming languages to express operations over abstract elements is far more effective that hand coding with low level device instructions.
- There are many approaches to convert various high level expressions into device instructions, and these have been show to be highly performant in most cases.
  - e.g. UNIX has been written in C since the mid 1970s, except for a few hundred lines of assembler needed to control certain machine specific features.
  - There are tool sets that can create new languages for new domains since the end of the 1970s - yacc & lex.

Observations

- Without abstractions and the right languages it will be extremely difficult to do orchestration in the right way.
- So we need to:
  - Agree common abstractions that we can talk about.
  - Agree on the operations over those abstractions.
  - Add these abstractions into existing programming languages or devise ways to call the network specific languages.
  - These should map down to the devices.
  - Need to try and eliminate most of the special scripts.
  - Only keep the really essential ones.

Observations

- There can be no *programmability* without programs, objects, and run-times.
- There can be no *machine learning / machine intelligence* if there are no representations of the underlying elements to reason about.
- How do we do *intent* if there are no declarative languages and no mechanism to do the translation.
- The lack of abstractions means that interacting with and managing networks has become a difficult and sometimes cumbersome task.

Conclusions

- Abstractions are useful, and the right abstractions give a huge improvement in power and flexibility.
- Although it seems there is a loss using abstractions, the gains can outweigh this.
- Now that compute and networking environments are being combined to support virtual deployments, it is extremely important that the relevant abstractions and languages are put in place.
- We need flexibility and dynamic control. This is necessary with the rise of SDN, NFV and SFC, plus the targets of AI, automation, analytics, and slicing.
- The old ways don't scale up.

Acknowlegements

- The work is supported by the 5GEx and NECOS projects.