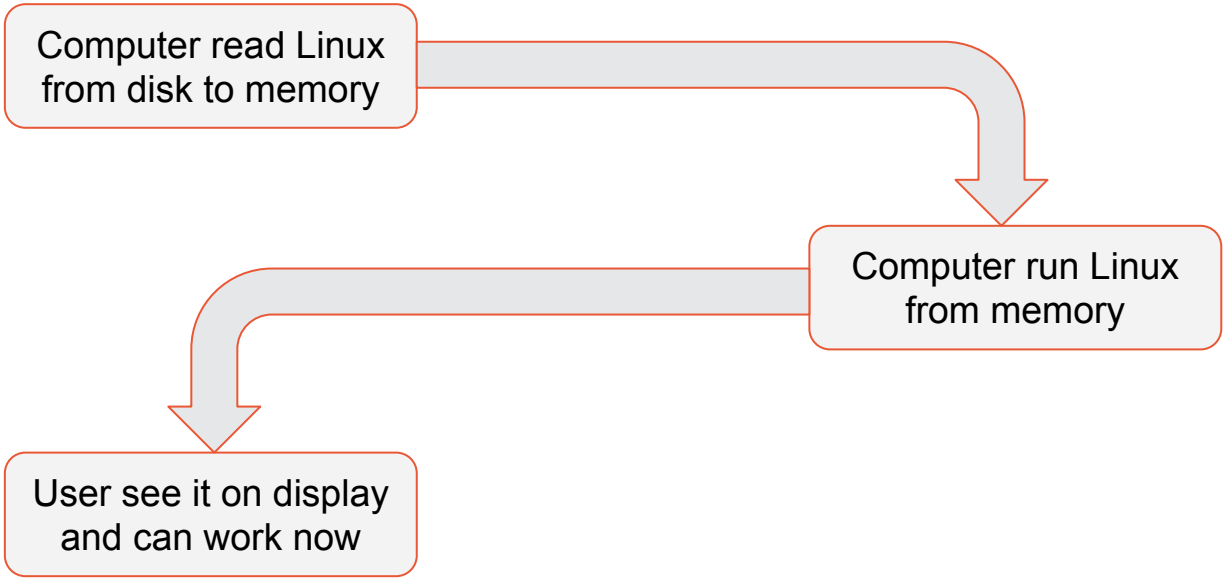
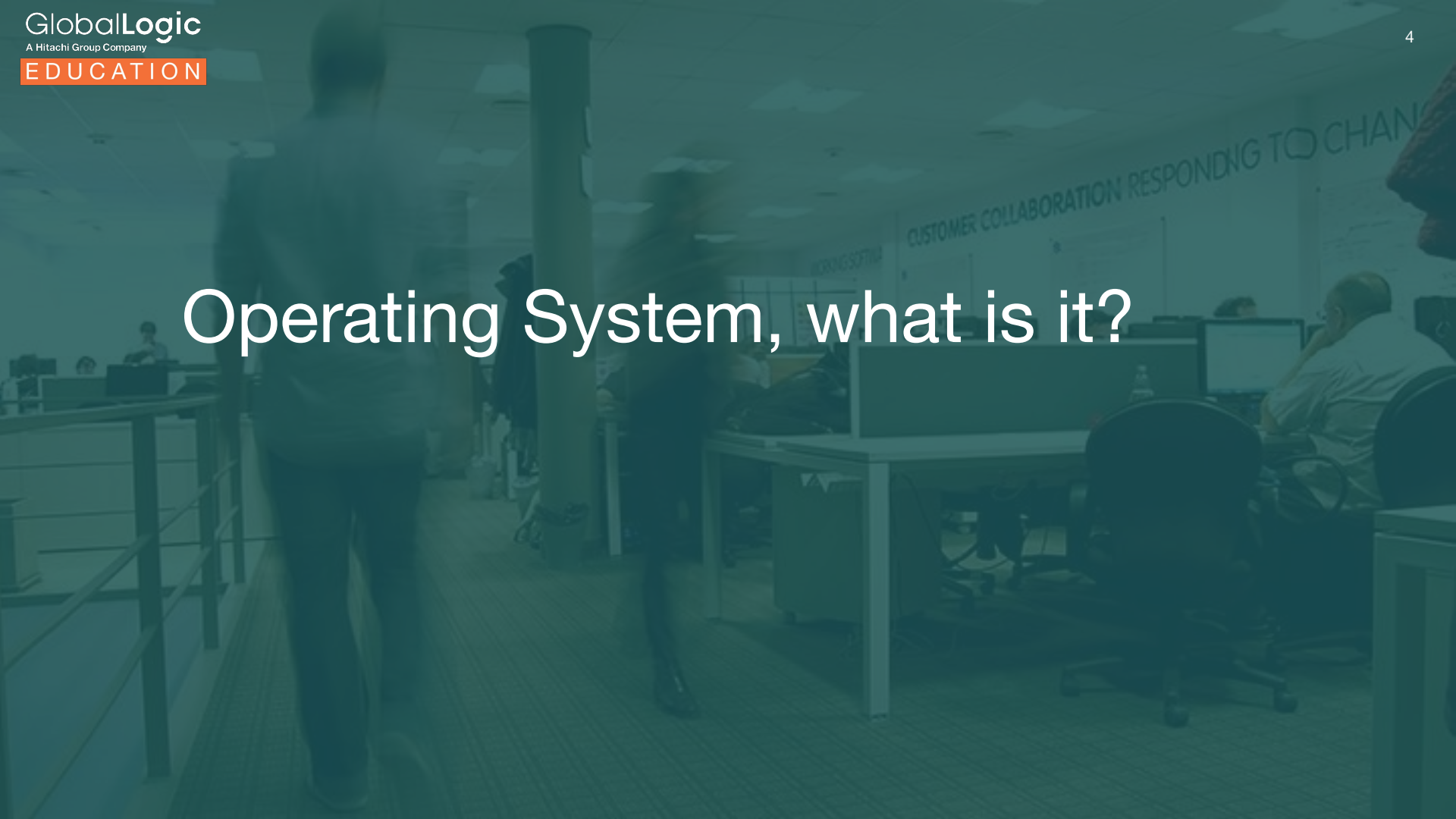

GlobalLogic
A Hitachi Group Company
EDUCATION
Booting a GNU/Linux
Andrii Beregovenko

Agenda

1. What is Operating System
2. CPU and ROM code
3. Initial boot stages
    a. 1st stage - BIOS, CSM, MBR, (U)EFI
    b. 2nd stage - BIOS, Grub, U-Boot, NTLDR.EXE
    c. 3rd stage - Linux Kernel, initrd, drivers, init
4. System boot stage
5. Linux distributions and their components
    a. Init system
    b. Core parts and frameworks
    c. systemd

Booting GNU/Linux OS





Operating System, what is it?

Operating System: definition

The set of software products that jointly controls the system resources and the processes using these resources on a computer system.

© Wikipedia

The low-level software that supports a computer's basic functions, such as scheduling tasks and controlling peripherals.

© Oxford Dictionaries

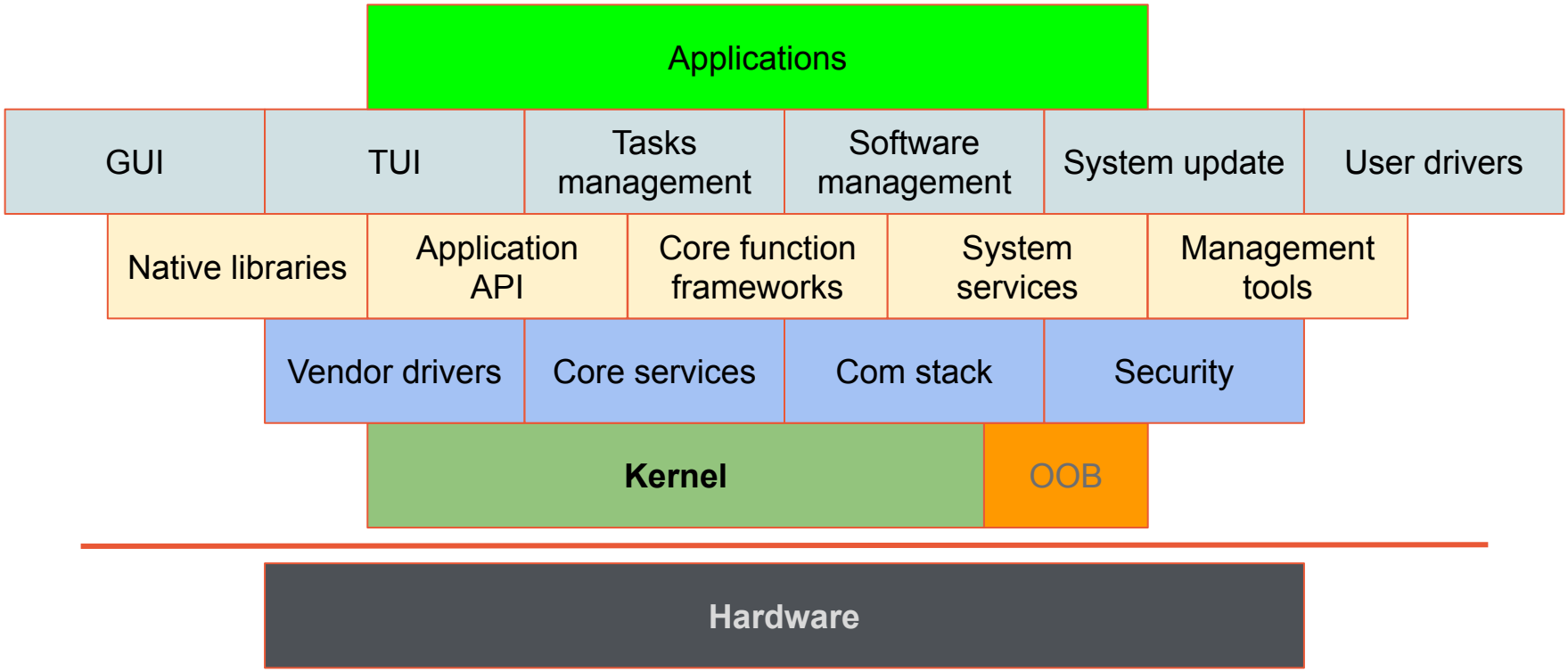
An operating system is the most important software that runs on a computer. Without an operating system, a computer is useless.

© Goodwill Community Foundation Inc.

A set of programs that control the way a computer system works, especially how its memory is used and how different programs work together.

© Cambridge Dictionary

Operating System: software stack



Operating System: functions

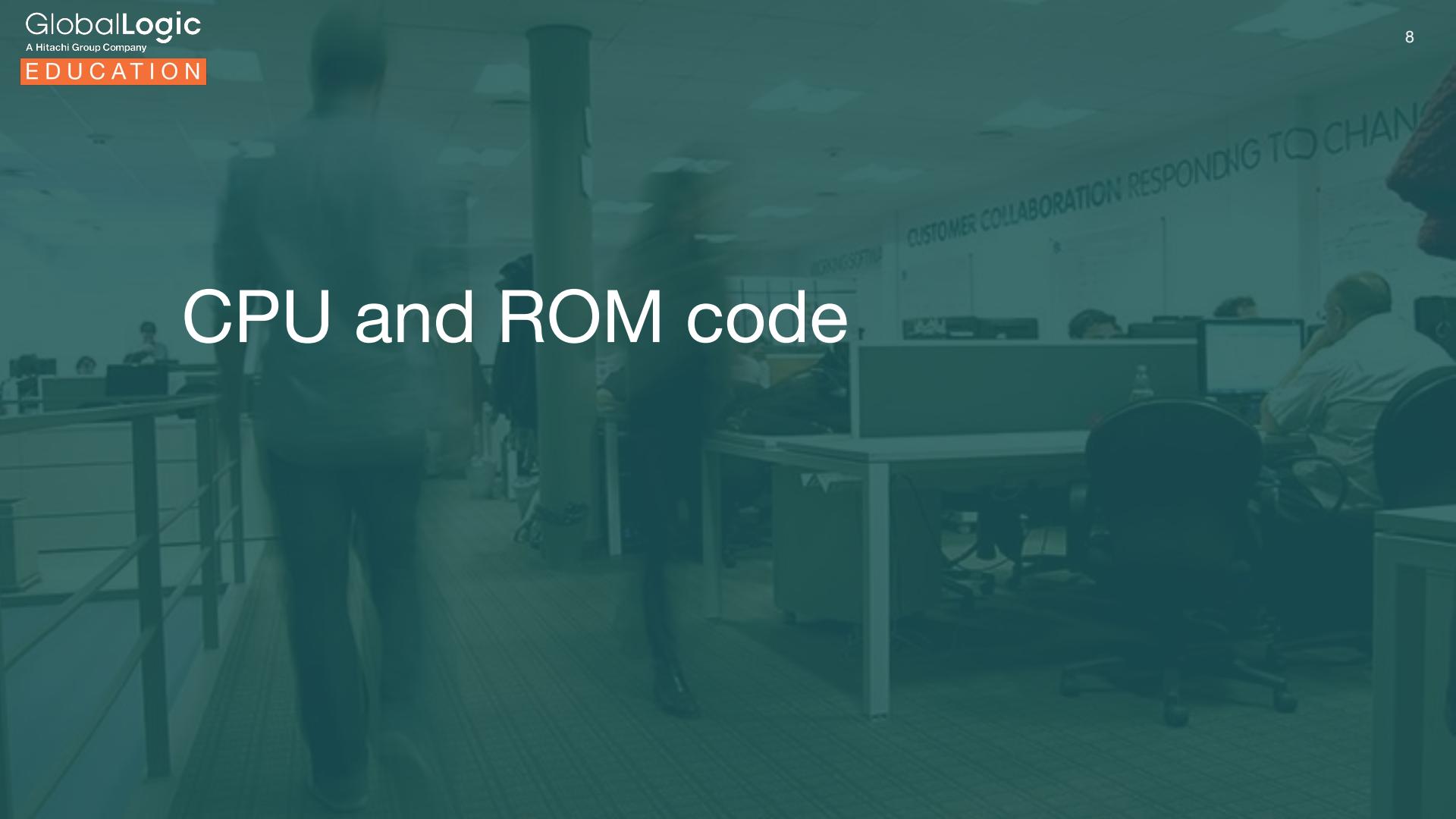
There are functional and nonfunctional requirements to the operating systems.

The functional requirements are:

- Manage a computer hardware
- Manage a computer installed software
- Enable communication between different software applications
- Maintain overall system security

While a nonfunctional requirements are mostly focused on user:

- The main purpose of the operating system is to stay invisible for end user
- The operating system shall require as less maintenance as possible
- If something goes wrong it shall be able to easily recover itself
- It should be cheap or free



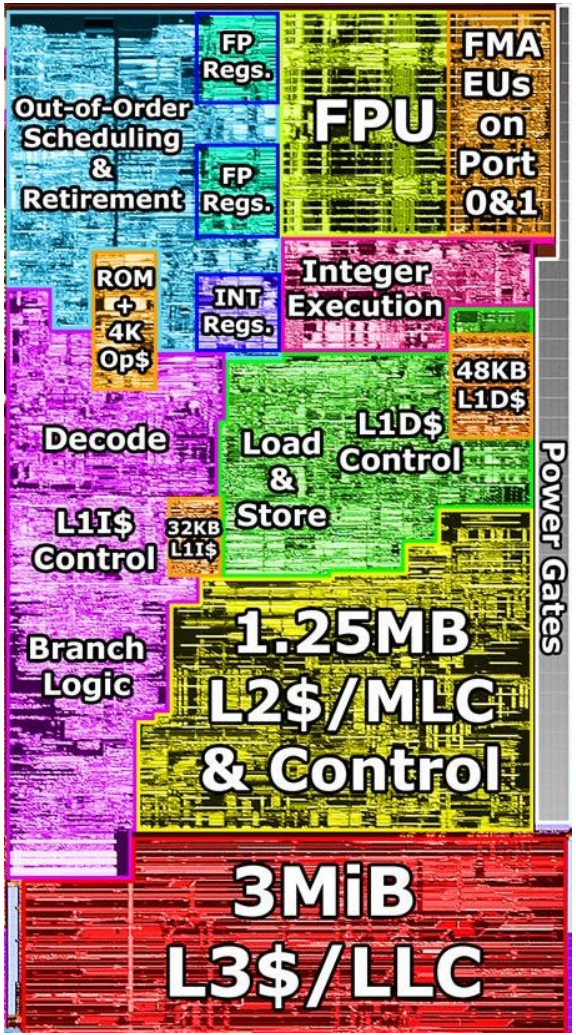
CPU and ROM code

What is CPU

The CPU consists of a big amount of transistor logic, which forms a different functional blocks

- Control Units
- Registers
- ALU
- MMU
- Clocks

To make everything work there is an internal memory. But amount of this memory is small, measured in KB and in some cases up to MB (in case of L3 caches)



ROM code

The activation of the CPU internals is a complicated process, thus there is a sequence of steps which needs to be done. This sequence of steps also is called as a *program* (in fact it does not differ much from our regular programs).

However, the CPU do not know yet how to load it from external memory/sources because *it is not yet activated*.

Thus the *program* is flashed into a CPU, directly into silicon, during a common lithography process. Obviously, this program cannot be changed, removed or deactivated, it will stay with a particular CPU unit its entire life.

This *program* usually is called a **ROM code**.

ROM code functions

The ROM code functions might vary depending on a particular CPU/SoC purpose:

- Configure internal clocks and necessary buses;
- Determine the boot source;
- Jump to next stage bootloader.

Usually the ROM code is executed on a first CPU core the only.

The complex SoCs might have several functional processing units and each of them might have its own ROM code. There might be dependencies as well.

The only way to modify behavior of the ROM code is eFuse, but even though this is done usually on a factory as a part of QC, Calibration, or assembly process.

Initial boot stages

Boot stages: overview

The boot of the modern platform software is a quite complicated process:

- The available hardware of a typical platform is quite rich;
- A tremendous number of features are implemented in tight collaboration with the hardware;
- Each functionality shall be prepared on a hardware level for future use.

This process involves multiple software parts are doing its job simultaneously.
In order to initialize all necessary hardware features the boot process have to process through multiple boot stages one by one in sequential order.

On each stage more and more system features will be initialized, configured, tested and, if needed, than activated.

First stage: PC style

- BIOS - a basic element of the PC compatible computer. Its main functions are as follows:
  - Identification of the CPU, configuration
  - Check the Memory presence, identification
  - Configuration of the motherboard power supply
  - Activation motherboard periphery
  - Activation of the south bridge
  - Activation of the PCIe devices
  - Activation of the disk subsystem
  - Activating CPU
- CPU starts code execution from disk storage

First stage: PC style - CSM

CSM stands for Compatibility Support Module, this is a way how PC compatible computer were booting since old days up until 5-10 years ago.

In this mode the the MBR is being searched on the disk (either HDD, or FDD, and even USB flash drive) at some particular offsets. The MBR contains actual bootloader code as well as some additional information which shall help with booting next stages. If MBR content is successfully identified, the BIOS code passes execution control to it, and thus boot continues.

Since this technology is known since at least 40 years, it is still partially supported even in the most modern motherboards.

First stage: PC style - UEFI

UEFI stands for Universal Extensible Firmware Interface.

The main idea behind this technology is to extend functionality of the motherboards to a system specific one, not yet booting in a full operating system. The EFI content usually is stored on a disk on a special partition with id 0xEF. Usually it if FAT formatted, where next stage bootloaders are stored as files. In addition to next stage bootloaders there might be other functional units which allows to do a vendor specific operations to maintain or recover a system.

Modern BIOS is able to read EFI partition and turn it into a boot menu, so that the user can choose how to proceed with booting.

The EFI is also responsible for chain of trust and key manipulations, which makes a maintenance of the security related feature more secure and reliable.

Second stage

On this stage the full blown bootloader is in charge. It is usually function rich and is able to use all available platform hardware and peripheral devices to satisfy platform boot process and execute all required steps.

Typically, after previous stage had finished, and passed the control to the second stage, the next step is decided based on the platform:

- In case of GNU/Linux on PC - load from disk and execute Grub;
- In case of embedded platform - execute U-Boot;
- In case of Windows, execute NTLDR.EXE(there is a tiny bootloader here, but it quickly finishes its work and jumps into ntldr)

The size of stage bootloader is counted from single Megs up to tens of Megs

Second stage

Usually the second stage is responsible for:

- Reconfiguring random access memory (if required);
- Reading the boot configuration from NV memory (like EEPROM or Flash);
- Make a judgement about a boot reason, and thus about further boot path;
- Load next boot step into the memory;
- Detect and prepare system specification and configuration;
- Prepare configuration segment in memory with periphery specifics.

On this step the system configuration is being fully known, optional features are discovered and, if configured so, are activated. If resources are shared between different MCU/CPU of the same hw platform, but different operating system environment this is usually a place system configuration is done.

After all these steps completed the control is passed to the next stage…

Third stage

At this stage, finally, the Linux kernel takes over a control.

Following steps are made:

- By using system configuration from previous stage the main platform software components are configured and activated;
- If initrd is provided it is extracted into a memory and mounted as a temporary root file system;
- The Linux kernel uses provided Device Tree and detect available hardware;
- For discovered hardware the drivers are loaded and activated;
- The real rootfs is mounted;
- /sbin/init is executed and thus the **first user-space process** started

First stage(non PC)

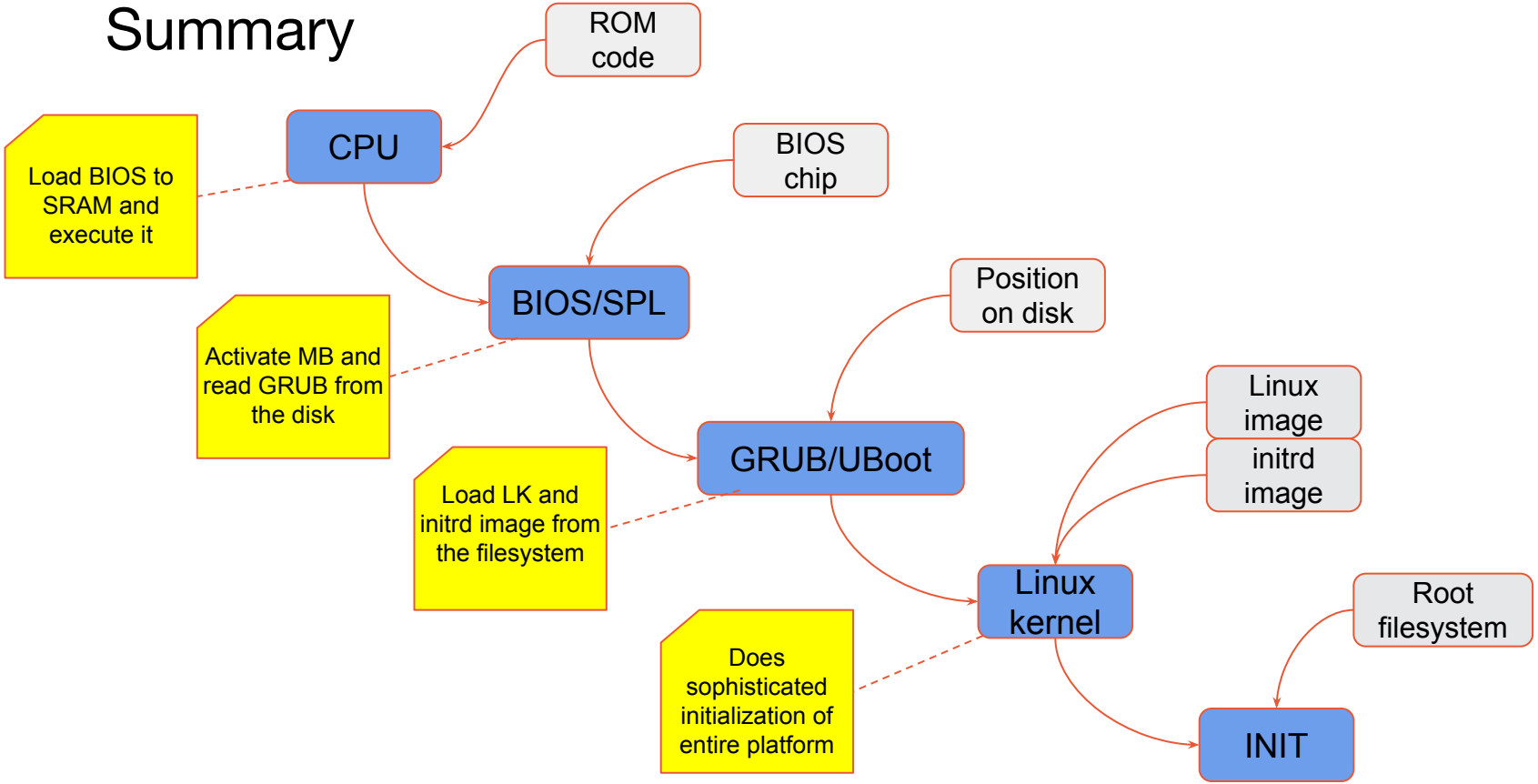
A typical embedded system is not that rich as PC compatible platforms due to natural business reasons.

- Embedded systems does not have a BIOS (usually);
- The amount of built in CPU memory is small;
- The hardware setup usually is less common and is more task specific;
- The boot sources might vary and thus cannot be part of the CPU ROM code;
- The boot source might require complicated initialization procedure.

Thus a basic devices and peripherie has to be configured in order to enable next stage loading.

The U-Boot provides a tiny footprint of the main U-Boot loader called SPL(or MLO, bootloader.bin, etc) which would have the only very limited and absolutely required set of functions and still will fit into CPU's own memory (usually it is counted in tens of K).

Summary


ROM code
CPU
Load BIOS to SRAM and execute it
BIOS chip
BIOS/SPL
Activate MB and read GRUB from the disk
Position on disk
GRUB/UBoot
Load LK and initrd image from the filesystem
Linux image
initrd image
Linux kernel
Does sophisticated initialization of entire platform
Root filesystem
INIT

System boot

Preconditions

At this moment of boot our system in fact is already fully loaded, hardware is initialized and thus can start making some value.

However, the operating system itself does not make much sense alone and does not bring any value. Thus, we have to load programs which would produce a value.

For that, following criteria should be met at this moment:

- All persistent storages(like local or remote disks) are discovered and ready to be used;
- The operating system component are ready to be executed (rootfs mounted)
- The userspace and on-demand drivers are ready to be loaded
- The Linux Kernel successfully executed Init and provided sufficien amount of resources to it

What is init

The Init is a name of the virtually any program which runs first on the system in the **User Space**.

- Any further processes will be spawned by Init and will be a child of the Init
- The Init is in charge to control start of the userspace environment
- The Init has to monitor basic system service and react appropriately
- The Init is responsible for the primary stability of your system in its basics

While list of the responsibilities does not look somehow big or complex, in reality it is a huge responsibility and Init usually is quite sophisticated piece of the software.

So let's see what Init does in reality

- assembles complete filesystem layout in accordance to fstab;
- loads extra drivers as per user configuration;
- starts such basic service as:
  - networking (ethernet, wifi, CAN/LIN busses, IP configuration);
  - userspace communication mechanisms (dbus, mqtt, sockets, watchdogs);
  - system components configuration (usb devices(mounting), time service, snmp, etc);
  - peripheral buses (spi, i2c) and devices/sensors on them;
  - system security services (firewall, MAC, DAC, audit);
- starts graphical subsystem (if present and requested);
- monitors key system components and restarts them upon crash or update.

In addition to that depends on configuration it might be also remote login services, web, mail, DNS, and application services, and so on…

Target of Init on different system types

Desktop PC

- Start Graphical interface
- Start audio subsystem
- Dynamically configure network connection
- Starts a different communication services
- Manage system services layout depends of the requested system configuration in a runtime
- Starts different backup and synchronization background services
- Manages a computer power domain and related states

Server

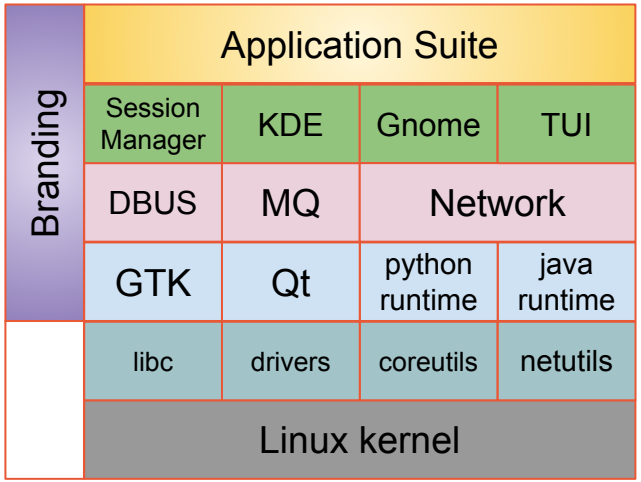
- Configure storage subsystem
- Configure network access
- Configure network security policies
- Configure local security mechanisms and services
- Perform a consistency checks and verifications
- Activate application services
- Activate network services
- Control activation chain trust
- Monitor services and applications

Embedded system

- Activates required peripheral devices
- Configure power domain and energy saving policies
- Synchronizes with service providers and cloud infrastructure
- Configure a User notification subsystems (lights, sounds)
- Maintain a device maintenance cycles and routines

Linux distributions and their components

A software distribution structure



The typical Linux distribution consists of the following parts:

- Precompiled Linux Kernel;
- Bundled set of 3rd party drivers, typically network, GPU, misc device drivers;
- System library and environment, typically GNU coreutils;
- Set of high level frameworks, like GTK, Qt, different language runtimes, etc;
- Set of preconfigured daemons and services;
- Bundled and preconfigured desktop environment;
- The distribution branding package which applies on different levels.

Init system

During more than 25 years of Linux distribution evolutions there was multiple software implementing an Init. Let's name them in the order of their popularity:

- SysVInit
- Upstart
- OpenRC
- Systemd

At this moment SystemD dominates across most popular distributions. The reason for that are different, but the most important one: systemd rethinks the scope of responsibility of such a key component as Init

Core parts and frameworks

- GNU LIBC
  - ld.so - dynamic linker/loader of the programs
  - libc - implementation of the basic system level API
- Disk management frameworks
  - md, dm, lvm2 - hardware and soft RAIDs, logical partitioning
- Security
  - SELinux, AppArmor - control access to system resources
- GNU coreutils
  - all the tools you love so much, like grep, cut, less, sed, etc…
- Shell interpreter
  - oh well, bash of course 😉

Core parts and frameworks

- GUI - Graphical user interface
  - X11
  - Wayland
- Sound - the sound services which enables an audio subsystem
  - PulseAudio
  - Pipewire
- Sessions - login/logout user sessions, enables system configuration
  - SDDM, GDM
- IDE: KDE, GNOME, etc - the user work environment
- NetworkManager
- ModemManager
- PolicyKit, polkit

systemd

SystemD has been one of the most dramatic stories in the entire Linux world for 10 years. This happened as a response to the way of interpreting the functions of modern computers (especially personal ones, that is less applicable to servers and the embedded world).

The systemd is not the only Init, but rather pretend to be a whole system separate management layer:

- it is event handling daemon;
- it is daemon and service manager;
- system configuration (different consistent states of the whole system);
- it provides access to containers;
- the userspace daemons and services
- much more…

Packages and meta packages

The GNU/Linux distributive package is a minimal standalone software delivery entity. Usually it contains libraries, binaries, resources or other deliverables that can provide a value to the overall software layout on the target system. In most of the cases there is no need to install all the packages that are related to the particular software unit.

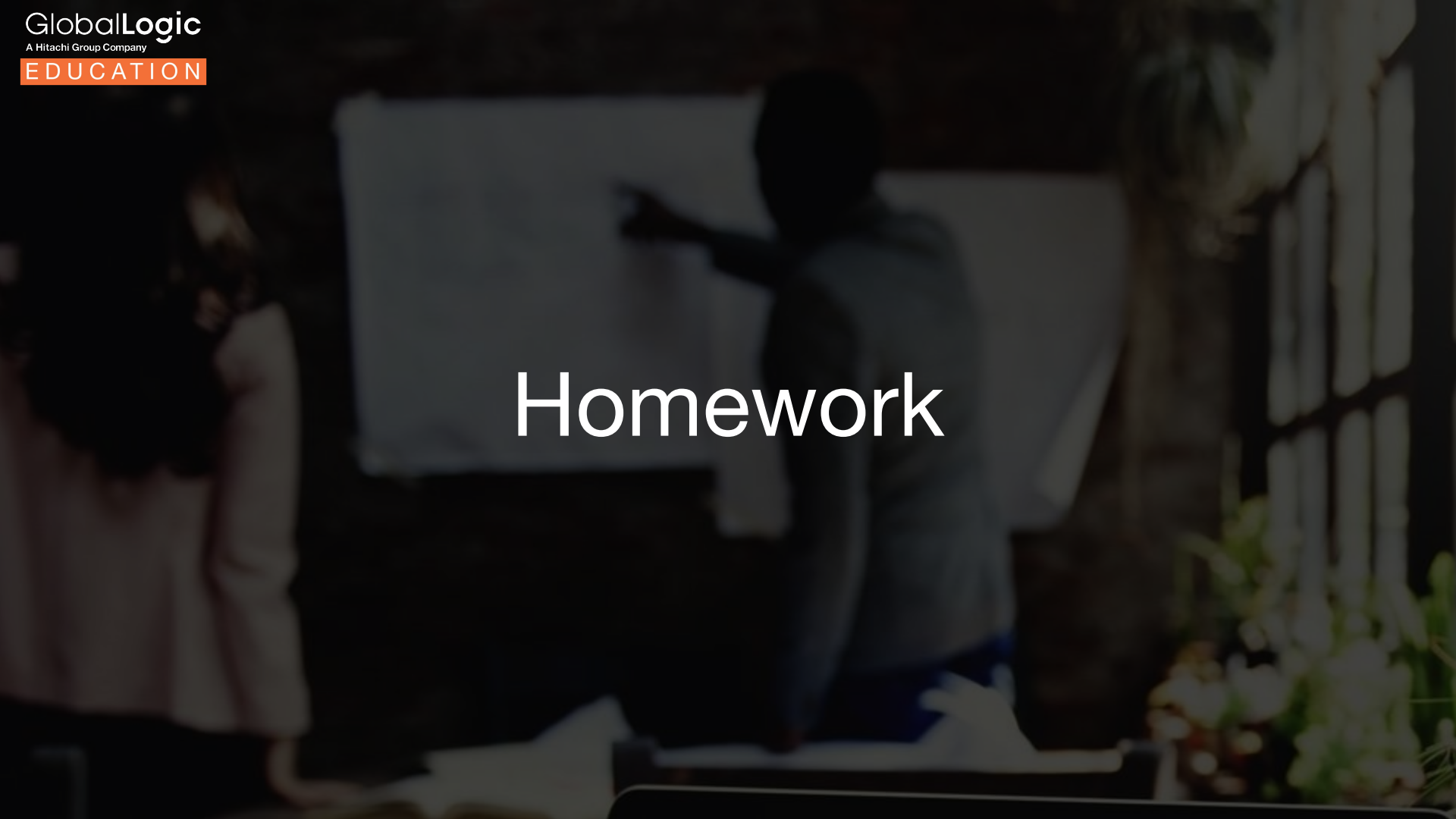
The meta-packages in its idea are opposite: they describes a function or feature of the system which is implemented with multiple packages or even multiple programs. By installing meta-package there will be installed one or multiple packages which together will enable desired functionality.

❗ In fact Debian (and derivatives) dominated the GNU/Linux distribution world because of smart and ahead of time design of the package management.

GlobalLogic
A Hitachi Group Company
EDUCATION

Q&A

GlobalLogic
A Hitachi Group Company
EDUCATION

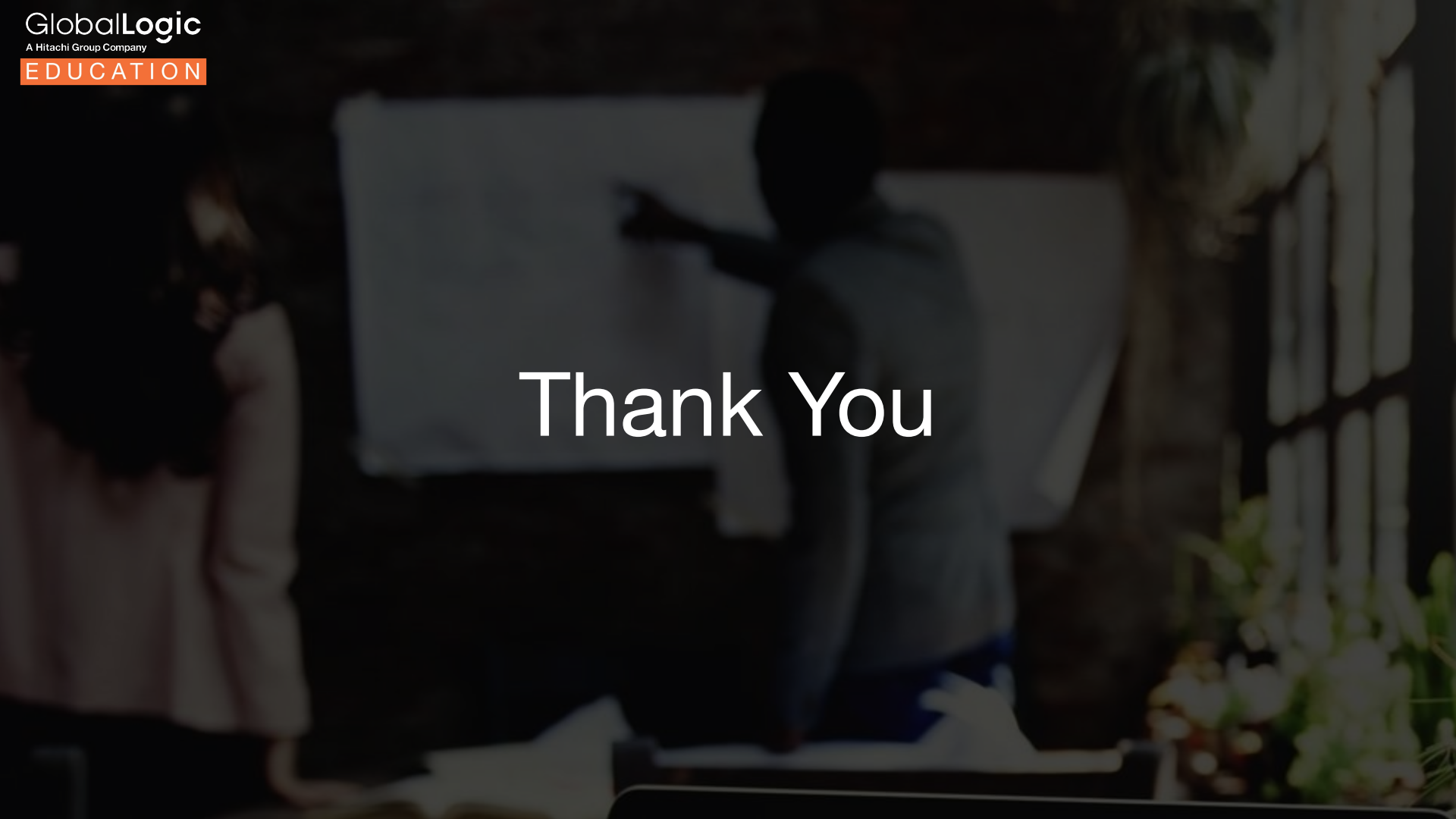


Homework

Homework

- Figure it our where is located a GRUB bootloader
- Save the bootloader to a separate file on a disk
- Remove GRUB bootloader
- Reboot VM to see OS does not boot anymore
- Using an Ubuntu live cd, boot into recovery mode and restore bootloader
- Take a few pictures/screenshots in a progress to reveal how you did it
- Note down a commands you've used to recover a bootloader while was booted from a liveCD image

Good luck!


GlobalLogic
A Hitachi Group Company
EDUCATION
Thank You