Devices and drivers

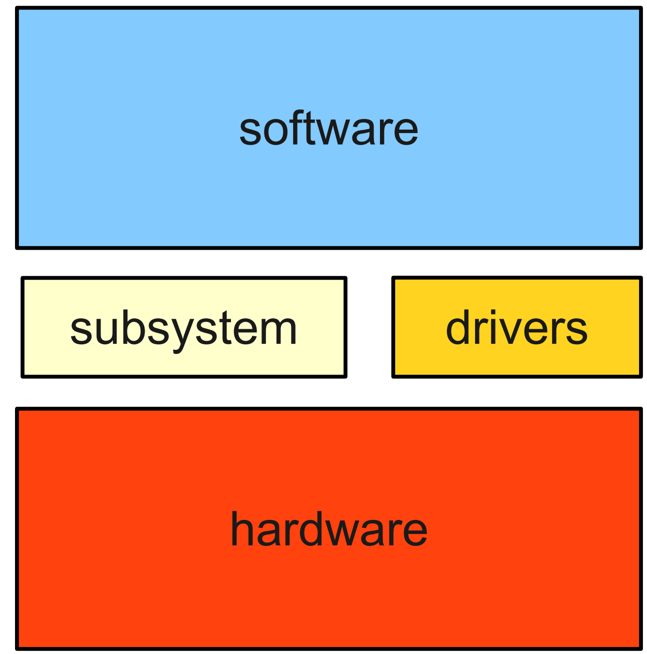
Martin Děcký

DEPARTMENT OF DISTRIBUTED AND DEPENDABLE SYSTEMS
http://d3s.mff.cuni.cz/

CHARLES UNIVERSITY IN PRAGUE
FACULTY OF MATHEMATICS AND PHYSICS

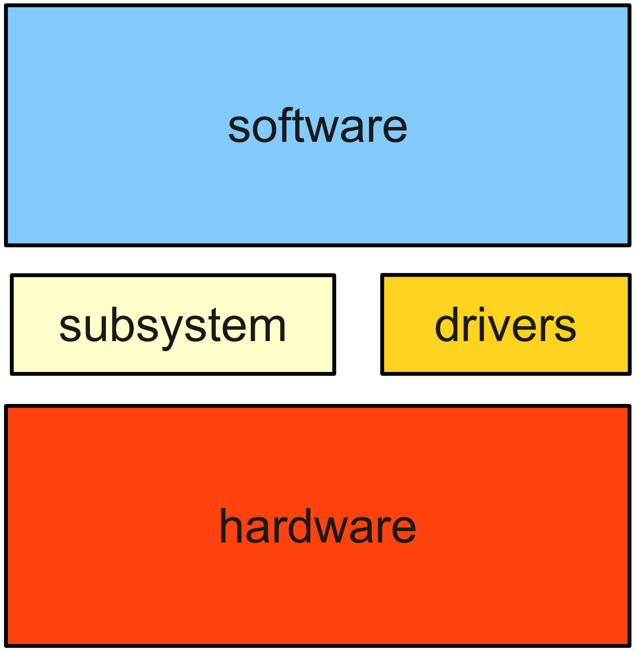
Device driver interfaces

- To the software
  - Interface abstraction
    - Special features
  - Platform & bus neutrality
  - Naming
  - Persistent identification
- To the hardware
  - Device registers
  - Interrupts
  - DMA



Device driver interfaces (2)

- To the device drivers subsystem
  - Discovery (probing)
    - Initialization
    - Device (tree) hierarchy
  - Configuration
    - Device allocation
    - Interrupt routing
  - Hot plug/unplug
    - Power management
  - Queueing, asynchronous requests



Software interface abstraction

- Unix approach
  - A limited number of device classes
    - Character devices
    - Block devices
    - Network devices
    - Custom methods (IOCTLs)
- Windows approach
  - Large and precisely defined set of device classes
    - Serial ports, Parallel ports, Drive controllers, Input devices, Printers, Scanners, Buses, GPUs, Sound cards

Software interface abstraction

- Plan 9
  - *Everything is a file* semantics
    - This time *really*
    - Total transparency
    - Complexity hidden inside the communication

Platform & bus neutrality

- Platform neutrality
  - High level language
  - Programming abstractions to hide implementation details
    - Ports vs. memory mapped registers
    - DMA memory locations
    - Memory consistency model
    - Endianess

Platform & bus neutrality (2)

```
#define IO_SPACE_BOUNDARY  ((void *) (64 * 1024))

void pio_write_8(ioport8_t *port, uint8_t val)
{
    if (port < (ioport8_t *) IO_SPACE_BOUNDARY) {
        asm volatile (
            "outb %b[val], %w[port]\n"
            :: [val] "a" (val),
               [port] "d" (port)
        );
    } else
        *port = val;
}
```

IA-32

```
void pio_write_8(ioport8_t *port, uint8_t val)
{
    *port = val;
}
```

MIPS

Platform & bus neutrality (3)

- Bus neutrality
  - Leaf node device drivers independent (to a degree) on the interconnection
    - VESA Local Bus, SBus, PCI, AGP, PCI-X, PCI Express, ExpressCard
    - ISA, EISA, MicroChannel
    - SCSI (multiple variants), iSCSI, SAS, USB MSD

```
int __devinit amd74xx_probe(struct pci_dev *dev, const struct
  pci_device_id *id)
{
    if (dev->vendor == PCI_VENDOR_ID_NVIDIA) {
        /* ... */
    }
}
```

Naming & identification

- Complex issue with multiple levels
  - Simple programmatic enumeration
    - Unix
      - /dev
      - /proc
      - /sys
    - Windows
      - Device paths
        - multi(0)disk(0)rdisk(0)partition(1)
      - Enumeration according to device classes

Naming & identification (2)

- Complex issue with multiple levels
  - Persistent identification of instances
    - The same physical device should be always identified by the same (persistent) identifier
      - Independent of physical position
      - Independent of enumeration order
    - GUID
      - Generated from hard-coded device IDs
    - Natural identification
      - MAC address
    - Human readable naming
      - “Network connection 1”, eth0, /dev/raid-volumes/root-fs
      - Location dependent (SMBIOS)

Naming & identification (3)

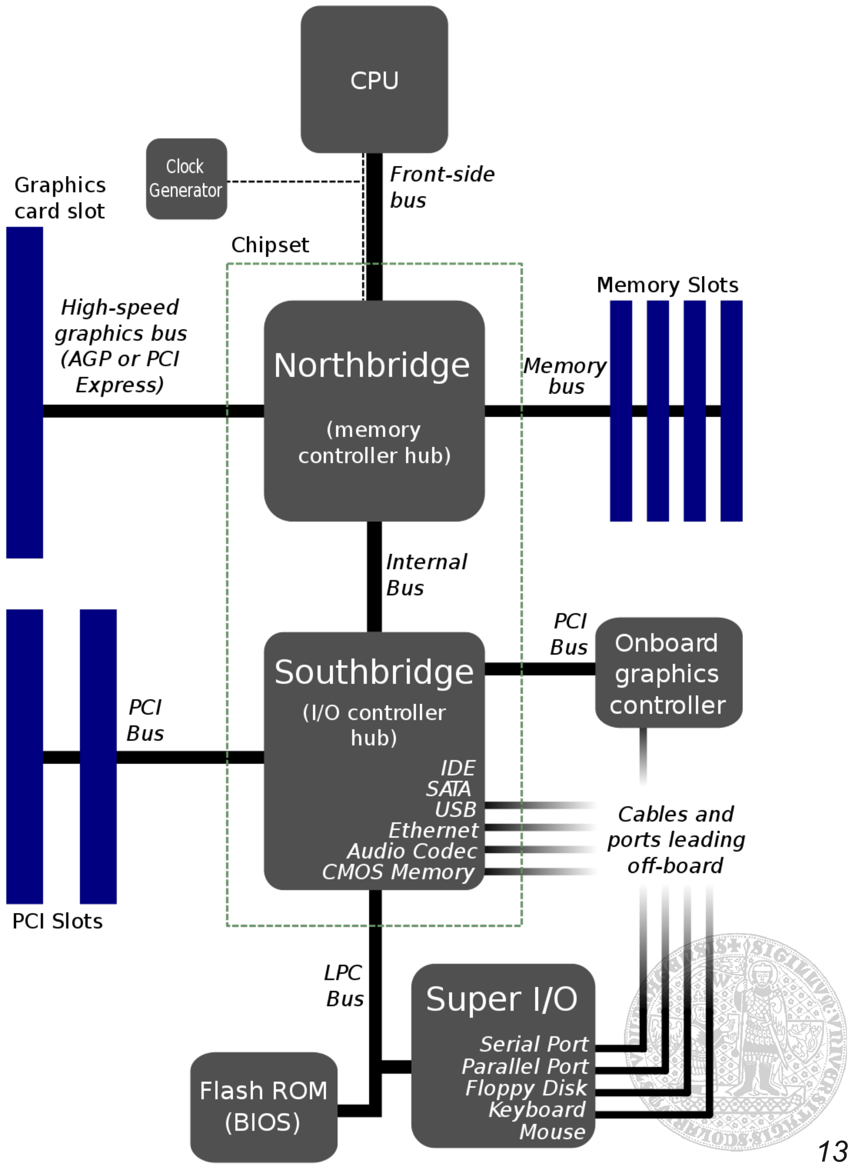
- Complex issue with multiple levels
  - User interface
    - The user wants to “click on the device”
    - The user wants to list all available devices
      - Context specific
    - Unix
      - Originally extremely user unfriendly in this manner
        - Even unfriendly with respect to detection & hot-plug
      - Huge Improvements in the last 6 years (udev)
    - Windows
      - Based on device classes, mostly working reasonable
      - Problems with unknown devices (identification), multiport devices, abundance of strange “tray helpers”

Hardware interfaces

- Major challenges
  - A single driver cannot mostly work by itself
    - Interrupt controllers & interrupt routing
      - Interrupt sharing
      - MSI
    - Bus mastering & DMA controllers
    - Buses (in hierarchy)
      - Physical hierarchy of interconnection does not reflect logical hierarchy
    - Configuration of resources
      - Jumpers, Plug-n-Play
    - Firmware & BIOS extensions

Hardware interfaces (2)

- Some examples
  - `lsscsi`
  - `lsscsi -L` *a:b:c:d*
  - `lspci`
  - `lspci -t`
  - `lspci -vvs` *xx:yy*
  - `lsusb`
  - `lsusb -t`
  - `lsusb -vvs` *x:y*



Example: Bus

- PCI
  - Mostly also applies to AGP, PCI-X, PCI Express (software point of view), ExpressCard
  - Configuration space (hardware registers)
    - Used to probe, inquiry and configure devices which are not yet configured
      - No ports, no registers mapped to memory, no interrupts
    - 64 standardized registers
      - Identification (vendor ID, device ID, subsystem vendor ID, subsystem device ID)
      - Resource requirements (memory ranges, port ranges, interrupts, flags, etc.)
    - More registers for vendor-specific functions

Example: Bus (2)

- DMI (Direct Media Interface)
  - Point-to-point interconnection between Intel north bridge (CPU) and south bridge (I/O)
  - AMD UMI (Unified Media Interface)
  - Somehow similar to PCI Express
    - Serial point-to-point links
    - Multiple lanes possible
    - DMI 2.0: 20 Gb/s (x4 link)

Example: Bus (3)

- LPC Bus (Low Pin Count Bus)
  - Interconnection between south bridge and Super I/O controller
    - PS/2, serial port, parallel port, floppy controller
    - Technically an ISA replacement
      - ISA: 16b, 8.33 MHz
      - LPC: 4b, 33.3 MHz
      - Replacement of legacy ISA devices
        - Interrupt controller, DMA controller, keyboard controller, etc.

Example: Bus (4)

- USB
  - Configured via an USB host controller
    - Asymmetric (host vs. device)
      - Actually not a *bus*, but a star-tree-topology
    - Usually acts as a PCI(something)-to-USB bridge
      - Devices not accessible directly
      - Communication via messages and stream pipes between controller and endpoints (serial configuration)
      - Initiated by the end driver, but managed by the controller driver
    - Vendor ID, product ID, device descriptor, interfaces, endpoints
    - Each device can act as an USB hub
      - Not transparent to the software

Example: Bus (5)

- FireWire (IEEE 1394)
  - Symmetric serial bus
    - Each node can be the bus arbiter
    - True peer-to-peer communication
      - The FireWire host controller is no special
    - Up to 63 peripherals
      - Physically a tree-chain topology
  - Configuration
    - Device type, well-known supported protocols
    - Each device has a unique EUI-64 identifier
      - Used for bus addressing purposes

Example: Bus (6)

- SCSI
  - Asymmetric parallel bus
    - True electrical bus with shared media (needs terminators on the end of the ribbon cable)
      - One level of nesting is allowed
    - Each device identified by BUS:ID:LUN
      - Manual or semi-manual (enclosure) configuration
    - Communication using commands
      - Fixed-size Command Descriptor Blocks send over the bus
        - Standard commands (Test Unit Ready, Sequential Read, Seek, etc.)
        - Vendor-specific commands
      - Command reordering

Example: Bus (7)

- SAS
  - Serial Attached SCSI
  - Point-to-point connections between the controller and endpoints
    - Actually no longer a bus, more like USB
    - Solves issues with terminators, manual configuration
  - On the high level compatible with parallel SCSI
    - Same device identification & standard SCSI commands
  - Electrically compatible with SATA
    - Same connectors
    - SAS controllers capable of driving SATA devices

Example: Bus (8)

- iSCSI
  - Actually not hardware at all
  - SCSI commands encapsulated in IP packets
    - Mostly over UDP
  - Target (represents SCSI endpoint)
  - Initiator (represent s SCSI client)

Example: Serial interconnect

- RS-232
  - (Mostly) slow serial bi-directional data communication
    - Robust electrical specification
      - Logical zero: +3 .. +15 V
      - Logical one: -15 .. -3 V
    - Robust signal timing
      - Asynchronous (but usually synchronous with an explicit clock signal for higher communication speeds)
    - Data transmission lines (TxD, RxD)
    - Control lines
      - Request to Send (RTS), Clear to Send (CTS), Data Terminal Ready (DTR), Data Set Ready (DSR)

Example: Serial interconnect (2)

- RS-232 (2)
  - Serial controller (NS 6550)
    - Computer endpoint of communication
    - Electrically challenging
    - Generates interrupts on each received character
  - Intelligent serial controllers (NS 16550)
    - (Small) FIFO buffer for sending and receiving
      - Automatic (configurable) flow control
      - Improving issues with interrupt handler latency at higher speeds

Example: Serial interconnect (3)

- Serial modem
  - Using analogue phone lines for point-to-point serial data connection
    - Various modulation and demodulation schemes
      - Overhead due to parity bits, flow control, etc.
      - Raw character transfer rate (baud) limited by the quality and specified properties of the phone lines
        - Signal range of some 4 kHz
        - Effective data transfer rate 28.8 Kb/s
        - With sophisticated modulation up to 33.3 Kb/s
        - With asymmetric modulation up to 56.6 Kb/s
          - On subscriber lines connected to digital switching board
  - Hayes (AT) command set (control language)

Example: Serial interconnect (4)

- Serial modem (2)
  - Computer interface
    - Originally connected to a serial port (*external* modems)
      - The low speed was not an issue
    - Later *internal* modems
      - Still behaved like a serial port
    - Later *controller-less* modems
      - Modulation and demodulation process executed in a reprogrammable DSP processor
    - Later *software* (*win-*) modems
      - Several phases (if not all) of modulation and demodulation process executed by the system CPU
      - The modem is nothing more than a single-purpose sound card

Example: Serial interconnect (5)

- Keyboard
  - Simple serial-like connection
    - Sending *scan codes* (sequence of bytes)
    - Control lines for LEDs, reset, etc.
  - Originally usually connected to a dedicated keyboard controller
    - On the system bus
      - Intel 8042
        - Actually a fairly complicated devices with many side-effects (system reset)
    - Synchronization, mode management, hardware character buffer, auto-repeat (delay, speed)

Example: Serial interconnect (6)

- Mouse
  - Bus mouse
    - Connected directly to a system bus (dedicated controller)
  - Serial mouse
    - Connected to a serial port
    - Sends events as a stream of bytes (usually *X*, *Y* deltas)
      - Multiple protocols (Genius, Microsoft)
      - Problems with synchronization, communication parameters
  - PS/2
    - Standardized serial protocol, commands for the mouse (reset)

Example: Parallel interconnect

- Original Centronics
  - Uni-directional asynchronous point-to-point transmission of 8 (later 16) bits per character
    - Speeds up to 1 MB/s
      - For printers
    - Cable length limitation (~ 1 m)
    - Only control flow and status lines in the other direction
- IEEE 1284
  - Bi-directional extension (backward compatible)
  - Scanners, external drives, SCSI commands

Example: Audio

- Basically a simple stream interface
  - Each sample passes a D/A or A/D converter
    - Usually PCM encoding
      - Sometimes simple stream differential compression (u-law, etc.)
      - Sometimes advanced audio encoding (DTS)
    - Multiple channels interleaved
  - Very isochronous timing required
    - Usually DMA or bus mastering is used
  - Auxiliary features
    - Analog mixers
    - MIDI interface (serial, low latency)

Example: Clocks

- RTC (Real-Time Clock)
  - Wall-clock time
  - Permanent (battery backed)
  - Coarse granularity
- Timer (Alarm Clock)
  - Scheduling, timing, accounting, alarms, watchdog, wake-up, profiling (calendar, events)
  - Fine precision
    - Intel 8253 & 8254
      - Two or three independent pins
      - Default divisor 65536 yielding 18.2 Hz (up to ~ 1 MHz)

Example: Clocks (2)

- Passive counters
  - Performance counters
  - Accurate with the frequency of the CPU
    - Also reasonable scale (64b)
    - Actual frequency may change (throttling)
  - No interrupts
- Dedicated watchdogs
  - As timer reaches 0, a non-maskable interrupt (or reset) is triggered

Example: Video

- Terminal-based
  - Command interface, character communication
    - Usually connected via simple serial, parallel or stream lines
    - Plain characters with (ASCII) control sequences
    - Escape sequences for advanced commands
      - Interpreted by the displaying terminal (e.g. ANSI, VT100)
      - **ESC**[2J
  - Usually coupled with input
    - Plain characters as input
    - Control characters and escape sequences for non-printable keys (backspace, cursor movement, etc.)

Example: Video (2)

- Matrix-based
  - Each element of the display matrix accessible randomly
    - Characters (e.g. VGA text mode), pixels (all others)
  - Older devices
    - Various strange ways how to write/read pixel values (I/O address space, bit planes, memory banking, etc.)
    - Various advanced controlling possibilities
      - Split modes
      - Interrupt on scanline
      - Interrupt on blanking signal
      - Hardware sprites

Example: Video (3)

- Common devices
  - Pixels trivially mapped into physical memory in a linear fashion
    - Single graphics mode at a time
    - Pseudo-color modes (index into a palette)
    - Direct-color (true-color) modes (pixel value directly represents RGB)
  - Some still common features
    - Page flipping
    - Hardware cursor (single sprite)
    - Hardware overlay (scaling, color space conversion)

Example: Video (4)

- Modern devices
  - Pixels still mapped into physical memory
    - Sometimes not linear, but tiled fashion
  - Video RAM used also for storing other data
    - Z-buffer, scene description (polygons, lights), textures, bump maps, stencil buffers, etc. (3D rendering)
      - A full-featured memory management is used
    - The device can use main memory for storing additional data
      - Static allocation (including frame buffer)
      - Dynamic allocation (AGP aperture, PCI-E bus mastering)
  - Complex and sometimes programmable GPU
    - Video decoding, 3D rendering
    - General-purpose programming (CUDA, OpenCL, etc.)

Example: Parallel interconnect

- IDE, (Parallel) ATA/ATAPI
  - 16 (later 32) bit parallel connection
    - Very simple bus addressing (1 bit master/slave)
  - Evolved slowly from original PC disk connections
    - Many revisions and incremental improvements
      - CHS: Original addressing scheme (cylinder/head/sector)
      - LBA: Linear (virtual) sector addressing
      - ATA: Command Block Registers
      - ATAPI: SCSI commands in the data “packets”
      - PIO: Command Block Registers used for data transfer
      - DMA: DMA controller used for reading/writing data
      - Multiword DMA: Transfer two words in one DMA transaction
      - Ultra DMA: Double the physical transfer rate per clock tick

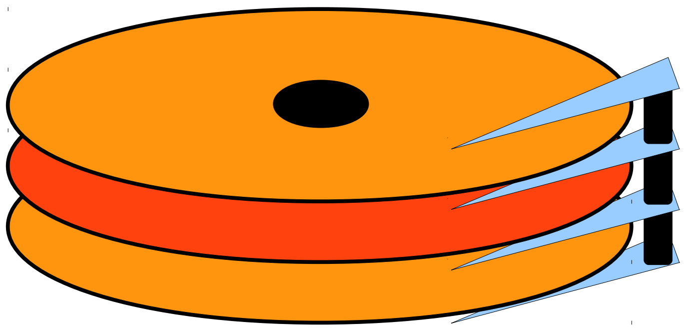
Example: Serial interconnect (7)

- SATA
  - Serial, point-to-point connection
  - Derived from ATAPI
    - Simple configuration (no master/slave), more devices per controller
    - Simpler cabling, longer cables, better robustness
    - Mostly ATAPI compatible low-level command set
      - In legacy mode even the SATA controller emulates ATAPI Command Block Registers
    - Command queueing & reordering
    - External connectors (eSATA)

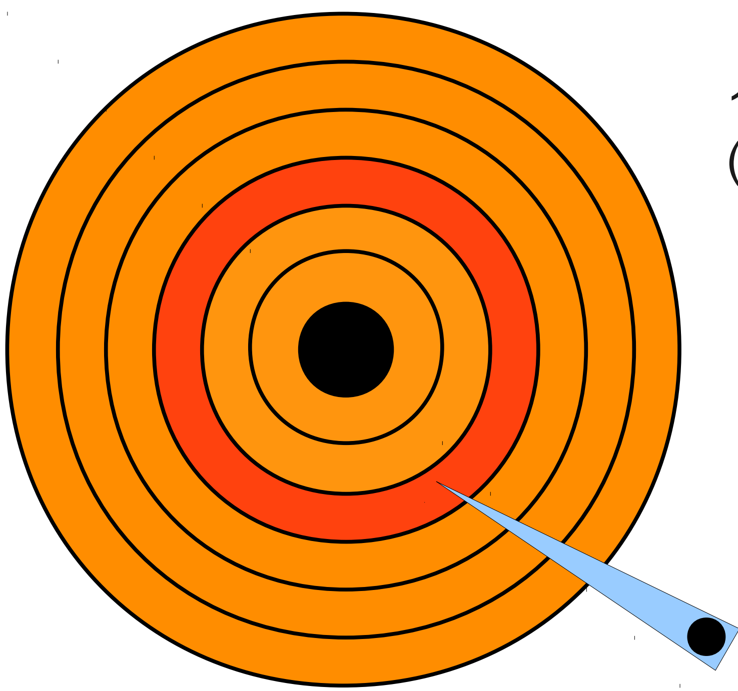
Example: Disks

- Block devices
  - Random addressing of fixed-size blocks
    - 512 bytes for most hard disks (4096 more recently)
    - 2048 bytes for CD-ROM (mode 2), 2352 bytes physically
  - Cylinder-Head-Sector (CHS) addressing
    - Reflecting physical hard disk layout
    - Original IDE (PATA), floppy, some legacy APIs (BIOS)
  - Linear Block Addressing (LBA)
    - Linearized view of the sectors on the device
    - Works for both hard disks (soft-sectored, internal remapping), CDs (blocks in spiral) and SSD

Example: Disks (2)

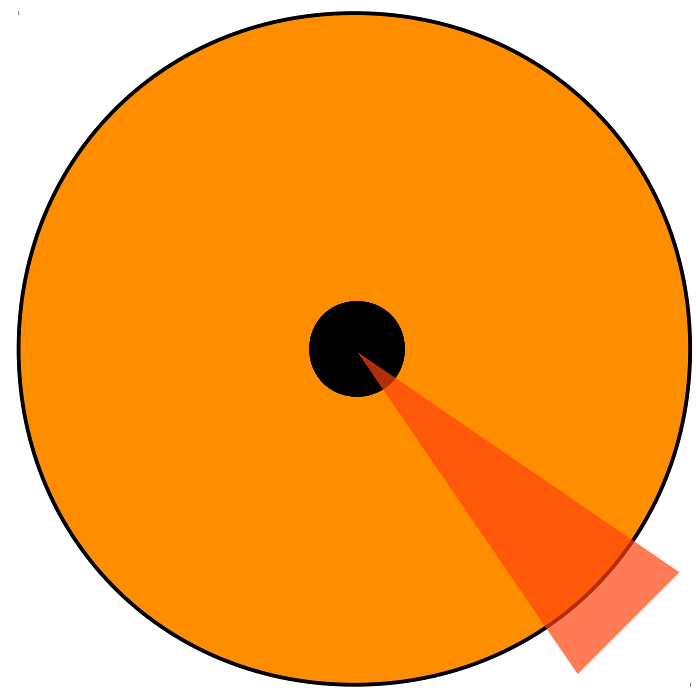


16 heads



1024 cylinders
(tracks)

63 sectors



interleaving factor (1)

Example: Disks (3)

- SSD
  - Flash memory chips (floating-gate transistors)
    - Single-level Cells, Multi-level Cells
    - NOR chips
      - Write bit-by-bit
      - Erase blocks 64 – 256 KB
    - NAND chips
      - Greater integration, more fault tolerant
      - Write pages 512 – 4096 bytes
      - Erase blocks 64 – 512 KB
  - Memory wear
    - Internal block wear leveling
    - Specially designed file systems (JFFS2)

Example: Disks (4)

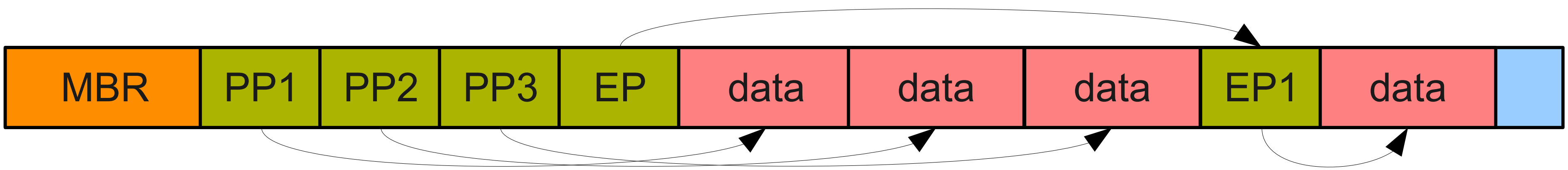
- Request queuing
  - FIFO
    - Extensive seeking probable
  - Shortest Seek First
    - Distant requests might starve
  - Bidirectional Elevator
    - Distant requests are allowed to starve only two passes
  - Unidirectional Sweep
    - Distant requests are allowed to starve only one pass
  - SATA Native Command Queueing
    - First party DMA (transfer of data of individual requests)

Example: Disks (5)

- Request schedulers
  - Implemented in software
    - Might consider hardware unrelated data
  - Linux anticipatory scheduler
    - Unidirectional sweep
    - Allows to serve close requests in opposite direction
    - Upper limit on request starve time
    - Read and writes separate, anticipating more read requests
  - Linux deadline scheduler
    - Unidirectional sweep with strict deadline for each request
  - Linux Complete Fairness Queueing (CFQ)
    - Serving requests in a weighted round robin by process priorities

Example: Disks (6)

- Partitioning
  - Standard partition tables
    - Master Boot Record



    - GUID Partition Table
    - BSD Disk Label
    - Sun Disk Label
    - Apple Partition Table
  - Logical volumes
    - Physical volumes
    - Volume groups
    - Logical volumes

Example: Disks (7)

- Fault tolerance
  - Error detection and correction
    - CRC, ECC (sectors, file systems)
  - Retries, controller reset
  - Bad blocks management
  - SMART diagnostics
  - Write-back caching, barriers
  - RAID
    - 1: mirroring
    - 2: bit striping & Hamming code
    - 3: byte striping & parity disk
    - 4: block striping & parity disk
    - 5: block striping & parity striping