

# 11., 12., Správa I/O Správa souborů

---

ZOS 2016, L. PEŠIČKA

# Pamatuj

---

- v C o paměť žádáme `malloc()` a máme ji uvolnit `free()`
- paměť nám přidělí knihovna – **alokátor paměti**, která spravuje volnou paměť
- pokud alokátor nemá volnou paměť k dispozici, **požádá operační** systém systémovým voláním o přidělení další části paměti (další stránky)

# Alokace paměti pro procesy

---

- explicitní správa paměti
  - Programátor se musí postarat o uvolnění paměti
- čítání referencí
  - Každý objekt má u sebe čítač referencí
- garbage collection
  - Pokročilé algoritmy

# Čítání referencí

---

- Ke každému objektu přiřazen čítač referencí (vytvořen: 1)
- Někdo si uloží referenci na objekt – zvýšení čítače
- Reference mimo rozsah platnosti nebo přiřazena nová hodnota – snížení čítače
- Čítač na nule – uvolnění objektu z paměti
- Nevýhoda – cyklus – dva ukazují na sebe ale nic dalšího na ně
- Nevýhoda - režie

# Garbage collection (GC)

---

- automatická správa paměti
- Speciální algoritmus (Garbage Collector) vyhledává a uvolňuje úseky paměti, které již proces nevyužívá
- Zjišťování, které objekty jsou z kořene programu nedostupné, nevede na ně žádný živý ukazatel (reference)
- Nejprve se používalo čítání referencí, potom pokročilejší algoritmy

# Garbage collection

---

- sledovací algoritmy

- Přeruší běh programu a vyhledávají dosažitelné objekty

- Algoritmus Mark and Sweep

- Všechny objekty – hodnota navštíven na FALSE
- Projde všechny objekty, kam se lze dostat
- Navštíveným nastaví navštíven na TRUE
- Na závěr – objekty s příznakem FALSE lze uvolnit z paměti
- Nevýhoda – přerušení běhu programu (nejde pro realtime)
- Nevýhoda – když nechává živé objekty na místě – fragmentace paměti

- Kopírovací algoritmus

- Rozdělí haldy na dvě části (aktivní a neaktivní)
- Pokud se při alokaci nevejde do dané části haldy – provede se úklid
- Úklid – prohození aktivní a neaktivní části, do aktivní se kopírují živé objekty ze staré
- Dvojnásobná velikost haldy, potřeba kopírovat objekty, které přežijí úklid

# Garbage Collection

---

- Generační algoritmy

- Paměť do několika částí (generací)
- Objekty vytvářeny v nejmladší, po dosažení stáří přesunuty do starší generace
- Pro každou generaci – úklid v různých časových intervalech, i různé algoritmy
- Nejmladší generace se zaplní – všechny dosažitelné v nejmladší zkopírovány do starší generace

# GC - použití

---

- část běhového prostředí
- přídatná knihovna
- jazyk Java (i např. 4 druhy garbage collectorů, vše generační)
- Platforma .NET
  - [https://msdn.microsoft.com/en-us/library/ee787088\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/ee787088(v=vs.110).aspx)



.NET z předchozího odkazu:

---

There are three generations of objects on the heap:

- **Generation 0.** This is the **youngest** generation and contains **short-lived** objects. An example of a short-lived object is a **temporary variable**. Garbage collection occurs most frequently in this generation.

Newly allocated objects form a new generation of objects and are implicitly generation 0 collections, unless they are **large objects**, in which case they go on the large object heap in a generation 2 collection.

Most objects are reclaimed for garbage collection in generation 0 and do not survive to the next generation.

- **Generation 1.** This generation contains short-lived objects and serves as a buffer between short-lived objects and long-lived objects.
- **Generation 2.** This generation contains **long-lived objects**. An example of a long-lived object is an object in a server application that contains static data that is live for the duration of the process.

.NET z předchozího odkazu:

---

## Survival and promotions

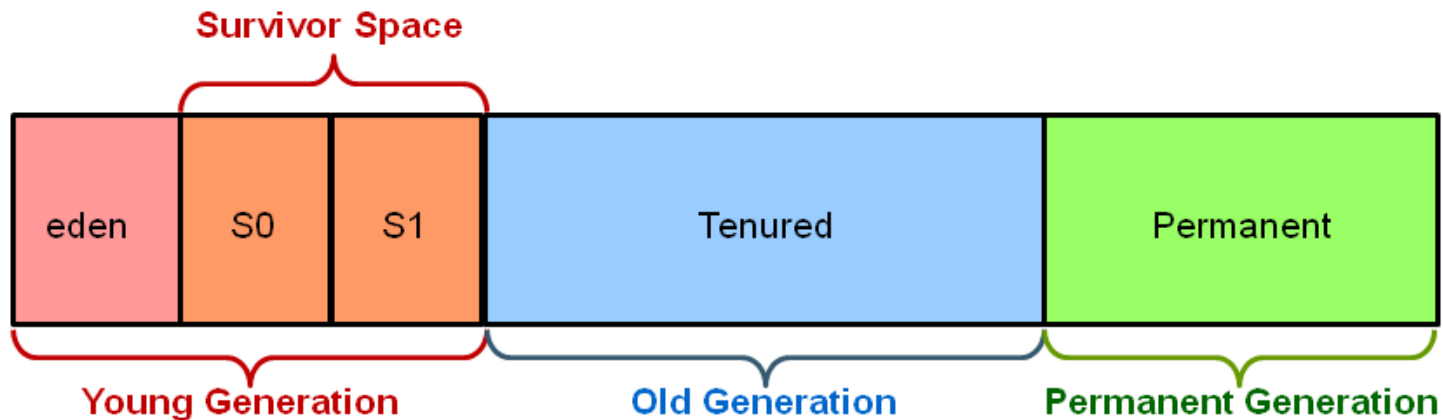
Objects that are not reclaimed in a garbage collection are known as **survivors**, and are promoted to **the next generation**. Objects that survive a generation **0** garbage collection are promoted to generation **1**; objects that survive a generation **1** garbage collection are promoted to generation **2**; and objects that survive a generation **2** garbage collection **remain in generation 2**.

When the garbage collector detects that the survival rate is high in a generation, it increases the threshold of allocations for that generation, so the next collection gets a substantial size of reclaimed memory. The CLR continually balances two priorities: not letting an application's working set get too big and not letting the garbage collection take too much time.

# Java - GC

---

## Hotspot Heap Structure



# malloc, brk, sbrk

---

- malloc není systémové volání, ale knihovní funkce  
viz `man 3 malloc` x `man 2 fork`
- malloc alokuje paměť z haldy
- velikost haldy se nastaví dle potřeby systémovým voláním `sbrk`
- `brk` – syscall – nastaví adresu konce datového segmentu procesu
- `sbrk` – syscall – zvětší velikost datového segmentu o zadaný počet bytů (0 – jen zjistím současnou adresu)

# používané vs. nepoužívané objekty

---

```
Object x = new Foo();
```

```
Object y = new Bar();
```

```
x = new Quux();
```

```
/* víme, že Foo object původně přiřazený x nebude nikdy dostupný, jde o syntactic garbage */
```

```
if ( x.check_something() )
```

```
    { x.do_something(y); }
```

```
System.exit(0);
```

```
/* y může být semantic garbage, ale nevíme, dokud x.check_something() nevrátí návratovou hodnotu */
```

# Velikost stránky v OS

---

- Standardní velikost je 4096 bytů (4KB)
- huge page size: 4MB
- large page size: 1GB

# Zjištění velikosti stránky - Linux

```
#include <stdio.h>
#include <unistd.h>

int main(void) {

    printf("Velikost stranky je %ld bytu.\n",
        sysconf(_SC_PAGESIZE) );

    return 0;

}
```

příkazem na konzoli:  
`getconf PAGESIZE`

`man sysconf`

eryx.zcu.cz: 4096  
ares.fav.zcu.cz: 4096

# Zjištění velikosti stránky - WIN

```
#include "stdafx.h"  
#include <stdio.h>  
#include <Windows.h>
```

```
int _tmain(int argc, _TCHAR* argv[]) {
```

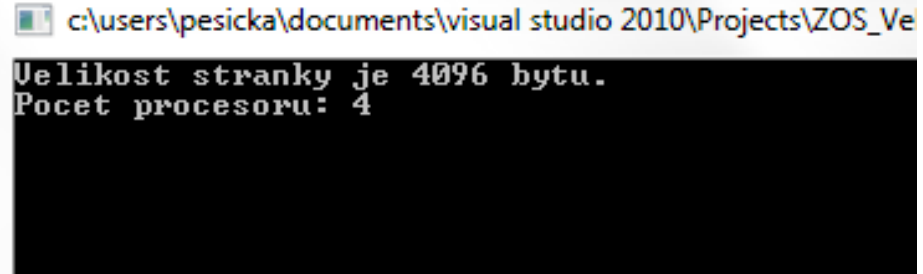
```
    SYSTEM_INFO si;
```

```
    GetSystemInfo(&si);
```

```
    printf("Velikost stranky je %u bytu.\n", si.dwPageSize);
```

```
    printf("Pocet procesoru: %u\n", si.dwNumberOfProcessors);
```

```
    getchar(); return 0; }
```



```
c:\users\pesicka\documents\visual studio 2010\Projects\ZOS_Ve
```

```
Velikost stranky je 4096 bytu.  
Pocet procesoru: 4
```



# OS

---

- Modul pro správu procesů
- Modul pro správu paměti
- **Modul pro správu I/O**
- Modul pro správu souborů
- Síťování

# Vývoj rozhraní mezi CPU a zařízeními

---

1. CPU řídí přímo periférii
2. CPU – řadič – periférie  
(aktivní čekání CPU na dokončení operace)
3. řadič umí vyvolat přerušení
4. řadič umí DMA
5. I/O modul
6. I/O modul s vlastní pamětí

# 1. CPU řídí přímo periferii

---

- CPU přímo vydává potřebné signály
  - CPU dekóduje signály poskytovaném zařízením
  - Nejjednodušší HW
  - Nejméně efektivní využití CPU
- 
- Jen v jednoduchých mikroprocesorem řízených zařízeních (dálkové ovládání televize)

## 2. CPU – řadič - periférie

---

### Řadič (device controller)

- Převádí příkazy CPU na elektrické impulzy pro zařízení
- Poskytuje CPU info o stavu zařízení
- Komunikace s CPU pomocí **registrů** řadiče na známých I/O adresách
- HW buffer pro alespoň 1 záznam (blok, znak, řádka)
- Rozhraní řadič-periférie může být standardizováno (SCSI, IDE, ...)

## 2. řadič – příklad operace zápisu

---

- CPU zapíše data do bufferu,  
Informuje řadič o požadované operaci
- Po dokončení výstupu zařízení nastaví příznak, který může CPU otestovat
- if přenos == OK, může vložit další data
- CPU musí dělat všechno (programové I/O)
- Významnou část času stráví CPU čekáním na dokončení I/O operace

# 3. Řadič umí vyvolat přerušení

---

- CPU nemusí testovat příznak dokončení
- Při dokončení I/O vyvolá řadič **přerušení**
- CPU začne obsluhovat přerušení
  - Obslužná procedura přerušení  
(Provádí instrukce na předdefinovaném místě)
  - Určí co dál
- Postačuje pro pomalá zařízení, např. sériové I/O

# 4. Řadič může přistupovat k paměti pomocí DMA (!!!!)

---

- DMA přenosy mezi pamětí a I/O zařízením
  - CPU inicializuje přenos, ale sám ho nevykonává
  - Řadič DMA – speciální obvod, zajišťuje blokové přenosy mezi I/O zařízením a pamětí
- 
1. CPU zadá požadavek řadiči DMA (adresu I/O zařízení, adresu v RAM, počet slov)
  2. DMA obvod provede přesun dat bez zásahu CPU
  3. CPU se zatím může věnovat dalším věcem (ale je omezen ve využití sběrnice)
  4. Po ukončení přenosu DMA obvod vyvolá přerušení

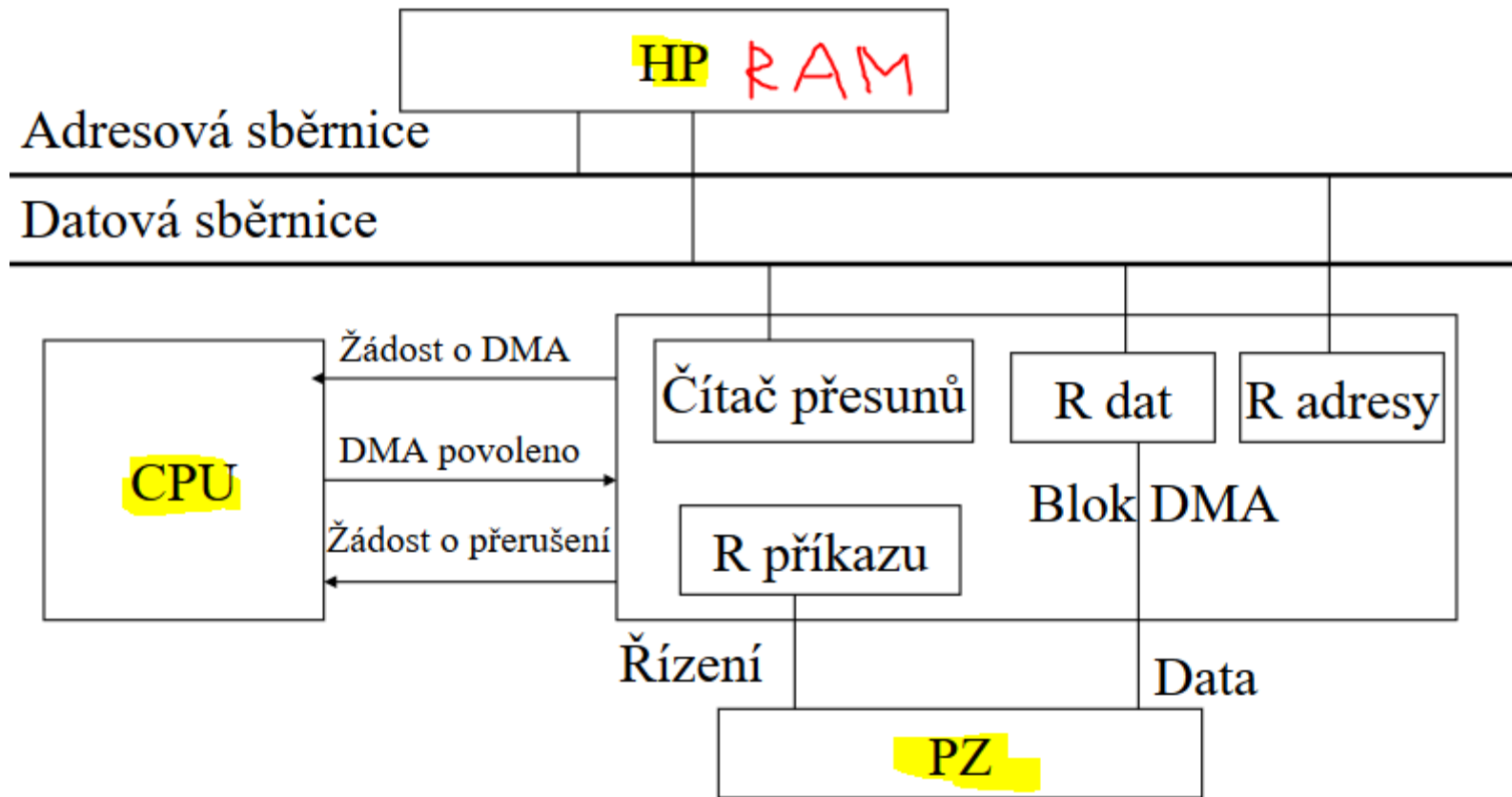
## 4. poznámky

---

- Bus mastering – zařízení převeze kontrolu nad sběrnici a přenos provede samo (PCI sběrnice)
- Vhodné pro rychlá zařízení – řadič disků, síťová karta, zvuková karta, grafická karta atd.



# DMA



R=registr

# 5. I/O modul umí interpretovat speciální I/O programy

---

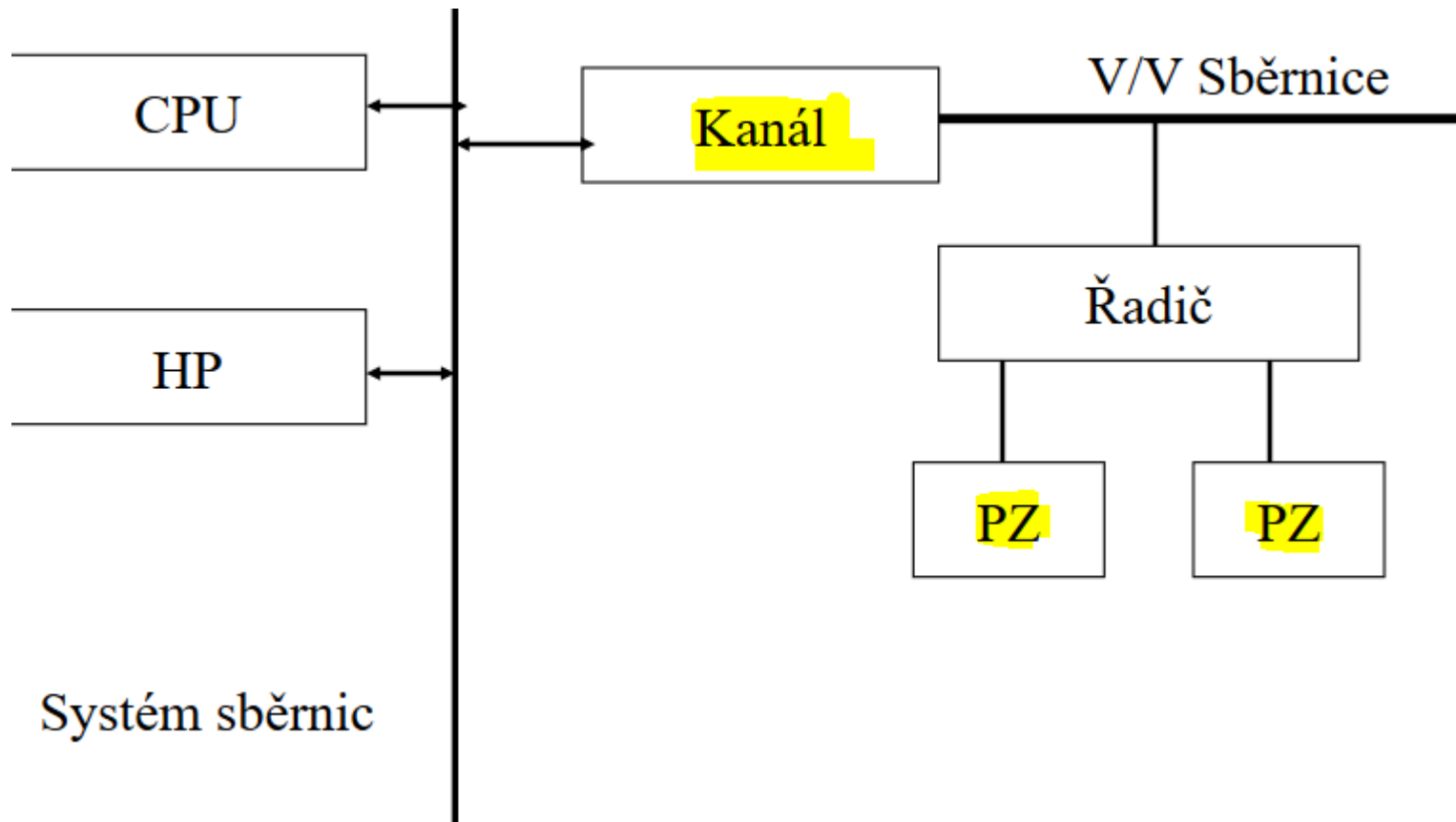
- I/O procesor
- Interpretuje programy v hlavní paměti
- CPU spustí I/O procesor  
I/O procesor provádí své instrukce samostatně

## 6. I/O modul s vlastní pamětí

---

- I/O modul provádí programy
- Má vlastní paměť(!)
  - Je vlastně samostatným počítačem
- Složité a časově náročné operace  
grafika, šifrování, ...

# Kanálová architektura (5,6)



# Komunikace CPU s řadičem

---

- Odlišné adresní prostory

- CPU zapisuje do registrů řadiče pomocí speciálních I/O instrukcí
- Vstup: **IN** R, port
- Výstup: **OUT** R, port

- 1 adresní prostor

- Hybridní schéma

# Ad – 1 adresní prostor

---

- Používá **vyhrazené adresy**
- Nazývá se *paměťově mapované I/O*
- HW musí pro dané adresy umět **vypnout cachování**
- Danou oblast můžeme namapovat do virtuálního adresního prostoru nějakého procesu (zpřístupnění I/O zařízení)

# Ad – hybridní schéma

---

- Řídící registry
  - Přístup pomocí I/O instrukcí
- HW buffer
  - Mapován do paměti
- Např. na PC  
(buffery mapovány do oblasti 640K až 1MB)

# RAID

---

- pevný disk
  - elektronická část + mechanická
  - náchylnost k poruchám
  - cena dat >> cena hw
- odstávka při výměně zařízení
  - náhrada hw, přenos dat ze zálohy - prostoj
  - SLA 24/7
- větší disková kapacita než 1 disk
- RAID
  - Redundant Array of Independent (Inexpensive) disks



# Disk

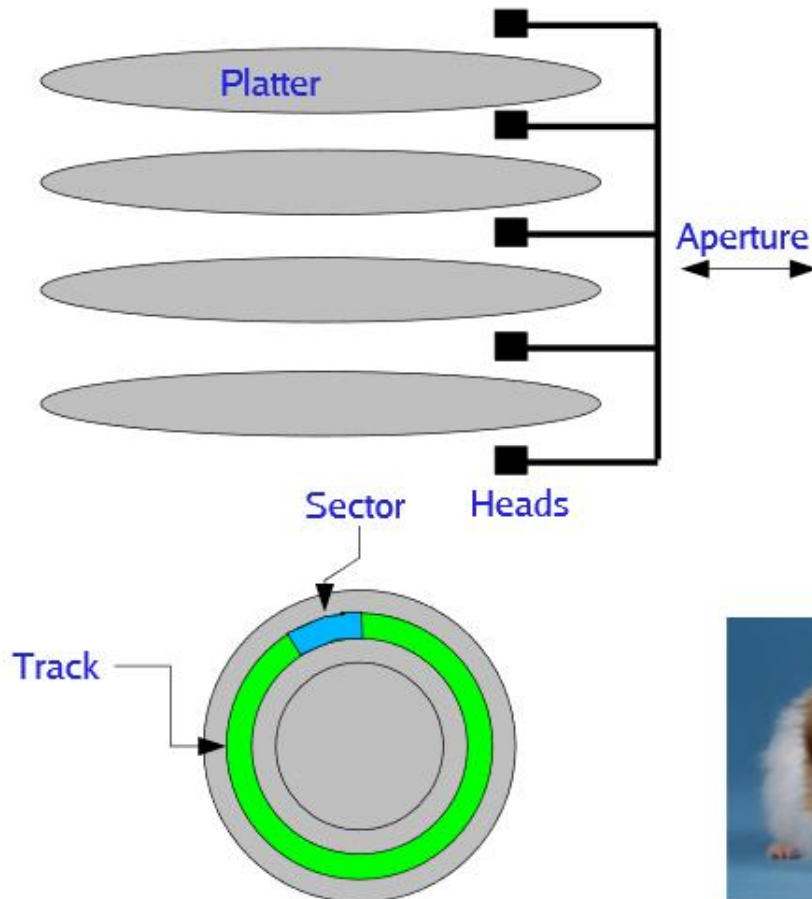
---

- Rotační disky
  - doba vystavení + rotační zpoždění
  - Stopa, sektor
- SSD disky
  - dražší, menší kapacita
- Mix
  - SSD disk v kombinaci s rotačním diskem

# A Disk Primer

Disks consist of one or more *platters* divided into *tracks*

- Each platter may have one or two *heads* that perform read/write operations
- Each track consists of multiple *sectors*
- The set of sectors across all platters is a *cylinder*



# SSD disk

---

- Vyšší cena za stejnou kapacitu
- Nemá pohyblivé části
- Rychlá přístupová doba (0.1ms)
- Může číst a zapisovat jednotlivé bajty
- Ale musí vymazat celý blok, než do něj zapíše
- Omezený počet cyklů mazání/zápisu

# RAID – používané úrovně

---

- RAID 0, 1, 5
- RAID 10 .. kombinace 0 a 1
- RAID 6 .. zdvojená parita
- pojmy:
  - SW nebo HW RAID
  - hot plug
  - hot spare
  - Degradovaný režim – jeden (či více dle typu RAIDu) z disků v poli je porouchaný, ale RAID stále funguje

# RAID 0

Dva režimy RAID 0:  
zřetězení a prokládání

- není redundantní, neposkytuje ochranu dat
- ztráta 1 disku – ztráta celého pole nebo části (dle režimu)
- důvod použití – může být výkon při režimu prokládání (např. střih videa)
- Dva režimy – zřetězení nebo prokládání (stripping)

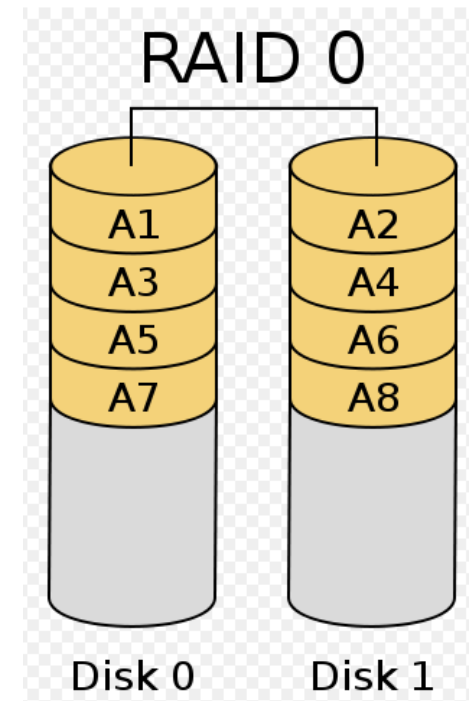
# RAID 0

## ■ Zřetězení

- Data postupně ukládána na několik disků
- Zaplní se první disk, pak druhý, atd.
- Snadné zvětšení kapacity, při poruše disku ztratíme jen část dat

## ■ Prokládání

- Data ukládána na disky cyklicky po blocích (stripy)
- Při poruše jednoho z disků – přijdeme o data
- Větší rychlost čtení / zápisu
  - Jeden blok z jednoho disku, druhý blok z druhého disku



Na obrázku je režim prokládání, zdroj: wikipedia (i u dalších obrázků)

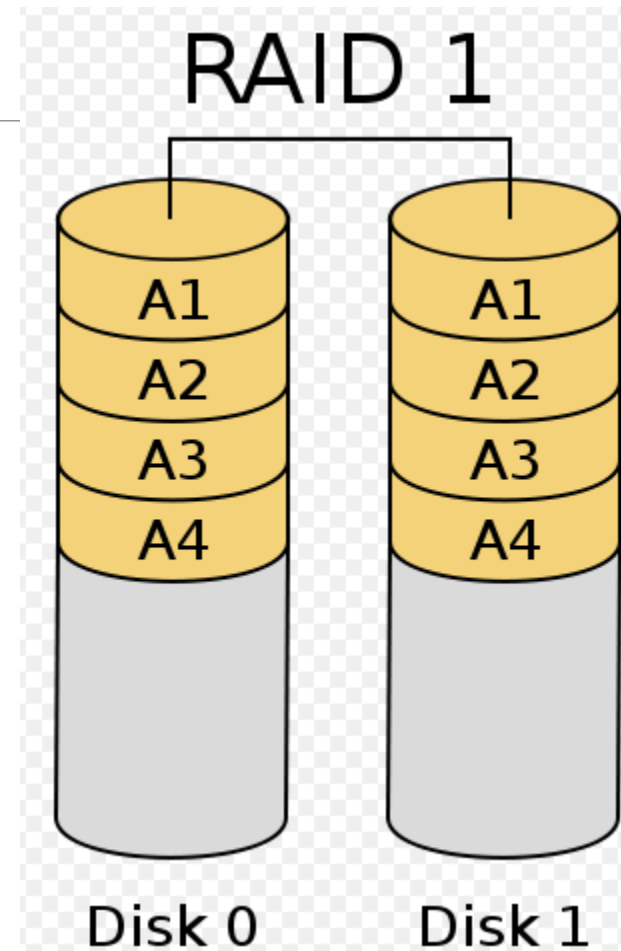
# RAID 1

---

- mirroring .. zrcadlení
- na 2 disky stejných kapacit totožné informace
- výpadek 1 disku – nevadí
- jednoduchá implementace – často čistě sw
- nevýhoda – využijeme jen polovinu kapacity
- zápis – pomalejší (stejná data na 2 disky)  
ovlivněn diskem, na němž bude trvat déle
- čtení – rychlejší  
(řadič - lze střídat požadavky mezi disky)

# RAID 1

---



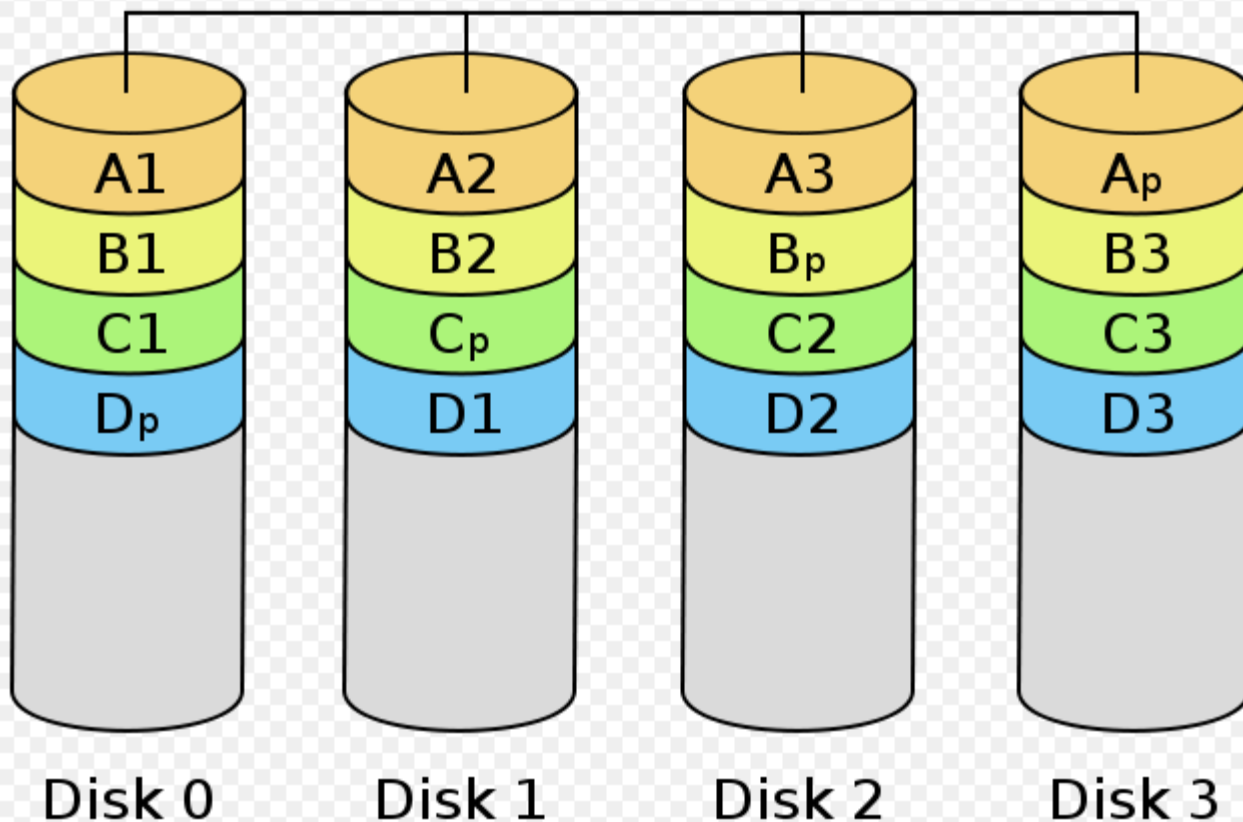


# RAID 5

---

- redundantní pole s distribuovanou paritou
- minimálně 3 disky
- režie: 1 disk z pole  $n$  disků
  - 5 disků 100GB : 400GB pro data
- výpadek 1 disku nevadí
- čtení – výkon ok
- zápis – pomalejší
  - 1 zápis – čtení starých dat, čtení staré parity, výpočet nové parity, zápis nových dat, zápis nové parity

# RAID 5

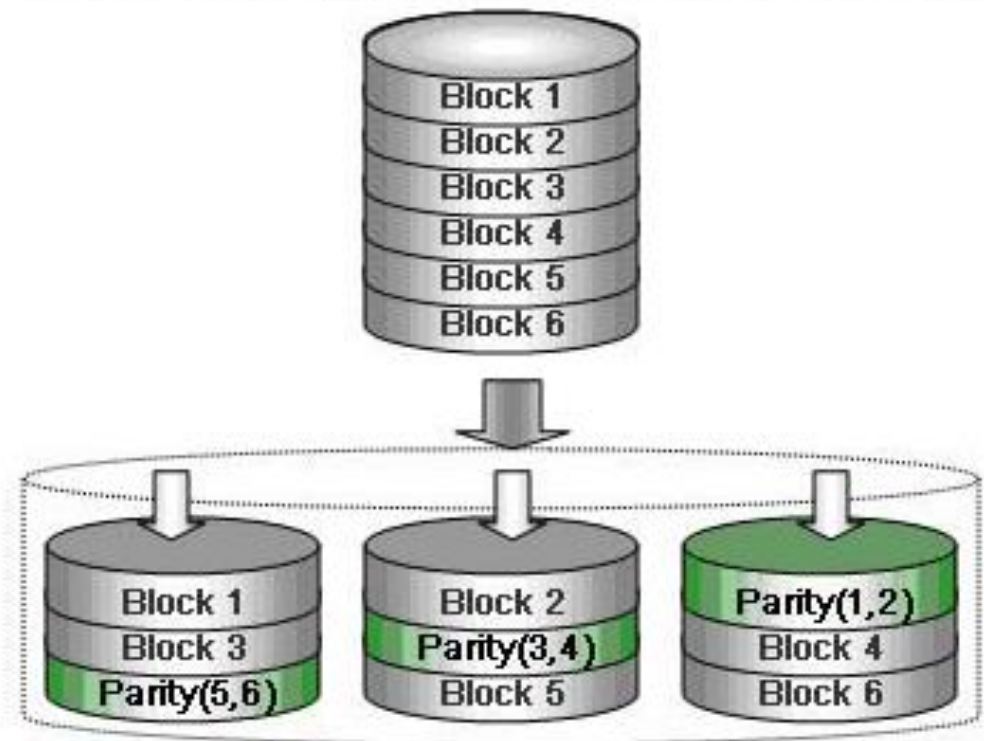


Např. RAID 5 z 4 disků 1TB, výsledná kapacita: 3 TB  
Může vypadnout 1 z disků a o data nepřijdeme

# RAID 5

---

Raid 5 - Disk Striping with Single Distributed Parity



Zdroj:

<http://www.partitionwizard.com/resize-partition/resize-raid5.html>

# RAID 5

---

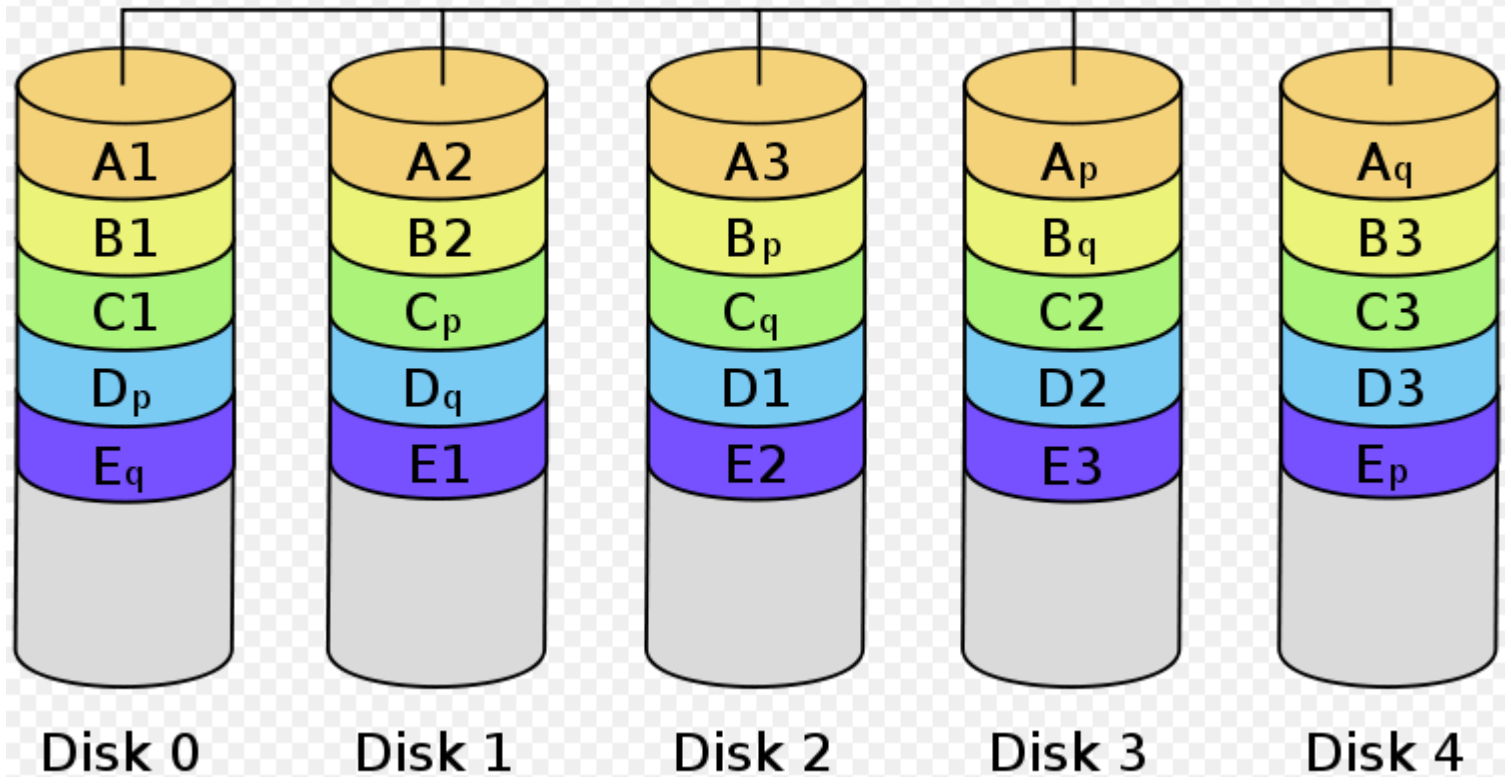
- nejpoužívanější
- detekce poruchy v diskovém poli
- hot spare disk
- použití hot plug disků

# RAID 6

---

- RAID 5 + navíc další paritní disk
- odolné proti výpadku dvou disků
- Rychlost čtení srovnatelná s RAID 5
- Zápis pomalejší

# RAID 6



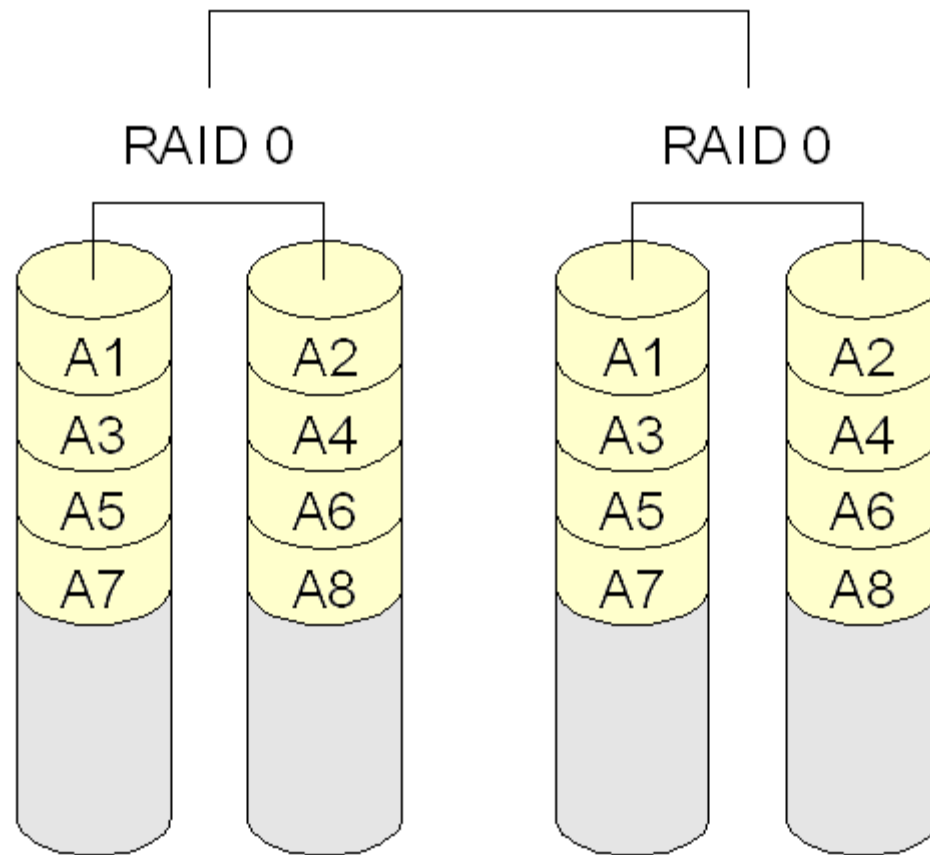
# RAID 10

---

- kombinace RAID 0 (stripe) a RAID 1 (zrcadlo)
- min. počet disků 4
- režie 100% diskové kapacity navíc
- nejvyšší výkon v bezpečných typech polích
- podstatně rychlejší než RAID 5, při zápisu
- odolnost proti ztrátě až 50% disků x RAID 5

RAID 0+1

RAID 1

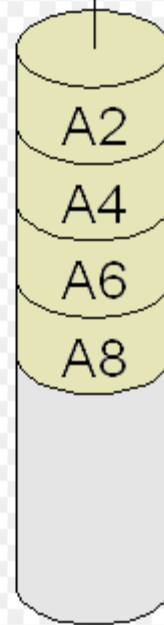
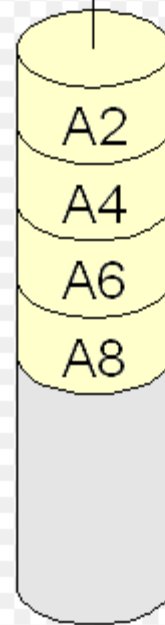
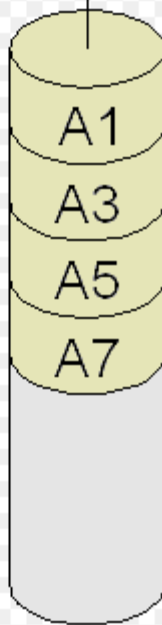
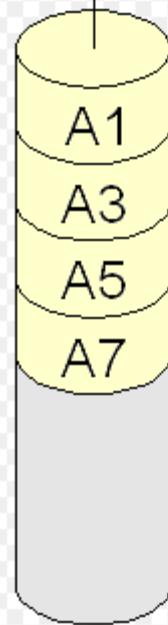




RAID 10  
RAID 0

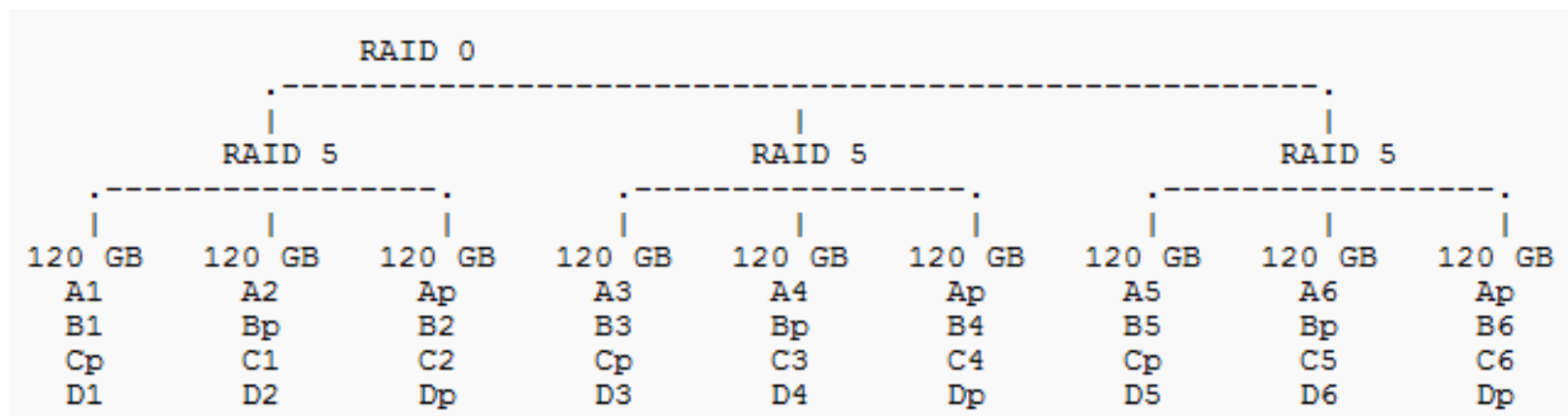
RAID 1

RAID 1



# RAID 50

---



Zdroj obrázků a doporučená literatura:

<http://cs.wikipedia.org/wiki/Raid>

# RAID 2

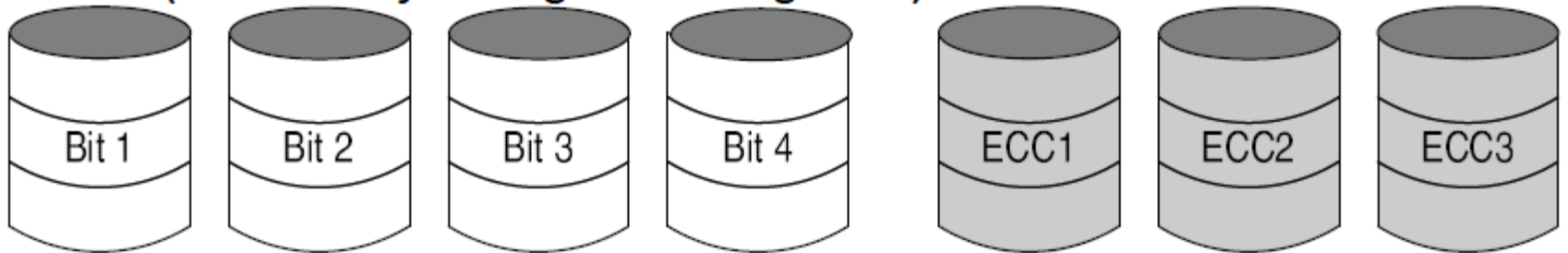
---

- Data **po bitech** stripována mezi jednotlivé disky
- Rotačně synchronizované disky
- Zabezpečení Hammingovým kódem
- Např. 7 disků
  - 4 bity datové
  - 3 bity Hammingův kód

# RAID 2

---

**RAID 2 (redundancy through hamming code)**

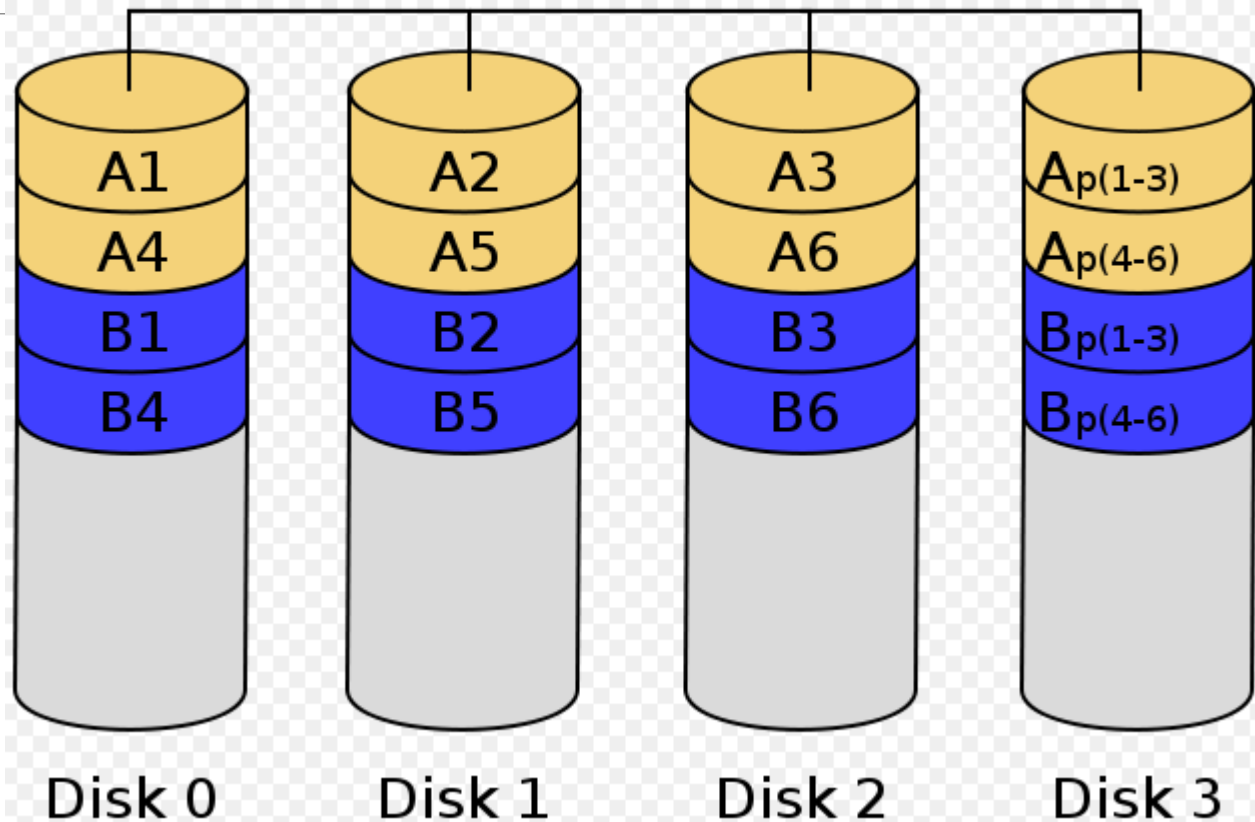


# RAID 3

---

- N+1 disků, bitové prokládání
- Rotačně synchronizované disky
- Na N data, poslední disk XOR parita
- Jen 1 disk navíc
- Paritní disk vytížen při zápisu na libovolný disk – vyšší opotřebení

# RAID 3



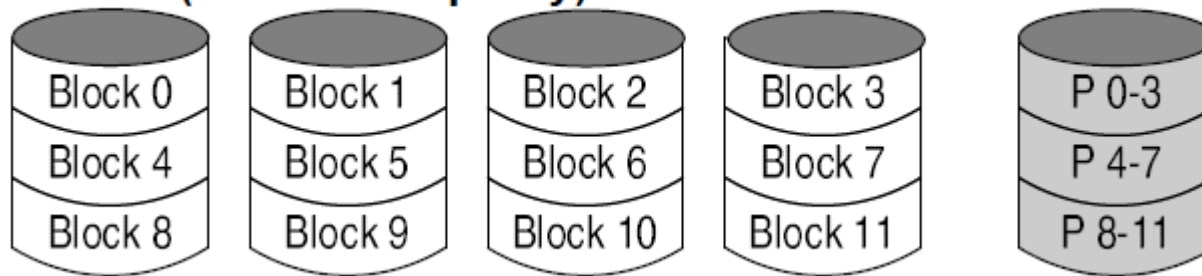
# RAID 4

---

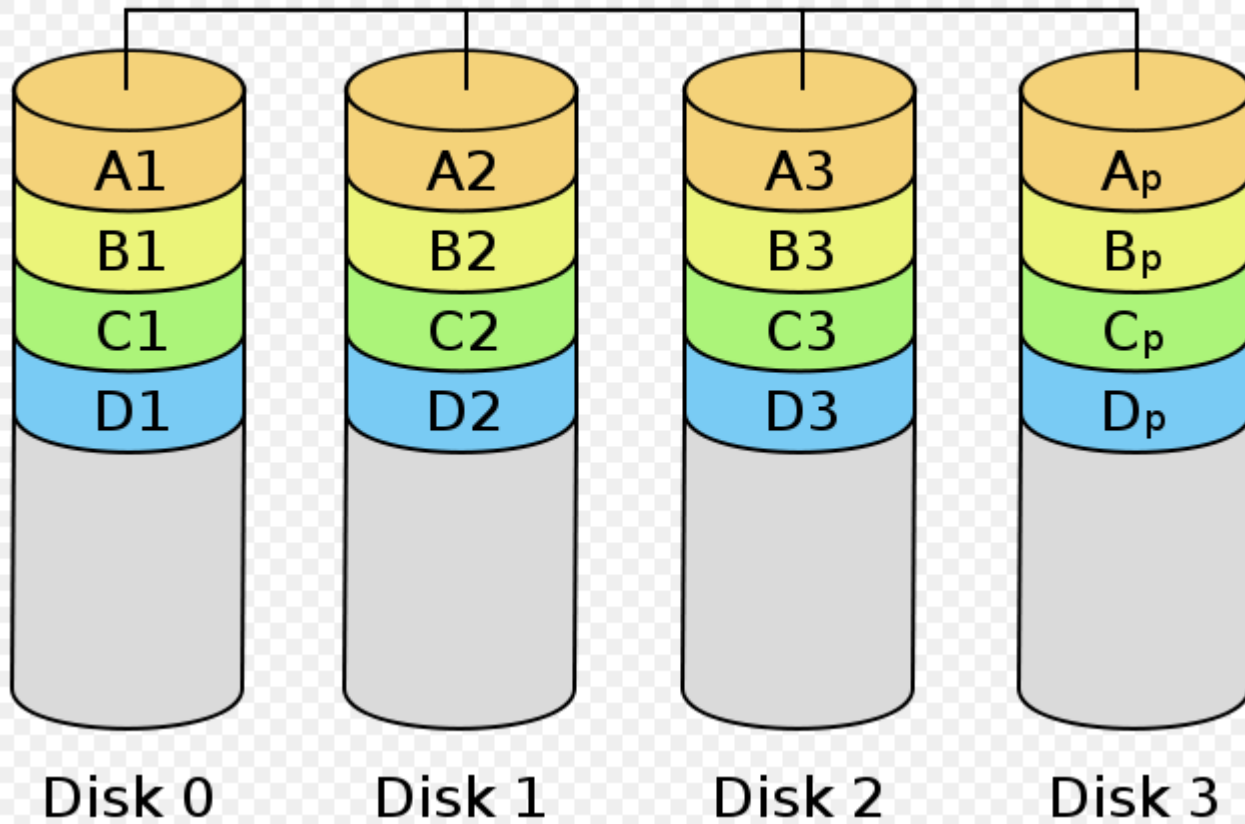
Disky stripovány po blocích, ne po bitech

Parita je opět po blocích

## **RAID 4 (block-level parity)**



# RAID 4





# Problém rekonstrukce pole

---

- rekonstrukce pole při výpadku – trvá dlouho
  - po dobu rekonstrukce není pole chráněno proti výpadku dalšího disku
  - náročná činnost – může se objevit další chyba, řadič disk odpojí a ... přijdeme o data...

# HOT SPARE DISK

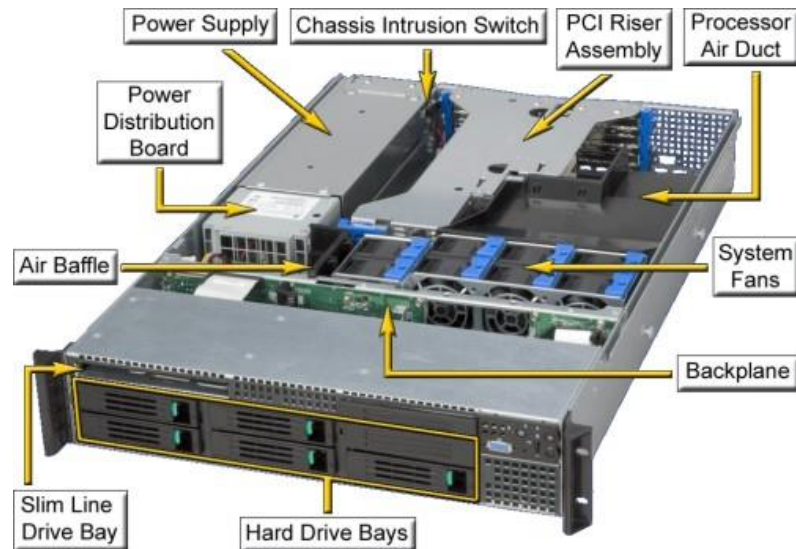
---

- záložní disk okamžitě připravený k nahrazení vadného disku
- při výpadku disku v poli automaticky aktivován hot spare disk a dopočítána data
- minimalizace rizika (časové okno)
  - Pole je degradované a je třeba vyměnit disk
  - Administrátor nemusí být poblíž
- hot spare disk lze sdílet mezi více RAIDy (někdy)

# HOT PLUG

---

- Snadná výměna disku za běhu systému
- Není třeba vypnout server pro výměnu disku
- „šuplík z přední strany serveru“





# Ukládání dat

---

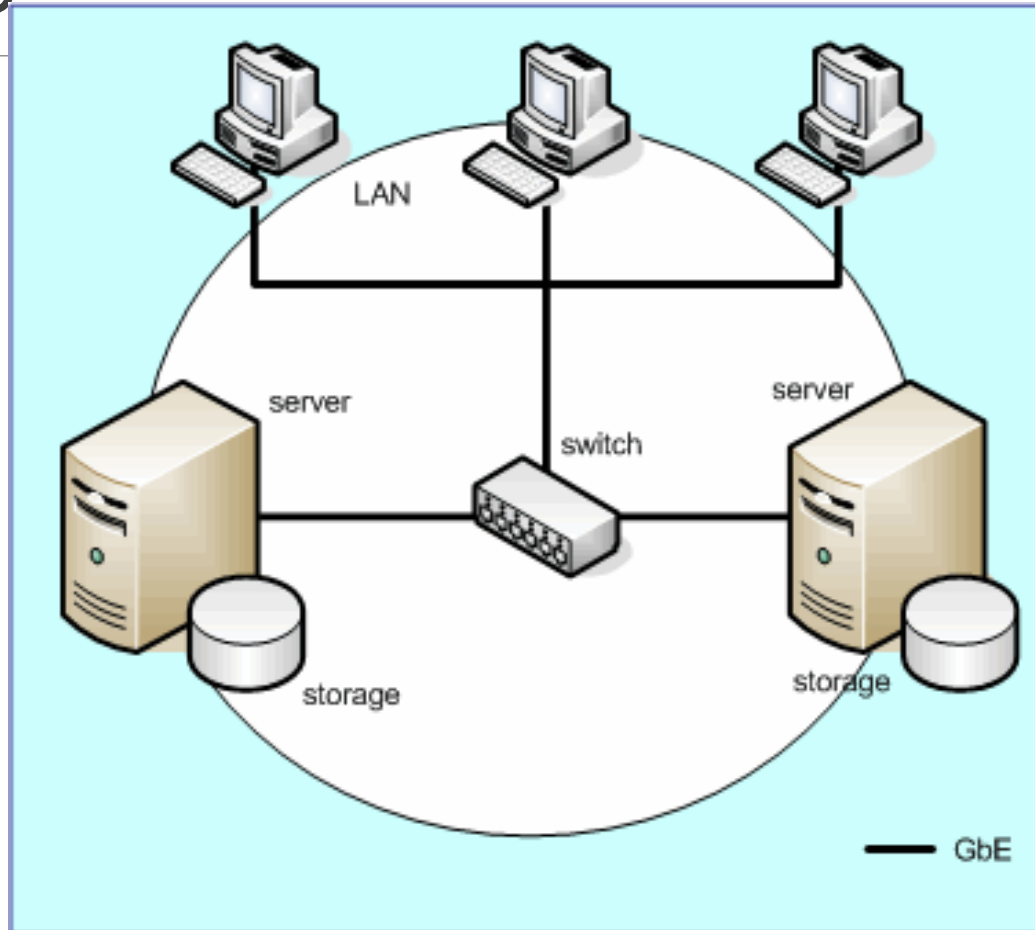
- DAS
- SAN
- iSCSI

# DAS

---

- Directly attached storage
- ukládací zařízení přímo u serveru
- nevýhody
  - porucha serveru – data nedostupná
  - některé servery prázdné, jiné – dat.prostor skoro plný
  - rozšiřitelnost diskové kapacity
- disky přímo v serveru
- externí diskové pole přes SCSI (vedle serveru)

# DAS



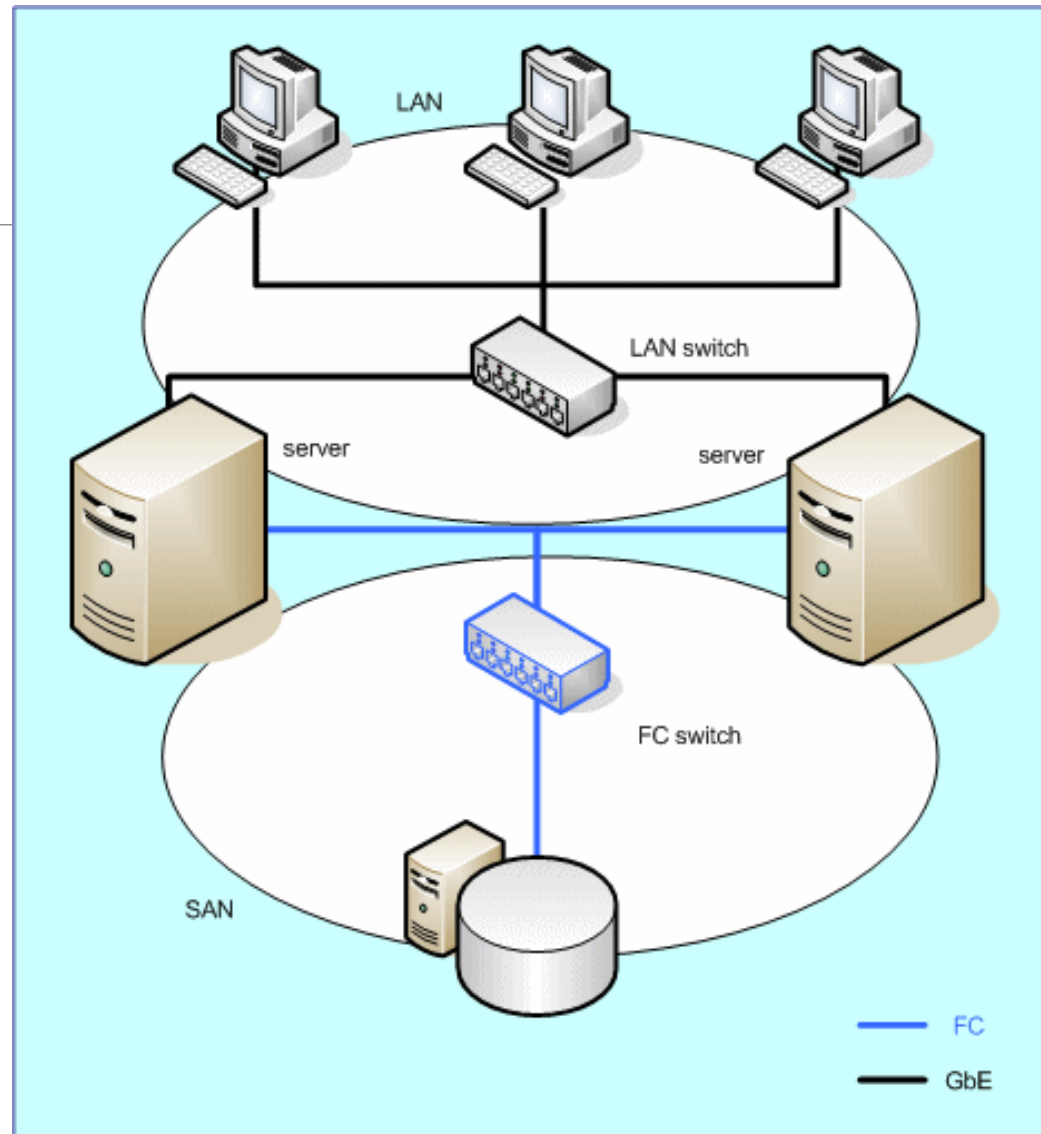
# SAN

---

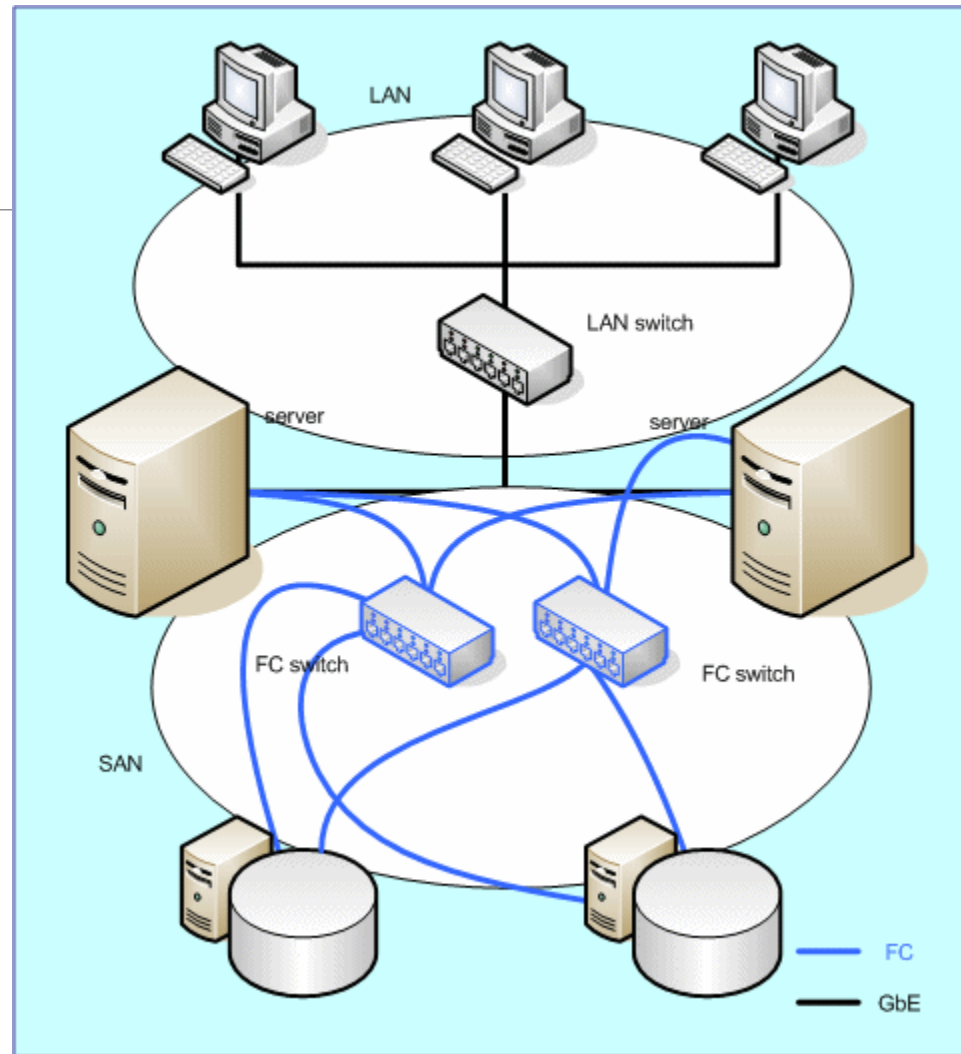
- Storage Area Network
- oddělení storage a serverů
- Fibre Channel – propojení, optický kabel
  
- např. clustery, společná datová oblast
- high availability solution



# SAN



# SAN

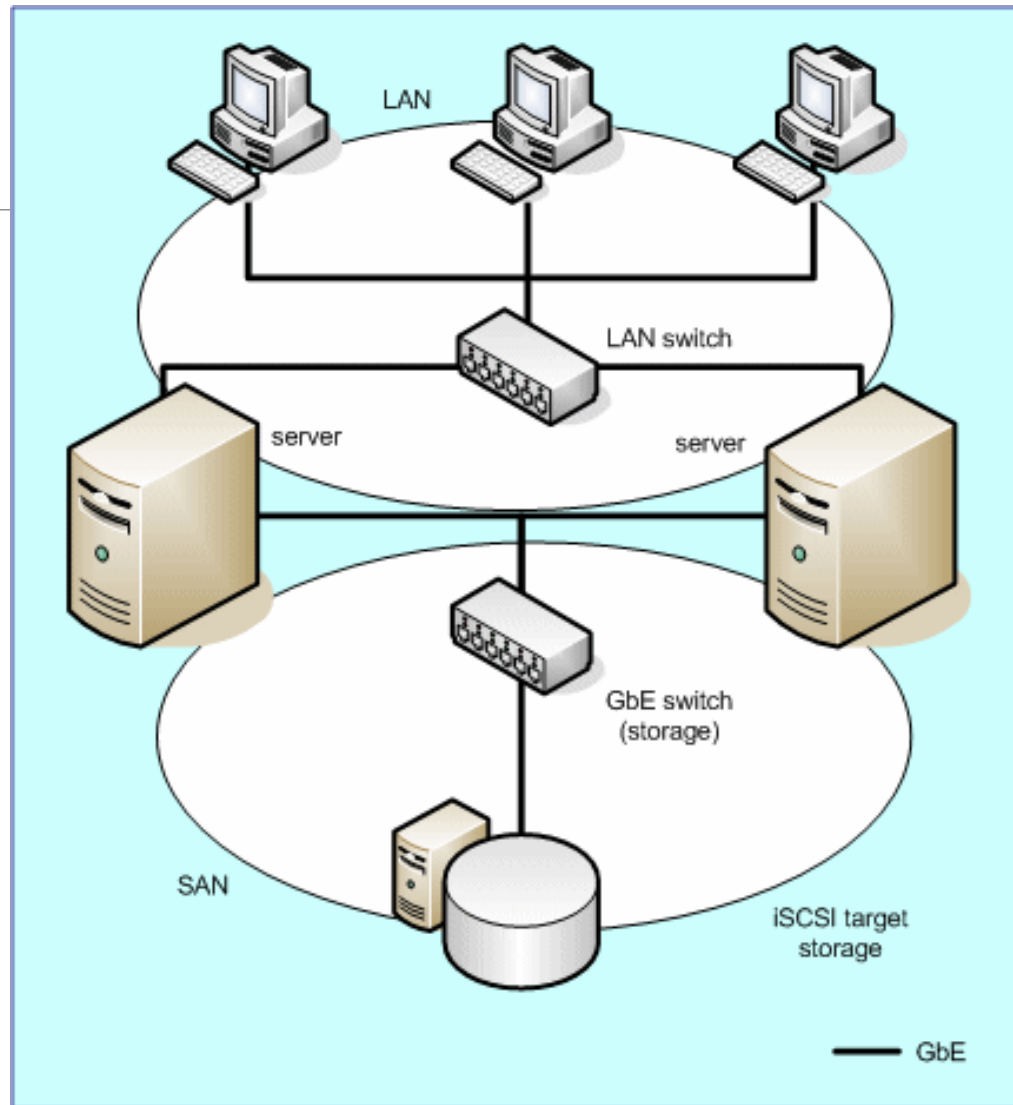


# iSCSI

---

- SCSI + TCP/IP
- SCSI
  - protokol, bez fyzické vrstvy (kabely, konektory)
  - zapouzdření do protokolů TCP/IP
- gigabitový Ethernet vs. drahý Fibre Channel
- SCSI – adaptér, disk
- iSCSI – initiator (adapter), target (cílové zař. disk/pole)

# iSCSI



# Použité odkazy a další informace

---

<http://www.vahal.cz/cz/podpora/technicke-okenko/ukladani-dat-iscsi.html>

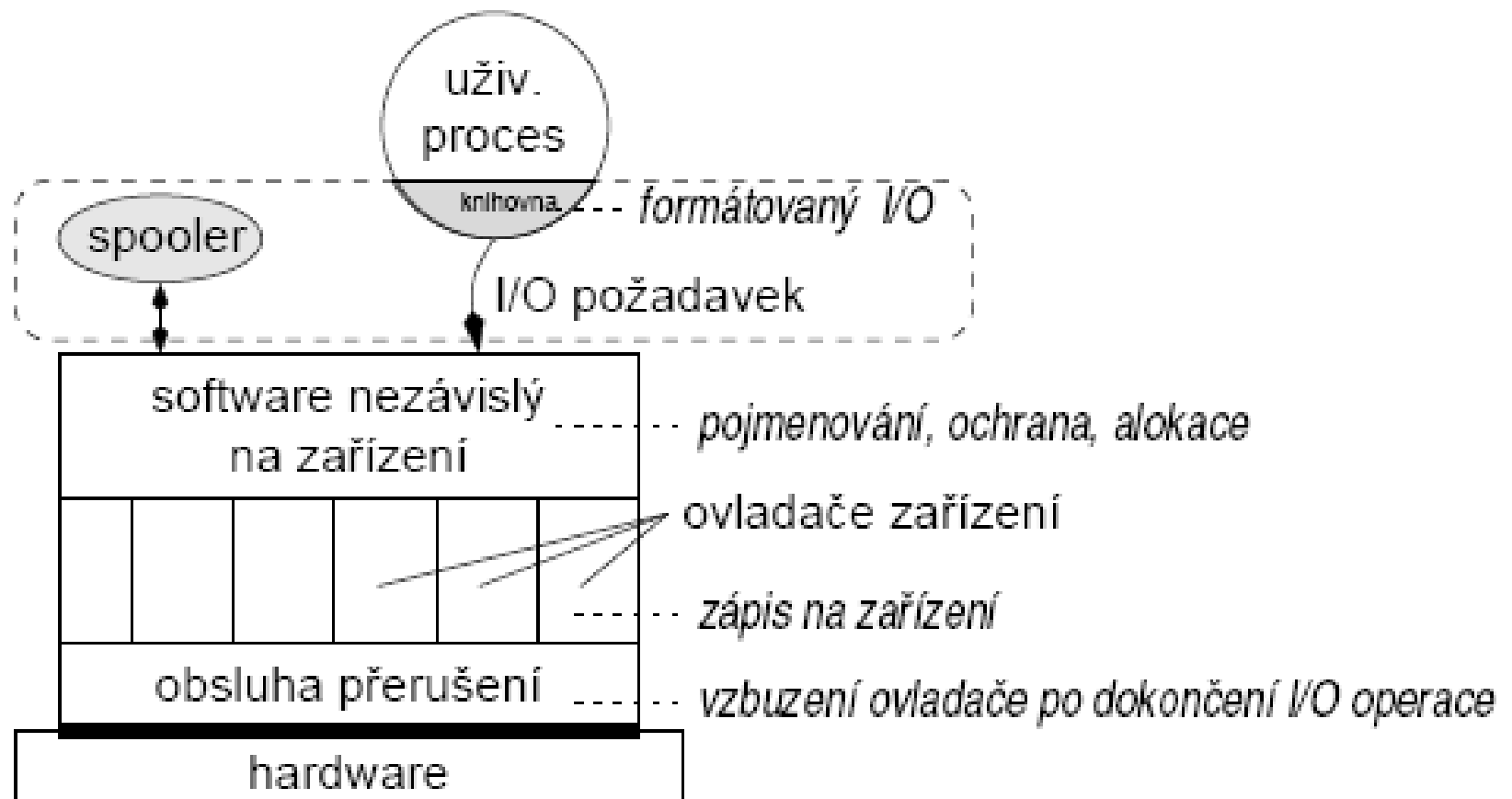
# Principy I/O software (!!!!)

---

typicky strukturován do 4 úrovní:

1. obsluha přerušení (nejnižší úroveň v OS)
2. ovladač zařízení
3. SW vrstva OS nezávislá na zařízení
4. uživatelský I/O SW

Toto je potřeba znát !!



# 1. Obsluha přerušení

---

- řadič vyvolá přerušení ve chvíli **dokončení** I/O požadavku
- snaha, aby se přerušením nemusely zabývat vyšší vrstvy
- ovladač zadá I/O požadavek, **usne** - P(sem)
- po příchodu přerušení ho obsluha přerušení **vzbudí** - V(sem)
- časově kritická obsluha přerušení – co nejkratší



## 2. Ovladače zařízení

---

- obsahují veškerý **kód závislý na konkrétním I/O zařízení** (např. ovladač zvukovky od daného výrobce)
- ovladač zná jediný hw podrobnosti
  - způsob komunikace s řadičem zařízení
  - zná detaily – např. ví o sektorech a stopách na disku, pohybech diskového raménka, start & stop motoru
- může ovládat všechna zařízení daného druhu nebo třídu příbuzných zařízení
  - např. ovladač SCSI disků – všechny SCSI disky

# Funkce ovladače zařízení

---

1. ovladači předán příkaz od vyšší vrstvy
  - např. **zapiš data do bloku n**
2. nový požadavek **zařazen do fronty**
  - může ještě obsluhovat předchozí
3. ovladač zadá **příkazy řadiči** (požadavek přijde na řadu)
  - např. nastavení hlavy, přečtení sektoru
4. **zablokuje se do vykonání** požadavku
  - neblokuje při rychlých operacích – např. zápis do registru
5. **vzbuzení** obsluhou přerušení (dokončení operace) – zkontroluje, zda nenastala chyba

## Funkce ovladače zařízení – pokrač.

---

6. pokud OK, předá výsledek (**status + data**) vyšší vrstvě
  - status – datová struktura pro hlášení chyb
7. Zpracuje další požadavky ve frontě
  - jeden vybere a spustí

# Poznámky

---

- ovladače často vytvářejí výrobci HW
  - dobře definované rozhraní mezi OS a ovladači
- ovladače podobných zařízení – stejná rozhraní
  - např. síťové karty, zvukové karty, ...

# Problémy s ovladači

---

- Chyba ovladače – pád systému
  - Běh v privilegovaném režimu (jádře)
  - Chyba v ovladači může způsobit pád systému
- Ovladač pro určitý HW i určitý OS
  - Můžete mít starší kameru s ovladačem pro Windows XP, ale třeba nebude použitelná ve Windows 8.1



### 3. SW vrstva OS nezávislá na zařízení

---

- I/O funkce **společné pro všechna zařízení daného druhu**
  - např. společné fce pro všechna bloková zařízení
- definuje rozhraní s ovladači
- poskytuje jednotné rozhraní uživatelskému SW

Tato vrstva často dělá studentům potíže. Přestože má jasně definované funkce – viz další slide

# Poskytované funkce (!!)

---

- **pojmenování** zařízení
  - LPT1 x /dev/lp0
- **ochrana zařízení** ( přístupová práva)
- **alokace a uvolnění** vyhrazených zařízení
  - v 1 chvíli použitelná pouze jedním procesem
  - např. tiskárna, plotter, magnetická páska
- **vyrovnávací paměti**
  - bloková zařízení – bloky pevné délky
  - pomalá zařízení – čtení / zápis s využitím bufferu

## Poskytované funkce – pokračování

---

- hlášení chyb
- jednotná velikost bloku pro bloková zařízení
- v moderních OS se zařízení jeví jako objekty v souborovém systému  
(v mnoha OS je tato vrstva součástí logického souborového systému)



## 4. I/O sw v uživatelském režimu

---

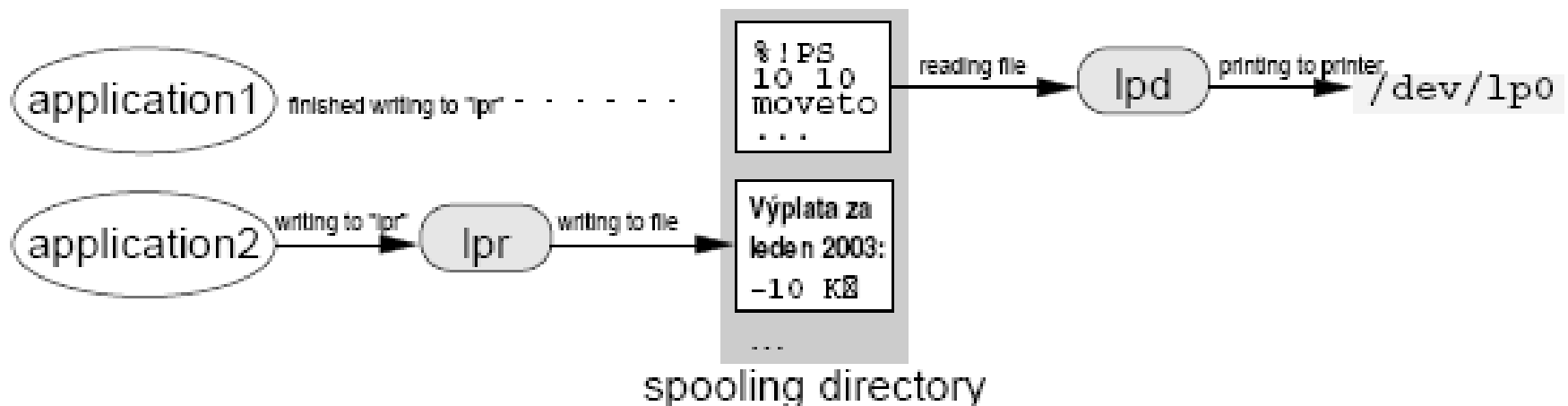
- programátor používá v programech **I/O funkce** nebo **příkazy** jazyka
  - např. printf v C, writeln v Pascalu
  - knihovny sestavené s programem
  - formátování - printf("%.2d:%.2d\n", hodin, minut)
  - často vlastní vyrovnávací paměť na jeden blok
- **spooling**
  - implementován pomocí procesů běžících v uživ. režimu
  - způsob obsluhy vyhrazených I/O zařízení (multiprogram.)
  - např. proces by alokoval zařízení a pak hodinu nic nedělal

# Příklad spoolingu – tisk Unix

---

- přístup k fyzické tiskárně – pouze 1 speciální proces
  - daemon **lpd**
- proces vygeneruje celý soubor, lpd ho vytiskne
  - proces chce tisknout, spustí **lpr** a naváže s ním komunikaci
  - proces předává tisknutá data programu lpr
  - lpr zapíše data do souboru v určeném adresáři
    - spooling directory – přístup jen lpr a lpd
  - dokončení zápisu – lpr oznámí lpd, že soubor je připraven k vytisknutí, lpd soubor vytiskne a zruší

# tisk Unix



**lpd** – démon (služba) čte ze spoolovacího adresáře a přistupuje k tiskárně

**lpr** – data, která chce aplikace vytisknout se zapisují do spoolovacího adresáře

Poznámka - spooling lze použít např. i pro přenos elektronické pošty

---

An orange scroll graphic with a vertical strip on the left and a horizontal strip at the top, framing the text.

Následuje modul OS pro správu souborů.

# Souborové systémy

---

- potřeba aplikací **trvale** uchovávat data
- **hlavní požadavky**
  - možnost uložit velké množství dat
  - informace zachována i po ukončení procesu
  - data přístupná více procesům
- **společné problémy** při přístupu k zařízení
  - alokace prostoru na disku
  - pojmenování dat
  - ochrana dat před neoprávněným přístupem
  - zotavení po havárii (výpadek napájení)

# Soubor

---

OS pro přístup k médiím poskytuje abstrakci od fyzických vlastností média – soubor

Definice:

soubor = pojmenovaná množina souvisejících informací, uložená na datovém médiu, se kterou lze pracovat nástroji operačního systému jako s jedním celkem

# Souborový systém

---

Definice:

Způsob organizace ukládání souborů (a adresářů), tak aby k nim bylo možné snadno přistupovat.

Souborové systémy – uloženy na vhodném typu elektronické paměti, která je buď v počítači (část disku, CD), nebo zpřístupněna přes počítačovou síť (Samba, NFS).

Souborový systém říká, **jak** jsou soubory na disk ukládány

# Souborový systém

---

- souborový systém (file system, fs)
  - konvence pro ukládání souborů a snadný přístup k nim
    - datové struktury a algoritmy
- část OS, poskytuje mechanismus pro ukládání a přístup k datům, implementuje danou konvenci



# Souborové systémy(fs)

---

- Současné OS – implementují více fs
  - kompatibilita (starší verze, ostatní OS)
- Windows (XP, Vista, 7, 8, 8.1, 10):
  - základní je NTFS
  - ostatní: FAT12, FAT16, FAT32, exFAT, ISO 9660 (CD-ROM)
- Linux
  - ext2, ext3, ext4, ReiserFS, JFS, XFS
  - ostatní: FAT12 až 32, ISO 9660, Minix, VxFS, OS/2 HPFS, SysV fs, UFS, NTFS

# Základní znalosti

---

V PC můžeme mít **více pevných disků**, např. dva:

Linux: **/dev/sda, /dev/sdb**

Každý disk se může dělit na **několik oddílů**:

**/dev/sda1, /dev/sda2, /dev/sda3** (1. disk)

**/dev/sdb1, /dev/sdb2** (2. disk)


Každý oddíl (partition) – nějaký **filesystem**:

**/dev/sda1 ext4**

**/dev/sda1 swap**

**/dev/sda1 ntfs**

**/dev/sdb1 fat32**

An orange scroll-shaped box with a dark orange border and a small orange tab on the right side. It contains text about disk formatting.

Formátování disku:  
**/sbin/mkfs.ext4 /dev/sda1**

# Základní znalosti

---

**fdisk** /dev/sda

- Zobrazení rozdělení disku na oddíly (partitions)
- Možnost toto rozdělení změnit

/sbin/mkfs.ext4 /dev/sda1

- Zformátování oddílu na vybraný filesystem (zde ext4)

# Počet oddílů

---

## 2 způsoby dělení

- **Master Partition Table (MPT)**
  - Master Boot Record (MBR) – na počátku disku
  - Umožňuje **4 oddíly primární**
  - Chceme-li více, **3 primární a 1 extended**, který lze dělit na další oddíly
- **GUID Partition Table (GPT)**
  - Nelimituje na 4 oddíly, např. Microsoft 124 oddílů
  - Používá např. Mac OS

# Struktura MBR

Struktura MBR					
Adresa			Popis	Délka v bajtech	
Hex	Oct	Dec			
0000	0000	0	Kód zavaděče	440 (max 446)	
01B8	0670	440	Volitelná signatura disku	4	
01BC	0674	444	Obvykle nuly; 0x0000	2	
01BE	0676	446	<b>Tabulka rozdělení disku (MPT)</b> (4 položky po 16 bajtech, IBM schéma oddílů)	64	
01FE	0776	510	55h	Signatura MBR; 0xAA55 <sup>[1]</sup>	2
01FF	0777	511	AAh		
Celková délka MBR: 446 + 64 + 2 =				512	

Zdroj obr.: wikipedia

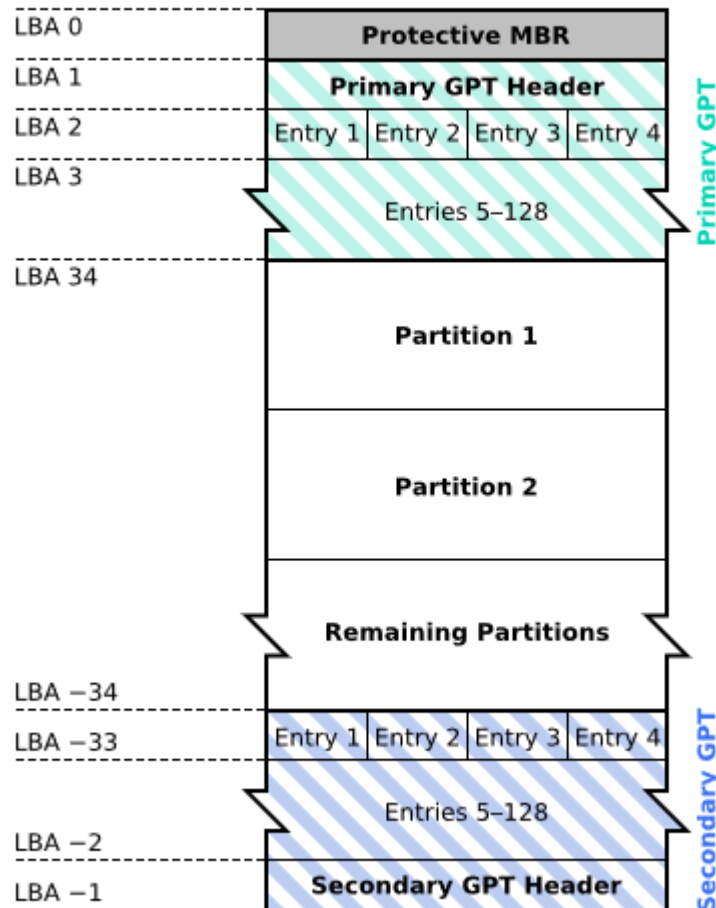
Na začátku disku je MBR record, který obsahuje:

**Kód zavaděče**  
(pokud se z disku bootuje, tak tento kód je puštěn z BIOSu)

**Tabulka rozdělení disku**  
(4 partitions)

# GUID Partition Table (GPT)

## GUID Partition Table Scheme



Nejsme omezeni na 4 primární oblasti

Součástí standardu **UEFI**, který by měl nahradit klasický BIOS (umožňuje např. secure boot)

Zdroj obr.: wikipedia

# GPT

---

- záložní kopii tabulky ukládá na konci disku
- Velikost GPT je na disku s 512B sektory  $34 \times 512B = 16KB$
- LBA – logický blok, velikost 512B
- První oddíl začíná na LBA 34

# Historický vývoj

---

- první systémy (děrné štítky, děrné pásky)
  - vstup – děrné štítky, výstup – tiskárna
  - soubor = množina děrných štítků
- magnetické pásky
  - vstup i výstup – pásky
  - soubor = pojmenovaná množina záznamů na magnetické pásce
  - záznam = strukturovaný datový objekt tvořený konečným počtem pojmenovaných položek
- Magnetické a optické disky
- SSD disky (bez pohyblivých částí)



# Soubor - definice

---

Soubor je **pojmenovaná množina informací** (dat), které mají určité společné vlastnosti.

Se souborem lze pracovat nástroji OS jako s jedním celkem.

## Soubor

- Jeden druh dat (textový, obrázek, zvuk, program)
- Složený (archiv .zip, ISO obraz disku atp.)

# Uživatelské rozhraní fs (file systémů)

---

- vlastnosti fs z pohledu uživatele
  - konvence pro pojmenování souborů
  - vnitřní struktura souboru
  - typy souborů
  - způsob přístupu
  - atributy a přístupová práva
  - služby OS pro práci se soubory

# Konvence pro pojmenování souborů



---

- vytvoření souboru – proces určuje jméno souboru
- různá pravidla pro vytváření jmen – různé OS
- Windows NT, XP x Unix a Linux
- rozlišuje systém malá a velká písmena?
  - Win32API nerozlišuje: **ahoj**, **Ahoj**, **AHOJ** stejná
  - UNIX rozlišuje: **ahoj**, **Ahoj**, **AHOJ** rozdílná jména

# Ukázka

---

Snaha vytvořit textový soubor ahoj.txt a Ahoj.txt ve stejném adresáři pod Windows:

Tento počítač > Nový svazek (D:) > work > a				
Název	^	Datum změny	Typ	Velikost
 Ahoj (2).txt		05.12.2016 9:24	Textový dokument	0 kB
 ahoj.txt		05.12.2016 9:24	Textový dokument	0 kB

# Pojmenování souborů

---

- jaká může být délka názvu souboru?
  - WinNT 256 znaků NTFS
  - UNIX obvykle alespoň 256 znaků (dle typu fs)
- množina znaků?
  - všechny běžné – názvy písmena a číslice
  - WinNT – znaková sada UNICODE
    - $\beta\epsilon\tau\alpha$  – legální jméno souboru
  - Linux – všechny 8bitové znaky kromě / a char(0)

# Pojmenování souborů

---

- přípony?

- MS DOS – jméno souboru 8 znaků + 3 znaky přípona
- Dnešní OS: Windows 10, Linux – i více přípon

- další omezení?

- Některé OS– mezera nesmí být první a poslední znak
- Ale Windows 10:  
`mkdir "sMezerou"`  
`cd "sMezerou"`

# Typy souborů

---

OS podporují více typů souborů:

- **obyčejné soubory** – převážně dále rozebírány
  - data zapsaná aplikacemi
  - obvykle rozlišení textové x binární
  - **textové** - řádky textu ukončené znaky CR (MAC), LF (UNIX), nebo CR+LF (MS DOS, Windows)
  - binární – všechny ostatní
  - OS rozumí struktuře spustitelných souborů

# Typy souborů

---

- adresáře

- systémové soubory, udržují strukturu souborového systému

- speciální - Linux , UNIX:

- znakové speciální soubory (/dev/tty)
- blokové speciální soubory
  - rozhraní pro I/O zařízení, /dev/lp0 – tiskárna
- pojmenované roury (příkaz mkfifo roura, nebo mknod)
- symbolické odkazy



# Vnitřní struktura (obyčejného) souboru

---

- Vnitřní struktura souboru
  - nestrukturovaná posloupnost bytů
  - posloupnost záznamů
  - strom záznamů
- **nestrukturovaná posloupnost bytů** (nejčastěji)
  - OS obsah souboru nezajímá, interpretace je na aplikacích
  - maximální flexibilita
  - programy mohou strukturovat, jak chtějí

# Vnitřní struktura souboru – posloupnost záznamů

---

## ■ posloupnost záznamů pevné délky

- každý záznam má vnitřní strukturu
- operace čtení –vrátí záznam, zápis – změnění / přidá záznam
- v historických systémech
- záznamy 80 znaků obsahovaly obraz děrných štítků
- v současných systémech se téměř nepoužívá

# Vnitřní struktura souboru – strom záznamů

---

## ■ strom záznamů

- záznamy nemusejí mít stejnou délku
- záznam obsahuje pole **klíč** (na pevné pozici v záznamu)
- záznamy **seřazený** podle klíče, aby bylo možné vyhledat záznam s požadovaným klíčem
- mainframy pro komerční zpracování dat

# Způsob přístupu k souboru (!!)

---

## Sekvenční nebo přímý

### ▪ **sekvenční** přístup

- procesy mohou číst data pouze v pořadí, v jakém jsou uloženy v souboru
- tj. od prvního záznamu (bytu), nemohou přeskakovat
- možnost „přetočit pásku“ a číst opět od začátku, **rewind()**
- v prvních OS, kde data na magnetických páskách

# Způsob přístupu k souboru (!!)

---

- **přímý** přístup (random access file)
  - čtení v libovolném pořadí nebo podle klíče
  - přímý přístup je nutný např. pro databáze
  - uživatel – např. přeskokování děje filmu
  - určení začátku čtení
    - každá operace určuje pozici
    - OS udržuje pozici čtení / zápisu, novou pozici lze nastavit speciální operací „seek“

# Způsob přístupu k souboru

---

- všechny běžné současné OS – soubory s přímým přístupem
- v některých OS pro mainframy:
  - při vytvoření souboru se určilo, zda je sekvenční nebo s přímým přístupem
  - OS mohl používat rozdílné strategie uložení souboru
- Speciální RT systémy – někdy mohou využít sekvenční soubory

# Atributy

---

- informace sdružená se souborem
- některé atributy interpretuje OS, jiné systémové programy a aplikace
- významně se liší mezi jednotlivými OS
- ochrana souboru
  - kdo je vlastníkem, množina přístupových práv, heslo, ...

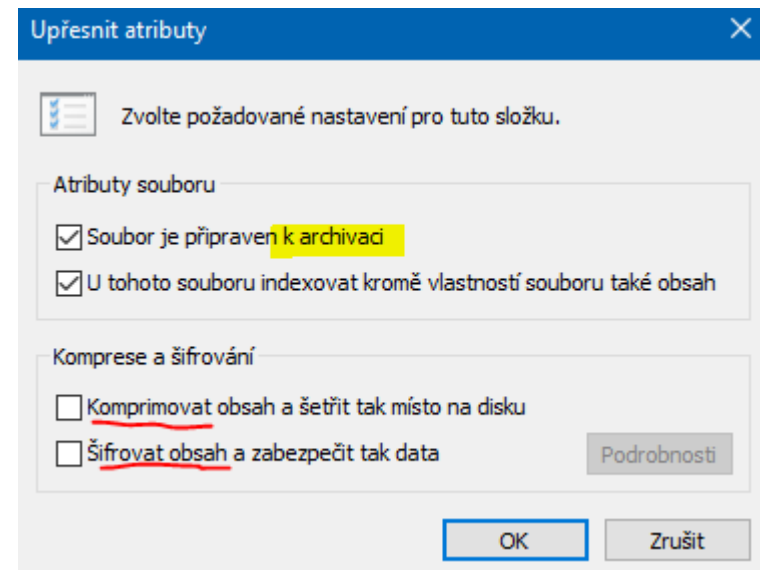
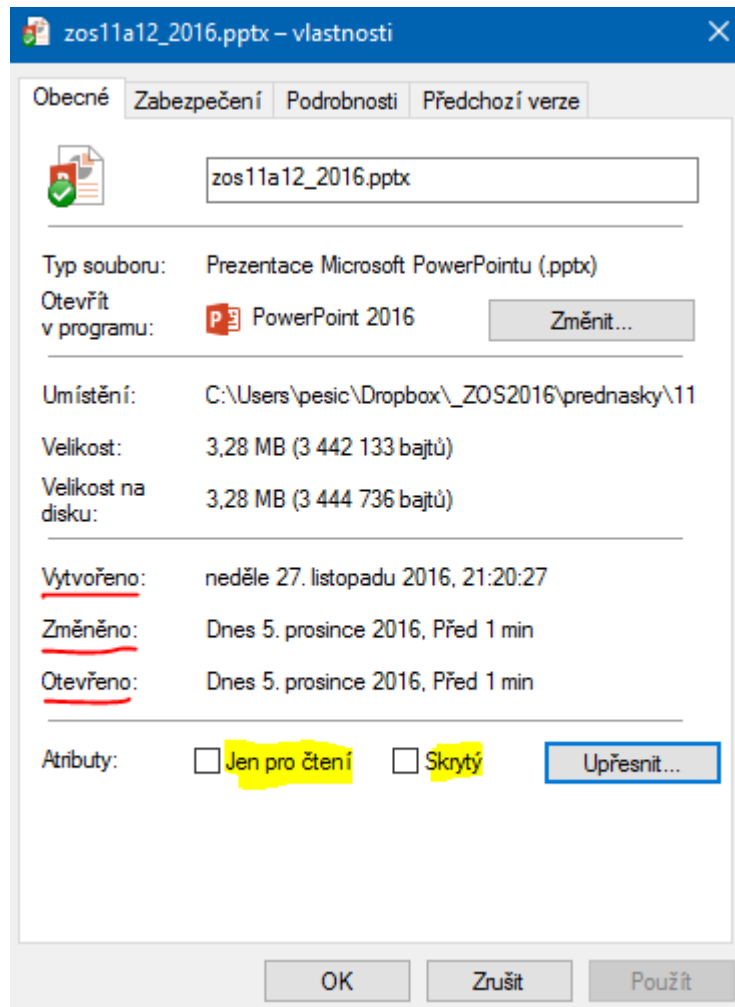
# Atributy - pokračování

---

- příznaky
  - určují vlastnosti souboru
  - **hidden** – neobjeví se při výpisu
  - **archive** – soubor nebyl zálohován
  - **temporary** – soubor bude automaticky zrušen
  - **read-only**,
  - text/binary, random access, ...
- přístup k záznamu pomocí klíče
  - délka záznamu, pozice a délka klíče
- velikost, datum vytvoření, poslední modifikace, poslední přístup



# Atributy – ukázka Windows 10



# Služby OS pro práci se soubory

---

- většina současných – základní model dle UNIXu
- základní filozofie UNIXu – méně je někdy více

# Základní pravidla

---

- **veškerý I/O prováděn pouze pomocí souborů**
  - obyčejné soubory – data, spustitelné programy
  - zařízení – disky, tiskárny
  - se všemi typy - zacházení pomocí **stejných** služeb systému
- obyčejný soubor – uspořádaná posloupnost bytů
  - význam znají pouze programy, které s ním pracují
  - interní struktura souboru OS nezajímá
- seznam souborů – **adresář**
  - adresář je také soubor
  - soubory a adresáře koncepčně umístěny v adresáři

# Základní pravidla - dokončení

---

- speciální soubory – pro přístup k zařízením
  - DOS – PRN:, COM1:
  - Linux - /dev/sda, /dev/tty, ...

# Jednotný přístup nebyl vždy

---

- Před příchodem UNIXu jednotný přístup nebyl samozřejmostí
- Téměř všechny moderní systémy základní rysy UNIX modelu převzaly
- většina systémů před UNIXem
  - samostatné služby pro čtení / zápis terminálu, na tiskárnu do souboru

systémy poskytovaly „více služeb“ - komplexní

x

model podle UNIXu – podstatně menší složitost

# Základní služby pro práci se soubory

---

## ■ otevření souboru

- než s ním začneme pracovat
- úspěšné – služba pro otevření souboru vrátí **popisovač souboru (file descriptor)** – malé celé číslo
- popisovač souboru používáme v dalších službách
  - čtení apod.

# Základní služby pro práci se soubory

---

## otevření souboru:

*fd = open (jmeno, způsob)*

- jméno – řetězec pojmenovávající soubor
- způsob – pouze pro čtení, zápis, obojí
- fd – vrácený popisovač souboru

otevření souboru nalezne informace o souboru na disku a vytvoří pro soubor potřebné datové struktury

popisovač souboru – **index to tabulky souborů** uvnitř OS

# Základní služby pro práci se soubory

---

## vytvoření souboru:

`fd=creat(jméno, práva)`

- vytvoří nový soubor s daným jménem a otevře pro zápis
- pokud soubor existoval – zkrátí na nulovou délku
- fd – vrácený popisovač souboru

Opravdu je creat nikoliv create



# Základní služby pro práci se soubory

---

operace **čtení** ze souboru:

**read(fd, buffer, počet\_bytů)**

- přečte *počet\_bytů* ze souboru *fd* do *bufferu*
- může přečíst méně – zbývá v souboru méně
- přečte 0 bytů – konec souboru

# Základní služby pro práci se soubory

---

## operace **zápisu** do souboru

### **write (fd, buffer, počet\_bytů)**

- Zapiše do souboru z bufferu daný počet bytů
- Zápis uprostřed souboru – přepíše, na konec – prodlouží

### **read() a write()**

- vrací počet skutečně zpracovaných bytů
- jediné operace pro čtení a zápis
- samy o sobě poskytují sekvenční přístup k souboru

# Základní služby pro práci se soubory

---

## **nastavení pozice v souboru:**

**lseek (fd, offset, odkud)**

nastaví offset příští čtené/zapisované slabiky souboru

odkud

- od začátku souboru
- od konce souboru (záporný offset)
- od aktuální pozice

poskytuje **přímý přístup** k souboru („přeskakování děje filmu“ )

# Základní služby pro práci se soubory

---

## **zavření souboru**

**close (fd)**

uvolní datové struktury alokované OS pro soubor

# Památovat

---

služba	popis
<b>creat</b>	Vytvoří soubor
<b>open</b>	Otevře soubor
<b>read</b>	Čtení
<b>write</b>	Zápis
<b>lseek</b>	Přímý přístup k souboru změna pozice pro čtení, zápis
<b>close</b>	Uzavření souboru

# Příklad použití rozhraní – kopírování souboru

---

```
int src, dst, in;

src = open("puvodni", O_RDONLY);           /* otevreni zdrojoveho */
dst = creat("novy", MODE);                 /* vytvoreni ciloveho */
while (1)
{
    in = read(src, buffer, sizeof(buffer)); /* cteme */
    if (in == 0)                           /* konec souboru? */
    {
        close(src);                        /* zavreme soubory */
        close(dst);
        return;                           /* ukončení */
    }
    write(dst, buffer, in);                /* zapiseme prectena data */
}
```

## Další služby pro práci se soubory

---

- změna přístupových práv, zamykání, ...
- závislé na konkrétních mechanismech ochrany
- např. UNIX
  - zamykání `fcntl (fd, cmd)`
  - zjištění informací o souboru (typ, příst. práva, velikost)
  - `stat (file_name, buf)`, `fstat (fd, buf)`
  - *man stat*, *man fstat*

## Paměťově mapované soubory (!!)

---

- někdy se může zdát `open/read/write/close` nepohodlné
- možnost mapování souboru do adresního prostoru procesu
- služby systému `mmap()`, `munmap()`
- mapovat je možné i jen část souboru
- k souboru pak přistupujeme např. přes pointery v C



## Paměťově mapované soubory - příklad

---

- délka stránky 4KB
- soubor délky 64KB
- chceme mapovat do adresního prostoru od 512KB
- $512 * 1024 = 524\,288$  .. od této adresy mapujeme
- 0 až 4KB souboru bude mapováno na 512KB – 516KB
- čtení z 524 288 čte byte 0 souboru atd.

# Implementace paměťově mapovaných souborů

---

- OS použije soubor jako **odkládací prostor** (swapping area) pro určenou část virtuálního adresního prostoru
- čtení / zápis na adr. 524 288 způsobí **výpadek stránky**
- do rámce se **načte** obsah první stránky souboru
- pokud je modifikovaná stránka vyhozena (nedostatek volných rámců), **zapíše** se do souboru
- po skončení práce se souborem se **zapíše** všechny modifikované stránky

# Problémy paměťově mapovaných souborů

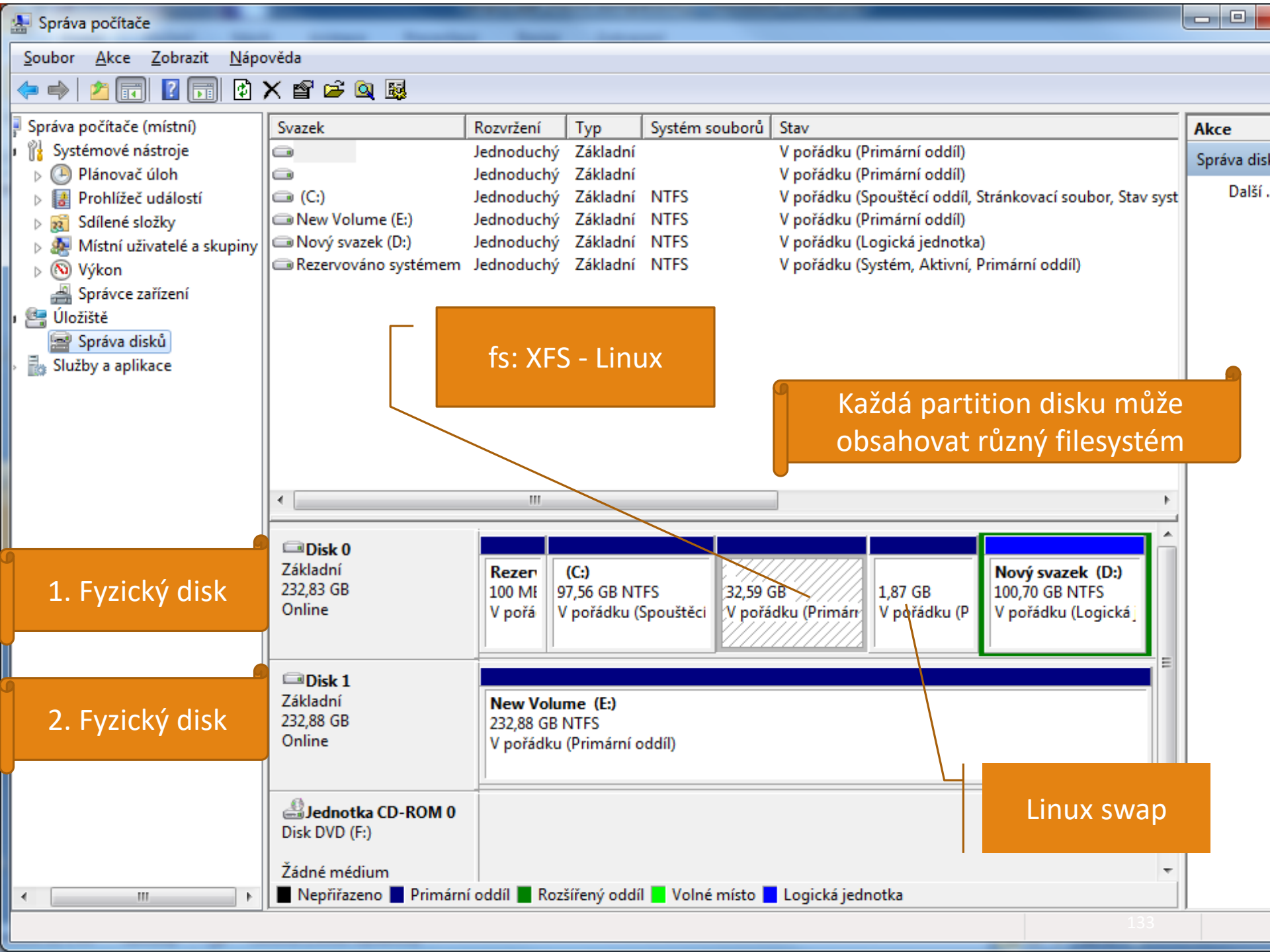
---

- není známa přesná velikost souboru, nejmenší jednotka je stránka
- problém nekonzistence pohledů na soubor, pokud je zároveň mapován a zároveň se k němu přistupuje klasickým způsobem

# Adresářová struktura

---

- Jeden oddíl (partition) disku obsahuje jeden filesystem (fs)
- filesystem – 2 součásti:
  - množina souborů, obsahujících **data**
  - **adresářová struktura** – udržuje informace o všech souborech v daném fs
- adresář **překládá** jméno souboru na informace o souboru (umístění, velikost, typ ...)



- Správa počítače (místní)
- Systémové nástroje
  - Plánovač úloh
  - Prohlížeč událostí
  - Sdílené složky
  - Místní uživatelé a skupiny
  - Výkon
  - Správce zařízení
- Úložiště
  - Správa disků
  - Služby a aplikace

fs: XFS - Linux

Každá partition disku může obsahovat různý filesystém

1. Fyzický disk

2. Fyzický disk

Linux swap

# Základní požadavky na adresář (příkazy)

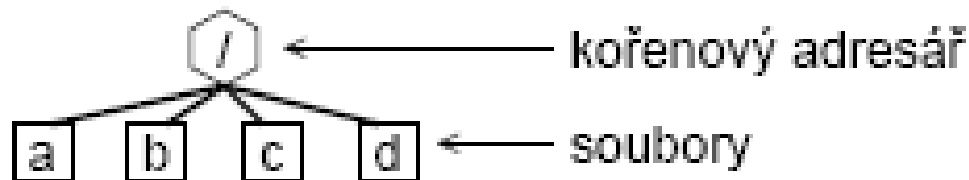
---

- procházení souborovým systémem (**cd**)
- výpis adresáře (**ls**)
- vytvoření a zrušení souboru (**rm, rmdir**)
- přejmenování souboru (**mv**)
  
- dále schémata logické struktury adresářů
  - odpovídá historickému vývoji OS

# Schémata logické struktury adresářů

---

- **jednoúrovňový adresář**
- původní verze MS DOSu
- všechny soubory jsou v jediném adresáři
- všechny soubory musejí mít jedinečná jména
- problém zejména pokud více uživatelů



nelze mít dva soubory  
příklad.c

# Dvouúrovňový adresář

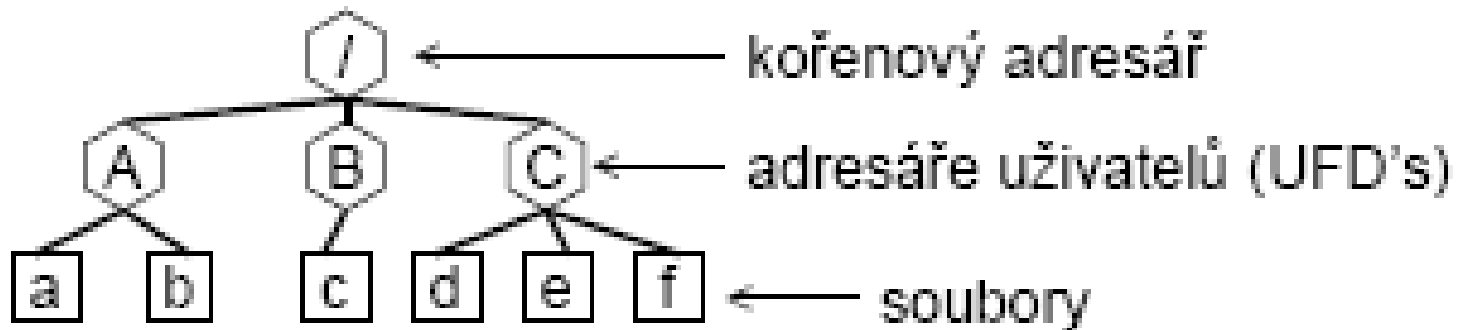
---

- adresář pro **každého uživatele** (User File Directory, UFD)
- OS prohledává pouze UFD , nebo pokud specifikováno adresář jiného uživatele **[user] file**
- výhoda – 2 uživatelé mohou mít soubor se stejným názvem
- speciální adresář pro systém
  - příkaz se hledá v adresáři uživatele
  - pokud zde není, vyhledá se v systémovém adresáři



# Dvouúrovňový adresář – pokr.

---



každý uživatel může být nanejvýš jeden soubor  
nazvaný priklad.c

# Adresářový strom

---

- zobecnění předchozího
- např. MS DOS, Windows NT
- adresář – množina souborů a adresářů
- souborový systém začíná kořenovým adresářem „/“
- MS DOS „\“, znak / se používal pro volby
- cesta k souboru – jméno uvedené ve volání open, creat
  - absolutní
  - relativní

# Cesta k souboru

---

absolutní cesta začíná:  
/ (Linux)  
C:\ (windows)

## ■ absolutní

- kořenový adresář a adresáře, kudy je třeba projít, název souboru
- oddělovače adresářů – znak „/“
- např. */home/user/data/v1/data12.txt*

## ■ relativní

- aplikace většinou přistupují k souborům v jednom adresáři
- defaultní prefix = pracovní adresář
- cesta nezačíná znakem /
- př. *data12.txt* , *data/v1/data12.txt*

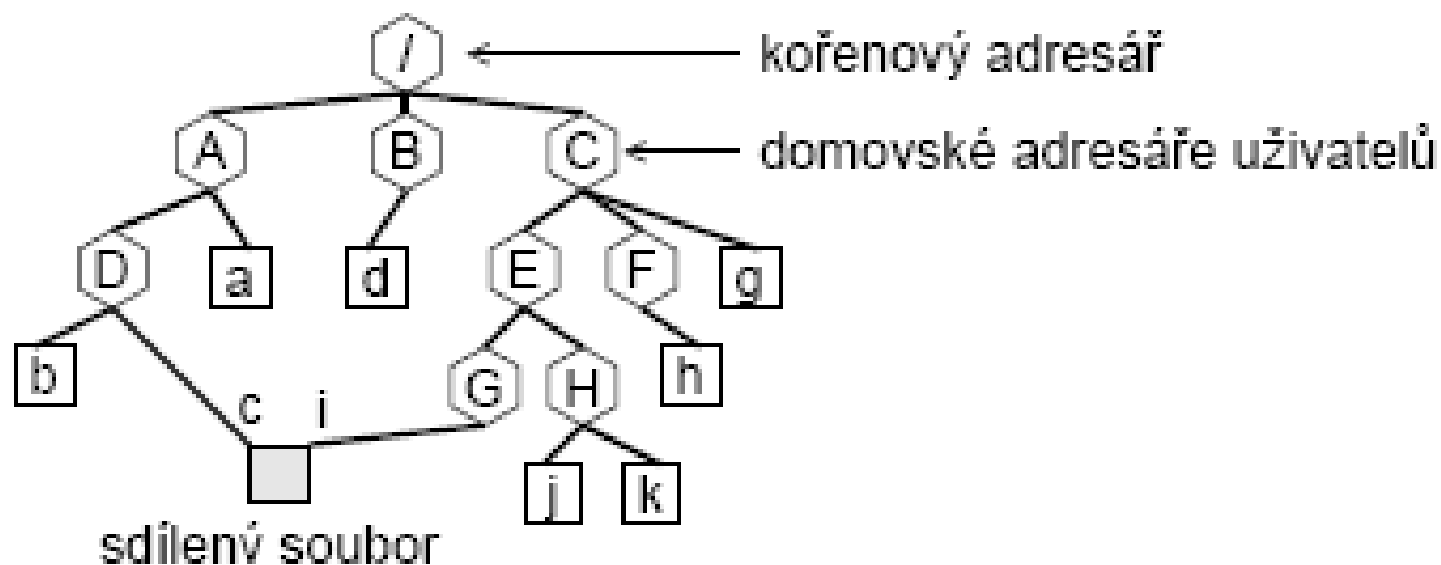
# Acyklický graf adresářů

---

- **linky** na stejný soubor z více míst
- použití: týmová spolupráce na určitém projektu
- **sdílení společného souboru** nebo podadresáře
  - stejný soubor (adr.) může být viděn ve dvou **různých** adresářích
- flexibilnější než strom, komplikovanější
- **rušení souborů / adresářů** – kdy můžeme zrušit?
  - se souborem sdružen počet odkazů na soubor z adresářů
  - každé zrušení sníží o 1, když 0 = není odkazován
- jak **zajistit aby byl graf acyklický**?
  - algoritmy pro zjištění cyklu, drahé pro fs



# Acyklický graf adresářů



stejný soubor viděný v různých cestách

# Obeční graf adresářů

---

- obtížné zajistit, aby graf byl acyklický
- prohledávání grafu
  - omezení počtu prošlých adresářů (Linux)
- rušení souboru
  - pokud cyklus, může být počet odkazů  $> 0$  i když je soubor již není přístupný
  - garbage collection – projít celý fs, označit všechny přístupné soubory; zrušit nepřístupné; (drahé, zřídka používáno)

# Nejčastější použití

---

- adresářový strom (MS DOS)
- UNIX od původních verzí acyklický graf  
**hard links** – sdílení pouze souborů – nemohou vzniknout cykly
- POZOR!  
Je nutné si uvědomit rozdíl mezi pojmy adresářový strom a acyklický graf.

# Základní služby pro práci s adresáři (!)

---

- téměř všechny systémy dle UNIXu

- **pracovní adresář** – služby:

- **chdir (adresář)**

  - nastavení pracovního adresáře

- **getcwd (buffer, počet\_znaků)**

  - zjištění pracovního adresáře



# Práce s adresářovou strukturou (!)

---

## vytváření a rušení adresářů

- mkdir (adresář, přístupová\_práva)
- rmdir (adresář) – musí být prázdný

## zrušení souboru

- remove (jméno\_souboru)

## přejmenování souboru

- rename (jméno\_souboru, nové\_jméno)
- provádí také přesun mezi adresáři

# Práce s adresářovou strukturou

---

## čtení adresářů – UNIX / POSIX

- DIRp = **opendir** (adresář)
  - otevře adresář
- položka = **readdir** (DIRp)
  - čte jednotlivé položky adresáře
- **closedir** (DIRp)
  - zavře adresář
- **stat** (jméno\_souboru, statbuf)
  - info o souboru, viz man 2 stat

př. DOS: findfirst / findnext

ke všem uvedeným  
voláním získáte v  
Linuxu  
podrobnosti pomocí:

**man 2 opendir**  
**man 2 readdir**  
**man 2 stat**

# Pevný disk: cluster, sektor

---

## Cluster

- nejmenší alokovatelná jednotka pro uložení souboru
- Skládá se z 1 nebo více sektorů

## Sektor

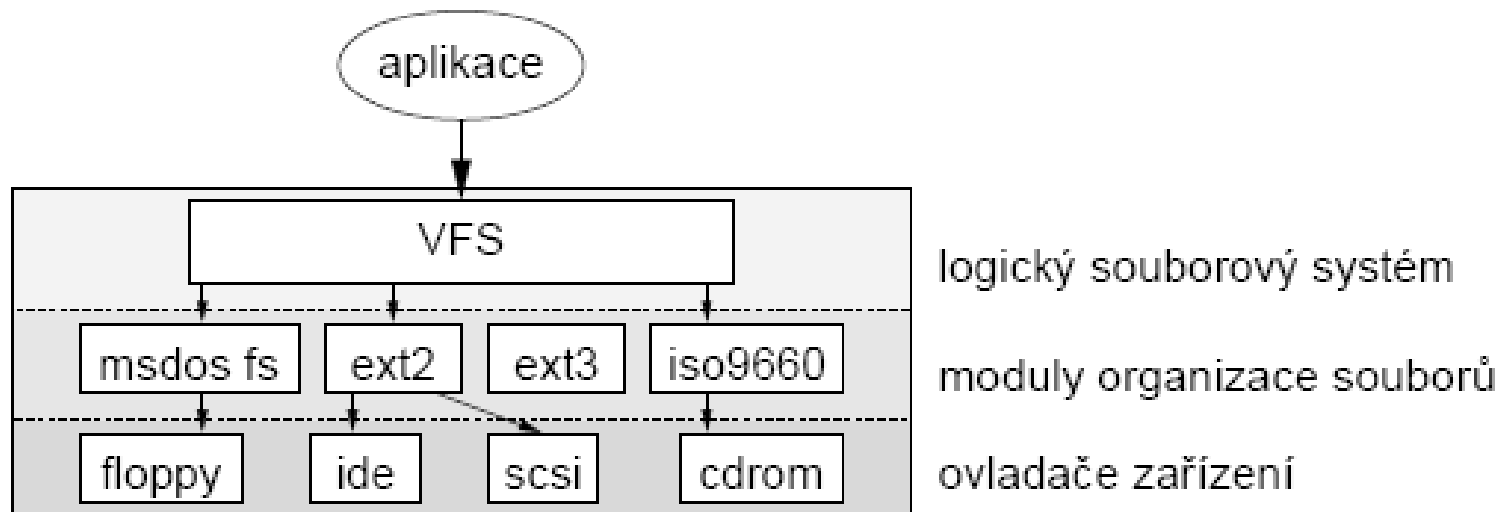
- 512B, 1KB, 4KB, ...

## Příklady

- Disk 512B cluster, 512B sektor
- Disk se 4KB clusterem – 8 sektorů

# Implementace souborových systémů (!!!)

---



# Implementace fs - vrstvy

---

1. Logický (virtuální) souborový systém
  - Volán aplikacemi
2. Modul organizace souborů
  - Konkrétní souborový systém (např. ext3)
3. Ovladače zařízení
  - Pracuje s daným zařízením
  - Přečte/zapíše logický blok

# Ad 1 – virtuální fs

---

- Volán aplikacemi
- Rozhraní s moduly organizace souborů
- Obsahuje kód **společný** pro všechny typy fs
- **Převádí** jméno souboru na informaci o souboru
- **Udržuje informaci** o otevřeném souboru
  - Pro čtení / zápis (režim)
  - Pozice v souboru
- **Ochrana a bezpečnost** (ověřování přístupových práv)

# Ad 2 – modul organizace souborů

---

- Implementuje **konkrétní** souborový systém
  - ext3, xfs, ntfs, fat, ..
- **Čte/zapisuje datové bloky** souboru
  - Bloky souboru číslovány 0 až N-1 (nebo  $a_1, a_2, \dots, a_N$ )
  - Převod čísla bloku souboru na diskovou adresu
  - Volání ovladače pro čtení či zápis bloku
- **Správa volného** prostoru + **alokace** volných bloků souborům
- Údržba datových struktur filesystemu

# Ad 3 – ovladače zařízení

---

- Nejnižší úroveň
- Zařízení: SATA disk, SCSI disk, DVD mechanika
- Interpretují požadavky:  
přečti logický blok 6456 ze zařízení 3

Pozn. číslo zařízení – Linux (hlavní a vedlejší)



## Jak VFS pracuje s konkrétním filesystemem (!)

- Nový filesystem, který chceme používat se nejprve v systému **zaregistruje**
- Díky registraci VFS ví, jak zavolat jeho metody open, read, write pro konkrétní soubor
- Při požadavku na soubor VFS napřed zjistí, na kterém filesystemu leží:
  - Viz např. příkaz mount
  - /bin/ls může ležet na ext4, /home/pesi/f1.txt na xfs
  - Pro čtení /bin/ls zavolá VFS->ext4->read
  - Pro čtení /home/pesi/f1.txt zavolá VFS->xfs->read

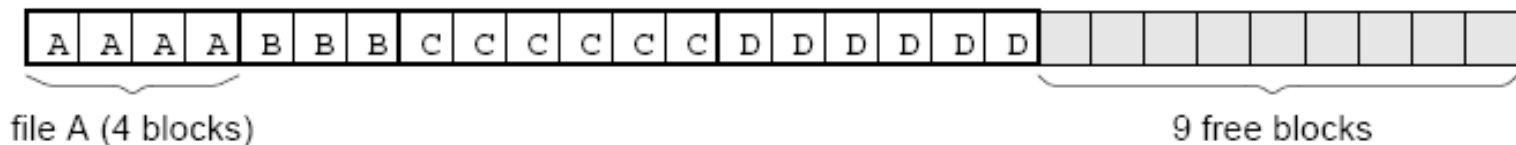
# Implementace souborového systému

---

- Nejdůležitější:
  - které diskové bloky patří každému ze souborů 😊
  - které diskové bloky jsou volné
- Předpokládáme:
  - fs je umístěn v nějaké diskové partition
  - bloky v diskové oblasti jsou očíslovány od 0

# Kontinuální alokace

- Soubor jako **kontinuální posloupnost** diskových bloků
- Příklad: bloky velikosti 1KB, soubor A (3.5KB) by zabíral 4 po sobě následující bloky
- Implementace
  - Potřebujeme znát číslo prvního bloku
  - Znat celkový počet bloků souboru (např. v adresáři)
- **Velmi rychlé čtení**
  - Hlavičku disku na začátek souboru, čtené bloky jsou za sebou



Lépe je bloky označovat: a1, a2, a3, a4

# Kontinuální alokace

---

- Problém – dynamičnost OS
  - soubory vznikají, zanikají, mění velikost
- Nejprve zapisovat sériově do volného místa na konci
- Po zaplnění využít volné místo po zrušených souborech
- Pro výběr vhodné díry – potřebujeme znát konečnou délku souboru – většinou nevíme..

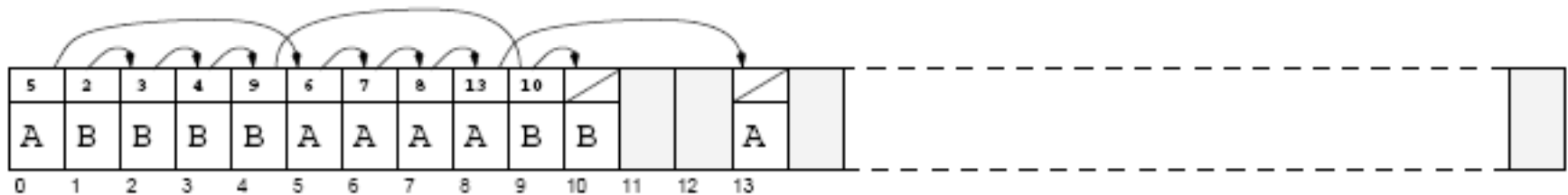
# Lze dnes využít kontinuální alokaci?

---

- Dnes se používá pouze na read-only a write-once médiích
- Např. v ISO 9660 pro CD ROM
- Výhodou DVD je, že dopředu víme velikosti souborů, které vypalujeme a tyto soubory se také typicky dále již nemění

# Seznam diskových bloků

- Svázat diskové bloky do seznamu – nebude vnější fragmentace
- Na začátku diskového bloku je uložen odkaz na další blok souboru, zbytek bloku obsahuje data souboru
- Pro přístup k souboru stačí znát pouze číslo prvního bloku souboru (může být součástí záznamu v adresáři) a velikost (kolik využito z posledního bloku)



# Seznam diskových bloků

---

- Sekvenční čtení – bez potíží
- Přímý přístup – simulován sekvenčním, pomalé (musí dojít ke správnému bloku)
- Velikost dat v bloku není mocnina dvou
  - Část bloku je zabraná odkazem na další blok
  - Někdy může být nevýhodou (kešování, optimalizace, ...)

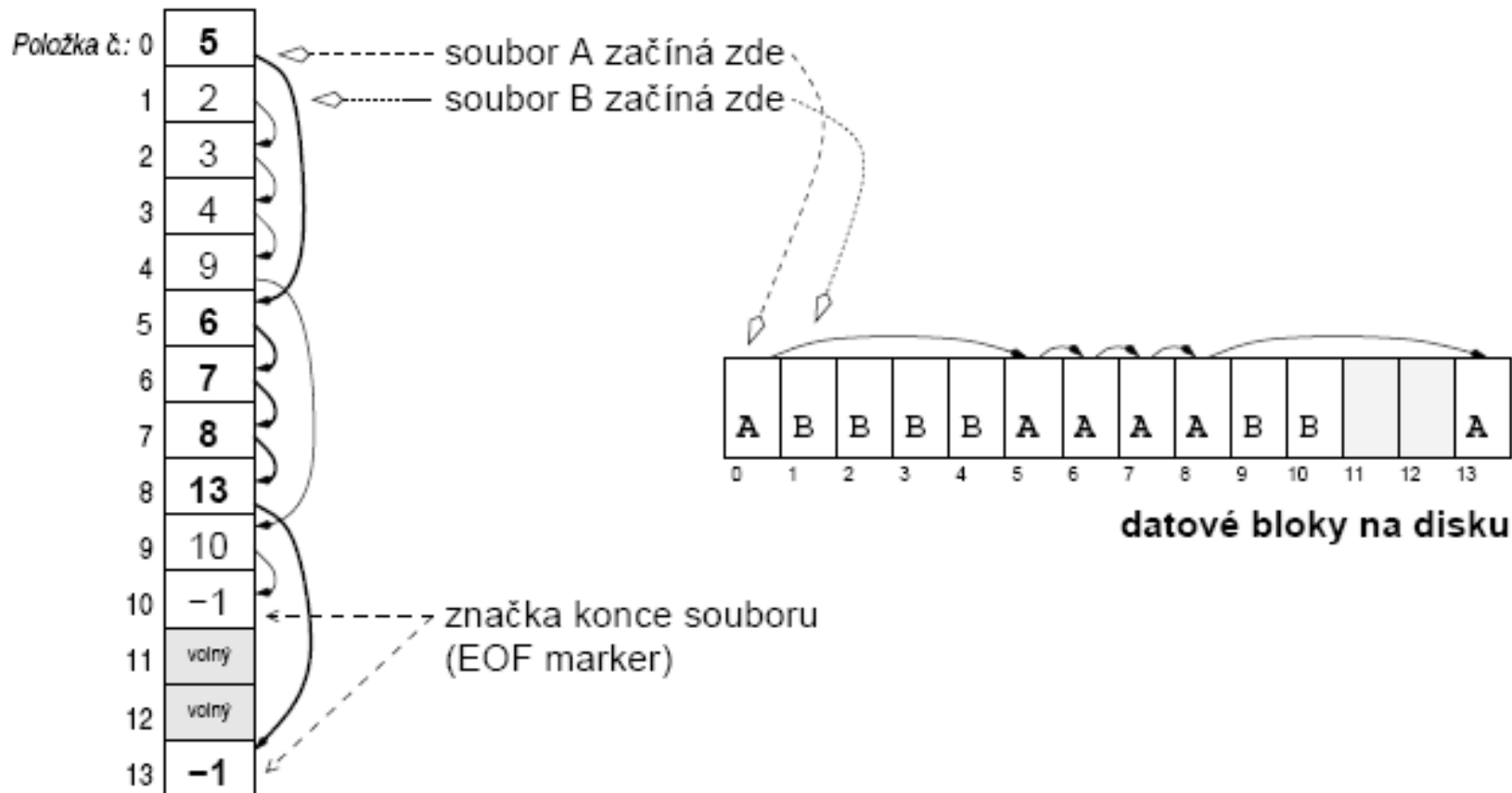
# FAT (!!)

---

- Přesunutí odkazů do samostatné tabulky FAT
- **FAT (File Allocation Table)**
  - Každému diskovému bloku **odpovídá** jedna položka ve FAT tabulce
  - Položka FAT obsahuje číslo **dalšího bloku** souboru (je zároveň odkazem **na další položku** FAT!)
  - Řetězec odkazů je ukončen speciální značkou, která není platným číslem bloku (poslední blok souboru)
  - Volný blok – značí, že je odpovídající datový blok volný
  - Bad block – značí, že je odpovídající datový blok poškozený



## Tabulka FAT



Položce číslo X ve FAT odpovídá datový blok X na disku

Položka ve FAT obsahuje odkaz na další datový blok na disku a tedy i na další položku ve FAT tabulce

## Tabulka FAT

Položka č.: 0	5	
1	2	
2	3	
3	4	
4	9	
5	6	
6	7	
7	8	
8	13	
9	10	
10	-1	
11	volný	
12	volný	
13	-1	

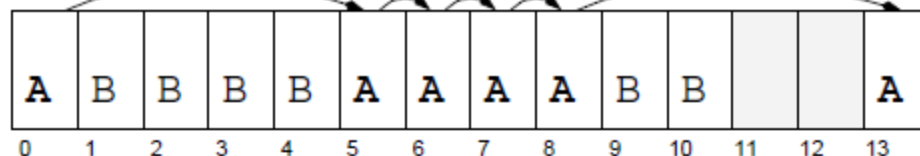
soubor A začíná zde  
soubor B začíná zde

Fakt tu je !  
Jdem dál na  
6

značka konce souboru  
(EOF marker)

5 znamená, že na indexu 5 je  
další odkaz na blok souboru A

Na indexu 5 je i datový blok  
souboru A (kus filmu)



datové bloky na disku

Co je na FAT důležité?

Pokud bych chtěl např. k souboru B přidat další datový blok,  
nemusím s ničím hýbat, pouze do FAT(10) vložím číslo 11, a do  
FAT(11) dám -1 a soubor B je prodloužený

# FAT

---

FAT je ukázka implementace souborového systému, kde v druhé části máme **datové bloky** (obsahující např. části jednoho filmu) a v první části máme **indexy**, které nám říkají, pod jakým číslem se nalézá další odkaz

Výhodou je, že s určitým souborem můžeme manipulovat, zrušit ho, prodloužit, atd., aniž bychom ovlivnili pozici ostatních souborů na disku

# FAT - defragmentace

---

- Úplná – obsazené bloky souborů půjdou na disku za sebou, poté následuje volné místo
- Částečná – upraví se jen tak, že napřed je obsazený prostor soubory a za ním volné bloky

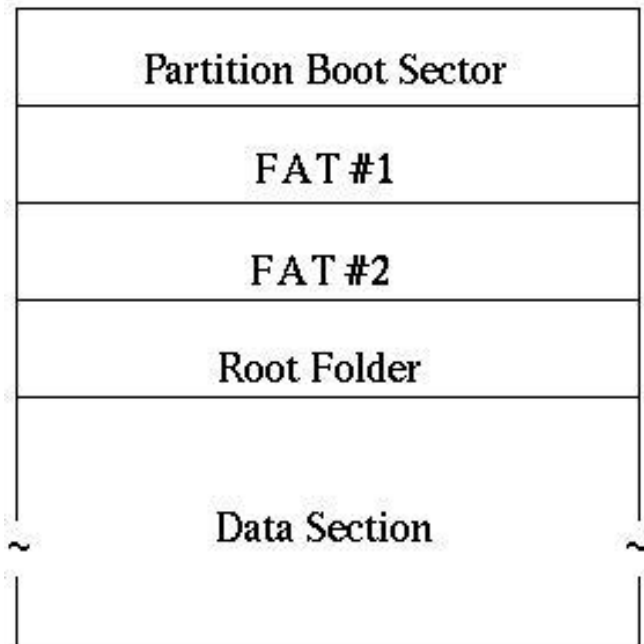
# FAT

---

- máme pevný disk (`/dev/sda`)
- např. druhá oblast disku (`/dev/sda2`)
- tato oblast disku je rozdělena:
  - boot block
  - FAT1 tabulka
  - FAT2 tabulka (záložní kopie)
  - datové bloky obsahující části souborů
    - první je adresář: jména souborů, velikost, kde začíná

# Diskový oddíl s FAT

---



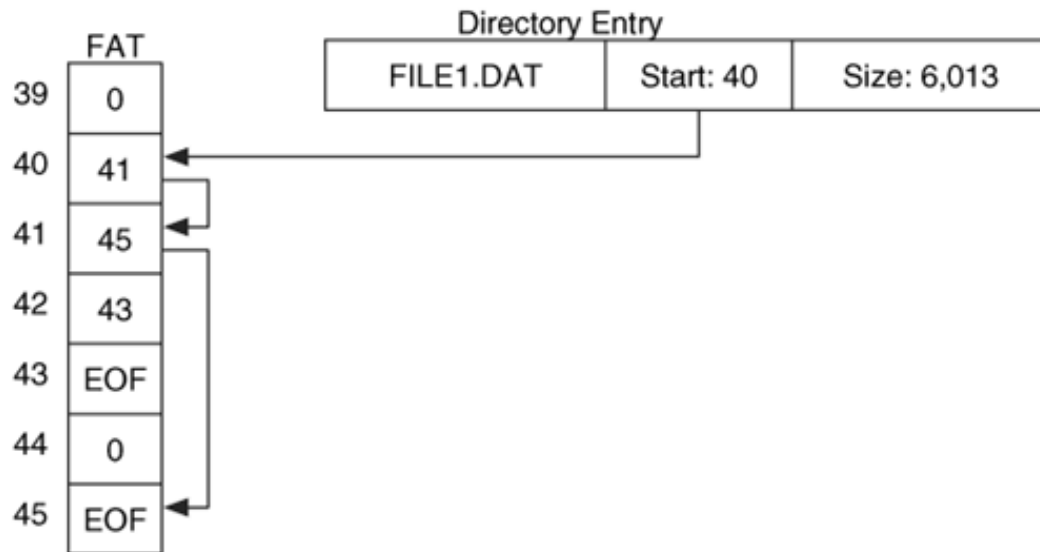
Diskový oddíl, který je naformátován na FAT (např. FAT32)

1. boot sektor (pokud se z daného oddílu bootuje)
2. kopie FAT tabulky, typicky jsou dvě
3. hlavní adresář daného oddílu
4. data

Zdroj obrázku:

<http://yliqepyh.keep.pl/fat-file-system-specifications.php>

# FAT – jak poznáme, kde soubor začíná?



Z adresáře poznáme:

- Jméno souboru
- Kde začíná (40)
- Velikost (6013 bytů)

Víme tedy, kde začíná soubor (40) a umíme určit, **jaká část posledního přiděleného bloku je využita daty souboru** (zbytek po dělení 6013 velikostí bloku)

V tomto obrázku by velikost bloku byla cca 2KB – 2048B  
Značka EOF je číslo symbolizující konec souboru.

# Vlastnosti FAT

---

Nevýhodou je velikost tabulky FAT

- 80GB disk, bloky 4KB => 20 mil. položek
- Každá položka alespoň 3 byty => 60MB FAT
- Výkonnostní důsledky (část FAT chceme v keši v RAM)

Použití

- Podporují DOS i Windows 10, Linux
- FAT12, 12 bitů,  $2^{12} = 4096$  bloků, diskety
- FAT16, 16 bitů,  $2^{16} = 65536$  bloků
- FAT32,  $2^{28}$  bloků, blok 4-32KB, cca 8TB
- ExFAT – používá B+ strom místo tabulky



# Příklady filesystemů (!!!)

---

## FAT

- MS DOS, paměťové karty
- Nepoužívá ACL – u souborů není žádná info o přístupových právech
- Jen atributy archive, hidden, read-only apod.
- Snadná přenositelnost dat mezi různými OS

## NTFS

- Používá se ve Windows XP/7.../10
- Používají ACL: k souboru je přiřazen seznam uživatelů, skupin a jaká mají oprávnění k souboru (!!!!)

## Ext2

- Použití v Linuxu, nemá žurnálování
- Práva – standardní unixová (vlastní, skupina, others), lze doplnit i ACL (komplexnější, ale samozřejmě přinesou zpomalení)

## Ext3

- Použití v Linuxu, má žurnál (rychlejší obnova konzistence po výpadku)

# Příklady filesystemů

---

## ext4

- stejně jako ext2, ext3 používá inody
- extenty – souvislé logické bloky
  - může být až 128MB oproti velkému počtu 4KB bloků
- nanosekundová časová razítka

## xf

## jfs

# NTFS

---

- nativní fs Windows od NT výše
- **žurnálování**  
všechny zápisy na disk se zapisují do žurnálu, pokud uprostřed zápisu systém havaruje, je možné dle stavu žurnálu zápis dokončit nebo anulovat => konzistentní stav
- **access control list**  
přidělování práv k souborům ( x FAT)
- **komprese**  
na úrovni fs lze soubor nastavit jako komprimovaný

# NTFS pokračování

---

- šifrování

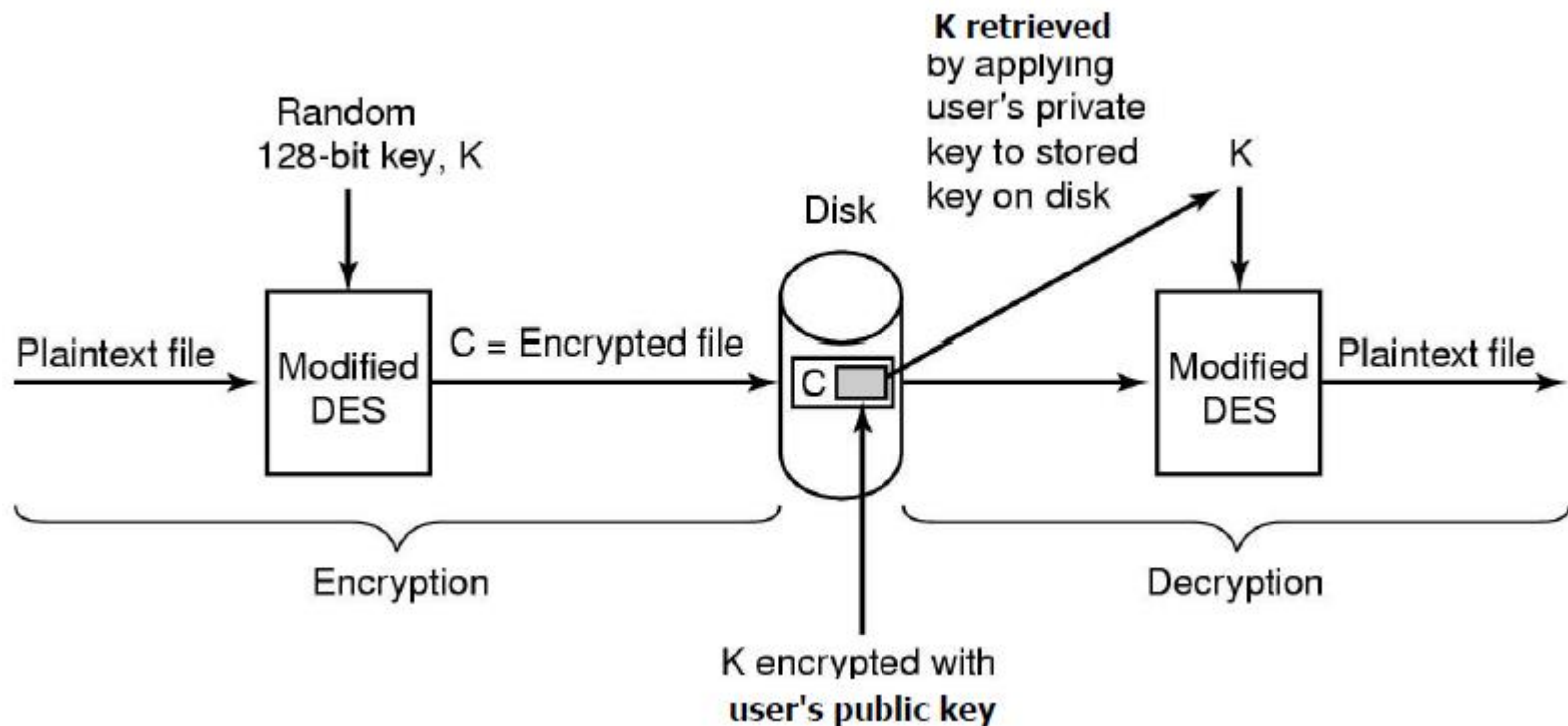
EFS (encrypting file system),  
transparentní – otevřu ahoj.txt, nestarám se, zda je  
šifrovaný

- diskové kvóty

max. velikost pro uživatele na daném oddíle  
dle reálné velikosti (ne komprimované)

- pevné a symbolické linky

# Šifrování



Při přístupu k souboru si odšifruji klíč K, a jeho pomocí získám zpět plaintext

# NTFS struktura (!!)

---

- 64bitové adresy clusterů .. cca 16EB
- clustery **číslovány od začátku partition** – logická čísla clusterů
- systém jako obří databáze  
záznam v ní odpovídá souboru
- základ 11 systémových souborů - metadata  
hned po formátování svazku
- **\$Logfile** – žurnálování
- **\$MFT** (Master File Table) – nejdůležitější (!!)  
záznamy o všech souborech, adresářích, metadatach  
hned za boot sektorem, za ním se udržuje zóna volného místa

# NTFS

---

Defaultní velikosti clusterů:

Volume size	NTFS cluster size
7MB – 512MB	512B
513MB – 1 024 MB	1KB
1 025MB – 2GB	2KB
2GB – 2TB	<b>4KB</b>

Při formátování si můžeme zvolit vlastní velikost clusterů až do 64KB

# NTFS struktura

---

- **\$MFTMirr** – uprostřed disku, obsahuje část záznamů \$MFT, při poškození se použije tato kopie
- **\$Badclus** – seznam vadných clusterů
- **\$Bitmap** – sledování volného místa  
0 – volný
- **\$Boot, \$Volume, \$AttrDef, \$Quota, \$Upcase, .**

podrobnosti:

<http://technet.microsoft.com/en-us/library/cc781134%28WS.10%29.aspx>



# NTFS atributy souborů

---

## \$FILE\_NAME

- jméno souboru
- velikost
- odkaz na nadřazený adresář
- další

## \$SECURITY\_DESCRIPTOR

- přístupová práva k souboru

## \$DATA

- vlastní obsah souboru



atributy  
definovány  
v  
\$AttrDef

# NTFS

---

## adresáře

- speciální soubory
- B-stromy se jmény souborů a odkazy na záznamy v MFT

## alternativní datové proudy (ADS)

- notepad poznamky.txt:**tajny.txt**
- často útočiště virů, škodlivého kódu

## zkopírováním souboru z NTFS na FAT

- ztratíme přístupová práva a alternativní datové proudy

# NTFS – způsob uložení dat (!!!)

---

- **kódování délkou běhu**

- od pozice 0 máme např. uloženo:  
A1, A2, A3, B1, B2, A4, A5, C1, ...

- soubor A bude popsán fragmenty

- **fragment**

- index
- počet bloků daného fragmentu

- v našem příkladě pro soubor A dva fragmenty:

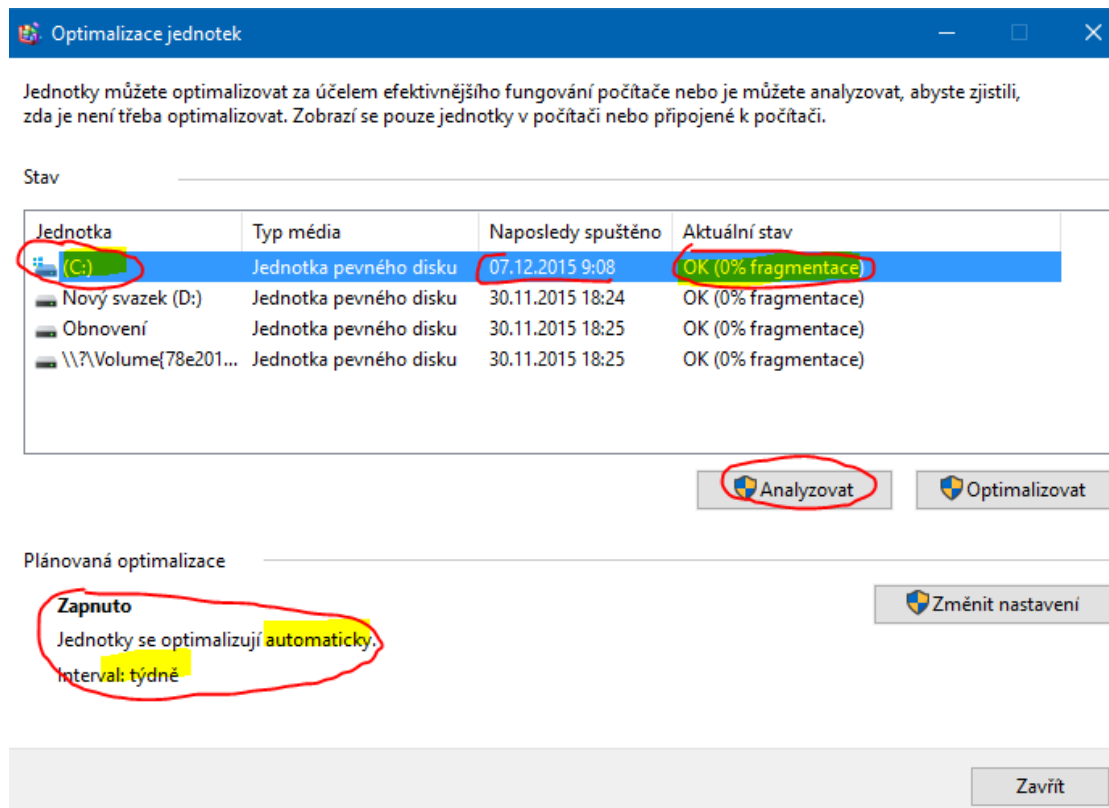
- 0, 3 (od indexu 0 patří tři bloky souboru A)
- 5, 2 (od indexu 5 patří dva bloky souboru A)

# NTFS – způsob uložení dat

---

- V **ideálním** případě **1 soubor = 1 fragment**  
(výhody kontinuální alokace)
- Defragmentovat můžeme jak celou partition, tak jen vybrané soubory (přes utility v sysinternals)
- Kontrola:
- Explorer -> disk C: -> pravá myš -> Vlastnosti -> Nástroje -> Optimalizovat a defragmentovat

# NTFS - defragmentace



# NTFS – ADS

## (Alternativní datové streamy)

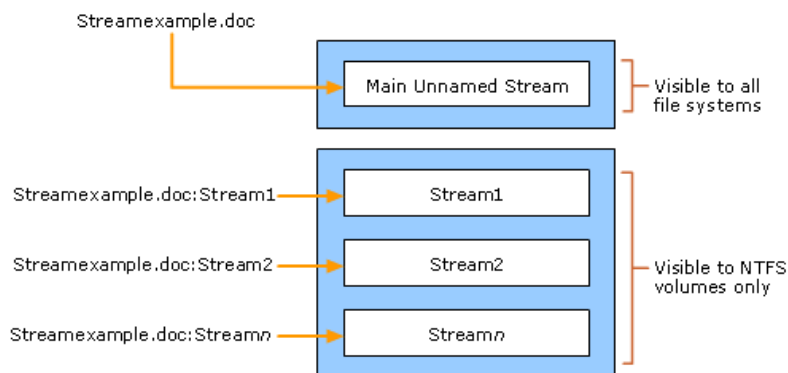
---

Na NTFS filesystému:

notepad poznamky.txt

notepad poznamky.txt:tajny.txt

### Unnamed and Named Streams



# NTFS – Sparse Files

---

Soubor 17GB

- Užitečná data 7GB
- Nuly 10GB

Na disku zabere 17GB, nebo 7GB u sparse file

FSUtil File CreateNew temp 0x100000

FSUtil Sparse SetFlag temp

FSUtil Sparse SetRange temp 0x100000

# Systemy využívající i-uzlů (!)

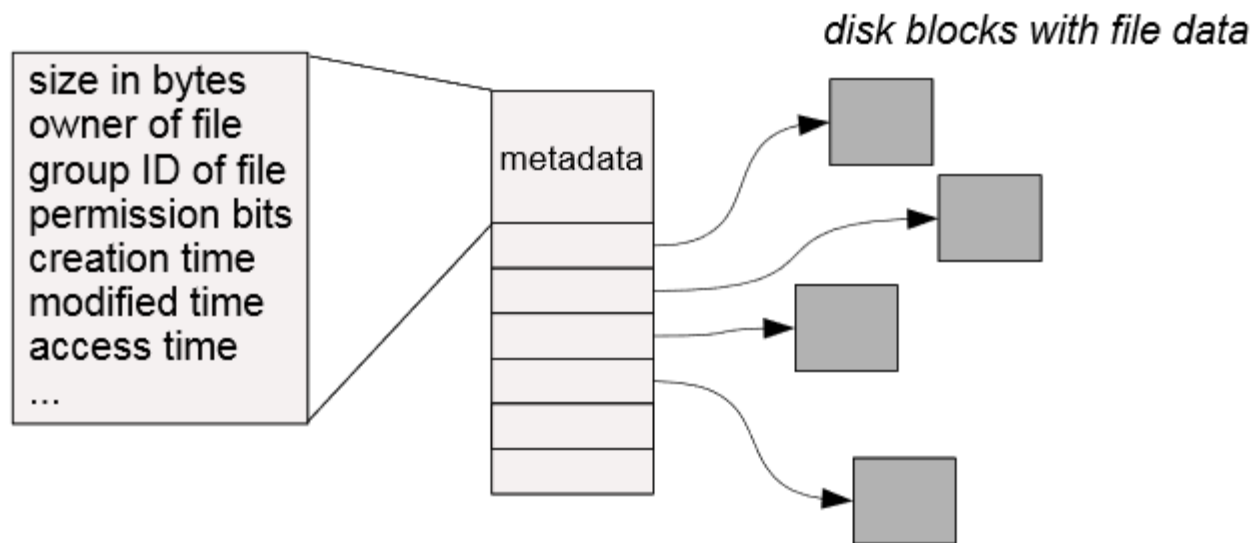
---

- Každý soubor (a tedy i adresář) je reprezentovaný i-uzlem (!!!!)
- **i-uzel** - datová struktura
  - **Metadata** popisující vlastníka souboru, přístupová práva, **velikost**
  - **Umístění bloků** souboru na disku
    - Přímé, nepřímé 1. 2. 3. úrovně
    - Abychom věděli, jaké bloky přistupovat



# i-uzel

i-uzel neobsahuje jméno souboru  
!!!

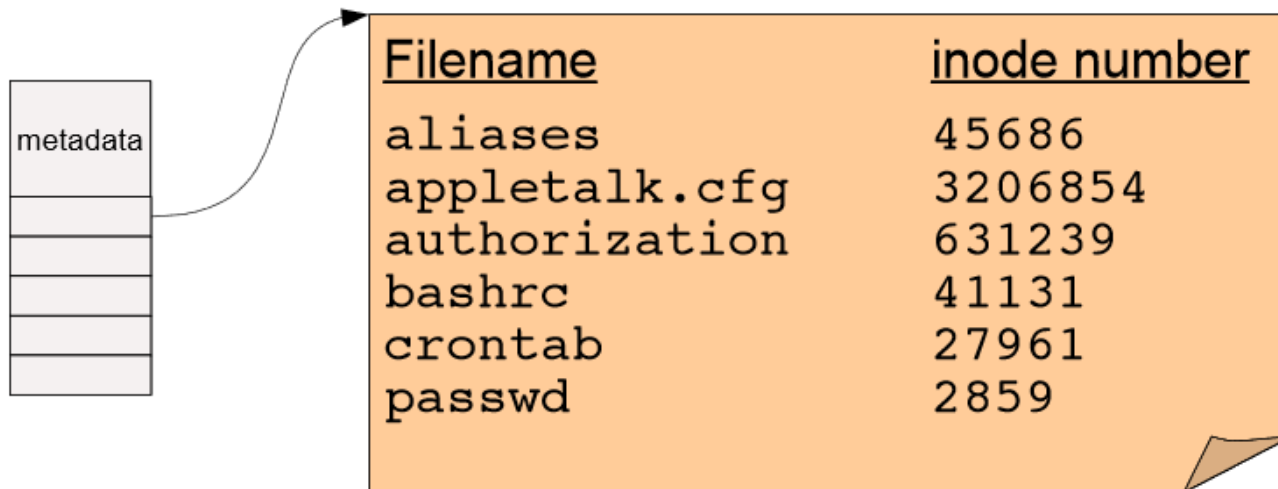


Obrázek znázorňuje jeden i-uzel (metadata , přímé adresy diskových bloků)  
Pamatuj: 1 i-uzel = 1 soubor (obyčejný, adresář)

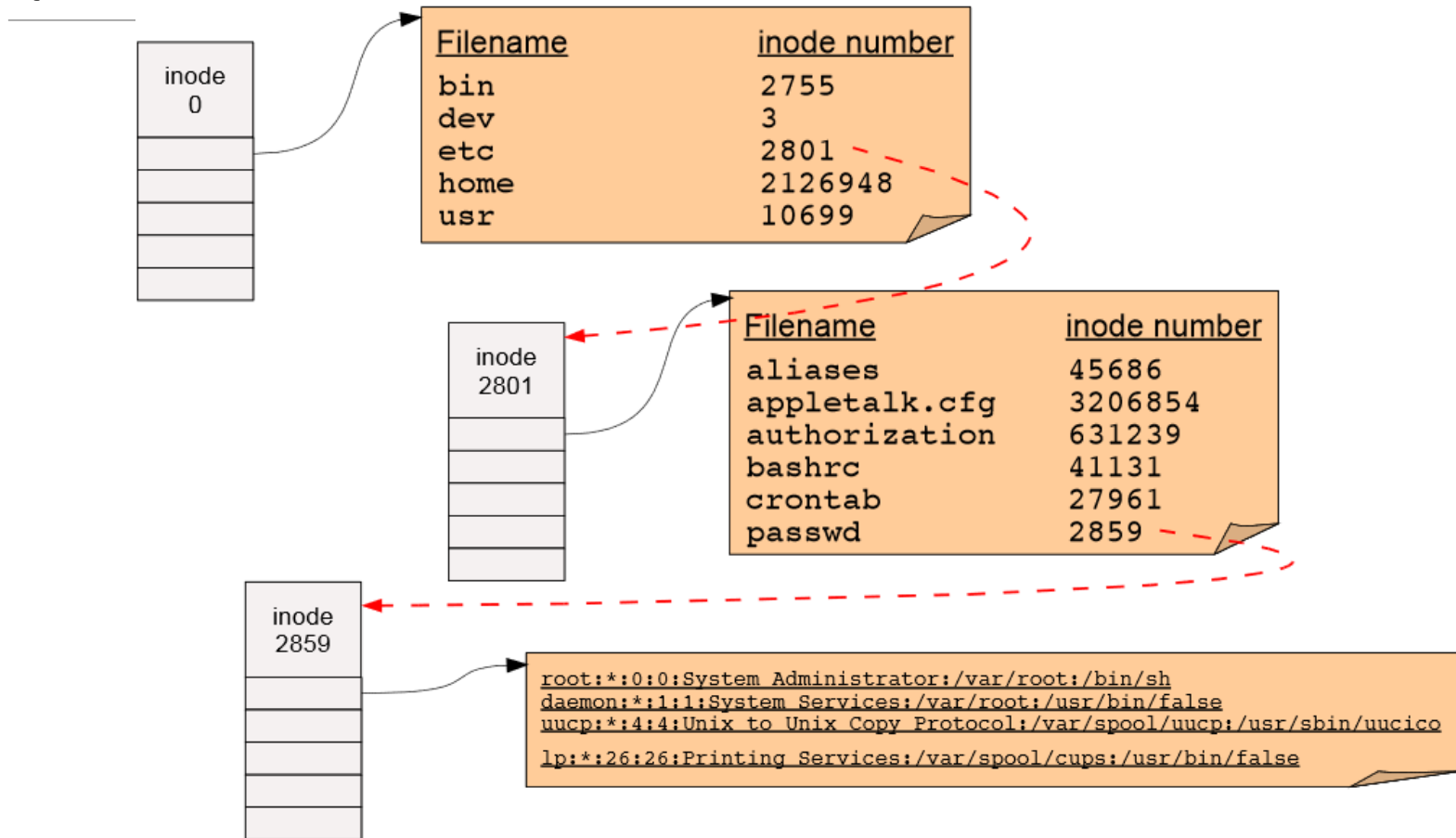
# Adresář systému s i-uzly (!)

---

Soubor obsahující dvojici  
(název\_souboru, číslo\_odpovídajícího\_i-uzlu)



# Prohledání cesty k souboru (zde např. /etc/passwd)



Projít: hlavní adresář, adresář etc a následně vlastní soubor passwd

# Umístění i-uzlů na disku

---

- 1 i-uzel = 1 soubor
- Pevný počet i-uzlů = max. počet souborů na daném oddílu disku (určeno při vytvoření fs)
- Pokud nám dojdou i-uzly, další soubor již nemůžeme vytvořit, ale pokud zbývají datové bloky, můžeme prodloužit stávající soubory

# Unixové systémy s využitím i-uzlů (původní koncepce)

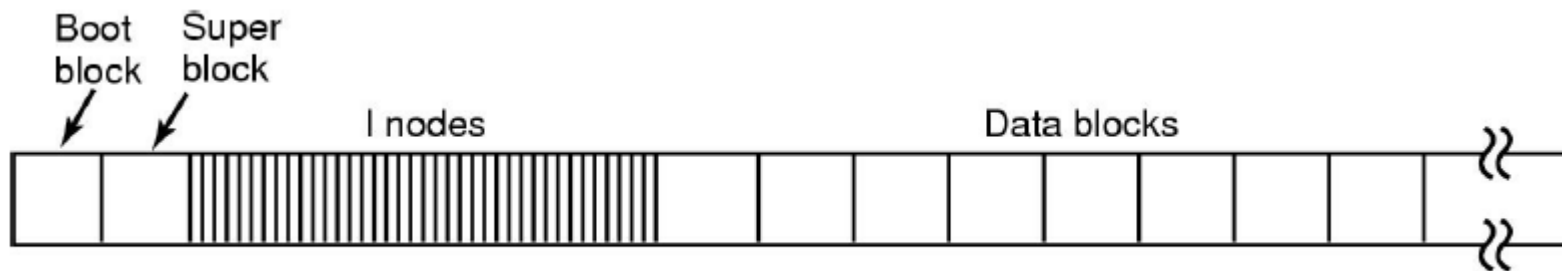
---

takto vypadá partiton disku (např. `/dev/sda1`)

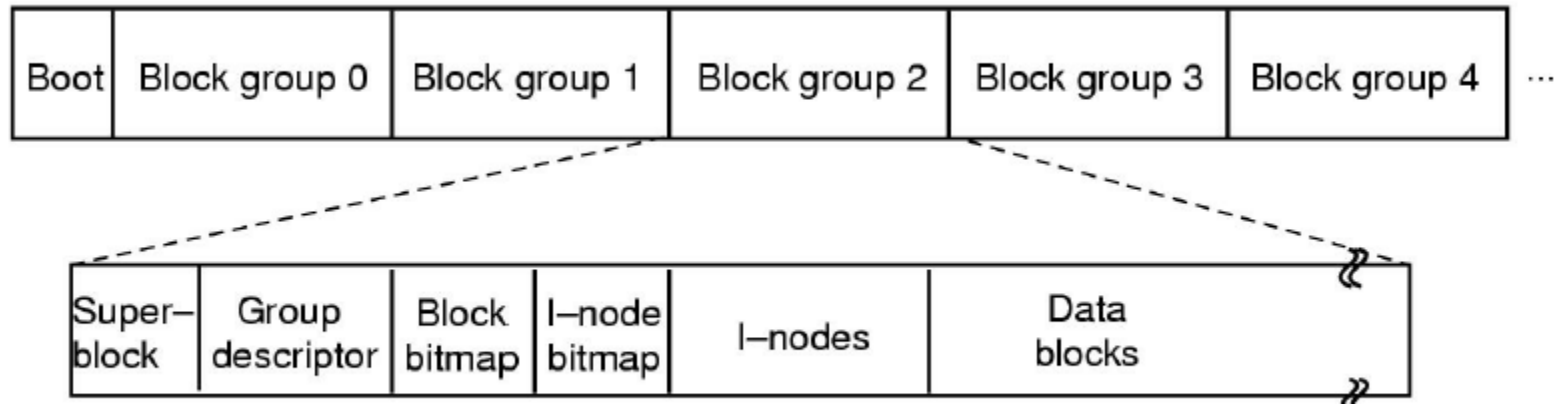
původní rozdělení v Unixových systémech

novější rozdělení, např. v ext2 viz další slide

superblock – příznak čistoty, verze, počet i-nodů, velikost alokační jednotky, seznam volných bloků



# Unixové systémy s využitím i-uzlů (novější, např. ext2)



skupiny i-nodů a datových bloků v jednotlivých skupinách (block group)  
**duplikace nejdůležitějších údajů** v každé skupině (superblock, group descriptor)

Jsou zde 2 bitmapy – který i-node je volný, který blok je volný (!!!)

# Vlastnosti

---

- klíčové informace jsou násobně duplikovány, např. superblock
- Bitmapa i-nodů říká, který i-node je volný
- Bitmapa datových bloků říká, který datový blok je volný
- I-nody i odpovídající data jsou blízko u sebe
- Chci vytvořit nový soubor
  - V bitmapě najdu volný i-node
  - Dále hledám v bitmapě datových bloků volné bloky pro data

# I-uzly (!!)

---

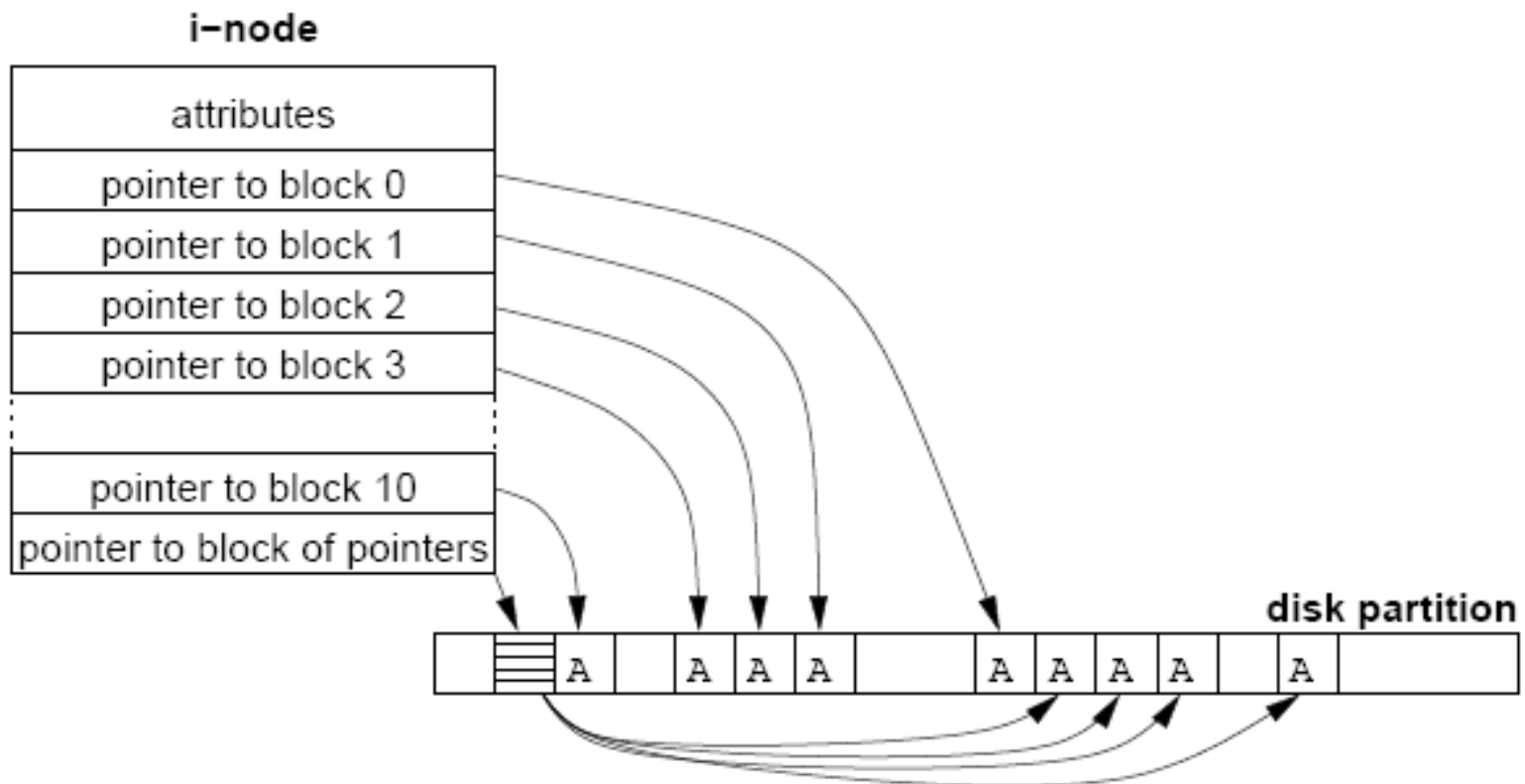
- S každým souborem sdružena datová struktura **i-uzel** (i-node, zkratka z index-node)
- i-uzel obsahuje
  - Atributy souboru  
(**velikost** souboru, **počet odkazů** na soubor, práva a pro koho jsou, časy vytvoření, modifikace,...)
  - Diskové adresy prvních N bloků souboru
  - 1 či více odkazů na diskové bloky obsahující další diskové adresy (případně obsahující odkazy na bloky obsahující adresy) – 1. 2. 3. nepřímé úrovně



# Poznámka

---

- Systémy s i-uzly jsou tradiční pro Unix
- Používají je dnešní filesystemy např. **ext2, ext3, ext4**



Pamatuj:

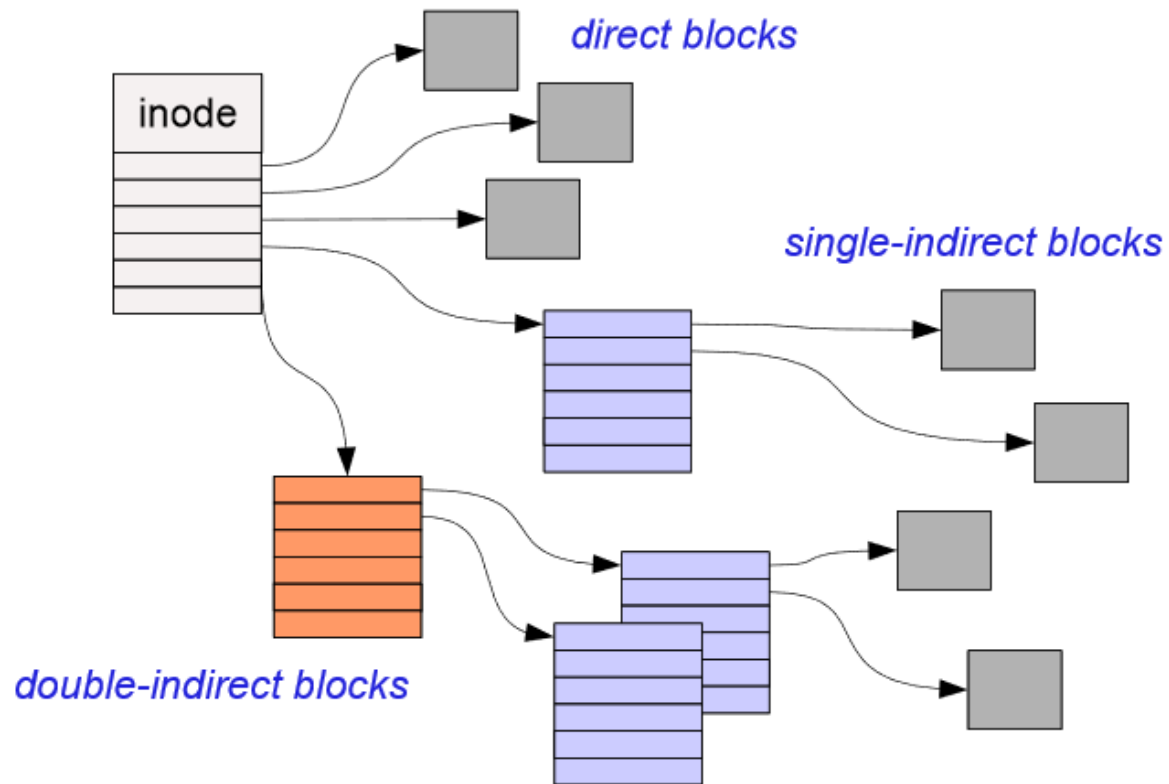
Cca **10-12 přímých odkazů** na bloky obsahující data souboru

1. **nepřímý** – odkaz na datový blok obsahující seznam odkazů na data
2. **nepřímý** – viz další slide
3. **nepřímý** – viz další slide

# Důležité

---

- Nepřímé odkazy
  - Datový blok, místo aby obsahoval data souboru, tak obsahuje **odkazy na další datové bloky** využívané souborem
  - Datový blok tedy obsahuje **metadata** (zde ukazatele), místo dat souboru
- Mohou být 1., 2., 3. úrovně
  - Odkaz na blok z i-nodu -> data souboru (přímé odkazy)
  - Odkaz na blok z i-nodu -> metadata -> data souboru (1. úroveň)
  - Odkaz na blok z i-nodu -> metadata -> metadata -> data souboru (2. úroveň)
  - Odkaz na blok z i-nodu -> md -> md -> md -> data souboru (3. úroveň)



© 2007 Matt Welsh – Harvard University

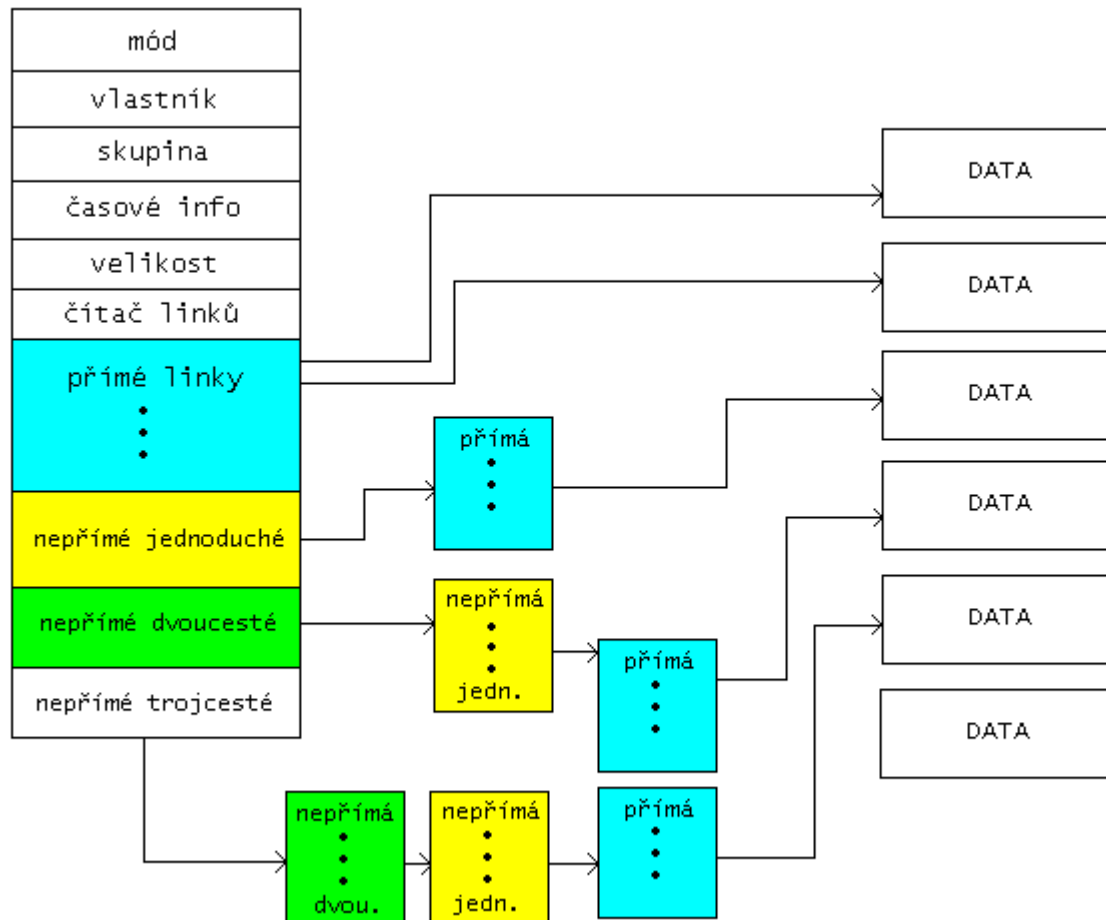
- Malé soubory – přímé odkazy na datové bloky => rychlý přístup k nim
- Velké soubory – využívají i nepřímé odkazy
- i-uzel má pevnou velikost – stejnou pro malý i velký soubor

# I-uzly - výhoda

---

- Po otevření souboru můžeme **zavést i-uzel** a případný blok obsahující další adresy **do paměti** => urychlení přístupu k souboru

# i-uzly dle normy POSIX



zdroj: <http://cs.wikipedia.org/wiki/Inode>

# i-uzly dle normy Posix

---

- **MODE** – typ souboru, **přístupová práva** (u,g,o)
- **REFERENCE COUNT** – počet odkazů na tento objekt  
(vytvoření hardlinku zvyšuje počet)
- **OWNER** – ID vlastníka
- **GROUP** – ID skupiny
- **SIZE** – velikost objektu
- **TIME STAMPS**
  - atime – čas posledního přístupu (čtení souboru, výpis adresáře)
  - mtime – čas poslední změny
  - ctime – čas poslední změny i-uzlu (metadat)

# i-uzly dle normy POSIX

- **DIRECT BLOCKS** – 12 přímých odkazů na datové bloky (data v souboru)
- **SINGLE INDIRECT** – 1 odkaz na datový blok, který **místo dat** obsahuje seznam přímých odkazů na datové bloky obsahující vlastní data souboru
- **DOUBLE INDIRECT** – 1 odkaz 2. nepřímé úrovně
- **TRIPLE INDIRECT** – 1 odkaz 3. nepřímé úrovně

v linuxových fs (ext\*) ještě FLAGS, počet použitých datových bloků a rezervovaná část – doplňující info  
(odkaz na rodičovský adresář, **ACL**, rozšířené atributy)



# Implementace adresářů

---

- Před čtením je třeba soubor otevřít
- *open (jméno, režim)*
- Mapování jméno -> info o datech  
poskytují adresáře !
- Adresáře jsou často speciálním typem souboru
- Typicky pole datových struktur, 1 položka na soubor

# 2 základní uspořádání adresáře (!!!)

---

1. Adresář obsahuje **jméno souboru, atributy, diskovou adresu souboru** (např. adresa 1.bloku) (implementuje DOS, Windows)
2. Adresář obsahuje **pouze jméno + odkaz** na jinou datovou strukturu obsahující další informace (např. i-uzel) (implementuje UNIX, Linux)

Běžné jsou oba dva způsoby i kombinace

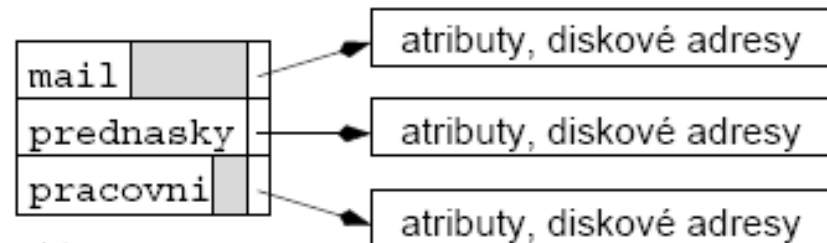
# 2 základní uspořádání adresáře (!!)

---

mail		atributy, diskové adresy
prednasky		atributy, diskové adresy
pracovni		atributy, diskové adresy

a)

- a) Používá např. FAT  
musí vědět:
- na jakém bloku soubor začíná
  - jak je soubor velký



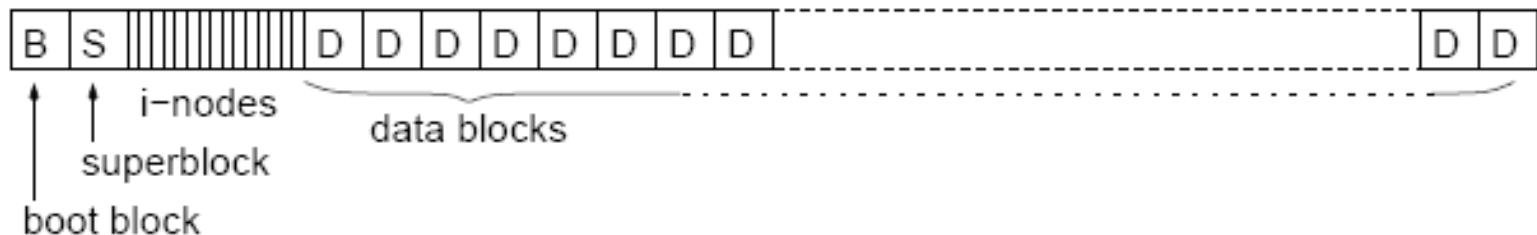
b)

- b) Používá např. ext2, ext3  
obecně systémy s i-uzly  
(i-uzel neobsahuje jméno souboru)

# Příklad filesystemu (Unix v7)

## ■ Struktura fs na disku

- **Boot blok** – může být kód pro zavedení OS
- **Superblok** – informace o fs  
(počet i-uzlů, datových bloků, odkaz na seznam volných bloků..)
- **i-uzly** – tabulka pevné velikosti, číslovány od 1
- **Datové bloky** – všechny soubory a adresáře



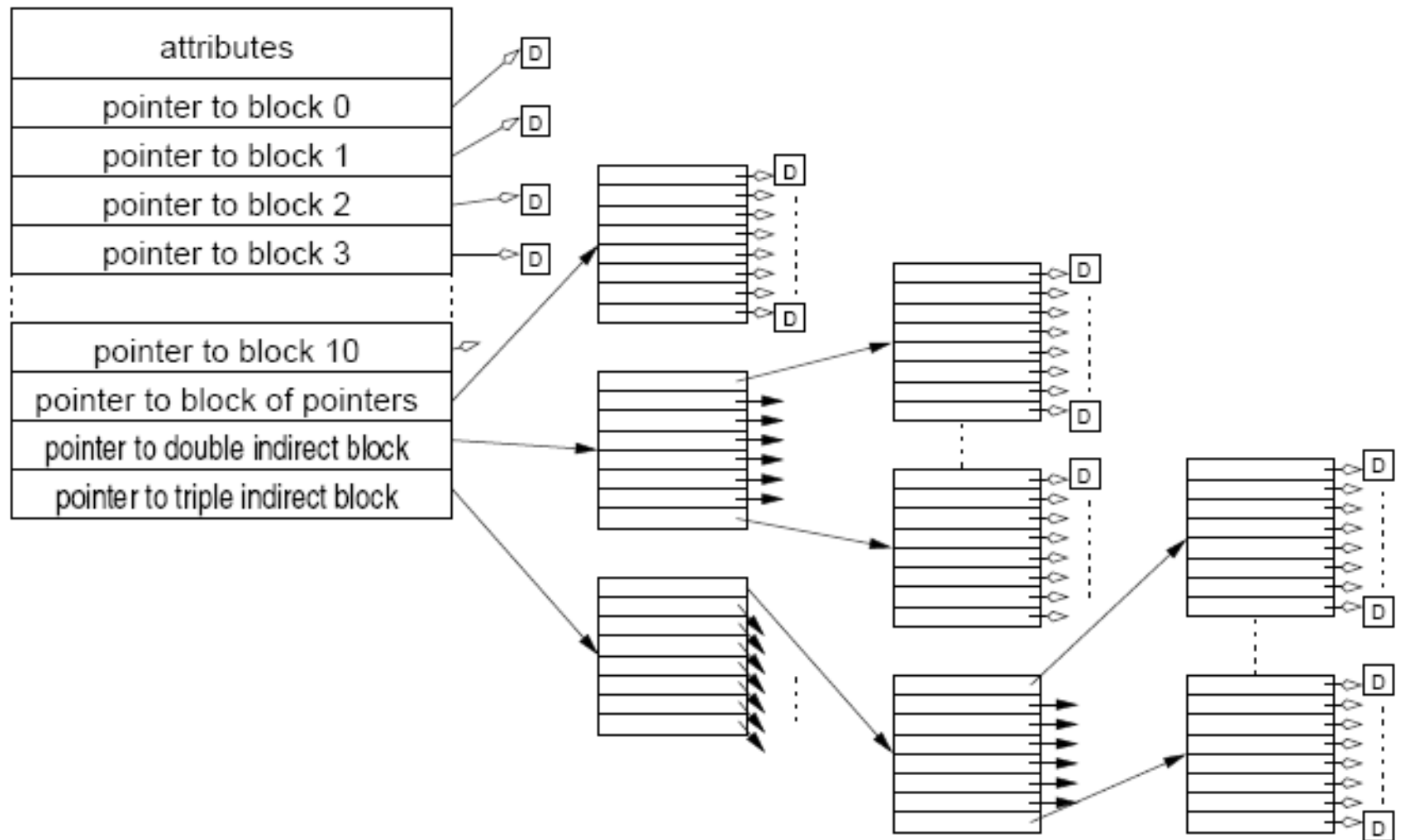
# Implementace souborů – i-uzly

---

i-uzel obsahuje:

- Atributy
- Odkaz na prvních 10(až 12) datových bloků souboru
- Odkaz na blok obsahující odkazy na datové bloky (**nepřímý odkaz**)
- Odkaz na blok obsahující odkazy na bloky obsahující odkazy na datové bloky (**dvojitě nepřímý odkaz**)
- **Trojitě nepřímý odkaz**

## UNIX v7 i-node



# Pokračování příkladu

---

- Implementace adresářů:  
tabulka obsahující jméno souboru a číslo jeho i-uzlu
- Info o volných blocích  
seznam, jeho začátek je v superbloku
- Základní model, současné fs jsou na něm založeny

# Adresáře v UNIX v7

---

adresář: obsahuje jméno souboru a číslo i-uzlu

Číslo i-uzlu je indexem do tabulky i-uzlů na disku

Každý soubor a adresář: právě 1 i-uzel

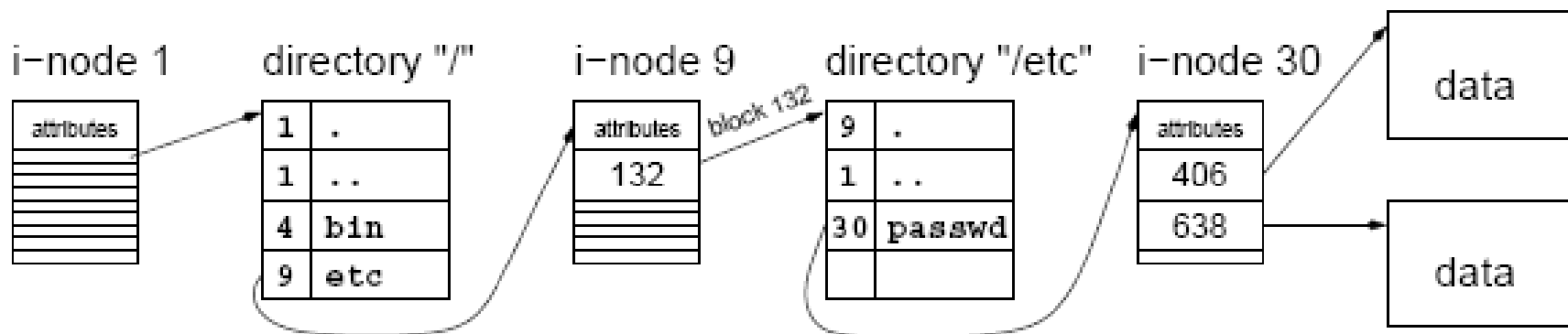
V i-uzlu: všechny atributy a čísla diskových bloků

Kořenový adresář: číslo i-uzlu 1



## Nalezení cesty k souboru `/etc/passwd`

- V kořenovém adresáři najdeme položku „`etc`“
- i-uzel číslo **9** obsahuje adresy diskových bloků pro adresář `etc`
- V adresáři `etc` (disk blok 132) najdeme položku `passwd`
- i-uzel **30** obsahuje soubor `/etc/passwd`
- (uzel, obsah uzlu, uzel, obsah uzlu)



# Příklad – adresář win 98

---

\* položka adresáře obsahuje:

- jméno souboru (8 bytů) + příponu (3 byty)
- atributy (1)
- NT (1) - Windows 98 nepoužívají (rezervováno pro WinNT)
- datum a čas vytvoření (5)
- čas posledního přístupu (2)
- horních 16 bitů počátečního bloku souboru (2)
- čas posledního čtení/zápisu
- spodních 16 bitů počátečního bloku souboru (2)
- velikost souboru (4)

\* dlouhá jména mají pokračovací položky

\* veškeré "podivnosti" této struktury jsou z důvodu kompatibility s MS DOSem

cd Progra~1 vs. cd Program Files

# Sdílení souborů

---

Soubor ve více podadresářích nebo pod více jmény

## Hard links (pevné odkazy)

- Každý soubor má datovou strukturu, která ho popisuje (i-uzel), můžeme vytvořit v adresářích více odkazů na stejný soubor
- Všechny odkazy (jména) jsou rovnocenné
- V popisu souboru (i-uzlu) musí být počet odkazů
- Soubor zanikne při zrušení posledního odkazu

Hard link v Linuxu: **ln** stare\_jmeno nove\_jmeno

# Sdílení souborů

---

## Symbolický link

- Nový typ souboru, obsahuje jméno odkazovaného souboru
- OS místo symbolického odkazu otevře odkazovaný soubor
- Obecnější – může obsahovat cokoliv
- Větší režie

Symbolický link v Linuxu: **ln -s** stare\_jmeno nove\_jmeno

# Správa volného prostoru

---

Info, které bloky jsou volné

Nejčastěji – bitová mapa nebo seznam

## Bitová mapa

- Konstantní velikost
- Snažší vyhledávání volného bloku s určitými vlastnostmi
- Většina současných fs používá bitovou mapu

# Správa volného prostoru

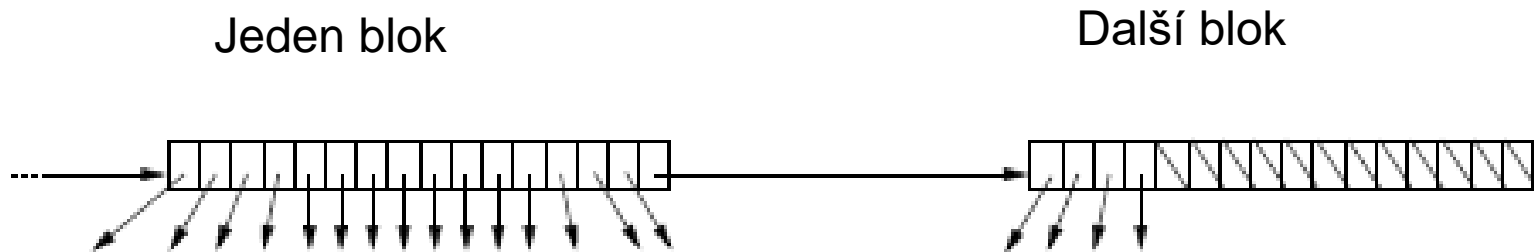
---

## Seznam diskových bloků

- Blok obsahuje **odkazy na volné bloky** a **adresu dalšího bloku** ...
- Uvolnění bloků - Přidáme adresy do seznamu, pokud není místo blok zapíšeme
- Potřebujeme bloky pro soubor – používáme adresy ze seznamu, pokud nejsou přečteme další blok adres volných bloků
- Pokud není na disku volné místo, seznam volných bloků je prázdný a nezabírá místo
- Problém najít volný blok s určitými vlastnostmi (např. ve stejném cylindru), prohledávat seznam, drahé,...

# Seznam diskových bloků

---



Blok obsahuje:

- Odkazy na volné bloky
- Adresu dalšího bloku v seznamu

# Kvóty

Účel – aby uživatel neobsadil celý disk a nechal místo i pro ostatní

Maximální počet bloků obsazených soubory uživatele

Ve víceuživatelských OS, na serverech

## Hard kvóta

- Pevná mez, uživatel ji nepřekročí

## Soft kvóta

- Po překročení uživatel dostane varování
- **Grace period** – po zadanou dobu může překročit soft kvótu, po uplynutí času už více neuloží



# Spolehlivost souborového systému

---

**Ztráta dat** má často horší důsledky než zničení počítače

- diplomová – bakalářská práce
- Fotografie za posledních 10 let

Filesystém musí být jedna z nejspolehlivějších částí OS, snaha chránit data

- Správa vadných bloků (hlavně dříve)
- Rozprostřít a duplikovat důležité datové struktury, čitelnost i po částečném poškození povrchu

# Konzistence fs

---

## Blokové zařízení

OS přečte blok souboru, změní ho, zapíše

## Nekonzistentní stav

může nastat při havárii (např. výpadek napájení) předtím, než jsou všechny modifikované bloky zapsány

## Kontrola konzistence fs

Windows: scandisk, **chkdsk**

UNIX: fsck , fsck.ext3, e2fsck .. viz man

Kontrolu spustí automaticky po startu, když detekuje nekorektní ukončení práce se systémem

# Testy konzistence fs

---

## Konzistence informace o diskových blocích souborů

- Blok (obvykle) patří jednomu souboru nebo je volný

## Konzistence adresářové struktury

- Jsou všechny adresáře a soubory dostupné?

důležité pochopit rozdíl:

- kontrola konzistence souboru
- kontrola, zda je soubor dostupný z nějakého adresáře

# Konzistence informace o diskových blocích souborů

---

Tabulka počtu výskytů bloku v souboru

Tabulka počtu výskytů bloku v seznamu volných bloků

Položky obou tabulek inicializovány na 0

Procházíme informace o souborech (např. i-uzly), inkrementujeme položky odpovídající blokům souboru v první tabulce

Procházíme seznam nebo bitmapu volných bloků a inkrementujeme příslušné položky ve druhé tabulce

# Konzistentní fs

---

Číslo bloku	0	1	2	3	4	5	6	7	8
Výskyt v souborech	1	0	1	0	1	0	2	0	1
Volné bloky	0	1	0	0	1	2	0	1	0

Blok je buď volný, nebo patří nějakému souboru, tj. konzistentní hodnoty v daném sloupci jsou buď (0,1) nebo (1,0)  
Vše ostatní jsou chyby různé závažnosti

# Možné chyby, závažnosti

---

(0,0) – blok se nevyskytuje v žádné tabulce

- Missing blok
- Není závažné, pouze redukuje kapacitu fs
- Oprava: vložení do seznamu volných bloků

(0,2) – blok je dvakrát nebo vícekrát v seznamu volných

- Problém – blok by mohl být alokován vícekrát !
- Opravíme seznam volných bloků, aby se vyskytoval pouze jednou

# Možné chyby, závažnosti

---

## (1,1) – blok patří souboru a zároveň je na seznamu volných

- Problém, blok by mohl být alokován podruhé !
- Oprava: blok vyjmemme ze seznamu volných bloků

## (2,0) – blok patří do dvou nebo více souborů

- Nejzávažnější problém, nejspíš už došlo ke ztrátě dat
- Snaha o opravu: alokujeme nový blok, problematický blok do něj zkopírujeme a upravíme i-uzel druhého souboru
- Uživatel by měl být informován o problému

# Je zde nějaká chyba? A když tak jaká?

---

	číslo bloku:	0	1	2	3	4	5	6	7	8	9	11	12	13	14	15
výskyt	v souborech:	1	1	0	0	1	0	0	1	1	1	1	0	1	0	0
	volné bloky:	0	0	1	1	0	1	1	0	0	0	0	1	0	1	1

	číslo bloku:	0	1	2	3	4	5	6	7	8	9	11	12	13	14	15
výskyt	v souborech:	1	2	0	0	1	0	0	1	1	1	1	1	1	0	0
	volné bloky:	0	0	1	1	0	0	1	0	0	0	0	1	0	1	1



# Kontrola konzistence adresářové struktury

---

Tabulka čítačů, jedna položka pro každý soubor

Program prochází rekurzivně celý adresářový strom

Položku pro soubor program zvýší pro každý výskyt souboru v adresáři

Zkontroluje, zda odpovídá počet odkazů v  $i$ -uzlu ( $i$ ) s počtem výskytů v adresářích ( $a$ )

$i == a$  😊 pro každý soubor

# Možné chyby

---

**i > a**

soubor by nebyl zrušen ani po zrušení všech odkazů v adresářích

není závažné, ale soubor by zbytečně zabíral místo  
řešíme nastavením počtu odkazů v i-uzlu na správnou hodnotu (a)

# Možné chyby

---

**i < a**

soubor by byl zrušen po zrušení i odkazů, ale v adresářích budou ještě jména

velký problém – adresáře by ukazovaly na neexistující soubory

řešíme nastavením počtu odkazů na správnou hodnotu

# Možné chyby

---

**a=0 , i > 0**

**ztracený soubor, na který není v adresáři odkaz**

ve většině systémů program soubor zviditelní na předem určeném místě

(např. adresář lost+found)

# Další heuristické kontroly

---

Odpovídají jména souborů konvencím OS?

- Když ne, soubor může být nepřístupný, změníme jméno

Nejsou přístupová práva nesmyslná?

- Např. vlastník nemá přístup k souboru,...

Zde byly uvedeny jen základní obecné kontroly fs

# Journaling fs

---

Kontrola konzistence je časově náročná

## Journaling fs

- Před každým zápisem na disk vytvoří na disku záznam popisující plánované operace, pak provede operace a záznam zruší
- Výpadek – na disku najdeme žurnál o všech operacích, které mohly být v době havárie rozpracované, zjednodušuje kontrolu konzistence fs

Příkladem fs s žurnálem je např. ext3, ext4

# Jak funguje žurnál (!!)

---

1. Zapíši do žurnálu
2. Když je žurnál kompletní, zapíšeme značku ZURNAL\_KOMPLETNI
3. Začneme zapisovat datové bloky
4. Je-li hotovo, smažeme žurnál

# Žurnál – ošetření výpadku (!!)

---

Dojde-li k výpadku elektřiny → nebyl korektně odmontovaný oddíl se souborovým systémem → pozná

Podívá se do žurnálu:

- a) **Je prázdný**  
→ není třeba nic dělat
- b) **Je tam nějaký zápis, ale není značka ZURNAL\_KOMPLETNI**  
→ jen smažeme žurnál
- c) **V žurnálu je zápis včetně značky ZURNAL\_KOMPLETNI**  
→ přepíšeme obsah žurnálu do datových bloků



# Co žurnálovat?

---

Všechny zápisy, tj. i do souborů

- Zapisují se metadata i data
- pomalejší

Zápisy metadat

- Rychlejší
- Může dojít ke ztrátě obsahu souboru, ale nerozpadne se struktura adresářů

# Výkonnost fs

---

Přístup k tradičnímu disku řádově pomalejší než přístup do paměti

- Seek 5-10 ms
- Rotační zpoždění – až bude požadovaný blok pod hlavičkou disku
- Rychlost čtení (x rychlost přístupu do paměti)

Použití **SSD** disků

- Rychlé, lehké, malá spotřeba (výdrž notebooků)
- menší kapacita, drahé
- 256GB cca 3-4 tisíce Kč

# Výkonnost fs, cachování

---

Cachování diskových bloků v paměti

Přednačítání (read-ahead)

do cache se předem načítají bloky, které se budou potřebovat při sekvenčním čtení souboru:

čtu blok A10 a rovnou nakešuji i blok A11

Redukce pohybu diskového raménka pro po sobě následující bloky souboru,...

# Mechanismy ochrany

---

Chránit soubor před neoprávněným přístupem

Chránit i další objekty

- HW (segmenty paměti, I/O zařízení)
- SW (procesy, semafore, ...)

**Subjekt** – entita schopná přistupovat k objektům  
(většinou **proces**)

**Objekt** – cokoliv, k čemu je potřeba omezovat přístup pomocí  
přístupových práv (např. **soubor**)

System **uchovává informace** o přístupových právech subjektů k  
objektům

# ACL x capability list

---

Dvě různé podoby

**ACL** – s objektem je sdružen seznam subjektů a jejich přístupových práv

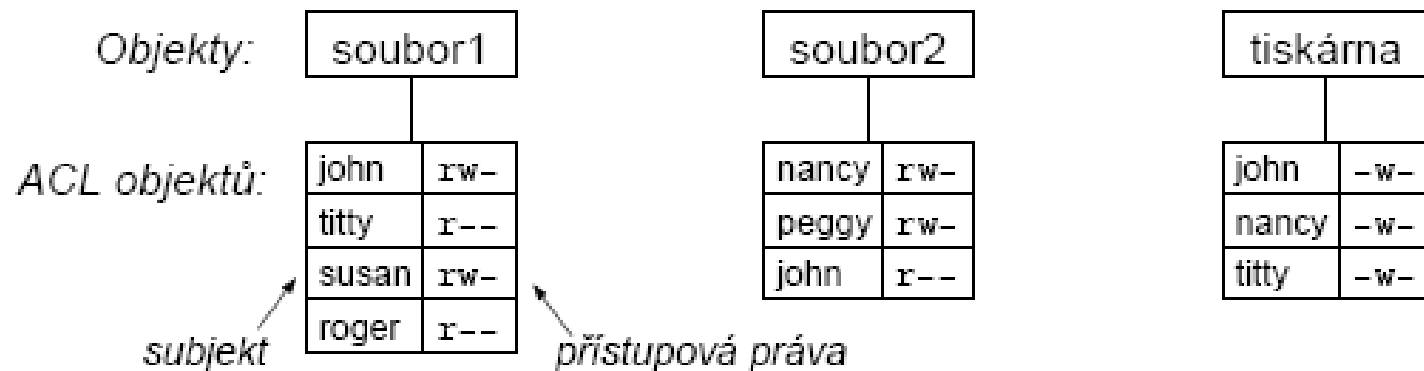
**Capability list** [kejpa-] – se subjektem je sdružen seznam objektů a přístupových práv k nim

# ACL (Access Control Lists)

---

S objektem je sdružen seznam subjektů, které mohou k objektu přistupovat

Pro každý uvedený subjekt je v ACL množina přístupových práv k objektu



# ACL

---

Sdružování subjektů do tříd nebo do skupin

- Studenti
- Zamestnanci

Skupiny mohou být uvedeny na místě subjektu v ACL

Zjednodušuje administraci

- Nemusíme uvádět všechny studenty jmenovitě

ACL používá mnoho moderních filesystemů  
(ntfs, xfs, ...)

# ACL – příklad (!)

---

Např v **NTFS**:

Se souborem **data1.txt** je spojena následující **ACL tabulka**, která určuje, kdo smí co s daným souborem dělat. Počet řádek tabulky záleží na tom, pro kolik uživatelů skupin budeme práva nastavovat.

Klasická unixová práva (u,g,o) jsou příliš limitovaná – když chceme více skupin, více uživatelů atd. potřebujeme ACL.

uživatel / skupina	id uživatele	práva
0	505 (Pepa)	rw
1	101 (Studenti)	r
1	102 (Zamestnanci)	rw



# Klasická unixová práva

---

- **chmod** 777 s1.txt
  - Práva pro vlastníka (u)
  - Práva pro skupinu (g)
  - Práva pro ostatní (o)
  - Typ práv: r, w, x, (s, t)
- Oproti ACL jsou omezující:
  - Chceme pro více skupin různá nastavení
  - Chceme pro více uživatelů různá nastavení

Klasická unixová práva u,g,o nejsou považována za ACL, Naopak systémy které je využívají se o ACL rozšiřují

# Klasická práva vs. ACL

---

ACL má přednost před klasickými unixovými právy  
(nastavenými chmod)

Tj. nastavením chmod 777 nezrušíme ACLka 😊

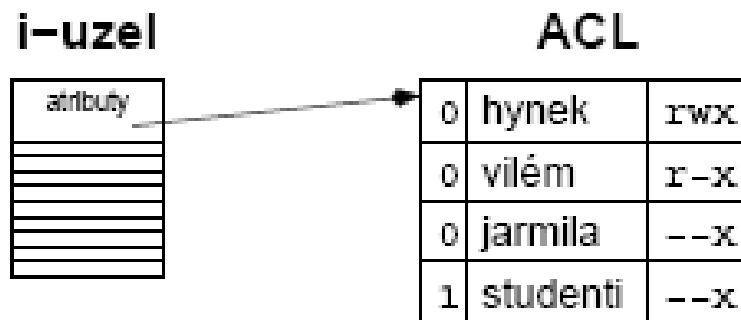
# Úloha: jak doimplementovat ACL do i-uzlu?

---

V i-uzlu by byla **část tabulky ACL**, pokud by se nevešla celá do i-uzlu, tak **odkaz na diskový blok** obsahující **zbytek ACL**

Každá položka ACL

- Subjekt: id uživatele či id skupiny + 1 bit rozlišení uživatel/skupina
- Přístupové právo Nbitovým slovem  
1 – právo přiděleno, 0 – právo odejmuto



# ACL příkazy pod Linuxem

---

**getfacl** soubor

**setfacl** -m user:pepa:rw s1.txt

Přečtěte si článek:

<http://www.abclinuxu.cz/clanky/bezpecnost/acl-prakticky>

# ACL a Linuxové fs

---

```
tune2fs -o acl /dev/sdXY
```

ACL je defaultní mount option u ext2,3,4 souborových systémů

```
setfacl -m "u:johny:rwx" abc
```

- Nastaví práva pro uživatele johny na soubor abc

```
getfacl abc
```

# ACL a Linuxové fs

---

getfacl abc

# file: abc

# owner: someone

# group: someone

user::rw-

user:johny:rwx

group::r--

mask::rwx

other::r--

# Jak zjistím, že se ACL používá?

---

## Identifying files/directories that have ACL's

While the standard unix permissions are displayed with the `ls -l` command; the defined ACL's are a little more verbose and are not a part of the long listing. The command `ls` will tell you if a file or directory does have acl's, it's just not that obvious.

```
root@testvm:/var/tmp# ls -la | grep appdir
drwxrwxr-x+ 2 root appgroup 4096 May 27 10:45 appdir
```

As you can see there is now a `+` at the end of the directories permissions. This `+` is the indicator that this file or directory has acl's, from here you can use the `getfacl` command to see what they are.

# Mechanismus capability lists (C-seznamy)

---

- S každým subjektem (procesem) sdružen seznam objektů, kterým může přistupovat a jakým způsobem (tj. přístupová práva)
- Seznam se nazývá capability list (C-list)
- Jednotlivé položky - capabilities



# Struktura capability

---

- Struktura capability
  - Typ objektu
  - Práva – obvykle bitová mapa popisující dovolené operace nad objektem
  - Odkaz na objekt, např. číslo uzlu, segmentu, atd..

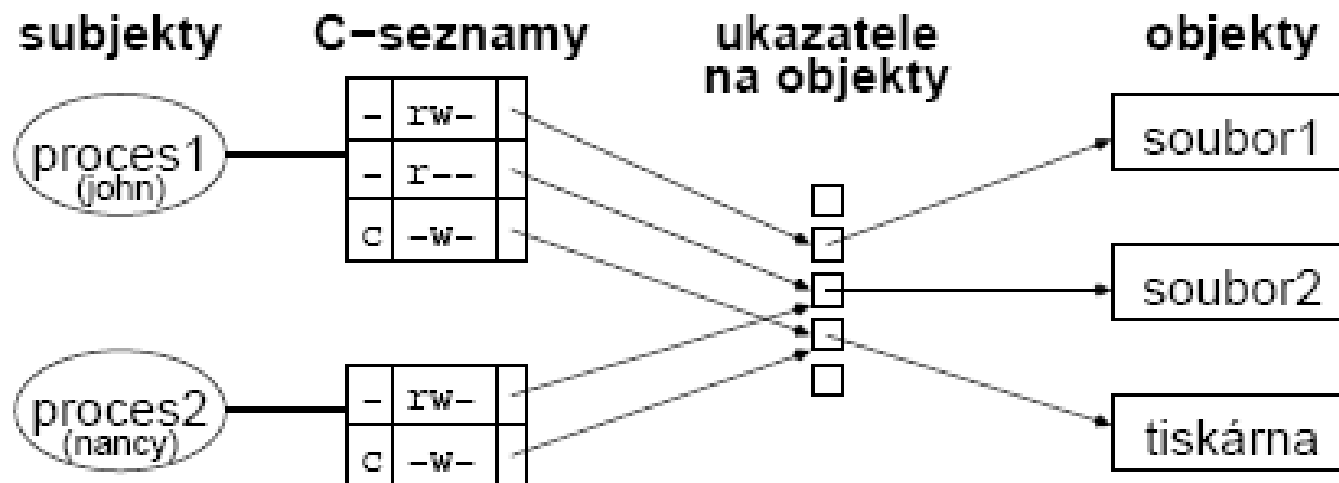
# Capability

---

- Problém – zjištění, kdo všechno má k objektu přístup
- **Zrušení přístupu velmi obtížné** – najít pro objekt všechny capability + odejmout práva
- Řešení: odkaz neukazuje na objekt, ale na **nepřímý objekt** systém může zrušit nepřímý objekt, tím zneplatní odkazy na objekt ze všech C-seznamů

# Capability list

---



# Capability list

---

Pokud jsou jediný způsob odkazu na objekt (bezpečný ukazatel, capability-based addressing):

Ruší rozdíl mezi objekty na disku, v paměti (segmenty) nebo na jiném stroji (objekty by šlo přesouvat za běhu)

Mechanismus C-seznamů v některých distribuovaných systémech (Hydra, Mach,...)

# Přístupová práva

---

## FAT – žádná

- Jen atributy typu read-only, archive, ...

## ext2

- klasická unixová práva (není to ACL)
- vlastník, skupina, ostatní (r,w,x,s,...)
- lze přidat ACL

## NTFS

- ACL
- lze měnit přes grafické UI, příkaz icacs, ...
- explicitně udělit / odepřít práva
- zdědit práva, zakázat dědění

# Proč je tolik filesystemů?

---

- různé fyzické vlastnosti úložišť
  - Ext3 – magnetické disky
  - JFFS2 – flash paměťová zařízení
  - ISO9660 – DVD, CDROM
- Různé kapacity
  - FAT16 – disky do 2GB
  - FAT32 – vhodná pro disky do 32GB
  - Btrfs – multi TB disková pole
- Různé požadavky
  - FAT16 není vhodná pro moderní PC, ale hodí se pro embedded zařízení
- Otevřenost standardů
  - NTFS – uzavřená specifikace

# Zálohování - motivace

---

Uvědomit si rozdíl mezi zálohováním a bezpečným úložištěm !

Můžeme mít poměrně bezpečné úložiště RAID 6 nebo RAID 1, ale pokud si uživatel omylem smaže nějaký soubor, tak mu RAID nepomůže

# Recycle bin, shadow copies

---

Některé systémy se snaží předcházet situaci, kdy si uživatel omylem smaže data.

**Recycle bin** – možnost obnovy z koše

**Shadow copies** – vrátit se k původní verzi souboru



# Zálohování

---

Na externí médium (externí disk, DVD)

I po síti – cloud, síťová složka

Pozor na **ransomware** viry

- Šifrují všechny dostupné síťové jednotky (lokální, síťové)
- Chtějí výpalné za odšifrování souborů

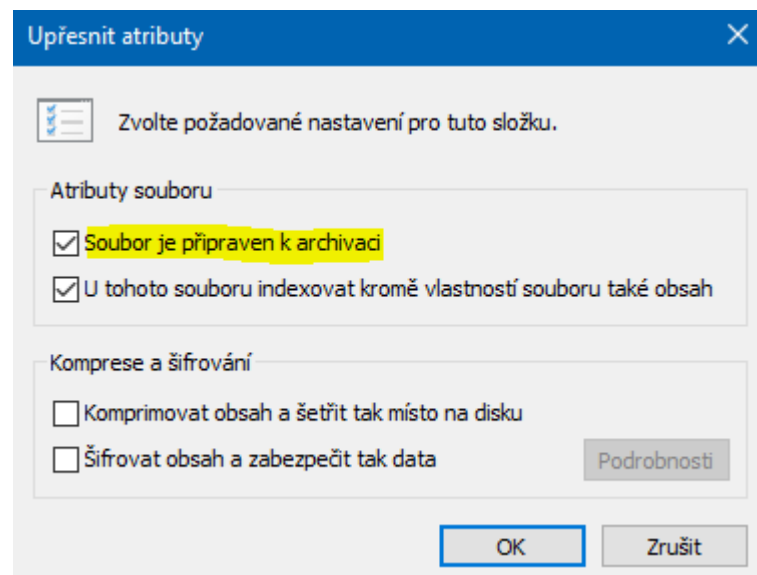
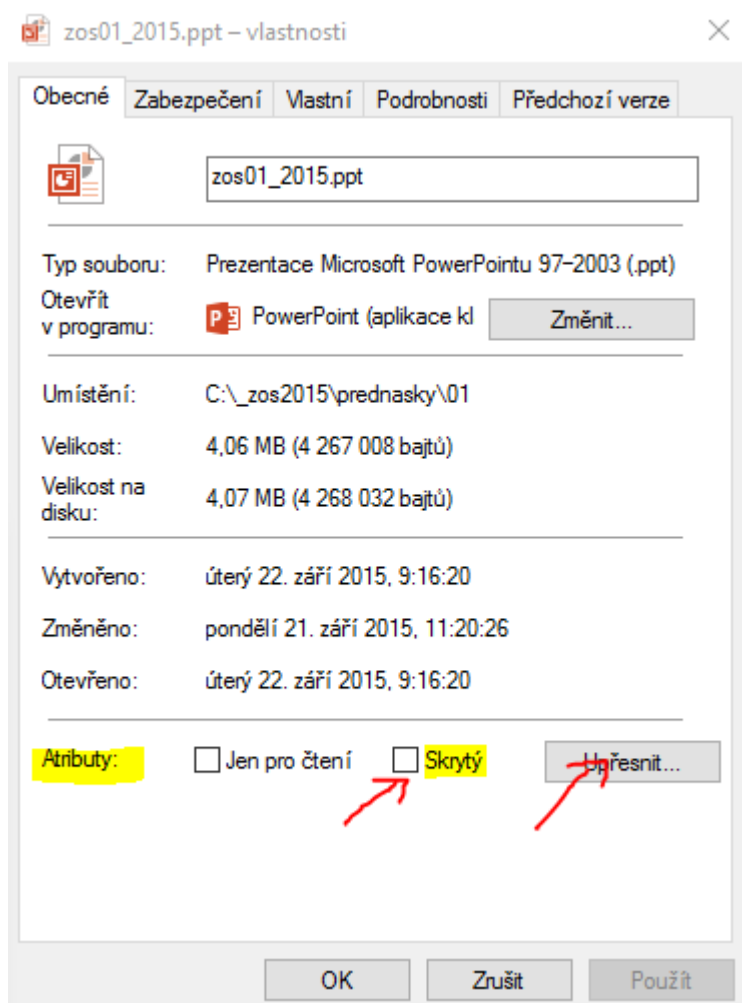
# Zálohování

---

Důležitou otázkou je: **Co zálohovat?**

Atribut **Archive**

- Nové soubory jej mají nastavený
- Při změně stávajícího souboru se také nastaví
- Při zálohování se atribut vynuluje
- Umožní dělat incrementální zálohy



# Typy záloh



manipulací s  
atributech  
archive

## ■ normální

- zálohuje, označí soubory jako "zazálohované"

## ■ copy

- zálohuje, ale neoznačí jako "zazálohované",
- nenaruší používané schéma zálohování

## ■ incremental

- zálohuje pouze vybrané soubory, tj. pokud nebyly "zazálohované" nebo byli změněny a označí je jako "zazálohované"

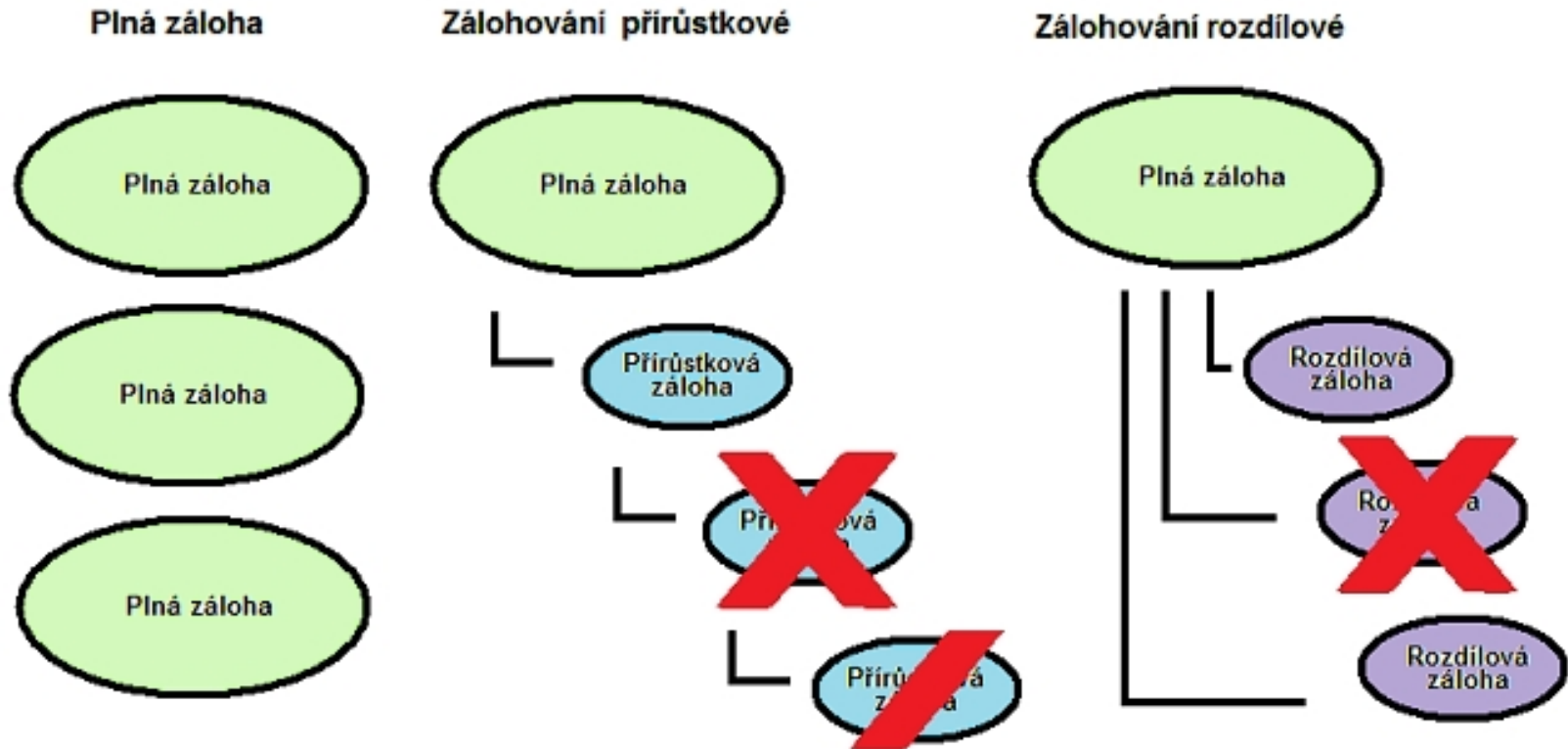
## ■ differential

- viz předchozí, ale neoznačuje jako "zazálohované"
- změny, které proběhly od plné zálohy
- diferenciální zálohy nejsou na sobě závislé

## ■ daily

- zálohuje soubory změněné dnes, ale neoznačuje je jako "zazálohované"

# Typy záloh



zdroj a dobrý materiál k přečtení:  
<http://www.acronis.cz/kb/diferencialni-zaloha/>

# Co se děje při spuštění PC?

---

01. Pustíme proud do počítače 😊

02. **Power on self-test** (řízen BIOSem)

test operační paměti, grafické karty, procesoru

test pevných disků, dalších ATA/SATA/USB zařízení

03. spustí z ROM paměti BIOSu **bootstrap loader**

prohledá boot sektor bootovacího zařízení (dle CMOS)

boot sektor - první sektor bootovacího zařízení (dříve první sektor disku)  
loader)

[http://cs.wikipedia.org/wiki/Power\\_On\\_Self\\_Test](http://cs.wikipedia.org/wiki/Power_On_Self_Test)

# Co se děje při spuštění PC?

---

04. pustí se **zavaděč** (GRUB2, LILO, ...)

může se skládat z více stupňů (stage), v boot sektoru je stage1  
možnost zvolit si jaký systém nabootuje (Linux, Windows)

05. zavaděč nahraje **jádro do paměti** a spustí ho

jádro píše na obrazovku info zprávy  
- můžeme prozkoumat příkazem **dmesg**

06. první proces **init**

/sbin/init , načte /etc/inittab , spouštění a vypínání služeb  
/etc/rc.d/rcX.d

# Co se děje při spuštění PC?

---

07. spustí program **getty** na virtuálních terminálech

zadáme uživatelské jméno

08. spustí se **login**

vyžádá si heslo, zkontroluje v /etc/passwd, /etc/shadow či jinde