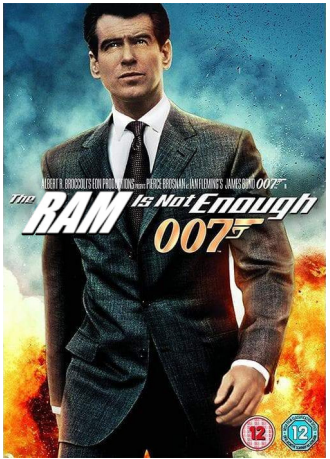
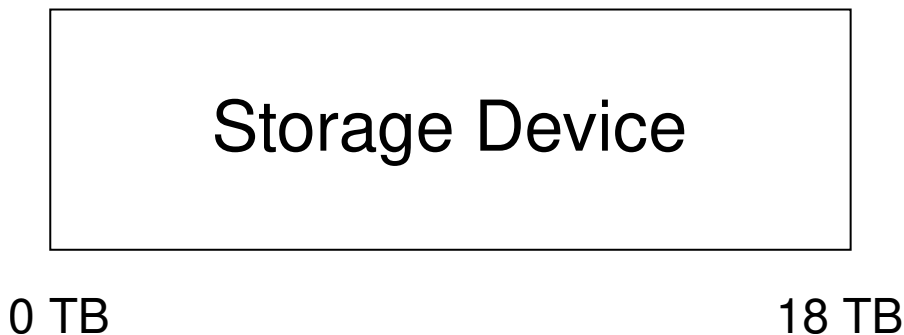
Operating Systems

File Systems

Daniel Gruss

2022-11-10


RAM
IS NOT
Enough
007


Storage Device
0 TB
18 TB


Storage Device
How to organize this?
0 TB
18 TB

Users view

- User does not want to see, know and understand
  - ➤ where and
  - ? how
- data is stored
- must be able to refer to data

→ we need names

Bild	5117 Produkte	Bewertung (Anzahl)	Testberichte	Angebote	LZ	Preis* (pro GiB)
	Samsung RDIMM 16GB, DDR3L-1600, CL11-11-11, reg ECC (M393B2G70BH0-YK0) Typ: DDR3L RDIMM 240-Pin, reg ECC • Takt: 1600MHz • Module: 1x 16GB • JEDEC: PC3L-12800R • Ranks/Bänke: dual rank, x4 • CAS Latency CL: 11 (entspricht ~13.75ns) • Row-to-Column Delay tRCD: 11 (entspricht ~13.75ns) • Row Precharge Time tRP: 11 ...	(zu wenige)		5		ab € 29,90 (€ 1,869/GB)
	Samsung RDIMM 16GB, DDR3L-1600, CL11-11-11, reg ECC (M393B2G70DB0-YK0) Typ: DDR3L RDIMM 240-Pin, reg ECC • Takt: 1600MHz • Module: 1x 16GB • JEDEC: PC3L-12800R • Ranks/Bänke: dual rank, x4 • CAS Latency CL: 11 (entspricht ~13.75ns) • Row-to-Column Delay tRCD: 11 (entspricht ~13.75ns) • Row Precharge Time tRP: 11 ...	(zu wenige)		36		ab € 35,00 (€ 2,188/GB)
	Samsung LRDIMM 32GB, DDR3-1866, CL13-13-13, ECC (M386B4G70DM0-CMA) Typ: DDR3 LRDIMM 240-Pin, ECC • Takt: 1866MHz • Module: 1x 32GB • JEDEC: PC3-14900L • Ranks/Bänke: quad rank, x4 • CAS Latency CL: 13 (entspricht ~13.93ns) • Row-to-Column Delay tRCD: 13 (entspricht ~13.93ns) • Row Precharge Time tRP: 13 ...	(zu wenige)		6		ab € 74,25 (€ 2,320/GB)
	Samsung RDIMM 32GB, DDR4-2133, CL15-15-15, reg ECC (M393A4K40BB0-CPB) Typ: DDR4 RDIMM 288-Pin, reg ECC • Takt: 2133MHz • Module: 1x 32GB • JEDEC: PC4-17000R • Ranks/Bänke: dual rank, x4 • CAS Latency CL: 15 (entspricht ~14.06ns) • Row-to-Column Delay tRCD: 15 (entspricht ~14.06ns) • Row Precharge Time tRP: 15 ...	(zu wenige)		12		ab € 79,00 (€ 2,469/GB)
	Patriot Signature Line DIMM 8GB, DDR4-2666, CL19-19-19-43 (PSD48G266681) Typ: DDR4 DIMM 288-Pin • Takt: 2666MHz • Module: 1x 8GB • JEDEC: PC4-21300U • Ranks/Bänke: single rank • CAS Latency CL: 19 (entspricht ~14.25ns) • Row-to-Column Delay tRCD: 19 (entspricht ~14.25ns) • Row Precharge Time tRP: 19 (entspricht ...	(zu wenige)		26		ab € 19,90 (€ 2,487/GB)
	Patriot Signature Line ohne Kühler DIMM 8GB, DDR3-1600, CL11 (PSD38G16002) Typ: DDR3 DIMM 240-Pin • Takt: 1600MHz • Module: 1x 8GB • JEDEC: PC3-12800U • CAS Latency CL: 11 (entspricht ~13.75ns) • Spannung: 1.5V • Modulhöhe: 30mm • Gehäuse: N/A • Beleuchtung: N/A • Besonderheiten: Standard-SPD • Garantie: (bitte ...	(zu wenige)		15		ab € 19,99 (€ 2,499/GB)
	Patriot Signature Line SO-DIMM 8GB, DDR3L-1600, CL11 (PSD38G1600L2S) Typ: DDR3L SO-DIMM 204-Pin • Takt: 1600MHz • Module: 1x 8GB • JEDEC: PC3L-12800S • CAS Latency CL: 11 (entspricht ~13.75ns) • Spannung: 1.35V • Modulhöhe: 30mm • Gehäuse: N/A • Beleuchtung: N/A • Besonderheiten: Standard-SPD • Garantie: (bitte ...	(zu wenige)		16		ab € 19,99 (€ 2,499/GB)
	Samsung RDIMM 8GB, DDR3L-1333, CL9-9-9, reg ECC (M393B1K70DH0-YH9) Typ: DDR3L RDIMM 240-Pin, reg ECC • Takt: 1333MHz • Module: 1x 8GB • JEDEC: PC3L-10667R • Ranks/Bänke: dual rank • CAS Latency CL: 9 (entspricht ~13.50ns) • Row-to-Column Delay tRCD: 9 (entspricht ~13.50ns) • Row Precharge Time tRP: 9 (entspricht ...	(zu wenige)		13		ab € 20,00 (€ 2,500/GB)

Bild	2700 Produkte	Bewertung (Anzahl)	Testberichte	Angebote	LZ	Preis* (pro TB)
	Patriot Burst Elite 1.92TB, SATA (PBE192TS25SSDR) Bauform: Solid State Drive (SSD) • Formfaktor: 2.5" • Schnittstelle: SATA 6Gb/s • Lesen: 450MB/s • Schreiben: 320MB/s • IOPS 4K lesen/schreiben: 40k/40k • Speichermodule: 3D-NAND QLC • TBW: 800TB • Zuverlässigkeitsprognose: 2 Mio. Stunden (MTBF) ...	(zu wenige)	1 Testbericht	11		ab € 103,90 (€ 54,115/TB)
	Intenso PCIe PREMIUM SSD 1TB, M.2 (3835460) Bauform: Solid State Module (SSM) • Formfaktor: M.2 2280 • Schnittstelle: M.2/M-Key (PCIe 3.0 x4) • Lesen: 2100MB/s • Schreiben: 1700MB/s • Speichermodule: 3D-NAND TLC • TBW: 600TB • Protokoll: NVMe 1.3 • Leistungsaufnahme: keine Angabe ...	(zu wenige)		40		ab € 55,99 (€ 55,990/TB)
	Intenso Top Performance SSD 2TB, SATA (3812470) Bauform: Solid State Drive (SSD) • Formfaktor: 2.5" • Schnittstelle: SATA 6Gb/s • Lesen: 520MB/s • Schreiben: 500MB/s • Protokoll: AHCI • Leistungsaufnahme: keine Angabe (maximal), keine Angabe (Betrieb), keine Angabe (Leerlauf), keine Angabe ...	(zu wenige)		41		ab € 117,03 (€ 58,515/TB)
	Patriot P210 1TB, SATA (P210S1TB25) Bauform: Solid State Drive (SSD) • Formfaktor: 2.5" • Schnittstelle: SATA 6Gb/s • Lesen: 520MB/s • Schreiben: 430MB/s • IOPS 4K lesen/schreiben: 50k/50k • Speichermodule: 3D-NAND (verschiedene Bestückungen möglich) • TBW: keine Angabe ...	(zu wenige)	1 Testbericht	35		ab € 58,89 (€ 58,890/TB)
	Intenso Top Performance SSD 1TB, SATA (3812460) Bauform: Solid State Drive (SSD) • Formfaktor: 2.5" • Schnittstelle: SATA 6Gb/s • Lesen: 520MB/s • Schreiben: 500MB/s • Protokoll: AHCI • Leistungsaufnahme: keine Angabe (maximal), keine Angabe (Betrieb), keine Angabe (Leerlauf), keine Angabe ...	★★★★★ 1 Bewertung		95		ab € 58,90 (€ 58,900/TB)
	TeamGroup CX2 SSD 2TB, SATA (T253X6002T0C101) Bauform: Solid State Drive (SSD) • Formfaktor: 2.5" • Schnittstelle: SATA 6Gb/s • Lesen: 540MB/s • Schreiben: 490MB/s SLC-Cached • Speichermodule: 3D-NAND TLC, Toshiba/WD, 64 Layer (BiCS3) • TBW: 1.6PB • Zuverlässigkeitsprognose: 1 Mio. Stunden ...	(zu wenige)	1 Testbericht	24		ab € 119,70 (€ 59,850/TB)
	Intenso Top Performance SSD 1TB, M.2 (3832460) Bauform: Solid State Module (SSM) • Formfaktor: M.2 2280 • Schnittstelle: M.2/B-M-Key (SATA 6Gb/s) • Lesen: 520MB/s • Schreiben: 500MB/s • Protokoll: AHCI • Leistungsaufnahme: keine Angabe (maximal), keine Angabe (Betrieb), keine Angabe	★★★★★ 2 Bewertungen		57		ab € 59,90 (€ 59,900/TB)

Bild	1468 Produkte	Bewertung (Anzahl)	Testberichte	Angebote	LZ	Preis* (pro TB)
	Toshiba Enterprise Capacity MG08ACA 16TB, 512e, SATA 6Gb/s (MG08ACA16TE) Formfaktor: 3.5", 26.1mm • Drehzahl: 7200rpm • Cache: 512MB • Leistungsaufnahme: 7.63W (Betrieb), 4W (Leerlauf) • Lautstärke: keine Angabe (Betrieb), 20dB(A) (Leerlauf) • Aufnahmeverfahren: Conventional Magnetic Recording (CMR), Two Dimensional ...	★★★★★ 45 Bewertungen	80 aus 1 Test	90		ab € 259,79 (€ 16,237/TB)
	Toshiba Enterprise Capacity MG09ACA 18TB, 512e, SATA 6Gb/s (MG09ACA18TE) Formfaktor: 3.5", 26.1mm • Drehzahl: 7200rpm • Cache: 512MB • Leistungsaufnahme: 8.35W (Betrieb), 4.16W (Leerlauf) • Lautstärke: keine Angabe (Betrieb), 20dB(A) (Leerlauf) • Aufnahmeverfahren: Flux Control Microwave Assisted Conventional Magnetic ...	★★★★★ 34 Bewertungen	2 Testberichte	81		ab € 295,89 (€ 16,438/TB)
	Toshiba Enterprise Capacity MG09ACA 18TB, 4Kn, SATA 6Gb/s (MG09ACA18TA) Formfaktor: 3.5", 26.1mm • Drehzahl: 7200rpm • Cache: 512MB • Leistungsaufnahme: 8.35W (Betrieb), 4.16W (Leerlauf) • Lautstärke: keine Angabe (Betrieb), 20dB(A) (Leerlauf) • Aufnahmeverfahren: Flux Control Microwave Assisted Conventional Magnetic ...	(zu wenige)	2 Testberichte	2		ab € 296,79 (€ 16,488/TB)
	Seagate Exos X - X18 18TB, 512e/4Kn, SATA 6Gb/s (ST18000NM000J) Formfaktor: 3.5" • Drehzahl: 7200rpm • Cache: 256MB • Leistungsaufnahme: 6.4W (Betrieb), 5.3W (Leerlauf) • Lautstärke: keine Angabe (Betrieb), keine Angabe (Leerlauf) • Aufnahmeverfahren: Conventional Magnetic Recording (CMR) • Sektoren: 4KB mit ...	★★★★★ 25 Bewertungen	1 Testbericht	114		ab € 299,98 (€ 16,666/TB)
	Toshiba Enterprise Capacity MG07ACA 14TB, 512e, SATA 6Gb/s (MG07ACA14TE) Formfaktor: 3.5", 26.1mm • Drehzahl: 7200rpm • Cache: 256MB • Leistungsaufnahme: 7.8W (Betrieb), 4.22W (Leerlauf) • Lautstärke: keine Angabe (Betrieb), 20dB(A) (Leerlauf) • Aufnahmeverfahren: Conventional Magnetic Recording (CMR) • Sektoren: 4KB ...	★★★★★ 19 Bewertungen	2 Testberichte	103		ab € 244,29 (€ 17,449/TB)
	Seagate SkyHawk +Rescue 4TB, SATA 6Gb/s (ST4000VX007) Formfaktor: 3.5" • Drehzahl: 5900rpm • Cache: 64MB • Leistungsaufnahme: 5.5W (Betrieb), 3.2W (Leerlauf) • Lautstärke: 34dB(A) (Betrieb), 30dB(A) (Leerlauf) • Aufnahmeverfahren: Conventional Magnetic Recording (CMR) • Sektoren: 4KB mit Emulation ...	★★★★★ 18 Bewertungen		106		ab € 69,86 (€ 17,465/TB)
	Seagate Exos X - X16 16TB, 512e/4Kn, SATA 6Gb/s (ST16000NM001G) Formfaktor: 3.5" • Drehzahl: 7200rpm • Cache: 256MB • Leistungsaufnahme: 6.3W (Betrieb), 5.0W (Leerlauf) • Lautstärke: keine Angabe (Betrieb), keine Angabe (Leerlauf) • Aufnahmeverfahren: Conventional Magnetic Recording (CMR) • Sektoren: 4KB mit ...	★★★★★ 21 Bewertungen	80 aus 1 Test	118		ab € 284,94 (€ 17,809/TB)
	Seagate Exos X - X16 14TB, 512e/4Kn, SATA 6Gb/s (ST14000NM001G) Formfaktor: 3.5" • Drehzahl: 7200rpm • Cache: 256MB • Leistungsaufnahme: 6.3W (Betrieb), 5.0W (Leerlauf) • Lautstärke: keine Angabe (Betrieb), keine Angabe (Leerlauf) • Aufnahmeverfahren: Conventional Magnetic Recording (CMR) • Sektoren: 4KB mit ...	★★★★★ 9 Bewertungen	80 aus 1 Test	93		ab € 249,48 (€ 17,820/TB)

Illusions

- **persistent** storage, although physical corruption happens all the time
- easy/fast (byte-addressable) **random** accesses, although built for sequential accesses (in blocks)
- almost **endless** capacity (for files, for data within a file), despite very real limitations
- **fast**, but actually slow
- **names** for files and directories, but actually just bits and bytes

Performance

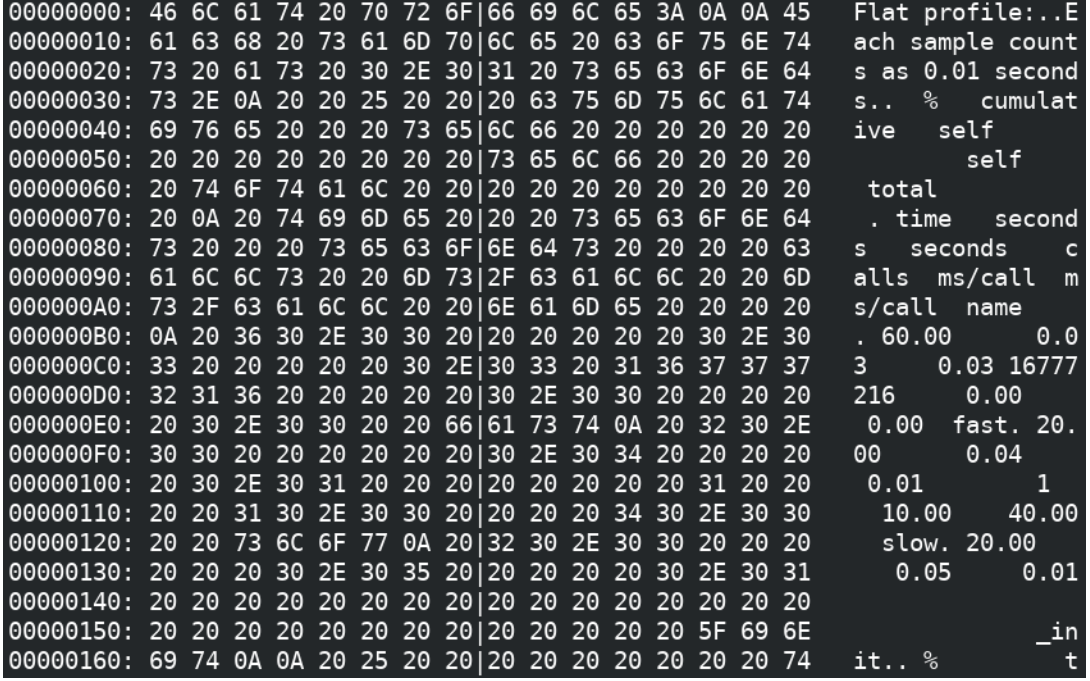
What to do when performance is bad? Caches!

- DRAM cache inside modern storage devices
- Page cache in software, in the OS

Yet another cache: the page cache

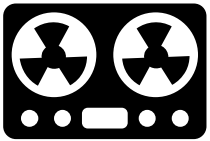
- Files buffered page-wise in “page cache”
- Lower access time for frequently accessed data
- Use up all the memory
  - Pages are freed on demand
- Deduplicate pages (copy-on-write)

```
00000000: 25 50 44 46 2D 31 2E 35|0A 25 B5 ED AE FB 0A 31   %PDF-1.5.%.....1
00000010: 32 20 30 20 6F 62 6A 0A|3C 3C 20 2F 4C 65 6E 67   2 0 obj.<< /Leng
00000020: 74 68 20 31 33 20 30 20|52 0A 20 20 20 2F 46 69   th 13 0 R.   /Fi
00000030: 6C 74 65 72 20 2F 46 6C|61 74 65 44 65 63 6F 64   lter /FlateDecod
00000040: 65 0A 3E 3E 0A 73 74 72|65 61 6D 0A 78 9C ED 5D   e.>>.stream.x..]
00000050: 6D 8F 1C B7 91 FE BE BF|82 30 2E C0 2C A0 ED E5   m........0..,...
00000060: 3B D9 BE 20 86 63 FB 9C|1C 6C 24 B6 94 04 38 6D   ;.. .c...l$...8m
00000070: 10 8C 76 7B 76 27 1A CD|AC 66 66 25 CB BF FE 9E   ..v{v'...ff%....
00000080: 62 BF 4C 77 4F 93 D3 BD|A3 03 0E 97 93 EC D5 BC   b.LwO...........
00000090: B0 8A C5 AA 62 D5 53 24|9B 2B 18 C7 DF 2B 81 1F   ....b.S$.+...+..
000000A0: 5E 4B 76 FB EE E2 FD C5|4F EC FD 85 B2 0C FF 19   ^Kv.....O.......
000000B0: A9 B2 3C 97 CC 39 CE B6|05 FB 1B 5B 5F 08 46 7F   ..<..9.....[_.F
000000C0: B7 F7 EC 7A CE D9 FD 6E|B8 E1 A2 FA 58 78 97 79   ...z...n....Xx.y
000000D0: AF 98 10 99 54 BA FA 26|E7 99 E0 86 59 9E 49 EE   ....T..&....Y.I.
000000E0: 98 96 19 CF 35 E3 99 B2|AE DB C0 49 B0 B4 43 0D   ....5......I..C.
000000F0: 1C 3E F1 4C 48 93 71 2B|D1 B7 C8 3C 17 4C F2 4C   .>.LH.q+...<.L.L
00000100: 5B D1 6D A2 6D 66 A5 68|9A 08 95 09 23 3B 4D A4   [.m.mf.h....#;M.
00000110: E0 99 E5 72 B0 89 16 99|06 13 2E 33 0D 1E 3C CF   ...r.......3..<.
00000120: B4 73 9D 6E 0C 68 72 83|16 3E D3 CA 33 63 32 6B   .s.n.hr..>..3c2k
00000130: 49 B0 CC F9 BC 6C 81 C6|DE 4A 67 D8 F1 0B E8 50   I....l...Jg....P
00000140: F9 CC 41 EB 52 E6 41 3F|46 E0 5F BC 6F 8F F5 D0   ..A.R.A?F._.o...
00000150: 82 A4 2C BF E1 59 2E 74|CD DF 29 FA C3 8E 5F 80   ..,..Y.t..)..._.
00000160: BF 31 2E B3 C2 35 1D 1C|91 D7 EC 15 54 8E A1 0E   .1...5......T...
00000170: 08 D0 D8 DB 9A CC 98 1C|4D 4D 69 96 5C 64 36 C7   ........MMi.\d6.
```


00000000: 46 6C 61 74 20 70 72 6F|66 69 6C 65 3A 0A 0A 45 Flat profile:..E
00000010: 61 63 68 20 73 61 6D 70|6C 65 20 63 6F 75 6E 74 ach sample count
00000020: 73 20 61 73 20 30 2E 30|31 20 73 65 63 6F 6E 64 s as 0.01 second
00000030: 73 2E 0A 20 20 25 20 20|20 63 75 6D 75 6C 61 74 s.. %   cumulat
00000040: 69 76 65 20 20 20 73 65|6C 66 20 20 20 20 20 20 ive   self
00000050: 20 20 20 20 20 20 20 20|73 65 6C 66 20 20 20 20   self
00000060: 20 74 6F 74 61 6C 20 20|20 20 20 20 20 20 20 20   total
00000070: 20 0A 20 74 69 6D 65 20|20 20 73 65 63 6F 6E 64   . time   second
00000080: 73 20 20 20 73 65 63 6F|6E 64 73 20 20 20 20 63 s   seconds   c
00000090: 61 6C 6C 73 20 20 6D 73|2F 63 61 6C 6C 20 20 6D alls ms/call m
000000A0: 73 2F 63 61 6C 6C 20 20|6E 61 6D 65 20 20 20 20 s/call name
000000B0: 0A 20 36 30 2E 30 30 20|20 20 20 20 20 30 2E 30 . 60.00   0.0
000000C0: 33 20 20 20 20 20 30 2E|30 33 20 31 36 37 37 37 3   0.03 16777
000000D0: 32 31 36 20 20 20 20 20|30 2E 30 30 20 20 20 20 216   0.00
000000E0: 20 30 2E 30 30 20 20 66|61 73 74 0A 20 32 30 2E   0.00 fast. 20.
000000F0: 30 30 20 20 20 20 20 20|30 2E 30 34 20 20 20 20 00   0.04
00000100: 20 30 2E 30 31 20 20 20|20 20 20 20 31 20 20 20   0.01   1
00000110: 20 20 31 30 2E 30 30 20|20 20 20 34 30 2E 30 30   10.00   40.00
00000120: 20 20 73 6C 6F 77 0A 20|32 30 2E 30 30 20 20 20   slow. 20.00
00000130: 20 20 20 30 2E 30 35 20|20 20 20 20 30 2E 30 31   0.05   0.01
00000140: 20 20 20 20 20 20 20 20|20 20 20 20 20 20 20 20
00000150: 20 20 20 20 20 20 20 20|20 20 20 20 20 5F 69 6E   _in
00000160: 69 74 0A 0A 20 25 20 20|20 20 20 20 20 20 20 74 it.. %   t

```
dgruss@hpx360dg / % mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=7954268k,nr_inodes=1988567,mode=755,inode64)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,nodev,noexec,relatime,size=1602616k,mode=755,inode64)
/dev/mapper/ubuntu--vg-root on / type ext4 (rw,noatime,errors=remount-ro)
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev,inode64)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k,inode64)
cgroup2 on /sys/fs/cgroup type cgroup2 (rw,nosuid,nodev,noexec,relatime,nsdelegate,memory_recursiveprot
pstore on /sys/fs/pstore type pstore (rw,nosuid,nodev,noexec,relatime)
efivarfs on /sys/firmware/efi/efivars type efivarfs (rw,nosuid,nodev,noexec,relatime)
bpf on /sys/fs/bpf type bpf (rw,nosuid,nodev,noexec,relatime,mode=700)
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,relatime,fd=30,pgrp=1,timeout=0,minproto=5,maxpro
mqueue on /dev/mqueue type mqueue (rw,nosuid,nodev,noexec,relatime)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime,pagesize=2M)
debugfs on /sys/kernel/debug type debugfs (rw,nosuid,nodev,noexec,relatime)
tracefs on /sys/kernel/tracing type tracefs (rw,nosuid,nodev,noexec,relatime)
fusectl on /sys/fs/fuse/connections type fusectl (rw,nosuid,nodev,noexec,relatime)
configfs on /sys/kernel/config type configfs (rw,nosuid,nodev,noexec,relatime)
none on /run/credentials/systemd-sysusers.service type ramfs (ro,nosuid,nodev,noexec,relatime,mode=700)
tmpfs on /run/qemu type tmpfs (rw,nosuid,nodev,relatime,mode=755,inode64)
```

accessing data



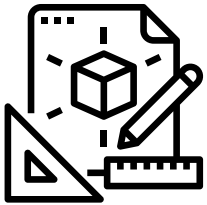
- tape drives, disks, ...
  - sequential access
  - one byte after the other
  - `seek()` other positions to access other parts
  - random access possible, but slow

File System Workload

Optimize for small files or large files?

- number of them?
- space they occupy?
- accesses to them?
- sequential vs. random access?
- size changes over time?

We want the best of both worlds!



- Small files:
  - Small blocks → low internal fragmentation
  - Fast concurrent operations
  - Files used together should be stored together (why?)
- Large files:
  - Large blocks → low external fragmentation
  - Contiguous allocation for fast sequential access
  - Efficient lookup within the file for random access

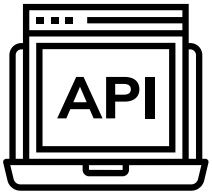
File System Abstraction

- Directory
  - Group of named files or subdirectories → store in a metadata block
  - Mapping from file name to file Metadata location
- >_ Path
  - String that uniquely identifies file or directory
  - `/var/www/teaching/courses/os`

File System Abstraction

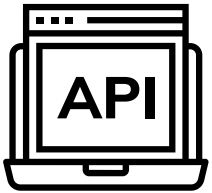
- Links
  - Hard link: name to metadata
  - Soft link: name to name
- Mount
  - Link name in one file system to root of another

Accessing files - API



- creating and deleting files: `create()`, `unlink()`
- linking files (creating a hard link) `link()`
- directory operations: `mkdir()`, `rmdir()`
- open to start accessing a file: `open()` (actually much more than just that)
- close to end accessing the file: `close()`

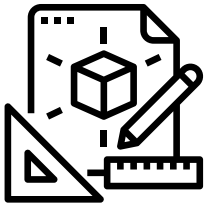
Accessing files - API



kind of like the tape drive model...?

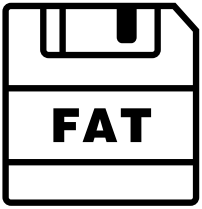
- reading from a file: `read()`
- writing to a file: `write()`
- positioning `seek()`
- force modification to storage: `fsync()`

How to organize the storage?



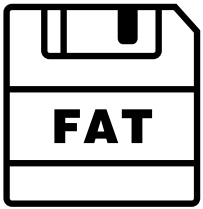
- split storage into blocks
  - what is a good block size?
  - file name $\rightarrow$ meta data + blocks
- how to find data blocks? $\rightarrow$ **file index**
- where are free data blocks on the storage? how to allocate them?
- Locality: blocks/files/folders?
- Reliability: crash during file system operation?

Naïve Approach: FAT - File Allocation Table



- old! (1970s)
- file system for MS-DOS and early Windows
- many enhancements
- Today: exFAT for SD-cards, USB sticks, ...

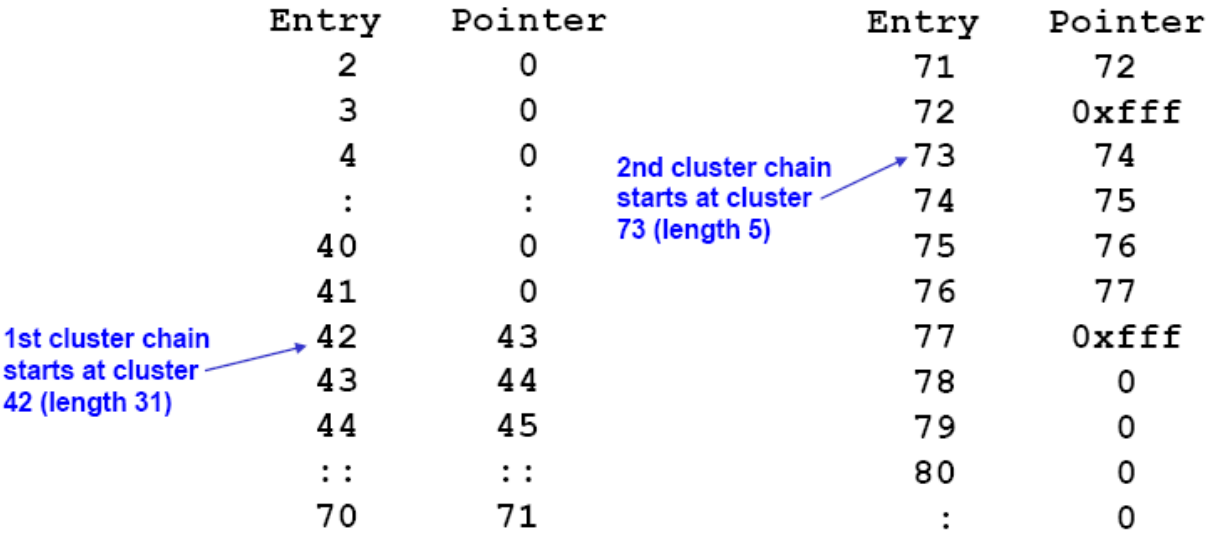
How does FAT work?

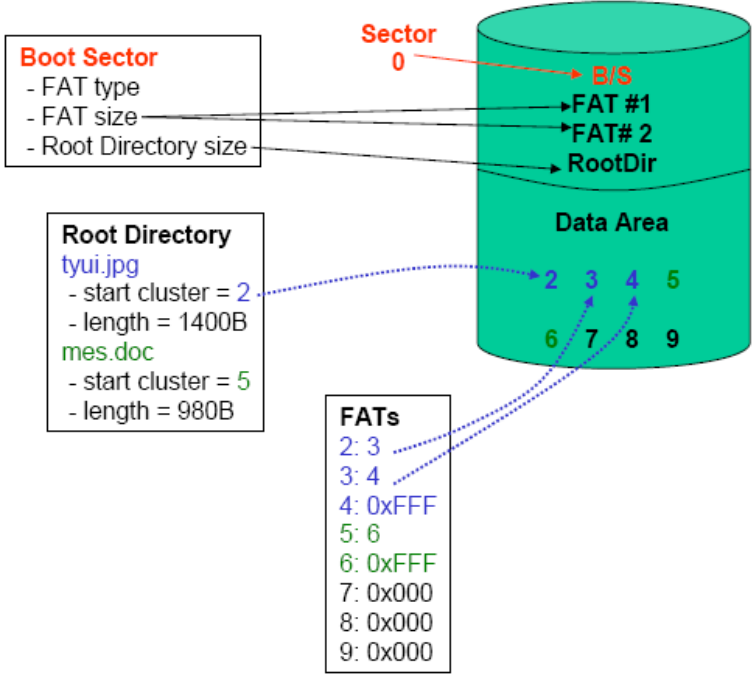


- Blocks? Sectors! Which size? 512 bytes
- Sectors are too small... 4096
- Files: cluster of 2^n sectors ($n = 0 \ldots 6$) – contiguous sectors!!
- FAT: cluster status + pointer to next one (if file is larger than one cluster)
  - Cluster number $\rightarrow$ works exactly like physical page number!
- Directory: file name, starting cluster, length

FAT Limits

- FAT12: 12bit FAT entry $\rightarrow 2^{12}$ clusters (512B-4KB) $\rightarrow$ max. 16 MB
- FAT16: 16bit FAT entry $\rightarrow 2^{16}$ (2KB-32KB) $\rightarrow$ max. 2 GB
- FAT32: 28bit FAT entry $\rightarrow 2^{28}$ (4KB-32KB) $\rightarrow$ max. 2 TB (limited by a 32-bit sector count field)


Entry Pointer
2 0
3 0
4 0
: :
40 0
41 0
42 43
43 44
44 45
:: ::
70 71
1st cluster chain starts at cluster 42 (length 31)
Entry Pointer
71 72
72 0xfff
73 74
74 75
75 76
76 77
77 0xfff
78 0
79 0
80 0
: 0
2nd cluster chain starts at cluster 73 (length 5)


Boot Sector
- FAT type
- FAT size
- Root Directory size
Sector
0
B/S
FAT #1
FAT# 2
RootDir
Data Area
2 3 4 5
6 7 8 9
Root Directory
tyui.jpg
- start cluster = 2
- length = 1400B
mes.doc
- start cluster = 5
- length = 980B
FATs
2: 3
3: 4
4: 0xFFF
5: 6
6: 0xFFF
7: 0x000
8: 0x000
9: 0x000

FAT

- FAT area for table
- Data area for the data of files, in clusters

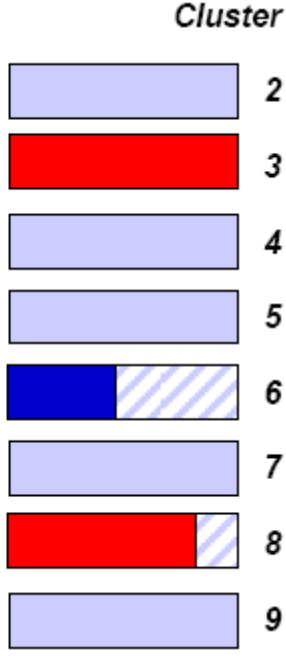
Root Directory

File Name	Size	Cluster
gary.txt	1034	6
hello.jpg	3973	3

File Allocation Table

Cluster	Next
2	0
3	8
4	0
5	0
6	0xFF
7	0
8	0xFF
9	0

Cluster = 2048 B = 4 sectors



Bytes	Purpose
0-2	Assembly code instructions to jump to boo
3-10	OEM name in ASCII
11-12	Bytes per sector (512, 1024, 2048, or 4096
13	Sectors per cluster (Must be a power of 2
14-15	Size of reserved area, in sectors
16	Number of FATs (usually 2)
17-18	Maximum number of files in the root di

Bytes	Purpose
0-35	(See previous table)
36	BIOS INT 13h (low leve
37	Not used
38	Extended boot signature

Sector(s)	Address
0	0x0000-0x
1-9	0x0200-0x

Root Directory

- after FAT(s) - or in FAT32: specified in boot sector
- new file entry needed? first / next-available search

Root Directory SFN Entry Data Structure

Bytes	Purpose
0	First character of file name (ASCII) *or* allocation status (0x00=unallocated, 0xe5=deleted)
1-10	Characters 2-11 of the file name (ASCII); the "." is implied between bytes 7 and 8
11	File attributes (see File Attributes table)
12	Reserved
13	File creation time (in tenths of seconds)*
14-15	Creation time (hours, minutes, seconds)*
16-17	Creation date*
18-19	Access date*
20-21	High-order 2 bytes of address of first cluster (0 for FAT12/16)*
22-23	Modified time (hours, minutes, seconds)
24-25	Modified date
26-27	Low-order 2 bytes of address of first cluster
28-31	File size (0 for directories)

File Attributes

Flag Value	Description
0000 0001 (0x01)	Read-only
0000 0010 (0x02)	Hidden file
0000 0100 (0x04)	System file
0000 1000 (0x08)	Volume label
0000 1111 (0x0f)	Long file name
0001 0000 (0x10)	Directory
0010 0000 (0x20)	Archive

** Bytes 13-22 are unused by DOS*

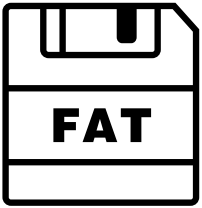
- Root directory (32 bytes each):
- also possible: extra 32 bytes for "long" filename

Allocating Files

1. Find free entry in directory
2. Find free entry in FAT for cluster, write sector number there and EOF into FAT
3. write start sector into directory

Extending files? if next FAT entry is free, move EOF to that instead

FAT discussion



- widely used - simple, wide supporting
- principle of locality?
  - fragmented files
  - iterate through directories and FAT frequently
- Limited metadata and access control
- No hard links
- limitation of volume and file size
- reliability techniques??

UNIX FFS

- Unix Fast File System - released mid 1980
- many data-structures identical to Ritchie/Thomposon's original UNIX FS (1970ies)
- Tree-based multi-level index

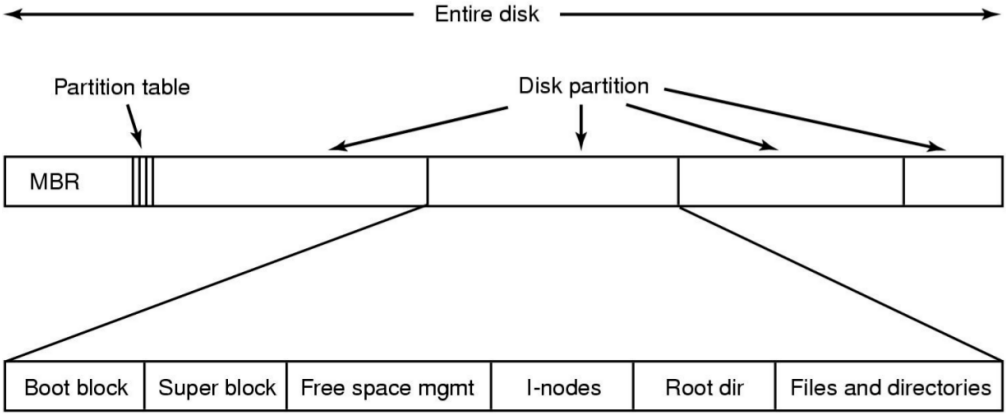
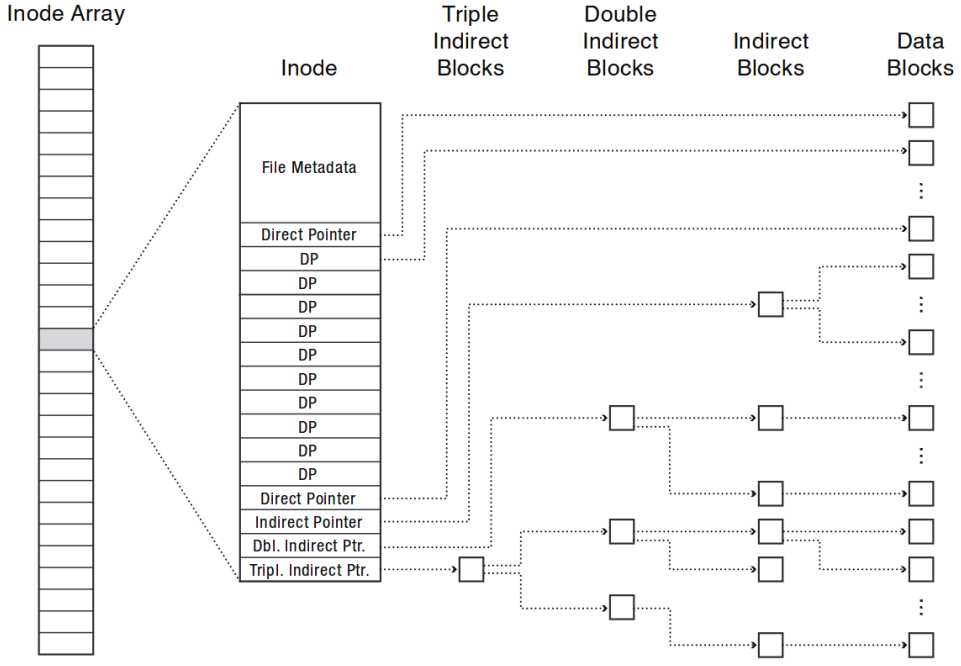
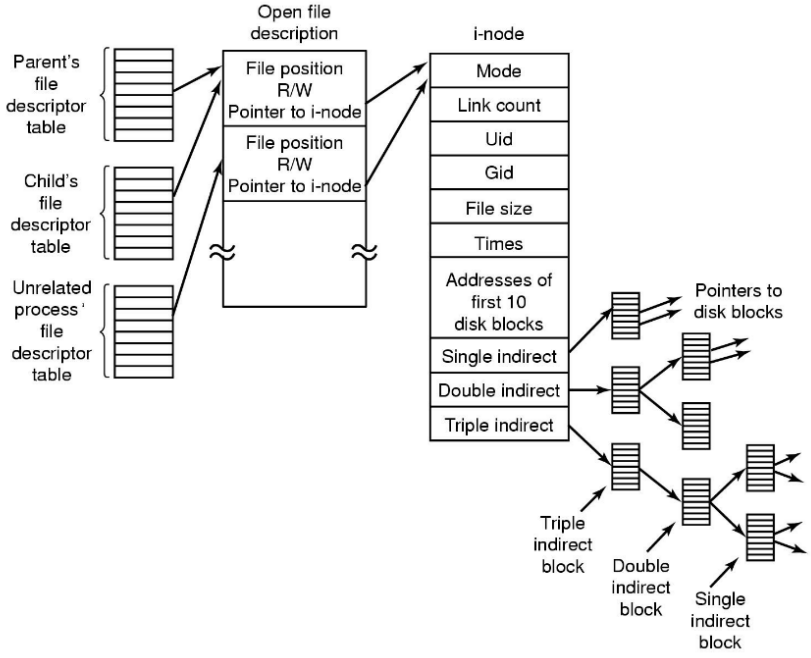


Figure 1: Disk layout, classical example

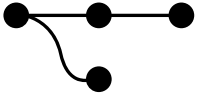
Implementation (“classical Unix”)

- boot block: Boot Loader, to boot system
- super block: Infos on file system, e.g.
  - size of partition, block size, free block list, ...
  - ...
- index nodes (inodes)
  - inodes $\leftrightarrow$ files
- data blocks


Inode Array
Inode
Triple Indirect Blocks
Double Indirect Blocks
Indirect Blocks
Data Blocks
File Metadata
Direct Pointer
DP
DP
DP
DP
DP
DP
DP
DP
DP
DP
Direct Pointer
Indirect Pointer
Dbl. Indirect Ptr.
Tripl. Indirect Ptr.

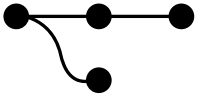

Open file description
i-node
Parent's file descriptor table
Child's file descriptor table
Unrelated process' file descriptor table
File position
R/W
Pointer to i-node
File position
R/W
Pointer to i-node
Mode
Link count
Uid
Gid
File size
Times
Addresses of first 10 disk blocks
Single indirect
Double indirect
Triple indirect
Pointers to disk blocks
Triple indirect block
Double indirect block
Single indirect block

Inode



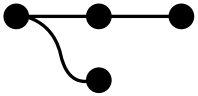
- Attributes:
  - type: file, directory, character special file, block special file
  - owner: user, group
  - created, modified, accessed times
  - size: in bytes and blocks
  - permissions (rwx)
  - NO filename

Inodes



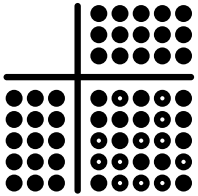
- files may have multiple names
  - directories contain names and numberings
  - multiple occurrences possible
  - “hard link”
  - inode contains link count

Inodes



- Inode link-count 0:
  - no more reference within file system exists
  - file can be deleted
- number of inodes limited
  - file system may be full, because
    - no free inode
    - all blocks used

Sparse files

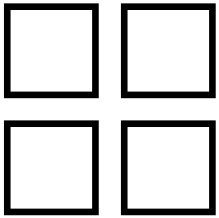


- Sparse file: one or more empty spaces are surrounded by file data
- empty space: needs not consume disk spaces

```
fd=creat("test.file",777);
lseek(fd,1000000000,SEEK_SET);
write(fd,"test",2);
close(fd);
```

- Should create a file of size ~1GB using one block
  - does the file system support it?

Block group placement



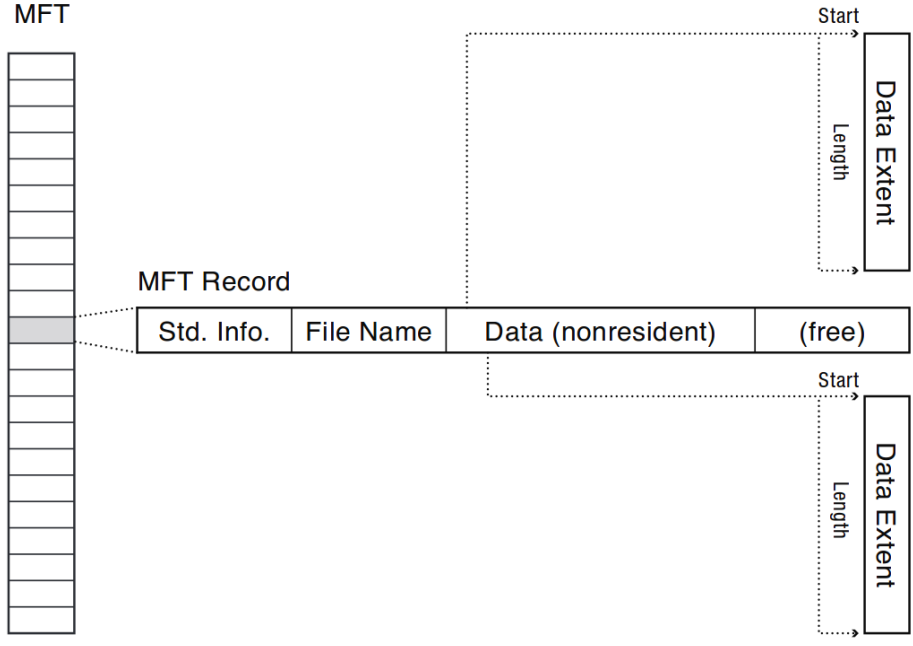
- places data to optimize concurrent access to
  - data blocks of a file
  - metadata of a file
  - different files from the same directory
- different directories may be far from each others

NTFS

- Microsoft **N**ew **T**echnology **F**ile **S**ystem
- released 1993
- many new features compared to FAT
  - new index structures
  - flexible metadata
  - improved security
  - improved reliability
- still the primary file system for Windows

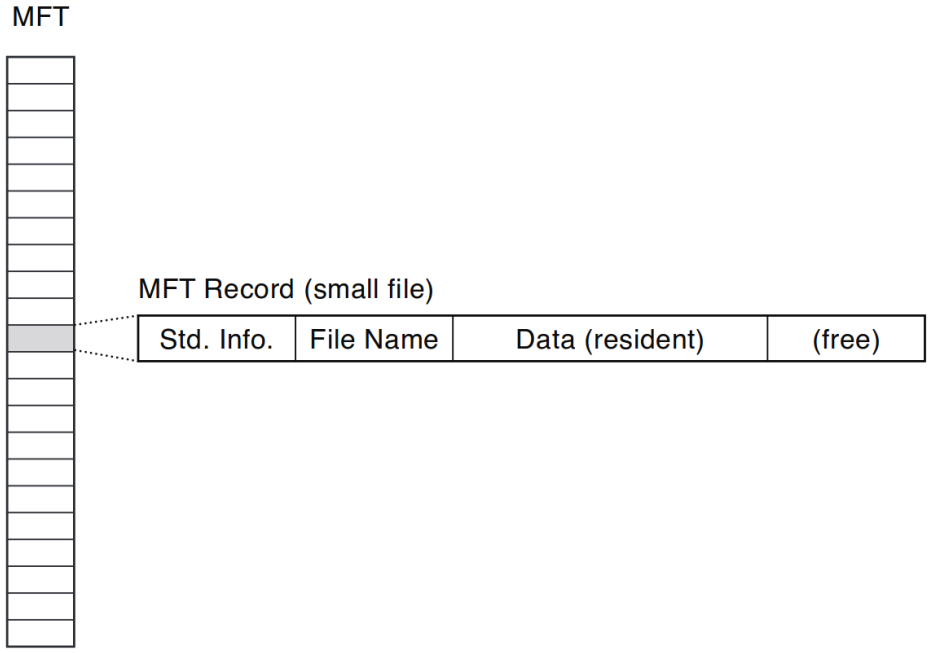
NTFS Index Structures

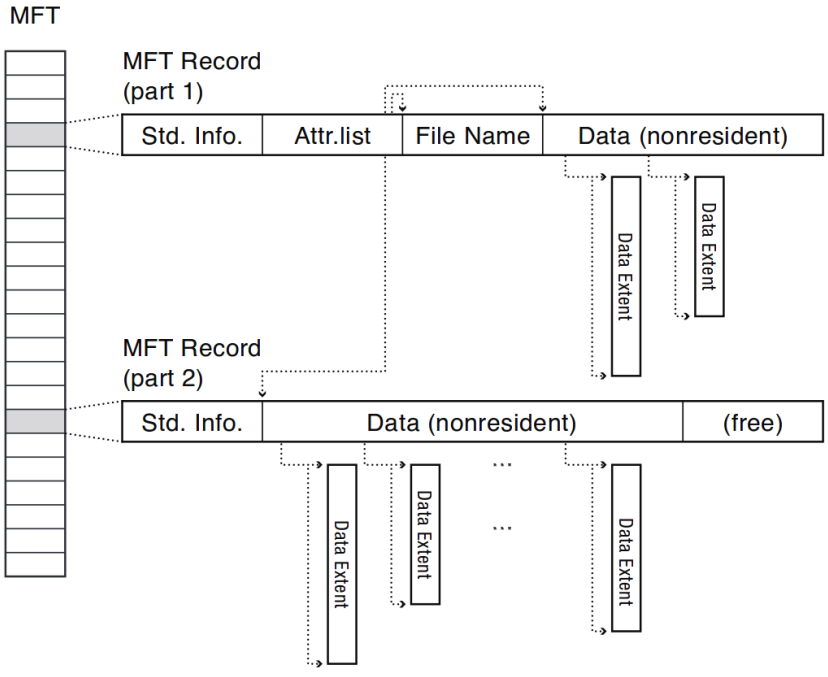
- **Extents**: variable sized region of a file stored in a contiguous region on the storage device
- **Flexible Tree and Master File Table(MFT)**: each file represented by a tree
  - small number of extents: shallow tree
  - badly fragmented file: deeper tree
- Root: stored in a MFT (similar to inode array)
  - array of 1KB MFT records
  - contains sequence of variable-size attribute records
    - can contain data and metadata
    - data is an attribute of a file-system

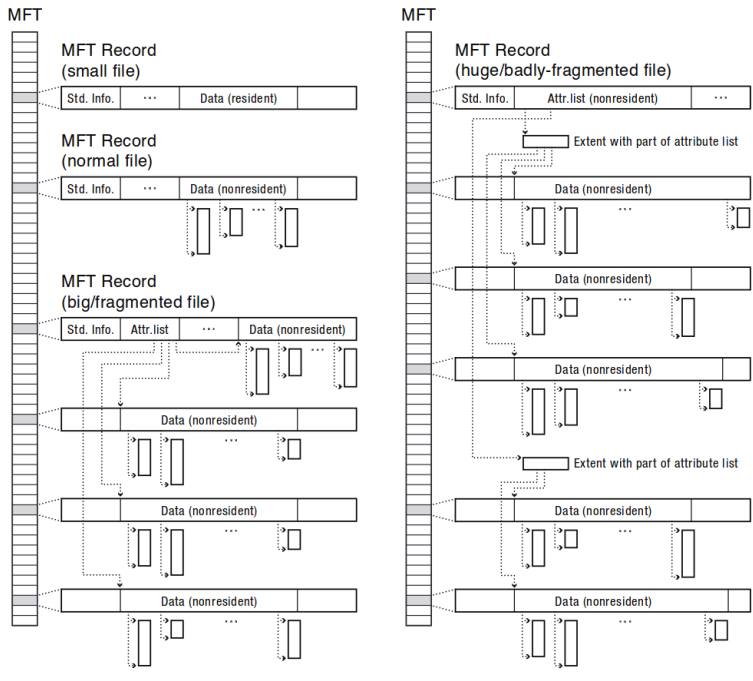

MFT
Start
Data Extent
Length
MFT Record
Std. Info.
File Name
Data (nonresident)
(free)
Start
Data Extent
Length

index for NTFS basic file

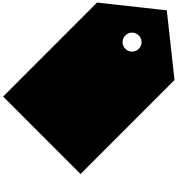
- MFT contains *nonresident data* attribute
  - sequence of extent pointers
  - specify starting block and length of blocks of an extent
  - extent: variable size - can be multiple GBs
- File small? attribute may even contain data


MFT
MFT Record (small file)
Std. Info.
File Name
Data (resident)
(free)


MFT
MFT Record
(part 1)
Std. Info.
Attr.list
File Name
Data (nonresident)
Data Extent
Data Extent
MFT Record
(part 2)
Std. Info.
Data (nonresident)
(free)
Data Extent
Data Extent
Data Extent

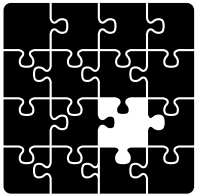

MFT
MFT Record
(small file)
Std. Info.
Data (resident)
MFT Record
(normal file)
Std. Info.
Data (nonresident)
MFT Record
(big/fragmented file)
Std. Info.
Attr.list
Data (nonresident)
Data (nonresident)
Data (nonresident)
Data (nonresident)
MFT
MFT Record
(huge/badly-fragmented file)
Std. Info.
Attr.list (nonresident)
Extent with part of attribute list
Data (nonresident)
Data (nonresident)
Data (nonresident)
Extent with part of attribute list
Data (nonresident)
Data (nonresident)

Metadata files



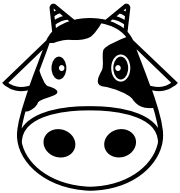
- no special regions for file system metadata
- all metadata in ordinary files:
  - file 5: root directory
  - file 6: free space bitmap
  - file 8: bad block list
  - file 9: security and access control information
  - file 0: master file tableofcontents
    - first sector contains a pointer to first MFT entry
- makes it easier to dynamically grow metadata

NTFS locality heuristics



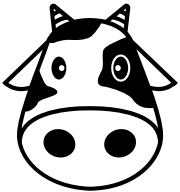
- variation of “best fit” - place a newly allocated file in the smallest free region large enough
- applications can indicate expected file size
- start of volume reserved for MFT table to avoid fragmentation

COW File systems



- COW file systems never overwrite existing data or metadata
  - write new versions to new locations
  - Example - append a block to a file

Why would we need COW file systems?



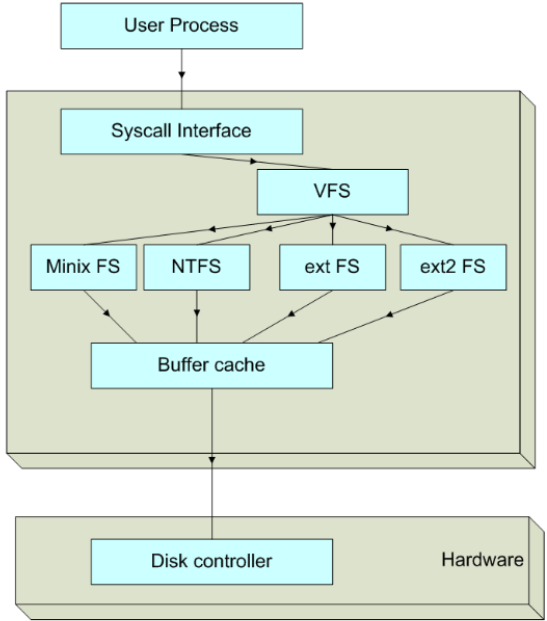
- small writes are expensive
- Caches filter reads
- Flash Storage / SSDs?

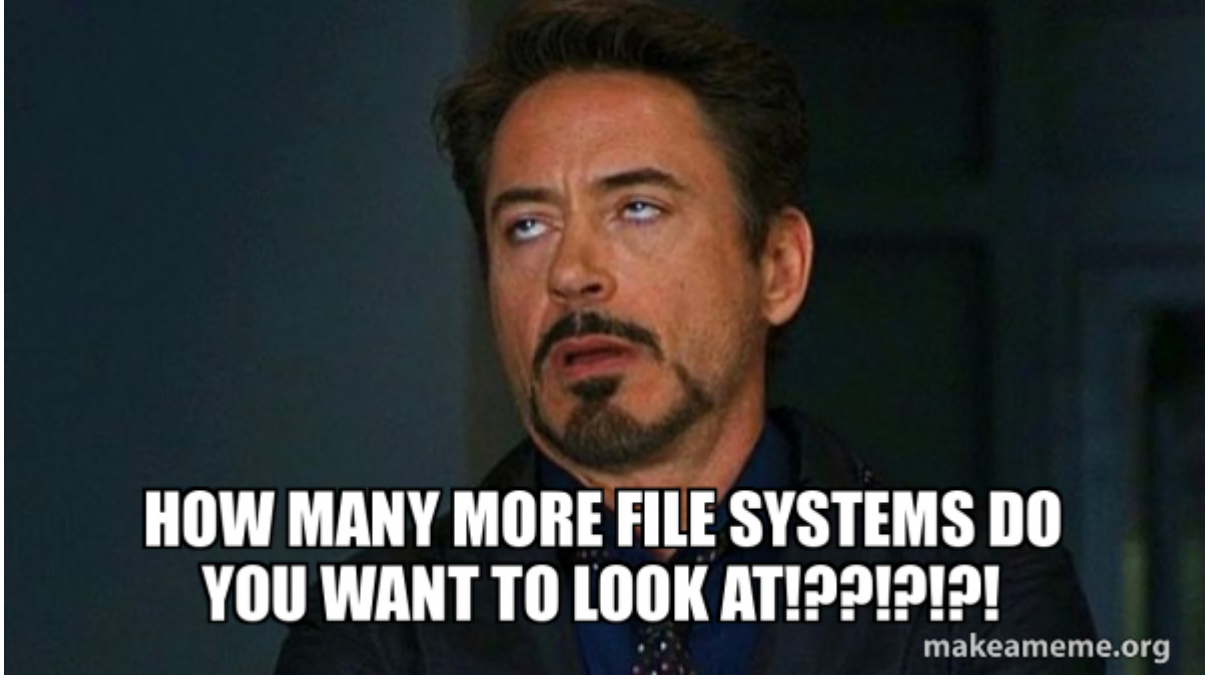
→ move data to new pages

- Versioning

ext/ext2/ext3/ext4

- linux-based file systems
- originally “cross-development” in Minix
- based on the Minix file system
- VFS: virtual file system layer


User Process
Syscall Interface
VFS
Minix FS
NTFS
ext FS
ext2 FS
Buffer cache
Disk controller
Hardware


HOW MANY MORE FILE SYSTEMS DO YOU WANT TO LOOK AT!??!?!?!
makeameme.org

- ext: extended file system (1992)
  - supported VFS
  - 2 GB disk size
  - 255 Byte file names

- designed for extensibility
- used until Kernel 2.6.17 volume size limited to 2TB
- also uses cylinder groups, superblocks, inodes, ...

ext2 has more attributes!

- c: compressed
- s: secured
- S: synchronized
- A: append mode

symbolic links

- symlink: special file that contains name of another file
- stored in file data blocks, or

→ inode contains actual file name

clean/dirty state (→ kind of a simple journal)

- after OS crash: fsck recommended/enforced

→ regular file system checks (fsck), even if clean

ext2 performance

- inodes and data blocks “close” to each other on hard disk
- on magnetic disks: reduces seek times

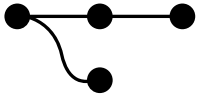
Preallocation:

- allocating a block to a file results in allocating up to 8 continuous blocks
- improves write- and read performance

Bitmaps for

- inode allocation
- data allocation

ext2 limits



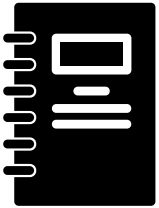
maximum file size

- depends on block size b
- $min((\frac{b}{4})^3 + (\frac{b}{4})^2 + \frac{b}{4} + 12) * b, 2^{32} * b)$
  - b=1KB: 16GB
  - b=4KB: 2 TB

ext3

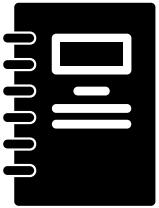
- based on ext2
- **journaling** file system
- file systems can grow dynamically
- hash tree for big directories

ext3 Journal



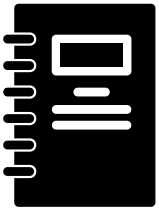
- changes to files stored in a journal
  - in principle a cyclic log
- first change noted in journal
- then executed in file system
- after crash: allows fixing inconsistencies easier

Journal - typical scenario



- file system is consistent
- changes are requested
- changes noted in journal
- changes executed in file system
- similar to stable storage concept

journaling - delete file example



deleting a file may need two steps:

1. remove reference from directory
2. delete inode

Crash between the two steps?

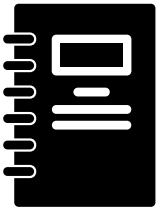
- orphaned inode
- inconsistency

Change their order?

- directory references non-existing inode

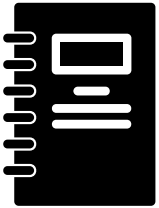
$\rightarrow$ using that inode may have fatal consequences

Journal



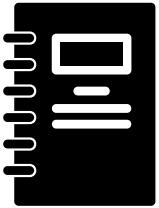
- Without Journal: fsck - file system check at reboot and **hope to find those inconsistencies**
- Now:
  - Read entries from journal
  - execute changes if required
- much faster than fsck
- changes become atomic:
  - either completed before the crash
  - or executed after the crash based on the journal
  - or not at all if not yet in the journal

implementing journals



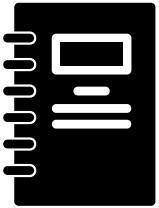
- storage: regular file, hidden file, special disk area, separate device?
- do we need a journal for the journal?
- must be able to check the integrity of the journal
  - checksum
  - ignore entries with incorrect checksum

physical journals



- writes a copy of each block
  - first into the journal
  - then on the disk
- Crash:
  - neither in journal nor on disk: no change
  - only in journal: copy to disk
  - already on disk: nothing to do
- high overhead
- acceptable for high correctness requirements

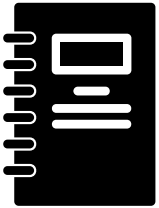
logical journals



- only meta-data written to journal
- trades safety against performance
- may lead to asynchronicity between meta-data and data

$\rightarrow$ for example, a correctly resized file but garbage content

journals in ext3



- full journal (no risk): 1. data → journal; 2. data → disk
- ordered (medium risk): 1. meta-data → journal; 2. data → disk
- write-back (highest risk): 1. meta-data → journal; 2. data "eventually" → disk (`sync`)
- no checksums on journal

- successor of ext3
- volume size up to 1 exibyte (2^{60})
- file size up to 16 tebibytes (2^{40})
- extents
- preallocation
- journals with checksum


ITS OVER
ITS FINALLY OVER
quickmeme.com