

ETH zürich @spoc\_eth

ADRIAN PERRIG & TORSTEN HOEFLER

## Networks and Operating Systems (252-0062-00)

### Chapter 8: Filesystem Implementation

ETH zürich @spoc\_eth

## Thanks for the feedback! ☺

- Some answers:
  - I'll provide references to books (I'm not **only** teaching from books)
    - Yesterday: "Operating Systems - An Advanced Course", Chapter 3a
    - Today: Anderson/Dahlin: 11/12 (partial), 13 (full)
  - Everything that's mentioned on the slides is essential
    - You should make sure you understand it
    - This may require listening ☺
    - Everything else and the stories I tell are optional
  - Why are your slides not self-contained?
    - See Rebecca Schumann "Digital Slideshows are the scourge of education"
  - I talked to the assistants to deepen parts of the lecture in the exercise
    - I "hope" that works --- they're open for additional feedback!
  - Next Friday is Karfreitag (probably free?)
    - Exercises will not be replaced but skipped! You may go to Thursday's exercise!

ETH zürich @spoc\_eth

## File system operations

We've already seen the file system as a naming scheme.

Directory (name space) operations:

- Link (bind a name)
- Unlink (unbind a name)
- Rename
- List entries

ETH zürich @spoc\_eth

## Acyclic-Graph Directories

- Two different names (aliasing)
  - If *dict* deletes *list* ⇒ dangling pointer

Solutions:

- Backpointers, so we can delete all pointers
  - Variable size records a problem
- Backpointers using a daisy chain organization
- Entry-hold-count solution

- New directory entry type
  - Link – another name (pointer) to an existing file
  - Resolve the link – follow pointer to locate the file

ETH zürich @spoc\_eth

## General Graph Directory

- How do we guarantee no cycles?
  - Options:
    - Allow only links to files and not directories
    - Garbage collection (with cycle collector)
    - Check for cycles when every new link is added
  - Restrict directory links to parents
    - E.g., "." and ".."
    - All cycles are therefore trivial

ETH zürich @spoc\_eth

## Access Control

**ETH zürich** social.ethz.ch @spoc\_eth

### Protection

- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List

**ETH zürich** social.ethz.ch @spoc\_eth

### Access control matrix

For a single file or directory:

Principals

	A	B	C	D	E	F	G	H	J	...
Read	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
Write	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>						
Append	<input checked="" type="checkbox"/>				<input checked="" type="checkbox"/>					
Execute	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>						
Delete	<input checked="" type="checkbox"/>									
List	<input checked="" type="checkbox"/>				<input checked="" type="checkbox"/>					
...										

Problem: how to scalably represent this matrix?

**ETH zürich** social.ethz.ch @spoc\_eth

### Row-wise: ACLs

- Access Control Lists
  - For each right, list the principals
  - Store with the file
- Good:
  - Easy to change rights quickly
  - Scales to large numbers of files
- Bad:
  - Doesn't scale to large numbers of principals

**ETH zürich** social.ethz.ch @spoc\_eth

### Column-wise: Capabilities

- Each principal with a right on a file holds a *capability* for that right
  - Stored with principal, not object (file)
  - Cannot be forged or (sometimes) copied
- Good:
  - Very flexible, highly scalable in principals
  - Access control resources charged to principal
- Bad:
  - Revocation: hard to change access rights (need to keep track of who has what capabilities)

**ETH zürich** social.ethz.ch @spoc\_eth

### POSIX (Unix) Access Control


- Simplifies ACLs: each file identifies 3 principals:
  - Owner (a single user)
  - Group (a collection of users, defined elsewhere)
  - The World (everyone)
- For each principal, file defines 3 rights:
  - Read (or traverse, if a directory)
  - Write (or create a file, if a directory)
  - Execute (or list, if a directory)

**ETH zürich** social.ethz.ch @spoc\_eth

### Example


```

drwx-x-x-x 9 htor htor 4096 May  9 13:14 pagai
ls -l /projekte/livn/livn-son < 09.05.13 19:00:49 >
total 860
drwx-x-x-x 3 htor htor 4096 Jan 29 15:58 autotconf
drwx-x-x-x 4 htor htor 4096 Dec 25 13:20 bindings
drwx-x-x-x 4 htor htor 4096 Jan 29 15:57 cmake
-rw-r--r-- 1 htor htor 16401 Dec 25 13:20 CMakelists.txt
-rw-r--r-- 1 htor htor 2782 Jan 29 15:57 CODE_OWNERS.TXT
-rw-r--r-- 1 htor htor 65932 Jan 29 15:57 configure
drwx-x-x-x 1 htor htor 10048 Dec 25 13:20 CREDITS.TXT
drwx-x-x-x 11 htor htor 4096 Apr  4 11:13 docimg
drwx-x-x-x 10 htor htor 4096 Jan 29 15:57 docs
drwx-x-x-x 10 htor htor 4096 Dec 25 13:20 examples
drwx-x-x-x 4 htor htor 4096 Dec 25 13:20 include
-rw-r--r-- 10 htor htor 4096 Jan 29 15:58 lib
-rw-r--r-- 1 htor htor 3254 Jan 29 15:57 LICENSE.TXT
-rw-r--r-- 1 htor htor 752 Dec 25 13:20 LLVMBuild.txt
-rw-r--r-- 1 htor htor 1805 Dec 25 13:20 Livn.spec.in
-rw-r--r-- 1 htor htor 9618 Jan 29 15:58 Makefile
-rw-r--r-- 1 htor htor 2596 Dec 25 13:20 Makefile.common
-rw-r--r-- 1 htor htor 12068 Jan 29 15:57 Makefile.config.in
-rw-r--r-- 1 htor htor 75048 Jan 29 15:57 Makefile.rules
-rw-r--r-- 1 htor htor 4096 Dec 25 13:21 projects
-rw-r--r-- 1 htor htor 687 Jan 29 15:58 README.txt
drwx-x-x-x 3 htor htor 4096 Dec 25 13:20 runtime
drwx-x-x-x 27 htor htor 4096 Jan 29 15:57 test
drwx-x-x-x 25 htor htor 4096 Dec 25 13:21 tests
drwx-x-x-x 11 htor htor 4096 Jan 29 15:57 unittests
drwx-x-x-x 32 htor htor 4096 Jan 29 15:57 utils
    
```

ETH zürich  [social.inf.ethz.ch](http://social.inf.ethz.ch)  
@spoc\_eth


## Full ACLs

- **POSIX now supports full ACLs**
  - Rarely used, interestingly
  - setfacl, getfacl, ...
- **Windows has very powerful ACL support**
  - Arbitrary groups as principals
  - Modification rights
  - Delegation rights


ETH zürich  [social.inf.ethz.ch](http://social.inf.ethz.ch)  
@spoc\_eth

## Our Small Quiz

- **True or false (raise hand)**
  - A file name identifies a string of data on a storage device
  - The file size is part of the file's metadata
  - Names provide a means of abstraction through indirection
  - Names are always assigned at object creation time
  - A context is implicit to a name
  - A context is implicit to an object
  - Name resolve may be specific to a context
  - Each file has exactly one name
  - The call "unlink file" always removes the contents of "file"
  - A fully qualified domain name is resolved recursively starting from the left
  - A full (absolute) path identifies a unique file (piece of data)
  - A full (absolute) path identifies a unique name
  - Stable bindings can be changed with bind()
  - Each name identifies exactly one object in a single context


ETH zürich  [social.inf.ethz.ch](http://social.inf.ethz.ch)  
@spoc\_eth

## File types

ETH zürich  [social.inf.ethz.ch](http://social.inf.ethz.ch)  
@spoc\_eth


## Is a directory a file?

- **Yes...**
  - Allocated just like a file on disk
  - Has entries in other directories like a file
- **...and No...**
  - Users can't be allowed to read/write to it
  - *Corrupt file system data structures*
  - *Bypass security mechanisms*
  - File system provides special interface
  - *opendir, closedir, readdir, seekdir, telldir, etc.*

ETH zürich  [social.inf.ethz.ch](http://social.inf.ethz.ch)  
@spoc\_eth

## Directory Implementation

- **Linear list** of (file name, block pointer) pairs
  - Simple to program
  - Lookup is slow for lots of files (linear scan)
- **Hash Table** – linear list with closed hashing.
  - Fast name lookup
  - collisions
  - fixed size
- **B-Tree** – name index, leaves are block pointers
  - Increasingly common
  - Complex to maintain, but scales well

ETH zürich  [social.inf.ethz.ch](http://social.inf.ethz.ch)  
@spoc\_eth

## File types

- **Other file types treated "specially" by the OS**
- **Simple, common cases:**
  - Executable files
  - Directories, symbolic links, other file system data
- **Some distinguish between text and binary**
- **Some have many types**
  - "Document" or "media" types
  - Used to select default applications, editors, etc.



ETH zürich @spoc\_eth

### Unix devices and other file types

- **Unix also uses the file namespace for**
  - Naming I/O devices (*/dev*)
  - Named pipes (FIFOs)
  - Unix domain sockets
- **More recently:**
  - Process control (*/proc*)
  - OS configuration and status (*/proc*, */sys*)
- **Plan 9 from Bell Labs**
  - Evolution of Unix: almost *everything* is a file

ETH zürich @spoc\_eth

### Executable files

- **Most OSes recognize binary executables**
  - Sometimes with a "magic number"
  - Will load, dynamically link, and execute in a process
- **Other files are sometimes recognized**
  - E.g. "#!" script files in Unix  
`#!/usr/bin/python`

ETH zürich @spoc\_eth

### File system operations

**File operations:**

- **Create and variants**
  - Unix: `mknod`, `mkfifo`, `ln -s`, ...
- **Change access control**
  - Unix: `chmod`, `chgrp`, `chown`, `setfacl`, ...
- **Read metadata**
  - Unix: `stat`, `fstat`, ...
- **Open**
  - Operation: `file` → `open file handle`


ETH zürich @spoc\_eth

### "Files" vs. "Open Files"

- **Typical operations on files:**
  - Rename, stat, create, delete, etc.
  - Open
- **Open creates an "open file handle"**
  - Different class of object
  - Allows reading and writing of file data


ETH zürich @spoc\_eth

### Open File Interface

ETH zürich  [spoc.inf.ethz.ch](http://spoc.inf.ethz.ch)  
@spoc\_eth


## Kinds of files

- Byte sequence**
  - The one you're probably familiar with
- Record sequence**
  - Fixed (at creation time) records
  - Mainframes or minicomputer OSes of the 70s/80s
- Key-based, tree structured**
  - E.g. IBM Indexed Sequential Access Method (ISAM)
  - Mainframe feature, now superseded by databases
  - In other words, moved into libraries

ETH zürich  [spoc.inf.ethz.ch](http://spoc.inf.ethz.ch)  
@spoc\_eth


## Byte-sequence files

- File is a vector of bytes**
  - Can be appended to
  - Can be truncated
  - Can be updated in place
  - Typically no "insert".
- Accessed as:**
  - Sequential files (rare these days)
  - Random access

ETH zürich  [spoc.inf.ethz.ch](http://spoc.inf.ethz.ch)  
@spoc\_eth

## Random access


- Support read, write, seek, and tell**
  - State: current position in file
  - Seek** absolute or relative to current position.
  - Tell** returns current index
- Index units:**
  - For byte sequence files, offset in bytes

ETH zürich  [spoc.inf.ethz.ch](http://spoc.inf.ethz.ch)  
@spoc\_eth

## Record-sequence files


- File is now a vector of fixed-size records**
  - Can be appended to
  - Can be truncated
  - Can be updated in place
  - Typically no "insert".
- Record size (and perhaps format) fixed at creation time**
  - Read/write/seek operations take records and record offsets instead of byte addresses

Compare with databases!

ETH zürich  [spoc.inf.ethz.ch](http://spoc.inf.ethz.ch)  
@spoc\_eth

## Memory-mapped files


- Basic idea: use VM system to cache files**
  - Map file content into virtual address space
  - Set the backing store of region to file
  - Can now access the file using load/store
- When memory is paged out**
  - Updates go back to file instead of swap space

ETH zürich  [spoc.inf.ethz.ch](http://spoc.inf.ethz.ch)  
@spoc\_eth

## On-disk data structures

**Disk addressing**

- Disks have tracks, sectors, spindles, etc.
  - And bad sector maps!
- More convenient to use **logical block addresses**
  - Treat disk as compact linear array of usable blocks
  - Block size typically 512 bytes
  - Ignore geometry except for performance (later!)
- Also abstracts other block storage devices
  - Flash drives (load-levelling, etc.)
  - Storage-area Networks (SANs)
  - Virtual disks (RAM, RAID, etc.)



**Implementation aspects**

- Directories and indexes
  - Where on the disk is the data for each file?
- Index granularity
  - What is the unit of allocation for files?
- Free space maps
  - How to allocate more sectors on the disk?
- Locality optimizations
  - How to make it go fast in the common case

**File system implementations**


	FAT	FFS	NTFS	ZFS
Index structure	Linked list	Fixed, asymmetric tree	Dynamic tree	Dynamic COW tree
Index granularity:	Block	Block	Extent	Block
Free space management	FAT Array	Fixed bitmap	Bitmap in file	Log-structured space map
Locality heuristics	Defragmentation	Block groups, Reserve space	Best fit, Defragmentation	Write anywhere, Block groups

See book for details

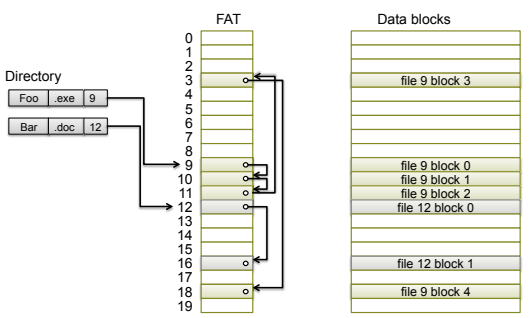
**FAT-32**

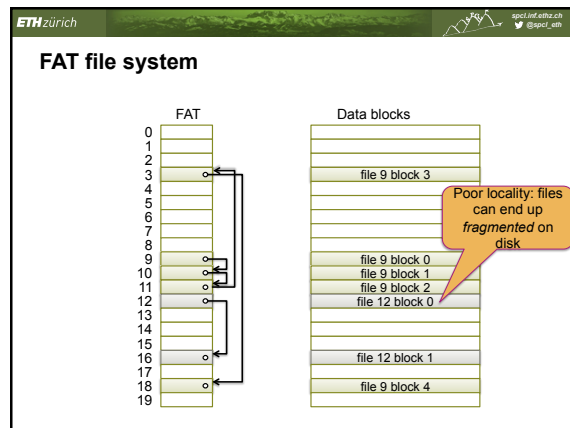
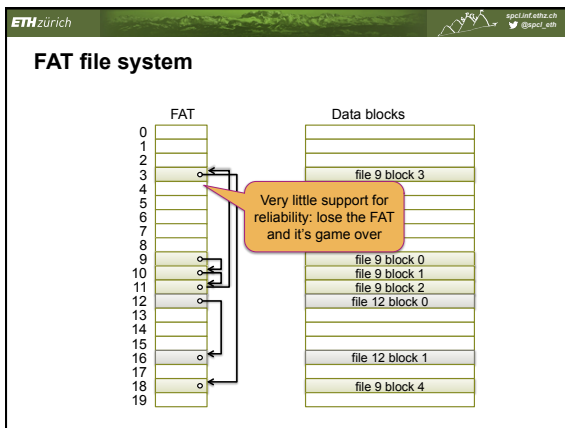
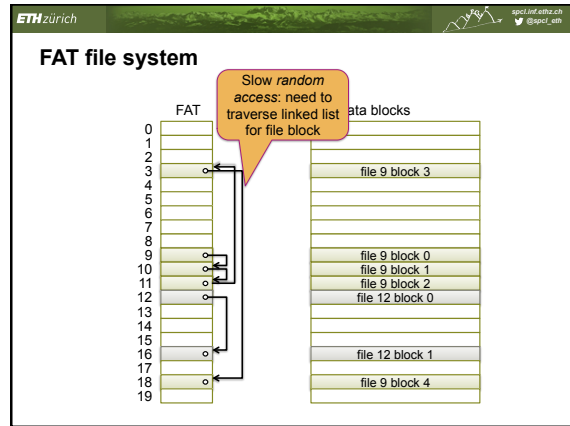
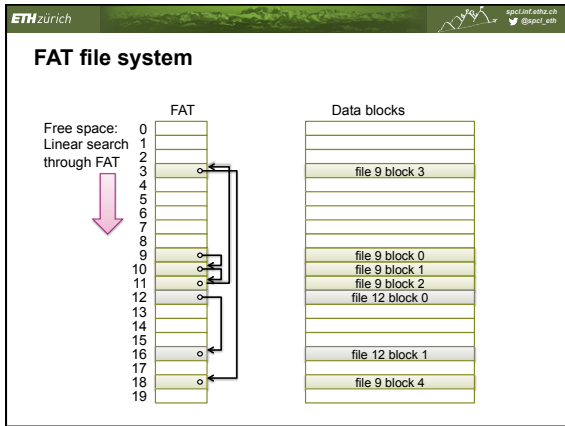
**FAT background**

- Very old – dates back to 1970s!
- No access control
- Very little metadata
- Limited volume size
- No support for hard links
- BUT still extensively used ☹
  - Flash devices, cameras, phones



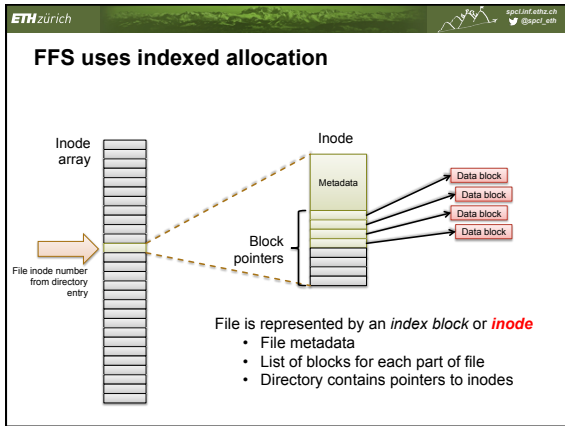
**FAT file system**





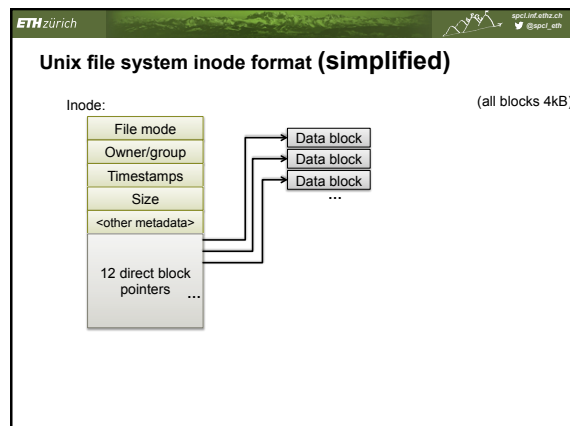
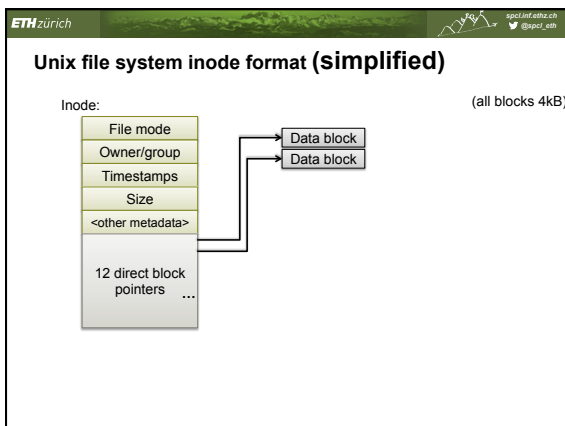
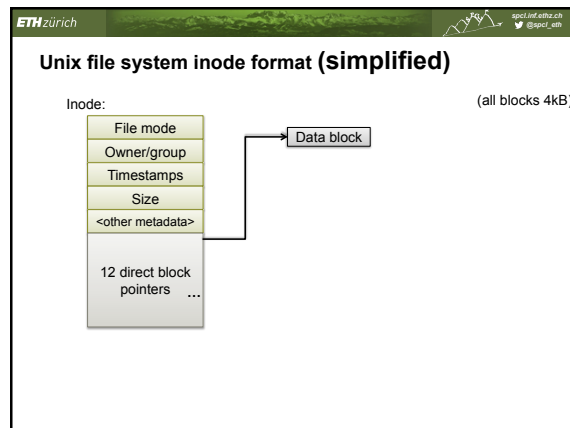
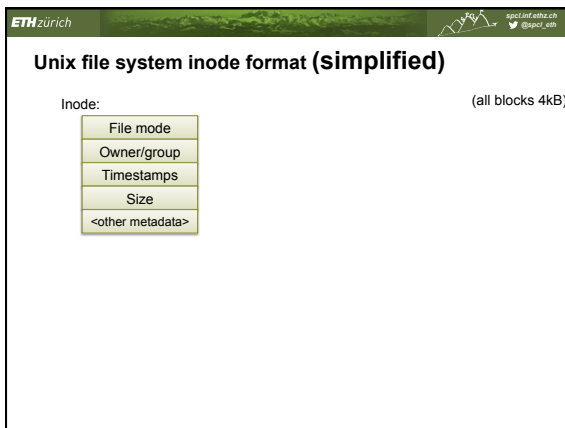
**FFS**

- Unix Fast File System (FFS)**
- First appeared in BSD in the mid 1980's
  - Based on original Unix FS, with performance optimizations
  - Basis for Linux ext{2,3} file systems

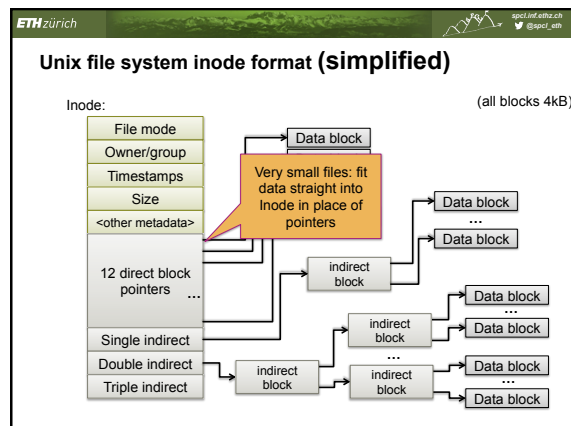
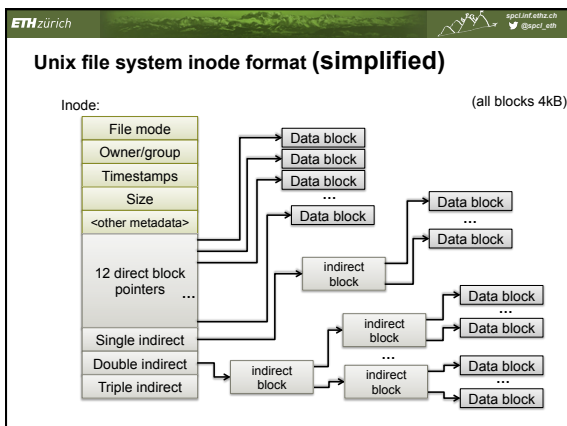
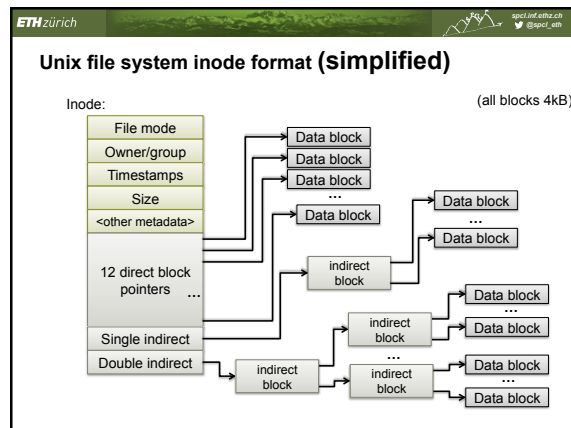
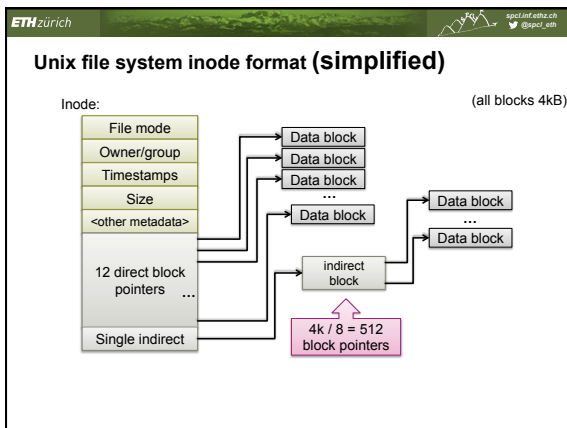
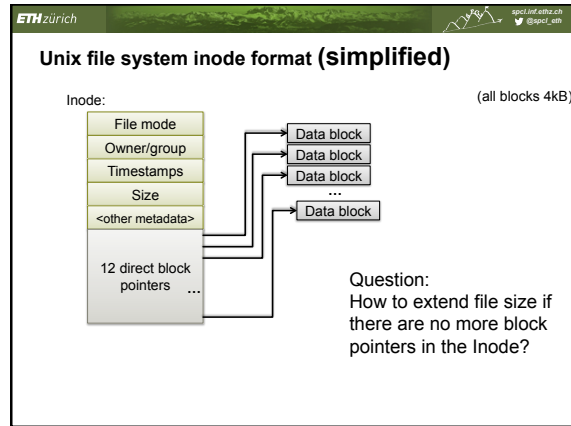
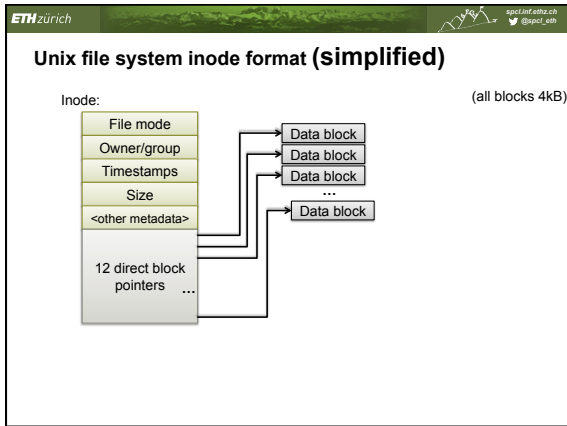


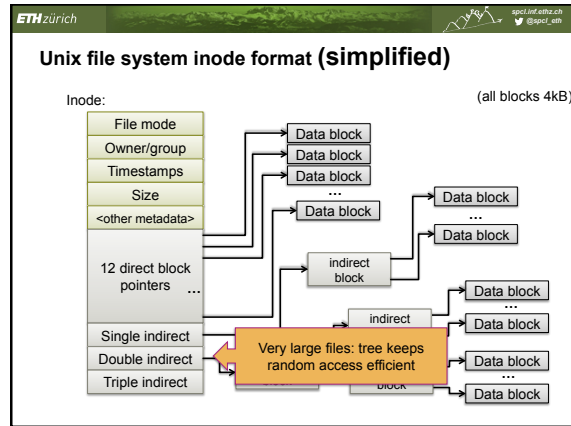
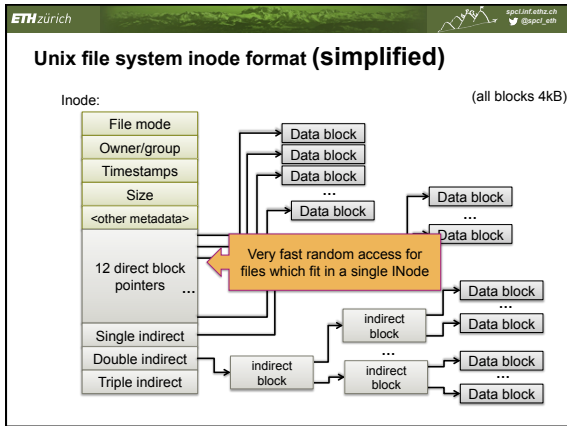
**Inode and file size in FFS**

- **Example:**
  - Inode is 1 block = 4096 bytes
  - Block addresses = 8 bytes
  - Inode metadata = 512 bytes
- **Hence:**
  - $(4096 - 512) / 8 = 448$  block pointers
  - $448 * 4096 = 1792\text{kB}$  max. file size

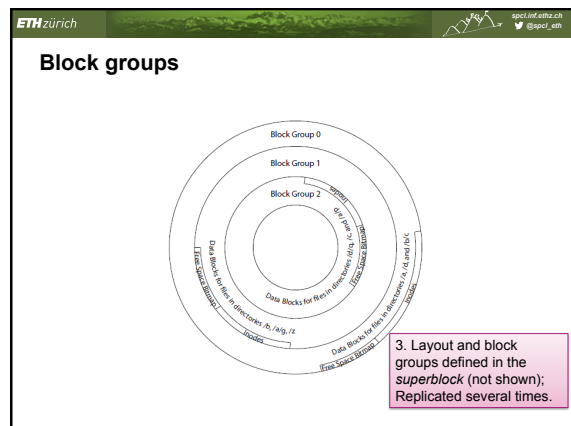
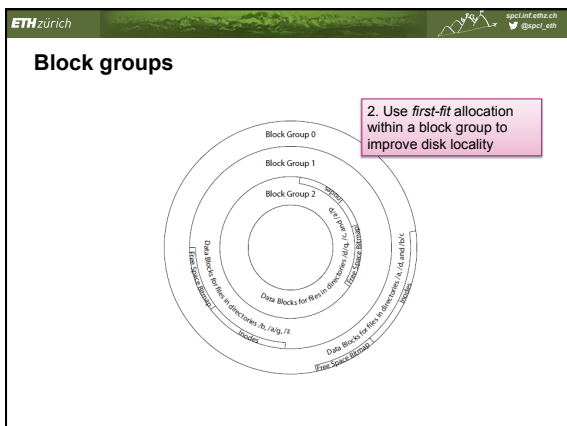
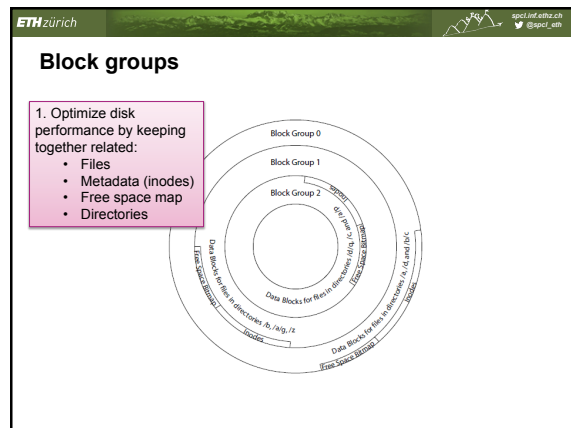








- ### Free space map
- **FFS uses a simple bitmap**
    - Initialized when the file system is created
    - One bit per disk (file system) block
  - **Allocation is reasonably fast**
    - Scan though lots of bits at a time
    - Bitmap can be cached in memory



ETH zürich

NTFS

ETH zürich

### NTFS Master file table

MFT

MFT record:

Std. info Attributes, data, metadata free

1 kB fixed size

Lots of options for what goes in here

ETH zürich

### NTFS small files

- Small file fits into MFT record:

Std. info Filename File data free

"resident" data

ETH zürich

### NTFS small files

- Small file fits into MFT record:

Std. info Filename File data free

"resident" data

- Hard links (multiple names) stored in MFT:

Std. info Filename1 Filename2 File data free

ETH zürich

### NTFS normal files

- MFT holds list of extents:

Std. info Filename Start, length Start, length Start, length free

Data (extent 0)

Data (extent 1)

Data (extent 2)

ETH zürich

### Too many attributes?

- Attribute list holds list of attribute locations

MFT entry

Std. info Attr. list Filename 1 Filename 2

2<sup>nd</sup> MFT entry

Std. info Start, length Start, length Start, length free

In addition, attributes can also be stored in extents => very large scaling (see book)

**ETH zürich** spezialinf.ethz.ch @spci\_eth

### Metadata files

- File system metadata in NTFS is held *in files!*

File num.	Name	Description
0	\$MFT	Master file table
1	\$MFTir	Copy of first 4 MFT entries
2	\$Logfile	Transaction log of FS changes
3	\$Volume	Volume information & metadata
4	\$AttrDef	Table mapping numeric IDs to attributes
5	.	Root directory
6	\$Bitmap	Free space bitmap
7	\$Boot	Volume boot record
8	\$BadClus	Bad cluster map
9	\$Secure	Access control list database
10	\$UpCase	Filename mappings to DOS
11	\$Extend	Extra file system attributes (e.g. quota)

**ETH zürich** spezialinf.ethz.ch @spci\_eth

### Metadata files

- File system metadata in NTFS is held *in files!*

File num.	Name	Description
0	\$MFT	Master file table
1	\$MFTir	Copy of first 4 MFT entries
2	\$Logfile	Transaction log of FS changes
3	\$Volume	Volume information & metadata
4	\$AttrDef	Table mapping numeric IDs to attributes
5	.	Root directory
6	\$Bitmap	Free space bitmap
7	\$Boot	Volume boot record
8	\$BadClus	Bad cluster map
9	\$Secure	Access control list database
10	\$UpCase	Filename mappings to DOS
11	\$Extend	Extra file system attributes (e.g. quota)

**ETH zürich** spezialinf.ethz.ch @spci\_eth

### Metadata files

- File system metadata in NTFS is held *in files!*

File num.	Name	Description
0	\$MFT	Master file table
1	\$MFTir	Copy of first 4 MFT entries
2	\$Logfile	Transaction log of FS changes
3	\$Volume	Volume information & metadata
4	\$AttrDef	Table mapping numeric IDs to attributes
5	.	Root directory
6	\$Bitmap	Free space bitmap
7	\$Boot	Volume boot record
8	\$BadClus	Bad cluster map
9	\$Secure	Access control list database
10	\$UpCase	Filename mappings to DOS
11	\$Extend	Extra file system attributes (e.g. quota)

**ETH zürich** spezialinf.ethz.ch @spci\_eth

### Metadata files

- File system metadata in NTFS is held *in files!*

File num.	Name	Description
0	\$MFT	Master file table
1	\$MFTir	Copy of first 4 MFT entries
2	\$Logfile	Transaction log of FS changes
3	\$Volume	Volume information & metadata
4	\$AttrDef	Table mapping numeric IDs to attributes
5	.	Root directory
6	\$Bitmap	Free space bitmap
7	\$Boot	Volume boot record
8	\$BadClus	Bad cluster map
9	\$Secure	Access control list database
10	\$UpCase	Filename mappings to DOS
11	\$Extend	Extra file system attributes (e.g. quota)

**ETH zürich** spezialinf.ethz.ch @spci\_eth

### Metadata files

- File system metadata in NTFS is held *in files!*

File num.	Name	Description
0	\$MFT	Master file table
1	\$MFTir	Copy of first 4 MFT entries
2	\$Logfile	Transaction log of FS changes
3	\$Volume	Volume information & metadata
4	\$AttrDef	Table mapping numeric IDs to attributes
5	.	Root directory
6	\$Bitmap	Free space bitmap
7	\$Boot	Volume boot record
8	\$BadClus	Bad cluster map
9	\$Secure	Access control list database
10	\$UpCase	Filename mappings to DOS
11	\$Extend	Extra file system attributes (e.g. quota)

**ETH zürich** spezialinf.ethz.ch @spci\_eth

### Metadata files

- File system metadata in NTFS is held *in files!*

File num.	Name	Description
0	\$MFT	Master file table
1	\$MFTir	Copy of first 4 MFT entries
2	\$Logfile	Transaction log of FS changes
3	\$Volume	Volume information & metadata
4	\$AttrDef	Table mapping numeric IDs to attributes
5	.	Root directory
6	\$Bitmap	Free space bitmap
7	\$Boot	Volume boot record
8	\$BadClus	Bad cluster map
9	\$Secure	Access control list database
10	\$UpCase	Filename mappings to DOS
11	\$Extend	Extra file system attributes (e.g. quota)

Question:  
Huh?  
Where is it then?  
Answer:  
First sector of volume points to first block of MFT

ETH zürich

**In-memory data structures**

ETH zürich

**Opening a file**

- Directories translated into kernel data structures on demand:

ETH zürich

**Reading and writing**

- Per-process open file table → index into...
- System open file table → cache of inodes

ETH zürich

**Efficiency and Performance**

- Efficiency dependent on:**
  - disk allocation and directory algorithms
  - types of data kept in file's directory entry
- Performance**
  - disk cache – separate section of main memory for frequently used blocks
  - free-behind and read-ahead – techniques to optimize sequential access
  - improve PC performance by dedicating section of memory as virtual disk, or RAM disk

ETH zürich

**Addendum**

- FAT file size limit (4 GB) – due to the file length entry in the FAT table (guess what, 32 bits)**
  - Volume size is up to 16 TB ( $2^{32} * 4k$  blocks)
- Insert data into the middle of a file**
  - May allocate a block (true)
  - But who wants to insert only block sizes?
  - Solution: Streams (D'oh, not in FFS/ext...?)

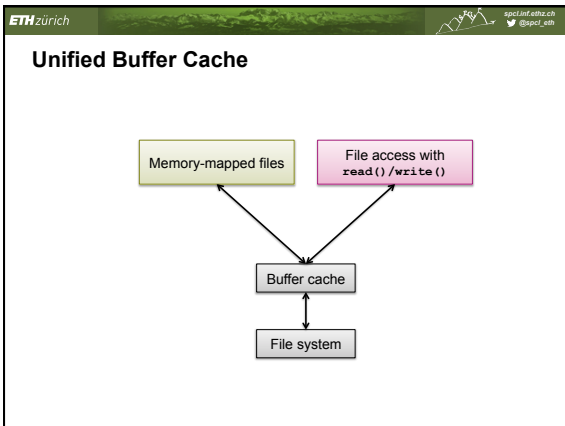
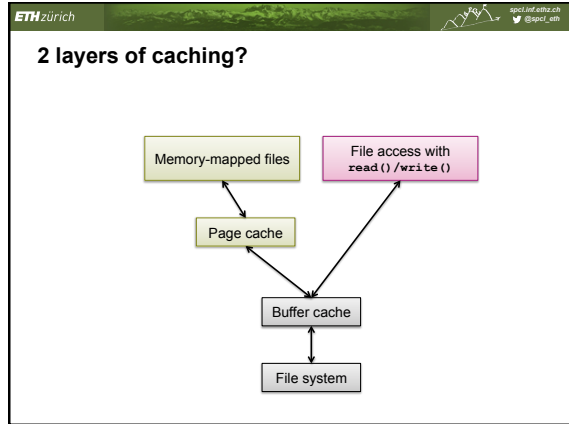
ETH zürich

**Addendum**

**3. Block Groups**

**Page Cache**

- A page cache caches pages rather than disk blocks using virtual memory techniques
- Memory-mapped I/O uses a page cache
- Routine I/O through the file system uses the buffer (disk) cache
- This leads to the following figure



**Recovery**

- Consistency checking – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies
- Use system programs to back up data from disk to another storage device (floppy disk, magnetic tape, other magnetic disk, optical)
- Recover lost file or disk by restoring data from backup