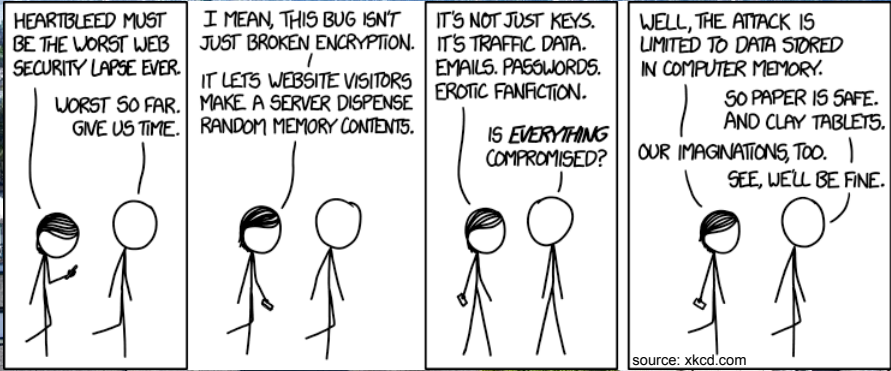


ETH zürich spcl.inf.ethz.ch
@spcl_eth

ADRIAN PERRIG & TORSTEN HOEFLER

Networks and Operating Systems (252-0062-00)

Chapter 8: Filesystem Implementation



HEARTBLEED MUST BE THE WORST WEB SECURITY LAPSE EVER.
WORST SO FAR. GIVE US TIME.

I MEAN, THIS BUG ISN'T JUST BROKEN ENCRYPTION. IT LETS WEBSITE VISITORS MAKE A SERVER DISPENSE RANDOM MEMORY CONTENTS.

IT'S NOT JUST KEYS. IT'S TRAFFIC DATA. EMAILS. PASSWORDS. EROTIC FANFICTION.
IS EVERYTHING COMPROMISED?

WELL, THE ATTACK IS LIMITED TO DATA STORED IN COMPUTER MEMORY. SO PAPER IS SAFE. AND CLAY TABLETS. OUR IMAGINATIONS, TOO. SEE, WE'LL BE FINE.

source: xkcd.com

ETH zürich spcl.inf.ethz.ch
@spcl_eth

Access Control

ETH zürich spci.inf.ethz.ch
@spci_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 spci.inf.ethz.ch
@spci_eth




Access control matrix

For a single file or directory:

Principals




	A	B	C	D	E	F	G	H	J	...
Rights	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"/>			<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    spcl.inf.ethz.ch
[@spcl_eth](https://twitter.com/spcl_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    spcl.inf.ethz.ch
[@spcl_eth](https://twitter.com/spcl_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
spl.inf.ethz.ch
@spl_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
spl.inf.ethz.ch
@spl_eth

Example

```




drwx--x--x  9 htor htor   4096 May  9 13:14 pagai
htor@lenny ~ -> ls -l projekte/llvm/llvm-svn < 09.05.13 19:08:49 >
total 860
drwx--x--x  3 htor htor   4096 Jan 29 15:58 autoconf
drwx--x--x  4 htor htor   4096 Dec 25 13:20 bindings
drwx--x--x  4 htor htor   4096 Jan 29 15:57 cmake
-rw-----  1 htor htor  16401 Dec 25 13:20 CMakeLists.txt
-rw-----  1 htor htor   2782 Jan 29 15:57 CODE_OWNERS.TXT
-rwx-----  1 htor htor 658352 Jan 29 15:57 configure
-rw-----  1 htor htor   10048 Dec 25 13:20 CREDITS.TXT
drwxr-xr-x 11 htor htor   4096 Apr  4 11:13 Debug
drwx--x--x 10 htor htor   4096 Jan 29 15:57 docs
drwx--x--x 10 htor htor   4096 Dec 25 13:20 examples
drwx--x--x  4 htor htor   4096 Dec 25 13:20 include
drwx--x--x 18 htor htor   4096 Jan 29 15:58 lib
-rw-----  1 htor htor   3254 Jan 29 15:57 LICENSE.TXT
-rw-----  1 htor htor    752 Dec 25 13:20 LLVMBuild.txt
-rw-----  1 htor htor   1865 Dec 25 13:20 llvm.spec.in
-rw-----  1 htor htor   8618 Jan 29 15:58 Makefile
-rw-----  1 htor htor   2599 Dec 25 13:20 Makefile.common
-rw-----  1 htor htor  12068 Jan 29 15:57 Makefile.config.in
-rw-----  1 htor htor  79586 Jan 29 15:57 Makefile.rules
drwx--x--x  4 htor htor   4096 Dec 25 13:21 projects
-rw-----  1 htor htor    687 Jan 29 15:58 README.txt
drwx--x--x  3 htor htor   4096 Dec 25 13:20 runtime
drwx--x--x 27 htor htor   4096 Jan 29 15:57 test
drwx--x--x 35 htor htor   4096 Dec 25 13:21 tools
drwx--x--x 11 htor htor   4096 Jan 29 15:57 unittests
drwx--x--x 32 htor htor   4096 Jan 29 15:57 utils

```

ETH zürich   sycl.inf.ethz.ch
 @sycl_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   sycl.inf.ethz.ch
 @sycl_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

10



ETH zürich   sycl.inf.ethz.ch
[@sycl_eth](https://twitter.com/sycl_eth)

File types

ETH zürich   sycl.inf.ethz.ch
[@sycl_eth](https://twitter.com/sycl_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   sycl.inf.ethz.ch
[@sycl_eth](https://twitter.com/sycl_eth)

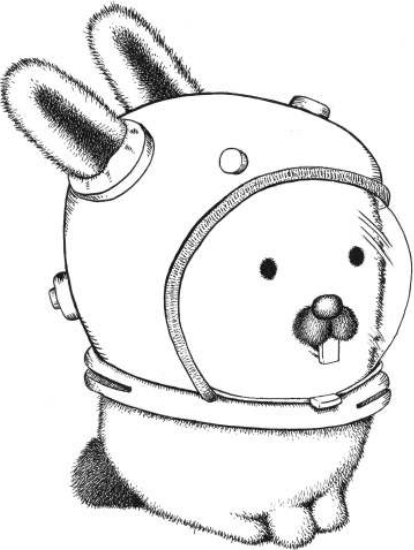
Directory Implementation

- **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   sycl.inf.ethz.ch
[@sycl_eth](https://twitter.com/sycl_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.



The slide features a header with the ETH zürich logo on the left, a green landscape graphic in the center, and the website 'spcl.inf.ethz.ch' and Twitter handle '@spcl_eth' on the right. The main content area contains the following text:



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   sycl.inf.ethz.ch
[@sycl_eth](https://twitter.com/sycl_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   sycl.inf.ethz.ch
[@sycl_eth](https://twitter.com/sycl_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   sycl.inf.ethz.ch
[@sycl_eth](https://twitter.com/sycl_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   sycl.inf.ethz.ch
[@sycl_eth](https://twitter.com/sycl_eth)

Open File Interface

ETH zürich   spci.inf.ethz.ch
 @spci_eth



Kinds of files

- 1. Byte sequence**
 - The one you're probably familiar with
- 2. Record sequence**
 - Fixed (at creation time) records
 - Mainframes or minicomputer OSes of the 70s/80s
- 3. 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   spci.inf.ethz.ch
 @spci_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   splotter.inf.ethz.ch
[@splotter_eth](https://twitter.com/splotter_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   splotter.inf.ethz.ch
[@splotter_eth](https://twitter.com/splotter_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   sycl.inf.ethz.ch
[@sycl_eth](https://twitter.com/sycl_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   sycl.inf.ethz.ch
[@sycl_eth](https://twitter.com/sycl_eth)

On-disk data structures

ETH zürich spci.inf.ethz.ch
@spci_eth

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.)



ETH zürich spci.inf.ethz.ch
@spci_eth

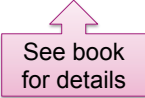
Implementation aspects




- **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

ETH zürich   spci.inf.ethz.ch
 @spci_eth

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



ETH zürich   spci.inf.ethz.ch
 @spci_eth

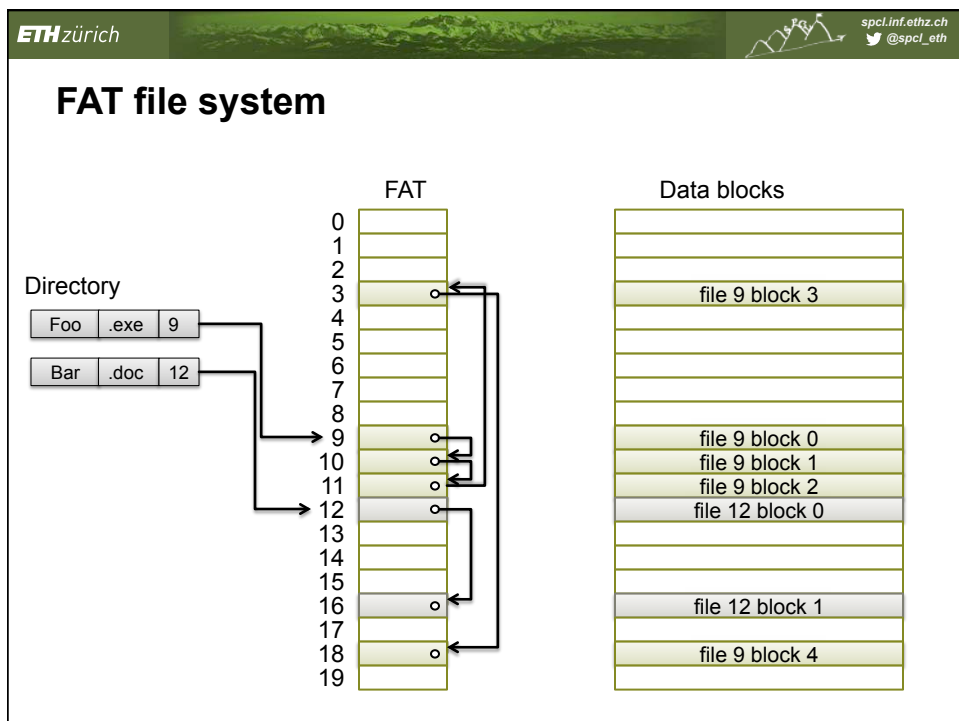
FAT-32

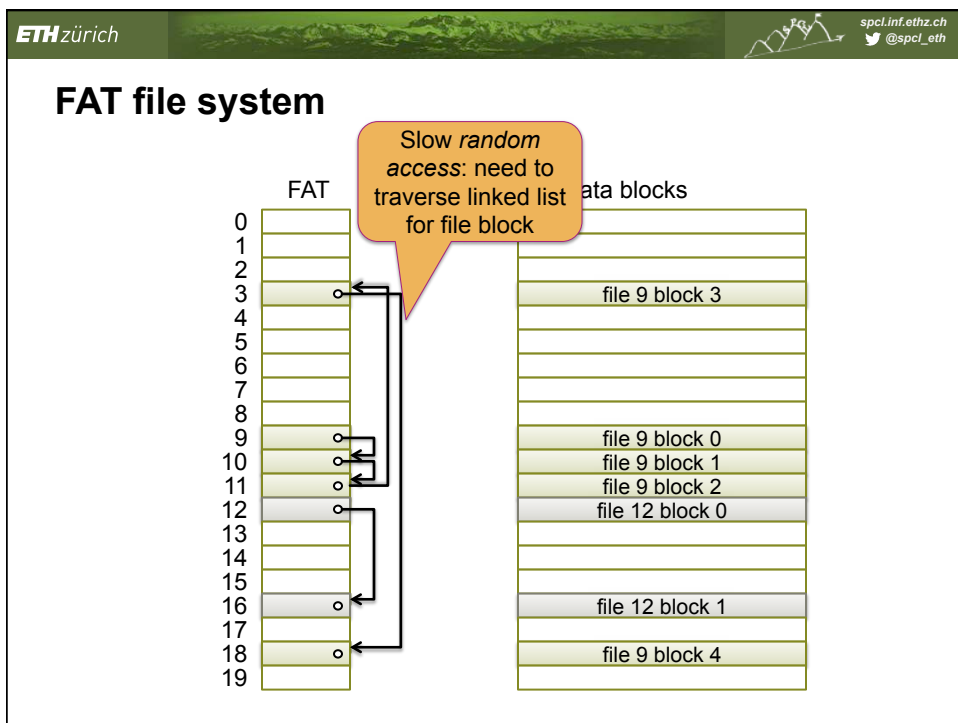
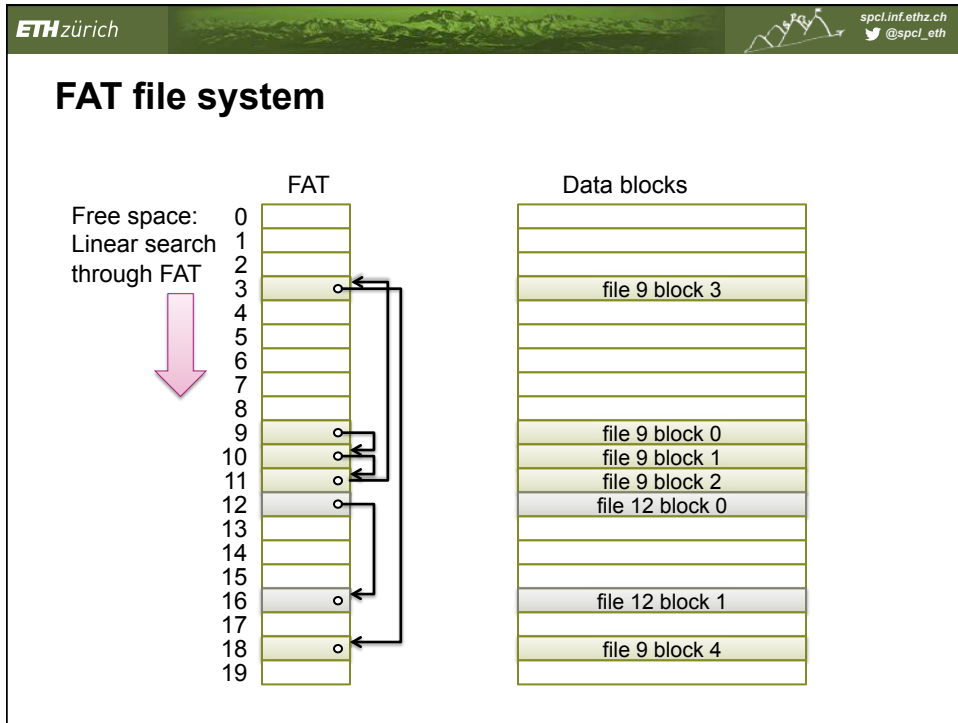
ETH zürich spc1.inf.ethz.ch
@spc1_eth

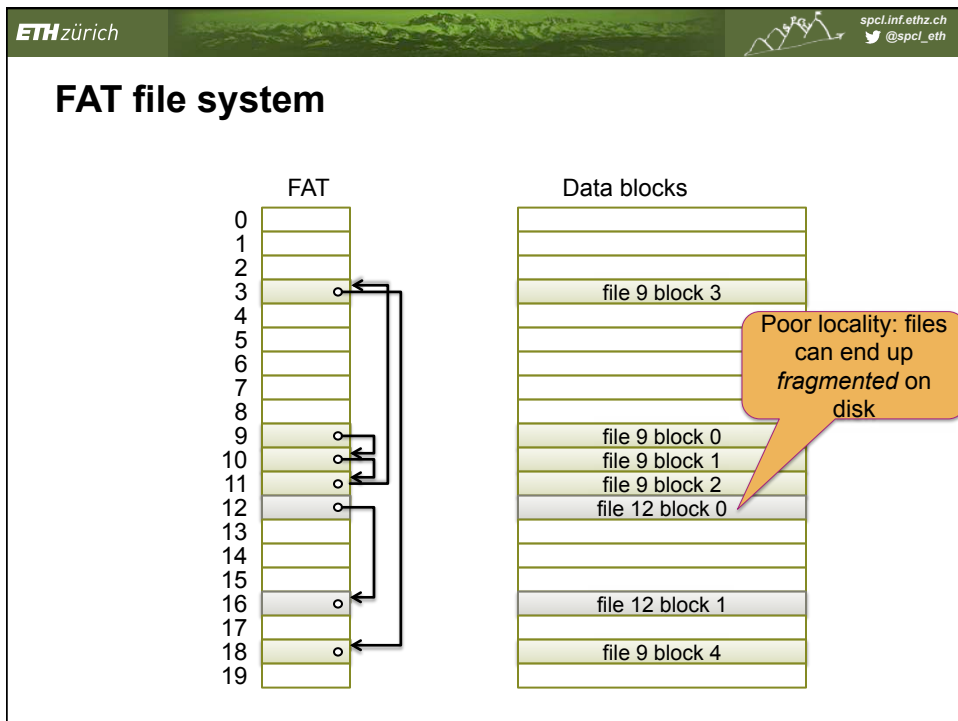
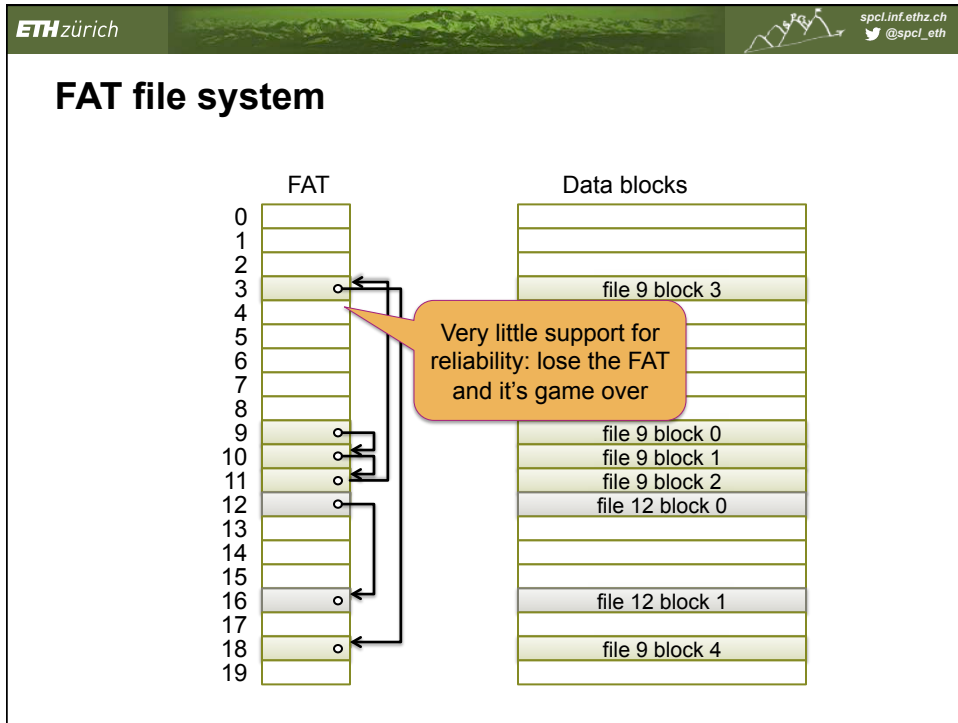
FAT background




- 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

- **Legend: During the development of Windows 3.0, it was customary to have regular meetings with Bill Gates to brief him on the status of the project. At one of the reviews, the topic was performance, and Bill complained, "You guys are spending all this time with your segment tuning tinkering. I could teach a twelve-year-old to segment-tune. I want to see some real optimization, not this segment tuning nonsense. I wrote FAT on an airplane, for heaven's sake."**










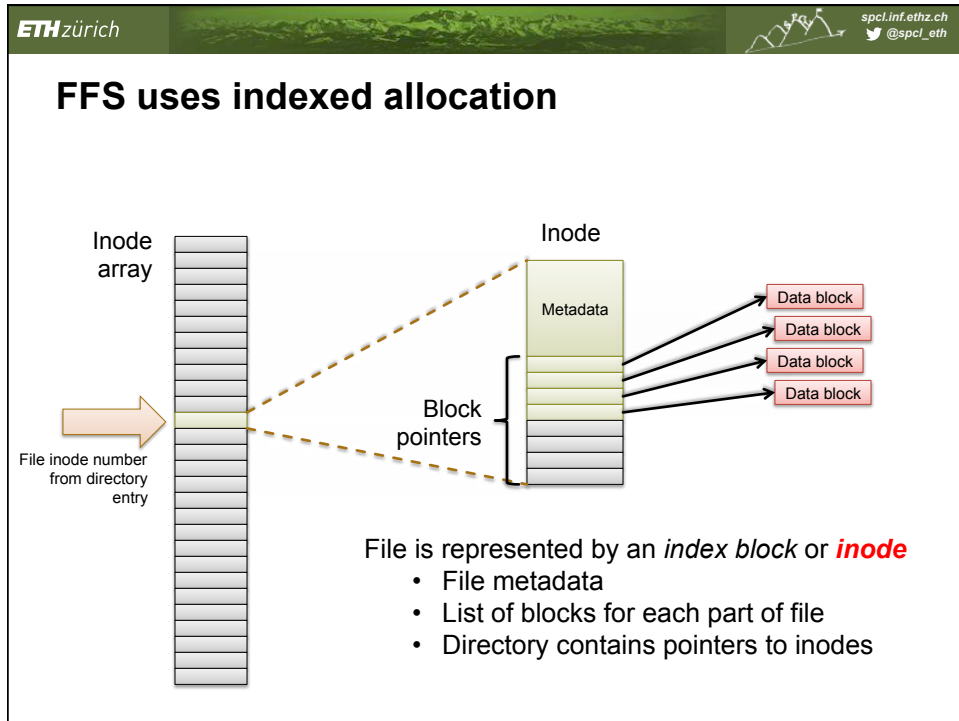
ETH zürich    spcl.inf.ethz.ch
[@spcl_eth](https://twitter.com/spcl_eth)

FFS

ETH zürich    spcl.inf.ethz.ch
[@spcl_eth](https://twitter.com/spcl_eth)

Unix Fast File System (FFS)

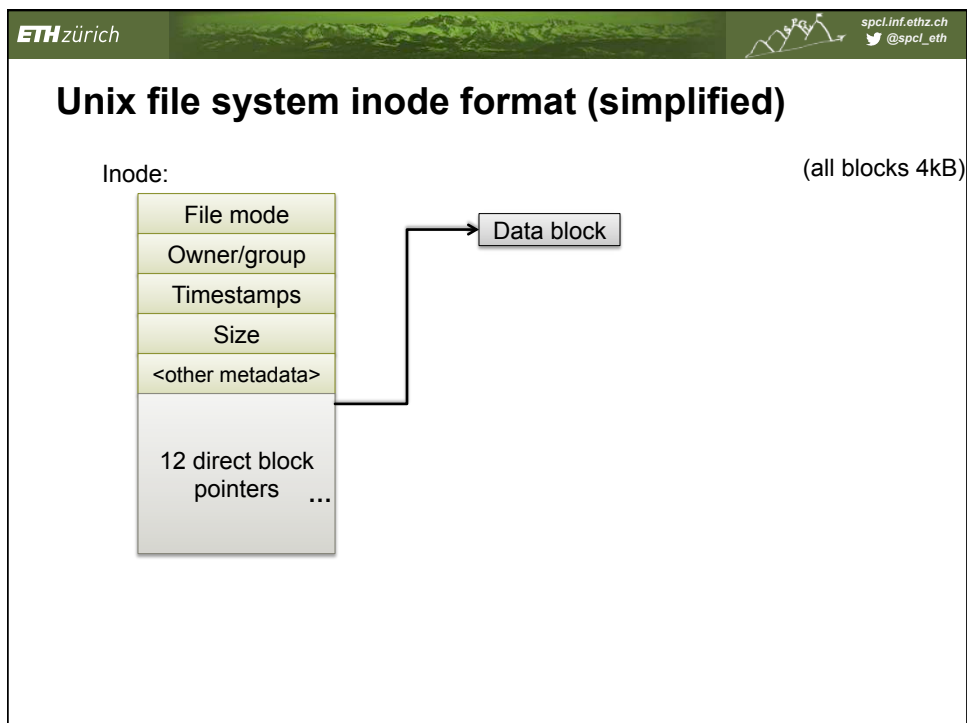
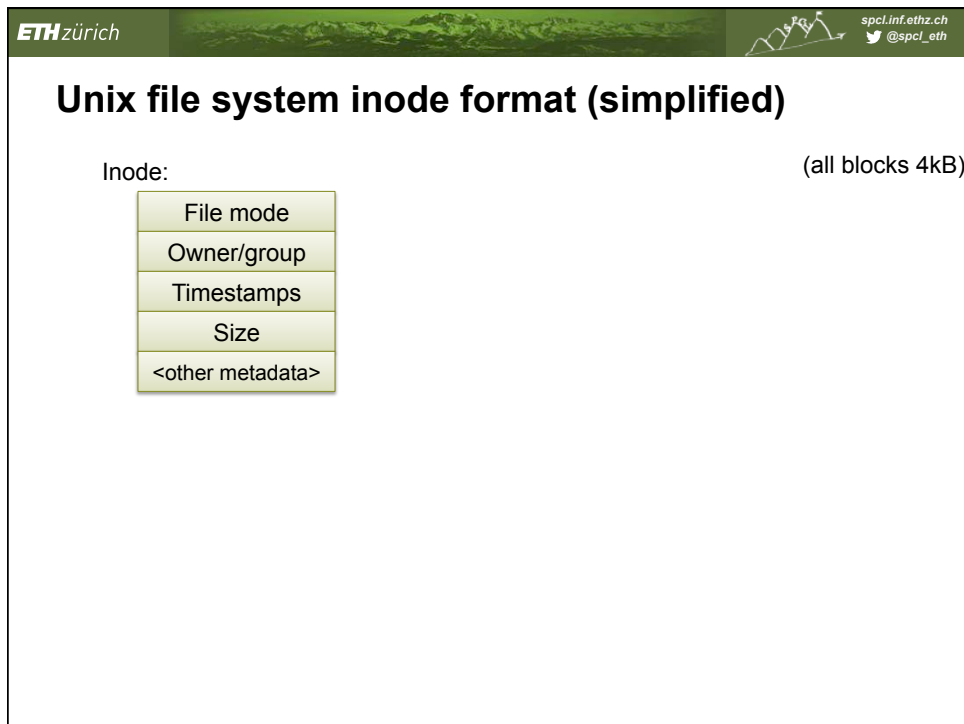
- First appeared in BSD in the mid 1980's
- Based on original Unix FS, with performance optimizations
- Basis for Linux ext{2,3} file systems
- **Recommended watching:**
 - Marshall Kirk McKusick "A Brief History of the BSD Fast Filesystem"
Keynote at USENIX FAST'15
(<https://www.youtube.com/watch?v=TMjgShRuYbg>)

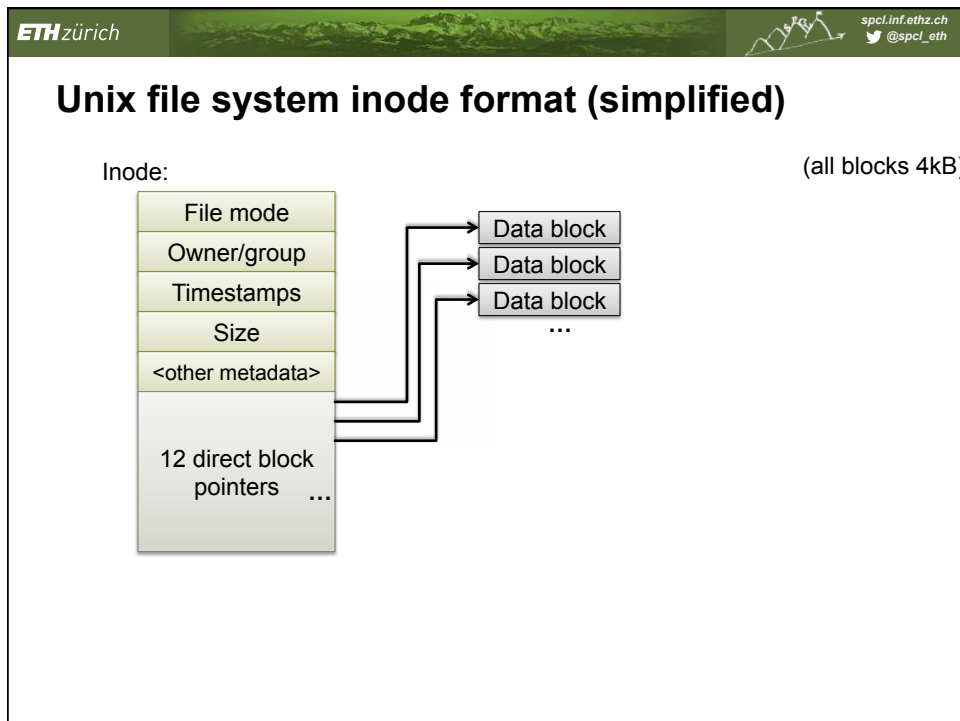
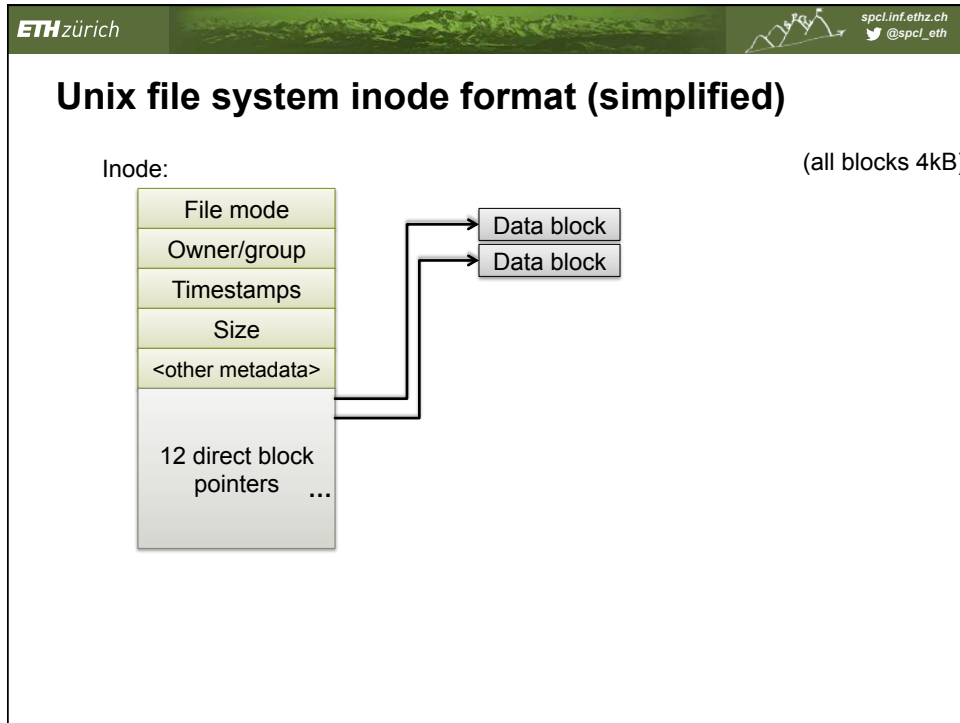


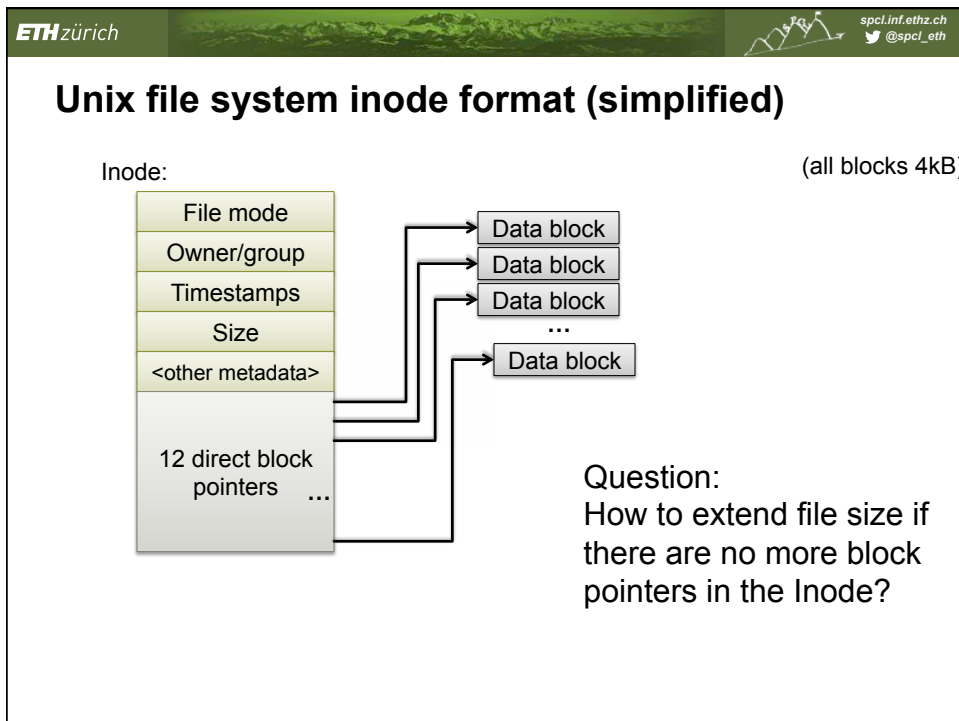
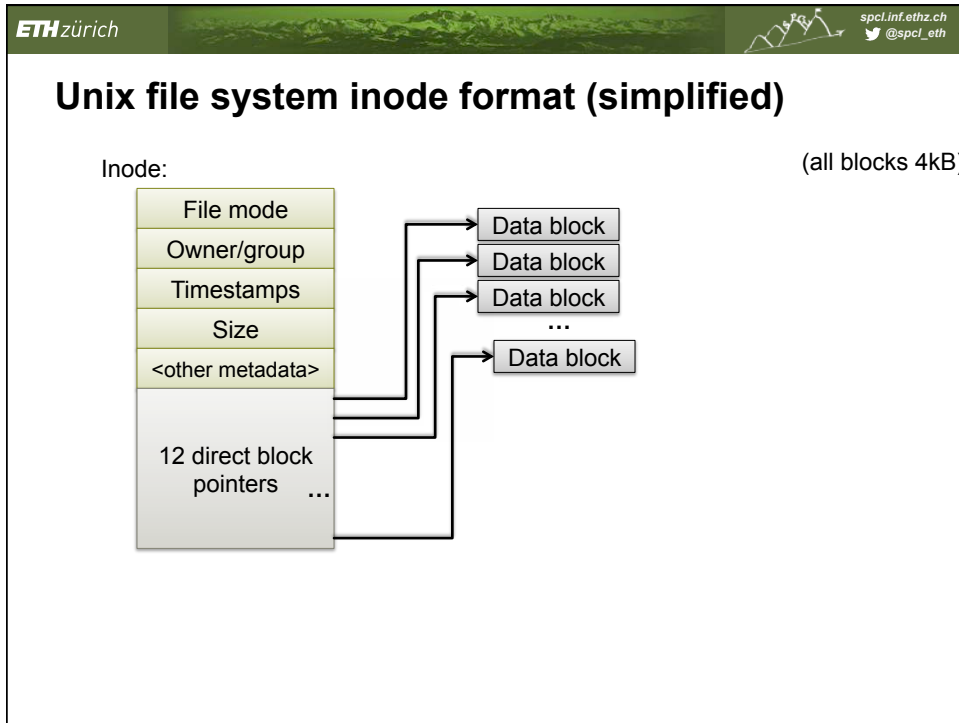
ETH zürich spci.inf.ethz.ch
@spci_eth

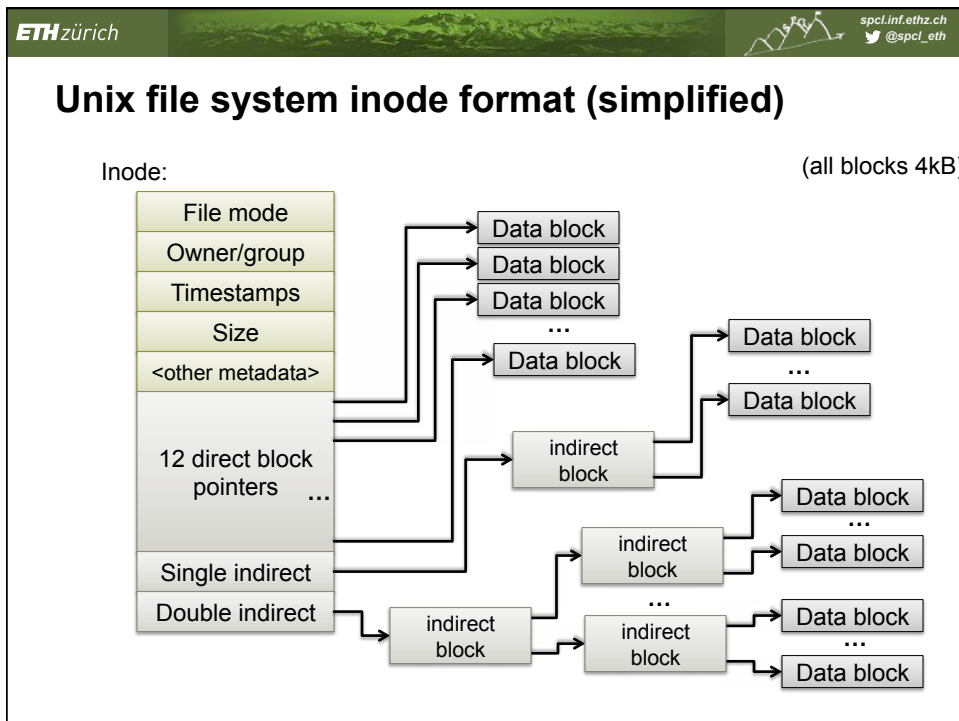
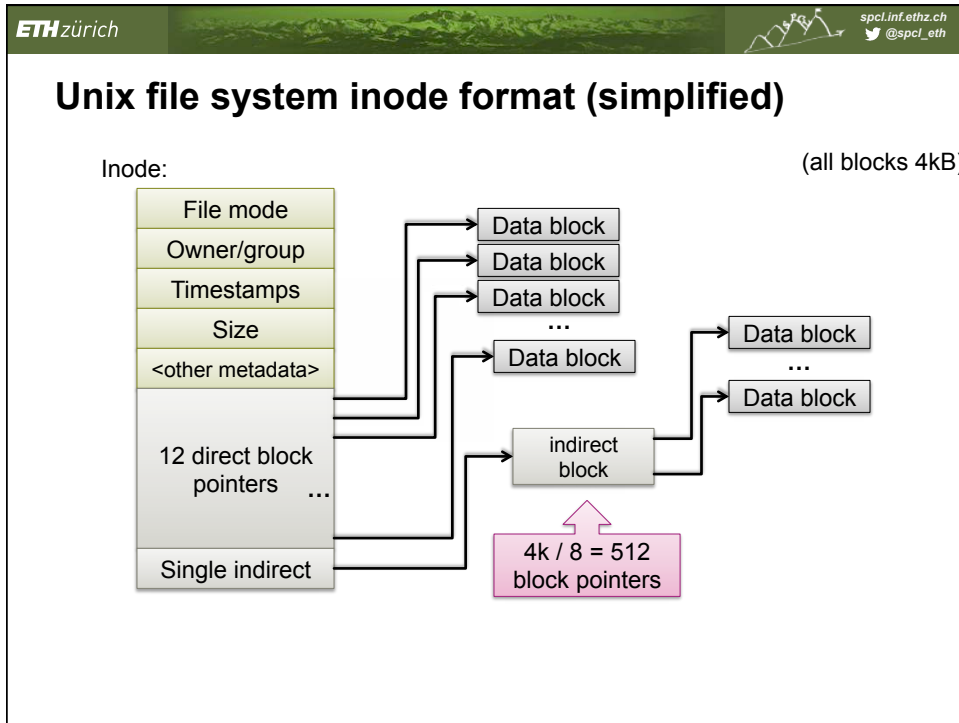
Inode and file size in FFS

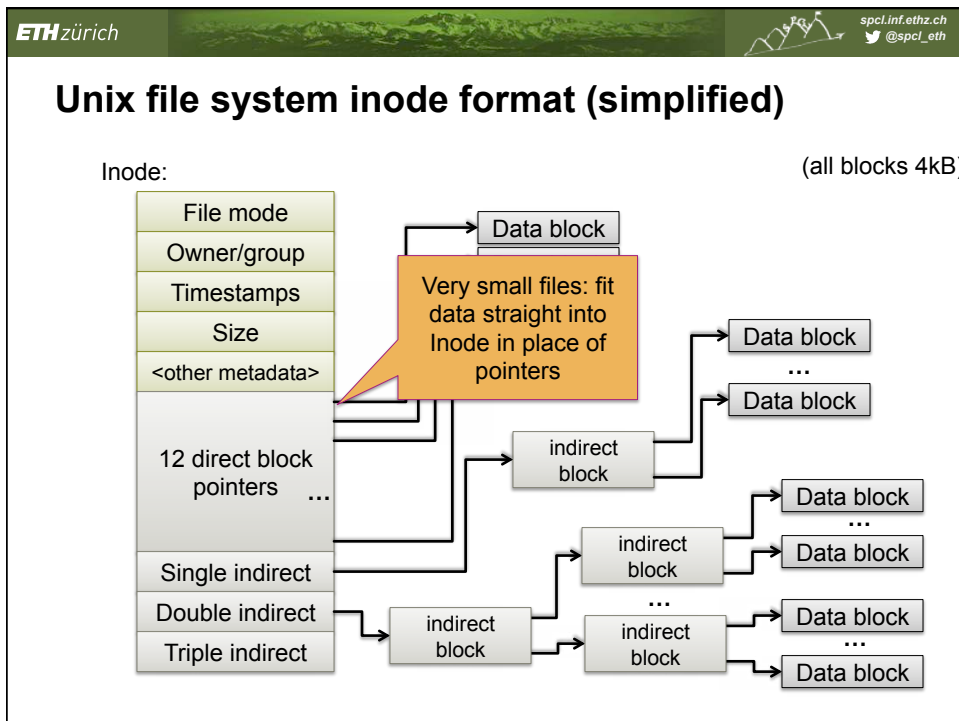
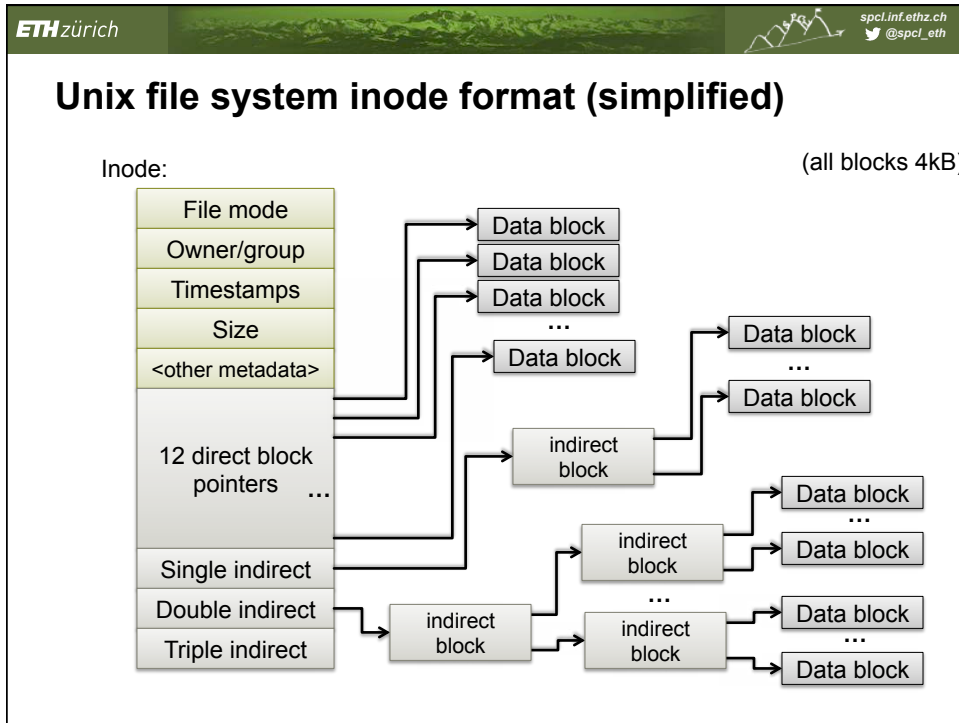
- **Example:**
 - Inode is 1 block = 4,096 bytes
 - Block addresses = 8 bytes
 - Inode metadata = 512 bytes
- **Hence:**
 - $(4,096 - 512) / 8 = 448$ block pointers
 - $448 * 4,096 = 1,792$ kB max. file size

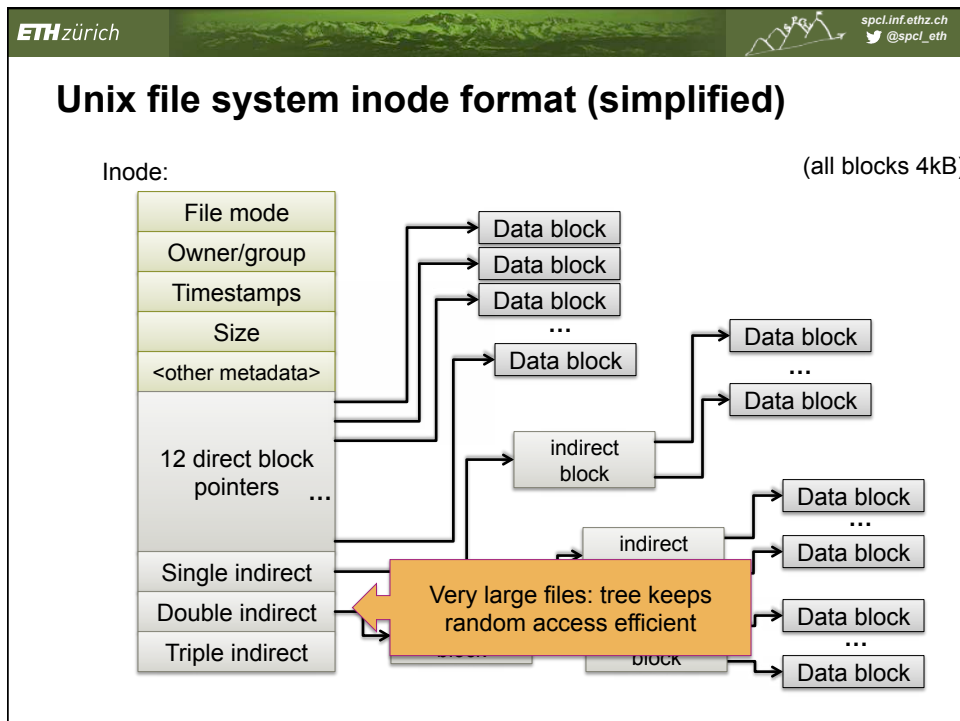
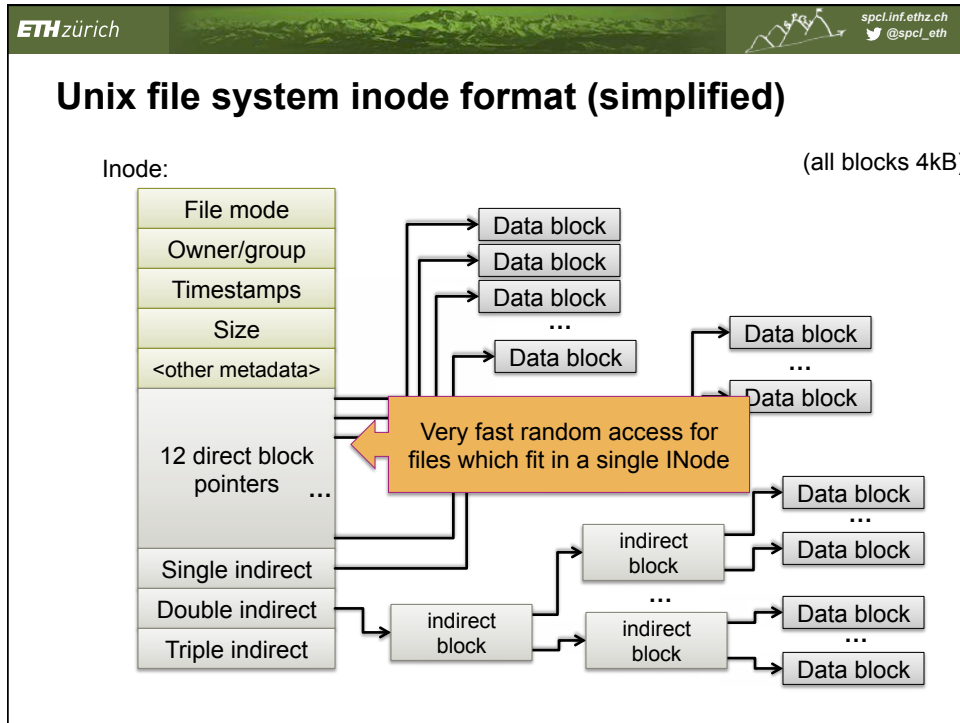












ETH zürich spci.inf.ethz.ch
@spci_eth

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 spci.inf.ethz.ch
@spci_eth

Block groups

1. Optimize disk performance by keeping together related:

- Files
- Metadata (inodes)
- Free space map
- Directories

The diagram illustrates three concentric circles representing Block Group 0, Block Group 1, and Block Group 2. Each group contains data blocks for files in specific directories and a free space bitmap. Block Group 0 contains data blocks for directories /a, /d, and /b/c. Block Group 1 contains data blocks for directories /b, /a/g, /z. Block Group 2 contains data blocks for directories /d/q, /r, /p, /s, /c. The diagram also shows inodes and free space bitmaps for each group.

ETH zürich spci.inf.ethz.ch
@spci_eth

Block groups

The diagram illustrates three concentric block groups, labeled Block Group 0 (outermost), Block Group 1 (middle), and Block Group 2 (innermost). Each block group contains data blocks for files in specific directories: Block Group 0 for /a, /d, and /b/c; Block Group 1 for /a, /d, and /b/c; and Block Group 2 for /a, /d, and /b/c. The diagram also shows 'Free Space Bitmap' and 'Indices' for each block group. A callout box in the upper right corner states: "2. Use *first-fit* allocation within a block group to improve disk locality".

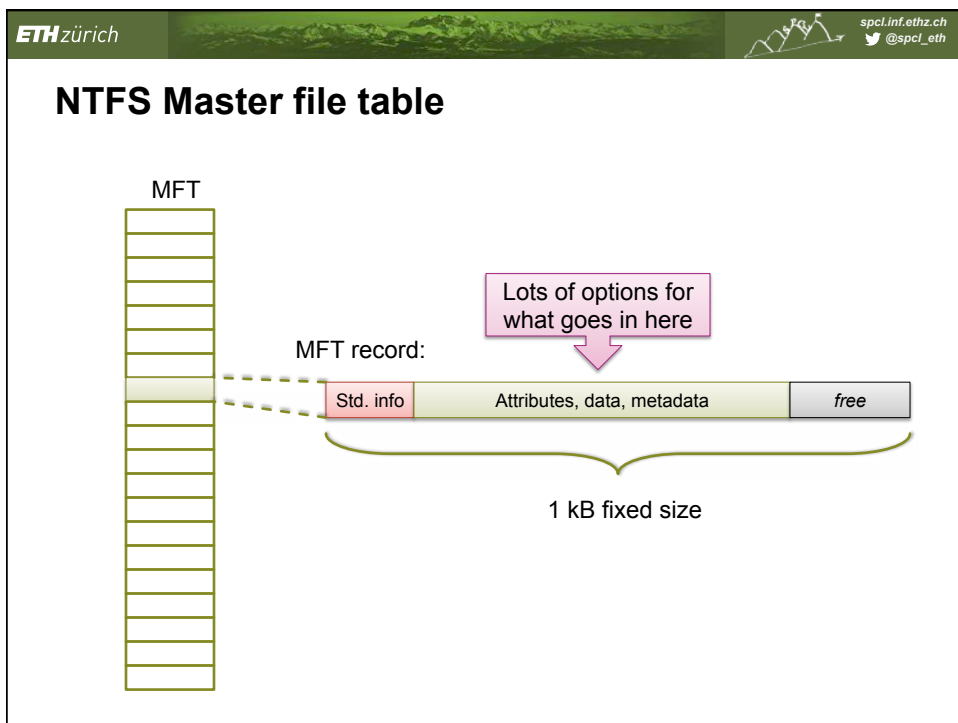
ETH zürich spci.inf.ethz.ch
@spci_eth

Block groups

The diagram illustrates three concentric block groups, labeled Block Group 0 (outermost), Block Group 1 (middle), and Block Group 2 (innermost). Each block group contains data blocks for files in specific directories: Block Group 0 for /a, /d, and /b/c; Block Group 1 for /a, /d, and /b/c; and Block Group 2 for /a, /d, and /b/c. The diagram also shows 'Free Space Bitmap' and 'Indices' for each block group. A callout box in the lower right corner states: "3. Layout and block groups defined in the *superblock* (not shown); Replicated several times."

ETH zürich spci.inf.ethz.ch
@spci_eth

NTFS



ETH zürich spci.inf.ethz.ch
@spci_eth

NTFS small files

- Small file fits into MFT record:

Std. info	Filename	File data	free
-----------	----------	-----------	------

"resident" data

ETH zürich spci.inf.ethz.ch
@spci_eth

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 spci.inf.ethz.ch
@spci_eth

NTFS normal files

- MFT holds list of *extents*:

ETH zürich spci.inf.ethz.ch
@spci_eth

Too many attributes?

- Attribute list holds list of attribute locations

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

ETH zürich spcl.inf.ethz.ch
@spcl_eth

Metadata files

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


File num.	Name	Description
0	\$MFT	Master file table
1	\$MFTirr	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 spcl.inf.ethz.ch
@spcl_eth

Metadata files

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


File num.	Name	Description
0	\$MFT	Master file table
1	\$MFTirr	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  spcl.inf.ethz.ch
@spcl_eth

Metadata files

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

File num.	Name	Description
0	\$MFT	Master file table
1	\$MFTirr	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  spcl.inf.ethz.ch
@spcl_eth

Metadata files

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

File num.	Name	Description
0	\$MFT	Master file table
1	\$MFTirr	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 spci.inf.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	\$MFTirr	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 spci.inf.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	\$MFTirr	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 spci.inf.ethz.ch
@spci_eth

In-memory data structures

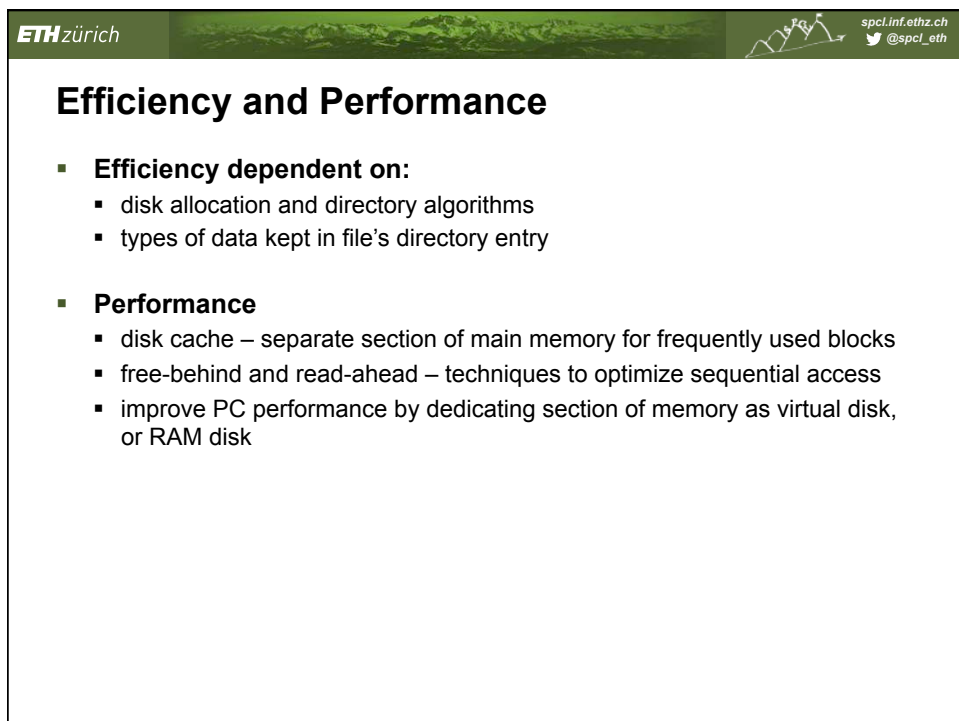
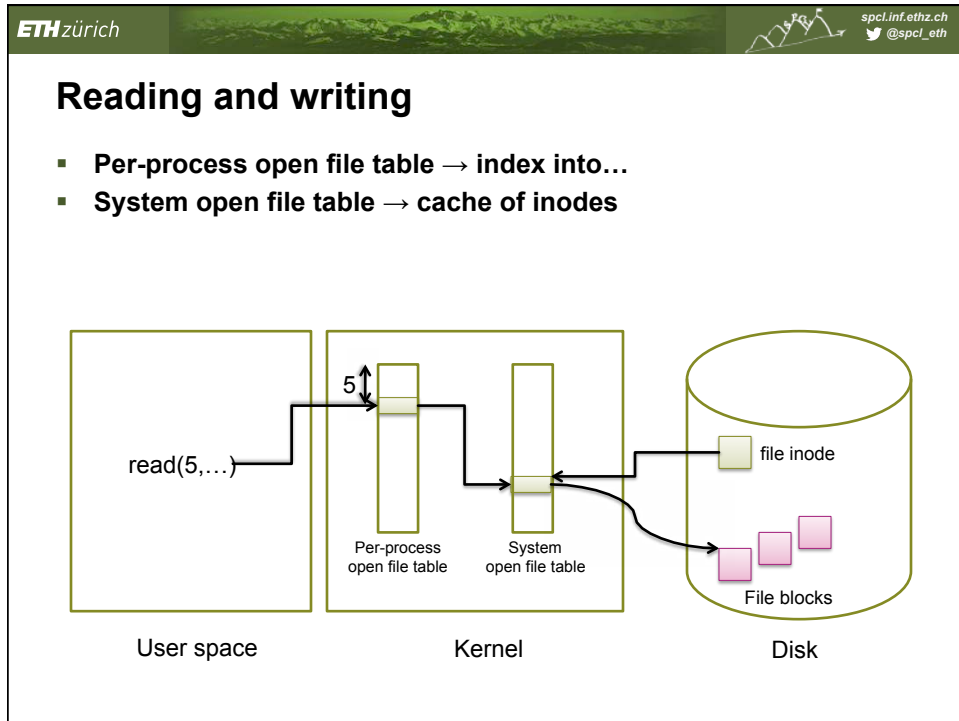
ETH zürich spci.inf.ethz.ch
@spci_eth

Opening a file

- Directories translated into kernel data structures on demand:

The diagram illustrates the process of opening a file across three environments: User space, Kernel, and Disk.

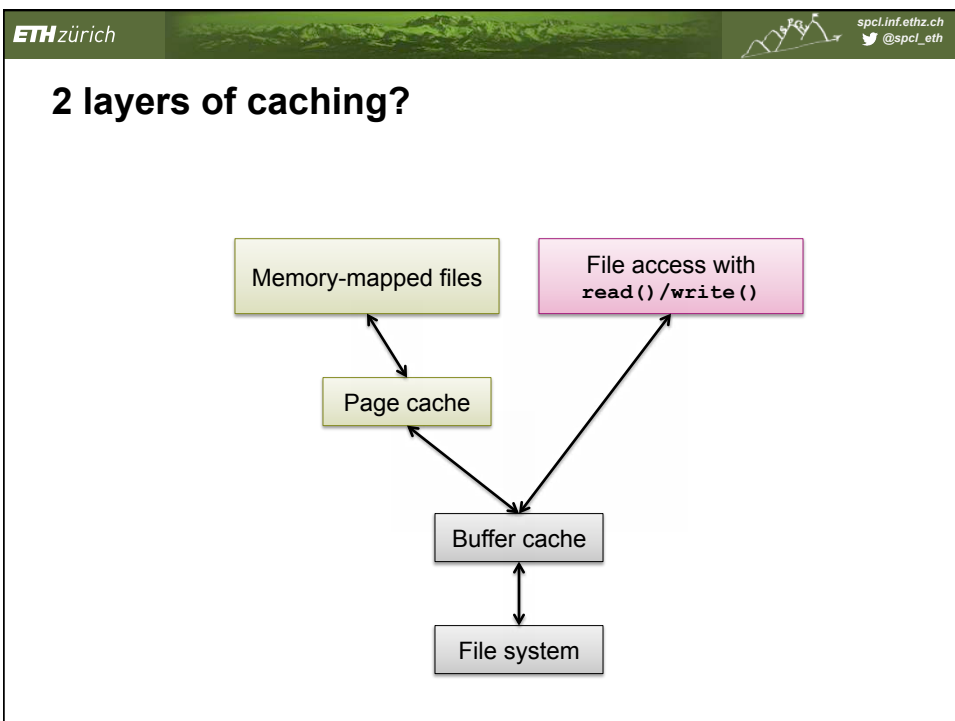
- User space:** A box containing the text `open("foo");`. An arrow points from this text to the Kernel.
- Kernel:** A box containing a `directory structure`. An arrow points from the User space to this structure. Inside the directory structure, a smaller box is highlighted in yellow. An arrow points from this highlighted box to the Disk.
- Disk:** A cylinder representing storage. It contains a `directory` (two pink boxes) and a `file inode` (a yellow box). An arrow points from the Kernel's directory structure to the Disk's directory. Another arrow points from the Disk's file inode back to the Kernel's directory structure.

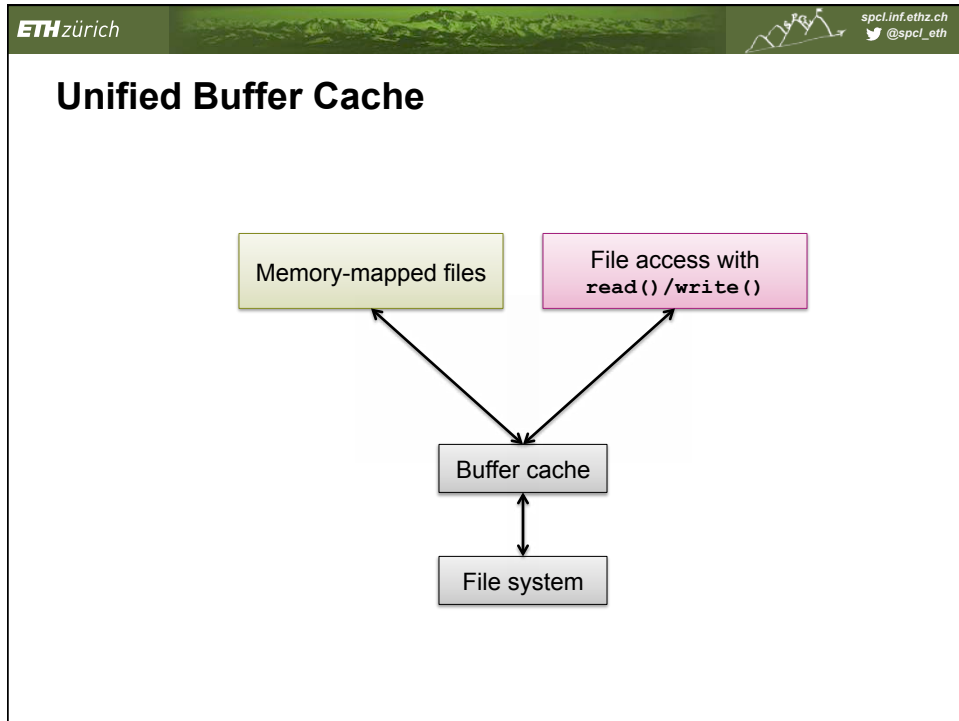


ETH zürich spci.inf.ethz.ch
@spci_eth

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





ETH zürich spci.inf.ethz.ch
@spci_eth

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**