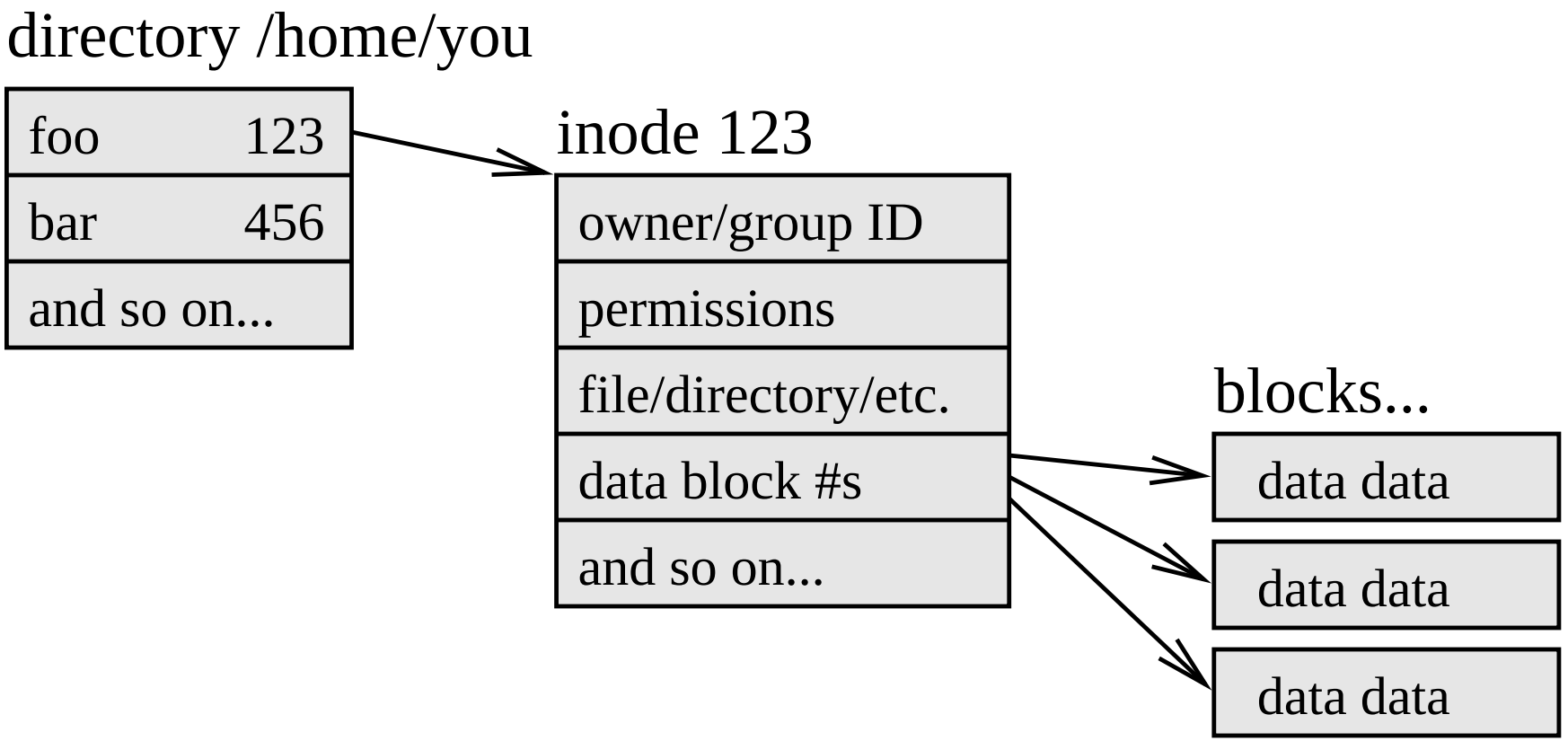
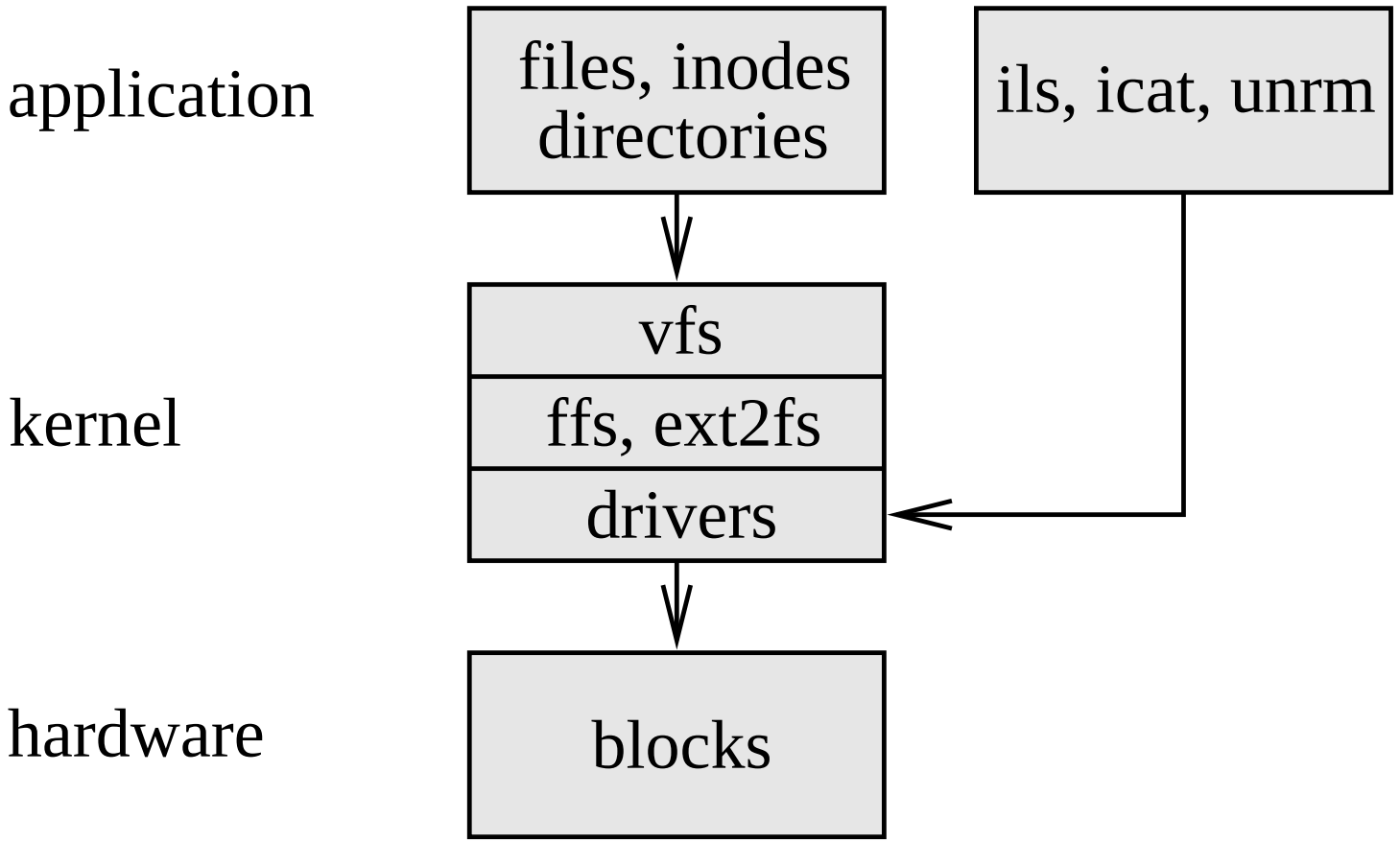
Recovering deleted files

The gory details

UNIX file system basics, recap



Bypassing the file system layer



Typical on-disk file system layout

Entire disk, not drawn to scale

- Disk label with disk partition layout
- Actual disk partitions

Partition or file system, not drawn to scale

- Superblock with file system layout (redundant copies)
- Bit maps for inode/data block allocation
- Actual inodes/data blocks

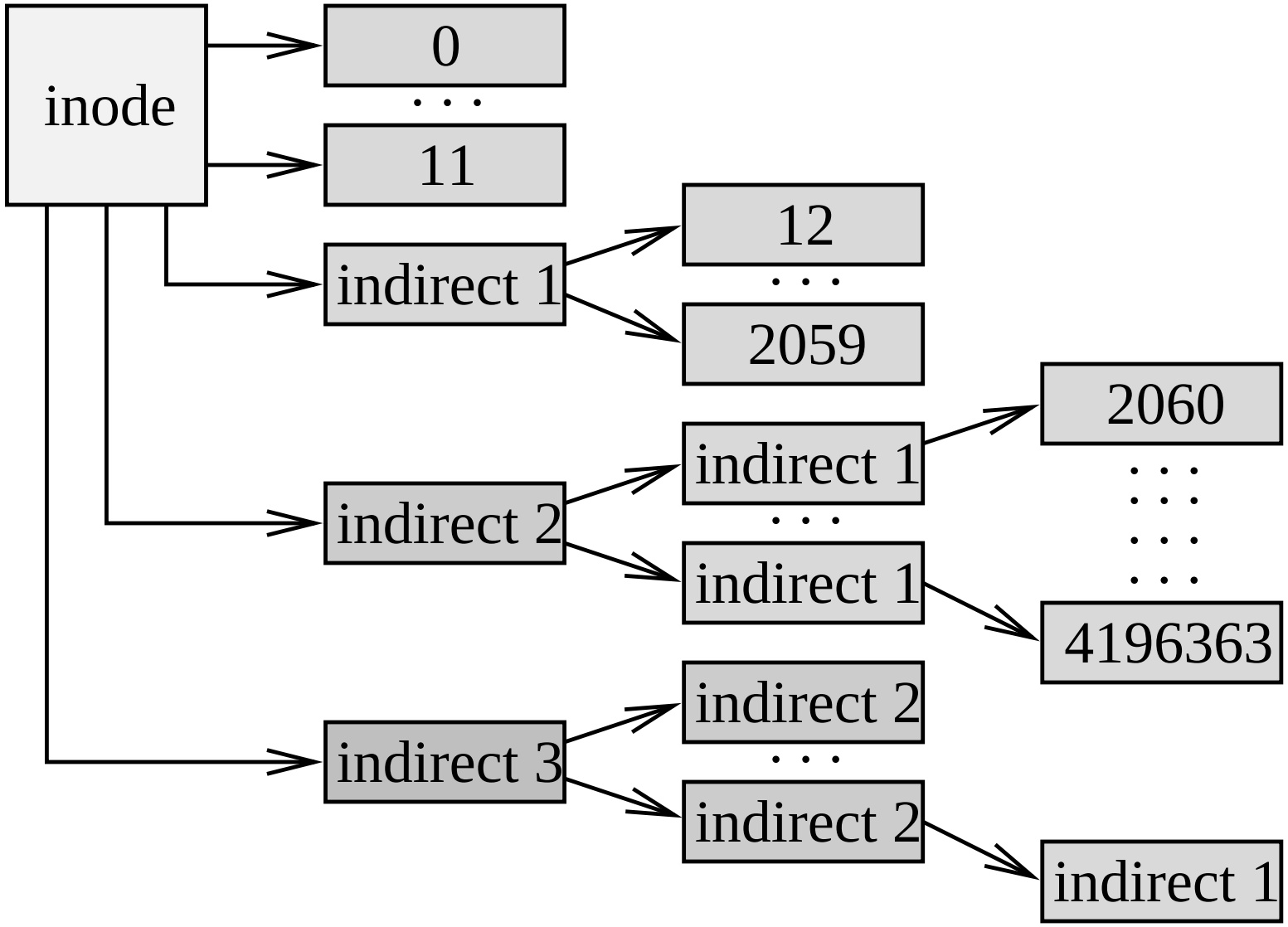
Inode information for removed files

- Ownership: numeric user and group ID
- Permissions: read/write/execute for owner, group, other
- Type: file, directory, symlink, device, FIFO, socket, etc.
- Time stamps:
  - last file Modification time
  - last file Access time
  - last status Change (e.g., owner, permissions, refcount)
- Reference count (0, 1, 2 etc.) - zeroed when removed
- File size in bytes - zeroed (except LINUX)
- List of data block numbers - zeroed (except LINUX)

ils, icat - file access by inode number

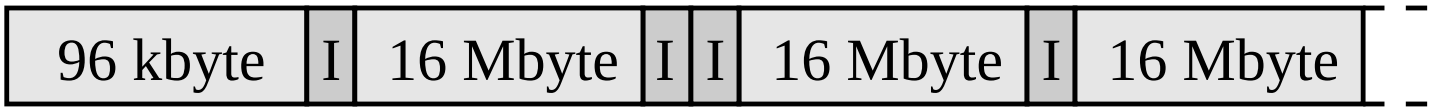
- List removed files (inode unallocated and/or refcount 0):
  `# ils device`
- List removed open files (inode allocated but refcount 0):
  `# ils -o device`
- Existing and removed files (inode allocated/unallocated):
  `# ils -l device`
- List specific inode(s):
  `# ils device inode...`
- Access file content by inode number:
  `# icat device inode >file`
- Part of the toolkit developed for this class

Direct and indirect blocks (FFS)

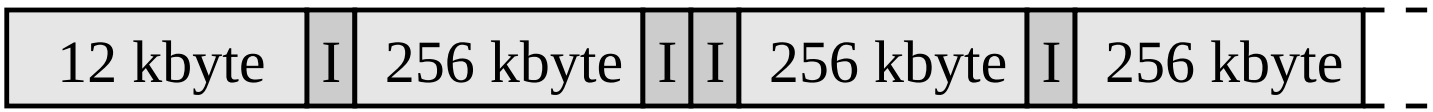


Sequential data allocation, ideal case

- UNIX Fast File System: variable block size 1..8 kbytes.



- LINUX ext2fs file system: fixed block size 1 kbytes.



- In reality, FFS spreads large files over clusters of blocks to avoid fragmentation of files and of free space. LINUX appears to simply allocate the next free 8 kbyte chunk.

unrm - file system dumpster diving

- Output has no indication of file boundaries!
- Output must be redirected to different file system or host!
- Use icat to exploit Linux removed inode data!
- Extract all removed data blocks

  `# unrm device`
- Extract removed data from a range of blocks

  `# unrm device first-last`
- Part of the toolkit developed for this class

Stashing data in the cracks of a UNIX system

Stashing by appending to files

- Exploit built-in file length information of image files executable files, etc.

```
% cat stuff >>executable-file
% cat stuff >>image-file
```

- Trivially easy to detect by comparing actual file size with built-in length information.

```
% check_exe executable-file
executable-file: 12345 bytes excess
```

Stashing by inserting comments

- Exploit ability to store comments inside executable or image files, etc.

```
% wrjpgcom -comment "‘cat stuff‘" file.jpg
% mcs -a "‘cat stuff‘" executable-file
```

- Detectable by looking for unusual comments (unusual length, unusual content, etc.).

```
% rdjpgcom filename | whatever
% mcs -p executable-file | whatever
```

- JPEG supports comment blocks up to 64 kbytes.

Stashing by inflating file segments

- Store data into the code or data segment of executable files.
- Detectable by analyzing the code segment and by proving that some code is unreachable.
- Detectable by analyzing the code segment and by proving that some data will never be touched.
- Left as an exercise to the reader. See the literature on the so-called "halting problem".

Stashing - wolf in sheep's clothes

- 3DES-encrypted data inside PGP header.
  Will resist brute force decryption attacks.

- PGP-encrypted data inside ZIP header.
  Result appears to be a corrupted ZIP file.

- Any sufficiently-obscure application-specific format.

- Mounting a file system on top of another one.

Stashing in left-over space

- Last data block of file (UNIX: 0.5 kbyte, MS: 10+kbytes)
- Padding of executable file segments (kbytes)
- Media bad block list (10+ kbytes)
- Disk partition boundaries (Mbytes)
- Unused disk partitions.
- Detection: this kind of space normally contains zeros or some trivial pattern.

Wiping data from a UNIX system

A really secure delete takes time

- It is possible to recover data from disk even after overwriting multiple times.
- It is possible to recover data from RAM even after powering off.
- Peter Gutmann, Secure Deletion of Data from Magnetic and Solid-State Memory, Sixth USENIX Security Symposium, San Jose, California, July 1996.

Steps to wipe a UNIX system

- Wipe files before removing them.
- Wipe free space.
- When shutting down the system:
  - Wipe swap space.
  - Wipe memory
- Wiping software: http://thc.pimmel.com/
  (see article on anonymizing UNIX systems)

Grafting to hide effects of wiping

- All-zero free blocks are unusual and could actually raise suspicion.
- Solution: overwrite free space with plausible data.
- Cloning/grafting: use copies of recently-accessed files from the system itself: mail, program source code, web pages/images, etc.