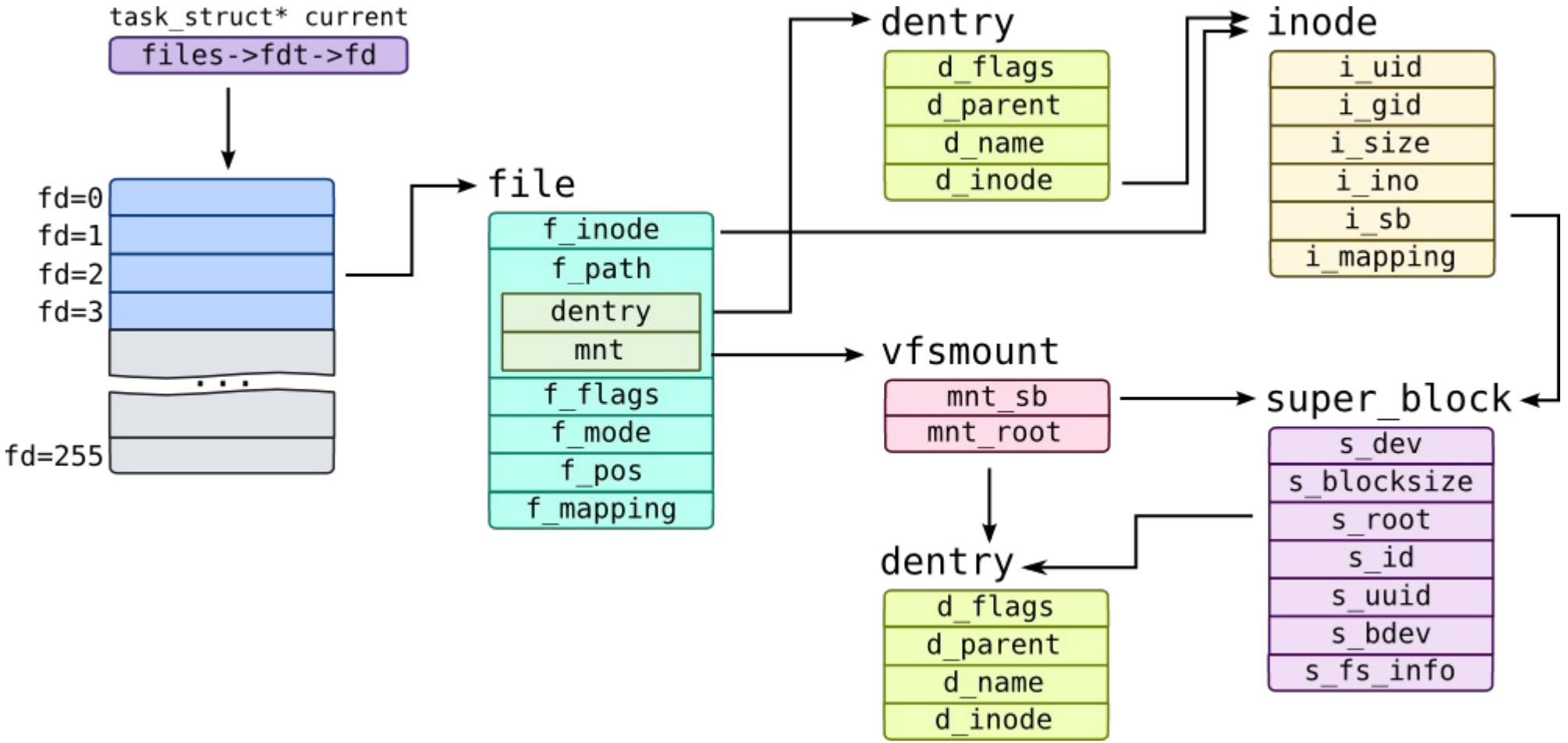
File systems

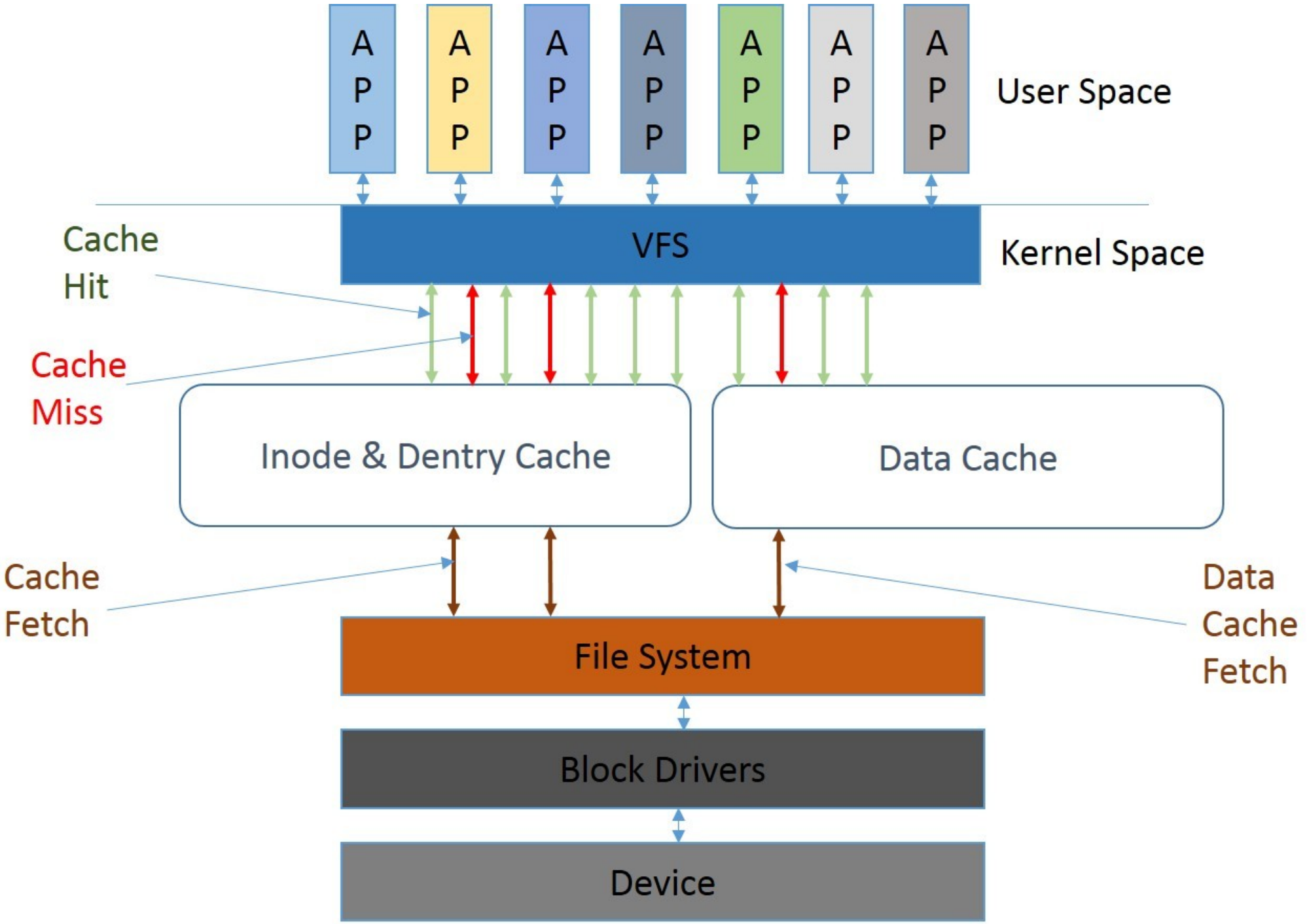
CSE 536 2024

jedimaestro@asu.edu



https://myaut.github.io/dtrace-stap-book/kernel/fs.html

```
man lsof
man stat
man dd
man df
man losteup
man lsblk
man mkfs.ext4
man mount
man rm
man strings
```



https://msreekan.com/tag/block-device/

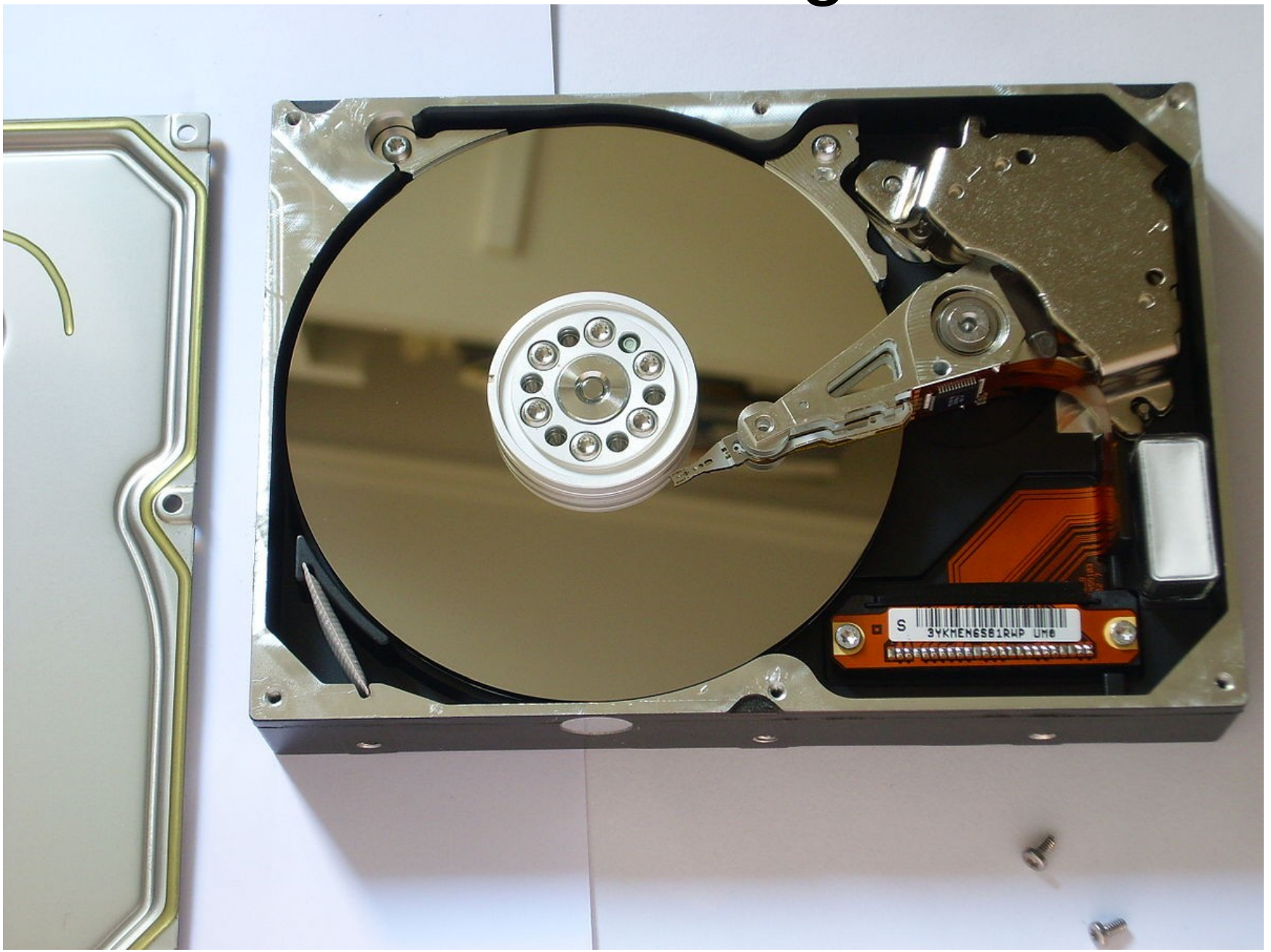
Digital forensics

- According to Wikipedia, you could be looking for: attribution, alibis and statements, intent, evaluation of source, document authentication
- File carving (*e.g.*, bifragment gap carving)
  - Electron microscopes
- Memory forensics (Volatility)
- Network forensics (PCAPs, NetFlow records, NIDS logs)
- Database forensics
- Timestamps in document or log file analysis
- Steganography
- Digital forensic processes
- Benford's law

Forensics tools

- File carvers
  - *E.g.*, Scalpel and foremost
- Log parsers
- Parsers/viewers for different kinds of files
  - SQLite, EXIF, *etc.*
- Linux commands that might be useful:
  - file, exif, sqlite3, losetup, mount, dd, ssdeep, grep, strings

File carving



Alessio Sbarbaro User_talk:Yoggysot - Own work

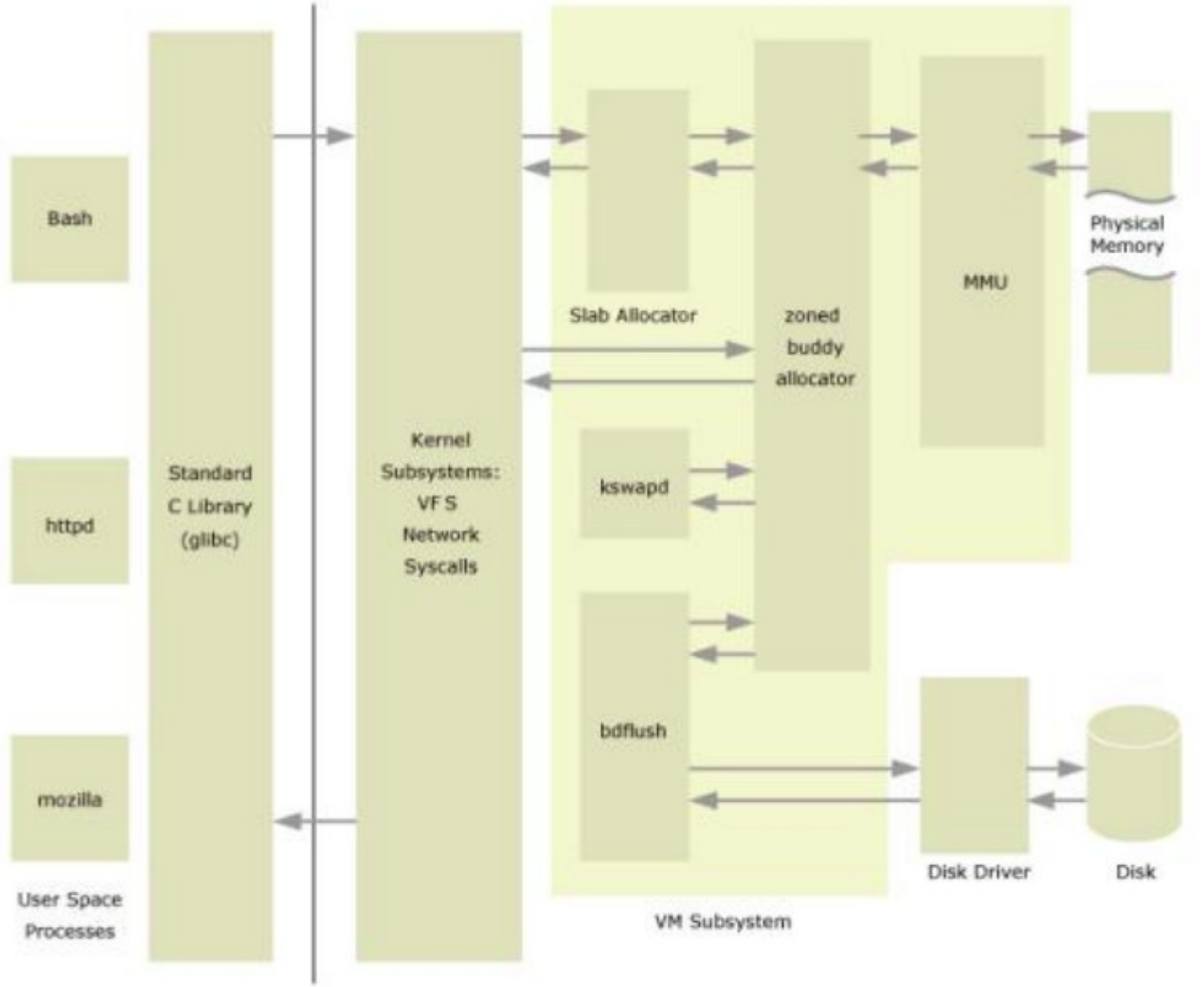
Memory forensics

```
$ python vol.py --profile=LinuxDebian-3_2x64 -f debian.lime linux_netstat
Proto   Source IP:Port          Destination IP:Port     State           Process
TCP     192.168.174.169:22      192.168.174.1:56705     ESTABLISHED     sshd/2787
TCP     0.0.0.0:22              0.0.0.0:0               LISTEN          sshd/2437
UDP     0.0.0.0:137             0.0.0.0:0               LISTEN          nmbd/2121
[snip]
```

man netstat

Some slides borrowed from…

https://students.mimuw.edu.pl/ZSO/Wyklady/06_memory2/BuddySlabAllocator.pdf



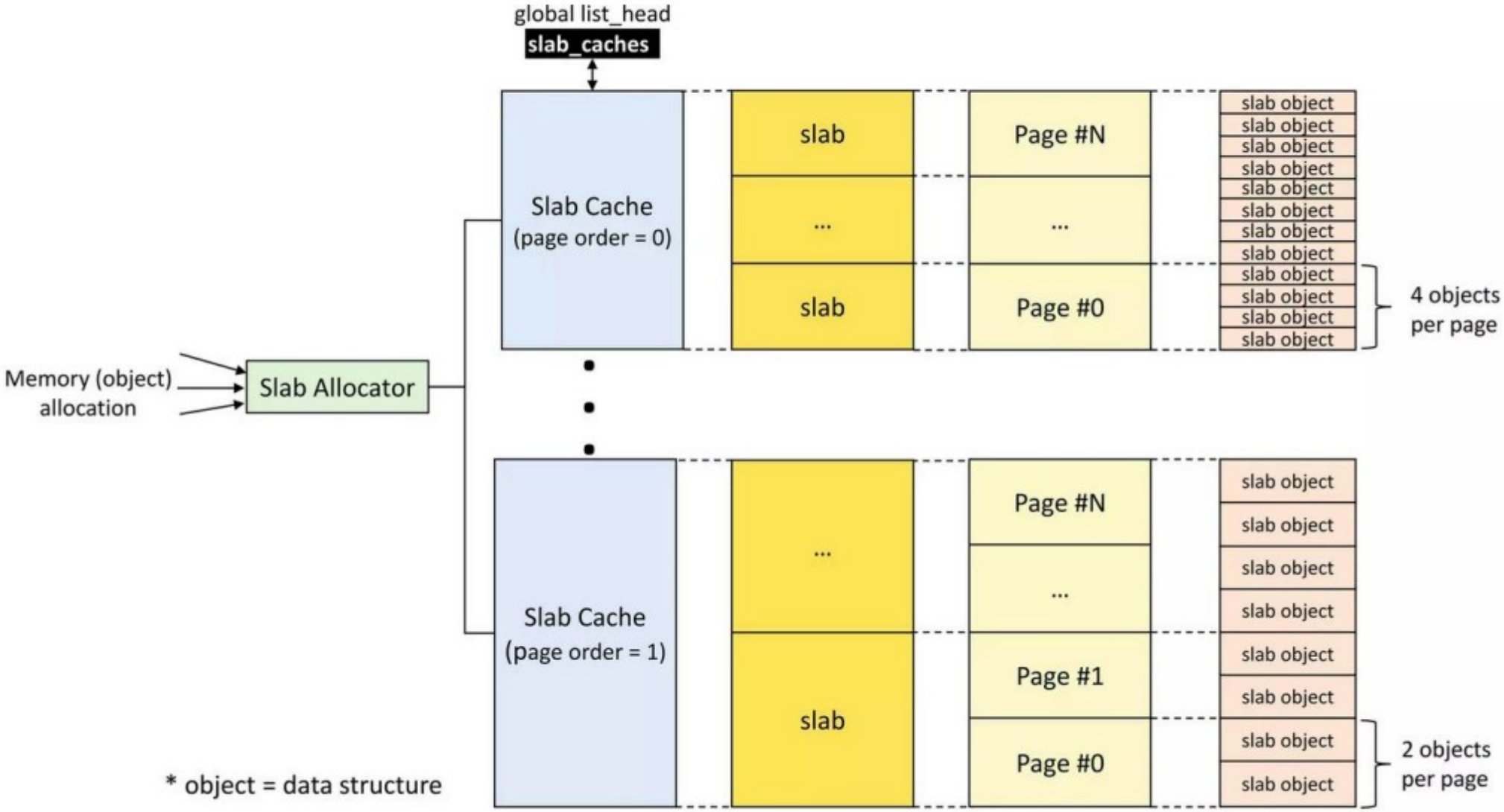
High overview of Virtual Memory subsystem
(source: N. Murray, N. Horman, Understanding Virtual Memory)

Buddy heap *vs.* slab allocator

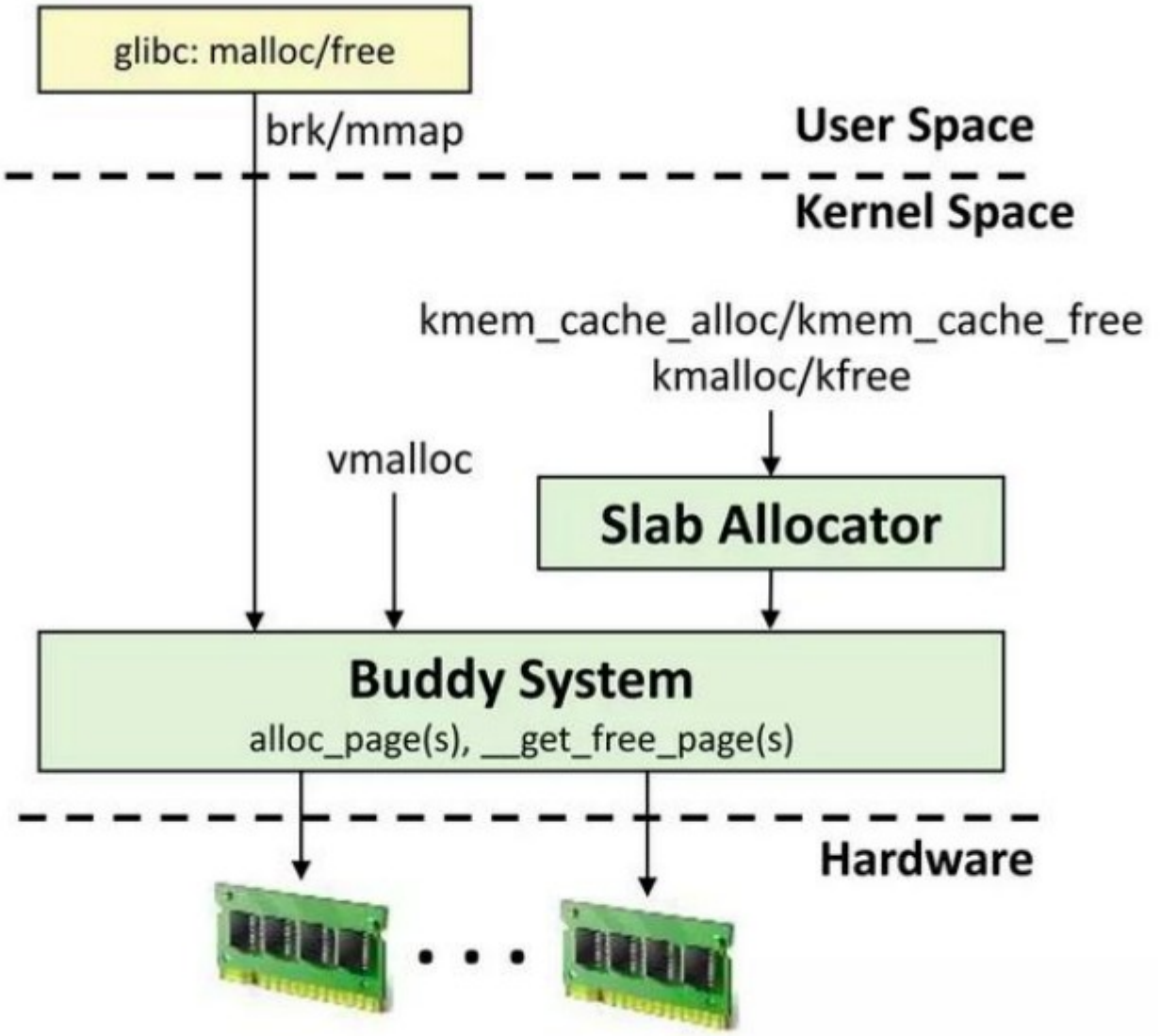
- Buddy heap
  - For grabbing 4KB pages at a time, contiguous
- Slab allocator
  - For grabbing 4KB pages at a time for common data structures

```
extern struct kmem_cache    *vm_area_cachep;
extern struct kmem_cache    *mm_cachep;
extern struct kmem_cache    *files_cachep;
extern struct kmem_cache    *fs_cachep;
extern struct kmem_cache    *sighand_cachep;
```

Slab allocator components



(source: Adrian Huang, Slab Allocator in Linux Kernel, 2022)



(source: Adrian Huang, Slab Allocator in Linux Kernel, 2022)

https://unix.stackexchange.com/questions/385323/is-the-size-of-inodes-fixed

So we *could* indeed make the inodes as large as the blocks, but in a real system this is probably not the case. In case you're wondering, the inode *structure* talks only about the pointers to the data blocks. So we have **156** bytes of pointers to the actual file contents, but the whole inode takes up 256 bytes - basically, we have 100 bytes we can use at our leisure, for whatever metadata we desire.

Treasure and tragedy in *kmem_cache* mining for live forensics investigation

Andrew Case[a], *Lodovico Marziale*[a], *Cris Neckar*[b], *Golden G. Richard, III*[c,*]

[a] *Digital Forensics Solutions, LLC, United States*
[b] *Neohapsis Inc., United States*
[c] *Dept. of Computer Science, University of New Orleans, Lakefront Campus, New Orleans, LA 70148, United States*

```
debian:~/slabwalk# insmod ./slabwalk.ko
debian:~/slabwalk# head -5 /var/log/messages
kernel: [35566.045181] inode: 106310 0 0
kernel: [35566.059469] inode: 106312 0 0
kernel: [35566.071471] inode: 139091 0 0
kernel: [35566.082007] inode: 106308 0 0
debian:~/slabwalk# ffind /dev/sda1 106310
/usr/share/zoneinfo/posix/America/Fortaleza/tmp
/cceZLcAc.o
debian:~/slabwalk# ffind /dev/sda1 139091
/var/run/sshd
debian:~/slabwalk# ffind /dev/sda1 106308
/usr/share/zoneinfo/posix/America/Fortaleza/tmp
/cceoInI5.o
```

Fig. 4 – Traversing the ext3 inode cache and using the Sleuthkit to obtain filenames.

been deleted since they were last used. Unfortunately, the normal algorithm for gathering the names of opened files, walking the list of the inode's directory entries, is not usable since the list is cleared on deallocation. Recovery of the names can still be achieved though with the help of the Sleuthkit, however, since the inode number is still accurate in the structure. Fig. 4 shows the results of traversing the free inode cache and then passing the inode number of free objects to *ffind* of the Sleuthkit, to perform a "dead" analysis of the inode in a filesystem image.

When testing this module, the ext3 filesystem was used and its inode cache, *ext3_inode_cachep*, was traversed. This cache actually contains *ext3_inode_info* structures, and these structures embed a regular inode structure. The *inode* structure allows the module to gather an inode's owner, group, mode, and inode number. Investigators can use this information to discover recently opened files, the permissions the file was opened with, and which user opened the file.

4.5. Socket buffers

4.6. Bound sockets

Before network servers can start accepting connections, they must *bind()* to a network port. In order to facilitate fast lookups of open and in-use ports, the Linux kernel tracks each in-use port within a quickly searchable data structure. For the TCP protocol, each port is kept within the *bind_bucket_cachep* of the *tcp_hashinfo* structure. This cache holds *inet_bind_bucket* structures whose member *port* contains the listening network port. This structure also contains a list of *sock* structures that reference the port, but unfortunately this list is cleared on deallocation. Enumeration of inactive entries of this cache will reveal ports that were previously used to accept network connections. In investigations where network capable malware was either present on the system or suspected to be present, this cache may be useful to help prove or disprove its existence. The port numbers can also be used to quickly point an investigator to interesting data within a network capture. Finally, combined with the network caches presented in the rest of this section, an investigator can gather a large amount of information related to past network activity solely from the machine under investigation. This will save investigative time normally spent examining server, IDS, and router logs.

4.7. Netfilter NAT table

The last cache explored, *nf_conntrack_cachep*, stores the Netfilter connection tracking information in *nf_conn* structures. Netfilter (netfilter.org) is the underlying framework for packet filtering in the Linux kernel, and the popular firewall technology, IPTables, is built upon it. In order to provide NAT capabilities, the connection tracking module must store the source and destination IP address and port for each translated connection. In the current Linux implementation, the incoming and outgoing translation for each connection is stored within the *tuplehash* member of each *nf_conn* structure.

Think about this while reading the reading assignment for Wednesday…

Suppose a process (Pegasus malware, VPN, Tor, etc.) wants to keep network activity from being logged to the hard drive. How hard is it to guarantee that?