APFS

No clever or witty subtitle.

Before we start.. If you want to follow along:

- Take the time to download:
  - http://technologeeks.com/tools/fsleuth (or fsleuth.linux for Linux)
  - Remove that stupid ".dms" extension (if using Safari)
    - (mv ~/Downloads/fsleuth.dms ~/Downloads/fsleuth)
  - chmod +x ~/Downloads/fsleuth
  - ~/Downloads/fsleuth

- Open a terminal command prompt
  - Because GUI is for wusses.

About this talk

- Just after this was announced, Apple *finally* released the spec..
  - (only took them two years)

- Nonetheless, the spec looks like Javadoc/doxygen, and is pretty vague
  - Not anything like TN1150 (HFS+)

- Research was reverse engineering, and spec filled in missing pieces

- Standing on the shoulders of giants:
  - APFS research of Kurt H. Hansen & Fergus Toolan (https://www.sciencedirect.com/science/article/pii/S1742287617301408)

APFS Features

The High Level View of APFS

APFS timeline

- New file system to replace venerable (15+ years) HFS+
  - Disappointed many who were expecting Apple to adopt ZFS
- Announced in 2016:
  - Initial MacOS 12 implementation was pretty bad:
    - Defined as "preview"
    - Full of incompatibilities with its own subsequent versions
    - No boot support (= EFI protocol)
  - Adopted first in iOS 10.3
    - iOS 11.3 moved to snapshot based mounts (more on this later)
  - Full adoption in MacOS 10.13
    - Still evolving in MacOS 14 (notably, supports defragmentation)

APFS features

- 64-bitness:
  - Support for ridiculous file sizes you'll never run into.
  - For-all-intents-and-purposes infinite number of files (2^{64} inodes)
  - Nanosecond-resolution timestamp since the Epoch (Jan 1st, 1970)
    - Y2K38 safe ☺

APFS features

- Built in volume management
  - R.I.P CoreStorage* and iOS's LwVM
  - Partition is now formatted as "Container"
  - Individual mountable filesystems are "Volumes"
  - All volumes share same container

Filesystem	Size	Used	Avail	Capacity	iused	ifree	%iused	Mounted on
/dev/disk1s1	**466Gi**	**399Gi**	**63Gi**	**87%**	**1753922**	**9223372036853021885**	**0%**	**/**
devfs	221Ki	221Ki	0Bi	100%	764	0	100%	/dev
/dev/disk1s4	**466Gi**	**3.0Gi**	**63Gi**	**5%**	**4**	**9223372036854775803**	**0%**	**/private/var/vm**
map -hosts	0Bi	0Bi	0Bi	100%	0	0	100%	/net
map auto_home	0Bi	0Bi	0Bi	100%	0	0	100%	/home

* - Goodbye, and Good Riddance!

APFS features

- Fast Directory Sizing
  - Directory totals are saved along with the directory’s own inode
  - Allows for faster applications of du(1) and of Finder’s Get Info
- Sparse file support
  - Large files with vast swaths of zero’ed out data
  - Using extents file system can store only actual data, working around “holes”

APFS features

- Cloning:
  - Rather than copy a file, maintain another reference to it
  - Any changes are stored as subsequent deltas
  - Proprietary system call `clonefileat(2)` (#462), pretty well documented

```
CLONEFILE(2)                BSD System Calls Manual                CLONEFILE(2)

NAME
     clonefile -- create copy on write clones of files

SYNOPSIS
     #include <sys/attr.h>
     #include <sys/clonefile.h>

     int
     clonefile(const char * src, const char * dst, int flags);

     clonefileat(int src_dirfd, const char * src, int dst_dirfd, const char * dst, int flags);

     fclonefileat(int srcfd, int dst_dirfd, const char * dst, int flags);

DESCRIPTION
     The clonefile() function causes the named file src to be cloned to the named file dst.  The cloned file dst shares its data blocks
     with the src file but has its own copy of attributes, extended attributes and ACL's which are identical to those of the named file
     src with the exceptions listed below

     1.   ownership information is set as it would be if dst was created by openat(2) or mkdirat(2) or symlinkat(2) if the current user
          does not have privileges to change ownership. If the optional flag CLONE_NOOWNERCOPY is passed, the ownership information is
          the same as if the the current user does not have privileges to change ownership

     2.   setuid and setgid bits are turned off in the mode bits for regular files.
```

APFS features

- Copy-on-Write
  - Contrary to other file systems, changes do not get written into same block
  - APFS is a Copy-on-Write filesystem
  - This makes APFS especially Flash Friendly (avoids P/E cycles wear)
  - Ensures much better resiliency in the face of possible crashes
  - Also makes APFS a forensic analyst’s dream
  - Surprisingly, though – no undelete functionality provided by Apple

APFS features

- Snapshots:
  - Similar to well-known (and darn useful) virtual machine snapshots
  - Used by Time Machine, through the `tmutil(8)` command-line

```
localsnapshot
        Create new local Time Machine snapshots of all APFS volumes included in the Time Machine backup.

listlocalsnapshots mount_point
        List local Time Machine snapshots of the specified volume.

listlocalsnapshotdates [mount_point]
        List the creation dates of all local Time Machine snapshots.

        Specify mount_point to list snapshot creation dates from a specific volume.

        Listed dates are formatted YYYY-MM-DD-HHMMSS.

deletelocalsnapshots date
        Delete all local Time Machine snapshots for the specified date (formatted YYYY-MM-DD-HHMMSS).

thinlocalsnapshots mount_point [purge_amount] [urgency]
        Thin local Time Machine snapshots for the specified volume.

        When purge_amount and urgency are specified, tmutil will attempt (with urgency level 1-4) to reclaim purge_amount in bytes by thinning snapshots.

        If urgency is not specified, the default urgency will be used.
```

  - Maintained by `fs_snapshot(2)` system call

```
FS_SNAPSHOT_CREATE(2)       BSD System Calls Manual       FS_SNAPSHOT_CREATE(2)

NAME
     fs_snapshot_create -- create read only snapshot of a mounted filesystem

SYNOPSIS
     #include <sys/attr.h>
     #include <sys/snapshot.h>

     int
     fs_snapshot_create(int dirfd, const char * name, uint32_t flags);

     int
     fs_snapshot_delete(int dirfd, const char * name, uint32_t flags);

     int
     fs_snapshot_list(int dirfd, struct attrlist * name, void * attrbuf, size_t bufsize, uint32_t flags);

     int
     fs_snapshot_rename(int dirfd, const char * old, const char * new, uint32_t flags);

     int
     fs_snapshot_mount(int dirfd, const char * dir, const char * snapshot, uint32_t flags);

     int
     fs_snapshot_revert(int dirfd, const char * name, uint32_t flags);

DESCRIPTION
     The fs_snapshot_create() function, for supported Filesystems, causes a snapshot  of the Filesystem to be created. A snapshot is a read only copy of the filesystem frozen at
     a point in time.  The Filesystem is identified by the dirfd parameter which should be a file descriptor associated with the root directory of the filesystem for which the
     snapshot is to be created.  name can be any valid name for a component name (except . and ..).  The fs_snapshot_delete() function causes the named snapshot name to be
     deleted and the fs_snapshot_rename() function causes the named snapshot old to be renamed to the name new.  Available snapshots along with their attributes can be listed by
     calling fs_snapshot_list() which is to be used in exactly the same way as getattrlistbulk(2).  The flags parameter specifies the options that can be passed. No options are
     currently defined.
```

APFS features

- Encryption
  - APFS Fuses two of Apple’s strongest encryptions:
    - FileVault (“Full Disk Encryption”)
      - Required to mount the volume
      - Remains in memory for lifetime of mount
      - Hardware accelerated on iOS and Macs with new T2 chip that’s popping up everywhere
    - NSFileProtectionClass (“Per File/Class Encryption”)
      - Required to access a file
      - One of four* protection classes
      - D: Available C: Until First Unlock B: unless open A: unless unlocked

* - Technically, five, but I’m ignoring class F here

APFS features

- Additional features (inherited from VFS) are:
  - Extended Attributes
    - Arbitrary key/value combinations, viewable through `ls -@`
  - Transparent File Compression
    - chattr(1) compressed, `ls –O`
    - Files compression metadata is in (invisible) com.apple.decmpfs extended attribute
    - Small files compressed directly into attribute value; larger files compressed on disk
  - Resource forks
    - com.apple.ResourceFork extended attribute (`ls -@`)
    - Also accessible through *filename*`/../namefork/rsrc` (yes, seriously)
    - Ensures compatibility with MacintoshFS, from 20 years ago*

* - Also, great way to hide data, if you're malware..

Apple's APFS tools

Binary	Purpose
apfsd(8)	APFS Volume Management Daemon. Invoked automatically to maintain mounted volumes.
apfs.util(8)	Extremely limited APFS file system utility
apfs_condenser	MacOS 14 – shrink/defrag containers (won't even output command line arguments)
apfs_invert	Apparently inverts container and volume (not brave enough to try this yet)
apfs_stats	Gets human readable statistics for IORegistry. Invoked by sysdiagnose(8)
fsck_apfs(8)	APFS file system checker; Invoked automatically when fsck(8) detects APFS
hfs_convert(8)	Converts HFS+ volumes to APFS
`mount_apfs(8)`	APFS file system mounter; Invoked with –t apfs (or when APFS is detected)
`newfs_apfs(8)`	Format a block device to create an APFS container and/or add volumes to an existing one
`slurpAPFSMeta`	Dumps APFS metadata from an APFS volume. Useful for debugging..

But how does it really work?

- Don’t ask. You don’t need to know.
- It’s the best file system. Ever*.
- It Just Works.™

* - ZFS advocates might disagree. But they’re just BSD-folk. This is Darwin. The very name of the OS shows how evolved it is.

Let’s get technical

The Low Level view of APFS

Ignorance was bliss. You might want to space out/Insta-Message-Snap-Post instead at this point

General file system nomenclature

Term	Meaning
Block	Atomic unit of disk space. Usually 512-8,192 bytes. APFS uses 4,096
Extent	Sub unit of a block, used when files are smaller than a block size so as to save space
File	A mapping of a logical name to a set of blocks and/or extents
Contiguity	A File (or free space) spanning sequential blocks. May impact (non-SSD) disk I/O performance
Fragmentation	Unallocated/freed blocks in non-contiguous chunks arising over time from file creation/deletion
SuperBlock	A special block on disk, usually at fixed location(s), providing file system metadata
Inode	Index node – metadata (block allocation, permissions, unique identifier) of file in file system.
`fsck(8)`	A command you don't want to find yourself executing.

A good file system must provide an optimal allocation of blocks (= less wasted space as possible), ensuring maximum contiguity (= minimal fragmentation), reliability, and recoverability, while minimizing I/O overhead.

APFS file system blocks

- A given block in an APFS file system may be:
  - **Free:** contents may be zeroed out, or left over from previous generation
  - **File data:** contents may be fragment of some file data stream
  - **APFS object:** One of specific types used by APFS for its metadata.

- APFS objects are easily recognizable by a Fletcher 64 checksum
  - If checksum is valid, it’s an object
  - If checksum is not valid, likely some stream fragment (or corrupt anyway)
  - Caveat: Zero and all 0xFF blocks (which aren’t valid objects)

APFS Objects

- All object nodes start with a 32-byte header:

64-bit ID indexed by the object map

Fast checksum, must be valid for block to be considered

Fletcher checksum	Object id (oid)		
Transaction id (xid)	blockType	flags	blockSubType

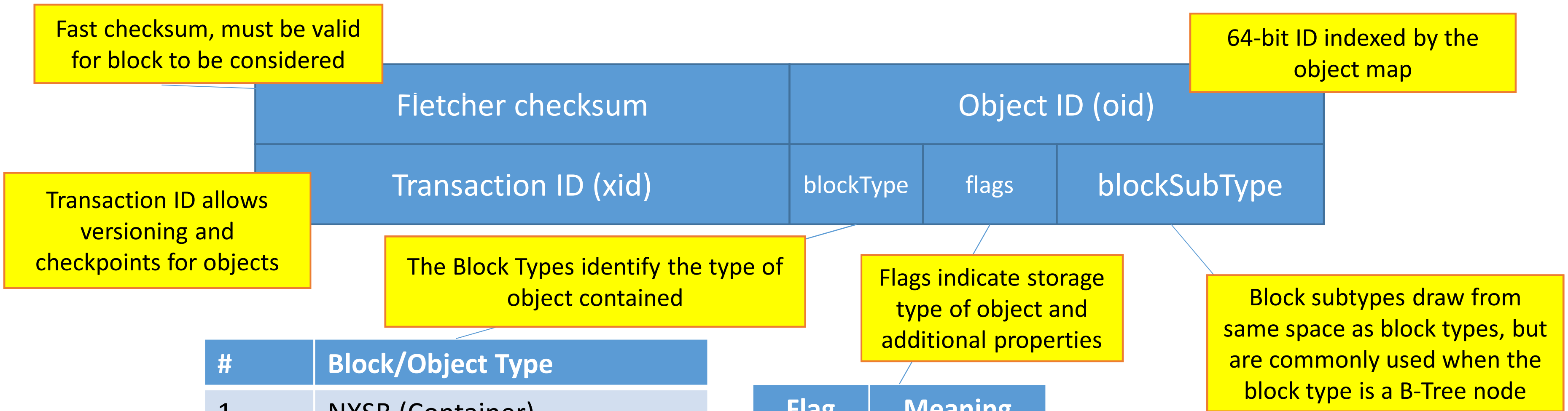
Allows versioning and checkpoints for objects

#	
1	NXSB (Container)
2	B-Tree root node
3	B-Tree non-root node
12	Object Map
13	APSB (Volume)

Flags indicate storage method of object

#	
0x0	Virtual
0x80..	Ephemeral
0x40..	Physical

Some 26 object types presently defined – these are the common ones



#	Block/Object Type
1	NXSB (Container)
2/3	B-Tree root/non-root node
5-9	Space Manager objects
11	Object Map
12	Checkpoint Map
13	APSB (Volume)
17/18	Reaper/Reap List
20	EFI Jumpstart (boot info)
22-23	Fusion Write Back Cache
24	Encryption Rolling Info
25,27	General Bitmap Tree/Block

Flag	Meaning
0	Virtual
0x8000	Ephemeral
0x4000	Physical
0x2000	No header
0x1000	Encrypted
0x0800	Transient

#	Block/Object Sub Type
10	Extent List Tree
11	Object Map
14	File System Tree
15	Block Reference Tree
16	Snapshot Metadata Tree
19	Object Map Snapshot
21	Fusion Middle Trees
26	General Bitmap Tree

APFS Objects

- Objects can be stored by one of three methods:
  - Physical objects are stored at a physical 64-bit block address
  - Ephemeral objects are stored on disk, but change during mount
  - Virtual objects may “move about” disk and address needs to be looked up

- An object map is used to look up physical addresses of virtual objects
  - Object map is a B-Tree
  - Container Object Map for global (container-scope) objects
  - Per-Volume Object Map for local (volume-scope) objects

To B or not to B(-Tree)

- B-Trees are fundamental data structures in modern file systems
- Used by HFS+, and unsurprisingly also in APFS (similar node format)
  - Allows for quick conversion of apfs_hfs_convert
- Enable efficient lookup of nodes in logarithmic time – $O(\log_b(n))$
  - 100 files – O(7) operations (for b=2)
  - 1,000,000 files – O (20) operations (for b=2)
  - 1,000,000,000 files – O(30) operations (for b=2)
  - In practice b is higher than 2 (e.g. 5), making operations even more efficient.

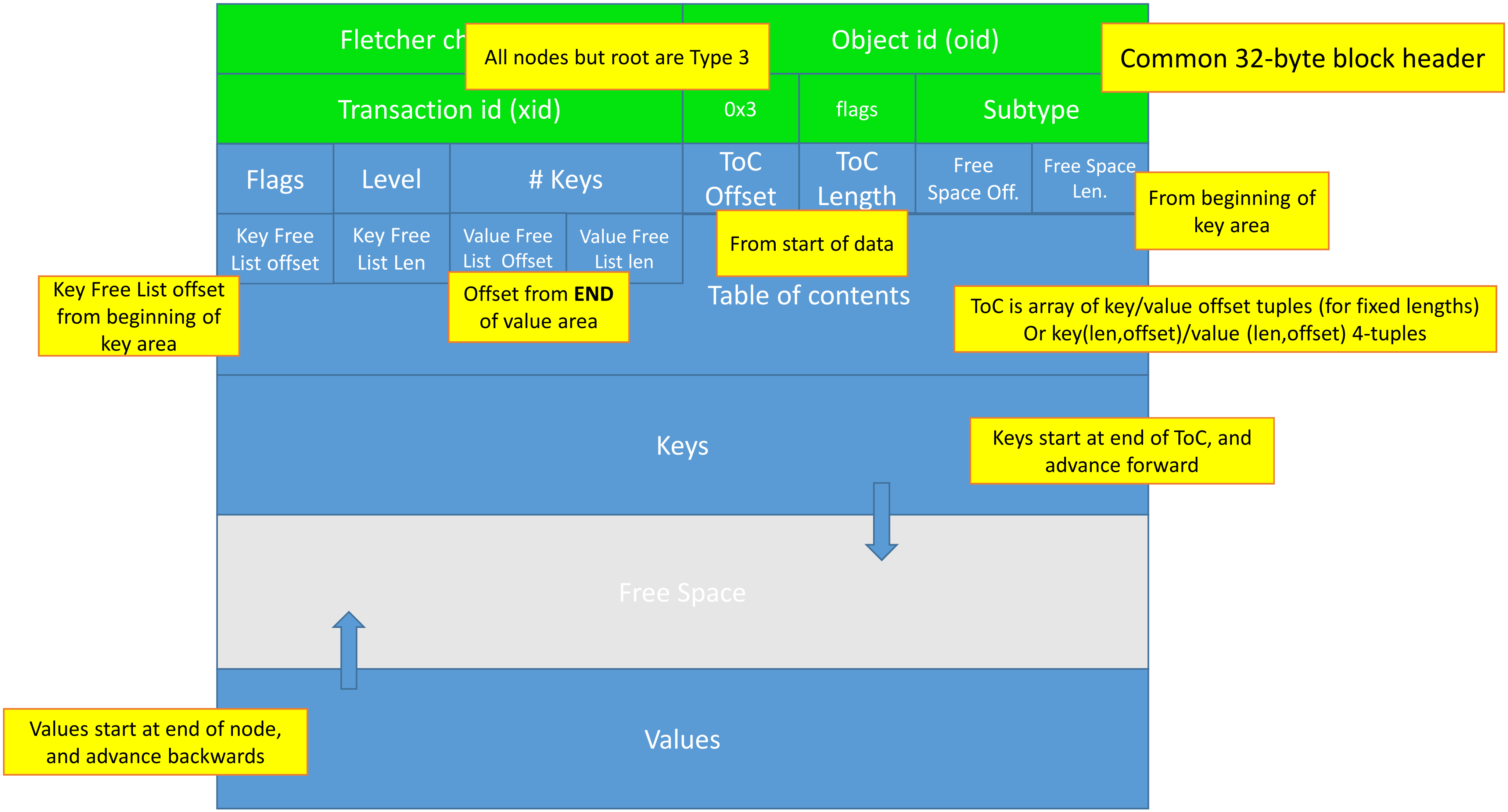
Don’t just B. B+

- APFS B-Tree are specific types called B+ Trees, which satisfy:
  - Every node can have a large number of children
  - Internal nodes index the smallest keys in their children
  - Insertion, deletion and search are all $O(\log_b n)$
  - Caveat: APFS implementation tree are not sibling linked.

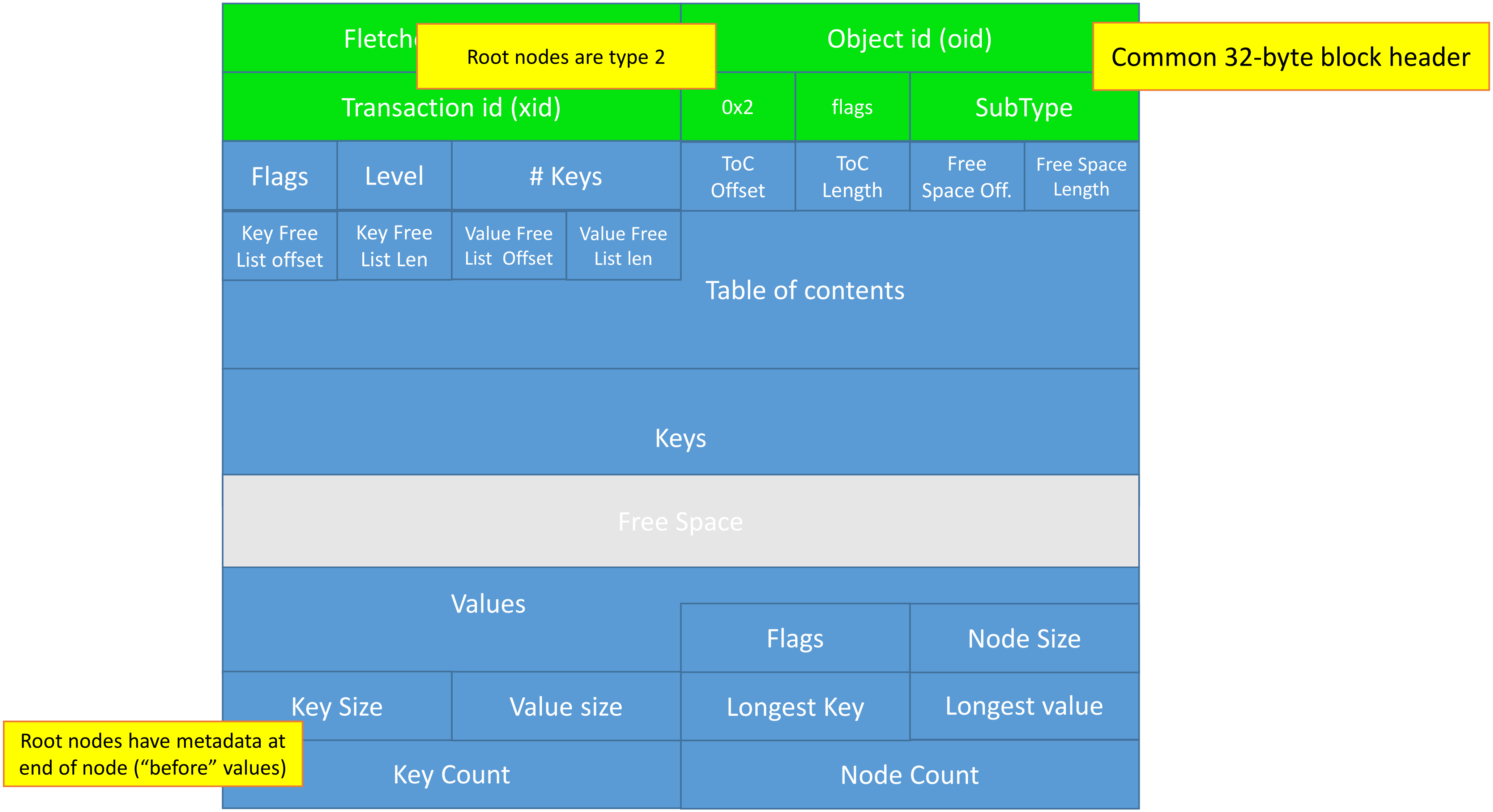
B-Tree Nodes

- B-Tree node format bears some similarities to that of HFS+
  - Because A) it works and B) it makes for really fast conversion
- Nodes are of block type “2” (root) or “3” (non-root) nodes
  - Contain fixed size header
  - Contain a “table of contents” (ToC) indicating keys, values and free space
  - Keys start in sequential order after ToC
  - Values start at end of block, reverse sequential order
  - Free space is in middle, fragmentation eventually managed by a free list
  - Root nodes also have a small trailer information blob

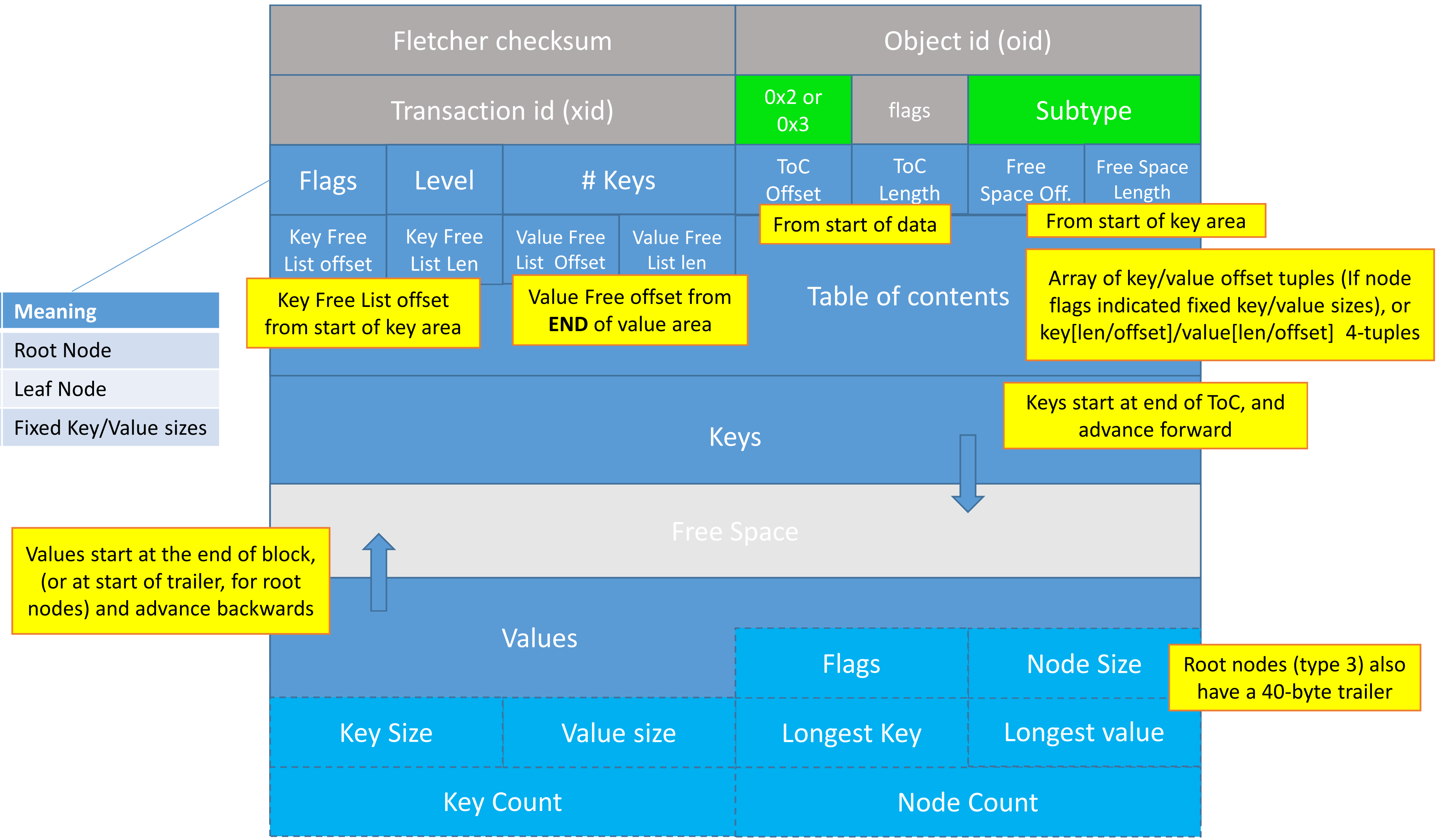
The APFS B-Tree Leaf/Middle Node



The APFS B-Tree Root Node



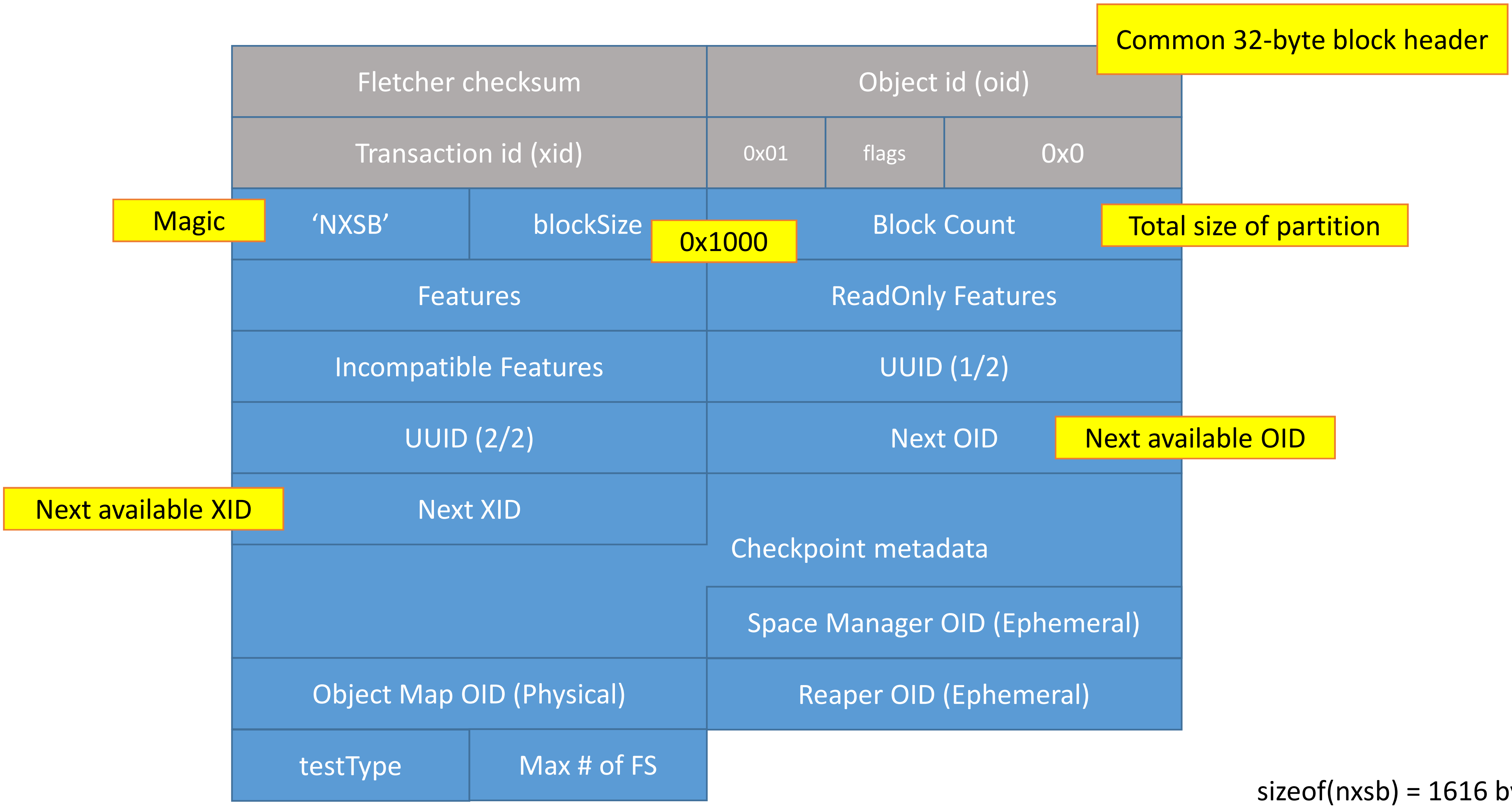
Flag	Meaning
0x1	Root Node
0x2	Leaf Node
0x4	Fixed Key/Value sizes



APFS Containers

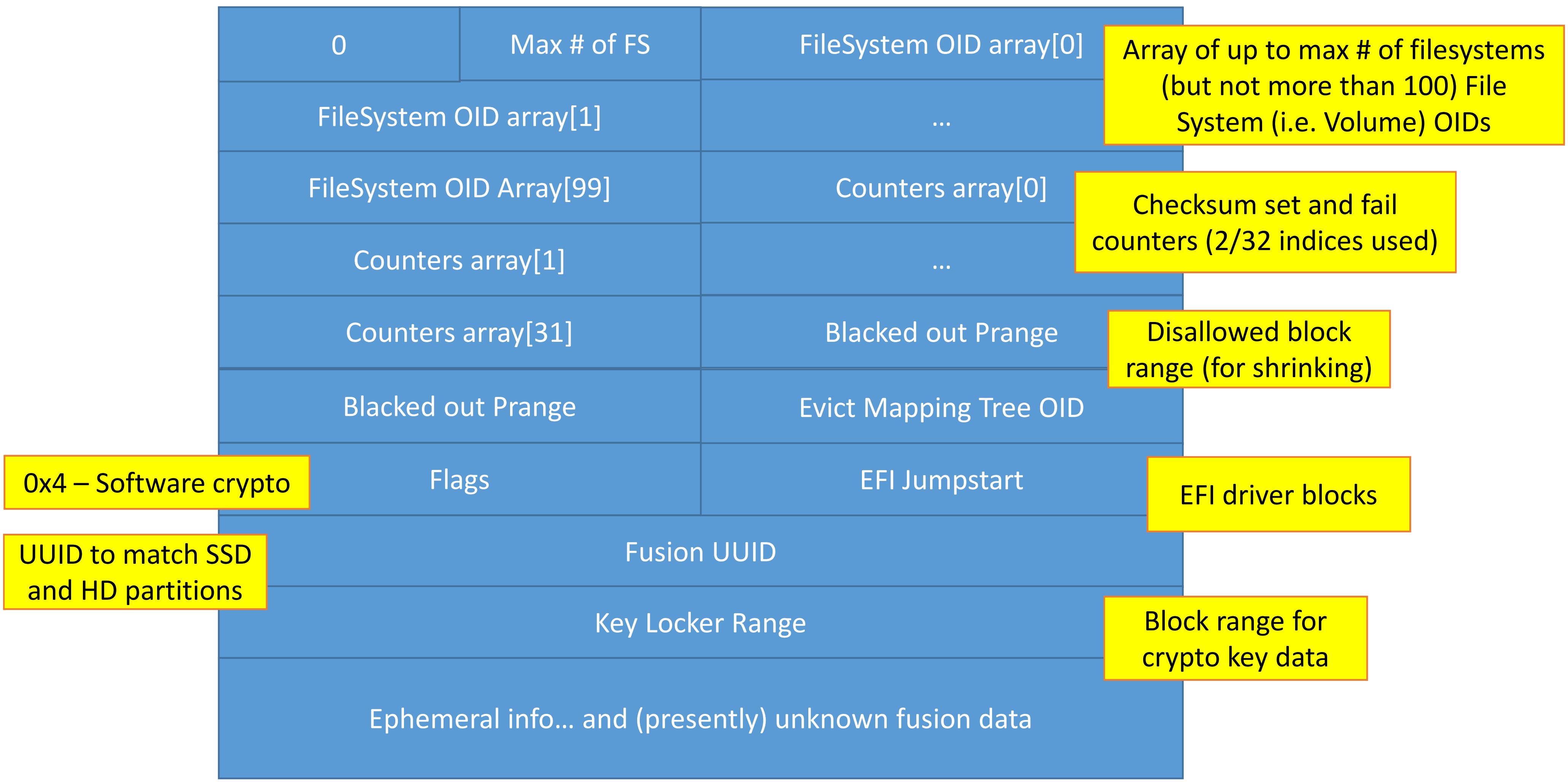
- The container (“nx”) is the top level object of the partitioned space
  - Contains one or more volumes (“apfs”)
  - Effectively acts as a logical volume manager
    - All volumes see and expand into the same free space
    - Single Space Manager (“spaceman”) handles block allocation
  - Container holds global object map

The APFS Container Superblock (NXSB)



sizeof(nxsb) = 1616 bytes

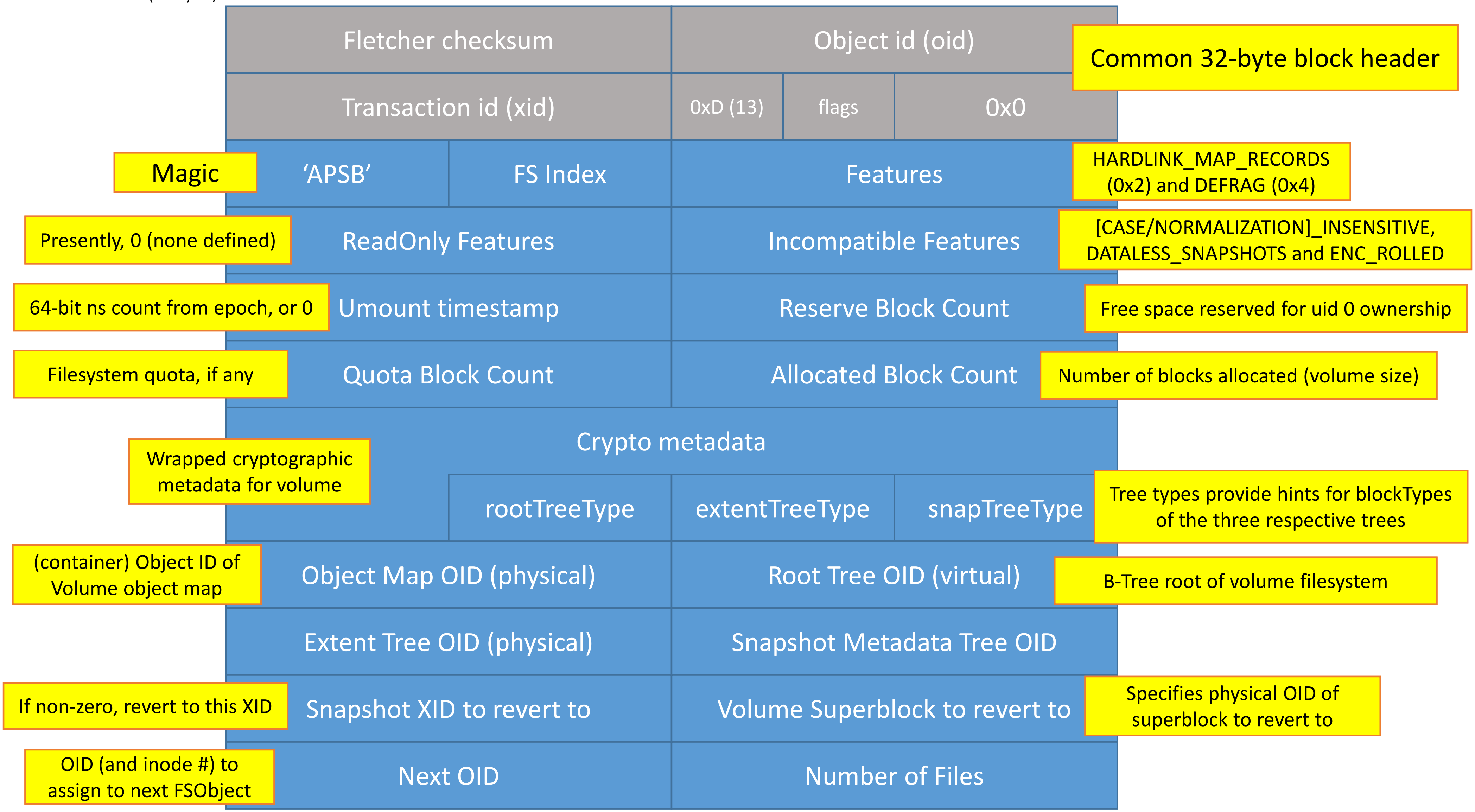
The APFS Container Superblock (NXSB)



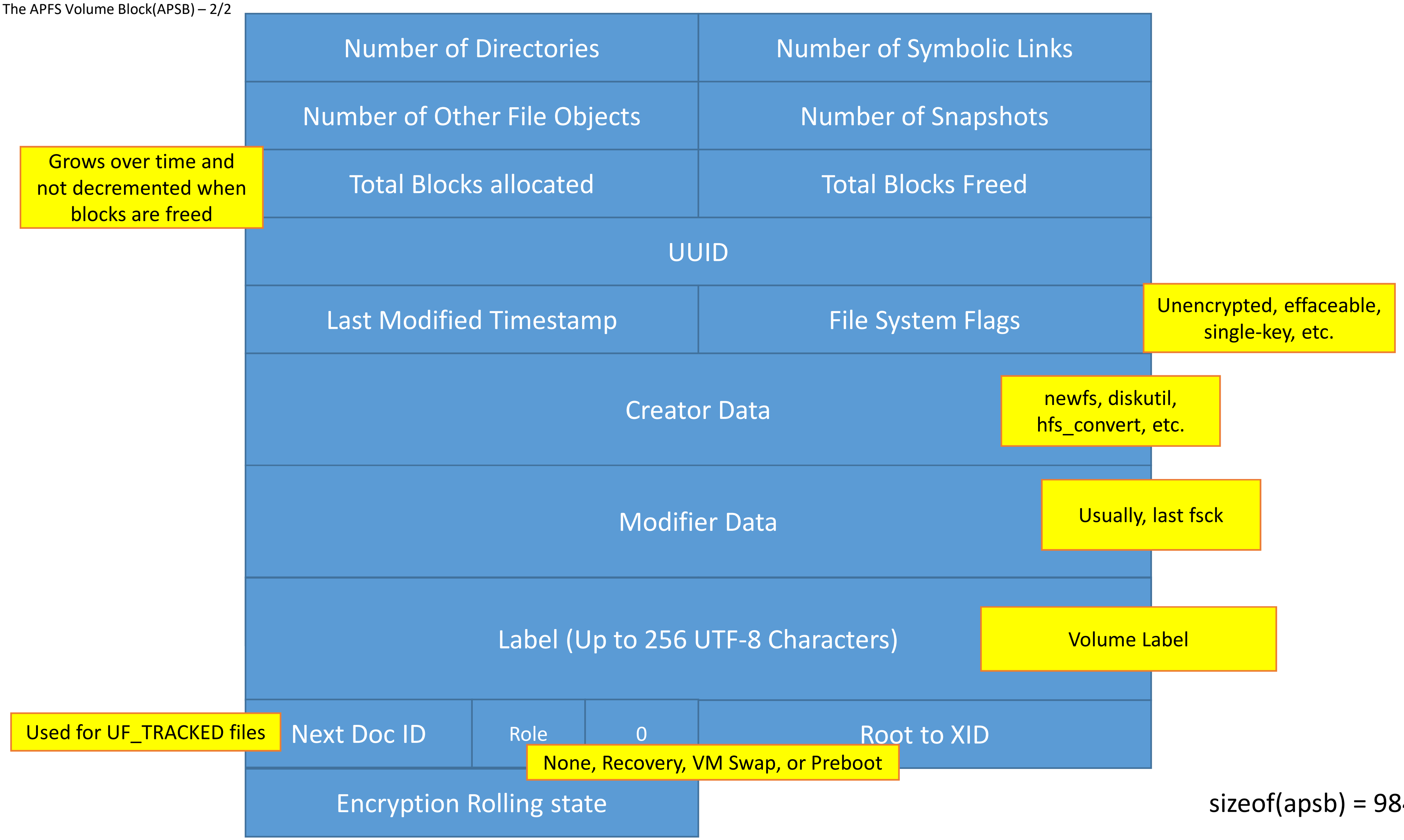
APFS Volumes

- The Volume (“apsb”) represents a mountable file system
  - Contains its own object map
  - Tied to a given xid (checkpoint)
- Changes frequently!
  - Every filesystem level change (add/remove file object, quotas, etc)
  - Deliberate snapshots

The APFS Volume Block(APSB) – 1/2



The APFS Volume Block(APSB) – 2/2



sizeof(apsb) = 984 bytes

Mounting

- Container superblock at 0x0 consulted
- Locate Checkpoint area to find other (previous) superblocks
- Find superblock with highest XID (may be the original superblock)
- Get Object Map of container
- Read fsOid array to see which volumes are in container
- Lookup fsOid in Object Map to get physical block
- Get Root Tree object ID from Volume’s OMAP
- Root file system will Be a Btree of type 2 (root) and subtype 14 (fstree)

Locating files

- Using Volume's Root Tree B-Tree (as found from Volume's omap)
- Every object is keyed by a 64 bit value:

Type(s)	Purpose
1/11	Snapshot Metadata/Name
2/8	Physical/File extent
3	Inode
4	Xattr
5/12	Sibling/Sibling Map
6	Data Stream (file contents)
9	Directory record (dentry)
10	Directory statistics

- 60 least significant bits provide inode #

Locating files

- Files, directories and symlinks MUST have Inode records
- It's not uncommon for one file system object to have multiple entries:
  - Symlinks MUST also have xattr (com.apple.fs.symlink)
  - Files usually have DSTREAMs, may have Extents and may have XATTR
    - If com.apple.decmpfs is present, it may actually hold file stream for small files
  - Directories commonly have records (their dentries), stats, and xattrs
  - Snapshots must have both metadata and name records.
- File metadata reconstructed by walking over all records with same id.
- Sibling dentries will be with same id, to which name is concatenated.
  - Dentry will reveal file/dir id, which will point to inode and any additional records.

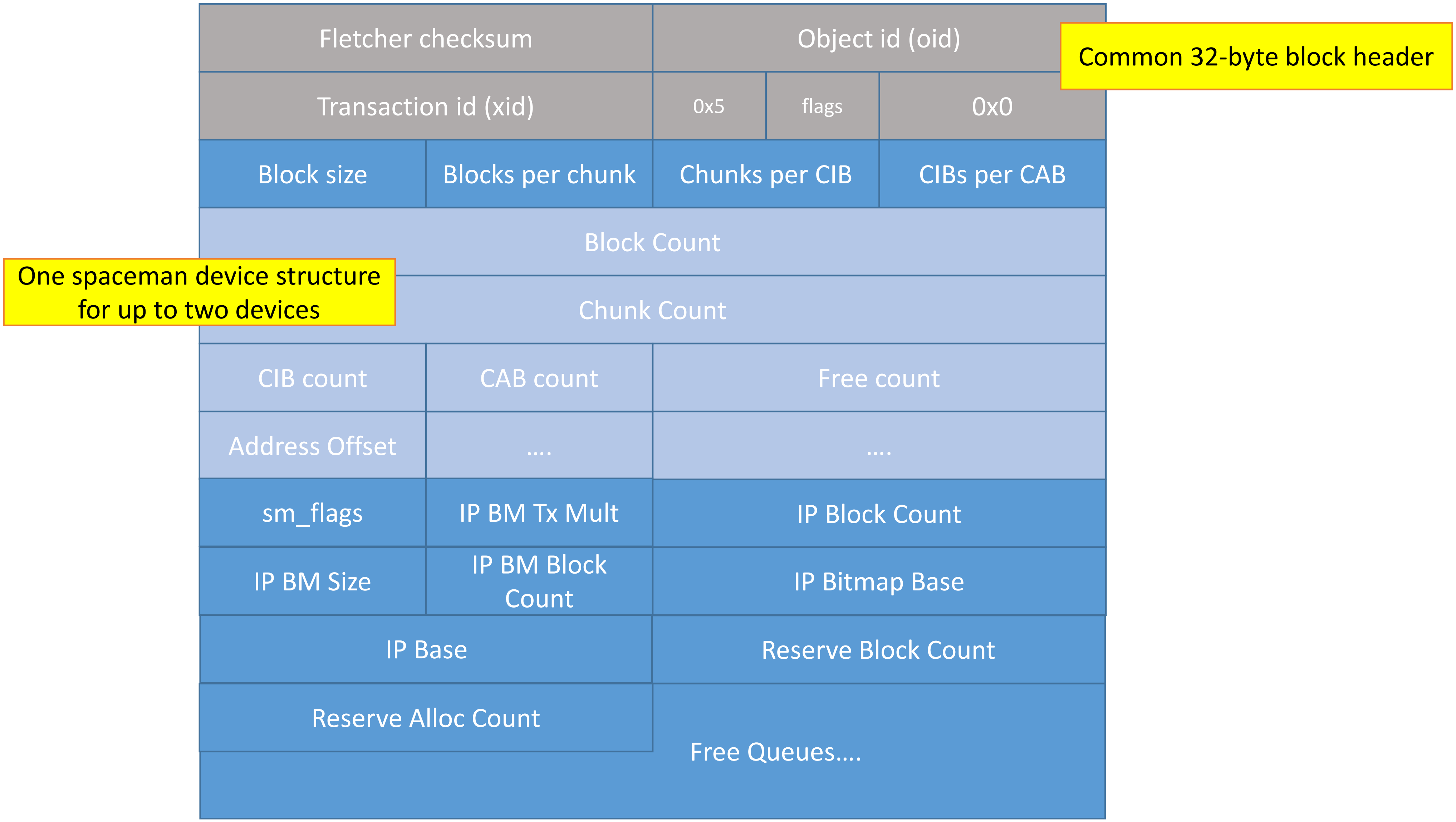
Locating files

- Inode record will appear first
- If file is compressed, it will have **`com.apple.decmpfs`** xattr
  - If file is small enough, content is embedded in attribute
  - Otherwise, **`com.apple.ResourceFork`** holds compressed content in data stream

- If file is uncompressed, it will have one or more extent (type 8) records
  - Extent record defines first block, size, and number of blocks for extent

SpaceMan

- The container uses a Space Manager ('spaceman') for all volumes
- POORLY DOCUMENTED in Apple's reference
- Painful to work with..
- Space manager tracks container free space using:
  - CIB: Chunk Info Block – containing bitmaps for contiguous chunks
  - CAB: CIB Address Blocks – grouping together CIB bitmaps
  - Internal Pool (IP) Bitmap


Fletcher checksum
Object id (oid)
Common 32-byte block header
Transaction id (xid)
0x5
flags
0x0
Block size
Blocks per chunk
Chunks per CIB
CIBs per CAB
Block Count
One spaceman device structure for up to two devices
Chunk Count
CIB count
CAB count
Free count
Address Offset
….
….
sm_flags
IP BM Tx Mult
IP Block Count
IP BM Size
IP BM Block Count
IP Bitmap Base
IP Base
Reserve Block Count
Reserve Alloc Count
Free Queues….

Let’s get our hands dirty

Building a safe testing ground for APFS work

Experimenting with APFS

- `hdiutil(1)` is your friend for handling disks
  - Command line interface of Disk Utility – faster, more efficient, mouse-free

```
#
# Creates an empty APFS disk with no label, which will automatically get mounted as "/Volumes/Untitled"
#
morpheus@Chimera (~)$ hdiutil create -size 50m -type UDIF  -attach ~/apfsTest.dmg -fs apfs
/dev/disk5                      GUID_partition_scheme
/dev/disk5s1                    Apple_APFS
/dev/disk6                      EF57347C-0000-11AA-AA11-0030654
/dev/disk6s1                    41504653-0000-11AA-AA11-0030654         /Volumes/untitled
created: /Users/morpheus/apfsTest.dmg
#
# You can create and partition/label differently or format at any time want
#
morpheus@Chimera (~)$ diskutil partitionDisk disk5 GPT apfs "TEST APFS" 100%
Started partitioning on disk5
Unmounting disk
Creating the partition map
Waiting for partitions to activate
Formatting disk6s1 as APFS with name TEST APFS
Mounting disk
Finished partitioning on disk6
/dev/disk6 (disk image):
 #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:      GUID_partition_scheme                        +52.4 MB    disk5
   1:                 Apple_APFS Container disk6         52.4 MB    disk5s1
```

Experimenting with APFS

- `diskutil(8)` 's apfs menu will get you as far as Apple allows:

```
morpheus@Chimera (~)$ diskutil apfs list
APFS Containers (... found)
|
+-- Container disk1 ....... THIS IS YOUR MAIN DISK. DON'T TOUCH ANYTHING HERE .....
..
+-- Container disk6 ... SOME RANDOM GUID ...
    ====================================================
    APFS Container Reference:     disk6
    Size (Capacity Ceiling):      52387840 B (52.4 MB)
    Capacity In Use By Volumes:   671744 B (671.7 KB) (1.3% used)
    Capacity Not Allocated:       51716096 B (51.7 MB) (98.7% free)
    |
    +-< Physical Store disk5s1 ... ANOTHER RANDOM GUID ...
    |   -----------------------------------------------------------
    |   APFS Physical Store Disk:   disk5s1
    |   Size:                       52387840 B (52.4 MB)
    |
    +-> Volume disk6s1 ... YET ANOTHER RANDOM GUID ...
        ---------------------------------------------------
        APFS Volume Disk (Role):   disk6s1 (No specific role)
        Name:                      TEST APFS (Case-insensitive)
        Mount Point:               /Volumes/TEST APFS
        Capacity Consumed:         24576 B (24.6 KB)
        FileVault:                 No
```

Experimenting with APFS

- To go any deeper takes `fsleuth(j)`
  - The tool formerly known as HFSleuth now also has APFS support
  - Freely downloadable from http://NewOSXBook.com/tools/fsleuth
  - Pure user-mode POSIX implementation (MacOS, *OS, Linux and even Cygwin!)

```
Morpheus@Chimera (~)$ fsleuth ~/apfsTest.dmg

   FSleuth - HFS+/APFS diagnostic tool: Version 2.0(Buenos Aires) Compiled on Oct  1 2018 (C) 2013,2018 Jonathan Levin.
   Free for non-commercial use. Latest version available from http://Technologeeks.com/tools.
   For licensing, a reusable library or even more features (e.g. encryption), please email products@technologeeks.com

Container spanning 49.96 MB (12790 blocks) with 1/1 volumes
Volume 1: (Block 0xa1) Label: 'TEST APFS'
            Contains 0 files, 0 directories, and 0 symlinks Size:    20.0 KB (5 blocks)
..
FSleuth(TEST APFS:/)>
```

- Outside MacOS, can run directly on the physical disk device/image
- In MacOS, requires an entitlement AAPL would never provide…

```
FSleuth(untitled:/)> help
APFS Commands:
volinfo             Display the volume header of the selected file system
volume              Select an APFS volume by label or number
oid                 Find block matching object ID (oid) specified
xid                 Set active transaction ID. Will make only objects matching that XID accessible
diff                Find FSTree differences between two XIDs (as 0x...) arguments
inode               lookup inode specified
container           Display APFS container details
block               Smart-Dump a block (specified by 0x...)
undelete            Undelete a specified file     Smart-Dump a block (specified by 0x...)
..
Filesystem-independent Commands:
fs                  Set active file system for operations to specific mount point or device
listfs              List all detected file systems and their types
pull                copy file to /tmp (requires active file system)
dir                 list files (requires active file system) - synonymous with ls
ls                  list files (requires active file system) - synonmous with dir
cd                  Change directory (requires active file system)
string              Search for string in entire partition/disk-image (lengthy!)
blockmap            Produce a map of all blocks on this volume (copious output!)
hexdump             Hex dump a block (specified as 0x...)
debug               Toggle Debug traces on/off
xml                 Toggle XML Output on/off
verbose             Toggle verbose mode on/off
color               Toggle color on/off
uncompress          Uncompress a DMG to _output_ (valid only on koly DMG inputs)
help                Display this help
?                   Display this help
!                   Shell command
quit                Quit this program
version             Display version
```

Practical example

Demonstrating Copy on Write behavior, and checkpoints

Take aways

- Whether or not you like it, APFS is here to stay
- Reference doc – better late than never, but really, too little.
  - Wait for MOXiI Volume II – it fills the gaps in Apple's (incomplete) document
- APFS CAN support undelete, but Apple doesn't want outside snapshots
  - Fsleuth will change that.
  - Still won't be useful outside forensics due to entitlement (or disable SIP…?)
- State of fusion is in confusion..
  - Fusion drive support undocumented, and largely irrelevant

The End

Or just the beginning –

APFS will stay with us until well after we will have all retired.

Resources

- Apple’s (finally-released-but-disappointing) APFS reference:
  - https://developer.apple.com/support/apple-file-system/Apple-File-System-Reference.pdf
- Seminal APFS research of Kurt H. Hansen & Fergus Toolan (https://www.sciencedirect.com/science/article/pii/S1742287617301408)
- FSleuth (formerly HFSleuth) download:
  - http://technologeeks.com/tools/fsleuth
  - Release version coming soon!
  - Pro version to be released via Technologeeks
- All this and further, even gorier details: *OS Internals, Volume II
  - Dedicated chapters on VFS and APFS
  - If you know anyone @AAPL – Please nag them to release the XNU-4903 sources..