Table of Contents

Introduction to Apple Inc.

Apple Inc.

Apple Inc. Is an American multinational technology company whose are headquartered in Cupertino, California. It designs, develops, and sells consumer electronics, computer software, and online services. The company's hardware products include the iPhone smartphone, the iPad tablet computer, the Mac personal computers, the iPod portable media player, the Apple Watch smartwatch, and the Apple TV digital media player. Apple's consumer software includes the macOS and iOS operating systems, the iTunes media player, the Safari web browser, and the iLife and iWork creativity and productivity suites. Its online services include the iTunes Store, the iOS App Store and Mac App Store, Apple Music, and iCloud.

Apple was founded by Steve Jobs, Steve Wozniak, and Ronald Wayne in April 1976 to develop and sell personal computers. It was incorporated as Apple Computer, Inc. Since its incorporation Apple Inc. Has been releasing the OS for its Macintosh systems. Which have been getting better and better over these years.

Macintosh Operating Systems

Initially, Apple Inc. Released Rhapsody Developer Grail1Z4 / Titan1U release on August 31, 1997. Apple Inc. Released Mac OS X Server 1.0 Hera and Mac OS X Developer Preview on March 16, 1999. Apple Inc. Also released Mac OS X Public Beta Kodiak on September 13, 2000. After that Apple Inc. Released Mac OS X 10.0 Cheetah, this was a way much better operating system then its previous versions. After that Apple Inc. Kept releasing different versions – better to better - of the Mac operating system. In 2012, Apple Inc. Released OS X 10.8 Mountain Lion and shortened the name of the system from Mac OS X to OS X and now with the release of macOS 10.12 Sierra, the name was changed from OS X to macOS. The most recent version of macOS 10.12 Sierra is 10.12.5 (16F73) which was updated in all Mac products on May 15, 2017. Furthermore, Apple Inc. Announced to release macOS 10.13 High Sierra on June 5, 2017. The public beta for macOS 10.13 High Sierra is now available to download, which is more stable and feature-complete. Apple says to that it will release its final version this autumn.

Macintosh Operating System – Code Names

Apple Inc. Gives a code name to its every macOS release. Here is a list of the operating systems versions released by Apple Inc. Along with their code names:

Version	Code Name	Release Date
Rhapsody Developer Release	Grail1Z4 / Titan1U	August 31, 1997
Mac OS X Server 1.0	Hera	March 16, 1999
Mac OS X Developer Preview	--	March 16, 1999
Mac OS X Public Beta	Kodiak	September 13, 2000
Mac OS X 10.0	Cheetah	March 24, 2001
Mac OS X 10.1	Puma	September 25, 2001
Mac OS X 10.2	Jaguar	August 24, 2002
Mac OS X 10.3	Panther	October 24, 2003
Mac OS X 10.4	Tiger	April 29, 2005
Mac OS X 10.5	Leopard	October 26, 2007
Mac OS X 10.6	Snow Leopard	August 28, 2009
Mac OS X 10.7	Lion	July 20, 2011
OS X 10.8	Mountain Lion	July 25, 2012
OS X 10.9	Mavericks	October 22, 2013
OS X 10.10	Yosemite	October 16, 2014
OS X 10.11	EI Capitan	September 30, 2015
macOS 10.12	Sierra	September 20, 2016
macOS 10.13	High Sierra	Beta version available since June, final version to be released this autumn (as announced by Apple Inc.)

WWDC

The Apple development team has been continuously adding better to better features and introducing newer technologies, which are beyond imagination. Apple Inc. Has taken the world towards modern to modern technology and is still working on many of its incredible new project. Every year Apple Inc. Holds a developers' conference. This developers' conference organized by Apple Inc. Is known as Apple Worldwide Developers Conference, WWDC. In this event the Apple Inc. Showcases its new software and technologies to the software developers. WWDC 2016 was held on June 13-17, while WWDC 2017 took place on June 5-9 in San Jose, California.

Introduction to Apple File System (APFS)

Apple Inc. Introduced APFS as a next generation file system as:

Introducing Apple File System

A snapshot of next generation in storage

Apple file system also abbreviated as APFS, is one of the biggest change came to every new Apple device. APFS is a modern file system set to replace the now thirty-year-old HFS file system. Apple file system was released for iOS devices on March 27, 2017, with the release of iOS 10.3 (iOS is an operating system used for mobile devices manufactured by Apple Inc.). APFS was being released for macOS devices beginning with macOS 10.13, released in beta on June 5, 2017 and it came to general release in the fall. In WWDC 2016, one whole session about Apple File System was organized in which the Apple developers' team told the developers about the new file system they introduced and its features. Some points that they discussed about APFS included:

- Introduction and motivation to APFS
- Why they decided to build APFS?
- Some of the new features they had added to APFS
- New APIs for apps in APFS

Apple Inc. Announced the Apple file system in WWDC 2016. According to the Apple developer team, at that time, they said that it was available in macOS Sierra and will be available as a developer preview technology in macOS Sierra.

Next Generation File System

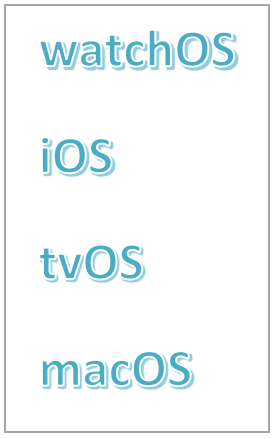
APFS is the Next Generation File System that the Apple developers have been building for apple products. It's modern and rich in features. In it such new features have been introduced which are not present in any of the existing file system, thus making it super-fast and more reliable than any other file system.

Designed to Scale from an Apple Watch to a Mac Pro

One of the core point of Apple File System is that it is designed to scale it from an Apple watch all the way up to a Mac Pro. The Apple developers wanted it to be such a file system which can fit easily for all the Apple products. So, the Apple developers made such a file system which is fully optimized for all Apple products hardware needs. Thus, Apple File System is a new, modern file system for iOS, macOS, tvOS, and watchOS.

The Apple developers developed APFS for both long time developers and all those people who are new to the platform, as the expected audience. So that they intended to cover all of the high level of the new file system with enough details so that everyone could follow along.

It was set to release in 2016 and APFS updated first on all the iDevices with iOS 10.3. APFS became the default filesystem on iOS 10.3. After that in 2017, it got updated in macOS and all other Apple Products' operating systems. In this way Millions of devices converted successfully to APFS by the WWDC 2017.



Apple Inc. Oss

Designed to Take Advantage of Flash / SSD Storage

The Apple developers designed APFS with solid state storage in mind i.e. to take advantage of flash and SSD storage because all of the apple products use Solid State Drives (SSDs). SSD is the flash storage chips inside Apple Watch, Apple TV, iPhone, iPad, MacBook, and high-end

Mac desktops. Apple File System is optimized for flash and SSD storage, but it can also be used with traditional hard disk drives (HDD) and external, direct-attached storage.

Engineered with Encryption as a Primary Feature

Apple File System is engineered with encryption as a primary feature from the very beginning, as they brought this idea to fruition.

Upgrading from HFS+ to APFS

Apple provides an in-place upgrade path from HFS+ to APFS. While this upgrade the user data will remain exactly where it is and Apple will write APFS metadata - brand new into the HFS+ free space. It is done for crash protection. This is a multi-second to multi-minute operation and over that time if anything bad happens to the device, the data on the device will remain safe and sound as if nothing ever happened.

Motivation for APFS

What About HFS+?

In 2016, Apple Inc. Was shipping HFS+ as primary file system but its original design was over 30 years old at that point. HFS was designed in an era when floppies and HDDs were state of the art. But now as the world has changed; since we now use SSDs and next generation storage technologies are evolving as well. So, it had to be changed so that it can grow and move with the latest technologies, needs and ideas.

Single-Threaded Data Structures

In any file system, each node in a B-tree implemented in memory storage consists of a memory block which is normally a handful of kilobytes. A B-tree block is full of keys and pointers. The block size is normally file-system dependent and chosen to use the file system's read and write operations efficiently. The pointers are not normal memory pointers, but rather they are disk addresses, chosen to be easily used by the supporting file system.

Structures in HFS+ are relatively single threaded so the B trees rely on a big walk meter to access or mutate them. It means that if any property of a file gets updated or is tried to be accessed or searched it takes a lot of time to do so.

Rigid Data Structures

The data structures in HFS+ are relatively rigid. By this it means that the things like the file record or catalog record in HFS+ (which is more or less equivalent to an inode in other filesystems) is fixed. Thus, in order to add new fields, to expand a file system, give it new features, Apple developers had to incur cost of backward incompatible long informant change. Therefore, adding new features means a lot of backwards-compatibility and forwards-compatibility issues.

Keeping in mind the structure of HFS+, the Apple developers had to think that what happens when they add a feature in HSF+ and take it way back to 10.5 (Apple OS version 10.5) and try to attach the same file system. So, if they are concerned about backwards compatibility as well as forward compatibility, if they kept on using the HFS+ (which had basics in a thirty-year-old file system) it would have been impossible to do so. So that, the Apple developers had to make something completely new altogether and hence, they thought about something new - APFS.

Why a New File System?

Designed (and Tuned) Specifically for Apple Products and Ecosystem

Apple Inc. Wanted something that was designed and tuned specifically for all Apple products and ecosystem. Other file systems serve other purposes and they do it well, in particular filers and enterprise level storage servers have a lot of features but they do not make sense in apple products because the users can simply make their own device a powerful server. MacOS Server brings even more power to the business, home office, or school. Designed for macOS and iOS, macOS Server makes it easy to collaborate, develop software, host websites and wikis, configure Mac and iOS devices, and remotely access a network. It’s also remarkably simple to install, set up, and manage. So that the Apple devices typically use a single storage device on them, which have a wide range of scale.

On an Apple watch, there is significantly less DRAM and storage then does a Mac Pro which has tons of gigabytes of DRAM potentially and terabytes of storage.

Scale File System Footprint to Support Apple Watch up to Mac Pro.

The Apple developers wanted something that was flexible and dynamic and reacts to the platform on which its running.

Enhance Security Capabilities

The Apple developers wanted to add new and enhanced security capabilities. On ios today Apple Inc. Already ships a version of HFS+ that uses per file encryption which means that every file is encrypted differently on storage from every other file on the file system. They wanted to take that a step further.

Add New Features

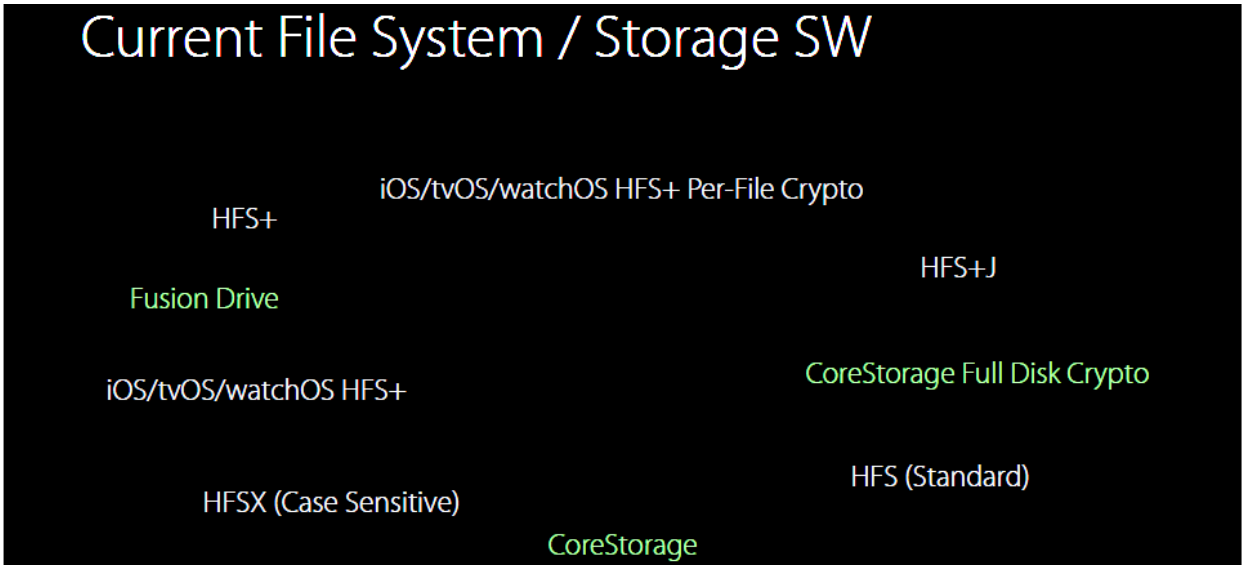
The Apple developers wanted to add new general file system features that were requested and the one they learned were important for the feature of their platform.

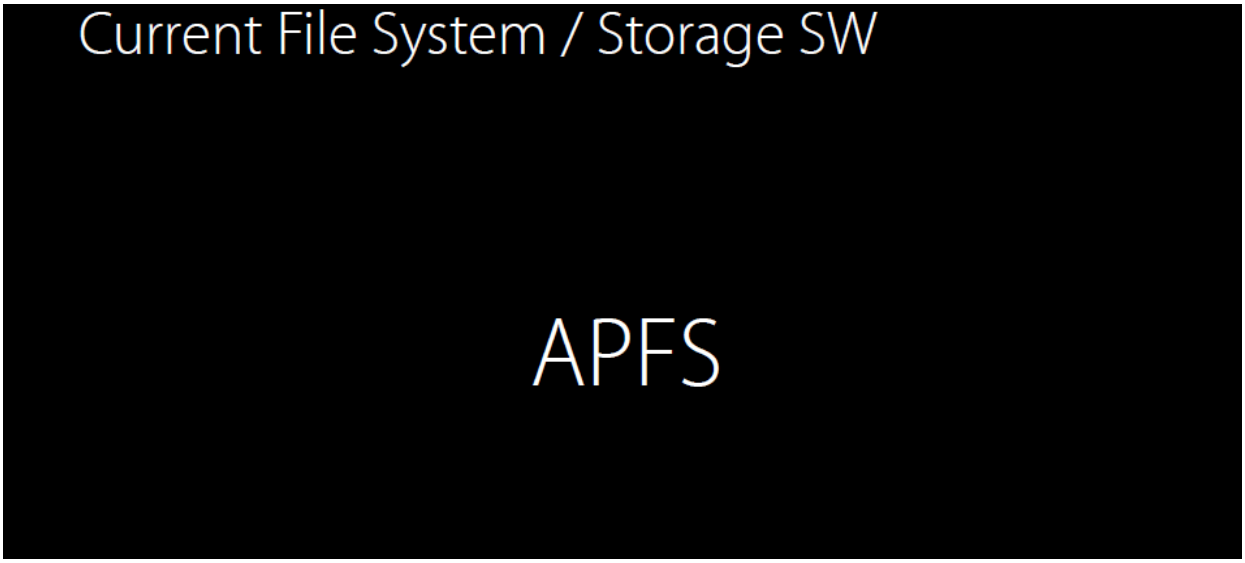
Today, people commonly store hundreds of gigabytes and access millions of files on high-speed, low-latency flash drives. People carry their data with them, and they demand that sensitive information be secure.

Current File System

APFS does not only comprise of HFS+. It is not just HFS+ in a little small bubble but it's actually many technologies that comprise this file storage software. From the beginning, it was ***HFS (Standard)***, almost over 30 year ago, then they added ***HFS+*** some years later. They added crash protection to it so they gave it a Journal - ***HFS+J*** and case sensitive variant as well – ***HFSX (Case Sensitive)***. They also added ***CoreStorage*** which gives Full Disk Encryption - C***oreStorage Full Disk Crypto***, as well as a ***Fusion Drive*** which bonds the speed of the SSD with the capacity of the hard drive. The Apple developers also introduced ios specific HFS+ as well as the one that supports per-file encryption.

Thus, their intention behind this was all of these technologies get replaced by one thing – the ***Apple File System (APFS)***. For a brief look up at the concept see the figure below:





New Features

Following are the new features that are added to APFS which make it different and superior from the other file systems:

- Improved file system fundamentals
- HFS compatibility
- Space sharing
- Cloning files and directories
- Snapshots (and reversions)

- Fast directory sizing
- Atomic safe-save primitives
- Encryption
- Sparse files
- Unicode and Normalization support
- Defragmentation

Improved File System Fundamentals

Flash / SSD-Optimized

APFS offers a key new feature for SSDs - Write coalescing. Flash SSD writes are slow, so instead of a series of many independent small writes as individual operations, APFS coalesces writes into a group into a single write and perform one big write for high performance by committing it to the disk as one operation. In this way APFS increases the SSD performance to a huge extent.

Crash-Protected

APFS employs a new and marvel copy and write meditator scheme. Every meditator writes, what is written to a new location on stable storage. Apple developers combine this with a transaction subsystem, which ensures that if the user loses power, if anything corrupts the machine panic or anything bad happens, the user will see a consistent view of what was on the disk or the user won't see it at all (the change).

Modern 64-Bit Native Fields

Apple's paving the way to store lots more data with APFS. HFS+ supports 32-bit file ids, for example, while APFS ups that to 64-bit. That means that today, your Mac can keep track of about 4 billion individual pieces of information on its hard drive. APFS ups that to 9 quintillion. Hence, *APFS supports over 9 quintillion files on a single volume*.

Nano-Second Time Stand Granularity

As, the hinted numbers have been expanded to 64 bits. The Apple developers have increased the time stand for the now 64 bits, hence now due to APFS the Apple products support Nano-second time stand granularity. They also support sparse files for the first time in the Apple File System. Now all the files directly represent the pointers where the blocks actually live on disk in an expanded 64 bits.

Extensible Design for Data Structure Growth

As, HFS+ data structure is relatively rigid but in APFS the data structure that represent the core inode are flexible. So, fields are now optional or the Apple developers may not have invented them yet. New fields that the Apple developers make choose to add down the line will be correctly recognized as not supported. Thus, APFS will not understand them if you attach that storage to today's version of macOS Sierra. In this way, the Apple developers can add *new features without fear of humming backwards compatibility*. This also allows them to add optional fields and some systems having the mere presence of a file that is enough to convey some information you don't necessarily need all the machinery that points on which block is it of disk because they are not needed so those fields are optional. So, if there is a zero-byte file, it will not occupy any memory space or memory block.

Optimized for Apple Software Ecosystem

The developers wanted to add features and optimize APIs that are extremely compelling for the Apple platforms moving forward. So, the Apple developers optimized APFS for the Apple software ecosystem so that the developers can add features and optimize the APIs easily and efficiently.

Low-Latency Design

In typical file systems, latency is often a tradeoff b/w latency and throughput. But the Apple developers have chosen to bring it on the side of latency. They do this because they want the user apps, when a user clicks on it on the desktop or tap on it on the phone, to come up quickly and responsively and with a very crisp animation. And the reason for that is when the developer or the user goes down to file system, they want to ensure that the user gets the answers that they need as quickly as possible.

Native Encryption Support

APFS has Native encryption support built in the file system. On HFS+, it uses per-file encryption but those are stored on disk through its standard attributes. In APFS it's not the case, these are now first-class citizens and first-class objects inside the file system. APFS is that it's designed with encryption in mind. It combines the full disk encryption features found in the latest versions of OS X with the data protection feature that encrypts every file individually on ios.

HFS Compatibility

APFS Supports and Replace HFS+ Functionality

If the users have apps that run on HFS+, they continue to run as it is on the user side without any visible change. Therefore, APFS supports and replaces HFS+ functionality. But there are three things they will not support, which are:

1. Exchange data
2. Switch of OS
3. Directory hard links for time machine

But every other API and behaviour will be supported by APFS just as it is on HFS+.

Space Sharing

Most Apple product end users of have only one partition because it's hard to make partition in an Apple system. The user has to take the disk out and changing it is relatively expensive. Free space on one partition does not translates into available free space at another partition. So, the Apple developers solved this with the feature of "Space sharing".

For example: A user is downloading latest and latest cat video. That file grows and it gets bigger. In fact, so big that the user has completely run out of space on a partition that the user was running. There is not a whole lot that the user can do in this case – if one is out of space, he is out of space! One thing the user could do at all is: completely destroying the partition immediately afterwards and then grow the partition that he was using. *(See the images at the end of this example for clear explanation.)*

The user can destroy partition 2; Partition 2 grows; and now the user has enough space to continue growing his cat video. But this is also inflexible and the sense of a little bit of problem if the file the user was downloading wasn't on partition 1 but it was in fact on partition 0. So, in this case, the file will grow bigger and then even if the user had free space that the user was confident that he was willing to destroy partition 2, he could destroy it. But then partition 0 could not grow because it is not adjacent to any other free space that the user just made available. So, the Apple developers thought that it is something that they could solve with space sharing.

Now the apple file system has come up with this base concept they call as - the container, Appley named; because it contains volumes or individual file systems. In this instance APFS

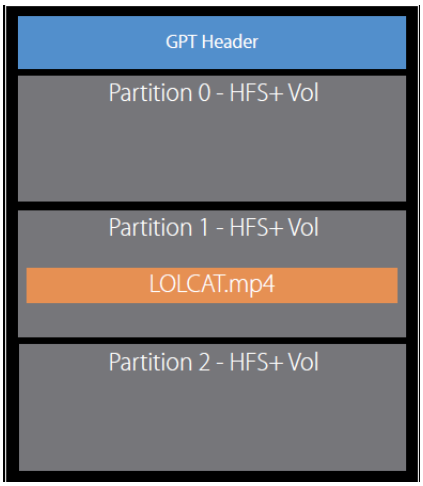
containers represent the lowest level of functionality and this is what encapsulates their block allocator as well as crash protection subsystem.

Let’s say we have Volume 0 which occupies some of the free space in the partition. The Volume can grow; Volume can shrink. But in all these cases, the free space will dynamically resize to what is currently available at the time the user requests it. The user can also create more than one Volume in the container, which will occupy incremental in more space. And then if the user wanted to grow partition 1, Volume 0 at this point you could do so.

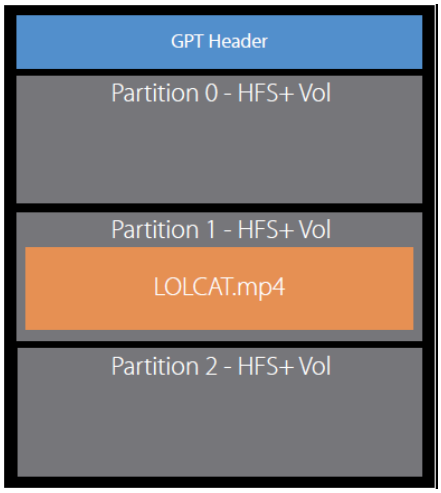
Now if the user asks for how much free space is available on the system, he will get the area that is in the green rectangle at the bottom. This is something that is a suddenly different from the user may have continued free space in the past.

If the user is using some paradigm like taking the total storage size, subtracting the used space to get the free space, that will no longer work because all of Volumes on the container are also participating in space sharing. Additionally, the user can’t necessarily add up all of the used space either.

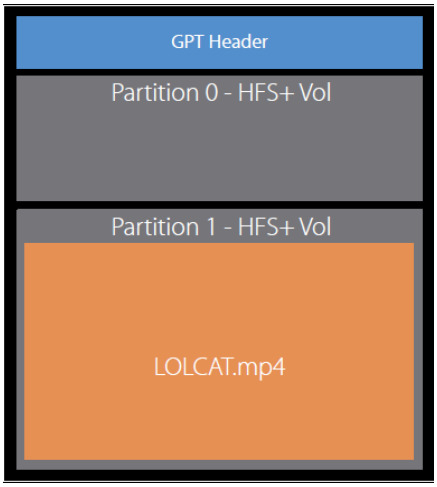
HFS+ Partitions



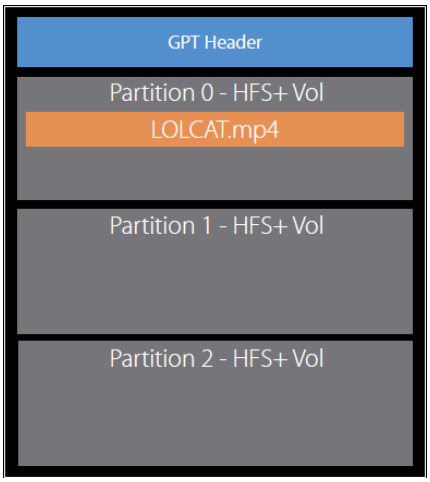
1. The user is downloading the Cat video



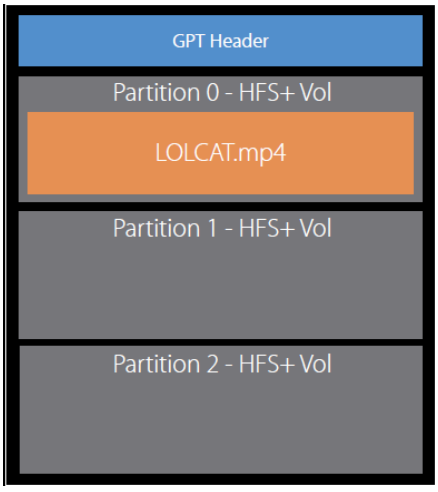
2. The Cat video grows bigger as to occupy the whole partition and now it can't grow!



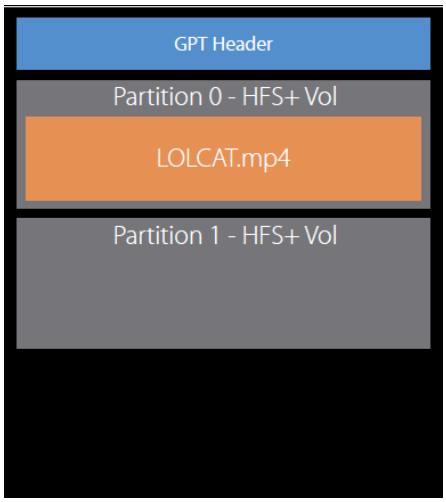
3. The user deletes the partition 2 and now the Cat video can grow



4. ***Problem situation***: The Cat video was in partition 0



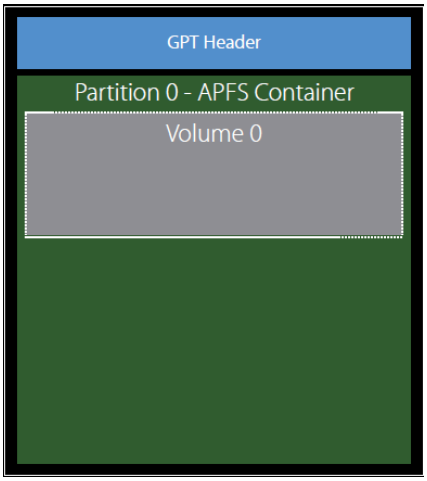
5. The video grows much bigger so as to occupy the whole partition



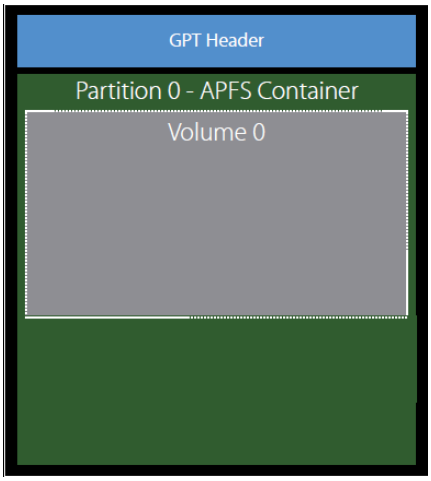
6. The user deletes the partition 2, but still the partition 0 can't grow as it is not adjascent to any freed space

Solution = APFS

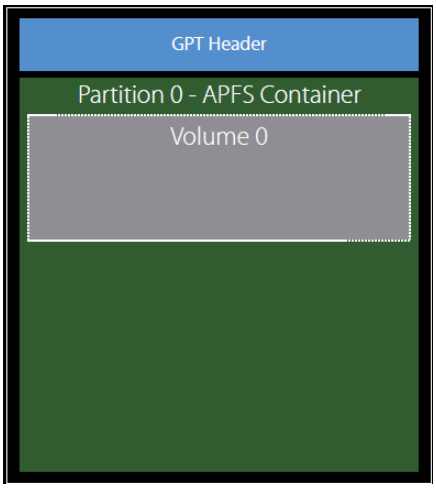
APFS Container and Volumes



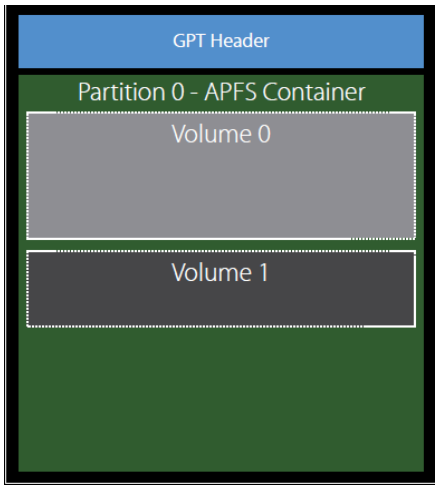
7. Volume 0 occupies some free space in the container



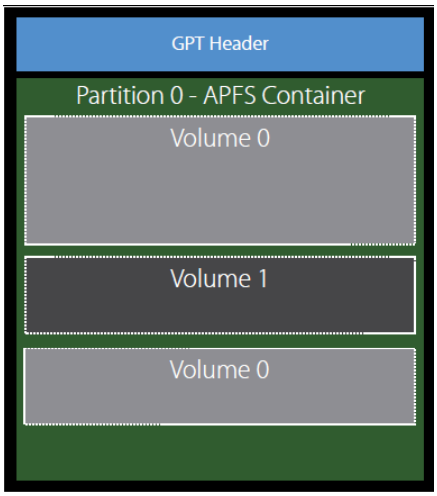
8. Volume can grow



9. Volume can shrink



10. The user creates another partition

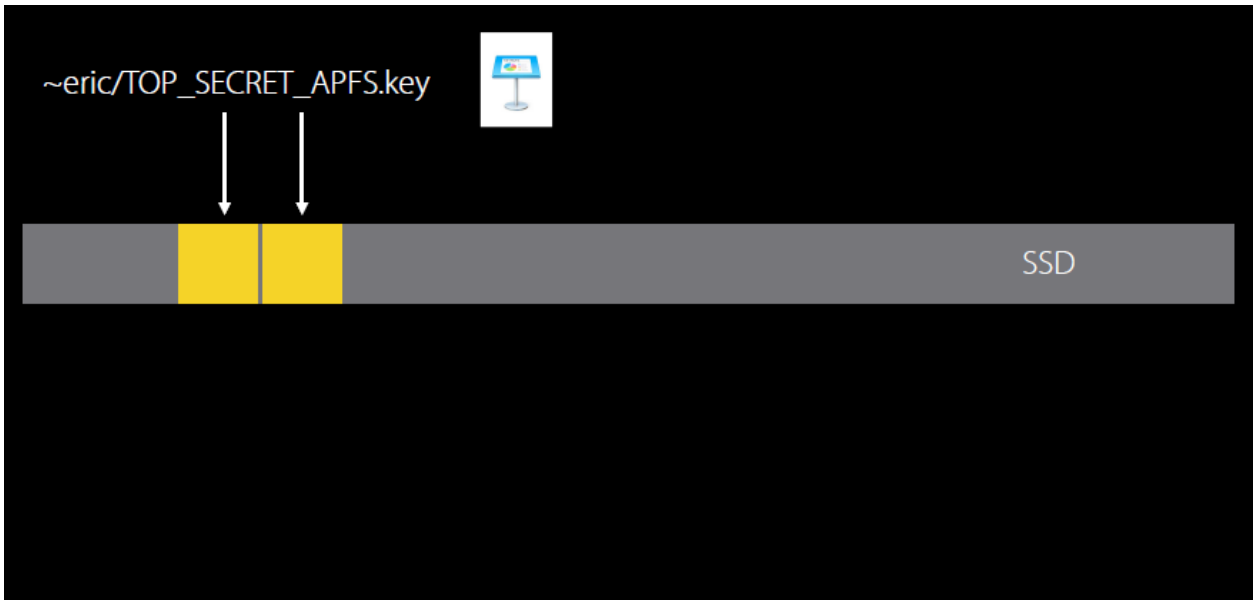


11. Now the user extends the Volume 0. Hence, the free space is the green area at the bottom

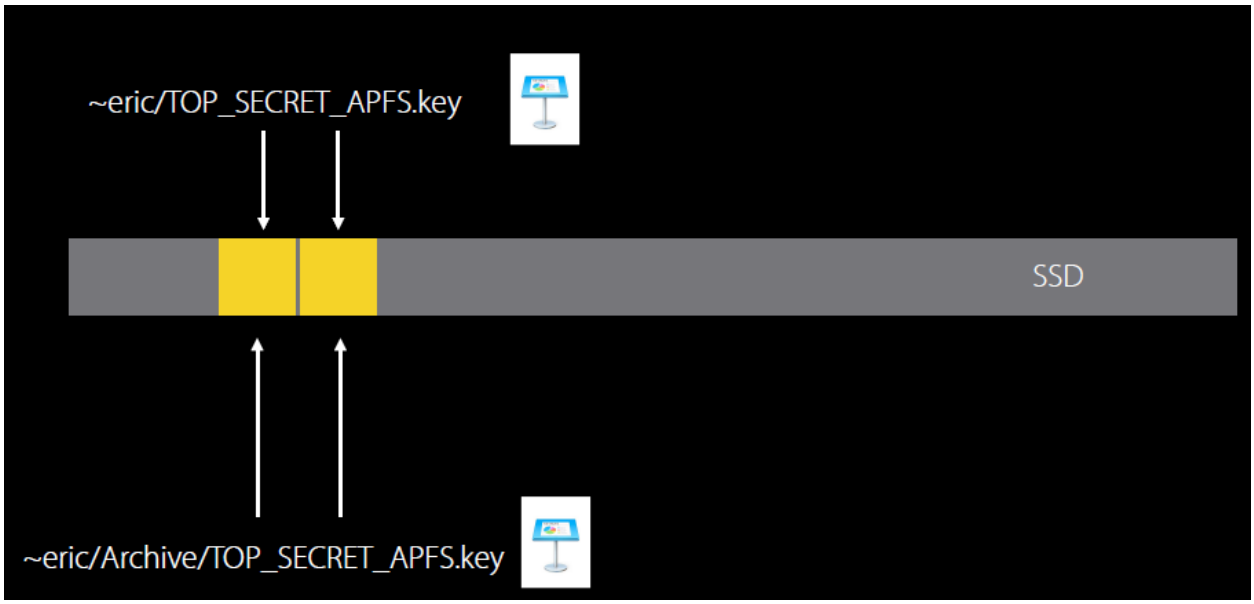
Cloning Files and Directories

A clone is a nearly instantaneous copy of a file or directory that occupies no additional space for file data. Clones allow the operating system to make fast, power-efficient file copies on the same volume without occupying additional storage space.

For example: We have a file TOP_SECRET_APFS. It resides in the home directory and it has references to two blocks of data. Now if someone wants to make an archive of this presentation that have existed at this point of time, he could copy the data by reading it all in and writing it back out. But that has obvious cost in terms of CPU, power and disk space usage.



TOP_SECRET_APFS residing in SSD

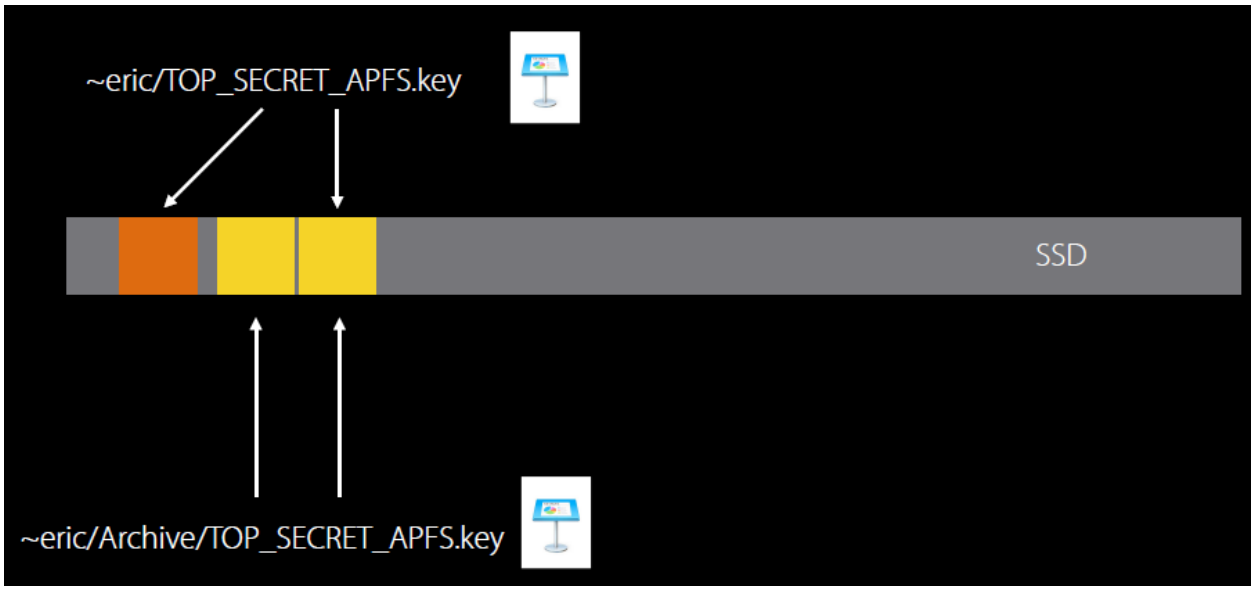


Clone of TOP_SECRET_APFS

Instead with APFS the user can clone the file. In cloning the file, the system copies the references to the data instead of the actual data. It is obviously much faster and if it's a large file, it is not using twice the amount of space. It is using exactly the same amount of space plus small incremental amount for the additional references to the data. What the clone guarantees in the file system is that if a modification is made either to the original or the clone, the file system writes that data to a new location. So, the clone remains untouched.

An important point is that when you have clones, at the time of the clone there will be no additional space used. As the user continues to make modification the user will start to use more and more space.

Modifications to the data write the new data elsewhere and continue to share the unmodified blocks. Changes to a file are saved as deltas of the cloned file, reducing storage space required for document revisions and copies.



Modification in the TOP_SECRET_APFS stored in the SSD

In addition, because ios and macOS support document and application bundles, APFS will also allow the user to clone an entire directory hierarchy. So, a document bundle is a directory that contains a set of files inside of it. APFS can clone that atomically as well.

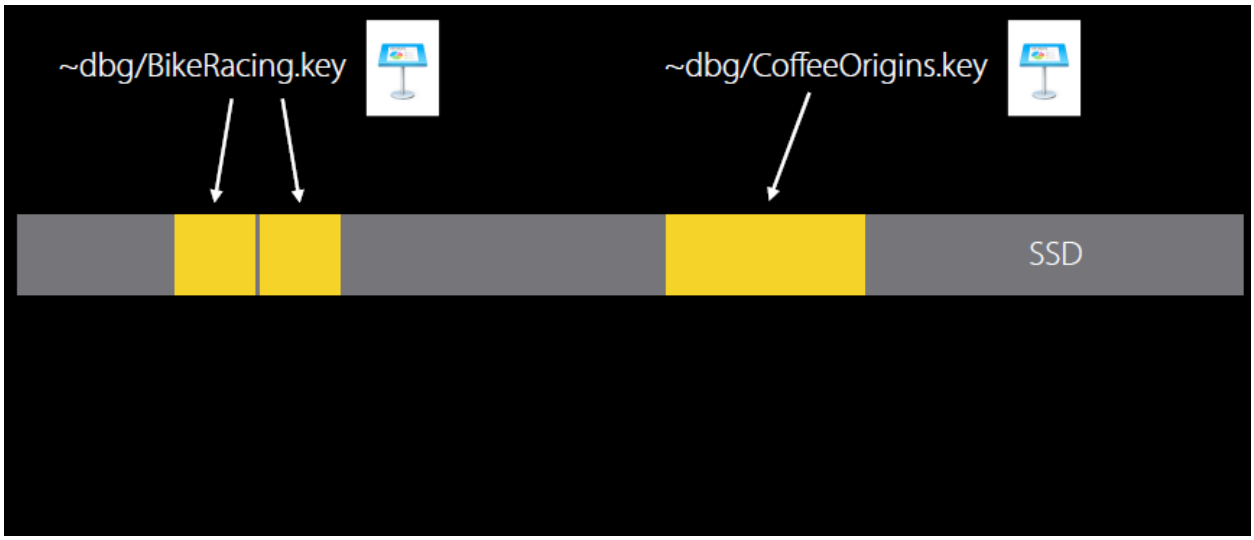
With this feature, after the conversion of an Apple device from HFS+ to APFS, a lot of space was freed up for the end-users. Due to 'cloning' the space occupied by the copy files was released.

Snapshots (and Reversions)

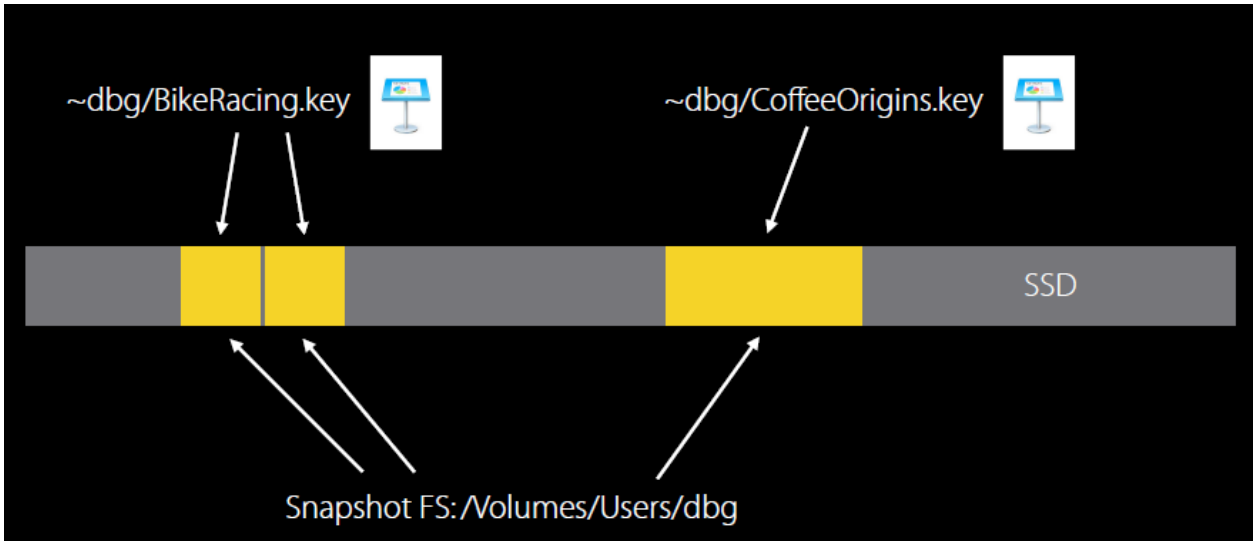
A volume snapshot is a point-in-time, read-only instance of the file system. The operating system uses snapshots to make backups work more efficiently and offer a way to revert changes to a given point in time.

Taking a Snapshot

For example: We have a representation of the file system, there are two files in it i.e. Bikeracing and CoffeeOrigins. Bikeracing has two blocks of data and CoffeeOrigins has one. If we take a snapshot of the file system, we now have a separate independently mountable, read-only copy of the file system that represents the state of the file system at that point of time the snapshot was taken.

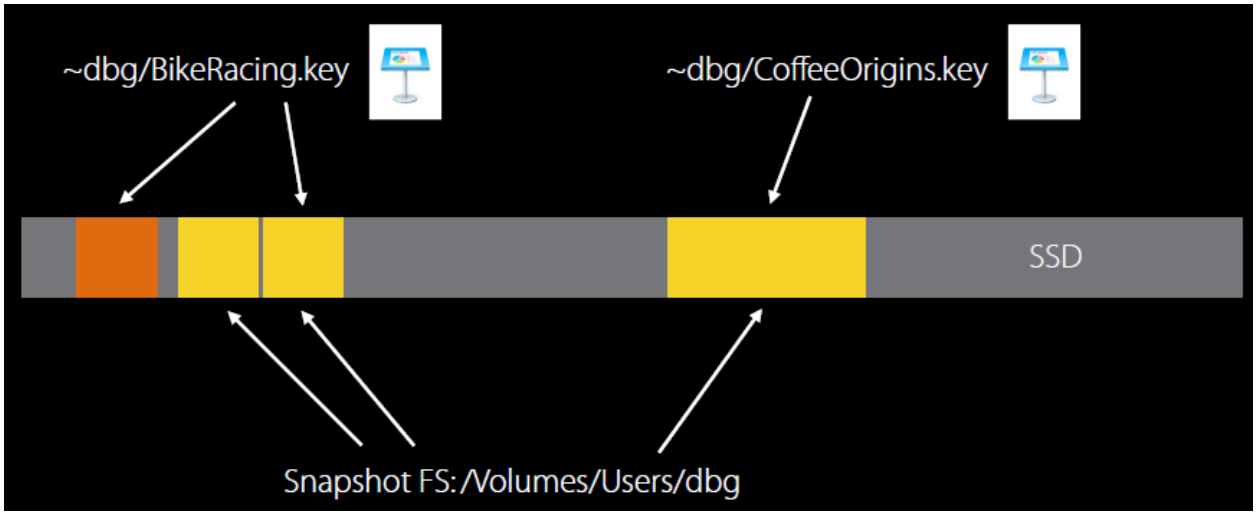


Files in the file system at the time of Snapshot



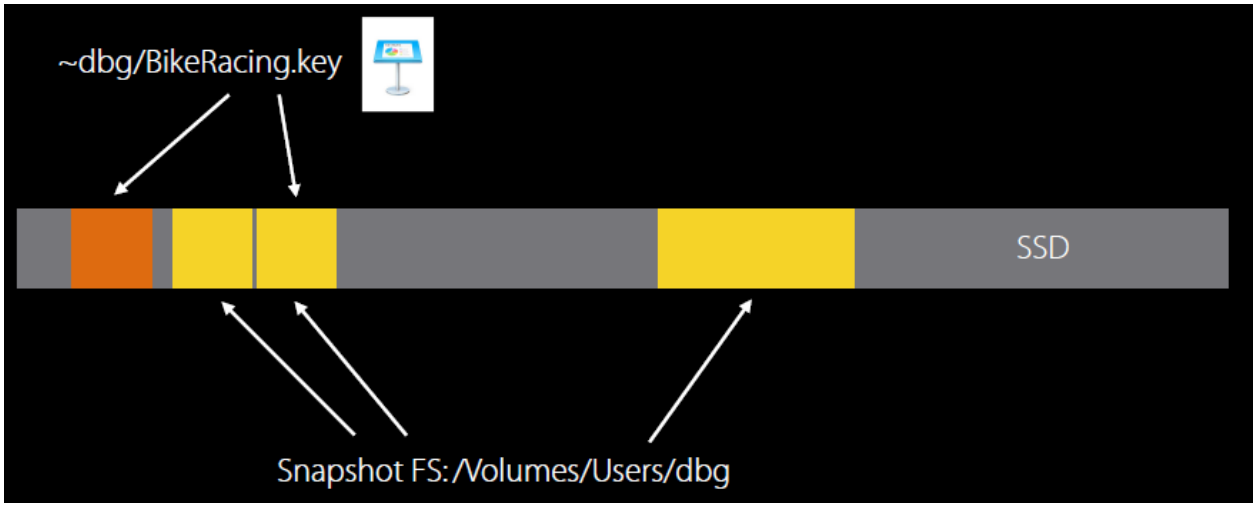
Snapshot of the filesystem

Much like with clones, if a write comes into the file system, the file system will put that data in a new location, preserving the integrity of the snapshot.



Modification in the files saved by the filesystem after taking the Snapshot

Likewise, if we were to delete CoffeeOrigins to try to free up some space, the file system can't reclain those blocks, but the Snapshot continues to refer to those blocks.



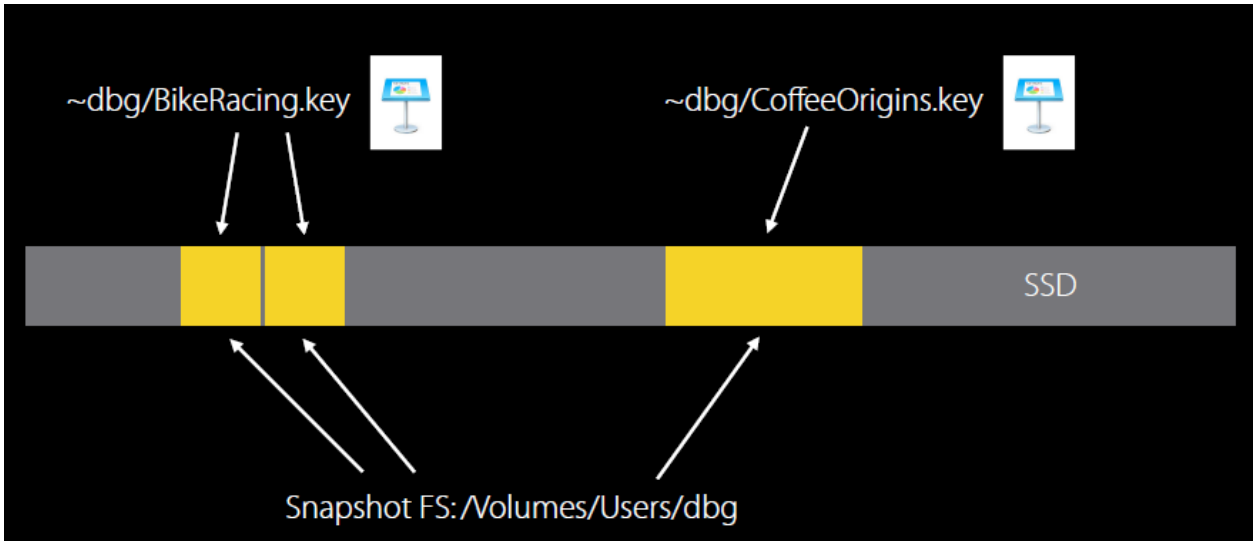
Even after deleting the CoffeeOriginsorigins file the Snapshot still refers those memory blocks

When a file is deleted which was present at the time of the snapshot, the blocks aren't reclaimed. So, snapshots can cause you to use all of the diskspace if the user doesn't harvest them periodically.

This feature in APFS is built with the expectation that the developers will probably use snapshots for the purposes of having a stable read-only copy from which they form backup.

Reverting to a Snapshot

APFS also supports the feature of reverting to a snapshot. For example: we have the same state of the file system as we had above in which the CoffeeOrigins file had been deleted from the file system but a snapshot was taken before its deletion so that the snapshot still refers to that block of data. Now, if the user wants to revert – do a global 'Undo'. Thus, he wants to the point in time of the file system – at the time the snapshot was taken. So that he can make the filesystem to revert to the state to the snapshot. The next time it is mounted the filesystem will rewind essentially to the point it was at the time of the snapshot and then allow the user to continue making changes from that point forward. Hence, the file CoffeeOrigins will come back and the changes to the file will be discarded.



Filesystem reverted to the time of the Snapshot

Hence, the snapshots continues to exist and th euser can revert as many times he like. Particularly, in ios the icloud backup can now use Snapshots in order to take a stable copy of the file system and upload everything to the Cloud without worying about whether or not a process might be modifying a file, getting writes and trimming it.

Fast Directory Sizing

How Much Space Does a Directory Hierarchy Use?

The feature – Fast Directory Sizing is actually an answer to the question that how much space does a directory hierarchy use?

Applications frequently need to compute the size for sizing an operation, provide progress to the user and the obvious way to do this is to open a directory hierarchy, get a read of all of the contents recursively and look at the size of all the items to add the size up. And *users would like to know the answer quickly.*

When a user calculates the size of directory in HFS+, it would take a couple of seconds to do so. The Apple developers were looking to improve it in the APFS.

Issue with Storing the Size of The Directory Hierarchy with The Directory Itself

The file system could keep track of this, the user can store the size of the directory hierarchy with the directory itself but that has *one main issue:*

- *How do you safely update your parent and its parent (and so on…)?*
- *Locking child to parent is a locking order violation in file systems*

When you have lock on a file when you are modifying it, you can't also lock its parent, because it's the lock order violation. File systems always lock from the parent to the child and never from the child to the parent. So, if it is done the other way, the user will have deadlock!

APFS Side-Steps the Problem!

As, the problem is storing the size with the directories, so the Apple developer thought to *store the size separately* somewhere else. So, by storing the size separately we can *use atomic operations to update the size* in a separate record maintained by the file system. And thus, there is no lock order violation. This comes at *small incremental cost for the additional size records* but that is basically lost in the noise with the I/O.

Fast directory sizing allows Apple File System to quickly compute the total space used by a directory hierarchy, and update it as the hierarchy evolves.

Fast directory sizing works by precomputing the size of directory as content is added and removed. The file system can enable fast directory sizing on empty directories. You cannot enable Fast Directory Sizing on directories containing files or other directories directly; you must instead first create a new directory, enable fast directory sizing on it, and then move the contents of the existing directory to the new directory.

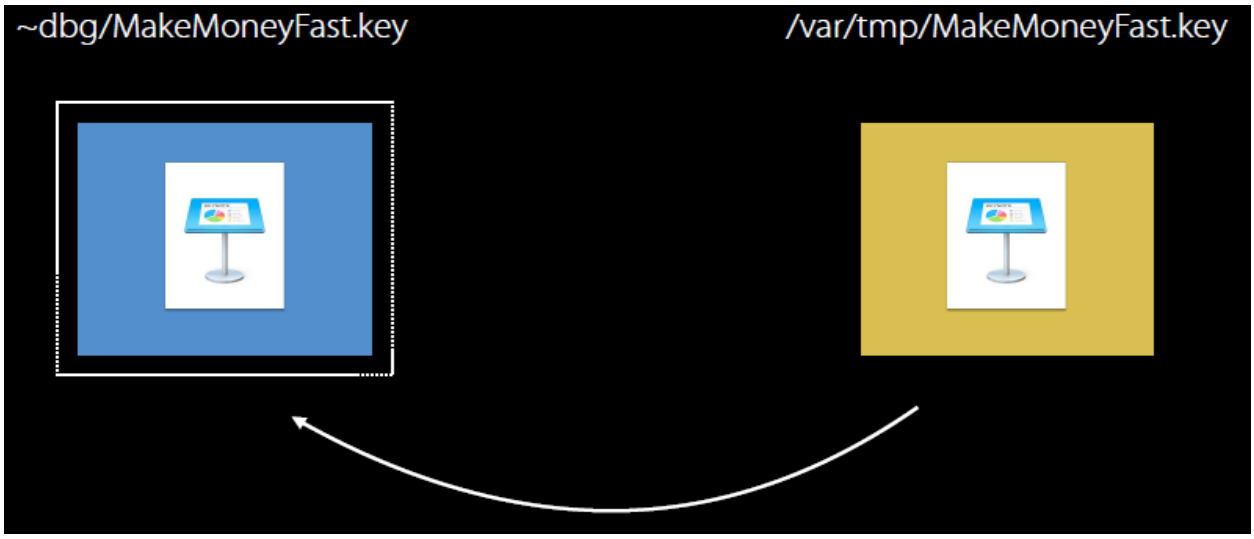
Atomic Safe-Save Primitives

Apple File System introduces a new Atomic Safe-Save primitive for bundles and directories. Atomic Safe-Save performs renames in a single transaction such that, from the user's perspective, the operation either is completed or does not happen at all.

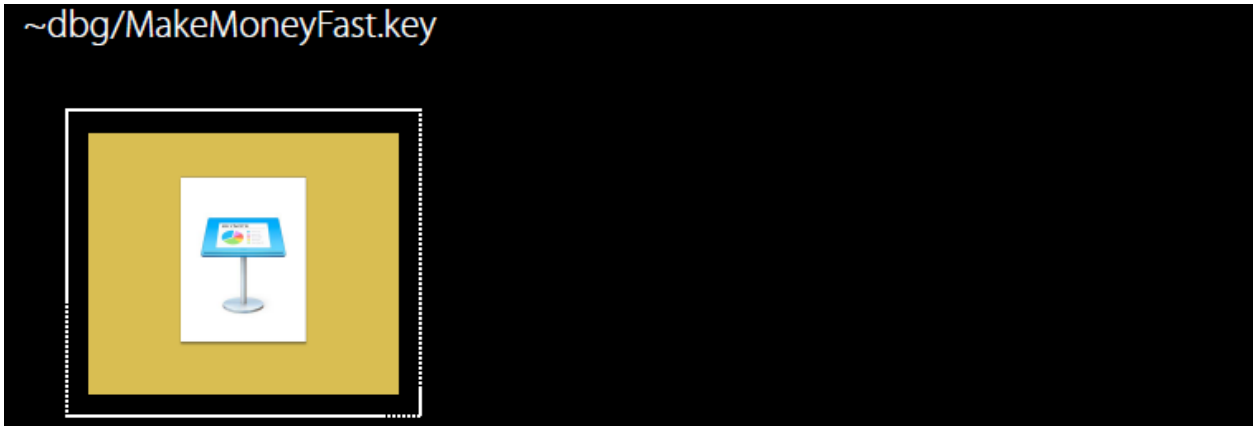
Atomic Safe-Save for a Regular File

Atomic Safe-Save(rename)

For example: There is a file in the file system named as MakeMoneyFast and the user comes up with a brilliant new scheme for making money fast and when the application saves that data initially to a temporary location. When the application is happy that everything has been writing and is safe out on disk, it will ask the file system to perform a rename. Renames are files have always been atomic. The file system guarantees that it either happen completely and it is safe or it doesn't happen at all. In addition, the file system will handle deleting the previous version of the document.



Replacing the updated file in memory



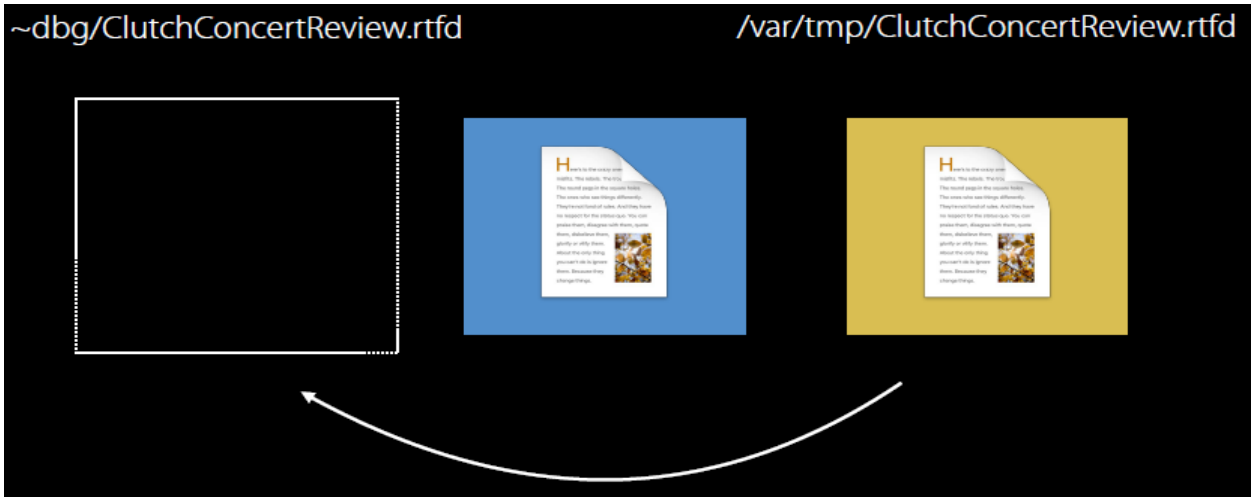
Deleting the previous version of the file

So, that's great for regular files but what happens if the user has a document bundle?

Atomic Safe-Save for a Document Bundle

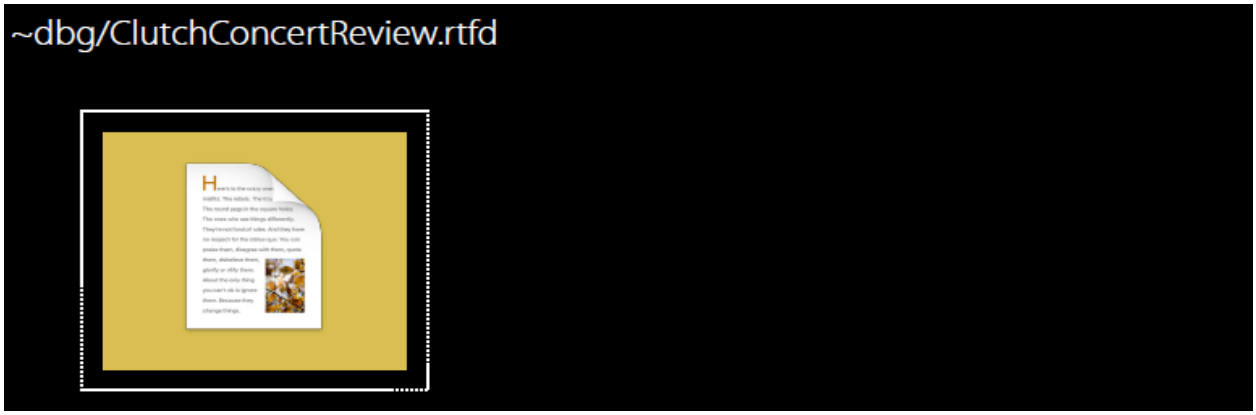
Non-Atomic Safe-Safe (directory)

For example: There is a document bundle named as ClutchConcertReview which is a directory that contains the assets of the document inside of it. The user updates the ClutchConcertReview, the change is written at a temporary place in the memory. But now, there is no way to atomic rename of the directory over the top of another directory, because POSIX semantics don't allow that if the destination has something inside of it. So, we begin playing a shell game. First the document has to be moved out of the way - the live document. So at this point if something were to go wrong and the application crash, the system loses power user's data is gone. Then the application moves the data into place and last it has to handle deleting the previous version of the directory – the document bundle. Hence, it is not atomic and it is not safe. And this is something that has kind of bugged the developers for a very long time and they wanted to improve it.



Moving out the previous version from place

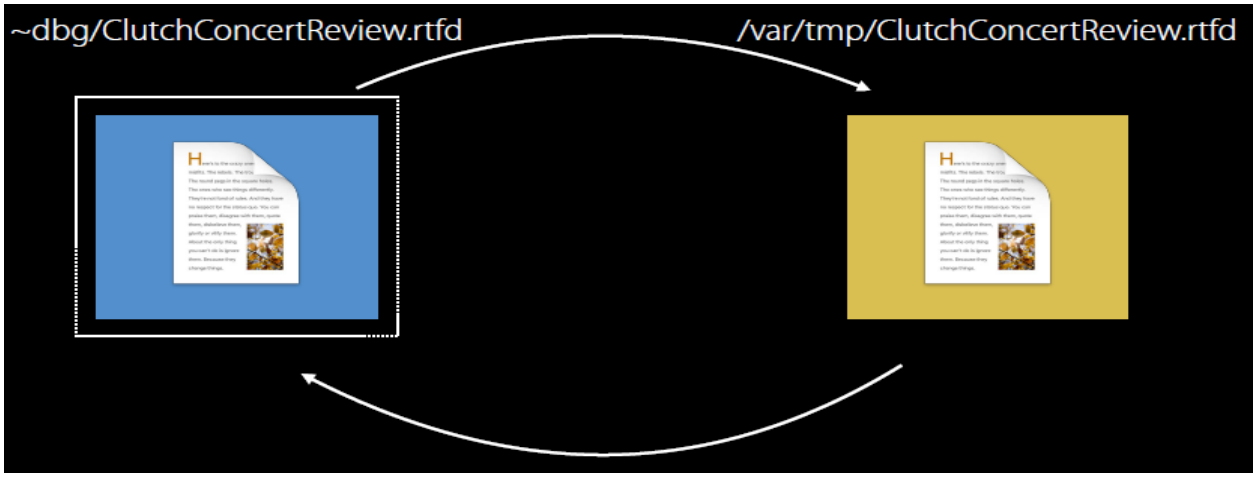
Replacing the updated version in memory

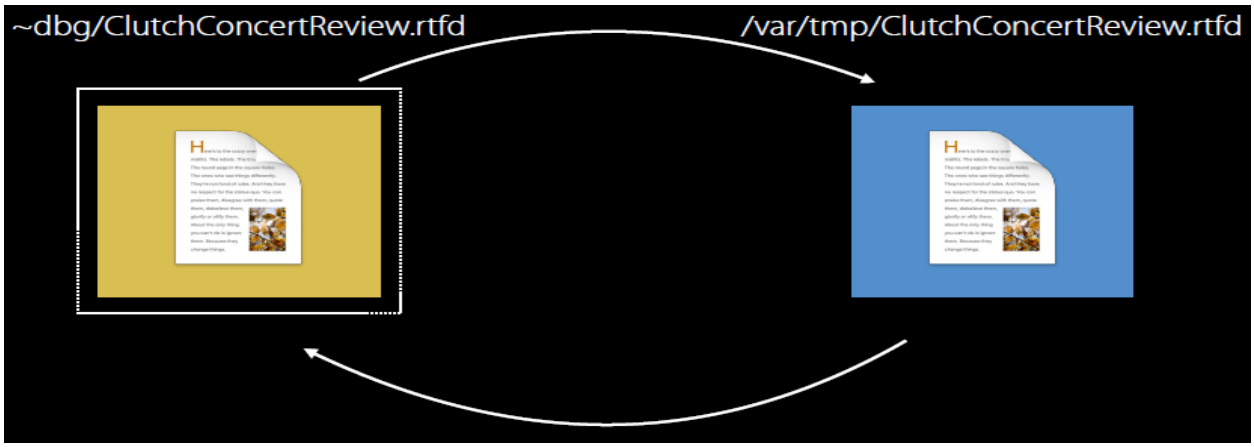


Deleting the older version of document bundle

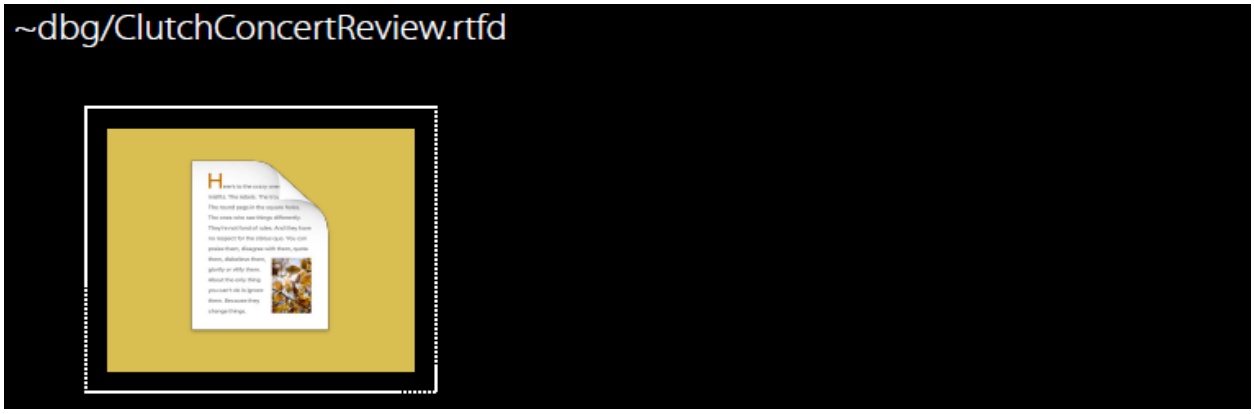
Atomic Safe-Save (renamex_np)

With APFS the Apple developers introduced the new system call named as *renamex_np* for non-POSIX, which allows an atomic safe-save of directory. So now when an application writes the data to its temporary location and asks for the rename operation APFS will atomically handle the swap and deleting the previous version of the document. So, this is now atomic and safe.





Swapping the document bundle versions



Deleting the older version of document bundle

This system call has already been adopted by the foundation for the user and hence the developers get the benefit of this improved behaviour on APFS and do not have to perform this operation by themselves.

Encryption

Encryption (HFS+)

In HFS+ on the Mac Apple developers use layer code core storage. Its beneath HFS and provides full disk encryption among everything. It's a rather sophisticated layer and it does a lot of things. On iOS, they have a different variant that stores encryption keys and extended attributes. The encryption keys work in conjunction with the accelerated AES hardware found on ios devices to provide per-file encryption.

Encryption (APFS)

APFS supports multiple levels of file system encryption. Apple File System supports the following encryption models for each volume in a container:

- No encryption
- Single-key encryption
- Multi-key encryption

No Encryption

All data and metadata is written in plain text to desk.

Single-Key Encryption

It is actually one key per Volume. All sensitive metadata and data are encrypted with the same key. This is essentially be equivalent to full disk encryption.

Multi-Key Encryption

It is the most sophisticated level that APFS supports. In it all sensitive metadata is encrypted with a single key which is distinct from the per-file keys that are used in encrypting individual files. In addition, because of how snapshots and clones work APFS supports per extent encryption. So, each region of the file can be encrypted with its own key. Hence, there are pre-file keys for file data and a separate key for sensitive metadata. This is unique and no other file system supports anything like this.

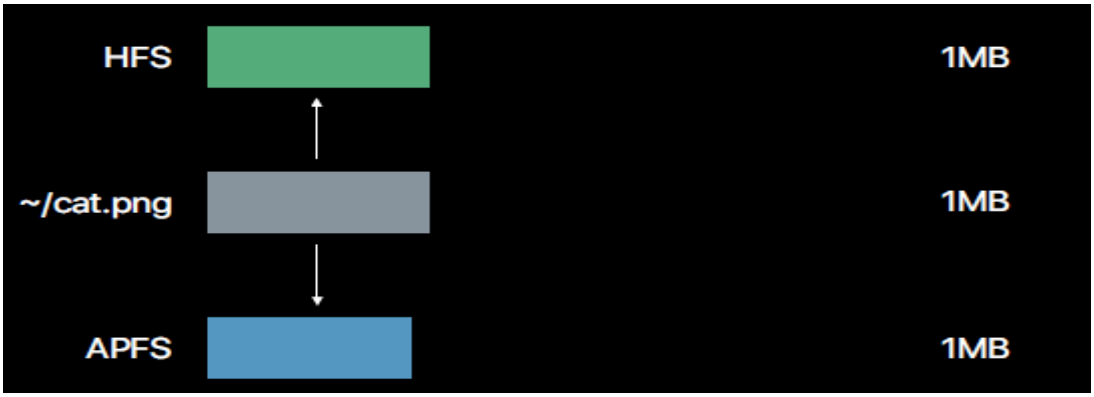
Multi-key encryption ensures the integrity of user data. Even if someone were to compromise the physical security of the device and gain access to the device key, they still couldn't decrypt the user's files.

Security and privacy are fundamental in the design of Apple File System. That's why Apple File System implements strong full-disk encryption, encrypting files and all sensitive metadata. Not only files but Snapshots are also encrypted in APFS. APFS unifies the file system encryption software across all Apple platforms. Which encryption methods are available depends on hardware and operating system support, and can vary for Mac, iPhone, iPad, Apple TV, and Apple Watch.

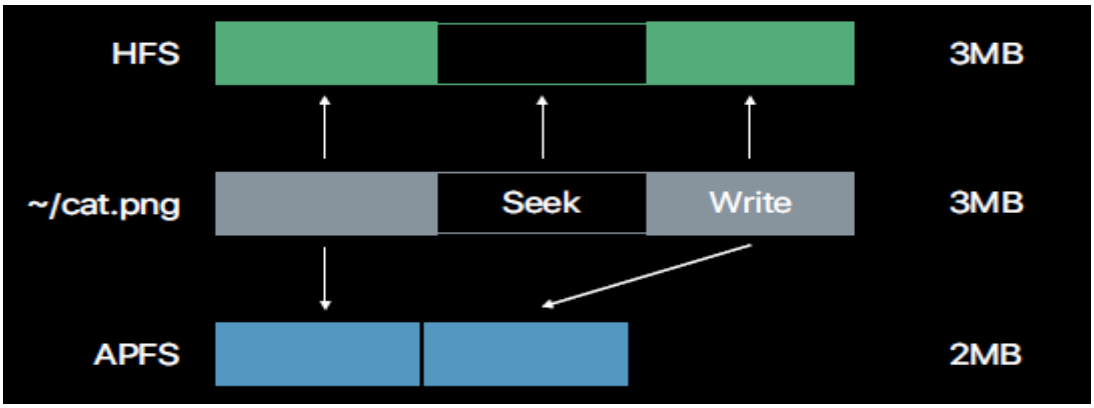
Sparse files

Apple File System supports sparse files - a more efficient way of representing empty blocks on disk. With sparse files, storage is allocated only when needed. This allows the logical size of files to be greater than the physical space occupied on disk. APIs can query both the logical and physical size of sparse files, with functionality to seek through and rewind back to the beginnings of holes and data sections.

For example: If there is a cat file, at first in both HSF and APFS there will be the same physical view and you want to edit that. If the application, for example, decides to Seek past the end of the file and place the addition there. In HFS the file system will fill the space that it seeked after with zeroes. But in APFS there is a sparse file, so while Seeking it will read zeroes but physically it will not take any memory block for that and the remaining file will be placed adjacent to the original file. It not only saves the space abut also makes the OS faster.



Initial physical look of the cat file in both HFS and APFS



Physical look of the file after Seek in both HFS and APFS

Unicode and Normalization Support

Prior to ios 11, APFS stored filenames as non-normalized UTF-8 i.e. If you have the file with one representation as the NFC v/s the one in NFD, both will be treated as two different file names for the purposes of lookup, creation and comparison. To rectify this, the Apple developers introduced two schemes:

1. Native scheme
2. Run-time normalization scheme

Difference Between APFS and HFS Unicode and Normalization Support

The difference between APFS and HFS Unicode and Normalization support is as follows:

APFS	HFS
Case-insensitive and normalization-insensitive by default	Case-insensitive and normalizing by default
Unicode 9.0	Unicode 3.2
Forward compatible	Not forward compatible
Unassigned code points are not allowed	Unassigned code points are allowed
Hash based comparison	Lexicographic comparison
Canonical folding	Simplified folding

Why Unicode 9.0 in APFS?

- It makes the Application truly global as it supports every language that is written and understood by man.
- It supports latest emojis.
- HFS also supports it but sacrifices some aspects of correctness. But APFS does not sacrifices correctness at all for it.

Defragmentation

In APFS there is an *intelligent defragmenter*. It means that the defragmenter in APFS is smart enough to understand that where performance prefers the most, which files are most defragmented and it defragments them first.

It *defragments only HDDs* and only defragmentation happens *only when the machine is idle* so that it doesn't interfere with the user's normal workflow.

Limitations of APFS in macOS Sierra

- **Data volumes only**

  Initially, APFS supported data volumes only and doesn't support booting from APFS. But now macOS High Sierra supports Apple File System for both bootable and data volumes.
- **Time Machine backups with APFS**

  Time Machine backups with APFS are not supported
- **FileVault / Fusion Drive Support**

  FileVault / Fusion Drive Support is still forth-coming
- **Case-sensitive**

  The current format is case-sensitive-only right now.

- **Compatibility**
    - APFS cannot be shared over AFP. So, if the user wants to do file sharing, he has to use SMB instead.
    - OS X Yosemite or earlier will not recognize Apple File System volumes. If the user tries to attach APFS with OS X Yosemite or earlier versions, they will not recognize APFS and as a result the user will inevitably get a dialog that he wouldn't like. *(But the Apple developers did it on purpose – for forward compatibility.)*

Volume Format Comparison of HFS+ and APFS

	Mac OS Extended (HFS+)	Apple File System (APFS)
Number of allocation blocks	2^{32} (4 billion)	2^{63} (9 quintillion)
File ids	32-bit	64-bit
Maximum file size	2^{63} bytes	2^{63} bytes
Time stamp granularity	1 second	1 nanosecond
Copy-on-write		✓
Crash protected	Journaled	✓
File and directory clones		✓
Snapshots		✓
Space sharing		✓
Native encryption		✓
Sparse files		✓
Fast directory sizing		✓

Appendix

AES	AES is Advanced Encryption Standard Instruction Set. The purpose of the instruction set is to improve the speed of applications performing encryption and decryption using the Advanced Encryption Standard (AES).
AFP	The Apple Filing Protocol (AFP) is a proprietary network protocol and part of the Apple File Service (AFS) that offers file services for macOS and the classic Mac OS.
B-tree	A B-tree is a self-balancing tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time.
Clones	Clones allow the operating system to make efficient file copies on the same volume without occupying additional storage space. Changes to a cloned file are saved as deltas, reducing storage space required for document revisions and copies.
Coalescing	Coalescing is the act of merging two adjacent free blocks of memory. When an application frees memory, gaps can fall in the memory segment that the application uses. Coalescing can be done as soon as blocks are freed, or it can be deferred until sometime later (known as deferred coalescing), or it might not be done at all. Coalescence and related techniques like heap compaction, can be used in garbage collection.
Code point	For Unicode, the particular sequence of bits is called a code point.
Cyphertext	Ciphertext is encrypted text. Plaintext is what the user has before encryption, and ciphertext is the encrypted result.
DRAM	A type of physical memory used in most personal computers. The term dynamic indicates that the memory must be constantly

	refreshed (reenergized) or it will lose its contents. RAM (random-access memory) is sometimes referred to as DRAM.
Encryption	Encryption is the most effective way to achieve data security. Encryption is basically, the conversion of electronic data into another form known as ciphertext, which cannot be easily understood by anyone except authorized parties.
Encryption key	An encryption key is a piece of information (a parameter) which specifies the transformation of plaintext into ciphertext, and vice versa for decryption algorithms.
Enterprise storage	There are two types of enterprise storage SAN (Shared Area Network – A shared network of storage) and NAS (Network Area Storage – A file system shared over a network)
Field	The record of a file in a file system is divided into different files called field. These fields may contain any information about the file such as its name, modification date, etc.
File encryption	File Encryption means providing security for files that are present on any storage media. Files that are usually not meant to be sent through network are stored locally on the storage media and are encrypted. These files are only temporarily decrypted while being used and then they are encrypted again after we finished using them. Encrypting stored files prevents others from reading, copying, or deleting encrypted files. Thus, the encrypted files cannot be accessed for reading by unauthorized users.
File locking	File locking is a mechanism that restricts access to a computer file by allowing only one user or process to access it in a specific time.
Hard link	A hard link is a directory entry that associates a name with a file on a file system.

Hard links for time machine	Hard Links are incremental backups but in effect they are full backups.
Inode	The inode is a data structure in a Unix-style file system that describes a filesystem object such as a file or a directory. Each inode stores the attributes and disk block location(s) of the object's data.
Latency	It is the amount of time between you telling your computer to do something and when it happens. It has a significant effect on performance.
NAND flash	NAND flash memory is a type of non-volatile storage technology that does not require power to retain data. An important goal of NAND flash development has been to reduce the cost per bit and increase maximum chip capacity so that flash memory can compete with magnetic storage devices like hard disks.
Rigid data structure	Rigid data structure is an implicit data structure consistent with a fixed family of partial orders.
SMB	When a user connects from a Mac to another computer using file sharing, the Mac automatically tries to use the Service Message Block (SMB) protocol to communicate.
Sparse file	A sparse file is a type of computer file that attempts to use file system space more efficiently when the file itself is mostly empty.
SSD and HDD	The traditional spinning hard drive (HDD) is the basic nonvolatile storage on a computer. HDD has a lot of space in it but it is slow as it uses magnetic disk to store data. An SSD makes the processing and all work faster like a lightning bolt. In an SSD, instead of a magnetic coating on top of platters, the data is stored on interconnected flash memory chips that retain the data even when there's no power present. These chips can also be permanently installed on the system's motherboard. But if we compare the two

	storage devices of the same size the SDD is costly but makes processing faster and an HDD has low cost but it has a lot of storage. If we take both storage devices of the same cost, SDD has very less storage space as compared to the HDD.
SSD Write coalescing	This involves gathering several short writes to adjacent SSD sectors to turn them into a single long write from the buffer into the NAND flash. One large write is less taxing to the chip than are several small writes.
Time machine	Time Machine is the best new feature added to Mac OS X. It is a backup and recovery system that Apple product users use.
Write coalescing	Every erase/write cycle will weaken NAND flash. In order to reduce the erase/write cycles, drive designers employ a volatile data buffer (similar in architecture to main system memory) as a temporary collection and staging area for data. This RAM can reside inside of the drive's controller or in external DRAM chips mounted on the drive's printed circuit board. This RAM is used to gather several short writes into one larger write that is then sent to the NAND. This minimizes the amount of writes to the NAND and also increases the drive's performance. This process is called "write coalescing."

References

- https://developer.apple.com/library/content/documentation/FileManagement/Conceptual/APFS_Guide/FAQ/FAQ.html
- https://forums.developer.apple.com/thread/50148
- https://www.cultofmac.com/435718/apfs-new-apple-file-system/
- https://arstechnica.com/gadgets/2016/06/digging-into-the-dev-documentation-for-apfs-apples-new-file-system/
- https://developer.apple.com/library/content/documentation/FileManagement/Conceptual/APFS_Guide/Features/Features.html
- http://www.zdnet.com/article/wwdc-2017-macoss-new-file-system/
- http://en.cppreference.com/w/cpp/filesystem/canonical
- https://en.wikipedia.org/wiki/Apple_Filing_Protocol
- https://en.wikipedia.org/wiki/Sparse_file
- https://en.wikipedia.org/wiki/Key_(cryptography)
- https://en.wikipedia.org/wiki/AES_instruction_set
- https://en.wikipedia.org/wiki/File_locking
- https://en.wikipedia.org/wiki/Defragmentation
- https://arstechnica.com/gadgets/2007/10/mac-os-x-10-5/14/
- https://superuser.com/questions/588414/why-does-os-x-time-machine-create-directory-hardlinks
- http://pondini.org/TM/Works.html
- https://en.wikipedia.org/wiki/Hard_link
- https://askubuntu.com/questions/210741/why-are-hard-links-not-allowed-for-directories
- https://www.backblaze.com/blog/apfs-apple-file-system/
- http://www.informit.com/articles/article.aspx?p=606585&seqNum=4
- http://dtrace.org/blogs/brendan/2011/05/11/file-system-latency-part-1/
- https://peterwitham.com/apple/apple-announces-apple-file-system-apfs/
- https://en.wikipedia.org/wiki/Field_(computer_science)
- https://en.wikipedia.org/wiki/Inode
- https://arstechnica.com/gadgets/2016/09/macos-10-12-sierra-the-ars-technica-review/7/
- http://thessdguy.com/tag/write-coalescing/