VIRTUALSYNC

TERABYTES ON DEMAND

BUNGiE

Overview

- Why we decided to develop VirtualSync
- Results we see today
- How it works
- Challenges


Destiny 2

BUNGiE

Destiny 2

- Destiny 2 is an action MMO
- Highly detailed, realistic art style
- New expansions add content every few months
- Source content stretching back more than a decade

- This all adds up to a massive amount of source content

BUNGiE

Content Files at Bungie

- Perforce
  - Source content — 8.6m files, ~8.06 TB per branch
  - Source code — 56k files, ~1.3 GB per branch
- Network Storage
  - Binary build artifacts — 11k files, ~9 GB
    - Tool and game exes, etc

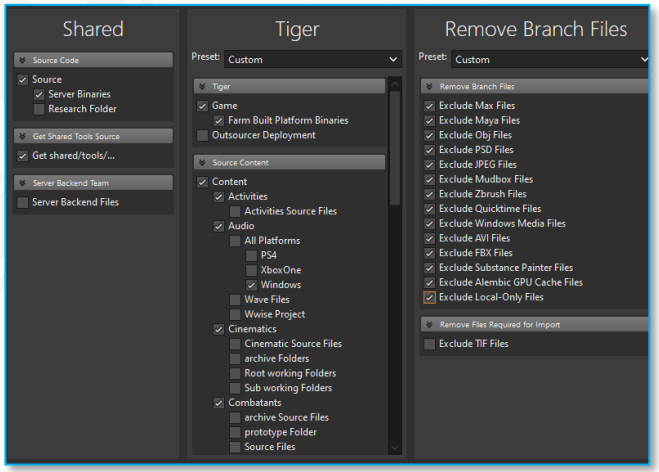
BUNGiE

Multiple Large Branches

- Perforce branches used to separate work by release date
  - Destiny Season 17, Season 18, the next expansion, etc
- 18+ branches with active development currently
- Often users require more than one branch locally
  - Content Creator: Primary dev branch + previous dev branch
  - Engineer / Support: Primary dev branch + many branches for support

Multiple Large Branches: Implications

- Users required massive HDD(s)
  - Slower spinners, no SSD/NVMe data drives
- Tool to shrink Perforce enlistment
  - Exclude depot paths, file extensions
  - Shrinks source content 8TB => ~2-4TB
  - User-facing complexity, error prone
  - Generates ~640 mapping lines per branch

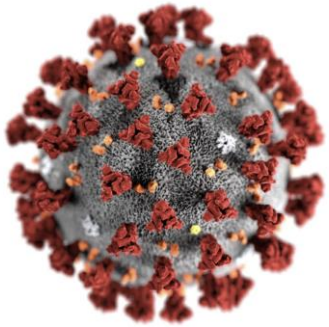


BUNGiE

Multiple Large Branches: Syncing in 2019

- User's required branch set changes as releases ship
  - First branch sync takes many hours
  - Often must de-enlist and delete old branch before enlisting new
- Incremental sync can still take an hour or more
  - Most syncs scheduled overnight
  - Mid-day syncs sometimes required to get critical fixes
- Users spot-sync files/directories to #head as workaround

Work From Home



- Slower and less reliable networks
- Laptops with massive, slow USB HDDs
- Incremental syncs take many hours or many days
- First syncs could take an entire weekend or longer

BUNGiE

Birth of VirtualSync

- Experimented before pandemic to solve in-office issues
- Skeptical of performance for remote users
  - Tool hitching and delays
- Investigated Windows mini-filter file system driver
- Stumbled across Windows Projected File System (ProjFS)
  - Implements the mini-filter file system driver for you
  - Handles most common mixed local + virtual file system behavior for you
  - C# API available
  - Used in ‘VFS for Git’

The Results

BUNGiE

VirtualSync: Usage Today

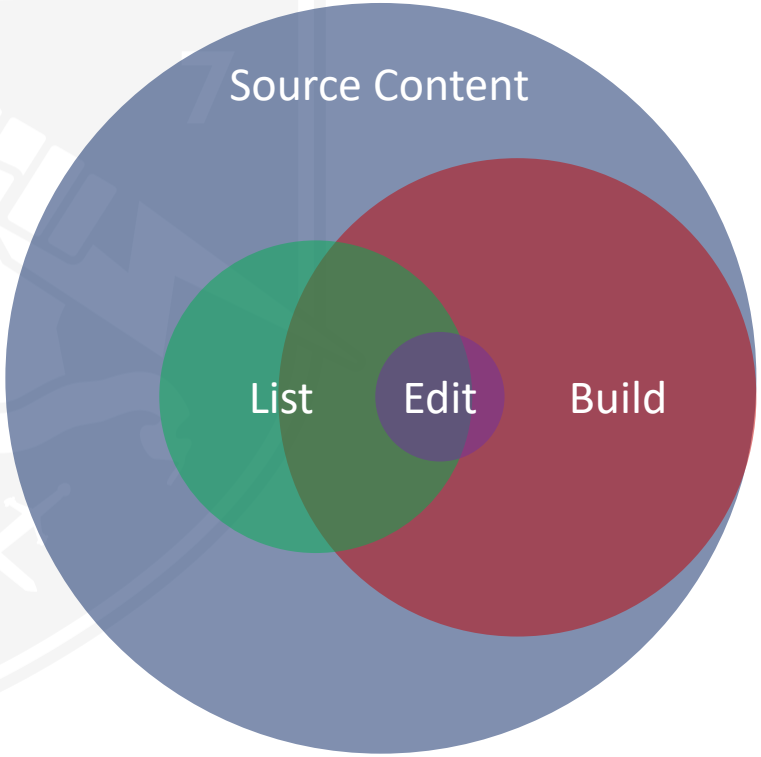
- 816 machines with 2,499 content branches enlisted

- Median branch enlistment size: 7.09 TB
- Median local HDD size: 1.74 GB (Max: 648 GB)!
  - Only 10 branches have more than 7% of their enlistment locally
- Savings across all machines: 16.03 PB

- Median new branch sync: 0:03:39.4
- Median incremental sync: 0:03:19.3

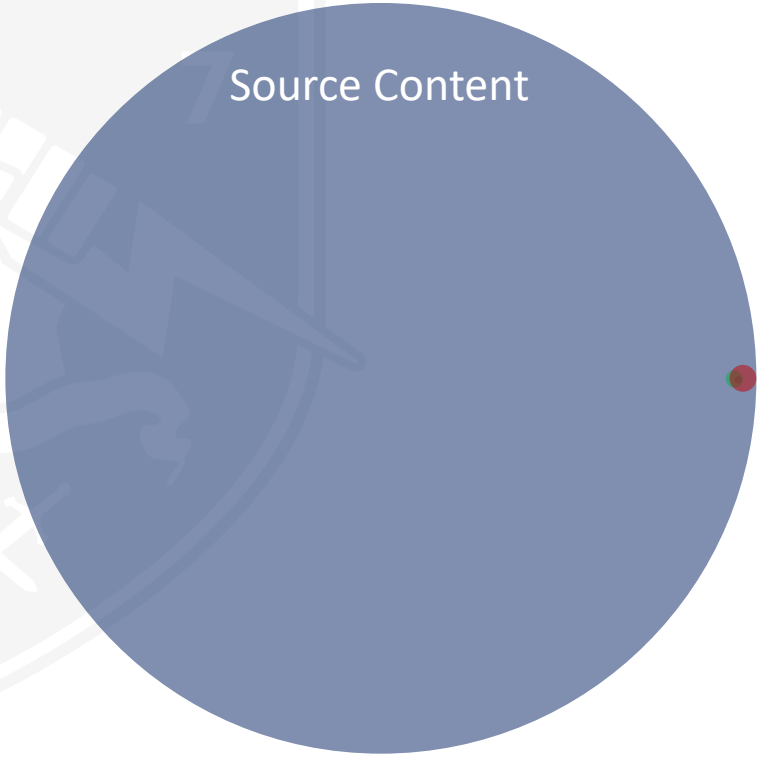
VirtualSync: Usage Today

- Only 1.74 GB out of ~7 TB local?
- Primary reasons files are read
  - List assets for selection
  - Edit an asset
  - Local content building



BUNGiE

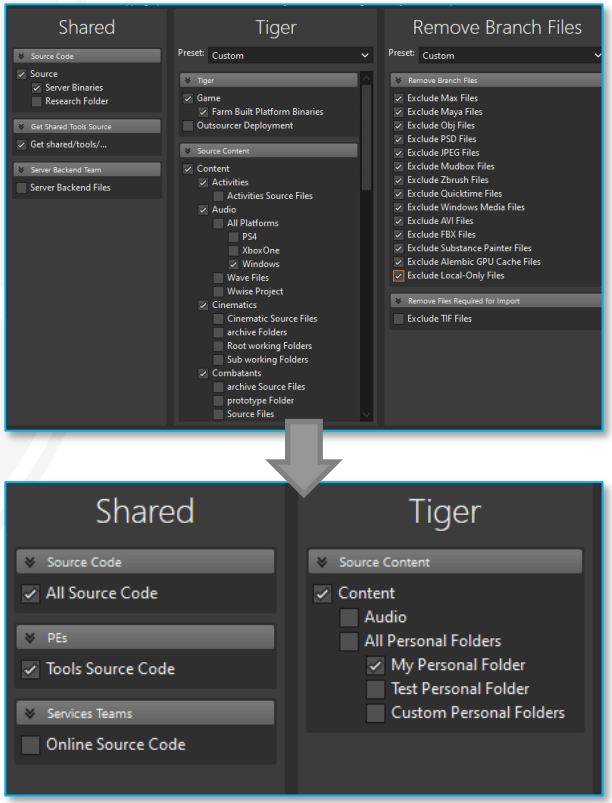
VirtualSync: Usage Today



BUNGIE

VirtualSync: Benefits

- Syncing is FAST
  - …even when remote!
  - …even new branch syncs!
- Drastically reduced HDD storage needs
  - We are switching to using SSD/NVMe data drives
- Simplified Perforce client
  - No more user-facing complex branch enlistment options
  - ~640 -> ~10 mapping lines per branch
  - Most Perforce interactions/queries significantly faster



BUNGIE

VirtualSync: Benefits

- Stronger guarantees that local state matches Perforce
  - Several ways local state can end up out-of-sync with traditional Perforce sync
  - Example:
    - User makes a file writable manually and modifies it
    - User or something else makes the file read-only
    - Perforce sync will only update that file if its revision change
- Potentially enable running our tools in containers, possibly cloud

The Tech

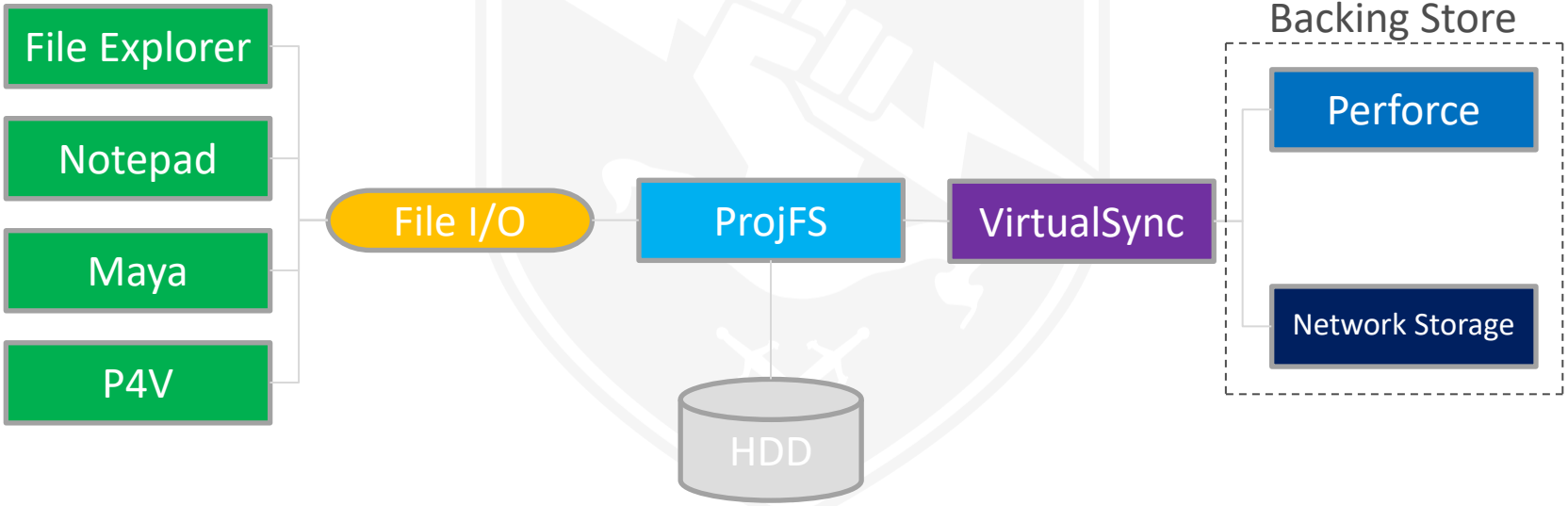
BUNGiE

ProjFS: Requirements

- Requires Windows 10 v1809 or higher
- Optional Windows component
  - Can be enabled via Windows UI, PowerShell, etc
- Relatively fast access to file metadata and data
  - No progress reporting, online vs offline user-visible state, etc
  - Cloud Files API for slower backing stores

BUNGiE

ProjFS: How it works



BUNGiE

ProjFS: How it works

- Provider is a Windows application
- Provider asks ProjFS to initialize its virtual file system, specifying…
  - Local directory to act as virtual file system root
  - Object implementing callbacks
    - Start/Get/End directory enumeration: return metadata for directory children
    - GetFileDataCallback: return file bytes
- Your virtual branch is up and running!

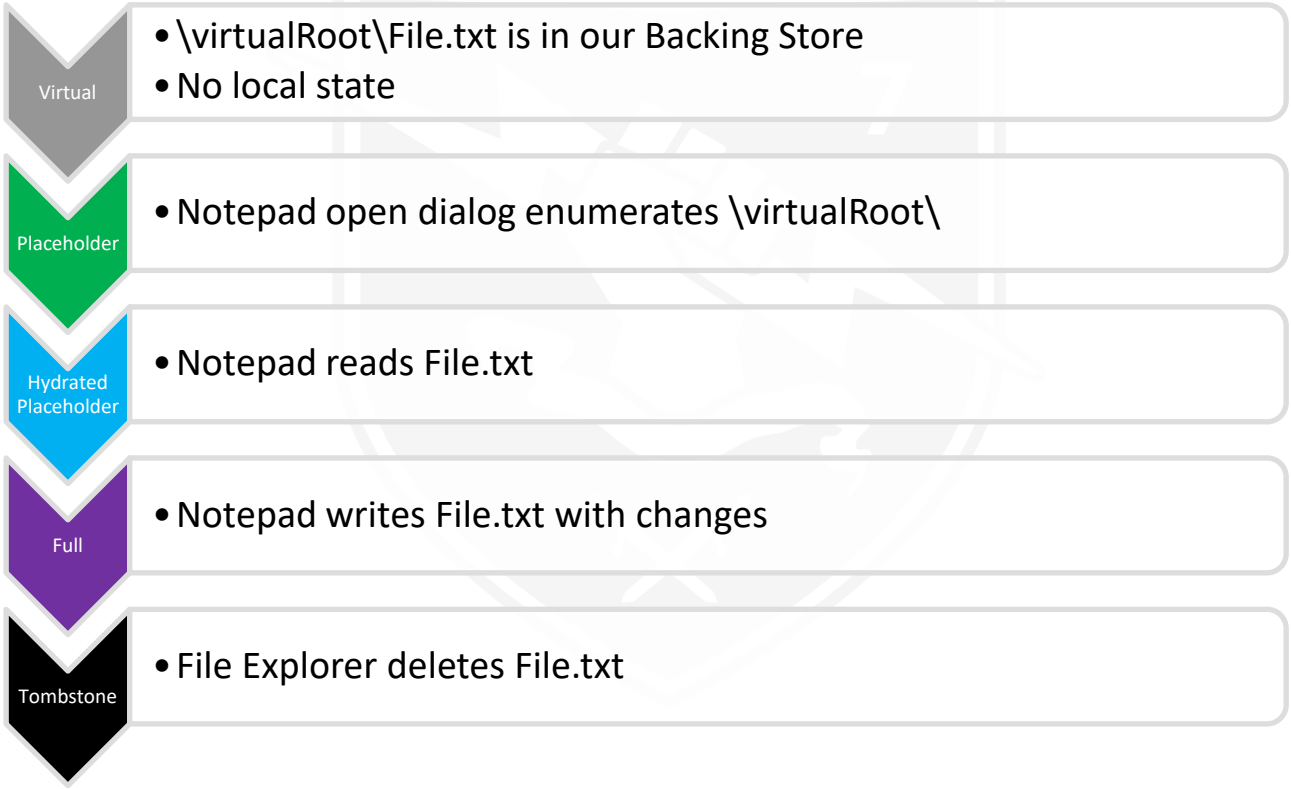
BUNGiE

ProjFS: How it works

- Callbacks are called in response to application actions
  - Callbacks given requesting application process name + id
- ProjFS caches metadata and file contents as reparse points
  - After first access/read, data has local file performance
  - Non-volatile
  - Protected against most user interactions while Provider is offline

BUNGiE

ProjFS: File States



BUNGIE

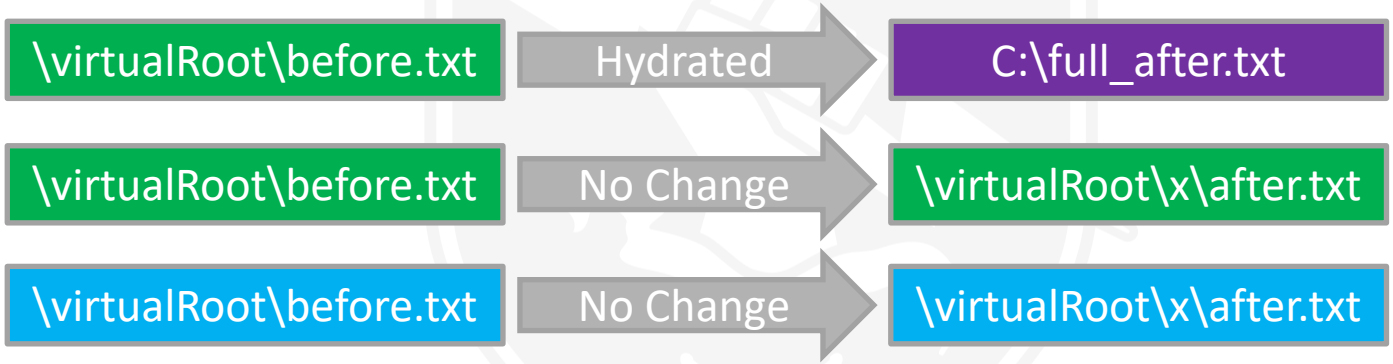
ProjFS: Directory States

Virtual	• Exists in Backing Store • No local state
Full	• Locally created • Does not show Virtual items
Placeholder	• Shows Virtual items
Tombstone	• Exists in Backing Store • Locally deleted

BUNGiE

ProjFS: Interesting Cases

- Renaming or moving Placeholder files



- Renaming Placeholder directories not supported
  - Can copy + delete to work around

VirtualSync: Sync Process

- What happens when the Backing Store changes?
- Update virtual Perforce files
  - P4 sync **–k** …
  - Only updates client's #have state, no local file system interactions
- Network Storage files
  - Provide a new binary build artifact manifest to VirtualSync to host virtually

VirtualSync: Post-Sync Process

- Users may have Placeholders/**Tombstones** that are now stale!

- Clear VirtualSync in-memory caches that may now be stale
- Clear ProjFS's Negative Path Cache

- Find and fix stale local file system state
  - A few options for how to go about finding stale state
  - We enumerate and reconcile all **local** file system state…

BUNGiE

VirtualSync: Post-Sync Process

- How to enumerate local file system state without triggering ProjFS?
- New FindFirstFileEx() flag: FIND_FIRST_EX_ON_DISK_ENTRIES_ONLY
- Returns enough to classify file/directory state
  - ‘Not found’ can mean Virtual or ‘does not exist’
- Will enumerate **Tombstones**
- Can still trigger callbacks in certain situations, but avoidable

BUNGiE

VirtualSync: Post-Sync Process

- Update Hydrated Placeholders, only if they are stale
- Metadata from callbacks (cached by ProjFS) include versioning info
  - ProviderId : byte[128]
    - P4 = 1
    - Binary Artifact = 4
    - Directory = 3
  - ContentId : byte[128]
    - P4 = #have revision
    - Binary Artifact = md5 hash from manifest
    - Directory = 3
- ProjFS UpdateFileIfNeeded()

BUNGiE

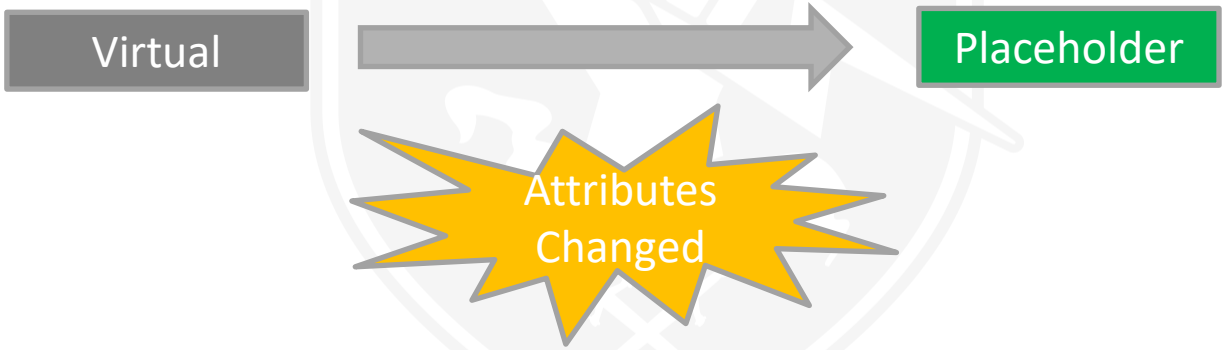
VirtualSync: Post-Sync Process

- Convert Full directories to Placeholders
- Clean up unnecessary local state
  - Convert Placeholders to Virtual
  - Convert read-only Full files in Backing Store to Virtual
  - Remove **Tombstones** if no longer in Backing Store

Challenges

Challenges: Filesystem Notifications

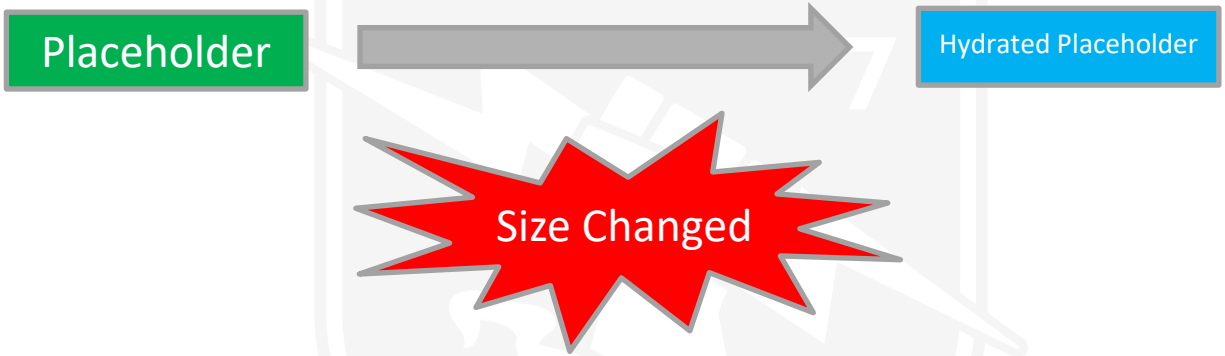
- Many of our tools watch for file system notifications
- ProjFS state changes produce new/additional notifications



  - Not enough information to filter
  - Triggers minimal tool work, not a big deal for us

BUNGiE

Challenges: Filesystem Notifications



- Can trigger disruptive or destructive tool actions (reload/refresh)
- Can safely filter with minimal effort:
  - Separate filesystem watchers to isolate Size notifications
  - When notified, check file state – if Hydrated Placeholder then ignore

BUNGiE

Challenges: Frequent Enumeration

- ProjFS does not cache list of enumeration results, so calls frequently
- Prohibitively expensive to query perforce/file share each callback

- Started with queries + an LRU cache of results
- Updated Binary Build to produce artifact manifest with metadata
- Perforce: Asynchronously build metadata cache for all files
  - ~0.5 – 3 minutes to build
  - While building, we fall back to query + cache
  - After post-processing, ~300MB for metadata + checksums for 8 million file
  - After cache built, only talk to Perforce to fetch file data

Challenges: Perforce Text-Typed Files

- Windows clients CR LF
- Linux P4 server LF (Unix line endings)
- Perforce reports file size and checksum based on server version
- Must supply correct file size with directory enumeration callback

- We ‘p4 print’ file bytes without client newline conversion to match
- Thankfully, few text-typed files and no tool issues with Unix LF

Challenges: HDD Footprint Creep

- Post-sync process only removes changed Hydrated Placeholders
- The set of hydrated files users need changes over time
- This results in HDD usage growing over time

- VirtualSync now tracks Hydrated Placeholder access via ProjFS notification
- Post-sync reconcile removes Hydrated Placeholders not accessed in X days (user configurable)

BUNGIE

Challenges: ContentWatcher

- Existing tool that tracks content file metadata
  - Read-only vs writable, checksum, file-to-file references
- Queried extensively by other tools
- First run in a new branch requires a full enumeration to set up state
- Filesystem notifications and NTFS journal used after that
- File-to-file references are shared, keyed by file checksum

Challenges: ContentWatcher Problem #1

- First run processing would create Hydrated Placeholders
- File data only to compute checksums and file-to-file references

- File-to-file references should already be in shared db
- VirtualSync exposes interface for tools to request file checksums
  - Virtual / Placeholder -> In memory metadata cache (from p4 or binary artifacts)
  - **Tombstone** -> Doesn't exist
  - Full -> computed
- First run processing now takes ~30 minutes, down from 3+ hours!

BUNGiE

Challenges: ContentWatcher #2

- First run still needs to enumerate all files, creating Placeholders
- ContentWatcher now queries VirtualSync directly for file list
  - Properly mixes virtual and local-only state
  - Only used in this case
    - Other use cases should just query ContentWatcher for that information

BUNGiE

Challenges: ContentWatcher #3

- Maintains state with NTFS journal and Filesystem notifications
- ProjFS introduced several new edge cases and event patterns
- Beyond ones discussed earlier, like
  - Removing **Tombstone** (during post-sync reconcile)
    - Could mean the file/directory is gone, or it could be back again (Virtual)
  - Renaming a Full file to the name of a **Tombstone**

- Wrote a unit test system that mimics Filesystem notifications and NTFS journal events for every case
- Updated ContentWatcher to process these events correctly

BUNGiE

Challenges: Remote Debugging

- File queries over network shares return different results
  - File attributes are different
    - FILE_ATTRIBUTE_REPARSE_POINT and FILE_ATTRIBUTE_SPARSE are not returned
  - WIN32_FIND_DATA.Reserved0 (reparse tag type) is always 0
  - Can’t fetch reparse point data
    - fsutil reparsepoint query \\remote_maching\placeholder.txt
    - DeviceIoControl(FSCTL_GET_REPARSE_POINT)
  - Can’t classify the state of a file/directory remotely
- ProjFS methods require the virtualization instance

BUNGIE

Challenges: Remote Debugging

- VirtualSync service uses HttpListener to publish a website
  - Access with http://<machineName>/virtualSync/
- Includes
  - Current detailed status / state
  - Recent API/throughput statistics
  - Recent errors/warning
    - This typically include next-step information and links to internal wiki articles
  - Last sync information
  - Many interactive diagnostic tools

BUNGiE

Challenges: Remote Debugging –Branch Status

VirtualSync Status - [redacted] c:\bungie\v610_cms\tiger (Up Refresh)

Name	c_bungie_v610_cms_tiger
Local Root	c:\bungie\v610_cms\tiger
Actual Root	c:\bungie\v610_cms\tiger
Excluded Relative Paths	.\build_reports\...; .\cache\...; .\ContentManifest\...; .\datamine\...; .\packages\...; .\temp\...; .\tfcache\...
Branch Start Time	10/21/2021 1:02:50 PM
Branch State	Healthy
Sync Description	BuildSet: bsid00075147; game 98878.21.10.20.1800 (CL 5685226); content 98878.21.10.20.1800-0 (uber_bvt - CL 5685430); online 116003.21.10.20.1800 (CL 5
Sync Occurred	10/21/2021 12:28:47 PM
Perforce Server	[redacted]
Perforce Client	[redacted]
Perforce User	[redacted]
Logs	\\[redacted]\virtualSync_v610.cms_cbungie\logs\20211021_130249 (Events Log, Perf Csv) [Force Log Flush]
Settings	Adjust Branch Settings
Diagnostic Tools	Diagnostic Tools

Branch Operations

Running Operation	(None)
Running Operation Started	
Last Operation	CleanLocalState(minimizedLocalDiskFootprint: False, stompPerforceWritableFilesToVirtualState: False, stompFileOverrideWritableFilesToVirtualState: False, I
Last Operation Completed	10/21/2021 1:54:56 PM taking 00:00:00.7912599
Last Operation ResultCode	[Details] Success

Current Status

Virtual FileShare Files	10487
Comprehensive Metadata Cache	Enabled (consume memory to speed up queries and enumerations)
Comprehensive Metadata Cache Status	Actively In Use
Comprehensive Metadata Cache Details	Metadata cached for (636,266 p4 + 907 fs) directories; (3,262,810 p4 + 10,487 fs) files; total referenced file bytes: (6.94 TB p4 + 9.24 GB fs
Directory Cache	15 / 65536 (0.02%)
Service Lifetime Placeholders Created Count	21
Service Lifetime Files Hydrated Count	15
Service Lifetime File Bytes Hydrated	195.91 MB
Service Lifetime Errors	0
Last Error	N/A
Working Set Size	922.32 MB
Page File Size	944.67 MB

BUNGIE

Challenges: Remote Debugging – Branch Status

Last Errors/Warnings and Major Events

10/21/2021 1:54:56 PM	Message	CleanLocalState completed successfully.
Lifetime 0 Warning / 0 Error / 0 Critical Events		

Branch Statistics (since 10/21/2021 1:42:50 PM, duration: 00:17:10.9741025)

	Count	Duration	Details
Fetch File Data	0 bytes	00:00:00	0 file(s)
Directory Enumeration	30	00:00:00.4006613	cache hits: 26 misses: 4
Get Placeholder File Info	25	00:00:00	cache hits: 25 misses: 0
Query FileName	4	00:00:00.0000155	cache hits: 1 misses: 3

Checksum Statistics (since 10/21/2021 1:42:44 PM, 00:17:17.1527740)

[Statistics for all 3 branch(es) hosted by same grouping (grouping key: [redacted])]

	Count	Duration	Details
In progress	0		(queued and/or being computed/retrieved)
Computed	2	00:00:00.0033613	
From Perforce	5	00:00:00.3638641	Avg Request Queue Time: 00:00:00.0443241; Batches: 4
From Local Disk(s)	2	00:00:00.0033613	Processed: 0 bytes
From FileShare(s)	0	00:00:00	Processed: 0 bytes
Aborted Requests	0		(requesting client disconnected)
WCF Post Result Overhead	7	00:00:00.0005978	(high values here likely indicate blocking in requesting app's WCF callback that should be fixed)

Perforce Statistics (since 10/21/2021 1:42:44 PM, 00:17:17.1527740)

[Statistics for all 3 branch(es) hosted by same grouping (grouping key: [redacted])]

	Count	Duration	Details
Connection Threads	8		(Threads that have created Perforce wrappers)
Connections Attempted	13	00:00:01.5776479	(Number of connections attempted + total time connecting)
Connections Succeeded	13		
Connections Disconnected	5		
Api Calls Attempted	42	00:02:26.8513593	(Number of api calls attempted + total time waiting on the api)
Api Calls Succeeded	42		

BUNGiE

Challenges: Remote Debugging – File Explorer

[Show entry data source] ,

Make All Virtual and Revive Tombstones | Make All Placeholders | Hydrate All Files (5.88 KB)

Name *	Size	LastWriteTime	State	AttributeDescription	Attributes	Reserved0	CachedDataVersion	
..(parent directory)								
local_only_subdirectory			Full	Directory	00000010	00000000		MakePl
virtual_subdirectory_w			Virtual	Directory, Virtual	00010010	00000000		
virtual_subdirectory_x			Placeholder	Directory, ReparsePoint, RecallOnDataAccess	00400410	9000001C		MakeVi
virtual_subdirectory_y			Placeholder	Directory, ReparsePoint, RecallOnDataAccess	00400410	9000001C		MakeVi
virtual_subdirectory_z			Tombstone	Directory, ReparsePoint, Hidden, System	00000416	A0000022		Revive
local_only.txt	0 bytes	10/21/2021 1:45:37 PM	Full	Archive	00000020	00000000		Reques
p4_file_a.txt	64 bytes	10/20/2021 3:58:43 PM	Virtual	Virtual	00010000	00000000	[PerforceRev] 1	MakePl
p4_file_b.txt	5.81 KB	10/20/2021 3:59:07 PM	Placeholder	ReparsePoint, ReadOnly, RecallOnDataAccess, Sparse, Archive	00400621	9000001C	[PerforceRev] 1	Hydrate
p4_file_c.txt	0 bytes	10/21/2021 1:46:34 PM	Tombstone	ReparsePoint, Hidden, System, Archive	00000426	A0000022		Revive
p4_file_c_renamed.txt	1.13 KB	10/20/2021 3:59:53 PM	HydratedPlaceholder	ReparsePoint, ReadOnly, Sparse, Archive	00000621	9000001C	[PerforceRev] 1 (moved from content\personal\bmoro\virtual_sync_examples\p4_file_c.txt)	MakeVi
p4_file_d.txt	0 bytes	10/21/2021 1:44:35 PM	Tombstone	ReparsePoint, Hidden, System, Archive	00000426	A0000022		Revive
6 files	7.00 KB							

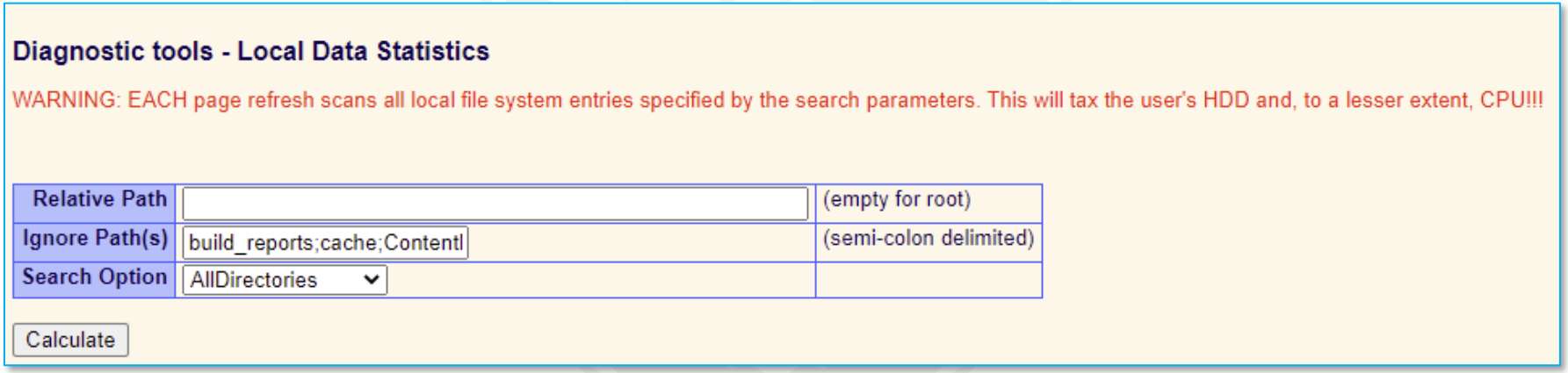
Action Details

MakePlaceholder	Cache file metadata locally (Virtual -> Placeholder)
Hydrate	Cache file contents locally (Virtual/Placeholder -> HydratedPlaceholder)
MakeVirtual	Remove locally cached state (Placeholder/HydratedPlaceholder/Full -> Virtual)
Revive	Convert a locally deleted file back to virtual (Tombstone -> Virtual)
MakePlaceholderDirectory	'Full' directories cannot have virtual items under them. Converting it to a placeholder enables it to contain virtual items.

For more information about the various terms used here see VirtualSync Terminology

BUNGiE

Challenges: Remote Debugging – Local State



BUNGiE

Challenges: Remote Debugging – Local State

Status	Completed
Duration	00:00:00.1138836

Full size of virtual branch under [root] if sync'd/fetched normally is approximately 6.95 TB (7,643,129,362,909 bytes) across 3,273,297 files.

VirtualSync is currently saving your local HDD roughly 6.95 TB across 3,273,256 files (note: full files are considered not virtual for the purposes of this calculation)!

	Virtual Paths Count	Virtual Paths Size	Branch Excluded Paths Count	Branch Excluded Paths Size	Total Count	Total Size
Directories	3127		0		3127	
Placeholder Count	27		0		27	
Hydrated Placeholders	41	199.66 MB	0	0 bytes	41	199.66 MB
Full Files	42	18.98 KB	0	0 bytes	42	18.98 KB
Local File Totals	110	199.68 MB	0	0 bytes	110	199.68 MB

Hydrated placeholders by extension (ordered by size)

Hydrated Extension	Count	Size
.zip	1	177.35 MB
.tfb64	3	13.31 MB
.dll	1	4.27 MB
.exe	4	3.73 MB
.tft	8	976.20 KB
.xml	7	24.13 KB
.cs	4	5.48 KB
.bat	5	5.11 KB
.txt	7	4.58 KB
.config	1	868 bytes
.signed	0	0 bytes
.watermark	0	0 bytes

BUNGiE

Challenges: Remote Debugging – ‘Clean’

Diagnostic tools - Clean Branch

Branch State	Healthy

Branch Operations

Running Operation	(None)
Running Operation Started	
Last Operation	CleanLocalState(minimizedLocalDiskFootprint: False, stompPerforceWritableFilesToVirtualState: False, stompFileOverrideWritableFilesToVirtualState: False, localDir
Last Operation Completed	10/21/2021 1:54:56 PM taking 00:00:00.7912599
Last Operation ResultCode	[Details] Success

Minimize Local Disk Footprint	False	Removes locally cached file contents to reduce local HDD usage. These files
Stomp FileOverride-backed (Binary Output) Writable Files To Virtual State	False	Removes local writable files if they exist in fileOverrides (binary outputs), mal
Stomp Perforce-backed Writable Files To Virtual State	False	WARNING: Removes local writable files if they exist in perforce, making then
Local State Removal Mode For Empty Or Virtual Directories	DoNotRemove	**WARNING: Not recommended while Rasputin is running!** Removes unne

Clean

WARNING: Running this can take anywhere from ~10s to a couple minutes depending on the amount of local content and options.

BUNGiE

Challenges: Remote Debugging – ‘Clean’

Clean Branch is RUNNING (page will auto refresh in ~1 seconds to show progress)...

Result	IN PROGRESS: Reconciling local HDD state...
Duration	00:00:06.0415943

Parameters

Delete Unchanged Metadata-only Placeholders	True
Delete Unchanged Hydrated Placeholders	False
Stomp Perforce Writable Files to Virtual State	False
Stomp FileOverride Writable Files to Virtual State	False
Local State Removal Mode for Empty or Virtual Directories	DoNotRemove
Hydrated Placeholders Days After Last Access To Expire	5
Has Backing Store Delta Info	False

Statistics

Errors	0
Directories	4777 placeholder directories; 40 full directories converted to placeholder; 0 directory tombstones deleted; 0 locally empty directories deleted/made virtual
MetadataOnly Placeholders	0 updated; 347 deleted
Hydrated Placeholders	654 updated; 6 (2.31 MB) deleted (6 expired)
Tombstones	0 file tombstones deleted; 0 files and 0 directories left alone due to pending delete actions
Full Files	305 processed - 305 (4.91 GB) remain (0 removed due to being deleted in backing store)
ReadOnly Files	0 (0 bytes) read-only files deleted
Writable Files	0 (0 bytes) writable files deleted
Other Reparse Points	0 ignored (0 corrupt placeholder repaired; 0 with invalid reparse tag removed)

BUNGiE

Challenges: Remote Debugging – ‘Clean’

Clean Branch is done!

Result	Success
Duration	00:00:15.5324866

Parameters

Delete Unchanged Metadata-only Placeholders	True
Delete Unchanged Hydrated Placeholders	False
Stomp Perforce Writable Files to Virtual State	False
Stomp FileOverride Writable Files to Virtual State	False
Local State Removal Mode for Empty or Virtual Directories	DoNotRemove
Hydrated Placeholders Days After Last Access To Expire	5
Has Backing Store Delta Info	False

Statistics

Errors	0
Directories	14336 placeholder directories; 52 full directories converted to placeholder; 0 directory tombstones deleted; 0 locally empty directories deleted/made virtual
MetadataOnly Placeholders	0 updated; 1539 deleted
Hydrated Placeholders	1697 updated; 13 (2.72 MB) deleted (13 expired)
Tombstones	0 file tombstones deleted; 0 files and 0 directories left alone due to pending delete actions
Full Files	433 processed - 433 (4.94 GB) remain (0 removed due to being deleted in backing store)
ReadOnly Files	0 (0 bytes) read-only files deleted
Writable Files	0 (0 bytes) writable files deleted
Other Reparse Points	0 ignored (0 corrupt placeholder repaired; 0 with invalid reparse tag removed)

BUNGiE

Challenges: Remote Debugging –Task Manager

Sampled at 10/21/2021 1:58:50 PM

System Boot Time (system-local)	10/21/2021 1:02:20 PM (00:56:34.2626001 elapsed)
Last System Resume	(no resumes since service started 00:56:12.2145296 ago)

Available RAM (standby+free+zero)	52.92 GB / 63.91 GB (82.80%)
In Use RAM	11.00 GB / 63.91 GB (17.20%)
PageFile Commit	14.68 GB / 67.91 GB (21.62%)
Cached (standby+system working set)	8.64 GB
Kernel	516.48 MB paged + 361.75 MB non-paged = 878.24 MB

Drive	Available	Total	Avg Queue	Avg Read	Avg Write
C:	623.31 GB (66.05%)	943.72 GB	0	0 bytes	0 bytes
D:	5.39 TB (98.67%)	5.46 TB	0	0 bytes	0 bytes
E:	5.39 TB (98.67%)	5.46 TB	0	0 bytes	0 bytes

(note: 'Avg CPU %' values computed as a delta between this and the previous sample (00:01:24.6963224 = 10/21/2021 1

Process Id	Name	** Working Set ↑ **	Page File Bytes	Average Cpu	Read IO Per Second	Write IO Per Second	
8460	Rasputin.exe	6.30 GB	6.52 GB	0.0%	0 bytes	0 bytes	"C:\bungie\v610
3820	VirtualSyncService.exe	922.11 MB	944.62 MB	0.0%	30 bytes	1.03 KB	C:\Bungie\Virtu
10760	BungieLauncher.exe	263.43 MB	296.37 MB	0.0%	0 bytes	0 bytes	"C:\bungie_glo

HttpListener for Inspection

- VirtualSync is our 2nd application to use HttpListener in this way
- Very powerful, easy to use
- Ability to send links to specific tools/pages very useful
  - Support request emails have link to related VirtualSync branch page
- Ability to link to internal wiki for help documents

- Highly Recommended!

Development

BUNGiE

Development

- Primary Development: Brandon Moro and Danny Frisbie
- Additional Support
  - Core Systems and Workflow Team
  - Bungie Perforce Admins
  - Bungie IT
- Big thanks to all of our Alpha and Beta testers!

BUNGiE

Development

- 7/15/2020 – Initial Prototyping and Development Begin
  - Building core VirtualSync service application
  - Updating branch enlistment, sync, content file monitoring tools
- 8/31/2020 – Alpha
  - Gathering and investigating usage data across core workflows
  - Bug fixes, optimizations, inspection improvements
- 10/15/2020 – Open Beta
  - More gathering and investigating usage data across all workflows
  - More Bug fixes, optimizations, inspection improvements
- 2/8/2021 – Live
  - Default method of enlisting a branch

BUNGiE

Conclusion

- ProjFS enabled us to quickly build and deploy our solution
- VirtualSync
  - Solved our major remote sync and branch enlistment issues
  - Simplified branch enlistment and switching active dev branch sets
- Secondary benefits
  - Content file processing / monitoring / checksums faster in virtual branches
  - Switching to SSD / NVMe data drives

BUNGiE

Conclusion

- People love it
- Unsolicited user feedback
  - “VirtualSync is goddamn sorcery. Amazing feature.”
  - “I JUST USED IT AND IT WAS SO MAGICAL!!!”
  - “This tech is hot! Thank you so much for everyone involved in making this happen”

BUNGIE®

IS HIRING

GDCEVENT@BUNGIE.COM

#WEAREBUNGIE

SCAN TO
LEARN MORE

BUNGiE

Appendix: Cut Challenges

The following slides are some challenges that were cut from the presentation for time.

BUNGiE

Challenges: Perforce Ditto Mapping

- Ditto mapping: client can map a depot path to multiple client paths
  - Each ditto mapped client file can be sync'd to different revisions

```
//depot/v530/main/... //my_client/v530_main/...
//depot/audio/dev/wwise_events/... //my_client/v530_main/audio/wwise_events/...

//depot/v540/main/... //my_client/v540_main/...
&//depot/audio/dev/wwise_events/... //my_client/v540_main/audio/wwise_events/...
```

BUNGiE

Challenges: Perforce Ditto Mapping

- P4 can return inconsistent results when querying by depot path for ditto mapped paths
  - P4 fstat results can be missing fields or omit some mappings
  - P4 dirs results can omit some subdirectories
  - Queries return correct results 99% of the time
- VirtualSync wants to use depot path queries for speed
- VirtualSync currently uses client path queries to workaround this

BUNGiE

Challenges: Perforce Shell Extension

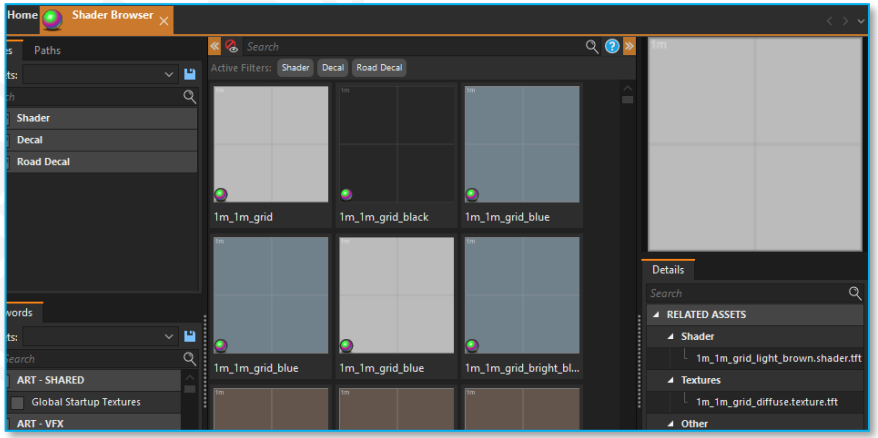
- The Perforce shell extension adds right-click menu actions
- Appears on Hydrated Placeholders, but not Placeholders
- Microsoft does this to avoid unexpected hydration
- Shell extension authors can use a flag to say they will behave
  - DFMR_OPTIN_HANDERS_ONLY

- Perforce is tracking a job to investigate supporting this

BUNGiE

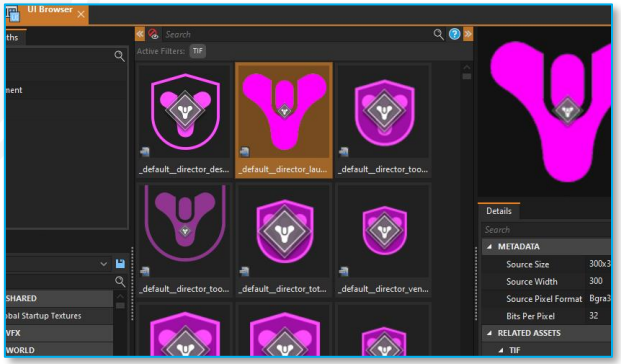
Challenges: Browser Tools

- Various browser-style tools
- Display information/thumbnail for assets of type X
- Loads all related files, build metadata and cache to local files
- Slow before VirtualSync
  - Overnight pre-caching process
  - Takes multiple nights to fully cache
- Tons of one-time hydration



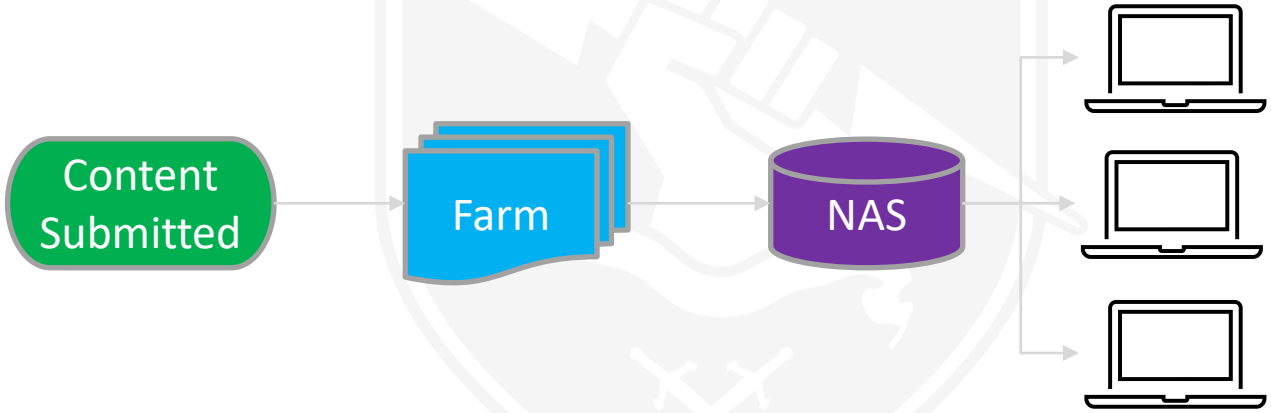
Challenges: Browser Tools – UI Art Example

- Need to scan existing UI icons/images frequently
- Source UI art stored as heavy, multi-layered .tifs
  - A 32x32 single icon may be stored in a 900 MB .tif!
- One branch
  - Contains ~32,500 UI .tifs totaling 537 GB
  - Flattened 256x256 thumbnails only 530 MB



Challenges: Browser Tools - Solution

- New service to build and supply file/asset metadata



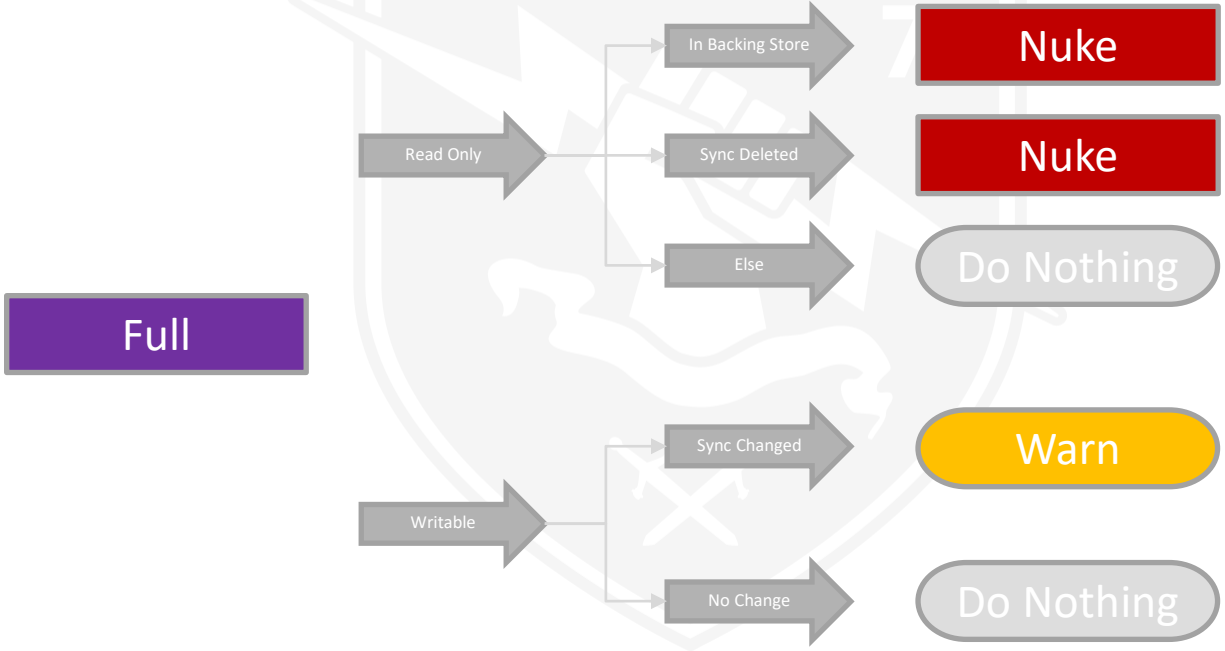
- Metadata ready and available from the moment you sync a branch
  - No local hydration required

Appendix: Post-Sync Update

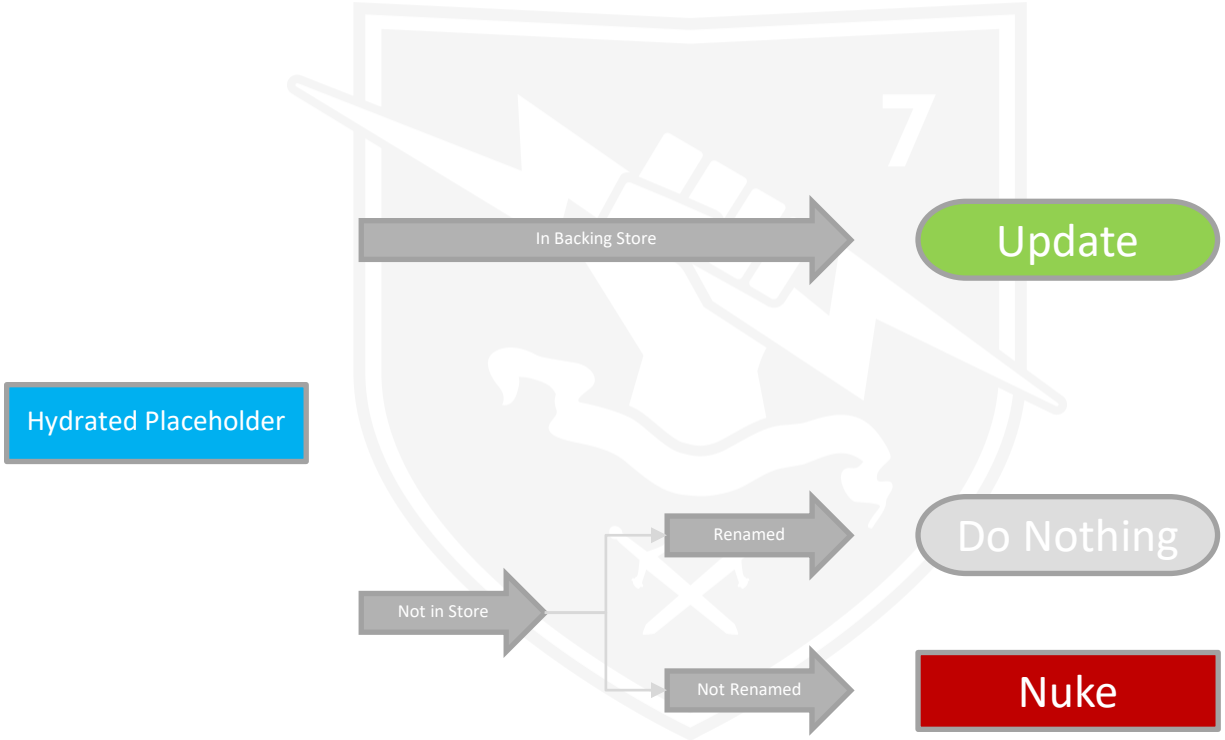
The following slides give more detail about exactly the checks and transformations we apply during our post-sync reconciliation process

Appendix: Post-Sync Update - Directories

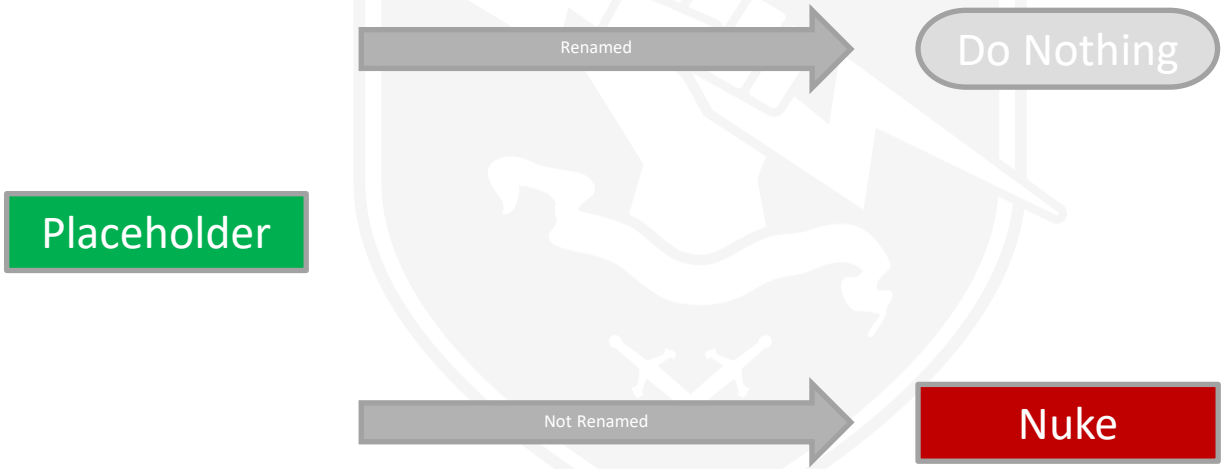
Appendix: Post-Sync Update - Files



Appendix: Post-Sync Update - Files



Appendix: Post-Sync Update - Files



Appendix: Post-Sync Update - Files