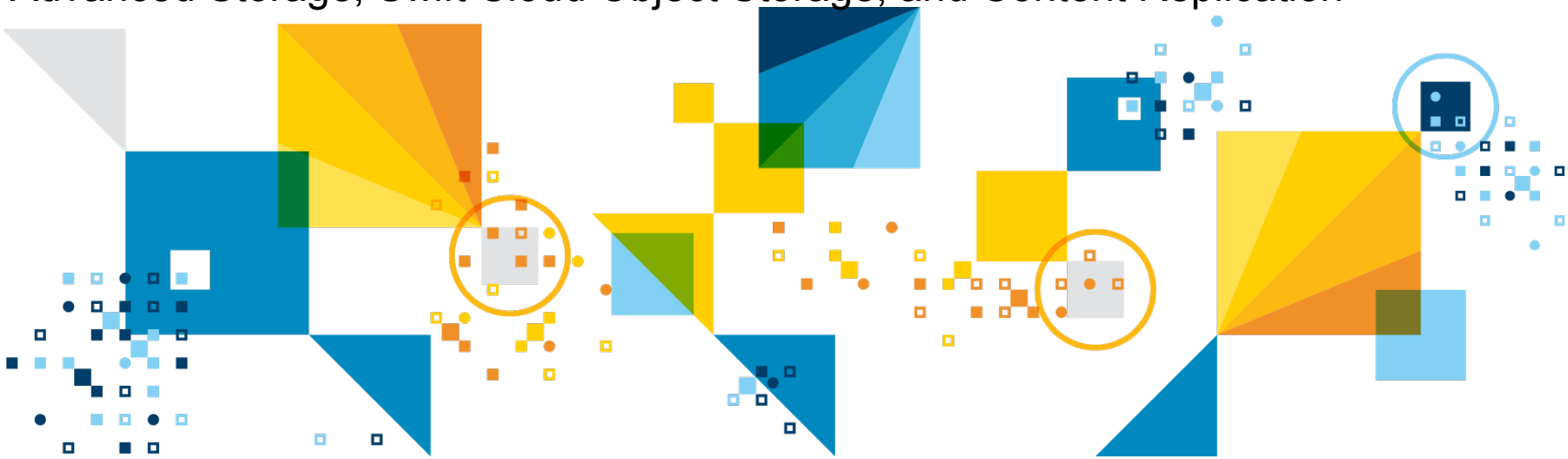
Bob Kreuch
IBM FileNet Content Manager Storage Team
October 22, 2014

Product Implementation Training (PIT)

IBM FileNet Content Manager 5.2.1GA

Advanced Storage, Swift Cloud Object Storage, and Content Replication



Introduction

- Course Overview
  - 5.2.1 Advanced Storage
  - Swift Cloud Object Storage
  - Content Replication
- Target Audience
  - Support Teams and Lab Services
- Suggested Prerequisites
  - Knowledge of P8 Environment
  - Knowledge of P8 Content Storage Areas
- Version Release Date - October 31, 2014

Course Roadmap

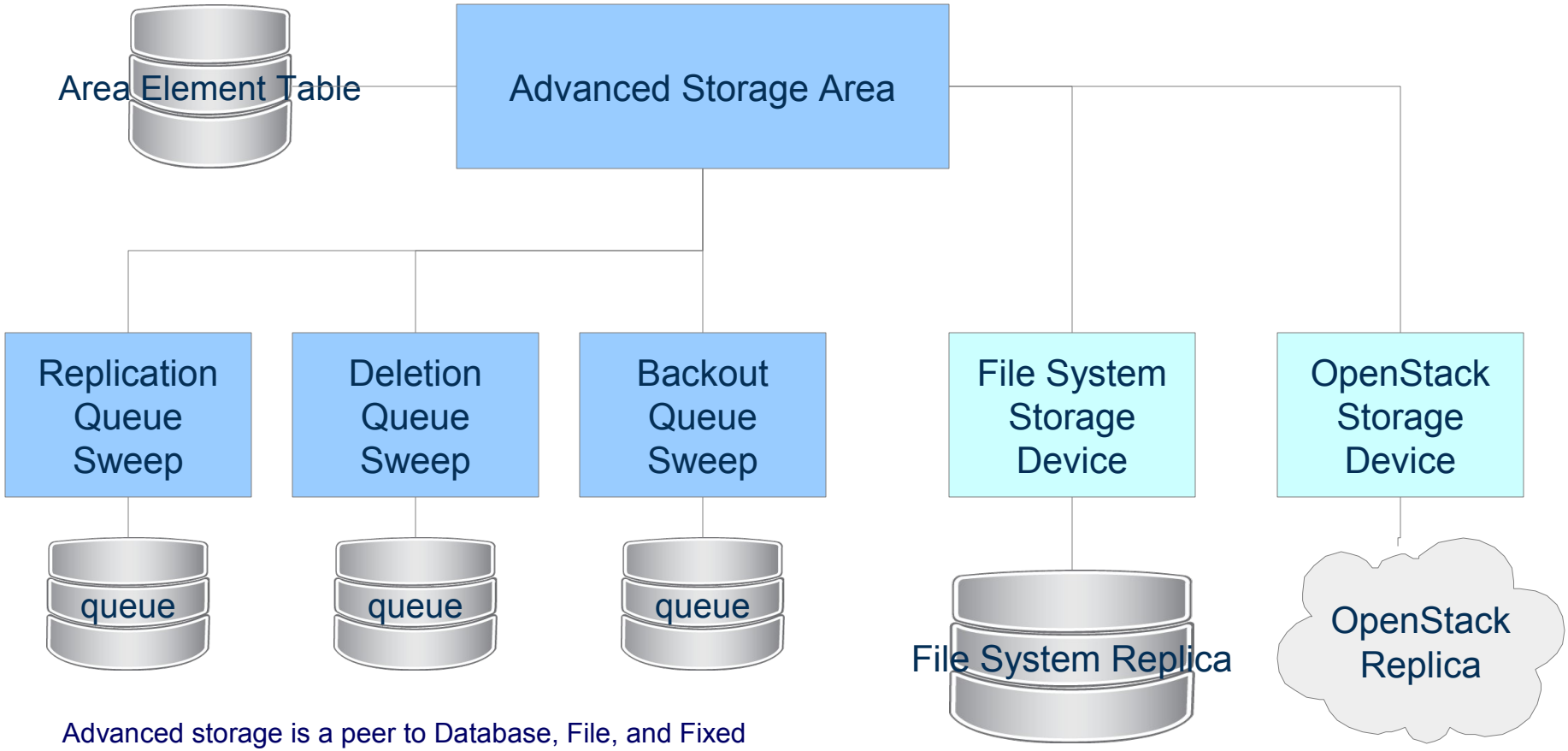
- ➔ Architecture overview
- Metadata overview
- Advanced storage devices
  - File system storage device
  - OpenStack storage device
- Demonstration

Advanced Storage Architecture Overview

Architecture overview

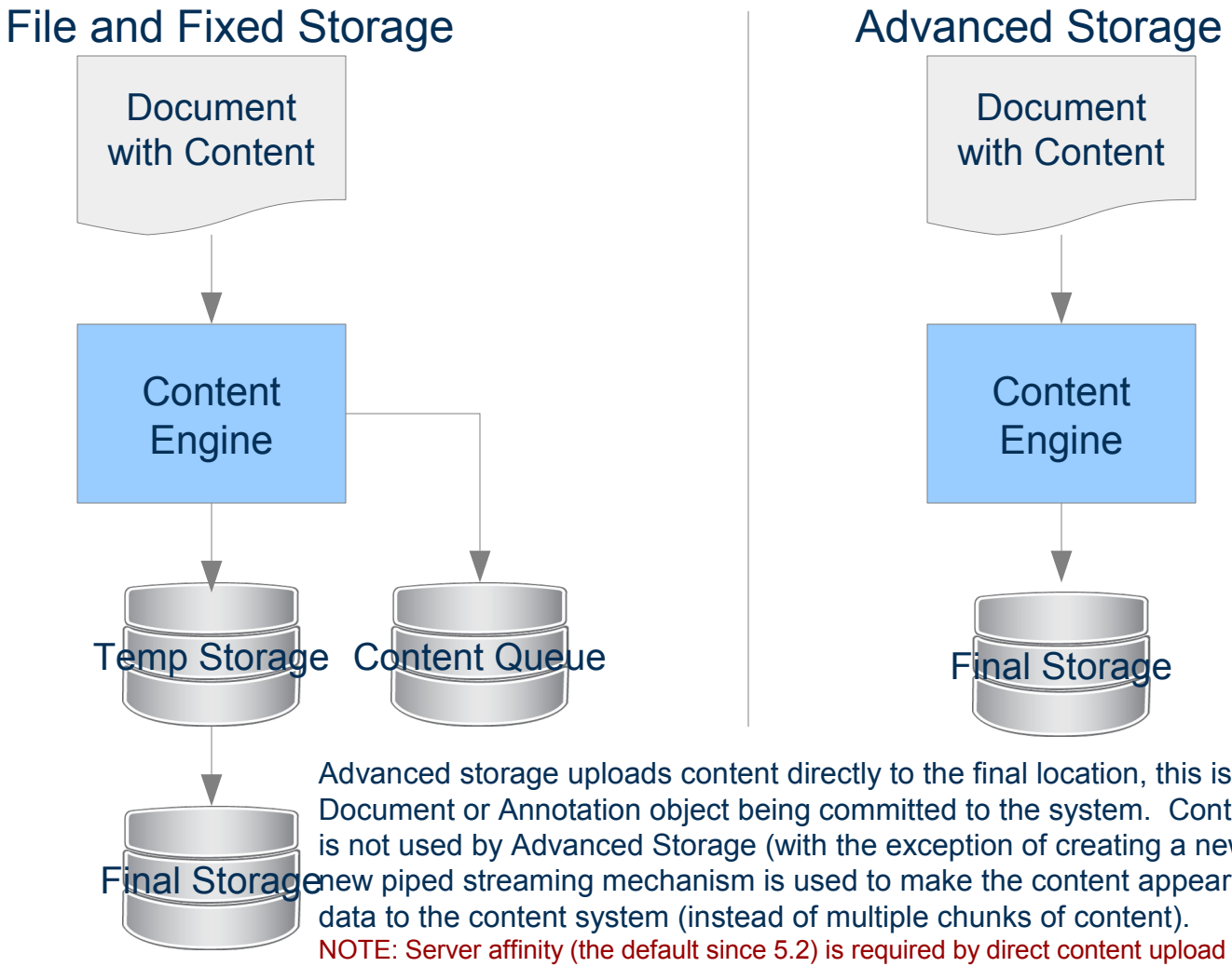
- Direct content upload
- Content replication
- Storage devices
  - OpenStack cloud object storage (a.k.a. Swift)
  - File system content storage
- Sweep framework based queue processing

Architecture overview



Advanced storage is a peer to Database, File, and Fixed Storage. Advanced storage is a new type of area, there is no concept of *backward compatibility* between advanced storage and any existing storage area (you can't transform an existing area into an advanced area without moving the content)..

Architecture overview - direct content upload



Advanced storage uploads content directly to the final location, this is done prior to the Document or Annotation object being committed to the system. Content Queue processing is not used by Advanced Storage (with the exception of creating a new Storage Device). A new piped streaming mechanism is used to make the content appear as a single stream of data to the content system (instead of multiple chunks of content).

NOTE: Server affinity (the default since 5.2) is required by direct content upload (required by Advanced Storage), with server affinity, all content upload chunks are sent to the same CPE server.

Content upload time-out

There are two types of time-outs that can occur when uploading content:

1. Time-out while reading or writing individual blocks of content to/from the content streams
2. Time-out of the overall content upload

The first type is controlled by the Domain wide Content Subsystem property:

ContentUploadTimeout (default = 60 seconds)

- If reading or writing a block of content exceeds the time-out, upload to the replica will fail
- The content upload may still be successful, depending on how replication has been defined, much more on that topic later in the slides
- *Note that internally the Content Engine makes slight adjustments to the ContentUploadTimeout (mentioned here since the adjustments may be seen in trace logs)*

The second type of time-out is intended to be fully automatic

- The intent is to allow upload of very large content, without the upload timing out
  - As long as individual blocks of content are successfully read from the source and written to the destination, the upload continues indefinitely

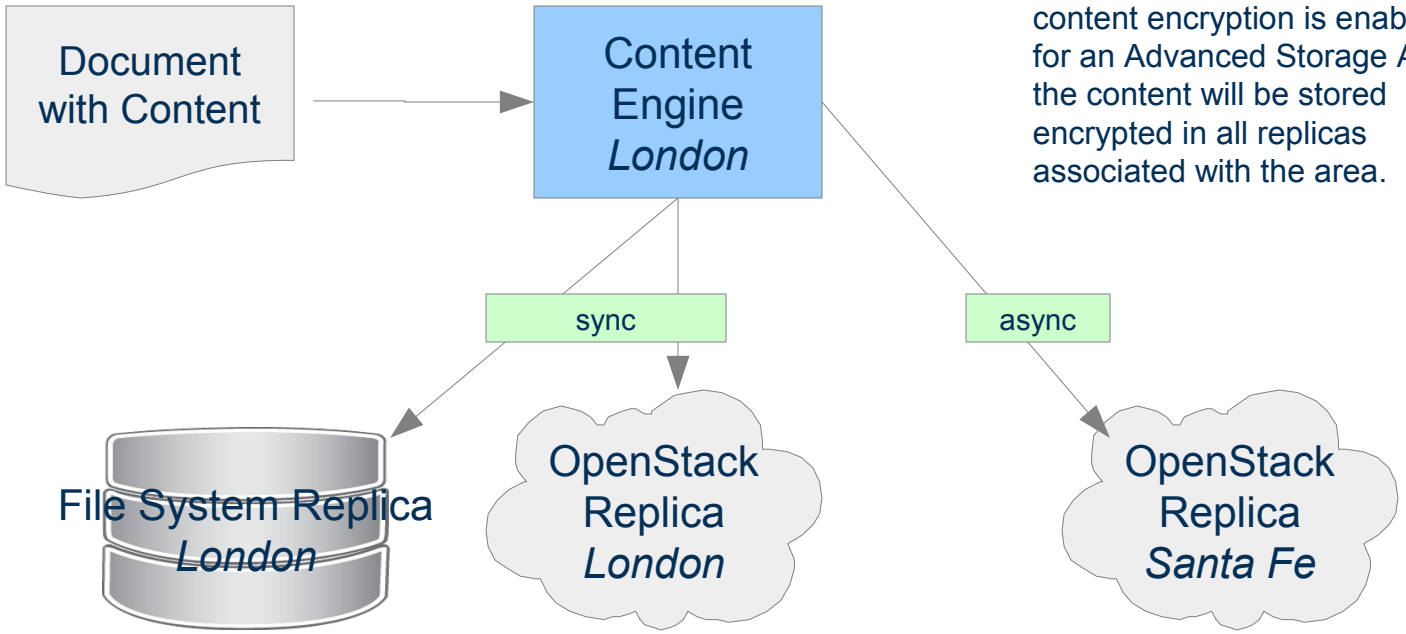
Architecture overview - content replication

This section discusses Content Platform Engine based content replication:

- Synchronous versus asynchronous replication
- Synchronous replicas for upload, required versus desired
- Replicas and site sensitivity
- Replica synchronization type (primary, secondary, asynchronous)
- Prioritizing replicas for upload and retrieval
- Replica synchronization type site overrides
- Replica connectivity and unreachable replicas
- Replica repair

Synchronous and asynchronous replication, common use case

Note that the content stored in each replica of an Advanced Storage Area is [eventually] identical, for example, if content encryption is enabled for an Advanced Storage Area, the content will be stored encrypted in all replicas associated with the area.



In this example, the Content Engine in London writes content synchronously to a file system replica and an OpenStack replica in London during upload and writes the content asynchronously to the OpenStack replica in Santa Fe. The following slides describe how an Advanced Storage Area is configured to support this and other replication scenarios.

Synchronous and asynchronous replication

Advanced Storage supports two types of content replication, synchronous and asynchronous:

- –Synchronous replication refers to writing content to a replica prior to the related Document or Annotation object being saved to the database
- –Asynchronous replication refers to writing content to a replica in the background (by the Replication Queue Sweep) after the related Document or Annotation object has been saved to the database

The set of replicas associated with an Advanced Storage Area are written either synchronously or asynchronously depending on configuration parameters and the state of the replicas:

- –The number of synchronous replicas required and desired
- –The site of the replica and the site of the Content Engine
- –If the replica is currently reachable by the Content Engine

Replicas required versus replicas desired

An Advanced Storage Area contains two properties that control how many replicas will be written synchronously during content upload:

- Replicas Required (CmSynchronousReplicasRequired)*
  - Replicas required must always be greater than zero (at least 1 replica required)
- Replicas Desired (CmSynchronousReplicasDesired)*
  - Replicas desired must always be greater than or equal to the replicas required

At a minimum, the number of required replicas must be written for the upload to be successful

- The related Document or Annotation object is not created (or updated) if the number of required replicas is not successfully written during upload (the save operation fails in that case)

If the number of desired replicas is greater than the number of required replicas, an attempt will be made to write the desired number – but if less than the desired number of replicas can be written, the upload is still considered successful as long as the required number is written

*In ACCE, replicas required is referred to as **Required synchronous devices**, and replicas desired is referred to as **Maximum synchronous devices**

Site of the Replica

- A replica is always in a site
- A Content Engine instance is always in a site
- If the Content Engine and the replica are in the same site, then the replica is local, if the Content Engine and the replica are in different sites, then the replica is *remote*
- Local replicas are [for the most part] preferred over remote replicas when determining which replicas should be used for upload or retrieval

Primary, Secondary, and Asynchronous Replicas

- A replica is defined as primary, secondary, or asynchronous via the storage device connection object, which ties the advanced storage area to the storage device (CmStorageDeviceConnection object, ReplicaSynchronizationType property)
- Both primary and secondary replicas are candidates for synchronous content upload
- Asynchronous replicas are not candidates for synchronous content upload, they can only be written asynchronously
- In general, primary replicas are higher priority than secondary replicas – but the site of the replica is also used when prioritizing replicas for upload or retrieval

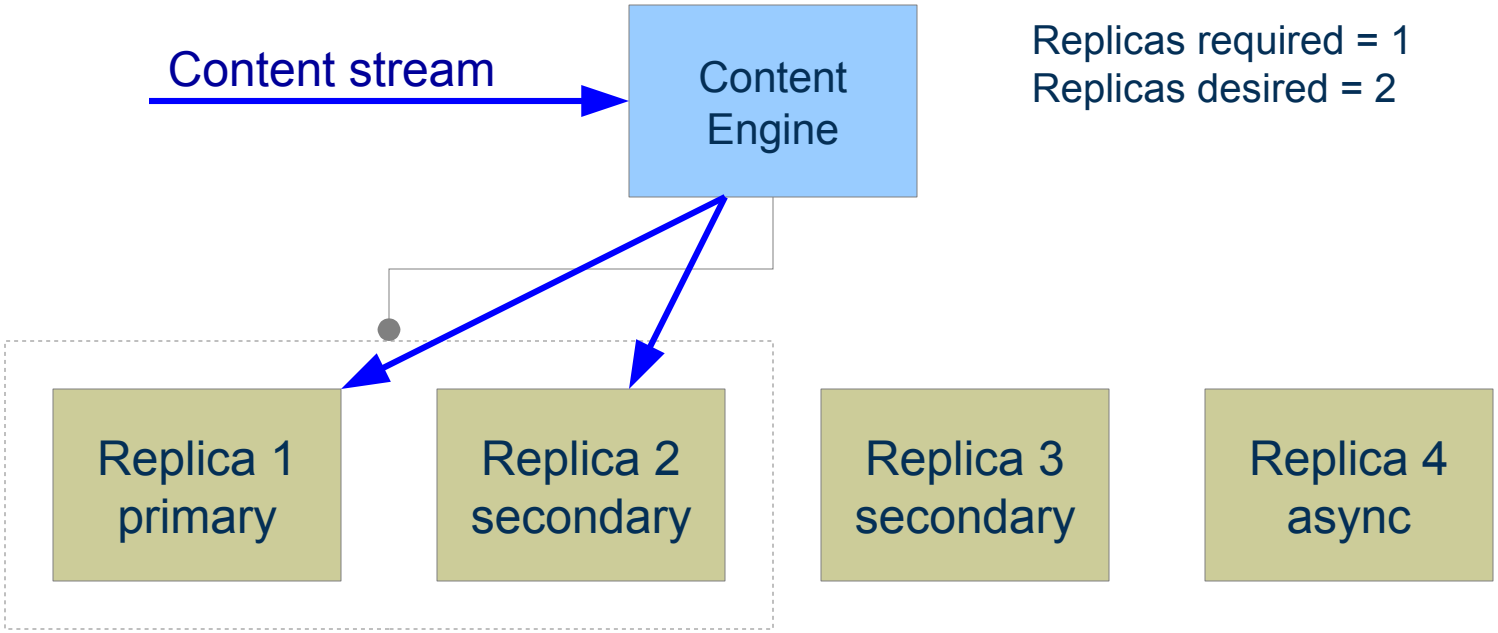
Replica prioritization – content upload

1. One Primary or Secondary replica in the local site (primary is preferred)
2. Primary local site replicas
3. Primary remote site replicas
    i. Indirect Access (Content Engine → Content Engine → Replica)*
    ii. Direct Access (Content Engine → Replica)*
4. Secondary local site replicas
5. Secondary remote site replicas
    i. Indirect Access (Content Engine → Content Engine → Replica)*
    ii. Direct Access (Content Engine → Replica)*

The Content Engine selects the set of candidate replicas based the number of desired/required replicas and the priority order above. The set must contain at least the number of required replicas, and may contain up to the number of desired replicas. Once the set has been selected, the Content Engine begins the upload process, with a separate thread writing to each replica. At the end of the upload, the Content Engine determines how many of the threads were successful, and will fail the upload operation if less that the required number of replicas were written.

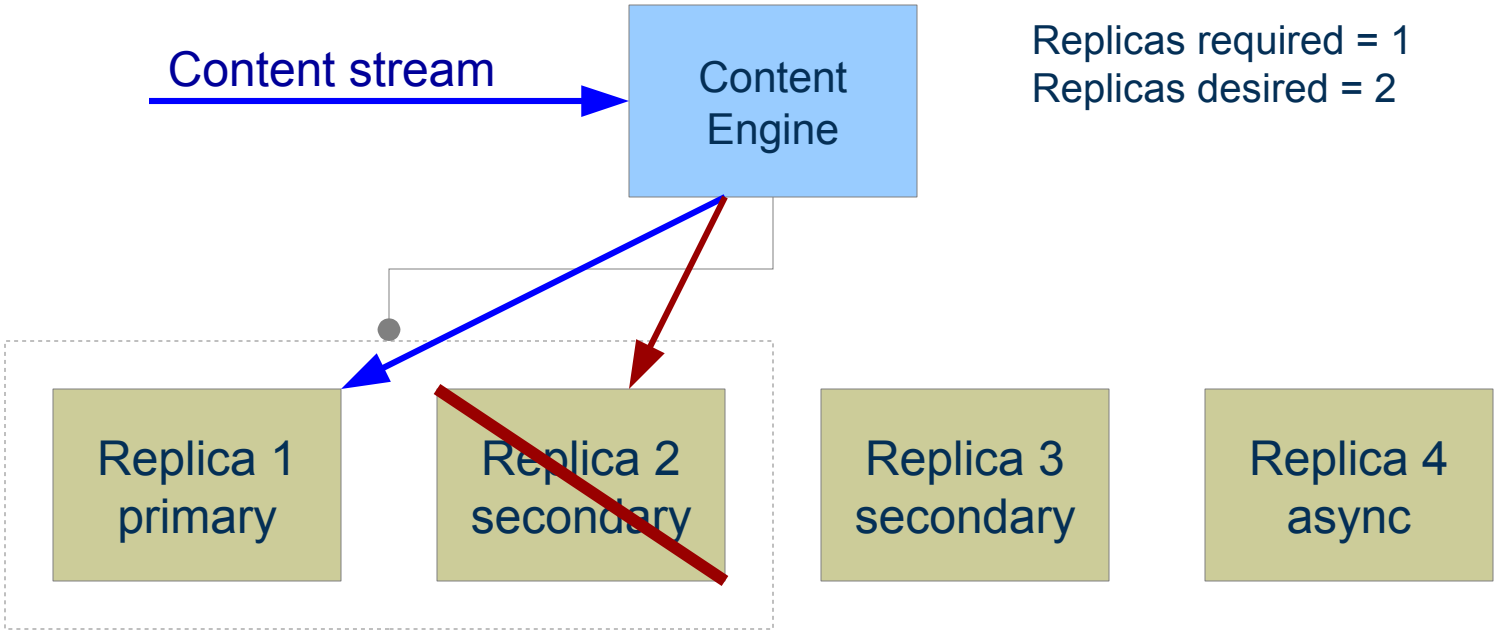
*The local Content Engine instance will use indirect access if the remote Content Engine instance is configured for *CPE Server Communication (see PxT)*. If the remote instance is not configured for CE to CE content transfer, then the local Content Engine instance will use direct access.

Replica prioritization – content upload, example of an *upload set (1/2)*



The content engine selects replica 1 and replica 2 for upload (to meet the number of replicas desired). The content flows through the content engine to both replicas during upload (each replica is written concurrently by a separate worker thread). The full content is not held in memory.

Replica prioritization – content upload, example of an *upload set (2/2)*



If a replica fails during the upload, the content engine will continue to write the content to the remaining replicas in the set that was originally selected (it does not attempt to write to any replica that was not originally selected). Once the upload has completed, the content engine will determine if the required number of replicas (1 in this case) has been successfully written, and if it has the upload is successful and the remaining replicas (including the replica that failed during upload) will be written by the replication queue sweep.

If the required number of replicas have not been written, the upload fails, and an exception is reported back to the end user.

Replica Prioritization – Content Retrieval

1. Primary local site replicas
2. Secondary local site replicas
3. Asynchronous local site replicas
4. Primary remote site replicas
5. Secondary remote site replicas
6. Asynchronous remote site replicas

The Content Engine will attempt to retrieve the content from each replica in priority order until it has successfully *opened* the content (see next slide). If attempting to open the content on a replica fails, the Content Engine will try the next replica with the highest priority (see CmContentIntegrityOptions, validate on retrieval and verify signature on retrieval). Once the content has been opened, a failure reading the content will result in a failure of the retrieval operation.

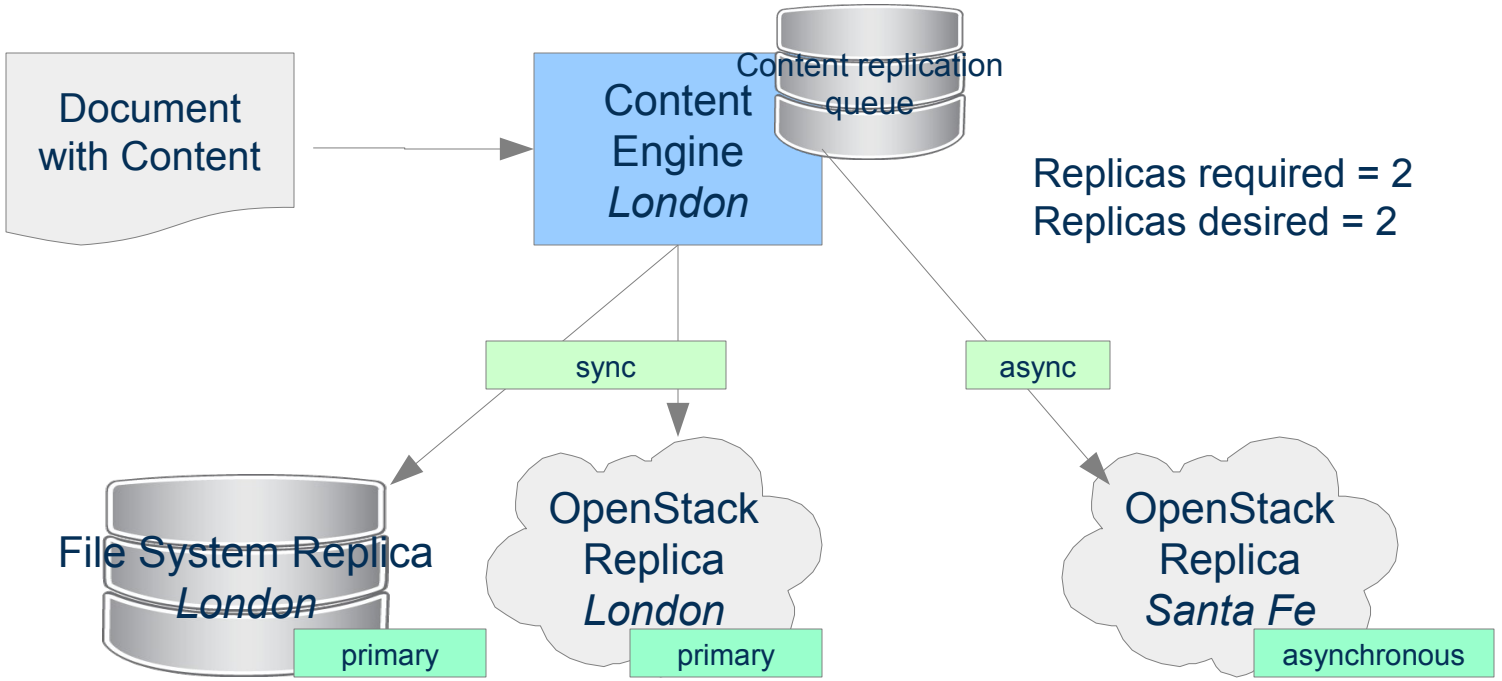
There's an upper limit on how long the Content Engine will try to open a single content item using the set of replicas. The limit is defined by the Advanced Storage Area property CmOpenTimeLimit (default is 30 seconds, see discussion of CmOpenTimeLimit later in the slides).

Note that within each of the six groupings above, the Content Engine will randomize the priority of each replica - for example, if there are two primary local site replicas, the Content Engine will randomly prioritize one replica over the other on each retrieval (to distribute the retrieval load across replicas).

Content Retrieval, Replica Failover

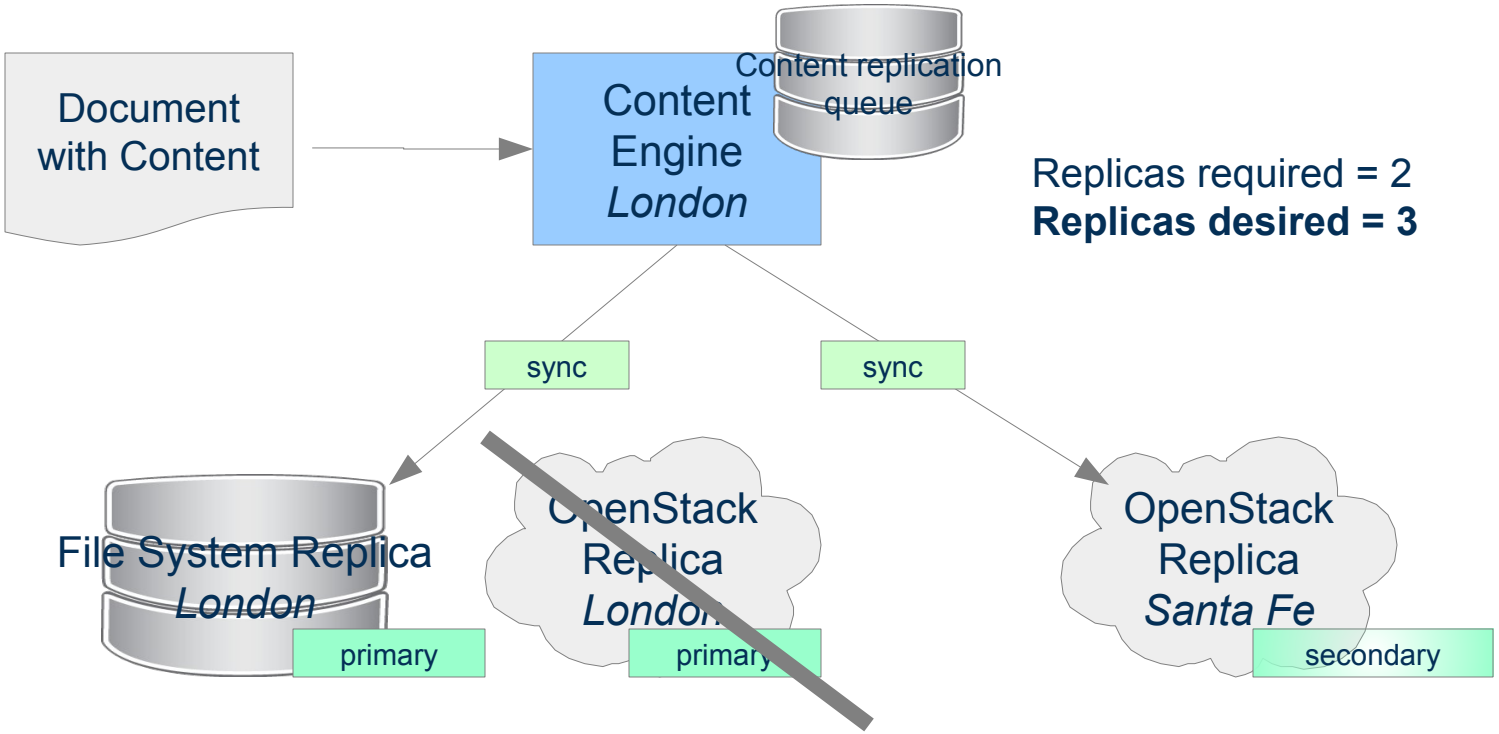
- The implementation of the content open operation described on the previous slide is left up to the Storage Handler, the handler may test the existence of the content, or it may perform an optimized open and not examine the content
  - The file system storage handler checks the existence of content on open
  - The OpenStack storage handler does not check the existence of content on open
  - *(note that the descriptions above are implementation specific and subject to change)*
- If the handlers open operation does not test the existence (or accessibility) of the content, and the content cannot be retrieved, then content retrieval won't failover to another replica (since the open operation will succeed, but retrieving the content will fail)
- If failover of content retrieval is a requirement, then the VALIDATE_ON_RETRIEVAL option of the CmContentIntegrityOptions property of the Advanced Storage Area must be enabled
  - The VALIDATE_ON_RETRIEVAL option will cause the Content Platform Engine to verify the existence and size of the content during the open operation, and will enable failover to another replica if the content doesn't exist, cannot be accessed, or is the wrong size

Common use case revisited



In the revised example, the replicas in London are marked primary and the replica in Santa Fe is marked asynchronous. The number of required and desired replicas is set to 2. The Content Engine in London will write to the two replicas in London during upload (and fail if it cannot write both), and then queue a request to write the content asynchronously to the Santa Fe replica, via the replication queue sweep.

Common use case, an alternative approach



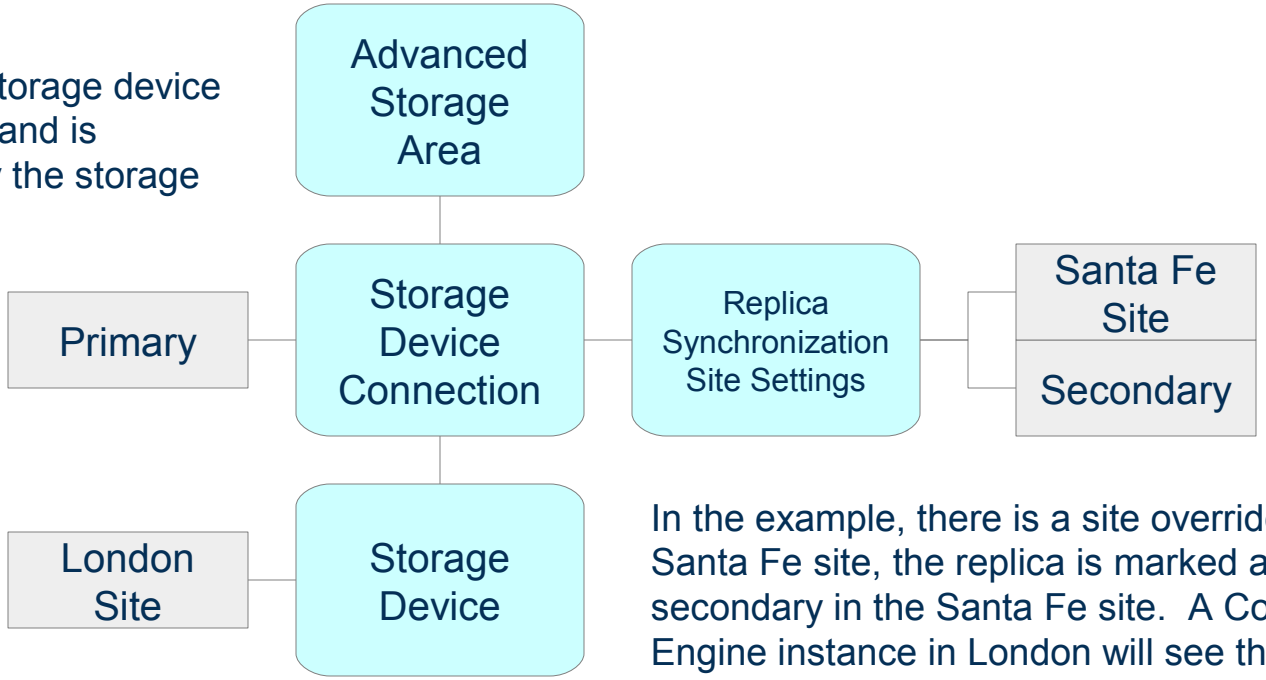
Advanced storage replication is very flexible and can support a wide variety of replication models. In this alternative example, the replicas desired is set to 3 (required is still set to 2), and the Santa Fe replica has been marked as secondary (not asynchronous). Normally all three replicas will be written during content upload, but if any one of the three should fail, the system can continue to operate with the remaining two replicas (replication requests will be queued up for the failed replica, and will be processed when the replica is back online).

Site overrides - if you thought it couldn't be any more complicated...

The primary, secondary, or asynchronous designation of a replica can be overridden for a site

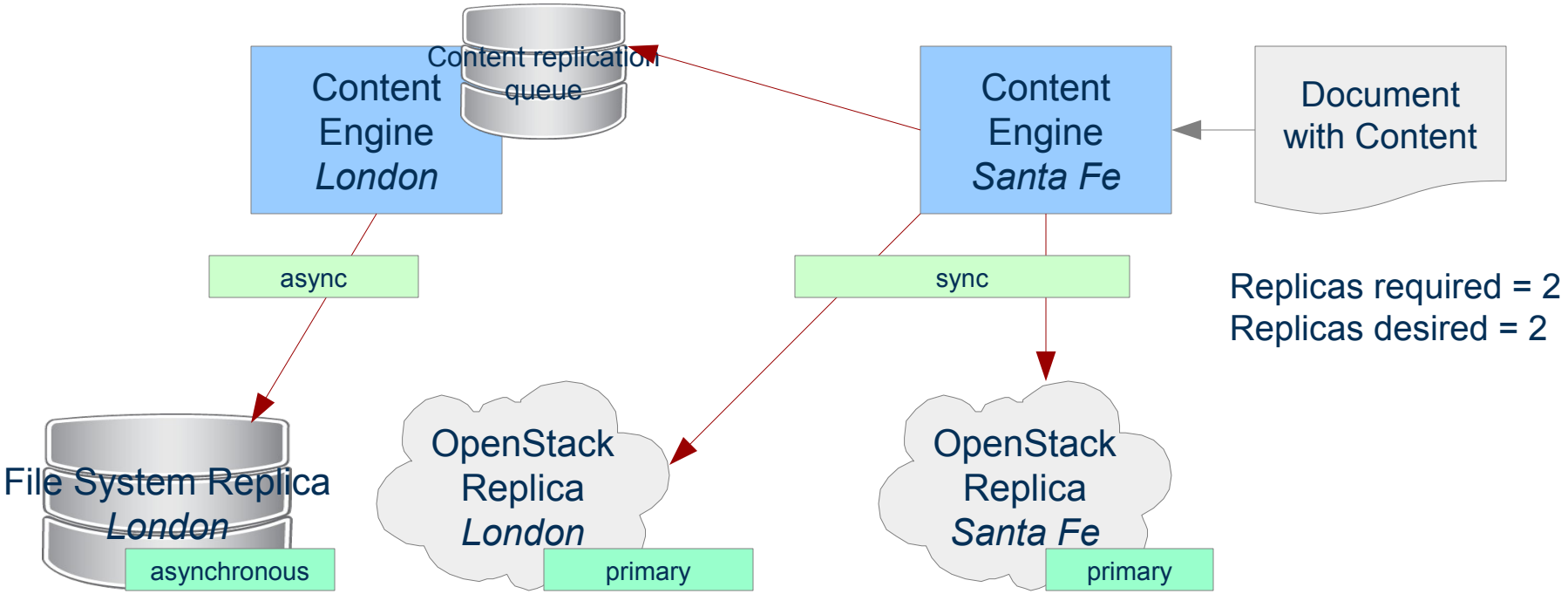
An override is defined by a replica synchronization site settings object (CmReplicaSynchronizationSiteSettings), which can be attached to a storage device connection object, a logical view is seen below:

In this example, the storage device is in the London site, and is marked as primary by the storage device connection.



In the example, there is a site override for the Santa Fe site, the replica is marked as secondary in the Santa Fe site. A Content Engine instance in London will see the replica as primary, a Content Engine instance in Santa Fe will see the replica as secondary.

Common use case with site overrides



In the revised example, site overrides are used in the Santa Fe site to cause content uploaded in the Santa Fe site to be written to a site local replica and to one of the remote replicas. The other remote replica is defined to be asynchronous to the Santa Fe site, it will be written from the replication queue. Note that the queue sweeps always run in the site of the Object Store (for example, if the Object Store is in the London site, the replication queue sweep will run on the London Content Engine server).

Storage Device Connectivity and Unreachable Replicas

Each Content Engine instance maintains the *reachable* status of each replica:

- –Is the replica reachable via direct access (yes/no)?
- –Is the replica reachable via *CPE Server Communication* content transfer (yes/no)?

The status is determined by testing access to the replica:

- –The Connectivity Task periodically tests access to replicas, by default there is a minimum 5 minute interval between tests of a replica
- –If there is an unexpected failure accessing the replica (i.e. a content retrieval operation fails), the Connectivity Task is notified to test the replica as soon as possible

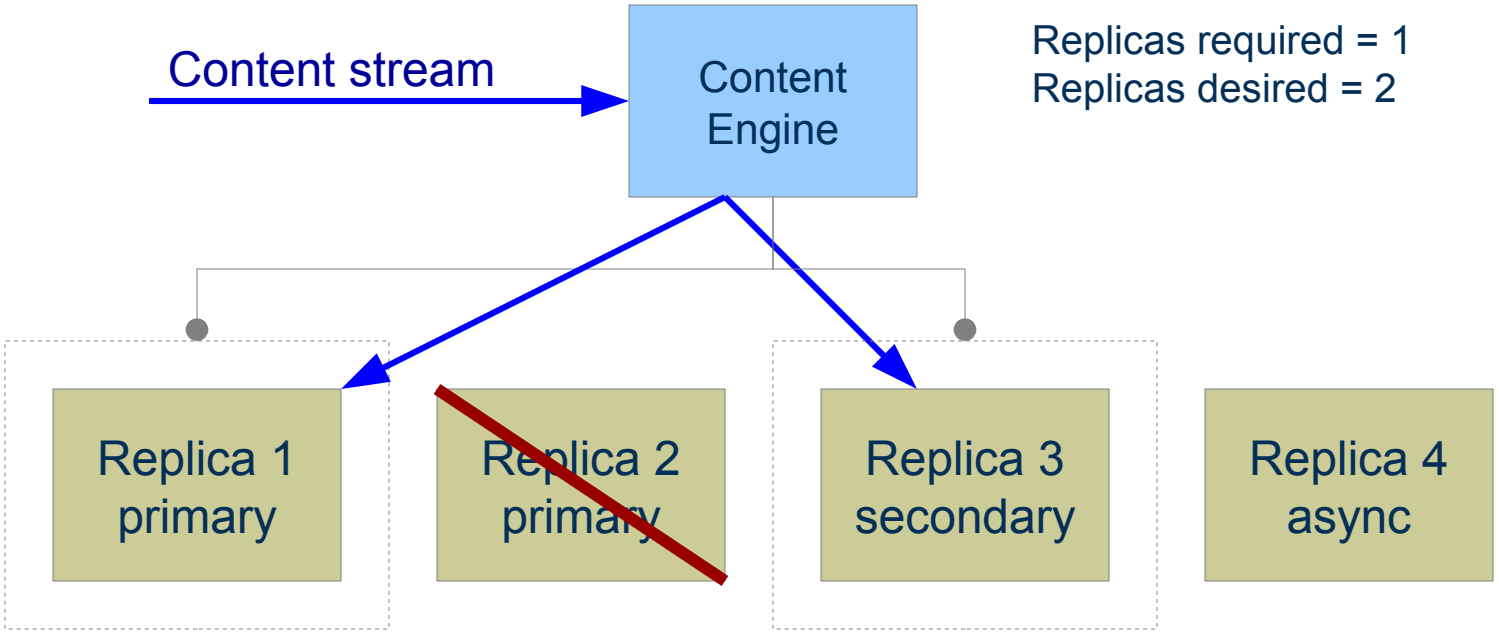
Once a replica is marked unreachable (both through indirect and direct access), that replica will be avoided when uploading or retrieving content:

- –Content upload : the replica will not be included in the set of replicas that are candidates for synchronous upload (the replica will be treated as a asynchronous replica)
- –Content retrieval : the replica will be skipped when attempting to open the content, the exception is when all replicas are marked unreachable, in that case the Content Engine will ignore the unreachable state and attempt to retrieve the content as if all replicas were marked reachable

The replica test interval used by the Connectivity Task can be overridden via the JVM parameter below:

Advanced.ConnectivityRefreshSeconds=n

Unreachable Replicas and the *upload set*



In this example, replica 2 has been marked unreachable, the content engine avoids using replica 2 for upload and retrieval (the set of replicas for upload includes a secondary, which is lower priority than replica 2, but replica 2 is skipped since it is unreachable).

Automatic replica repair using the consistency checker

Use Case:

- –Replica is damaged and the *media* is restored to a previous point in time*
  - •There is data loss (for example, content added in the previous two days is gone)
- –The CPE consistency checker tool is used to instruct the Content Engine to re-create the missing content in the damaged replica:
  - •The CmContentIntegrityOptions property of the advanced area(s)** must have the ContentIntegrityOption.AUTO_REPAIR_ON_VALIDATE_CONTENT option set
  - •A date range is used to limit the content that needs to be checked
  - •The checker will report missing content errors during the validation run
  - •The Content Engine will queue replication requests to re-replicate the content to the restored replica (assuming there is at least 1 good copy of the content)
  - •Once all the replication requests have been processed, the consistency checker can be re-run to determine if all the content was repaired

* Note that the replica only needs to be repaired if the *media* is restored to a previous point in time. If a replica just goes offline and then is brought back online after some period of time, the replica does not need to be restored – while the replica was offline, requests will have been written to the replication queue (for the offline replica), those requests will be processed once the replica has been brought back online.

** All advanced areas that use the replica must be processed by the consistency checker (if the replica is shared by multiple advanced areas).

Architecture overview - storage device

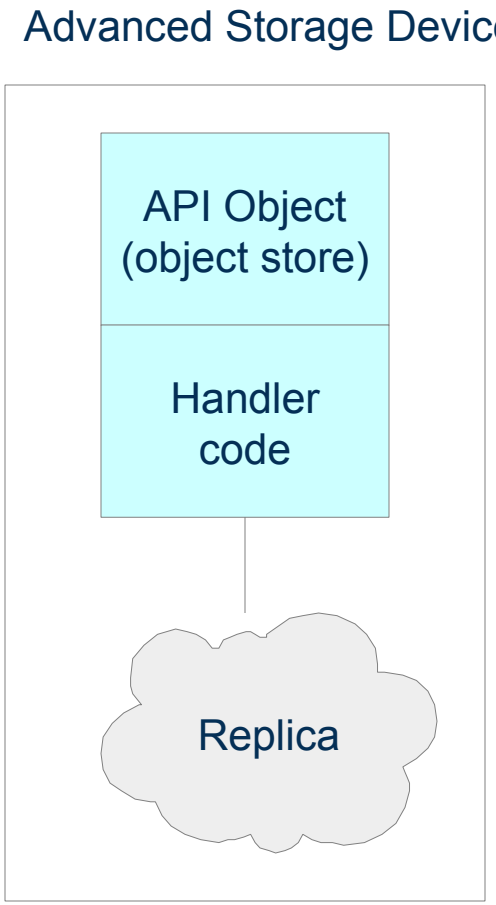
An advanced storage device consists of three components:

1. API object that defines the properties of the device, for example the path to a directory or the URL and credentials of an OpenStack object store

2. Handler code that acts as the interface to the device, the handler code implements an interface defined by the Content Engine (StorageHandlerInterface)

3. The storage media (i.e. the replica), for example, a volume on a disk drive or an OpenStack object store

OOTB 5.2.1 GA will support a file system based storage device and a OpenStack (Swift) based storage device.



Architecture overview - OpenStack cloud object storage, a.k.a. Swift

Cloud object storage is supported via the OpenStack content object storage model

- The OpenStack REST interface is used to communicate with OpenStack cloud storage*

In theory the OOTB OpenStack storage device should be able to support any cloud store that implements OpenStack storage using version 1 or version 2 authentication**

- The OpenStack device has only been tested with SoftLayer

In theory, cloud object storage is an ideal storage scheme for IBM customers***:

- Content objects are outsourced to a 3rd party repository
- Content objects are stored under a highly redundant scheme
- Content objects are accessible from anywhere in the World
- The cloud storage model should reduce the cost the maintaining a large set of content objects

*See OpenStack Object Storage API v1 Reference for more information
**Version 1 refers to OpenStack Object Storage API v1 authentication, Version 2 refers to OpenStack Identity API v2 authentication
*** The caveat is that cloud object storage implies high latency, which may make it difficult to meet performance requirement for content ingestion and/or retrieval in some environments

Architecture overview – file system storage

- File system storage is supported via the built in file system storage device
- The file system storage device stores content in a directory structure, much like the existing file storage area
- Advanced storage uses direct content upload, and not the content queuing mechanism, which in some environments may reduce database and file system overhead*, and eliminate performance issues related to complex content queue queries
- Customers using file storage areas should be encouraged to consider using advanced storage with a file system device for new storage areas

* Note that internal performance testing has not shown that an advanced storage file system device performs better than a legacy file storage area, but in some environments the reduction in either database activity and file system rename operations may favor advanced storage.

Architecture overview - queue sweeps

Content replication

- –As content is ingested, requests are written to the replication queue to write content to asynchronous replicas*. The queue sweep processes the replication queue, retrieves the content from a synchronous replica, and writes the content to one or more asynchronous replicas.

Content deletion

- –When a Document or Annotation is deleted, or when specific content elements of a Reservation or an Annotation are deleted, the deletion of the related content is queued in the deletion queue, and then processed by the deletion queue sweep. The deletion queue sweep processes the queue, and deletes the content from all replicas. *See next topic on content deletion delay.*

Abandoned content backout

- –Content that is abandoned during content upload is eventually deleted from replicas by the abandoned content cleanup sweep.

* A replica can be treated as asynchronous for various reasons, including being configured as asynchronous for the advanced storage area (for the site), not being written synchronously due to the replicas required/desired settings, or being offline when content is ingested.

Advanced storage queue sweeps, maximum sweep workers

- By default, the Maximum Sweep Workers for the Content Replication and Deletion queue sweeps are set to fairly low values:
  - Replication Queue Sweep = 4 workers
  - Deletion Queue Sweep = 2 workers
- The maximum number of workers for the replication and deletion queue sweeps may need to be increased if queue processing is falling behind document ingestion and/or deletion
- The number of workers should be increased gradually to determine the impact of more background worker threads on overall system response times
- IBM System Dashboard counters can be used to monitor the number of active replication and deletion queue sweep worker threads

Object Store → Sweep Framework → Sweep Queue : Content Replication Sweep → Sweep Active Workers

Object Store → Sweep Framework → Sweep Queue : Content Deletion Sweep → Sweep Active Workers

- Note that the default maximum sweep workers for the abandoned content deletion queue sweep is set to 1, and normally shouldn't need to be increased (in most cases backout is considered to be a lower priority operation)

Content Deletion Delay Object Store Parameter

A new Object Store property, AdvancedStorageDeletionDelay*, can be used to delay deletion of content stored in an advanced storage area

The held until date property of deletion requests (inserted into the deletion queue) is always set to the current time plus the AdvancedStorageDeletionDelay property value

- –The deletion queue item will not be processed until the held until date [time] is reached

The deletion delay is intended to support a window where the database can be rolled back to a previous point in time without needing to restore content that was deleted after the roll back time

- –For example, if the deletion delay is 24 hours, the database can be rolled back to any time within the 24 hour window without the content having been physically deleted
- –Note that this scheme may require encrypting the content, since content that was created within the rollback window will not be referenced in the recovered database (the content will be orphaned on the replica, and there is no mechanism to find and delete it)

* The default value for AdvancedStorageDeletionDelay is 600 seconds.

Course Roadmap

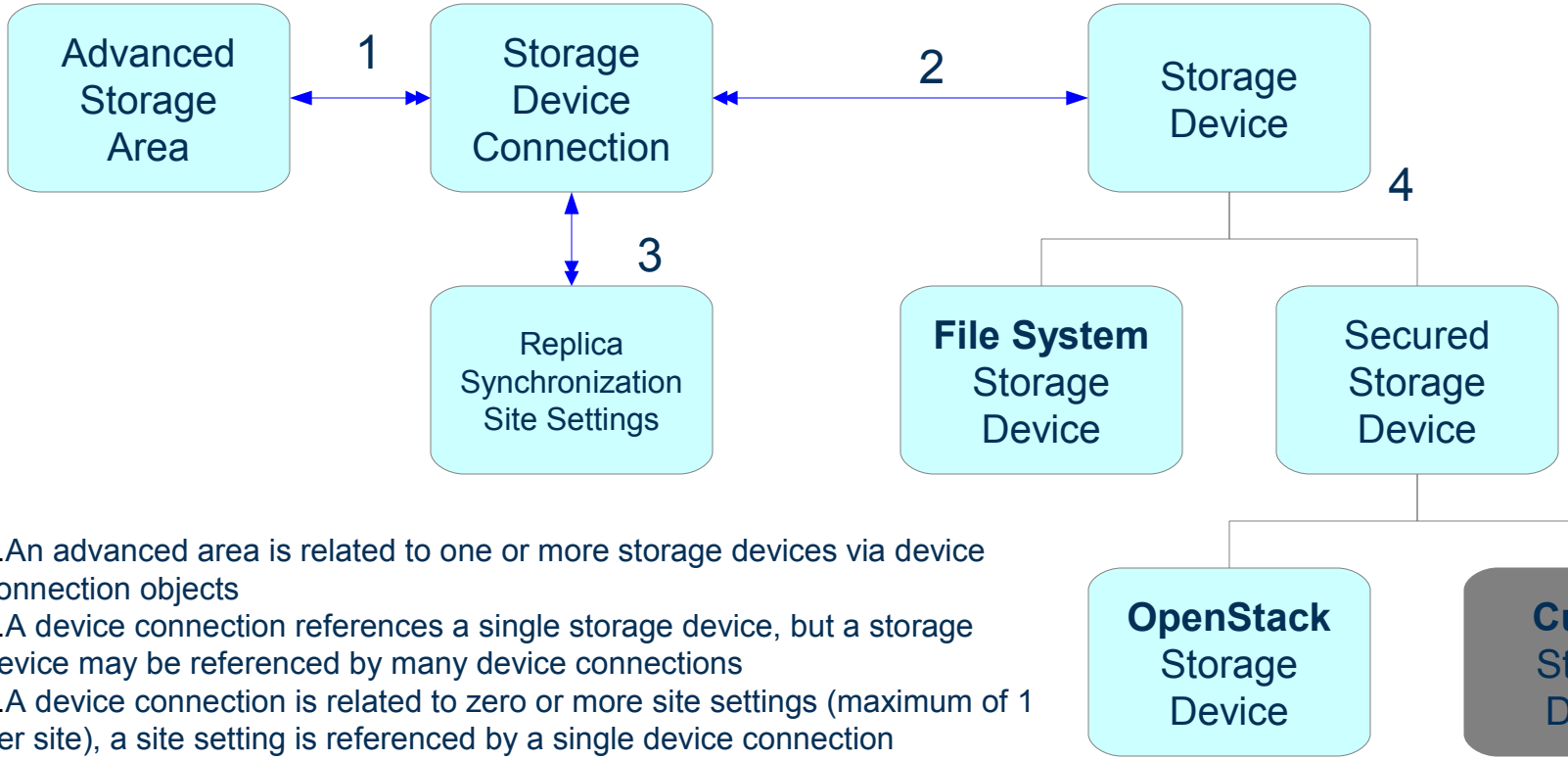
- Architecture overview
- ➔ Metadata overview
- Advanced storage devices
  - File system storage device
  - OpenStack storage device
- Demonstration

Metadata overview

This section discusses the metadata associated with advanced storage

- Advanced Storage Area (CmAdvancedStorageArea)
- Storage Device (CmStorageDevice)
  - File System Storage Device (CmFileSystemStorageDevice)
  - Secured Storage Device (CmSecuredStorageDevice)
  - >> Open Stack Storage Device (CmOpenStackStorageDevice)
  - >> Custom Storage Device (CmCustomStorageDevice)
- Storage Device Connection (CmStorageDeviceConnection)
- Site Settings (CmReplicaSynchronizationSiteSettings)

Metadata overview – class hierarchy and object relationships



1.An advanced area is related to one or more storage devices via device connection objects
2.A device connection references a single storage device, but a storage device may be referenced by many device connections
3.A device connection is related to zero or more site settings (maximum of 1 per site), a site setting is referenced by a single device connection
4.A storage device is an instance of one of the device sub-classes : file system or OpenStack (custom devices are reserved for potential future use)

Advanced storage area properties

Most of the properties of the advanced storage area are inherited from the base storage area, only the properties that are specific to advanced storage areas (or introduced with advanced storage) are described here

- Content integrity options
- Content Id batch size and time to live
- Replicas required and desired (see discussion in the overview section)
- Content open time limit
- Storage device connections

CmContentIntegrityOptions

VALIDATE_ON_CREATION	Validate existence and size after content upload or replication (on the target replica or replicas).
VALIDATE_ON_RETRIEVAL	Validate existence and size at the beginning of content retrieval (prior to returning content to the end user), attempt to retrieve from another replica if the content is determined to be invalid.
VERIFY_SIGNATURE_ON_CREATION	If VALIDATE_ON_CREATION is specified and the content signature has been computed, verify the signature after content upload or replication.
VERIFY_SIGNATURE_ON_VALIDATE_CONTENT	If the content signature has been computed, verify the signature within the validateContent API.
VERIFY_SIGNATURE_ON_RETRIEVAL	If VALIDATE_ON_RETRIEVAL is specified and the content signature has been computed, verify the signature prior to returning content to the end user, attempt to retrieve from another replica if the content is determined to be invalid.
AUTO_REPAIR_ON_VALIDATE_CONTENT	If the validateContent API detects invalid content (missing, invalid size, [optional] invalid signature), initiate auto repair of the content (via the replication queue). *Note that signature validation is only applied if a) the signature has been computed, and b) the VERIFY_SIGNATURE_ON_VALIDATE_CONTENT option is specified).*
AUTO_REPAIR_ON_RETRIEVAL	If content retrieval detects invalid content (missing or invalid size) on a replica being used for the retrieval operation, initiate auto repair of the content (via the replication queue). *Note that this option does not apply signature validation, the signature will only be validated if a) the signature has been computed, and b) the VERIFY_SIGNATURE_ON_RETRIEVAL option is specified).*
COMPUTE_CONTENT_SIGNATURE	If enabled, the content signature is always computed during content upload.

Storage Area CmContentIntegrityOptions Property, continued

VALIDATE_ON_RETRIEVAL – special cases

- –Validation is not performed when content is retrieved as part of replication. If content validation is desired during replication, the VALIDATE_ON_CREATION option should be used (content will be validated after it is written to the target replica).
- –Validation is not performed when content is retrieved for CBR indexing (for performance reasons).

AUTO_REPAIR_ON_RETRIEVAL – special cases

- –Repair is only performed if the Content Engine attempts to access the content on a replica where the content is either missing or damaged (if the Content Engine retrieves the content from a *good* replica before attempting to retrieve it from a *bad* replica, the repair operation will not be invoked).
- –The exclusions listed above for VALIDATE_ON_RETRIEVAL apply to repair also.

CmContentIdBatchSize and CmContentIdBatchTTL

- To support the abandoned content backout scheme, the identifiers for content items are generated by the Content Engine in batches
- The CmContentIdBatchSize property determines the size of each batch (how many content ids are in each batch)
- The CmContentIdBatchTTL property determines the life span of a batch, once a batch reaches its end of life, content ids from the batch will no longer be used for new content
- The default values for these properties should normally be used, in some rare cases it may be necessary to adjust these values for better ingestion performance, resolve a database issue, etc. (the values should only be adjusted after an investigation by IBM support, and only adjusted as recommended by IBM support)

CmOpenTimeLimit

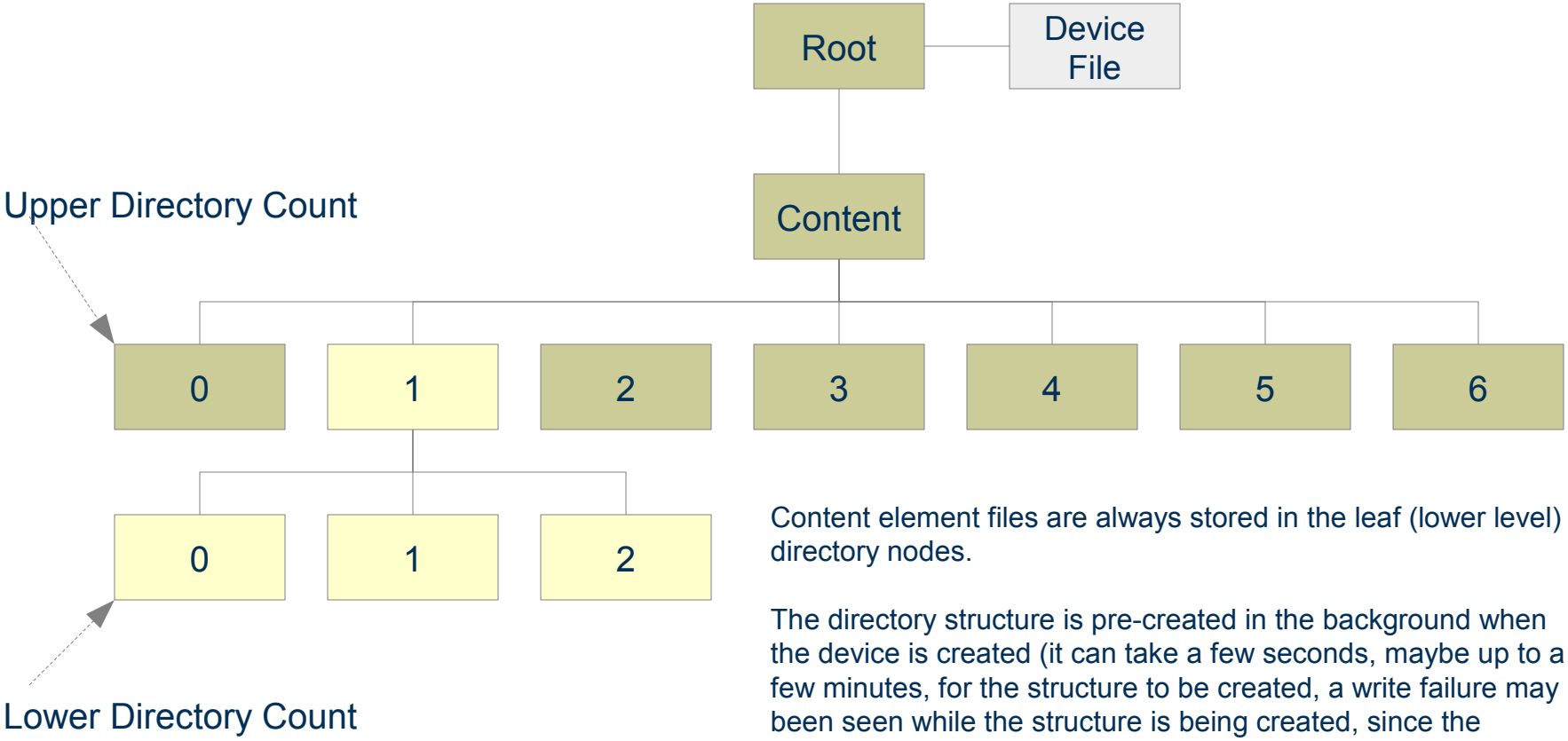
The content open time limit property is used to control how long the Content Engine will attempt to open content from the set of replicas before failing the retrieval operation, note that this limit doesn't define a hard stop time, it defines an upper limit that is checked at the end of each attempt to open content for retrieval...

- –For example, the time limit is set to 30 seconds and you have three replicas
- –There's something wrong with the network, it takes 28 seconds to attempt to retrieve from the first replica, which fails
- –Since its still under the 30 second limit, the Content Engine attempts to retrieve from the second replica
- –Consider these two scenarios...
  - •After 14 seconds, the second replica also reports a failure
  - •>> It's now over the 30 second limit, the Content Engine fails the retrieval request
  - •After 14 seconds, the second replica reports success (content opened)
  - •>> Even though it's over the 30 second limit, the Content Engine considers the open to be successful, and returns the content

Course Roadmap

- Architecture overview
- Metadata overview
- ➔ Advanced storage devices
  - File system storage device
  - OpenStack storage device
- Demonstration

File system device - directory structure



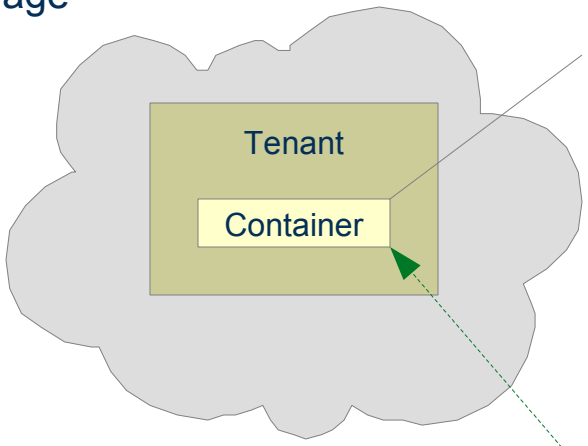
Content element files are always stored in the leaf (lower level) directory nodes.

The directory structure is pre-created in the background when the device is created (it can take a few seconds, maybe up to a few minutes, for the structure to be created, a write failure may been seen while the structure is being created, since the directory must exist when a Content Element file is created).

The Content Id is used as the Content Element filename, for example the Id below...
{A346EC8C-5D8C-452e-8239-1995AD26B0C1}
Results in the filename below...
A346EC8C-5D8C-452e-8239-1995AD26B0C1

OpenStack Storage Device

Swift Cloud Object Storage



A346EC8C-5D8C-452e-8239-1995AD26B0C1

Object name

The Content Id is used as name of the object in the Swift store (leading and trailing brackets trimmed)

{A346EC8C-5D8C-452e-8239-1995AD26B0C1}

Content Id

Content is stored in a container within a tenant. An OpenStack advanced storage device maps to a single tenant / container.

Course Roadmap

- Architecture overview
- Metadata overview
- Advanced storage devices
  - File system storage device
  - OpenStack storage device
- ➔ Demonstration

Demo

Create an advanced storage area with a file system replica and a OpenStack replica
- 1 required, 1 desired

Create a document with content using the advanced storage area
- Show content replication queue count
- Show content in replicas

Retrieve the content

Corrupt the content in the file system replica

Use the consistency checker to repair the content
- New consistency checker options for replica repair

Product Help/Documentation/Resources

P8 5.2.1 Information Center (available October 31st)

http://www.ibm.com/support/knowledgecenter/SSNW2F_5.2.1/

Planning and preparing for FileNet P8 installation

- Performing the required installation preparation tasks
  - IT administrator installation tasks
    - Advantages of advanced storage areas
    - Replication models for advanced storage areas
    - Preparing advanced storage areas

Administering Content Platform Engine

- Defining the repository infrastructure
  - Storing content
    - Advanced Storage Areas
    - Replication
    - Advanced Storage Devices
    - Advanced storage area: Repairing a replica