Welcome to Unity OE v4.1: Scalability and Performance.

Copyright ©2016 EMC Corporation. All Rights Reserved. Published in the USA. EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED "AS IS." EMC CORPORATION MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Use, copying, and distribution of any EMC software described in this publication requires an applicable software license. The trademarks, logos, and service marks (collectively "Trademarks") appearing in this publication are the property of EMC Corporation and other parties. Nothing contained in this publication should be construed as granting any license or right to use any Trademark without the prior written permission of the party that owns the Trademark.

Revision Date: December 2016

Revision Number: MR-7WN-UNOEV41SPF.4.1
The course covers the new scalability and performance features that were implemented in the Unity OE v4.1 release. Topics include: Data-in-Place upgrades, Persistent File Data Cache (PFDC), and Inline Compression (ILC).

One of the key new features to improve scalability for Unity was to add support for Data-in-Place conversions. This module will look at the Unity models and the hardware upgrade path/s for each model. Students should be able to identify the DIP conversion process and the considerations to be aware off when performing the conversion. Other topic include identifying some possible DIP scenarios and locating the proper documentation to support the product.

The PFDC feature is new for the Unity OE v4.1 release and is designed as a mechanism to delay allocating space within the LUN for data that potentially can be compressed. PFDC delays potentially expensive mapping operations until after an I/O request has been acknowledged back to the host.

For improved space savings on backing storage, an Inline Compression feature was added in Unity OE v4.1. ILC allows for data to be compressed on a given LUN (if the data is compressible).

By writing data in a compressed format, customers can better utilize the space savings on the storage system. The ILC feature relies upon the PFDC feature to work. ILC topics include an overview of the feature, where the compression engine fits into the software layer, and some examples of how a host write gets processed by the compression engine.
Module: Data-in-Place Conversions (DIP)

Upon completion of this module, you should be able to:

- Describe the DIP feature
- Identify the upgrade path for Unity storage arrays
- State the considerations for implementing a DIP conversion
- Describe the Data-in-Place conversion process

This module focuses on new scalability and performance features incorporated in the Unity OE v4.1 release. Students will be able to identify and describe the new DIP feature, identify the upgrade path for Unity storage arrays, state the considerations for implementing a DIP conversion, and describe the Data-in-Place conversion process.
Data-in-Place Conversions (DIP)

- Upgrade from a lower model to any higher performing model of the same type
  - Both All-Flash and Hybrid systems
    - All-Flash -> All-Flash
    - Hybrid -> Hybrid
  - Physical systems only (No UnityVSA)
  - Unity DC arrays do NOT support DIP conversions
- DIP conversion is a CUSTOMER procedure
  - Uses a wizard to guide the steps
  - Offline procedure
- D@RE does not affect ability to perform DIP conversion

Data-in-Place conversions are now supported with the Unity OE v4.1 release. Data-in-Place (DIP) conversion is a procedure used to upgrade storage processors while leaving all data and configurations intact. For example, a Unity 300 can be upgraded any higher performing model (Unity 400, Unity 500, Unity 600). This new feature addresses the case where the customer wants higher performance and/or hit the system limits for the current model.

Conversions allow for a Unity array to be upgraded to any of the higher models. DIP conversions are supported for Unity physical systems only and not UnityVSAs.

Conversions include both All-Flash and Hybrid systems, All-Flash > All-Flash and Hybrid > Hybrid systems. The exception and limitation to this statement is the Unity DC arrays do NOT support DIP conversions.

A DIP conversion is an offline CUSTOMER procedure that steps users through a wizard.

For customers with D@RE enabled, the procedure does not affect the ability to perform DIP conversion.
The slide shows the available upgrade paths for the Unity OE v4.1 release. As the slide shows, any Unity storage array can be upgraded to a higher performing array.

DIP upgrades are applicable for the entire Unity product line, both hybrid and All-Flash.

**Notes:** Downgrading a Unity array is not supported.

Unity DC arrays do NOT support DIP conversions.
In order to perform a DIP conversion, there are a few things to consider. Since the operation must be performed offline, both SPs must be powered off before the swap can be completed.

Only SPs are swapped; chassis and drives stay intact.

Any CNA SFPs, I/O modules and power supplies need to be moved during SP swap. No need to replace internal components such as the DIMMs, M.2 SSD device or fans, etc.

- Hybrid model to All-Flash model or vice versa is not supported.
- DIP conversions are only performed on Unity systems from lower to higher models.
- DIP is not supported on UnityVSA models (no hardware).
Data-in-Place conversions can be performed from Unisphere by navigating to the **System > Service** menu and selecting **Service Tasks > Hardware Upgrade**. Be aware the hardware upgrade procedure may take up to 120 minutes. This procedure will involve full shutdown of the storage system, halting of all I/O services, and physically replacing hardware components. The Upgrade Wizard displays the current hardware model and prompts the user to select a target Unity model for the upgrade. Note that only higher performing models will be displayed as available options.

Once a target model is selected, it is recommended that you perform a system health check prior to starting the upgrade. This ensures the system is in a healthy and upgradable state. If there are issues that need attention, "Warning" and/or "Error" messages may be displayed. Note that Warnings typically do not prevent the system from performing the upgrade whereas "Errors" need to be resolved before continuing. Read the Summary page and then continue the upgrade.

Start the upgrade process. The process now goes through a series of steps. It will perform a health check, prepare the system.

The system halts so the Customer or CE can safely pull the power cables, swap SPs and power supplies. Note the swap process is dependent on the number of I/O modules that will be relocated, number of SFPs, and whether new blades are out of the shipping package, etc.

Once the swap process completes, the upgrade will continue by updating the firmware, reimaging the M.2 device, starting the system services (stack startup time depends on the number of configured storage resources and other configurations), and performing a cleanup.

The result should now display the target system personality. The total time to complete the process is anywhere from 48 to 120 minutes. Note the Unisphere GUI displays the upgrade time of 120 minutes.
Before performing a Data-in-Place conversion, the Customer or CE should have conversion kit on site.

The kit includes the following:

- Two (2) brand new SPs
- Six (6) cable clamps
- New part number sticker sheet
- One page document pointing the user to Unity InfoHub
- Data-in-Place Conversion Guide available online
When the Upgrade wizard halts the array, the CE or customer will begin the physical hardware replacement steps. The steps are performed in the following order:

1. Ensure both SPs have halted. The SP fault light and power light should be off on both SPs due to halt; and power supply fault lights turn solid amber.
2. Put cable clamps on I/O module, CNA, and onboard port cables for both SPs.
3. Remove the power cords for SPA and SPB and wait for the DAE to power off.
4. Remove all other cables for SPB.
5. Remove old SPB.
6. Insert new SPB.
7. Insert CNA SFPs, I/O modules, power supplies from old SPB, and insert all cables except power for new SPB.
8. Repeat steps 4-7 for SPA
9. Insert power cords in both SPA and SPB
10. Put new part number sticker over PSNT tag for new model. Note the serial number does not change and update the Install Base (IB) after the procedures are complete.
There are scenarios that could potentially disrupt the DIP upgrade process due to customer error. Note their will always be a way to back out of a DIP conversion.

The customer mistakenly initiates a DIP for a different model than the actual hardware. In this case, the DIP process says there’s a hardware mismatch and the SPs will boot into service mode. The customer can then do one of two things, replace the SPs with the old ones and cancel the DIP conversion. The customer then has his original system back. Or as a second option, if the CE has the correct SPs for the conversion model he was trying to upgrade, insert those SPs and continue the conversion process.

There could be a case where the system boots up with a new SP, (an SP different from the model selected for the DIP) and an old SP (original SP) For this scenario, one of the SPs goes into service mode and the DIP process tells the user there’s an issue such as a hardware mismatch. The customer can either replace the “old” SP with the new SP to finish DIP upgrade, or replace the “new” SP with the old SP and cancel the DIP process.

If the DIP process fails before SPs are powered off, the customer can cancel the upgrade process in the wizard and troubleshoot as necessary.

If a user follows the HW Upgrade wizard to the point where both SPs are shutdown, the user can then simply power the system back up to Cancel the upgrade process automatically.
Data-in-Place conversions use the log information in the same OE upgrade logs as before. Log information can be found by opening the upgrade.log at /var/tmp/upgrade/upgrade.log. Bootflash files are located in the /var/log/boot directory.

Upgrade documentation includes the EMC Unity Family Data in Place Conversion Guide found on the Unity InfoHub https://community.emc.com/docs/DOC-51785 or by using the EMC Unity Product Support Page at http://support.emc.com.
This demonstration will cover the process to perform a Data-in-Place conversion from a Unity 300 to a Unity 400 storage system.
Check Your Knowledge

Which option is a valid Data-in Place conversion choice?
A All-Flash to All-Flash  
B. Hybrid -> All-Flash  
C. All-Flash to UnityVSA  
D. Hybrid to Unity DC arrays

The correct answer is “A”. All-Flash to All-Flash.

You cannot go to All-Flash from a hybrid nor are UnityVSA and Unity DC arrays valid.
Module Summary

Key points covered in this module:

- Data-in-Place feature overview
- Upgrade path for Unity storage arrays
- Considerations for implementing a DIP conversion
- Data-in-Place conversion process

The module covered the new Data-in-Place conversion feature targeted at improving scalability for the Unity product. We looked at the available hardware upgrade path/s for each model. Students should be able to identify the DIP conversion process and the considerations to be aware of when performing the conversion. Other topics include identifying some possible DIP scenarios and locating the proper documentation to support the product.
Module: Persistent File Data Cache (PFDC)

Upon completion of this module, you should be able to:
- Describe the PFDC feature and functions
- Explain PFDC architecture and operations
- Describe the implications of using PFDC other data services

This module focuses on Persistent File Data Cache (PFDC) feature implemented with the Unity OE v4.1 release. The module provides an overview of the feature, architecture, operations, and implications when using other Unity data services.
The PFDC feature is new for the Unity OE v4.1 release and is designed as a mechanism to delay allocating space within the LUN for data that potentially can be compressed. PFDC delays potentially expensive mapping operations until after an I/O request has been acknowledged back to the host.

PFDC is supported for both file and block but not VVols. PFDC is backward compatible meaning it is supported on existing resources that were created before any upgrade operations.

The PFDC feature requires no interaction from a user's point of view and leverages existing Unity architecture including pool slices and system cache.

No special configuration is required as the feature is built into the code. There is however a recommendation to have enough flash capacity in a pool to accommodate for all metadata slices.
The data log or circular buffer is created for temporarily storing write I/O. Writes are acknowledged back to the host quicker since the mapping of metadata is deferred until after the host acknowledgement. The data log stores the writes in preparation for sending the data to the ILC engine.

PFDC shortcuts the mapping layer of the Common Block File System (CBFS) by writing into a circular buffer hosted by CBFS on a few slices provisioned on one or more private LUNs when the file system is created (mkfs).

When the FS is created, the data log is built from two x 256 MB slices of the storage resource creating a single 512 MB region per storage resource (excluding VVol resources).

Slices are reserved on disk as with regular slices. Data log I/Os land in cache, as with other writes, but I/Os are removed from the data log before ever being flushed to disk.
EMC recommends provisioning a Flash tier in hybrid pools. The minimum recommended Flash capacity is about 5% of the pool capacity. This ensures metadata can be accessed and updated quickly in order to flush the PFDC buffers. The highest tiered capacity will reserve space for new metadata slice allocations which is 10% of the tier’s capacity.

FAST VP will try to maintain even an distribution within the extreme performance tier at every re-optimization procedure.

During the creation of a pool, if enough FLASH capacity is not available, the user will get an alert which points to the Unity Best Practices Guide.
In order to ensure low latency and keep up with the incoming I/O, the PFDC must be able to flush the data log in a timely fashion. This requires that mapping information and metadata associated with the file system be readily available on flash storage. If not, the PFDC will bypass (go into bypass mode) the data log and revert to the traditional I/O path where any incoming write will request a mapping from the underlying Common Block File System (CBFS) prior to acknowledging the host I/O.

In the Unity OE v4.1 release, metadata slices are considered as having the highest priority over user data regardless of their temperature. Unity OE v4.1 ensures that metadata is always given precedence in the storage pool for the fastest disks. This is a change from previous Unity releases where the MLU slice manager and FAST VP simply ignored the LUN or file system tier preference for metadata and removed the FAST VP “No Movement Policy”.

Similar to the relocation operation in previous releases, a new FAST VP “re-optimization” process identifies displaced metadata slices and automatically moves them to flash drives. This operation is dynamic and independent of relocation cycles. The lightweight slice relocation window only considers metadata slices. When an operation that frees slices on the flash tier runs (such as a snap deletion, LUN deletion, pool expansion, etc.), any ‘misplaced’ metadata slices (CBFS metadata that is housed on non-Flash based storage) are moved up to the flash tier within a short amount of time (outside of a relocation window).
A storage pool consists of a number of RAID Groups, which are built from drives of a certain type, for example, a 2+2 RG built from flash drives or a 4+2 RG built with HHDs.

Atop each RAID Group, a Private LUN will be created for each different RG. These private LUNs are split into a continuous array slices that are 256MB in size. Slices hold user data, metadata or may not be used at all. This is the granularity at which FAST VP moves slides to the various tiers in the pool.

Once the Private LUNs are partitioned out in 256MB slices, they are consolidated into a single Unified storage pool (also known as a slice pool).

Depending on the performance of the RAID Groups (depending mostly on the type of drives used), they all are separated into three tiers:

- Extreme Performance (EP) including RAID groups built from SSD drives
- Performance – built from SAS drives
- Capacity – built from NL-SAS ones
The slide shows the underlying unified storage pool that is used by both block and file resources. Once the pool is created, different containers will be built for the resource. The storage pool is responsible for the distribution of storage space between storage resources. Storage pool consumes storage space from Private LUNs based on RAID groups and allocates/deallocates storage space (for its tenants) in units called slices, each slice of 256 Mbyte size.

The NAS server is the storage system component responsible for creating file system and providing access to the file systems using standard protocols such as CIFS, NFS with access control. A NAS server container will be created when the user creates a NAS server. The container and will be the underlying backing storage used by the client file systems. In a similar manner, a file system container can be created. This container holds a single file that will be the backing storage for the file system accessed by the client. When a snapshot is taken, the snapshot will be a replica of the file system.
The example compares the write I/O path for pre-Unity OE v4.1 releases (without PFDC) to the current implementation with PFDC.

In a pre Unity OE v4.1 release write operation, a host write comes in to the array.

A mapping operation takes place to determine the location for the incoming I/O. This takes time to complete!

The I/O is placed into cache and written to the UFSLOG and an acknowledgement is sent back to the host.

A flush to the backend disks is a read-modify-write operation so in effect, a single Host I/O has a write penalty of 4. (1. Read data, 2. Read parity, XOR the data. 3. Write out the new data, 4. Write out the new parity).

With PFDC, The host performs a write.

The write I/O gets placed in a data log (circular buffer) and is immediately acknowledged to the host.

The mapping operation is done by a separate thread in the background saving time and decreasing latency.

A flush to the backend the system will attempt to do full stripe writes when possible.
Prior to the Unity OE v4.1 release, a host write first must be mapped to the underlying private LUN and written to system cache before being acknowledged back to the host. PFDC was implemented to provide a mechanism for delaying allocating of space within the LUN for data that potentially can be compressed. The example shows a host write to LBA 75.

The first thing that must be done is to map the host LBA to a file system block in the LUN container (CBFS).

Once the mapping occurs, we must map the LBA to the underlying Private LUN.

After the mapping to the Private LUN, the data is written to cache and acknowledged back to the host.

At some later time, the data will be written to the backend storage in a read-modify-write operation incurring four backend I/Os to the disks.
The example shows the data log slices in gold and their location in the LUN container (blue) and corresponding private LUN in the storage pool. Logically, the data log sits between the host and the mapping layers but physically is on the two 256 MB slices. As with all writes, a write to the data log will end up in cache. Before being flushed to disk, the data log entries will be re-mapped.

When the hosts writes to a LUN, in this case at LBA 75, the write indirectly corresponds to a file system data block that is unknown at this point. The write is written to the predefined file system (data log) block and the private LUN slice and then to cache and acknowledged back to the host immediately. The write is still in the data log and not yet mapped to its final location.

As other IOs come in we continually fill the data log and cache sequentially, in the same manner as the first write.

At a later time when the data log is full, the data will be re-mapped to its final destination in the file system in sequential fashion. Note this is only a movement of metadata, no memory copy operation is done.

Since the I/Os are aggregated at the system cache level, they appear as sequential when in fact, they were random at the start. At this point when a flush operation happens we can write the stripe down to the SAS drives as a full stripe write operation.
In order to do full stripe writes, we have to ensure we have continuous data to write. The windowing feature does this by segmenting the file system into 512 KB partitions. At any point in time we need to have some of these windows open to hold the writes so we can push the data down as full stripe write. The windowing process happens until a LUN starts running out of physical space. So over time the ability to perform full stripe writes due to windowing become less and less.

Data blocks are allocated in a way which forms a sequential stream in cache in 512 KB contiguous chunks. The File system maintains a set of free blocks in 512KB chunks and these blocks can be allocated when available.

Windows are produced by two means, a background space making process using defragmentation technology or by overprovisioning, using free space from pool if available.
The slide shows a PFDC remap operation. The example shows the file block address space. The space is divided into 512 KB windows. The intent of remapping is to aggregate blocks of data into a 512 KB space for PFDC to write sequentially to the backend drives in a full stripe write operation.

Incoming random I/Os gets written to the data log. In the example, data block I gets written to the data log followed by G, D, F, R.

If there are free windows available, the data is remapped to fill a 512 MB window. Once the remap occurs, the original data block pointers are invalidated. The process allows for better co-location on the file system and reduces processing.
Space Maker is an internal, background process, transparent to the user. Space Maker tries to limit the fragmentation by creating free block windows in a file system through the relocation of partially allocated windows. The end result provides large continuous sequential allocation windows for more efficient storage.

Space Maker will stop if a file system is deleting a snapshot or the FSR is in the process of evacuating a slice. This ensures that block relocation as part of space Maker does not interfere with these two operations as they all do extensive operations on data blocks.
The example shows the Space Maker operation.

First, Space Maker iterates the slices and determines if there are partial slices that are candidates for relocation.

In the example, slice “A” is evacuated and moved to a more suitable window and the 512 KB window is freed up.

Space Maker along with prefetching can improve performance for sequential read workloads by prefetching data blocks in cache.
Check Your Knowledge

How many data slices will be used by a single LUN resource?

A. 2 x 256 MB  
B. 1 x 512 MB  
C. 2 x 512 MB  
D. 1 x 256 MB

The correct Answer is “A”.

The log is 2 x 256 MB slices created from the pool for each resource so a single resource in this case a LUN would use 2 x 256 MB.
Module Summary

Key points covered in this module:

- PFDC feature overview
- the PFDC architecture and operation

The module covered the Persistent File Data Cache feature implemented in the Unity OE v4.1 release. We discussed the feature functions, architecture and operations. We saw how PFDC delays allocating space in the LUN until the Unity Compression feature determines how much space is needed for the compressed (or not compressed) data.
This module focuses on the new feature of Unity Compression also referred to as Inline Compression (ILC). Students should be able to understand and describe the ILC features and capabilities.

Student will also be able describe the ILC architecture and how ILC is integrated into the current software model. Lastly, the students should understand how the ILC feature operates.
Unity Compression or ILC is a new feature introduced in the Unity OE v4.1 release. ILC is intended to lower the cost per storage consumed by reducing the overall storage footprint.

ILC provides the ability to reduce the amount of storage needed for user data on a storage device, by compressing portions of the data at the time the data is first written or on data that is being overwritten. CPU resources that might otherwise go unused, are employed to perform the compression on the write path.

ILC is supported on physical hardware only. For Hybrid arrays, ILC is only supported on All Flash pools with no additional licenses required.

For the Unity OE v4.1 release, ILC will support compression on thin LUNs and VMFS Datastores only.

ILC depends on the infrastructure provided by PFDC. If the LUN is not using PFDC, then compression will not be used.

**Note:** If you need to convert a pool to a hybrid pool (both Flash and non-Flash drives), any LUNs that use compression must be deleted or moved. Hybrid pools cannot have compression enabled, and you cannot create a compression-enabled LUN in a hybrid pool. An all-Flash pool can contain both ILC enabled and non-ILC LUNs.
Unity Compression for LUNs

- ILC is disabled on all new LUNs by default
  - A checkbox is present in the LUN creation wizard
  - select the checkbox to enable ILC when creating a LUN

- ILC will not automatically enable on existing LUNs created on previous Unity codes when Unity OE v4.1 is installed
  - Can be enabled on existing LUNs on a LUN by LUN basis

In Unity OE v4.1, all new LUNs created on an All-Flash pool will have ILC disabled by default. The LUN creation wizard will have a checkbox and will have to be selected (checked) during the LUN creation if ILC is to be enabled on that LUN.

The example on the left shows the LUN properties page and the result of the initial creation of “ILC LUN”. Note the “Compression” box is unchecked, disabling ILC.

The example on the right displays the LUN properties page of All Flash LUN that has compression enabled. Note the Compression checkbox is displayed for thin LUNs in All-Flash pools only. It allows the thin LUN to be compressed to save space.

If an upgrade is performed from Unity GA release to Unity OE v4.1 release, existing LUNs will not automatically be enabled and the user will have to manually select ILC on a LUN by LUN basis.
Enabling ILC on a LUN

- Compression can be enabled or disabled at any time
- If enabling on an existing LUN:
  - All existing data is left uncompressed
  - Existing data is only compressed upon rewrite
- If disabling inline compression:
  - Data is left compressed until the data is overwritten or migrated by using “LUN Move”

Compression on a LUN can be enabled or disabled at any time. When enabled on an existing LUN, all existing data is left uncompressed and will only be compressed upon a rewrite of the data.

When disabling ILC, data is left compressed until the data is overwritten or migrated by using a “LUN Move” operation.
The “Compress Now” can be used to compress existing data on a LUN. The process however, requires a background move of the LUN data within the same pool so there could be a slight impact on the pool performance during the move operation. The option is initiated from the Block > LUNs pages by selecting the LUN and using the “More Options” > Compress Now selection.

The example shows the “AF LUN 1” has compression enabled as shown by the “Yes” under the Compression column on the right. By using the “More Options” dropdown, we can initiate a “Compress Now” process on the LUN. Note the graphic is shown with the Compression and Compression Savings columns. You must select the “Customized View from the menu bar and check the appropriate box to see these options.

If a LUN is in the process of moving, any replication cannot be configured on the LUN and once the move completes, any existing snapshots will be deleted.
The core logic of compressing data bytes resides inside the Common Block File System (CBFS). ILC is built upon the “flush” mechanism of PFDC. When data blocks are written to the CBFS container (FS), they are written to the data log of the PFDC and then to system cache and acknowledged back to the host. No mapping has been done at this point in time.

When PFDC flushes the data log, the flush operation calls the compression library to compress the data (using the Lempel-Zev algorithm) and then writes the compressed data to the buffer cache. Once the data is determined compressible, a number of contiguous blocks or “compression segments” are created on the CBFS and the data will be written to that segment.

The Unified Control Path provides the interfaces to the customer to control which LUNs are subject to compression, and also reports to the user the amount of achieved compression.

ILC requires Persistent File Data Cache (PFDC). Host writes are acknowledged before data is compressed. If PFDC is bypassed, compression is bypassed.

Data is compressed before de-staging writes to disk to reduce the amount of data that is written to disk. The ILC process does require CPU cycles to compress/decompress data.
The example shows a host write operation and how PFDC handles the I/O.

Without PFDC in place when a write comes in, we have to determine a location (LBA) in the LUN container then write to cache before we acknowledge the host.

With PFDC we point to a temporary location in the data log to save the data. The data is then written to system cache and the host acknowledgment is sent.

At some later time, the data is flushed out to a more ‘permanent’ location as determined by the file system mapping.
The example describes a PFDC flush operation. In the example, we see two pieces of data in R/W cache and the temporary pointers occupying the data log in the LUN Container. The data has not yet been moved to its final destination in the LUN Container at this point.

Data is aggregated and sorted in the data log and then stored in the buffer cache.

Data is processed by the compression library. After processing by the compression algorithm, a "product" is produced and savings are calculated. The product is then written to buffer cache.

If the system deems there is savings the system will look for a place to save the product. A set of N blocks will be allocated in the file system known as a "compressed segment" and the new compressed data will be written to this "compressed segment".

The new compressed data is then written to system cache replacing the old data and as well as freeing the locations in the data log.

Note that not all data is a candidate for compression. Certain data patterns are not highly compressible such as: video files, zipped files, etc.
ILC Space Maker

- Blocks can be freed as writes to new locations occur
  - Due to remaps, snapshot deletions, etc.
- Space maker is used to free unused blocks within a LUN
  - Attempts to make large, contiguous areas of free space
- If compression is disabled and compressed segments still exist, space maker process will continue to run on all blocks

With space maker, blocks can be freed by one process while allowing writes to any new location to continue.

Space maker is used to free unused blocks within a LUN and attempts to make large, contiguous areas of free space.

If compression is disabled and compressed segments still exist, the space maker process will continue to run on all blocks.
ILC can interoperate on compressed LUNs or VMFS Datastores.

When replicating, compressed data is uncompressed in memory and sent to the destination. The data is compressed on the destination if compression is enabled.

Replication can occur to or from a system running a previous Unity OE.

If replicating to a system running a previous Unity OE, data is written uncompressed on the destination array.

If replicating from a system running a previous Unity OE, compression can be disabled or enabled on the destination array.
Unity Snapshots allow for compressed data on LUN to be snapped. The left diagram shows four data blocks three of which are compressed and one that is not. Changes made to any of the shared data blocks will invoke a re-direct on write operation (ROW). Snapshots take advantage of compressed blocks on the source to save space. Blocks updated on the Snapshot can also be compressed if the source storage resource has compression enabled.

As shown on the right of the slide, when a block is modified on the source in this case “C” a ROW will occur and pointers will be updated.

The same is true for writes to the snapshot. When the write occurs, the original data “B” in the example, is saved and the pointers updated. Over time, snapshots are deleted and space is able to be freed by space maker.
All Flash pools cannot be expanded with SAS or NL-SAS disks while compressed segments exist within the pool. LUNs with compression enabled contain compressed segments, LUNs with compression previously enabled on it may still contain compressed segments.

Use LUN Move to relocate LUNs to a different Pool or LUN within the same Pool with compression disabled.
Estimating ILC Potential Savings

• Compression estimation tool is provided
  – Created by Mitrend
  – Similar to XtremIO data reduction tool

• Runs on:
  – Windows XP or later
  – ESXi 4.x or later
  – Linux 64-bit
  – Solaris 10 (x86/Sparc)
  – Solaris 11 (x86/Sparc)

ILC savings can be calculated using a tool available on Mitrend. For those familiar with the XtremIO reduction tool, the Unity version provides a similar look and feel.

The tool will run on:

• Windows XP or later
• ESXi 4.x or later
• Linux 64-bit
• Solaris 10 (x86/Sparc)
• Solaris 11 (x86/Sparc)
Demo: Compressed LUNs

This demo covers:

• Unisphere Dashboard
• Compressed LUN creation and editing

This demonstration will cover the Unisphere GUI options available for compressed LUNs. The demo will walk students through the steps to view saved capacity, create a compressed LUN, and edit the properties of a compressed LUN.
Check Your Knowledge

Which statement applies when the “Compressed Now” option is selected on a LUN?

A. All LUNs with ILC enable will be compressed
B. The selected LUNs data is relocated
C. All existing and new data will be compressed
D. Existing snapshots will be retained after compression

The correct answer is “B”.

The data on the selected LUN must be relocated to another Pool LUN.
In this module we talked about the Inline Compression feature and how it allows data to be compressed on a given LUN. By writing data in a compressed format, customers can better utilize the space savings on the storage system.

It was noted that for ILC to function, it required PFDC to be operational. We followed a write operation through the various software layers to the ILC compression library and saw how and where the LUN compression operation takes place. Students were made aware of which data services interoperate with ILC.
The course looked at the new scalability and performance features that were implemented in the Unity OE v4.1 release.

One of the key new features to improve scalability for Unity was to add support for Data-in-Place conversions. In the module, we looked the Unity models and the hardware upgrade path for each model. We also identified the DIP conversion process for performing the hardware swap operation and the considerations when doing so. Other topics included looking at some possible DIP scenarios and locating the proper documentation to support the product.

PFDC was designed as a temporary storage area called a data log in order to delay allocating space within the LUN container for the data’s final mapping address. Once the data was in the data log, it was aggregated and sorted in preparation for sending it to the ILC engine.

Inline Compression allows for data to be compressed on a given LUN if, in fact, the data was determined to be compressible. By writing data in a compressed format, customers can better utilize the space savings on the storage system. The ILC feature relies upon the PFDC feature to work. ILC topics included an overview of the feature, where the compression engine fits in the software layer, and showed some examples of how a host write goes through the compression engine.