connect OpenVMS Boot Camp 2011

September 18 – 22
Sheraton Hotel – Needham, MA

I302

OpenVMS Volume Shadowing Performance

Keith Parris, HP

Shadowing Performance

- Shadowing is primarily an availability tool, but:
  - Shadowing can often improve performance (e.g. reads), however:
  - Shadowing can often decrease performance (e.g. writes, and during merges and copies)

Shadow Copy/Merge Performance: Why Does It Matter?

- Excessive shadow-copy time increases Mean Time To Repair (MTTR) after a disk failure, or a site outage in a disaster-tolerant cluster
- Shadow Full Merges can have a significant adverse effect on application performance
- Shadow Full Copies or Full Merges can generate a high data rate (e.g. on inter-site links for Disaster Tolerant clusters)
  - Acceptable shadow full-copy times and link costs are often the major factors in selecting inter-site link(s) for multi-site disaster-tolerant clusters

Shadowset State Hierarchy

- Mini-Merge state
- Copy state
  - Mini-Copy state
  - Full Copy state
- Full Merge state
- Steady state

Transient states

Copy and Merge

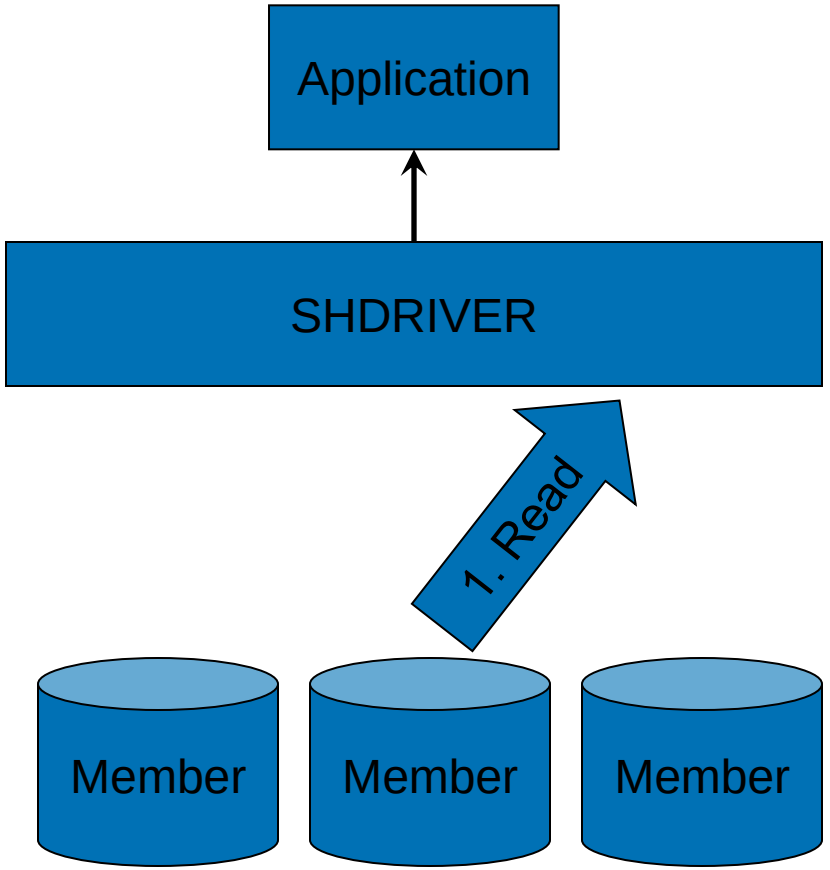
- How a Copy or Merge gets started:
  - A $MOUNT command triggers a Copy
  - This implies a Copy is a Scheduled operation
  - Departure of a system from the cluster while it has a shadowset mounted triggers a Merge
  - This implies a Merge is an Unscheduled operation

I/O Algorithms Used by Shadowing

- Variations:
  - Reads vs. Writes
  - Steady-State vs. Copy vs. Merge
    - And for Copies and Merges,
      - **Full- or Mini- operation**
      - **Copy or Merge Thread I/Os**
      - **Application I/Os: *Ahead of* or *Behind* the Copy/Merge "Fence"**

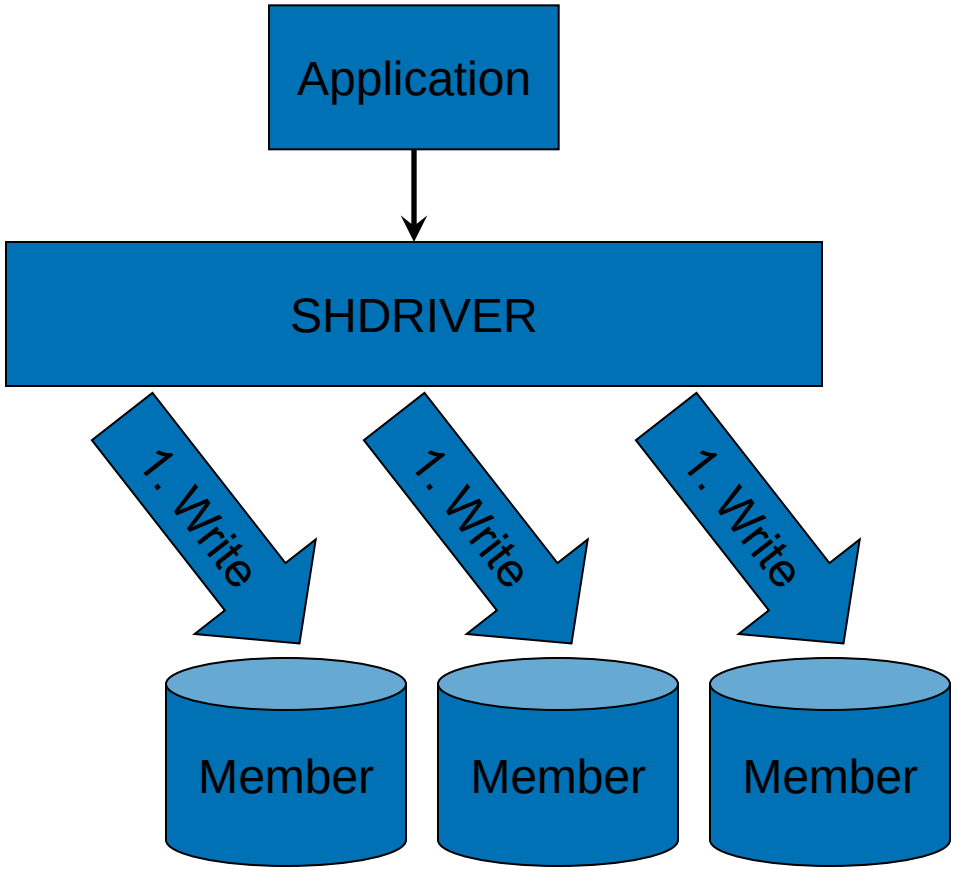
Application Read during Steady-State

- Read from any single member:
- Select a single source member to read from, based on lowest sum of queue length to each disk from the local node + the Read_Cost value for the disk
- Return data to application



Application Write during Steady-State

- Write to all members in parallel
- Once last write has completed, return success status to application

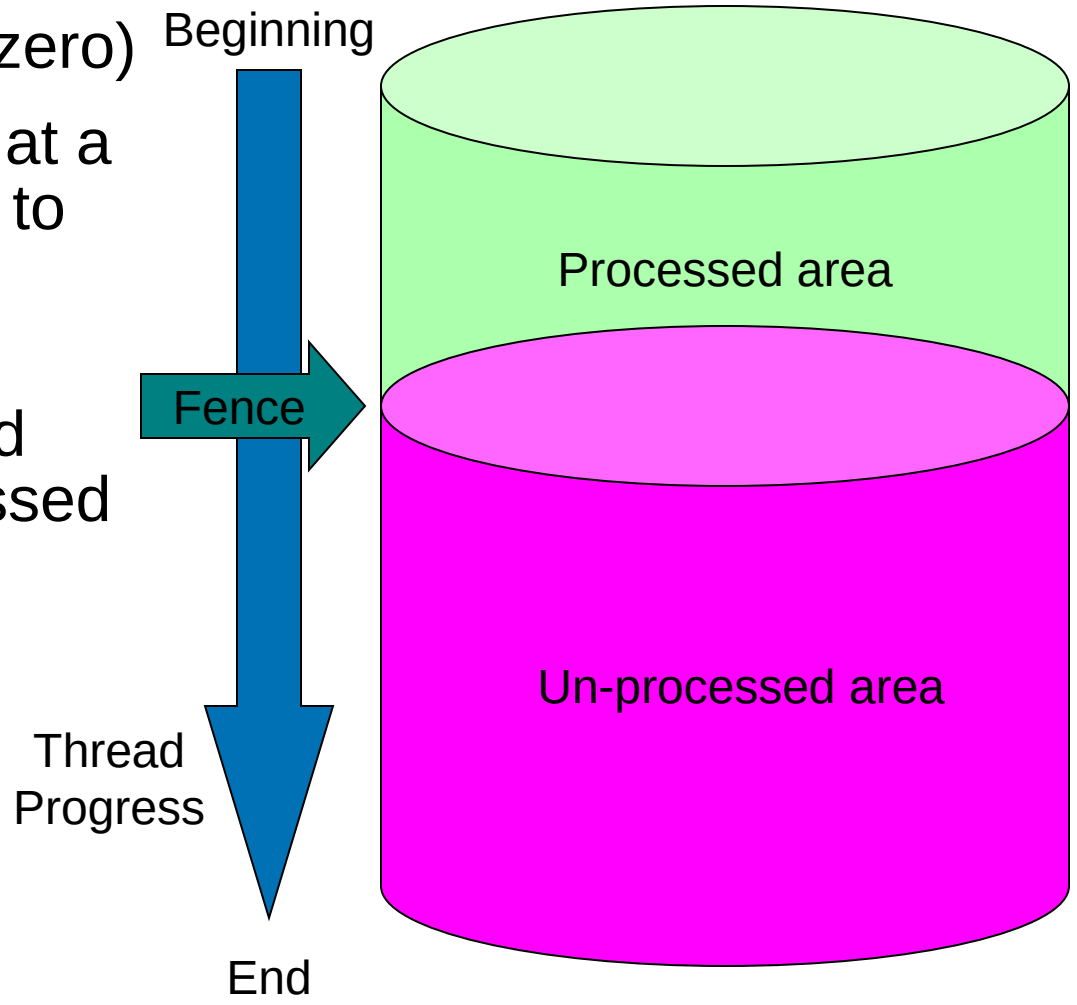


Full-Copy and Full-Merge Thread Algorithms

- Action or Event triggers Copy or Merge:
  - $MOUNT command triggers Copy (scheduled operation)
  - System departure from cluster while it has shadowset mounted triggers Merge (unscheduled operation)
- SHADOW_SERVER on one node picks up Thread from work queue and does I/Os for the Copy or Merge Thread
  - SHADOW_MAX_COPY parameter controls maximum number of Threads on a node
- SHAD$COPY_BUFFER_SIZE logical name controls number of blocks handled at a time
- No double-buffering or other speed-up tricks used

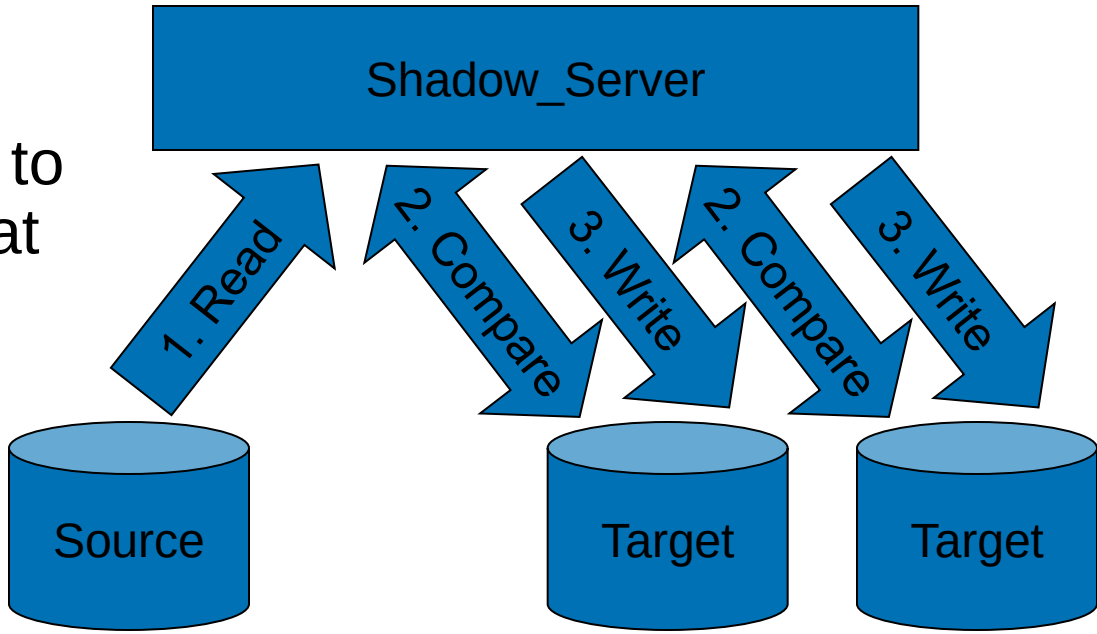
Full-Copy and Full-Merge Thread Algorithms

- Start at first Logical Block on disk (LBN zero)
- Process 127 blocks at a time from beginning to end
- Symbolic “Fence” separates processed area from un-processed area



Full-Copy Thread Algorithm

1. Read from (a) source member
2. Compare with target member(s)
3. If different, write data to target and start over at Step 1.



Full-Copy Thread Algorithm

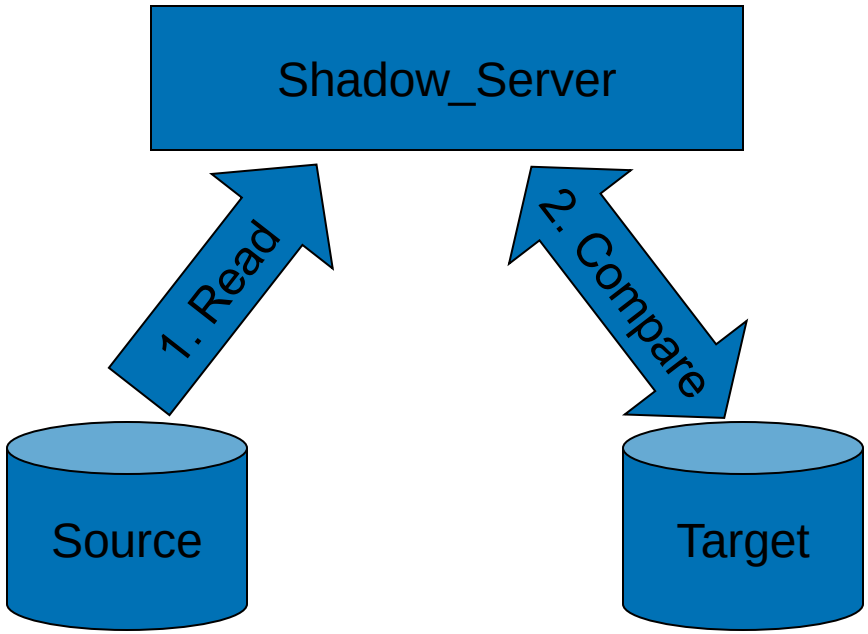
- Why the odd algorithm?
- To ensure correct results in cases like:
  - With application writes occurring in parallel with the copy thread
    - application writes during a Full Copy go to the set of source disks first, then to the set of target disk(s)
  - On system disks with an OpenVMS node booting or dumping while shadow copy is going on
- To avoid the overhead of having to use the Distributed Lock Manager for every I/O

Speeding Shadow Copies

- Implications:
  - Shadow copy completes fastest if data is identical beforehand
    - Fortunately, this is the most-common case – re-adding a shadow member into shadowset again after it was a member before

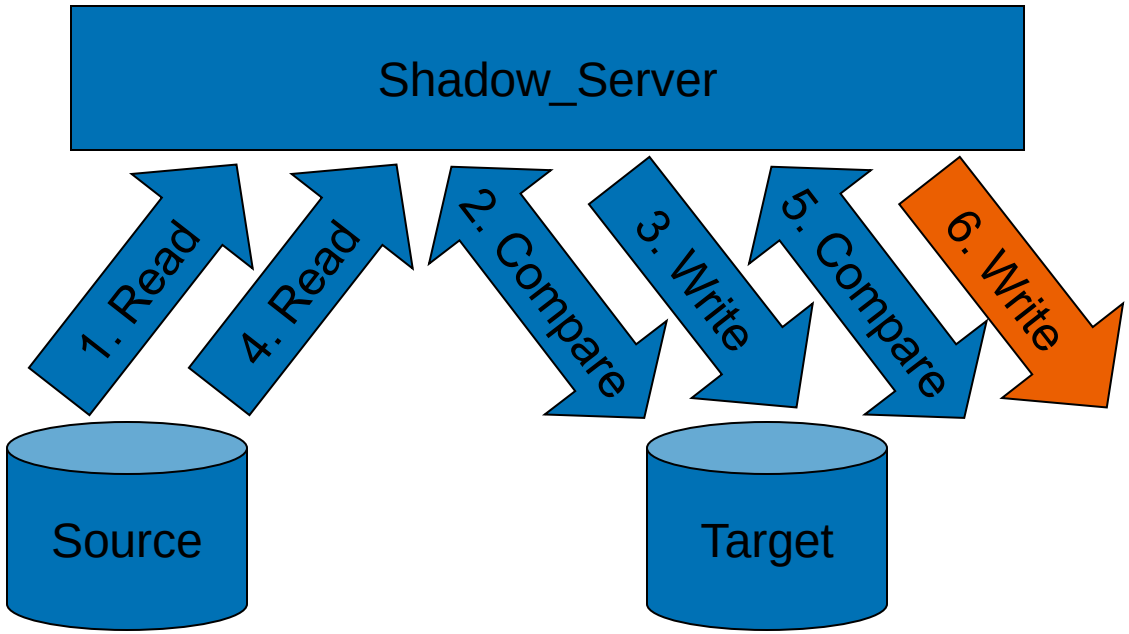
Full-Copy Thread Algorithm (Data Identical)

1. Read from source
2. Compare with target



Full-Copy Thread Algorithm (Data Different)

1. Read from source
2. Compare with target (difference found)
3. Write to target
4. Read from source
5. Compare with target (difference seldom found). If different, write to target and start over at Step 4.



Full-Copy Thread Algorithm (Data Different)

When will data be different?

- Replacing a failed disk with a new replacement disk
- Moving from an older disk model to a new, potentially larger disk model
- Re-using a disk previously a member of another shadowset
- etc.

Speeding Shadow Copies

- If data is very different, empirical tests have shown that it is faster to:
  1. Do BACKUP/PHYSICAL from source shadowset to /FOREIGN-mounted target disk
  2. Then do shadow copy afterward

  than to simply initiate a shadow copy with differing data.

  It is faster to do these two operations in sequence than to simply add the shadowset member in with different data.
  - Caution: If you're not running 7.3-2 or later (or don't have a recent Mount ECO kit on earlier versions) be sure to clobber the SCB on the target disk with a $MOUNT/OVERRIDE=SHADOW (or $INITIALIZE) command before adding new member to shadowset
    - Or else Mount gets fooled by identical SCBs and new disk is added as a Merge Member instead of as a Full-Copy Target as it should. Problem is fixed in 7.3-2 and in Mount ECO kits for earlier versions.

Why a Merge is Needed

- If a system has a shadowset mounted, and may be writing data to the shadowset…
- If that system crashes, then:
  - The state of any “in-flight” Write I/Os is indeterminate
    - All, some, or none of the members may have been written to
  - An application Read I/O could potentially return different data for the same block on different shadowset members
- Since Shadowing guarantees that the data on all shadowset members must appear to be identical, this uncertainty must be removed
  - This is done by a Merge operation, coupled with special treatment of Reads until the Merge is complete

What Happens in Full Merge State

- Full Merge operation will:
  - Read and compare data on all members
  - Fix any differences found
  - Moves across disk from start to end, 127 blocks at a time
  - Progress tracked by a “Fence” separating the merged (behind the fence) area from the unmerged (ahead of the fence) area
- Special treatment of Reads:
  - Every application read I/O **must** be merged:
    - Read data from each member (for just the area of the read)
    - Detect any differences
    - Overwrite any different data with data from Master member
    - Return the now-consistent read data to the application
  - This must be done for any Reads ahead of the Merge Fence

HBVS Merge Operations

Q: Why doesn't it matter which member we choose to duplicate to the others on a merge? Don't we need to ensure we replicate the newest data?

A: Shadowing reproduces the semantics of a regular disk:

1) Each time you read a block, you always get the same data returned

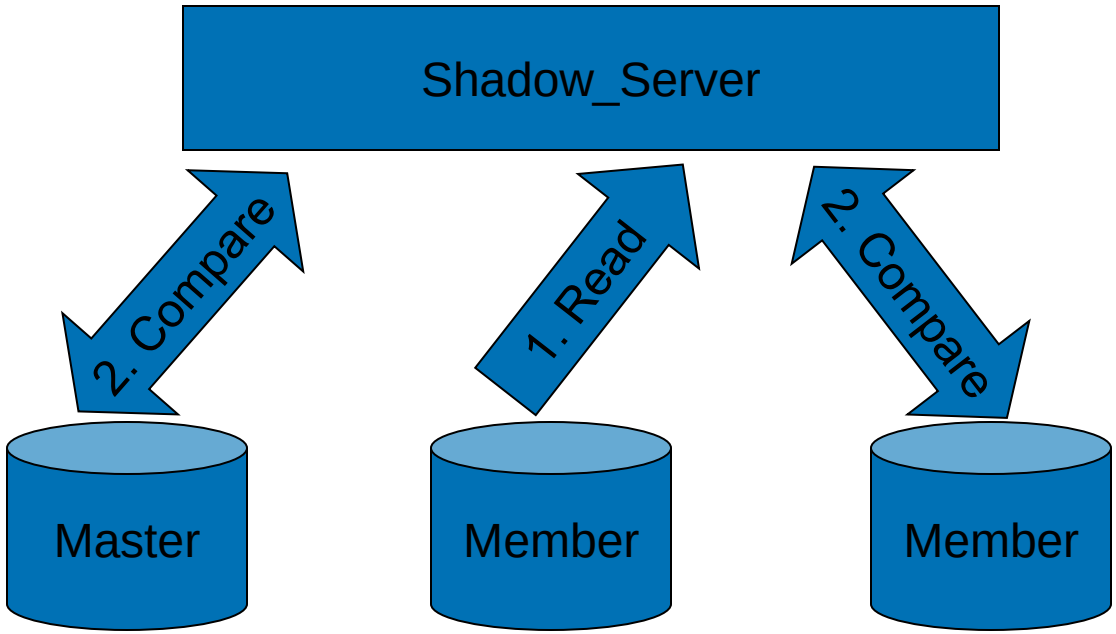
2) Once you do a write and get successful status back, you always expect to get the new data instead

3) If the system fails while you are doing a write, and the application doesn't get successful status returned, you don't know if the new or old data is on disk. You have to use journaling technology to handle this case.

Shadowing provides the same behavior.

Full-Merge Thread Algorithm

1. Read from any member
2. Compare with other member(s)
3. If different, do a Fix-Up: halt all I/Os to the shadowset, fix up differences using data from the Master member, then allow I/Os to continue



Creating Shadowsets

- Traditional method historically of creating a shadowset was to create a 1-member shadowset, then initiate a Full Copy operation

- But you can do:
  - $INITIALIZE/SHADOW=(disk1,disk2,disk3…) label

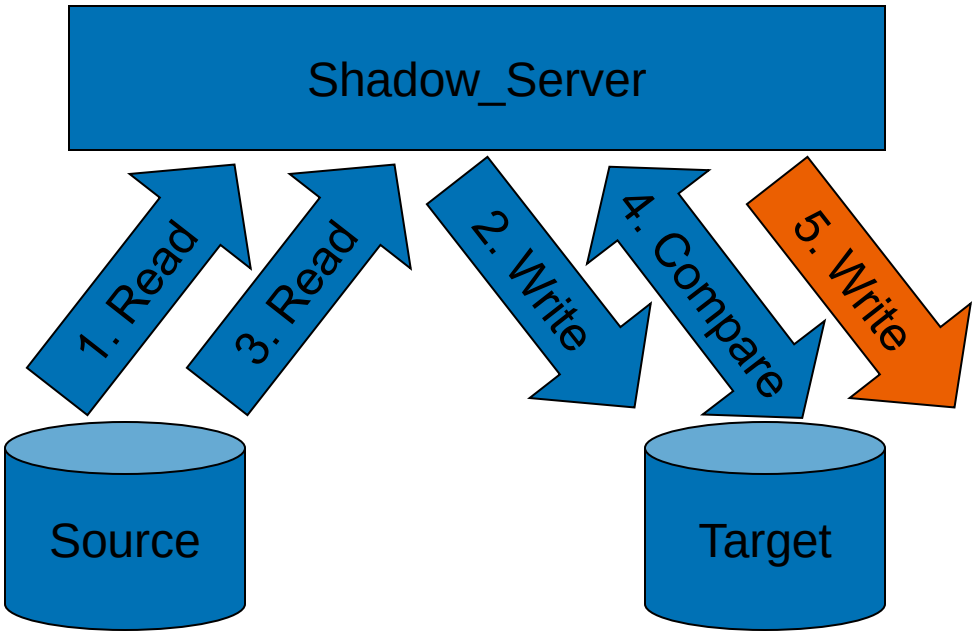
- Warning: Unless all of disk is written (i.e. by filling it with data, or using INITIALIZE/ERASE), the first Full Merge will be a busy one

Speeding Shadow Copies: Mini-Copy Operations

- If one knows ahead of time that one shadowset member will temporarily be removed, one can set things up to allow a Mini-Copy instead of a Full-Copy when the disk is returned to the shadowset
- For unexpected member removal, proper setup allows Automatic Mini-Copy on Volume Processing (AMCVP) to convert a mini-merge write bitmap designated as Multi-Use for use in a subsequent Mini-Copy

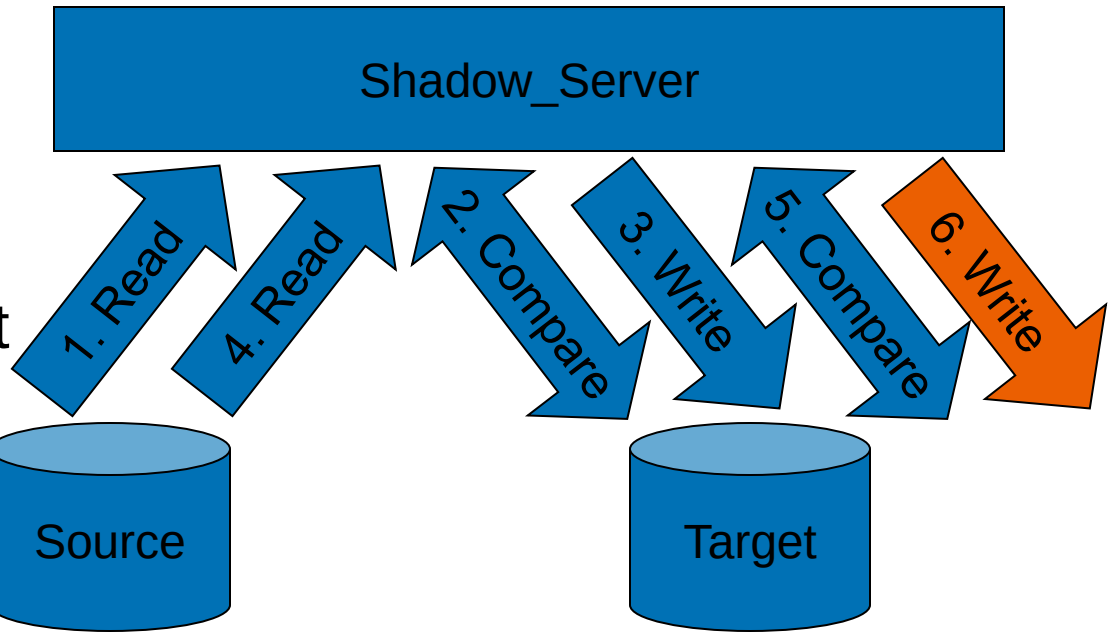
Mini-Copy Thread Algorithm

- **(Only for areas the write bitmap indicates have changed):**
- Read from (a) source member
- Write to target member(s)
- Read from (a) source member
- Compare with target member(s)
- If different (quite unlikely), write data to target and start over at Step 3.



Recall the Full-Copy Thread Algorithm (with Data Different)

1. Read from source
2. Compare with target (difference found)
3. Write to target
4. Read from source
5. Compare with target (difference seldom found). If different, write to target and start over at Step 4.



Mini-Copy vs. Full-Copy

- If data is different, Full-Copy algorithm requires 5 I/O operations per 127-block segment:
  1. Read source
  2. Compare target
  3. Write target
  4. Read source
  5. Compare target
- Mini-Copy requires only 4 I/O operations per segment:
  1. Read source
  2. Write target
  3. Read source
  4. Compare target
- So Mini-Copy is faster even if 100% of data has changed

Compare operation

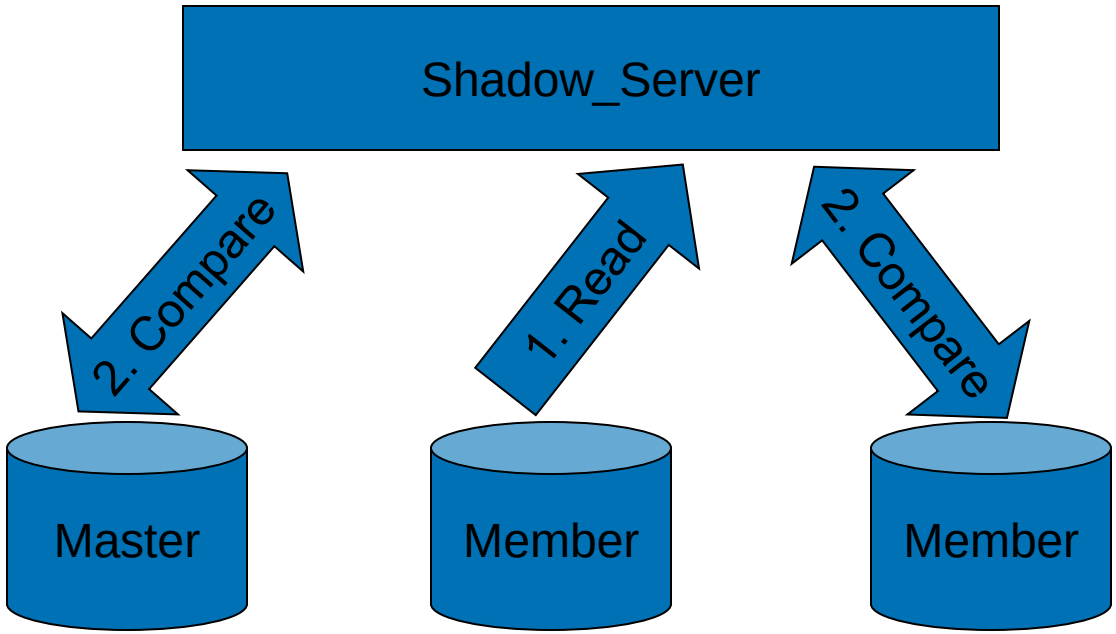
- On MSCP controllers (HSJ, HSD), this is an MSCP *Compare Host Data* operation
  - Data is passed to the controller, which compares it with the data on disk, returning status
    - Cache is explicitly bypassed for the comparison
      - So Read-Ahead and Read Cache do not help with Compares
    - But Read-Ahead can be very helpful with Source disk Reads, particularly when the disks are close to identical
- On SCSI / Fibre Channel controllers (HSZ, HSG, EVA), this is emulated in DKDRIVER
  - Data is read from the disk into a second buffer and compared with the data using the host CPU (in interrupt state, at IPL 8)
    - Read-ahead cache can be very helpful, particularly when disks are close to identical

Speeding Shadow Copies

- Read-Ahead cache in controllers and disks can assist with source read I/Os
- By default, Shadowing reads from source members in a round-robin fashion
  - But with 3-member shadowsets, with two source members, I/Os are divided among 2 sources. With 6 members, from 5 sources!
- $SET DEVICE /COPY_SOURCE device
  - Applied to shadowset member, makes the member the preferred source for copies & merges
  - Applied to Virtual Unit, current master member is used as source for copies & merges.

Full-Merge Thread Algorithm

1. Read from any member
2. Compare with other member(s)
3. If different, do a Fix-Up: halt all I/Os to the shadowset, fix up differences using data from the Master member, then allow I/Os to continue



Host-Based Mini-Merge

- One or more OpenVMS systems keep a bitmap of areas that have been written to recently
  - Other nodes notify (update) these nodes as new areas are written to
  - Bitmaps are cleared periodically
- If a node crashes, only areas recently written to, as indicated by the bitmap, must be merged
- If all the bitmaps are lost, a Full Merge is needed

Speeding Shadow Copies/Merges

- Because of the sequential, non-pipelined nature of the shadow-copy algorithm, progressing across the entire LBN range of the disk, to speed shadow copies/merges:
  - Rather than forming controller-based stripesets and shadowing those, shadow individual disks in parallel, and combine them into RAID-0 (0+1) arrays with host-based RAID software

Speeding Shadow Copies/Merges

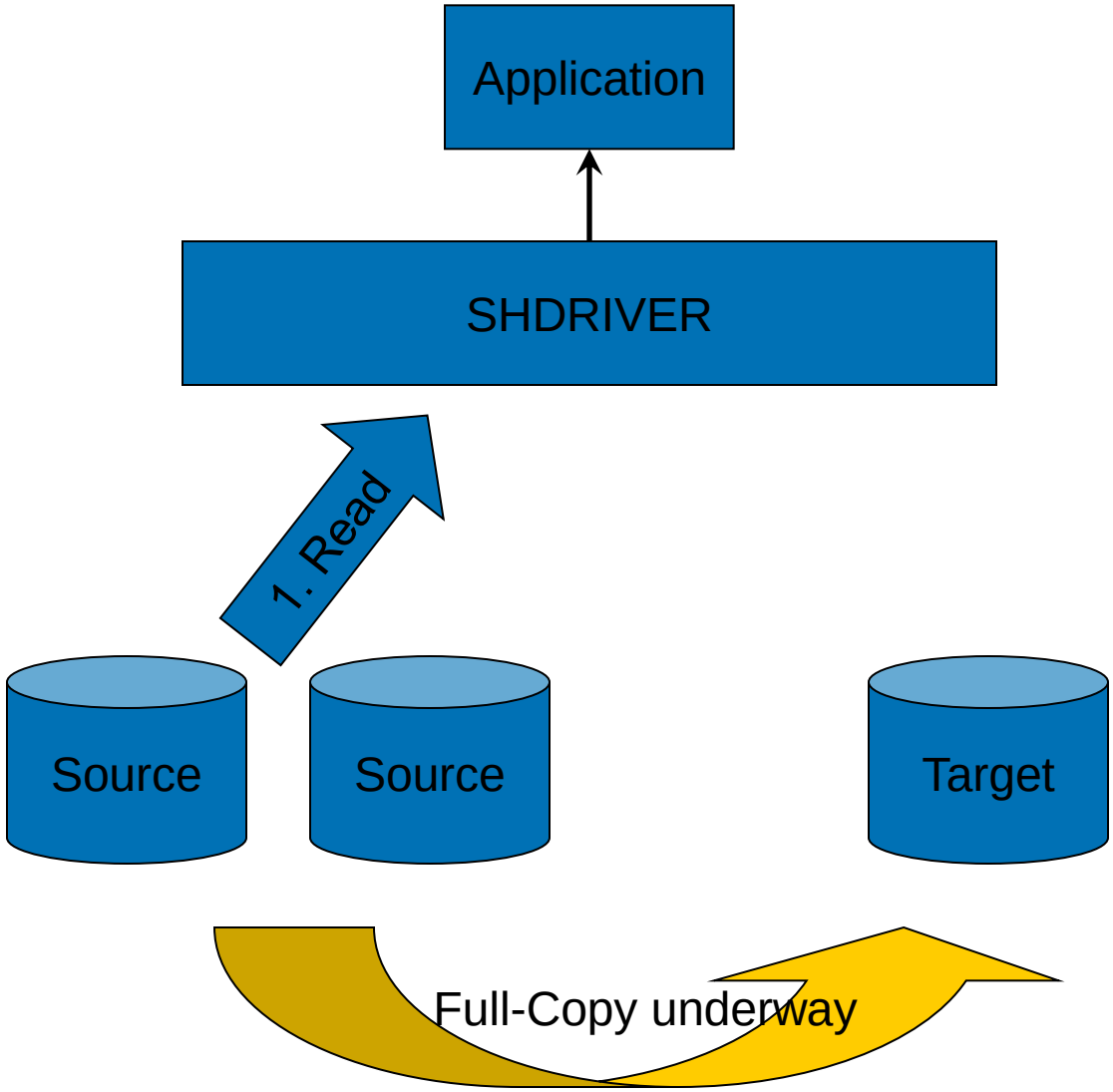
- Dividing a disk up into 4 partitions at the controller level, and shadow-copying all 4 in parallel takes only 40% of the time required to shadow-copy the entire disk as a whole (i.e. 2½ times faster)

Application I/Os

- During Full-Copy
- During Full-Merge

Application Read during Full-Copy

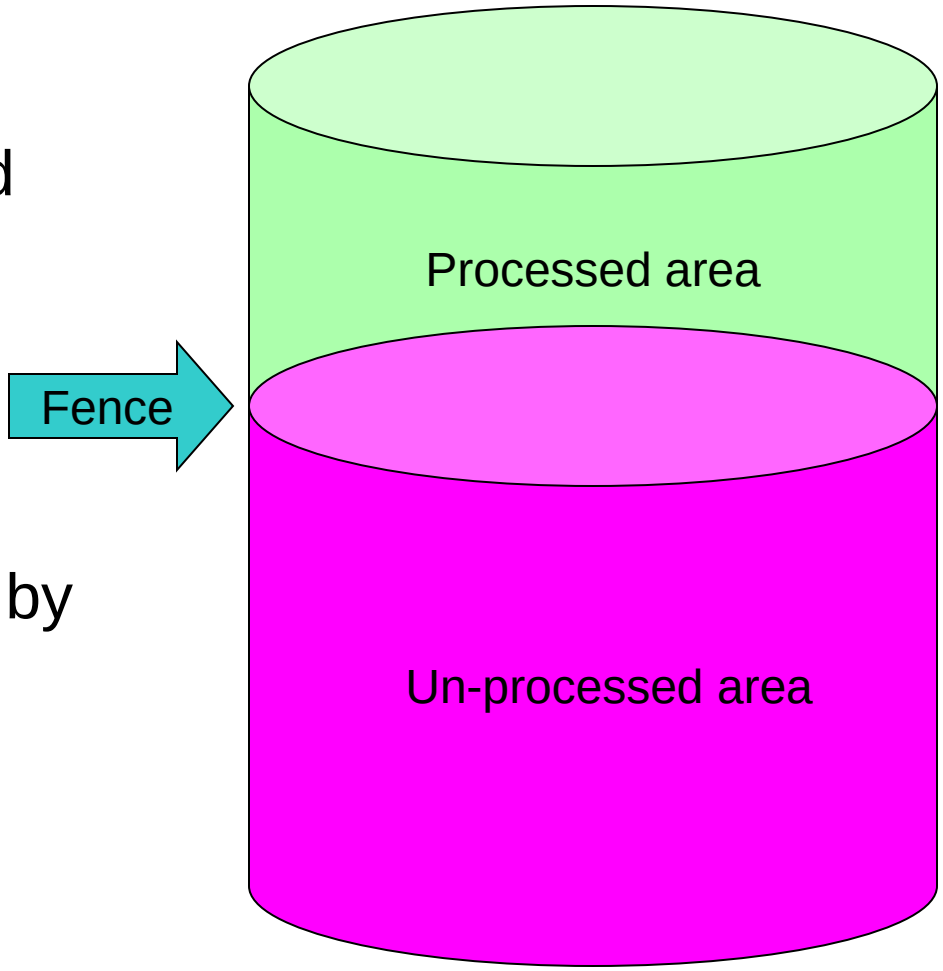
- Select a single source member to read from, based on lowest sum of local queue length + Read_Cost
- Never direct an application read to a target member
- Return data to application



Application Read During Full-Merge

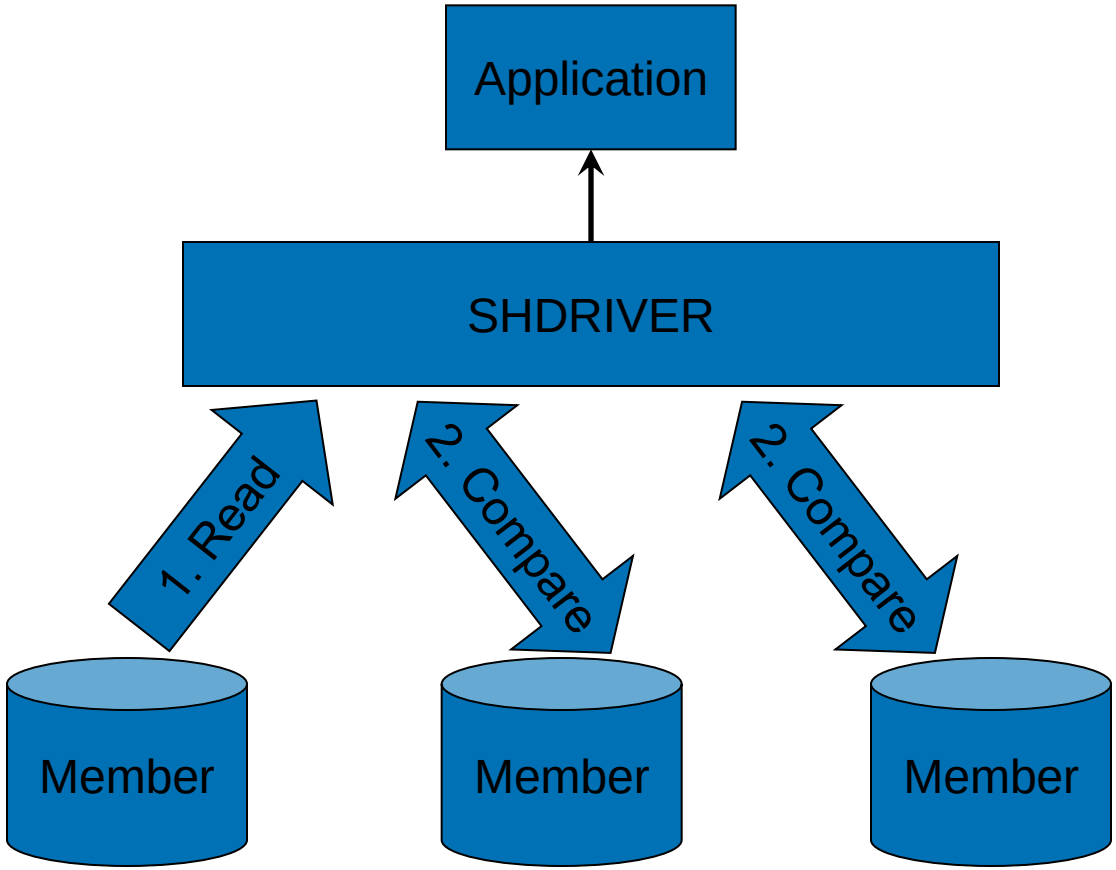
- In Processed area:
- Pick one member to read from

- In Un-processed area:
- Merge the area covered by the read operation, then
- Return data



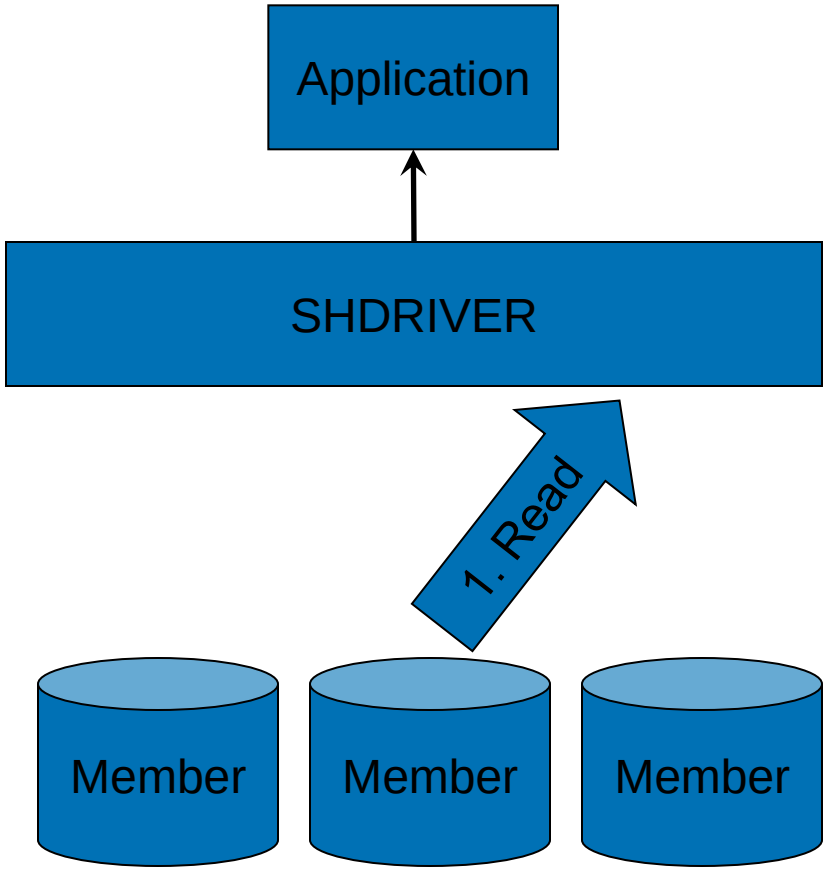
Application Read during Full-Merge; not-yet-merged area

- Read from any member
- Compare with other member(s)
- If different, do a Fix-Up: halt I/Os to the shadowset, fix up any differences using data from the Master Member, then allow shadowset I/Os to continue
- Return data to application



Application Read during Full-Merge; already-merged area

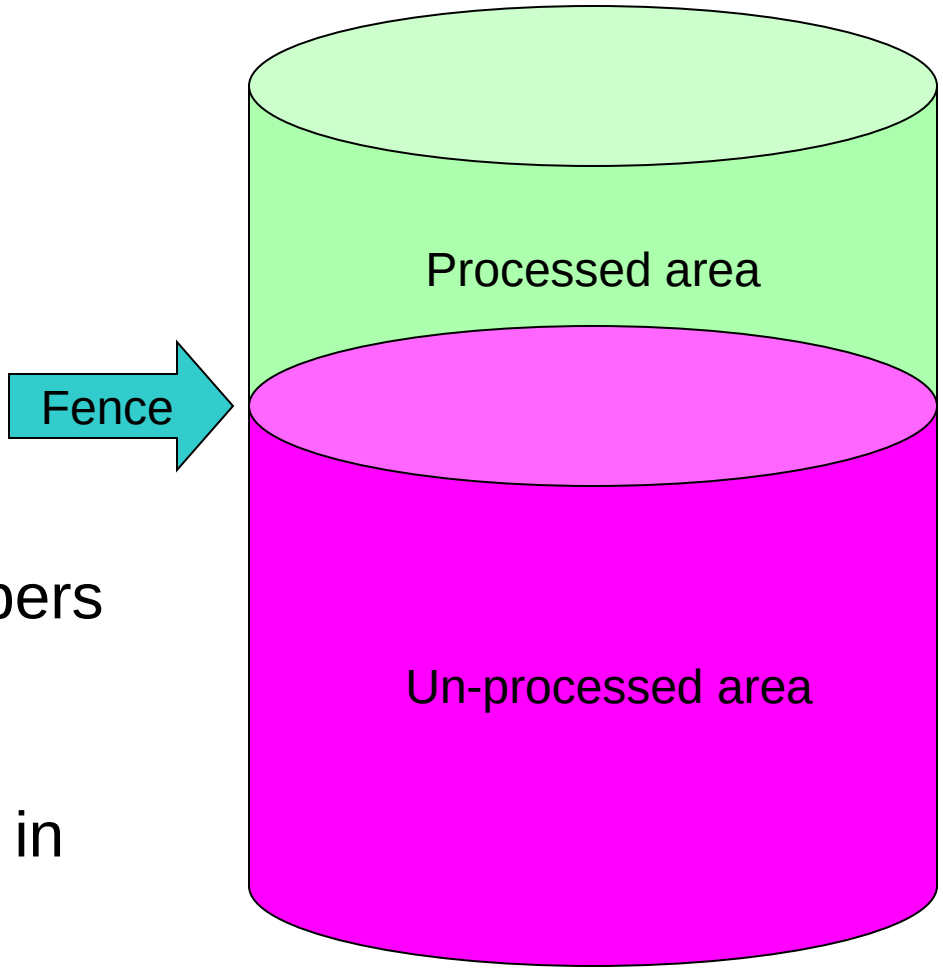
- Read from any single member



Application Write During Full-Copy

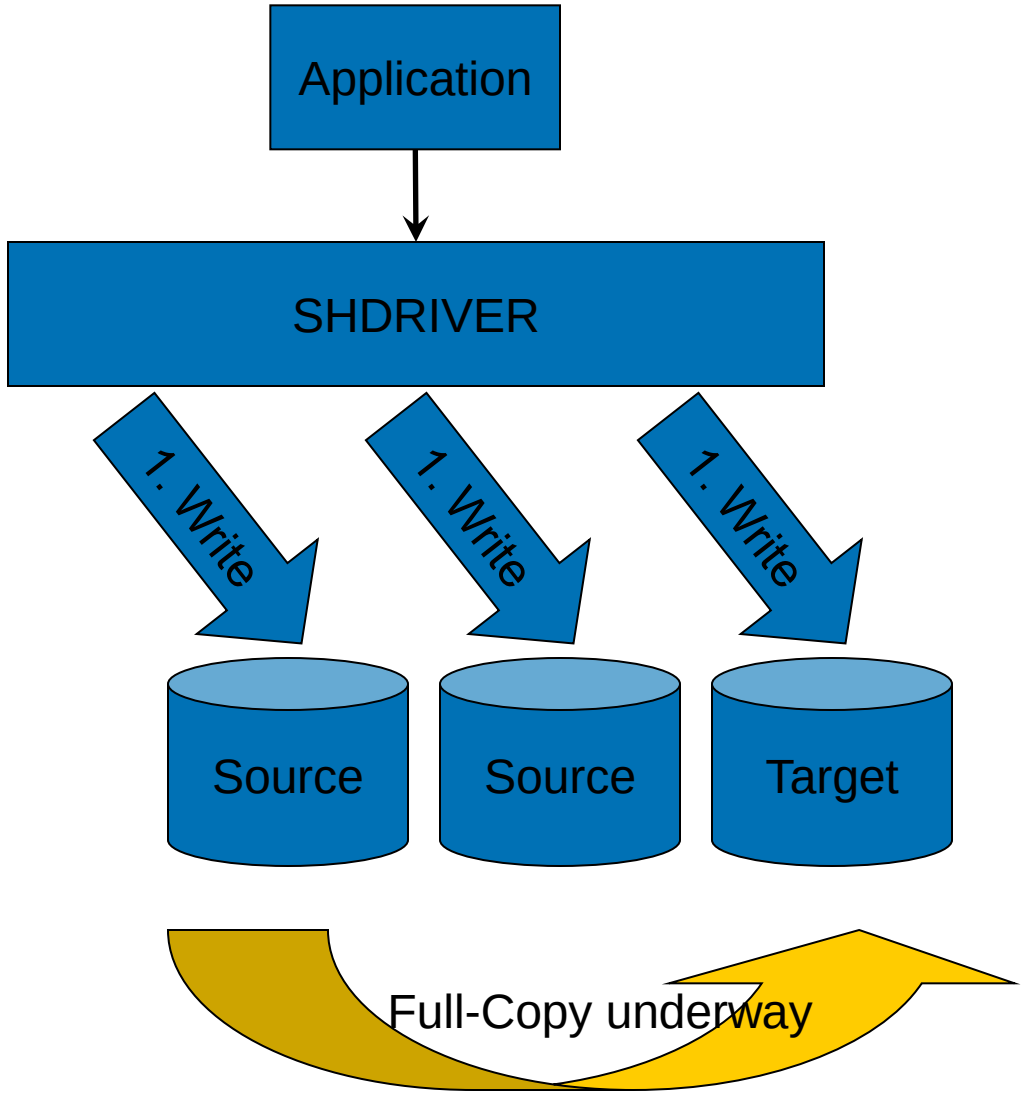
- In Processed area:
- Write to all members in parallel

- In Un-processed area:
- Write to all source members in parallel, wait for these writes to complete, then
- Write to all target disk(s) in parallel



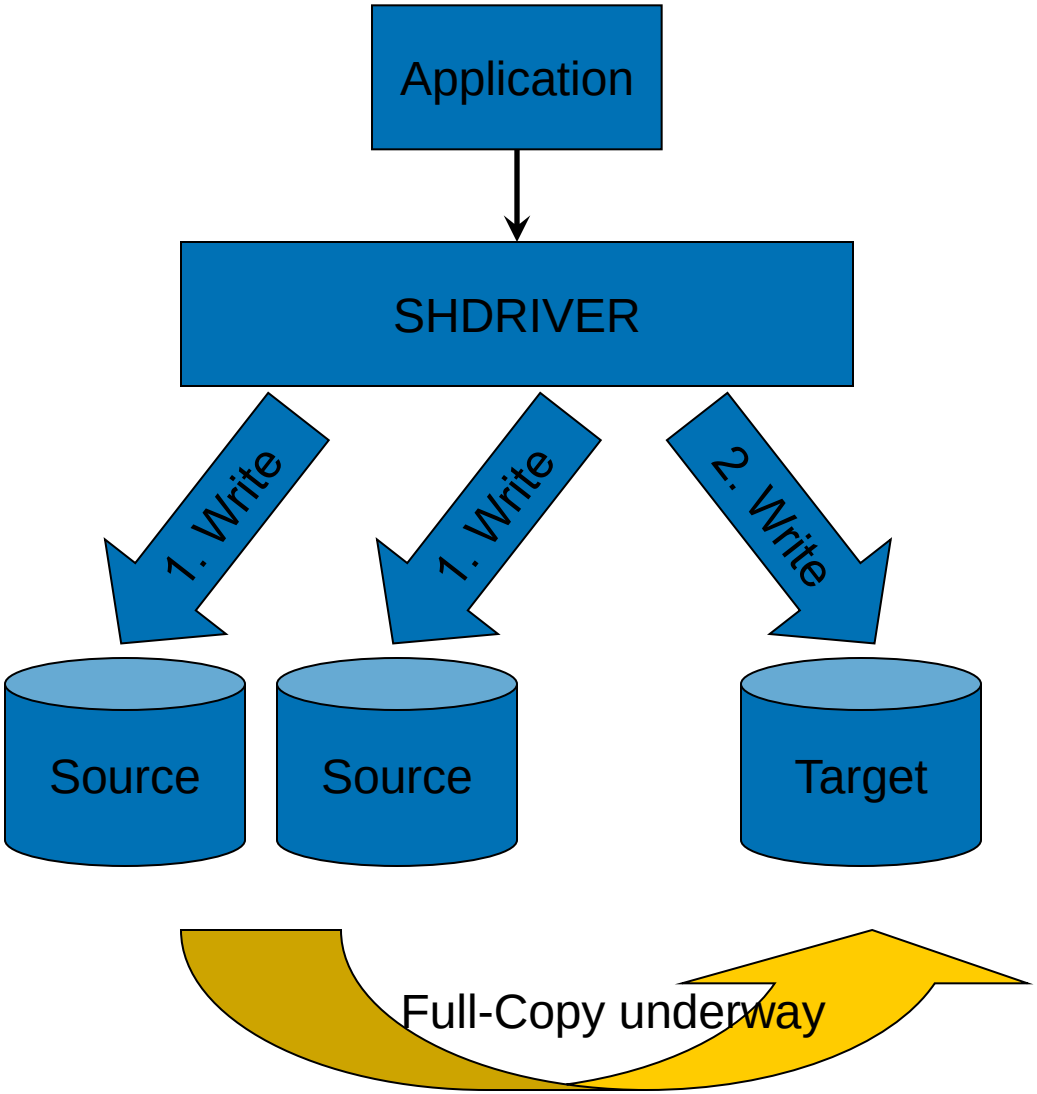
Application Write during Full-Copy; already-copied area

- Write to all members in parallel



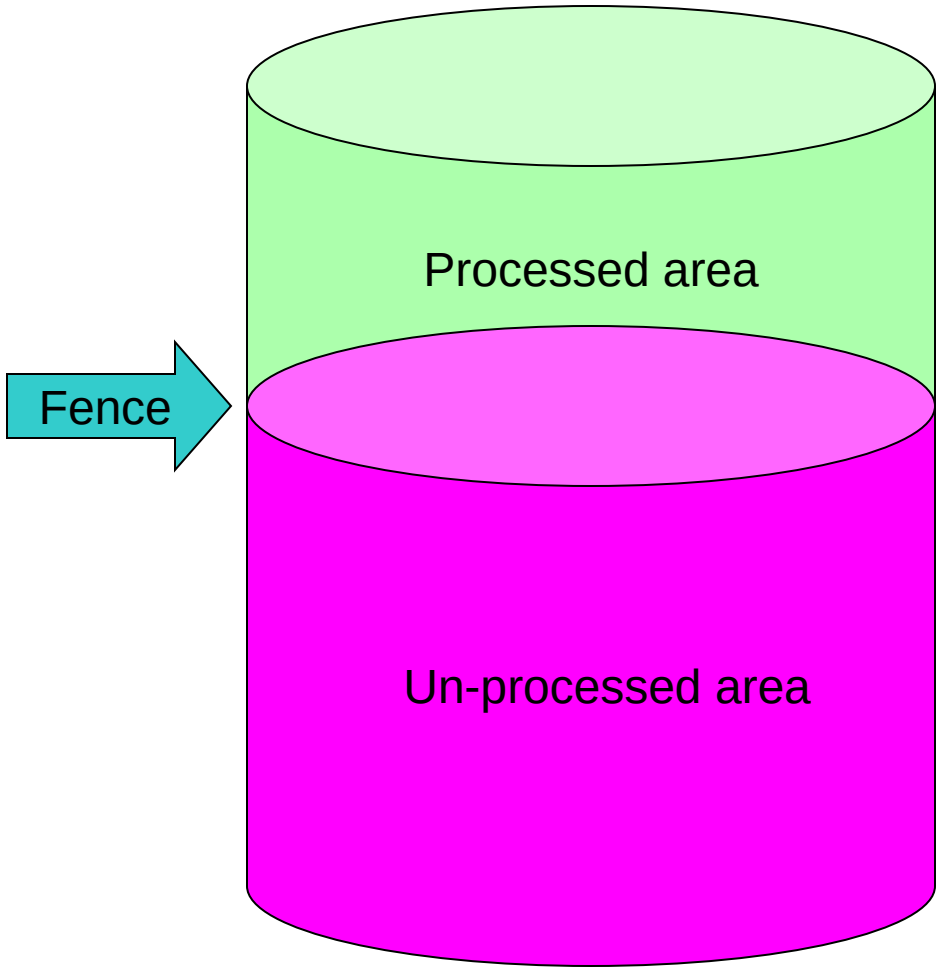
Application Write during Full-Copy; not-yet-copied area

- Write to all source member(s); wait until these writes have completed, then
- Write to all target(s)



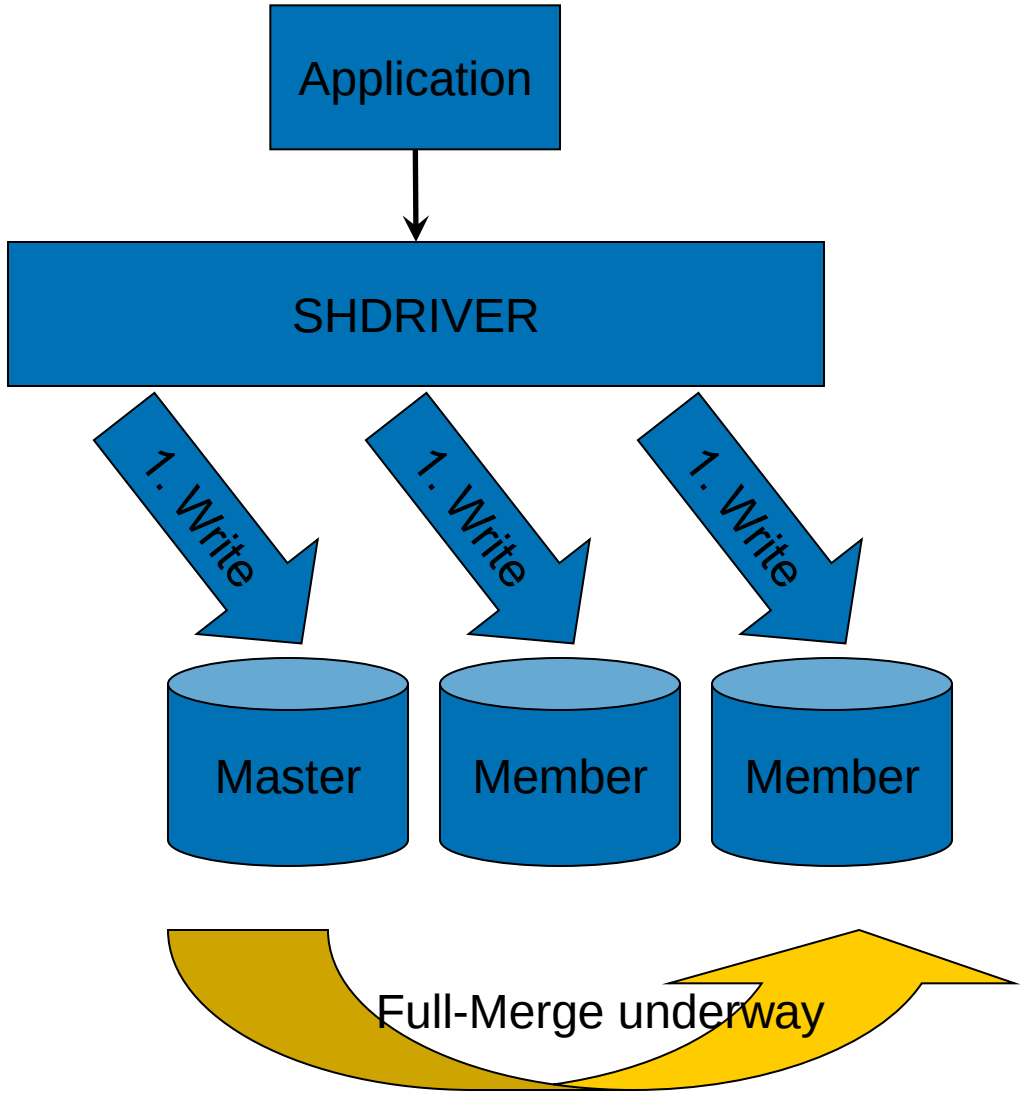
Application Writes During Full-Merge

- Regardless of area:
- Write to all members in parallel



Application Write during Full-Merge

- Write to all members in parallel



Shadowset Member Selection for Reads: Read Cost

- Shadowing assigns a default “read cost” for each shadowset member, with values in this order (lowest cost to highest cost):
  1. DECram device
  2. Directly-connected device in the same physical location
  3. Directly-connected device in a remote location
  4. DECram MSCP-served device
  5. All other MSCP-served devices
- Shadowing adds the Read Cost to the Queue Length and picks the member with the lowest value to do the read
  - Round-robin distribution results for disks with equal values

Shadowset Member Selection for Reads: Read Cost

- For even greater control, one can override the default “read cost” to a given disk from a given node:

  $ SET DEVICE /READ_COST=nnn 1DGAnnn:

  For reads, OpenVMS adds the Read Cost to the queue length and picks the member with the lowest value to do the read

- To return all member disks of a shadowset to their default values after such changes have been done, one can use the command:
  - $ SET DEVICE /READ_COST = 1 DSAnnn:

Shadowing Between Sites: Local vs. Remote Reads on Fibre Channel

- With an inter-site Fibre Channel link, Shadowing can't tell which Fibre Channel disks are local and which are remote, so we have to give it some help:
  - To indicate which site a given disk is at, do:

    $ SET DEVICE/SITE=xxx 1DGAnnn:
  - To indicate which site a given OpenVMS node is at, do:

    $ SET DEVICE/SITE=xxx DSAnn:

    for each virtual unit, from each OpenVMS node, specifying the appropriate SITE value for each OpenVMS node.

Shadowing Between Sites: Local vs. Remote Reads on Fibre Channel

- Here's an example of setting /SITE values:

```
$ DEFINE/SYSTEM/EXEC ZKO 1
$ DEFINE/SYSTEM/EXEC LKG 2
$! Please note for this example that:
$! $1$DGA4: is physically located at site ZKO.
$! $1$DGA2: is physically located at site LKG.
$! The MOUNT command is the same at both sites:
$ MOUNT/SYSTEM DSA42/SHAD=($1$DGA4, $1$DGA2) VOLUMELABEL

$! At the ZKO site ...
$ SET DEVICE/SITE=ZKO DSA42:

$! At the LKG site ...
$ SET DEVICE/SITE=LKG DSA42:

$! At both sites, the following commands would be used
$! To specify at which site the disks are located
$ SET DEVICE/SITE=ZKO $1$DGA4:
$ SET DEVICE/SITE=LKG $1$DGA2:
```

Shadowing Between Sites: The $SHOW SHADOW Command

- Here's an example of output from the SHOW SHADOW command:

```
$ show shadow dsa100:

_DSA100:  Volume Label: TEST
  Virtual Unit State:   Steady State
  No Enhanced Shadowing Features in use

  VU Timeout Value      3600    VU Site Value           0
  Copy/Merge Priority   5000    Mini Merge      Disabled
  Served Path Delay       30

  Device $1$LDA1                Master Member
    Read Cost                2    Site 0
    Member Timeout         120

  Device $1$LDA2
    Read Cost                2    Site 0
    Member Timeout         120

  Device $1$LDA3
    Read Cost                2    Site 0
    Member Timeout         120
```

Shadowing Between Sites in Multi-Site Clusters

- Because Direct operations are lower in latency than MSCP-served operations, even when the inter-site distance is small:

It is generally best to have an inter-site Fibre Channel link or SAN Extension, if possible.

- And because inter-site latency can be much greater than intra-site latency, due to the speed of light:

If the inter-site distance is large, it is best to direct Read operations to the local disks, not remote disks

- Write operations have to go to all disks in a shadowset, remote as well as local members

If the inter-site distance is small, performance may be best if reads are spread across all members at both sites

Shadowing Between Sites

- Directing Shadowing Read operations to local disks, in favor of remote disks:
  - OpenVMS 7.3 allowed you to tell OpenVMS at which site member disks are located:
    - $ SET DEVICE/SITE=x DSAnnn: !on each node
    - $ SET DEVICE/SITE=x 1DGAn: !on each device
    - SHADOW_SITE SYSGEN parameter added in 7.3-1 (but never used)
    - OpenVMS 7.3-2 added $SET SHADOW and $SHOW SHADOW commands, so you can now do $ SET SHADOW/SITE
    - SHADOW_SITE_ID SYSGEN parameter added in 7.3-2
      - Determines default; eliminates need to do $SET SHADOW/SITE
  - Setting bit 16 (%x10000) in SYSGEN parameter SHADOW_SYS_DISK was a much-earlier method

Speeding Shadow Copies in Multi-Site Clusters

- When using the BACKUP/PHYSICAL trick to speed shadow copies:
  - For even more speed-up, perform the BACKUP/PHYSICAL operation on a node on the target side
    - Because remote (MSCP-served or Fibre Channel) writes take a minimum of 2 round trips, whereas remote reads take a minimum of only 1 round trip

Speeding Shadow Copies in Multi-Site Clusters

- Doing shadow copy work from a node at target site, not source site, is also most efficient, for the same reason. It also uses less inter-site bandwidth.
  - MSCP-served copies send data in only one direction cross the inter-site link, regardless of where the shadow copy threads run. This uses only ½ of a full-duplex link.
  - If inter-site bandwidth is limited, and a Fibre Channel connection or SAN Extension is in place between sites, it may be helpful for some fraction of shadow copy threads to run in the opposite direction, to also utilize the inter-site link bandwidth in the reverse direction

Speeding Shadow Copies in Multi-Site Clusters

- To control which node does shadow copy:
  - 1) Set dynamic SYSGEN parameter SHADOW_MAX_COPY to a large positive value on target-site node(s)
  - 2) Set SHADOW_MAX_COPY to 0 on all other nodes
  - 3) Do $MOUNT to add member to shadowset; wait briefly
  - 4) Reset SHADOW_MAX_COPY parameter to original values on all nodes
- As of OpenVMS 8.3, priority for copy & merge operations may be set with new command:
  - $ SET SHADOW /PRIORITY=p DSAn:

Speeding Shadow Copies in Multi-Site Clusters

- Determining which node is performing a shadow copy:
  - $SHOW SHADOW /ACTIVE

```
$ show shadow/active
Device     Volume          Device
 Name       Label          Status
_DSA2:     COMMON          Copy Active (13%) on NODE1
```

  - Using SDA:
    - From each cluster node, do:
      1. SDA> SET PROCESS SHADOW_SERVER
      2. SDA> SHOW PROCESS/CHANNELS
      3. and look for Busy channel to disk of interest
    - Or look for node holding a lock in Exclusive mode on a resource name of the form $DSAnnnn$_COPIER

More performance tips

- $SET SHADOW /ENABLE=SPLIT_READ_LBNS device
  - Logically divides the shadowset up into equal-sized LBN ranges and always directs reads to a givenLBN range to the same member. This aids in read and read-ahead cache efficiency

Questions?

Speaker Contact Info:

- E-mail: Keith.Parris@hp.com or keithparris@yahoo.com

- Web: http://www2.openvms.org/kparris/

- $SET SHADOW /ENABLE=SPLIT_READ_LBNS device
  - Logically divides the shadowset up into equal-sized LBN ranges and always directs reads to a givenLBN range to the same member. This aids in read and read-ahead cache efficiency