OAK RIDGE National Laboratory

Advanced Lustre File Layouts

Rick Mohr

Senior HPC Systems Engineer

Oak Ridge National Laboratory

ORNL is managed by UT-Battelle LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

Overview

This tutorial will cover...

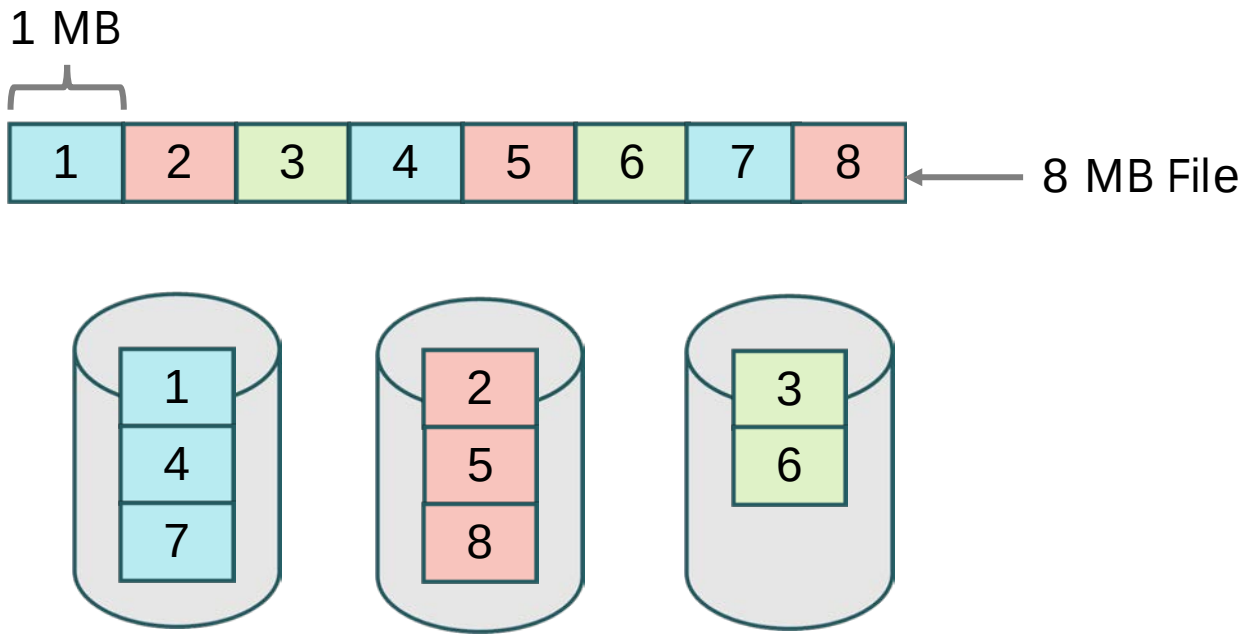
- Normal file layouts
- Progressive File Layouts (PFL)
- Data on MDT (DoM)
- File Level Redundancy (FLR)
- Self-Extending Layouts (SEL)

... but it will NOT cover

- Directory striping
- Choosing file layouts to optimize I/O performance

Normal File Layouts

- File is split into chunks and distributed across selected Object Storage Targets (OSTs) in round-robin fashion
- Splitting is controlled by two main parameters
  - Stripe count
  - Stripe size



Example:

stripe_count = 3

stripe_size = 1 MB

Setting File Stripe Parameters

- File striping is controlled with the `lfs setstripe` command

```
lfs setstripe –c 3 –S 1M data
```

- File must not exist before running `lfs setstripe`

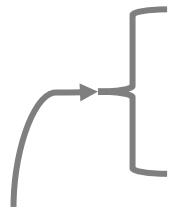
```
lfs setstripe: setstripe error for 'data': stripe already set
```

- Other striping options
  - `-i|--index` = Choose starting OST index for striping (default = -1)
  - `-o|--ost` = Specify OST indices to use
  - `-p|--pool` = Choose OSTs from the specified OST pool (default = none)

Viewing File Stripe Parameters

- Use `lfs getstripe` to view the layout for a file

```
[tmp]# lfs getstripe data
data
lmm_stripe_count:   3
lmm_stripe_size:    1048576
lmm_pattern:        raid0
lmm_layout_gen:     0
lmm_stripe_offset: 2
          obdidx           objid           objid           group
               2           29564         0x737c       0xb00000417
               7           29644         0x73cc       0xc00000403
               6           29662         0x73de       0xe40000414
```

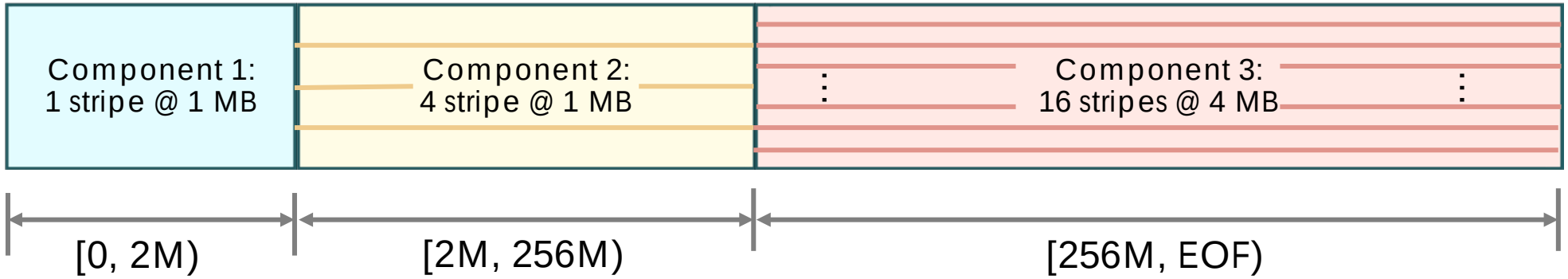


Allocated OSTs

Progressive File Layouts (PFL)

Progressive File Layouts (PFL)

- PFL allows more control over a file's layout
  - Normal layout applies single stripe count and stripe size to entire file
  - PFL can use different normal layouts on different parts of the file (making it one of the types of composite layouts)
  - Potentially optimize IO for files with non-uniform data structures



Creating PFL Layouts

- The `-E|--component-end` option is used to denote different components in the layout
- General format:

```
lfs setstripe -E end1 <stripe_opts> -E end2 <stripe_opts> … <file>
```

- Example:

```
lfs setstripe  -E 2M -c 1  -S 1M \
               -E 256M -c 4  -S 1M \
               -E -1 -c 16  -S 4M \
               data.txt
```

Can also use `-E eof`

Creating PFL Layouts (cont.)

- Components must be specified in order

  First component starts at offset = 0, next component starts where previous component ends

  ```
  lfs setstripe -E eof -c 16 -E 2M -c 1 data.txt
  ```

- First component inherits default values from parent/root directory. Later components inherit values from previous component.

  ```
  lfs setstripe -E 2M -c 1 -S 1M -E 64M -c 4 -E eof –S 4M
  ```

  stripe count = 1
  stripe size = 1M

  stripe count = 4
  stripe size = 1M

  stripe count = 4
  stripe size = 4M

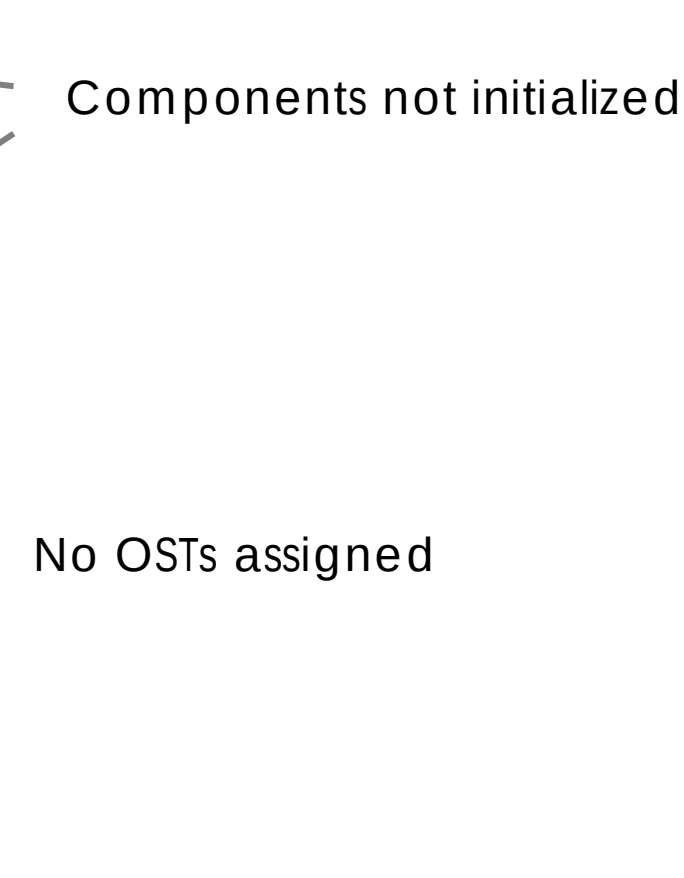
Lazy Initialization

- Lustre always allocates OSTs for the first component, but only allocates OSTs for other components when needed.

```
#> lfs getstripe data.txt
data.txt
  lcm_layout_gen:    3
  lcm_mirror_count:  1
  lcm_entry_count:   3
    lcme_id:             1
    lcme_mirror_id:      0
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   2097152
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 6
      lmm_objects:
      - 0: { l_ost_idx: 6, l_fid: [0xe40000414:0x7489:0x0] }
```

Lazy Initialization (cont.)

```
lcme_id:                2
lcme_mirror_id:         0
lcme_flags:             0
lcme_extent.e_start:    2097152
lcme_extent.e_end:      268435456
  lmm_stripe_count:     4
  lmm_stripe_size:      1048576
  lmm_pattern:          raid0
  lmm_layout_gen:       0
  lmm_stripe_offset:    -1

lcme_id:                3
lcme_mirror_id:         0
lcme_flags:             0
lcme_extent.e_start:    268435456
lcme_extent.e_end:      EOF
  lmm_stripe_count:     16
  lmm_stripe_size:      4194304
  lmm_pattern:          raid0
  lmm_layout_gen:       0
  lmm_stripe_offset:    -1
```



Lazy Initialization (cont.)

- Writing data will cause components to be initialized on-the-fly

```
# dd if=/dev/zero of=data.txt bs=1M count=4
```

- After 2M is written, Lustre initializes second component

```
lcme_id:              2
lcme_mirror_id:       0
lcme_flags:           init
lcme_extent.e_start:  2097152
lcme_extent.e_end:    268435456
  lmm_stripe_count:   4
  lmm_stripe_size:    1048576
  lmm_pattern:        raid0
  lmm_layout_gen:     0
  lmm_stripe_offset:  7
  lmm_objects:
  - 0: { l_ost_idx: 7, l_fid: [0xc00000403:0x746e:0x0] }
  - 1: { l_ost_idx: 8, l_fid: [0xc4000040e:0x7457:0x0] }
  - 2: { l_ost_idx: 12, l_fid: [0xe80000419:0x2a75:0x0] }
  - 3: { l_ost_idx: 1, l_fid: [0xac000040e:0x74c1:0x0] }
```

Dynamic Layout Changes

- With lazy initialization, only the first component is required to be specified when the file layout is set
- Other components can be added (and even deleted) dynamically using `lfs setstripe`
- There are some caveats:
  - Components can only be deleted starting from the last one
  - Deleting a component will cause all data in that component to be lost
  - Cannot write past the end of the last component

Dynamic Layout Example

```
[tmp]# lfs setstripe -E 2M -c 1 data.txt

[tmp]# dd if=/dev/zero of=data.txt bs=1M count=1
1+0 records in
1+0 records out
1048576 bytes (1.0 MB, 1.0 MiB) copied, 0.00139419 s, 752 MB/s

[tmp]# lfs setstripe --component-add -E 10M -c 4 data.txt

[tmp]# lfs setstripe --component-add -E -1 -c 8 data.txt

[tmp]# dd if=/dev/zero of=data.txt bs=1M count=20
20+0 records in
20+0 records out
20971520 bytes (21 MB, 20 MiB) copied, 0.0671822 s, 312 MB/s

[tmp]# ls -lh data.txt
-rw-r--r-- 1 root root 20M May  2 01:40 data.txt
```

Create initial component...

...then add two more

Data extends Into the third component

Dynamic Layout Example (cont.)

```
[tmp]# lfs setstripe --component-del -I 3 data.txt
```

component index*

Deleting last component...

```
[tmp]# ls -lh data.txt
-rw-r--r-- 1 root root 10M May  2 01:40 data.txt
```

...truncates the file and destroys data in the deleted component

```
[tmp]# dd if=/dev/zero of=data.txt bs=1M count=20
dd: error writing 'data.txt': No data available
11+0 records in
10+0 records out
10485760 bytes (10 MB, 10 MiB) copied, 0.00900609 s, 1.2 GB/s
```

Can't write past end of the last component

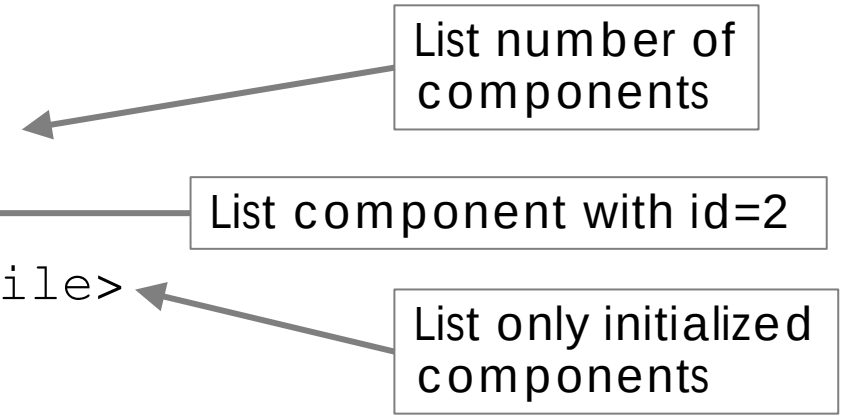
*Not necessarily sequential. Check `lcme_id` field in output from `lfs getstripe`

Other Useful Commands

- Component-related options for `lfs getstripe`

  (Check man page for full set of options)

```
lfs getstripe --component-count <file>
lfs getstripe –I2 <file>
lfs getstripe --component-flag=init <file>
```



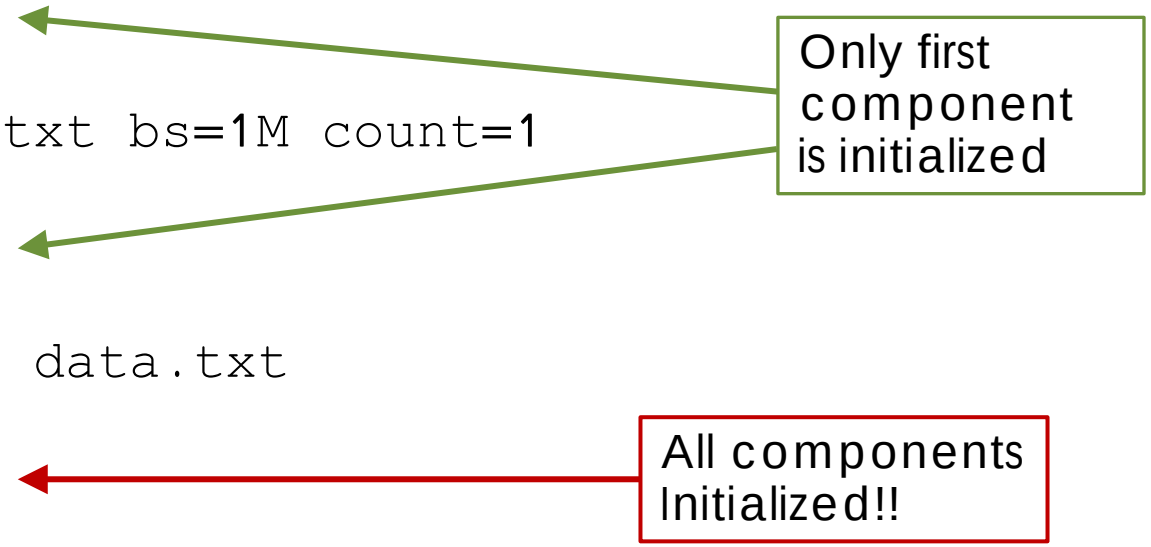
- Use the lfs migrate command to change a normal layout to PFL (and vice versa)

```
lfs setstripe -c 1 data.txt
dd if=/dev/zero of=data.txt bs=1M count=100
lfs migrate -E 2M -c 1 -E 20M -c 4 -E -1 -c 16 data.txt
```

Important Note About File Appends

- Components are normally only initialized when data is written to them…except in the case of file appends

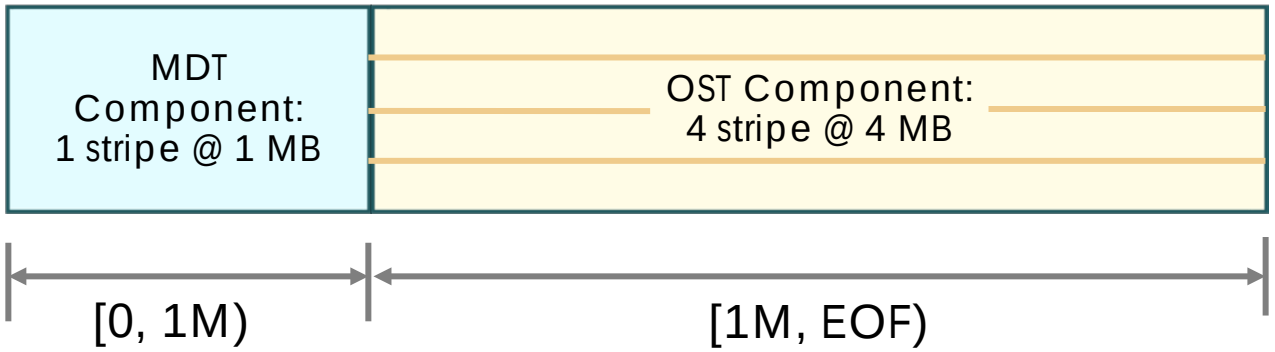
```
[tmp]# lfs setstripe -E 100M -c 1 -E 10G -c 4 -E -1 -c -1 data.txt

[tmp]# lfs getstripe data.txt

[tmp]# dd if=/dev/zero of=file.txt bs=1M count=1

[tmp]# lfs getstripe data.txt

[tmp]# echo ”This is a test” >> data.txt

[tmp]# lfs getstripe data.txt
```



Data on MDT (DoM)

DoM Basics

- DoM layouts are intended to improve small file I/O performance by placing all (or part) of the file on the MDT
- A DoM layout is a composite layout (and is actually just a special case of PFL)
  - Only the first component resides on the MDT
  - The first component always has stripe_count = 1
  - The MDT used for the first component is the same MDT that stores the inode for the file



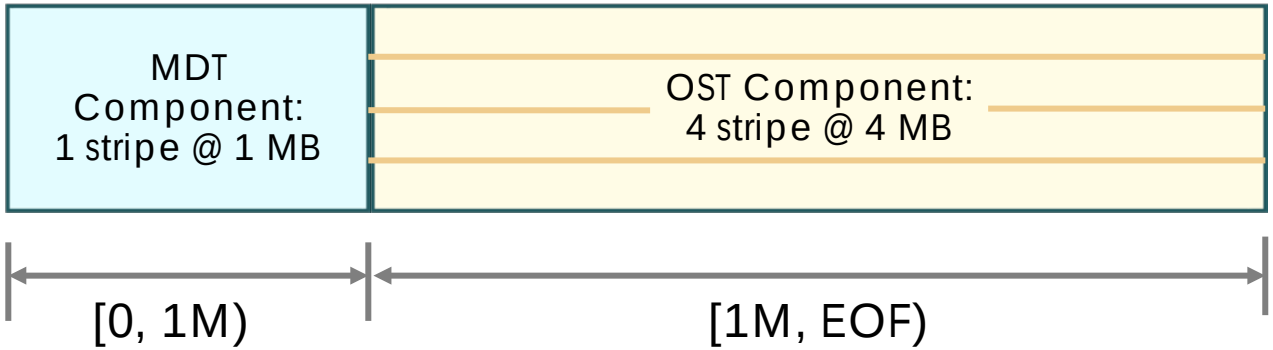
Creating DoM Layout

- DoM layout are created in a similar fashion to PFL layouts

```
lfs setstripe -E <end1> -L mdt -E <end2> [stripe_opts]…
```

- Example:

```
lfs setstripe -E 1M -L mdt -E eof -c 4 -S 4M dom_file
```



Displaying DoM Layout

```
[tmp]# lfs getstripe dom_file

dom_file
  lcm_layout_gen:    2
  lcm_mirror_count:  1
  lcm_entry_count:   2
    lcme_id:                     1
    lcme_mirror_id:              0
    lcme_flags:                  init
    lcme_extent.e_start:         0
    lcme_extent.e_end:           1048576
      lmm_stripe_count:  0
      lmm_stripe_size:   1048576
      lmm_pattern:       mdt
      lmm_layout_gen:    0
      lmm_stripe_offset: 0
```

DoM layout looks nearly identical to a PFL layout

stripe size = extent end

```
lcme_id:                     2
lcme_mirror_id:              0
lcme_flags:                  0
lcme_extent.e_start:         1048576
lcme_extent.e_end:           EOF
  lmm_stripe_count:  4
  lmm_stripe_size:   4194304
  lmm_pattern:       raid0
  lmm_layout_gen:    0
  lmm_stripe_offset: -1
```

Pattern is 'mdt' instead of 'raid0'

DoM for Sys Admins

- System administrators can control the maximum size of the DoM components
- On the MDS server, run

Temporary

```
lctl set_param -n lod.*MDT<index>*.dom_stripesize=<max>
```

or

Persistent

```
lctl set_param -P lod.*MDT<index>.lod.dom_stripesize=<max>
```

- For the `dom_stripesize` value:
  - Default value is 1 MB
  - No smaller than 64 KB and no larger than 1 GB
  - Must be 64 KB aligned

Dom for Sys Admins (cont.)

- DoM can be disabled by setting `dom_stripesize` to 0
  - This can be done on a per-MDT basis as well
- This will disable DoM component creation for any new files or layouts
  - Existing files with DoM components will remain unchanged
  - If a directory has a default layout defined that contains a DoM component, new files in that directory can still be created with DoM components
- DoM files can be identified using `lfs find`

```
lfs find <dir> -L mdt
```

File Creation with DoM Disabled

If DoM is disabled, attempting to create a DoM file will not fail. The layout just gets automatically altered.

```
[tmp]# lfs setstripe -E 1M -L mdt -E eof -c 2 dom_file

[tmp]# lfs getstripe dom_file
dom_file
  lcm_layout_gen:    1
  lcm_mirror_count:  1
  lcm_entry_count:   1
    lcme_id:             1
    lcme_mirror_id:      0
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  2
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 4
      lmm_objects:
      - 0: { l_ost_idx: 4, l_fid: [0xb8000041e:0x743c:0x0] }
      - 1: { l_ost_idx: 10, l_fid: [0xcc000041a:0x73be:0x0] }
```

File Level Redundancy (FLR)

File Level Redundancy (FLR)

- FLR allows users to define one or more mirrors for a file to provide extra data protection.
- When writing to a mirrored file, only one of the mirrors is updated. Other mirrors are marked as stale and need to be resynced.
  - This is the "FLR Delayed Write" implementation
- All mirrors (that are not stale) can be used for reading data
  - Can be used to improve read performance for files that are accessed by many processes in parallel

Creating FLR Layouts

- Unlike other file layouts, this one does not use `lfs setstripe`. Instead, there is a special `lfs mirror` command.

```
lfs mirror create -N[count] [stripe_opts] [--flags=<flags>] … <file>
```

- The mirrors can be either normal layouts, composite layouts, or a mixture of both

```
lfs mirror create -N2 --flags=prefer -c 2 -N -E 10M -c 1 -E eof -c 4 data.txt
```

Two mirrors have the same normal layout

Third mirror uses PFL layout

Displaying FLR Layouts

```
[tmp]# lfs getstripe data.txt
  lcm_mirror_count:  3
  lcm_entry_count:   4
    lcme_mirror_id:         1
    lcme_flags:             init,prefer
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  2
      lmm_stripe_size:   1048576
      lmm_objects:
      - 0: { l_ost_idx: 7, l_fid: [0xc00000403:0x7479:0x0] }
      - 1: { l_ost_idx: 6, l_fid: [0xe40000414:0x7493:0x0] }

    lcme_mirror_id:         2
    lcme_flags:             init,prefer
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  2
      lmm_stripe_size:   1048576
      lmm_objects:
      - 0: { l_ost_idx: 15, l_fid: [0xdc0000419:0x2acf:0x0] }
      - 1: { l_ost_idx: 9, l_fid: [0xc80000405:0x73e4:0x0] }
```

Number of mirrors

Note: Some fields omitted for brevity

Each mirror has unique ID

Displaying FLR Layouts (cont.)

```
lcme_mirror_id:      3
lcme_flags:          init
lcme_extent.e_start: 0
lcme_extent.e_end:   10485760
  lmm_stripe_count:  1
  lmm_stripe_size:   1048576
  lmm_objects:
  - 0: { l_ost_idx: 0, l_fid: [0xe00000416:0x73f9:0x0] }

lcme_mirror_id:      3
lcme_flags:          0
lcme_extent.e_start: 10485760
lcme_extent.e_end:   EOF
  lmm_stripe_count:  4
  lmm_stripe_size:   1048576
  lmm_stripe_offset: -1
```

Only first component of PFL is initialized

All components in the same mirror have identical `lcme_mirror_id` fields

Resync and Verify

- When writing data, the `lcme_flags` field for some components may change to indicate they are out of sync:

```
lcme_flags: init,prefer
```


```
lcme_flags: init,stale,prefer
```

- This is fixed by running

```
lfs mirror resync <file>
```

- Mirrored data can also be checked for consistency

```
lfs mirror verify <file>
```

Extending a Mirrored File

- Additional mirrors can be added to an existing file

```
lfs mirror extend –N[mirror_count] [stripe_options] … <file>
```

  - If the file is not a mirrored file already, it will be converted to one
  - Existing data is copied to the new mirror
- An existing file can also be added as a mirror to another file

```
lfs mirror extend [--no-verify] –N –f victim_file <file>
```

  - The user needs to ensure that victim_file contains the same data

Splitting a Mirrored File

- A specified mirror can be split from a mirrored file into its own separate file

```
lfs mirror split –mirror-id <ID> [-f <new_file>] <mirrored_file>
```

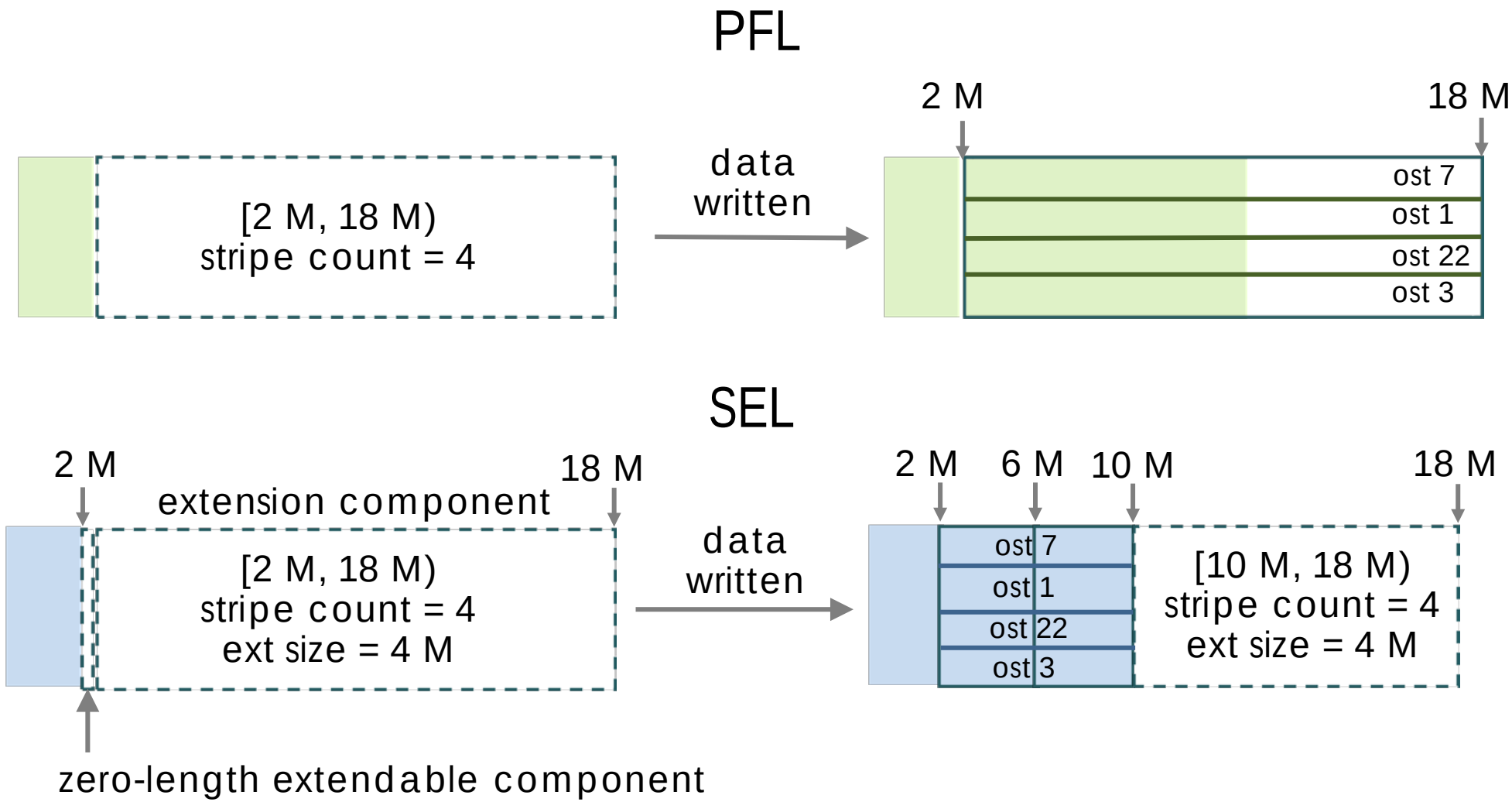
- If the `–f` option is not used, then the default name for the new file will be `<mirrored_file>.<mirror_id>`
- To destroy the mirror instead of splitting into its own file, just use the `–dI--destroy` option instead of `-f`

Self-Extending Layouts (SEL)

Self-Extending Layouts

- SEL is an extension of the PFL feature that allows the MDS to change the PFL layout dynamically if it detects OSTs running low on space
- PFL delays instantiation of some components
- SEL splits non-instantiated components into two parts
  1. An extendable component that is a regular PFL component covering a part of the region
  2. An extension component that is never instantiated and covers the remainder of the region

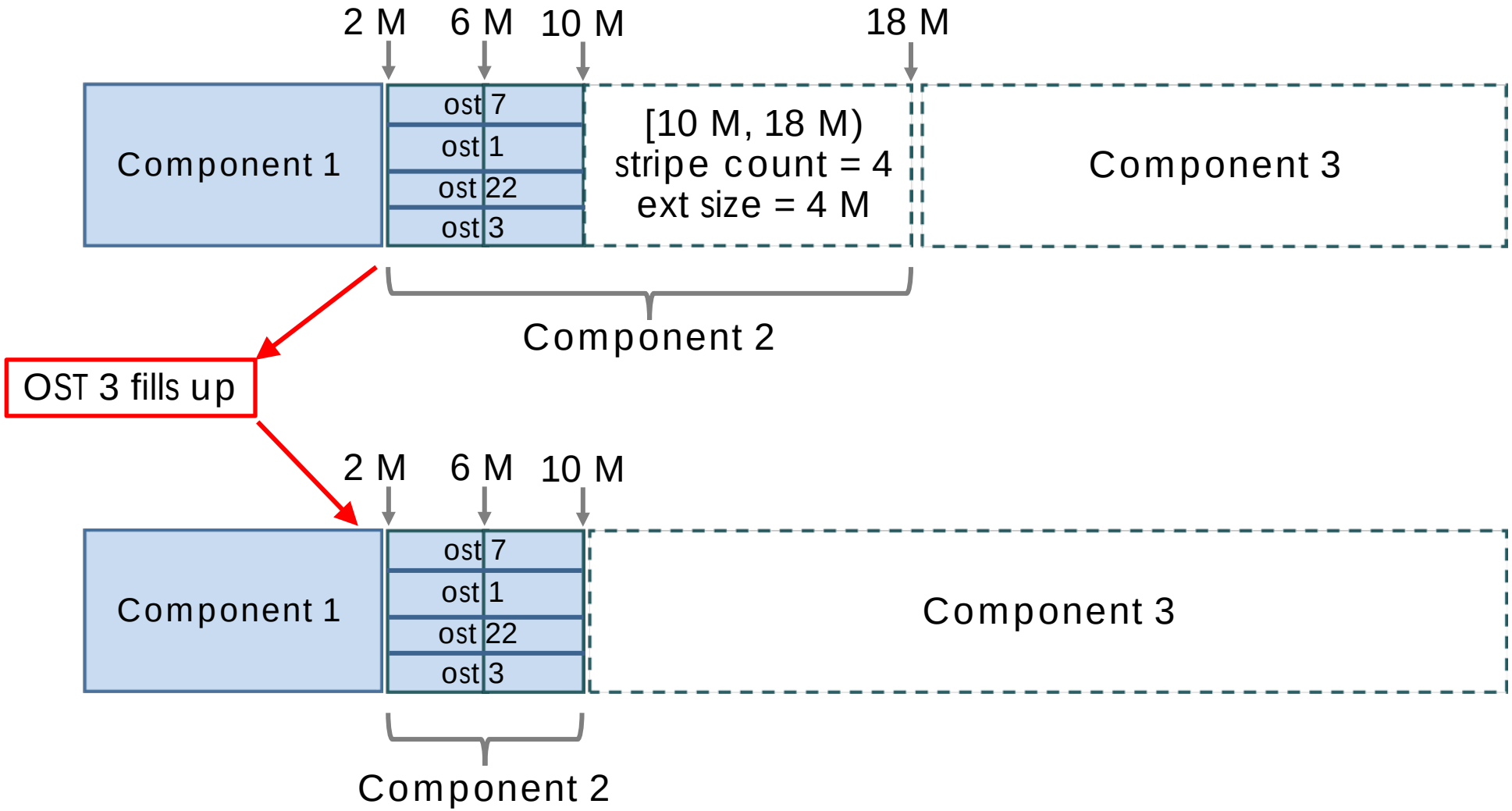
PFL Component vs. SEL Component



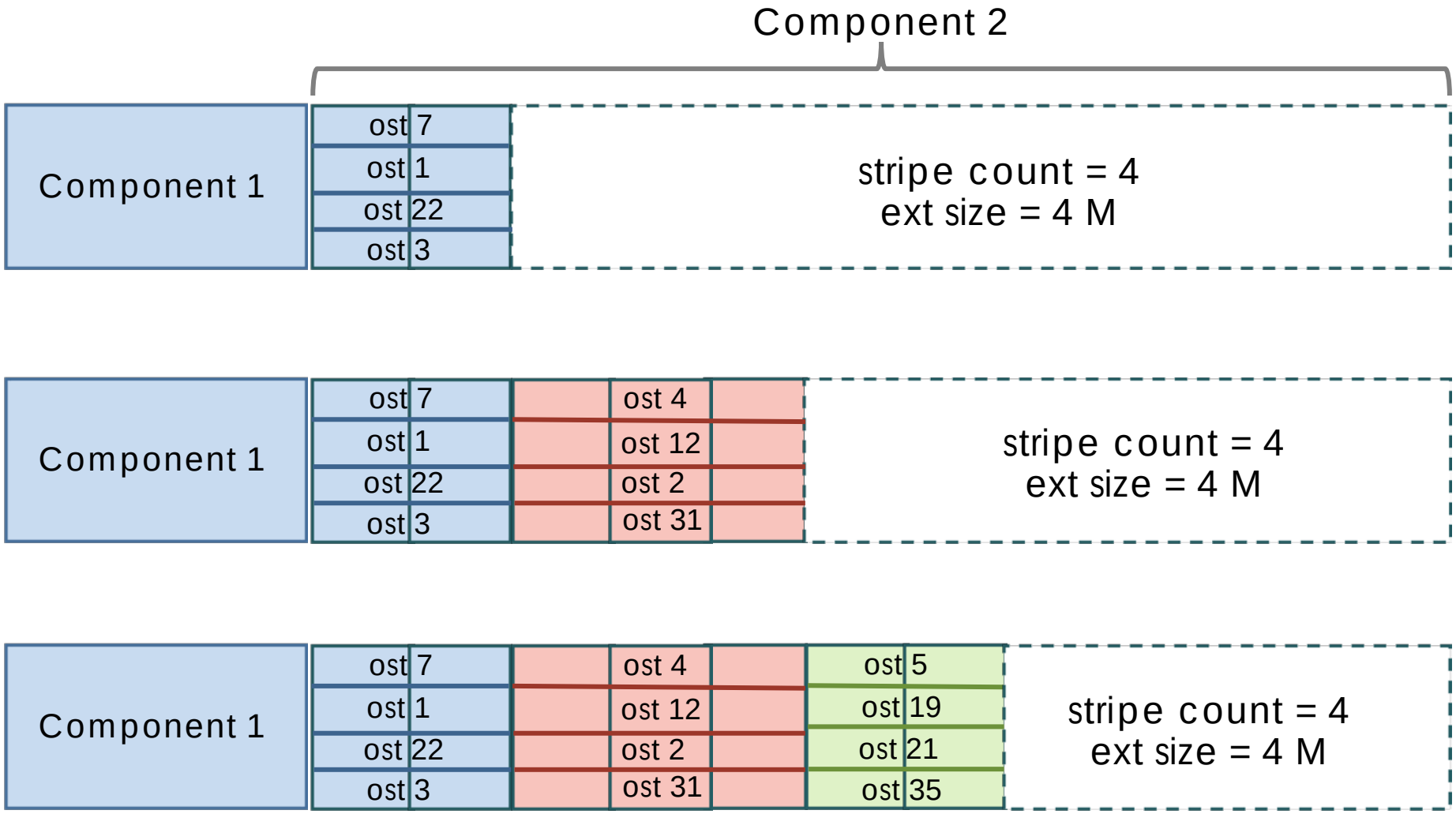
Extension Policies

- The benefit of SEL comes from what it does when an OST starts to fill up. There are four policies that handle various cases:
  1. Extension – When OSTs in current component are not low on space, continue using them (illustrated in the previous slide)
  2. Spill Over – If current component is not the last component, and one of the current OSTs is low on space, switch to next SEL component
  3. Repeating – If current component is the last component and one of the OSTs is low on space, create a new component with the same layout as the current component (but using different OSTs)
  4. Forced extension – If current component is the last component, and an attempt to repeat the layout fails due to low space, just keep using the current OSTs

Spill Over Policy



Repeating Policy



Creating a Self-Extending Layout

- Use the same command as for PFL...

```
lfs setstripe -E end1 <stripe_opts> -E end2 <stripe_opts> … <file>
```

- ...but add a `-z | --extension-size` option with the other stripe options

```
lfs setstripe -E 1G -z 64M -c 1 -E -1 -z 256M -c 4 data.txt
```

Viewing Self-Extending Layout

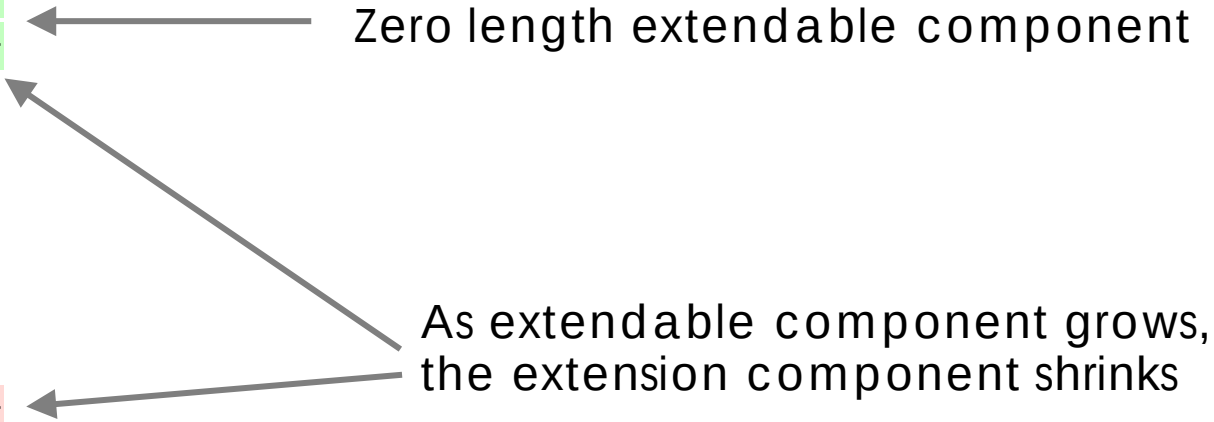
Note: Only some of the output fields are shown for brevity

```
[tmp]# lfs getstripe data.txt
data.txt
  lcm_entry_count:   4
    lcme_id:             1
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   67108864
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_stripe_offset: 10
      lmm_objects:
      - 0: { l_ost_idx: 10, l_fid: [0xcc000041a:0x73c2:0x0] }

    lcme_id:             2
    lcme_flags:          extension
    lcme_extent.e_start: 67108864
    lcme_extent.e_end:   1073741824
      lmm_stripe_count:  0
      lmm_extension_size: 67108864
      lmm_stripe_offset: -1
```

Denotes extension component

Viewing Self-Extending Layout (cont.)

```
lcme_id:              3
lcme_flags:           0
lcme_extent.e_start:  1073741824
lcme_extent.e_end:    1073741824
  lmm_stripe_count:   4
  lmm_stripe_size:    1048576
  lmm_stripe_offset:  -1

lcme_id:              4
lcme_flags:           extension
lcme_extent.e_start:  1073741824
lcme_extent.e_end:    EOF
  lmm_stripe_count:   0
  lmm_extension_size: 268435456
  lmm_stripe_offset:  -1
```



Acknowledgments

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Questions ?