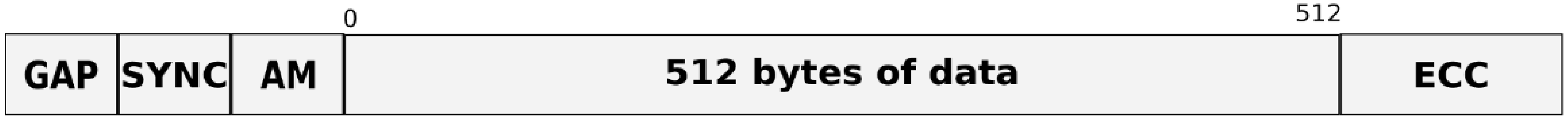
Aligning Stacked Devices

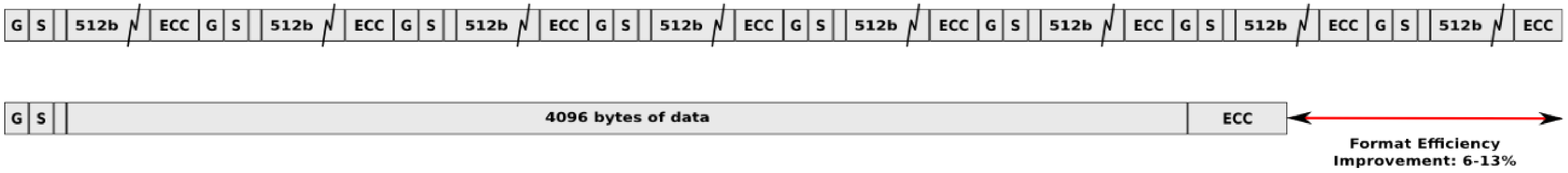
Mike Snitzer
Senior Software Engineer, Red Hat
June 11, 2010

I/O Limits – Quest for increased drive capacity

- Each sector on current 512 byte sector disks is quite a bit bigger than 512 bytes because of fields used internally by the drive firmware



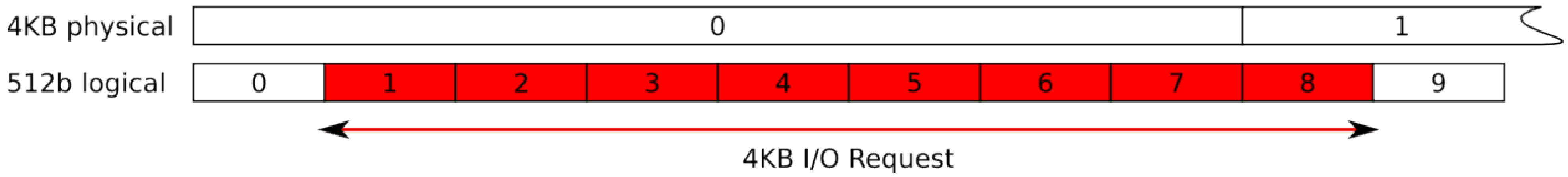
- The only way to increase capacity is to reduce overhead associated with each physical sector on disk



- Top: 8 x 512B sectors, each with overhead, needed to store 4KB of user data
- Bottom: 4KB sector drives can offer the same with much less overhead

I/O Limits – Transitioning to 4KB

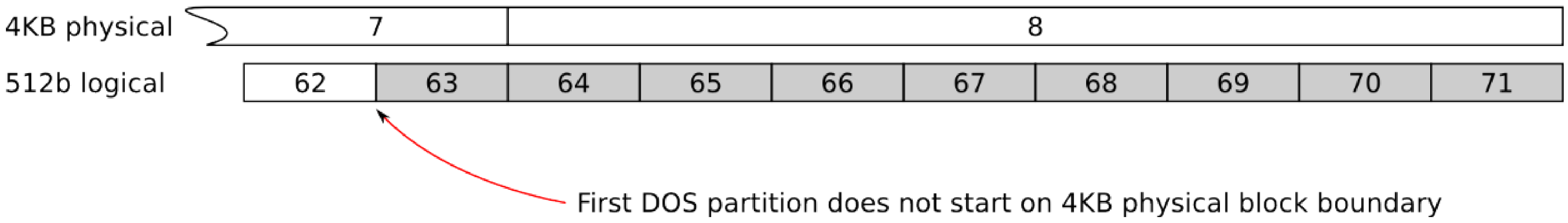
- 4K sector drives **may or may not** accept unaligned IO
- If they **do** accept unaligned IO there will be a performance penalty
  - Vendors will support a legacy OS with drives that have a 512B logical blocksize (external) and 4K physical blocksize (internal)
  - Misaligned requests will force drive to perform a read-modify-write



  - Vendors working on techniques to mitigate the R-M-W in firmware
    - R-M-W will cause a significant drop in performance: induces increased latency and lowers IOPS
    - There is quite a bit of inertia behind trying to preserve 512b sector support

I/O Limits – Alignment

- DOS partition tables default to putting the first partition on LBA 63


4KB physical
7
8
512b logical
62
63
64
65
66
67
68
69
70
71
First DOS partition does not start on 4KB physical block boundary

- Desktop-class 4KB drives can be formatted to compensate for DOS partitioning
  - sector 7 is the lowest aligned logical block, the 4KB sectors start at LBA -1, and consequently sector 63 is aligned on a 4KB boundary
  - Linux >= 2.6.31 allows partition tools, LVM2, etc to understand that this compensation is being used (*alignment_offset*=3584 bytes), from:

    /sys/block/$DEVICE/alignment_offset

I/O Limits – Performance I/O hints

- Linux also provides the ability to train upper storage layers based on hardware provided I/O hints
  - Preferred I/O granularity for random I/O
    - *minimum_io_size* - the smallest request the device can perform w/o incurring a hard error or a read-modify-write penalty (e.g. RAID chunk size)
  - Optimal sustained I/O size
    - *optimal_io_size* - the device's preferred unit of receiving I/O (e.g. RAID stripe width)
- Available through sysfs:

  /sys/block/$DEVICE/queue/minimum_io_size

  /sys/block/$DEVICE/queue/optimal_io_size

Stacking I/O Limits – Overview

- All layers of the Linux I/O stack have been engineered to propagate the various I/O Limits up the stack.
- When a layer consumes an attribute or aggregates many devices, it must expose appropriate I/O Limits so that upper-layer devices or tools will have an accurate view of the storage as it transformed.
- Examples:
    - Only one layer in the I/O stack should adjust for a non-zero *alignment_offset*
        - once a layer adjusts for it it will export a device with an *alignment_offset* of zero
    - A striped LVM logical volume must export a *minimum_io_size* and *optimal_io_size* that reflects chunk_size and stripe count

Stacking I/O Limits – LVM

- LVM2 >= 2.02.51 (2.02.62 saw last small related fix)
  - Added devices/data_alignment_detection to lvm.conf
  - Added devices/data_alignment_offset_detection to lvm.conf
  - Added --dataalignmentoffset to pvcreate to shift start of aligned data area.
- LVM will read I/O Limits to determine the optimal start of the data area (takes into account *alignment_offset, minimum_io_size* and *optimal_io_size*)
  - LVM defaults to creating a 64K aligned data area
  - But I/O Limits support allows for additional precision
  - DM uses LVM2 determined start when stacking limits

Stacking I/O Limits – Block layer and DM

- Block layer (Linux >= 2.6.31) has infrastructure to stack I/O limits
  - *blk_stack_limits(top, bottom, start)* verifies alignment and stacks {physical,logical}_block_size and {minimum,optimal}_io_size
  - *physical_block_size*, *logical_block_size* and *minimum_io_size* use max() when stacking top and bottom device limits
  - *optimal_io_size* uses lcm()
- DM now has infrastructure to detect if a combination of devices will lead to a misaligned DM device
  - Each DM target implements an *.iterate_devices* method that calls block layer's *blk_stack_limits* for each underlying device (during table load)
  - The final stacked limits get assigned to the DM device's queue when the DM device is resumed

Stacking I/O Limits – How it is made possible

- It all starts with the SCSI and ATA protocols
  - The standards have been extended to allow devices to provide alignment and I/O hints when queried
  - Not all vendors' hardware will “just work”
- Linux now retrieves the alignment and I/O hints that a device reports
- Linux presents I/O Limits through uniform sysfs attributes for all block devices. Ioctl interface is also available.
- DM, LVM2, cryptsetup have been updated to support I/O Limits
  - Also Ext[234], XFS, libblkid, parted, fdisk, anaconda, virtio
- See: http://people.redhat.com/msnitzer/docs/io-limits.txt
- Thanks to Martin K. Petersen

QUESTIONS?