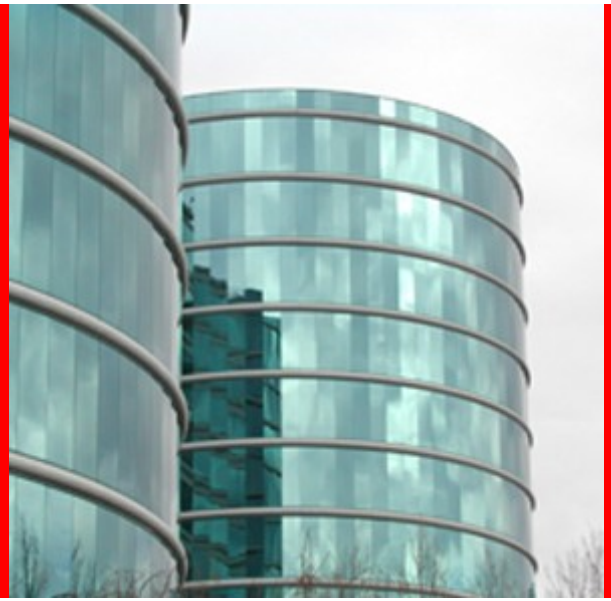

ORACLE®



ORACLE®

Linux and Advanced Storage Technologies

Martin K. Petersen <martin.petersen@oracle.com>
Consulting Software Developer, Linux Kernel Engineering

Blocks and Alignment

Blocks

- For decades we have had a common abstraction for block storage devices: A drive with 512b sectors
- From an addressing standpoint we have moved away from C/H/S to *logical block* addressing. The abstraction is now a linear address space from 0..*n* in units of 512b
- Disk drives have continued to use 512b as internal allocation unit aka *sector* aka *physical block size*
- However, many other storage devices ranging from USB sticks to enterprise arrays have been using internal blocks bigger than 512b for a long time

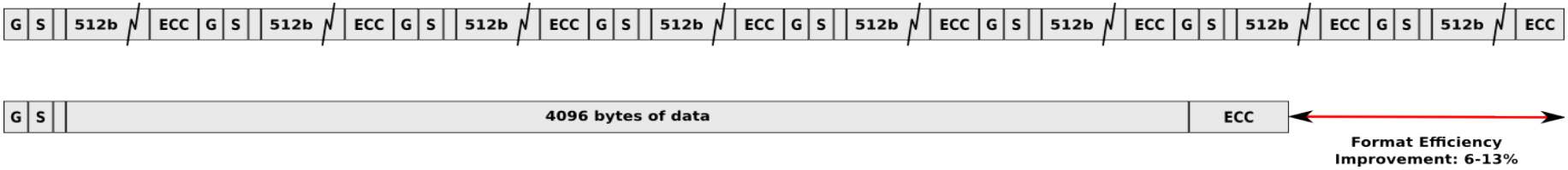
Blocks

- Because these devices did not disclose their physical block size we have occasionally ended up misaligning I/O requests
- Caches in RAID arrays have mitigated the penalty for submitting misaligned I/Os
- Extensions to ATA and SCSI protocols now allow storage devices to indicate their preferred block sizes, whether they contain spinning media, etc.
- SSDs and disk drives with physical blocks >512b exhibit significant performance penalties on misaligned I/Os

Disk Drives: 512-byte Physical Blocks

- Each sector on a disk is actually quite a bit bigger than 512 bytes thanks to fields used internally by the drive firmware
- These fields help to position the read/write head, help ensure the right location is found and contain an ECC that protects the data portion of the sector
- Together these fields eat up a lot of physical storage space and disk drive manufacturers are pretty close to the physical limits as far as track density goes
- This means the only way to increase capacity is to reduce overhead

Disk Drives: 4096-byte Physical Blocks

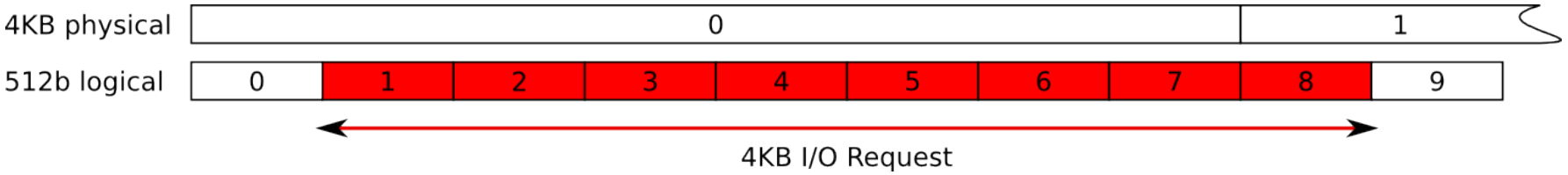


- The solution is to switch to 4096b physical blocks
- Despite potentially having multiple sync fields per blocks and a bigger ECC there is still a substantial capacity gain
- Most operating systems use 4096b pages and filesystem blocks so moving away from 512b units is not a big deal
- However, legacy operating systems are hardwired to 512b sectors and can not use drives which expose 4096b logical blocks

Disk Drives: Desktop vs. Enterprise

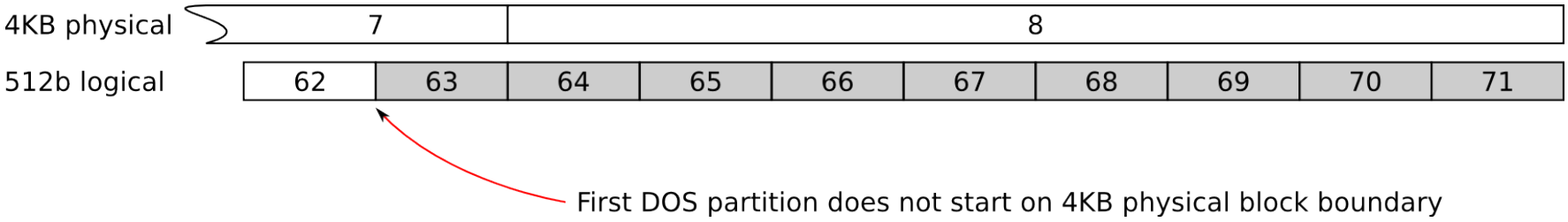
- Desktop drives
  - vendors will keep shipping ATA drives with 512b logical block addressing but which use 4096b physical blocks internally
  - drives with 4096b blocks may happen over time
- Server drives
  - three variants:
    - 512b/512b legacy
    - 512b/4096b emulation (nearline, SSD)
    - 4096b/4096b native (RAID array drives, SSD)
- 4096b logical block size needs work in BIOS/EFI/boot ROM space and progress has been slow

Alignment



- The desktop class drives are only *emulating* 512b sectors. If you submit a misaligned request, the drive will have to resort to read-modify-write
- This means the platter has to do an extra revolution, inducing latency and lowering IOPS
- Vendors are working on techniques to mitigate this in drive firmware. Without mitigation the drop in performance is quite significant

Alignment: DOS Partitions



- DOS put first partition on LBA 63 by default and now we're stuck with it
- Consequently, laptop/desktop drives may ship formatted so that LBA 63 is aligned on a 4096b physical boundary to ease the pain for XP users
- Only the first partition will be naturally aligned. And only if DOS partition tables are used
- Vista and Windows 7 will align first partition on a 1MB+ε boundary

Linux I/O Topology

- Linux gathers block sizes and alignment information and exports I/O topology in a generic fashion regardless of device type:
  - parted and fdisk make use of industry default 1MB alignment
  - RAID devices report stripe size and width
  - DM adjusts beginning of data in volumes
  - MD reports but does not currently adjust alignment
  - device stacking handled correctly
  - mkfs checks and warns about misalignment
- Linux 2.6.31+, Fedora/EL6 have the right bits

Discard

Discard: Solid State Drives

- Flash cells have a limited number of write cycles
- Write amplification due to erase block size further shortens a drive's life
- Several approaches are being used to remedy this:
  - Alignment
  - Over-provisioning. Drive has more physical storage capacity than is reported to the OS
  - Trim is used to mark regions that are no longer in use and which do not need wear leveling

Discard: Thin Provisioning

- Enterprise storage utilization is pretty low. I.e. only a fraction of the physical storage capacity is being used
  - Some space is lost due to parity and spares
  - Some applications require many IOPS, many spindles
  - Best practice is to make bigger LUNs “just in case”...
- The solution to this is thin provisioning, the opposite of the SSD approach. Array tells OS it has more storage capacity than it actually does
- Makes it easy for the applications/virtual hosts
- Storage admin gets an email when physical disk space is running low

Discard

- Solid state devices and thin provisioning arrays have something in common:
  - Both need a way to mark previously used space as unused
- Linux' discard functionality is an abstract way for filesystems to communicate that a block range is no longer needed
- At the bottom of the stack we translate the discard into the relevant ATA or SCSI commands
- However, things are not as simple as they seem...

Discard: 4 ways and counting...

- ATA DSM TRIM
  - No command queueing
  - Reasonably fast at clearing many ranges in one command
- SCSI WRITE SAME
  - Two variants
  - Essentially free on several arrays
  - Only one block range per command
- SCSI UNMAP
  - Many block ranges
  - Not supported by all vendors

Discard

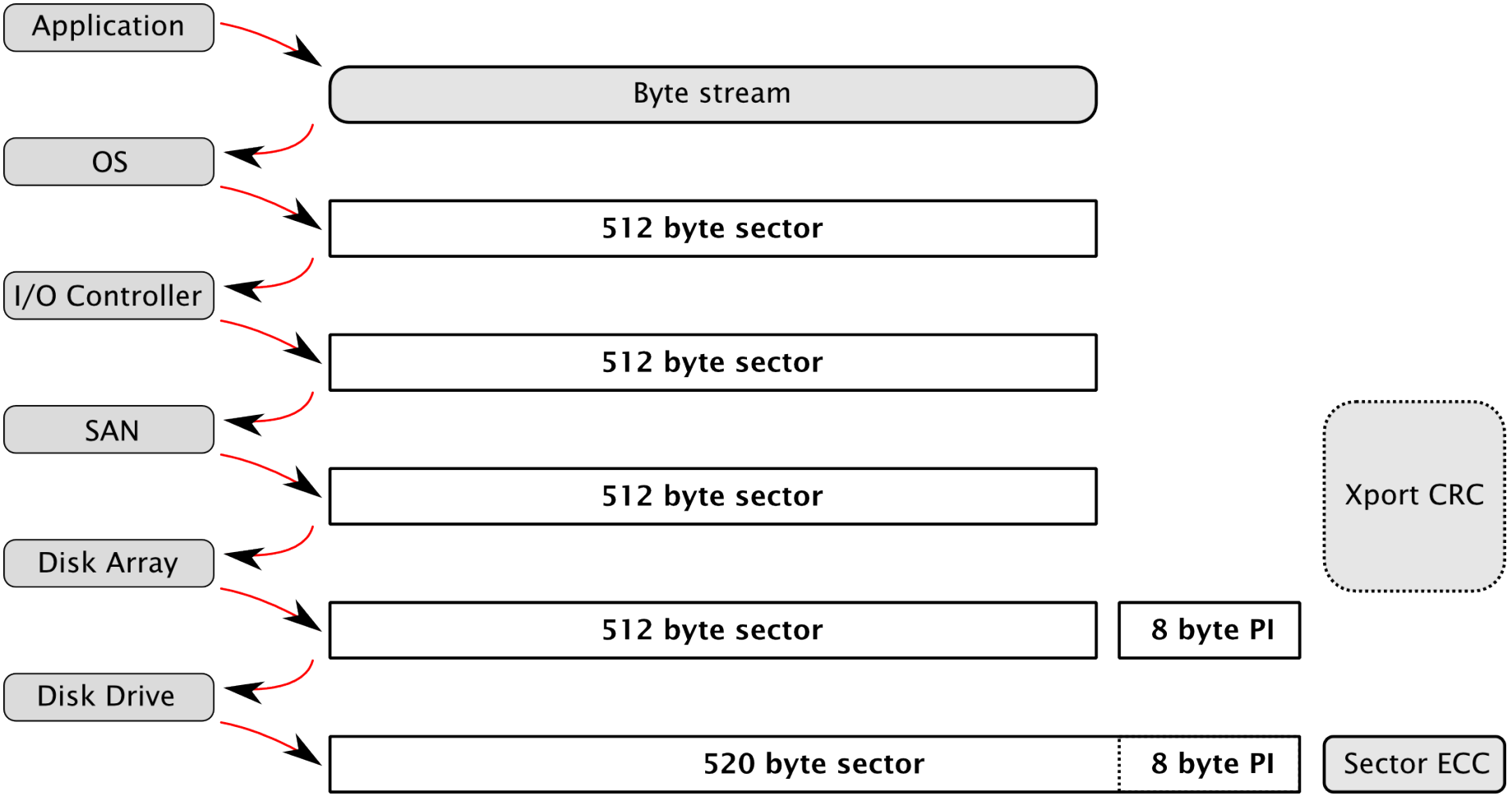
- One size does not fit all, and ATA and SCSI protocols are moving targets
- Variations in performance between devices are making it hard to optimize
- Three-pronged approach:
  - hdparm for direct device access
  - Command line-initiated scrub via filesystem ioctl
  - Realtime discard filesystem mount option
- Initial discard support went into 2.6.33
- Device Mapper support is done
- Discard coalescing for TRIM and UNMAP is WIP

Data Integrity

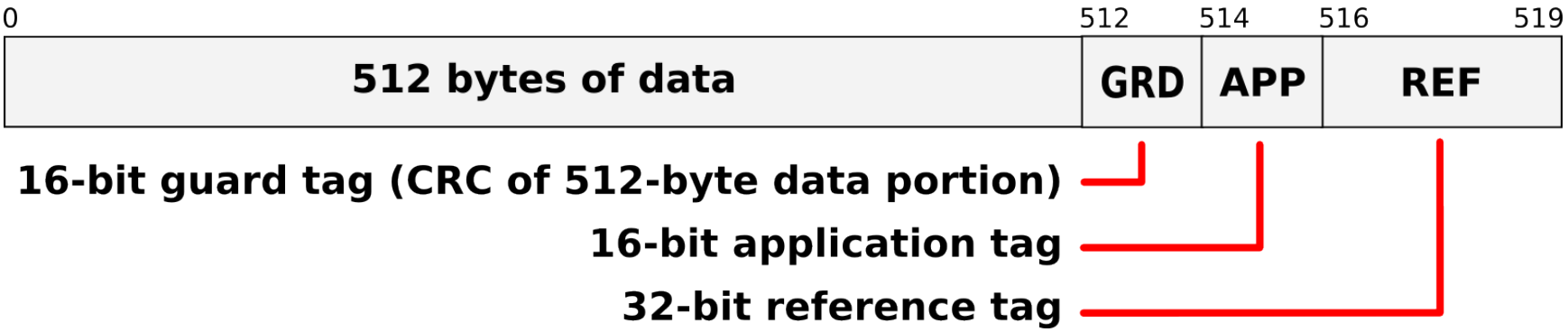
Data Integrity

- Tendency to focus on latent sector corruption inside disk drives: Media defects, head misses
  - btrfs block checksums enable corruption *detection* at READ time
  - however, it could take months before you find out and the original buffer is lost
- T10 DIF and DIX:
  - are about preventing *in-flight* corruption
  - tackle *content corruption* errors & *data misplacement* errors
  - allow us to detect problems when they happen, before the original buffer is erased from memory
  - and before bad data ends up being stored on disk

Data Integrity: Normal I/O Example

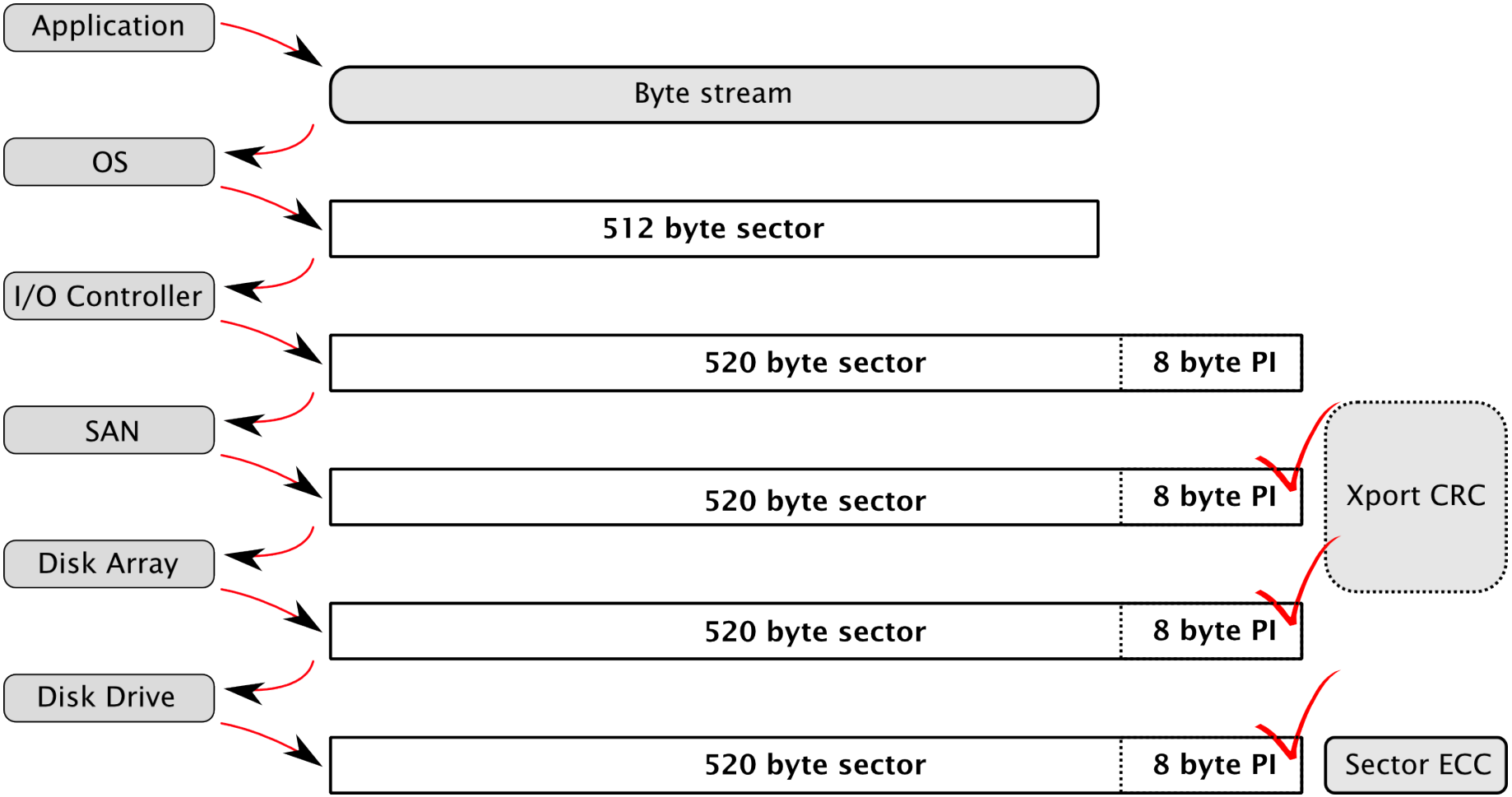


Data Integrity: T10 Data Integrity Field



- Standardizes those extra 8 bytes
- Prevents content corruption and misplacement errors
- Protects path between HBA and storage device
- Protection information is interleaved with data on the wire, i.e. effectively 520-byte logical blocks

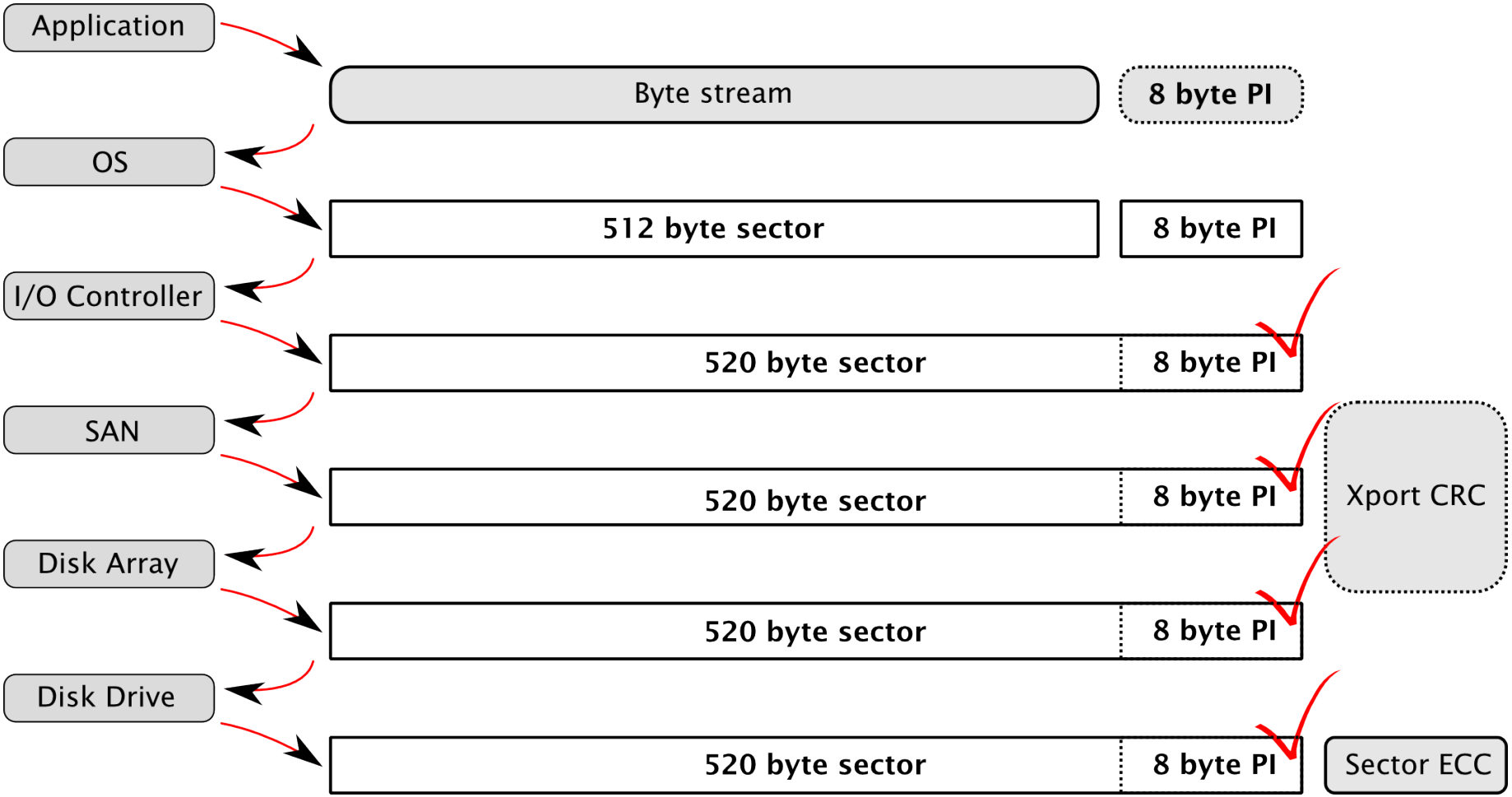
Data Integrity: T10 Data Integrity Field Example



Data Integrity Extensions

- We'd like to extend T10 DIF all the way up to the application, enabling true end-to-end data integrity protection
- The Data Integrity Extensions (DIX)
  - Enable DMA transfer of protection information to and from host memory
  - Separate data and protection information buffers to avoid inefficient 512+8+512+8+512+8 scatter-gather lists
  - Provide a set of commands that tell HBA how to handle the I/O:
    - *Generate, Strip, Pass, Verify, etc.*

Data Integrity Extensions + T10 DIF Example



Data Integrity

- Kernel support in 2.6.27
- Generic application API is work in progress in SNIA Data Integrity Technical Working Group

Conclusion

- The 512-byte sector monoculture is a thing of the past
- We are tracking and interacting with relevant storage standards bodies
- Other interesting technologies coming up in the solid state storage space

- Linux & Advanced Storage Interfaces

  http://oss.oracle.com/~mkp/

SOFTWARE. HARDWARE. COMPLETE.