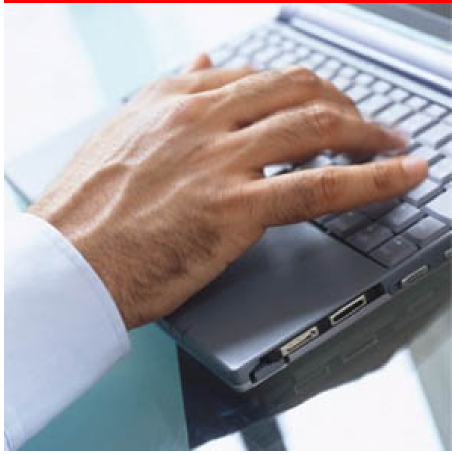
ORACLE®

Linux Data Integrity

Martin K. Petersen
Software Developer, Linux Engineering

Topics



- DIF/DIX
  - Data Corruption
  - T10 DIF
  - Data Integrity Extensions
- Linux Data Integrity
  - Filesystems
  - User Application Interfaces

DIF, DIX & Data Integrity

Data Corruption

- Tendency to focus on corruption inside disk drives
  - Media developing defects
  - Head misses
- However, corruption can - and often does - happen while data is in flight
  - Modern transports like FC and SAS have CRC on the wire
  - Which leaves library / kernel / firmware errors
  - Examples: Bad buffer pointers, missing or misdirected writes
- Industry demand for end-to-end protection
  - Oracle HARD is widely deployed
  - Other databases and mission-critical business apps
  - Nearline/archival storage wants belt and suspenders

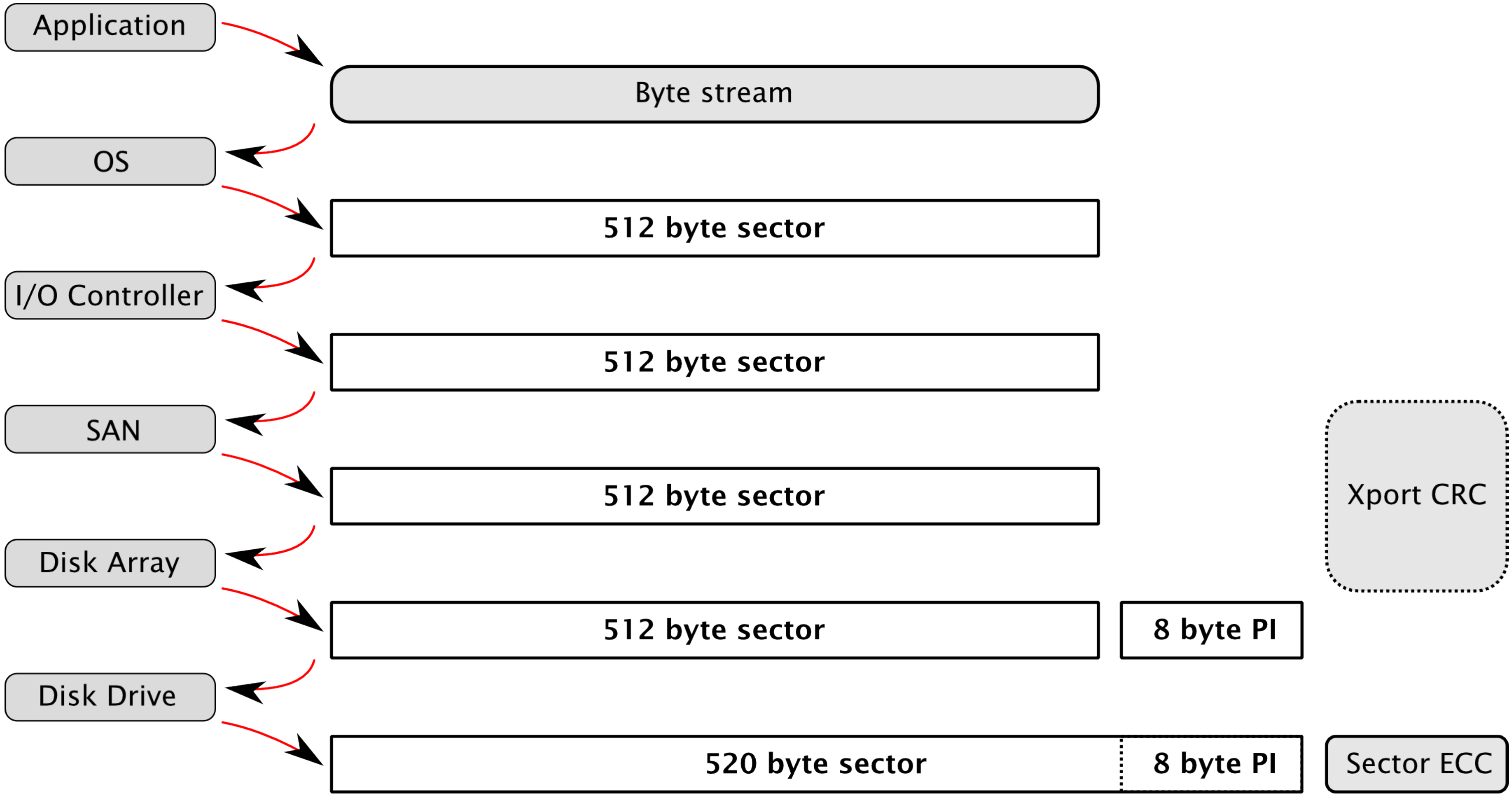
Data Corruption

- DIF/DIX are *orthogonal* to logical block checksums
  - We still love you, btrfs!
  - Logical block checksum errors are used for *detection* of corrupted data
  - Detection happens at READ time
  - ... which could be months later, original buffer is lost
  - Any redundant copies may also be bad if original buffer was garbled
- DIF/DIX are about *proactively preventing* corruption
  - Preventing bad data from being stored on disk in the first place
  - ... and finding out about problems before the original buffer is erased from memory

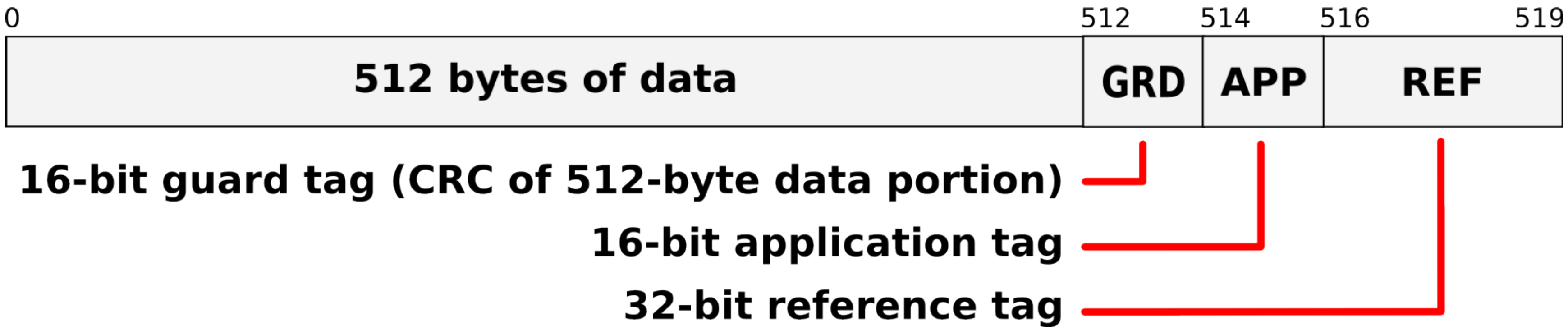
Disk Drives

- Most disk drives use 512-byte sectors
- A sector is the smallest atomic unit the drive can access
- Each sector is protected by a proprietary ECC internal to the drive firmware
- 4096-byte sectors are coming
- Enterprise drives (Parallel SCSI/SAS/FC) support 520/528 byte “fat” sectors
- Sector sizes that are not a multiple of 512 bytes have seen limited use because operating systems deal with everything in units of 512, 1024, 2048 or 4096 bytes
- RAID arrays make extensive use of fat sectors

Normal I/O

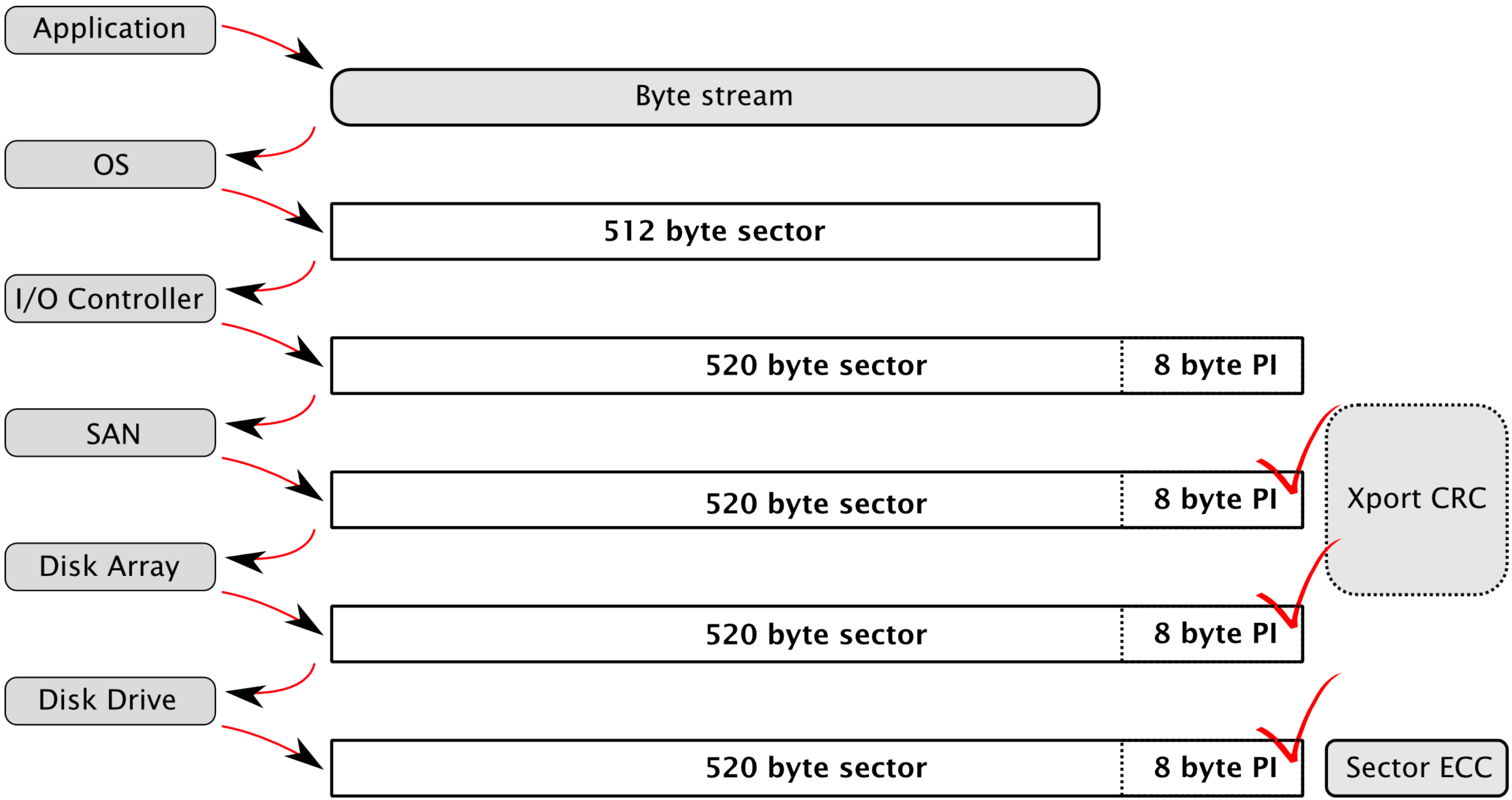

Application
Byte stream
OS
512 byte sector
I/O Controller
512 byte sector
SAN
512 byte sector
Xport CRC
Disk Array
512 byte sector
8 byte PI
Disk Drive
520 byte sector
8 byte PI
Sector ECC

T10 Data Integrity Field



- Prevents data corruption and misplacement errors
- Only protects path between HBA and storage device
- Protection information is interleaved with data on the wire, effectively 520-byte sectors
- SATA T13/External Path Protection proposal uses same PI format

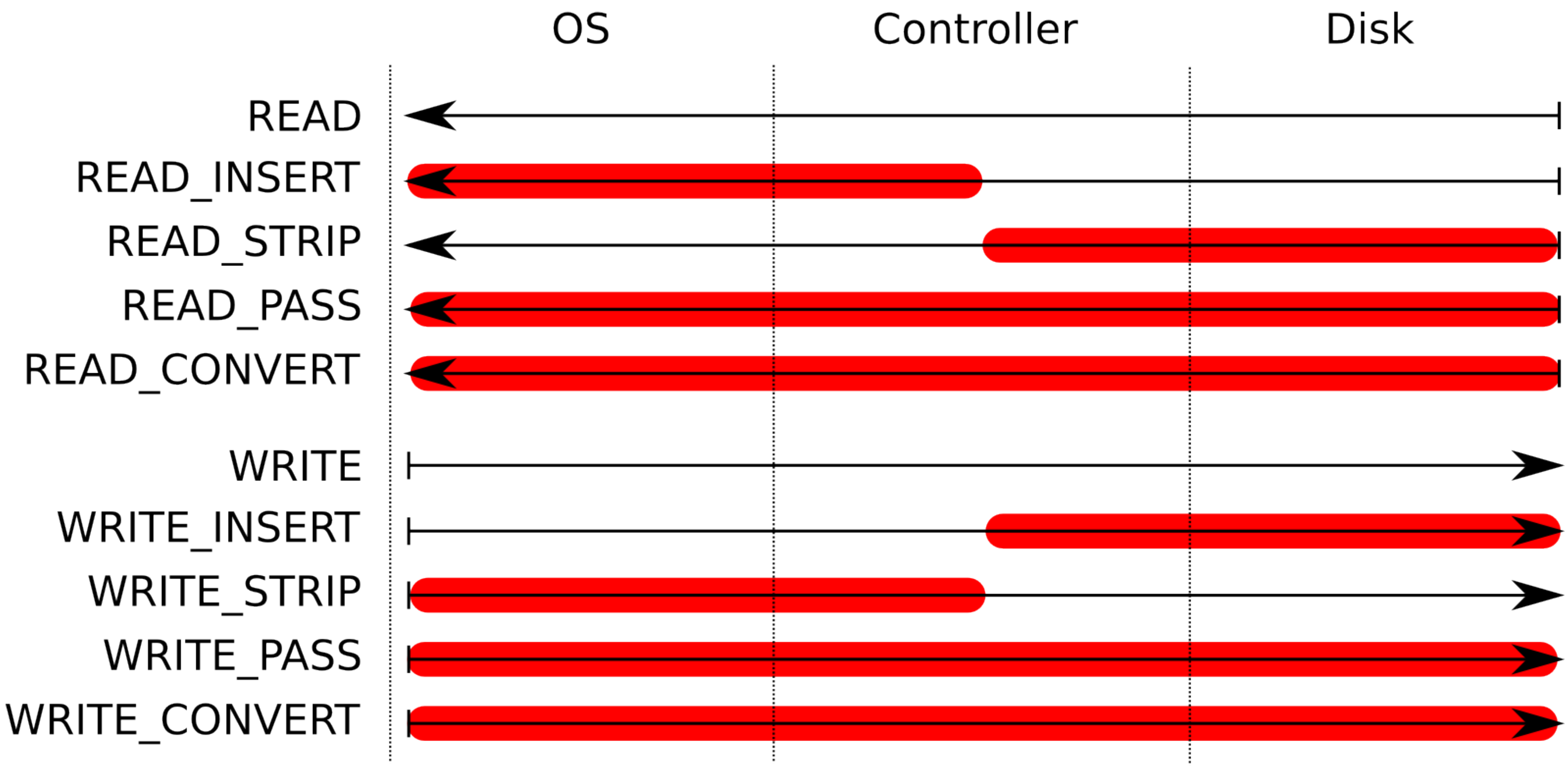
T10 Data Integrity Field I/O



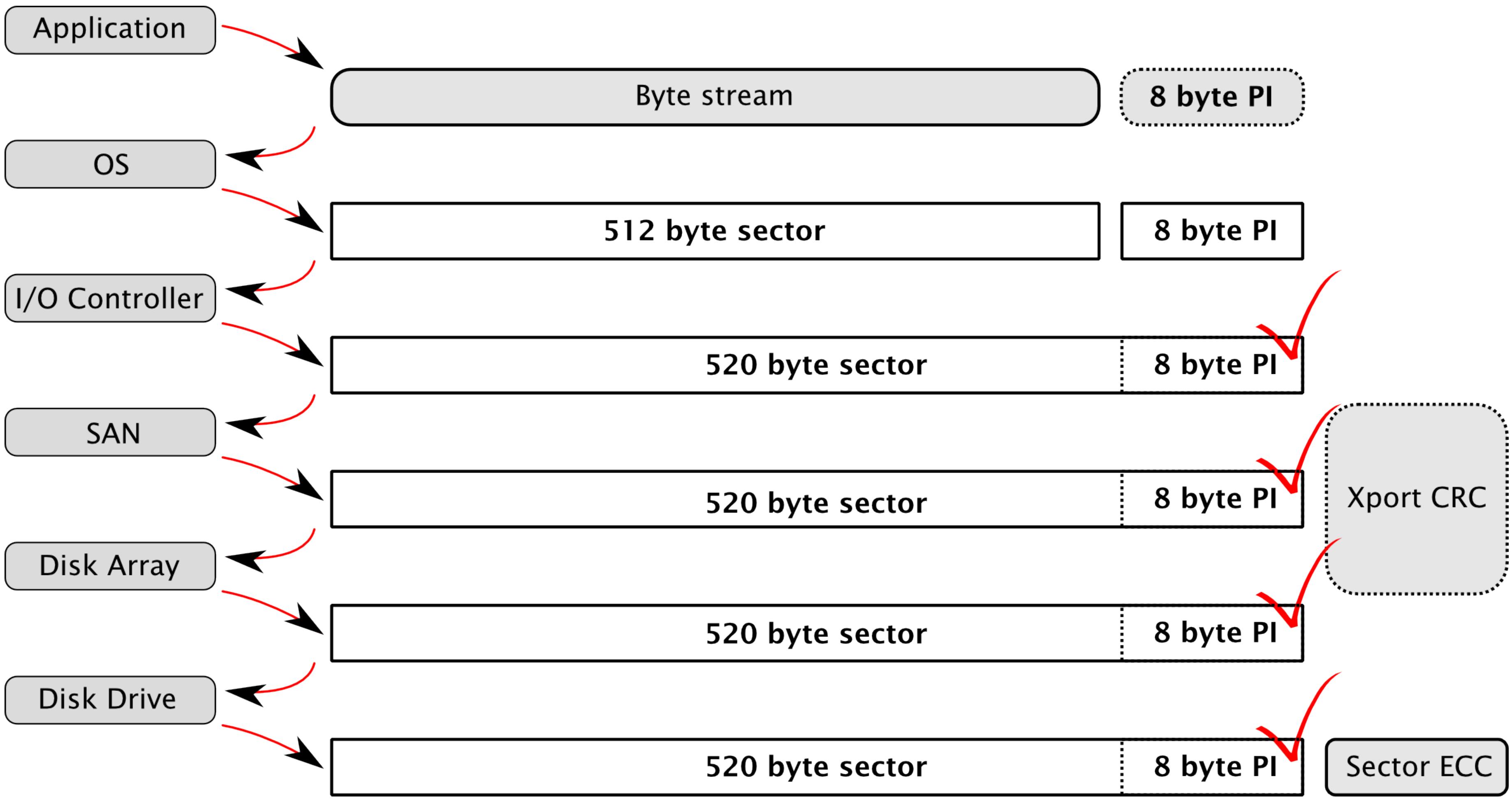
Data Integrity Extensions

- Attempt to extend T10 DIF all the way up to the application, enabling true end-to-end data integrity protection
- Essentially a set of meta commands for SCSI/SAS/FC controllers
- The Data Integrity Extensions:
  - 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 scatterlists
  - Provide a set of commands that tell HBA how to handle I/O:
    - *Generate, strip, pass, convert and verify*

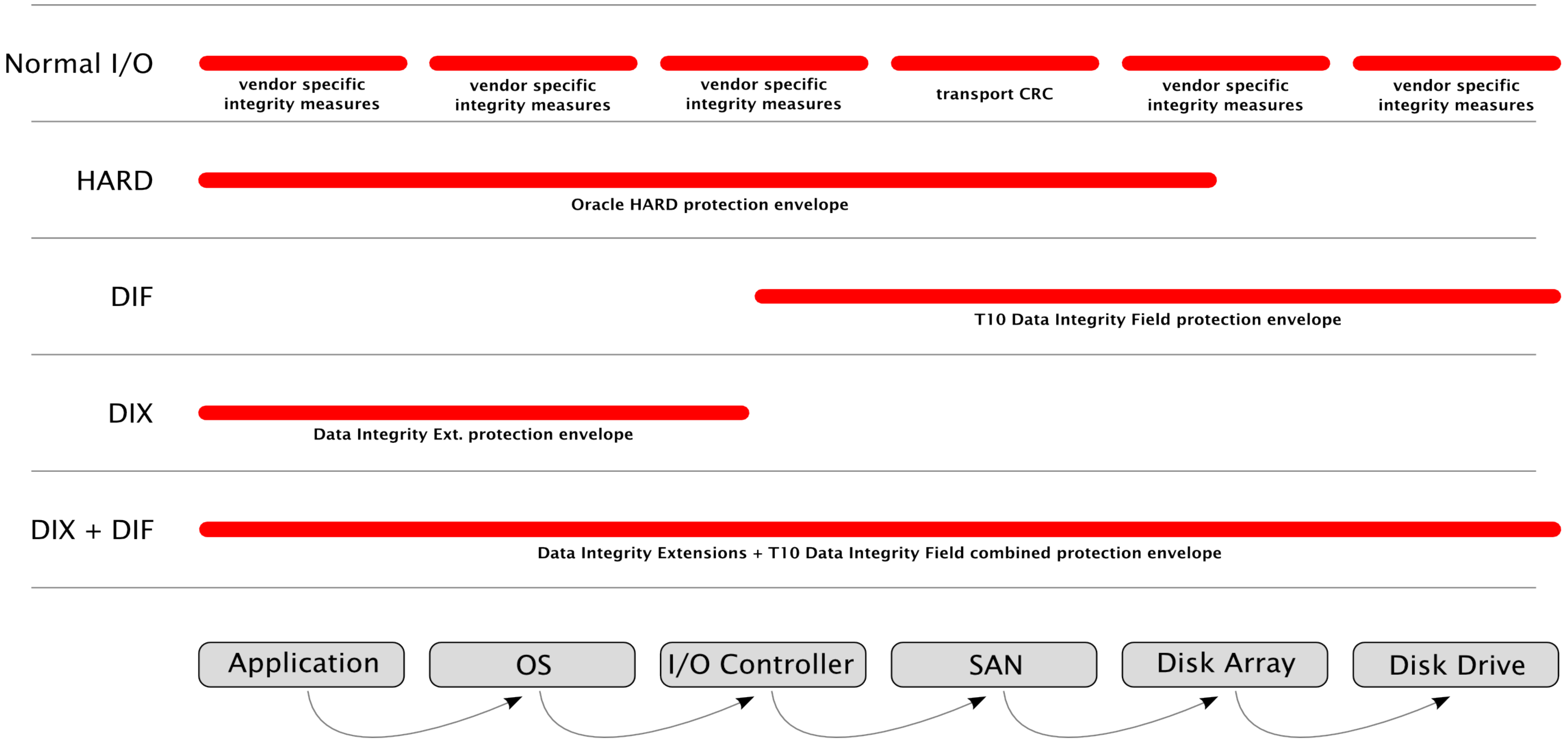
DIX Operations



Data Integrity Extensions + DIF I/O



Protection Envelopes



Linux Data Integrity

Block Layer

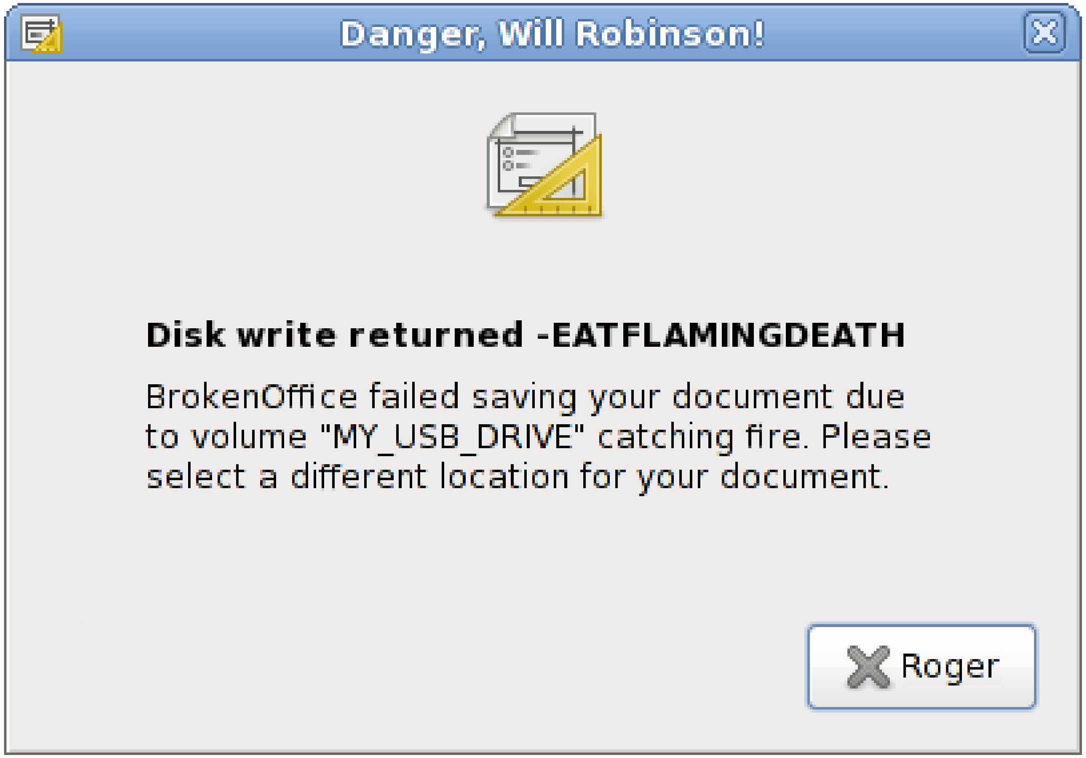
- `struct bio`
  - `bio_integrity_payload`
    - Integrity `bio_vec` + housekeeping hanging off of `bio`
    - Filesystem can explicitly attach it...
    - ... or block layer can auto-generate on WRITE
    - Block layer can verify on READ
  - Format of protection information opaque to block layer
- `struct block_device`
  - Has an integrity profile that gets registered by ULD
  - Layered devices must ensure all subdevices have same profile

Filesystems

- DIF application tag:
  - 2 bytes per sector for Type 1 + 2
  - 6 bytes per sector for Type 3
- FS can attach arbitrary structures which will be interleaved between the available tag space in an I/O
- Essentially allows logical (filesystem) block tagging
- FS can use tags to implement checksumming without changing on-disk format
- Another option is to write stuff that will aid recovery (back pointers, inode numbers, etc.)

User Application Interfaces

Wouldn't it be nice if...



Our UNIX Heritage

- Then:
  - `cat foo | frob | mangle > bar`
  - Applications were short lived
  - -EIO meant that the pipeline broke and operator had to fix it
  - Input easily reproducible by restarting pipeline
- Now:
  - Oracle, mysqld, OpenOffice.org, firefox, etc.
  - Applications run forever
  - -EIO never gets to most applications thanks to buffered writes
  - Data mainly comes from user input and the network, often not reproducible
  - But we're still using the old API

Async I/O

- There are other options:
  - Linux AIO
  - POSIX aio
  - fibrils/syslets
- But hardly anybody is using them
- Almost complete lack of interest
- Apparently existing interfaces are good enough for applications that don't really care about data
- And/or errors happen infrequently enough that they are not considered a real problem
- Anal-retentive applications use direct or sync I/O

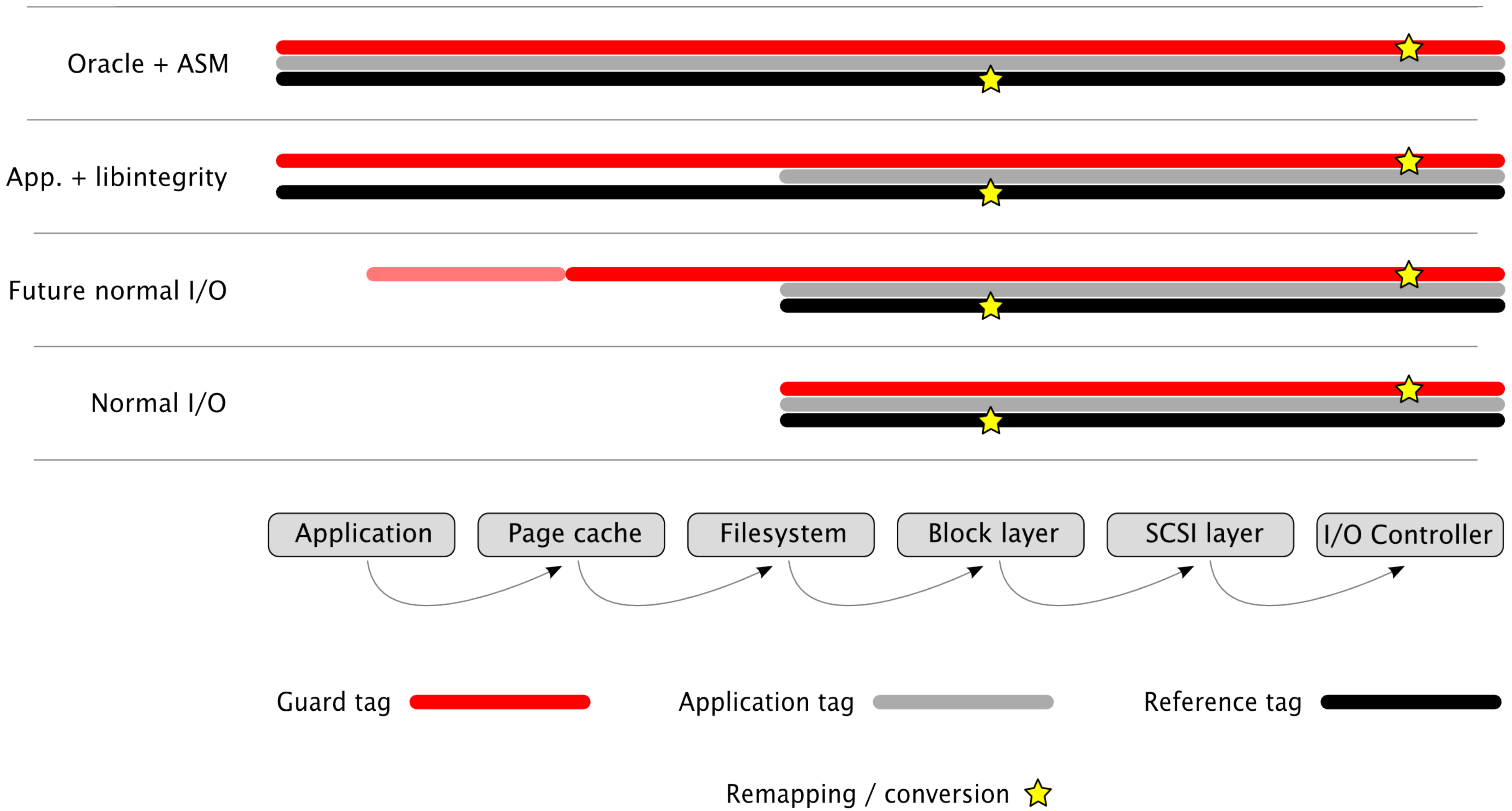
Oracle's swiss army knife: ASM

- “Automatic Storage Management”, essentially a logical volume manager internal to the Oracle DB
- ASMLib: Userland library that implements the Oracle-specified interface
- oracleasm: Kernel module that receives IOCBs from ASMLib and feeds them to the block layer as bios
- Supports protection information passthrough
- Could potentially be made generic
- Async I/O on steroids
- Woohoo! Yet another API!

So where do we go from here?

- Interface must be as close to the traditional read()/write() model as possible or nobody will bother
  - How do we get protection information in and out of the kernel?
  - Must also work for current aio users
  - Augment existing interfaces or create new API?
- Completion
  - Which async error notification mechanism of the week?
  - And how do we go about extended error information?
  - -EIO in itself isn't exactly helpful

User API vs. Data Integrity



More Info

- http://oss.oracle.com/projects/data-integrity/
  - Documentation
  - DIX specification
  - Patches
  - Funny hats
  - Source repository