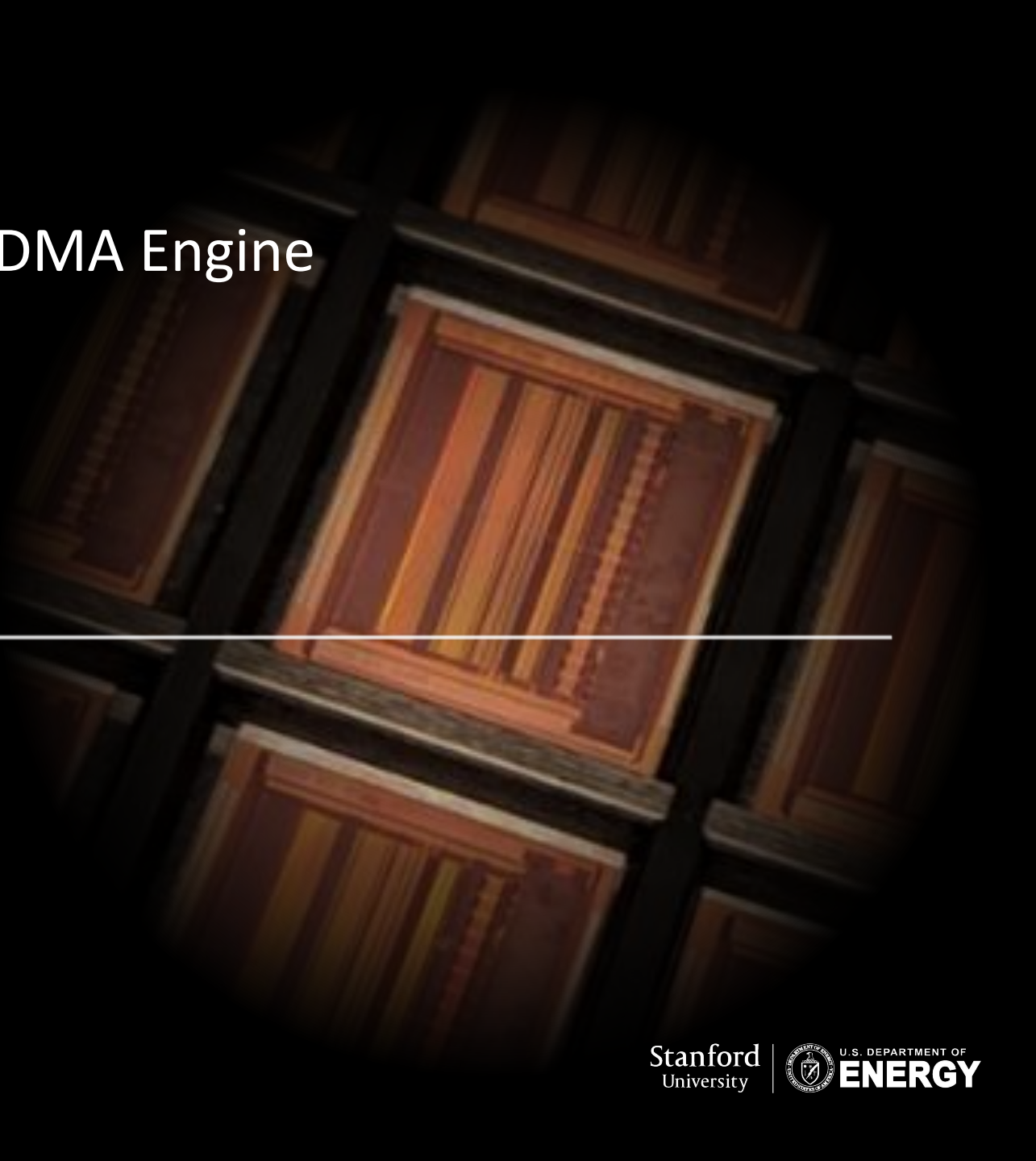
An Open Source General Purpose DMA Engine For DAQ Systems



Ryan Herbst For SLAC TID Instrumentation

November 9, 2023

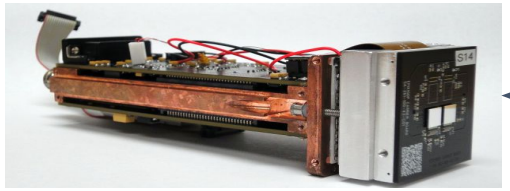
SLAC NATIONAL ACCELERATOR LABORATORY

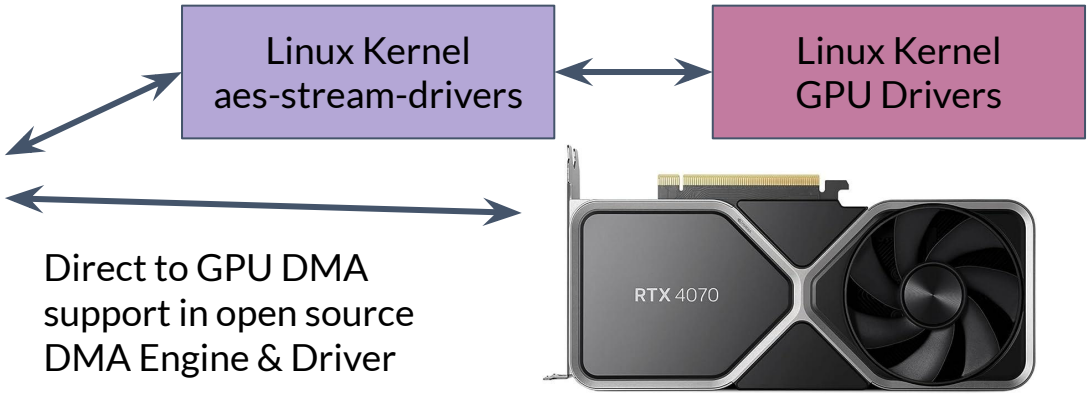
Stanford University | U.S. DEPARTMENT OF ENERGY

Contents

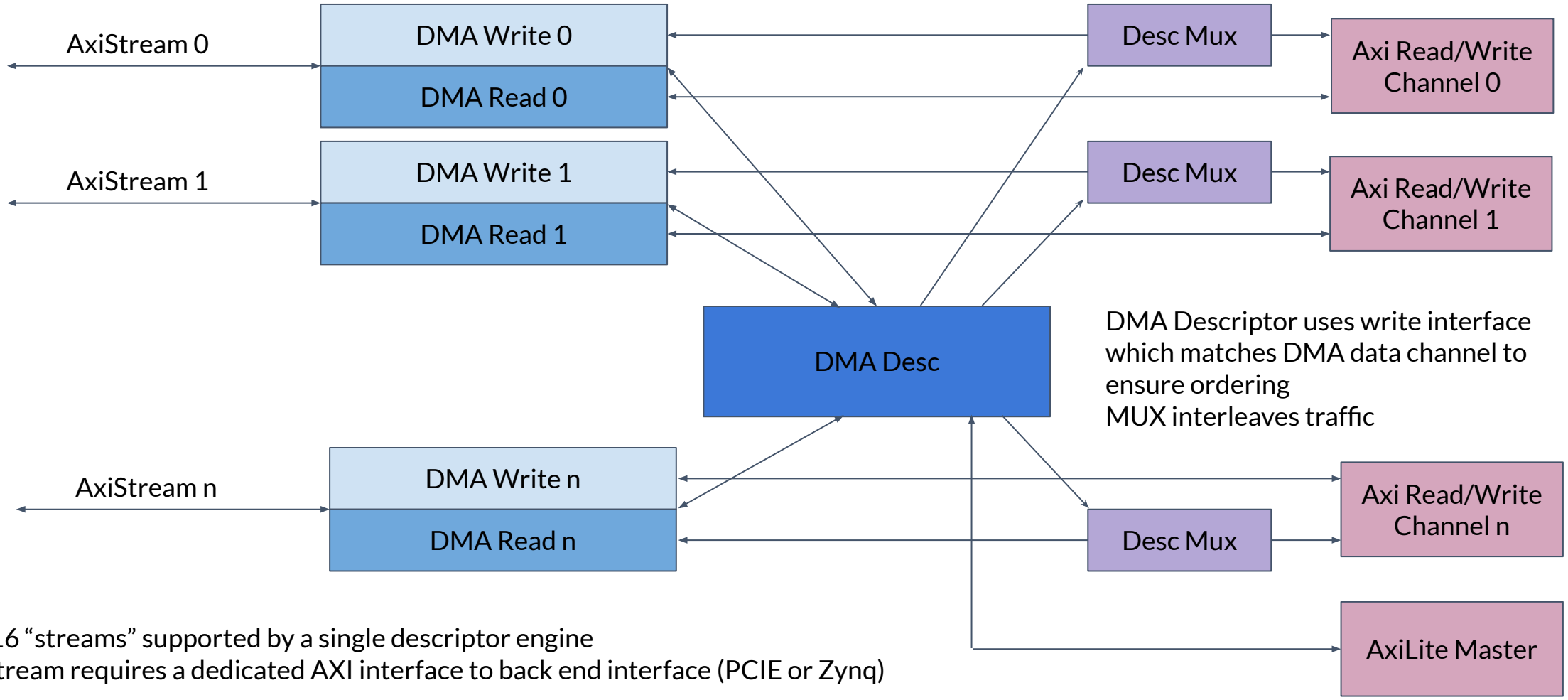
Key Features

- Supports multiple hardware lanes
  - Supports interleaved AXI-Stream traffic (tDEST) on each hardware lane
  - Per tID back pressure support
- Supports Zynq ACP interface for automatic cache line processing
  - Common API for Zynq based and server based applications
- Configurable cache modes
  - Coherent (no caching)
  - Stream (cache with pre-post dma flush
  - Zynq ACP mode (hardware managed caching)
- Supports both interrupt and polling mode operation
- Supports user space memory mapping of DMA buffers
  - Minimizes memory to memory copies
- Supports multiple buffer receive and transmit to/from user space to reduce user to kernel space overhead
- Supports larger frame receive across multiplied DMA buffers
- Supports both high bandwidth and high buffer rate applications
- Open source firmware and kernel driver





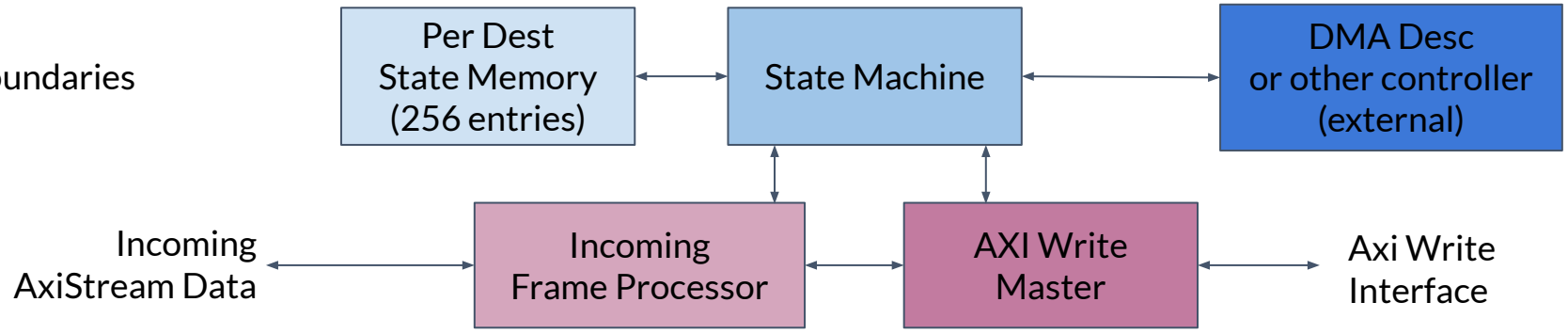
Overview



- Up to 16 “streams” supported by a single descriptor engine
- Each Stream requires a dedicated AXI interface to back end interface (PCIE or Zynq)

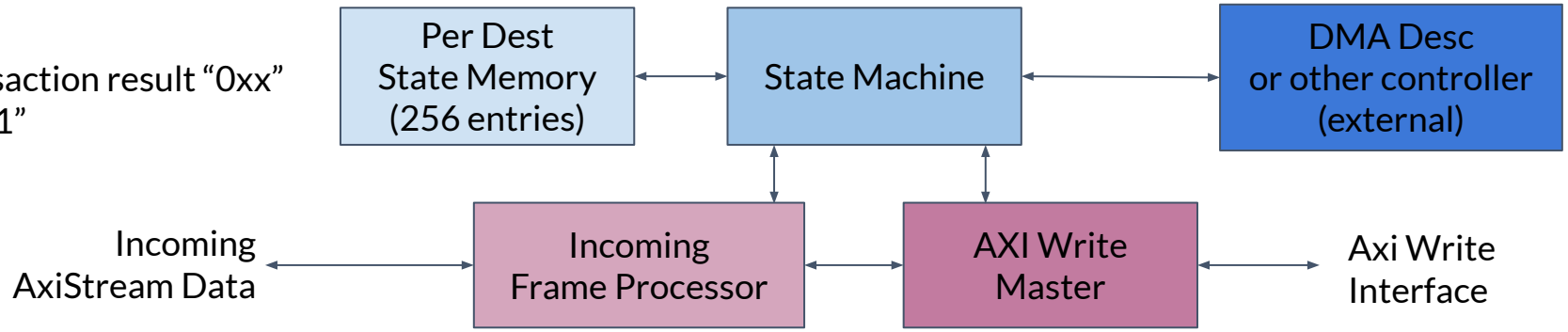
DMA Write Engine

- **Designed to work with or without DMA Desc block**
  - Useful for ring buffer controllers or direct to accelerator (GPU) DMA transactions
- **WriteDMA engine can support up to 256 interleaved frames**
  - Identified by tDEST
  - Interleave must occur on a full width boundary of the AxiStream interface
  - Sideband memory is used to keep track of the write buffer address and receive count for each tDEST
- Transaction burst size is set at compile time
- Each new frame on a unique tDest results in a descriptor request to the external controller
  - Contains the tID and tDest field
- Descriptor response contains the following:
  - Buffer address
  - Max frame size
  - Meta enable flag and meta address
  - Continue enable flag
  - Drop enable flag
  - Unique buffer ID
- All transactions aligned to 4K boundaries



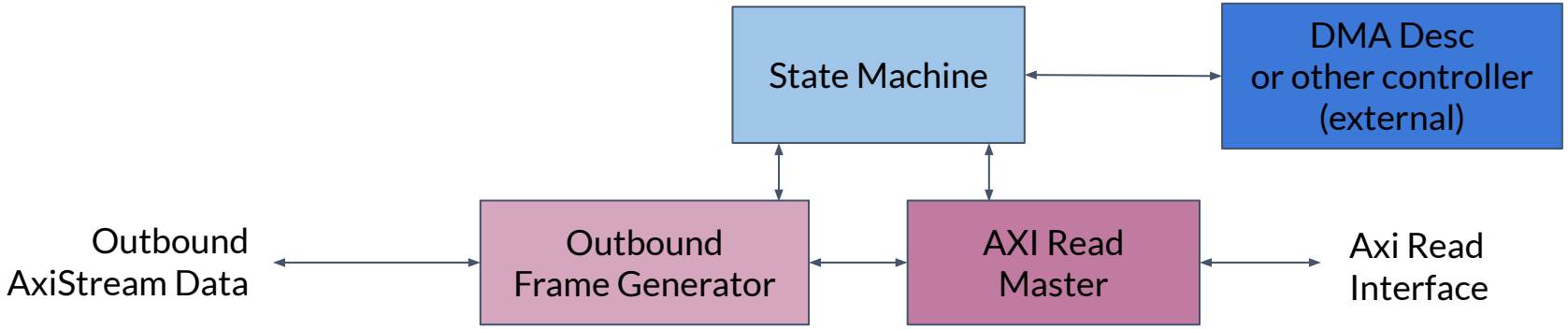
DMA Write Engine

- Drop enable flag results in incoming frame being discarded
  - As a result of global or per tDEST/tID configuration
- **Continue enable flag determines if a buffer overflow results in an error or if the remaining data is simply continued in another buffer**
  - Allows very large frames beyond a fixed buffer size to be supported
- **Meta enable flag is for including received frame information at a specified "meta" address**
  - Includes received size, first and last "user" fields, continue flag and error flags in a 64-bit word
  - Typically used for special applications such as DMA writes directly to a GPU or other accelerator
- Completed frames are handed back to controller with appropriate meta information
  - Buffer ID
  - First and last "user" fields
  - Frame size
  - Frame tDest
  - Frame tID
  - Frame continue flag
  - Frame error result field
- Result field can be one of:
  - Non zero write AXI transaction result "0xx"
  - Transaction timeout "011"
  - Overflow Flag "100"



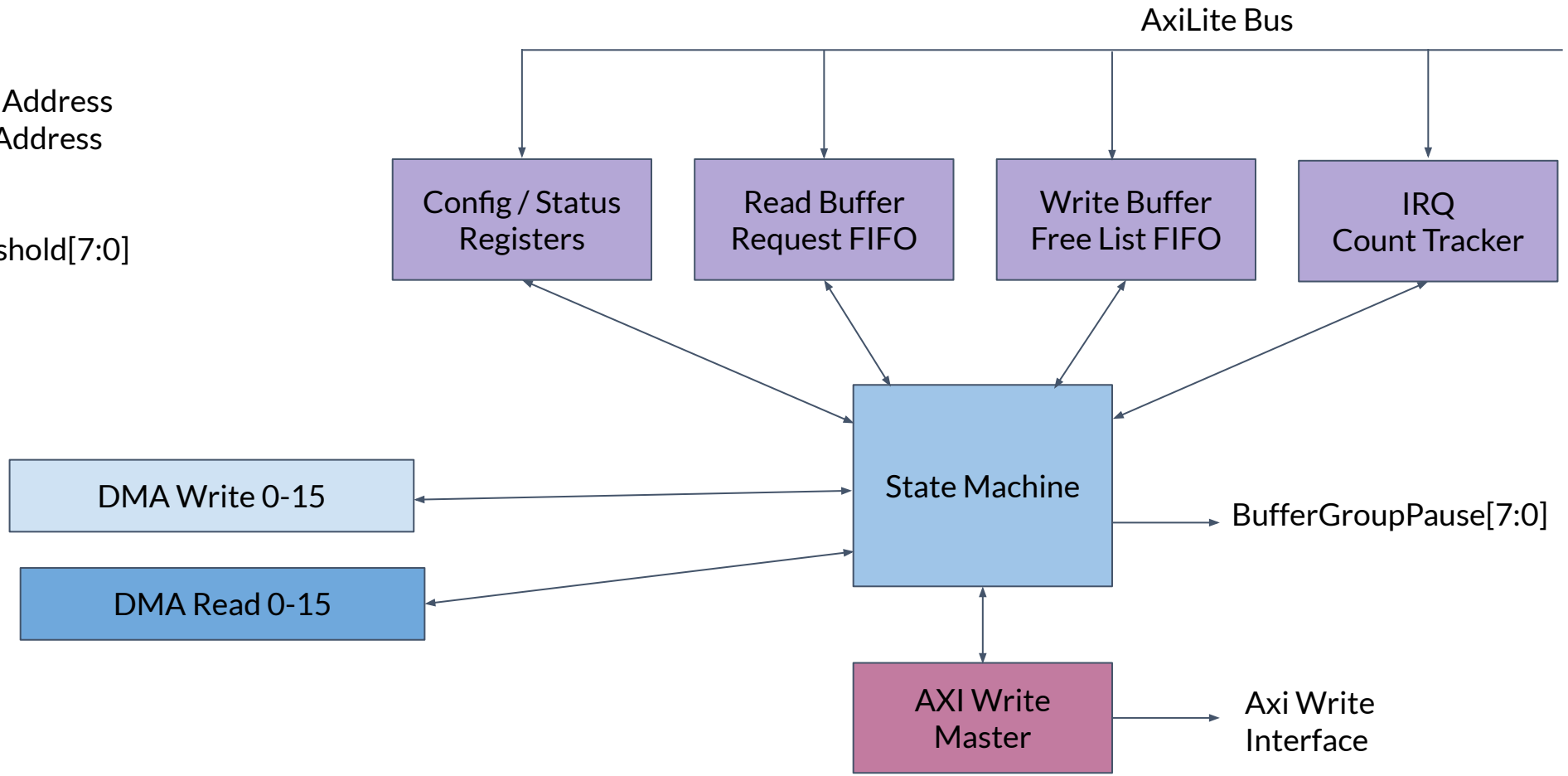
DMA Read Engine

- **Designed to work with or without DMA Desc block**
  - Useful for ring buffer controllers
- Read engine supports a single frame at a time
- Transaction is initiated by request from external controller
  - Buffer ID
  - Read Address
  - Read size
  - tDest value
  - tId value
  - First and last “user” fields
- All transactions aligned to 4K boundaries
- **DMA engine will read ahead a compile time configured number of bytes to ensure read pipeline remains full**
- DMA engine will respond to external AXI stream backpressure



Descriptor Controller

- **Coordinates DMA transactions for up to 16 read/write lanes**
- Configuration Values
  - Enable
  - InterruptEnable
  - ContinueEnable
  - DropEnable
  - Write Ring Buffer Address
  - Read Ring Buffer Address
  - MaxFrameSize
  - IntHoldOff
  - BufferGroupThreshold[7:0]

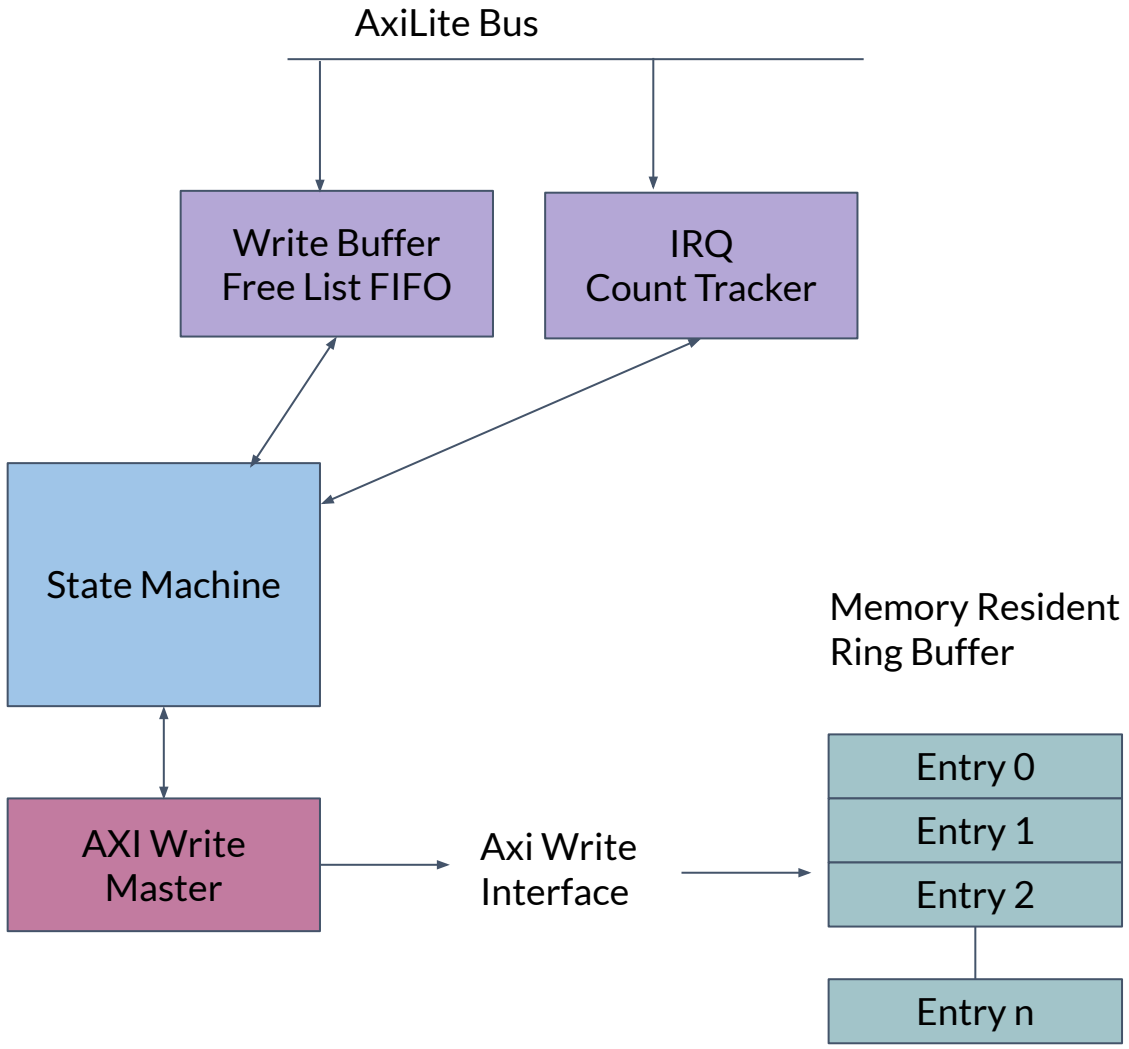


Descriptor Controller Operation For Write Data

- Each write engine will request a DMA buffer when it requires one
  - Typically occurs when a new frame is received with a unique tDEST number
- When the write is complete the write engine will return the buffer to the controller with associated meta-data
- The Descriptor Controller state machine will form a 128-bit word which contains the buffer and meta data information
  - **This 128-bit word is then DMA'd to the next location in a memory resident ring buffer**
  - Bit 127 serves as a "valid" bit for the entry
- Each buffer receive requires the following transactions:
  - 1 32-bit write to disable IRQ (within interrupt handler)
  - **2 32-bit writes to return buffer to free list (address + ID)**
  - 1 32-bit write to ack and enable interrupts

- **Each irq call can result in multiple buffer transfers**
  - IRQ request count tracks number of transaction to manage
  - Driver disables interrupt and handles multiple read and write buffer returns
  - Ack/Enable write include enable flag and number of buffers which were serviced

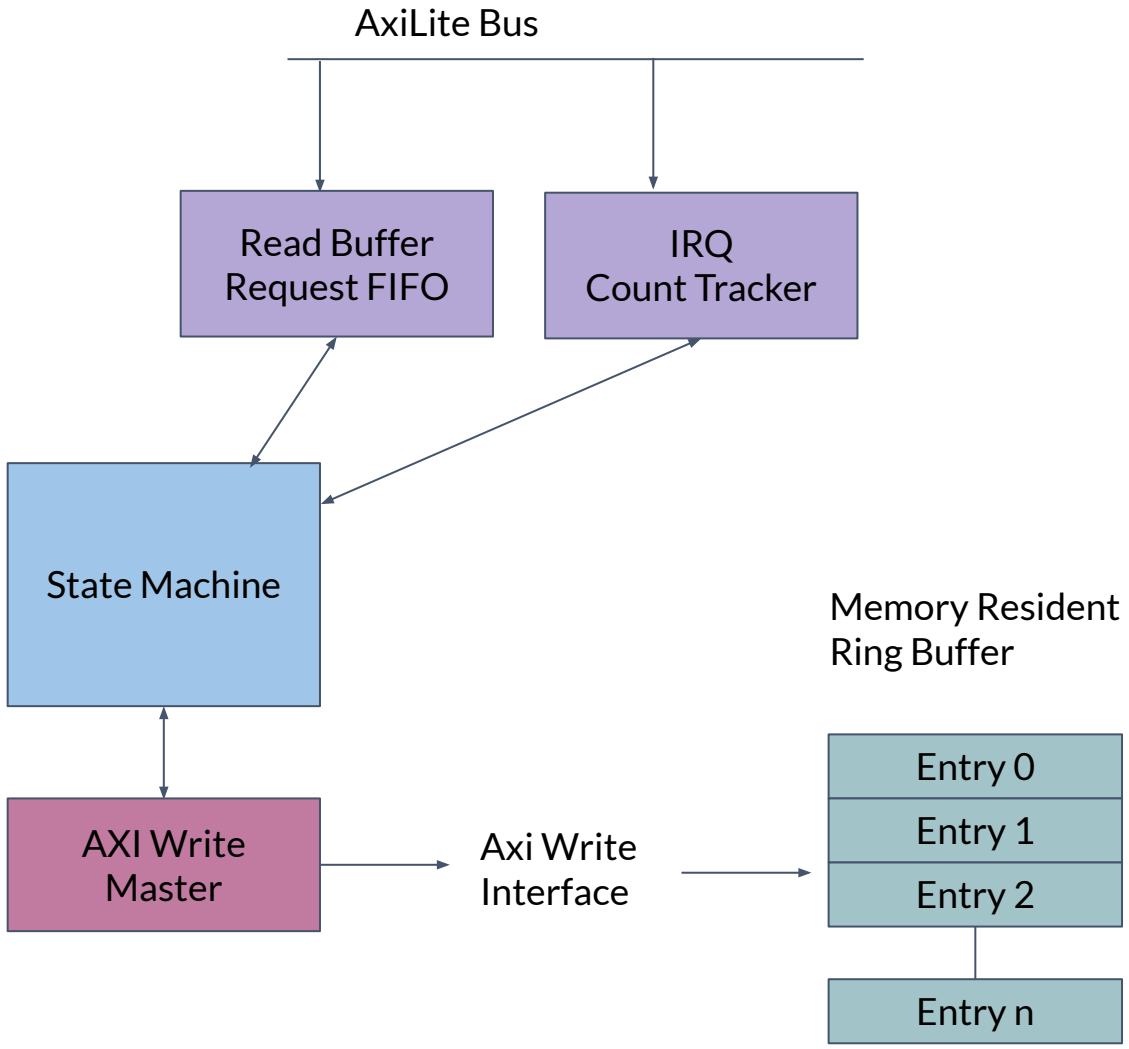
- **No register read transactions involved in inbound DMA!**



Descriptor Controller Operation For Read Data

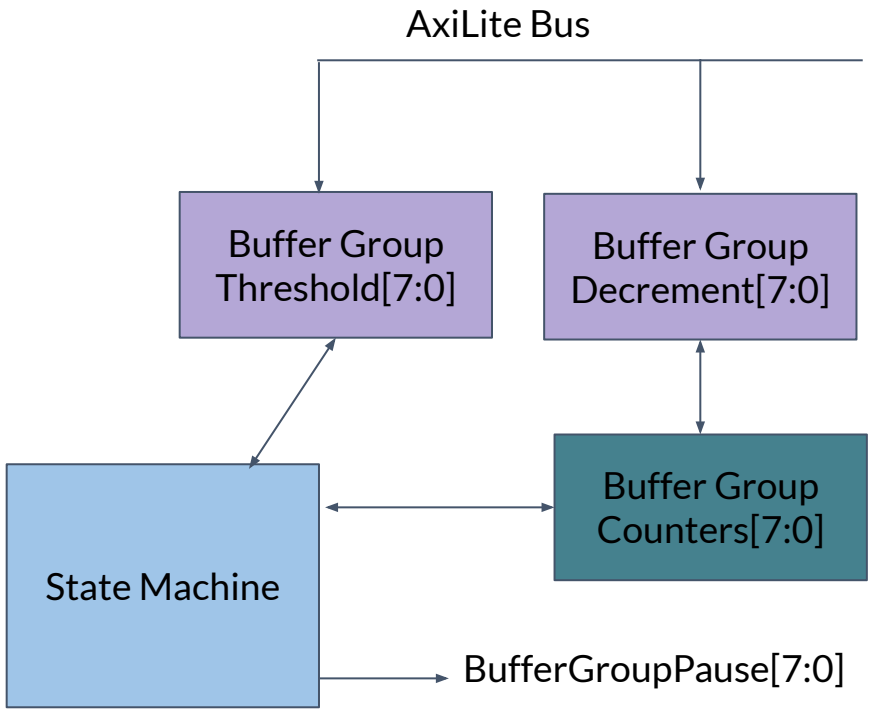
- A read transaction is started when the driver posts a reset request to the "Read Buffer Request FIFO"
  - 128 bit write (4 x 32 bits)
- Read request is then passed to the associated read DMA engine
- When the read is complete the read engine will return the buffer to the controller with associated meta-data
- The Descriptor Controller state machine will form a 128-bit word which contains the buffer and meta data information
  - **This 128-bit word is then DMA'd to the next location in a memory resident ring buffer**
  - Bit 127 serves as a "valid" bit for the entry
- Each read transmit requires the following transactions:
  - **4 x 32-bit write to generate read request**
  - On buffer return:
    - 1 32-bit write to disable IRQ (within interrupt handler)
    - 1 32-bit write to ack and enable interrupts

- **No register read transactions involved in outbound DMA!**

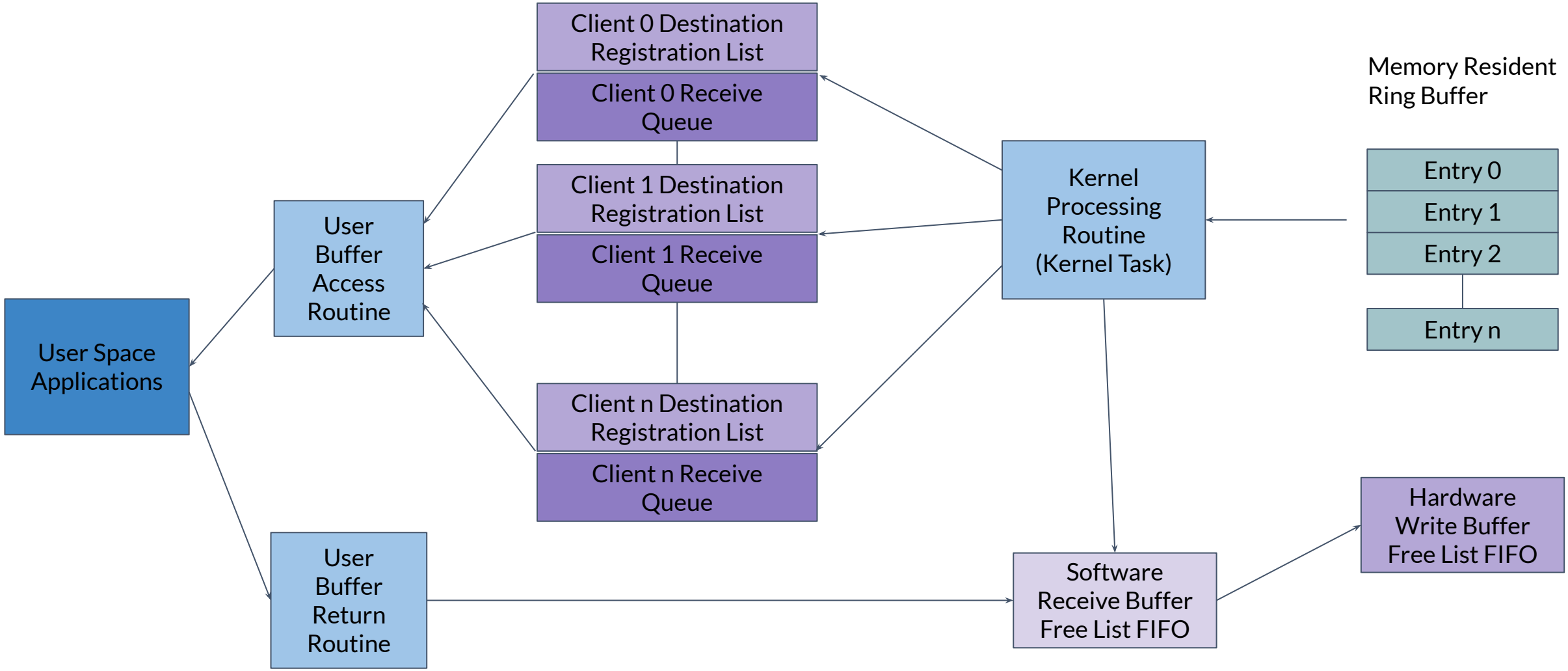


Buffer Group Thresholds & Pause Signals

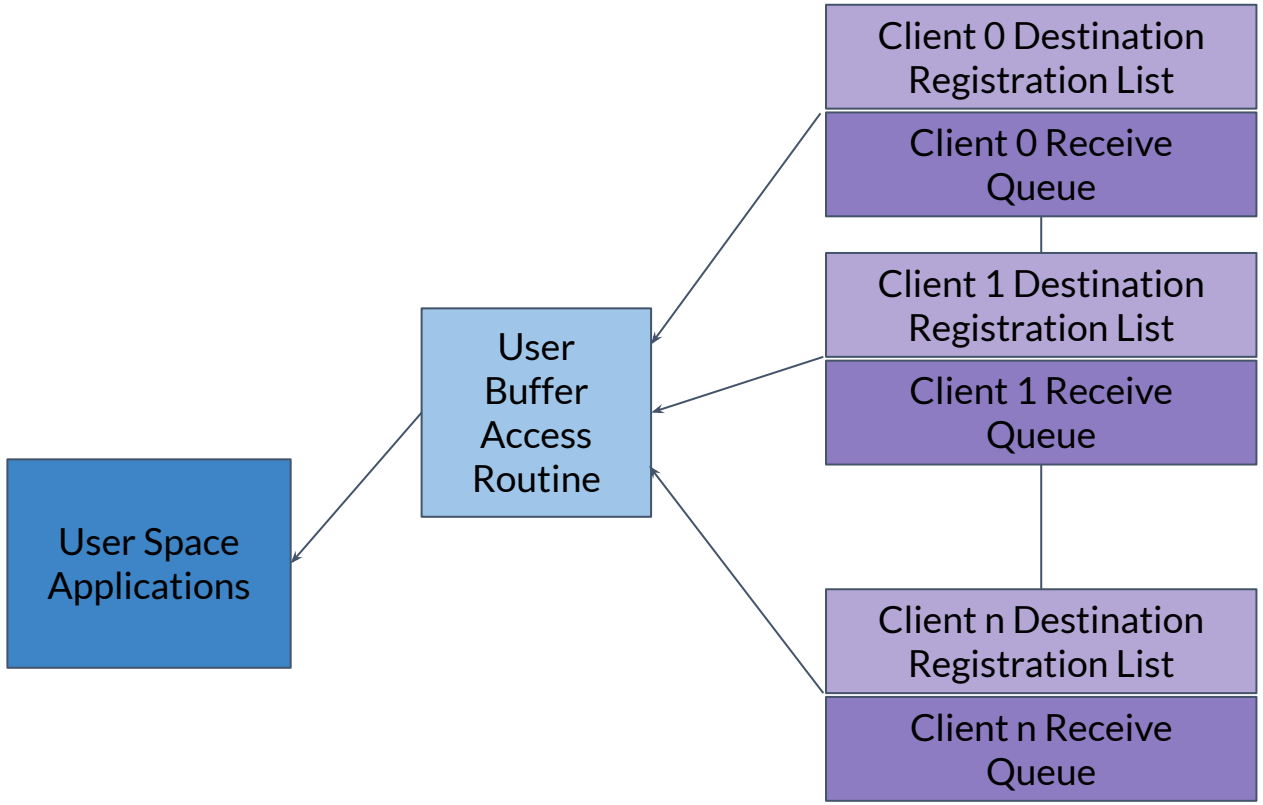
- A single buffer free list is shared among all of the separate DMA write controllers
  - Can result in buffer starvation if some channels are generated more traffic than others
  - User level software slow downs on some channels may result in buffer free list starvation
- **Buffer Groups provide a mechanism to balance out buffer usage from various sources depending on their AxiStream tID field**
  - The lower three bits of the tID field define which buffer group the traffic is associated with
  - Buffer groups can be independent of the write engine or the tDEST field of the incoming frames
- The buffer group logic keeps track of the number of outstanding buffers that are associated with each buffer group
  - The associated buffer group counter is incremented each time a buffer is allocated for a given buffer group
  - The counter is decremented when the buffer is returned to the free list from the driver software
  - The associated BufferGroupPause signal is asserted when the number of outstanding buffers for a group exceeds the configured threshold
  - External logic can then pause the incoming frames for a particular buffer group



DMA Kernel Driver - Receive Processing (read calls)



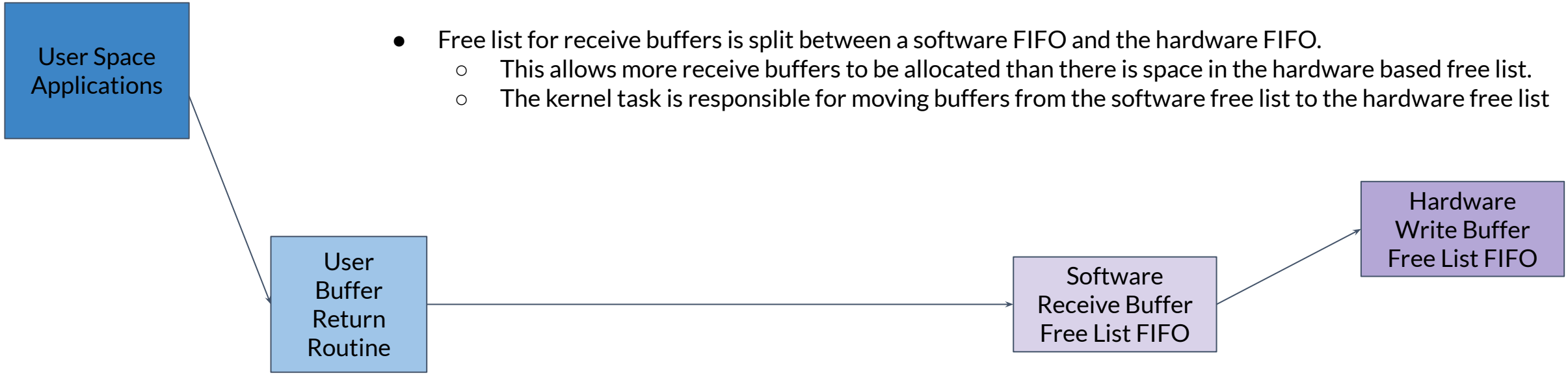
DMA Kernel Driver - Receive Processing (read calls)



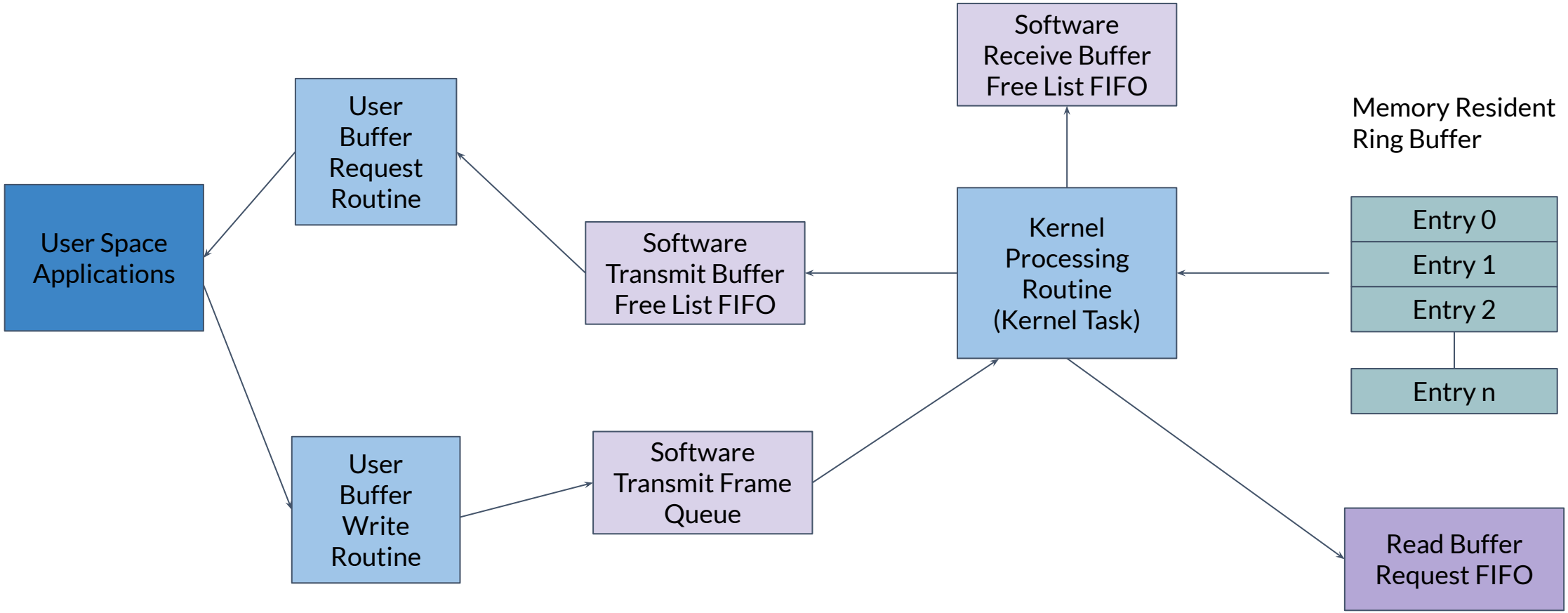
- When a user space application opens the file descriptor associated the following operations occur:
  - **The user space application registers for the destination/channel combinations it wishes to receive data from**
  - An associated software driver queue is created for the user space application
  - dmaSetMask call is used to register for receive channel and destination
    - Each channel/dest can only be registered once
- **Optionally the user space application may memory map the kernel buffers into user space**
  - The user application then tracks a list which associates each "buffer index" but a virtual memory address in user space
  - dmaMapDma call is used
- Possible user space receive calls:
  - dmaRead
    - Receive a single frame with copy to user space
  - dmaReadIndex
    - Receive a single frame using user space mapped buffer
  - dmaReadBulkIndex
    - **Receive one or more frames using user space mapped buffers**

DMA Kernel Driver - Receive Processing (read calls)

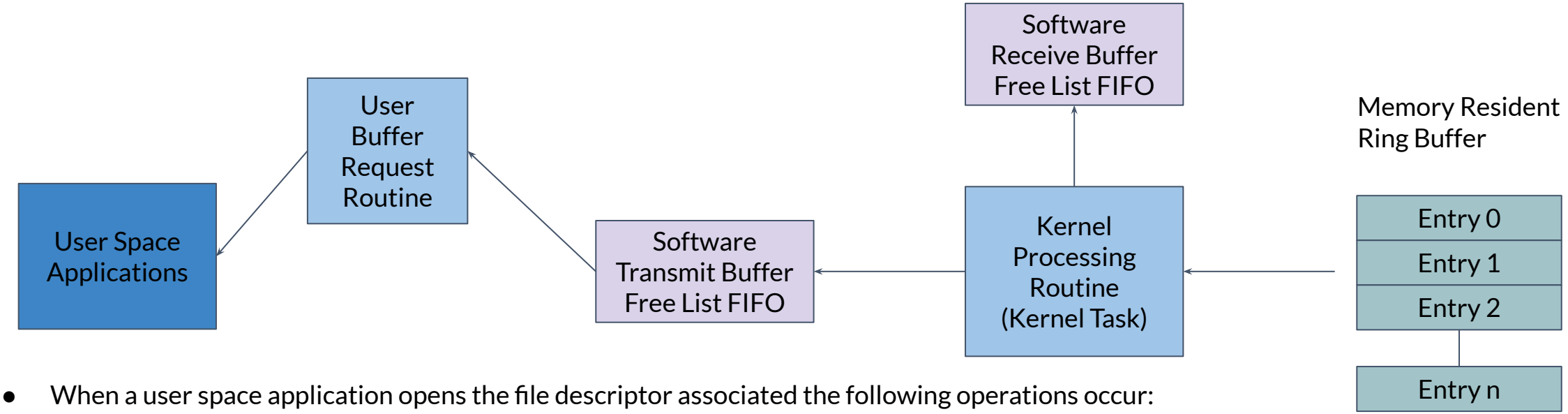
- **When user space mapped buffers are utilized the user application owns the buffer until it is returned**
  - dmaRetIndex
    - Return a single buffer via index
  - dmaRetIndexes
    - Return multiple buffers via index
- **User space buffers may also be passed back to dmaWriteIndex call, useful for receiving data, multipulating contents and then transmitting modified frame.**
  - Buffer is then used in outbound data transmit and then returned to receive free list when outbound transmit is complete
- Free list for receive buffers is split between a software FIFO and the hardware FIFO.
  - This allows more receive buffers to be allocated than there is space in the hardware based free list.
  - The kernel task is responsible for moving buffers from the software free list to the hardware free list



DMA Kernel Driver - Transmit Processing (write calls)

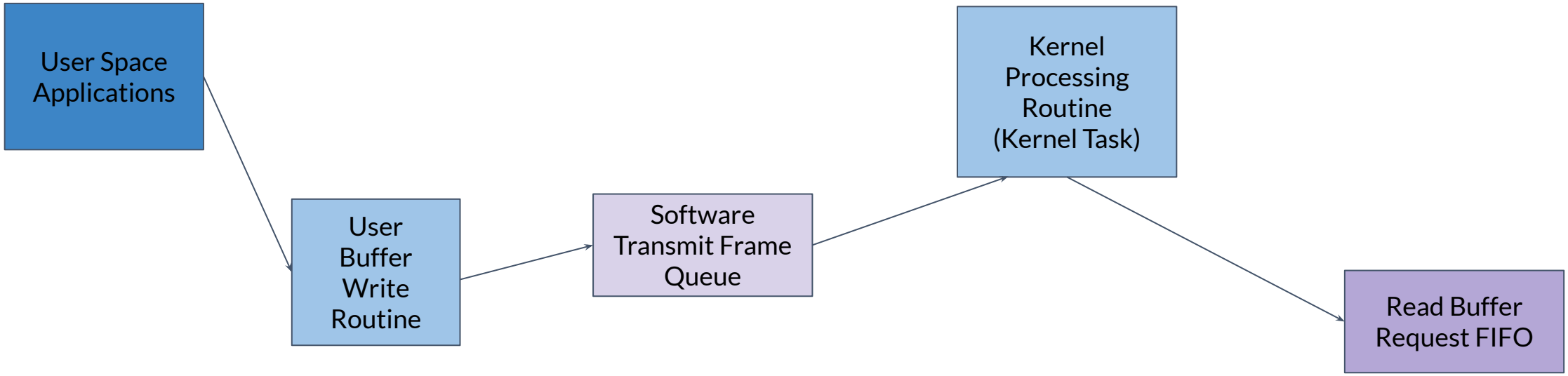


DMA Kernel Driver - Transmit Processing (write calls)



- When a user space application opens the file descriptor associated the following operations occur:
  - **The user space application registers for the destination/channel combinations it wishes to receive data from**
- **Optionally the user space application may memory map the kernel buffers into user space**
  - The user application then tracks a list which associates each “buffer index” but a virtual memory address in user space
  - dmaMapDma call is used
- When memory mapped kernel buffers are used the user space application must first request a buffer to be allocated:
  - dmaGetIndex
    - Request a buffer be allocated for outbound data transmit (write call)
- When using standard “copy from user space” write calls a buffer is internally allocated for copy within the kernel driver

DMA Kernel Driver - Transmit Processing (write calls)



- When the user space application is ready to transmit data, it will perform one of the following calls:
  - dmaWrite
    - Transmit a single frame with copy from user space
  - dmaWriteIndex
    - Transmit a single frame using user space mapped buffer
  - dmaWriteVector
    - Transmit one or more frames using copy from user space, passing an iovec structure
  - dmaWriteIndexVector
    - **Transmit one or more frames using user space mapped buffers, passing an index array**

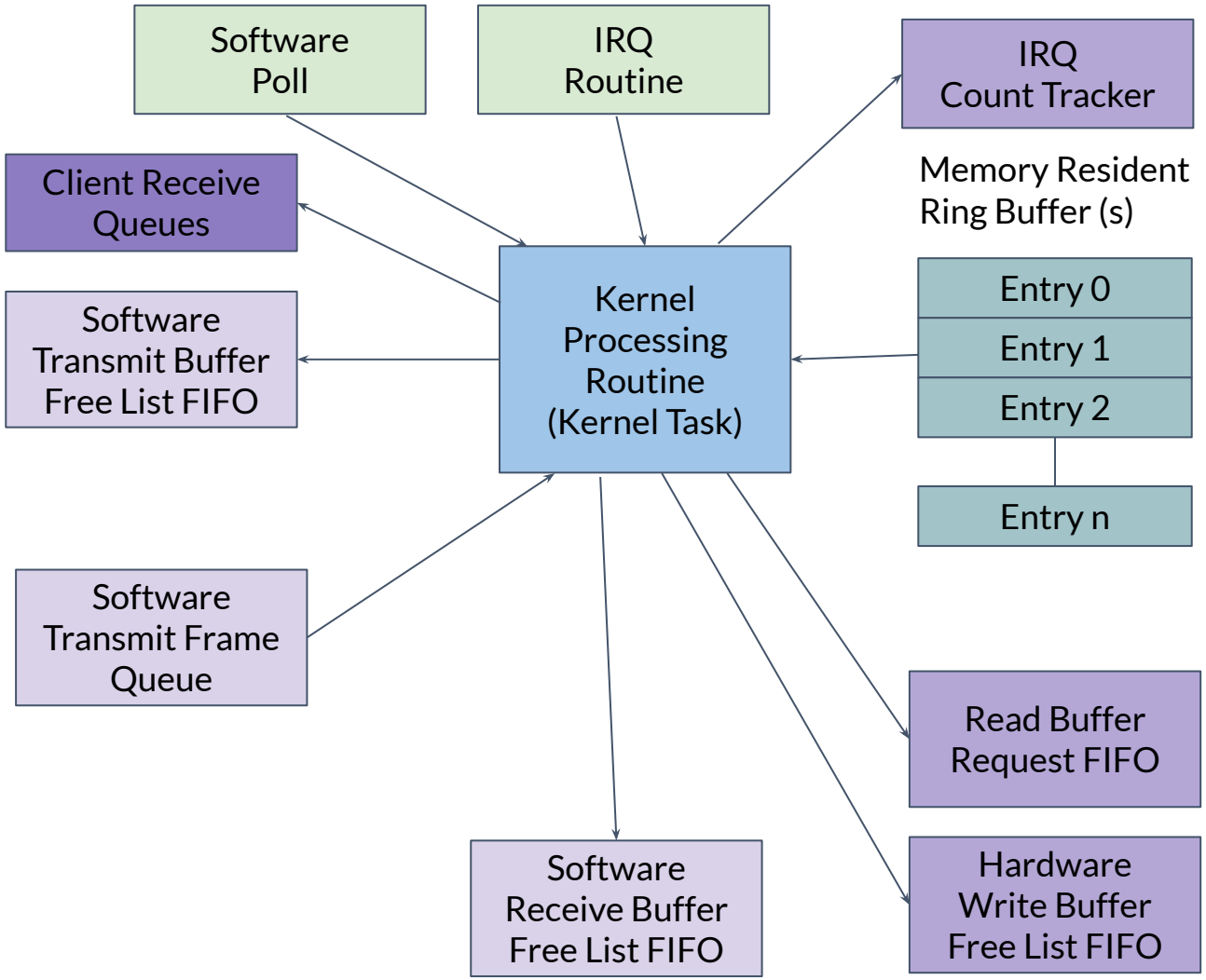
- The kernel driver will then inserted the requested frame into a software resident transmit queue
  - The kernel task will then deliver the buffer information to hardware

DMA Kernel Driver - Kernel Processing Routine

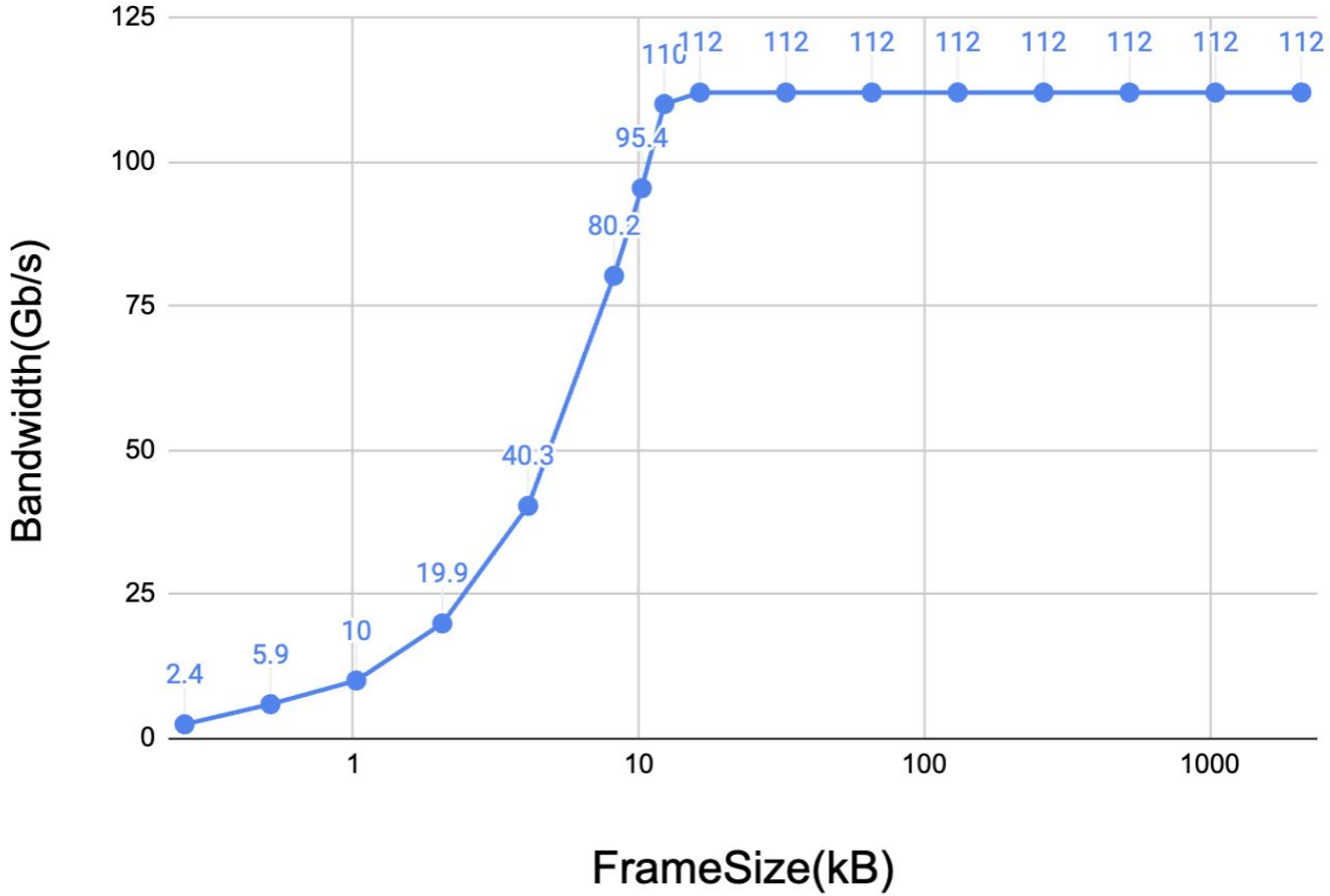
- The kernel processing routine is responsible for the following functions
  - **Monitor inbound receive memory ring buffer**
  - Move inbound buffer to appropriate user space receive queue
  - Monitor return transmit memory ring buffer
  - Move buffers to software receive free list
  - **Move buffers from software receive free list to hardware free list**
  - **Move buffers from software transmit queue to hardware read buffer request FIFO**

- Kernel processing routine is normally triggered by hardware interrupt
  - **Kernel task is then scheduled**
  - When inbound data is received and entry is added to receive ring buffer
  - When outbound data transmit is complete and entry is added to transmit return ring buffer
  - A periodic trigger from hardware to ensure housekeeping

- **Kernel processing routine can also be run in a poll mode with persistent kernel task running**
  - When interrupts are not wanted by real time systems



DMA Engine / Driver Performance



- Demonstrate up to **112 Gb/s for large frames** (PCIe GEN3 x 16, 1MB frames)
- Demonstrate **> 1MHz frame rate for small frame** (<128B) without frame batching

Conclusion

- The SLAC open source DMA engine and associated driver has been a workhorse for LCLS2 DAQ along with multiple HEP experiments
  - ZYNQ based data processing (LSST, HPS)
  - LDMX PCIE express receiver
  - NEXO PCIE express receiver and processor
- We have been able to maintain a consistent firmware and software API across multiple PCIE express hardware boards
  - Consistent API for ZYNQ and PCI-Express based boards
- Support for high bandwidth as well as high even rate transactions
- Support for polling and interrupt mode operation
- To Do List:
  - Support user space driver operation for single consumer real time applications
    - Wanted for D3D fusion reactor real time environment
- Firmware and associated driver are open source
  - https://github.com/slaclab/surf
  - https://github.com/slaclab/aes-stream-drivers