Using Multicore Navigator

Multicore Applications

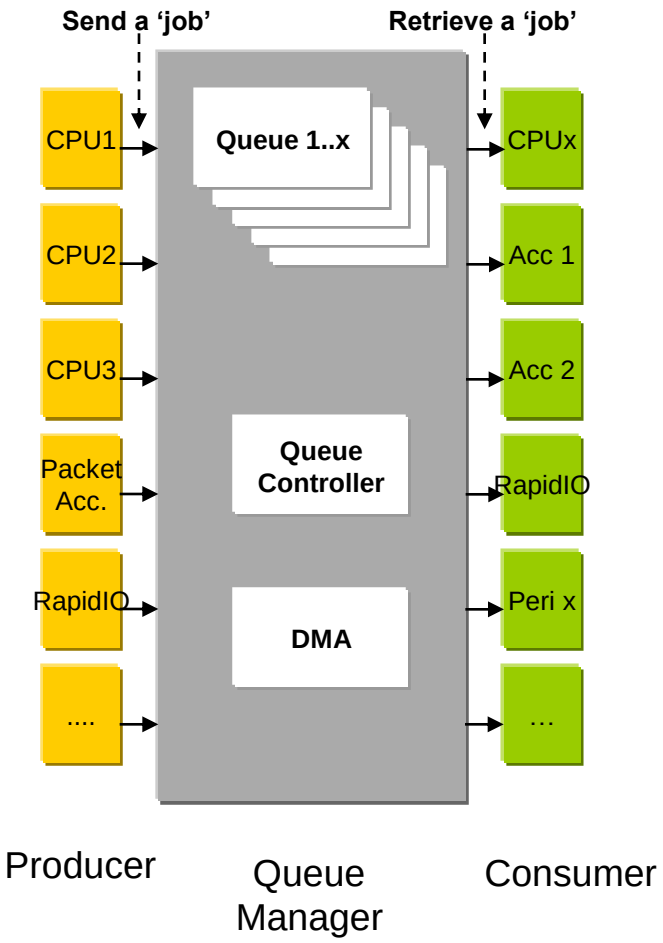
Agenda

1. **Multicore Navigator Architecture Overview**
    a. Queue Manager Subsystem (QMSS)
    b. Packet DMA (PKTDMA)
2. Working Together
3. Configuration

What is Multicore Navigator?

- Multicore Navigator is a hardware mechanism that facilitates data movement and multicore co-working
- Supports multiple users (players)
  - Each core in a multicore system
  - High bit-rate peripherals including Serial Rapid I/O (SRIO), Antennae Interface (AIF2), Network Coprocessor (NETCP), and FFTC
- Users can think of the Navigator as a mailbox mechanism with many additional and improved functions.
- Designed to be a “fire and forget” system; Load the data and the system handles the rest, without CPU intervention
  - Configuration effort is performed during initialization
  - Enables short and fast run-time operation

Hardware Queue



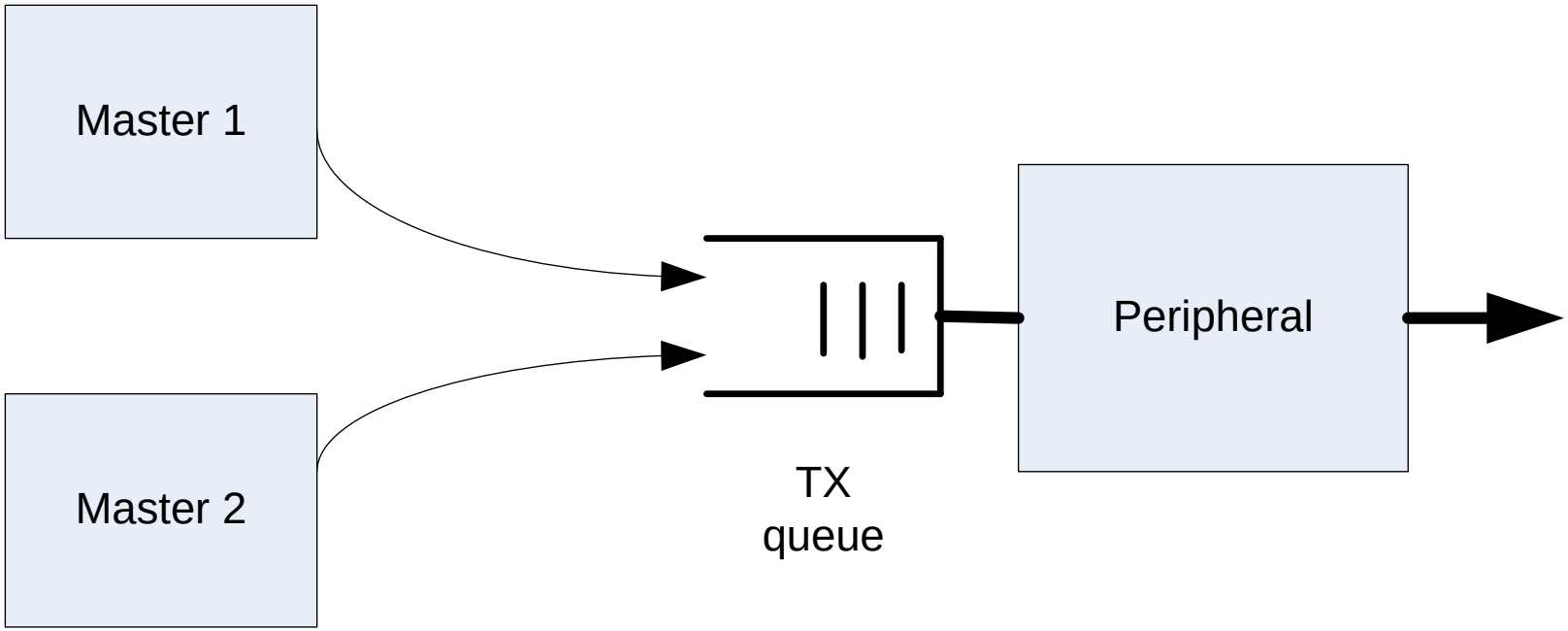
- Producer writes 'jobs' into a Queue.
- Consumer reads 'jobs' from the Queue
- Supports Multiple In – Multiple Out
  - Multiple Producers can write to the same Queue
    ⇨ Used to share common Hardware
  - Multiple Consumers can read from the same Queue
    ⇨ Used for Load Balancing
  - Example: A master CPU writes 'Process AMR channel' into the queue and the next free DSP core (one of many) or an accelerator can read the job and process it
- Abstracts the Consumer
  - Consumer can be a Hardware IP (accelerator, peripheral) or a software (ie a CPU core)
  - Transparent for the Producer
  - ⇨ 'Easy' to upgrade to new hardware. The 'job gets done'.
  - ⇨ Minimize changes to Host software, Easy maintenance

Multicore Navigator: Typical Use Cases

- Exchanging messages between cores
  - Synchronize execution of multiple cores
  - Move parameters or arguments from one core to another
- Transferring data between cores
  - Output of one core as input to the second
  - Allocate memory in one core, free memory from another, without leakage
- Sending data to peripherals
- Receiving data from peripherals
- Load Balancing and Traffic Shaping
  - Enables dynamic optimization of system performance

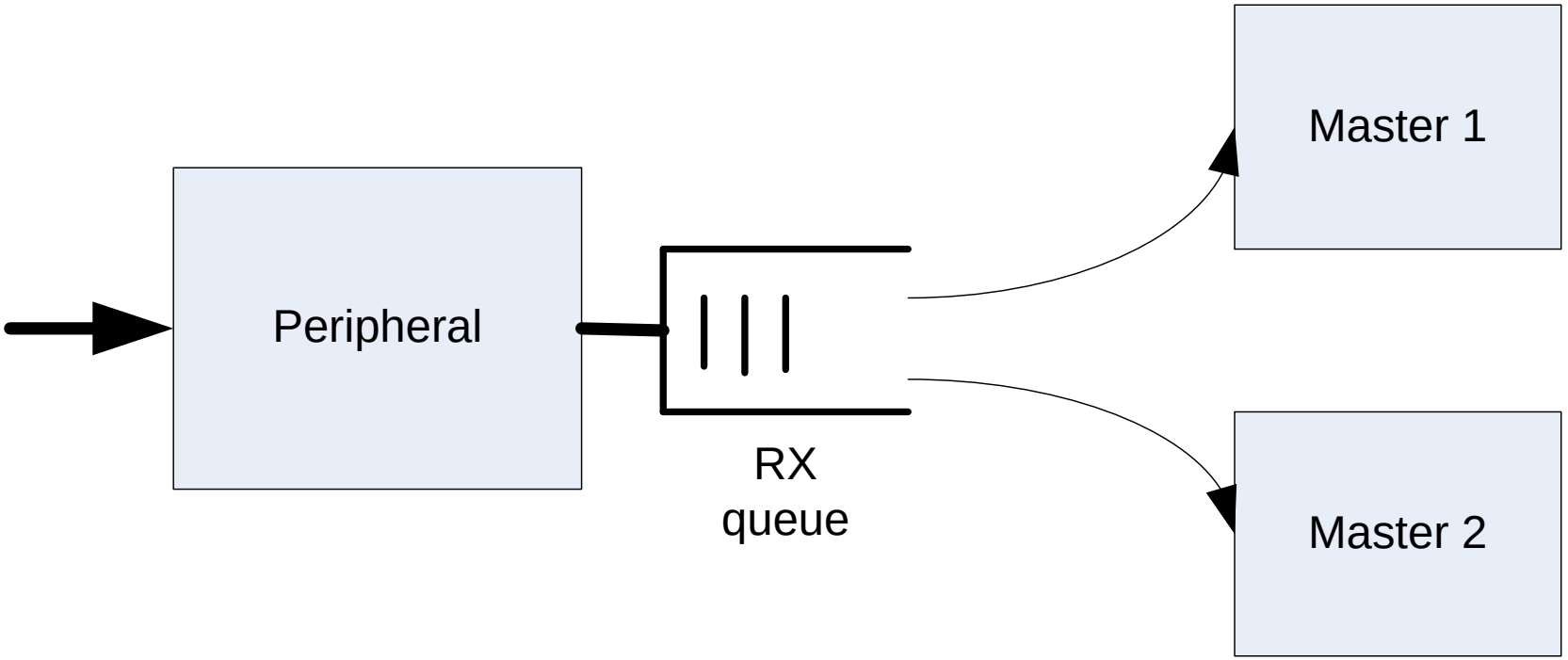
Resource Sharing Example

- typical cases for resource sharing is multiple masters request same service, for example, multiple masters transfer packets through Ethernet, that is, multiple masters push packets to same TX queue.



Load Balancing Example

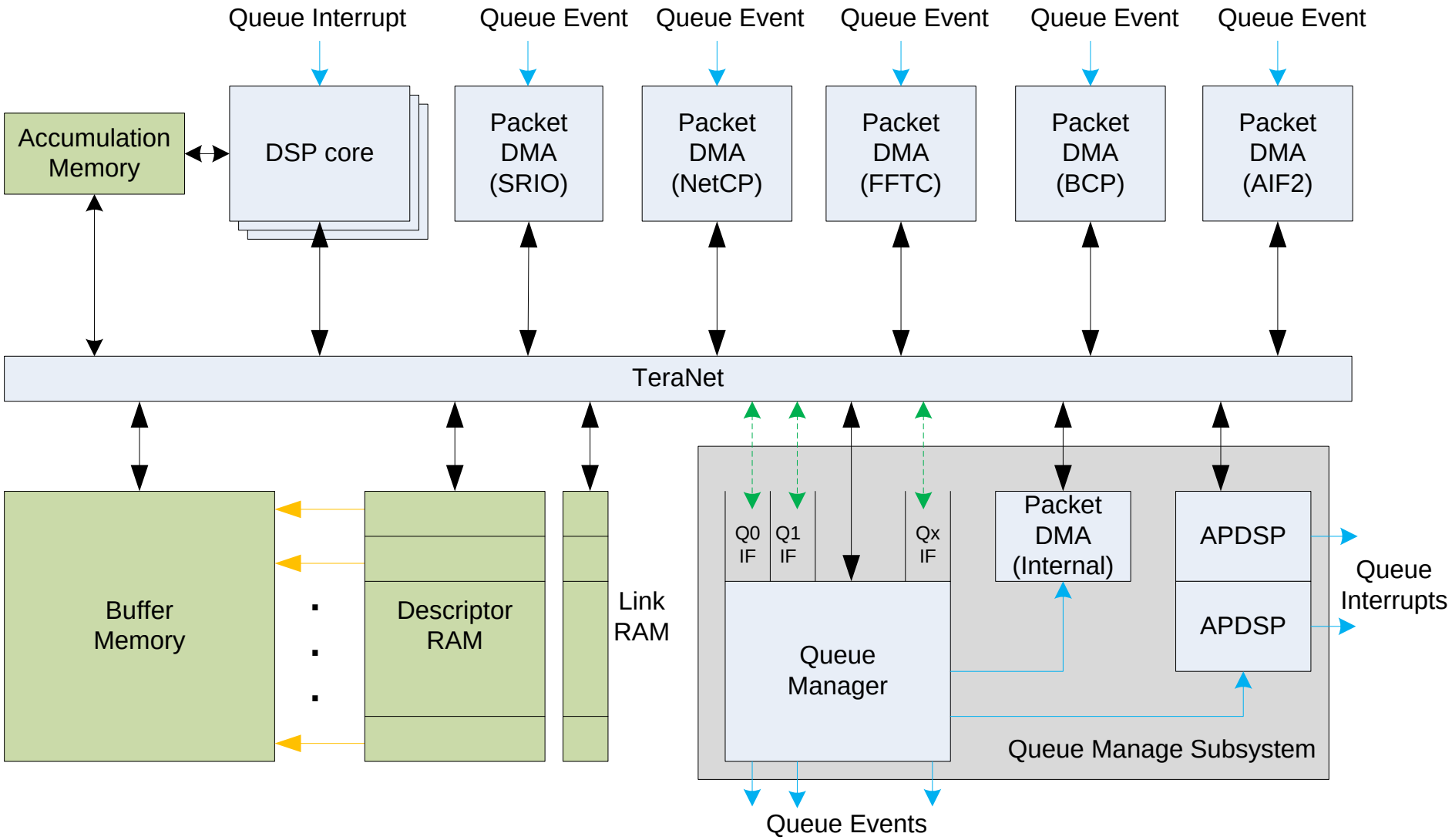
- typical case for load balancing is multiple masters process same type of services, for example, multiple masters process received packets from Ethernet, that is, multiple masters pop packets from same RX queue.
- If all masters try their best to get packets from the same RX queue, the loading between masters is balanced automatically.



Navigator Components

- Hardware-based Queue Manager Sub-System (QMSS)
- Specialized Packet DMAs (PKTDMA)
  - NOTE: PKTDMA is commonly referenced in commands and code as CPPI (Communication Peripheral Port Interface)
  - Multiple PKTDMA instances reside within qualified Keystone peripherals (QMSS, SRIO, NETCP, AIF2, BCP)

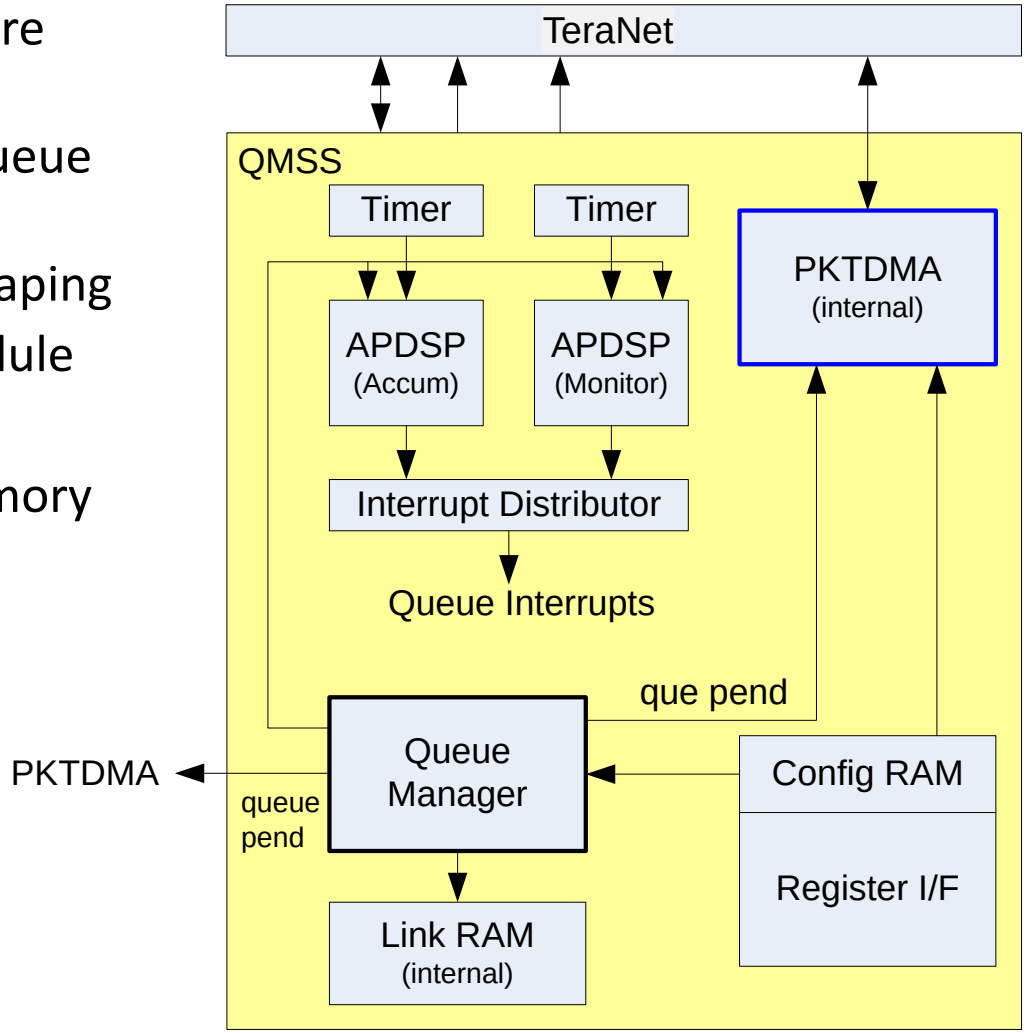
Multicore Navigator architecture



QMSS: Components Overview

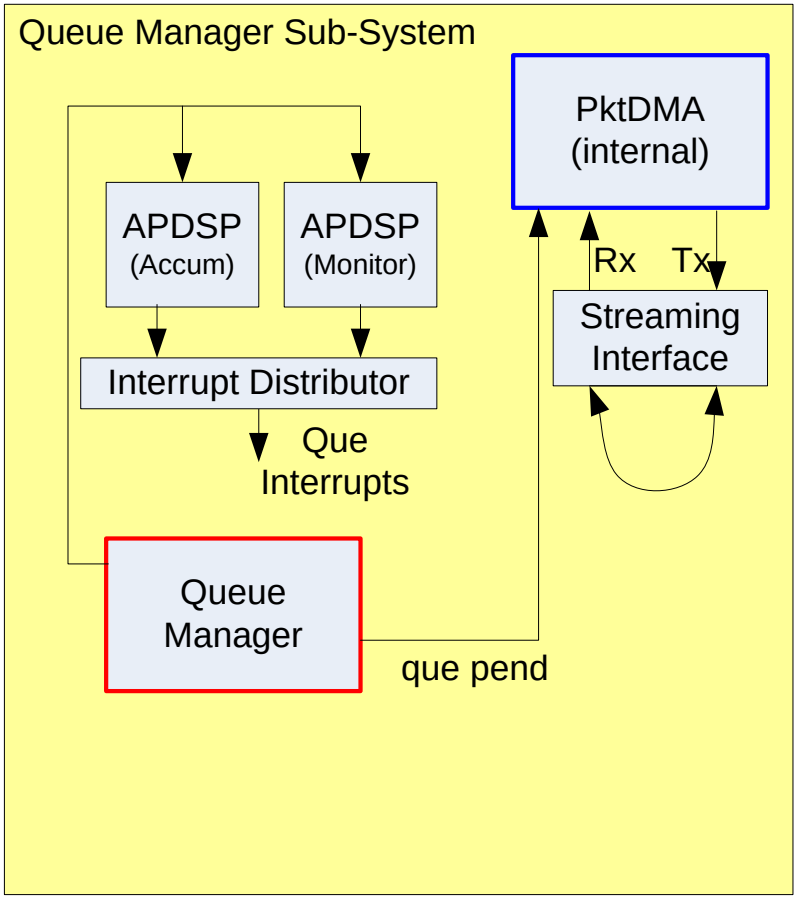
Major HW components of the QMSS:

- Queue Manager
- Two PDSPs (Packed Data Structure Processors):
  - Descriptor Accumulation / Queue Monitoring
  - Load Balancing and Traffic Shaping
- Interrupt Distributor (INTD) module
- Two timers
- Internal RAM for descriptor memory
- PKTDMA that supports all cores



Infrastructure Packet DMA

- The Rx and Tx Streaming I/F of the QMSS PKTDMA are wired together to enable loopback.
- Data packets sent out the Tx side are immediately received by the Rx side.
- This PKTDMA is used for core-to-core transfers and peripheral-to-DSP transfers.
- Because the DSP is often the recipient, a descriptor accumulator can be used to gather (pop) descriptors and interrupt the host with a list of descriptor addresses. The host must recycle them.



QMSS: Queues

- Queues are like a mailbox. Descriptors are pushed and popped to and from queues.
- Navigator transactions typically involve two queues:
  - The TX queue of the source
  - The RX queue of the destination
- There are 8192 queues within the QMSS (see mapping on next slide).
- Each queue can be either general purpose queue or associated with functionality.
- Queues associated with queue pending signals should not be used for general use, such as free descriptor queues (FDQs). Others can be used for any purpose.

QMSS: Queue Mapping

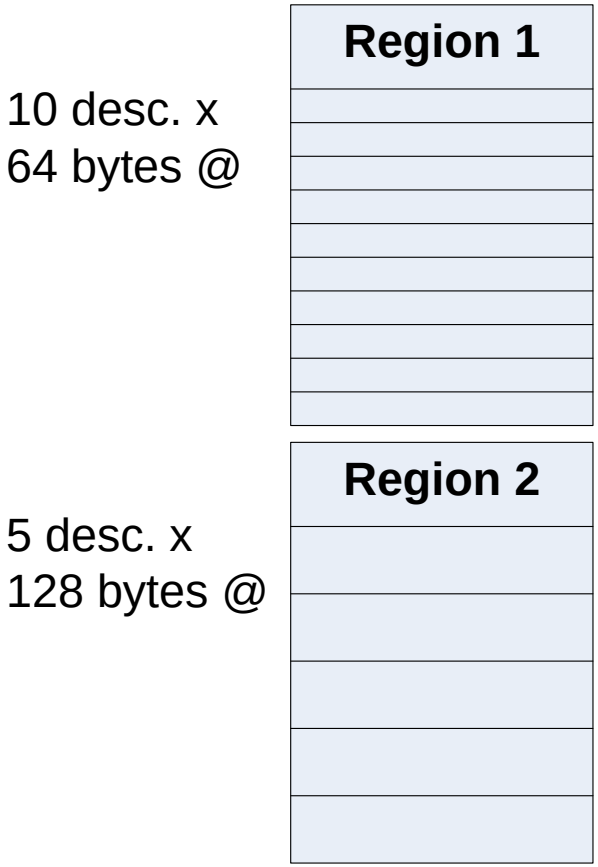
Queue Range	Count	Hardware Type	Purpose
0 to 511	512	pdsp/firmware	Low Priority Accumulation queues
512 to 639	128	queue pend	AIF2 Tx queues
640 to 651	12	queue pend	PA Tx queues (PA PKTDMA uses the first 9 only)
652 to 671	20	queue pend	CPintC0/intC1 auto-notification queues
672 to 687	16	queue pend	SRIO Tx queues
688 to 695	8	queue pend	FFTC_A and FFTC_B Tx queues (688..691 for FFTC_A)
696 to 703	8		General purpose
704 to 735	32	pdsp/firmware	High Priority Accumulation queues
736 to 799	64		Starvation counter queues
800 to 831	32	queue pend	QMSS Tx queues
832 to 863	32		Queues for traffic shaping (supported by specific firmware)
864 to 895	32	queue pend	vUSR queues for external chip connections
896 to 8191	7296		General Purpose

QMSS: Descriptors

- Descriptors are messages that move between queues and carry information and data.
- Descriptors are allocated in the memory region (see next slide).
- 20 memory regions are provided for descriptor storage (LL2, MSMC, DDR).
- 1 or 2 linking RAMs that (link list) index the descriptors (internal memory to QMSS or other memory)
- Up to 16K descriptors can be handled by internal Link RAM (Link RAM 0)
- Up to 512K descriptors can be supported in total.

QMSS: Descriptor Memory Regions

All Navigator descriptor memory regions are divided into *equal-sized* descriptors. For example:



Memory regions are *always* aligned to 16-byte boundaries and descriptors are *always* multiples of 16 bytes.

Linking RAM

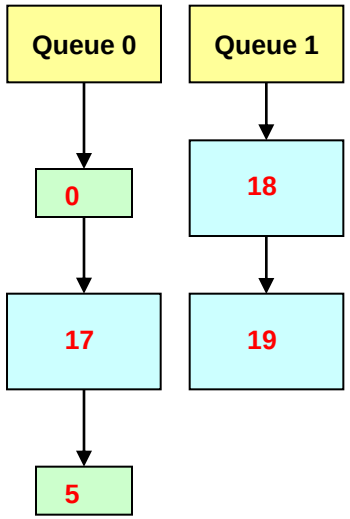
- Linking RAM contains 1 entry for each descriptor . Linking RAM entry is effectively an extension of the descriptor
- Linking RAM stores Forward data pointer that is critical for the PUSH / POP operations performed by the Queue Manager
- Linkage between physical address of descriptor and physical address of Linking RAM is performed inside the QM using information provided in the Queue Management configuration registers
- Linking RAM is typically placed in local memory for speed. This allows data elements to be linked and unlinked in a queue very quickly, even though the buffers themselves may be in external memory
- There is no limit to the length of a single queue, only a limit on the total number of data elements in the system.
- 2 configurable Linking RAM regions

Linking RAM

Forward Pointer Table

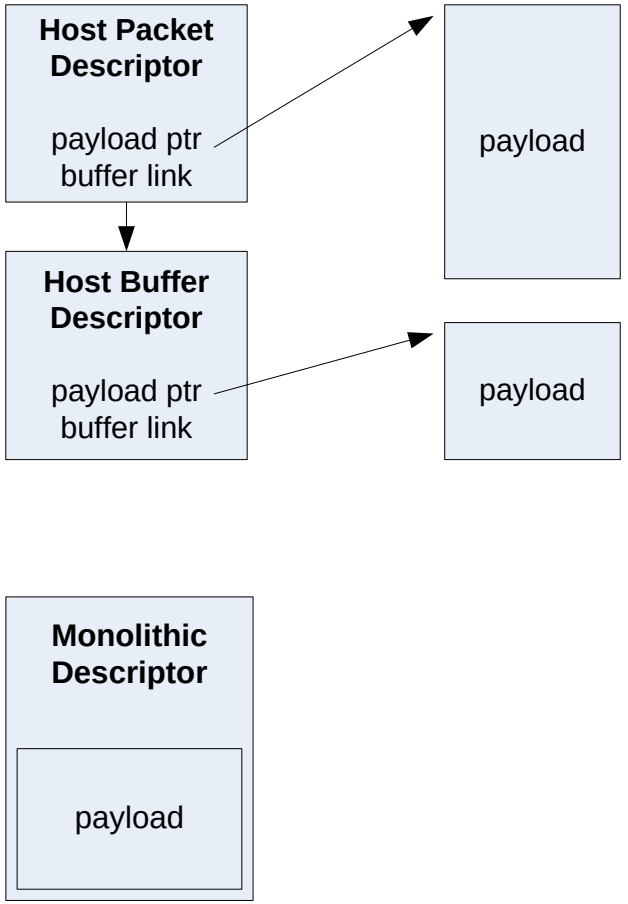
17	-	-	-
-	x	-	-
-	-	-	-
-	-	-	-
-	5	19	x

Queue Contents



QMSS: Descriptor Types

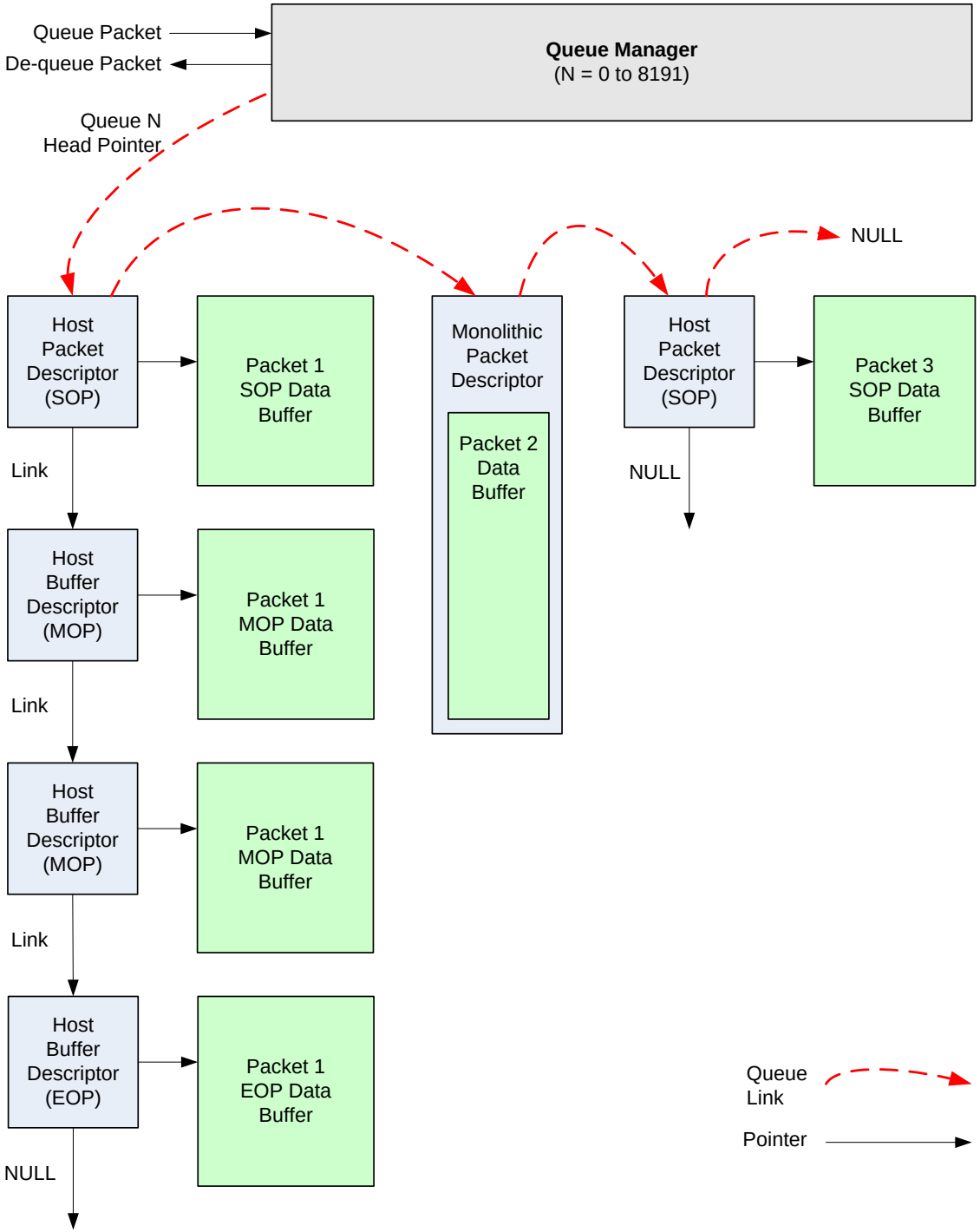
- Two descriptor types are used within Navigator:
  - Host type provide flexibility, but are more difficult to use:
    - Contains a header with a pointer to the payload.
    - Can be linked together (packet length is the sum of payload (buffer) sizes).
  - Monolithic type are less flexible, but easier to use:
    - Descriptor contains the header and payload.
    - Cannot be linked together.
    - All payload buffers are equally sized (per region).



Descriptor Queuing

This diagram shows several descriptors queued together. Things to note:

- Only the Host Packet is queued in a linked Host Descriptor.
- A Host Packet is always used at SOP, followed by zero or more Host Buffer types.
- Multiple descriptor types may be queued together, though not commonly done in practice.

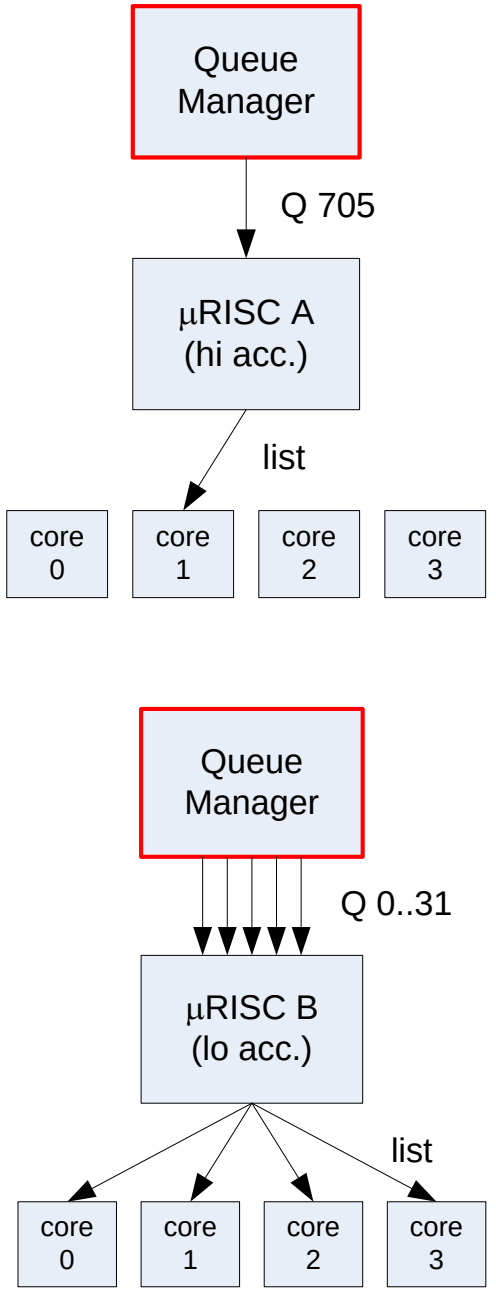


Descriptor Accumulators

- Accumulators keep polling queues.
- Run in background, interrupts core with list of popped descriptor addresses.
- Core software must recycle.

- High-Priority Accumulator:
  - 32 channels, one queue per channel
  - All channels scanned each timer tick (25us)
  - Each channel/event maps to 1 core
  - Programmable list size and options

- Low-Priority Accumulator:
  - 16 channels, up to 32 queues per channel
  - 1 channel scanned each timer tick (25 us)
  - Each channel/event maps to all cores
  - Programmable list size and options



Packet DMA (PKTDMA)

Major components for each instance:

- Multiple RX DMA channels
- Multiple TX DMA channels
- Multiple RX flow channels. RX flow defines behavior of the receive side of the navigator.

Packet DMA (PKTDMA) Features

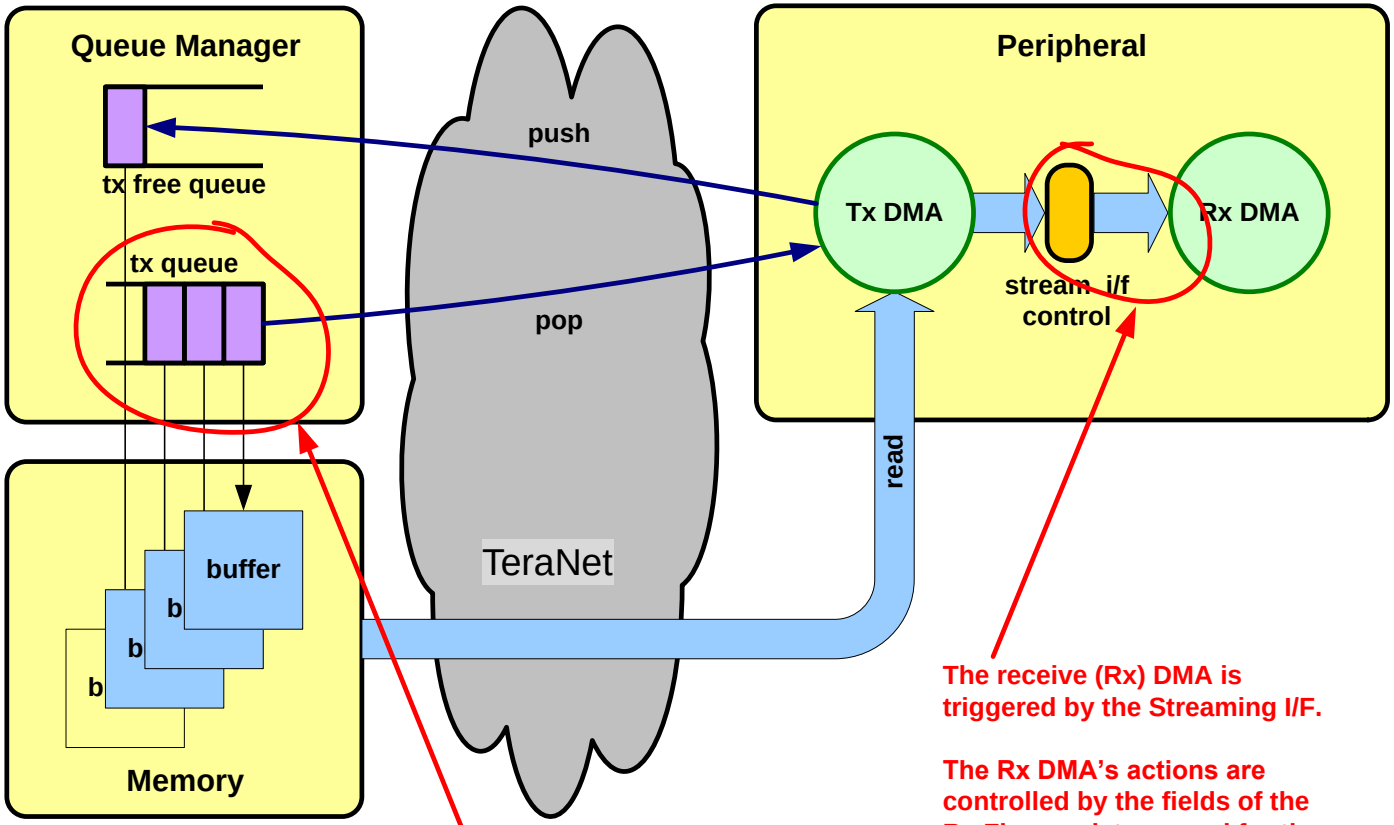
- Independent Rx and Tx modules:
  - Tx module:
    - Tx channel triggering via hardware qpend signals from QM.
    - Tx control is programmed via descriptors.
    - 4 level priority (round robin) Tx Scheduler
      - Additional Tx Scheduler Interface for AIF2 (wireless applications only)
  - Rx module:
    - Rx channel triggering via Rx Streaming I/F.
    - Rx control is programmed via an “Rx Flow” (more later)
- 2x128 bit symmetrical Streaming I/F for Tx output and Rx input
  - Wired together for loopback within the QMSS PKTDMA instance.
  - Connects to matching streaming I/F (Tx->Rx, Rx->Tx) of peripheral
- Packet-based, so neither the Rx or Tx care about payload format.

Agenda

1. Multicore Navigator Architecture Overview
   a. Queue Manager Subsystem (QMSS)
   b. Packet DMA (PKTDMA)
2. **Working Together**
3. Configuration

Packet DMA Control

Understanding how the PKTDMAs are triggered and controlled is critical.



The transmit (Tx) DMA is triggered by a DSP task or other hardware pushing to a Tx queue.

The Tx DMA's actions are controlled by the fields of the descriptor.

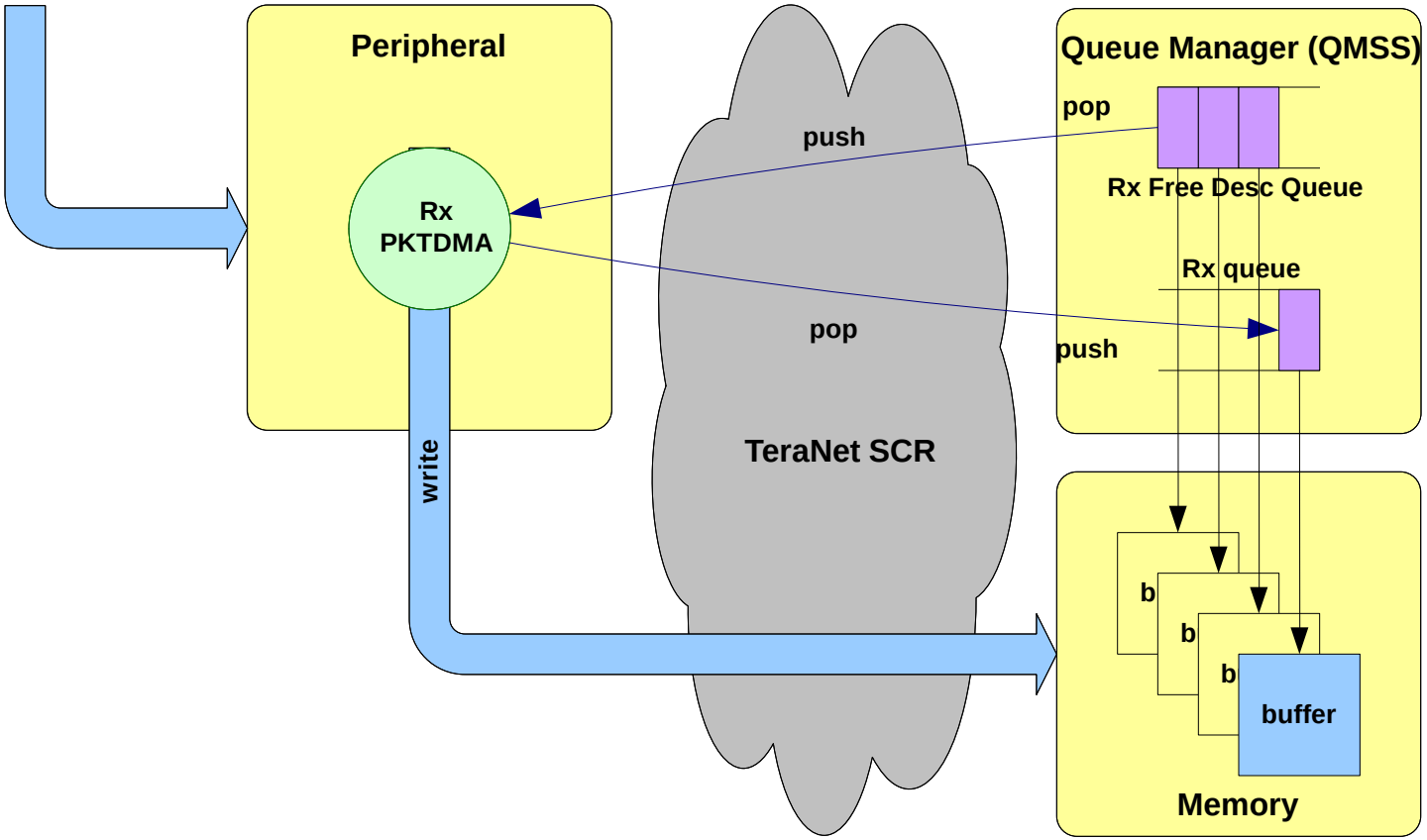
The receive (Rx) DMA is triggered by the Streaming I/F.

The Rx DMA's actions are controlled by the fields of the Rx Flow registers used for the packet being sent.

In Infrastructure (loopback) mode, certain fields in the Tx descriptor drive the Rx DMA.

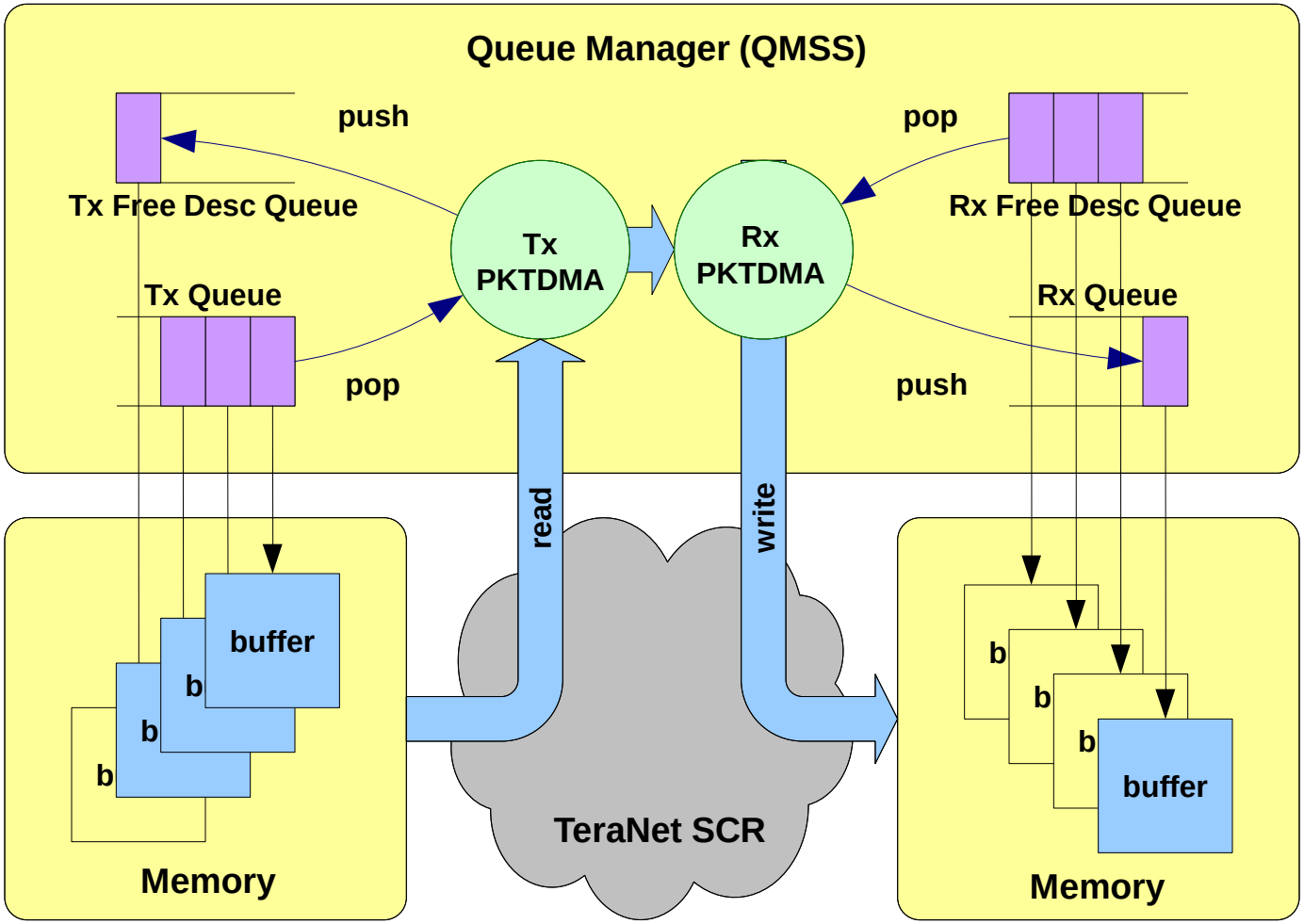
Receive Example

- Rx PKTDMA receives packet data from Rx Streaming I/F.
- Using an Rx Flow, the Rx PKTDMA pops an Rx FDQ.
- Data packets are written out to the descriptor buffer.
- When complete, Rx PKTDMA pushes the finished descriptor to the indicated Rx queue.
- The core that receives the descriptor must recycle the descriptor back to an Rx FDQ.



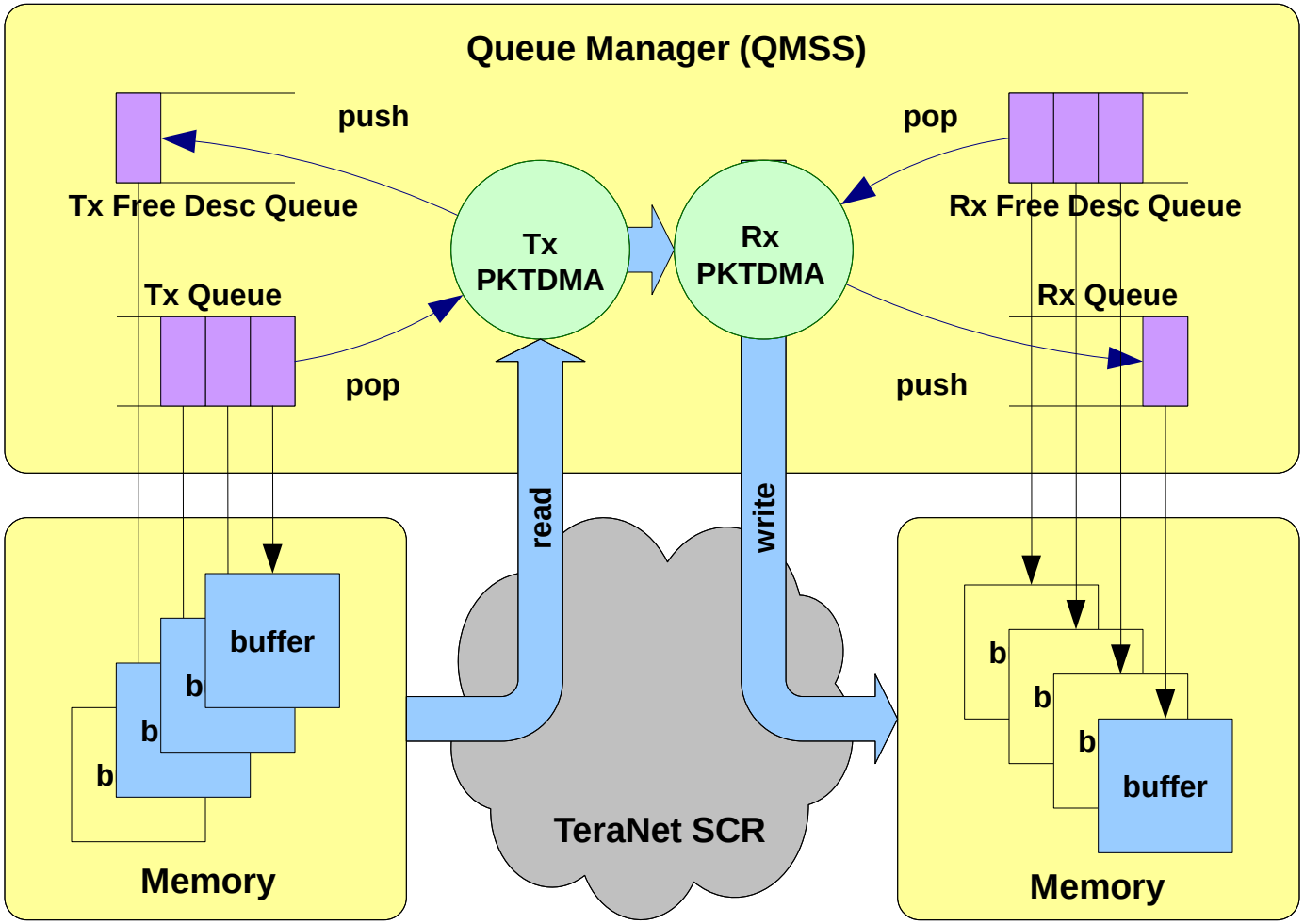
Core-to-Core (Infrastructure) Example 1/2

- The DSP (or a peripheral) pushes a descriptor onto a Tx queue of the QMSS PKTDMA.
- The Tx PKTDMA pops the descriptor, sends the data out the Streaming I/F, and recycles the descriptor.
- The Rx PKTDMA is triggered by the incoming Streaming I/F data and pops an Rx FDQ.



Core-to-Core (Infrastructure) Example 2/2

- The Rx PKTDMA then pushes the finished descriptor to an Rx queue.
- If the Rx queue is an Accumulation queue, the accumulator pops queue and eventually interrupts the DSP with the accumulated list.
- The destination DSP consumes the descriptors and pushes them back to an Rx FDQ.



How Does it Work During Run Time?

For example, Core A wants to send a message to Core B.

- Core A picks available descriptor (e.g., message structure) that is either partially or completely pre-built.
  - As needed, Core A adds missing information.
- Core A pushes the descriptor into a queue.
  - At this point, Core A is done.
- The Navigator processes the message and sends it to a queue in the receive side of Core B where it follows a set of pre-defined instructions (Rx flow), such as:
  - Interrupt Core B and tell it to process the message.
  - Set a flag so Core B can pull and change a flag value on which Core B synchronizes.
  - Move buffer into Core B memory space and interrupt the core.
- After usage, the receive core recycles the descriptors (and any buffer associated with) to prevent memory leaks.

Agenda

1. Multicore Navigator Architecture Overview
    a. Queue Manager Subsystem (QMSS)
    b. Packet DMA (PKTDMA)
2. Working Together
3. **Configuration**

What Needs to Be Configured?

- **Link Ram -** Up to two LINK-RAM
  - One internal, Region 0, address 0x0008 0000, size up to 16K
  - One External, global memory, size up to 512K
- **Memory Regions** - Where descriptors actually reside
  - Up to 20 regions, 16 byte alignment
  - Descriptor size is multiple of 16 bytes, minimum 32
  - Descriptor count (per region) is power of 2, minimum 32
  - Configuration – base address, start index in the LINK RAM, size and number of descriptors
  - **The way the region is managed**
- **Loading PDSP firmware**

What Needs to Be Configured?

- **Descriptors**
  - Create and initialize.
  - Allocate data buffers and associate them with descriptors.
- **Queues**
  - Initialize FDQ.
  - Configure transmit and receive queues.
- **PKTDMA**
  - Configure all PKTDMA in the system.
  - Define receive flows.
  - Special configuration information used for PKTDMA.