AAL TRANSPORT ABSTRACTION

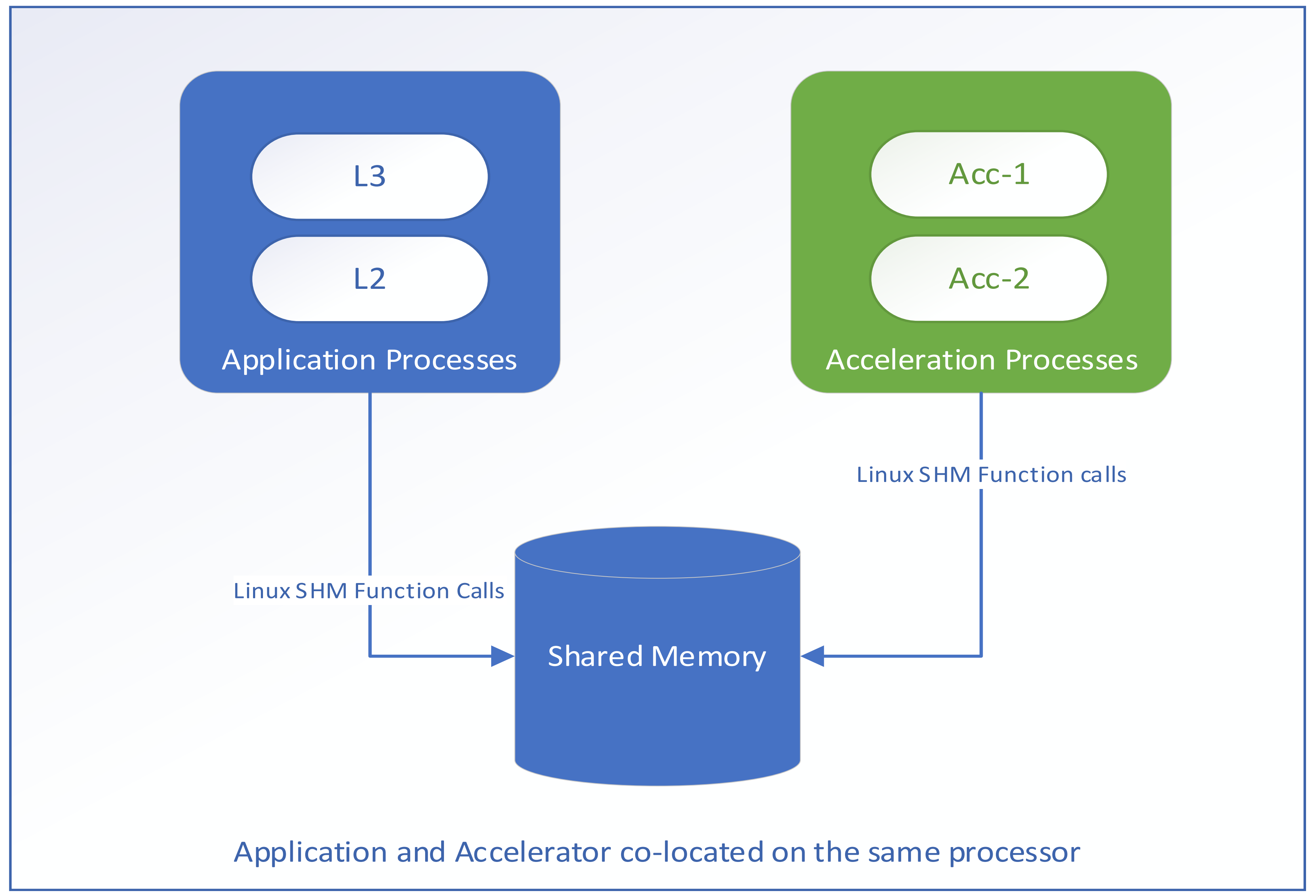
DECEMBER 16 , 2021

WHY ?

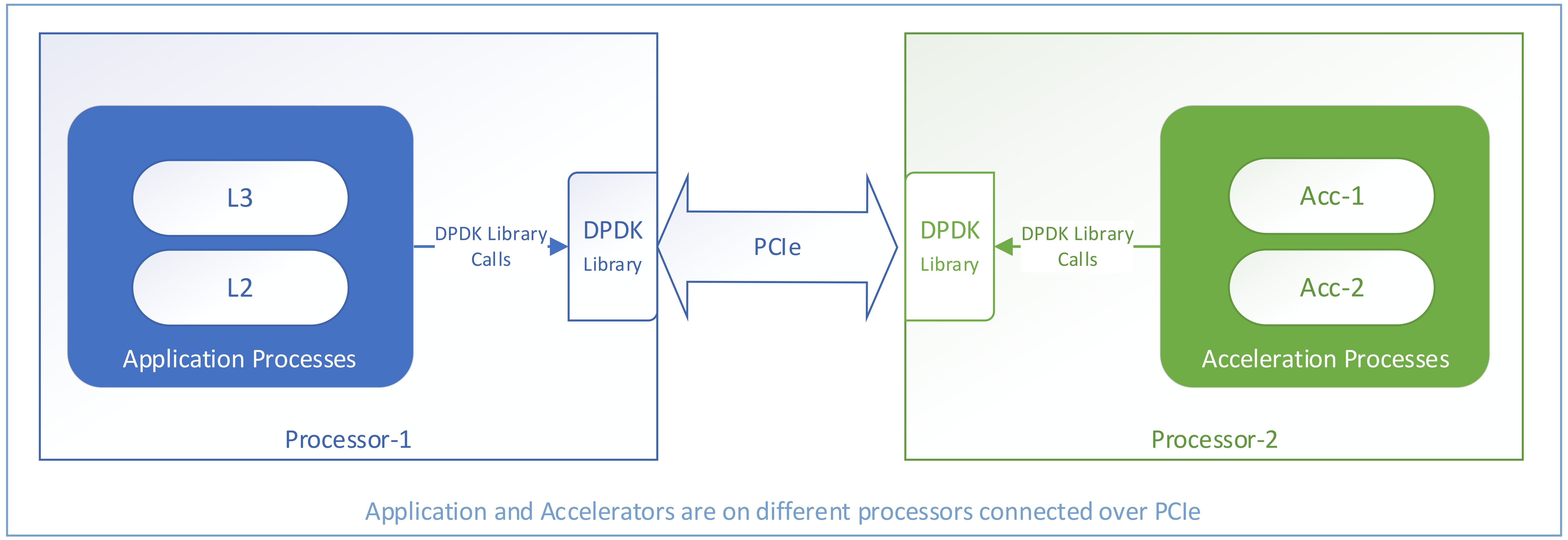
RAN is not a plug and play solution yet

- FAPI has standardized contents of L2-L1C communication
- Still dedicated effort required for each flavor of L2-L1 integration
- Issues addressed during integration - Timing & Transport
- *Transport can be easily addressed*

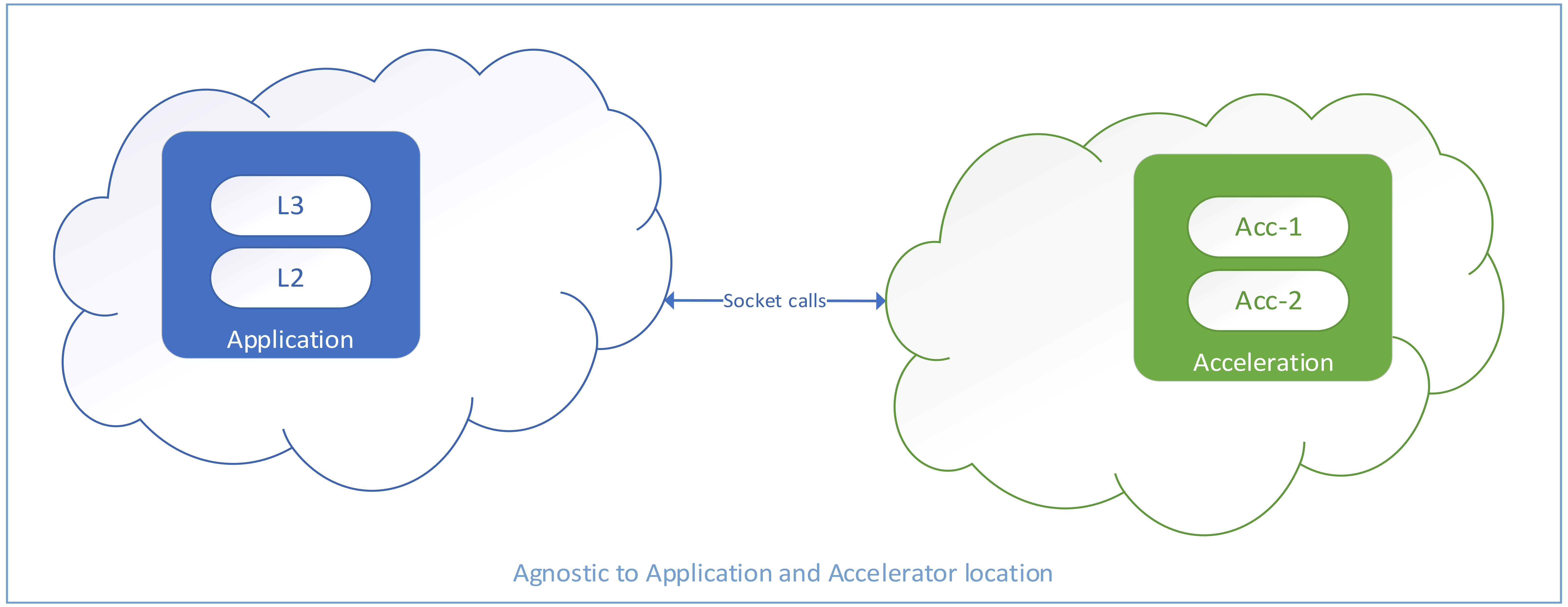
WHY ?



Application and Accelerator co-located on the same processor

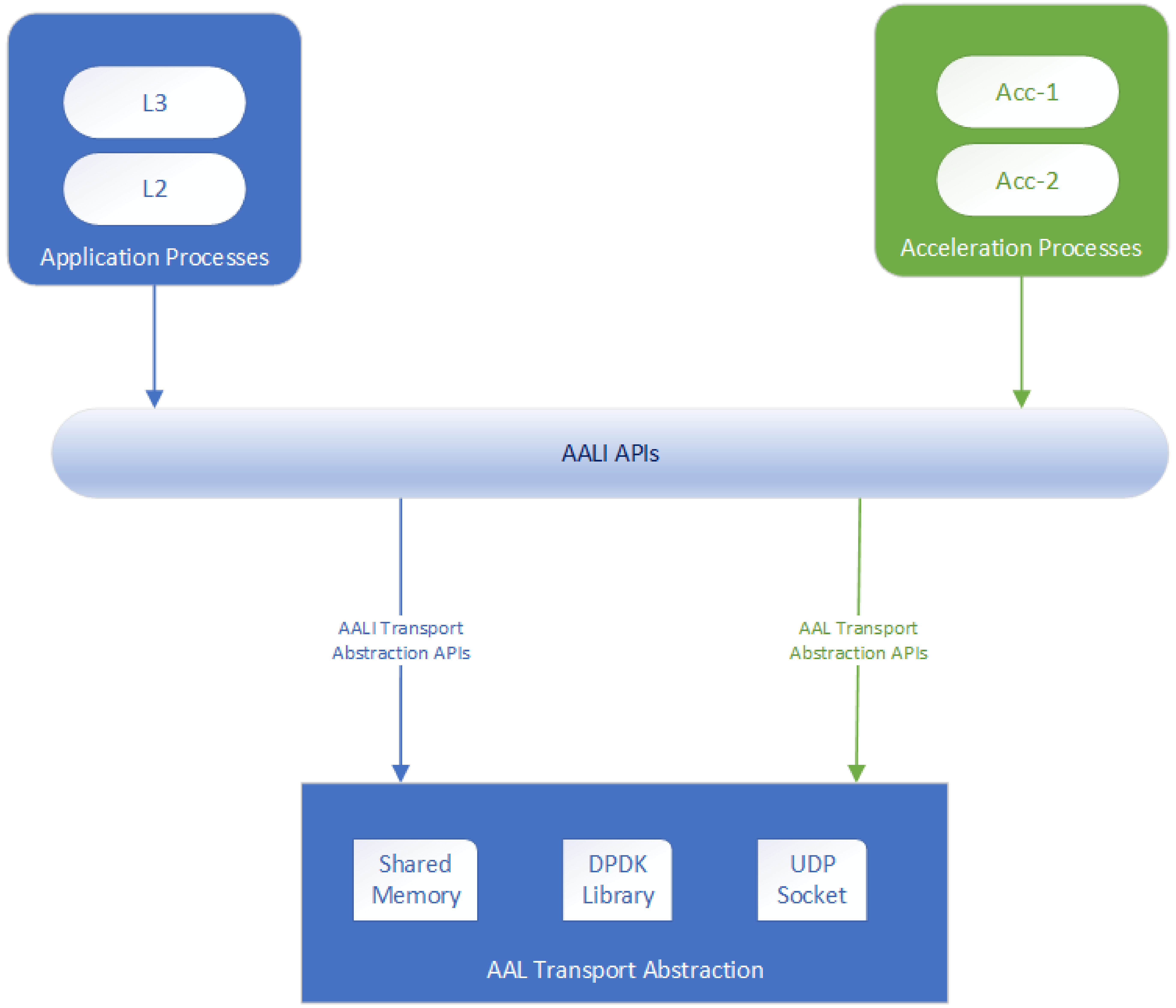


Application and Accelerators are on different processors connected over PCIe



Agnostic to Application and Accelerator location

WHAT ?


L3
L2
Application Processes
Acc-1
Acc-2
Acceleration Processes
AALI APIs
AALI Transport Abstraction APIs
AAL Transport Abstraction APIs
Shared Memory
DPDK Library
UDP Socket
AAL Transport Abstraction

WHAT ?

- AAL Transport Abstraction standardizes the transport interface for different kinds of transport
  - Interface API remains constant while the implementation of API changes for different transport
- L1 to publish 2 profiles
  - AAL profiles for inline acceleration
  - AAL Transport profiles for communication
- Application publishes transport profiles for communication as well
- Capability negotiation happens at Orchestration level

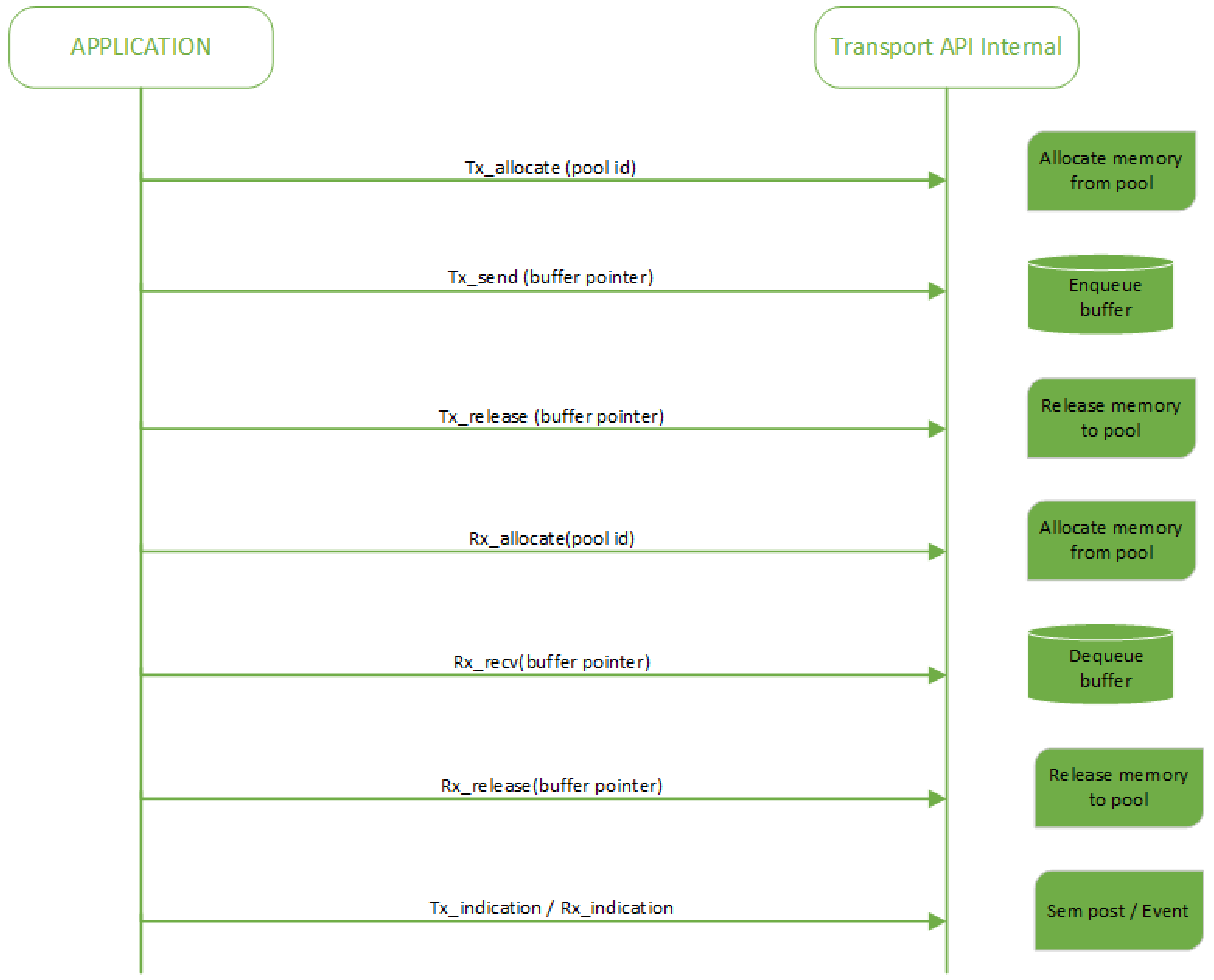
HOW ?

- Types of transport
  - Shared Memory (Same processor)
  - DMA (Different processors connected over PCIe interface)
  - Ethernet (Agnostic of application location)
- APIs for configuration
  - Shared Memory / DMA / UDP configuration
  - Memory Pool for Data Messages (TB), Config Messages (Slot level)
- APIs for message/data exchange
  - Buffer allocate/deallocate APIs
  - Send/Receive APIs
  - Indication APIs

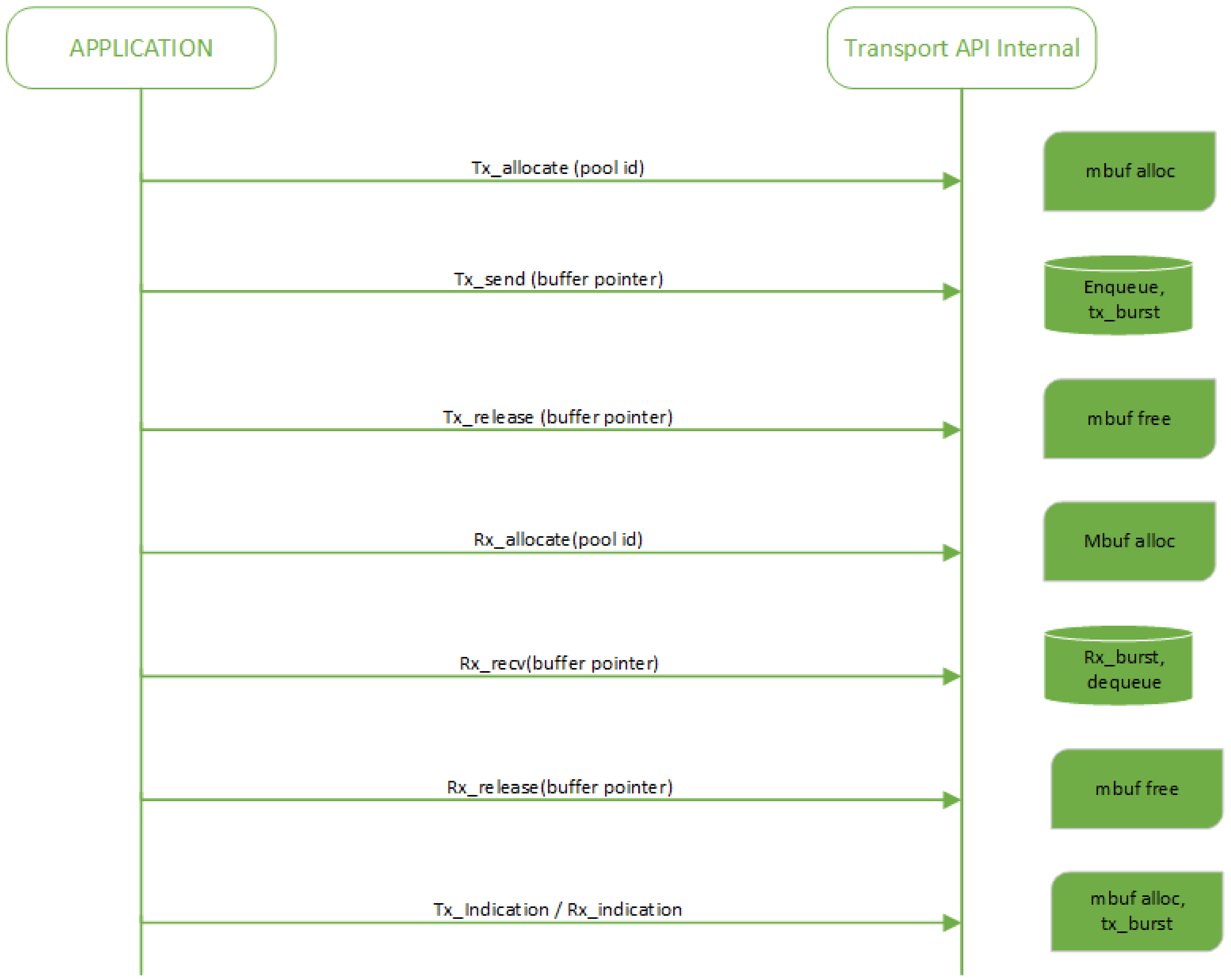
APIS FOR MESSAGE/DATA EXCHANGE

Function	Description
tx_allocate	Input – pool-id Output – buffer pointer Returns pointer to a buffer allocated from the specified pool. Initializes reference count to 1 on the buffer
tx_send	Input – pointer to the buffer Sends the buffer according to the configured underlying transport. Transfer of ownership of buffer from Application to AAL Transport Abstraction. Increments reference count on the buffer. AAL Transport Abstraction calls tx_release on the buffer after data has been sent.
tx_release	Input – pointer to the buffer Decrements reference count and releases the allocated buffer back to its pool if reference count is 0
rx_allocate	Input – pool-id Output – buffer pointer Returns a pointer to a buffer allocated from the specified pool. Initializes reference count to 1 on the buffer. This primitive is optional to support.
rx_recv	Input – pointer to the buffer. If NULL, AAL Transport Abstraction allocates the buffer. Receives data into the buffer according to the configured underlying transport. Transfer of ownership from AAL Transport Abstraction to Application.
rx_release	Input – pointer to the buffer Decrements reference count and releases the allocated buffer back to its pool if reference count is 0
tx_indication	Provides a mechanism to send a signal / TTI indication
rx_indication	Provides a mechanism to receive signal / TTI indication

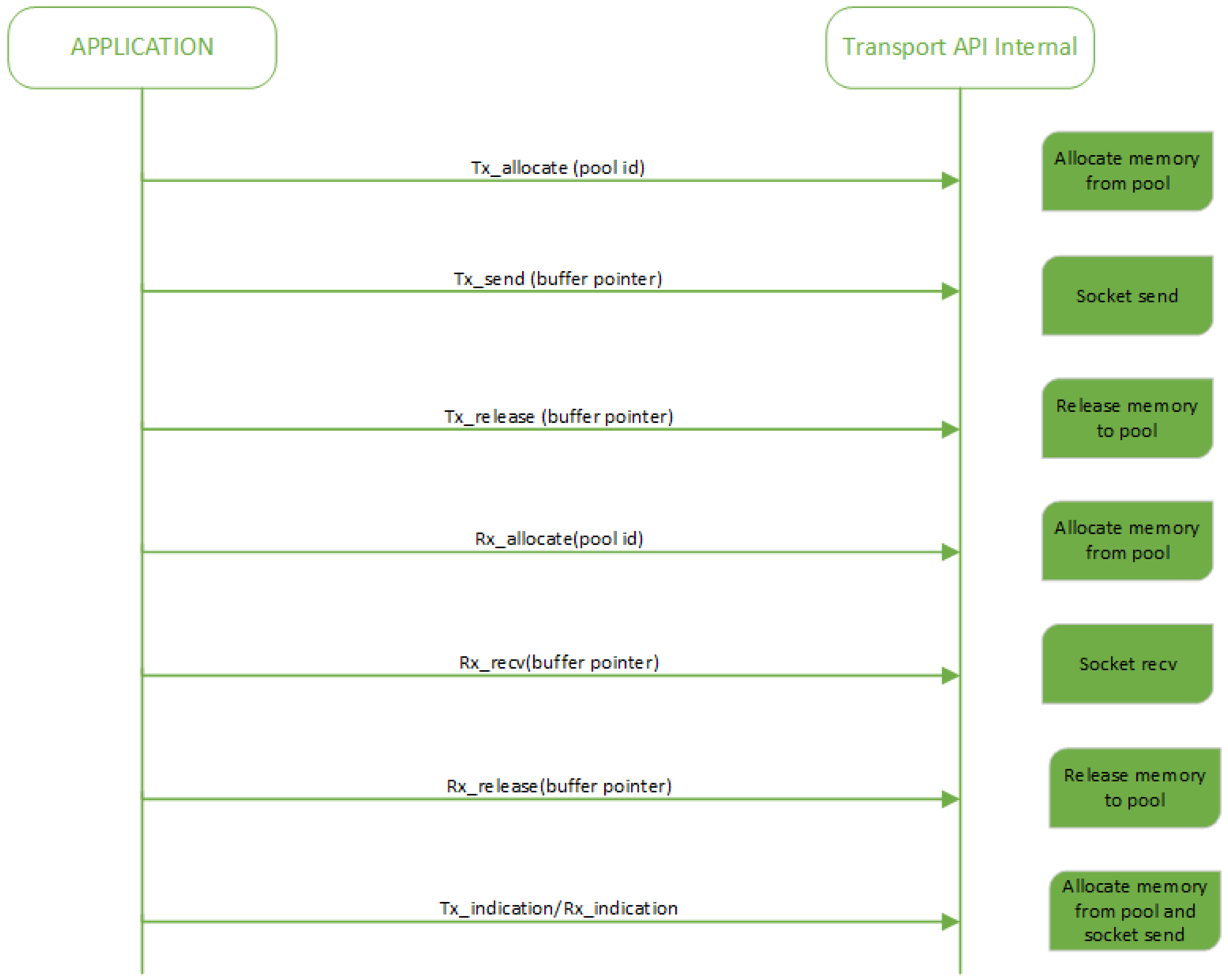
IPC OVER SHARED MEMORY



DMA USING DPDK



IPC OVER UDP


APPLICATION
Transport API Internal
Tx_allocate (pool id)
Allocate memory from pool
Tx_send (buffer pointer)
Socket send
Tx_release (buffer pointer)
Release memory to pool
Rx_allocate(pool id)
Allocate memory from pool
Rx_recv(buffer pointer)
Socket recv
Rx_release(buffer pointer)
Release memory to pool
Tx_indication/Rx_indication
Allocate memory from pool and socket send

NVIDIA®