Designing an AMBA-based SoC with a PCI Express Interface

Paul Cassidy

R&D Engineer, Staff

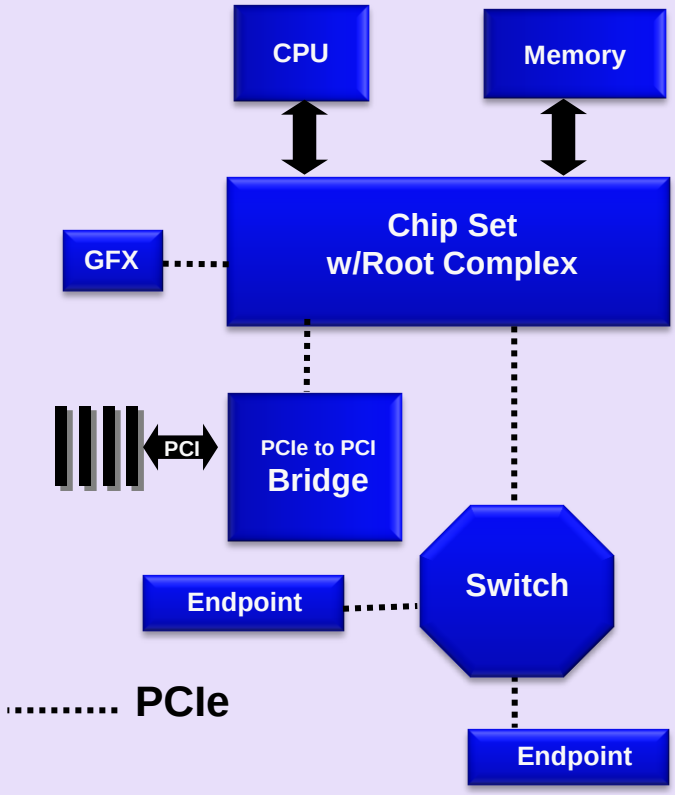
Disclaimer

- All opinions, judgments, recommendations, etc. that are presented herein are the opinions of the presenter of the material and do not necessarily reflect the opinions of the PCI-SIG®.
- The material included in this presentation reflects current thinking of the presenter and should not be evaluated as direction from the PCI-SIG®.

Agenda

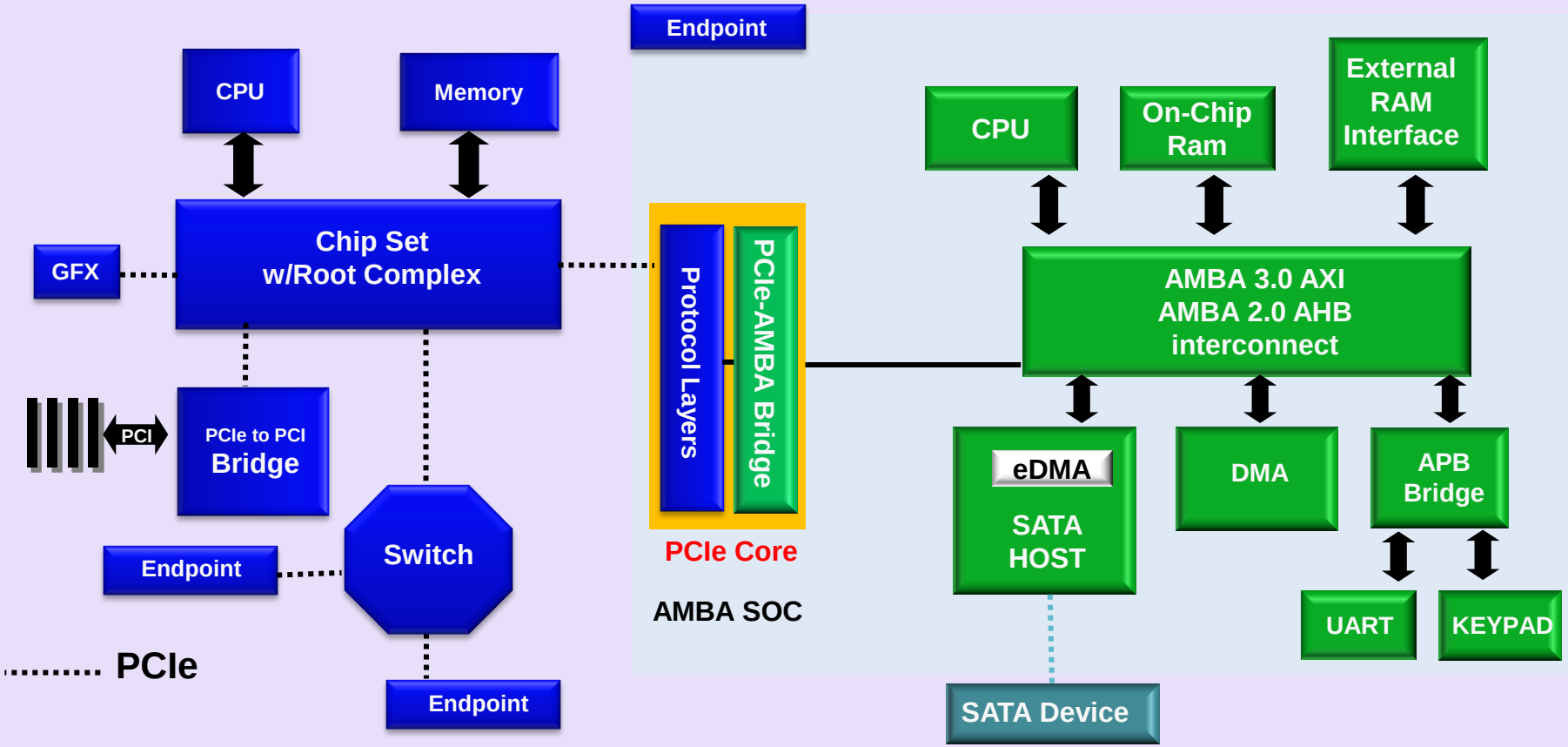
- Introduction
- Protocol Translation
- Data Flows
- Performance
- Summary

Adding AMBA System as PCIe Endpoint



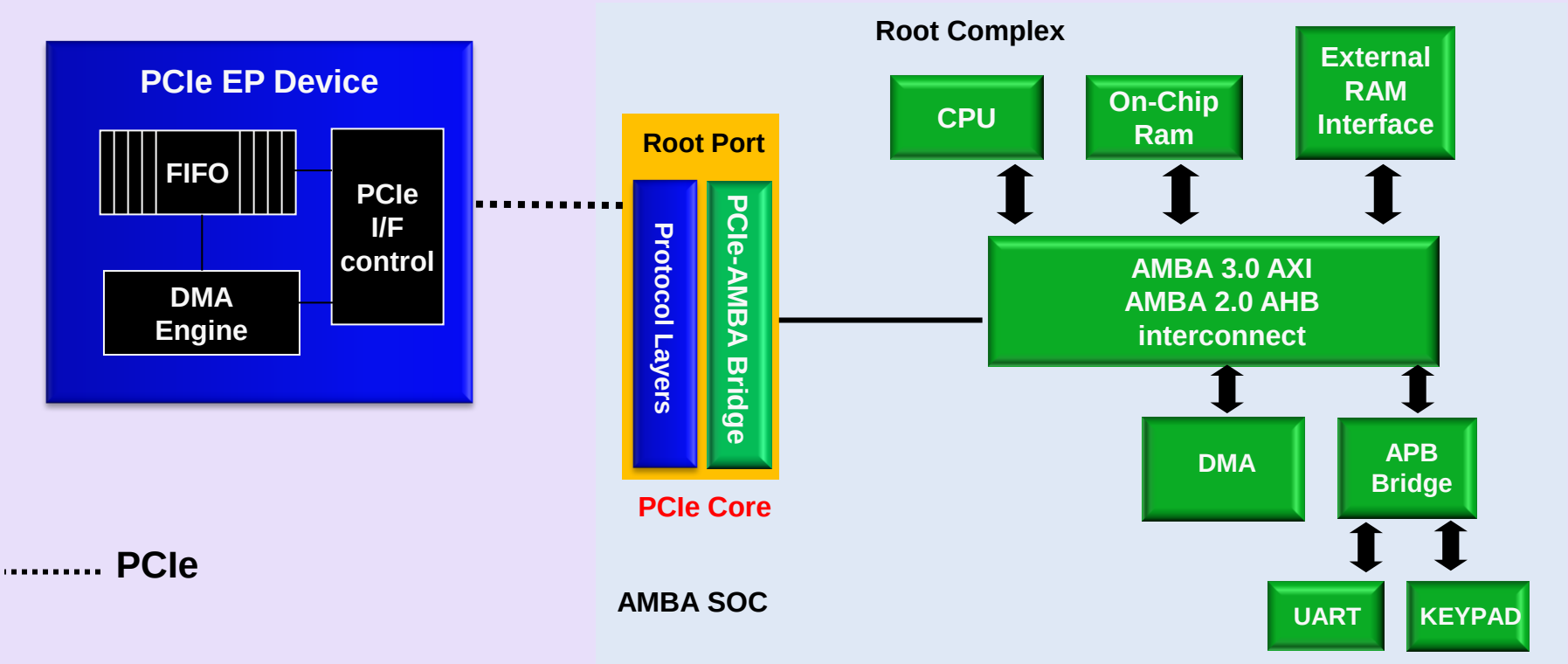
- Root Complex
  - ✓ **Connects CPU/Memory to I/O**
  - ✓ **Configures PCI Express links**
- Switch
  - ✓ **Provides fanout**
- Bridge
  - ✓ **Connects PCI Express and PCI/PCI-X**
- Endpoint
  - ✓ **Requester or Completer of PCI Express transactions**
  - ✓ **Connects PCI Express to end-user application**

Adding AMBA System as PCIe Endpoint



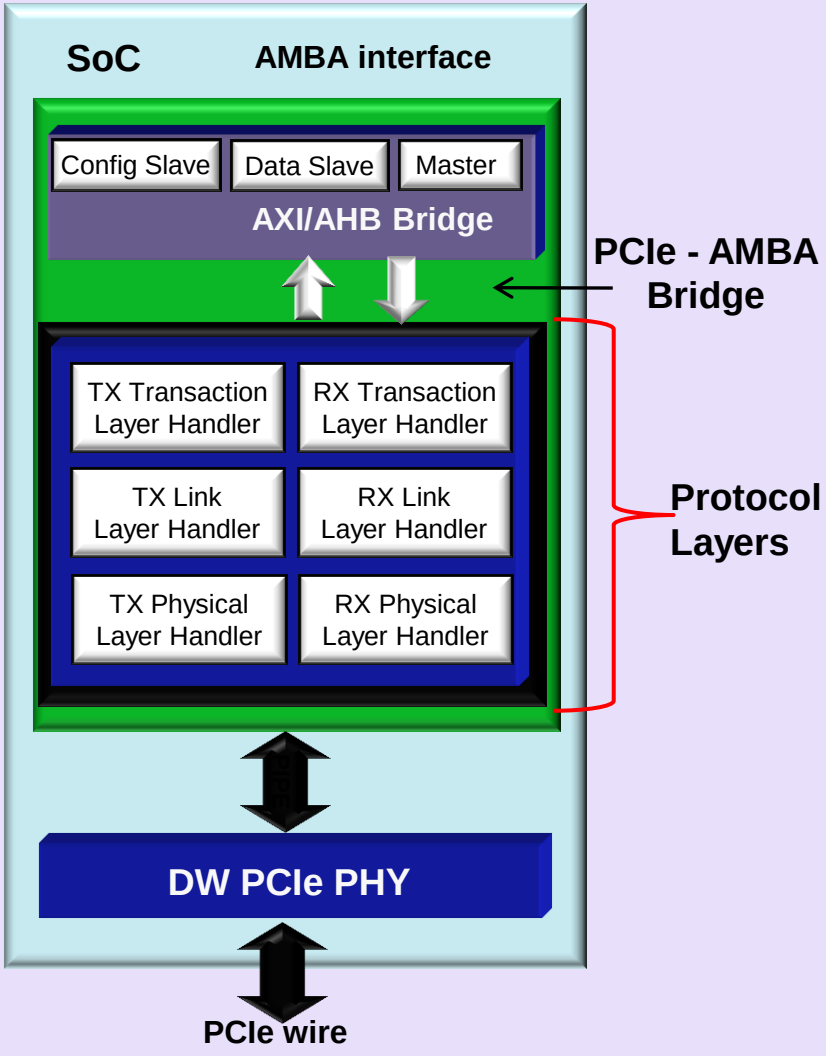
- Allows PCI Express endpoint devices, implemented in AMBA SoC to be connected to PCIe hierarchy
- Bridge, abstracts out the PCIe protocol and provides standard AMBA interface
- To system CPU, AMBA appears as endpoint device
- To SoC CPU, PCIe provides a view of system memory

Adding PCIe Endpoint to a Root Complex AMBA System



- Allows PCI Express hierarchy to be connected to AMBA Root Complex SOC

PCIe Protocol Layers



- **Physical Layer Handler**
  - ✓ Link Training/negotiation
  - ✓ Receive de-skewing, Descrambling
  - ✓ Special sequence detection/transmission
  - ✓ TS1/TS2/Skip/Electrical Idle/Fast Training
  - ✓ Packet Discovery/De-framing
- **Link Layer Handler**
  - ✓ LCRC checking/Generation
  - ✓ TLP sequence checking
  - ✓ Ack-Nack replay protocol
  - ✓ DLLP packet generation/reception
- **Transaction Layer Handler**
  - ✓ BAR matching
  - ✓ TLP filtering rules
  - ✓ Flow control protocol
  - ✓ ECRC checking/generation
  - ✓ Credit checking
  - ✓ TLP assembly/disassembly
  - ✓ IDO, TLP processing hints, LTR, OBFF etc

Terminology 1

- **Inbound Request/Outbound Completions**

  PCIe transactions that enter the PCIe core from the PCIe wire. Requests are driven onto AMBA bus by PCIe core AMBA bus master. Outbound completions from the targeted AMBA application slave are driven onto the PCIe core AMBA master response bus onto the PCIe wire.

- **Outbound Request/Inbound Completions**

  AMBA transactions that are driven by application AMBA masters to PCIe AMBA slave which are driven onto the PCIe wire. Inbound completions enter the core from the wire side and are driven onto AMBA slave response bus by the PCIe core.

Terminology 2

- **Maximum Transaction Unit**
  - ✓ **PCIe MTU or Max_Payload_Size:** Enumeration maximum data payload that PCIe core can generate and is expected to receive
    - PCIe core advertises its capability linked to buffering depths
    - Range is 128 bytes to 4KB
  - ✓ **AMBA MTU**: Maximum burst length by the AMBA bus width e.g. 64 x 4
    - AMBA MTU is typically less than or equal to PCIe MTU
- **PCIe Maximum Read Request Size**
  - ✓ Maximum read request size that PCIe core can receive from the wire
    - Range is 128 bytes to 4KB
    - PCIe MTU can be configured to be different than PCIe Maximum read request size
    - PCIe core does NOT advertise a capability

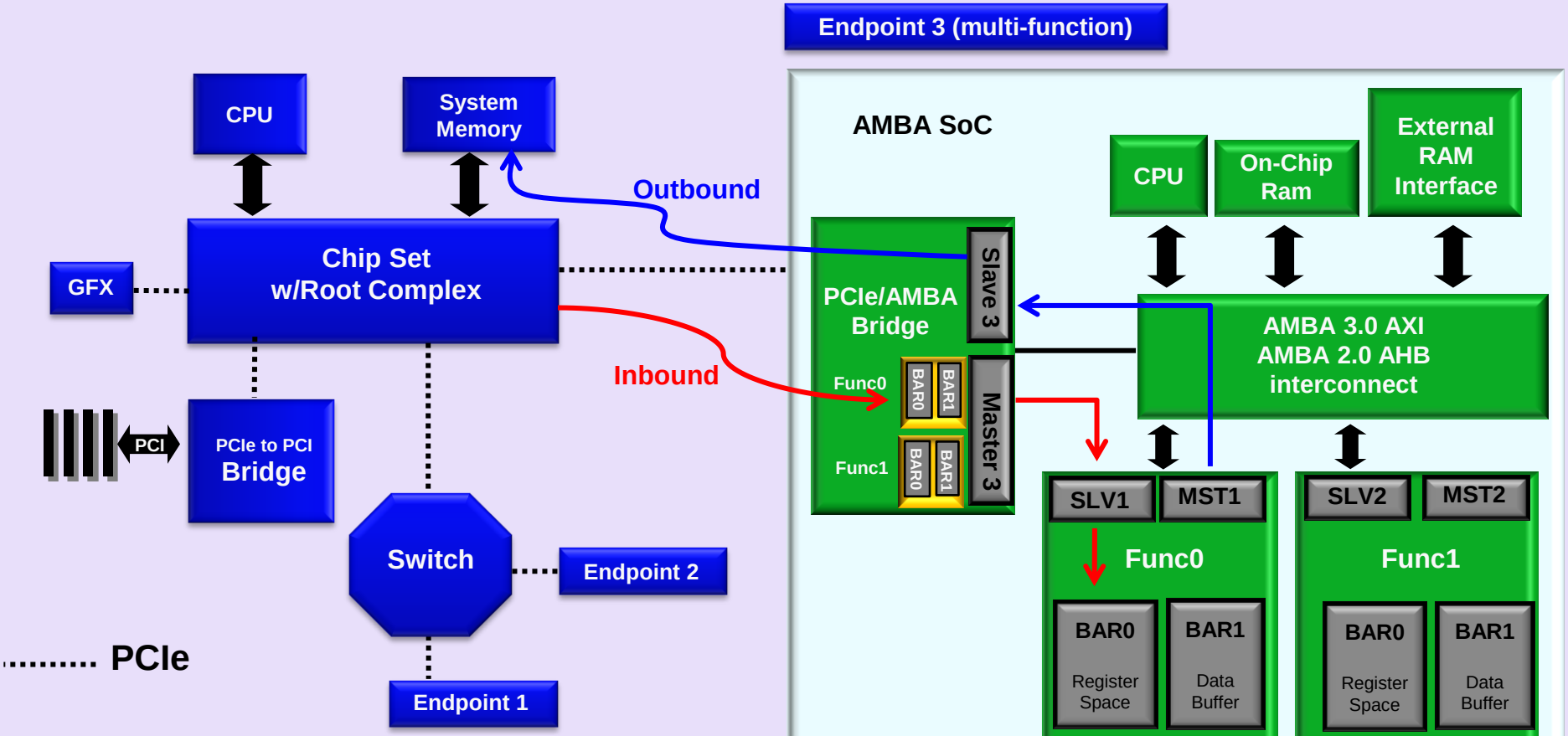
Agenda

- Introduction
- **Protocol Translation**
- Data Flows
- Performance
- Summary

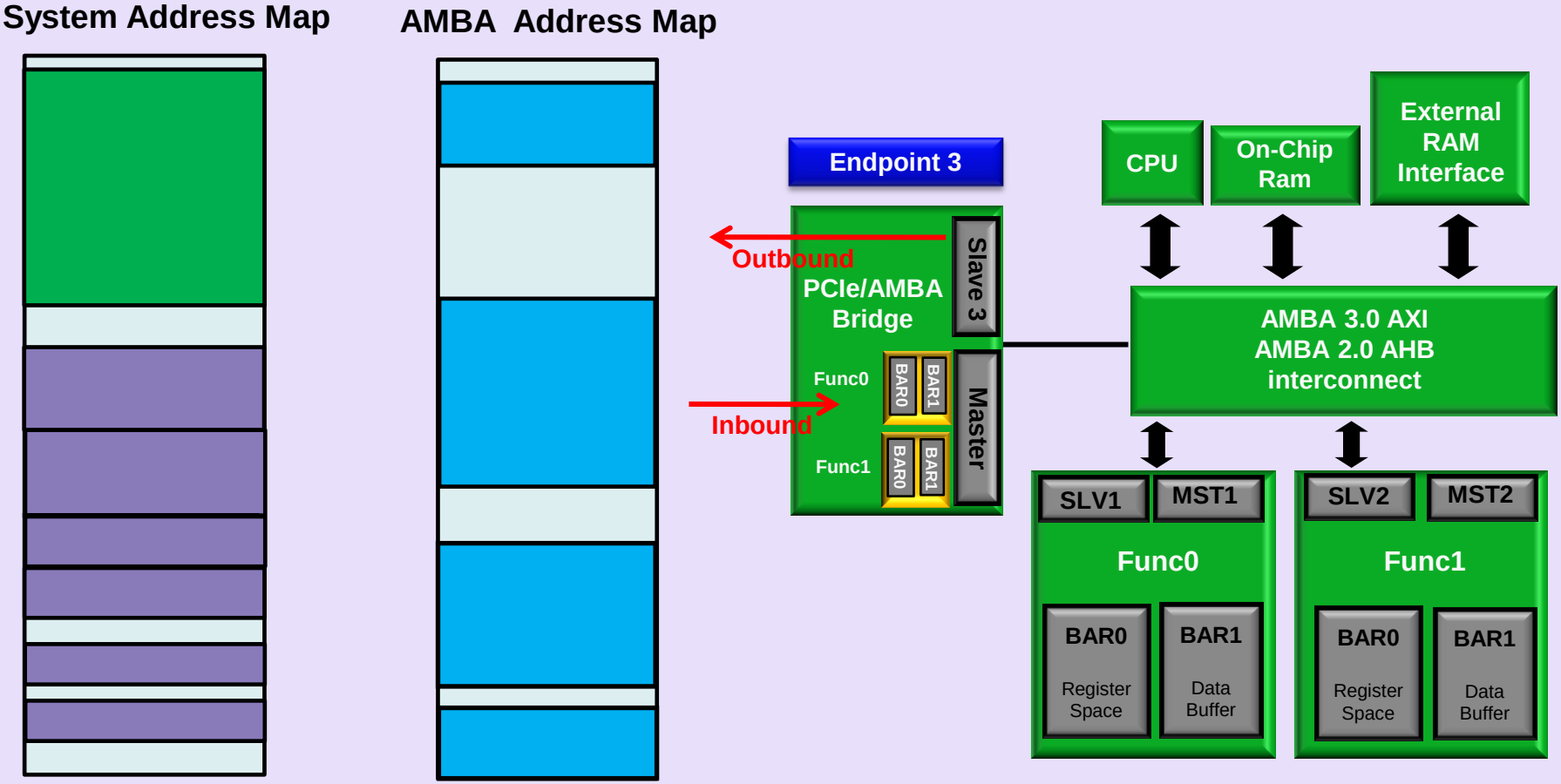
Protocol Translation Considerations

- PCIe protocol has multiple address spaces with different TLP types targeting each
  - ✓ Uses memory mapped addresses to target different entities in the PCIe hierarchy
  - ✓ Can have separate IO address space
  - ✓ Uses configuration to setup the system environment
- PCIe uses messaging for legacy interrupts, supports vendor defined messages
- PCIe addressing is dynamic assigned at enumeration
- AMBA has a single **fixed** address space with a single burst type
- Header attributes
  - ✓ PCIe has many other header attributes that need translation between PCIe and AMBA domains e.g. function number, traffic class, message code etc

Example PCIe Hierarchy with AMBA Endpoint

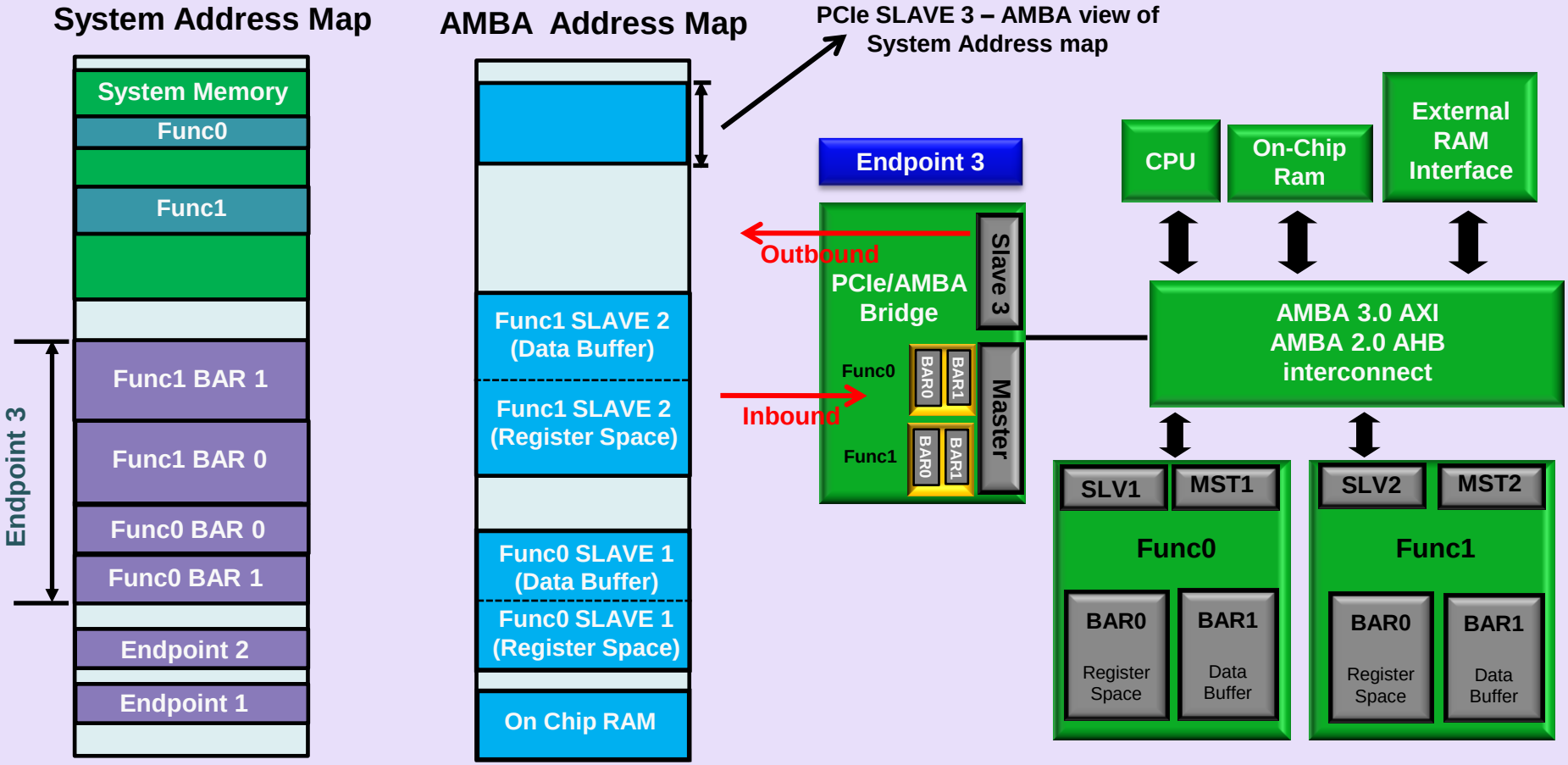


PCIe ⟷ AMBA Address Mapping



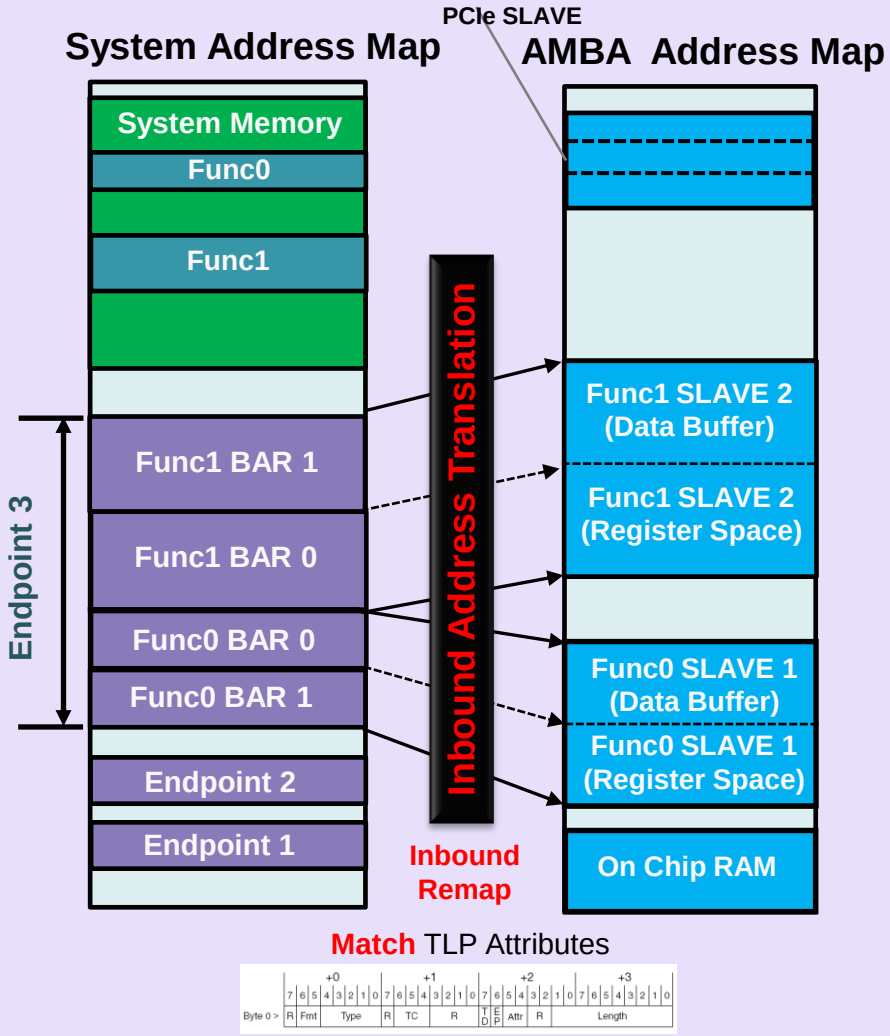
BAR = Base Address Register

PCIe ⟷ AMBA Address Mapping



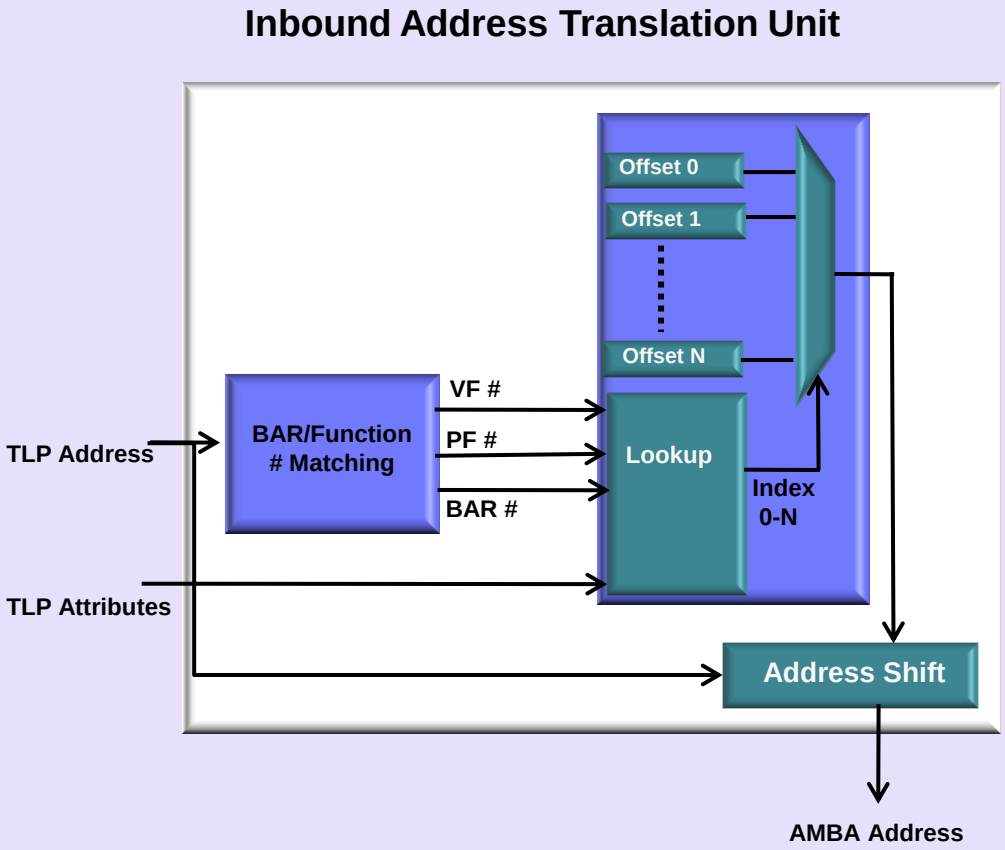
BAR = Base Address Register

Inbound PCIe to AMBA Address Mapping



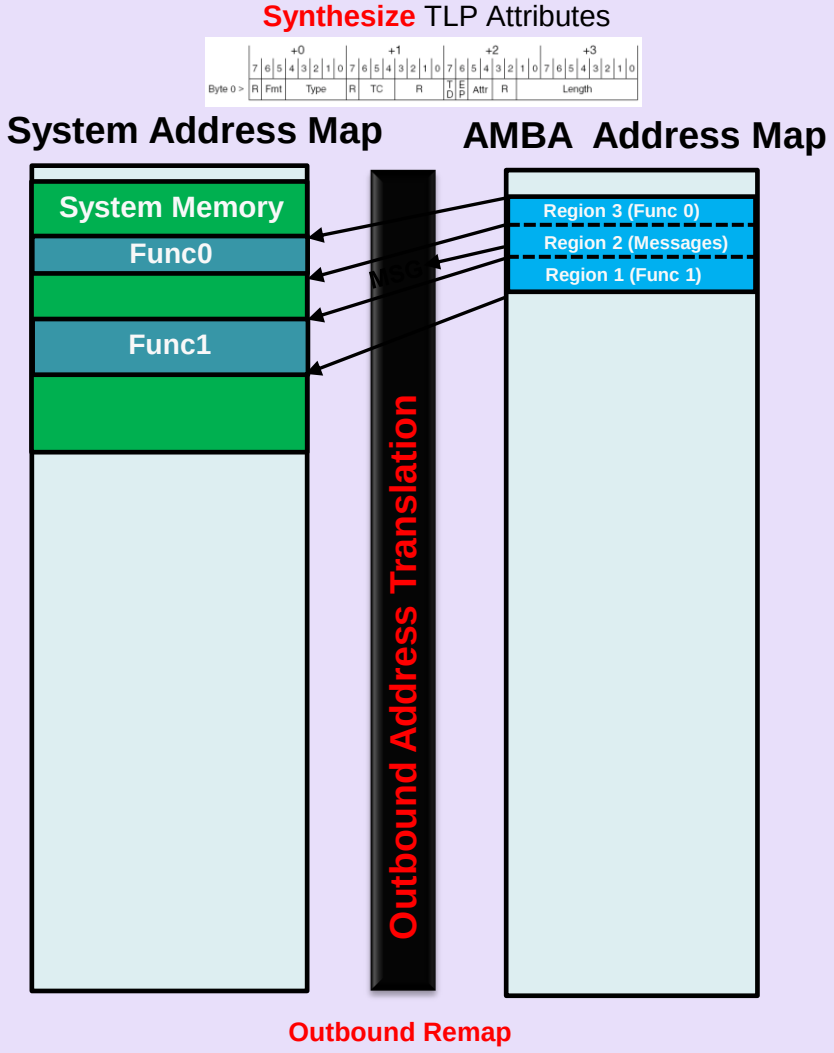
- Memory and I/O transactions access a particular BAR in a particular function.
- A function maps to a slave in AMBA
- A BAR maps to a resource within the slave
- So, a 'hit' to a particular BAR/function combination indicates which resource in which slave is being accessed
- The required address translation for every BAR/function combination is known at AMBA SoC build time

Inbound Address Translation Unit



- TLP type and header attributes may also be used to remap the inbound TLP address
  - ✓ An alternative is to provide AMBA sideband signals to signal PCIe attributes

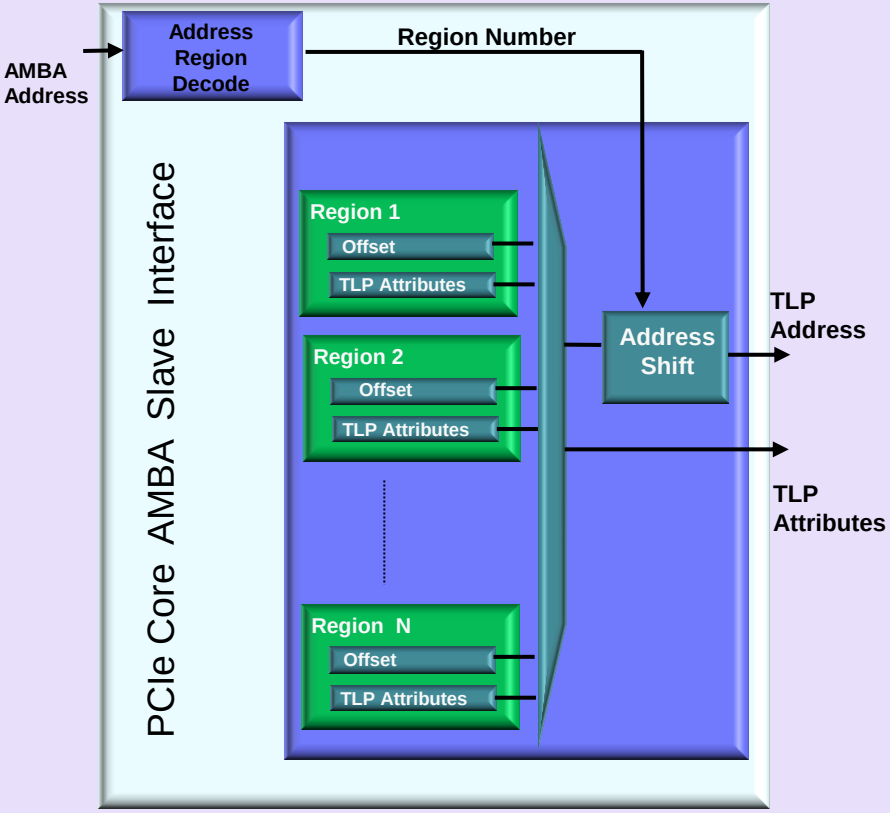
Outbound AMBA to PCIe Address Mapping



- PCIe Core appears as AMBA Slave
  - ✓ Share AMBA address space with other slaves
  - ✓ AMBA view into PCIe system address space
- Regionalized PCIe AMBA Slave, each region
  - ✓ Can represent an assignment to system memory for a particular function
  - ✓ 'Hit' to that region can be remapped as accesses anywhere in PCIe address space.
  - ✓ 'Hit' to a region selects programmed TLP header attributes assigned to that region
    - TLP Type, function #, RO, TC, requester ID
- A region can also be assigned for In-band message generation
- An alternative to generate TLP attributes is to
  - ✓ Provide sideband signals on AMBA OR
  - ✓ A command buffer which stores PCIe address and attributes

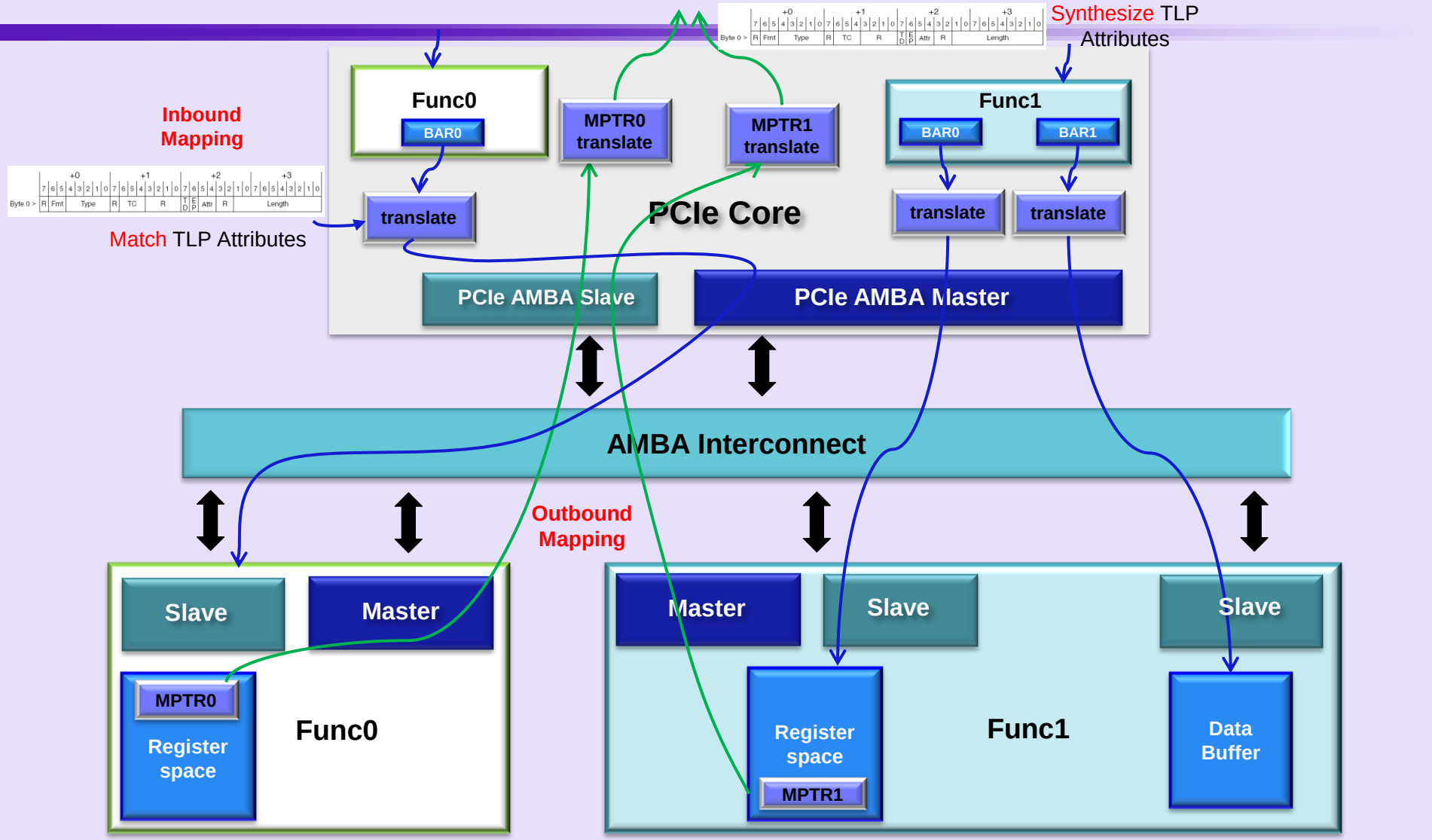
Outbound Address Translation Unit

Outbound Address Translation Unit (ATU)



- Region offset register stores the base address of the function assigned *physical* address space
- How does a region's offset register get its value?
- Function driver Programs Address pointer value implemented in function register, accessed via BAR access and translated to AMBA slave.
- Local CPU reads this register and programs ATU region accordingly
- AMBA master of device function accesses AMBA slave of PCIe device
- PCIe device generates PCIe TLP with address of buffer in system memory.
- There are other methods, with various advantages and drawbacks
  - ✓ Multi layered AMBA bus
  - ✓ Modify fabric

Mapping PCIe to AMBA



Agenda

- Introduction
- Protocol Translation
- **Data Flows**
- Performance
- Summary

Background
PCIe Ordering Rules

1. P must be allowed to pass NP
2. P must not pass P, unless RO bit is set :
3. P can optionally pass CPL
4. CPL must be allowed to pass NP
5. CPL must not pass P, unless the RO bit is set
6. CPL can optionally pass CPL
   - Not for split completions
   - AXI: Requests with same AXI ID must return in order
7. NP can optionally pass NP
8. NP can optionally pass CPL
9. NP must not pass P, unless the RO bit is set

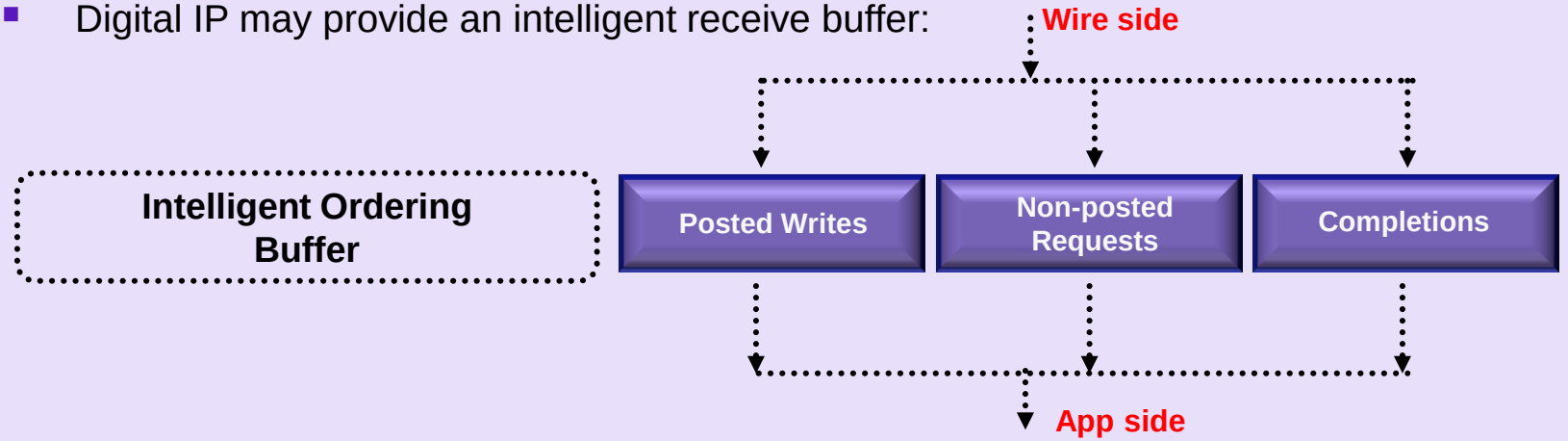
P = Posted

NP = Non Posted

CPL = Completion

Background
PCIe Ordering Solutions

- Digital IP may provide an intelligent receive buffer:

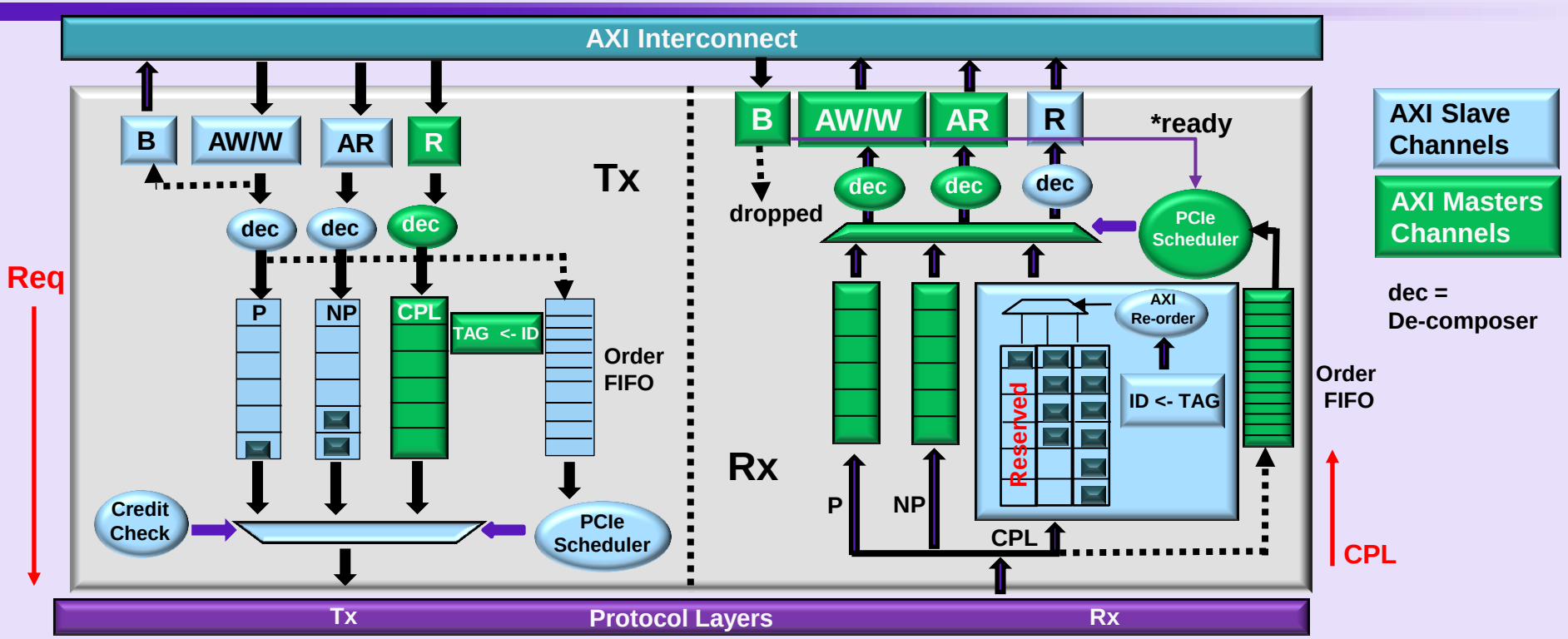


- 'Must Pass' rules require a multi-queue architecture
- The 'Must not Pass' rules requires synchronization and a notion of time order arrival to be maintained during de-queue
  - ✓ Hardware buffers automatically allow packets to pass or be blocked based on arrival order and packet type
- Other solutions
  - ✓ Limit issuing of read requests to prevent deadlock – S/W
  - ✓ Control location of producer/consumer flags so that bypass does not matter – System

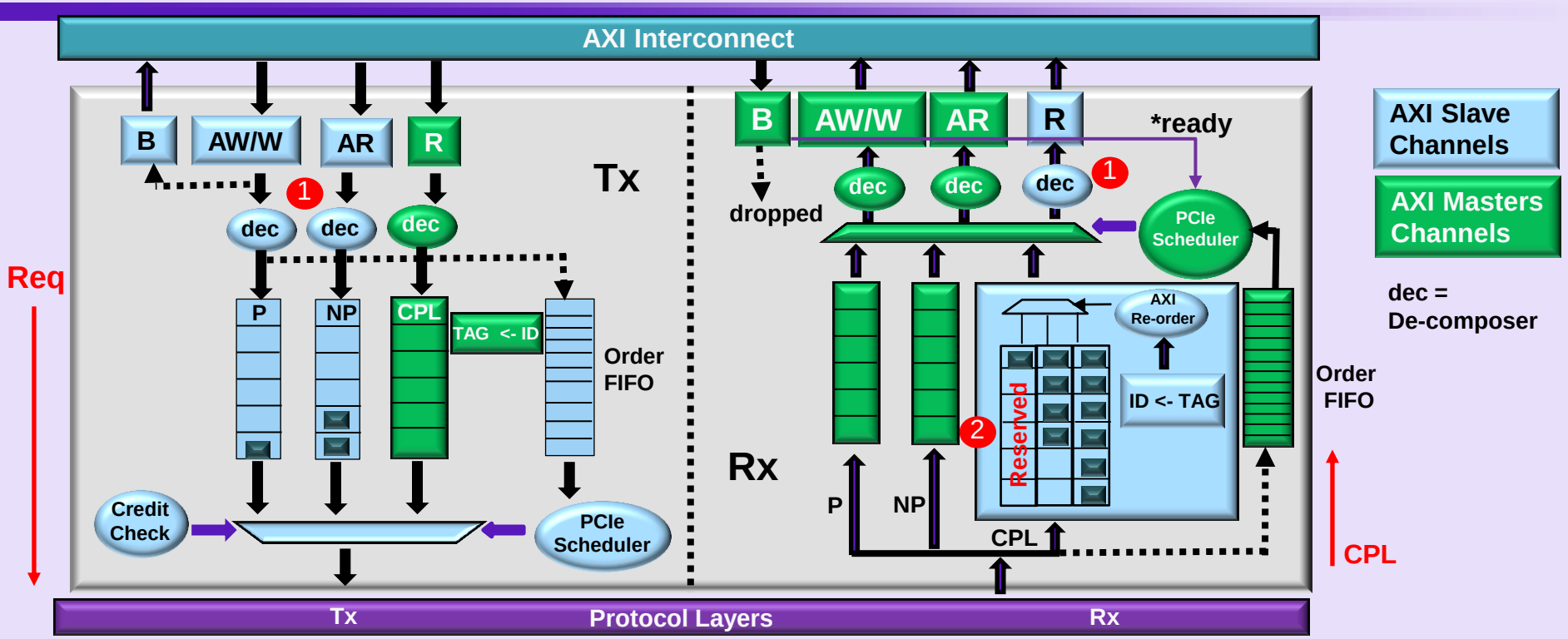
AMBA Bridge Architecture Considerations

1. **Manage differing transfer sizes between PCIe and AMBA**
   - ✓ Max payload is configured independently between AMBA and PCIe
   - ✓ Max read request size is configured independently between AMBA and PCIe
2. **Transfer IDs between PCIe and AXI – Tag management**
   - ✓ The max number of outstanding read requests is configured independently between AMBA and PCIe
   - ✓ Dynamic association between PCIe Tag number and AXI ID/AHB Master number and vice-versa depending on direction
3. **Bridging between AMBA and PCIe**
   - ✓ PCIe and AMBA ordering enforcement
   - ✓ Error management
   - ✓ Power management
     - – Bridge between PCIe active and device low power states and AXI low power interface
   - ✓ Interrupt and messaging mechanisms

AXI: Outbound Req/ Inbound CPL

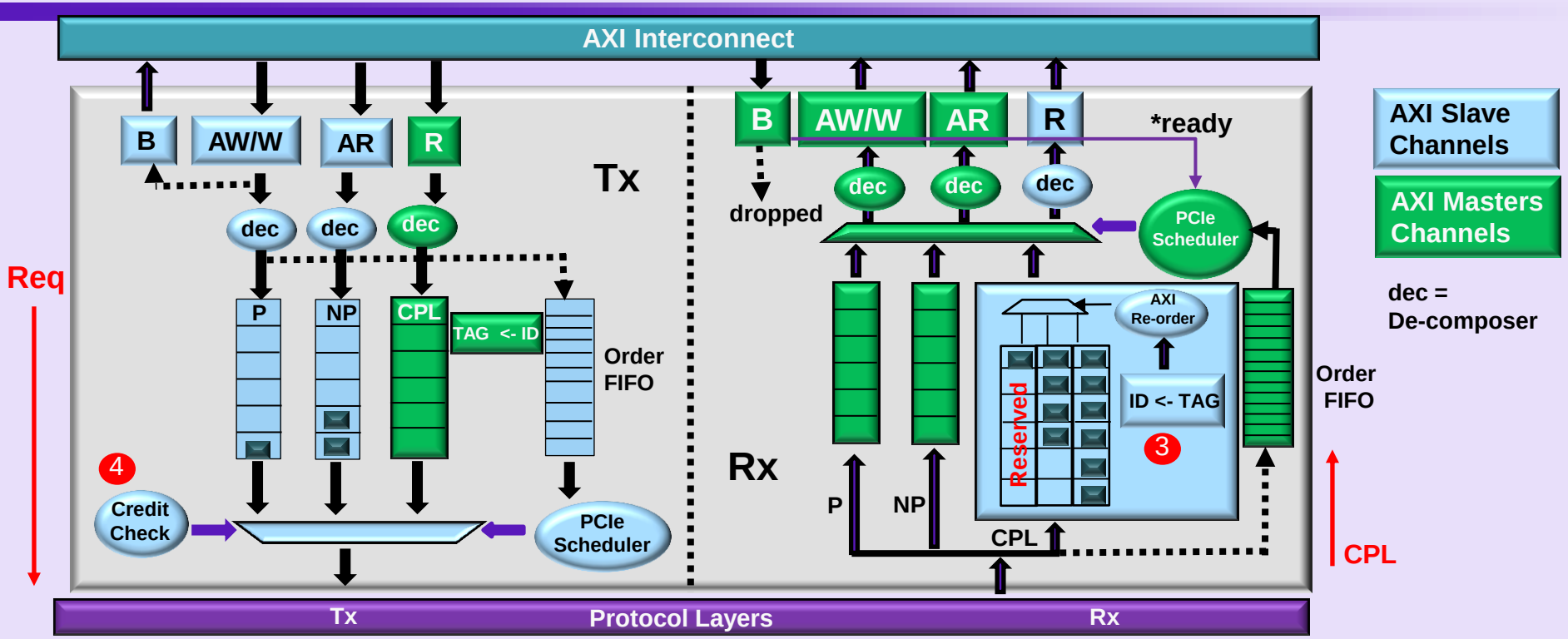


AXI: Outbound Req/ Inbound CPL



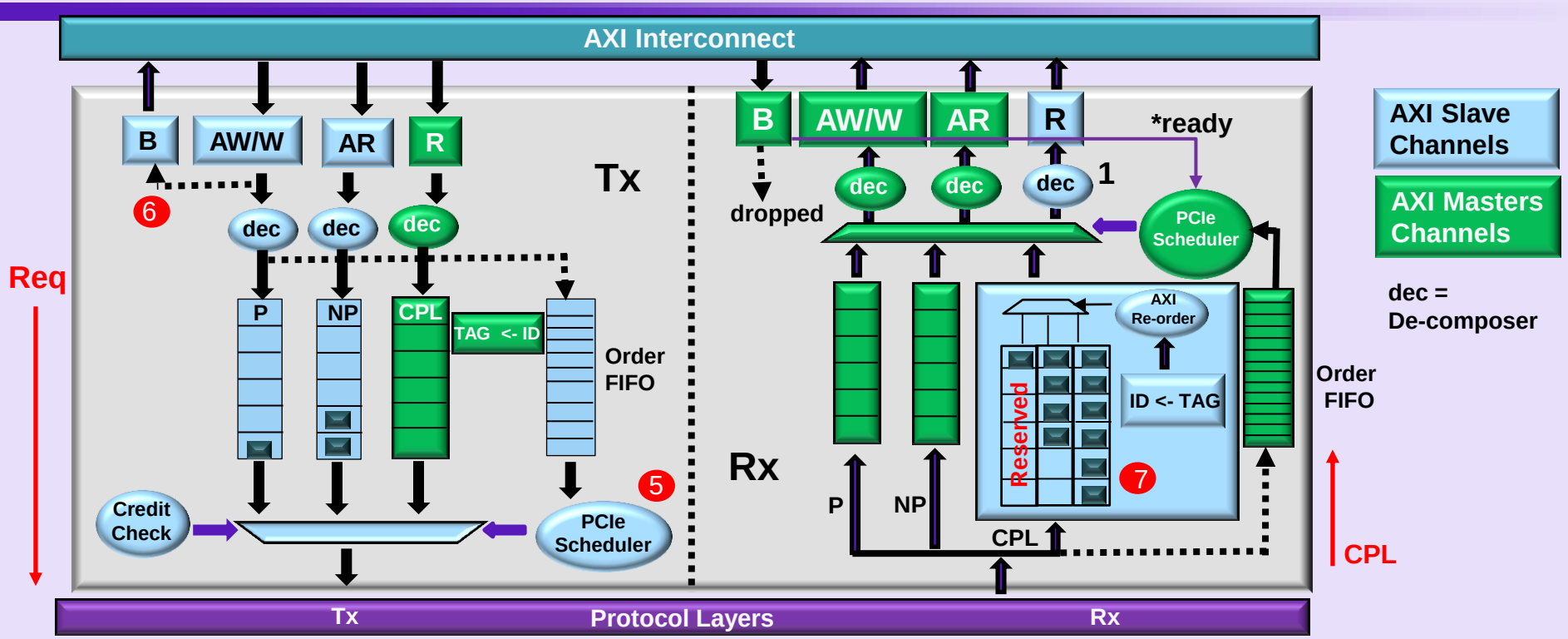
1. Decompose, both request and cpl, when mismatch in AMBA and PCIe MTU
2. For NP, reserve segment in Inbound CPL Queue as infinite credits advertised, as required by PCIe protocol

AXI: Outbound Req/ Inbound CPL



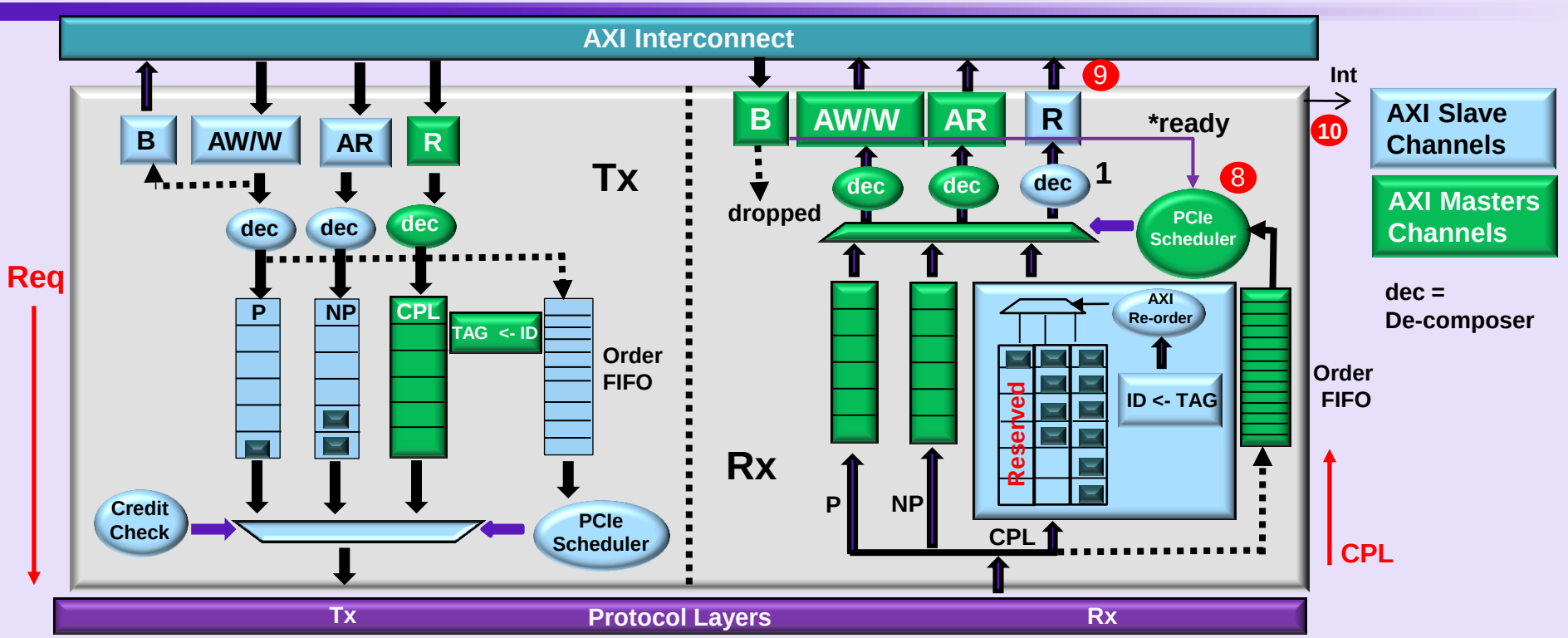
3. Store AXI ID to PCIe TAG mapping, required for inbound CPL
4. P/NP/CPL: Stall on lack of credits

AXI: Outbound Req/ Inbound CPL



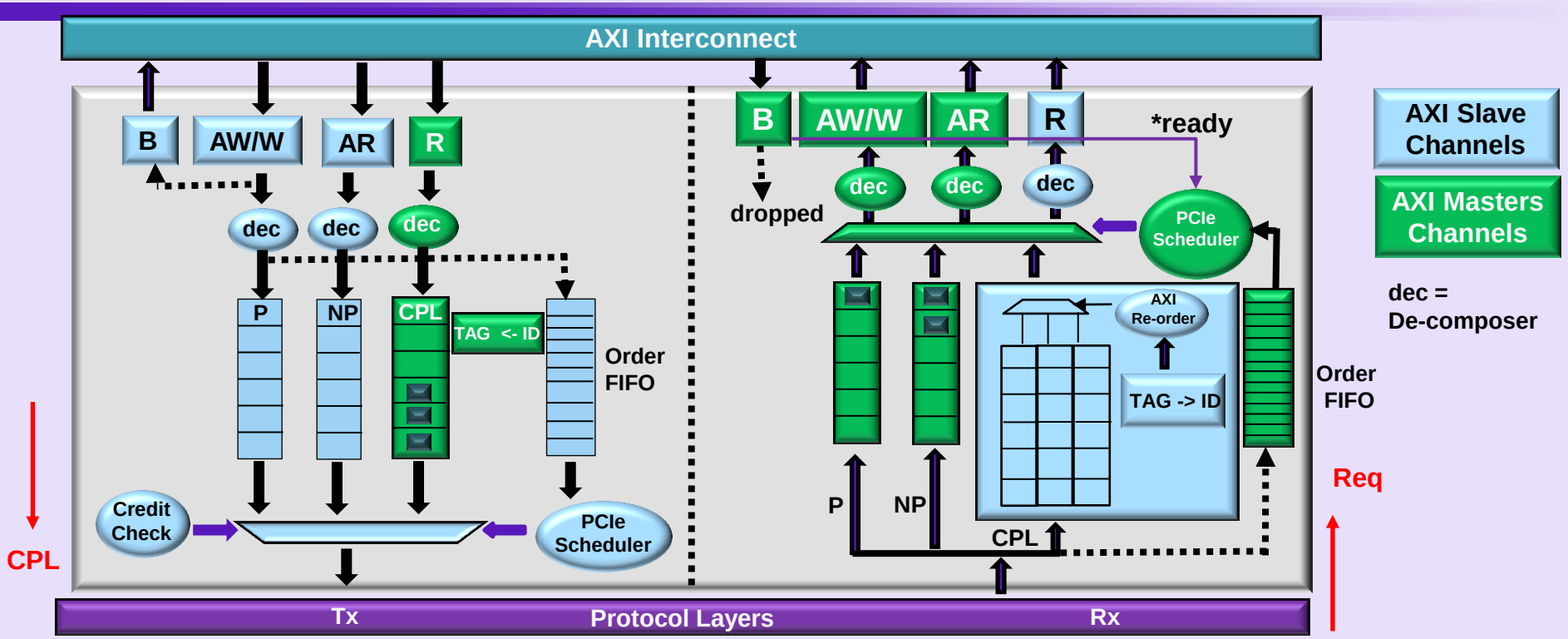
5. NP(Slave AR)/ CPL(Master R) stalled if previous P(Slave AW) blocked
6. P writes shown, for NP write would need to wait for wire CPL
7. Re-order buffer ensures in-order response

AXI: Outbound Req/ Inbound CPL

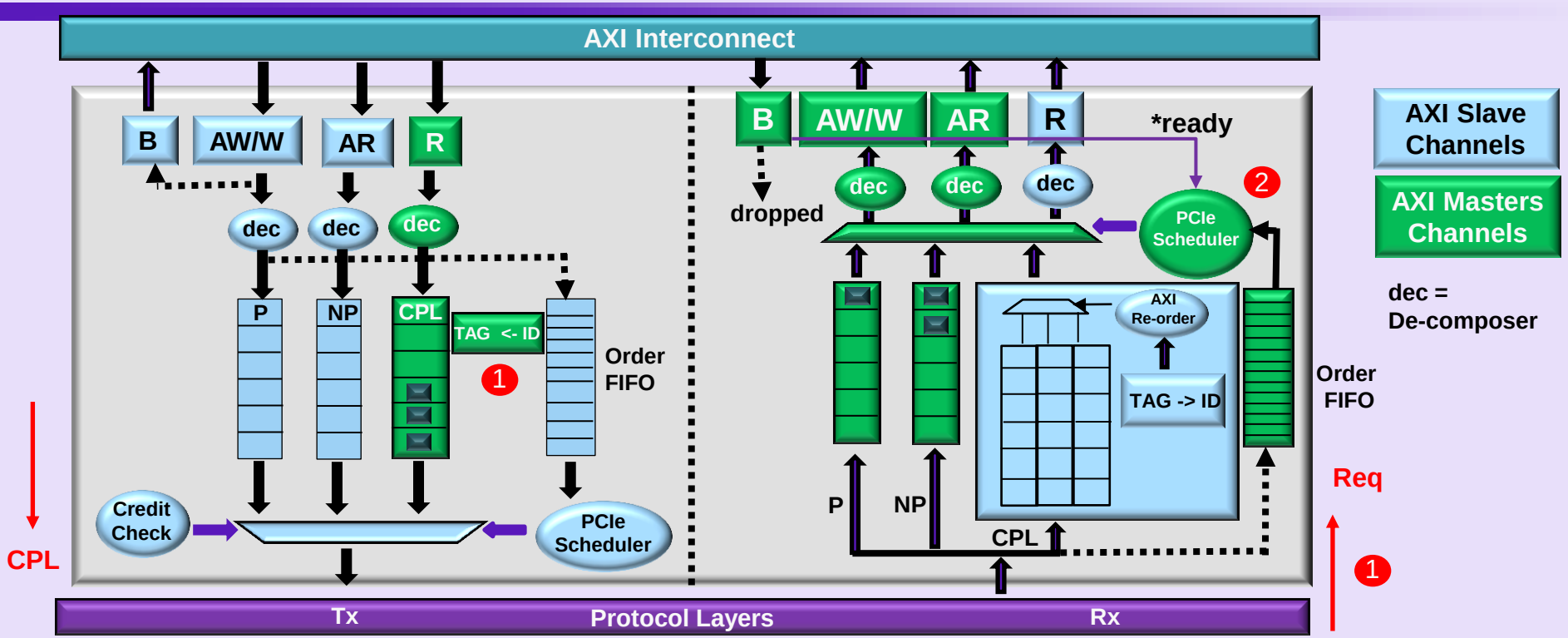


8. CPL(Slave R) stalled if previous P(Master AW) blocked
9. Normally map ANY CPL error to ***SLVERR (AXI)*** to application master
10. Inbound error message due to P request failure generates interrupt

AXI: Inbound Req/ Outbound CPL

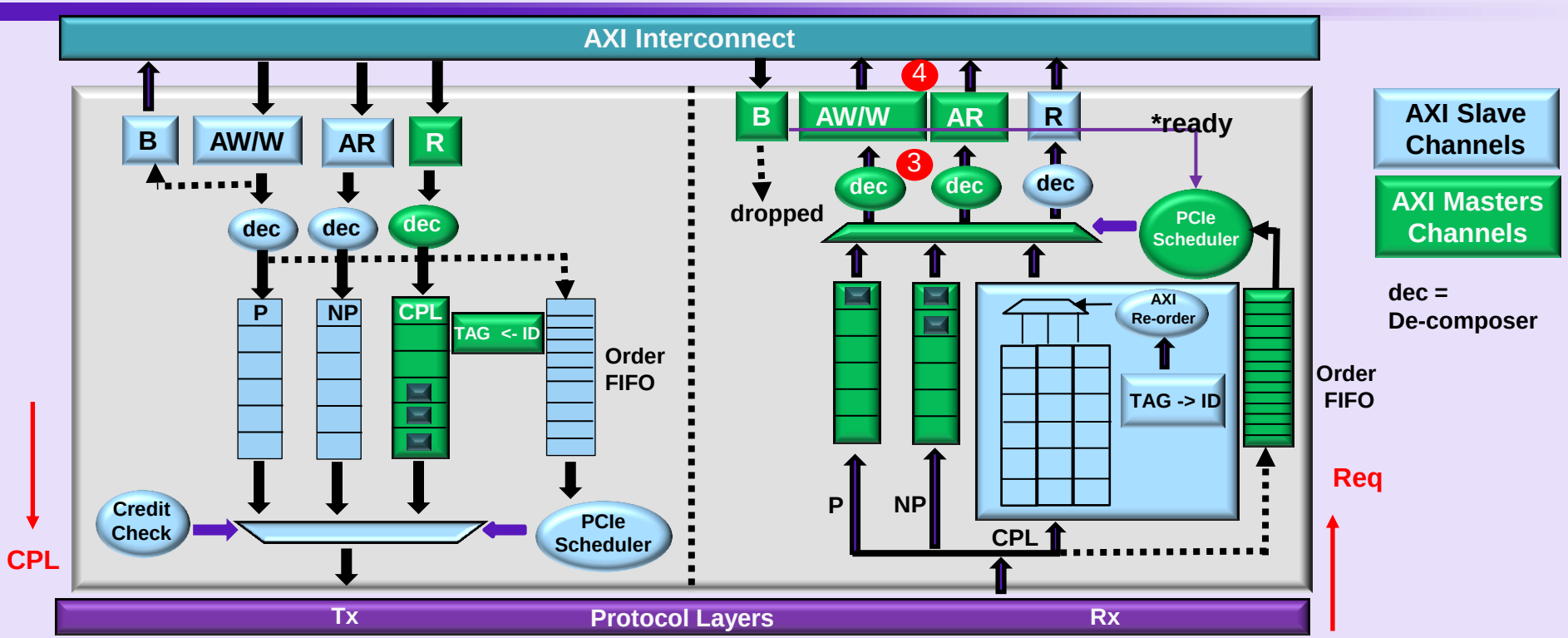


AXI: Inbound Req/ Outbound CPL



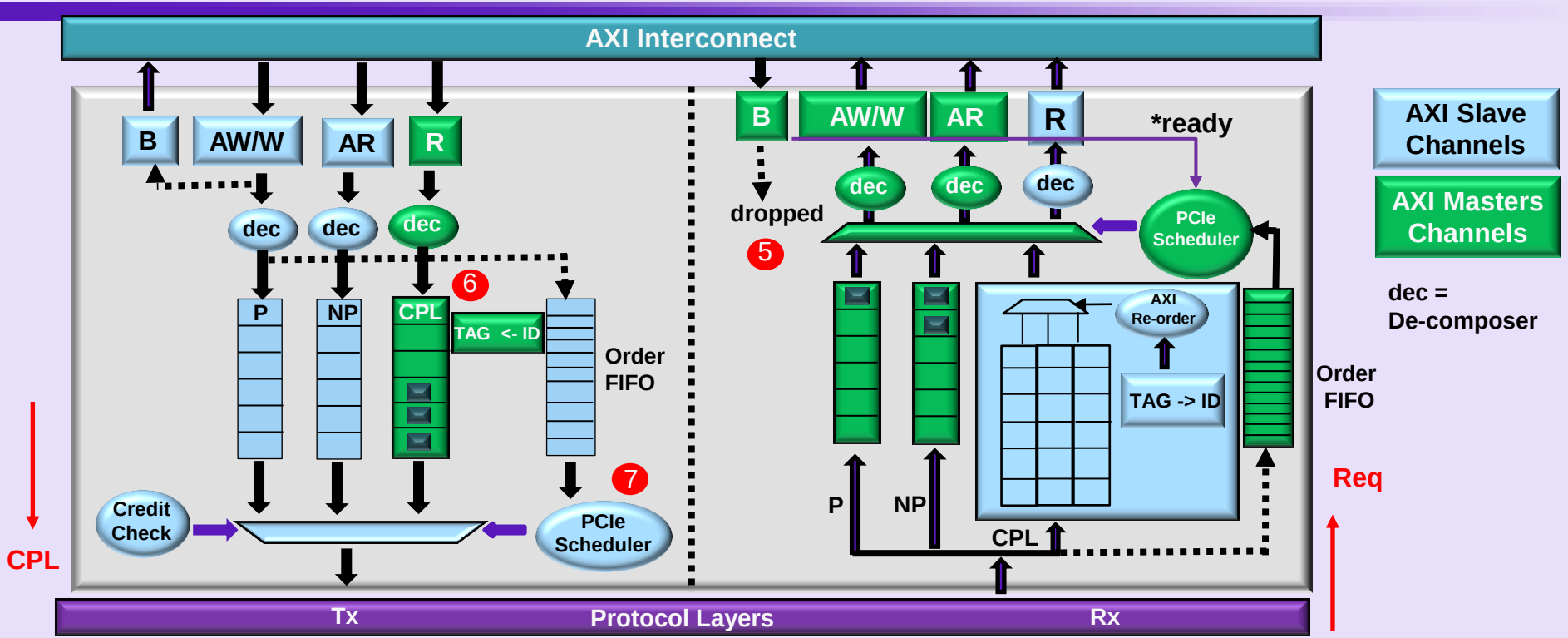
1. Store PCIe tag to AXI ID mapping, required for outbound CPL
2. NP (Master AR)/CPL(Slave R) stalled if previous P(Master AW) blocked – PCIe ordering rules

AXI: Inbound Req/ Outbound CPL



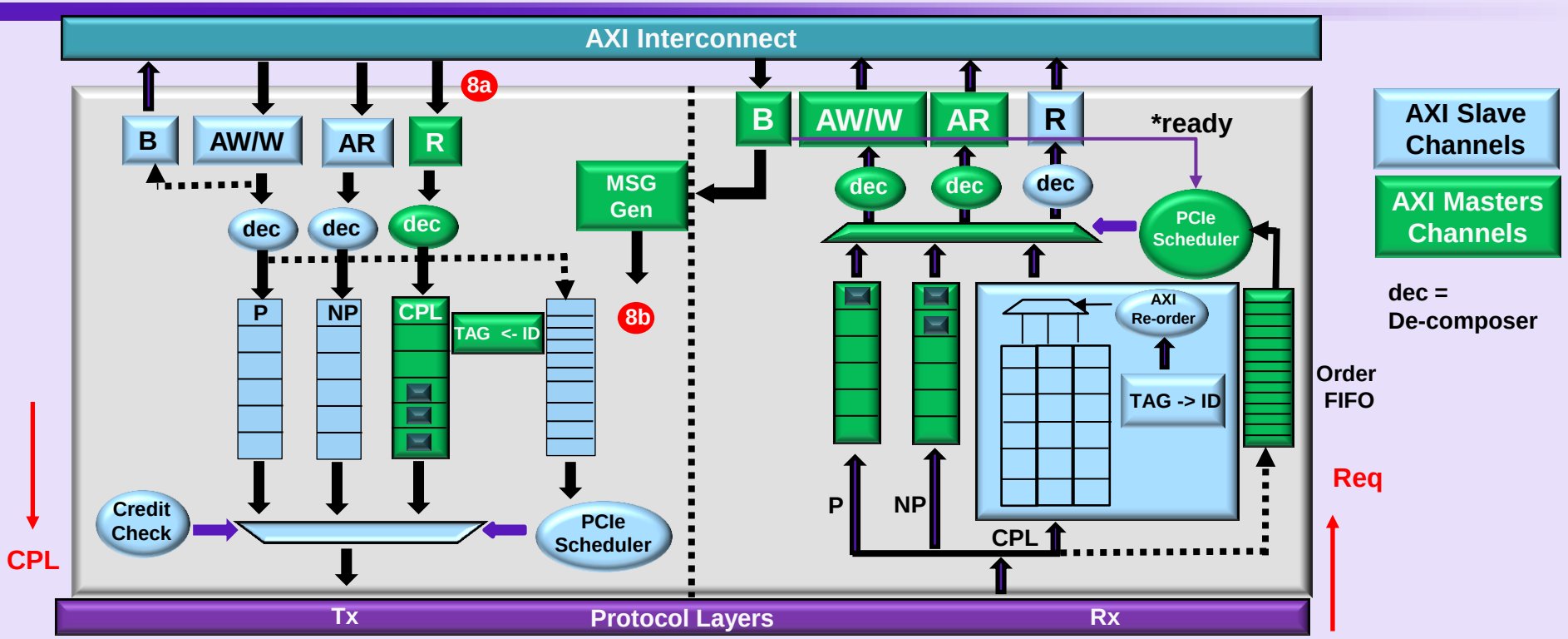
3. Decompose request when mismatch in AMBA and PCIe MTU and NCBE
4. AXI ID assignment for maximum performance while maintaining PCIe ordering
   a) NP requests assigned different* AXI ID's, Posted requests same ID, stall NP to ensure no passing of P in AXI fabric

AXI: Inbound Req/ Outbound CPL



5. P Writes shown, for NP write then AXI B response generates CPL
6. Use Split completions of *Max_Payload_Size* to service *Max_read_request_size*
   a) Minimizes latency and buffering
7. CPL(Master R) stalled if previous P(Slave AW) blocked

AXI: Inbound Req/ Outbound CPL



8. When application AXI slave responds with SLVERR or DECERR
   a. In response to inbound NP read
      - Generate completion with UR or CA
   b. In response to inbound posted write
      - Need to report via error messaging

Clocking/Reset

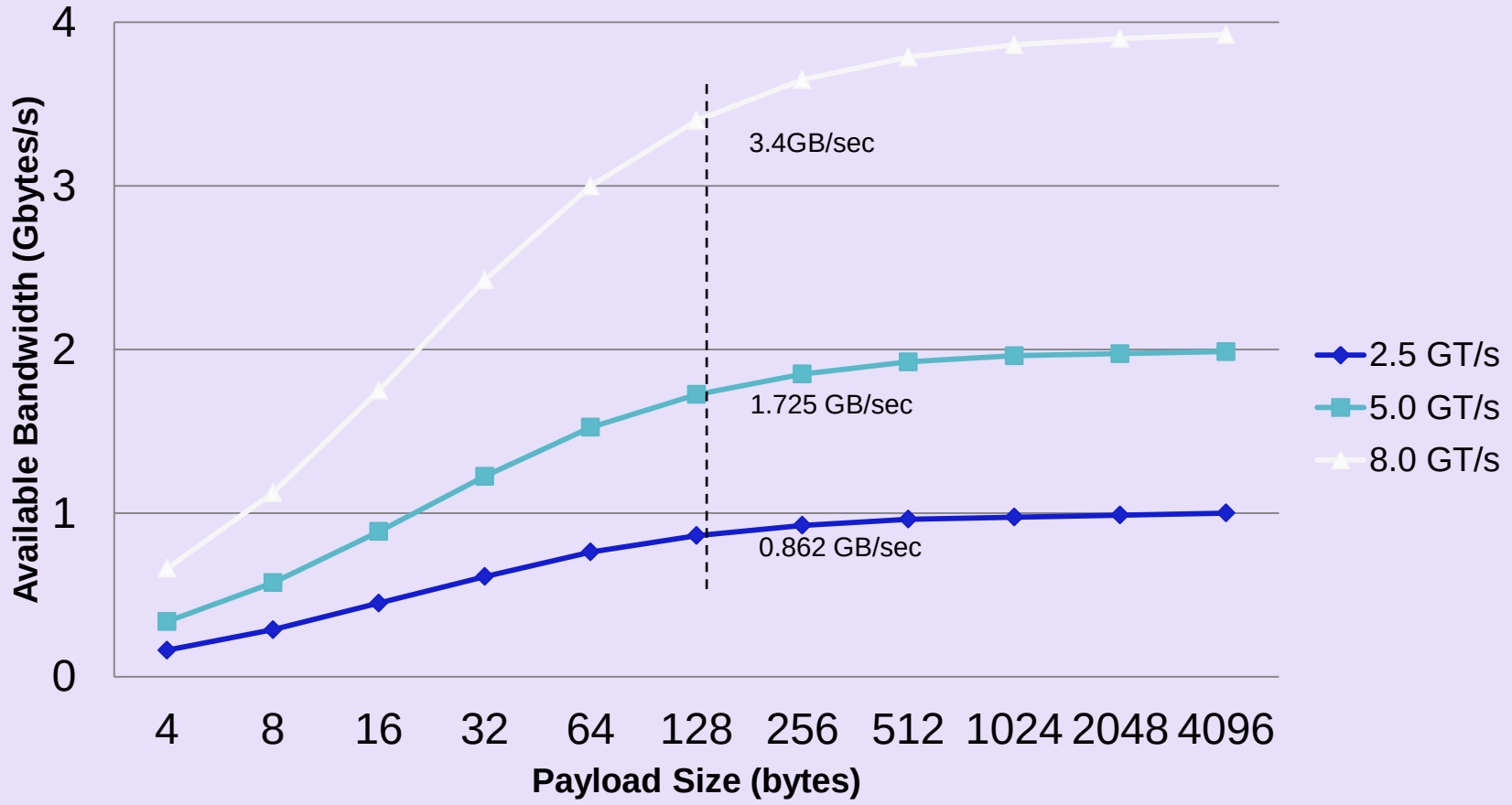
- **Clock**
  - ✓ PCIe protocol layer clock rate is fixed
  - ✓ For a given number of lanes and core datawidth
  - ✓ PCIe clock from PLL in PCIe PHY
  - ✓ AMBA clock rates can vary
    - Clock source is SoC PLL
  - ✓ Data is synchronized from one domain to the other

- **Reset**
  - ✓ When the PCIe link goes down it expects the device logic of the PCIe core to be reset
  - ✓ The AMBA subsystem operates asynchronously from the PCIe link
  - ✓ Outstanding transactions on the AMBA bus, master/slave, need to be flushed gracefully with error response before reset to prevent AMBA deadlock

Agenda

- Introduction
- Protocol Translation
- Data Flows
- **Performance**
- Summary

PCI EXPRESS EFFECTIVE THROUGHPUT *(per direction)*

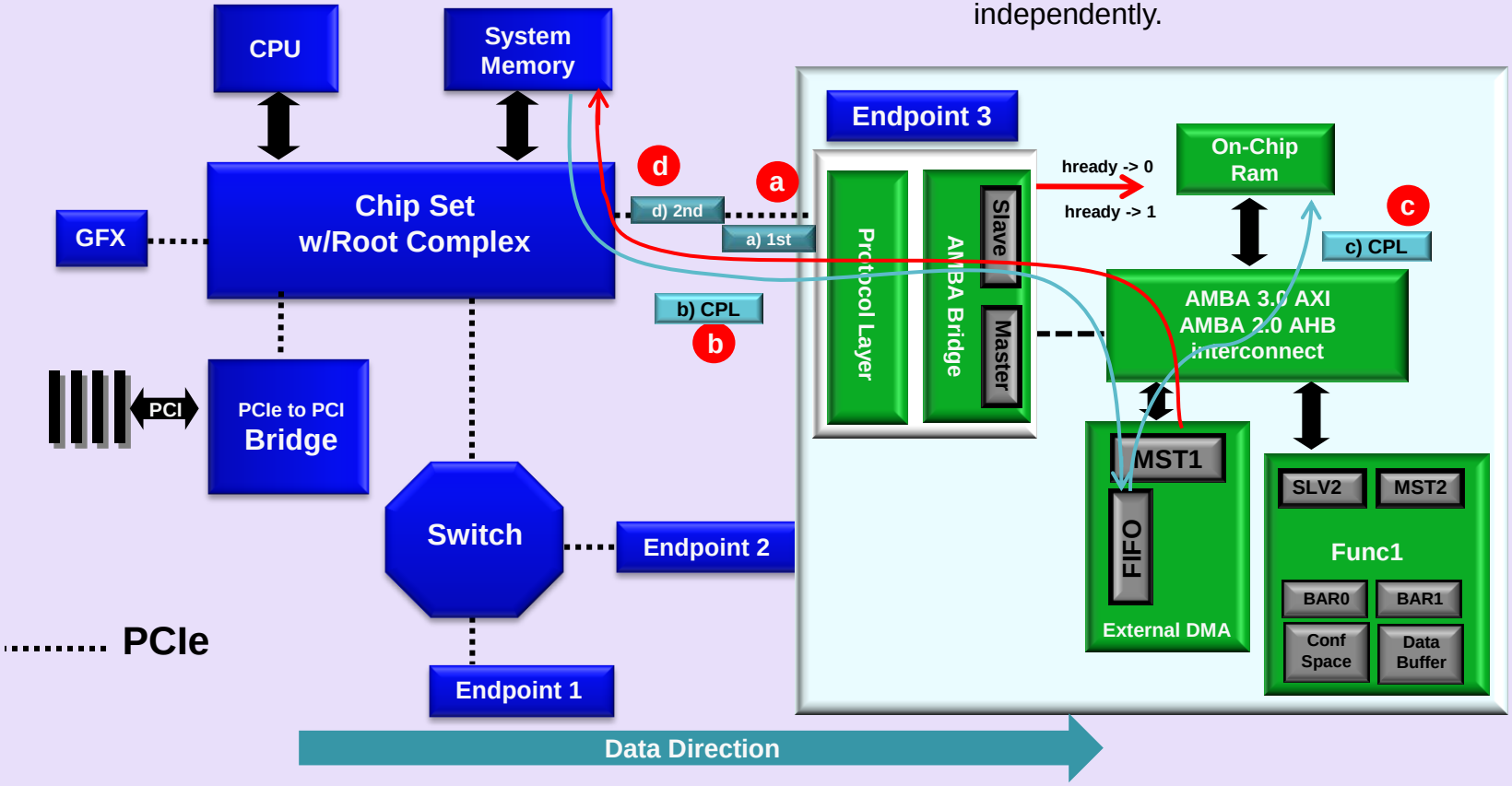


PCIe available bandwidth for 2.5 GT/s, 5.0 GT/s and 8.0 GT/s x4 lanes

AHB Read Performance With External DMA

a) 1st Read request of AMBA MTU by AMBA DMA
b) CPL response Internal DMA buffer
c) DMA masters bus and drives data to on chip RAM
d) 2nd Read request of AMBA MTU

- B/W limited by AMBA MTU and round trip delay
  - AHB Single threaded
  - AXI, mitigated by fact the AXI is multi-threaded, however,
    - #PCIe Tags(32) and #AXI tags configured independently.

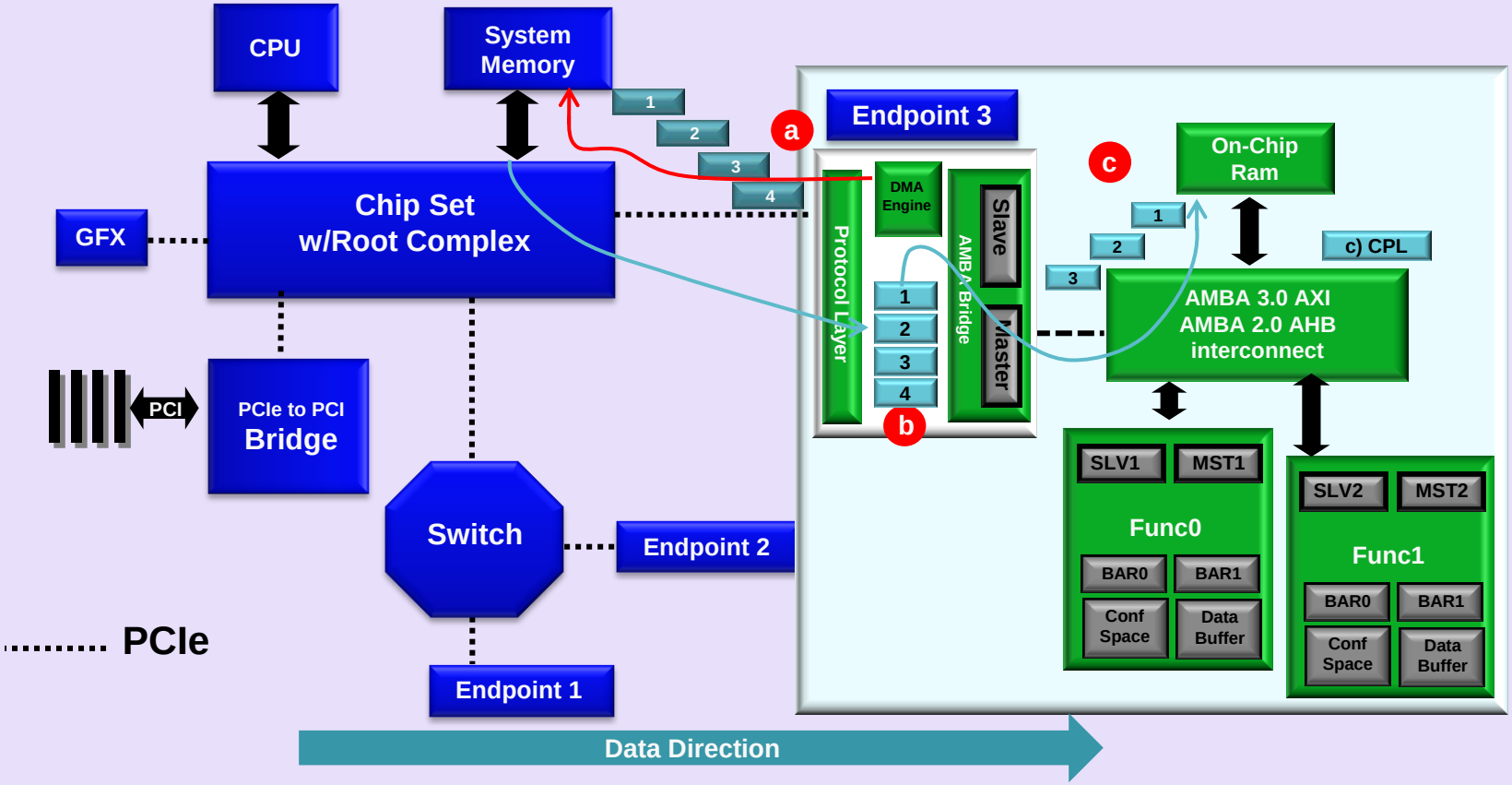


AHB Read Performance With Embedded DMA

a) DMA generates multiple Read requests (#PCIe Tags) of PCIe MTU

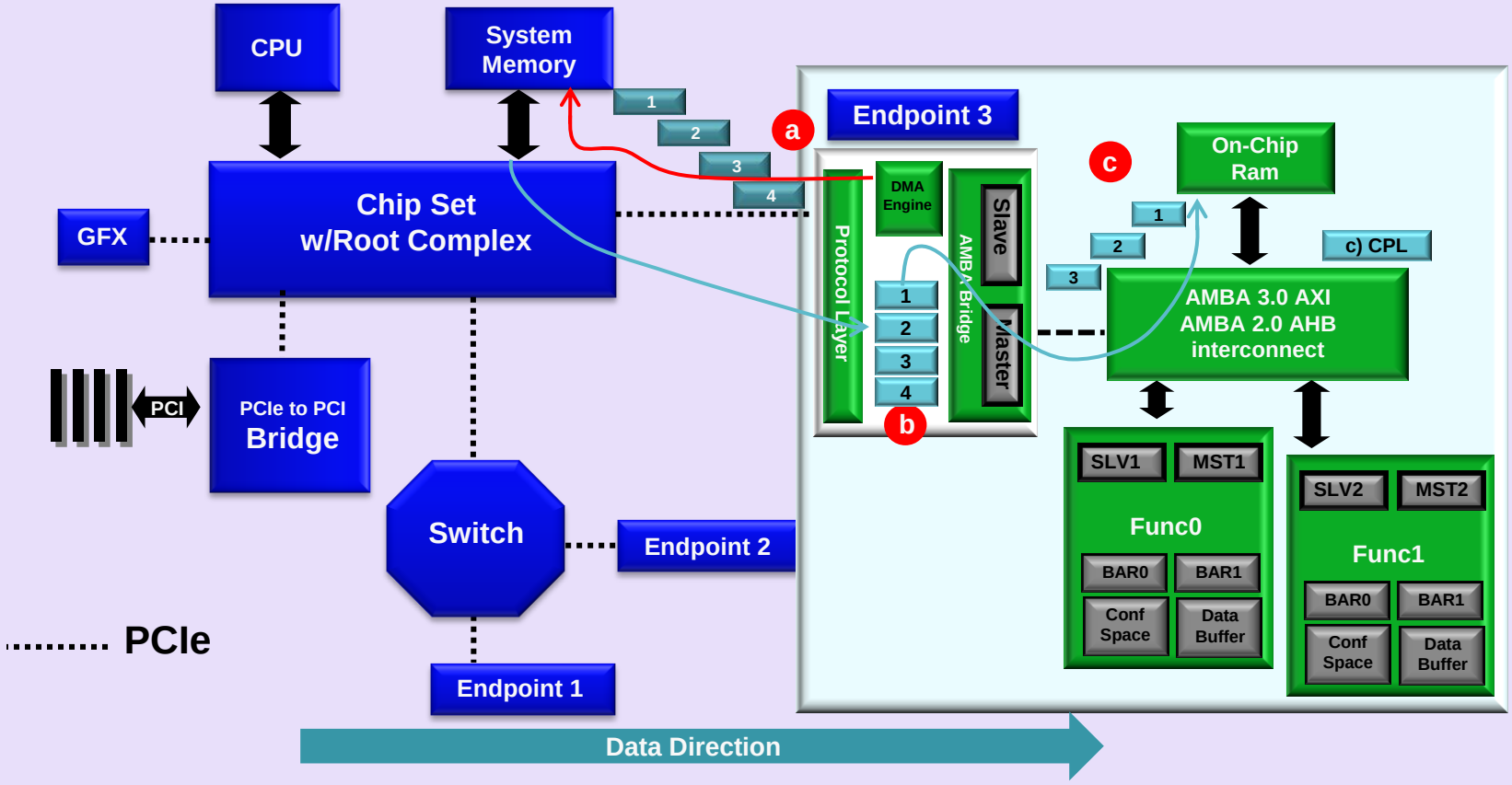
b) Host system responds and CPL's stored in eDMA

c) eDMA masters AMBA bus to drive data to On-Chip RAM



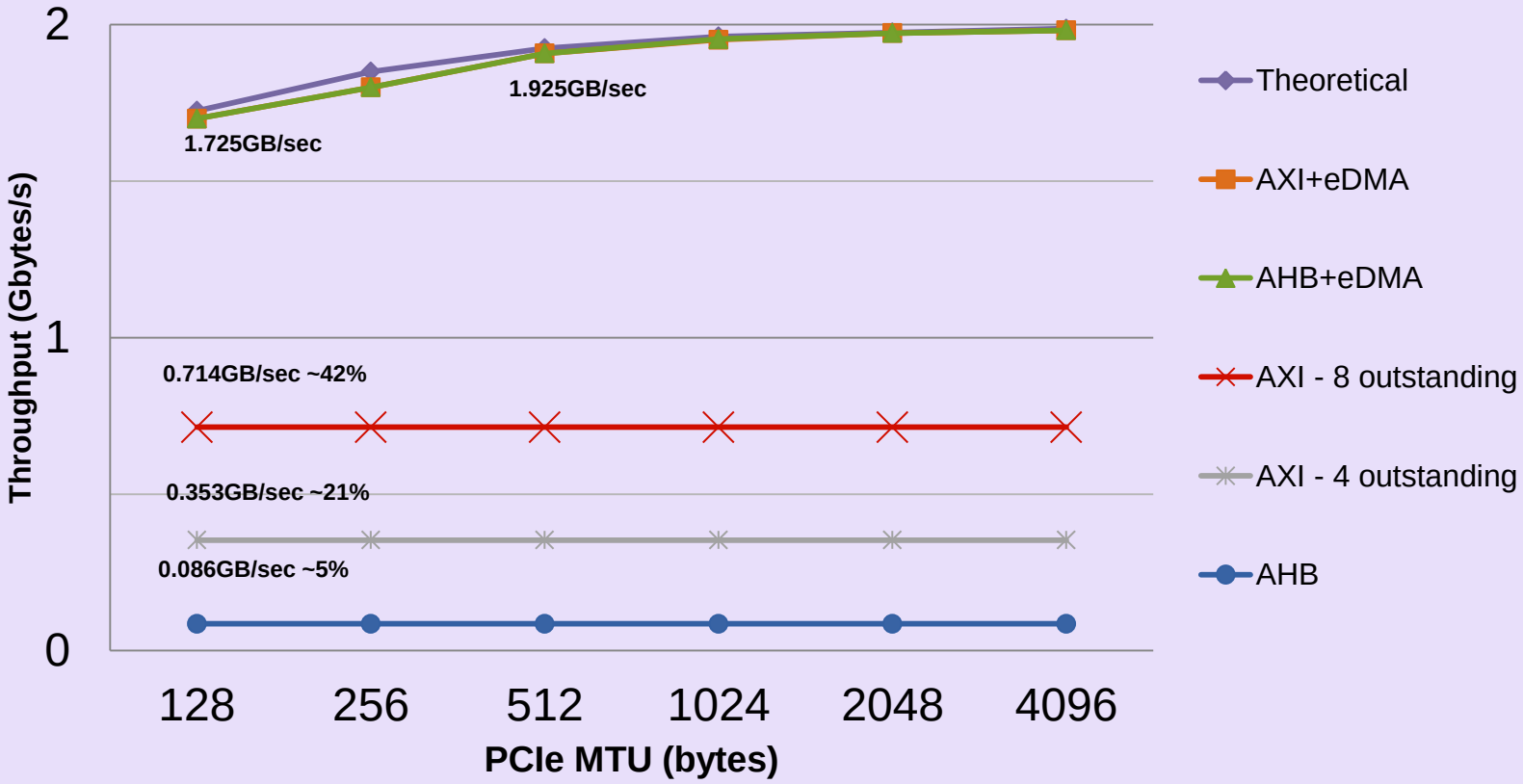
AHB Read Performance With Embedded DMA

- Improved B/W limited by round trip delay by removing dependency between #PCIe Tags and #AXI Tags
- Removed B/W limited by AMBA MTU
- Improved AMBA Bus performance
  - ✓ Number of accesses on AMBA bus halved



Remote to Local Transfer

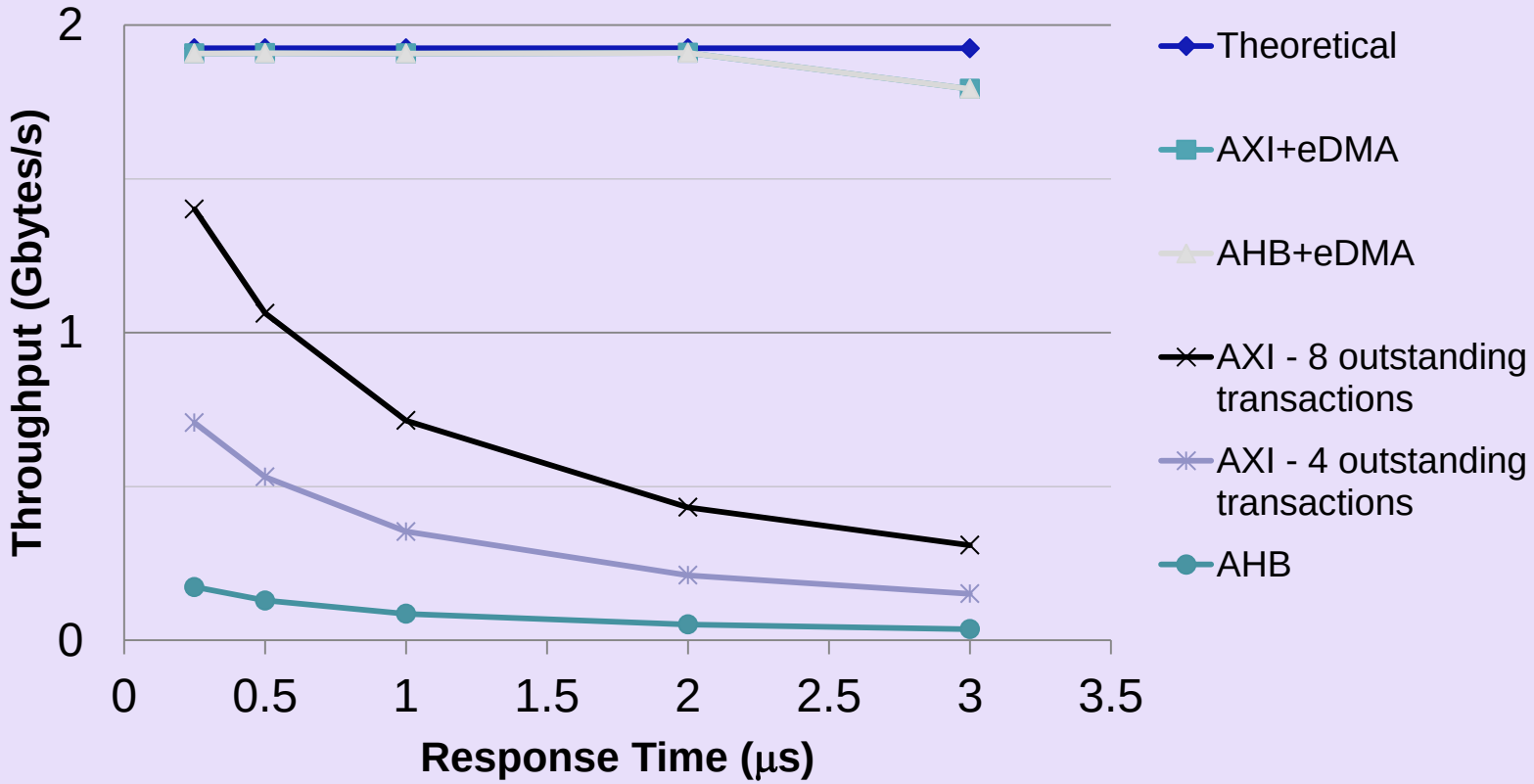
Variable PCIe MTU, Fixed Response Time



- 5.0 GT/s x4 – 32 PCIe Tags – 16 tags allocated for eDMA traffic
- 1 us wire response time
- **AMBA MTU:** Fixed at 128 bytes i.e. 16 beat burst on 64 bit AMBA bus

Remote to Local Transfer

Fixed PCIe MTU, Variable Response Time

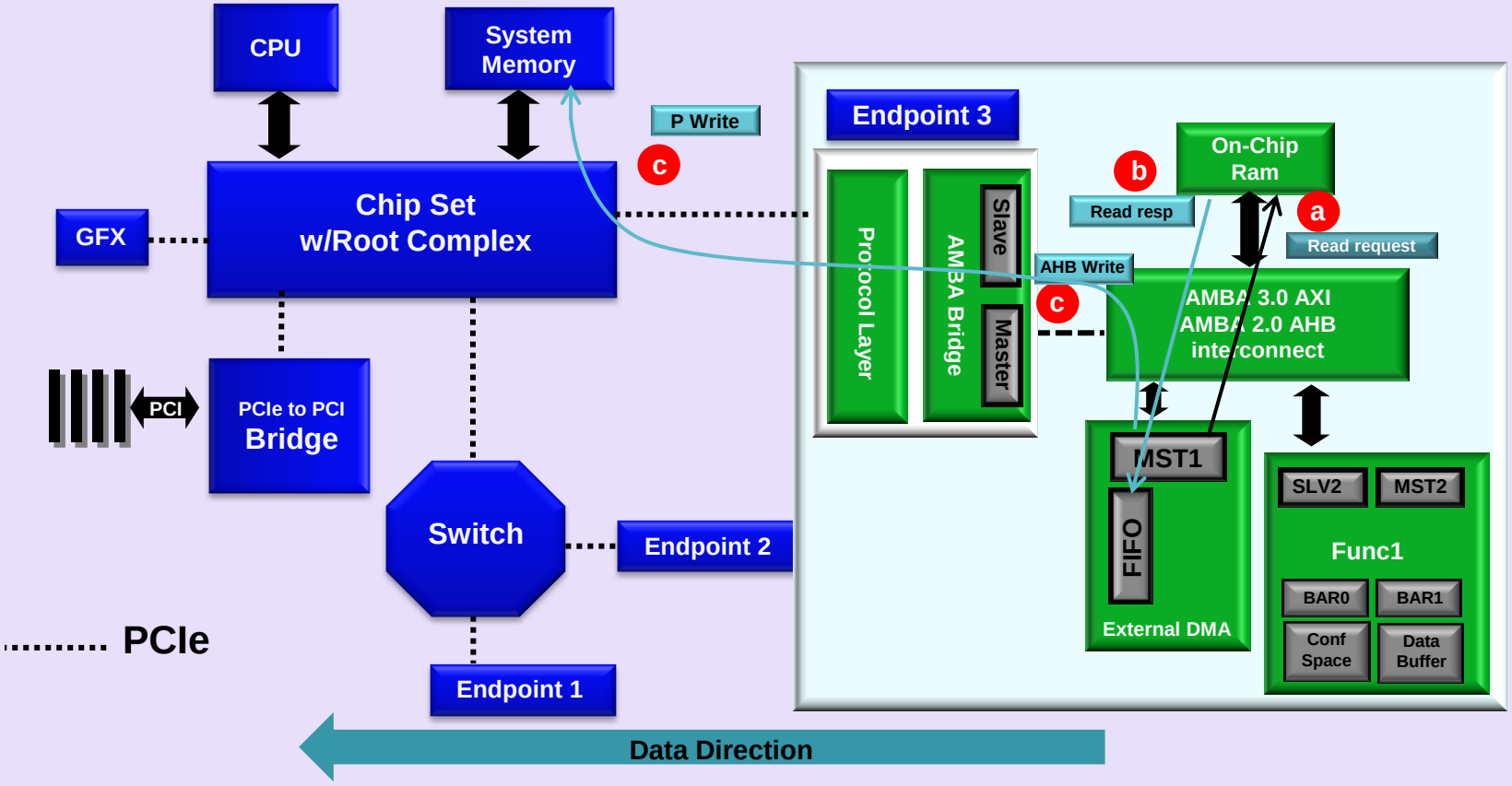


- 5.0 GT/s x4 – 32 PCIe Tags – 16 tags allocated for eDMA traffic
- 512 bytes PCIe MTU
- **AMBA MTU:** Fixed at 128 bytes

AMBA Write Performance External DMA

a) Read request of AMBA MTU by AMBA DMA
b) Response to Internal DMA buffer
c) DMA masters bus and drives data to System Memory

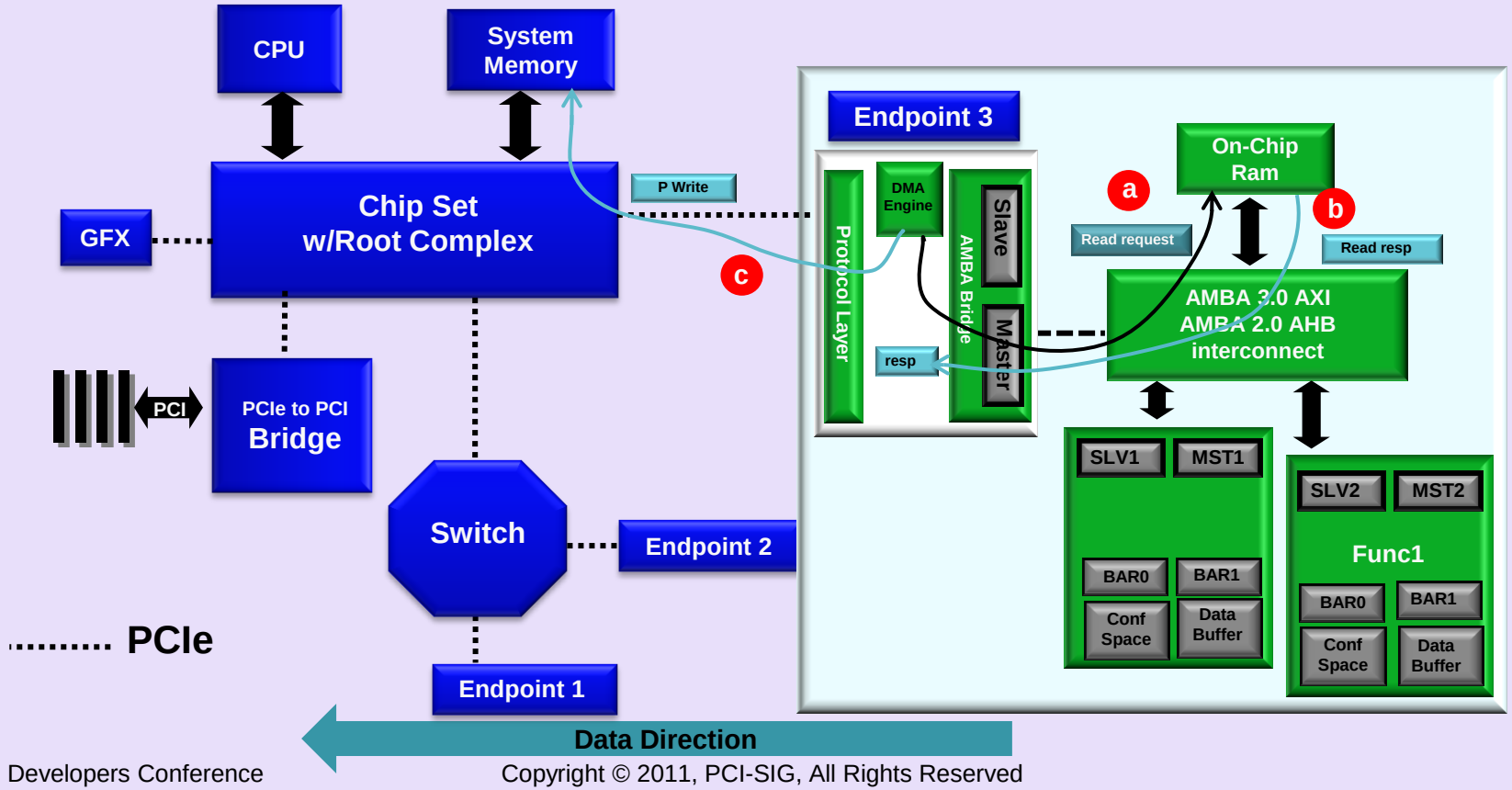
a) B/W *not* limited by round trip delay – Assuming posted writes
b) B/W limited by AMBA MTU as P Write on PCIe link has AMBA MTU of data and not PCIe MTU



AMBA Write Performance Embedded DMA

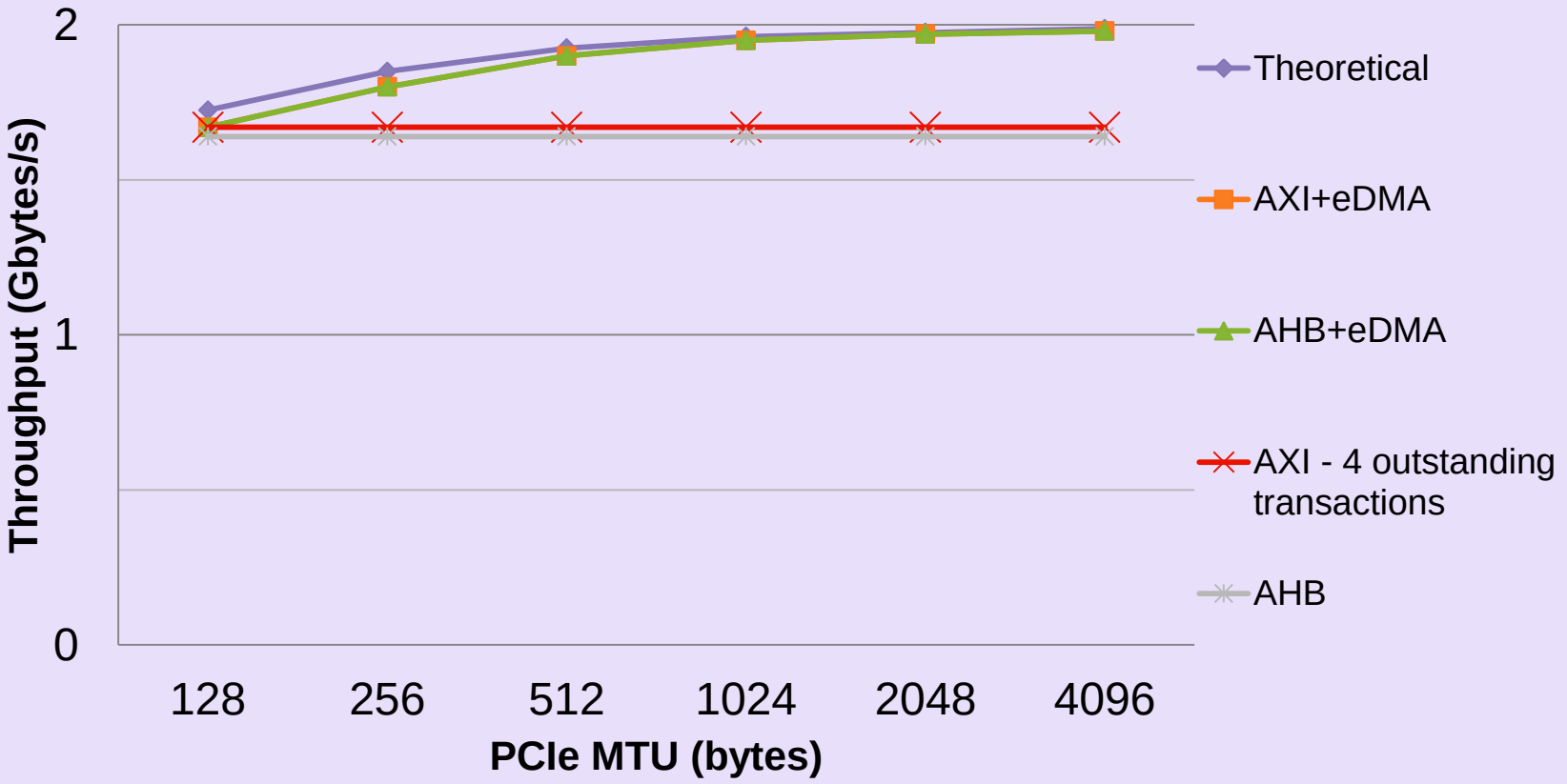
a) Read request by embedded DMA
b) Embedded DMA engine collects PCIe MTU data
c) PCIe core masters PCIe bus

- ✓ Improved AMBA bus performance by halving number of AMBA transactions
- ✓ Remove AMBA MTU limitation of packets on PCIe wire



Local to Remote Memory Transfer

PCIe MTU, Fixed Response Time



- 5.0 GT/s x4 – 32 PCIe Tags – 16 tags allocated for eDMA traffic
- 1 us wire response time – AMBA MTU: Fixed at 128 bytes

Agenda

- Introduction
- Protocol Translation
- Data Flows
- Performance
- **Summary**

Summary

- PCIe-AMBA Bridge
  - ✓ Allows AMBA Express Endpoint IO devices to be connected to Root Complex
  - ✓ Allows Express Endpoint IO devices to be connected to AMBA Root Complex
- PCIe host system is not aware of AMBA Endpoint
- Address mapping required between PCIe and AMBA for multiple enumerated PCIe address maps to a single fixed AMBA address map
- Mapping of PCIe packet header attributes also needs to be considered
- PCIe and AXI Ordering rules impact buffering choices and require coherency between AMBA *master/slave* ports and AMBA *read/write* channels
- Mismatches in allowed AMBA and PCIe payloads need to be designed in
- An embedded DMA is required to saturate completion path when reading from remote memory

Thank you for attending the
PCI-SIG Developers Conference 2011.

For more information please go to
www.pcisig.com