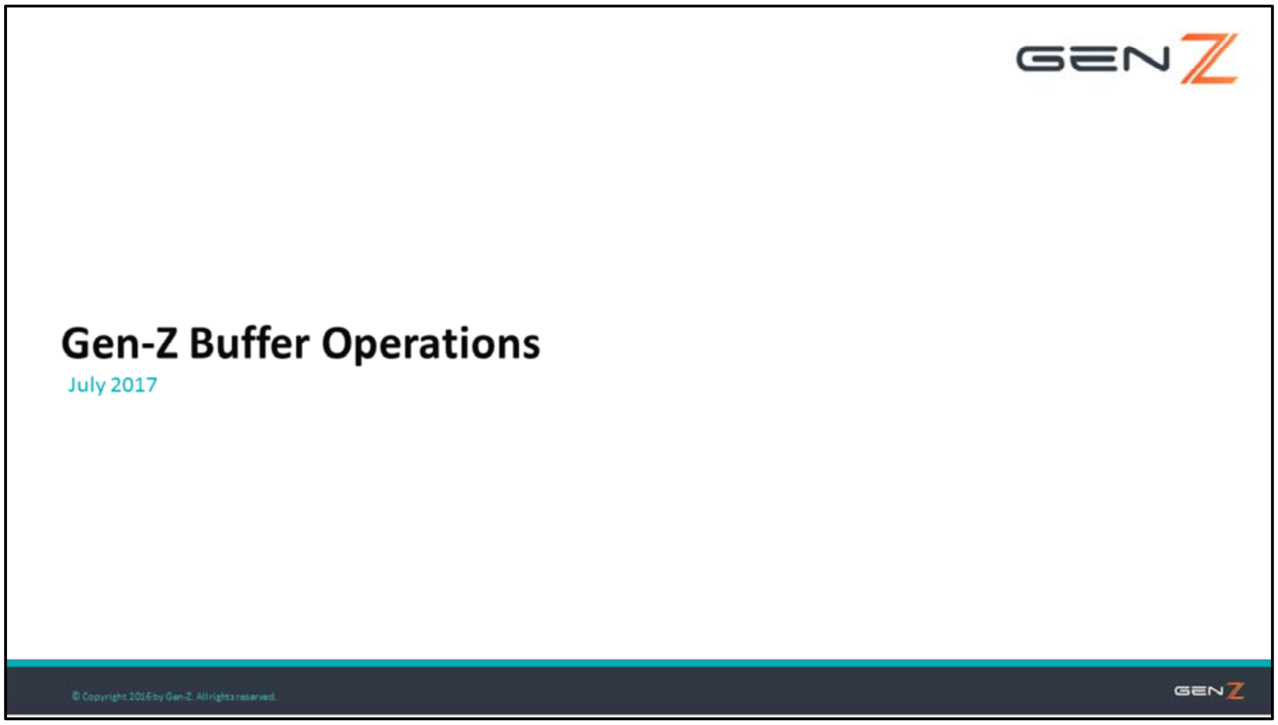
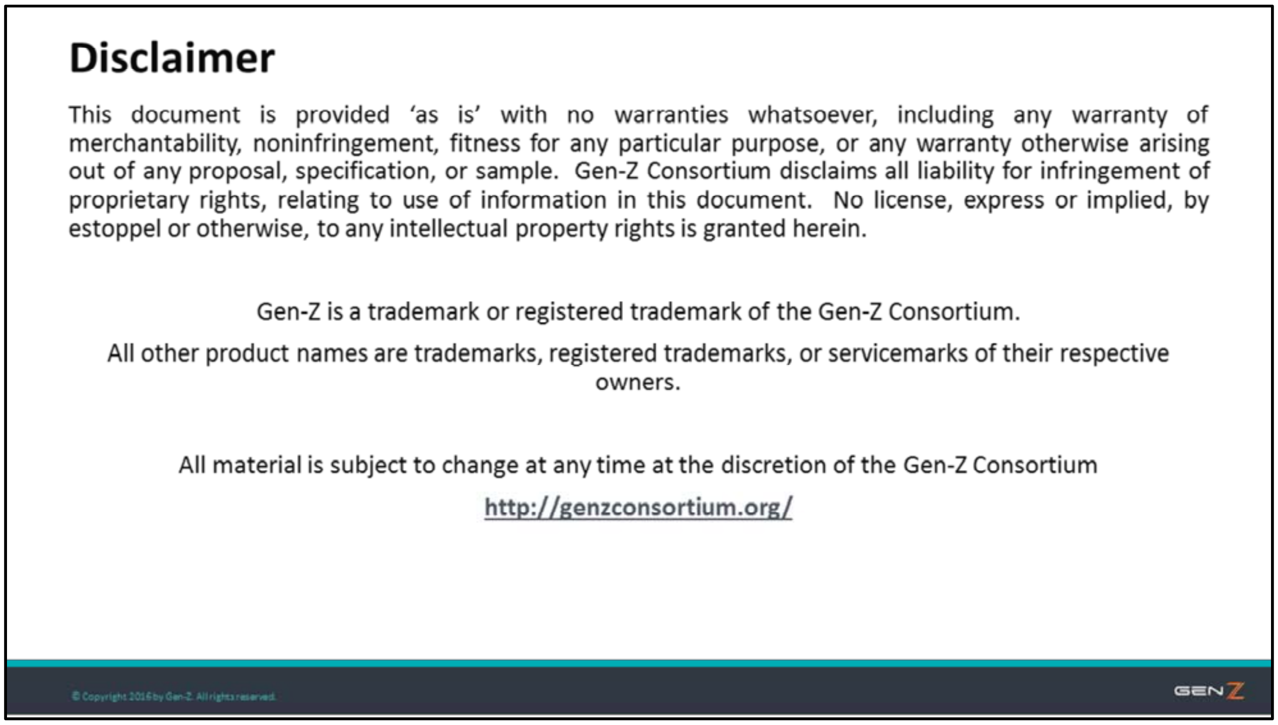


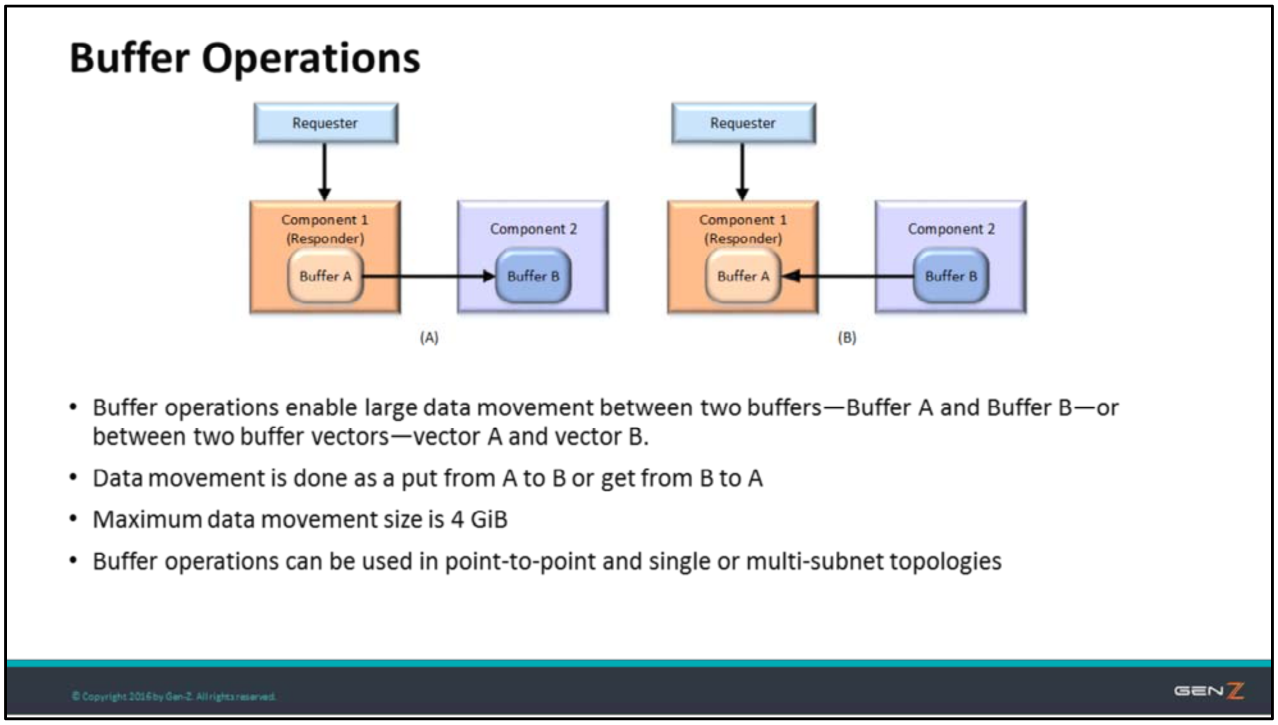
Gen-Z Buffer Operations

July 2017

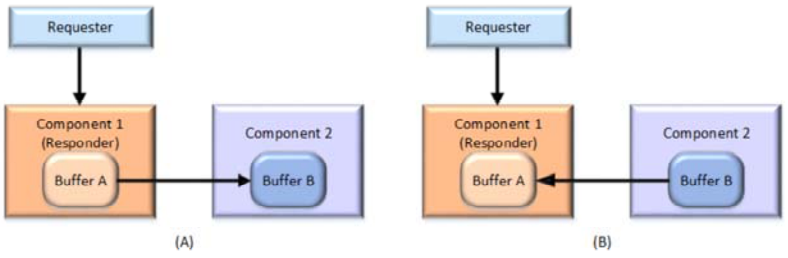
This presentation covers Gen-Z Buffer operations. Buffer operations are used to move large quantities of data between components.



Disclaimer



Buffer Operations

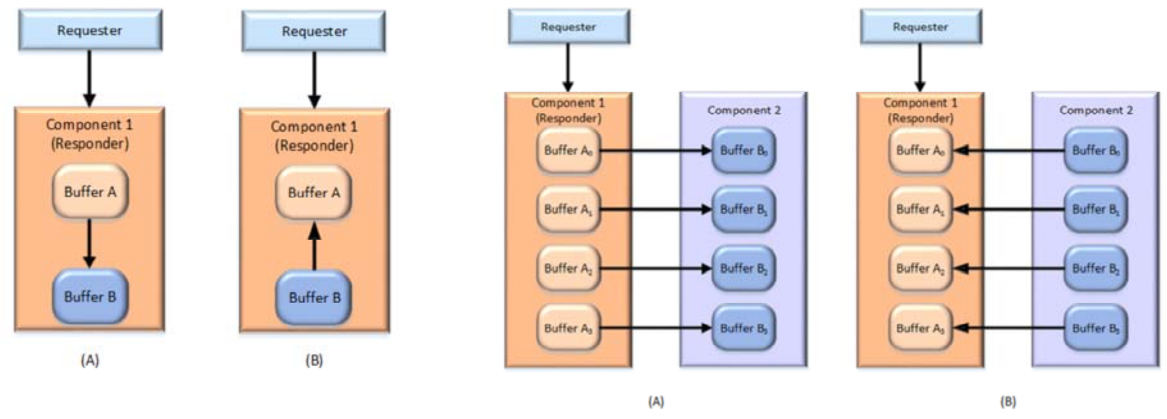


- Buffer operations enable large data movement between two buffers—Buffer A and Buffer B—or between two buffer vectors—vector A and vector B.
- Data movement is done as a put from A to B or get from B to A
- Maximum data movement size is 4 GiB
- Buffer operations can be used in point-to-point and single or multi-subnet topologies

Buffer operations are used to get data put or push up to 4 GiB of data from buffer A to buffer B, or to get or pull data from buffer B to buffer A.

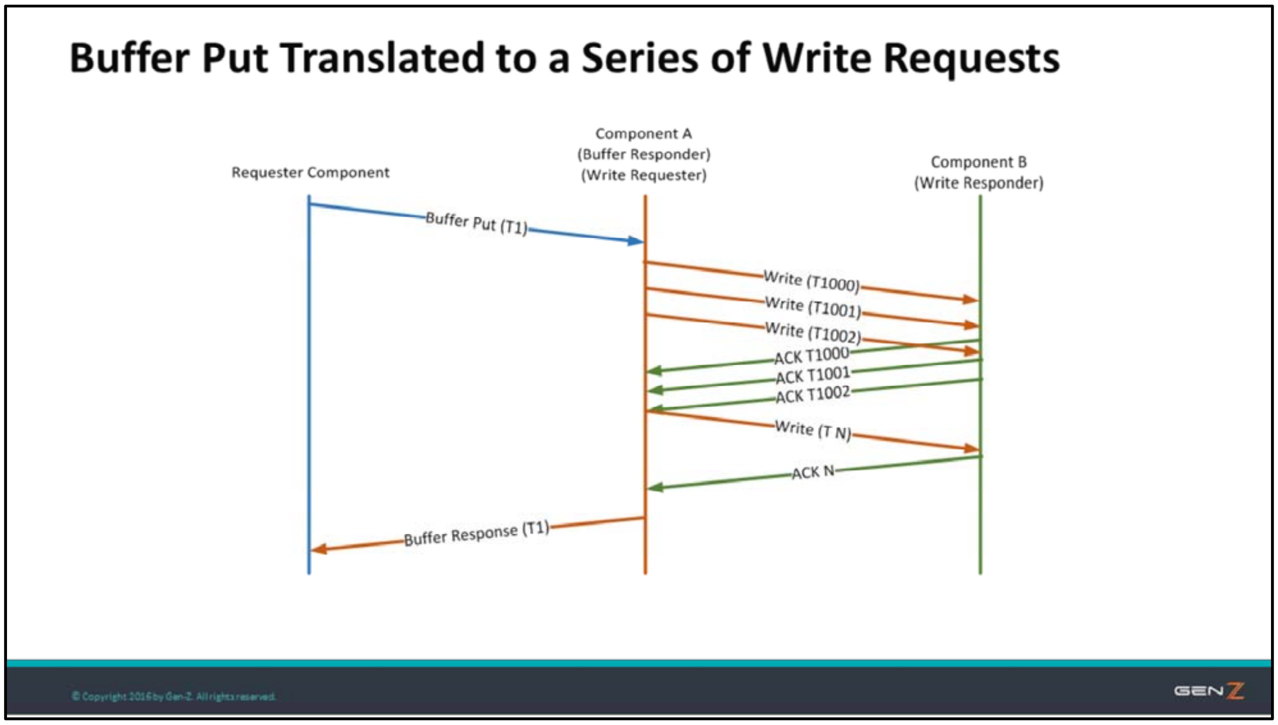
Buffer operations can used in point-to-point, single-subnet, and multi-subnet topologies.

Buffer Operations (continued)



Buffer operations can be used to move data within a component. For example, if a component contains two addressable resources (primary and secondary media), then a buffer operation can be used to put or get data from one media to the other. Further, if the Responder contains an accelerator (e.g., an encryption / compression / graph / KVS / etc. engine), then as the data is moved between the two buffers, the Responder can perform additional accelerator-specific operations in parallel (these accelerations are transparent to the application).

A vector buffer operations enables up to 4 independently-addressed buffers to be moved between components, or within a single component.



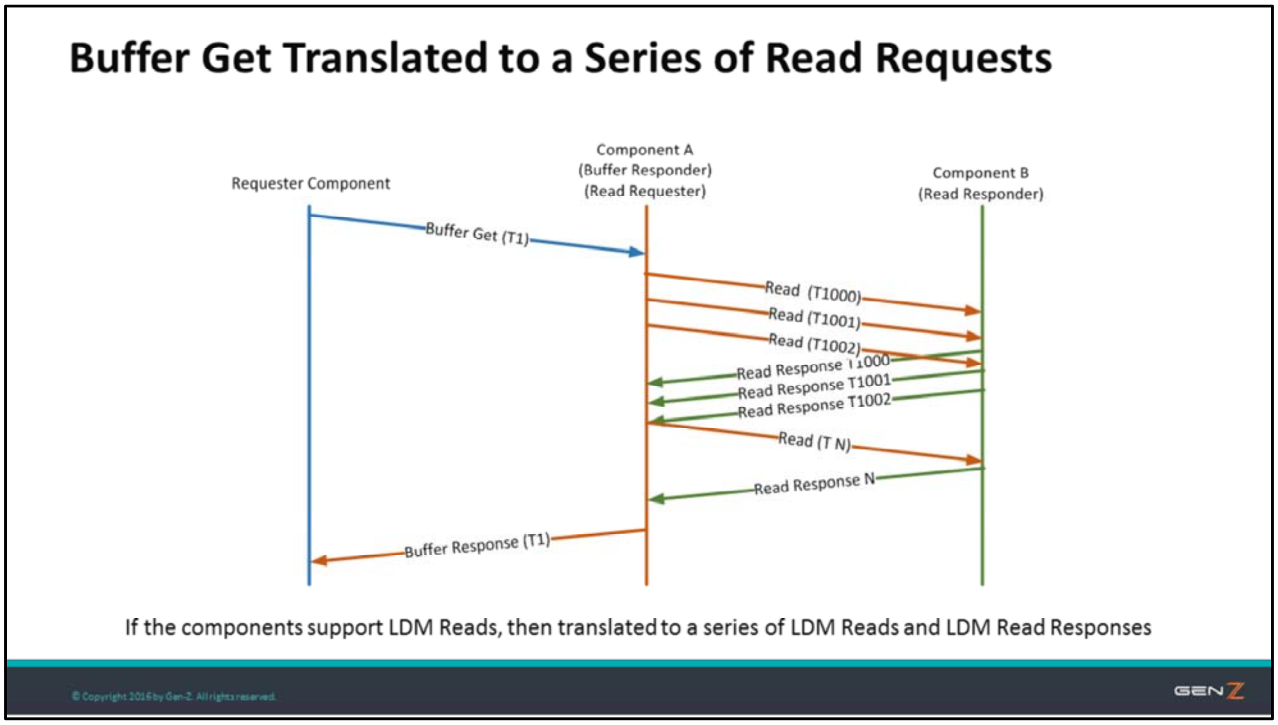
Buffer Put Translated to a Series of Write Requests

A Buffer Put operation is translated by the Responder (component A) into a series of Write requests to component B. Only the Requester and Component A need to support buffer operations; component B only needs to support normal read and write operations.

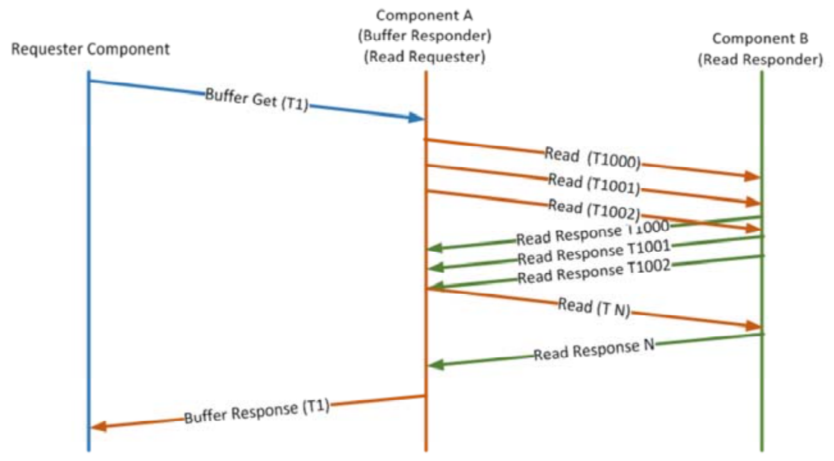
This example illustrates how buffer operations can eliminate unnecessary data movement. For example, in some file or storage solutions, the data needs to flow through the host processor and memory (this is often accomplished by having component A perform a series of DMA Writes data to host memory, signal component B when the DMA Writes are completed, and then component B performs a series of DMA Reads to pull the data down.

Though not shown in this example, the Requester could issue a Buffer Put request to instruct component A to write a buffer to the Requester's memory, i.e., the Requester contains buffer B. This provides multiple advantages:

- Eliminates the need to provision and program work queues as well as the associated control plane protocol overheads
- Eliminates per Requester static resource provisioning—a Responder supports N outstanding buffer operations usable by multiple Requesters based on workload needs
- Enables all data movement to be offloaded to the Responder. Significant software stack reductions are possible (perhaps a single processor instruction to perform a put or get operation).



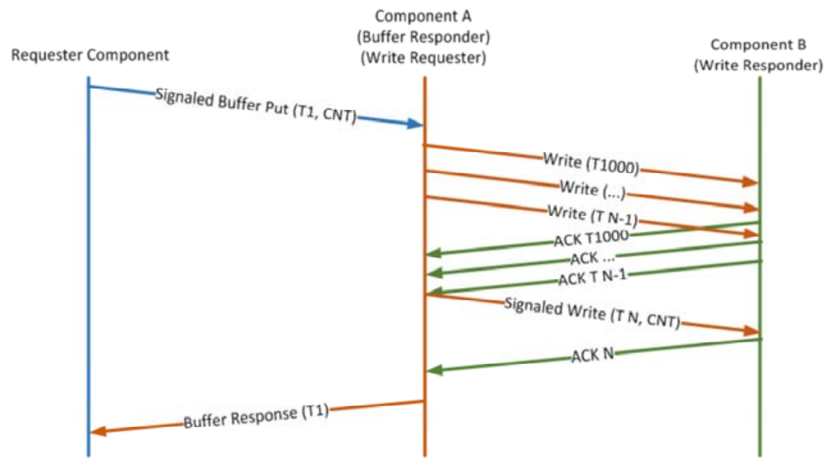
Buffer Get Translated to a Series of Read Requests



If the components support LDM Reads, then translated to a series of LDM Reads and LDM Read Responses

A Buffer Get operation is conceptually the same as a Buffer Put operation, except it uses Read or Large Read operations instead of Write operations. Buffer Get operations can improve solution efficiency by enabling data to be pulled at the rate that component A can consume the data.. This can reduce fabric congestion as well as improve component A’s responsiveness since it controls when the data is read.

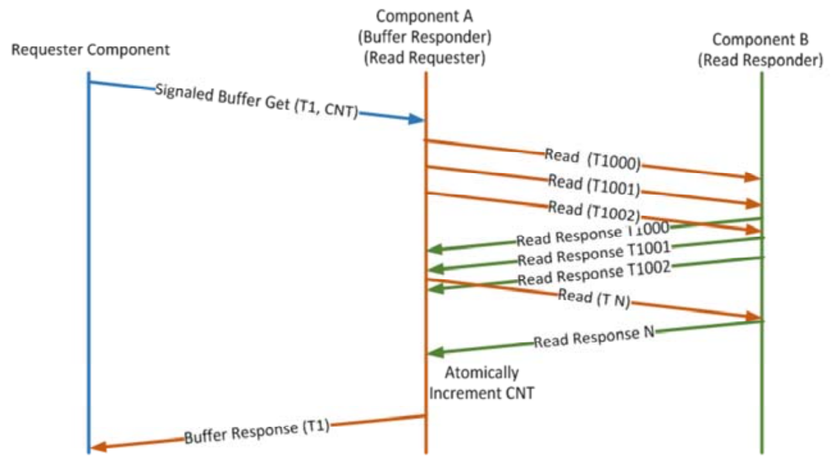
Buffer Signaled Put


Requester Component
Component A (Buffer Responder) (Write Requester)
Component B (Write Responder)
Signaled Buffer Put (T1, CNT)
Write (T1000)
Write (...)
Write (T N-1)
ACK T1000
ACK ...
ACK T N-1
Signaled Write (T N, CNT)
ACK N
Buffer Response (T1)

- Buffer Signaled Put performs a Put operation and then performs a Signaled Write
- Signaled Write enables multiple threads / components to periodically poll for buffer completion
  - Signaling eliminates the need to tightly coordinate multiple threads / nodes which can improve performance

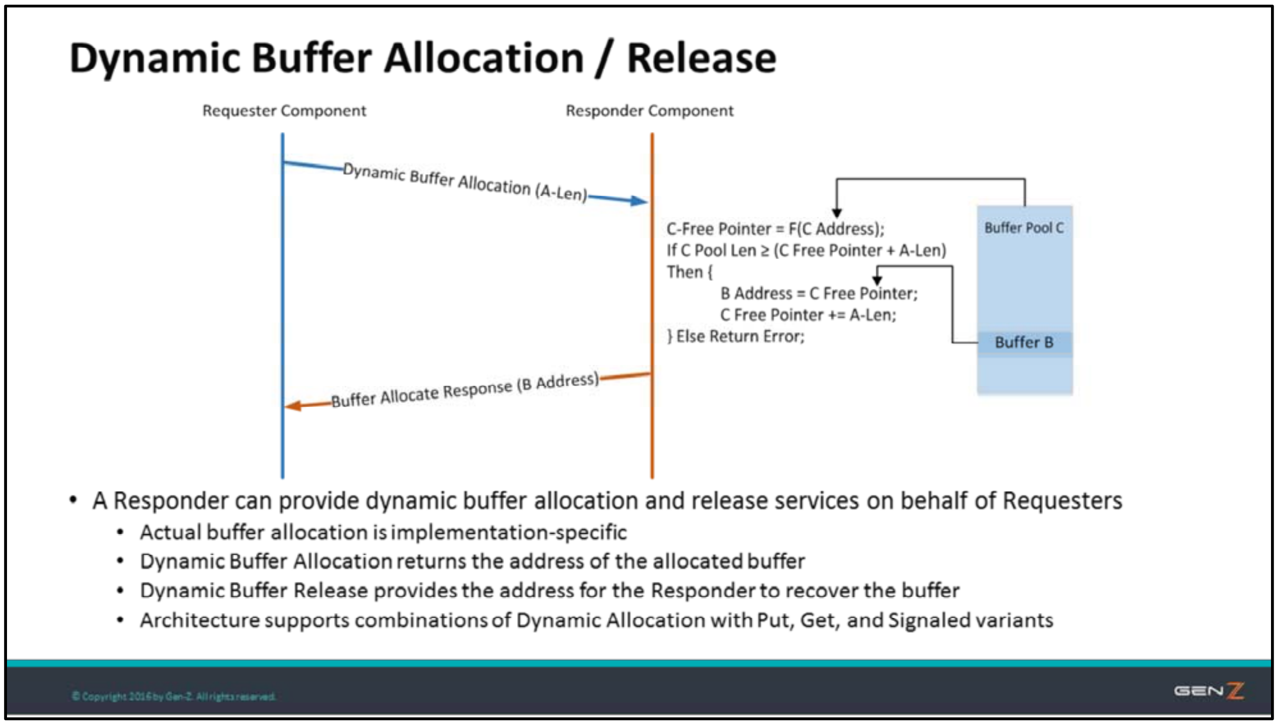
There are multiple types of Buffer operations. This slide illustrates a Buffer Signaled Put operation. A Buffer Signaled Put operation uses a Multi-Op Signaled Write to move the last bit of data. Upon receipt of a Signaled Write, the Responder atomically updates the signaled address. Components that periodically poll the signaled address will detect the data modification, and take action. Using a Signaled Write eliminates the need to explicitly coordinate the put operation completion (the Responder does even need to know which or how many other components or process threads are polling the signaled address). Eliminating coordination cost / complexity can yield significant performance gains and improve solution quality and robustness.

Buffer Signaled Get



- Buffer Signaled Get performs a Get operation and then performs a local atomic increment at the target address
  - Signaling eliminates the need to tightly coordinate multiple threads / nodes which can improve performance

A Buffer Signaled Get is the read analog of a Buffer Signaled Write. Instead of a Signaled Write as the last operation in the put, Component A performs a local atomic increment to signal the other components or process threads.



Dynamic Buffer Allocation / Release

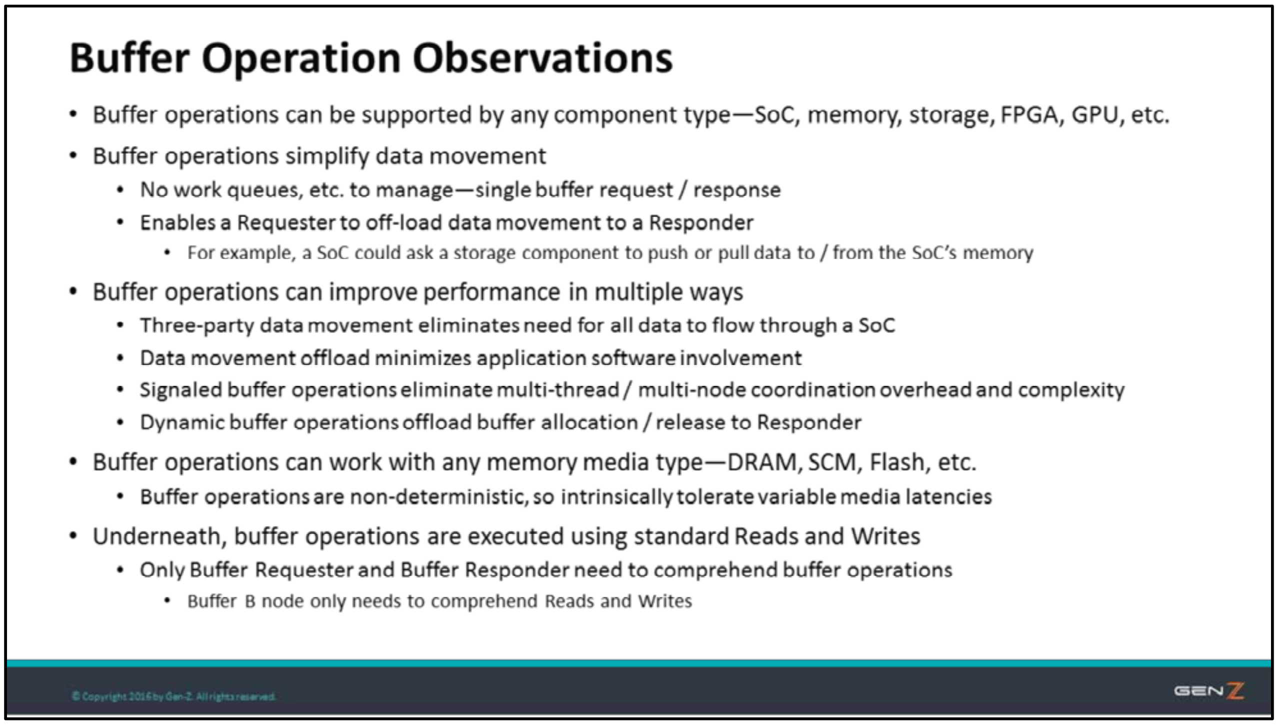
Requester Component
Responder Component
Dynamic Buffer Allocation (A-Len)
Buffer Pool C
Buffer B
Buffer Allocate Response (B Address)

```
C-Free Pointer = F(C Address);
If C Pool Len ≥ (C Free Pointer + A-Len)
Then {
        B Address = C Free Pointer;
        C Free Pointer += A-Len;
} Else Return Error;
```

- A Responder can provide dynamic buffer allocation and release services on behalf of Requesters
  - Actual buffer allocation is implementation-specific
  - Dynamic Buffer Allocation returns the address of the allocated buffer
  - Dynamic Buffer Release provides the address for the Responder to recover the buffer
  - Architecture supports combinations of Dynamic Allocation with Put, Get, and Signaled variants

GEN Z

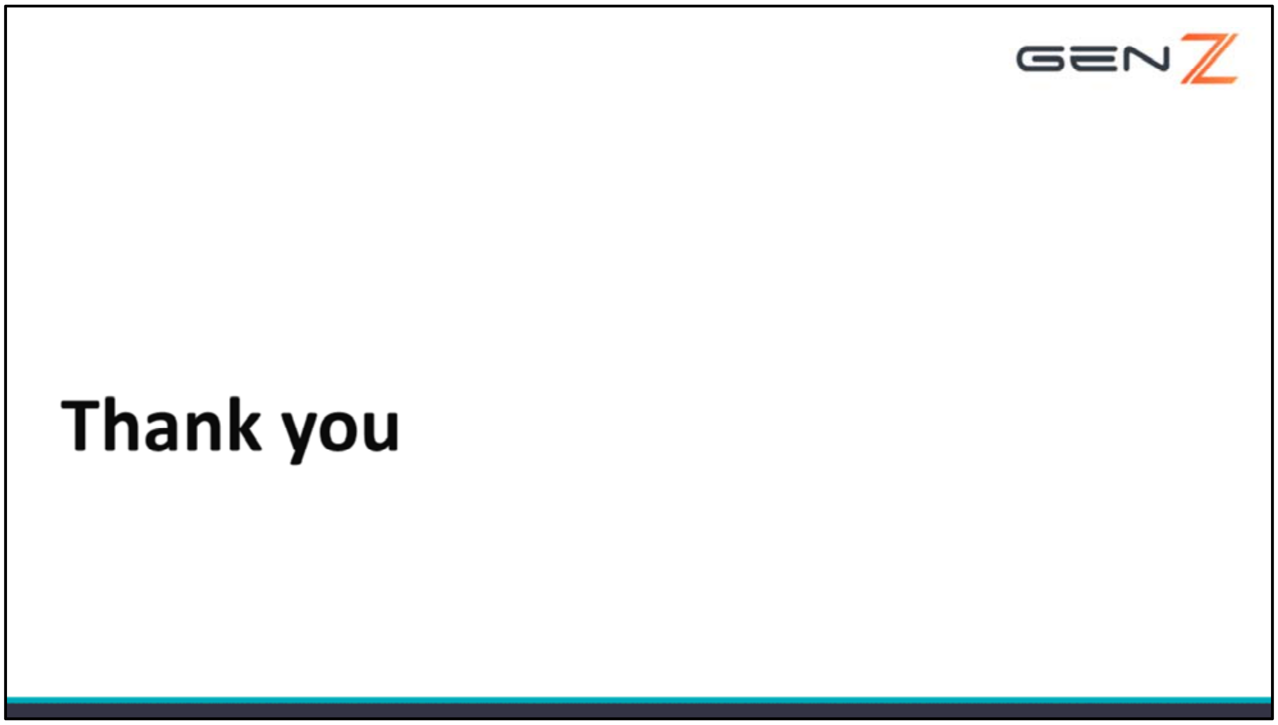
Gen-Z supports Dynamic Buffer Allocation and Release operations.   Allocation can be executed as a standalone operation or in combination with a Put, Get, or Signaled variants. Dynamic Buffer Allocation eliminates the need for a Requester to explicitly manage buffer allocation. For example, consider an application that continually receives new data objects such as pictures or videos. Instead of having to identify a storage or memory resource capable of storing each object, the Requester can simply issue a Dynamic Buffer Put to a Responder which automatically allocates the required buffer and receives the subsequent put data. If the Responder is a Transparent Router (TR), then it could transparently represent any number of memory and storage components, thus supporting very large addressable resources, variety of data preservation technologies (e.g., RAID), aggregate performance (e.g., memory interleave), memory and storage tiers, etc. Further, since many service providers analyze objects to identify faces, places, etc., signaled put and get operations can be used to automatically inform any number of accelerators to automatically analyze the object and generate additional meta data or to improve data analytics.



Buffer Operation Observations

- Buffer operations can be supported by any component type—SoC, memory, storage, FPGA, GPU, etc.
- Buffer operations simplify data movement
  - No work queues, etc. to manage—single buffer request / response
  - Enables a Requester to off-load data movement to a Responder
    - For example, a SoC could ask a storage component to push or pull data to / from the SoC's memory
- Buffer operations can improve performance in multiple ways
  - Three-party data movement eliminates need for all data to flow through a SoC
  - Data movement offload minimizes application software involvement
  - Signaled buffer operations eliminate multi-thread / multi-node coordination overhead and complexity
  - Dynamic buffer operations offload buffer allocation / release to Responder
- Buffer operations can work with any memory media type—DRAM, SCM, Flash, etc.
  - Buffer operations are non-deterministic, so intrinsically tolerate variable media latencies
- Underneath, buffer operations are executed using standard Reads and Writes
  - Only Buffer Requester and Buffer Responder need to comprehend buffer operations
    - Buffer B node only needs to comprehend Reads and Writes

Buffer operations provide numerous application functional and performance benefits. Buffer operations are more than just simple put and get data movement mechanisms. Buffer operations can be used to augment or enhance solution stacks while minimizing data movement, or can be combined with acceleration technology to provide value-add processing transparent to applications and middleware.



Thank you

This concludes this presentation. Thank you.