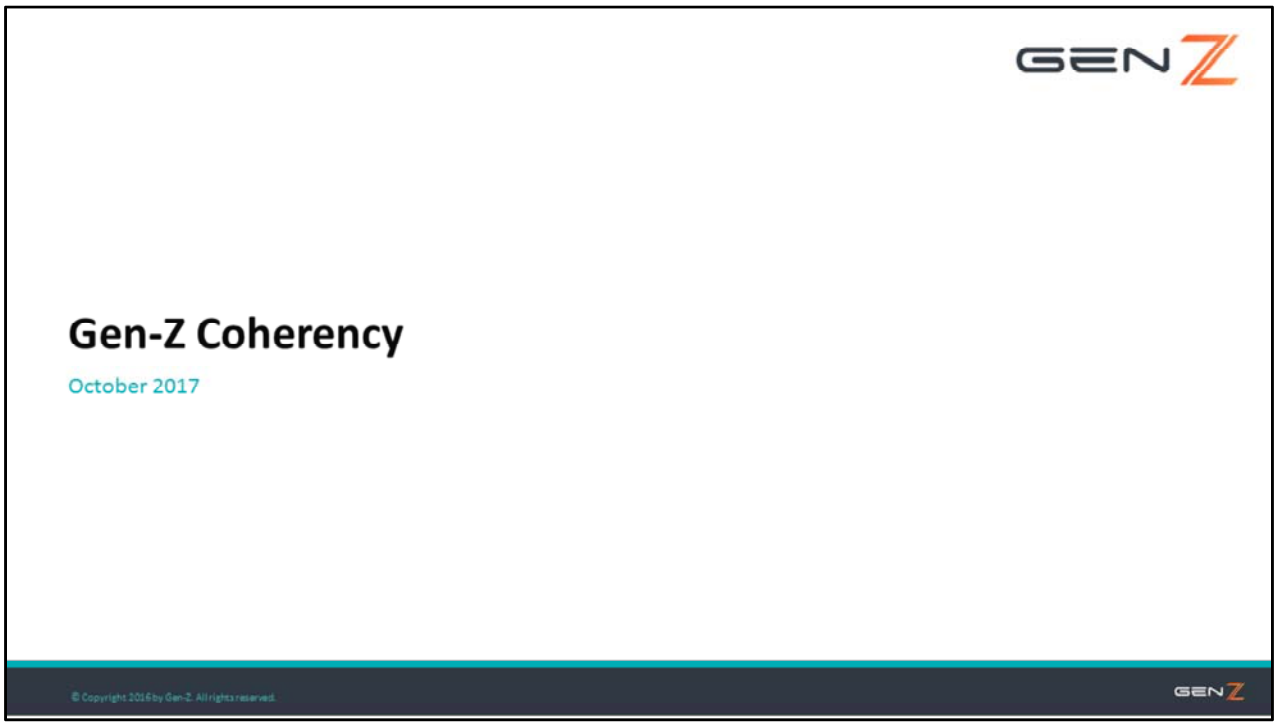


This presentation covers Gen-Z coherency operations and semantics.

Disclaimer

This document is provided 'as is' with no warranties whatsoever, including any warranty of merchantability, noninfringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Gen-Z Consortium disclaims all liability for infringement of proprietary rights, relating to use of information in this document. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

Gen-Z is a trademark or registered trademark of the Gen-Z Consortium.

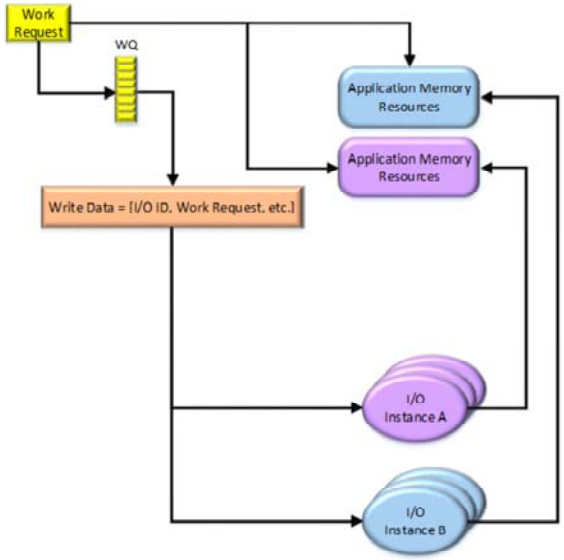
All other product names are trademarks, registered trademarks, or servicemarks of their respective owners.

All material is subject to change at any time at the discretion of the Gen-Z Consortium

http://genzconsortium.org/

Traditional I/O Work Queue Model

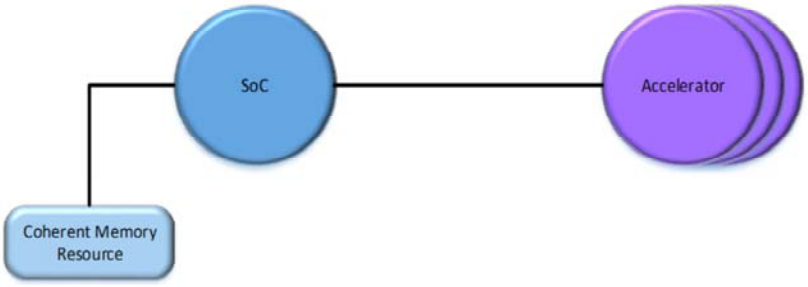
- Application memory is mapped to enable access
- Middleware / device driver:
  - Creates a set of work queues associated with one or more I/O instances that move the data to / from the component
  - Generates work queues and completion queues
  - Posts work requests and executes completion processing
- I/O component:
  - Reads work request
  - Generates a series of read / write operations to move application data to the I/O component
  - Generates a completion queue entry
  - Generates an interrupt to inform the driver
- Challenges:
  - Significant overhead and latency limit performance and what can be accelerated by I/O components
  - Requires component-specific device drivers and relatively complex OS infrastructure
  - Non-transparent



GEN Z

The traditional I/O work queue model is well-understood, highly-optimized, and pervasive. It will continue to serve the industry for many years to come. However, it comes with a cost in terms of complexity, overhead, etc. that gate performance, and limit the type of services that can be handled by an I/O component. Further, it requires the ecosystem to be aware that some services are being handled by the I/O component.

Coherency Model

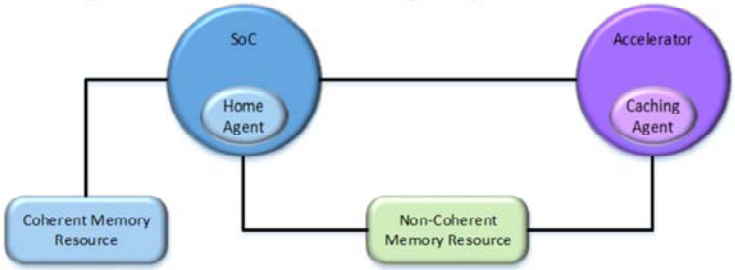


- Coherency uses memory-semantic communications
  - Data is naturally exchanged using read and write operations
    - No work / completion queues, work requests, interrupts, etc.
  - Does not require I/O device drivers nor OS I/O infrastructure
    - Coherent memory is visible to the I/O component (in this example, a set of one or more accelerators)
    - SoC updates to coherent memory cause the I/O components to take action
    - I/O components update coherent memory to communicate results

GEN Z

Coherent communications eliminates control plane communications and overheads. Instead, the application communicates a set of memory pages and data structures using an application-specific mechanism or a priori knowledge to the peer component, e.g., an accelerator. The component performs a series of coherency operations to be informed of and access new data updates.

Gen-Z Home Agent and Caching Agent

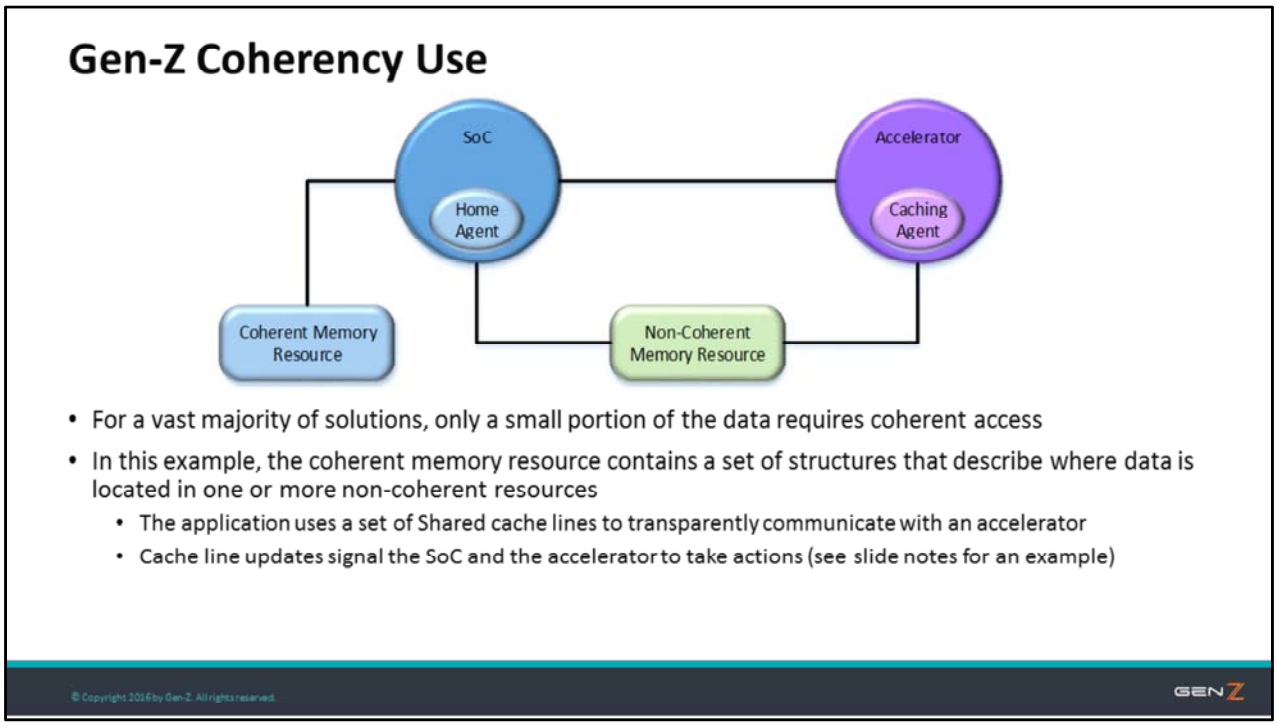


- A Home Agent is logic within a component that maintains a portion of the coherent address space associated with the blue memory component, providing data and ownership responses to Gen-Z coherency request packets.
  - Home Agent enforces coherency for any request packets targeting the Coherent Memory Resource
- A Caching Agent is logic within a component that initiates Gen-Z coherency request packets to an address space, and can maintain copies of data in its own cache structure.
- Gen-Z coherency optimized for SoC-integrated Home Agents and Accelerator-integrated Caching Agents
  - Dramatically simplifies coherency implementation
  - Isolates SoC-to-Accelerator coherency from SoC-to-SoC coherency to improve scalability and performance

GEN Z

In many respects, the Home Agent acts as a large probe filter, i.e., it filters out everything but cache line updates that the coherent component, e.g., an accelerator has acquired Exclusive or Shared access. This significantly simplifies the cost and complexity of the Caching Agent, and reduces coherent communications to a bare minimum.

Solutions may be composed of any mix of coherent and non-coherent components. For example, large data sets are located in non-coherent memory resources—DRAM, SCM, NVM, storage, etc. These can be directly accessed by any component without the cost and complexity found with coherent communications.

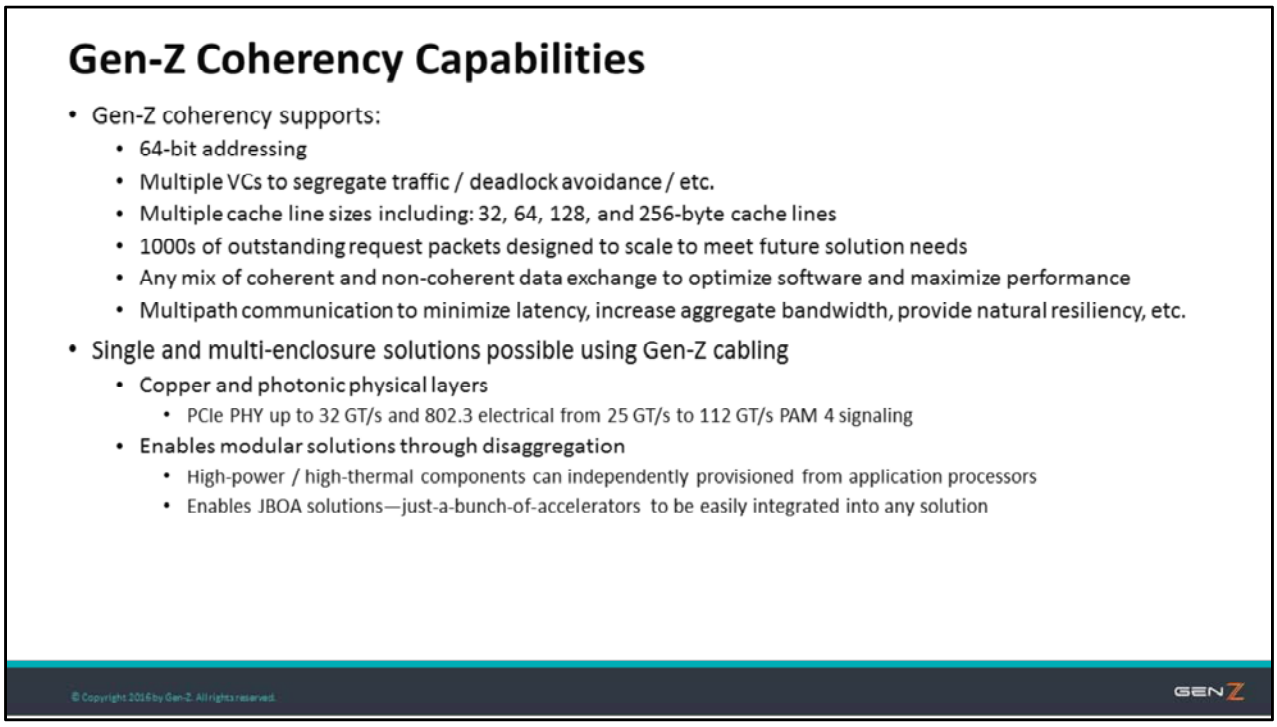


In this example, the non-coherent memory resource contains the application data set that can be directly accessed by the accelerator. To identify the specific data for the accelerator to access, the following steps are taken:

1. OS / middleware configures an address range associated with the coherent memory resource t to permit coherent access. This address range can be used to communicate control information or application data.
2. The application / middleware communicates a subset of this address range to the accelerator, e.g., a small number of pages that contain control structures used to manage application data.
3. The application / middleware communicates the address of a pointer that indicates where to locate the next set of application data to be processed is located.
4. Using this pointer, the Caching Agent within the accelerator performs a *Read Shared* to acquire a shared copy of the cache line and a current copy of the pointer value. The SoC's Home Agent updates its cache line tracking state to reflect the Caching Agent that has Shared access.
5. When the application / middleware has new data to be processed by the accelerator, it updates the pointer value. In order to coherently update the pointer, the Home Agent issues an Invalidate request to the accelerator's Caching Agent. The Caching Agent evicts the cache line from its local cache, and acknowledges the request.

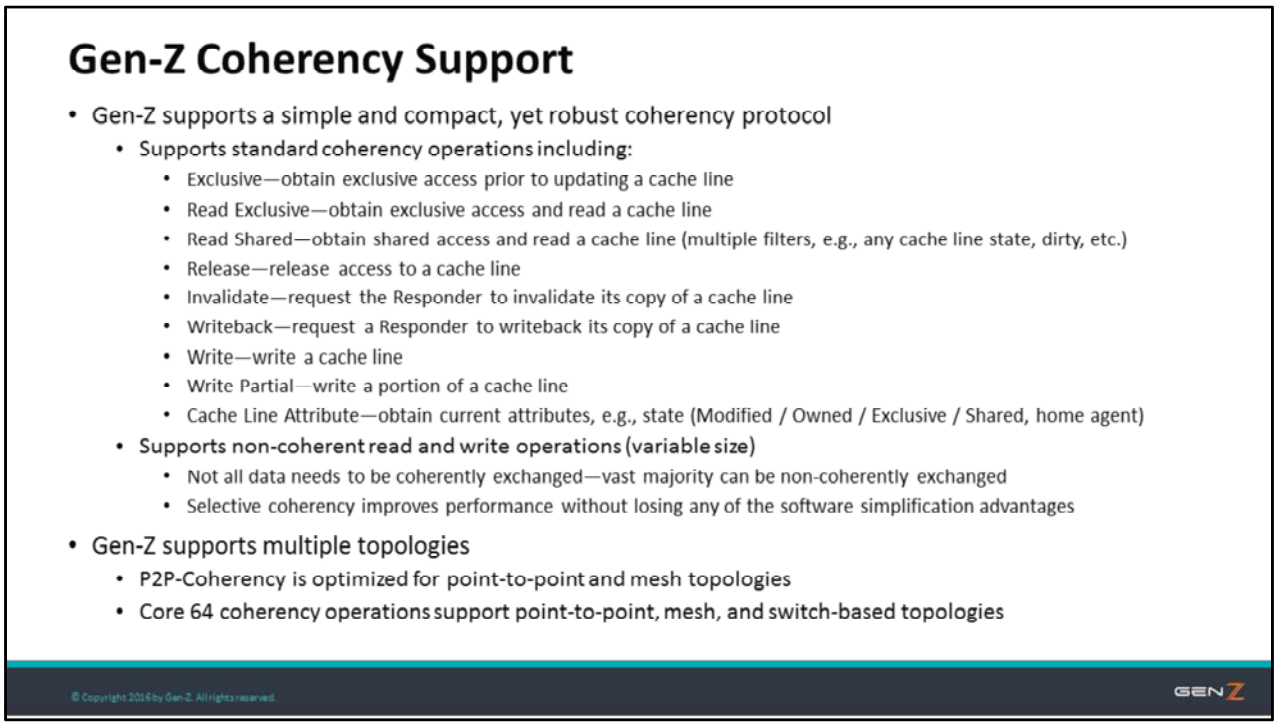
6. Upon receiving the Invalidate acknowledgment, the Home Agent permits the cache line to be updated.
7. Upon acknowledging the Invalidate request, the accelerator issues a new *Read Shared* to acquire a Shared copy of the cache line and a current copy of the pointer value. Using the pointer value, the accelerator accesses the application data. The application data could be in the coherent memory resource (accessible using any mix of coherent or non-coherent operations) or in the non-coherent memory resource (accessible using non-coherent operations).
8. When the accelerator has completed its work, it issues an Exclusive request to a coherent cache line in the coherent memory resource in order to perform a coherent write request to a flag that indicates completion.

Once the page(s) that contain the pointer and completion flag are set up, the application / middleware can transparently accelerate data located in the blue or the green memory components using any mix of coherent and non-coherent operations by repeating steps 4-8. Further, this can be applied to any number of memory components, or any mix of component types.



Gen-Z specifies a complete coherency protocol optimized for connecting any mix of components—SoC, GPU / GPGPU, FPGA, DSP, custom ASIC, etc.

The primary attributes of Gen-Z Coherency are listed above. Gen-Z coherency can be used in point-to-point, mesh, or switch topologies. It can operate over a variety of copper and photonic physical layers and signaling rates. Single and multi-enclosure solutions are possible. Multi-enclosure solutions enable high-power and high-thermal components to be disaggregated from the application processors. For example, multi-enclosure solutions enable accelerator providers to interoperate with any platform provider and to quickly and easily provision and customize the accelerator enclosure to meet solution-specific needs. This will enable Just-a-Bunch-of-Accelerator solutions (similar to JBODs) to be constructed, unlocking solution innovation and increasing customer agility.



Gen-Z coherency supports the typical coherency operations required to support a variety of coherency schemes.

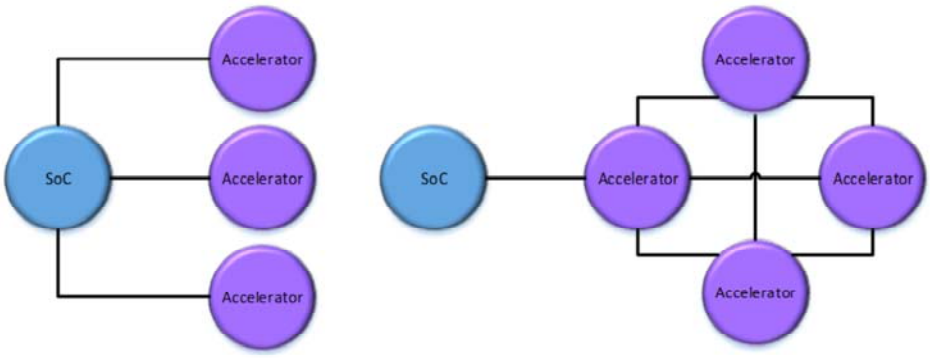
Gen-Z coherency also supports mixing coherent and non-coherent traffic and data sizes when communicating between components. In many cases, this can yield a significant improvement in protocol efficiency (more efficient data transfer) and in application performance (majority of the data exchanged between components does not require coherency semantics to be applied). For example, coherency could be applied to control operations such to application software pointers being directly passed to an accelerator via a coherent write and all subsequent data access being performed using non-coherent read and write operations.

The P2P-Coherency OpClass is optimized for use in point-to-point and mesh topologies. Components such as accelerators and SoCs can provision multiple Gen-Z links per component to provide switch-free connectivity between multiple components. Though Gen-Z switch latency should be much lower than alternative technologies (e.g., a Gen-Z switch latency should be 30-50 ns depending upon switch radix which is significantly lower than alternatives at 100-150 ns), many developers want to minimize latency as much as possible. Further, high-volume solutions that require coherency will consist of a small number of components (e.g., just a SoC and an accelerator), so point-to-point and meshing

are sufficient to meet these solution needs.

Core 64 OpClass supports point-to-point, mesh, and switch-based topologies. It supports the same coherency operations as supported with P2P-Core. It is intended for larger scale solutions or solutions containing a mix of coherent and non-coherent components, e.g., a memory-centric architecture where all components need to access shared memory or shared compute resources and only a subset need to communicate using coherency operations.

Point-to-Point / Meshed Topology Support



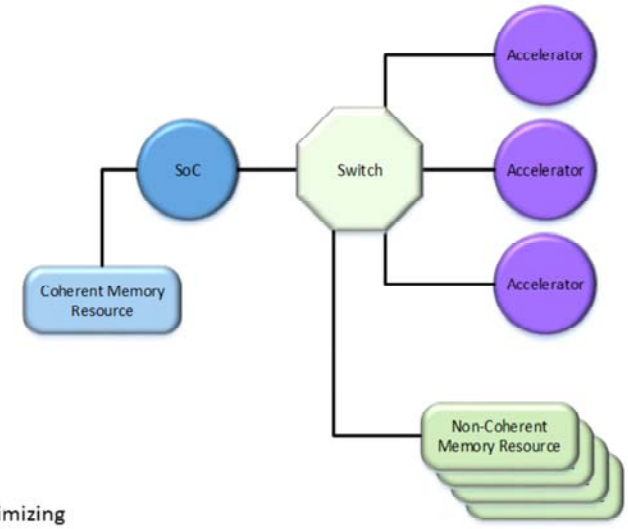
- Range of point-to-point and meshed topologies supported
- Optimized for SoC-to-accelerator or accelerator-to-accelerator communications
- Can intermix coherent and non-coherent data movement

GEN Z

P2P-Coherency can be used to connect components in a variety of point-to-point and meshed topologies. Components can be provisioned within a single enclosure or in multi-enclosure solutions attached through copper or optical cable solutions. Gen-Z architecture enables a component to support up to 4096 interfaces and links, thus provides more than enough connectivity to meet any solution's needs. This enables P2P-Coherency components to provision as many links as necessary to provide any level of connectivity.

Switch Topology Support

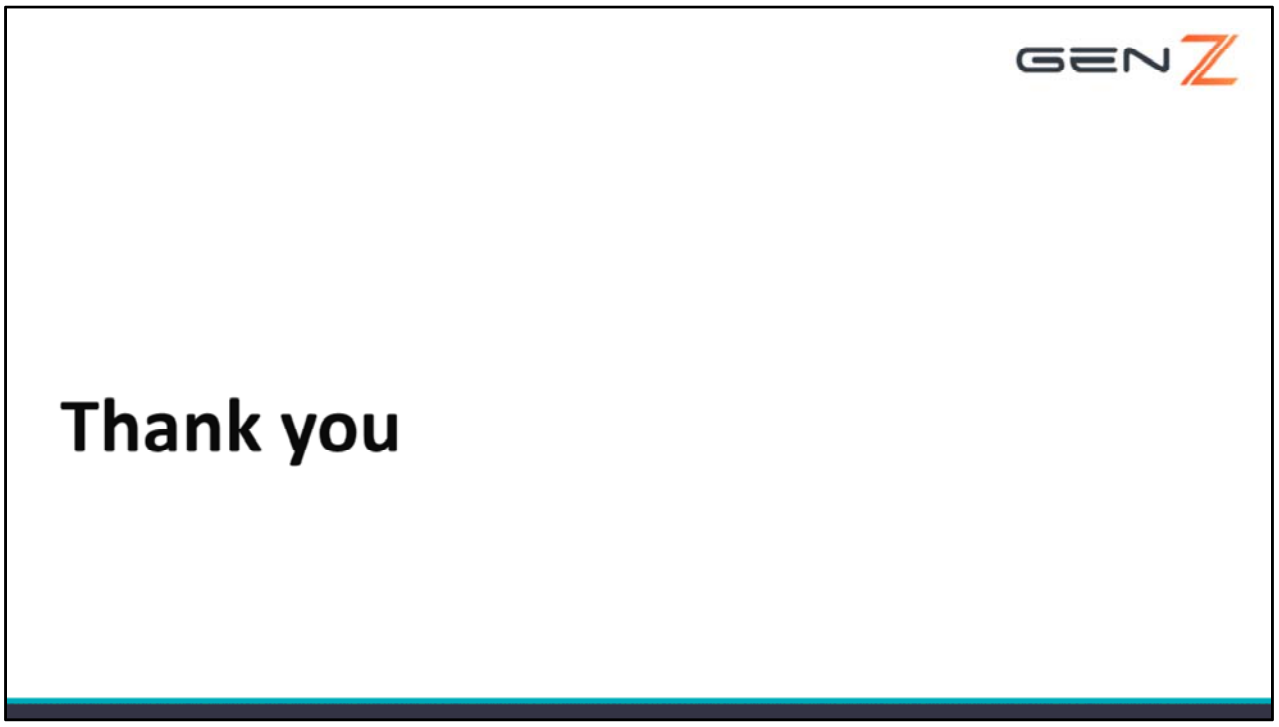
- Range of switch topologies supported
- Gen-Z switch latency will vary by radix
  - 8 interfaces @ ~10 ns
  - 16-32 interfaces @ ~30 ns
  - 32-64 interfaces @ ~50 ns
- Supports any mix of component types
- Supports processor-centric architectures
  - Components access only coherent memory resources
- Supports memory-centric architectures
- Components directly access non-coherent memory resources to minimize data access latency
- Supports any mix of coherent and non-coherent communications
  - Improved solution performance and scalability by minimizing the amount of data moved using coherent operations



GEN Z

Core 64 OpClass supports the same coherency operations as well as variable-length non-coherent read and write operations as P2P-coherency. Selective use of coherent operations as well as the ability to intermix any component types provides maximum solution composition flexibility without compromising performance. Core 64 can be used in single and multi-enclosure configurations to scale and meet any solution's needs.

Gen-Z switch latency is a key point of differentiation. Switch latency in alternative technologies range from 120-150 ns. A Read Exclusive-Read Response operation through a single 32-interface switch will incur ~60 ns of switch-specific latency compared to 240-300 ns of switch-specific latency.



This concludes this presentation. Thank you.