SNIA DEVELOPER CONFERENCE
SDC 25
By Developers FOR Developers

Hyatt Regency Santa Clara, CA
September 15-17, 2025

Simulating CXL.mem for Fun and Profit

Diman Zad Tootaghaj, HPE Labs

www.sniadeveloper.org

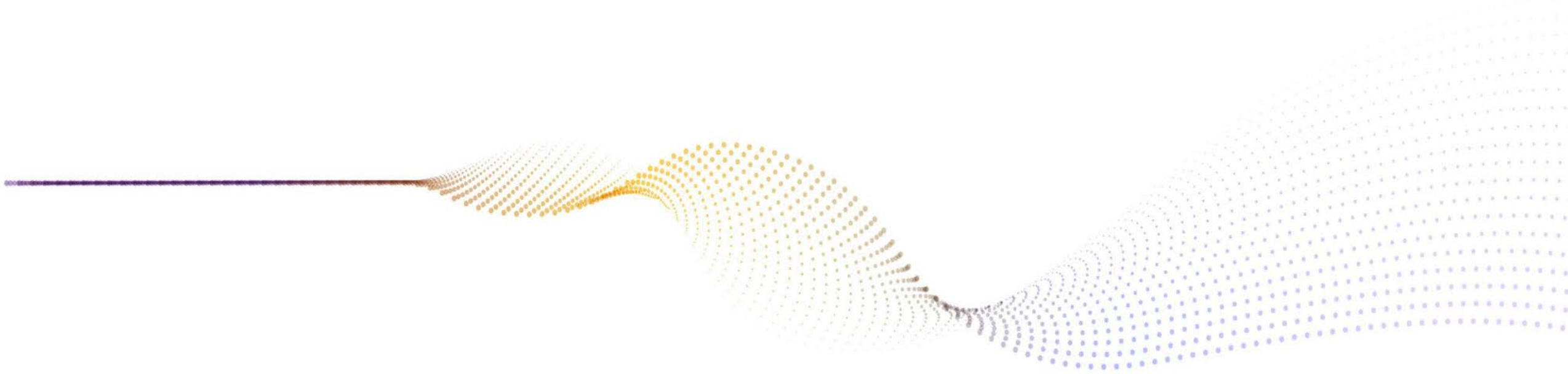
HPE Labs

About Me

- Senior Researcher at Hewlett Packard Enterprise Labs in California.
- Ph.D. in Computer Science and Engineering from The Pennsylvania State University.
- research focuses on advancing computer networks and distributed systems, with interests spanning serverless computing, sustainability in computing, network programmability, SmartNICs, SD-WAN, edge-as-a-service platforms, serverless orchestration and auto-scaling, CXL, optimization algorithms, and consensus protocols.

Agenda

- CXL Primer
- Cache Coherency
- Our CXL.mem Simulator
- Use Cases of the Simulator

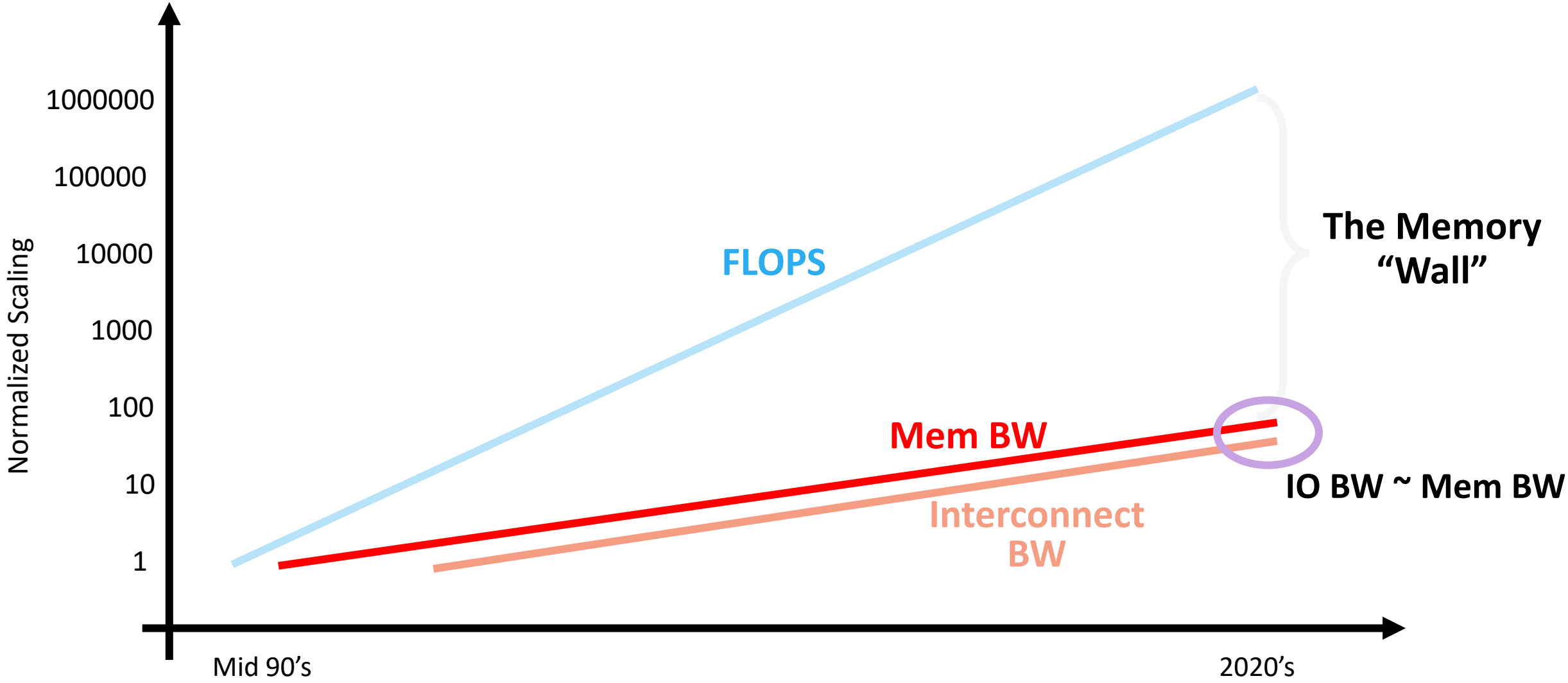


CXL Primer

HPE Labs

SDC 25

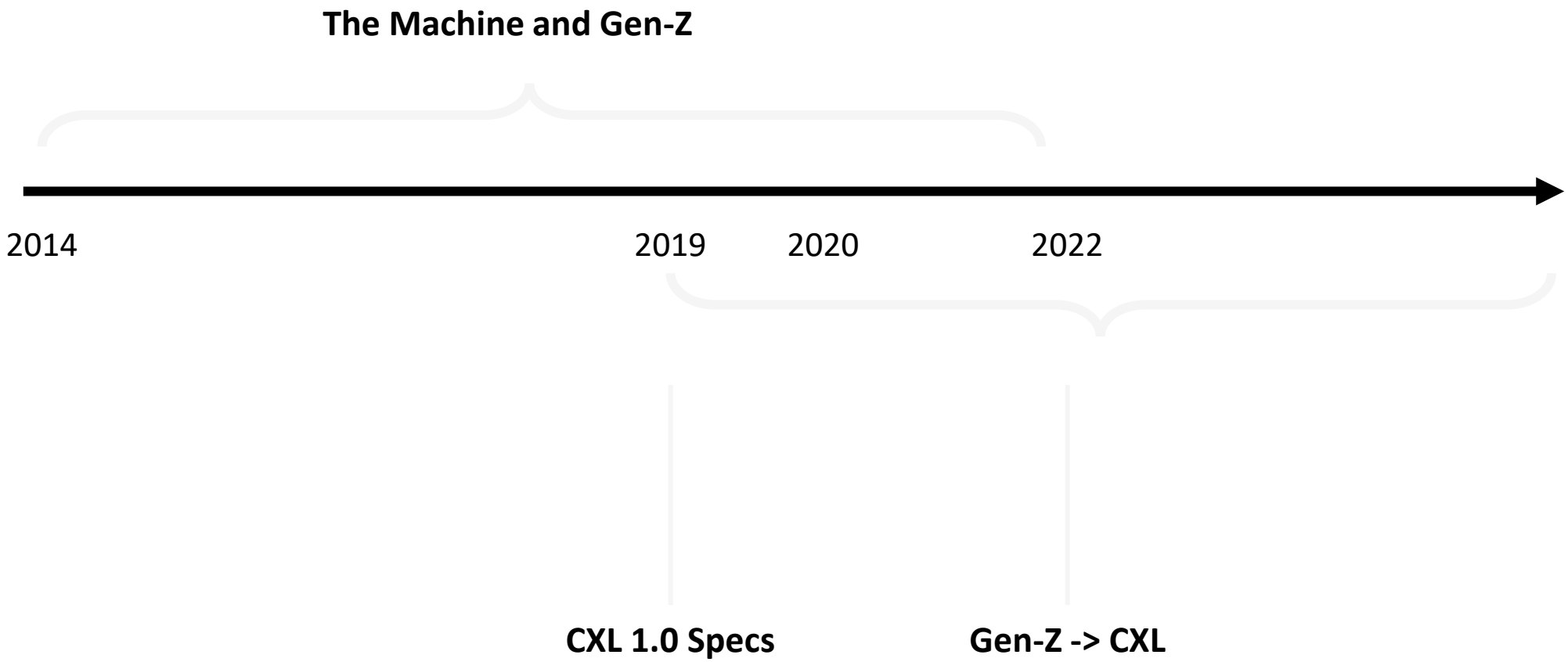
The Need for Memory Semantic Fabrics



HPE Labs

SDC 25

HPE and Memory Semantic Fabrics



HPE Labs

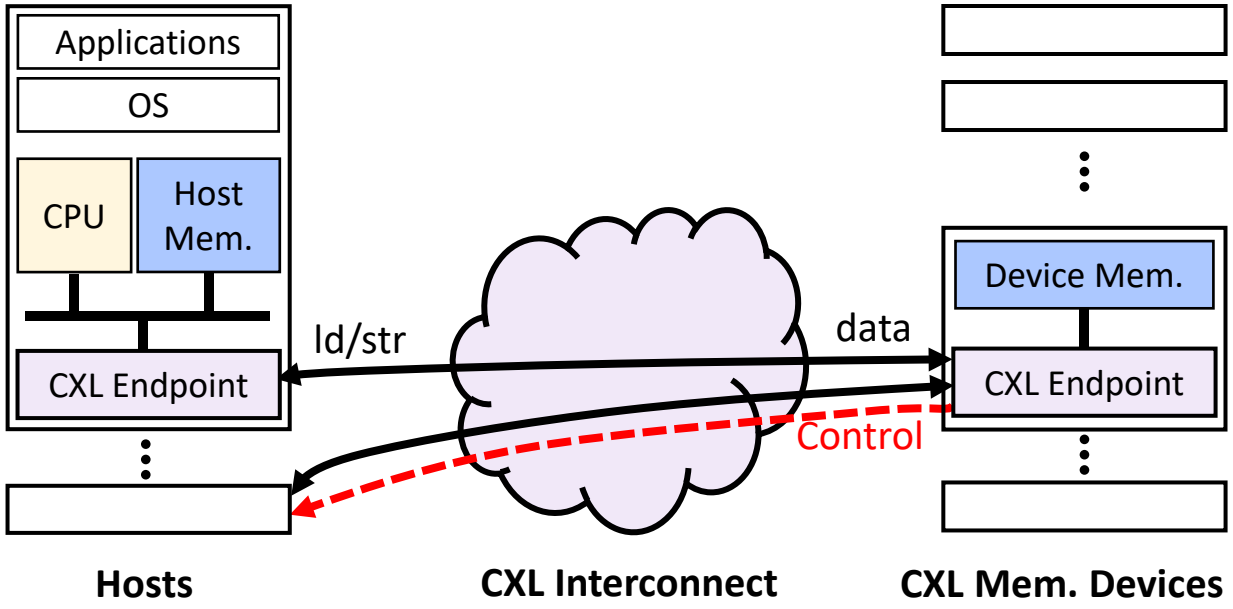
SDC 25

Compute eXpress Link (CXL)

- PCIe-based *cache-coherent* interconnect
- Contains three protocols: **`io`**, **`cache`**, and **`mem`** that specify interactions among *hosts*, *devices*, and *accelerators*

Compute eXpress Link (CXL)

- CXL.mem
  - uses **memory semantics** (i.e., `ld` and `str`) → no application modifications
  - enables memory **pooling** and **sharing**

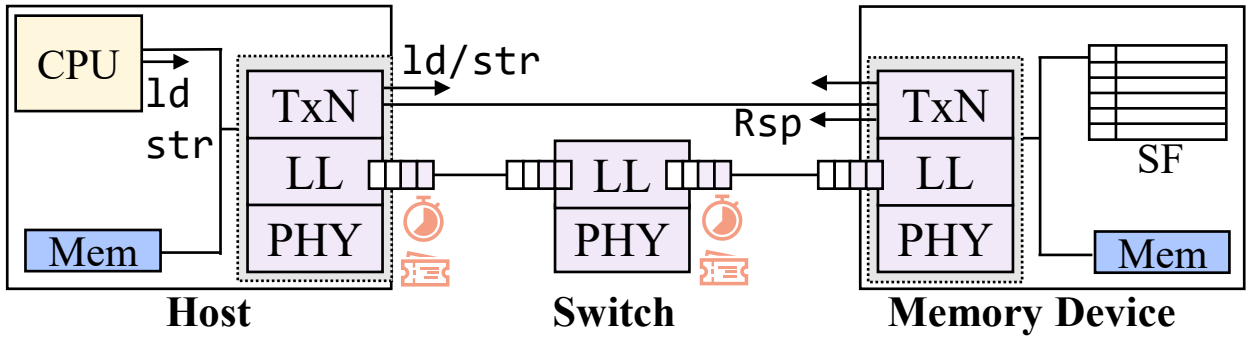


HPE Labs

SDC 25

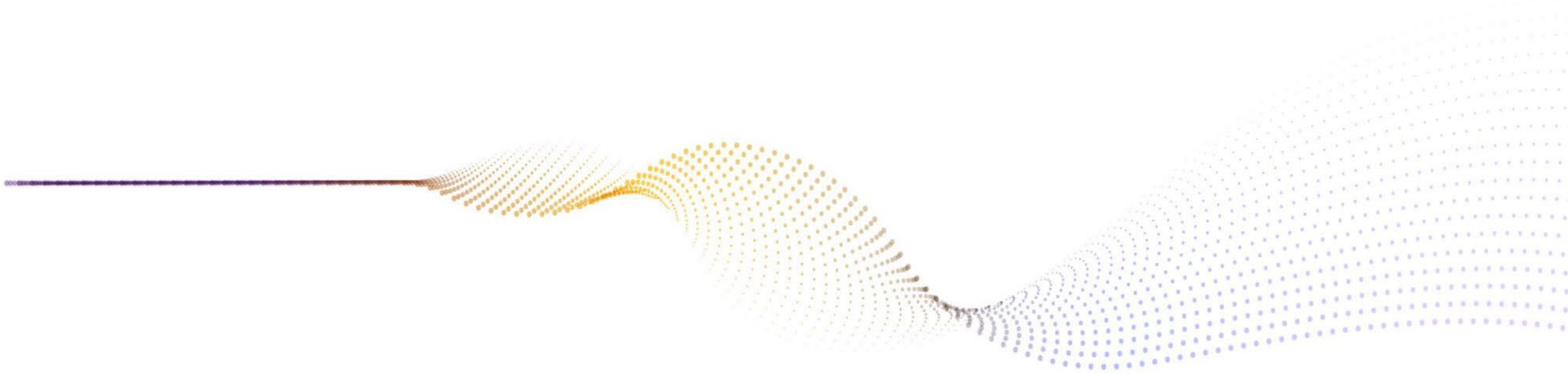
Compute eXpress Link (CXL)

- CXL protocols span three layers:
  - **Transaction layer:** Cache coherency protocol, access serialization, snoop filter, back invalidation
  - **Link layer:** Credit-based flow control, LLR
  - **Physical layer**



HPE Labs

SDC 25



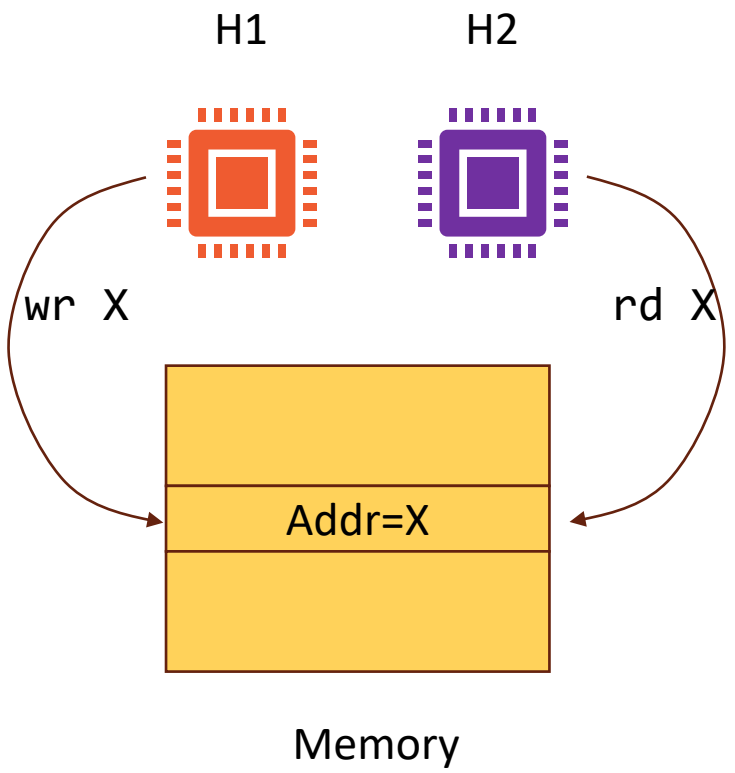
Cache Coherency

HPE Labs

SDC 25

Cache Coherency: Distributed Shared Memory

- Two hosts are sharing memory X
- If H1 writes to X and H2 reads from X,
  - H2 must read the written value by H1
  - If the read and write are sufficiently spaced
- Writes to a single address are serialized

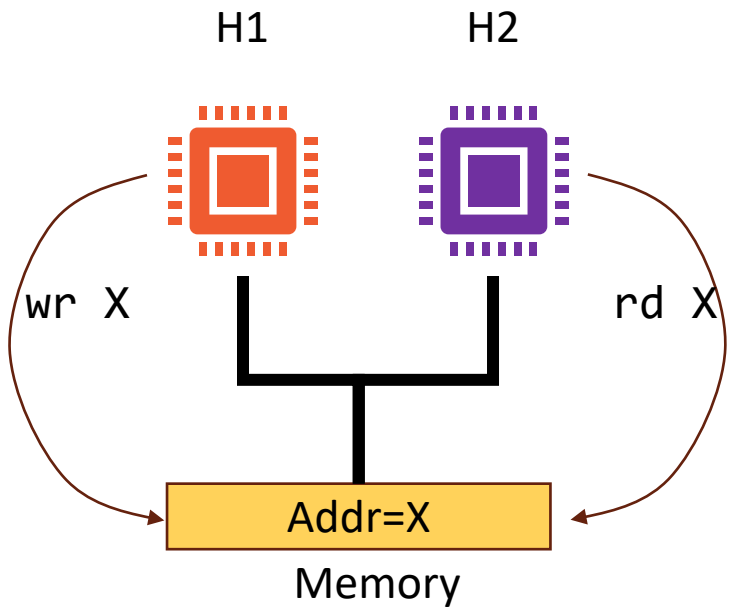


HPE Labs

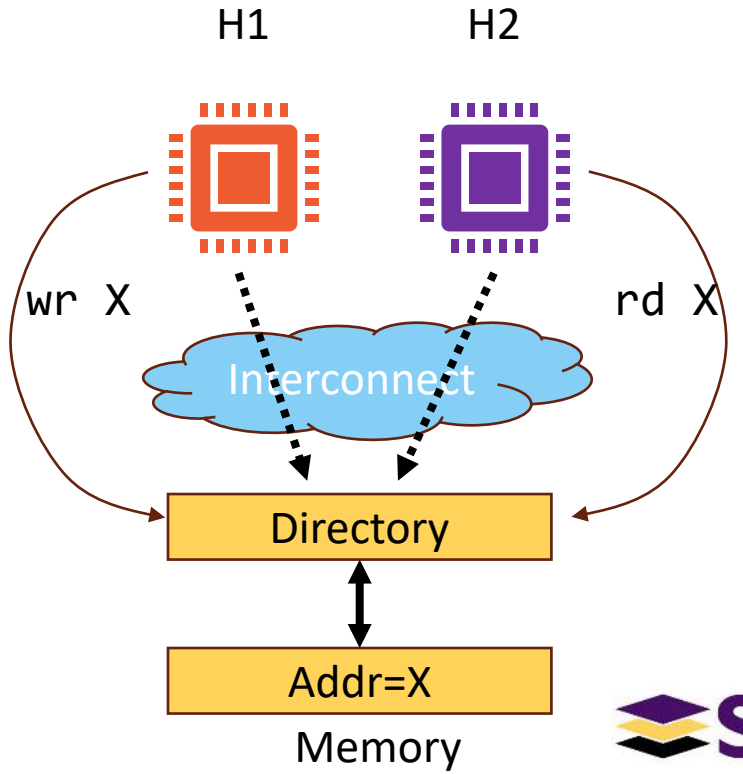
SDC 25

Two Main Approaches

- Snooping
- Broadcast all Req/Resp over a bus
- Hosts react to each Req/Resp



- Directory-based
- Data and coherency info are stored in a central location (a Directory)



HPE Labs

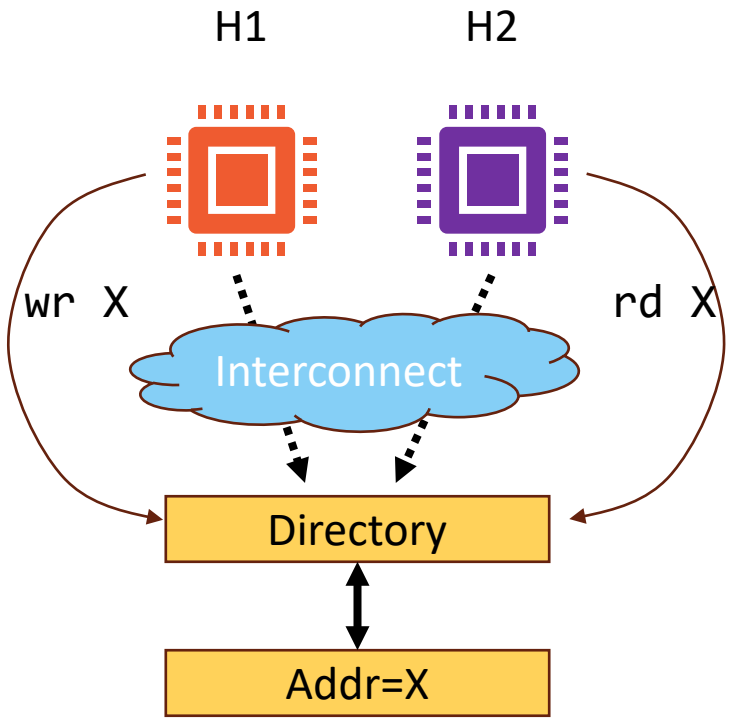
SDC 25

Cache Coherency Protocols

- State of a cache line:
  - **M**odified: present only in host memory, and is modified
  - **E**xclusive: present only in host memory, and is not modified
  - **S**hared: present in multiple host memories, and is not modified
  - **I**nvalid: unusable
- List of Sharers of a cache line:
  - The hosts sharing that line
- The protocol specifies when a cache line moves from one state to another (finite state machine)

Cache Coherency in CXL

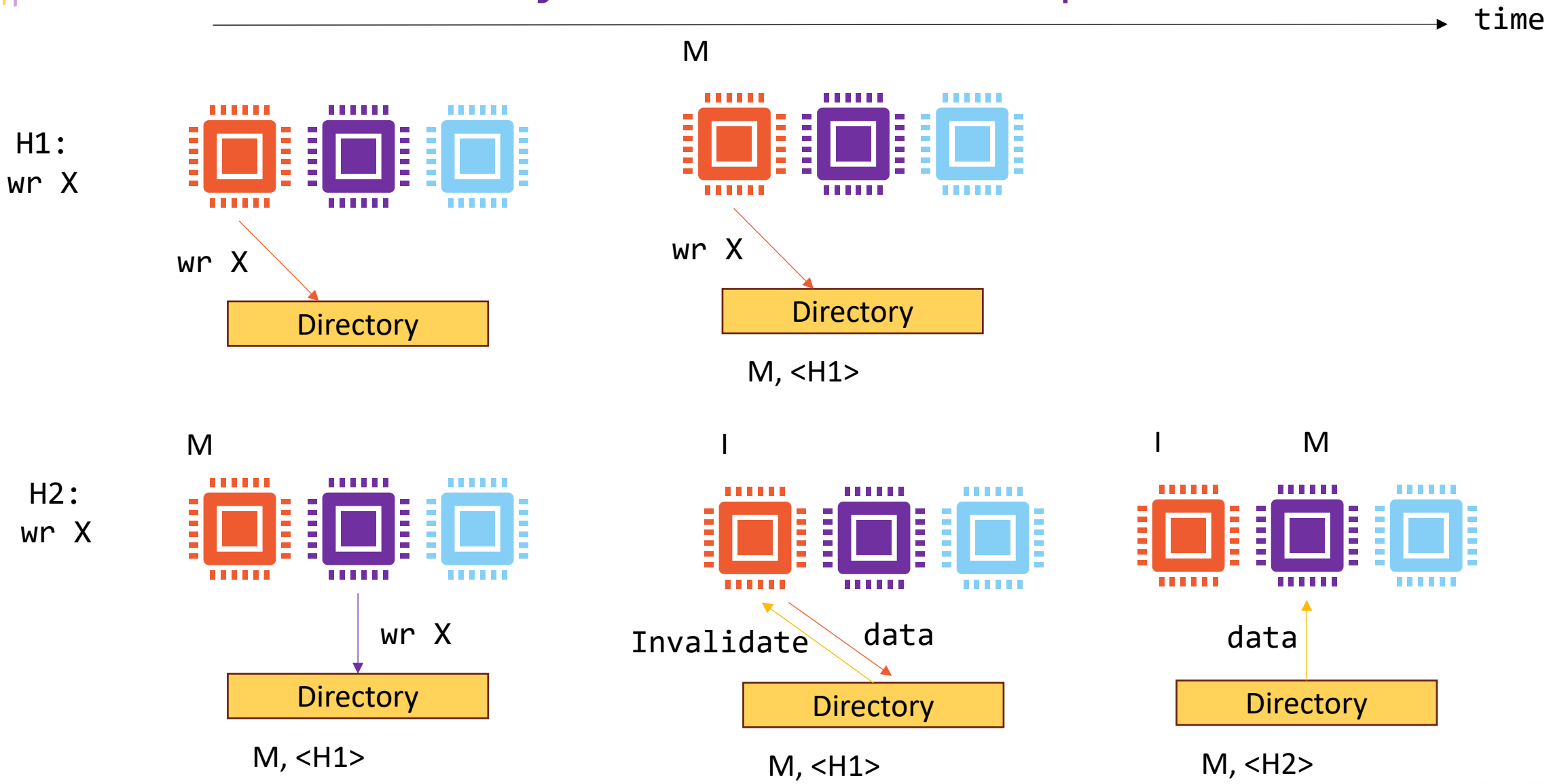
- Implemented in the *Transaction Layer*
- CXL relies on a directory-based approach to implement **sharing**
  - Better scalability
  - Less bandwidth overhead
- Accesses to different cache lines can be processed in **parallel**



HPE Labs

SDC 25

Cache Coherency Protocols: Example #1



HPE Labs

SDC 25

Key Observations

- Concurrent accesses to the same cache line need to be **serialized**
- Transactions **traverse the network** to reach memory devices
- One transaction can result in **several messages** over the network
- Memory and hosts may **need to wait** till a transaction being processed

Therefore…

CXL's cache coherency
imposes **delay overheads** on apps

How to **quantify** this delay?!

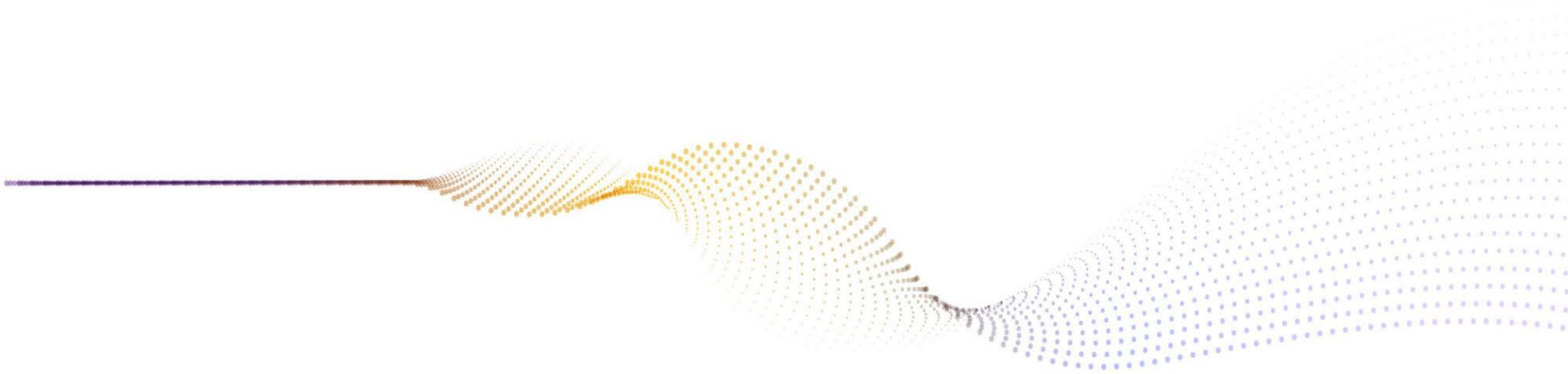
- Building a CXL hardware testbed is expensive
  - Time to deliver CXL components is long
  - Cost is high for scale-out the testbed

- But

  CXL 3.0+ hardware components don’t even exist!

This Talk…

- Simulate the CXL.mem protocol
- Validate our CXL.mem simulator
- Analyze the benefits/overheads of CXL
- Validate new CXL ideas and algorithms



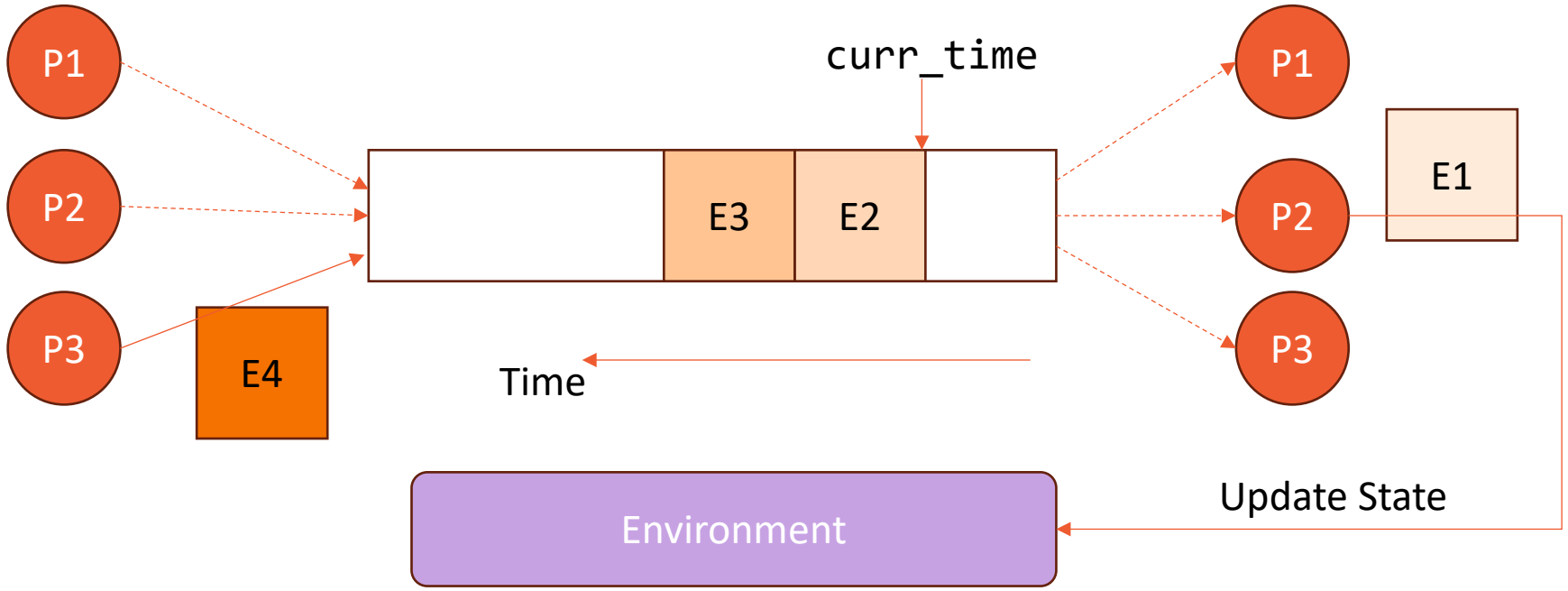
The CXL.mem Simulator

HPE Labs

SDC 25

Discrete Event Simulation (DES)

- **Processes** interact with each other via a shared **Priority Queue**
- P1 produces an E1 at time T1
- P2 consumes that Event at T2



HPE Labs

SDC 25

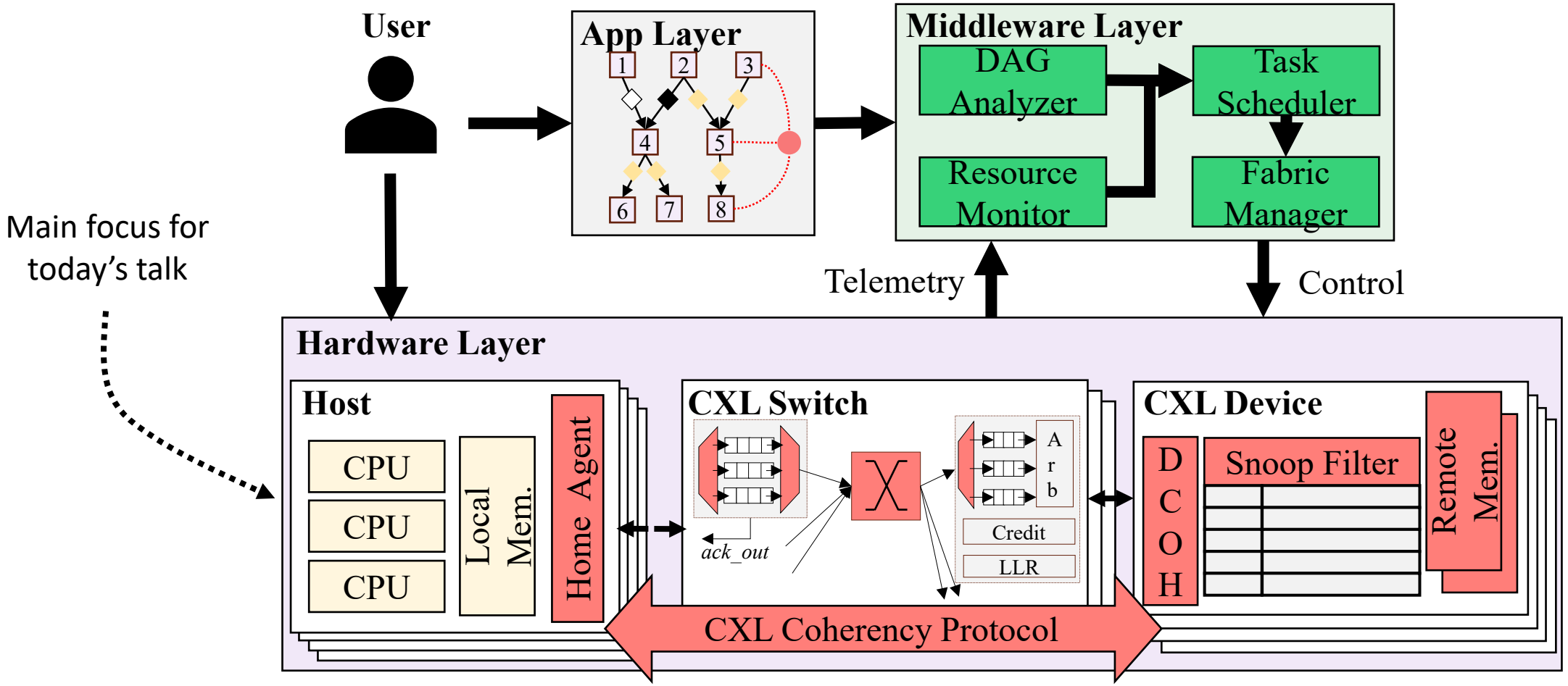
SimPy

- A DES framework written in Python
- A Process is defined as a **Python generator**
- Processes interact with other via **shared resources** constructs
  - E.g., Store

- The Priority Queue of the DES is mostly hidden from the user

- We built our CXL.mem simulator on top of SimPy

Overview of Our Simulator

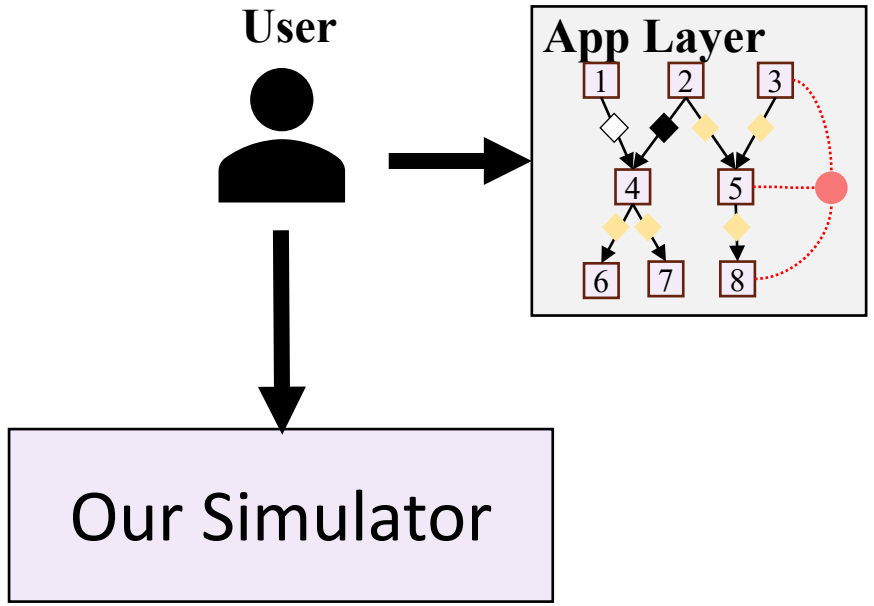


HPE Labs

SDC 25

Overview of Our Simulator

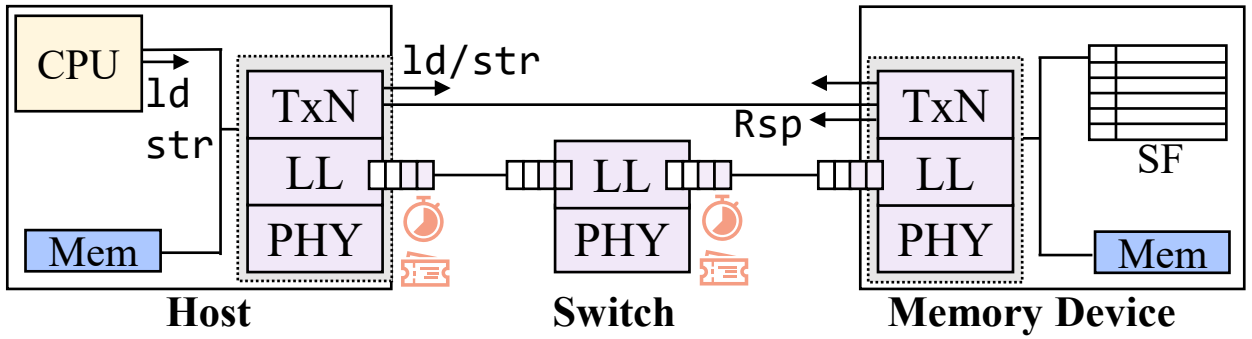
- Apps are represented as DAGs
  - A node is a unit of execution
  - An edge is a data dependency
- **Compute instructions** are simulated using waiting functions
- **Memory instructions** are passed to our simulator to calculated their execution times (similar to *SST Motifs*)



HPE Labs

SDC 25

Main Components



- **Link Layer**
- Port
- Buffer
- Link

- **Transaction Layer**
- Cache coherence
- De/Packetization

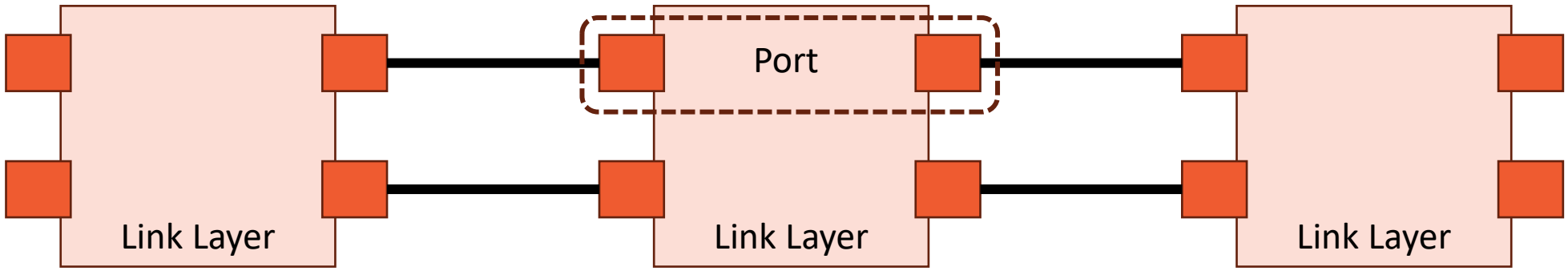
- **Other Components**
- CPU (RTC)
- Memory
- PHY retimers

HPE Labs

SDC 25

Link Layer (LL)

- Memory devices, hosts, and switches have the same LL processing
- Our Simulator
  - breaks down the implementation into main components
  - composes several components to construct a switch or an endpoint
- The user can configure each component individually as needed

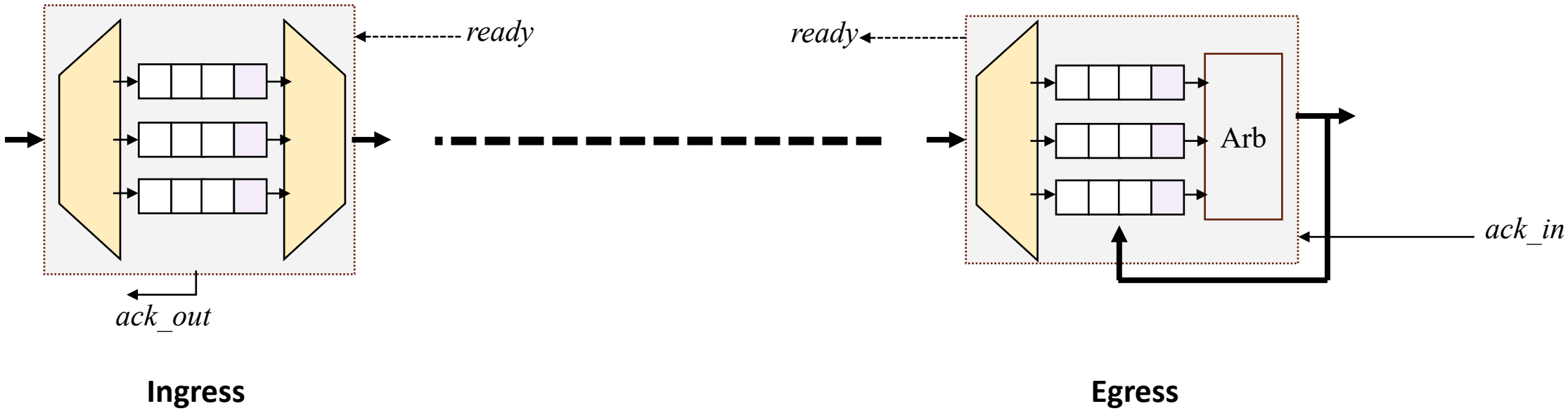


HPE Labs

SDC 25

Link Layer (LL): Port

- Each LL Port consists of:
  - Ingress Buffers
  - Egress Buffers
- # Buffers = # Virtual Channels
  - To avoid deadlocks at LL

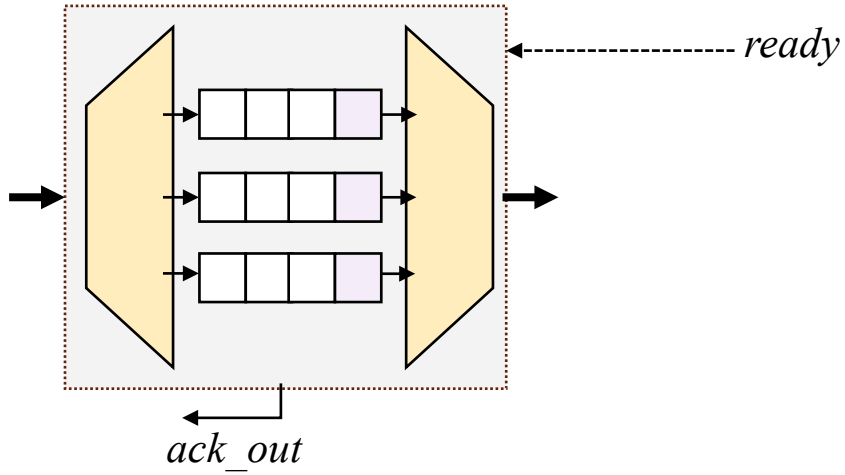


HPE Labs

SDC 25

Link Layer (LL): Ingress Buffer

- **Forwarding**
  - Receives packets from upstream components
  - Sends packets to the next Egress Buffer
  - Uses a TCAM to decide the next Egress Buffer
- **Credit Management and Reliability**
  - Sends ACK packets with available credits
  - Credit Size = Packet Size

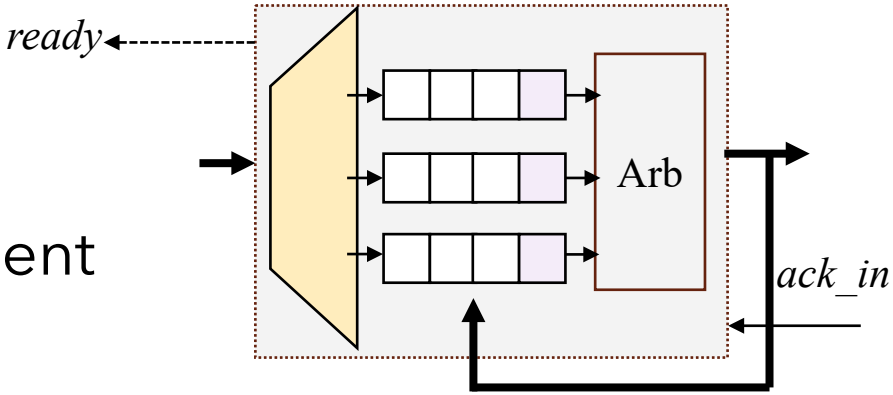


HPE Labs

SDC 25

Link Layer (LL): Egress Buffer

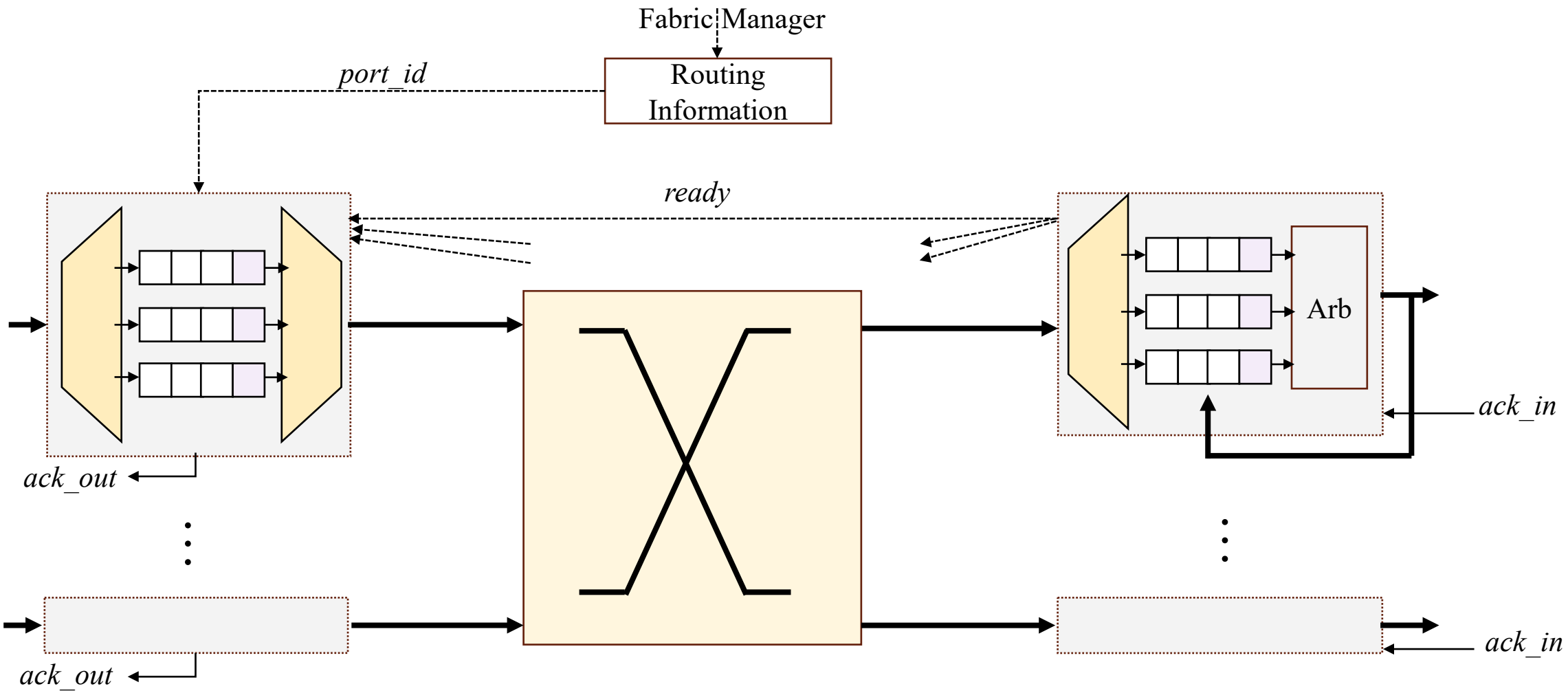
- **Forwarding**
  - Receives packets from upstream components
  - Sends packets to the next downstream component
    - *iff* there is Available Credit
- **Link Layer Retry (LLR)**
  - Maintains a **timer** for each tx packet
  - If an ACK is not received within a period, the packet is retransmitted
- **Credit Management**
  - Receives ACK from downstream components (e.g., Ingress Buffer)
  - Updates the Available Credit size



HPE Labs

SDC 25

Link Layer (LL): Switch

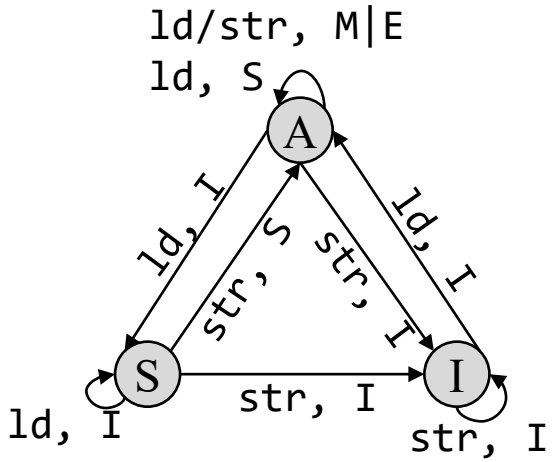


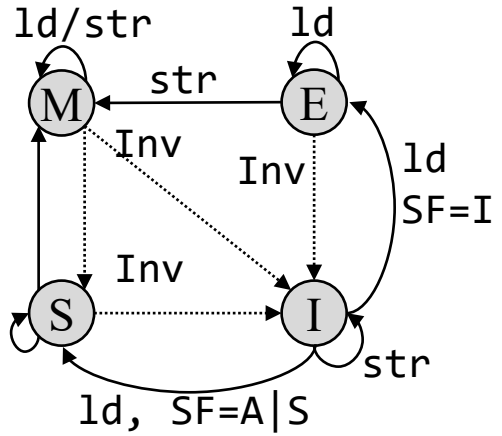
HPE Labs

SDC 25

Transaction Layer (TxN)

- We implement **Directory-based** protocols:
  - ASI at memory (similar to CXL)
  - MESI at hosts
- We also implemented logic to:
  - Packetize TxN messages to LL packets
  - Re-assemble LL packets to a TxN message
- Each packet has a sequence number





HPE Labs

SDC 25

Bandwidth-based Components

- Links, Memory, etc.
- We implement **Token Bucket** algorithm for these components

Simulator APIs

Network and Topology APIs

```
    topo: FatTreeTopo = FatTreeTopo(1, 2, layers_count=layers_count)
    placer: Placer = AlternatingAcrossRacks(topo.racks_cnt,
                                            topo.nodes_per_rack,
                                            )
    builder: TopoBuilder = TopoBuilder(topo, placer)
    builder.build()
```

```
add_host(...)
add_device(...)
add_switch(...)
add_link(...)
build_from_topo(...)
```

Application APIs

```
# Frontend: Bring compose_post arguments from client
app.MemStoreInstr(instr_id=i, obj=frontend_obj),
# Logic: Process compose_post
app.MemLoadInstr(instr_id=qps+i, obj=frontend_obj),
app.MemStoreInstr(instr_id=(2*qps)+i, obj=logic_obj),
# Cache/Storage: Store compose_post results
app.MemLoadInstr(instr_id=(3*qps)+i, obj=logic_obj),
```

```
for i in range(qps):
    frontend_obj = app.Object(object_id=i+1, size=frontend_obj_size)
    logic_obj = app.Object(object_id=qps+i+1, size=logic_obj_size)
    if not use_cxl_sharing:
        cache_storage_obj = app.Object(object_id=(2*qps)+i+1,
                                       size=cache_storage_obj_size)
```

```
compose_post_app = app.Application(env,
                                   app_id=i,
                                   instrs=compose_post_instrs)

core_idx = (i % config.NUM_MAX_CXL_HOST) % h1_core  # Round-robin
h1.submit_application(compose_post_app, core_idx)
```


HPE Labs

SDC 25

Simulator API – Example

A fully simulated CXL system using less than **20 lines of code**

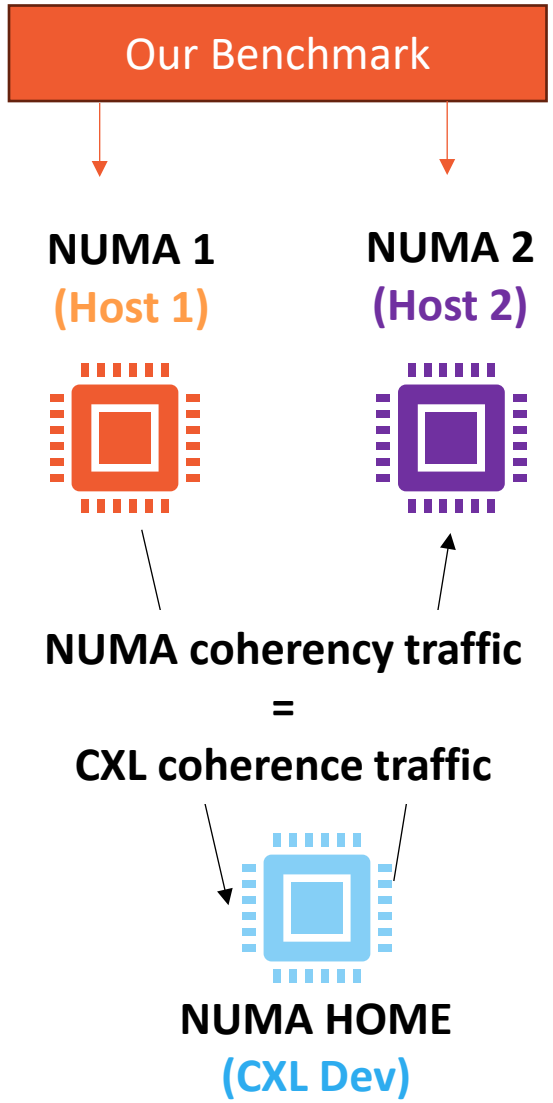
```
env = simpy.Environment

topo: FatTreeTopo = FatTreeTopo(…)
placer: Placer = MixedIntraRack(…)

builder: TopoBuilder = TopoBuilder(topo, placer)
builder.build

net = CXLNetwork(env)
net.build_from_topo(topo, **cfg)

app = YourWorkload(…)
middleware = YourMiddleware(env, net)

middleware.submit(app)
```


HPE Labs

SDC 25

Validating Our Simulator

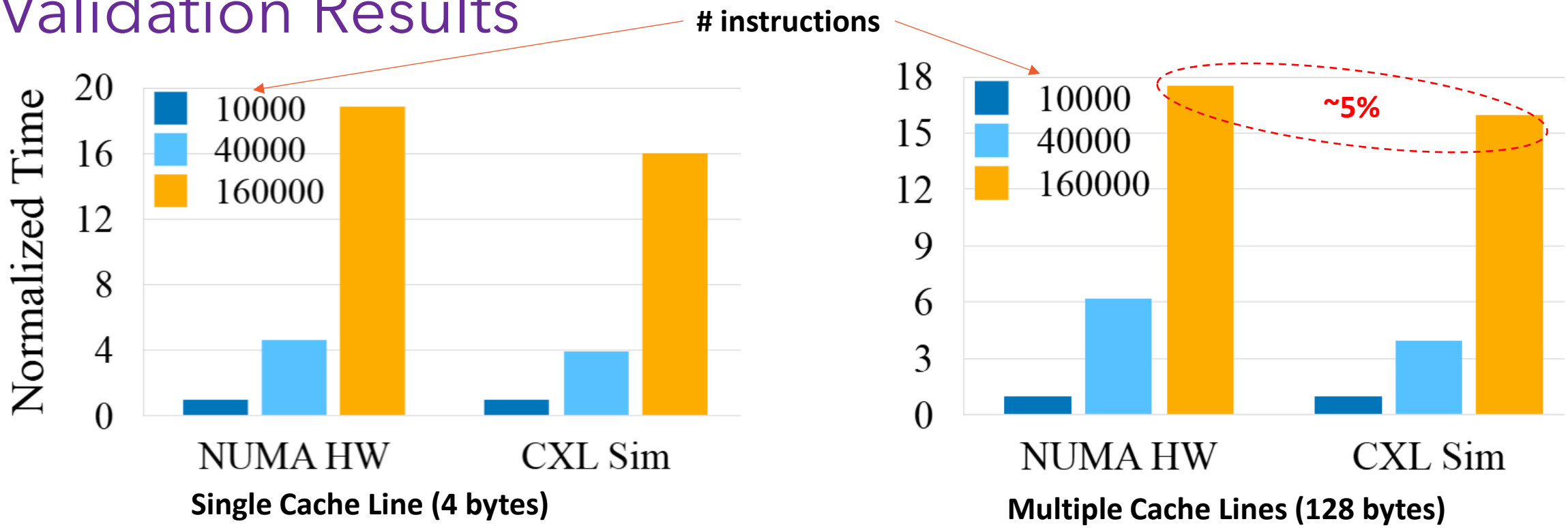
- NUMA and CXL implement similar coherency protocols
- We implement a benchmark that triggers coherence traffic
- We run this benchmark in a **real** NUMA system
- We compare the results from our simulator with the NUMA results
  - We configure the components of our simulator to match NUMA



HPE Labs

SDC 25

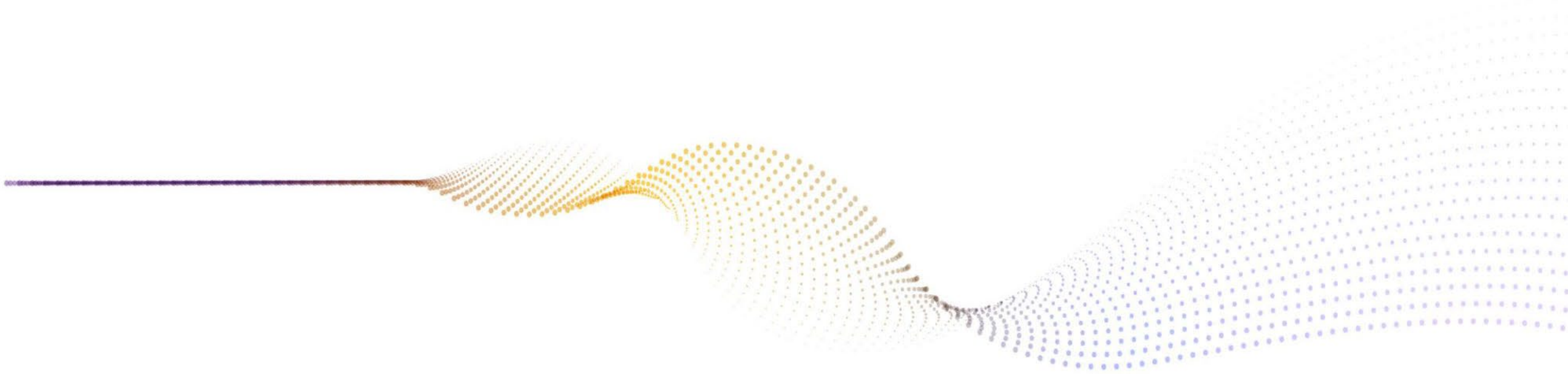
Validation Results



- **Similar trend** of coherency overhead compared to that of real NUMA hardware
- Slightly underestimate the running time
  - We do not simulate congestion at the CXL link layer and memory controller or the impacts of CPU micro-architecture

HPE Labs

SDC 25



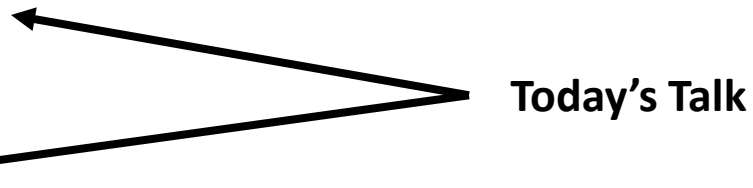
Use Cases of the Simulator

HPE Labs

SDC 25

Examples of Our Simulator Use Cases

- Analysis of Cache Coherency overheads
- Analysis of existing Task Schedulers



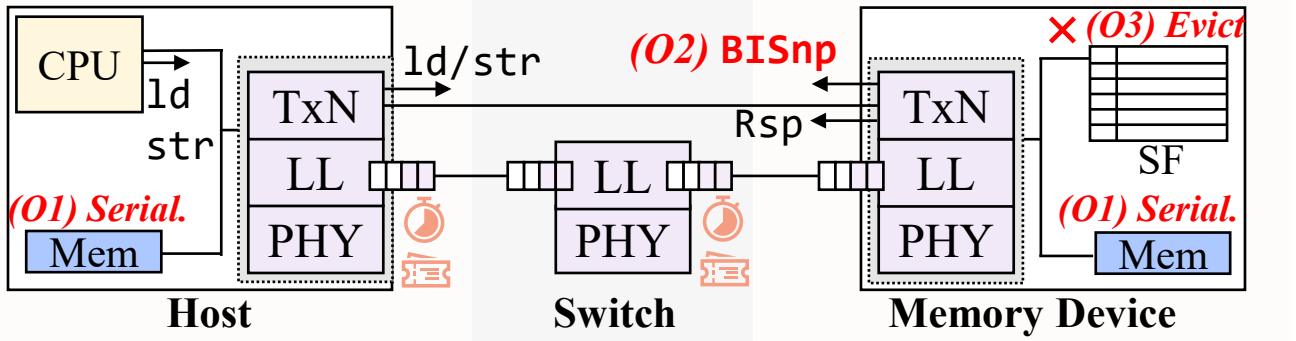
- Explore the configuration space of applications
- Validate new CXL ideas and algorithms
- Guide the deployment of CXL interconnects
- …

HPE Labs

SDC 25

#1: Analysis of Cache Coherency overheads

- Accesses to shared CXL memory are subject to several overheads
- We analyze these overheads using our simulator. We:
  - Categorize the CXL overheads into **(O1) Serialization**, **(O2) BISnp**, and **(O3) SF Eviction**
  - Use a representative application (called `compose_posts`) as a workload
  - Control the workload conditions to trigger different cache coherency events

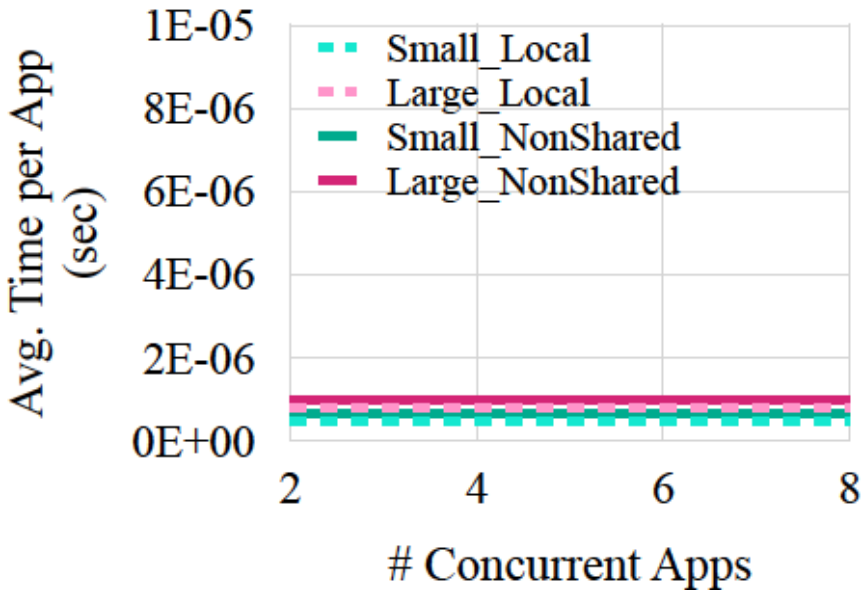


HPE Labs

SDC 25

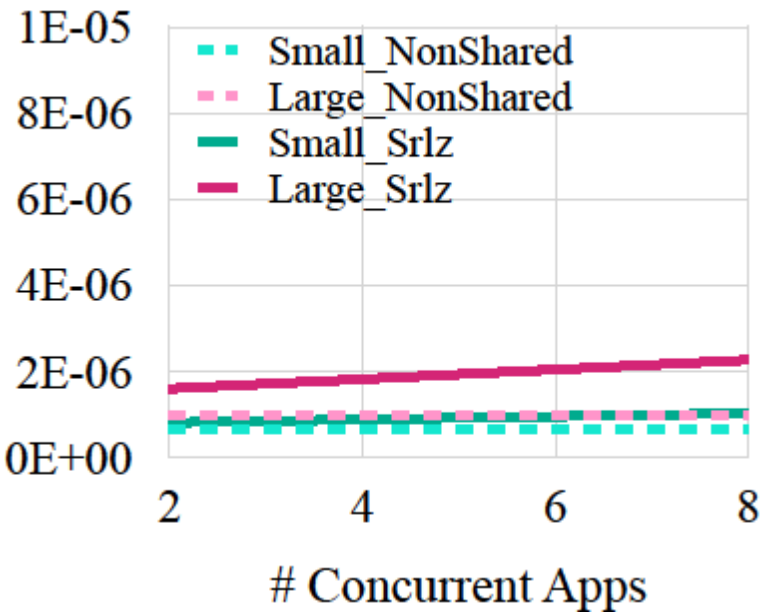
#1: Analysis of Cache Coherency overheads

Impact of CXL Access



CXL access delays impose non-negligible overheads on applications **even without memory sharing**

Impact of Serialization



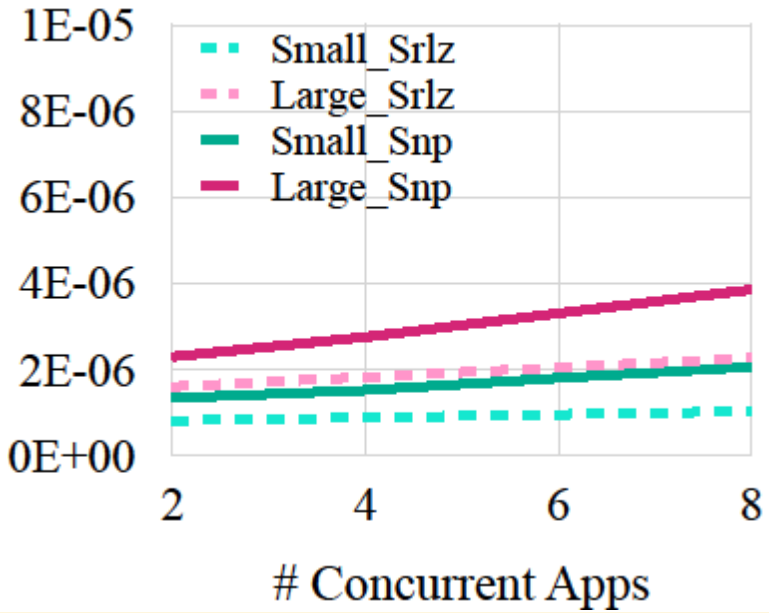
CXL serialization can significantly **increase the request time** as it increases the backlog of pending requests.

HPE Labs

SDC 25

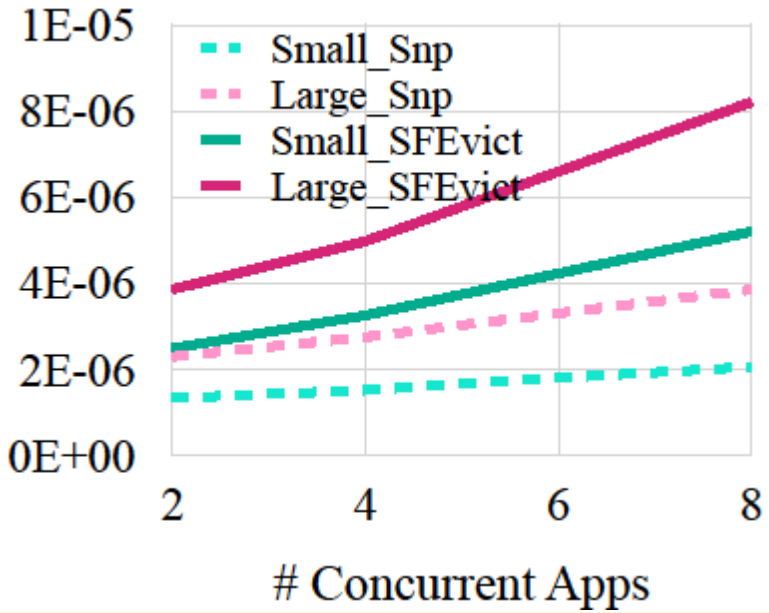
#1: Analysis of Cache Coherency overheads

Impact of BISnp



The cache coherency protocol requires a device to **wait until all** involved cores invalidate the cache line.

Impact of SF Eviction



SF eviction has a **cascading effect** as it substantially increases the **rate of snooping** messages.

HPE Labs

SDC 25

#2: Analysis of Existing Task Schedulers

- Middleware systems will need to evolve to handle CXL
- We explore how **current task schedulers** perform using our Simulator

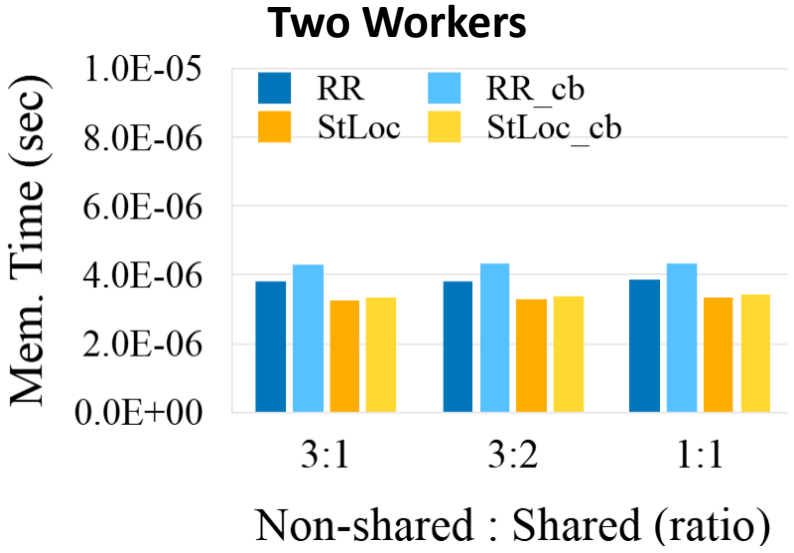
- Two schedulers
  - Round Robin (RR), e.g., Linux
  - Strictly-Local (StLoc), e.g., Spark
- A cache-bypass variation (RR_cb and StLoc_cb)
  - Similar to message passing behaviour

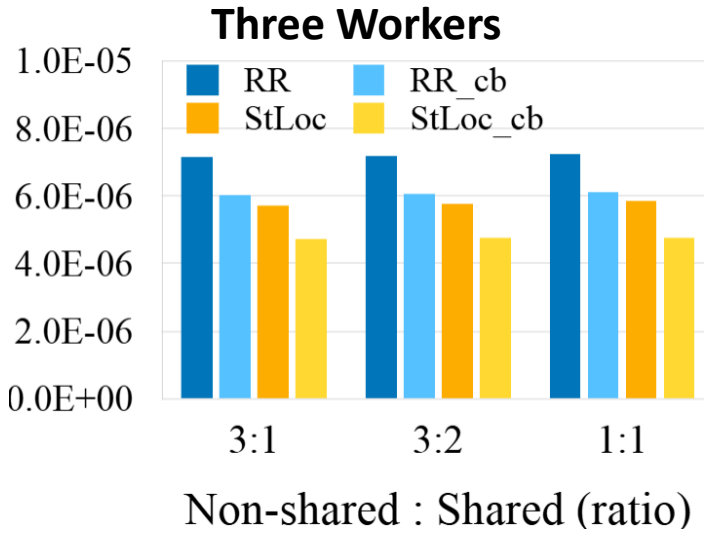
#2: Analysis of Existing Task Schedulers

- We use the PiEstimation workload
- We control two parameters:
  - Number of workers
  - Ratio of accesses of non-shared to shared state

#2: Analysis of Existing Task Schedulers

- RR cannot localize memory regions compared to StLoc
- For three workers, the overheads of coherency increases, and thus the memory time
- Cache-bypass shows that coherence traffic is the main cause of that time increase.





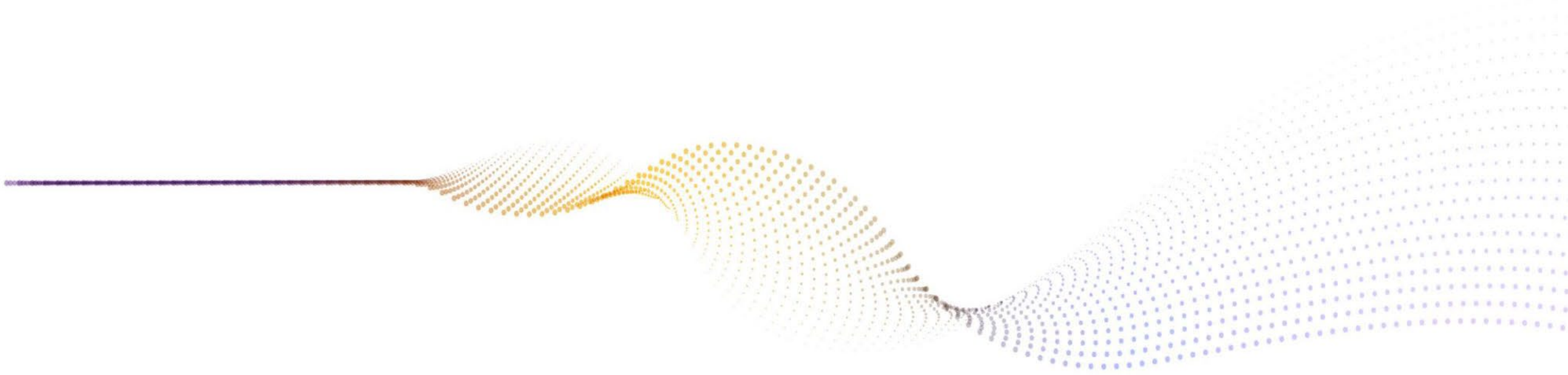
HPE Labs

SDC 25

Conclusions

- The limited availability of CXL 3.0 components has been a major roadblock in the community

- Our simulator
  - provides simple APIs to build CXL setups quickly and accurately
  - focuses on the impact of cache coherency on application performance
  - supports various use cases and scenarios



Thank you for attending!

Please remember to rate this session. You get access the presentations at
http://sniadeveloper.org/conference

khaled.diab@hpe.com

HPE Labs

SDC 25