arm

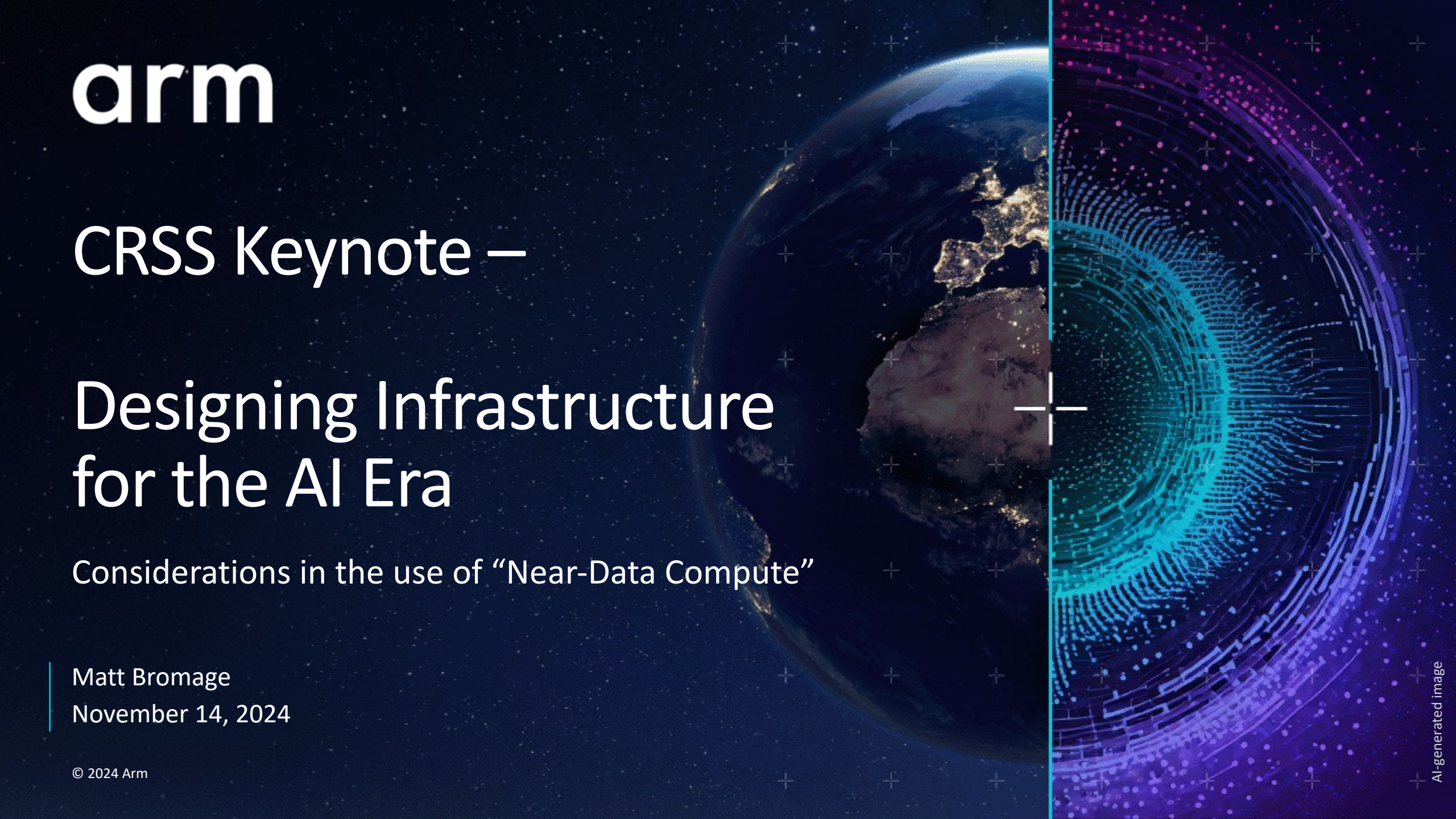
CRSS Keynote –

Designing Infrastructure for the AI Era

Considerations in the use of “Near-Data Compute”

Matt Bromage

November 14, 2024

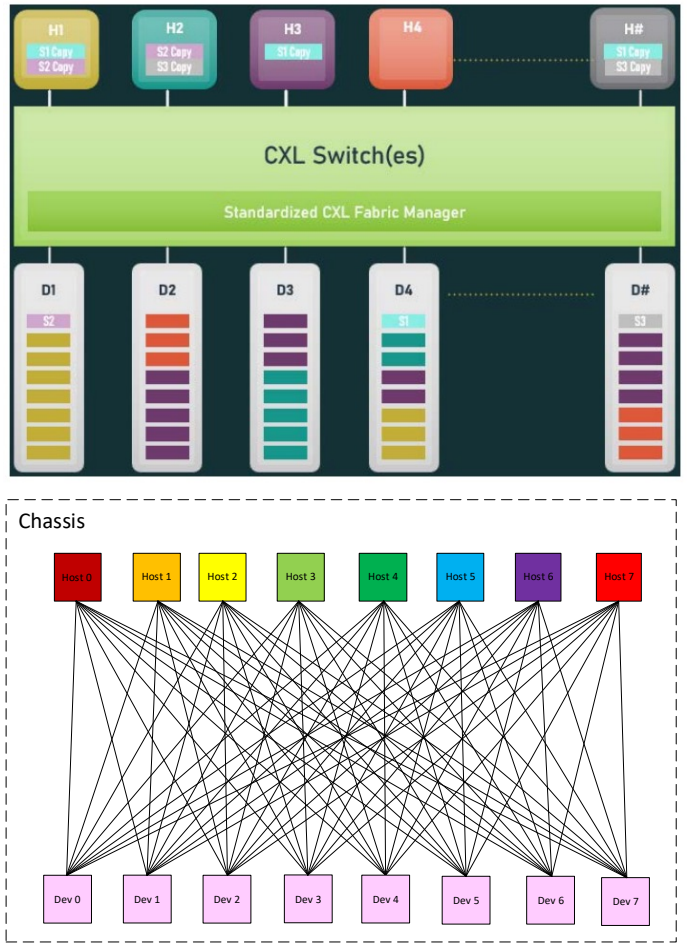


AI-generated image

The Cost of AI Inferencing

“Far” Memory Can Reduce Infrastructure Costs

- How?
  - DRAM reuse
  - Smaller DIMM capacities
  - Reduction of “stranded memory”
  - Server disaggregation
- But at what cost?
  - Higher latencies
  - Additional hardware (CXL controllers, CXL switches, chassis)
  - Additional complexity (tiering, mgmt., RAS considerations, security)



arm

Not all Inferencing is Equal…

- **Key Insight** - not all inferencing has the same QoS requirements (ex: paid vs free, batch vs live)
  - By relaxing QoS requirements you can extend the allowable computational time for inferencing tasks, opening the door to using CXL-attached memory to reduce infra costs

- **Thesis:** by using CXL-attached memory paired with computational cores (Near-Memory Compute) and offloading AI inferencing tasks according to memory sensitivity, we can reduce both the datacenter infrastructure costs as well as the inferencing latencies inflated by using CXL memory

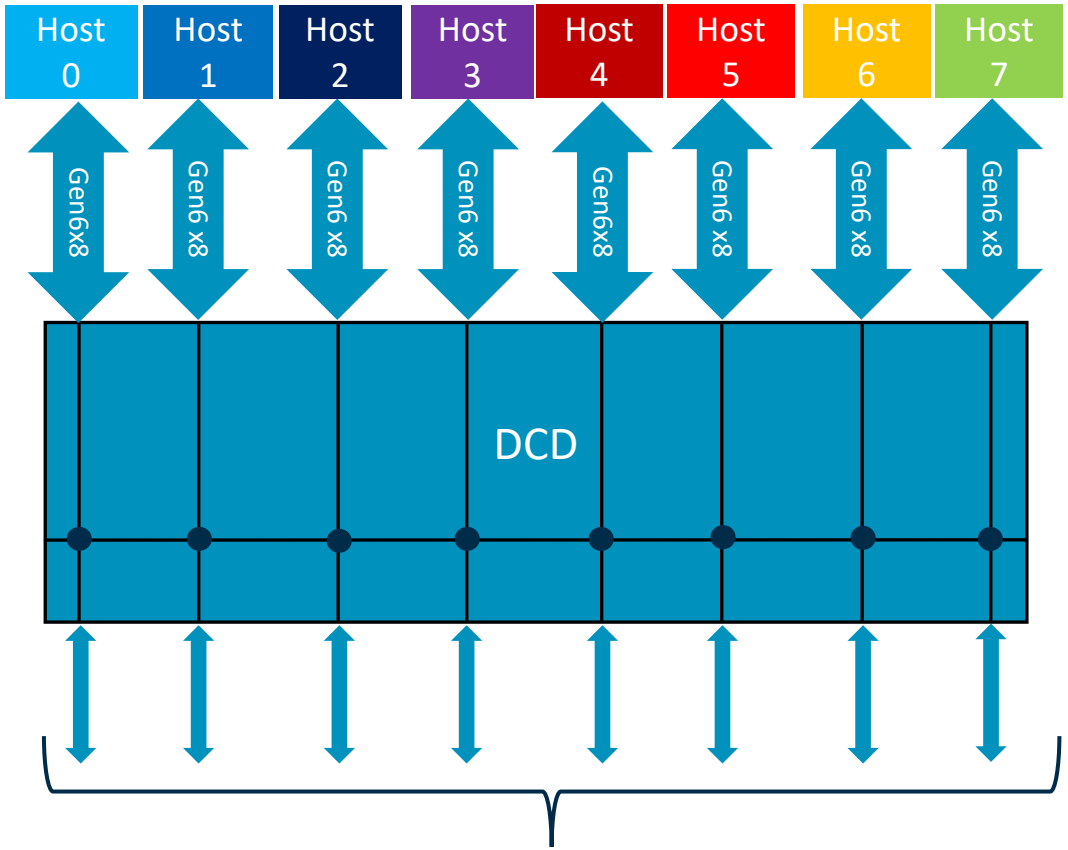
The Case for Near-Memory Compute

- OCP Composable Memory Systems (CMS) workgroup

UDON: A case for offloading to general purpose compute on CXL memory

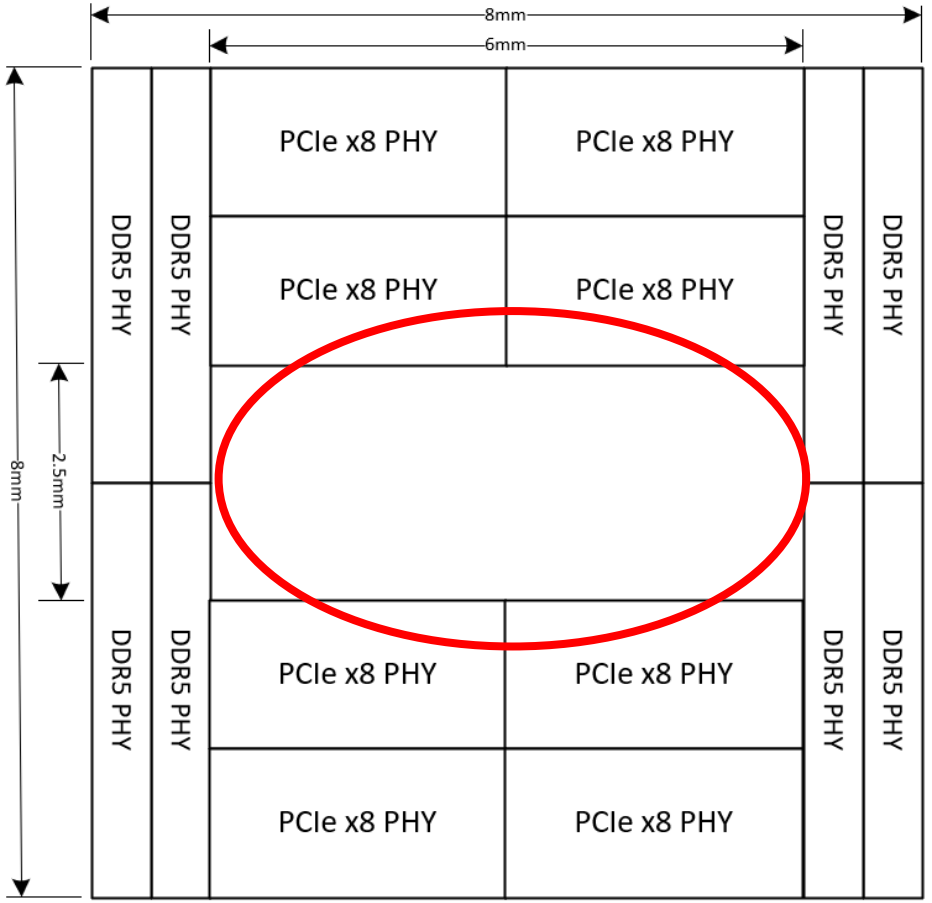
Adding HW Coherency + Compute to CXL Controllers

- Chip size is dominated by IO blocks



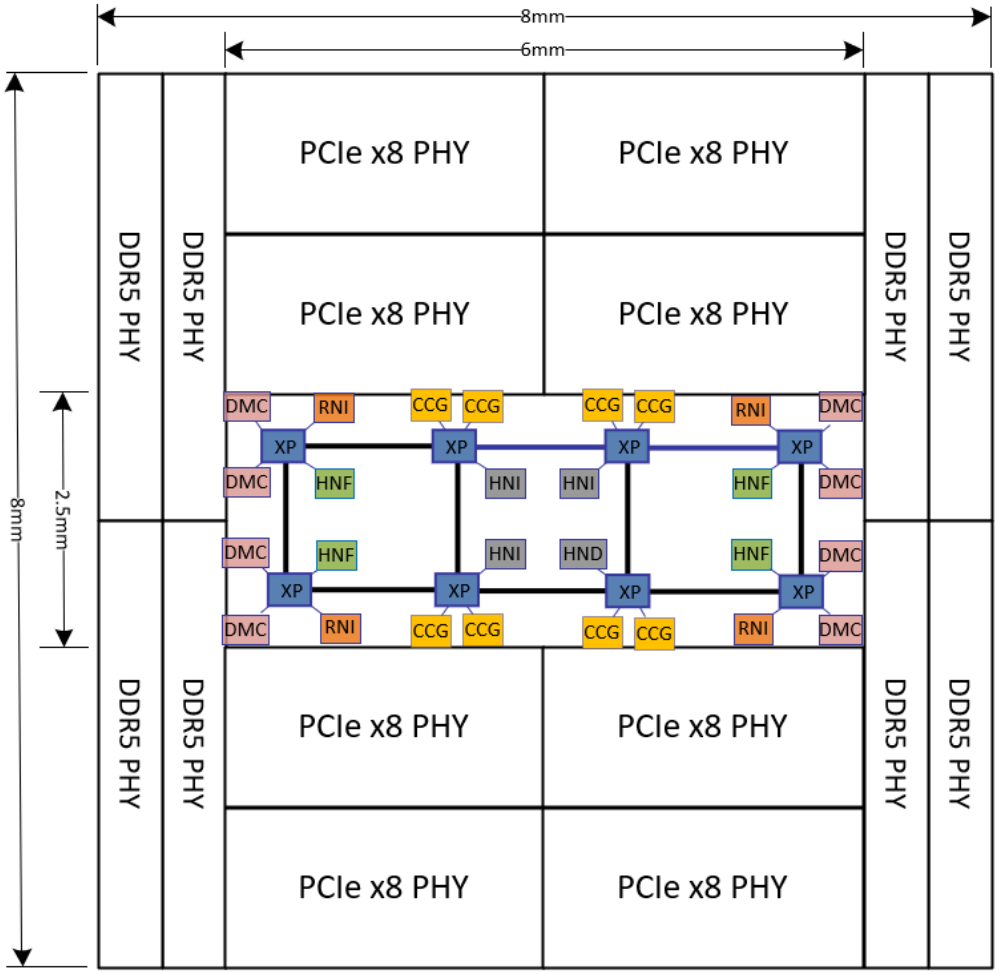
8 DDR5 channels @6400:

- **2 TB Capacity**
- **200 GB/s BW.**



Measurements assuming 5nm design

Silicon Area Estimates



Coherency logic and compute cores can fit inside the 2.5 x 6 (15 sqmm) box:

Pooled FAM Logic: ~ 7 sq mm

- Memory/CXL Controllers
- 8x8 Switching logic

+

Shared FAM Logic: ~ 6 sq mm

- Home Nodes for host/device coherency
- Snoop Filter

+

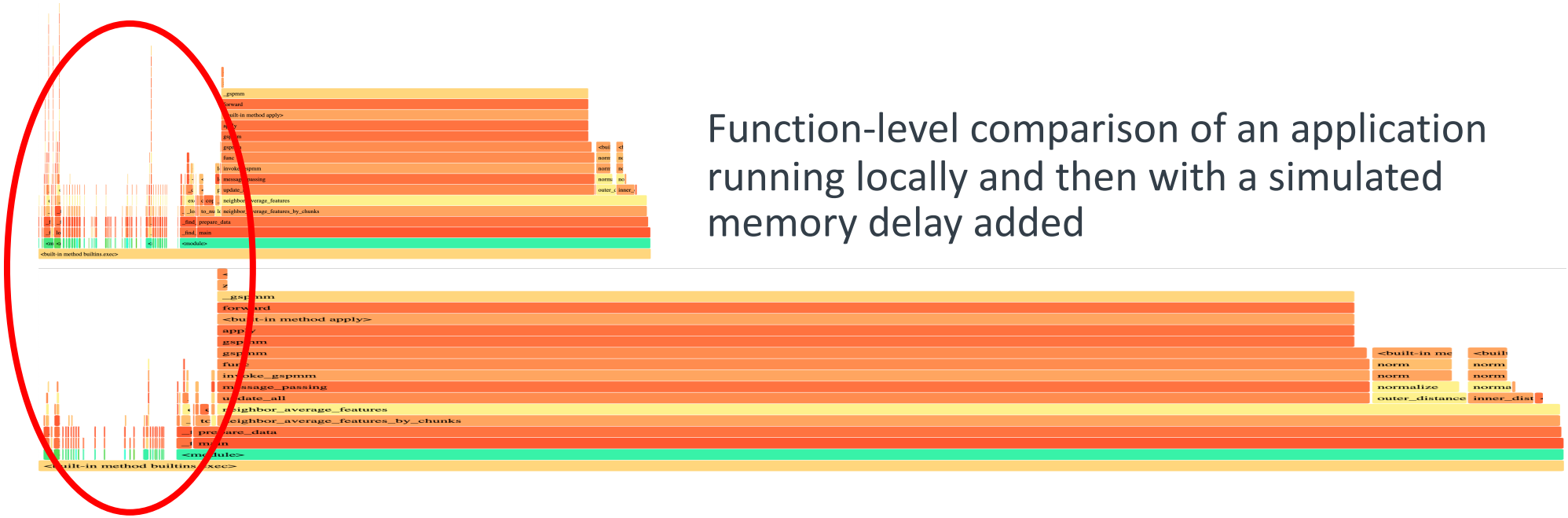
Optional Capabilities ~ 2 sq mm

- System Level Cache
- Near-Memory Compute (NMC) cores

arm

Application Profiling for Targeted Offload

Start of application not as affected by memory latency increases



Function-level comparison of an application running locally and then with a simulated memory delay added

Applications have various levels of sensitivity to memory latency increases and are not harmed uniformly. **We should therefore investigate offloading at the function level of granularity**

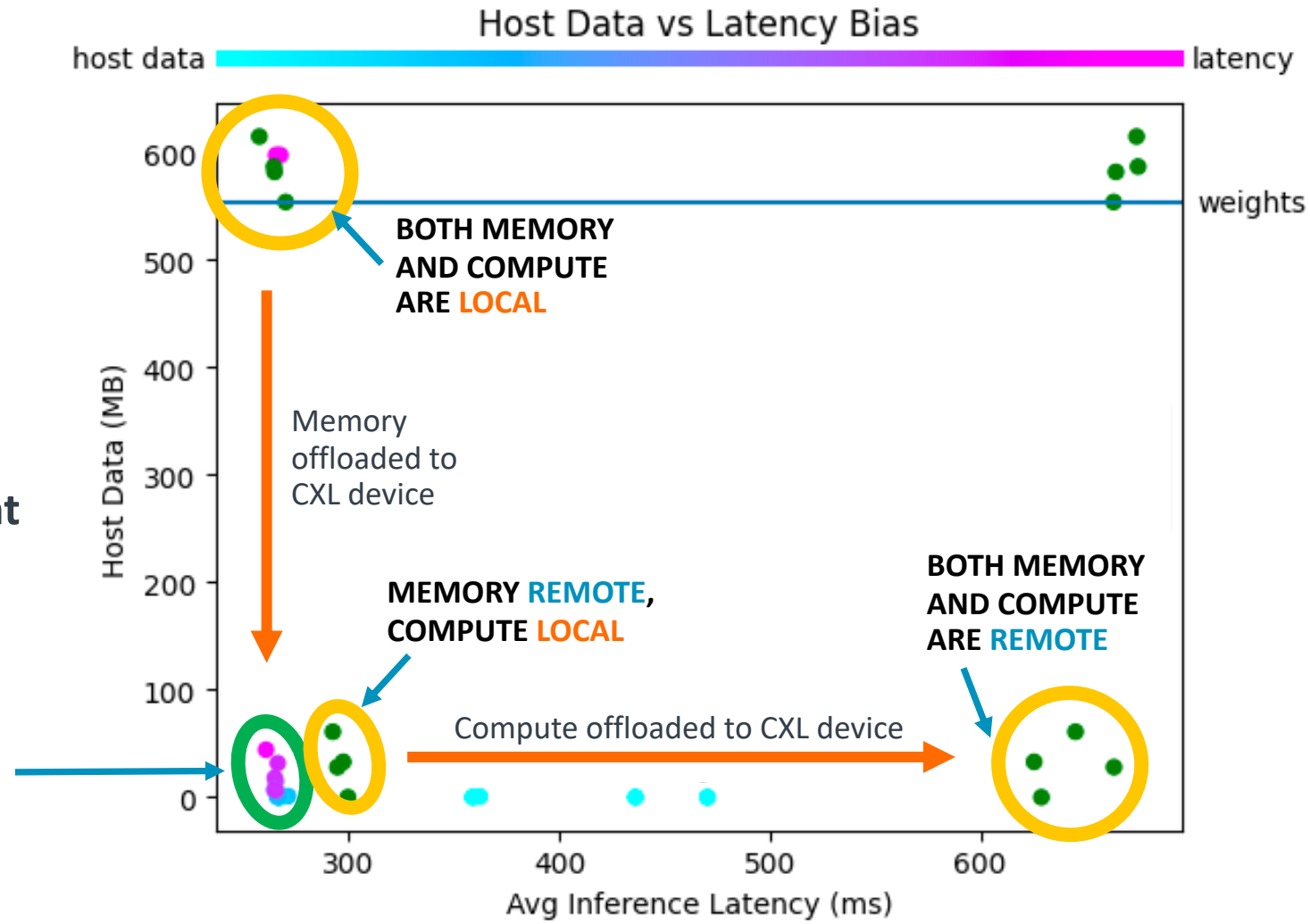
Designing a Compute Placement Algorithm

- We need to define a multi-objective function:
  - **Maximize** the “far” memory allocation
  - **Minimize** the total run-time of the inferencing model (referred to as “latency”)
  - **Weighted sum cost function** w/ weights [0-1] selected to prioritize host data placement or latency
- We wrote a combined **Memory-Compute Placement Algorithm** which takes the cost function bias as input, as well as **profiled** runtime costs for running each layer in every combination of local and remote
- This algo outputs annotations for each layer of the ML model, designating various memory and compute location as either local or remote
- We used a modified **Tensorflow** runtime to accomplish both control levers for memory and compute placement, both during profiling and executing with these annotations
  - **Memory** is placed either local or remote via libnuma
  - **Compute** is placed local or remote via thread affinity

Results

For optimal placement of both compute and memory, we want to hit the bottom-left corner: **use as much far memory as possible *and* take as little time as possible**

By utilizing the Memory-Compute Placement algorithm, average latency is reduced by ~15% while also making use of far CXL memory



Key Takeaways

- **AI infrastructure** is expensive! So too will be dedicated inferencing data centers…
- **“Far” memory** can be part of the solution to reduce costs, with some tradeoffs
  - Flexibility on QoS requirements is the key
- The addition of **Near-Memory Compute** can help mitigate the latency and bandwidth limitations inherent in using “far” memory
  - In some cases, NMC can completely recoup all lost performance
- **Data Placement** and **Compute Placement** are both important
  - In fact, the most efficient use of far memory *requires* compute offload; not something data placement can solve alone
- **Challenges** of course exist:
  - CXL controllers need to include dedicated Near Memory Compute resources
  - Shared addressing or full coherence between host and device is needed for efficient offload
  - Standardization of offload instructions - kernel sys calls need to be handled correctly
  - Cross-ISA offloading requires additional consideration
  - Software profiling for memory sensitivity is required for this approach

arm

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు