Composable Systems
Are we there yet?

Christian Pinto
Staff Research Scientist
IBM Research Europe
christian.pinto@ibm.com

First things first

Call things by their proper name

Composable **Disaggregated** Infrastructure (CDI)

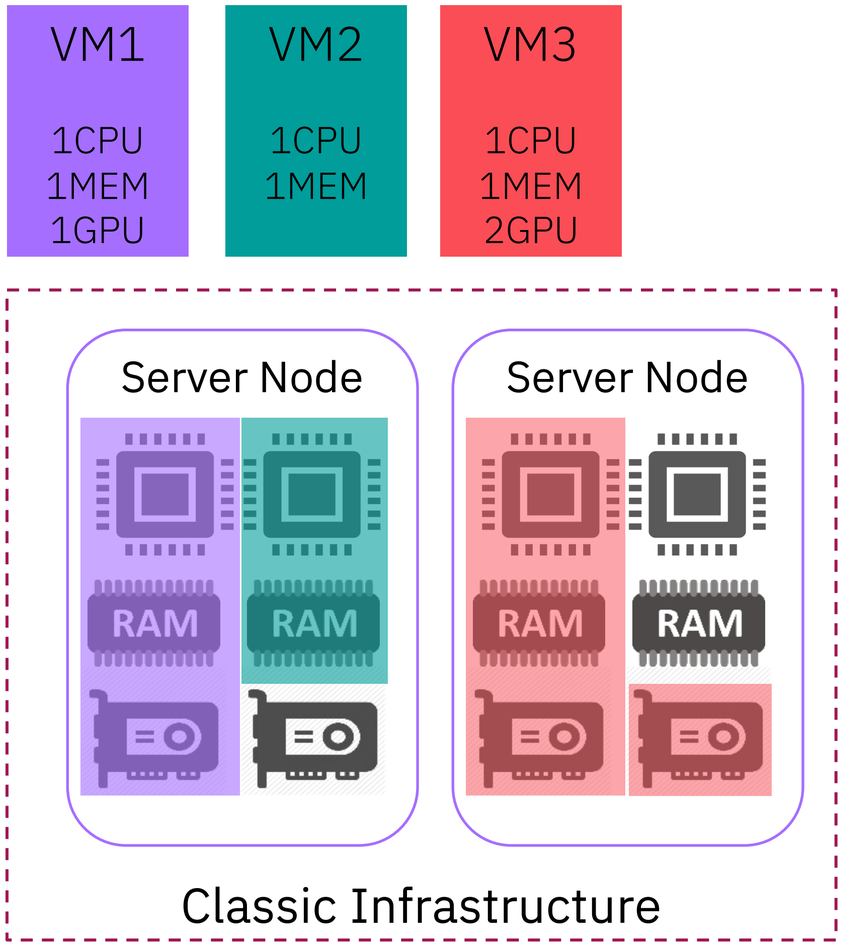
Background

CDI constituents

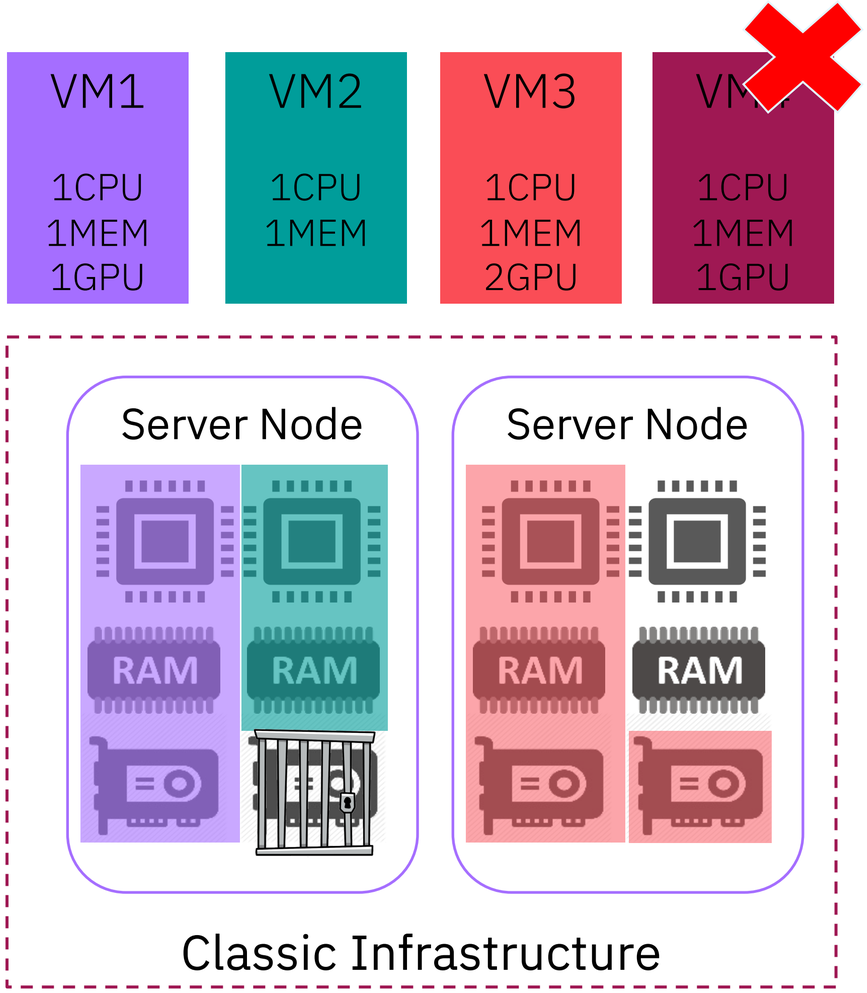
A set of disaggregated compute, memory, storage and network elements that can be assembled to produce virtual compute servers and clusters on-demand.

Software to provision and manage the underlying physical resources, to assemble servers, and to support and optimize workflow deployment across those physical resources.

CDI in action



CDI in action



Possible solutions

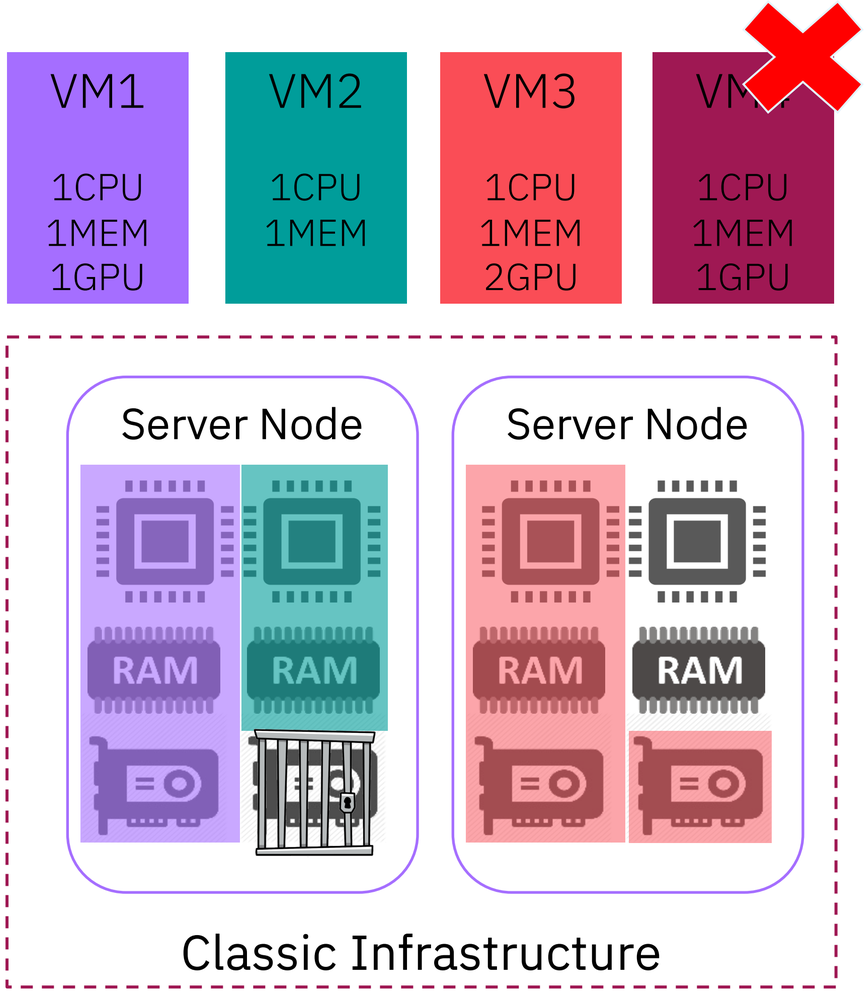
Overprovisioning

More resources on each node to make sure most jobs can be hosted

Specialised nodes

Large memory nodes, large gpu node or nodes with specific accelerators (GPUs, FPGAs)

CDI in action



Possible solutions

Overprovisioning

More resources on each node to make sure most jobs can be hosted

Specialised nodes

Large memory nodes, large gpu node or nodes with specific accelerators (GPUs, FPGAs)

Issues

- Inefficient utilization of hardware
- Over engineered power and cooling
- Complex update cycles

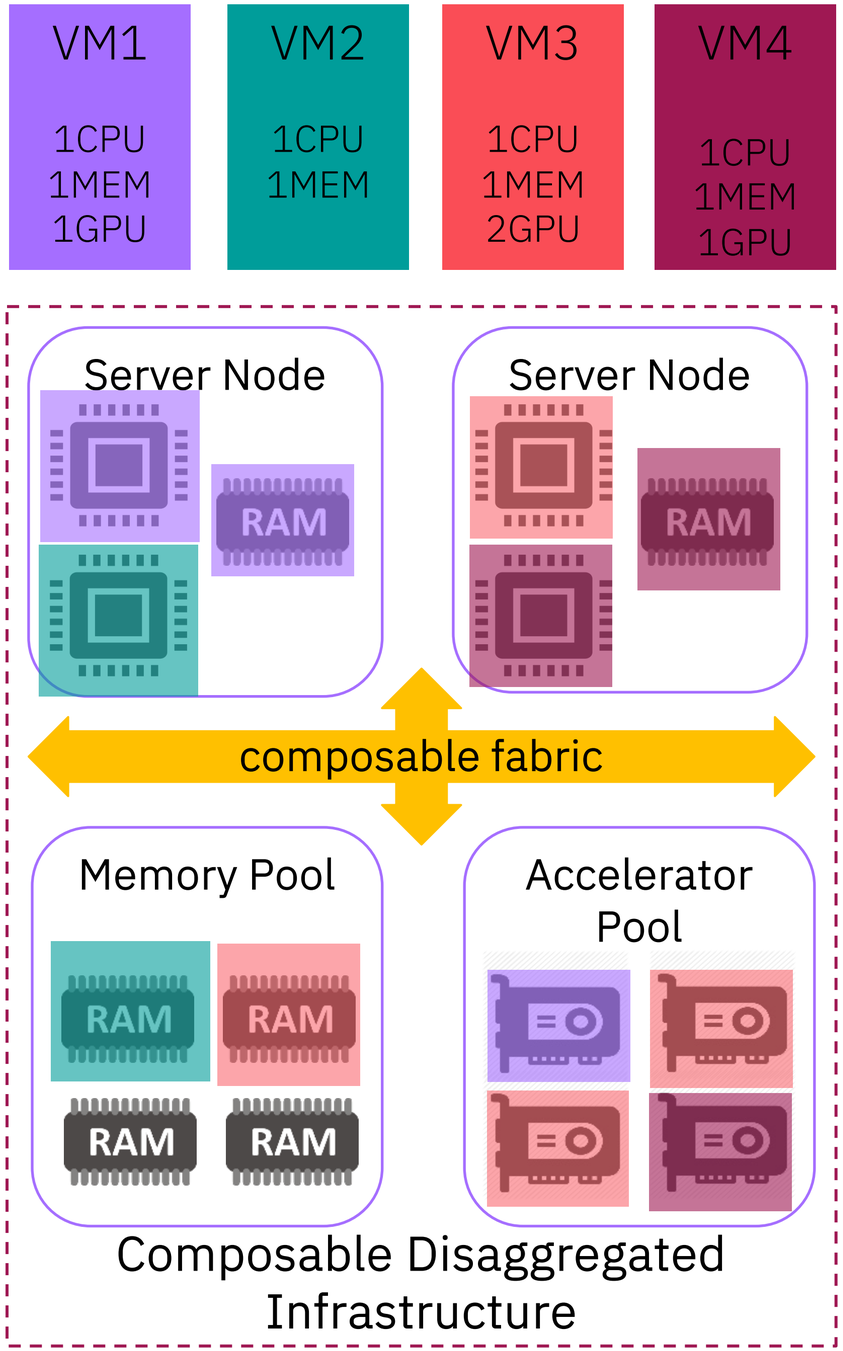
CDI in action

Flexibility

Resources are pooled and connected through a composability fabric

Adaptability

Nodes can be dynamically adapted to incoming workloads



CDI in action

Flexibility

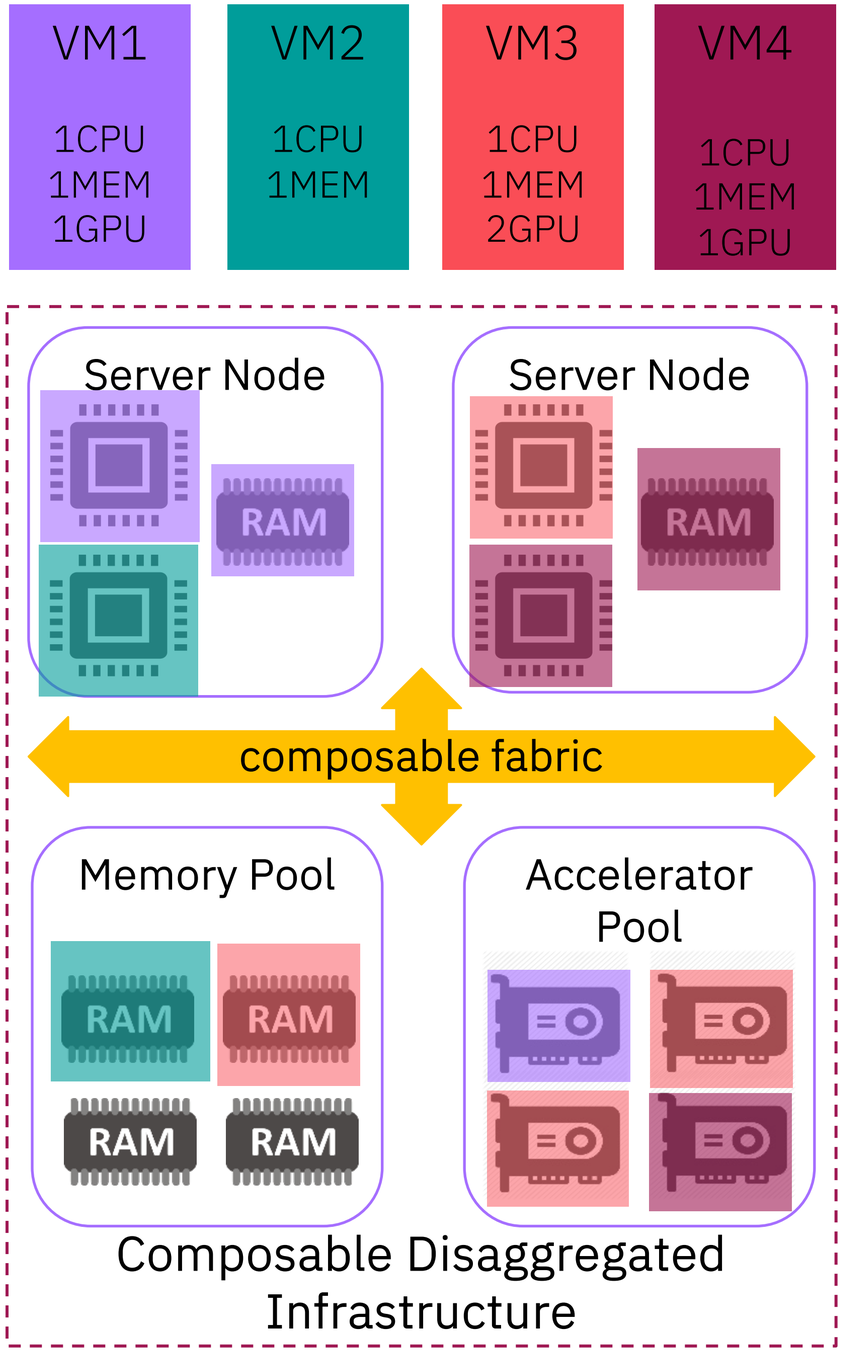
Resources are pooled and connected through a composability fabric

Adaptability

Nodes can be dynamically adapted to incoming workloads

Questions

- Another fabric?
- Latency, latency, latency…
- Who’s managing all this goodness?
- Do I need to change my applications?



Value Matrix

Flexibility and Adaptability

System can be continuously balanced and reconfigured in software to support varying workload requirements.

Responsiveness

Dynamic provisioning can even occur within an executing application – e.g. add more memory or add a GPU to an execution step.

Efficiency

Virtual nodes can be dynamically configured when needed so less stranding of resources, more efficient utilization, and less power for a given throughput .

Upgradability

Individual components can be upgraded without having to replace complete system.
Allows customer to be on leading edge technology.

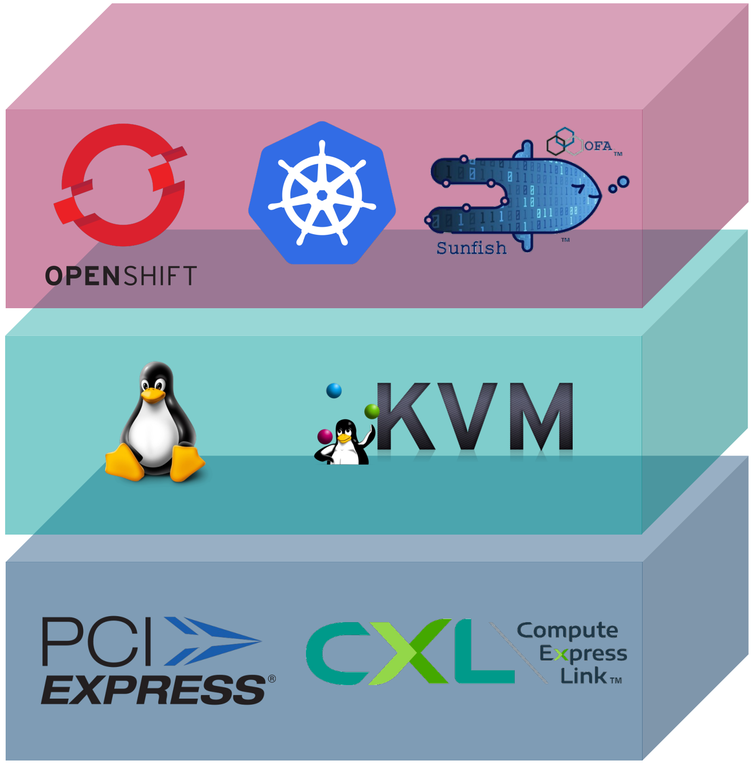
Reduced Power Consumption

Because composed nodes can be correctly assembled for each different application, much less resource stranding with much less wasted power.

Sustainability

New technologies and upgrades can occur at component level, each component group can have different lifespan.
No need to “crush” whole systems every 5 years.

Outline of this talk

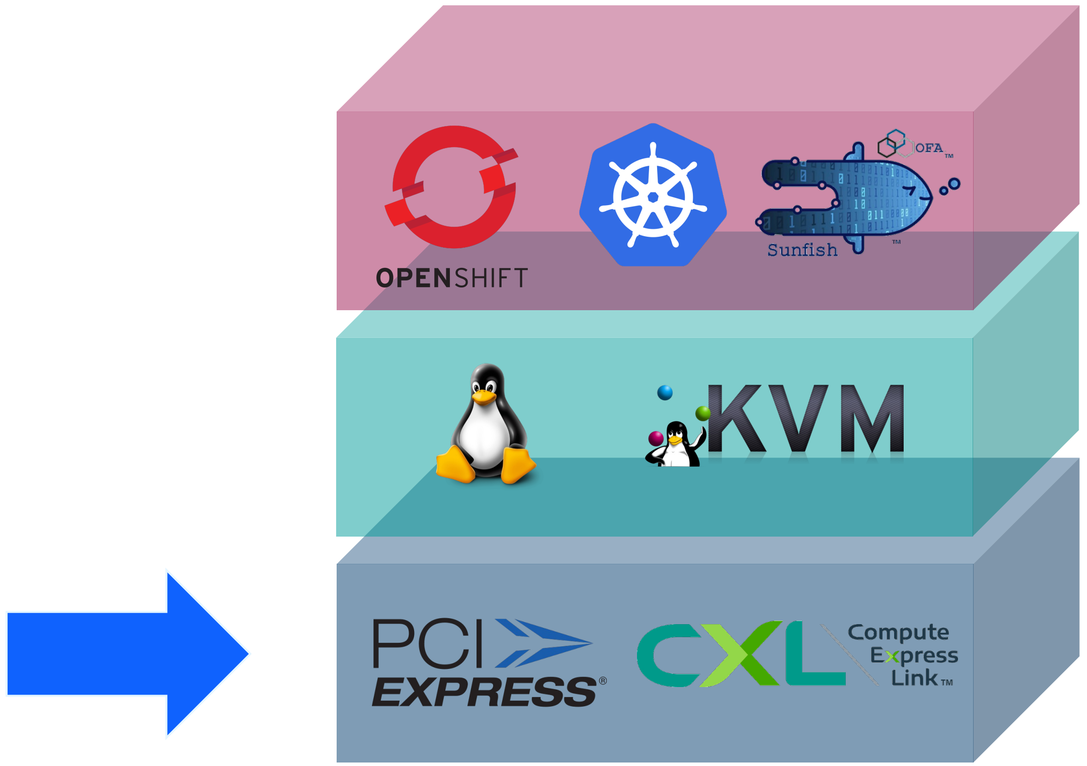


3. Orchestration SW

2. Low-level SW

1. Hardware

Outline of this talk

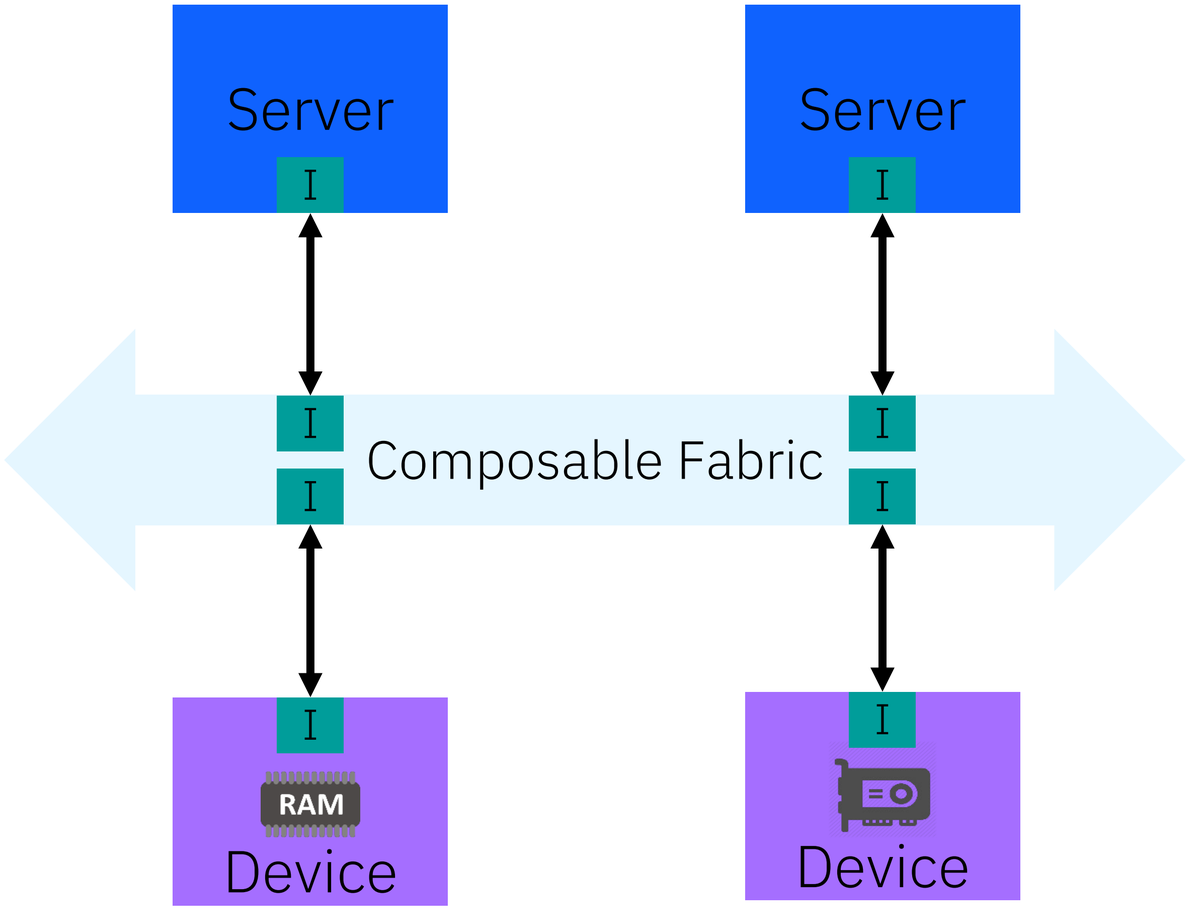


3. Orchestration SW

2. Low-level SW

1. Hardware

Hardware for CDI



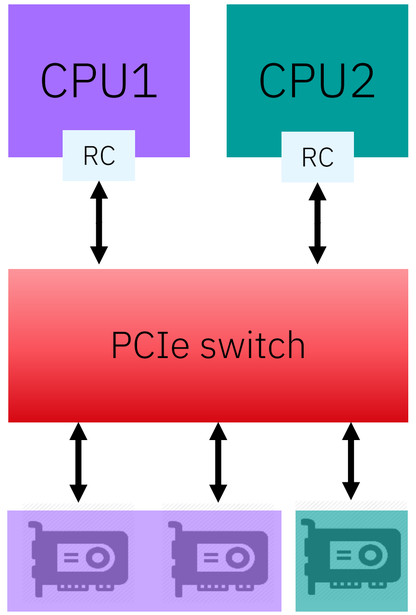
Common interface for interaction over fabric

Existing protocols

- PCIe
- CXL (OpenCAPI, GenZ)

PCIe based fabric

- Targeting composition of I/O devices only
  - GPUs
  - FPGAs
  - Disks
  - Network cards
- Increase utilization of expensive accelerators (e.g., GPUs)



Already available off-the-shelf

- Liqid [1]
- GigaIO [2]
- H3 Platform [3]

[1] https://www.liqid.com/
[2] https://gigaio.com/
[3] https://www.h3platform.com/solution/composable-ai

PCIe based fabric

PCIe based solutions work well

- No changes required to either CPU or devices.
- Well established support in OS.
- Effective in increasing utilization of expensive hardware.

But…

- PCIe does not support byte-addressable access to memory:
  - ATTENTION: PCIe devices BARs are not memory, they are a configuration space for accessing device registers.
  - Do not support cache coherence between CPU and devices.
- Number/type of devices that can be actually composed to a system is limited by BIOS.
- Devices hot-plug/unplug is still not supported on all devices.

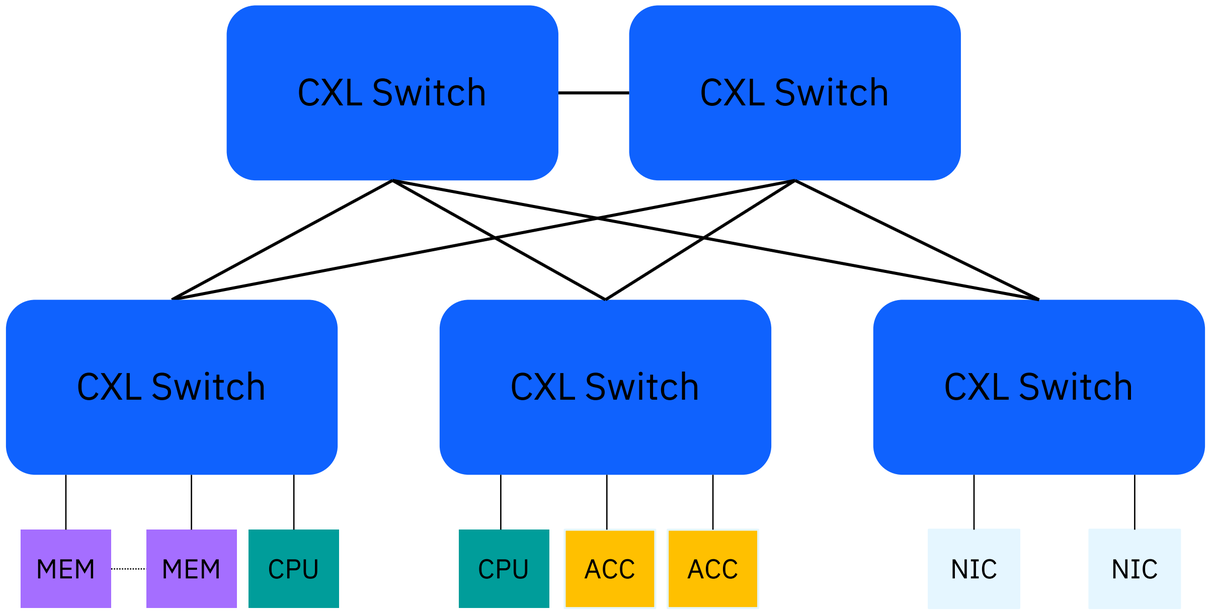
CXL based fabric

Relies on PCIe (5.0+) physical to multiplex 3 protocols

- cxl.io
- cxl.cache
- cxl.mem

Features (CXL 3.1)

- All PCIe feature (cxl.io)
- Byte addressable memory (cxl.memory)
- cpu-device cache coherence (cxl.cache)
- Global Fabric Attached Memory
- Coherent shared memory across hosts
- p2p device communication
- Multi layer fabric



Support for 3 device types

- Type-1: Accelerators (cxl.io, cxl.cache)
- Type-2: Accelerators with memory (cxl.io, cxl.cache, cxl.mem)
- Type-3: Memory expanders (cxl.io, cxl.mem)

CXL based fabric

However….

- Today’s CPUs only support CXL1.1
  - Only direct attached devices, no switches, not fully composable
- Most of the research on Type-3 devices so far
- Accessing memory over CXL incurs in extra latency

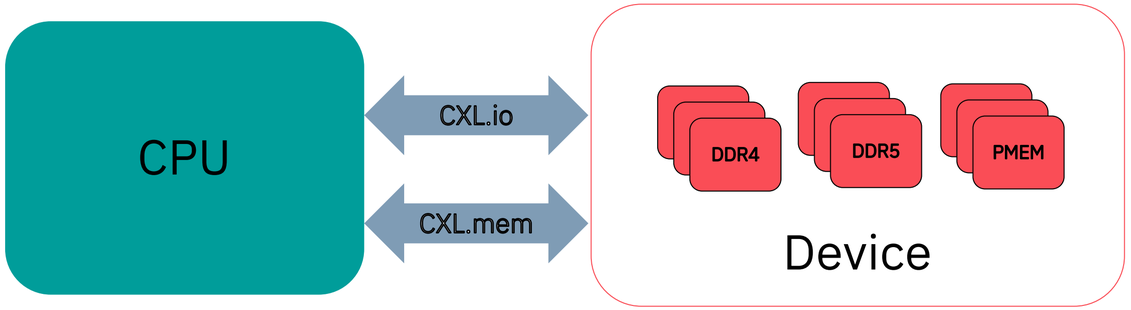
Partially false, CXL switches started appearing that provide some initial “composability” capabilities [1]

[1] https://www.youtube.com/watch?v=fTfHMQxg7t8

CXL based fabric

CXL Type 3 Device

Simply put: a CXL device that expands system memory by abstracting out the actual memory technology.



Volatile memory

Presented to the OS (Linux) as a CPU-less NUMA node.

Memory allocations follow the rules of a normal NUMA system.

and/or

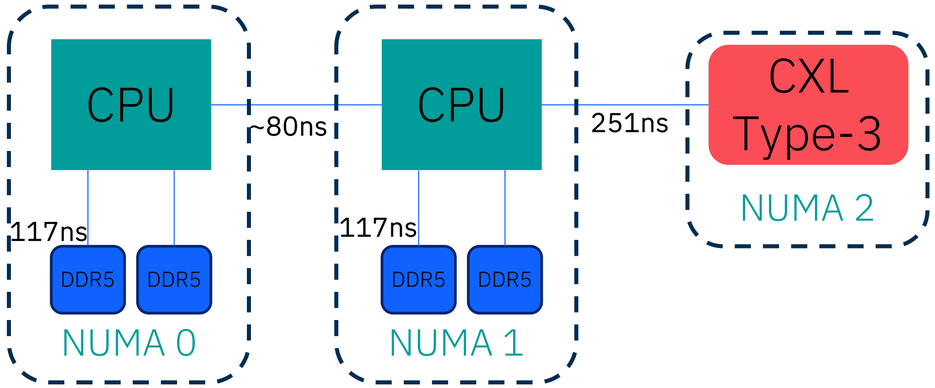
Persistent memory

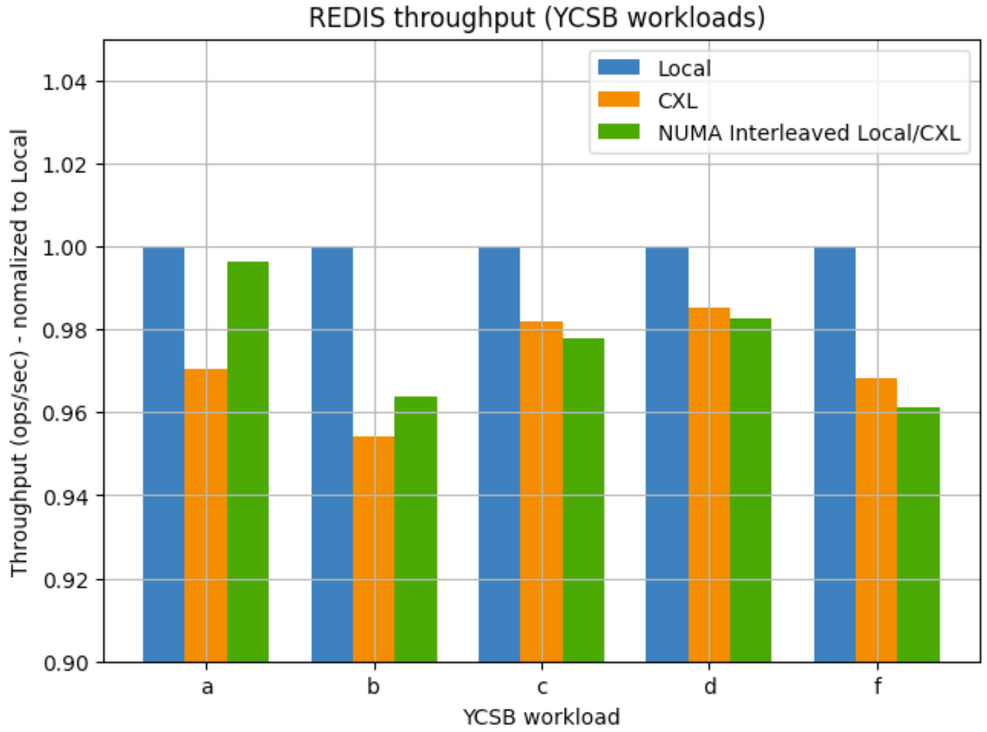
Presented to the OS (Linux) as a pmem device.

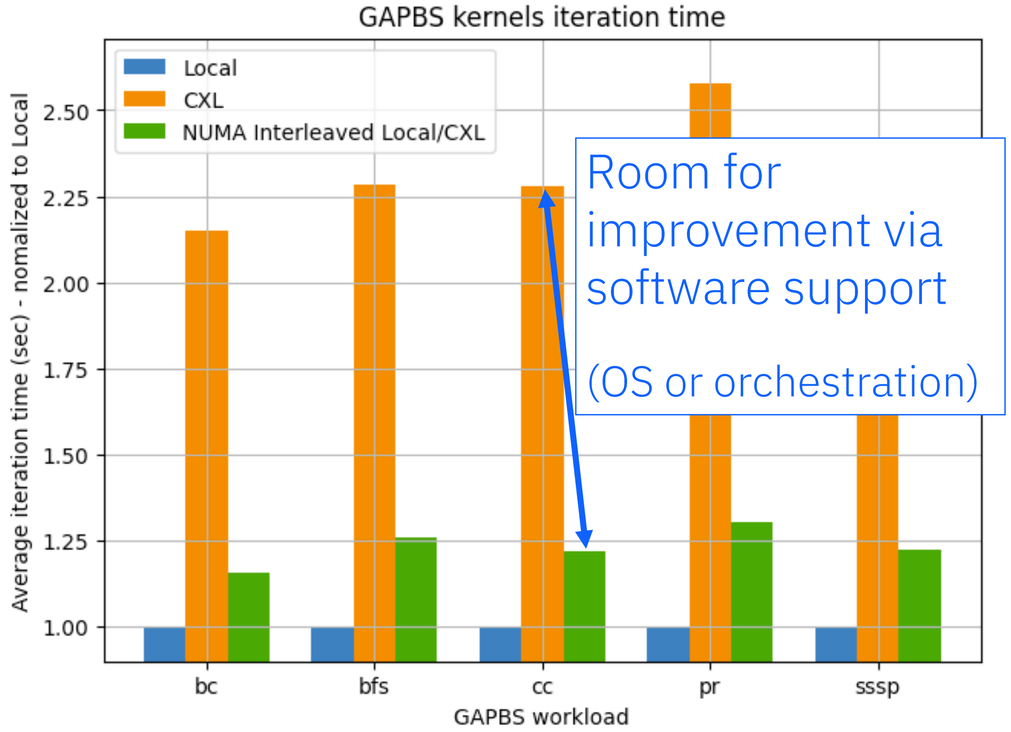
daxctl can be used for controlling the device.

CXL based fabric

Is latency that big of an issue? Not necessarily!

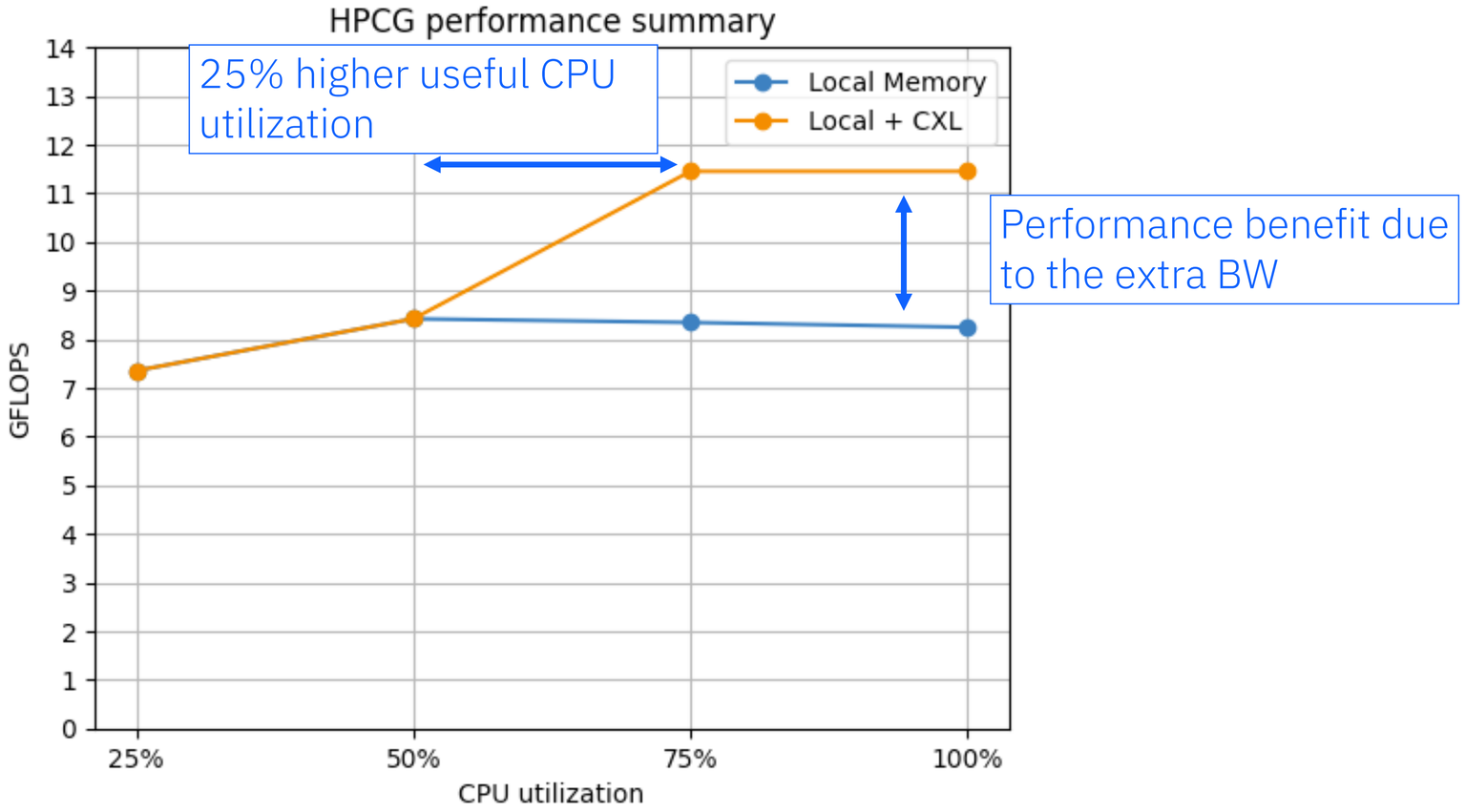






CXL based fabric

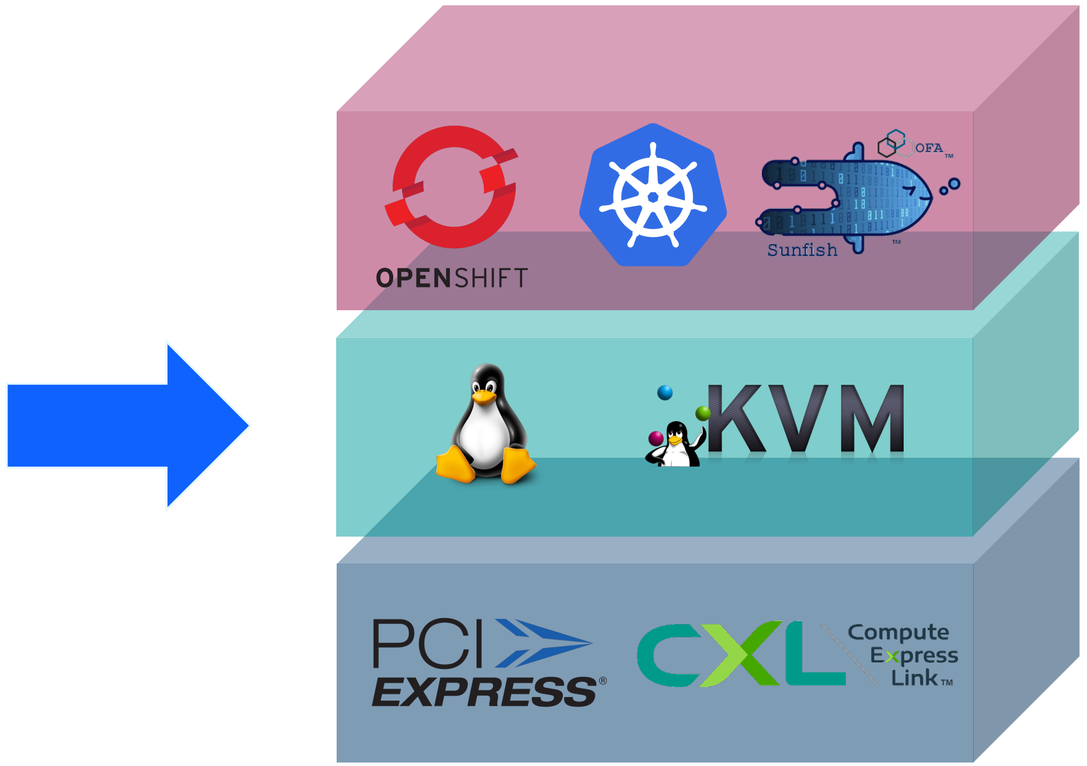
And it is not all about the latency....



What’s next

- Improved BIOS support from server manufacturers to support “dynamic hardware”.
- Improved devices support for hot-plug/unplug.
- Further research on CXL Type-1 and Type-2 devices.
  - Although great on paper, they need to be validated on the field.

Outline of this talk



3. Orchestration SW

2. Low-level SW

1. Hardware

Operating system and low-level software for CDI

Device drivers

CDI devices discovery, hot-plug/unplug, lifecycle management

Memory Management

Integration of CDI based memory with main memory management system.

Performance oriented optimizations.

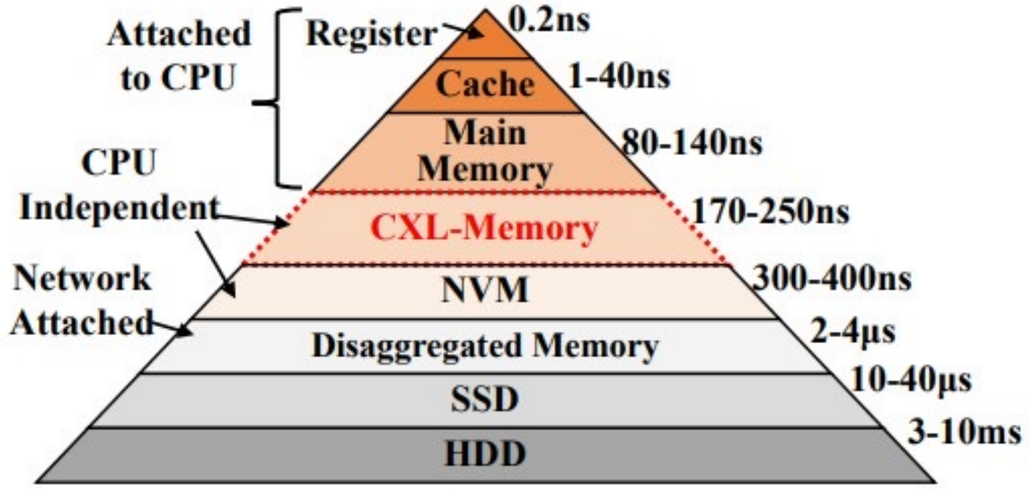
Virtualization

Exploit CDI for improved virtualization services

Memory management

Treat CXL based memory as yet another memory tier

- CXL memory is a NUMA node
- Memory allocations fall-back to the CXL NUMA node when the local node is full
- Leverage AutoNUMA [1] for automated balancing of memory pages across NUMA nodes

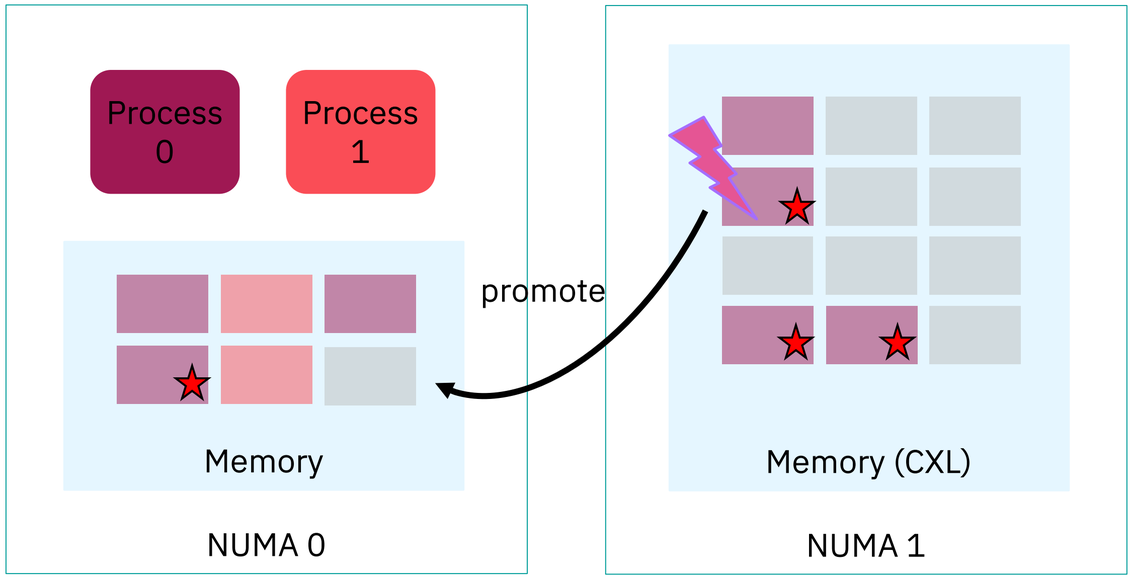


[1] https://mirrors.edge.kernel.org/pub/linux/kernel/people/andrea/autonuma/autonuma_bench-20120530.pdf

Linux AutoNUMA

Assumption: Most frequently accessed (hot) pages should be closer to computation (local memory)

1. Pages are periodically marked by os thread (knuma_scand)
2. Accessing a marked page issues a lightweight page fault (hinting fault)
3. Page are migrated to the local node if misplaced w.r.t. computation



Issues:

- Prone to promotion-demotion loops
- Not good at identifying hot pages (promote at first access)
- Marking too many pages generates too many page faults

Transparent Page Placement (TPP) built on top of AutoNUMA to provide better pages placement and solve these issues [1]

[1] https://dl.acm.org/doi/10.1145/3582016.3582063

Beyond Linux AutoNUMA and TPP

However…

Relying only on NUMA hinting faults does not provide a precise indication of the hotness of a page because of its short time horizon

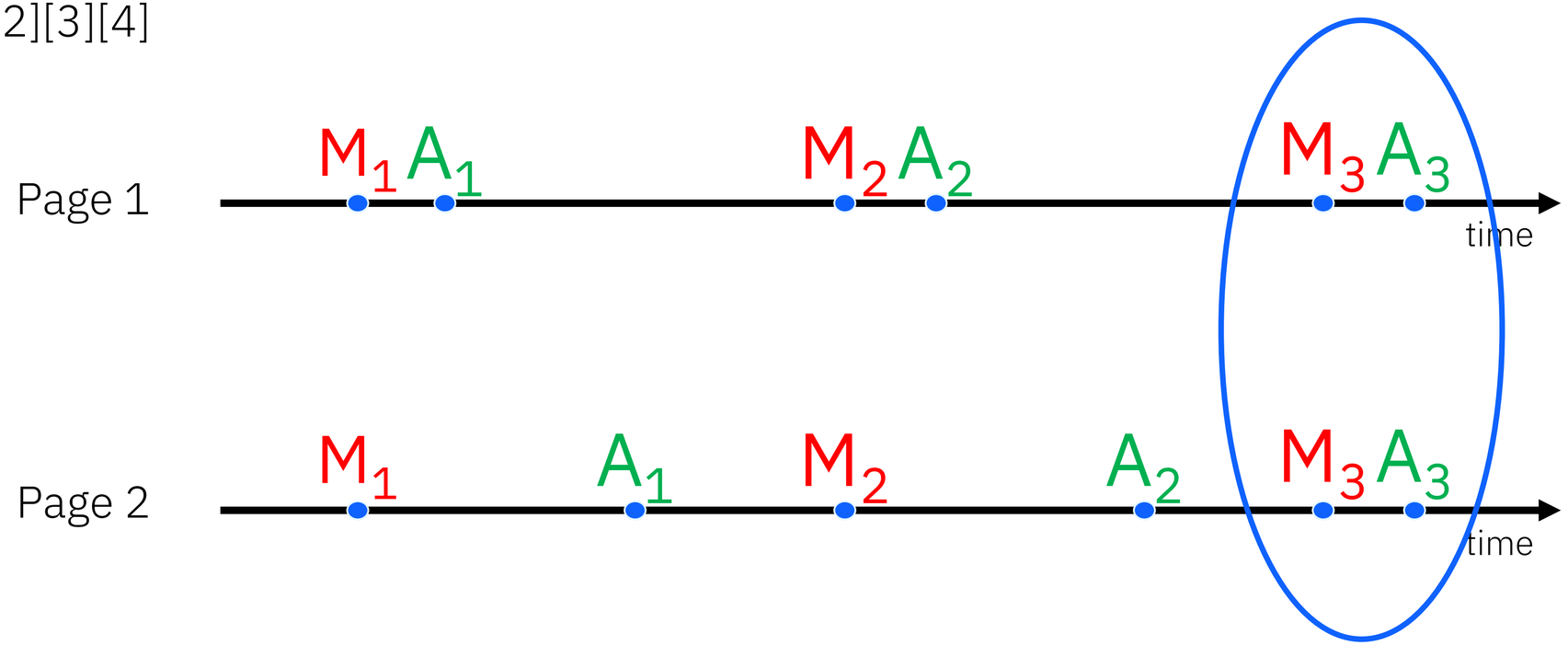
A page migration makes sense only if its network costs are lower than the benefits

Unnecessary page migrations can lead to contention across processes and/or systems accessing memory on the same device

Beyond Linux AutoNUMA and TPP

Existing techniques predict the hotness of a page by:

- Last access measurements (e.g., last NUMA hinting fault) [1][2][3][4]



Using only last access measurements ignores past access patterns and makes the two pages have the same “hotness”

[1] AutoNUMA: https://mirrors.edge.kernel.org/pub/linux/kernel/people/andrea/autonuma/autonuma_bench-20120530.pdf
[2] TPP: https://dl.acm.org/doi/10.1145/3582016.3582063
[3] Nimble: https://dl.acm.org/doi/10.1145/3297858.3304024
[4] MultiClock: https://ieeexplore.ieee.org/document/9773194

Beyond Linux AutoNUMA and TPP

Existing techniques predict the hotness of a page by:

- Use of exponentially weighted moving averages (EWMA) to smooth previous measurements [1][2]

Works well with stationary access patterns (small measurements variation) but we have measured high prediction errors for dynamic ones.

[1] Hemem: https://dl.acm.org/doi/pdf/10.1145/3477132.3483550
[2] MemTis: https://dl.acm.org/doi/10.1145/3600006.3613167

Beyond Linux AutoNUMA and TPP

Most of the existing techniques migrate pages based on a target threshold [1][2][3]

Not taking the network (fabric) cost into account leads to suboptimal decision where the cost of the migration is higher than the actual benefit from accessing local memory

[1] AutoNUMA: https://mirrors.edge.kernel.org/pub/linux/kernel/people/andrea/autonuma/autonuma_bench-20120530.pdf
[2] TPP: https://dl.acm.org/doi/10.1145/3582016.3582063
[3] Nimble: https://dl.acm.org/doi/10.1145/3297858.3304024

Beyond Linux AutoNUMA and TPP

Creates ground for exploring alternative solutions.

Can we use machine learning for predicting page hotness based on historical patterns and network related costs?

- What is the impact of inferencing a ML model in the critical path of system memory management?
- Which type of models could we be using?

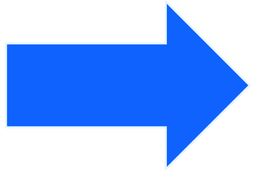
STAY TUNED!!!

Applications to virtualization

Leverage disaggregated memory for KVM virtual machine migration

Today’s migration based on pre/post copy for improving migration of memory pages between hosts:

- (post-copy) Blocking page faults for triggering page transmission
- (pre-copy) Long migration time for write intensive workloads



zero-copy VM migration leveraging disaggregated memory

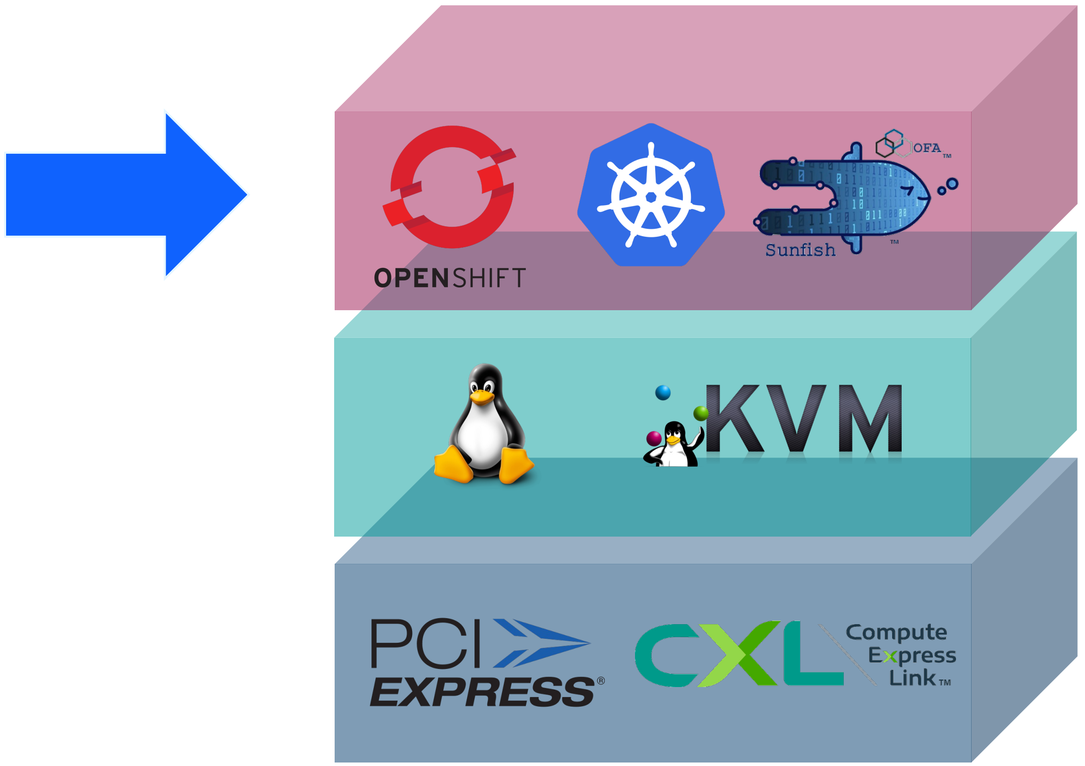
- Memory is remapped to a remote host instead of copied
- Minimal VM blackout as only minimal data is transferred

Many more details in Andreas Grapentin’s presentation

What’s next

- Further research on optimized memory management techniques that take fabric costs into account
- Better integration of disaggregated hardware with virtualization
- Continuous OS drivers improvement for integration of more features from CXL (already happening)

Outline of this talk



3. Orchestration SW

2. Low-level SW

1. Hardware

Orchestration SW for CDI

Management of composable hardware lifecycle (composing, decomposing, errors management…)

Orchestration of workloads on Composable Disaggregated Infrastracture

Composable Hardware management

Programming models

Workload managers

Vendor API

Vendor API

Vendor API

Servers Pool

Server Node

RAM

Memory Pool

RAM

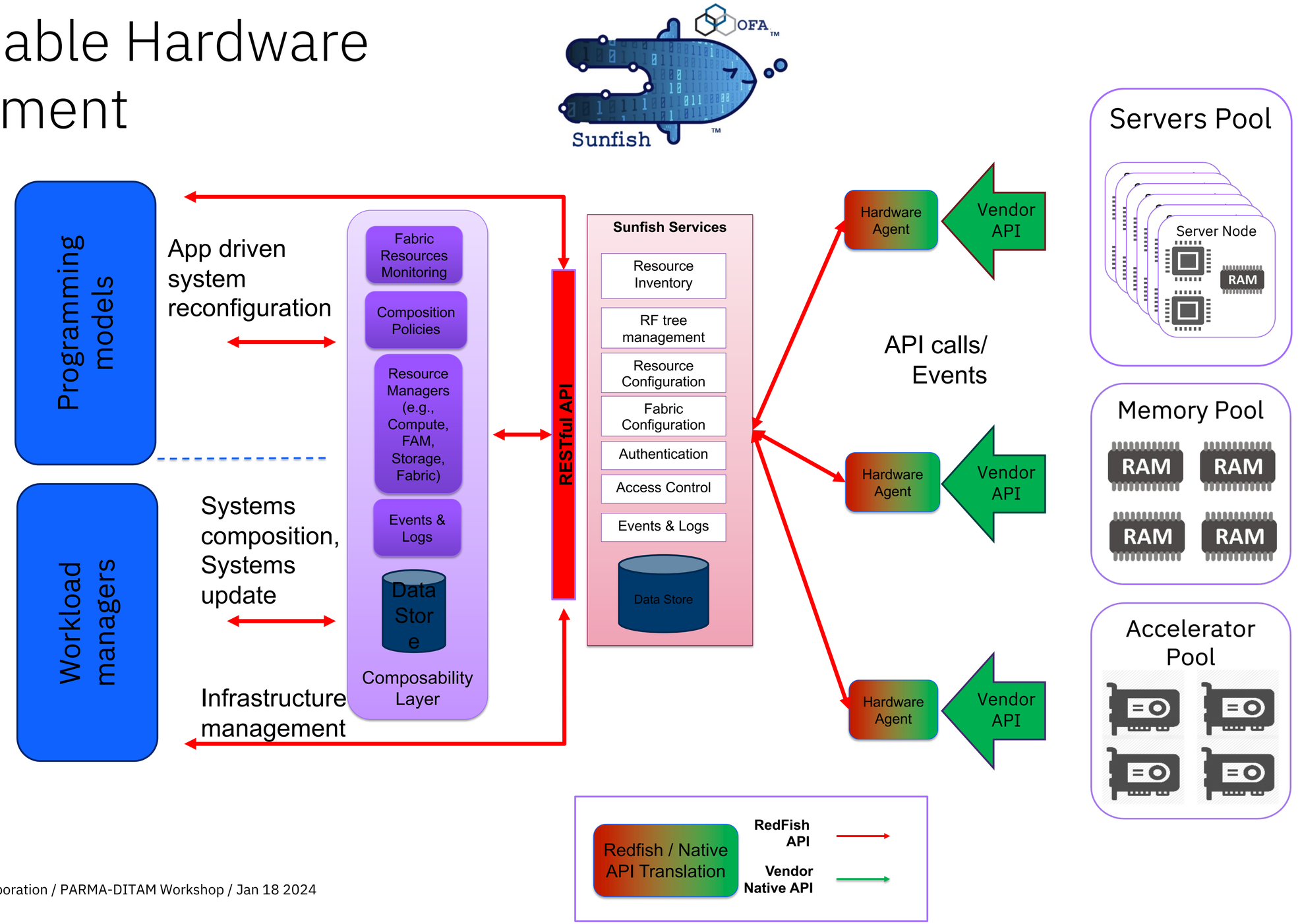
RAM

RAM

RAM

Accelerator Pool

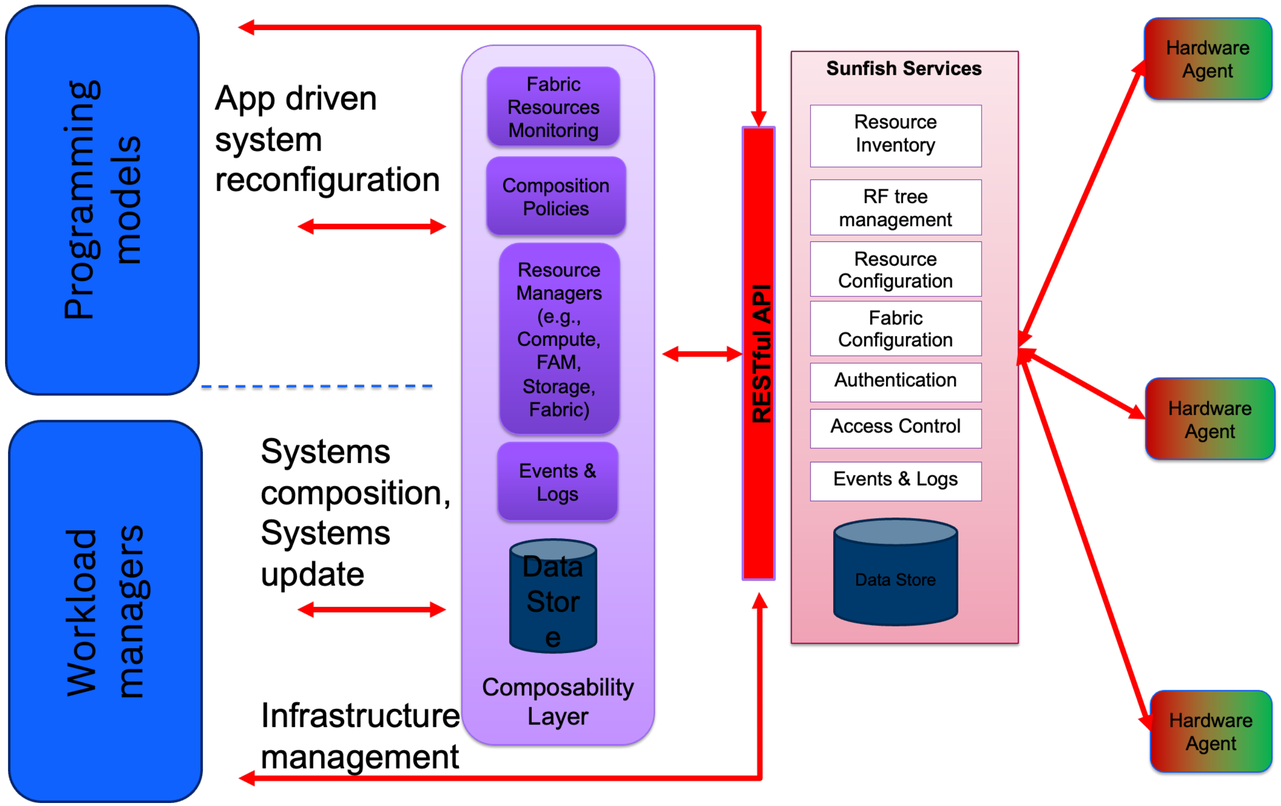
Composable Hardware management



OFA Sunfish

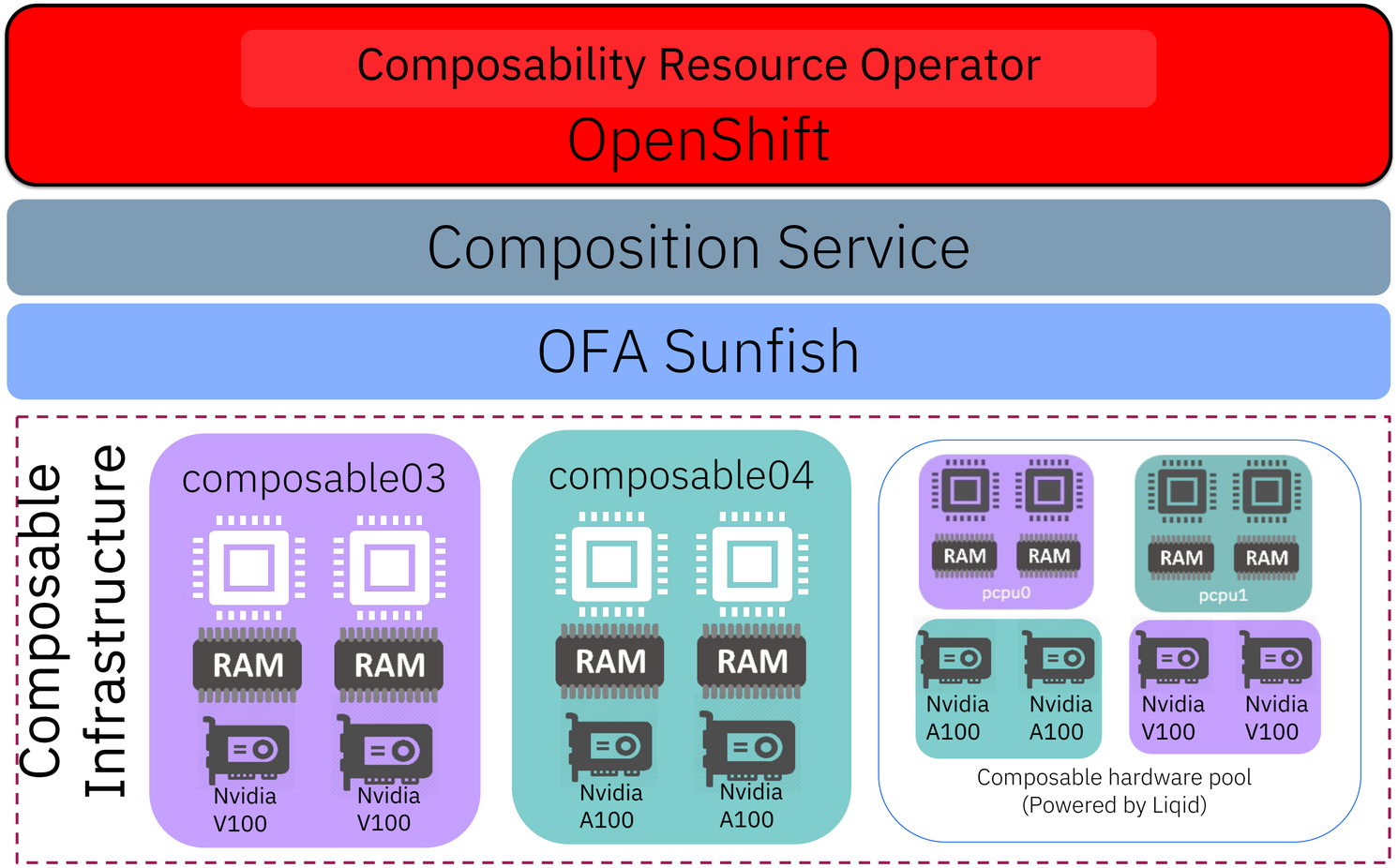
OpenFabrics Alliance response to managing complex composable systems.

- Empower infrastructure clients with:
- Unique API based on DMTF Redfish and SNIA Swordfish
- Access to the entire infrastructure (with access control)
- Dynamic (re-)configuration of fabrics to fulfil workloads requirements
- Access to multiple resource managers



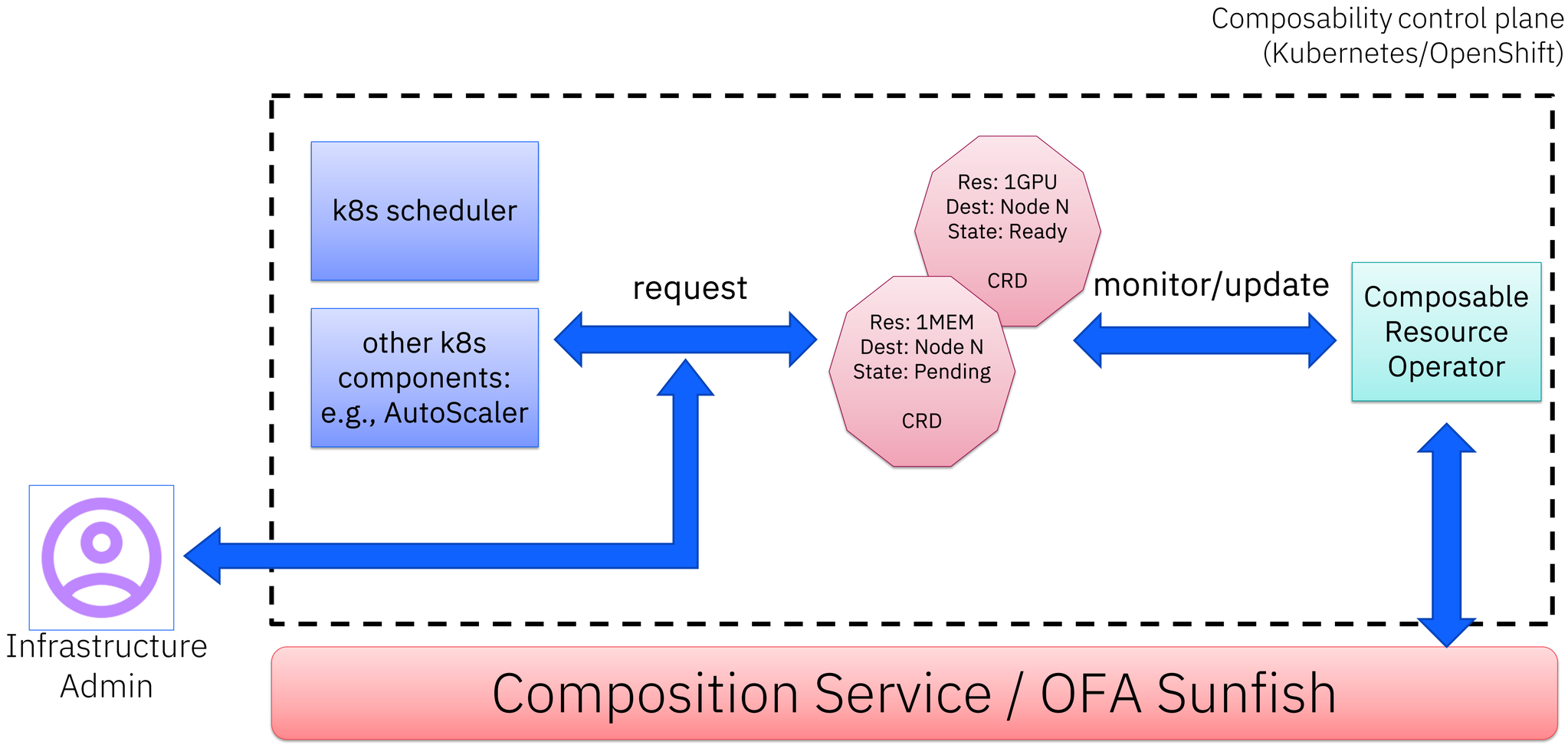
https://www.openfabrics.org/openfabrics-management-framework/

Orchestration of workloads on CDI



https://research.ibm.com/blog/composable-systems-openshift

Composability Resource Operator



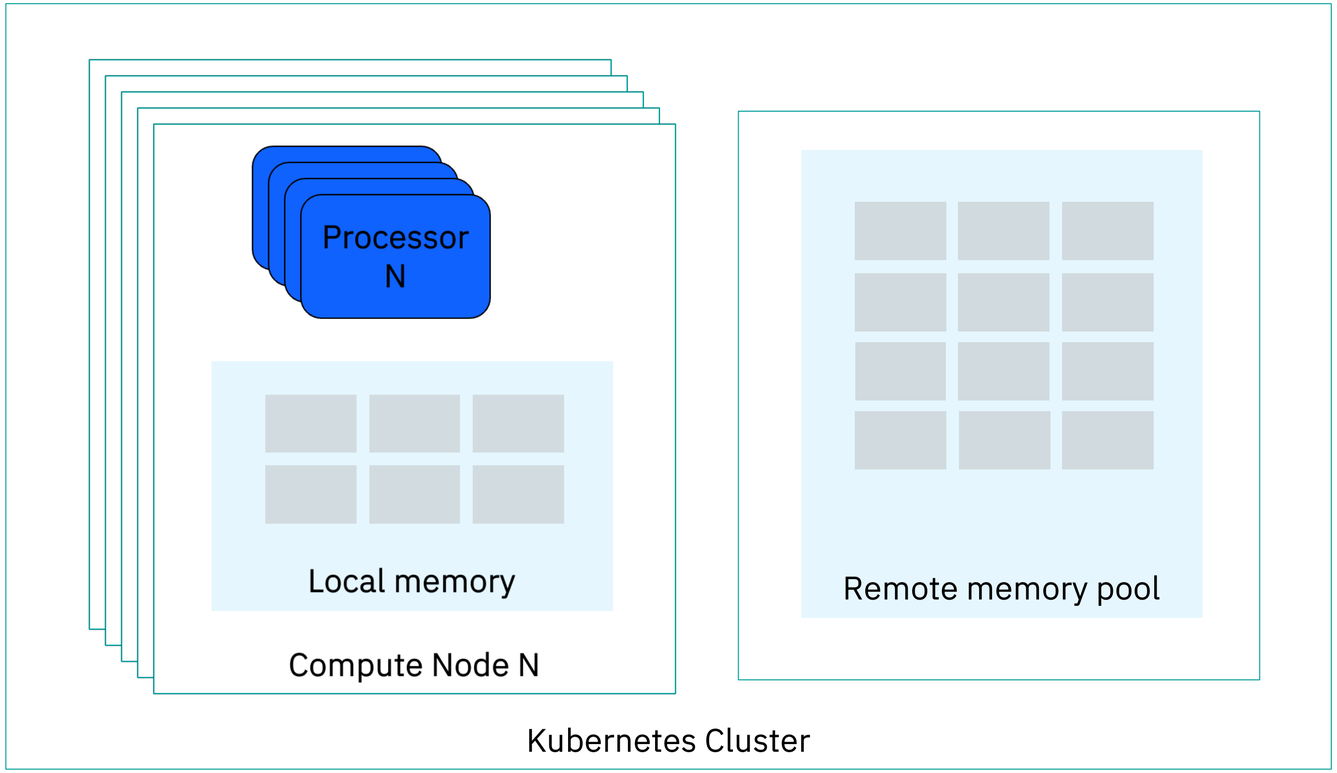
Composition Request

```
apiVersion: com.ie.ibm.hpsys/v1alpha1
kind: CompositionRequest
metadata:
  name: composabilityrequest-sample
  namespace: default
spec:
  targetNode: servernode1
  resources:
    scalar_resources:
      gpu:
        size: 2
        model: "Tesla_V100"
```

Only info on device type.

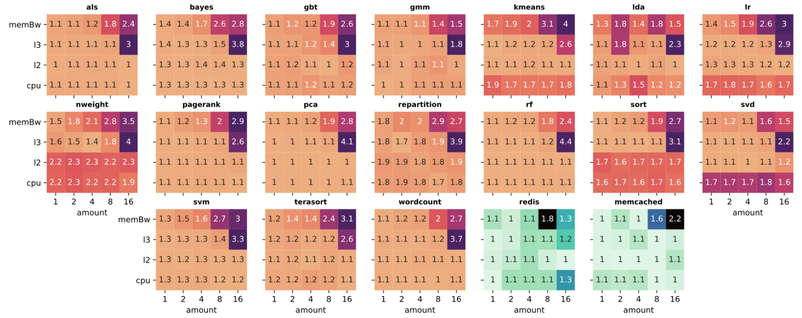
Vendor specific details are abstracted out from the user

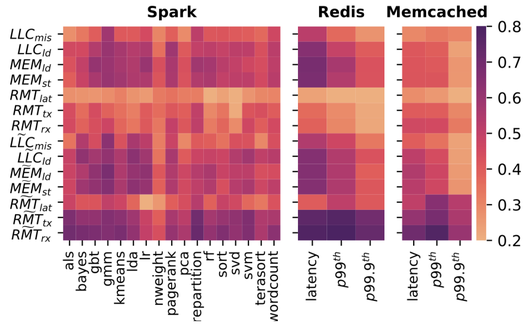
Application to memory orchestration



How do we decide if an application scheduled on the cluster should use local or remote memory?

Application to memory orchestration

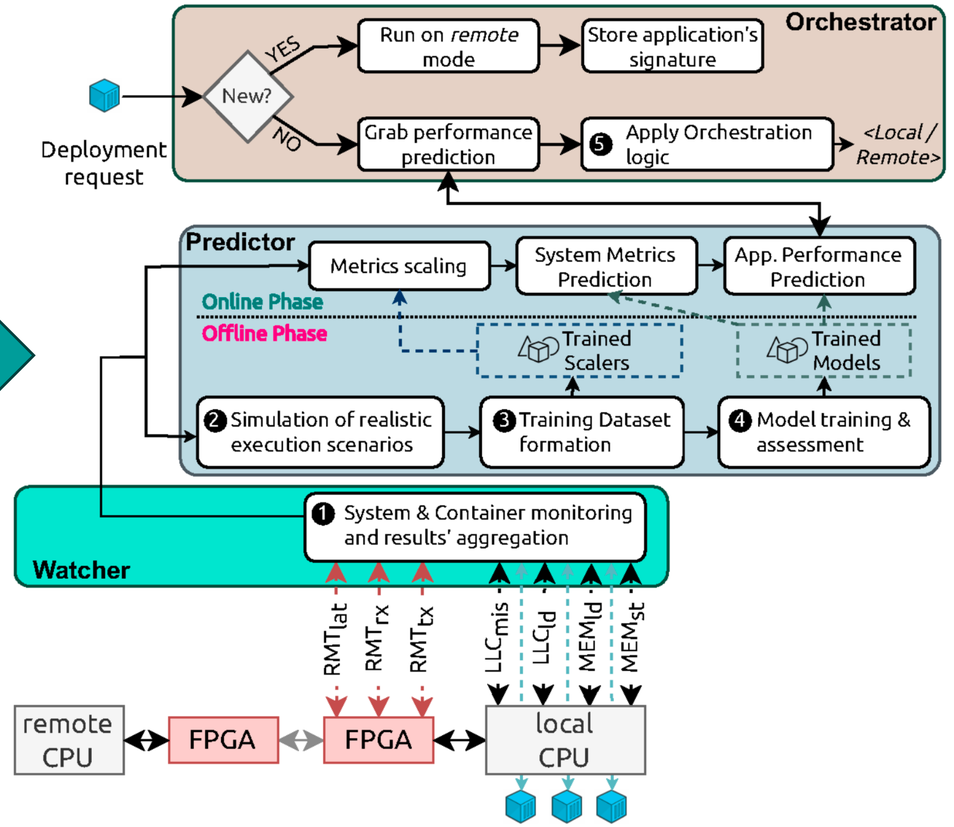




Characterization of applications under interference and correlation analysis between application performance and historical (120 sec) and runtime system performance metrics (cpu cache, local memory, remote memory).

LLCmiss, MEMld, MEMst, RMTlat, RMTtx....

ADRIAS[1]



almost 1/3 of deployed applications on remote memory with ~10% performance impact

outperforms naive scheduling approaches (random and round-robin), by providing up to ×2 better performance

[1] https://ieeexplore.ieee.org/abstract/document/10070939

What’s next

- Community to converge on one standard for composable hardware management (Sunfish 😉)
- Explore integration with the Kubernetes/OpenShift scheduler

- Explore use of disaggregated resources in programming models and APIs (e.g., improved mpi collectives)

Summary

Hardware:

- Some level of composability available over PCIe and CXL ramping up fast
- Ad-hoc configurations required to match available hardware
- Most of todays devices are not really designed for being plugged/unplugged from a live system

OS Support

- Further research is required into managing remote disaggregated memory

Orchestration

- No de-facto standard for management of composable hardware
- Fragmented integration with workloads management frameworks

Back to the original question

Composable Systems: are we there yet?

Not yet, but the basic components are in place