KubeCon | CloudNativeCon

North America 2024

KubeCon CloudNativeCon
North America 2024

Architecting Tomorrow: The Heterogeneous Compute Resources for New Types of Workloads

Alexander Kanevskiy, Intel
2024-11-13

Legal Notices and Disclaimers

Why are you talking about this subject?

- Hardware engineer by education
- Software engineer by career
- Working on Kubernetes since 2016
  - enabling Intel accelerators: 2017 - …
  - platform enablement, performance and power: 2018 - …
- CNCF TAG-Runtimes
  - Container Orchestrated Devices WG
- SIG-Node and WG Device Management
- CDI, DRA, NRI and related resource management topics

Alexander Kanevskiy
Principal Engineer, Cloud Software
Intel

https://kad.github.io/presentation-recordings

Why talking about this subject now?

- Raise of AI workloads caught CNCF and Kubernetes ecosystems not well prepared for extensive use of accelerator devices
  - Device Plugins were not good enough
  - Dynamic Resource Allocation (DRA) is on the way to Beta and then eventually to GA
- Evolution of modern CPUs from all the vendors is progressing faster than CNCF software ecosystem can adopt to it
- New reality constantly breaking assumptions on hardware or workload behaviors
- New challenges arising. Let’s be prepared for them this time

What to expect from this presentation?

- The information about hardware is based on my experience, understanding and interpretation of information coming from publicly available articles, whitepapers and documentation
- Some of the hardware properties are hidden in CSPs, but available in enterprise infrastructures

- This presentation
  - focuses on hardware aspects and implications to cloud workloads
  - not going to tell which node instance type to select
  - not going to tell what shall you do to optimize any of parameters of your particular or specific example workload

- However, it is intended to explain in simple words to what pay attention to

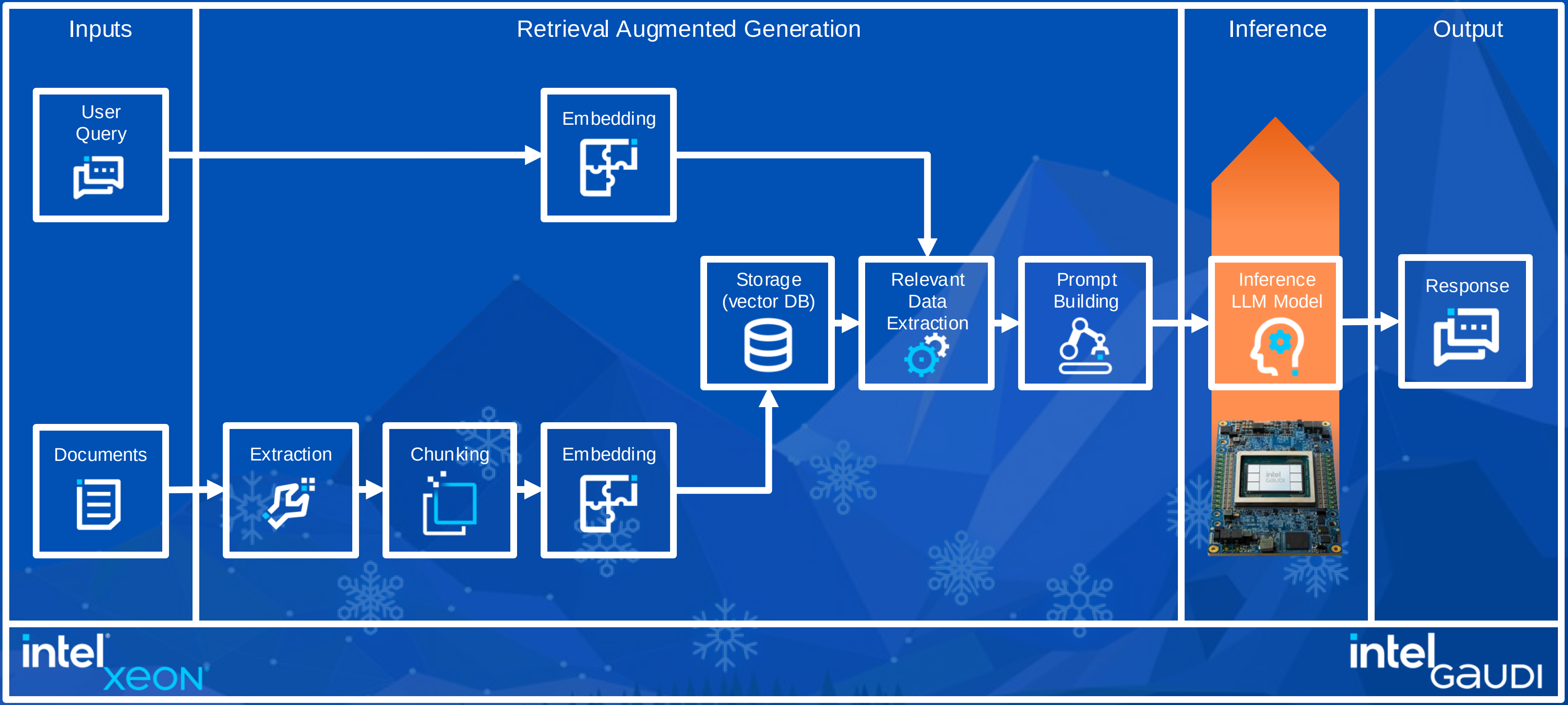
- In other words: I can’t give you fish, but will try my best to teach how to fish

What also can be interesting to listen to

- Kubernetes WG Device Management - Advancing K8s Support for GPUs - John Belamaric, Google; Patrick Ohly, Intel; Kevin Klues, NVIDIA
  - Wednesday November 13, 2024 14:30 – 15:05
  - https://sched.co/1hovp
- Platform Performance Optimization for AI - a Resource Management Perspective - Antti Kervinen, Intel & Dixita Narang, Google
  - Wednesday, November 13, 2024, 16:30 – 17:05
  - https://sched.co/1i7m0
- Better Together! GPU, TPU and NIC Topological Alignment with DRA - John Belamaric, Google & Patrick Ohly, Intel
  - Friday November 15, 2024 11:00 - 11:35
  - https://sched.co/1i7pv

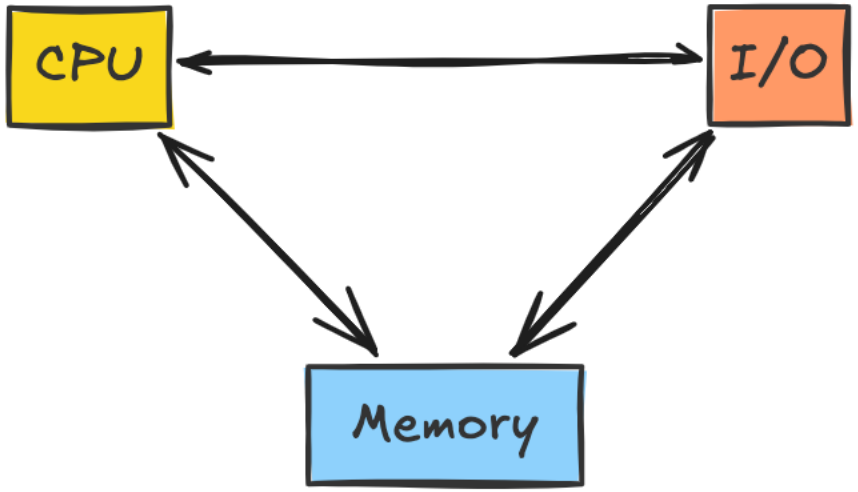
OPEA: example of RAG in ChatQnA

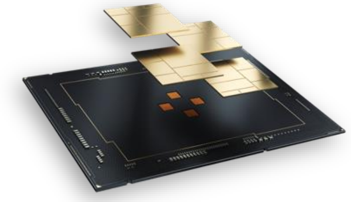
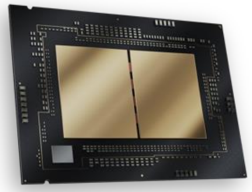
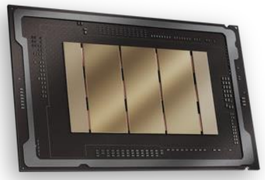
KubeCon CloudNativeCon North America 2024



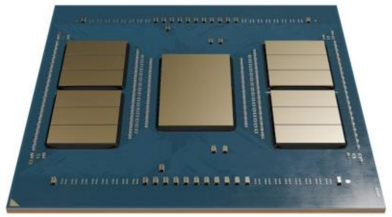
https://opea.dev/

Compute Resources

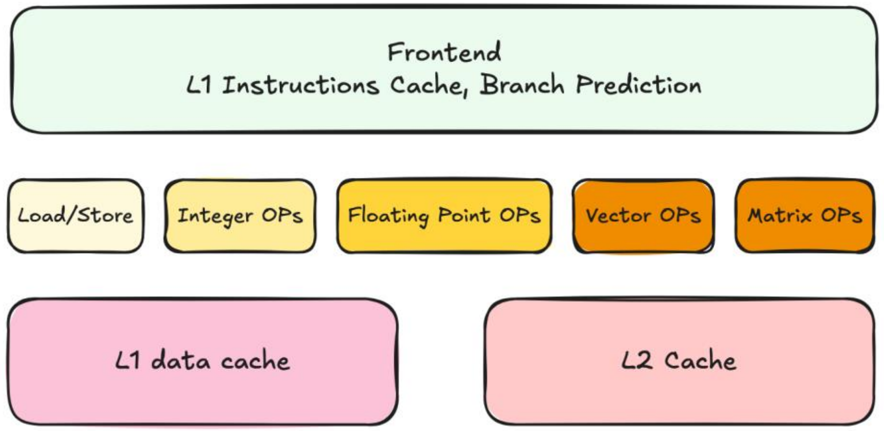


Anatomy of Cores



Anatomy of a single CPU core

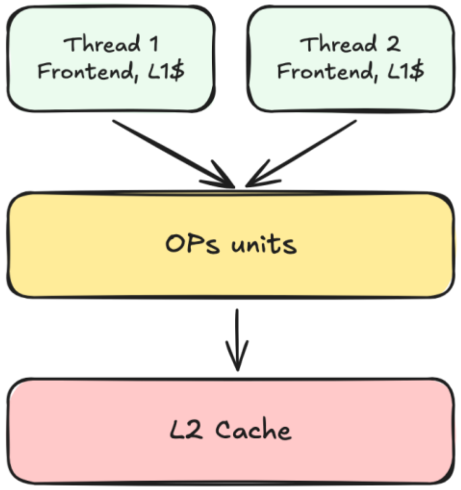


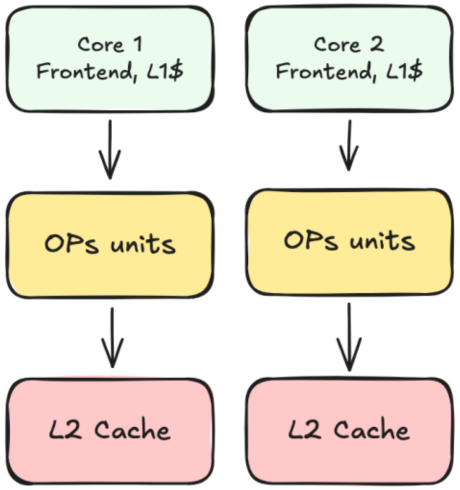
Application Type	Store/Load	Integer	Floating-Point	Vector	Matrix
Microservices	20-30%	40-60%	5-10%	5-10%	<5%
Web Applications (Front-End + Back-End)	20-40%	40-60%	1-5%	<5%	<5%
Data-Intensive Applications (Big Data, ETL)	30-40%	30-50%	20-40%	5-10%	10-20%
CI/CD Pipelines	10-30%	40-60%	<5%	<5%	<5%
Machine Learning / AI	10-20%	10-20%	50-70%	Up to 70%	Up to 70%
Gaming Applications (Real-time, Multiplayer)	10-20%	40-60%	10-30%	10-20%	10-20%
Content Delivery & Media Streaming	30-50%	20-40%	10-20%	10-20%	5-10%
Serverless Frameworks (Event-driven)	10-30%	40-60%	<5%	<5%	<5%
Big Data Analytics (Kafka, Spark, Hadoop)	30-40%	20-40%	20-40%	5-10%	10-20%
Database Applications (Relational, NoSQL)	20-40%	50-70%	5-10%	<5%	<5%
Edge Computing Applications	20-40%	40-60%	5-10%	5-10%	5-10%

Minimal CPU grouping: SMT vs dual-core

- SMT (or Intel® Hyper-Threading Technology)
  - Two frontends for single set of OPs units
  - Separate L1, but common L2 cache

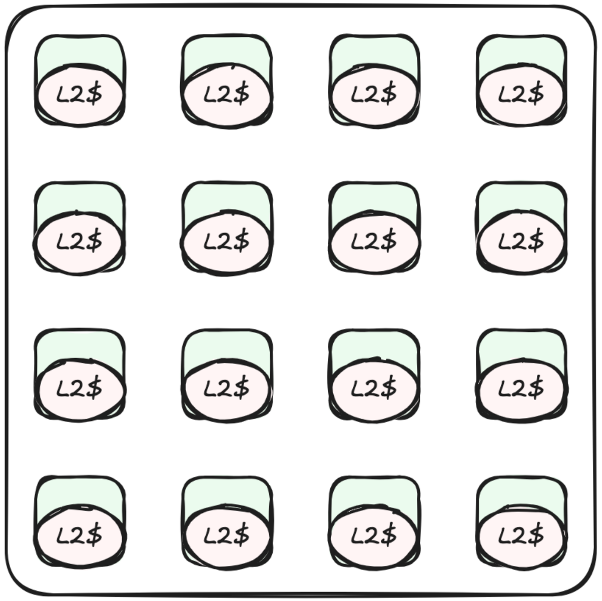
- Multi-core core groups
  - Independent sets components: frontends, OPs units, L1 and L2 caches



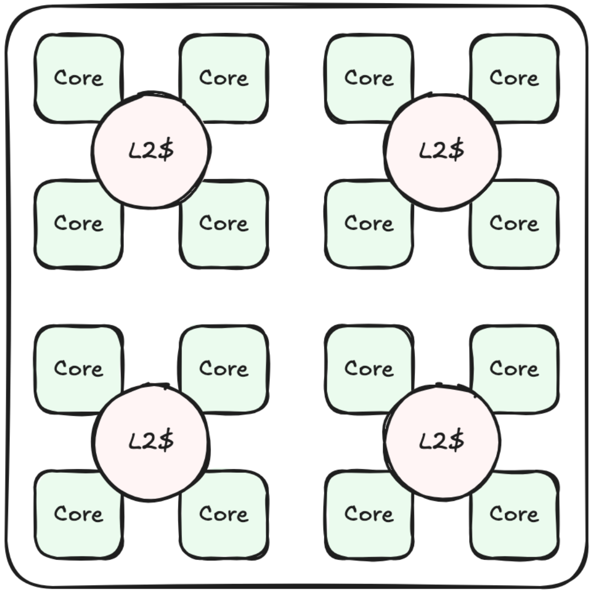


CPU Grouping: clusters of the cores

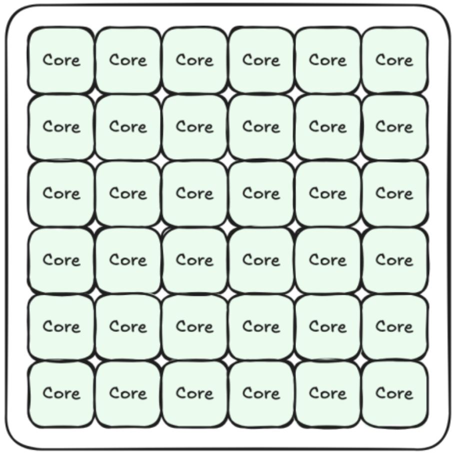
- Traditional groups of the cores
  - Physically located close to each other



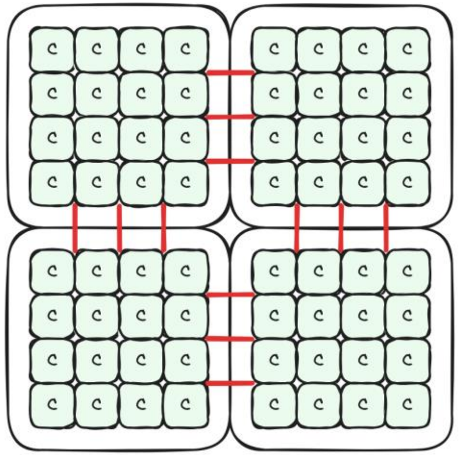
- Clusters of the cores
  - Common on E-Cores and on ARM designs



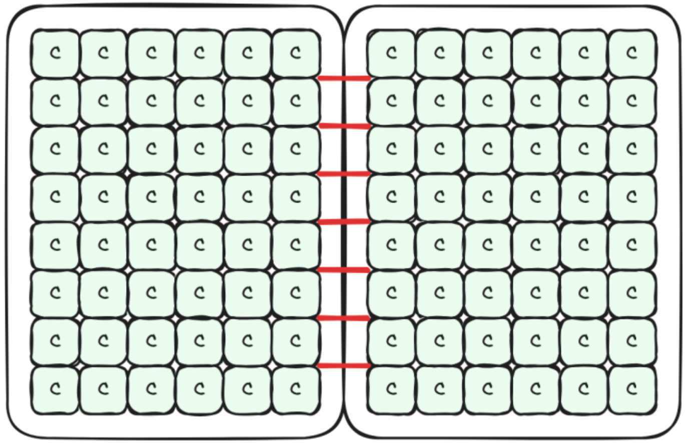
CPU Grouping: tiles, chiplets or dies



Traditional layout

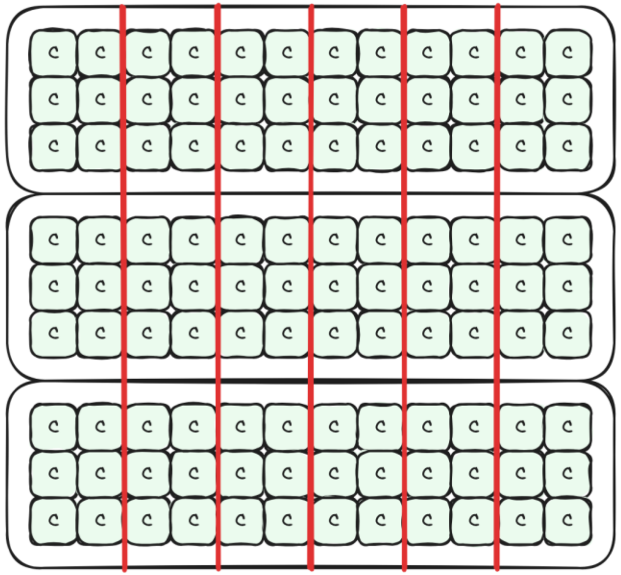


4th Gen Intel® Xeon®

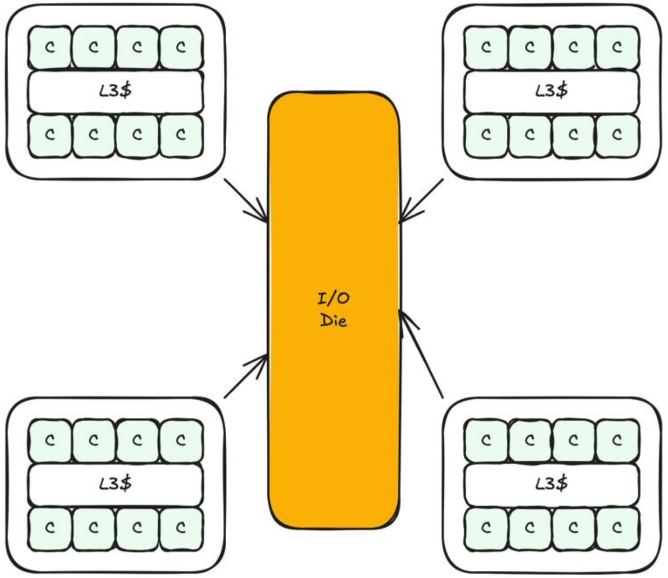


5th Gen Intel® Xeon®

CPU Grouping: tiles, chiplets or dies



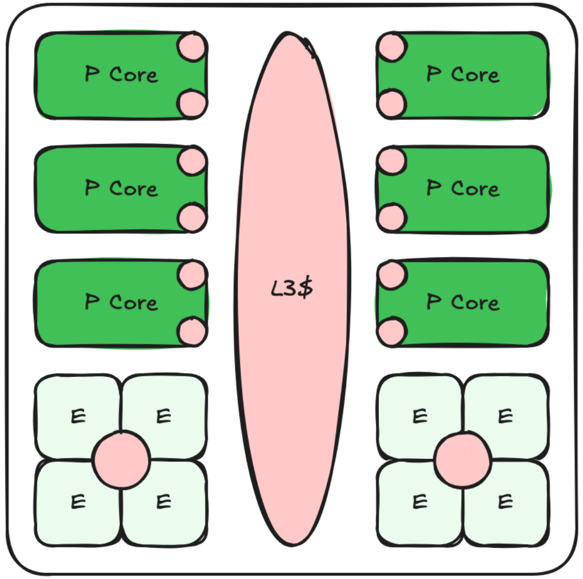
Intel® Xeon® 6



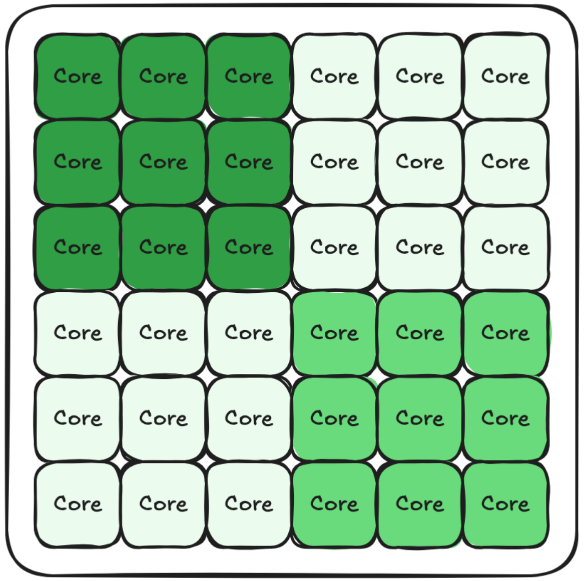
AMD Epyc

CPU Power

- P and E cores in hybrid CPUs

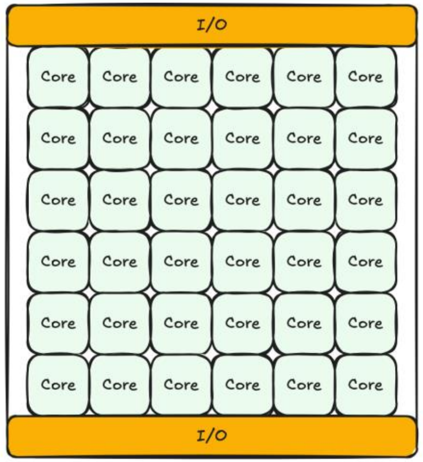


- Different power and performance settings within CPU with same core types

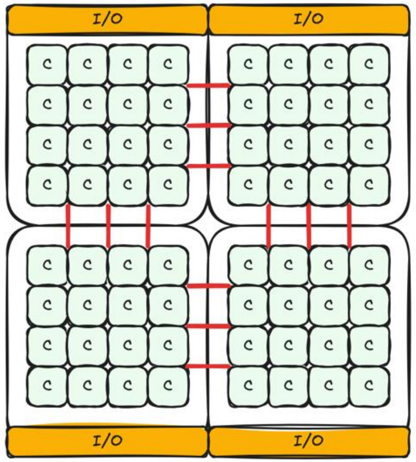


Let’s look at I/O

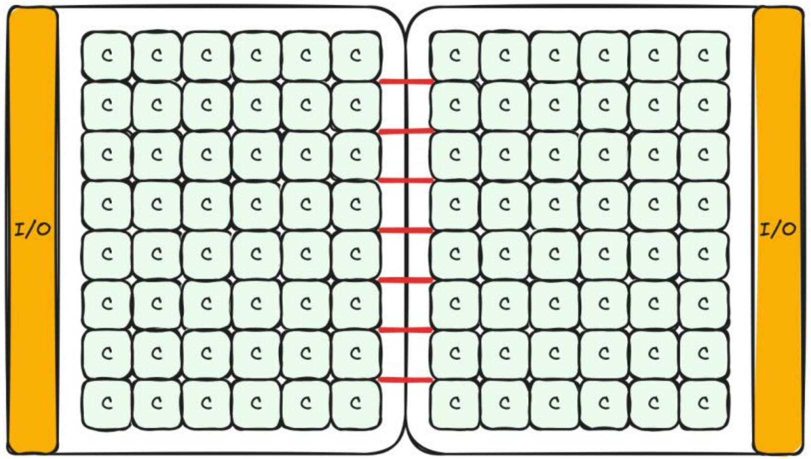
I/O: PCIe, CXL and integrated accelerators



Traditional

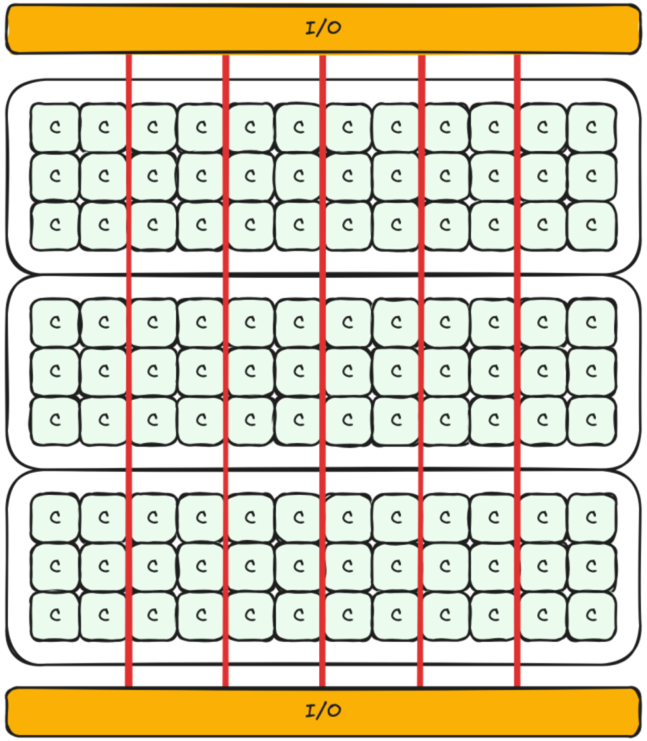


4th Gen Intel® Xeon®

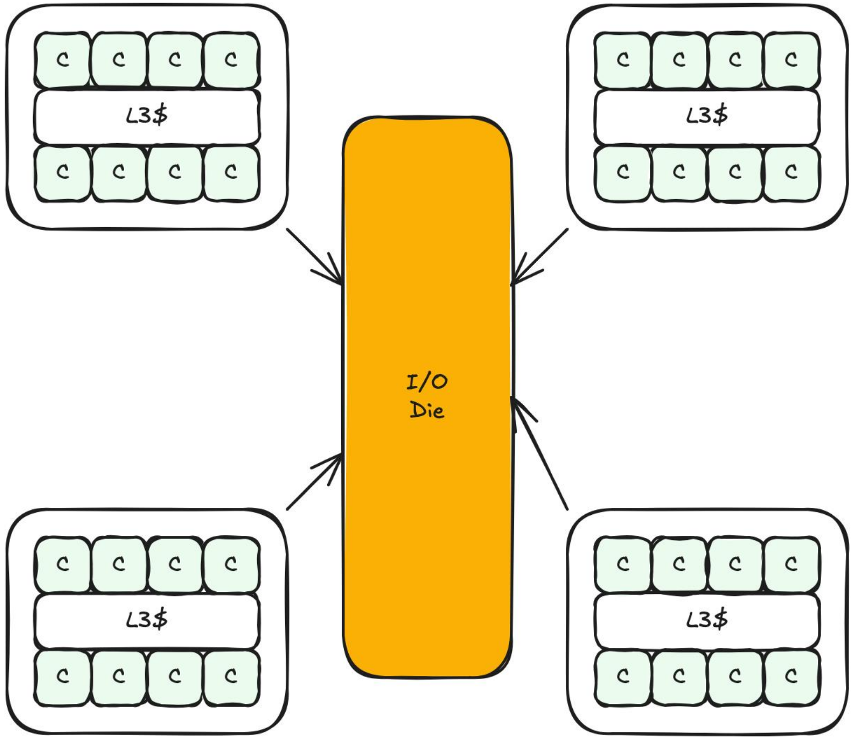


5th Gen Intel® Xeon®

I/O: PCIe, CXL and integrated accelerators



Intel® Xeon® 6



AMD Epyc

Compute Express Link

CXL devices

https://computeexpresslink.org/

What is CXL?

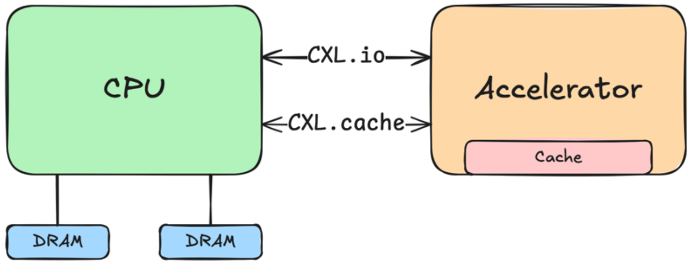
- Compute Express Link (CXL) is an open standard interconnect for high-speed, high-capacity CPU-to-device and CPU-to-memory connections, designed for high performance data center computers

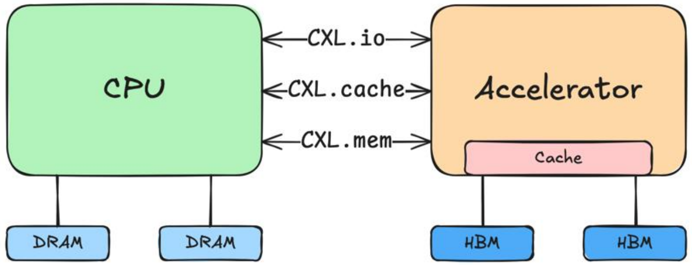
- Established in 2019
  - Founding members: Alibaba Group, Cisco Systems, Dell EMC, Meta, Google, Hewlett Packard Enterprise (HPE), Huawei, Intel Corporation, Microsoft
  - Joined later: AMD, Nvidia, Samsung Electronics, Xilinx

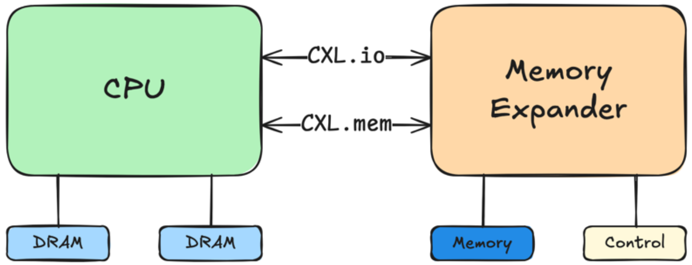
- CXL is built on the serial PCI Express physical and electrical interface and includes
  - CXL.io – based on PCIe 5.0 (and PCIe 6.0 after CXL 3.0) with a few enhancements
    - provides configuration, link initialization and management, device discovery and enumeration, interrupts, DMA, and register I/O access using non-coherent loads/stores
  - CXL.cache – defines interactions between a host and a device
    - coherently access and cache host CPU memory with a low latency request/response interface
  - CXL.mem – allows host CPU to coherently access device-attached memory
    - load/store commands for both volatile (RAM) and persistent non-volatile (flash memory) storage

CXL device types

- Type 1 (CXL.io and CXL.cache)
  - coherently access host memory, specialized accelerators (e.g. smart NIC) with no local memory.
  - Devices rely on coherent access to host CPU memory.
- Type 2 (CXL.io, CXL.cache and CXL.mem)
  - coherently access host memory and device memory, general-purpose accelerators (GPU, ASIC or FPGA) with high-performance GDDR or HBM local memory
  - Devices can coherently access host CPU's memory and/or provide coherent or non-coherent access to device local memory from the host CPU
- Type 3 (CXL.io and CXL.mem)
  - allow the host to access and manage attached device memory, memory expansion boards and persistent memory
  - Devices provide host CPU with low-latency access to local DRAM or byte-addressable non-volatile storage.





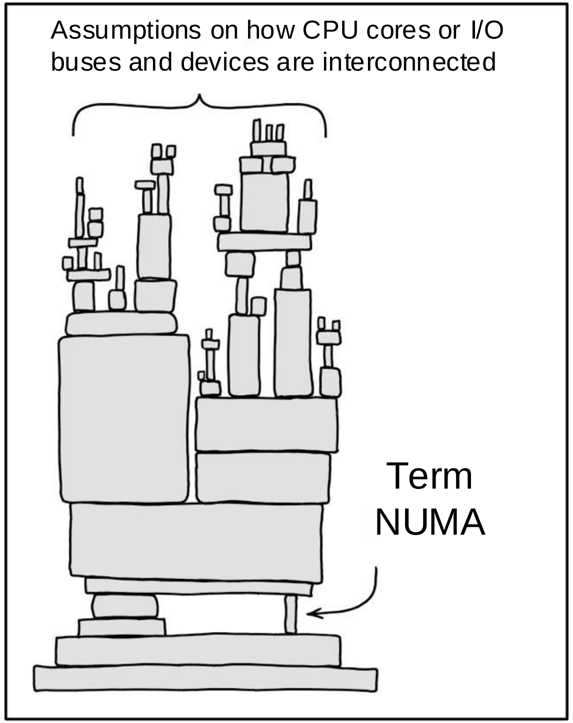


What about Memory?

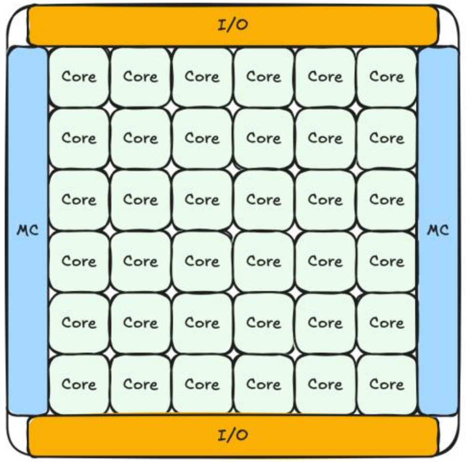
NUMA – It is all about memory

- **“NUMA”** stands for **Non-Uniform Memory Access**
- Types of memory
  - DRAM
  - HBM
  - PMEM
  - CXL.memory
- Each NUMA zone can hide underneath several types of memory
- NUMA zone can be related to physical layout or logical partitioning (such as SNC or NPS)

- And reminder:
  - “**C**” in “NUMA” stands for “CPU”
  - “**P**” in “NUMA” stands for “PCIe”

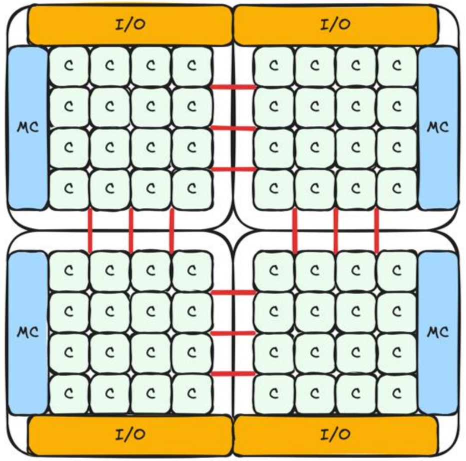

Assumptions on how CPU cores or I/O buses and devices are interconnected
Term NUMA

Memory controllers and NUMA zones



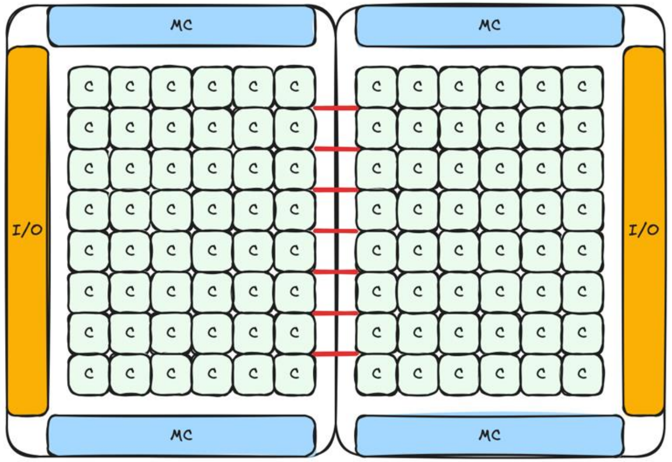
Traditional

Socket = NUMA zone
Multi-channel MCs



4th Gen Intel® Xeon®

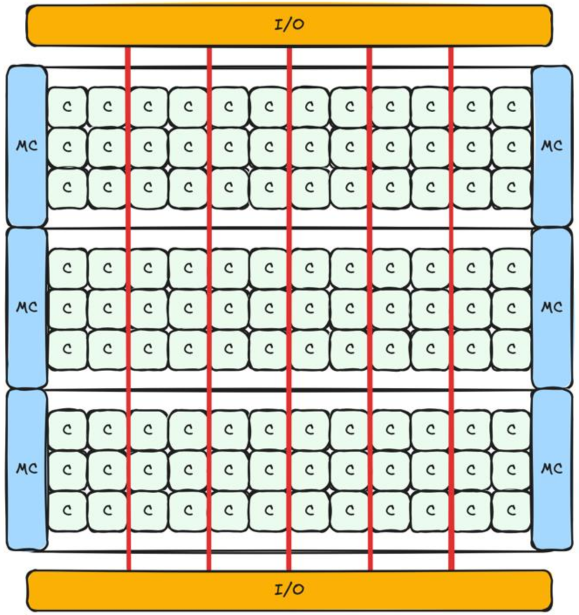
NUMA zones: 1, 2, 4
Multi-channel MCs on each tile



5th Gen Intel® Xeon®

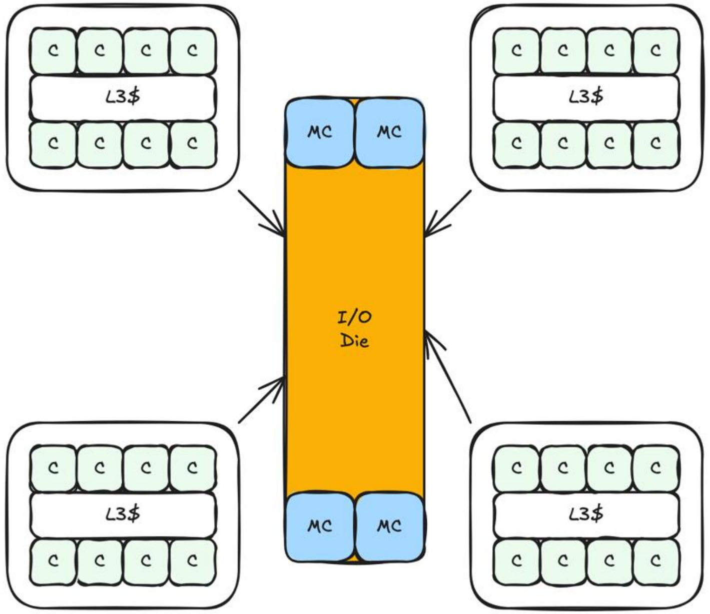
NUMA zones: 1, 2
Multi-channel MCs on each tile

Memory controllers and NUMA zones



Intel® Xeon® 6

NUMA zones: 1, 3
Multi-channel MCs on each tile



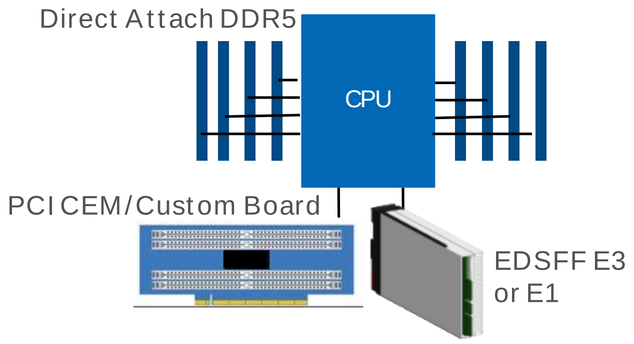
AMD Epyc

NUMA zones: 1, 4
Multi-channel MCs on I/O die

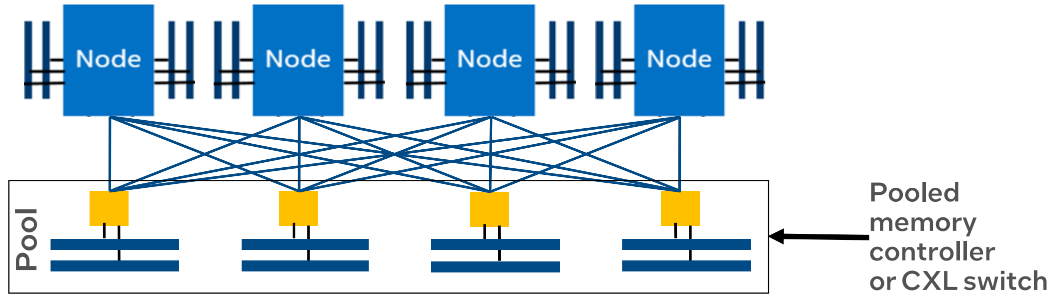
CXL.memory

CXL.Memory

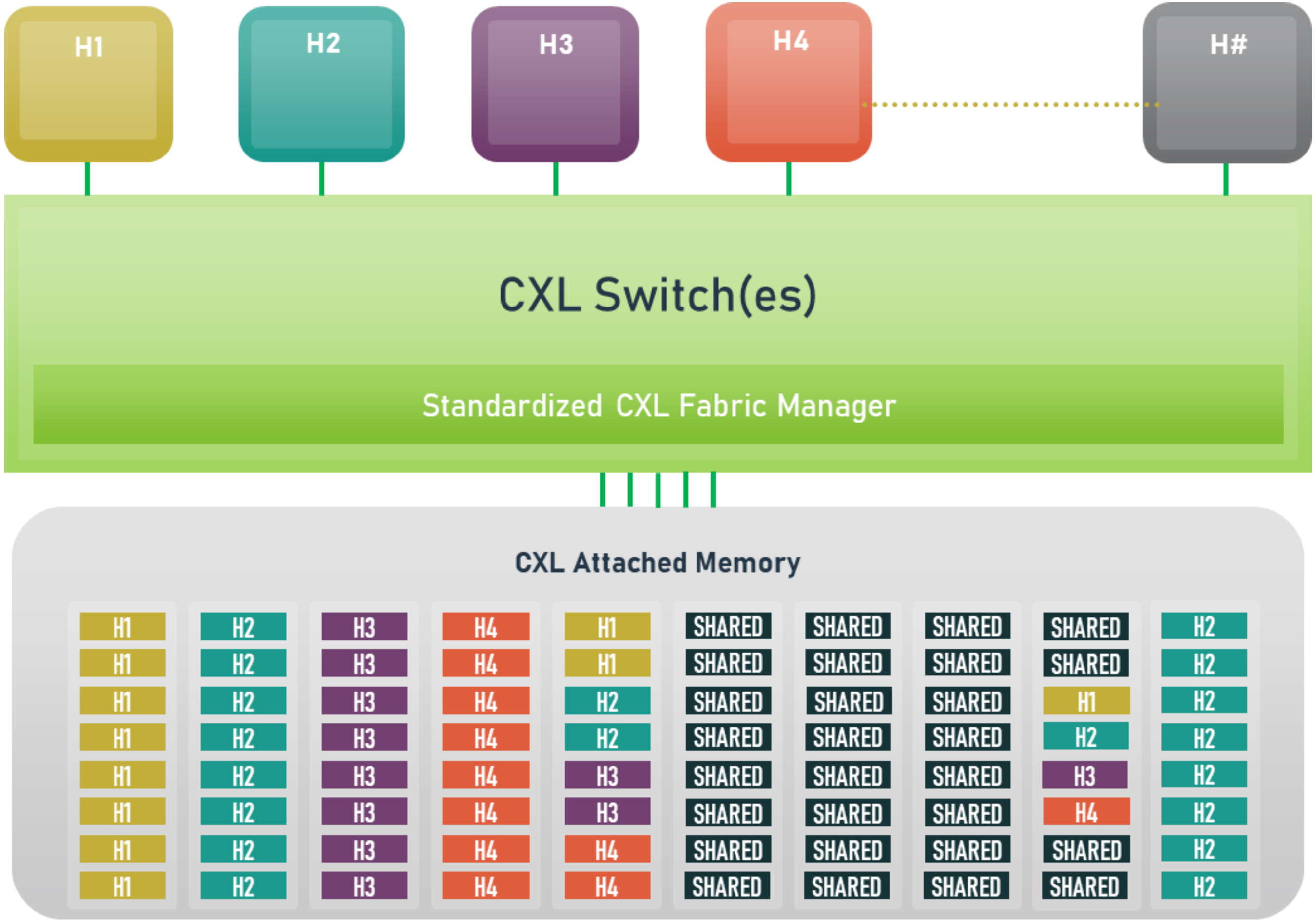
- Internal CXL boards
- Easily accessible devices in EDSFF form-factor



- Top-of-the-rack Memory Pools

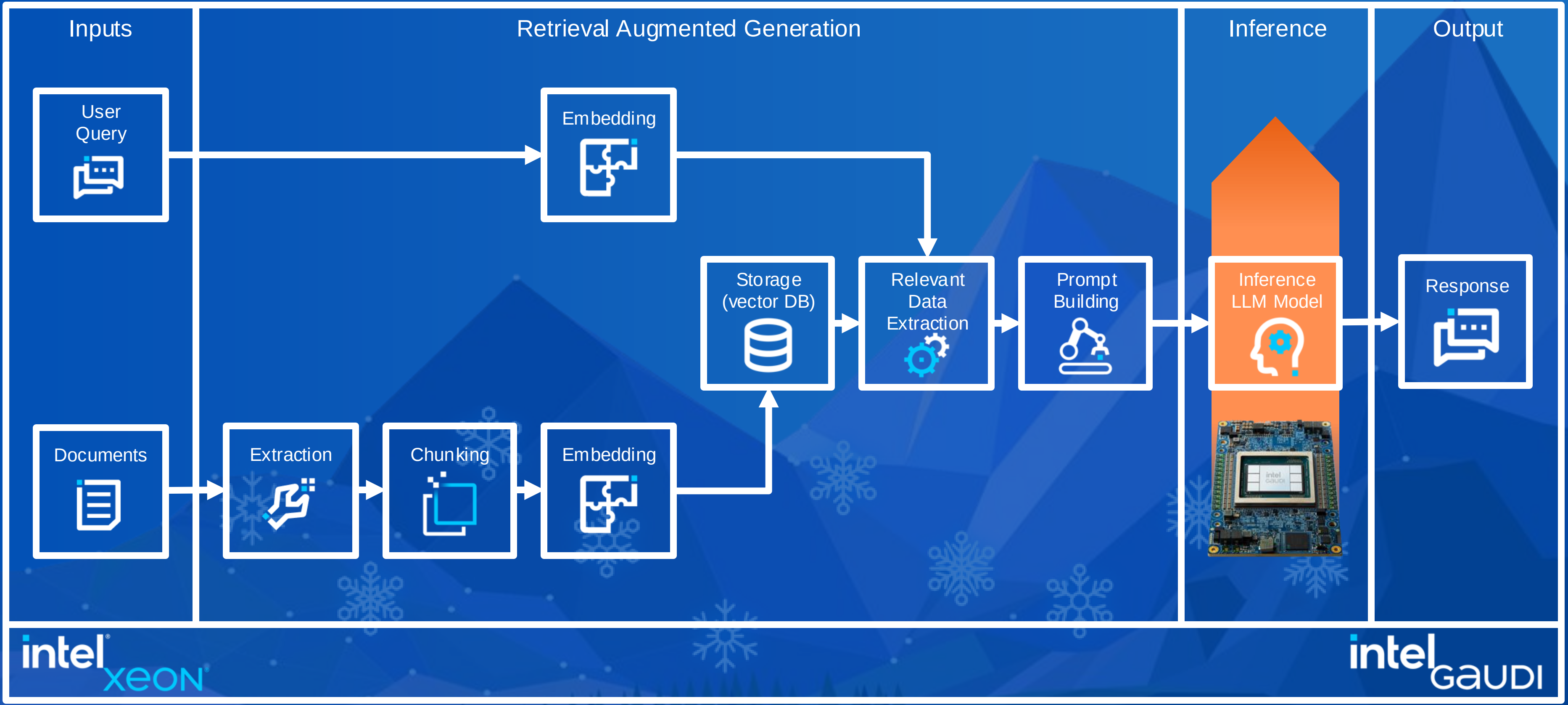


CXL memory pooling


H1
H2
H3
H4
H#
CXL Switch(es)
Standardized CXL Fabric Manager
CXL Attached Memory
H1 H1 H1 H1 H1 H1 H1 H1
H2 H2 H2 H2 H2 H2 H2 H2
H3 H3 H3 H3 H3 H3 H3 H3
H4 H4 H4 H4 H4 H4 H4 H4
H1 H1 H2 H2 H3 H3 H4 H4
SHARED SHARED SHARED SHARED SHARED SHARED SHARED SHARED
SHARED SHARED SHARED SHARED SHARED SHARED SHARED SHARED
SHARED SHARED SHARED SHARED SHARED SHARED SHARED SHARED
SHARED SHARED H1 H2 H3 H4 SHARED SHARED
H2 H2 H2 H2 H2 H2 H2 H2

Different types of workload on Node

OPEA: example of RAG in ChatQnA

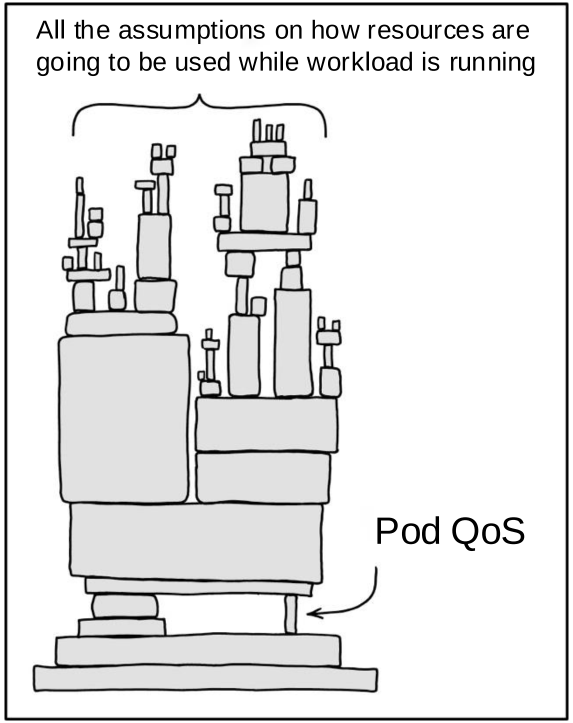


https://opea.dev/

Workload’s class of service

- Over life of Kubernetes accumulated a lot of assumptions on what kind of workload behavior it could be, and mapped those to only 3 choices in Pod QoS: Guaranteed, Burstable and BestEfforts
- Pod QoS is single value for all containers in the Pod, which leads to problems with e.g. Service Mesh containers or sidecar containers

- We need to be flexible on definition for Pod QoS and individual containers within the Pod
- Explicit declaration of QoS, instead of implicitly calculated out of requests/limits

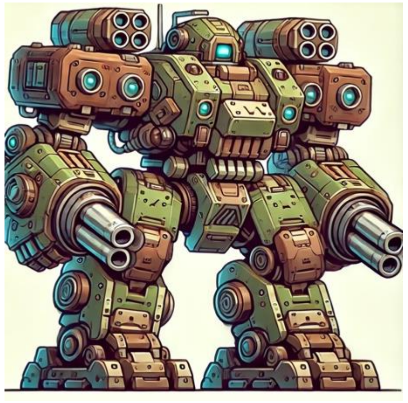

All the assumptions on how resources are going to be used while workload is running
Pod QoS

Join to the discussion in the [KEP-3008: QoS-class resources]

Native resources & Pod QoS

- All internals of Kubelet for native resources are based on subset of assumptions for few workloads from 10+ years old hardware era
- “Herd of Managers” are only care about Guaranteed QoS, which leads in significant resource waste, e.g. for accelerated AI workloads
- With different types of devices, patterns and problems with communication between CPU, memory and device might be significantly different
- For different types of applications patterns on CPU and memory usages might be also significantly different

- It is time introduce ways for workload to define **not only ”how much”, but also “how”** for requested resources
- Time to think about Pod/Container affinity/anti-affinity within the Node?



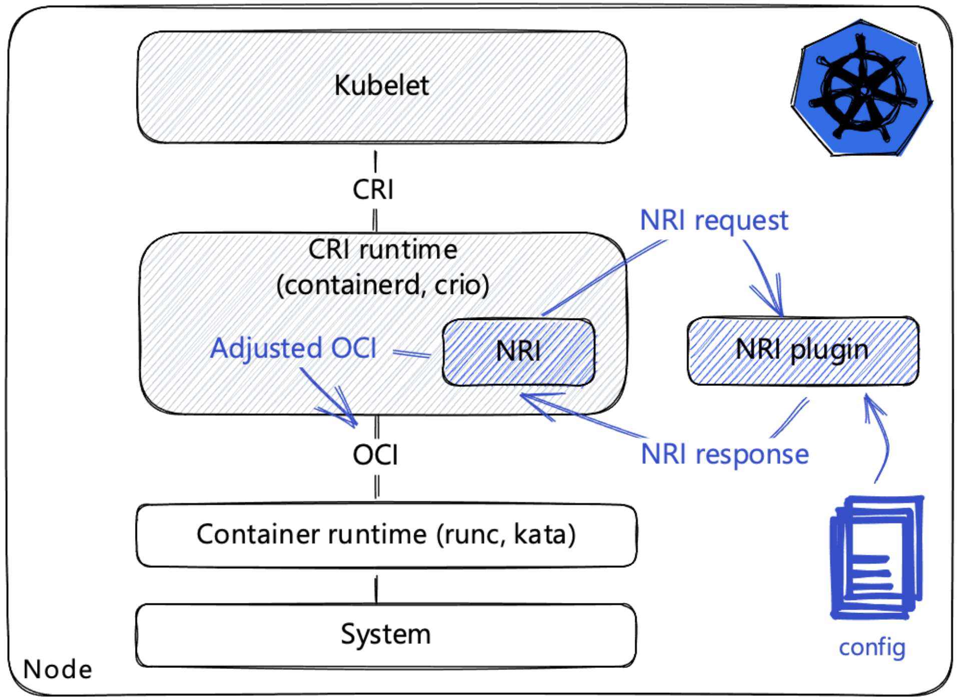
KubeCon CloudNativeCon North America 2024

How to try?

NRI: Node Resource Interface

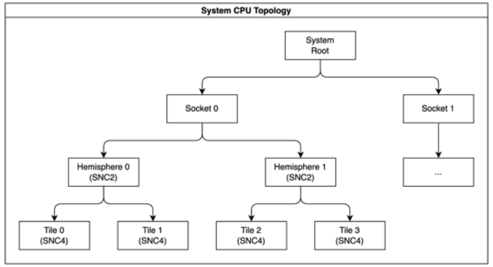
- What
  - Common plugin interface to enhance container runtimes
  - Originally, by Apple as subproject of containerd
  - Improved version adopted by CRI-O
  - Now maintained by many community participants
- Usage
  - Flexible resource management
  - Hook injection
  - Logging and debugging
  - Security introspection and enforcement
  - Prototyping Runtimes extensions
- Availability
  - Containerd 1.7+
  - CRI-O 1.26+
    - OpenShift 4.13+

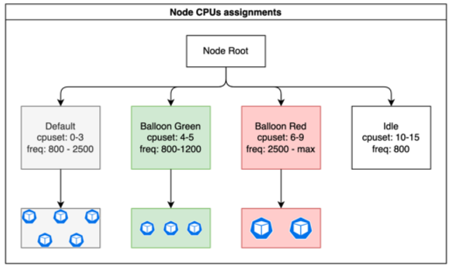
- Maturity: enabled by default in cri-o 1.30 and containerd 2.0

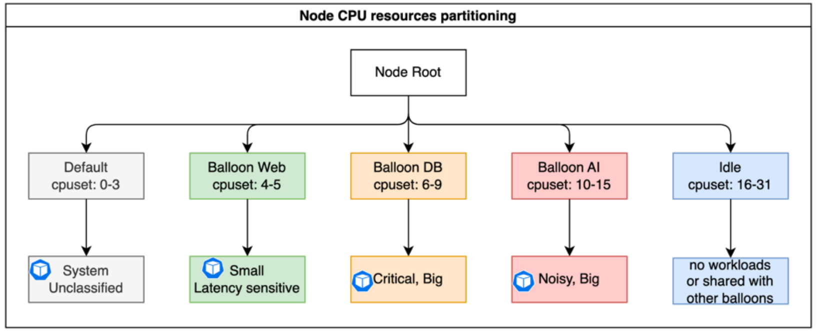


NRI: multitool for trying different ideas

- Reference resource policies plugins
- Topology-Aware
  - 0-configuration CPU, memory & device alignment
  - Burstable + Guaranteed + BestEfforts
  - Container affinity / anti-affinity
  - Support for all complex topologies of modern hardware, including P/E cores
- "Balloons": flexible workload groupings
  - Pin containers to inflating/deflating disjoint set of CPU pools
  - CPU and uncore frequency controls
  - Dynamic power and idle controls




Node CPUs assignments
Node Root
Default cpuset: 0-3 freq: 800 - 2500
Balloon Green cpuset: 4-5 freq: 800-1200
Balloon Red cpuset: 6-9 freq: 2500 - max
Idle cpuset: 10-15 freq: 800



GitHub: https://github.com/containers/nri-plugins/
Helm Charts ArtifactHUB.io: https://artifacthub.io/packages/search?repo=nri-plugins

Key takeaways: time to think and act NOW

- Observe and understand your hardware and workloads
  - It saves money and electricity if you can do same amount of work with more efficient usage
- If we want to be prepared for future hardware or software advances, it is time to think about our users: the owners of workloads
  - Stop assuming, give a way to explicitly declare what resource and how exactly it is needed for workload
- Accumulated technical debt is already huge
  - Fighting assumptions and deprecating old ideas takes years

“We can’t boil the ocean…”

… but let’s find and boil few small lakes!

Very Finnish Problems by Joel Williams

KubeCon CloudNativeCon

North America 2024

Please scan the QR Code above to leave feedback on this session

KubeCon | CloudNativeCon

North America 2024