How CXL Computational Memory Will Revolutionize Data Processing
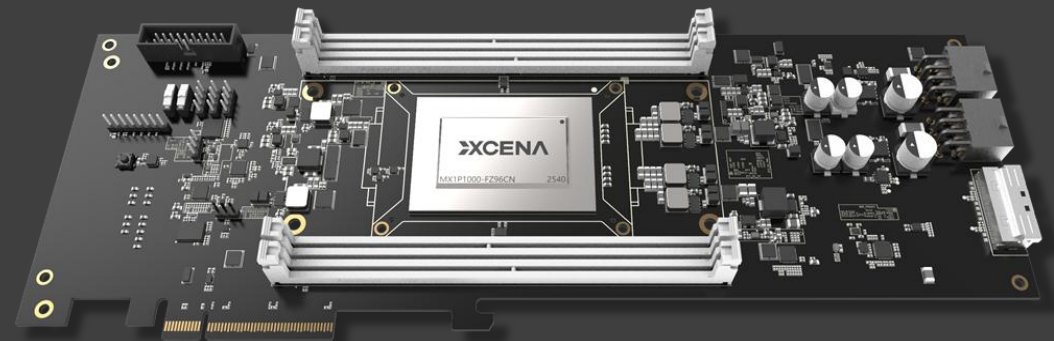
Harry Juhyun Kim

CPO and Co-founder

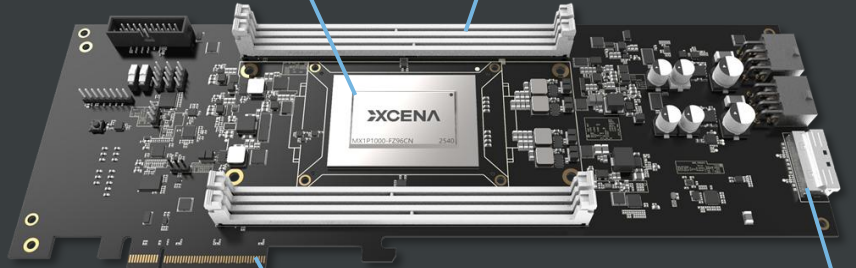# MX1: CXL MEMORY + DATA PROCESSING

XCENA

CXL COMPUTATIONAL MEMORY FOR LARGE-SCALE DATA
CHECK OUT MX1 AT BOOTH #734

Novel CXL Hardware

1000s of Custom
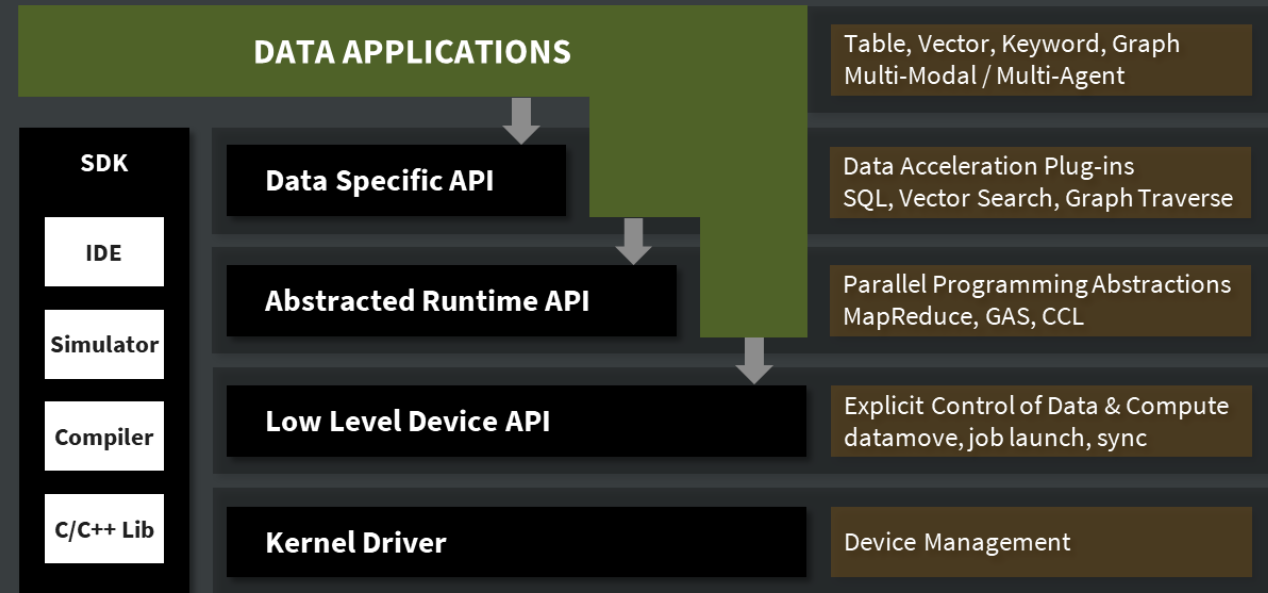RISC-V Cores
+
TFLOPS
Vector Engine

DDR5 x 4Ch
~1TB

CXL 3.0 HDM-DB
with Back-invalidation
Cache Coherence

SSD-backed
CXL Expansion

Rich Software Framework

DATA APPLICATIONS

Table, Vector, Keyword, Graph
Multi-Modal / Multi-Agent

SDK
- IDE
- Simulator
- Compiler
- C/C++ Lib

Data Specific API

Data Acceleration Plug-ins
SQL, Vector Search, Graph Traverse

Abstracted Runtime API

Parallel Programming Abstractions
MapReduce, GAS, CCL

Low Level Device API

Explicit Control of Data & Compute
datamove, job launch, sync

Kernel Driver

Device Management

FMS

AI+DATA WORLFLOWS

AI/ML IS THE BIGGEST CONSUMER OF DATA TODAY
THE LARGEST TABLES AND LAKEHOUSES ARE BUILT FOR AI
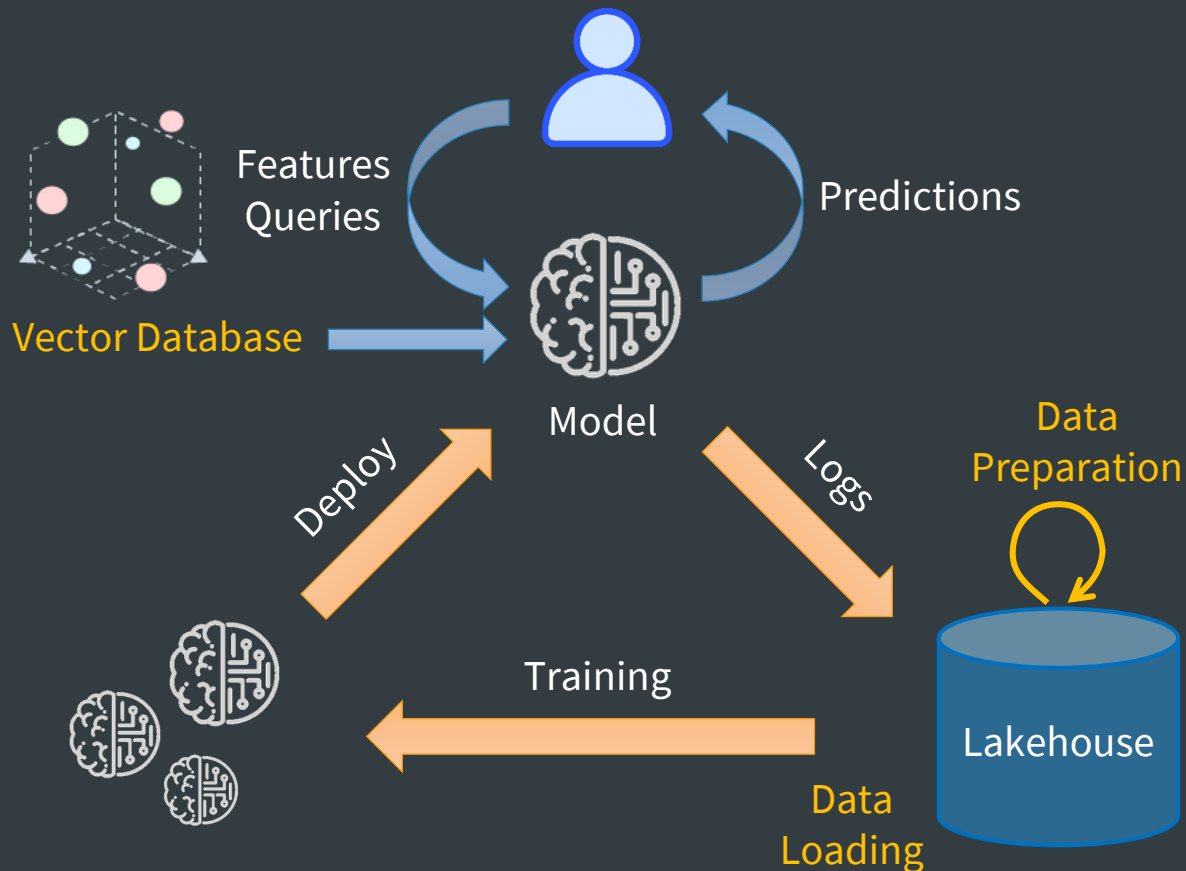
Vector Database
- Vector Indexing and Searching
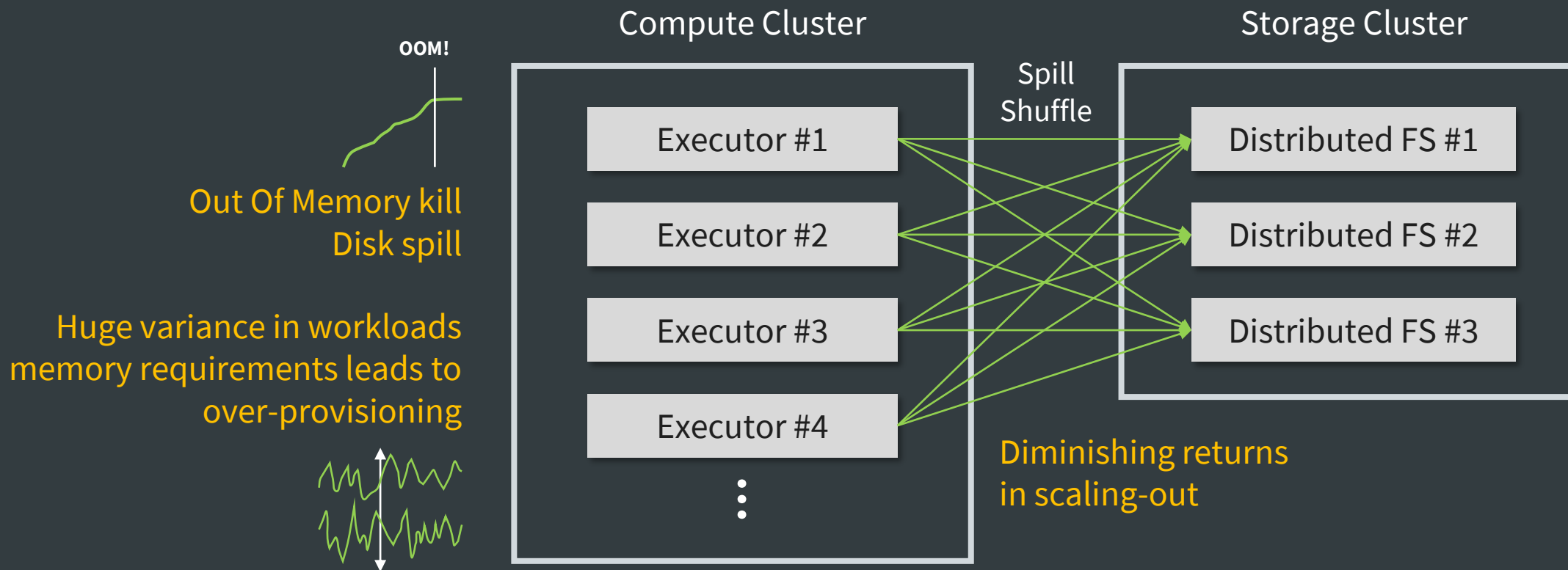- RAG Pipelines

Data Preparation
- Preparing tables
- Embeddings, long sequences of features
- Batch ETL pipelines similar to traditional data transformation

Data Loading
- Feeding GPUs
- Latency sensitive
- Order sensitive (Training)
  → random/chronological/...

Source: Veloxcon 2025 https://youtu.be/nk9abk0evLk

CHALLENGES IN DATA ANALYTICS @ SCALE

MEMORY INEFFICIENCIES AND DIMINISHING RETURNS IN SCALING CALL FOR A NEW MEMORY SOLUTION

OOM!

Out Of Memory kill
Disk spill

Huge variance in workloads
memory requirements leads to
over-provisioning

Compute Cluster

Executor #1
Executor #2
Executor #3
Executor #4

Spill
Shuffle

Storage Cluster

Distributed FS #1
Distributed FS #2
Distributed FS #3

Diminishing returns
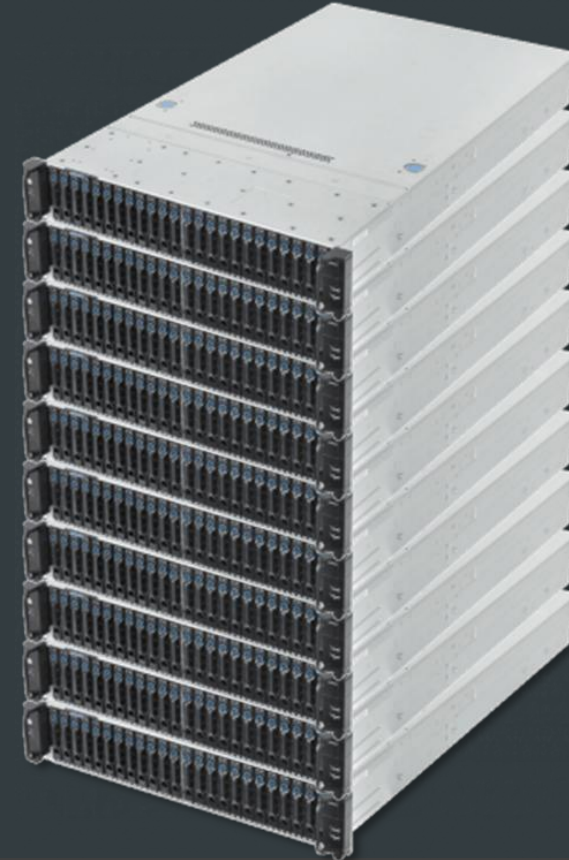in scaling-out

SOLUTION?

WHAT IF A SINGLE DENSE MEMORY AND COMPUTE NODE COULD REPLACE A CLUSTER OF 10?

Dense Memory and Compute Node
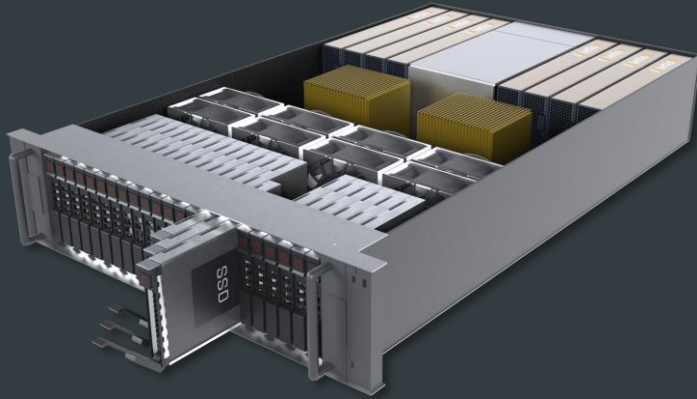
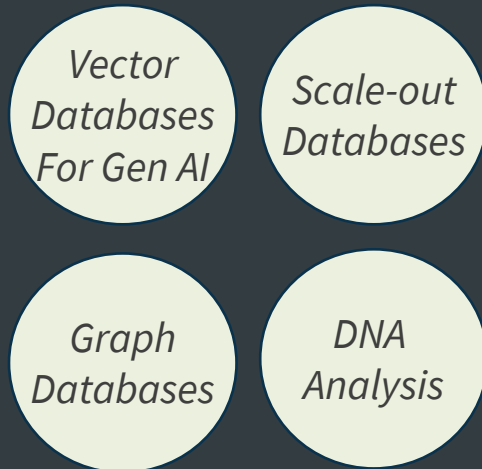Cluster of Nodes

=

DATA PROCESSING QUADRANT

EMBARRESSINGLY PARALLEL, LOW COMPUTE INTENSITY, MEMORY BOUNDED
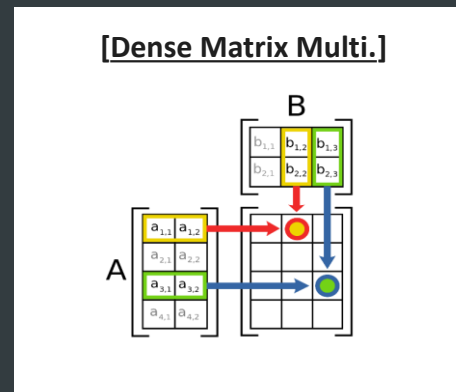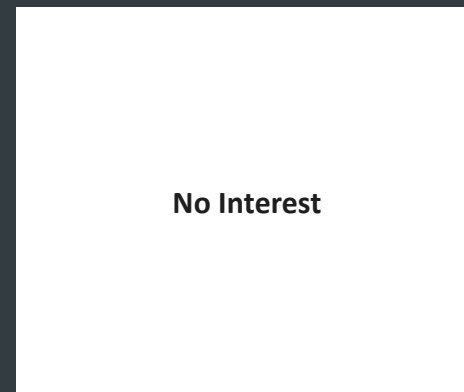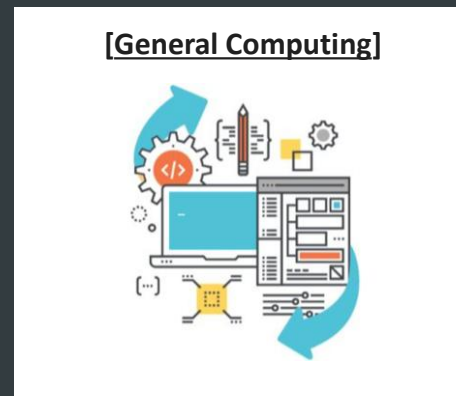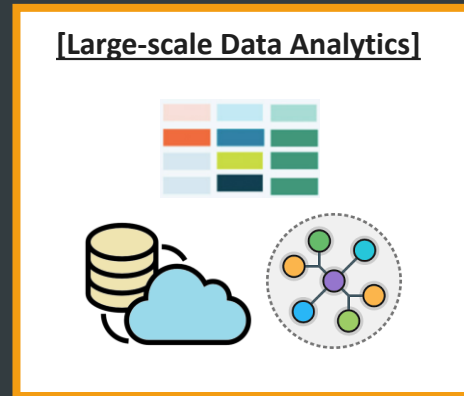
Disaggregation, Computational Memory

DDR

CXL MEMORY

Big Data Processing

Vector Databases For Gen AI

Scale-out Databases

Graph Databases

DNA Analysis

Computational Intensity Per Memory Access

Low → *High*

Bandwidth/Capacity Expansion

DDR

LPDDR

CXL MEMORY

General Computing

[Large-scale Data Analytics]

[General Computing]

No Interest

[Dense Matrix Multi.]

Operation Diversity Per Memory Access

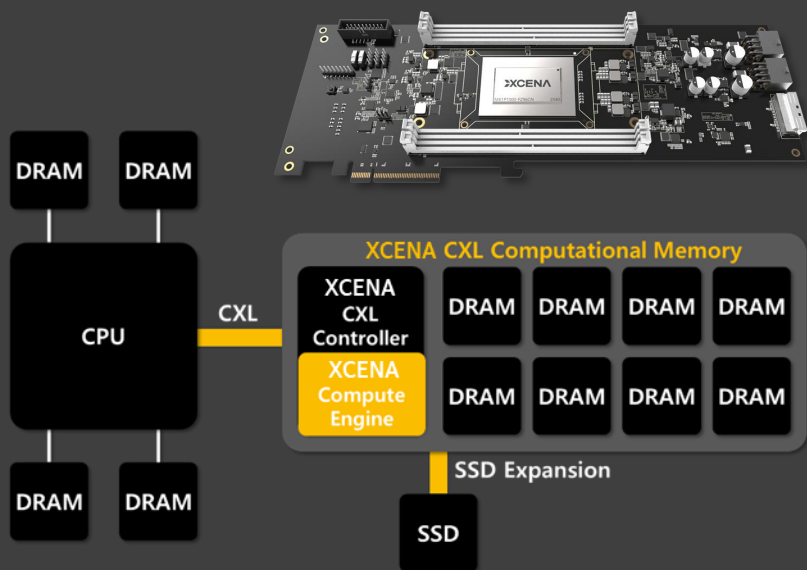**HBM, Explicit Memory**

HBM

GDDR

Low
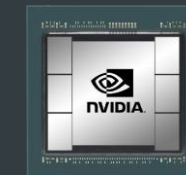
Artificial Intelligence

XCENA

CXL COMPUTATIONAL MEMORY

XCENA

BEYOND JUST ANOTHER CXL MEMORY EXPANDER

DATA PROCESSING LIBRARY

DATA ANALYTICS ACCELERATION

TCO SAVINGS
SEAMLESS INTEGRATION

COMPUTING HW-SW SOLUTION

NEAR-MEMORY PROCESSING

LESS DATA MOVEMENT
LOWER CPU UTIL
LOWER POWER
BETTER PERFORMANCE

CXL MEMORY
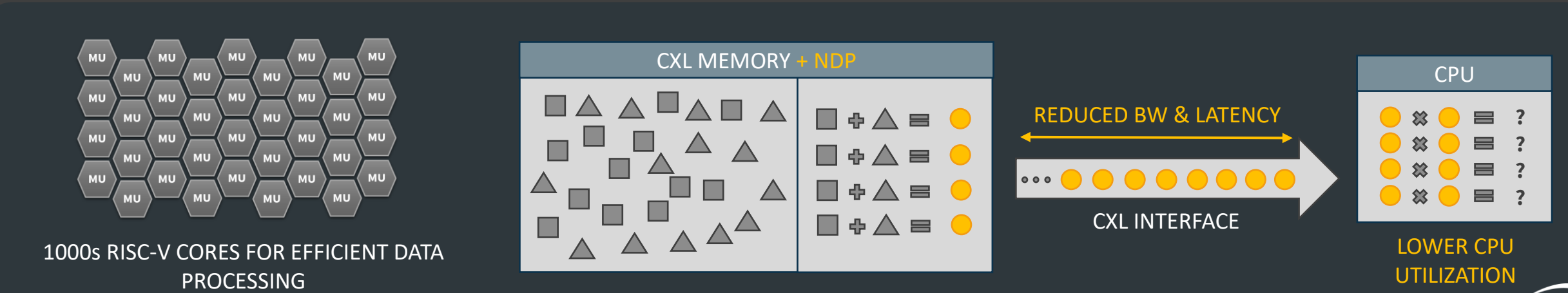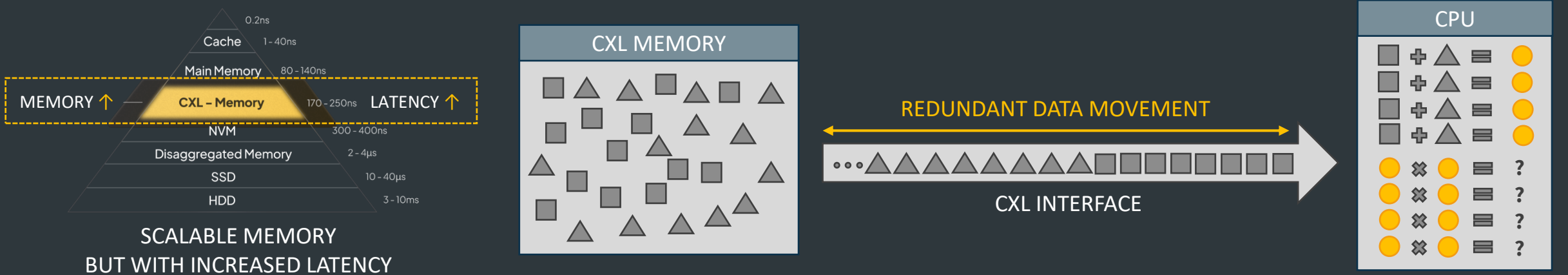
HIGH PERFORMANCE CXL DRAM EXPANSION

INFINITE MEMORY

PB-SCALE MEMORY WITH SSD EXPANSION

MORE MEMORY
CHEAPER MEMORY

GENERAL CXL PLAYERS

FMS

NEAR-MEMORY PROCESSING

More Capacity, Extra Bandwidth, +Latency
Reducing data movement by Near-Memory Processing

SCALABLE MEMORY
BUT WITH INCREASED LATENCY

CXL MEMORY

REDUNDANT DATA MOVEMENT

CXL INTERFACE

CPU

1000s RISC-V CORES FOR EFFICIENT DATA
PROCESSING

CXL MEMORY + NDP

REDUCED BW & LATENCY

CXL INTERFACE
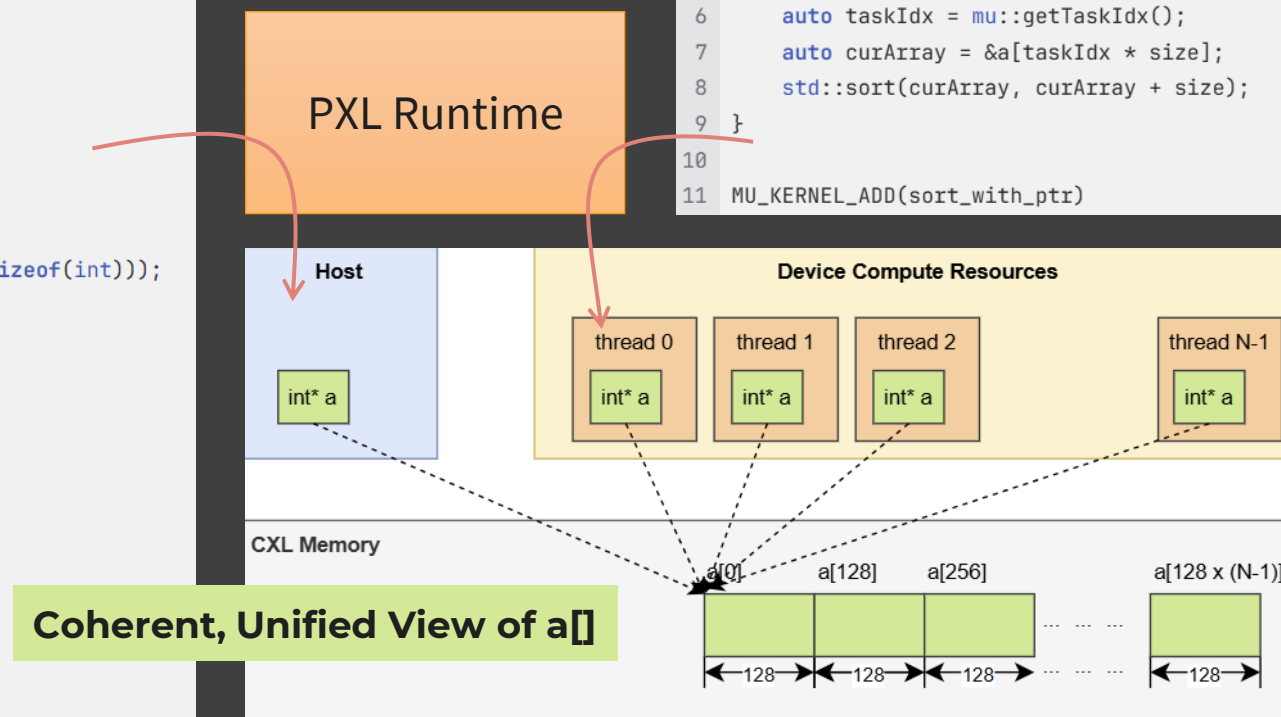
CPU

LOWER CPU
UTILIZATION

PARALLEL PROGRAMMING

Parallel Xceleration Library (PXL)
Map API encapsulates all the complexities of device management and provides a FRAMEWORK of thought for developers.

```cpp
1   #include "pxl/pxl.hpp"
2
3   // setup the device
4   const char* filename = "mu_kernel/mu_kernel.mubin";
5   const char* muFuncName = "sort_with_ptr";
6   auto context = pxl::runtime::createContext(0);
7   auto module = pxl::createModule(filename);
8   auto job = context->createJob();
9   job->load(module);
10  auto muFunction = module->createFunction(muFuncName);
11
12  // allocate cxl memory
13  int* a = reinterpret_cast<int*>(context->memAlloc(N * 128 * sizeof(int)));
14  // setup data
15  for (size_t i = 0; i < N; i++)
16  {
17      for (size_t j = 0; j < 128; j++)
18      {
19          a[i * 128 + j] = 128 - j;
20      }
21  }
22
23  auto map = job->buildMap(muFunction, N);
24  auto ret = map->execute(a, 128);
25  map->synchronize();
```

```cpp
1   #include <algorithm>
2   #include "mu/mu.hpp"
3
4   void sort_with_ptr(int* a, int size)
5   {
6       auto taskIdx = mu::getTaskIdx();
7       auto curArray = &a[taskIdx * size];
8       std::sort(curArray, curArray + size);
9   }
10
11  MU_KERNEL_ADD(sort_with_ptr)
```

PXL Runtime

Host
int* a

Device Compute Resources
thread 0 — int* a
thread 1 — int* a
thread 2 — int* a
thread N-1 — int* a

CXL Memory
a[0]   a[128]   a[256]   a[128 x (N-1)]
128   128   128   128

Coherent, Unified View of a[]

FMS

DATA SPECIFIC LIBRARY

Spark SQL query is compiled and planned by Spark, translated into Velox plans by Gluten, and executed by XFLARE on MX1.

```
1  spark.read.parquet("persons.parquet").createOrReplaceTempView("persons")
2  spark.sql("""
3      SELECT   *
4      FROM     persons
5      ORDER BY age DESC
6  """).show()
```
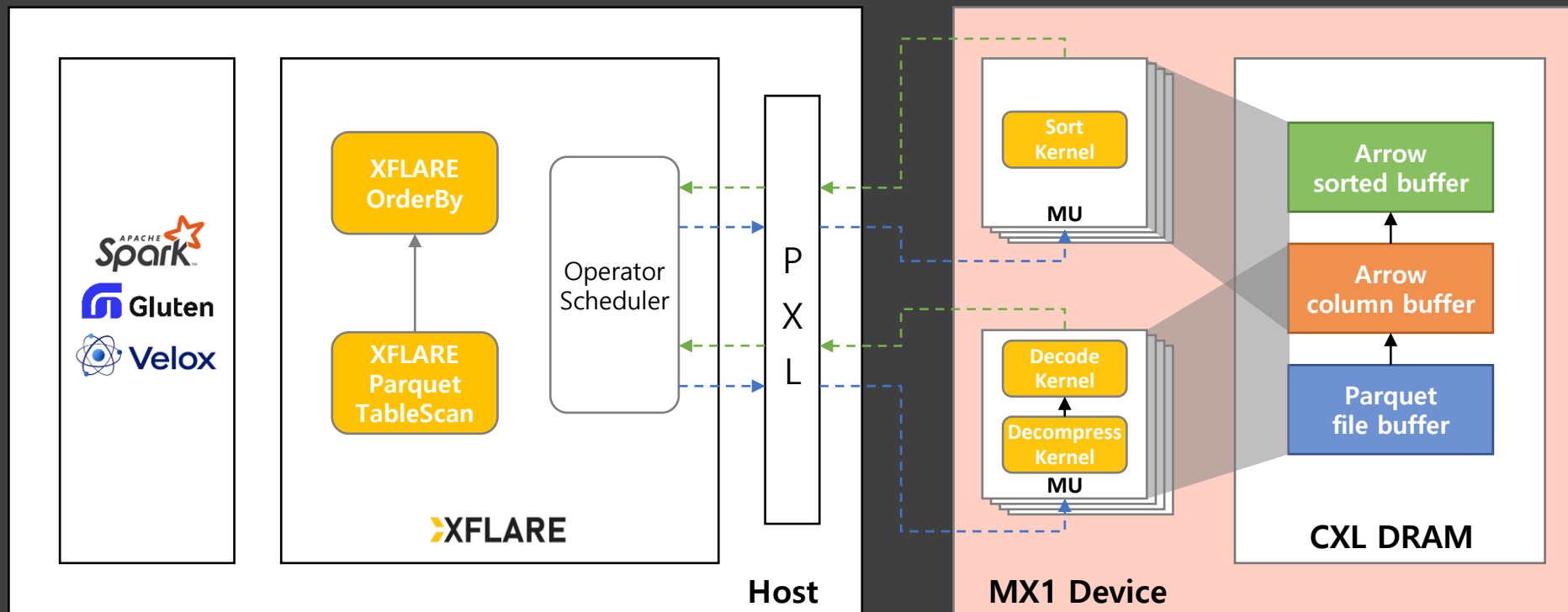
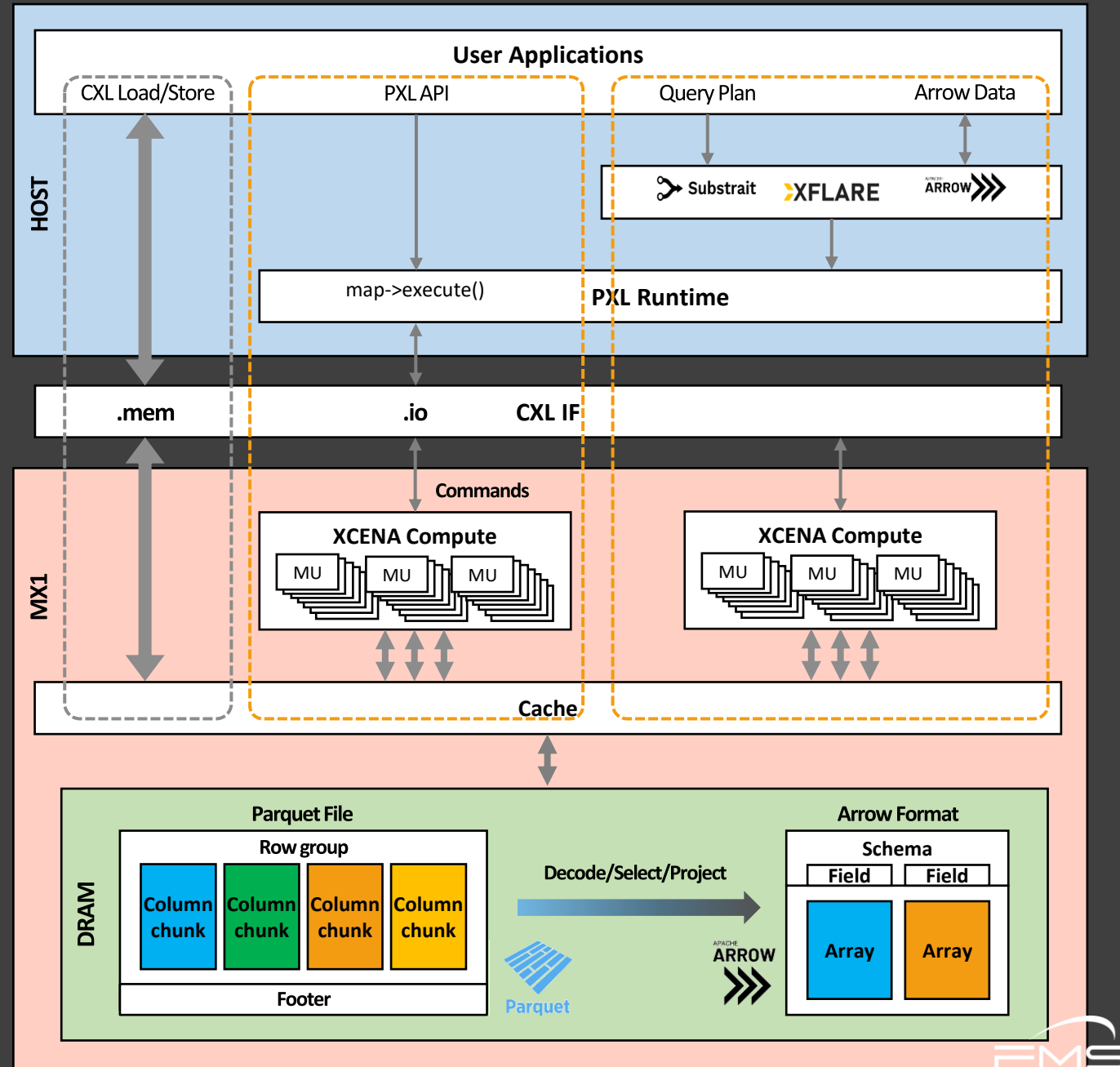# DON'T MOVE DATA, MOVE COMPUTATIONS!

CXL LOAD/STORE
Use CXL memory as just another part of host memory.

PXL API
PXL allows you to manage resources and execute user kernels in parallel.

XFLARE
Common formats (Substrait, Arrow) and high-level APIs simplify query offloading.

XCENA.COM/SDK

QEMU CXL 3.0 Host + Computational Memory
XCENA SDK & Runtime Library
Example Codes and Tutorials
→ ALL IN A DOCKER IMAGE

XCENA

THANK YOU

HARRY KIM, CPO

harry.kim@xcena.com

http://xcena.com

https://www.linkedin.com/company/xcena/

FMS