

Farview

Disaggregated Memory with Operator Off-loading for Database Engines

Dario Korolija*, Dimitrios Koutsoukos*, Kimberly Keeton[±],
Konstantin Taranov*, Dejan Milojičić[‡], Gustavo Alonso*

* Systems Group, Dept. of Computer Science, ETH Zurich

‡ Hewlett Packard Enterprise

± Hewlett Packard Enterprise (now at Google)

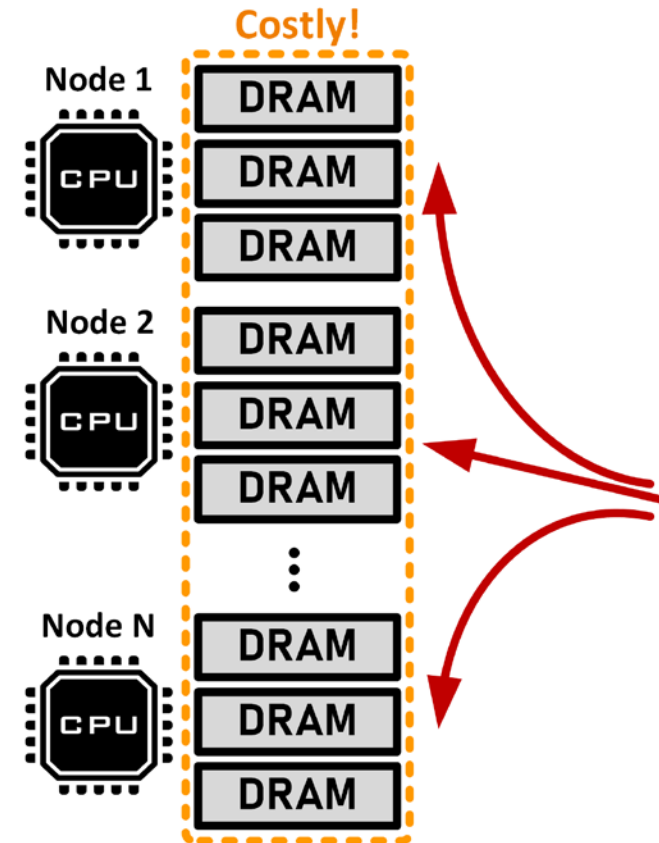
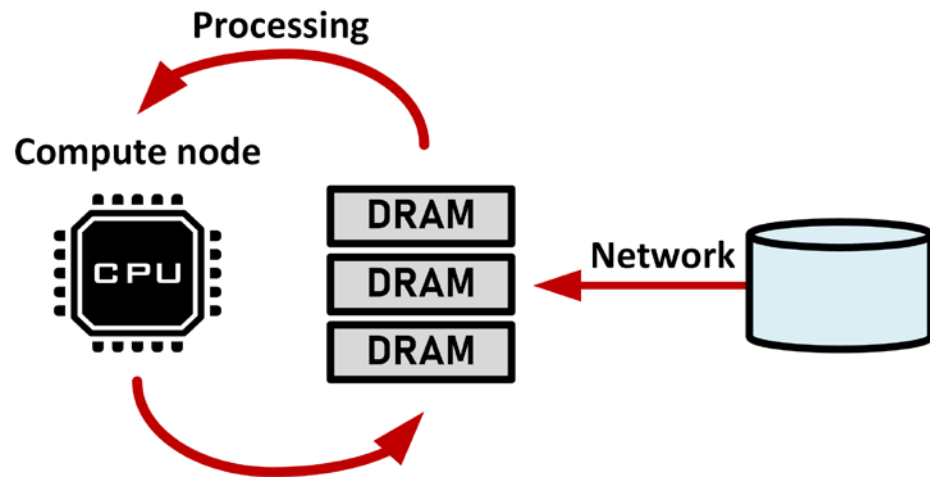


Farview

Motivation

- The I/O overheads one of the main factors in the overall performance
- More and more data kept in local **DRAM** of compute nodes

1. Excessive data movement
2. Memory capacity limitation



Disaggregation of compute, and storage and storage

Network Requirements for Resource Disaggregation

Peter X. Gao UC Berkeley, Alshay Nanyam UC Berkeley, Sagar Karandikar UC Berkeley, Joao Carneiro UC Berkeley, Sangjin Han UC Berkeley, Rachit Agrawal Cornell University, Sylvia Ratnasamy UC Berkeley, Scott Shenker UC Berkeley/UCS

Abstract
Traditional datacenters are designed as a collection of servers, each of which tightly couples the resources required for computing tasks. Recent industry trends suggest a paradigm shift to a disaggregated datacenter (DDC) architecture containing a pool of resources, each built as a standalone resource blade and interconnected using a network fabric. A key enabling (or blocking) factor for disaggregation will be the network – to support good application-level performance it becomes critical that the network fabric provide low latency communication even under the increased traffic load that disaggregation introduces. In this paper, we use a workload-driven approach to derive the minimum latency and bandwidth requirements that the network in disaggregated datacenters must provide to avoid degrading application-level performance and explore the feasibility of meeting these requirements with existing system designs and commodity networking technology.

1 Introduction
Existing datacenters are built using servers, each of which tightly integrates a small amount of the various resources needed for a computing task (CPU, memory, storage). While such server-centric architectures have been the mainstay for decades, recent efforts suggest a forthcoming paradigm shift towards a disaggregated datacenter (DDC), where each resource type is built as a standalone resource “blade” and a network fabric interconnects these resource blades. Examples of this include Facebook’s Disaggregated Rack [8], HP “The Machine” [13], Intel Rack Scale Architecture [19], SeaMicro [24] as well as prototypes from the computer architecture community [3], [6], [5].

These industrial and academic efforts have been driven largely by hardware architects because CPU, memory and storage technologies exhibit significantly different trends in

AsymNVM: An Efficient Framework for Implementing Persistent Data Structures on Asymmetric NVM Architecture

Teng Ma Tsinghua University, Mingxing Zhang Tsinghua University & Sogang University, Kang Chen Tsinghua University, Xuehai Qian University of Southern California, Zhaoyang Song Alibaba, Yongwei Wu Tsinghua University, Xuehai Qian University of Southern California, Los Angeles, CA

Abstract
The byte-addressable non-volatile memory (NVM) is a promising technology since it simultaneously provides DRAM-like performance, disk-like capacity, and persistency. The current NVM deployment with byte-addressability is symmetric, where NVM devices are directly attached to servers. Due to the higher density, NVM provides much larger capacity and should be shared among servers. Unfortunately, in the symmetric setting, the availability of NVM devices is affected by the specific machine it is attached to. High availability can be achieved by replicating data to NVM on a remote machine. However, it requires full replication of data structure in local memory – limiting the size of the working set.

This paper rethinks NVM deployment and makes a case for the asymmetric, byte-addressable non-volatile memory architecture, which decouples servers from persistent data storage. In the proposed AsymNVM architecture, NVM devices (i.e., back-end nodes) can be shared by multiple servers (i.e., front-end nodes) and provide recoverable persistent data structures. The asymmetric architecture, which follows the industry trend of resource disaggregation, is made possible due to the high performance network (e.g., RDMA). At the same time, AsymNVM leads to a number of key problem such as, still relatively long network latency, persistency bottleneck, and simple interface of the back-end NVM nodes.

1 Introduction
Emerging non-volatile memory (NVM) is blurring the line between memory and storage. These kinds of memories, such as Intel Optane DC persistent memory [13], phase change memory (PCM) [5], [4], [10], spin-transfer torque magnetic memory (STTM) [1], and memristor are byte-addressable,

HyperLoop: Group-Based NIC-Offloading to Accelerate Replicated Transactions in Multi-Tenant Storage Systems

Daehyeok Kim^{1*}, Amirsaman Memaripour^{1*}, Anirudh Badam³, Yibo Zhu¹, Hongqiang Harry Liu^{3†}, Jitu Padhye¹, Shachar Raaijndel², Zhen Swanson^{2†}, Vyas Sekar¹, Srinivasan Seshan¹

Abstract
Storage systems in data centers are an important component of large-scale online services. They typically perform replicated transactional operations for high data availability and integrity. Today, however, such operations suffer from high tail latency even with recent kernel bypass and storage optimizations, and thus affect the predictability of end-to-end performance of these services. We observe that the root cause of the problem is the involvement of the CPU, a precious commodity in multi-tenant settings, in the critical path of replicated transactions. In this paper, we present HyperLoop, a new framework that removes CPU from the critical path of replicated transactions in storage systems by offloading them to commodity RDMA NICs, with non-volatile memory as the storage medium. To achieve this, we develop new and general NIC offloading primitives that can perform memory operations on all nodes in a replication group while guaranteeing ACID properties without CPU involvement. We demonstrate that popular storage applications can be easily optimized using our primitives. Our evaluation results with microbenchmarks and application benchmarks show that HyperLoop can reduce 99th percentile latency + 800x with close to 0% CPU consumption on replicas.

1 Introduction
Distributed storage systems, Replicated transactions, RDMA, NIC offloading.

ACM Reference Format: Daehyeok Kim, Amirsaman Memaripour, Anirudh Badam, Yibo Zhu, Hongqiang Harry Liu, Jitu Padhye, Shachar Raaijndel, Zhen Swanson, Vyas Sekar, Srinivasan Seshan. 2018. HyperLoop: Group-Based NIC-Offloading to Accelerate Replicated Transactions in Multi-Tenant Storage Systems. In SIGCOMM '18. ACM SIGCOMM 2018 Conference, August 28–29, 2018, Budapest, Hungary. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3255653.3255722>

1 Introduction
Distributed storage systems are an important building block for modern online services. To guarantee data availability and integrity, these systems keep multiple replicas of each data object on different servers [3, 4, 8, 9, 17, 18] and rely on replicated transactional operations to ensure that updates are consistently and atomically performed on all replicas.

Such replicated transactions can incur large and unpredictable latencies, and thus impact the overall performance of storage-intensive applications [32, 57, 58, 75, 86, 92]. Recognizing this problem, both networking and storage communities have proposed a number of solutions to reduce average and tail latencies of these systems.

Networking proposals include kernel bypass techniques, such as RDMA (Remote Direct Memory Access) [64], and user-space networking technologies [26, 90]. Similarly, there have been efforts to integrate non-volatile main memory (NVM) [6, 11], and user-space solid state disks (SSDs) [7, 29, 80] to bypass the OS storage stack to reduce latency.

While optimizations such as kernel bypass do improve the performance for standalone storage services and appliance-like systems where there is only a single storage running in

Can Far Memory Improve Job Throughput?

Emmanuel Amaro UC Berkeley, Christopher Branner-Augustin UC Berkeley, Zhilong Luo UC Berkeley, Marc K. Aguilera VMware Research, Amy Overholt UC Berkeley, Sylvia Ratnasamy UC Berkeley, Aurajit Panda New York University, Scott Shenker UC Berkeley, UCS

Abstract
As memory requirements grow, and advances in memory technology slow, the availability of sufficient main memory is increasingly the bottleneck in large compute clusters. One solution to this is memory disaggregation, where jobs can remotely access memory on other servers, or *far memory*. This paper first presents faster swapping mechanisms and a far memory-aware cluster scheduler that make it possible to support far memory at rack scale. Then, it examines the conditions under which this use of far memory can increase job throughput. We find that while far memory is not a panacea, for memory-intensive workloads it can provide performance improvements on the order of 10% or more even without changing the total amount of memory available.

ACM Reference Format: Emmanuel Amaro, Christopher Branner-Augustin, Zhilong Luo, Amy Overholt, Marc K. Aguilera, Aurajit Panda, Sylvia Ratnasamy, and Scott Shenker. 2020. Can Far Memory Improve Job Throughput?. In *Fifteenth European Conference on Computer Systems (EuroSys '20)*, April 27–28, 2020, Heraklion, Greece. ACM, New York, NY, USA, 16 pages. <https://doi.org/10.1145/3342193.3387322>

1 Introduction
The rising popularity of in-memory workloads such as machine learning applications and key-value stores is causing memory demands in compute clusters to grow rapidly [14]. At the same time, because of the end of Moore’s law, DRAM

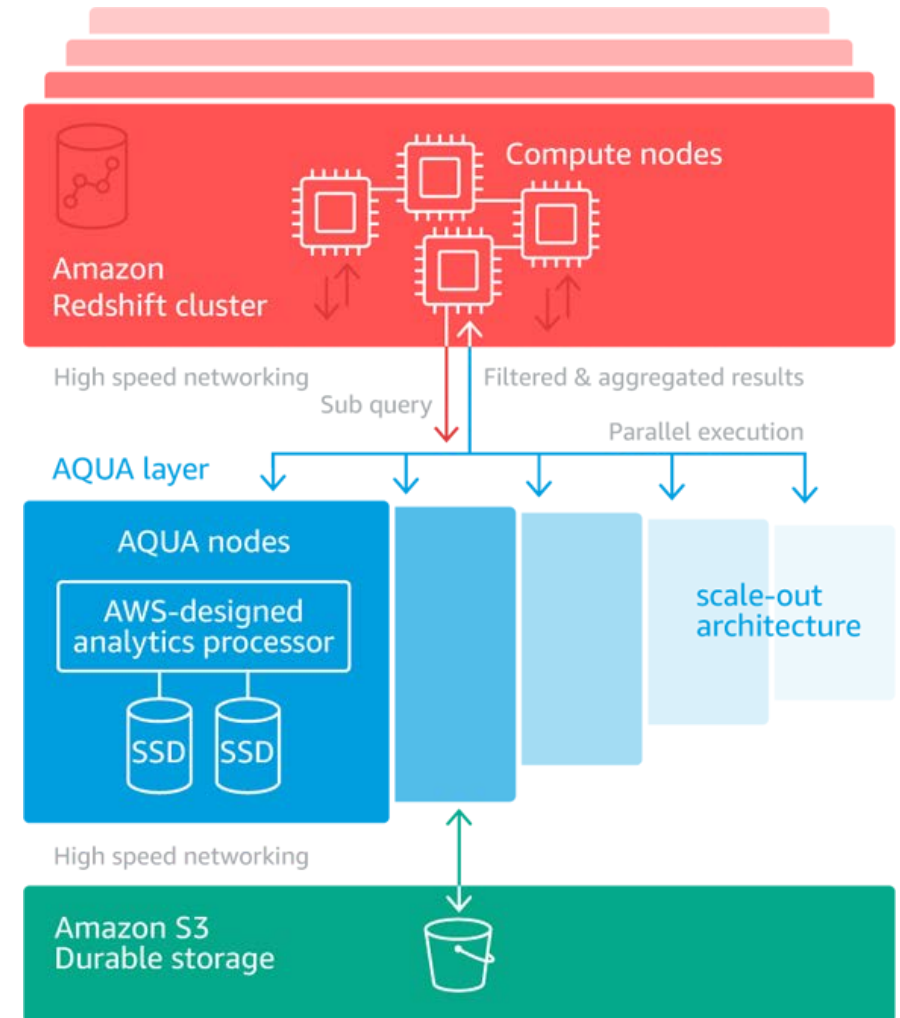
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ASPLoS 20, March 16–20, 2020, Lausanne, Switzerland.
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7204-9, \$15.00.
<https://doi.org/10.1145/337376.337371>

Farview

Motivation

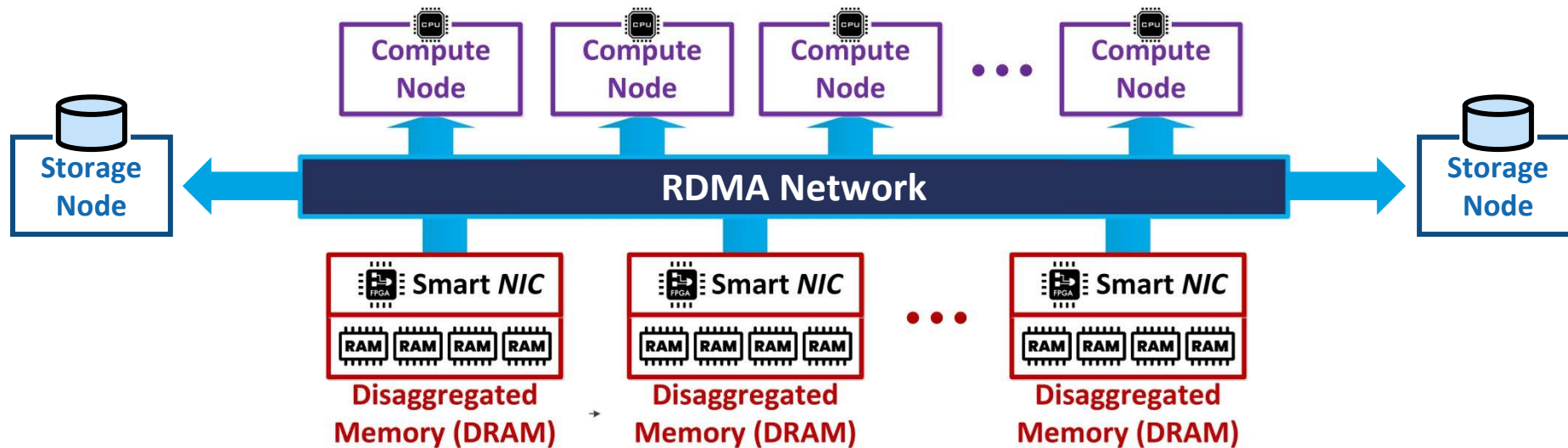
- FPGAs as smart accelerator for disaggregated resources
- Amazon **AQUA**
<https://aws.amazon.com/blogs/aws/new-aqua-advanced-query-accelerator-for-amazon-redshift/>
- Offload of analytic computation to FPGA
- Pushing computation closer to data
- Reduces data movement
- Reduces CPU and network overheads



Farview

Introduction

- **FPGA-based smart NIC** making **DRAM** available as a pool of network attached memory, provisioned on demand, accessible over high performance **RDMA**.
- Capability to perform line-rate data processing with minimal overheads
- **Farview acts as a disaggregated buffer cache with operator pushdown capabilities**



Farview

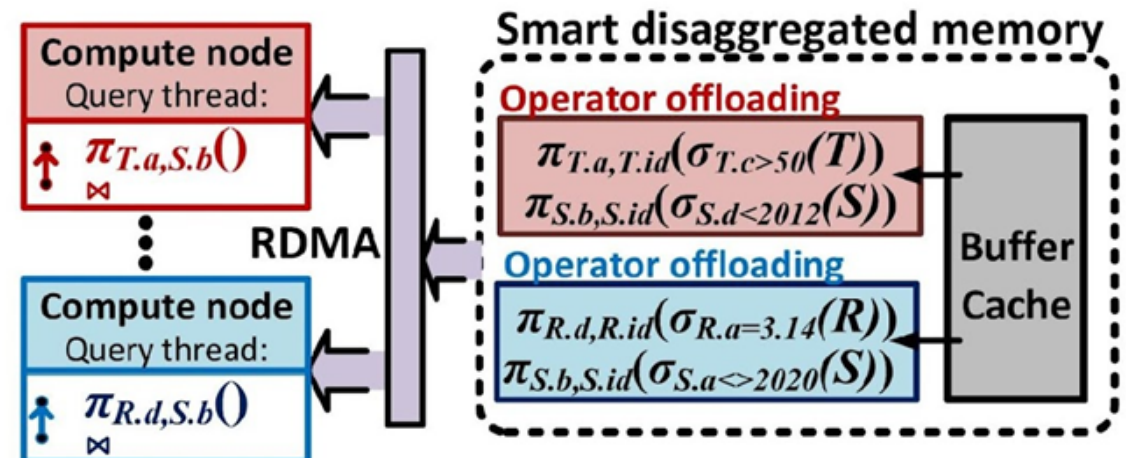
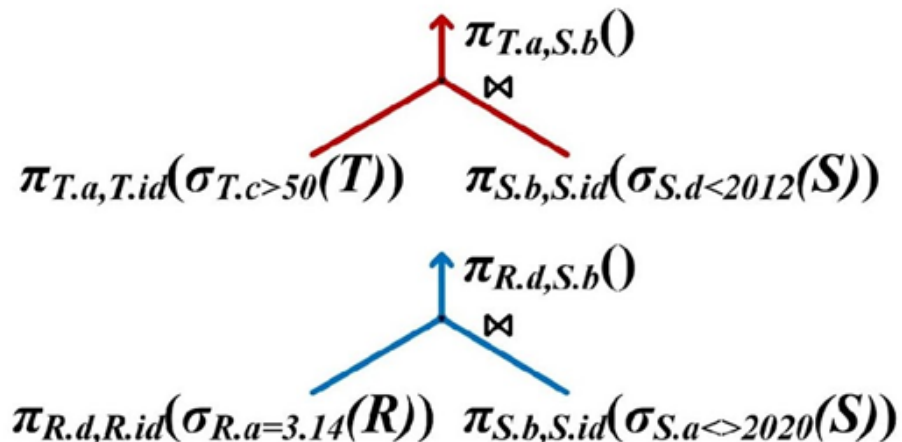
Overview

- **Farview** positioned to address the issues of inefficient data movement and memory capacity limitations
- Consider the following two queries:

```
SELECT T.a, S.b
FROM T, S
WHERE T.id = S.id
AND T.c > 50 AND S.d < 2012;
```

```
SELECT R.d, S.b
FROM R, S
WHERE R.id = S.id
AND R.a = 3.14 AND S.a <> 2020;
```

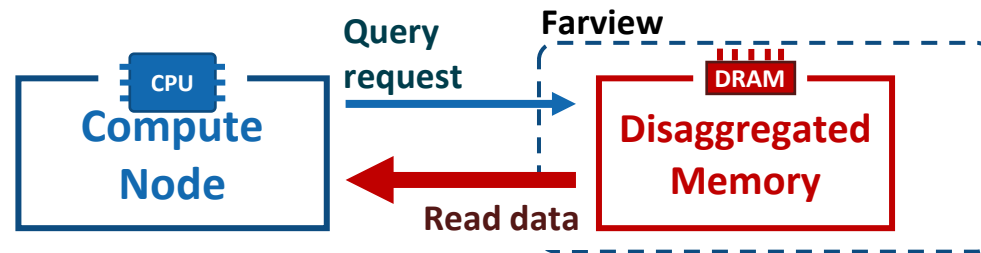
- **Farview** centralizes the buffer cache in disaggregated memory and pushes down operators



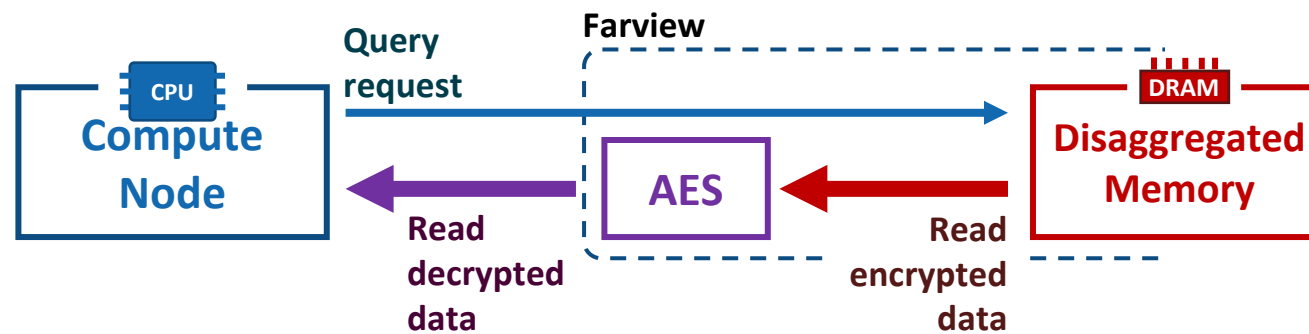
Farview

Example

1. No processing in **Farview**, simple **READ** operation:

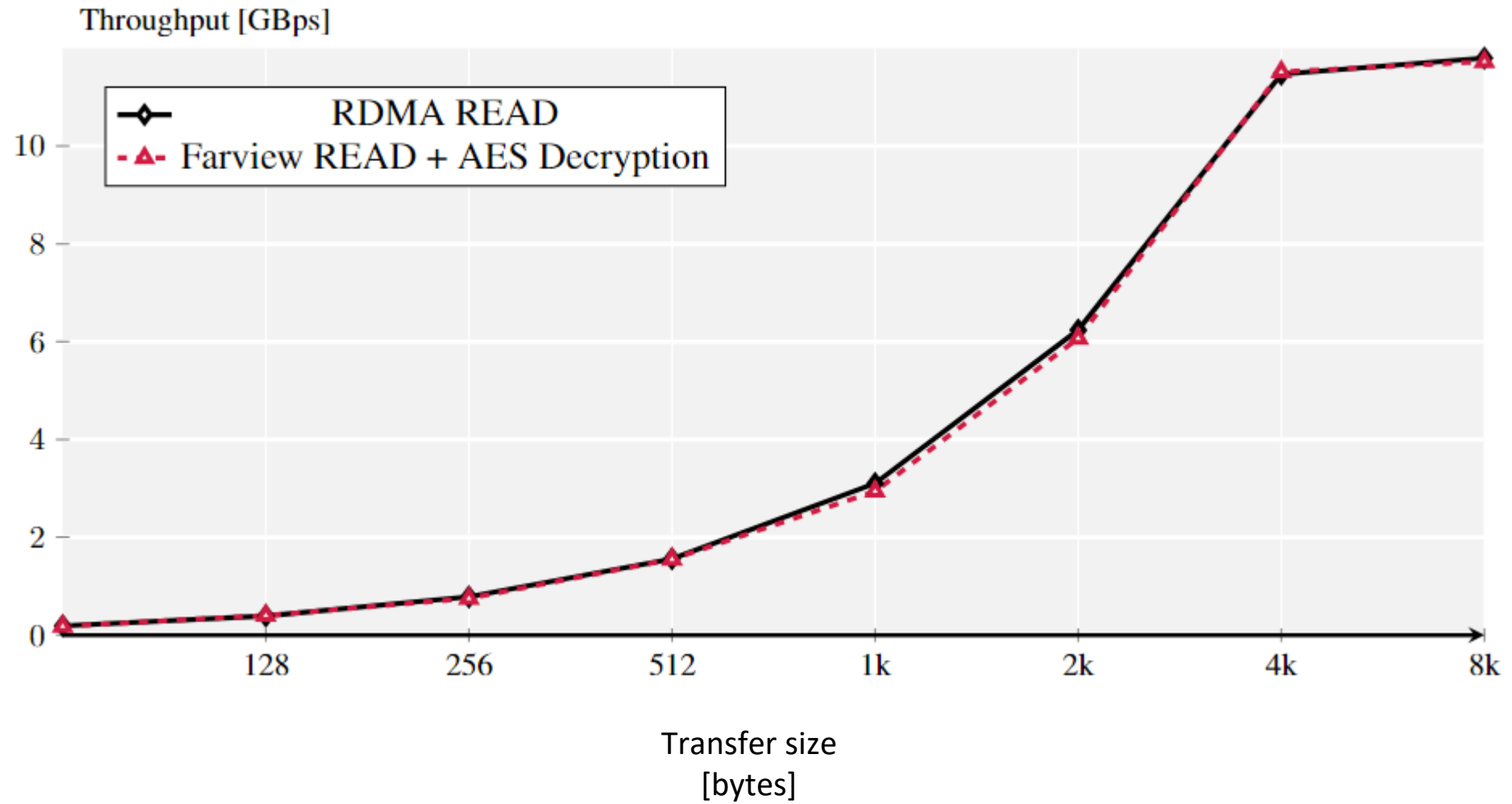


2. AES decryption on the same data as it is being read:



Farview

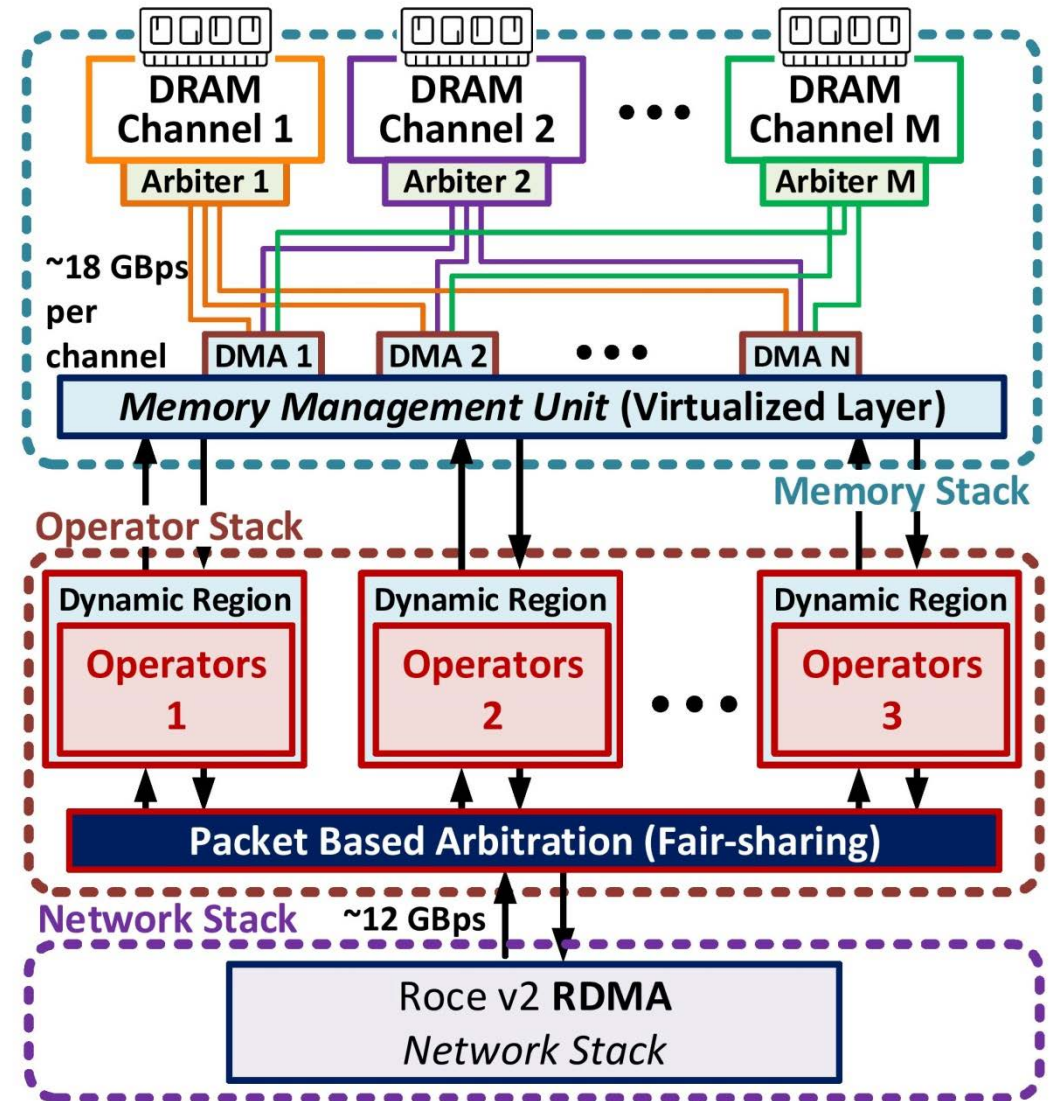
Example



Farview

System Architecture

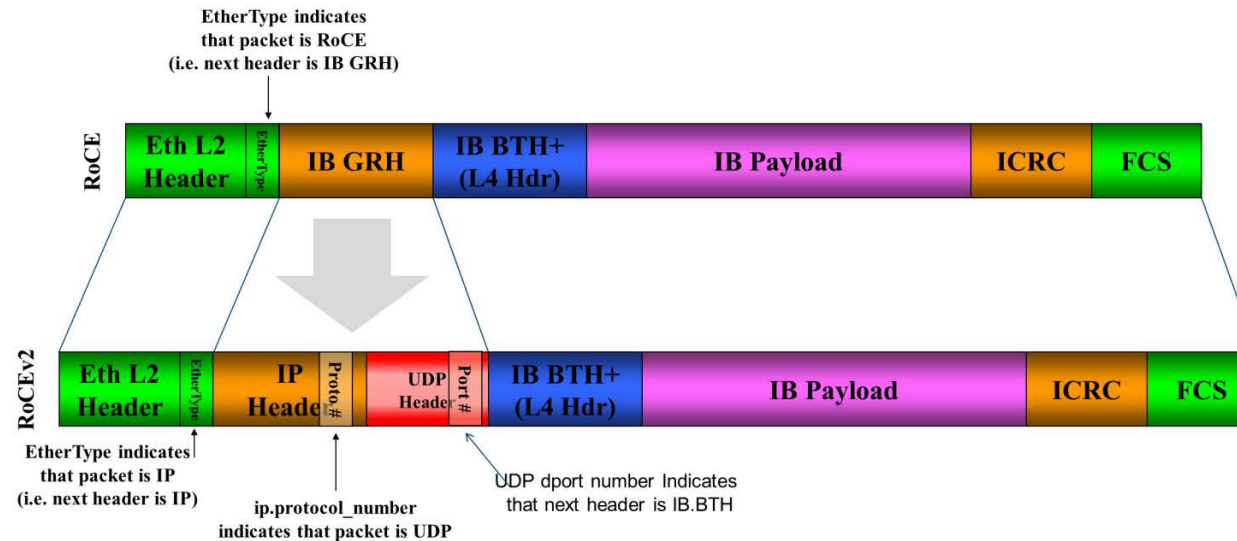
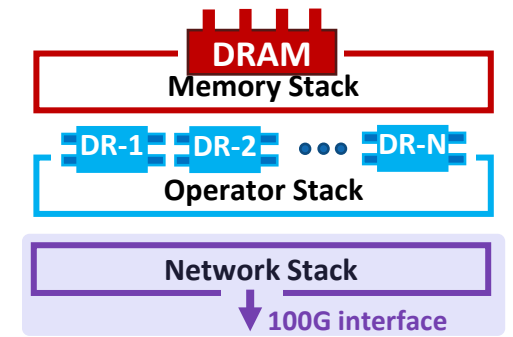
- Several components needed:
 - DRAM (or HBM)
 - Memory controllers
 - Memory management unit
 - Network stack
 - Mechanism to support concurrent accesses to the memory
 - Stream processing capacity
 - Mechanism to swap the operators
- Three distinct modules
 - Operator stack
 - Memory stack
 - Network stack



Farview

Network stack (RDMA)

- Open source RDMA stack (UC, RC)^[1]
- RDMA over Converged Ethernet (RoCE v2)
- Implemented on top of UDP/IPv4/IPv6 (far lower overhead than iWARP)
- InfiniBand (IB) transport packets over Ethernet (READ, WRITE, SEND)



<https://docs.nvidia.com/networking/display/WINOFv55053000/RoCEv2>

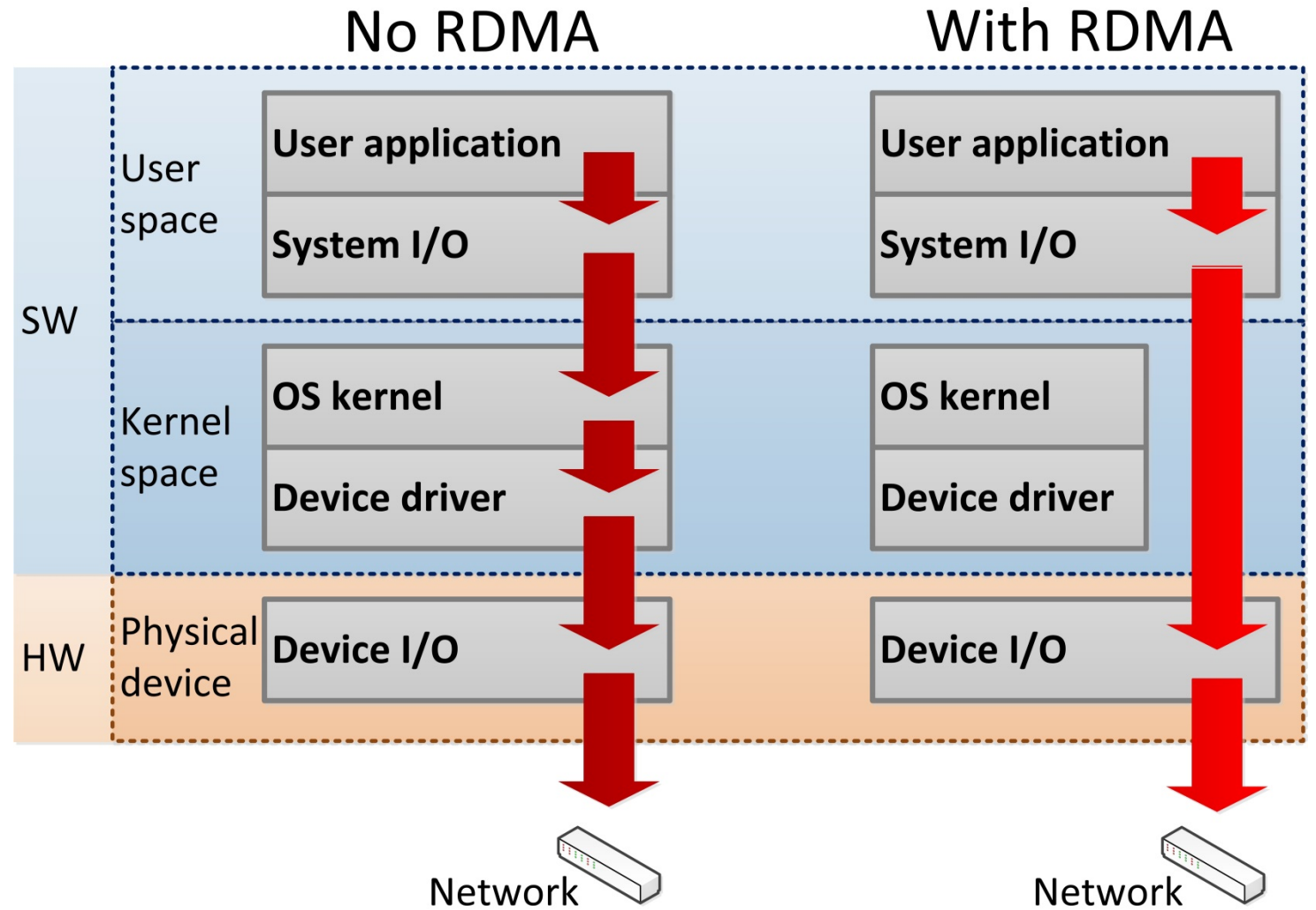
[1] **StRoM**: Smart Remote Memory, EuroSys '20

David Sidler, Zeke Wang, Monica Chiosa, Amit Kulkarni, Gustavo Alonso

Farview

Why RDMA?

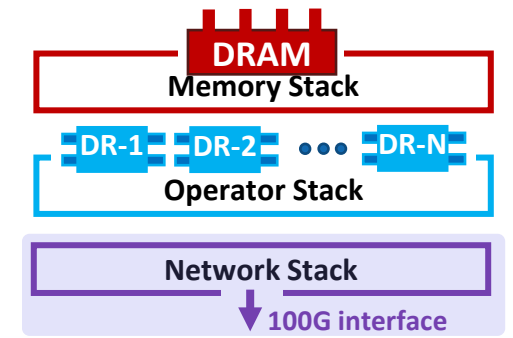
- Bypasses kernel space
- Zero-copy data movement
- Cheap pipelined processing (directly on the NIC)



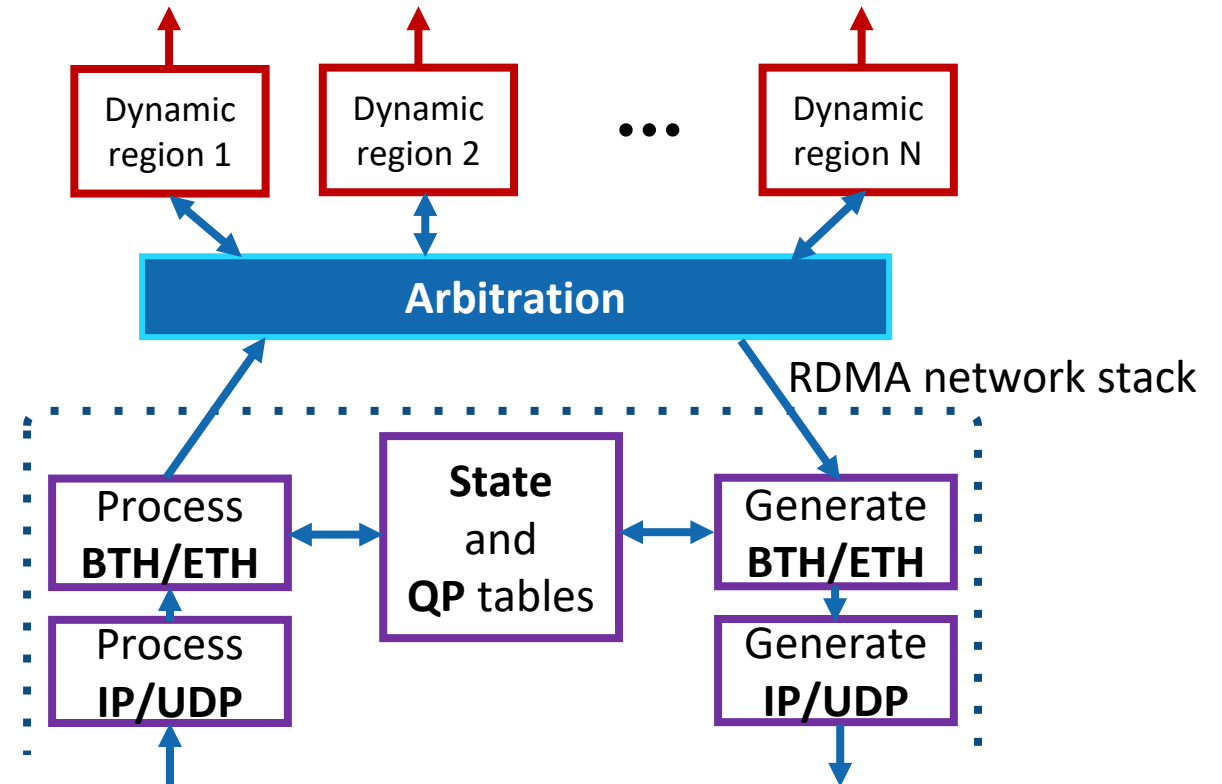
Farview

Network stack

- Manages all external connections and requests for all concurrent accesses
 - Supports **RoCE v2** at **100Gbps**
 - Open source network stack^[1]
- Two-sided verbs used for invocation of queries by clients
- Comparable latencies to traditional one-sided RDMA verbs



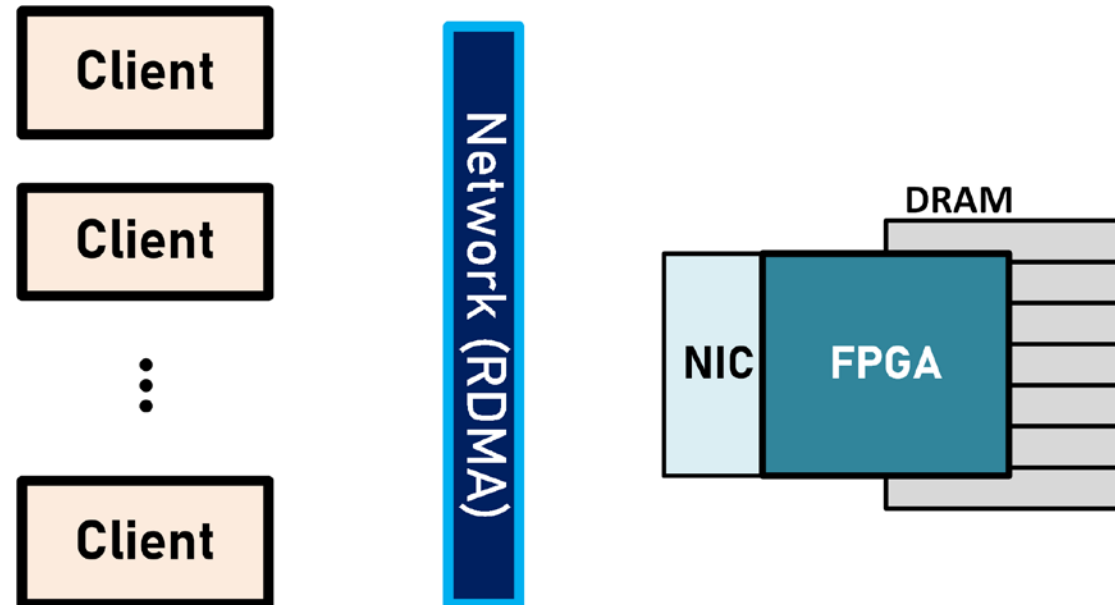
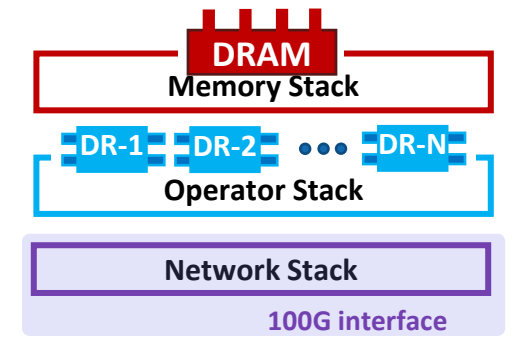
Network stack architecture:



Farview

RDMA abstractions

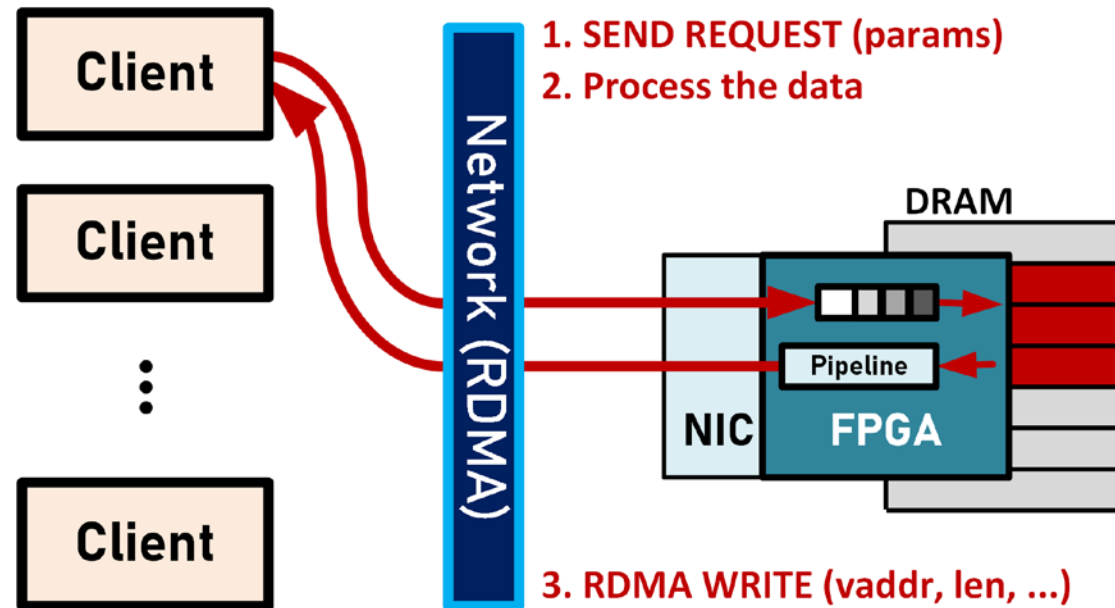
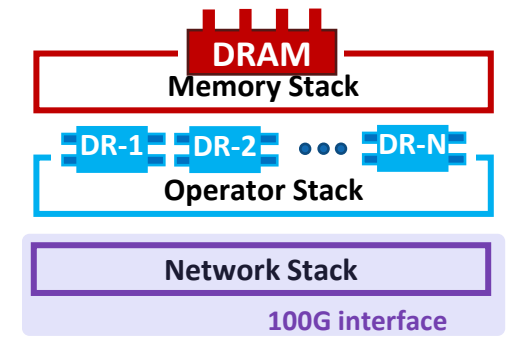
- Remote memory abstraction (one sided verbs)
- Are current RDMA abstractions accurate for modern heterogeneous hardware?
- **Accelerator on the path, no longer just a memory access ...**



Farview

RDMA abstractions

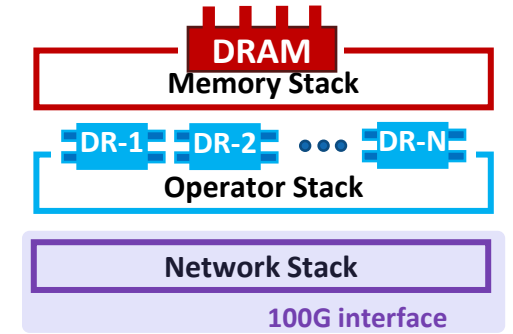
- Use existing IB verbs
- Performance of one-sided RDMA verbs (no overheads)
- Pass generic parameters (RPC)
- Use **SEND** (two-sided) + **WRITE** (one-sided)



Farview

RDMA abstractions

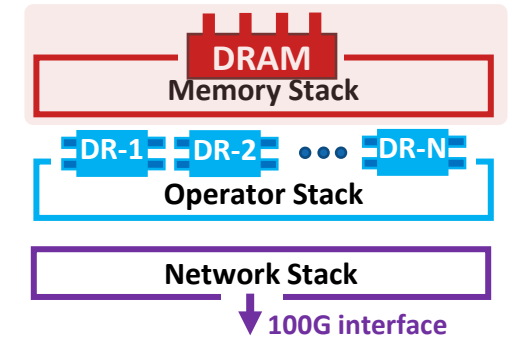
- **SEND** (two-sided) + **WRITE** (one-sided)
- **Pros:**
 - Performance, latency comparable to traditional one-sided RDMA verbs
 - Generic enough (can pass as many parameters as needed)
- **Cons:**
 - Buffer information needs to be passed (could be evaded by extending the RDMA REQUEST verb instead)
 - Not a proper abstraction for what we are doing
- **A lightweight reliable layer on top of UDP?**



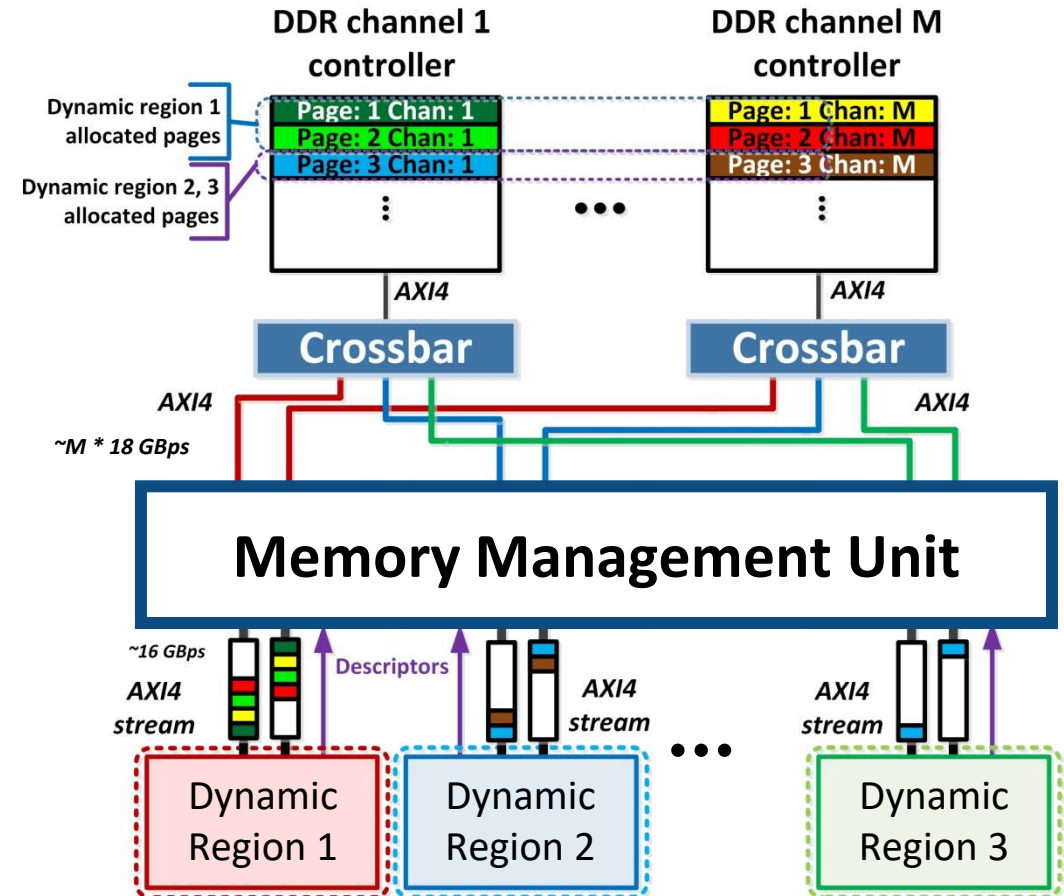
Farview

Memory stack

- Memory buffer pool
- Memory organized into multiple channels (interface same for DRAM and HBM)
- Striping abstraction to optimize the bandwidth
- Can process data at higher rates than the available network bandwidth
- No host connections (PCIe ...) overheads or bandwidth bottlenecks



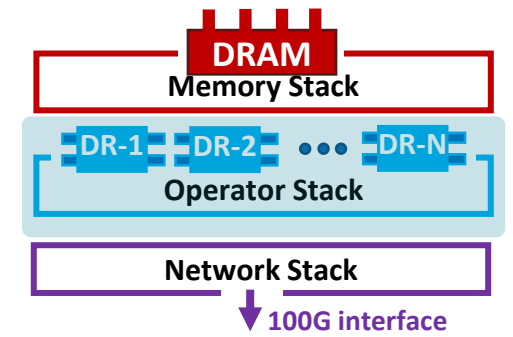
Memory stack architecture:



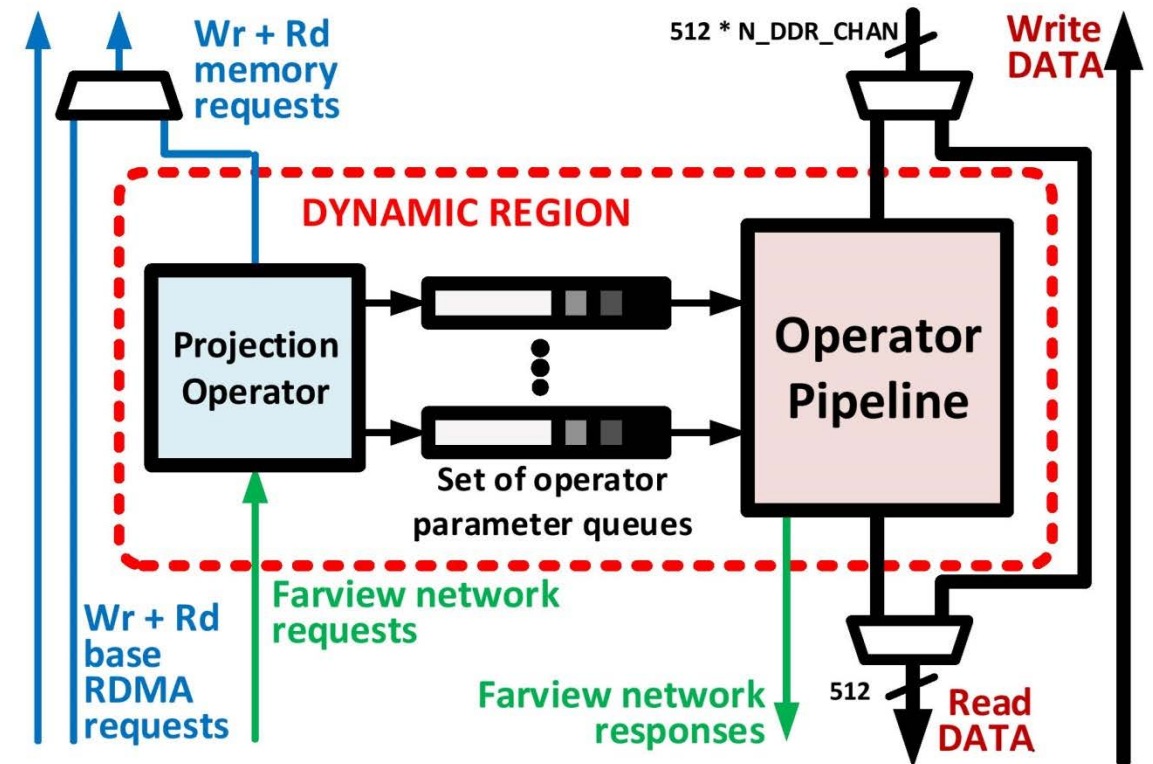
Farview

Operator stack

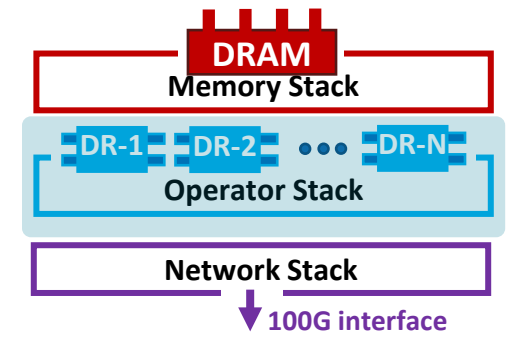
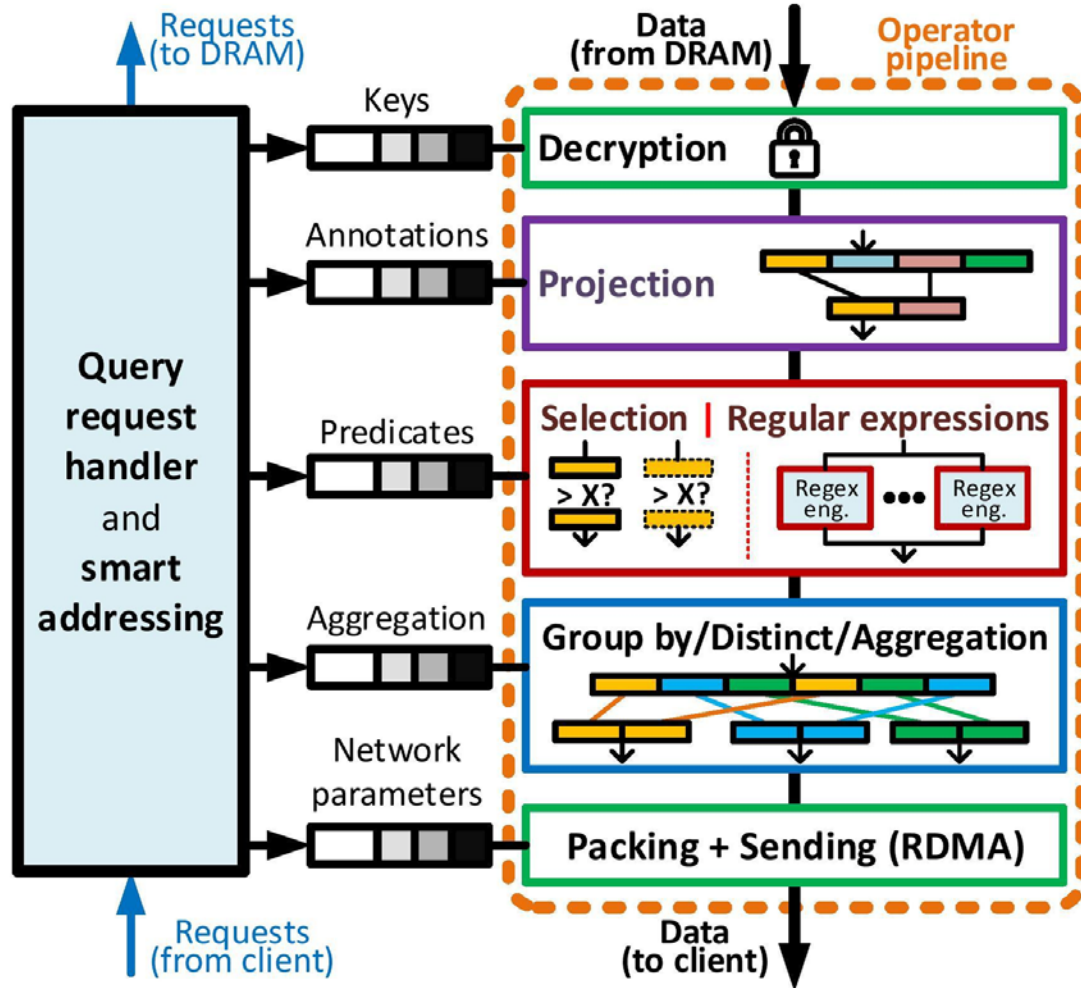
- **Operator stack** split into multiple isolated dynamic regions that operate concurrently (multiple clients)
- **Operator pipeline** can execute a range of queries
- **Operator pipelines** are swappable during runtime



A single dynamic region and interfaces:



Farview Operators

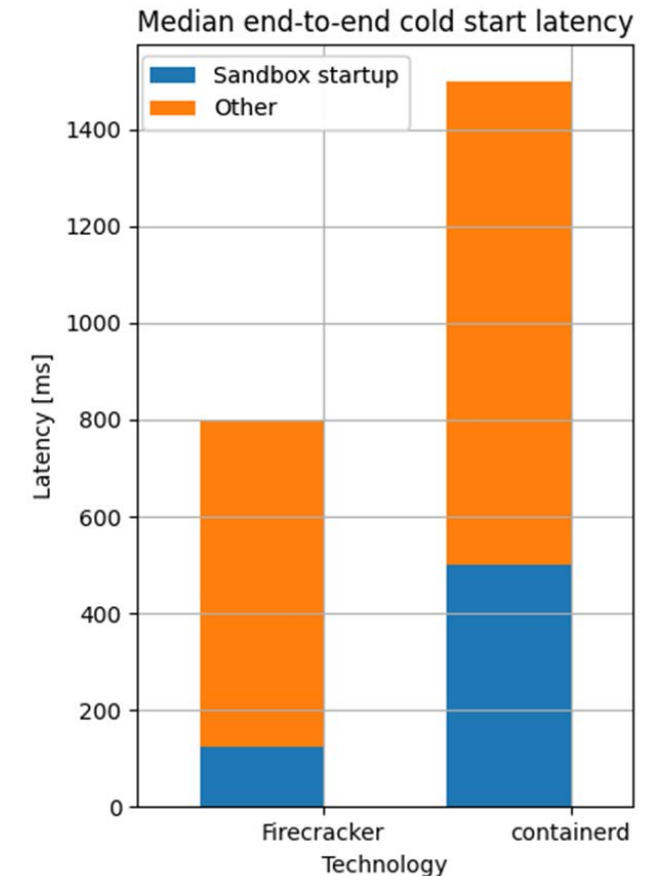
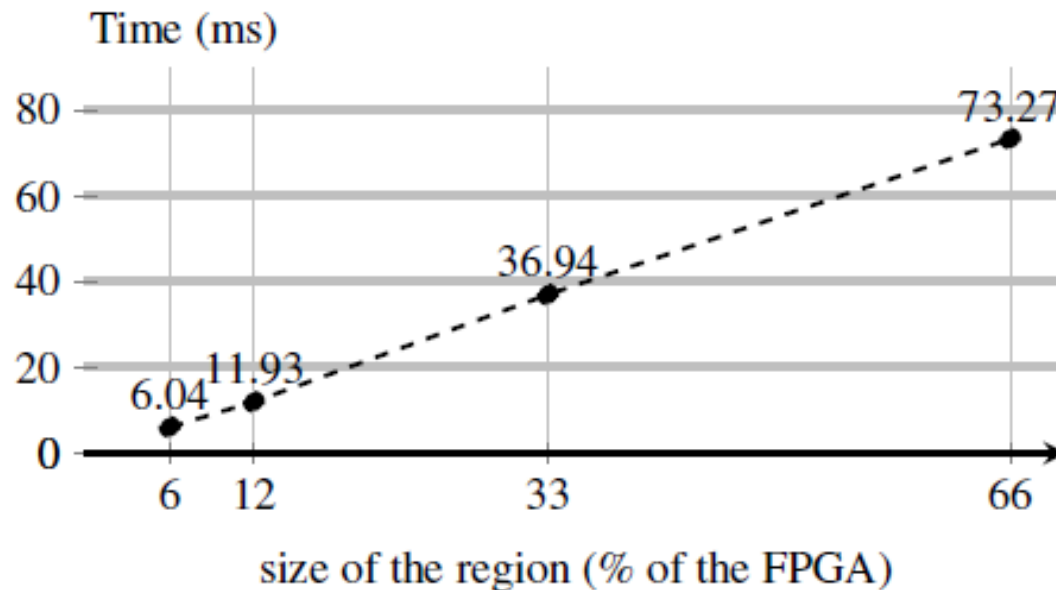
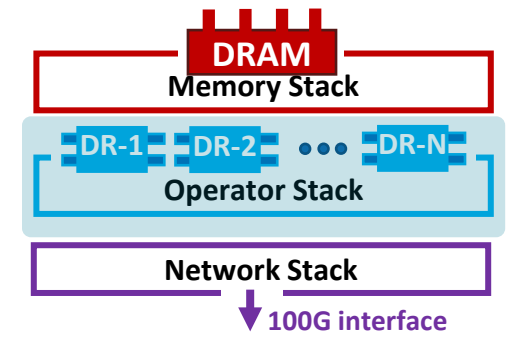


- An example of an *operator pipeline*
- Farview currently supports a variety of operators:
 - **Projection operators** (smart addressing, projection)
 - **Selection operators** (selection, regular expression matching, vectorized selection)
 - **Grouping operators** (distinct, group by and aggregation)
 - **System operators** (encryption/decryption, parsing, packing)
- Row store / column store
- Easily extendable

Farview

Operator Pipeline Swap

- Operator pipelines can be swapped during *runtime*
- Gives **Farview** a much needed flexibility in comparison to traditional accelerators
- Swap time in the order of milliseconds
- Could have applications in microservices domain



Farview

Programmatic Interface

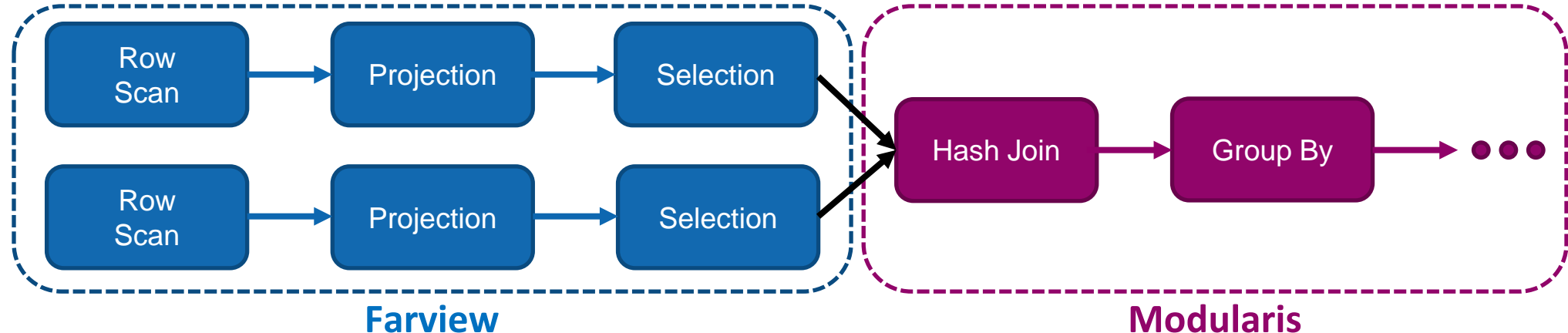
- High level data API covering both the
 - **critical path operations** and
 - **connection management operations**
- API written in C++
- Intended to be used by Farview query compiler.

- `bool` openConnection(Qpair *qp, Fview *node);
- `bool` loadPipeline(Qpair *qp, `int32_t` opid);
- `void` tableRead(Qpair *qp, Ftable *ft);
- `void` tableWrite(Qpair *qp, Ftable *ft);
- ...
- `void` farView(Qpair *qp, Ftable *ft, `uint64_t` *params);
- `void` select(Qpair *qp, Ftable *ft, `uint64_t` *proj_flags, `uint64_t` *sel_flags, `float` predicate) {
 ...
 farView(qp, ft, params);
}

Farview

Frontend - database engine

- How to interact with Farview from client on a higher level?
- **Modularis**^[2] is a distributed query processing system supporting relational queries with different backends (including RDMA)
- Offloading of certain operators within Modularis (Projections, Selections, Aggregations) to Farview



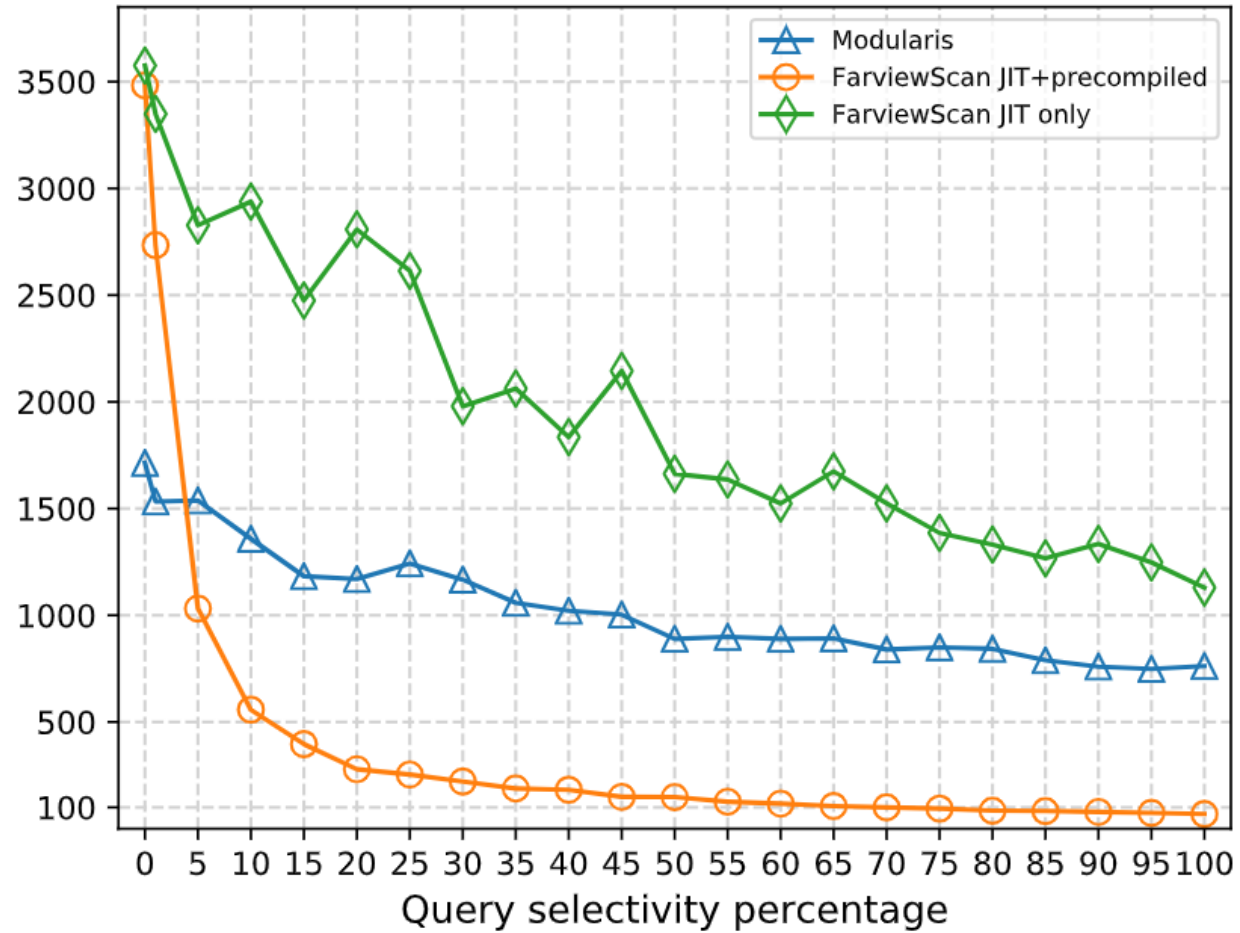
[2] **Modularis**: Modular Relational Analytics over Heterogeneous Distributed Platforms, VLDB '20
Dimitrios Koutsoukos, Ingo Müller, Renato Marroquín, Ana Klimovic, Gustavo Alonso

Farview

Frontend - database engine

Queries per second

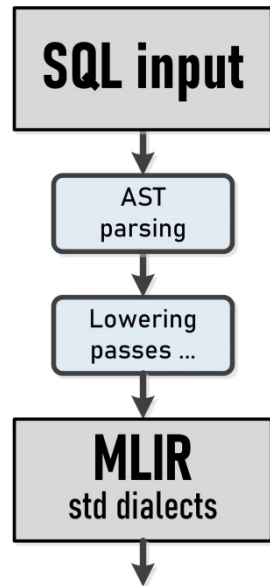
Bandwidth [Qps] for queries of varying selectivity over a 2MB Table.



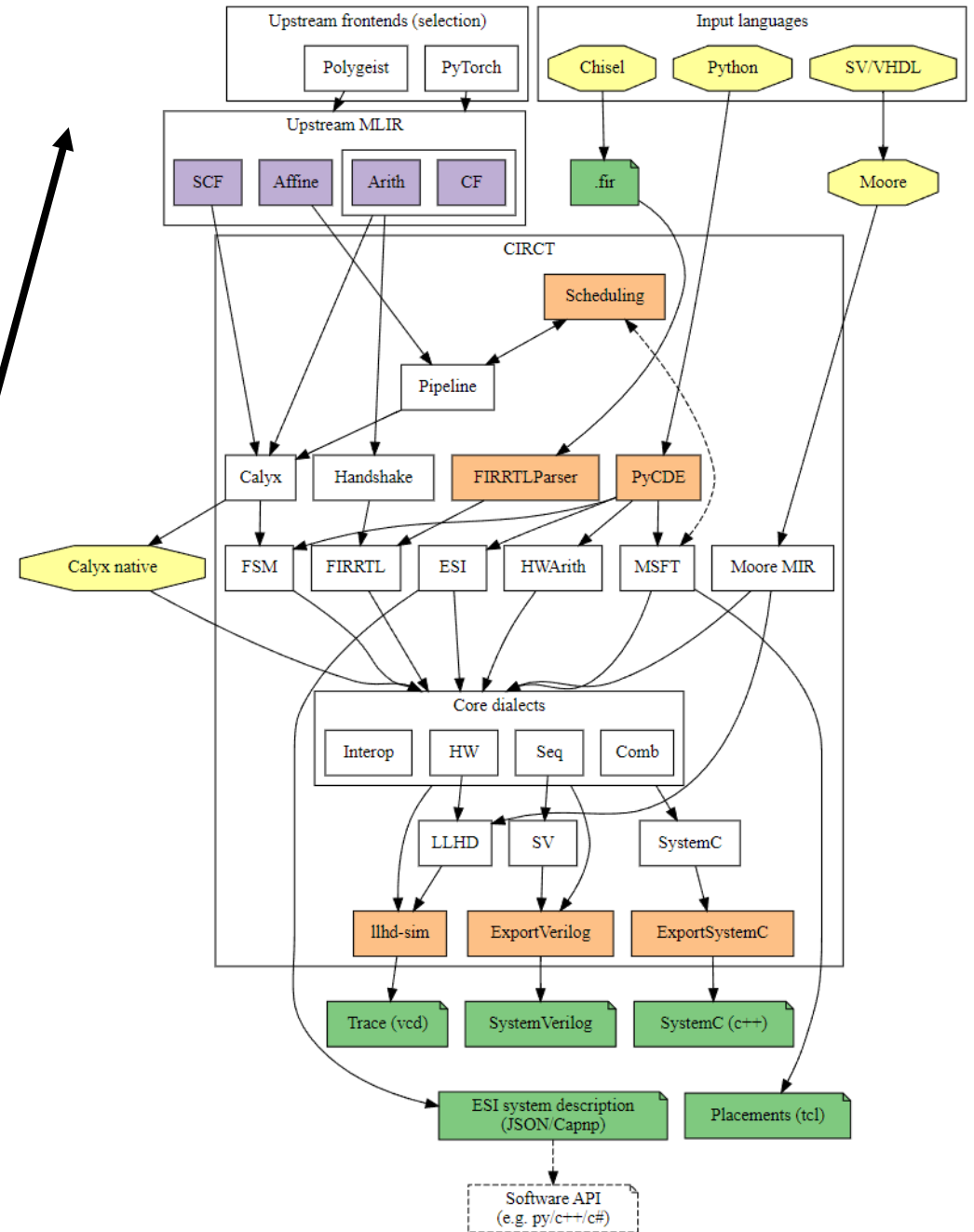
Farview

MLIR compiler

- How to create the operators?
- MLIR is a novel approach to building a compiler
- CIRCT^[3] built within MLIR, produces HDL
- Modularis being ported to MLIR



Different backends ...

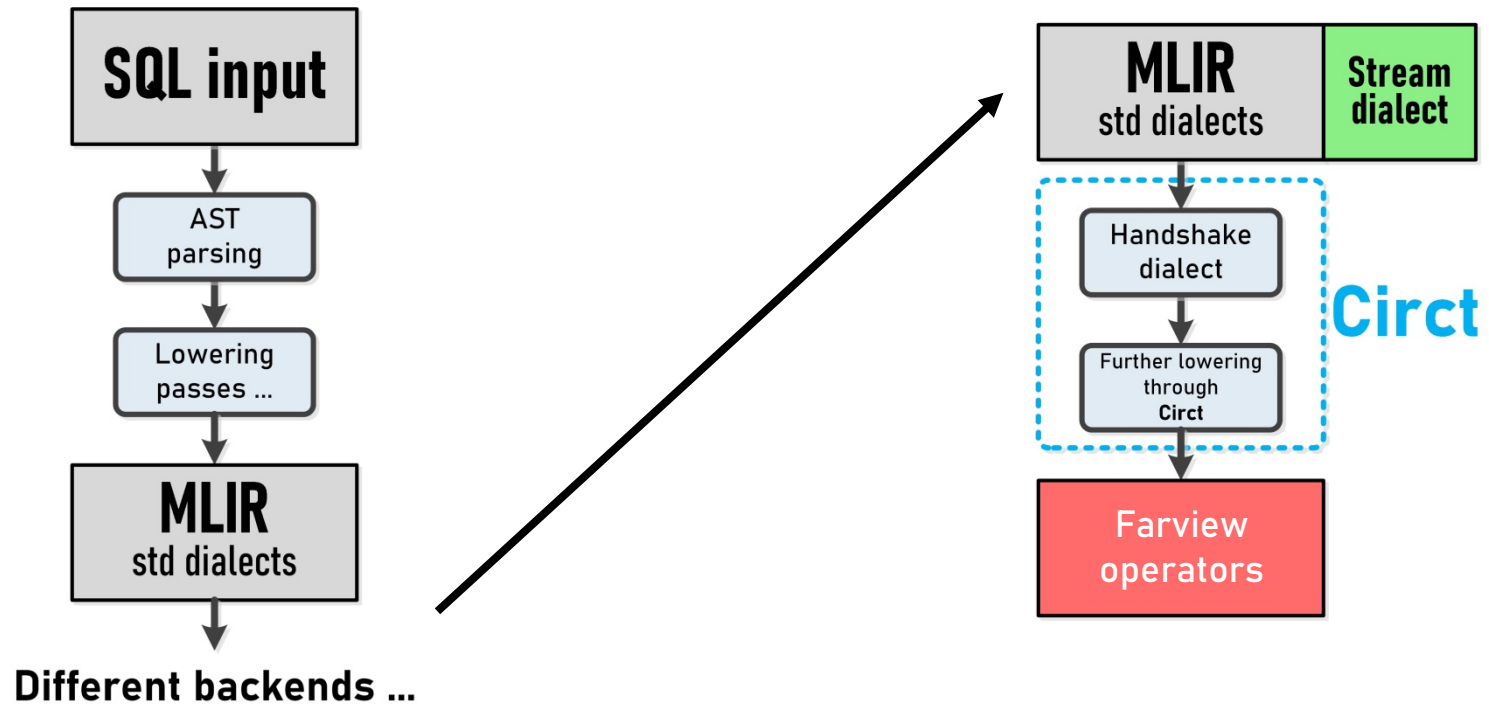


[3] <https://github.com/llvm/circt.git>

Farview

Stream dialect

- Stream-dialect => Stream-CIRCT project

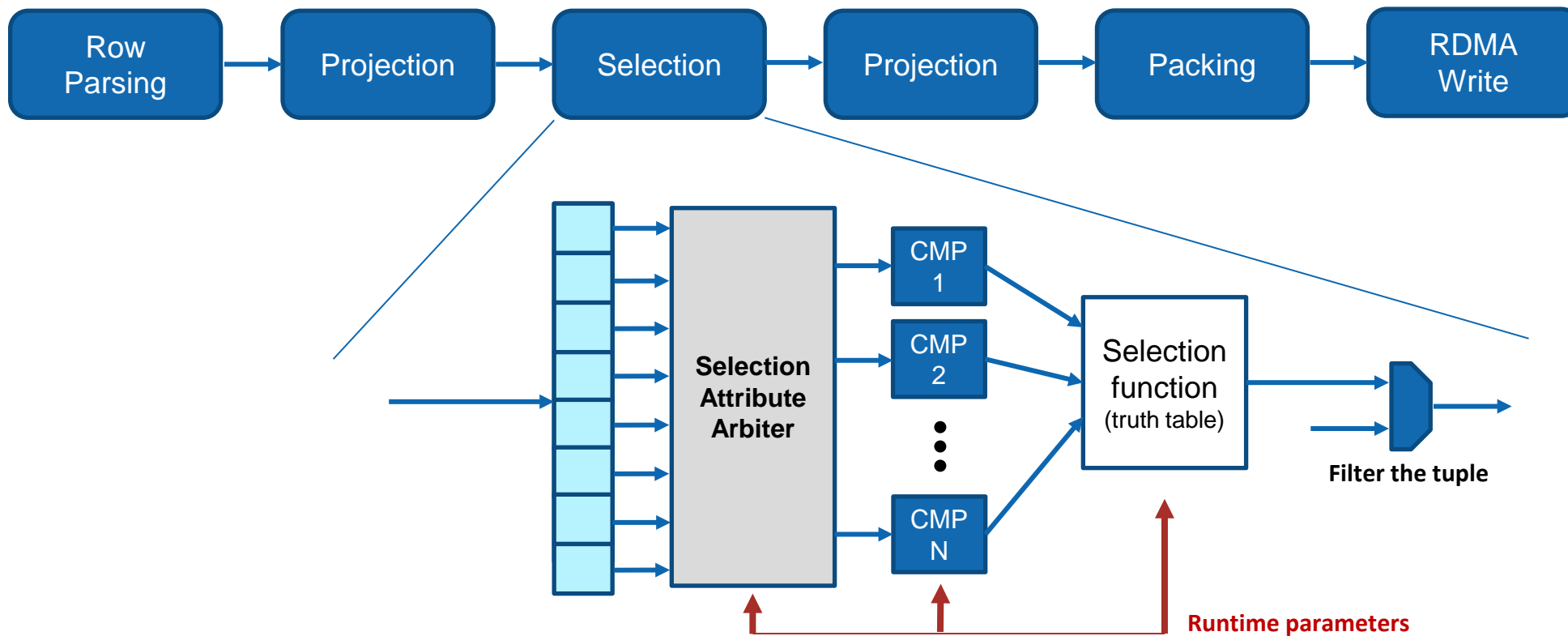


Evaluation

- **Farview** smart disaggregated buffer cache compared to two baselines:
 - **(LCPU)** Buffer cache implemented in local memory with processing on the local CPU
 - **(RCPU)** Remote buffer cache, no FPGA, implemented on a remote machine (Mellanox NIC)
- Benchmark comparisons to baselines performed for all implemented operators
 - RDMA throughput and response times microbenchmarks
 - Projection and smart addressing
 - Selection (100%, 50%, 25% selectivity)
 - Distinct queries
 - Group by queries
 - Regular expression matching
 - En/decryption
 - Multiple concurrent clients

Benchmarks (SELECT)

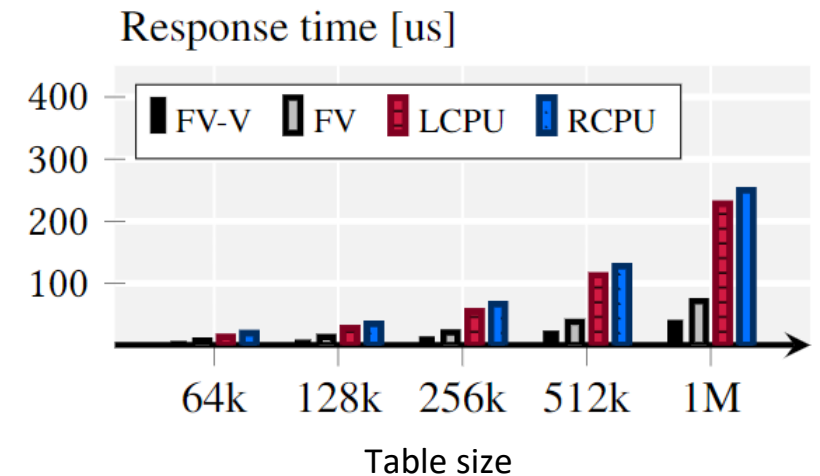
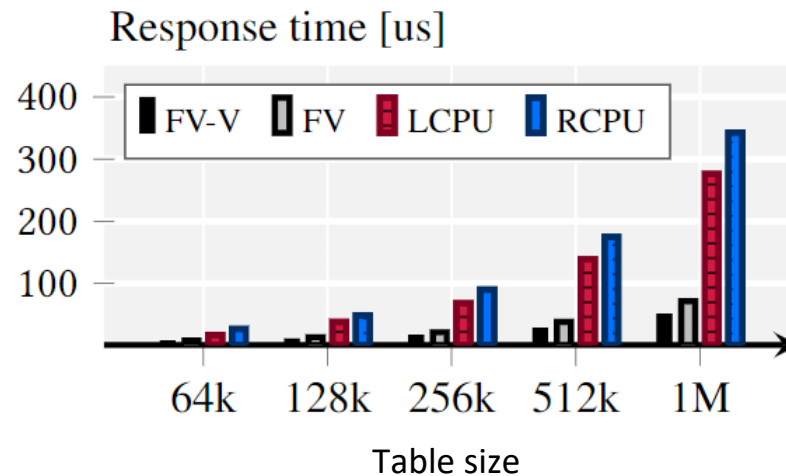
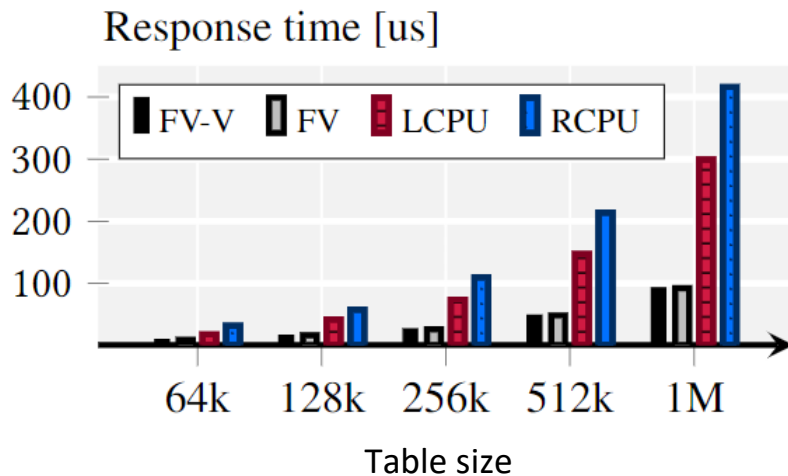
- Predicates provided at runtime
- Selection circuit generic enough for a wide range of selection queries



Benchmarks (SELECT)

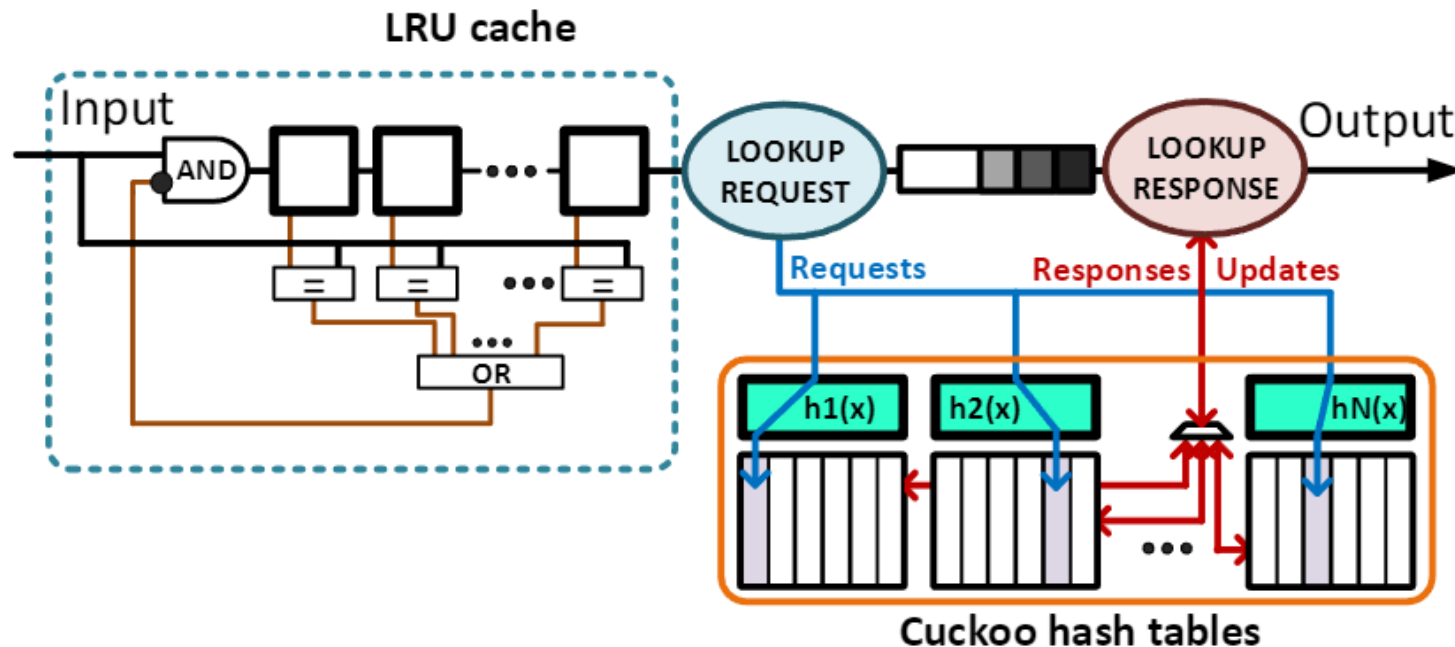
- **Farview** outperforms both baselines across different selectivity levels (due to the high filtering throughput of an FPGA with direct attached memory)
- Further increase in performance with parallelized pipelines (vectorization)

Response times for selection queries with 100%, 50% and 25 % selectivity, respectively:



Benchmarks (DISTINCT/GROUP BY)

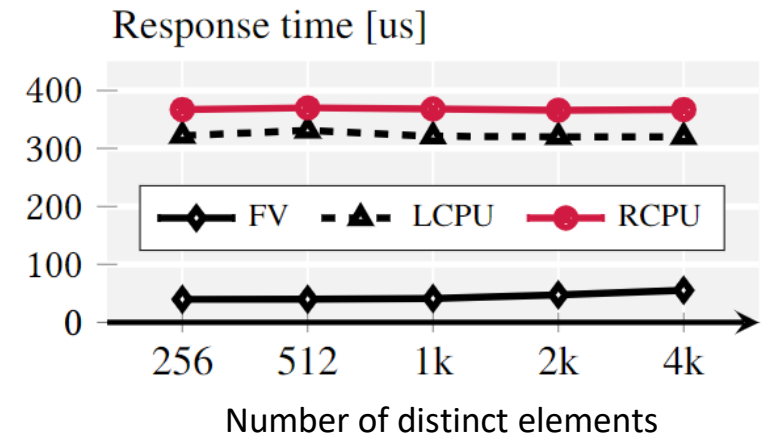
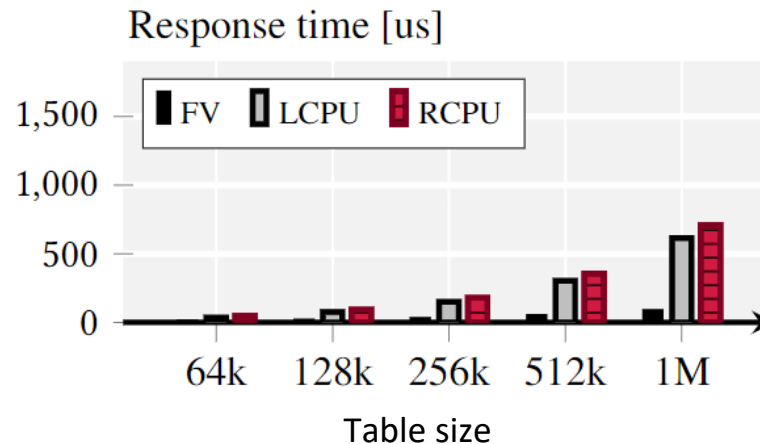
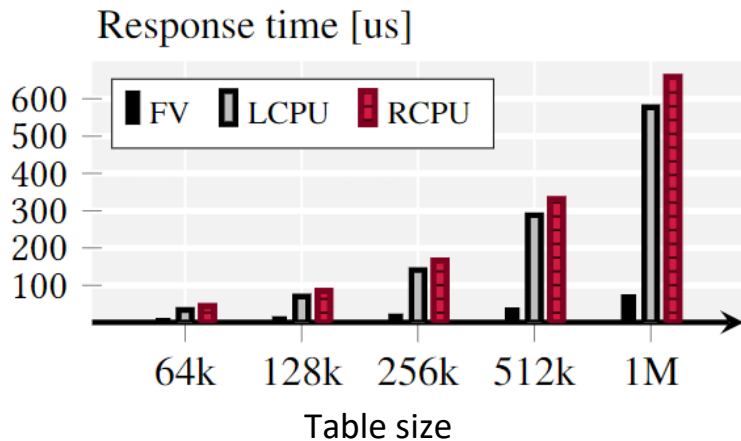
- Line-rate Distinct/Group by pipelines
- Cuckoo-hashing to reduce collisions
- No external bucket memory used (limited space in FPGA, might result in overflows)



Benchmarks (DISTINCT/GROUP BY)

- **Farview** outperforms both baselines, advantage scales with the data size
- Tested scenarios where FPGA memory is sufficient
- In case of too many collisions, additional post processing needs to be done on the client side

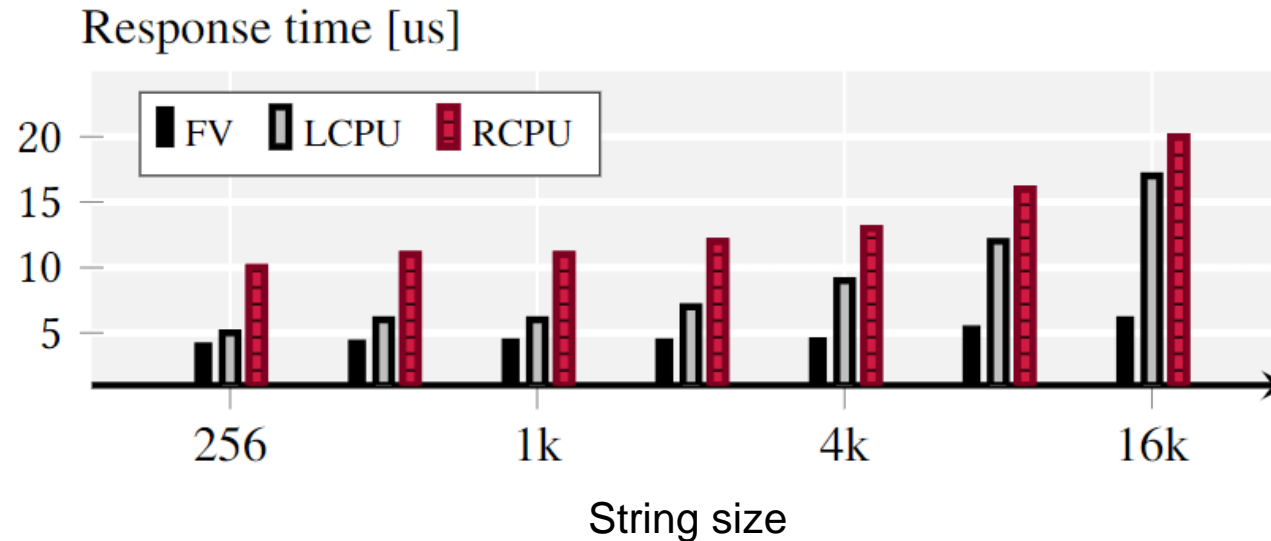
Response time comparisons for a **distinct** query, a **group by** query, and a **group by** query by on a stable number of elements



Benchmarks (REGEXP_LIKE)

- Regular expression matching is compute intensive in software
- Efficient compilation into NFAs on FPGAs^[4]

Comparison of **REGEXP_LIKE** operator on different string lengths

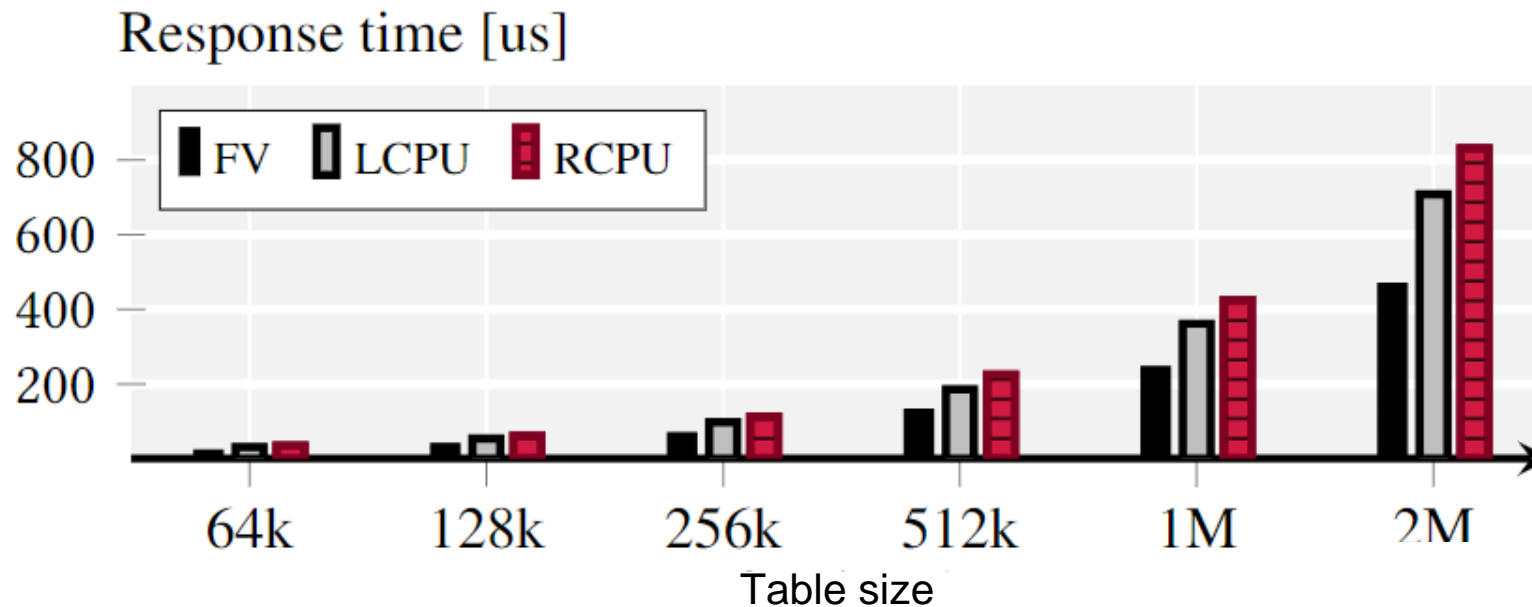


[4] *Runtime Parameterizable Regular Expression Operators for Databases*, FCCM '16
Zsolt István, David Sidler, Gustavo Alonso

Benchmarks (Concurrent Operators)

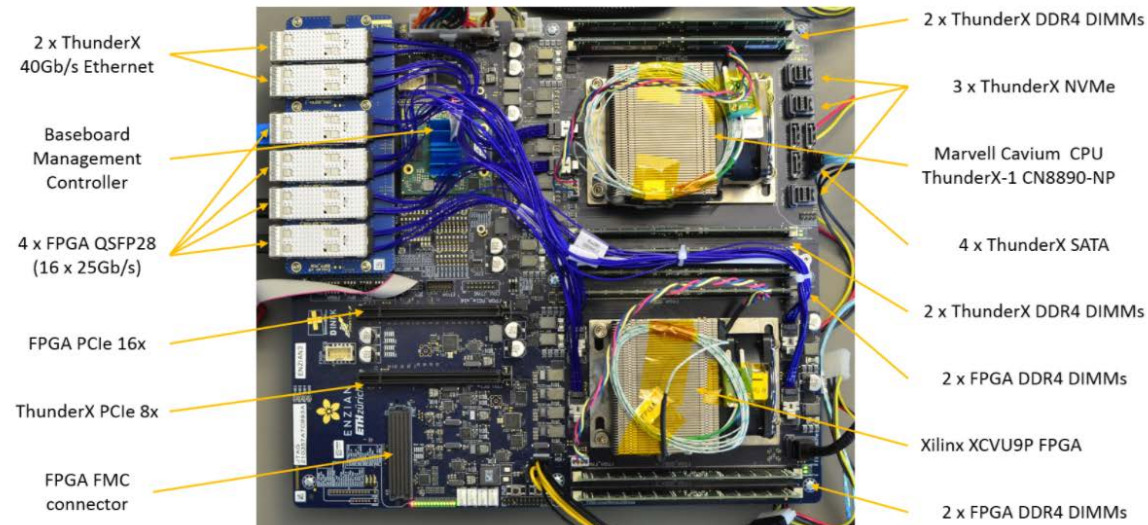
- Performance of concurrent operator pipelines (DISTINCT operation)
- Contention between multiple regions reduces the advantage of Farview

Performance comparison between 3 concurrent dynamic regions serving clients



Current and Future Work

- Implementation of the frontend for **Farview** (*Joining Farview and Modularis*)
- **Farview** stream circuit compiler
- Implementation of a storage layer
- Extending the operator set (*joins ...*)
- Distributed operation within the HACC cluster
- Larger scale deployments taking advantage of **Enzian** with 1 TB of DRAM per board:



<http://enzian.systems>

Questions?