arm

# High-performance network software for data analytics – what we've learned and what is next?

Arm

Pavel (Pasha) Shamis, Sr. Principal Eng.

Salishan Conference on High-Speed Computing, 2022

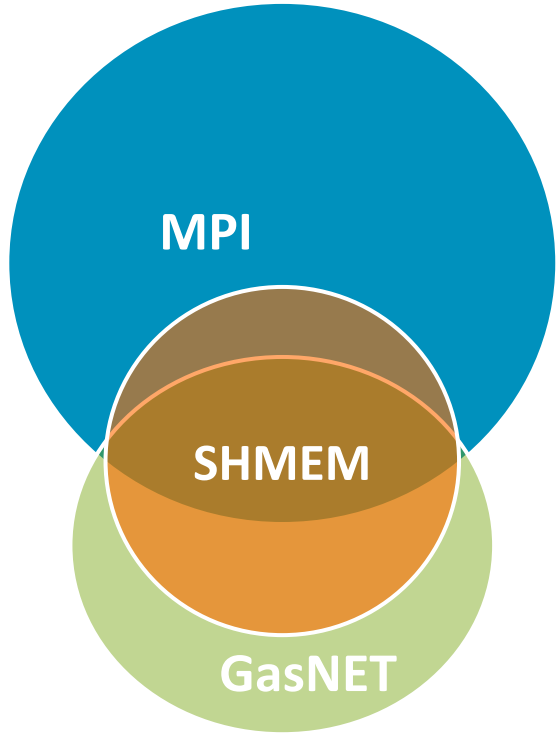


# My goals

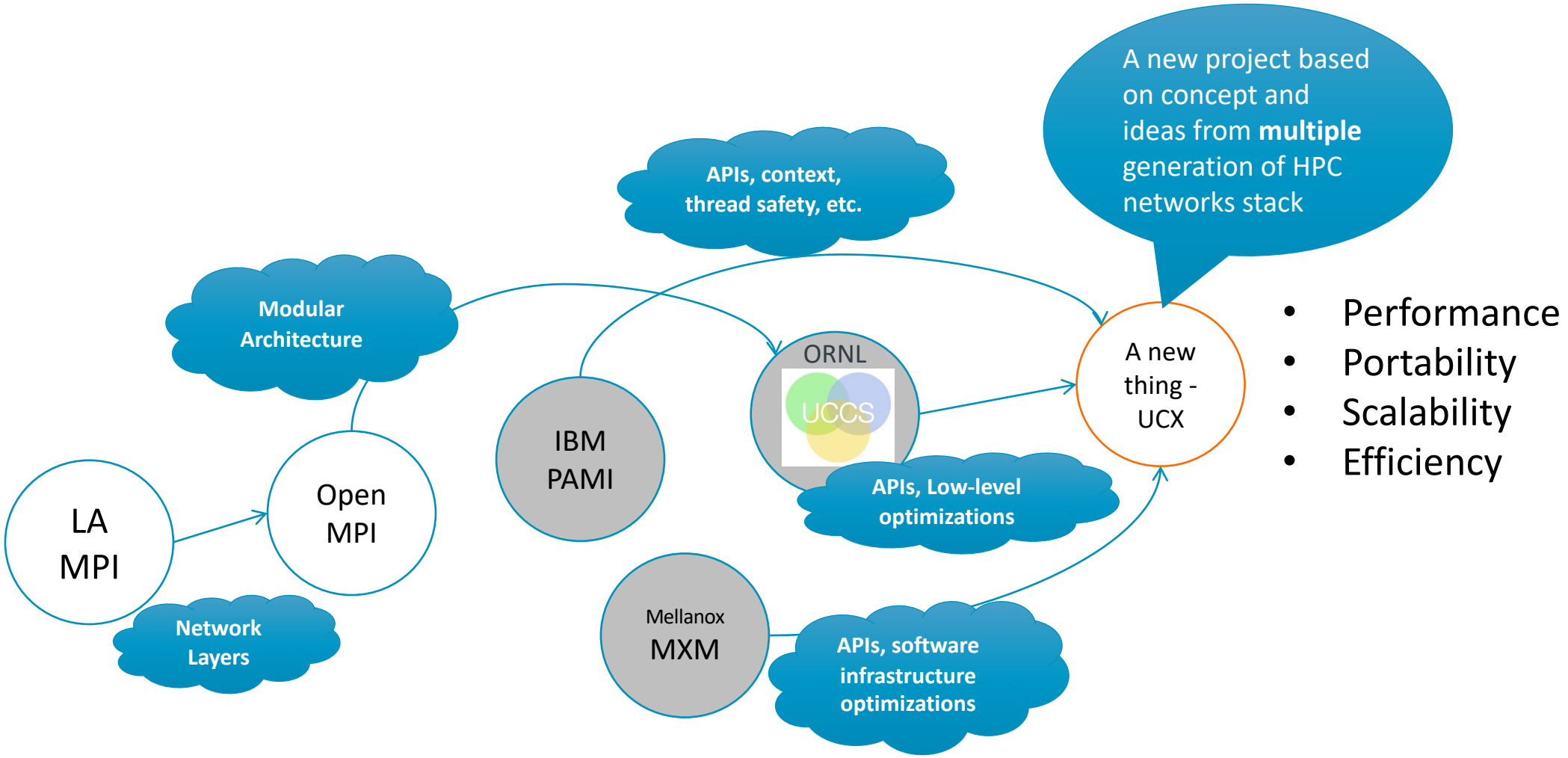
- Try not to brag about the awesome software we built
- Share the journey of building communication middleware for data analytics/HPC and lessons learned
- Share thoughts on the emerging generation of IO and storage hardware and what we shell do about the software

# How it is started …

- An ask from USG users for high-performance OpenSHMEM (SHMEM) implementation
- OpenSHMEM conceptually is a very thin wrapper on to of RDMA (Put/get/atomics) hardware abstractions
  - Cray got it right – simple API and semantics, lightweight, maps directly on hardware
  - Users expect a near bare-metal network performance
- From MPI developer perspective OpenSHMEM semantics on the low-low level is very similar to MPI low-level internal layers
- OpenSHMEM is a relatively small community and vendor cost of a stand-alone product is very high (10s of millions)
  - Let's reuse as much code as we can
- OpenSHMEM use-cases look-and-feels like data analytics processing
  - Put, get, update, reshuffle, sort/search in memory data structures
  - In a fact some people considered to map database semantics on top of SHMEM




# The perfect storm …




- Performance
- Portability
- Scalability
- Efficiency

# Setting the project…

- Goals: Performance, portability, productization
- Partnership: first meeting participants ORNL, Sandia, UTK, IBM, Mellanox
- Legal and licensing: BSD3, copyrights, company (UCF).
- Project Architecture: protocols, transports, services, memory
- Main use-cases: MPI, SHMEM , potentially 3rd party languages, storage, data analytics
- Accelerators: GPU as a first-class device
- Productization: programming infra, CI, unit testing, code review methodology, etc.

# Progress

- **PoC**
- API: Blocking RMA, some atomics, Blocking Send/Receive (tags)
- Networks: RDMA Verbs, Vebs “bypass”, Cray uGNI, basic shared memory

OpenMPI/SHMEM

- **HPC**
- UCP API and ABI Backward Compatibility
- API: Non-blocking Send/Receive, Stream, Atomics
- Networks: Devx, Multirail
- Shared Memory: KNEM, CMA, XPMEM
- Initial GPU support

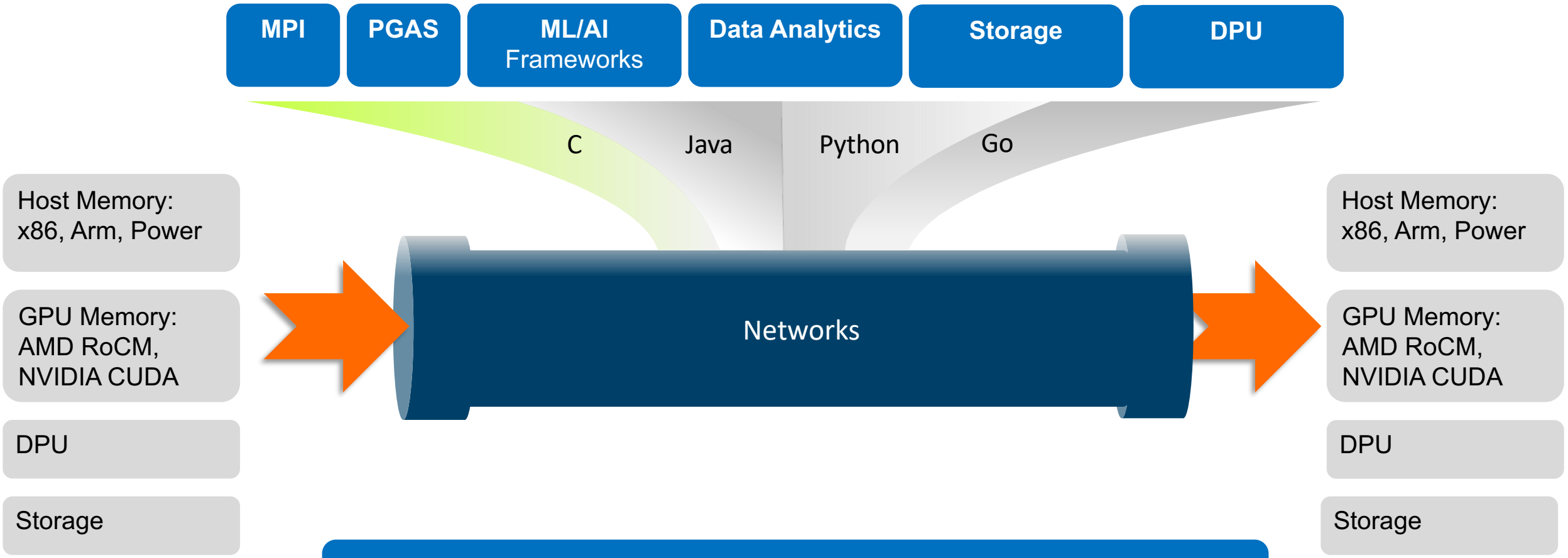

Mpich, OpenMPI, SHMEM*, GasNet, Charm++, commercial prod

- **AI and Data Analytics**
- Python , Java
- Active messages
- TCP
- Extendable ABI interface
- Performance: more of hardware offloads
- Multi-GPU, Multi-NIC

Rapids/DASK, Spark, Arrow, NCCL, File systems

- **Disaggregated Datacenter**
- Go
- Active messages with zero-copy
- Wire-level protocol backward compatibility
- UCT: Extendable ABI interface
- SmartNICs
- GPU corner cases
- Support for new network architectures

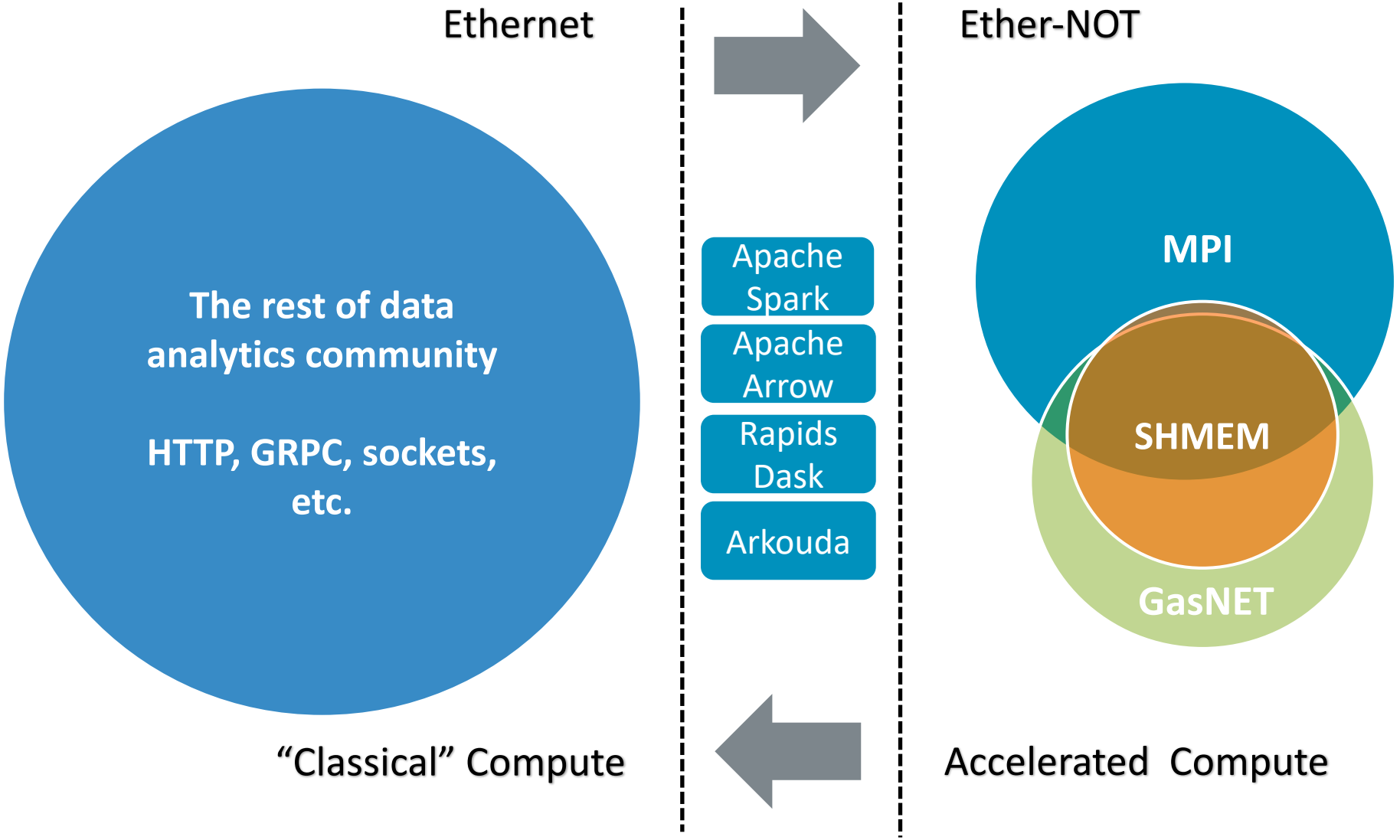
# UCX




**High-Performance Universal Data Mover**

# Convergence is real and it is not just about hardware

"Smart" networks and accelerators is a new normal




# Lessons learned

- PoC – proof that you can work together
  - Find hardware/software partners that share your LONG-TERM strategy/vision and economic incentives
    - Who/what will drive the project once you are out of funding or contract expiration?
  - Target multi-market hardware/software solution (HPC, Data analytics, ML, AI, Storage, etc.)
- Hardware vs Software
  - it is constant negotiation and hardware engineers are NOT always wrong
- Majority of HPC network APIs work mostly for HPC community
  - RPC-like  (active messages) or streaming semantics works well for the rest of the world
- Collaboration with Linux kernel community
  - RDMA-CORE: There is over 15 years' experience in building kernel/user space interfaces
- There are less and less people that “speak” C
- Architect project for growth and constant change
  - You will get API and ABIs wrong
    - iterate, extend, and improve while preserving backward compatibility (as long as it makes sense)
  - Plan for heterogeneity in processing element, memory, and interconnect
- Upstream first
  - We are CI maniacs

arm

# The next generation of IO and storage hardware and what we do about this ?



# Datacenter Native System Architecture

Typical Server

Host CPU

Application

Data Processing

Data Filters | Compression | Encryption | Confidentiality

Virtualization

PCIe Functions

NIC | SSD | SSD | SSD | SSD | SSD

Security
Efficiency
TCO

Datacenter Server

Host CPU

Application

Data Processing

PCIe/CXL/CCIX/Network functions

Data Filters | Virtualization | Confidentiality

DPU

arm

C CSD | E CSD | C CSD | E CSD | C CSD

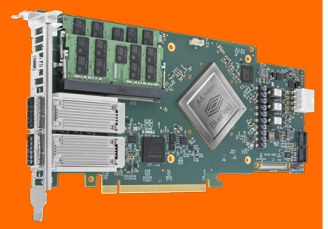
arm

Fixed-function accelera

"Invisible" Compute


Computation Storage

DPU



# CSD deployments: Alibaba PolarDB and Amazon Redshift with AQUA

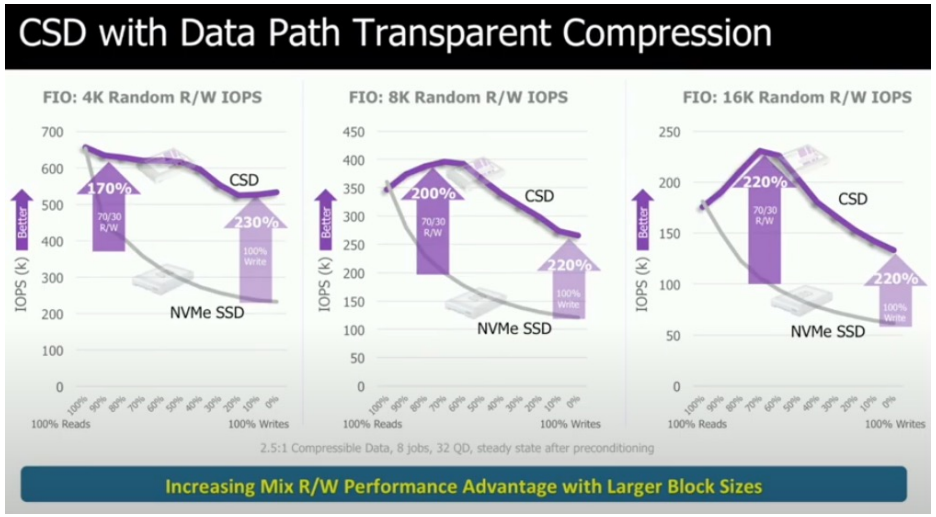
Alibaba's PolarDB is their hosted relational storage service. Accelerated by ScaleFlux CSDs as shown at USENIX FAST '20, they claim a 2x performance gain over a more traditional architecture.

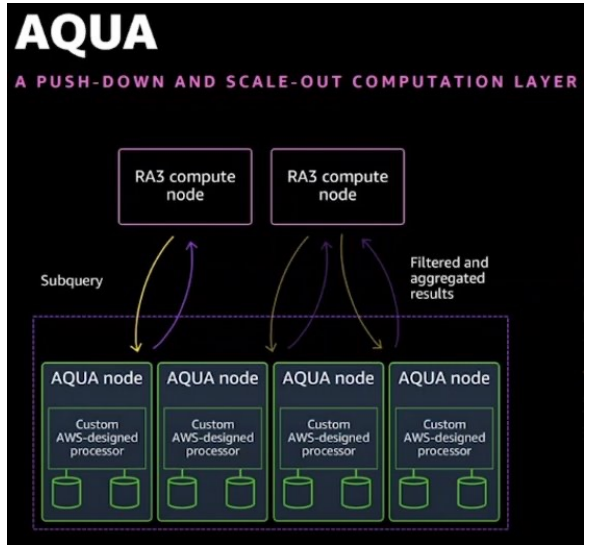
https://www.usenix.org/conference/fast20/presentation/cao-wei

Redshift is Amazon's relational database service. AQUA is an architectural improvement that uses CSDs to execute SQL filter instructions.

Claims 10x query performance, and 3x overall better performance at the same cost as Redshift without AQUA.

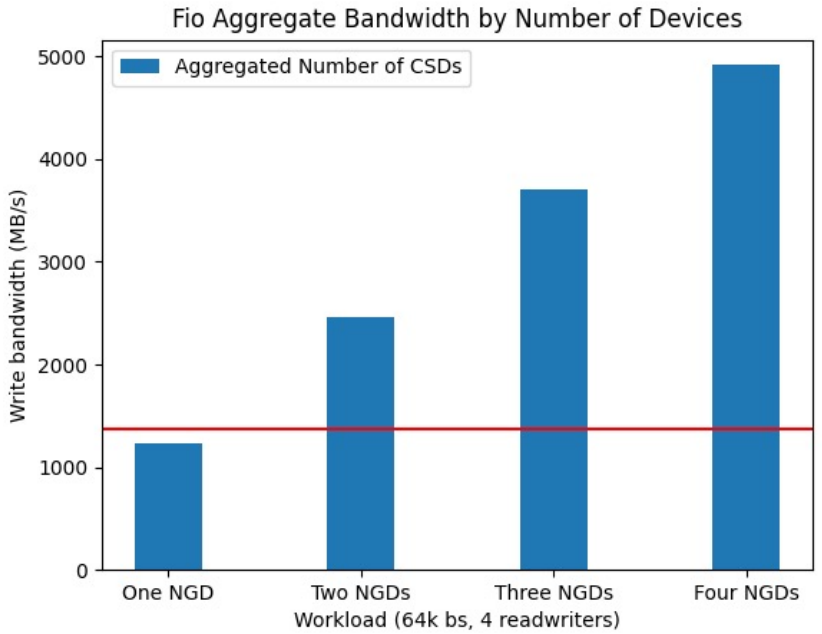
https://www.youtube.com/watch?v=0LVyd-kqpdE

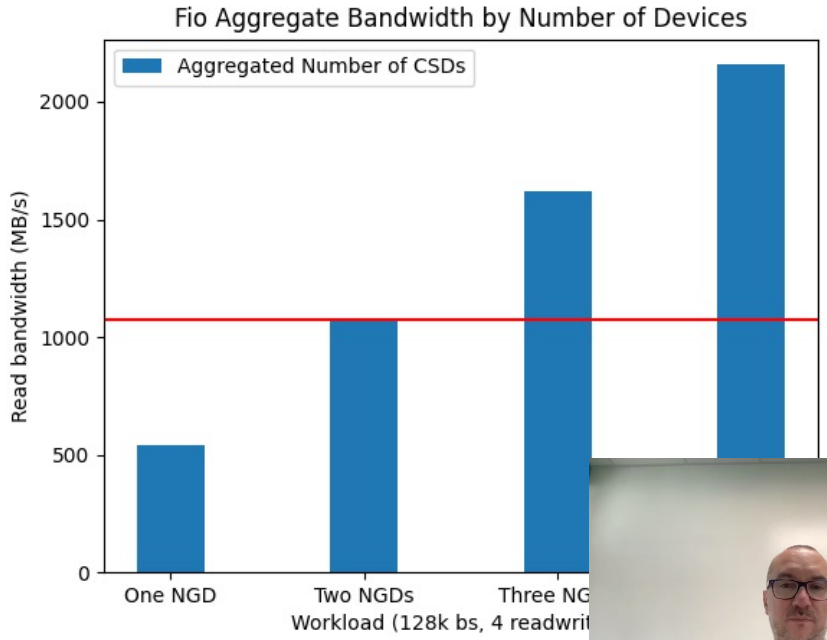



# Computational Storage Potential

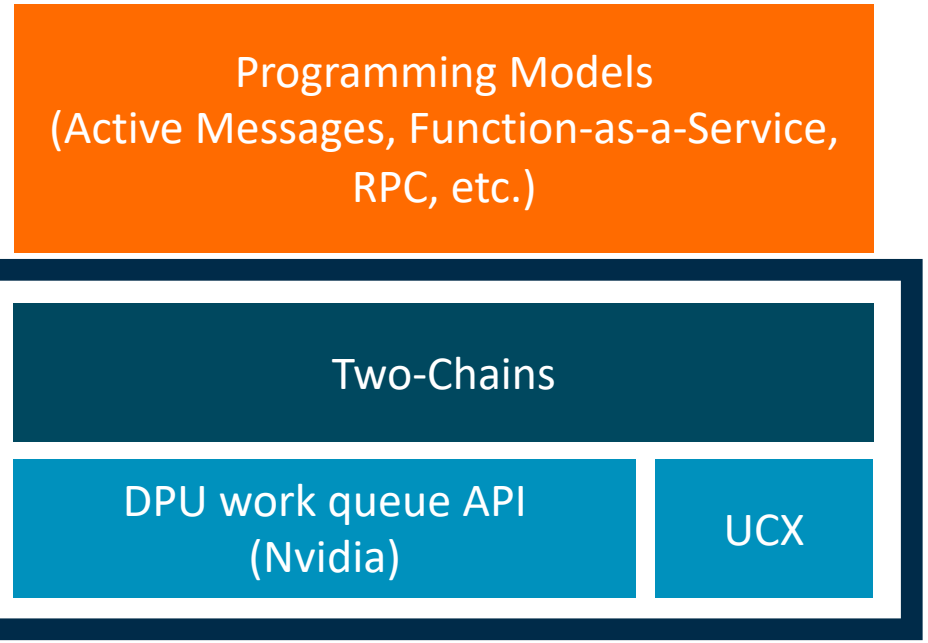
- Testing setup: AMD Epyc (7401P) with 4 NGD (4 x Cortex-A53) CSDs attached. All workloads dispatched simultaneously.
- Two drives can outperform an AMD EPYC core on reads, and slightly more than one outperforms a single core on writes.
- For sufficiently partitionable data an array of CSDs could free data concerns from the host entirely
- SSD ~12W power envelope (including storage and 4 cores) vs AMD Epyc (7401P) 237W/24c ~ 9.8w per core (2 threads)
  - U.2 power budget is under 25W
  - M.2 power budget is under 9W

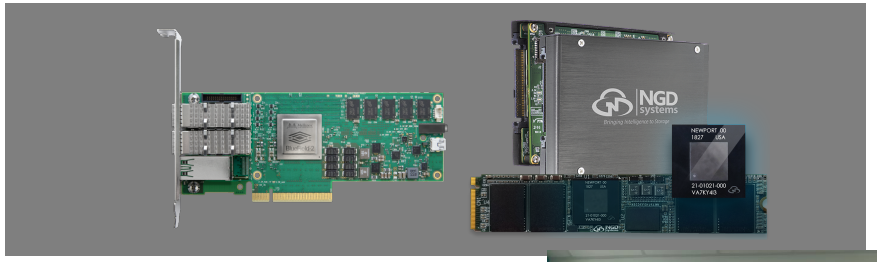




# The *Two-Chains* Framework

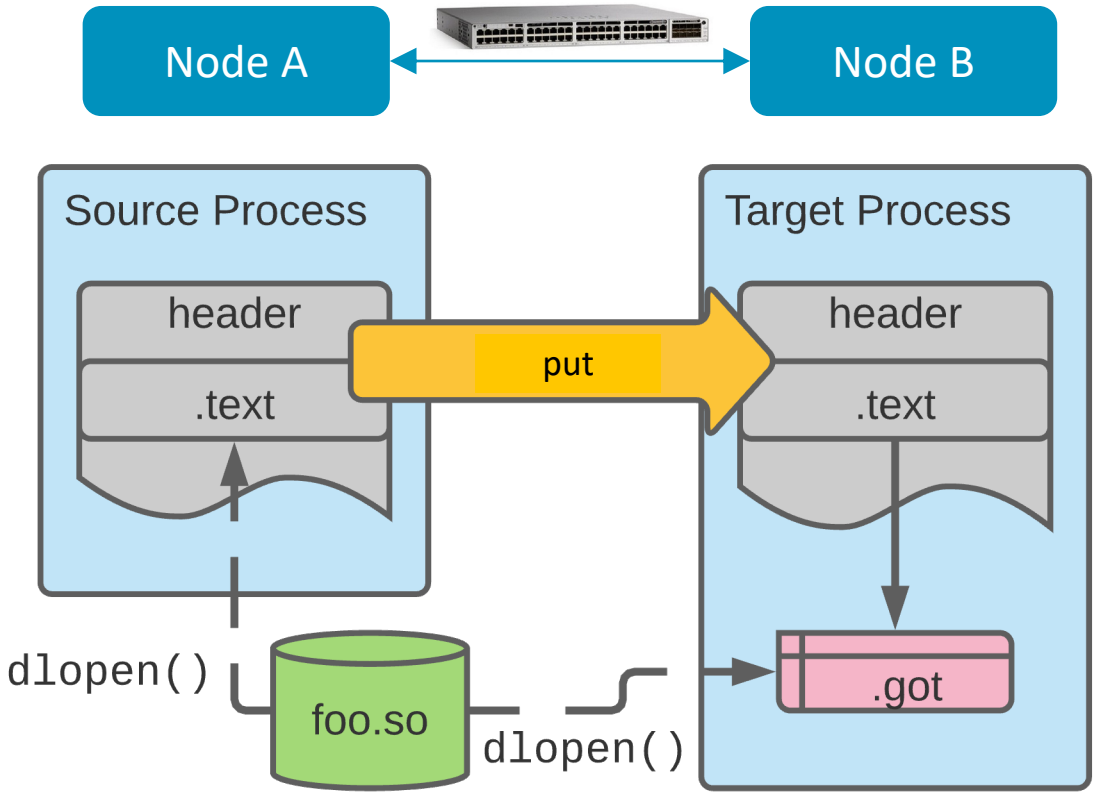
- Provides packaging, transfer and execution of injected functions (***ifuncs***) on local and remote processes
  - Functions are loaded as dynamic libraries
  - Messages contain binary code and data
- Fast, lightweight and portable
  - Leverages Arm's instruction and data cache stashing
  - Injected functions (***ifuncs***) are written in regular C code
  - Works on local/remote CPUs, DPUs and CSDs
- Extension of the UCX framework
  - Works with TCP, Shared Memory, RoCE, IB, etc.
- *Two-Chains: High Performance Framework for Function Injection and Execution*, IEEE CLUSTER 2021
- *UCX Programming Interface for Remote Function Injection and Invocation*, OpenSHMEM 2021




https://github.com/open

# The *Two-Chains* Workflow




| FRAME LEN | GOT OFFSET | PAYLOAD OFFSET | IFUNC NAME |
|---|---|---|---|
| SIGNAL | CODE | | |
| PAYLOAD | | | |
| | | | SIGNAL |

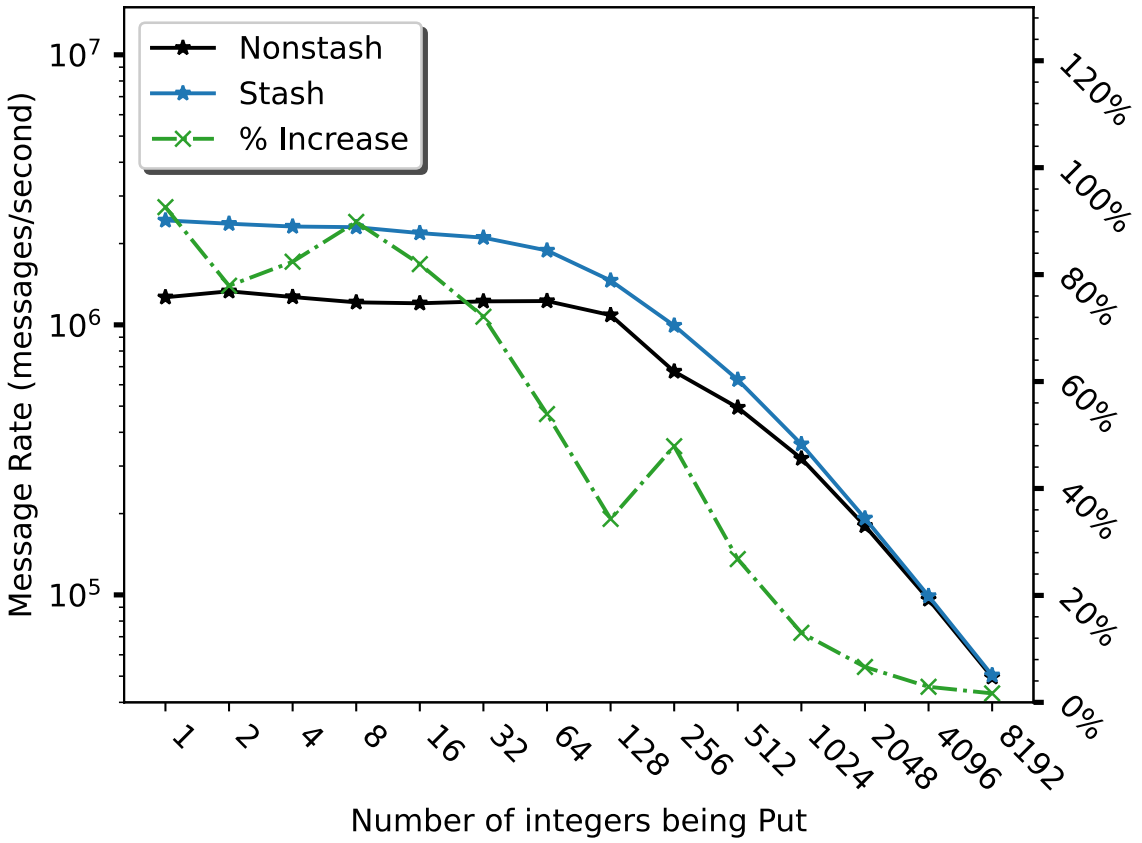
*ifunc* (code + data) wire protocol

```
ldr x1, :got:printf
blr x1
```

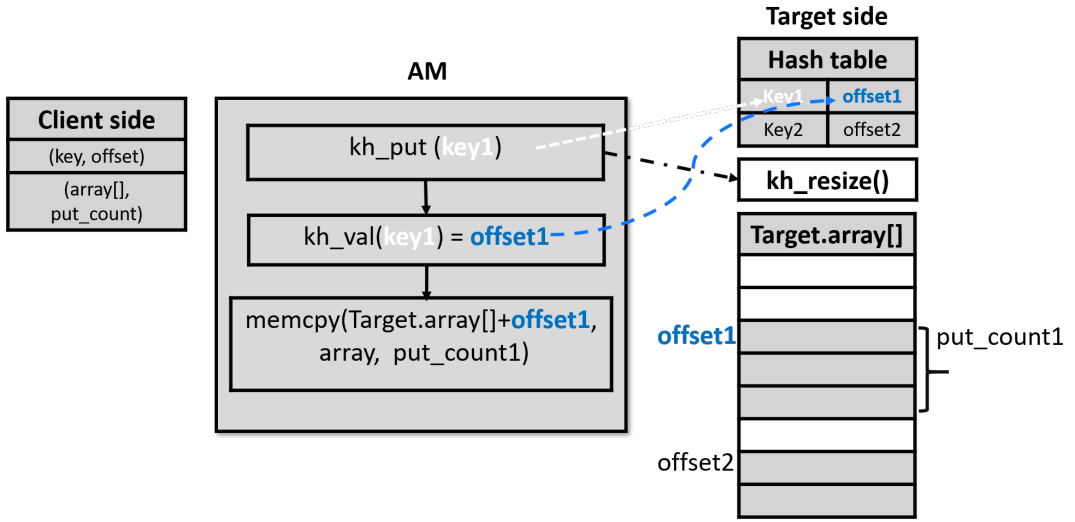
patch_asm.py

```
ldr x1, foo$got
ldr x1, [x1, #:got_lo12:fib]
blr x1
```

# *Two-Chains*: Performance highlights



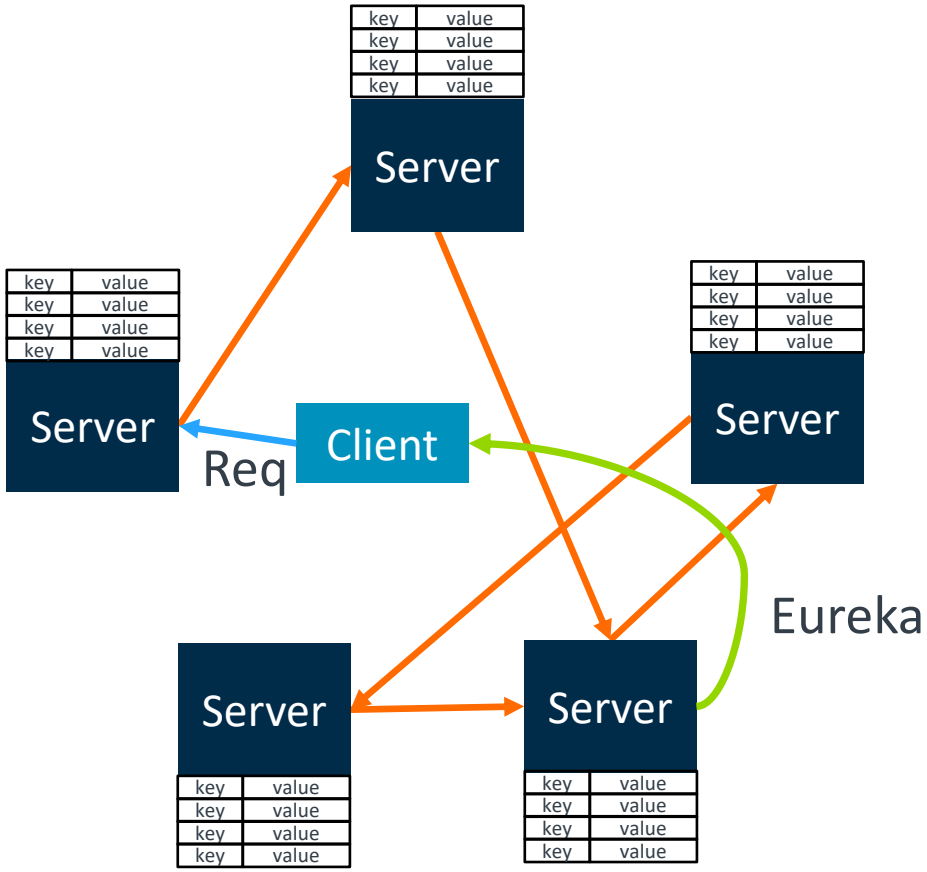

Google's *SNAP*-like indirect-put network protocol



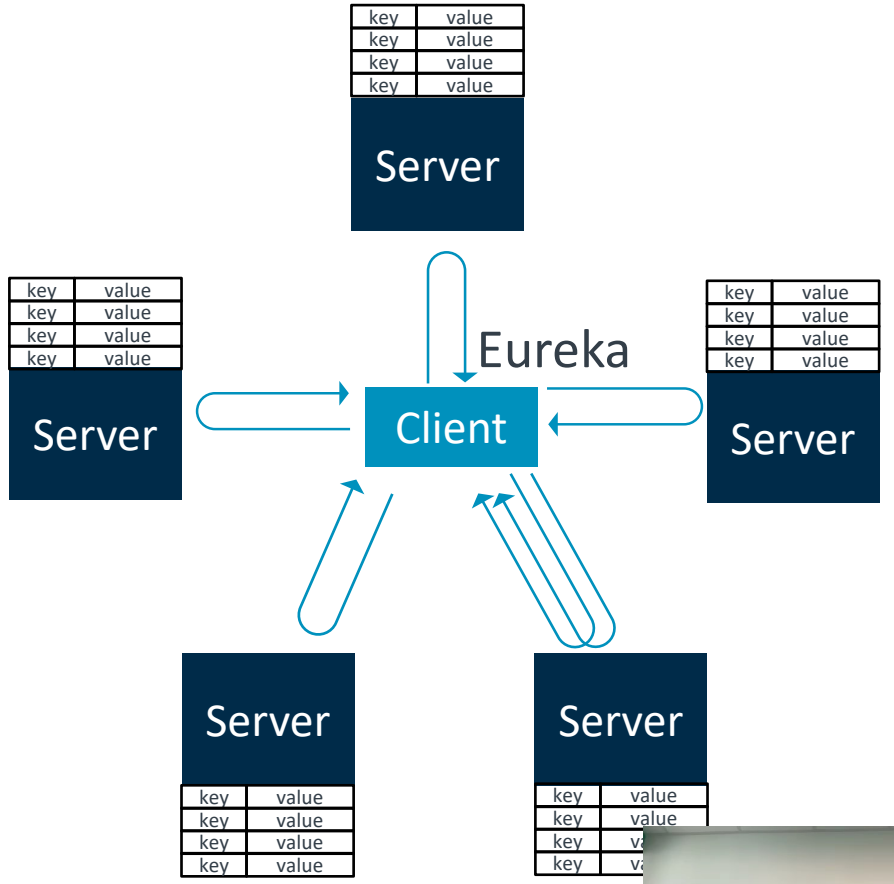

Hardware optimizations for function/data injection (cache stashing, WFE, etc.)

- 92% increase in indirect put injection rate
- 31% reduction in latency
- up to 2.4x improvement in tail latency
- Between 2.5x and 3.8x CPU cycle red when polling using WFE

# Pointer Chasing Benchmark




Ifunc




Remote-get

# Pointer Chasing Benchmark - Preliminary Results

| Client-Server | Depth | Method | Pointer Chase Rate | Improvement |
|---|---|---|---|---|
| A70-A70-A70 | 10 | ifunc | 31,080 chases/sec | 17.91% |
| | | get | 26,360 chases/sec | |
| | 1,000 | ifunc | 375 chases/sec | 41.31% |
| | | get | 265 chases/sec | |
| N1-N1-N1 | 10 | ifunc | 72,280 chases/sec | 62.24% |
| | | get | 44,550 chases/sec | |
| | 1,000 | ifunc | 867 chases/sec | 103.16% |
| | | get | 427 chases/sec | |

# Future Challenges & Opportunities

Security

Security is going to be the main challenge

- Running side-by-side with the “trustzone” of the datacenter
- Partitioning of hardware, isolation, virtualization, trust establishment across all elements in distributed system

[1] Confidential Compute Consortium- Veracruz: https://github.com/veracruz-project/veracruz

[2] Icecap (seL4): https://gitlab.com/arm-research/security/icecap/icecap

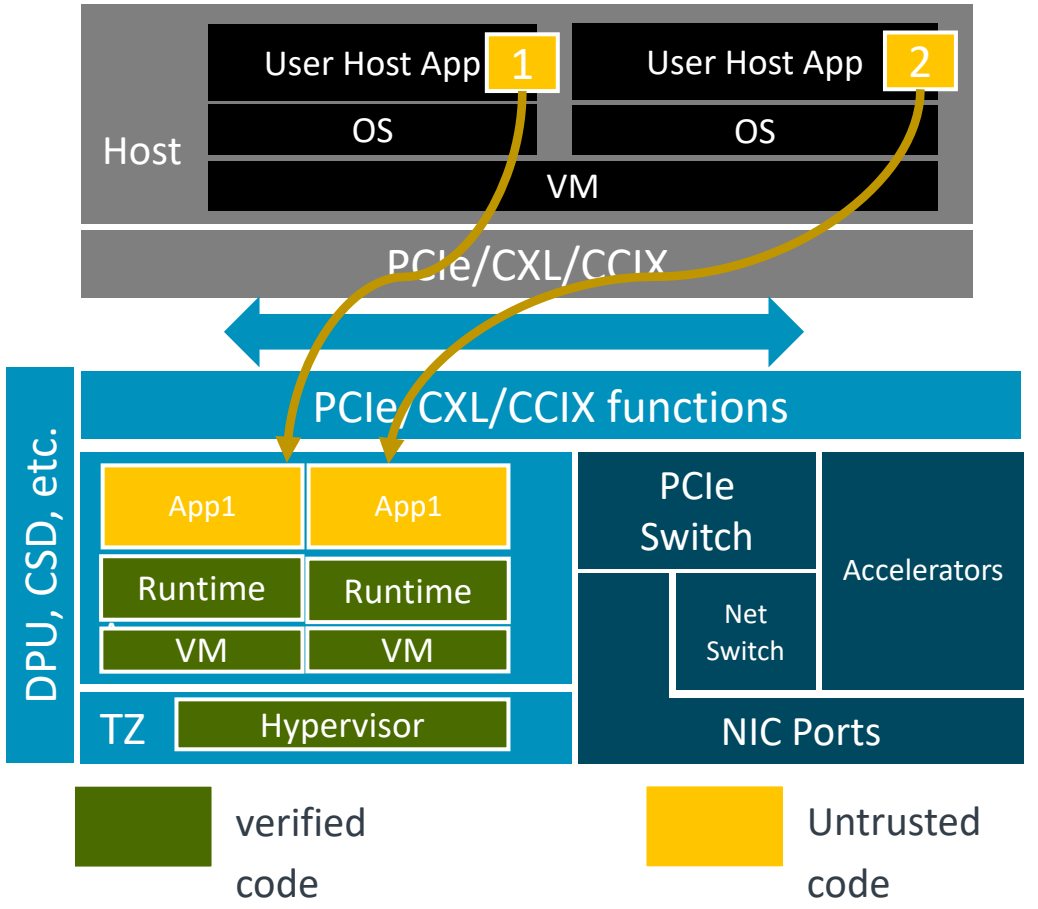
Programming models

Memory with computing elements may not have application-class cores

- Realtime and micro-controllers, accelerators, etc.
- What is the right representation of the code
- Minimizing assumption about underlying runtime

[1]Two-Chains: High Performance Framework for Function Injection and Execution, IEEE CLUSTER 2021

[2] https://github.com/openucx/ucx-two-chains




# Acknowledgments


- Luis Pena, Jon Hermes, Alexandre Ferreira, Eric Van Hensbergen, Kshitij Sudan, Jason Molgaard
- Wenbin Lu (SBU)
- Steve Poole (LANL)


arm

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos