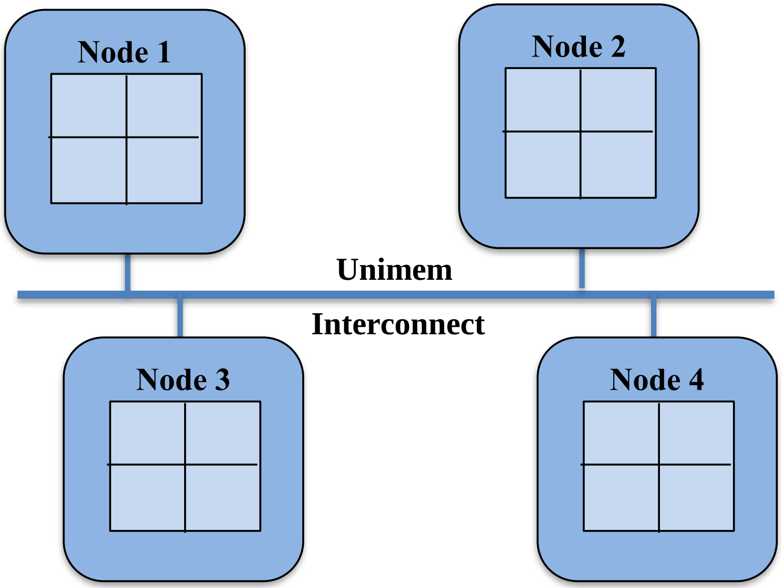
A Flexible & Efficient Shared Memory Abstraction with Minimal HW Assistance

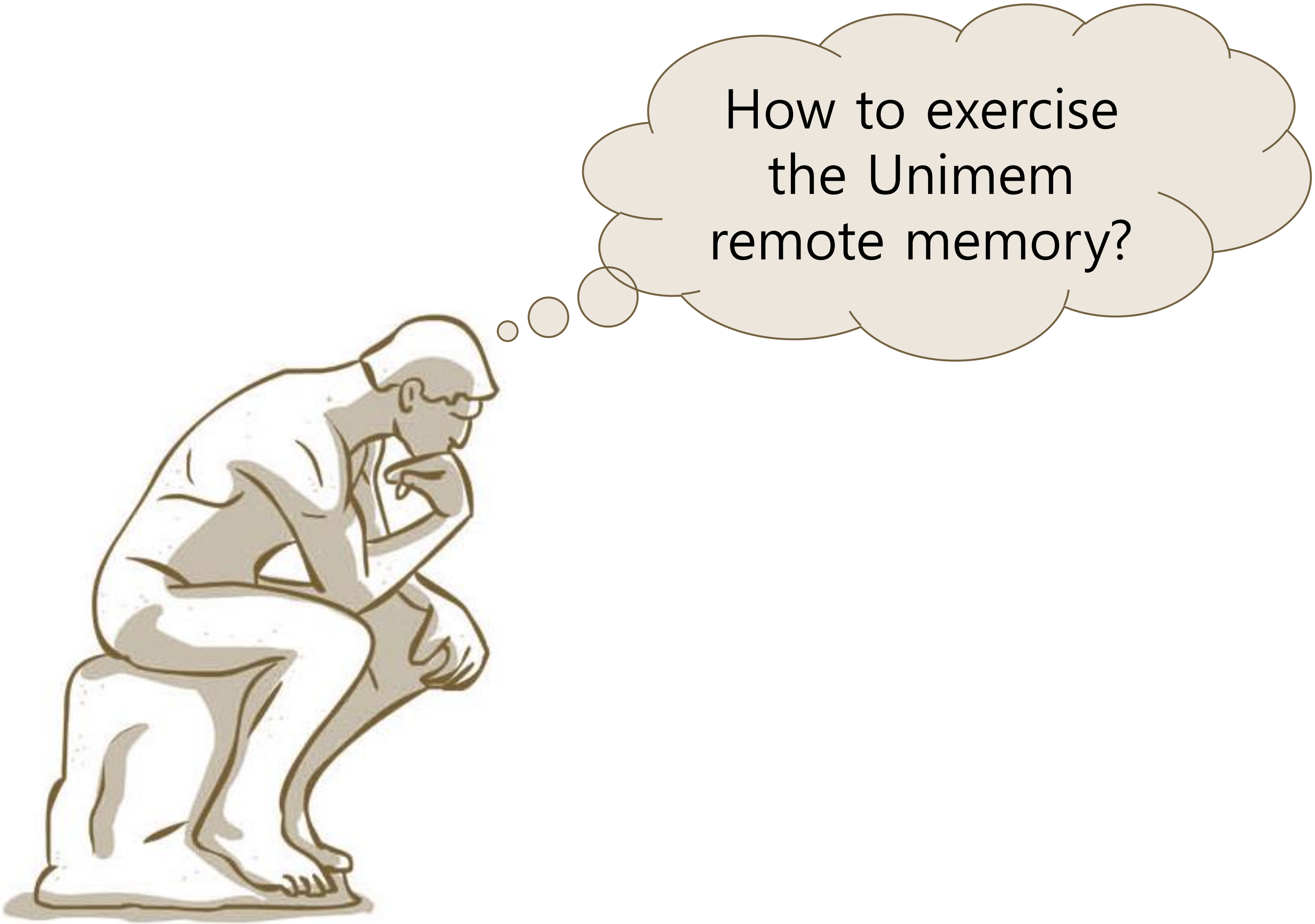
Nikolaos Kallimanis Manolis Marazakis Nikolaos Chrysos

Computer Architecture and VLSI Systems (CARV) Laboratory
FORTH – ICS

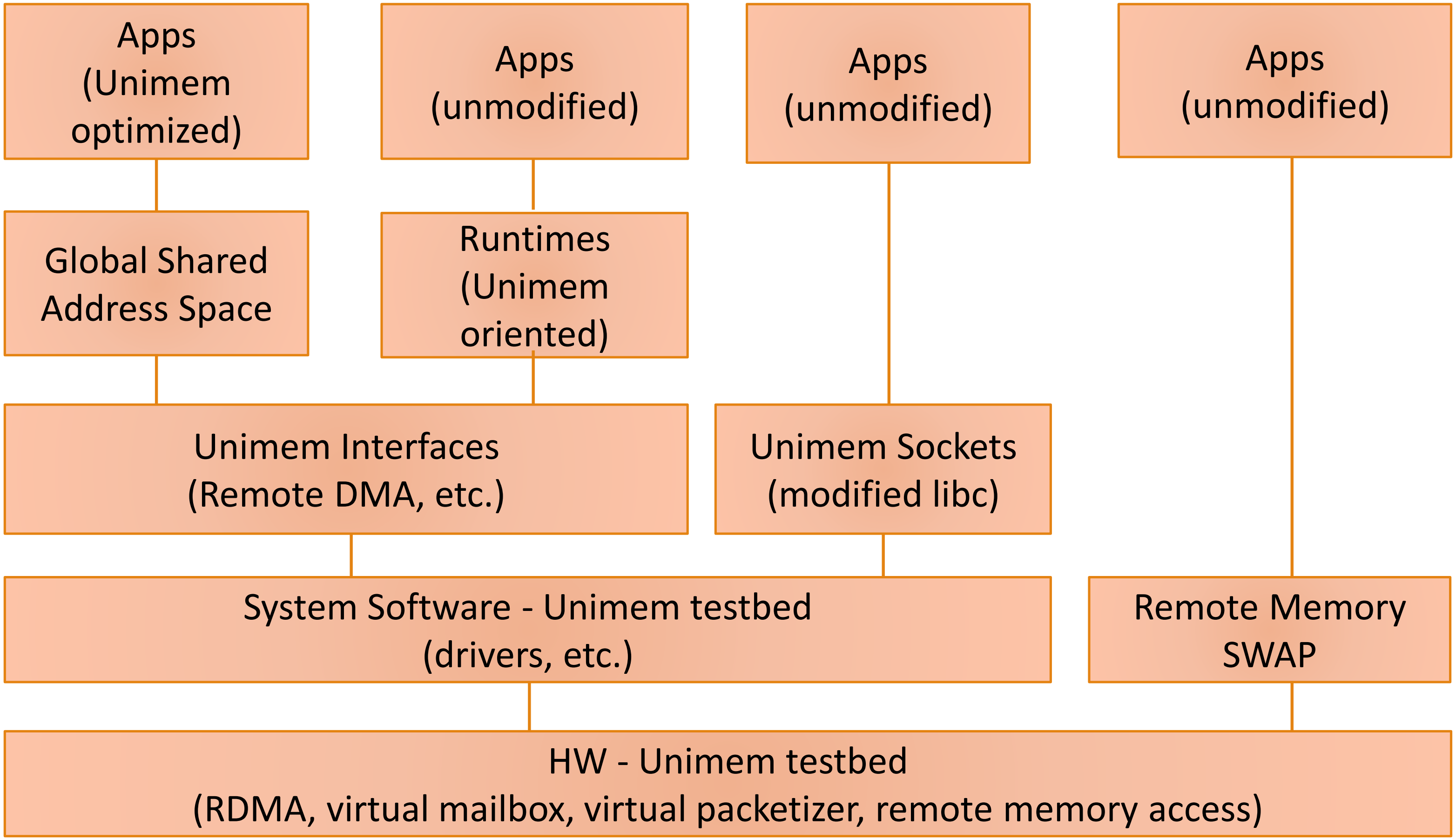
Unimem Architecture

- Communication mechanisms of the Unimem architecture:
  1. Load/Store instructions across remote nodes.
  2. Every page of physical memory is **cacheable only in a single node.**
  3. Efficiently copying large amounts of memory from/to remote nodes.
  4. Send and receive of small atomic messages in a low latency manner.
- Towards exascale.




How to exercise the Unimem remote memory?

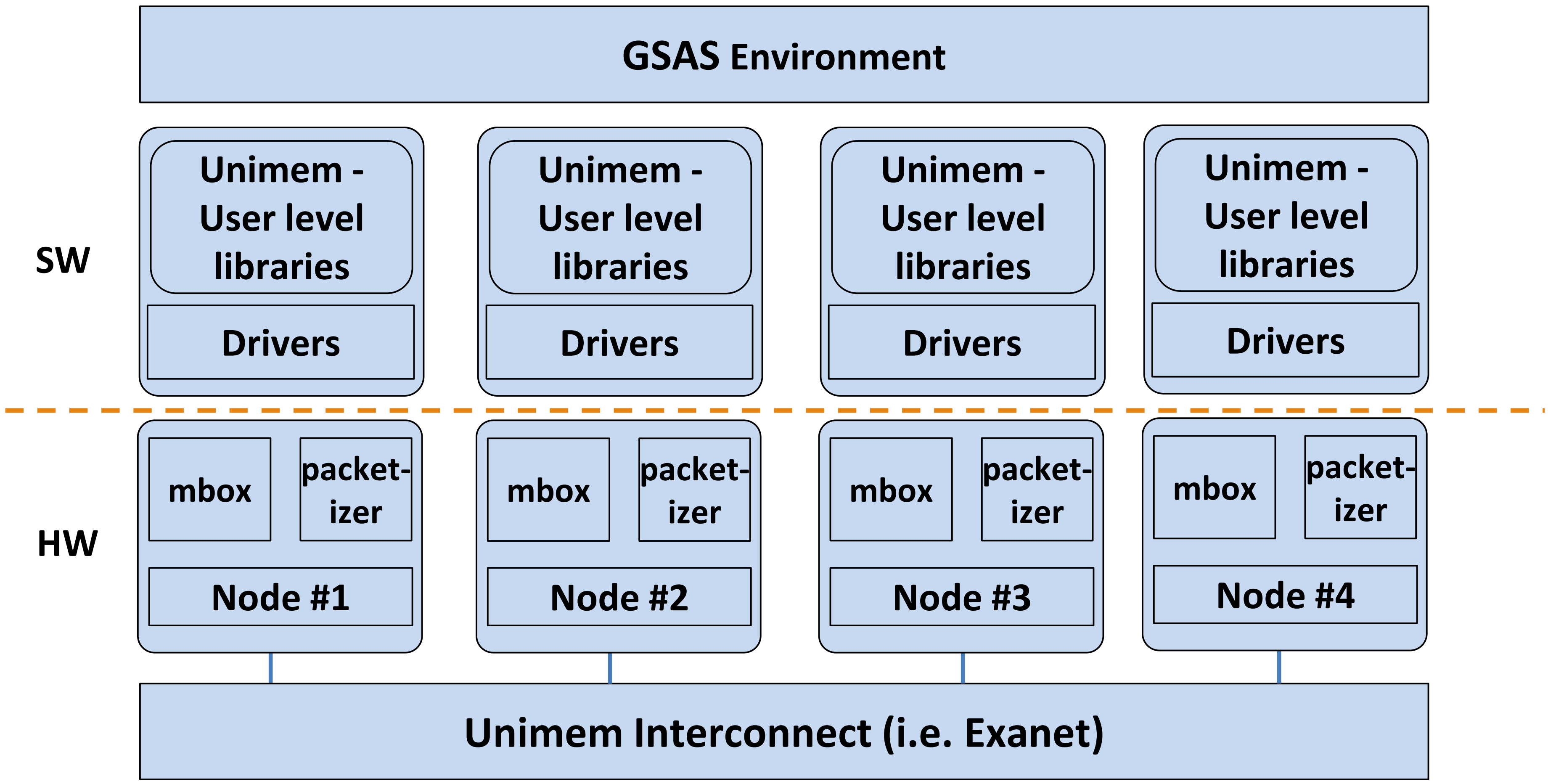
Unimem's APIs



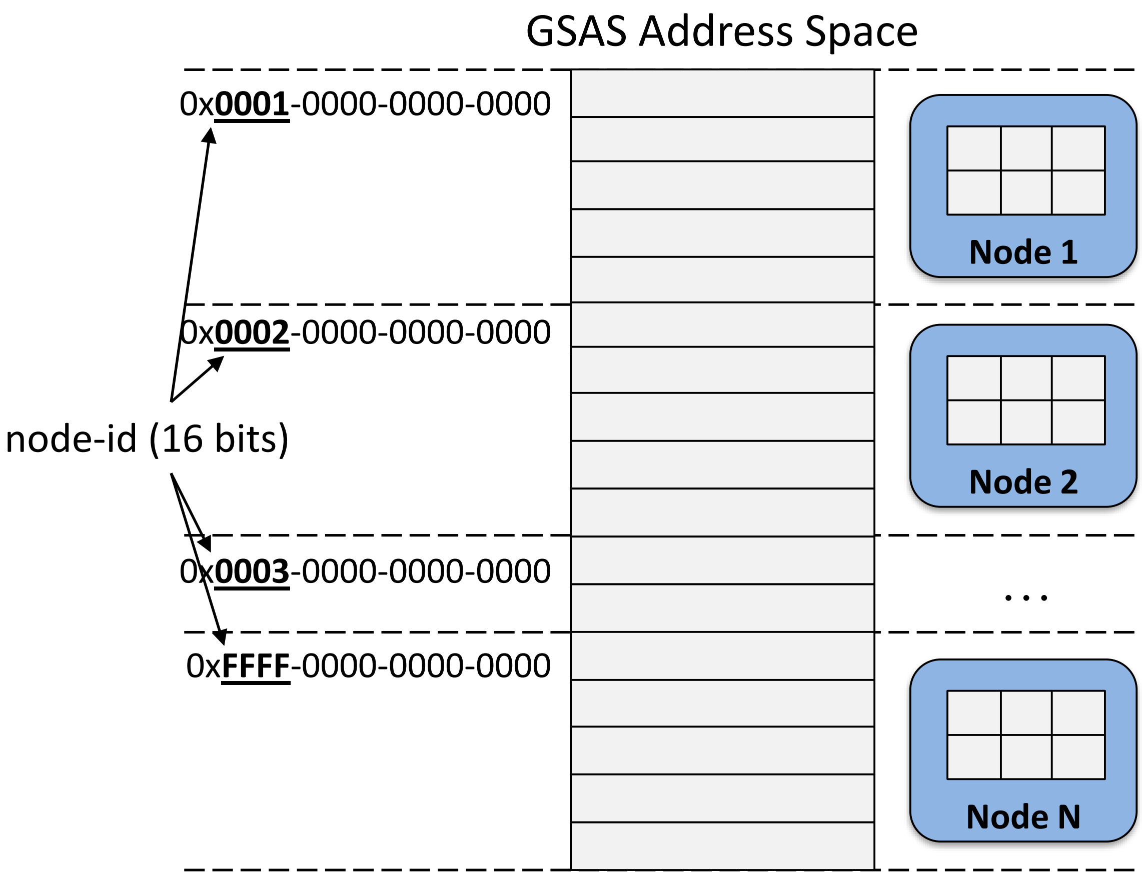
GSAS - Global Shared Address Space

1. Global Shared Address Space across system's remote nodes.
2. It is mostly implemented based on mechanisms for sending/receiving small messages atomically.
   - No complex hw-coherence protocols.
   - Flexibility.
3. API resembles to shared memory communication.
4. Applications can use this API for synchronization and for utilizing remote memory.
5. Data are cached in the node that reside on → **cacheable at single node.**
   - This is a Unimem property.

GSAS - SW & HW Stack

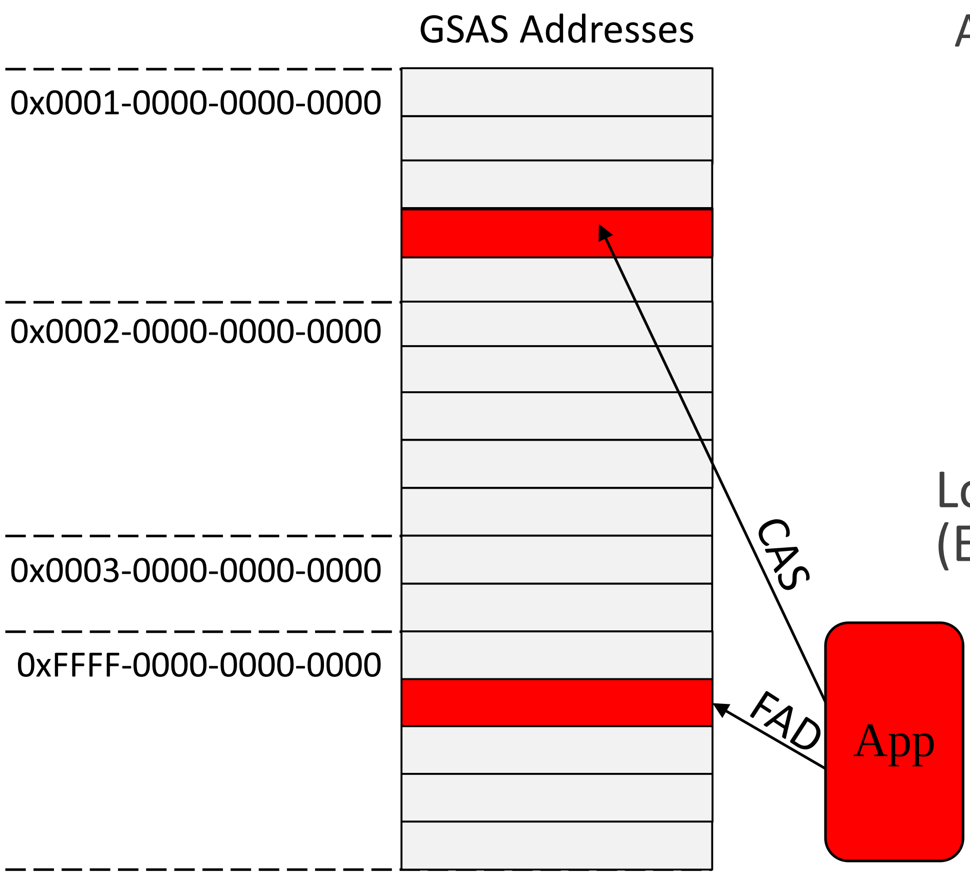


Overview of the GSAS environment



- 64-bit Global Address Space.
- The first 16 bits contain the routing information.
  - node-id.
- The remaining 48 bits indexing the memory of each node.

GSAS - Functionality



Applications that use the GSAS API are able to:

- **Allocate** of memory in any remote node.
- **Spawn** new processes on any remote node.
- Execute ***atomic operations*** (i.e., CAS, FAD, SWAP, etc.) on any remote memory location.

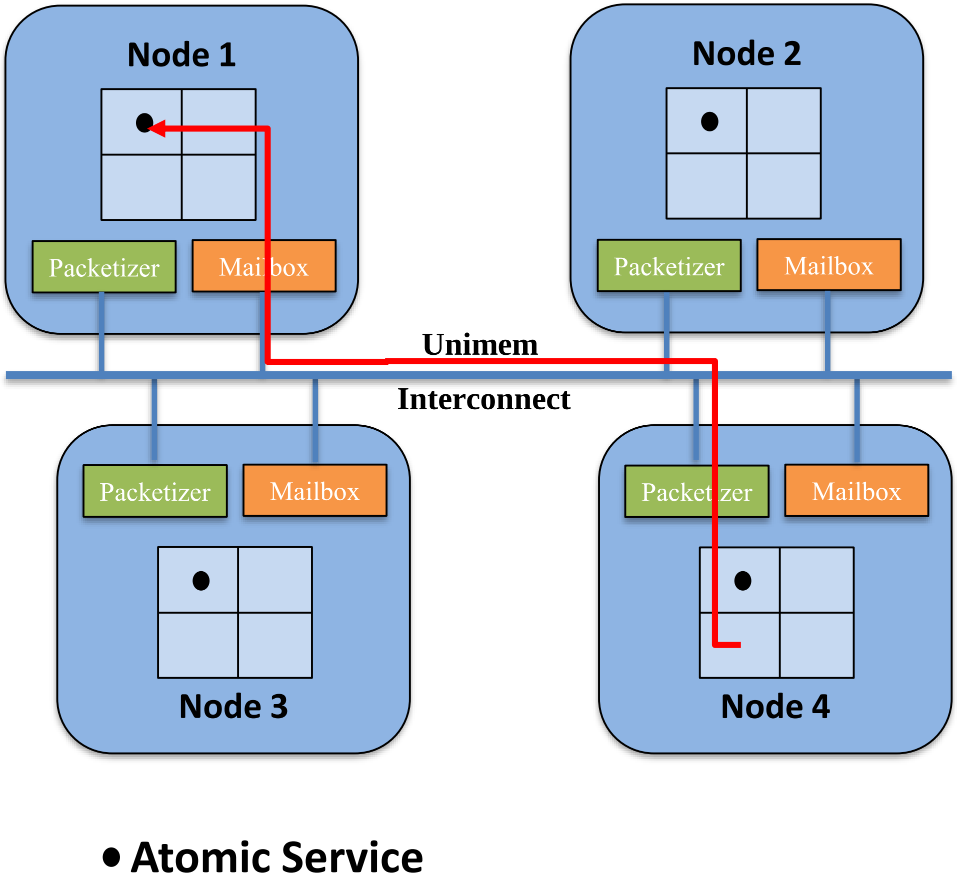
Low latency primitives (Exanet on QFDB prototype).

- ≈ 1 – 2 μsec for a remote issued atomic instruction.
- A few nsec for local issued atomic instruction.

Application Programming Interface

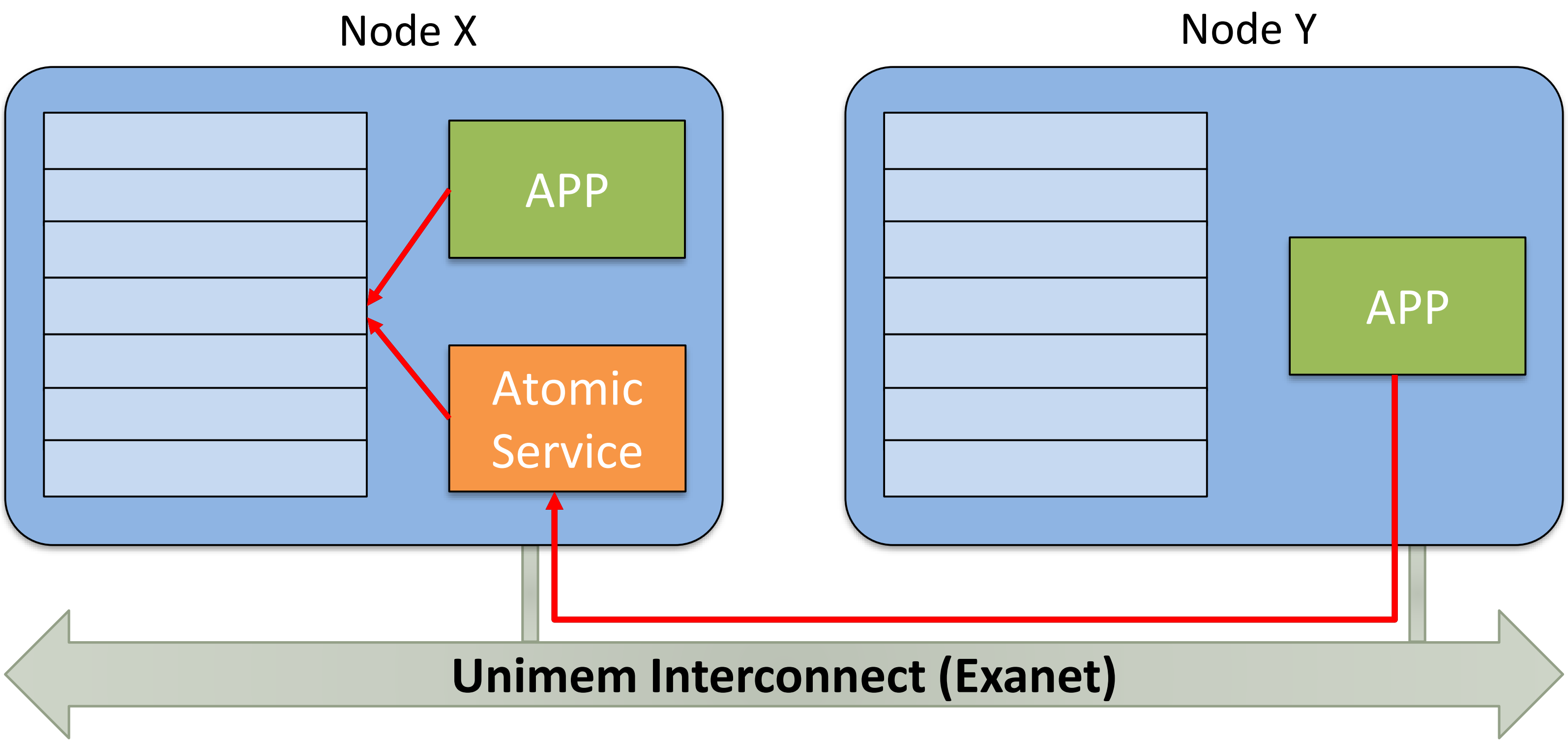
allocSharedPage	Allocation of remote/local memory
freeSharedPage	Free allocated memory
remoteFork	Spawn of a new process on some remote node
Read/Write	Read/Write operations
CAS	Compare&Swap operations
FAD	Fetch&Add operations
SWAP	SWAP operations
BarrierJoin/ BarrierDestroy	Functionality for Barriers

Overview of the GSAS environment



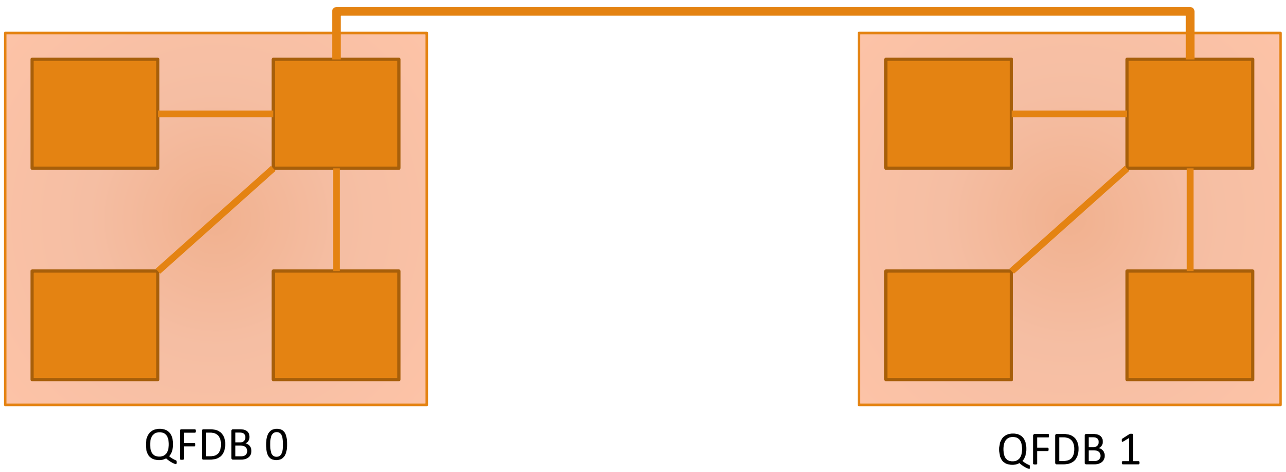
- There is an **atomic service** at each node that serves remote requests.
- Atomic service is running on core 0 on every node of the system.
- Apps and the atomic service communicate through **small atomic messages** with low latency.
- There is a **user-space library** that handles the requests on the issuer side.

GSAS – Architecture



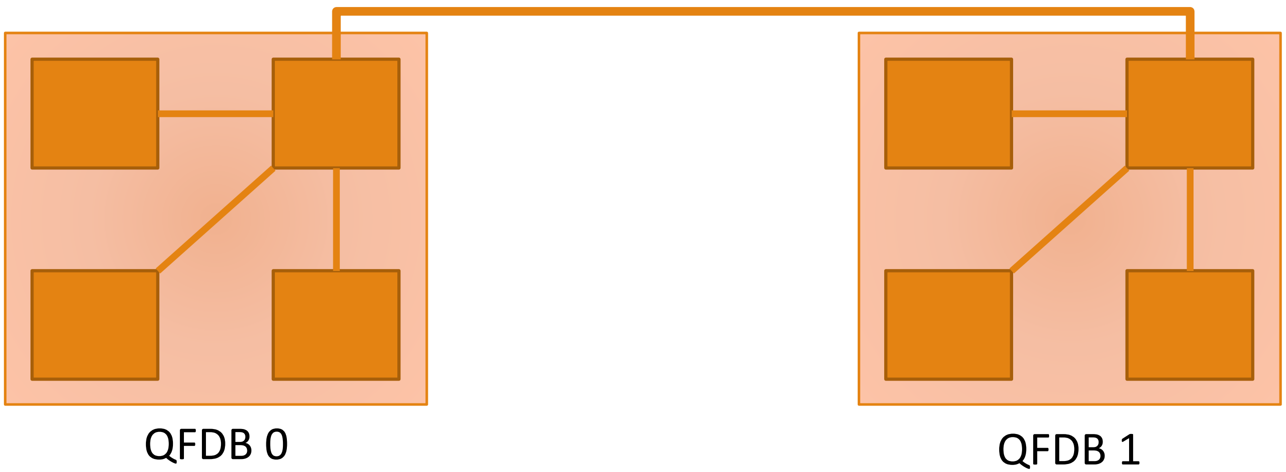
- Processes perform atomic operations on the allocated space on any node (local or remote).
  - i.e., CAS, FAD, atomic READ, etc.
- Only operations performed by remote process are applied by the atomic service ⇨ Improved Performance.

Experimental Testbed



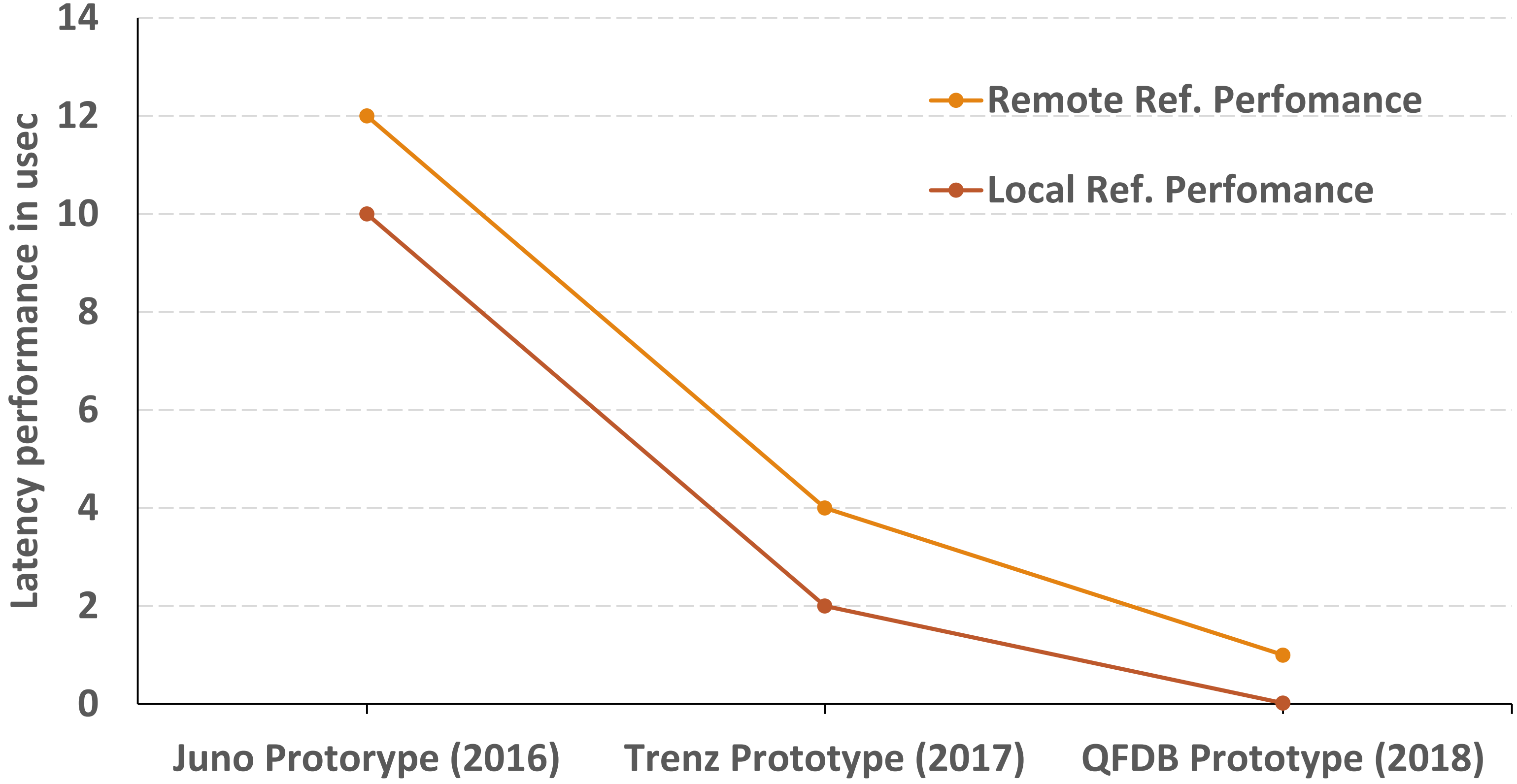
- Experiments on a Unimem testbed (2 QFDB board):
- Each board is equipped with 4 nodes, each of which:
  - Zynq MP Ultrascale+ SoC.
  - 4 Arm A53 cores @ 1.4 GHz.
  - 16 GB of local DDR4.
  - Exanet network interfaces.

Latency Microbenchmarks

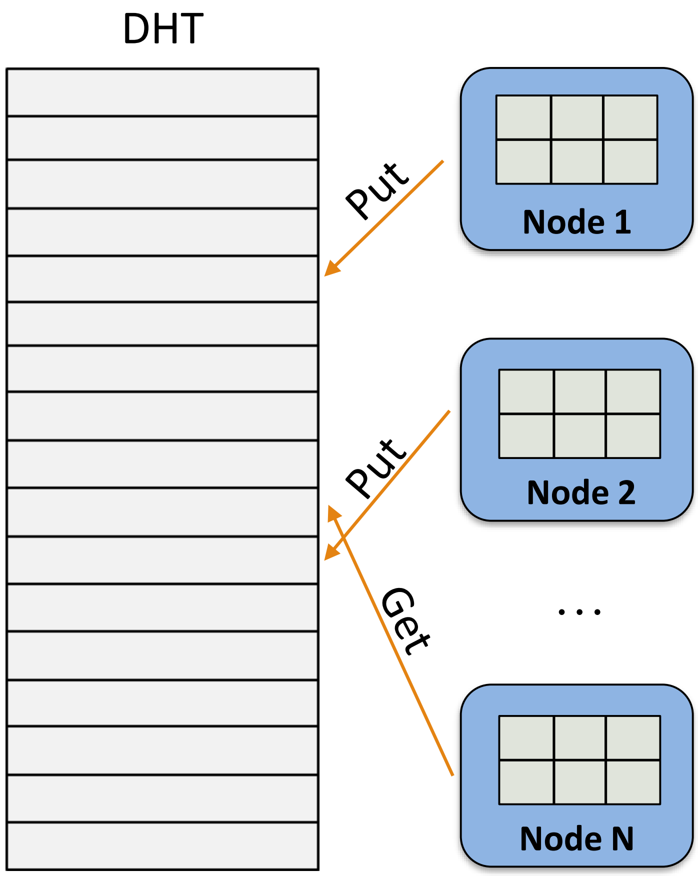


	Trenz prototype	QFDB (1 hop)	QFDB (2 hop)	Comments
GSAS Write	4.0 usec	1.0 usec	1.5 usec	64-bit write
GSAS Read	4.0 usec	1.5 usec	2.0 usec	64-bit read
GSAS Fetch&Add	4.0 usec	1.5 usec	2.0 usec	64-bit Fetch&Add
Small Message Transfer	1.9 usec	0.7 usec	1.2 usec	32 bytes - one way

GSAS Performance Evolution

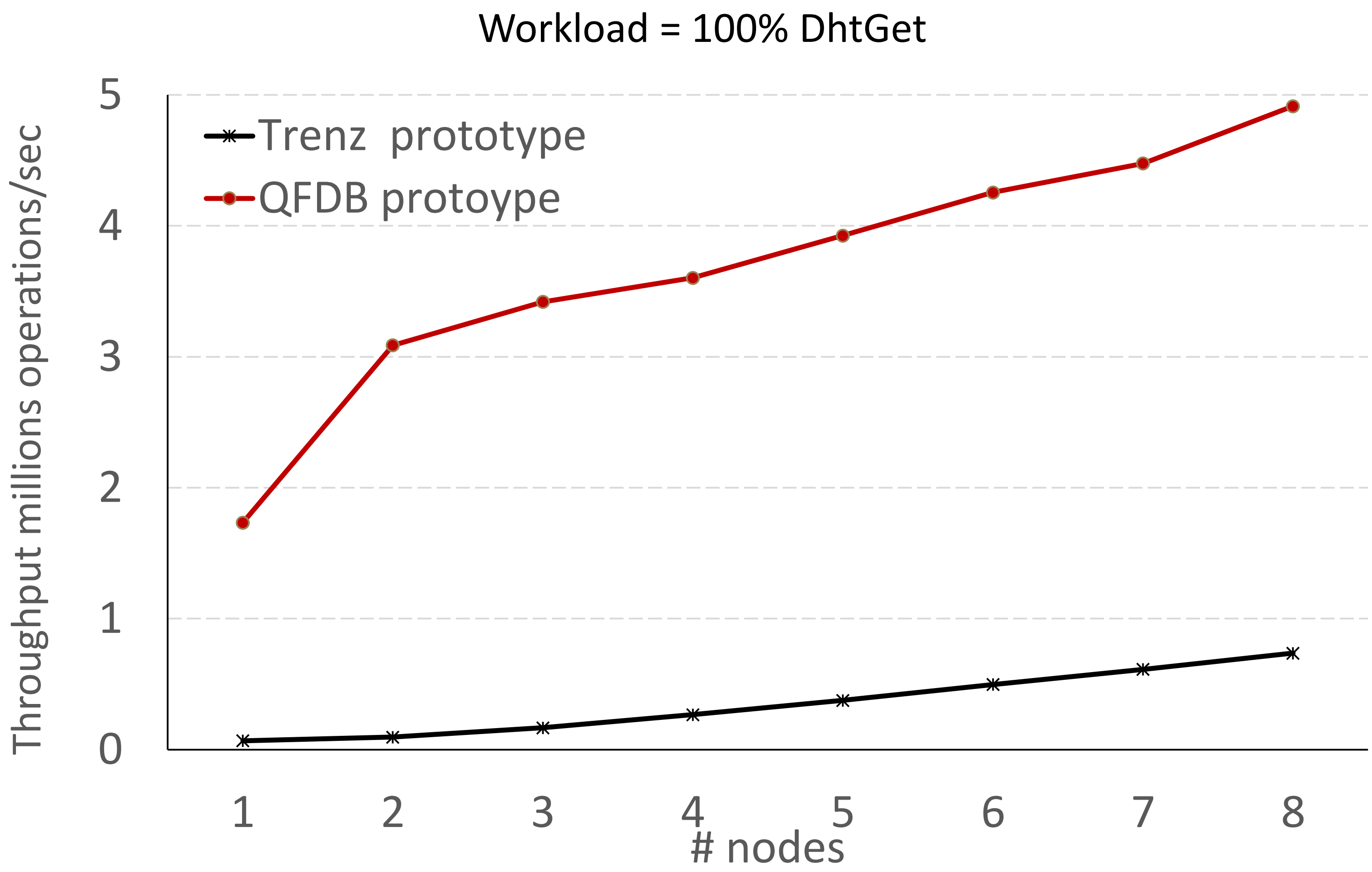


Example App: Distributed Hash Table

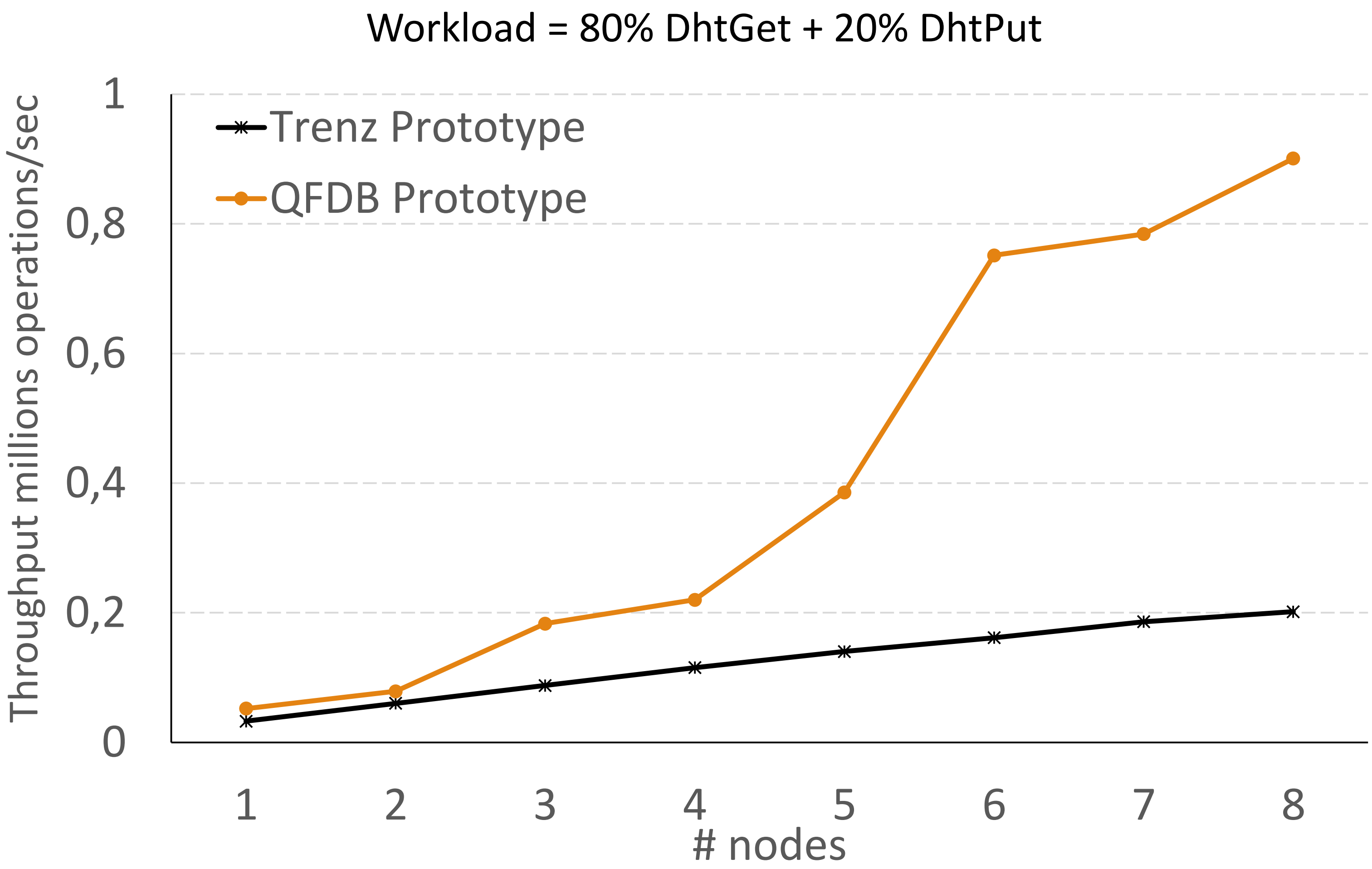


- A concurrent Distributed Hash Table is implemented on top of the GSAS environment.
- Any thread that runs on any node of the system is able to access/ modify the stored data by using ***Put*** and ***Get*** operations.
- Put and Get operations are executed concurrently.
  - There is no dedicated server for serving the requests.
- The data structure is able to use the memory of the all available cores.

DHT Performance on GSAS



DHT Performance on GSAS



Conclusions

- GSAS provides Global Shared Address Space across system's remote nodes.
- It is mostly implemented based on mechanisms for sending/receiving small atomic messages.
  - No complex hw-coherence protocols.
  - Flexibility.
- API resembles to shared memory communication.
- The latency of remote operations is about 1 – 2 usec (1 or 2 hop distance).
- A GSAS use-case example is considered, i.e. a Distributed HashTable.

Thank You