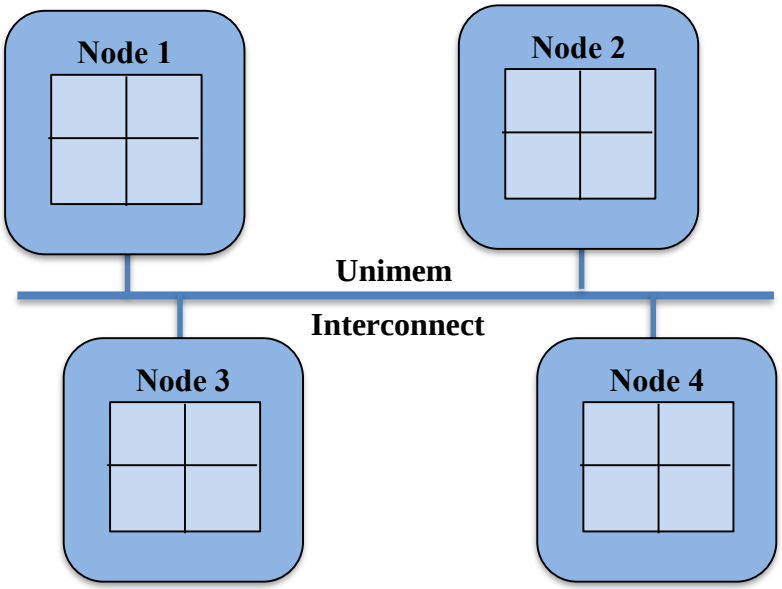
Use-cases for Remote Memory in the Unimem Architecture

Nikolaos Kallimanis Manolis Marazakis Emmanouil Skordalakis

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

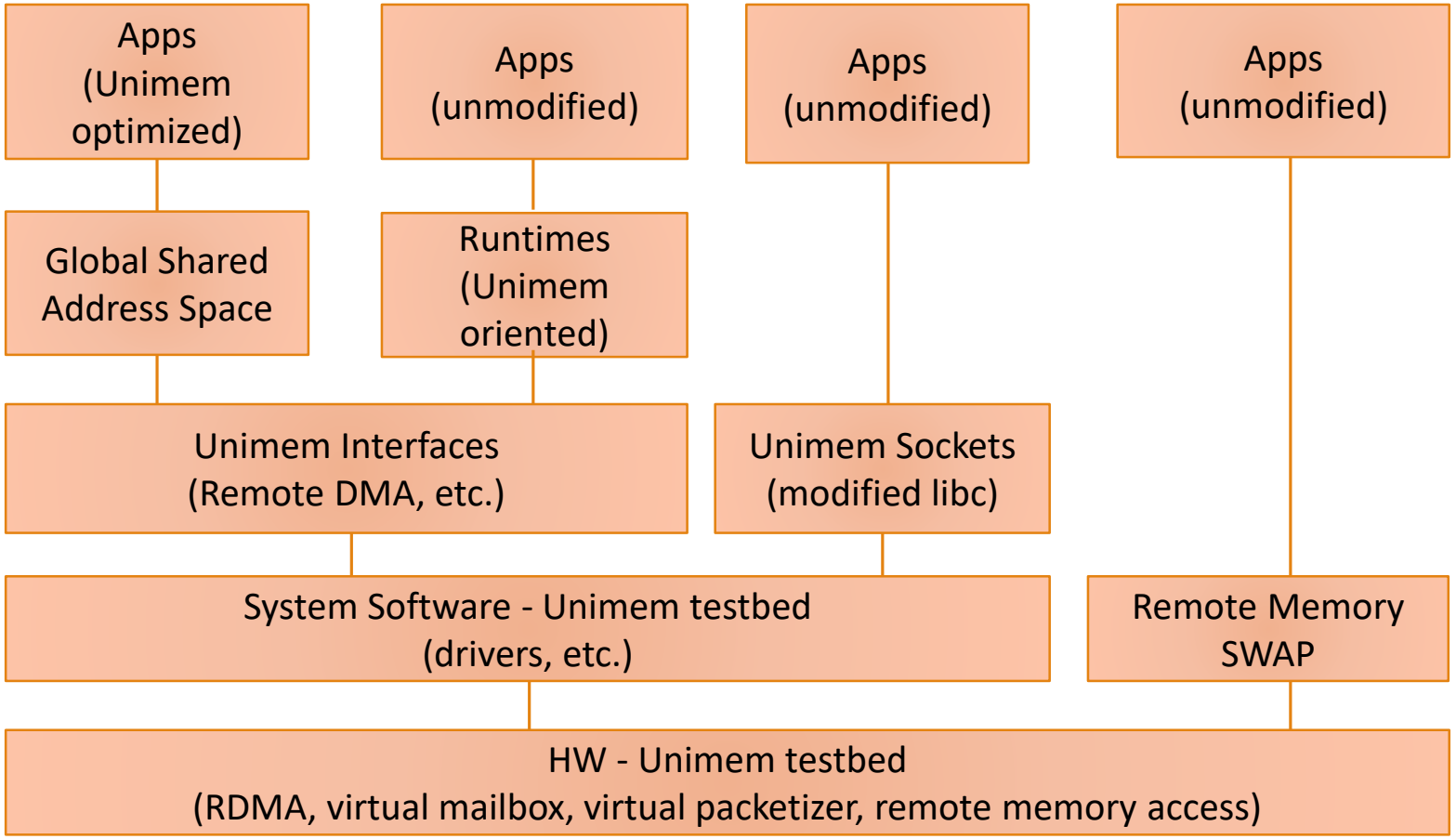
Unimem Architecture

- 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.




How to exercise the Unimem remote memory?

Unimem's APIs



Exercising Remote Memory

GSAS - Global Shared Address Space

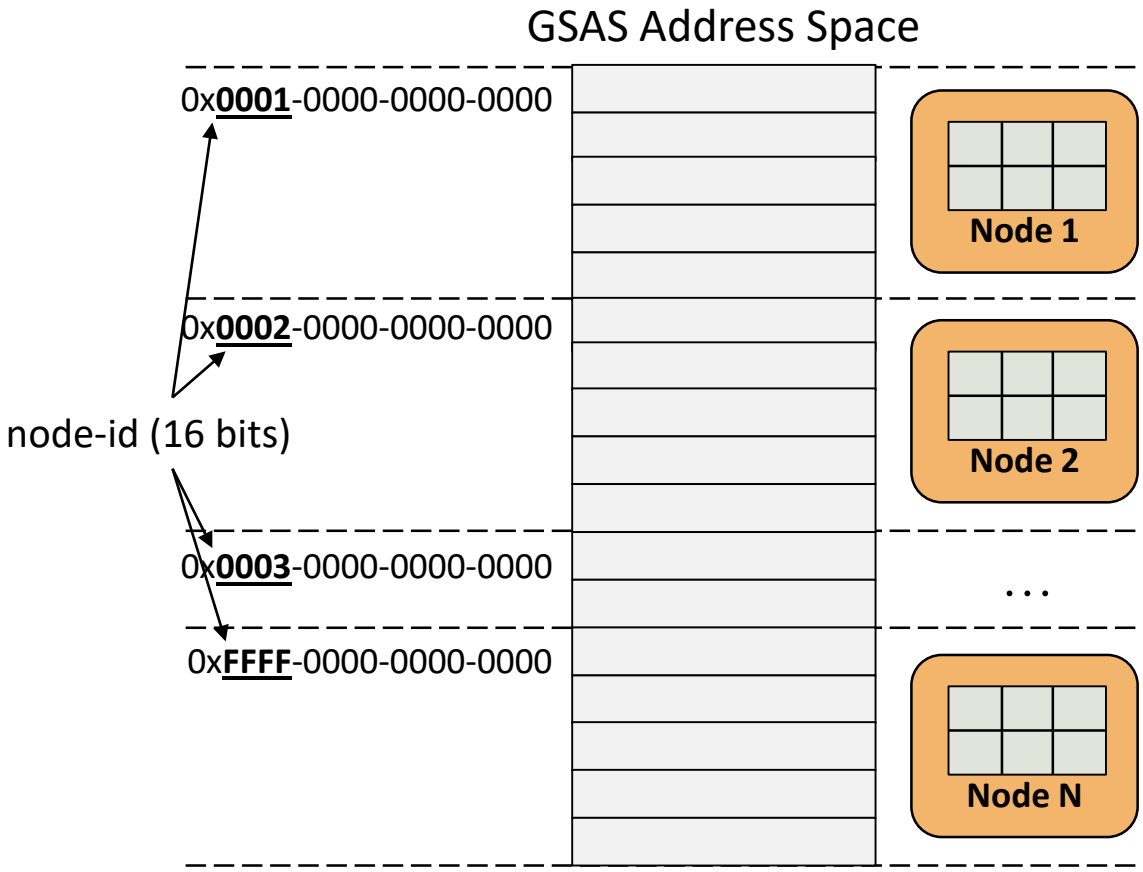
1. Global Shared Address Space across system's remote nodes.
2. It is mostly implemented based on mechanisms for sending/receiving small atomic messages.
3. API resembles to shared memory communication.
4. Applications can use this API for synchronization and for using remote memory.
5. Data are cached in the node that reside on → **cacheable at single node.**

KRAM - Remote Memory SWAP

1. It uses remote node's (unused) memory to create a SWAP.
2. It uses the Xilinx CDMA engine, or memcpy for transferring data.
3. Transparent to user application (unmodified apps).
4. Extends the memory that applications can use.
5. Data are cached in the local processor that application runs → **cacheable at single node.**

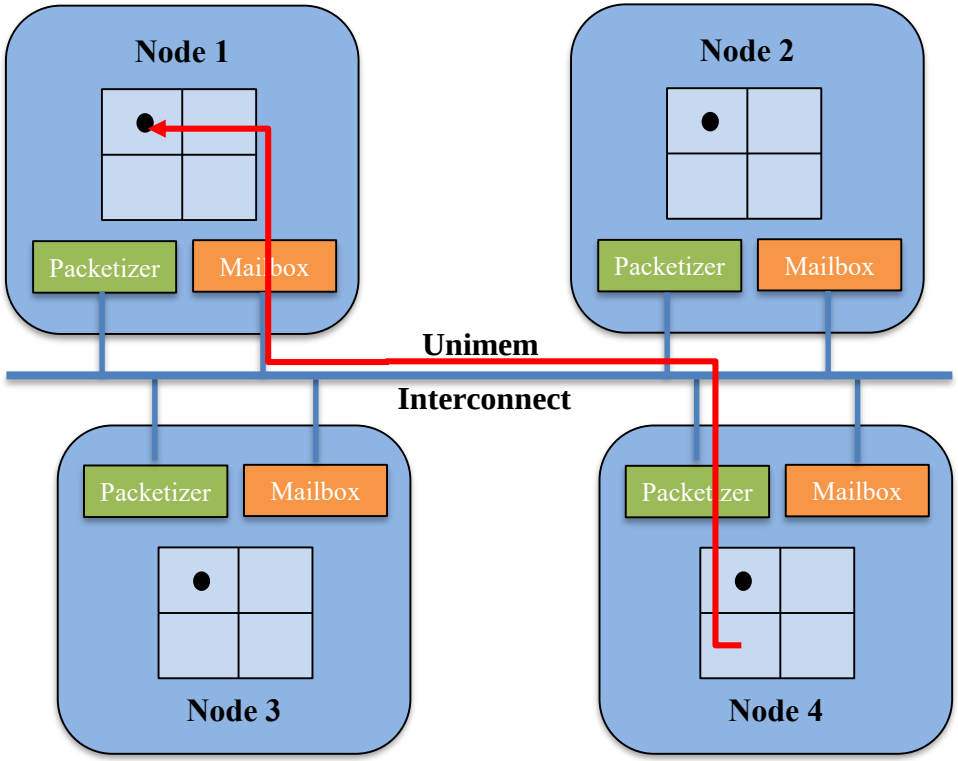
- In both cases data are cached at a single node (Unimem property).
- No complex hw-coherence protocols.
- Flexibility.

Overview of the GSAS environment



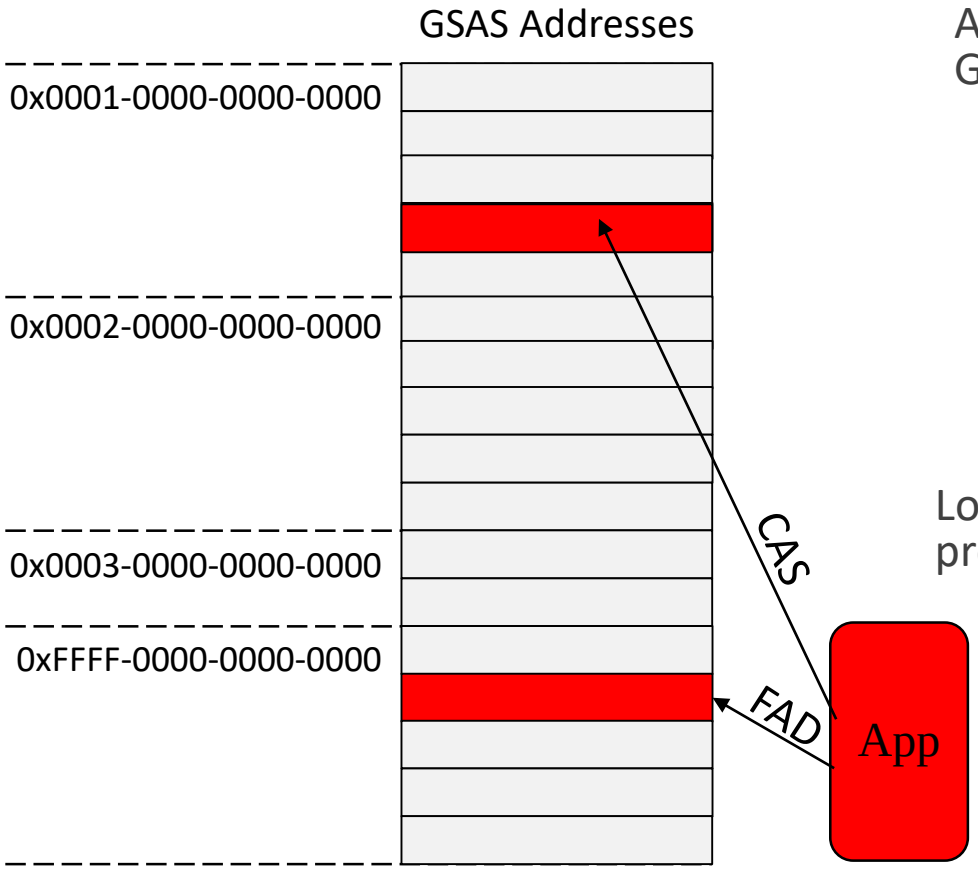
- 64-bit address space.
- The first 16 bits contain the routing information.
  - node-id.
- The remaining 48 bits indexing the memory of each node.

Overview of the GSAS environment



- There is an **atomic service** at each node that serves 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.

Overview of the GSAS environment



An application that uses the GSAS API is able to:

- ***Allocation*** of memory in any remote node.
- ***Spawning*** a new process on any remote node.
- Executing ***atomic operations*** (i.e., CAS, FAD, SWAP, etc.) on any remote memory location.

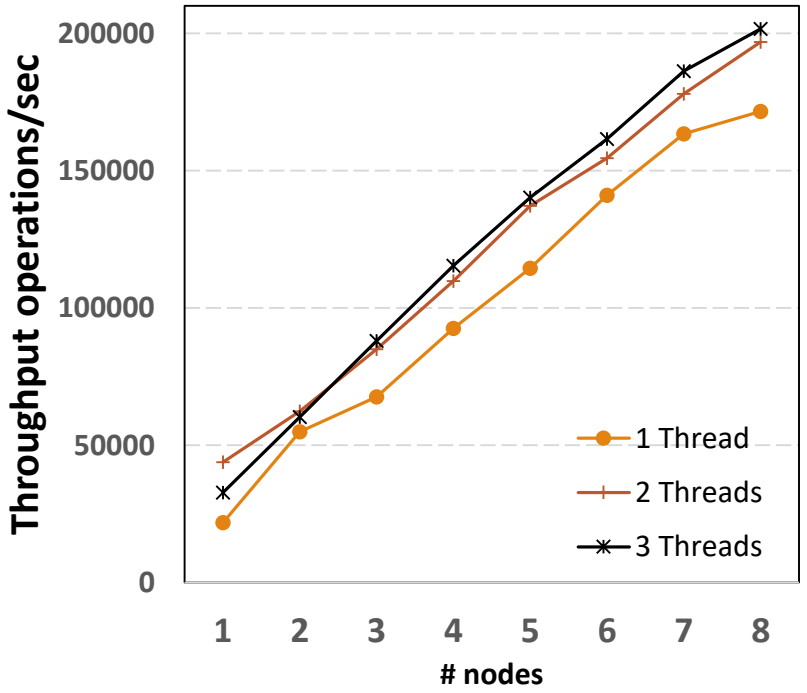
Low latency primitives (current prototype).

- ≈ 2.0 µsec for a local issued atomic instruction.
- ≈ 3.9 µsec for a remote issued atomic instruction.

Performance/DHT on GSAS

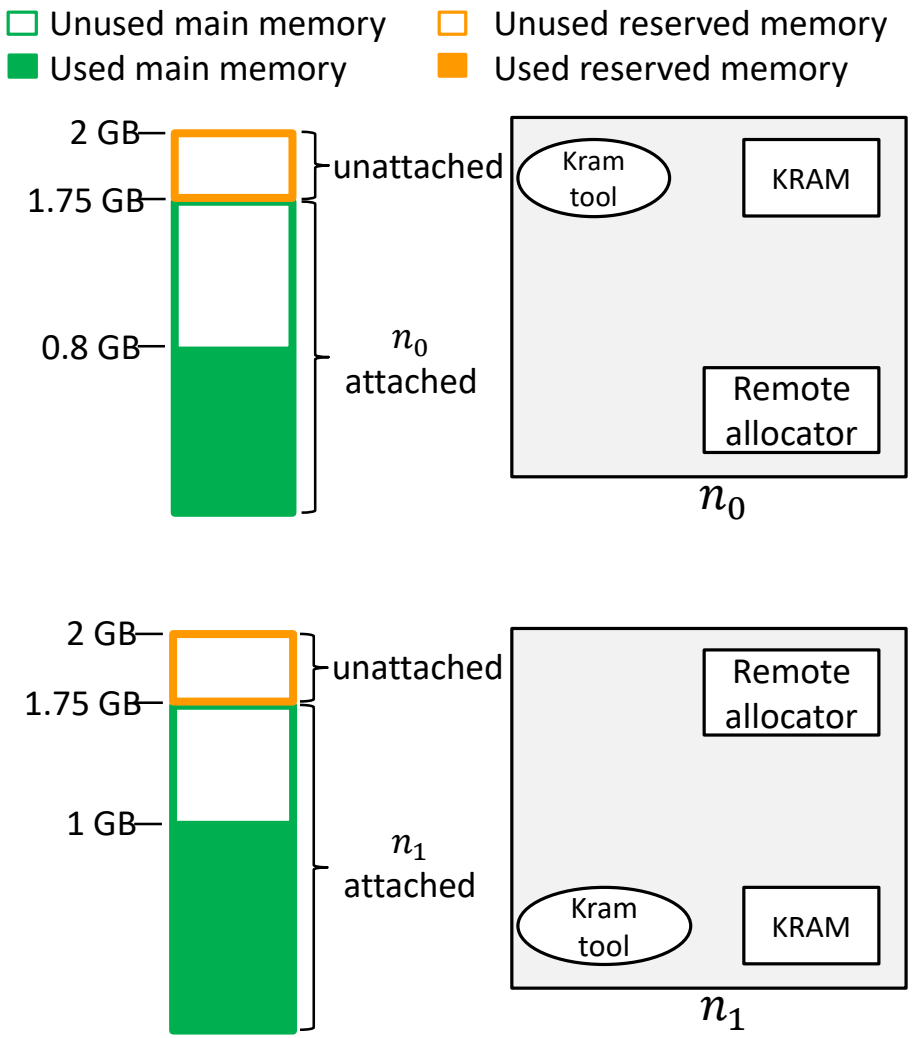
GSAS use case example:

- An in-memory **concurrent Distributed Hash Table (DHT)** is based on GSAS.
- It supports:
  - *DhtPut* → **Store pairs** of $\langle key, value \rangle$ items in GSAS address space.
  - *DhtGet* → **Retrieves** the value that corresponds to some key.
- Experiments on Unimem testbed (8 Trenz nodes):
  - Zynq MP Ultrascale+ SoC.
  - 4 Arm A53 cores & 2 GB of local DDR4.
  - Each thread executes pairs of DhtPut & DhtGet operations.
  - Throughput of operations/sec is measured for different # of nodes.
  - Experiments were performed for 1, 2 and 3 threads per node.



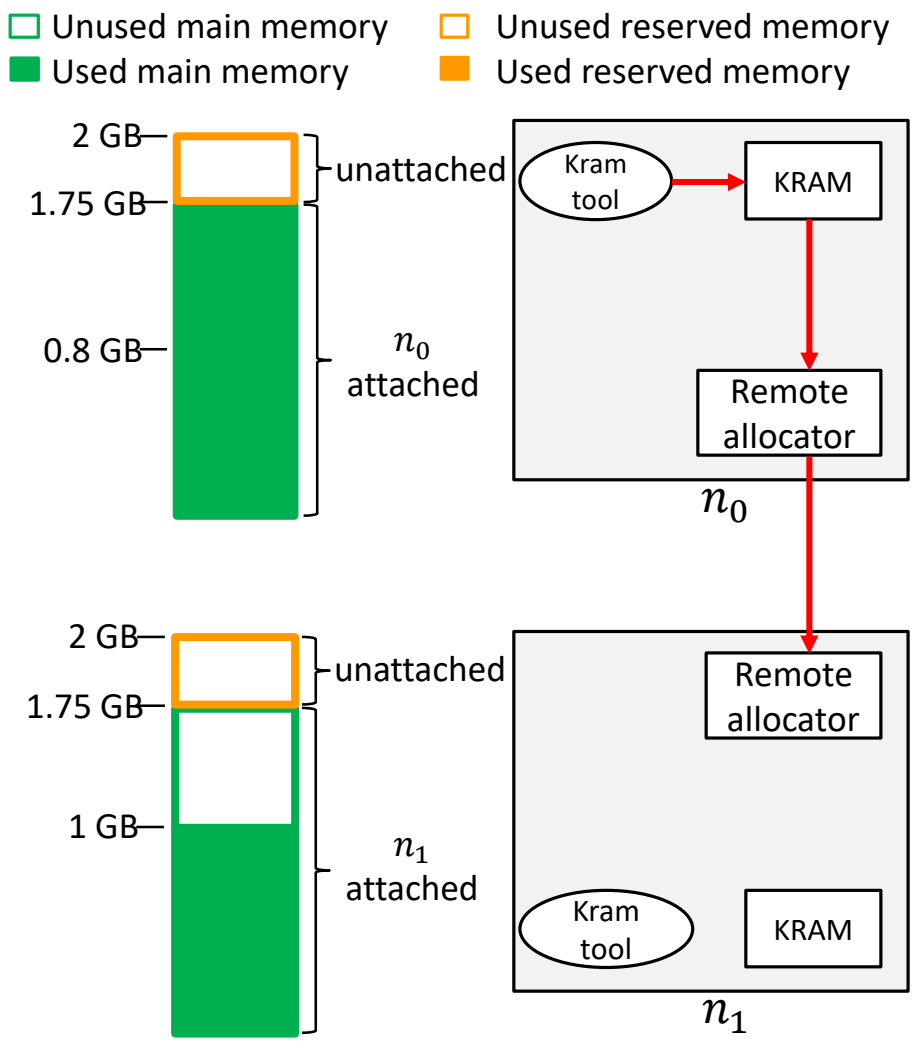
KRAM – Concept/Architecture

- 250 MB are dedicated for the remote allocator service.
- The maximum amount of local memory that local applications can use is 1.75 GB.
- For more memory, KRAM creates a swap device to extend the memory.



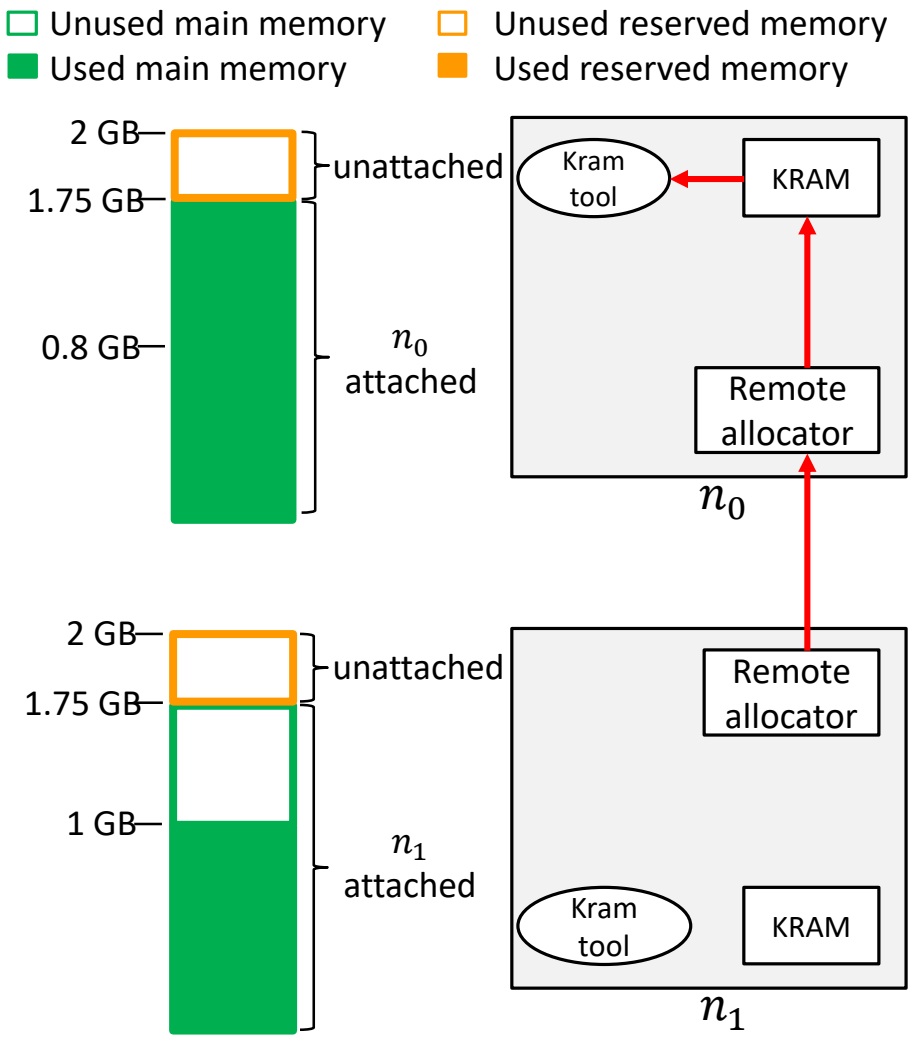
KRAM -Requesting More Memory

1. User requests a swap device of a specific size.
2. KRAM requests memory from the local remote allocator service.
3. The remote allocator communicates with neighbor allocators for more memory.



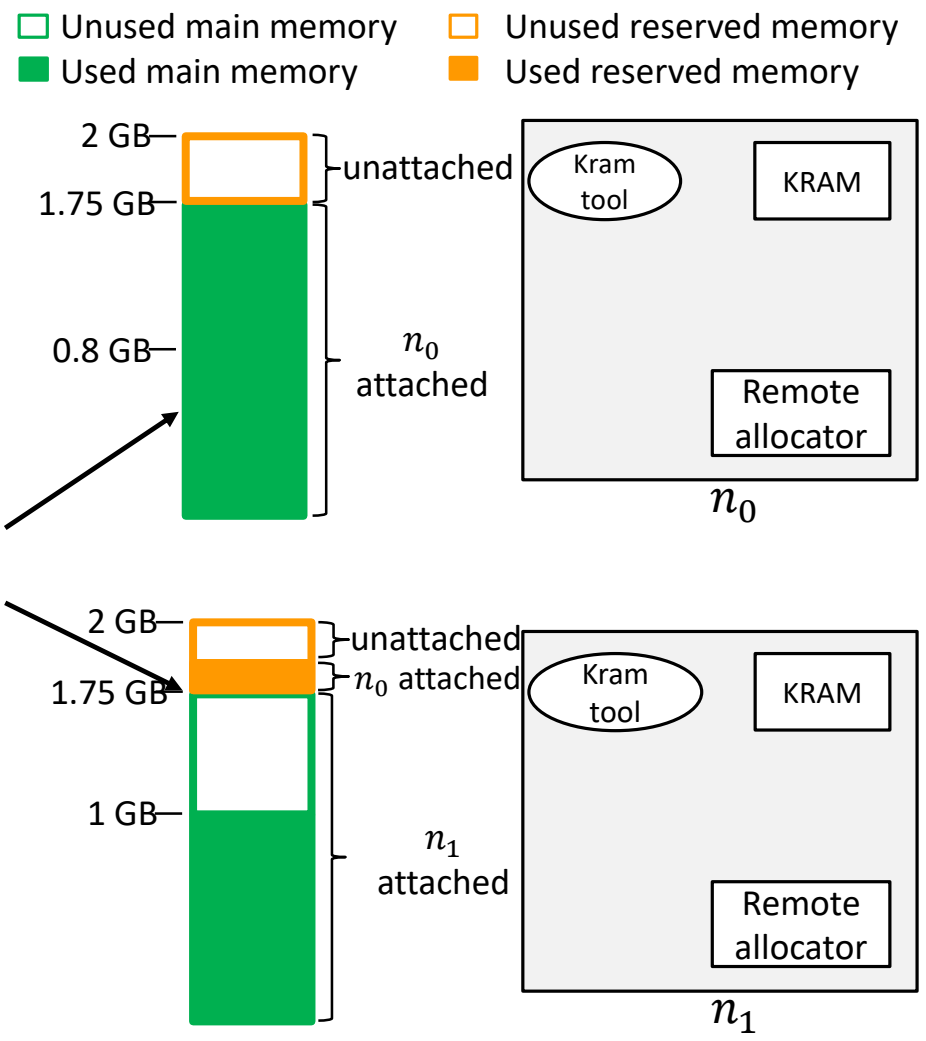
KRAM – Allocating Remote Memory

4. The remote allocator of some neighbor sends the physical address of the free remote memory.
5. The remote allocator sends the address to KRAM.
6. KRAM creates a swap device.

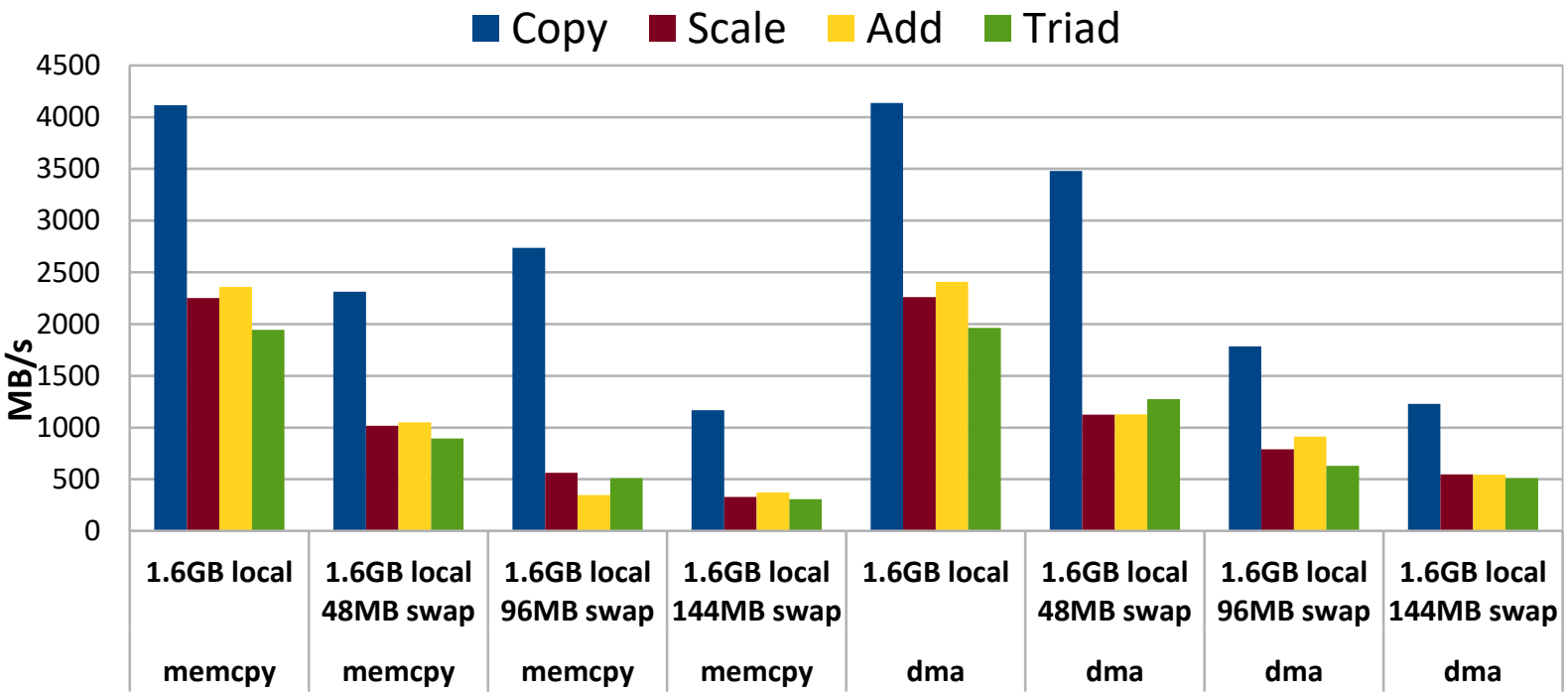


KRAM – Swapping

- Now, Applications on node n_0 are able to use:
  - 1.75GB from local memory
  - Some memory from remotes



KRAM – Stream Performance



- 3 nodes with specifications:
  - 2 GB RAM (1.75 GB main, 0.25GB reserved)
  - Zynq MP Ultrascale+ SoC
- The Stream benchmark which calculates MB/s using the Copy, Scale, Add and Triad functions.

- 6 runs were performed, with **dma mode** and with **memcpy mode**
  1. Local memory only.
  2. Remote memory of 48 MB.
  3. Remote memory of 96 MB.
  4. Remote memory of 144 MB.

Thank You