Hewlett Packard
Enterprise

A RECORD BASED POINTER TO FAM

C, Amitha
2nd June, 2023

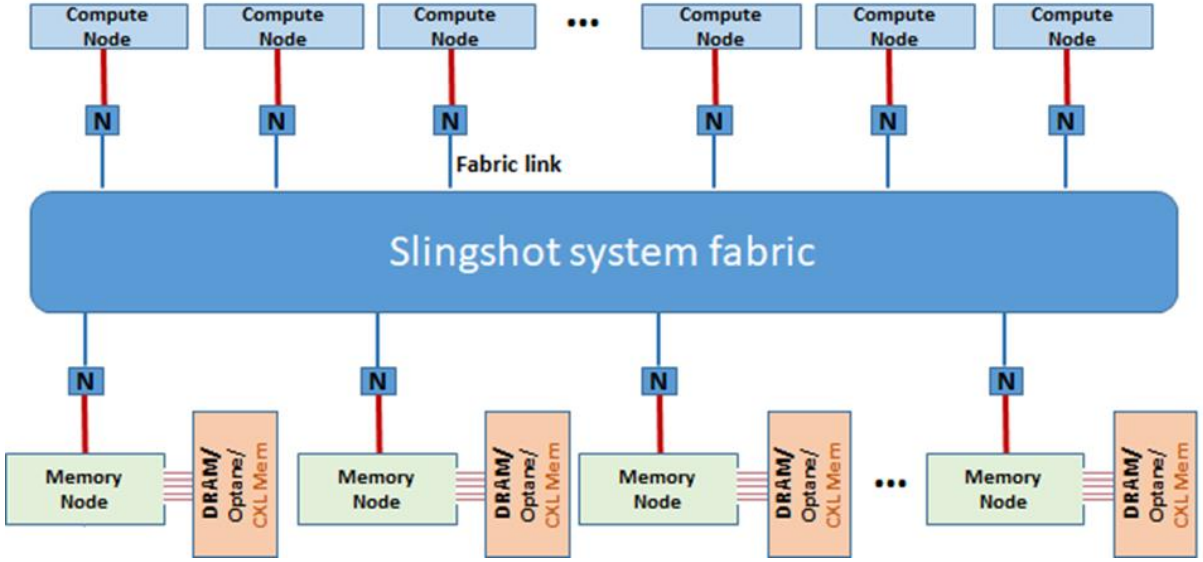
Co-Authors:
Clarete Riana Crasta
Brad Chamberlain
Sharad Singhal
Dave Emberson
Porno Shome

AGENDA

- Fabric-Attached Memory(FAM) – Context
- Why Chapel?
- FAM enablement in Chapel
  - Our previous work on FAM access from Chapel
  - Pointer to FAM – Design
  - Example use-case of FAM pointer
- Status and Next Steps

FABRIC-ATTACHED MEMORY(FAM)

- Converging memory and storage
  - Resource disaggregation leads to high capacity shared memory pool
  - Local volatile memory provides lower latency, high performance tier
- Distributed heterogeneous compute resources
  - High-speed interconnect
  - Operating system instance per compute node
- Fabric Attached Memory is
  - Large – enabling workloads with large data sets
  - Shared – enabling communication across compute nodes through FAM



CHAPEL

Our Goal:

Enable FAM access through multiple programming languages to make FAM available for a variety of workloads.

FAM enablement in Chapel, because Chapel is :

- **written for HPC**
- **scalable**: Designed to be as scalable as MPI & OpenMP parallel computing
- **fast:** performance competes with or beats C/C++ & MPI & OpenMP
- **portable:** runs on laptops, clusters, the cloud, and HPC systems
- **Programmable**: Designed with programmer productivity in mind
- **open source:** hosted on GitHub, permissively licensed

Guiding Philosophy

- Access FAM-resident data with minimal language changes
- Abstraction of FAM access from the application

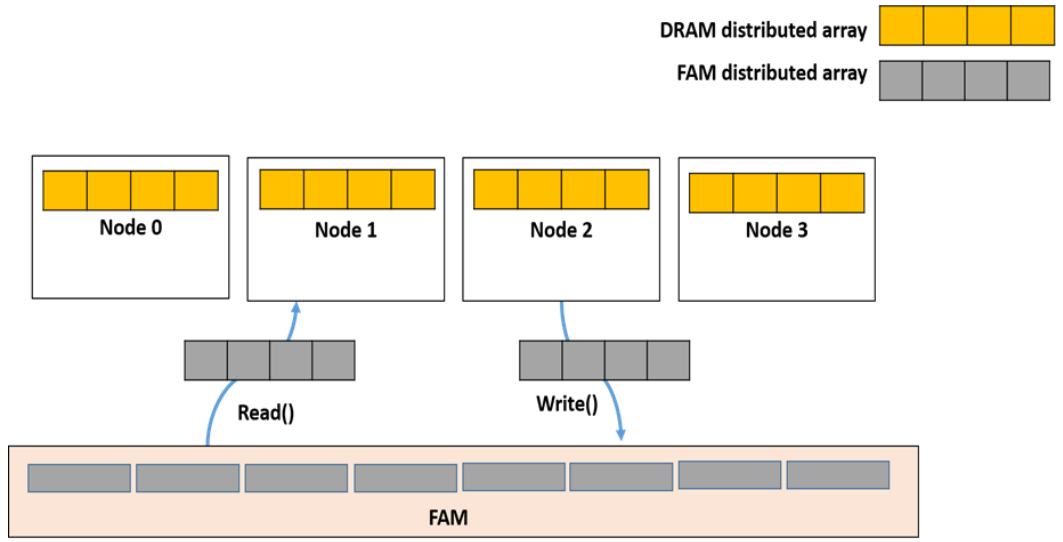
FAM ENABLEMENT IN CHAPEL – PREVIOUS WORK

Chapel bindings for FAM

- Enable calls to low-level FAM APIs from the application
- Developed thin C wrapper over OpenFAM C++ library
- Chapel module called “OpenFAM” under /modules/packages
- Provides no abstraction

FAM distributed arrays

- Array resides on FAM
- Supports implicit parallelism through domain partitioning
- Converts high level array operations into FAM-specific accesses underneath
- Abstracts away FAM access details from the application



FAM ACCESS THROUGH POINTER TYPE

Problem Solved

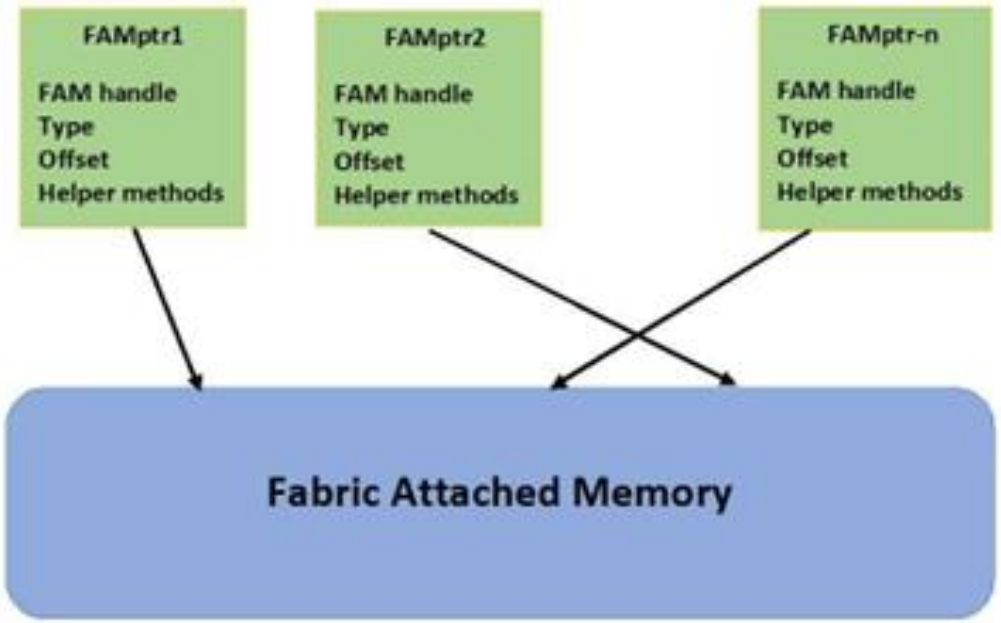
- Extend the support for FAM access using language construct
- Enable data-structures like list , map etc for FAM

Proposed Solution

- Enable support for pointer to FAM (FAMptr)
- Use OpenFAM library for accessing FAM
- Support pointer arithmetic
- Provides abstraction with no performance overhead

High Level Design:

- Points to pre-existing data residing in FAM
- FAMptr includes handle to the FAM data location and details like data type, offset, size etc.
- Access to FAMptr internally gets translated to OpenFAM APIs
- Allocation and destroy of OpenFAM region and data items are done through Chapel binding APIs



FAM POINTER TYPE - STATUS

Current Status

- Developed new module - FAMTypes
- Creation and deletion of FAMptr
- Read, write to FAM location through FAMptr
- Increment and decrement FAMptr
- Pointer arithmetic operations on FAMptr

In progress

- Atomic access to FAM through FAMptr
- Explore different use-cases for FAMptr

Next Steps

- Enhance the FAMptr implementation and support

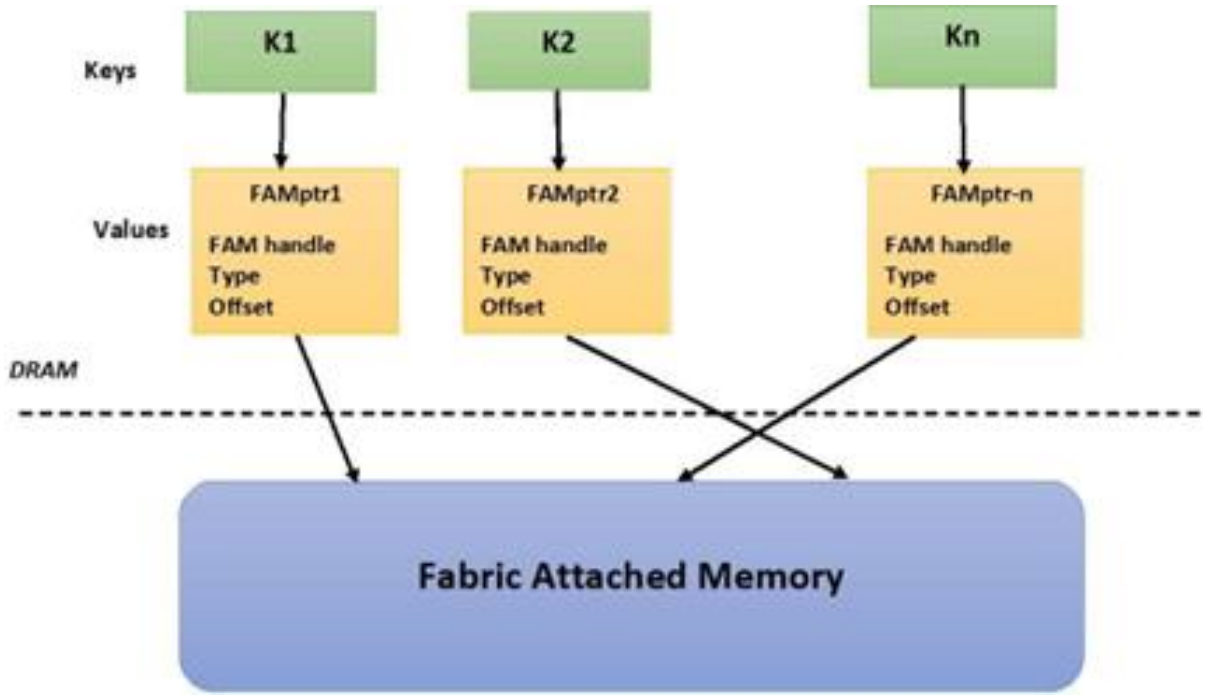
```
1    use FAMTypes;                      // new module for FAMptr definition

     // use bindings to allocate/lookup OpenFAM regions and data items
      . . .
     // declare pointer to integer on FAM represented by FAM descriptor fd
50   var ptr1= new FAMptr(int, fd);
51   var ptr2=ptr1;               // ptr2 and ptr1 pointing to same FAM location
      . . .
65   ptr1.write(10);                         // write to FAM using ptr1
66   writeln("1st elem =", ptr2.read());   // read from FAM using ptr2
     . . .
80   ptr2.increment();               // point to next int
81   var ptr3 = ptr1+1;              // pointer arithmetic
      . . .
112   var ptr4= new FAMptr(int)        // ptr4  pointing to nothing initially
       . . .
```

Examples of FAM access using FAMptr from Chapel

EXAMPLE USE-CASE FOR FAMPTR- KVS

Key-value pairs for FAM

- Key resides on DRAM while actual payload on FAM
- Each key is associated with a value which is a FAMptr
- Any query or update to the user data through key-value pair is internally translated into FAM data access



Contact Details

amitha.c@hpe.com

ACKNOWLEDGMENT

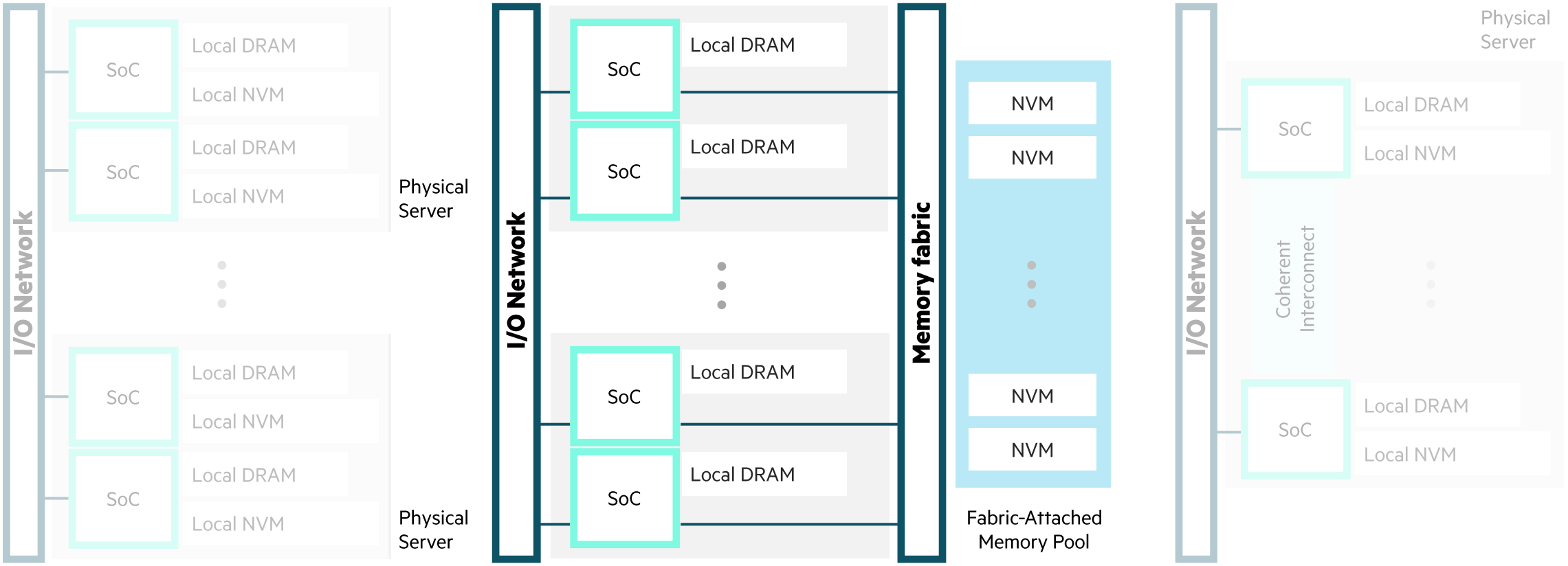
We would like to thank Harumi Kuno for reviewing and providing valuable suggestion on this work. We also thank current and past FAM hardware and software development team members for all the work on Fabric Attached Memory.

REFERENCES

1. Peng, R. Pearce, and M. Gokhale, “On the Memory Underutilization: Exploring Disaggregated Memory on HPC Systems,” in 2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), Sep. 2020, pp. 183–190. doi: 10.1109/SBAC-PAD49847.2020.00034
2. Chapel project home page: https://chapel-lang.org/
3. ‘Chapel’, Bradford L. Chamberlain, Programming Models for Parallel Computing, edited by Pavan Balaji, published by MIT Press, November 2015.
4. Chapel: Standard Layouts and Distributions: https://chapel-lang.org/docs/modules/layoutdist.html
5. B. L. Chamberlain, S. J. Deitz, D. Iten, S-E Choi, “User-Defined Distributions and Layouts in Chapel: Philosophy and Framework” in 2nd USENIX Workshop on Hot Topics in Parallelism (HotPAR’10), June 2010, Berkeley, CA, https://www.usenix.org/legacy/events/hotpar10/tech/full_papers/Chamberlain.pdf
6. https://github.com/OpenFAM/OpenFAM
7. K. Keeton, S. Singhal, and M. Raymond, “The OpenFAM API: A Programming Model for Disaggregated Persistent Memory,” in OpenSHMEM and Related Technologies. OpenSHMEM in the Era of Extreme Heterogeneity, Cham, 2019, pp. 70–89. doi: 10.1007/978-3-030-04918-8_5.
8. “DAOS and Intel® OptaneTM Technology for High-Performance Storage,” Intel. https://www.intel.com/content/www/us/en/high-performance-computing/daos-high-performance-storage-brief.html (accessed Aug. 27, 2020).
9. C, Amitha et al., “Extending Chapel to Support Fabric Attached Memory”, Presented at CUG 2022, Monterey, California, May. 2022. https://chapel-lang.org/papers/CUG_2022.pdf
10. https://memcached.org/
11. https://www.computeexpresslink.org/download-the-specification

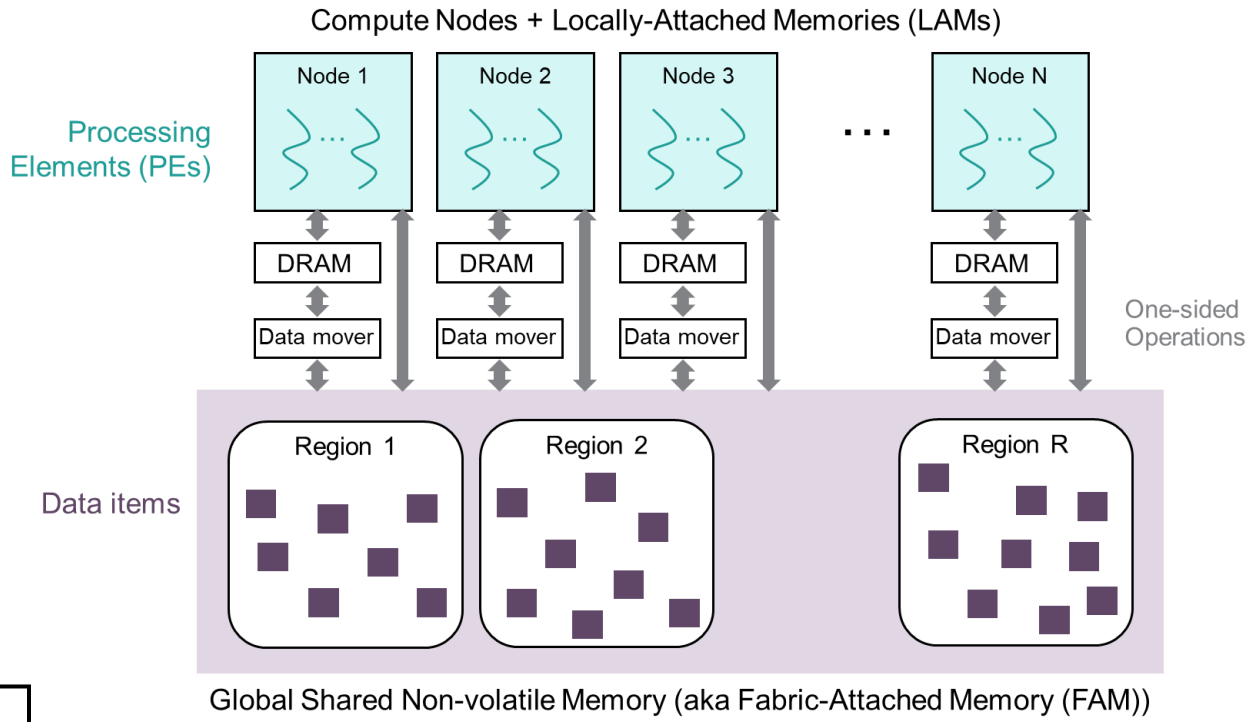
THANK YOU

FABRIC-ATTACHED (DISAGGREGATED) MEMORY IN CONTEXT



OPENFAM

- **Purpose**:
  - Develop an API and reference implementation to enable programmers to easily program FAM.
- **Challenges**
  - API should be “natural” to HPC programmers.
  - Usable across scale-up machines, existing scale-out clusters, and emerging FAM architectures.



More detail available from

Keeton K., Singhal S., Raymond M. (2019) *The OpenFAM API: A Programming Model for Disaggregated Persistent Memory*. In: Pophale S., Imam N., Aderholdt F., Gorentla Venkata M. (eds) OpenSHMEM and Related Technologies. OpenSHMEM in the Era of Extreme Heterogeneity. OpenSHMEM 2018. Lecture Notes in Computer Science, vol 11283. Springer, Cham

Open source reference implementation: https://github.com/OpenFAM

Status:

- Reference implementation is available
  - Omnipath and Infiniband clusters
- Currently we are
  - Optimizing the implementation
  - Adapting it for slingshot