



**Hewlett Packard**  
Enterprise

# **ENABLING FAM ACCESS IN CHAPEL**



Clarete Riana Crasta, Master Technologist

April 14th, 2022

# AGENDA

---

- Fabric-Attached Memory(FAM) – Context
- Chapel
- FAM access from Chapel
  - FAM Distributed Arrays – Design
  - Multi-Dimensional arrays
- Status and Next Steps
- Competitive approaches



# MORE DATA SOURCES AND MORE DATA

## Record

**40 petabytes**

200B rows of recent transactions for Walmart's analytic database (2017)

## Engage

**4 petabytes a day**

Posted daily by Facebook's 2 billion users (2017)

2MB per active user

## Act

**40,000 petabytes a day\***

4TB daily per self-driving car  
10M connected cars by 2020

Front camera  
**20MB / sec**

Front ultrasonic sensors  
**10kB / sec**

Infrared camera  
**20MB / sec**

Front radar sensors  
**100kB / sec**

Side ultrasonic sensors  
**100kB / sec**

Front, rear and top-view cameras  
**40MB / sec**

Crash sensors  
**100kB / sec**

Rear ultrasonic cameras  
**100kB / sec**

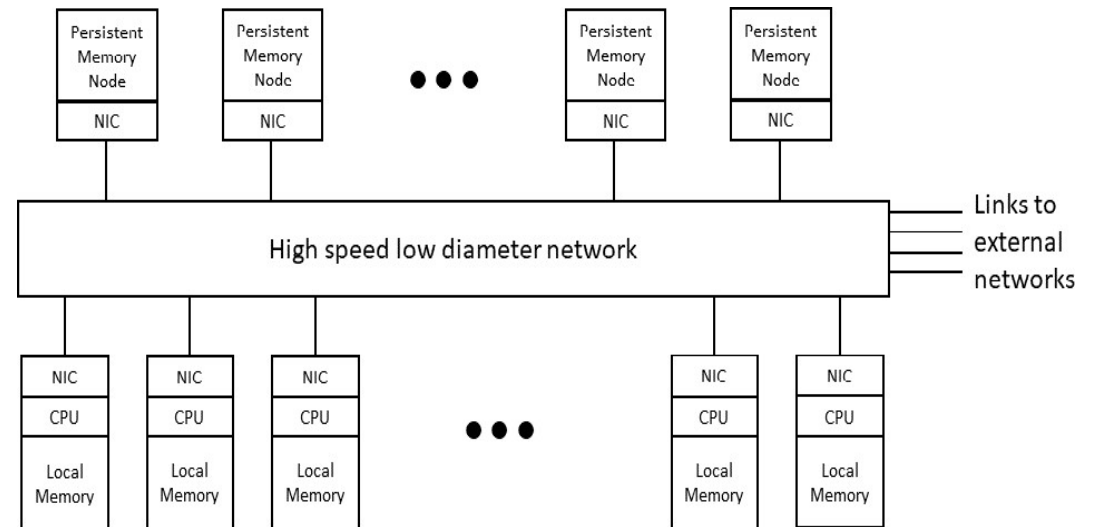
Rear radar sensors  
**100kB / sec**

\* Driver assistance systems only

Walmart's transaction database of 40 petabytes - Forbes article, Jan. 23, 2017 <https://www.forbes.com/sites/bernardmarr/2017/01/23/really-big-data-at-walmart-real-time-insights-from-their-40-petabyte-data-cloud/#524df6af6c10>  
 Facebook: Brandwatch, Mar. 5, 2018 <https://www.brandwatch.com/blog/47-facebook-statistics/>  
 Number of self-driving cars : Business Insider, June 2016 <https://www.businessinsider.com/report-10-million-self-driving-cars-will-be-on-the-road-by-2020-2015-5-6>  
 Car stats : Intel newsroom <https://newsroom.intel.com/editorials/krzanich-the-future-of-automated-driving/>

# FABRIC-ATTACHED (PERSISTENT) MEMORY

- 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, low-latency interconnect
- Software
  - Memory-speed persistence
  - Direct access to all fabric-attached NVM across the fabric
  - Non-coherent accesses across compute nodes
  - Operating system instance per compute node



# CHAPEL

---

Chapel simplifies parallel programming through elegant support for:

- **distributed arrays** that can leverage thousands of nodes' memories and cores
- **a global namespace** supporting direct access to local or remote variables
- **data parallelism** to efficiently use the cores of a laptop, cluster, or supercomputer
- **task parallelism** to create concurrency within a node or across the system

Chapel also supports

- multithreaded parallel programming model
- abstractions for data distribution and placement of computation closer to the data

Ref: <https://chapel-lang.org/>



# CHAPEL

---

## Our Goal:

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

Start with FAM enablement in Chapel, because Chapel is :

- **written for HPC**
- **scalable**: runs on laptops, clusters, the cloud, and HPC systems
- **fast**: performance competes with or beats C/C++ & MPI & OpenMP
- **portable**: compiles and runs in virtually any \*nix environment
- **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 ACCESS FROM CHAPEL

## Distributed Arrays

- An important aspect of large-scale programming on HPC clusters
- Chapel distributes the elements of the array across nodes, and so the tasks associated with the elements
- Array distributions provide a “global view” as if it was a local array

## Proposed Solution

- New distribution module - Array resides on FAM
- Modified the Chapel external module to define new distribution policy for FAM
- Provides support for named array allocation in the application
- Supports implicit parallelism through domain partitioning

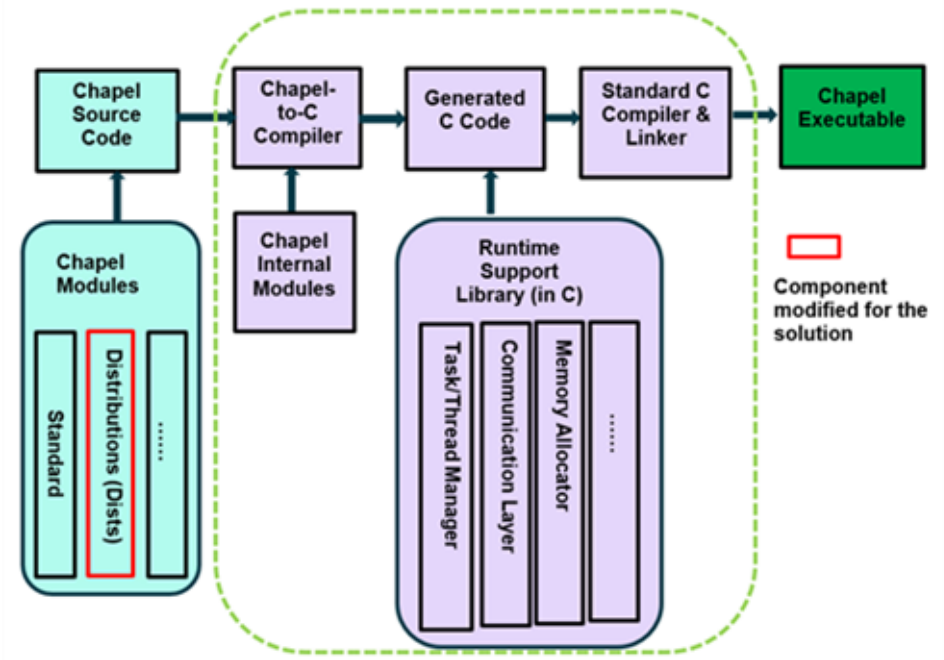


Figure 1: Chapel components



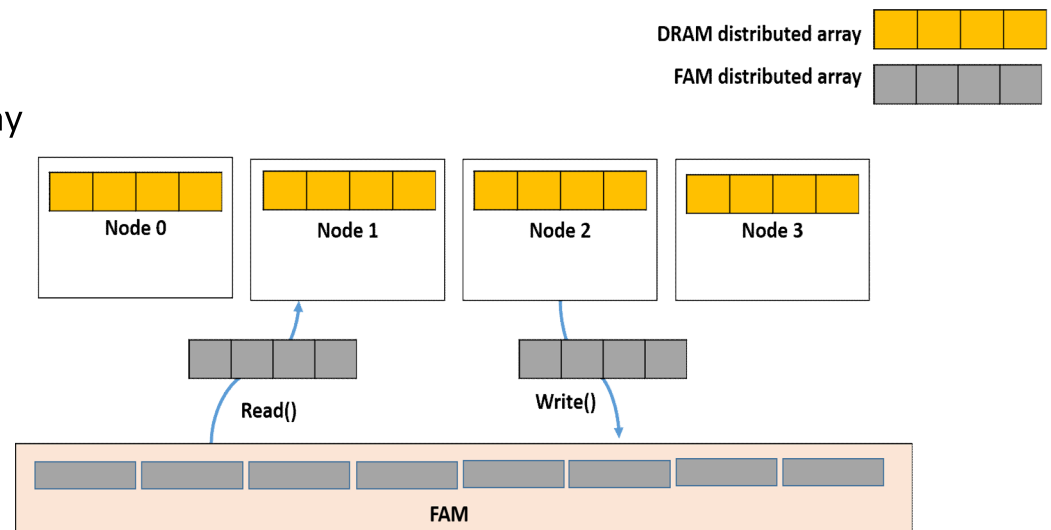
# FAM DISTRIBUTED ARRAYS - DESIGN

## High Level Design:

- FAM distribution module converts high level array operations into FAM-specific accesses underneath
- Complete array is allocated on FAM by the locale creating the array
- Each locale is then assigned a partition upon which to operate
- Array operations executed in parallel by target nodes
  - Example: forall, reduce or scan are divided into multiple tasks based on the partitioning, and executed in parallel by the target nodes.

## Design ensures that:

- Applications can allocate and reuse arrays located on FAM
- Our solution honors Chapel's programming philosophy, e.g., programmer productivity
- Management of FAM data allocation and accesses are abstracted away from the application
- Semantics of a FAM array is as close to that of existing Chapel distributions as possible





# FAM DISTRIBUTED ARRAYS - STATUS

Enable longer-term vision

- Current Status

- Preliminary implementation

- Array allocation, Array lookup, Array Destroy
    - Random indexed access
    - Iteration (serial and parallel loops with zippering)
    - Bulk transfers
    - Reduce and scan
    - Array slicing and re-indexing

```
const Dfam = {1..1000000} dmapped Fam() ; // FAM distributed domain
// Allocate new FAM distributed array
var FamArr: [Dfam] int;
FamArr.allocate(name="MyArray1", auto_destroy=false);
```

```
const Dfam = {1..1000000} dmapped Fam() ; // FAM distributed domain
var FamArr: [Dfam] int;
FamArr.find(name="MyArray2");
```

```
//Indexed element access
FamArr[100] = 100; // write a value to 100th element

var x = FamArr[50](); // read the value from 50th element
```

```
// Serial Loop on FAM array
for fa in FamArr do
    fa = 1; // assign the value "1" to array elements serially

// Parallel Loop on FAM array
forall fa in FamArr do
    fa = fa * 2; // Update the array data in parallel

forall (fa,ba) in zip(FamArr,BlockArr) do
    fa = ba; // Copy elements from Block array to FAM array
```

Examples of FAM access from Chapel

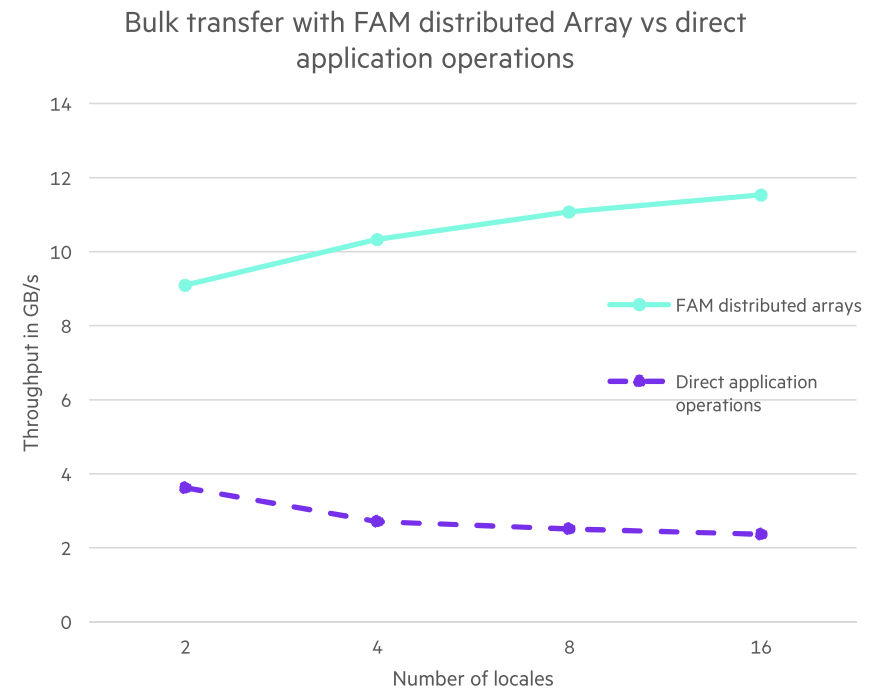
# BULK TRANSFER RESULTS

## Preliminary results with bulk transfer

- 25 GiB array copied from FAM to a DRAM distributed array using the bulk transfer operation
- Array directly copied from the application using OpenFAM APIs
- With bulk transfer
  - Throughput increases as the number of locales increase due to task parallelism with FAM distributed array
- Without bulk transfer
  - Throughput drops as the number of locales is increased as a result of the communication overhead between locales with application copy

## Configuration:

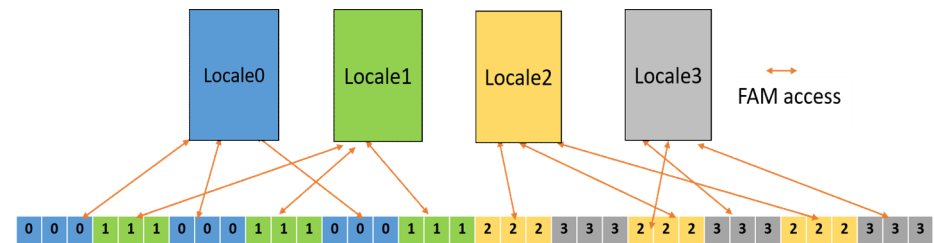
- Chapel 1.24
- 40 Xeon Gold 6248 cores (80 hyper threaded cores) with 128 GB memory running RHEL 8.3
- Infiniband cluster interconnected using 12.5 GB/s link fat-tree
- One of the nodes used as memory server



# MULTI-DIMENSIONAL ARRAYS

- Arrays are divided into blocks similar to Chapel's Block distribution
  - Block partition may result in non-contiguous elements per locale.
- Data may be stored on FAM contiguously
  - To access a given block, multiple data access requests are involved.

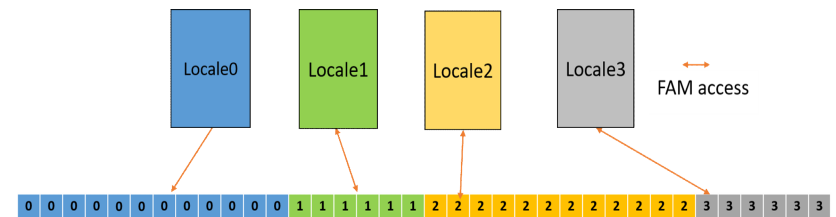
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
2	2	2	3	3	3
2	2	2	3	3	3
2	2	2	3	3	3



# MULTI-DIMENSIONAL ARRAYS

- Array is divided based on the number of rows and locales
  - For arrays greater than two dimensions, only the first dimension is partitioned based on the number of participating nodes
  - Each node gets a partition where all of its indices are contiguous
- Data is stored on FAM by row-major
  - Only a single OpenFAM data access call to access its data elements
  - Can result in unused locales if number of locales are more than the number of rows

0	0	0	0	0	0
0	0	0	0	0	0
1	1	1	1	1	1
2	2	2	2	2	2
2	2	2	2	2	2
3	3	3	3	3	3



## ALTERNATE APPROACHES TO PRESENTING FAM IN CHAPEL

---

- **FAM as a locale:** FAM as new top-level locality in Chapel
  - Chapel Locales[] array, provides an abstraction of all the localities
  - FAM as a remote locality visible to the other compute localities
  - Differs semantically from other localities, since it is a memory-only locality
- **FAM as a sub-locale:** Introduce FAM as a virtual sub-locale
  - Inspired by the KNL locality model for Intel's Xeon Phi processors with high-bandwidth on-package memory (HBM)
  - Requires significant compiler changes to support external memory over the network
- **FAM as an object:** New class that represent FAM objects
  - Abstract the FAM access semantics within this FAM class
  - Helper methods within the FAM class handle FAM access details through the communication layer



## **FAM ACCESS FROM CHAPEL – LOOKING AHEAD**

---

- Next Steps:
  - Characterize performance of FAM distributed arrays
  - Enable multi-dimensional array
  - Evaluate other proposals for enabling FAM access in Chapel
    - Enabling FAM as a Chapel object class
    - Present FAM as a sub-locale
  - Explore FAM usage in Arkouda - NumPy-Like Arrays at Massive Scale Backed by Chapel
- Competitive Approaches:
  - Represent data in FAM using file-system or key-value store abstractions
  - FAM access to Chapel applications cluster-wide through external libraries such as OpenFAM, or DAOS
  - Involve programming overhead, contrasts with Chapel's philosophy of programmer productivity



**THANK YOU**



## ACKNOWLEDGMENT

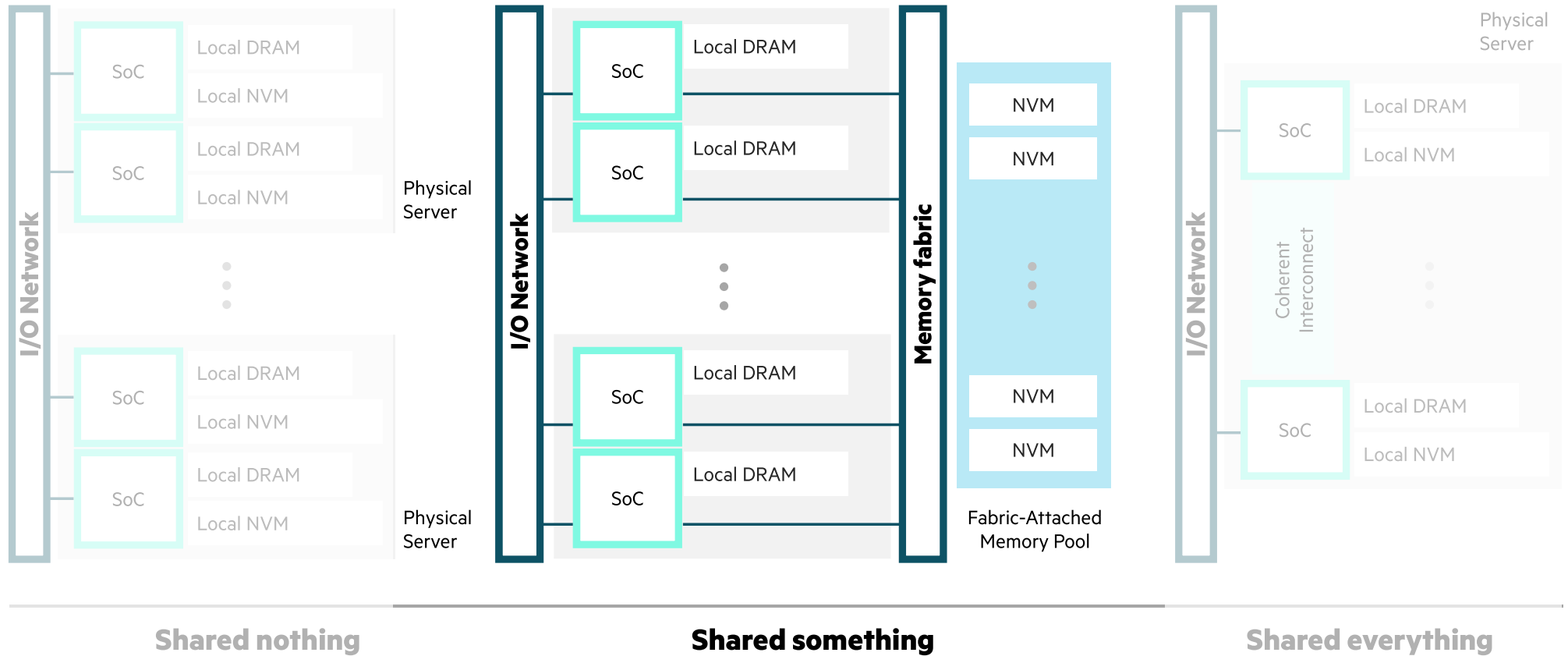
- We would like to thank Sanish Suresh, Greg Titus, Elliot Ronaghan, Michael Ferguson, Shome Porno, Chinmay Ghosh and Dave Emberson for their contributions to this project.

## REFERENCES

- [1] I. 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: Productive Parallel Programming," Apr. 01, 2022. <https://chapel-lang.org/> (accessed Apr. 01, 2022).
- [3] T. M. Press, "Programming Models for Parallel Computing | The MIT Press," Apr. 01, 2022. <https://sicm.mitpress.mit.edu/books/programming-models-parallel-computing> (accessed Apr. 01, 2022).
- [4] B. L. Chamberlain, S. J. Deitz, D. Iten, and S.-E. Choi, "User-defined distributions and layouts in chapel: philosophy and framework," in *Proceedings of the 2nd USENIX conference on Hot topics in parallelism*, USA, Jun. 2010, p. 12.
- [5] B. L. Chamberlain, S. Choi, S. J. Deitz, D. Iten, and V. Litvinov, "Authoring user-defined domain maps in chapel," 2011.
- [6] "OpenFAM: A library for programming Fabric-Attached Memory," Aug. 30, 2021. <https://openfam.github.io/index.html> (accessed Aug. 29, 2021).
- [7] "Domain Map Standard Interface — Chapel Documentation 1.24." <https://chapel-lang.org/docs/technotes/dsi.html> (accessed Aug. 31, 2021).
- [8] "OpenFAM: A library for programming Fabric-Attached Memory." <https://openfam.github.io/index.html> (accessed Aug. 29, 2021).
- [9] "An Approach To Data Distributions in Chapel - R.E. Diaconescu, H.P. Zima, 2007." <https://journals.sagepub.com/doi/abs/10.1177/1094342007078451?journalCode=hpcc> (accessed Aug. 31, 2021).
- [10] "Locale Models — Chapel Documentation 1.16." <https://chapel-lang.org/docs/1.16/technotes/localeModels.html#readme-knlm> (accessed Apr. 01, 2022).
- [11] A. Sodani, "Knights landing (KNL): 2nd Generation Intel® Xeon Phi processor," in *2015 IEEE Hot Chips 27 Symposium (HCS)*, Aug. 2015, pp. 1–24. doi: 10.1109/HOTCHIPS.2015.7477467.
- [12] "DAOS and Intel® Optane™ Technology for High-Performance Storage," *Intel*. <https://www.intel.com/content/www/us/en/high-performance-computing/daos-high-performance-storage-brief.html> (accessed Apr. 01, 2022).
- [13] Y. Shan, S.-Y. Tsai, and Y. Zhang, "Distributed shared persistent memory," in *Proceedings of the 2017 Symposium on Cloud Computing*, New York, NY, USA, Sep. 2017, pp. 323–337. doi: 10.1145/3127479.3128610.
- [14] "Rethinking software runtimes for disaggregated memory," *Penn State*. <https://pennstate.pure.elsevier.com/en/publications/rethinking-software-runtimes-for-disaggregated-memory/fingerprints/> (accessed Apr. 01, 2022).
- [15] M. Merrill, W. Reus, and T. Neumann, "Arkouda: interactive data exploration backed by Chapel," in *Proceedings of the ACM SIGPLAN 6th on Chapel Implementers and Users Workshop*, New York, NY, USA, Jun. 2019, p. 28. doi: 10.1145/3329722.3330148.
- [16] *Arkouda (αρκούδα): NumPy-like arrays at massive scale backed by Chapel*. Bears-R-U, 2021. Accessed: Aug. 29, 2021. [Online]. Available: <https://github.com/Bears-R-U/arkouda>



# FABRIC-ATTACHED (DISAGGREGATED) MEMORY IN CONTEXT



# OPENFAM

- **Purpose:**

- Develop an API and reference implementation to enable programmers to easily program FAM.

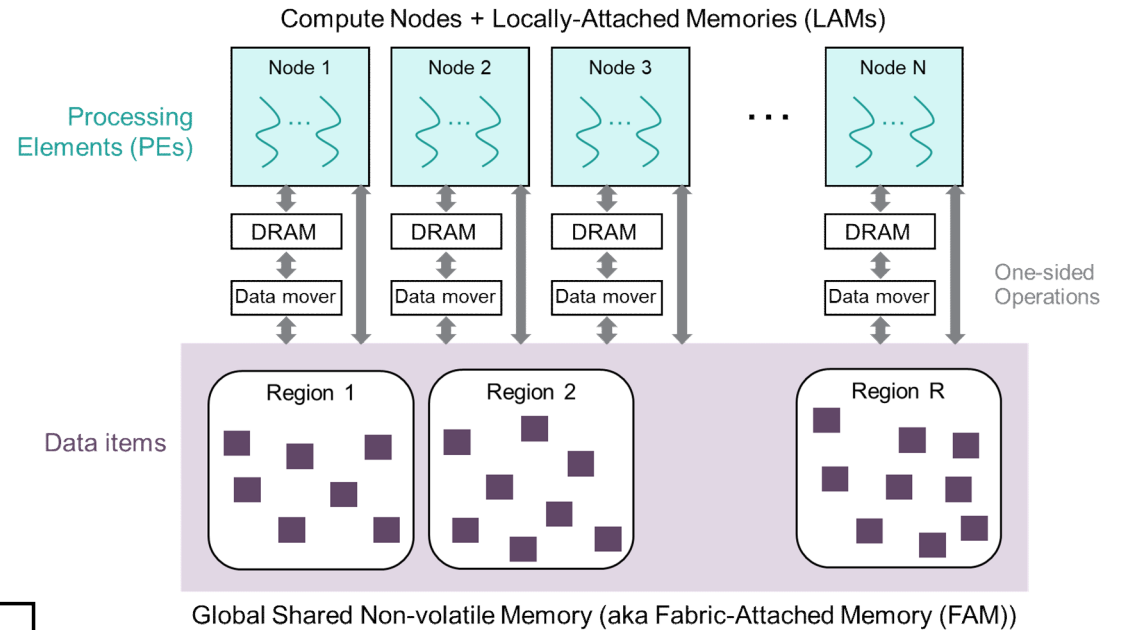
- **Challenges**

- 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

