LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Exploiting Hierarchical Exascale Hardware using a PGAS Approach

DASH: Data Structures and Algorithms with Support for Hierarchical Locality

Karl Fürlinger
Lehrstuhl für Kommunikationssysteme und Systemprogrammierung
LMU München

SPPEXA

DFG Deutsche Forschungsgemeinschaft

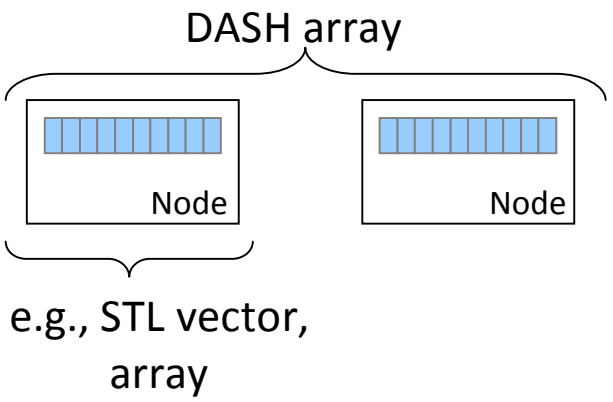
DASH – Overview

- DASH is a data-structure oriented C++ template library that realizes the PGAS (Partitioned Global Address Space) model

```
dash::Array<int> a(1000);

a[23]=412;
cout<<a[42]<<endl;
```

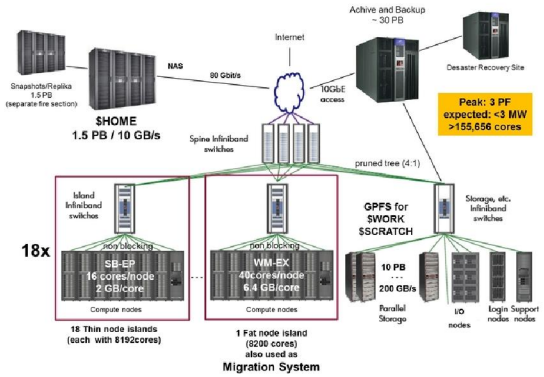
  - Array **a** can be stored in the memory of several nodes
  - a[i] transparently refers to local memory or to remote memory via operator overloading

- Not a new language to learn
  - Can be integrated with existing (MPI) applications
- Support for hierarchical locality
  - Team hierarchies and locality iterators

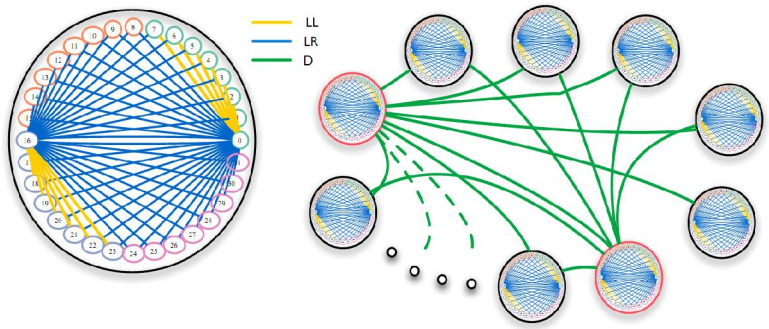


Hierarchy in Machines

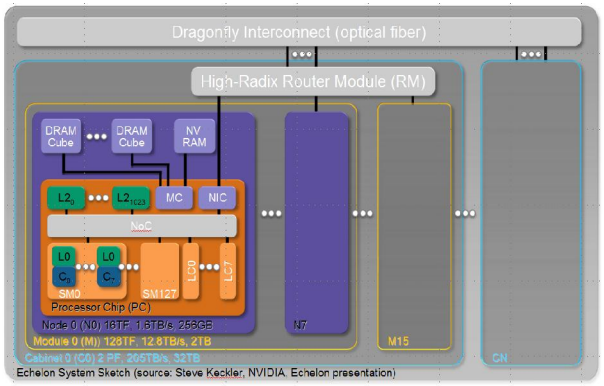
- Machines are getting increasingly **hierarchical**
  - Both **within** nodes and **between** nodes
  - Data locality is the most crucial factor for **performance** and **energy efficiency**



Source: LRZ SuperMUC system description.



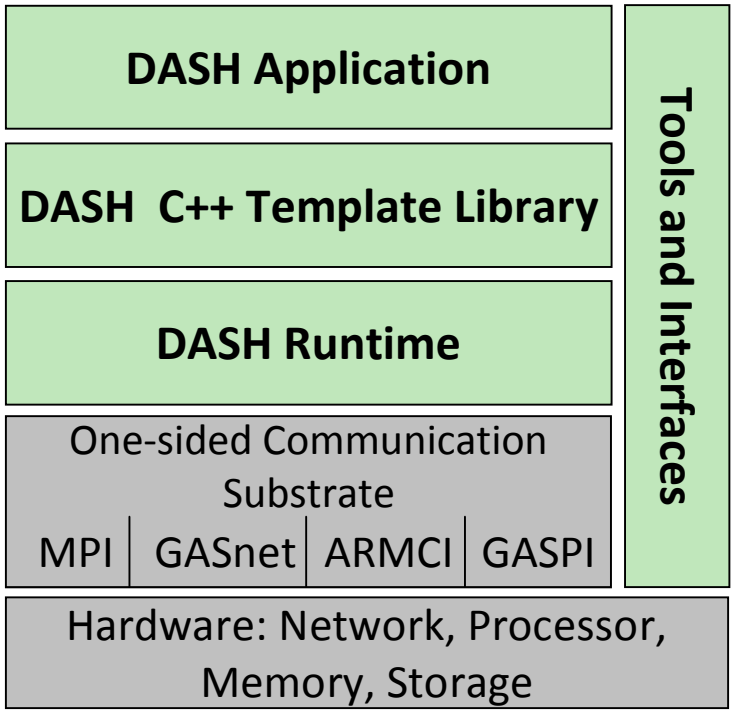
Source: Bhatele et al.: Avoiding hot-spots in two-level direct networks. SC 2011.



Source: Steve Keckler et al.: Echelon System Sketch

Hierarchical locality **not well supported** by current approaches. PGAS languages usually only offer a **two-level differentiation** (local vs. remote).

DASH – Overview and Project Partners



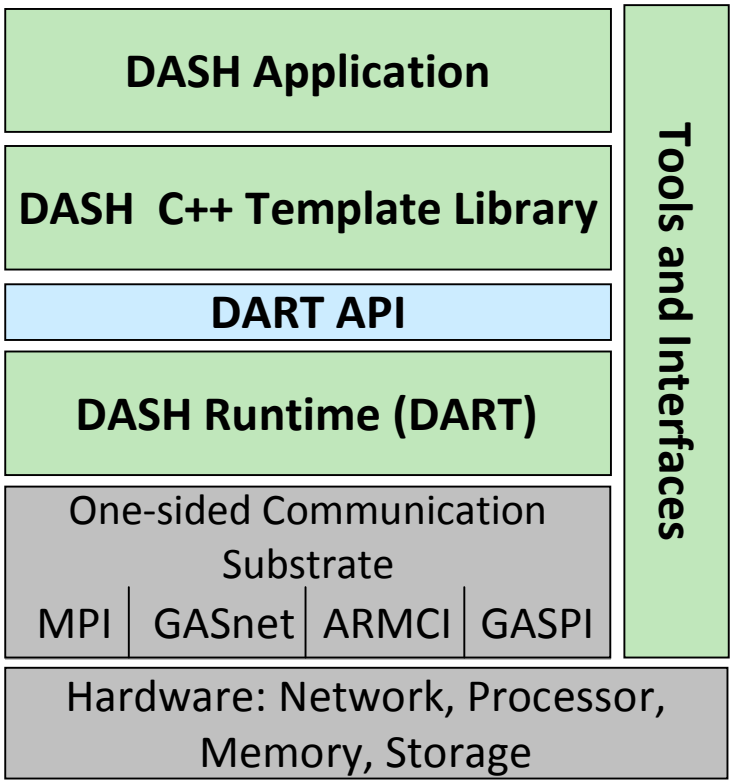
Component of DASH

Existing component/ Software

- Funded under the DFG priority programme "Software for Exascale Computing" (SPPEXA)

- Project Partners
  - LMU Munich (K. Fürlinger)
  - HLRS Stuttgart (J. Gracia)
  - TU Dresden (A. Knüpfer)
  - KIT Karlsruhe (J. Tao)
  - CEODE Beijing (L. Wang, associated)

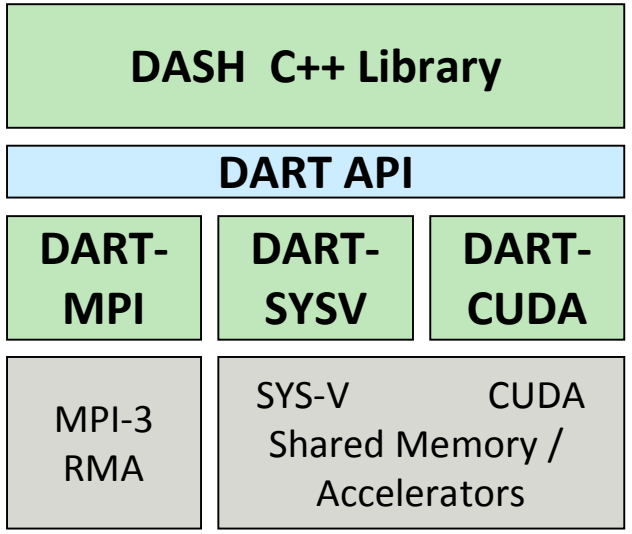
DART: The DASH Runtime Interface



- The DART API
  - Plain-C based interface
  - Follows the SPMD execution model
  - Defines **Units** and **Teams**
  - Defines a **global memory** abstraction
  - Provides a **global pointer**
  - Defines one-sided access operations (puts and gets)
  - Provides collective and pair-wise synchronization mechanisms

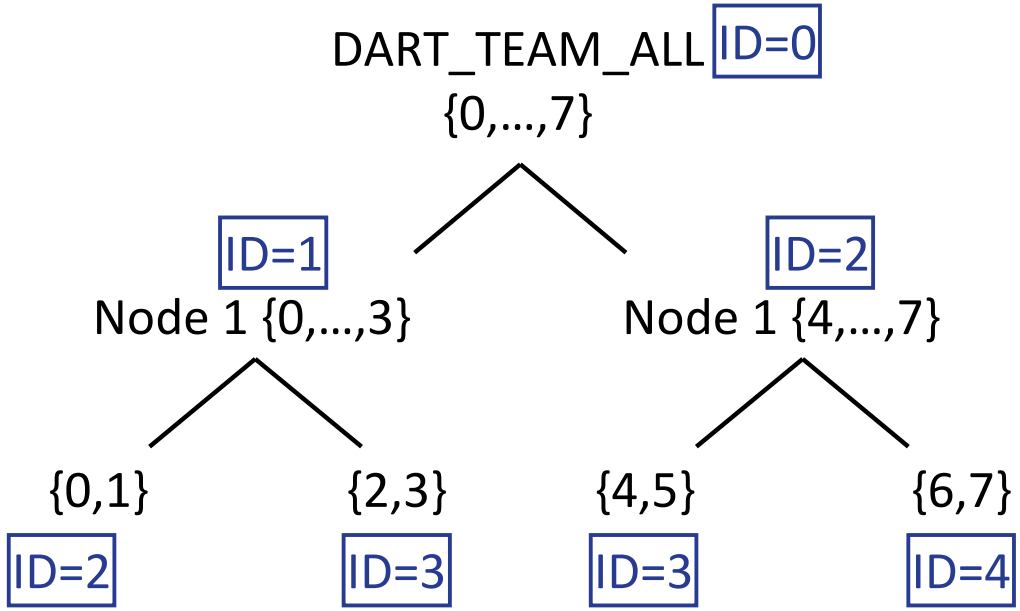
Three DART Implementations

- **DART-MPI**
  - Uses MPI-3 RMA
  - Scalable runtime
- **DART-SYSV** shared-memory based implementation
  - For shared-memory nodes only
  - Proof of concept & testing of DASH
- **DART-CUDA** extends DART-SYSV with support for accelerators
  - Research vehicle for the next iteration of the DART interface (execution model)

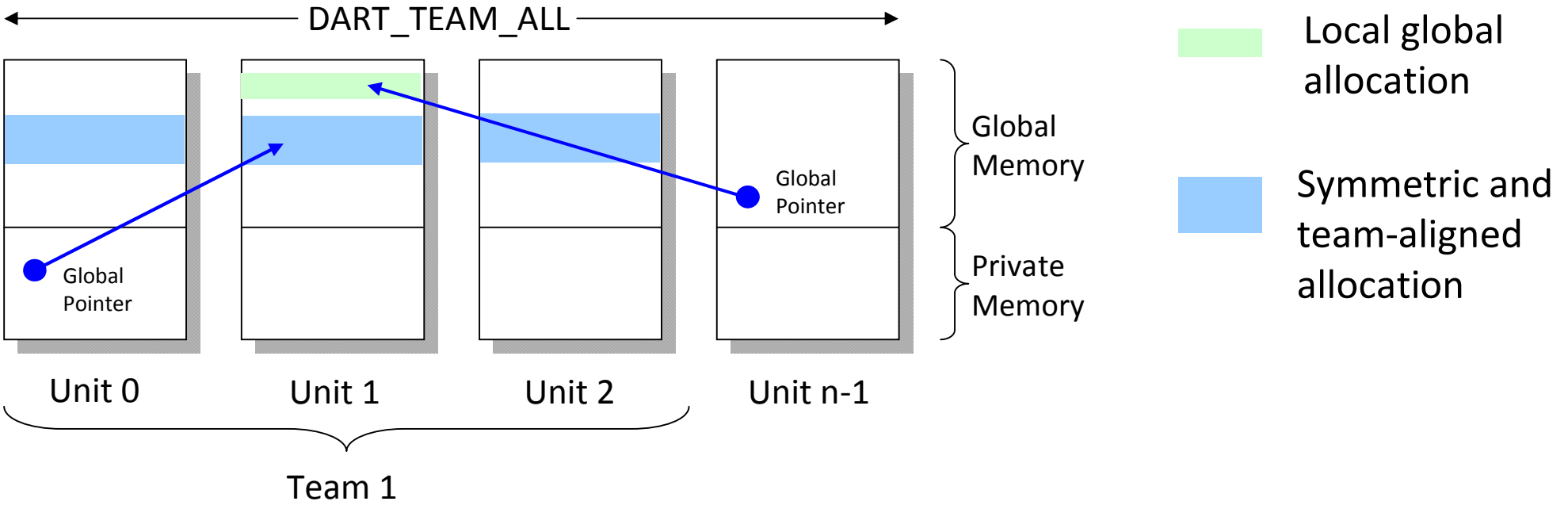


Units and Teams

- **Units**:
  - Individual participants in a DART/DASH program
  - Corresponds to thread/process/image in other PGAS appr.
- **Team**:
  - Ordered set of units
  - Subteams as subsets of a parent team
  - Local uniqueness guarantees



Memory Allocation and Access

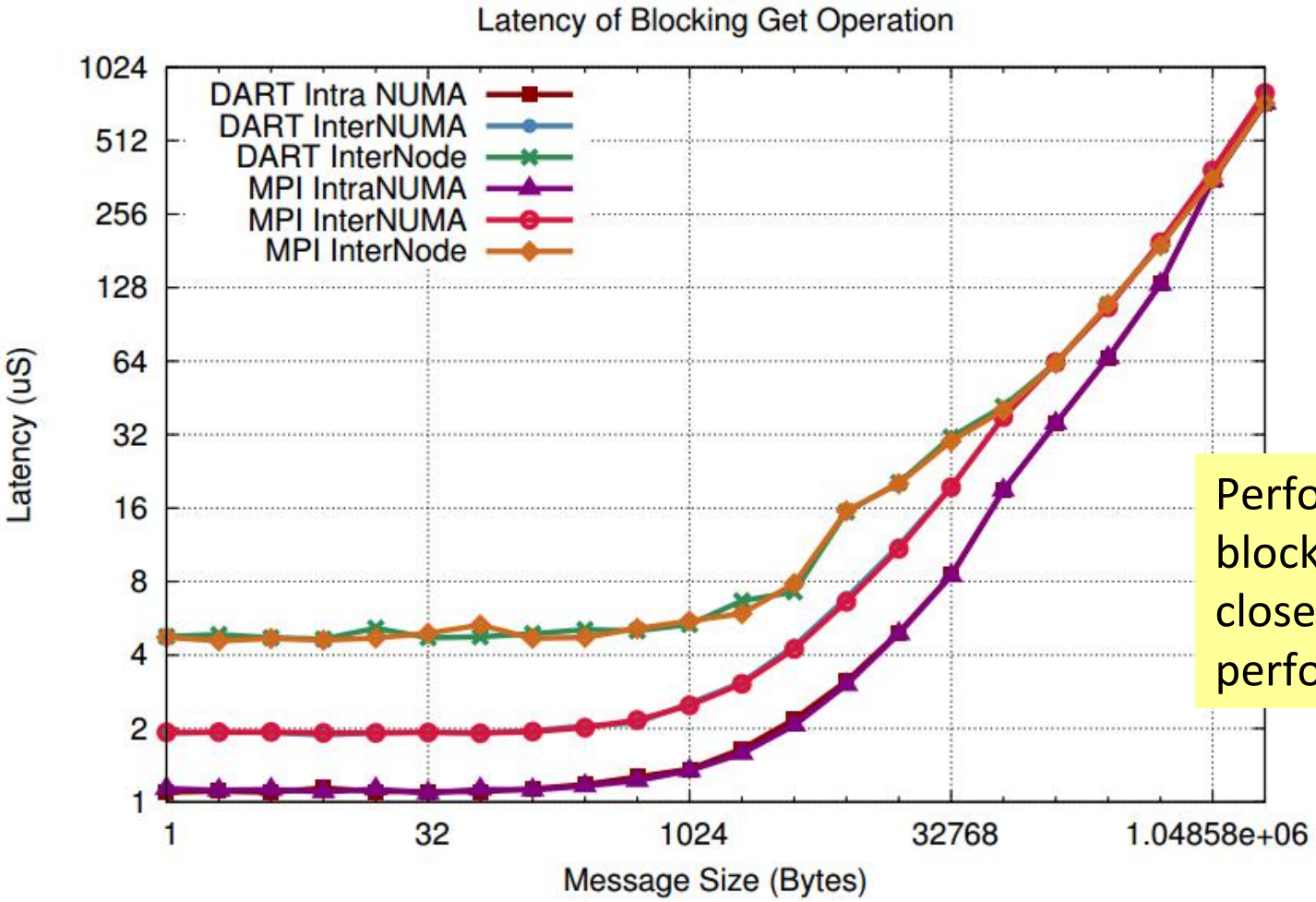


- Symmetric and team-aligned allocation
  - The same memory is allocated at each unit and each member of the team can easily compute the address of any location in any unit's part of the allocation

- Local global allocation
  - Globally accessible, no alignment guarantees, tied to DART_TEAM_ALL

Memory Access

- Communication: One-sided puts and gets
  - Blocking and non-blocking versions

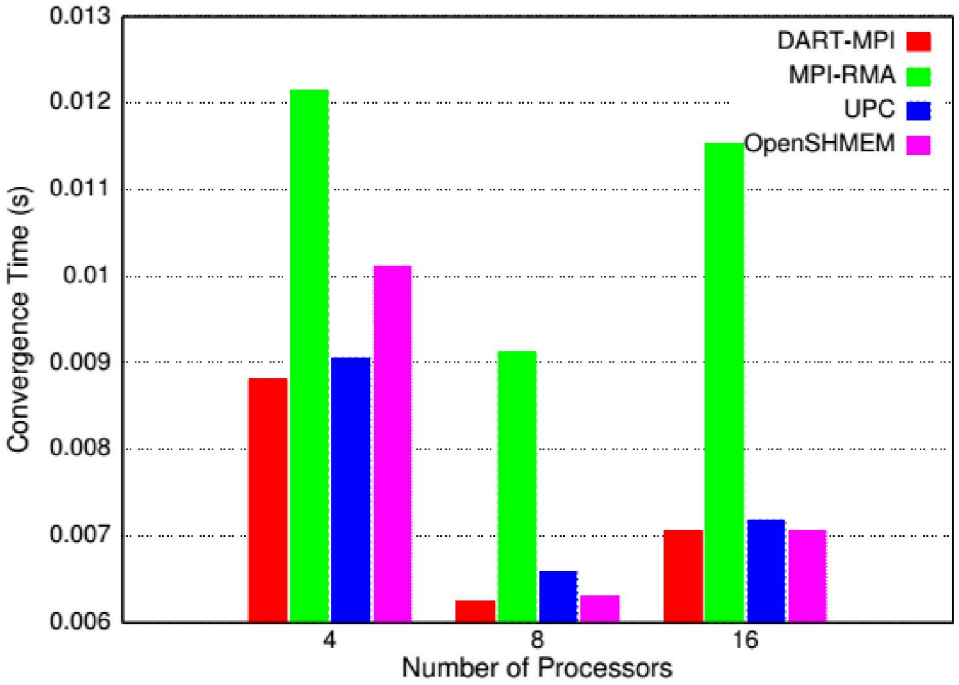


Performance of blocking puts and gets closely matches MPI performance

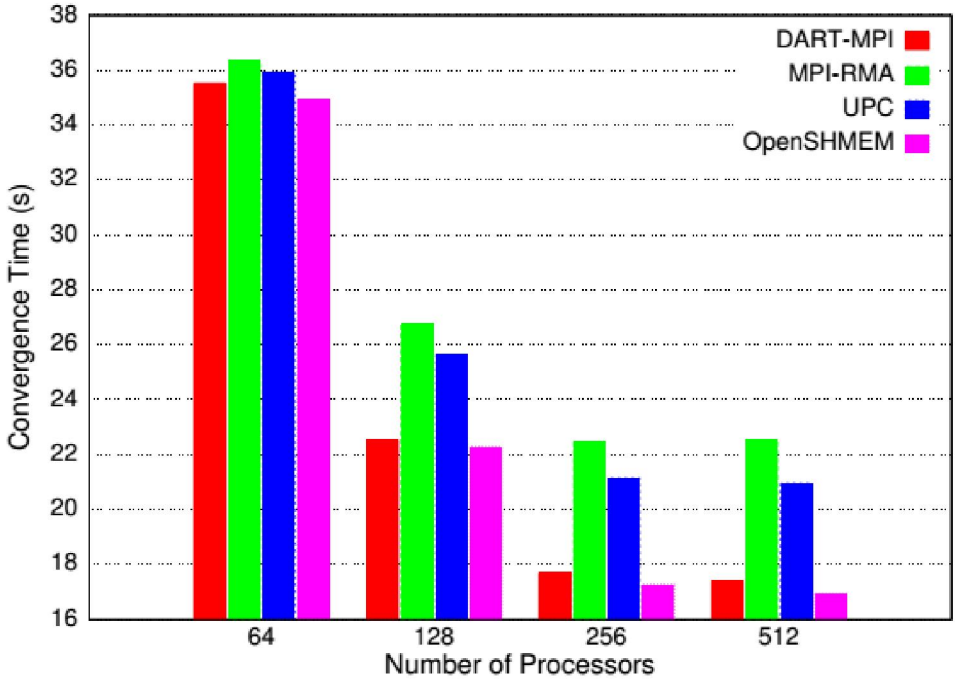
New: Shared Memory Communicators

- 5-point Stencil Example (Cray XC40, HLRS Hornet)

64x64 Grid



1024x1024 Grid



Shared memory communicator greatly improves performance – up to and beyond UPC and OpenSHMEM levels.

DASH (C++ Template Library)

- 1D array as the basic data type

```
int main(int argc, char* argv[])
{
  dash::init(&argc, &argv);

  // an allocation with the default team
  dash::Array<int> a(1000);

  dash::finalize();
}
```

- DASH follows a **global-view** approach, but local-view programming is supported as well
- Standard algorithms can be used but may not yield best performance
- **lbegin**(), **lend**() allow iteration over local elements only

Data Distribution Patterns

- A **Pattern** controls the mapping of an index space onto units

```
// blocked distribution of n elements
// onto the members of the default team
dash::Pattern p1(n, BLOCKED);
```

- No datatype is specified for a pattern, no mem. allocation is performed
- A team can be specified explicitly
- Patterns guarantee a similar mapping for different containers
- Patterns can be used to specify parallel execution

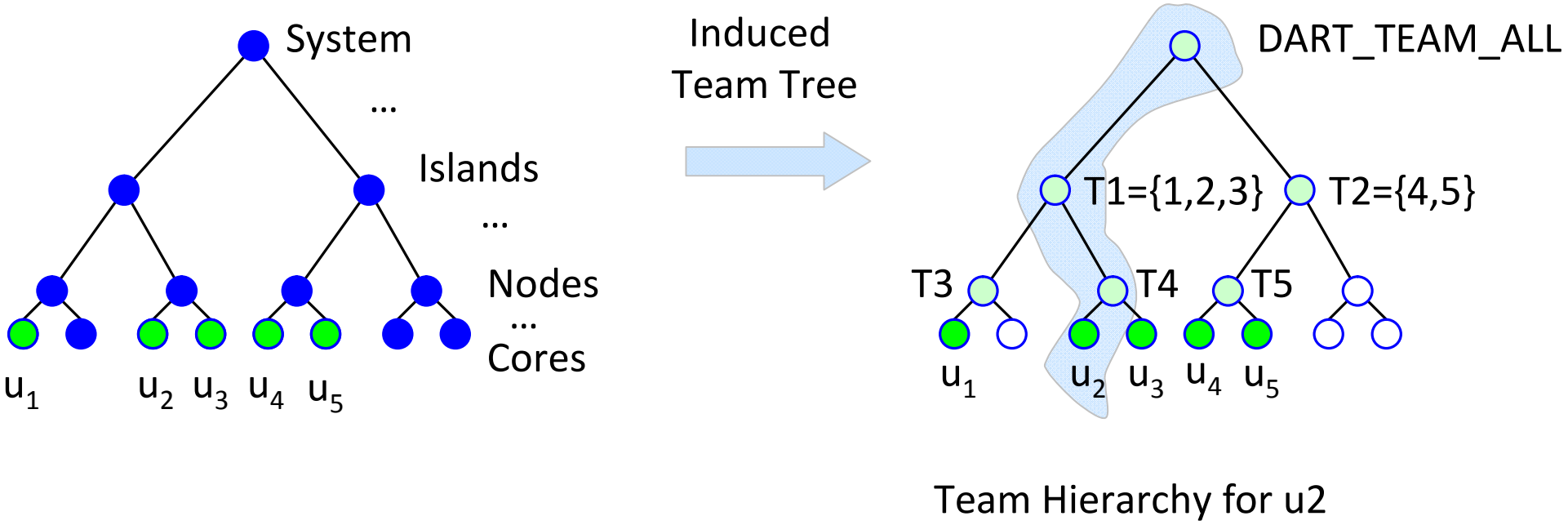
Global and Local Element Access

```
dash::Array<int> arr(1000);

// subscripting: [] and .at()
arr[823]=44;
int j = arr.at(999);

// standard algorithm with local
// iterators and lambda expression
mysum=0;
std::for_each(arr.lbegin(), arr.lend(),
        [&mysum](int x) { mysum+=x;} );

// range-based for (global)
for( auto el: arr )
  sum+=el;

// range-based for (local)
for( auto el: arr.local )
  mysum+=el;
```

- Subscripting and .at member function
- Local and global iterators
- Range-based for (global and local data via **.local** proxy object)

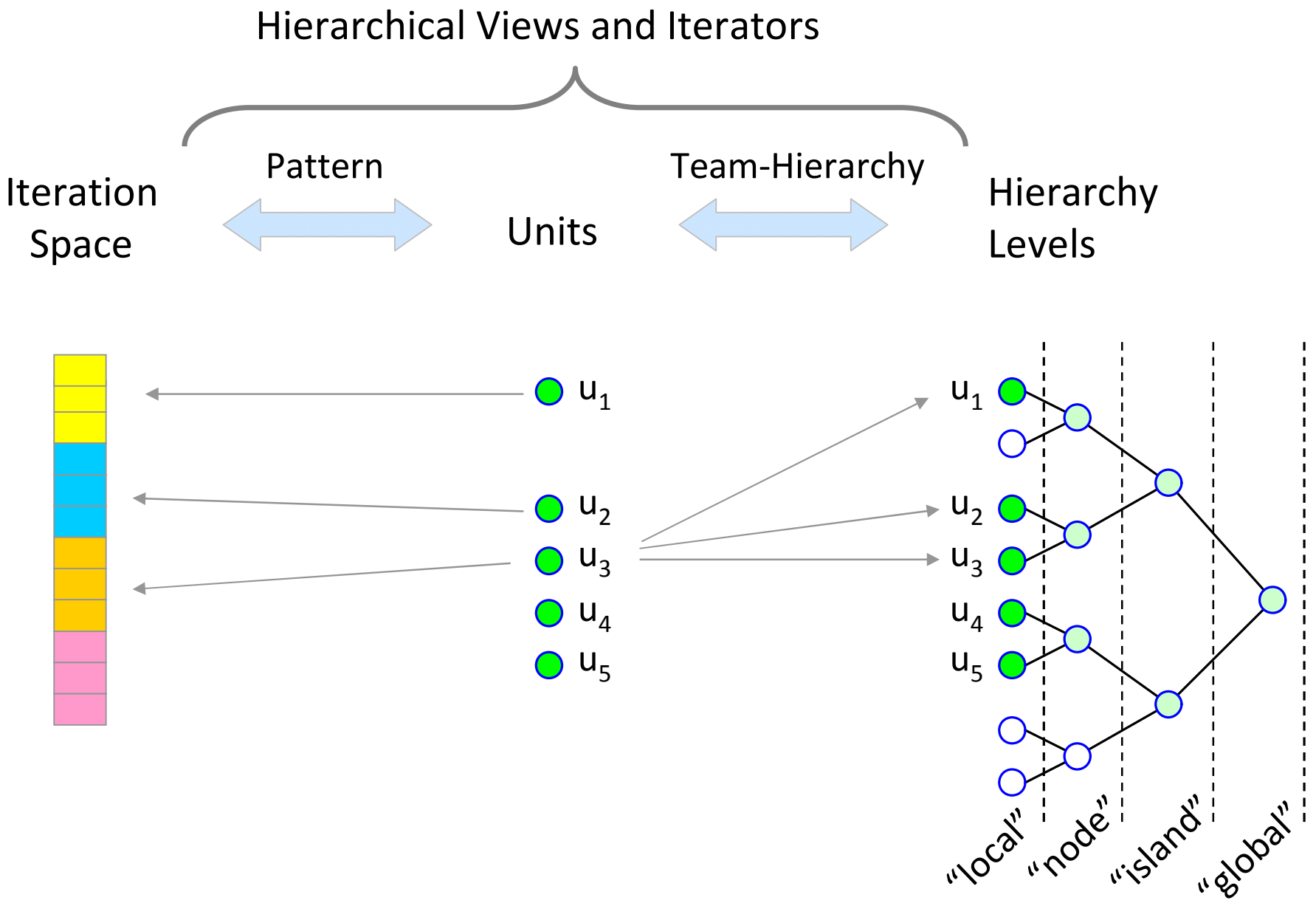
Machine-Tree, Team-Tree

- Machine Tree represents the hardware
  - Induces a team tree of units

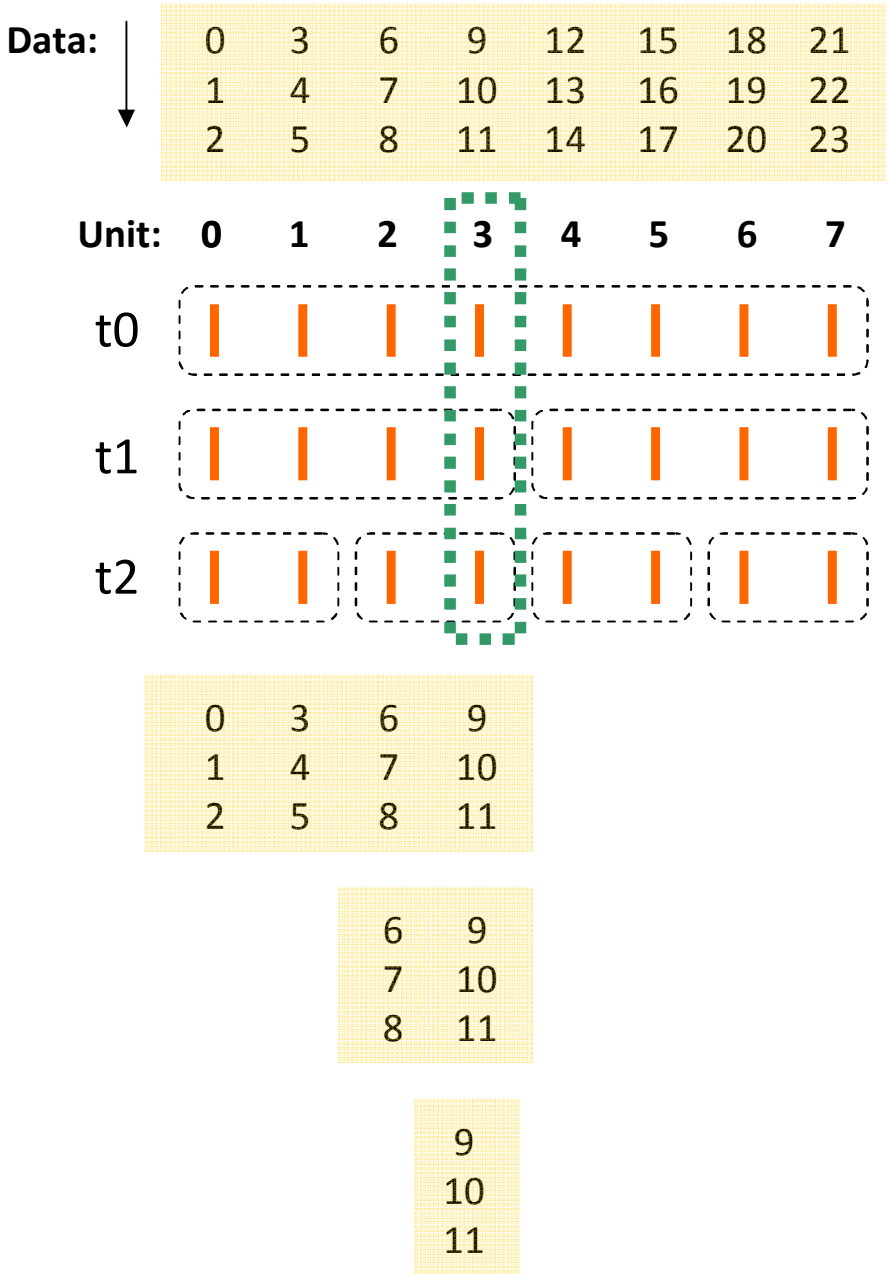


Team Hierarchy for u2

Hierarchical Iterators



Hierarchical Views and Iterators

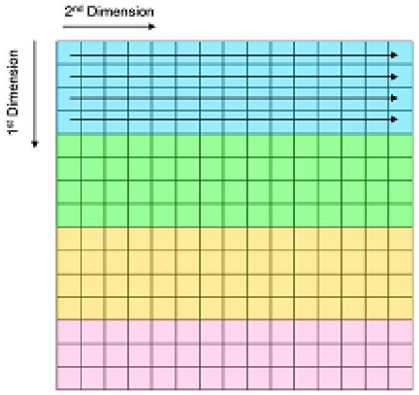



```
dash::Array<int> a(24);
// a = {0,1,2,3,4,...,23}
```

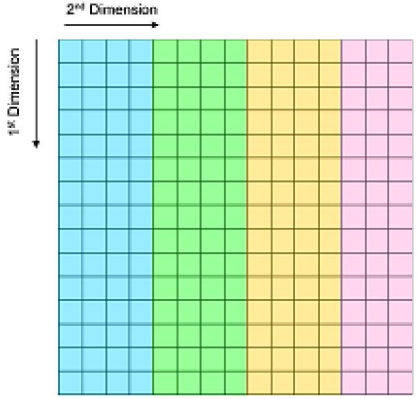
```
dash::Team& t0 = dash::Team::All();
dash::Team& t1 = t0.split(2);
dash::Team& t2 = t1.split(2);
```

```
if(myid==3) {

  // access on level of t1
  auto hv1 = a.hview<1>();
  for( auto el: hv1 ) { cout<<el; }

  // access on level of t2
  auto hv2 = a.hview<2>();
  for( auto el: hv2 ) { cout<<el; }

  // local access
  auto hv3 = a.hview<-1>();
  for( auto el: hv3 ) { cout<<el; }
}
```

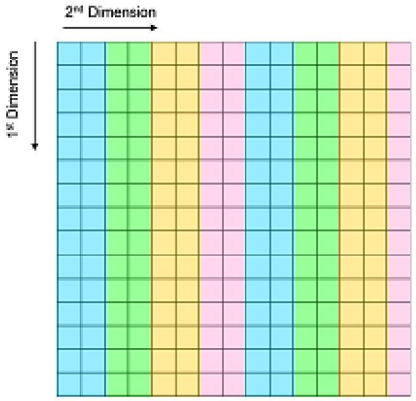
Ongoing: N-Dimensional Pattern and Matrix



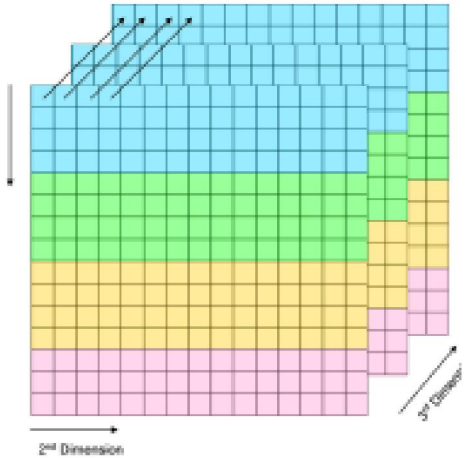
(BLOCKED, NONE)



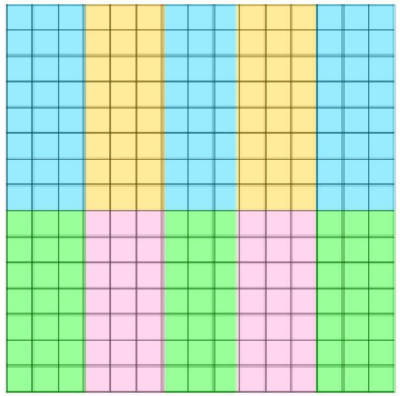
(NONE, BLOCKED)



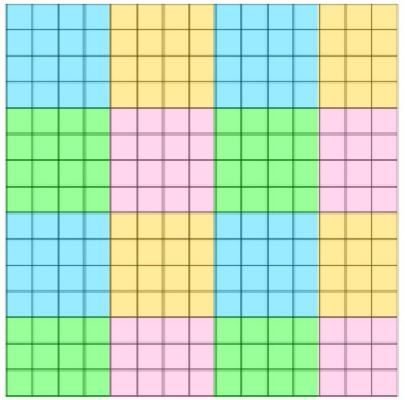
(NONE, BLOCKCYCLIC(2))



(BLOCKED, NONE, NONE)



(BLOCKED, BLOCKCYCLIC(3))

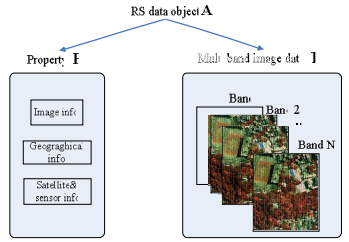


(BLOCKCYCLIC(4), BLOCKCYCLIC(4))

Tools & Interfaces and Applications

- Applications
  - Molecular Dynamics App (Stuttgart)
  - Remote Sensing App (CEODE)

- Tools and Interfaces
  - Performance and debugging tools interface
  - Parallel I/O to and from the data structures
  - Ongoing: debugger ingetration



- Both areas are on-going work and the focus of the second half of the project

Status

- DART: Final v1.0 spec
  - Available online: http://www.dash-project.org/dart/
  - DART can be the foundation for other PGAS approaches
  - Next iteration: execution model

- DASH
  - DART-MPI + DASH release in the works (array and matrix)
  - Next iteration: dynamic data structures

Thank you for your attention!