LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

DASH: A C++ PGAS Library for Distributed Data Structures and Parallel Algorithms

www.dash-project.org

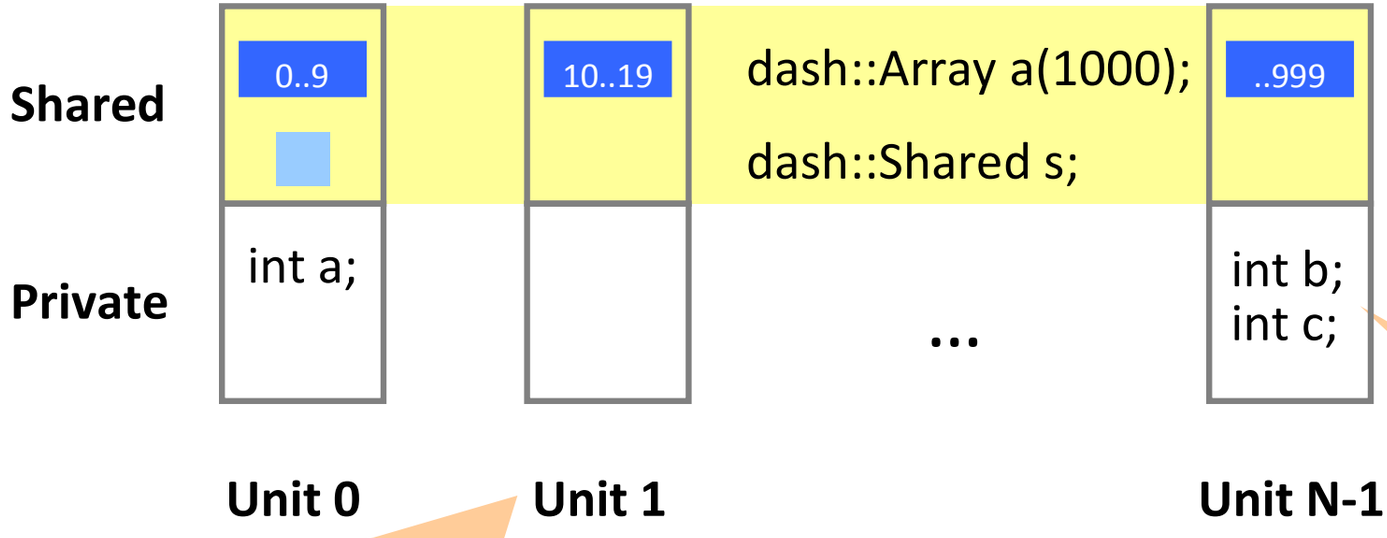
Karl Fürlinger
Ludwig-Maximilians-Universität München

Outline

- Overview and project structure
- DASH features
  - Efficient access to local data
  - STL conformity / iterator interface
  - Distributed multidimensional arrays
  - Teams
  - HDF5 input/output
- Porting LULESH (on-going work)
- What's planned next for DASH

DASH - Overview

- DASH is a C++ template library that offers
  - Distributed data structures and parallel algorithms
  - A complete PGAS (part. global address space) programming system without a custom (pre-)compiler
- Terminology



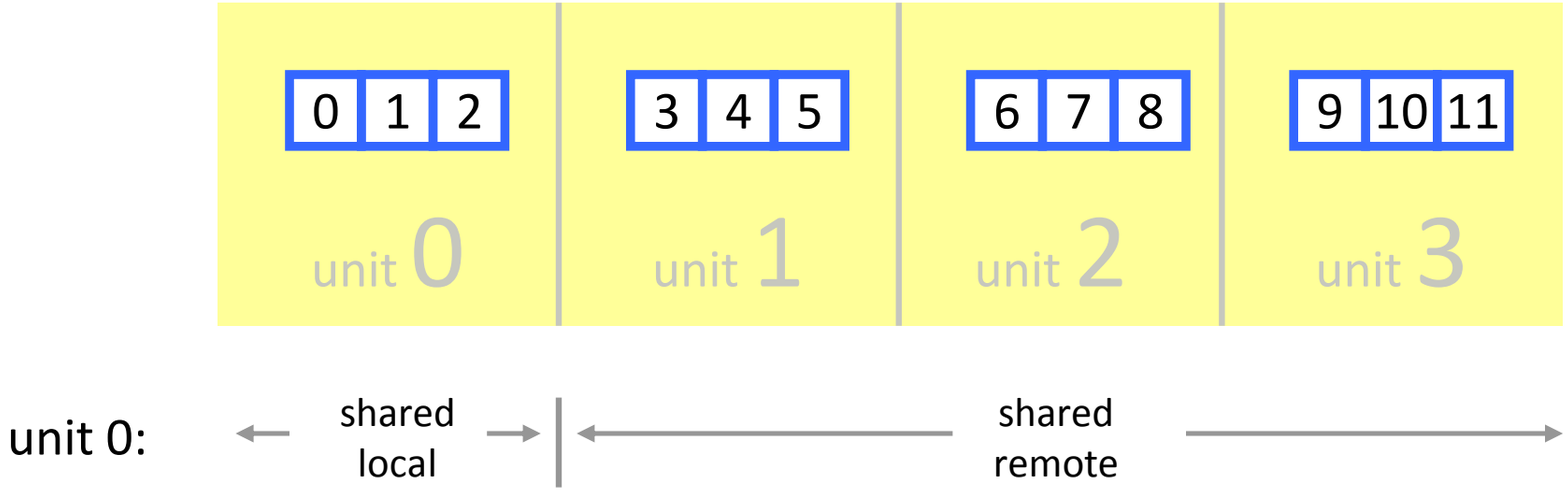
Shared data: managed by DASH in a virtual global address space

Private data: managed by regular C++ mechanisms

Unit: The individual participants in a DASH program, usually full OS processes.

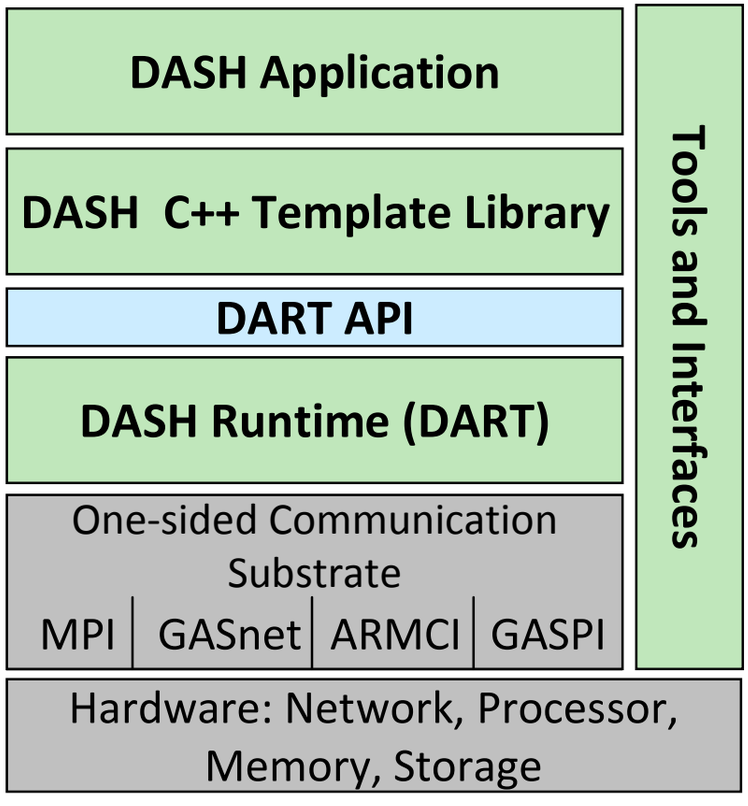
The “Partitioned“ in PGAS

- Example: dash::Array with 12 elements



- Data affinity
  - Elements can be accessed by any unit, but each one has an explicit and well defined home (owner) unit
  - Data locality important for performance
  - Support for the *owner computes* execution model

DASH Project Structure



	Phase I (2013-2015)	Phase II (2016-2018)
LMU Munich	Project management, C++ template library	Project management, C++ tempalte library, DASH data dock
TU Dresden	Libraries and interfaces, tools support	Smart data structures, resilience
HLRS Stuttgart	DART runtime	DART runtime
KIT Karlsruhe	Application case studies	
IHR Stuttgart		Smart deployment, Application case studies

dash

www.dash-project.org

SPPEXA

DASH is one of 16 SPPEXA projects

DART: The DASH Runtime

- The DART Interface
  - Plain-C based interface (“dart.h”)
  - Follows the SPMD execution model
  - Provides global memory abstraction and global pointers
  - Defines one-sided access operations (puts and gets) and synchronization operations

- Several implementations
  - **DART-SHMEM**: shared-memory based implementation
  - **DART-CUDA:** supports GPUs, based on DART-SHMEM
  - **DART-GASPI**: Initial implementation using GASPI
  - **DART-MPI**: MPI-3 RMA based “workhorse” implementation

Hello World in DASH

```
#include <iostream>
#include <libdash.h>

using namespace std;

int main(int argc, char* argv[])
{
  pid_t pid; char buf[100];

  dash::init(&argc, &argv);
  auto myid = dash::myid();
  auto size = dash::size();
  gethostname(buf, 100); pid = getpid();

  cout<<"'Hello world' from unit "<<myid<<
    " of "<<size<<" on "<<buf<<" pid="<<pid<<endl;

  dash::finalize();
}
```

Initialize the programming environment

Determine total number of units and our own unit ID

Print message. Note SPMD model, similar to MPI.

```
$ mpirun -n 4 ./hello
'Hello world' from unit 2 of 4 on nuc03 pid=30964
'Hello world' from unit 0 of 4 on nuc01 pid=25422
'Hello world' from unit 3 of 4 on nuc04 pid=32243
'Hello world' from unit 1 of 4 on nuc02 pid=26304
```

Distributed Data Structures

- DASH offers distributed data structures
  - Support for flexible data distribution schemes
  - Example: dash::Array<T>

```
dash::Array<int> arr(100);

if( dash::myid()==0 ) {
  for( auto i=0; i<arr.size(); i++ )
    arr[i]=i;
}

arr.barrier();
if(dash::myid()==1 ) {
  for( auto el: arr )
    cout<<(int)el<<" ";
  cout<<endl;
}
```

DASH global array of 100 integers, distributed over all units, default distribution is BLOCKED

Unit 0 writes to the array using the global index i. Operator [] is overloaded for the dash::Array.

Unit 1 executes a range-based for loop over the DASH array

```
$ mpirun -n 4 ./array
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
88 89 90 91 92 93 94 95 96 97 98 99
```

Accessing Local Data

- Access to the local portion of the data is exposed through a local-view proxy object (.local)

```
dash::Array<int> arr(100);

for( auto i=0; i<arr.lsize(); i++ )
  arr.local[i]=dash::myid();

arr.barrier();
if(dash::myid()==dash::size()-1 ) {
  for( auto el: arr )
    cout<<(int)el<<" ";
  cout<<endl;
}
```

.lsize() is short hand for .local.size() and returns the number of local elements

.local is a *proxy object* that represents the part of the data that is local to a unit.

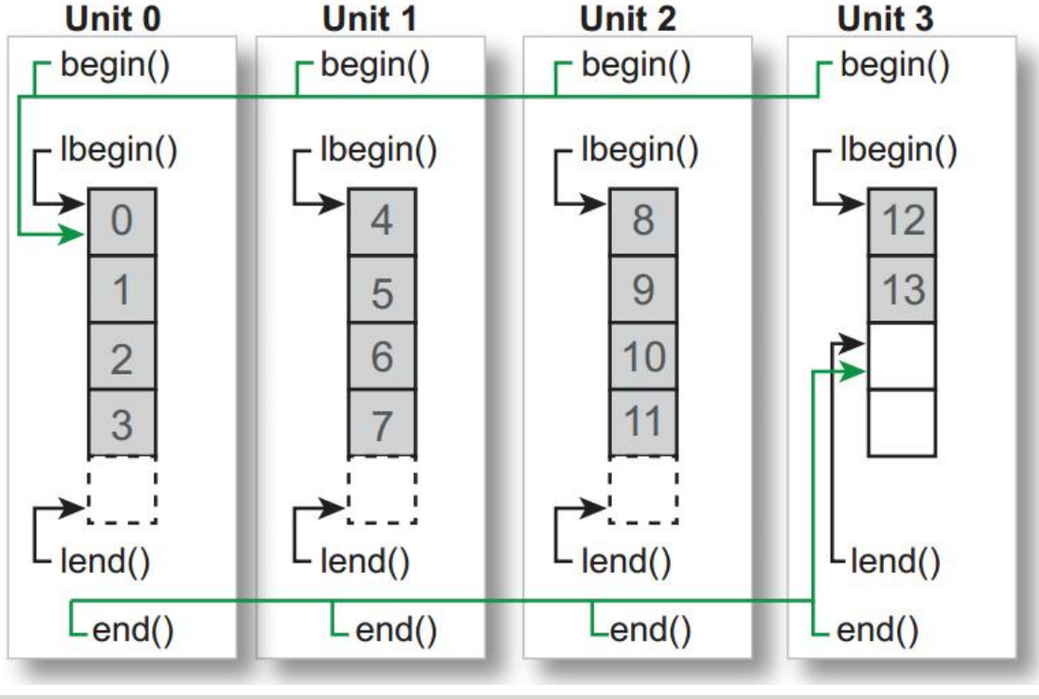
```
$ mpirun -n 4 ./array
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3
```

Global-view and Local-view in DASH

- DASH supports both *global-view* and *local-view* semantics

	Global-view	Local-view	LV shorthand
range begin	arr.begin()	arr.local.begin()	arr.lbegin()
range end	arr.end()	arr.local.end()	arr.lend()
# elements	arr.size()	arr.local.size()	arr.lsize()
element access	arr[glob_idx]	arr.local[loc_idx]	

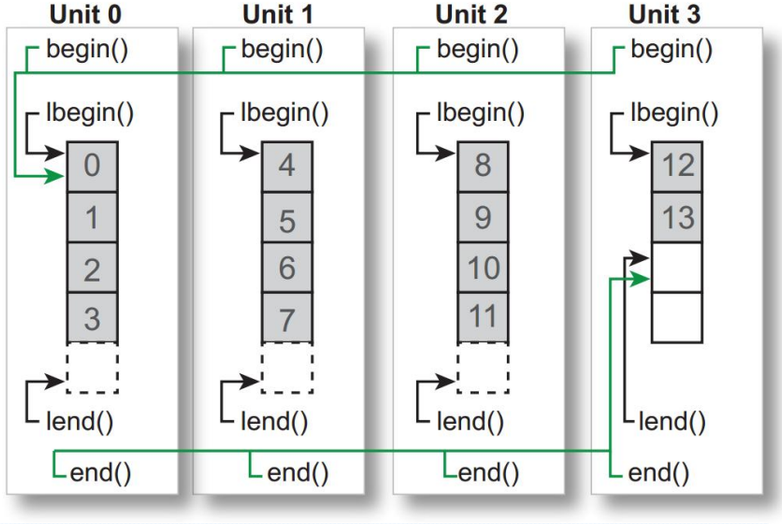
- Example
  - dash::Array with 14 elements, distributed over 4 units
  - default distribution: BLOCKED
  - Blocksize = ceil(14/4)=4



Efficient Local Access (1)

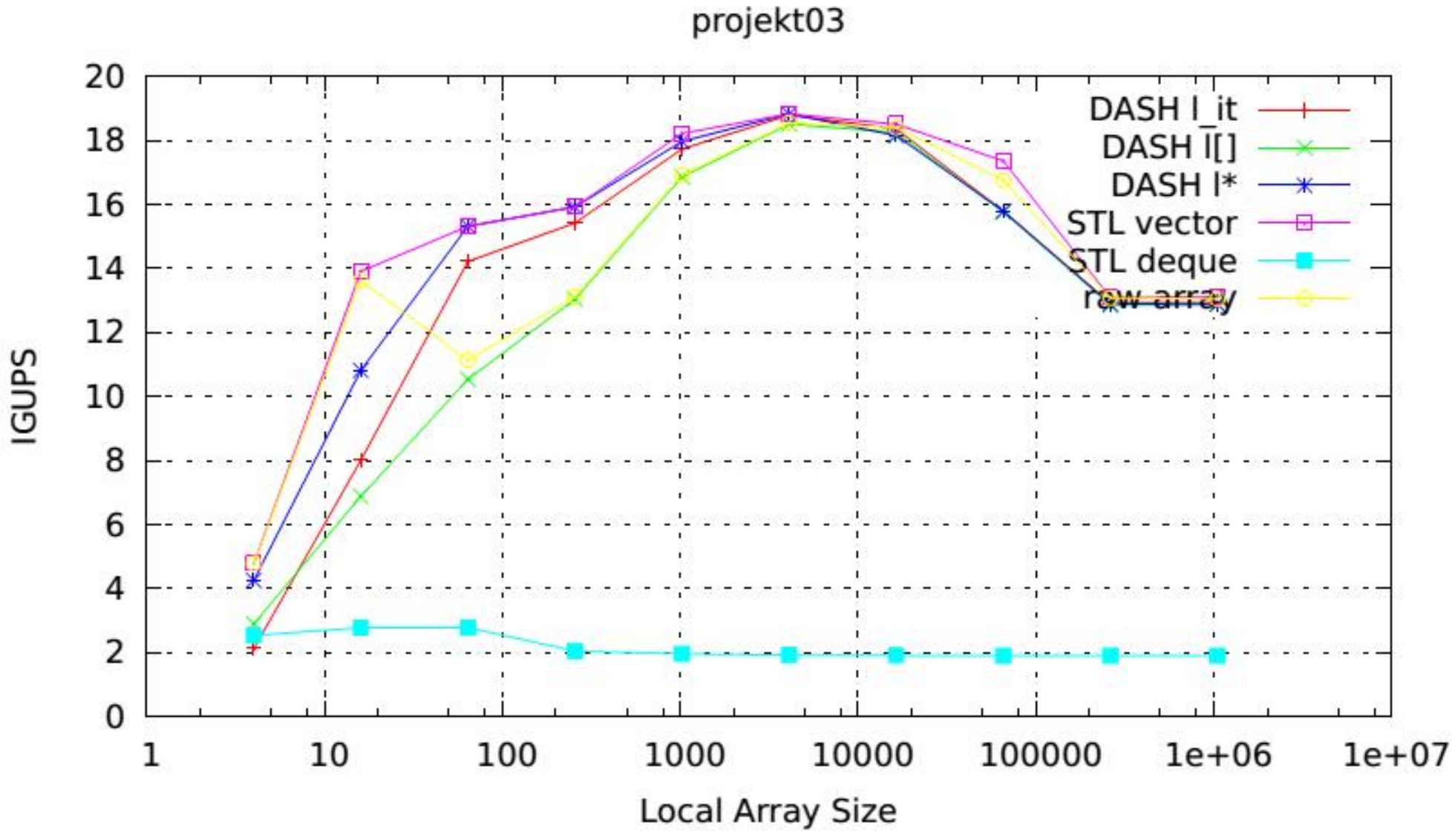
- Several options for access to local data

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

// get raw pointer to local mem.
int *p1 = arr.local.begin();
int *p2 = arr.lbegin(); //p1==p2

// access via local index
arr.local[22] = 33;

// range-based for loop
for( auto el : arr.local )
  cout<<el<<" ";

// access using local iterators
for( auto it=arr.lbegin(); it!=arr.lend(); ++it ) {
  (*it) = foo(...);
}
```



Efficient Local Access (2)

- IGUPs Benchmark: independent parallel updates



Using STL Algorithms

- STL algorithms can be used with DASH containers
  - Both on the local view and the global view

```
#include <libdash.h>

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

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

  if( dash::myid()==0 ) {
    // global iterators and std. algorithms
    std::sort(a.begin(), a.end());
  }

  // local access using local iterators
  std::fill(a.lbegin(), a.lend(), 23+dash::myid());

  dash::finalize();
}
```

Collective constructor, all units involved

STL algorithms work with DASH global iterators

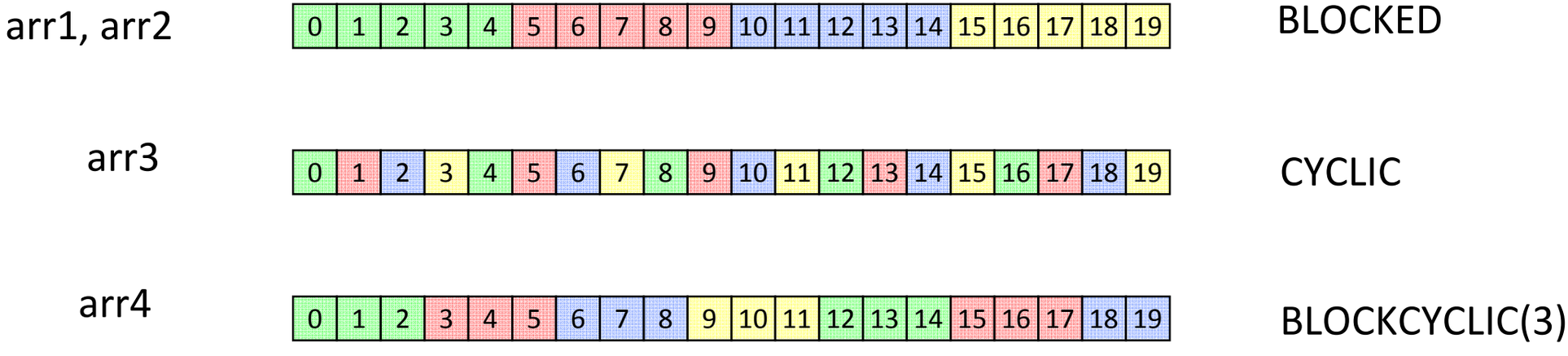
STL algorithms work with DASH local iterators

Data Distribution Patterns

- The data distribution pattern is configurable

```
dash::Array<int> arr1(20); // default: BLOCKED

dash::Array<int> arr2(20, dash::BLOCKED)
dash::Array<int> arr3(20, dash::CYCLIC)
dash::Array<int> arr4(20, dash::BLOCKCYCLIC(3))
```

- Assume 4 units



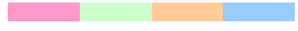
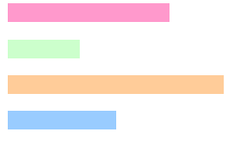
Accessing Local Data

- The previous example with a cyclic distribution:

```
// this is the only changed line
dash::Array<int> arr(100, dash::CYCLIC);

for( auto i=0; i<arr.lsize(); i++ )
  arr.local[i]=dash::myid();

arr.barrier();
if(dash::myid()==dash::size()-1 ) {
  for( auto el: arr )
    cout<<(int)el<<" ";
  cout<<endl;
}
```

```
$ mpirun -n 4 ./array
0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0
1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1
2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2
3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3
0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3
```

DASH Distributed Data Structures Overview

Container	Description	Data distribution
Array<T>	1D Array 	static, configurable
NArray<T, N>	N-dim. Array 	static, configurable
Shared<T>	Shared scalar 	fixed (at 0)
Directory(*)<T>	Variable-size, locally indexed array 	manual

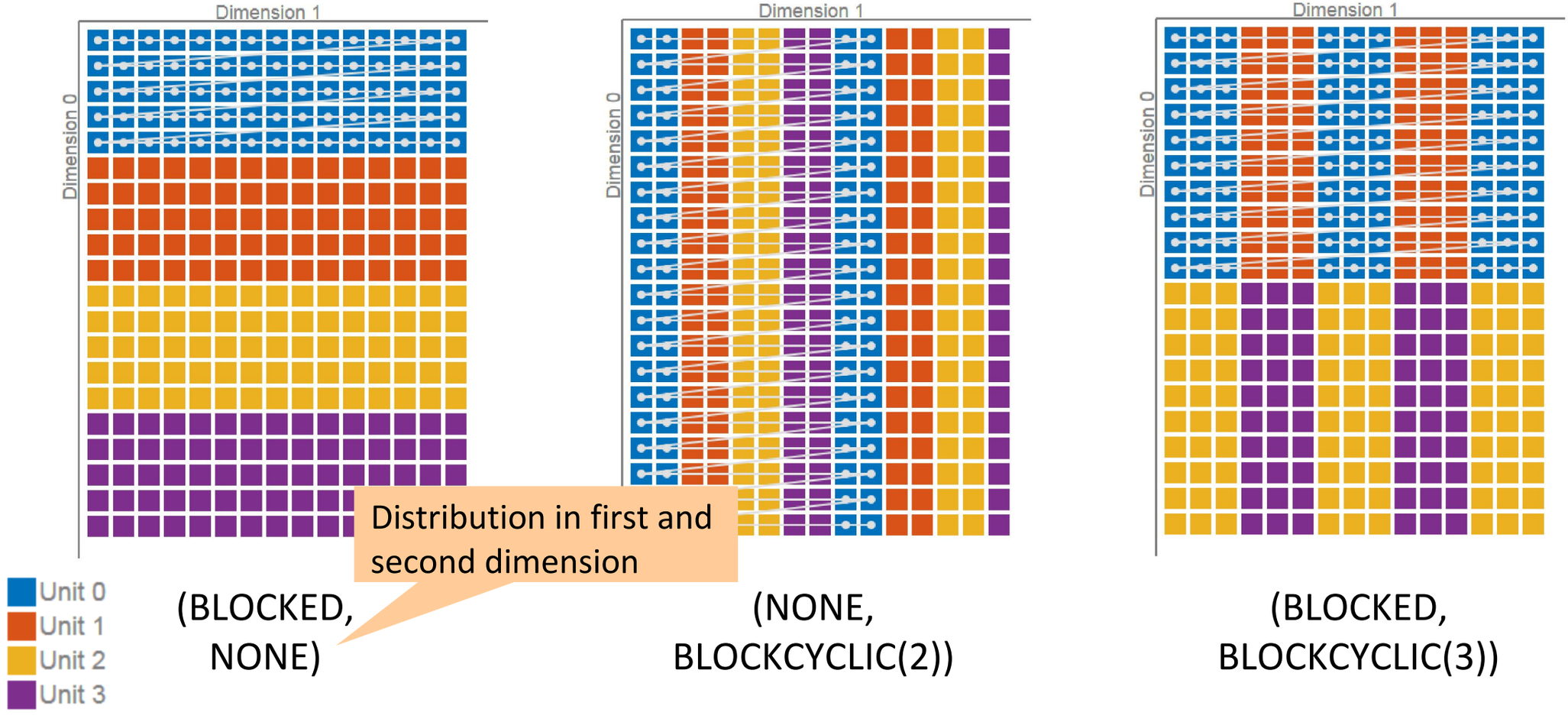
(*) Under construction

Multidimensional Data Distribution (1)

- dash::Pattern<N> specifies N-dim data distribution
  - Blocked, cyclic, and block-cyclic in multiple dimensions

Pattern<2>(20, 15)

Size in first and second dimension



Distribution in first and second dimension

(BLOCKED, NONE)

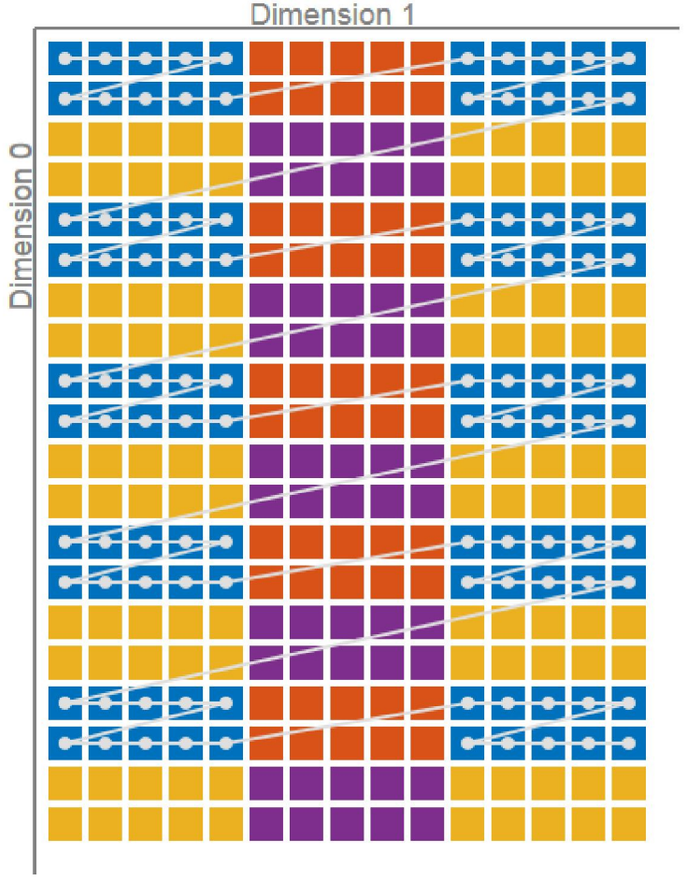
(NONE, BLOCKCYCLIC(2))

(BLOCKED, BLOCKCYCLIC(3))

Multidimensional Data Distribution (2)

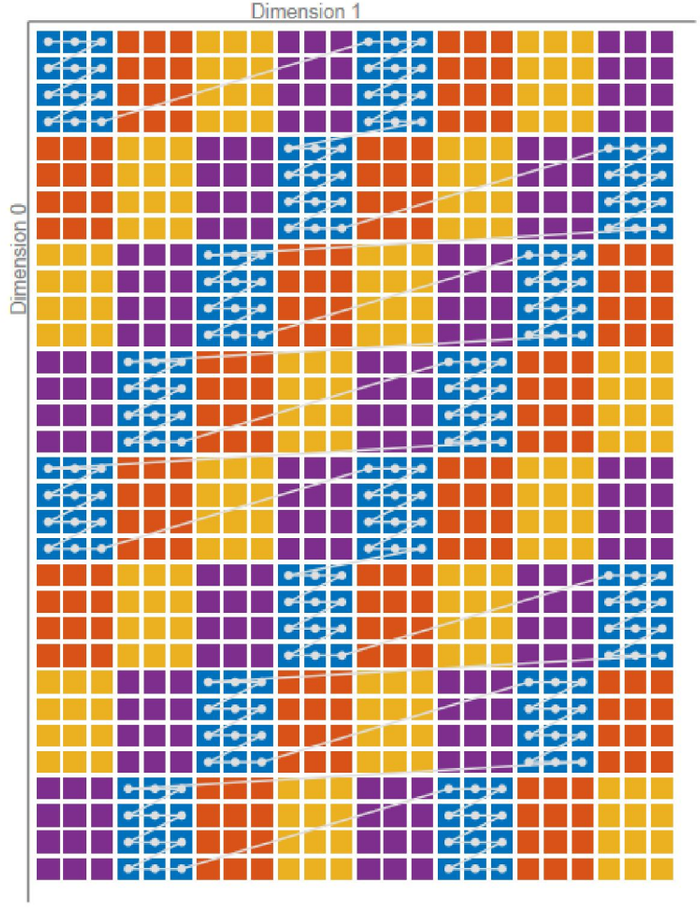
- Tiled data distribution and tile-shifted distribution

TilePattern<2>(20, 15)



(TILE(2), TILE(5))

ShiftTilePattern<2>(32, 24)

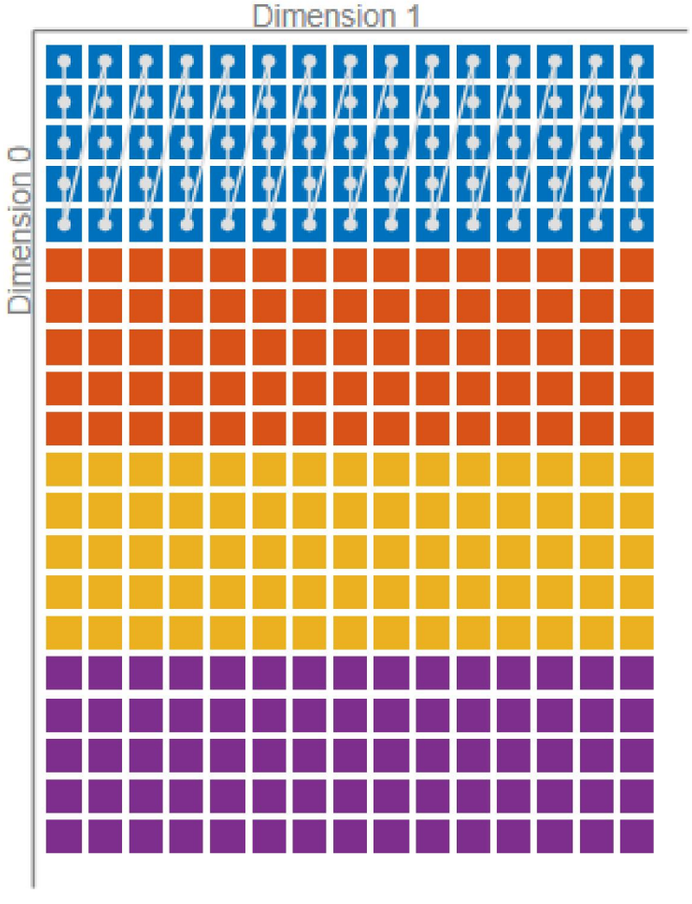


(TILE(4), TILE(3))

Multidimensional Data Distribution (3)

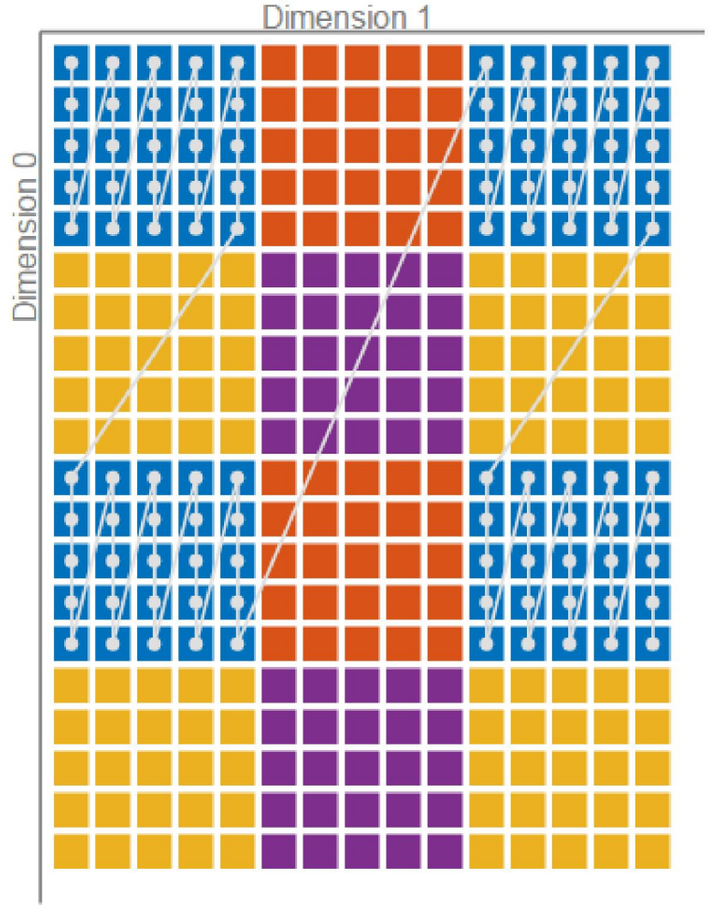
- Row-major and column-major storage

Pattern<2, COL_MAJOR>(20, 15)



(BLOCKED, NONE)

TilePattern<2, COL_MAJOR>(20, 15)



(TILE(5), TILE(5))

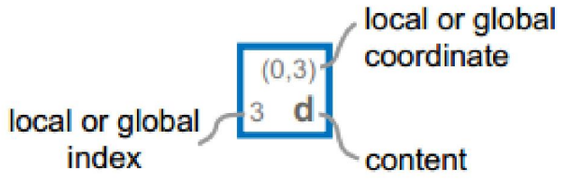
The N-Dimensional Array

- dash::NArray (dash::Matrix) offers a distributed multidimensional array abstraction
  - Dimension is a template parameter
  - Element access using coordinates or linear index
  - Support for custom index types
  - Support for row-major and column-major storage

```
dash::NArray<int, 2> mat(40, 30); // 1200 elements

int a = mat(i,j);   // Fortran style access
int b = mat[i][j];  // chained subscripts

auto loc = mat.local;

int c = mat.local(i,j);
int d = *(mat.local.begin()); // local iterator
```

DASH NArray Global View and Local View

- Local view works similar to 1D array



Global View

(0,0) 0 a	(0,1) 1 b	(0,2) 2 c	(0,3) 3 d
(1,0) 4 e	(1,1) 5 f	(1,2) 6 g	(1,3) 7 h
(2,0) 8 i	(2,1) 9 j	(2,2) 10 k	(2,3) 11 l
(3,0) 12 m	(3,1) 13 n	(3,2) 14 o	(3,3) 15 p
(4,0) 16 q	(4,1) 17 r	(4,2) 18 s	(4,3) 19 t
(5,0) 20 u	(5,1) 21 v	(5,2) 22 w	(5,3) 23 x
(6,0) 24 y	(6,1) 25 z	(6,2) 26 A	(6,3) 27 B

```
dash::NArray<char, 2> mat(7, 4);
cout << mat(2, 1) << endl;  // prints 'j'

if(dash::myid()==1 ) {
  cout << mat.local(2, 1) << endl; // prints 'z'
}
```

Local View (Unit 0)

(0,0) 0 a	(0,1) 1 b	(0,2) 2 c	(0,3) 3 d
(1,0) 4 m	(1,1) 5 n	(1,2) 6 o	(1,3) 7 p
(2,0) 8 y	(2,1) 9 z	(2,2) 10 A	(2,3) 11 B

Local View (Unit 1)

(0,0) 0 e	(0,1) 1 f	(0,2) 2 g	(0,3) 3 h
(1,0) 4 q	(1,1) 5 r	(1,2) 6 s	(1,3) 7 t

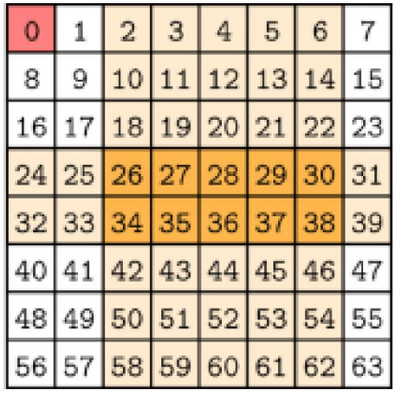
Local View (Unit 2)

(0,0) 0 i	(0,1) 1 j	(0,2) 2 k	(0,3) 3 l
(1,0) 4 u	(1,1) 5 v	(1,2) 6 w	(1,3) 7 x

Multidimensional Views

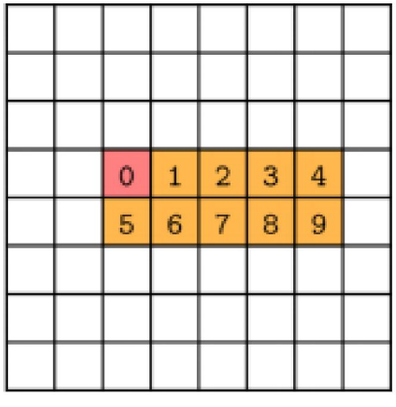
- DASH offers lightweight multidimensional views

mat.begin()




```
// 8x8 2D array
dash::NArray<int, 2> mat(8,8);

// linear access using iterators
dash::distance(mat.begin(), mat.end()) == 64

// create 2x5 region view
auto reg = matrix.cols(2,5).rows(3,2);
```

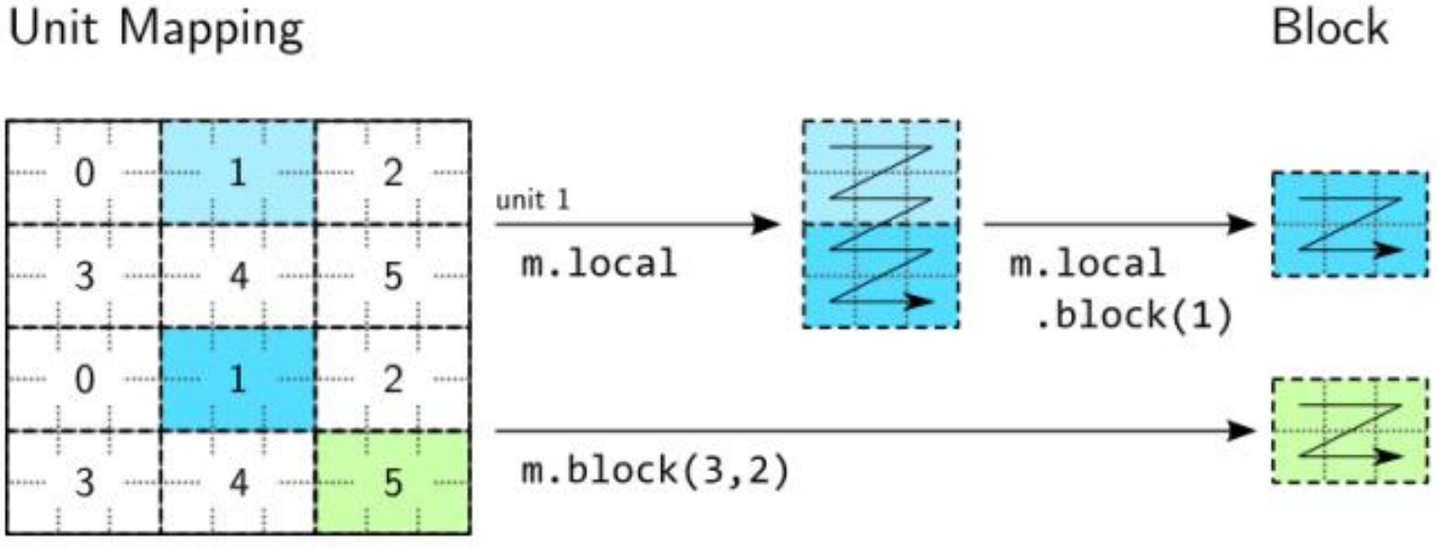
mat.cols(2,7)
.rows(3,5)
.begin()




```
// region can be used just like 2D array
cout << reg (0,0) << endl;

dash::distance(reg.begin(), reg.end()) == 10
```

Block Views

- dash::NArray can be accessed by blocks



DASH Algorithms (1)

- There are a few DASH equivalents for STL algorithms, e.g., dash::**fill**, dash::**for_each**, etc.

- Example: Set all elements in the range to ‘val’

```
dash::GlobIter<T> dash::fill(GlobIter<T> begin,
                             GlobIter<T> end, T val);
```

- Implementation:
  - Perform a projection of global range to local range
  - Apply STL algorithm (e.g., std::fill) on local range
  - Combine results when needed (e.g., dash::min_element)

DASH Algorithms (2)

- Examples
  - dash::fill — arr[i] <- val
  - dash::generate — arr[i] <- func()
  - dash::for_each — func(arr[i])
  - dash::transform — arr2[i] = func(arr1[i])
  - dash::accumulate — sum(arr[i]) (0<=i<=n-1)
  - dash::min_element — min(arr[i]) (0<=i<=n-1)
  - dash::max_element — min(arr[i]) (0<=i<=n-1)

DASH Algorithms (3)

- Example: Find the min. element in a distributed array

```
dash::Array<int> arr(100, dash::BLOCKED);

for( auto i=0; i<arr.lsize(); i++ ) {
  arr.local[i]=rand()%100;
}
arr.barrier();

auto min = dash::min_element(arr.begin(),
                             arr.end());

if( dash::myid()==0 ) {
  cout<<"Minimum: "<<(int)*min<<endl;
}
```

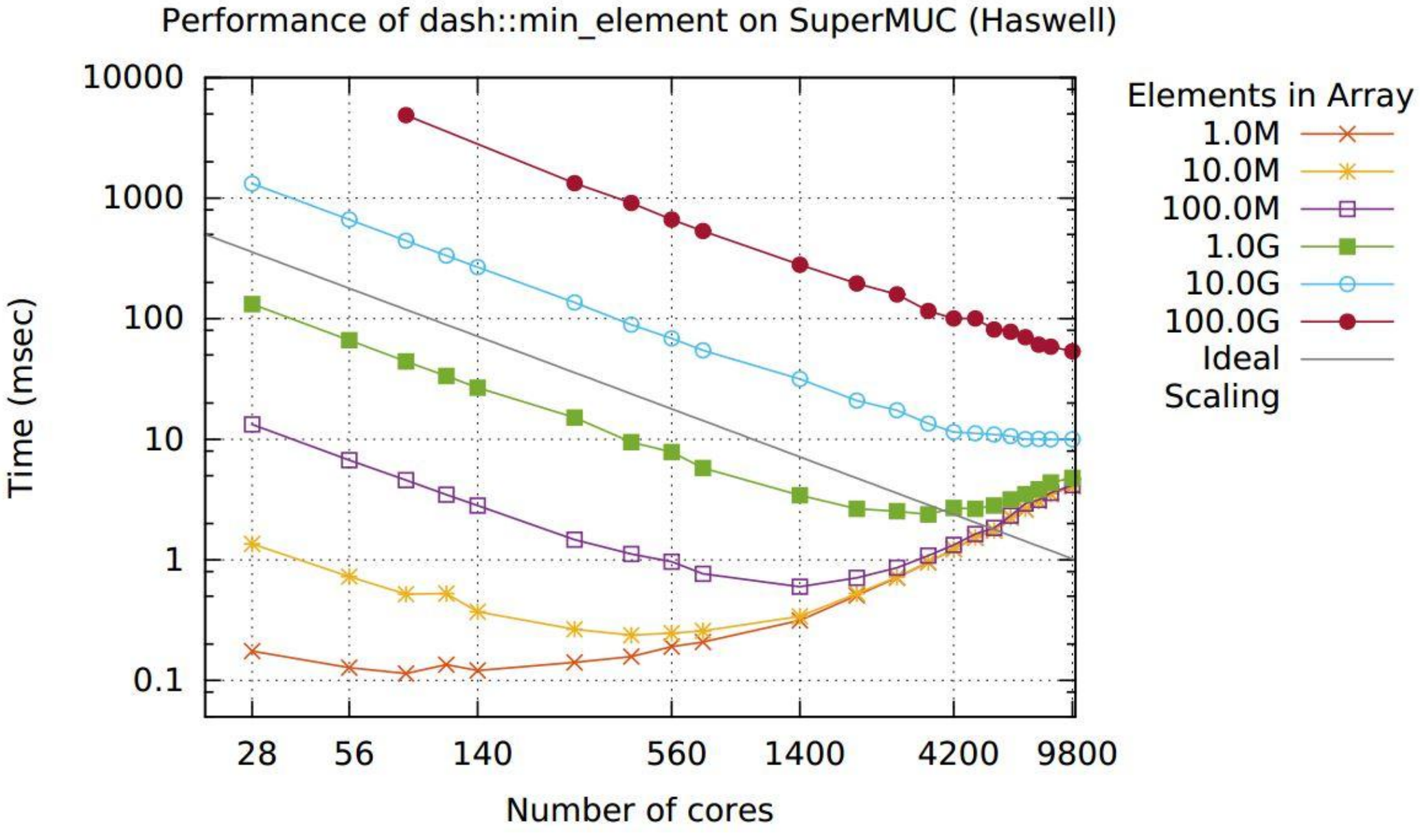
Collective call, returns global pointer To min. element

Identify local range and call std::min_element

Get the min. element value and print it

- Features
  - Still works when using CYCLIC or any other distribution
  - Still works when using a range other than [begin, end)

Performance of dash::min_element()



HDF5 Input/Output

- Support of direct parallel I/O of dash::Array and dash::NArray
  - Pattern is stored as meta data and can be restored from HDF5 file

```
// write array
dash::Array<double> arr1(1000);
dash::io::HDF::write(arr1, "file.hdf5", "dataset");

// read into array
dash::Array<double> arr2;
dash::io::HDF::read(arr2, "file.hdf5", "dataset");

// use stream interface instead
dash::io::HDFOutputStream os("outfile.hdf5");
os << dash::io::HDF5Dataset("data") << arr1 ;
```

Asynchronous Communication

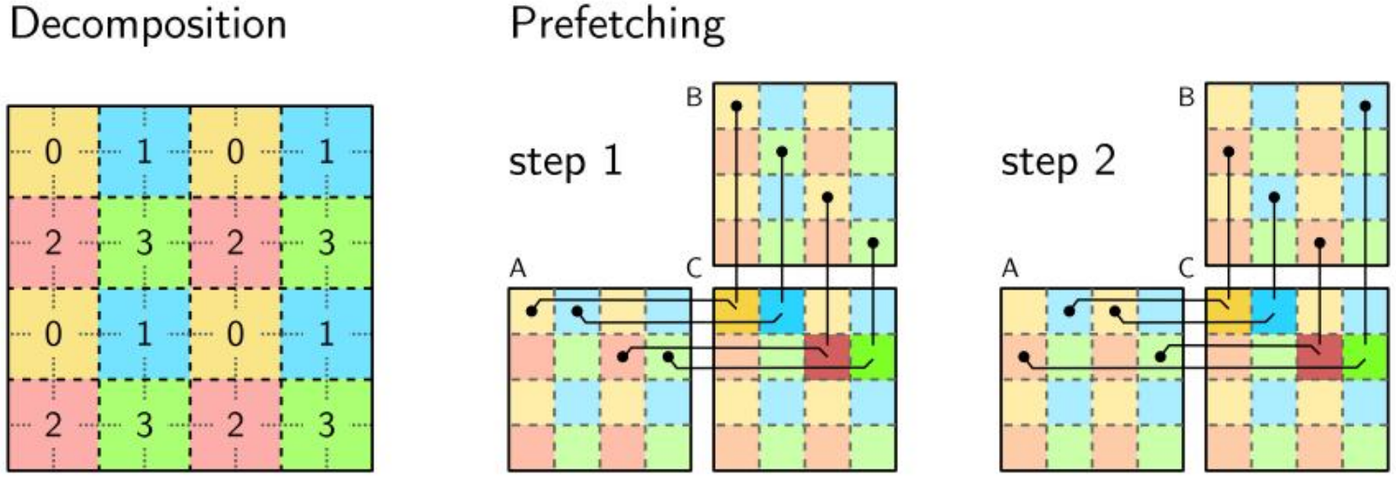
- Realized via two mechanisms
  - Async. copy operation (dash::copy_async())
  - .async proxy object on DASH containers

```
// async. copy of global range to local memory
auto fut = dash::copy_async(block.begin(), block.end(),
                            local_ptr);
...
fut.wait();
```

```
for (i = dash::myid(); i < NUPDATE; i += dash::size()) {
  ran = (ran << 1) ^ (((int64_t) ran < 0) ? POLY : 0);
  // using .async proxy object
  // .async proxy object for async. communication
  Table.async[ran & (TableSize-1)] ^= ran;
}
Table.flush();
```

Application Example: S(R)UMMA Algorithm

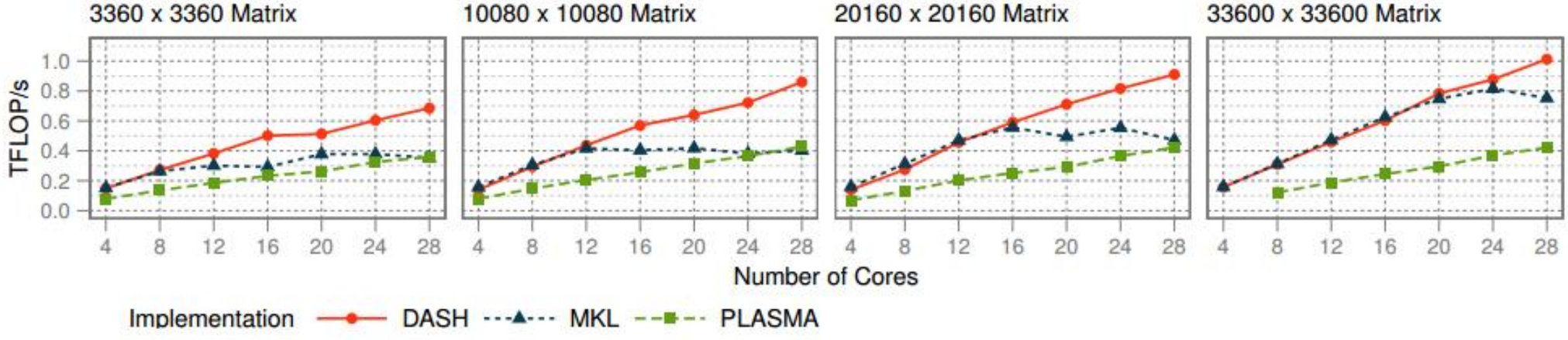
- Block matrix-matrix mulitplication algorithm with block prefetching



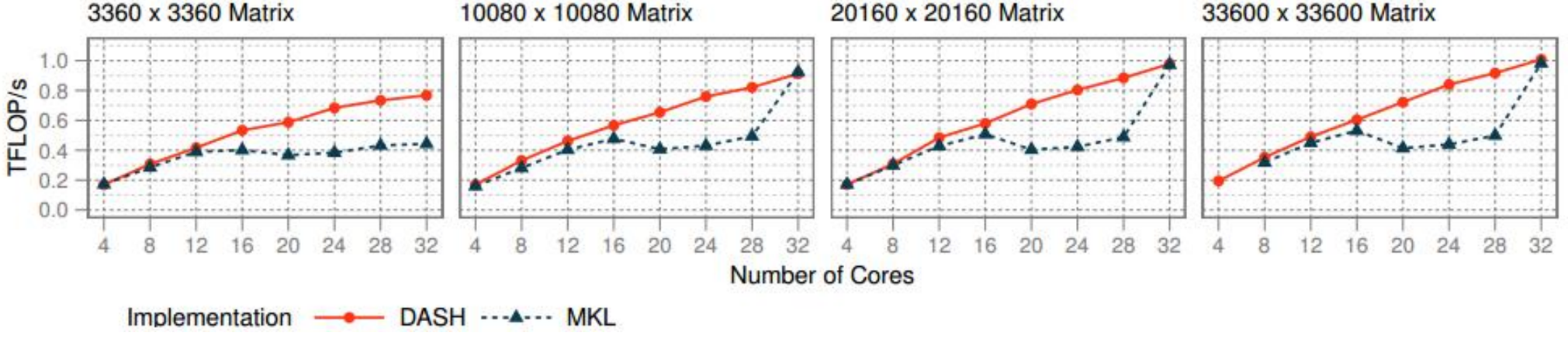

```
while(!done) {
  blk_a = ...
  blk_b = ...
  // prefetch
  auto get_a = dash::copy_async(blk_a.begin(), blk_a.end(), lblk_a_get);
  auto get_b = dash::copy_async(blk_b.begin(), blk_b.end(), lblk_b_get);
  // local DGEMM
  dash::multiply(lblk_a_comp, lblk_b_comp, lblk_c_comp);
  // wait for transfer to finish
  get_a.wait(); get_b.wait();
  // swap buffers
  swap(lblk_a_get, lblk_a_comp);  swap(lblk_b_get, lblk_b_comp);
}
```

DGEMM on a Single Shared Memory Node

- LRZ SuperMUC, phase 2: Haswell EP, 1.16 Tflop/sec



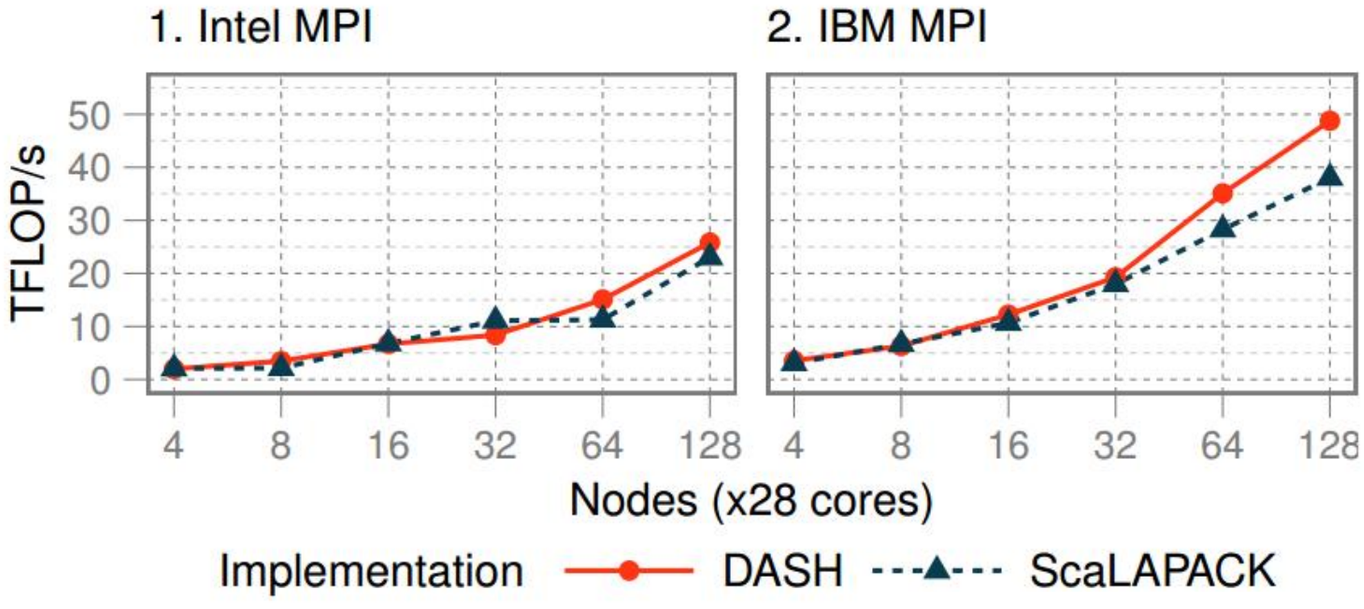
- NERSC Cori, phase 1: Haswell EP, 1.17 Tflop/sec peak



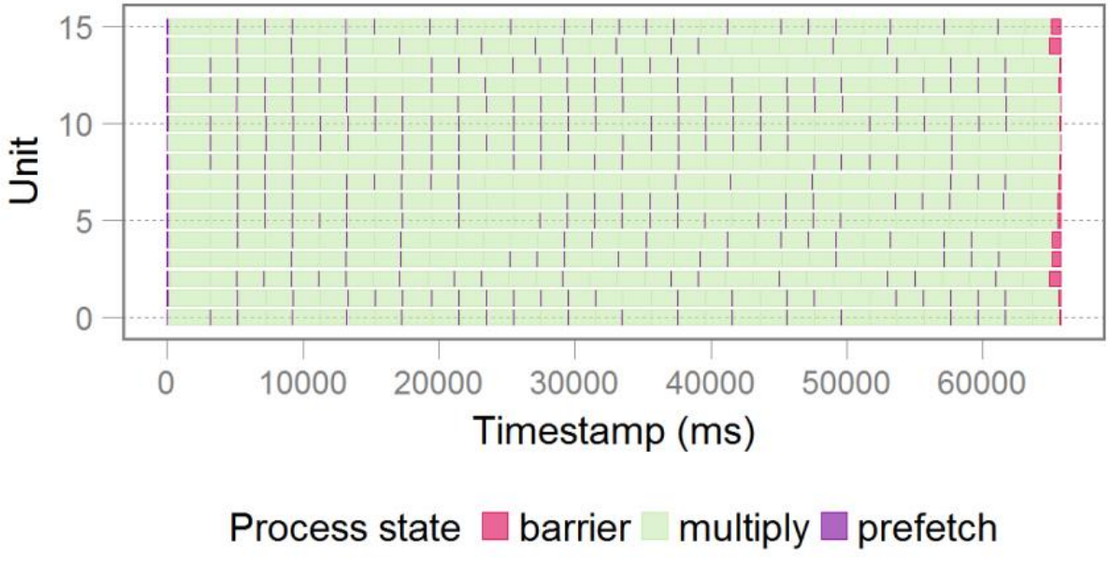
Tobias Fuchs and Karl Fürlinger. **A Multi-Dimensional Distributed Array Abstraction for PGAS**. In *Proceedings of the 18th IEEE International Conference on High Performance Computing and Communications (HPCC 2016)*. Sydney, Australia, December 2016. accepted for publication

PDGEMM: DASH vs. ScaLAPACK Multinode

- Strong scaling on SuperMUC



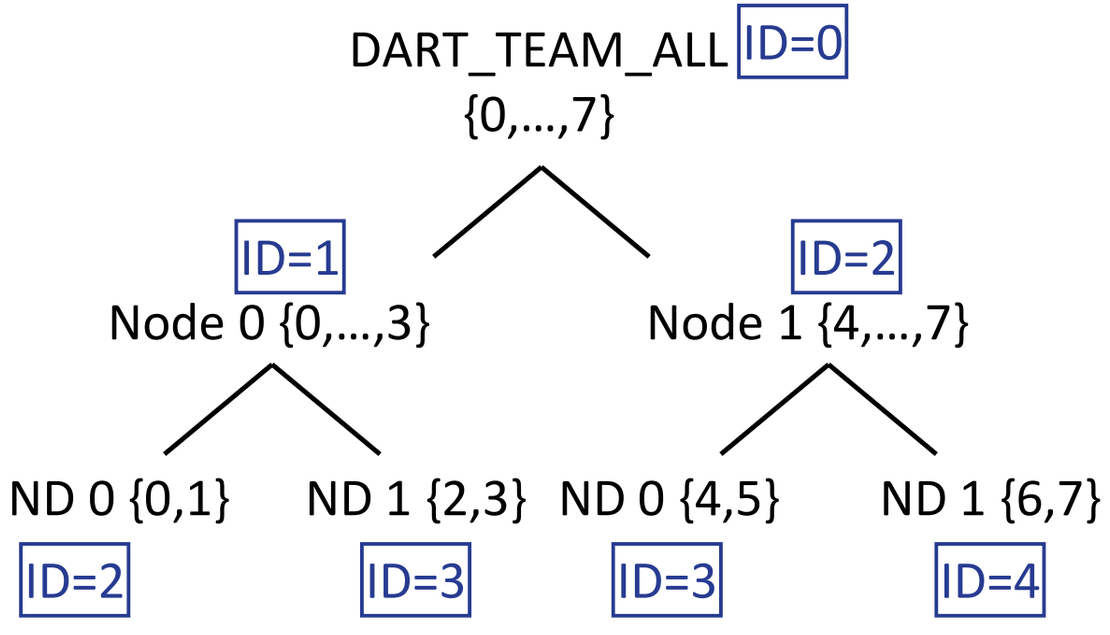
- Trace: Overlapping communication and computation



Teams

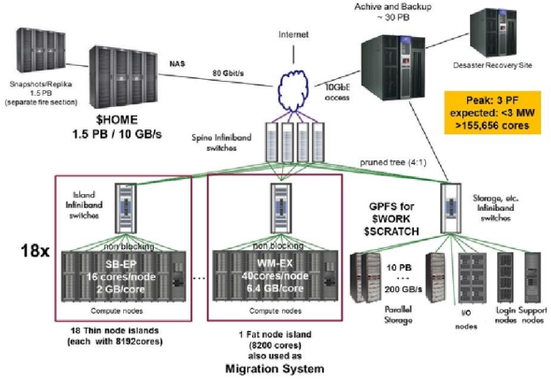
- Teams are everywhere in DASH (but not always visible)
  - Team: ordered set of units
  - Default team: dash::Team::All() - all units that exist at startup
  - team.split(N) – split an existing team into N sub-teams

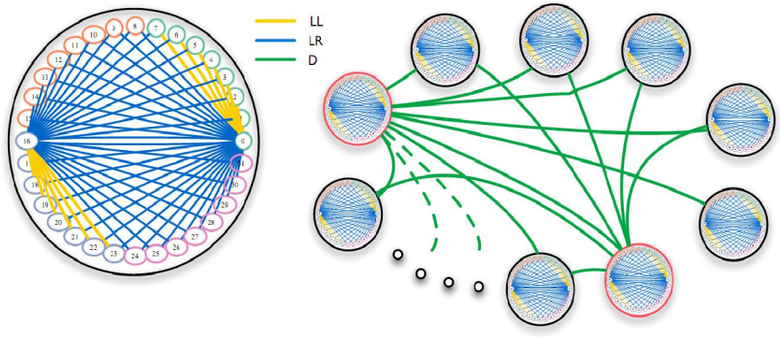
```
dash::Team& t0 = dash::Team::All();

dash::Array<int> arr1(100);
// same as
dash::Array<int> arr2(100, t0);

// allocate array over t1
auto t1 = t0.split(2)
dash::Array<int> arr3(100, t1);
```

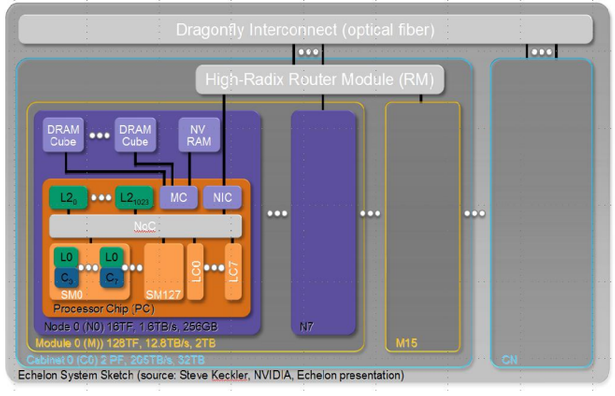


Hierarchical Machines

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







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

Porting LULESH (Work in Progress)

- LULESH: A shock-hydrodynamics mini-application

```
class Domain // local data
{
private:
std::vector<Real_t> m_x;
// many more...

public:
Domain() { // C'tor
  m_x.resize(numNodes);
  //...
}

Real_t& x(Index_t idx) {
  return m_x[idx];
}
};
```

MPI Version

```
class Domain // global data
{
private:
dash::NArray<Real_t, 3> m_x;
// many more...

public:
Domain() { // C'tor
  dash::Pattern<3> nodePat(
     nx()*px(), ny()*py(), nz()*pz(),
     BLOCKED, BLOCKED, BLOCKED);
  m_x.allocate(nodePat);
}

Real_t& x(Index_t idx) {
  return m_x.lbegin()[idx];
}
};
```

DASH Version

Porting LULESH (goals)

- Remove limitations of MPI domain decomposition
  - Cubic number of MPI processes only (P x P x P)
  - Cubic per-processor grid
  - DASH allows any P x Q x R configuration

- Avoid replication, manual index calculation, bookkeeping

```
if (rowMin | rowMax) {
   /* ASSUMING ONE DOMAIN PER RANK, CONSTANT BLOCK SIZE HERE */
   int sendCount = dx * dz ;

   if (rowMin) {
      destAddr = &domain.commDataSend[pmsg * maxPlaneComm] ;
      for (Index_t fi=0; fi<xferFields; ++fi) {
         Domain_member src = fieldData[fi] ;
         for (Index_t i=0; i<dz; ++i) {
            for (Index_t j=0; j<dx; ++j) {
               destAddr[i*dx+j] = (domain.*src)(i*dx*dy + j) ;
            }
         }
         destAddr += sendCount ;
      }
      destAddr -= xferFields*sendCount ;

      MPI_Isend(destAddr, xferFields*sendCount, baseType,
                myRank - domain.tp(), msgType,
                MPI_COMM_WORLD, &domain.sendRequest[pmsg]) ;
      ++pmsg ;
```

implicit assumptions

manual index calculation

manual bookkeeping

Replicated about 80 times

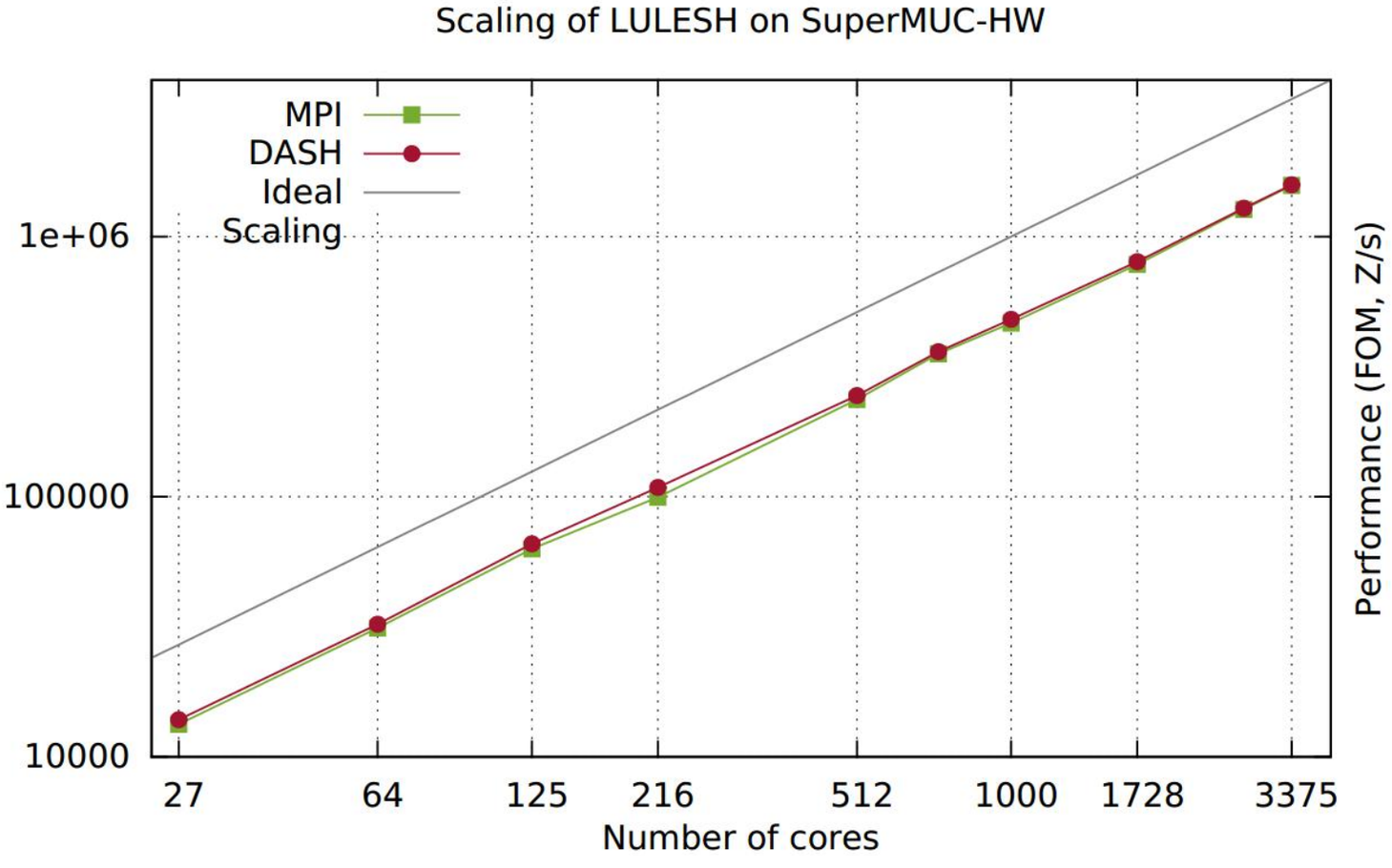
Porting LULESH (options)

- DASH can be integrated in existing applications and allows for incremental porting

- Porting options:
  1. Port data structures, but keep communication as-is (using MPI two-sided)
     - Can use HDF5 writer for checkpointing
  2. Keep explicit packing code but use one-sided put instead of MPI_Irecv/MPI_Isend
     - Similar to UPC++ version, potential performance benefit
  3. Use DASH for communication directly
     - auto halo = …; dash::swap(halo) …
     - less replicated code, more flexibility
  4. Use dash::HaloNArray
     - N-dimensional array with built-in halo areas

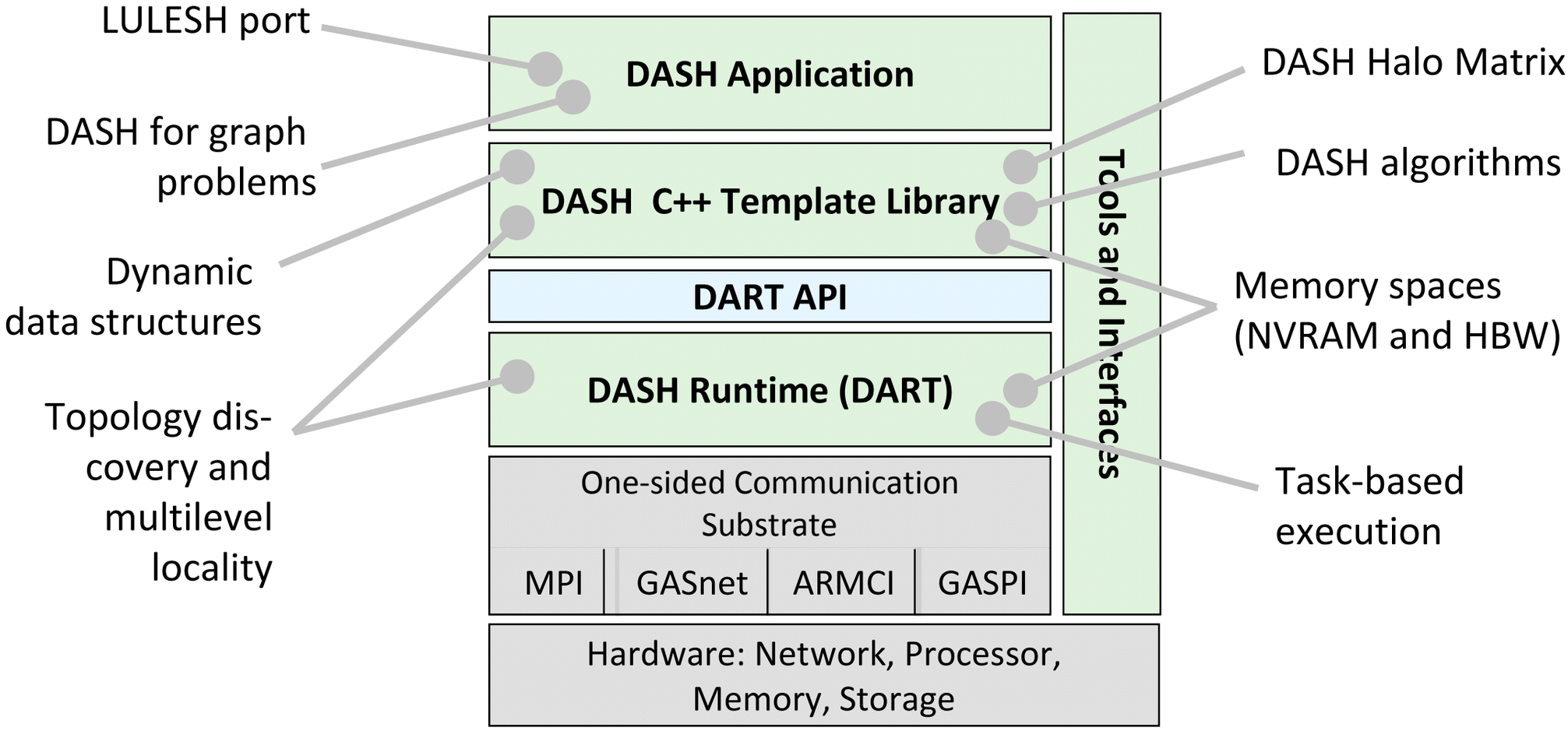
Performance of the DASH Version (using put)

- Performance and scalability (weak scaling) of LULESH, implemented in MPI and DASH



Karl Fürlinger, Tobias Fuchs, and Roger Kowalewski. **DASH: A C++ PGAS Library for Distributed Data Structures and Parallel Algorithms**. In *Proceedings of the 18th IEEE International Conference on High Performance Computing and Communications (HPCC 2016)*. Sydney, Australia, December 2016. accepted for publication.

DASH: On-going and Future Work



Summary

- DASH is
  - A complete data-oriented PGAS programming system (i.e., entire applications can be written in DASH),
  - A library that provides distributed data structures (i.e., DASH can be integrated into existing MPI applications)

- More information
  - http://www.dash-project.org/

Acknowledgements

- Funding

DFG Deutsche Forschungsgemeinschaft

Bundesministerium für Bildung und Forschung

- The DASH Team

  T. Fuchs (LMU), R. Kowalewski (LMU), D. Hünich (TUD), A. Knüpfer (TUD), J. Gracia (HLRS), C. Glass (HLRS), H. Zhou (HLRS), K. Idrees (HLRS), F. Mößbauer (LMU), K. Fürlinger (LMU)

- DASH is now on GitHub
  - https://github.com/dash-project/dash/
  - http://www.dash-project.org/