LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

DASH: A Data-centric PGAS C++ Template Library

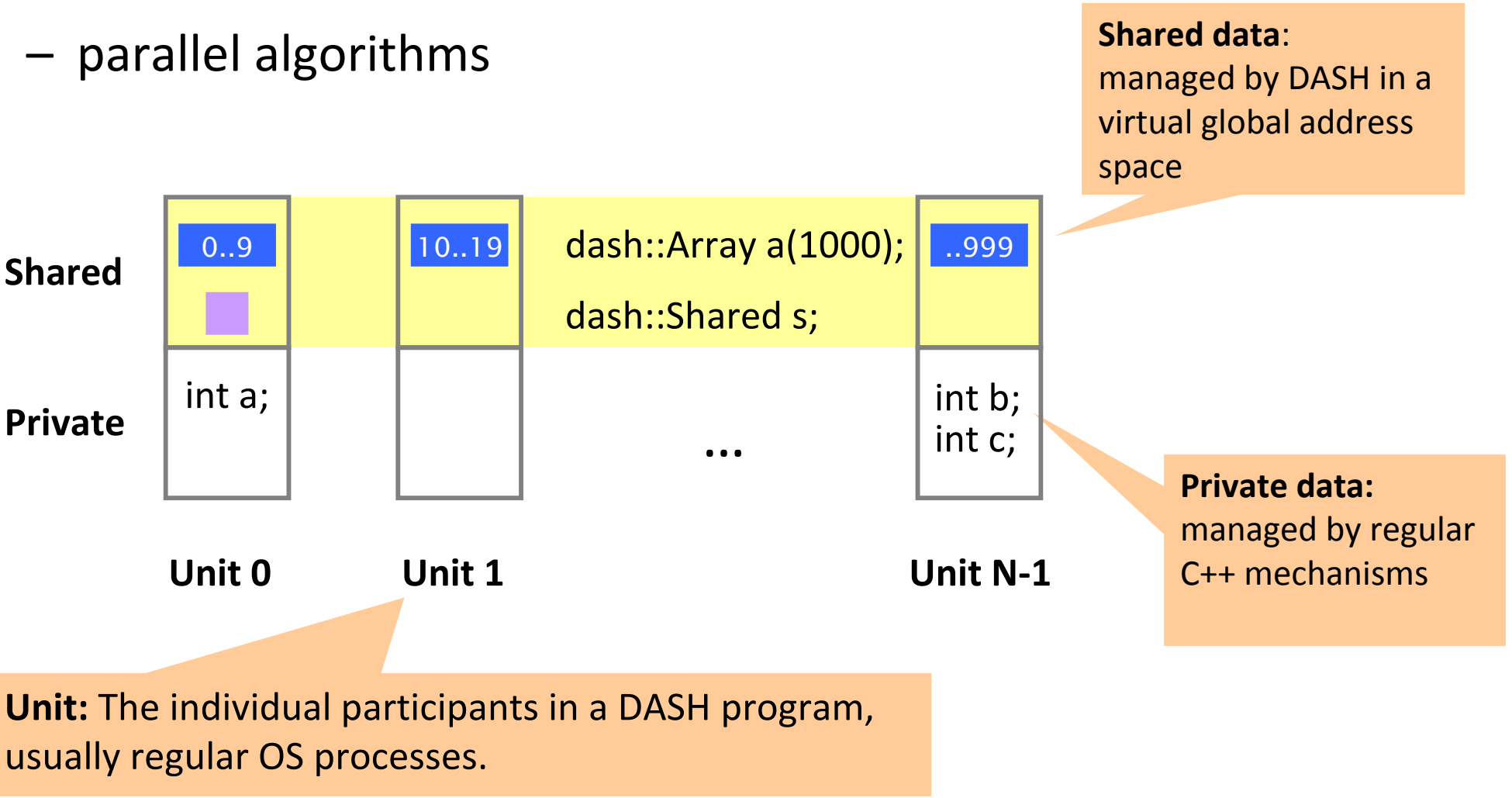
Karl Fürlinger, Tobias Fuchs,
Roger Kowalewski, Matthias Maiterth

MNM-Team
Ludwig-Maximilians-Universität München

http://www.dash-project.org

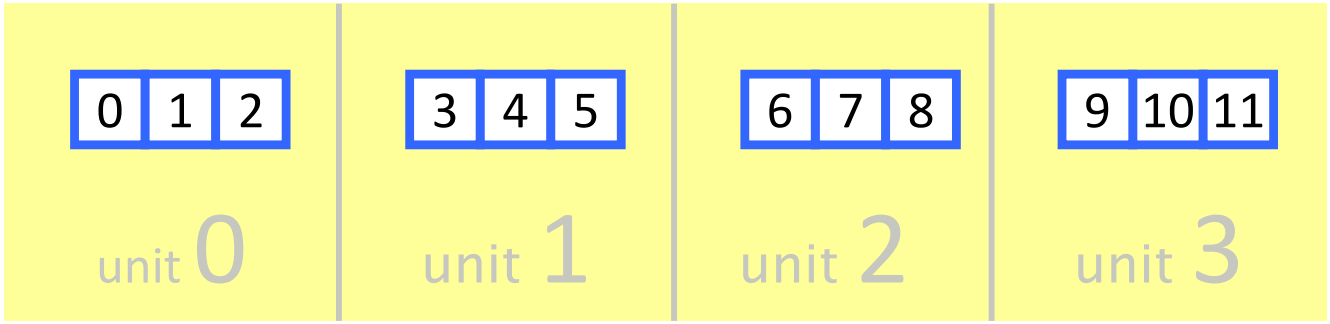
DASH - Overview

- DASH is a C++ template library that offers
  - a realization of the PGAS (part. global address space) concept
  - distributed data structures
  - parallel algorithms



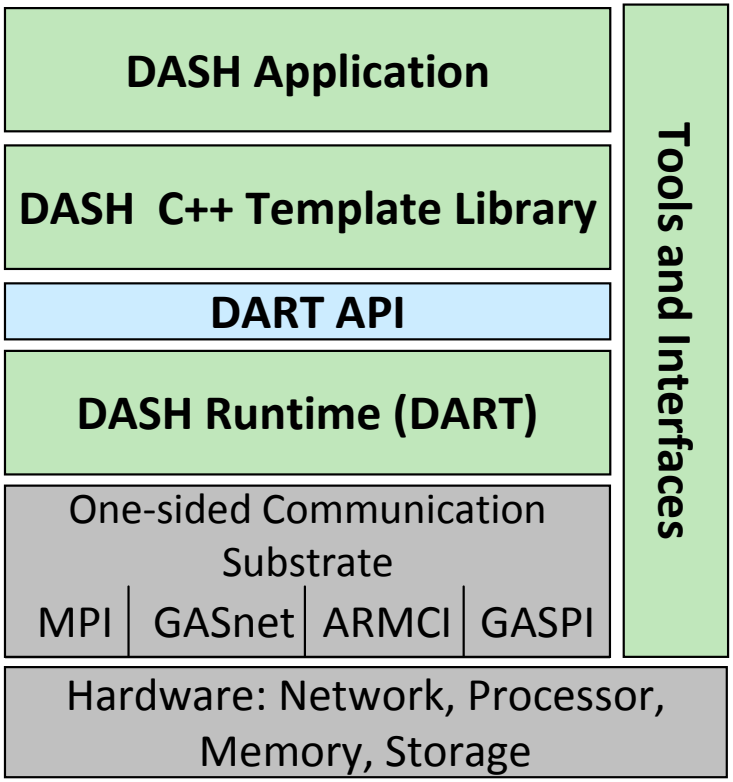
The "P" in PGAS

- Example: dash::Array with 12 elements



- Data elements can be accessed by any unit, but each unit has explicit and well defined affinity to a home unit
  - Locality important for performance
  - Support for the *owner computes* execution pattern

DASH Project Structure



	Phase I (2013-2015)	Phase II (2016-2018)
LMU Munich	Project management, C++ tempalte 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 is a SPPEXA project, funded by DFG

SPPEXA
www.sppexa.de

DFG
www.dfg.de

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

Find out number of units and our own ID

Print message; note SPMD model, like 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, with 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()==size-1 ) {
  for( auto el: arr )
    cout<<(int)el<<" ";
  cout<<endl;
}
```

Global DASH array of 100 integers, distributed over all units, default distribution: BLOCKED

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

The last unit 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 structure is exposed through a *local view* proxy object

```
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;
}
```

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

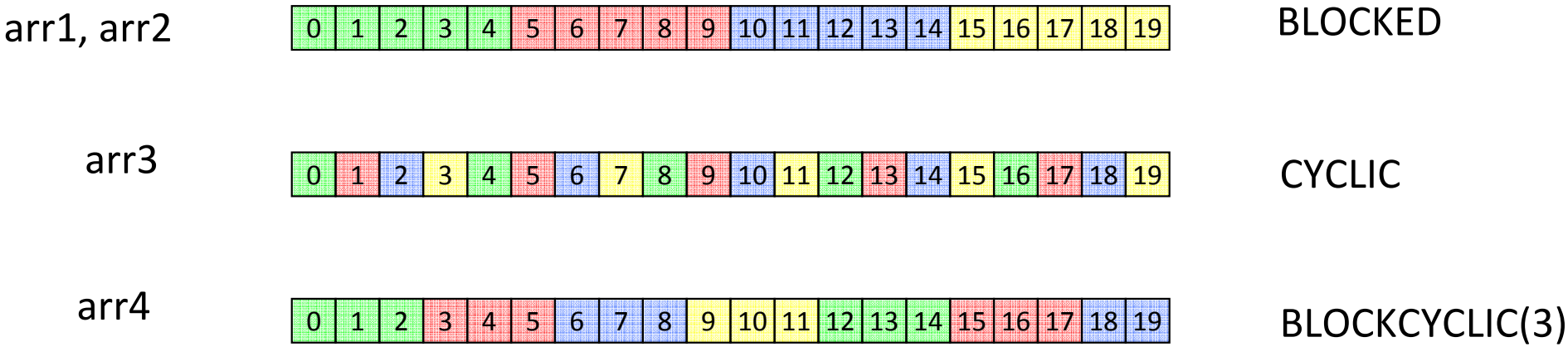
```
$ mpirun -n 4 ./array
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3
```

Data Distribution Patterns

- The distribution of data elements to units 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))
```

- Assuming 4 units



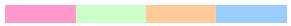
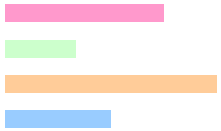
Accessing Local Data

- The previous example with a cyclic distribution:

```
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
```

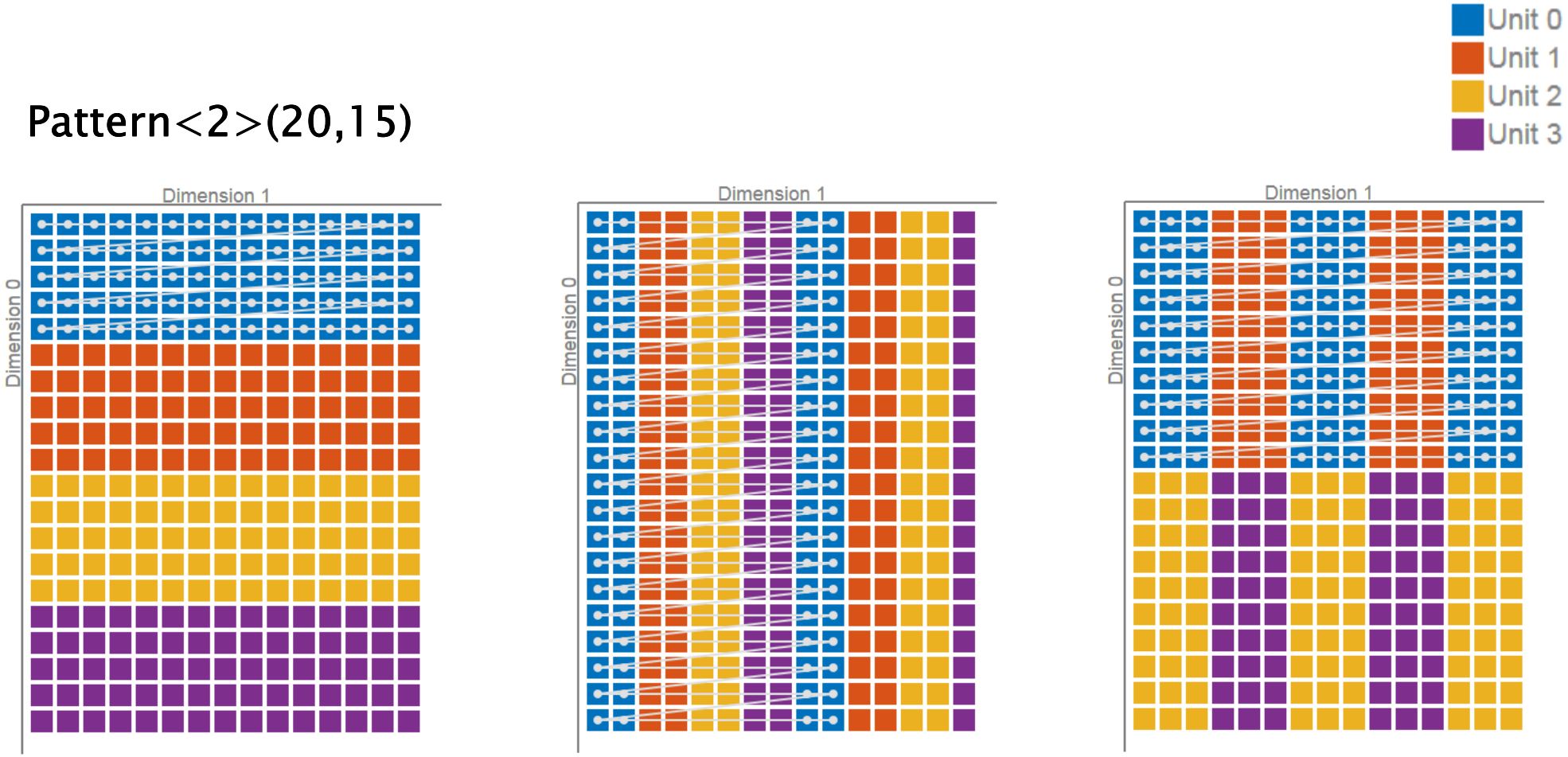
More Distributed Data Structures

Container	Description	Data distribution
Array<T>	1D Array 	static, flexible
Matrix<T, N>	N-dim. array 	static, flexible
Shared<T>	shared scalar 	fixed (at 0)
Directory<T>	variable-size, locally indexed array 	manual, variable

Multi-Dimensional Data Distribution (1)

- Blocked, cyclic, and block-cyclic in multiple dimensions

Pattern<2>(20,15)



BLOCKED, NONE

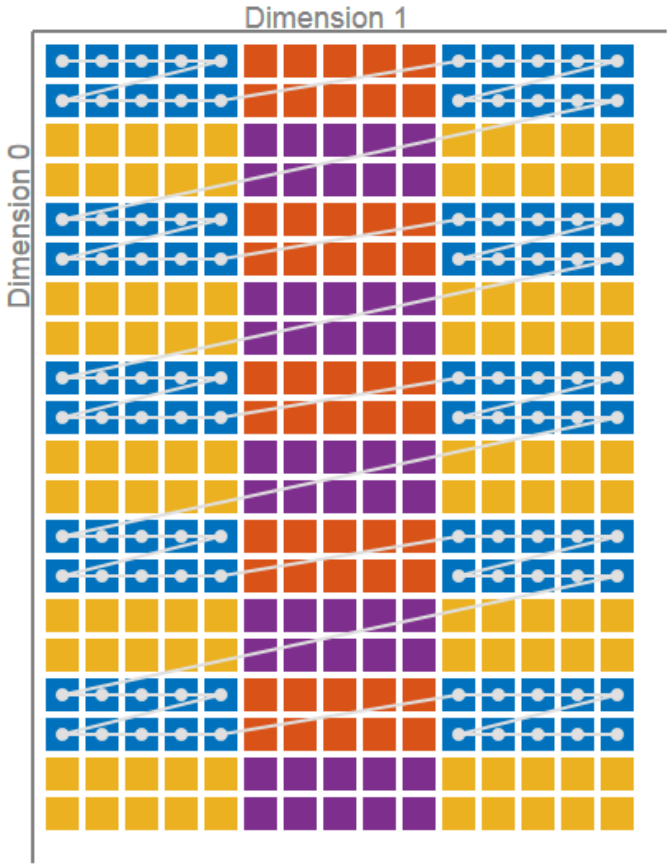
NONE, BLOCKCYCLIC(2)

BLOCKED, BLOCKCYCLIC(3)

Multi-Dimensional Data Distribution (2)

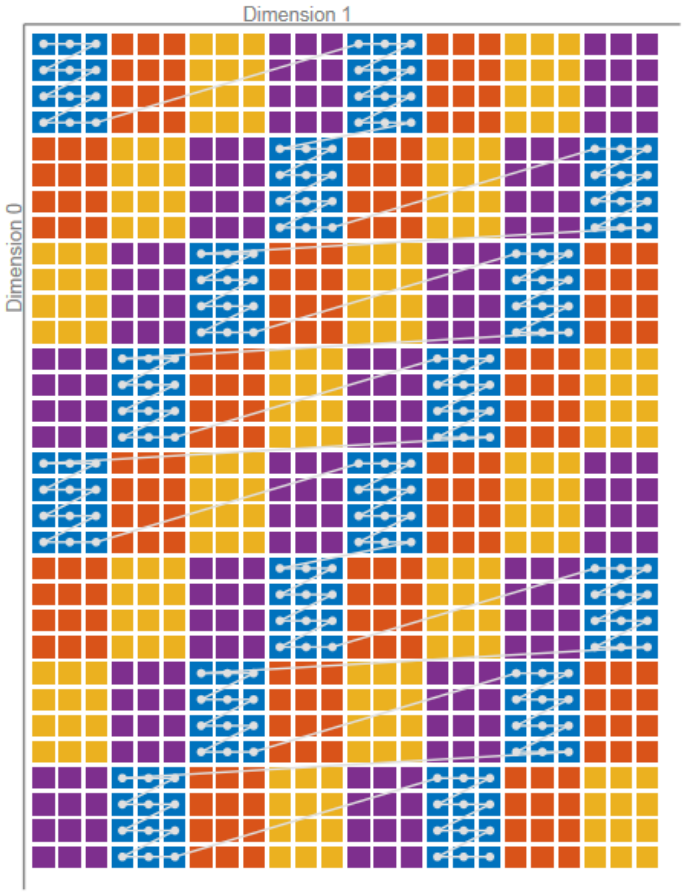
- Tiled data distribution and tile-shifted distribution

TilePattern<2>(20,15)



TILE(2), TILE(5)

ShiftTilePattern<2>(32,24)

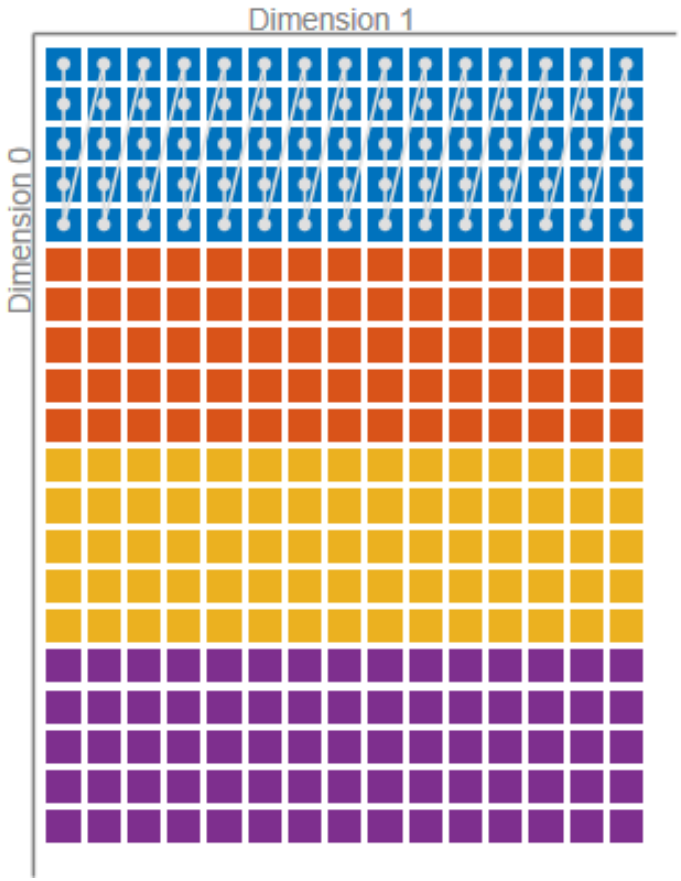


TILE(4), TILE(3)

Multi-Dimensional Data Distribution (3)

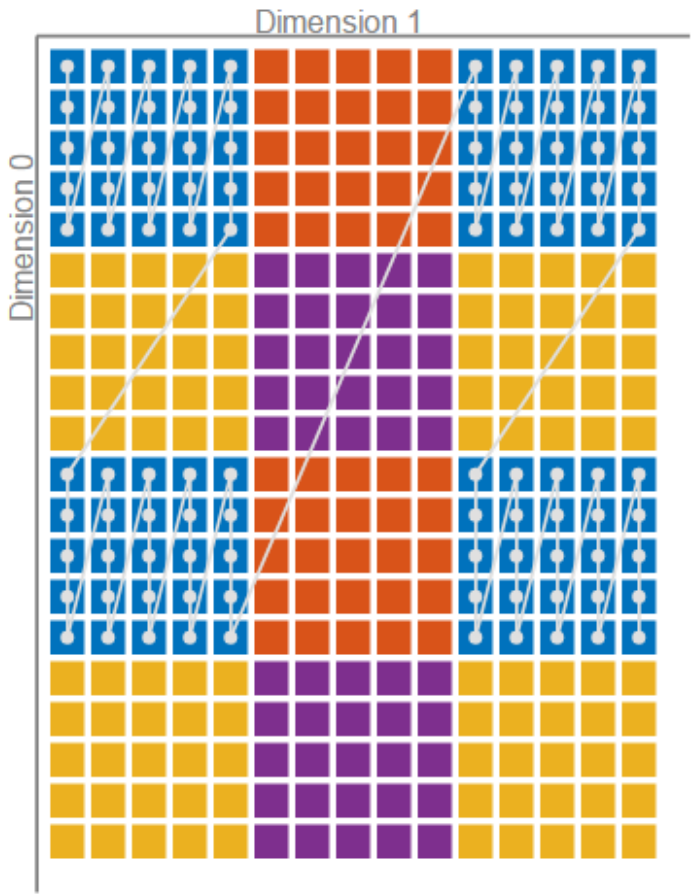
- Row-major and column-major storage

Pattern<2, COL_MAJOR>(20,15)



BLOCKED, NONE

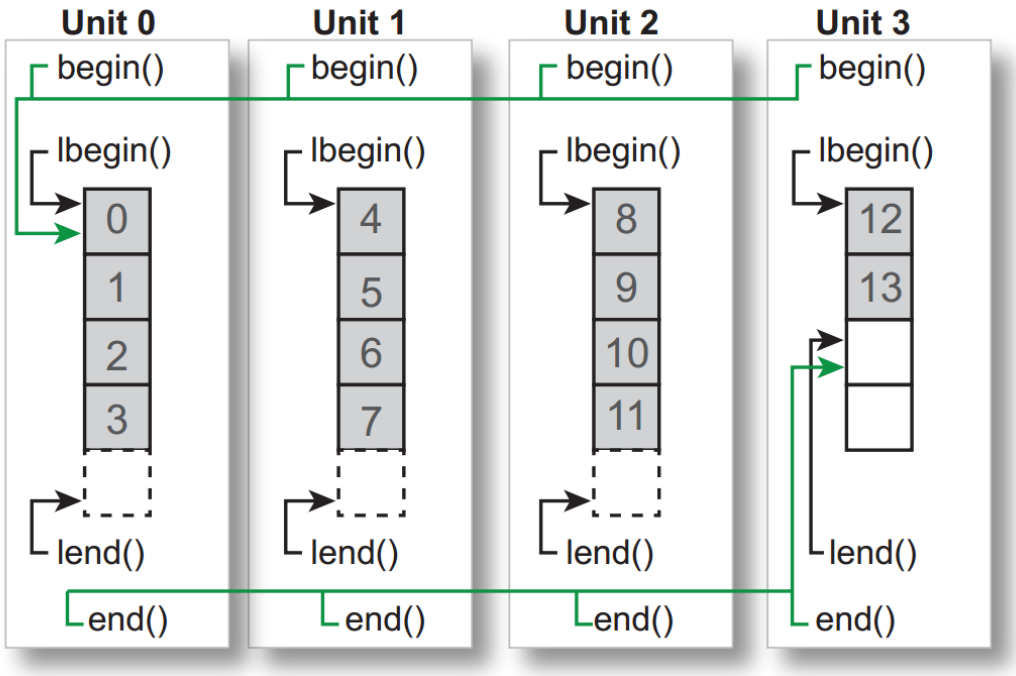
TilePattern<2, COL_MAJOR>(20,15)



TILE(5), TILE(5)

DASH Features: Fast Local Access

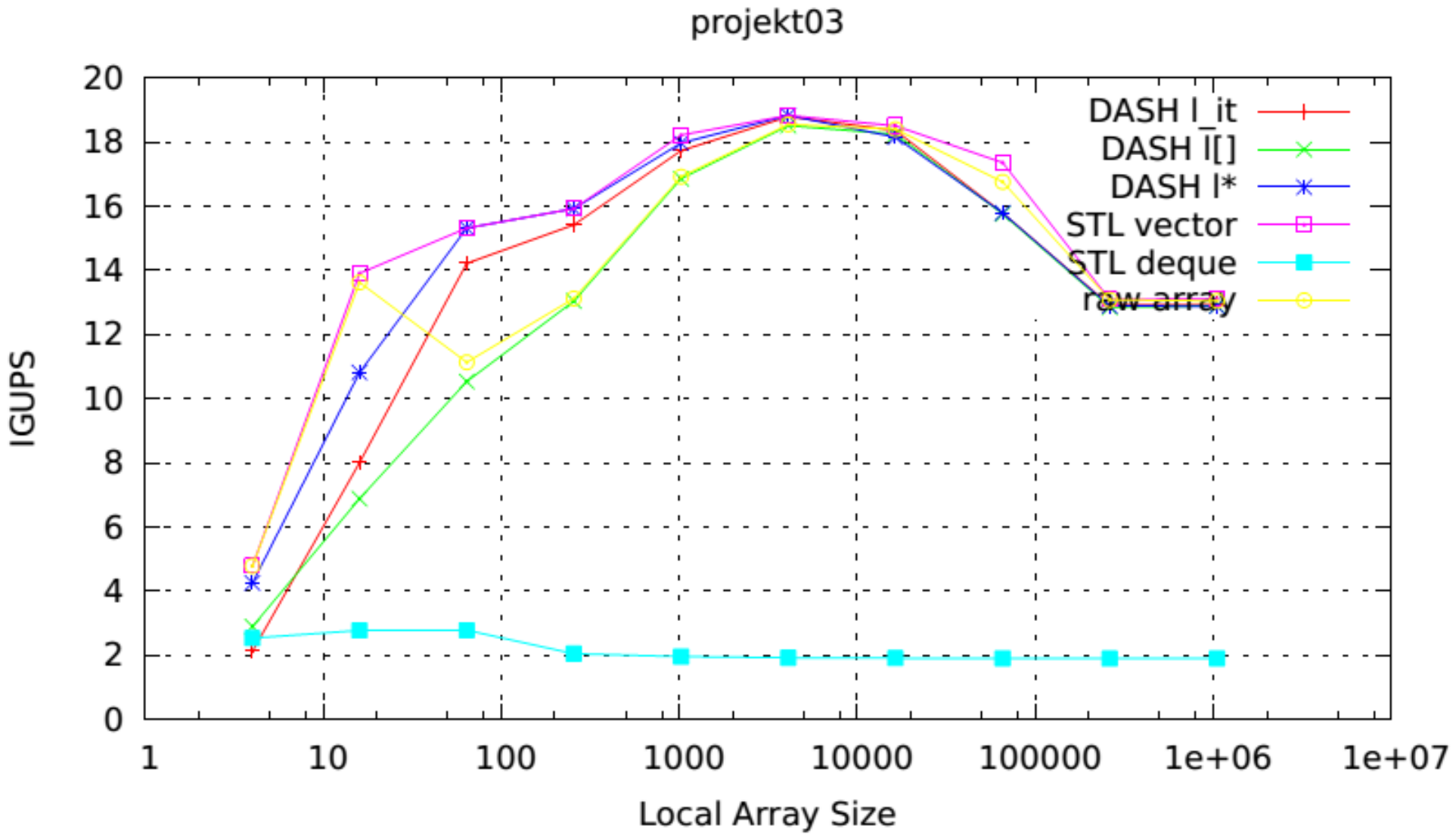
- Efficient local data access
  - Each unit exposes its local memory via a local proxy object
  - Zero overhead when working with local proxy object




```
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
for( auto el : arr.local )
  cout<<el<<" ";
```

DASH Features: Fast Local Access

- IGUPs Benchmark: independent parallel updates



DASH Features: Using STL Algorithms

```
#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

Access by global iterator, works with std. algorithms

Access by local iterator, works with std. algorithms as well

DASH Algorithms

- 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;
}
```

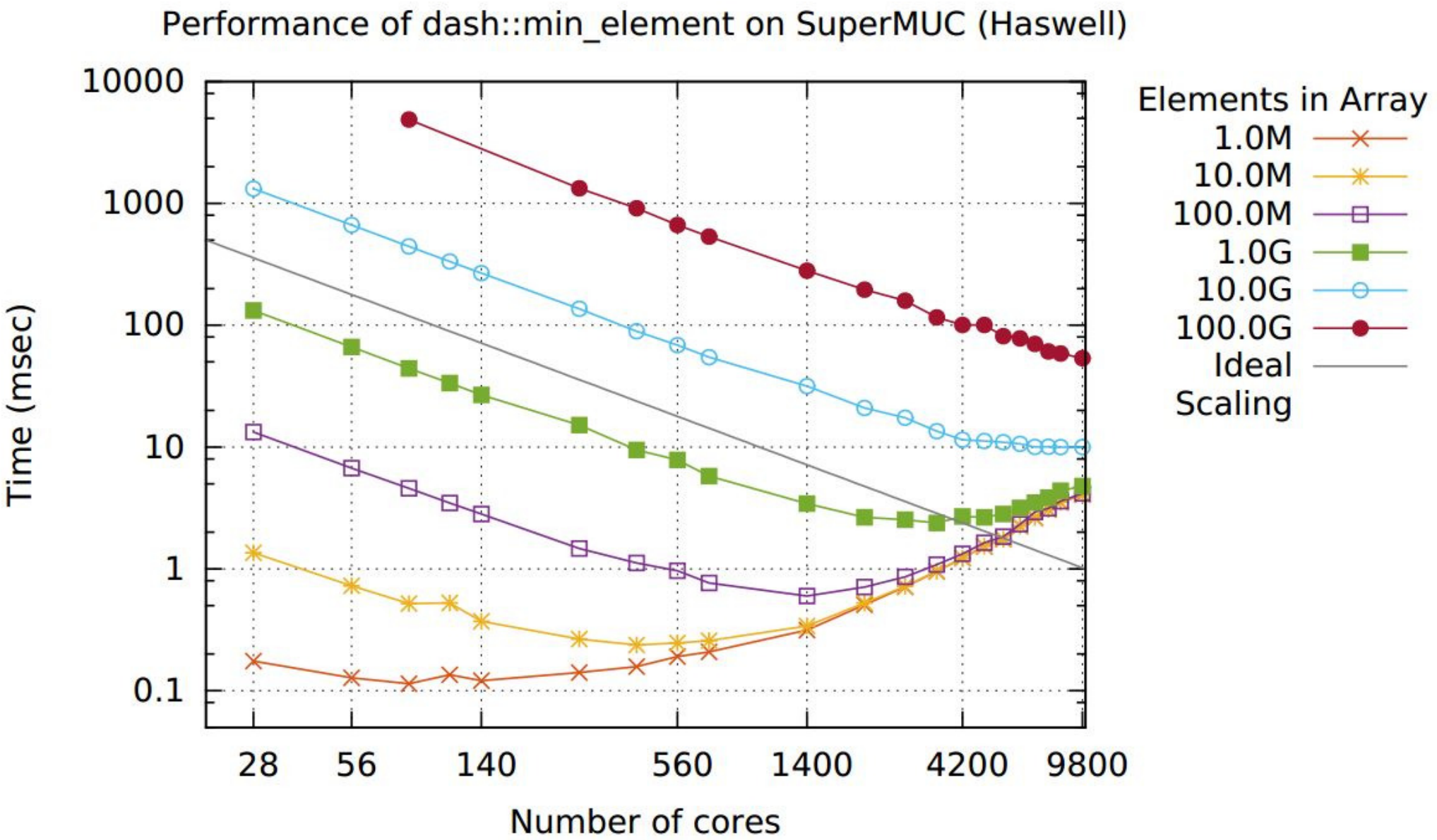
Collective call, returns global pointer To min. element

Identify local range and call std::min_element

Get the min. element and print it

- Features
  - Still works when using CYCLIC or any other distribution
  - Still works when using a range other than [begin, end)
  - Still works when using a data type other than int (even composite types)

Performance of dash::min_element()



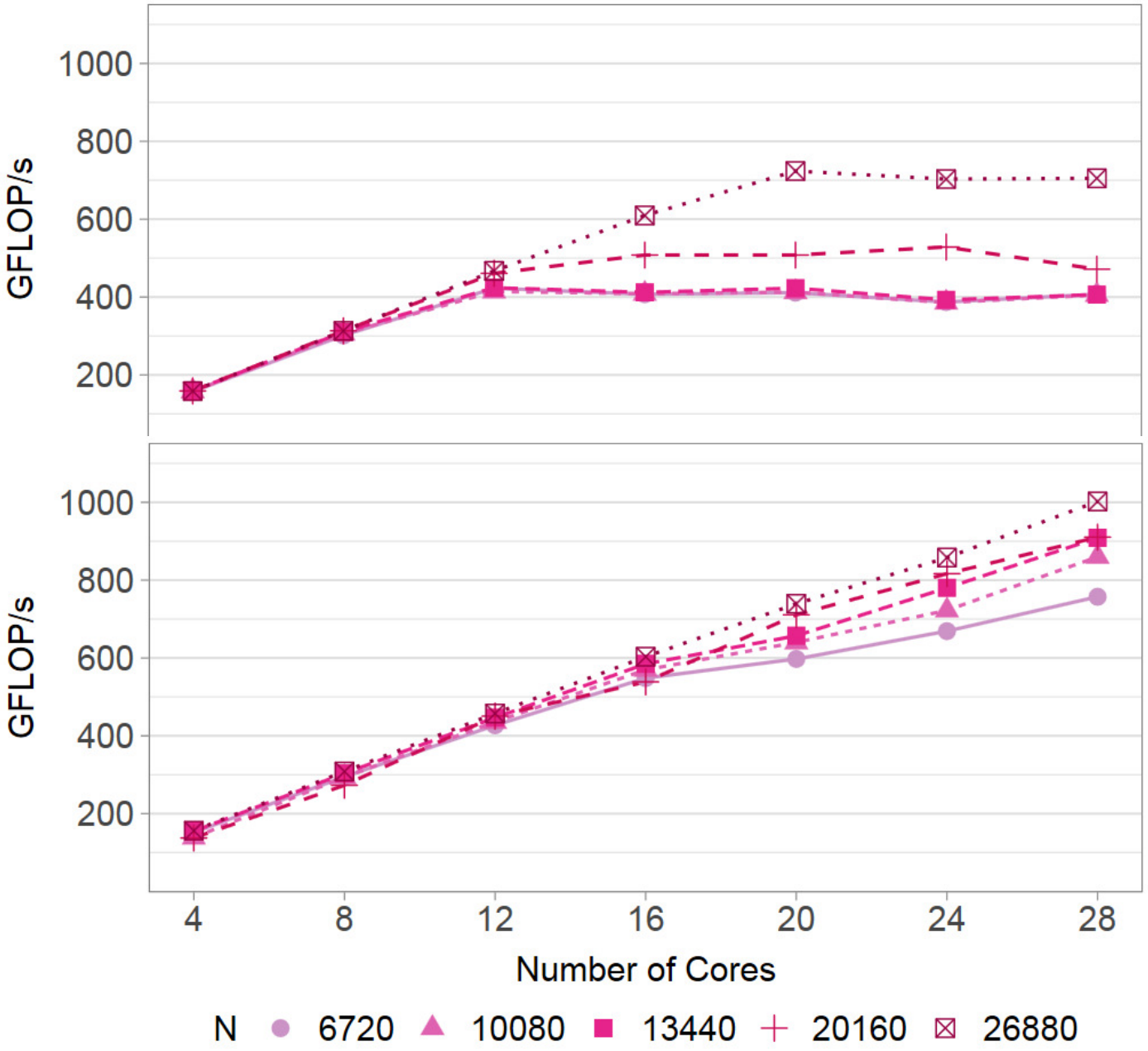
Features: Asynchronous Communication

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

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

```
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();
```

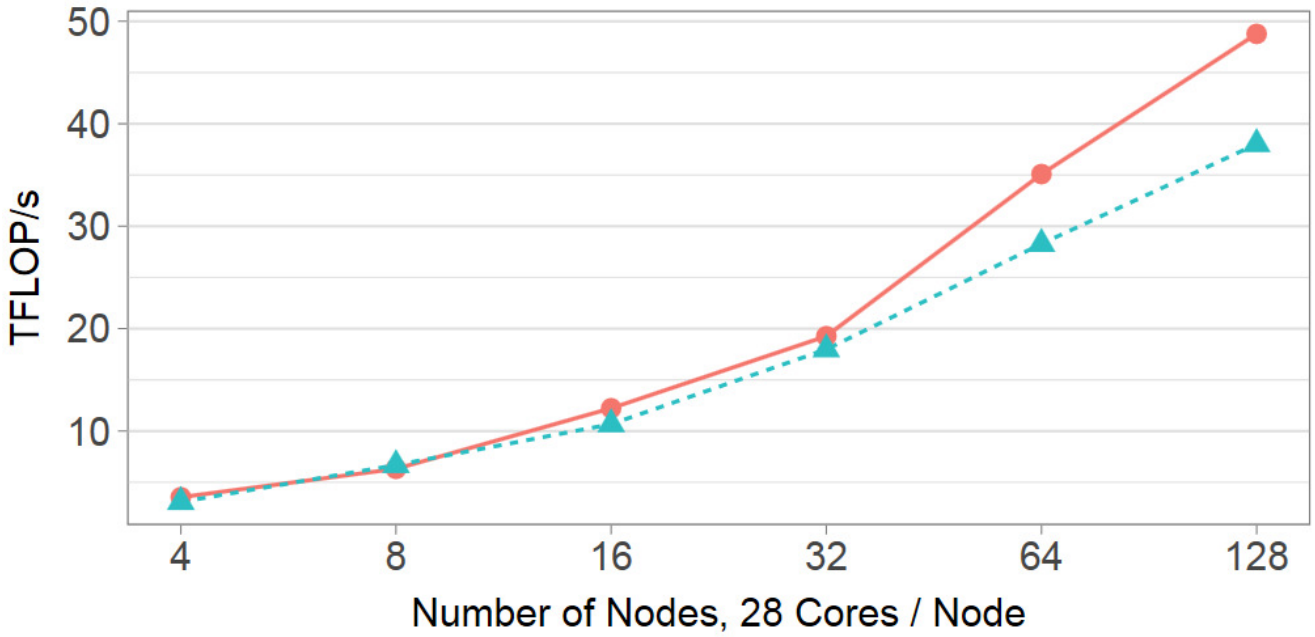
DGEMM on a Single Shared Memory Node

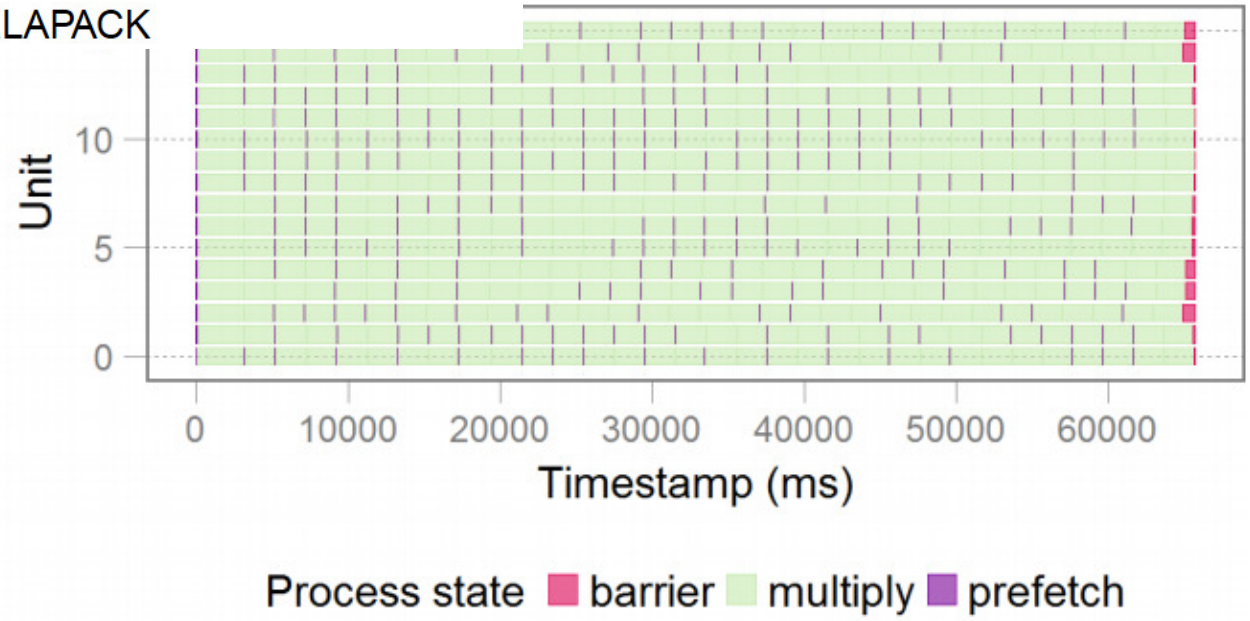


Multi-threaded MKL

dash::summa using single-threaded MKL

DASH summavs. ScaLAPACK, Multiple Nodes





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/
  - Version v0.2.0 available for download

- 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)