# New Accelerations for Parallel Programming

HPC Advisory Council – Spain 2012

Todd Wilde – Director of Technical Computing and HPC

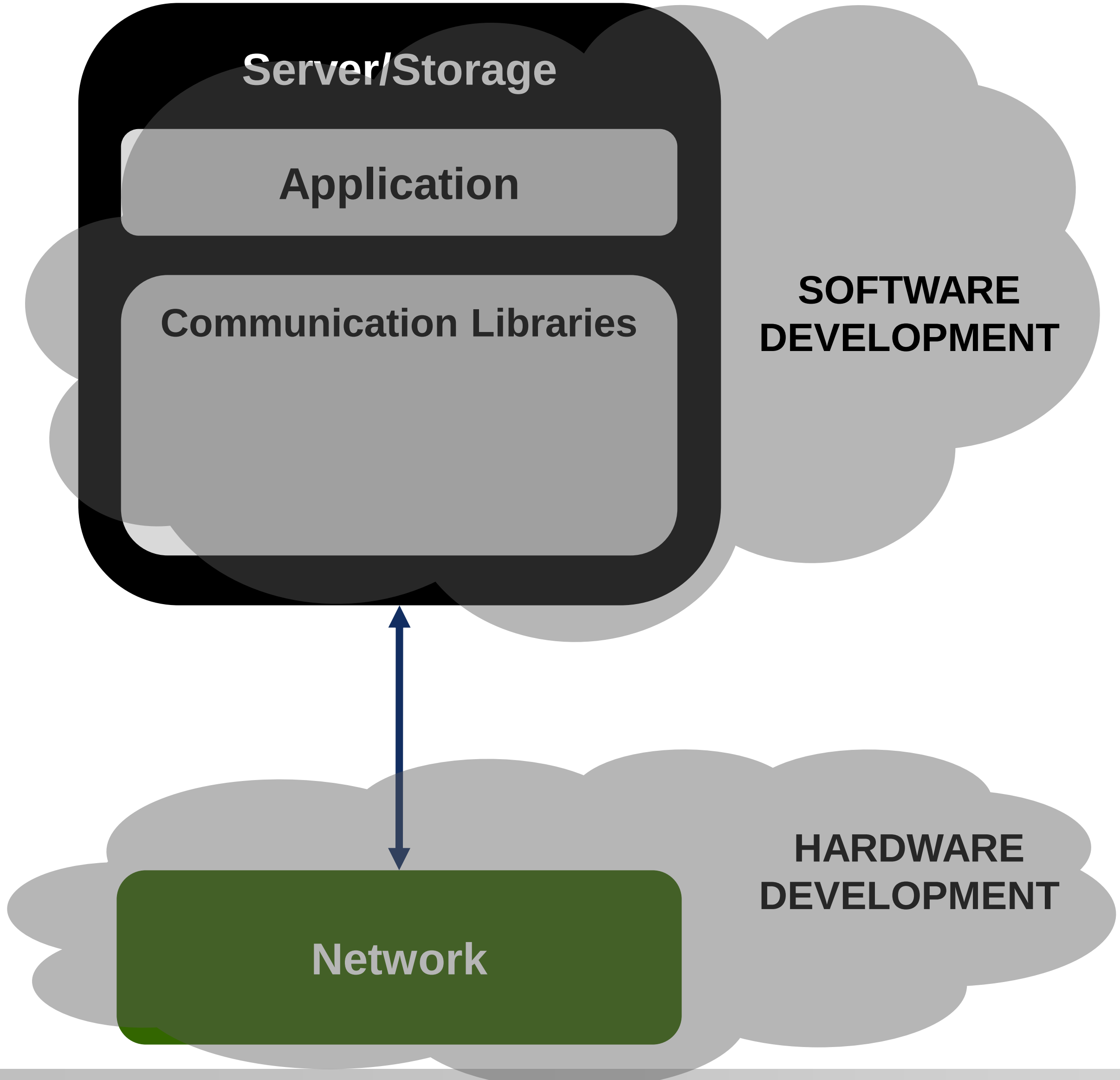
todd@mellanox.com

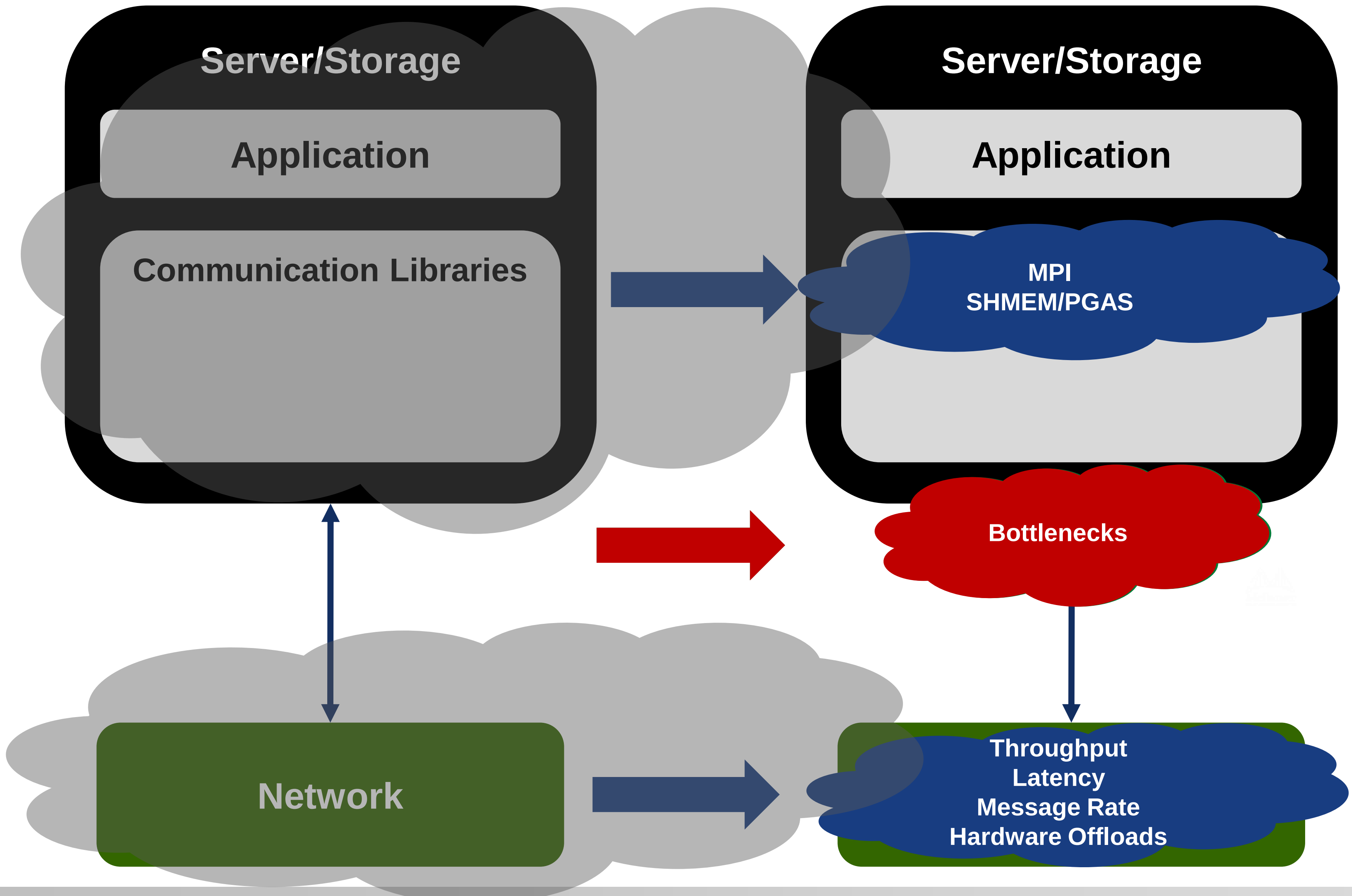
# Agenda

- The Co-Design Architecture for Parallel Programming Languages
- An Introduction to PGAS Languages
- FCA – Fabric Collective Accelerations
- Mellanox/HP Collaboration On InfiniBand Scalability for One-Sided Communication

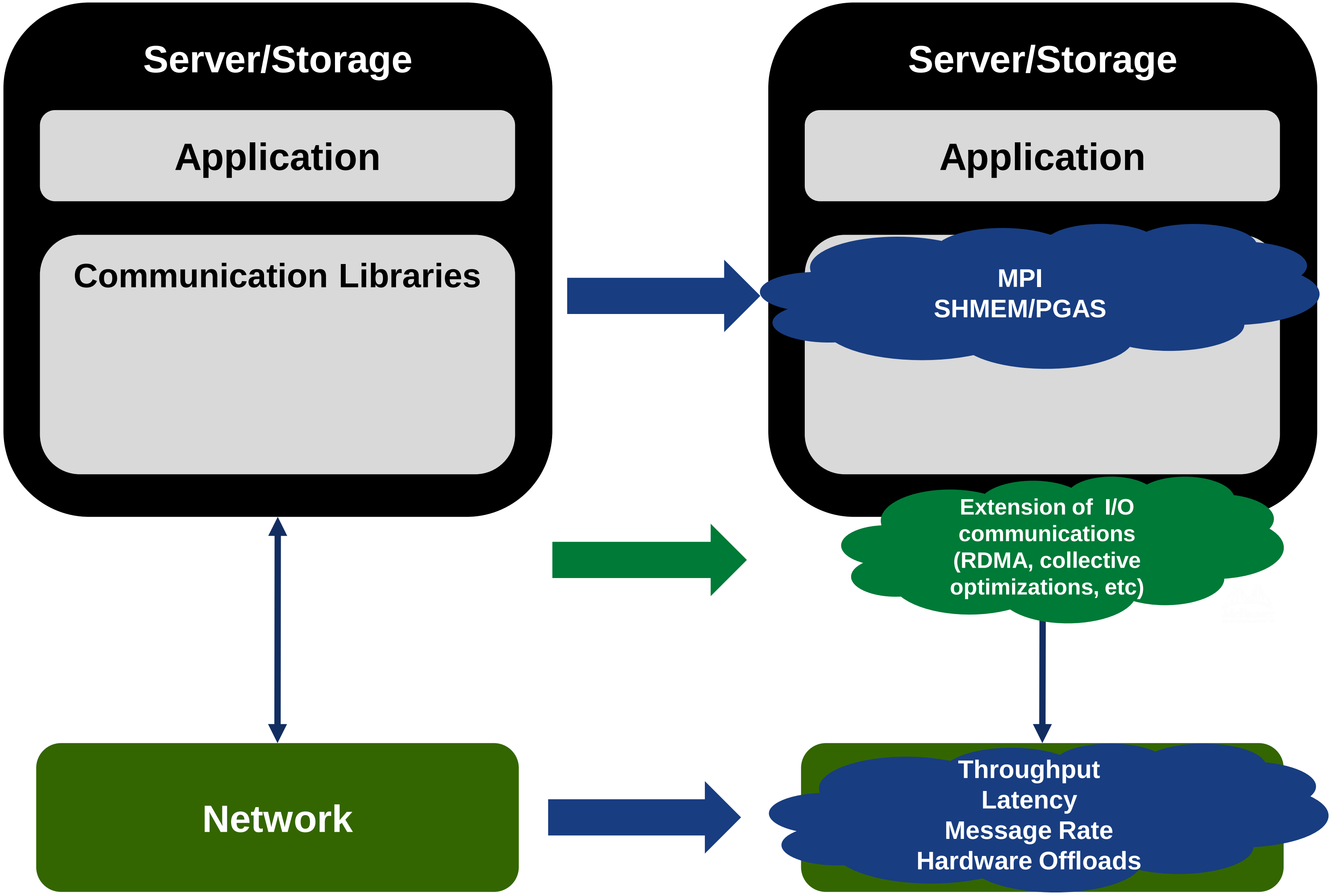
# The Co-Design Architecture



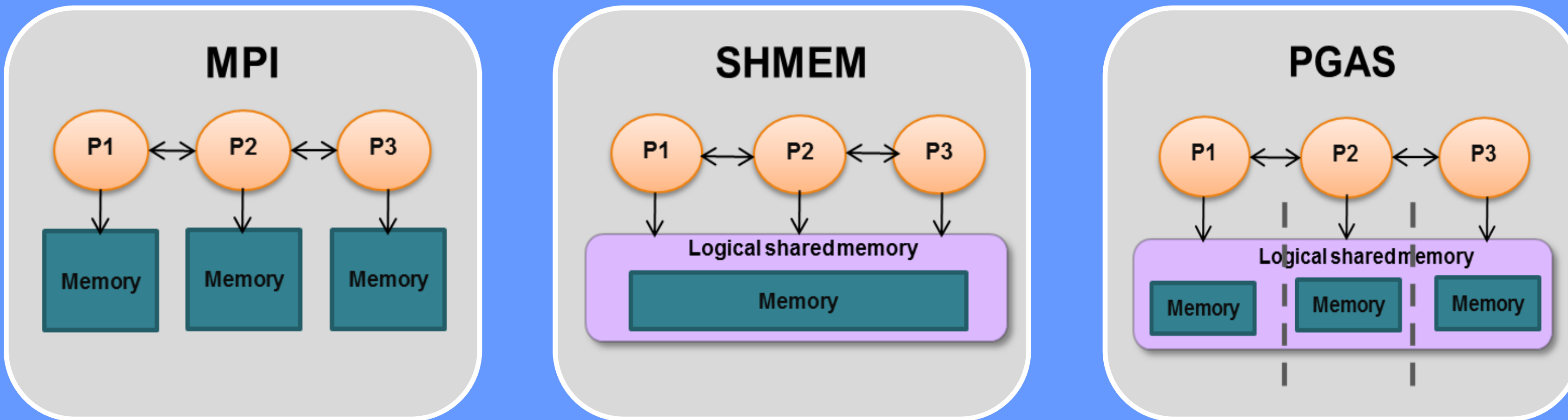
# The Co-Design Architecture



# The Co-Design Architecture



# Mellanox ScalableHPC Accelerate Parallel Applications




## MXM

- Reliable Messaging Optimized for Mellanox HCA
- Hybrid Transport Mechanism
- Efficient Memory Registration
- Receive Side Tag Matching

## FCA

- Topology Aware Collective Optimization
- Hardware Multicast
- Separate Virtual Fabric for Collectives
- CoreDIrect Hardware Offload

## InfiniBand Verbs API

# An Introduction to PGAS Languages

# Introduction to PGAS Languages

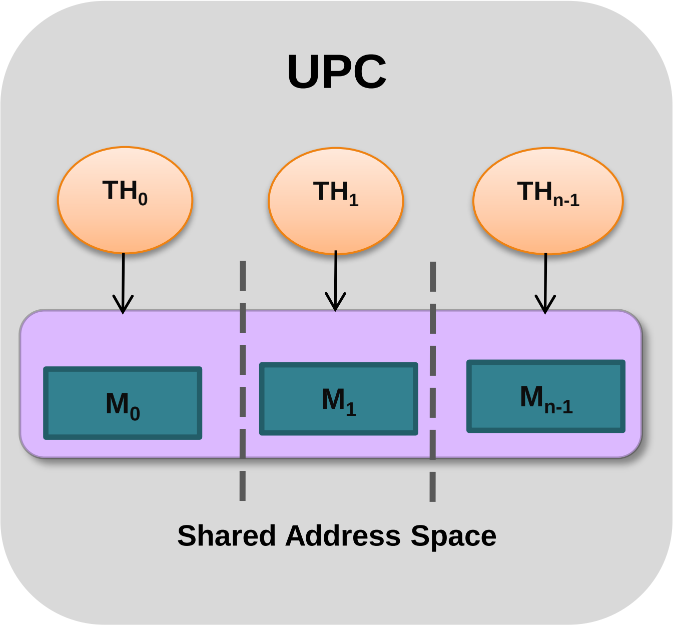
- PGAS – Partitioned Global Address Space – Best of both worlds
  - Message passing and shared memory methods
- Explicitly-parallel programming model with SPMD parallelism like MPI
  - Fixed at program start-up, typically 1 thread per processor
- Global address space model of memory
  - Allows programmer to directly represent distributed data structures
- Address space is logically partitioned
  - Local vs. remote memory (two-level hierarchy)
- SHMEM is being used/proposed as a lower level interface for PGAS implementations.
- Multiple PGAS languages: UPC (C), CAF (Fortran), Titanium (Java)

# PGAS Language Example - UPC

- UPC – Unified Parallel C
  - Open source compiler from LBNL/UCB
  - Currently operates over Infiniband Verbs RC connections via GASnet interface

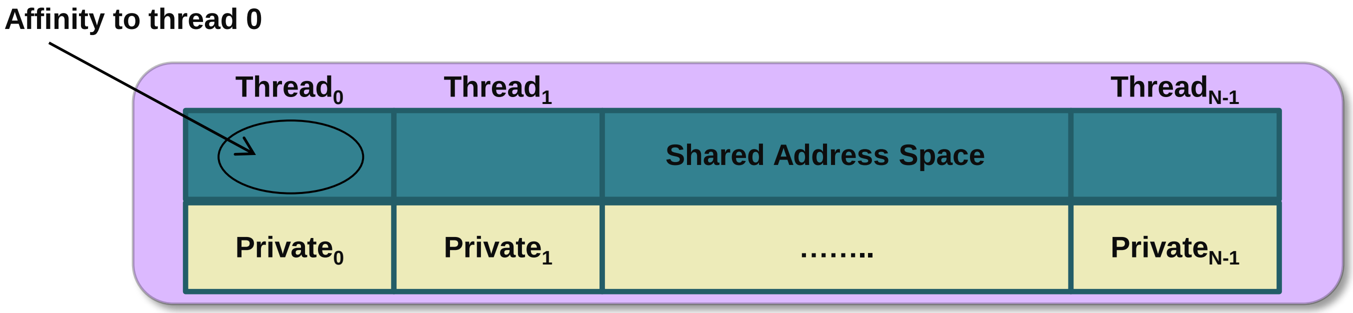
- Utilizes a distributed shared memory programming model
  - Similar to traditional shared memory model, but allows for data locality
  - Distributed shared memory is divided into partitions where each $M_i$ is associated with thread $TH_i$.

- Features include:
  - Simple statements for remote memory access
  - Minimization of thread communication overhead by exploiting data locality



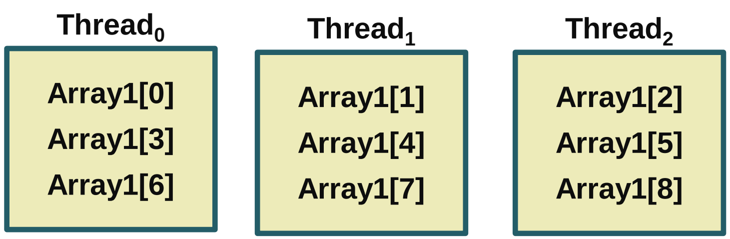
# UPC Memory Model

- UPC memory is divided into private and shared space
- Each thread has its own private space in addition to a portion of the shared space
- A UPC *shared* pointer can access any locations in the shared space.   A *private* pointer may reference only addresses in it's private space or local portion of shared space.

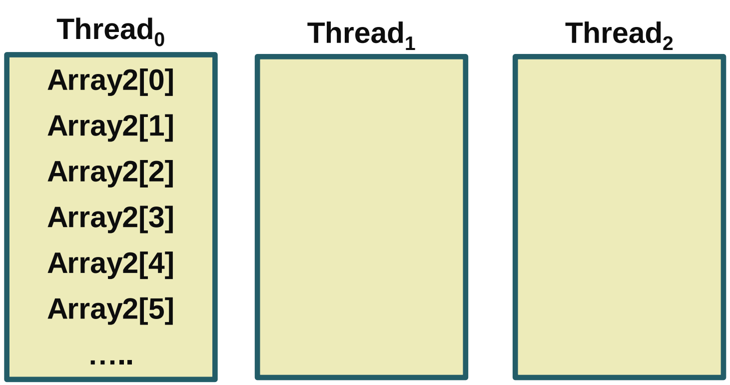


# UPC Memory Model – Shared Memory Usage

- Must use shared qualifier in variable declaration
  - *shared [block_size] type variable_name* : means that variable_name is distibuted across memory space in the span of block_size per thread.
- Examples:
  - shared [1] int array1[N]




  - shared [N] int array2[N]



# UPC Execution Model

- A number of threads working independently in a SPMD fashion
  - Number of threads specified at compile-time or run-time; available as program variable **`THREADS`**
  - **`MYTHREAD`** specifies thread index (`0..THREADS-1`)
  - **`upc_barrier`** is a global synchronization: all wait
  - **`upc_forall`** is similar to for-loop but also indicates which thread will run the loop iteration

- There are two compilation modes
  - Static Threads mode:
    - THREADS is specified at compile time by the user
    - The program may use THREADS as a compile-time constant
  - Dynamic threads mode:
    - Compiled code may be run with varying numbers of threads

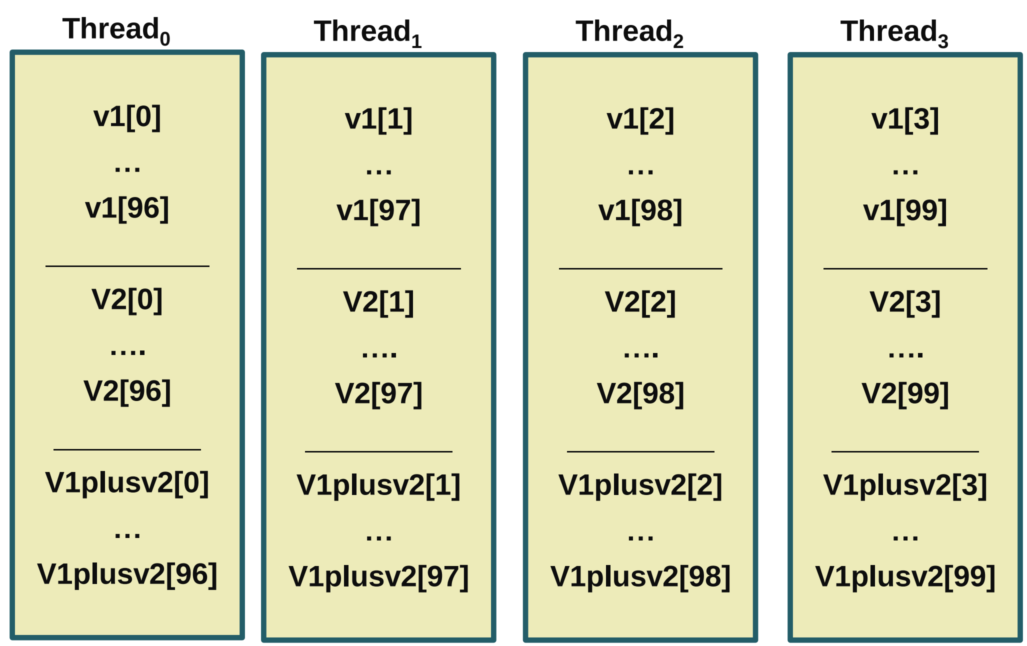
# My first UPC program

```
#include <upc_relaxed.h>
#include <stdio.h>

void main()
{
   if (MYTHREAD==0){
      printf("Rcv’d: ‘Starting Execution’ from THREAD %d\n",MYTHREAD );
   }

   printf("Hello World from THREAD %d (of %d THREADS)\n", MYTHREAD,
THREADS);
}
```

# UPC Programming – Array Copy Example

```
#include<upc_relaxed.h>
#define N 100
shared int v1[N], v2[N], v1plusv2[N];

void main()
{
   int i;
   for(i=0;i<N;i++)
      if(MYTHREAD==i%THREADS)
         v1plusv2[i]=v1[i]+v2[i] ;
}
```

Owner computes



# UPC Programming – Array Copy Example (optimization)

```
#include<upc_relaxed.h>
#define N 100
shared int v1[N], v2[N], v1plusv2[N];

void main()
{
   int i;
   upc_forall(i=0;i<N;i++,i)
      ~~if(MYTHREAD==i%THREADS)~~
         v1plusv2[i]=v1[i]+v2[i] ;
}
```

Owner computes

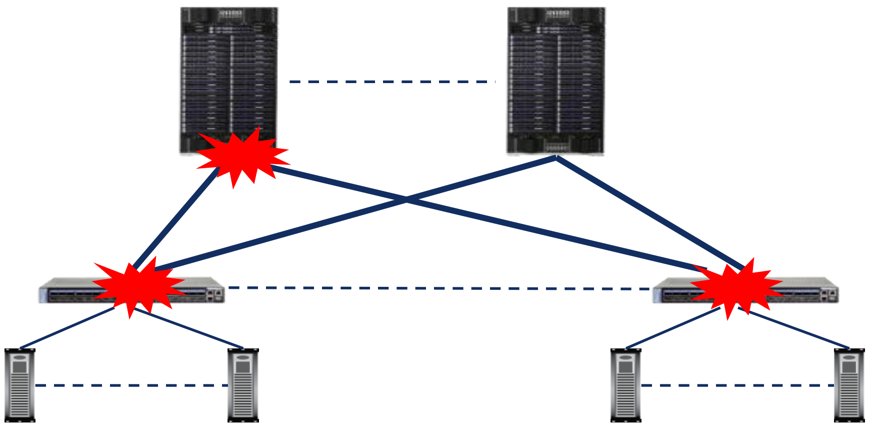
# Fabric Collective Accelerations

# What are Collective Operations?

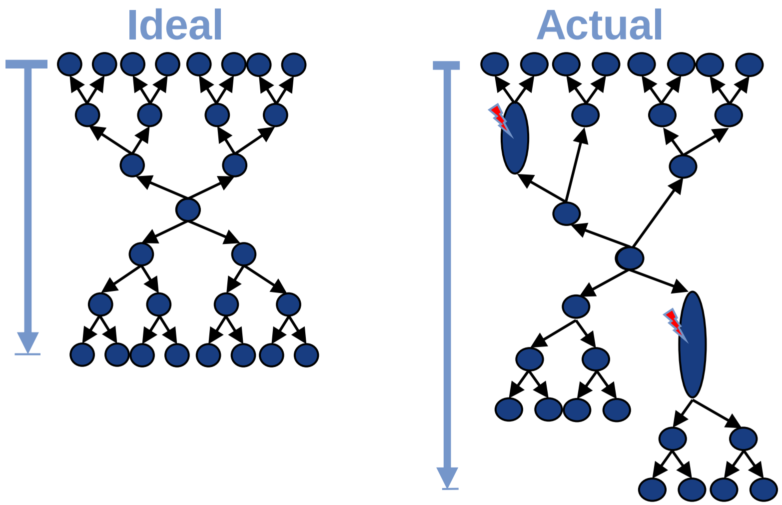
- Collective Operations are Group Communications involving all processes in job
- Synchronous operations
  - By nature consume many ‘Wait’ cycles on large clusters
- Popular examples
  - Barrier
  - Reduce
  - Allreduce
  - Gather
  - Allgather
  - Bcast

# Collective Operation Challenges at Large Scale

- Collective algorithms are not topology aware and can be inefficient

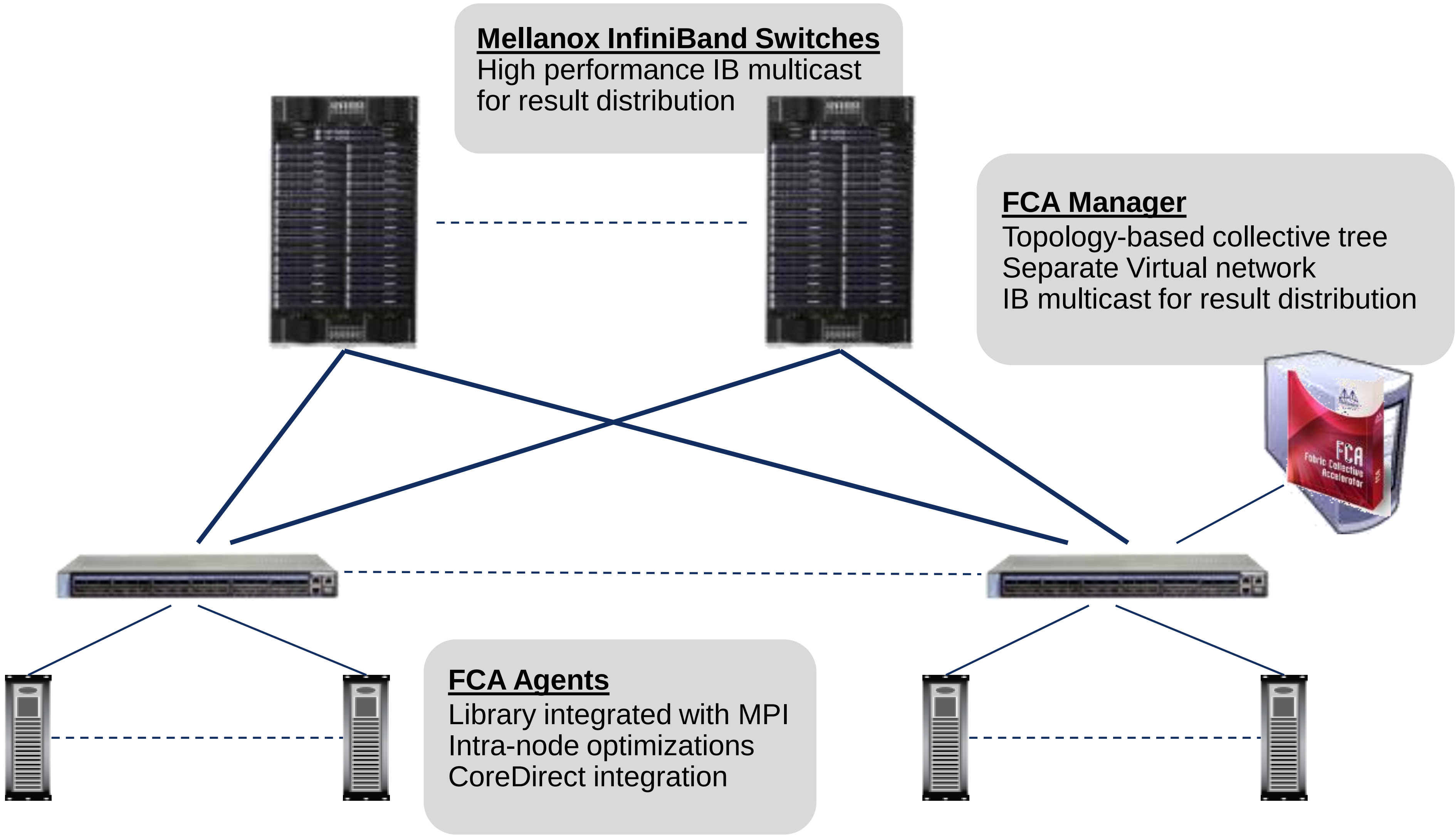


- Congestion due to many-to-many communications



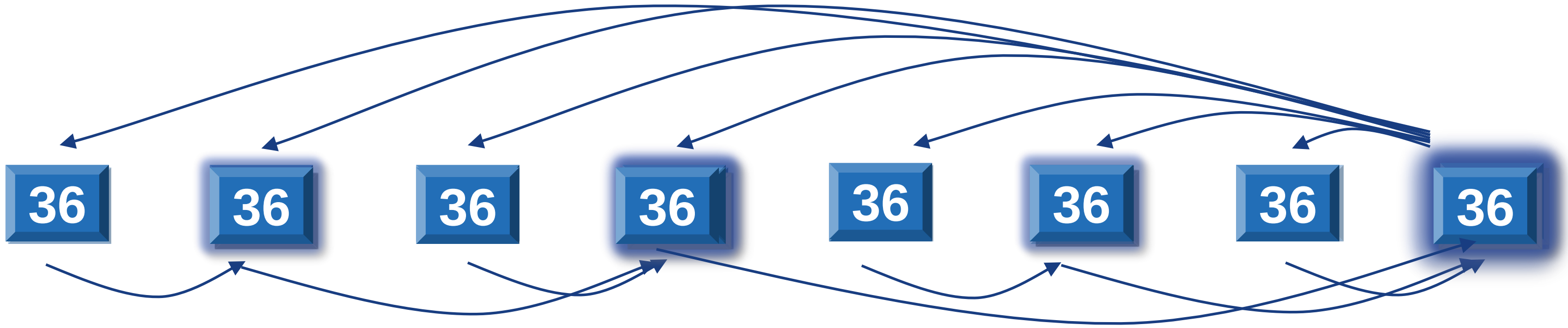

- Slow nodes and OS jitter affect scalability and increase variability

# Mellanox Fabric Collectives Accelerations (FCA)



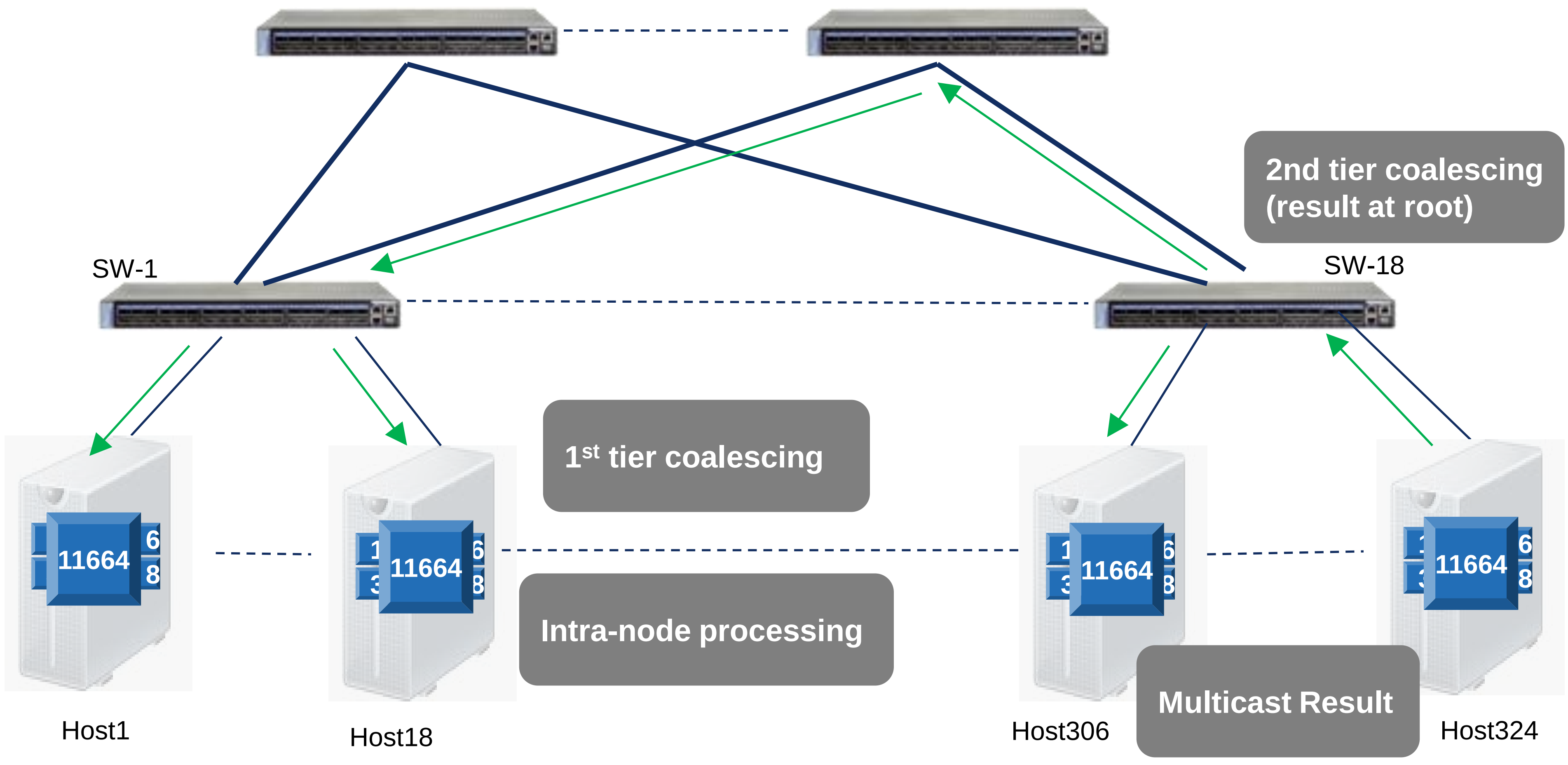
# Collective Example – Allreduce using Recursive Doubling

- Collective Operations are Group Communications involving all processes in job




- A 4000 process Allreduce using recursive doubling is 12 stages

# Scalable Collectives with FCA



# Thank You

HPC@mellanox.com

PAVING THE ROAD TO EXASCALE

ADVANCING NETWORK PERFORMANCE, EFFICIENCY, AND SCALABILITY.