# SC14 OpenSHMEM TUTORIAL

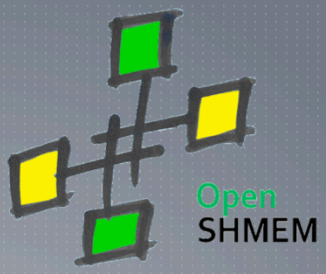
Presenters: Tony Curtis, Deepak Eachempati, Dounia Khaldi, Aaron Welch
University of Houston, Texas
Oscar Hernandez, Pavel Shamis, Manju Gorentla Venkata
Oak Ridge National Laboratory

1

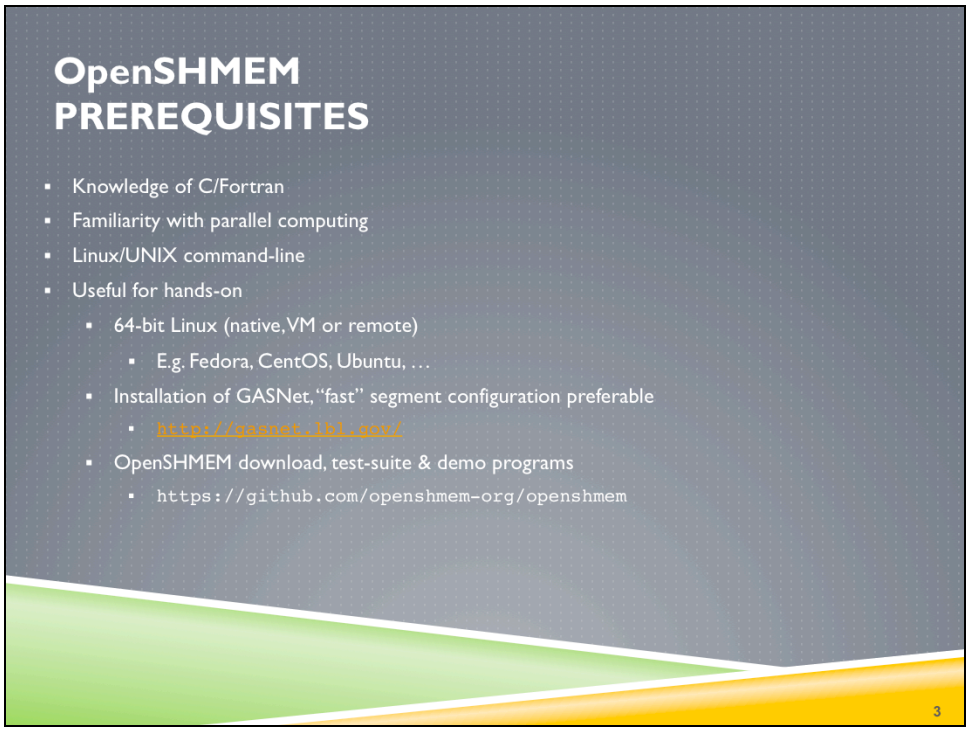
# OpenSHMEM WORKSHOP OUTLINE

http://www.openshmem.org/

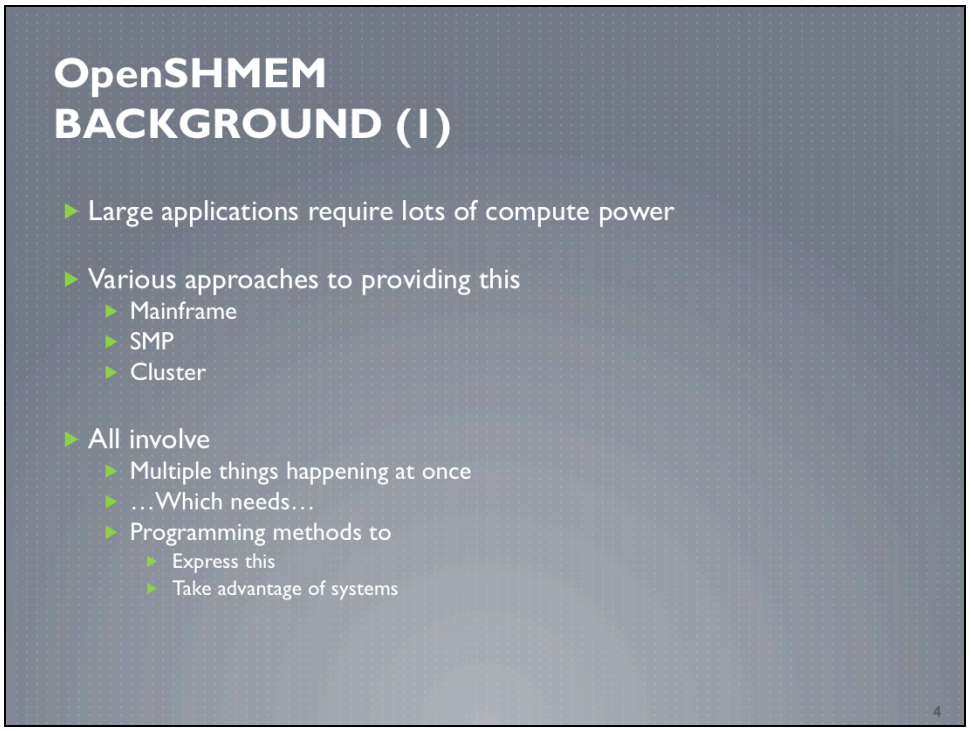
- Prerequisites
- Background
- Concepts
- History and Implementations
- The OpenSHMEM Project
- OpenSHMEM routines
- OpenSHMEM and Hardware
- Developing OpenSHMEM Applications
- OpenSHMEM Implementations
- OpenSHMEM: The Future…



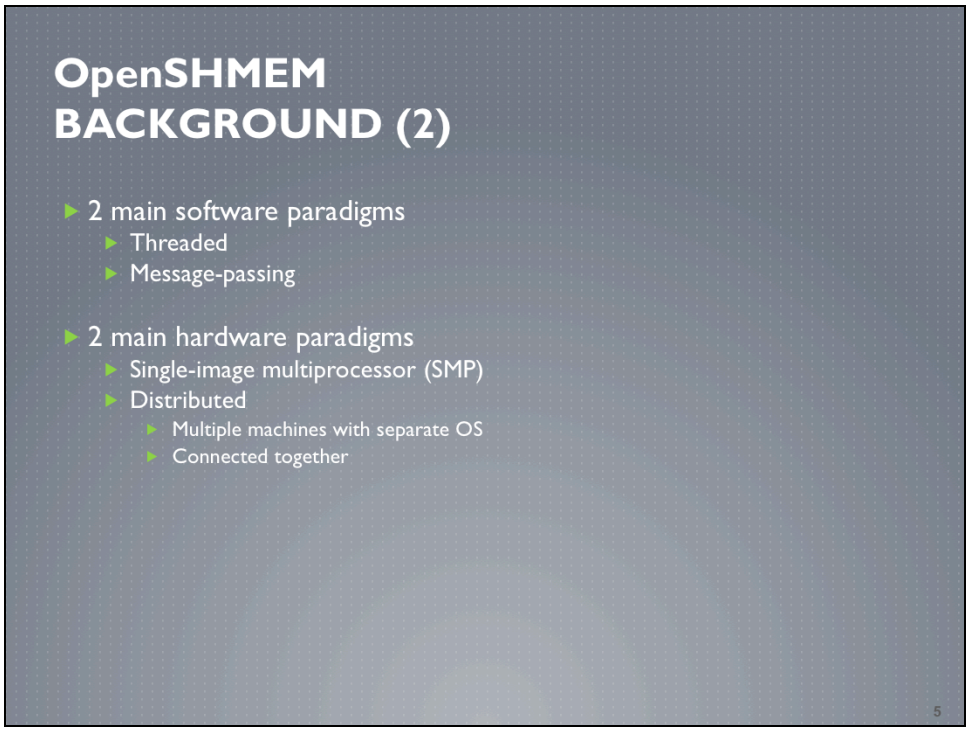

2




These are the platform and software requirements (some of them, anyway) for installing the reference implementation. Precise package requirements are different on different distributions.




Talk about the setting for parallel computing and programming. There are different approaches (libraries/languages) to expressing parallelism.




Talk about the different ways of getting parallel systems. The narrative is heading towards the 2nd point as being more interesting for OpenSHMEM.

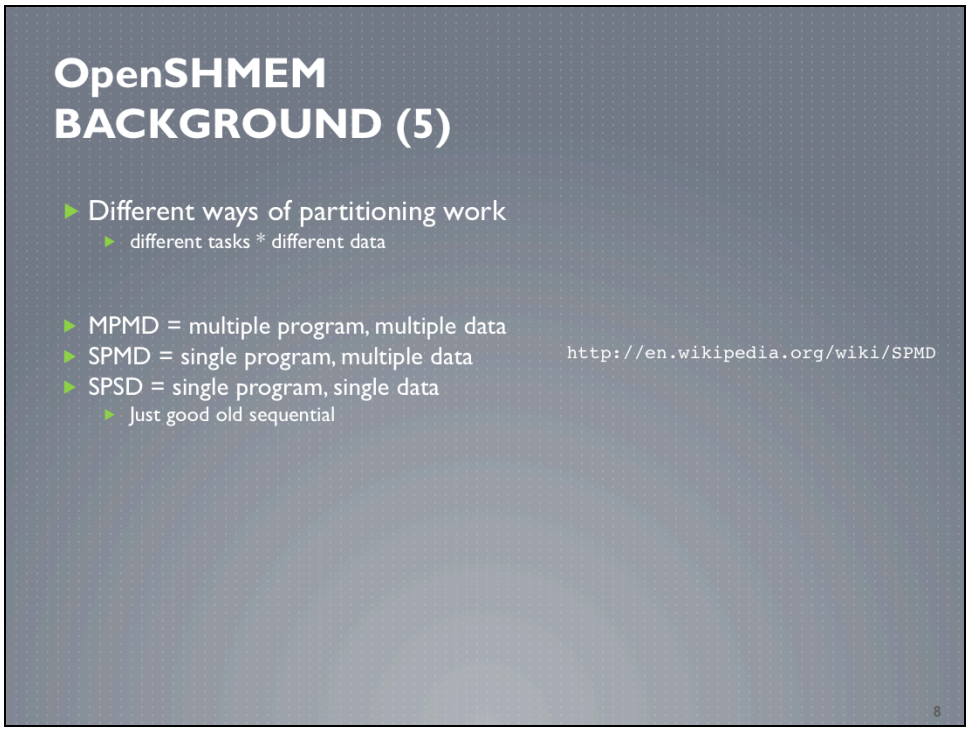
# OpenSHMEM BACKGROUND (3)

- Programming environments provide abstraction
- In particular
  - A language or library can be used on many machine types
  - Implementation hides differences & leverages features
- 2 dominant models
  - MPI
  - OpenMP
- First, a little background for context…

6

# OpenSHMEM BACKGROUND (4)

- Concurrent
  - Multiple things logically happen at once
  - May be emulated
    - E.g. time slicing on shared machine
- Parallel
  - = Concurrent +
  - Things really happen independently
  - On separate processors
- Work is partitioned in some way across resources

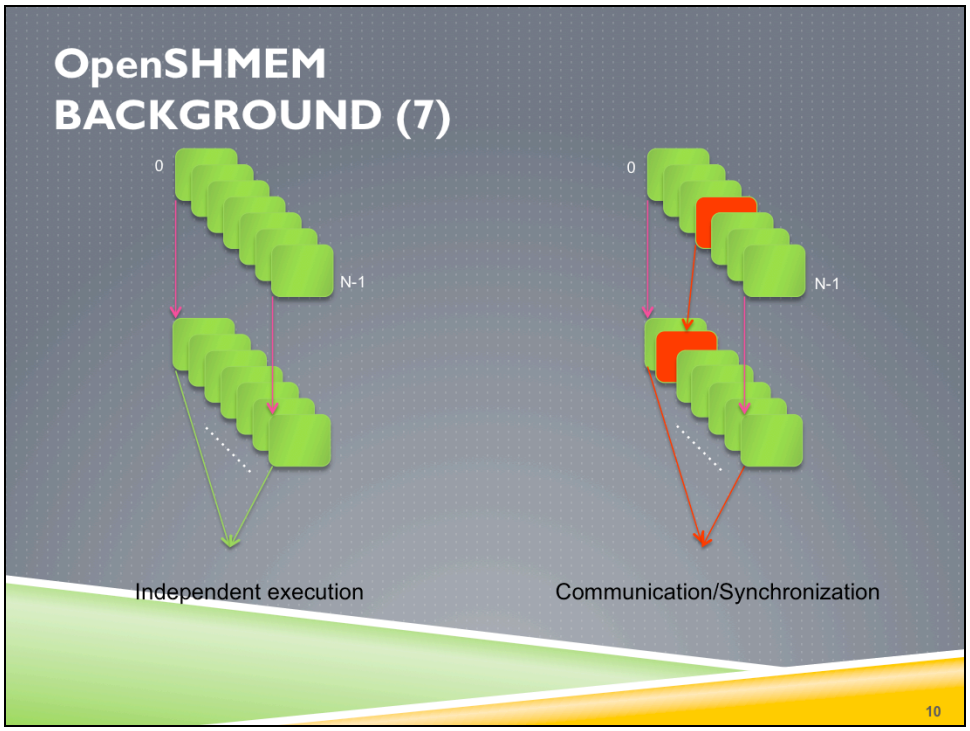



SPMD is the OpenSHMEM model. Diagram coming in a couple of slides.

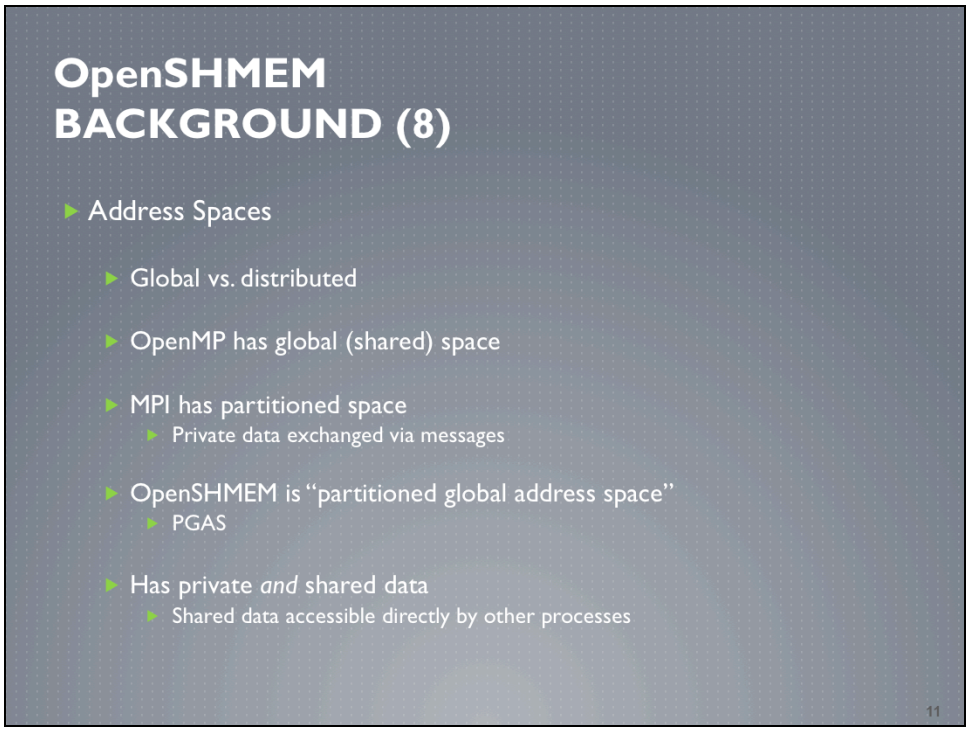
# OpenSHMEM BACKGROUND (6)

- SPMD
  - Program launches many processes
  - Each starts with same code (SP)
  - And then typically operates on some specific part of the data (MD)
  - Processes may then communicate with each other
    - Share common data
    - Broadcast work
    - Collect results
- PVM and MPI are well-known examples
- OpenSHMEM is SPMD

9




The execution model has multiple instances of the same program running independently of each other, *until* they need to communicate and later synchronize to make sure variables have been fully updated. Synchronization introduces ordering points. It also can be a huge overhead…

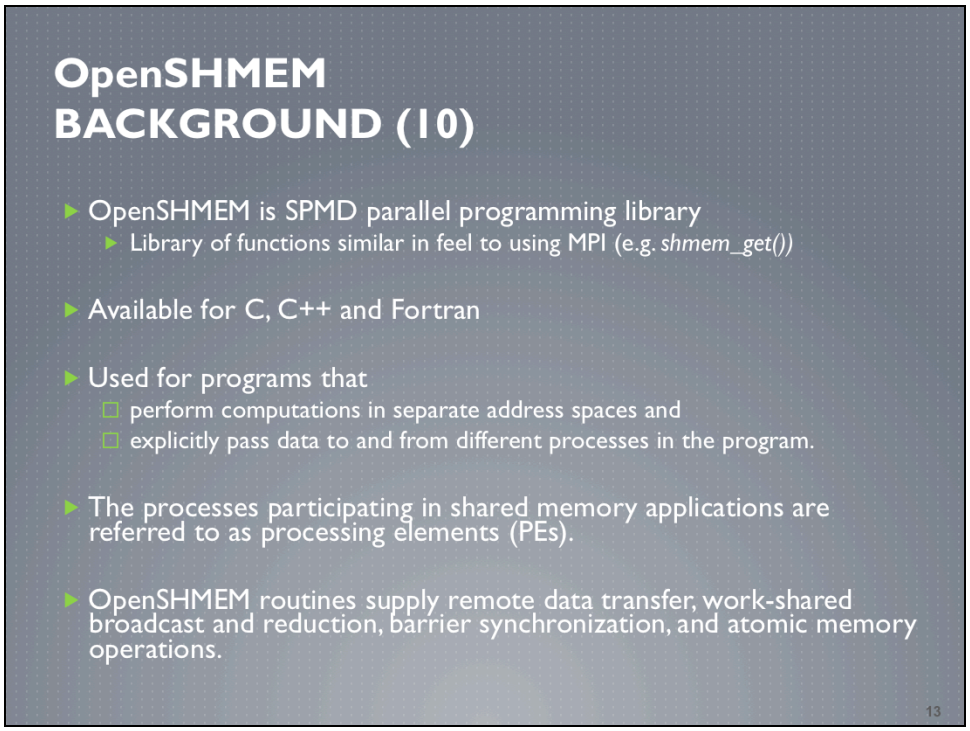



Where data lives in OpenSHMEM programs and how it is viewed across a program's memory space. Lead into the ecosystem of PGAS programming models.

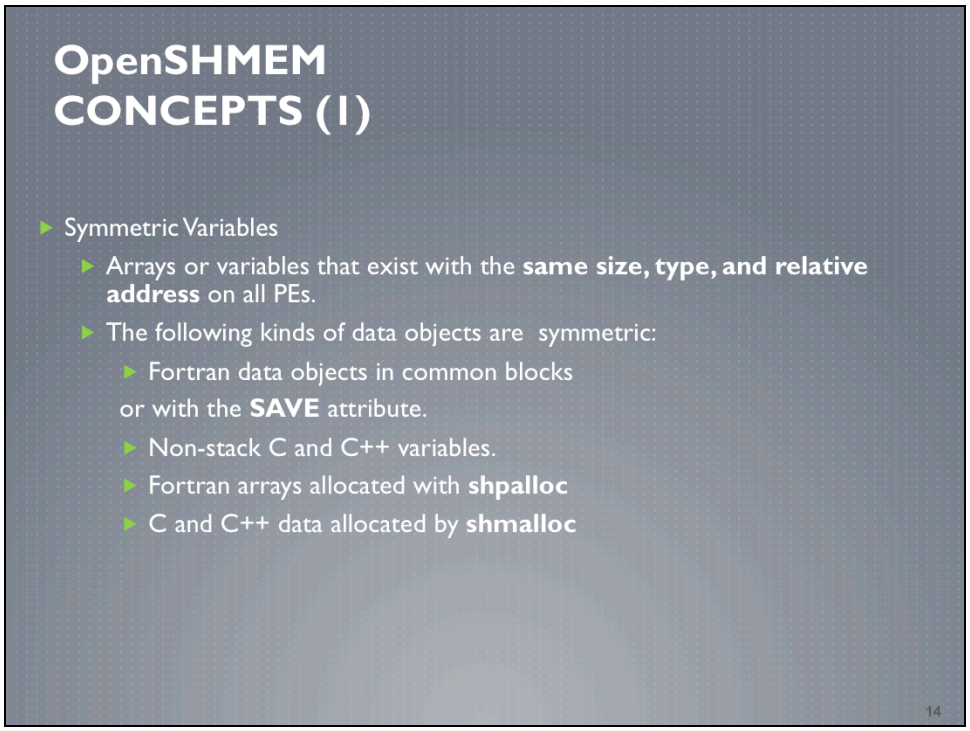
# OpenSHMEM BACKGROUND (9)

- The PGAS family
  - Libraries include…
    - GASNet
    - ARMCI / Global Arrays
    - CCI
    - GASPI/GPI
  - Languages include…
    - Chapel
    - Titanium
    - X10
    - UPC
    - CAF
    - (Often built on these libraries)

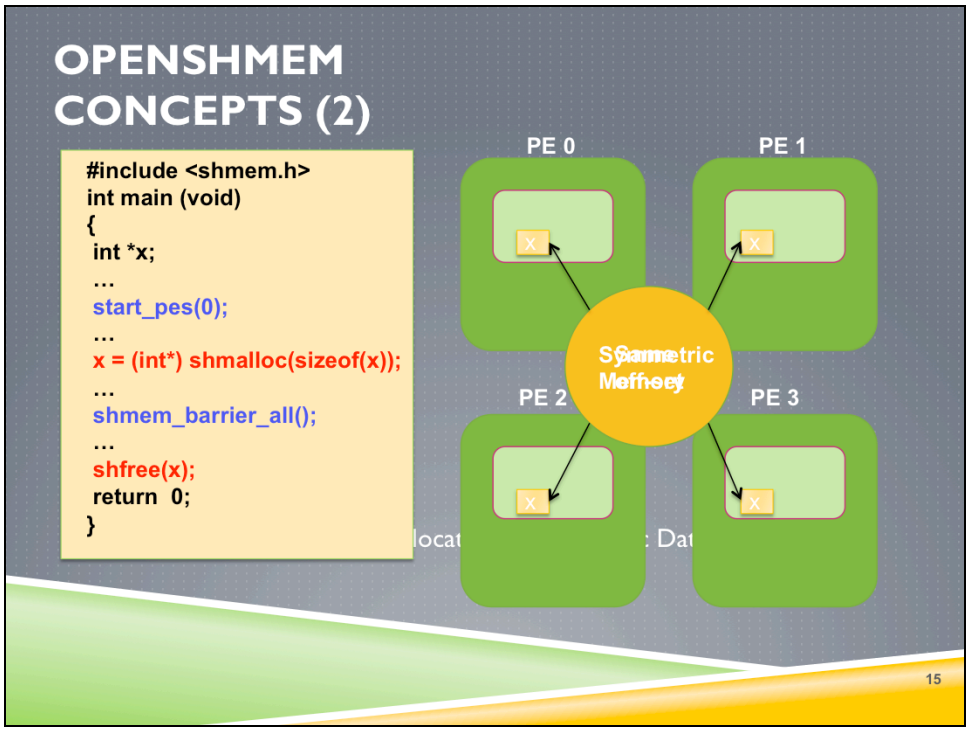
12



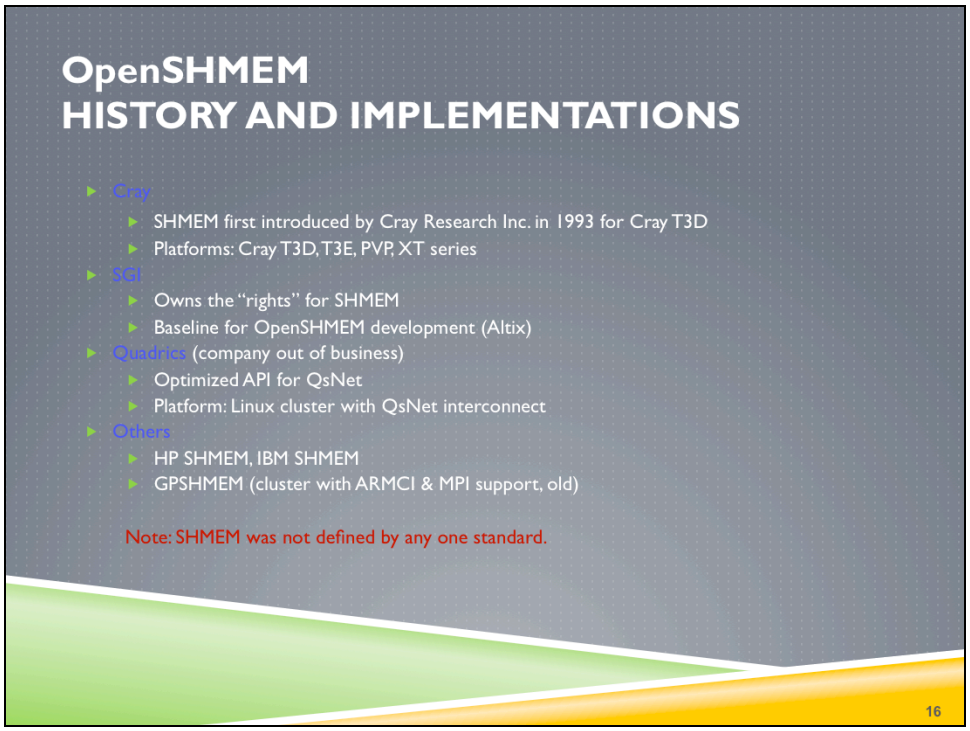

By “similar to MPI” we don’t mean OpenSHMEM is necessarily going to be a drop-in replacement, because the 1-sided nature of the calls and the way in which memory space is presented is different. But it’s the same idea of taking an existing language and adding library calls to get (distributed) parallelism.



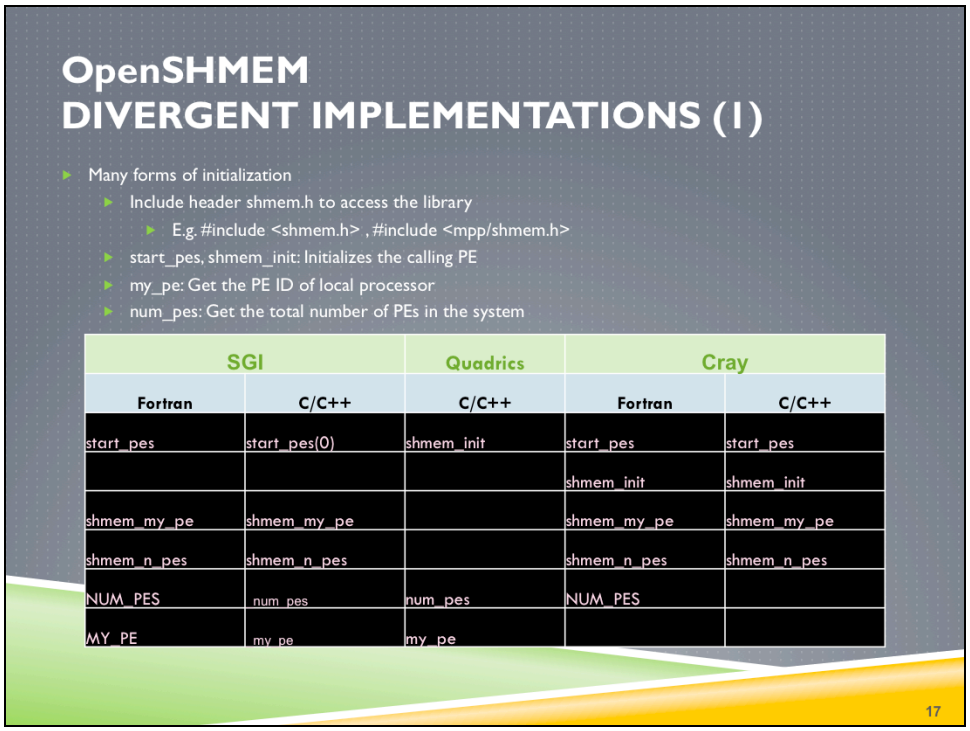

**The local existence of a corresponding symmetric object implies that a data object is remotely accessible.**



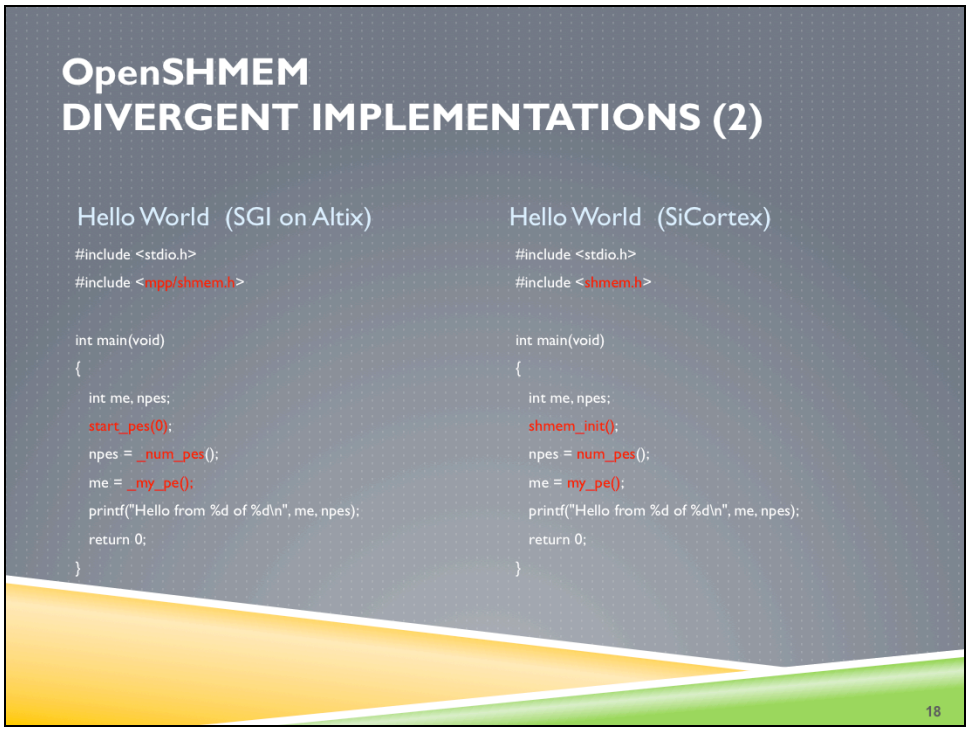

Here's a snippet from a simple program that does execution-time dynamic allocation of memory. Full slide has an animation showing the progression of the program and memory being allocated. Need to talk about what symmetric means.



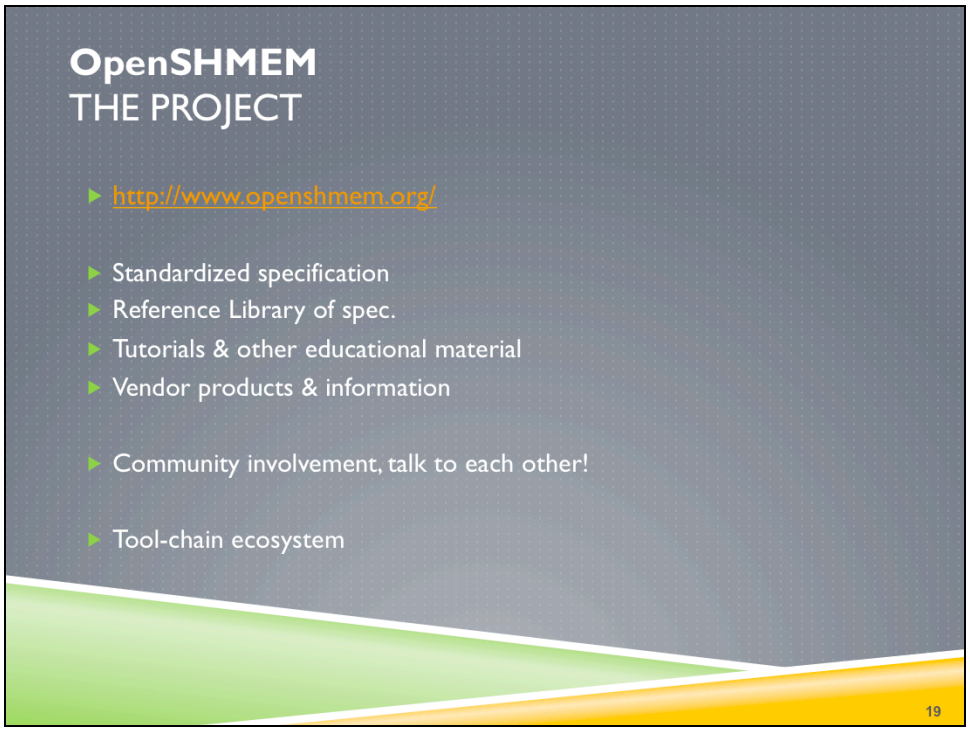

Read up on the Cray T3D.



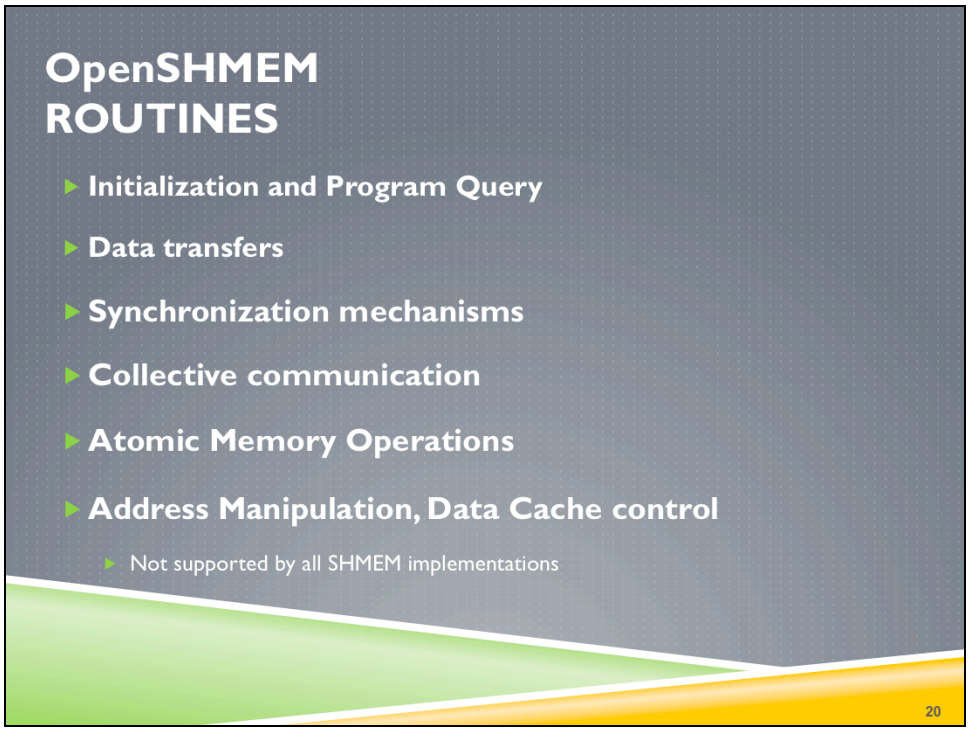

This is what happens when there's no standard. You can see the various different calls that showed up in different APIs from different vendors. Different routine names, and different parameter sets. In particular, the initialization call is completely different across different vendor SHMEMs.




The good old “hello world” program shows the different routines and header locations on different platforms. The programs will not compile with the other vendors’ SHMEMs.




So enough people were interested in forging a standard for SHMEM, which ended up with the obvious name of OpenSHMEM. The website has quite a lot of things on it, not just software, but publications, vendor information and of course tutorials like this one.




These are the routines in the API. Need to talk about how these fir together and are actually used. Hands-on should give people a better feeling for usage. Slides tend to be just a laundry list that don't show context and there's not enough space on the screen to show interesting code.

# OpenSHMEM INITIALIZATION & QUERY

- void start_pes(int n)
  - Initialize the OpenSHMEM program
  - "n" means "number of PEs" but now ignored, set to 0
  - Number of PEs taken from invoking environment
    - E.g. from MPI or job scheduler
  - PEs numbered 0 .. (N – 1) in flat space
- int _num_pes(void)
- int shmem_n_pes(void)
  - **return number of PEs in this program**
- int _my_pe(void)
- int shmem_my_pe(void)
  - **return "rank" of calling PE**

# OpenSHMEM DATA TRANSFER (1)

- Put
  - Single variable
    - void shmem_TYPE_p(TYPE *target, TYPE value, int pe)
      - TYPE = double, float, int, long, short
  - Contiguous object
    - void shmem_TYPE_put(TYPE *target, const TYPE *source, size_t nelems, int pe)
      - For C: TYPE = double, float, int, long, longdouble, longlong, short
      - For Fortran: TYPE=complex, integer, real, character, logical
    - void shmem_putSS(void *target, const void *source, size_t nelems, int pe)
      - Storage Size (SS, bits) = 32, 64, 128, mem (any size)
  - Target must be symmetric

# OpenSHMEM DATA TRANSFER (2)

- Example: Cyclic communication via puts

```
{
    /*Initializations*/
    int src;
    int *dest;
    …..
    start_pes(0);
    ….
    src  = me;
    dest = (int *) shmalloc (sizeof (*dest));
    nextpe = (me + 1) % npes;        /*wrap around */

    shmem_int_put (dest, &src, 1, nextpe);
    more_work_goes_here (…
    shmem_barrier_all();
    x = dest * 0.995 + 45 * y;
    …
}
```

Automatic data element

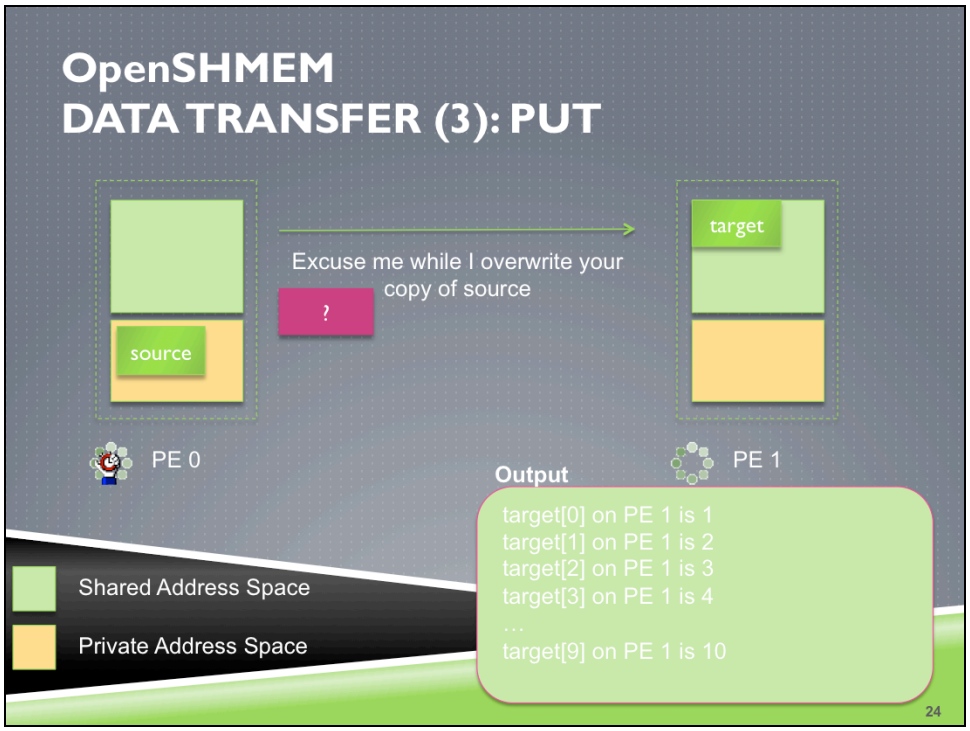
Symmetric data element

Synchronization before use

**Points To Remember**

- 'Destination' has to be symmetric
- Consecutive puts are not guaranteed to finish in order
- Put returns after the data has been copied out of the source
- Completion guaranteed only after synchronization

23


OpenSHMEM
DATA TRANSFER (3): PUT
target
Excuse me while I overwrite your copy of source
?
source
PE 0
PE 1
Output
target[0] on PE 1 is 1
target[1] on PE 1 is 2
target[2] on PE 1 is 3
target[3] on PE 1 is 4
…
target[9] on PE 1 is 10
Shared Address Space
Private Address Space
24

**shmem_put():**

# OpenSHMEM DATA TRANSFER (4)

- Get
  - Single variable
    - **TYPE shmem_TYPE_g(TYPE *target, TYPE value, int pe)**
      - For C: TYPE = double, float, int, long, longdouble, longlong, short
      - For Fortran: TYPE=complex, integer, real, character, logical
  - Contiguous object
    - **void shmem_TYPE_get(TYPE *target, const TYPE *source, size_t nelems, int pe)**
      - For C: TYPE = double, float, int, long, longdouble, longlong, short
      - For Fortran: TYPE=complex, integer, real, character, logical
    - **void shmem_getSS(void *target, const void *source, size_t nelems, int pe)**
      - Storage Size (SS, bits) = 32, 64, 128, mem (any size)
  - Source must be symmetric

# OpenSHMEM DATA TRANSFER (5)

- Example: Summation at PE 0

```
{
    /*Initializations*/
    int *src;
    int target, sum;
    …..
    start_pes(0);
    ….
    src  = (int *) shmalloc (sizeof (*src));
    src = me;
    sum=me;
    if(me == 0){
        for(int i = 1,i<num_pes();i++){
          shmem_int_get(&target, src, 1
          sum = sum + target;
        }
    }
    …
}
```

Automatic data element

Symmetric data element

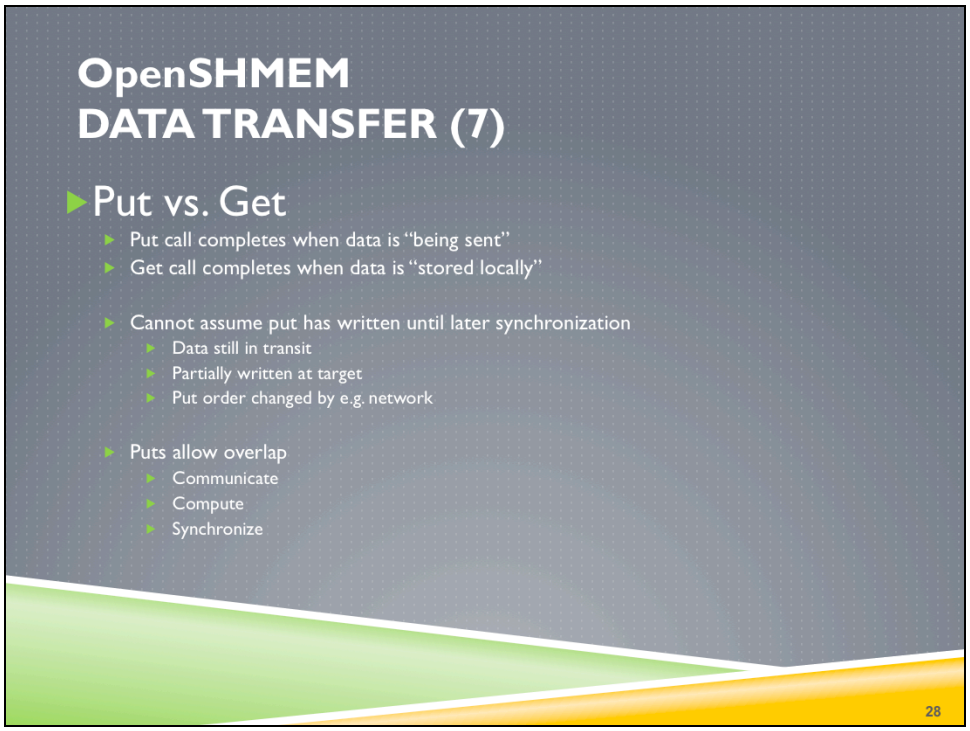
No synchronization before use

**Points To Remember**

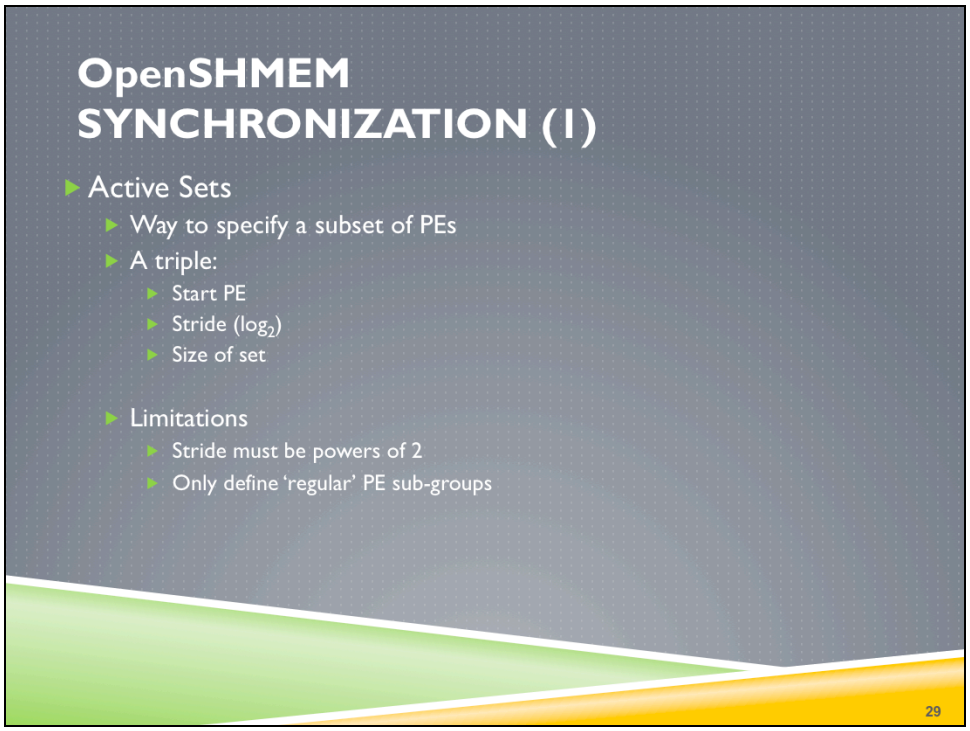
- 'Source' has to be remotely accessible
- Consecutive gets finish in order
- The routines return after the data has been delivered to the 'target' on the local PE

# OpenSHMEM DATA TRANSFER (6)

- Strided put/get
  - void shmem_TYPE_iput(TYPE *target, const TYPE *source, ptrdiff_t tst, ptrdiff_t sst, size_t nelems, int pe)
    - For C: TYPE = double, float, int, long, longdouble, longlong, short
    - For Fortran: TYPE=complex, integer, real, character, logical
    - tst and sst indicate stride between accesses of target and source resp.
  - And the sized variants as for put/get




This slide is to stress that puts and gets behave differently. It is clearly a better idea to have puts in your program with deferred synchronization so you can generate more overlap of comms/comp or comms/comms. It's about getting more work done in different places at the same time. Puts are an offload function.




Every element of this array must be initialized with the value _SHMEM_SYNC_VALUE before any of the PEs in the active set enter the routine. OpenSHMEM *currently* only has these regular log2 active sets but there's a number of proposals out there for generalizing them. There are interesting headaches associated with separate memory allocations belonging to active sets, where we lose the idea of all allocations being globally symmetric.

# OpenSHMEM SYNCHRONIZATION (2)

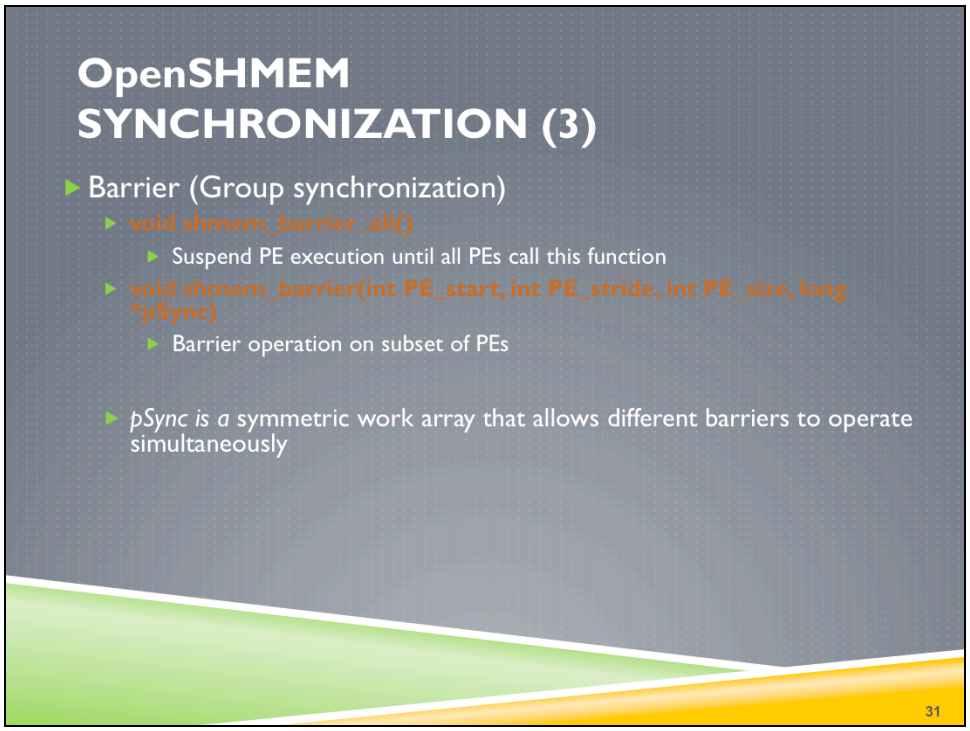
- Quick look at Active Sets
  - Example 1
    - PE_start = 0, logPE_stride = 0, PE_size = 4

    **ACTIVE SET?** PE 0, PE 1, PE 2, PE 3
  - Example 2
    - PE_start = 0, logPE_stride = 1, PE_size = 4

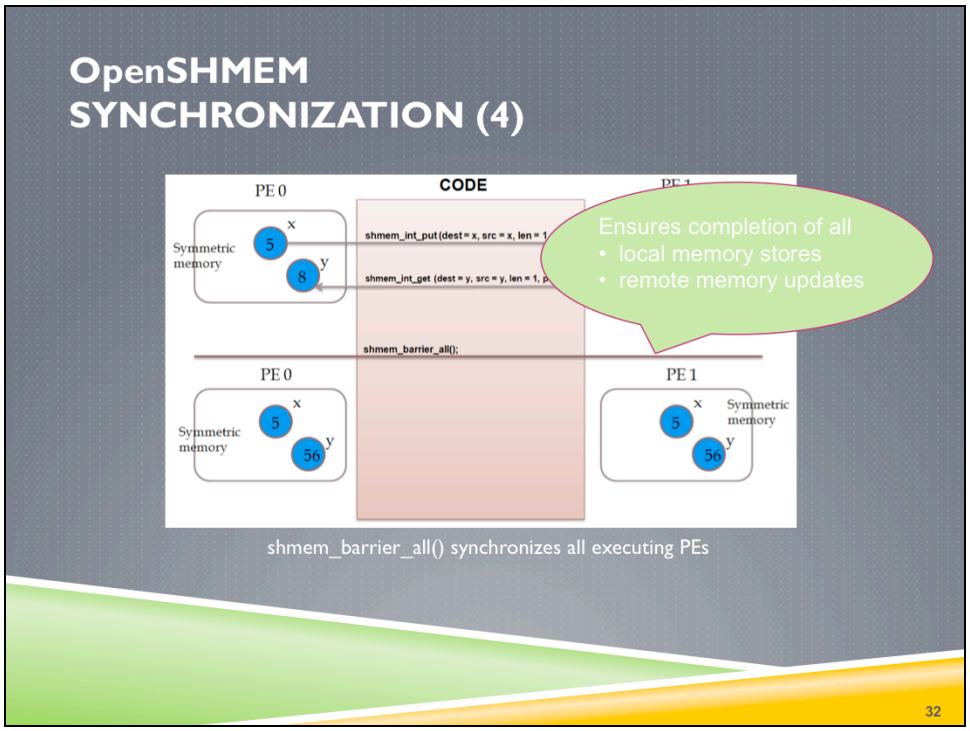
    **ACTIVE SET?** PE 0, PE 2, PE 4, PE 6
  - Example 3
    - PE_start = 2, logPE_stride = 2, PE_size = 3

    **ACTIVE SET?** PE 2, PE 6, PE 10

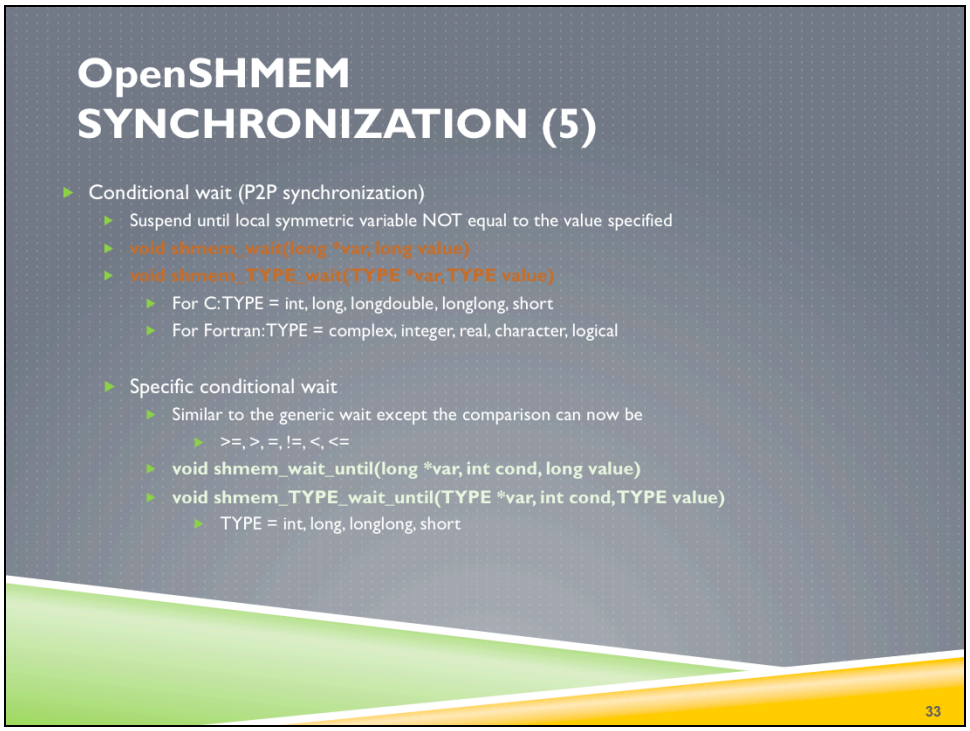
30



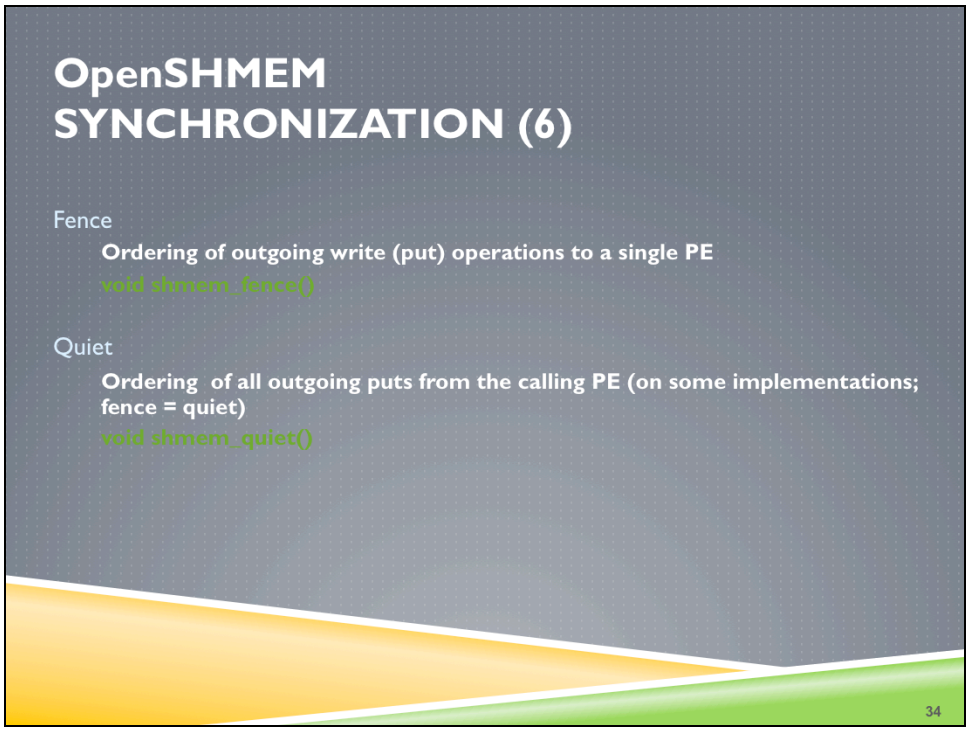

People want the explicit pSync parameter to go away in future versions.




A barrier_all is of course global  After memory allocation operations it is required but clearly could be a big slowdown on very large program runs.  Tie this observation in with the idea of memory spaces in active sets.




Waits are used as a synchronization tool that is a LOT lighter than a big barrier.




Fence can be implemented as quiet as it is a strict subset of quiet. But the opposite implementation is not true. Quiet affects all outbound puts. You can think of it as turning a valve on a compute node's network card and waiting for things to flush out before allowing more traffic…

# OpenSHMEM SYNCHRONIZATION (7)

## Example Fence

```
int main (int argc, char **argv)
{
....
....
  shmem_int_put (dest1, src1, 1, nextpe);
  shmem_fence();
  shmem_int_put (dest1, src2, 1, nextpe);
  ....
  shmem_barrier_all ();
  shfree (dest);

  return 0;
}
```

## Example Quiet

```
int main (int argc, char **argv)
{
....
....
  shmem_int_put (dest1, src1, 1, nextpe);
  shmem_int_put (dest2, src2, 1, nextpe+1);
  shmem_quiet();
  ....
  shmem_barrier_all ();
  shfree (dest);

  return 0;
}
```

35

## OpenSHMEM
## COLLECTIVE COMMUNICATION (1)

- Broadcast
  - One-to-all symmetric communication
  - No update on root
  - **void shmem_broadcastSS(void *target, void *source, size_t nelems, int PE_root, int PE_start, int PE_stride, int PE_size, long *pSync)**
    Storage Size (SS, bits) = 32, 64

36

# OpenSHMEM
# COLLECTIVE COMMUNICATION (2)

37

```
…
…
int *target, *source;
target= (int *) shmalloc( sizeof(int) );
source= (int *) shmalloc( sizeof(int) );
*target= 0;
*source= 101;
if (me == 1) {
    *source = 222;
  }
shmem_barrier_all();
shmem_broadcast32(target, source, 1 0, 0, 0, 4, pSync);

 printf("target on PE %d is %d\n", _my_pe(), *target);
…
```

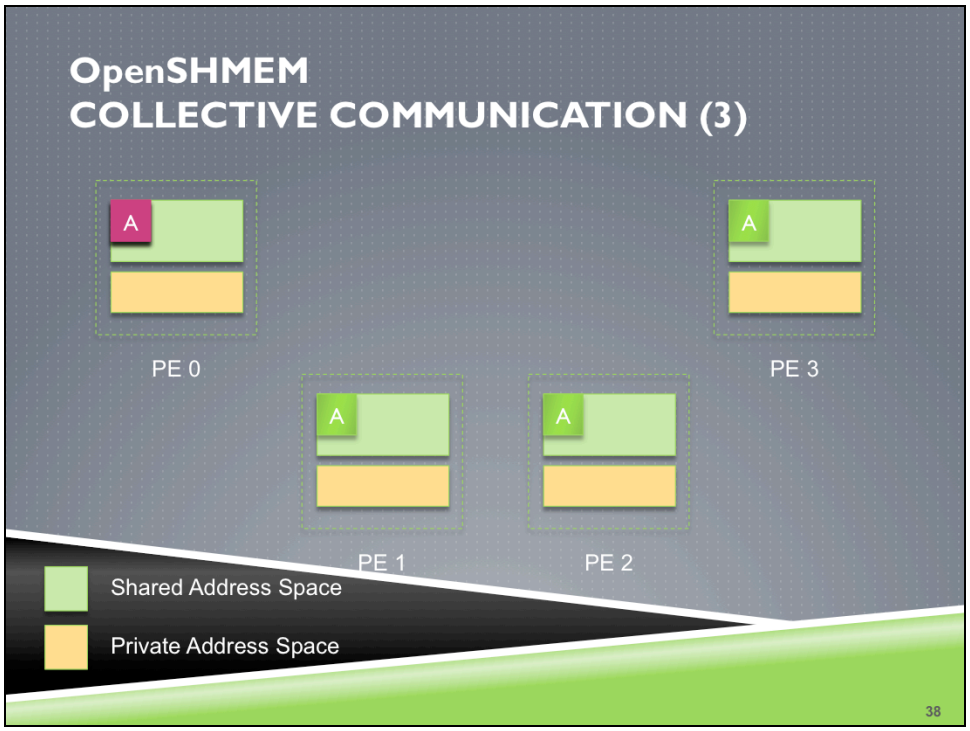
collective operation

**Output**

```
target on PE 0 is 0
target on PE 1 is 222
target on PE 2 is 222
target on PE 3 is 222
```

of the PE
of
active set

**Code snippet showing working of shmem_broadcast**

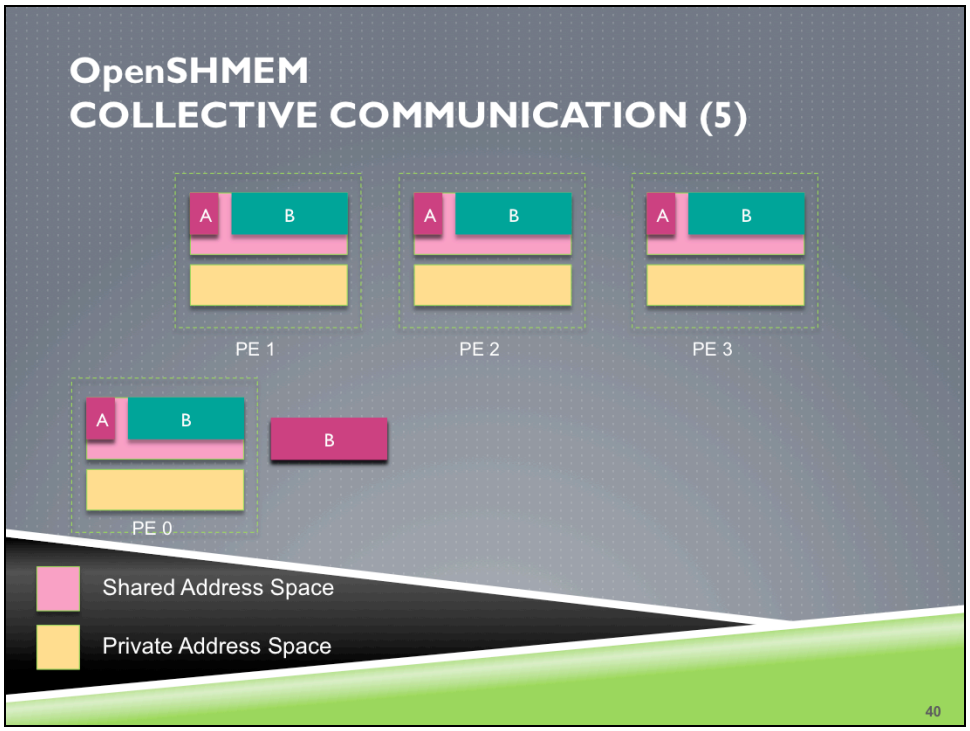

OpenSHMEM
COLLECTIVE COMMUNICATION (3)
A
A
PE 0
PE 3
A
A
PE 1
PE 2
Shared Address Space
Private Address Space
38

# OpenSHMEM
# COLLECTIVE COMMUNICATION (4)

Storage Size (SS, bits) = 32, 64

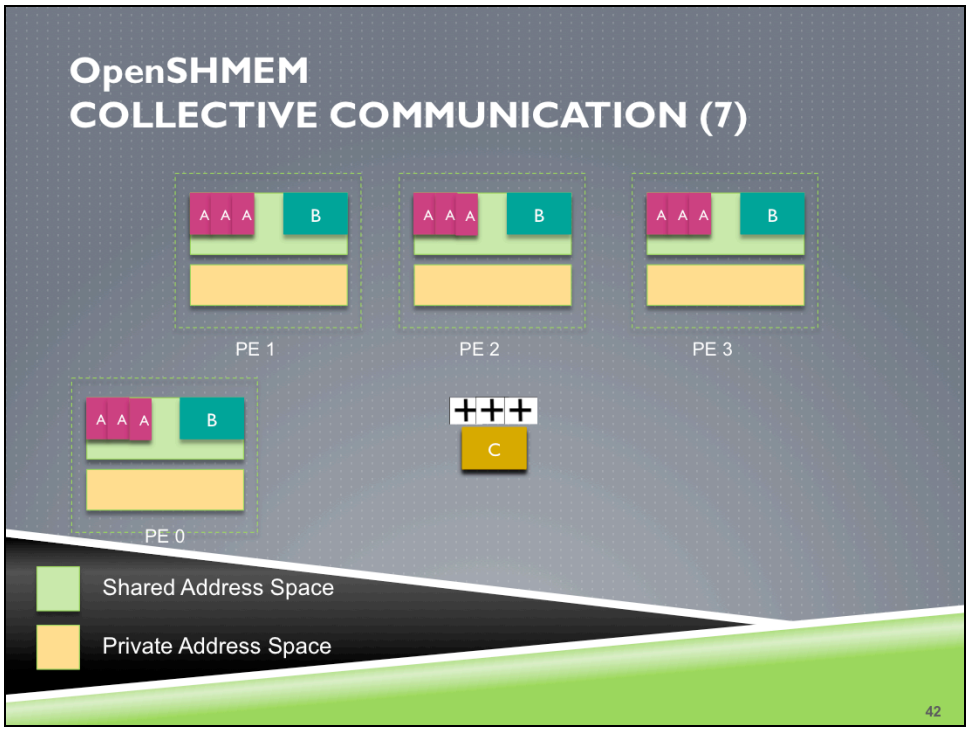
- Collection
  - Concatenates blocks of symmetric data from multiple PEs to an array in every PE
  - Each PE can contribute different amounts
  - **void shmem_collectSS(void *target, void *source, size_t nelems, int PE_start, int PE_stride, int PE_size, long *pSync)**
  - Concatenation written on all participating PEs
  - **shmem_fcollect variant**
    - When all PEs contribute exactly same amount of data
    - PEs know exactly where to write data, so no offset lookup overhead

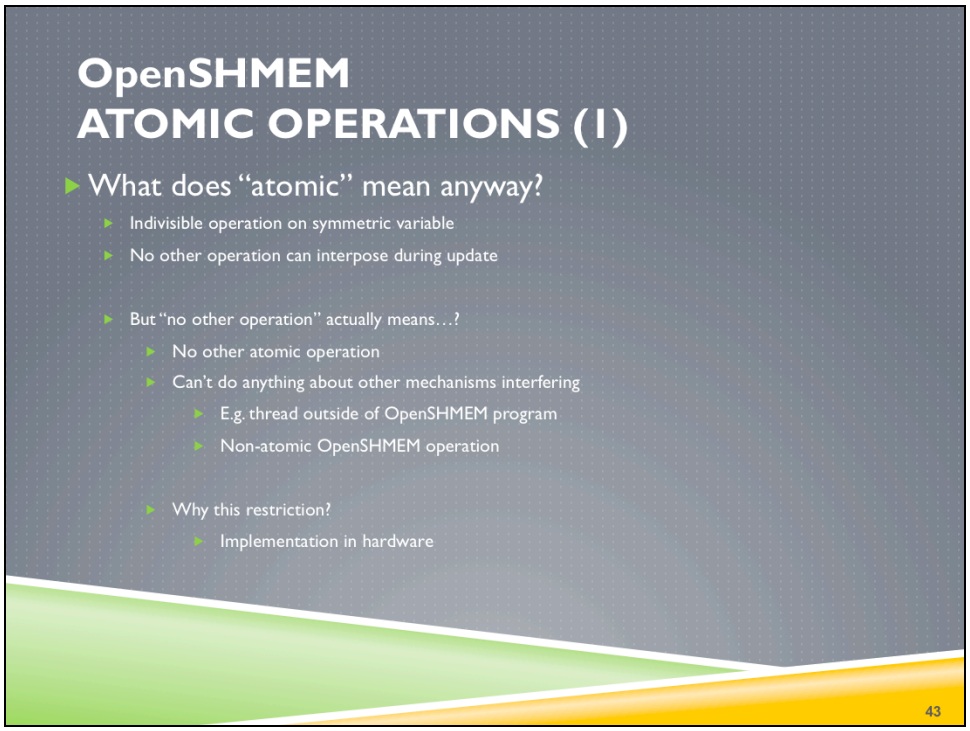
39


OpenSHMEM
COLLECTIVE COMMUNICATION (5)
A
B
PE 1
A
B
PE 2
A
B
PE 3
A
B
PE 0
B
Shared Address Space
Private Address Space
40

# OpenSHMEM
# COLLECTIVE COMMUNICATION (6)

- Reductions
  - Perform commutative operation across symmetric data set
    - **void shmem_TYPE_OP_to_all(TYPE *target, TYPE *source, int nreduce, int PE_start, int PE_stride, int PE_size, TYPE *pWrk, long *pSync)**
      - Logical OP = and, or, xor
      - Extrema OP = max, min
      - Arithmetic OP = prod(uct), sum
      - TYPE = int, long, longlong, longdouble, short, complex
  - Reduction performed and stored on all participating PEs
  - pWrk and pSync allow interleaving
  - E.g. compute arithmetic mean across set of PEs
    - sum_to_all / PE_size


OpenSHMEM
COLLECTIVE COMMUNICATION (7)
A A A
B
PE 1
PE 2
PE 3
PE 0
C
Shared Address Space
Private Address Space
42




Offloading operations to hardware generates more overlap and allows the CPU(s) to do other useful things, rather than service network traffic. Vendors put atomics, collectives and other ops. onto network hardware. Arithmetic/logic ops can also be put onto memory controllers for example = do the ops where the data is, don't incur cost of moving data back and forth (network, memory, …)

# OpenSHMEM ATOMIC OPERATIONS (2)

- Atomic Swap
  - Unconditional
    - **long shmem_swap(long *target, long value, int pe)**
    - **TYPE shmem_TYPE_swap(TYPE *target, TYPE value, int pe)**
      - TYPE = double, float, int, long, longlong
      - Return old value from symmetric target
  - Conditional
    - **TYPE shmem_TYPE_cswap(TYPE *target, TYPE cond, TYPE value, int pe)**
      - TYPE = int, long, longlong
    - Only if "cond" matches value on target

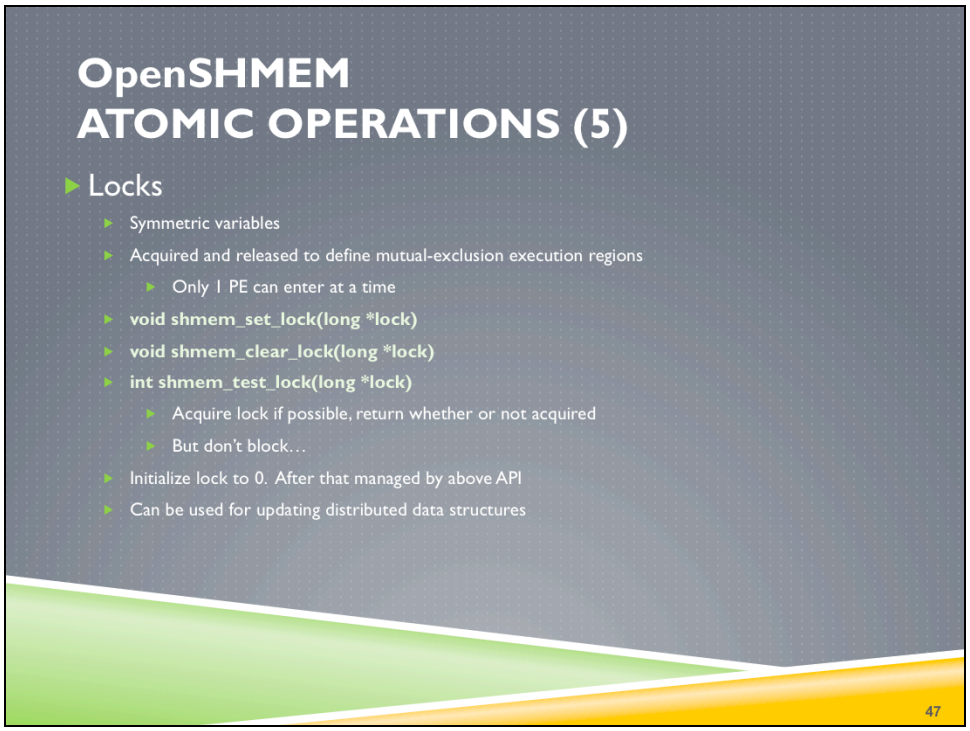
# OpenSHMEM ATOMIC OPERATIONS (3)

- Arithmetic
  - increment (= add 1) & add value
  - void shmem_TYPE_inc(TYPE *target, int pe)
  - void shmem_TYPE_add(TYPE *target,TYPE value, int pe)
    - TYPE = int, long, longlong
  - Fetch-and-increment & fetch-and-add value
  - TYPE shmem_TYPE_finc(TYPE *target, int pe)
  - TYPE shmem_TYPE_fadd(TYPE *target,TYPE value, int pe)
    - TYPE = int, long, longlong
  - Return previous value at target on PE

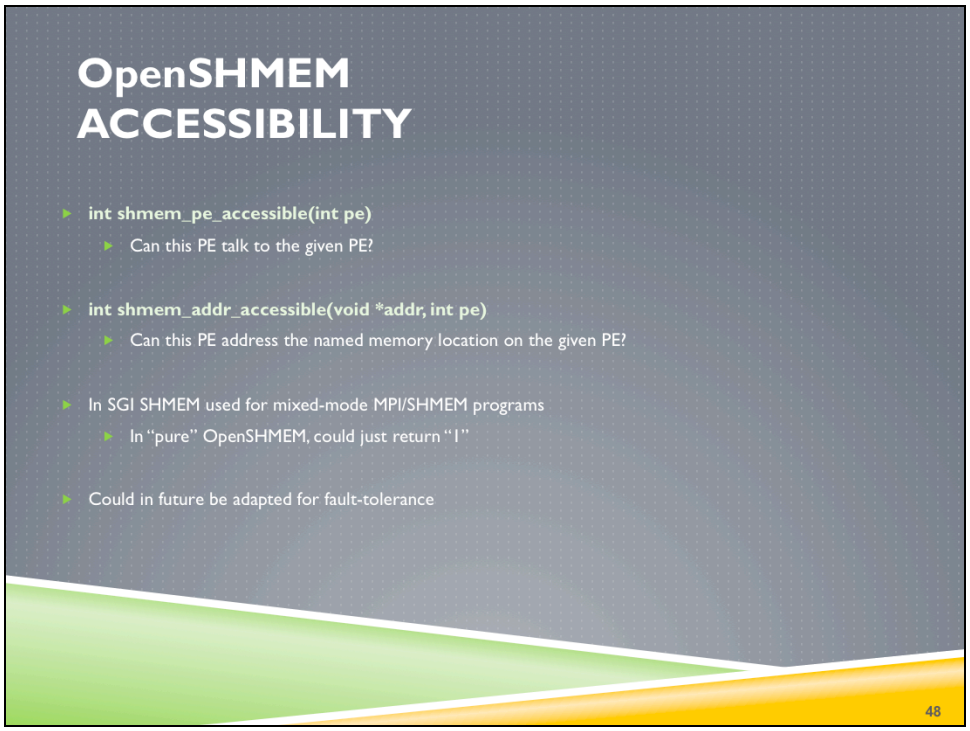
# OpenSHMEM
# ATOMIC OPERATIONS (4)

```
…
…
long *dest;
dest = (long *) shmalloc( sizeof(*dest) );
 *dest= me;
 shmem_barrier_all();
….
new_val = me;
 if (me== 1) {
    swapped_val = shmem_long_swap(target, new_val, 0);
    printf("%d: target = %d, swapped = %d\n", me, *target, swapped_val);
   }
shmem_barrier_all();
…
```

46



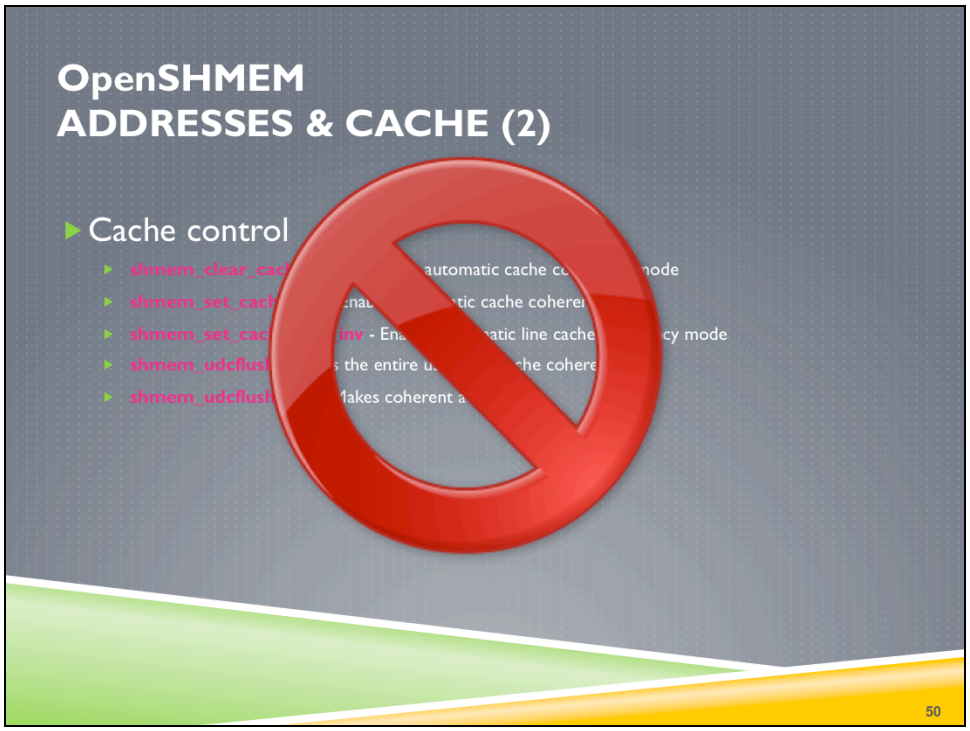

Lock use should be minimized as they can turn into performance killers.



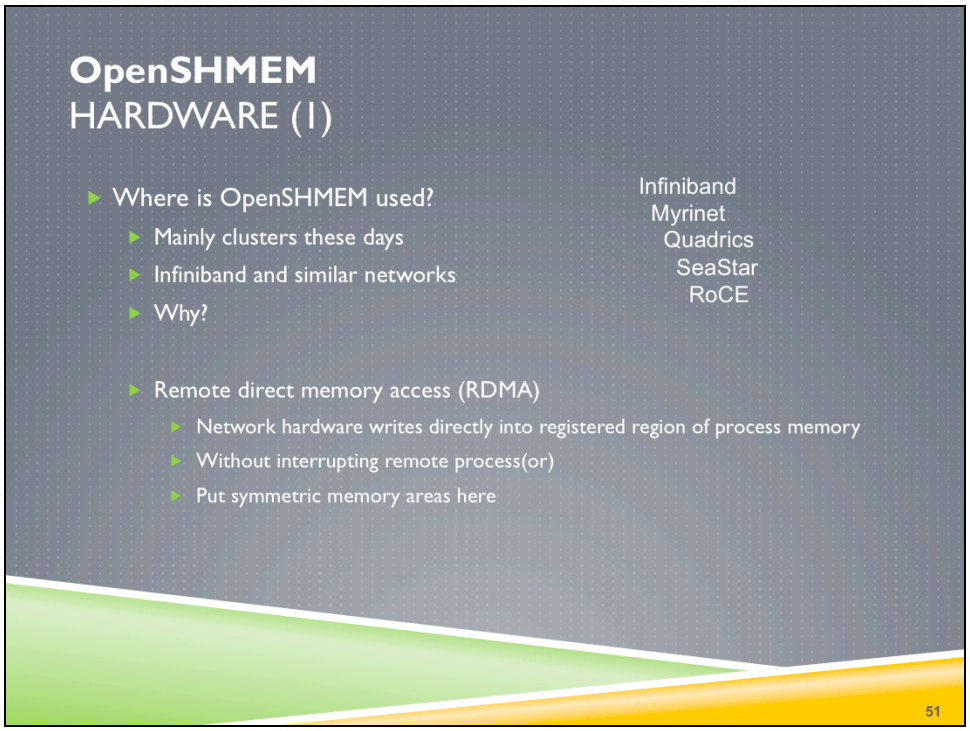

Fault-tolerance is a big deal. There are going to be lots of discussions about how and where to add FT and resilience capabilities to systems, and is a systemic issue, not just something to bolt onto OpenSHMEM.

# OpenSHMEM ADDRESSES & CACHE (1)

- Address manipulation
  - void *shmem_ptr(void *addr, int pe)
    - Returns a pointer to a data object on a remote PE
  - Only of use on platforms where memory physically accessible
  - i.e. puts/gets are simple memory accesses



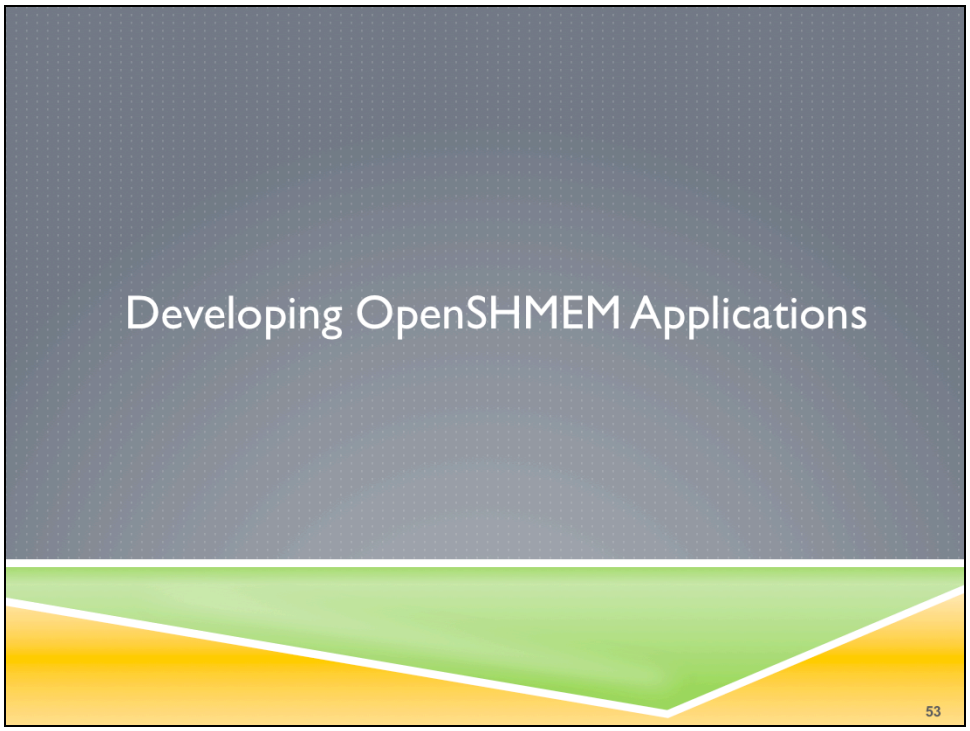

Access to cache lines was in one of the vendor SHMEMs. In OpenSHMEM, we're not going to deal with platform-specific cache needs.




This is possibly the most important thing in this tutorial. RDMA means we really can offload and overlap things. If you can make visible (register) all your symmetric memory areas with the network hardware then you can go faster with zero-copy and zero-interrupt.

## OpenSHMEM HARDWARE (2)

- Offload
  - Infiniband HCAs can do
    - Atomics
    - Collectives
    - Memory pinning
  - Meaning CPU free to do other things
  - Reduced software footprint (QPs)
  - OpenSHMEM library issues offload instructions rather than doing atomics etc.


Developing OpenSHMEM Applications
53

## OpenSHMEM
## LOOKING FOR OVERLAPS (1)

- How to identify overlap opportunities
  - Put is not an indivisible operation
    - Send local, reuse local, on-wire, stored
    - Can do useful work on other data in between


Express ideas about how to look for overlap. This can mean that simple attempts to do MPI -> OpenSHMEM translations could miss lots of opportunities. We need tools that can look at source code and make suggestions or changes. This is a plug for the OpenSHMEM Analyzer.

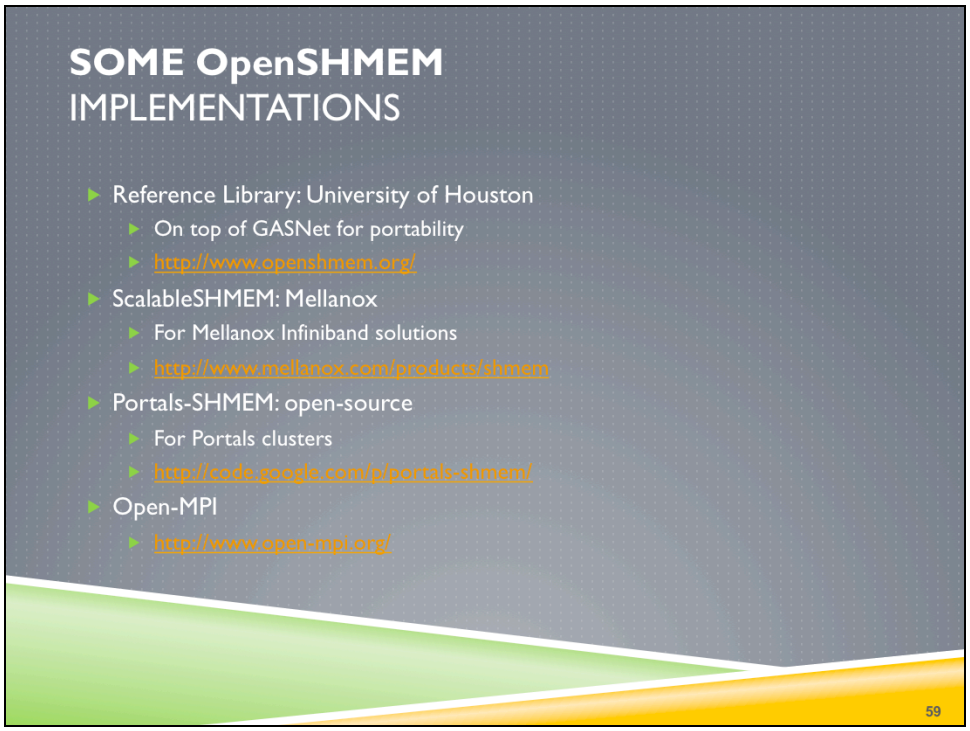
## OpenSHMEM
## LOOKING FOR OVERLAPS (3)

- How to identify overlap opportunities
  - How could we change OpenSHMEM to get even more overlap?
    - Divide application into distinct communication and computation phases to minimize synchronization points
    - Use of point-to-point synchronization as opposed to collective synchronization

56


More detail about real apps and their OpenSHMEM implementations will come from the ORNL material. ORNL will also have real performance data about scaling.


Talk about some of the proposed API extensions. Stress this is not "in camera" it is a public process.




We're not the only implementation out there. Other people have produced OpenSHMEMs.

# OpenSHMEM
## SUMMARY

- SPMD Library for C and Fortran programs
- Point-to-point data transfer
- Broadcast/collective transfer operations
- Synchronization
- Atomic operations

60

# OpenSHMEM REFERENCES

- Stephen W. Poole, Oscar Hernandez, Jeffery A. Kuehn, Galen M. Shipman, Anthony Curtis, and Karl Feind. OpenSHMEM - Toward a Unified RMA Model. Published in Encyclopedia of Parallel Computing, Springer US. Pages 1379-1391, 2011
- **OpenSHMEM and Related Technologies. Experiences, Implementations, and Tools**, First Workshop, OpenSHMEM 2014, Annapolis, MD, USA, March 4-6, 2014, Proceedings
- OpenSHMEM & Infiniband Research: visit DK Panda's group
  - http://nowlab.cse.ohio-state.edu/

# ACKNOWLEDGEMENTS

This work was supported by the United States Department of Defense & used resources of the Extreme Scale Systems Center at Oak Ridge National Laboratory.

62