# OpenSHMEM* on Portals

Keith D Underwood

March, 2014

# Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.



**Optimization Notice**

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

# OpenSHMEM* and the Impact of System Evolution

# OpenSHMEM* vs MPI-3 RMA

```
shmem_int_put(

  char *target,

  const char *source,
  size_t nelems,

  int pe
);

shmem_int_p(
  char *target,
  const char *source,
  int pe
);
```

```
MPI_Put(
  MPI_Datatype origin_dtype,
  MPI_Datatype target_dtype,
  MPI_Aint target_disp,
  MPI_Win win,
  const void *origin,
  int origin_count,
  int target_count,
  int target_rank,
);
```

# OpenSHMEM* : Thin and Light

OpenSHMEM* API is very thin

- As small as target PE, target address, and immediate data
- Also has source pointer and length version

Smallest implementations used 3 flit (8B flit) Put Request

- Limited node space
- Limited framing
- Depended on link level reliability
- As little as 1 flit response

T3D* /T3E* used more bits of overhead

| Element | Size |
|---|---|
| Target Node | 15 bits |
| Source Node | 15 bits |
| Framing | 34 bits |
| Target Address | 64 bits |
| 8B Payload | 64 bits |
| Total# | 24 B |
| Total# w/ Resp. | 32 B |

#These totals are not indicative of any specific implementation

# Early SHMEM* System Environment

Systems were small(er)

- No more than 2048 nodes
- Two cores per node

Memory was symmetric

- Simplifies programming and implementation

Round-trip bandwidth delay product was small

- Hundreds of megabytes per second
- 1.5 μs of round-trip latency

Applications were simple

- Written in a single API/language
- Written by a single author (effectively)

MPI was new

- Could be layered on SHMEM* at small scale

Systems were single user (sort of)

- At a minimum, partitioned
- Limited need for protection

Filesystems were (almost) an afterthought

# System Characteristics: Then and Now

| Characteristic | Then | Now |
|---|---|---|
| Maximum Scale (Nodes) | 2048 | 128K |
| Cores per Node | 2 | 16-60 (and growing) |
| Memory per Node | Hundreds of MB | Tens of GB |
| OS | Microkernel | “Standard” Linux* |
| Protection need | “Hard” Partitioned | Multi-user |
| Network Ordering | Deterministic Request / Adaptive Response | Adaptive |
| Bandwidth | 150 MB/s | 10 GB/s |
| Round-trip delay | 1.5 μs | 1.5 μs |
| Message Rate | 18 Mmsgs/s | 120 Mmsgs/s |
| Messaging API | None Established | MPI dominant, PGAS emerging |
| Applications | (relatively) simple | Tens of Libraries |

# What do those changes bring?(1)

More bits! Lots more bits…

- Bigger LID space
- Many more packets in flight (transaction tracking bits)
- Protection
  - Pkey
  - Other protection bits
- End-to-end reliability

| Item | Bits | Impacts |
|---|---|---|
| LID | +4 | Both |
| transactions | +3 | Both |
| PKey | +16 | Both |
| End-to-End | +32 | Both |
| Sequence # | +16 (or more) | Both |
| RKey | +32 | Request |

# What do those changes bring? (2)

Harder translation

- Local to the target node, and process context specific
- Hard to do at system level
- Deeper page table hierarchies

Symmetric addresses are harder to guarantee

- Can be done (if you turn off some options in Linux*)…
- …most of the time

Fence is not free

- Fence is typically quiet

More process contexts

- Wire must ID context at initiator/target
- Allow more processes than cores (resource allocation issue)

# OpenSHMEM* over Portals 4

http://code.google.com/p/portals-shmem/

# Portals Philosophy

Use building blocks…

- Use composable pieces
- Maintain API symmetry where reasonable
- If done properly, one functionality serves many needs

…to capture application semantics for many APIs…

- Portals does not implement fence(), quiet(), or barrier(), but has the tools to build them
- Portals is not MPI_Isend()/MPI_Irecv(), but does handle:
  - Matching
  - Unexpected messages

…in a way that is friendly to offload.

- Blocks are “simple”

# Attempt #1: Initial Extensions to Portals 4

Added several general purpose building blocks

- Nonmatching interface
- Counting events for lightweight completion
- Breadth of atomic operations
  - MPI sets a high bar for this one
- Triggered operations for collectives

Achieved fundamental goal: building blocks were reusable

- Counting events + Atomic operations + triggered operations == collectives

# What We Learned

Nonmatching interface must be very limited to get it right

- Wanted something that could be fully pipelined
- Multiple substantial revisions to achieve that

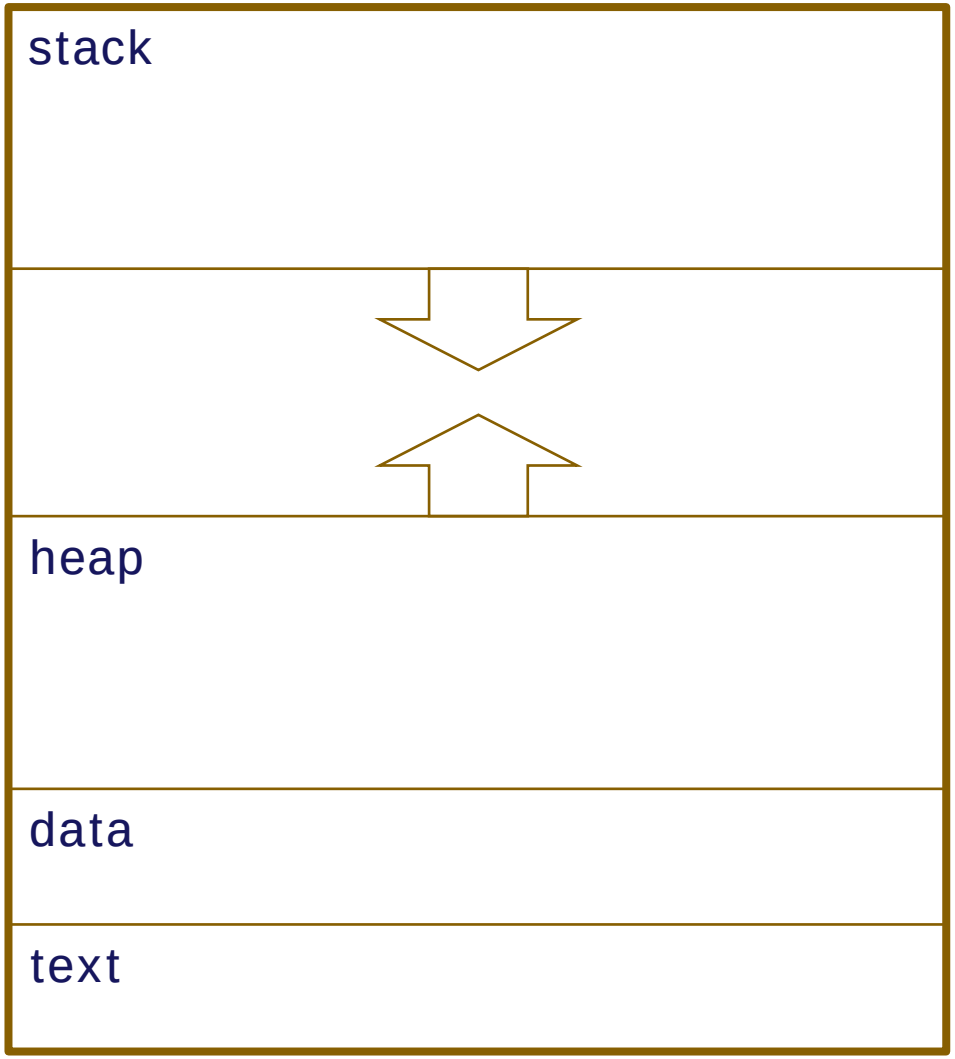
Needed a path for blocking puts

- SHMEM* puts expected to be locally complete on return
- Portals puts were fully nonblocking
  - Added query for size below which Portals could locally complete

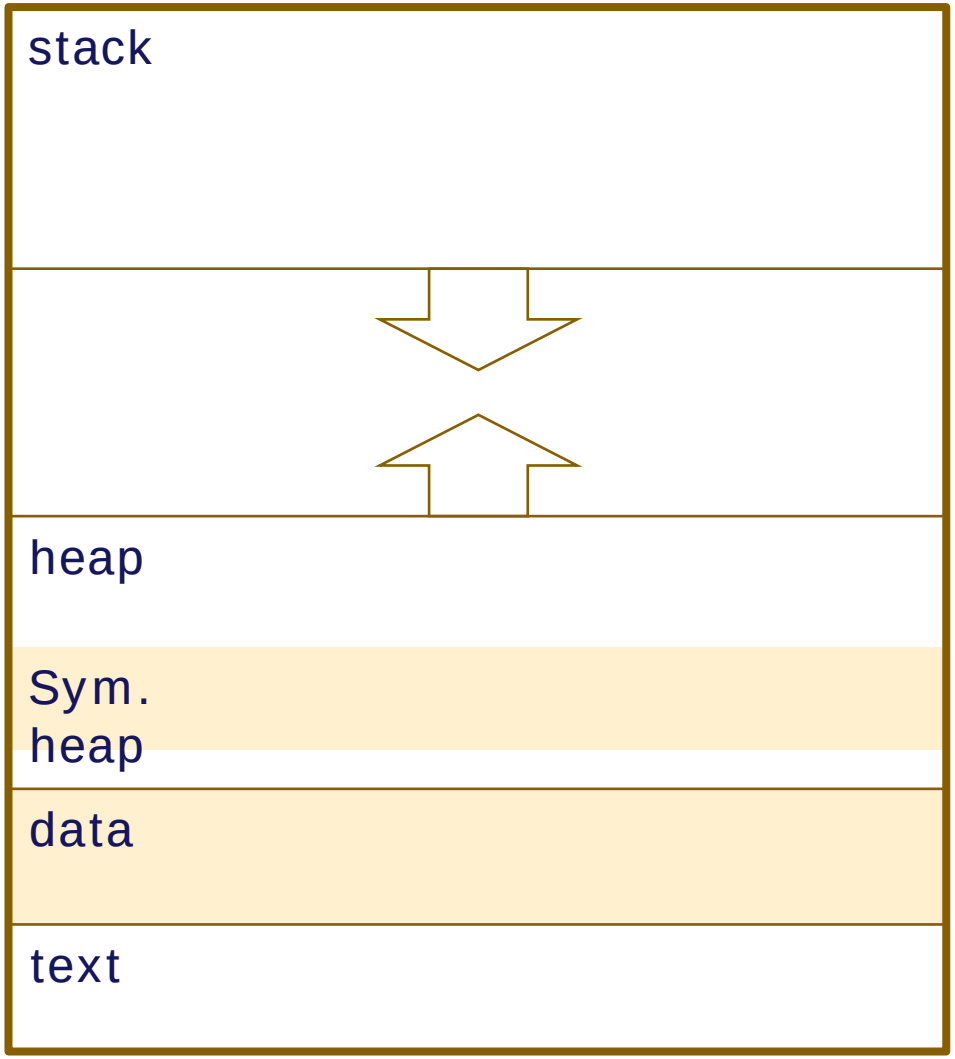
Ordering is… hard…

- Many, many assumptions about ordering, atomicity, operation interactions in various PGAS implementations
- Completely rewrote the ordering definition

# Memory Layout



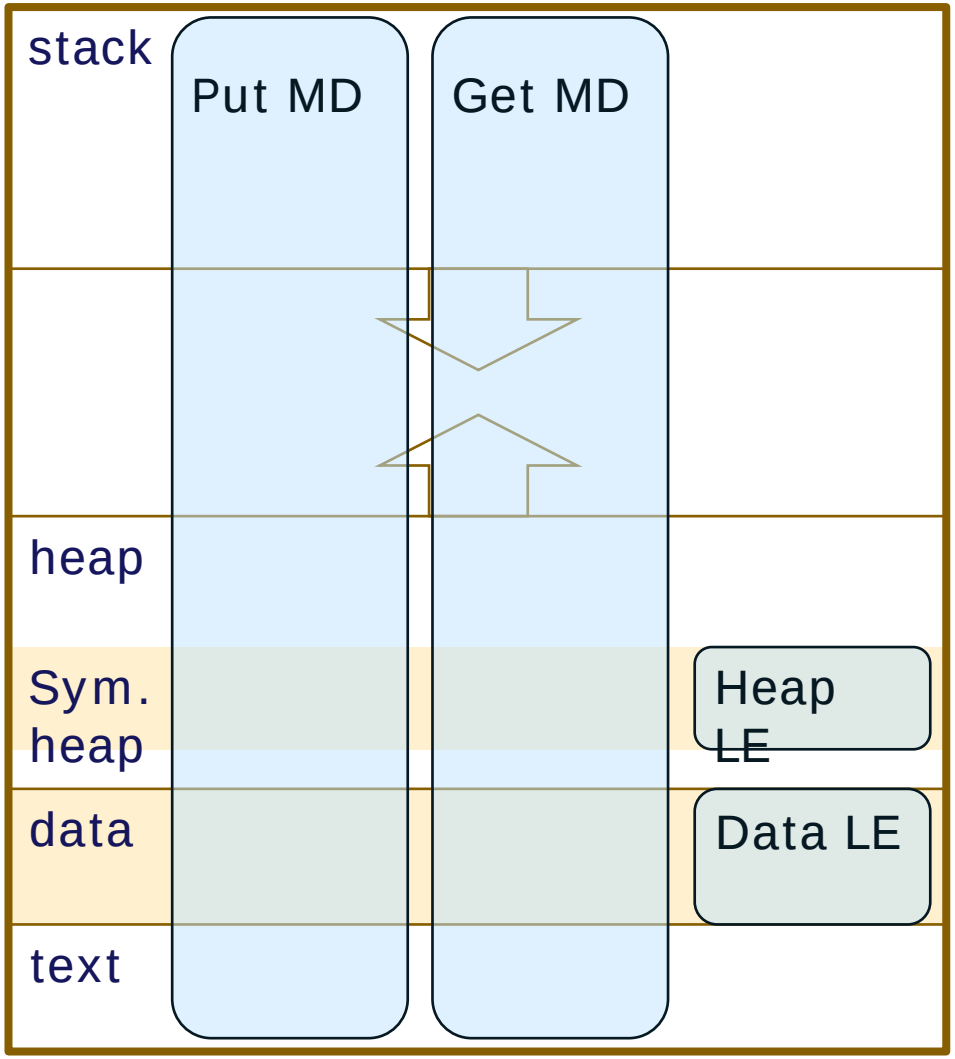
# Memory Layout






Optimization Notice

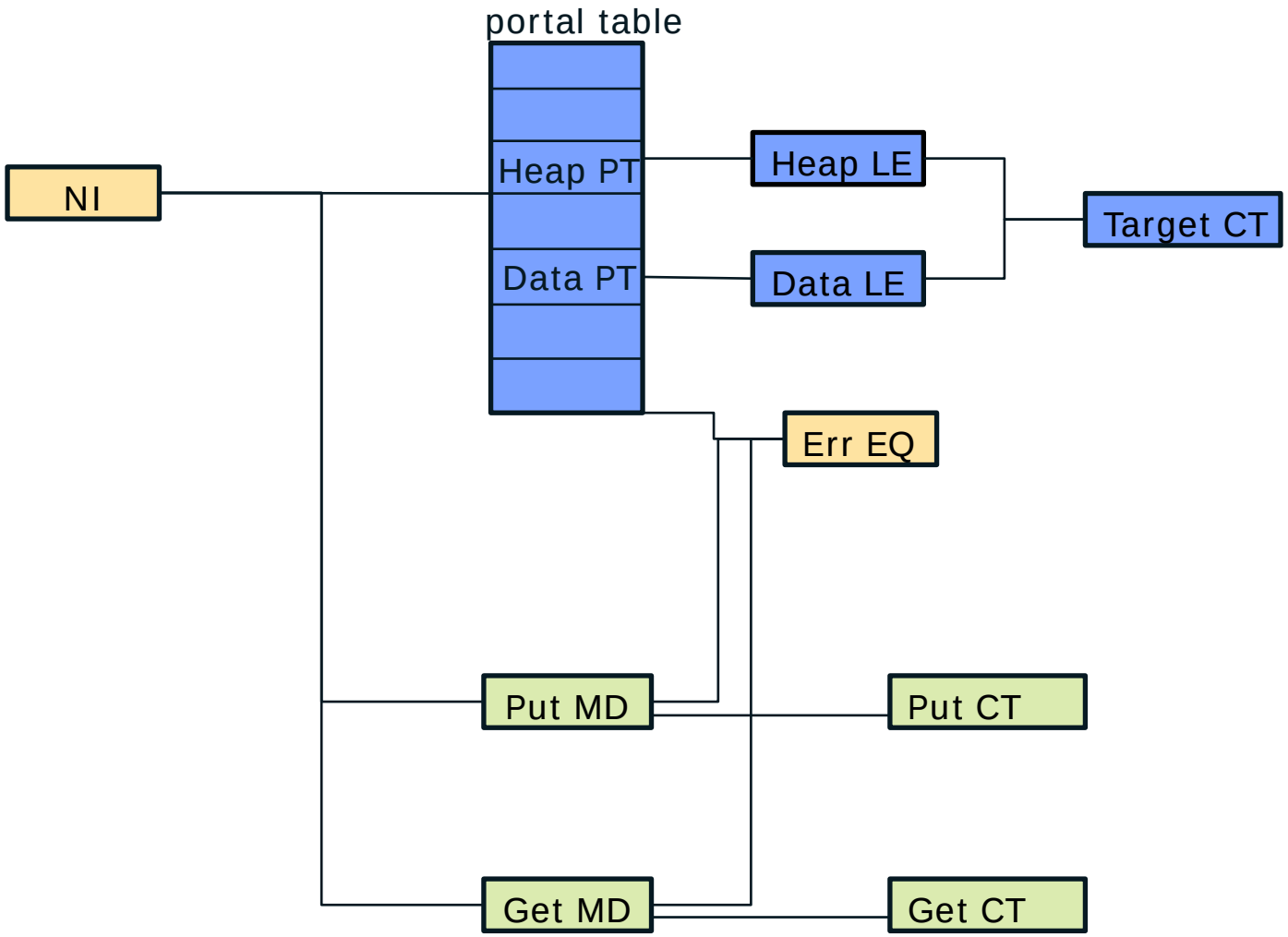
# Memory Layout






Optimization Notice

# Portals Data Structures



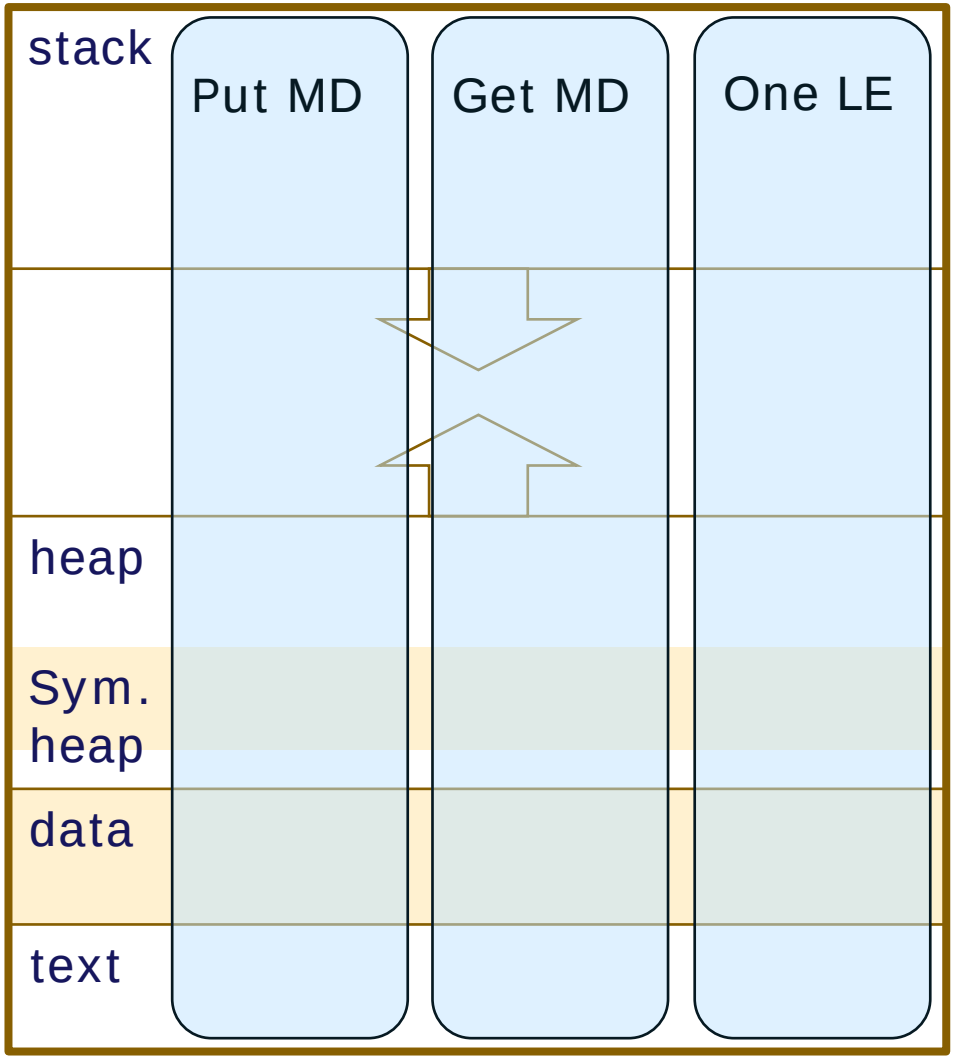
# Put Operations

```
void shmem_long_p(long *addr, long value, int pe) {
    ptl_process_t peer;
    ptl_pt_index_t pt;
    long offset;
    peer.rank = pe;
    GET_REMOTE_ACCESS(addr, pt, offset);

    PtlPut(shmem_internal_put_md_h,
           (ptl_size_t) &value,
           sizeof(value),
           PTL_CT_ACK_REQ,
           peer,
           pt,
           0,
           offset,
           NULL,
           0);

    shmem_transport_portals4_pending_put_counter++;

}
```

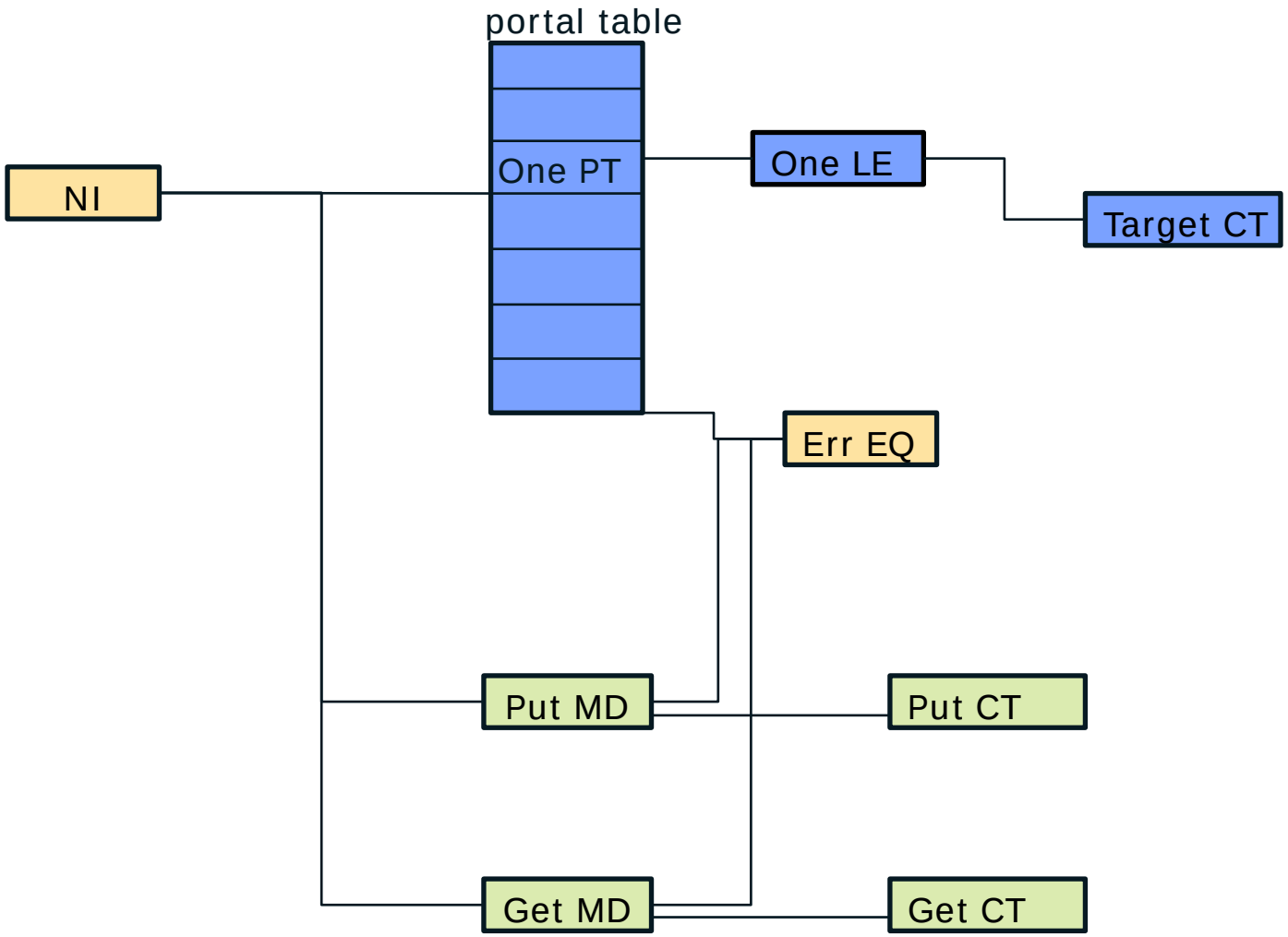
# GET_REMOTE_ACCESS()

```
#ifdef ENABLE_REMOTE_VIRTUAL_ADDRESSING
#define PORTALS4_GET_REMOTE_ACCESS_ONEPT(target, pt, offset)                \
  do {                                                                      \
    pt = (one_pt);                                                          \
    offset = (uintptr_t) target;                                            \
  } while (0)
#else
#define PORTALS4_GET_REMOTE_ACCESS_TWOPT(target, pt, offset)                \
  do {                                                                      \
    if ((void*) target < shmem_internal_heap_base) {                        \
      pt = (data_pt);                                                       \
      offset = (char*) target - (char*) shmem_internal_data_base;           \
    } else {                                                                \
      pt = (heap_pt);                                                       \
      offset = (char*) target - (char*) shmem_internal_heap_base;           \
    }                                                                       \
  } while (0)
#endif
```

# Memory Layout



# Variant: Portals Data Structures



# Get Operations

```
void shmem_double_get(double *target, const double *source,
                      size_t len, int pe) {
   ...

   ptl_ct_event_t ct;
   peer.rank = pe;
   GET_REMOTE_ACCESS(source, pt, offset);

   PtlGet(shmem_internal_get_md_h,
          (ptl_size_t) target,
          len * sizeof(double),
          peer,
          pt,
          0,
          offset,
          0);
   shmem_internal_pending_get_counter++;
   PtlCTWait(shmem_internal_get_ct_h,
         shmem_internal_pending_get_counter,
         &ct);

}
```

# Fence & Quiet

```
void shmem_quiet(void) {
    ptl_ct_event_t ct;

    /* wait for remote completion (acks) of all pending events*/
    PtlCTWait(shmem_internal_put_ct_h,
              shmem_internal_pending_put_counter, &ct);

}


void shmem_fence(void) {
   if (shmem_internal_total_data_ordering == 0) {
       shmem_quiet();
    }
}
```

# Address Wait

```
void shmem_int_wait(int *var, int value) {
  ptl_ct_event_t ct;

  while (*var == value) {
    PtlCTGet(target_ct_h, &ct);
    if (*var != value) return;
    PtlCTWait(target_ct_h, ct.success + 1, &ct);
  }
}
```

# Lock

Implements MCS locks

Encodes last/next/signal into one 64 bit field

Uses masked swap to change individual bits

# Strided Operations: Still Open…

Portals 4 team has evaluated strided operations multiple times

Strided operation definitions are not very consistent across APIs

- SHMEM*
- MPI
- GASNet*
- ARMCI*

# Impacts on Hardware

# Some Challenges

Remote completions: is a quiet really quiet?

- PCI* Express uses a posted write model
- Hard to know when it is “really done” without a lot of overhead

Atomics have implementation challenges

- Caches on a PCI* Express device would require a way to flush that cache to re-enter a “safe” state. When would you do that?
- No caches on a PCI* Express device mean repeated atomics (e.g. lock contention) would be slow

Hardware collective engines require setup

- Would be nice to have an allocated descriptor of PEs that will be in a collective

# System Scale Challenges

End-to-end reliability and blocking puts are incompatible

- Old way:
  - TCP: we will copy your data into the kernel and do end-to-end retransmit
  - RDMA: we will retransmit from user space
  - Custom networks: we will send your data once and work really hard on getting it there
- End-to-end (user space to user space) retransmits is one of the few tools left in the reliability toolbox

Where did I put that PE ?

- At 10 million PEs, there is a table lookup that is bad
- Hardware can help – if you can help us help you

# Longer Term

How can we make OpenSHMEM* better?

- SHMEM* has always been focused on being a thin layer
- Let's keep it that way
  - Perform hardware / software co-design
  - Only add the things that hardware can reasonably support

# Thoughts on the Future

# Evolution

Threading: we need a definition of how threading works

- This will impact hardware
- This will take time to get right…
  - …for the API/users
  - …for the hardware

Learn from MPI: Communicators are probably too heavyweight, but…

- Collectives could use a place to attach a fixed set of nodes
- It would be nice to support layered libraries
  - Collective isolation
  - Completion semantics
  - Library relative addressing / protection

Consider fault tolerance: what trade-offs belong in hardware vs software?

- Portals can tell you when individual messages fail:
  - Can you use that information?
  - How would OpenSHMEM* tell the user?

# Think about the ecosystem

Transparency is good

- Set objectives for releases
- Set target dates for releases
- Discuss everything before changing it: the crowd can be wiser than you think
- Prototype ***everything*** in open source before adding it

Don't move randomly

- Document motivations
- Be transparent

Don't break compatibility (often)

- Extensions are good, changing existing semantics is bad

Don't require weird OS hooks or configurations

- "Most of the time" is not good enough

# Call for Participation: OFA WG

https://www.openfabrics.org/downloads/OFWG/

http://lists.openfabrics.org/cgi-bin/mailman/listinfo/openframeworkwg

https://www.openfabrics.org/component/content/article/167-sept-18-2013-openfabrics-alliance-announces-formation-of-openframework-working-group.html

intel®