On Demand Paging for User-level Networking

Liran Liss

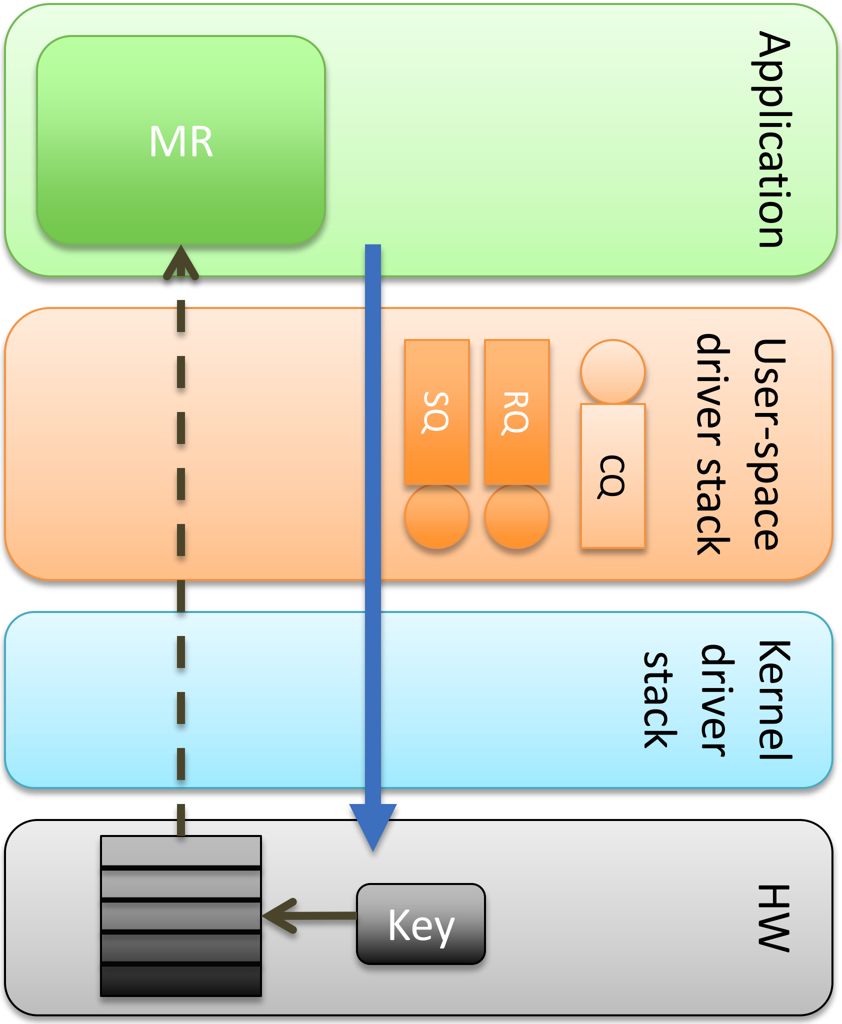
Mellanox Technologies

Agenda

- Memory registration
- RDMA programming challenges
- On Demand Paging (ODP)
- Page pre-fetching
- Initial testing
- Future work
- Conclusions

Memory Registration

- Apps register Memory Regions (MRs) for IO
  - Referenced memory must be part of process address space at registration time
  - Memory key returned to identify the MR
- Registration operation
  - Pins down the MR
  - Hands off the virtual to physical mapping to HW

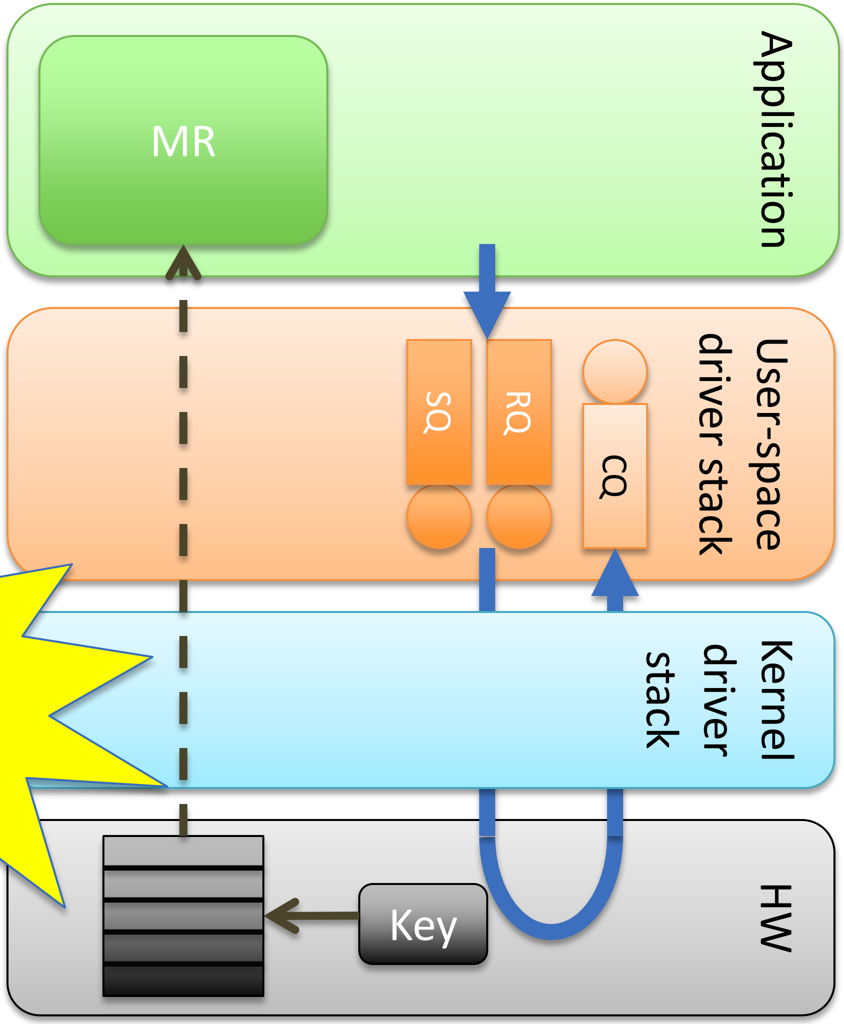


Memory Registration – continued

- Fast path
  - Applications post IO operations directly to HCA
  - HCA accesses memory using the translations referenced by the memory key

Wow !!!

But…



Challenges

- Size of registered memory must fit physical memory
- Applications must have memory locking privileges
- Continuously synchronizing the translation tables between the address space and the HCA is hard
  - Address space changes (malloc, mmap, stack)
  - NUMA migration
  - fork()
- Registration is a costly operation
  - No locality of reference

Achieving High Performance

- Requires careful design
- Dynamic registration
  - Naïve approach induces significant overheads
  - Pin-down cache logic is complex and not complete
- Pinned bounce buffers
  - Application level memory management
  - Copying adds overhead

On Demand Paging

- MR pages are never pinned by the OS
  - Paged in when HCA needs them
  - Paged out when reclaimed by the OS
- HCA translation tables may contain non-present pages
  - Initially, a new MR is created with non-present pages
  - Virtual memory mappings don’t necessarily exist

Semantics

- ODP memory registration
  - Specify IBV_ACCESS_ON_DEMAND access flag
- Work request processing
  - WQEs in HW ownership must reference mapped memory
    - From Post_Send()/Recv() until PollCQ()
  - RDMA operations must target mapped memory
  - Access attempts to unmapped memory trigger an error
- Transport
  - RC semantics unchanged
  - UD responder drops packets while page faults are resolved
    - Standard semantics cannot be achieved unless wire is back-pressured

Advantages

- Simplified programming
  - MPI rendezvous without dynamic registrations
  - No dedicated buffer pools to manage
- Practically unlimited memory registrations
  - No special privileges are required
- Physical memory optimized to hold working set

```
SendSomething()
{
    char buf[SIZE];
    WQE wqe;
    ...
    FillBuf(buf);
    wqe.sge[0].addr = buf;
    wqe.sge[0].length = SIZE;
    wqe.sge[0].lkey = STACK_KEY;
    ...
    Post_Send(wqe);
    while (!PollCQ());
}
```

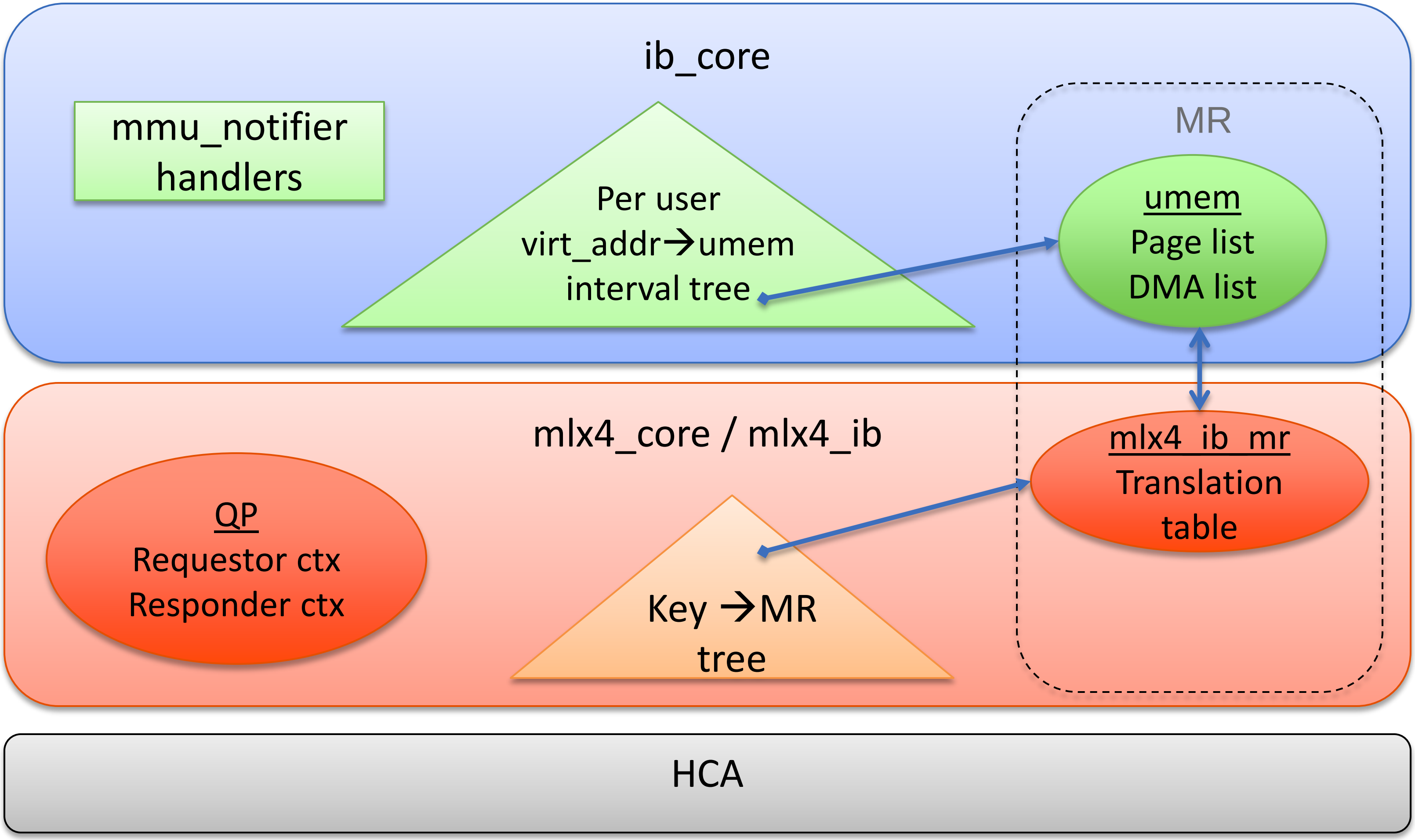
Design

- Kernel only
  - Transparent to applications
- Generic code (ib_core) tasks
  - Manage page invalidations
    - Register for MMU notifier calls
    - Provide context for invalidations
    - Locate intersection between page invalidations and MRs
  - Support page faults
    - Synchronize between invalidations and page faults
    - Page-in user pages and map to dma

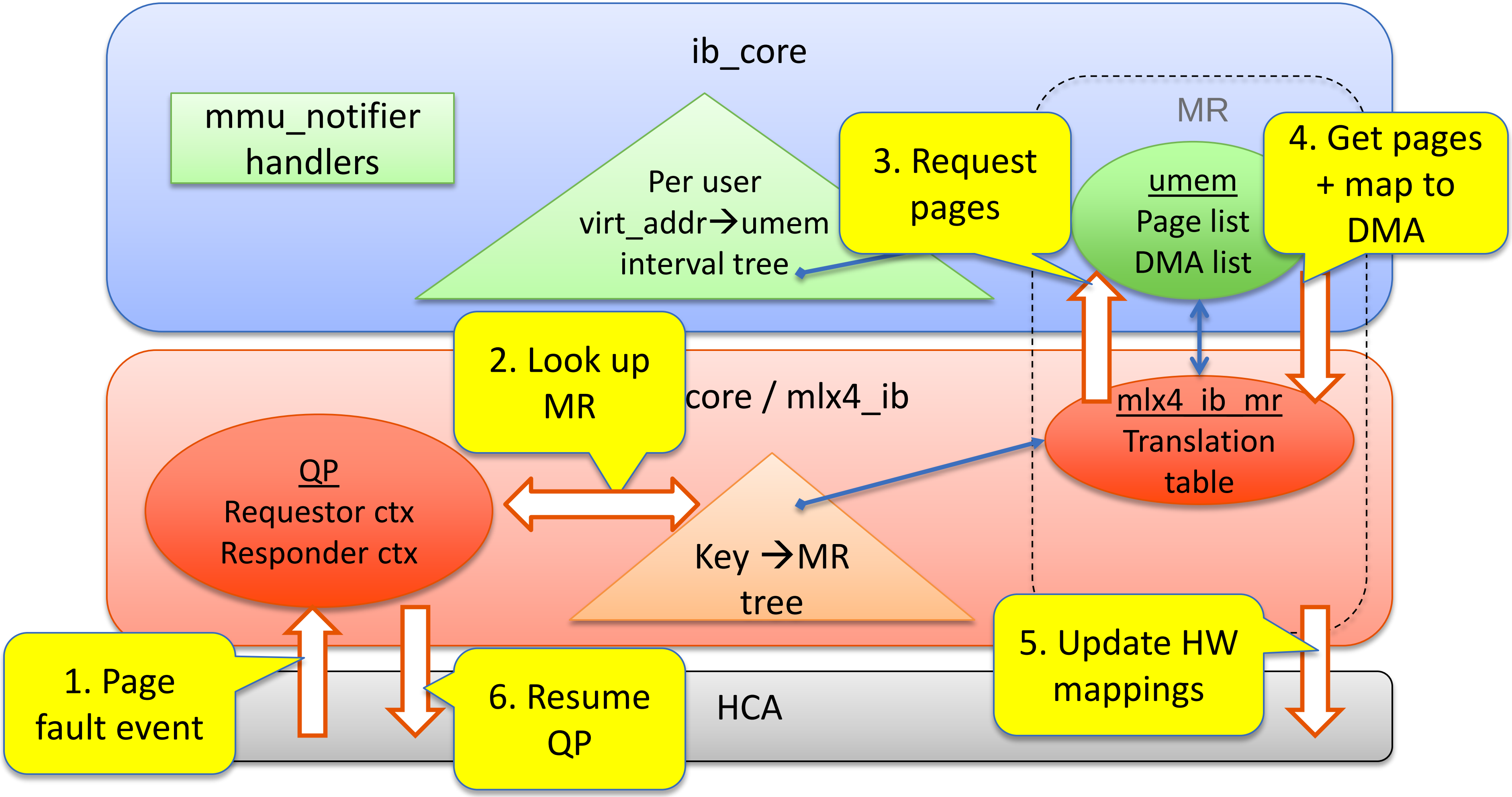
Design – continued

- Driver code (mlx4_core/ib) tasks
  - Process page faults
    - Catch and classify HW page faults
    - Provide context for page faults
      - Per-QP work_struct for requester/responder
  - Handle HW page invalidations

Data Structures

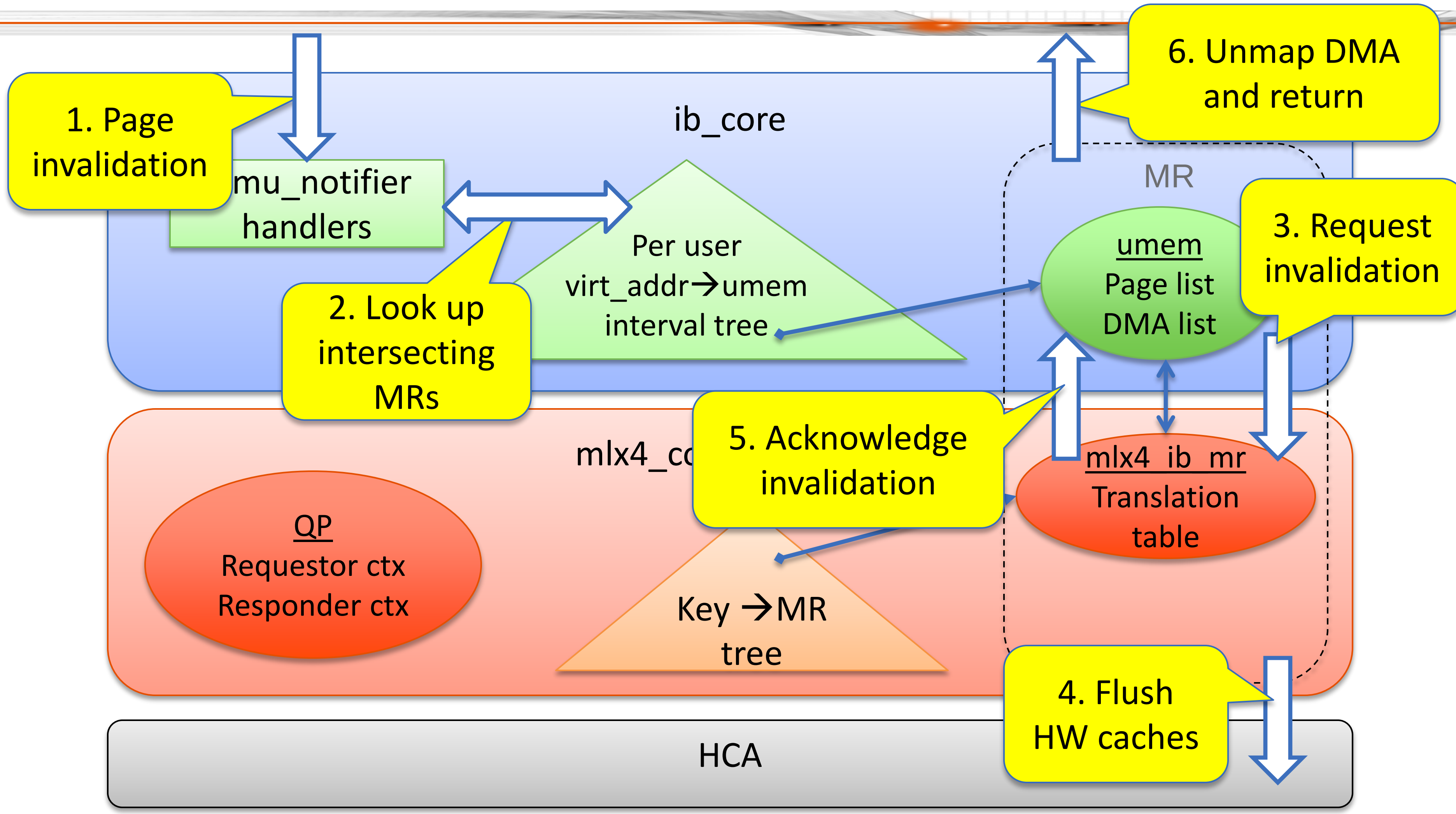


Page-in Flow



ib_core
mmu_notifier handlers
Per user virt_addr→umem interval tree
3. Request pages
MR
umem
Page list
DMA list
4. Get pages + map to DMA
2. Look up MR
core / mlx4_ib
mlx4_ib_mr
Translation table
QP
Requestor ctx
Responder ctx
Key →MR tree
1. Page fault event
6. Resume QP
HCA
5. Update HW mappings

Invalidation Flow



Page Pre-fetching

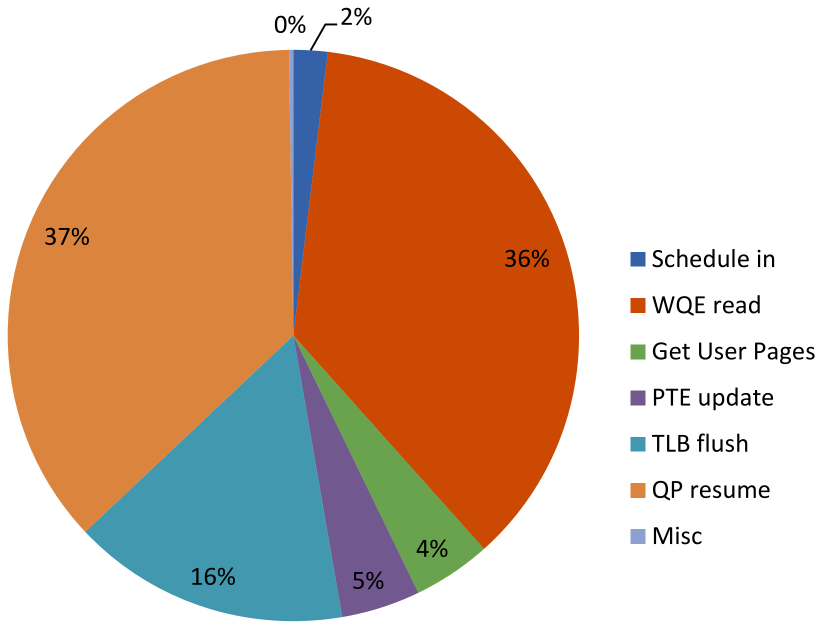
- New Verb for pre-fetching pages
- Uses
  - Warming up new memory mappings
  - MPI rendezvous optimization
  - UD responder optimization

Initial Testing

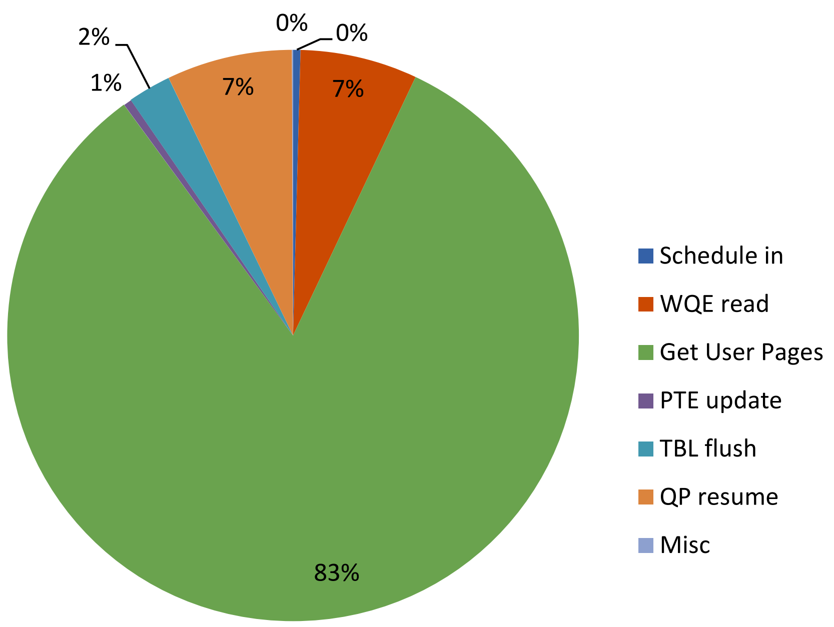
- ODP support
  - Implemented all RC transport flows for IB and RoCE
    - Excluding SRQ and memory windows
  - UD over IB and RoCE
  - Raw Ethernet QP
- Inter-operability
  - Latency of non-ODP applications running concurrently hardly affected
  - Mixed requestors/responders also work well
- Native performance for memory-resident ODP pages
- Page-in performance
  - 4K page fault takes approximately 135us
  - 4M page fault takes approximately 1ms

Execution Time Breakdown (Send Requestor)

4K Page fault (135us total time)



4M Page fault (1ms total time)



Future work: Huge MR Support

- Support MRs in the size of TBs
- Implicit ODP
  - Register complete application address space up-front
  - Effectively eliminate memory registration
- Meta-data size must be a function of currently mapped memory instead of MR size
  - Applies to all data structures (IB core, driver, and HW)
- Memory Windows (MWs) become the main vehicle for controlling access rights

Future Work: Improve OS integration

- Update PTE accessed/dirty bits according to IO accesses
- Page invalidation batching
  - Page eviction in the swapper
  - NUMA migration process
- Extend ODP to guest physical→machine translations for virtualization

Conclusions

- RDMA performance is great
  – But requires careful design
- ODP simplifies RDMA programming and deployment
  – Moves memory management to the OS
  – Lifts memory-pinning limits
- ODP does not sacrifice performance or interoperability
- ODP eliminates memory registrations!!!
  – Coming up soon

Thank You

Backup

OPENFABRICS ALLIANCE

Concurrent Page Faults

- Each QP has at most 2 concurrent page faults
  - Requestor
  - Responder
- Faulting QP temporarily suspended until fault is resolved by SW
  - Even if another QP satisfies the fault in the meantime
  - Required for correct completion semantics

Page-in / Invalidation Races

- Invalidations may race with page faults
  - HW will complete all in-flight memory accesses to an invalidated range before completing the invalidation
  - New accesses will trigger a page fault normally
- Page-in requests are not serviced while handling mmu_notifier invalidations
  - QP is resumed without updating the page tables
  - HW will retry access optionally triggering another page fault
  - Simplifies the code considerably

Forward Progress

- Challenge
  - Single MTU-sized packet may refer to multiple S/G entries in WQEs
  - Single RDMA-W transaction may span multiple pages
- Forward progress generally ***not*** guaranteed
  - Pages are not pinned → inherent race with page invalidation
  - Not any different than CPU accesses
- Alleviate by paging-in multiple pages at once
  - Read multiple SGEs in WQE page faults
  - Pre-fetch large consecutive ranges in RDMA faults
- Not an issue in practice