# Symmetric Remote Keys

UCF 2023 – Dec 7

Thomas Vegas (UCX team)

Artem Polyakov (Arch team)

# OpenSHMEM

- **OpenSHMEM is**
  - an implementation of Partitioned Global Address Space (PGAS) parallel programming model for HPC
  - a standard API targeting portable and uniformly predictable results of OpenSHMEM programs
  - comprised of Processing Elements (PE) - typically OS processes - communicating with each other using OpenSHMEM primitives

- **OpenSHMEM memory model**
  - An OpenSHMEM program consists of data objects that are **private** to each PE and data objects that are **remotely accessible** by all PEs
  - Remotely accessible data objects are called **Symmetric Data Objects**.
  - For symmetric data, each object is represented on all PEs with same name, type, and size
  - Symmetric data is remotely accessed via OpenSHMEM API
  - Symmetric data objects are referenced in OpenSHMEM operations through the **Symmetric Address** – a pointer to the **local object** that corresponds to the desired **remotely accessible object**.
  - Symmetric objects are grouped in SHMEM Segments

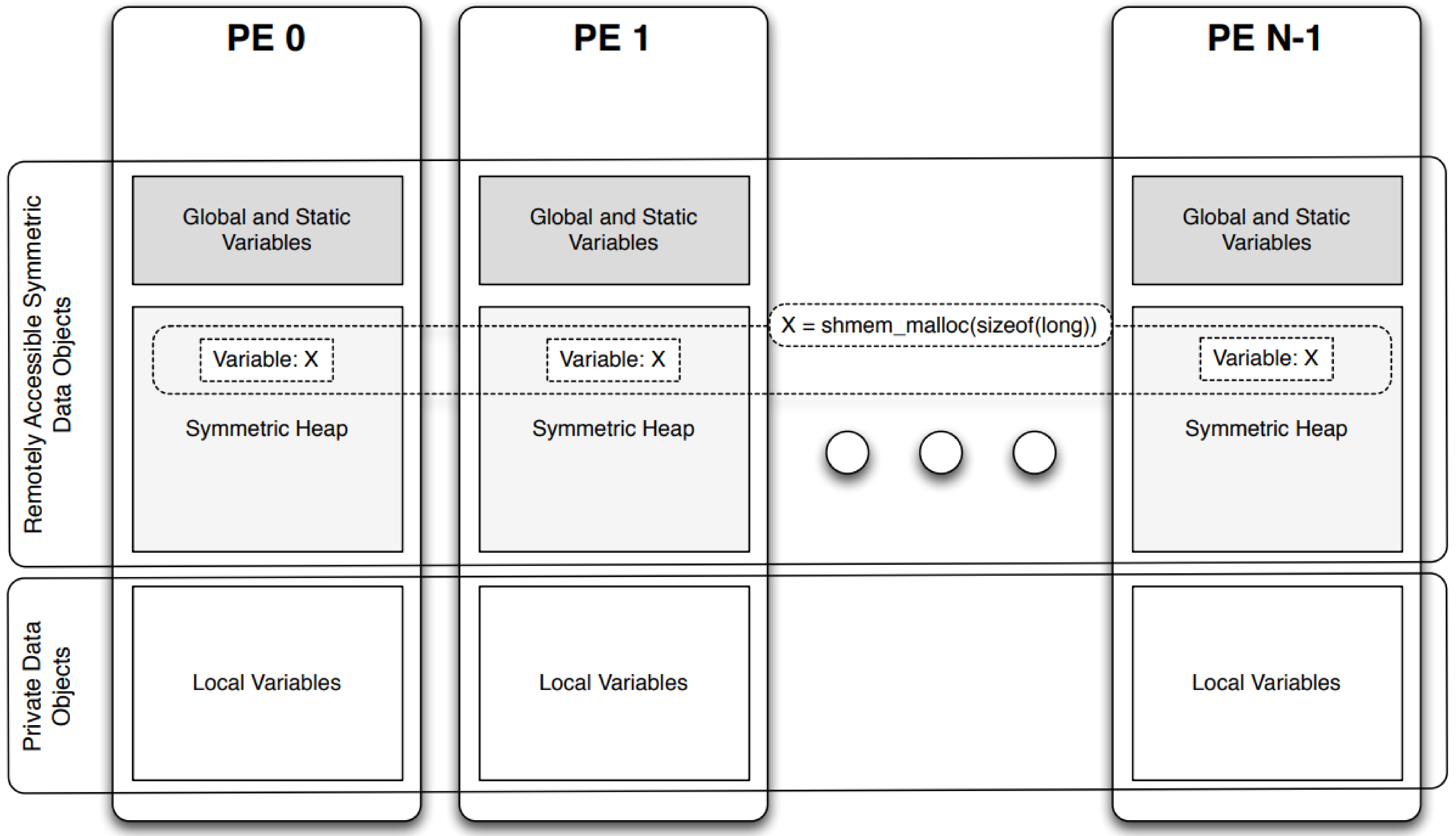

PE 0
PE 1
PE N-1
Remotely Accessible Symmetric Data Objects
Private Data Objects
Global and Static Variables
Variable: X
Symmetric Heap
X = shmem_malloc(sizeof(long))
Local Variables

Figure 1: OpenSHMEM Memory Model

## Symmetric Remote Keys

OpenSHMEM as first user

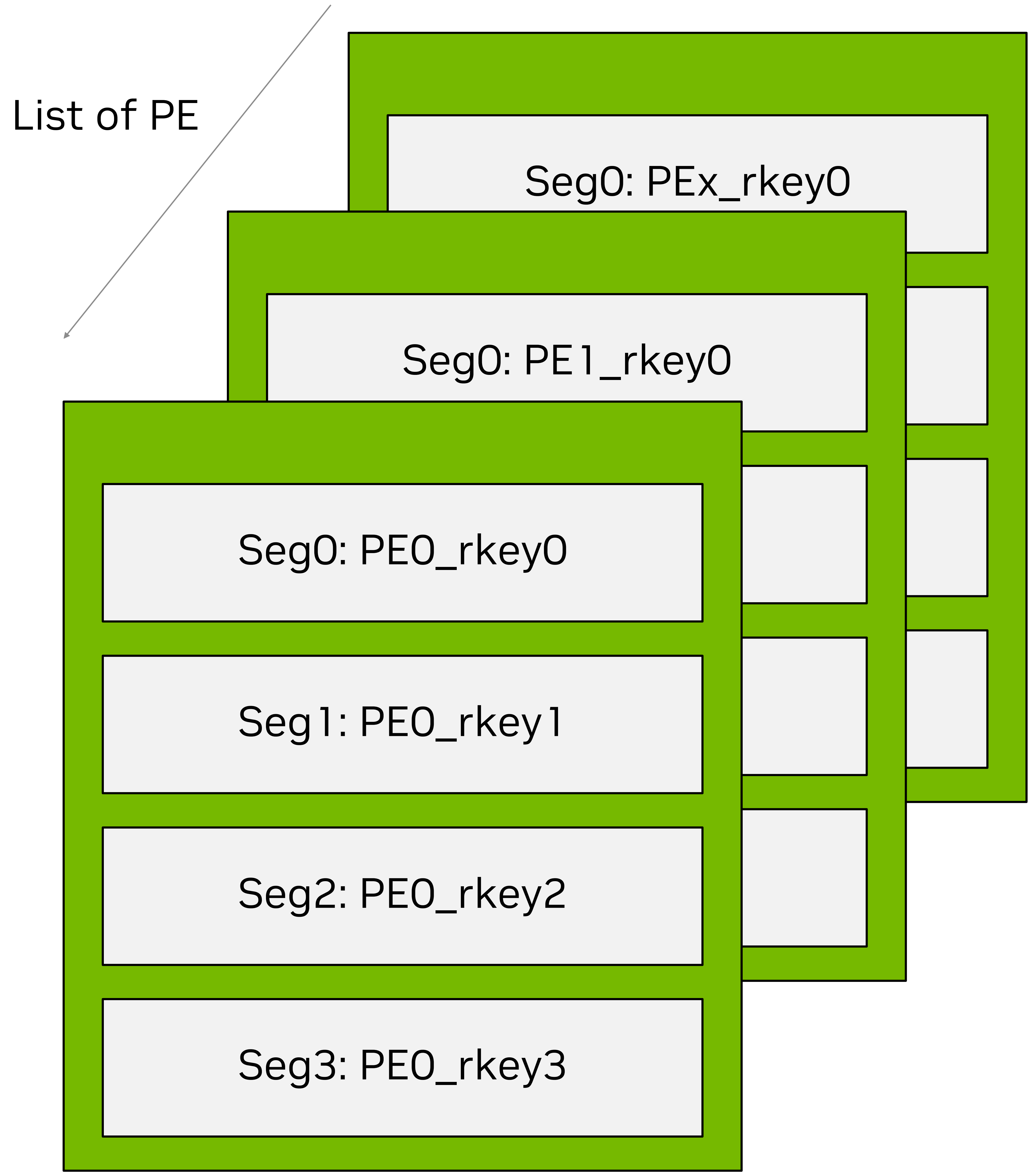
- **OpenSHMEM**
  - Shared memory over multiple processes on multiple nodes
  - Uses UCX, UCP API to perform data transfers / atomic operations

- **UCX**
  - Communication framework for high-bandwidth and low-latency network

- **Goal – reduce the number of used remote keys**
  - Implementing symmetric remote keys in UCX
  - Extend Infiniband memory registration

# OSHMEM: Manipulates segments

- **OSHMEM PE:** Processing Element
  - One or many PE per node (PPN)
- **Each PE (rank) creates the exact same list of segments**
  - A segment is a contiguous memory region
  - Created simultaneously by all PEs
  - Typical 3-5 segments
  - SHMEM allocation functions are operating within the segments
- **Each segment is registered to be remotely accessible**
- **Each PE can access every remote segment in the application**
- **Each PE has remote key for every segment of all other PEs**



# UCP Remote key

On a given cluster

- **For OSHMEM use case**
  - RDMA network is used for inter-node communication (i.e. InfiniBand)
  - Shared memory is utilized for intra-node communications
  - HPC nodes are typically homogeneous (hw, network)
- **UCP Remote key**
  - Contains one memory key per memory domain
- **`ucp_rkey_h` descriptor has**
  - A list of remote keys for each transport
  - But also:
    - Memory domain contained
    - Remote configuration identifier
    - Endpoint configuration index

## Suggestion

- **For OSHMEM use case**
  - RDMA network is used for inter-node communication (i.e. InfiniBand)
  - Shared memory is utilized for intra-node communications
  - HPC nodes are typically homogeneous (hw, network)
- **Avoid storing extra information for homogeneous systems**
  - Deduplication meta-information related to
    - memory domains
    - Endpoint configurations
- **Reduce the number of unique inter-node keys**
  - Demonstrated for InfiniBand via a capability to reduce the randomization of remote keys
    - This allows to deduplicate at IB transport level
- **Shared memory keys**
  - Do not scale with the number of nodes and do not require deduplication

# Introduce deduplication procedure 1/2

UCX: UCP new API

- **API Change: Added new flag for** `ucp_ucp_mem_map()`
  - `UCP_MEM_MAP_SYMMETRIC_RKEY`
  - Indicates that the memory is allocated in environments with symmetric memory
    - There is a potential benefit from deduplication
  - UCP is making the best effort to support
    - Non-supporting transport work as usual
- **New API, works with every transport**
  - 
```
ucs_status_t ucp_rkey_compare(ucp_worker_h worker, ucp_rkey_h rkey1, ucp_rkey_h rkey2,
                              const ucp_rkey_compare_params_t *params, int *result);
```
- **OSHMEM procedure**
  1. Receive and unpack rkey
  2. Compare all rkey parameters with existing received and unpacked keys in the bucket
  3. If identical: discard latest, reuse previous reference ucp_rkey_h
  4. Else if different: add to store bucket, use current ucp_rkey_h
- **For OSHMEM transport keys**
  - Shared memory – not supported
  - Infiniband – is supported

# Introduce deduplication procedure 2/2

## UCX: UCP new API

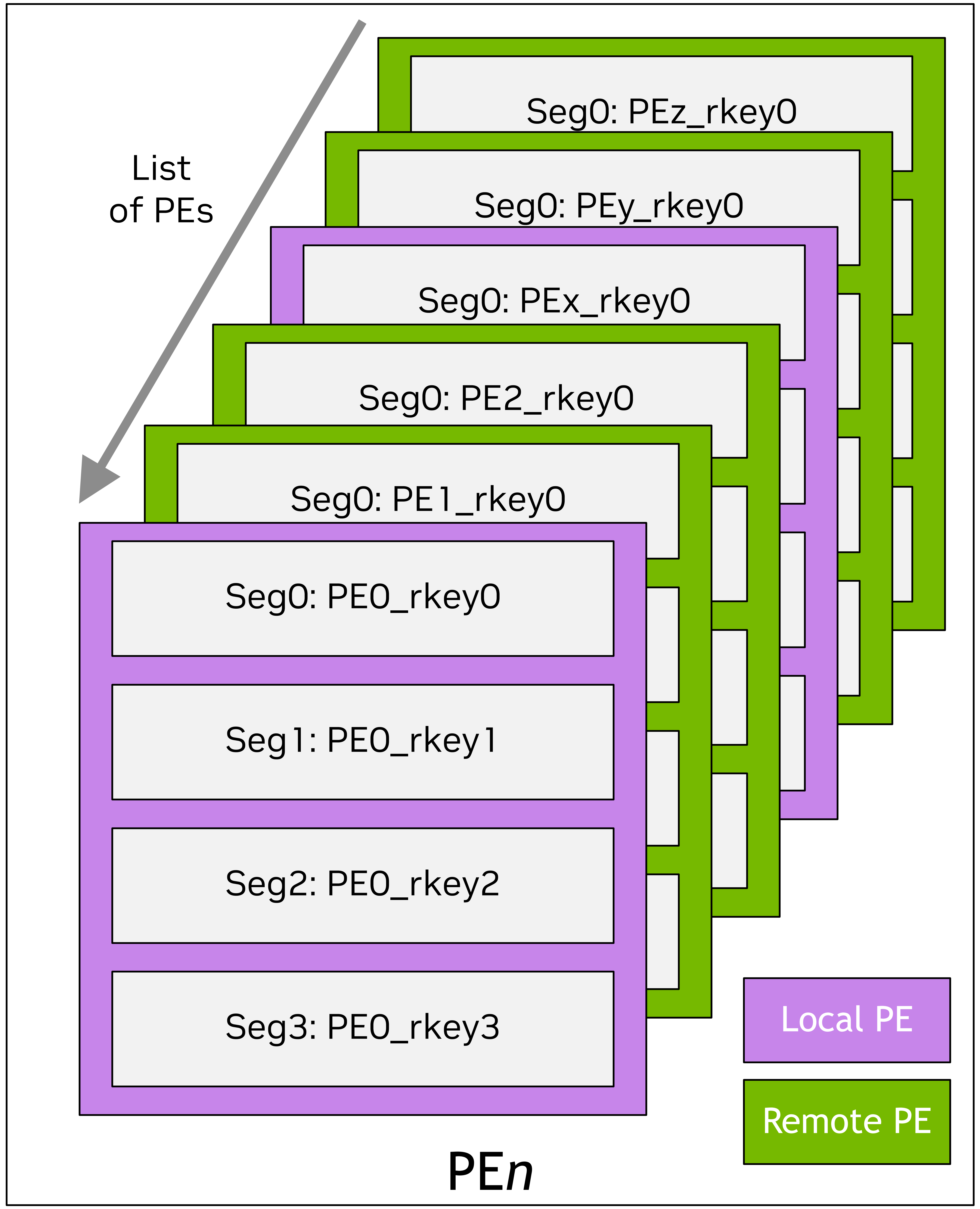
- **API Change: Added new flag for** `ucp_ucp_mem_map()`
  - `UCP_MEM_MAP_SYMMETRIC_RKEY`
  - Indicates that the memory is allocated in environments with symmetric memory
    - There is a potential benefit from deduplication
  - UCP is making the best effort to support
    - Non-supporting transport work as usual
- **New API, works with every transport**

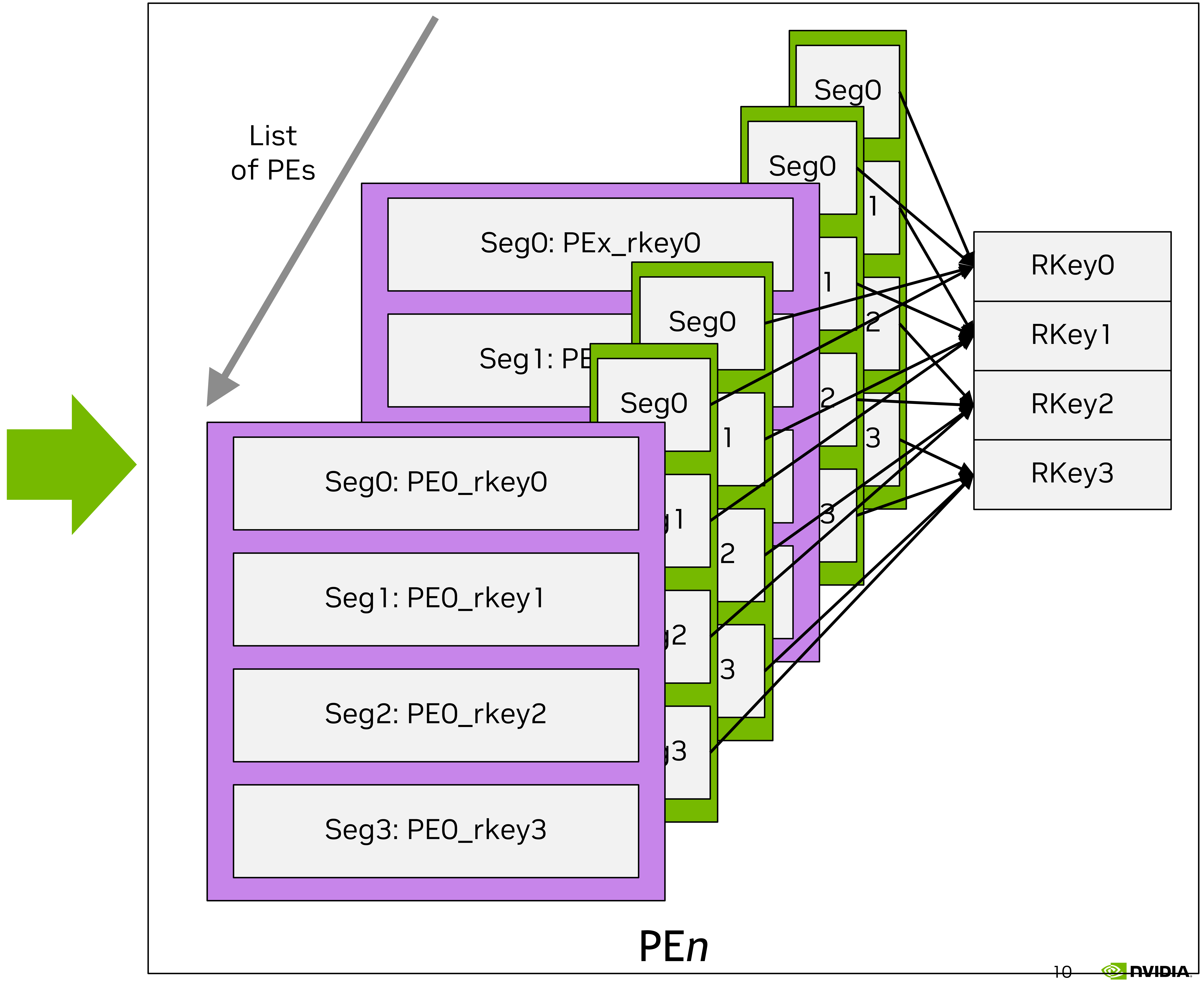
  -
    ```
    ucs_status_t ucp_rkey_compare(ucp_worker_h worker, ucp_rkey_h rkey1, ucp_rkey_h rkey2,
                                  const ucp_rkey_compare_params_t *params, int *result);
    ```
  - "`result`" field is introduced to optimize the deduplication procedure
    - Possible values (assigned at UCP/UCT discretion):
      - "`-1`" => `rkey1` is "less" than `rkey2`
      - "`+1`" => `rkey1` is "greater" than `rkey2`
      - "` 0`" => `rkey1 == rkey2`
    - Allows applying binary search during deduplication procedure
      - => *log2*(*n*) search time.

# Allocation Symmetric remote keys

List of PEs
Seg0: PEz_rkey0
Seg0: PEy_rkey0
Seg0: PEx_rkey0
Seg0: PE2_rkey0
Seg0: PE1_rkey0
Seg0: PE0_rkey0
Seg1: PE0_rkey1
Seg2: PE0_rkey2
Seg3: PE0_rkey3
Local PE
Remote PE
PE*n*
List of PEs
Seg0
1
2
3
Seg0: PEx_rkey0
Seg1: PE
Seg0: PE0_rkey0
Seg1: PE0_rkey1
Seg2: PE0_rkey2
Seg3: PE0_rkey3
RKey0
RKey1
RKey2
RKey3
RKey4
RKey5
RKey6
PE*n*

## Allocation Symmetric remote keys (ideal case)





# Notes

UCX: UCP new API

- **Memory registration performance, wrt allocation scheme**
  - ppn 48, cx6, 30 segments: 4-7 ms per segment creation
  - ppn 48, cx6, 2 segments (default): 1-2 ms per segment creation
  - ppn 28, cx6, 30 segments: 2-3 ms per segment creation, no different with standard allocation mode
- **Deduplication (log(n))**
  - 3 nodes: 32 segments, 48 ppn
  - Intra-node, keys are stored: not deduplicated
  - Inter-node, at least two other nodes to trigger deduplication

NVIDIA®