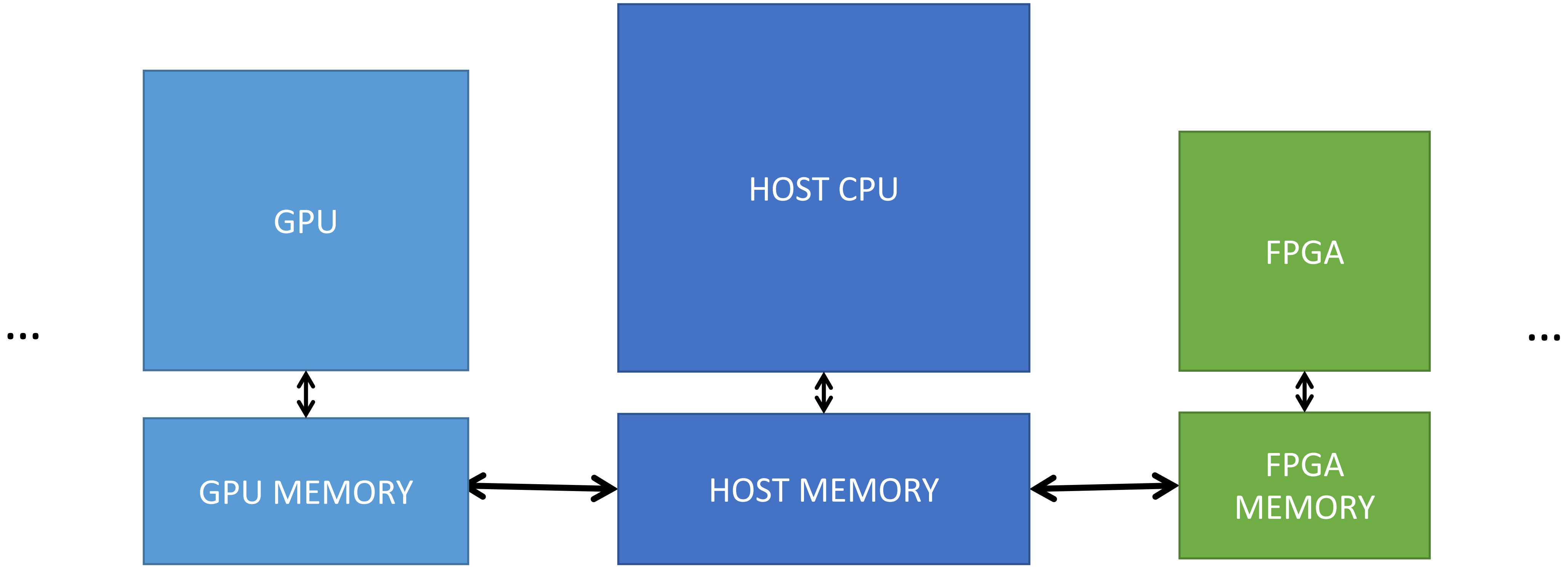
Automatic Copying of Pointer-based Data Structures for Distributed Memories

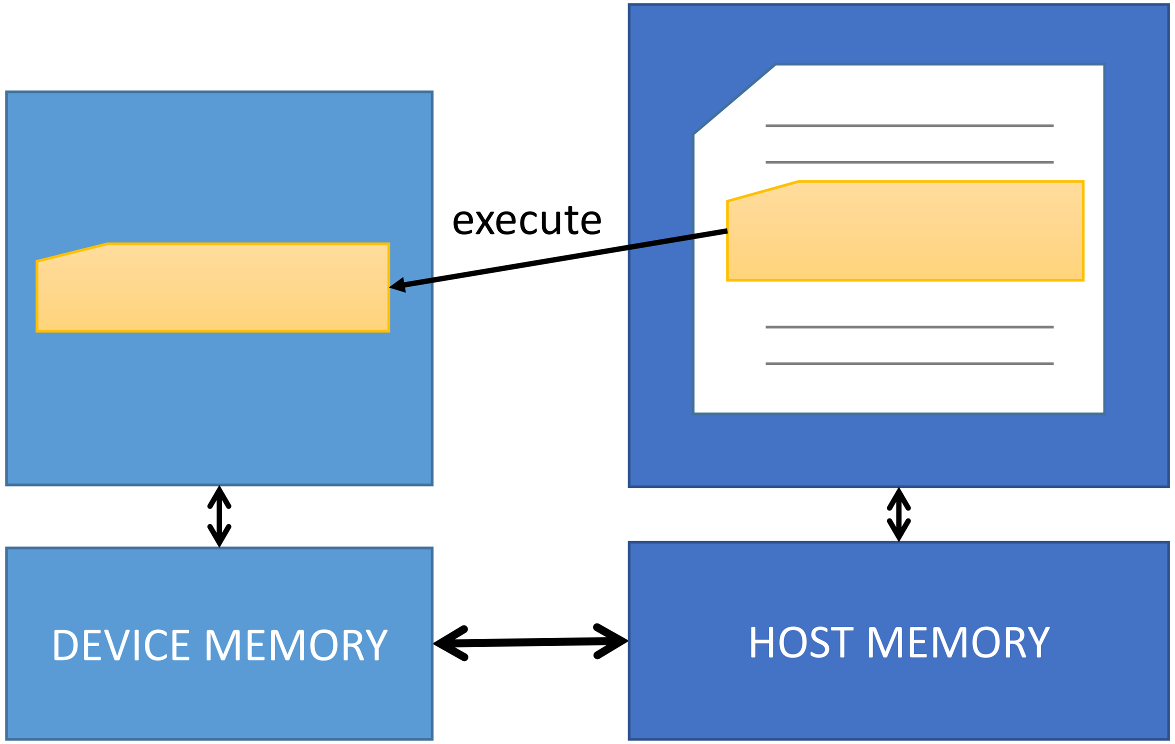
Hyojin Sung, Tong Chen, and Zehra Sura

Heterogeneous Systems



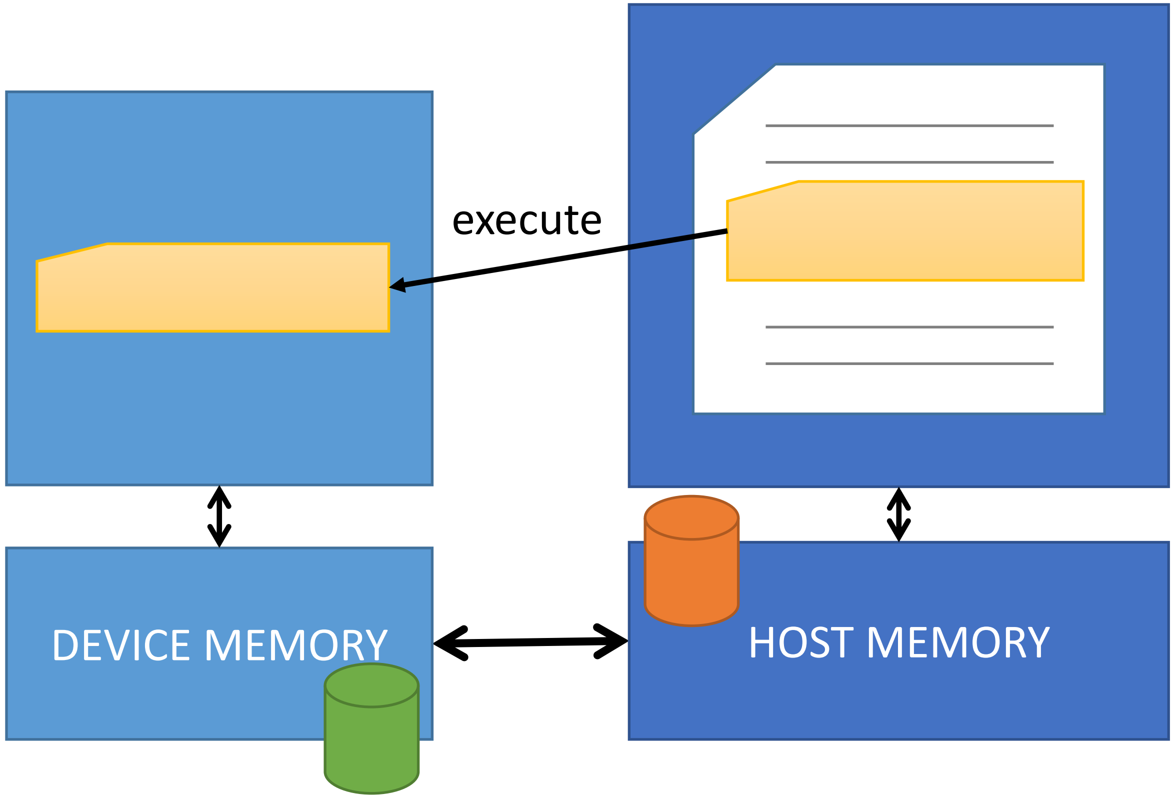
Popular for performance and power efficiency

Programming for Heterogeneous Systems



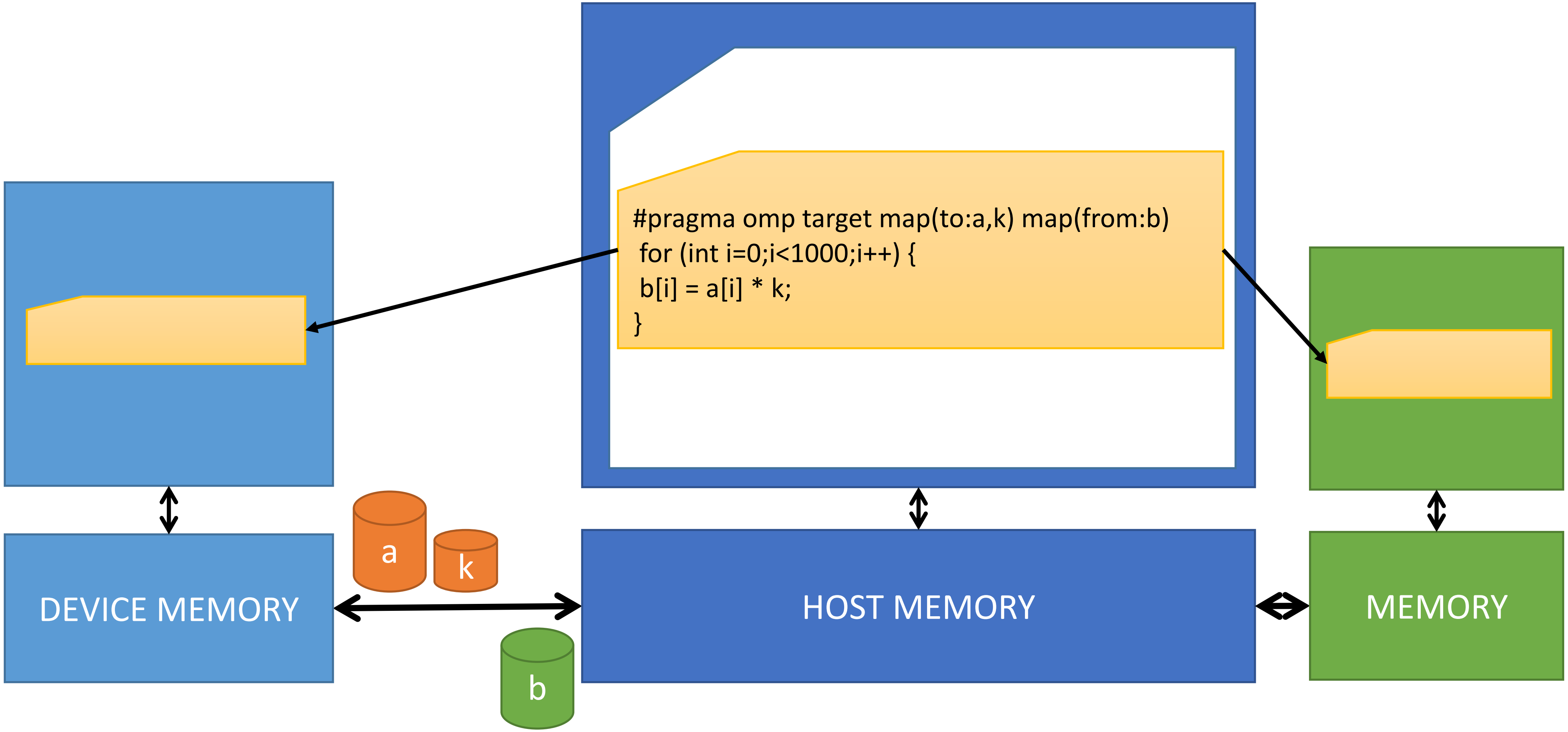
Host offloads computation to devices

Programming for Heterogeneous Systems



Challenge: Moving **data** efficiently between different memories

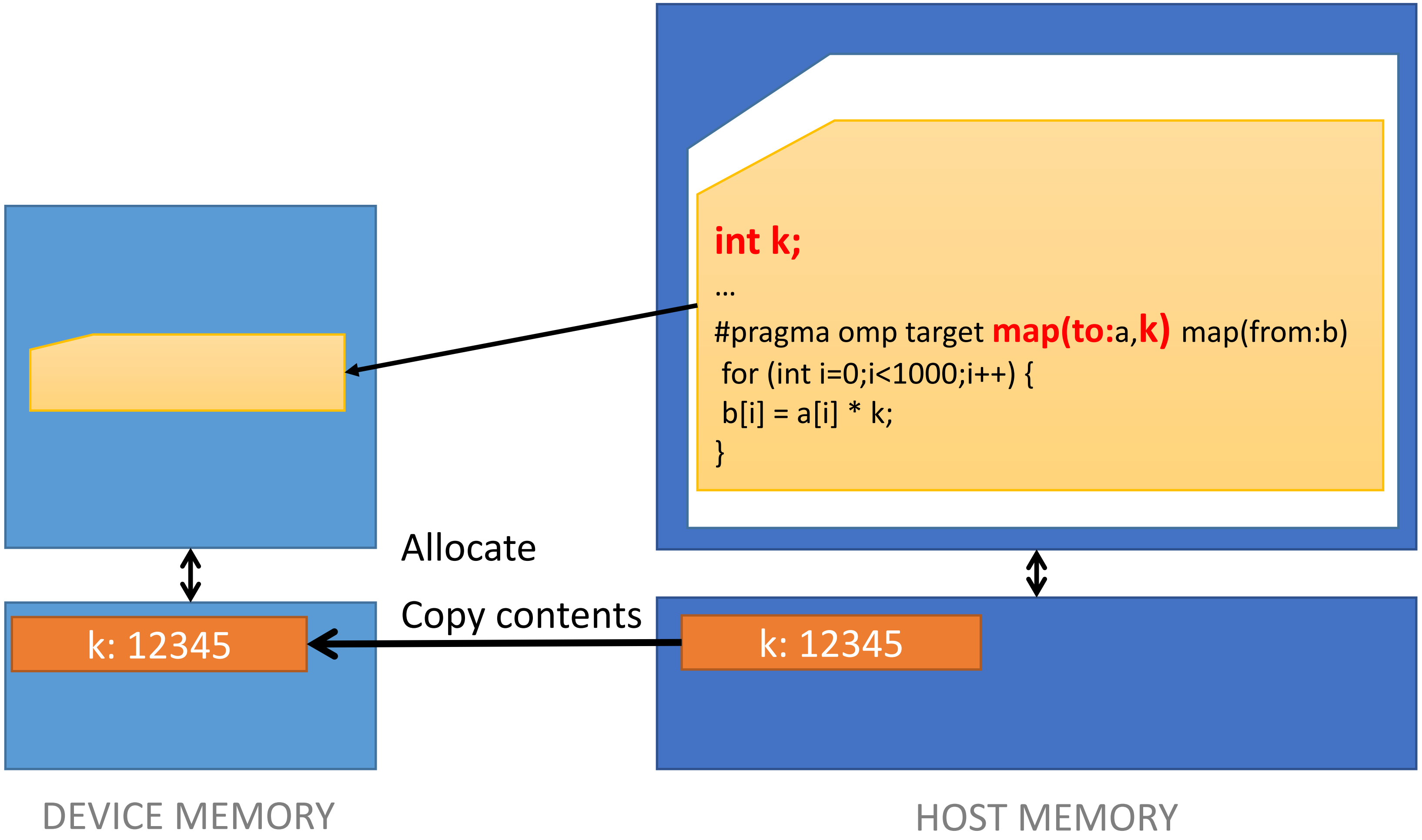
Pragma-based Programming for Heterogeneous Systems



```
#pragma omp target map(to:a,k) map(from:b)
 for (int i=0;i<1000;i++) {
 b[i] = a[i] * k;
}
```

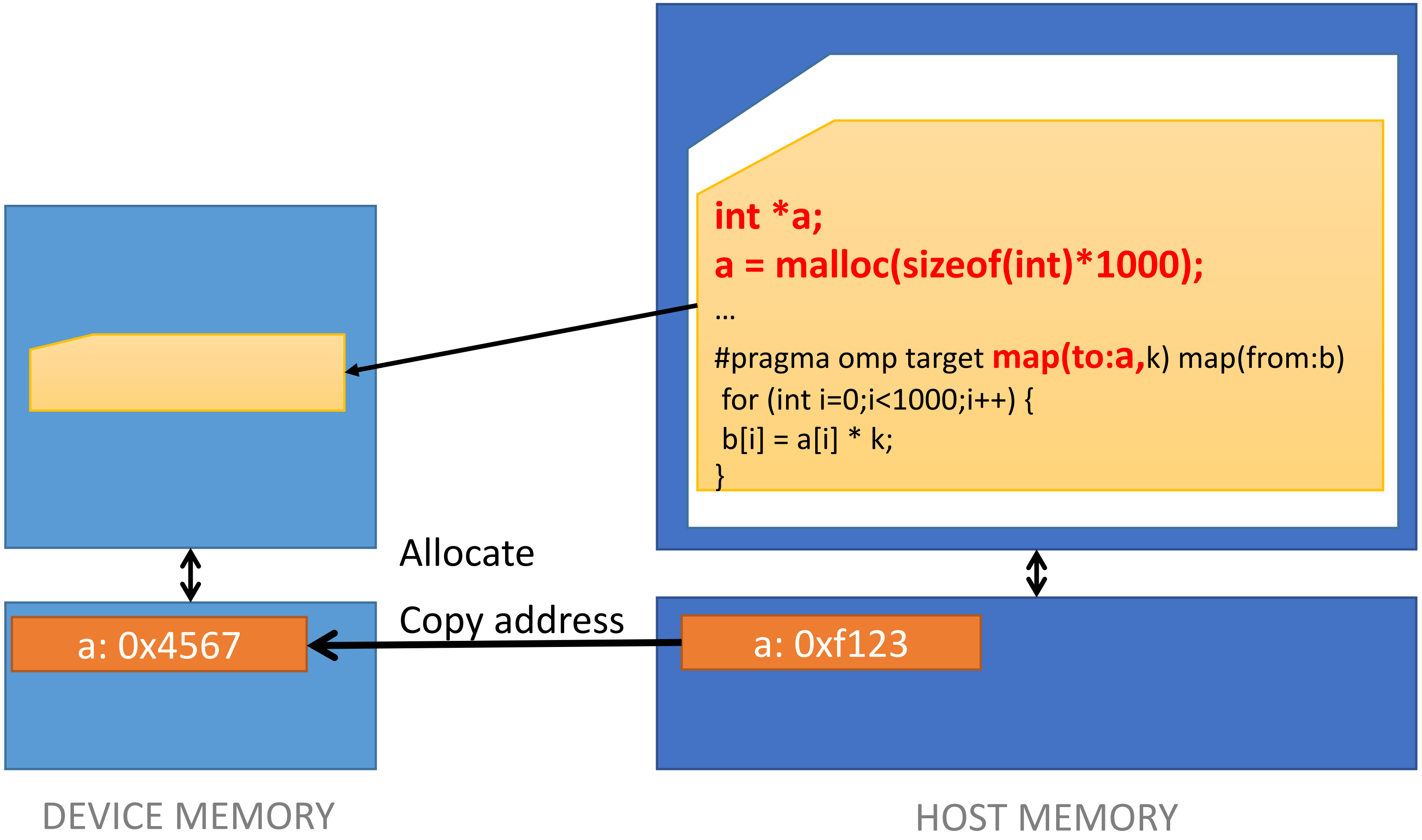
Programming productivity and code portability

(Scalar) Data Mapping in Pragma-based Programs



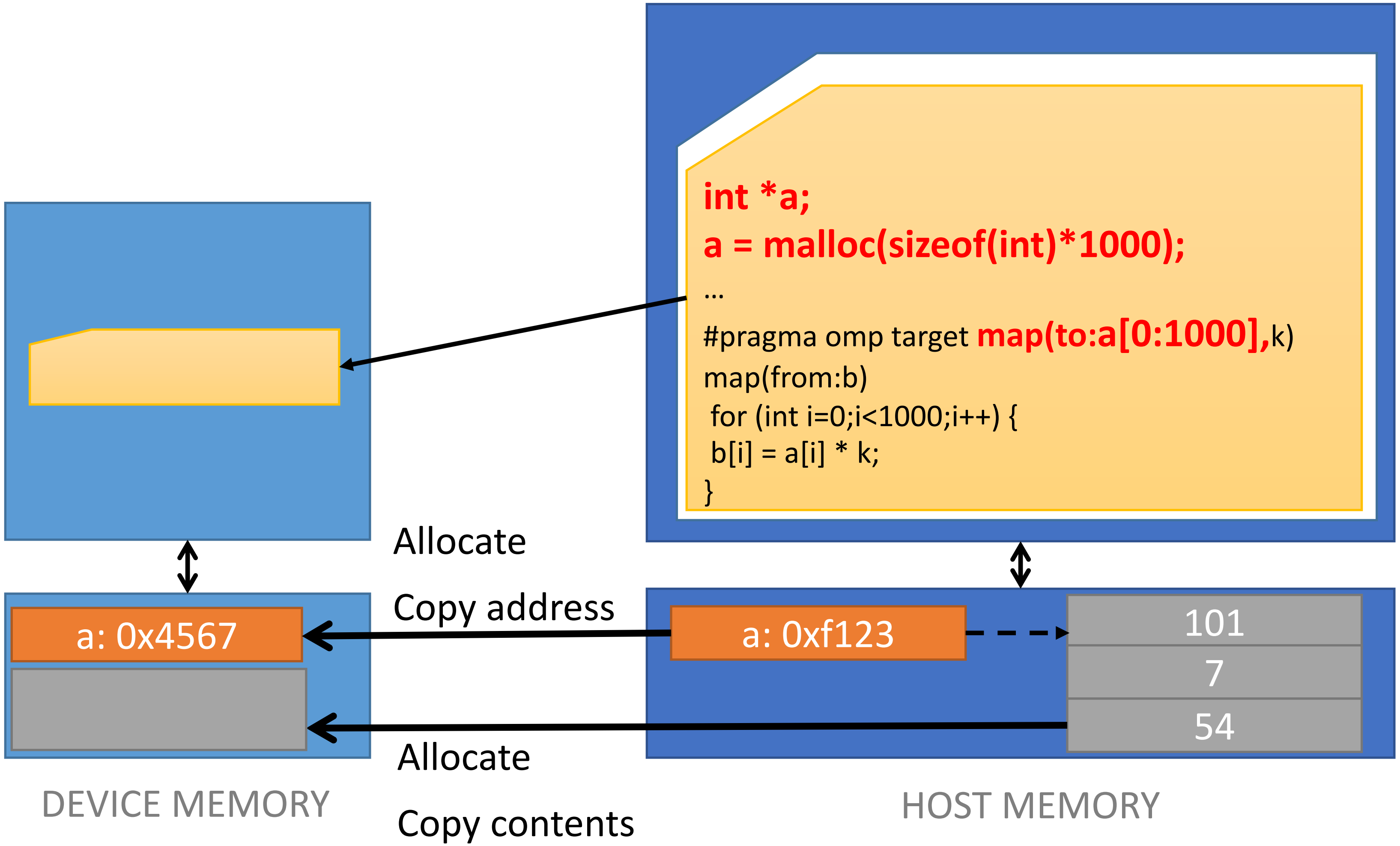
```
int k;
…
#pragma omp target map(to:a,k) map(from:b)
 for (int i=0;i<1000;i++) {
 b[i] = a[i] * k;
}
```

Pointer-based Data Mapping in Pragma-based Programs



```
int *a;
a = malloc(sizeof(int)*1000);
…
#pragma omp target map(to:a,k) map(from:b)
 for (int i=0;i<1000;i++) {
 b[i] = a[i] * k;
}
```

Pointer-based Data Mapping in Pragma-based Programs

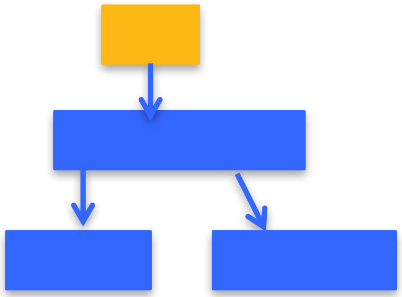


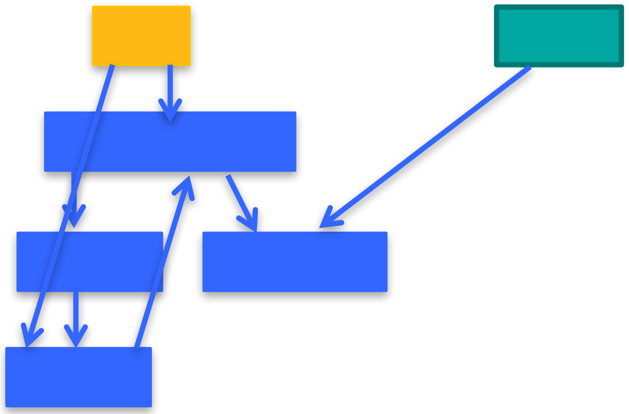
```
int *a;
a = malloc(sizeof(int)*1000);
…
#pragma omp target map(to:a[0:1000],k)
map(from:b)
 for (int i=0;i<1000;i++) {
 b[i] = a[i] * k;
}
```

“deep copy” of pointer-based data is required

Goal: Automatic copying of pointer-based data

- Currently, data transfer pragmas are limited
  - Manual ”deep copy” is tedious and error-prone
- Functionality: handles general data structures
  - Support arbitrary data structures: multiple levels of pointers, recursive data types
  - Map all the memory objects reachable from the mapped variable (top-level)





Challenges: Automatic copying of pointer-based data

- Need information about the pointers, such as size and type

- Need to maintain the mapping information at runtime
  - Finer-grained address mapping between host and device
  - Reference count on device copy of memory objects

Opportunities: Automatic copying for Fortran

- Need information about the pointers, such as size and type
  - ”allocatable” arrays and pointers used for dynamically allocated data structures in Fortran
  - **Dope vectors**, not just raw pointers, are used to represent them
  - Dope vector contains meta data:
    - Status of pointer: allocated or not
    - Address of pointed object
    - Size and shape of pointed object

No extra work from users!

- Need to maintain the mapping information at runtime
  - Finer-grained address mapping between host and device
  - Reference count on device copy of memory objects

Opportunities: Automatic copying for Fortran

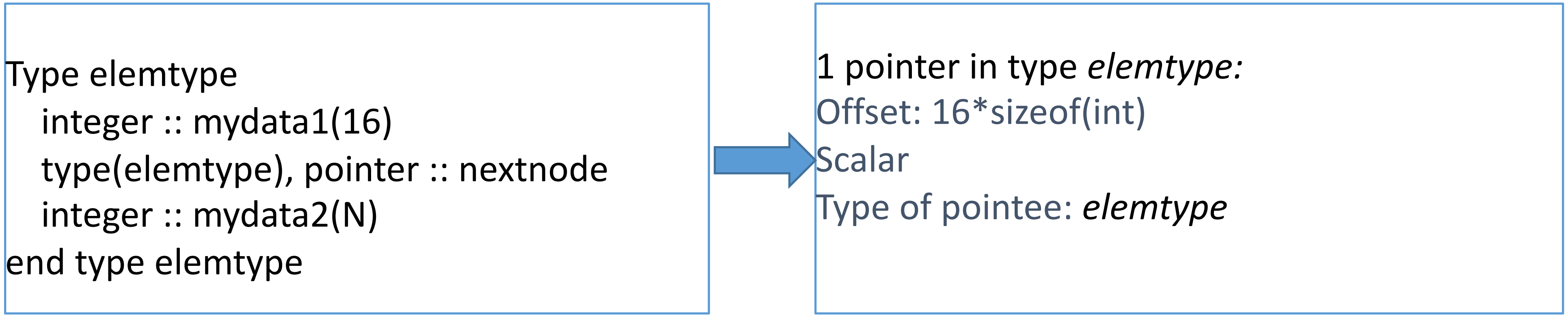
- Need information about the pointers, such as size and type
  - ”allocatable” arrays and pointers used for dynamically allocated data structures in Fortran
  - **Dope vectors**, not just raw pointers, are used to represent them
  - Dope vector contains meta data:
    - Status of pointer: allocated or not
    - Address of pointed object
    - Size and shape of pointed object

No extra work from users!

- Need to maintain the mapping information at runtime
  - Finer-grained address mapping between host and device
  - Reference count on device copy of memory objects
    - Compiler and runtime support for OpenMP device offloading (part of CORAL project)

Overview: how to handle deep copy in Fortran

- Compiler collects type information
  - For each pointer field in the user-defined type
    - Offset
    - scalar or array
    - type of the pointee



```
Type elemtype
   integer :: mydata1(16)
   type(elemtype), pointer :: nextnode
   integer :: mydata2(N)
end type elemtype
```

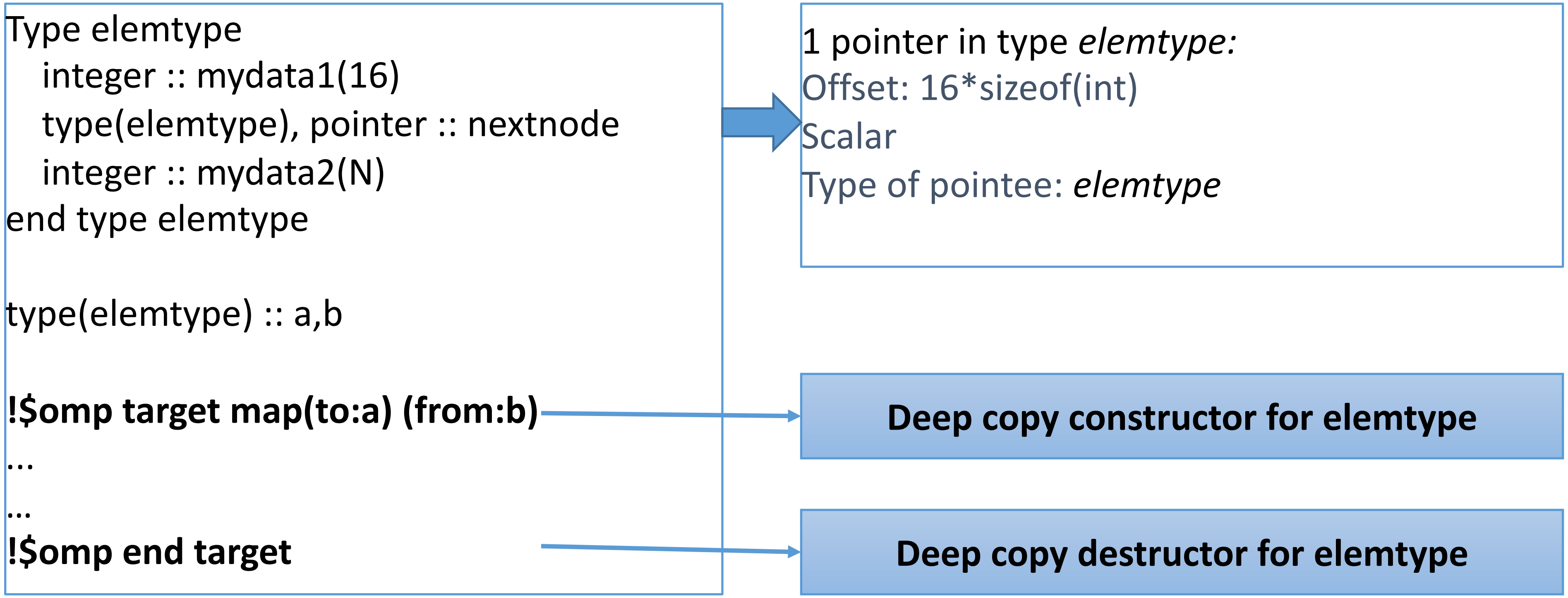
1 pointer in type *elemtype:*
Offset: 16*sizeof(int)
Scalar
Type of pointee: *elemtype*

User-defined type for linked list nodes

Compiler-generated info (from dope vector)

Overview: how to handle deep copy in Fortran

- Compiler creates deep copy constructor and destructor for each type
    - A general implementation parameterized with type info
- OpenMP runtime library calls constructor and destructor at data map boundary



```
Type elemtype
   integer :: mydata1(16)
   type(elemtype), pointer :: nextnode
   integer :: mydata2(N)
end type elemtype

type(elemtype) :: a,b

!$omp target map(to:a) (from:b)
...
...
!$omp end target
```

1 pointer in type *elemtype:*
Offset: 16*sizeof(int)
Scalar
Type of pointee: *elemtype*

Deep copy constructor for elemtype

Deep copy destructor for elemtype

Overview: how to handle deep copy in Fortran

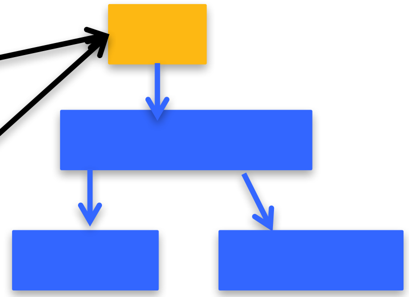
- Traverse all the reachable memory objects from the mapped pointer
  - Recursively call the constructor/destructor for each dope vector contained
- Spanning tree algorithm to ensure each memory object is handled exactly once

```
Type elemtype
   integer :: mydata1(16)
   type(elemtype), pointer :: nextnode
   integer :: mydata2(N)
end type elemtype

type(elemtype) :: a,b

!$omp target map(to:a) (from:b)
...
...
!$omp end target
```

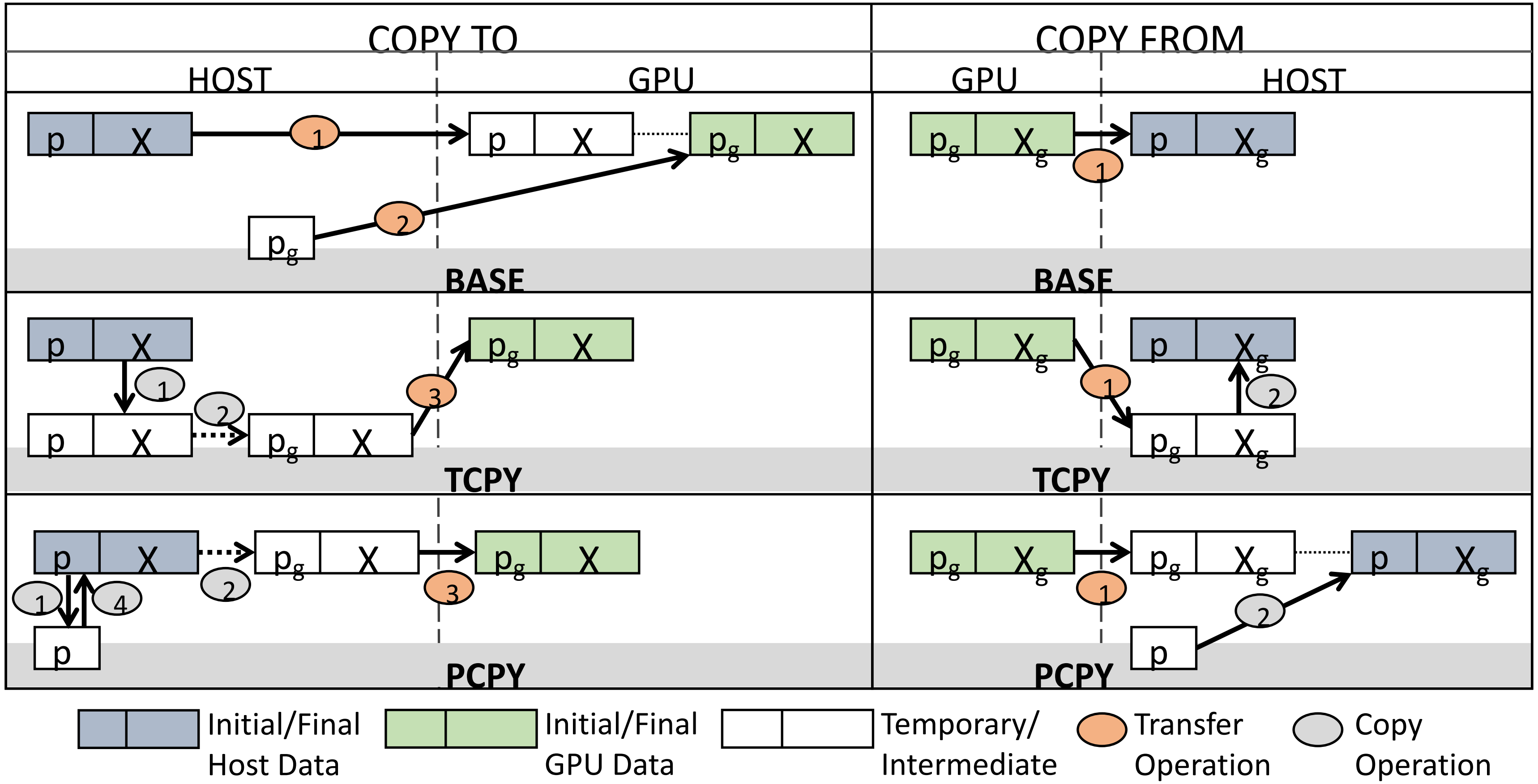
1 pointer in type *elemtype:*
Offset: 16*sizeof(int)
Scalar
Type of pointee: *elemtype*

Deep copy constructor for elemtype

Deep copy destructor for elemtype



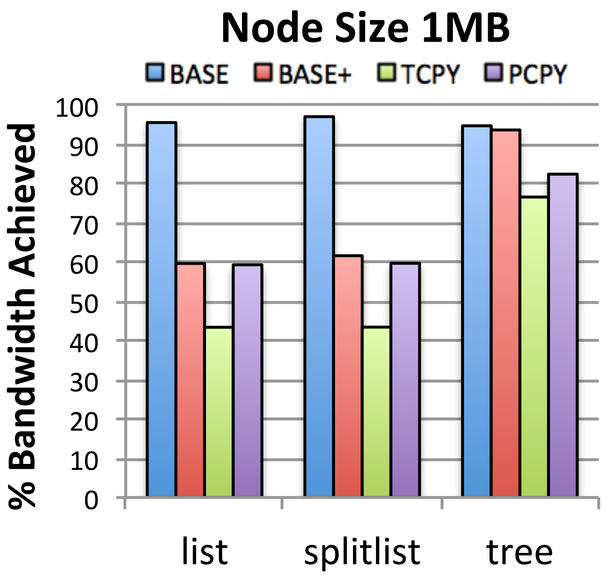
Perform Node Copy



- BASE
  - Copy pointer and data separately: different addresses on host and device
- Optimizations
  - Reduce # of transfers
  - TCPY, PCPY: temporary copies on CPU

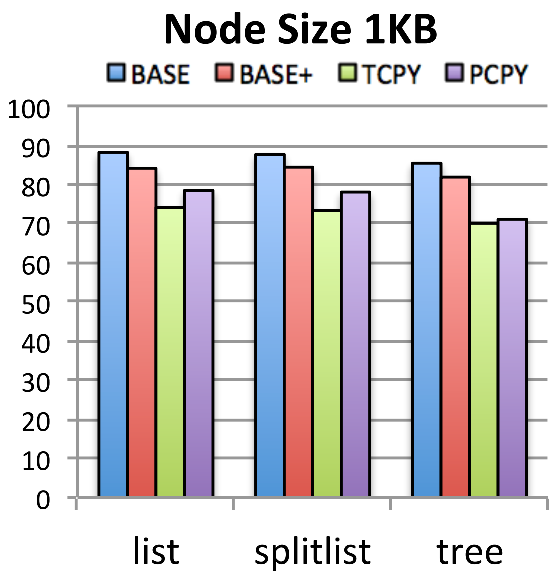
Experimental Results

- Kernels that recursively access linked lists and tree
- Comparison to CUDA version with data transfers only
  - No OpenMP overhead
  - No management overhead for mapped data



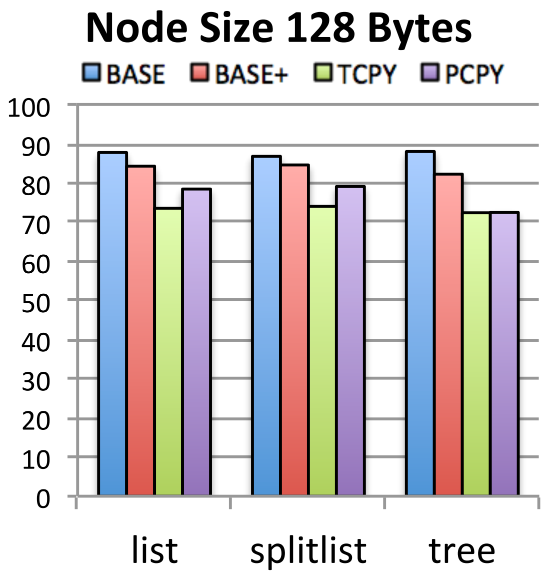
(a)

List



(b)

Split list



(c)

Tree

Conclusions and Future Work

- Automatic copying of arbitrary data structures between CPU and GPU
  - Take advantage of language feature: dope vector in Fortran
  - No extra burden on users

- We will further improve the functionality and reduce the overhead
  - Asynchronous data transfer
  - Compiler analysis/user pragma to reduce the amount of data to be transferred
  - Increased parallelism with deep copying
  - Mutable data structure

- Expand the work to languages other than Fortran
  - introduce smart pointer abstraction in C/C++ systems
  - Library framework or template classes for metadata representation
  - Transparent enablement with errors/warnings when automatic system cannot handle

Thank you!