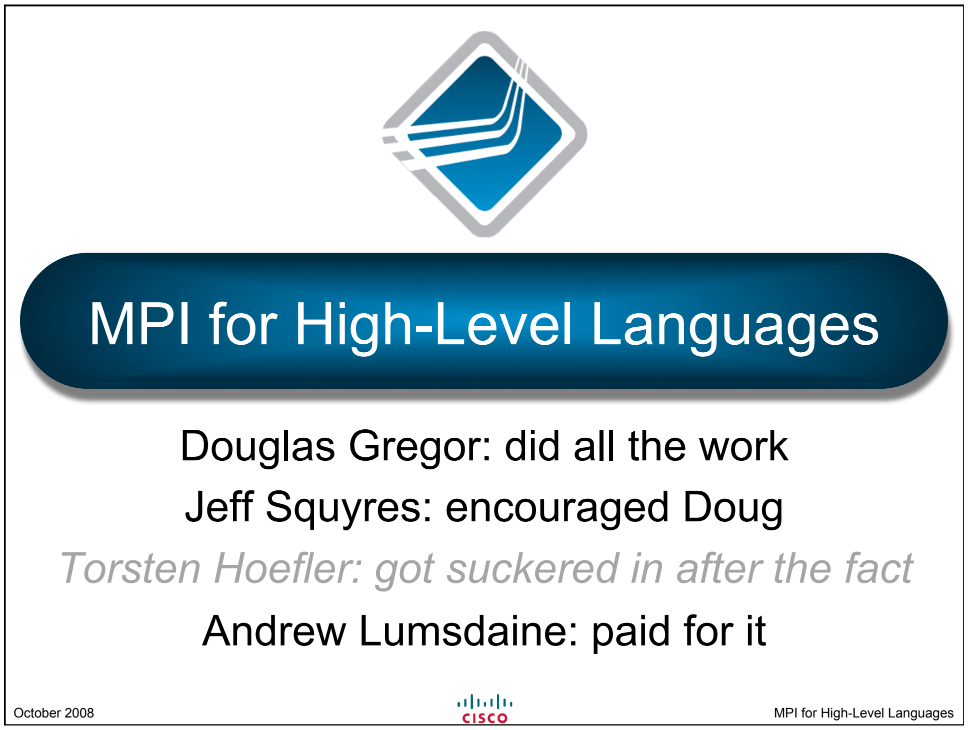


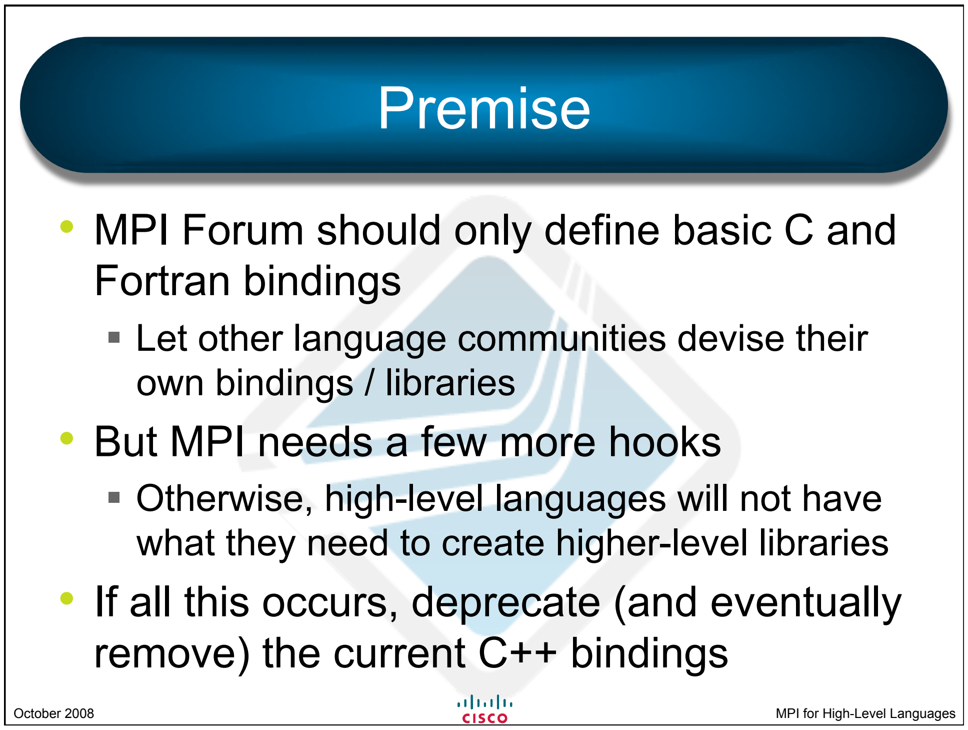
# MPI for High-Level Languages

Douglas Gregor: did all the work

Jeff Squyres: encouraged Doug

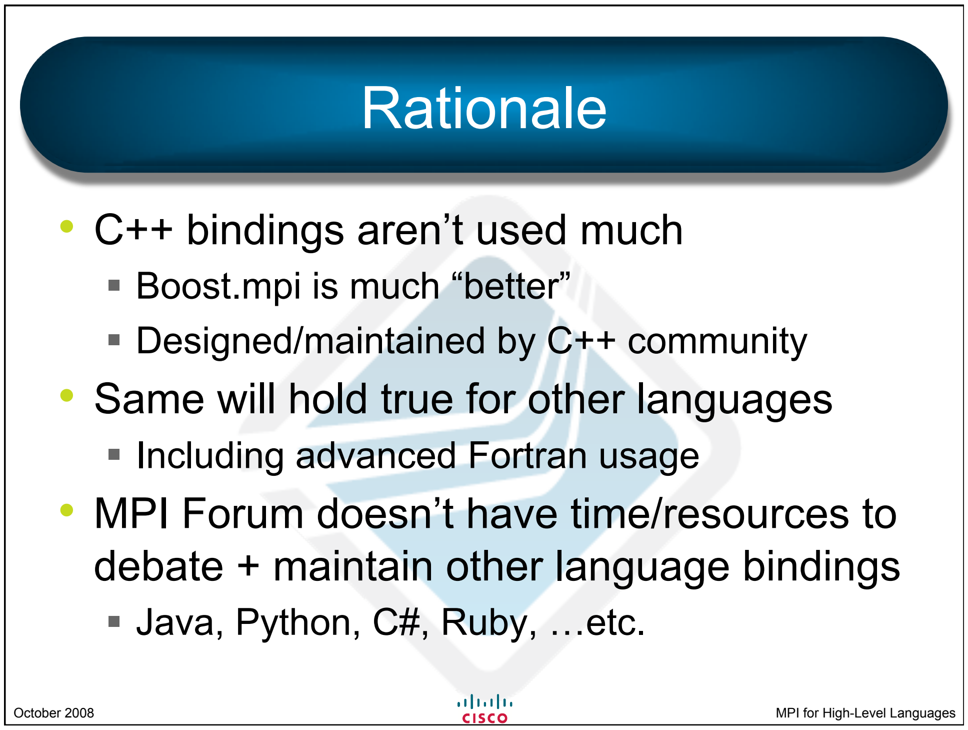
*Torsten Hoefler: got suckered in after the fact*

Andrew Lumsdaine: paid for it



## Premise

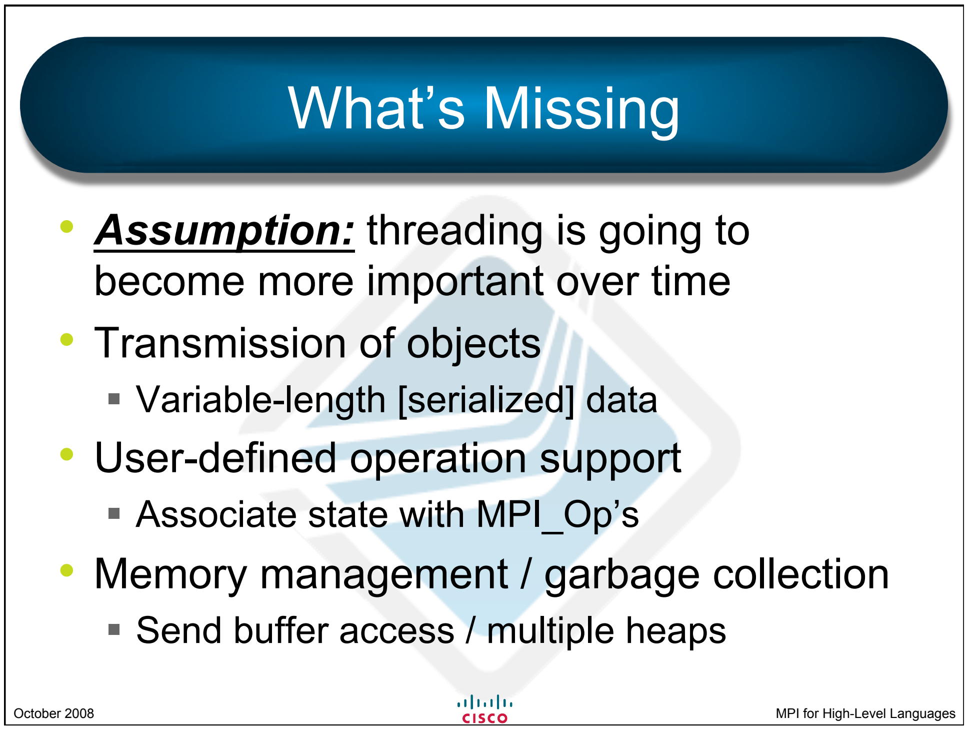
- MPI Forum should only define basic C and Fortran bindings
  - Let other language communities devise their own bindings / libraries
- But MPI needs a few more hooks
  - Otherwise, high-level languages will not have what they need to create higher-level libraries
- If all this occurs, deprecate (and eventually remove) the current C++ bindings



## Rationale

- C++ bindings aren't used much
  - Boost.mpi is much "better"
  - Designed/maintained by C++ community
- Same will hold true for other languages
  - Including advanced Fortran usage
- MPI Forum doesn't have time/resources to debate + maintain other language bindings
  - Java, Python, C#, Ruby, …etc.

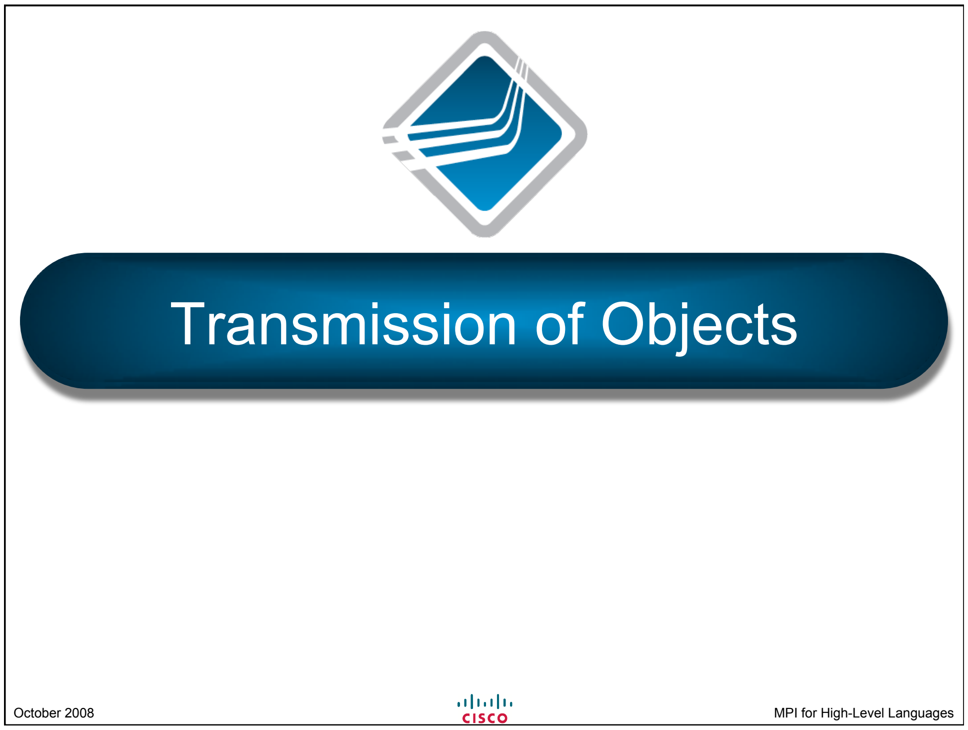
October 2008 MPI for High-Level Languages



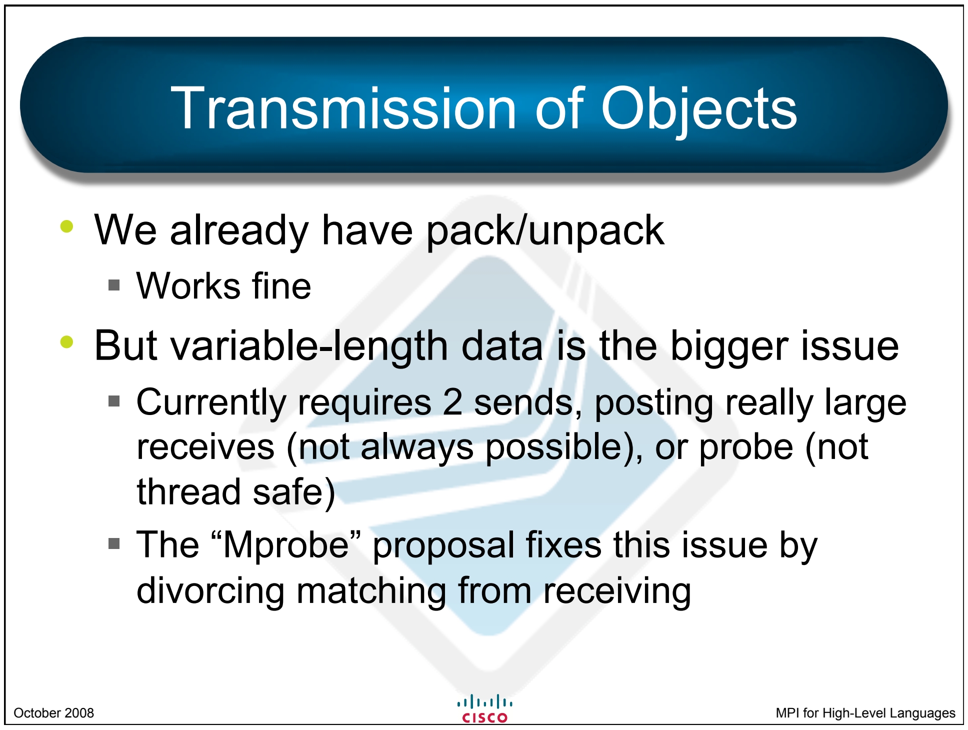
## What's Missing

- ***Assumption:*** threading is going to become more important over time
- Transmission of objects
  - Variable-length [serialized] data
- User-defined operation support
  - Associate state with MPI_Op's
- Memory management / garbage collection
  - Send buffer access / multiple heaps

October 2008 MPI for High-Level Languages



## Transmission of Objects



## Transmission of Objects

- We already have pack/unpack
  - Works fine
- But variable-length data is the bigger issue
  - Currently requires 2 sends, posting really large receives (not always possible), or probe (not thread safe)
  - The “Mprobe” proposal fixes this issue by divorcing matching from receiving

## Variable Length Pt2Pt Data

```
// do the match
MPI_Message msg;
MPI_Mprobe(…, &msg, &status)
// extract the size
MPI_Get_count(&status, MPI_BYTE, &count)
// malloc for that size
char *buffer = malloc(count);
// do the receive
MPI_Mrecv(buffer, count, MPI_BYTE, &msg, …);
// deserialize the message
```

## Variable Length Coll. Data

- Currently no provisions for this
  - Reductions are problematic
  - Example: MPI_ALLREDUCE on strings to catenate into a big string
- Problems:
  - Collectives take a fixed count
    - Can bcast the count first, but may not know final size until reduction is performed
  - MPI_User_function takes a fixed count and cannot realloc inoutvec

## Variable Length 1Sided Data

- No provisions or proposals at this time
- Follow the one-sided committee and see what happens

## Variable Length Solution: Blobs

```
struct MPI_Blob {
  MPI_Aint length; // or some new type
  MPI_Aint address;
};
```

- Can use blobs for pt2pt, collectives
  - Per previous, wait and see for one-sided

## Blobs and Reductions

```
char *string = …;
MPI_Blob out, in;

// setup “out” blob
out.length = strlen(string);
out.address = string;

// Do the reduce, get result in a new blob
MPI_Allreduce(&out, &in, 1, MPI_BLOB,
 concatOp, MPI_COMM_WORLD);
```

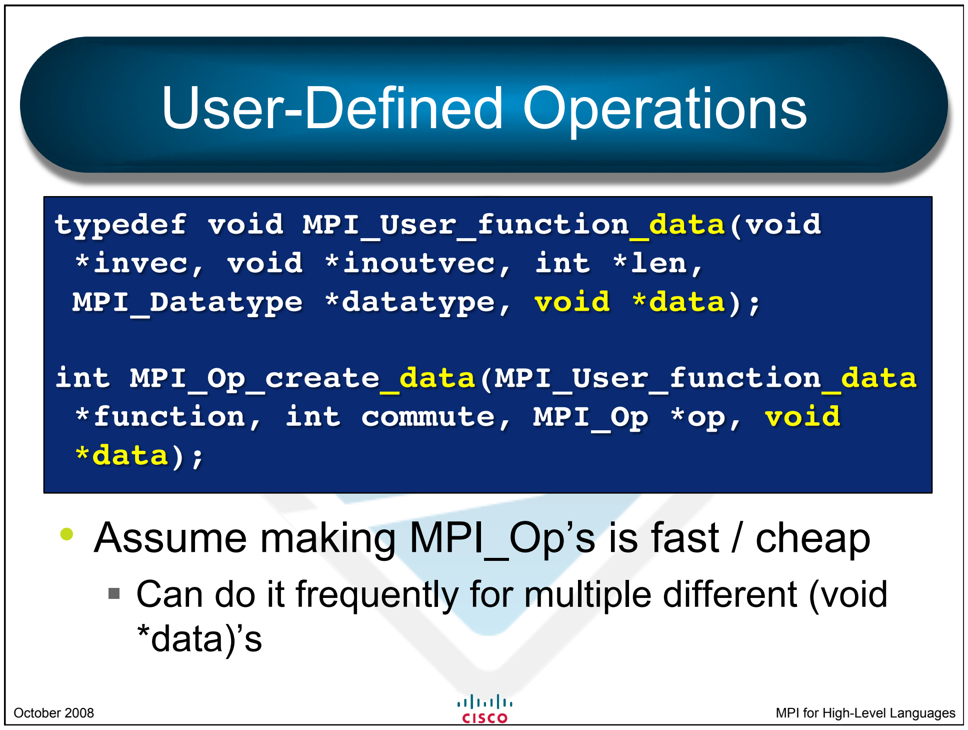
## Blob Restrictions

- Cannot be used with MPI_TYPE_CREATE_STRUCT
- Cannot be used with MPI_IN_PLACE (?)
- Where does the memory come from?
  - Implementation has to alloc blob memory that is given back to the user (“in” blob in previous example)
  - Require MPI_ALLOC/FREE_MEM?

## User-Defined Operations

## User-Defined Operations

- Problematic for multi-threaded
  - No way to associate state with an MPI_Op (e.g, an object)
- Do not want to change all reduction collective function signatures
- Simple solution
  - Allow associating a (void*) to new MPI_Op creation function



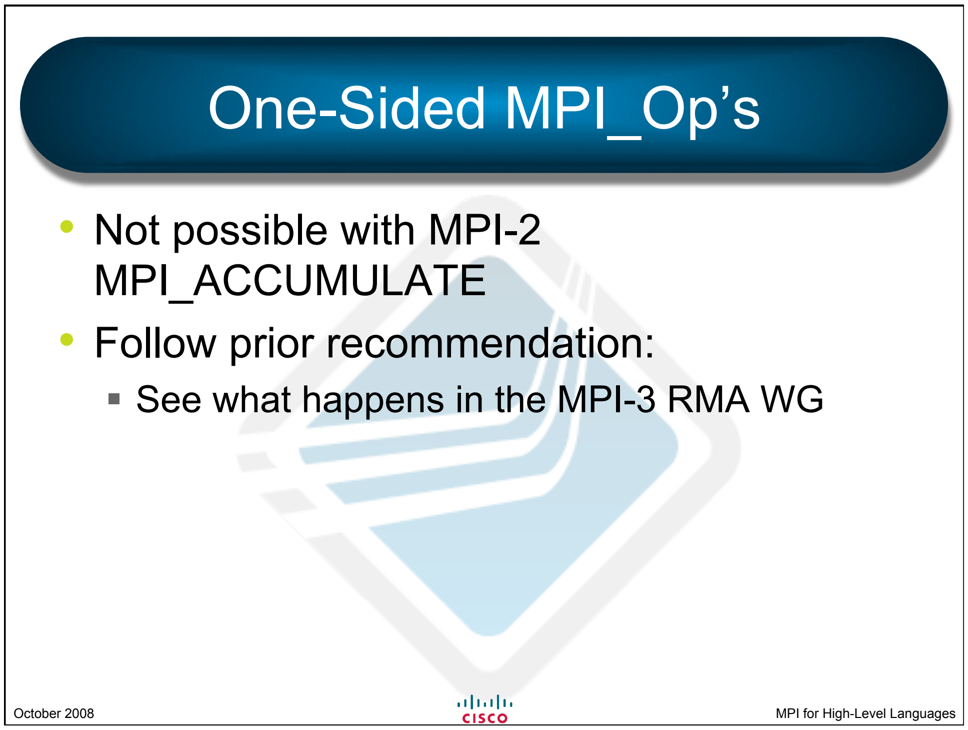
## User-Defined Operations

```
typedef void MPI_User_function_data(void
 *invec, void *inoutvec, int *len,
 MPI_Datatype *datatype, void *data);

int MPI_Op_create_data(MPI_User_function_data
 *function, int commute, MPI_Op *op, void
 *data);
```

- Assume making MPI_Op’s is fast / cheap
  - Can do it frequently for multiple different (void *data)’s

October 2008 MPI for High-Level Languages



## One-Sided MPI_Op’s

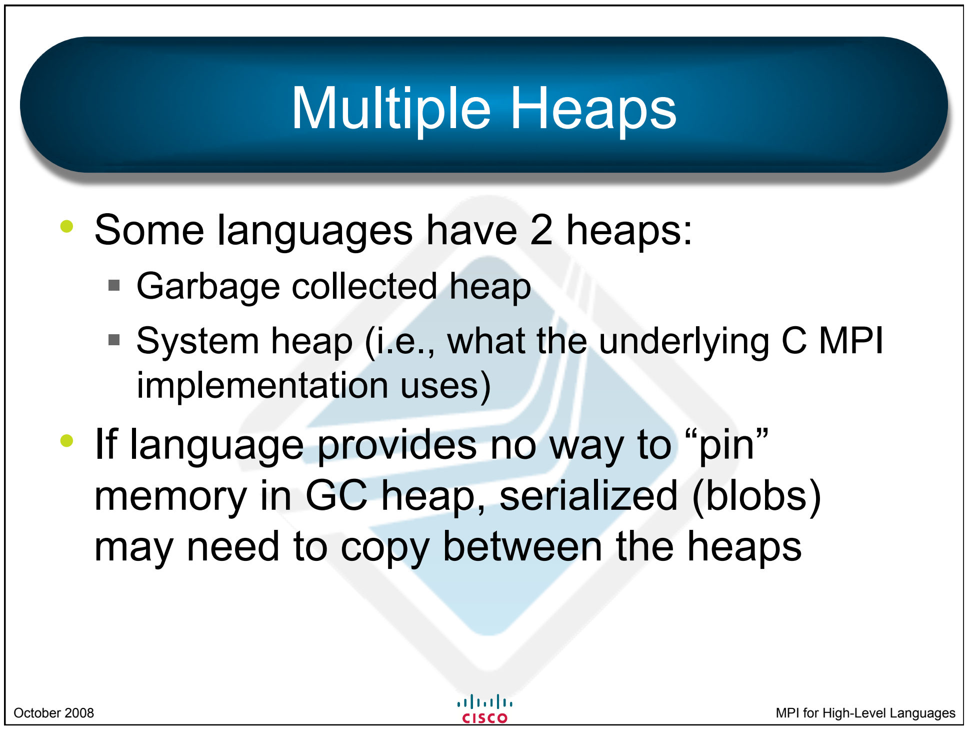
- Not possible with MPI-2 MPI_ACCUMULATE
- Follow prior recommendation:
  - See what happens in the MPI-3 RMA WG

October 2008 MPI for High-Level Languages

## Memory Management / Garbage Collection

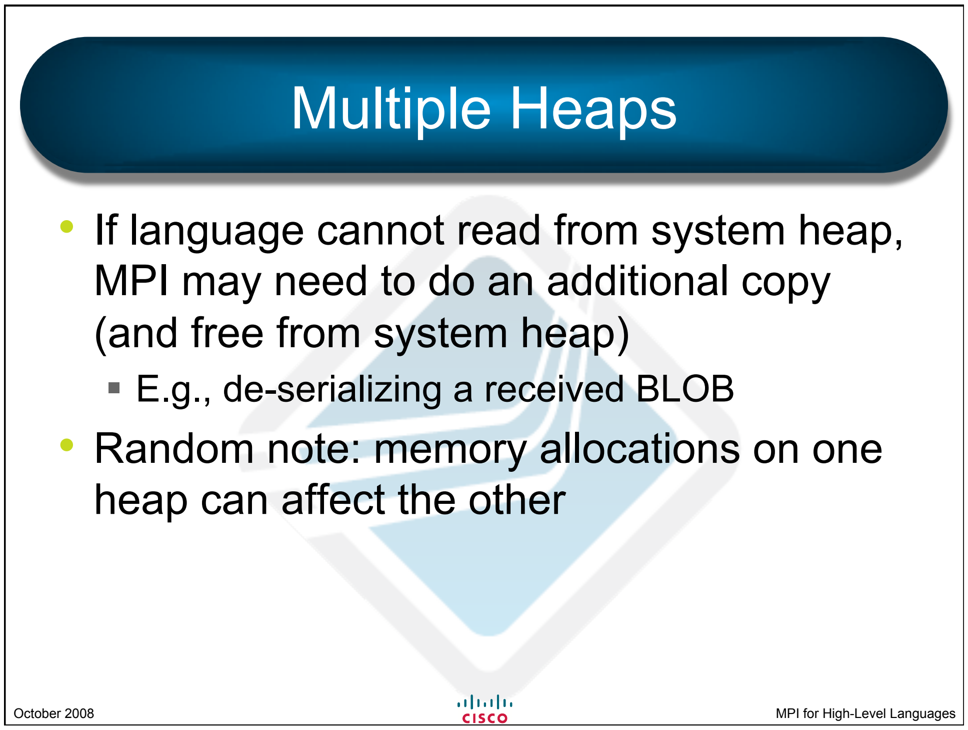
## Multi-Send Buffer Access

- Garbage collectors may scan memory that is being used in non-blocking sends
- Microsoft's proposal addresses this issue
  - Trac ticket #45
  - Looks like this ticket will be included in 2.2



## Multiple Heaps

- Some languages have 2 heaps:
  - Garbage collected heap
  - System heap (i.e., what the underlying C MPI implementation uses)
- If language provides no way to “pin” memory in GC heap, serialized (blobs) may need to copy between the heaps



## Multiple Heaps

- If language cannot read from system heap, MPI may need to do an additional copy (and free from system heap)
  - E.g., de-serializing a received BLOB
- Random note: memory allocations on one heap can affect the other

## So… What’s the Problem?

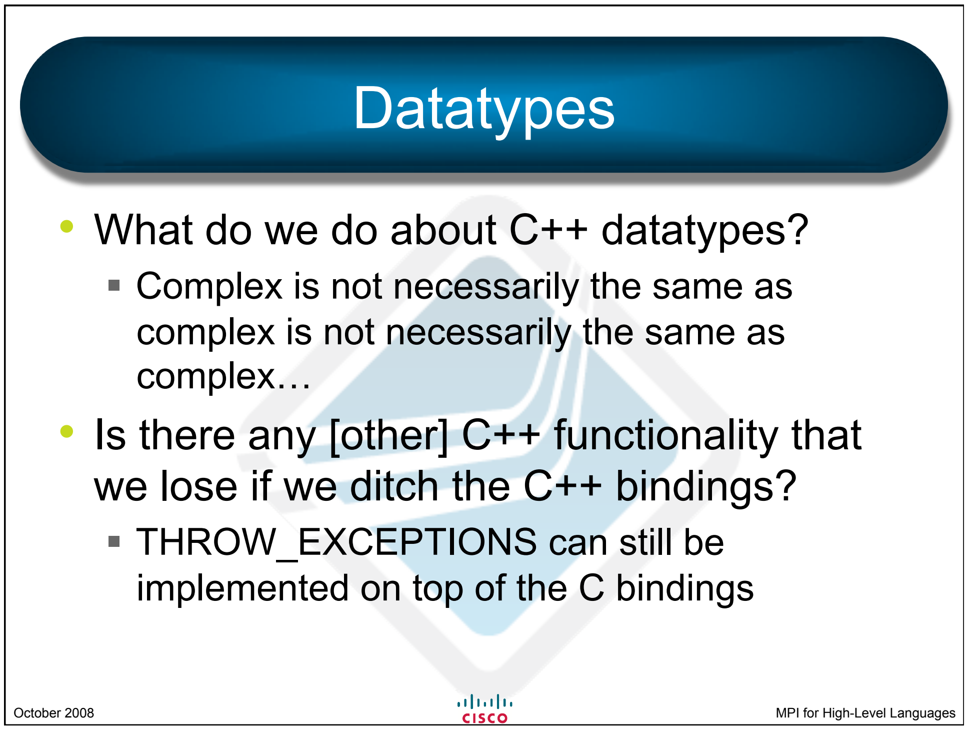
- Some MPI extensions may be needed to help eliminate the need for extraneous copies of serialized data between system and GC heaps.
- May also be useful with some networks that require registered memory…?
- Don’t know exactly what is needed yet
  - Still working on it…
  - Feedback would be appreciated

## Moving Forward

## Moving Forward

- Some points in this proposal are good
  - Others need [a lot] more discussion
  - Need language specialists for these issues!
- Probably need a group where to discuss these issues
  - New WG?
  - Rename Fortran WG → Language WG?
  - Still have some Fortran-specific issues to discuss

## Additional Issues



## Datatypes

- What do we do about C++ datatypes?
  - Complex is not necessarily the same as complex is not necessarily the same as complex…
- Is there any [other] C++ functionality that we lose if we ditch the C++ bindings?
  - THROW_EXCEPTIONS can still be implemented on top of the C bindings