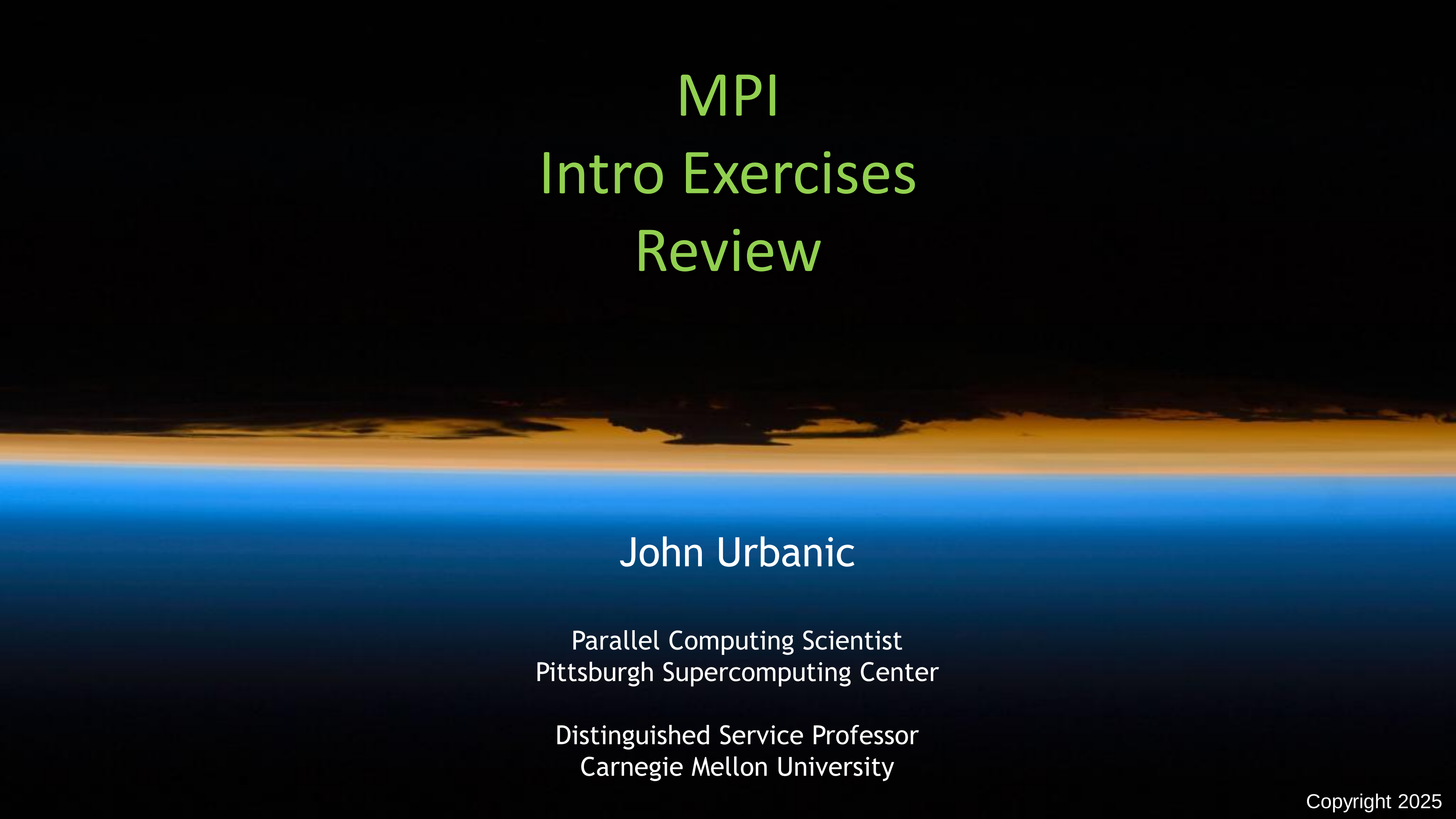

MPI
Intro Exercises
Review

John Urbanic

Parallel Computing Scientist
Pittsburgh Supercomputing Center

Distinguished Service Professor
Carnegie Mellon University

Exercise 1
One C Solution

```
#include <stdio.h>
#include "mpi.h"

main(int argc, char** argv){

  int my_PE_num, number_to_send, message_received;
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_PE_num);

  number_to_send = my_PE_num;

  if (my_PE_num==7)
      MPI_Send( &number_to_send, 1, MPI_INT, 0, 10, MPI_COMM_WORLD);
  else
      MPI_Send( &number_to_send, 1, MPI_INT, my_PE_num+1, 10, MPI_COMM_WORLD);

  MPI_Recv( &message_received, 1, MPI_INT, MPI_ANY_SOURCE, 10, MPI_COMM_WORLD, &status);

  printf("PE %d received %d.\n", my_PE_num, message_received);

  MPI_Finalize();

}
```

Exercise 1
A Possible Fortran Solution

```
program shifter
implicit none

include 'mpif.h'

integer my_pe_num, errcode, numbertosend, message_received
integer status(MPI_STATUS_SIZE)

call MPI_INIT(errcode)

call MPI_COMM_RANK(MPI_COMM_WORLD, my_pe_num, errcode)

numbertosend = my_pe_num

if (my_pe_num.EQ.7) then
   call MPI_Send(numbertosend, 1, MPI_INTEGER, 0, 10, MPI_COMM_WORLD, errcode)
else
   call MPI_Send(numbertosend, 1, MPI_INTEGER, my_pe_num+1, 10, MPI_COMM_WORLD, errcode)
endif

call  MPI_Recv(message_received, 1, MPI_INTEGER, MPI_ANY_SOURCE, 10, MPI_COMM_WORLD, status, errcode)

print *,'PE', my_pe_num, ' received ', message_received, '.'

call MPI_FINALIZE(errcode)
end
```

Exercise 1
Output

```
c557-603$ mpicc solution1.c
c557-603$ mpirun -n 8 a.out
PE 2 received 1.
PE 0 received 7.
PE 4 received 3.
PE 3 received 2.
PE 5 received 4.
PE 1 received 0.
PE 7 received 6.
PE 6 received 5.
```

MPI Anti-patterns

There are some common MPI beginner *anti-patterns* or *code smells* that you should beware of.

First is failing to recognize when all of the PEs are doing the same thing, just working with different data. It you find yourself writing code that looks like this:

```
if my_pe == 1:
    do something
if my_pe == 2:
    do something similar
if my_pe == 3:
    do something similar
if my_pe == 4:
    do something similar
if my_pe == 5:
    do something similar
```

You are probably falling into this trap. I won't ask for a show of hands, but I know many of you tried that here.

MPI Reality Check

Remember that we are writing *scalable* code here. Ask yourself if this would work with 10,000 PEs.

```
if my_pe == 1:
    do something
if my_pe == 2:
    do something similar
if my_pe == 3:
    do something similar
if my_pe == 4:
    do something similar
if my_pe == 5:
    do something similar
if my_pe == 6:
    do something
if my_pe == 7:
    do something similar
if my_pe == 8:
    do something similar
if my_pe == 9:
    do something similar
if my_pe == 10:
    do something similar
if my_pe == 11:
    do something
if my_pe == 12:
    do something similar
if my_pe == 13:
    do something similar
if my_pe == 14:
    do something similar
if my_pe == 15:
    do something similar
if my_pe == 16:
.
.
.
.
.
.
.
.
```

Obviously not. Nor would it work with a runtime variable number of PEs, which should always be your goal (like our "finding Pi" example).

"Not suspicious" PE specific logic

There are a few exceptions to this. The manager/worker paradigm is very common, and recognizable. And you may find you need some special logic for the "boundaries". We've already seen both of these cases:

Worker / Manager
(from Pi code)

```
if( my_pe_num == 0 ){
    printf("How many intervals? ");
    scanf("%d", &n);
}
```

Boundary Code

```
if (my_PE_num==7)
      MPI_Send( &number_to_send, 1, MPI_INT, 0, 10, MPI_COMM_WORLD);
else
      MPI_Send( &number_to_send, 1, MPI_INT, my_PE_num+1, 10, MPI_COMM_WORLD);
```

Note that these also behave well when answering the question "what would this look like on 10,000 PEs?"

Same is true of variables

Apply the same thought process to variables. If you find yourself creating multiple definitions of a variable, you are probably not thinking parallel. Asking yourself if this will scale to 10,000 PEs will also keep you on track.

```
dt1 = 0

dt2 = 0

dt3 = 0

dt4 = 0

dt5 = 0

dt6 = 0

dt7 = 0

dt8 = 0
.
.
.
.
.
.
.
.
```

Exercise 1
Technically not perfect.

```
#include <stdio.h>
#include "mpi.h"

main(int argc, char** argv){

  int my_PE_num, number_to_send, message_received;
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_PE_num);

  number_to_send = my_PE_num;

  if (my_PE_num==7)
      MPI_Ssend( &number_to_send, 1, MPI_INT, 0, 10, MPI_COMM_WORLD);
  else
      MPI_Ssend( &number_to_send, 1, MPI_INT, my_PE_num+1, 10, MPI_COMM_WORLD);

  MPI_Recv( &message_received, 1, MPI_INT, MPI_ANY_SOURCE, 10, MPI_COMM_WORLD, &status);

  printf("PE %d received %d.\n", my_PE_num, message_received);

  MPI_Finalize();

}
```

Deadlock!

Exercise 1
For the pedants…

```
#include <stdio.h>
#include "mpi.h"

main(int argc, char** argv){

  int my_PE_num, number_to_send, message_received;
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_PE_num);

  number_to_send = my_PE_num;

  if (my_PE_num==7){
    MPI_Recv( &message_received, 1, MPI_INT, MPI_ANY_SOURCE, 10, MPI_COMM_WORLD, &status);
    MPI_Ssend( &number_to_send, 1, MPI_INT, 0, 10, MPI_COMM_WORLD);
  }
  else{
    MPI_Ssend( &number_to_send, 1, MPI_INT, my_PE_num+1, 10, MPI_COMM_WORLD);
    MPI_Recv( &message_received, 1, MPI_INT, MPI_ANY_SOURCE, 10, MPI_COMM_WORLD, &status);
  }

  printf("PE %d received %d.\n", my_PE_num, message_received);

  MPI_Finalize();

}
```

Breaks the Deadlock!

Exercise 2
Impossible Solution

- There is no possible solution.
- You can not accomplish this task with the commands you were given.
- It is simply impossible to be sure there isn’t a node somewhere “out there” that hasn’t yet responded.
- It is possible to create many “solutions” that will work *almost* all of the time. Particularly on a tightly coupled machine like Stampede.
- What if Bridges was nodes spread around the solar system. Would your answer still work?
- *It is generally not hard to write MPI codes that will always work. I gave you a really tricky problem to keep you humble, and not even all of our most basic set of commands to use.*

Exercise 2
Almost Solution

```
#include <stdio.h>
#include "mpi.h"

main(int argc, char** argv){

  int my_PE_num, numberofnodes, data;
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_PE_num);

  if (my_PE_num==0)
    for (numberofnodes=1;numberofnodes<512;numberofnodes++)
      if(MPI_Send( &data, 1, MPI_INT, numberofnodes, 10, MPI_COMM_WORLD) != MPI_SUCCESS)
          break;

  printf("The number of nodes is %d.", numberofnodes);

  MPI_Finalize();

}
```

Actually MPI has a very comprehensive error handling capability. You can redefine it to abort (the default here), return an error condition, or even call your own handler to do anything you want.

This would work here:

```
MPI_Comm_set_errhandler(MPI_COMM_WORLD, MPI_ERRORS_RETURN)
```