MPI — Message Passing Interface

- The nonblocking calls `MPI_Isend` and `MPI_Irecv` (where the 'I' stands for 'immediate' or 'incomplete') do not wait for their counterpart. They tell the runtime system here is some data and please send it as follows or here is some buffer space, and expect such-and-such data to come. (VE-V2-116–118)
- Issuing the `MPI_Isend` / `MPI_Irecv` call is sometimes referred to as posting a send/receive.
- Note from the syntax of the non-blocking receive that it does not have a status object. This is because the receive is not finished on that line, it only finishes when the `MPI_Wait` call is finished and here it returns a status.
- The non-blocking functions return an `MPI_Request` object which is really not an object but simply a pointer flag that can be used to track requests.
- The `MPI_Wait` call is blocking. It also clears the request pointer and sets it to NULL, specifically, `MPI_REQUEST_NULL`.
- Note that you can mix blocking and non-blocking send and receive calls. Example: NBSendRecv

MPI — Message Passing Interface

– Some things to be careful about. This mode is getting back to the race conditions we had with shared memory systems.
  - Between the `MPI_Irecv` and the `MPI_Wait` you cannot assume that the receive buffer has the sent value in it. Only after the wait is finished can you be sure of the buffer value. This may mean, depending on your application, that you need separate buffers to work with.
  - After an `MPI_Isend` you do not know when the sending buffer has been transmitted. So altering the contents after the send may alter the sent buffer. Again, there may be a need for separate buffers.

  Example: NBSendRecv2

– You can combine several wait commands if you want control over a number of sends and receives by using an array of requests. The `MPI_Waitall` command will wait (block) until a the requests in the array have been satisfied. (VE-V2-120)
  Example: NBWaitAll

MPI — Message Passing Interface

- The `MPI_Waitany` command will wait (block) until one request in the array have been satisfied, it will return the index of that satisfied request from the array. (VE-V2-122)
  Example: NBWaitAny
  Example: NBWaitAny2
- Note that the `MPI_Recv` call can match any of the send routines, and conversely a message sent with `MPI_Send` can be received by `MPI_Irecv`.
- Each of the blocking collective communication commands have a non-blocking counterpart that produces a request. (VE-V2-83)
  Blocking and non-blocking don't match: either all processes call the non-blocking collectives calls or all call the blocking one. For example the following is incorrect.

```
if (rank==0)
   MPI_Reduce(&x /* ... */ root, comm );
else
   MPI_Ireduce(&x /* ... */ );
```

MPI — Message Passing Interface

The basic idea behind all of this is that processes can do *other* work (that is, calculations that do not depend on the information being communicated between the processes) while the communication is in progress. This is a bit dependent on the hardware of your system but most modern processors will support this processing mode. If processes are doing other work while a communication is in progress the latency involved with synchronization is reduced since processors are not idling and waiting for the communication to finish. Hence the term *latency hiding*.

- `MPI_Ireduce` is the non-blocking reduce. Works like the sends and receives, wait command synchronizes. Syntax is the same, just the addition of an `MPI_Request` as the last parameter.
  Example: NBArrayReduce
- `MPI_Ibcast` is the non-blocking form of a broadcast. Syntax is the same, just the addition of an `MPI_Request` as the last parameter.
  Example: NBPiApproxRSMPIBcast

MPI — Message Passing Interface

- `MPI_Iallreduce` is the non-blocking form of the Allreduce. Syntax is the same, just the addition of an `MPI_Request` as the last parameter.
  Recall the standard deviation calculation:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_1 - \mu)^2} \qquad \text{where} \qquad \mu = \frac{1}{n}\sum_{i=1}^{n} x_1$$

  Example: NBAllReduceStdDev
- `MPI_Iscatter` is the non-blocking form of scatter. Syntax is the same, just the addition of an `MPI_Request` as the last parameter.
  Example: NBAllReduceStdDevSct
- `MPI_Igather` is the non-blocking form of gather. Syntax is the same, just the addition of an `MPI_Request` as the last parameter.
  Example: NBMatVecMult
- The other collective communications as well as the vector forms of these collective communications are similar in both method and syntax.

Reference Roadmap

The slide outlines contain references to the main course materials. Not everything has a reference but nearly all the materials can be found in the following references. The references are of the form `(<text>-<pages>)` so (PM-123) means page 123 of An Introduction to Parallel Programming by Peter S. Pacheco and Matthew Malensek.

- (VE-V1) — The Science of Computing: The Art of High Performance Computing, Vol 1 by Victor Eijkhout
- (VE-V2) — Parallel Programming in MPI and OpenMP: The Art of HPC, Vol 2 by Victor Eijkhout
- (VE-V3) — Introduction to Scientific Programming in C++17/Fortran2008: The Art of HPC, Vol 3 by Victor Eijkhout
- (VE-V4) — Tutorials for High Performance Scientific Computing: The Art of HPC, Vol 4 by Victor Eijkhout
- (PM) — An Introduction to Parallel Programming by Peter S. Pacheco and Matthew Malensek.
- (KH) — Programming Massively Parallel Processors: A Hands-on Approach by David B. Kirk and Wen-mei W. Hwu.
- (SAB) — High Performance Computing Modern Systems and Practices by Thomas Sterling, Matthew Anderson, and Maciej Brodowicz.
- (GLS) — Using MPI: Portable Parallel Programming with the Message-Passing Interface by William Gropp, Ewing Lusk, and Anthony Skjellum.
- (GHTL) — Using Advanced MPI: Modern Features of the Message-Passing Interface by William Gropp, Torsten Hoefler, Rajeev Thakur, and Ewing Lusk.