Distributed (Parallel) Simulation with ns-3

WNS3 2015 Tutorial, Castelldefels (Barcelona), Spain

https://www.nsnam.org/wiki/AnnualTraining2015

May 12, 2015

Peter D. Barnes, Jr (LLNL)
Ken Renard (ARL)

Why should you care about distributed (parallel) simulation?

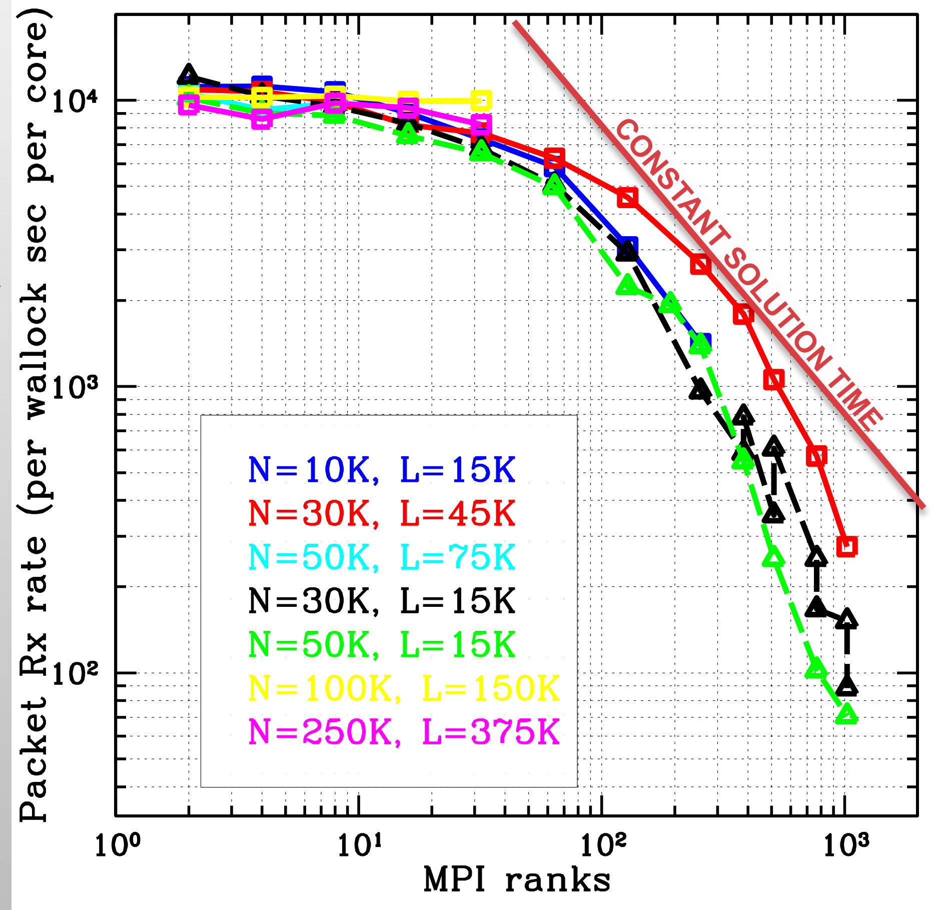
- Faster execution
  - Measure ~10^4 packet receives/wall clock second/core
- Large models, too big for one compute node
- Heavy-weight nodes
  - DCE applications
  - Virtual machines
  - Core routers with large forwarding tables

Motivation for High Performance, Scalable Network Simulation

- Reduce simulation run-time for large, complex network simulations
  - Complex models require more CPU cycles and memory
    - MANETs, robust radio devices
    - More realistic application-layer models and traffic loading
    - Load balancing among CPUs
  - Potential to enable real-time performance for NS-3 emulation
- Enable larger simulated networks
  - Distribute memory footprint to reduce swap usage
  - Potential to reduce impact of N^2 problems such as global routing
- Allows network researchers to run multiple simulations and collect significant data

ns-3 Execution Scaling

- $10^{\sim 4}$ packets/core/sec
  - Independent of model size
  - 100 cores is 100x faster than 1 core



How many hardware threads do you have?

This laptop

- MacBook Pro, mid 2009
- Intel Core 2 Duo:
  2 hardware threads

My other computer

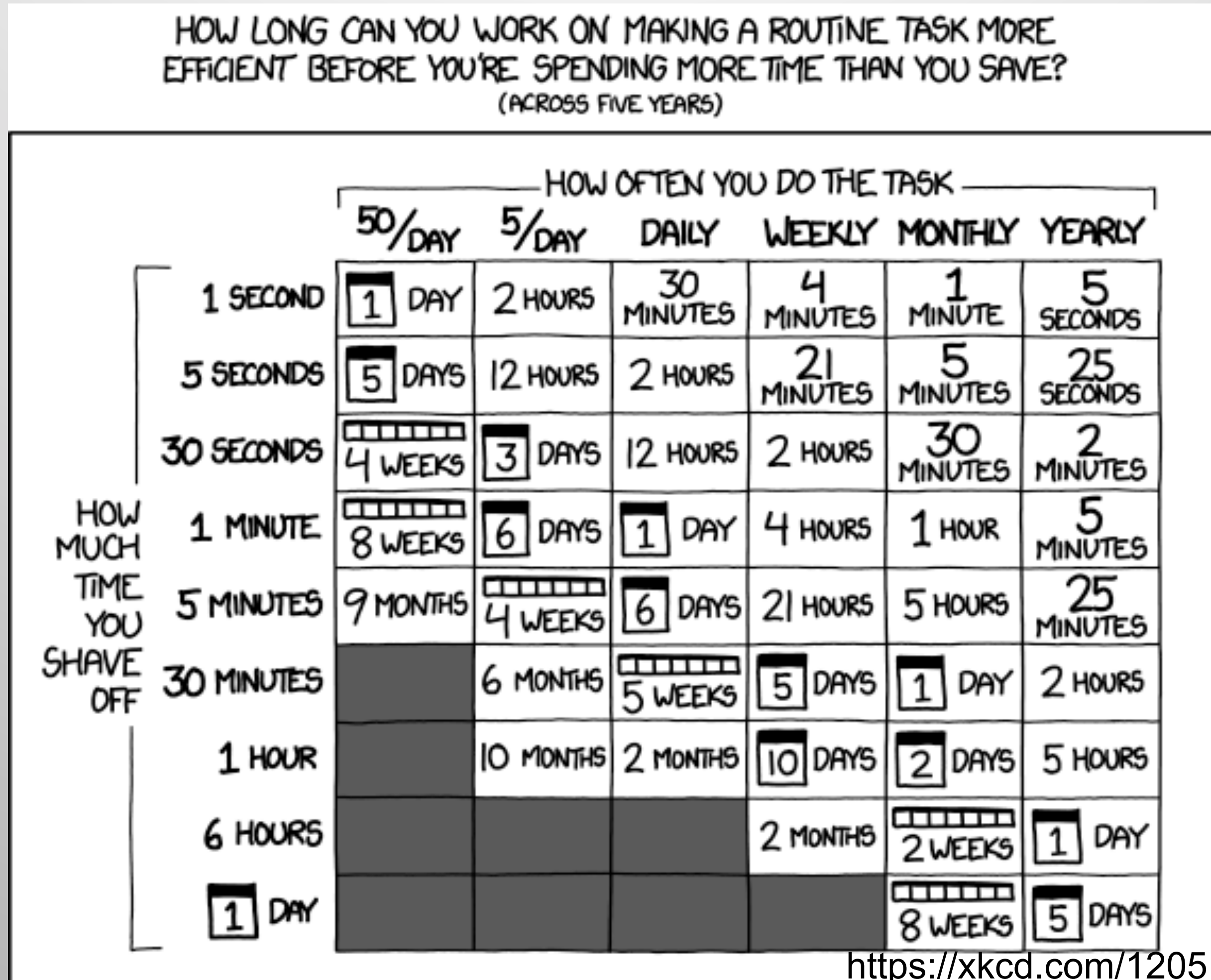
- BlueGene/Q
  - Currently TOP500 #3
  - 8M hardware threads

Outline

- Motivation
- Intro to MPI
- Parallel Discrete Event Si
- PDES in ns-3
- Constructing models
- Example
- Error Conditions
- Performance
- Future Capabilities

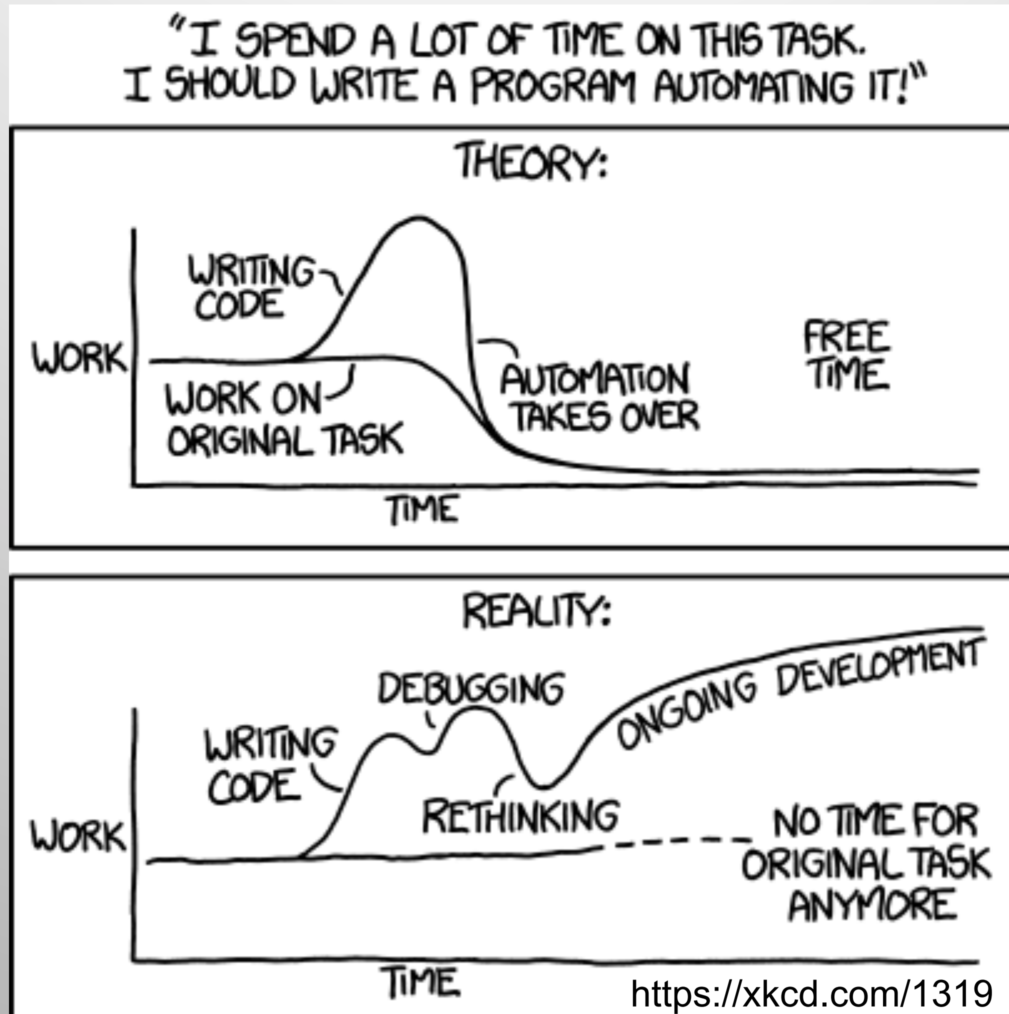
Big topics. Focus on the concepts and terms needed to understand //ns-3

Parallelizing ns-3 Models Is Straightforward, But…



https://xkcd.com/1205

And the Reality…



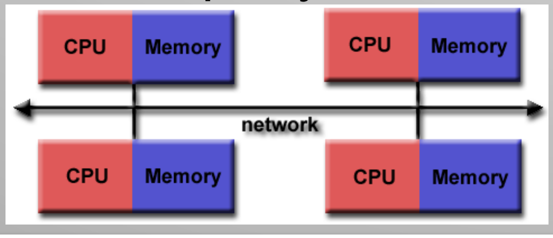
MPI Topics

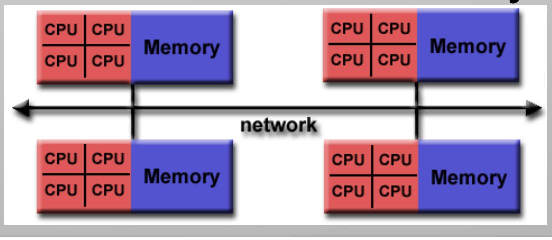
- Core features
- API specification
- Ranks and communicators
- Point-to-point messages
- Collectives
- Getting and using MPI

- Examples
  - Hello World
  - Simple messaging
  - Ghost cell pattern

Message Passing Interface (MPI) Features 1

- *De facto* standard programming model for parallel scientific codes (but see Charm++ for an alternative)
- Basic functionality is sending messages (data) between processes, which are called *ranks*
- Core features
  - API specification
    - ~Language independent (FORTRAN, C, C++, Python, Java,…)
    - Supports (doesn't preclude) high performance and scalability
  - Point-to-point (src,dst) messaging, as well as collectives
    - Broadcast, reduction (compute min value), …
  - Works equally well on distributed and shared memory





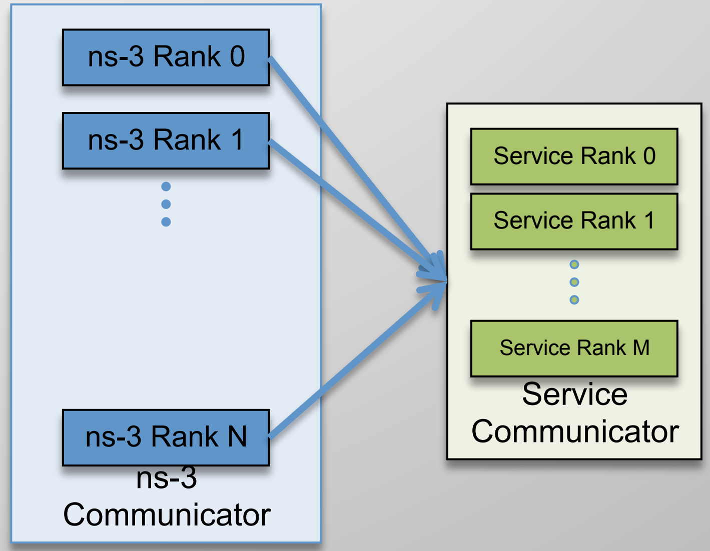
https://computing.llnl.gov/tutorials/mpi/

MPI Features 2

- API specification, implementation up to "vendor"
  - Vary in performance, runtime launch, …
  - Architecture-specific libraries can target specialized hardware
    - High-speed interconnects: Infiniband, PAMI, …
    - Specialized network topologies: Fat-tree, Dragonfly, 5-d torus
    - Specialized network interfaces: low-latency, high-throughput
    - Multi-path routing
  - Multiple implementations can coexist (but not interoperate)
    - OpenMPI, MPICH, IBM, …
    - Language-specific: mpi4py, mpiJava, …

MPI Concepts
Ranks and Communicators

- Processes are called *ranks*
- Communicator
  - Group of ranks, numbered [0,R) within the group
  - Enables separating messages by purpose
  - Initial default communicator is MPI_COMM_WORLD
  - Several functions for creating communicators, to support specific topologies



MPI Concepts
Point-to-Point Messages

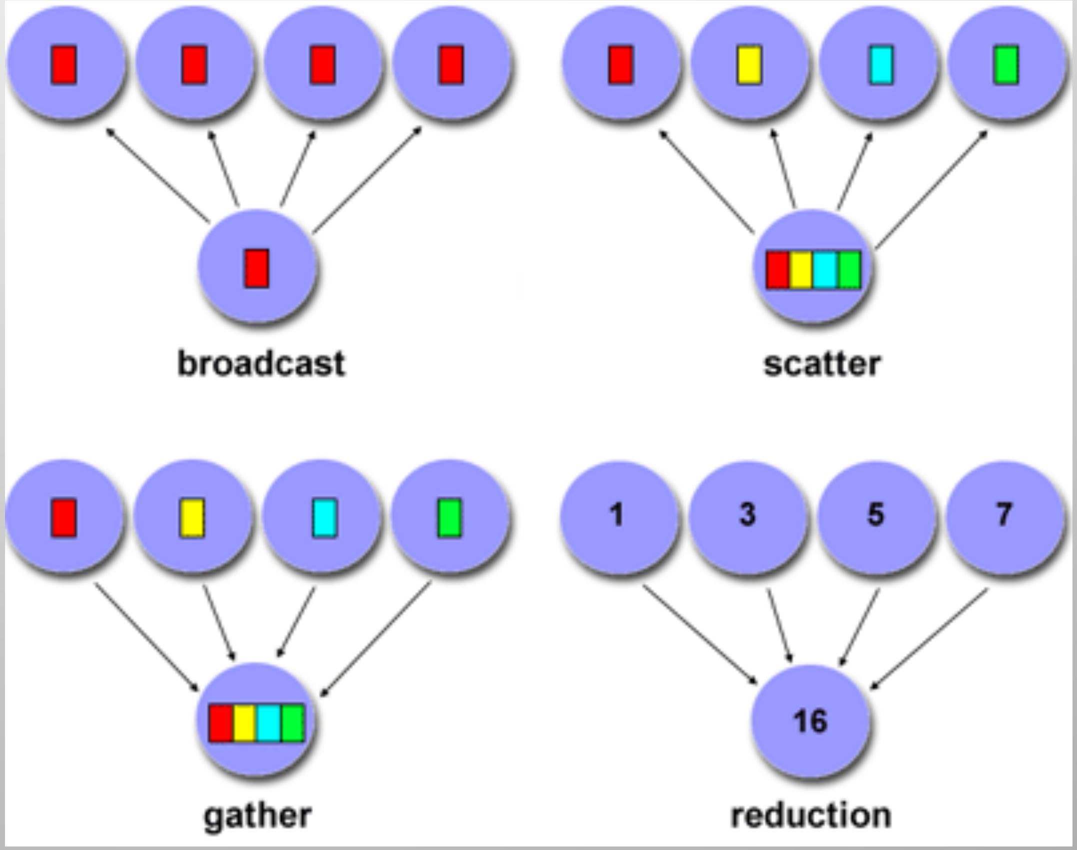
Send a message to a specific rank in a communicator

- `MPI_Send(data, data_length, data_type, destination, tag, communicator)`
  - `Data/data_length` Message contents
  - `data_type` MPI-defined data types
    Or a custom data type (*i.e*, a `struct`)
  - `destination` Rank Number
  - `tag` Application tag to distinguish types
  - `communicator`
- Matching `MPI_Recv()`
- Blocking and non-blocking versions
- Various optimized calls, for managing memory
- Wait for a message, test for new messages

MPI Concepts 3
Collective Communications

Higher level patterns involving more than 2 ranks

- Synchronization
  - MPI_Barrier
- Data movement
  - MPI_Bcast
    MPI_Scatter
    MPI_Gather
    MPI_Allgather
- Reductions
  - MPI_Reduce
    MPI_Allreduce
- Combinations
  - MPI_Reduce_scatter
    MPI_Alltoall
    MPI_Scan



MPI Full API

Environment Management					
MPI_Abort	MPI_Errorhandler_create	MPI_Errorhandler_free	MPI_Errorhandler_get	MPI_Errorhandler_set	MPI_Error_class
MPI_Error_string	MPI_Finalize	MPI_Get_processor_name	MPI_Get_version	MPI_Init	MPI_Initialized
MPI_Wtick	MPI_Wtime				
Point-to-Point Communication					
MPI_Bsend	MPI_Bsend_init	MPI_Buffer_attach	MPI_Buffer_detach	MPI_Cancel	MPI_Get_count
MPI_Get_elements	MPI_Isend	MPI_Iprobe	MPI_Irecv	MPI_Irsend	MPI_Isend
MPI_Issend	MPI_Probe	MPI_Recv	MPI_Recv_init	MPI_Request_free	MPI_Rsend
MPI_Rsend_init	MPI_Ssend	MPI_Ssend_init	MPI_Start	MPI_Startall	MPI_Test
MPI_Test_cancelled	MPI_Testall	MPI_Testany	MPI_Testsome	MPI_Wait	MPI_Waitall
MPI_Waitany	MPI_Waitsome				
Collective Communication					
MPI_Allgather	MPI_Allgatherv	MPI_Allreduce	MPI_Alltoall	MPI_Alltoallv	MPI_Barrier
MPI_Bcast	MPI_Gather	MPI_Gatherv	MPI_Op_create	MPI_Op_free	MPI_Reduce
MPI_Reduce_scatter	MPI_Scan	MPI_Scatter	MPI_Scatterv		
Process Group					
MPI_Group_compare	MPI_Group_difference	MPI_Group_excl	MPI_Group_free	MPI_Group_incl	MPI_Group_intersection
MPI_Group_range_excl	MPI_Group_range_incl	MPI_Group_rank	MPI_Group_size	MPI_Group_translate_ran	MPI_Group_union
Communicators					
MPI_Comm_compare	MPI_Comm_create	MPI_Comm_dup	MPI_Comm_free	MPI_Comm_group	MPI_Comm_rank
MPI_Comm_remote_group	MPI_Comm_remote_size	MPI_Comm_size	MPI_Comm_split	MPI_Comm_test_inter	MPI_Intercomm_create
MPI_Intercomm_merge					
Derived Types					
MPI_Type_commit	MPI_Type_contiguous	MPI_Type_extent	MPI_Type_free	MPI_Type_hindexed	MPI_Type_hvector
MPI_Type_indexed	MPI_Type_lb	MPI_Type_size	MPI_Type_struct	MPI_Type_ub	MPI_Type_vector
Virtual Topology					
MPI_Cart_coords	MPI_Cart_create	MPI_Cart_get	MPI_Cart_map	MPI_Cart_rank	MPI_Cart_shift
MPI_Cart_sub	MPI_Cartdim_get	MPI_Dims_create	MPI_Graph_create	MPI_Graph_get	MPI_Graph_map
MPI_Graph_neighbors	MPI_Graph_neighbors_cou nt	MPI_Graphdims_get	MPI_Topo_test		
Miscellaneous					
MPI_Address	MPI_Attr_delete	MPI_Attr_get	MPI_Attr_put	MPI_Keyval_create	MPI_Keyval_free
MPI_Pack	MPI_Pack_size	MPI_Pcontrol	MPI_Unpack		

Getting and Using MPI

- Check your package manager
- Only the API defined
  - Tool names and configuration vary
  - OpenMPI commands used here for illustration
- Building your code
  - Typically a compiler wrapper script, to ensure correct includes and libraries: `$ mpicc …`
  - Often hidden inside your build system (`Makefile`, `wscript`)
- Multiple executables
  - Possible to run different executables on different ranks
  - But job launch commands depend on package, so not portable
  - Typically build everything into one executable, select functions based on rank id at runtime

Getting and Using MPI

- Where to run ranks?
  - Single computer: typically defaults to all hardware threads
  - Ad hoc cluster: `--hostfile` node names and max number of ranks
  - HPC cluster: typically via a batch job system, which selects physical nodes and launches jobs as a shell script
  - "Overcommitment": running more ranks than cores or hardware threads
- Launching
  `$ mpirun –n <nranks> <executable>`
  - Need to launch on each host

Example OpenMPI Hosts File

```
# This is an example hostfile.  Comments begin with #
#
# The following node is a single processor machine:
foo.example.com

# The following node is a dual-processor machine:
bar.example.com slots=2

# The following node is a quad-processor machine,
# over-subscribing disallowed
yow.example.com slots=4 max-slots=4
```

Example OpenMPI Build and Run

```
$ mpicxx -o hello hello.cc
$ mpirun -np 4 ./hello
Hello World from rank 3 of 4 (35986)
Hello World from rank 0 of 4 (35983)
Hello World from rank 1 of 4 (35984)
Hello World from rank 2 of 4 (35985)
```

Parallel Hello World

- Typical Structure
  1. Include header
  2. Initialize MPI with command-line args
  3. Get world size, my rank index
  4. Parallel code
     - Send messages, synchronize…
  5. Clean shutdown

6. Build and Launch

hello.cc

```
#include <mpi.h>                                            (1)

int
main (int argc, char **argv)
{
  int size, rank, rc;

  rc = MPI_Init (&argc, &argv);                             (2)
  if (rc != MPI_SUCCESS)
    MPI_Abort(MPI_COMM_WORLD, rc);

  MPI_Comm_size (MPI_COMM_WORLD, &size);                    (3)
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);

  printf ("Hello World from rank %d of %d                   (4)
           (%d)\n", rank, size, getpid ());

  MPI_Finalize();                                           (5)
}
```

Example OpenMPI Build and Run

```
$ mpicxx -o hello hello.cc                                  (6)
$ mpirun -n 4 ./hello
Hello World from rank 3 of 4 (35986)
Hello World from rank 0 of 4 (35983)
Hello World from rank 1 of 4 (35984)
Hello World from rank 2 of 4 (35985)
```

Simple Messaging Example

Rank0 --“Hello”--> Rank1

- Typical Structure
  1. Check #ranks
  2. Message data buffers
  3. Each rank runs different code
  4. Sends paired with Recv
  5. Send and Recv data lengths, types match

Example OpenMPI Build and Run

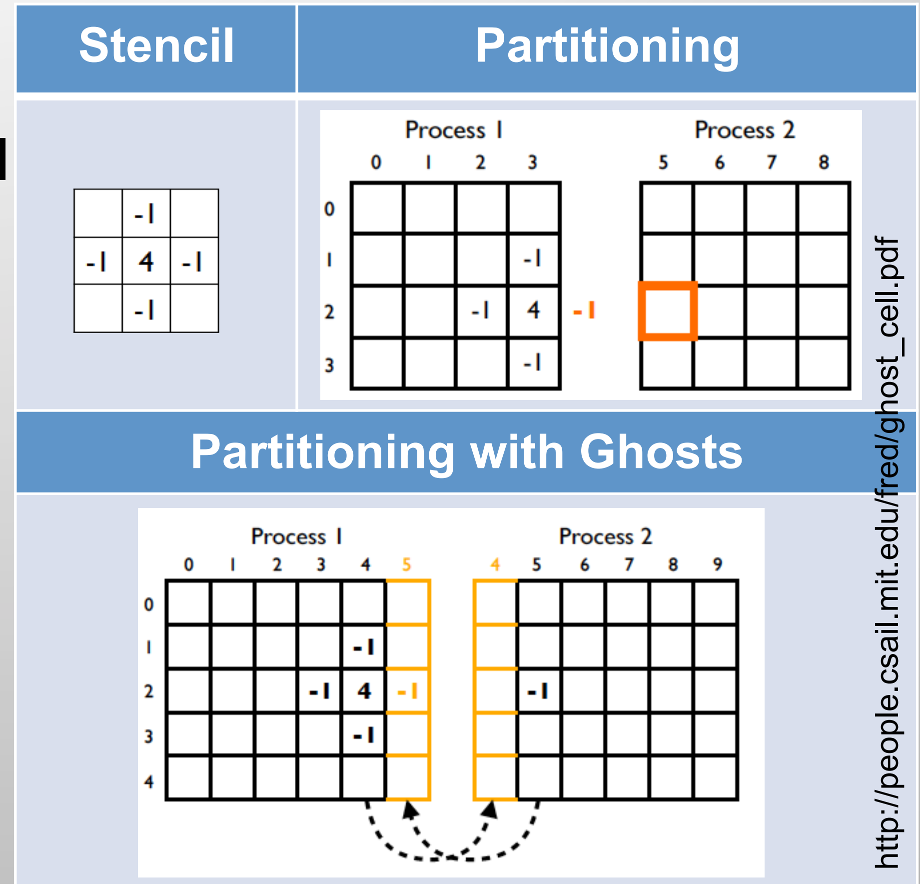
```
$ mpicxx -o send1 send1.cc
$ mpirun -np 4 ./send1
Rank 1 received message "Hello" (5) from rank 0 tag 0.
```

Parallel Body of send1.cc

```
(1) if (size < 2) {
      printf ("Need two ranks\n");
      MPI_Abort(MPI_COMM_WORLD, 0);
    }

(2) char *msg = (char *)"Hello";
    int msg_len = strlen(msg);
    char in_msg[msg_len + 1];  // leave space to add null

(3) if (rank == 0) {
      int dest = 1;
      rc = MPI_Send (msg, msg_len, MPI_CHAR, dest,
                     0, MPI_COMM_WORLD);
    }

(4) if (rank == 1) {                          (5)
      int count = 0;
      MPI_Status stat;

      rc = MPI_Recv (&in_msg, msg_len, MPI_CHAR,
                     MPI_ANY_SOURCE, 0, MPI_COMM_WORLD,
                     &stat);
      in_msg[msg_len] = (char) 0;
      MPI_Get_count (&stat, MPI_CHAR, &count);
      printf("Rank %d received message \"%s\" (%d) “
             “from rank %d tag %d.\n",
             rank, in_msg, count,
             stat.MPI_SOURCE, stat.MPI_TAG);
    }
```

Ghost Cell Design Pattern

- Decomposition
  - Need neighbors' data: stencil
  - Some neighbors are remote
- Solution:
  - Ghosts replicate data
  - Two-phase execution
    - — Exchange neighbor data *Communication*
    - — Compute local update *Computation*



Maximize *computation/communication*. Overlap computation and communication.

PDES Topics

- Discrete Event Simulation
  - Mathematical paradigm and time control
  - State and time evolution
  - Event scheduling
  - Time consuming processes
- Parallel DES
  - Logical processors
  - Causality
  - Granted-time synchronization
  - Lookahead
  - Null-message synchronization

Classification of Simulation Techniques

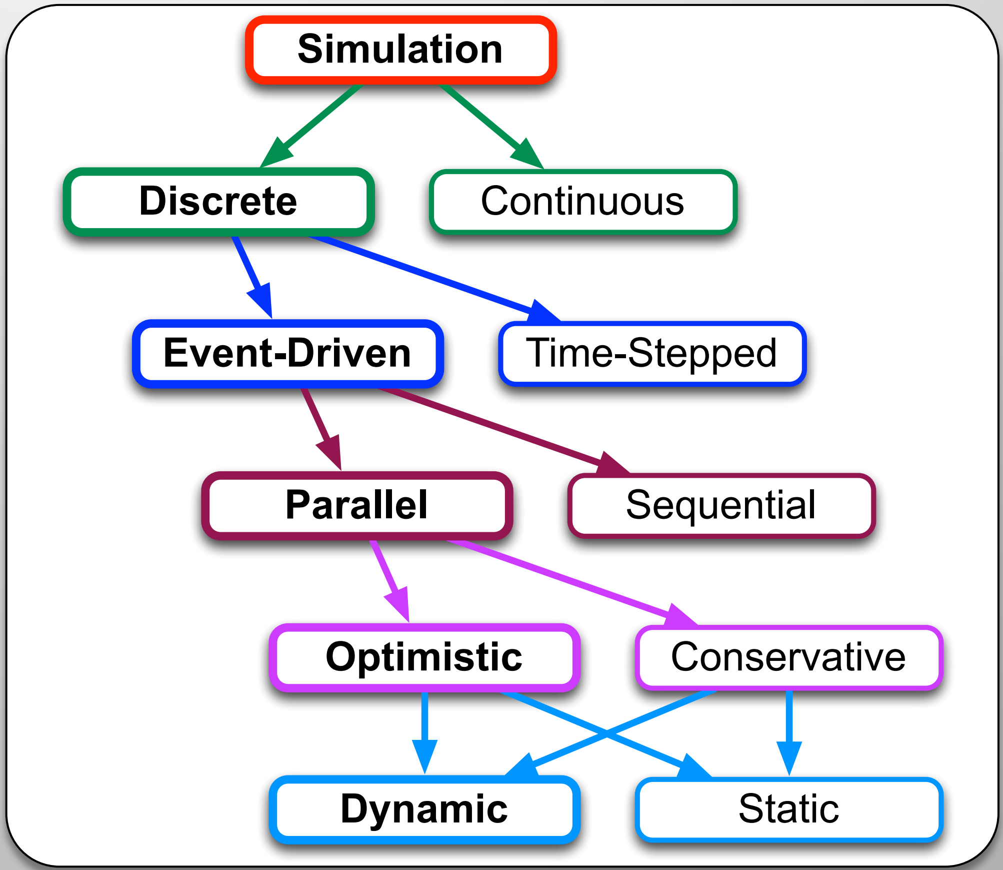
Mathematical Paradigm

Static *vs.* dynamic time control

Parallelism?

Synchronization Style

Load Distribution



Mathematical Paradigm

Aspect	Discrete	Continuous
Form of model	Discrete systems Automata, agents, particle systems, stochastic processes, *etc.*	Ordinary or partial differential equations
Time, space, state	Continuous or discrete	All continuous
State changes	Discontinuous in time Constant between state changes	Continuous in time Occasional discontinuities Piecewise differentiable
Mathematical tools	Probability and statistics	Numerical analysis

- Discrete simulation is natural when there are no underlying physical equations

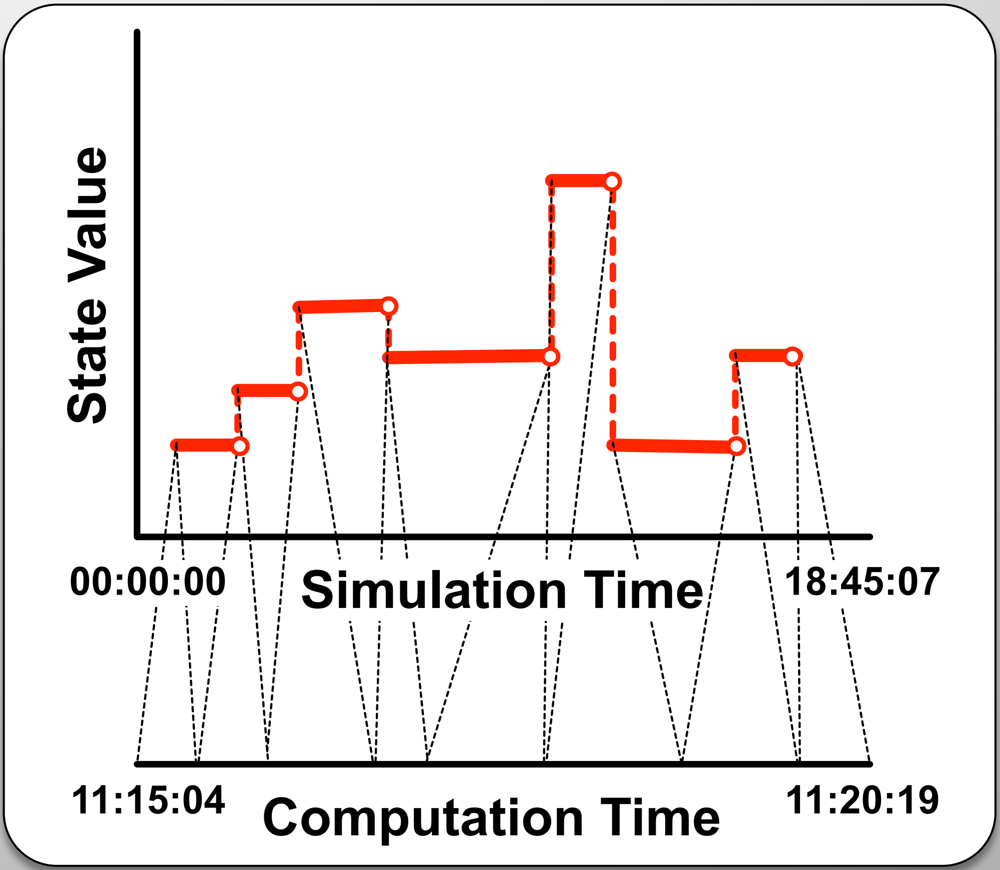
Time Control

Aspect	Event-Driven	Time-Stepped
Event times	Dynamically computed	Statically chosen
Time resolution	(Ideally) Floating point time Zero lower limit on resolution: inherently multi-scale	(Usually) Integer time Nonzero lower limit on resolution
Event distribution in space and time	Sparse and irregular	Dense and regular
Appropriate for	Irregular, asynchronous and/or multi-scale models	Spatially *and* temporally regular models

- Event-driven execution imposes no timescale
  - Supports simulation with wide dynamic range in natural time scales and/or long quiescent periods

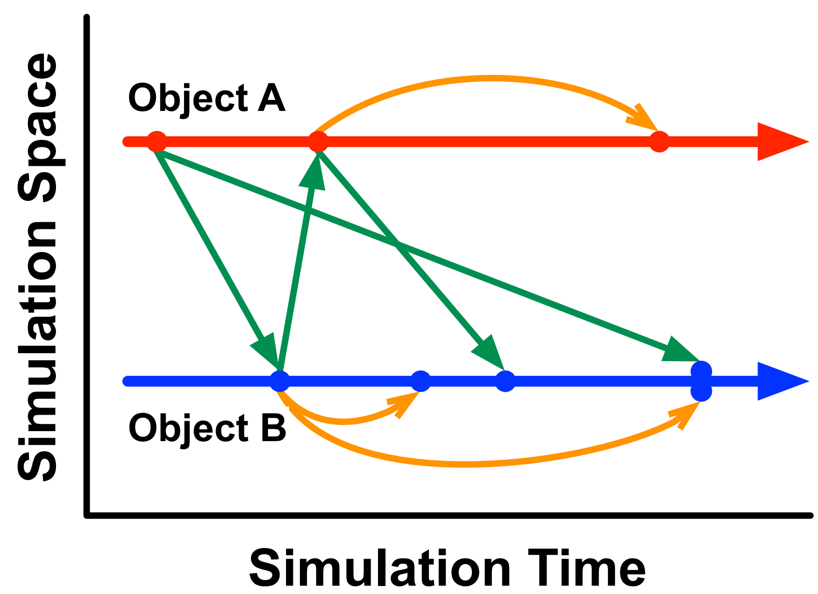
DES State and Time Evolution

- State values change discontinuously in simulation time
  - Constant between state changes
  - Time interval between state changes is not fixed
- Computation (real world time) required to compute new state value
  - Computation occurs at a fixed value of simulation time
- Rate of model evolution not fixed
  - Faster or slower than real time
  - (Best effort) real time, to interoperate with external real systems



DES Event Scheduling

- Objects communicate by sending *messages* = schedule events
  - *Event* is a function call, to be executed Δt in the future–*no backwards arrows*
  - Typically an event schedules one or more future events
- All event types allowed
  - Event to self
  - Event sends multiple messages
  - Event sends no messages
  - Events can tie
  - Non-FIFO scheduling



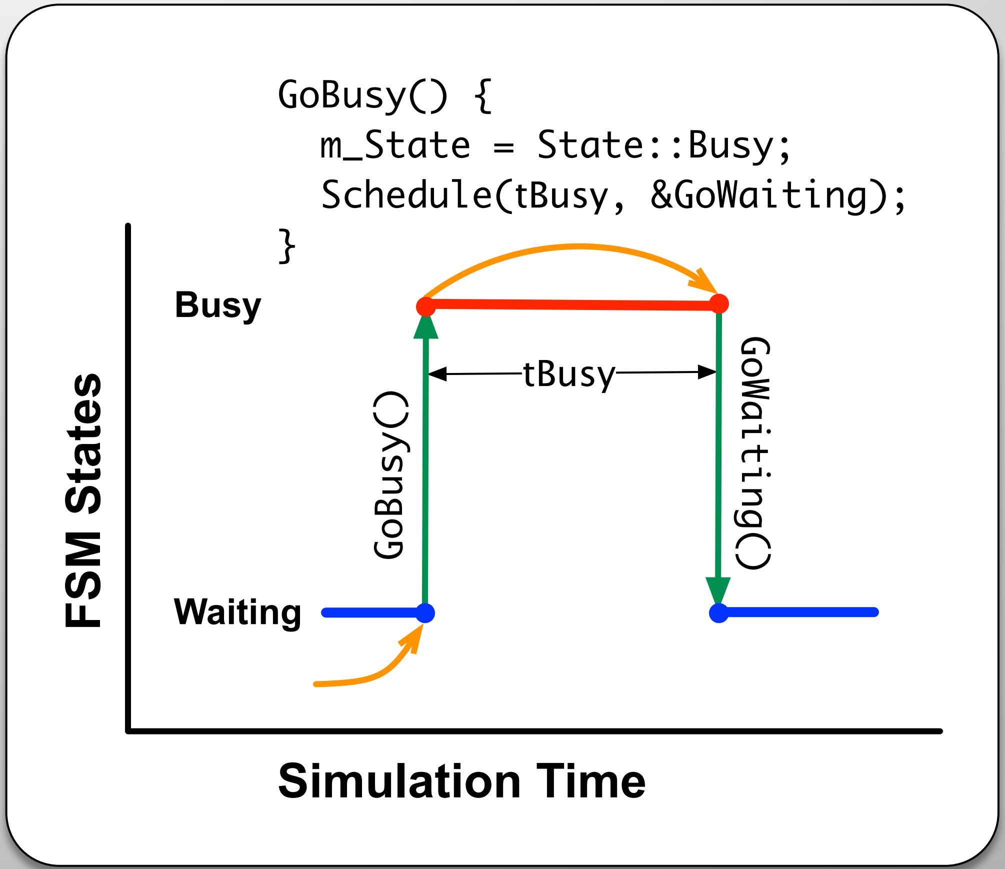
Sequential DES Main Event Loop

```
createInitialObjects();
eventList.insert(initialEvents);          // Priority queue on event time

while ( !(terminationCondition() || eventList.empty()) ) do
{
  event e = eventList.removeMinSimTime(); // Choose next event

  simTime = e.getEventTime();             // Set virtual time and unpack event
  object = e.getEventObject();
  method = e.getMethod();
  args = e.getArgs();

  object.method(args);                    // Invoke the event method
                                          // May change state of object
                                          // May schedule future events
                                          // May create or destroy objects
                                          // May cancel (delete) future events
}

finalize();
```

Clean separation between *Simulator* and *Application Model*.

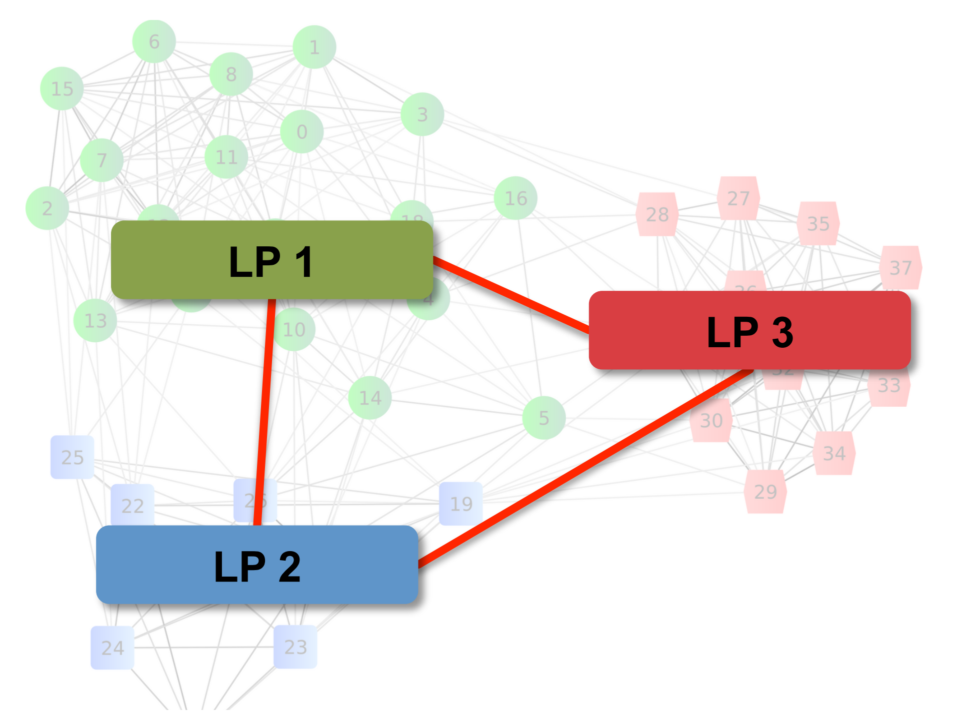
Modeling Time-Consuming Process

- Model state values change at an instant in simulation time
  - So how to model time-consuming processes?
- Finite state machine
  - Model state is the FSM state
  - Events cause FSM transitions, schedule future transitions



Parallel Discrete Event Simulation

- Decompose model into *Logical Processes*
  - Separate objects and event queues
  - Execute independently
  - Events for other LPs become messages
  - ~ MPI Ranks

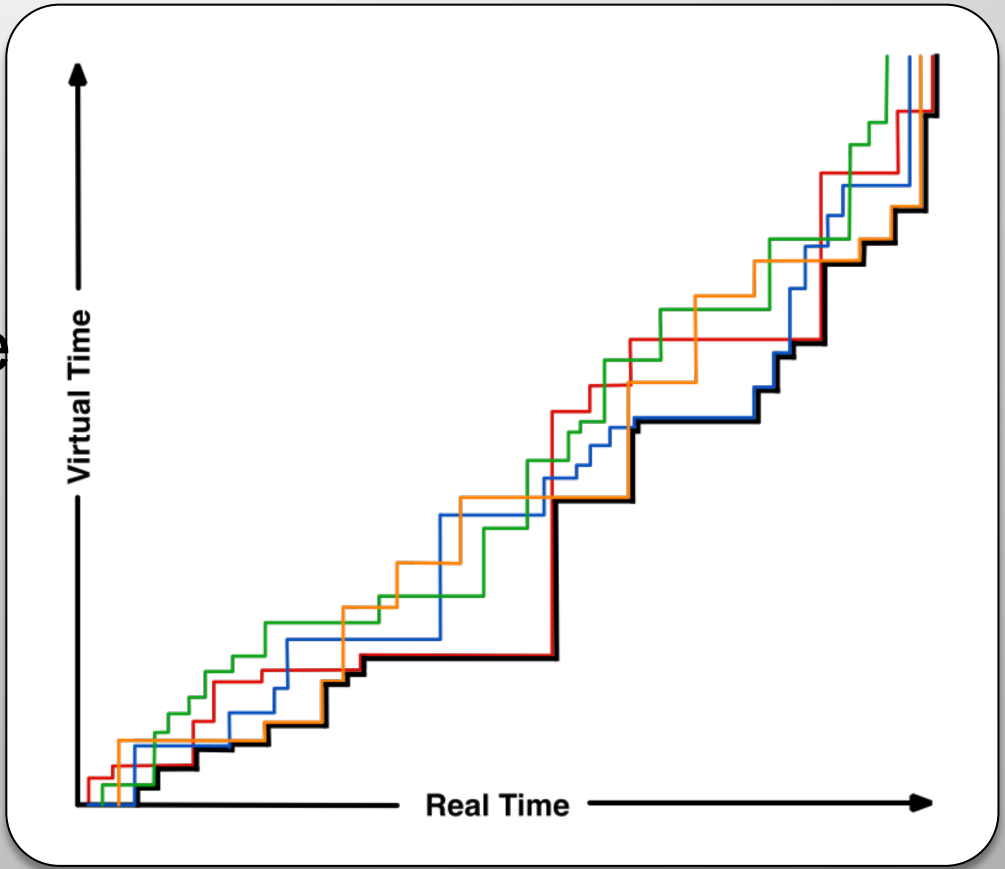


http://www.enggprojects.in/2011/07/adaptive-de-clustering-of-data.html

Parallel execution *must* produce exact same results as sequential!

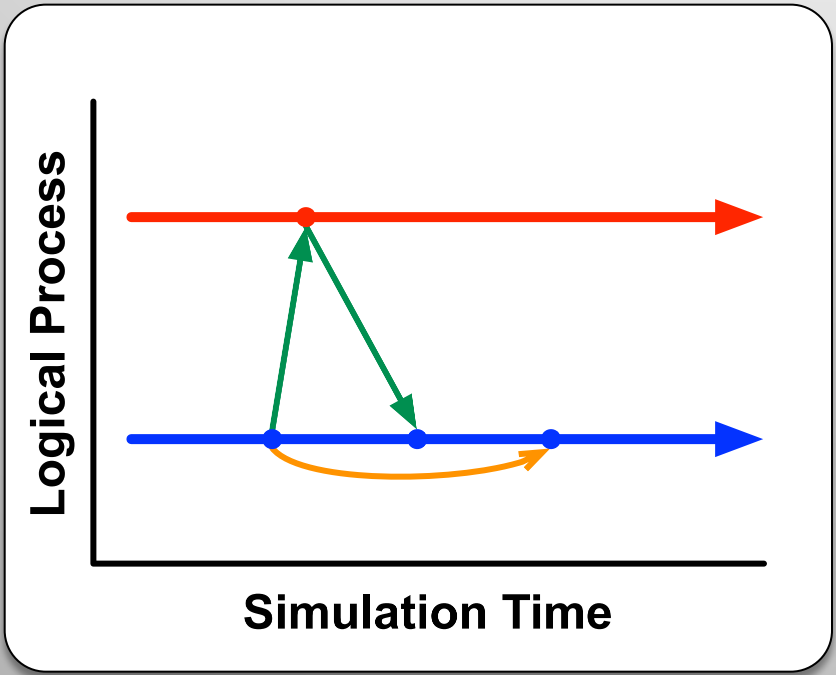
PDES Execution: LPs Advance Independently

- Sometimes ahead in virtual time, sometimes behind
- More or less real time per event
- Never backwards!
  - Hallmark of conservative execution (Ask me about optimistic execution ☺)

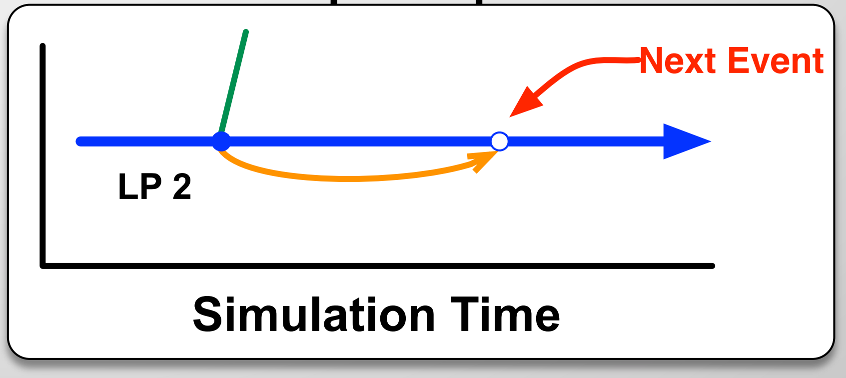


Need for Synchronization of LPs: Prevent Causality Violations

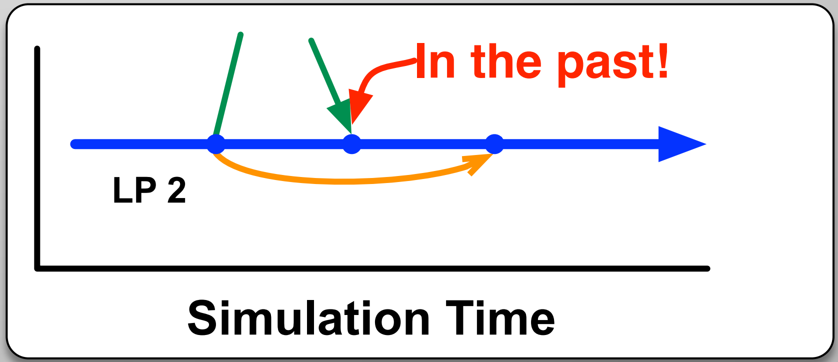
- Sequential event sequence



- LP 2's perspective:



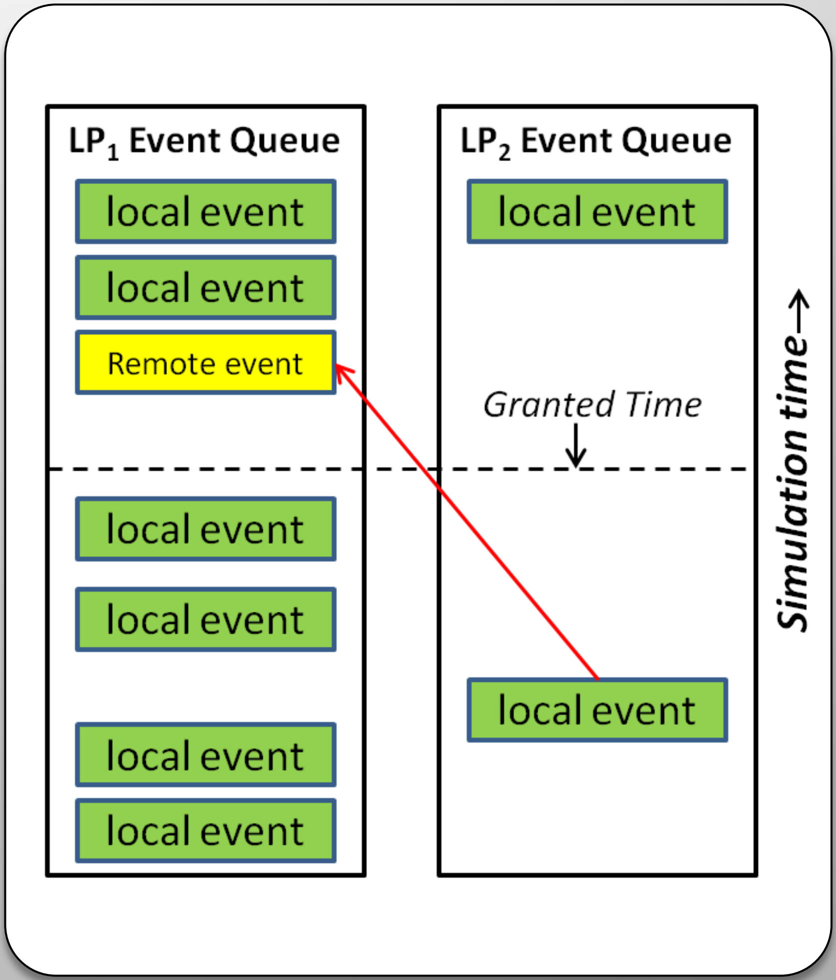
- Arrival of LP1 event



Need to guarantee no messages arrive in the past (for conservative PDES).

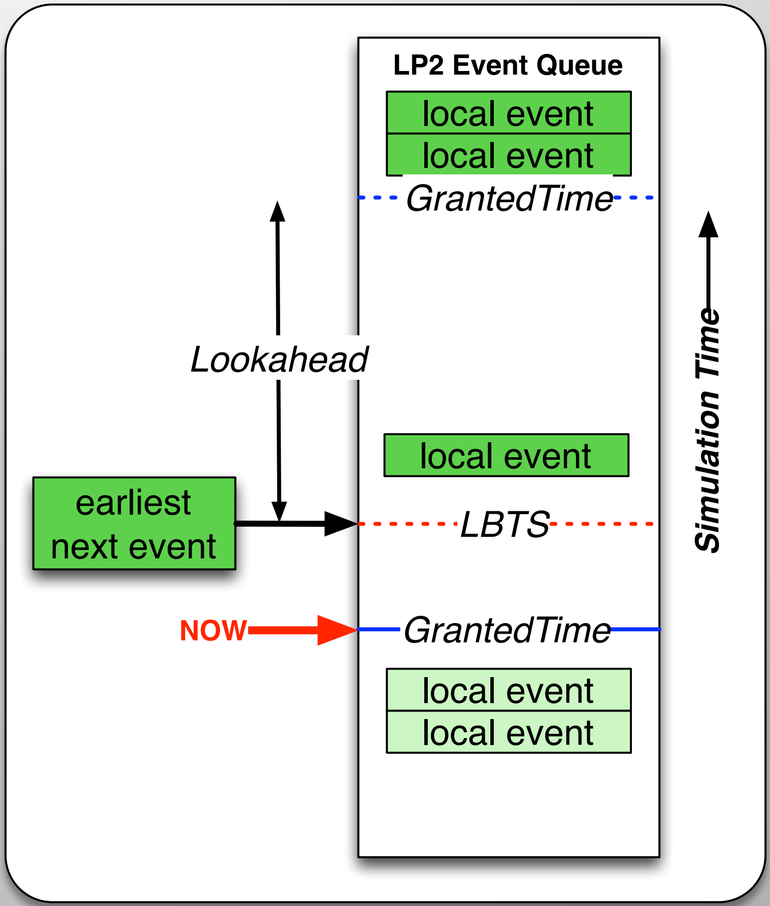
Granted Time Window Synchronization

- *If* we could guarantee no remote events will arrive before *GrantedTime*
  - All events before *GrantedTime* are safe
  - At *GrantedTime* need to synchronize:
    - Receive and schedule events from other LPs
    - Compute new *GrantedTime*
- Performance
  - Even workload distribution limits cpu idle time
  - Maximize *GrantedTime* to execute more events in parallel between synchronization



Lookahead and LBTS Provide the Granted Time Guarantee

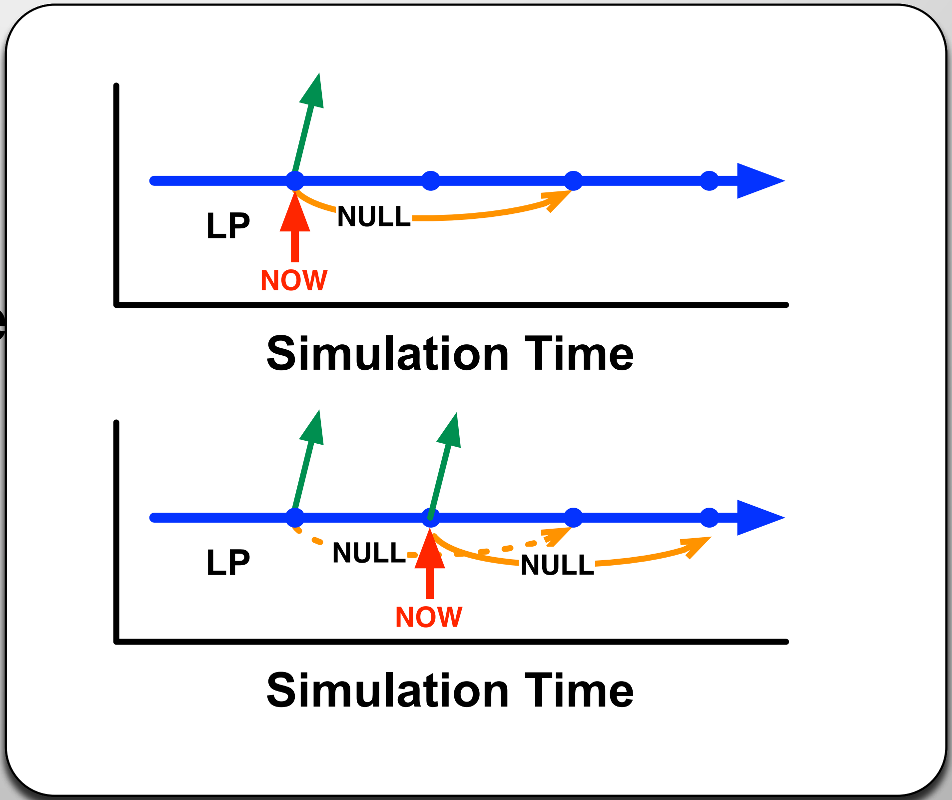
- Model must provide *Lookahead*
  - Minimum delay for remote events
    - Example: network channel link latency + transmission time for smallest packet
- Lower Bound Time Stamp (*LBTS*)
  - Min next event time across all LPs
- *GrantedTime = LBTS + Lookahead*
- Synchronization across LPs is expensive
  - Typically barrier (to wait for slowest LP)
  - Plus (at least one) all gather or reduction
  - Each of these is log(N_{LP}) in time



Finding large *Lookahead* is key to performance

Null-Message Alternative for Static and Sparse LP Graphs

- *GrantedTime* assumes all LPs can message all other LPs
- But my LP graph is sparse. Why synchronize with everyone?
  - Every message sent could communicate my virtual time
  - Guarantee at least one message every *Lookahead*
  - Send *Null-message* when necessary



To Learn More…

- Much of this material from a short course presented spring 2014
  - David Jefferson, LLNL, co-inventor of Optimistic PDES
  - 15 sessions
    - Sequential DES
    - Ties, LBTS, Lookahead
    - Chandy, Misra, Bryant: YAWNS
    - Deadlock
    - Null Messages
    - Dynamic Object Creation
    - Critical Path
    - Speedup
    - Optimistic DES, TimeWarp
    - Global Virtual Time
    - Commitment
    - Checkpointing
    - Rollback and Reverse Computation
    - Dynamic Load Balancing
    - Mixed Discrete and Continuous
  - Slides and videos publicly available: http://pdes-course-2014.ucllnl.org

Parallel ns-3

- History
- PDES in ns-3
- Mechanics
  - Enabling
  - Running
- PDES Simulators
  - `GrantedTime`
  - `NullMessage`
- Parallel Models 1
  - The easy way
- Lookahead
- Under the covers:
  - `PointToPointRemoteChannel`
  - `PointToPointNetDevice`
- Parallel Models 2
  - The hard way
  - Limitations

Parallel ns-3 History

- Initial release in ns-3.8
  - J. Pelkey and G. Riley, “Distributed Simulation with MPI in ns-3,” WNS3 2011, Barcelona, Spain.
  - Roots from:
    - Parallel/Distributed ns (pdns)
    - Georgia Tech Network Simulator (GTNetS)
- Publications
  - K. Renard, *et al*, “A Performance and Scalability Evaluation of the ns-3 Distributed Scheduler,” SimuTools 2012
  - S. Nikolaev, *et al*, “Performance of Distributed ns-3 Network Simulator,” SimuTools 2013
  - WNS3 2015

PDES in ns-3

Sequential ns-3

- LP is implicit
  - `ns3::Simulator`
- Event messages
  - Explicit future function calls

    `Schedule (delay, &fn,…)`
- Virtual time discipline

  `DefaultSimulatorImpl`
  `RealtimeSimulatorImpl`
  `VisualSimulatorImpl`

Parallel ns-3

- Each rank is an LP
- Event messages
  - Local to LP: explicit future function calls
  - Remote: implicit message send
- Virtual time discipline

  `DistributedSimulatorImpl`
  `NullMessageSimulatorImpl`
- Lookahead (later)

Enabling Parallel ns-3

- Configure with `--enable-mpi`
  - Tries to run `mpic++`
    - Recognizes OpenMPI and MPICH libraries
  - Defines `NS3_MPI` and either `NS3_OPENMPI` or `NS3_MPICH`
- Followed by usual build

Configuring ns-3 With MPI

```
$ ./waf configure --enable-mpi
Setting top to                          : ...
...
---- Summary of optional NS-3 features:
Build profile                     : debug
...
MPI Support                       : enabled
...
'configure' finished successfully (1.295s)

$ ./waf build
...
```

Running Parallel ns-3 Scripts

- Waf can't distinguish sequential and parallel
  - Need to specify `mpirun` and number of ranks explicitly

Running Parallel Scripts with `waf` and `mpirun`

```
$ ./waf --run simple-distributed
Waf: Entering directory `build/debug'
Waf: Leaving directory `build/debug'
'build' finished successfully (2.118s)
This simulation requires 2 and only 2 logical processors.
Command ['build/debug/src/mpi/examples/ns3-dev-simple-distributed-debug'] exited with code 1

#  Multiple ranks on a single computer:
$ ./waf --run simple-distributed --command-template="mpirun -np 2 %s"
Waf: Entering directory `build/debug'
Waf: Leaving directory `build/debug'
'build' finished successfully (2.104s)
At time 1.02264s packet sink received 512 bytes from 10.1.1.1 port 49153 total Rx 512 bytes
At time 1.0235s packet sink received 512 bytes from 10.1.2.1 port 49153 total Rx 512 bytes
At time 1.02437s packet sink received 512 bytes from 10.1.3.1 port 49153 total Rx 512 bytes
At time 1.02524s packet sink received 512 bytes from 10.1.4.1 port 49153 total Rx 512 bytes

# Multiple computers:
$ mpirun -np 2 ./waf -run simple-distributed
```

Switching Between GrantedTime and NullMessage Simulators

- Use environment variable

```
$ NS_GLOBAL_VALUE=\
  “SimulatorImplementationType=ns3::NullMessageSimulatorImpl”
\
  ./waf --run ...
```

- Use command line:

Selecting the Parallel Simulator from the Command Line

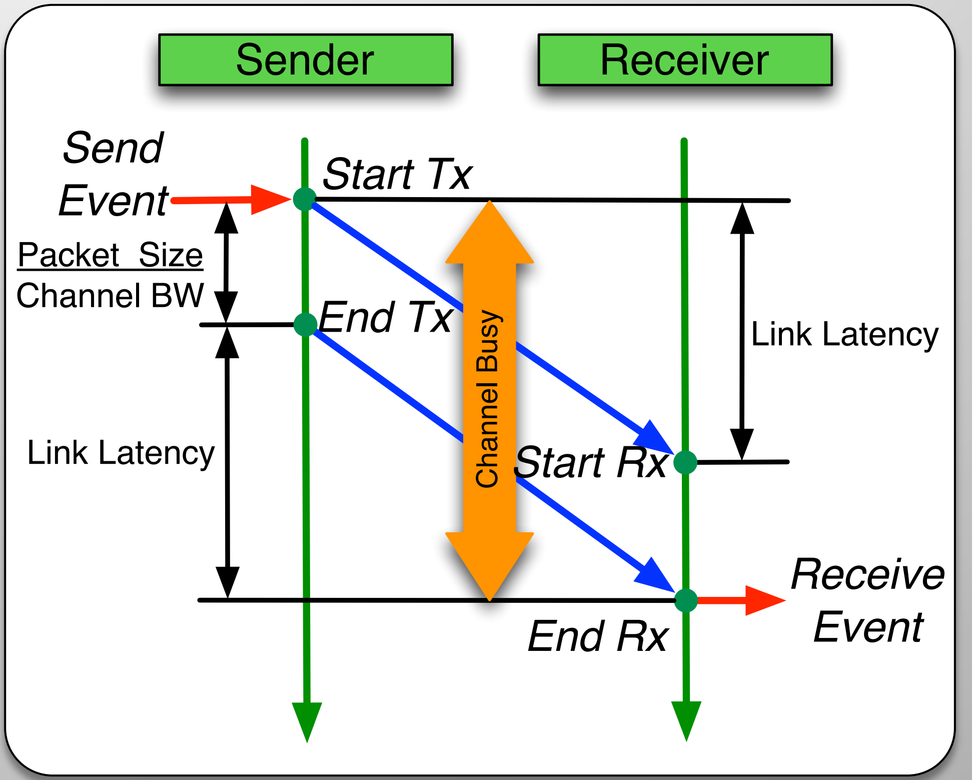
```
bool nullmsg = false;
CommandLine cmd;
cmd.AddValue ("nullmsg", "Enable the use of null-message synchronization", nullmsg);
cmd.Parse (argc,argv);
...
if(nullmsg) {
  GlobalValue::Bind ("SimulatorImplementationType",
                     StringValue ("ns3::NullMessageSimulatorImpl"));
} else {
  GlobalValue::Bind ("SimulatorImplementationType",
                      StringValue ("ns3::DistributedSimulatorImpl"));
}
MpiInterface::Enable (&argc, &argv);
```

Constructing Distributed Models The Easy Way

- All ranks construct the full topology
  - All Nodes, NetDevices and Channels
    - Label Nodes with rank: `Node::Node (uint32_t systemId)`
  - All Internet stacks and addresses
  - Good
    - Single code for model construction, runs sequential and parallel
    - Event execution happens in parallel
    - Enables GOD and NIx-vector routing to work
  - Bad
    - Memory is used for nodes/stacks/devices that “belong” to other ranks (But come to my talk tomorrow ☺)
- Install local applications only
  - Non-local nodes (not on my rank) should not have applications

Where to Get Lookahead?

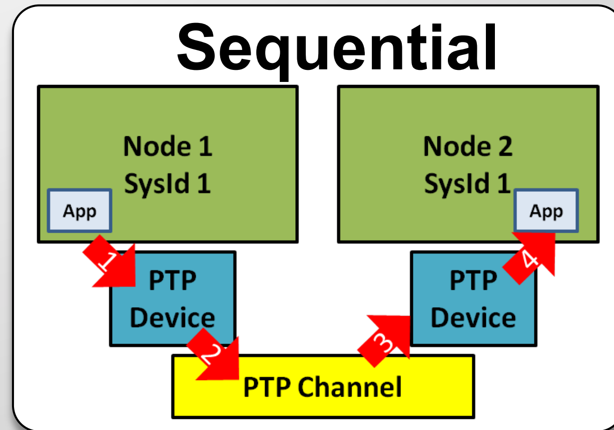
- Primarily from link latency
- What about shared channels like CSMA or wireless?
  - Latency can be zero
  - Multiple NetDevices
  - ➢ *Can't span ranks!*
- Only PointToPoint links can cross ranks
  - Global *Lookahead* is smallest cross-rank latency

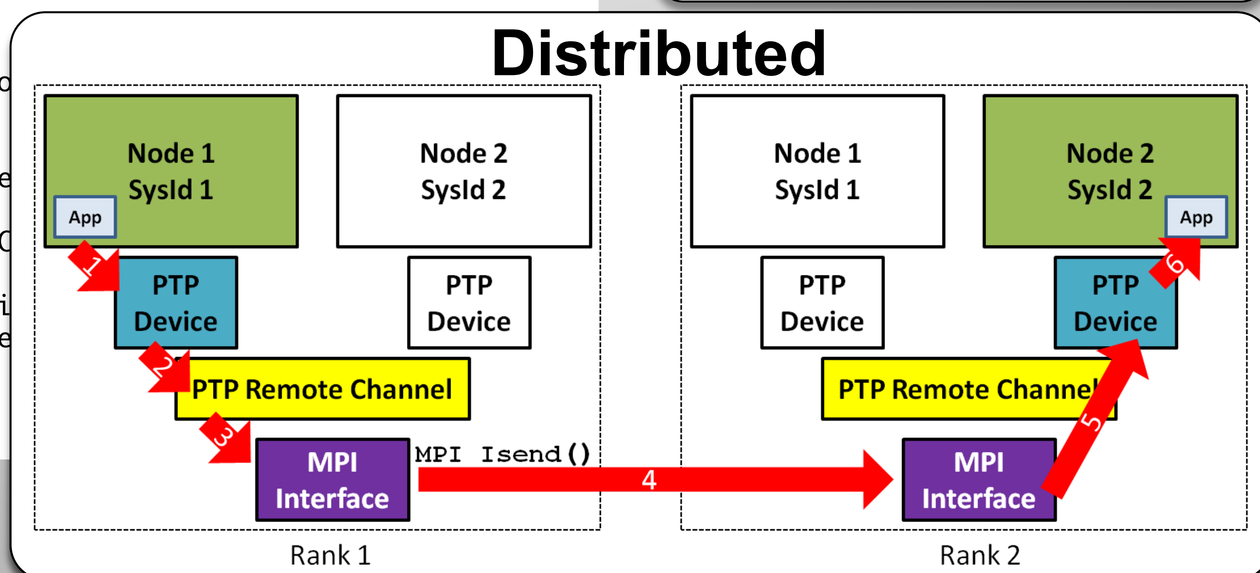


Under the Covers: PointToPointHelper::Install

src/point-to-point/helper/point-to-point-helper.cc

```
bool useNormalChannel = true;
Ptr<PointToPointChannel> channel = 0;

if (MpiInterface::IsEnabled ())  {
  uint32_t currSystemId = MpiInterface::GetSystemId ();
  if (a->GetSystemId () != currSystemId ||
      b->GetSystemId () != currSystemId) {
    useNormalChannel = false;
  }
}
if (useNormalChannel) {
  channel =
    m_channelFactory.Create<Po
} else {
  channel =
    m_remoteChannelFactory.Cre

  Ptr<MpiReceiver> mpiRecA = C
  mpiRecA->SetReceiveCallback
    (MakeCallback (&PointToPoi
  devA->AggregateObject (mpiRe

  // Same for b
}
```





Under the Covers: Sending a Packet from `PointToPointNetDevice`

PointToPointNetDevice Call Chain

```
PointToPointNetDevice::Send() {
  TransmitStart() {
    PointToPointRemoteChannel::TransmitStart() {
      MpiInterface::SendPacket();
```

- `MpiInterface::SendPacket()`
  - — *Packet data*
  - — *Receive time* – Local Now() + Latency + Packet Tx duration
  - — Remote SystemId (rank)
  - — Remote *NodeId*
  - — Remote *InterfaceId*
- Serialize packet and destination data
- Send to remote rank with non-blocking `MPI_Isend()`

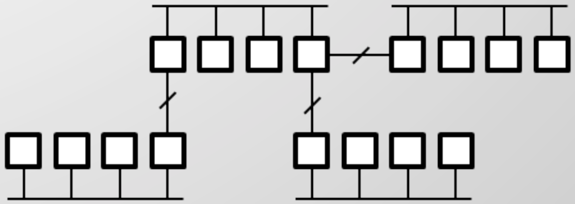
Under the Covers: Getting a Remote Packet to the `PointToPointNetDevice`

At end of *GrantedTime*, `DistributedSimulatorImpl` calls `GrantedTimeWindowMpiInterface::ReceiveMessages()`

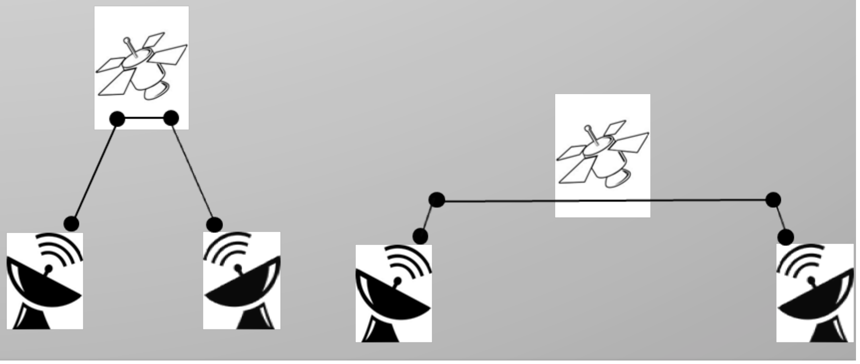
- Reads all pending MPI messages
  - Deserialize target *Receive time*, *NodeId* and *InterfaceId*
  - Deserialize *packet data*
  - Find Node by *NodeId*
  - Find `NetDevice` on Node with correct InterfaceId
  - Get `MpiReceiver` object aggregated to the `NetDevice`
    - `MpiReceiver` merely holds the correct `NetDevice` Callback
  - Schedule `MpiReceiver::Receive` event at *Receive time*

Building a Distributed ns-3 Simulation

- Choose partitioning strategy
  - Label contiguous regions which can't be partitioned
    - — CSMA and wireless
  - Select regions which will share a rank
    - — Find large point-to-point latencies for good *Lookahead*
    - — Minimize communication between ranks

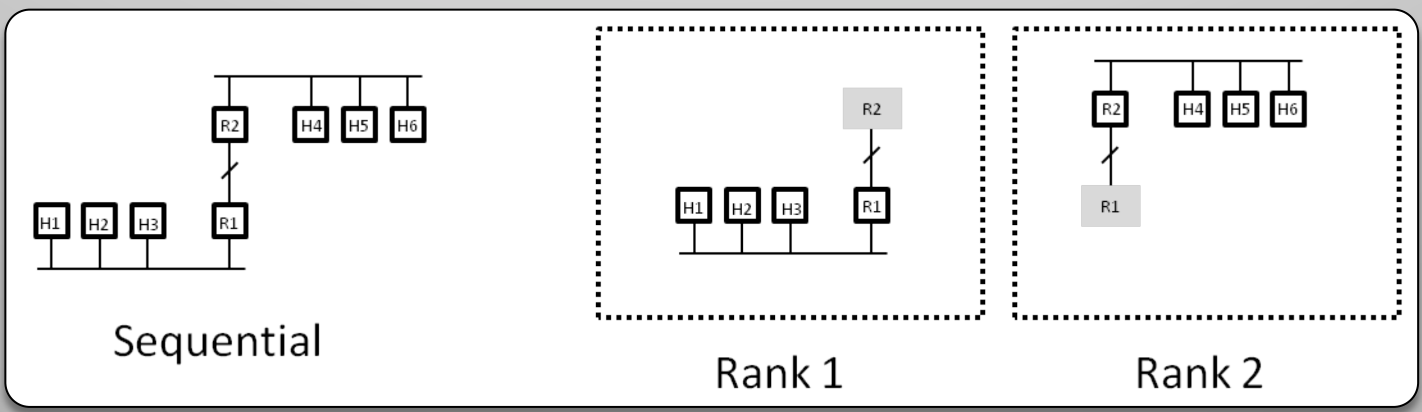


- Build topology as normal, assigning Nodes to ranks
  `CreateObject<Node> (rankId)`
- Rewrite topology to improve partitioning
  - CSMA with only 2 nodes
  - Move latency



Constructing Distributed Models The Hard Way

- Use the ghost cell design pattern to save memory
  - Only create local Nodes, Applications, Internet stacks, NetDevices and Channels
  - Plus “ghost” nodes: remote endpoint of PointToPointRemoteChannel
- Requires *manual intervention*
  - Global and NIX routing do not see entire topology
    - Add static, default routes manually. Hint: IPv6 allows for more “aggregatable” routes
  - Ghost nodes will likely have incorrect remote NodeId, InterfaceId
  - Must align interface identifiers by hand in same fashion



Limitations of Distributed NS3

- Partitioning is a manual process
- Partitioning is restricted to Point-To-Point links only
  - Partitioning within a wireless network is not supported
    - *Lookahead* is very small and dynamic
- Need full topology in all LPs
  - Exception with careful node ordering, interface numbering, and manual routing

examples/tutorial/third.cc — src/mpi/examples/third-distributed.cc

(These have diverged slightly in ns-3-dev. Differences minimized here.)

1. Include mpi-module.h
2. Same topology, split across Point-to-point link

examples/tutorial/third.cc

```
* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
*
*
*
*
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

include "ns3/core-module.h"
include "ns3/point-to-point-module.h"
include "ns3/network-module.h"
include "ns3/applications-module.h"
include "ns3/wifi-module.h"
include "ns3/mobility-module.h"
include "ns3/csma-module.h"
include "ns3/internet-module.h"

/ Default Network Topology
/
/   Wifi 10.1.3.0
/                 AP
/  *    *    *    *
/  |    |    |    |    10.1.1.0
/ n5   n6   n7   n0 -------------- n1   n2   n3   n4
/                   point-to-point  |    |    |    |
/                                   ================
/                                     LAN 10.1.2.0

sing namespace ns3;

S_LOG_COMPONENT_DEFINE ("ThirdScriptExample");

nt
ain (int argc, char *argv[])

 bool verbose = true;
 uint32_t nCsma = 3;
 uint32_t nWifi = 3;
 bool tracing = false;
```

src/mpi/examples/third-distributed.cc

```
* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */

rogram is free software; you can redistribute it and/or modify
er the terms of the GNU General Public License version 2 as
hed by the Free Software Foundation;
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

include "ns3/core-module.h"
include "ns3/point-to-point-module.h"
include "ns3/network-module.h"
include "ns3/applications-module.h"
include "ns3/wifi-module.h"
include "ns3/mobility-module.h"
include "ns3/csma-module.h"
include "ns3/internet-module.h"

include "ns3/mpi-module.h"

/ Default Network Topology
/ (same as third.cc from tutorial)
/ Distributed simulation, split along the p2p link
/ Number of wifi or csma nodes can be increased up to 250
/                          |
/                 Rank 0   |   Rank 1
/ -------------------------|----------------------------
/   Wifi 10.1.3.0
/                 AP
/  *    *    *    *
/  |    |    |    |    10.1.1.0
/ n5   n6   n7   n0 -------------- n1   n2   n3   n4
/                   point-to-point  |    |    |    |
/                                   ================
/                                     LAN 10.1.2.0

sing namespace ns3;

S_LOG_COMPONENT_DEFINE ("ThirdExampleDistributed");
```

Example

1. Different log component name
2. Command line argument to select Null message

examples/tutorial/third.cc

```
// |    |    |    |    10.1.1.0
// n5   n6   n7   n0 -------------- n1   n2   n3   n4
//                    point-to-point  |    |    |    |
//                                    ================
//                                      LAN 10.1.2.0

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("ThirdScriptExample");

int
main (int argc, char *argv[])
{
  bool verbose = true;
  uint32_t nCsma = 3;
  uint32_t nWifi = 3;
  bool tracing = false;

  CommandLine cmd;
  cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
  cmd.AddValue ("nWifi", "Number of wifi STA devices", nWifi);
  cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);
  cmd.AddValue ("tracing", "Enable pcap tracing", tracing);

  cmd.Parse (argc,argv);

  // Check for valid number of csma or wifi nodes
  // 250 should be enough, otherwise IP addresses
  // soon become an issue
  if (nWifi > 250 || nCsma > 250)
    {
      std::cout << "Too many wifi or csma nodes, no more than 250 each." << std::
      return 1;
    }

  if (verbose)
    {
      LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
      LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
    }
```

src/mpi/examples/third-distributed.cc

```
//                 Rank 0    |    Rank 1
// -------------------------|----------------------------
// n5   n6   n7   n0 -------------- n1   n2   n3   n4
//                    point-to-point  |    |    |    |
//                                    ================
//                                      LAN 10.1.2.0

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("ThirdExampleDistributed");

int
main (int argc, char *argv[])
{
  bool verbose = true;
  uint32_t nCsma = 3;
  uint32_t nWifi = 3;
  bool tracing = false;
  bool nullmsg = false;

  CommandLine cmd;
  cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
  cmd.AddValue ("nWifi", "Number of wifi STA devices", nWifi);
  cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);
  cmd.AddValue ("tracing", "Enable pcap tracing", tracing);
  cmd.AddValue ("nullmsg", "Enable the use of null-message synchronization",

  cmd.Parse (argc,argv);

  // Check for valid number of csma or wifi nodes
  // 250 should be enough, otherwise IP addresses
  // soon become an issue
  if (nWifi > 250 || nCsma > 250)
    {
      std::cout << "Too many wifi or csma nodes, no more than 250 each." << std::
      return 1;
    }

  if (verbose)
    {
      LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
      LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
```

examples/tutorial/third.cc

1. Condition on NS3_MPI
2. Null message selector
3. Initialize MPI
4. Get rank #, number of ranks
5. Check number of ranks
6. Use symbolic names for each rank
7. Create point-to-point nodes

```
CommandLine cmd;

...

if (verbose)
  {
    LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
    LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
  }

NodeContainer p2pNodes;
p2pNodes.Create (2);

PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);

NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);

CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);

NodeContainer wifiStaNodes;
wifiStaNodes.Create (nWifi);
```

src/mpi/examples/third-distributed.cc

```
  // Sequential fallback values
  uint32_t systemId = 0;
  uint32_t systemCount = 1;

#ifdef NS3_MPI

  // Distributed simulation setup; by default use granted time window algorithm.
  if(nullmsg)
    {
      GlobalValue::Bind ("SimulatorImplementationType",
                         StringValue ("ns3::NullMessageSimulatorImpl"));
    }
  else
    {
      GlobalValue::Bind ("SimulatorImplementationType",
                         StringValue ("ns3::DistributedSimulatorImpl"));
    }

  MpiInterface::Enable (&argc, &argv);

  systemId = MpiInterface::GetSystemId ();
  systemCount = MpiInterface::GetSize ();

  // Check for valid distributed parameters.
  // Must have 2 and only 2 Logical Processors (LPs)
  if (systemCount != 2)
    {
      std::cout << "This simulation requires 2 and only 2 logical processors. <<
      return 1;
    }

#endif // NS3_MPI

  // System id of Wifi side
  uint32_t systemWifi = 0;

  // System id of CSMA side
  uint32_t systemCsma = systemCount - 1;

  NodeContainer p2pNodes;
  Ptr<Node> p2pNode1 = CreateObject<Node> (systemWifi); // Create node with r
  Ptr<Node> p2pNode2 = CreateObject<Node> (systemCsma); // Create node with rank
  p2pNodes.Add (p2pNode1);
  p2pNodes.Add (p2pNode2);

  PointToPointHelper pointToPoint;
```

Example

examples/tutorial/third.cc

```
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
```

1. Create CSMA nodes on one rank
2. Create Wifi nodes on another rank

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);

NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);

CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);

NodeContainer wifiStaNodes;
wifiStaNodes.Create (nWifi);
NodeContainer wifiApNode = p2pNodes.Get (0);

YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());

WifiHelper wifi = WifiHelper::Default ();
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();

Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));

NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, mac, wifiStaNodes);

mac.SetType ("ns3::ApWifiMac",
```

src/mpi/examples/third-distributed.cc

```

NodeContainer p2pNodes;
Ptr<Node> p2pNode1 = CreateObject<Node> (systemWifi); // Create node with ra
Ptr<Node> p2pNode2 = CreateObject<Node> (systemCsma); // Create node with rank 1
p2pNodes.Add (p2pNode1);
p2pNodes.Add (p2pNode2);

PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);

NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma, systemCsma);  // Create csma nodes with rank 1

CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);

NodeContainer wifiStaNodes;
wifiStaNodes.Create (nWifi, systemWifi); // Create wifi nodes with rank 0
NodeContainer wifiApNode = p2pNodes.Get (0);

YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());

WifiHelper wifi = WifiHelper::Default ();
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();

Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));

NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, mac, wifiStaNodes);

mac.SetType ("ns3::ApWifiMac",
```

1

2

Example

examples/tutorial/third.cc

```
stack.Install (csmaNodes);

Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);

address.SetBase ("10.1.3.0", "255.255.255.0");
address.Assign (staDevices);
address.Assign (apDevices);

UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));

UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps =
  echoClient.Install (wifiStaNodes.Get (nWifi - 1));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

Simulator::Stop (Seconds (10.0));

if (tracing == true)
  {
    pointToPoint.EnablePcapAll ("third-distributed-wifi");
    phy.EnablePcap ("third-distributed-wifi", apDevices.Get (0));
    csma.EnablePcap ("third-distributed-wifi", csmaDevices.Get (0), true);

    pointToPoint.EnablePcapAll ("third-distributed-csma");
    phy.EnablePcap ("third-distributed-csma", apDevices.Get (0));
```

1. Install devices, addresses and Internet stack everywhere
2. Install applications only on rank-local nodes

src/mpi/examples/third-distributed.cc

```
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);

address.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);

address.SetBase ("10.1.3.0", "255.255.255.0");
address.Assign (staDevices);
address.Assign (apDevices);

// If this rank is systemCsma,
// it should contain the server application,
// since it is on one of the csma nodes
if (systemId == systemCsma)
  {
    UdpEchoServerHelper echoServer (9);

    ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma))
    serverApps.Start (Seconds (1.0));
    serverApps.Stop (Seconds (10.0));
  }

// If this rank is systemWifi
// it should contain the client application,
// since it is on one of the wifi nodes
if (systemId == systemWifi)
  {
    UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
    echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
    echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
    echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

    ApplicationContainer clientApps =
      echoClient.Install (wifiStaNodes.Get (nWifi - 1));
    clientApps.Start (Seconds (2.0));
    clientApps.Stop (Seconds (10.0));
  }

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

Simulator::Stop (Seconds (10.0));

if (tracing == true)
  {
```

Example

examples/tutorial/third.cc

```
  address.SetBase ("10.1.1.0", "255.255.255.0");

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps =
    echoClient.Install (wifiStaNodes.Get (nWifi - 1));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

  Simulator::Stop (Seconds (10.0));

  if (tracing == true)
    {
      pointToPoint.EnablePcapAll ("third-distributed-wifi");
      phy.EnablePcap ("third-distributed-wifi", apDevices.Get (0));
      csma.EnablePcap ("third-distributed-wifi", csmaDevices.Get (0), true);

      pointToPoint.EnablePcapAll ("third-distributed-csma");
      phy.EnablePcap ("third-distributed-csma", apDevices.Get (0));
      csma.EnablePcap ("third-distributed-csma", csmaDevices.Get (0), true);
    }

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

1. Enable PCAP tracing on local nodes?
2. Close MPI cleanly

src/mpi/examples/third-distributed.cc

```
    {
      UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
      echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
      echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
      echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

      ApplicationContainer clientApps =
        echoClient.Install (wifiStaNodes.Get (nWifi - 1));
      clientApps.Start (Seconds (2.0));
      clientApps.Stop (Seconds (10.0));
    }

  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

  Simulator::Stop (Seconds (10.0));

  if (tracing == true)
    {
      // Depending on the system Id (rank), the pcap information
      // traced will be different.  For example, the ethernet pcap
      // will be empty for rank0, since these nodes are placed on
      // on rank 1.  All ethernet traffic will take place on rank 1.
      // Similar differences are seen in the p2p and wireless pcaps.
      if (systemId == systemWifi)
        {
          pointToPoint.EnablePcapAll ("third-distributed-wifi");
          phy.EnablePcap ("third-distributed-wifi", apDevices.Get (0));
          csma.EnablePcap ("third-distributed-wifi", csmaDevices.Get (0), true);
        }
      else // systemCsma
        {
          pointToPoint.EnablePcapAll ("third-distributed-csma");
          phy.EnablePcap ("third-distributed-csma", apDevices.Get (0));
          csma.EnablePcap ("third-distributed-csma", csmaDevices.Get (0), true);
        }
    }

  Simulator::Run ();
  Simulator::Destroy ();

#ifdef NS3_MPI
  // Exit the MPI execution environment
  MpiInterface::Disable ();
#endif

  return 0;
}
```

Script Output–Identical

```
$ ./waf –run third
Waf: Entering directory `build/debug'
Waf: Leaving directory `build/debug'
'build' finished successfully (2.152s)
At time 2s client sent 1024 bytes to 10.1.2.4 port 9
At time 2.01796s server received 1024 bytes from 10.1.3.3 port 49153
At time 2.01796s server sent 1024 bytes to 10.1.3.3 port 49153
At time 2.03364s client received 1024 bytes from 10.1.2.4 port 9
```

```
$ ./waf --run third-distributed \
--command-template="mpirun -n 2 %s --tracing"
Waf: Entering directory `build/debug'
Waf: Leaving directory `build/debug'
'build' finished successfully (2.050s)
At time 2s client sent 1024 bytes to 10.1.2.4 port 9
At time 2.01796s server received 1024 bytes from 10.1.3.3 port 49153
At time 2.01796s server sent 1024 bytes to 10.1.3.3 port 49153
At time 2.03364s client received 1024 bytes from 10.1.2.4 port 9
```

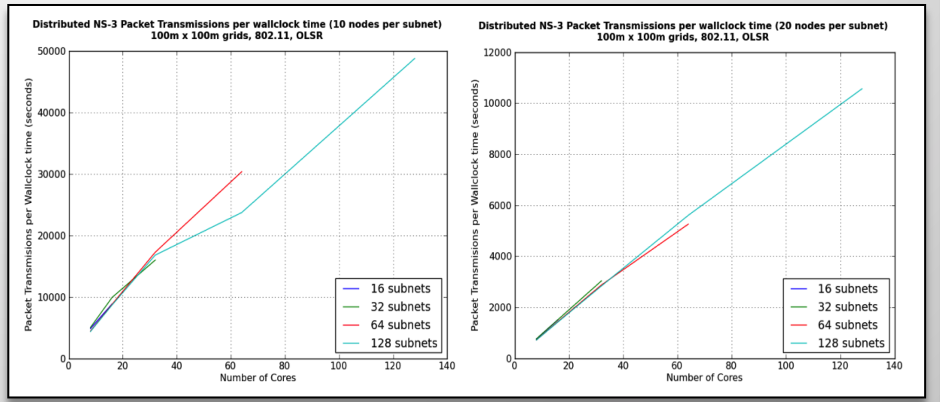
Cryptic Error Conditions

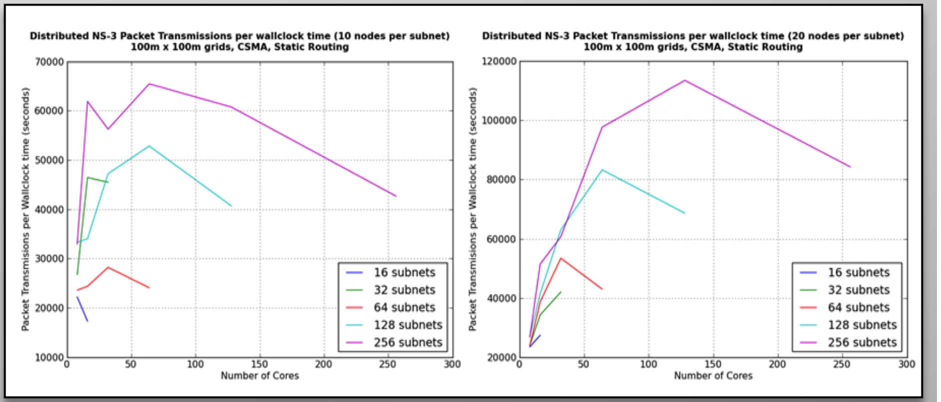
- `Can't use distributed simulator without MPI compiled in`
  - Not finding or building with MPI libraries
  - ➢Reconfigure NS-3 and rebuild

- `assert failed. cond="pNode && pMpiRec", file=../src/mpi/model/mpi-interface.cc, line=413`
  - Mis-aligned node or interface IDs

Performance Optimizations

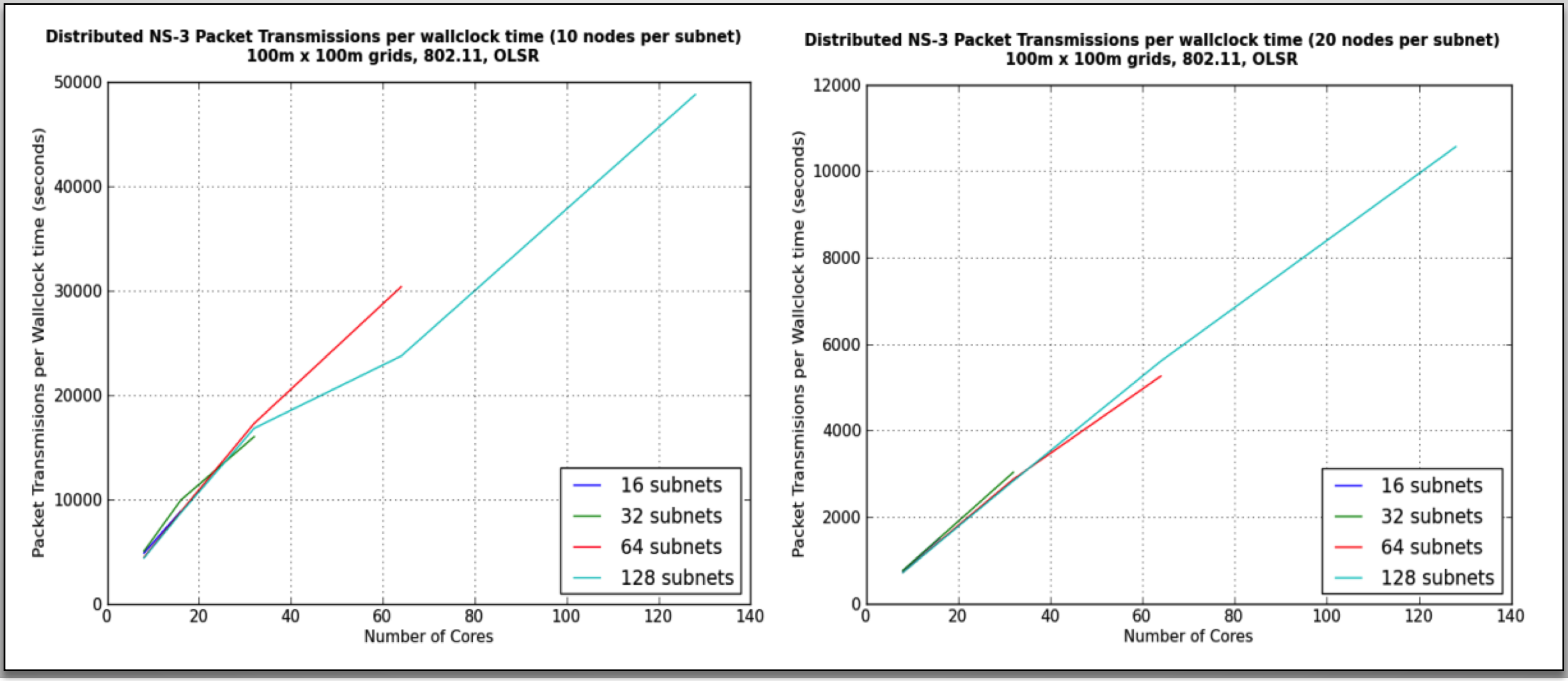
- Larger Lookahead
- Synchronization cost grows exponentially with LP count
  - More work per LP is better
  - Speed gains up to $10^{2\text{-}3}$ ranks, depending on model
- Appropriate performance metric
  - Events/sec can be misleading with varying event cost
  - Packet transmissions (or receives) per wall-clock time





Parallel Performance with Large Computation Load: 802.11+OLSR

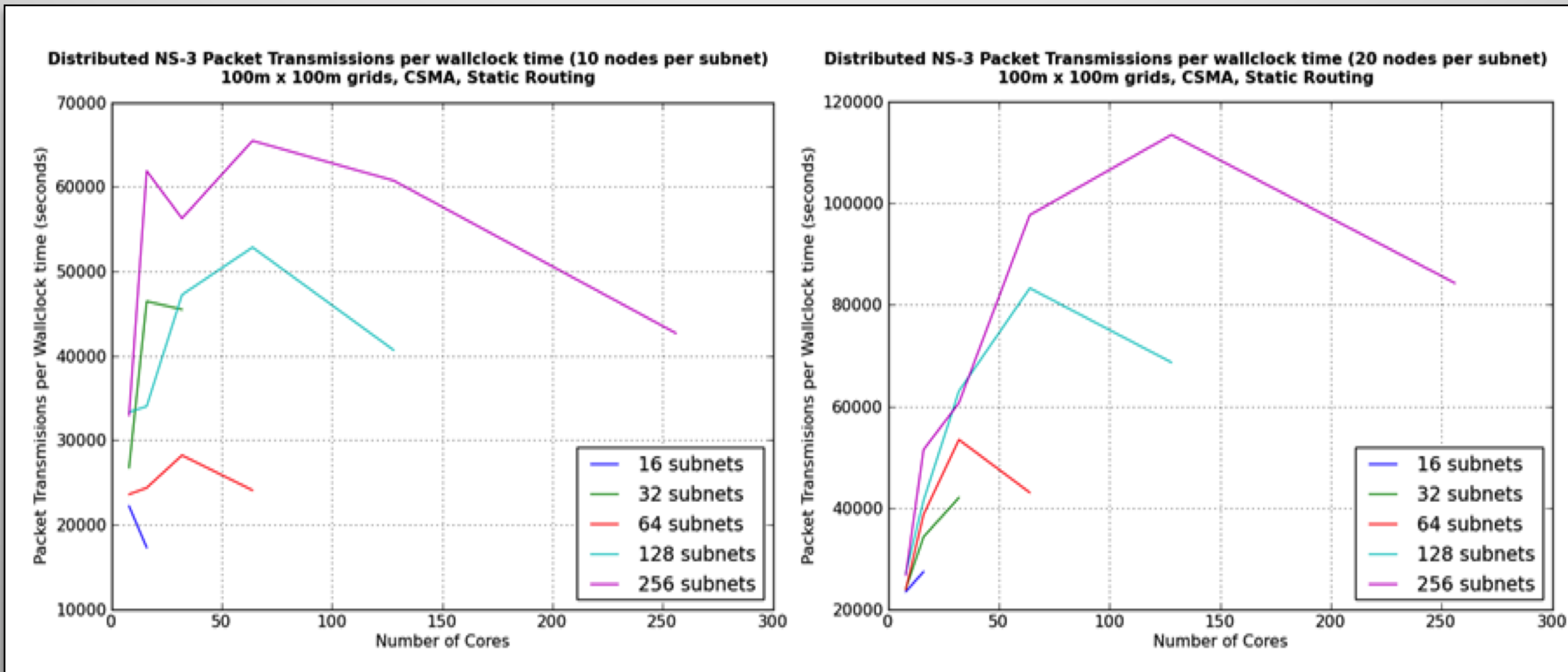
- Linear scaling out to 128 ranks



Performance

Parallel Performance with Small Computation Load: CSMA+Static

- Performance drops at modest number of ranks



ns-3

Work in Progress

- Automatic memory scaling
  - Automatic ghost nodes, globally unique node IDs
  - (See my talk tomorrow ☺)
- Automatic partitioning, ghost alignment
- Distributed Real Time
  - Versus simultaneous real-time emulations:
    - LP-to-LP messaging gives greater *Lookahead* than independent ns-3 instances connected by emulated network devices
- Scalable default routing
  - AS-like routing between LPs
  - Scalable replacement for GOD or Nix-vector routing with ghost nodes

(Mostly) Parallel Partitioning Tools