SAND2010-5718P

# Exascale Co-design with Sandia's Structural Simulation Toolkit (SST)

**Problem**

- Exascale computing has been identified as a critical enabling technology for meeting numerous national challenges
- The nature of computing will change dramatically: billions of threads, faults, energy to solution, and time to solution are all critical issues
- Applications must be adapted before machines are available
  - Cost of adapting applications is high
- High cost of machines will require lower design margins
  - Provide what the application need but no more
  - Coarse measures such as bandwidth per FLOPS not sufficient

**Solution**

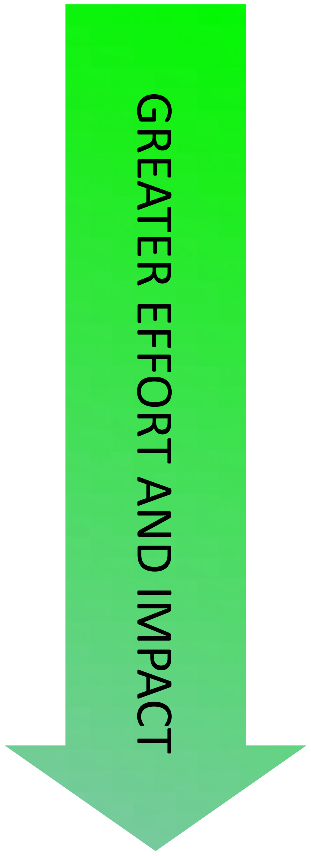
- The SST project is creating a multi-scale computer architecture simulator that will be used in the design and procurement of large-scale parallel machines as well as in the design of algorithms for these machines. Our goal is to become the preeminent simulator for performance prediction and analysis of applications running on large-scale machines
- Facilitate a consortium of industry, government, and academia to implement components and use them

**Impact Measures**

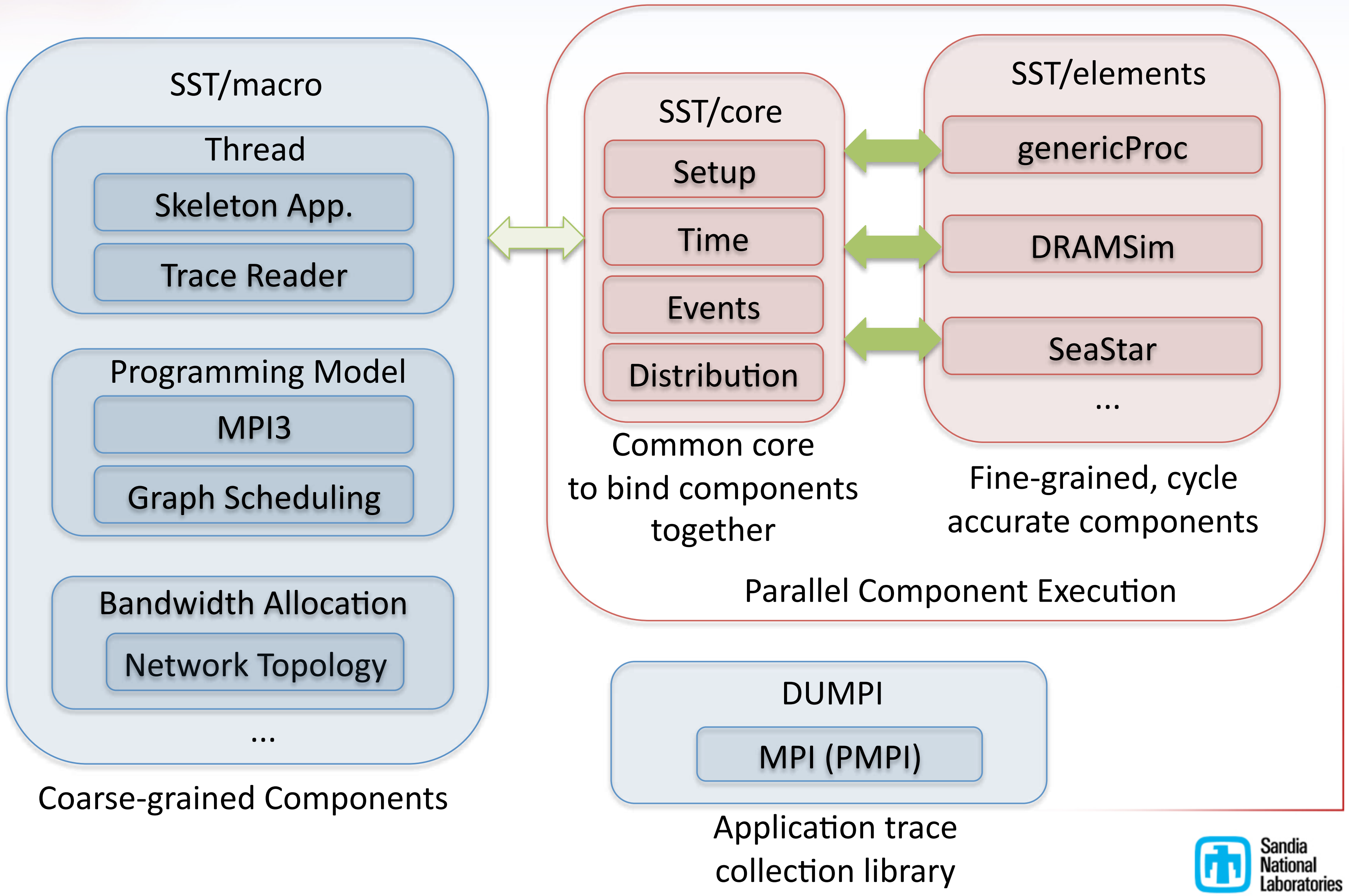
- ASC and ECDC applications team use SST to evaluate algorithms and programming models and act on results
- Leadership machine bids evaluated using SST
- Hardware vendors improve architecture in response to SST studies

# Examples of SST usage and impact

| **SST Impact** | **SST Activity** |
|---|---|
| Develop acceptance tests and estimate performance before machine is built. | Obtain traces for applications and compact applications. Use SST for parameter studies. |
| Understand performance and issues for machines several years from deployment. | Write skeleton applications for extreme scale studies. |
| Allow co-design of advanced architectures and applications many years before deployment. | Simulate new architectural feature such as extended memory semantics and transactional memory. |

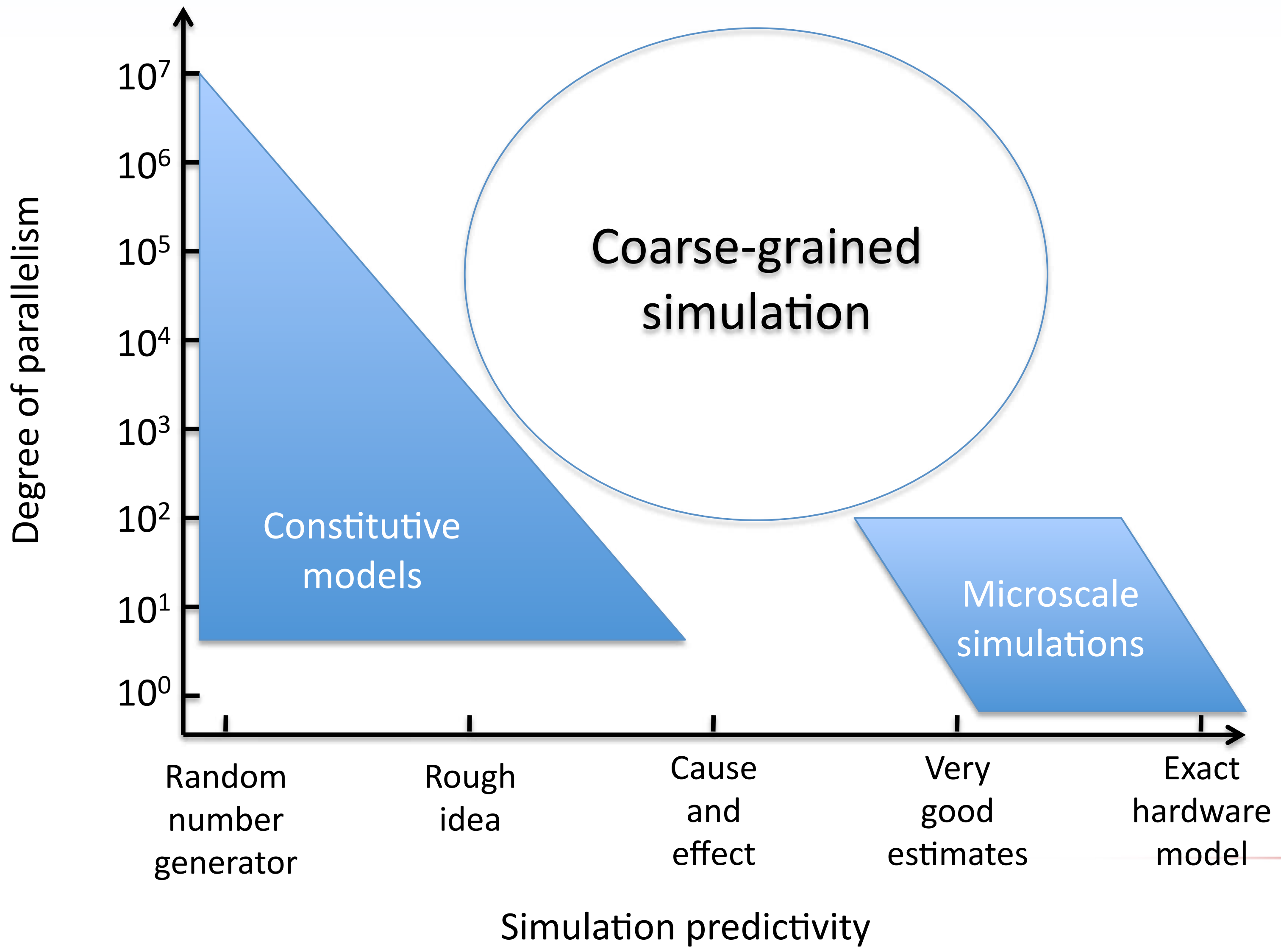


# SST System Architecture



Sandia National Laboratories

## SST/macro target application space



## SST/macro design goals

- Correctly identify causal relationships
  - Discrete inputs (network topology, node configuration, etc.)
  - Continuous inputs (noise/imbalance, bandwidth, latency, …)
  - Emergent properties (resource contention, middleware behavior, …)
- Play “what if” games
  - Interpolating and extrapolating runtime behavior
  - Implementation effects for communication routines
  - Multiple networks, perfect/”magic” operations
- Test changes to application, middleware, or resource management
  - Reordering code blocks, scheduling effects, etc.
- Test novel programming models
  - Fault-tolerant or fault-oblivious execution models
  - Alternatives to MPI, parallel runtime designs
  - Mixed programming models

Sandia National Laboratories

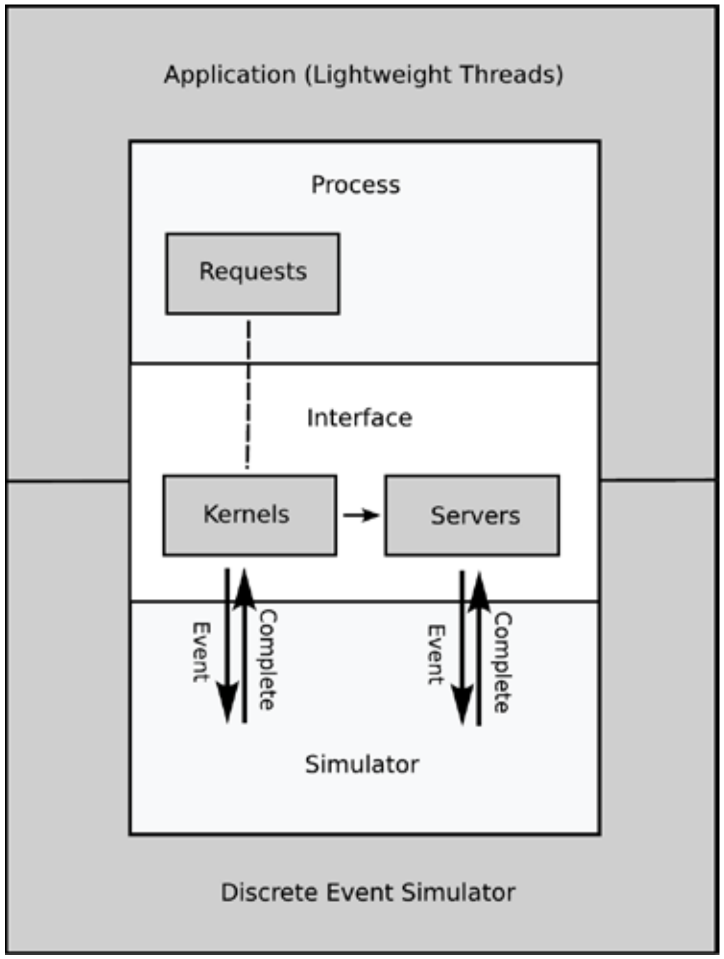
# SST/macro approach

- To meet our design goals, we need:
  - A network model for resource contention
  - Very fast processor and memory models
  - Realistic middleware models (e.g. MPI)
  - Support for complex application models (no direct manipulation of state machines)
- We don't need the exact answer, but we need the correct macroscale behaviors
  - … with a deterministic simulator
  - … which must be reasonably fast
    - … for a wide range of simulation system sizes ($10^1$-$10^6$ nodes)
    - … and realistic application workloads ($10^6$-$10^9$ MPI messages)

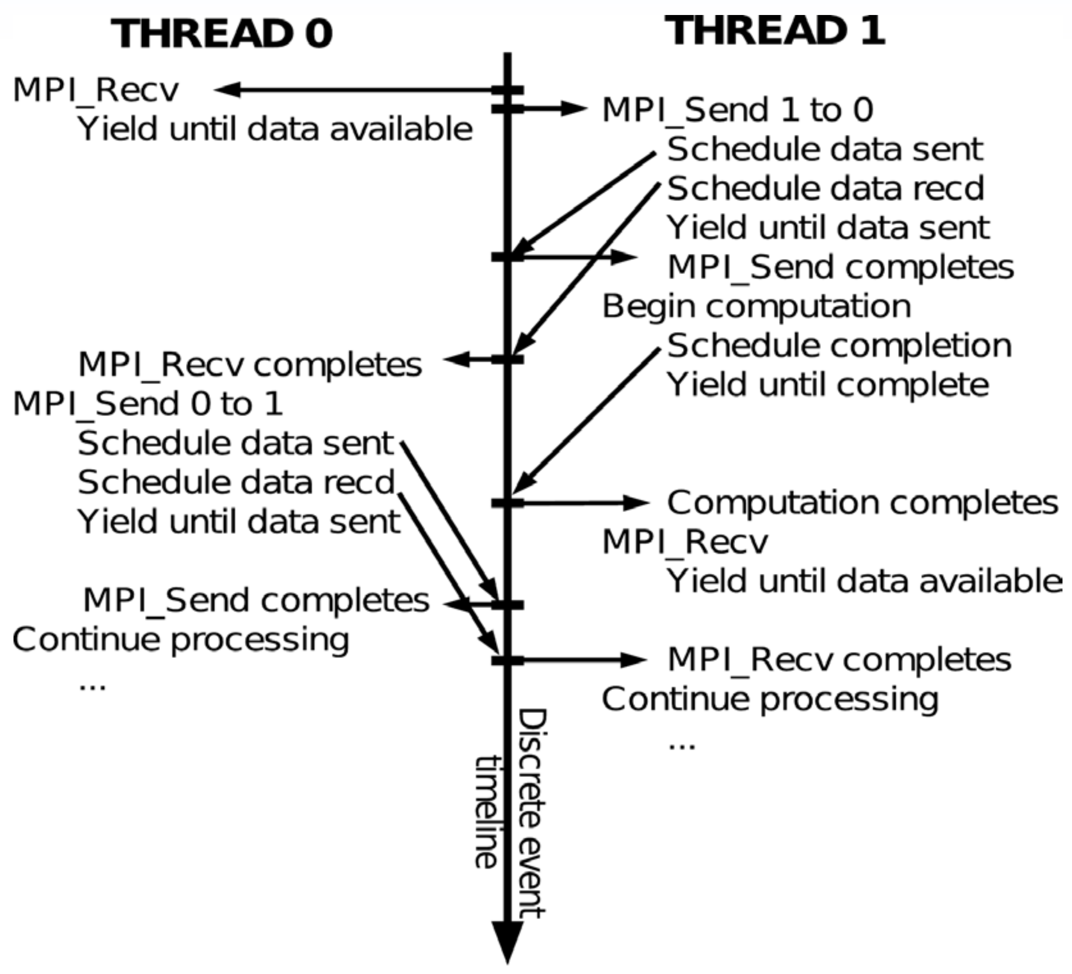
Sandia National Laboratories

# High-level simulator design

- “Process” holds application process state
  - User-space threads
  - Private stacks
- “Interface” bridges threads and DES
  - Context switching for blocking calls
  - Management of active/ blocked queues
- “Simulator” handles DES
  - Single-threaded (lock-free)

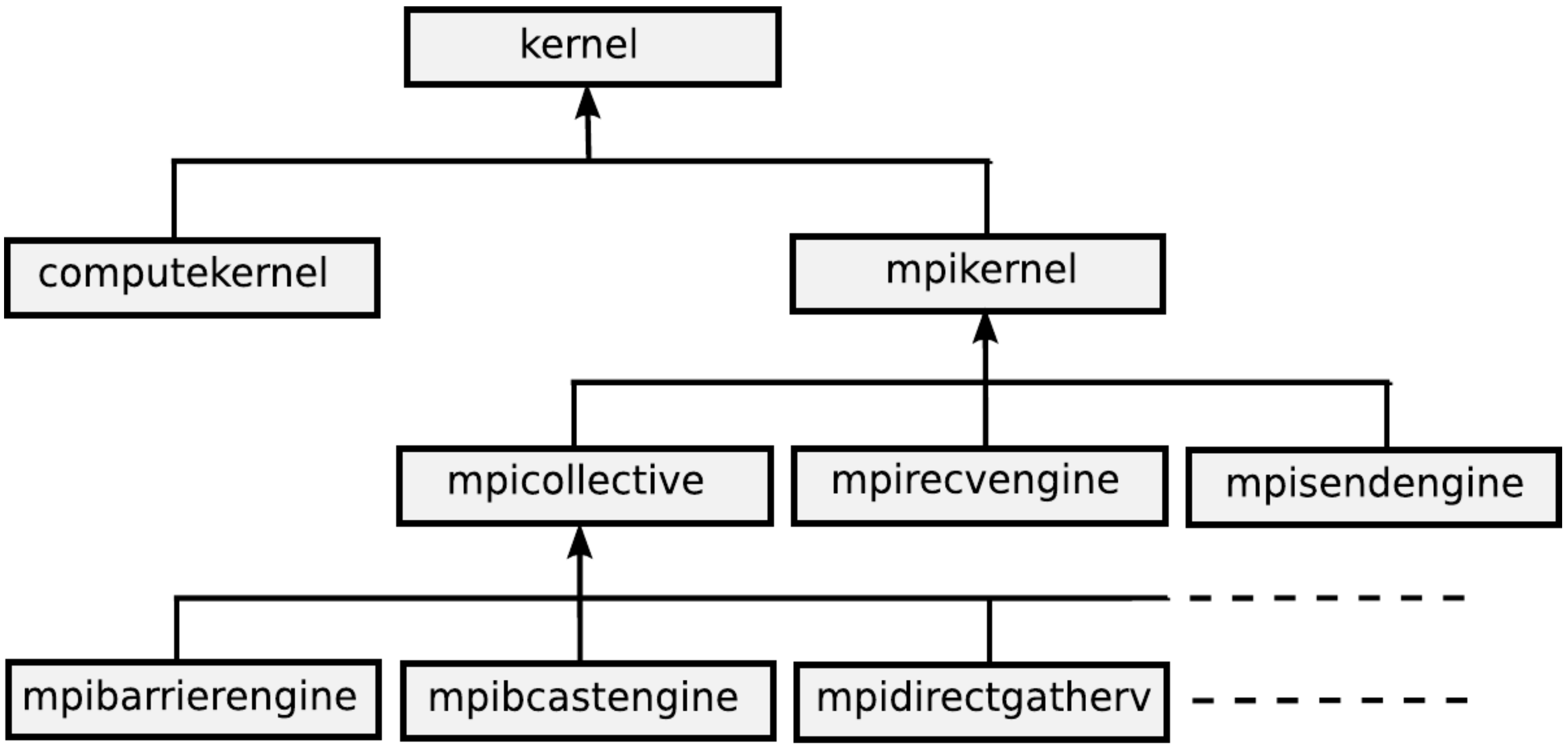


# Example process control flow

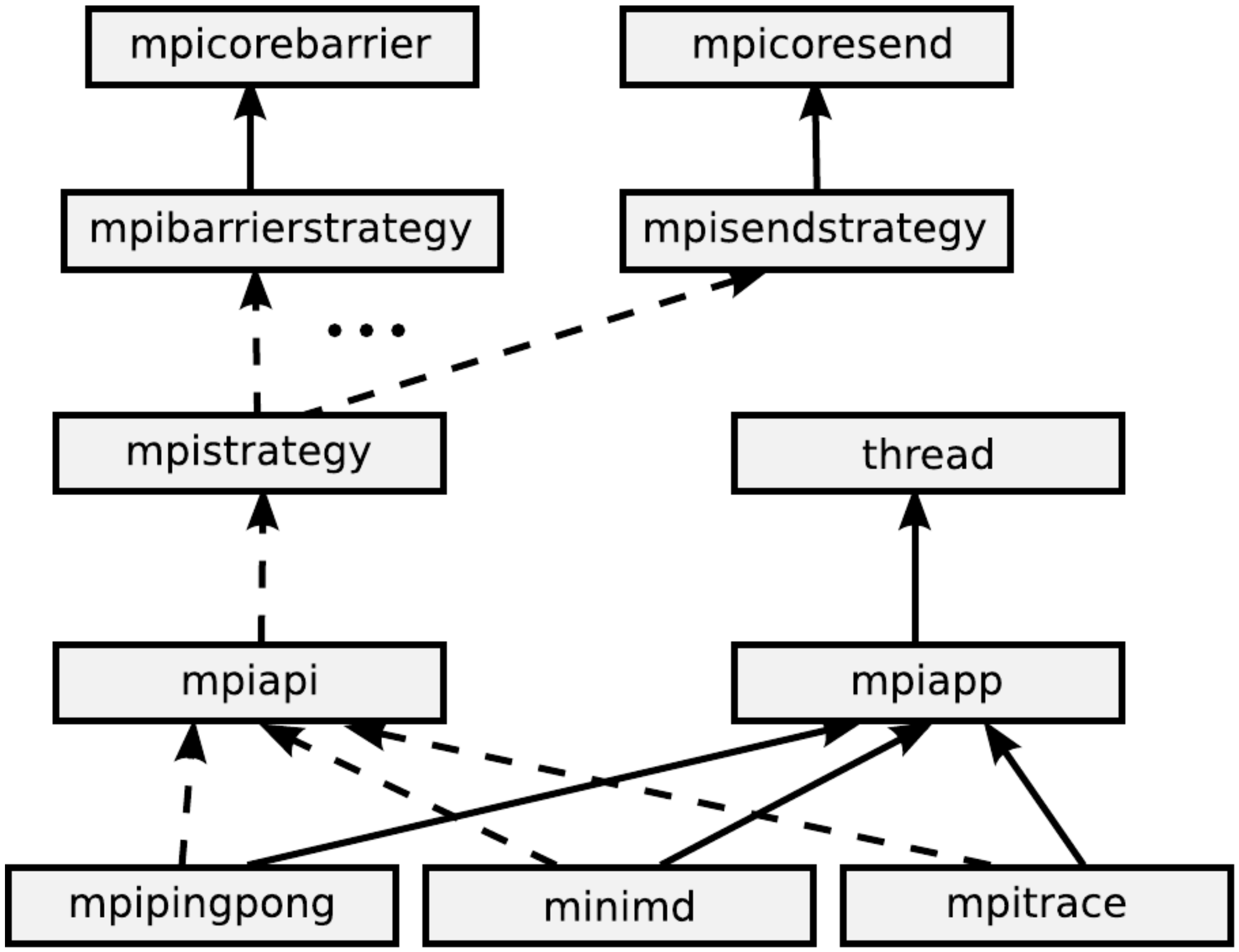


Two simulated processes exchanging data via MPI send/recv pairs

# Example inheritance diagram

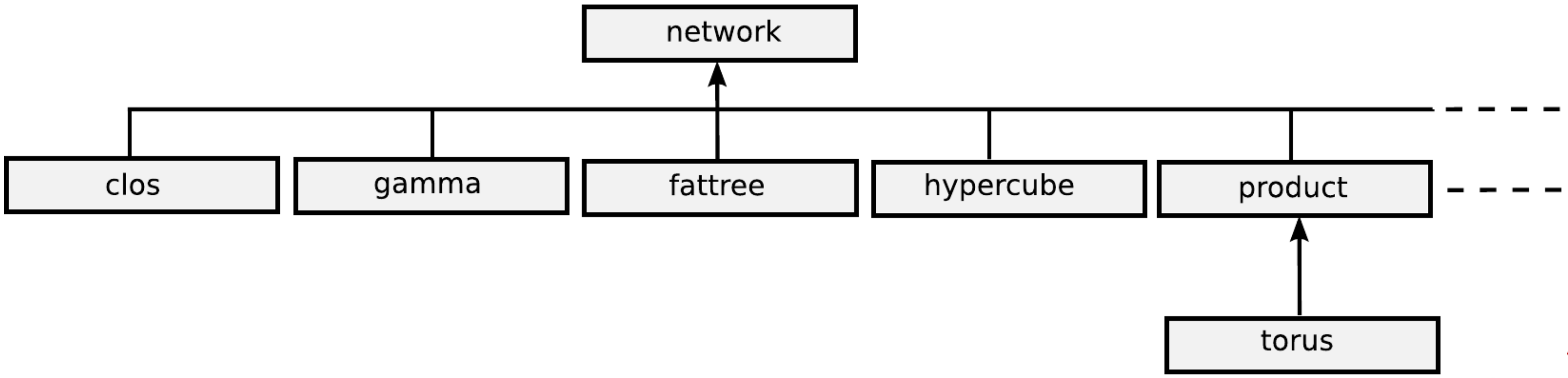


# Partial collaboration diagram for MPI

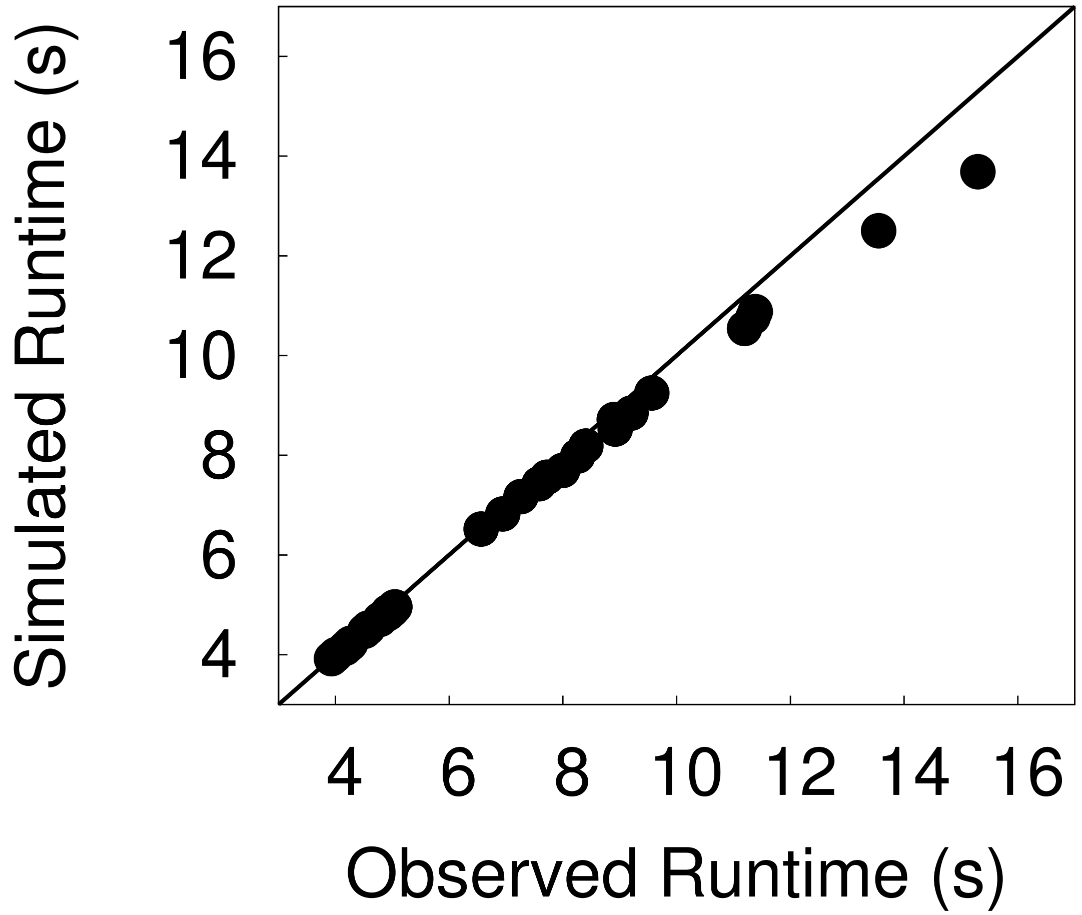


# Networks

- Currently supports simplified network routing/ sharing
  1. Contention-free
  2. Circuit with fair sharing for concurrent flows
- Multiple topologies supported

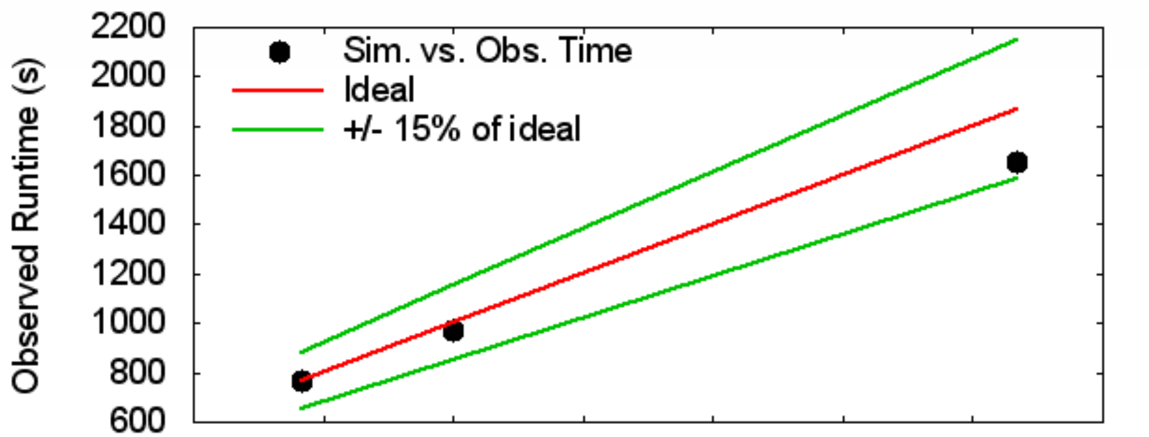


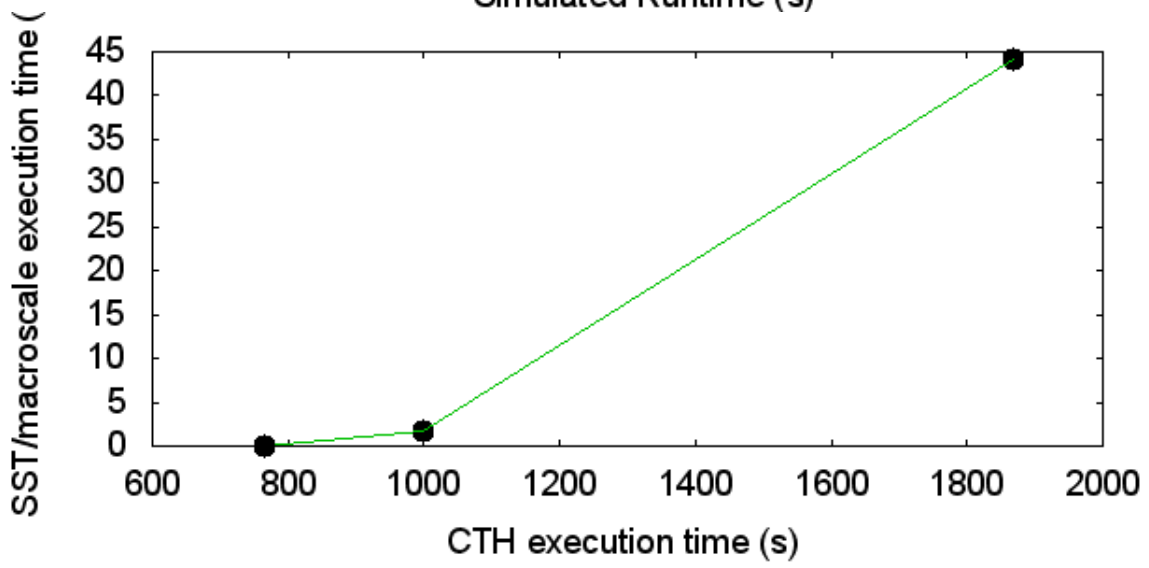
## MPI trace example:  AMG2006



AMG2006 on a variety of node counts and decompositions.

# MPI trace example: CTH





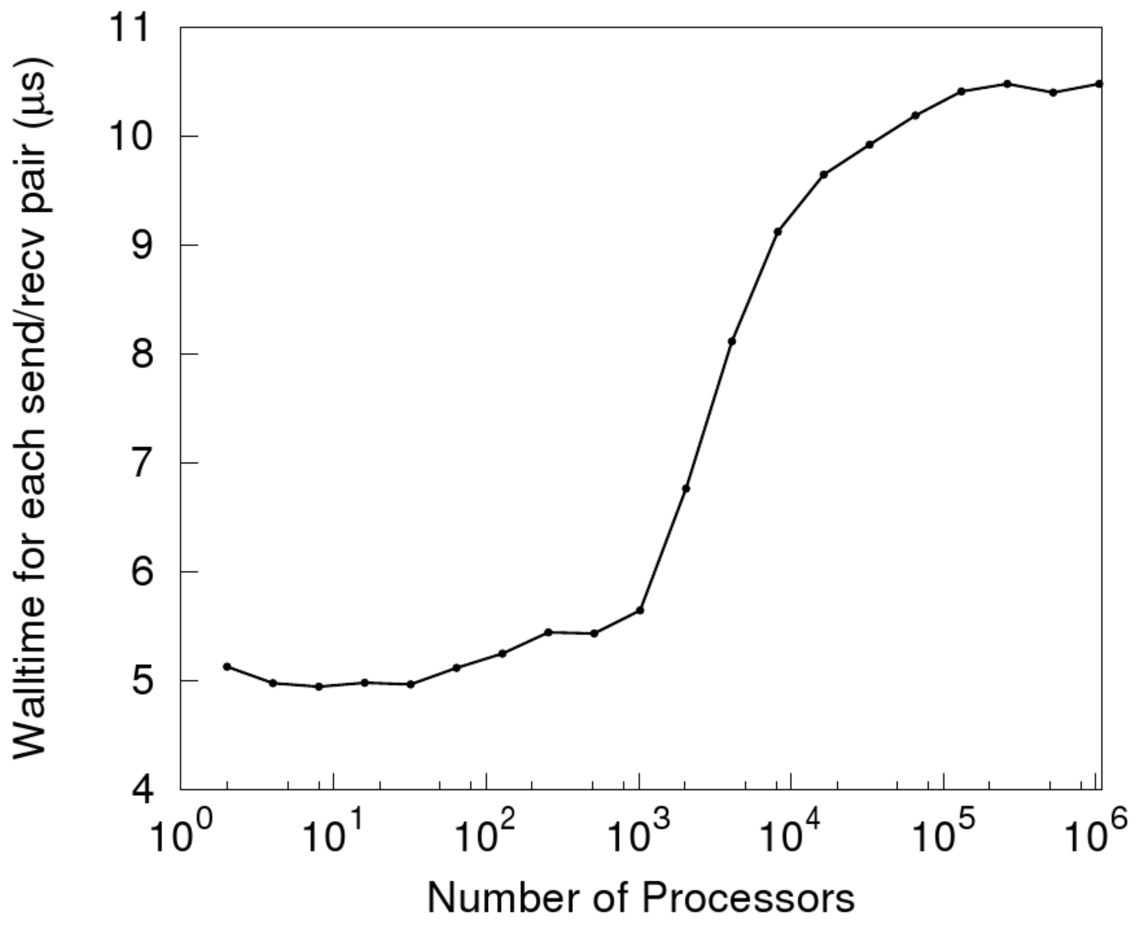
CTH runs on 1, 2, and 16 nodes courtesy of Courtenay Vaughan

Sandia National Laboratories

# Skeleton applications

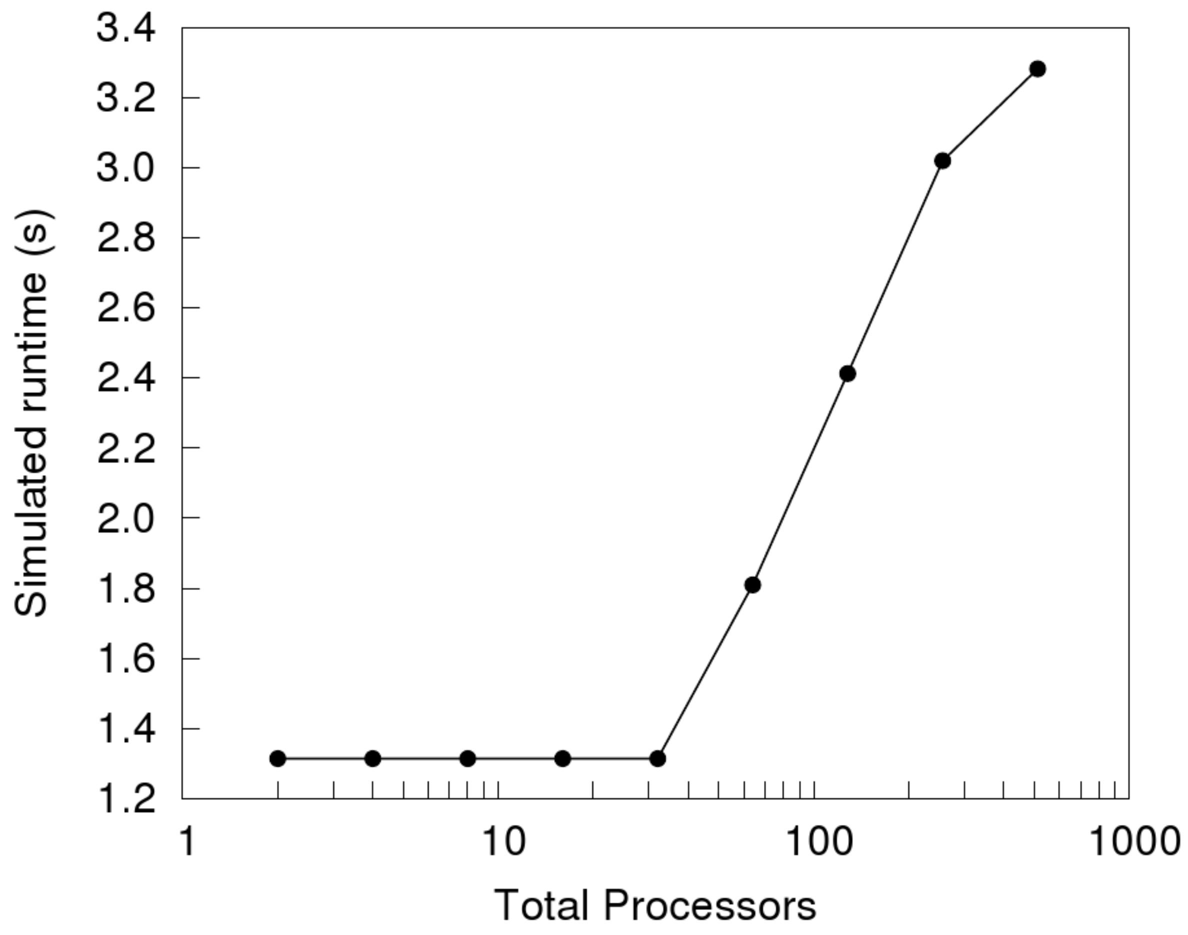
```
void mpipingpong::run() {
   this->mpi_->init();
   mpicomm world = this->mpi_->comm_world();
   mpitype type = mpitype::mpi_double;
   int rank = world.rank().id;
   int size = world.size().id;
   if(! ((size % 2) && (rank+1 >= size))) {
      // With an odd number of nodes, rank (size-1) sits out
      mpiid peer(rank ^ 1); // partner nodes 0<=>1, 2<=>3, etc.
      mpiapi::const_mpistatus_t stat;
      for(int half_cycle = 0; half_cycle < 2*niter; ++half_cycle) {
         if((half_cycle + rank) & 1)
            mpi_->send(count_, type, peer, mpitag(0), world);
         else
            mpi_->recv(count_, type, peer, mpitag(0), world, stat);
      }
   }
   mpi_->finalize();
}
```

# Simulator performance



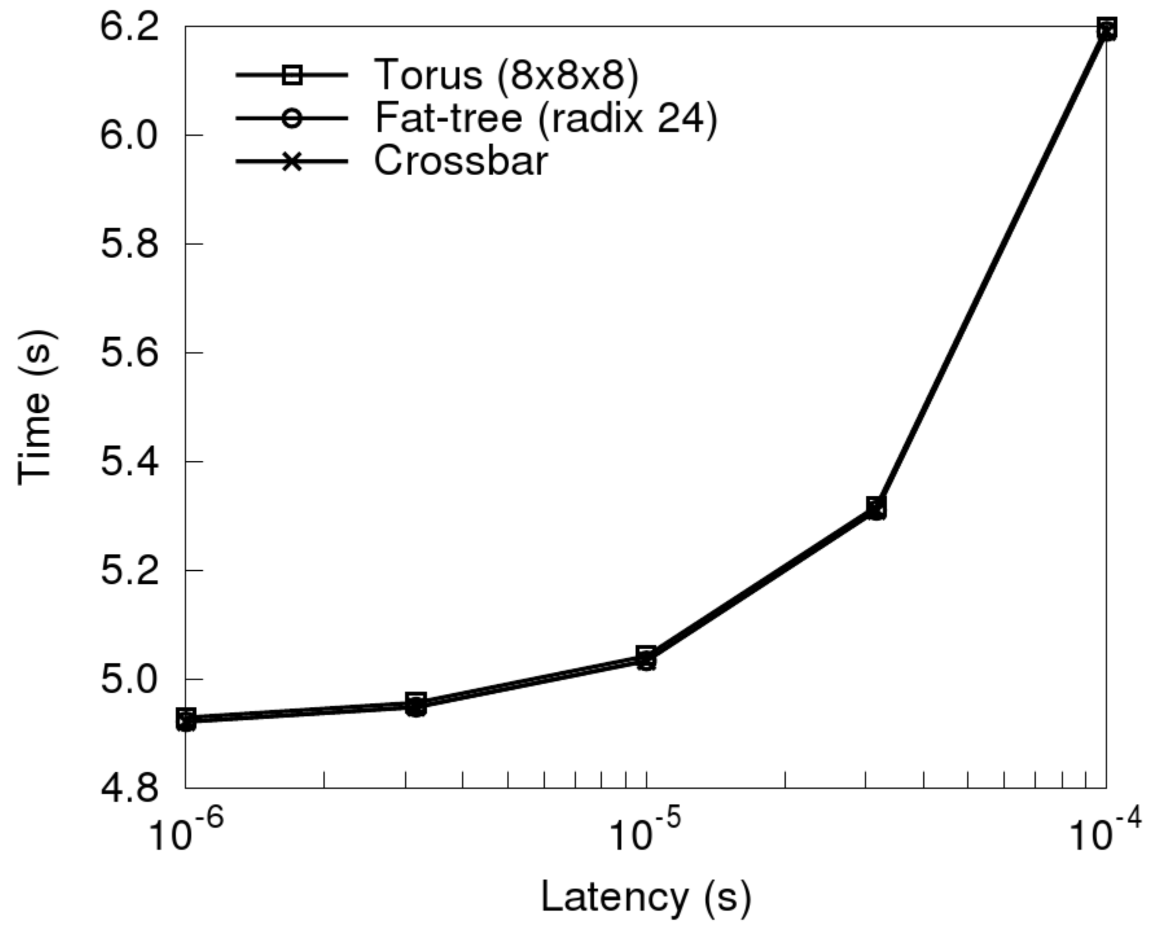
MPI ping-pong on a contention-free network. Total of 4M messages.

## Simulated traffic congestion



MPI ping-pong on fat-tree network. Fixed total of 65536 messages.

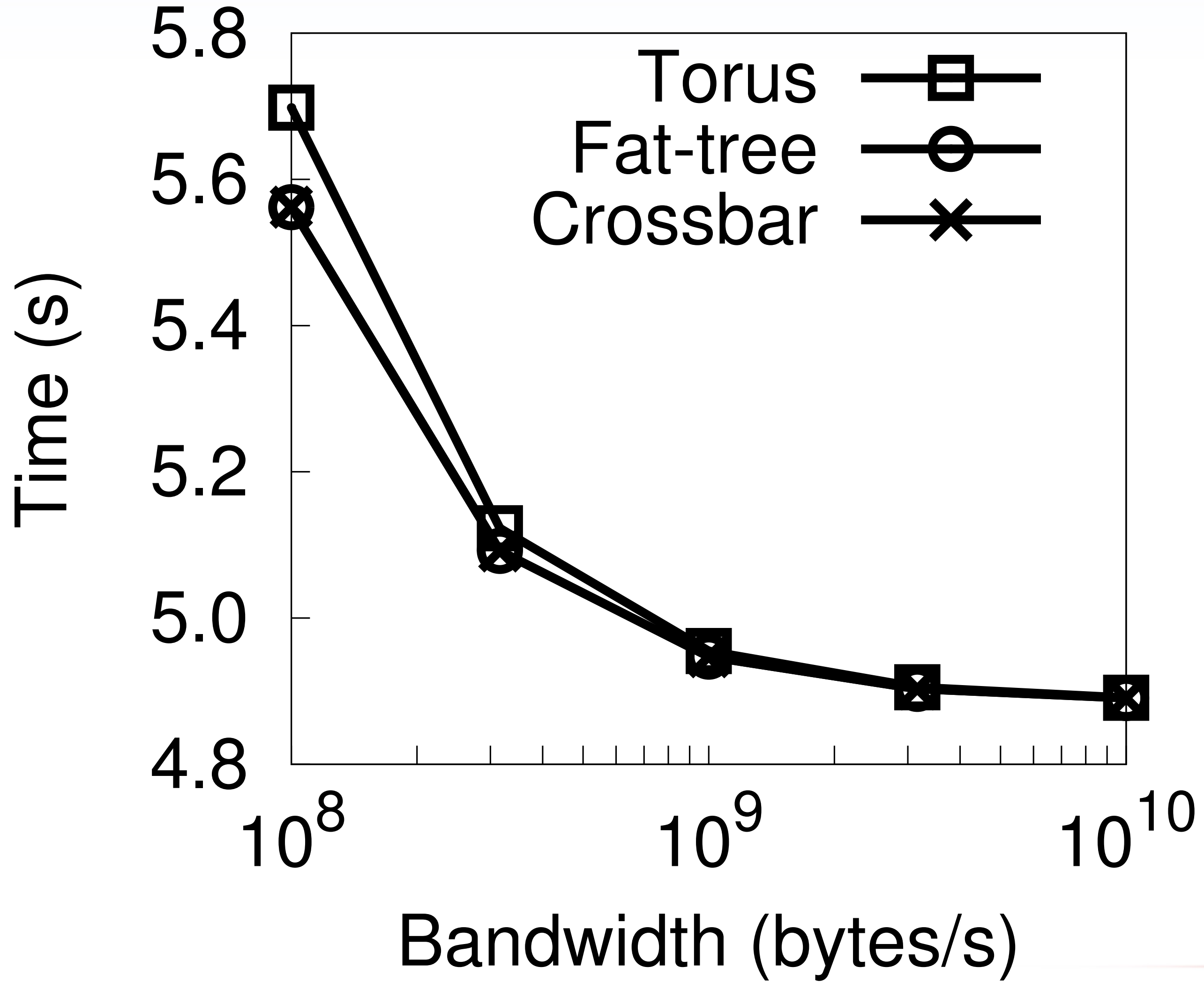
# Example parameter sweep:  Latency



AMG2006 on 128 nodes. Bandwidth constant at 1 GB/s.

Sandia National Laboratories

## Example parameter sweep: Bandwidth



AMG2006 on 128 nodes. Latency constant at 3 µs.

## DUMPI: The MPI tracer

- PMPI link-time library for trace file generation
- Full fingerprints for all MPI-2 functions
- Writes a (reasonably compact) binary trace file
- Negligible runtime overhead
- Reasonably portable C code

| libdumpi | common | libundumpi |
|---|---|---|
| PMPI bindings<br>Type mapping<br>Call tree tracing (gcc/icc) | MPI type identifiers<br>MPI function identifiers<br>Trace file IO<br>Timers<br>Performance counters | Parsing of trace files |

## Example of data output by DUMPI

(converted using dumpi2ascii)

```
MPI_Allgatherv entering at walltime 1274314439.744512000,        \
cputime 0.201756000 seconds in thread 0.
int commsize=16
int sendcount=1024
MPI_Datatype sendtype=14 (MPI_DOUBLE)
int recvcounts[16]=[1024, 1024, 1024, 1024, 1024, 1024, 1024,    \
1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024, 1024]
int displs[16]=[0, 1024, 2048, 3072, 4096, 5120, 6144, 7168,     \
8192, 9216, 10240, 11264, 12288, 13312, 14336, 15360]
MPI_Datatype recvtype=14 (MPI_DOUBLE)
MPI_Comm comm=2 (MPI_COMM_WORLD)
MPI_Allgatherv returning at walltime 1274314439.749554000,       \
cputime 0.202159000 seconds in thread 0.
```

Sandia National Laboratories

# Callback-driven parsing of DUMPI files

- Each MPI call is assigned a callback function

```
typedef int (*dumpi_allgatherv_call)(const dumpi_allgatherv *prm,
             uint16_t thread, const dumpi_time *cpu, const dumpi_time
*wall,
             const dumpi_perfinfo *perf, void *userarg);
```

- The full set of arguments to the MPI function is passed to the callback in a struct

```
typedef struct dumpi_allgatherv {
    /** Not an MPI argument.  Added to index relevant data in the struct.
*/
    int  commsize;
    /** Argument value before PMPI call */
    int  sendcount;
    /** Argument value before PMPI call */
    dumpi_datatype  sendtype;
    /** Argument value before PMPI call.  Array of length [commsize] */
    int * recvcounts;
    /** Argument value before PMPI call.  Array of length [commsize] */
    int * displs;
    /** Argument value before PMPI call */
    dumpi_datatype  recvtype;
    /** Argument value before PMPI call */
    dumpi_comm  comm;
} dumpi_allgatherv;
```

Sandia National Laboratories

# SST/macro Team Members

**Contributors**

Curtis Janssen (PI)

Helgi Adalsteinsson (Everything)

Scott Cranford (I/O modeling)

Damian Dechev (Skeleton app. generation)

David Evensky (Data collection, param. studies)

Joseph Kenny (Proc. models, prog models)

Jackson Mayo (Interested observer)

Ali Pinar (Network models)

Sandia National Laboratories