BigSim Tutorial

Celso L. Mendes - Ryan Mokos

Department of Computer Science
University of Illinois at Urbana-Champaign

{cmendes,mokos}@illinois.edu
http://charm.cs.uiuc.edu

April 20, 2011

9th Annual Charm++ Workshop

Tutorial Outline

- Part I: Emulation
  - Introduction to BigSim, AMPI
  - Code Conversion to AMPI
  - BigSim Emulator
  - Trace Utilities
- Part II: Simulation
  - BigSim Simulators
  - Network Models
  - Projections Visualization
  - Simulation Statistics

Introduction to BigSim

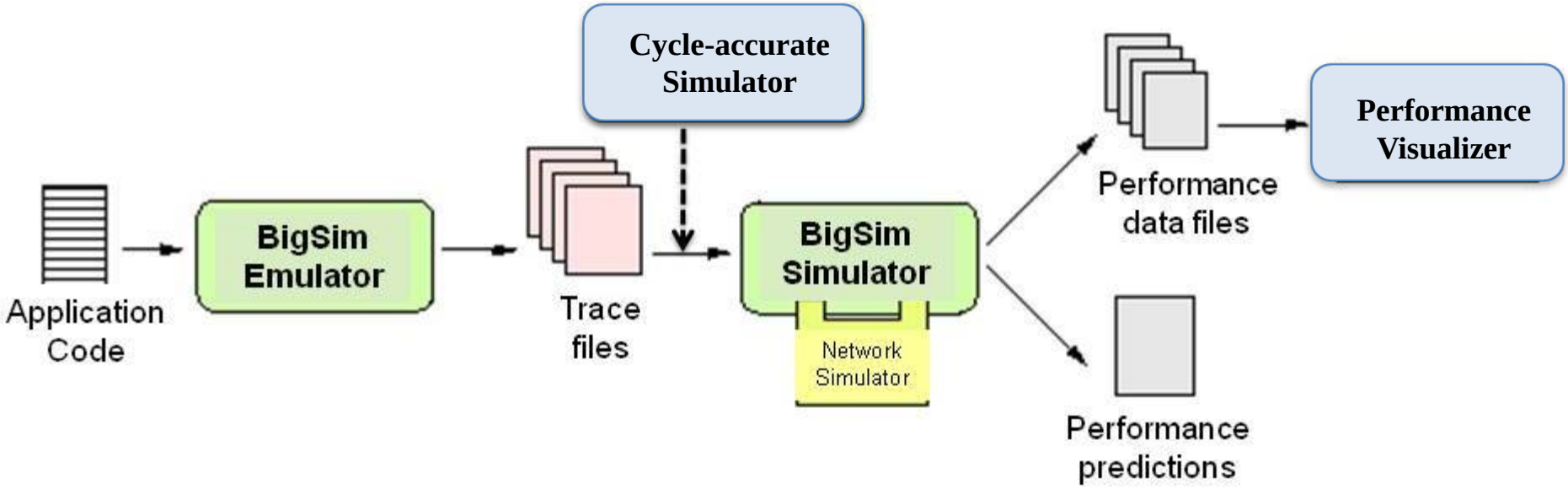
- BigSim Simulation System
  - Goal: simulate application behavior on large machines
  - Focus: find performance bottlenecks
  - History
    - Originally built to simulate early Blue Gene
    - Adapted later for other types of machines
    - Used currently for Blue Waters simulations

Introduction to BigSim

- BigSim's Capabilities:
  - Whole-application simulation for large systems
  - Simulations at varying levels of fidelity
- What BigSim *Cannot* Do:
  - Cycle-accurate simulations of a processor
    - But it *can* use results from such simulators!
  - Model irregular /non-deterministic applications
  - Model cache or virtual memory effects, I/O devices

Introduction to BigSim (cont.)

- BigSim Structure:

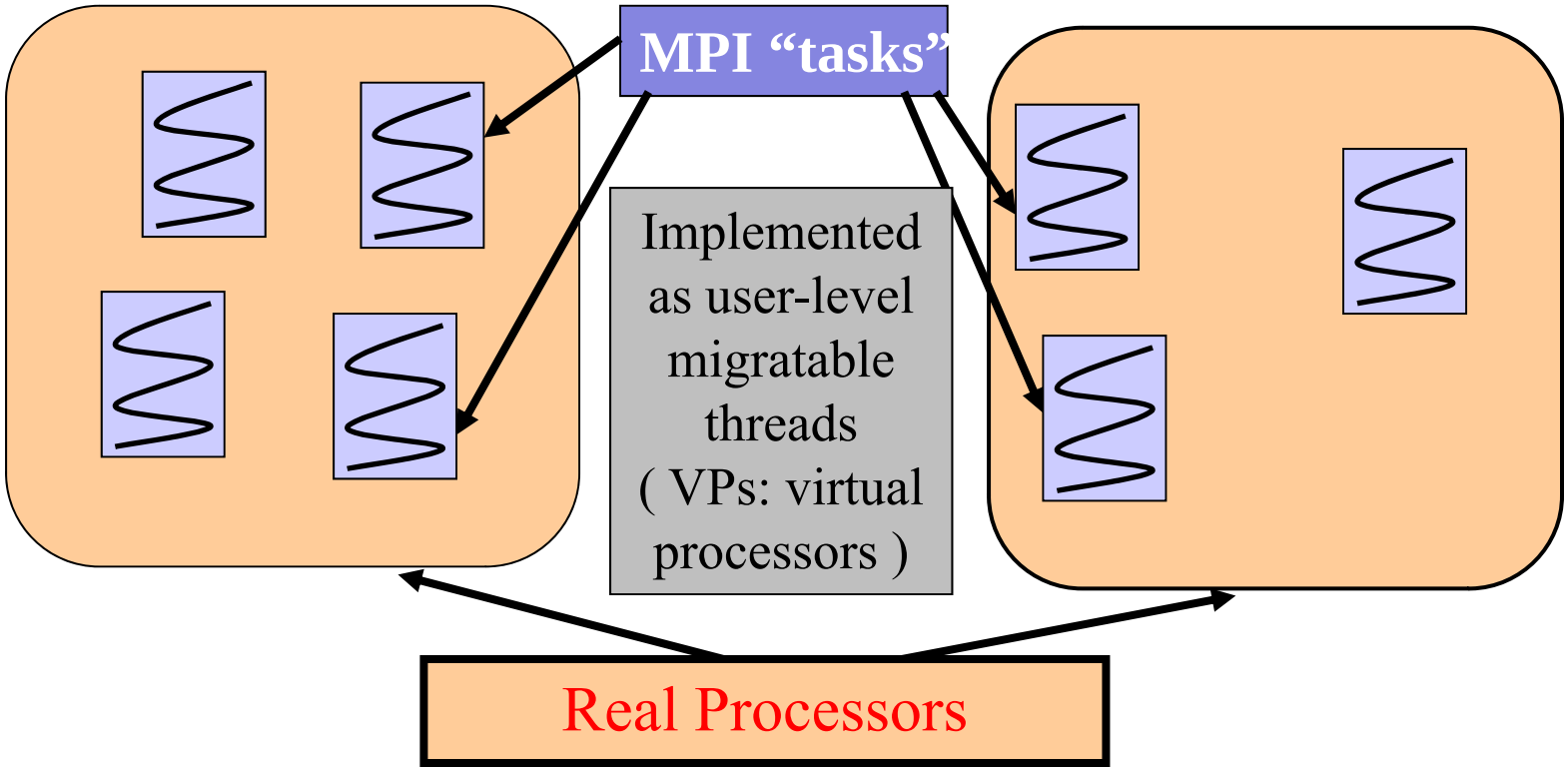


Introduction to BigSim (cont.)

- BigSim: Application Requirements
  - MPI codes:
    - Must be converted to Adaptive MPI (AMPI)
      - i.e. must not have global or static variables
  - Charm++ codes:
    - Must have dependencies manually added
    - Structured-dagger programs are OK

AMPI Overview

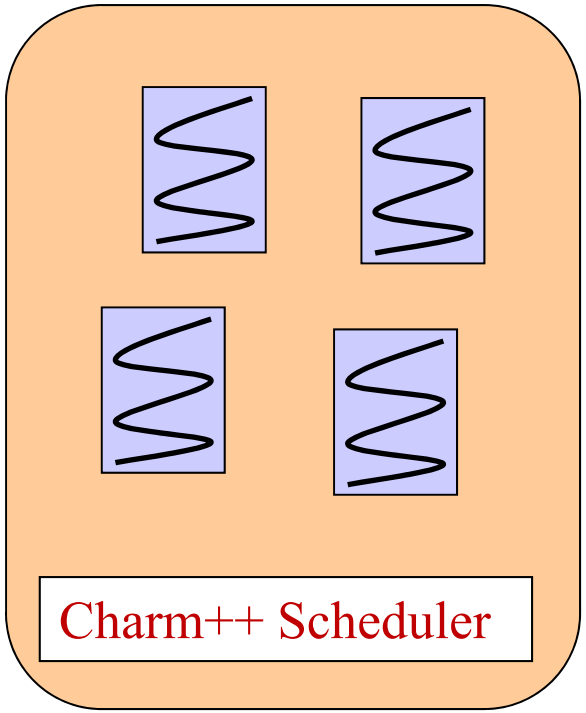
- Virtualization: MPI ranks → Charm++ threads



AMPI Overview (cont.)

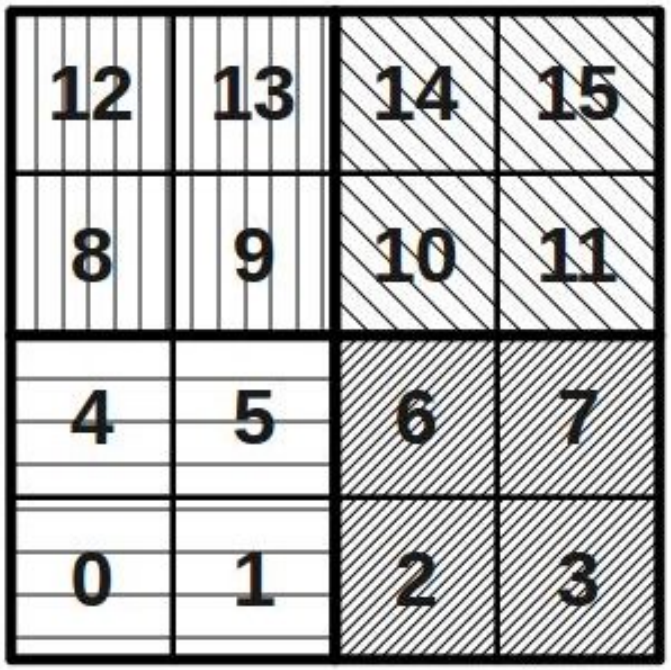
- AMPI Execution Model:
  - Multiple user-level threads per process
  - Typically, one process per physical processor
  - Charm++ Scheduler coordinates execution
  - Virtualization ratio: R = #VP / #P (over-decomposition)

BigSim: each VP represents a processor of target system!



AMPI Overview (cont.)

- Virtualization Example in BigSim Emulation:



- P=4, VP=16
- Emulation is run on P=4 (i.e. on 4 existing processors)
- Simulated system: 16 processors
- MPI program sees 16 “ranks”
- Emulation runs: very similar to regular AMPI executions (with additional generation of traces)

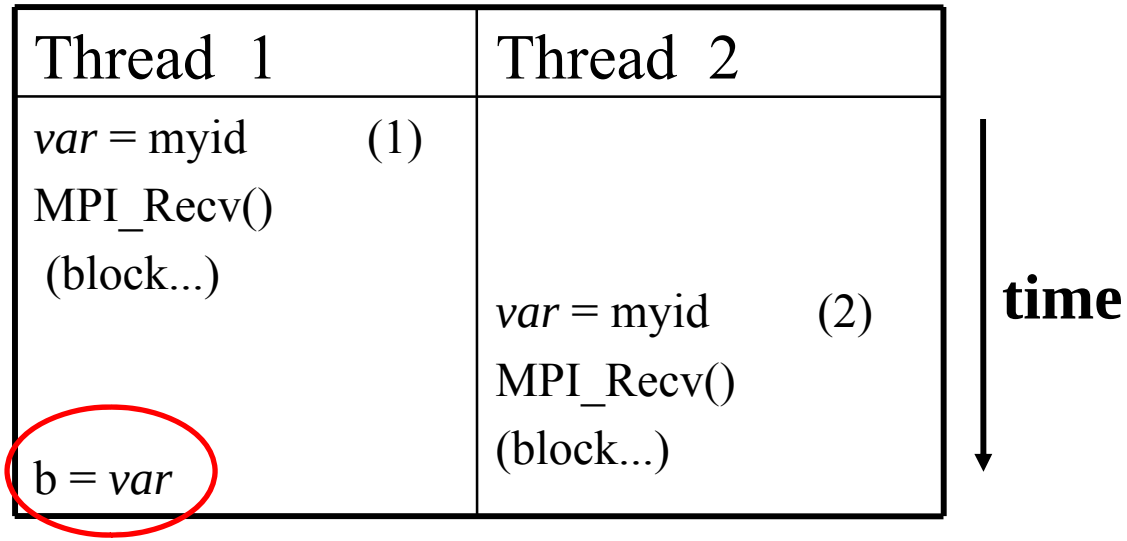
Code Conversion to AMPI

- Fortran codes:
  - Replace `program` by `subroutine MPI_Main`
- C/C++ codes:
  - Just make sure that `mpi.h` is included in the same file as `main()`
- Both Fortran/C/C++ codes:
  - Handle (e.g. remove) global and static variables

Code Conversion to AMPI (cont.)

- Global and static variables are a problem in multi-threaded programs (similar problem in OpenMP):
  - Globals/statics have a single instance per process
  - They become shared by all threads in the process
  - Example:

If *var* is a global/static, incorrect value is read!



Code Conversion to AMPI (cont.)

- General Solution: Privatize variables in thread
- Approaches:
  a) Source-to-source transformation, via Photran
  b) Swap global variables – *GOTglobals*
  c) Use TLS scheme - *TLSglobals*

  Specific approach to use must be decided on a case-by-case basis

Code Conversion to AMPI (cont.)

- First Approach: Source-to-source transform
  - Move globals/statics to an object, then pass it around
  - Automatic solution for Fortran codes: *Photran*
  - Similar idea can be applied to C/C++ codes
  - \+ Totally portable across systems/compilers
  - \+ May improve locality and cache utilization
  - \+ No extra overhead at context-switch
  - \- Requires new implementation for each language

Code Conversion to AMPI (cont.)

- Example of Transformation: C Program

Original Code:

```
int var;   /* global variable */
...


int main(...) {
  ...
  MPI_Init(...);
  MPI_Comm_rank(...,&my_rank);
  ...
  MPI_Recv(...);
  var = my_rank;
  sub();
  ...
}


void sub() {
  int x;   /* local variable */
  ...
  MPI_Wait(...);
  x = var;
  ...
}
```

Transformed Code:

```
struct data{
  int var;
};
...
int main(...) {
  struct data *d;
  ...
  MPI_Init(...);
  d = (struct data*)malloc(sizeof(struct data));
  MPI_Comm_rank(...,&my_rank);
  ...
  MPI_Recv(...);
  d->var = my_rank;
  sub(d);
  ...
}
void sub(struct data *d){
  int x;
  ...
  MPI_Wait(...);
  x = d->var;
  ...
}
```

Code Conversion to AMPI (cont.)

- Example of Transformation: Fortran Program

Original Code:

```
MODULE variables
  INTEGER :: var   ! global variable
END MODULE variables
...


PROGRAM Main
  USE variables
  ...
  CALL MPI_Init(...)
  CALL MPI_Comm_rank(...,my_rank)
  ...
  CALL MPI_Recv(...)
  var = my_rank
  CALL Sub
  ...
END PROGRAM Main


SUBROUTINE Sub
  USE variables
  INTEGER :: x   ! local variable
  ...
  CALL MPI_Wait(...)
  x = var
  ...
END SUBROUTINE Sub
```

Transformed Code:

```
MODULE variables
  TYPE data
    INTEGER :: var
  END TYPE data
END MODULE variables
...
PROGRAM Main
  USE variables
  TYPE(data) :: d
  ...
  CALL MPI_Init(...)
  CALL MPI_Comm_rank(...,my_rank)
  ...
  CALL MPI_Recv(...)
  d%var = my_rank
  CALL Sub(d)
  ...
END PROGRAM Main

SUBROUTINE Sub(d)
  USE variables
  TYPE(data) :: d
  INTEGER :: x
  ...
  CALL MPI_Wait(...)
  x = d%var
  ...
END SUBROUTINE Sub
```

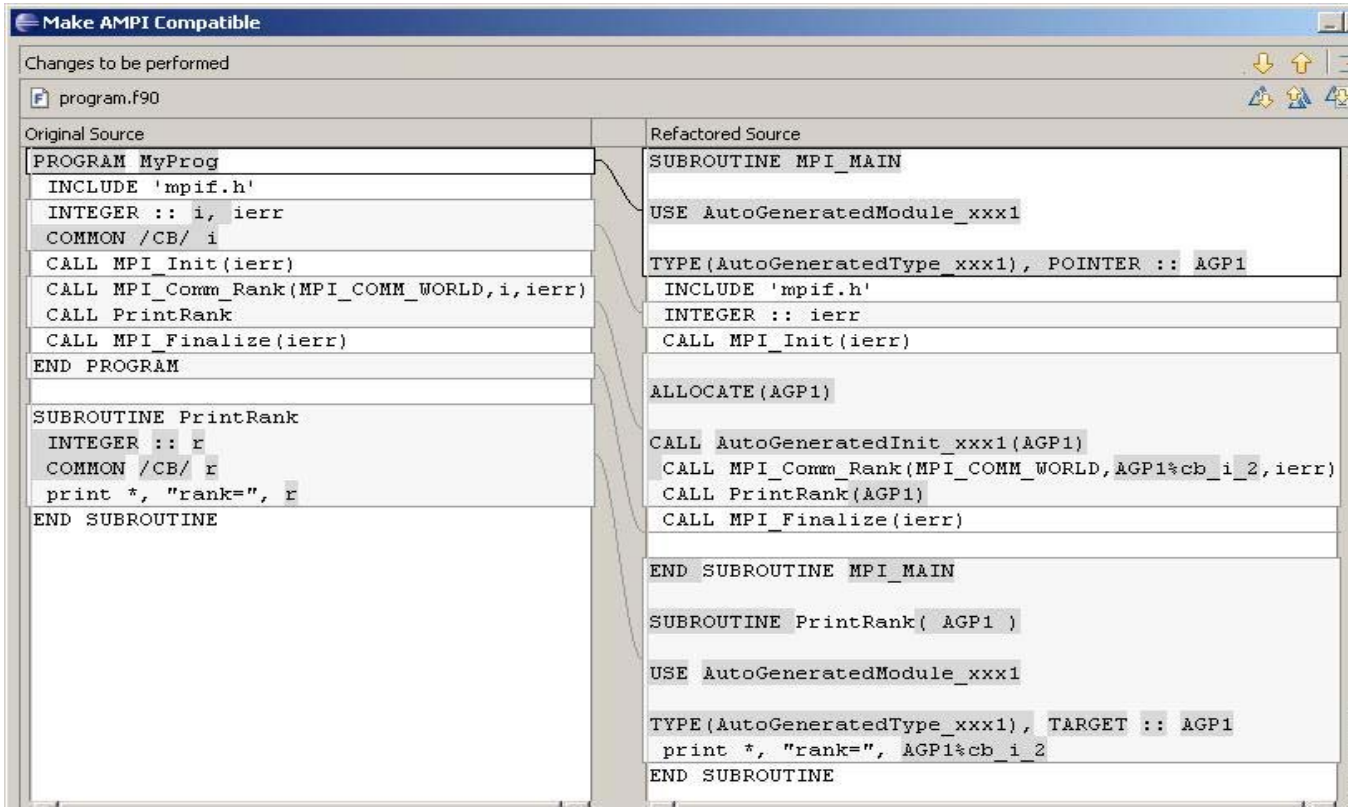
Code Conversion to AMPI (cont.)

- Automated Transformation: **Photran** Tool
  - Eclipse-based IDE, implemented in Java
  - Incorporates automatic refactorings for Fortran codes
  - Operates on "pure" Fortran 90 programs
  - Code transformation infrastructure:
    - Construct rewriteable ASTs
    - ASTs are augmented with binding information
    - AMPI-transformer not yet in public Photran distribution

Source: Stas Negara & Ralph Johnson
http://www.eclipse.org/photran/

Code Conversion to AMPI (cont.)

Photran's AMPI-Transformer GUI:



Source: Stas Negara & Ralph Johnson
http://www.eclipse.org/photran/

Code Conversion to AMPI (cont.)

- Second Approach: GOT-Globals
  - Leverage ELF – Execut. & Linking Format (e.g. Linux)
  - ELF maintains a Global Offset Table (GOT) for globals
  - Switch GOT contents at thread context-switch
  - Implemented in AMPI via build flag *–swapglobals*
  - \+ No source code changes needed
  - \+ Works with any language (C, C++, Fortran, etc)
  - \- Does not handle static variables
  - \- Context-switch overhead grows with num. variables

Code Conversion to AMPI (cont.)

- Third Approach: TLS-Globals
  - Originally employed in kernel threads
  - In C/C++ code, variables can be annotated with *__thread*
  - Modified/adapted *gfortran* compiler available
  - Implemented in AMPI via build flag *–tlsglobals*
  + Handles uniformly both globals and statics
  + No extra overhead at context-switch
  - Although popular, not yet a standard for compilers
  - Current Charm++ support only for x86 platforms

Code Conversion to AMPI (cont.)

- Summary of Current Privatization Schemes:
  - Program transformation is very portable
  - TLS scheme may become supported on Blue Waters, depending on work with IBM

Privat. Scheme	X86	IA64	Opteron	MacOS	IBM Power	SUN	IBM BG/P	Cray XT	Windows
Prog. Transf.	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
GOT Globals	Yes	Yes	Yes	No	No	Maybe	No	No	No
TLS Globals	Yes	Maybe	Yes	No	Maybe	Maybe	No	Yes	Maybe

BigSim Emulator

- Major Emulator Features:
  - Goal: emulate application behavior on target system
  - Platform for execution: existing (smaller) machine
  - Emulation is run using the Charm++ runtime system
  - Abstraction: many multiprocessor SMP nodes, connected via message-passing
  - Final result: traces about computation and communication

BigSim Emulator (cont.)

- Preparing BigSim Emulator for Use:
  - Download Charm++
    - http://charm.cs.uiuc.edu/download/downloads.shtml
  - Build Charm++/AMPI with "bigemulator" option:
    - e.g. *./build AMPI net-linux-x86_64 bigemulator –O*
    - This builds Charm++ and emulator libraries under subdir *net-linux-x86_64-bigemulator/*
  - Compiler wrappers available for MPI applications:

net-linux-x86_64-bigemulator/bin/ampicc,ampicxx,ampif90,...

BigSim Emulator (cont.)

- BigSim Emulator Usage:
  - Emulation is run via *charmrun* (like any AMPI run)
    - e.g. *charmrun +p4 +vp 16 prog_emul*
  - Emulation controlled via flags or configuration file
    - Command-line flags:
      - Any Charm++/AMPI flags will work as usual
      - Other BigSim-specific flags can be employed too
    - Configuration file: specified via *+bgconfig config_file*
    - Command-line flags have precedence over config. file

BigSim Emulator (cont.)

- BigSim Emulator – Major Flags:
  - +*x*, +*y*, +*z*: specify number of SMP target nodes
    - Only make full sense on target systems with 3D-topology
    - Typically, one can use +*x*=*K*, +*y*=*1*, +*z*=*1* where *K* is the number of nodes in the target system
  - +*cth*, +*wth*: specify communic. and worker threads
    - Typically, *cth=1*, *wth=M* where *M* is the number of cores in each node of the target system
  - +*bglog*: produce trace files at end, for simulation

BigSim Emulator (cont.)

- BigSim Emulator – Other Flags:
  - *+bgcpufactor*: specify ratio between speeds of emulating and target processors (time intervals will be multiplied by that ratio during simulation)
  - *+bgstacksize:* defines the stack size, in bytes, for each VP during emulation; default is 32 Kbytes

BigSim Emulator (cont.)

- Equivalent Emulator Configuration File:

```
x  4
y  1
z  1
cth 1
wth 8
stacksize  8000
cpufactor    0.5
log      yes
```

BigSim Emulator (cont.)

- BigSim Emulation: A Working Example

Simple Ring code, in MPI:

```
#include "mpi.h“
#include <stdio.h>
#define TIMES 10

int main(int argc, char *argv[])
{
    int myid, numprocs, i, value=0;
    double time;
    MPI_Status status;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);

    time = MPI_Wtime();

    for (i=0; i<TIMES; i++) {
      if (myid == 0) {
        MPI_Send(&value,1,MPI_INT,myid+1,999,MPI_COMM_WORLD);
        MPI_Recv(&value,1,MPI_INT,numprocs-1,999,MPI_COMM_WORLD,&status);
      }
      else {
        MPI_Recv(&value,1,MPI_INT,myid-1,999,MPI_COMM_WORLD,&status);
        value += myid;
        MPI_Send(&value,1,MPI_INT,(myid+1)%numprocs,999,MPI_COMM_WORLD);
      }
    }

    if (myid==0) printf("Sum=%d, Time=%g\n", value, MPI_Wtime()-time);
    MPI_Finalize();
}
```

BigSim Emulator (cont.)

- Ring Example Run with AMPI (no BigSim)
  - Projections' timeline view for a P=8 execution:

BigSim Emulator (cont.)

- BigSim Emulation: A Working Example

Ring code, augmented for use with BigSim:

```
#include "mpi.h“
#include <stdio.h>
#define TIMES 10

#if CMK_BLUEGENE_CHARM
extern void BgPrintf(const char *);
#define BGPRINTF(x)  if (myid == 0) BgPrintf(x);
#else
#define BGPRINTF(x)
#endif

int main(int argc, char *argv[])
{
    int myid, numprocs, i, value=0;
    double time;
    MPI_Status status;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    time = MPI_Wtime();
    BGPRINTF("Start of major loop at %f \n");
    for (i=0; i<TIMES; i++) {
      if (myid == 0) {
        MPI_Send(&value,1,MPI_INT,myid+1,999,MPI_COMM_WORLD);
        MPI_Recv(&value,1,MPI_INT,numprocs-1,999,MPI_COMM_WORLD,&status);
      }
      else {
        MPI_Recv(&value,1,MPI_INT,myid-1,999,MPI_COMM_WORLD,&status);
        value += myid;
        MPI_Send(&value,1,MPI_INT,(myid+1)%numprocs,999,MPI_COMM_WORLD);
      }
    }
    BGPRINTF("End of major loop at %f \n");
    if (myid==0) printf("Sum=%d, Time=%g\n", value, MPI_Wtime()-time);
    MPI_Finalize();
}
```

BigSim Emulator (cont.)

Ring code - emulation output:

```
>   ./charmrun +p4 ring_emul +bgconfig bg_config ++local
Charmrun> started all node programs in 0.008 seconds.
Charm++: scheduler running in netpoll mode.
Reading Bluegene Config file bg_config ...
BG info> Simulating 4x1x1 nodes with 1 comm + 8 work threads each.
BG info> Network type: bluegene.
alpha: 1.000000e-07     packetsize: 1024        CYCLE_TIME_FACTOR:1.000000e-03.
CYCLES_PER_HOP: 5       CYCLES_PER_CORNER: 75.
BG info> cpufactor is 0.500000.
BG info> floating point factor is 0.000000.
BG info> BG stack size: 8000 bytes.
BG info> Using WallTimer for timing method.
BG info> Generating timing log.
LB> Load balancer ignores processor background load.
Start of major loop at 0.000635
End of major loop at 0.002520
Sum=4960, Time=0.00188543
[0] Number is numX:4 numY:1 numZ:1 numCth:1 numWth:8 numEmulatingPes:4 totalWorkerProcs:32 bglog_ver:6
[1] Wrote to disk for 1 BG nodes.
[2] Wrote to disk for 1 BG nodes.
[3] Wrote to disk for 1 BG nodes.
[0] Wrote to disk for 1 BG nodes.
```

BigSim Emulator (cont.)

Ring code – files resulting from emulation:

```
> ls -l
-rw-r--r-- 1 cmendes kale     73 2011-04-19 12:26 bg_config
-rw-r--r-- 1 cmendes kale     72 2011-04-19 12:31 bgPrintFile.0
-rw-r--r-- 1 cmendes kale     60 2011-04-19 12:31 bgTrace
-rw-r--r-- 1 cmendes kale 147336 2011-04-19 12:31 bgTrace0
-rw-r--r-- 1 cmendes kale 143991 2011-04-19 12:31 bgTrace1
-rw-r--r-- 1 cmendes kale 139070 2011-04-19 12:31 bgTrace2
-rw-r--r-- 1 cmendes kale 140278 2011-04-19 12:31 bgTrace3
-rwxr-xr-x 1 cmendes kale 109683 2011-04-19 11:48 charmrun
-rw-r--r-- 1 cmendes kale   1040 2011-04-18 23:18 ring.c
-rwxr-xr-x 1 cmendes kale 3569777 2011-04-19 12:24 ring_emul


> cat bgPrintFile.0
[0] Start of major loop at 0.000635
[0] End of major loop at 0.002520
```

Output from simulated PE 0

Files with traces (*bgTraces*): comput. and communic. events

BgPrintf: mechanism to "timestamp" places in code
(%f is replaced by simulated time)

C version: BgPrintf(string)

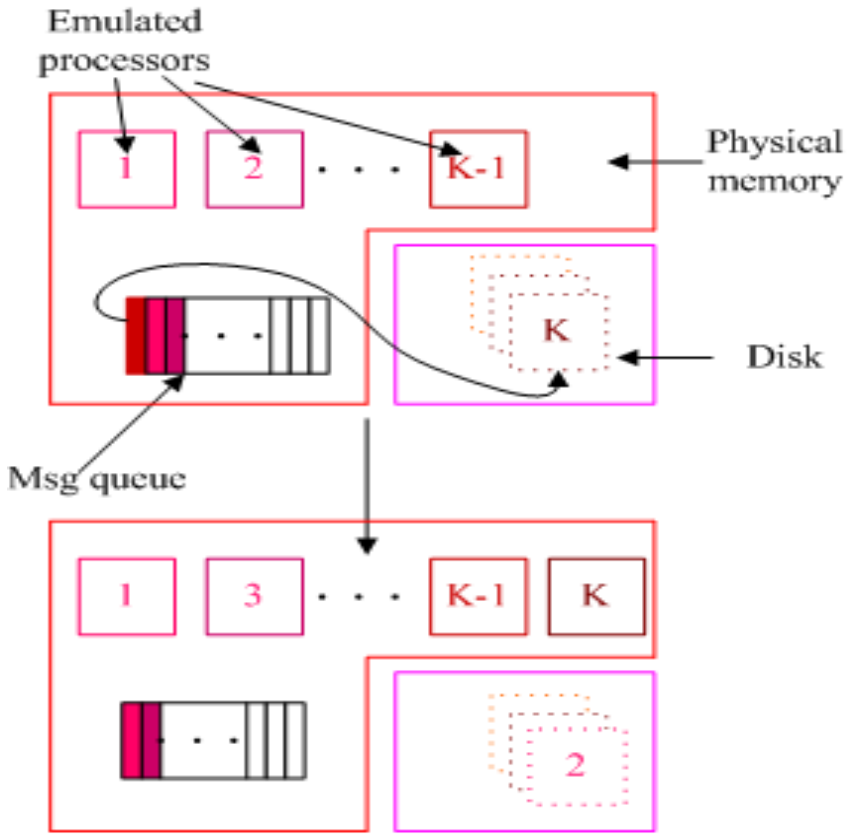
Fortran version: fbgprintf(string)

BigSim Emulator (cont.)

- Other Emulation Features:
  - Skip-points
    - Mark locations in source code: *AMPI_Set_StartEvent( )*
    - Marker records will be generated in the bgTrace file
    - Useful at simulation phase
  - More accurate modeling of sequential performance
    a) Based on performance counters
    b) Instruction-level/cycle-accurate simulation
    c) Model-based (time most-used functions and interpolate to create a model)

BigSim Emulator (cont.)

- Out of Core Emulation:
  - Motivation
    - Applications with large memory footprint
    - VM system can not handle well
  - Use hard drive
    - Similar to checkpointing
  - Message driven execution
    - Peek msg queue => what execute next? (prefetch)



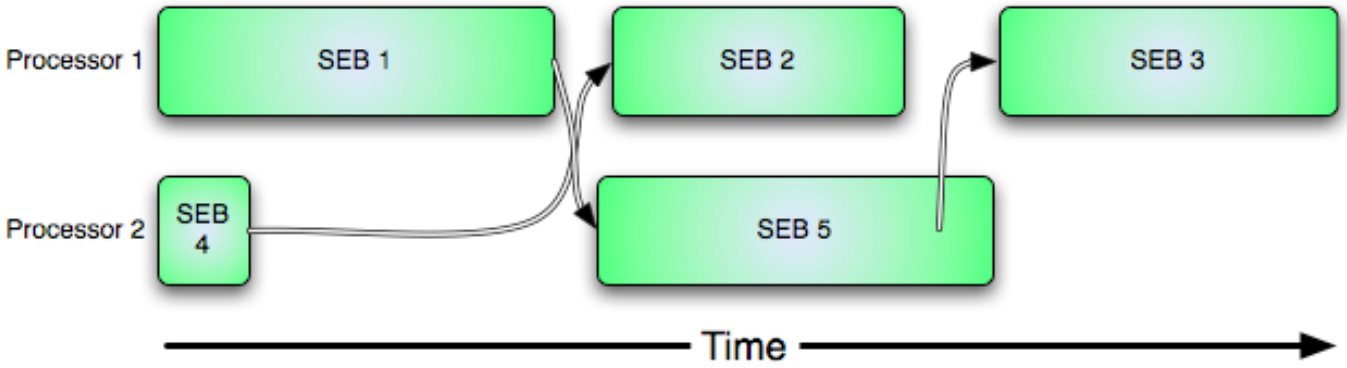
BigSim Emulator (cont.)

- Use of Out-of-Core Emulation :
  - Change charm/tmp/Conv-mach-bigemulator.h
  - #define BIGSIM_OUT_OF_CORE 1
  - Recompile Charm++ and application
  - Run the emulated application with *+bgooc 1024*

Trace Utilities

- Contents of the Trace Files:

Traces for 2 target processors:



Each SEB has:

- startTime, endTime
- Incoming Message ID
- Outgoing messages
- Dependences

Tools for reading bgTrace binary files:

1.charm/example/bigsim/tools/loadlog
Convert to human-readable format

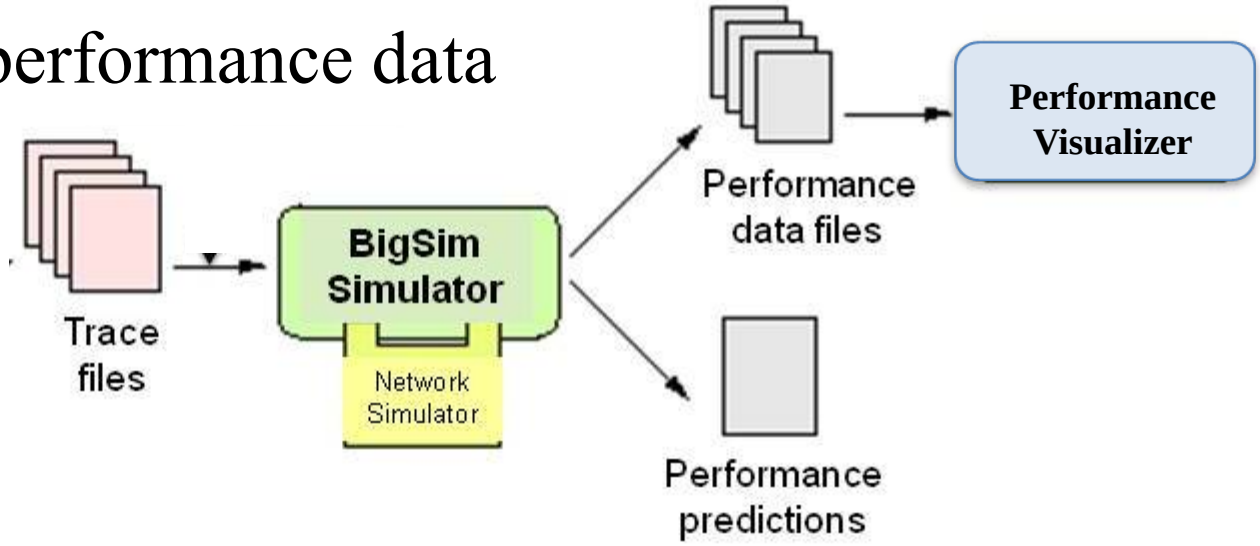
2.charm/example/bigsim/tools/log2proj
Convert to trace projections log files

Trace Utilities (cont.)

- *LogAnalyzer* : Tool for Analyzing Trace Files
  - Various options available:
    - Display number of records for each target processor
    - Dump events from a target processor, in ASCII
    - Show number of msgs *sent* by each target processor
    - Show number of msgs *received* by each target processor
  - Can optionally be used in interactive mode
    - *LogAnalyzer –i*
  - Distributed with the BigSim simulation component

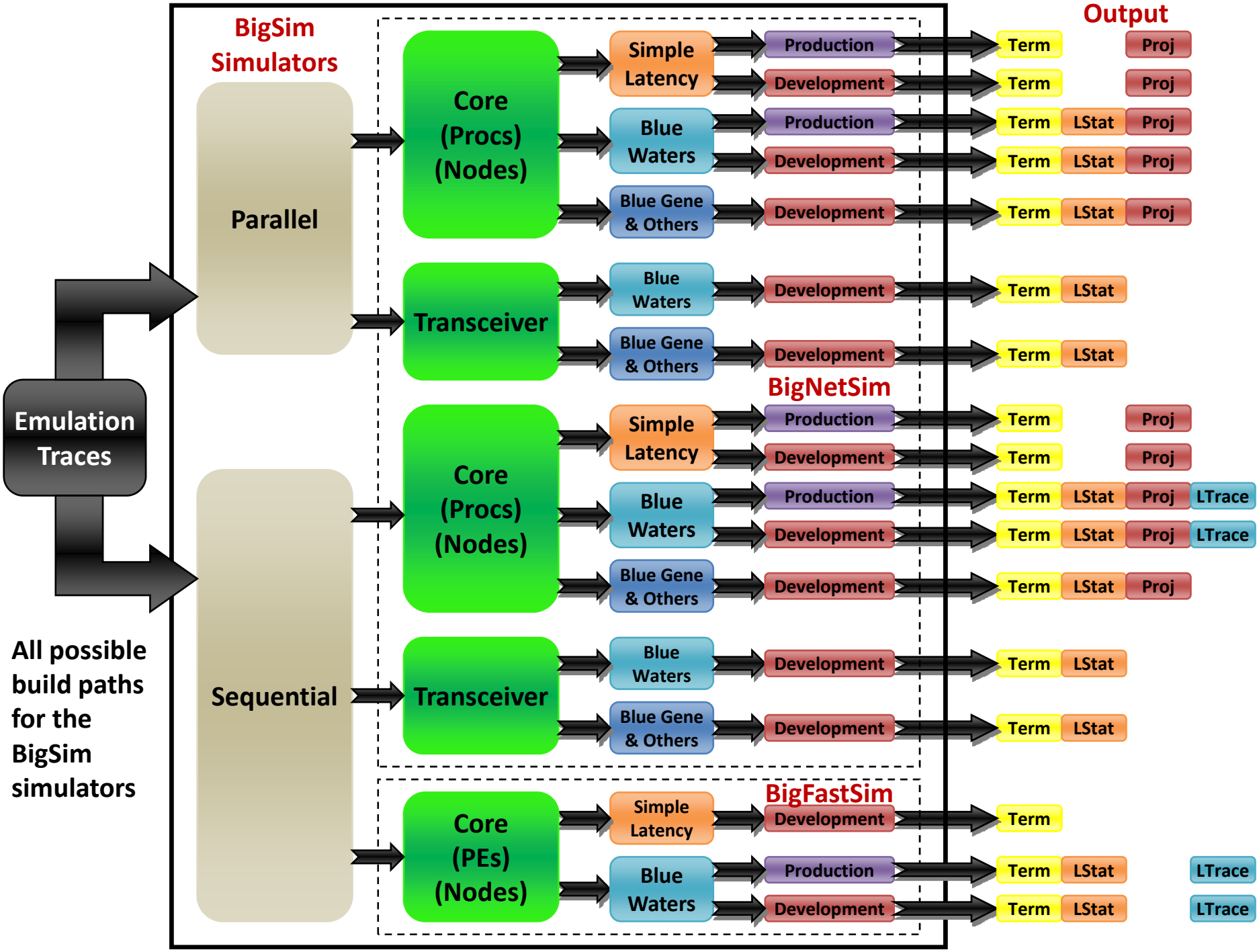
BigSim Simulation Phase

- Goals:
  - Process traces adjusting times
  - Generate performance data



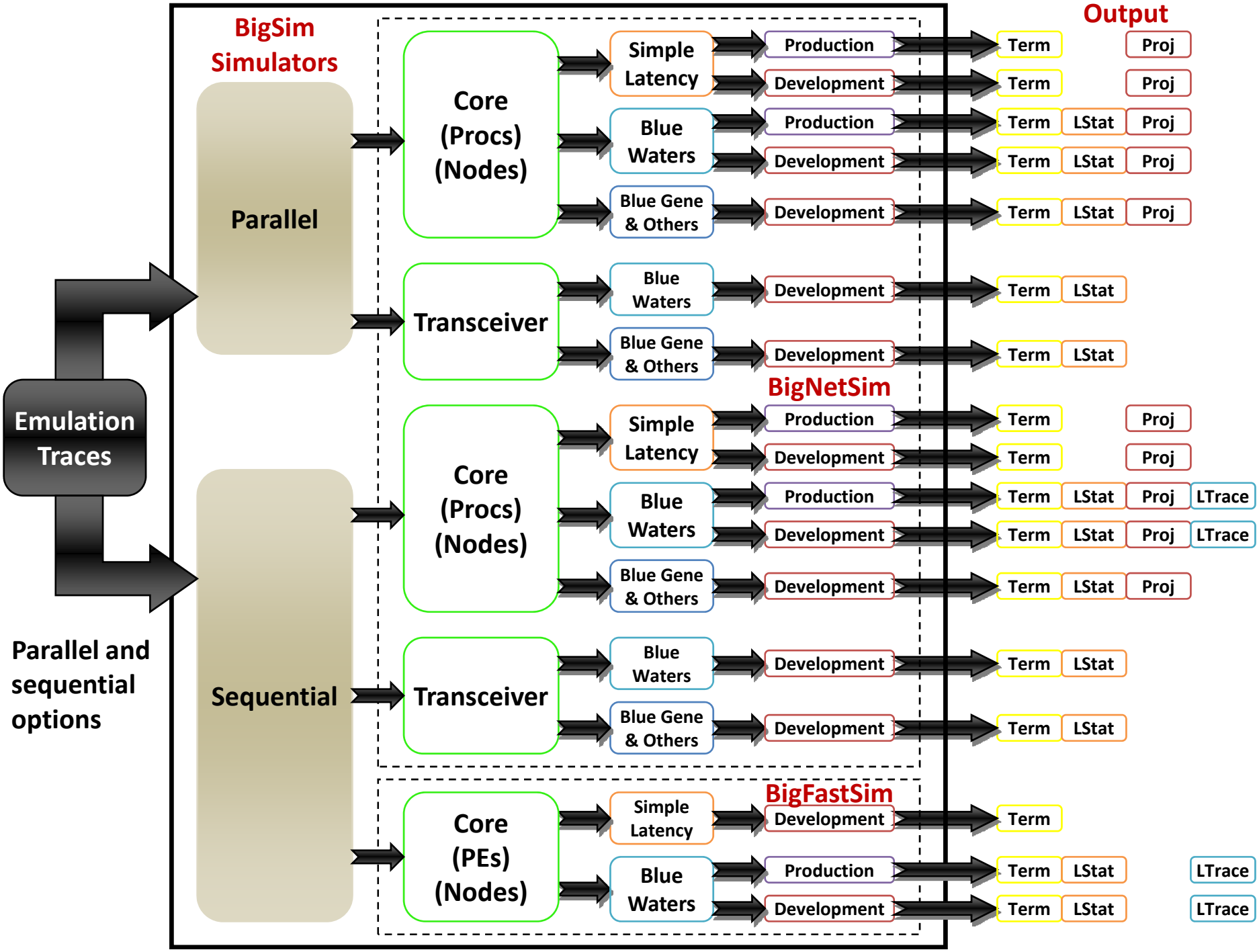
Simulation Explanation Approach

- Difficult to explain all features/dimensions of BigSim simulators at once
  - Lots of options and "branches"
- Approach
  - Show overview of possible build/config paths
  - Explain features and capabilities
  - Examples of specific build/config paths
  - Discuss output


BigSim Simulators
Output
Emulation Traces
Parallel
Sequential
Core (Procs) (Nodes)
Transceiver
Core (PEs) (Nodes)
Simple Latency
Blue Waters
Blue Gene & Others
Production
Development
Term
LStat
Proj
LTrace
BigNetSim
BigFastSim
All possible build paths for the BigSim simulators

BigSim Simulation

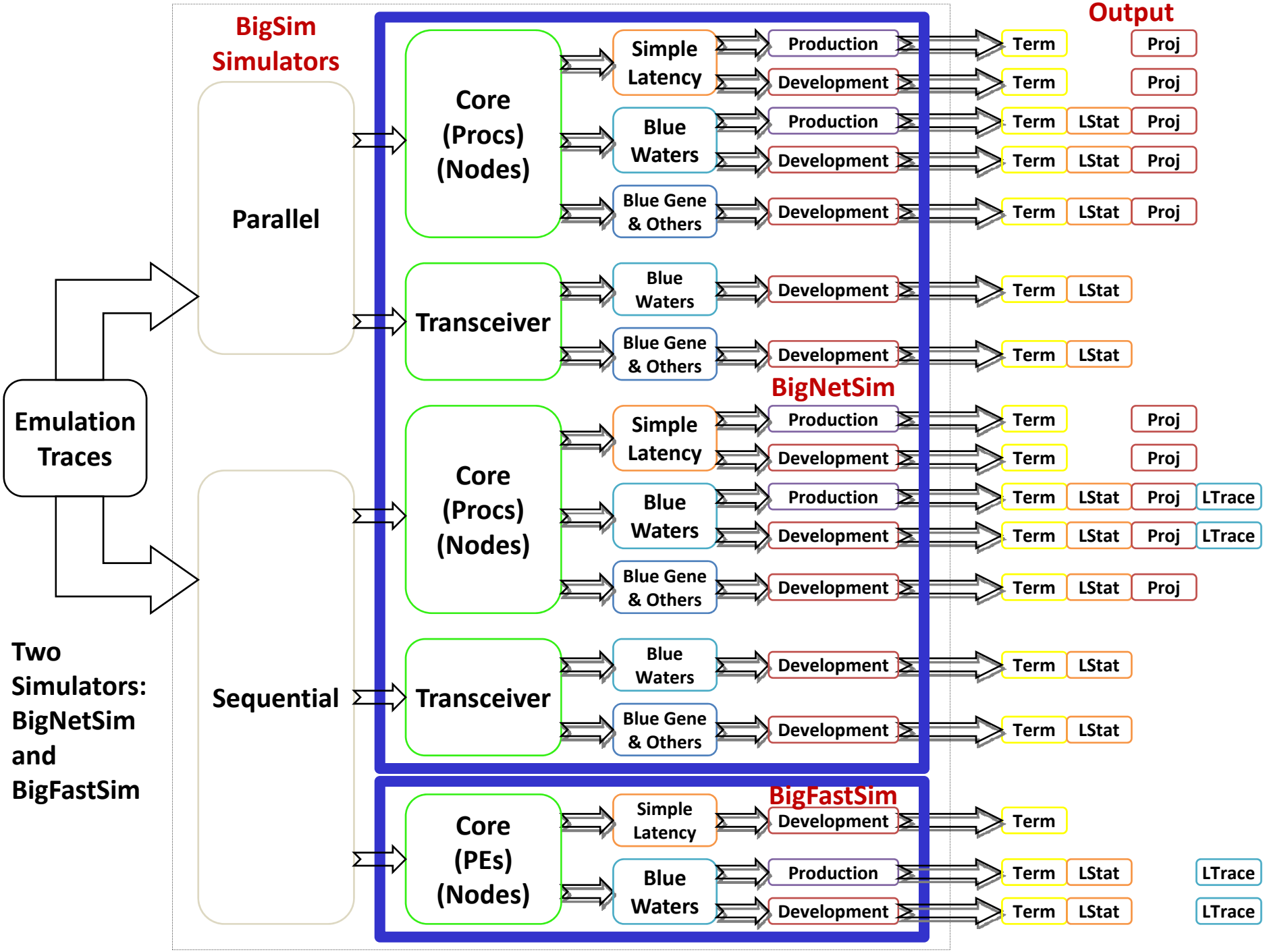
- Packet-level
- Post-mortem simulation on emulation traces
  - Application only run once on emulator to generate traces
- Goal: examine network performance
  - Less concerned about final run time prediction
  - Most useful for analyzing communication-bound applications


BigSim Simulators
Output
Emulation Traces
Parallel and sequential options
Parallel
Sequential
Core (Procs) (Nodes)
Transceiver
Core (PEs) (Nodes)
Simple Latency
Blue Waters
Blue Gene & Others
Production
Development
Term
LStat
Proj
LTrace
BigNetSim
BigFastSim

Sequential vs. Parallel

Mode	Current Advantages	Current Disadvantages
Sequential	• Faster than parallel • Total memory footprint less than parallel	• Has to fit in memory on 1 core/node • Run time can still be long for large simulations (hours or days)
Parallel	• Distributes some of the memory across multiple nodes	• Slower than sequential (optimal run time is usually at least 50% more)

- Attempts are being made to improve parallel performance to achieve run times, at scale, that are less than or equal to sequential


BigSim Simulators
Output
Emulation Traces
Parallel
Sequential
Core (Procs) (Nodes)
Transceiver
Core (Procs) (Nodes)
Transceiver
Core (PEs) (Nodes)
Simple Latency
Blue Waters
Blue Gene & Others
Blue Waters
Blue Gene & Others
Simple Latency
Blue Waters
Blue Gene & Others
Blue Waters
Blue Gene & Others
Simple Latency
Blue Waters
Production
Development
Term
LStat
Proj
LTrace
BigNetSim
BigFastSim
Two Simulators: BigNetSim and BigFastSim

Two Simulators

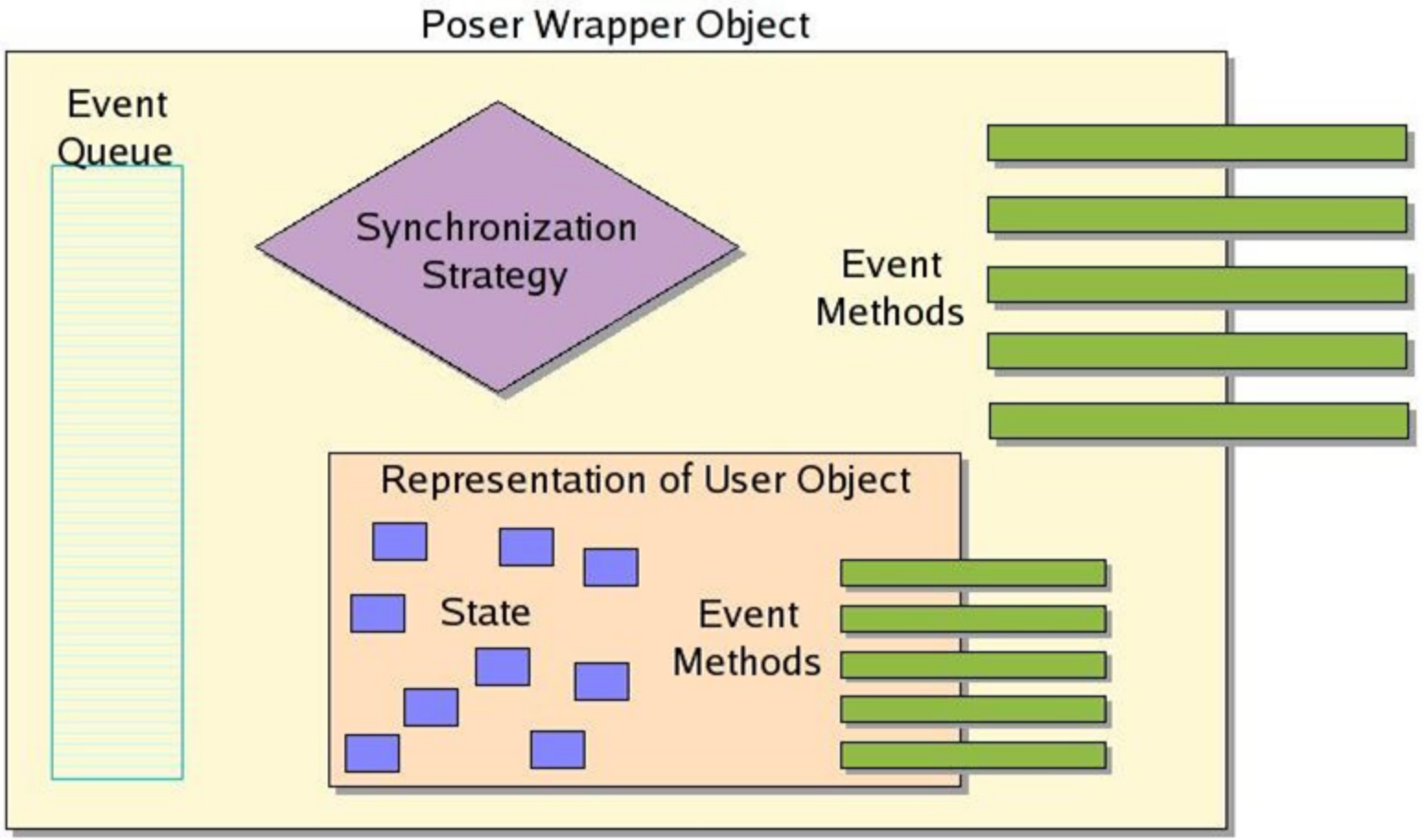
- BigNetSim
  - Original simulator
  - Parallel or sequential execution
- BigFastSim
  - New simulator
  - Strictly sequential
  - Better performance compared to BigNetSim

BigSim Simulators - BigNetSim

- Parallel Discrete Event Simulator (PDES)
- Built on POSE (Parallel Object-oriented Simulation Environment)
  - Simulation framework built on Charm++
  - Each simulation object (processor, node, switch, etc.) implemented as a poser
- Can take advantage of Charm++ features
  - Checkpoint-to-disk allows restart after hardware failures or end of allocation time
  - Load balancing

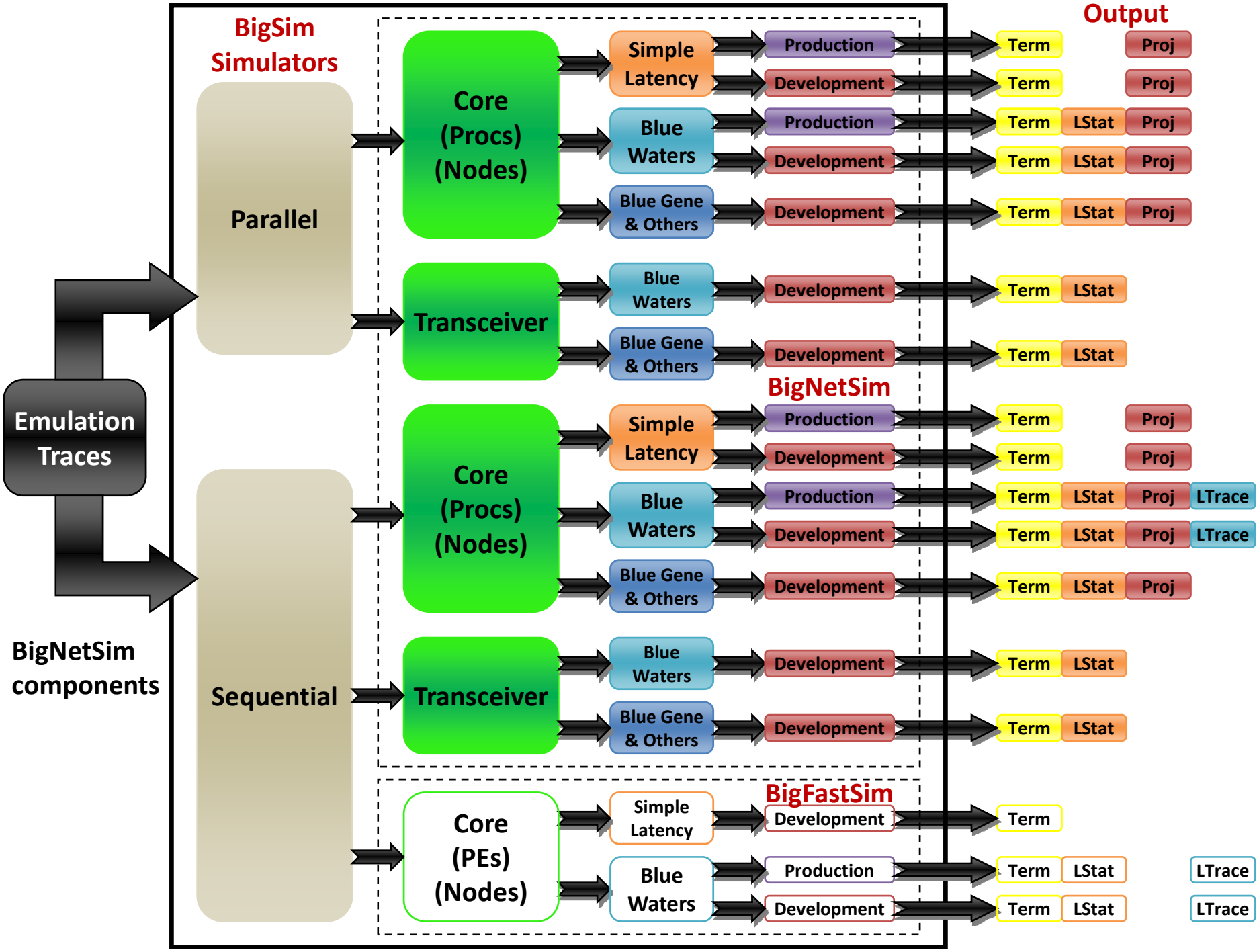
POSE

- Each poser is a tiny simulation



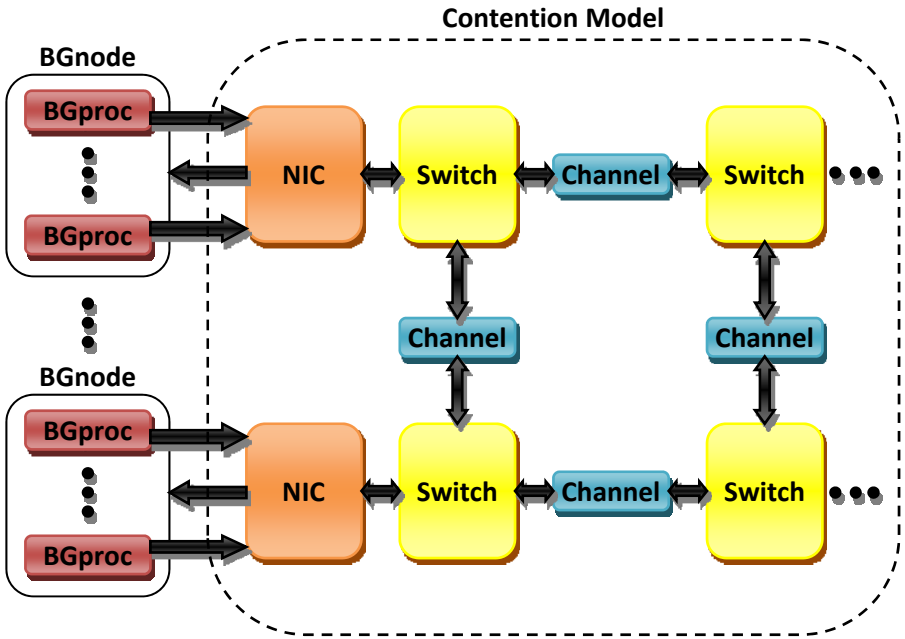
BigSim Simulators – BigFastSim

- Stand-alone C++ program
- Does not use Charm++ runtime, but does use some libraries
- Runs only on 1 processor
  - Results in faster execution and smaller memory footprint
- Interface is similar to BigNetSim
  - Still converging


BigSim Simulators
Output
Emulation Traces
BigNetSim components
Parallel
Sequential
Core (Procs) (Nodes)
Transceiver
Core (PEs) (Nodes)
Simple Latency
Blue Waters
Blue Gene & Others
Production
Development
Term
Proj
LStat
LTrace
BigNetSim
BigFastSim

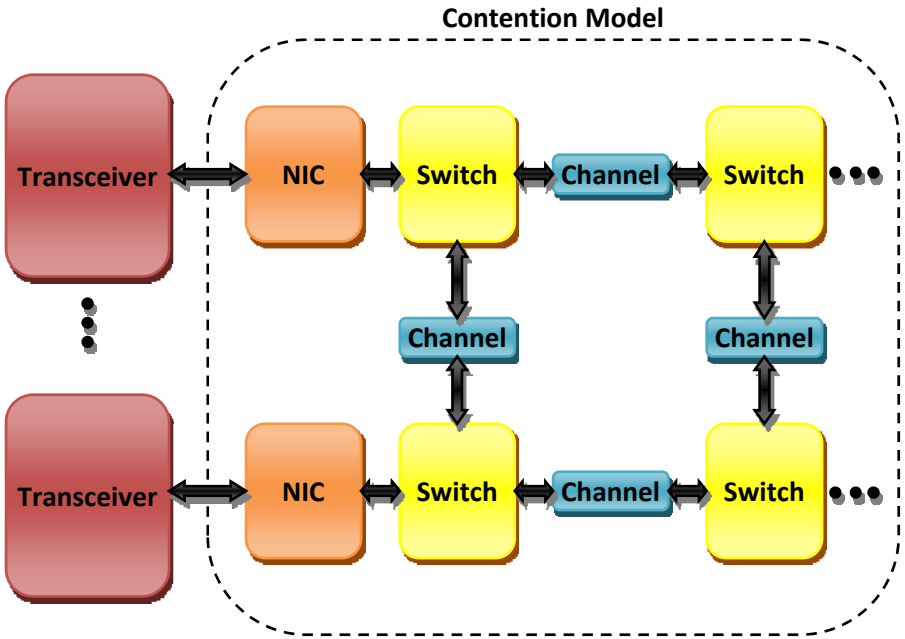
BigNetSim Component Structure for Trace-Driven Simulations

- Full-contention model
- Core: nodes, procs
- Network: NICs, Switches, Channels



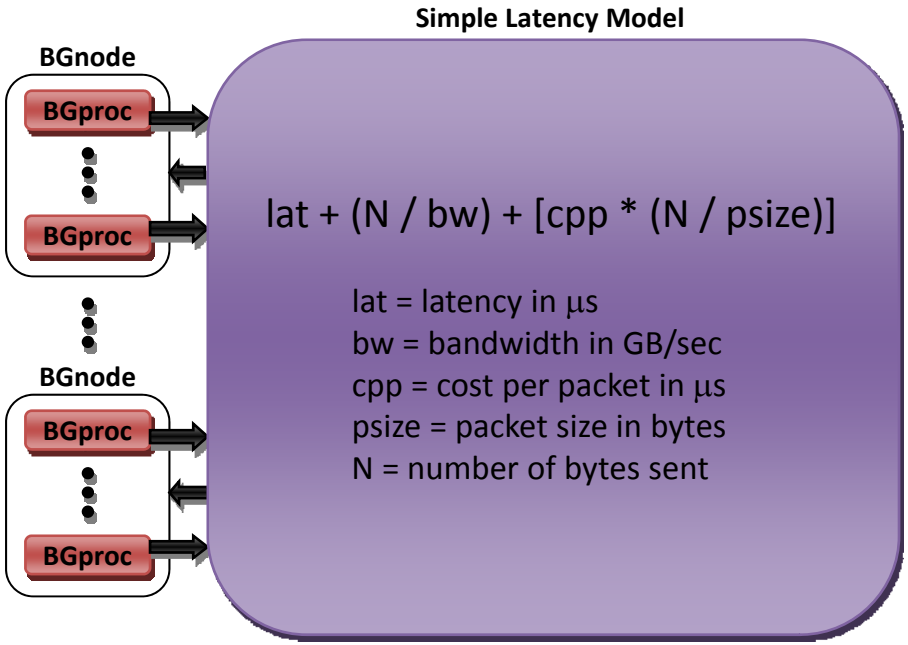
BigNetSim Component Structure for Transceiver Simulations

- Full-contention model
- Core components replaced with Transceivers
- Transceiver = traffic pattern generator



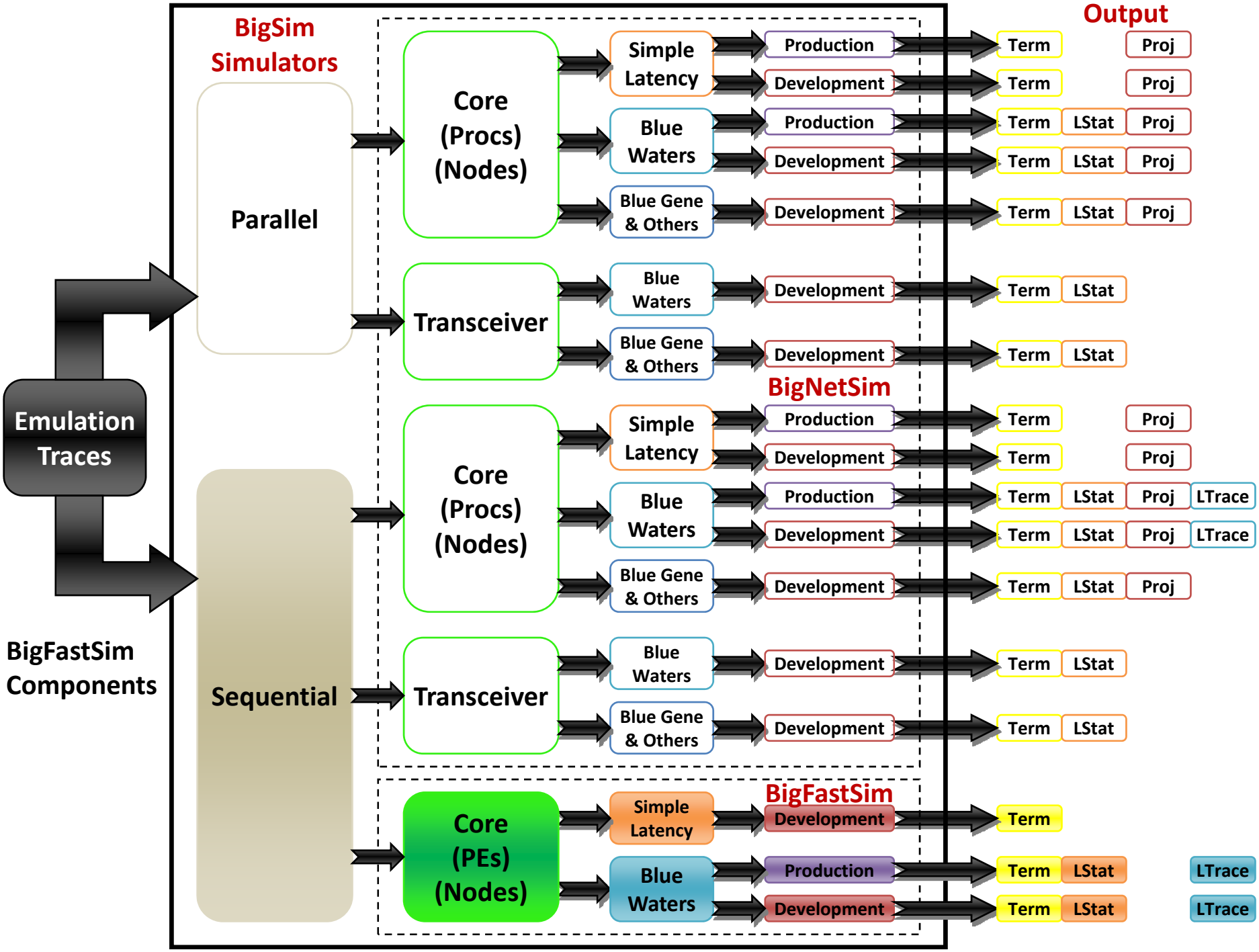
BigNetSim Component Structure for Simple Latency Simulations

- Network components replaced with simple equation
- No contention



BigNetSim – Development vs. Production

- Development path
  - Standard way of building and running in the past
  - Many more options and parameters
  - Much more confusing
- Production path
  - Recent addition
  - Many settings hidden from user
  - User only has to worry about a handful of command-line parameters


BigSim Simulators
Output
Emulation Traces
BigFastSim Components
Parallel
Sequential
Core (Procs) (Nodes)
Transceiver
Core (PEs) (Nodes)
Simple Latency
Blue Waters
Blue Gene & Others
Production
Development
Term
LStat
Proj
LTrace
BigNetSim
BigFastSim

BigFastSim Build Paths

- Has PEs (procs) and nodes like BigNetSim
- Currently only has Simple Latency and Blue Waters models
  - Both production and development paths exist for Blue Waters
- Projections output and other network models are planned as future additions

Compile and Run Sequence

- General procedure and options
- Specific examples

Basic Workflow (All Paths)

- Download and compile Charm++
  - Compile POSE
  - Compile bigsim
- Download and compile simulator
  - Configure simulator
  - Compile simulator
- Run simulator
- Analyze output

Download and Compile Charm++ Components – Linux

- Download latest version from PPL repository

  http://charm.cs.uiuc.edu/download/

- Compile POSE and bigsim

  cd charm

  ./build pose net-linux

  ./build bigsim net-linux

BigNetSim Build Path

- Download latest code from SVN repository

  svn co https://charm.cs.uiuc.edu/svn/repos/BigNetSim

- Directory structure: BigNetSim/trunk/
  - Network models: BlueGene/ Redstorm/ etc.
  - Simple Latency model: SimpleLatency/
  - Network config: Topology/ Routing/ InputVcSelection/ OutputVcSelection/
  - Core simulation files: Main/
  - Tools: tools/
  - Working directory: tmp/

Configure BigNetSim

- Modify BigNetSim/trunk/Makefile.common
  - Change CHARMBASE so it points to your Charm++ directory
  - Change OPTS to the same OPTS used to build Charm++ (e.g., -O3 -DCMK_OPTIMIZE, etc.)
    - These must match or errors may occur at run time
- For specific networks: copy netconfig file from network directory to BigNetSim/trunk/tmp and modify

Compile BigNetSim

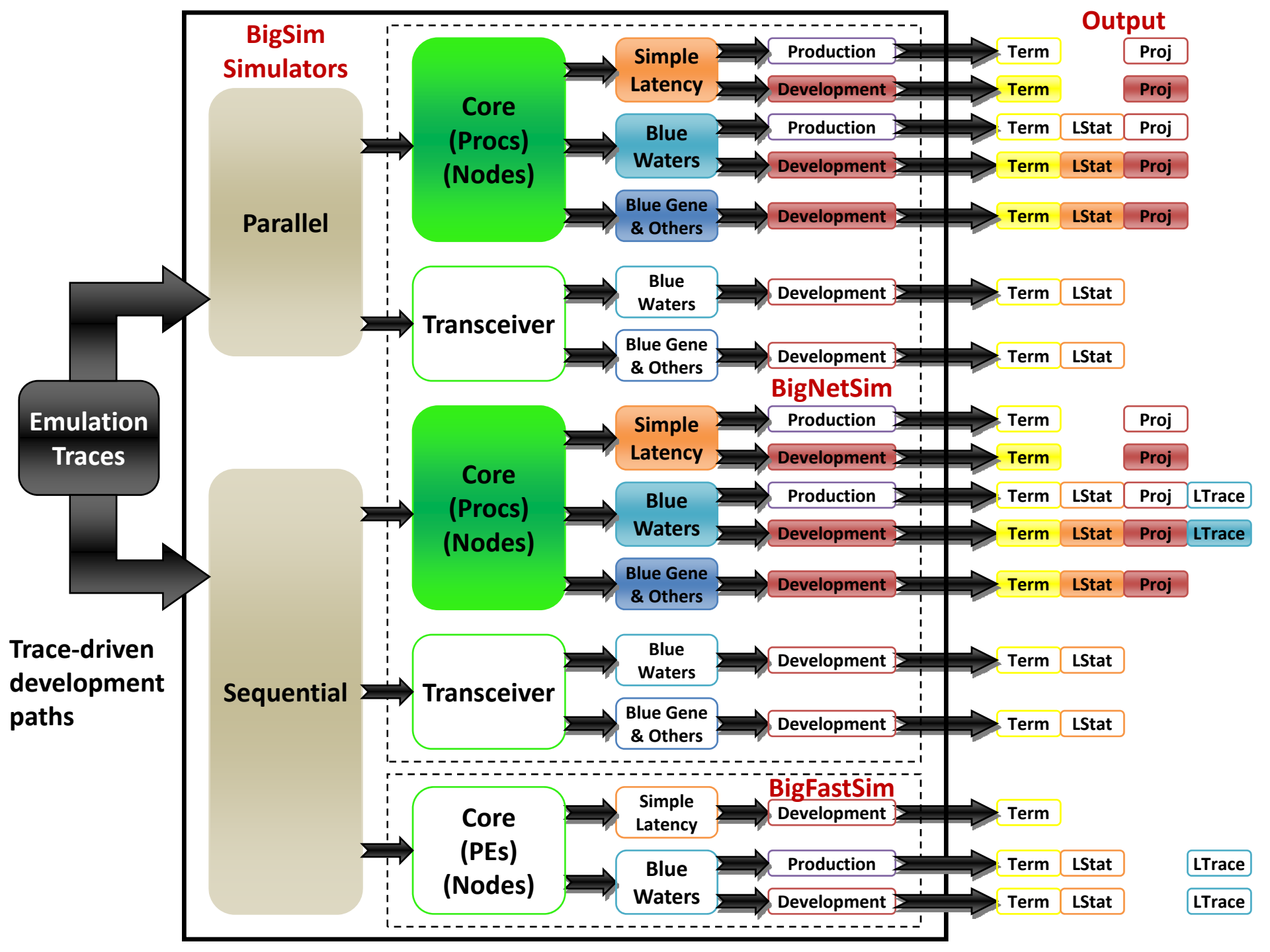
- *cd* into desired network directory
  - E.g., BigNetSim/trunk/SimpleLatency or BigNetSim/trunk/BlueGene
- Build options
  - Parallel version

    make
  - Sequential version (optimized for running in 1 processing core)

    make SEQUENTIAL=1
  - Production version (instead of development version)
    - Currently only SimpleLatency and BlueWaters

    parallel: *make PRODUCTION=1*

    sequential: *make SEQUENTIAL=1 PRODUCTION=1*

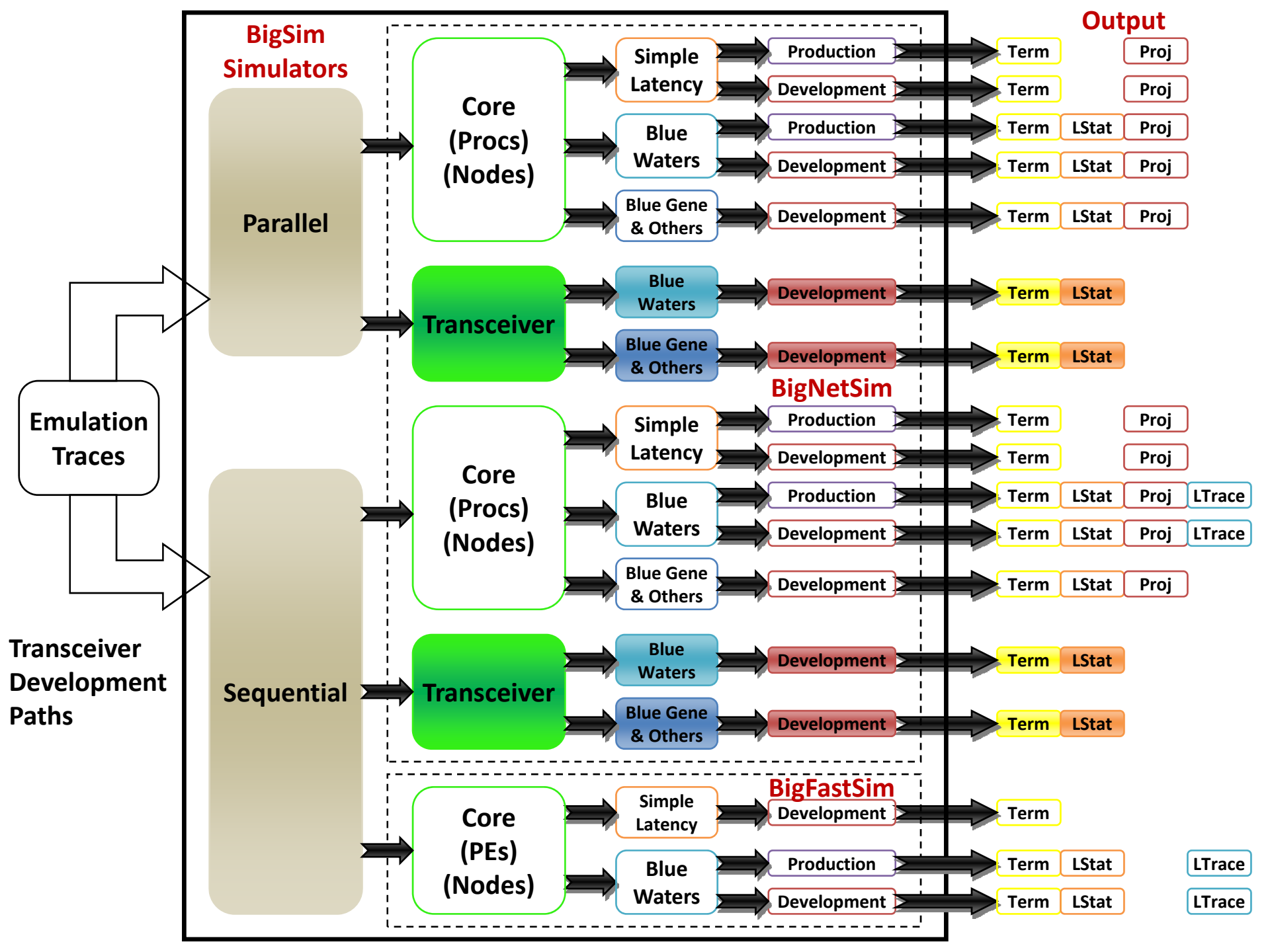

BigSim Simulators
Output
Emulation Traces
Trace-driven development paths
Parallel
Sequential
Core (Procs) (Nodes)
Transceiver
Core (PEs) (Nodes)
Simple Latency
Blue Waters
Blue Gene & Others
Production
Development
Term
LStat
Proj
LTrace
BigNetSim
BigFastSim

Running BigNetSim – Development Path

- cd BigNetSim/trunk/tmp
- Trace-driven simulation
  - Copy bgTrace files into /tmp directory
  - Sequential build

    ./bigsimulator <params>
  - Parallel build

    ./charmrun +p<#procs> bigsimulator <params>
- See BigNetSim online manual for possible parameters (see last slide for URL)


BigSim Simulators
Output
Emulation Traces
Parallel
Sequential
Core (Procs) (Nodes)
Transceiver
Core (PEs) (Nodes)
Simple Latency
Blue Waters
Blue Gene & Others
Production
Development
Term
LStat
Proj
LTrace
BigNetSim
BigFastSim
Transceiver Development Paths

Running BigNetSim – Development Path (cont.)

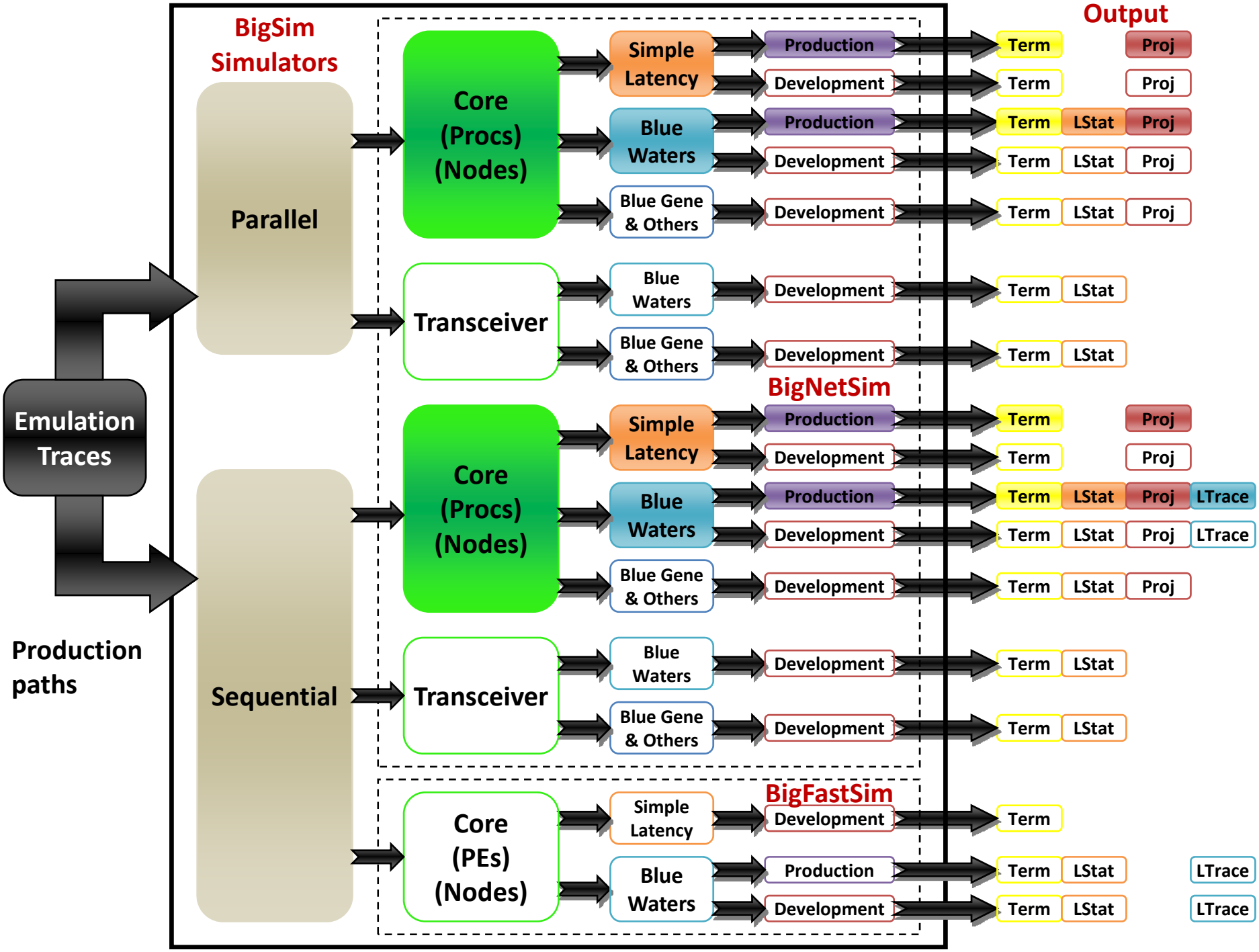
- Transceiver simulation
  - Set netconfig parameter USE_TRANSCEIVER to 1
  - Sequential build

    ./bigsimulator arg1 arg2 arg3 arg4 arg5 arg6
  - Parallel build

    ./charmrun +p<#procs> bigsimulator arg1 arg2 arg3 arg4 arg5 arg6

Transceiver Parameters

Arg	Value	Meaning
1	0	Latency-only model
	1	Full contention model
2	1	Deterministic traffic
	2	Poisson traffic
3	1	Pattern: K Shift
	2	Pattern: Ring
	3	Pattern: Bit Transpose
	4	Pattern: Big Reversal
	5	Pattern: Bit Complement
	6	Pattern: Uniform Distribution
	7	Pattern: Multi Ping Pong
	18	Pattern: Turbulence YZ Transpose
	19	Pattern: Turbulence XY Transpose
4	#	Number of Messages
5	#	Message Size (bytes)
6	#	Load Factor


BigSim Simulators
Output
Emulation Traces
Production paths
Parallel
Sequential
Core (Procs) (Nodes)
Transceiver
Core (Procs) (Nodes)
Transceiver
Core (PEs) (Nodes)
Simple Latency
Blue Waters
Blue Gene & Others
Blue Waters
Blue Gene & Others
Simple Latency
Blue Waters
Blue Gene & Others
Blue Waters
Blue Gene & Others
Simple Latency
Blue Waters
BigNetSim
BigFastSim
Production
Development
Term
LStat
Proj
LTrace

Running BigNetSim – Production Path

- Currently only available for trace-driven Simple Latency and Blue Waters runs
  - Note: for now, non-NDA users may only have access to Blue Waters production executables on Blue Print
  - Copy bgTrace files into /tmp directory
  - Sequential build

    ./bigsimulator <params>
  - Parallel build

    ./charmrun +p<#procs> bigsimulator <params>

Simple Latency Production Parameters

Bandwidth and latency must be specified

-bw <double>	Link bandwidth in GB/s
-lat <double>	Link latency in µs

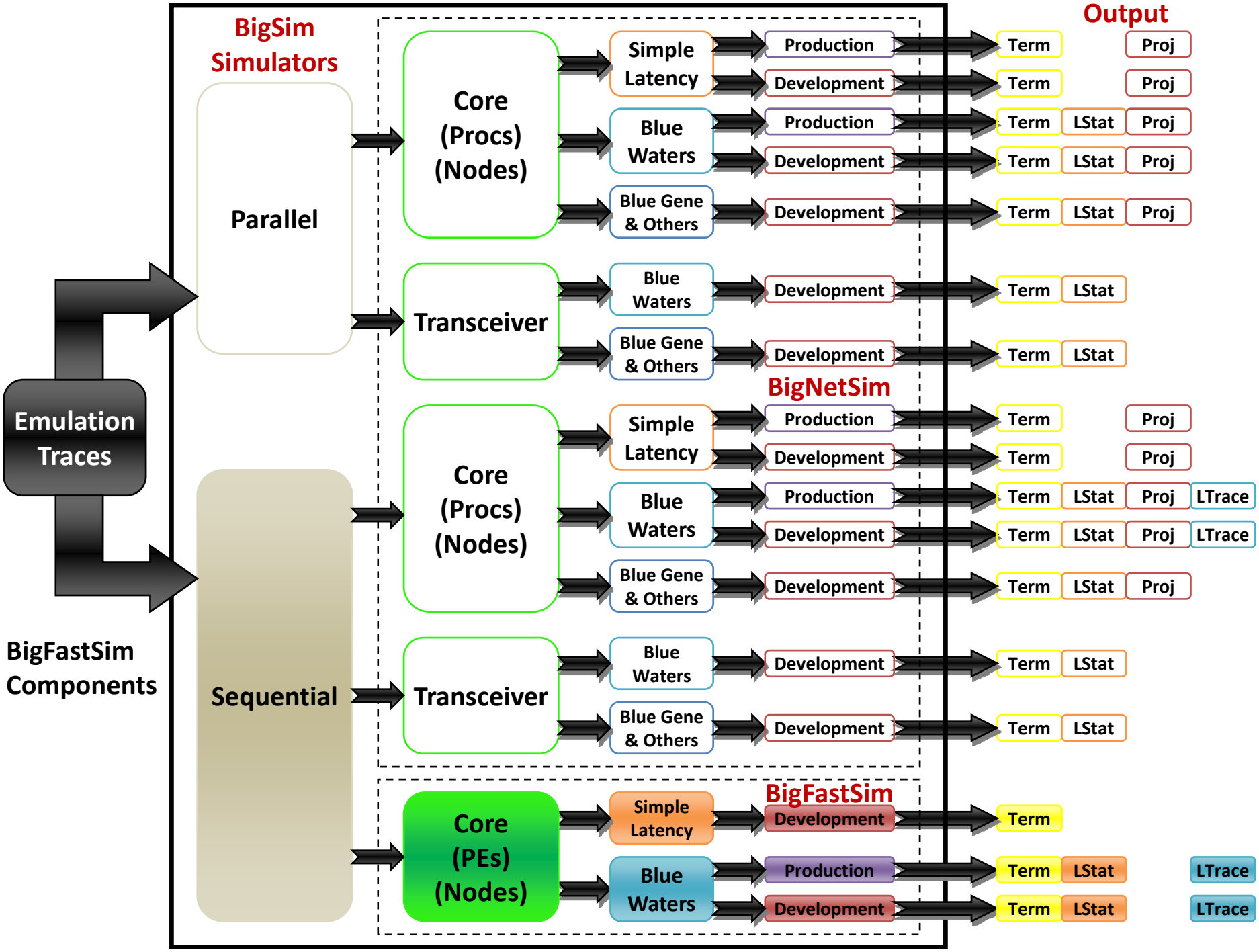
Other optional arguments

-help	Displays all available arguments
-cpp <double>	Cost per packet in µs
-psize <int>	Packet size in bytes
-bw_in <double>	Intra-node bandwidth in GB/s (defaults to -bw value if not specified)
-lat_in <double>	Intra-node latency in µs (defaults to 0.5µs if not specified)
-check	Checks for unexecuted events at the end of the simulation
-cpufactor <double>	A constant by which SEB execution times are multiplied; defaults to 1.0
-debuglevel <0|1>	0: no debug statements 1: high-level debug statements and summary info
-projname <string>	Sets the name of the projections logs that will be corrected based on network simulation
-skip_start <int>	Sets the skip point at which simulation execution begins
-skip_end <int>	Sets the skip point at which simulation execution ends
-tproj	Generate projections logs based only on network simulation

Blue Waters Production Parameters

No arguments are required; all are optional

-help	Displays all available arguments
-check	Checks for unexecuted events at the end of the simulation
-cpufactor <double>	A constant by which SEB execution times are multiplied; defaults to 1.0
-debuglevel <0|1>	0: no debug statements
	1: high-level debug statements and summary info
-linkstats	Enable link stats for display at the end of the simulation
-projname <string>	Sets the name of the projections logs that will be corrected based on network simulation
-skip_start <int>	Sets the skip point at which simulation execution begins
-skip_end <int>	Sets the skip point at which simulation execution ends
-tproj	Generate projections logs based only on network simulation
-traceutil	Enable tracing of link utilization
-tracesat	Enable tracing of link saturation
-tracecont	Enable tracing of link contention


BigSim Simulators
Output
Emulation Traces
BigFastSim Components
Parallel
Sequential
Core (Procs) (Nodes)
Transceiver
Core (PEs) (Nodes)
Simple Latency
Blue Waters
Blue Gene & Others
Production
Development
Term
LStat
Proj
LTrace
BigNetSim
BigFastSim

BigFastSim Build Path

- Download latest code from git repository

  git clone git://charm.cs.uiuc.edu/BigFastSim

- Directory structure:
  - Core simulation files: BigFastSim/
  - Simulation objects (PEs, nodes, etc.): BigFastSim/entities/
  - Simulation events: BigFastSim/events/
  - Network models: BigFastSim/networks/<network name>
  - Build directory: BigFastSim/Release/

Configure BigFastSim

- Modify BigFastSim/Release/makefile
  - Change CHARMPATH so it points to your Charm++ directory

Compile BigFastSim

- *cd* into BigFastSim/Release/ if not already there
- Build options
  - Simple latency version

    make
  - Blue Waters version (not in public repository)

    make BLUEWATERS=1

Running BigFastSim

- Copy bgTrace files into BigFastSim/Release/
- Execute program

  ./seqSimulator <params>

BigFastSim Parameters

No arguments are required; all are optional

-bw <int>	Link bandwidth in GB/s; defaults to 1
-lat <int>	Link latency in µs; defaults to 1
-wsize <int>	Event window size for reading trace files incrementally; defaults to no window
-check	Checks for unexecuted events at the end of the simulation
-cpufactor <double>	A constant by which SEB execution times are multiplied; defaults to 1.0
-skip_start <int>	Sets the skip point at which simulation execution begins
-skip_end <int>	Sets the skip point at which simulation execution ends
-noise <file_name_prefix>	Prefix of file names containing captured noise. There should be files named <file_name_prefix>0-[cores-1] for each target machine core's noise.
-pnoise <file_name_prefix>	Prefix of file names containing noise pattern. There should be files named <file_name_prefix>0-[cores-1] for each target machine core's noise.
-msgreplace <new_size> [<cutoff>]	Replaces the message size of all messages with size more than cutoff to new_size; cutoff defaults to 1000 bytes
-sebreplace <seb_name> <new_time> [<cutoff>]	Replaces the execution time of all SEBs with the name seb_name and time more than cutoff to new_time; cutoff defaults to 1000 ns

Simulator Performance Comparison

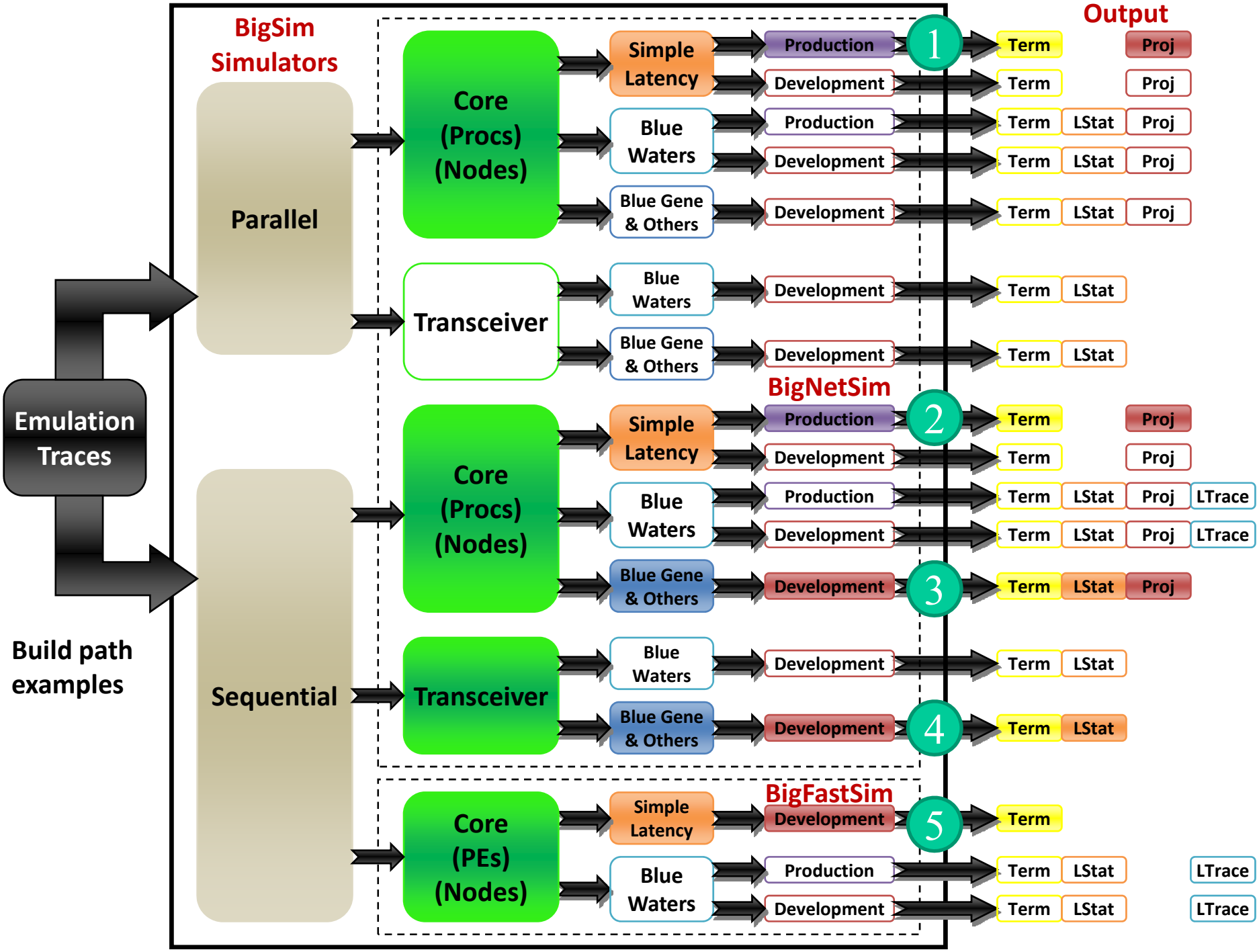
Sequential BigNetSim vs. BigFastSim: Simple Latency

Application	Machine	Total Run Time			Memory Footprint		
		BNS (s)	BFS (s)	BNS/BFS	BNS (MB)	Approx. BFS (MB)	BNS/BFS
AMPI All-to-All – 5 iters: 256 VPs (Skip Point 1)	PPL Nahalem Desktop	19.3	12.5	0.65	305	170	0.56
MILC: 4k VPs	Blue Print	17,080	14,210	0.83	2,334	N/A	N/A

Sequential BigNetSim vs. BigFastSim: Blue Waters

Application	Machine	Total Run Time			Memory Footprint		
		BNS (s)	BFS (s)	BNS/BFS	BNS (MB)	Approx. BFS (MB)	BNS/BFS
AMPI All-to-All – 5 iters: 256 VPs (Skip Point 1)	PPL Nahalem Desktop	143.3	52.3	0.36	310	190	0.61
MILC: 4k VPs	Blue Print	24,810	16,400	0.66	2,334	N/A	N/A

Note: The same build opts and equivalent command-line parameters were used for both simulators. Additionally, final virtual times differed by no more than a few percent.


BigSim Simulators
Output
Emulation Traces
Build path examples
Parallel
Sequential
Core (Procs) (Nodes)
Transceiver
Core (PEs) (Nodes)
Simple Latency
Blue Waters
Blue Gene & Others
Production
Development
BigNetSim
BigFastSim
1
2
3
4
5
Term
Proj
LStat
LTrace

Build Path Example 1: Parallel BigNetSim, Simple Latency, Production

1. Download charm

   http://charm.cs.uiuc.edu/download/

2. Compile POSE and BigSim

   cd charm

   ./build pose net-linux -O3

   ./build bigsim net-linux -O3

3. Download BigNetSim

   svn co https://charm.cs.uiuc.edu/svn/repos/BigNetSim

4. In BigNetSim/trunk/Makefile.common

   Point CHARMBASE to charm directory

   Set OPTS to -O3

5. Compile BigNetSim

   cd BigNetSim/trunk/SimpleLatency

   make PRODUCTION=1

6. Run on 4 PEs with link latency = 1μs and link bandwidth = 10 GB/s

   cd BigNetSim/trunk/tmp

   copy bgTrace files here

   ./charmrun +p4 ./bigsimulator -lat 1 -bw 10

Build Path Example 2: Sequential BigNetSim, Simple Latency, Production

1. Download charm

   http://charm.cs.uiuc.edu/download/

2. Compile POSE and BigSim

   cd charm

   ./build pose net-linux -g

   ./build bigsim net-linux -g

3. Download BigNetSim

   svn co https://charm.cs.uiuc.edu/svn/repos/BigNetSim

4. In BigNetSim/trunk/Makefile.common

   Point CHARMBASE to charm directory

   Set OPTS to -g

5. Compile BigNetSim

   cd BigNetSim/trunk/SimpleLatency

   make SEQUENTIAL=1 PRODUCTION=1

6. Run with link latency = 5μs, link bandwidth = 12 GB/s, and print summary debug info

   cd BigNetSim/trunk/tmp

   copy bgTrace files here

   ./bigsimulator -lat 5 -bw 12 -debuglevel 1

Build Path Example 3: Sequential BigNetSim, Blue Gene, Development

1. Download charm

   http://charm.cs.uiuc.edu/download/

2. Compile POSE and BigSim

   cd charm

   ./build pose net-linux -DCMK_OPTIMIZE=1

   ./build bigsim net-linux -DCMK_OPTIMIZE=1

3. Download BigNetSim

   svn co https://charm.cs.uiuc.edu/svn/repos/BigNetSim

4. In BigNetSim/trunk/Makefile.common

   Point CHARMBASE to charm directory

   Set OPTS to -DCMK_OPTIMIZE=1

5. Compile BigNetSim

   cd BigNetSim/trunk/BlueGene

   make SEQUENTIAL=1

6. Run with full contention model (1st param = 1) starting at skip point 2 (2nd param = 2)

   cd BigNetSim/trunk/tmp

   copy bgTrace files here; copy netconfig file from BigNetSim/trunk/BlueGene to here and modify

   ./bigsimulator 1 2

Build Path Example 4: Sequential BigNetSim, Transceiver, Blue Gene, Development

1. Download charm

   http://charm.cs.uiuc.edu/download/

2. Compile POSE and BigSim

   cd charm

   ./build pose net-linux -g

   ./build bigsim net-linux -g

3. Download BigNetSim

   svn co https://charm.cs.uiuc.edu/svn/repos/BigNetSim

4. In BigNetSim/trunk/Makefile.common

   Point CHARMBASE to charm directory

   Set OPTS to -g

5. Compile BigNetSim

   cd BigNetSim/trunk/BlueGene (any full network model will work; SimpleLatency won't)

   make SEQUENTIAL=1

6. Run with full contention model, deterministic traffic, ring pattern, 10 messages, 1024 bytes per message, and load factor of 0.1 (freq. of message sends)

   cd BigNetSim/trunk/tmp

   copy bgTrace files here; copy netconfig file from BigNetSim/trunk/BlueGene to here and modify

   ./bigsimulator 1 1 2 10 1024 0.1

Build Path Example 5: Sequential BigFastSim, Simple Latency, Development

1. Download charm

   http://charm.cs.uiuc.edu/download/

2. Compile POSE and BigSim

   cd charm

   ./build pose net-linux -O

   ./build bigsim net-linux -O

3. Download BigFastSim

   git clone charmgit:BigFastSim

4. In BigFastSim/Release/makefile

   Point CHARMPATH to charm directory

5. Compile BigFastSim

   make (in BigFastSim/Release)

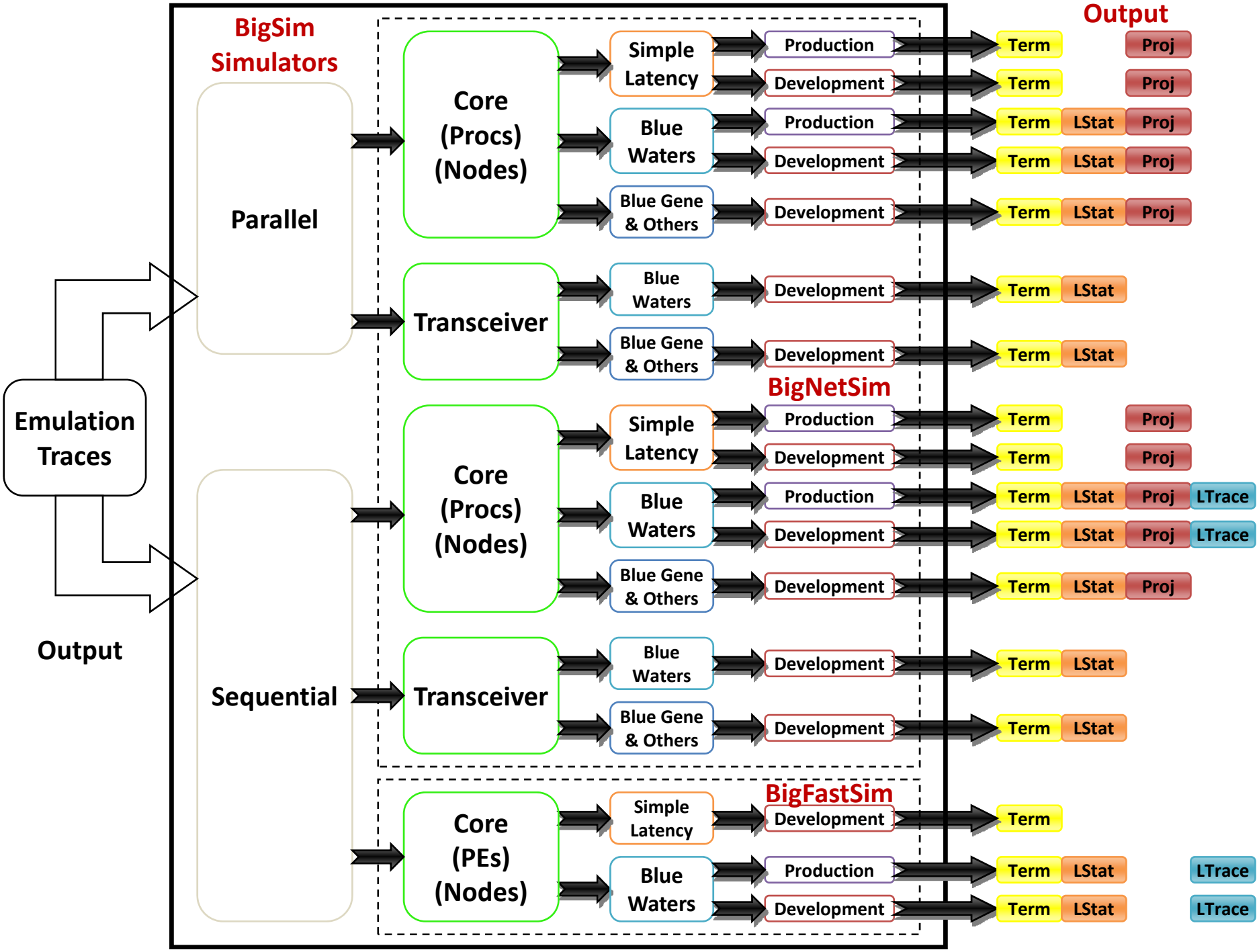
6. Run with link latency 1us, bandwidth 7 GB/s, window size 500, checking for unexecuted events, and starting at skip point 1

   copy bgTrace files here (BigFastSim/Release)

   ./seqSimulator -bw 7 -wsize 500 -check -skip_start 1

BigNetSim Output

- Four types of output
  - Terminal output
    - Final virtual time
    - BgPrintf statements
  - Link statistics
  - Projections logs
  - Link traces


BigSim Simulators
Output
Emulation Traces
Output
Parallel
Sequential
Core (Procs) (Nodes)
Transceiver
Core (Procs) (Nodes)
Transceiver
Core (PEs) (Nodes)
Simple Latency
Blue Waters
Blue Gene & Others
Blue Waters
Blue Gene & Others
Simple Latency
Blue Waters
Blue Gene & Others
Blue Waters
Blue Gene & Others
Simple Latency
Blue Waters
Production
Development
Term
LStat
Proj
LTrace
BigNetSim
BigFastSim

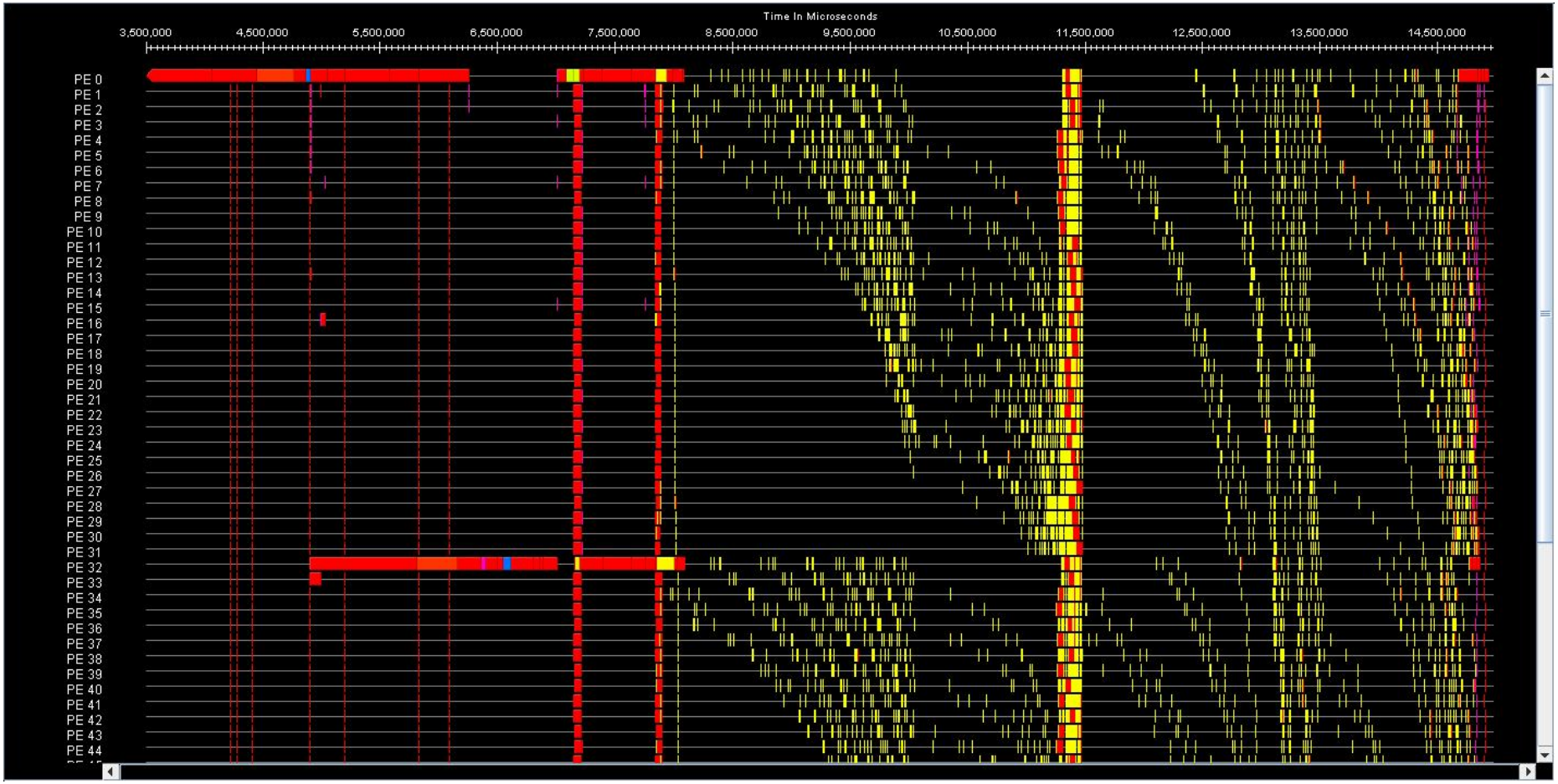
Terminal Output and Link Stats

```
Charm++: standalone mode (not using charmrun)
Charm++> Running on 1 unique compute nodes (8-way SMP).
================= Simulation Configuration ================
Production version: 1.0 (10/13/2010)
Simulation start time: Fri Oct 15 13:11:09 2010
Number of physical PEs: 1
POSE mode: Sequential
Network model: Blue Waters
...
===========================================================
Construction phase complete
Initialization phase complete
Info> invoking startup task from proc 0 ...
Info> Starting at the beginning of the simulation
Info> Running to the end of the simulation
Entire first pass sequence took about 18.532318 seconds
[0:user_code] #MILC# - WHILE Loop Iterarion Starting at 0.509469
[0:user_code] #MILC# - LL-Fat Starting at 0.510801
...
Sequential Endtime Approximation: 906988512
Final link stats [Node 0, Channel  0, LL Link]: ovt: 906953211,
   utilization time: 257562, utilization %: 0.028397, packets sent:
   2290 gvt=906988512
Final link stats [Node 0, Channel 11, LR Link]: ovt: 906953211,
   utilization time: 631426, utilization %: 0.069618, packets sent:
   1827 gvt=906988512
1 PE Simulation finished at 74.104628.
Program finished.
```

Projections Visualization

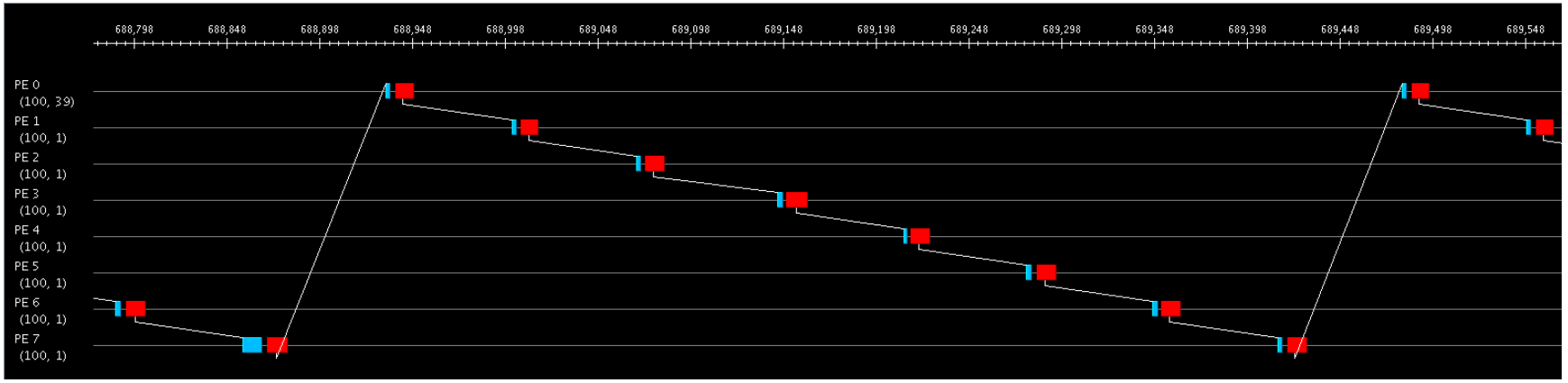
- Copy emulation Projections logs and sts file into directory with executable
  - Two ways to use:
    - Command-line parameter: -projname <name>
      - Creates a new set of logs by updating the emulation logs
      - Assumes emulation Projections logs are: <name>.*.log
      - Output: <name>-bg.*.log
      - Disadvantage: emulation Projections overhead included
    - Command-line parameter: -tproj
      - Creates a new set of logs from the trace files, ignoring the emulation logs
      - Must first copy <name>.sts file to tproj.sts
      - Output: tproj.*.log
      - Advantage: no emulation Projections overhead included

Projections – All-to-All

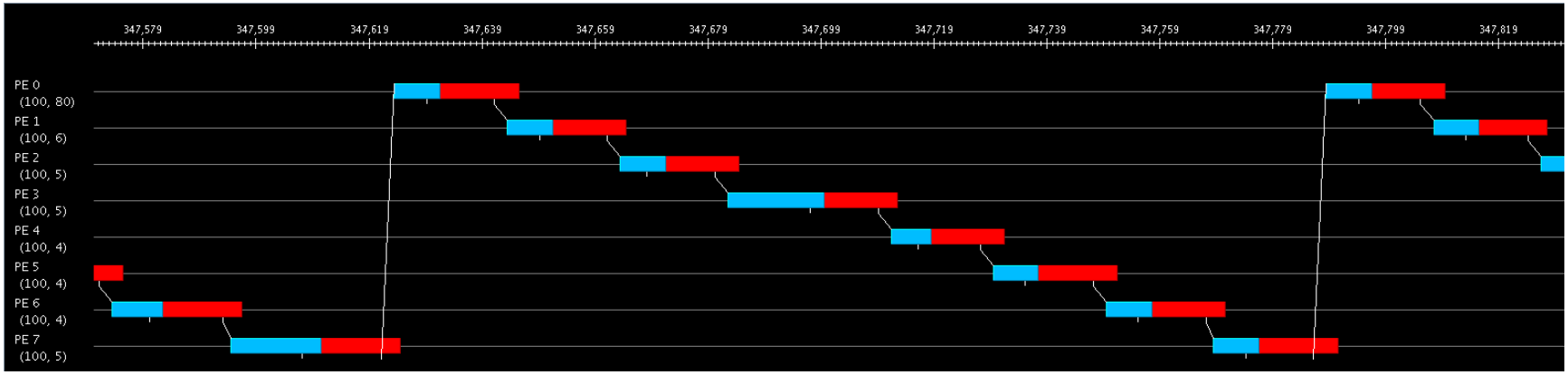


Projections – Ring

Emulation

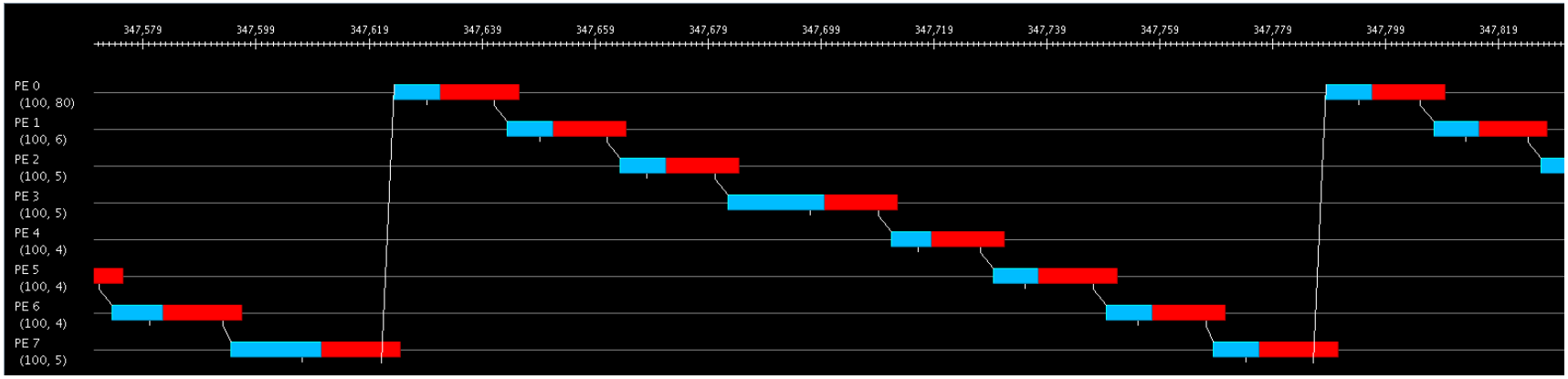


Simulation: -lat 1 (latency = 1μs) generated with -tproj

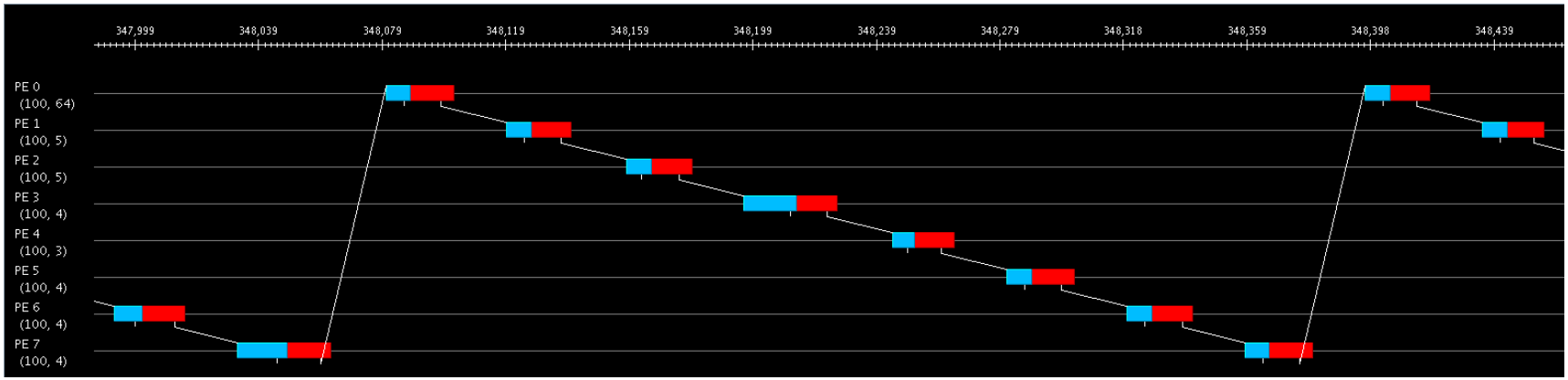


Projections – Ring

Simulation: -lat 1 (latency = 1μs) generated with -tproj



Simulation: -lat 20 (latency = 20μs) generated with -tproj



Link Tracing

- Printed link stats only give util % over entire run
- Trace usage of each link
  - Utilization
  - Saturation
  - Contention
- Only links with traffic are traced
- Disadvantages
  - Slows simulation and increases memory usage
- Only for Sequential Blue Waters model currently
  - Run with -traceutil, -tracesat, and -tracecont

Link Tracing Definitions

- Utilization – a link is in use whenever a packet is being transmitted over it
- Saturation – a link is saturated when the next packet won't fit in its output buffer
- Contention – a link is in contention when the next packet won't fit in its output buffer and it contains packets from more than one source node

LinkAnalyzer

- Tool for analyzing the binary link traces generated by BigNetSim
- Not in the public repository right now

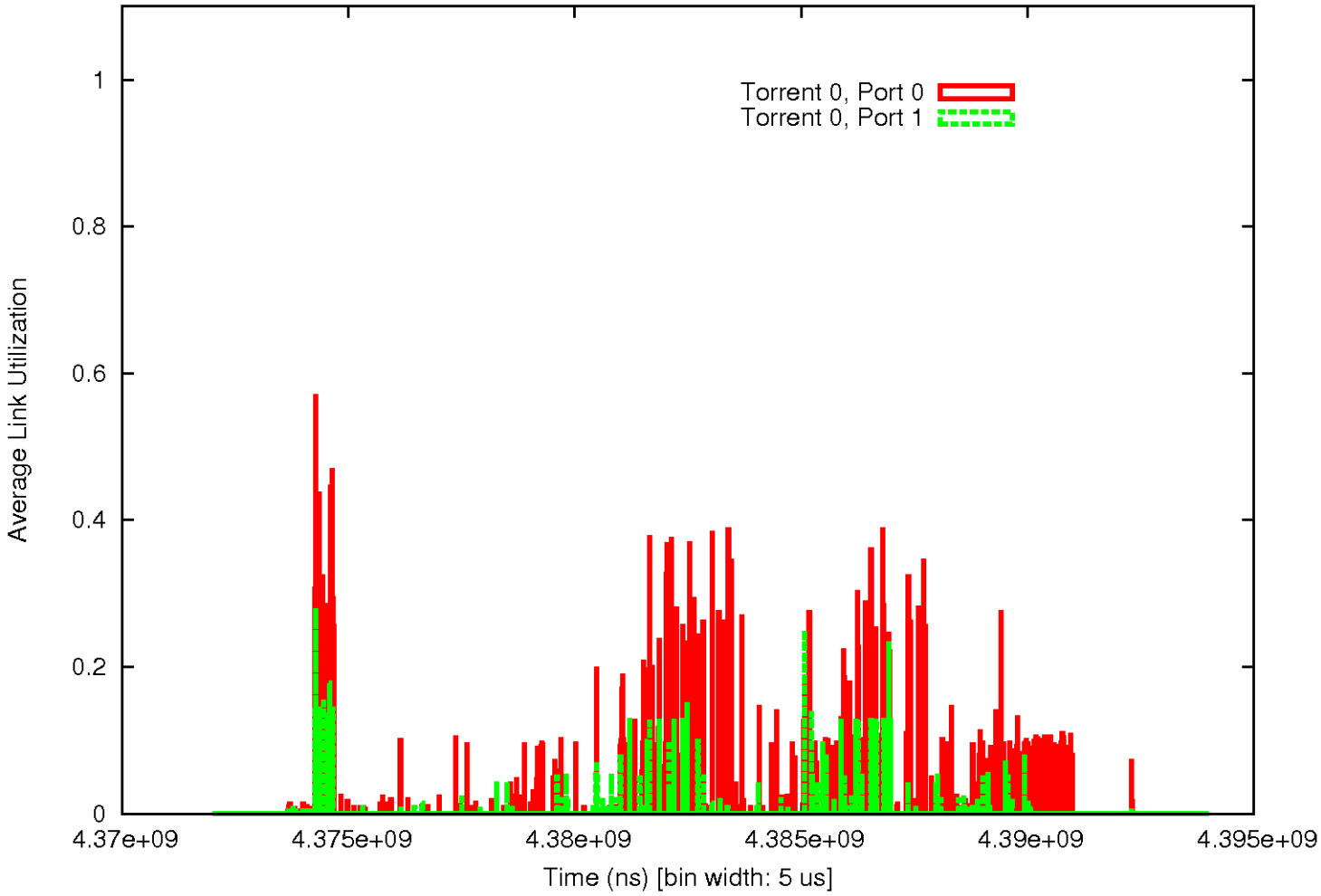
LinkAnalyzer Options

- Studies for all types (util, sat, cont)
  - Generate summary report (sorted link stats)
  - Convert binary traces to ASCII files
  - Individual (usage % vs. time) and composite (usage % vs. time over all links) link histograms
  - Generate degree-of-[util|sat|cont] for a select period of time (# links in use|contention|saturation vs. time over all links)
  - Same "degree-of-" study as the previous except for each node
- Calculate total bandwidth out of each node

LinkAnalyzer Summary Report

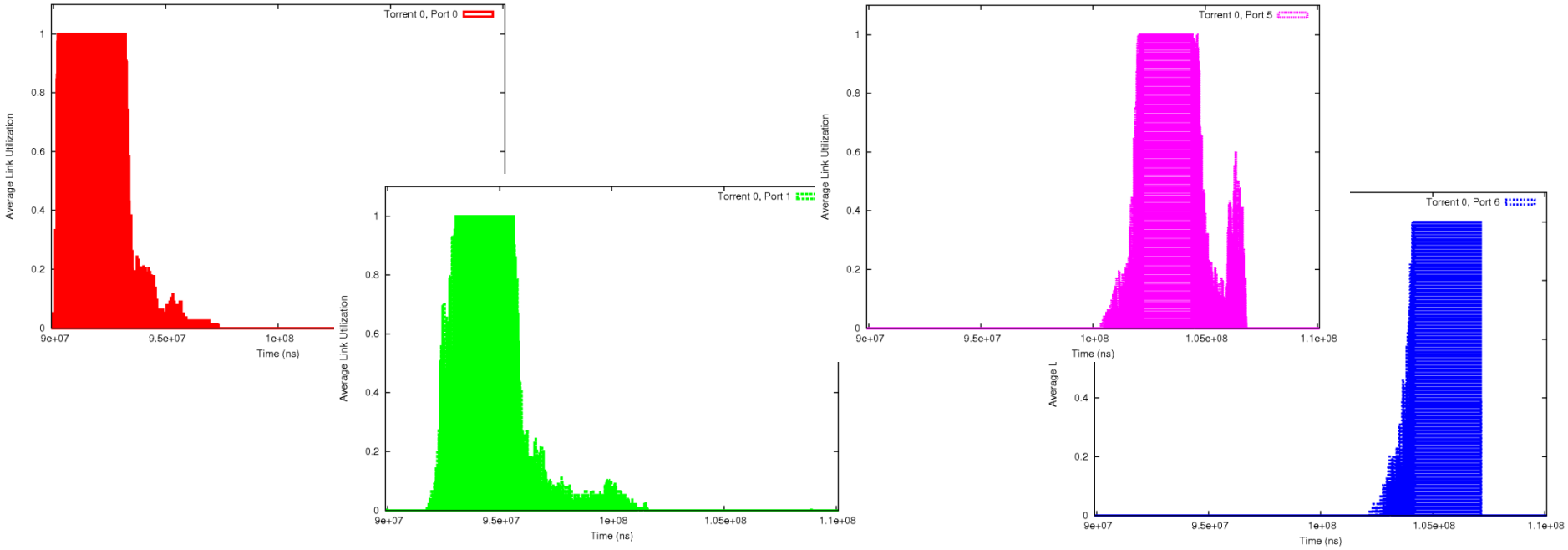
Node	Port	Link	util time	util %
31	30	LR	421888	35.999266
31	29	LR	421888	35.999266
31	28	LR	421888	35.999266
31	27	LR	421888	35.999266
31	26	LR	421888	35.999266
31	25	LR	421888	35.999266
...				
0	9	LR	421888	35.999266
0	8	LR	421888	35.999266
0	7	LR	421888	35.999266
31	6	LL	132096	11.271615
31	5	LL	132096	11.271615
31	4	LL	132096	11.271615
31	3	LL	132096	11.271615
31	2	LL	132096	11.271615
...				

Link Utilization Histogram – NAMD (5µs bins)



Link Trace Use Case – All-to-All

- Link utilization histograms (10µs bins) for different LL links
- Show inefficient link usage



Extensibility

- Other network models can be plugged into the main network framework in BigNetSim
  - Routing algorithm
  - Topology
  - Input and output virtual channel selection strategies
- See 2008 and 2009 BigSim tutorials for more info (see last slide for URLs)

Additional Resources

- BigSim Manuals: http://charm.cs.uiuc.edu/manuals/
- Online Tutorial: at NCSA's Web-based CI-Tutor
- Recent Charm++ Workshop Tutorials and Talks
  - 2008 BigSim tutorial (bottom of page)
    - http://charm.cs.illinois.edu/workshops/charmWorkshop2008/slides.html
  - 2009 BigSim tutorial (bottom of page)
    - http://charm.cs.uiuc.edu/workshops/charmWorkshop2009/program.html
  - 2010 BigSim talk (near top of page)
    - http://charm.cs.uiuc.edu/charmWorkshop/program.php
  - 2010 PRAC Workshop talk
    - http://charm.cs.uiuc.edu/talks/BigSimPRAC10.ppt
- E-mail PPL for help: ppl@cs.uiuc.edu

BigSim Support

- NSF Grants:
  - NGS 0103645
  - CSR-SMA-0720827
- NSF/NCSA Blue Waters Grant:
  - OCI-0725070
- Machine Time:
  - TeraGrid machines: allocation TG-ASC050039N
  - NCSA machines: BlueWaters' project allocation