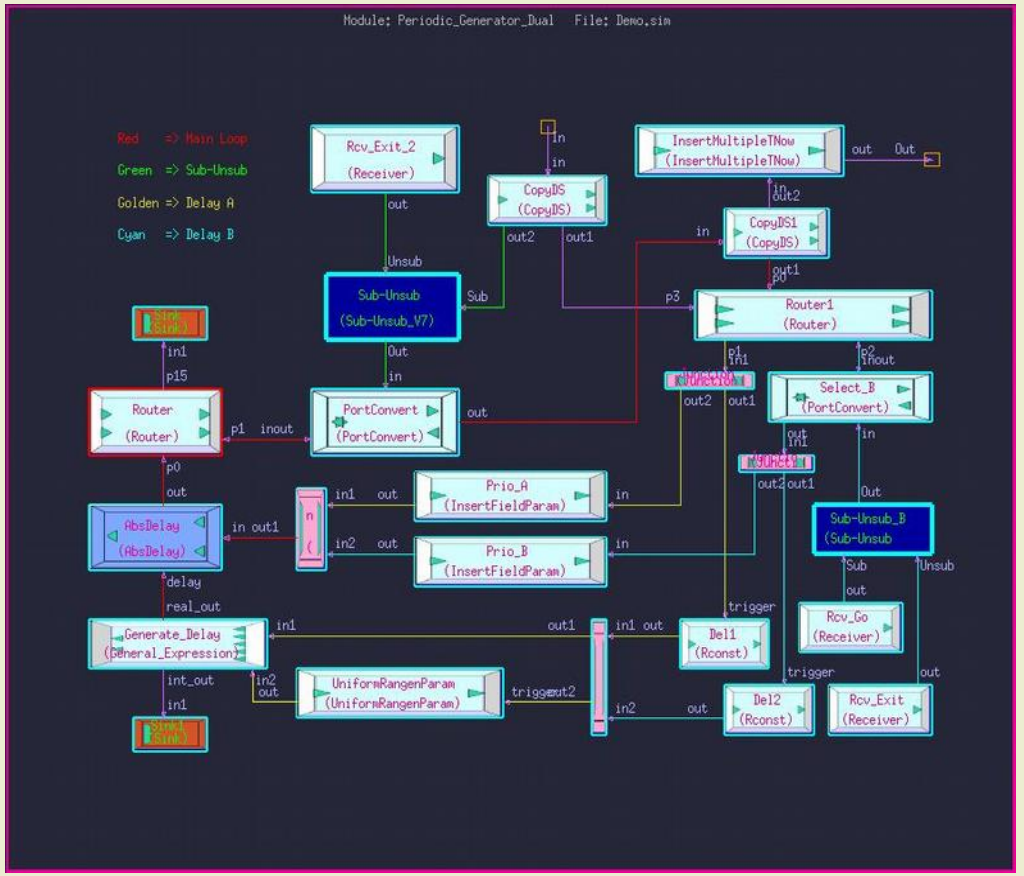
CSIM's General Blocks Library

Jan 2017

Outline

- History
- Why General Blocks?
- Advantages
- Disadvantages
- Status
- Library Description
- Example simulations



History

- The General Blocks library was developed as a replacement for BONeS (Block Oriented Network Simulator)
  - BONeS was developed at the University of Kansas, and was commercially available from ~1988-1999 (Comdisco/Alta/CADence)
  - BONeS was used by a significant community
  - The General Blocks library was initially developed 1999-2002. Significant enhancements have occurred since, and are ongoing.

Why Use General Blocks?

- Advantages of the General Blocks library
  - Ability to leverage significant residual BONeS expertise, compare known results
  - The block oriented approach (>330 mostly small, simple blocks) enables fine granularity in architecture definition and tracing
  - Extremely flexible; can easily implement new modules that would be more difficult in other model libraries

Advantages

- Sophisticated models for server resources
  - Priority, preemption, round robin
- Popups provide message details for selected blocks
- Extremely flexible mechanism for representing messages (data_structs.txt)
- Minimum need to become involved with C code
- Many built-in statistical models and display mechanisms
- Significant upgrades recently to help accelerate the initial design/debug cycle
- Useful for modeling data-processing systems, or other systems (ex. not signal processing), not covered by DFG modeling methods.

Disadvantages

- Does not inherently separate hardware from software
  - Cannot use DFG (Data Flow Graph) Schedulers.
- Can be computationally inefficient for large models (flip side of *flexible*)
- Oriented toward static topologies
- Not specialized for modelling specific kinds of systems.

CSIM: an Open Architecture Tool

- CSIM is based upon a “toolbox” approach
  - “CSIM” is actually the assembly of many independent tools and libraries; it is not a monolithic (“stovepipe”) chunk of code
  - The key independent tools/libraries include:
    - CSIM precompiler
    - CSIM kernel library
    - GUI
    - Simview
    - XGraph
    - NumUtils, general_utils
    - The Model Libraries

CSIM: an Open Architecture Tool

- CSIM is based upon a “toolbox” approach
  - CSIM leverages the existence of available applications, tools, utilities and standards
    - Minimizes CSIM-specific development, maintenance and documentation (“avoid re-inventing the wheel”)
    - Examples of applications/tools/libraries leveraged
      - Compilers (cc, gcc, etc.)
      - Debuggers (gdb, ddd, etc.)
      - Text editors (vi, emacs, wordpad, textedit, etc.)
      - Libraries (C language, GTK, OpenGL, Motif, OTK, etc.)
      - Graphical viewers/editors (xv, gimp, etc.)
      - Data standards (xml, xpm, etc.)

Application

- Typically, the General Blocks Library is used to model and simulate networked computer resources to:
  - Identify points of contention
  - Estimate performance limits or bottlenecks
  - Evaluate processor utilizations
  - Evaluate system latencies
  - etc.
- The types of outputs typically obtained include:
  - Scatter plots, histograms and statistical measures of latency data

Status

- The General Blocks library currently contains more than 330 models in the following groups:
  - * Arithmetic
  - * Comparison
  - * Conversions
  - * Counters
  - * Data Type Operations
  - * Data Structure Access
  - * Delays
  - * Execution Control
  - * File Access
  - * Generators
  - * Logical
  - * Loops
  - * Memory
  - * Miscellaneous
  - * Plot Generation
  - * Quantity Shared Resource
  - * Queues And Servers
  - * Probes
  - * Queues
  - * Servers
  - * Statistics
  - * Switches
  - * Timers
  - * Traffic Generators

Most Recent Additions

- Additional models have been added to the library
- These new models include:
  - Admin
  - Append_Route_List
  - Append_String
  - Generic_Batcher
  - Generic_UnBatcher
  - LockRealTime
  - Num_to_String
  - PlotLive
  - PortConvert
  - QSR1
  - QSR2
  - Receiver
  - Router
  - Sender
  - Switch_5way

General Blocks Library Devices

New_Models
- Generic_Batcher
- Generic_UnBatcher
- Receiver
- Sender
- PlotLive
- LockRealTime
- QSR1
- QSR2
- Switch_5way

Vectors
- VCreate
- Setup_VElem
- VLen
- Access_Vector
- GVCreate
- Setup_GVElem
- GVLen
- Access_GVector

Traffic_Generators
- Uniform_PulseTrain
- Poisson_PulseTrain
- Enabled_Uniform_PulseTrain
- Enabled_Poisson_PulseTrain
- Enabled_PulseTrain
- Arbitrary_PulseTrain

Timers
- Start_Timer
- Set_Alarm
- Service_Timer
- Residual_Time
- Reset_Timer
- Cancel_Timer
- Cancel_Alarm
- Alarm_Active

Switches
- True_N_Times

T_GT_Startup
T_GE_ParamSwitch_4way
Switch
Real_Within_Boundaries
Rand_Switch_Param
Rand_Switch
R_LT_C
R_LE_C
R_GT_C
R_GE_C
R_EQ_C
MemorySwitch
I_LT_C
I_LE_C
I_GT_C
I_GE_C
I_EQ_C
Enabled_Switch
Bypass

Statistical
- WeightedMeanAndVariance
- Weighted_General_Moments
- Throughput
- Time_Average
- MeanAndVariance
- Histogram
- Global_Statistics
- General_Nth_Moment
- Find_Bin
- Dimensioned_Time_Average
- Dimensioned_Ensemble_Average
- Construct_TimeAverage_Stats
- Construct_Dimensioned_Stats
- Batch_Timing
- Batch_Statistics
- Batch_Rmin
- Batch_Rmax
- Batch_Mean
- Average

Server_Resource
- SR_Server_Utilization_Probe
- SR_Server_Utilization_Per_Priority_Probe
- SR_Server_Response_Probe
- SR_Server_Occupancy_Probe
- SR_Preempt_Server_Utilization_Probe
- SR_Preempt_Server_Utilization_Per_Priority_Probe
- SR_Preempt_Server_Response_Probe
- SR_Preempt_Server_Occupancy_Probe
- Set_Resource
- Set_Preempt_Resource
- Service_wRoundRobin
- Service_wPriority_Preemption
- Service_wPriority

QueuesAndServers
- FIFOwServers
- MultipleServers
- ParallelQueues
- PQwServers

Queues
- Simple_LIFO
- Simple_FIFO
- FIFOwPriority
- FIFO_wPeek

QuantityShared_Resource
- Set_QResource
- FreeBasic
- Free
- ConsumeResourceUnits
- ChangeCapacity
- AllocatePriority
- AllocateParam
- AllocateBasic
- Allocate

Probes
- WriteTnow
- ThroughputDelayProbe
- ThroughputVsTimeProbe
- TextualDescriptionProbe
- SystemLatencyProbe
- ScatterPlotZ
- ScatterPlotQ
- ScatterPlot
- SelectFieldProbe
- RealvsTimeProbe
- ProcessTimeLineProbe
- InsertStatFields
- HistogramProbeF2_F1
- HistogramProbe
- GenericProbe
- GenericHyperGraphProbe
- EventProbe_with_Comm
- EventProbe
- CreateCDFfileInit
- CreateCDFfileF2_F1
- CreateCDFfile
- BatchStatisticsProbe_f2_f1
- BatchStatisticsProbe
- BatchNthMomentProbe_f2_f1
- BatchNthMomentProbe
- BatchMeanProbe_f2_f1
- BatchMeanProbe

Plot_Generation
- BuildPlot_Ytime
- BuildPlot_Yonly
- BuildPlot_Y
- BuildPlot_XY
- BuildPlot
- BuildHistogram

Number_Generators
- UserCDF_RanGen
- UniformRangenParam
- UniformRangen
- U_0_to_1_RanGen
- TStop
- TNow
- Rconst
- PoissonRangenParam
- PoissonRangen
- N01_Rangen
- NormalRangen
- NormalRangenParam
- IU_Parem
- IU_NE_C
- IU_MinMax_Param
- IU_MinMax
- IU
- Iconst
- GammaRangenParam
- GammaRangen
- ExponRanGenParam
- ExponRanGen
- BinomialRangenParam
- BinomialRangen

Miscellaneous
- TimeBetweenTriggers
- SystemCall
- ServiceSetup
- Print_message
- PrintEnvelope
- Print_real
- Print_int
- Dijkstra
- Central_Utilities
- Ack_Setup

General Blocks Library Devices II

Number_Generators
- UserCDF_RanGen
- UniformRangenParam
- UniformRangen
- U_0_to_1_RanGen
- TStop
- TNow
- Rconst
- PoissonRangenParam
- PoissonRangen
- N01_Rangen
- NormalRangen
- NormalRangenParam
- IU_Parem
- IU_NE_C
- IU_MinMax_Param
- IU_MinMax
- IU
- Iconst
- GammaRangenParam
- GammaRangen
- ExponRanGenParam
- ExponRanGen
- BinomialRangenParam
- BinomialRangen

Miscellaneous
- TimeBetweenTriggers
- SystemCall
- ServiceSetup
- Print_message
- PrintEnvelope
- Print_real
- Print_int
- Dijkstra
- Central_Utilities
- Ack_Setup

Memory
- WriteMemory
- RealLocalMem
- ReadMemory
- MultipleBuffers
- Mem_increment
- Mem_decrement
- LocalMem_wCopy
- LocalMemRef
- LocalMem
- IntLocalMemory
- ActiveReadMemory

Loops
- Real_Do_Param
- Real_Do
- Int_Do_Param
- Int_Do_1_N
- Int_Do_0_Nminus1
- Int_Do

Logical
- False
- True
- Nxor
- Xor
- Nor
- Nand
- Not
- Or
- And

Graphical_Interface
- Slider_box
- PromptInt
- PromptFloat
- PopUpMessage
- Navigate_View
- MPGraph
- Hilite_Box
- GenericProbePopup
- ColorController
- ColorBox
- Button_box

File_Access
- WriteInfo_Numeric
- WriteFile_String
- WriteFile_Real
- WriteFile_Field
- WriteFile_AppendField
- WriteFile_Int
- ReadFile_String
- ReadFile_Real
- ReadFile_Line
- ReadFile_Int
- OpenFileWrite
- OpenFileRead
- OpenFileAppend
- CloseFile

Execution_Control
- Wrapup
- Terminate
- OneWay
- OnePulse
- Merge
- Init
- Gate_Switch
- Gate
- Execute_in_order_4
- Execute_in_order_3
- Execute_in_order
- Control_Signal_Generator

Delays
- FixedProcDelay
- FixedAbsDelay
- AbsDelay

Data_Structure_Access
- TypeSwitch
- SelectField
- MakeRealDS
- InsertTNow
- InsertMultipleFieldParams
- InsertMultipleTNow
- InsertFieldTNow
- InsertFieldParam
- InsertField
- Declare_DS
- Create_DS
- Coerce_DS

Data_Structure_Operations
- TypeOf
- TypeConst
- TypeCompatible
- Tequals
- Split_wDelay
- Split3
- Split
- Sink
- Junction
- Join
- Copy2
- CopyDS_wDelay
- CopyDS

Counters
- UpDownCounterChangeValue
- UpDownCounter
- SimpleCounter
- Int_Accumulator
- GlobalCount
- Counter
- CircularCounter
- Accumulator

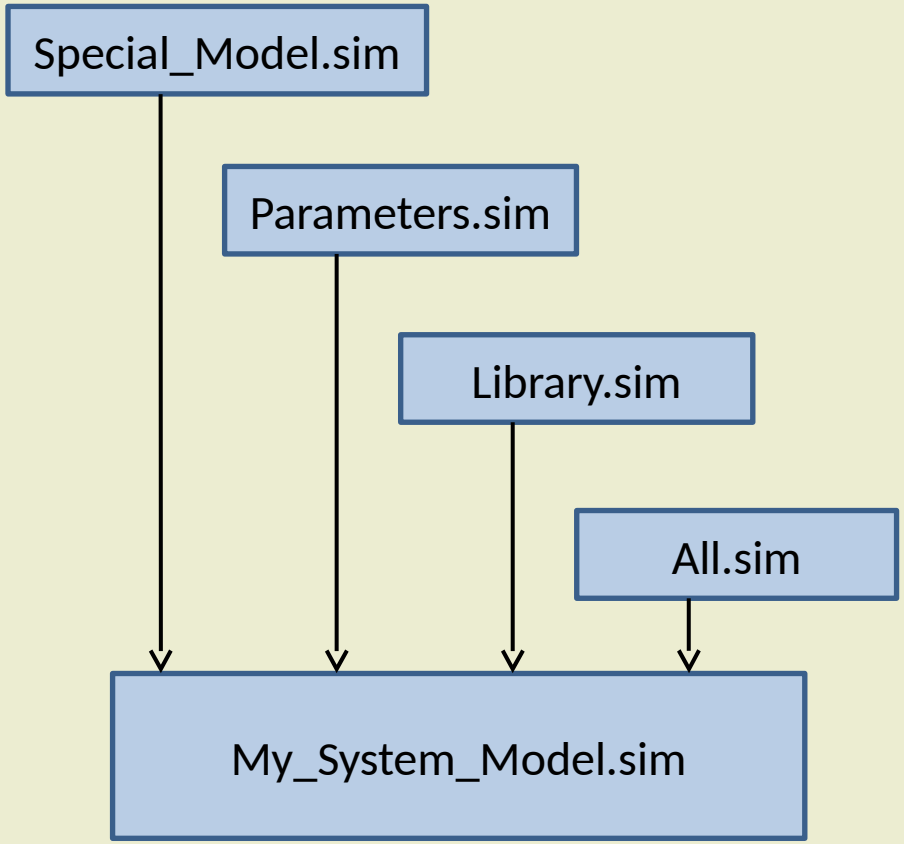
Conversions
- Truncate
- Round
- Int_to_Real

Comparison
- StringEqualsParam
- Set_Equals
- R_LessThanOrEqual
- R_LessThan
- R_GreaterThanOrEqual
- R_GreaterThan
- R_Equals
- Odd
- I_LessThanOrEqualE
- I_LessThan
- I_GreaterThanOrEqual
- I_GreaterThan
- I_Equals
- Even

Arithmetic
- Increment
- Decrement
- I_add
- I_subtract
- Imult
- I_mult
- I_div
- I_divprotect
- Imod
- I_mod
- Iabs
- Imin
- Imax
- Ichs
- Igain
- R_add
- R_subtract
- R_mult
- R_div
- R_divprotect
- Rsqrt
- Rabs
- Rmin
- Rmax
- Rchs
- Rgain
- sin_X
- cos_X
- tan_X
- ln_X
- exp_X
- X_powr_Iconst
- X_powr_Y
- five_input_expression
- one_input_expression_R
- one_input_expression_I
- Rlimiter
- Ilimiter
- Reciprocal
- General_Expression

Library Configuration

- General Blocks based simulations generally utilize several libraries
- All.sim contains the basic elements (devices) of the General Blocks library.
- Library.sim contains information to group the All.sim models into manageable hierarchical groups
- One or more local libraries, containing module level and sometimes device level models, are generally referenced
- The User's simulation model will reference these libraries



General Blocks (GB) Files

- *Library.sim* is used to organize the models into logical groupings for display and access by the CSIM gui
- *All.sim* contains the detailed implementation code for all of the models in the distributed GB library
- *data_structs.txt* contains the definitions for all compound data structures (message definitions) that will be used in simulation
- All simulations will require an *All.sim* and a *data_structs.txt*; *Library.sim* is optional (although very useful).
- task_table.dat is required for simulations which require the Admin model
- CSIM will provide additional object files.

Excerpts from Library.sim

- %include $CSIM_ROOT/model_libs/general_blocks/All.sim

- <DEFINE_LIBRARY> Counters
- <MODEL> UpDownCounterChangeValue </MODEL>
- <MODEL> UpDownCounter </MODEL>
- <MODEL> SimpleCounter </MODEL>
- <MODEL> Int_Accumulator </MODEL>
- <MODEL> GlobalCount </MODEL>
- <MODEL> Counter </MODEL>
- <MODEL> CircularCounter </MODEL>
- <MODEL> Accumulator </MODEL>
- </DEFINE_LIBRARY>

- <DEFINE_LIBRARY> Conversions
- <MODEL> Truncate </MODEL>
- <MODEL> Round </MODEL>
- <MODEL> Int_to_Real </MODEL>
- </DEFINE_LIBRARY>

- <DEFINE_LIBRARY> Comparison
- <MODEL> StringEqualsParam </MODEL>
- <MODEL> Set_Equals </MODEL>
- <MODEL> R_LessThanOrEqual </MODEL>
- <MODEL> R_LessThan </MODEL>
- <MODEL> R_GreaterThanOrEqual </MODEL>
- <MODEL> R_GreaterThan </MODEL>
- <MODEL> R_Equals </MODEL>
- <MODEL> Odd </MODEL>
- <MODEL> I_LessThanOrEqualE </MODEL>
- <MODEL> I_LessThan </MODEL>
- <MODEL> I_GreaterThanOrEqual </MODEL>
- <MODEL> I_GreaterThan </MODEL>
- <MODEL> I_Equals </MODEL>
- <MODEL> Even </MODEL>
- </DEFINE_LIBRARY>

- <DEFINE_LIBRARY> Plot_Generation
- <MODEL> BuildPlot_Ytime </MODEL>
- <MODEL> BuildPlot_Yonly </MODEL>
- <MODEL> BuildPlot_Y </MODEL>
- <MODEL> BuildPlot_XY </MODEL>
- <MODEL> BuildPlot </MODEL>
- <MODEL> BuildHistogram </MODEL>
- </DEFINE_LIBRARY>

- <DEFINE_LIBRARY> Number_Generators
- <MODEL> UserCDF_RanGen </MODEL>
- <MODEL> UniformRangenParam </MODEL>
- <MODEL> UniformRangen </MODEL>
- <MODEL> U_0_to_1_RanGen </MODEL>
- <MODEL> TStop </MODEL>
- <MODEL> TNow </MODEL>
- <MODEL> Rconst </MODEL>
- <MODEL> PoissonRangenParam </MODEL>
- <MODEL> PoissonRangen </MODEL>
- <MODEL> N01_Rangen </MODEL>
- <MODEL> NormalRangen </MODEL>
- <MODEL> NormalRangenParam </MODEL>
- <MODEL> IU_Parem </MODEL>
- <MODEL> IU_NE_C </MODEL>
- <MODEL> IU_MinMax_Param </MODEL>
- <MODEL> IU_MinMax </MODEL>
- <MODEL> IU </MODEL>
- <MODEL> Iconst </MODEL>
- <MODEL> GammaRangenParam </MODEL>
- <MODEL> GammaRangen </MODEL>
- <MODEL> ExponRanGenParam </MODEL>
- <MODEL> ExponRanGen </MODEL>
- <MODEL> BinomialRangenParam </MODEL>
- <MODEL> BinomialRangen </MODEL>
- </DEFINE_LIBRARY>

Example Model from All.sim

```
DEFINE_DEVICE_TYPE:   R_add
  PORT_LIST( in1, in2, out );
  DOCUMENTATION:
   /*****************************************************************/
   /* The model adds the value of in1 and in2                       */
   /* Input Ports                                                   */
   /*  in1     Data Type: REAL                                      */
   /*  in2     Data Type: REAL                                      */
   /* Output Ports                                                  */
   /*  out     Data Type: REAL                                      */
   /* Parameters( none )                                            */
   /*****************************************************************/
  END_DOCUMENTATION.
  DEFAULT_ICON( $CSIM_MODEL_LIBS/general_blocks/Icons/2_1.ppm );

  DEFINE_THREAD:   start_up`
   {
    Envelope *a, *b;
    float x, y;     in len;

    while (1)
     {
      RECEIVE( "in1", &a, &len );
      x = consume_real(a);
      RECEIVE( "in2", &b, &len );
      y = consume_real(b);
      x = x + y;
      a = make_real_envelope( x );
      SEND( "out", a, 1 );
     }
   }
  END_DEFINE_THREAD.

END_DEFINE_DEVICE_TYPE.
```

General Blocks Messages

- Data structures are used to represent messages.
- In the General Blocks Library, there can be several types of messages
  - "Simple" data structures definitions are built-in
    - Int, real, string
  - Compound data structures are defined by the user in the ***data_structs.txt*** file
    - Assemblies of simple data structures
- Typically, data structures contain several fields:
  - Some may contain information about the message, i.e. message size, message priority, message creation time
  - Others may be used to hold information about the system state, probe data, calculation results, etc.
- Some General Blocks "devices" (i.e. Built-in models) operate with compound data structures
  - Others require specific simple data structures
- User models may require specific compound data structures

Example *data_structs.txt*

```
<DEFINE_DATA_STRUCTURES>

struct Throughput_Delay_DS
{
  real Mean_Delay=0
  real Var_Delay=0
  real Mean_Throughput=0
  real Var_Throughput=0
  int Nsamples
}

struct Basic_Statistic
{ real mean
   real variance
   real min
   real max
   int  Nsamples=0
}

struct Timing_Packet
{ real Time_Created
  real Intermediate_Time
  real Time_Finished
  int Length
  int Type
}

struct Event_Data
{ real EVENT_START_TIME=0
   int EVENT_SEQUENCE_NUMBER=0
   int EVENT_TYPE_PARAMS_INDEX=0
   real PREV_LINKED_EVENT_START_TIME=0
   int PREV_LINKED_EVENT_SEQ_NUMBER=0
   int SOFT_RESET_COMMAND=0
   real EVENT_LENGTH_X_100_NSEC=1000
}

struct Application_Message_Transaction_DS
{ int Application_Message_Type_Code=0
   int Application_Message_Sequence_Number=0
   int Application_Message_Source=0
   int Application_Message_Destination=0
   int Application_Message_Size_Bytes=10
   int Application_Message_Priority=0
   real Application_Message_Create_Time=0
   real Application_Message_Start_XMIT_Time=0
   real Application_Message_Complete_XMIT_Time=0
   real Application_Message_RCV_Complete_Time=0
   real Application_Message_Destination_Time=0
   Event_Data Application_Message_User_Data
   gvec My_Vector_Data
}

</DEFINE_DATA_STRUCTURES>
```

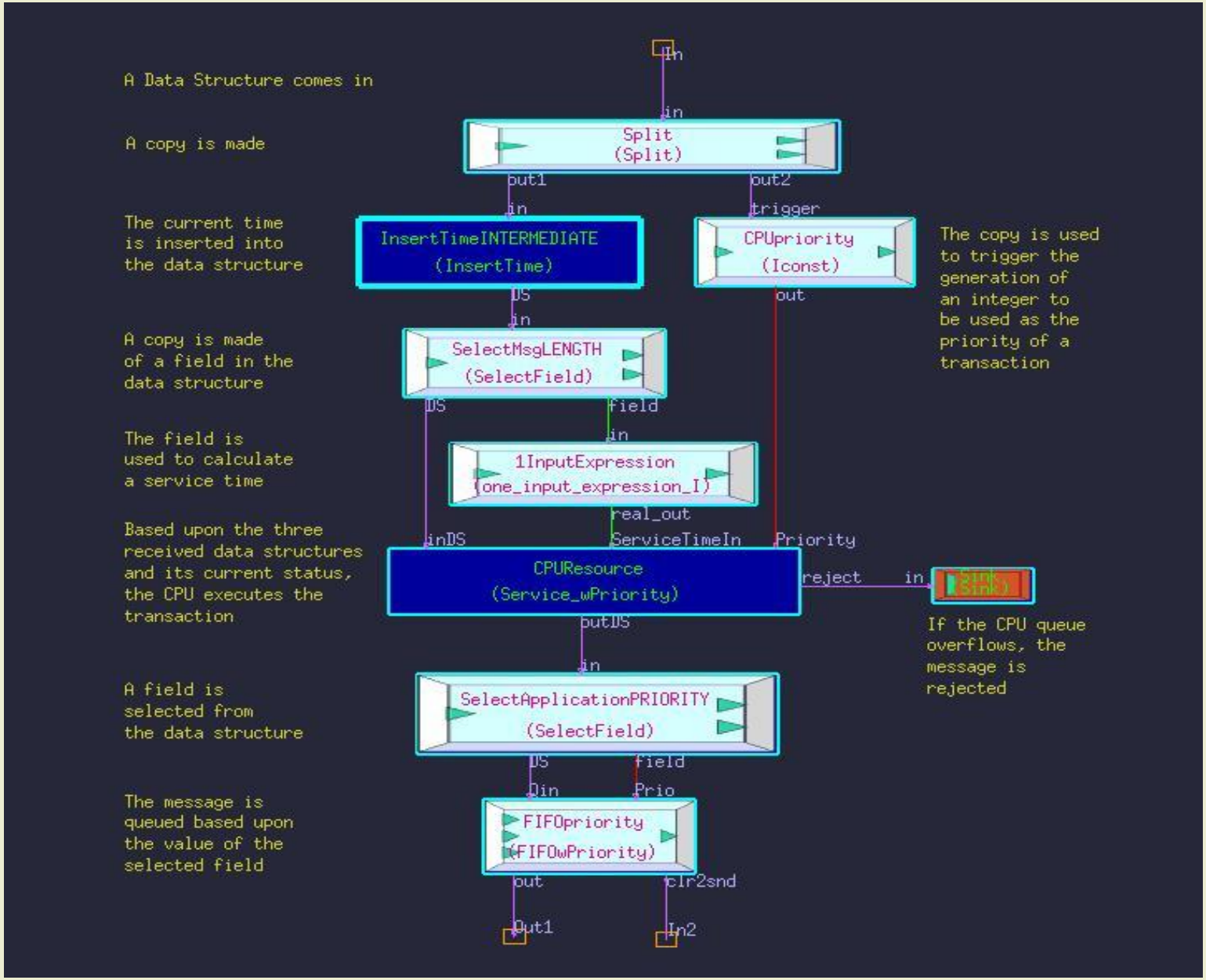
Creating Legible Models

- General “rules”
  - Limit the number of boxes to about 15
  - Orient the flow to run top to bottom rather than left to right
  - Maximize use of the available canvas
  - Jog wires to improve signal legibility

Example of Message Flows

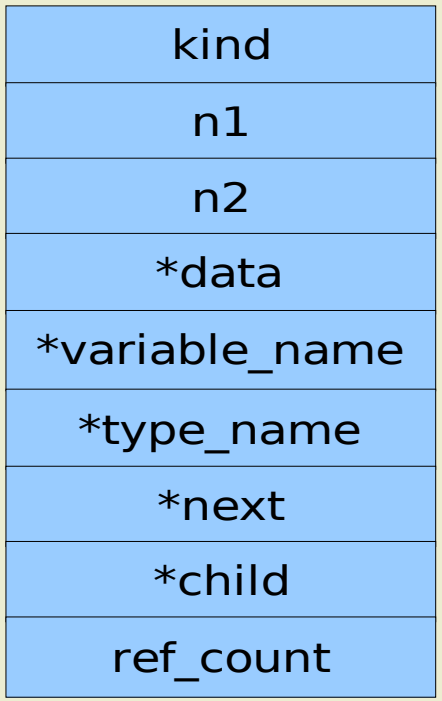
- The compound data structure used here is:

```
<DEFINE_DATA_STRUCTURES>
struct CompuSys
{
  char MsgType=Heartbeat
  char StackACK
  char ACK=NoACK
  int  NUMBER
  int  MsgLENGTH
  int  PRIORITY
  real CREATED
  real COMPLETED
  real MEAN
  real EARLIEST
  real LATEST
  real INTERMEDIATE
}
</DEFINE_DATA_STRUCTURES>
```



Data Structures Approach

- The Data_Type_Container (Envelope) is the atomic component of data structures for the general blocks library
- Compound data structures are built from linked lists of Envelopes
- Organization of an Envelope:



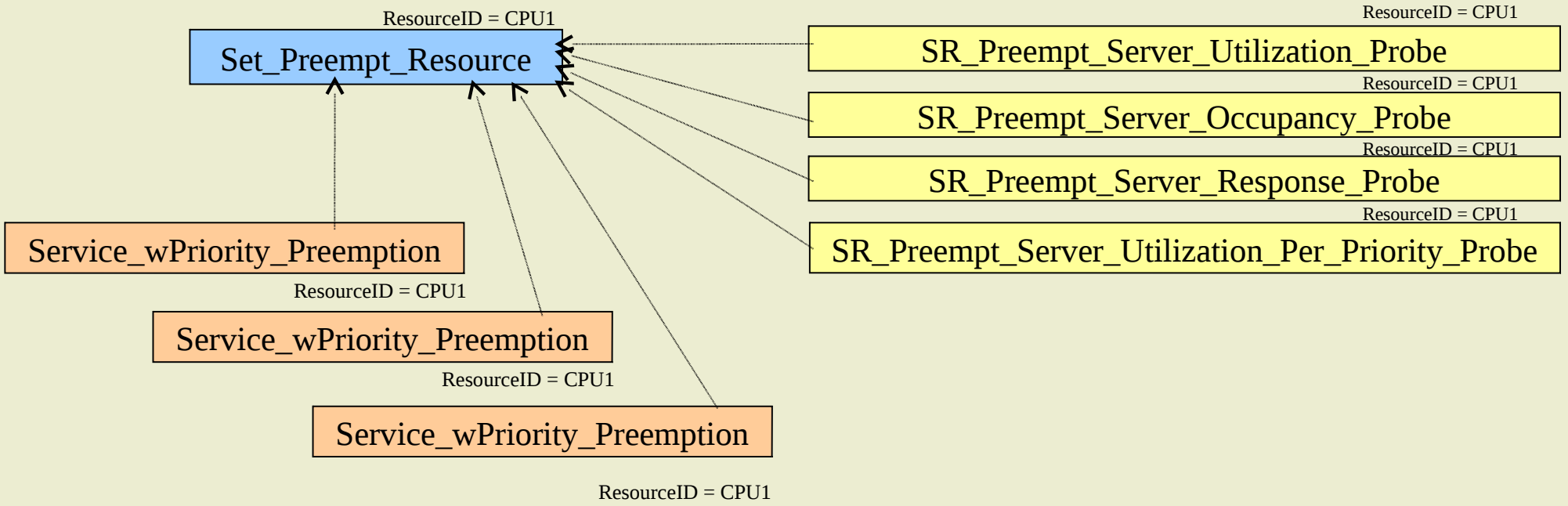

```
struct Data_Type_Container
  {
   int    kind, n1, n2;         /* Type and dimension(s).
*/
   void   *data;
   char   *variable_name, *type_name;
   struct Data_Type_Container *next, *child;
   int ref_count;
  } *DATA_STRUCTURE_DEFINITIONS=0;


typedef struct Data_Type_Container Envelope;
```

Copying Messages

- There are two methods for copying (splitting) messages (data structures)
  - Pass a pointer (very fast)
  - Make a deep copy of the data structure (can be slow)
- Different models use one or the other approach (i.e. Junction uses pointers, Copy_DS makes a deep copy)
- Deep copying may be required if both copies of the DS will be modified
- Pointer copying may be used if the copy is only being used as a trigger (for example)

Resources, Servers and Probes



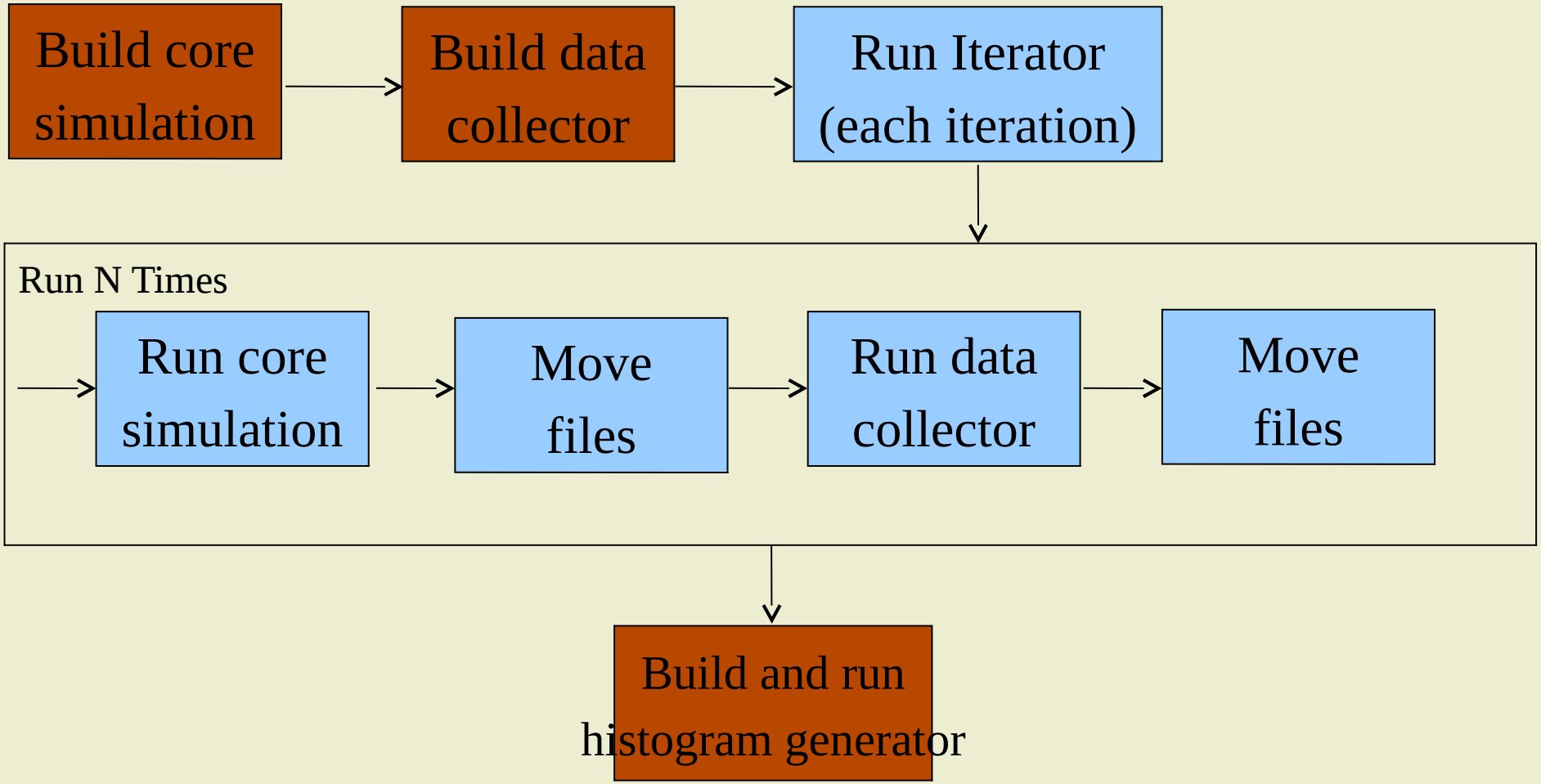
- The properties of a Resource (i.e. CPU) are defined using a Set_Resource device
- Many (i.e. hundreds) of Servers (i.e. Service_wPriority_Preemption) may be mapped to a single Resource
- An individual Server is often used to represent the execution of a particular piece of software
- The correlation between resources, servers and probes is set by the ResourceID attribute

- Up to four Probes (as shown) may be attached to a given Resource
- The Utilization probes output two files:
  - Batched and global utilization
- The other probes each output four files:
  - Batched and global average
  - Batched and global peak

Examples

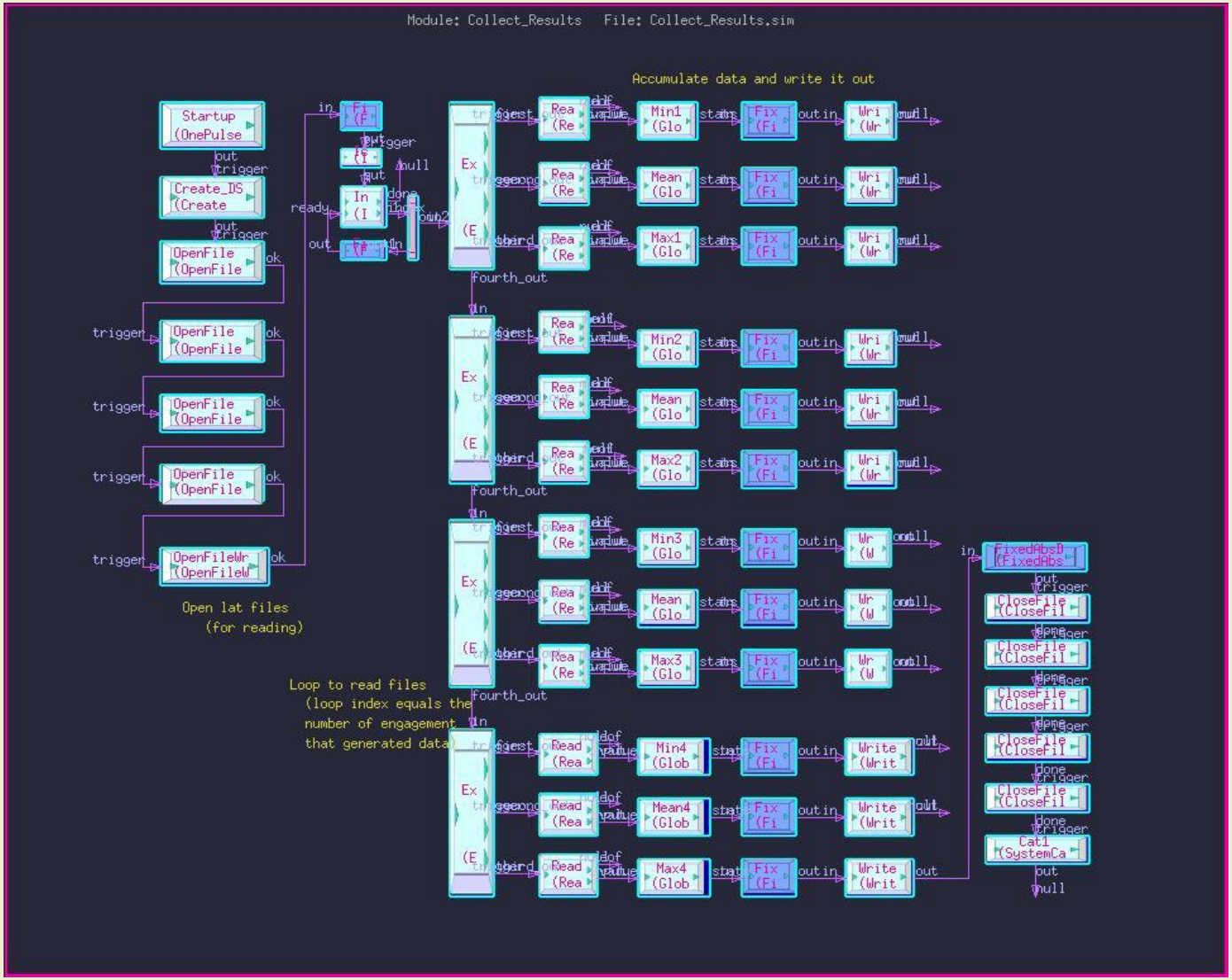
- “Histogram testcase”
  - Objective:
    - Need to run many Monte Carlo iterations of a simulation
    - Need to collect latency statistics (min, mean and max) for four point pairs (12 data points per iteration)
    - Need to identify the global min, mean and max for each
    - Need a histogram of the complete data set for one of the point pairs
    - Need to generate all required output fully automatically
  - Approach:
    - Use the Iterator to run iterations and collect min, mean and max
    - Use a separate “simulation” to (redundantly) collect min, mean and max
    - Use another separate “simulation” to assemble a global histogram
    - Tie together with several scripts
    - Demonstrate some “unusual” applications of a CSIM model

Block Diagram of “hist_test”



Using General Blocks as a Visual Programming Environment

- This CSIM "model" reads four files (scatter plot data), calculates the min, mean and max values for each, and appends the results onto other files.



The Phases of Development

- **Phase 1: Initial Model Development/Debug**
  - Graphical display can be extremely valuable in facilitating verification and debug
- **Phase 2: Data Generation and Analysis**
  - Usually, data generation (i.e. Monte Carlo) is most effectively completed using automated, non-graphical methods
  - Analysis of the collected data usually utilizes graphical methods (plotting, graphing, etc)
- **Phase 3: Results presentations/marketing**
  - Presentations to management/customers can benefit from attractive real-time graphical demos

Careful organization of the model in the beginning will greatly benefit the eventual real-time graphical demos

Starting A New Model

- To Start a new General-Blocks model:

Include reference to GenBlocks model library

  - *File / Import by Reference*
  - *$CSIM_MODEL_LIBS/general_blocks/Library.sim*

Begin drawing block diagrams

- The main file to include is *Library.sim*
  - Lists and categorizes all models
  - Includes *All.sim*

- The *All.sim* file contains all the block models

Tricks to Speed Development

- Stepwise, “build a little, test a little” process works best
- Recommended flow for a new model
  - Import the required libraries and define all known top level variables and macros
  - Identify a small, well understood chunk of functionality
    - Implement, simulate and debug, and verify that the simulation results are as expected
  - Add another chunk; repeat
- Always work with the smallest model possible; use stubs whenever appropriate
- Use pop ups, event probes, process timelines, etc. to help verify connectivity

Running Simulations Faster (Summary)

- For the fastest simulation turnaround:
  - Run nongraphically
  - Compile with optimization (O2)
  - Execute from the local /tmp directory
  - Direct stdout and stderr into a file
  - Run from the fastest machine available

Running Simulations Faster (Details)

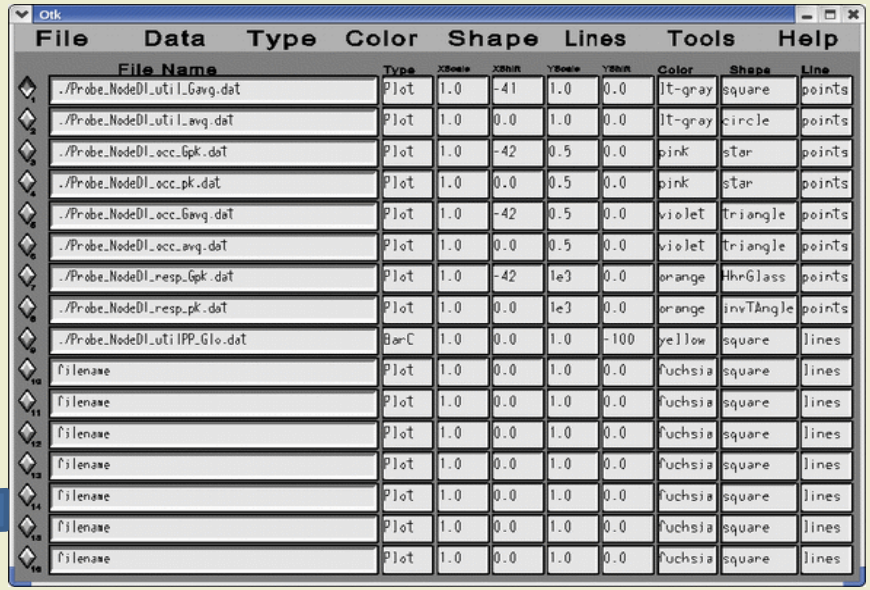
- Graphical simulations will run slower than nongraphical
- A running graphical simulation will run faster
  - while animation is turned off
  - By increasing the time display increment (slightly)
  - By directing terminal output to a file (stdout & stderr)
- You can build a faster graphical simulation
  - By turning off debugging (removing -g from gcc cmd)
  - By turning on optimization (adding -O2 to gcc cmd)
  - By copying all files to the local /tmp directory and executing there

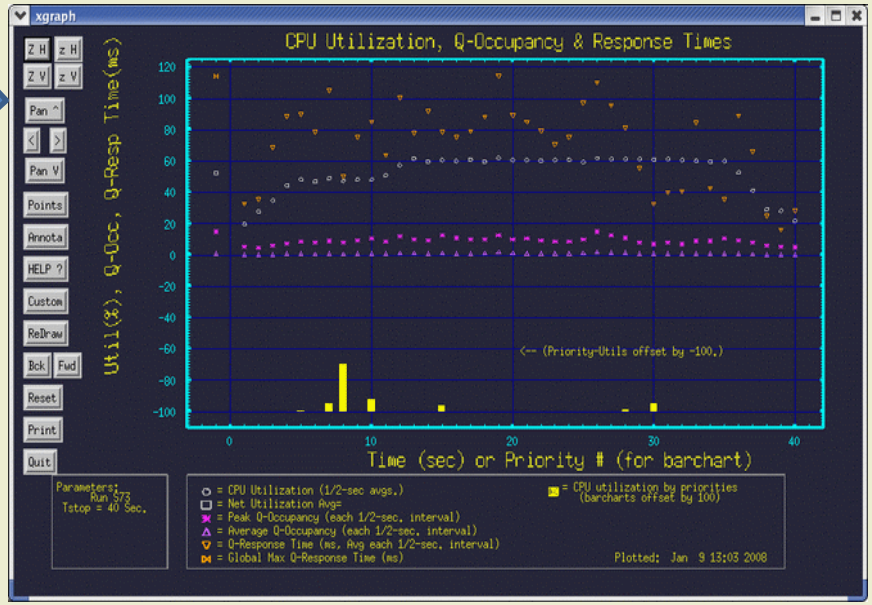
Running Simulations Faster (Details)

- Efficient simulations are *always* faster than inefficient simulations
  - Don't simulate anything that doesn't *need* to be simulated
    - Build times are proportional to the number of devices (boxes)
    - Simulation time is proportional to the number of device-events
  - Don't simulate a longer period than necessary
  - Don't simulate unnecessary details
  - Extraneous devices, inefficiently implemented models, etc. slow things down proportionately

ugui

- Ugui is a tool developed to simplify the display of multi-file xgraph plots (an xgraph front end)
- Up to 16 files (data sets) may be combined into a single plot
- Each data set may have individual:
  - Colors
  - Line types
  - Point shapes
  - X and/or y shifts
  - X and/or Y scale factors
- Text and/or legend files can be included





Required Files

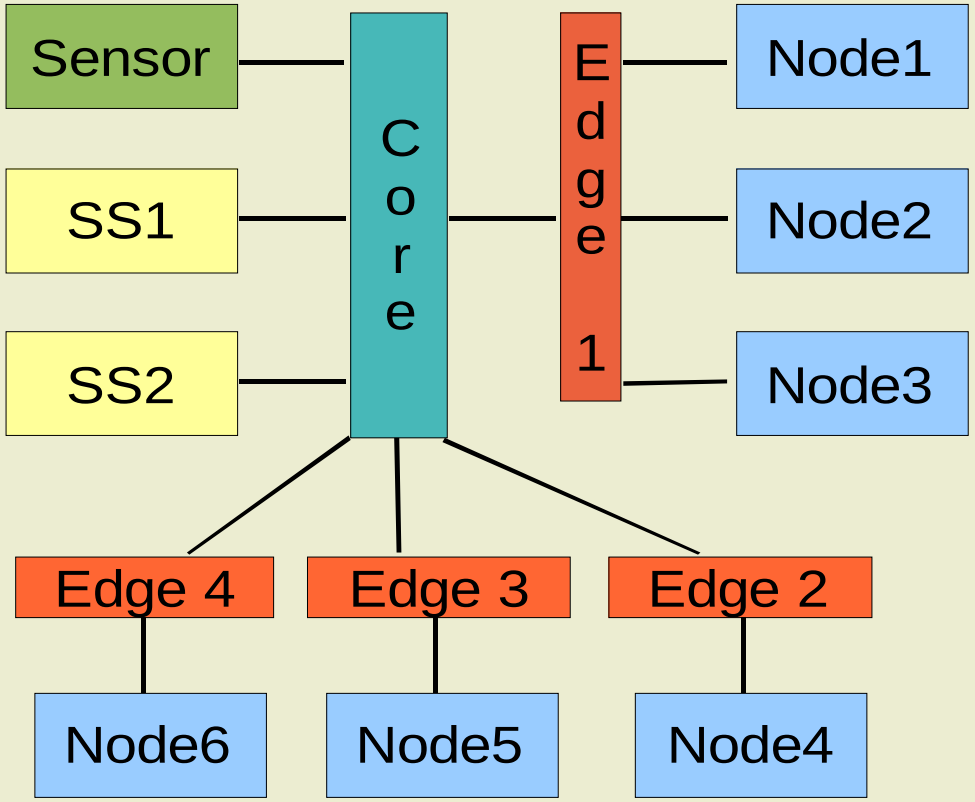
- Files required locally for initial build:
  - yourSimFile.sim, data_structs.txt, and (for now) soc_lib.c
    - The Library.sim and/or All.sim files are referenced from $CSIM_ROOT/model_libs/general_blocks
    - Controlled local library files (i.e. IMA_Lib.sim) are referenced from their repositories
- Build-created files
  - sim.exe and top_tab.dat are required
  - out.c and netinfo are not required
  - INTERMED*.csim files indicate a problem
- Some models require input data files
  - Control_Signal_Generator, Arbitrary_PulseTrain

Required Files (cont)

- Ancillary tool-related files
  - xgraph
    - Output data files from CSIM (*.dat)
    - Optional annotations/labeling data (title.doc)
  - tlpp (tlpp_gui)
    - EventHist.dat, tlpp.com
  - ugui
    - Setups saved in a *.raw file
    - Generates an xgraph_plot.com file

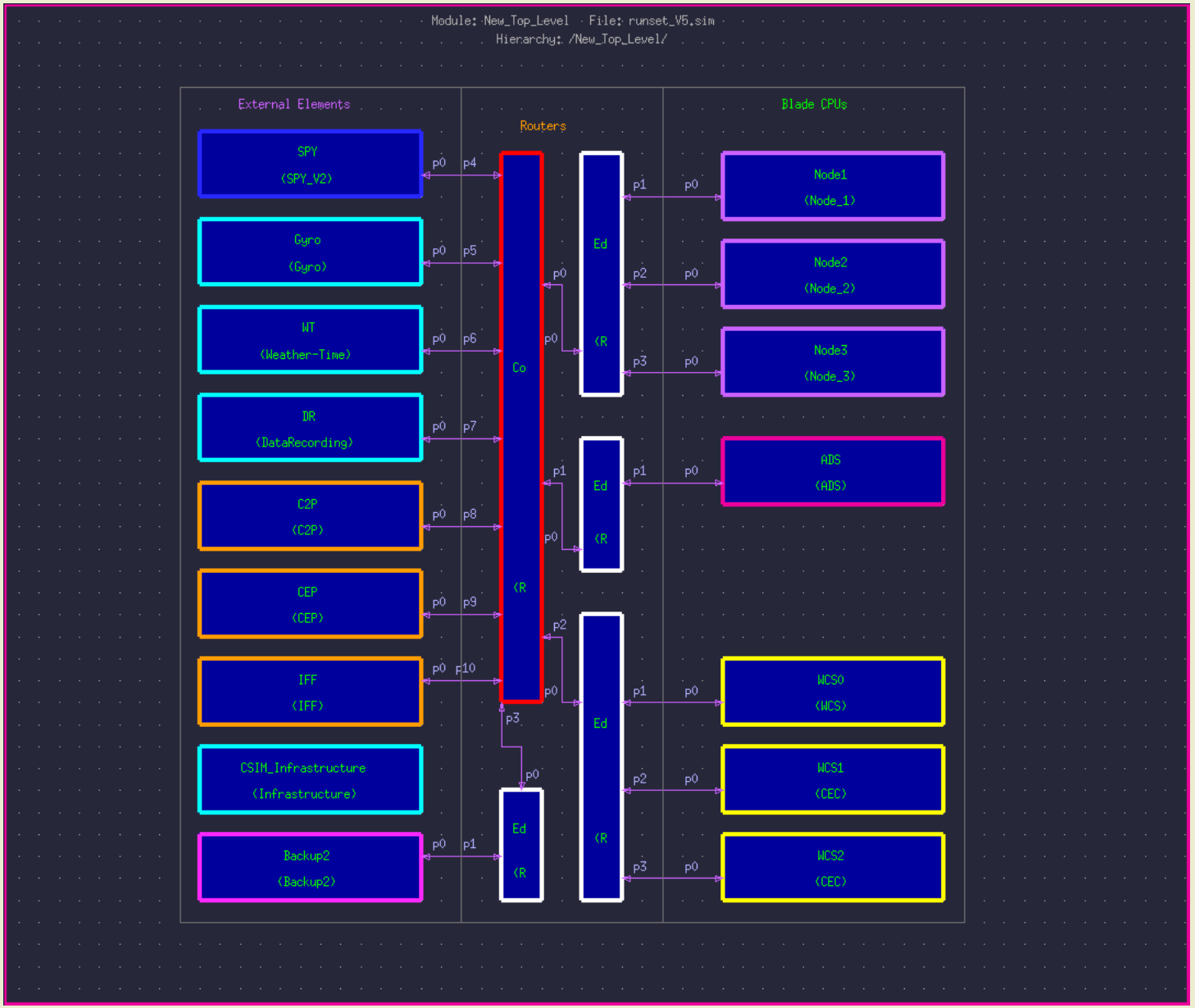
A More Complex Example

- Consider the drawn distributed system
- A Sensor and two other subsystems are attached to a Core Switch
- The Core Switch connects to four Edge Switches
- Each Edge Switch connects to a number of Nodes
- We are interested in the Processor Utilization, Latencies and other performance metrics
- What do the Multi Core and Limited Threads Models do for us in analyzing this system?

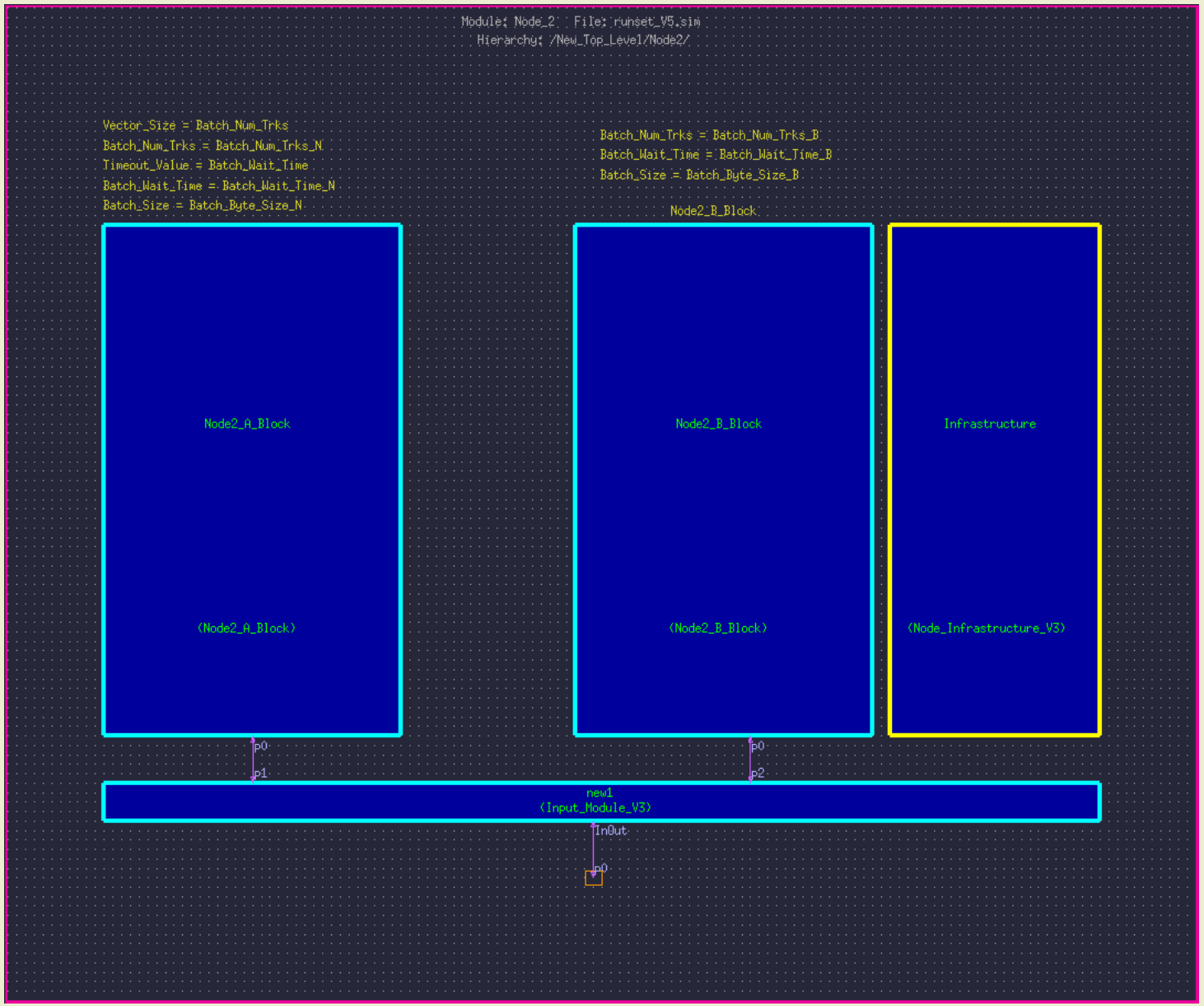


Example System

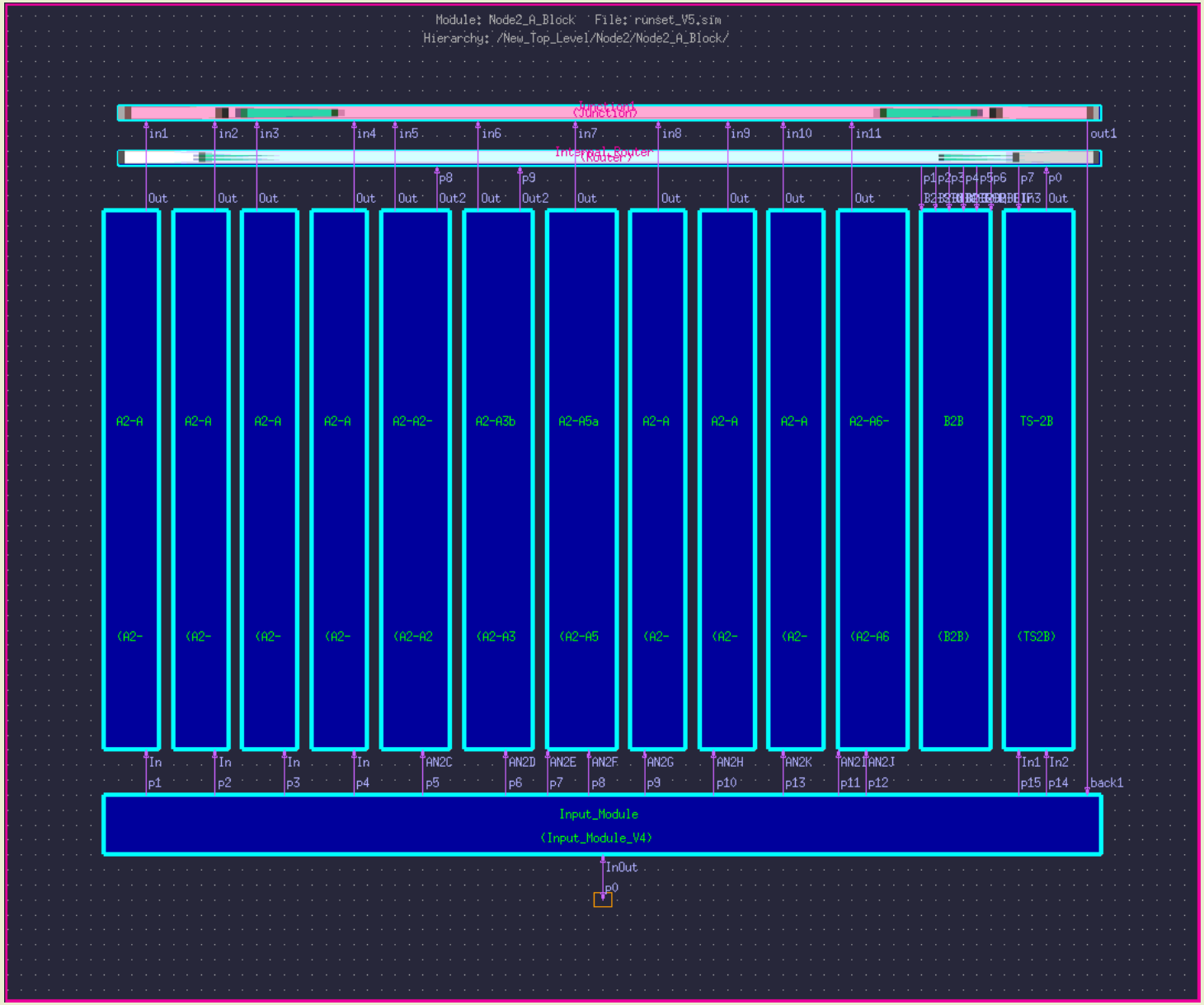
Top Level Block Diagram



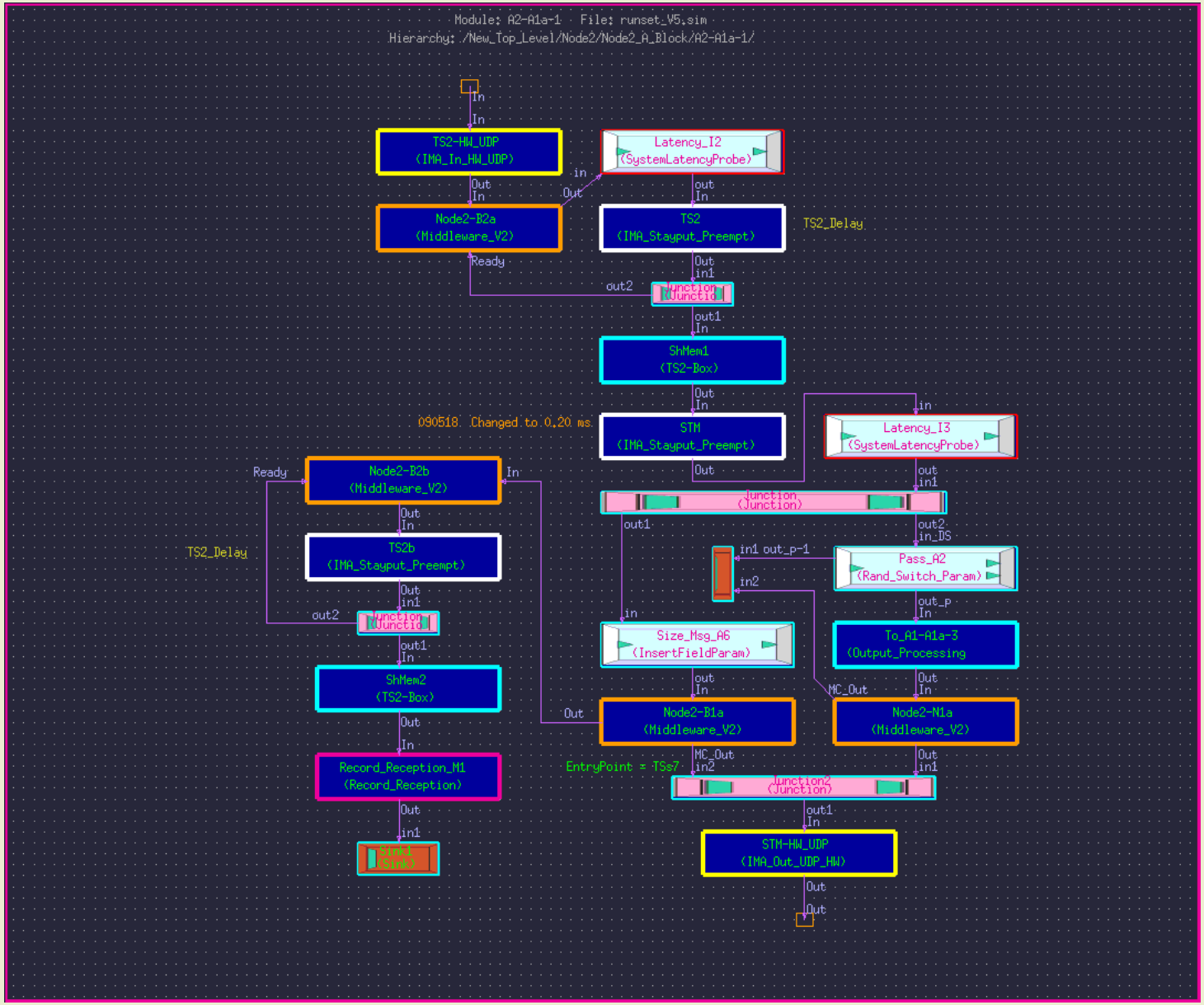
Details of Node_2



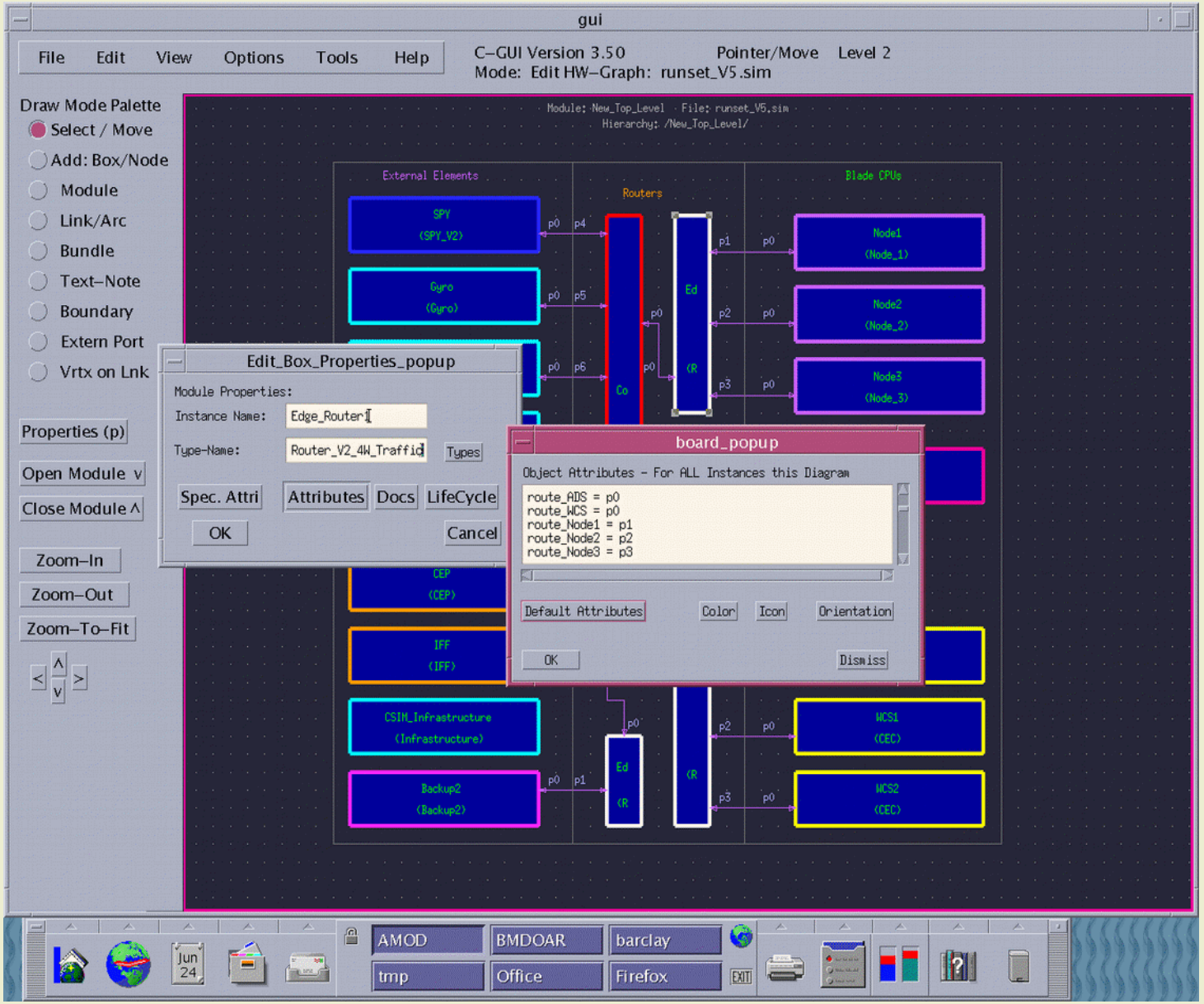
Details of Node_2_A_Block



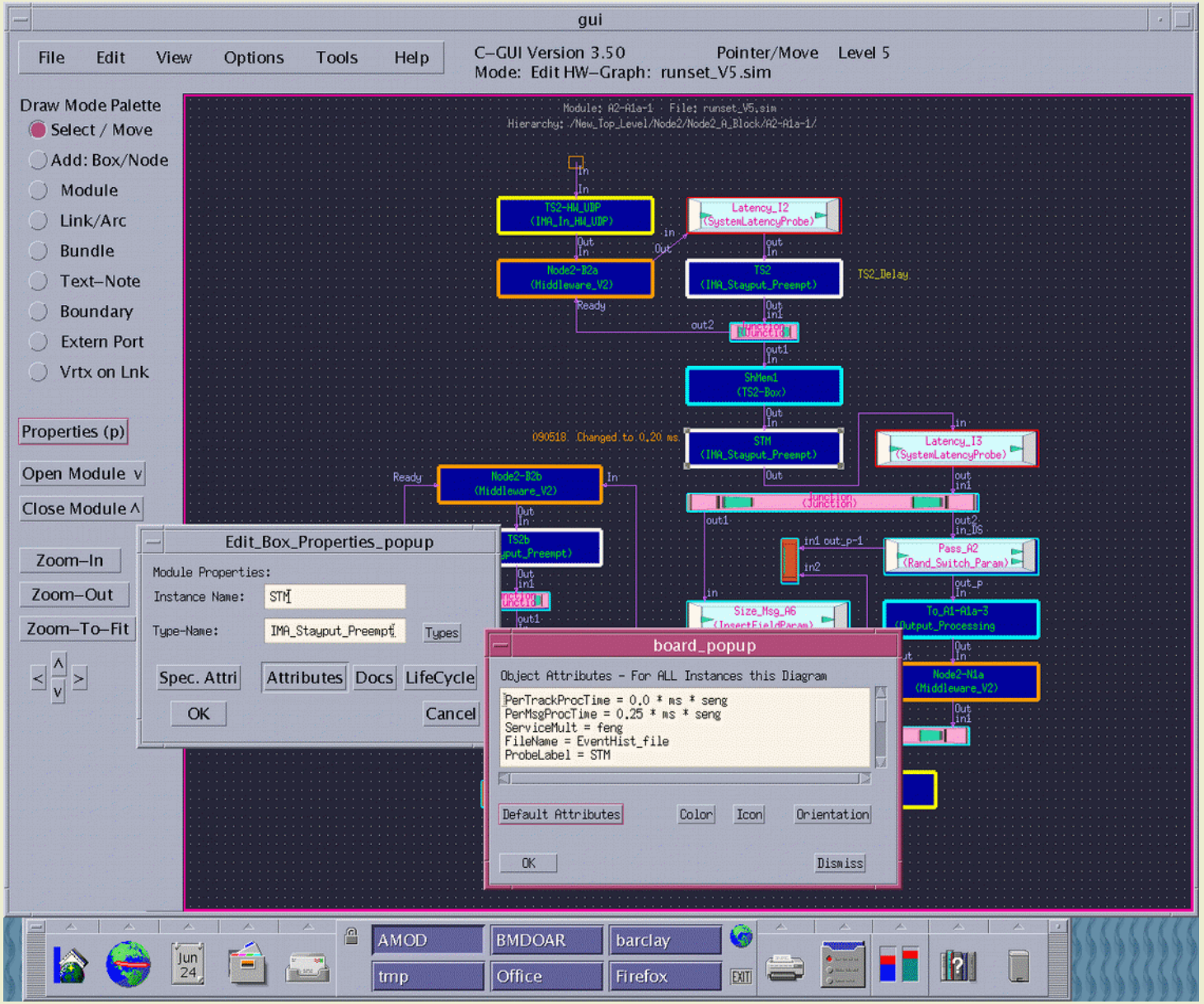
Detail of A2-A1a-1



Router Attribute Menus



Application Attribute Menus



New Models

A New Capability in General Blocks-based modeling

- Models can be Dynamic and Self Configuring

Models can more accurately represent the actual behavior of networks

Batcher/Unbatcher Models

- Generic_Batcher and Generic_UnBatcher
  - Developed to extend and simplify the GenericVector set of models (GVCreate, GVLen, Access_GVector, Setup_GVElem)
- Typically used to represent a set of small messages combined into a single large message
- Generic_Batcher combines messages
- Generic_UnBatcher separates messages
- Enables accurate measurement of end to end latencies

Admin Model

- The Admin is a *scheduler*, oriented to distributing periodic tasks among a group of processors in a networked environment:
- Operation:
  - A message, containing a task name, is sent to the Admin to request initiation of the task. The “tasks” are typically comparable to a sequence diagram.
  - The admin uses the specified algorithm (four are currently supported) to assign the task to a processor. It updates its status table.
  - The admin sends a message to the assigned processor to notify it to accept a task of the specified type and ID
  - The processor interprets the message and starts the task.
  - At the completion of a task, the processor sends a message to the Admin to report the task completion.
  - The Admin updates its status table.

Admin Task Assignment

- A file (task_table.dat) defines:
  - Task names, processor names, scheduling algorithms and maximum task loading for each processor
- An example task table:

		CPU1	CPU2	CPU3	CPU4	CPU5	CPU6
tsk1	fill_u	8	7	0	0	7	8
tsk2	fill_d	5	0	0	0	0	5
tsk3	u_task	3	0	4	4	0	3
tsk4	u_all	5	2	6	6	2	5

Sender/Receiver

- The Sender and Receiver models use named synchrons to “wirelessly” send data structures between points
- One to one, one to many, many to one and many to many configurations can be supported
- Typically used to distribute control signals, alarms and triggers

Router Model

- Router has 16 bidirectional ports
  - Flexible specification of routing rules, i.e.
    - route_24_50_20 = p5
    - route_24_50_20_7 = p1
    - route_DEFAULT = p2
    - route_3_2_1_1_0_3_7up = p1
    - route_cabinet2_card3_cpu4 = p4
    - route_24_50_F = p3
- Supports multicast publish, subscribe
  - multi_w_x_y_z = p1_p2_p3_p4
- Supports dynamic subscribe/unsubscribe
  - subscribe_24_50_20_6 = p6_p8
  - unsubscribe_24_50_20_1 = p13_p14
- Can be used as data sorter/selector

Router Enablers

- Used to create routing info:
  - Append_Route_List
  - Append_String
  - Num_to_String
  - String_to_Num
- Used to split a full duplex link:
  - PortConvert

Multi-Core Processor Model

- Developed to more accurately represent the execution of modern multi-core processors
  - Previous approaches (scaling application execution times) are inadequate
- Initially represented as ideal
- Currently, 1-16 cores can be represented
- Recently upgraded to include load-dependent performance degradation
- Speedup, as a function of active cores, specified as a table, i.e., Speedup_2 = 1.5

Multi Core Model

- There are several (relatively) independent capabilities lumped under the heading Multi Core Model
  - Multi Core Model (basic)
  - “Amdahl” Performance Degradation Capability
  - Limited Thread Capability
  - Multi-part Thread Capability
  - “CoreLocked” Capability

Multi Core Model (basic)

- The basic Multi Core Model enables the representation of multiple tasks executing simultaneously on a processor
- Model behavior:
  - Number of cores can be specified separately for all processors (i.e. MaxNumTasks)
  - Any task can execute on any core
  - No performance degredation modeled

Multi Core Model (Amdahl)

The Amdahl model extends the basic model with generic performance degredation
  - Not actually hard-wired to Amdahl's model
- The speedup behavior is specified in a table-like set of attributes, i.e., A80 is:
  - Speedup_2 = 1.67
  - Speedup_3 = 2.14
  - Speedup_4 = 2.5
- Model behavior:

Limited Thread Model

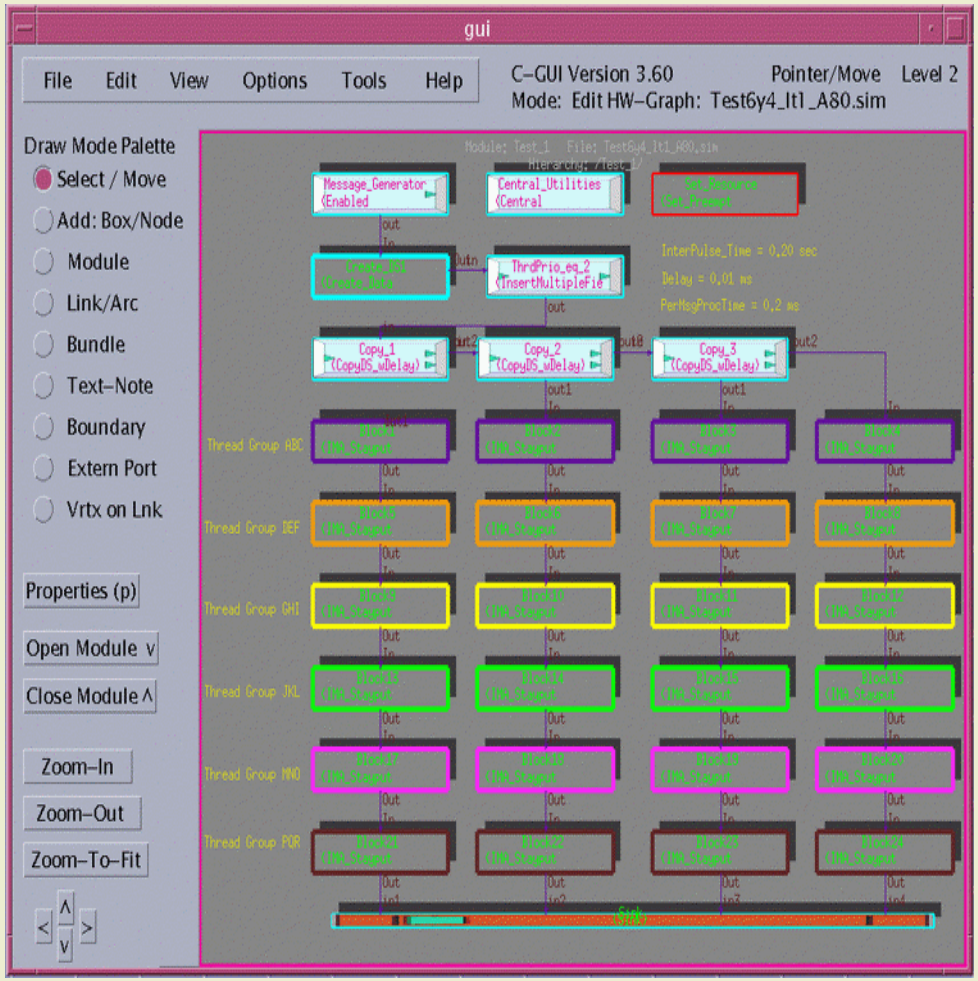
- The Limited Thread is an extension to the basic model
- The Multi Core model by itself does not specifically model software which is single threaded or otherwise limited
  - Handled indirectly through the specification of Speedup values
- If information regarding software limitations or system configuration are available, the Limited Thread Model can be used to specifically represent single threaded or otherwise limited threaded software
- The Multi Core Model and Limited Thread model are independent
- Operation of Limited Threads (LT)
  - Software which is limited is tagged with a set of attributes
  - When a message is received to initiate an execution, tags and status are checked; LTs may span multiple blocks
    - If this LT is not active, execution is started and the Active tag is set
    - If this LT is active, and this message does not represent the Active thread, the incoming message is queued; each identified LT has its own queue independent of all other queues
    - If this message represents the Active thread, it is started
    - When a thread exits, the Active tag is reset and the next thread is released

Core Locked Model

- The Core Locked model enables the representation of threads (tasks, applications) that execute only on specific core(s)
- Any thread can be specified to execute on any or all of the available cores
- Specification of a thread (example):
  - NumLimitThrds = 7
  - Thread_Name_1 = DEF
  - Max_Threads_1 = 1
  - Max_Thrd_Prio_1 = 3
  - Thrd_Map_1 = "1 3 4"

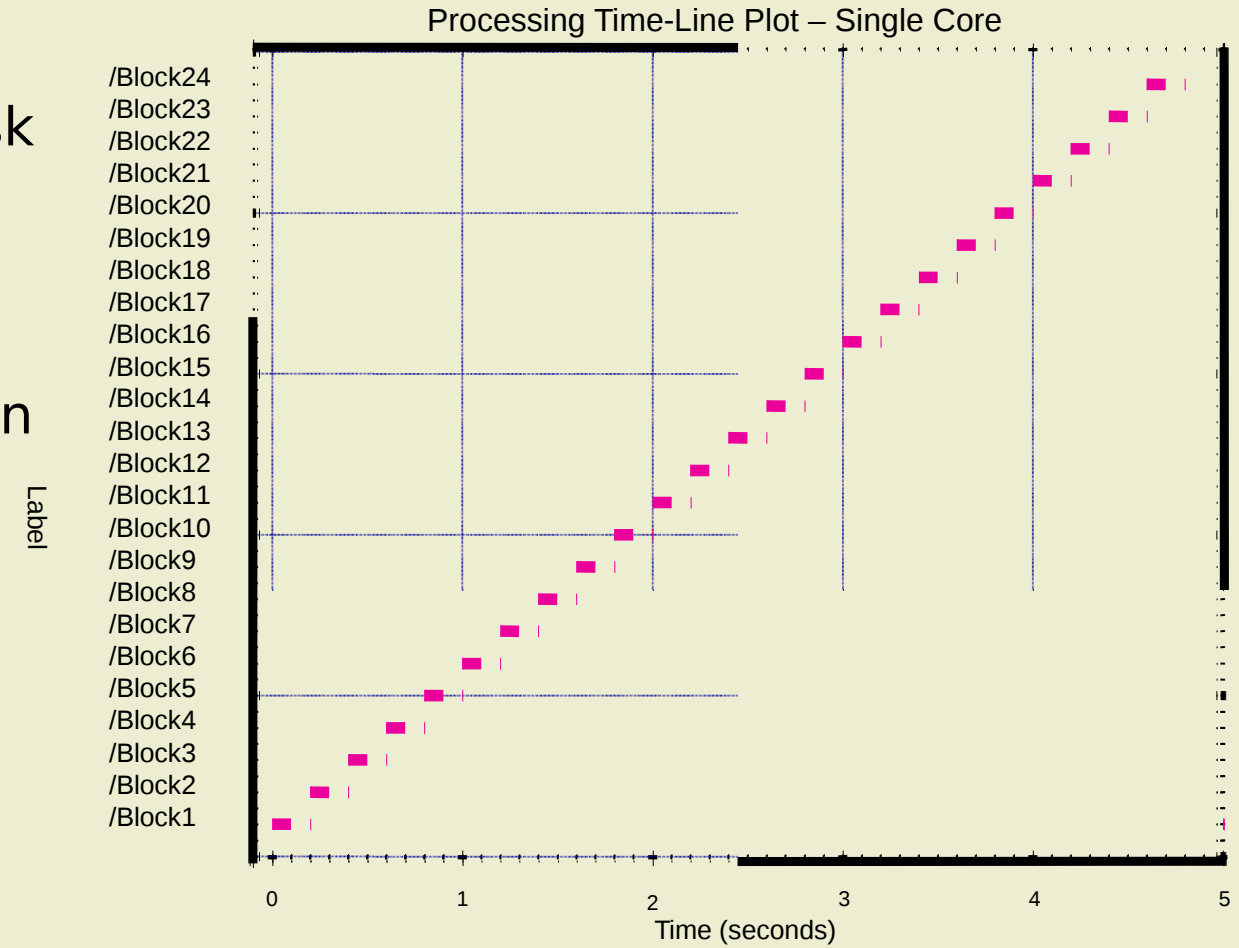
Simple Example of Multi Core and Limited Thread Operation

- This simple model will illustrate the behavior.
- Four identical sequences of tasks are indicated by the columns of six colored boxes.
- Boxes of the same color (row) are in the same LT group
- Each column begins execution quickly (0.01 milliseconds) after the column to its left.



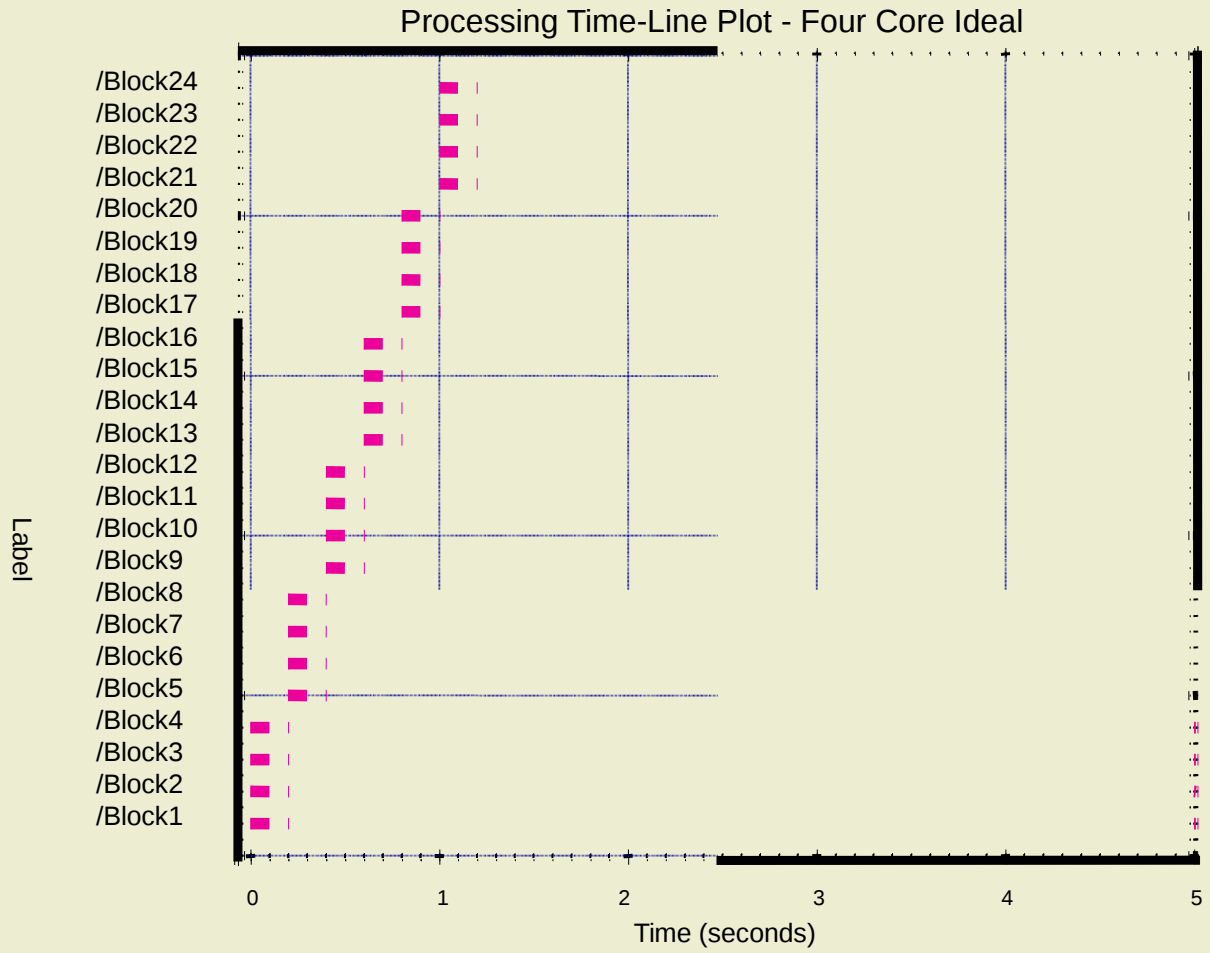
Single Core Execution

- In the case of a Single Core, the task execution is fully sequential, from Block1 through Block24.
- The last block completes execution at T=4.8 seconds



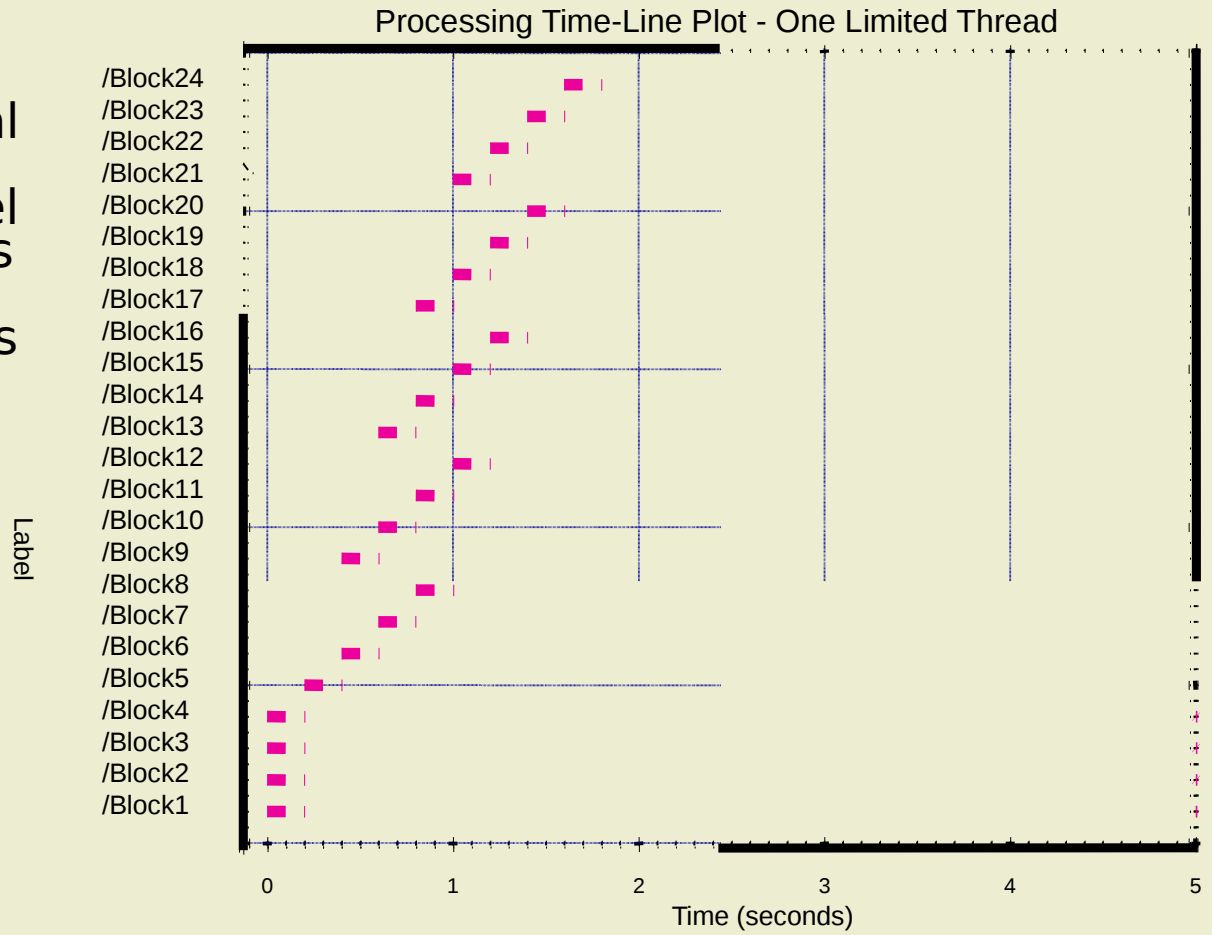
Ideal Four Core Execution

- In the case of the Ideal Four Core, the task execution is fully parallel
- The last block completes execution at T=1.2 seconds



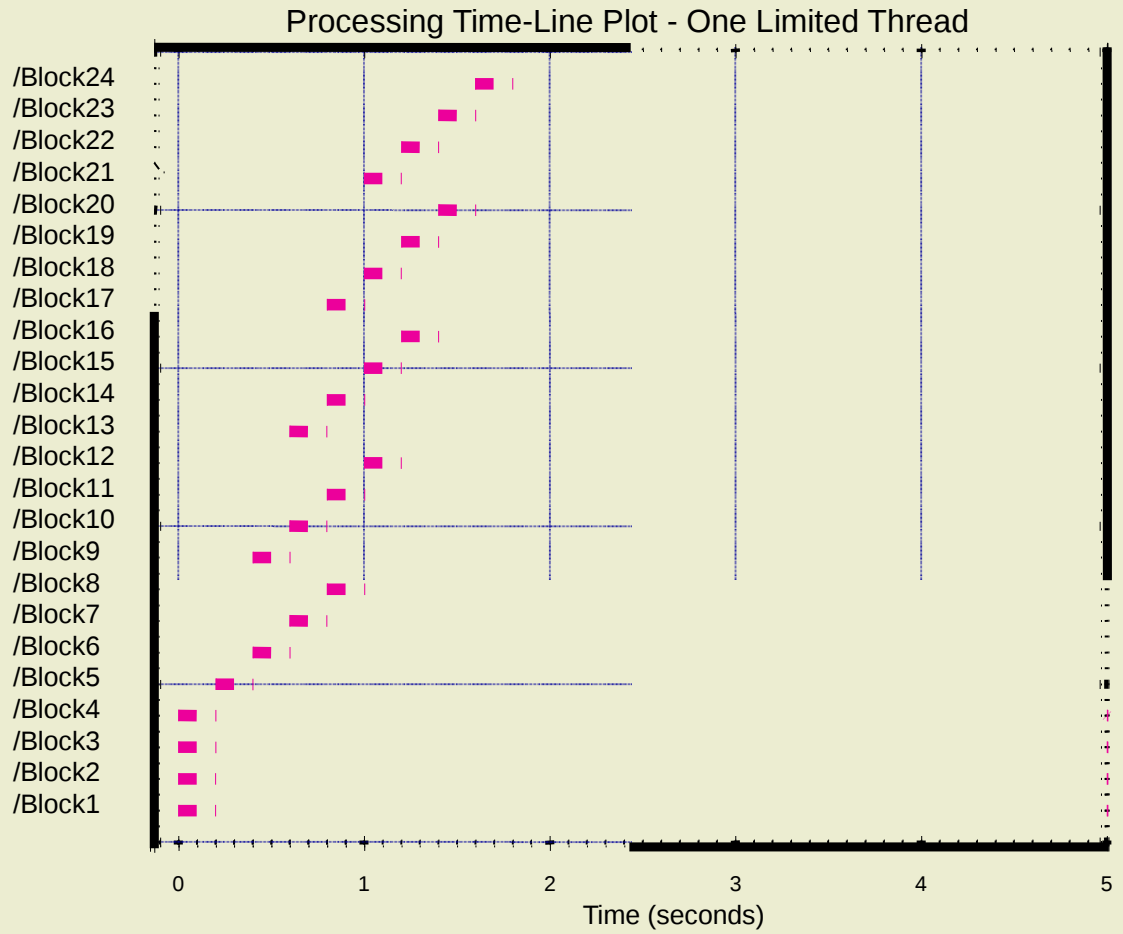
Four Core, Amdahl = 0.8 Execution

- In the Four Core, Amdahl 0.8 case, the task execution is fully parallel
- The Amdahl slowdown is apparent
- The last block completes execution at
  - Speedup_4 = 2.5 for Amdahl 0.8, 4 core
  - T=4.8/2.5 seconds
  - T=1.92 seconds



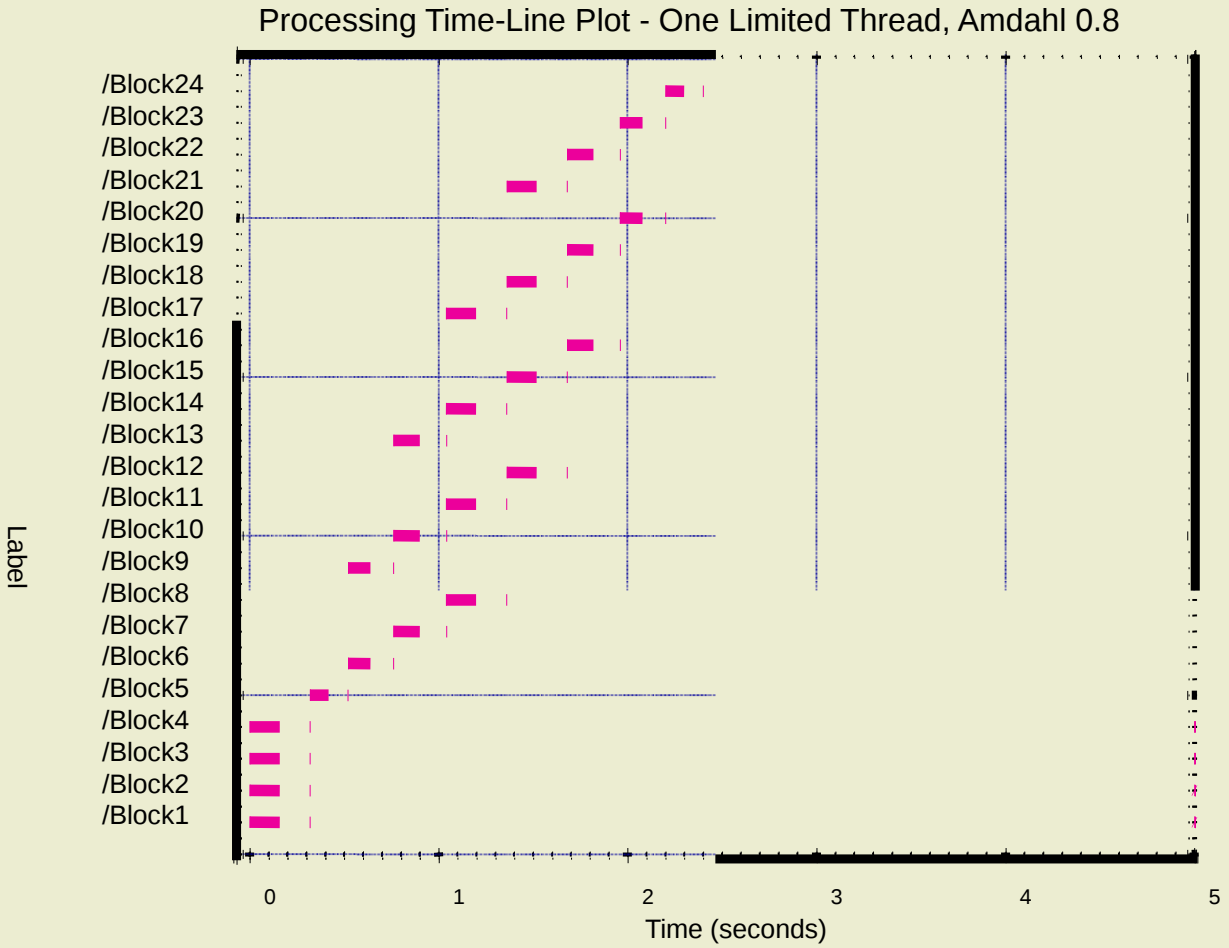
One Limited Thread Execution

- In the One Limited Thread case, the second row of tasks, Block5 through Block8 (orange boxes) represent single threaded software
- Amdahl slowdown is turned off
- The last block completes execution at 1.8 seconds`

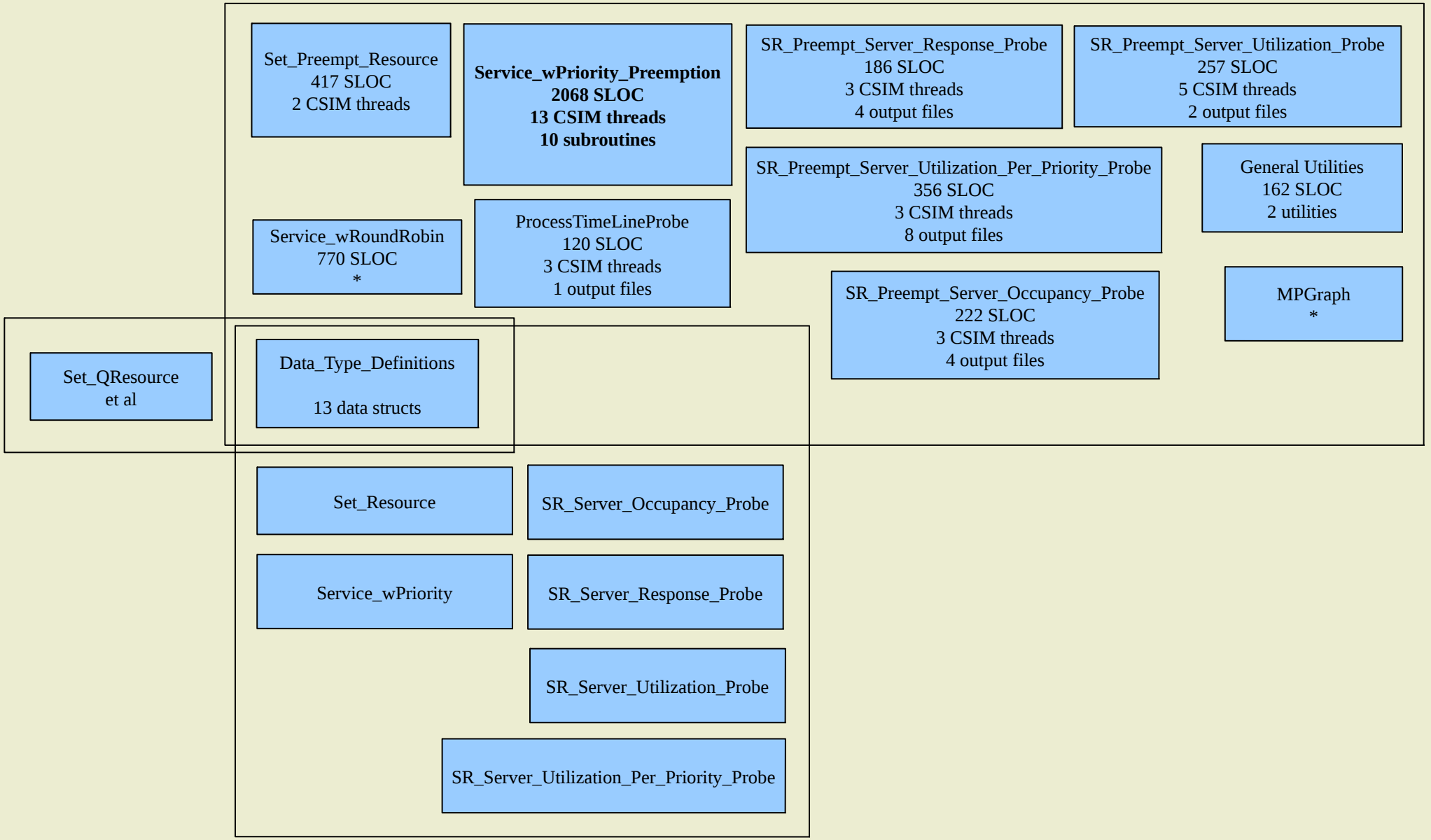


One Limited Thread, Amdahl 0.8 Execution

- In the One Limited Thread, Amdahl 0.8 case, the second row of tasks, Block5 through Block8 (orange boxes) represent single threaded software
- Amdahl slowdown is set for Amdahl 0.8
- Note the load dependent dilation
- The last block completes execution at 2.4 seconds



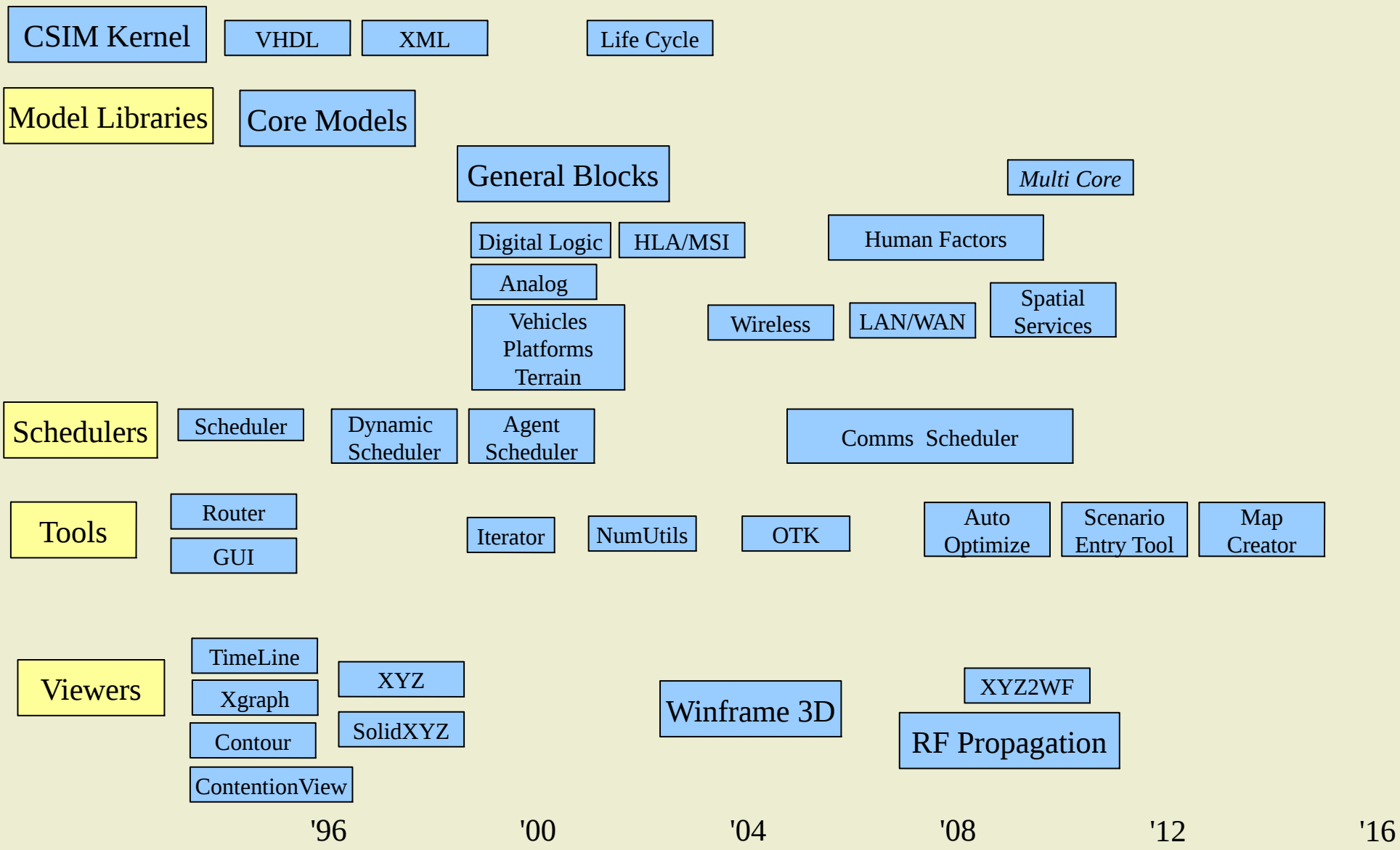
Service Models Dependencies



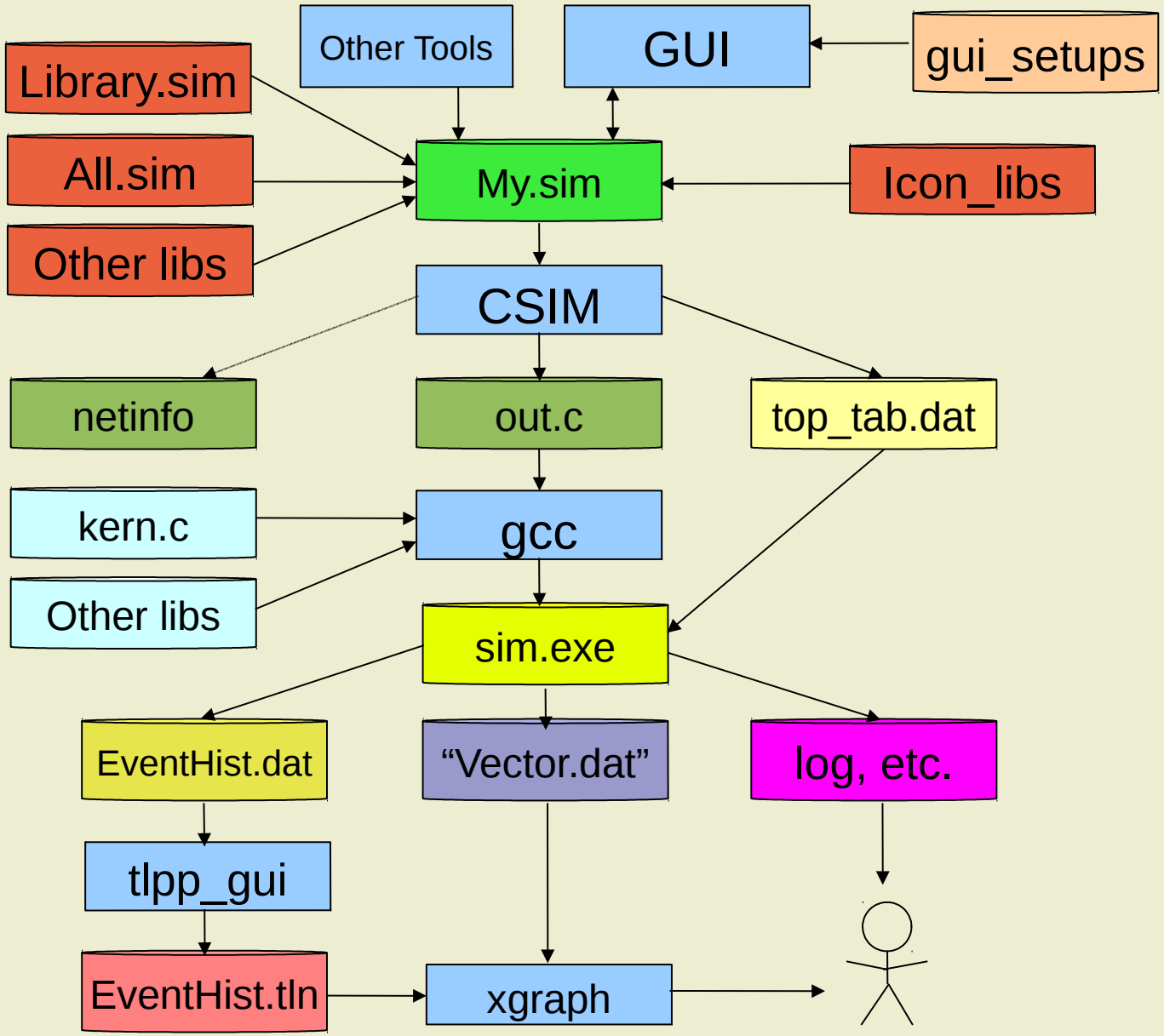
SPP Model Capabilities

- Original
  - Priority
  - Preemption
  - Utilization
  - Utilization Per Priority
  - Occupancy
  - Response
- Added
  - Process Timeline
  - Self Test
  - Multi Core (ideal)
  - “Amdahl” Degradation
  - Limited Thread
  - Limited Thread Continuation
  - Core Locking
  - Utilization Per Core
  - Utilization By Thread
  - Utilization By Core
  - Thread Queue Occupancy

Recent CSIM Evolution



General Blocks Data Flow



Generic Development Flow

- Construct model
- Build simulation
- Execute simulation
- Review output data