Trace-driven co-simulation of high-performance computing systems using OMNeT++

Cyriel Minkenberg, Germán Rodríguez Herrera
IBM Research GmbH, Zurich, Switzerland

Overview

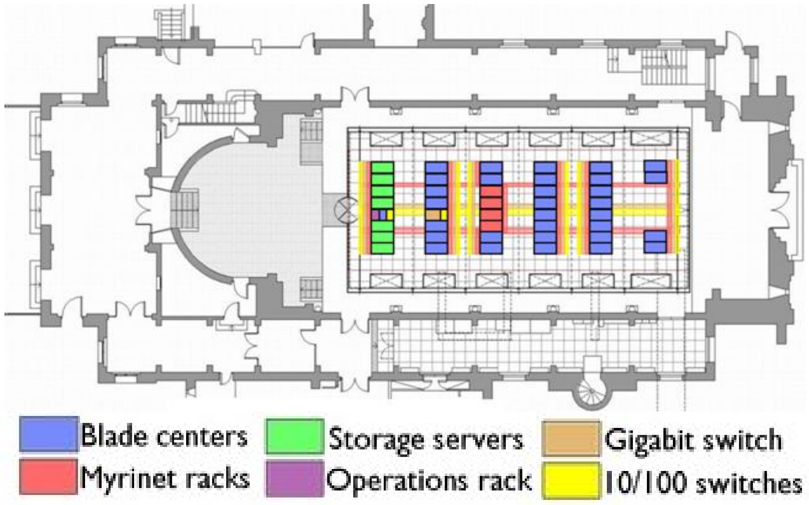
- Context
  - MareNostrum ➔ MareIncognito
  - Design of interconnection networks for massively parallel computers
- Models & tools

Computation	*Dimemas*
Communication	*Venus*
Visualization	*Paraver*

- Integrated tool chain
  - Venus architecture
  - Co-simulation with Dimemas
  - Paraver tracing
  - Accuracy: topologies, routing, mapping, Myrinet/IBA/Ethernet models
- Sample results
- Conclusions & future work

Background: MareNostrum

- Blade-based parallel computer at Barcelona Supercomputing Center (BSC)
- 2,560 nodes
- 10,240 IBM PowerPC 970MP processors at 2.3 GHz (2,560 JS21 blades)
- Peak performance of 94.21 Teraflops
- 20 TB of main memory
- 280 + 90 TB of disk storage
- Interconnection networks
  - Myrinet and Gigabit Ethernet
- Linux: SuSe Distribution
- 44 racks, 120 m^2 floor space



MareIncognito

- Joint project between BSC and IBM to developed a follow-on for MareNostrum, codenamed *MareIncognito*
- 10+ PFLOP/s machine for 2011 timeframe comprising in the order of 10 - 20K 1+ TFLOP blades
- Our focus: **Design a performance- and cost-optimized interconnection network for such a system**
  - Gain deeper understanding of HPC traffic patterns and the impact of the interconnect on overall system performance
  - Use this understanding to optimize the design of the MareIncognito interconnect
    - Reduce cost & power without sacrificing much performance
    - Topology, bisectional bandwidth, routing, contention (internal & external), task placement, collectives, adapter & switch implementation

Interconnect design

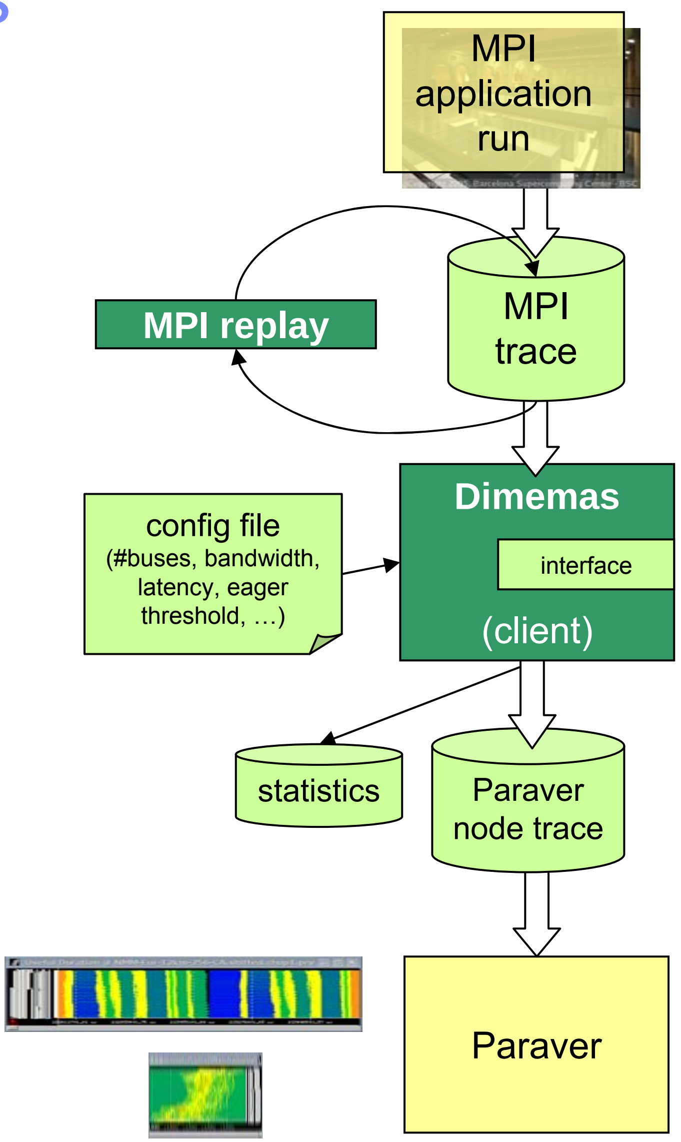
- Ever-increasing levels of parallelism and distribution are causing shift from processor-centric to interconnect-centric computer architecture
- Interconnect represents a significant fraction of overall system cost
  - Switches, cables
  - Maintenance
- We need tools to predict system performance with reasonable accuracy
  - Absolute performance
  - Parameter sensitivity; trend prediction ("what if?")
  - Accurate model of application behavior (compute nodes)
  - Accurate model of communication behavior (interconnect)
- Many tools do either of these things well; very few manage both simultaneously with sufficient accuracy

Compute node model

- From the network perspective, compute node acts as traffic source and sink
- In communication network design (telco, internet) typical approaches are
  - "Synthetic" models based on some kind of stochastic processes
    - may (or may not) be reasonable if a sufficient level of statistical multiplexing is present
    - relation to reality unclear at best
  - Replaying traces recorded on, e.g., some provider's backbone
  - In either case, semantics of traffic content are rarely considered: no causal dependencies between communications
- Traffic in HPC systems has strong causal dependencies
  - These stem from control and data dependencies inherent in a given parallel program
  - These dependencies can be captured by running a program on a given machine and recording them in a trace file
  - If the program has been written using the Message Passing Interface (MPI) library, this basically amounts to a per-process list of send/recv/wait calls
  - Such a trace can be replayed observing the MPI call semantics to correctly reproduce communication dependencies

Compute node model: *Dimemas*

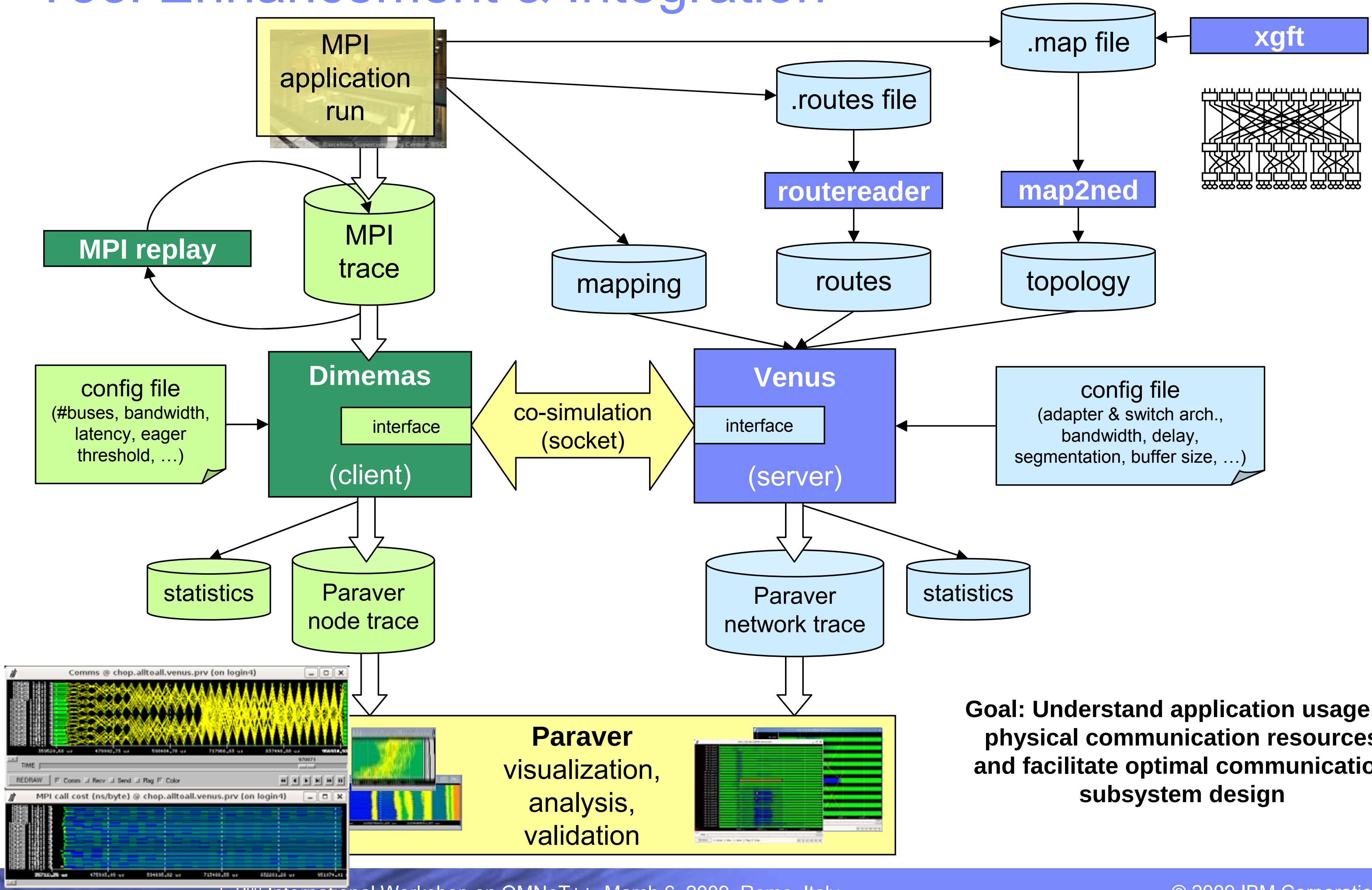
- Dimemas (BSC simulator)
  - Rapidly simulates MPI traces collected from real machines
  - Faithfully models MPI semantics & node architecture
  - Sensitivity: helps identify coarse-grain factors and relevant communication phases
  - Leverages CEPBA Tools trace generation, handling and visualization
  - Models interconnection network at a very high abstraction level
- Paraver (BSC visualization)
  - Visualizes MPI communication patterns at the node level
  - Allows "debugging" of inter-process communication, e.g., load imbalance, contention



Interconnect model: *Venus*

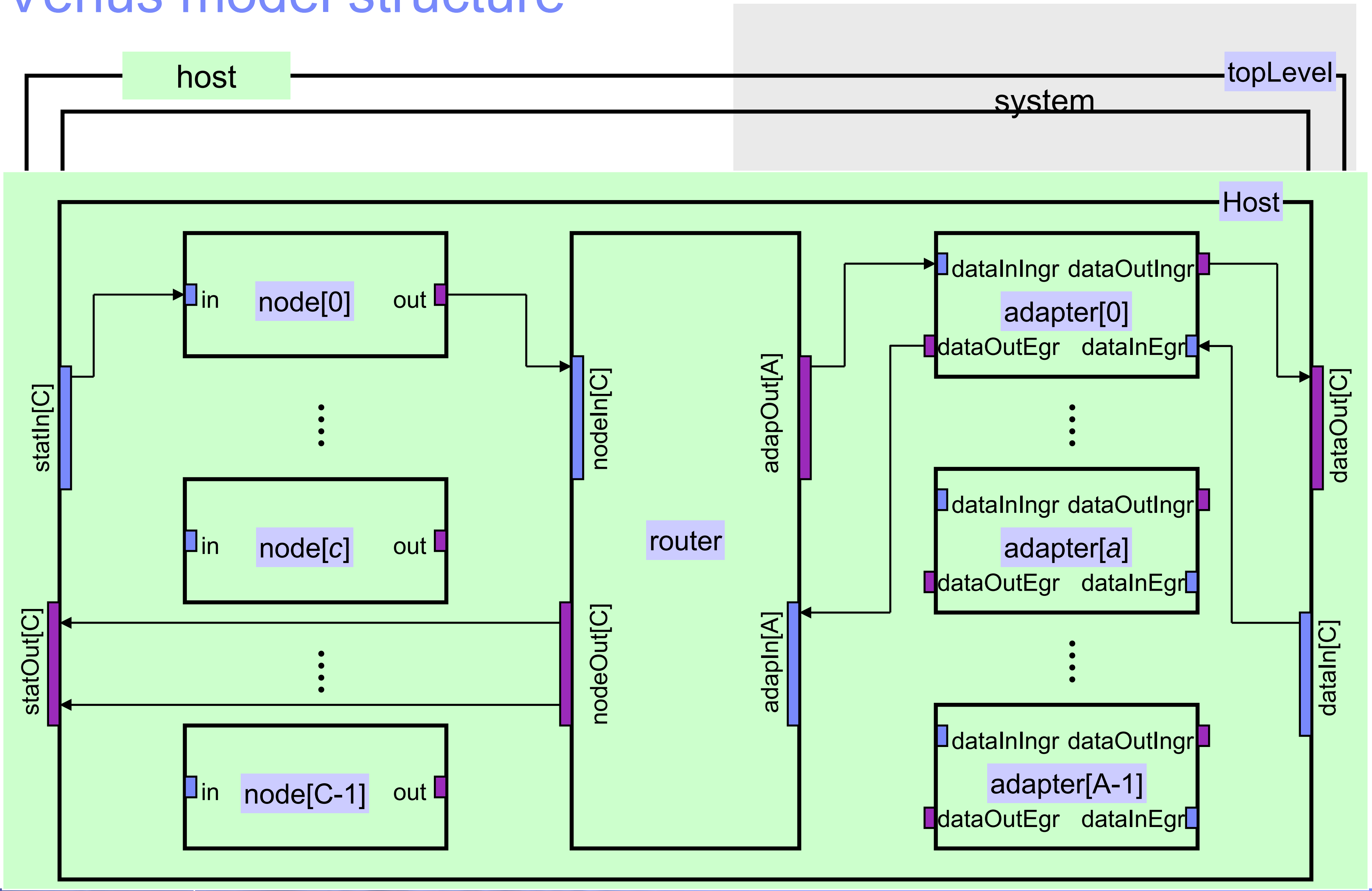
- **Simulates interconnect at flit level**
  - Event-driven simulator using OMNeT++ (originally based on MARS simulator)
  - Supports wormhole routing and segmentation for long messages
- **Provides various detailed switch and adapter implementations**
  - Myrinet, 10G Ethernet, InfiniBand
  - Generic input-, output-, and combined input-output-queued switches
- **Provides various topologies and routing methods**
  - Extended Generalized Fat Tree (XGFT), Mesh, 2D/3D Torus, Hypercube, arbitrary Myrinet topologies
- **Supports various routing methods**
  - Source routing, table lookup
  - Algorithmic (online), Myrinet routes files (offline)
  - Static, dynamic
- **Highly configurable**
  - Topology, switch/adapter models, buffer sizes, link speed, flit size, segmentation, latencies, etc.
- **Supports MareNostrum/MareIncognito**
  - Server mode to co-simulate with Dimemas via socket interface
  - Outputs paraver-compatible trace files enabling detailed observation of network behavior
  - Detailed models of Myrinet switch and adapter hardware
  - Translation tool to convert Myrinet map file to OMNeT++ ned topology description
  - Import facility to load Myrinet route files at simulation runtime; supports multiple routes per flow (adaptivity)
  - Flexible task to node mapping mechanism
  - Tool to generate Myrinet map and routes files for arbitrary XGFTs

Tool Enhancement & Integration

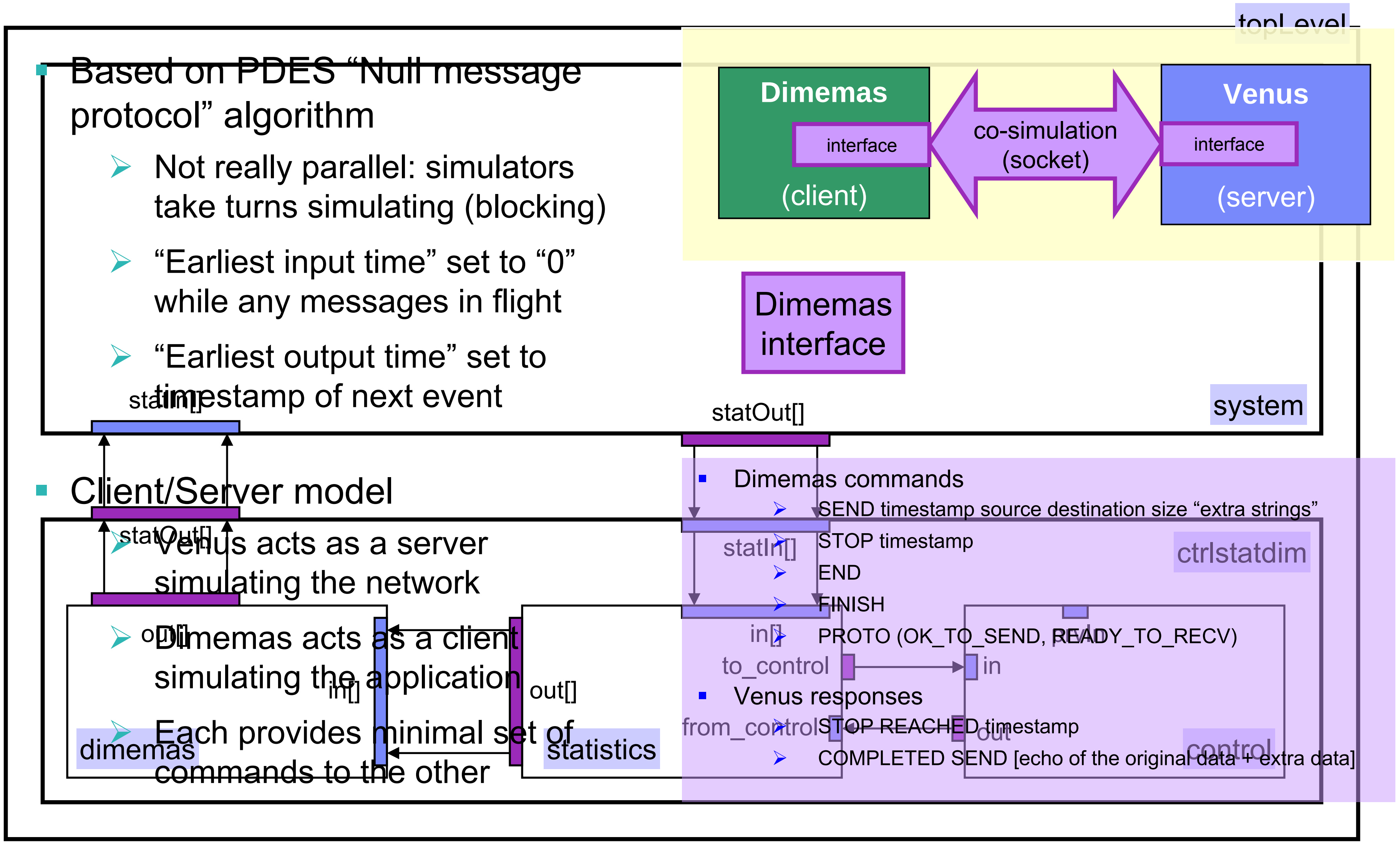

MPI application run
.map file
xgft
.routes file
routereader
map2ned
MPI replay
MPI trace
mapping
routes
topology
config file (#buses, bandwidth, latency, eager threshold, …)
Dimemas
interface
(client)
co-simulation (socket)
Venus
interface
(server)
config file (adapter & switch arch., bandwidth, delay, segmentation, buffer size, …)
statistics
Paraver node trace
Paraver network trace
statistics
Paraver visualization, analysis, validation
Comms @ chop.alltoall.venus.prv (on login4)
MPI call cost (ns/byte) @ chop.alltoall.venus.prv (on login4)

Goal: Understand application usage of physical communication resources and facilitate optimal communication subsystem design

Venus model structure



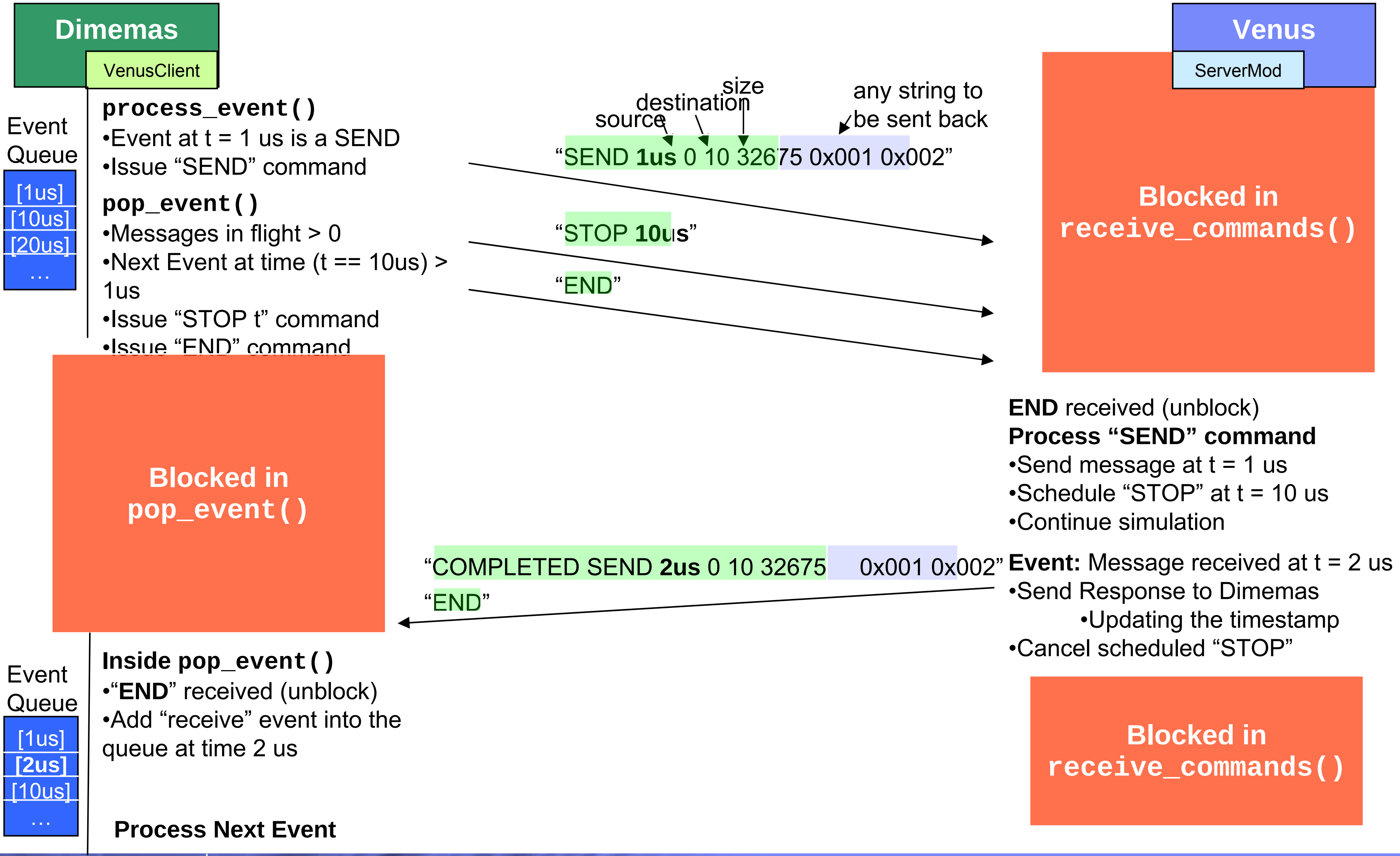
Hybrid Dimemas & Venus simulation



- Based on PDES "Null message protocol" algorithm
  - Not really parallel: simulators take turns simulating (blocking)
  - "Earliest input time" set to "0" while any messages in flight
  - "Earliest output time" set to timestamp of next event
- Client/Server model
  - Venus acts as a server simulating the network
  - Dimemas acts as a client simulating the application
  - Each provides minimal set of commands to the other

- Dimemas commands
  - SEND timestamp source destination size "extra strings"
  - STOP timestamp
  - END
  - FINISH
  - PROTO (OK_TO_SEND, READY_TO_RECV)
- Venus responses
  - STOP REACHED timestamp
  - COMPLETED SEND [echo of the original data + extra data]

D&V Co-simulation: SEND example



process_event()
- Event at t = 1 us is a SEND
- Issue "SEND" command

pop_event()
- Messages in flight > 0
- Next Event at time (t == 10us) > 1us
- Issue "STOP t" command
- Issue "END" command

"SEND **1us** 0 10 32675 0x001 0x002"

"STOP **10us**"

"END"

END received (unblock)

Process "SEND" command
- Send message at t = 1 us
- Schedule "STOP" at t = 10 us
- Continue simulation

Event: Message received at t = 2 us
- Send Response to Dimemas
    - Updating the timestamp
- Cancel scheduled "STOP"

"COMPLETED SEND **2us** 0 10 32675 0x001 0x002"

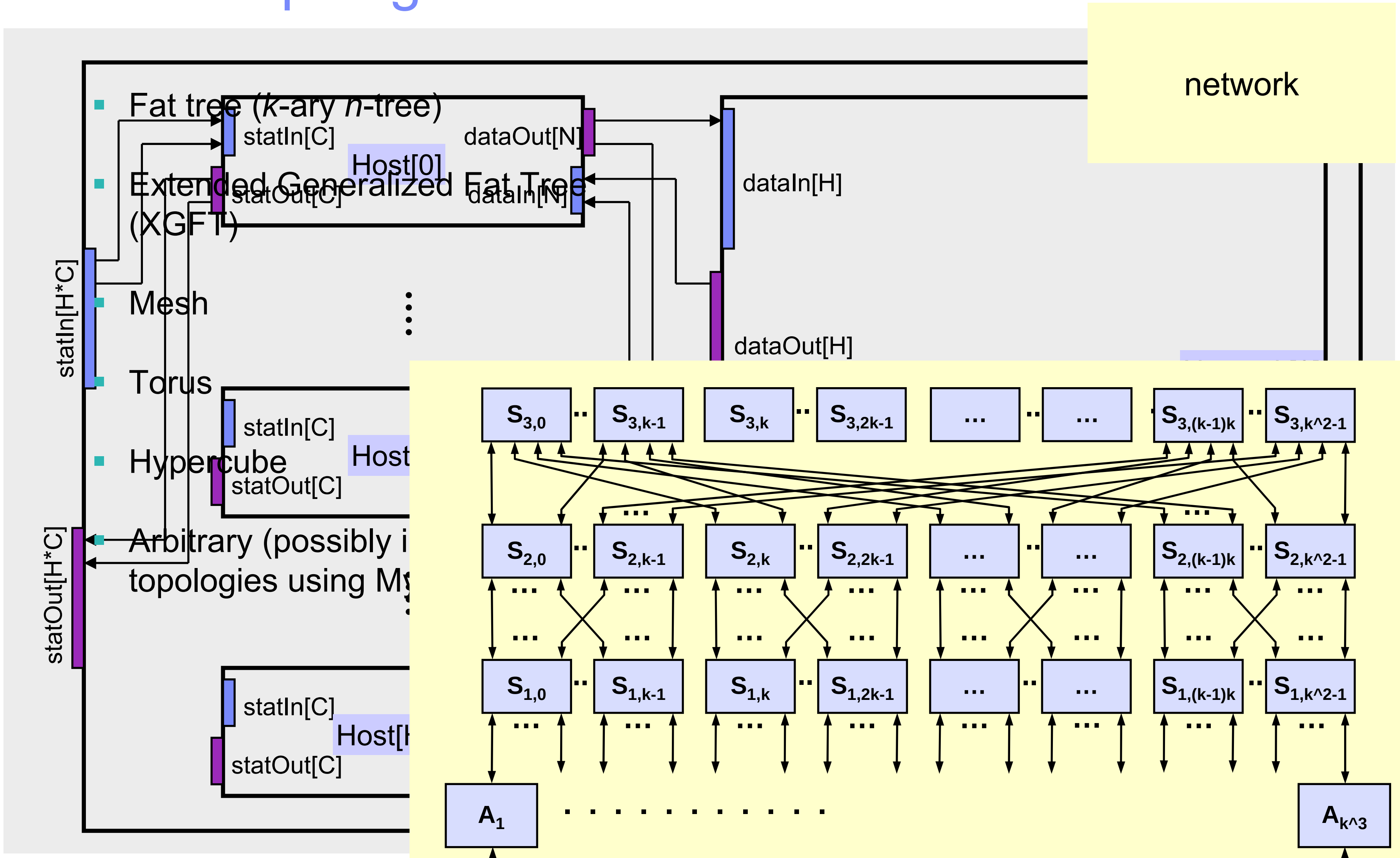
"END"

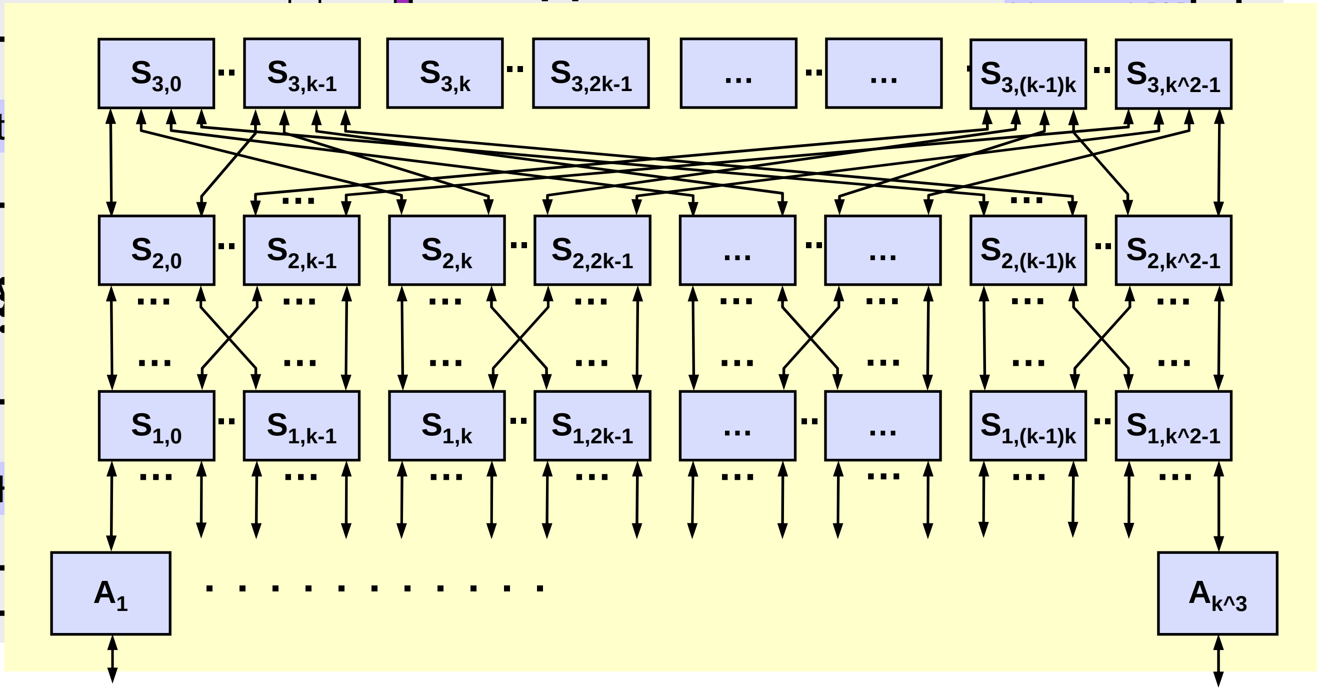
Inside pop_event()
- "**END**" received (unblock)
- Add "receive" event into the queue at time 2 us

Process Next Event

Network topologies

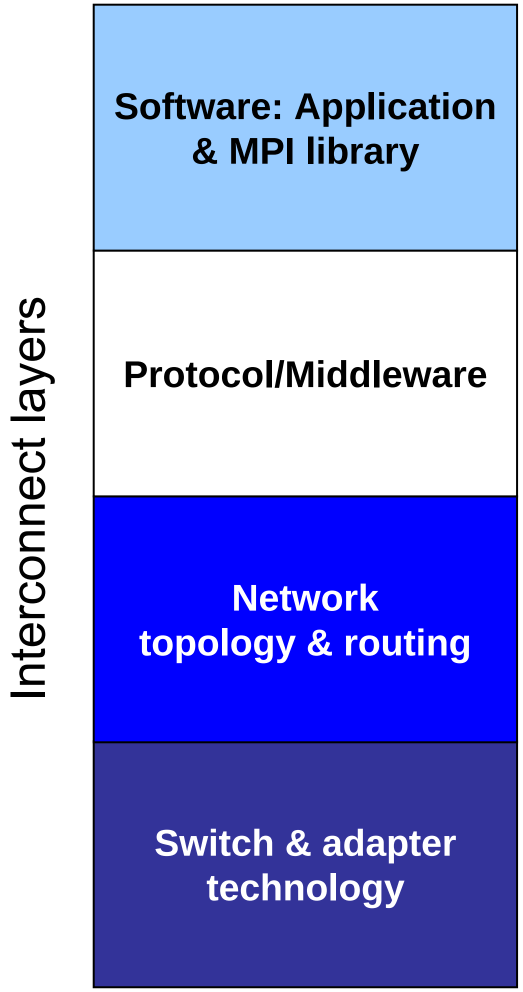
- Fat tree (*k*-ary *n*-tree)
- Extended Generalized Fat Tree (XGFT)
- Mesh
- Torus
- Hypercube
- Arbitrary (possibly i
  topologies using My





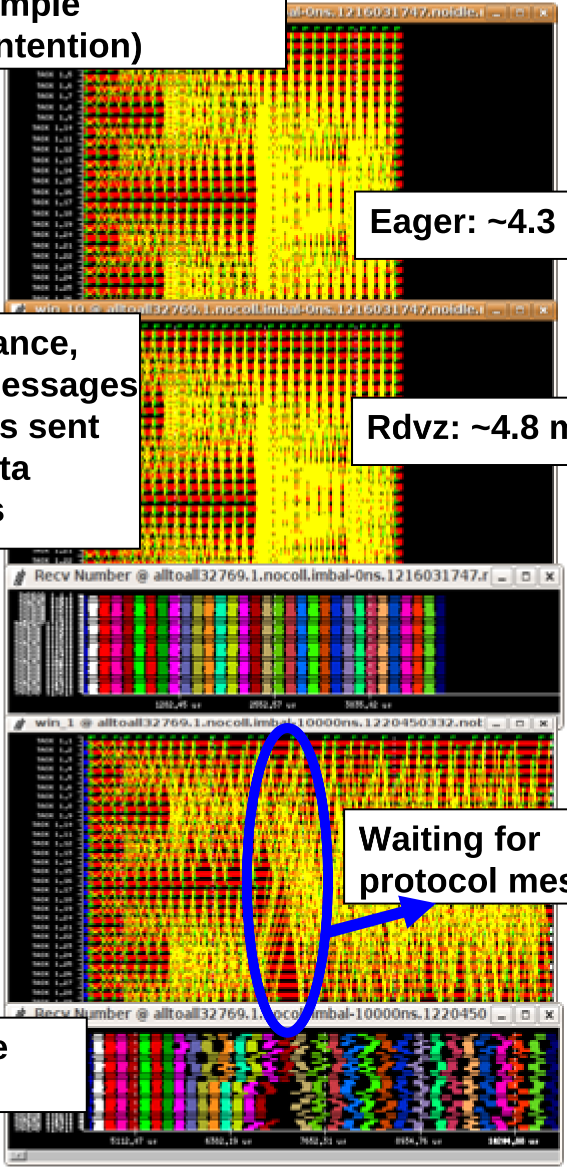
Insights gained at multiple levels

- **Application & MPI library level**
  - Blocking vs. nonblocking calls
- **MPI protocol level**
  - Eager vs. rendez-vous
- **Topology & routing**
  - Static source based routing
  - Contention and pattern aware routings
  - Slimmed networks
  - External vs. internal contention
- **Hardware**
  - Head-of-line blocking
  - Switch arbitration policy
  - Segmentation
  - Deadlock
  - Automatic communication-computation overlap
- **Putting it all together: Validation with real traces in a real machine**
  - Identifying the sources of network performance losses
  - Collectives variability

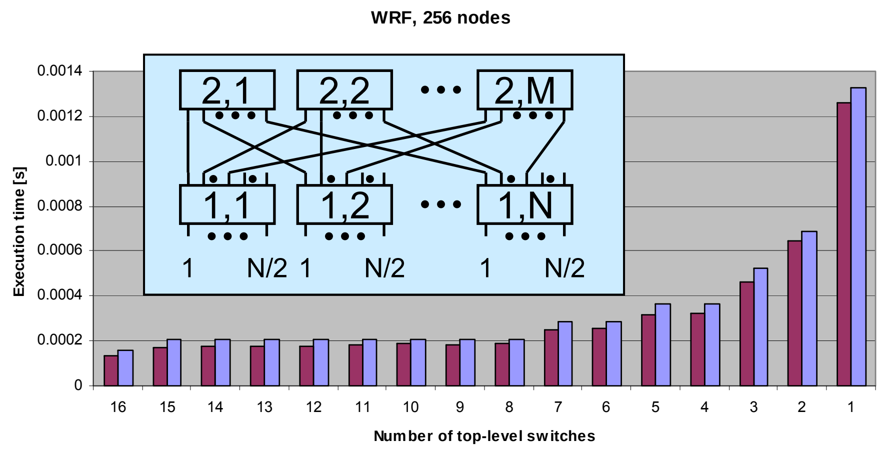

Interconnect layers
Software: Application & MPI library
Protocol/Middleware
Network topology & routing
Switch & adapter technology

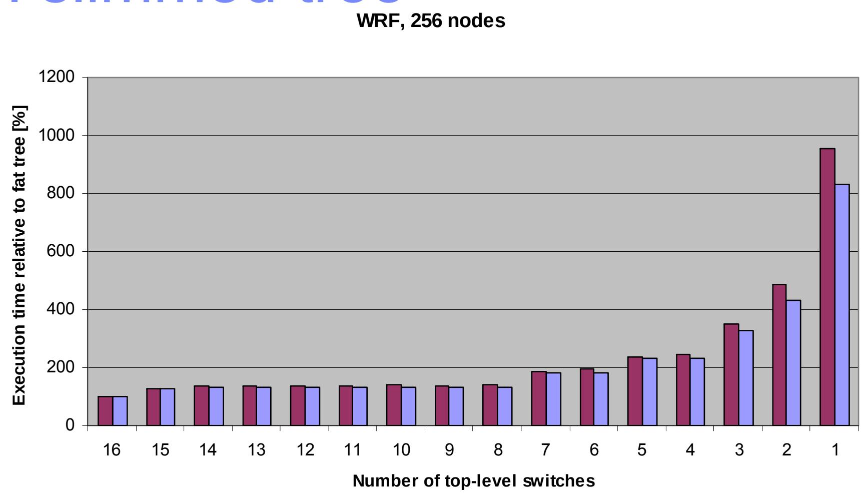
At the protocol level

- Eager/Rendez-vous
  - Rendez-vous protocol needs control messages
    - Impact is relatively small if the control messages never wait after a data segment to get sent
  - If control messages are sent after a long segment, the sender will be delayed
    - Smaller segments
    - Out of Band protocol messages
  - The delay propagates to the other threads
- Segment size trade-off?
  - 16KB as in Myrinet for long messages is too long to interleave urgent control messages

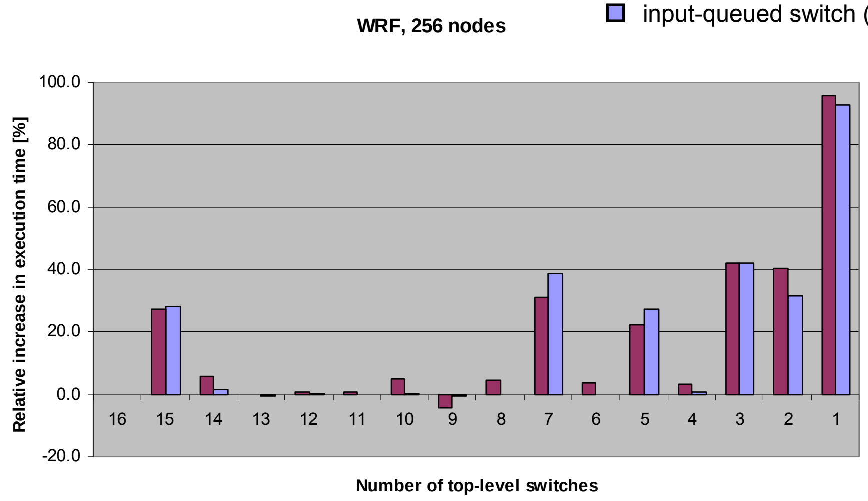

Alltoall example
(no internal contention)
Eager: ~4.3 ms
No imbalance, control messages are always sent before data segments
Rdvz: ~4.8 ms
Recv Number @ alltoall32769.1.nocoll.imbal-0ns.1216031747.r
Waiting for protocol messages
Very small imbalance <10us, time: ~6.3 ms

Performance of WRF on two-level slimmed tree





■ output-queued switch
■ input-queued switch (Myrinet)



- Dimemas
  - number of input links = 256
  - number of output links = 256
  - latency = 8 us
  - **cpu_ratio = 1.0**
  - eager threshold = 32768 bytes
- Venus
  - link speed = 2 Gb/s, flit size = 8 B (duration = 32 ns)
  - Myrinet switch and adapter models
  - Myrinet segmentation
  - adapter buffer size = 1024 KB
  - switch buffer size per port = 8 KB

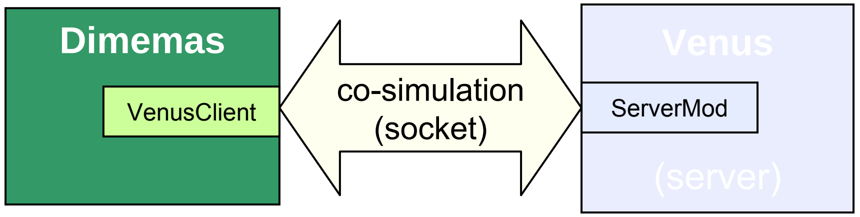
Conclusions & future work

- Conclusions
  - Created OMNeT-based interconnection network simulator that faithfully models real networks in terms of topology, routing, flow control, switch- and adapter architecture
  - Integrated network simulator with trace-based MPI simulator to capture reactive traffic behavior, thus obtaining high accuracy in simulating the interactions between computation and communication
  - Enabled entirely new insights into the effect of the interconnection network on overall system performance under realistic traffic patterns

- Future work
  - Allow multiple simulators to connect to Venus simultaneously
  - Modify MPI library to directly redirect communications to Venus
  - Make Venus itself run in parallel
  - Extended Venus to produce power estimates

BACKUP

D&V co-simulation: The client



```
/* DIMEMAS CODE */
int main() {
//DIMEMAS CODE
vc_initialize();

// EVENT MANAGER
While (events()  || venus_pending_events ()) {
     pop_event ();       BLOCKING
     // DIMEMAS CODE
     SEND: (…)
       vc_send(…)
     RDVZ_RECV: (…)
       vc_ready_recv(…)
     RDVZ_SEND: (…)
       vc_ready_send(…)
}
vc_finish()
// DIMEMAS CODE
}
```

Open socket to Venus
Create event queue for pending events

Modified conditions to stop the simulation – now the “normal” queue can be empty, because some relevant events are pending to be finished by Venus

Check if Venus communicated something

Block if necessary (check the number of messages in flight)
Extract event for “events” queue: Reception of messages in Venus is handled here

Extract the relevant event from the “events” queue, and put them into the “pending_events” queue

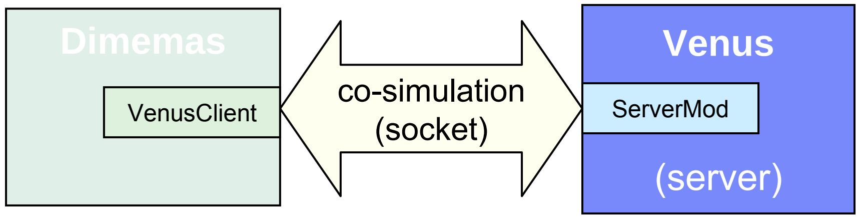
Send the commands to Venus through the socket

Increase the number of messages “in_flight”; this will cause pop_event() to relinquish control to Venus if necessary

Close the socket

D&V co-simulation: The server

- ServerMod is interposed between the statistics module and the network.
- Acts as a Generator and Sink of “User Messages”



```
/* SERVERMOD CODE */
int initialize() {
}
int receive_commands() {
   while (command = read_from_socket())
       execute (command);
}

int handleMessage() {
   STOP:
       waitForCommands = true;
       send_through_socket(STOP REACHED);
   USER_MESSAGE:
       waitForCommands = callback(UserMessage);
      // [ . . . ]
   if (waitForCommands)
       receive_commands();      B L O C K I N G
}
```

Bind/listen/accept (server socket)

Insert into the network “User Messages” as instructed by Dimemas or schedule “STOP”s to relinquish control back to Dimemas

Commands are read until an “END” command is found; then Venus simulation continues

If a STOP is reached, tell Dimemas and block (in receive_commands()) until Dimemas schedules a next safe “Earliest Output Time” (STOP)

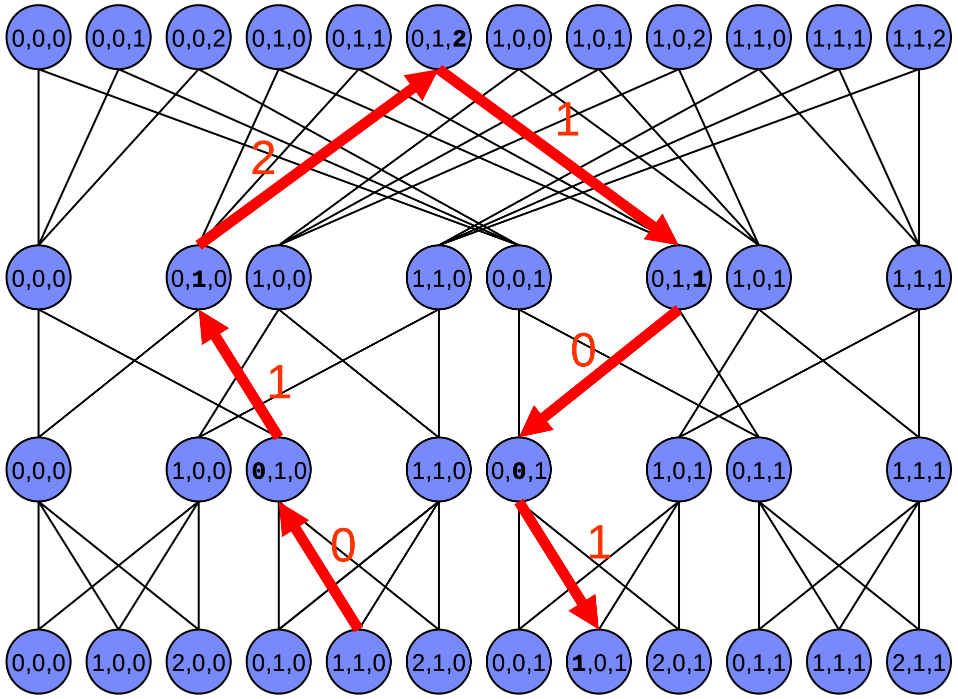
When ServerMod sees a “User Message” it executes the associated callback:

- If it is a protocol message, new messages are injected into the network
- If it is a Data message that completed, inform Dimemas, cancel the scheduled STOP, and block (receive_commands);

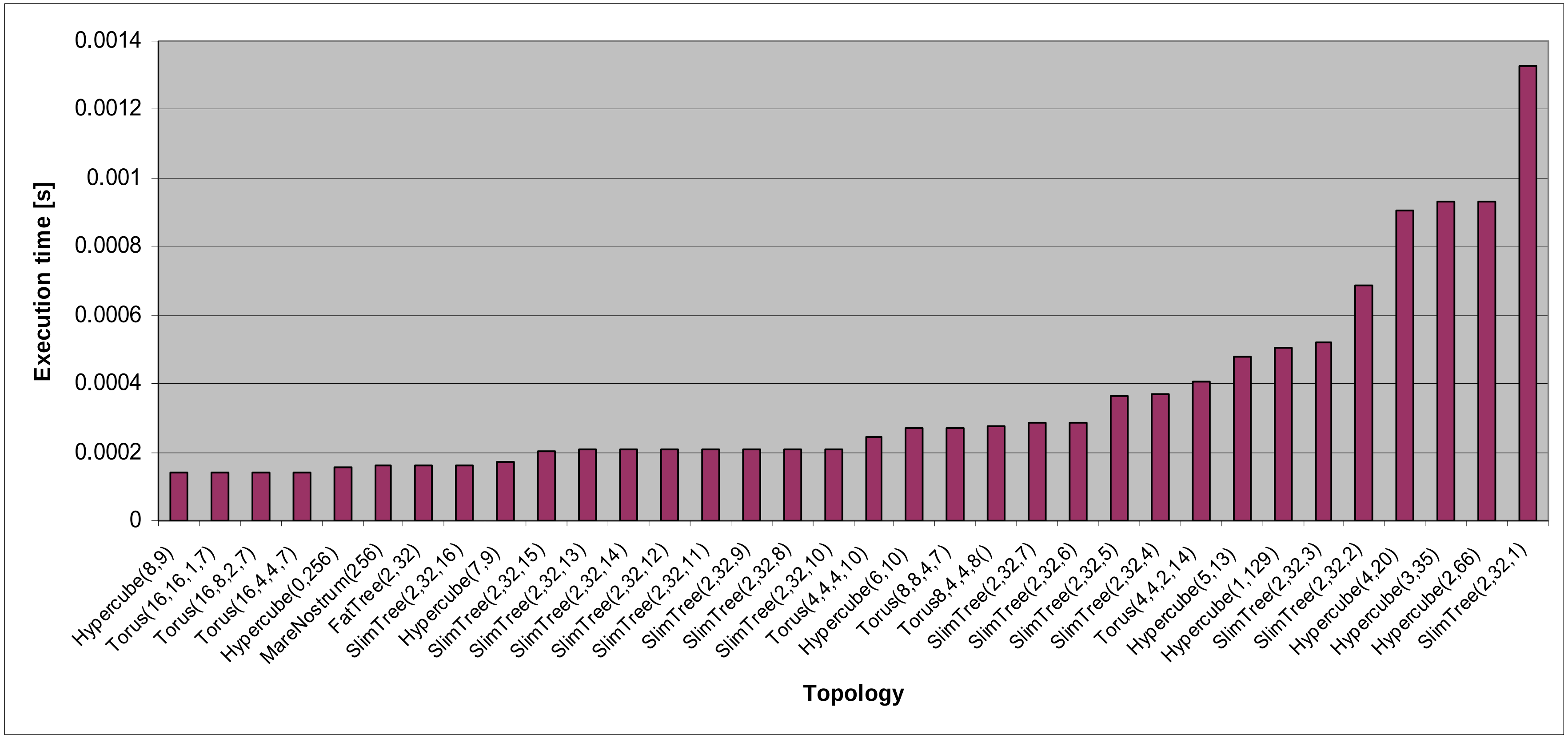
Extended Generalized Fat Trees

- XGFT (h ; m_1, … , m_h; w_1, … , w_h)
- h = height
  - number of levels-1
  - levels are numbered 0 through h
  - level 0 : compute nodes
  - levels 1 … h : switch nodes
- m_i = number of children per node at level i, $0 < i \leq h$
- w_i = number of parents per node at level i-1, $0 < i \leq h$
- number of level 0 nodes = $\prod_i m_i$
- number of level h nodes = $\prod_i w_i$

XGFT (3 ; 3, 2, 2 ; 2, 2 ,3)



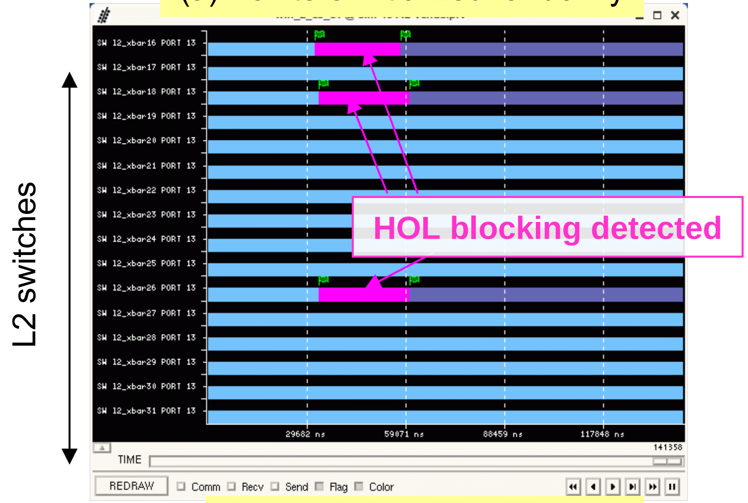
Performance of 256-node WRF on various topologies – cpu ratio 1.0



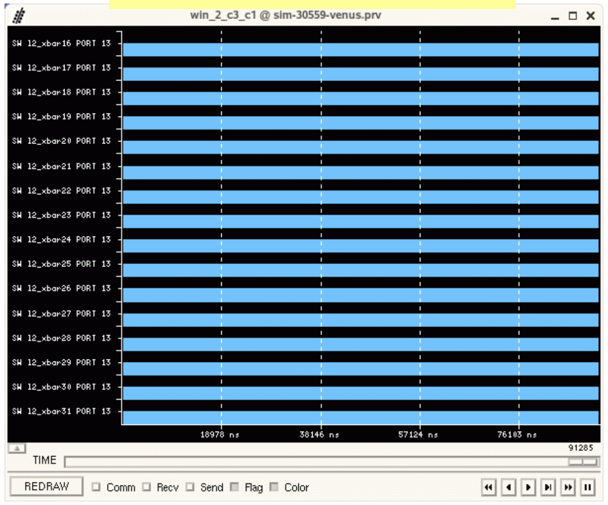
Impact of scheduling policy

- The **scheduling policy** is the determining factor (infinite CPU, IQ switch)
  - Round-robin pointers are initialized randomly
  - When contention occurs (on output 15), the “wrong” selection (which depends on the position of the pointer) results in HOL blocking; the scheduler should first serve the input on which another message will arrive
  - Unfortunately, it has no crystal ball…

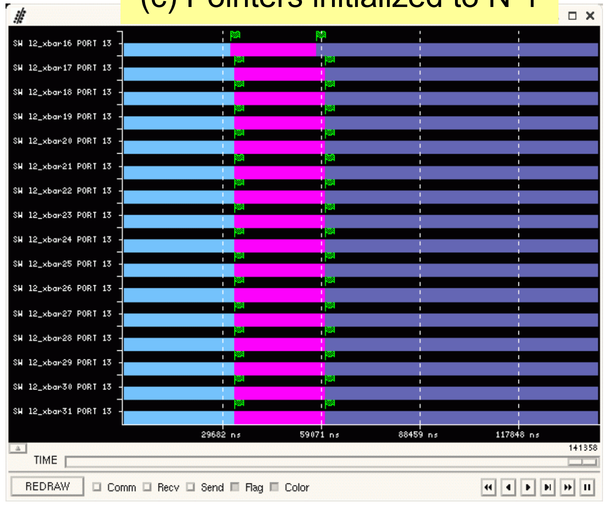
(a) Pointers initialized randomly



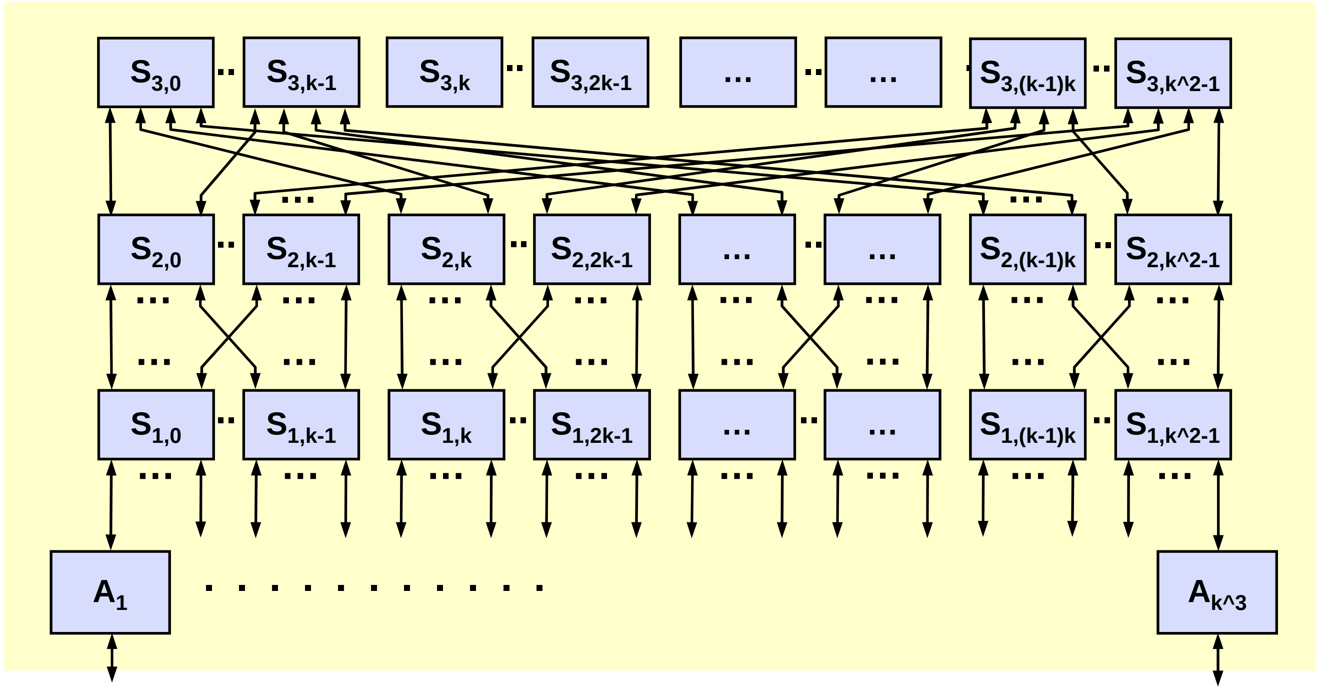
(b) Pointers initialized to 0



(c) Pointers initialized to N-1

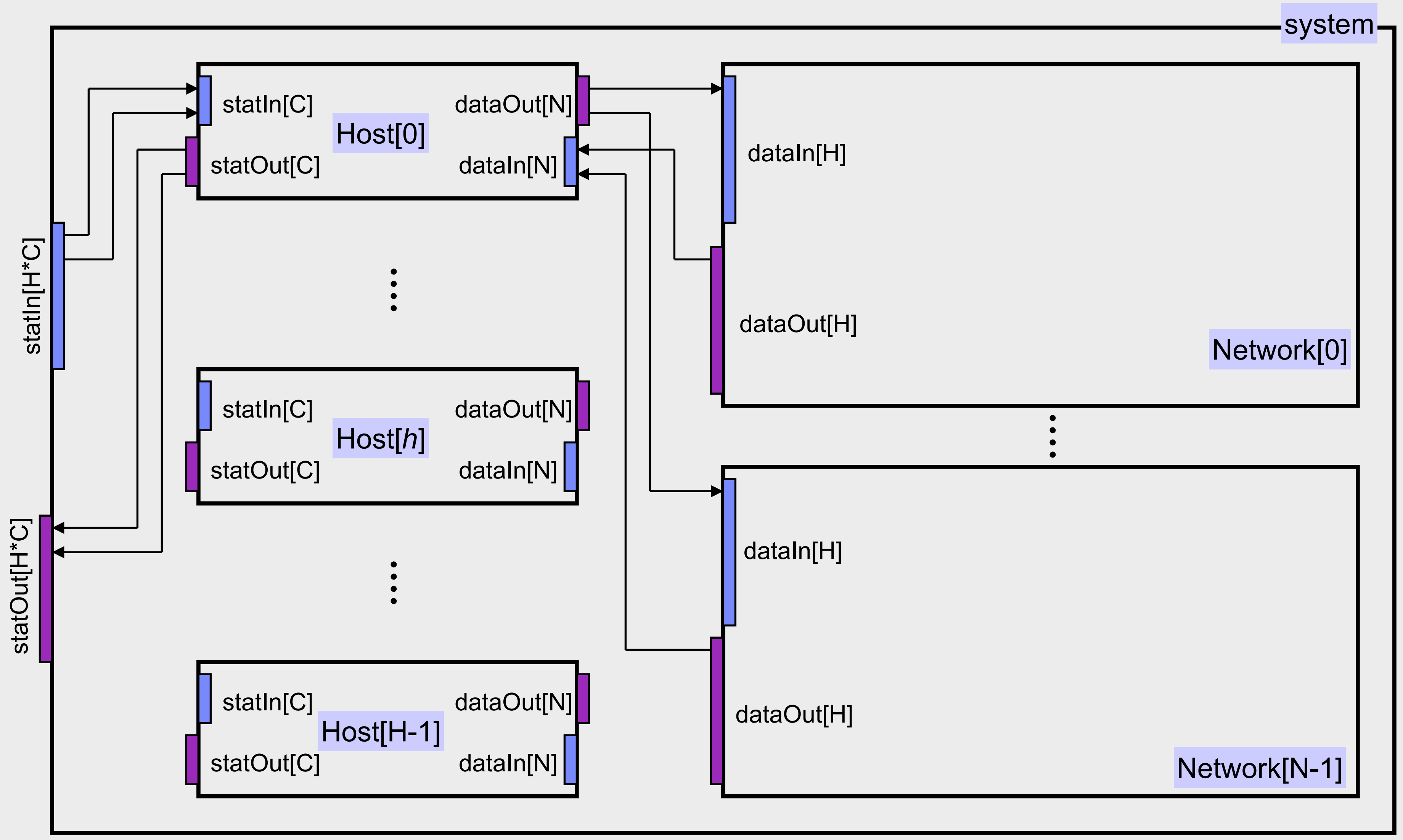


Backup

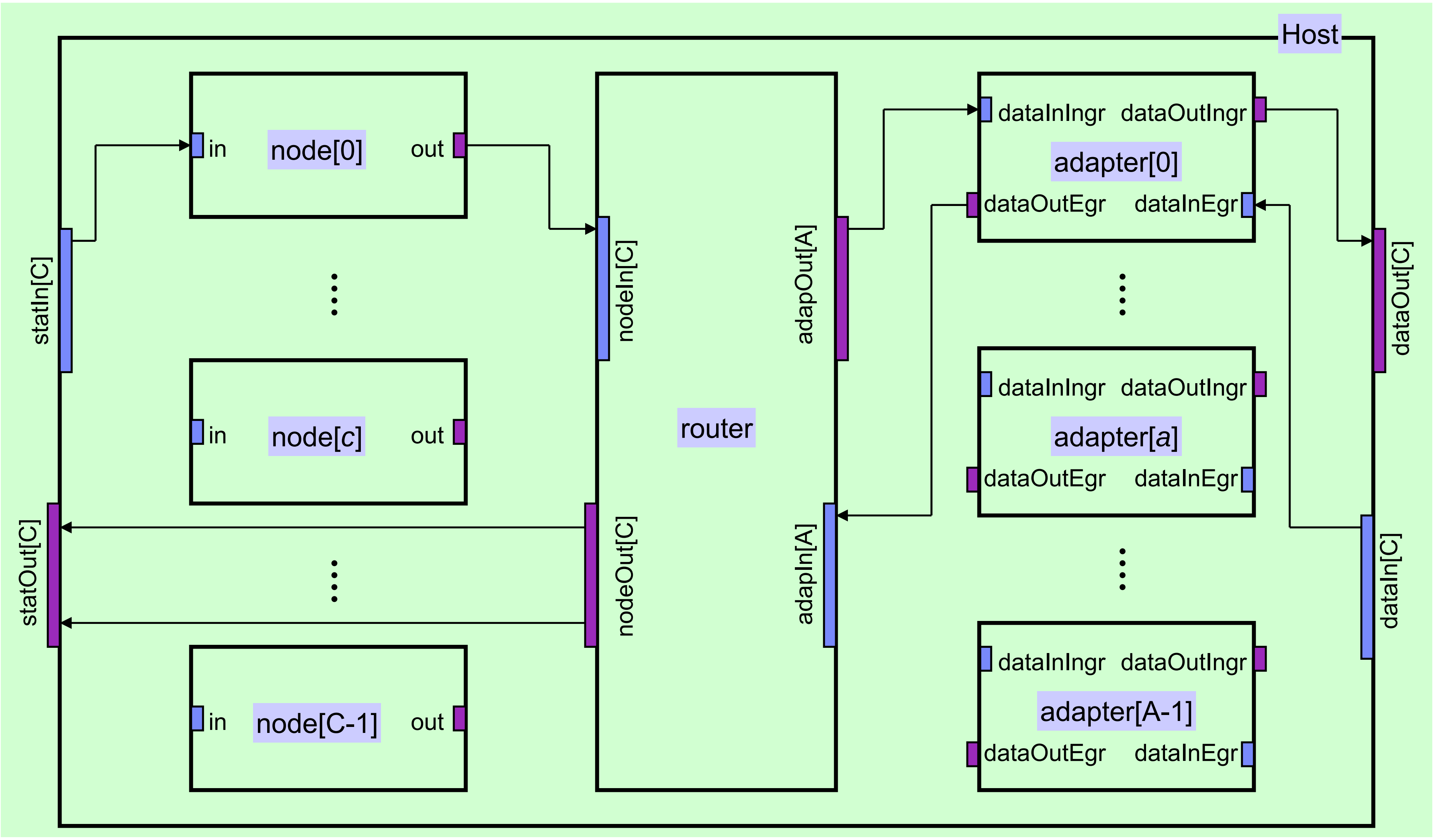


$S_{3,0}$
$S_{3,k-1}$
$S_{3,k}$
$S_{3,2k-1}$
…
…
$S_{3,(k-1)k}$
$S_{3,k\text{^}2-1}$
$S_{2,0}$
$S_{2,k-1}$
$S_{2,k}$
$S_{2,2k-1}$
…
…
$S_{2,(k-1)k}$
$S_{2,k\text{^}2-1}$
$S_{1,0}$
$S_{1,k-1}$
$S_{1,k}$
$S_{1,2k-1}$
…
…
$S_{1,(k-1)k}$
$S_{1,k\text{^}2-1}$
A_1
$A_{k\text{^}3}$

Multi-rail system

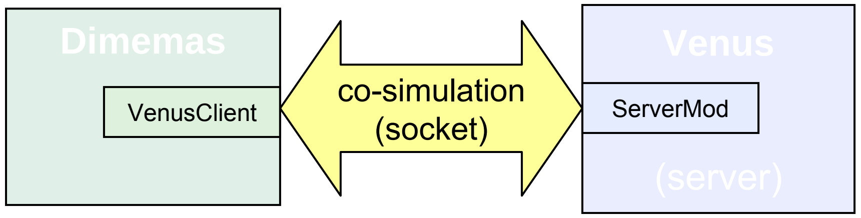


MultiHost module



D&V Co-simulation: Protocol (hints)

- Human-readable ASCII text
- Sent through standard Unix Sockets
- Few Commands and Responses
- Extensible

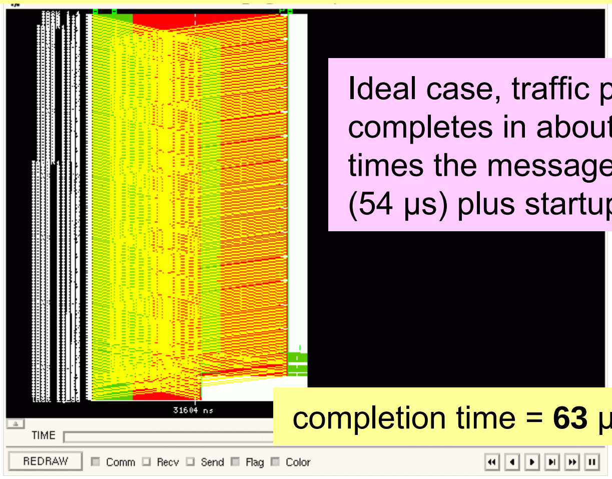


- Commands
  - SEND timestamp source destination size “extra strings”
    - The “extra strings” enconde the events that Dimemas wil update upon reception of the message.
  - STOP timestamp
  - END
  - FINISH
  - PROTO (OK_TO_SEND,READY_TO_RECV)
    - (parameters as for SEND)

- Responses:
  - STOP REACHED timestamp
  - COMPLETED SEND [echo of the original data + extra data]

Impact of switch architecture

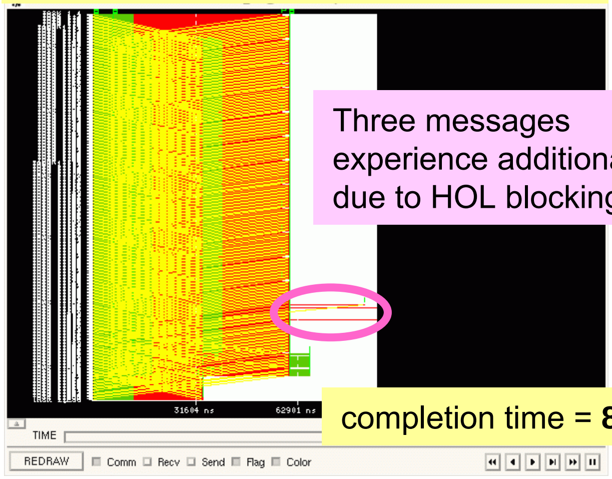
Infinite CPU speed, output-queued switch



Ideal case, traffic pattern completes in about two times the message duration (54 µs) plus startup latency

completion time = **63** µs

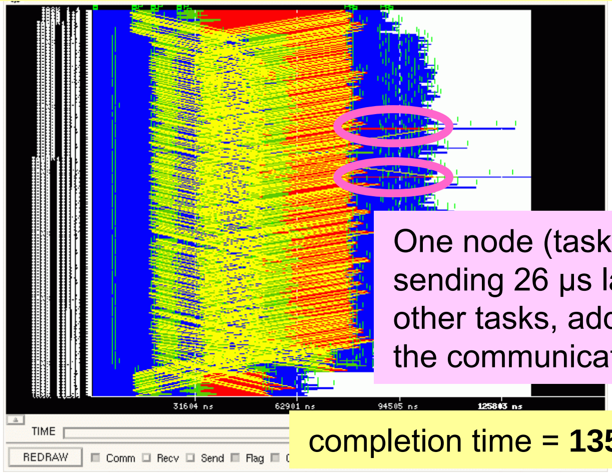
Infinite CPU speed, input-queued switch



Three messages experience additional delay due to HOL blocking. Why?

completion time = **87** µs

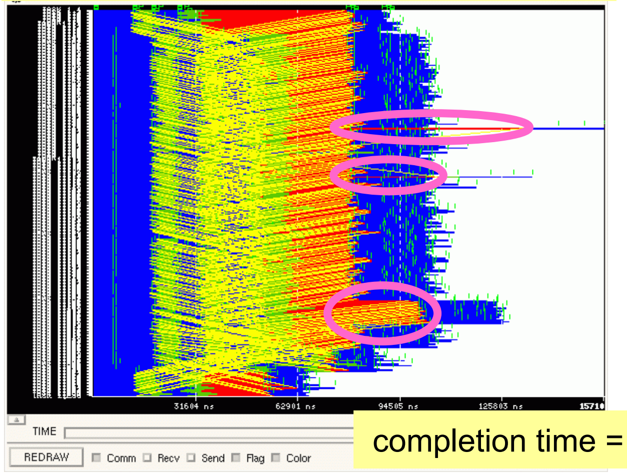
Normal CPU speed, output-queued switch



One node (task 95) starts sending 26 µs later than all other tasks, adding 26 µs to the communication time

completion time = **135** µs

Normal CPU speed, input-queued switch



completion time = **157** µs