ModSim at Tactical Computing Labs

John Leidel, Chief Scientist
tactcomplabs.com

Outline

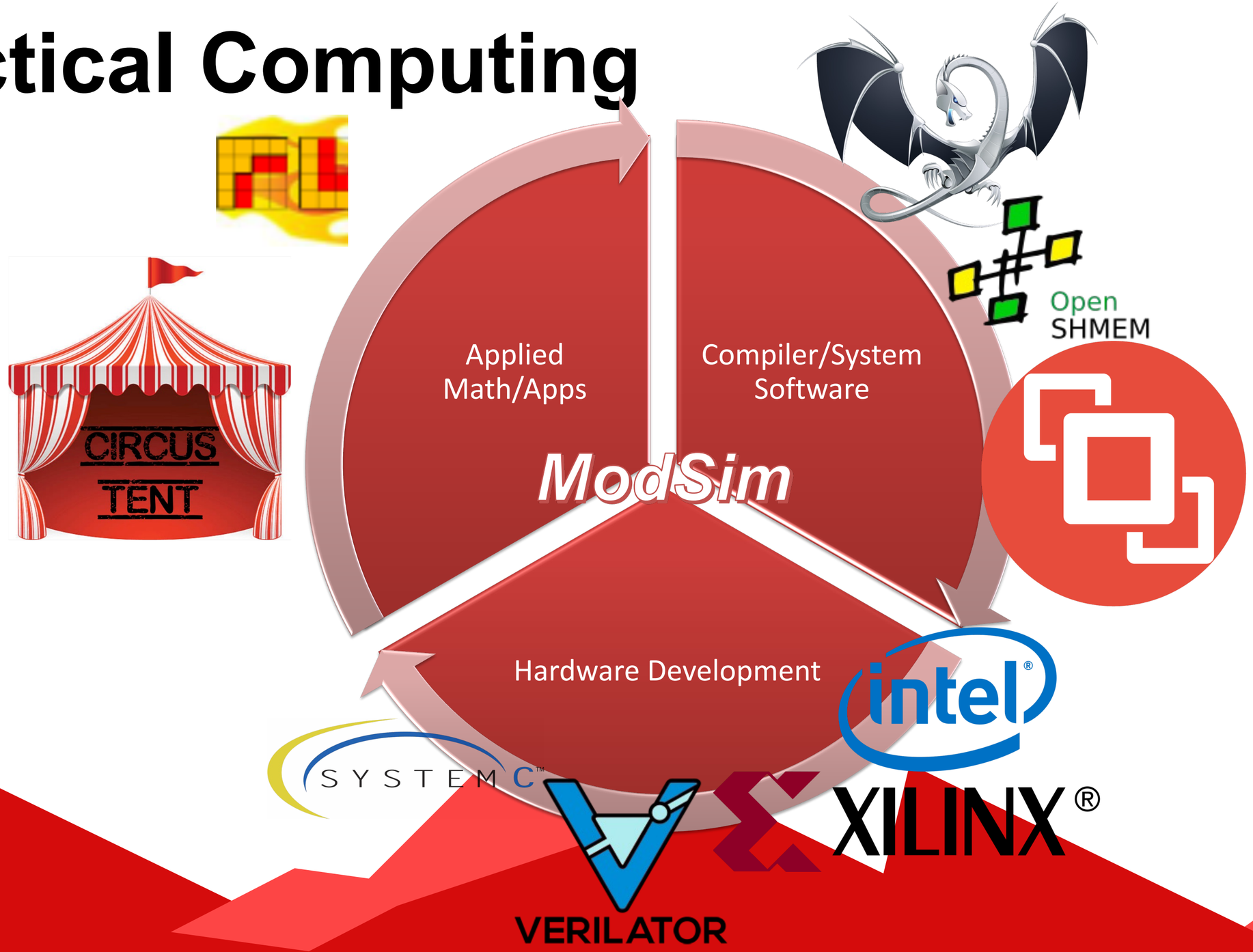
MODSIM HISTORY

CURRENT PROJECTS

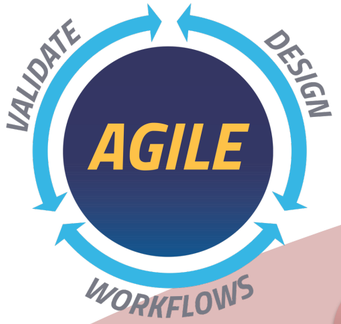
FUTURE EXPERIMENTS

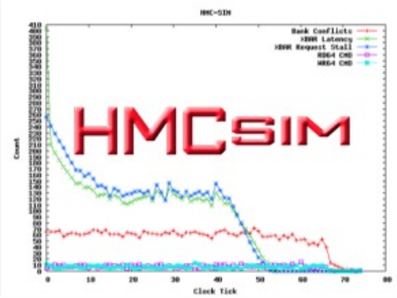
History of ModSim at TCL

Tactical Computing



History of ModSim at TCL





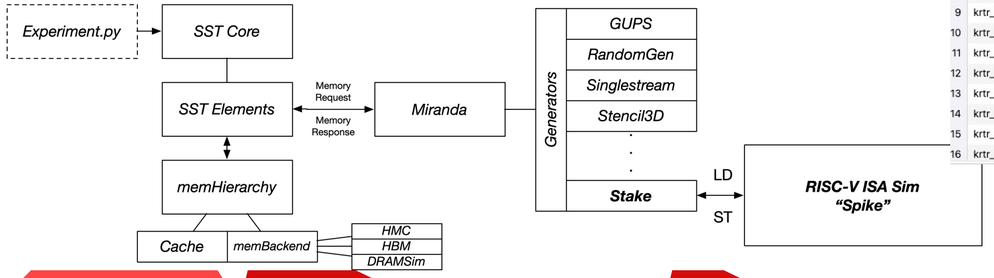
HMCSim

Stake

Rev

AGILE

SST-Data,
SST-Bench,
SST-Dbg,
VerilatorSST



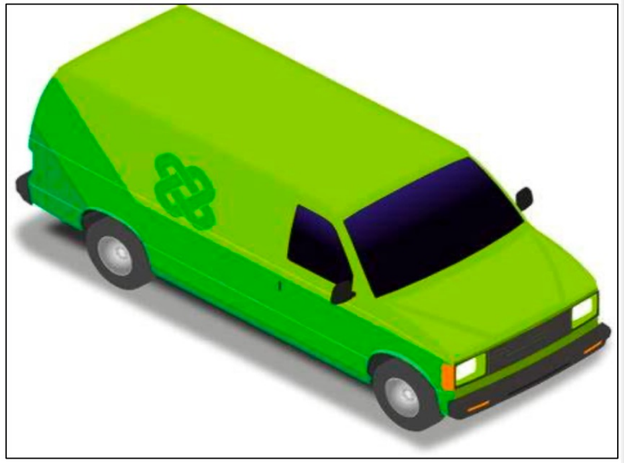
	Component...	StatisticName	StatisticSubId	StatisticType	Sum_u64	SumSQ_u64
	Search column...	Search column...	Search column...	Search column...	Search column...	Search colum
1	krtr_req_0	send_bit_count	north	Accumulator	144576	
2	krtr_req_0	output_port_stalls	north	Accumulator	0	
3	krtr_req_0	xbar_stalls	north	Accumulator	93	
4	krtr_req_0	send_bit_count	south	Accumulator	200000	
5	krtr_req_0	output_port_stalls	south	Accumulator	6	
6	krtr_req_0	xbar_stalls	south	Accumulator	180	
7	krtr_req_0	send_bit_count	east	Accumulator	88512	
8	krtr_req_0	output_port_stalls	east	Accumulator	0	
9	krtr_req_0	xbar_stalls	east	Accumulator	13	
10	krtr_req_0	send_bit_count	west	Accumulator	0	
11	krtr_req_0	output_port_stalls	west	Accumulator	0	
12	krtr_req_0	xbar_stalls	west	Accumulator	0	
13	krtr_req_0	send_bit_count	local0	Accumulator	0	
14	krtr_req_0	output_port_stalls	local0	Accumulator	0	
15	krtr_req_0	xbar_stalls	local0	Accumulator	0	
16	krtr_req_0	send_bit_count	local1	Accumulator	48192	

Current Projects

IARPA AGILE

- ***Goal:*** Design and develop a scalable HPC system for extreme scale analytics workloads
- ***Challenge 1***: The workloads do no execute efficiently today
- ***Challenge 2***: We have 36 months to complete the task
  - **18 months**: full system simulation
  - **18 months**: macro scale emulation

AGILE Program Details



Program Objectives:

- Enable data analytic problems that involve 10X more data.
- Time to solution 10 - 100 times faster.

Research Effort:

- Develop validated designs that achieve or exceed the AGILE Program Target Metrics.
- These results will be validated by an independent test and evaluation team.

Deliverables:

- **Phase 1:** System-level functional model of architecture. Including runtime.
- **Phase 2:** Detailed (RTL) design for proposed AGILE system architecture, including runtime.

https://www.iarpa.gov/research-programs/agile

Summary

AGILE Program has three tier evaluation process:

1. **End-to-end applications (Workflows)** that measure full system performance
   - Data sets at different scales
   - Data ingestion and preparation
   - Multiple computational components
2. **Kernels derived from Workflows**
3. **Industry standard benchmarks (ISBs)**
   - Breadth-first search
   - Triangle counting
   - Jaccard coefficient

AGILE utilizes ModSim to estimate and evaluate the performance of the design models, when executing the AGILE Applications

AGILE Workflow: Target Metrics

Workflow 1: Knowledge Graph		
Metric	**Today**	**AGILE Target**
Data Ingestion Rate (file): Time to read a data file and build internal data structures	1 G (G=10^9) data element file per 1 minute	1 G data-element per 1 second (60x faster)
Data Ingestion Rate (streaming): Time to process streaming data and insert data into internal data structures	0.1 G data-elements per second from a single source, single data type	10 G data-elements per second from 3 or more sources and data types (100x faster for each of 3 sources)
Learn models: Time to construct embedding and train GNN models	1,440 minutes	30 minutes (50x faster)
Classify vertices: Time to retrain model and classify unlabeled vertices in data streams	> 1,440 minutes	30 minutes (50x faster)
Predict and infer a new relationship: Time to retrain model and infer a new relationship in data streams	> 1,440 minutes	30 minutes (50x faster)
Perform reasoning: Time to reason about higher-order relationships using multi-hop reasoning	1 to 2 hops and branching factor not greater than 3 in 30 minutes	3 to 5 hops and score dependent branching in a minute (30x faster)

Workflow 2: Detection		
Metric	**Today**	**AGILE Target**
Data Ingestion Rate (file): Time to read a data file and build internal data structures	1 G (G=10^9) data element file per minute	1 G data-element per 1 second (60x faster)
Data Ingestion Rate (streaming): Time to process streaming data and take action (insert/delete/modify) on internal data structures	0.1 G data-elements per second from a single source, single data type	10 G data-elements per second from 3 or more sources and data types (100x faster for each of 3 sources)
Exact Pattern Matching	Patterns with a less than 10 events and branches completed in minutes	Patterns with more than 20 events and branches with time and locality constraints completed in seconds (60x faster)
Approximate Pattern Matching	NOT DONE	Patterns with more than 20 events and branches with time and locality constraints completed in seconds
Partial Pattern Matching with alerts	NOT DONE	Alerts issued in real time

Workflow 3: Sequence Data		
Metric	**Today**	**AGILE Target**
Data ingestion and record processing rate	1.3 M records per second from a single source	15 M records per second from 3 or more sources (10x faster for 3 sources)
Time to extend de Bruijn graphs merge bubbles and remove hairs (Steps A and B)	58 hours	1.5 hours (40x faster)
Time to iteratively prune de Bruijn graph (Step C)	227 hours	4.5 hours (50x faster)
Time to align and assemble contigs (Steps D and E)	113 hours	2.3 hours (50x faster)
Time to align and classify final contigs with known genomes (Not shown in Figure 15)	NOT DONE	Completed

Workflow 4: Networks of Networks		
Metric	**Today**	**AGILE Target**
Construct 1 PB network-of-network graph	120 minutes	2 minutes (60x faster)
Identify top *k* influential nodes (simple model)	60 minutes	1 minute (60x faster)
Identify top *k* influential nodes (enhanced model)	600 minutes	30 minutes (20x faster)
Model disruption and corrective actions	120 minutes	2 minutes (60x faster)
Incremental analysis	NOT DONE	Never recomputed from scratch

Goal: Co-designed Software+Hardware for FORZA Execution Model

Software

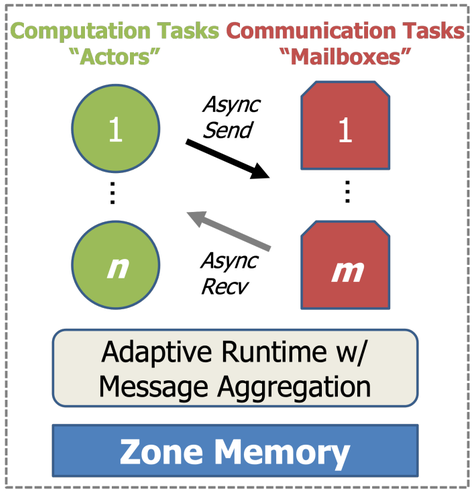
Workflow-driven Co-design

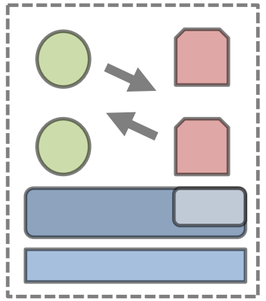
Hardware

New Algorithms + New Runtime
for Asynchronous Visit Model

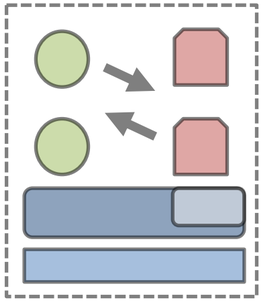
1 *precinct* ("0.25U node") contains 8 *zones*
1 zone contains 4 *ZAPs* ("cores")
1 ZAP *weave core* contains 512 *HARTs*

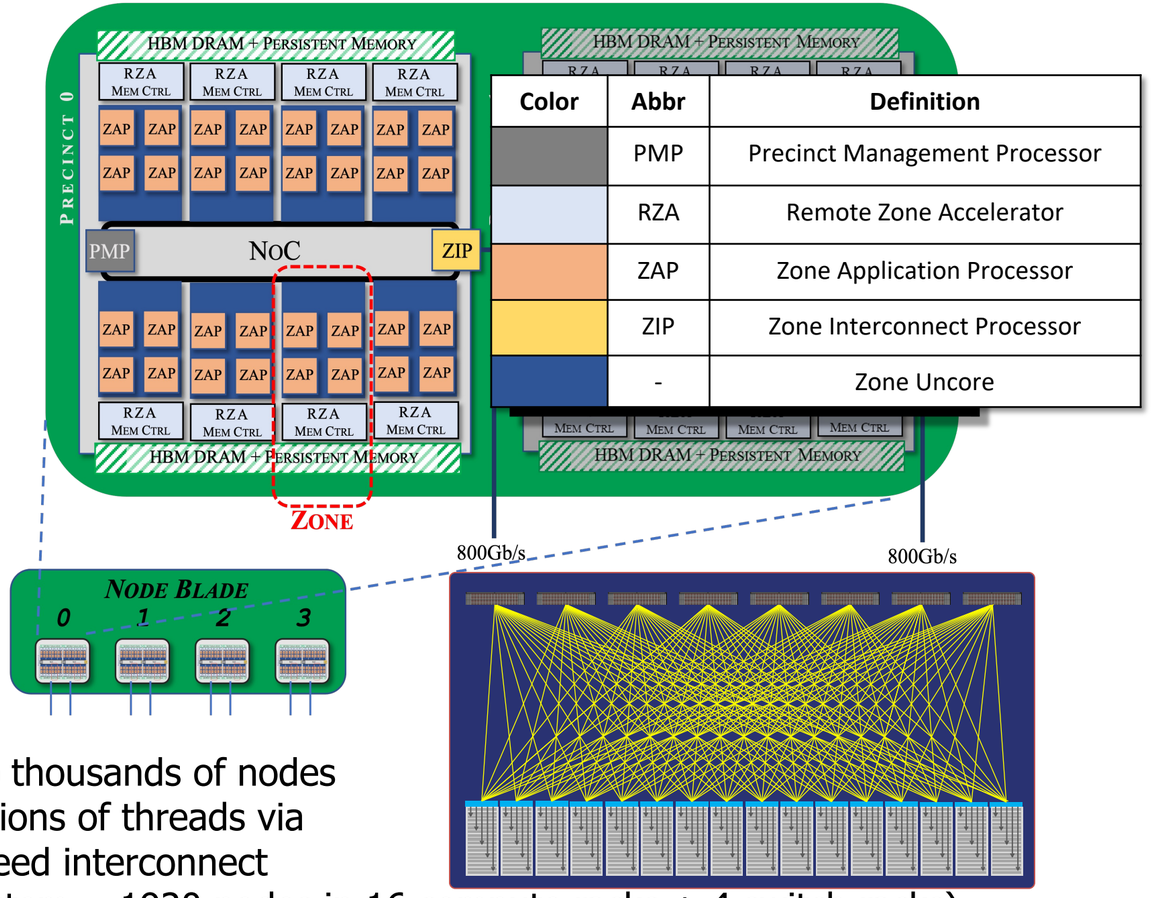
```
for {l_ij, l_ik ∈ L̂ | l_ij = l_ik = 1, j < k} do
    Let pe ← FindOwner(L_jk)   ▷ Find a rank that owns L_jk
    Actor_p.send(pe, j, k)   ▷ Send an active message (non-blocking)
Wait()   ▷ Wait for the completion of local send/recv
return AllReduce(c)
function ActorProcess(j, k)   ▷ The message handler: j is row number, and k is col number
    if {l_jk ∈ L̂ | l_jk = 1} then c += 1
function FindOwner(row)   ▷ Returns a rank that is responsible for row
    return row % P   ▷ 1-D Cyclic distribution
```





Massive asynchronous parallelism . . .





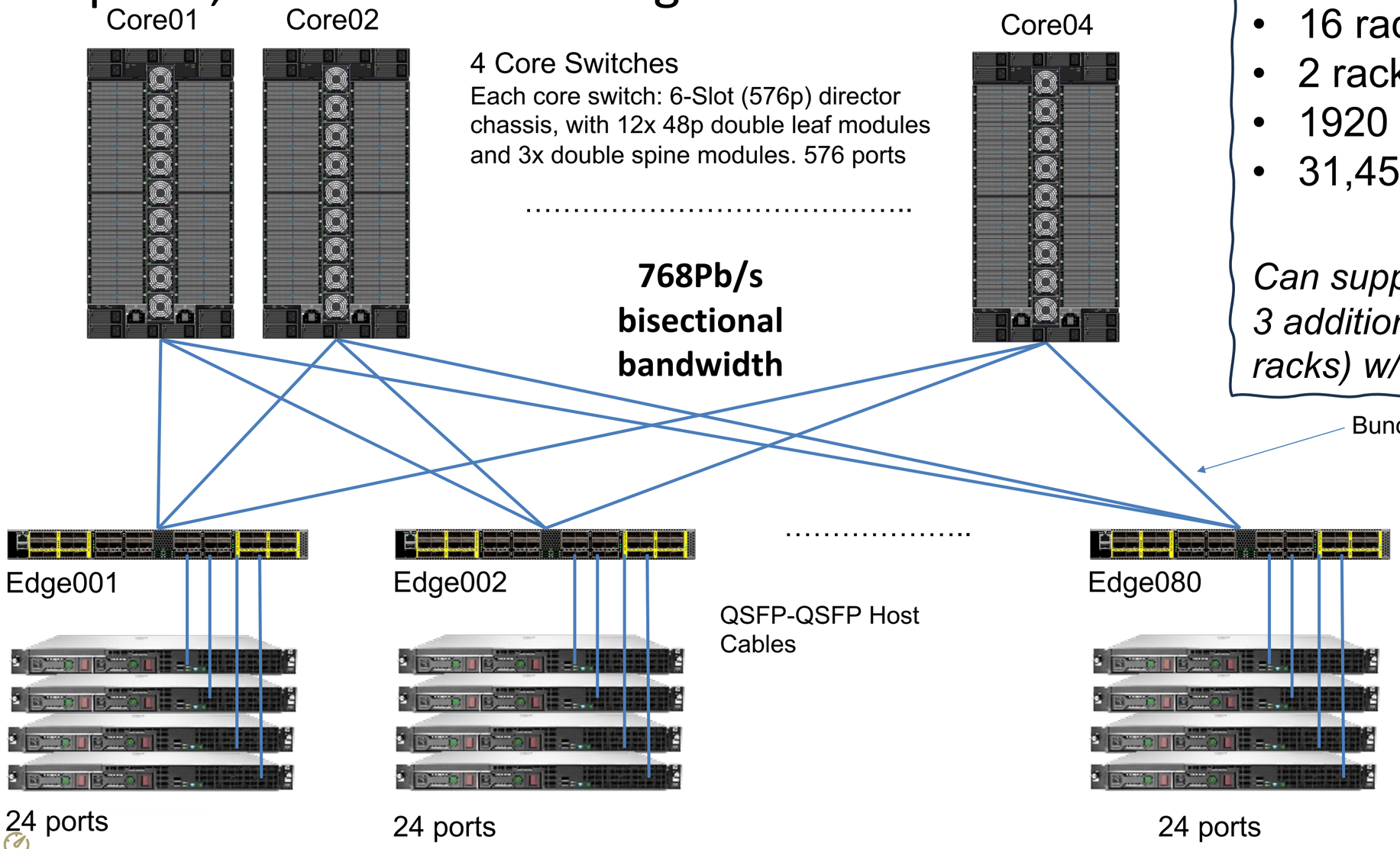
Color	Abbr	Definition
	PMP	Precinct Management Processor
	RZA	Remote Zone Accelerator
	ZAP	Zone Application Processor
	ZIP	Zone Interconnect Processor
	-	Zone Uncore

Scale to thousands of nodes and millions of threads via high-speed interconnect
(Full system = 1920 nodes in 16 compute racks + 4 switch racks)

FORZA

Full Bisection Bandwidth Tree

2304 ports, 3-tier non-blocking fat-tree

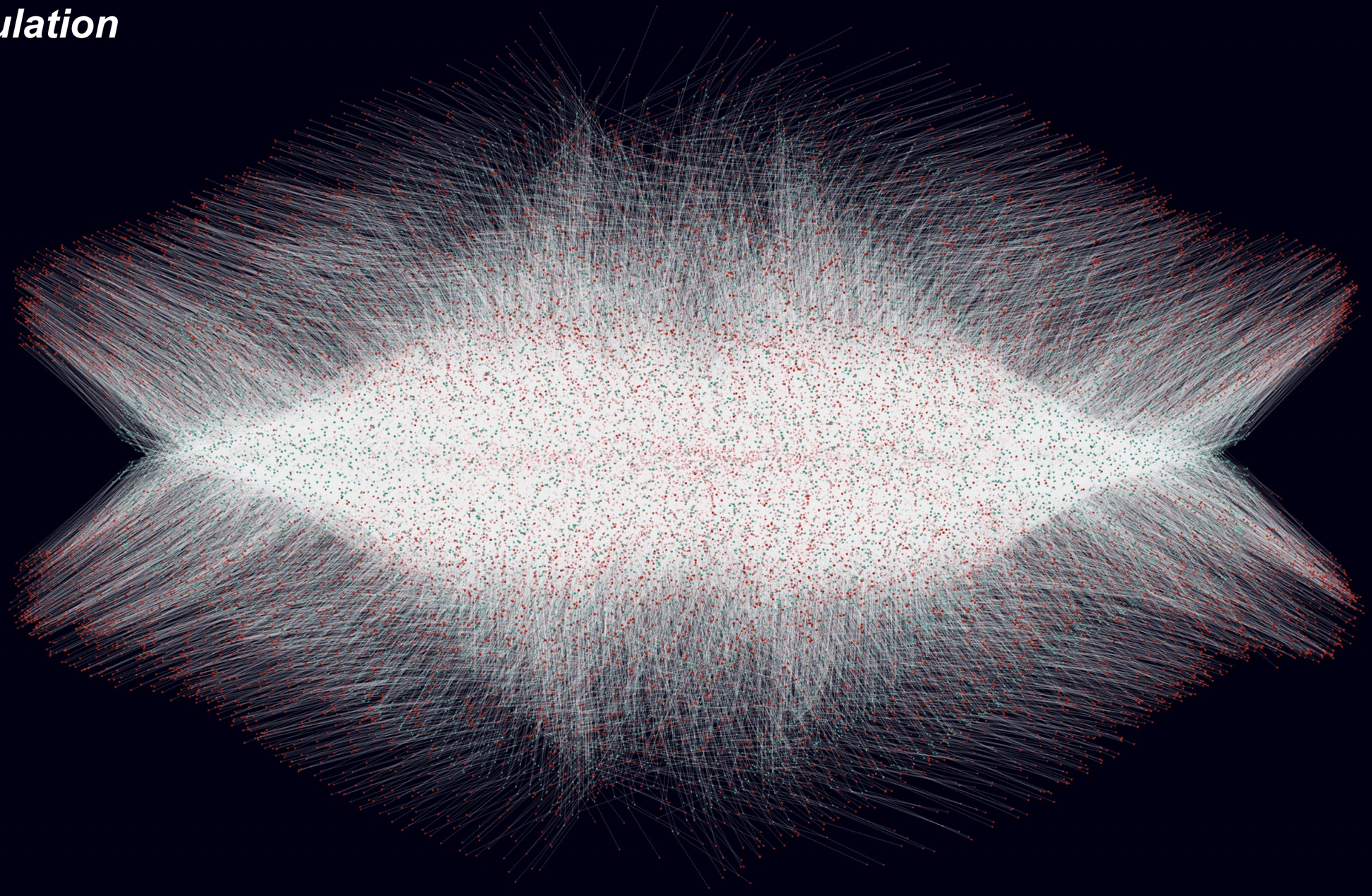


- 16 racks of compute
- 2 racks for core switches
- 1920 precincts total
- 31,457,280 threads

Can support up to 2304 precincts (w/ 3 additional compute racks OR 48U racks) w/o additional core switches

FORZA

Phase I: 32,768 node simulation graph



What does this mean for modeling and simulation?

SST Tooling

scripts	adding get_sst_element to the cmake infrastructure	2 months ago
FindSSTCore.cmake	adding get_sst_element to the cmake infrastructure	2 months ago
LICENSE	Initial commit	2 months ago
README.md	Adding find_package support to SST & Adding README	2 months ago

README Apache-2.0 license

Find Package Support for SST

Add the following to your SST CMake Project:

```
# Set this to the directory containing FindSSTCore.cmake
list(APPEND CMAKE_MODULE_PATH "/path/that/contains/FindSSTCore.cmake")

# Attempt to find SST Core
find_package(SSTCore REQUIRED)
```

Then, assuming you are making a library (myComponent)... Add the available `SST_INCLUDE_DIRS` to your project

```
# Add an executable that uses SST Core
add_executable(myComponent myComp.cpp)
target_include_directories(sst_test PRIVATE ${SSTCORE_INCLUDE_DIRS})
```

NOTE: The `FindSSTCore.cmake` file requires you to pass `-DSST_INSTALL_DIR=${install_prefix}`

SST-Dbg

- Debugging simulation components is hard!
- Often necessary to record additional data outside of SST::Statistics
- This is a common API and interface to do so!

SST-Data

- Compute bound to I/O bound?
- AGILE simulation runs create 10's TB's of statistics data
- This provides additional storage mechanisms and in-situ visualization

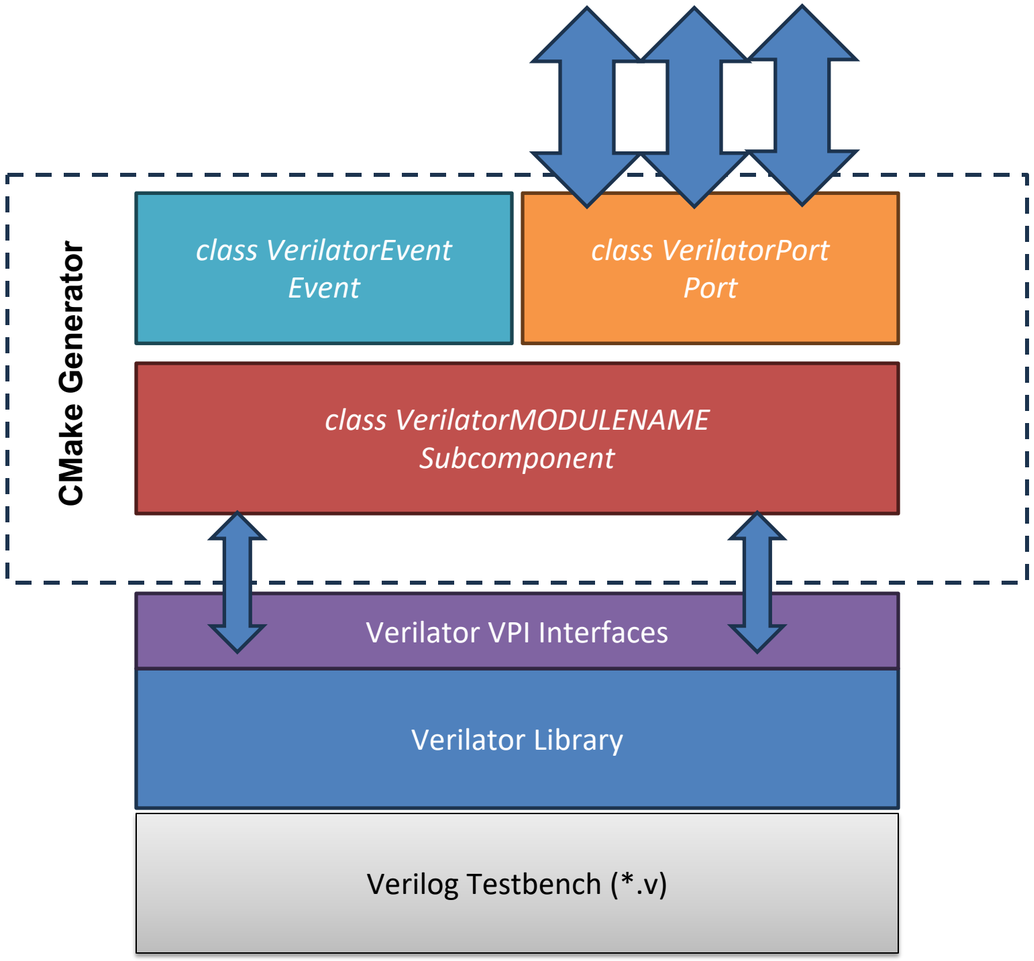
SST-Build/SST-Test

- Cmake-based common build & test harnessess

VerilatorSST

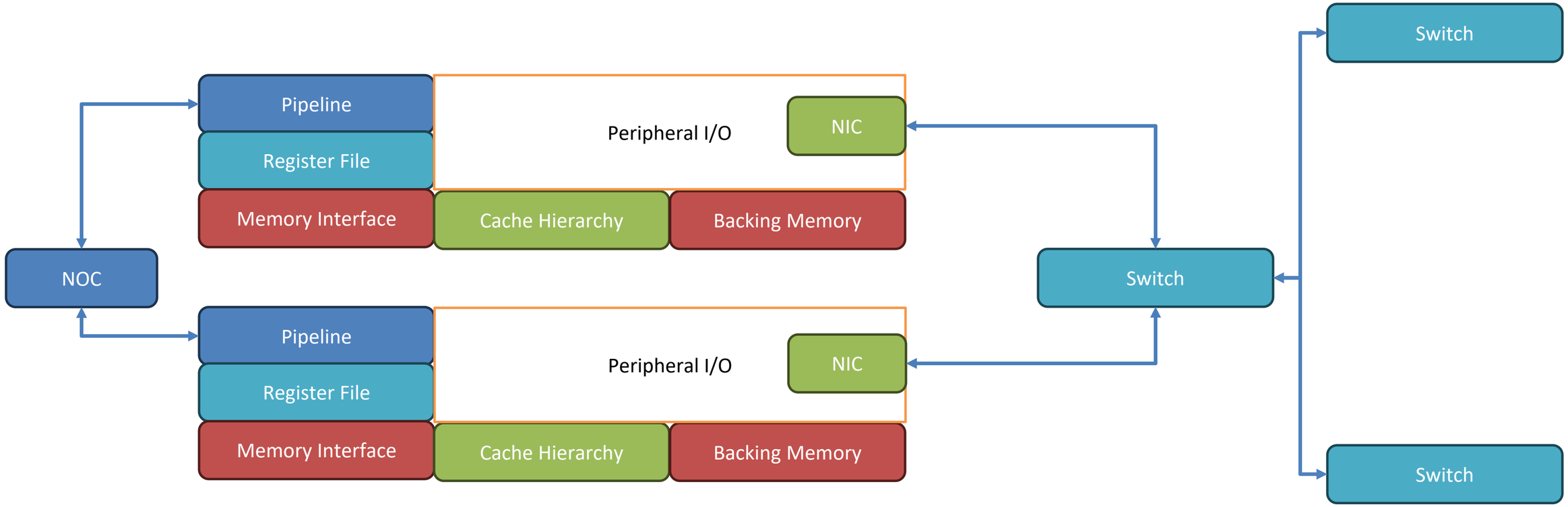
- Migrating from functional simulation to emulation is difficult!

- ***VerilatorSST*** component
  - Combines traditional SST component infrastructure with Verilator backends
  - Combines Cmake build infrastructure with standard Verilator APIs
  - Presents Verilog testbench I/Os as external ports
  - Permits users to replace existing, functional components with RTL simulation
- ***Currently in development!***

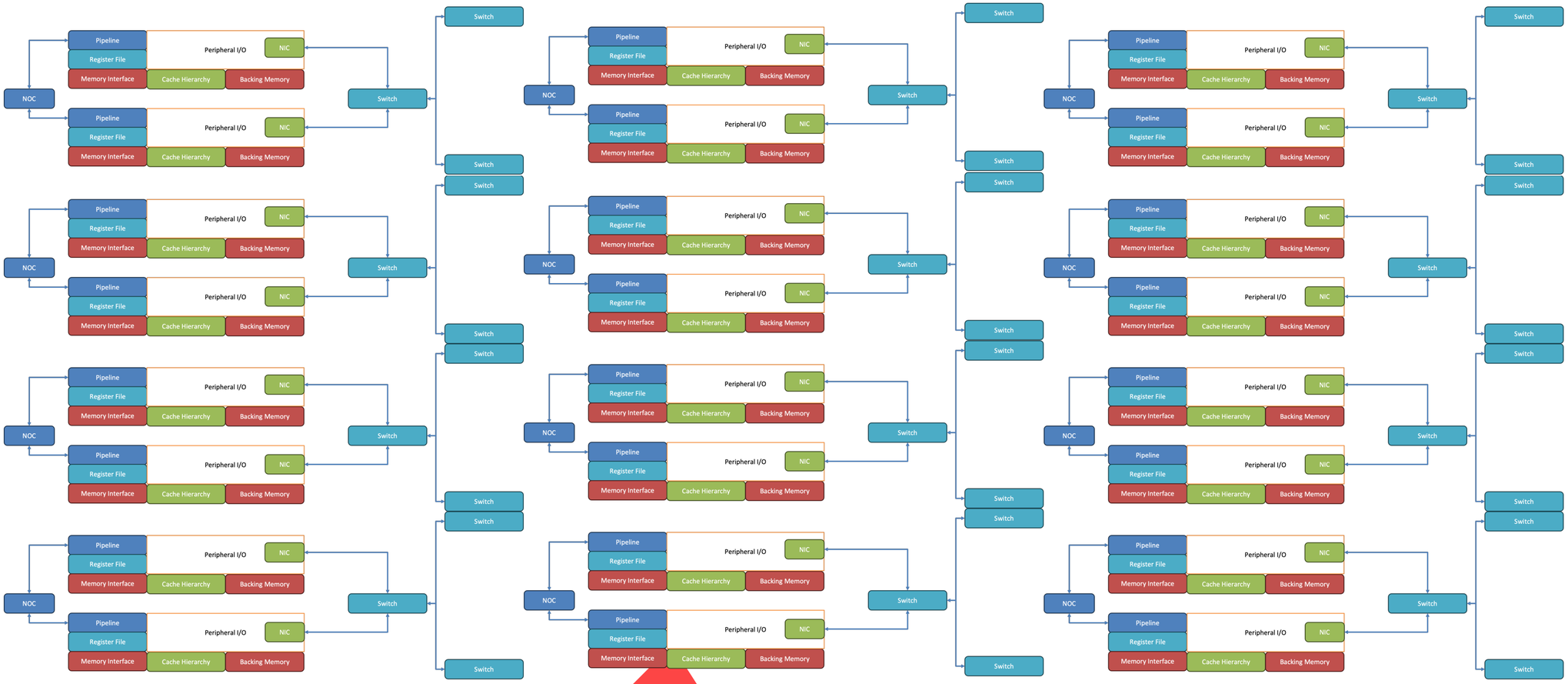


Future Experiments

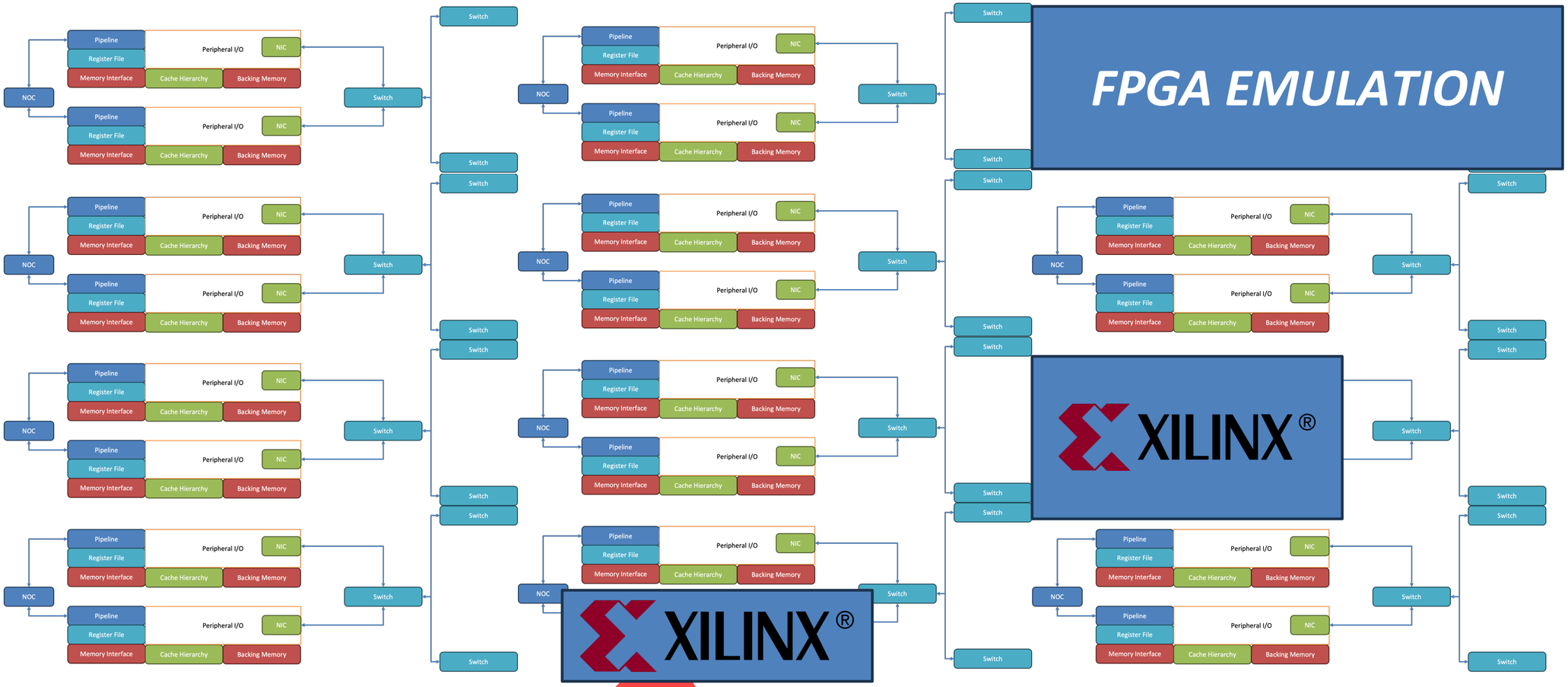
Full System Integration



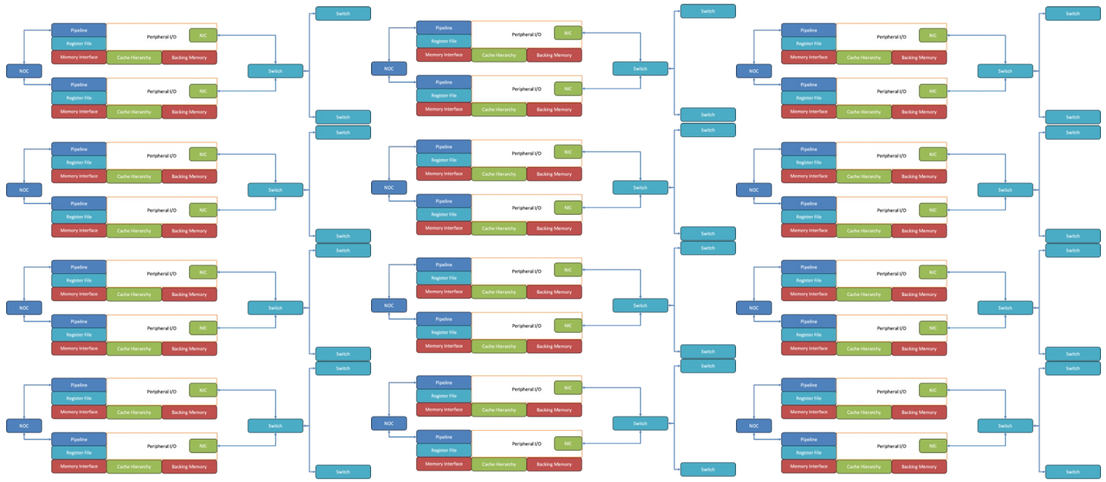
Full System Integration

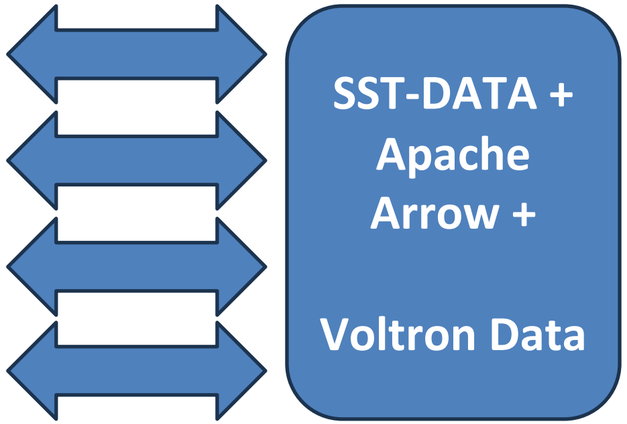


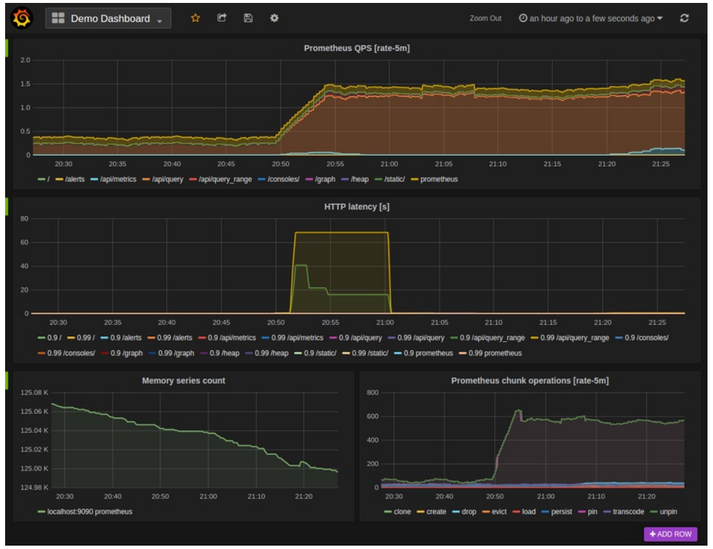
Full System Integration



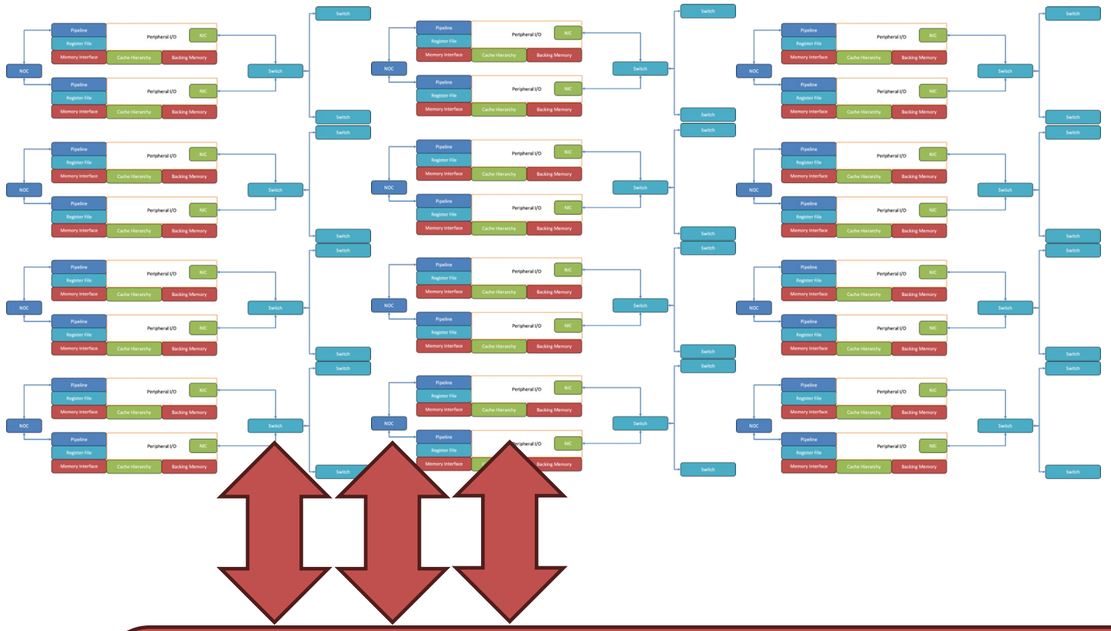
Advanced Simulation Analytics



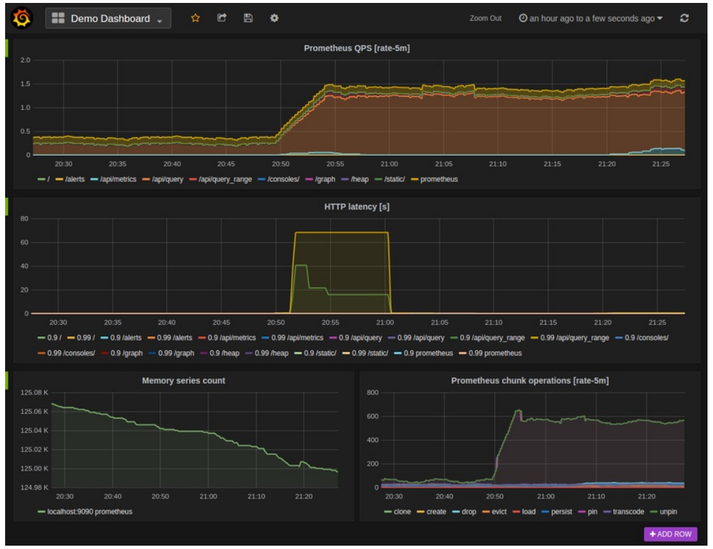




Predictive Modeling & Sim



SST-DATA + Apache Arrow + Voltron Data



Simulation Analytics


TCL