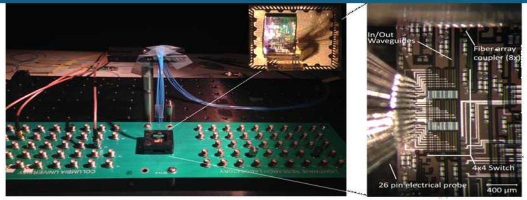
SAND2020-1642C

Supercomputer in a workstation: simulation as a development platform for network architectures



PRESENTED BY

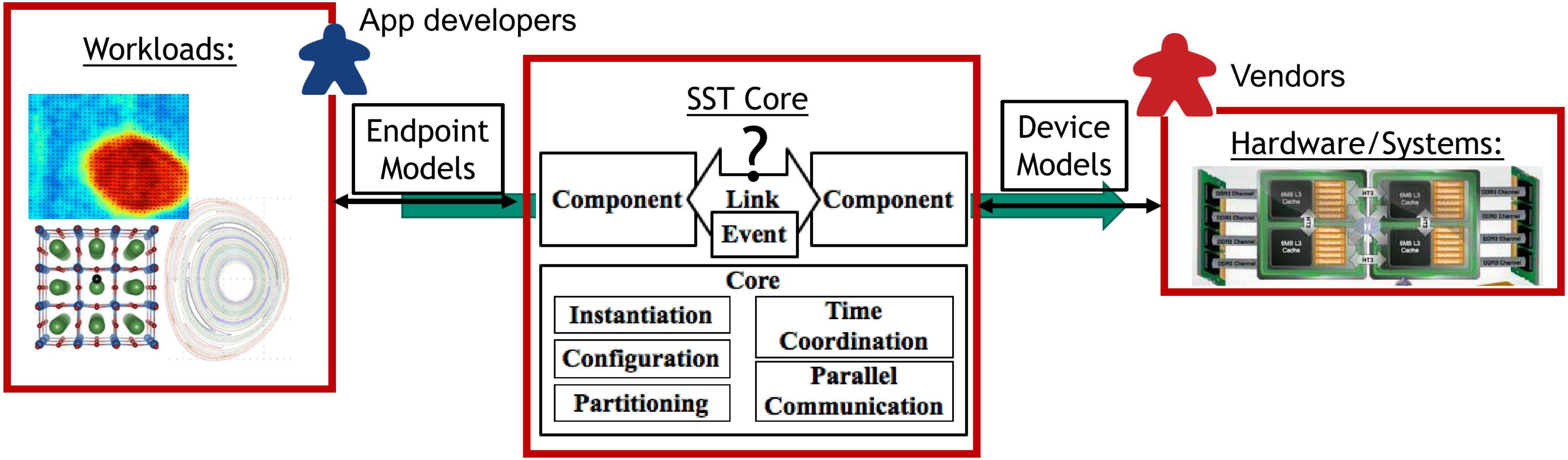
Jeremiah Wilke, Sandia National Labs, Livermore, CA

Collaborators: Joseph Kenny, Cannada Lewis, Samuel Knight

SIAM PP, Seatle, WA, 2020

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Applications and systems software need mechanism to convey requirements between application teams and system vendors



The Structural Simulation Toolkit provides analysis framework for answering these questions

The lab needs to work with vendors to advance new software and new hardware from idea to production

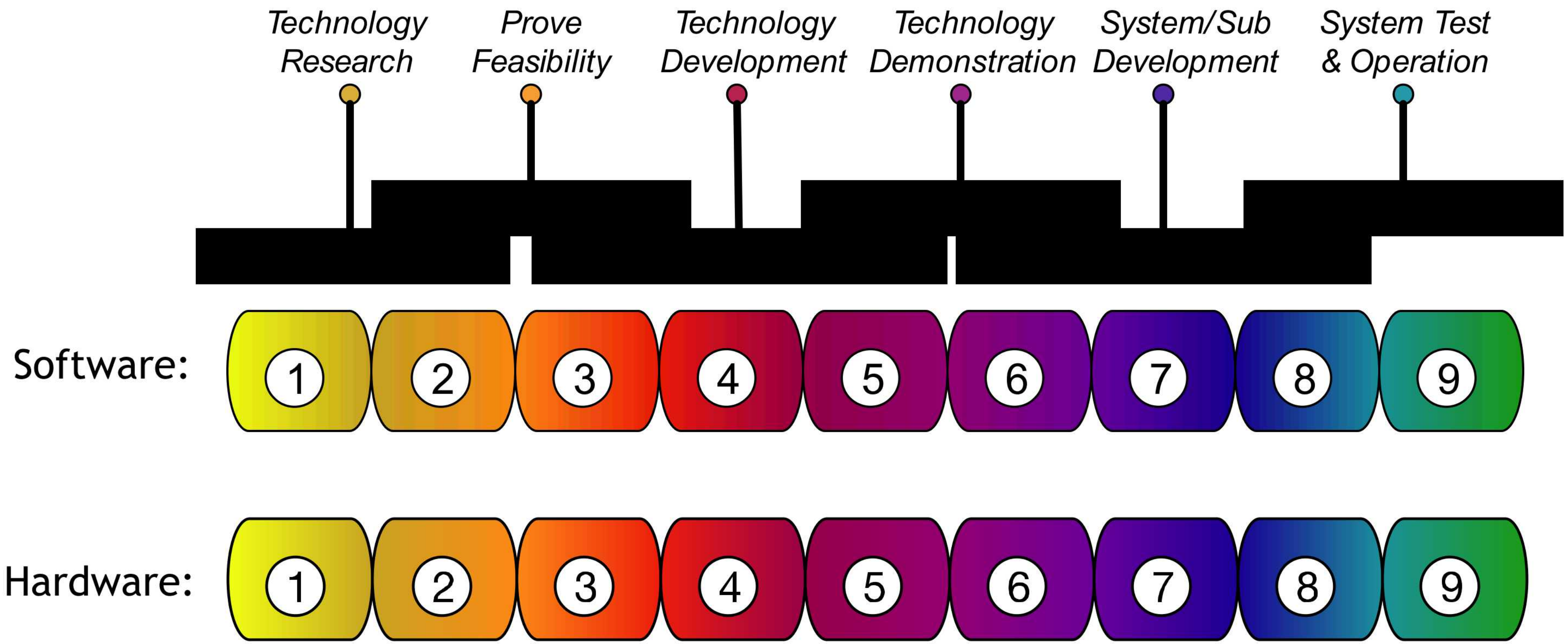
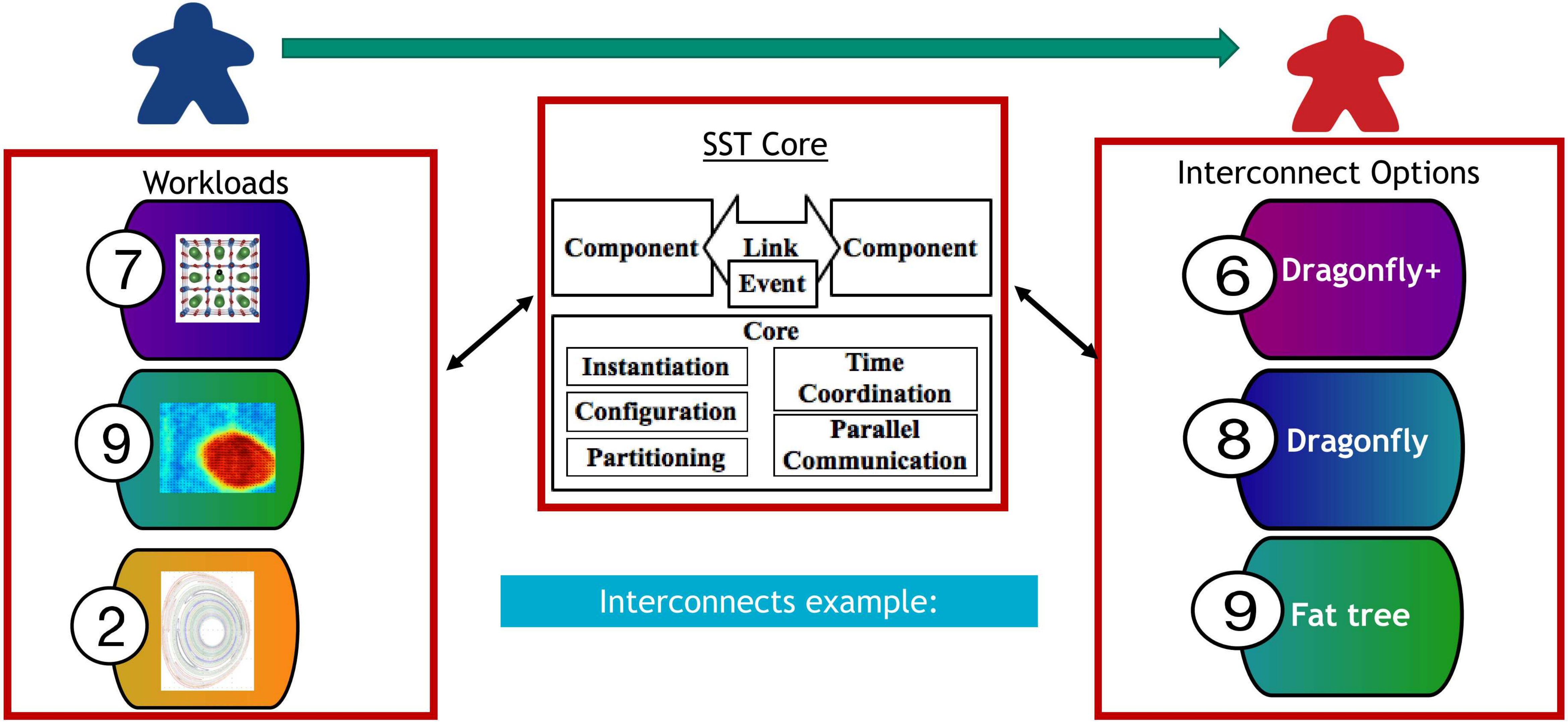


Figure: Technology readiness levels used by Sandia to categorize transition from idea to product

SST is an analysis tool for choosing best procurements or best architectures to focus software development on

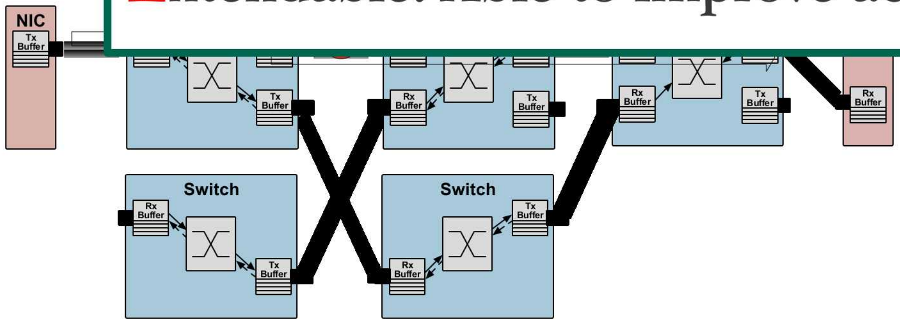


Conveying application requirements through simulation requires "endpoint model" that generates realistic traffic

Challenge is scale: Can I simulate a supercomputer without an even bigger supercomputer?

•Opti
• V
•Opti
• E
•Opti
• F
•Opti

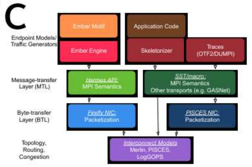
- **CO-design: Engagement with both app developers and network vendors**
- **Validation/Verification: Possible to demonstrate correctness on existing system**
- **Flexibility: Able to tune with different parameters**
- **Efficiency: Able to execute on limited compute resources**
- **Fruitful: Provides useful results, preferably more than one-off study**
- **Extendable: Able to improve accuracy and detail if needed**



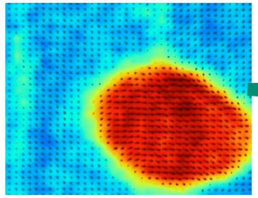
The "traffic pattern" on the network characterizes our unique requirements

Compiler tools can eliminate rate-limiting step in generating endpoint models for interconnect designs

Simulator-specific models



Auto-generate "skeleton" with compiler

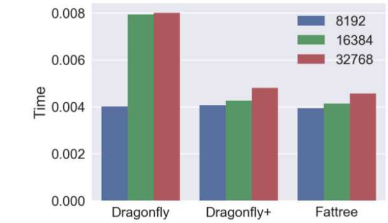


Run online model

Run skeleton app

On-line models directly running applications and real network stacks are more accurate, more useful

Output results

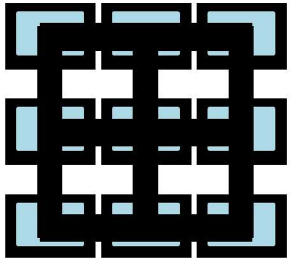


Run app

Trace replay

```
MPI_Send(…)
…
MPI_Allreduce()
```

Choose design



Output trace

```
MPI_Send(…)
Compute(x ms)
MPI_Allreduce(...)
Compute(y ms)
...
```

Related Work: Simulators, Performance Analysis Tools, and Network Runtimes

Related Project	Description	Where	
Score-P + OTF2	Profiling and tracing tools	Jülich (with DOE funding)	https://www.vi-hps.org/projects/score-p/
Tracer/CODES	Interconnect simulator largely based on traces	Argonne and Lawrence Livermore	https://github.com/LLNL/TraceR/
OMNet++	Parallel simulation framework popular with internet networks	Academic Community	http://omnetpp.org
SMPI/SimGrid	Simulation framework for running MPI apps	INRIA	https://github.com/simgrid/simgrid

SST/macro is unique in its ability to leverage compiler support, mixed fidelity models, and HPC focus

Designing exascale interconnects is a challenge across the entire software stack with many lab projects involved

These design questions often involve either hardware or software that doesn't exist yet!

Applications

1) Choose scalable algorithm (weak,strong)
2) Express communication pattern to network stack using API

Network Software Stack

1) Collective algorithms
2) Choose and implement protocols
3) Choose service levels
4) Provide API for applications
5) Place jobs on nodes

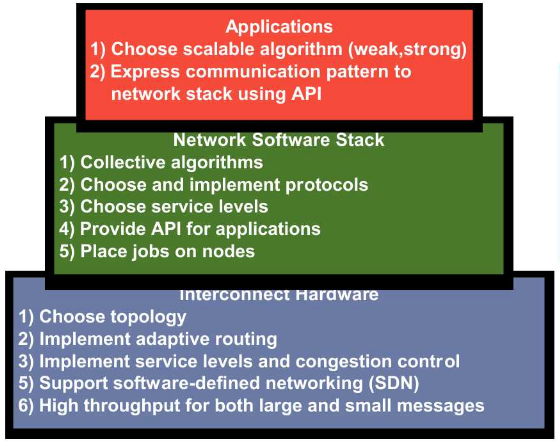
Interconnect Hardware

1) Choose topology
2) Implement adaptive routing
3) Implement service levels and congestion control
5) Support software-defined networking (SDN)
6) High throughput for both large and small messages

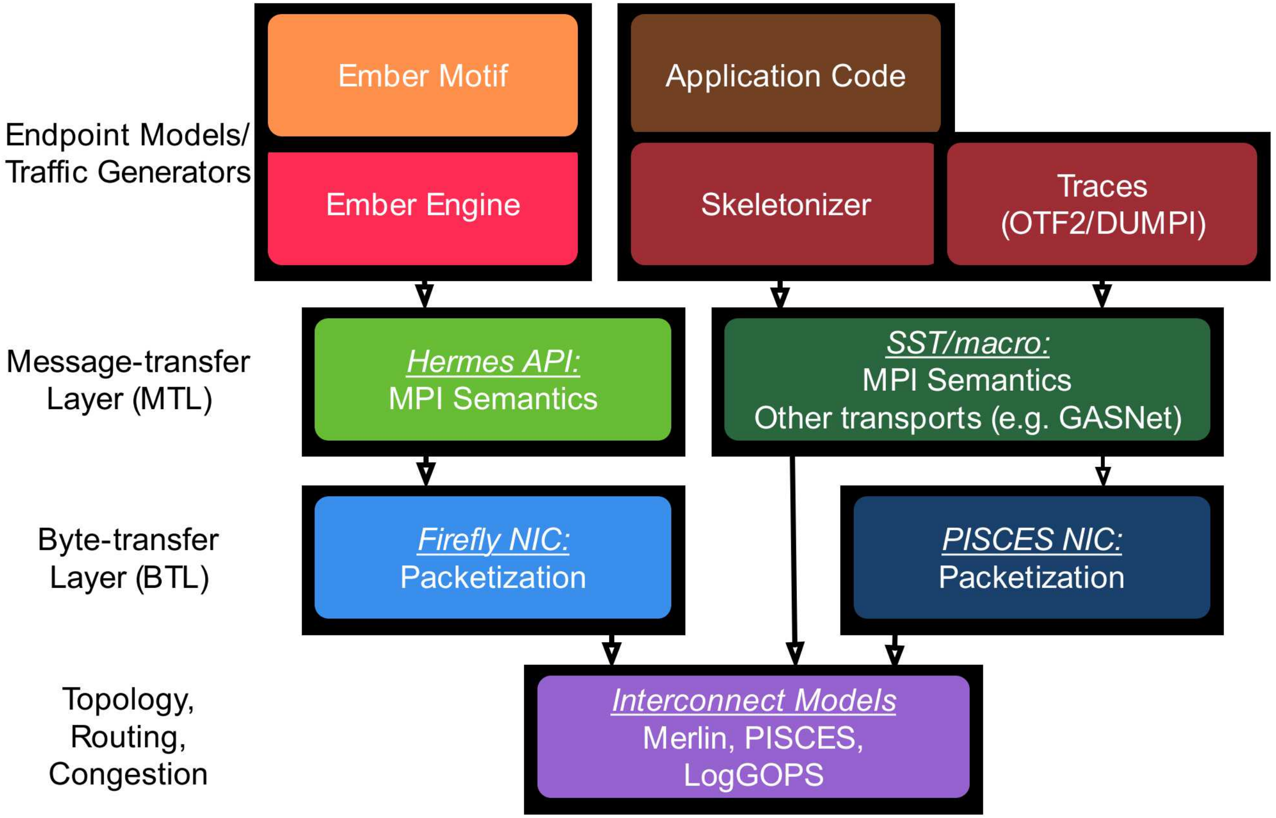
OpenMPI
MVAPICH
MPICH
GASNet
upc++
UCX
Open
OPENFABRICS ALLIANCE
portals

Figure: Some of the projects with DOE funding/collaborations affecting the network stack. Many others including Charm++, Legion, DARMA

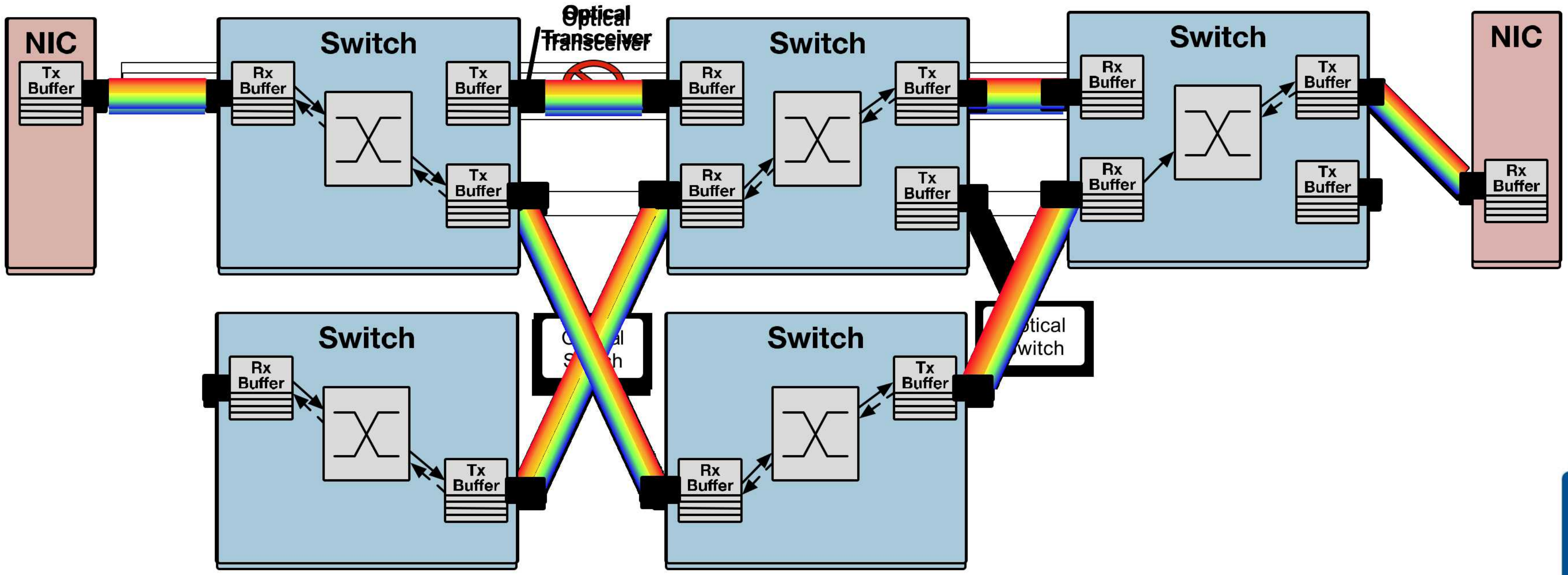
Theoretical studies difficult to extend into working products when only running *simulator-specific* communication libraries



Each design issue requires an implementation in SST



Illustrative example: Reconfigurable optical interconnects study shows how challenging technology transitions are

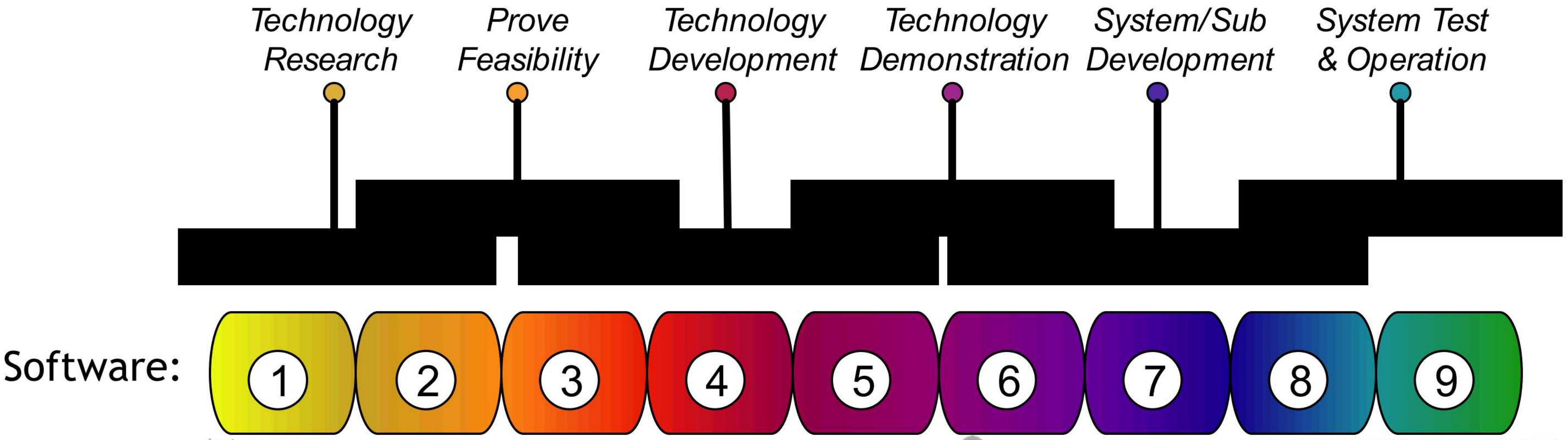


Results showed 2X speedup with *reduced* energy

Collaboration with Keren Bergman

Figure: Two traffic flows contend for bandwidth across electrical network
Figure: Electrical links replaced with optical links for higher bandwidth density
Figure: Reconfigurable switches *move* bandwidth to alleviate hotspots
Figure: Two traffic flows no longer contend for the same network path

Transitioning from an interesting idea in a simulator-specific model to a ready product is challenging



No hardware exists to advance TRL of software stack!

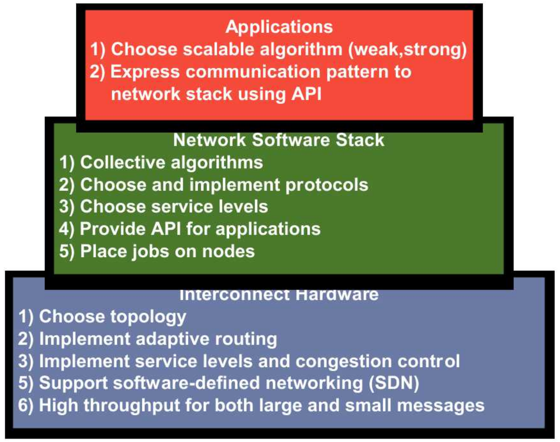
[redacted]

OpenFlow

Initial idea -> Demo with OpenFlow -> Add to MPI stack -> Deploy on system

Simulator produces ideas at TRL 1-3

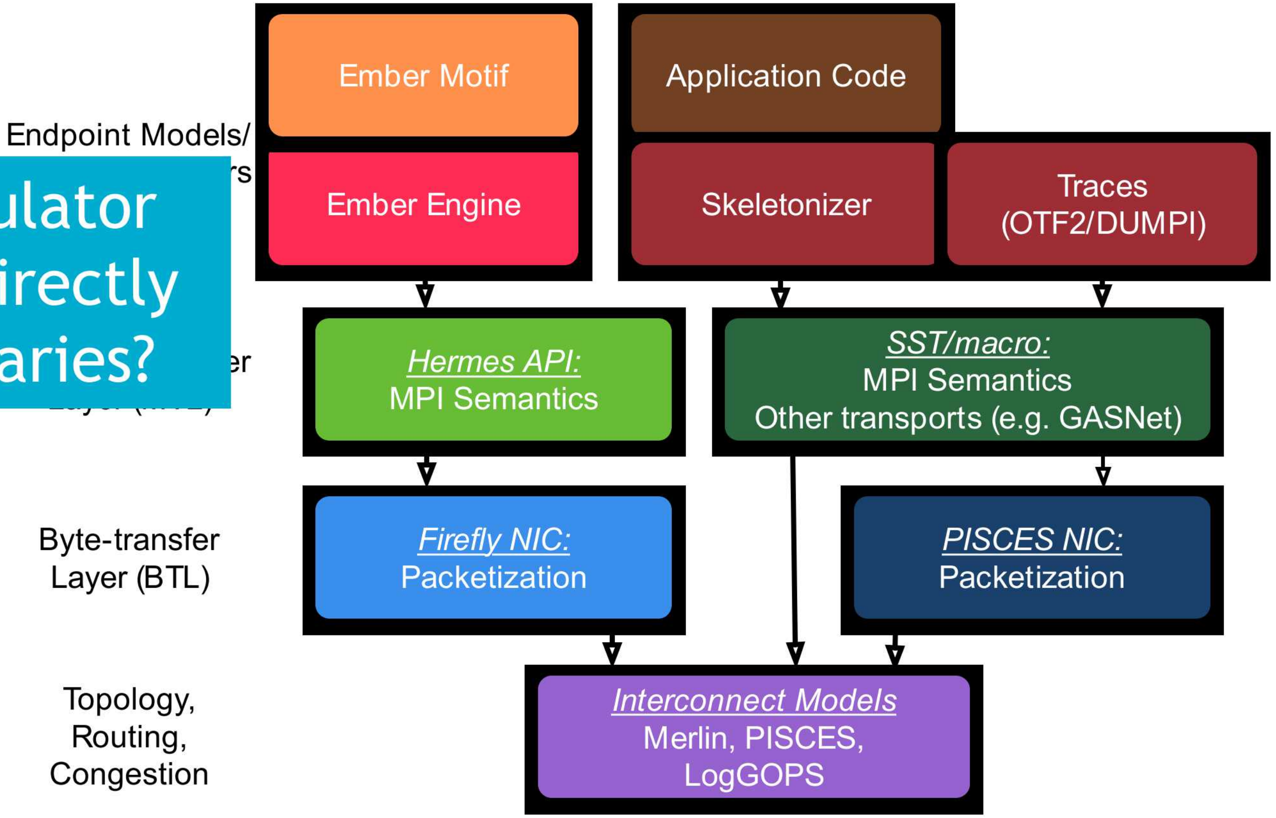
Theoretical studies difficult to extend into working products when only running *simulator-specific* communication libraries



OPENFABRICS ALLIANCE

MVAPICH

What if each simulator implementation directly used existing libraries?



Solving problem by directly simulating real application code requires overcoming the challenge of scale

Solution: Compiler support to automatically generate endpoint models by eliminating expensive memory/compute

Original Source Code:

```
double* big = new double[N];
MPI_Sendrecv(big,…);
for (i=0; i < N; ++i){
  expensive_compute();
}
MPI_Allreduce(…);
```

1) Developer adds pragmas

Modified Source Code:

```
#pragma sim null_variable
double* big = new double[N];
MPI_Sendrecv(big,…);
#pragma sim compute
for (i=0; i < N; ++i){
  expensive_compute();
}
MPI_Allreduce(…);
```

2) Clang source-to-source

Auto-skeletonized Source Code:

```
double* big = nullptr;
MPI_Sendrecv(big,…); //modeled
modelCompute(N,…);
MPI_Allreduce(…); //modeled
```

3) sim++ src.cpp
Redirect MPI calls

Auto-skeletonized Object Code:

```
call SIM_MPI_Sendrecv(….);
call modelCompute(N);
call SIM_MPI_Allreduce(…);
```

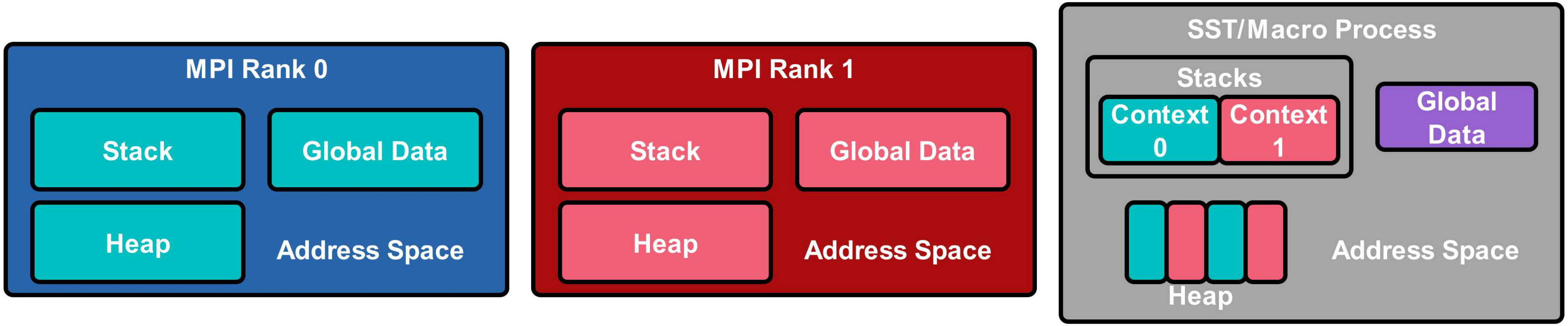
4) Link to simulator with SIM_MPI_X symbols
sim++ -o sim.x -lsim

Simulation Endpoint Model

64 GB memory
100 GF compute

Simulator needs to achieve both "encapsulation" and "skeletonization" to provide scalable simulation

- Simulator runtime must mimic memory separation of a distributed system
- Each virtual process needs a private:
  - **Stack** – User space-threads for scalable stack separation
  - **Heap** – Each individual heap allocation already "private"
  - **Globals** – Skeletonizer renames global variables to be accessible in a thread-local context
- Resulting simulation emulates concurrent execution of many *virtual* processes in one *physical* simulator processes (or a few simulator processes for parallel discrete event simulation – PDES)



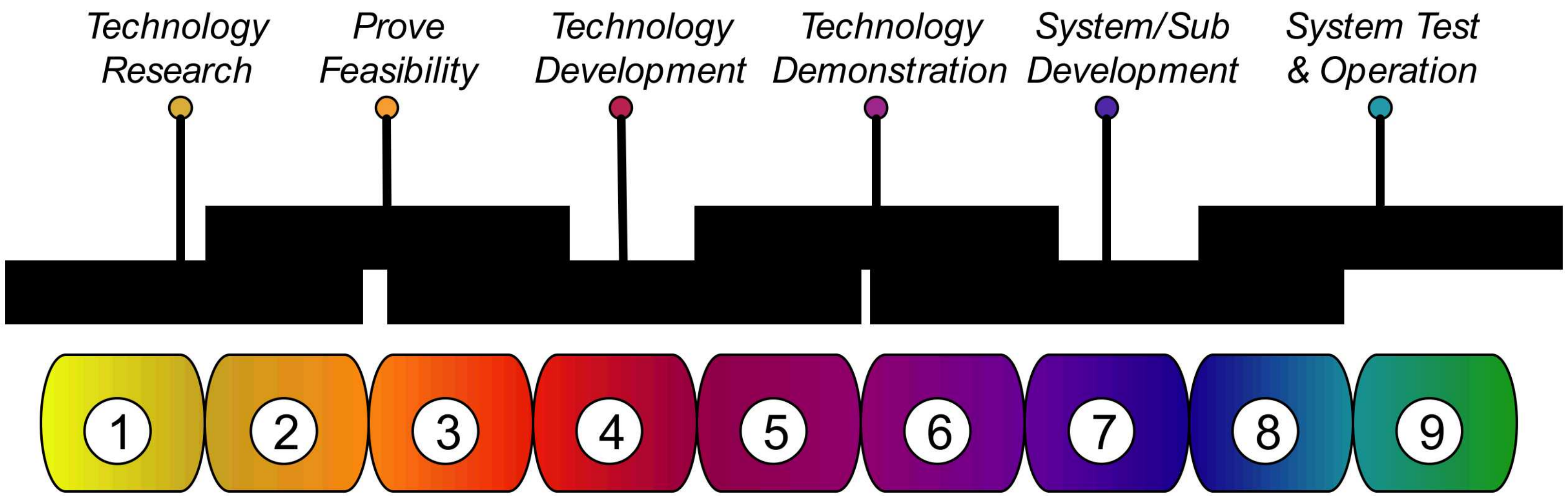
High-fidelity simulation is possible for exascale network, but not for the entire exascale system

	High-Fidelity Sim of 1s (100x Overhead)		Exascale System		Coarse-Grained Sim of 1s (100x Cost Reduction)	
	Compute	Memory	Compute	Memory	Compute	Memory
Nodes	100 ExaOPs	25 PB	1 ExaOP/s	5 PB	5 TeraOPs	40 GB
Network Interface	1 PetaOPs	5 TB	400 GigaOP/s	500 GB	1 TeraOPs	5 GB
Switches	5 PetaOPs	100 GB	50 TeraOP/s	25 GB	5 TeraOPs	20 GB

Using the supercomputers of today to design the supercomputers of tomorrow

A coarse-grained simulation is feasible on a powerful workstation.
A mixed-fidelity (detailed network, coarse-grained nodes) is feasible with an existing supercomputer!

Shorten time to production-ready by eliminating rate-limiting step: don’t need access to non-existent supercomputer



Compiler support allows simulator to advance TRL far beyond just interesting ideas

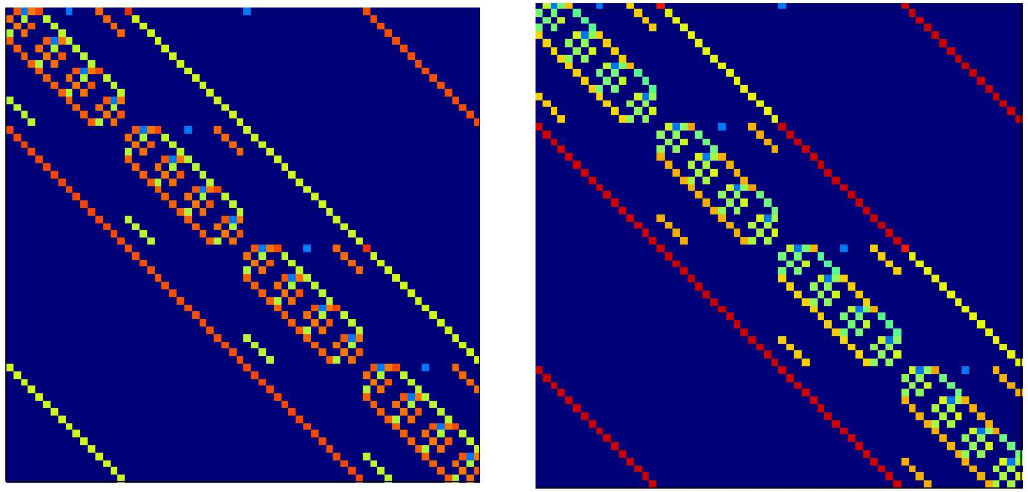
$$$$$
Every day hardware sits underutilized because software stack isn’t ready costs money

OpenFlow

Every day hardware or software development is delayed sacrifices lab computing leadership

Auto-skeletonization via compiler overcomes scaling challenges by reproducing behavior without expensive compute

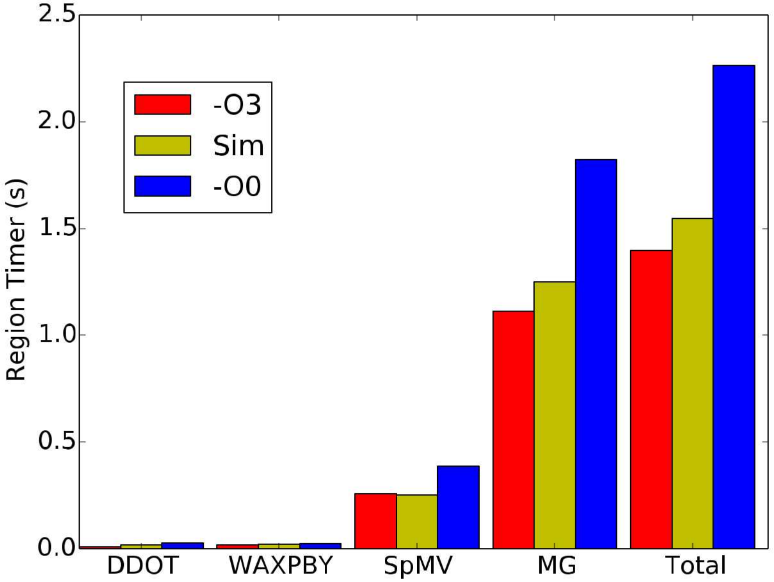
CoMD traffic patterns



Skeleton

Actual

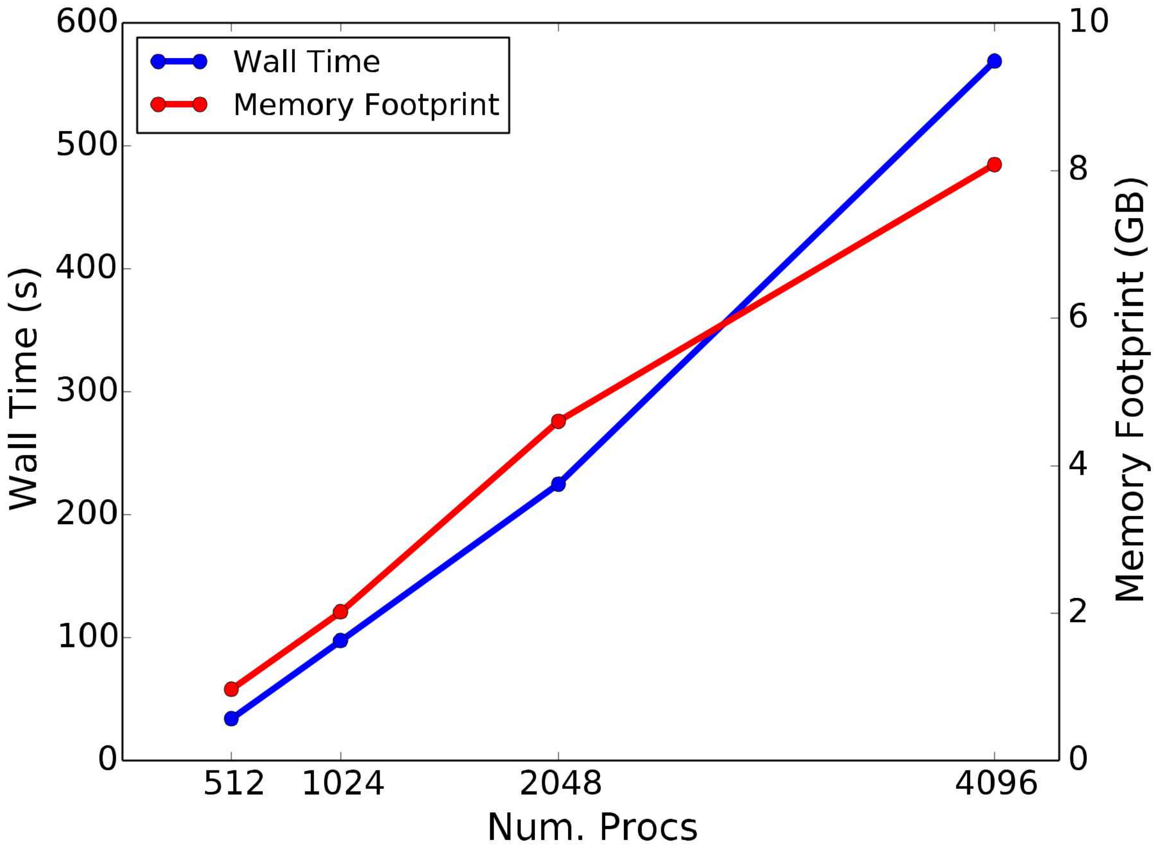
HPCG Compute Times



Despite approximations, traffic pattern and compute times are reasonably reproduced

Auto-skeletonization via compiler overcomes scaling challenges by reproducing behavior without expensive compute

Figure: Memory and compute of GASNet library in simulator



Application with GASNet runtime running directly in simulator, but injects traffic into *simulated network*

Running non-skeletonized version would be TBs memory!

Move beyond basic source-level models to more accurate and more flexible computational models: Machine Learning

```
int ComputeSPMV_ref( const SparseMatrix & A, Vector & x, Vector & y) {

  assert(x.localLength>=A.localNumberOfColumns); // Test vector lengths
  assert(y.localLength>=A.localNumberOfRows);

#ifndef HPCG_NO_MPI
    ExchangeHalo(A,x);
#endif
  const double * const xv = x.values;
  double * const yv = y.values;
  const local_int_t nrow = A.localNumberOfRows;
#ifndef HPCG_NO_OPENMP
  #pragma omp parallel for
#endif
  for (local_int_t i=0; i< nrow; i++)  {
    double sum = 0.0;
    const double * const cur_vals = A.matrixValues[i];
    const local_int_t * const cur_inds = A.mtxIndL[i];
```

Capture nrow as kernel metadata

Automatically detect OpenMP regions and instrument for fitting models

Move beyond basic source-level models to more accurate and more flexible computational models: Machine Learning

```
const local_int_t nrow = A.localNumberOfRows;

f0_ComputeSPMV_ref_pp_ComputeSPMV_ref_cpp61_memoize_start(nrow)
#pragma omp parallel for
for (local_int_t i = 0; i < nrow; i++) {
    ble sum = 0.0;
    st double *const cur_vals = A.matrixValues[i];
    st local_int_t *const cur_inds = A.mtxIndL[i];
    st int cur_nnz = A.nonzerosInRow[i];

    (int j = 0; j < cur_nnz; j++)
    um += cur_vals[j] * xv[cur_inds[j]];
    yv[i] = sum;
}
f0_ComputeSPMV_ref_pp_ComputeSPMV_ref_cpp61_memoize_end();
```

Added instrumentation with automatic capture of nrow.

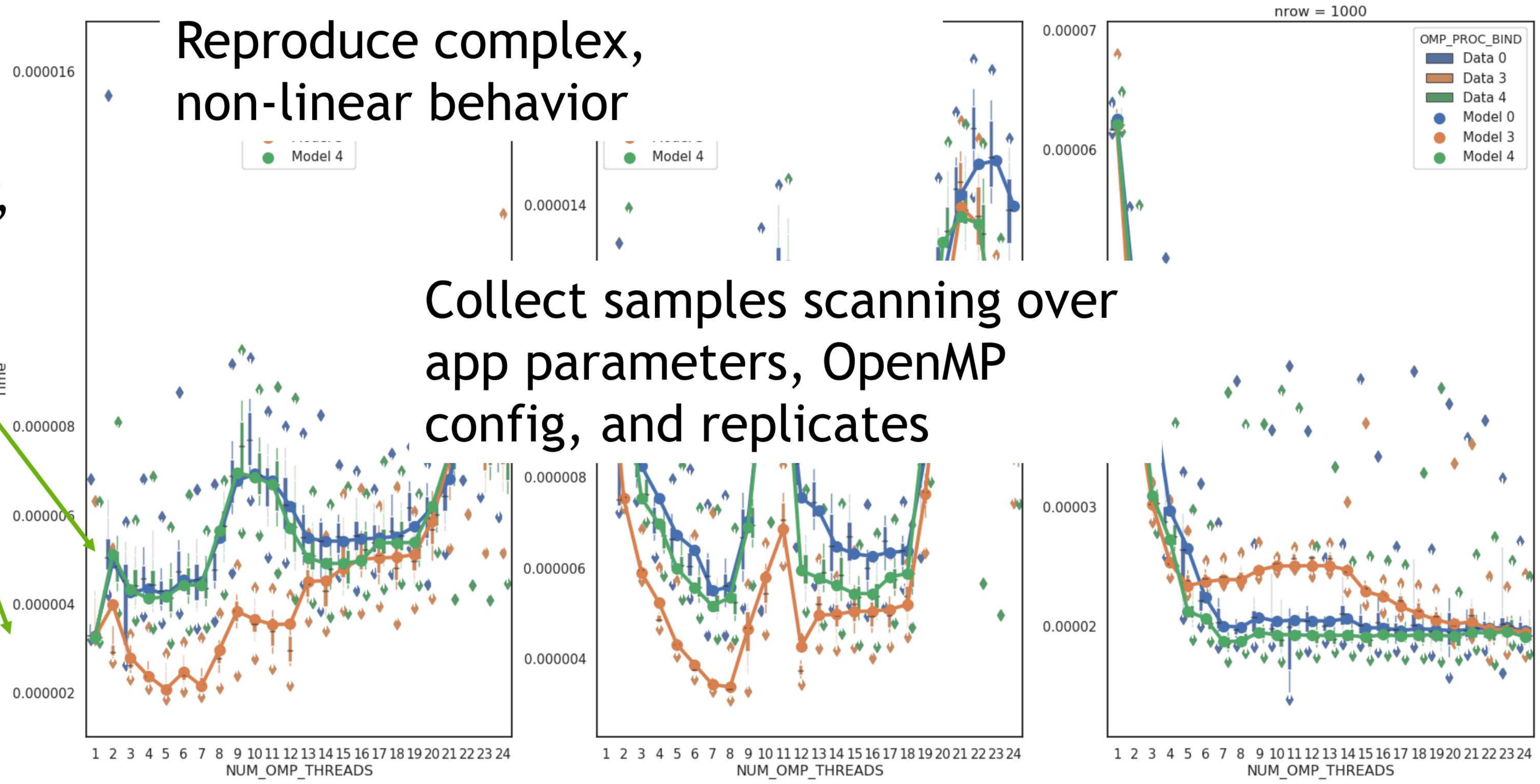
Also captured inside the backend are NUM_OMP_THREADS, OMP_PROC_BIND and OMP_PLACES

Move beyond basic source-level models to more accurate and more flexible computational models: Gradient-Boosted Trees

Median, upper, lower bounds generated

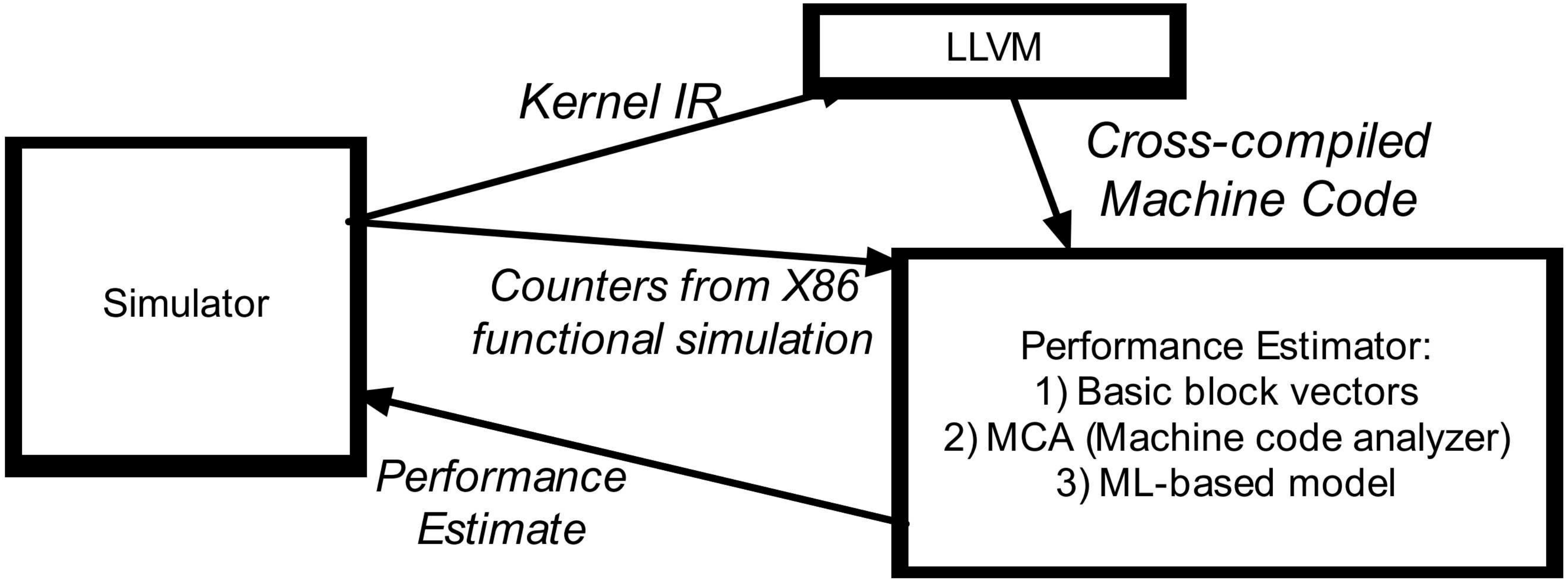
Reproduce complex, non-linear behavior

Collect samples scanning over app parameters, OpenMP config, and replicates



Move beyond instrumentation-based models and provide models for configurable architectures: LLVM + ML

- Don't rely on existing system for benchmarking – estimate performance for *new* architectures
- We still want *fast, functional* simulation on X86, e.g. – but collect enough performance counters to estimate performance on different architecture
- Proposal: Embed LLVM IR in simulator executable



Acknowledgments

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.