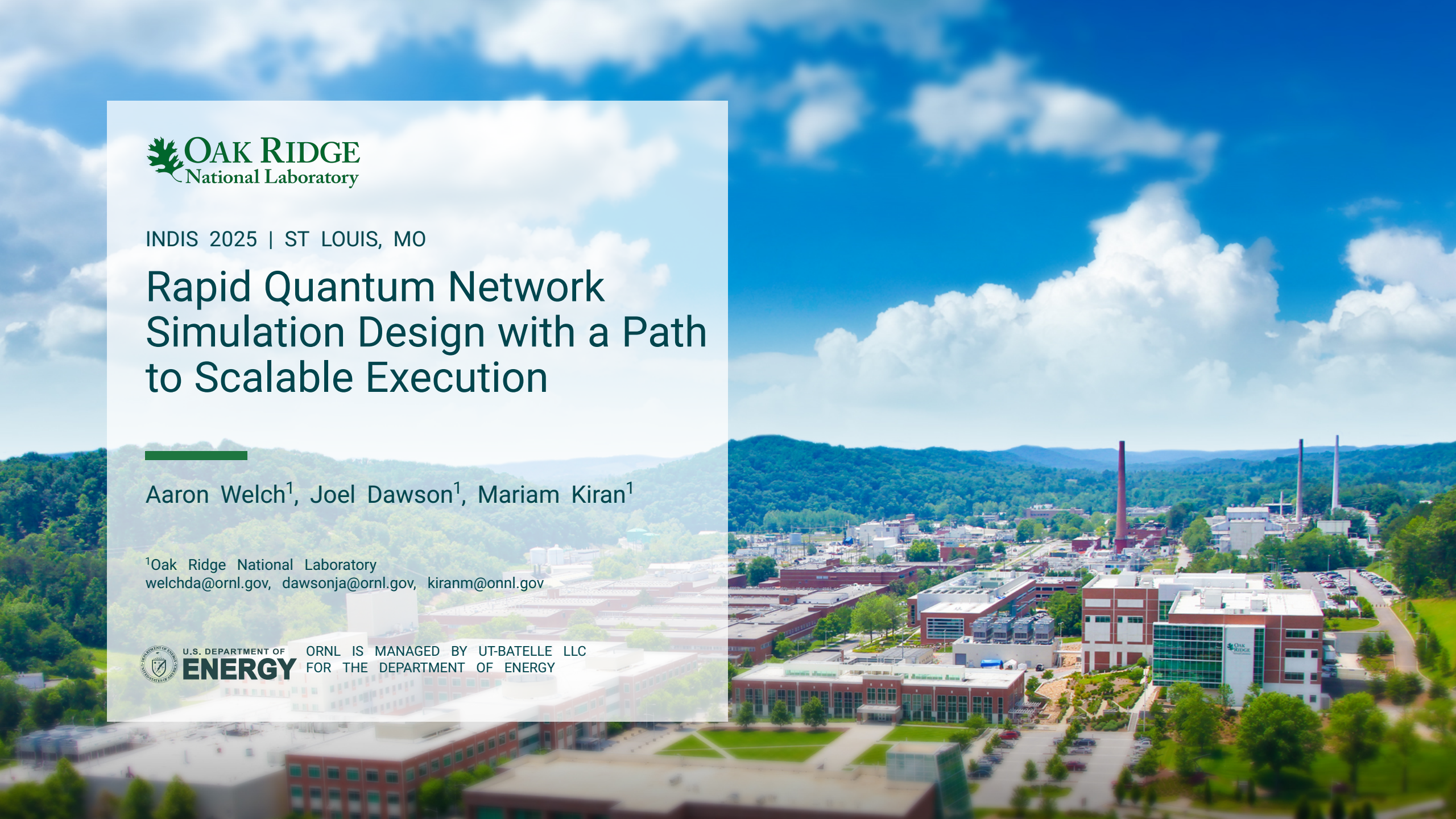


OAK RIDGE National Laboratory

INDIS 2025 | ST LOUIS, MO

Rapid Quantum Network Simulation Design with a Path to Scalable Execution

Aaron Welch[1], Joel Dawson[1], Mariam Kiran[1]

[1]Oak Ridge National Laboratory
welchda@ornl.gov, dawsonja@ornl.gov, kiranm@onnl.gov

U.S. DEPARTMENT OF ENERGY

ORNL IS MANAGED BY UT-BATELLE LLC FOR THE DEPARTMENT OF ENERGY

Too Long, Didn’t Listen

- Quantum networking is highly valuable, but simulations of it remain difficult/expensive
- To make simulations easier to design and iterate upon, we introduce a simulation framework allowing experiments to be designed in a dataflow fashion using abstract and reusable blocks
- We study the potential for performance improvement of parallel execution using another existing framework
- We demonstrate how simulations designed in the former could be ported to the latter for running at larger scales

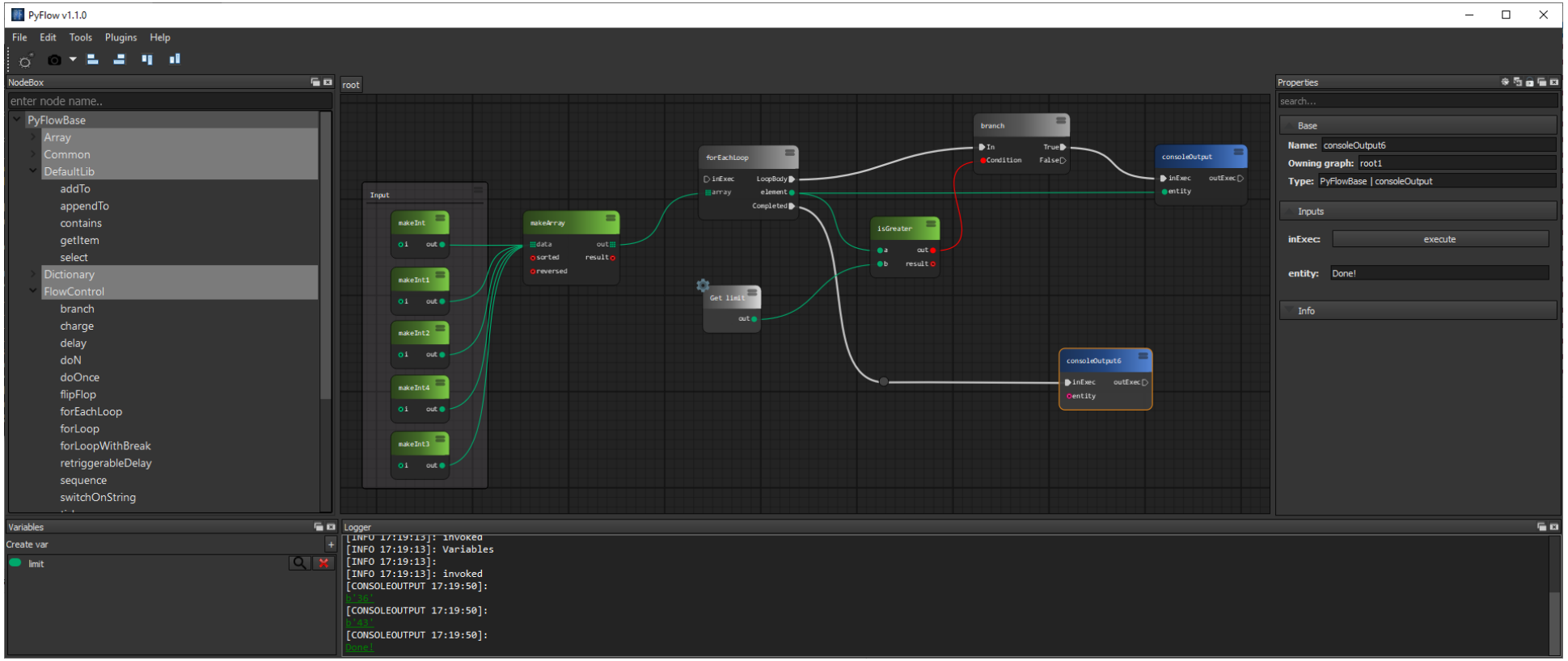
BISQIT

- Block-diagram Integrated Simulation framework for Quantum Information Technologies (BISQIT), or Baked-In Security for Quantum Information Technologies (BISQIT)
- Composed primarily of three components:
  - Quantum-Classical Interface Component (QCIC) — the individual components of the simulated system
  - Relations — the links between the different components
  - QCSim — combines components and their relations into a simulation experiment
- Uses a publisher-subscriber model to facilitate data transfer

BISQIT

- Relations (edges) conceptually denote a logical or mathematical dependency between two or more members of one or more sets
  - Can operate in parallel
- QCIC “blocks” abstract the transduction of data between classical and quantum forms
  - Likely maps to discrete hardware components in many cases
  - May contain a series of other QCIC blocks and relations within it
- QCSim represents a particular graph of components and models the simulation parametres and data flowing through it

Design Philosophy



From the PyFlow GitHub, used here for illustration purposes only

SeQUeNCe

- SeQUeNCe is a traditionally serial discrete event simulation framework designed in Python for high precision
- Comprised of six components:
  - Simulation kernel
  - Hardware module
  - Entanglement management module
  - Network management module
  - Resource management module
  - Application module
- Assumes no loss and perfect reliability for the classical channels
  - Accuracy becomes uncertain for saturated networks or if failures occur
- Past efforts sought to upgrade SeQUeNCe with parallel execution capabilities

Parallelisation of SeQUeNCe

- Prior work uncovered five key observations pertaining to parallelisation potential:
  - Events on quantum channels are dominant based on both quantity and simulation execution time.
  - The execution time for these events is highly consistent.
  - Events between pairs of quantum key distribution (QKD) terminals are evenly distributed over the simulated timeline.
  - The latency of quantum channel transmission is dominated by its propagation delay, which is lower than that of an equivalent classical channel.
  - Different QKD sessions are largely independent.
- Simulated routers were selected as the unit of parallelisation
  - The simulation timeline gets split into *n* parallel timelines, subdivided into synchronisation epochs
  - Within epochs, processes synchronise/exchange events, compute the end of the next epoch, then process local events
- The quantum state manager (QSM) gets broken up into a hierarchy of *n* local QSMs and one global QSM

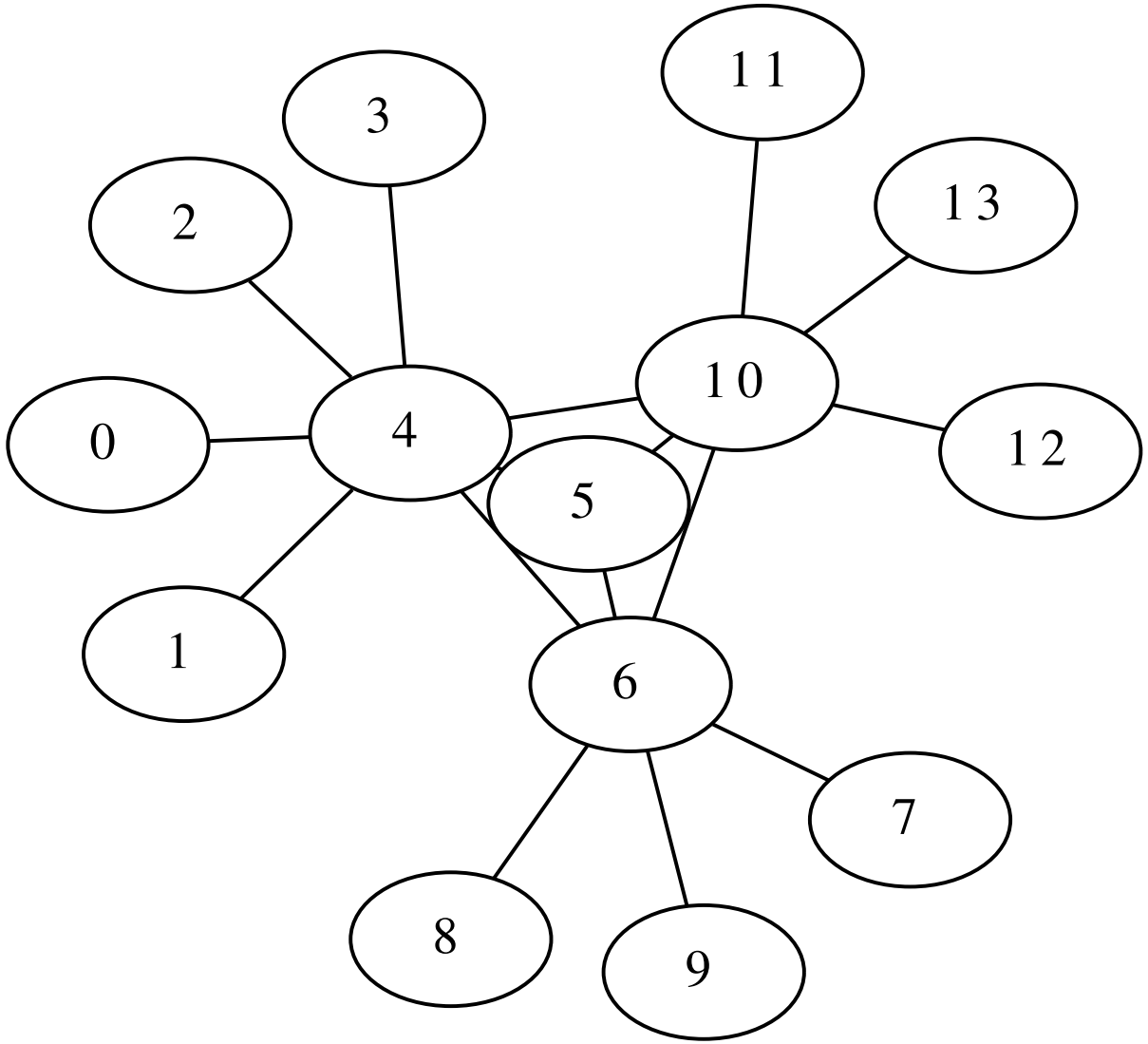
Adapting BISQIT Simulations

- While it could be a bit less straightforward to go the other way around, adapting a BISQIT simulation to SeQUeNCe can be done rather simply
- QCIC blocks can be mapped to the `Entity` class
  - Management of resource acquisition may need to be modified to go through SeQUeNCe's resource management module
- Relations between components can be mapped directly to events in SeQUeNCe
- The application layer doesn't have as direct of a transition, but can be rewritten from the contents of BISQIT's QCSim component

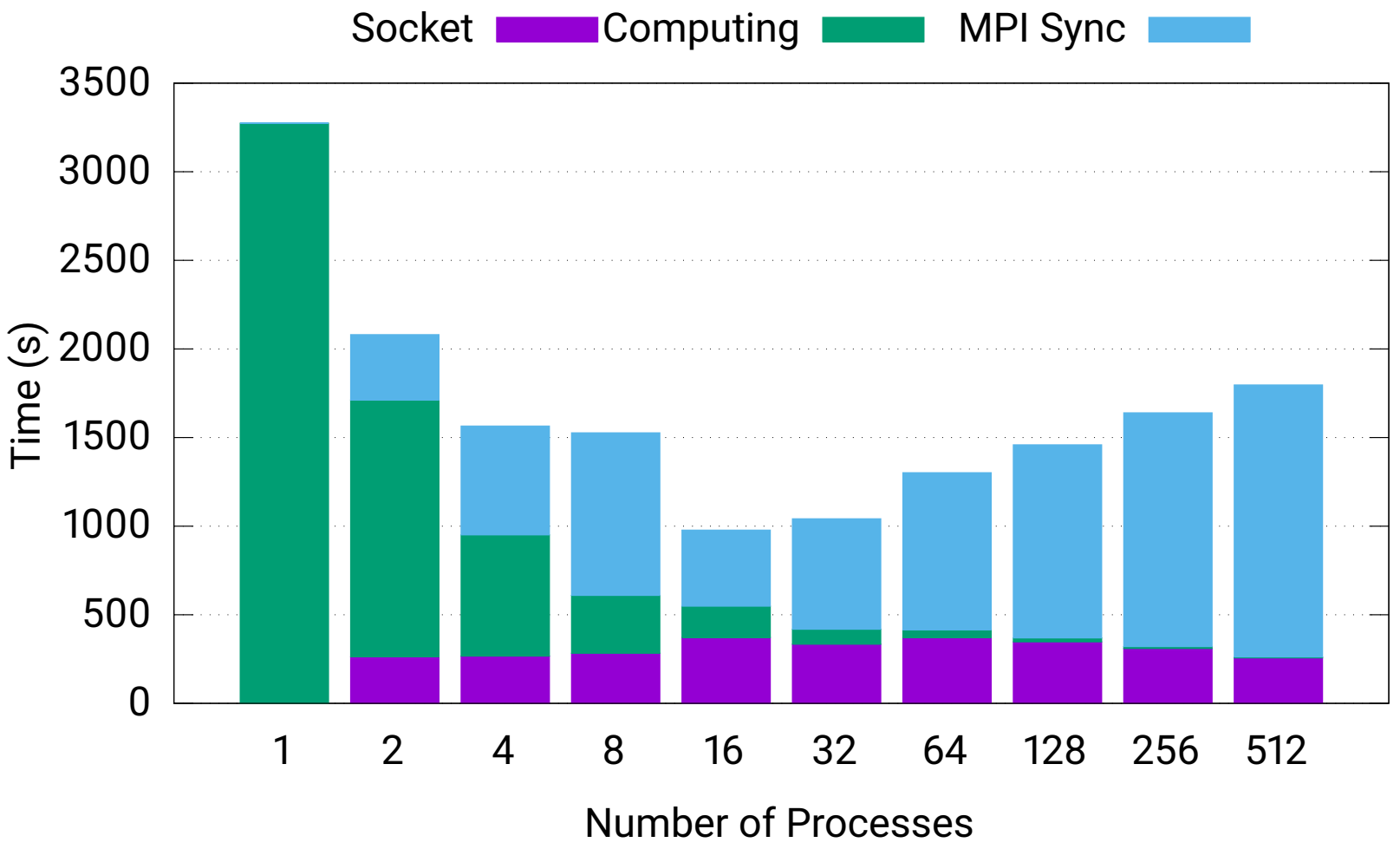
SeQUeNCe Performance Study

- To evaluate parallel performance, we reproduced two QKD experiments from SeQUeNCe's past efforts
  - These used either linear or autonomous system network topologies, though we will only focus on the latter here
- We simulated 1024 routers distributed across up to 512 processes on up to 8 nodes of ORNL's Frontier, plus an additional node for the global QSM
- Each compute node has:
  - A single 64-core AMD Epyc 7A53 2 GHz CPU
  - 512 GB of memory
  - Four HPE Slingshot 11 200 Gbit/s Cassini NICs connected to four AMD MI250X GPUs
- We used Python 3.10 for the run-time environment.

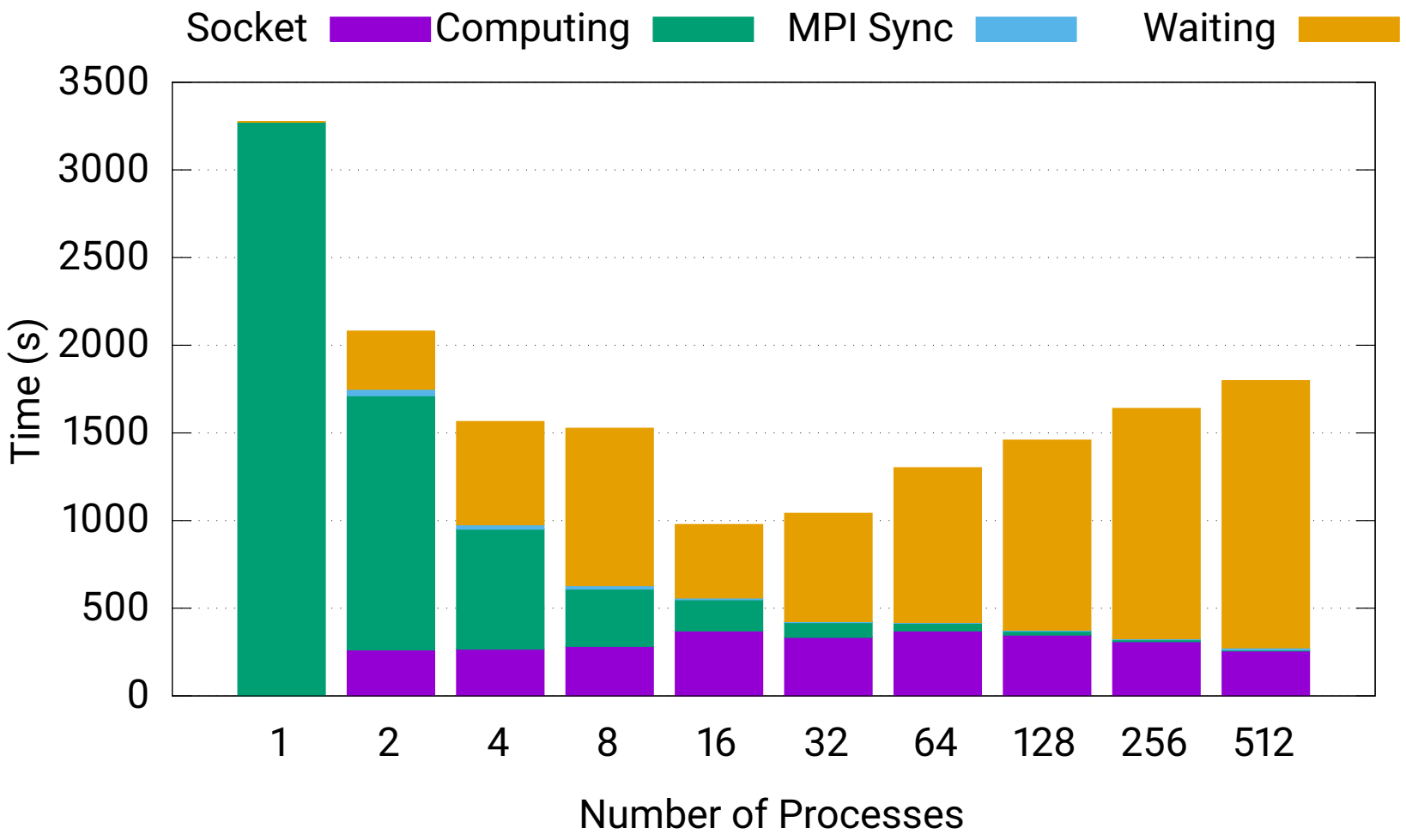
Autonomous System Network Topology



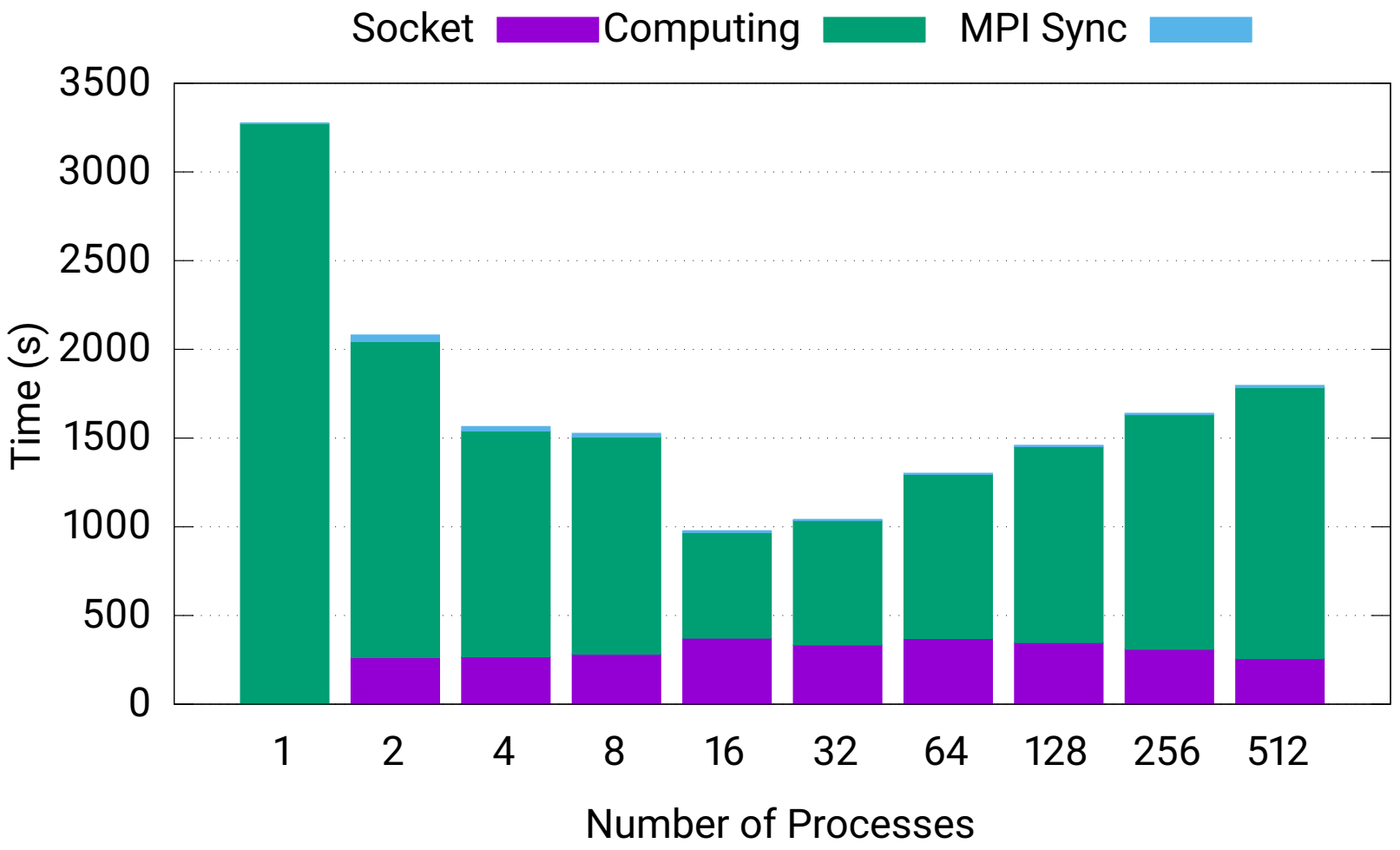
Performance



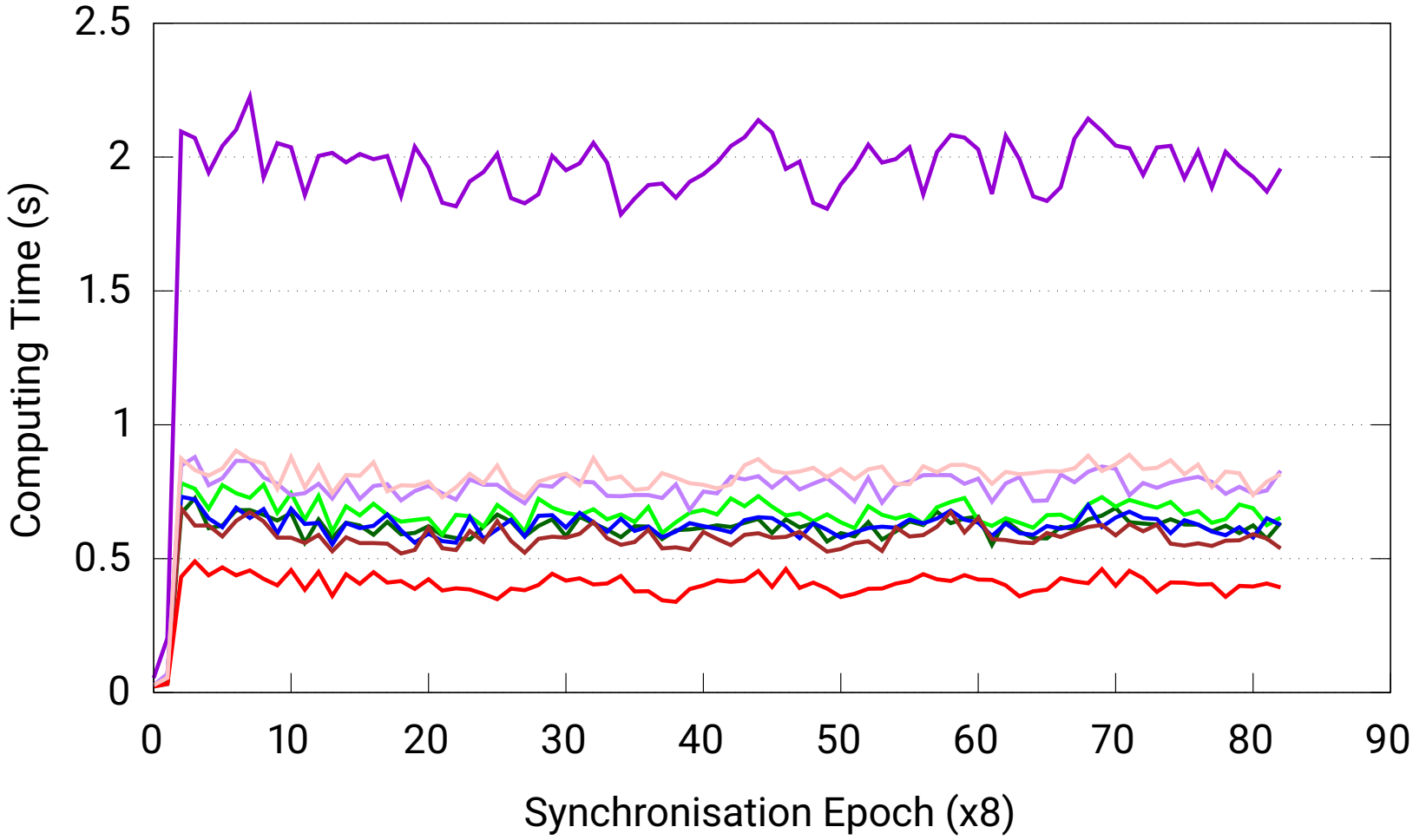
Performance Breakdown



Performance, Redefined



System Computing Time Per Process



Conclusions

- The intuitive design of BISQIT makes prototyping of new simulation designs fast and easy, but:
  - It does not provide as clean of a way to represent execution timelines
  - Is not currently well suited for large simulation runs
  - Work on an interface to PyFlow could go even further toward simplifying and democratising simulation design
- Porting simulations initially designed with BISQIT to SeQUeNCe for effective testing is easy and straightforward
- SeQUeNCe proved to be able to provide more efficient/scalable execution through parallelisation, but the scaling potential is limited by the workload and resulting work imbalances

Questions?