Colmena:
Steering Ensemble Simulations at ExaScales

EXASCALE COMPUTING PROJECT

Logan Ward
Assistant Computational Scientist
Data Science and Learning Division
Argonne National Laboratory

ParslFest

6 October 2020

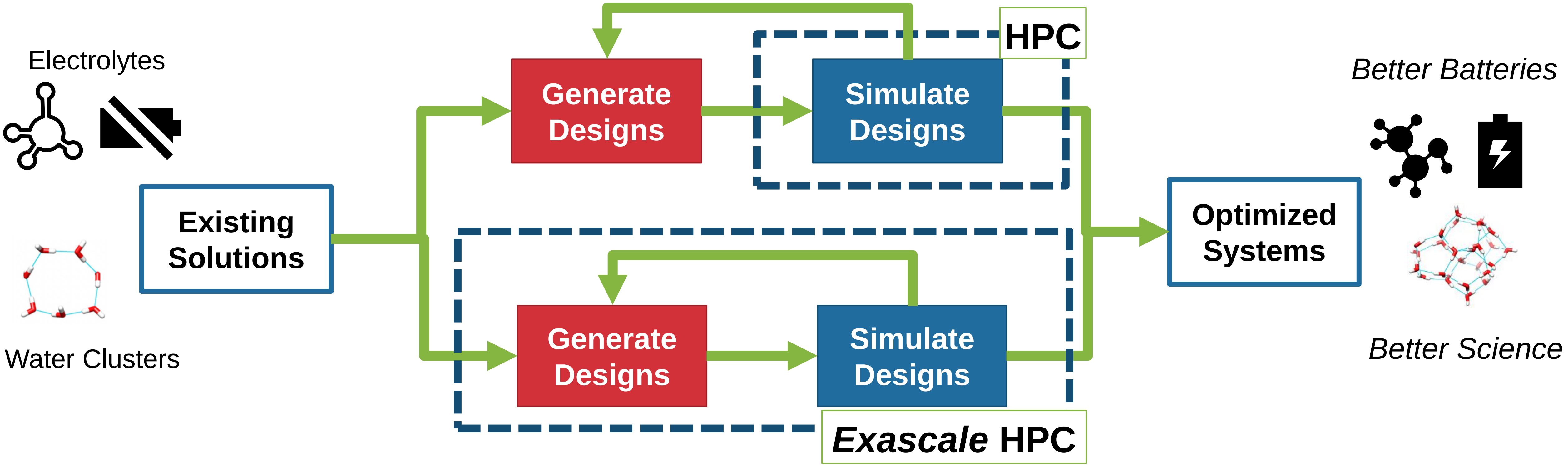
National Nuclear Security Administration

U.S. DEPARTMENT OF ENERGY | Office of Science

Expanding Computational Design to the ExaScale

Current Model: Humans steer HPC, HPC performs simulations ***(Months-Years)***

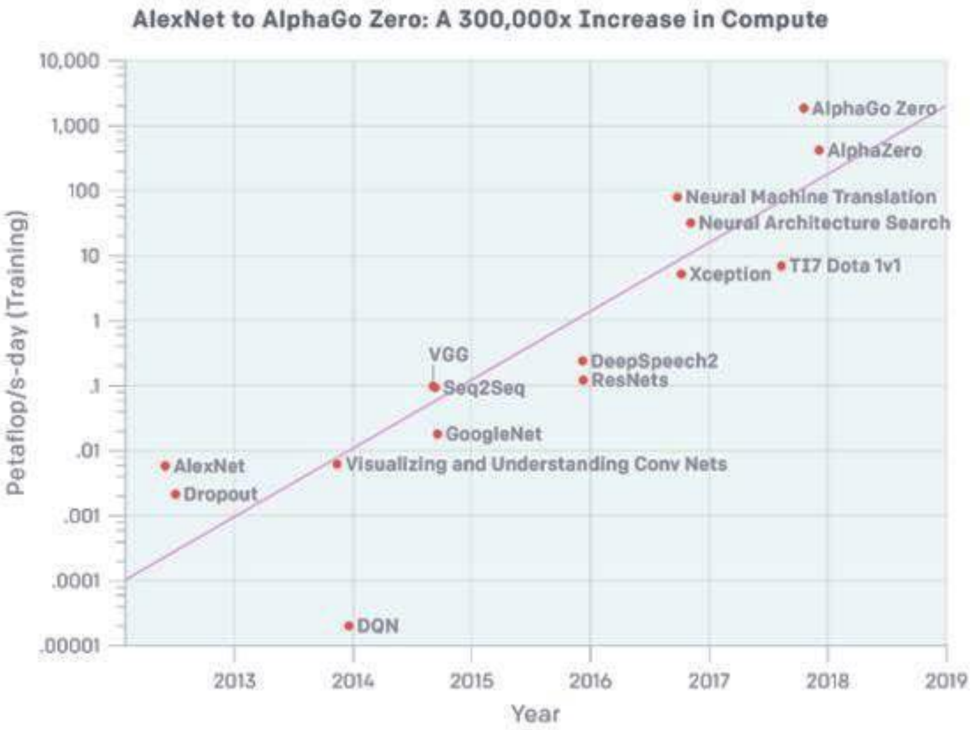
Current Model Won't Scale. Humans are **slow.** Slow decisions, slow to learn



Needed Solution: HPC steering itself (Days-Weeks)!

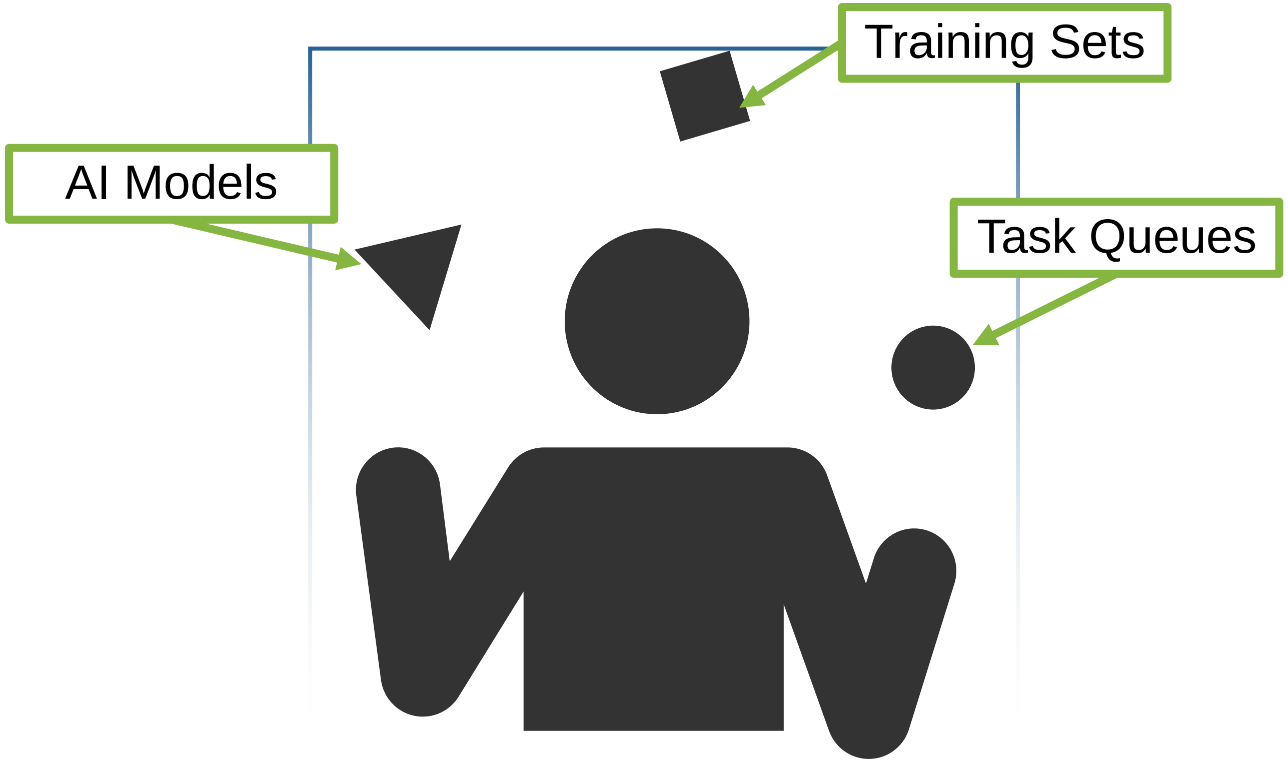
"Self-Steering HPC" is Difficult

AI Tasks Require Dedicated Compute



Source: https://openai.com/blog/ai-and-compute/, https://www.i-programmer.info/news/105/11823

Heterogenous Workflow Components

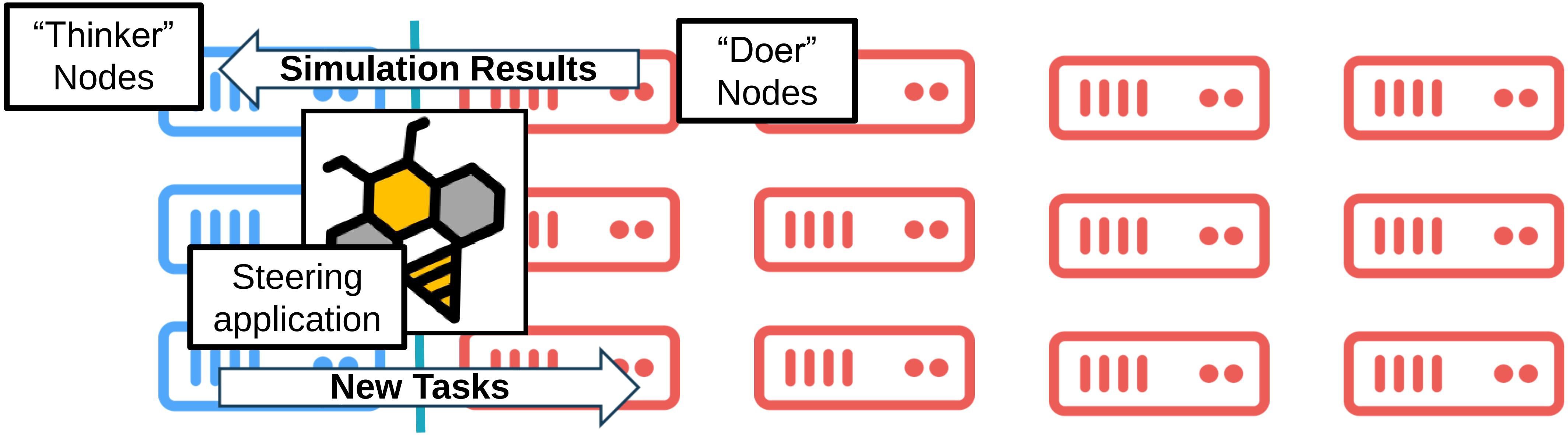


Our Goal: Design software to mitigate these two issues

EXASCALE COMPUTING PROJECT

Our Approach: **Colmena**

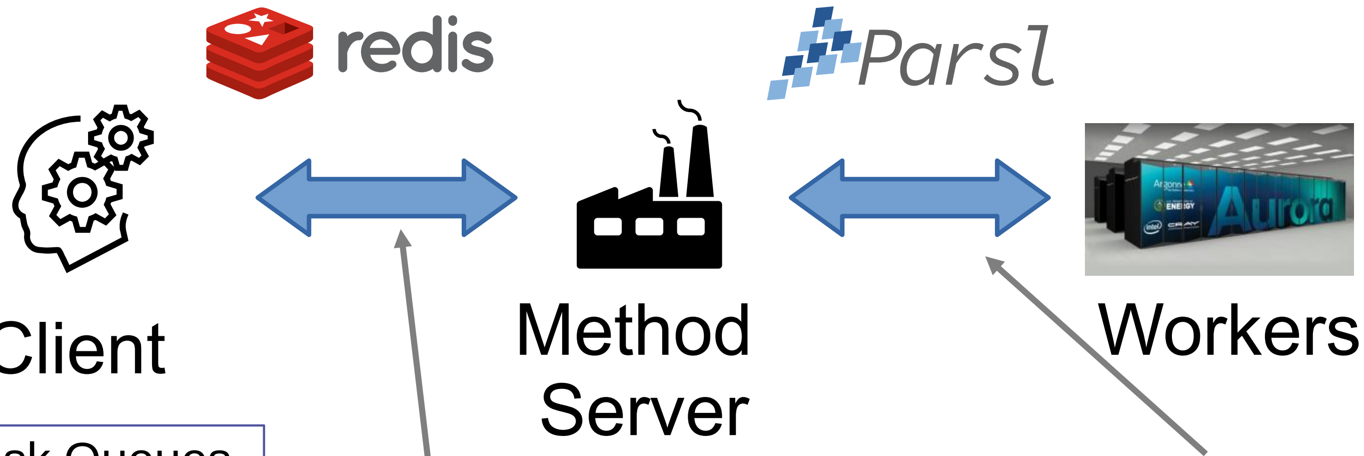
Concept: Steering application that submits tasks to separate resources



Design Goals:
- Simple expression of "AI in the loop" workflows
- Ability to partition resources between different tasks
- Extreme scaling, deployable on multiple resources

Colmena Design and example applications

Colmena is a wrapper over Parsl



Programming Model: Task Queues

```
# Primitive Units
queue.send_inputs(1)
result = queue.get_result()
```

Advantages:
- Multiple producer/consumers
- Minimal submission overhead

Disadvantages:
- No status checking
- Task workflows difficult

Message Format: JSON objects


```
{
    "inputs": [[1, 1], {"operator": "add"}],
    "method": "reduce",
    "value": 2,
    "success": true,
    "time_created": 1593498015.1324,
    "time_input_received": 1593498015.133,
    "time_compute_started": 1593498018.856,
    "time_running": 1.8e-05,
    "time_result_sent": 1593498018.858,
    "time_result_received": 1593498018.860
}
```


- Track task overhead
- No client/server lock-in to Python

Task Engine: Parsl

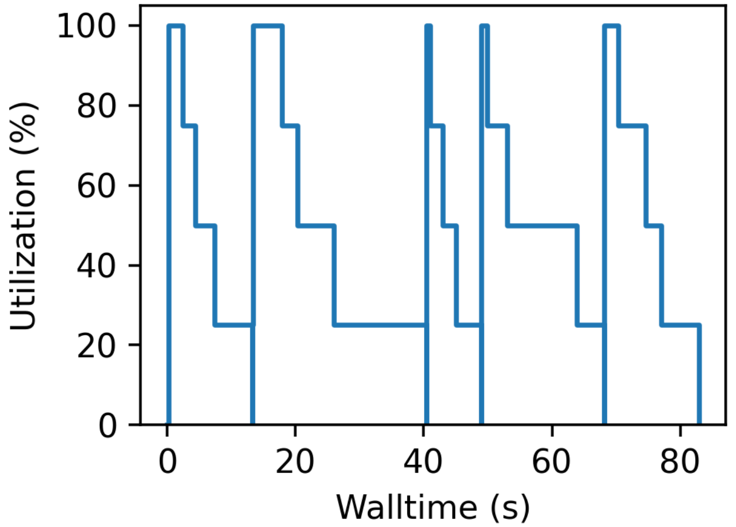
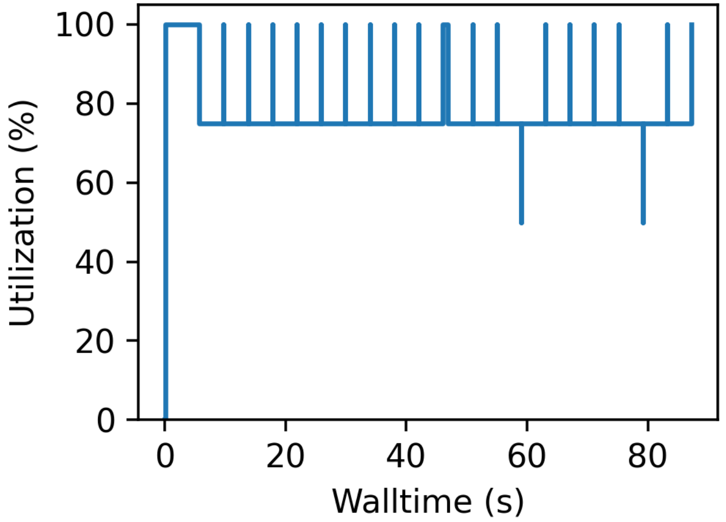
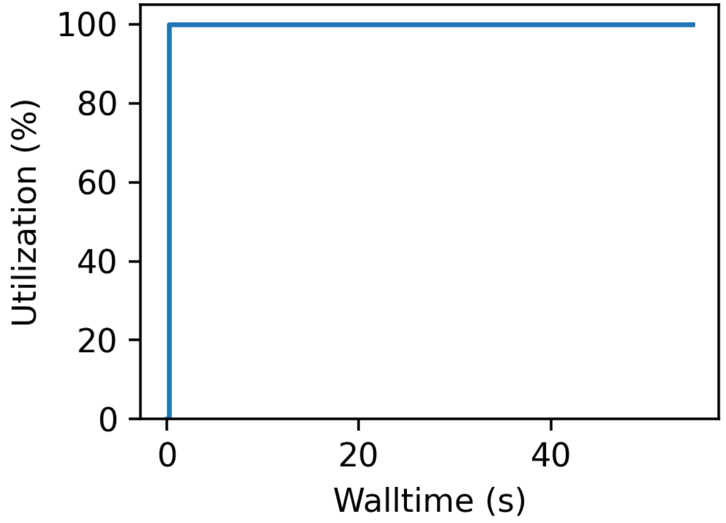
Advantages:
- Supports most HPC and cloud services
- Easily configure multiple worker types and multi-site workflows

Disadvantages:
- Limited support for ensembles of MPI applications [*in progress*]

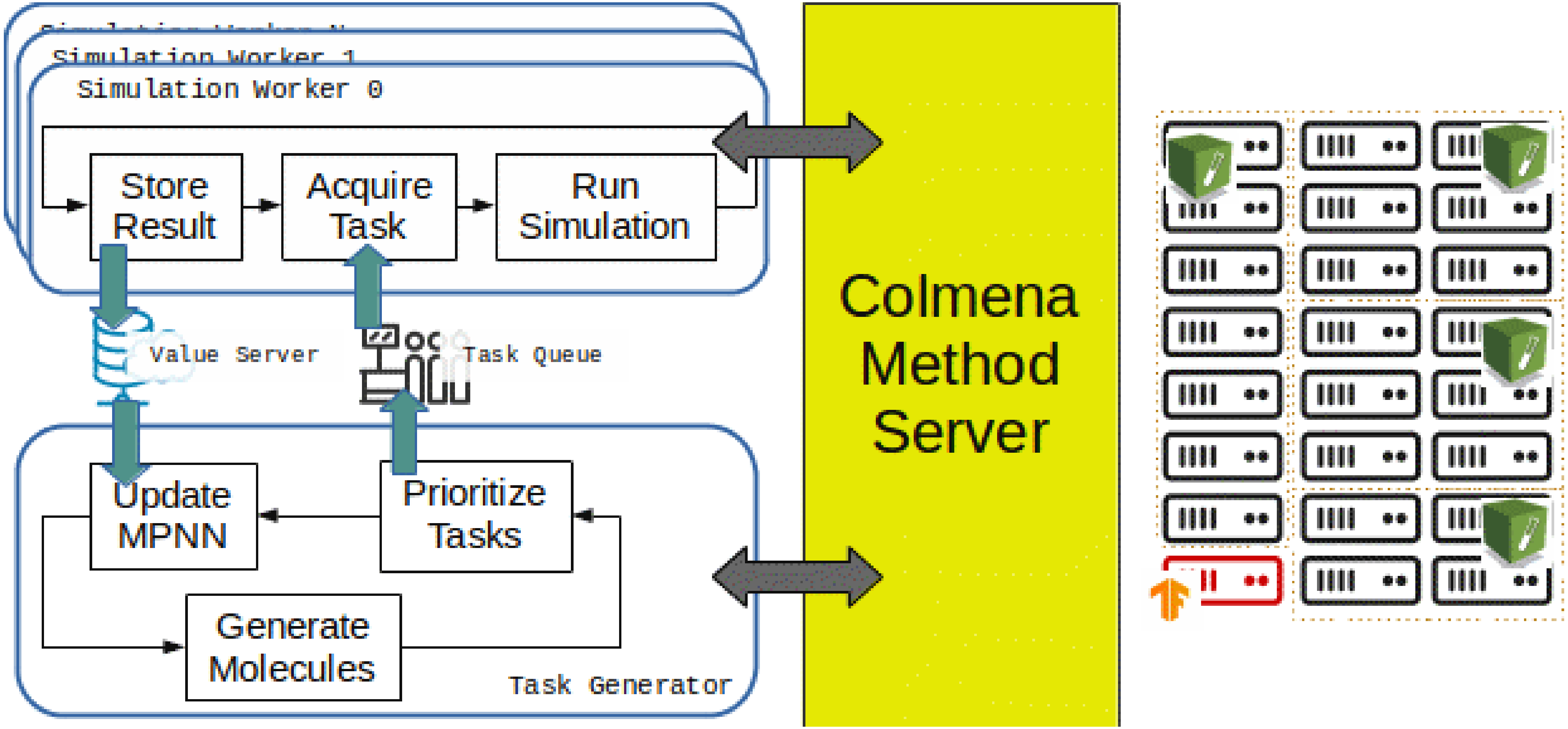
Colmena simplifies writing parallel optimizers

Faster task generation rates

Fewer calls to “select next tasks” code

Batch Optimizer	Streaming Optimizer	Interleaved Optimizer
		
Wait for N tasks to complete, then pick next batch	*Pick new tasks as soon as one completes*	*Maintain a task queue*

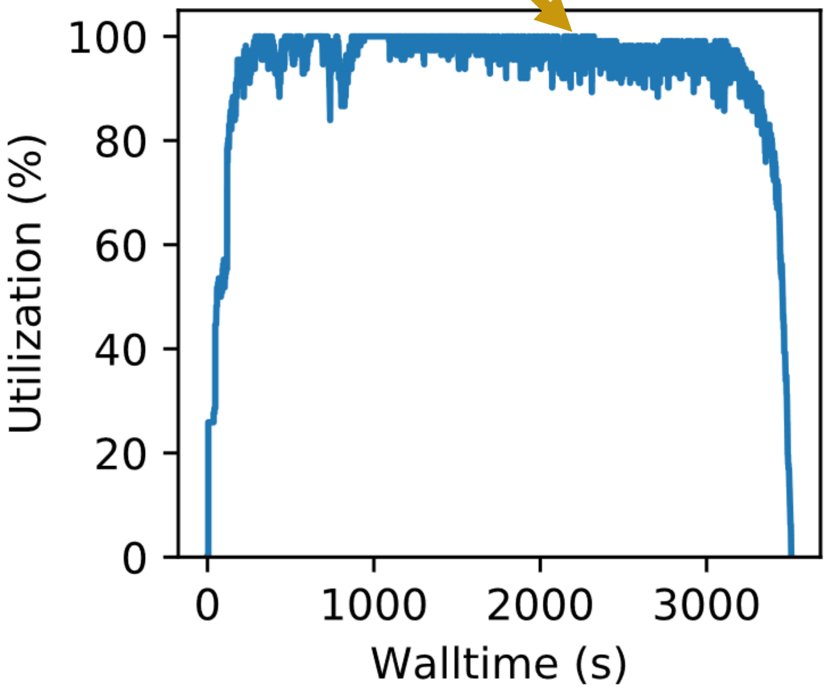
Example Application: Molecular Design with RL and NWChem

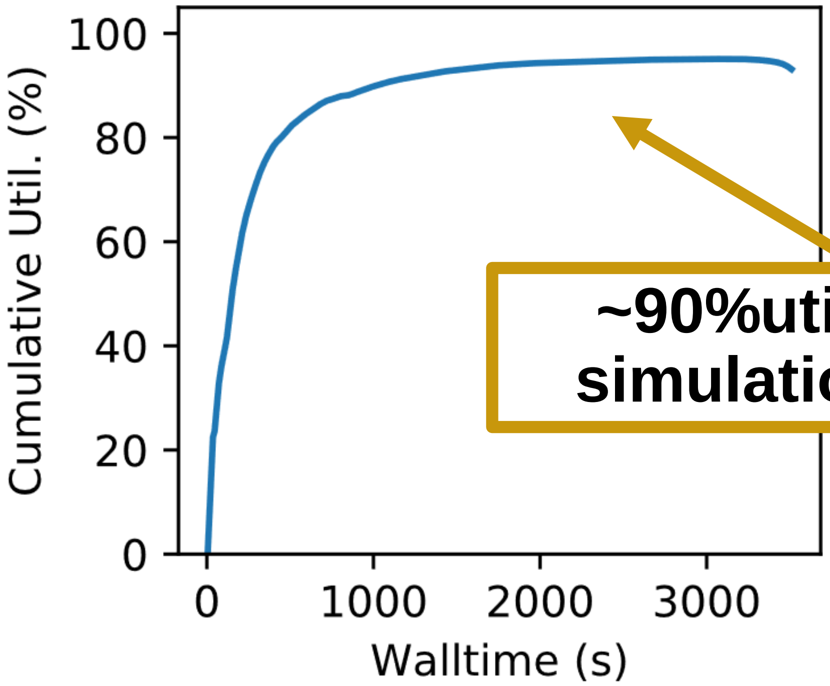


Colmena gives detailed task tracking

Sustained rate of ~3 task/sec

8 node debugging run:
- 112 simulation workers on 7 nodes
- 2 AI workers on 1 node

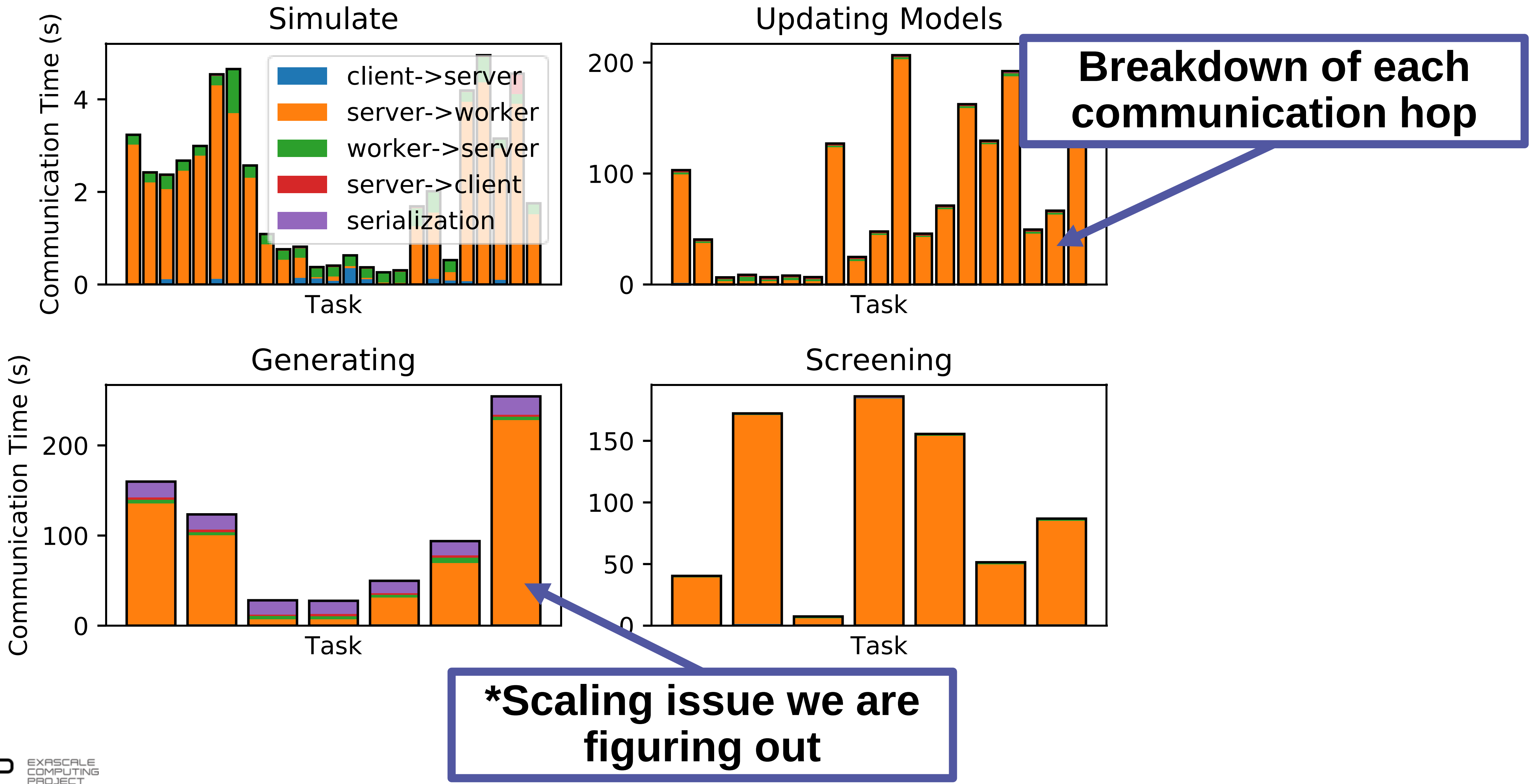




~90%utilization of simulation workers

*Scaling issue we are figuring out

Colmena gives detailed overhead measurements



EXASCALE COMPUTING PROJECT

Conclusions

- **Short version:** Building a library for OED/Active Learning on HPC
- **Where we are:** Building initial molecule design applications
  - https://colmena.readthedocs.io/en/latest/, https://github.com/exalearn/colmena
- **Where we are going:**
  - Understanding the full landscape of "exascale OED“
  - Studying communication overheads in steering algorithms
  - Evaluating optimal algorithms for learning at scale

Contact me! LWard@anl.gov

Acknowledgements

Funding: DOE Exascale Computing Project, ExaLearn Co-design Center

The team:

UC/Argonne: Yadu Babuji, Kyle Chard, Ryan Chard, Ian Foster, Ganesh Sivaraman, Rajeev Thakur

Brookhaven National Laboratory: Frank Alexander, Anthony DeGennaro, Shantenu Jha, Byung-Jun Kim, Kris Reyes, Li Tan

Example application: “Interleaved,” AI-in-the-loop optimizer