

# Future Programming Challenges for DOE

## from the Application Developer Perspective

David Richards  
ASC Co-design Project Deputy

April 7, 2016



LLNL-PRES-688064

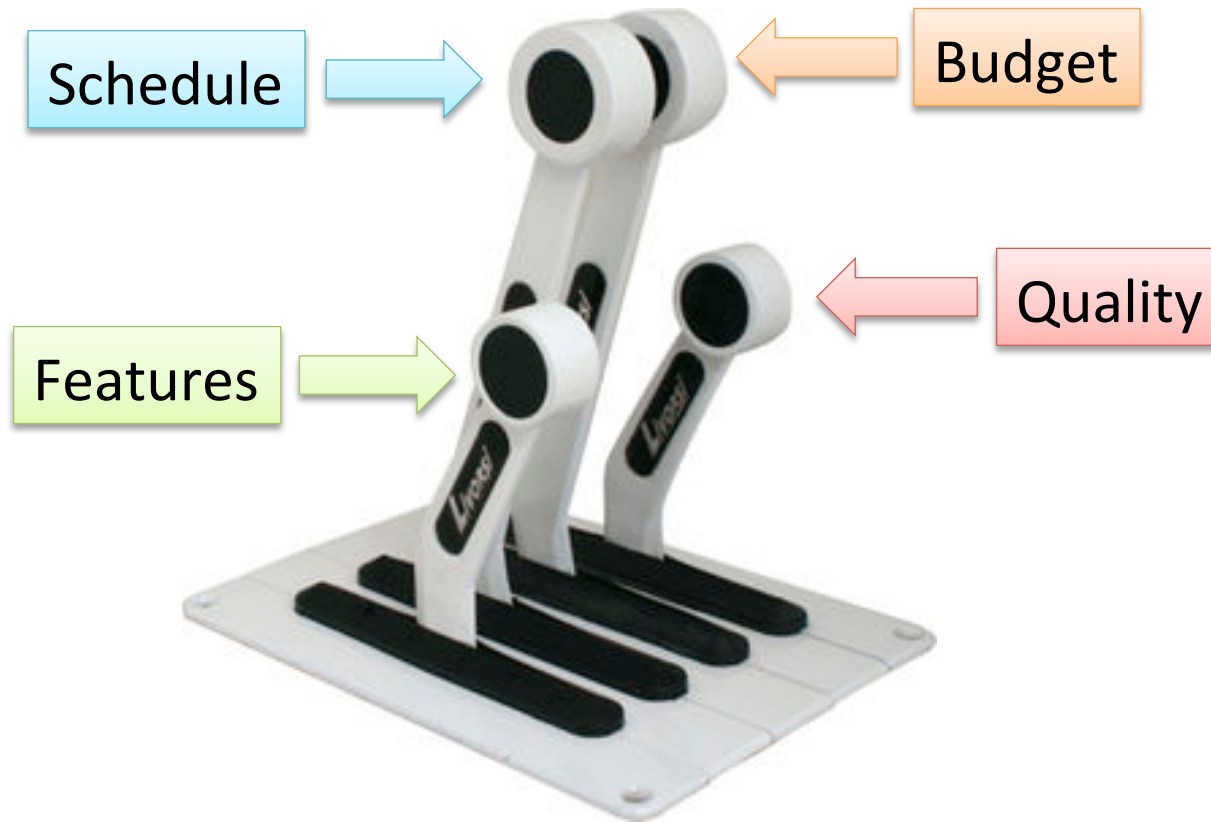
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

 Lawrence Livermore  
National Laboratory

# Every software project has a control panel

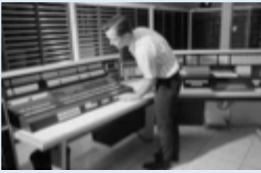
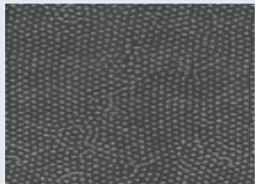

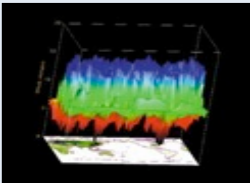



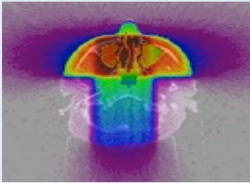
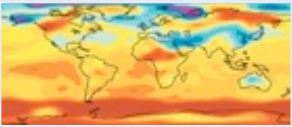

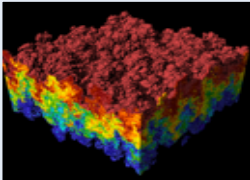


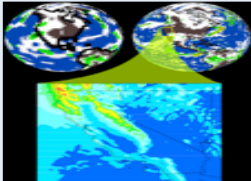
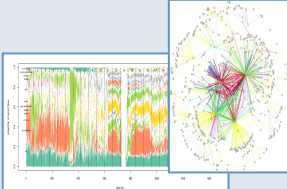


# Software projects have four control levers

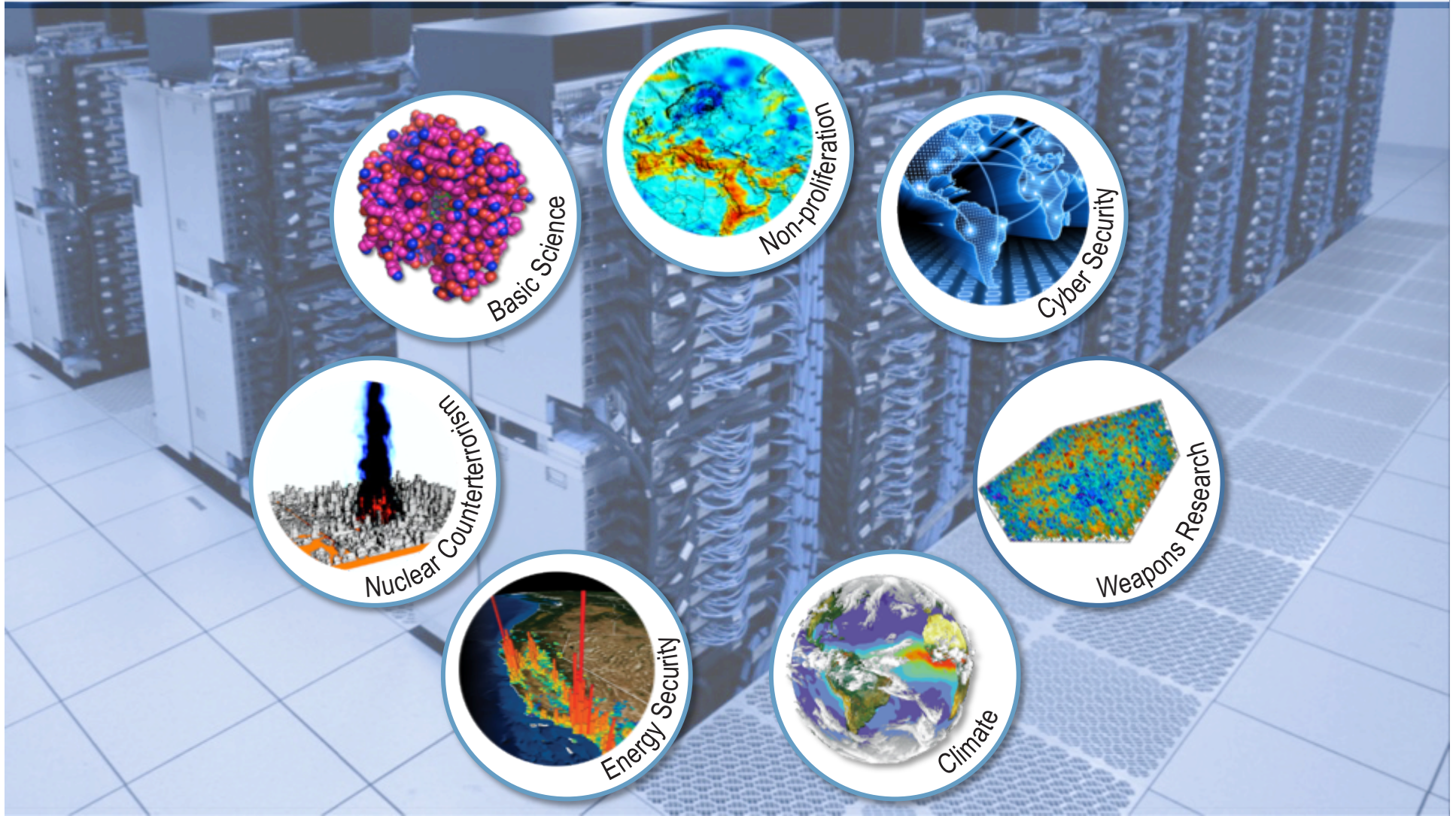


Managers can set any three levers.  
The fourth cannot be independently set!

# Livermore has been synonymous with supercomputing since our founding

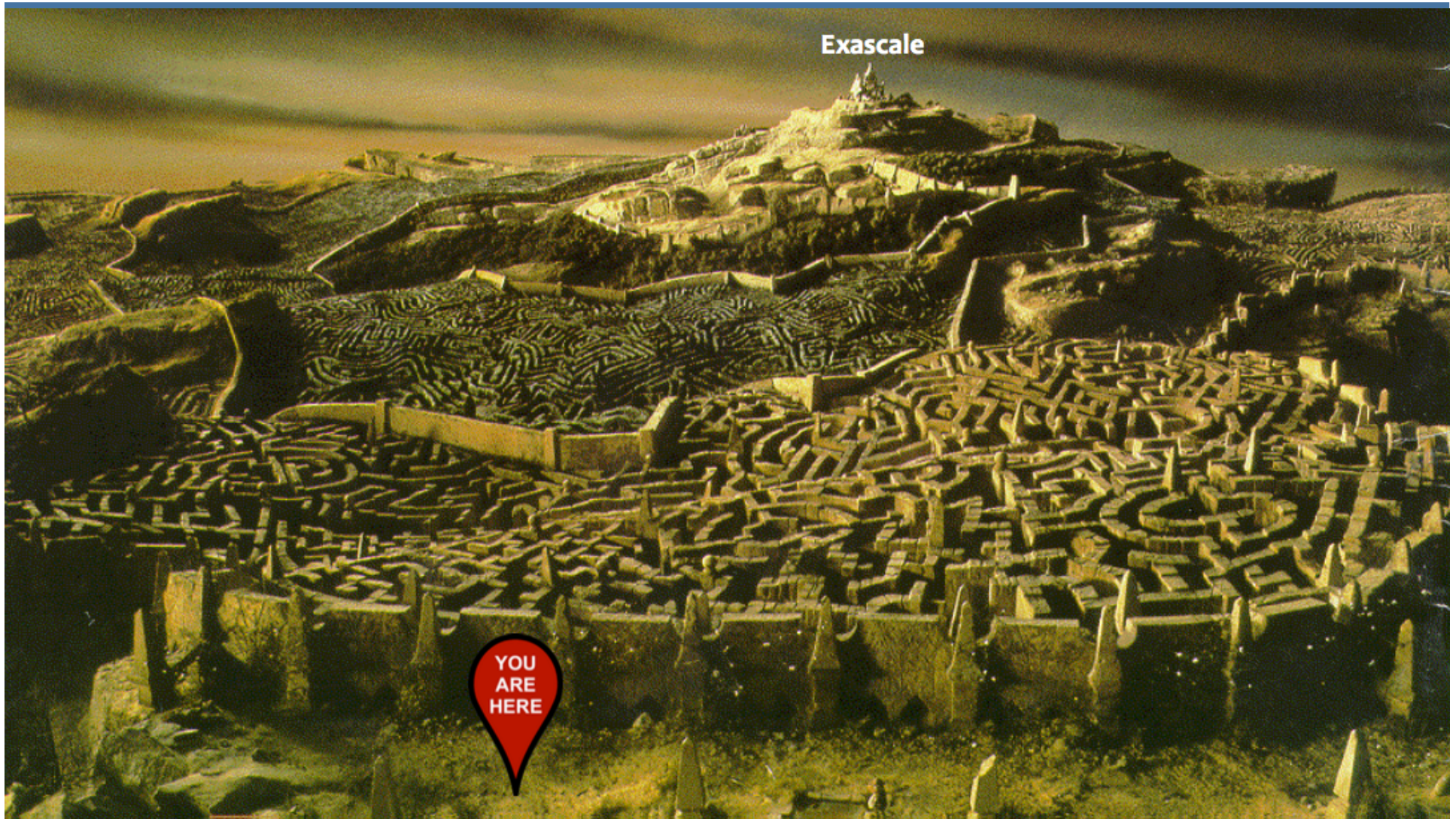
1960s	1970s	1980s	1990s	2000s	2010s
 <p>CDC 3600</p>  <p>Pioneering simulations of particle tracking</p>	 <p>CDC 7600</p>  <p>Ozone mixing models</p>	 <p>CRAY 1</p>  <p>Dynamics in three dimensions</p>	 <p>ASCI Blue-Pacific</p>  <p>Helping the medical community plan radiation treatment</p>  <p>Global climate modeling</p>	 <p>BlueGene</p>  <p>Breakthrough visualizations of mixing fluids</p>  <p>Unprecedented dislocation dynamics simulations</p>	 <p>Sequoia</p>  <p>Detailed predictions of ecosystems</p>  <p>Discovering patterns of behavior in the data</p>

# High performance computing is central to nearly every LLNL program



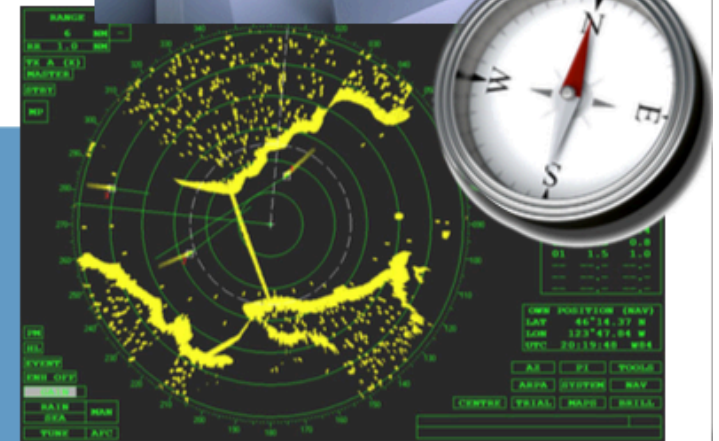
# Preparing large, complex codes for exascale

Trial and error is not an option



# Instead, use expert reconnaissance to find the best path forward

- Work with vendors to see what's ahead (and try to guide them in favorable directions).
- Develop and use proxy apps to quickly try out different strategies.
- Assess programming models, work with compiler developers to support the ones needed by DOE/LLNL.
- Learn to use the newest, best tools so we can teach others.



Need people who have experience developing efficient scientific applications and working with the latest programming models and hardware

# Exascale needs heros?





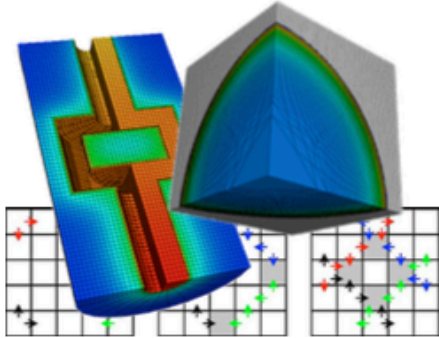
# You can't have everything!

(Where would you put it?)

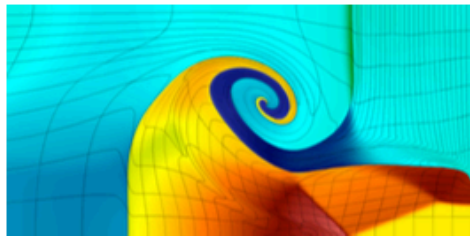


Different resource allocations produce different outcomes.  
Not right or wrong, just different

# Advanced Architectures Portability Specialists efforts are focused on three key areas



**Paths Forward**  
Rapidly assessing new programming models and hardware using proxy apps



**Application Impact**  
Working with code teams to understand real world use cases not captured by proxies



**Communication & Outreach**  
Interacting with vendors, researchers to share lessons learned and gather best practices

# LLNL (ASC) exascale programming environments

---

## Common

- C++, C, Fortran
- MPI + X
- OpenMP 4.X
- Raja + Chai
- Kokkos
- Generated Code /  
Embedded DSL

## Less Common

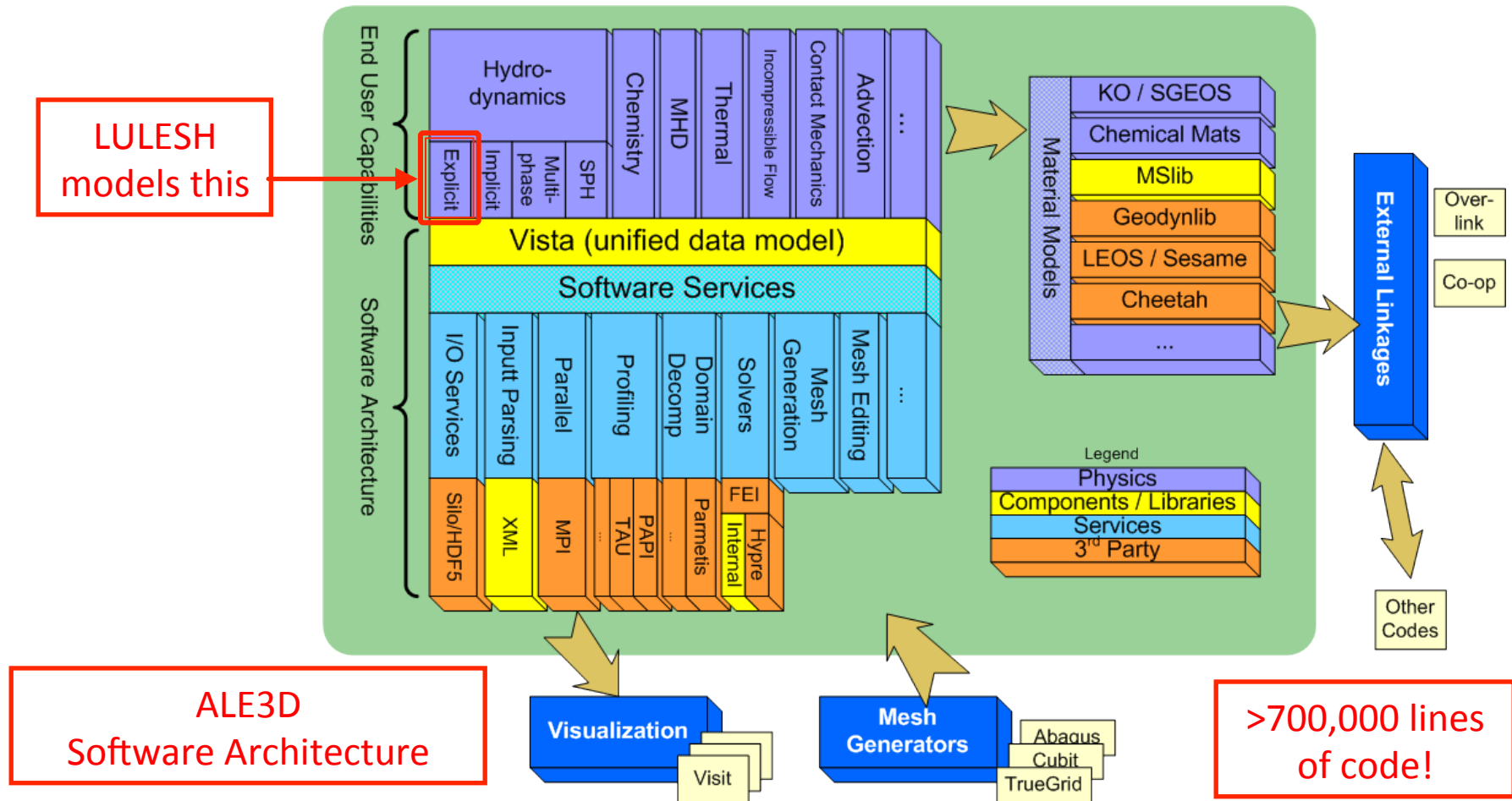
- Task-based models
- PGAS models
- CUDA
- OpenACC

# Applications and Complexity

---

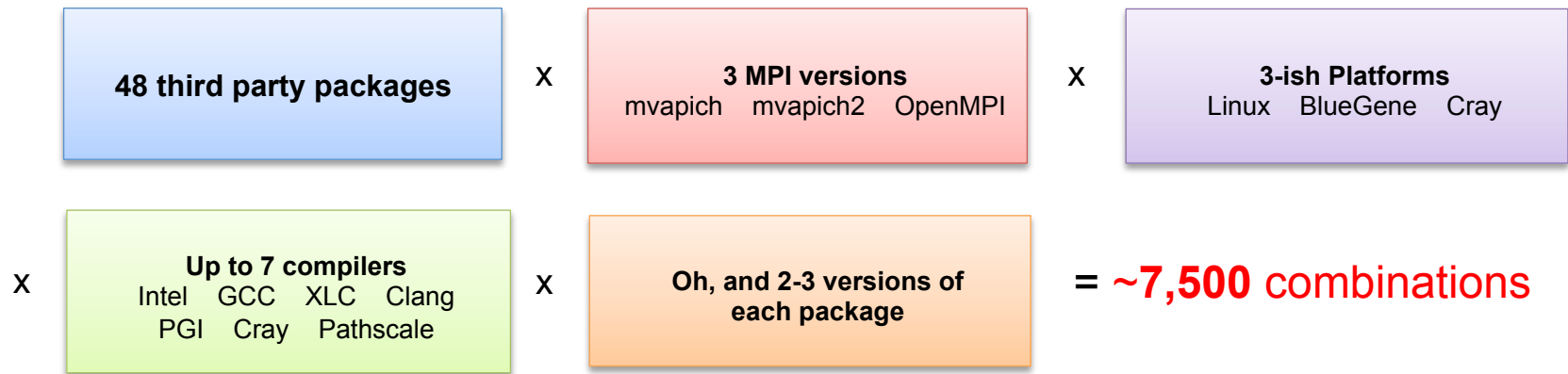
- Applications are more complex than you think
  - The following are **NOT** apps:
    - Proxy apps
    - Cholesky, Fibonacci, Matrix Multiply
    - SAXPY
- Typical reactions on first encounter with a *real* application:
  - “This is the most complex thing we have ever seen.”
  - “Oh, your array sizes aren’t declared at compile time?”
  - “None of the techniques we know will work for this.”
  - “We’ll get back to you.”

# LULESH models explicit hydrodynamics, which is a very small fraction of a real application code



# Complexity is multiplied by the hardware and software stack

- Not much standardization in HPC: every machine/app has a different software stack
- Sites share unique hardware among teams with *very* different requirements
  - Users want to experiment with many exotic architectures, compilers, MPI versions
  - All of this is necessary to get the best *performance*
- Example environment for some LLNL codes:



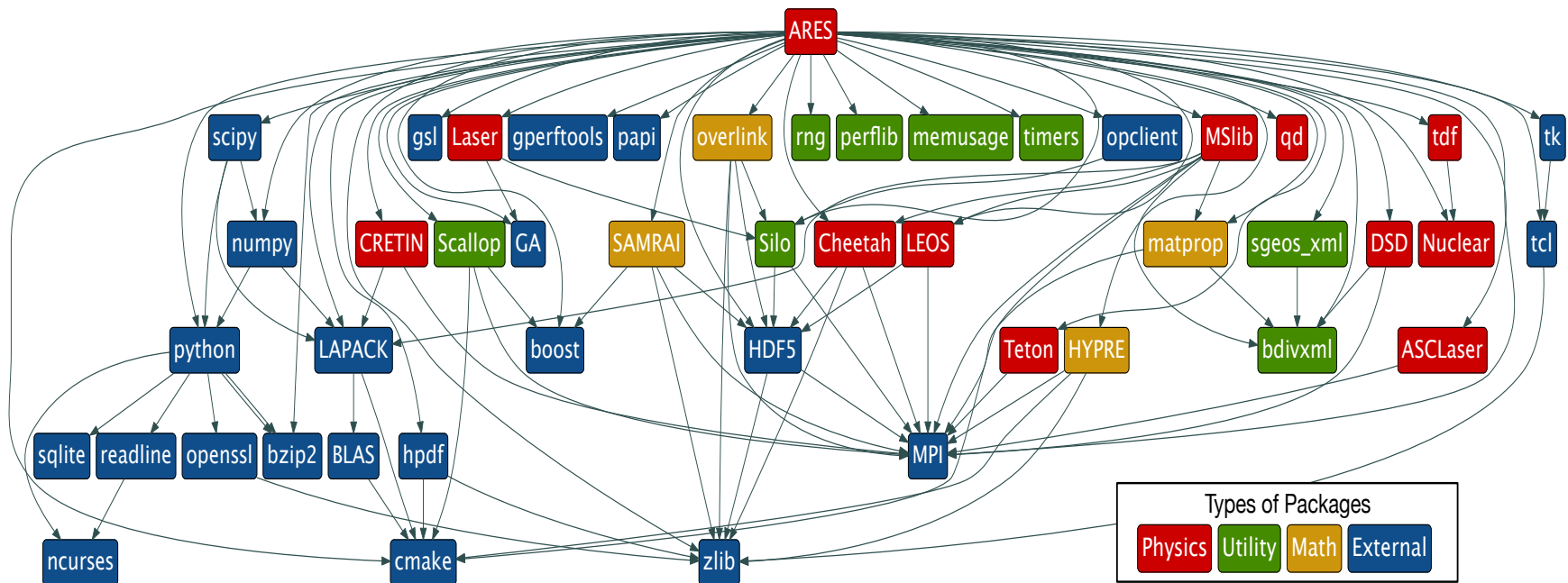
We want an easy way to quickly sample the space, to build configurations on demand!

# What's Hard About Building/Porting?

---

- Many libs (50) – public/local, each with own build system, can be specific to platform/compiler/site
  - Other people's code/make systems – difficult to generalize or get working on unexpected configurations
  - Change to not use default compiler? Port to uncommon (new) architecture? Use at other site (paths/groups)?
- Some compilers pickier than others, can be very finicky – what works for one doesn't work for all
  - May need different or more system include files, or have include file order dependency issues
  - May not have features you expect – C++11 (XLC), threads (CLANG), forking (Cray), atomics (PGI)
  - Parameter types subtly different, casting confusion, namespace contention, templating issues
- Even widely available libs can be difficult (like Python) due to cutting edge architectures
  - For Sequoia and its lightweight kernel, IBM supplies a patch for building Python
  - Patch is specific to particular version of Python and is for IBM's xlc compiler – tweaks needed for any other
  - Cross-compilation issues, front-end/back-end node differences, static vs. dynamic build issues
  - Version of compiler too – both major (features, like C++11) and minor (Optimization or bugs!)
- Even after compiled – library may not link!
  - Unresolved externals, undefined or multiply defined calls (even system calls), Fortran underscoring
  - Hunt down missing system libs, order of libraries being loaded can be an issue too! Wrong or missing paths
- Even after executable made – may not run!
  - Illegal instructions, mismatched MPI or system lib versions, memory alignment issues, missing load paths
- Confounding error messages, pages of spew due to one error, difficult to understand or find
  - Even after solved and hacked for one configuration, changes may break other configurations
  - Hard to keep straight all the fixes and versions and compilers and architectures and libraries – many combos
  - Very frustrating, confusing, tedious – to the point of quitting! ...What can make this task less painful?

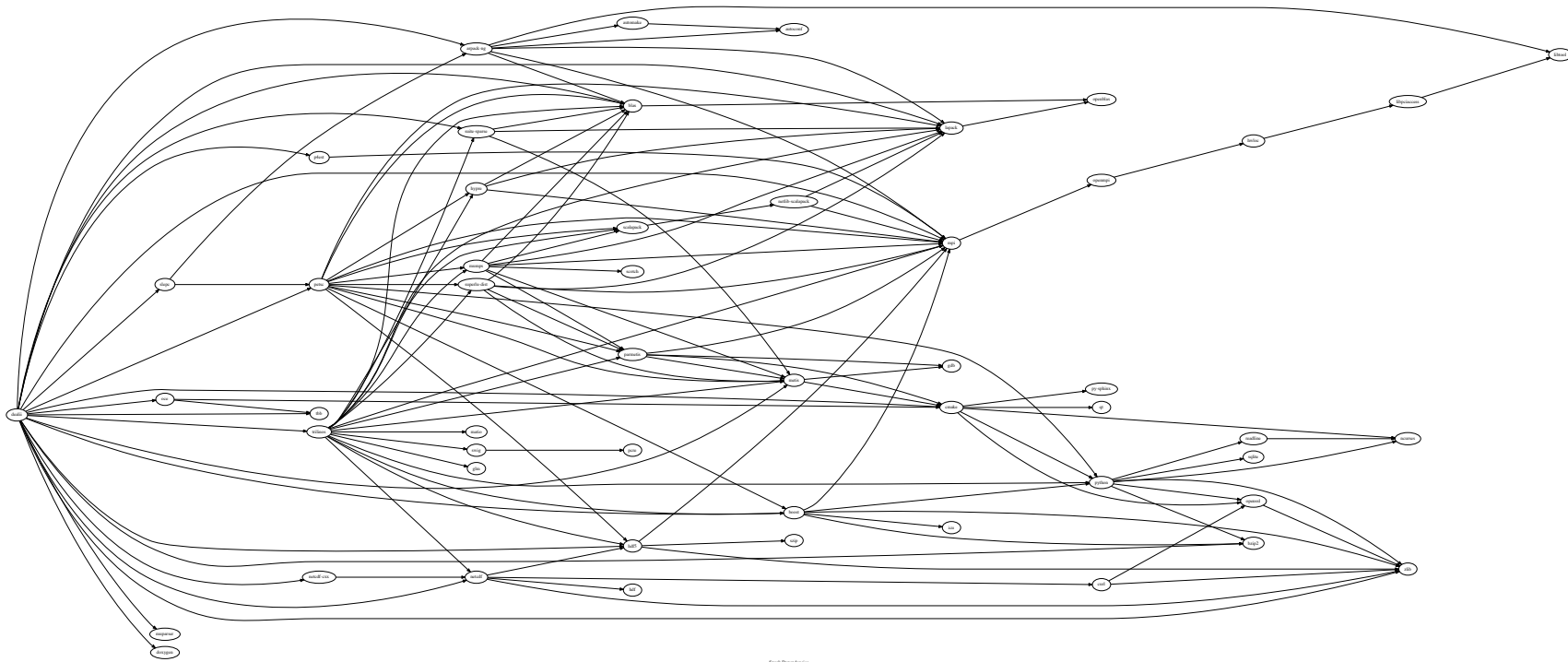
# Spack builds real LLNL codes



- ARES is a 1, 2, and 3-D radiation hydrodynamics code
- Spack automates the build of ARES and all of its dependencies
  - The ARES configuration shown above has 47 dependencies



# Trend is for increasing complexity



Spack dependency graph for deal.II, an open source finite element library

# Applications, developers, & tools

---

- How do application developers choose their tools?
- What can tool developers do to help application developers?

# My Basic Toolbox

printf

grep

awk

emacs

gnuplot

CXX

App internals

# Using printf and grep doesn't mean you're a knuckle-dragging troglodyte

---



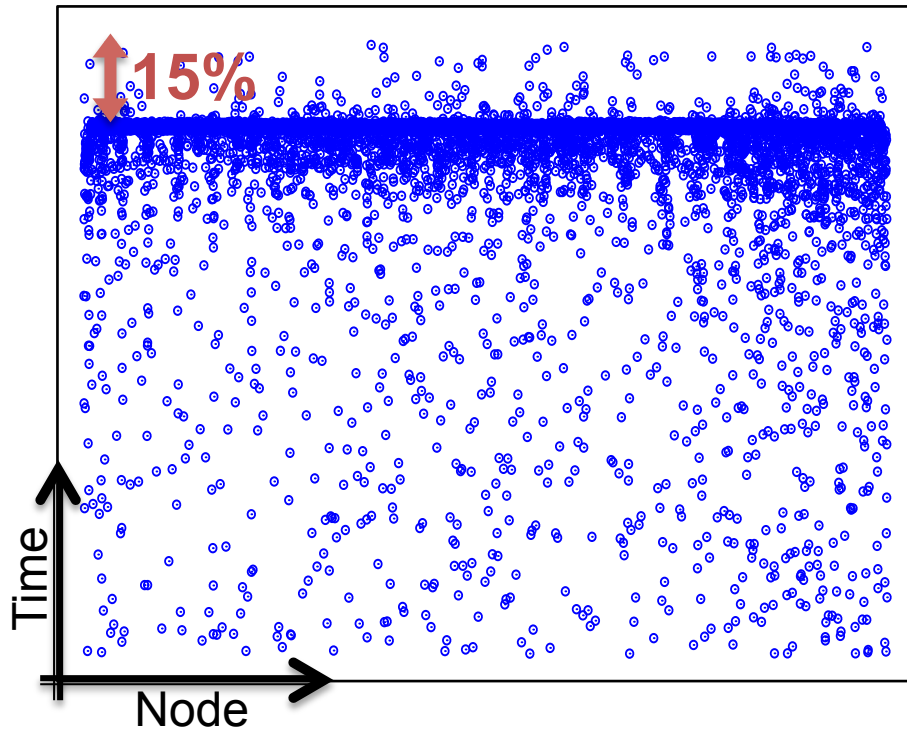
# Using printf and grep doesn't mean you're a knuckle-dragging troglodyte

- Non-debugger users tend to:
  - Work at larger scale
  - Complain about debugger performance. Recall numerous examples of tool failures
  - Employ extensive diagnostics built into their applications
  - Work on platforms where tool availability/performance is poor
  - Face complex data driven anomalous code behavior
  - Hate sitting in traffic jams
- Debugger users tend to:
  - Work at smaller scale (or they're very patient)
  - Use debuggers daily
  - Employ extensive verbosity and diagnostic features built into their applications
  - Work in unfamiliar code bases
  - Encounter problems that are not data driven

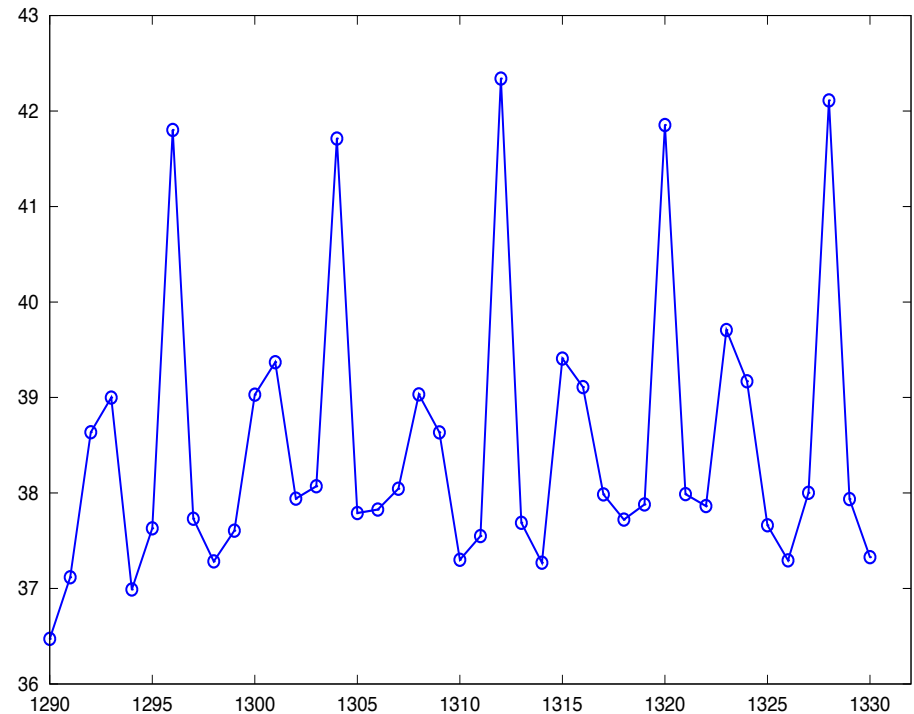
Standard tools fail to meet many common developer needs

# Finding the cause of a performance problem: Cache conflicts & memory alignment

Execution time for non-gates

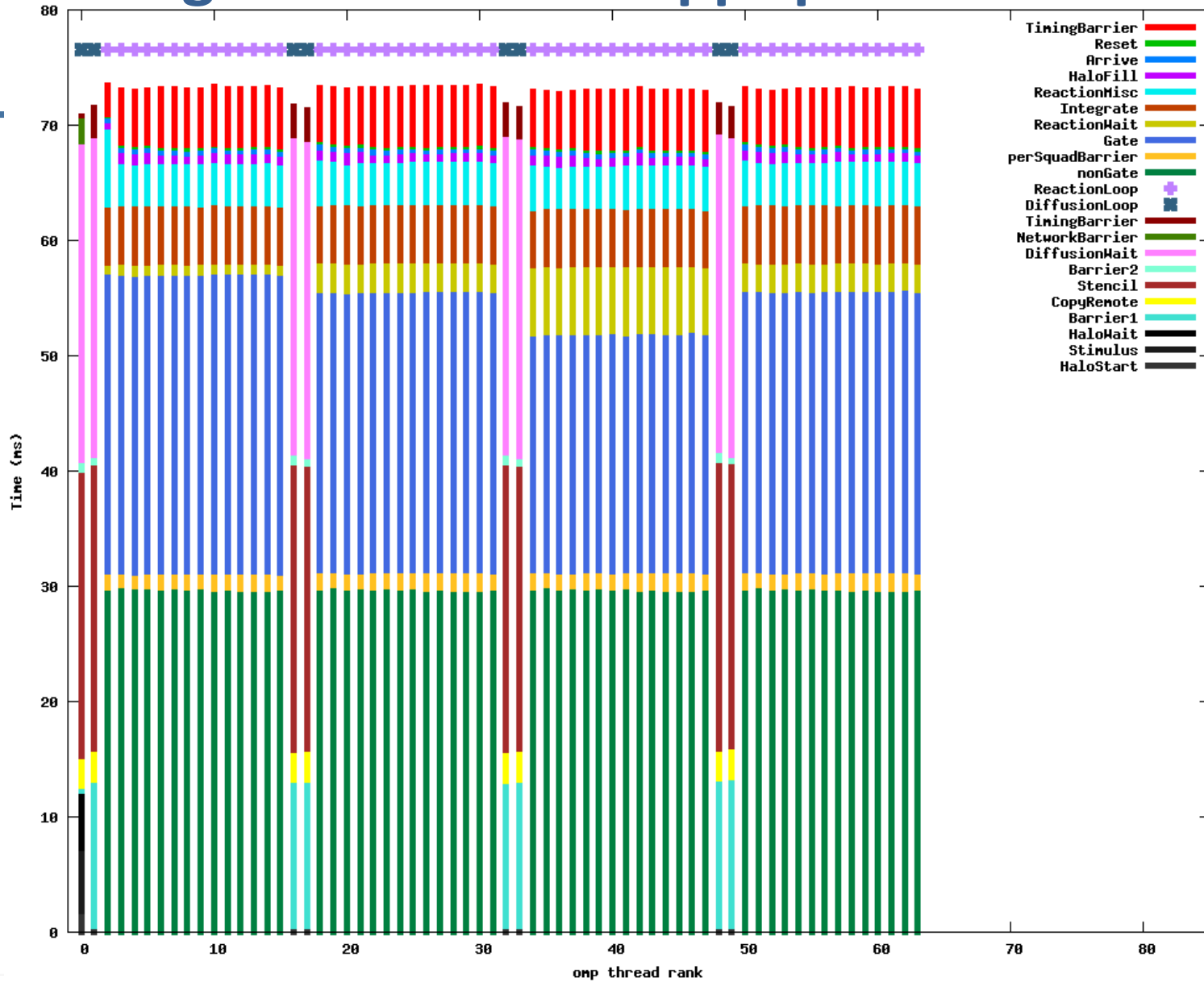


Execution time vs number of cell vectors on a node



Very small differences in memory alignment on a few nodes causes a noticeable performance decrease

# Threading time line with app specific data



## My Extended Toolbox

STAT

gprof

hpctoolkit

totalview

gdb

rose

This list should probably be longer

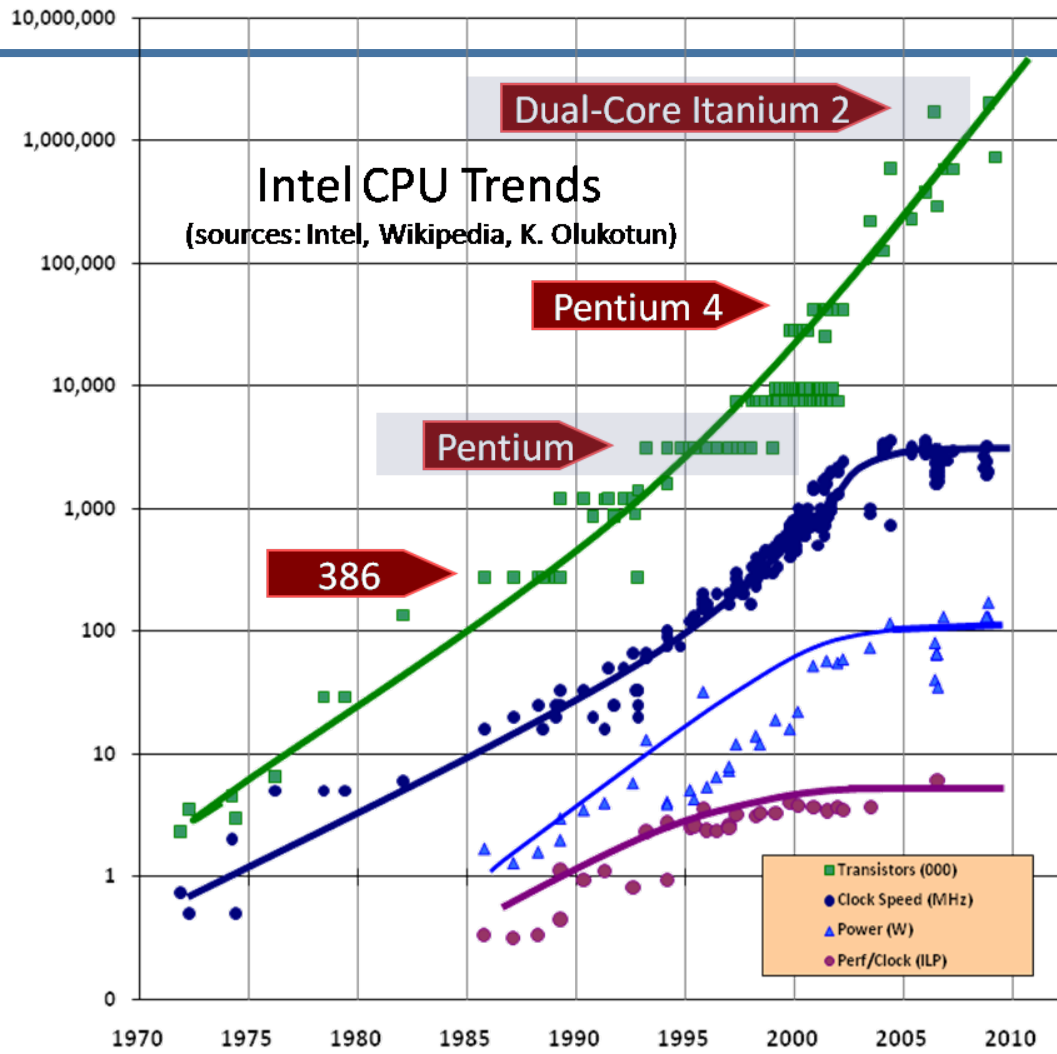


# Recommendations for tool developers

---

- Make sure your tools work!
  - Misleading or incorrect metrics
  - Tools that only work on toy examples
  - Worry about poor performance or scalability
- Developers must comprehend increasingly large data sets
  - Analytic and visualization capabilities are critical
- Focus on emerging exascale pain points
  - Analysis of fine grained parallelism
  - Memory hierarchies and data movement
  - Debugging and tuning “new” programming models
- Use co-design with application developers to build better tools
  - Tools can work with the application instead of on the application
  - Identify application practices that will make tools work better
  - Provide information in the application domain

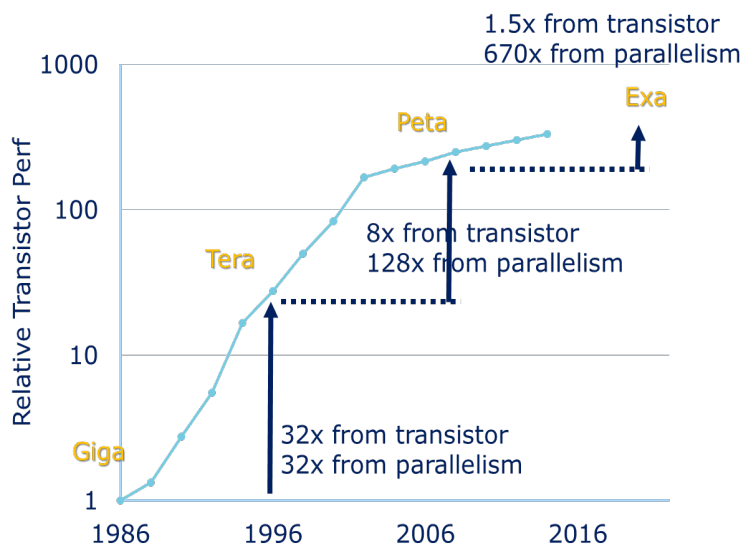
# TANSTAAFL: The End of Clock Scaling



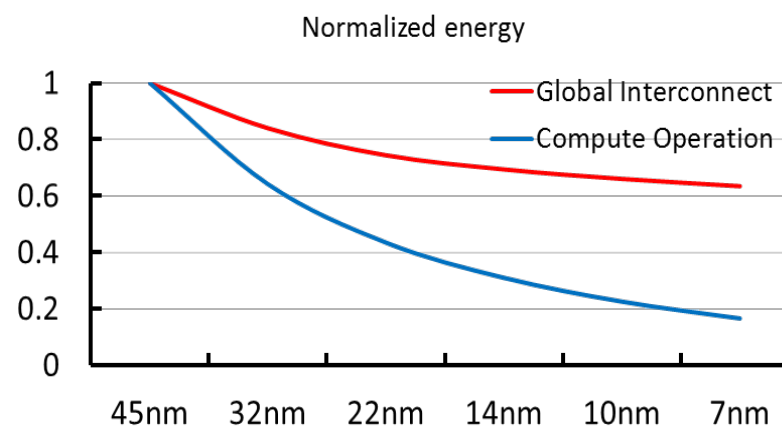
from [The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software](#)

# Historic scaling trends are running out of gas

## Transistors don't scale

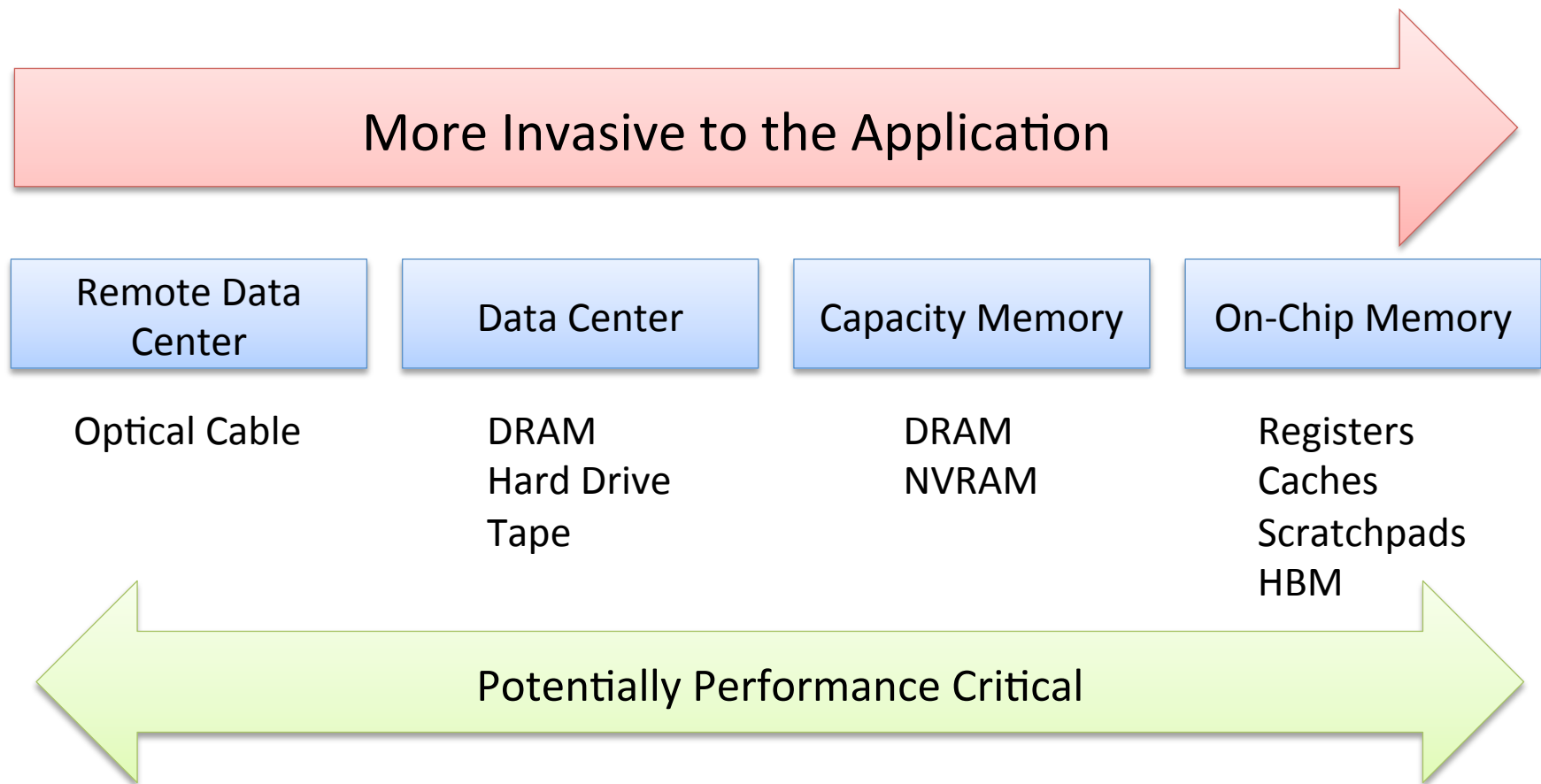


## Interconnects don't scale



Graphs from Shekhar Borkar

# Memory is getting more complicated



Current cache technology is unlikely to solve this problem

# This is a scalability problem.

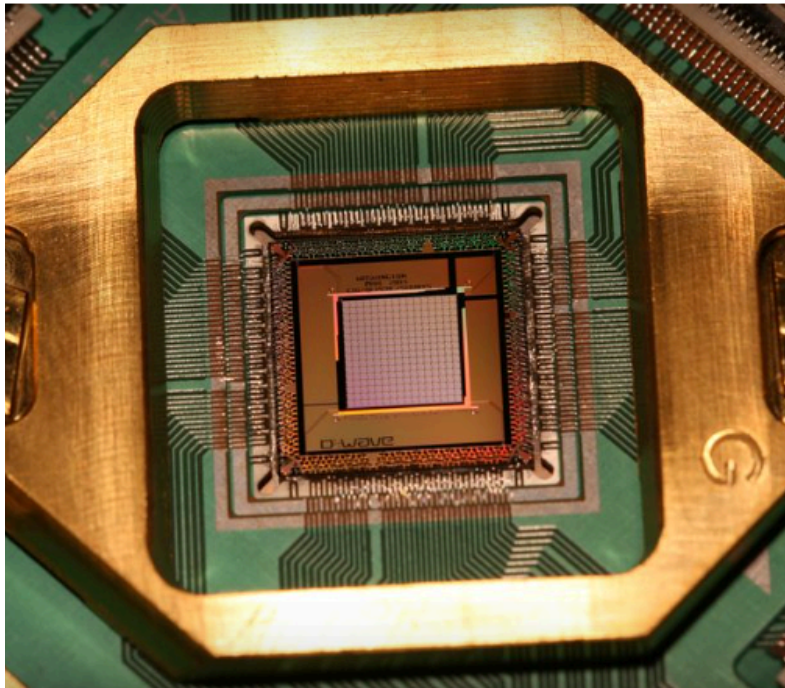
---



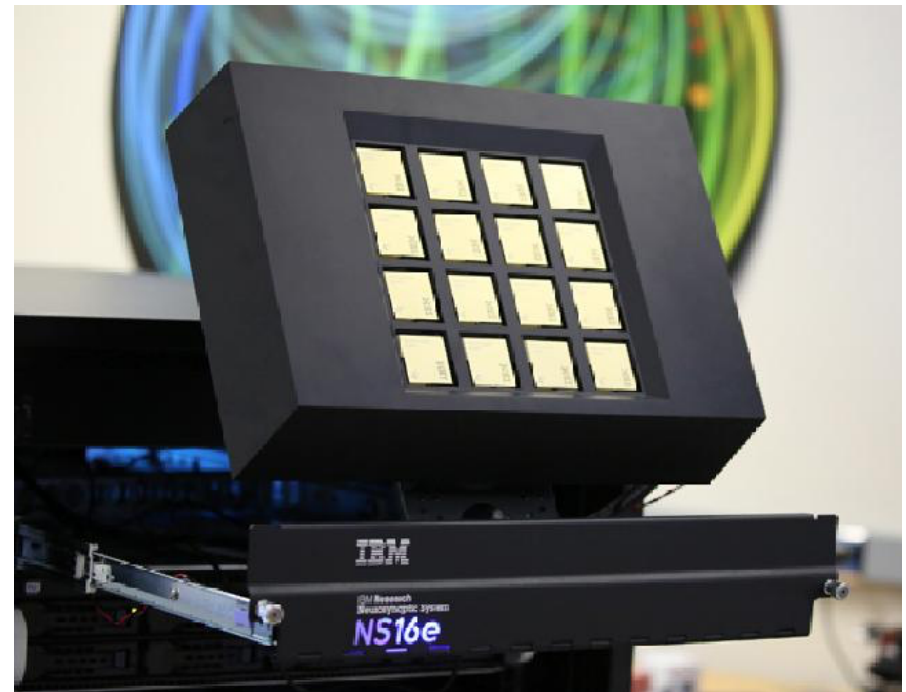
How can we solve the problem of the human in the loop?

# Beyond Moore's Law

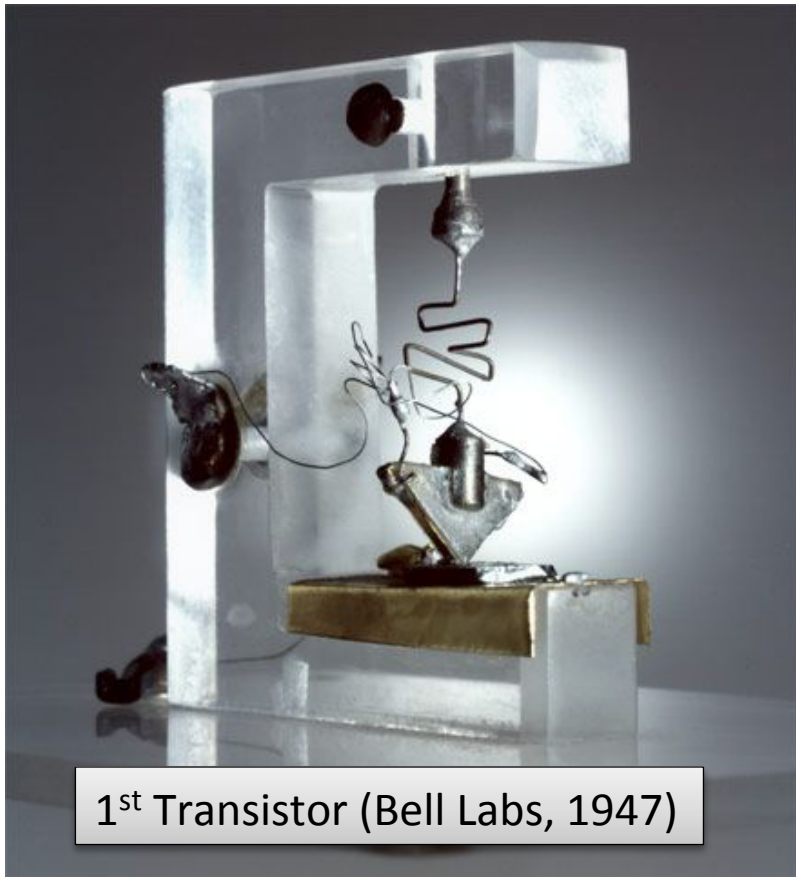
D-Wave quantum annealing, LANL



True North brain-inspired, LLNL



“It’s tough to make predictions,  
especially about the future.” – Yogi Berra



Would you have predicted the iPhone?

# Beyond Moore's Law: More questions than answers

---

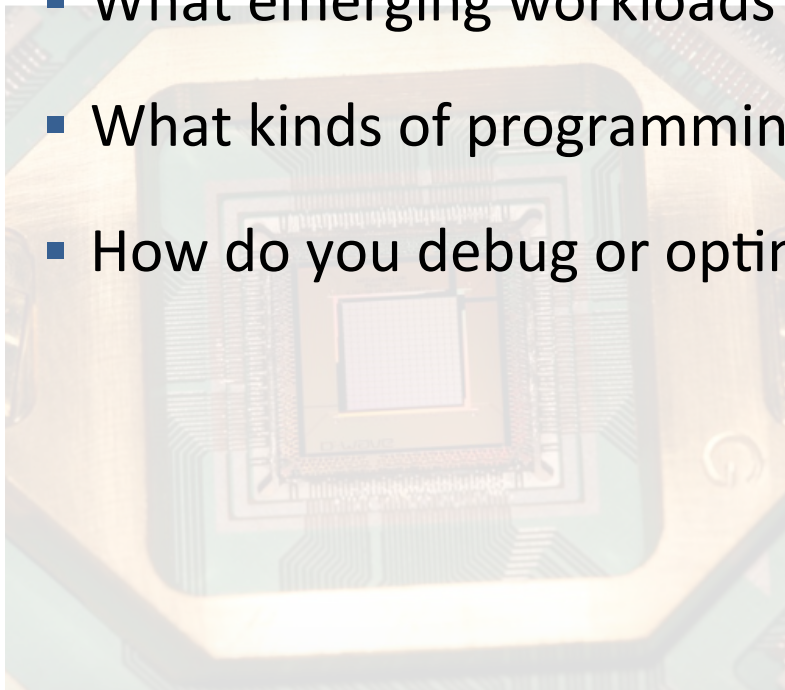
- How do our current workloads map to these capabilities?

D-Wave quantum annealing, LANL

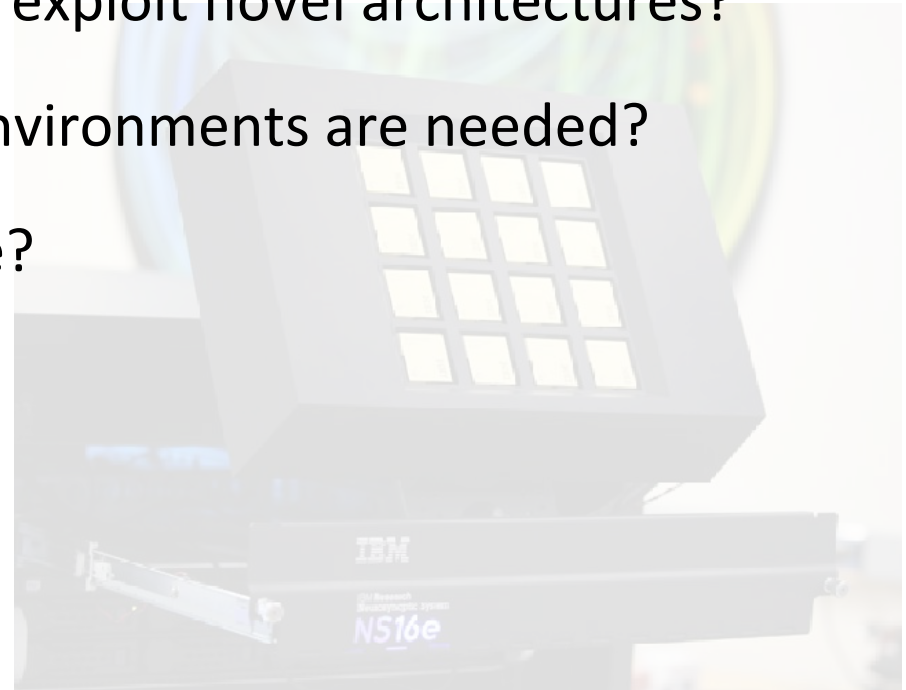
- What emerging workloads can exploit novel architectures?

- What kinds of programming environments are needed?

- How do you debug or optimize?



True North brain-inspired, LLNL



I predict a long path to mainstream adoption



# Summary/Conclusion

---

- Understand the resource allocation choices that drive application development
  - Work within these choices, not against them
- Help applications express and expose concurrency
  - Some applications and algorithms will get left behind
- Develop methods to choreograph memory traffic
- Build tools to understand bottlenecks
  - Why is my code slow
  - Powerful data query interfaces
- How will we extract understanding from computation/data



**Lawrence Livermore  
National Laboratory**