Enhancing User Productivity with Tools for Petascale Computing

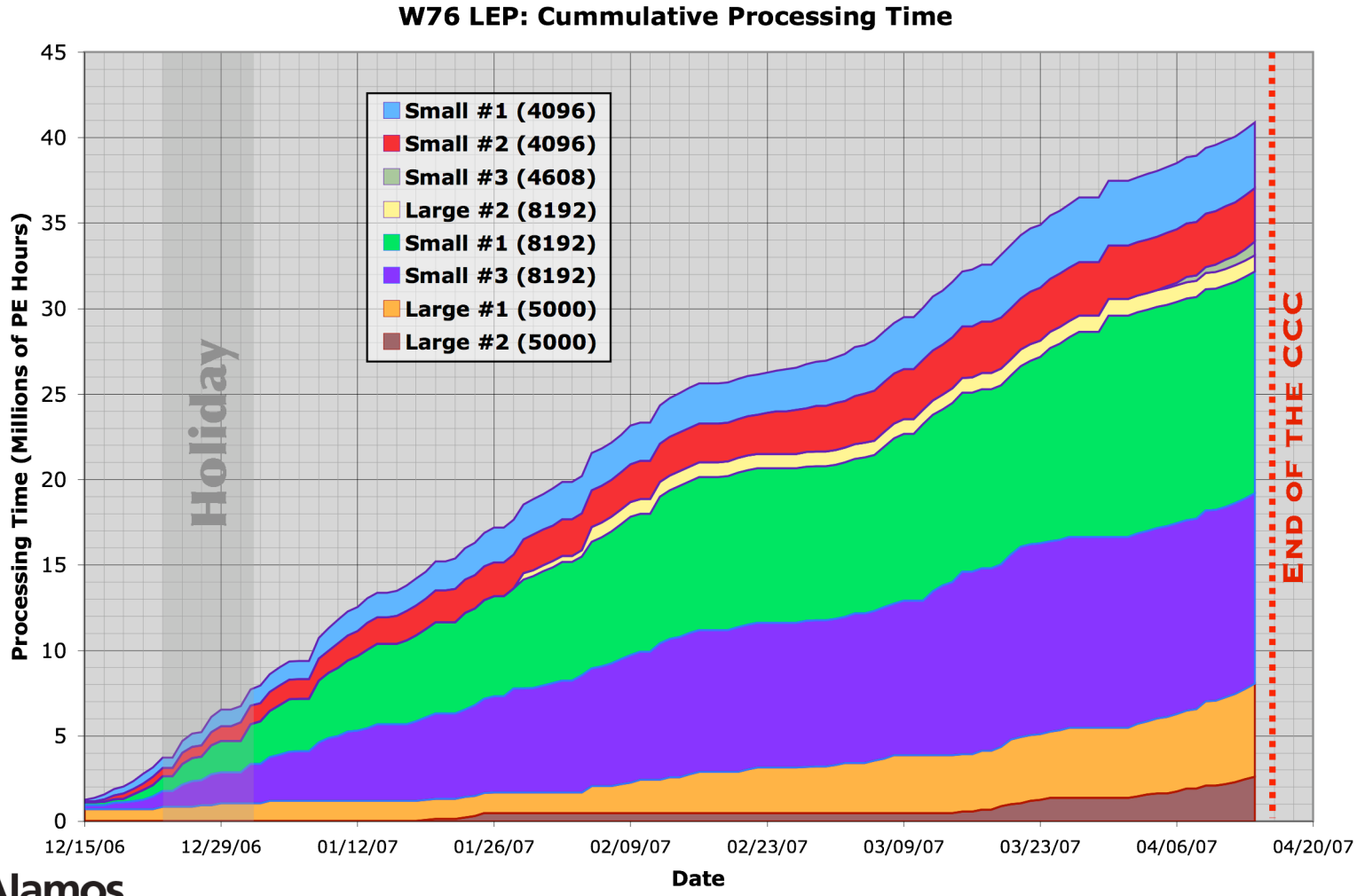
John T. Daly

High Performance Computing Environments (HPC-4)

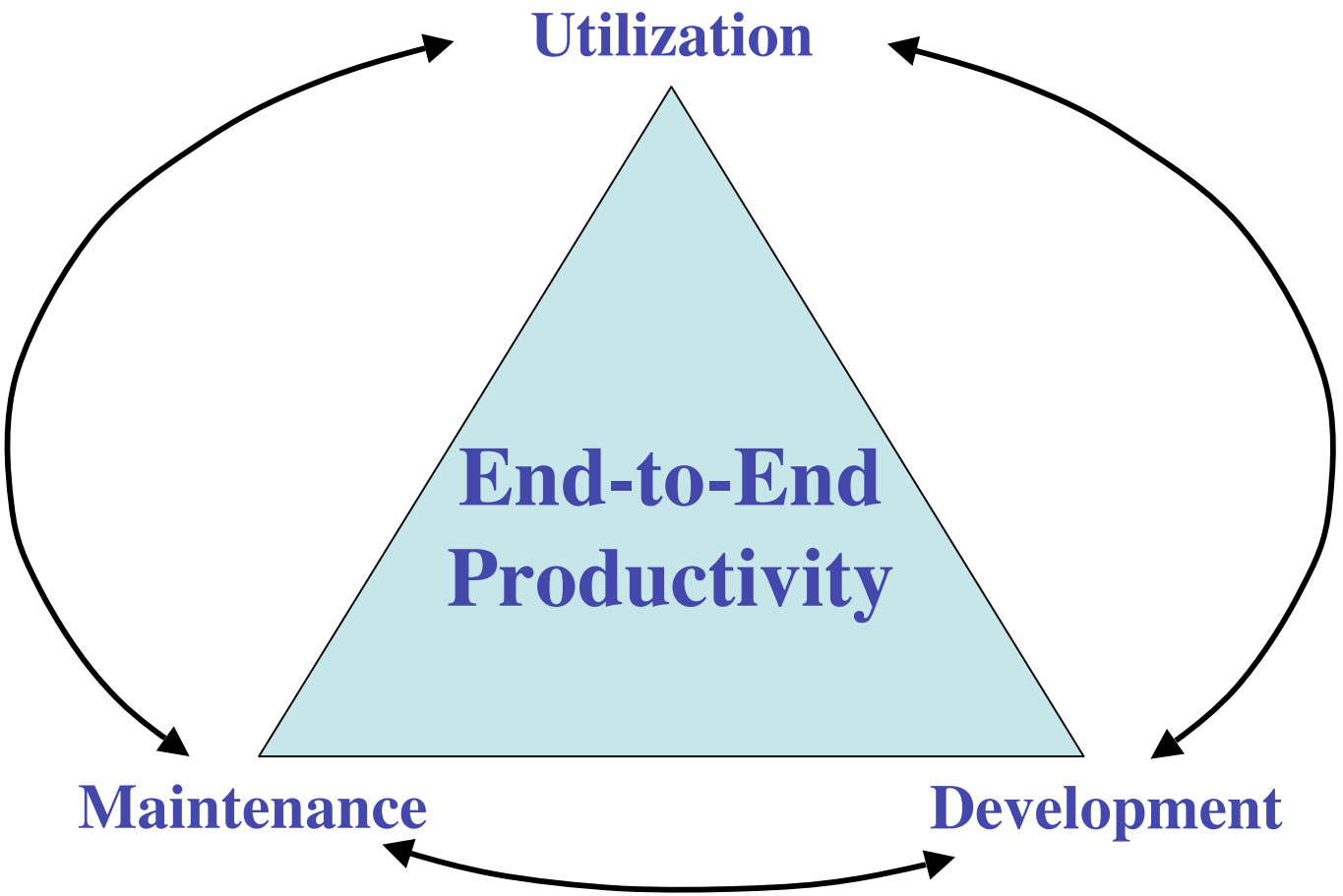
SDTPC Workshop, August 1-2, 2007

jtd@lanl.gov

41 Million PE Hours on Purple, BG/L, and Red Storm (13% of Total FLOPS on Top 500 List)



Tools Needed to Support a Cycle of Developing, Maintaining, Utilizing Codes



Development Tools: Scientific Code is Always in "Development" -- Even "Production" Code

- Programming environments
  - Languages and compilers (e.g., Fortran 77/90, C, C++)
  - Programming models (e.g., MPI, shmem)
  - Support over the application's life-cycle (i.e., 5-10 years in initial development and 5-20 years in production)
- Static correctness checking
  - Memory (e.g. Valgrind)
  - Parallelism (e.g. thread-safe checking)
- Optimization and tuning
  - Compilers that create fast code, not bugs (e.g. DEC Fortran)
  - Low overhead parallel profiling
  - Prediction and modeling

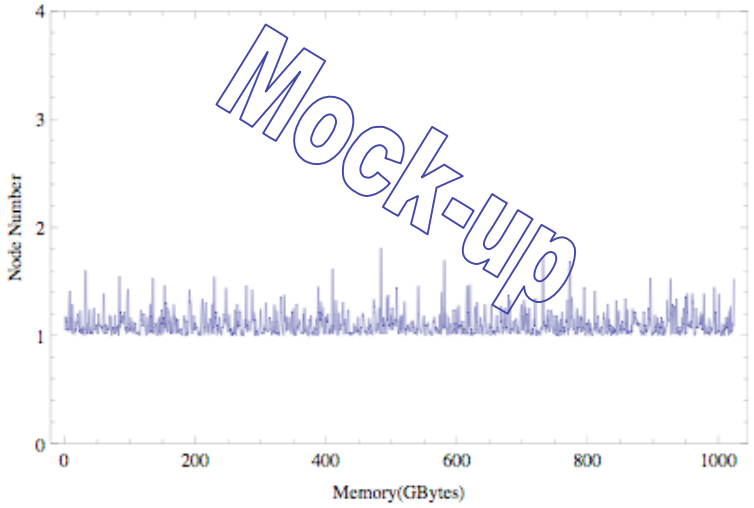
Maintenance Tools: Quickly Fix Existing Bugs without Creating New Bugs

- Configuration management
  - Version tracking and checkout
  - Code coverage
  - Regression testing
- Debugging
  - Comprehensive for small numbers of threads (i.e., 100's of PEs)
  - Lightweight for massively parallel (i.e., 1000's of PEs)
  - Low overhead for memory debugging
- Analysis Tools
  - Graphical description of peta-data (e.g., gnuplot, xdiff)
  - Analytical discovery (e.g., Matlab, Mathematica)

Utilization Tools: Facilitate Productivity by Streamlining and Automating the Workflow

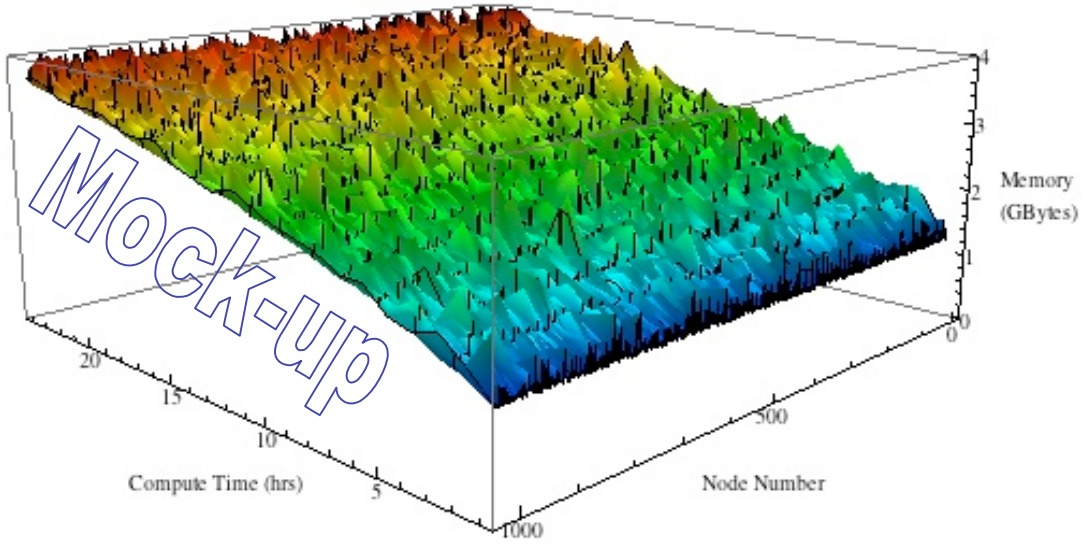
- Resource management
  - Job scheduling with dependencies
  - Memory usage information available to the application
  - Need exit information returned through launcher and scheduler
- Application throughput -- “users runs scripts, not apps”
  - Full Posix and scripting support (e.g., python, crontab)
  - Fast and scalable parallel I/O for checkpointing
  - Monitoring and automated task migration/restart
- Archival Storage
  - Persistent and fault-tolerant stores and retrieves
  - Handles a few big files or many small files
  - Fast metadata for thousands of files

Example of a Useful Tool: Graphical Display of per Node Memory Footprint for a 1024 Node Job



Visualize a snapshot of the job's memory footprint …

… or its complete time history

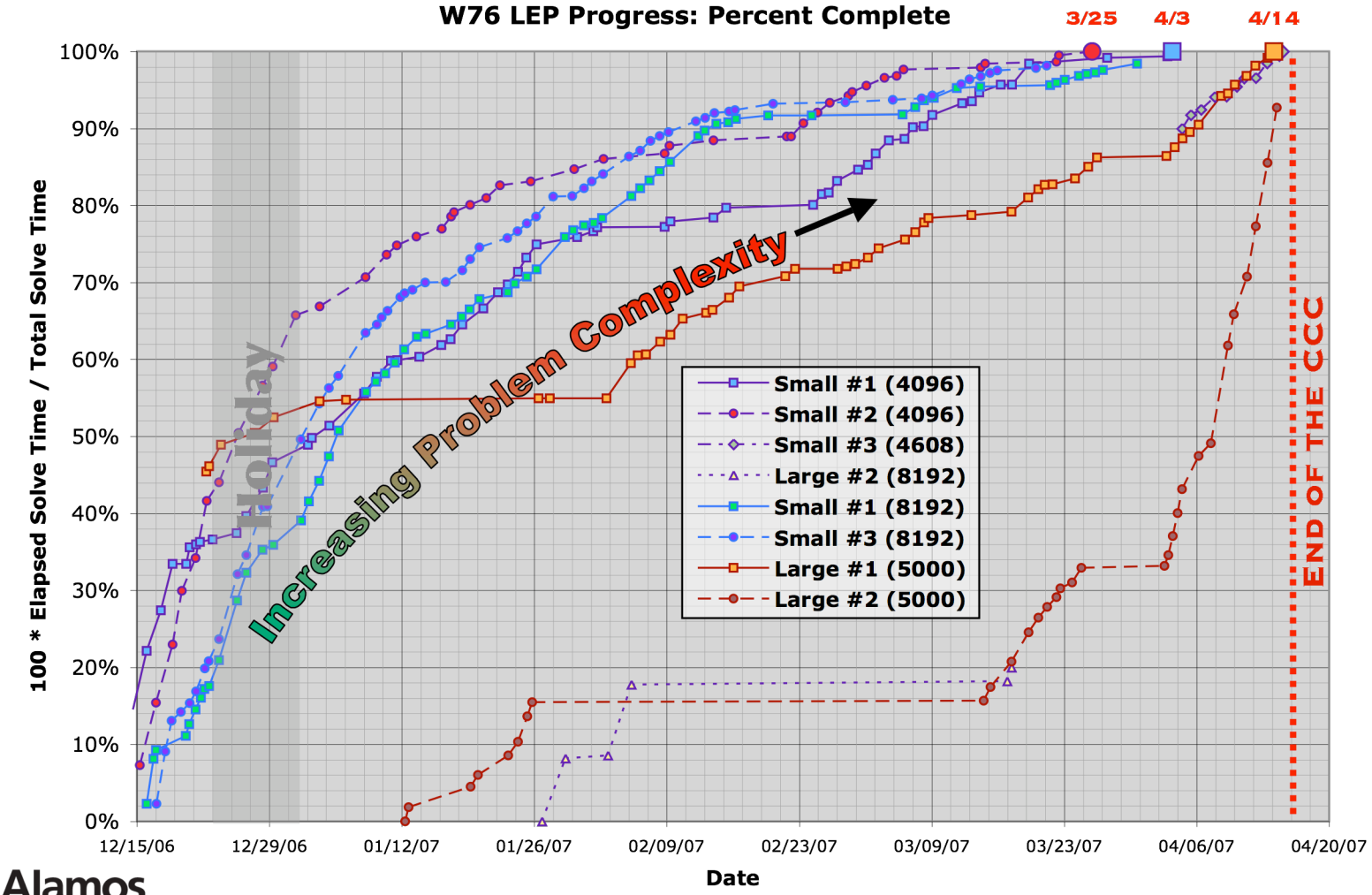


Los Alamos NATIONAL LABORATORY EST.1943
Operated by the Los Alamos National Security, LLC for the DOE/NNSA
LA-UR-07-5148
NNSA

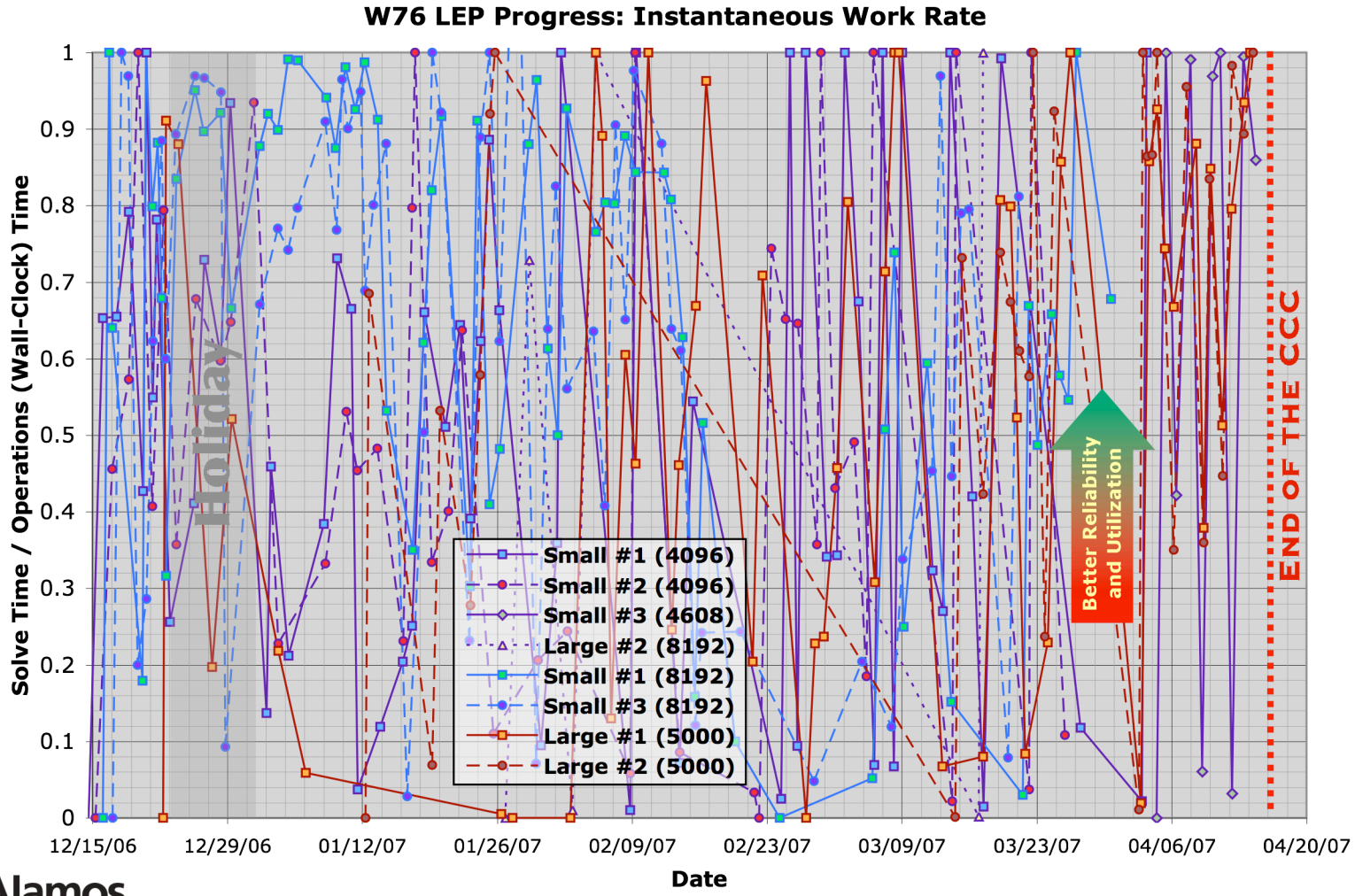
A Few More General Observations and Issues Regarding Tools

- Third-party complications
  - Lack of diversity -- there may not be lot of tool options out there
  - Per PE licensing can be *expensive* when computing at scale
- Support and maintenance of the tools themselves
  - Maintenance is important because tools may break frequently in environments at scale
  - Must be a sufficient knowledge base or tools will not get used
- Extreme scale computing ⇒ unique requirements
  - Developers and users have needs for tools that commercial HPC may have little or no incentive to provide (e.g., resilience)
  - Tools in vogue with the broader HPC community may be of little value for *our* users (e.g., checkpointing to memory)

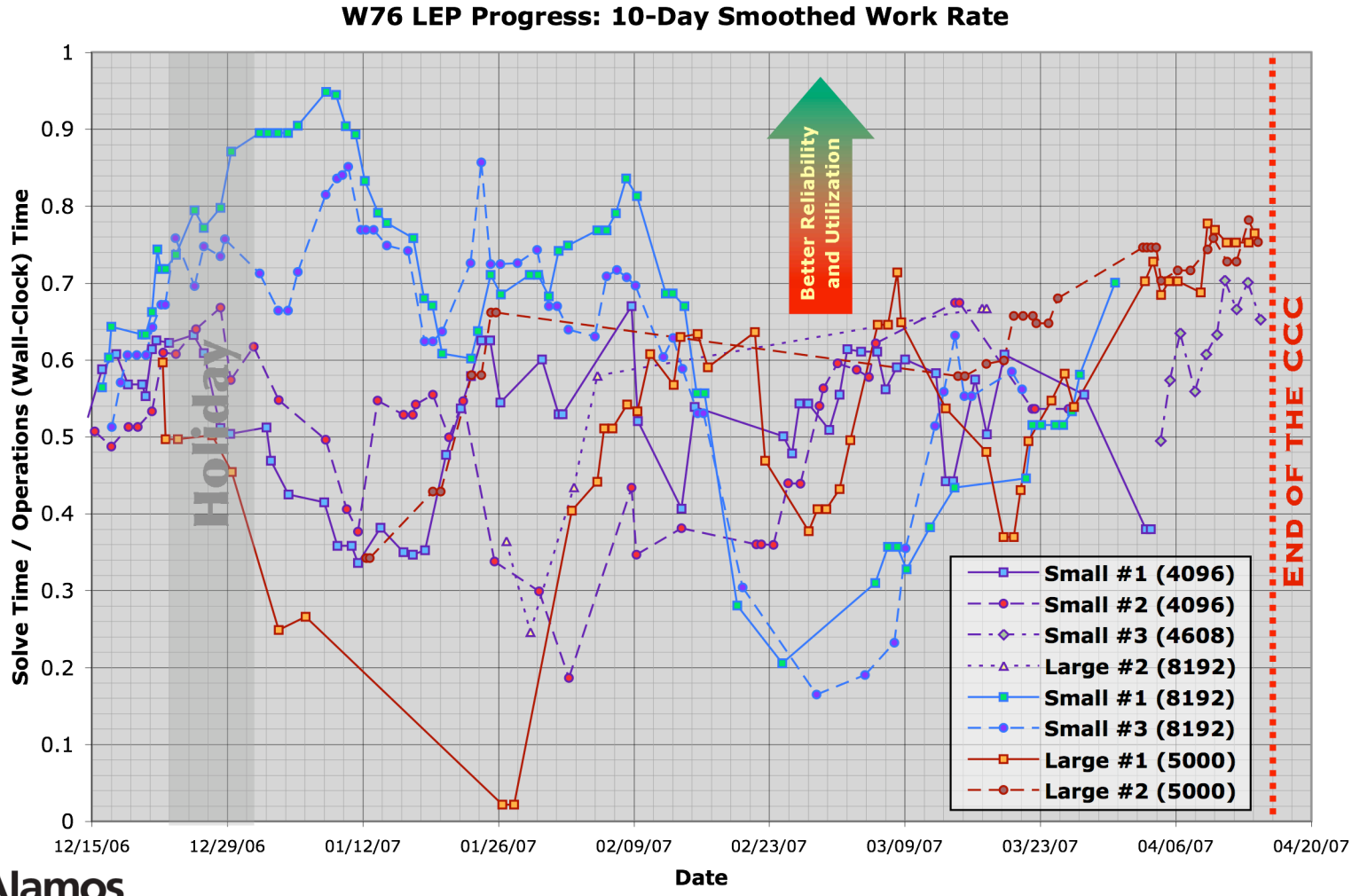
Monitoring Jobs Provides Data that Can Inform Code Development and Maintenance



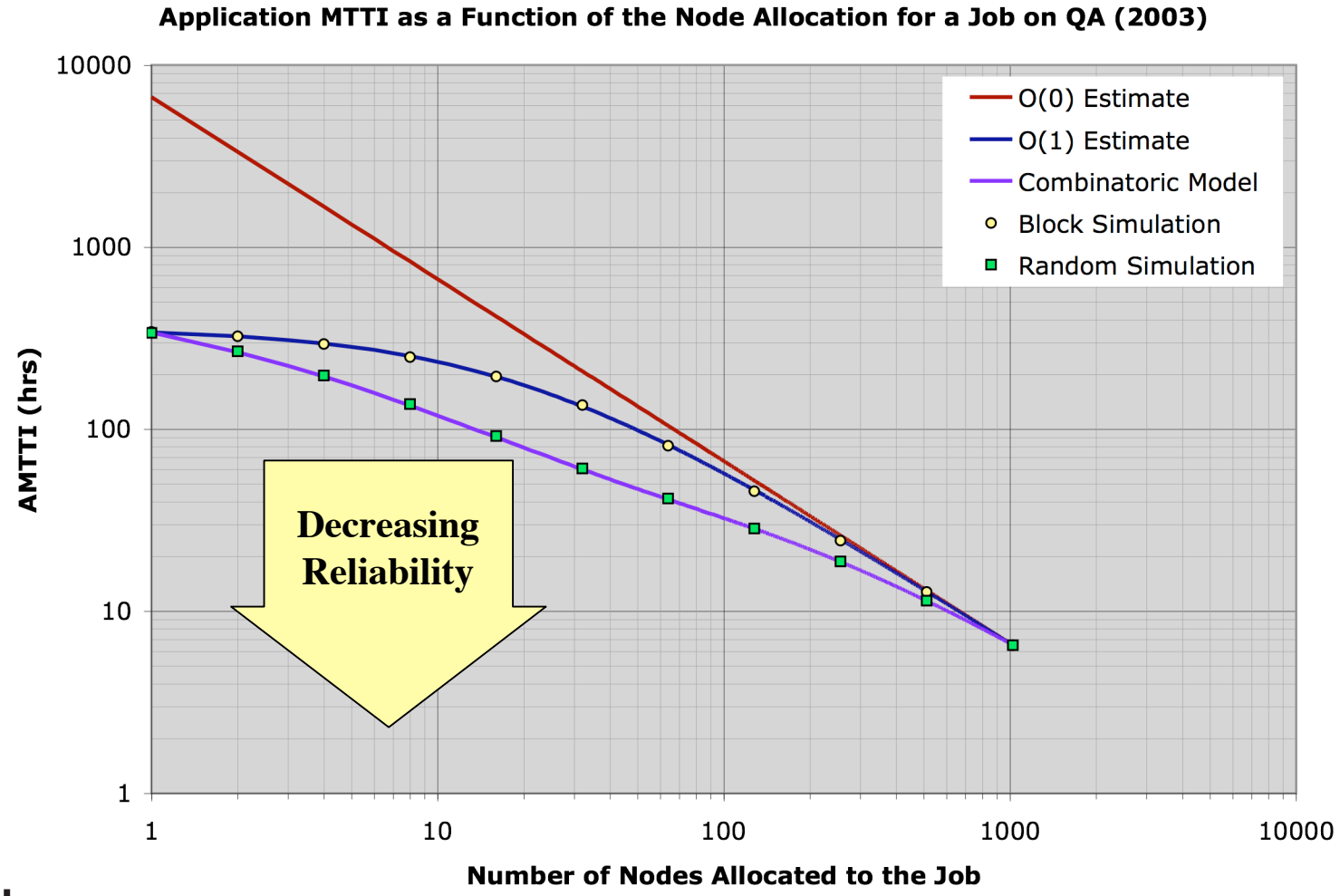
For Large Production Runs the Work Rate Can Be All Over The Map (~20 Job Restarts per Day)



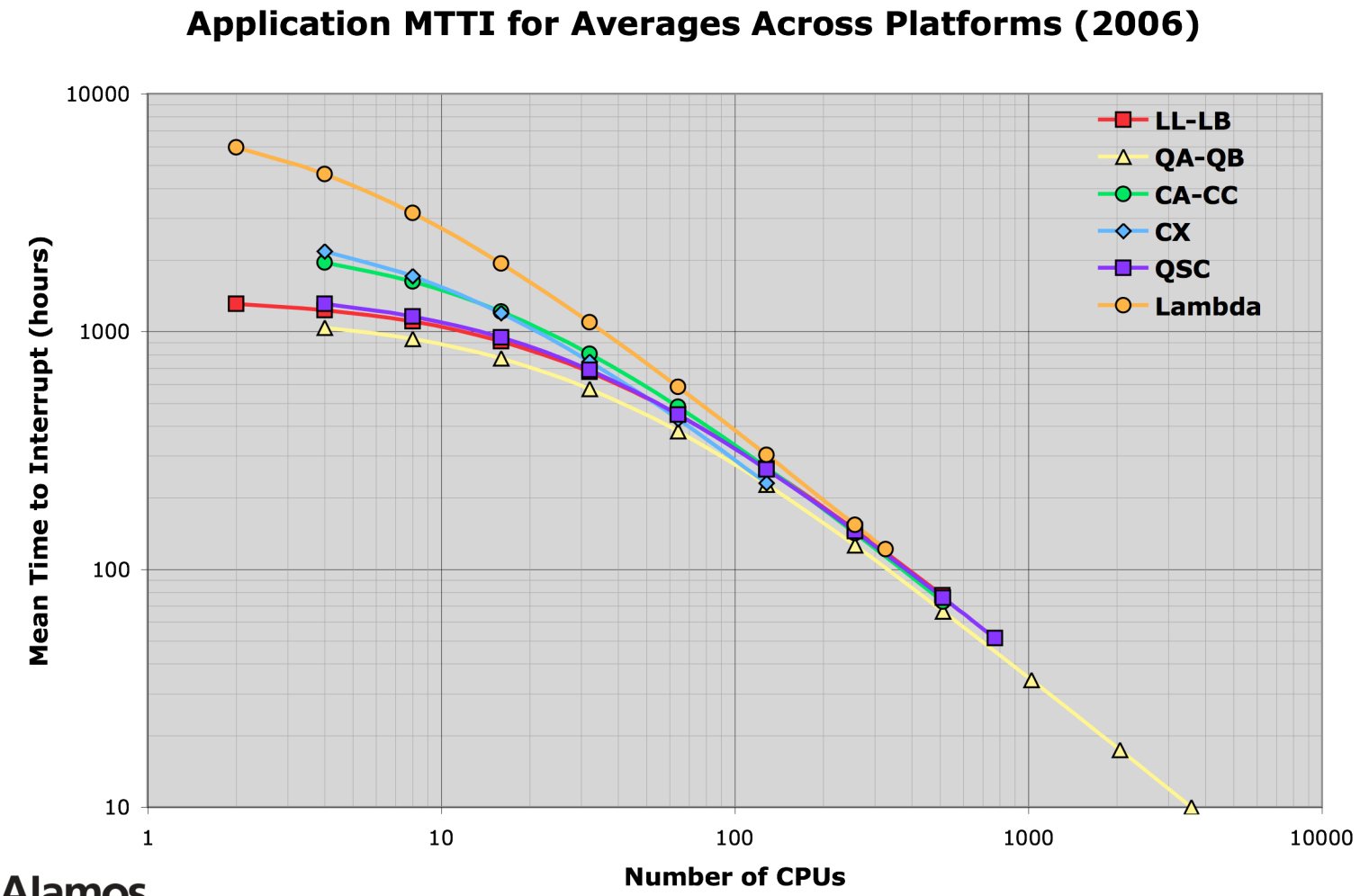
Smoothing the Data Gives Insight Into How Well the Platforms and Codes are Performing



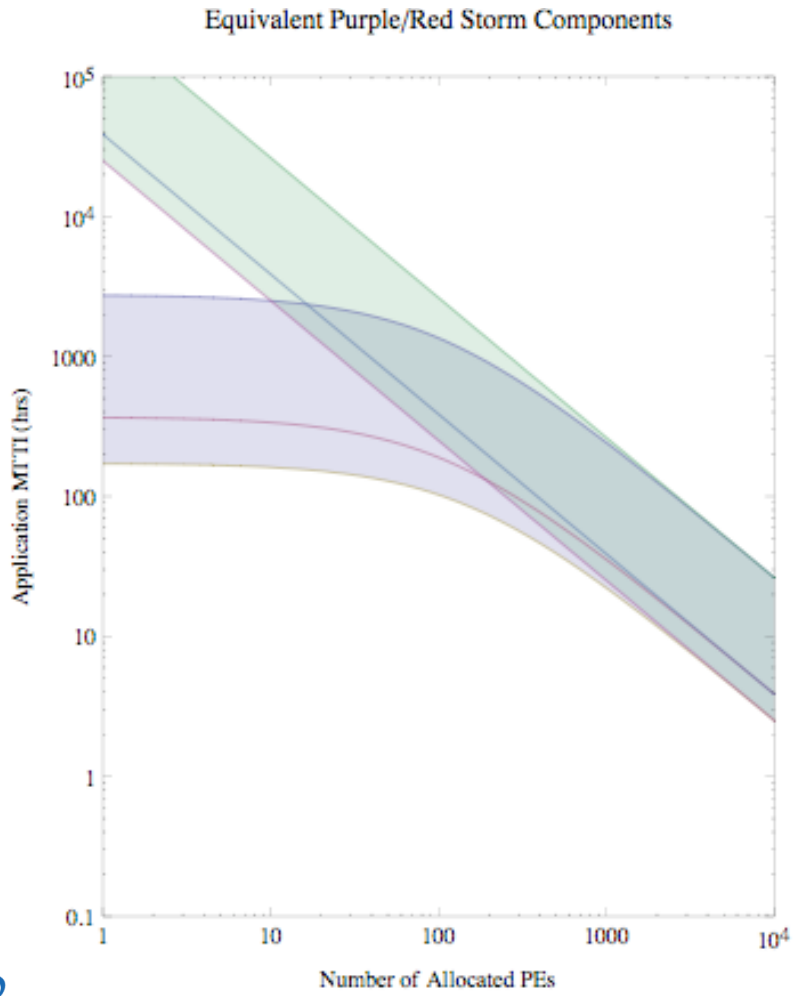
Application MTTI Is Much Worse Than Inversely Proportional to System MTTI

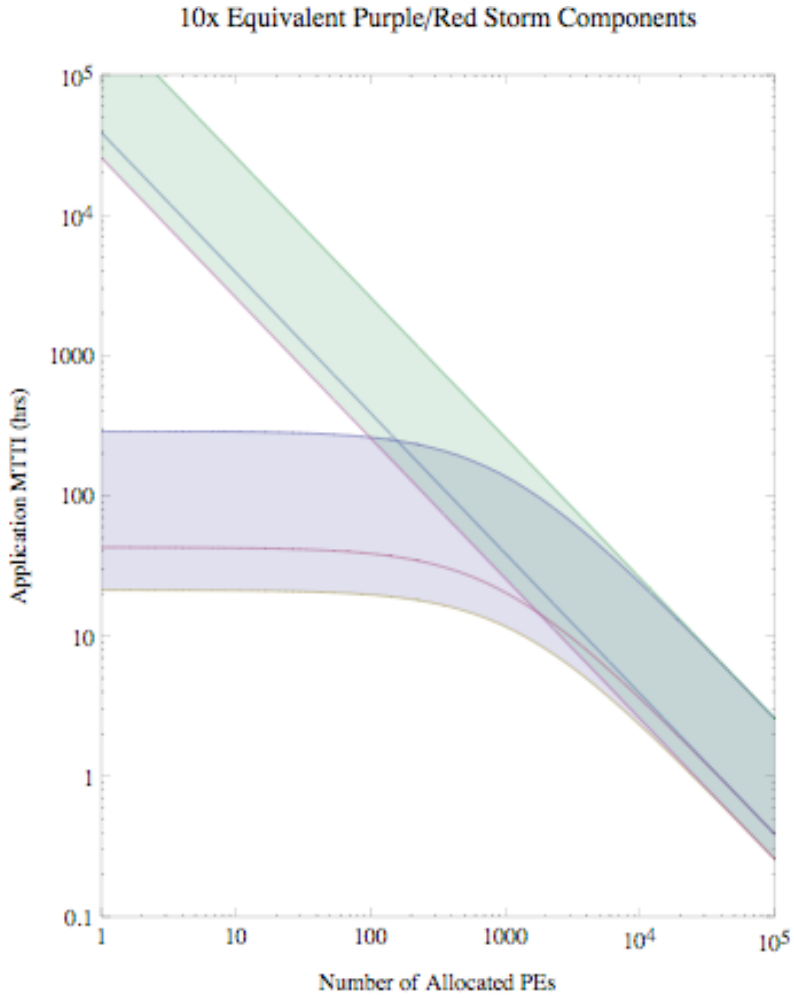


AMTTI Data Collected From 21 Different LANL Platforms Show Remarkably Similar Trends

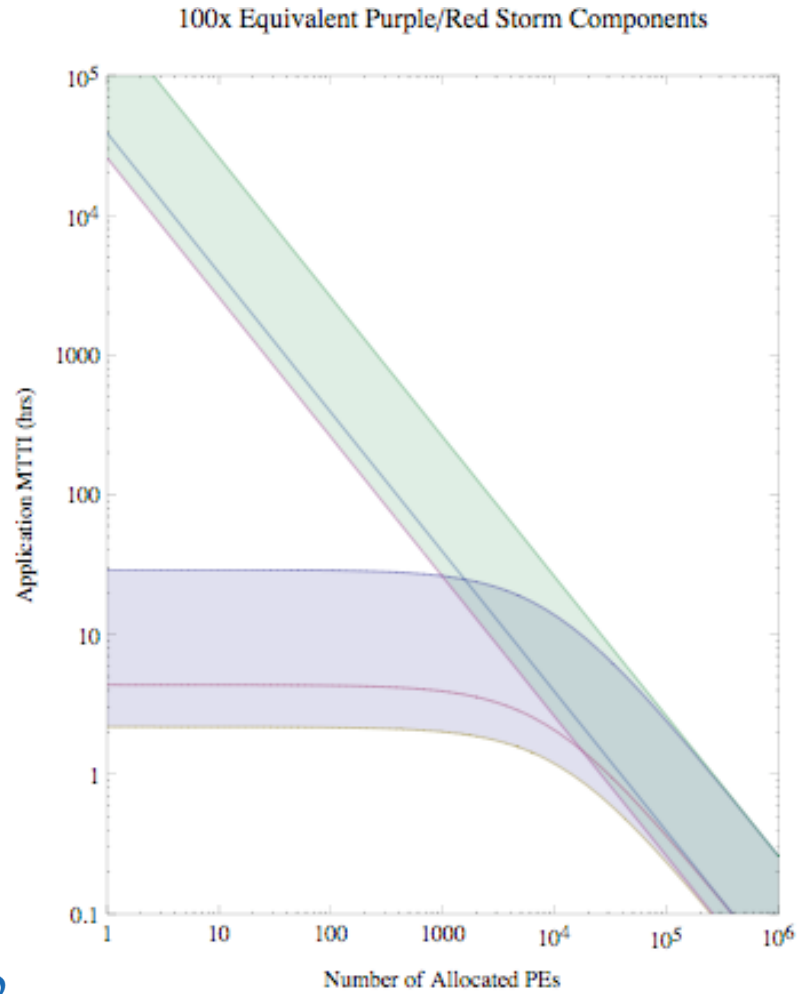


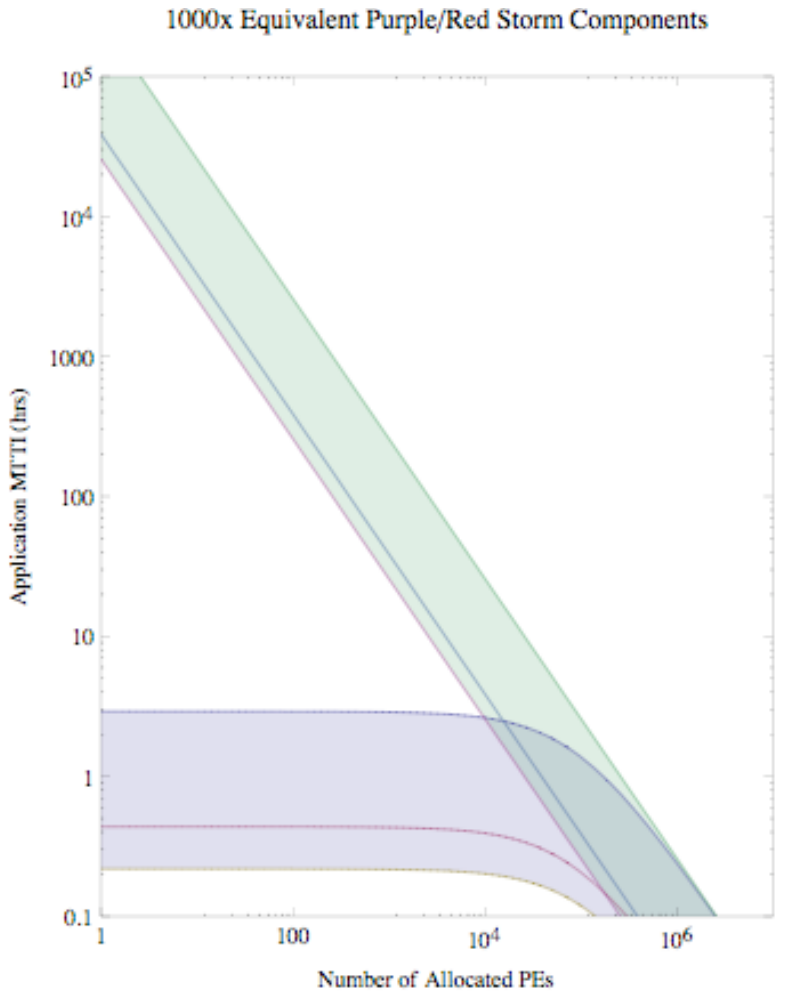
Application MTTI vs. Increasing Component Count: Projections Based on Available Data



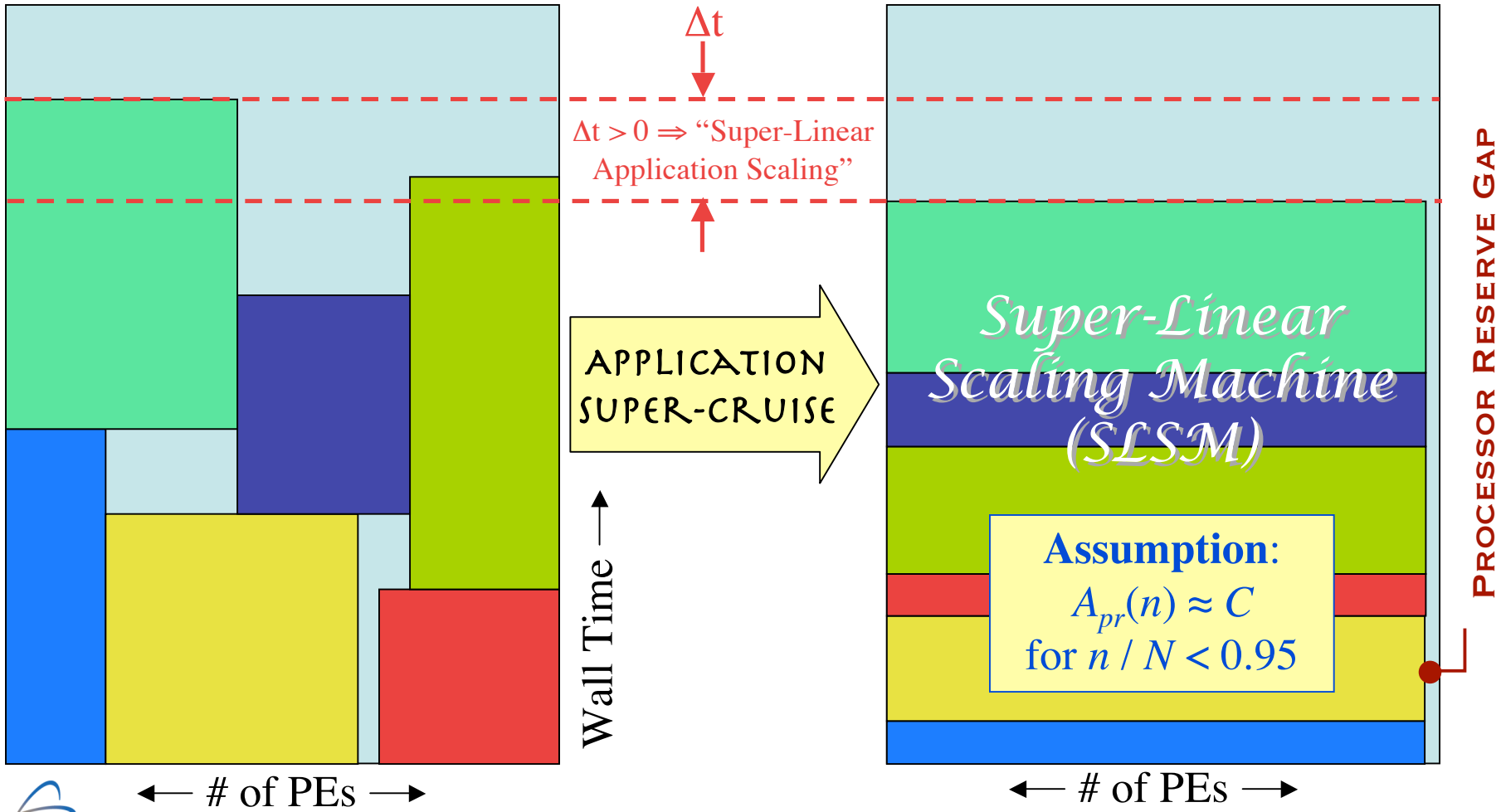


As Componen Count Continues to Increase Even Small Jobs May Become Unreliable





New Paradigm*: Schedule Jobs Serially on Available Processors to Increase Throughput



* Daly, *Los Alamos Technical Report* (LA-UR-07-2594)

Platform Reliability Impacts Selection of Tools and Strategies for Extreme-Scale Computing

	High Reliability	Low Reliability
Correctness	Static Checking	Dynamic Checking and Data Integrity
Resource Management	Heterogeneous, Many-Job Scheduling	Homogeneous, Multi-Job Resource Aware Scheduling
Application Throughput	Application Tuning and Optimization	Job Monitoring Data Integrity Automatic Restart And Migration

A Summary of Productivity Enhancing Tools

- Today's Wish-List
  - Programming environment supported over life-cycle of codes
  - Parallel performance analysis tools
  - Lightweight massively parallel debugging
  - Low overhead memory debugging
  - Memory usage available via application interface
  - Fire-and-forget archival storage with fast metadata access
- Tomorrow's Wish-List
  - Application progress monitoring and automated restart
  - Resource aware job scheduling and task migration
  - Runtime protection against data corruption