

Running Applications Successfully at Extreme Scale: What is Needed?

John T. Daly

**ASCR Computer Science Research
Principal Investigators Meeting
Denver, March 31 - April 2, 2008**

The difference between “success” and “failure” is sometimes only a matter of perspective

I have not failed. I’ve just found 10,000 ways that it won’t work.

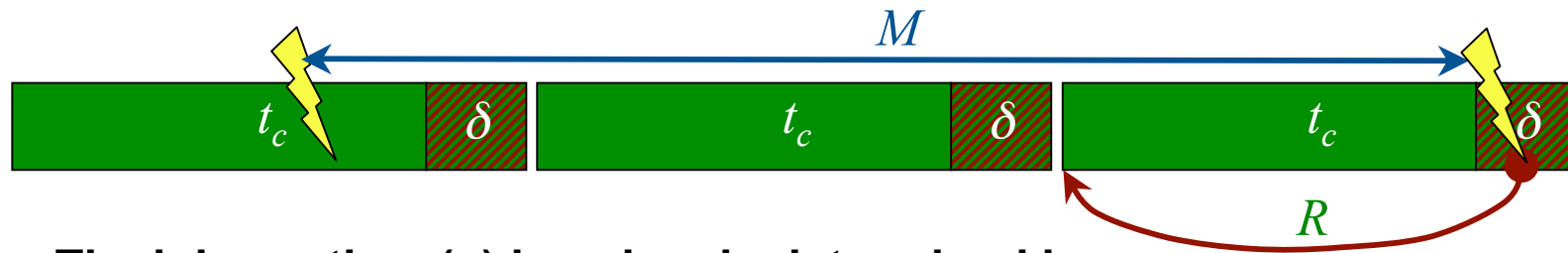
-- attributed to Thomas A. Edison

Success from the perspective of the application is a “correct” solution in a “timely” fashion

- **Correctness: How do I know the system executed my code correctly?**
 - Fault-Tolerance -- handling detectable but uncorrectable errors
 - Recovery oriented techniques based on the last known “good” state
 - Resilience -- protecting against undetectable errors
 - Low overhead, *in situ* verification of data and calculations

- **Timeliness: How long am I willing to wait for the solution?**
 - Performance -- fast execution in an interrupt-free environment
 - Coding for increasingly complex memory hierarchies
 - Scaling in a many-core heterogeneous environment
 - Resilience -- forward progress in an interrupt-rich environment
 - Identify system failure modes
 - Analyze the impact of failures on the application
 - Keep the application running

Solve Time vs. Run Time*: Quantifying the effectiveness of recovery oriented methods for fault-tolerance

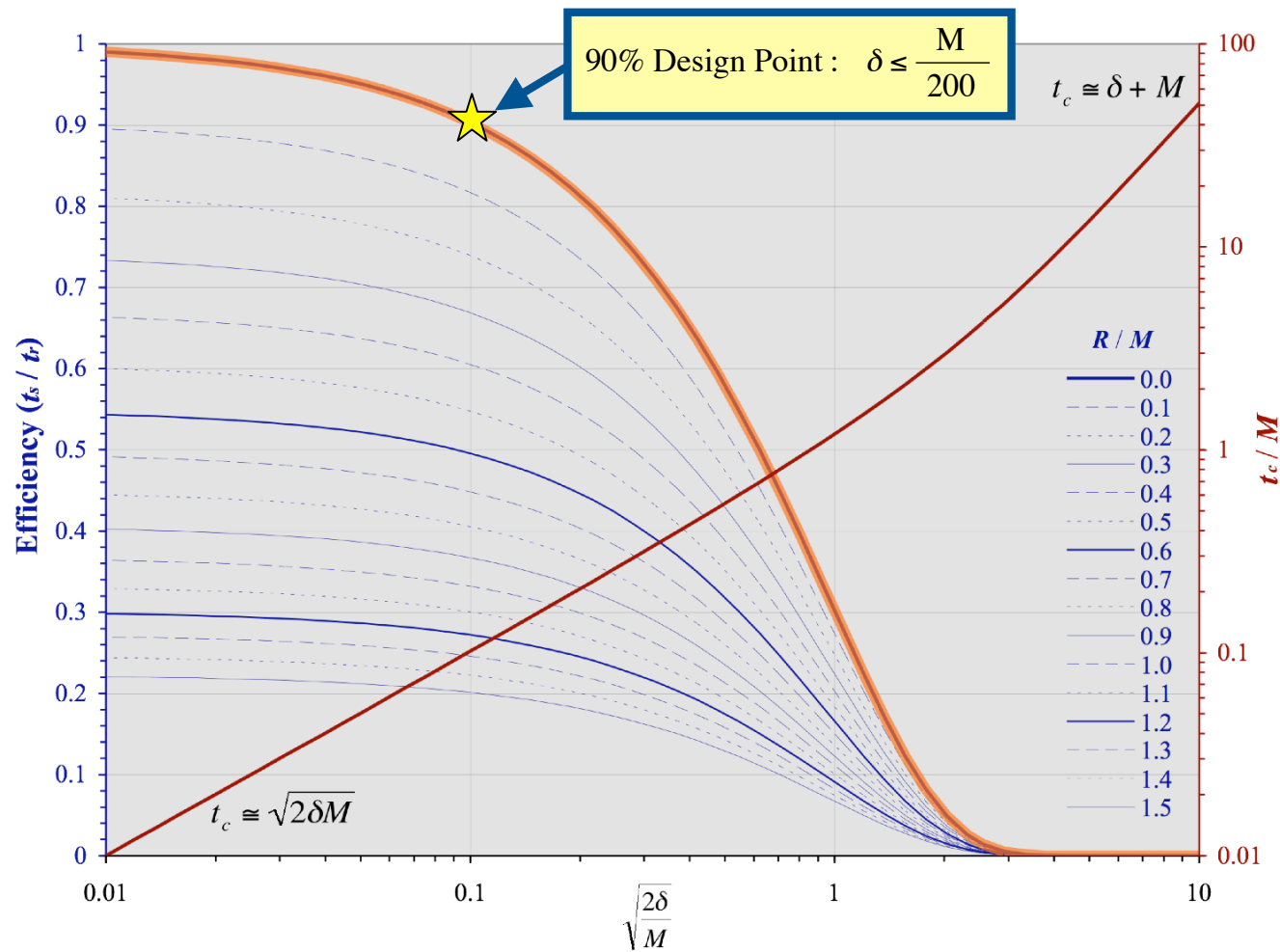


- **The job run time (t_r) is uniquely determined by**
 - Solve time (t_s) -- time required to complete an uninterrupted job
 - Application MTTFE (M) -- expected time before the next interrupt
 - Dump Time (δ) -- overhead required to save a known good state
 - Checkpoint Interval (t_c) -- time elapsed between saving states
 - Restart Overhead (R) -- time to restart from a previous saved state
- **Assuming that failure interarrival times are distributed exponentially, the application run time will be**

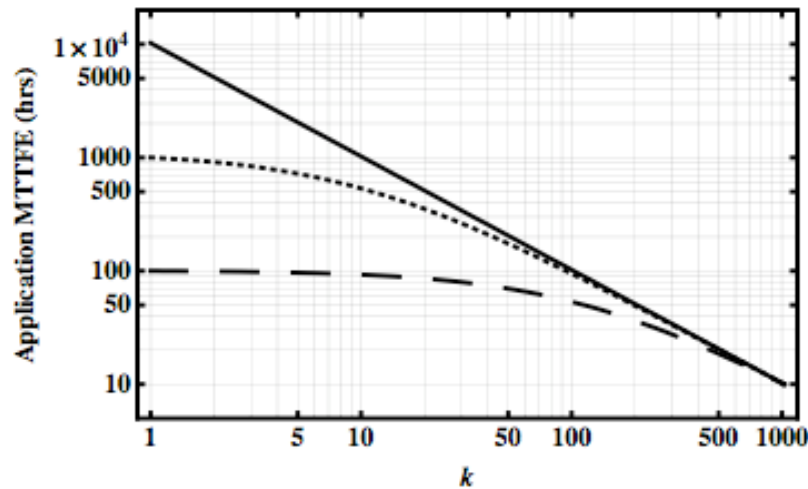
$$t_r = M e^{\frac{R}{M}} \left(e^{\frac{t_c}{M}} - 1 \right) \frac{t_s}{t_c - \delta} \quad \text{for } \delta \ll t_s$$

* J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps", Future Generation Computer Systems 22 (2006) 300-312

Efficiency of recovery oriented fault tolerance is limited by reliability and overhead required to save job states

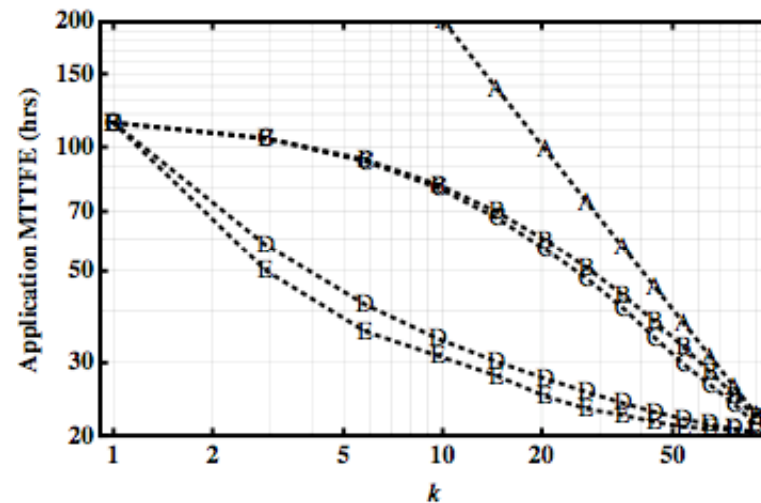


Application mean time to fatal error* is a better metric than system MTBF for quantifying the user's experience



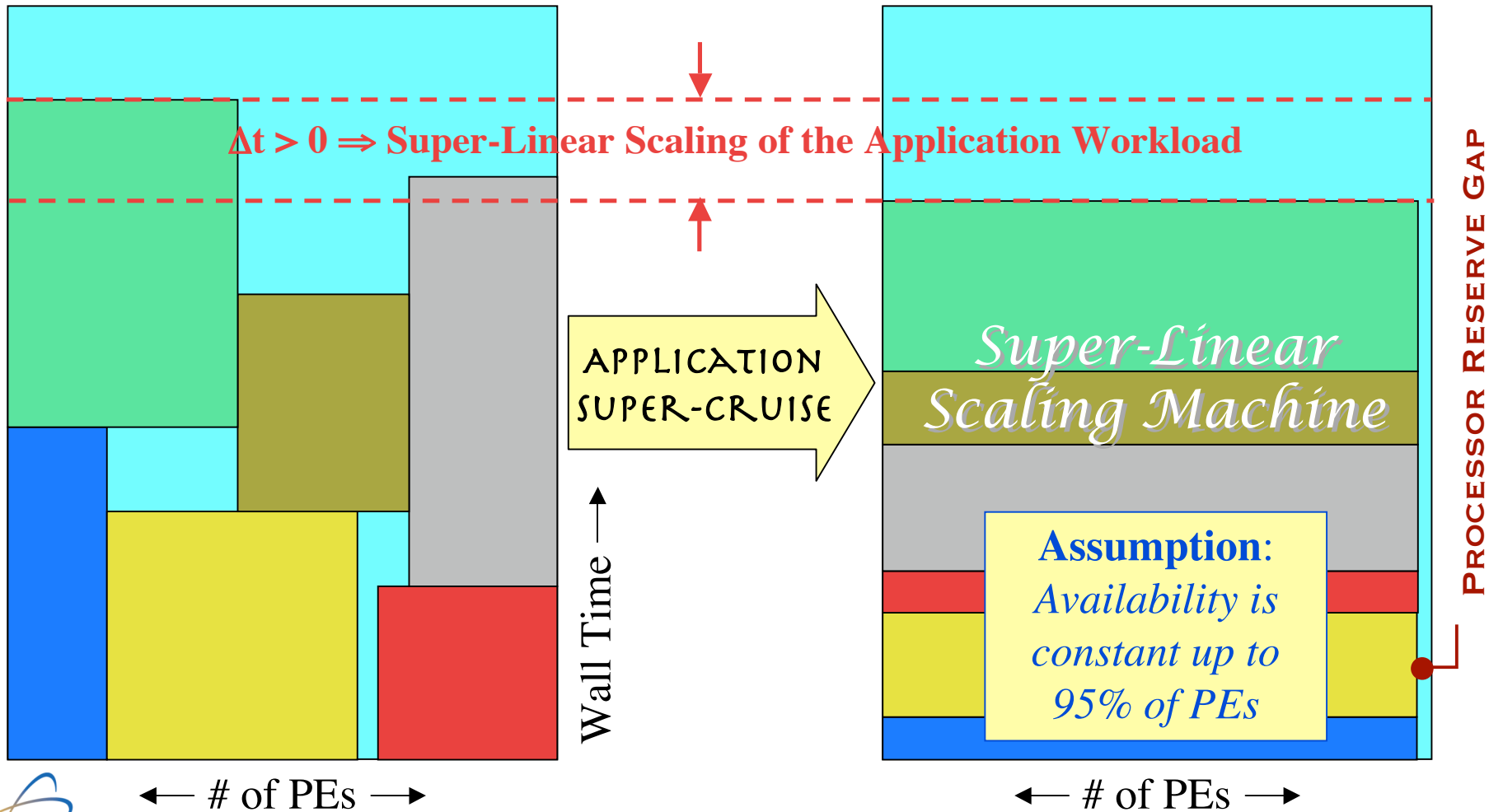
First order approximation of application mean time to fatal error demonstrates super-linear per processor reliability scaling

- A -- Inverse Proportionality
- B -- First Order Approximation
- C -- Exact (Contiguous Nodes)
- D -- Exact (Random Nodes)
- E -- Exact (Worst Case Nodes)
- k -- number of processors



* Daly, Pritchett-Sheats, and Michalak, "Application MTTFE vs. platform MTBF: a fresh perspective on system reliability and application throughput for computations at scale", proceedings of Workshop on Resilience in HPC, CCGrid (2008) *forthcoming*

Paradox of recovery based fault-tolerance: as reliability plateaus the workload can scale *super-linearly*

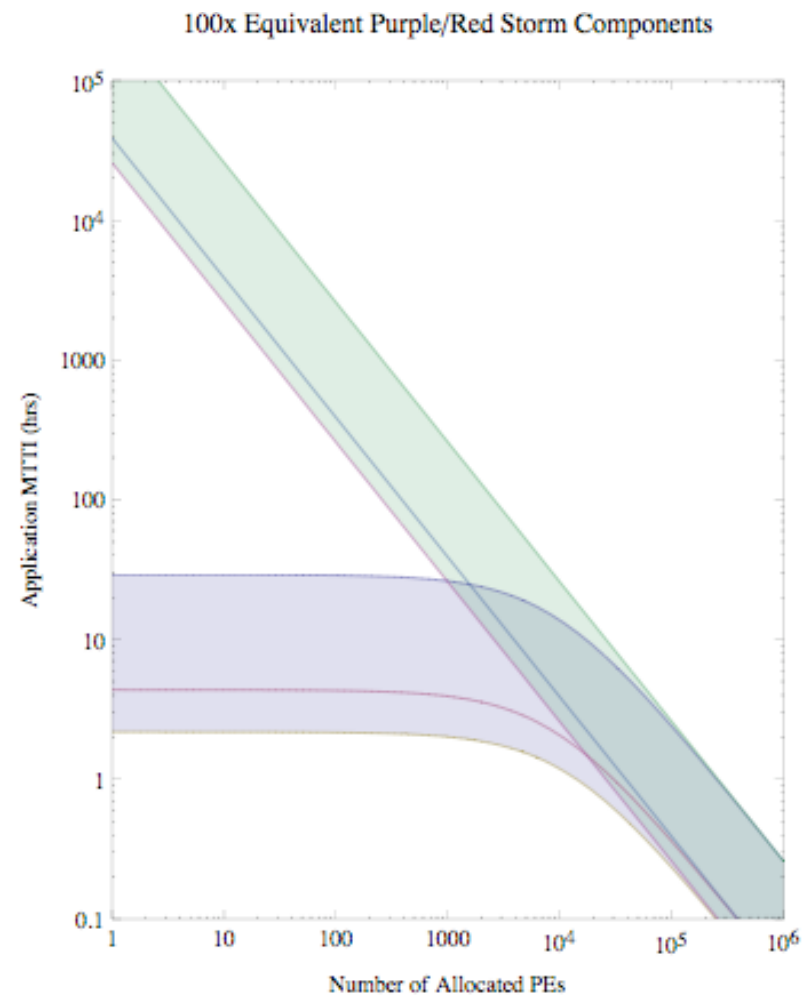
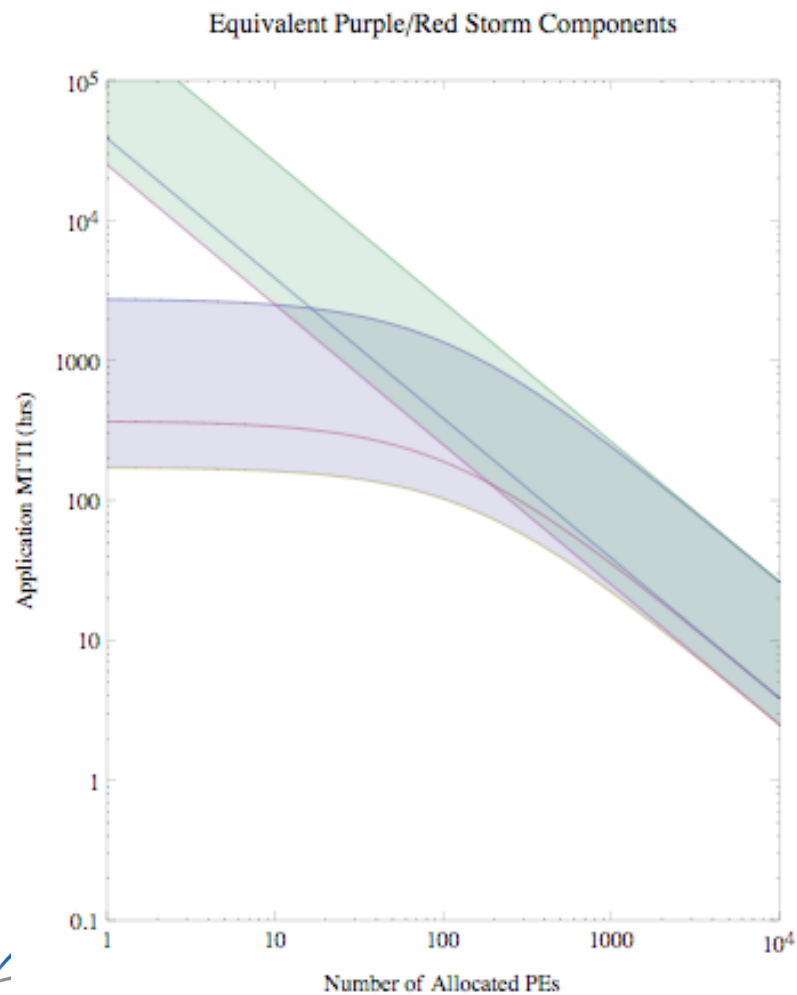


Automated job monitoring and restart* is a necessity for running big jobs on existing large scale systems

- **Running 41 million PE hours over 16 weeks on Red Storm, Purple, and BG/L, it was typical to restart applications 10-20 times per day**
 - System hardware (nodes, disk controllers, interconnect)
 - System software (job scheduler, queue limits, file system)
 - Application errors
 - Human error
- **The applications are kept running by an automated job monitoring and restart script**
 - Tracks progress by monitoring output directories and killing hung jobs
 - Resubmits interrupted jobs but bails out for recurring failures
 - Uses cron to restart itself even after a full system reboot
- **Job monitoring provides a cheap, portable, and low-overhead means for gathering data to measure AMTTFE**

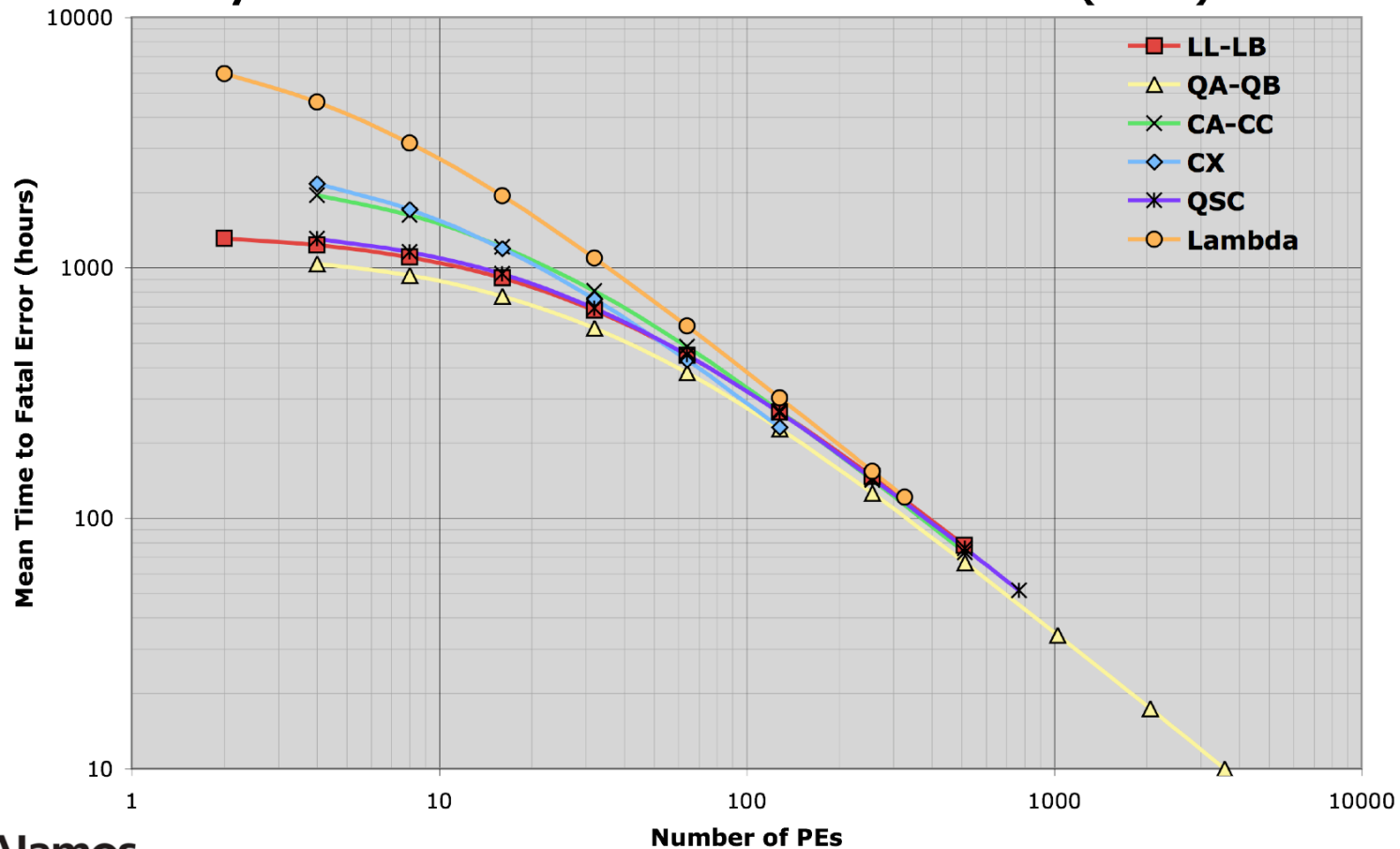
* J. T. Daly, "Facilitating High-Throughput ASC Calculations",
ADTSC Nuclear Weapons Highlights '07

BUT: Extrapolating reliability data indicates even small jobs will experience frequent interrupts at extreme scale



Analysis of 140,000 interrupt events confirms that per PE reliability is extremely consistent across platforms

Summary of Application Reliability as Measured from System Data Across 21 Los Alamos Platforms (2006)



Recovery based methods are not adequate to guarantee application success as systems continue to scale

■ Trend of Decreasing System Reliability

- System Software: complexity is increasing to accommodate heterogeneous, many-core architectures
- System Hardware: component counts are increasing, but per component reliability remains the same

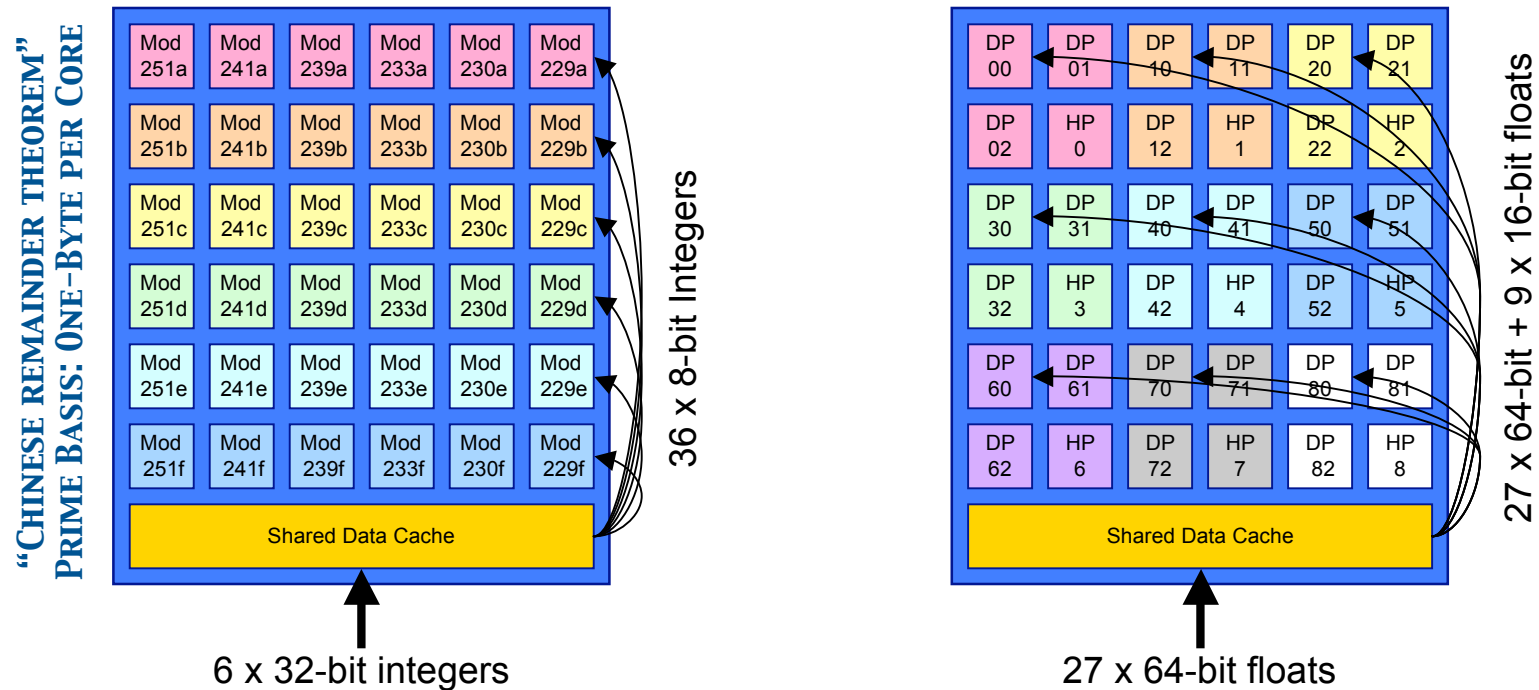
■ Unaccounted for Downtimes -- “3x Problem”

- Preliminary investigations at Los Alamos indicate that as much as two-thirds of application “downtime” is not accounted for in the system monitoring data
 - system software interrupts (est. 50% of total interrupts)
 - common-cause failures that impact multiple applications (e.g. file systems)
- **Need:** methods of monitoring system reliability from the application’s perspective

■ Resource Overhead for Fault-Tolerance -- “2x Problem”

- Most solutions require up to two-fold “overhead” of extra or redundant resources
- Los Alamos systems have no “extra” resource -- applications use everything
- **Need:** solutions that minimize “overhead” required to successfully complete work

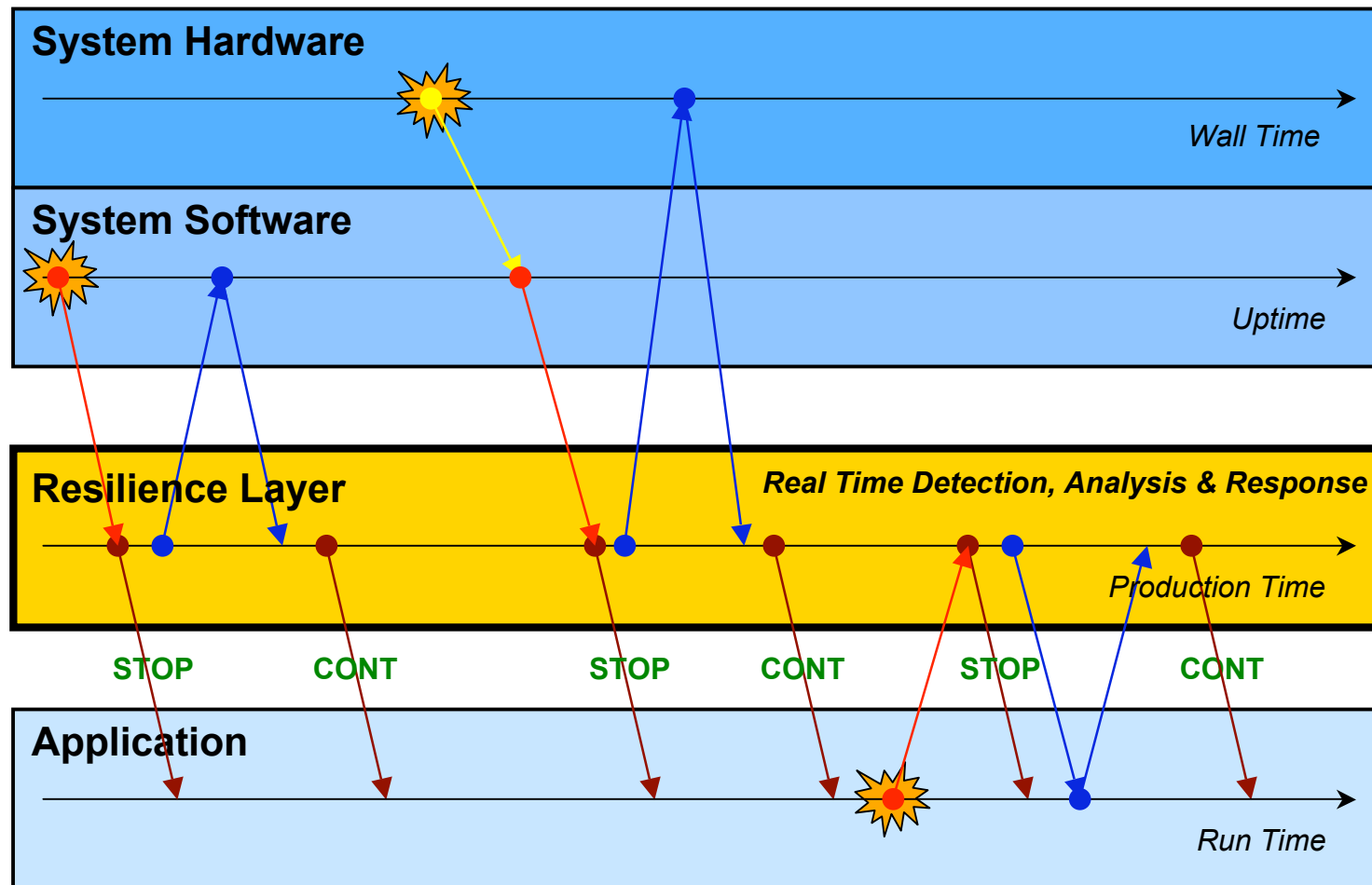
EXAMPLE: Resilience for *correctness* implemented on a many-core chip by using byte and half-precision ops



4 integer ops per 6 cores
50% overhead
Full correctness checking

3 floating ops per 4 cores
33% overhead
Partial correctness checking

EXAMPLE: Resilience for *timeliness* works by isolating the application from failure generated interrupts



Conclusion: Resilience will be essential for correctness and timeliness of applications running at extreme scale

- Reliability is poorly estimated by available system data which needs to be much better reconciled to the application's perspective
- Common cause failures cause an application mean time to fatal error (AMTTFE) that is less than inversely proportional to system MTBF for small jobs
- As systems grow in size and complexity, they will continue to become less reliable for jobs of all sizes
- Current strategies for scheduling and running applications on platforms with relatively large numbers of interrupts are sub-optimal
- Resilience can improve job throughput and provide valuable reliability data with tools that monitor job progress, protect against errors, and prevent unnecessary interrupts