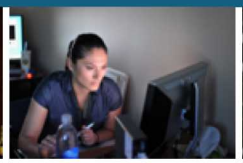
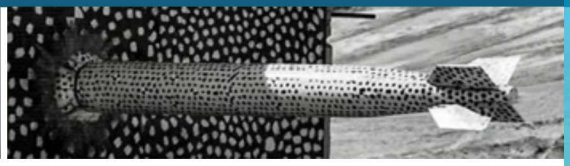
SAND2019-3281C

Sandia National Laboratories

Resilience for Exascale Computing and Beyond





PRESENTED BY

Keita Teranishi

U.S. DEPARTMENT OF ENERGY | NNSA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Resilience is necessary!

Applications should behave as expected.

- Complete all iterations
- Produce correct output
- Finish all computations within a given time window

Hardware reliability is improving

- How about system reliability?
- How about the rate of successful application execution?
- Is there any justifiable data to show the improvement of the reliability of systems and applications?

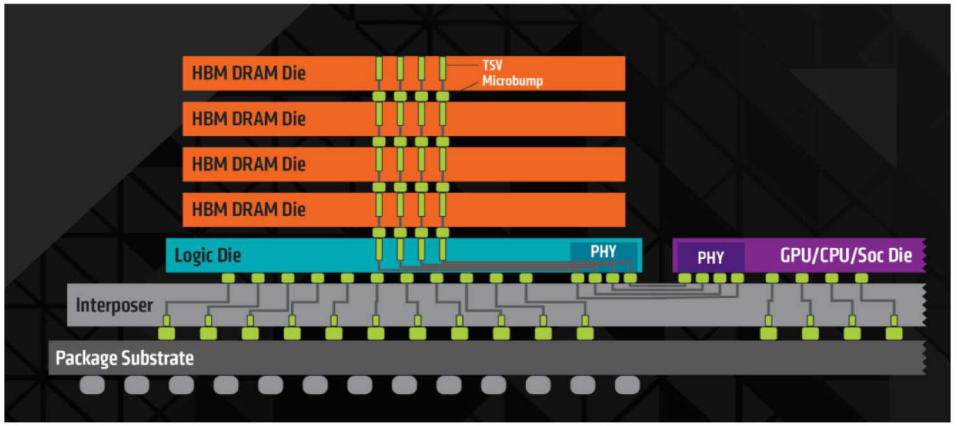
Checkpoint/Restart (C/R) with the latest IO subsystem technology can handle **some types** of application failures

- New types of failures

Different components have different reliability

HBM	DDR
High Bandwidth	Low Bandwidth
Low Reliability	High Reliability

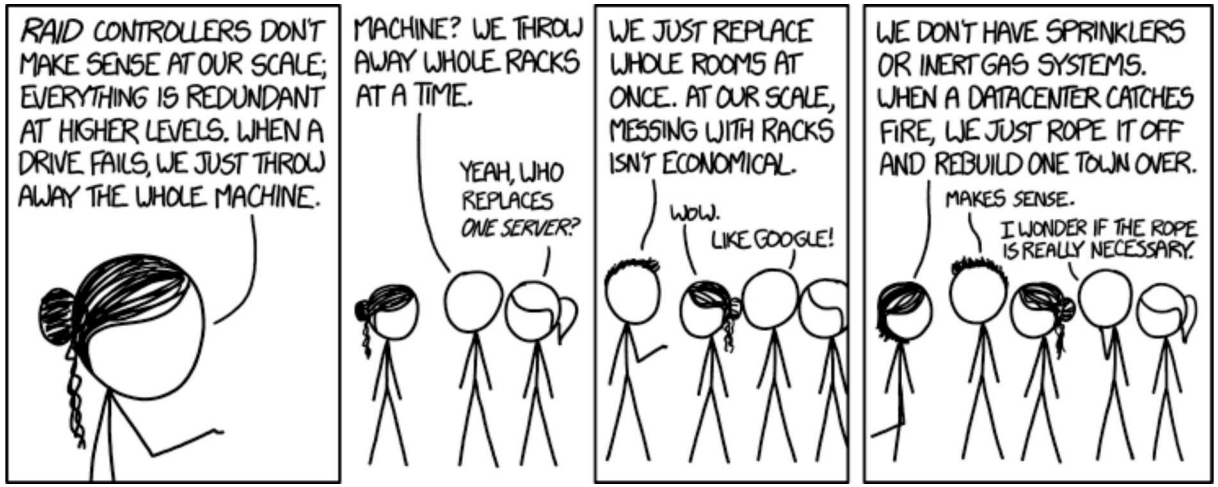
Courtesy: AMD and UCSD



Different hardware components have different reliability

- Different reliability per Components
  - CPU, GPU, HBM, DDR, NVRAM, Network
- Even for the same component type, reliability differs among different manufacturers
- How to manage complex interaction between components?
  - Are errors and failures contained within a component?
  - Which software component manage these? Runtime, OS or Middleware**?**

Resilience is not achieved just through redundancy.



From the single application perspective, large scale HPC system already provide very good redundancy.

- The question is how to serve extra resources to individual applications while many other applications are sharing some system resources (network and IO)

New memory and IO technology exhibits complex tradeoffs of performance and reliability

Resilience is not patch-work solution

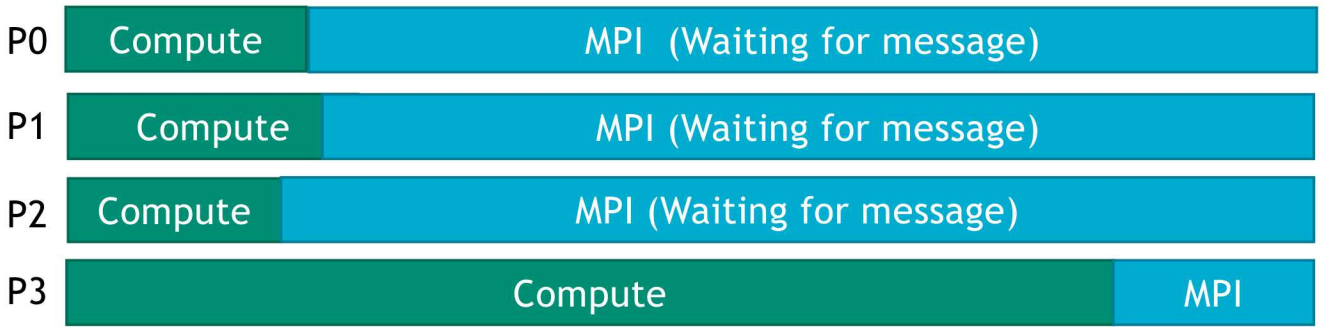
Adding checkpoint/restart involves extra coding:

- A couple of C/R library projects funded by ECP
  - Many library calls to bind application data to checkpoint storage
- No HPC programming models embrace resilience/fault tolerance.

Recovery involves coordination with runtime, OS, middleware and IO subsystems.

Enterprise distributed computing already embraces resilience in the programming model: e.g. RDD in Spark.

Resilience is essential for managing performance variability



Performance variability is a new type of system failure.

- Trinity at LANL experienced a 25x slowdown of a single compute node
  - Errors in a single bad DRAM module
  - Took a few days to find the problem
  - MPI did not report any errors
- Resulted in 25x application delays

C/R cannot handle this problem.

Is software resilient?

Anecdotes and some reports indicate that the majority of failures are caused by software rather than hardware.

- Network and Global file systems failing to handle corner case situations
- Bugs in application programs
  - User's mistake sometimes triggers failure in network and middleware
- Any software bugs manifested in an undeterministic manner
  - Application users see them as system failure

How to verify and quantify reliability/resilience of large scale HPC systems?

Chip manufacturers have established a method for verification and quantification of reliability and resilience

- Formal method verification

How about interaction between different hardware components?

- Any methodology to quantify resilience?

How about software?

- Traditional software testing methodology work for HPC?
- Software and hardware interaction is even more complex.

Solution 1: Embracing Resilience in Applications, Programming Model, and Runtime

Build programming models and libraries should embrace resileince and fault tolerance as well as performance portability

Examples:

- Resilience support by Kokkos
  - Data and computation abstraction by Kokkos includes unreliable computation and persistency of storage devices
- Application domain specific framework and scientific libraries that embrace resilience
  - Resilient Scientific Libraries (sparse linear system solvers) Effort by INRIA Bordeaux
  - Domain Specific Language with built-in fault tolerance capability in weather/climate simulation applications (ESCAPE2 project)
- Provide a better interface to OS, Middleware so that apps/runtime can see the status of the system
  - Better vertical integration for resilience
  - Alternatives for “abort” (fault oblivious computation)

Solution 2:Dynamic scheduling of computation and resource allocation

Programming model and middleware should support dynamic scheduling for resource allocation and task/job placement

- It's hard to imagine that 100% of compute nodes are occupied by applications.
- Middleware and runtime systems should provide extra resource to a job that is suffering from "local failures" if necessary

Examples:

- MPI-ULFM (Fenix), MPI ReInit
- Asynchronous Many Task Runtime

Solution 3:Quantify resilience and fault tolerance in system-level

The community needs a methodology of quantifying resilience and fault tolerance of computing systems beyond chips and individual hardware components

How to extra patterns and structure of computing systems?

- Resilient design patterns (by Engelmann) to extract the structure and composition of HPC resilience
- Containment Domains to accommodate hierarchical structure of computing systems

How to evaluate/quantify?

- Formal methods
- Specification language and tools to create resilience and fault tolerance abstractions
- Simulation with abstract computing systems