Fault Tolerance Challenges and Solutions

Presented by

Al Geist

Computer Science Research Group
Computer Science and Mathematics Division

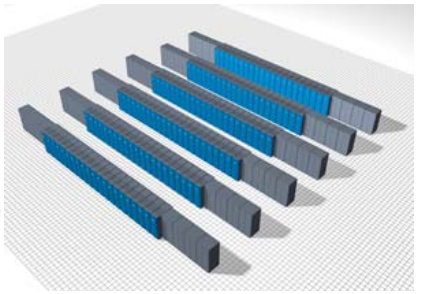
Research supported by the Department of Energy's Office of Science
Office of Advanced Scientific Computing Research

SC07

OAK RIDGE National Laboratory

Rapid growth in scale drives fault tolerance need

Example: ORNL LCF hardware roadmap



1 PF
(128 new cabinets)
175 TF

300 TF
(84 quad)
71 TF

2009

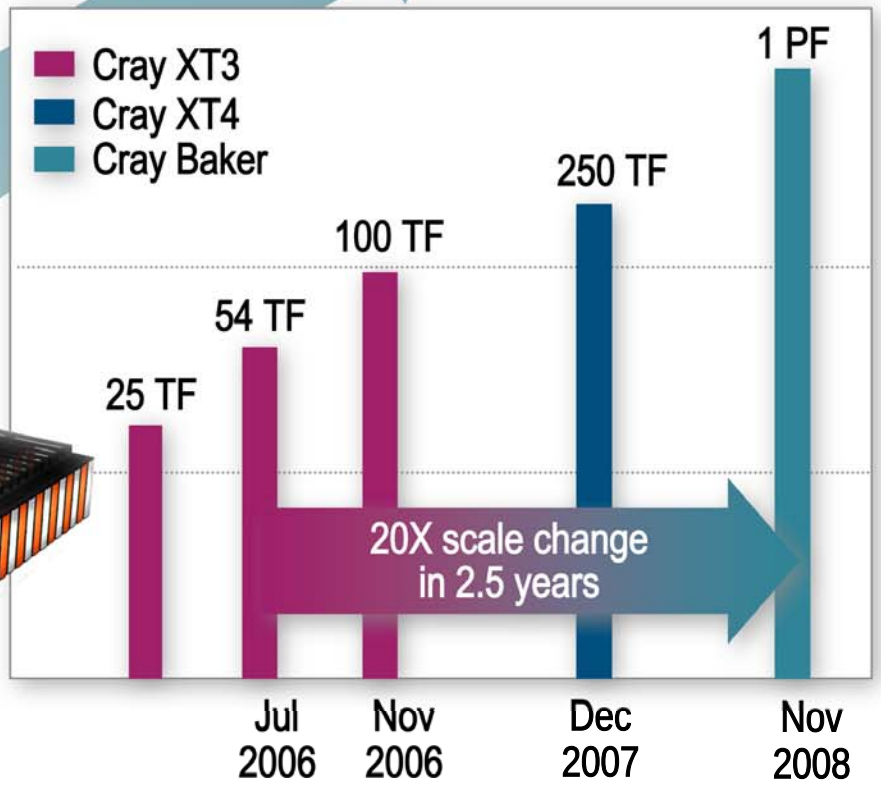
54 TF
(56 cabinets)
5294 nodes
10,588 processors
21 TF

2007

Today

Late
2006

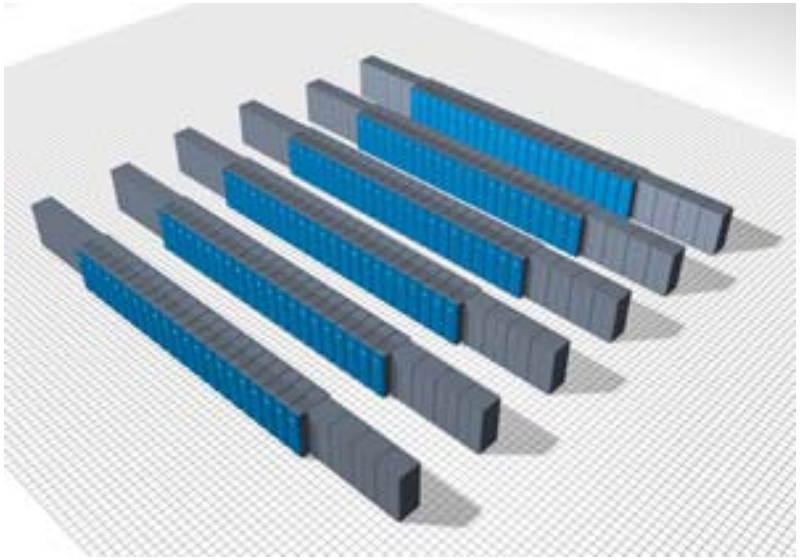
119 TF
(124 cabinets)
11,706 nodes
23,412 processors
46 TF



OAK RIDGE National Laboratory

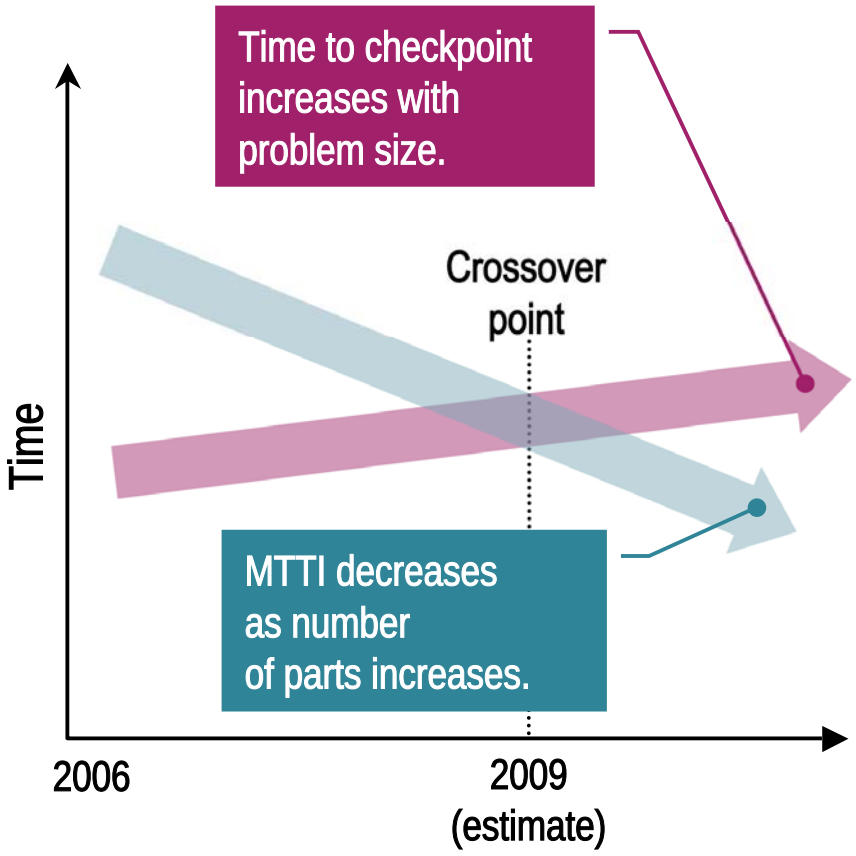
Today's applications and their runtime libraries may scale, but they are not prepared for the failure rates of sustained petascale systems

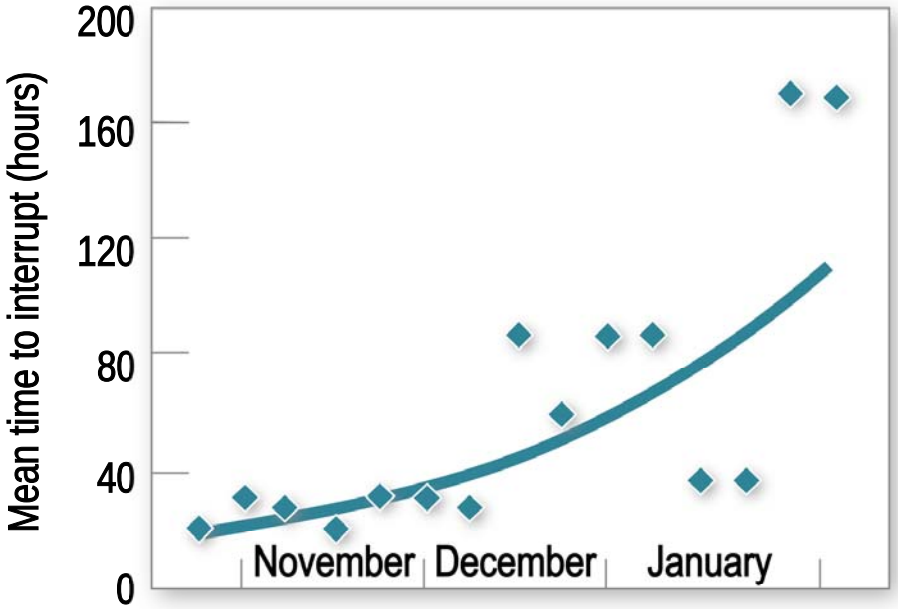
- Assuming a linear model and a failure rate 20X what is seen today.
- The RAS system automatically configures around faults – up for days.
- But every one of these failures kills the application that was using that node!



ORNL 1 PF Cray "Baker" system, 2009

Today's fault-tolerance paradigm (checkpoint) ceases to be viable on large systems



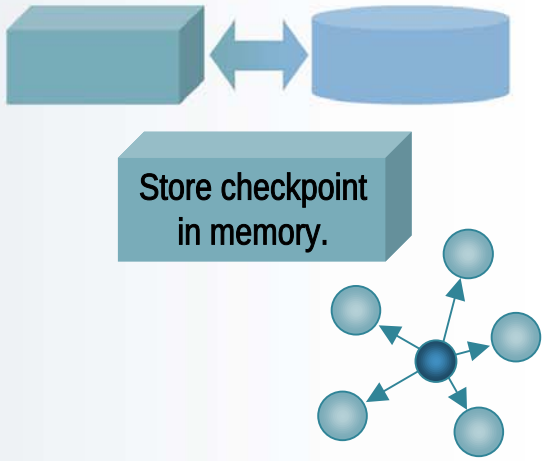


Good news: MTTI is better than expected for LLNL BG/L and ORNL XT4 (6–7 days, not minutes).

Need to develop new paradigms for applications to handle faults

Some state saved

1. Restart from checkpoint file
   [large apps today].
2. Restart from diskless checkpoint
   [Avoids stressing the I/O system and causing more faults].
3. Recalculate lost data from in-memory RAID.
4. Lossy recalculation of lost data
   [for iterative methods].



No checkpoint

5. Recalculate lost data from initial and remaining data.
6. Replicate computation across system.
7. Reassign lost work to another resource.
8. Use natural fault-tolerant algorithms.

Need to develop rich methodology to "run through" faults.

24/7 system can't ignore faults

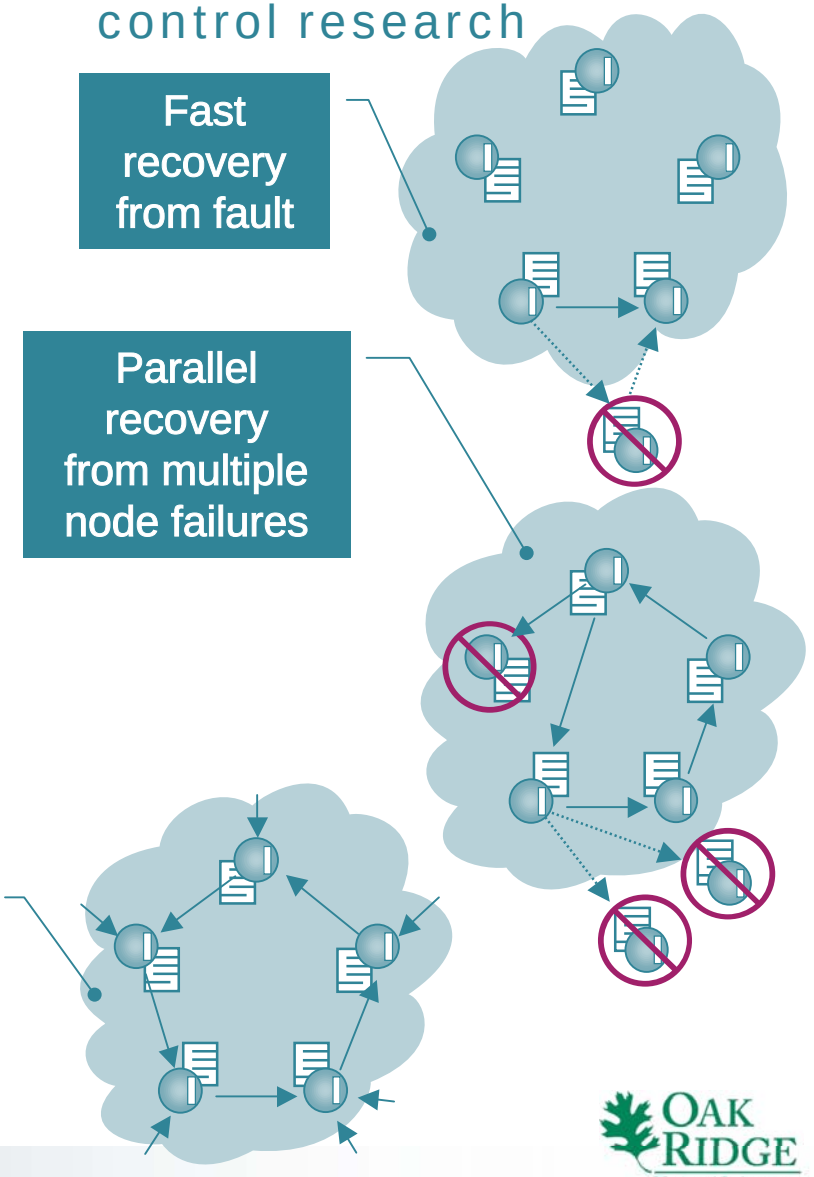
The file system can't let data be corrupted by faults.	I/O nodes must recover and cover failures.
The heterogeneous OS must be able to tolerate failures of any of its node type and instances.	For example: A failed service node shouldn't take out a bunch of compute nodes.
The schedulers and other system components must be aware of dynamically changing system configurations.	So that tasks get assigned around failed components.

Harness P2P control research

Fast recovery from fault

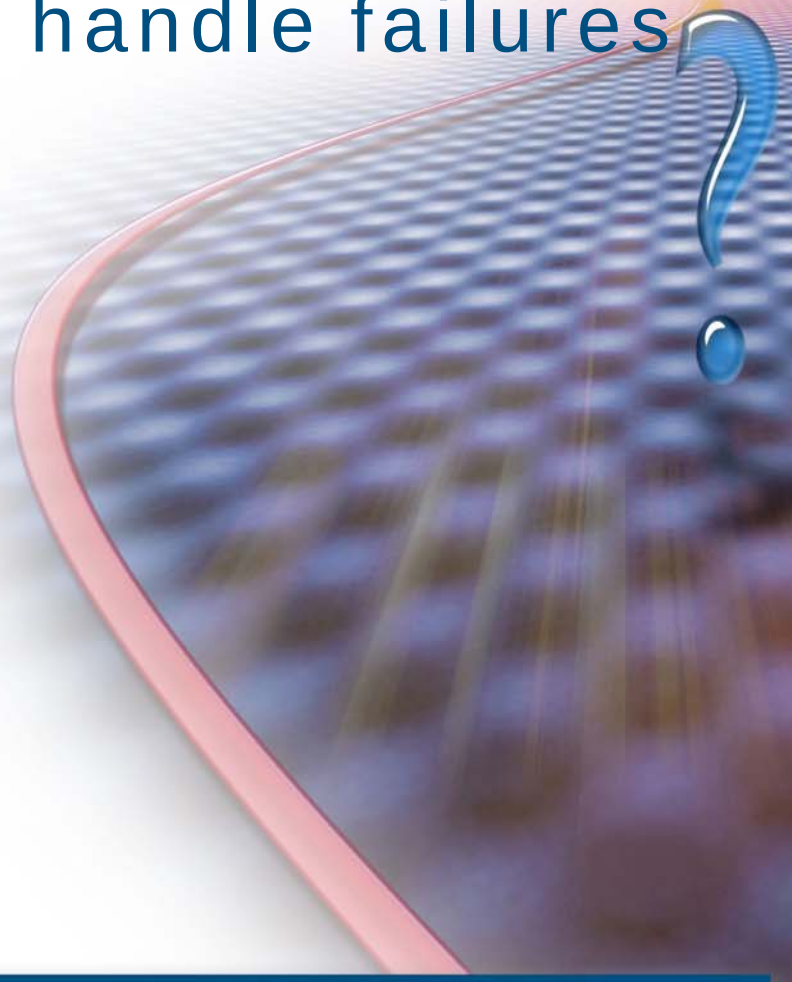
Parallel recovery from multiple node failures

Support simultaneous updates



6 options for system to handle failures

1. Restart–from checkpoint or from beginning.
2. Notify application and let it handle the problem.
3. Migrate task to other hardware before failure.
4. Reassign work to spare processor(s).
5. Replicate tasks across machine.
6. Ignore the fault altogether.



Need a mechanism for each application (or component) to specify to the system what to do if a fault occurs.

5 recovery modes for MPI applications

Harness project's FT-MPI explored 5 modes of recovery

1. ABORT: Just do as vendor implementations.
2. BLANK: Leave holes (but make sure collectives do the right thing afterward).
3. SHRINK: Reorder processes to make a contiguous communicator (some ranks change).
4. REBUILD: Respawn lost processes and add them to MPI_COMM_WORLD.
5. REBUILD_ALL: Same as REBUILD except that it rebuilds all communicators, and groups and resets all key values, etc.

These modes affect the size (extent) and ordering of the communicators.

It may be time to consider an MPI-3 standard that allows applications to recover from faults.

4 ways to fail anyway

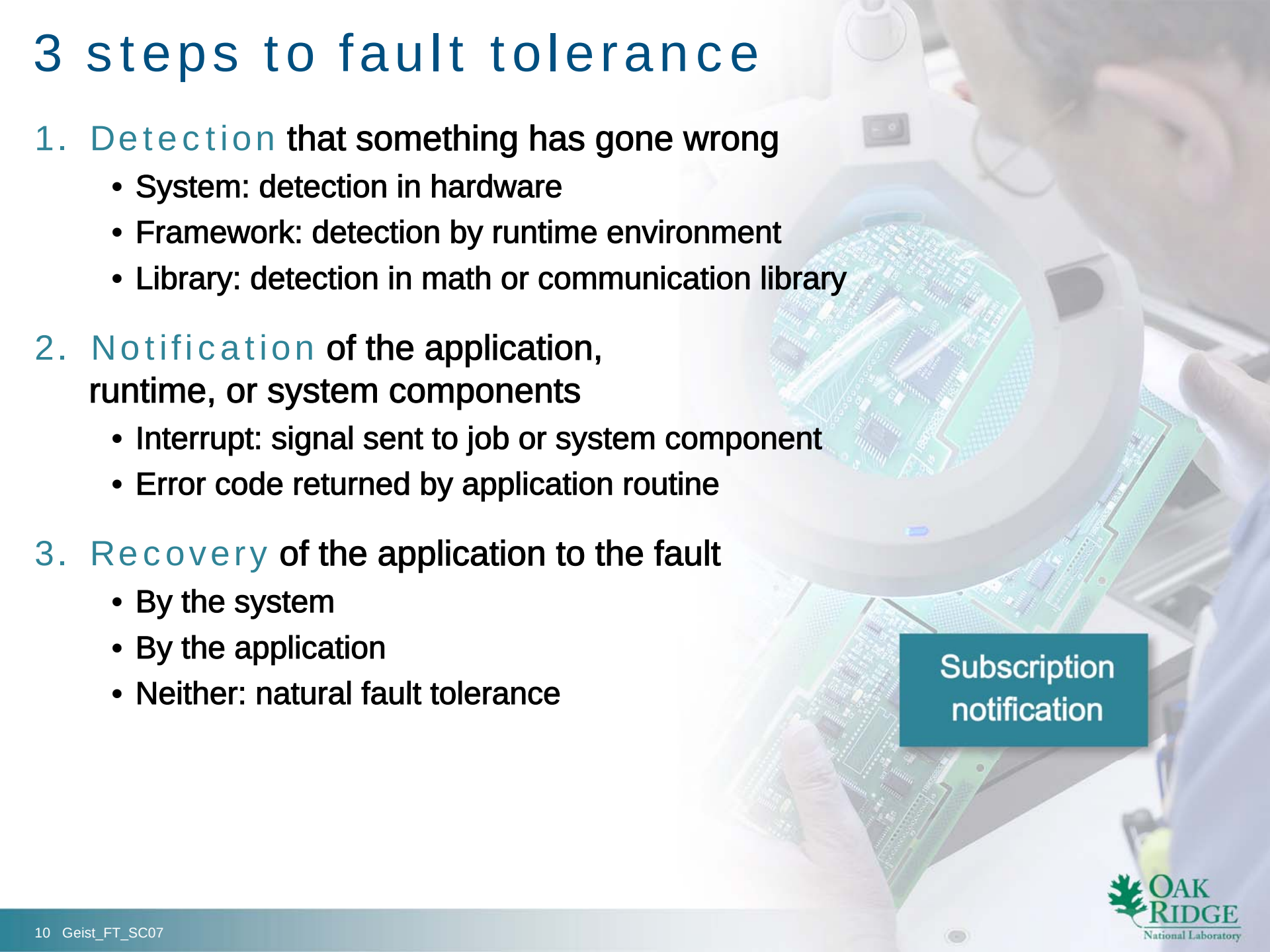
Validation of an answer on such large systems is a growing problem. Simulations are more complex, solutions are being sought in regions never before explored.

1. Fault may not be detected.
2. Recovery introduces perturbations.
3. Result may depend on which nodes fail.
4. Result looks reasonable, but it is actually wrong.

- Can't afford to run every job three (or more) times.
- Yearly allocations are like $5M–$10M grants.

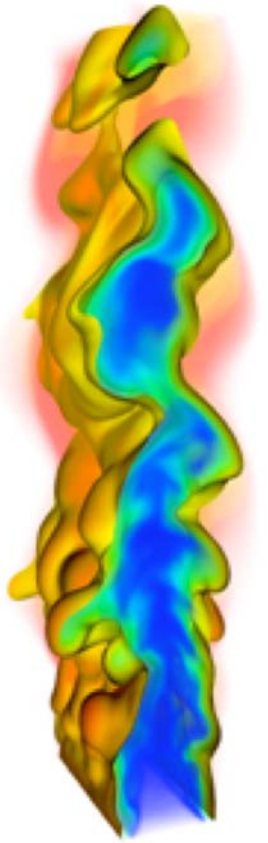
3 steps to fault tolerance

1. Detection that something has gone wrong
   - System: detection in hardware
   - Framework: detection by runtime environment
   - Library: detection in math or communication library
2. Notification of the application, runtime, or system components
   - Interrupt: signal sent to job or system component
   - Error code returned by application routine
3. Recovery of the application to the fault
   - By the system
   - By the application
   - Neither: natural fault tolerance



2 reasons the problem is only going to get worse

The drive for large-scale simulations in biology, nanotechnology, medicine, chemistry, materials, etc.



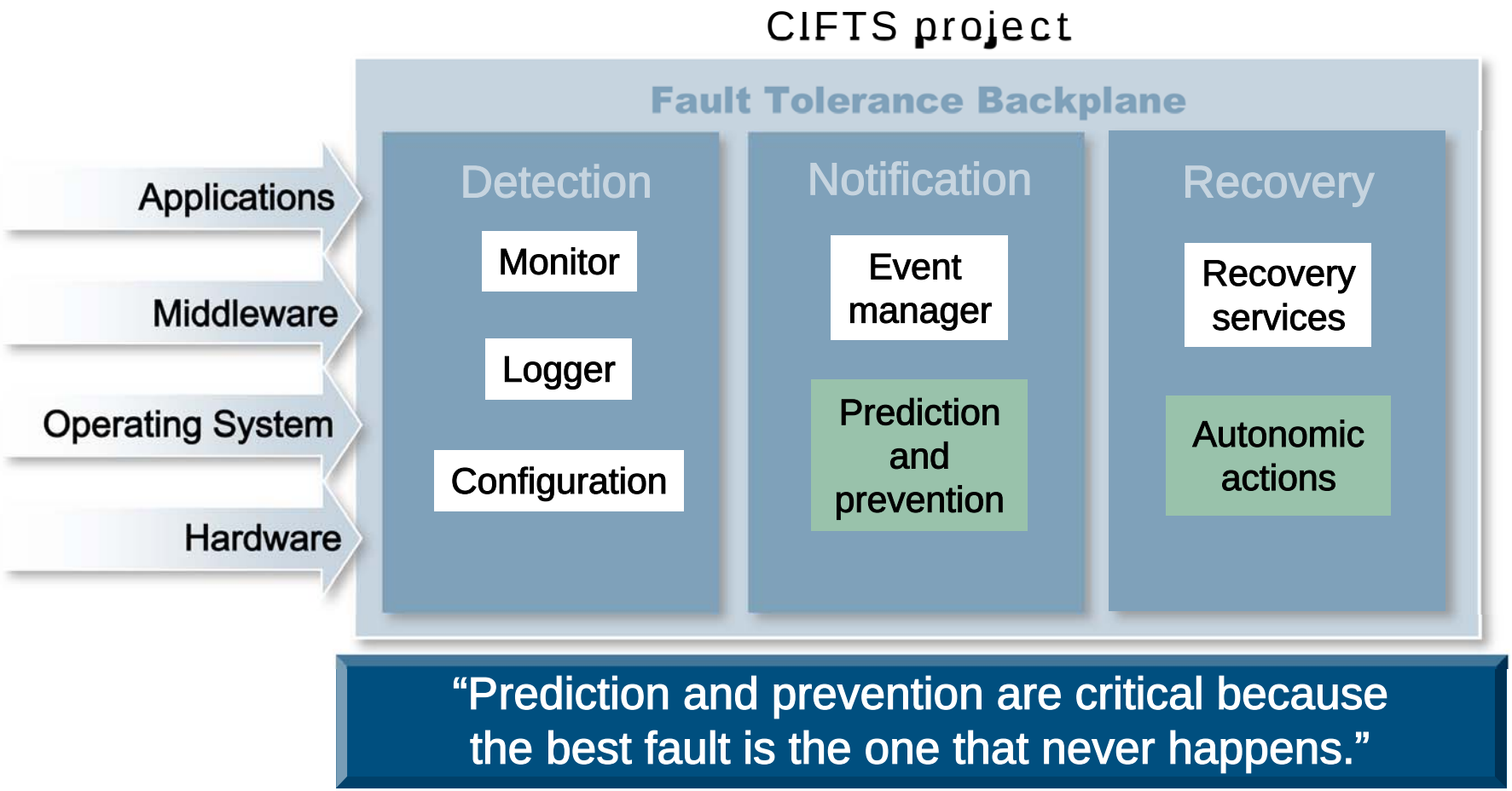
1. Require much larger problems (space):
   - easily consume the 2 GB per core in ORNL LCF systems.
2. Require much longer to run (time)
   - science teams in climate, combustion, and fusion want to run for a dedicated couple of months.

From a fault tolerance perspective:

- Space means that the job 'state' to be recovered is huge.
- Time means that many faults will occur during a single run.

1 holistic solution

We need coordinated fault awareness, prediction, and recovery across the entire HPC system from the application to the hardware.



Project under way at ANL, ORNL, LBL, UTK, IU, OSU

Contact

Al Geist

Computer Science Research Group
Computer Science and Mathematics Division
(865) 574-3153
gst@ornl.gov

OAK RIDGE National Laboratory