



“Everything You Know Is Wrong”

(Reflections On a Few Basic Assumptions)

Robert L. Clay, Ph.D.
Manager, Scalable Modeling and Analysis Systems
Sandia National Laboratories

SOS-18 Workshop
March 18, 2014
St. Moritz, Switzerland

Robert L. Clay, SOS-18

Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin company, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

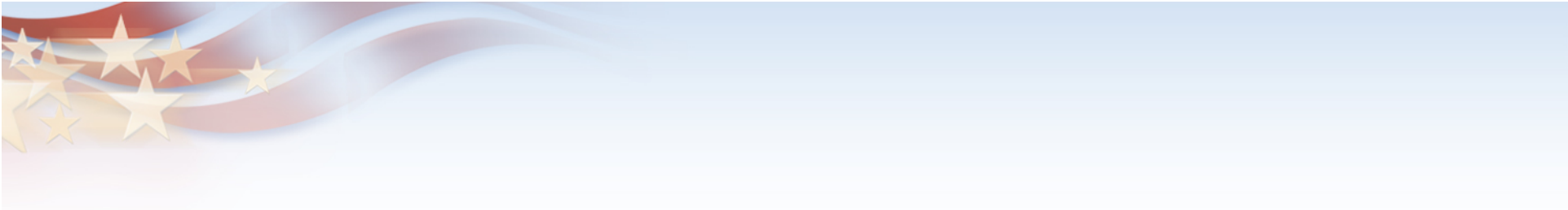


Inspired by Firesign Theatre



“This album addresses and parodies pseudo-scientific beliefs of the mid-1970s.” – Wikipedia.

And a sense that we need to rethink a few things



**More specifically, let's
examine some of our
assumptions around
HPC resilience.**

What is HPC Resilience?

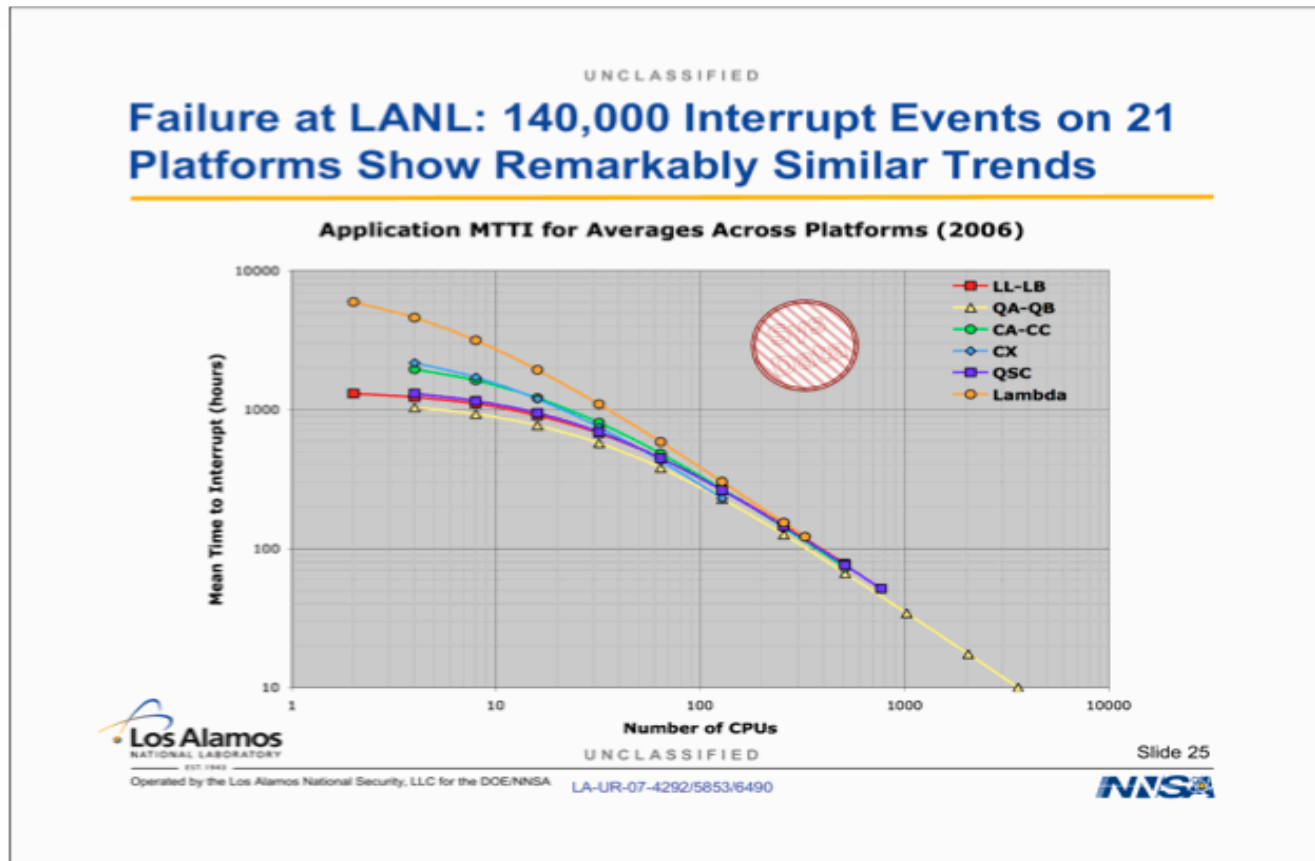
- We define resilient HPC as *correct and efficient computations at scale despite system degradations and failures.*
- Resilience is a cross cutting issue:
 - ✧ Hardware
 - ✧ Operating System
 - ✧ System Management
 - ✧ Runtime (Execution Model)
 - ✧ Application / Algorithms
 - ✧ Multi-layer (any/all combinations of the above)



Assumption 1: Computers are reliable digital machines.

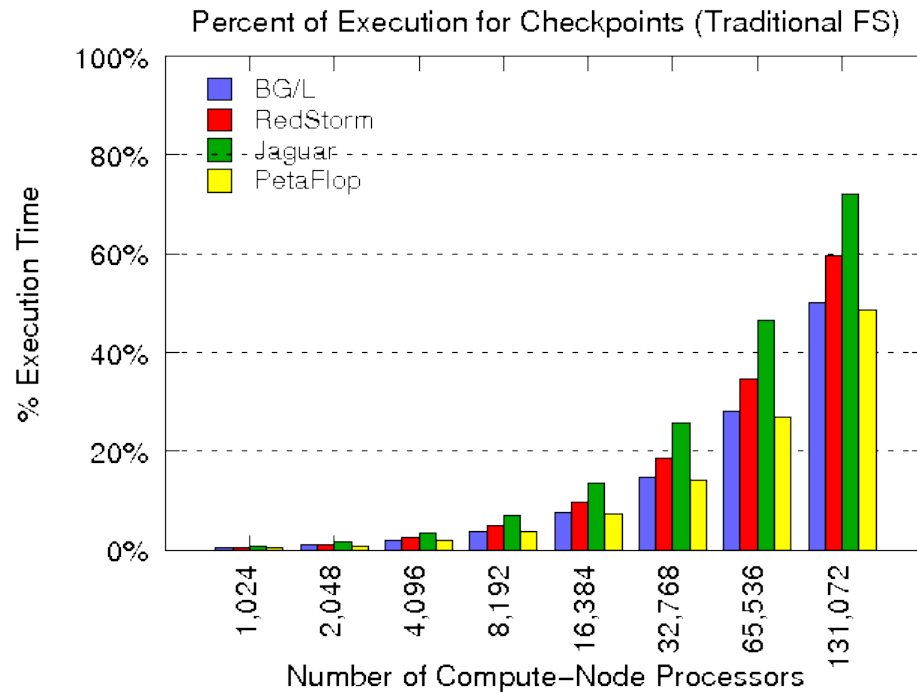
Doesn't get much more basic than this, but it's wrong [at some scale].

MTTI is shrinking as # cores grows

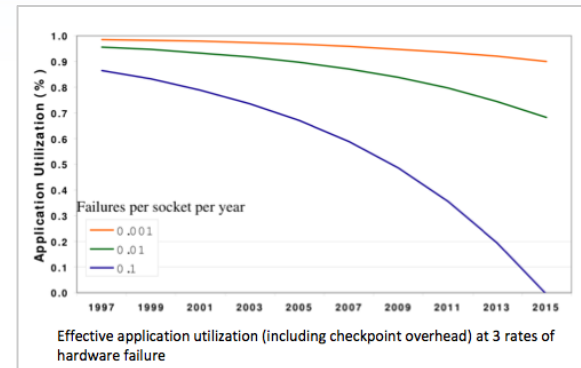


(Courtesy of John Daly)

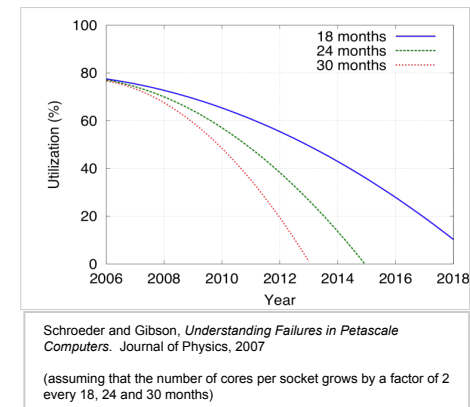
Checkpoint trend isn't good



Oldfield et al., *Modeling the Impact of Checkpoints on Next-Generation Systems*. MSST, 2007



(Courtesy of Lucy Nowell & Sonia Sachs)



Machine utilization is going to zero! (Not really)

Checkpoint/Restart: Disproportional response to local failures

- **Single node failures account for the major (~2/3) HPC system failures**
- **Short MTBFs due to the increase of error-prone components**
 - **Today: ~ twice a day**
 - **2020: every 30-60 minutes?**
- **Hardware Solution is infeasible**
 - **Performance loss**
 - **Power budget (20MW per system)**
- **Current response**
 - **Kill all processes**
 - **Recover system**
 - **Dependent on Global File System to keep application state**

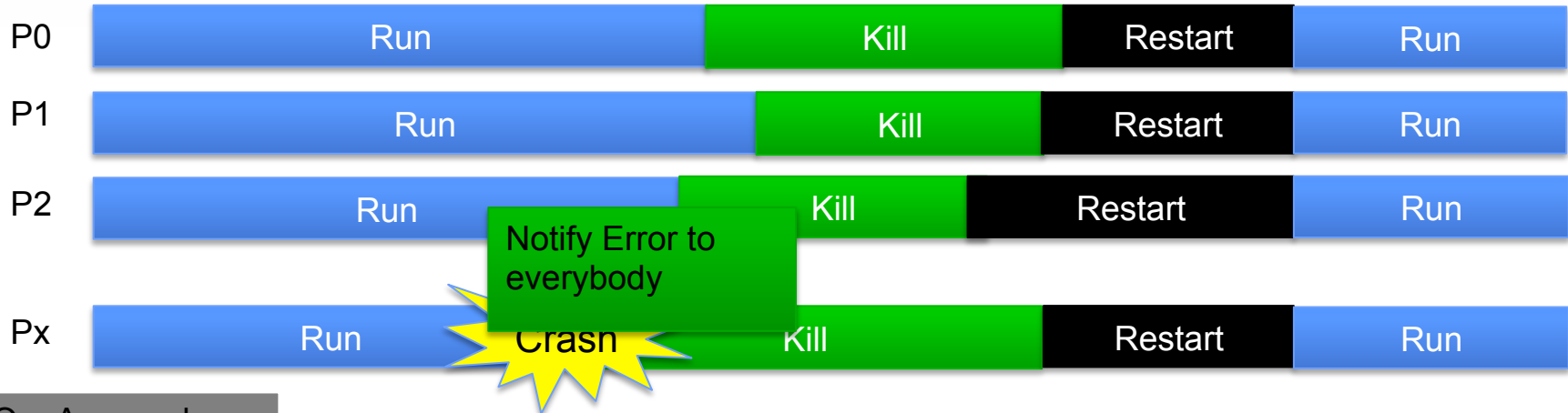
We seek a Local Failure Local Recovery (LFLR) resilient programming model to allow proportional response to single node/process failure

proportional

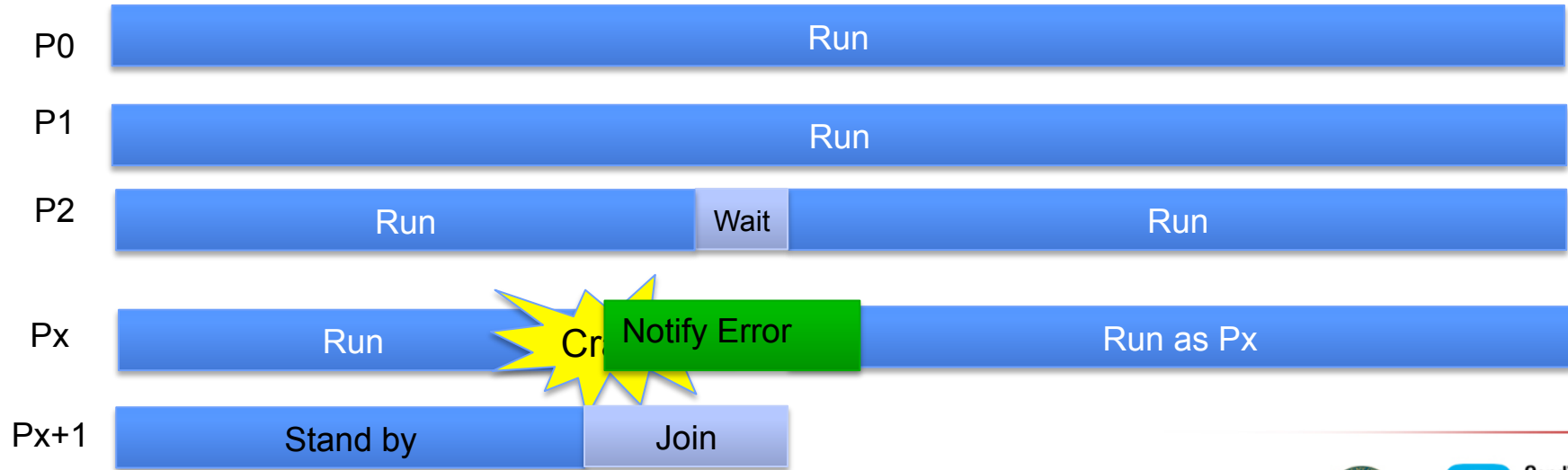


LFLR Programming Model

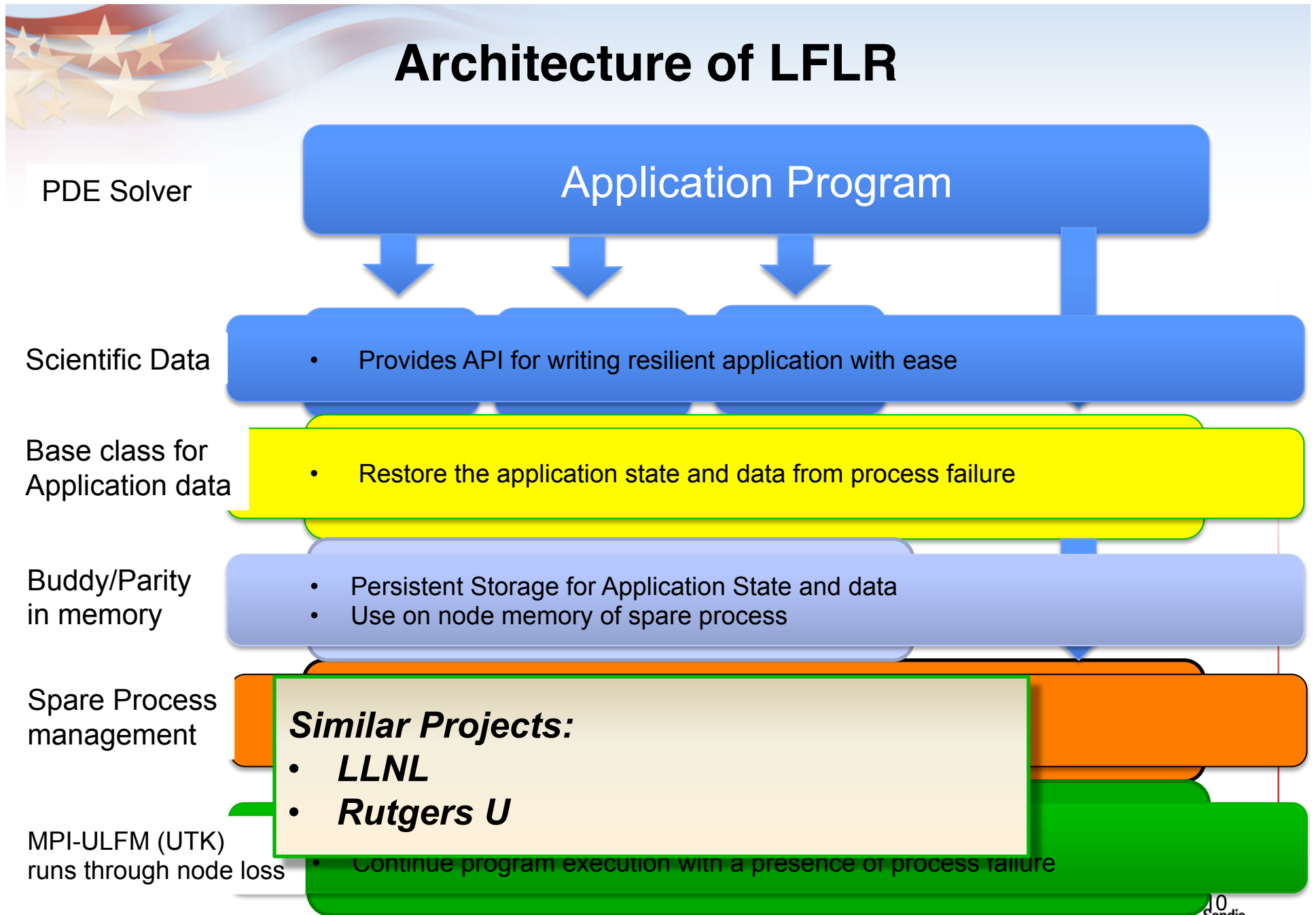
Checkpoint Restart



Our Approach



Architecture of LFLR

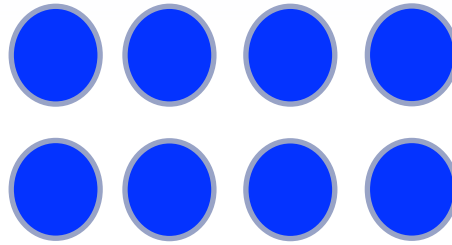


MPI-ULFM: User Level Fault Mitigation

- Proposed for MPI-3.1 standard
- MPI calls (recv, irecv, wait, collectives) notify errors when the peer process(es) dies
- Healthy processes can continue
- Several MPI calls for fixing MPI communicator
 - **MPI_Comm_agree** : Check the global status of MPI_Comm
 - **MPI_Comm_revoke**: Invalidate MPI Communicator
 - **MPI_Comm_shrink**: Fix MPI Communicator removing dead process
- User is responsible for the recovery after MPI_Comm_shrink call
- Prototype code is available at <http://fault-tolerance.org>
 - Developed by U of Tennessee (G. Bosilica)

Scalable Recovery Via Spare Process Reserve

MPI processes for computation

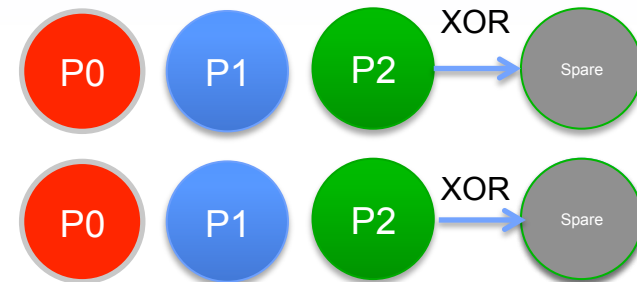


Spare MPI Processes

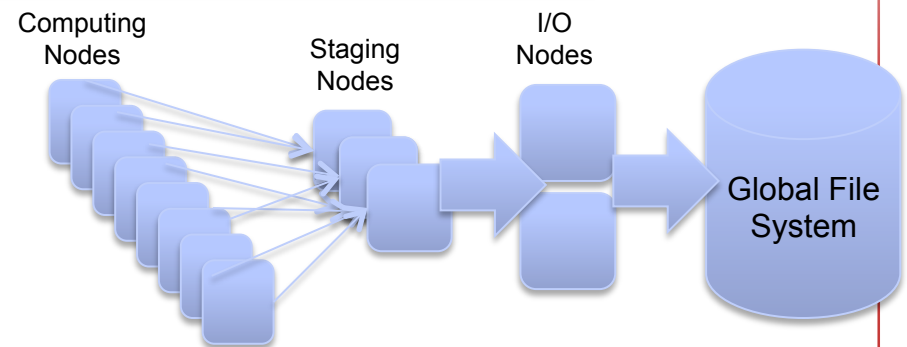
- ULFM-MPI only provides minimum set of APIs for process loss
 - Many apps need to remap the work after communicator shrink ☹️
 - Vendor's MPI (such as Cray) does not support `MPI_Comm_spawn`
- Allocate hot spare process to replace the lost process
 - Can be used for the other resiliency features
- 3 MPI calls to perform rank re-assignment
 - `MPI_Comm_shrink`
 - `MPI_Comm_create`
 - `MPI_comm_split`

Persistent Storage and Its Options

- In-memory, persistent storage
 - RAID-like redundancy
 - Performed by group (of 128 or 256)
- Staging nodes
 - Dedicated nodes to store temporary data
 - We ex
- Caching
 - Explo
 - Handl
 - Scalable Checkpoint and Restart (by Mohror et al.)



We employ in-memory storage of spare processes dedicated for checksum/parity

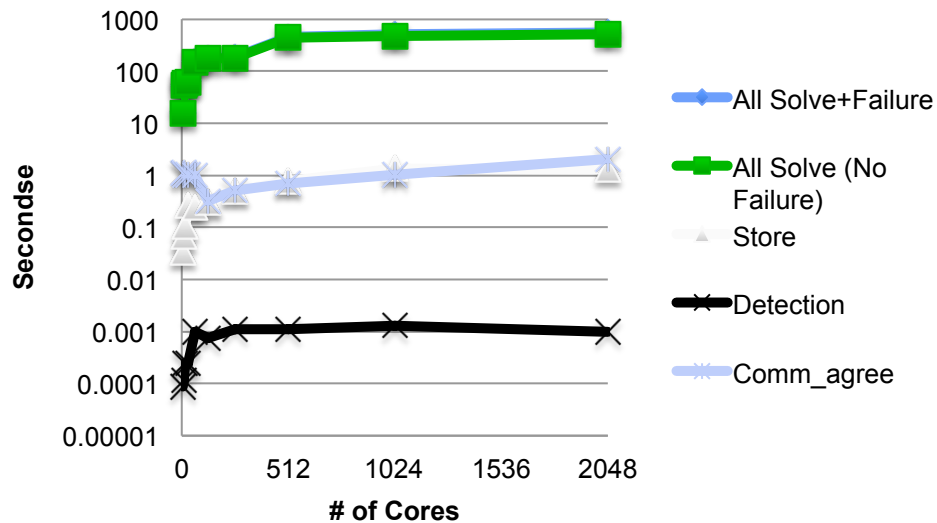


Preliminary Result

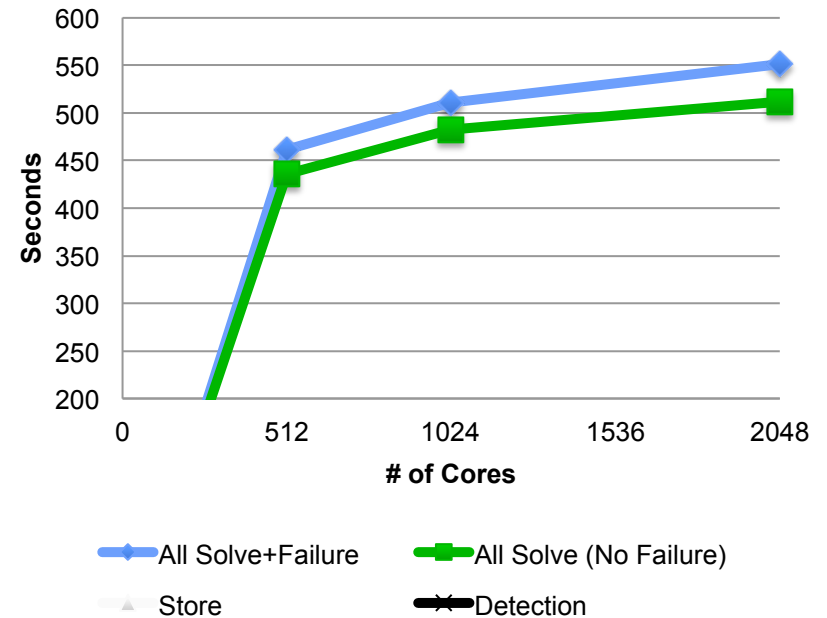
- **Time Stepping PDE (MiniFE Mantevo proxy app)**
 - 3D Finite Element
 - Multiple Linear System Solution
 - RHS is updated by LHS in the previous linear system solve
- **Resiliency Features**
 - Spare Process is used for recovery
 - Application info are stored only once
 - Vectors are stored in every time step
- **Weak scaling**
 - 64x64x64 for ULFM for 4 cores and increase the problem size ($x*y*z$) linearly
 - Cray Cluster with SandyBridge (2.6Mhz) 16 cores (2CPU) per node, FDR Infiniband
 - Process failure during linear system solve (2048 PEs)
 - MPI-ULFM with our own fix for resilient collective

Results with MPI-ULFM

Performance Time Stepping MiniFE



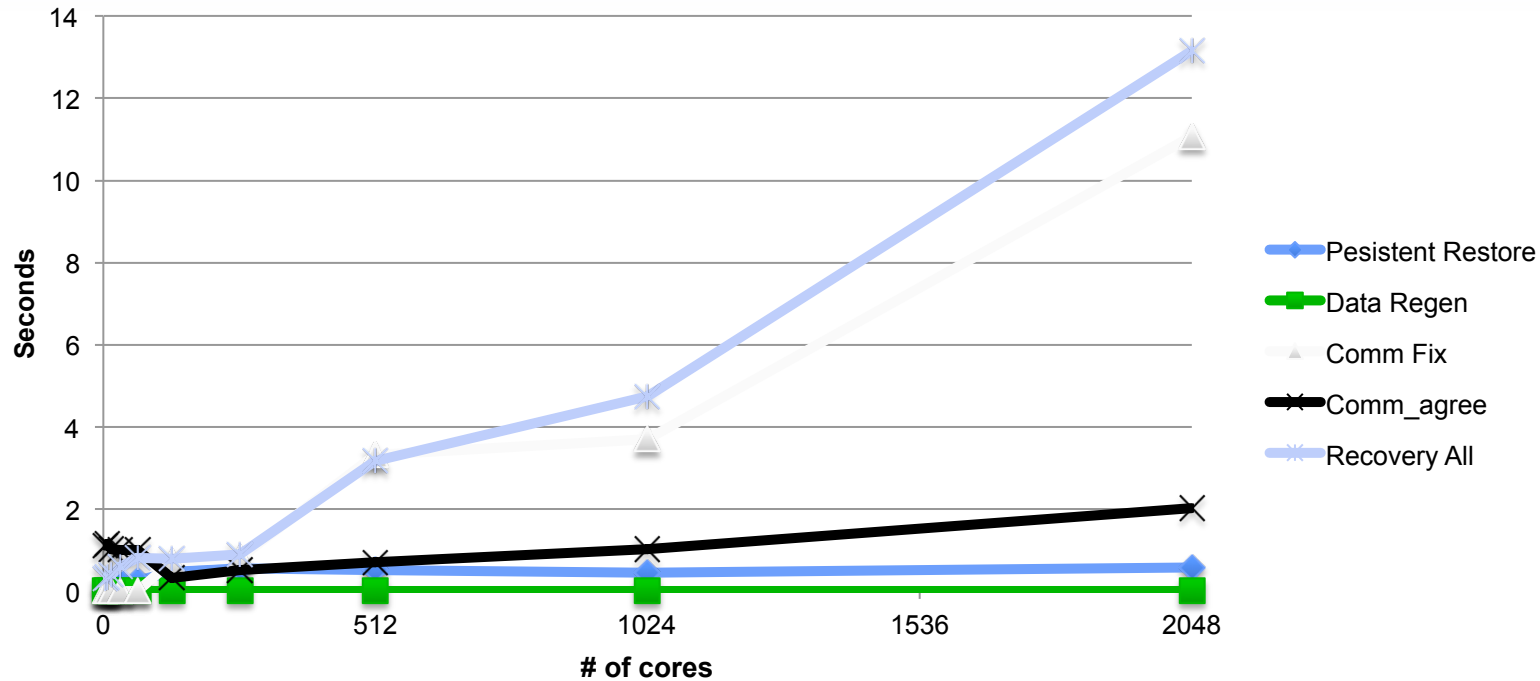
Performance of Time Stepping MiniFE



- **Group size = 128**
- **Negligible overhead for Persistent Data Store**
- **Negligible overhead for Failure Detection**
- **Recovery cost increases from 512 cores or larger**

Results with MPI-ULFM

Recovery Cost



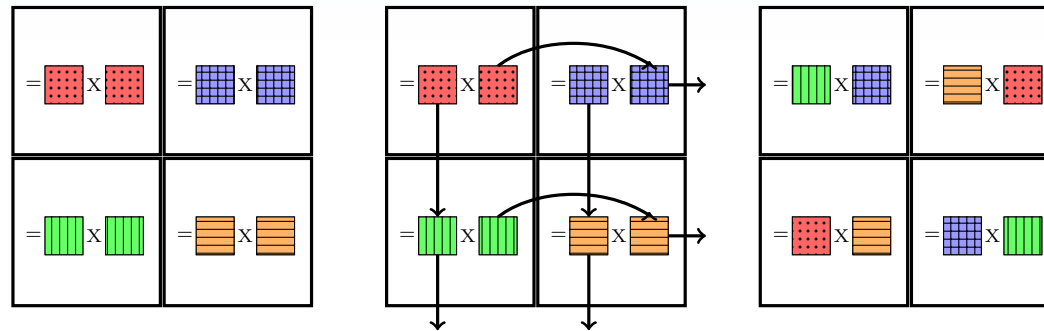
- **Negligible Cost for data recovery**
 - **Very scalable**
- **Scalability Issues in Communicator fix**
 - **Already reduced by >10x (improvements ongoing)**



Assumption 2: We don't need to change our codes much.

Also known as “MPI is fine”. Also known as “MPI + X” where X is undefined, but it will work itself out over time. *The real question may be whether the CSP BSP programming model will work well at exascale.*

Programming model exploration for resilience with simulation



Systolic matrix-matrix multiplication involves “synchronous” migration of matrix blocks.
Start with MPI.

Actual MPI code

```

208 for (int iter=0; iter < niter; ++iter){
209     /** Prefetch next iteration */
210     MPI_Isend(left_block, nelems_left_block, MPI_DOUBLE,
211              row_send_partner, row_tag, MPI_COMM_WORLD, &reqs[0]);
212     MPI_Isend(right_block, nelems_right_block, MPI_DOUBLE,
213              col_send_partner, col_tag, MPI_COMM_WORLD, &reqs[1]);
214     MPI_Irecv(next_left_block, nelems_left_block, MPI_DOUBLE,
215              row_recv_partner, row_tag, MPI_COMM_WORLD, &reqs[2]);
216     MPI_Irecv(next_right_block, nelems_right_block, MPI_DOUBLE,
217              col_recv_partner, col_tag, MPI_COMM_WORLD, &reqs[3]);
218
219     DGEMM('T', 'T', nrows, ncols, nlink, 1.0, left_block, nrows,
220           right_block, ncols, 0, product_block, nrows);

```

Simulator code

```

208 for (int iter=0; iter < niter; ++iter){
209     /** Prefetch next iteration */
210     MPI_Isend(left_block, nelems_left_block, MPI_DOUBLE,
211              row_send_partner, row_tag, MPI_COMM_WORLD, &reqs[0]);
212     MPI_Isend(right_block, nelems_right_block, MPI_DOUBLE,
213              col_send_partner, col_tag, MPI_COMM_WORLD, &reqs[1]);
214     MPI_Irecv(next_left_block, nelems_left_block, MPI_DOUBLE,
215              row_recv_partner, row_tag, MPI_COMM_WORLD, &reqs[2]);
216     MPI_Irecv(next_right_block, nelems_right_block, MPI_DOUBLE,
217              col_recv_partner, col_tag, MPI_COMM_WORLD, &reqs[3]);
218
219     DGEMM('T', 'T', nrows, ncols, nlink, 1.0, left_block, nrows,
220           right_block, ncols, 0, product_block, nrows);

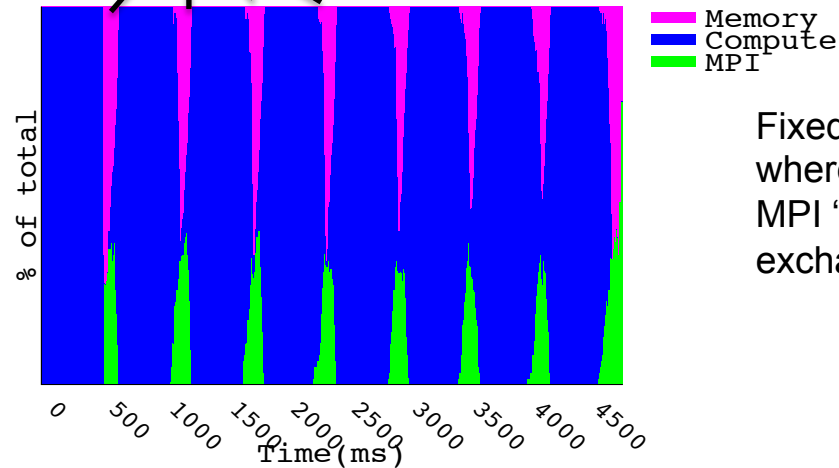
```

With a few linker tricks, you get direct compilation of source code. No DSL! Only one source to maintain!

Programming model exploration: SPMD resilience results

Synchronous MPI
data exchange

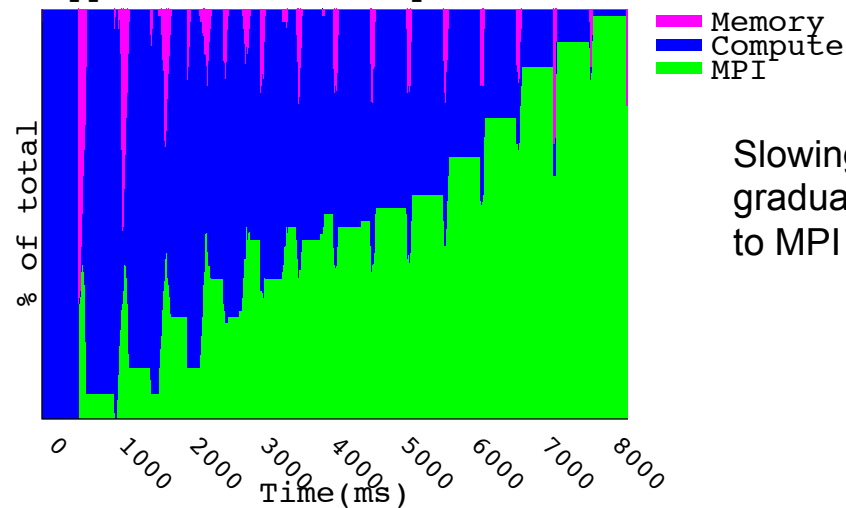
Application Activity Over Time



If all nodes the same
speed...

Fixed-time quanta (FTQ) shows
where app is spending time. Here
MPI “stutters” during synchronous
exchange

Application Activity Over Time

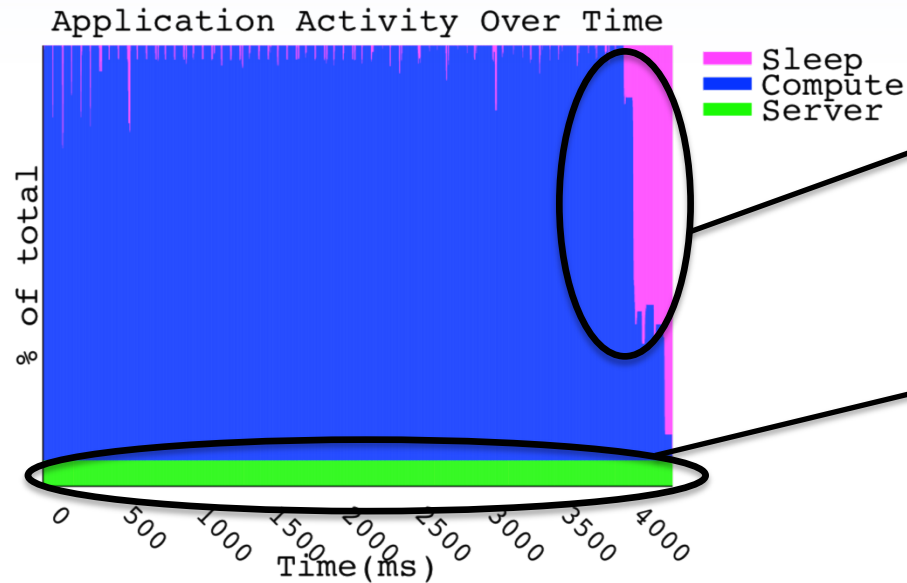


If one node
overheats or has
bad DIMM and
slows down...

Slowing 1 core on 1 of 8 nodes
gradually chokes off computation due
to MPI synchronization

Programming model exploration: asynchronous, task-DAG model

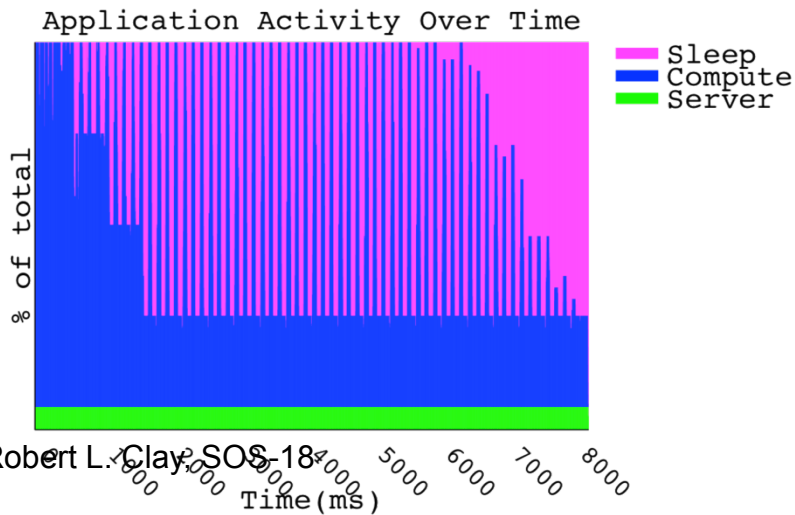
If all nodes the same speed...



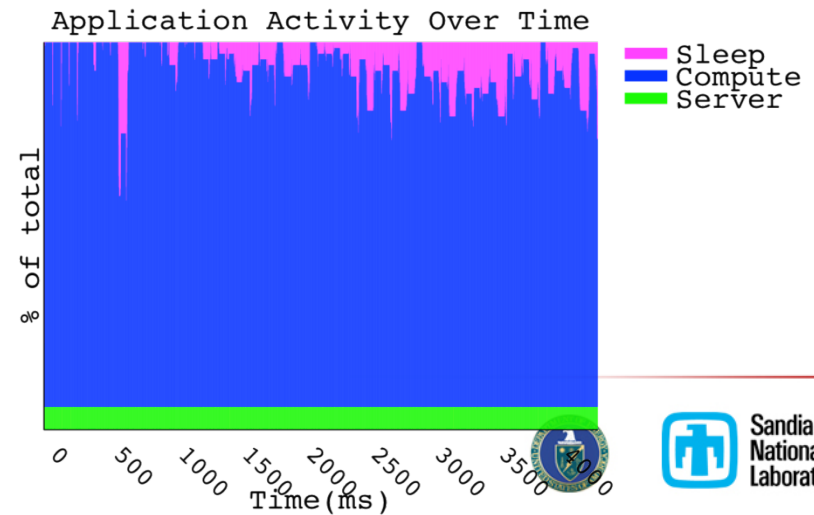
Termination detection/
work stealing needs to
be optimized

Data movement service
is constant overhead –
single thread dedicated
to communication

If node slows down...

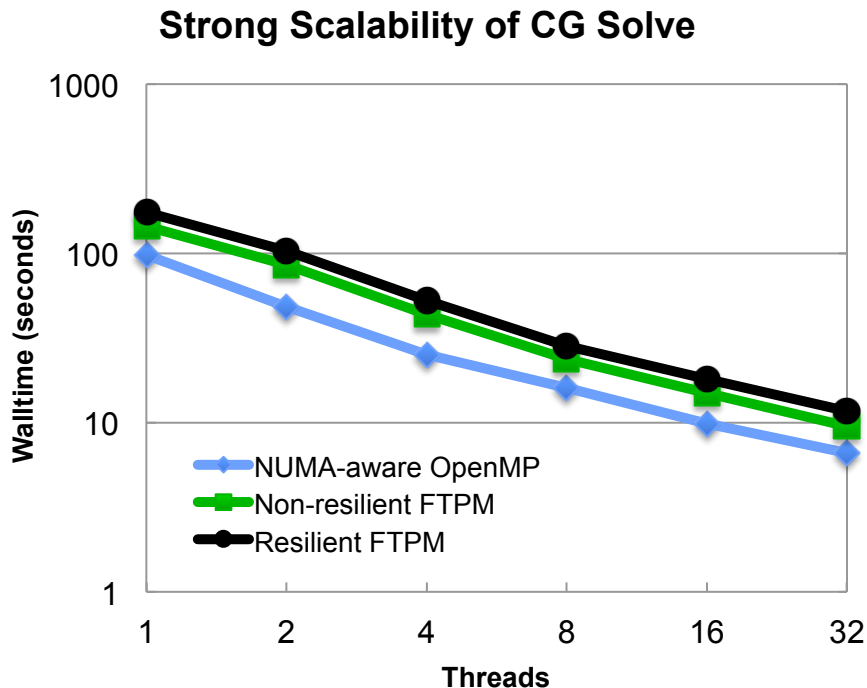


With load balancing...



Demonstrated resilience to silent data corruption in our on-node, task-based conjugate gradient solver driven by miniFE proxy app

- *Automatically* detected/corrected multi-bit silent data corruption in user data structures using triple-modular redundancy for scalars and 2D checksums for vectors and matrices (application/algorithm agnostic)



- Technique applied selectively by self-stabilizing CG algorithm in order to lower protection cost
 - 0.8% memory overhead on protected data structures
 - 20% increase in runtime due to checksum validation on every 20th iteration

Benchmarks from SGI Altix UV 10 with four 8-core Nehalem EX and 512 GB globally-shared memory



Assumption 3: Well, at least the algorithms will work.

Maybe, maybe not.

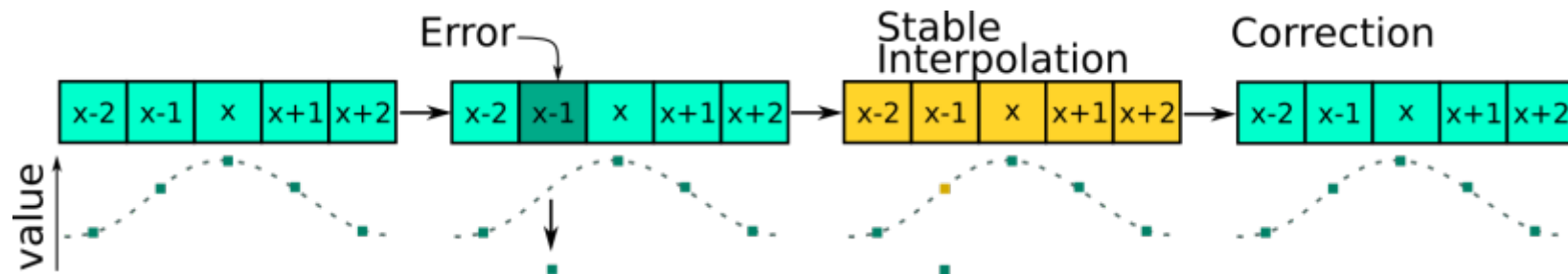
Error-Correcting Algorithms Can Mitigate Silent Errors & Offer New Co-design Options

- Even at commodity scale, ECC memory & ECC processors show the rising need for error correction



ECC memory

- With increasing scale and with power limitations, errors can occur “silently” without indication that something is wrong
- Numerical algorithms already deal with error from truncation, etc.; **specially designed algorithms can mitigate silent bit flips** as well

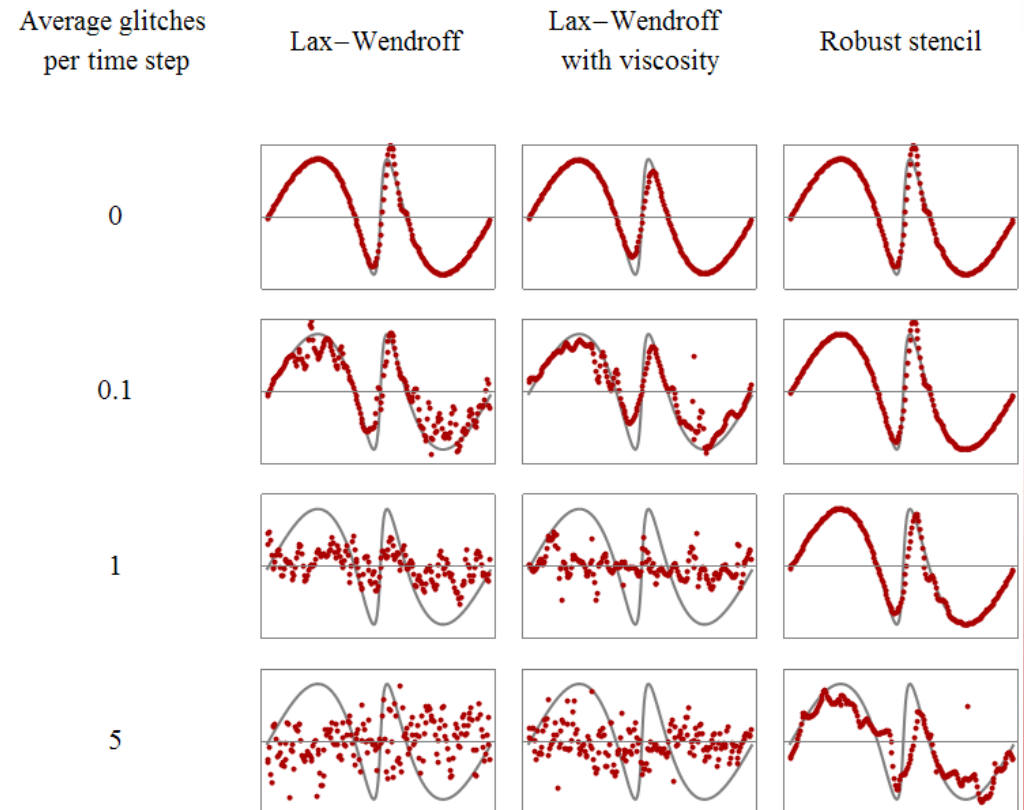


- These **robust stencil** algorithms not only address scale-up of current silent-error rates, but may enable **new “lossy” architecture options** with more power-efficient accelerators or reduced latency

Robust stencils can discard outliers to mitigate bit flips in PDE solving

- A simple 1D advection equation $\partial u/\partial t = \partial u/\partial x$ illustrates the behavior of finite-difference schemes
- The robust stencil here computes a second-order u at position i from one of these subsets after discarding the most extreme value:

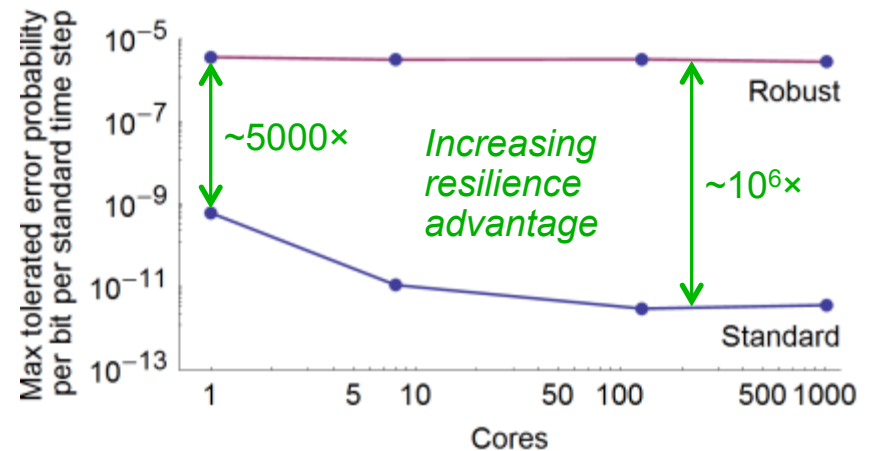
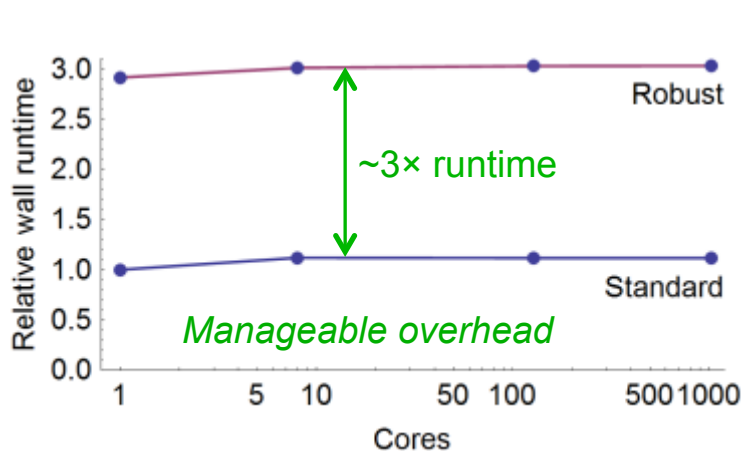
- $\{ i-3, \quad i-1, \quad i+1, \quad i+3 \}$
- $\{ \quad i-2, \quad i, \quad i+2 \}$
- $\{ \quad i-1, \quad i, \quad i+1 \}$



Simple demo in
Mathematica

Preliminary Weak-Scaling Experiments Show Favorable Trends for Robust Stencil

- As a research tool for ongoing use, we have implemented a modular C++/MPI framework for explicit Cartesian PDE solvers
 - Captures “halo exchange” pattern in generic form
- Preliminary results from many short runs, 10^6 grid cells per core



• Further questions:

- How does resilience scale with longer runs and more realistic PDEs?
- How realistic is our way of emulating memory bit flips?
- What happens if bit flips also occur in message communication?



Summary

- Obviously, the title was a joke. But, I do think we need to re-examine some of our long-standing assumptions.
- Some of the ongoing work looks promising for certain important failure modes (e.g., node loss).
- The CSP BSP programming model may not hold up long term. Asynchronous, many-task programming models are interesting.
- Silent data corruption may require we re-evaluate things at an even deeper level.



Acknowledgements

- Rob Armstrong (Robust Stencils)
- Janine Bennett (pmodels)
- Mike Heroux (LFLR)
- Hemanth Kolla (pmodels)
- Jackson Mayo (Robust Stencils)
- Philippe Pebay (SST/macro)
- Nicole Slattengren (pmodels)
- Keita Teranishi (LFLR)
- Jeremiah Wilke (SST/macro)





Thank You

Robert L. Clay
rlclay@sandia.gov
+1 (209) 610-2929

**Favorite assumption that didn't make the list:
If you buy a bigger computer, your code will
run faster.**