SAND2014-2182C

"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

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 Theater

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?

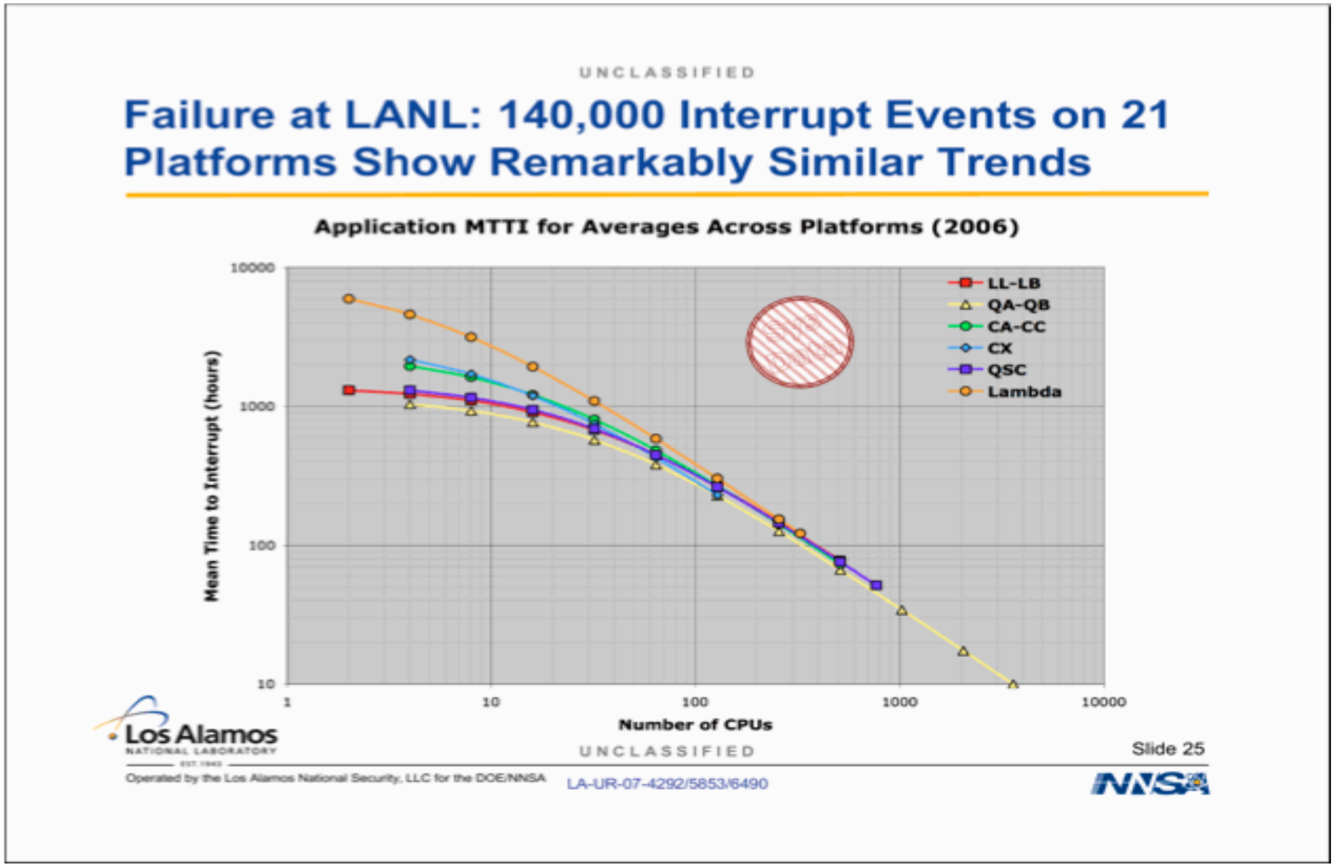
- We define resilient HPC as *correct and efficient computations at scale despite system degradations and failures*.

- 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



UNCLASSIFIED

Failure at LANL: 140,000 Interrupt Events on 21 Platforms Show Remarkably Similar Trends

Los Alamos NATIONAL LABORATORY EST. 1943

UNCLASSIFIED

Slide 25

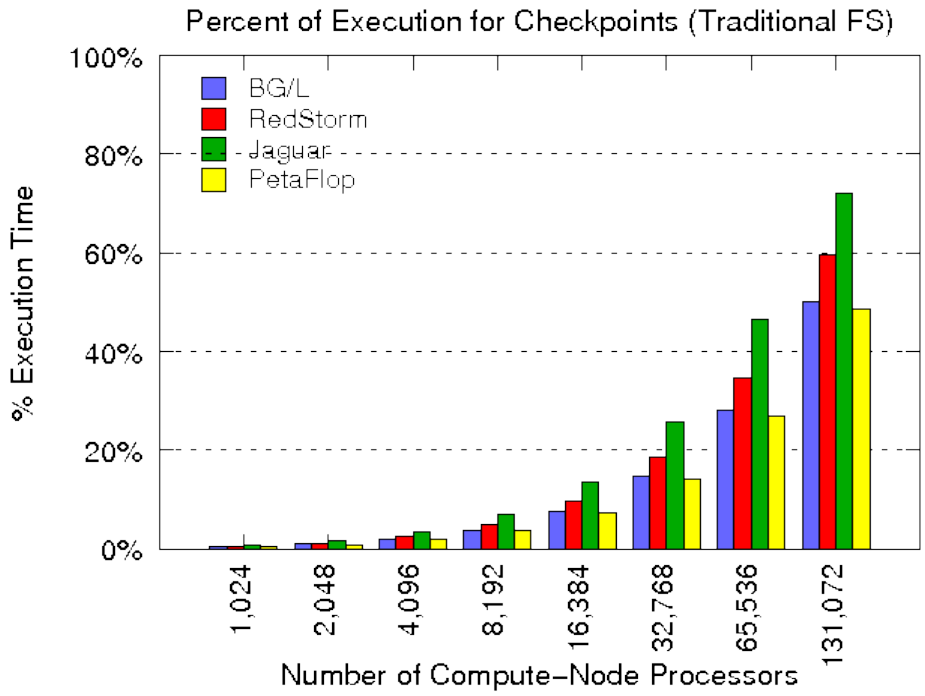
Operated by the Los Alamos National Security, LLC for the DOE/NNSA

LA-UR-07-4292/5853/6490

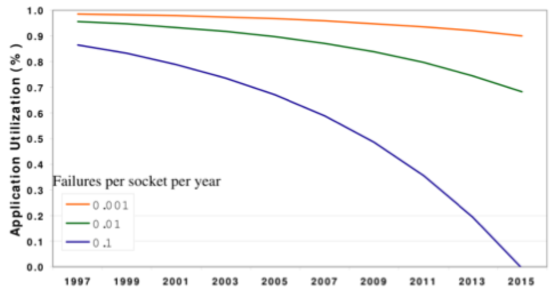
NNSA

(Courtesy of John Daly)

Checkpoint trend isn’t good

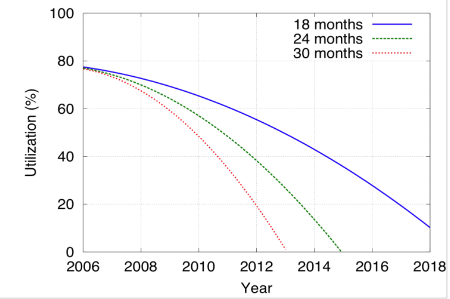


Oldfield et al., *Modeling the Impact of Checkpoints on Next-Generation Systems*. MSST, 2007



Effective application utilization (including checkpoint overhead) at 3 rates of hardware failure

(Courtesy of Lucy Nowell & Sonia Sachs)



Schroeder and Gibson, *Understanding Failures in Petascale Computers*. Journal of Physics, 2007

(assuming that the number of cores per socket grows by a factor of 2 every 18, 24 and 30 months)

Machine utilization is going to zero! (Not really)

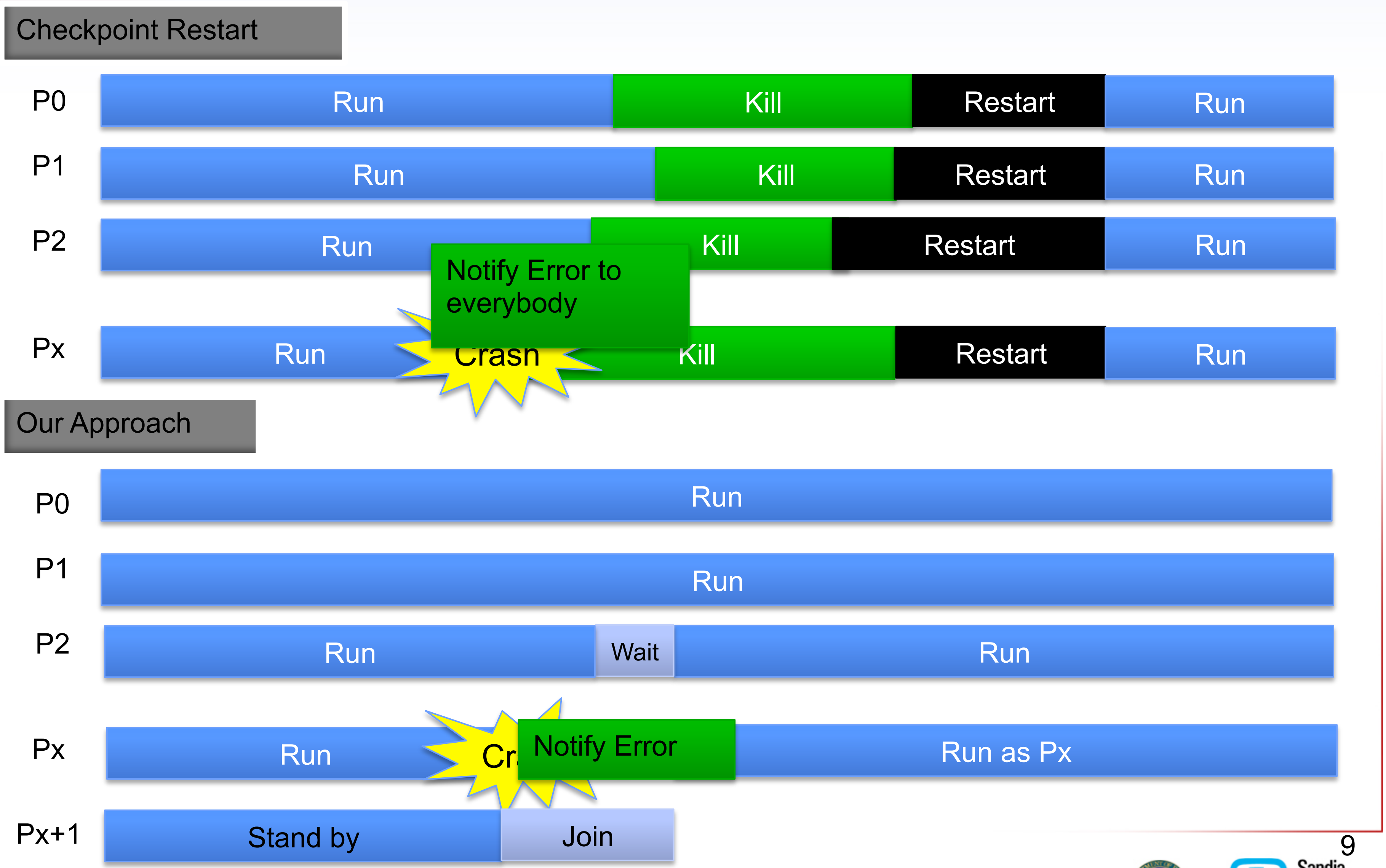
Robert L. Clay, SOS-18

Checkpoint/Restart: Disproportional response to local failures

- Single node failures account for the major HPC system failures
  - 85% on LLNL clusters (Moody et al. 2010)
  - 2/3 on Titan (ORNL)
- Short MTBFs due to the increase of error-prone components
  - Titan crashes twice a day
  - 2020: Every 30 minutes-1 hour?
- Hardware Solution is infeasible
  - Performance loss
  - Power
- Current roportional respons
  - Kill all
  - Recovery involves global restart
  - 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

LFLR Programming Model



Checkpoint Restart

P0 Run Kill Restart Run

P1 Run Kill Restart Run

P2 Run Kill Restart Run

Notify Error to everybody

Px Run Crash Kill Restart Run

Our Approach

P0 Run

P1 Run

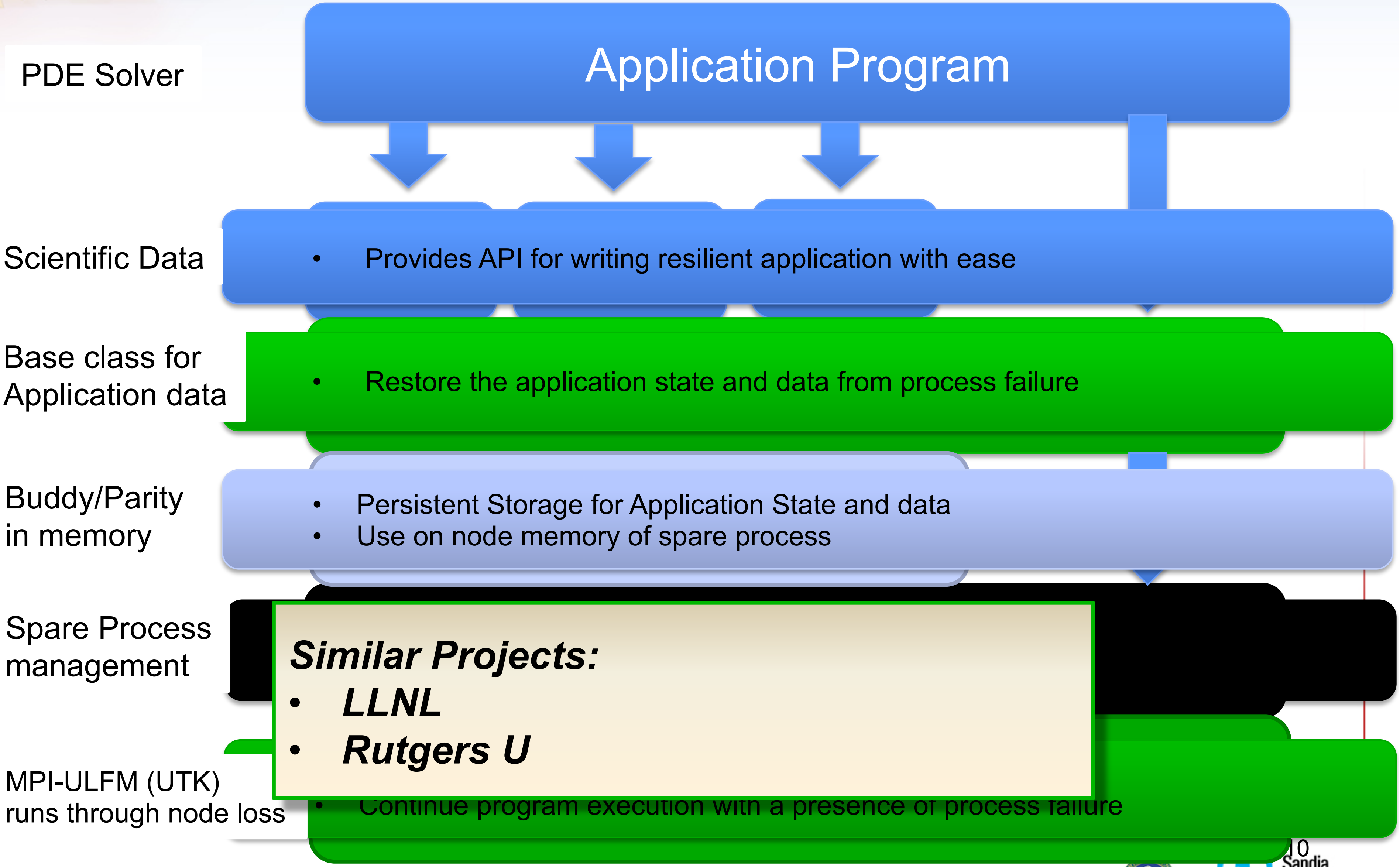
P2 Run Wait Run

Px Run Cr Notify Error Run as Px

Px+1 Stand by Join

Robert L. Clay, SOS-18

Architecture of LFLR



PDE Solver

Application Program

Scientific Data

- Provides API for writing resilient application with ease

Base class for Application data

- Restore the application state and data from process failure

Buddy/Parity in memory

- Persistent Storage for Application State and data
- Use on node memory of spare process

Spare Process management

MPI-ULFM (UTK) runs through node loss

- Continue program execution with a presence of process failure

Similar Projects:

- ***LLNL***
- ***Rutgers U***

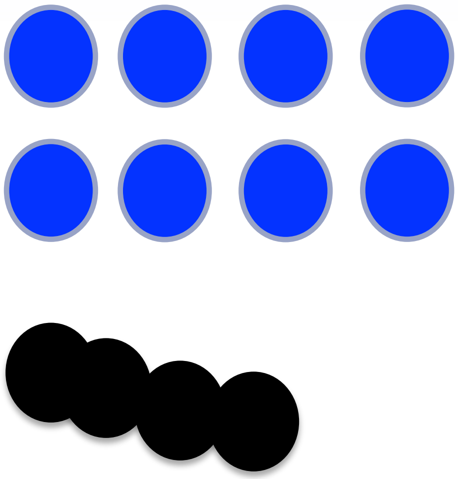
Robert L. Clay, SOS-18

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

Scalable Recovery through Spare Process Reserve

MPI processes for computation

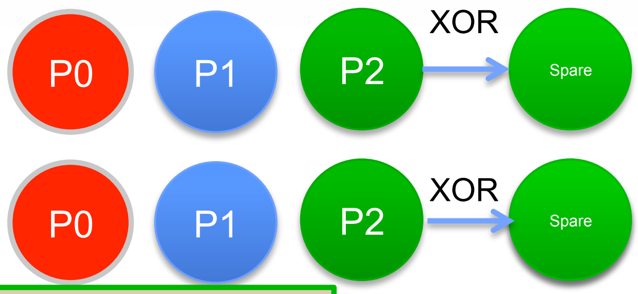


Spare MPI Processes

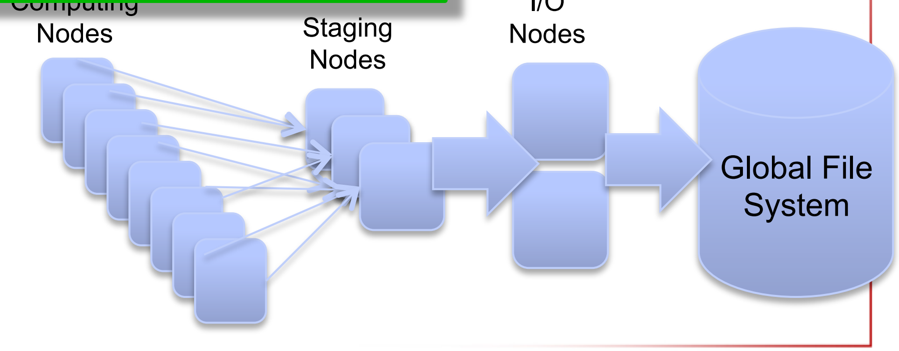
- ULFM-MPI only provides minimum set of APIs for process loss
  - Many apps need to remap the work after communicator shrink ☹
  - Vendor's MPI (such as Cray) does not support MPI_Comm_spawn
- Allocate hot spare process to replace the lost process
  - Can be used for the other resiliency features
- 3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

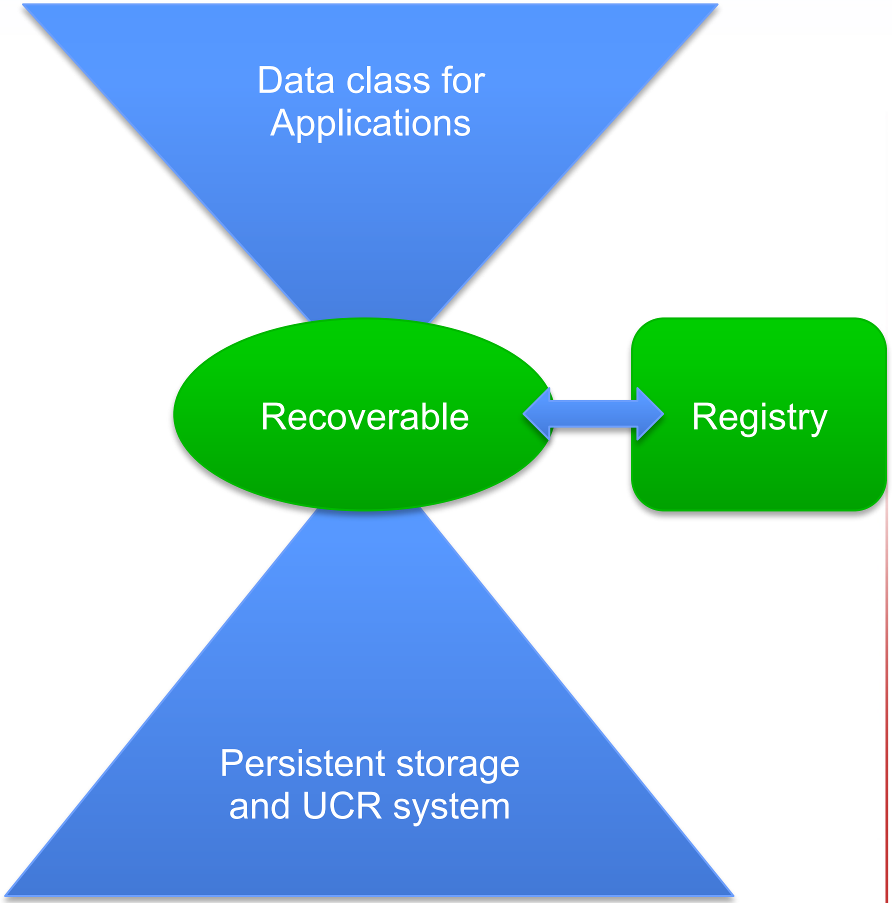


We employ in-memory storage of spare processes dedicated for checksum/parity



Scientific data structure for LFLR

- Object-oriented approach for scientific data structure
  - Trilinos and PETSc
- Recoverable class provides
  - Virtual methods for data specific recovery
  - Access to data redundancy protocol
    - Coordination with the spare process and persistent storage
  - Monitor allocated data objects
    - Recovery without specifying the data to be recovered
    - Simple for C++ meta-programming
    - Need “Destroy” or “Free” call for C/Fortran programming


Data class for Applications
Recoverable
Registry
Persistent storage and UCR system

Constructing Resilient Application: Case Study

- Iterative Algorithm
  - Time-stepping PDE
  - Nonlinear System Solver
  - Require multiple linear system solution

- Identify appropriate granularity for persistent storage access
  - Single iteration of linear system solver is too short
  - A few seconds per linear system solve in Sierra on 8192 Cray XE6 nodes

- Recovery
  - Crash in single linear system solve needs to recover the state outside linear system solver
    - E.g. time step, nonlinear step, mesh,
  - Recovery manager can recover all data and state
    - Spare process to keep chronological state
    - Data specific recovery
      - Matrix is regenerated from Mesh

Resilient Time-Stepping Solver

```
Create Mesh M
Compute Matrix A out of M
Save M in Persistent Storage
Do until the last time step
        $b_i$ and $b_{i-1}$ in Persistent Storage
        Get new $b_i$ from $x_{i-1}$ (Update Boundary Condition)
        Solve $Ax_i=b_i$ (Linear System Solution)
        if the linear system solver fails, try the same iterative step
end do
```

Process loss is checked periodically

- **Local vector is stored with the subscript (iteration count) info**
- **Allow linear system solver to crash or end up with wrong solution**
  - Process loss
  - Convergence failure due to silent data corruption
- **Repeat the same iteration when linear system solver fails**
  - Need to get x_{i-1} and b_{i-1}

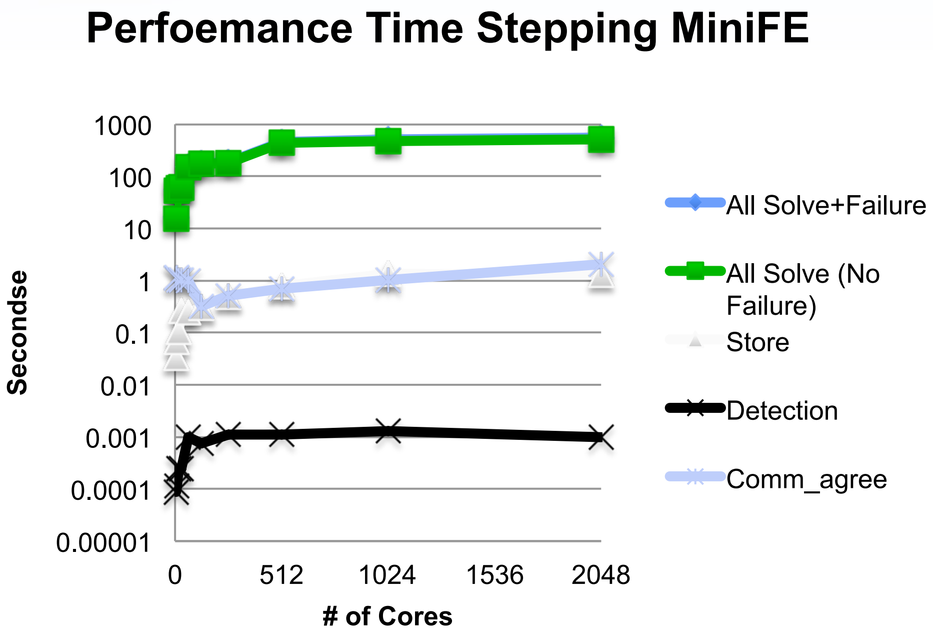
Preliminary Result

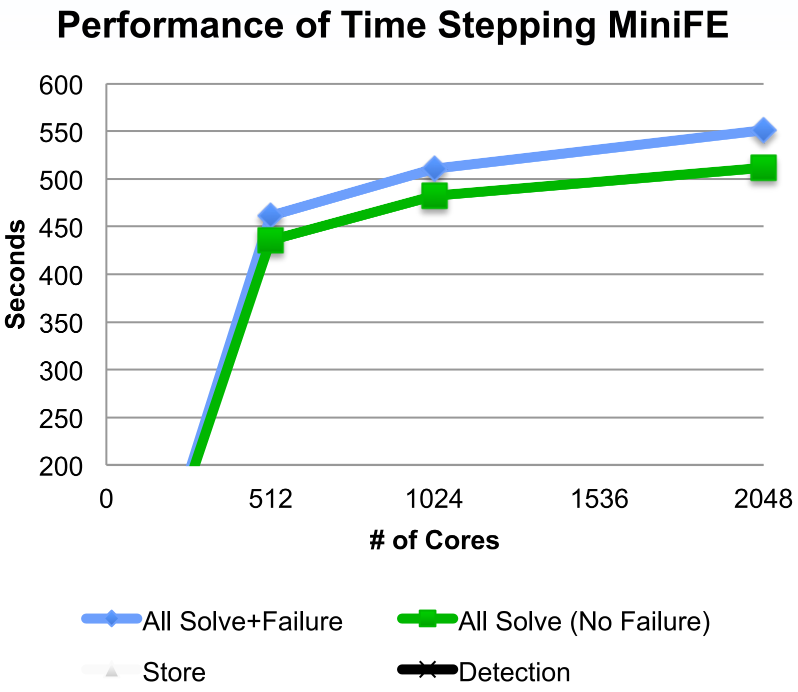
- Time Stepping PDE
  - 3D Finite Element
  - Multiple Linear System Solution
  - RHS is updated by LHS in the previous linear system solve

- 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

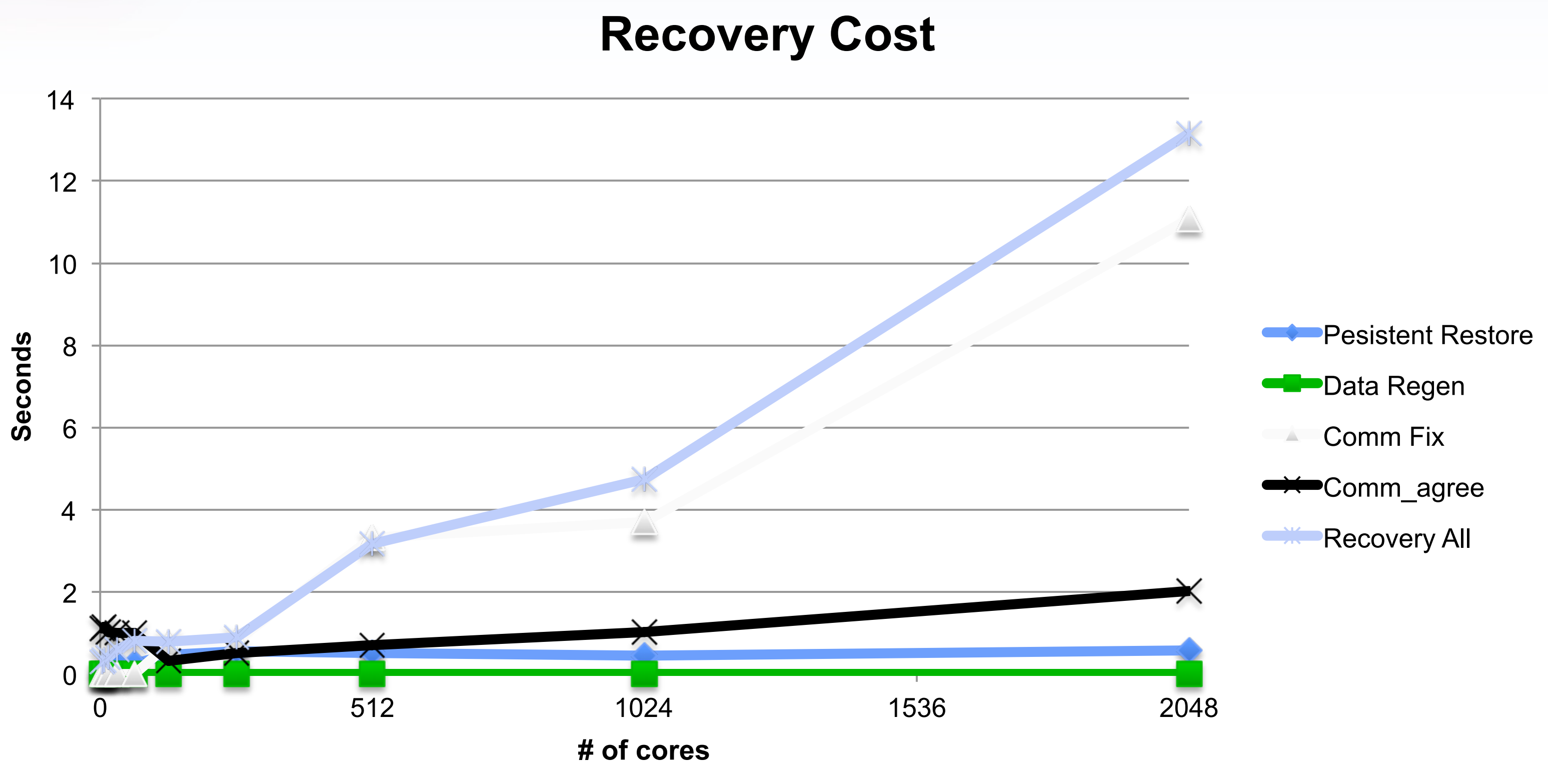
Results with MPI-ULFM





- 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



- **Negligible Cost for data recovery**
  - **Very scalable**
- **Scalability Issues in Communicator fix**

Sandia National Laboratories

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

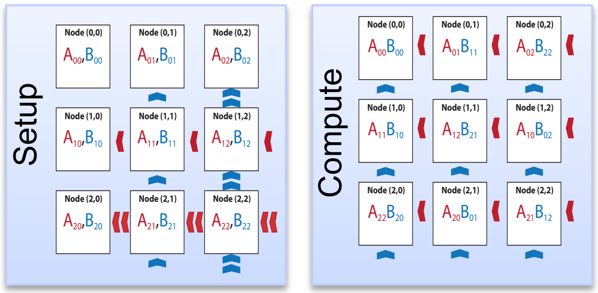
Also known as "MPI is fine". Also known as "MPI + X" where X is undefined, but it will work itself out over time. The real question may be whether the CSP BSP programming model will work well at exascale.

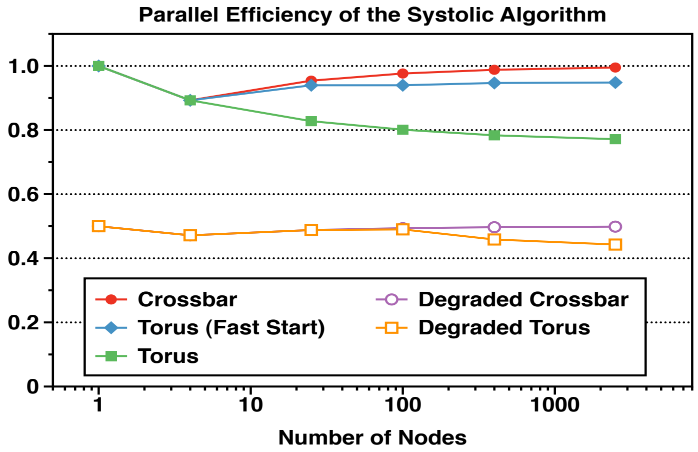
Existing SPMD programming models are inherently NOT fault tolerant

The move to exascale only makes things worse

- Global checkpoints no longer feasible
- Global collectives costly
- Applications/runtime must handle soft and hard failures
- Asynchronous execution to hide memory & I/O latency
- Deep memory hierarchies require tuning

Example: Systolic Matrix Multiplication

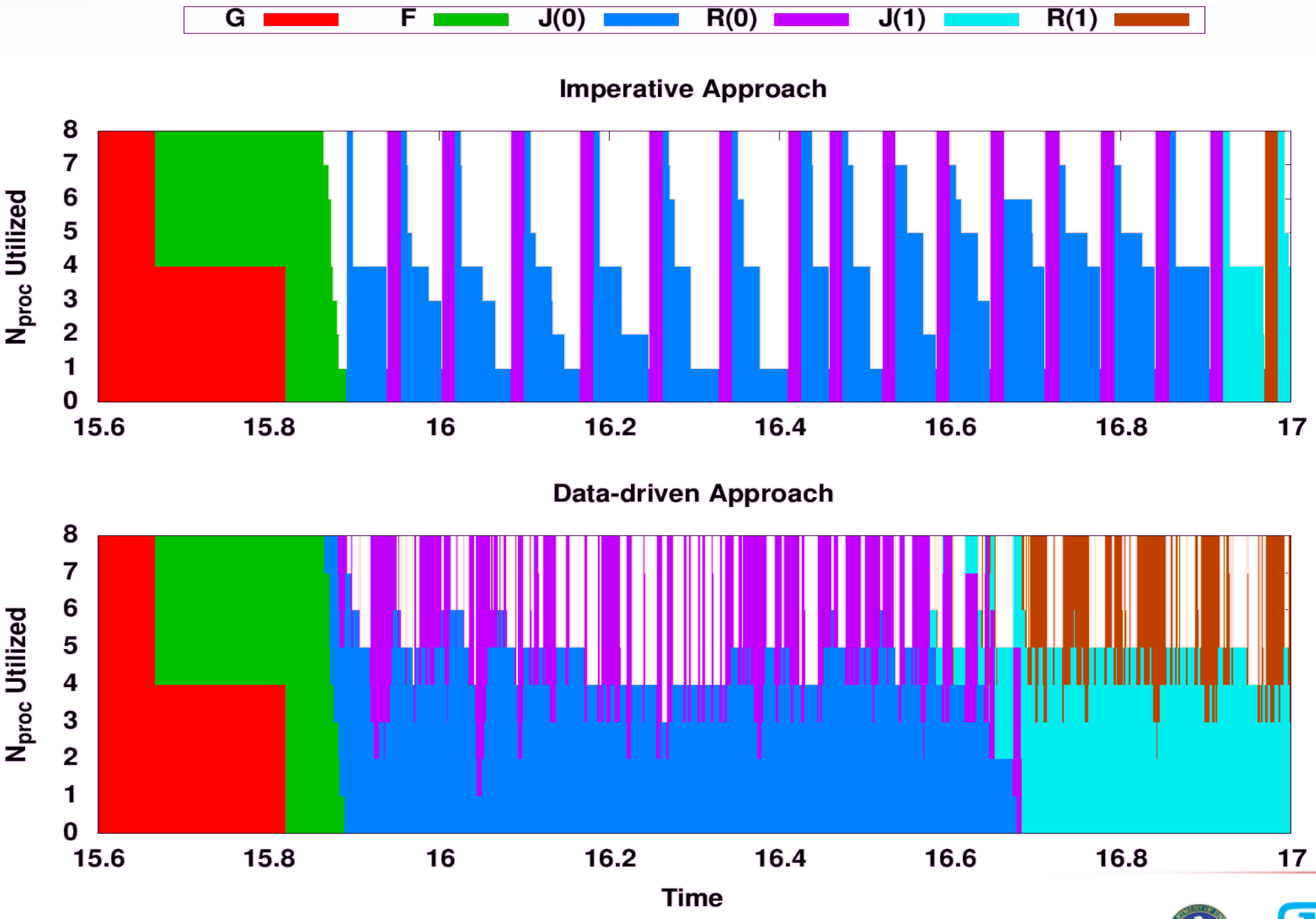

Setup
Node (0,0) A_{00}, B_{00}
Node (0,1) A_{01}, B_{01}
Node (0,2) A_{02}, B_{02}
Node (1,0) A_{10}, B_{10}
Node (1,1) A_{11}, B_{11}
Node (1,2) A_{12}, B_{12}
Node (2,0) A_{20}, B_{20}
Node (2,1) A_{21}, B_{21}
Node (2,2) A_{22}, B_{22}
Compute
Node (0,0) $A_{00}B_{00}$
Node (0,1) $A_{01}B_{11}$
Node (0,2) $A_{02}B_{22}$
Node (1,0) $A_{11}B_{10}$
Node (1,1) $A_{12}B_{21}$
Node (1,2) $A_{10}B_{02}$
Node (2,0) $A_{22}B_{20}$
Node (2,1) $A_{20}B_{01}$
Node (2,2) $A_{21}B_{12}$



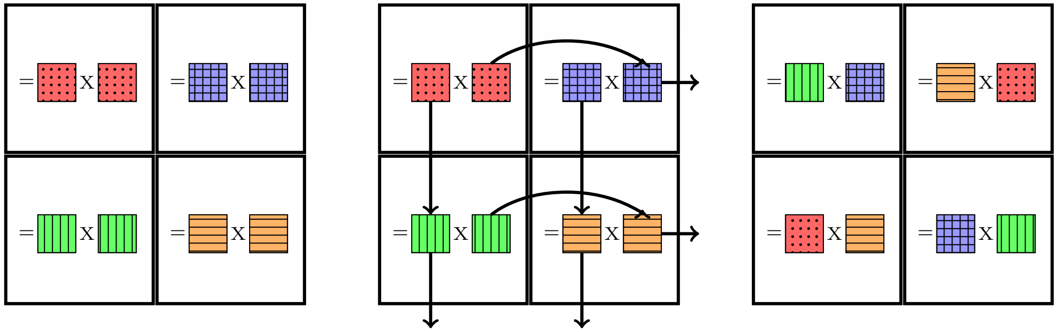
The implicitly synchronous systolic algorithm cannot recover from node degradation

C. L. Janssen, H. Adalsteinsson, J. P. Kenny, *Using simulation to design extreme-scale applications and architectures: programming model exploration*, ACM SIGMETRICS Performance Evaluation Review, **38**, pp. 4-8, 2011.

Simulated timings for 16 shells on 8 processors

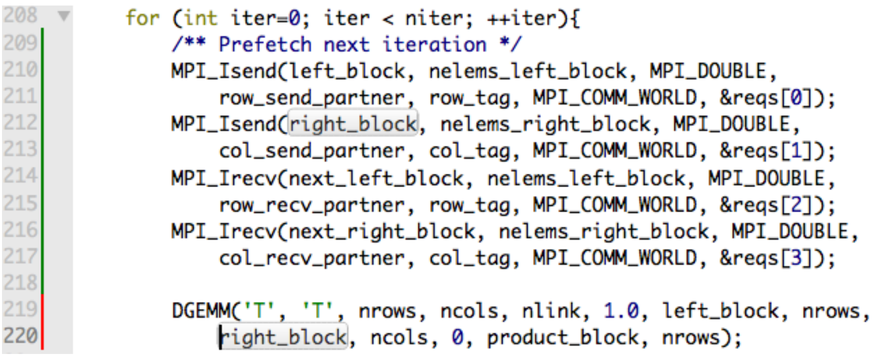


Programming model exploration for resilience with simulation



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

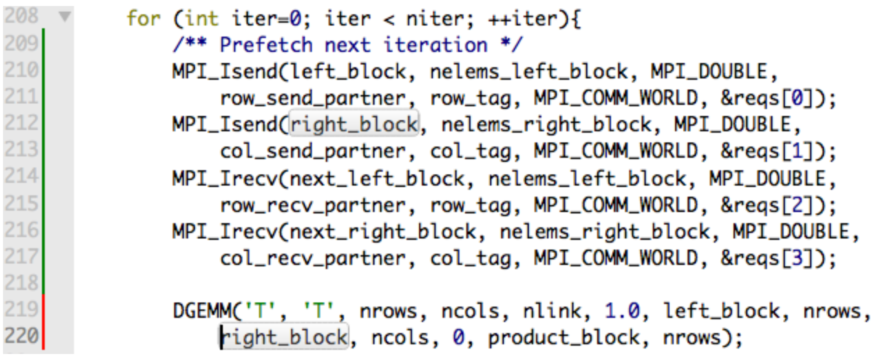
Actual MPI code



```
for (int iter=0; iter < niter; ++iter){
    /** Prefetch next iteration */
    MPI_Isend(left_block, nelems_left_block, MPI_DOUBLE,
        row_send_partner, row_tag, MPI_COMM_WORLD, &reqs[0]);
    MPI_Isend(right_block, nelems_right_block, MPI_DOUBLE,
        col_send_partner, col_tag, MPI_COMM_WORLD, &reqs[1]);
    MPI_Irecv(next_left_block, nelems_left_block, MPI_DOUBLE,
        row_recv_partner, row_tag, MPI_COMM_WORLD, &reqs[2]);
    MPI_Irecv(next_right_block, nelems_right_block, MPI_DOUBLE,
        col_recv_partner, col_tag, MPI_COMM_WORLD, &reqs[3]);

    DGEMM('T', 'T', nrows, ncols, nlink, 1.0, left_block, nrows,
        right_block, ncols, 0, product_block, nrows);
```

Simulator code



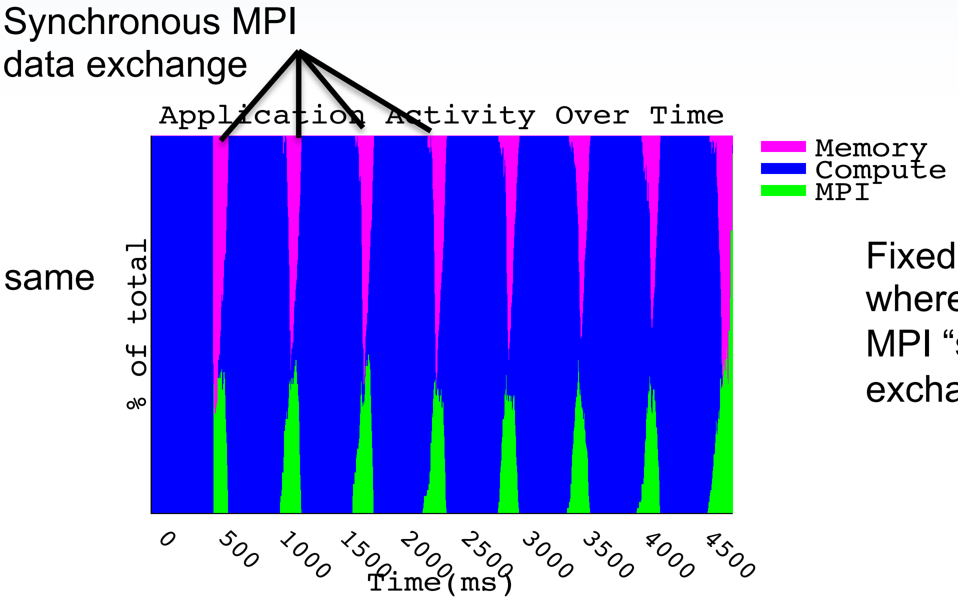
```
for (int iter=0; iter < niter; ++iter){
    /** Prefetch next iteration */
    MPI_Isend(left_block, nelems_left_block, MPI_DOUBLE,
        row_send_partner, row_tag, MPI_COMM_WORLD, &reqs[0]);
    MPI_Isend(right_block, nelems_right_block, MPI_DOUBLE,
        col_send_partner, col_tag, MPI_COMM_WORLD, &reqs[1]);
    MPI_Irecv(next_left_block, nelems_left_block, MPI_DOUBLE,
        row_recv_partner, row_tag, MPI_COMM_WORLD, &reqs[2]);
    MPI_Irecv(next_right_block, nelems_right_block, MPI_DOUBLE,
        col_recv_partner, col_tag, MPI_COMM_WORLD, &reqs[3]);

    DGEMM('T', 'T', nrows, ncols, nlink, 1.0, left_block, nrows,
        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!

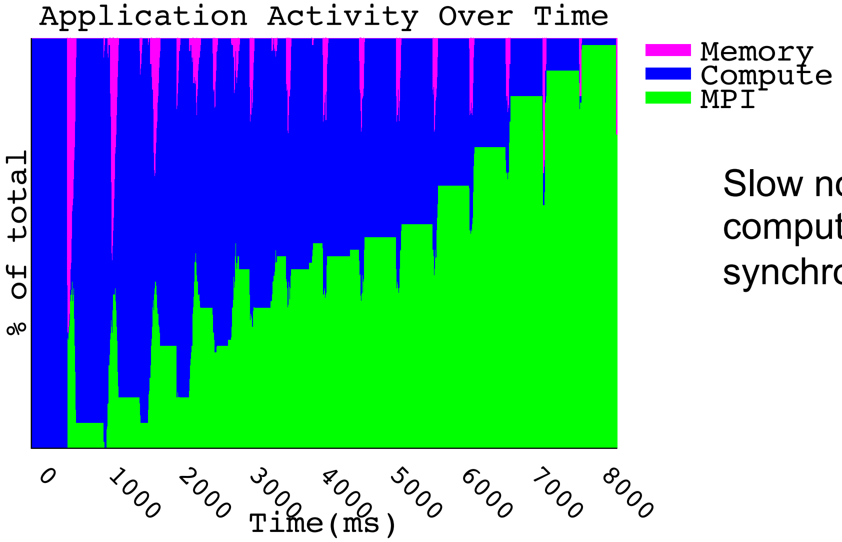
Programming model exploration for resilience : simulator results

If all nodes the same speed…



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

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

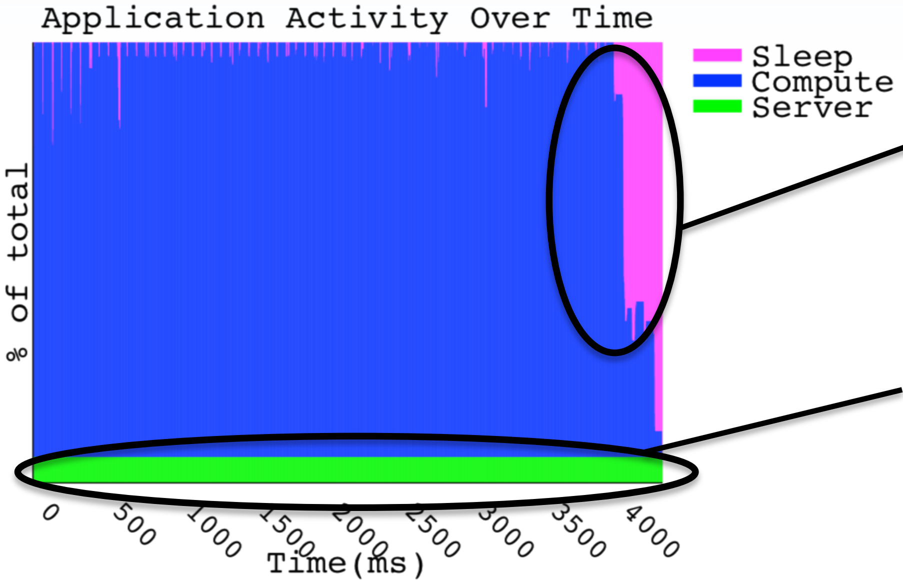


Slow node gradually chokes off computation due to MPI synchronization…

Robert L. Clay, SOS-18

Programming model exploration: asynchronous, task-DAG model

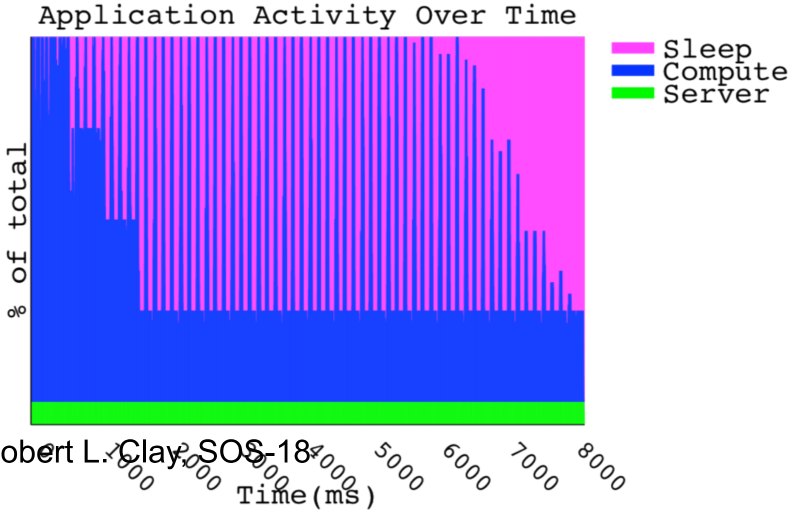
If all nodes the same speed…



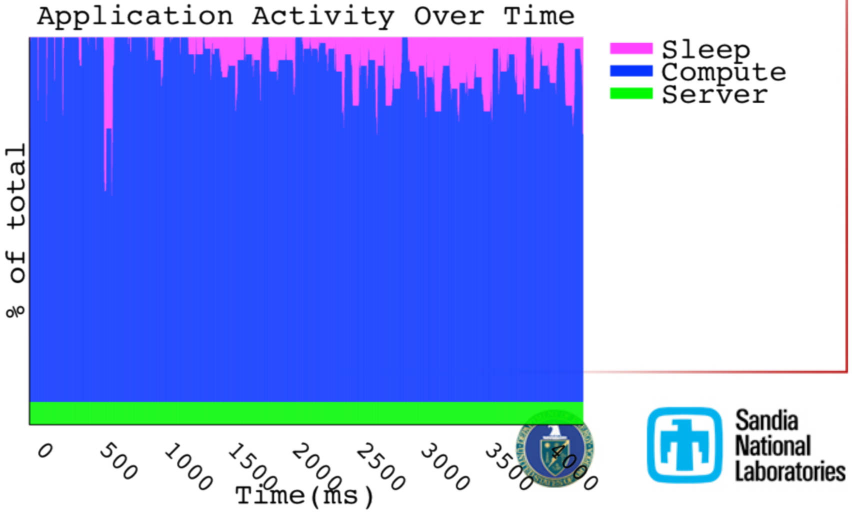
Termination detection/ work stealing needs to be optimized

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

If node slows down…



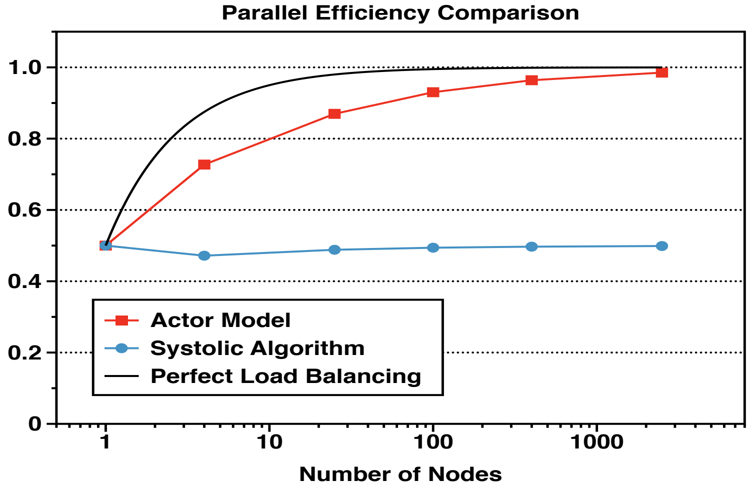
With load balancing…



Asynchronous many-task programming models are fault tolerant!

- Simulation permits straightforward investigation of alternative programming models
- Work-stealing approaches will play a role in dealing with large-scale machines lacking perfect homogeneity

Actor Model Matrix Multiplication (asynchronous, many task)

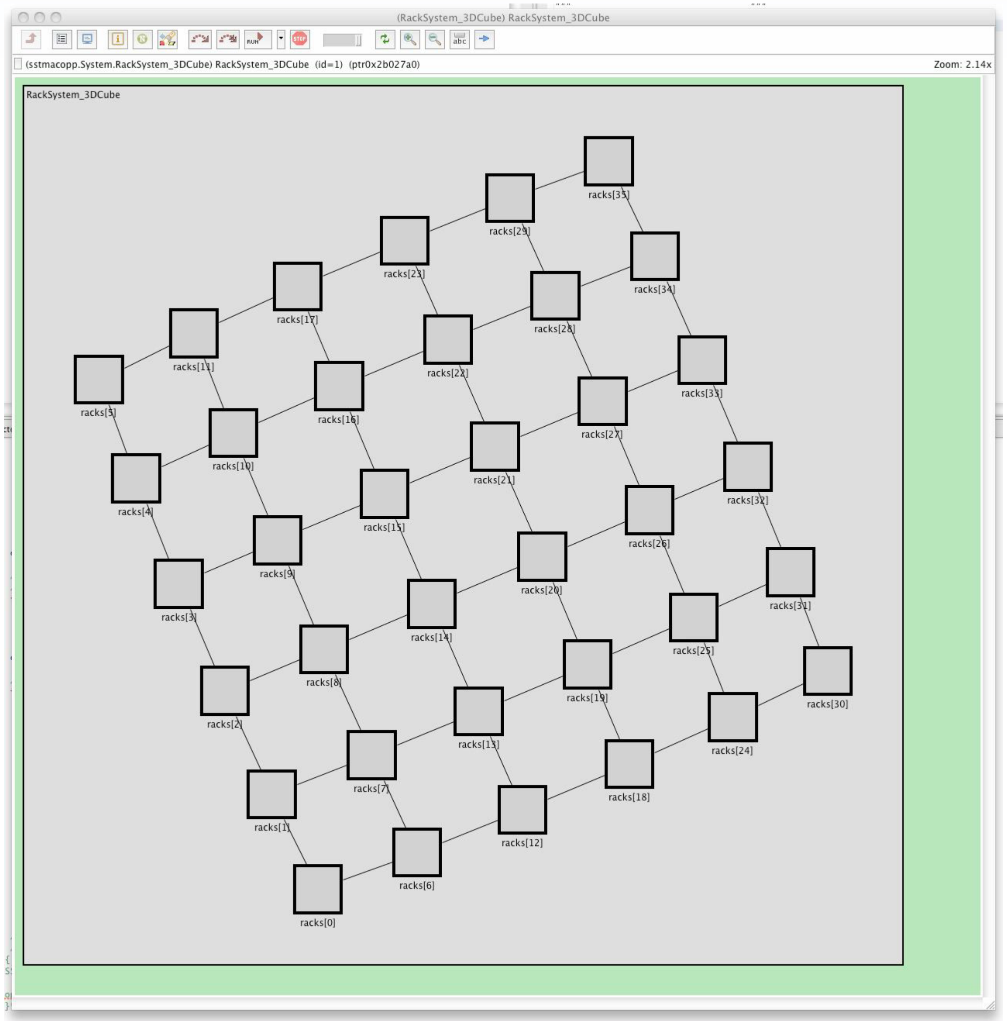


- Research Questions:
  - Is MPI+X (*global* checkpoint/restart) enough?
  - If not, what programming models can reach what scales?
  - If no programming model can reach scales of interest for a given application without algorithmic changes, how might algorithms be adapted?
  - Co-design of architecture tradeoffs between memory, I/O, power, and application performance

SST Experiment: Actor Load Balancing

Legend

- Black - initializing
- Green – working
- Yellow border – prefetching
- Red – idle
- Purple – work stealing



(RackSystem_3DCube) RackSystem_3DCube

(sstmacopp.System.RackSystem_3DCube) RackSystem_3DCube (id=1) (ptr0x2b027a0)

Zoom: 2.14x

Asynchronous, task-based programming model with work stealing balances load under dynamic conditions, including faults and degradation.

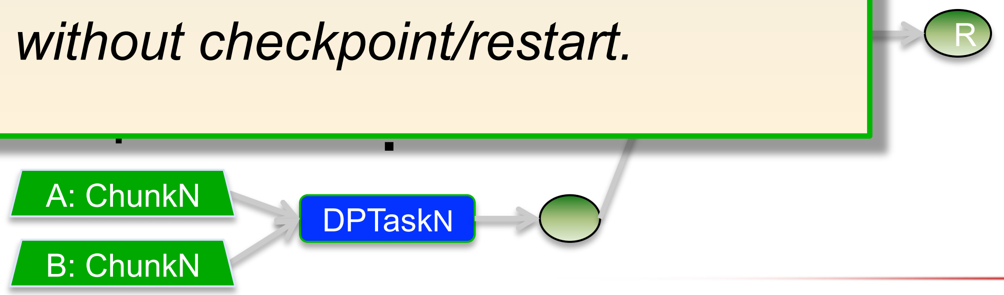
Can asynchronous, many-task programming models facilitate scalable resilience on extreme-scale systems?

- **Our approach:**
  - ***Dynamically scheduled, asynchronous tasks:*** maximize use of resources by load balancing and redistributing work from failed nodes
  - ***Locality and minimal data movement:*** move work to data; multithreaded, NUMA-aware scheduling on each node in distributed environment
  - ***Automatic data repair:*** silent data corruption is detected and repaired using triple modular redundancy or 2D checksums
  - ***Automatic task recovery:*** transaction-like semantics allow task replay after data is corrected

AMT programming models enable marching toward the correct solution in the face of both soft and hard faults without checkpoint/restart.

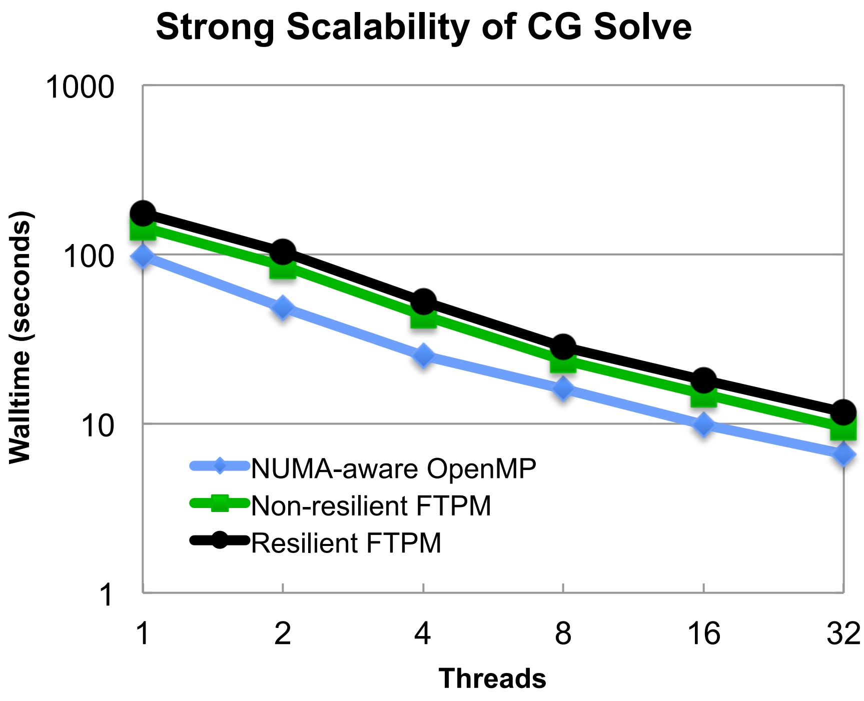
Example (

Dot proc
over-dec
and *B* to produce result *R*



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.

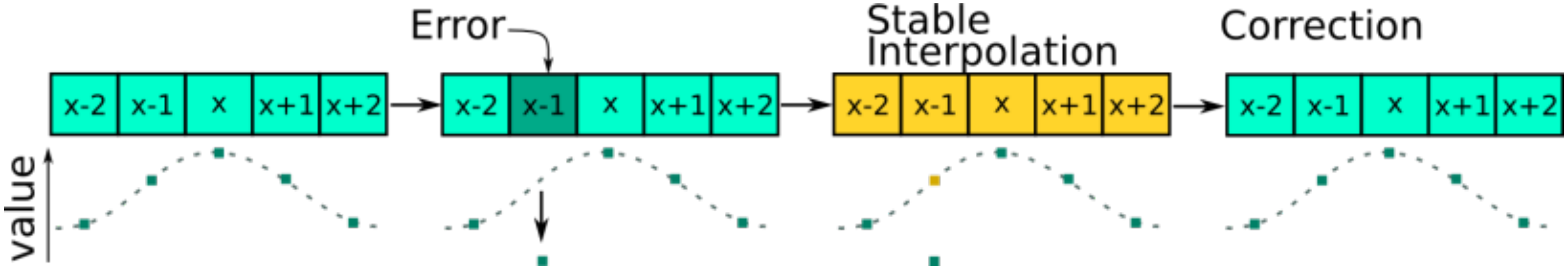
Maybe, maybe not.

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

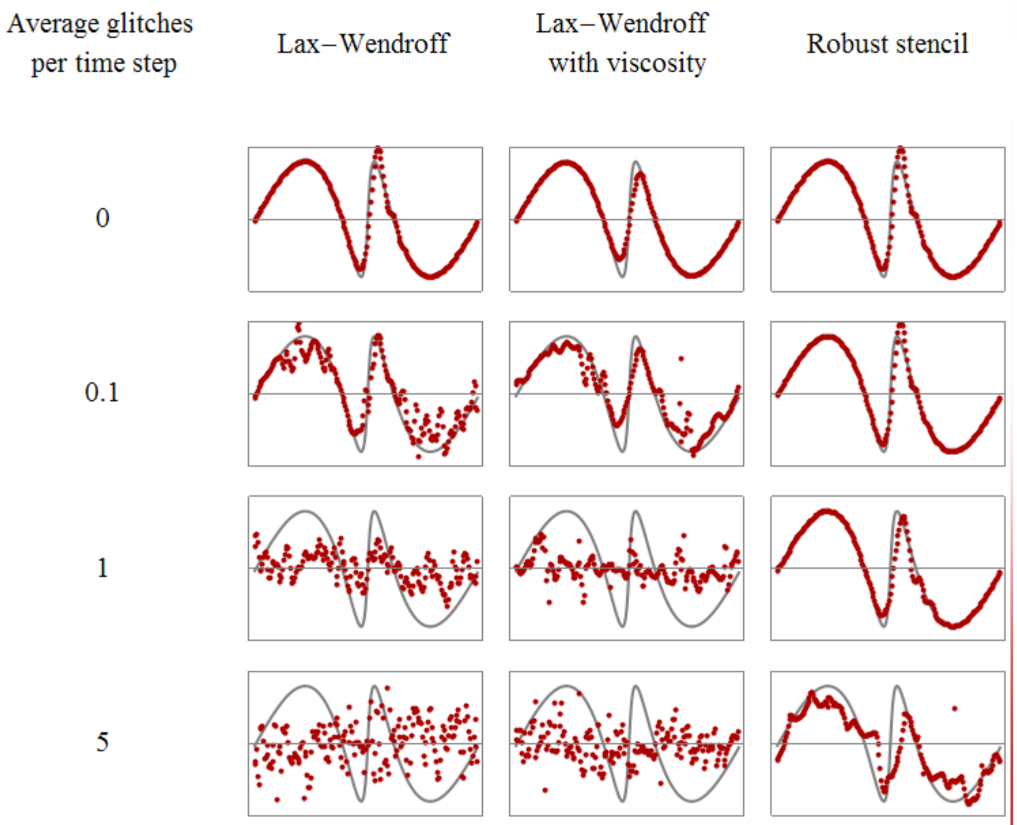
- **With increasing scale and with power limitations, errors can occur "silently" without indication that something is wrong**
- **Numerical algorithms already deal with error from truncation, etc.; specially designed algorithms can mitigate silent bit flips as well**



- **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

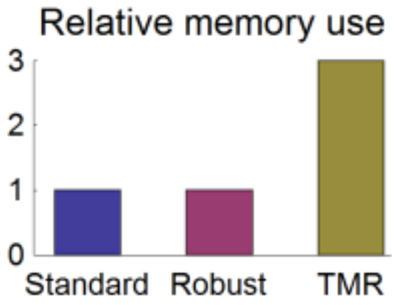
- **A simple 1D advection equation $\partial u/\partial t = \partial u/\partial x$ illustrates the behavior of finite-difference schemes**
- **The robust stencil here computes a second-order u at position *i* from one of these subsets after discarding the most extreme value:**
  - $\{\ i-3,\ i-1,\ i+1,\ i+3\ \}$
  - $\{\ i-2,\ i,\ i+2\ \}$
  - $\{\ i-1,\ i,\ i+1\ \}$

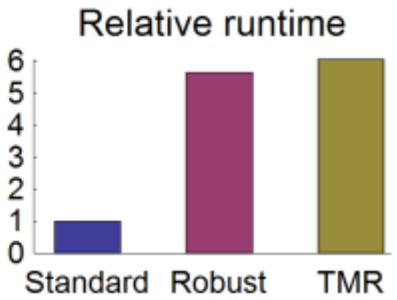


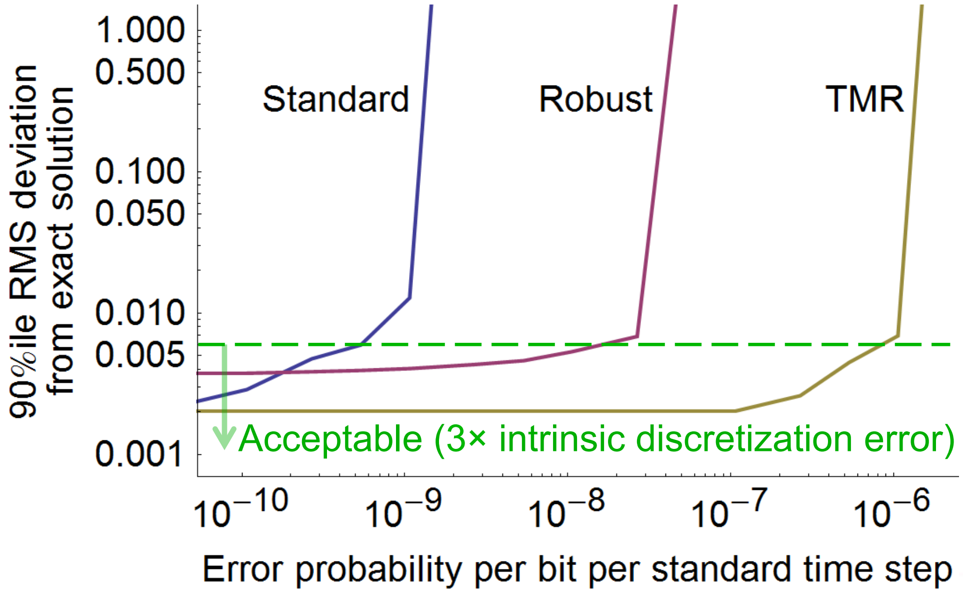
Simple demo in Mathematica

Bit-flip Injection at Machine Level Confirms Effectiveness of Our Robust Stencil

- Focus on silent-error models affecting **floating-point**
  - Relaxing FP correctness may benefit designs (e.g., GPUs)
- Test: During C++ PDE simulation, asynchronously perform raw **memory bit flips** in the FP solution array
  - Can also be a proxy for *processor* bit flips that corrupt FP ops
- Compare brute-force triple modular redundancy (TMR)



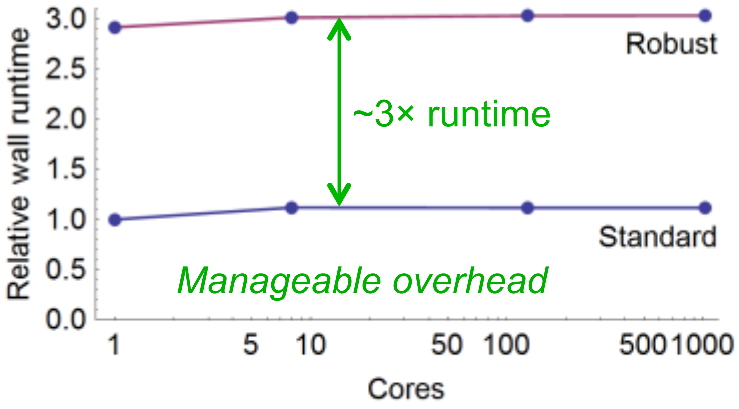




Here, the robust stencil provides substantial bit-flip tolerance at lower cost than TMR

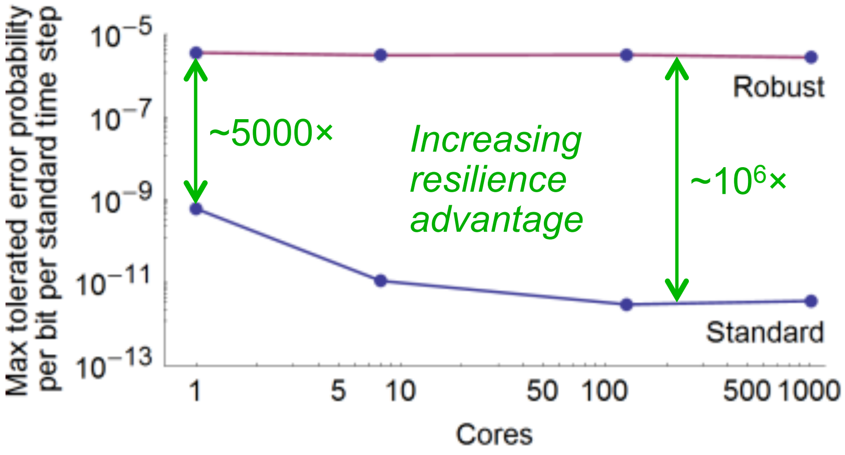
Preliminary Weak-Scaling Experiments Show Favorable Trends for Robust Stencil

- **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**



~3× runtime

Manageable overhead



~5000×

Increasing resilience advantage

~10^6×

- **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?

Acknowledgements

- Rob Armstrong (Robust Stencils)
- Janine Bennett (pmodels)
- Gilbert Hendry (SST/macro)
- 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