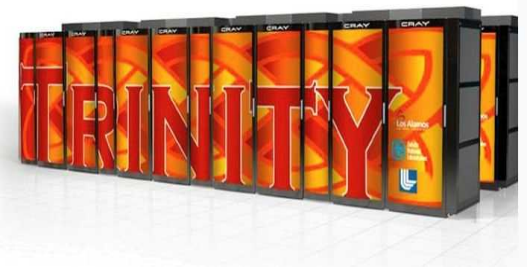
SAND2017-2702C




# Toward Resilient Task Parallel PDE Solvers

**Keita Teranishi**[1], Marc Gamell[2], Nicole Slattengren[1] and Manish Parashar[2]

Sandia National Laboratories, CA, USA

Rutgers University, NJ, USA

Funded by ASC CSSE Program.

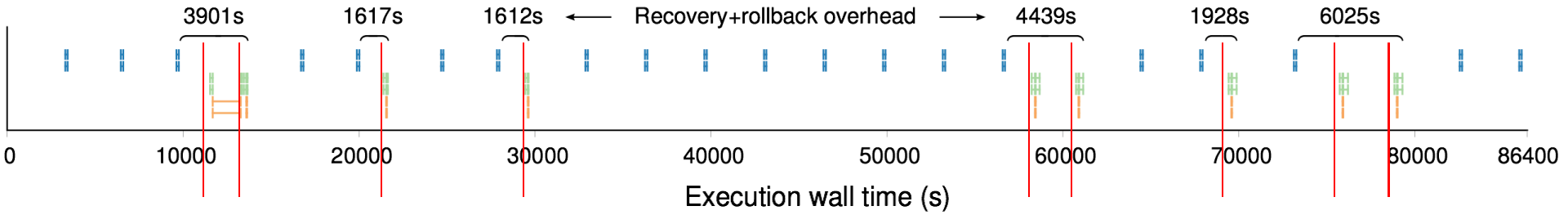
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

# Scalable Resilience for HPC applications

Sandia National Laboratories

- Future systems are expected to be less reliable
  - More components, shrinking, power etc.
- Anecdotes indicate that the majority of application failures happens at single node, i.e. local failures.
  - Exisitng Checkpoint-Restart approach is not a proportional response to local failures.
  - Undue cost associated with “Termiante + Restart”

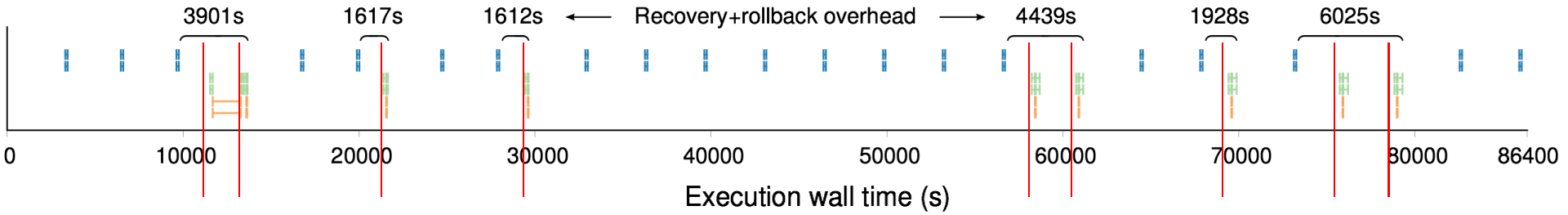
# Motivating Use Case – S3D Production Runs




- 24-hour tests using Titan (**125k cores**)
  - Reported MTBF of 8 hours
- 9 process/node failures over 24 hours
- Failures are promoted to **job failures**, causing all 125k processes to exit
- Checkpoint (5.2 MB/core) has to be done to the PFS

| | | *Total cost* |
|---|---|---|
| Checkpoint (per timestep) | **55 s** | 1.72 % |
| Restarting processes | **470 s** | 5.67 % |
| Loading checkpoint | **44 s** | 1.38 % |
| Rollback overhead | **1654 s** | 22.63 % |
| Total overhead | | **31.40 %** |

# Motivating Use Case – Problem Summary




- Current **checkpoint cost, ~1 min**
- Total **recovery+rollback cost, ~36 min**

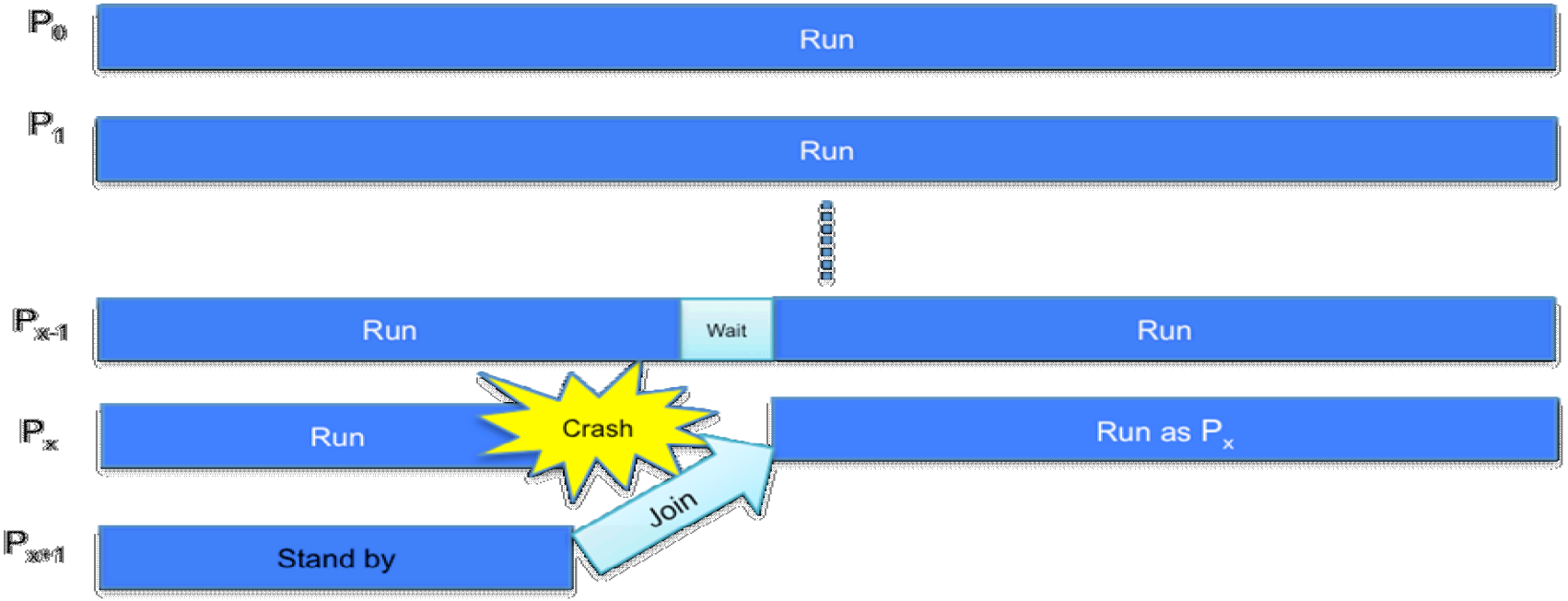
→ Infeasible

*Traditional C/R or runtime-based offline techniques are*

- *not efficient in current systems*
- *not possible in future systems*

# Our Solution: Online Failure Recovery

- Software framework to augment existing apps with resilience capability
  - The remaining processes stay alive with **isolated** process/node failure
  - Multiple implementation options for recovery
    - Roll-back, roll-forward, asynchronous, algorithm specific, etc.
  - **Hot Spare Process for recovery**

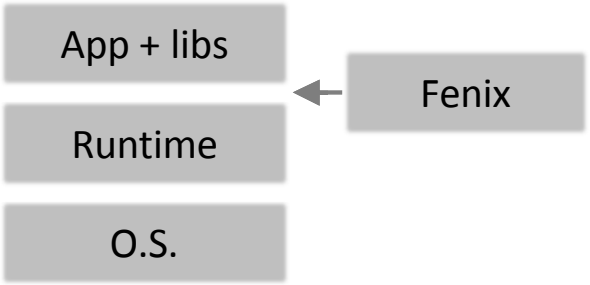


# Solution #1 :Global Online Recovery

Sandia National Laboratories

**1. Process recovery:** Recover failures without promoting to job failures

- Framework for online, semi-transparent recovery
- Targets SPMD, message passing applications
- Tolerates **hard failures (spare or spawned ranks)**
- Keep process memory (may contain valuable data or checkpoints)




**2. Data recovery:** Optimize checkpointing

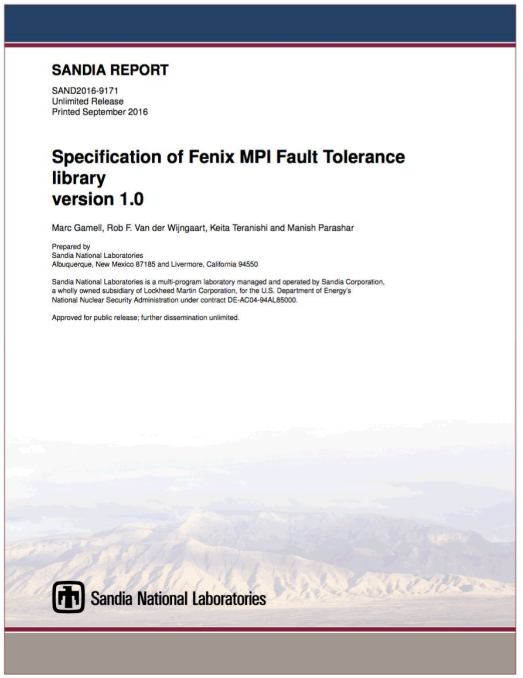
- Store application-specific data in-memory
- **Coordinate checkpoint creation implicitly**
  - Fully coordinated checkpointing ⟶ consistency, but barrier-like constructs
  - Uncoordinated checkpointing ⟶ efficient, but no consistency guarantees

| Implicitly Coordinated Checkpointing |
| --- |
| **Applications know consistent points!** |

  - Implicit coordination: create consistent checkpoints without communication
  - Consistency guaranteed by **checkpointing at the same "logical" time**

# Fenix 1.0 Specification (SAND2016-9171)

- Fault Tolerant Programming Framework for MPI Applications
  - Separation between process and data recovery
    - Allows third party software for data recovery
    - Multiple Execution Models
  - Process recovery
    - Extend **MPI-ULFM (prototype of MPI Fault Tolerance)** to shield the users from low-level MPI features
    - Process recovery through spare process pool
    - Process failure is checked at PMPI layer and recovery happens under the cover
  - Data recovery
    - In-memory data redundancy
    - Multi-versioning (similar to GVR by U Chicago &ANL)


SANDIA REPORT

SAND2016-9171
Unlimited Release
Printed September 2016

Specification of Fenix MPI Fault Tolerance library
version 1.0

Marc Gamell, Rob F. Van der Wijngaart, Keita Teranishi and Manish Parashar

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

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

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Sandia National Laboratories

# S3D Modifications

- Only 35 new, changed, or rearranged lines in S3D code

Only 35 new, changed, or rearranged lines in S3D code

Original vs Fenix-enabled

```
REAL :: stime
REAL, ALLOCATABLE, DIMENSION(:,:,:, :) :: yspc

! Other initializations

! Setup MPI, Cartesian MPI grid, etc.
call initialize_topology(6, nx, ny, nz, &
                         npx, npy, npz, &
                    iorder, iforder )

! Setup grid - scale arrays for stretched grid
! used in derivatives, coordinates useful for
! generating test data
call initialize_grid(6)

! Allocate derivative arrays
call initialize_derivative(6)

allocate(T(nx,ny,nz))
allocate(P(nx,ny,nz))
allocate(deriv_result(nx,ny,nz,nslvs,3))
allocate(deriv_sum(nx,ny,nz,nslvs))
allocate(yspc(nx,ny,nz, nslvs))
allocate(wdot(nx,ny,nz, nslvs) )
allocate(rho(nx,ny,nz))    !HK
 ! Setup test data
xshift = (xmax - xmin)*0.1
yshift = (ymax - ymin)*0.1
zshift = (zmax - zmin)*0.1

do k = 1, nz
 do j = 1, ny
  do i = 1, nx
   !HK in Kelvin
   T(i,j,k)  = 1000.0*(sin(x(i)-xshift)*sin(y(j)-yshift)*sin(z(k)-
zshift)) + 1500.0
   yspc(i,j,k,:) = 0.01
   yspc(i,j,k, 1) = 0.1
   yspc(i,j,k, 2) = 0.7
   yspc(i,j,k,3) = 0.05
   yspc(i,j,k,4) = 0.05
   yspc(i,j,k,nslvs) = 1.0 - sum( yspc( i,j,k, 1:nslvs-1) )
   P(i,j,k) = 12.0*pres_atm  !HK. 12 atm expressed in SI units
  enddo
 enddo
enddo

TIMESTEP: do itime = 1, ntsteps

  ! ITERATE AND UPDATE YSPC

enddo TIMESTEP
```

```
#include "fenix_f.h"
REAL, TARGET :: stime
  REAL, ALLOCATABLE, DIMENSION(:,:,:, :), TARGET :: yspc
  INTEGER ckpt_itime, ckpt_yspc;
  INTEGER, TARGET :: world;

itime = 1
! Other initializations

allocate(T(nx,ny,nz))
  allocate(P(nx,ny,nz))
  allocate(deriv_result(nx,ny,nz,nslvs,3))
  allocate(deriv_sum(nx,ny,nz,nslvs))
  allocate(yspc(nx,ny,nz, nslvs))
  allocate(wdot(nx,ny,nz, nslvs) )
  allocate(rho(nx,ny,nz))    !HK

call MPI_Init(ierr)
```

nction

y module

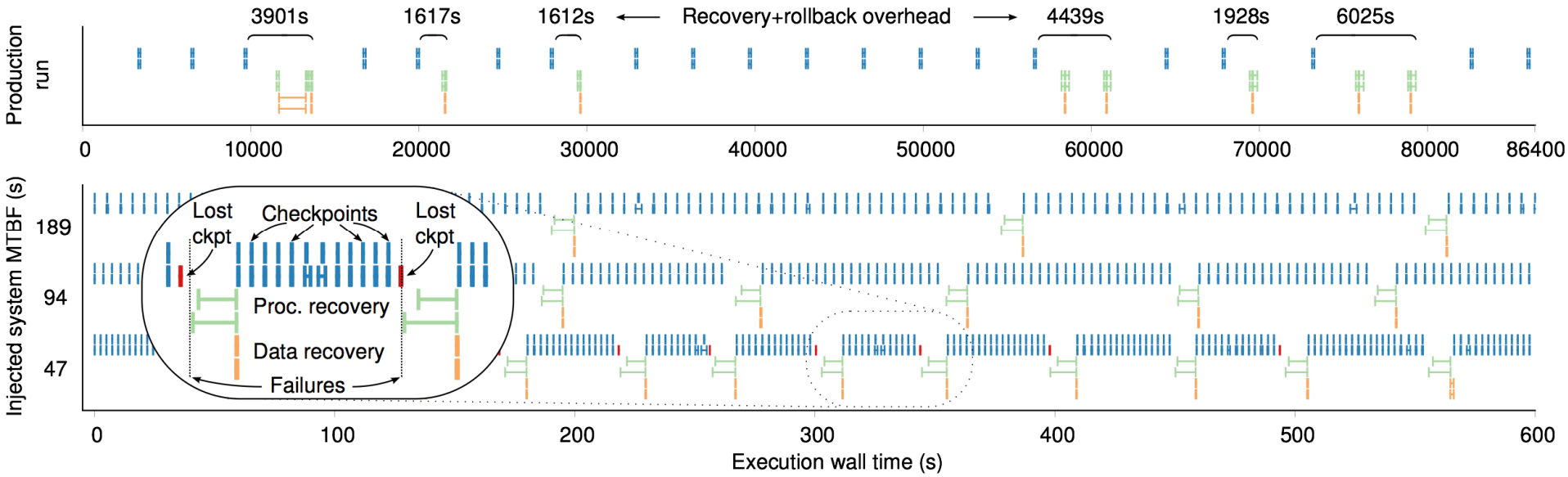
```
  if(process_status.eq.FENIX_PROC_NEW) then
    yspc(i,j,k,:) = 0.01
    yspc(i,j,k, 1) = 0.1
    yspc(i,j,k, 2) = 0.7
    yspc(i,j,k,3) = 0.05
    yspc(i,j,k,4) = 0.05
    yspc(i,j,k,nslvs) = 1.0 - sum( yspc( i,j,k, 1:nslvs-1) )
  endif
    P(i,j,k) = 12.0*pres_atm  !HK. 12 atm expressed in SI units
  enddo
 enddo
enddo

do
  if(mod(itime-1,CHECKPOINT_PERIOD).eq.0) then
    call FT_Checkpoint(ckpt_yspc);
    call FT_Checkpoint(ckpt_itime);
  endif

  ! ITERATE AND UPDATE YSPC

  itime = itime + 1
  if( itime .gt. ntsteps ) exit
enddo
```

vs Fenix-enabled

```
call MPI_Comm_size(MPI_COMM_WORLD, npes, ierr)
! Create communicator duplicate for global calls
call MPI_Comm_dup(MPI_COMM_WORLD, gcomm, ierr)

! Create communicators for the x, y, and z directions
call MPI_Comm_split(gcomm, mypy+1000*mypz, myid, xcomm,ierr)
call MPI_Comm_split(gcomm, mypx+1000*mypz, myid, ycomm,ierr)
call MPI_Comm_split(gcomm, mypx+1000*mypy, myid, zcomm,ierr)

! Create MPI Comminicators for boundary planes.  This is
used in the Boundary conditions
call MPI_Comm_split(gcomm, xid, myid, yz_comm, ierr)
call MPI_Comm_split(gcomm, yid, myid, xz_comm, ierr)
call MPI_Comm_split(gcomm, zid, myid, xy_comm, ierr)
```

```
call MPI_Comm_rank(world, myid, ierr)
call MPI_Comm_size(world, npes, ierr)
! Create communicator duplicate for global calls
gcomm = world

! Create communicators for the x, y, and z directions
call MPI_Comm_split(gcomm, mypy+1000*mypz, myid, xcomm,ierr)
call MPI_Comm_split(gcomm, mypx+1000*mypz, myid, ycomm,ierr)
call MPI_Comm_split(gcomm, mypx+1000*mypy, myid, zcomm,ierr)
call FT_Comm_add(xcomm);
call FT_Comm_add(ycomm);
call FT_Comm_add(zcomm);

! Create MPI Comminicators for boundary planes.  This is
used in the Boundary conditions
call MPI_Comm_split(gcomm, xid, myid, yz_comm, ierr)
call MPI_Comm_split(gcomm, yid, myid, xz_comm, ierr)
call MPI_Comm_split(gcomm, zid, myid, xy_comm, ierr)
call FT_Comm_add(yz_comm);
call FT_Comm_add(xz_comm);
call FT_Comm_add(xy_comm);
```

# Global Online Recovery – Results




| | *MTBF* | *Total overhead* |
|---|---|---|
| Production | **2.6 h** | **31 %** |
| Global recovery | 189 s | 10 % |
| Global recovery | 94 s | 15 % |
| Global recovery | **47 s** | **31 %** |

- Uses S3D (scientific application)
- Titan Cray XK7 (#3 on top500.org)
- Injecting node failures (16-core failures)

# Solution #2: Local Online Recovery

Sandia National Laboratories

- Fenix-1.0 is a first step toward local recovery
  - Avoid global termination and restart
  - All processes rollback to the Fenix_Init() call
  - Natural for algorithms and applications that makes collective calls frequently
- Some applications fit more scalable recovery model
  - Stencil Computation
  - Master-Worker execution model
- Solution: Local Online Recovery

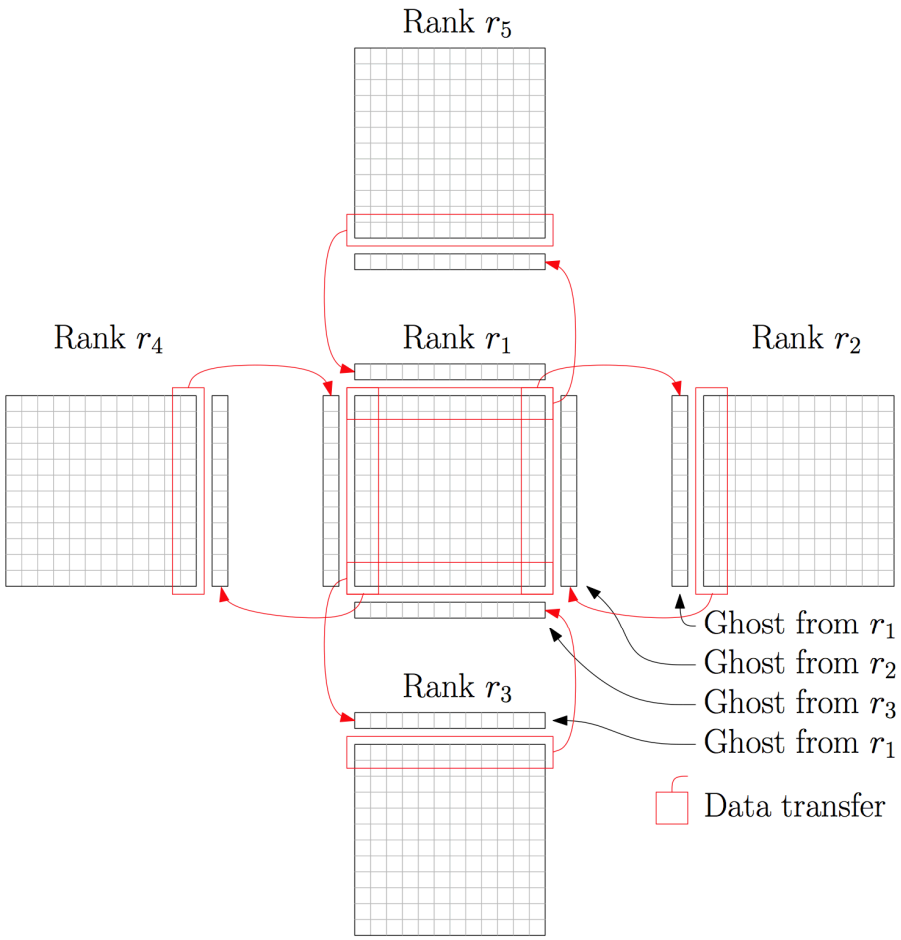
# Local Recovery Methodology

Sandia National Laboratories

1. Replace failed processes
2. Rollback to the last checkpoint (only replaced processes)
3. Other processes continue with the simulation

- How do we guarantee consistency?
  - Implicitly coordinated checkpoint
  - Log messages since last checkpoint in local sender memory
  - Message logging has been studied in MPI fault tolerance and Actor Execution Model (Charm++)
    - Performance may not be optimal for many parallel applications
    - **Stencil computation provides built-in message logging == Ghost Points**
- Implemented in new framework: **FenixLR**

# Target: Stencil-based Scientific Applications

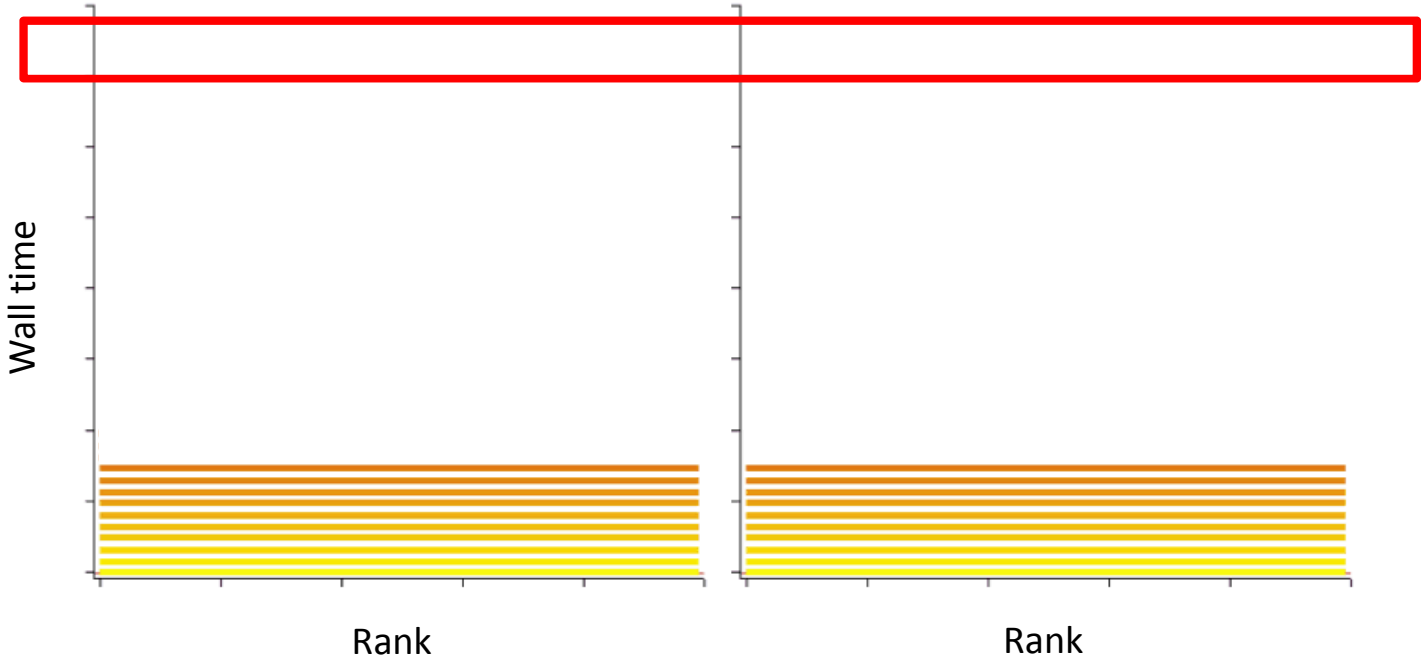



- Application domain is partitioned using a block decomposition across processes
- Typically, divided into iterations (*timesteps*), which include:
  - Computation to advance the local simulated data
  - Communication with immediate neighbors
- Example: PDEs using finite-difference methods, S3D

# Performance Model of Local Recovery

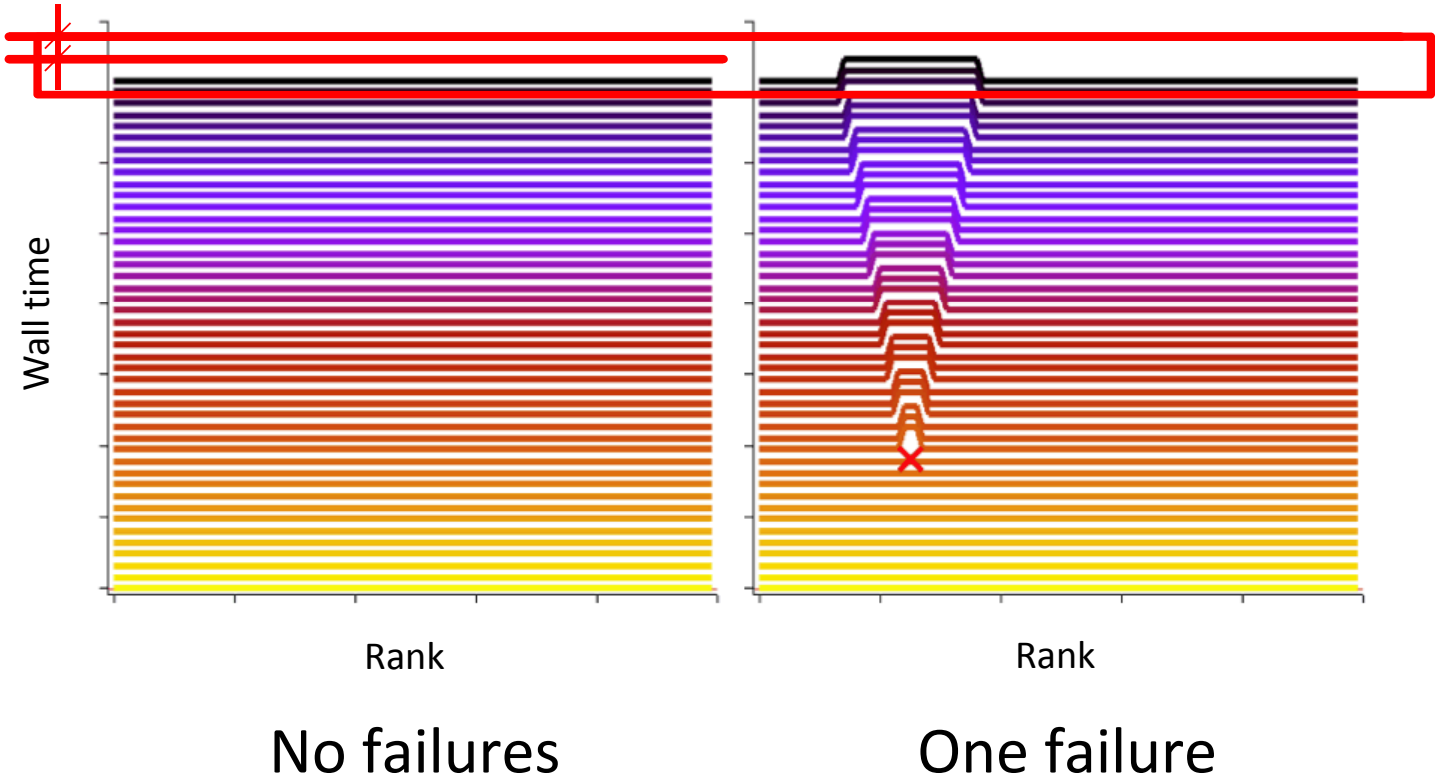
Sandia National Laboratories

Simulated execution of a 1D PDE



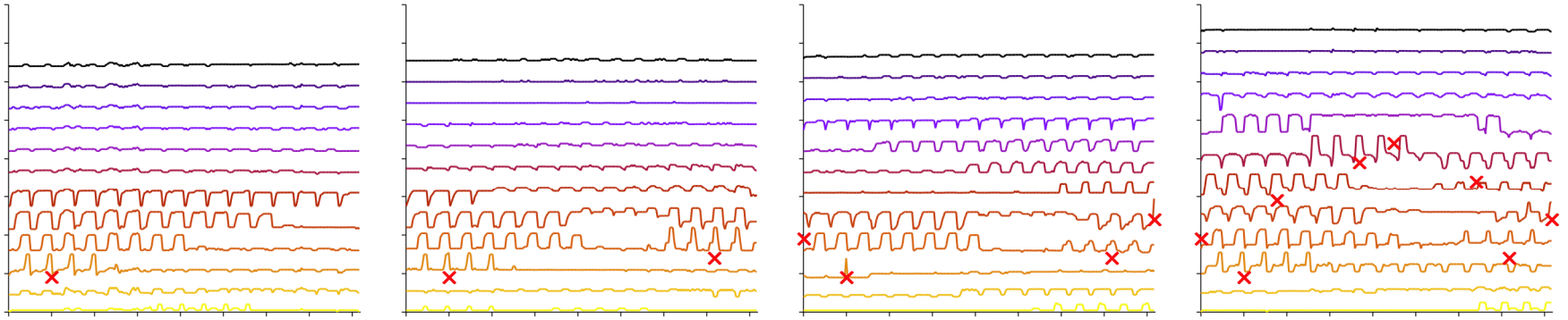
# Effect of Multiple Failures with Local Recovery

Simulated execution of a 1D PDE

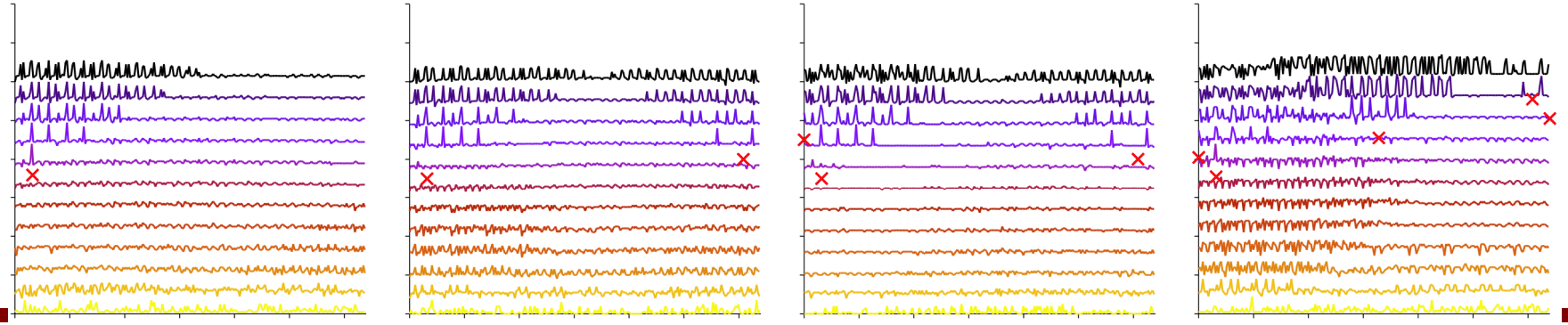


# Experimental Evaluation with S3D

- Same experiment executed injecting different number of failures
- X axis is rank number, but more complex to see than 1D, because 3D domain is mapped to core ranking in a linear fashion
- Note that total overhead is as if only one failure occurred (except in 4224c 8f)



(a) 4224c 1f (b) 4224c 2f (c) 4224c 4f (d) 4224c 8f



(q) 64128c 1f (r) 64128c 2f (s) 64128c 3f (t) 64128c 5f

# Experimental Evaluation

Sandia National Laboratories

**Goal**

- Evaluate local recovery techniques using S3D on Titan to show
  - Low overhead while recovering from node failures every 5 seconds
  - Failure recovery is scalable
  - Recovery overhead is not proportional to system size

**Experiments**

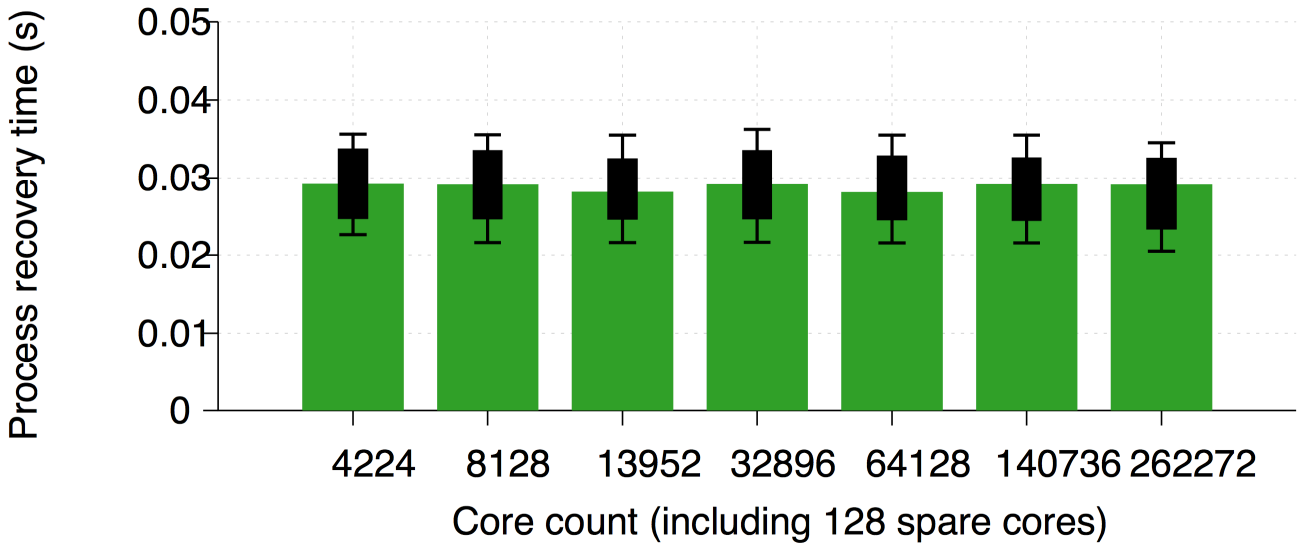
- **Recovery scalability** up to 262272 cores
- **Total overhead** of fault tolerance

**Methodology**

- Study the overheads related to the recovery processes
- Compare local vs global recovery
- Failure recovery cost can be decomposed into:
  - **Environment recovery**
  - Checkpoint fetching from neighbor (scalable, 130MB/core)
  - Rollback cost (average of 1/2 iteration time, O(2.5 seconds), scalable)

# Recovery Scalability

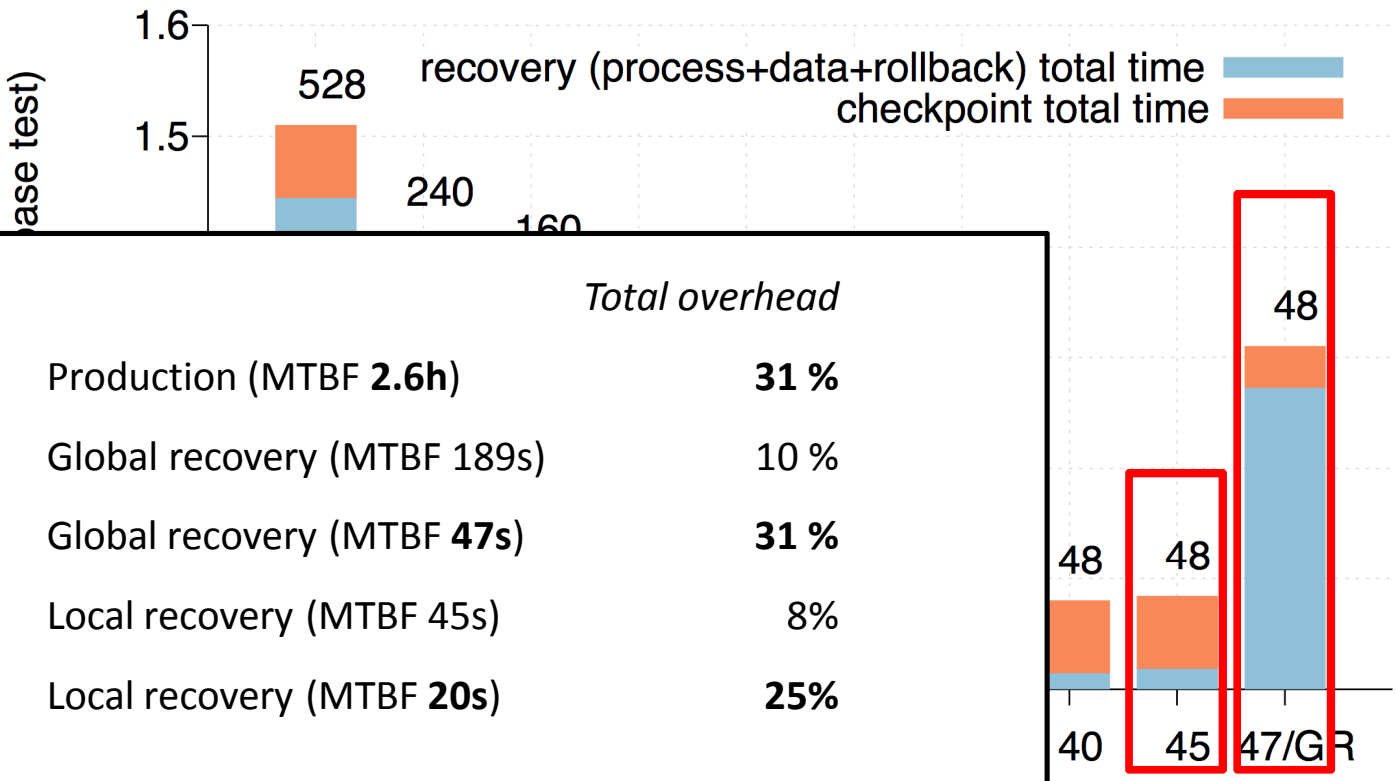
- Using MTBF of 10s
- Core count from 4224 to 262272 (including 128 spare cores)
- Result shows the average recovery time for all failures injected.




- Concl
  - Process recovery time is independent of system size
  - Good scalability

# Total Overhead of Fault Tolerance

- End-to-end time vs failure-free, checkpoint-free time
- Overall overhead:
  - Checkpoint
  - Process/da recovery
  - Rollback
- 4096 cores + spare cores
- Right-most bar global recovery with MTBF of 4
- Local recovery has scalability advantages over global recovery

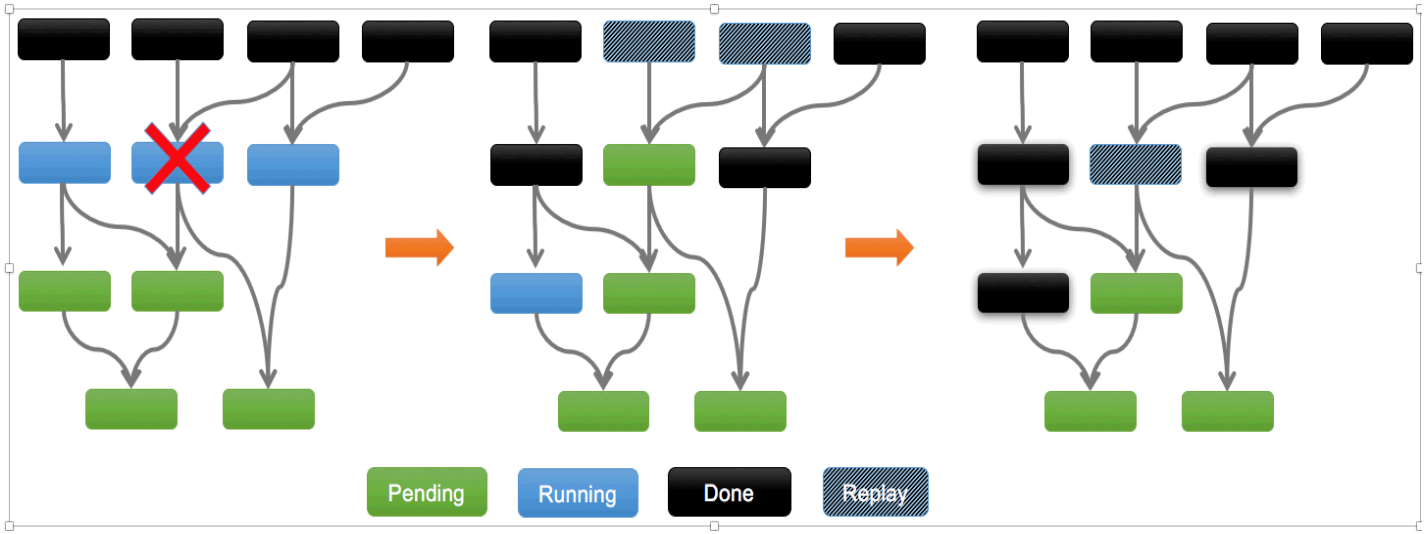



| | *Total overhead* |
|---|---|
| Production (MTBF **2.6h**) | **31 %** |
| Global recovery (MTBF 189s) | 10 % |
| Global recovery (MTBF **47s**) | **31 %** |
| Local recovery (MTBF 45s) | 8% |
| Local recovery (MTBF **20s**) | **25%** |

- **Local recovery is superior to global recovery in this scenario:**
  - **compare MTBF 45s (8%)**
  - **with MTBF 47/GR (31%)**

# Solution #3: Toward Resilient Asynchronous Many Task (AMT) Parallel Execution Model

Sandia National Laboratories



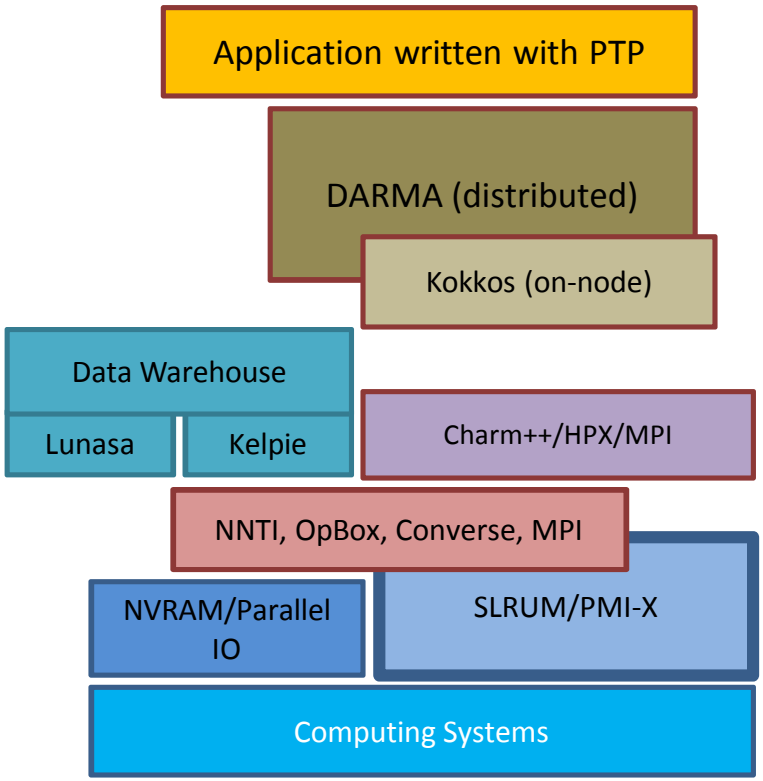

- AMT allows
  - Concurrent task execution
  - Overlap of communication and computation
  - Over-decomposition of Data
- Node/Process Failure is manifested as loss of task and data
  - Generic model for online local recovery
  - Recovery is done through task replay

# Research Challenges

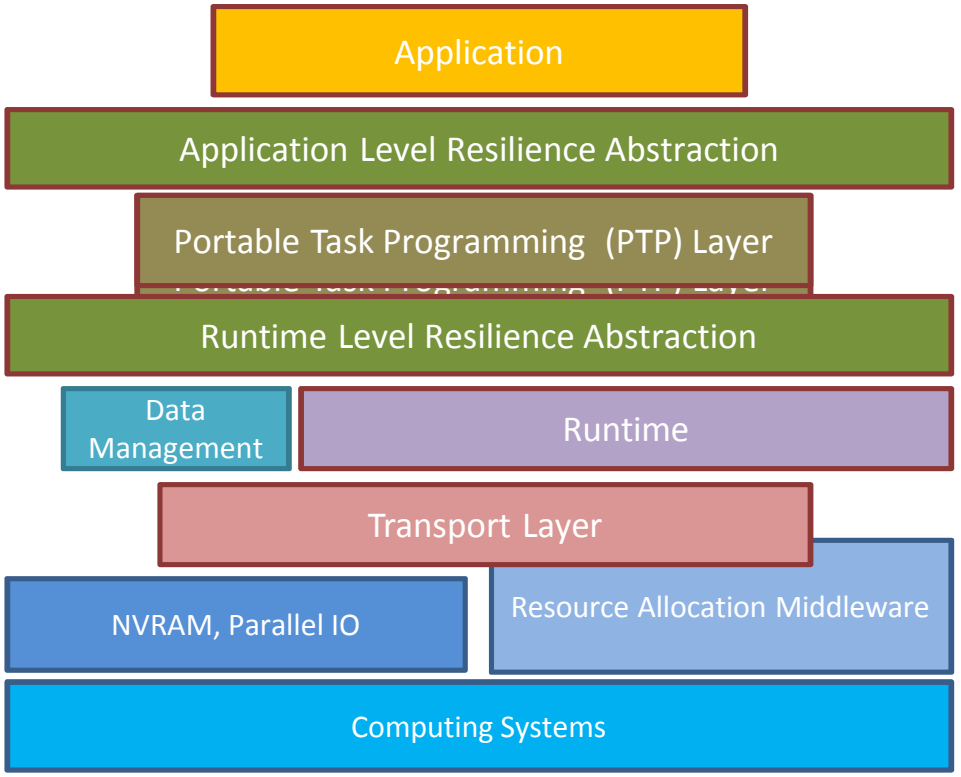
Sandia National Laboratories

- How to express the resilience to the users?
- How to maintain the persistence of task and data?
  - How to maintain the persistence of task scheduler?
  - Need to store all runtime information?
    - Similar to system-level checkpoint (TASCEL project at PNNL)
- How to reschedule task replay?
- How to build a framework?

# Ongoing Work: Sandia ATDM Software Stack



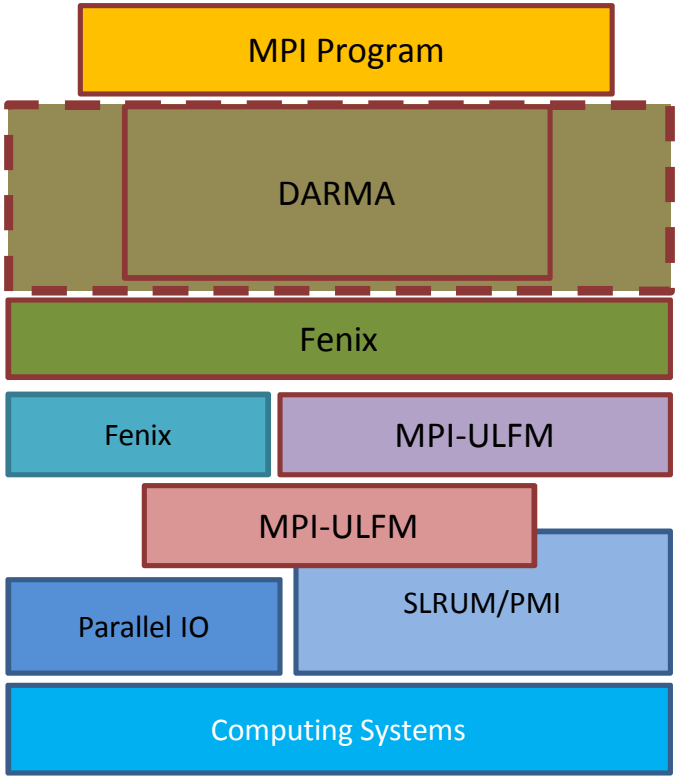

Sandia-ATDM Software Stack

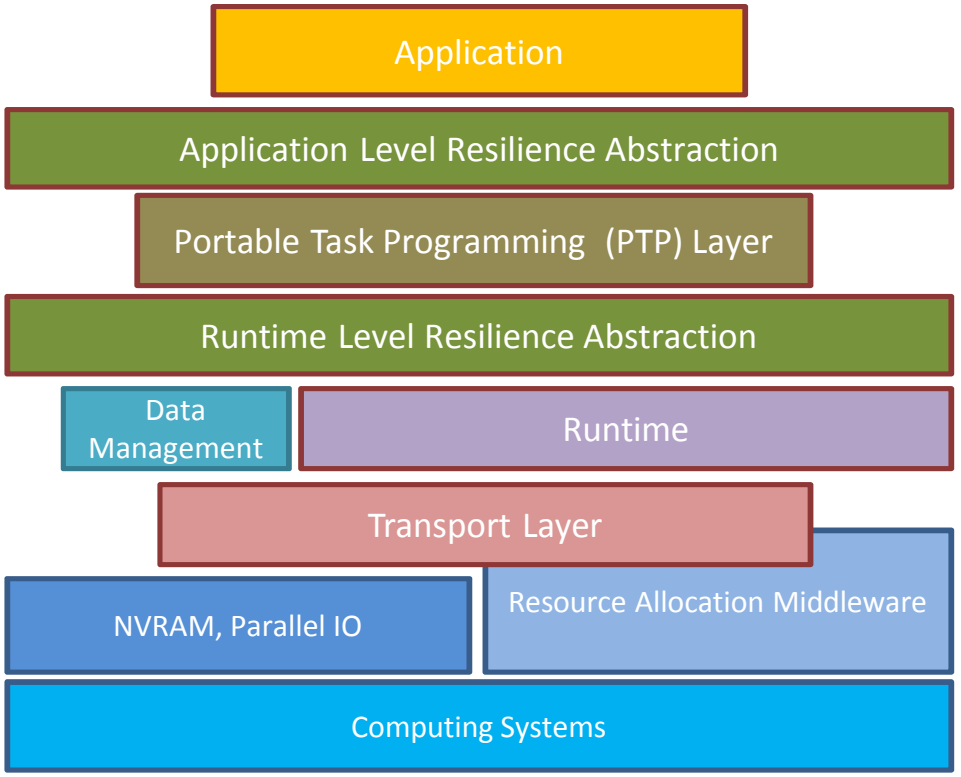



ATDM Software Stack

- ATDM Program (Advanced Technology Development and Mitigation) to prepare the software for future HPC systems
- Resilience AMT will serve abstraction in the frontend and backend to bridge between applications and runtime

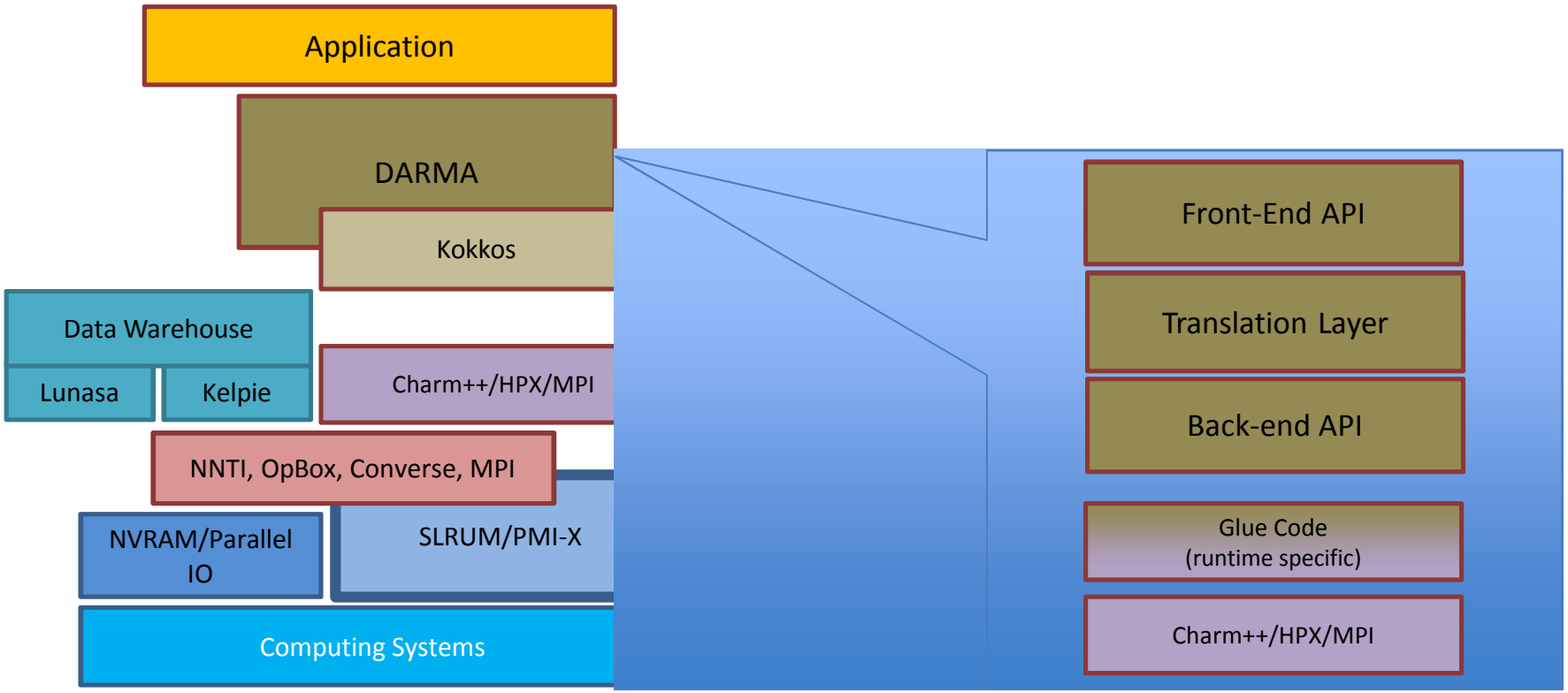
# Example: Fenix(1.0)-ULFM Model




- Fault Tolerant Message Passing for SPMD
  - Assuming MPI-ULFM or its equivalent
  - No task parallel programming layer
  - Fenix takes major responsibility in data management. A better model is desired.
- Solution: Resilient DARMA, Sandia's AMT Framework

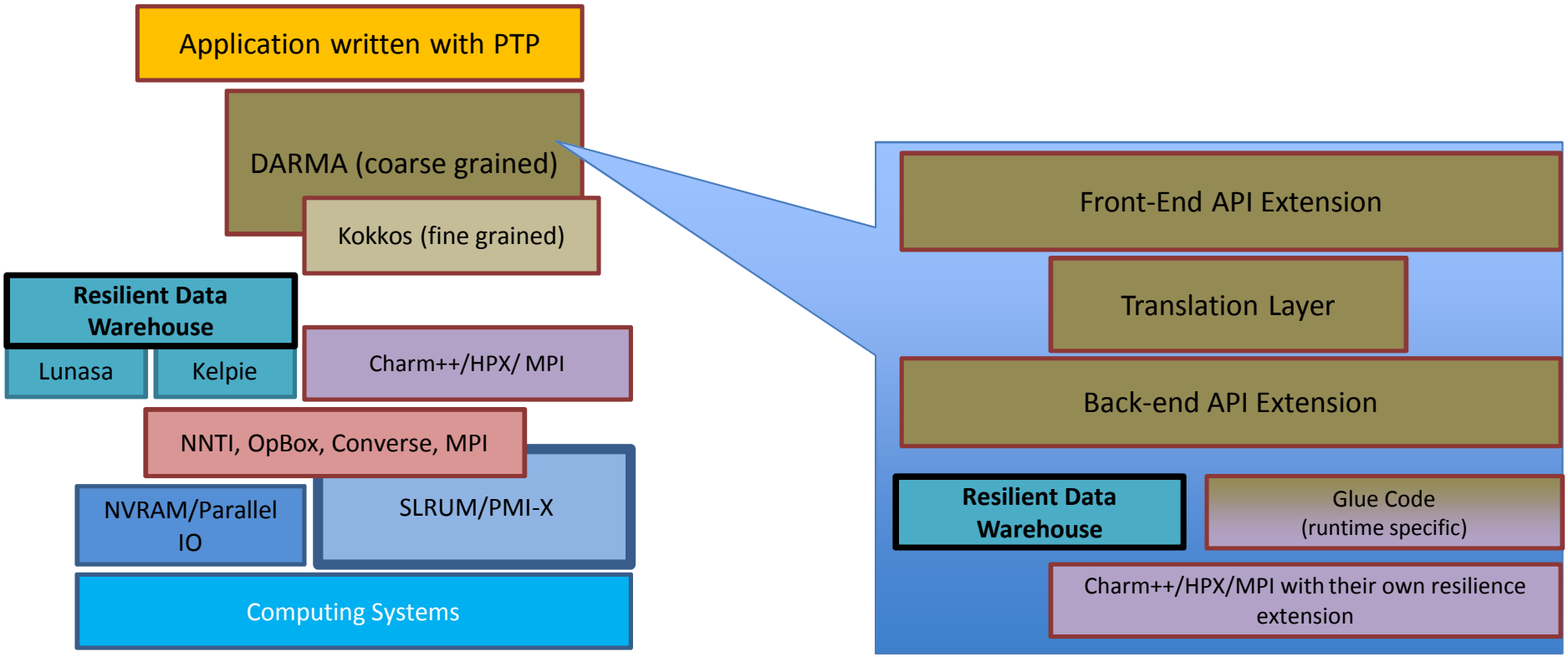
# Architecture of DARMA




Sandia-ATDM Software Stack

- Provides an abstraction in multiple levels for:
  - Ease of programming
    - Sequential Semantics
    - Publish/Fetch semantics for data exchange between tasks
  - Support of multiple runtime options
    - DARMA's data and task are translated to the runtime specific representations

# Resilient DARMA




- API extensions
  - Frontend allows the users to describe application-specific based resilience
  - Backend leverages resilient data warehouse for persistent storage for efficient task replay
  - Backend leverages resilience specific to runtime
    - Checkpoint capability of Charm++
    - Fenix+MPI

# Conclusion

Sandia National Laboratories

- Scalable Application Recovery at Scale
  - Extend Fault-Tolerant MPI prototype
    - Hot spare procesess
    - In-meory checkpoiting
  - Application specific message logging to allow localized online recovery
- Future work explore resilience in AMT runtime
  - Require vertical integration
  - Lots of opportunities on the horizon

# Acknowledgement


- Janine Bennet, Robert Clay, Michael Heroux and Jeremiah Wilke (Sandia National Labs)
- Sanjay Chatterjee and Vivek Sarkar (Rice U)
- George Bosilca, Aurélien Bouteiller and Thomas Herault