SAND2022-11865C

# Integrating process, control-flow, and data resiliency layers

## Using a hybrid Fenix/Kokkos approach

**Matthew Whitlock**, Nic Morales, George Bosilca, Aurelien Bouteiller, Bogdan Nicolae, Keita Teranishi, Elisabeth Giem, Vivek Sarkar

IEEE Cluster - September 9, 2020

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-NA0003525.0003525.

# Sneak Peak

- Consider Resilience technologies in layers
- Redesign layers for seamless, **application-level** integration
- Achieve combined goals of
  - Performance, Scalability
  - Customizability
  - Programmability, Maintainability

# Roadmap

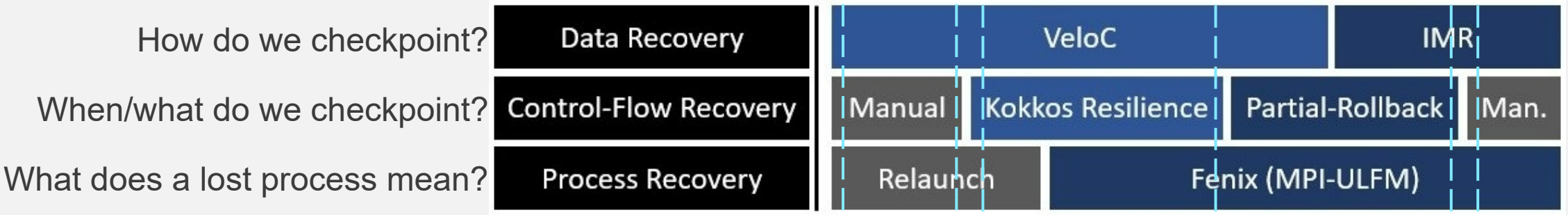
1. **Why** do anything?
   - Resilience is important and annoying
2. **What** did we do?
   - Our goals
3. **How** did we do it?
   - Tools we used
   - Our protocol
4. How well did we do?
   - Performance
   - Programmability
5. Conclusions

# Why: Software resilience is important to HPC

- Blue Waters study [1]
  - Average Mean time to Failure (MTTF) of 4.2 hours
  - ~9% machine productivity loss from system-software failures
- Exascale only exacerbates reliability concerns [2]
- Hardware vs Software resilience [3]
  - Reliability = Power Consumption = Money and heat
    - Chip designers looking to offload reliability to software
  - Software resilience is inherently co-design

# Why: Current approaches are insufficient

- Developers are expensive - their time is valuable and limited
  - High performance reliability is often highly code intrusive
  - Online failure handling explodes state space
- Consider resilience in layers
  - Representative technologies within each layer




- We have implemented and tested each vertical bar

# Why: Current approaches are insufficient


- Two categories of current approaches

| | Single Layer | Multi Layer |
|---|---|---|
| Strengths | ▪ Diverse/flexible tools<br>▪ Easier on pre-existing apps | ▪ High performance<br>▪ Strongly integrated layers<br>▪ Often automated into runtimes |
| Weaknesses | ▪ Performance requires multiple<br>▪ Not designed for integration | ▪ Intrusive code rewrites<br>▪ Pre-integrated, fixed/custom |
| Examples | VeloC, ULFM, MPI-Reinit, CRUM, … | FMI, CPPC, CRAFT, ACR, … |

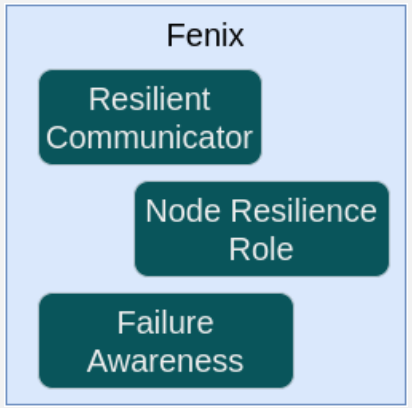
- Lack of simple and customizable multi-layer resilience

# What: Simplify application-level integration


- Extend tools to enable seamless integration
  - Technology agnostic approach
  - Simplify information sharing
- Meet applications where they're at with resilience
  - Minimize rewrites
- Maximize runtime configurability
  - Future-proofing

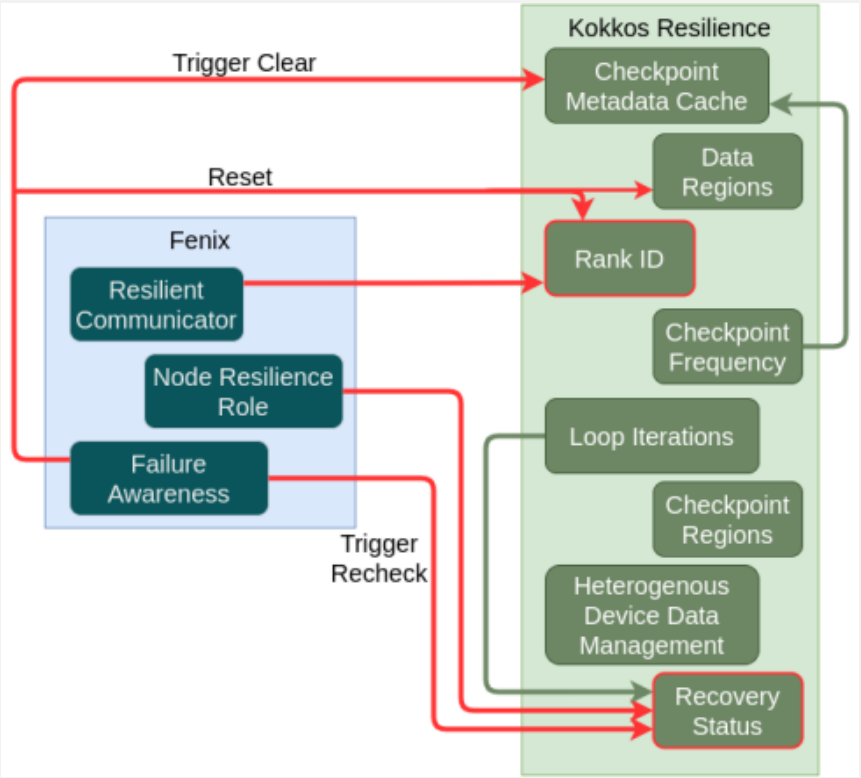
# How: Our tools

# How: Our tools

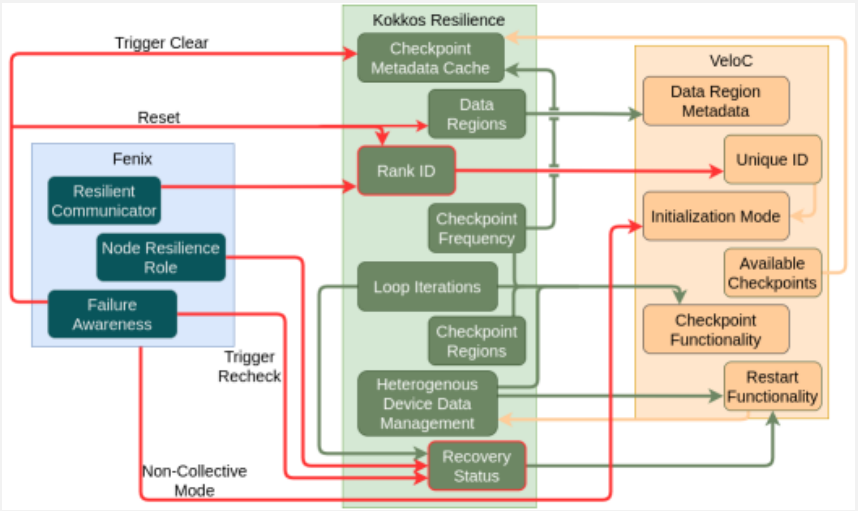

- Relaunch
  - Typical usage, tear down processes relaunch in job submission script
- ULFM - https://fault-tolerance.org/
  - User Level Fault Mitigation
  - MPI Specification Change Proposal, available in OpenMPI 5
  - Low level, for minimal specification changes
  - Enables error reporting for rank failures
  - Enables removing failed ranks from communicators
- Fenix - https://github.com/epizon-project/Fenix
  - Simplifies ULFM-based recovery
  - Designed to integrate with existing resilience
  - Spare ranks replace failed ranks (configurable)
  - Long-jump to initialization (configurable)
  - Enables Partial-Rollback/IMR

# How: Our tools

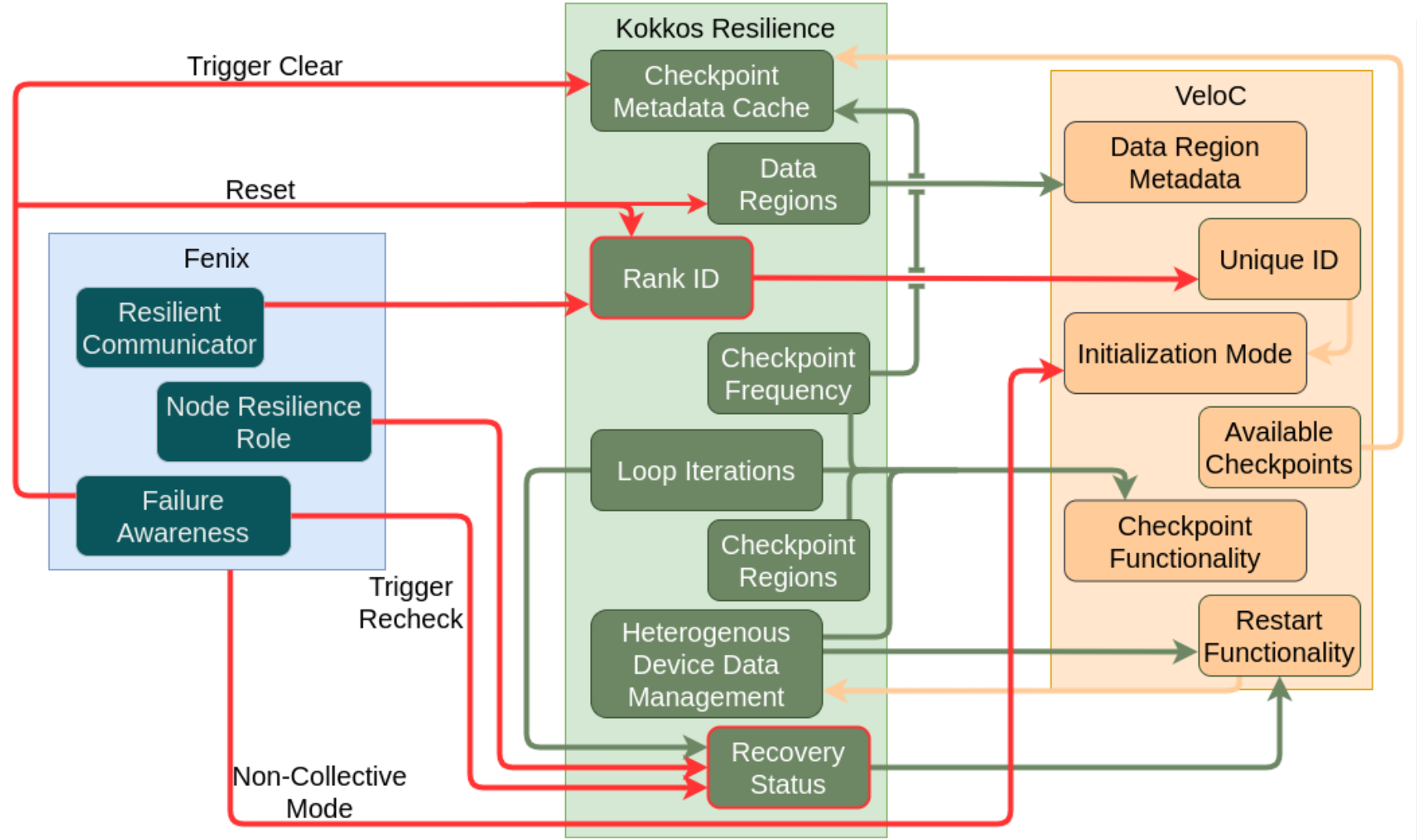

- Kokkos - https://github.com/kokkos/kokkos
  - Modern C++ parallelism library
  - Performance portable, heterogeneous
  - Typically for-loop based parallelism
  - Parallel lambdas/functors
  - Custom data structures (Kokkos::View)
- Kokkos Resilience - https://github.com/kokkos/kokkos-resilience
  - Gathers control-flow and data usage knowledge from Kokkos
  - Resilient lambdas
  - Internal VeloC integration, automate checkpoint/recovery
- Partial rollback
  - Strategy available with online process recovery
  - Requires integration
  - Only recover on failed ranks
  - Limited applications support

# How: Our tools

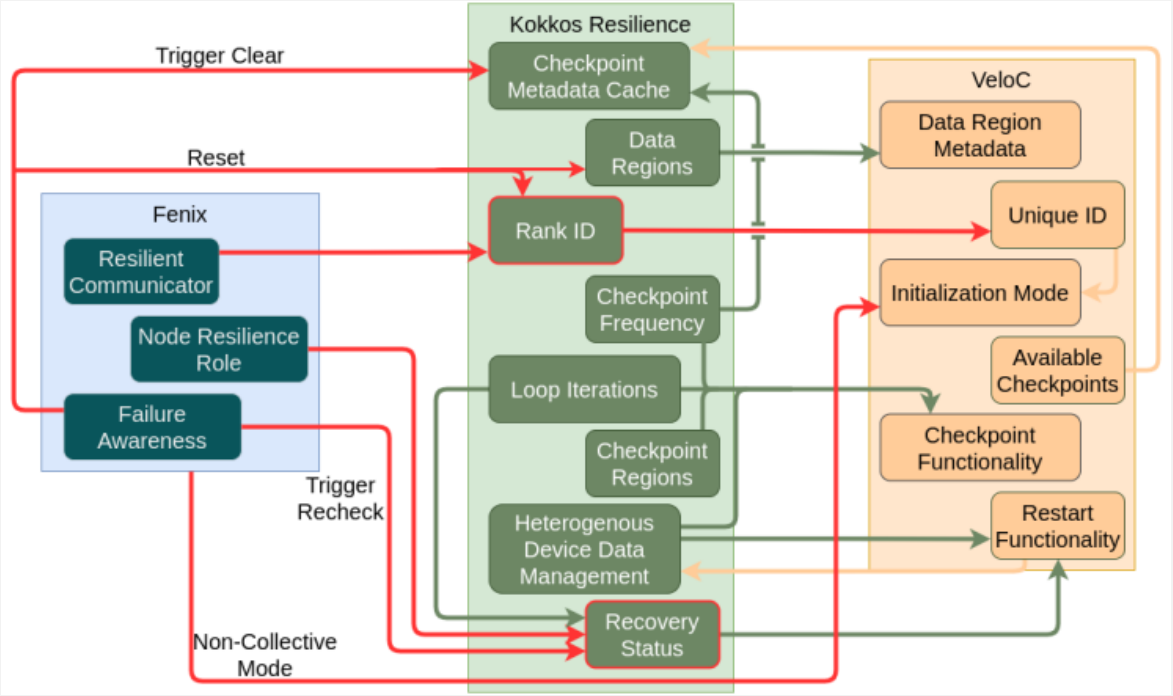

- VeloC - https://github.com/ecp-veloc/veloc
  - VEry Low Overhead Checkpointing
  - Contemporary, actively developed
  - Asynchronous
  - Multi-level checkpointing
- IMR
  - In Memory Redundancy
  - Fenix-provided
  - Memory based checkpointing
  - Buddy ranks, RAID-like

# How: Integration strategy



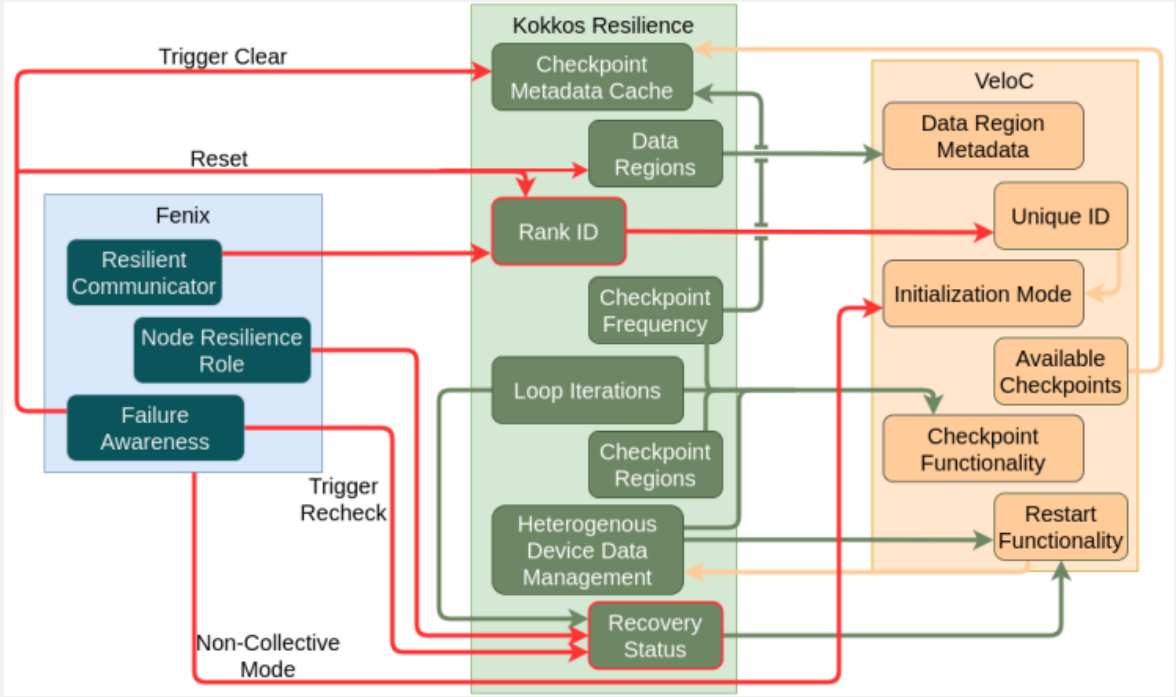
# How: Kokkos Resilience Modifications

- Kokkos Resilience
  - Remove all dependence on a static communicator
  - Allow application to request state reset
  - New input parameters to control management pattern of VeloC
- Changes remain agnostic to particular tool choices
- Just assumes that it has a currently -functional communicator



# How: VeloC Accommodations

- VeloC
  - No internal changes needed, must change usage from typical
  - Initialize without a communicator
  - User manually finds consensus
  - Performance tuning
    - VeloC serverlet assumes new process connection after failure
    - Online recovery partially serializes pre/post failure operations



# How: Integration strategy

```
1.    Non-communicative init
2.  MPI_Init()
3.    Communicative init
4.  for(i=0;…){
5.    Work using MPI_COMM_WORLD
6.  }
7.  MPI_Finalize()
```

- (3.) Fenix_init()
  - longjmp point
  - role reports local failure/recovery status
- (9.) ctx.reset(res_comm)
  - User-initiated state reset
- (12.) Checkpoint lambda automatically checkpoint/restarted
- (13.) Simply ctrl-f and replace MPI_COMM_WORLD with res_comm

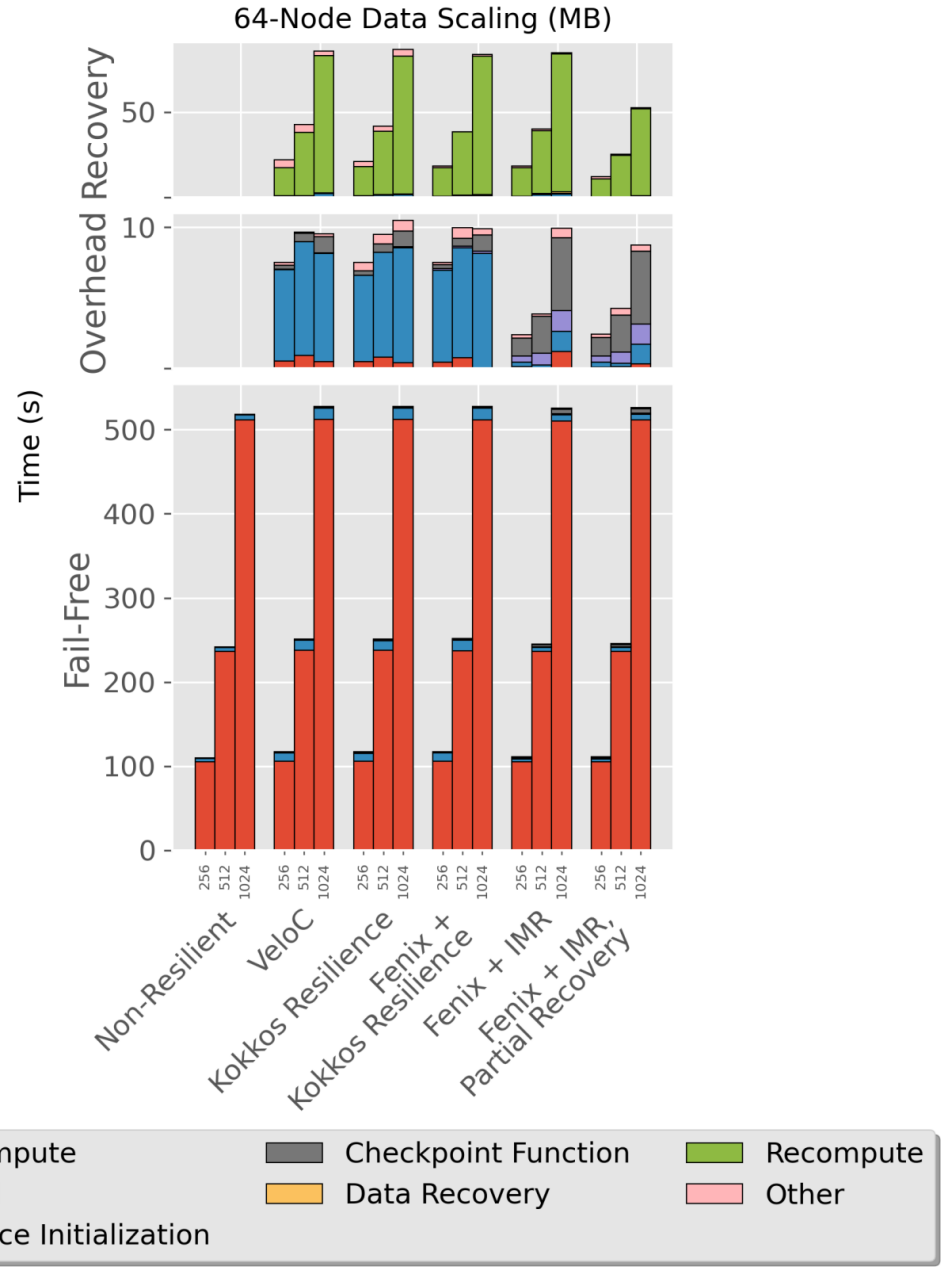
```
1.    Non-communicative init
2.  MPI_Init()
3.  Fenix_Init(MPI_COMM_WORLD, &res_comm, &role)
4.  if(role == initial)
5.    Communicative init
6.  if(role != survivor)
7.    ctx = KokkosResilience::make_context(res_comm)
8.  else
9.    ctx.reset(res_comm)
10. i = ctx.latest_version
11. for(i;…){
12.   KokkosResilience::checkpoint(*ctx, i, (){
13.     Work using res_comm
14.   });
15. }
16. MPI_Finalize()
```

# Testing: Benchmark and MiniApp performance

- VeloC's Heat Distribution (Heatdis) benchmark
  - 2D Heat distribution stencil
  - Tested with many combinations of resilience layers
  - Used to compare basic performance
- Sandia's MiniMD mini application
  - Molecular dynamics simulation
  - Tested with our resilient protocol
  - More realistic performance verification
  - Programmability test
- Platform:
  - 100-node Cray XC40 system
  - 2 socket Intel Haswell nodes, 32 cores per node
  - One rank per node
  - Lustre distributed filesystem

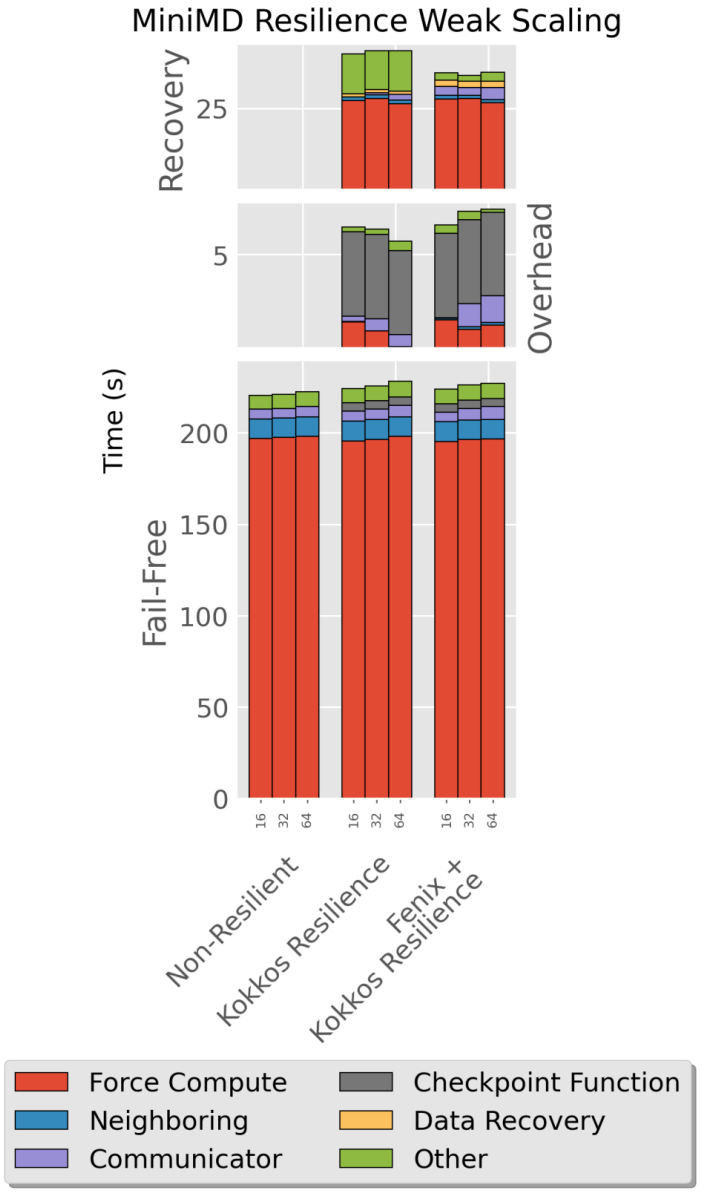
# Results: Simplified integration outperforms single-layer resilience - Heatdis

- Adding layers has little-to-no failure-free overhead
- Multi-layer recovery improves performance even for naïve implementations
- Potential for large performance improvements
- Best data layer tool depends on amount of data
  - Flexibility is key



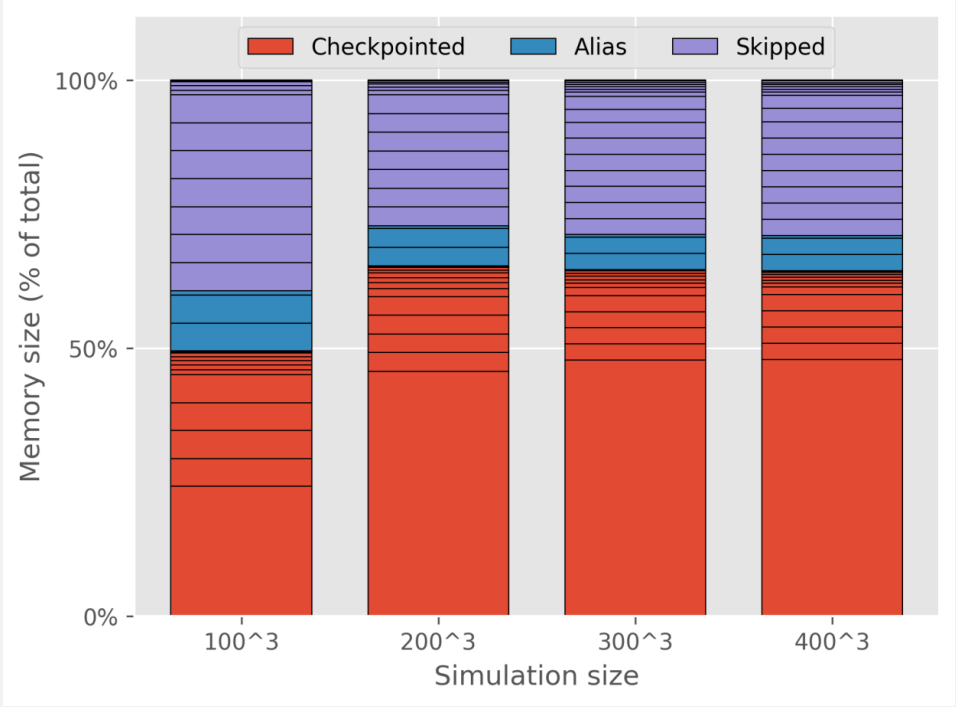
# Results: Overhead and recovery costs depend on communication/compute balance - MiniMD

- Applications regions have different behaviors
  - Force Compute: highly compute-bound
  - Neighboring: Mixed compute/communication
  - Communicator: highly communication-bound
- Microcosm of different application types
  - Again, flexibility is best



# Results: The integration is efficient

- 61 Kokkos::View objects
  - We don’t want to manually inspect
  - Some contain duplicate data (aliased)
    - User can specify alias names
  - Some are references to the same data (Skipped)
    - Automatically detected
  - Avoids 50-100% unneeded checkpoint overhead
- Are we over checkpointing?
  - Small # of sizable Views to be inspected, compared to initial 61




Each box represents a View object, with its height proportional to its size

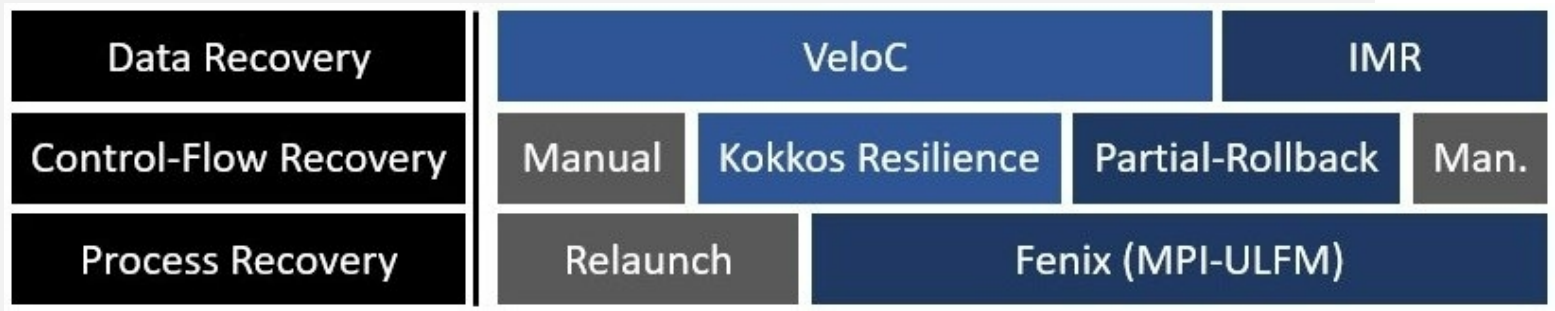
# Results: Programmability is high

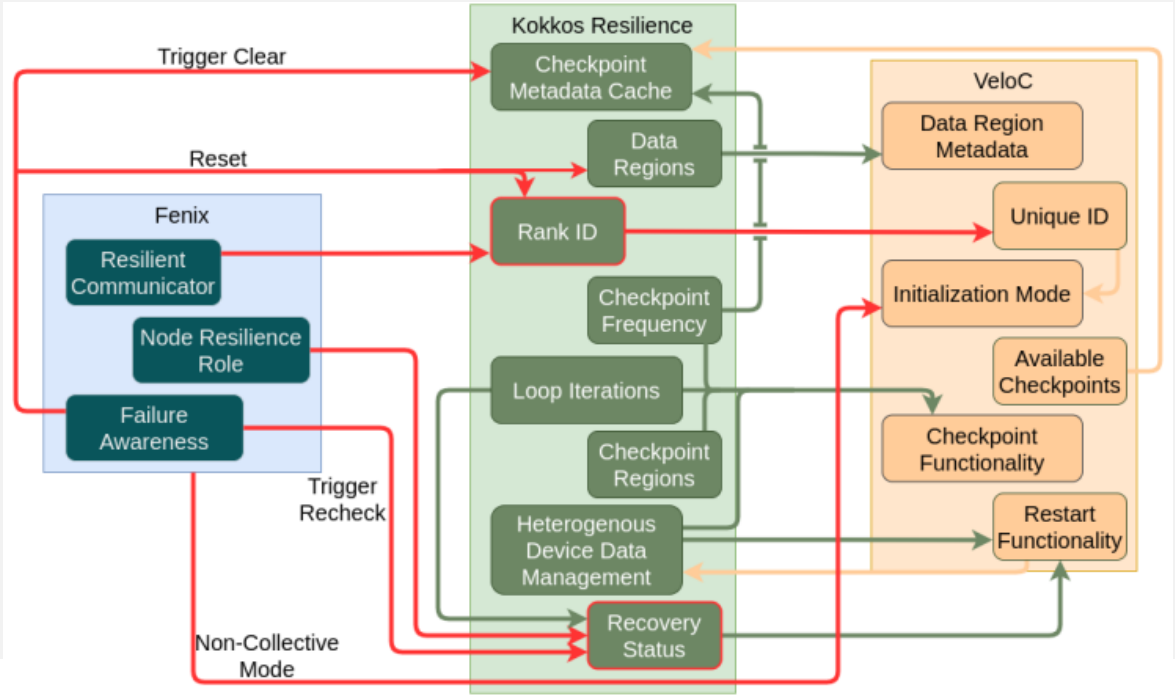
- MiniMD
  - 61 Kokkos::View data objects
  - 20+ source files
  - 148 MPI calls across 15 source files
- Programming cost
  - Kokkos Resilience changes only a small handful of files
  - Fenix changes similarly limited to two files, ~20 lines of code
- Unintegrated cost estimate
  - ULFM error handling roughly per MPI call
    - Or approach rebuilding Fenix
  - VeloC calls per Kokkos::View
    - Manually inspect what/when to checkpoint
  - Highly intrusive!
- This integration of Kokkos Resilience and Fenix was enabled by our work

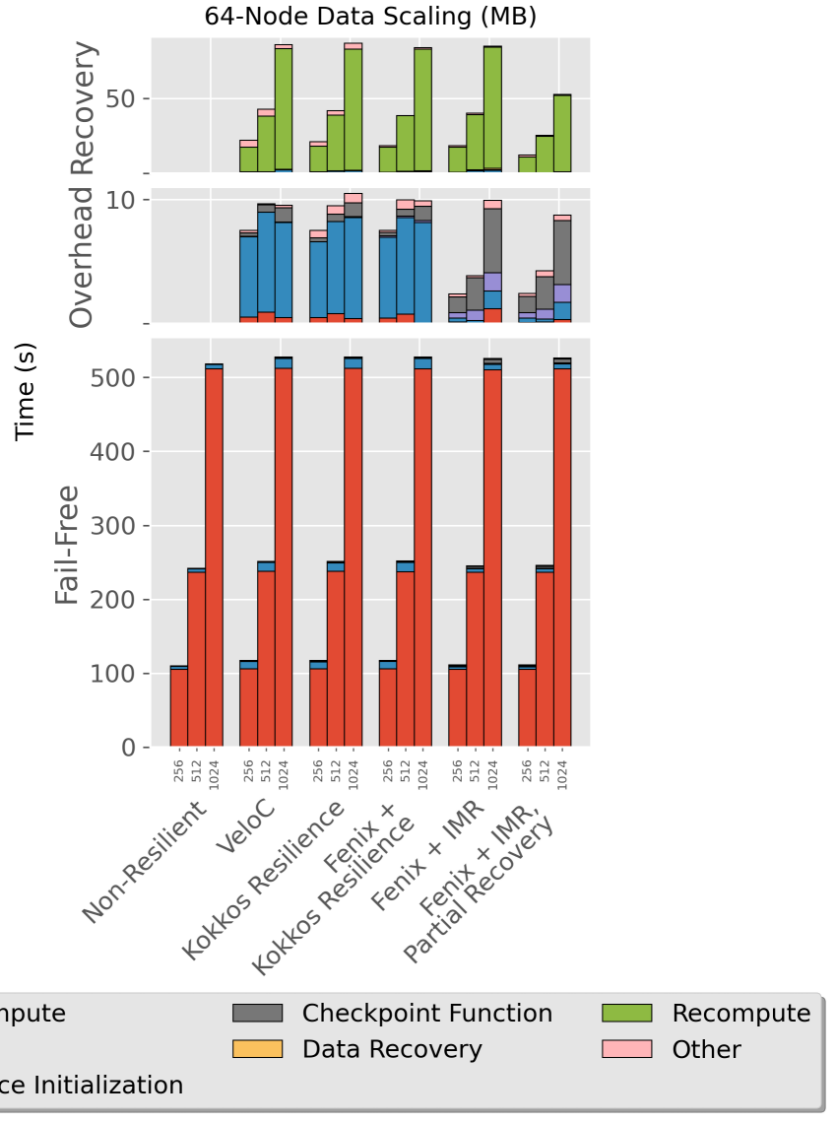
# Conclusions: App-level integration can be performant, simple, and flexible

- Outperforms single-layer resilience
- Introduces flexibility in resilience strategies
- Simpler to use than single-layer resilience
- Simpler to add to existing applications than internally integrated multi-layer systems
- Future work
  - Room for more complex recovery to be introduced with better integration
  - Checkpoint/recovery with a shrinking/growing communicator
    - Data/Control-flow to redistribute work
  - More localized error handling

# Q&A – Thanks!


Data Recovery
VeloC
IMR
Control-Flow Recovery
Manual
Kokkos Resilience
Partial-Rollback
Man.
Process Recovery
Relaunch
Fenix (MPI-ULFM)


Kokkos Resilience
Trigger Clear
Checkpoint Metadata Cache
Reset
Data Regions
Fenix
Rank ID
Resilient Communicator
Checkpoint Frequency
Node Resilience Role
Loop Iterations
Failure Awareness
Checkpoint Regions
Trigger Recheck
Heterogenous Device Data Management
Recovery Status
Non-Collective Mode
VeloC
Data Region Metadata
Unique ID
Initialization Mode
Available Checkpoints
Checkpoint Functionality
Restart Functionality


64-Node Data Scaling (MB)
Overhead Recovery
50
10
Time (s)
Fail-Free
500
400
300
200
100
0
256
512
1024
Non-Resilient
VeloC
Kokkos Resilience
Fenix + Kokkos Resilience
Fenix + IMR
Fenix + IMR, Partial Recovery
App compute
App MPI
Resilience Initialization
Checkpoint Function
Data Recovery
Recompute
Other

# References

[1] C. Di Martino, Z. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer, “Lessons learned from the analysis of system failures at petascale: The case of blue waters,” in 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2014, pp. 610–621

[2] Ramon Canal, Carles Hernandez, Rafa Tornero, Alessandro Cilardo, Giuseppe Massari, Federico Reghenzani, William Fornaciari, Marina Zapater, David Atienza, Ariel Oleksiak, Wojciech PiĄtek, and Jaume Abella. 2020. Predictive Reliability and Fault Management in Exascale Systems: State of the Art and Perspectives. ACM Comput. Surv. 53, 5, Article 95 (September 2021), 32 pages. https://doi.org/10.1145/3403956

[3] S. Heldens, P. Hijma, B. V. Werkhoven, J. Maassen, A. S. Z. Belloum, and R. V. Van Nieuwpoort, “The landscape of exascale research: A data-driven literature analysis,” ACM Comput. Surv., vol. 53, no. 2, mar 2020. [Online]. Available: https://doi.org/10.1145/3372390