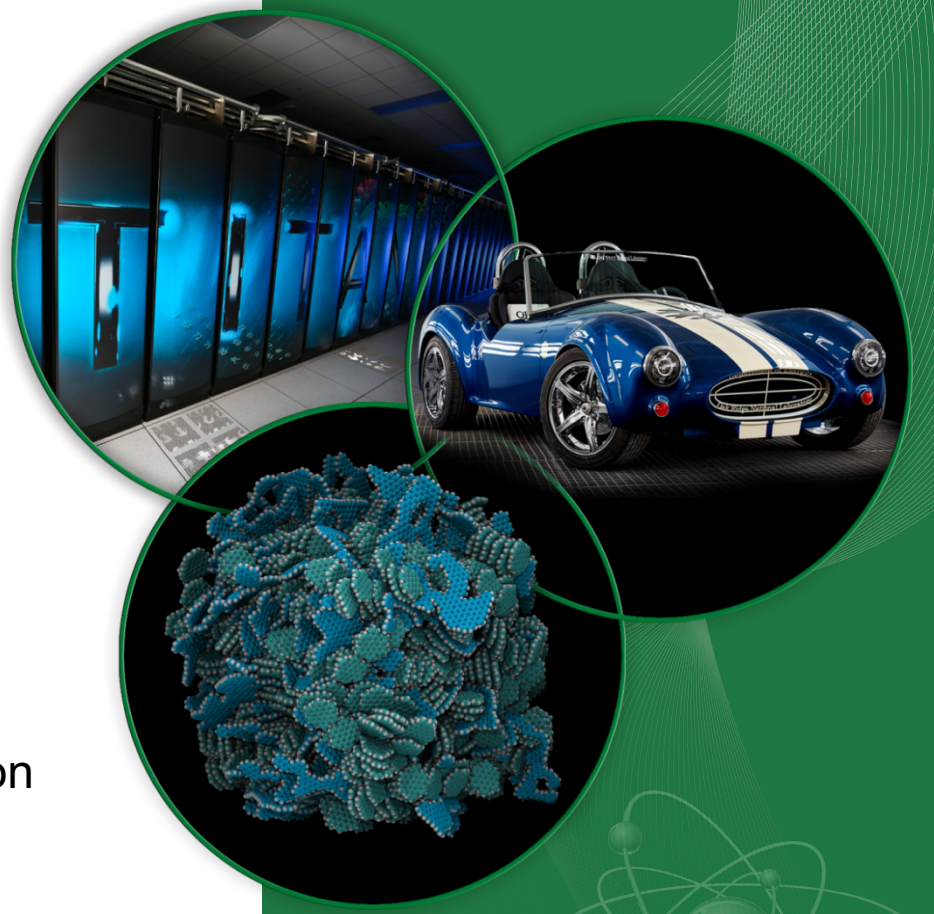
Resilience Design Patterns: A Structured Approach to Resilience at Extreme Scale



Saurabh Hukerikar
Christian Engelmann

Computer Science Research Group
Computer Science & Mathematics Division
Oak Ridge National Laboratory, TN, USA

18th SIAM Conference on Parallel Processing for Scientific Computing (PP), Tokyo, Japan, March 7-10, 2018

Resilience, Why?

- ***Resiliency*** in high performance computing (HPC) applications: the ability to gracefully handle errors and recover from failures.
- Errors and failures are common place in HPC systems today.
  - Large-scale systems with a number of complex & diverse software and hardware components,
  - Technology scaling trends in hardware components,
  - Complex compute, memory, interconnect and storage architectures,
  - Cost (design, area, power, engineering) of achieving error-free large scale systems is too high.
- The situation is only expected to get worse, as we move towards the goal of achieving more computational power, i.e., *Exascale systems*.

Motivation

- **The Paradox of Choice:** Several resilience solutions [hardware, system software, algorithm-based, programming model-based, etc.]

× Incomplete understanding of protection coverage against high-probability & high-impact vs. less likely & less harmful faults

× No evaluation methods & metrics that consider
  - Fault impact scope, handling coverage and handling efficiency
  - Performance, resilience and power trade-offs

× No mechanisms and interfaces for coordination for avoidance of costly overprotection

× No portability across architectures and software environments

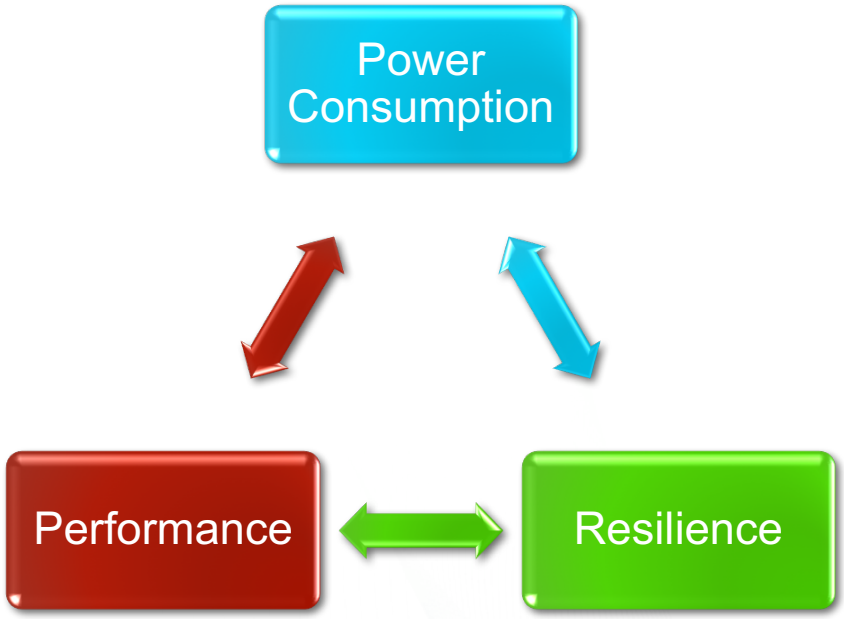
Resilience Design Patterns
A Structured Approach to Resilience at Extreme Scale

- Design Patterns
  - Structural elements that capture an idea in architectural design
  - Patterns describe the essence of a solution to a problem that occurs often in practice
  - Every pattern is an unfinished design
- Each pattern describes a problem, which occurs repeatedly, and then describes the core of the solution to that problem, in such a way that this **solution may be used many times over, without ever doing it the same way twice.**

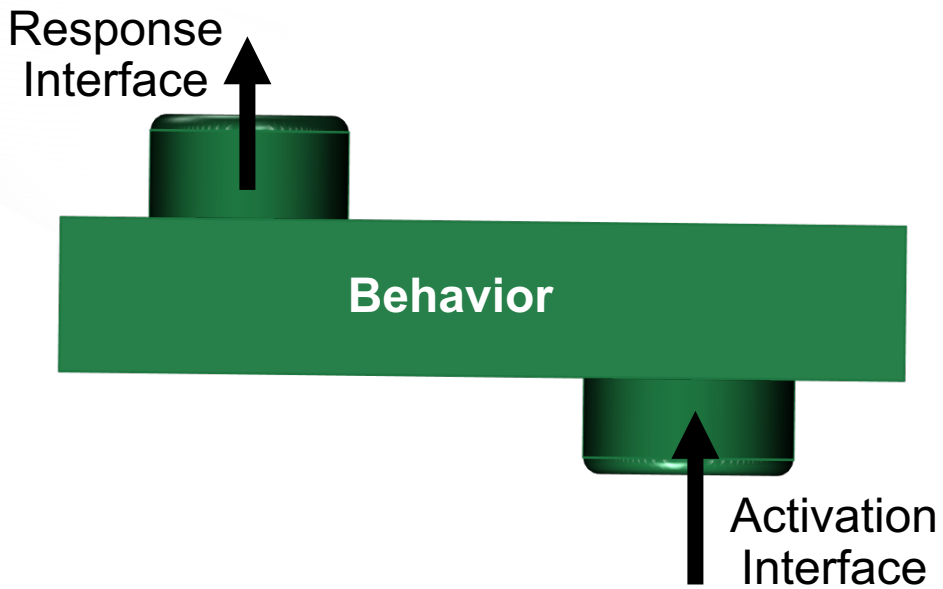
The Design Pattern Solution for Resilience

Resilience design patterns as tools to:

- Identify & evaluate repeatedly occurring resilience problems
- Discover the **essential** components of solutions
- Use the patterns as **building blocks:**
  - Enables coordination of solutions using hardware and software components
  - Systematic evaluation of alternative solutions
  - Navigate the trade-off space between Power, Performance, Resilience



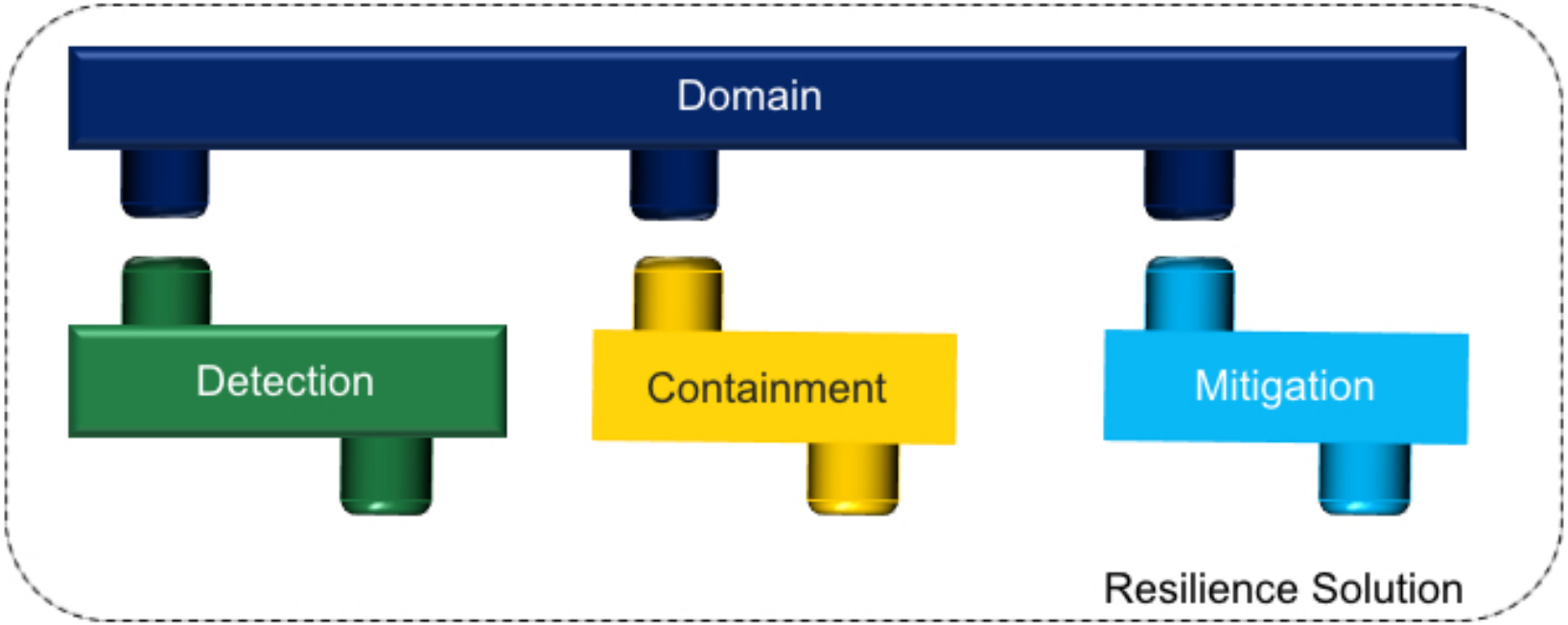
Anatomy of a Resilience Design Pattern



Abstract definition of a resilience design pattern

- Basic template of a resilience design pattern is defined in an event-driven paradigm
- Instantiation of pattern behaviors may cover combinations of detection, containment and mitigation capabilities.
- Enables writing patterns in consistent format to allow readers to quickly understand context and solution.

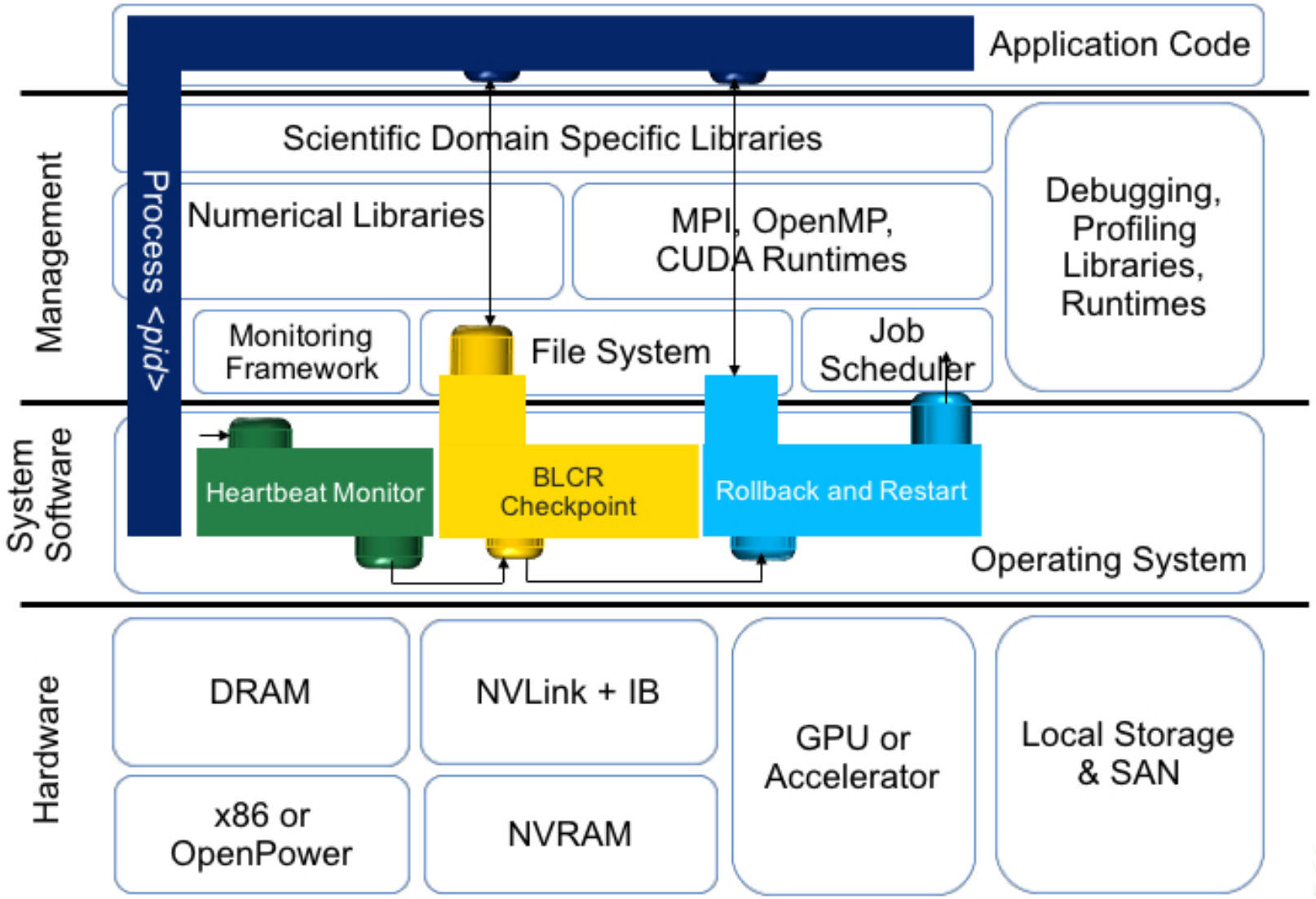
Anatomy of a Resilience Design Pattern



Abstract definition of a resilience design pattern

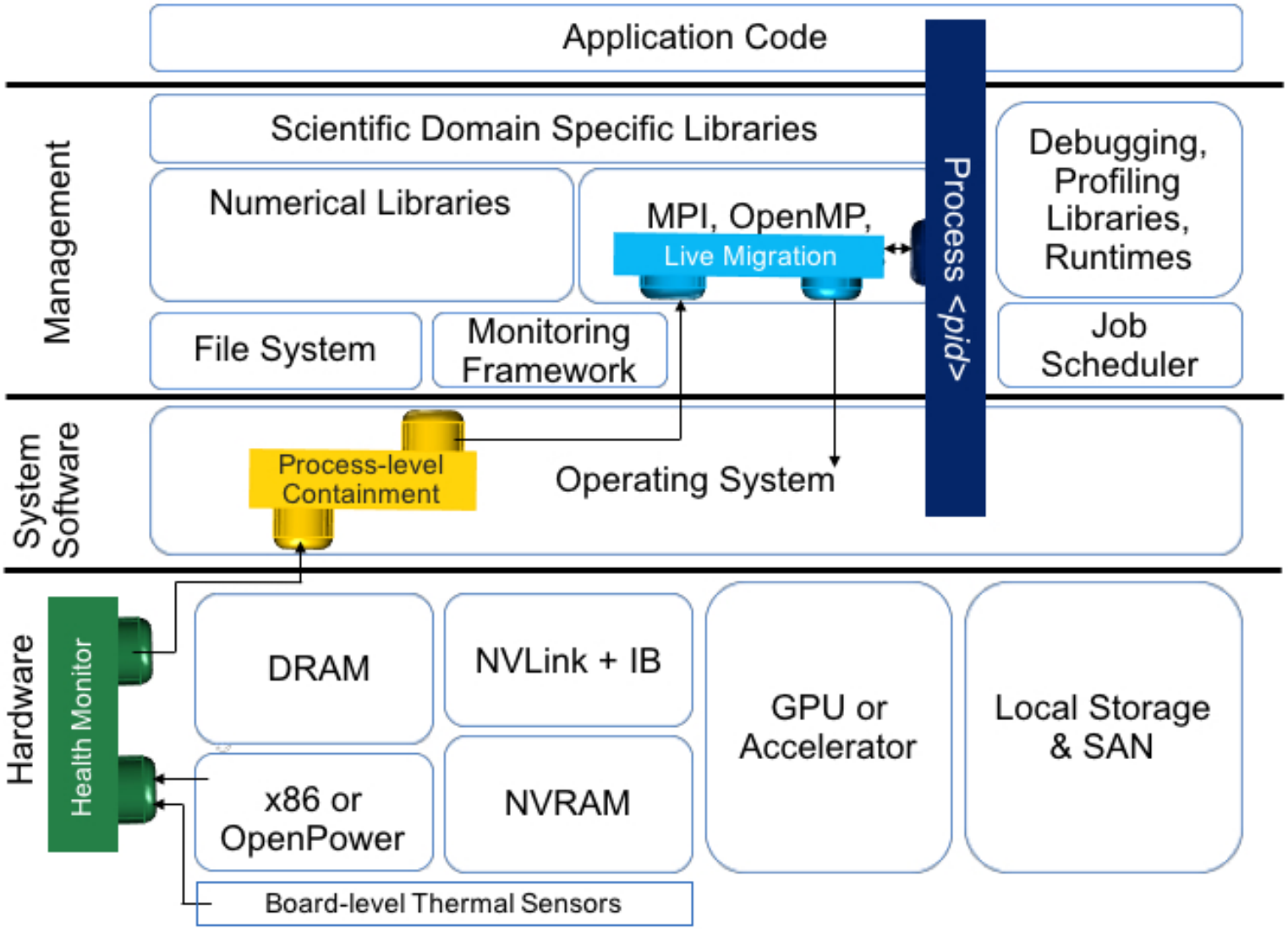
- Basic template of a resilience design pattern is defined in an event-driven paradigm
- Instantiation of pattern behaviors may cover combinations of detection, containment and mitigation capabilities.
- Enables writing patterns in consistent format to allow readers to quickly understand context and solution.

Case Study: Checkpoint and Rollback



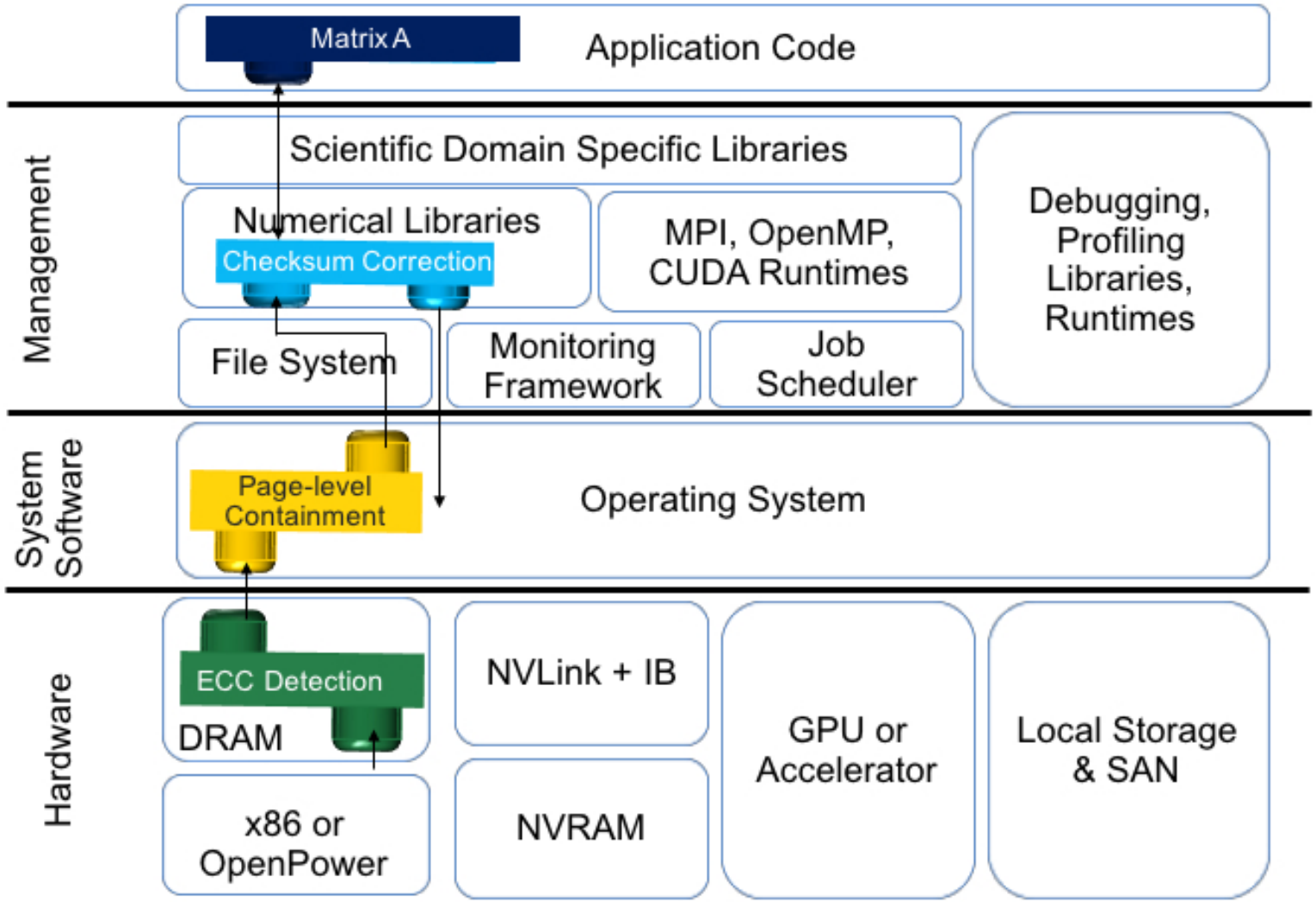
8

Case Study: Proactive Process Migration



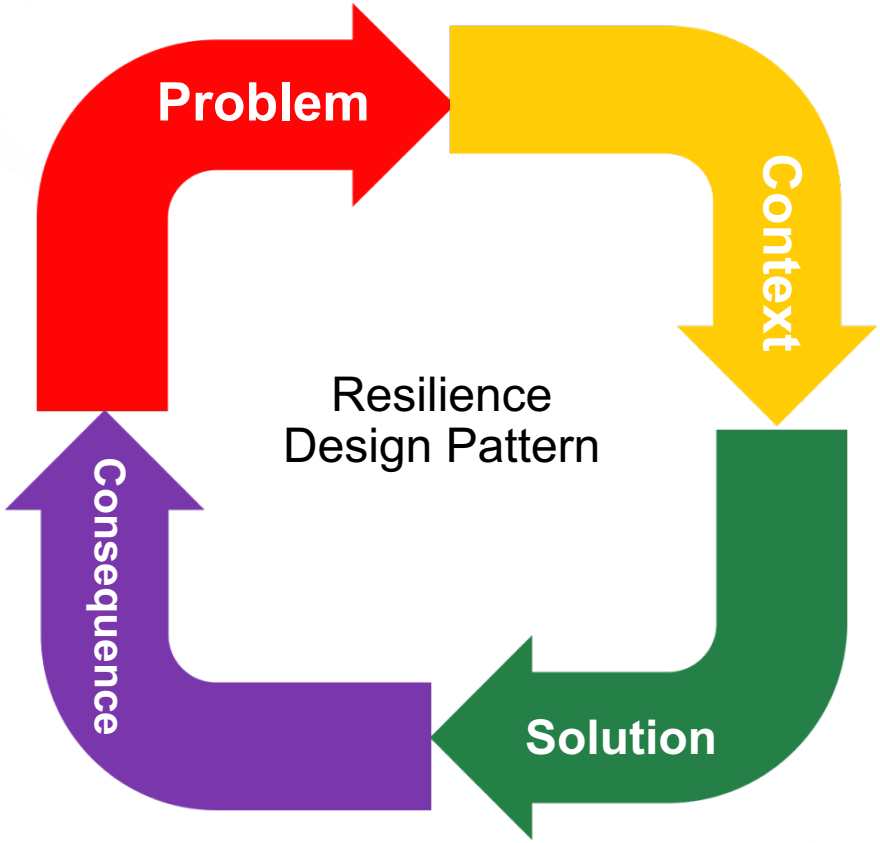
9

Case Study: Cross-Layer Hardware/Software Hybrid Solution



Describing Resilience Design Patterns

Essential elements of Resilience Design Patterns

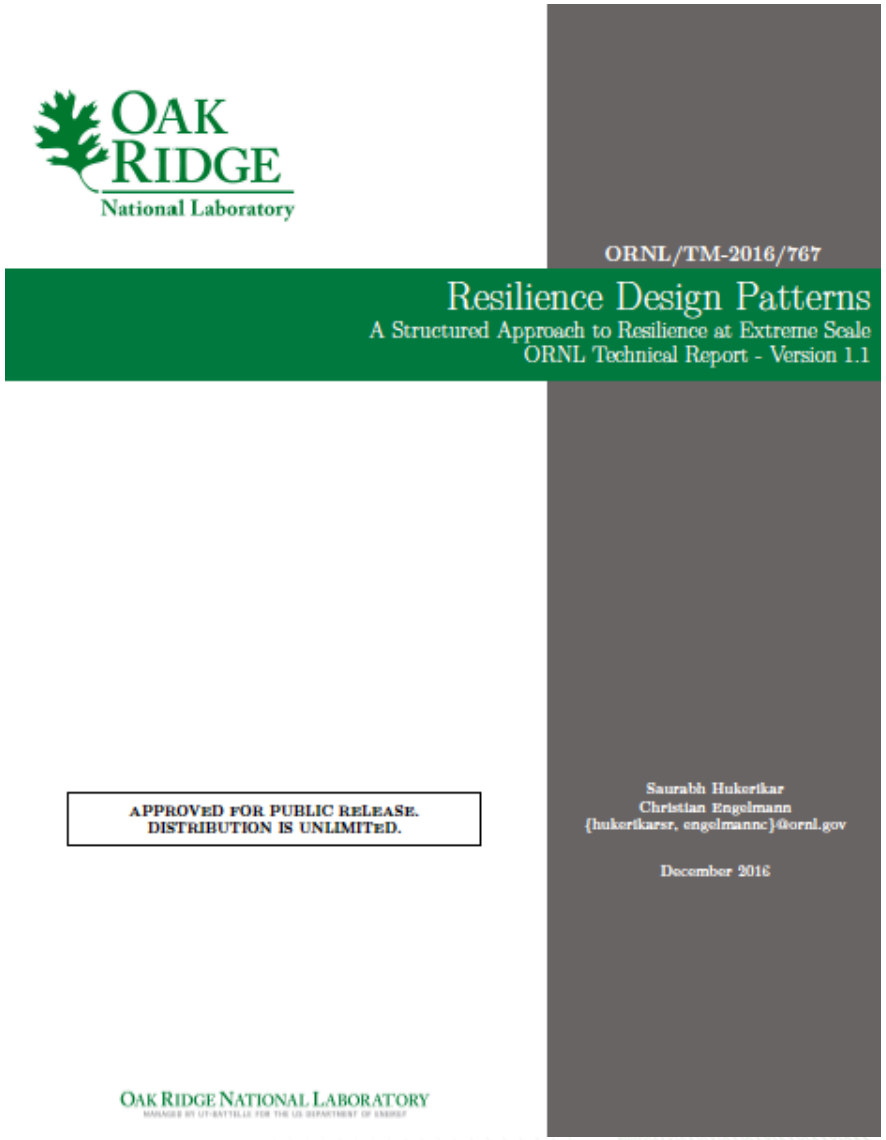


- Each pattern describes a problem, the core of the solution, and the implications for design, **but** is highly adaptable to any design context
- Patterns must be meaningful for hardware and software components

Resilience Design Patterns Specification

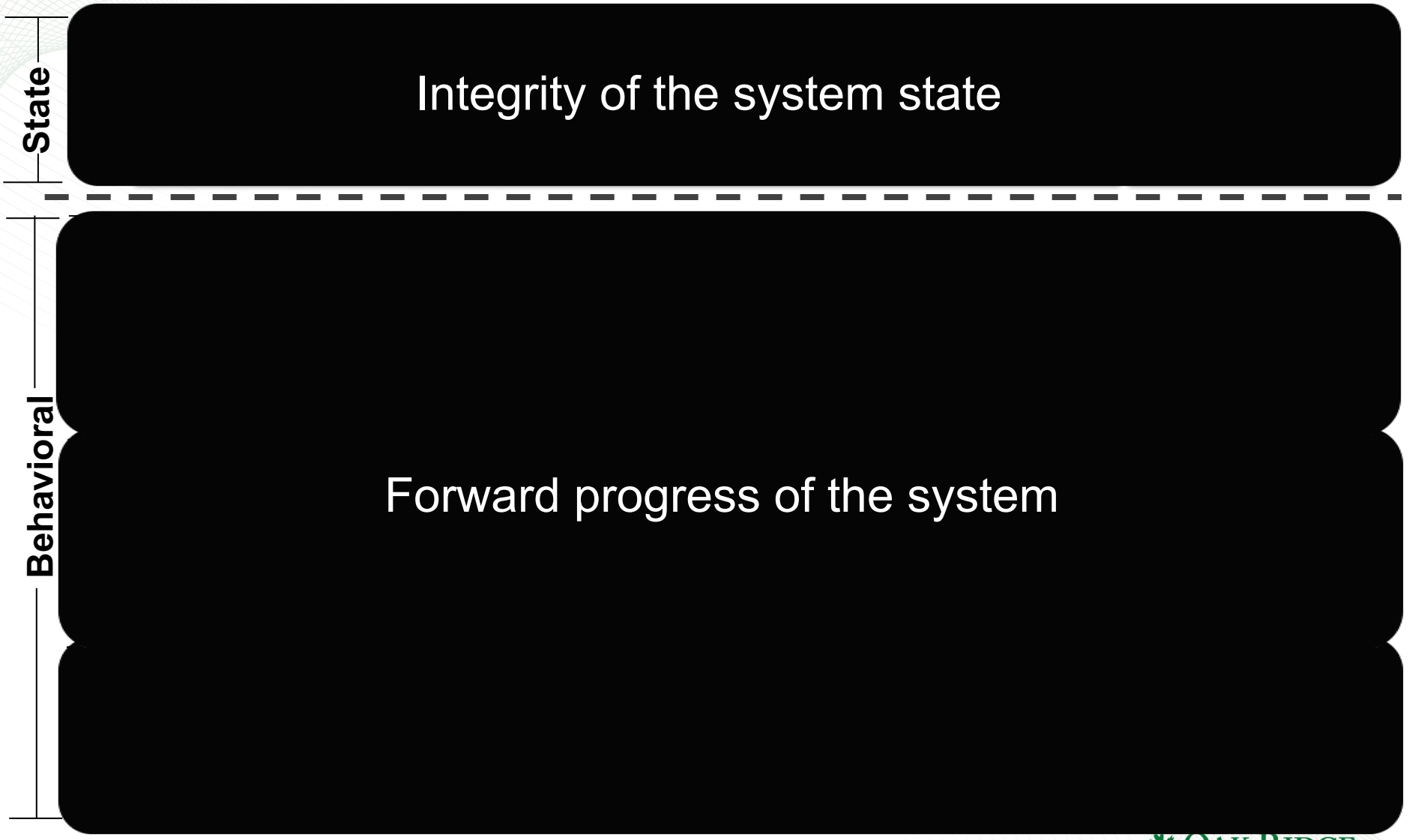
Specification Document v1.1:

- Complete catalog of resilience design patterns
- Detailed descriptions of the components that make up each solution
- Solutions may be adapted to any system architecture, software environment



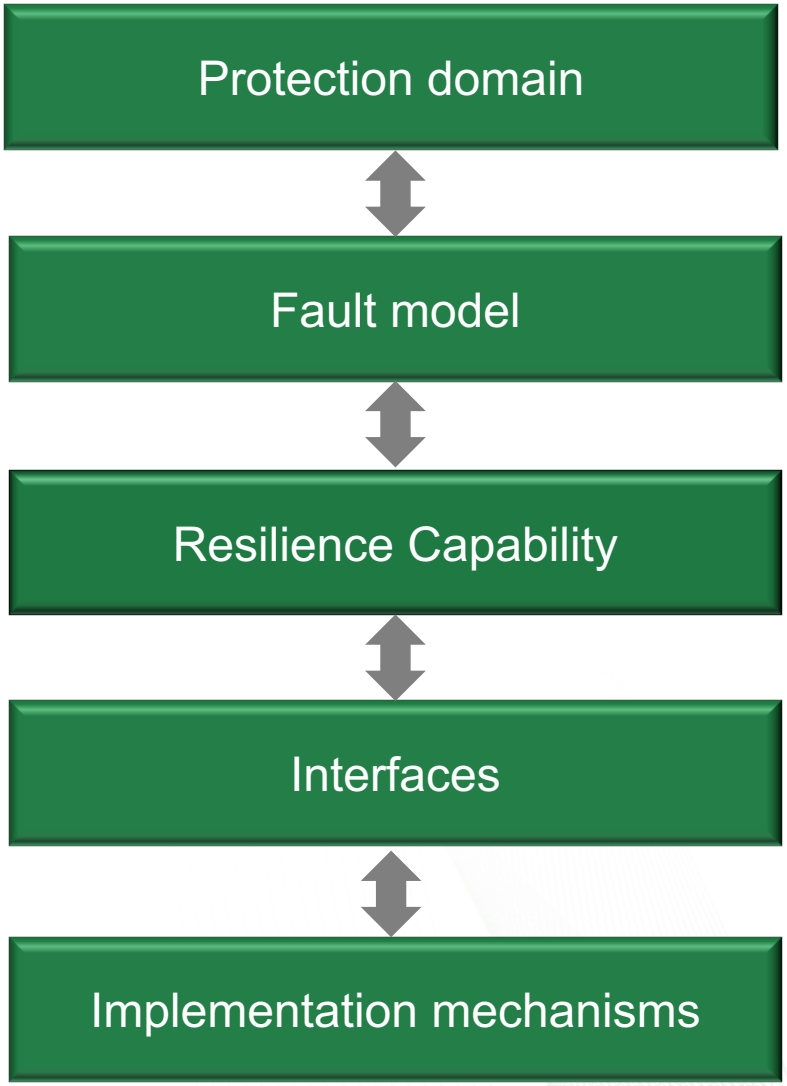
OAK RIDGE National Laboratory

Resilience Design Patterns



Design Spaces Framework

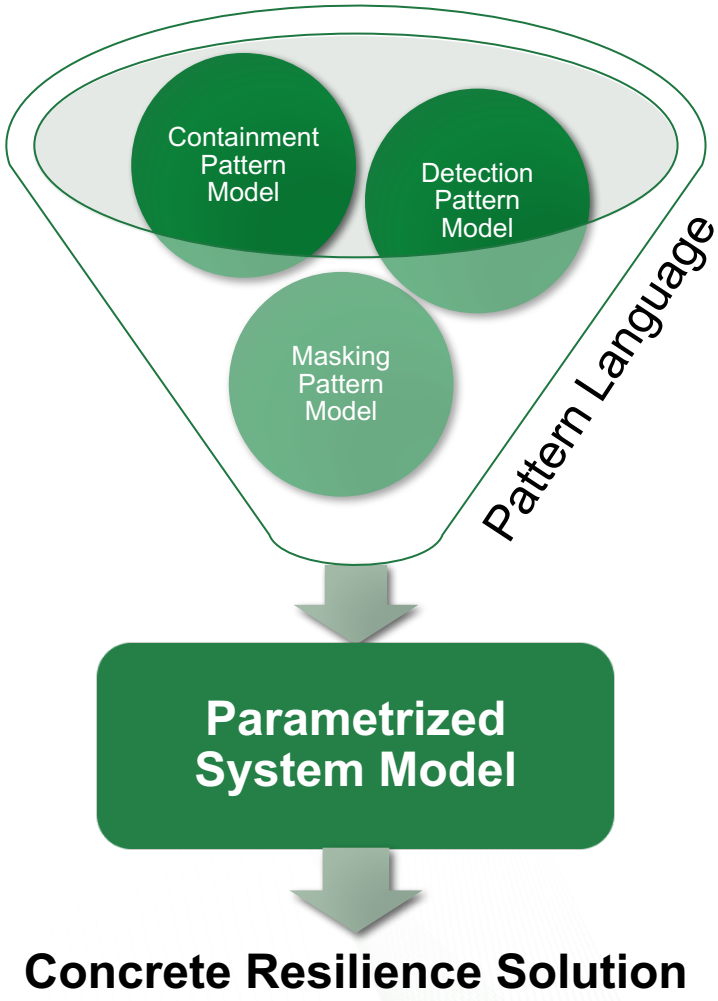
- Resilience design can be viewed as a **series of refinements**
- We define the process in terms of five ***design spaces***
- Navigating each design space progressively adds more detail to the overall design of the resilience solution



Systematic Modeling & Design of Resilience Solutions

Design space exploration by abstracting system and applications models and evaluating high-level resilience design pattern-based solutions:

- Discover suitability of given combination of patterns to a specific problem
- Predict behavior on different hardware architectures and software environments: Quality of service (QoS) delivered to HPC applications
- Find performance bottlenecks, scaling problems
- Optimize the trade-offs between resilience and performance



Pattern Models

- Preliminary mathematical models for the various resilience design patterns in our catalog
- Pure performance analysis of a HPC system/application tends to be optimistic: ignores the failure/repair behavior in the system.
- On the other hand, pure reliability analysis tends to be too conservative since performance considerations are not taken into account.
- Goal is to develop methods to analyze the costs and benefits in terms of the interplay between performance and reliability of a solution
  - Models enable quantitative evaluation of the design alternatives
  - Models may be used by simulation framework to understand performance characteristics of resilience solution on future extreme-scale systems

Modeling the Fault Diagnosis Pattern

- Pattern requires system structure consisting of monitoring and monitored system.
  - Cause-Effect or Effect-Cause analysis of monitored system parameters to detect a defect or anomaly in system
- Performance implications of applying fault diagnosis pattern:

$$T_{system} = T_0 + \sum_{k=1}^{n} t_{inference}/\eta$$

where T_0 is time without diagnosis; n is number of observed parameters; η is frequency of interaction

- No tangible improvement in reliability of the system using the diagnosis pattern since it only recognizes the presence of a fault, its root cause and location

Modeling the Reconfiguration Pattern

- Pattern supports the modification of the interconnection between modules (components or computational tasks) in the system
  - The system assumes one of several valid configurations in response to a fault, error or failure
- Performance characteristics of reconfig. pattern:

$$T_{system} = T_{FF} + (1 - T_{FF}).\frac{n-1}{n} + T_R$$

  - Based on assumption that the reconfiguration entails isolation of the module that experiences an event and excludes the faulty module from the new configuration.
  - Where T_R is time lost to reconfigure modules, *(n – 1)/n* is loss in capacity on account of reconfiguration
- Reliability of the system using the reconfiguration design pattern:

$$R(n,t) = 1 - \prod_{i=1}^{n}(1 - R_i(t))$$

Modeling the Checkpoint Roll-back Pattern

- Most well-studied fault tolerance solution in HPC
- Performance characteristics of C/R pattern:

$$T_{system} = o + \delta/r$$

where *r* is the rate of checkpointing

$$T_{system} = (T_{FF} + \gamma)/\eta$$

where $T_{FF} = o + \delta/r$.

- Reliability of the system using the C/R design pattern:

$$R(t) = 1 - e^{-(T_{FF}+\gamma)/\eta}$$

Assumption: Operational lifetime of the system (or the execution time of an application) can be partitioned into distinct phases, which include the regular execution phase (o), the interval for creating checkpoints (γ), and the interval for recovery upon occurrence of an event (δ) to account for the operational state lost on account of the event.

Modeling the Roll-forward Pattern

- Pattern enables the system to recover and resume operation from the point of occurrence of an error or a failure
  - Uses checkpointed state and/or event logging to infer state information
- Performance characteristics of a system using the roll forward pattern:

$$T_{system} = o + \delta/r$$

δ is the interval required for checkpointing the state pattern
logging interval $\delta = \mathrm{M} \,.\, \mathrm{t}_{logging}$

- Reliability of the system using the n-version design pattern:

$$R(t) = 1 - e^{-(T_{PF} + M.t_{logging})/\eta}$$ … message logging-based implementation

$$= 1 - e^{-(T_{PF} + \gamma)/\eta}$$ … checkpointing-based implementation

Modeling the Redundancy Pattern

- N-modular redundancy: identical replicas of scope defined by state pattern; majority voting between replicas
- Performance characteristics of a system using the redundancy pattern:

$$T_{system} = T_{SER}.((1 - \mathcal{A}) + \beta.\mathcal{A})) + T_{MV}$$

where β = 1 when pattern configuration applies redundancy spatially, β = d for temporal redundancy

- Reliability of the system using the redundancy pattern:

$$R(t) = 1 - \prod_{i=1}^{d} t/\lambda = 1 - (t/\lambda)^d$$

where *d* refers to the degree of redundancy, i.e., the number of copies of the pattern behavior replicated.

Modeling the Design Diversity Pattern

- Design diversity pattern entails the creation of a N-version system design:
  - Distinct implementations of the same design specification are created
- Probability density function (PDF) of failure:

$$P(t) = ((1 - P(V)) \sum_{k=1}^{n+1} P(A_k) + P(V))$$

assuming *n* versions configured to operate independently with majority voting

- Reliability of the system using the n-version design pattern:

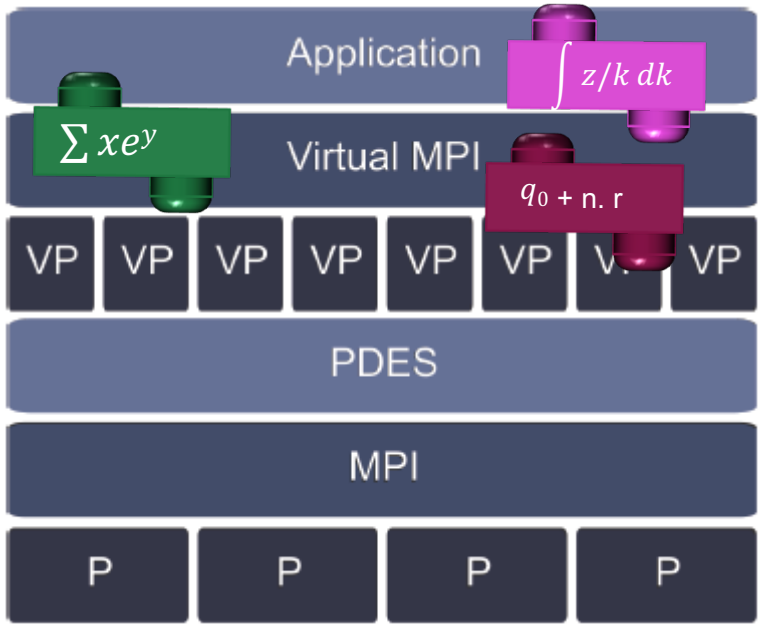
$$R(t) = 1 - ((1 - P(V)) \sum_{k=1}^{n+1} P(A_k) + P(V)).F(t)$$

P(V) represents the probability that the majority voting procedure cannot select the correct result from n version design

Planned work: Integration with xSim

xSim

- Performance simulator capable of running parallel applications by oversubscribing host system with millions of virtual processors
- Supports investigation at extreme scale, beyond the capabilities of existing simulation efforts
- Lightweight parallel discrete event simulation (PDES) to emulate the behavior of future architecture choices
- Resilience pattern models can be plugged into simulator to evaluate performance scalability of existing and new solutions



xSim with resilience pattern models

Conclusion & Future Work

- Resilience is a critical challenge for extreme-scale HPC
- Design pattern catalog based on well-known and well-understood solutions in highly structured format
  - Solutions are designed to be meaningful for HPC applications' algorithms, numerical libraries, system software, and architecture
  - Exploration of alternative solutions; Refinement and optimization of solutions for key design factors: **performance**, **resilience**, and **power consumption**
- New version of the Resilience Design Pattern Specification & Catalog will include the discussed models
- Future work will include power consumption models as well

Thank-you!