OAK RIDGE National Laboratory

Concepts for OpenMP Target Offload Resilience

Christian Engelmann (ORNL)
Geoffroy R. Vallée (Sylabs, Inc)
Swaroop Pophale (ORNL)

Contact: engelmannc@ornl.gov

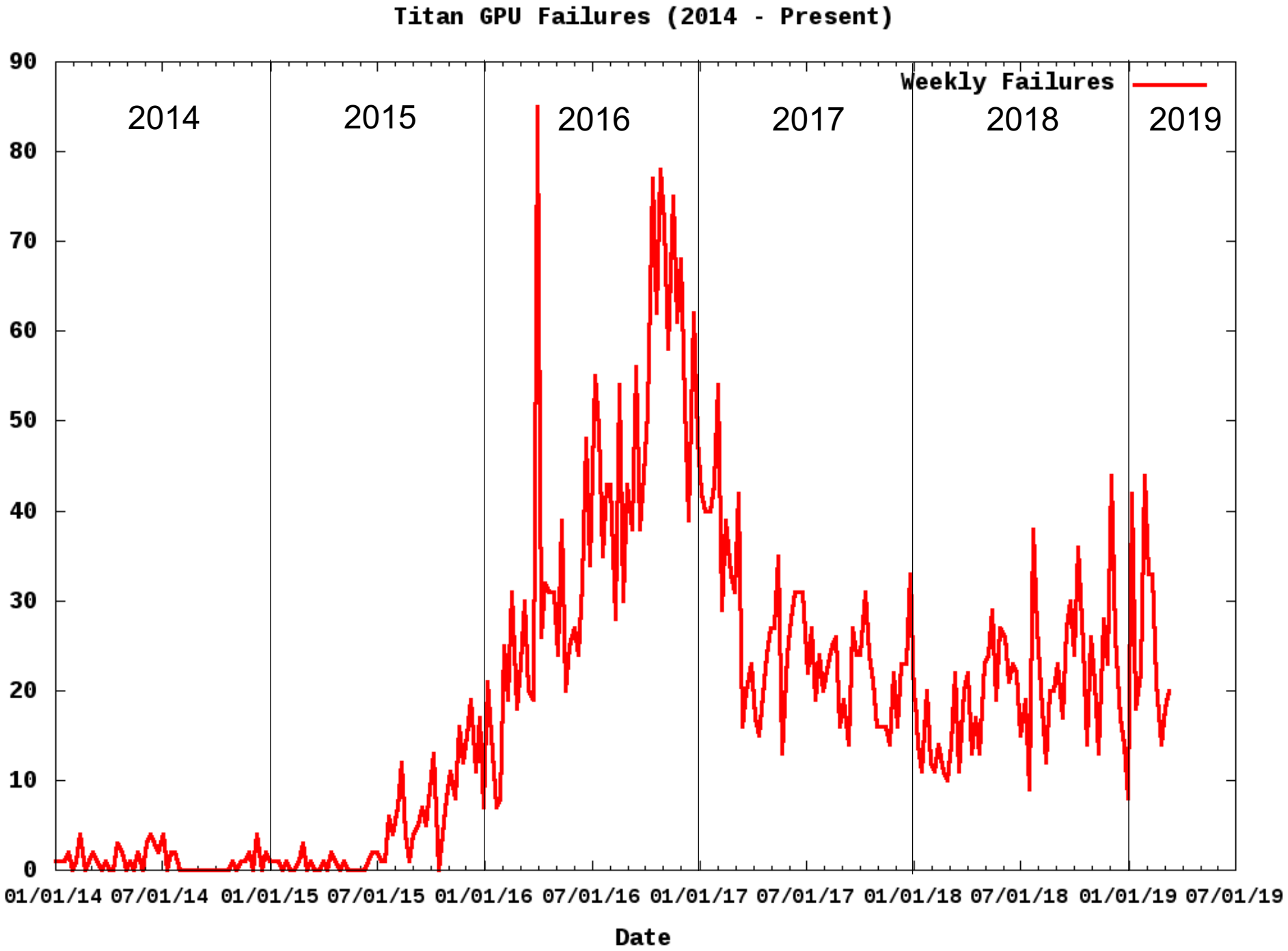
ORNL is managed by UT-Battelle, LLC for the US Department of Energy

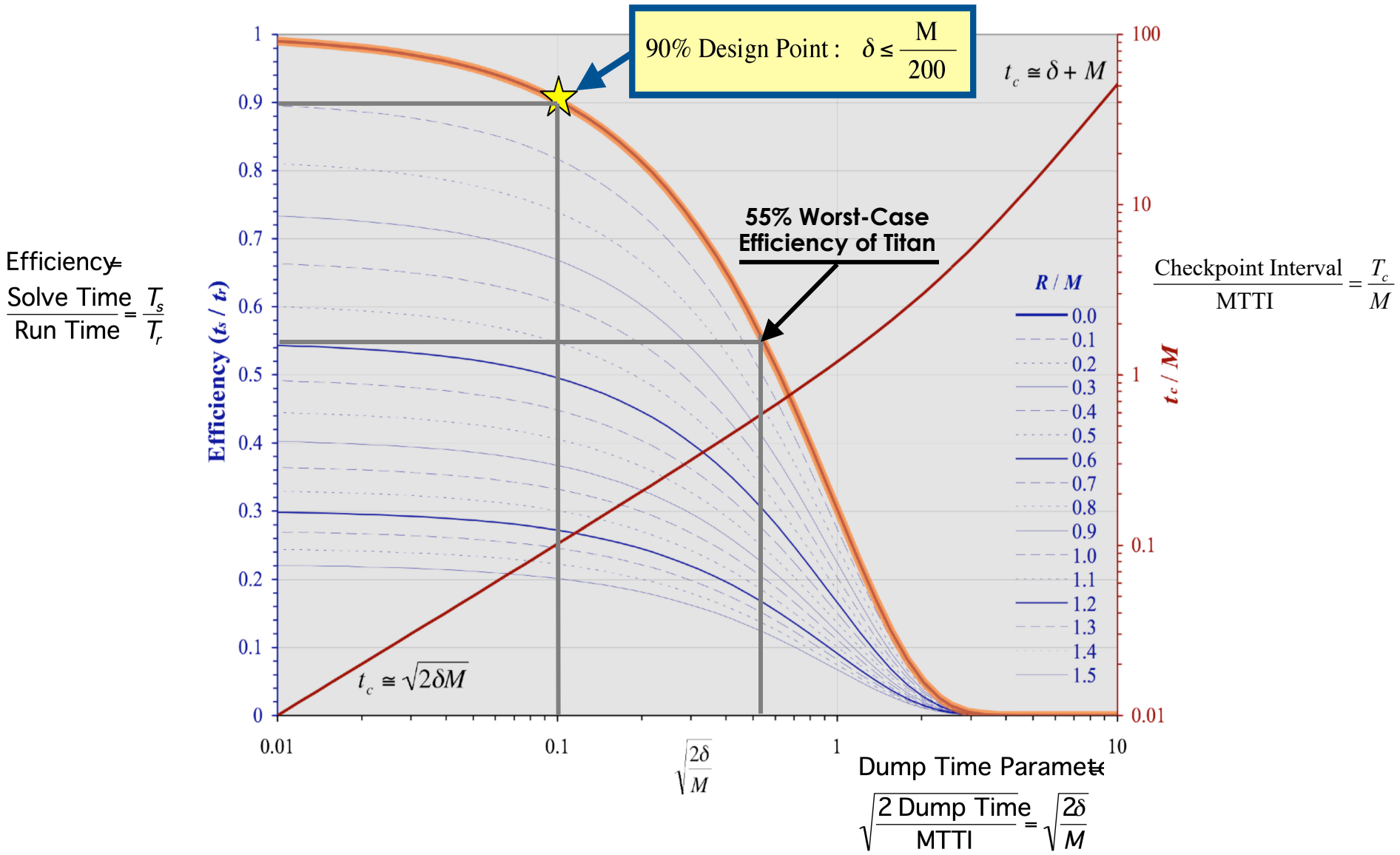
Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory

U.S. DEPARTMENT OF ENERGY

Background

- Resilience is a key challenge in extreme-scale computing
  - Less reliable components (worst case: 12 GPU failures/day on Titan)
  - More components (Summit has almost 50% more GPUs than Titan)
  - More dependencies (1 GPU failure takes a 6-GPU node out on Summit)
- Heterogeneity adds significant complexity to the extreme-scale hardware/software ecosystem
  - No fine-grain protection domains & resilience at the component level
  - Coarse-grain checkpoint/restart at the job level is standard


Titan GPU Failures (2014 - Present)
Weekly Failures
2014
2015
2016
2017
2018
2019
90
80
70
60
50
40
30
20
10
0
01/01/14 07/01/14 01/01/15 07/01/15 01/01/16 07/01/16 01/01/17 07/01/17 01/01/18 07/01/18 01/01/19 07/01/19
Date



J. T. Daly. Methodology and metrics for quantifying application throughput. In Proceedings of the Nuclear Explosives Code Developers Conference (NECDC) 2006, Los Alamos, NM, USA, Oct. 23-27, 2006.

Problem Statement

- The burden for providing resilience is currently on the user
  - Global checkpoint/restart is currently the only practical solution
- A programming model needs to provide resilience with an easy to use interface to permit wide-spread adoption
  - Have clear error & failure models and corresponding abstractions
  - Hide the complexities of protection domains and resilience strategies
  - Offer efficient resilience with little programming burden

Proposed Solution

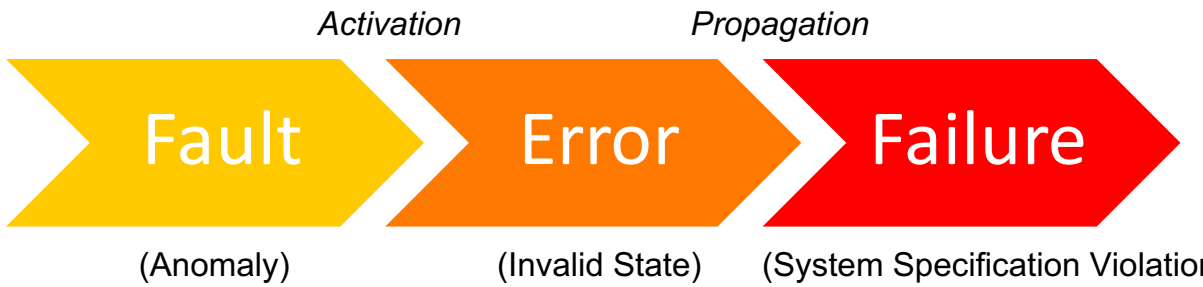
- Offer an easy to use and generic Quality of Service (QoS) interface for resilience
  - Use abstract, easy to understand, terms and programming constructs
  - Enable users to define their resilience needs
- Establish QoS contracts between the application and the system
  - Offer resilience QoS contract options for accelerator offload
  - Report contract breaches back to the application
- Embed QoS interfaces, coordination mechanisms and resilience strategies in the OpenMP language and runtime
  - An OpenMP that is resilient to accelerator errors/failures: **rOpenMP**

Accomplished/*Planned* Work

1. Create models of the impact of GPU errors and failures on the OpenMP runtime environment
2. Develop resilience strategies and corresponding protection domains for applications using OpenMP
3. Create OpenMP QoS language extensions to allow applications to describe resilience needs
4. Develop OpenMP runtime extensions *and policies* to meet application needs with resilience strategies
5. *Create prototype and demonstrate its capabilities with the miniQMC miniapp on Summit*

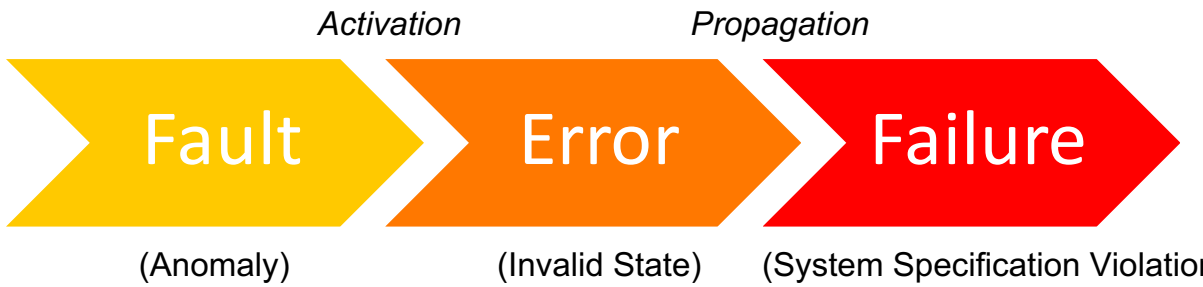
Models: General Fault Classes

- $\{benign, dormant, active\}$ $\{permanent, transient, intermittent\}$ $\{hard, soft\}$ *fault*
  - *Benign:* An inactive fault that does not activate.
  - *Dormant:* An inactive fault that potentially becomes active at some point.
  - *Active:* A fault that causes an error at the moment it becomes active.
  - *Permanent:* The presence of the fault is continuous in time.
  - *Transient:* The presence of the fault is temporary.
  - *Intermittent:* The presence of the fault is temporary and recurring.
  - *Hard:* A fault that is systematically reproducible.
  - *Soft:* A fault that is not systematically reproducible.
  - The following common terms map to these fault classes:
    - *Latent fault:* Any type of *dormant fault*.
    - *Solid fault:* Any type of *hard fault*.
    - *Elusive fault:* Any type of *soft fault*.



Models: General Error Classes

- $\{undetected, detected\}$ $\{unmasked, masked\}$ $\{hard, soft\}$ *error*
  - *Undetected:* An error whose presence is not indicated.
  - *Detected:* An error whose presence is indicated by a message or a signal.
  - *Masked:* An error whose impact is compensated so that the system specification is satisfied despite the incorrect state; its propagation is limited.
  - *Unmasked:* An error that has not been compensated and has the potential to propagate.
  - *Hard:* An error caused by a permanent fault.
  - *Soft:* An error caused by a transient or intermittent fault.
  - The following common terms map to these error classes:
    - *Latent error* or *silent error:* Any type of *undetected error*.
    - *Silent data corruption (SDC):* An *undetected unmasked hard* or *soft error*.



Models: General Failure Classes

- $\{undetected, detected\}$ $\{permanent, transient, intermittent\}$ $\{complete, partial, Byzantine\}$ *failure*
  - *Undetected:* A failure whose occurrence is not indicated.
  - *Detected:* A failure whose occurrence is indicated by a message or a signal.
  - *Permanent:* The presence of the failure is continuous in time.
  - *Transient:* The presence of the failure is temporary.
  - *Intermittent:* The failure is temporary but recurring in time.
  - *Complete:* A failure that causes service outage of the system.
  - *Partial:* A failure causing a degraded service within the functional specification.
  - *Byzantine:* A failure causing an arbitrary deviation from the functional specification.
  - The following common terms map to these failure classes:
    * *Fail-stop:* An *undetected* or *detected failure* that completely halts system operation, which often causes an irretrievable loss of state.
    * *Fail-safe:* A mode of system operation that mitigates the consequences of a system failure.

Models: OpenMP Target Offload Error and Failure Classes

- $\{undetected,\ detected\}\ \{unmasked,\ masked\}\ \{hard,\ soft\}\ target\ device\ error$
- $\{undetected,\ detected\}\ \{unmasked,\ masked\}\ \{hard,\ soft\}\ target\ task\ error$
- $\{undetected,\ detected\}\ \{permanent,\ transient,\ intermittent\}\ \{complete,\ partial,\ Byzantine\}\ target\ device\ failure$
- $\{undetected,\ detected\}\ \{permanent,\ transient,\ intermittent\}\ \{complete,\ partial,\ Byzantine\}\ target\ task\ failure$

Models: Mapping of Titan GPU Errors/Failures to Classes

Table 1. Mapping of Titan GPGPU errors to the OpenMP offloading error classes

Error	**Error Class**
Target device ECC double-bit error	Detected unmasked soft target device error
Target device SDC	Undetected unmasked soft target device error
Target task SDC	Undetected unmasked soft target task error

Table 2. Mapping of Titan GPGPU failures to the OpenMP offloading failure classes

Failure	**Failure Class**
Target device PCI width degrade	Detected transient partial target device failure Detected intermittent partial target device failure Detected permanent partial target device failure
Target device PCI disconnect	Detected permanent complete target device failure
Target device DPR	Detected transient complete target device failure
Target device SXM power off	Detected permanent complete target device failure
Target task abort	Detected permanent complete target task failure
Target task delay	Detected permanent partial target task failure

Strategies

- Error and failure detection and notification
  - Detections by the device/OS must be reported to the OpenMP runtime
    - Language feature (such as a callback) for application feedback is needed to potentially decide on the course of action (such as if an error is acceptable or not)
  - Detections by the application must also be reported to the runtime
    - Language feature for raising notifications to the OpenMP runtime is needed as well

Strategies

- Fail-fast and graceful shutdown
  - Detection, notification and controlled termination as soon as possible
  - Graceful shutdown avoids error propagation and failure cascades
  - Also enables proper error/failure reporting and root-cause analysis
  - Should be the default behavior of OpenMP runtime and applications

Strategies

- Graceful degradation
  - Continue operation after an error or failure at the cost of performance or correctness that is deemed acceptable
  - May mean to continue with less or slower devices
  - Requires runtime support to dynamically remove devices
- Rollback recovery
  - This we know how to do: Save task data and re-execute if needed
    - VOCL-FT has done this for OpenCL-accelerated applications
  - Language feature to limit the maximum number of rollbacks needed

Strategies

- Redundancy
  - Dual- or triple-redundant execution of tasks
  - Language feature to specify redundancy and type needed
  - Output comparison for error detection and masking
- Redundancy in time
  - Execute the same task at the same time on multiple devices
- Redundancy in space
  - Execute the same task on the same device multiple times

The Quality of Service (QoS) Approach

- Allow application developers to specify their resilience strategy without focusing on the implementation details
- Create a contract that maps application resilience requirements to the underlying hardware/software capabilities
- Specify the resilience strategy without focusing on implementation details

OpenMP QoS language extensions

- **QoS contract:** A set of QoS parameters that reflect resilience requirements by identifying resilience strategies
- **QoS parameters:** Generic *get/set* interface, using: (1) key/value pairs, (2) bounded values and (3) ranges of values
- **QoS parameter scope:** Code block and related data
- **QoS classes:** Offer coherent sets of parameters that achieve popular resilience strategies

OpenMP QoS language extensions

```
#pragma omp qoskv resilience (TASK_REDUNDANCY, BOOL, TRUE)
#pragma omp qoskv resilience (TASK_REDUNDANCY_FACTOR, INT, 3)
#pragma omp qoskv resilience (TASK_REDUNDANCY_MAJORITY, INT, 2)
#pragma omp qoskv resilience (TASK_REDUNDANCY_COMPARE, BOOL, TRUE)
{
  #pragma omp target ...
  ...
}
```

OpenMP QoS language extensions: QoS classes

```
#pragma omp qoskv resilience (TASK_TRIPLE_REDUNDANCY, BOOL, TRUE)
{
  #pragma omp target ...
  ...
}
```

Event-based OpenMP QoS runtime extensions

QoS Parameters

- Key/value pairs, bounded values, & ranges of values
- QoS classes

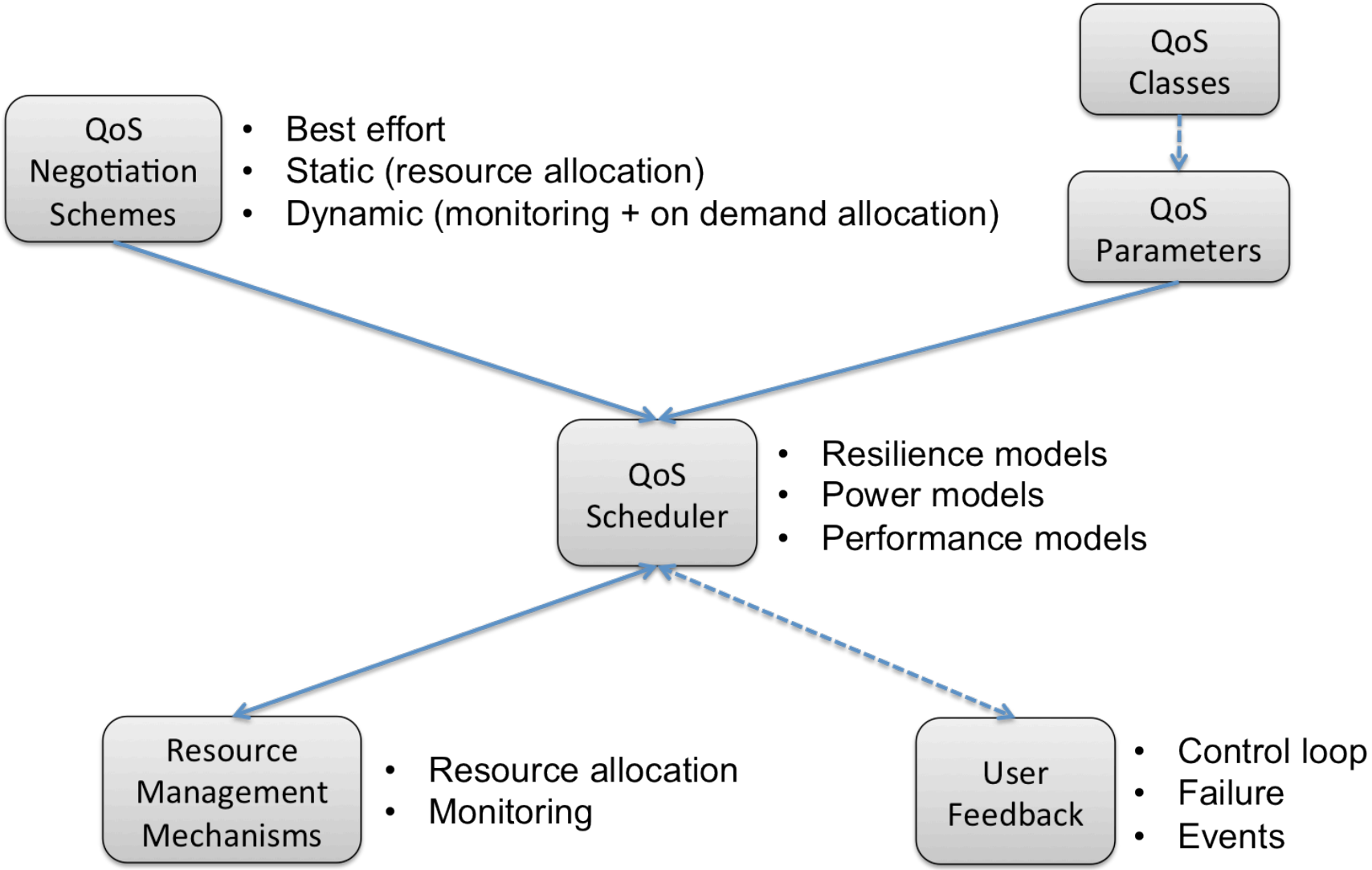
QoS Negotiation Schemes

- Guaranteed
- Best effort

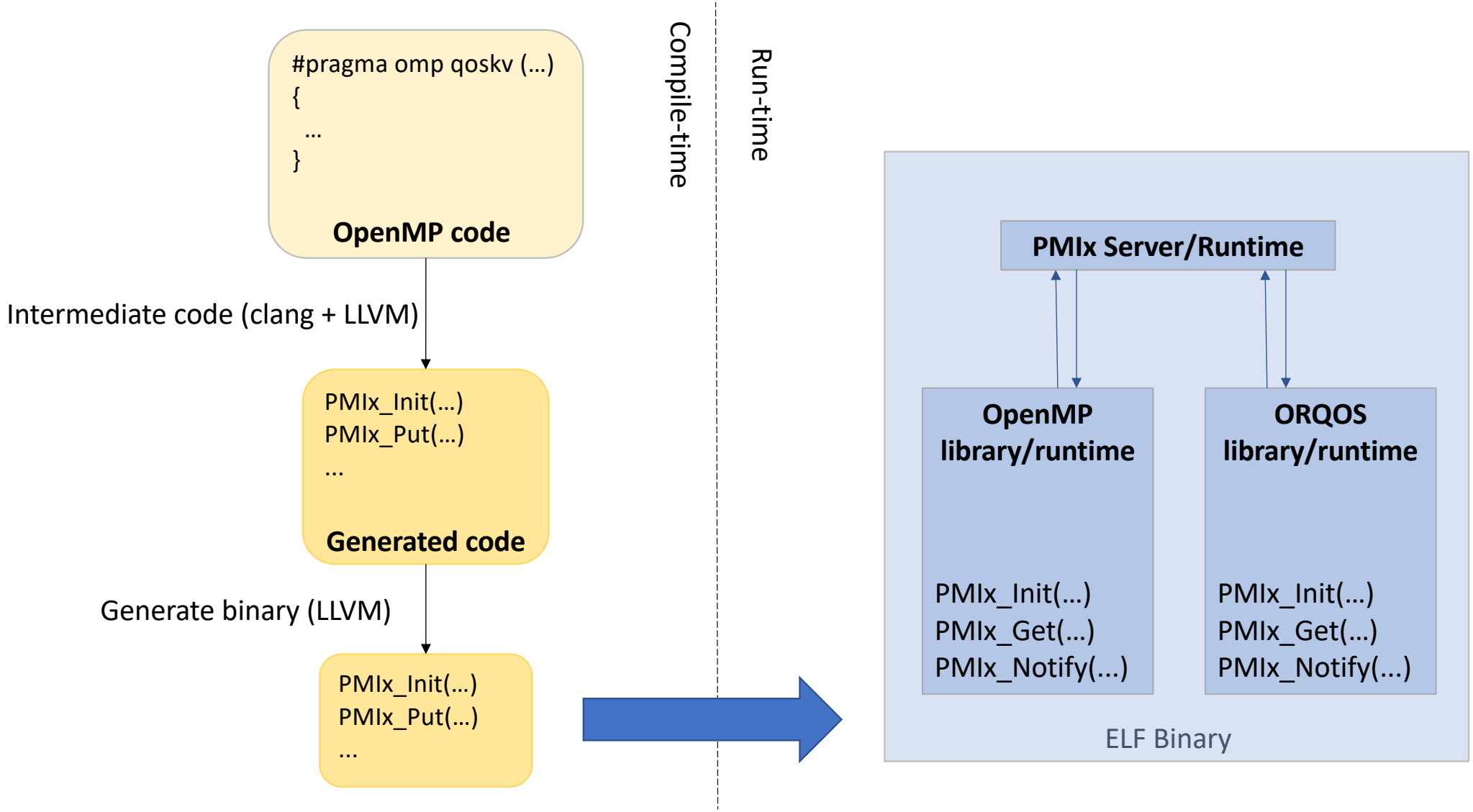
QoS Scheduler

- QoS contracts
- Resource management & monitoring
- Resilience responses & breach of contract notifications

Schematic Overview of the QoS Solution



Compile-time Workflow and Run-time Interactions of the Implemented Prototype using LLVM 7



Accomplishments

- Created models of the impact of GPU errors and failures on the OpenMP runtime environment
- Developed resilience strategies and corresponding protection domains for applications using OpenMP
- Created OpenMP QoS language extensions to allow applications to describe resilience needs
- Developed OpenMP runtime extensions to meet application needs with resilience strategies

Future Work

- Further extend the OpenMP QoS language and runtime extensions
- Create QoS policies to meet application needs with strategies
- Create the final prototype and demonstrate its capabilities
- Expand the QoS concept to other aspects and the trade-off between them: performance, resilience and energy
- Expand the QoS concept to MPI and MPI+OpenMP
- Create intent-based QoS extensions for architecture agnostic approaches
- Develop an adaptive runtime with self-awareness

OAK RIDGE National Laboratory

Concepts for OpenMP Target Offload Resilience

Christian Engelmann (ORNL)
Geoffroy R. Vallée (Sylabs, Inc)
Swaroop Pophale (ORNL)

Contact: engelmannc@ornl.gov

ORNL is managed by UT-Battelle, LLC for the US Department of Energy

Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory

U.S. DEPARTMENT OF ENERGY