# HPC2014

Yuu Ohmura (Watanabe Lab.)
2015/01/19

# Today's Paper

## A System Software Approach to Proactive Memory-Error Avoidance

**Carlos H. A. Costa - IBM T. J. Watson Research Center, Yorktown Heights, NY**

**Yoonho Park - IBM T. J. Watson Research Center, Yorktown Heights, NY**

**Bryan S. Rosenburg - IBM T. J. Watson Research Center, Yorktown Heights, NY**

**Chen-Yong Cher - IBM T. J. Watson Research Center, Yorktown Heights, NY**

**Kyung Dong Ryu - IBM T. J. Watson Research Center, Yorktown Heights, NY**

**SC '14 Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis**

# Outline

I. INTRODUCTION

II. MEMORY HEALTH AND FAILURE PREDICTABILITY

III. PROACTIVE MEMORY MANAGEMENT

IV. EXPERIMENTAL EVALUATION

V. RELATED WORK

VI. CONCLUSION AND FUTURE WORK

# I. INTRODUCTION

- **Frequent system failure due to an increased number of memory errors is a major obstacle to scaling current HPC technology[1].**

# Memory failures

**Memory failure are cased by soft or hard errors.**

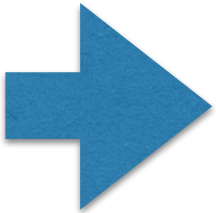
- **Soft errors are transient errors.**
  - **ex) neutrons and alpha particles**
- **Hard errors are permanent, recurring errors.**
  - **ex) wear-and-tear and aging of transistors and metal**

**ECC mechanisms can address both soft and hard errors.**

# The predictability of memory hard error

**Several studies point to the predictability of memory hard errors [5],[6].**

- **the potential benefits in the early identification of unhealthy memory**
- **how the occurrence of correctable memory errors is concentrated in a small fraction of memory in few node**



**using prediction algorithm based on correctable error pattern, proactively avoid using the failing memory**

# Proactive Memory-Error Avoidance

There are two main challenges.

1. The first challenge is to enable real-time access to corrected-error information.

2. The second challenge is to efficiently analyze the error information.

# II. MEMORY HEALTH AND FAILURE PREDICTABILITY

**They derived a generic prediction algorithm by analyzing a raw RAS log.**

**RAS ··· Reliability and Availability Service**

# Correctable memory error reporting format in BG/P

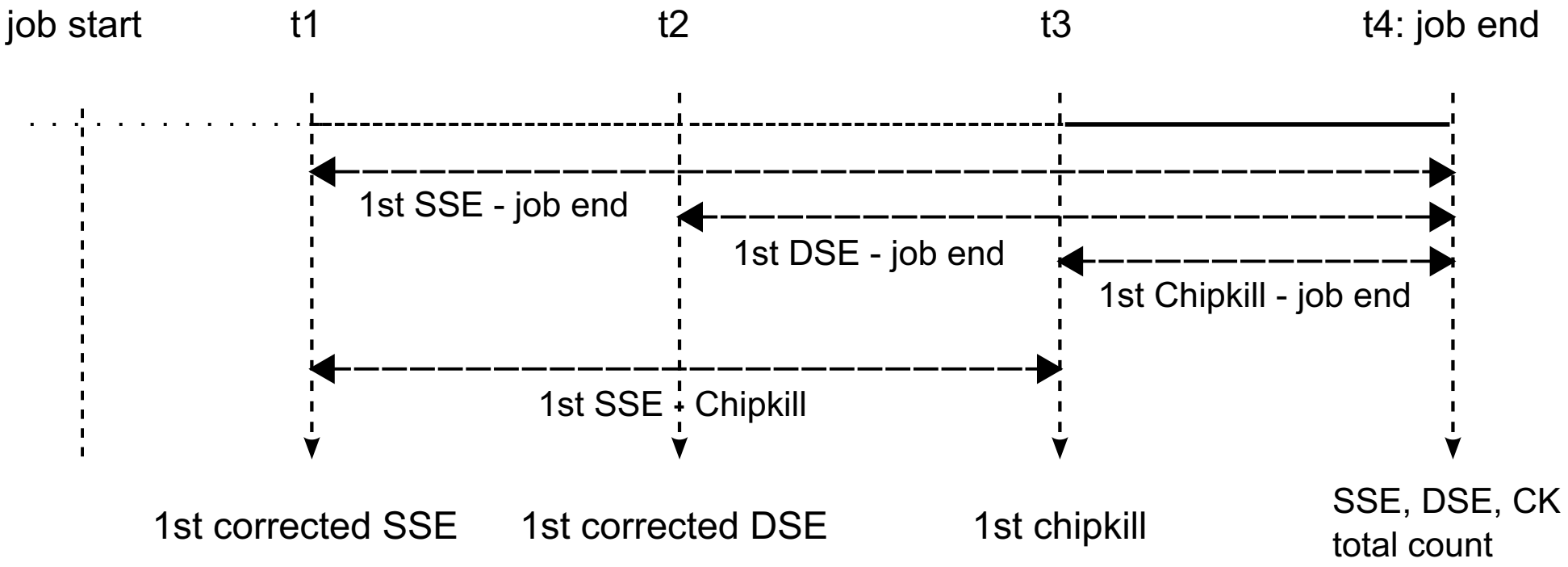



Fig. 1. Correctable memory error reporting format in BG/P.

# A. Spatial Correlation(1/2)

- They show how the spatial distribution of correctable errors can be used to identify nodes with a high chance of developing non-trivial error corrections.

- One approach to identify a faulty area is to identify the repetition of correctable errors in the same address.

# A. Spatial Correlation(2/2)

- **Two very distinct sets are observed.**

  1. **are nodes that experienced instances of single-symbol errors.**

  2. **are nodes that activated Chipkill.**

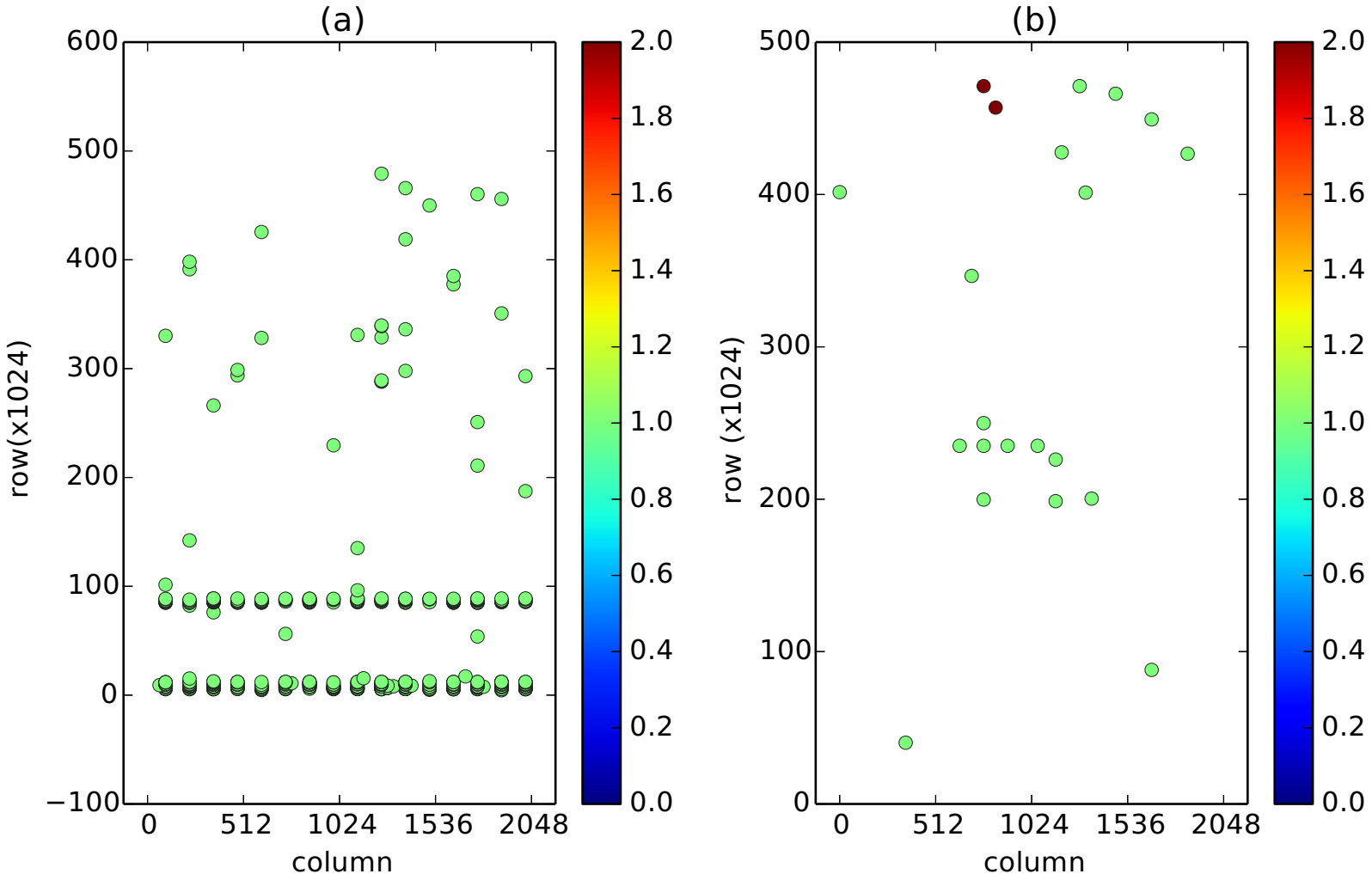



Fig. 4. Example of error address distribution in a memory bank for two nodes not activating Chipkill: (a) R21-M0-N10-J30 and (b) R44-M1-N01-J26.

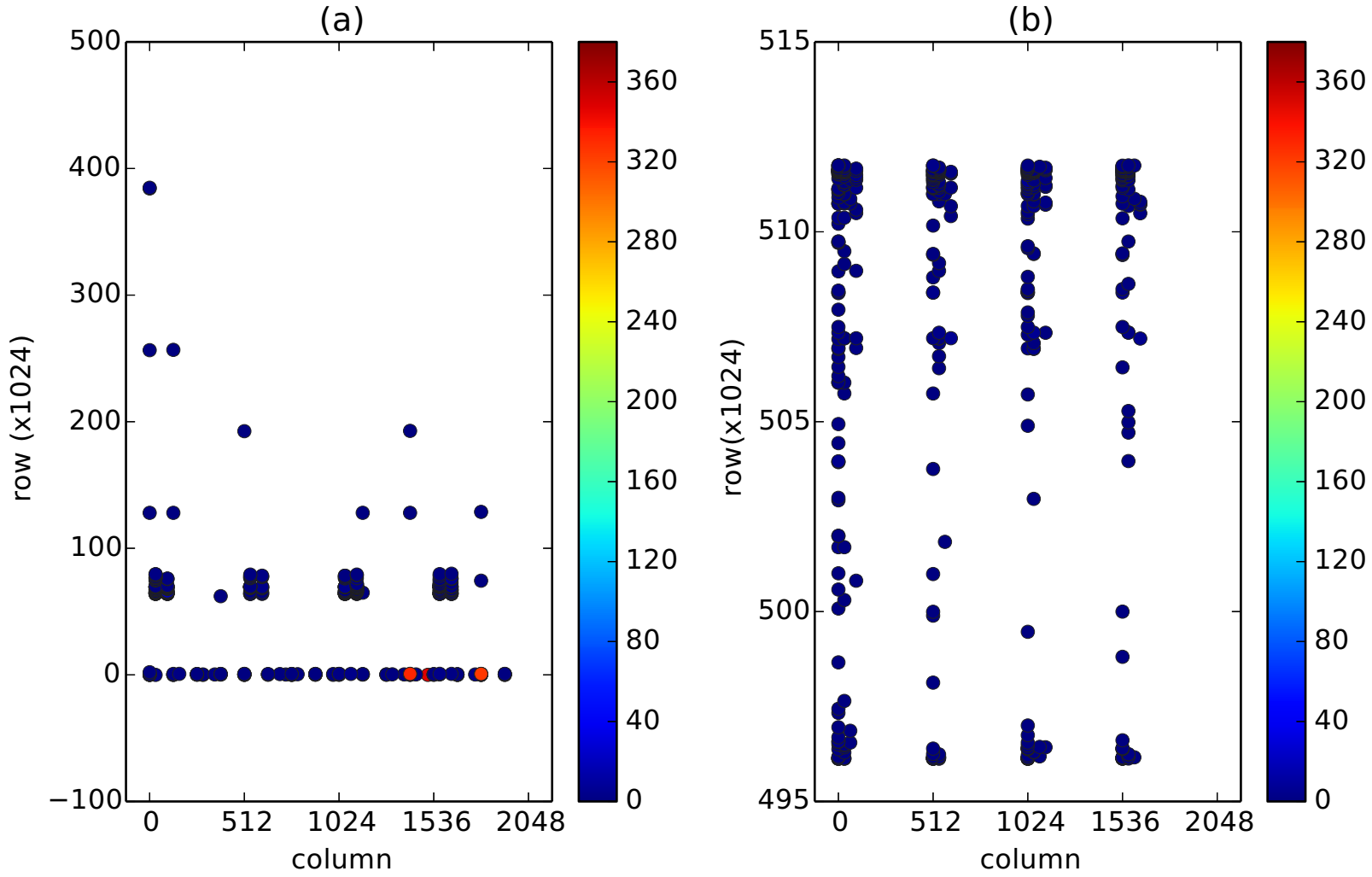



Fig. 2. Example of error address distribution in a memory bank for two nodes activating Chipkill: (a) R34-M0-N13-J24 and (b) R20-M0-N14-J24.

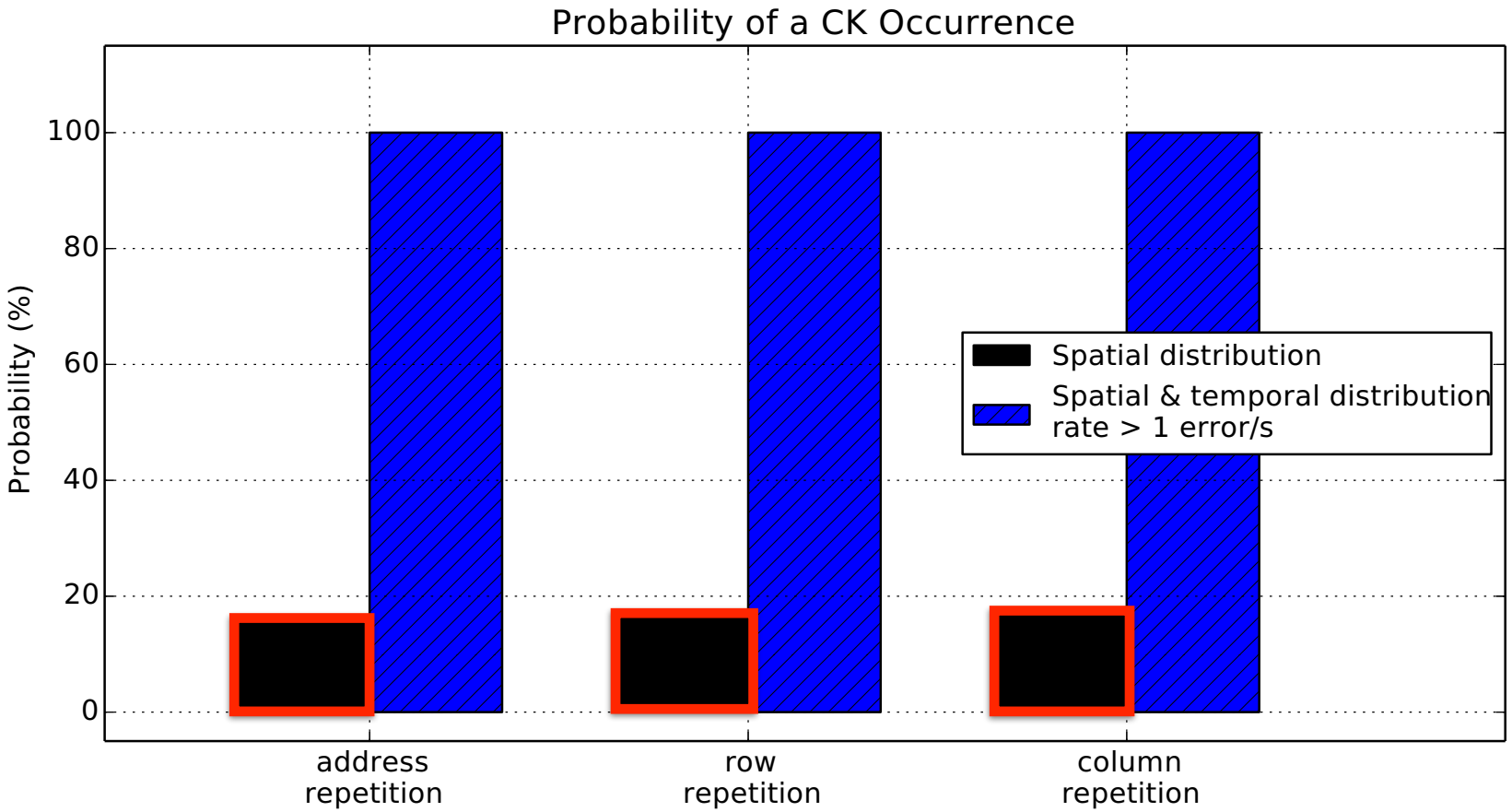



Fig. 3. Probability of observing a Chipkill event based on spatial and temporal distribution of repeated correctable errors for all jobs.

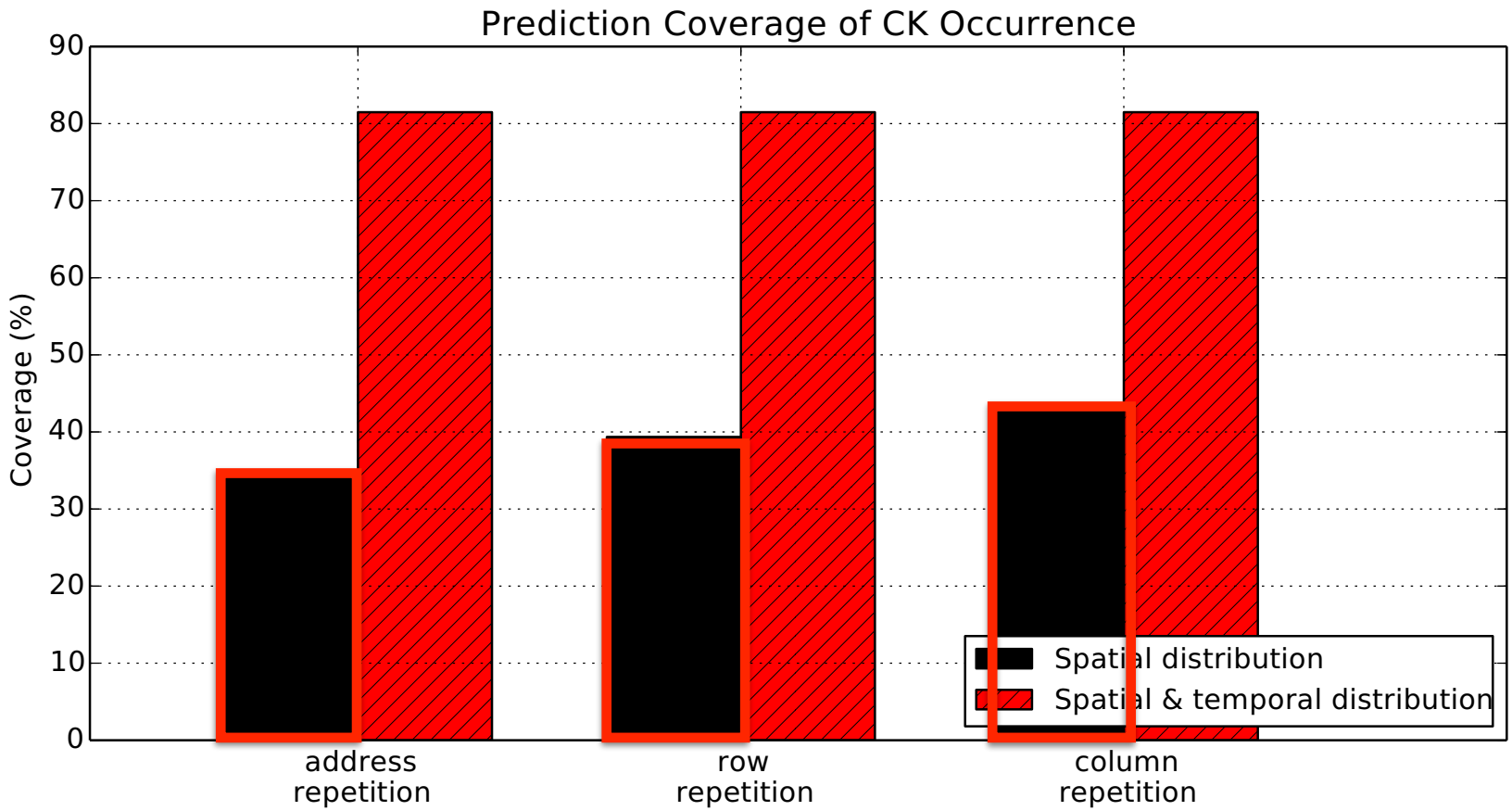



Fig. 5. Coverage of Chipkill prediction based on spatial and temporal distribution of repeated correctable errors for all jobs.

# B. Temporal Correlation

- We use timing information.
  - an error rate
    - based on the first occurrence of an error and the total number of errors.
  - error timestamps
    - measure the average time between the first occurrence of a single-symbol error and a potential Chipkill correction

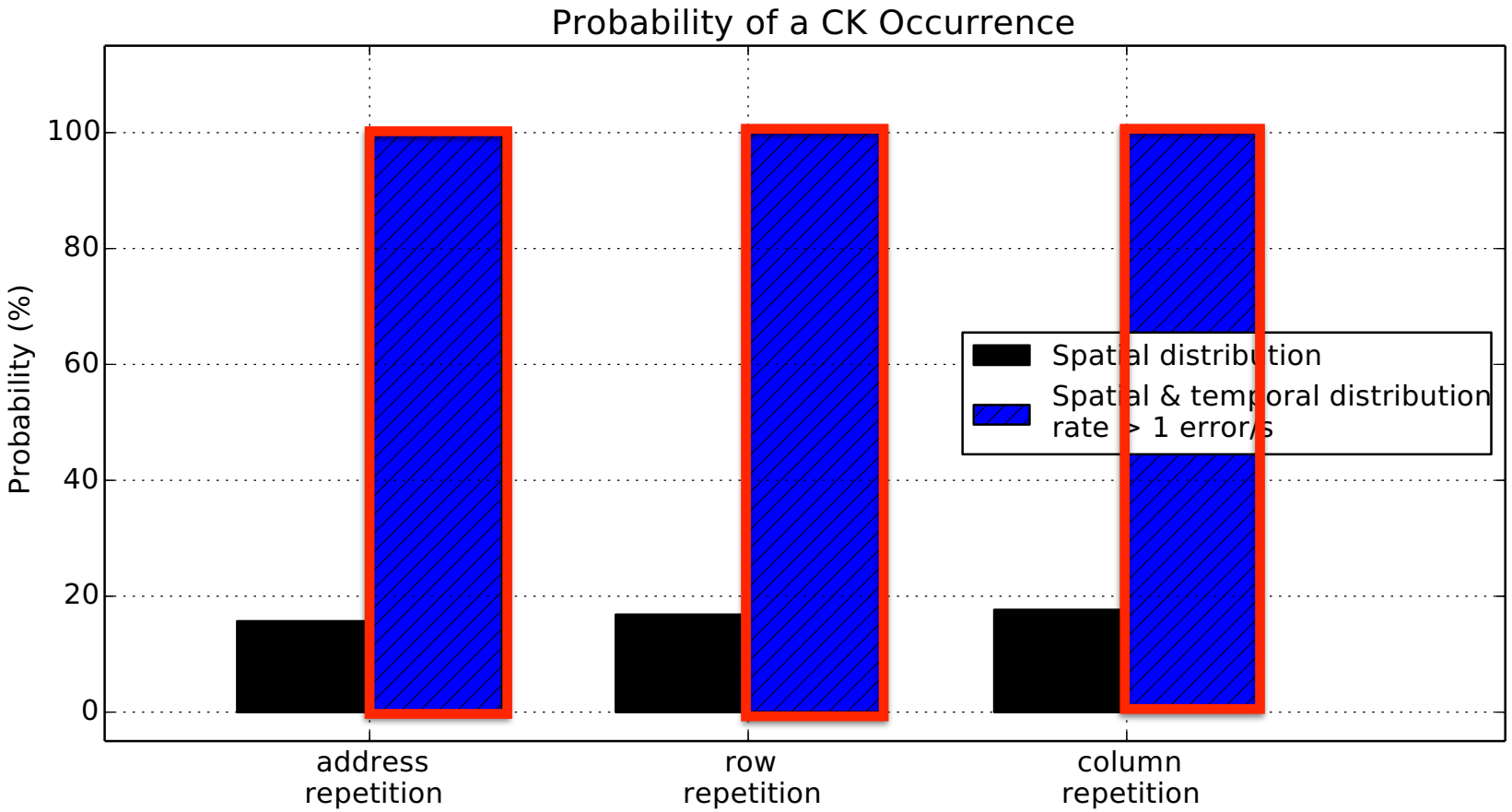



Fig. 3. Probability of observing a Chipkill event based on spatial and temporal distribution of repeated correctable errors for all jobs.

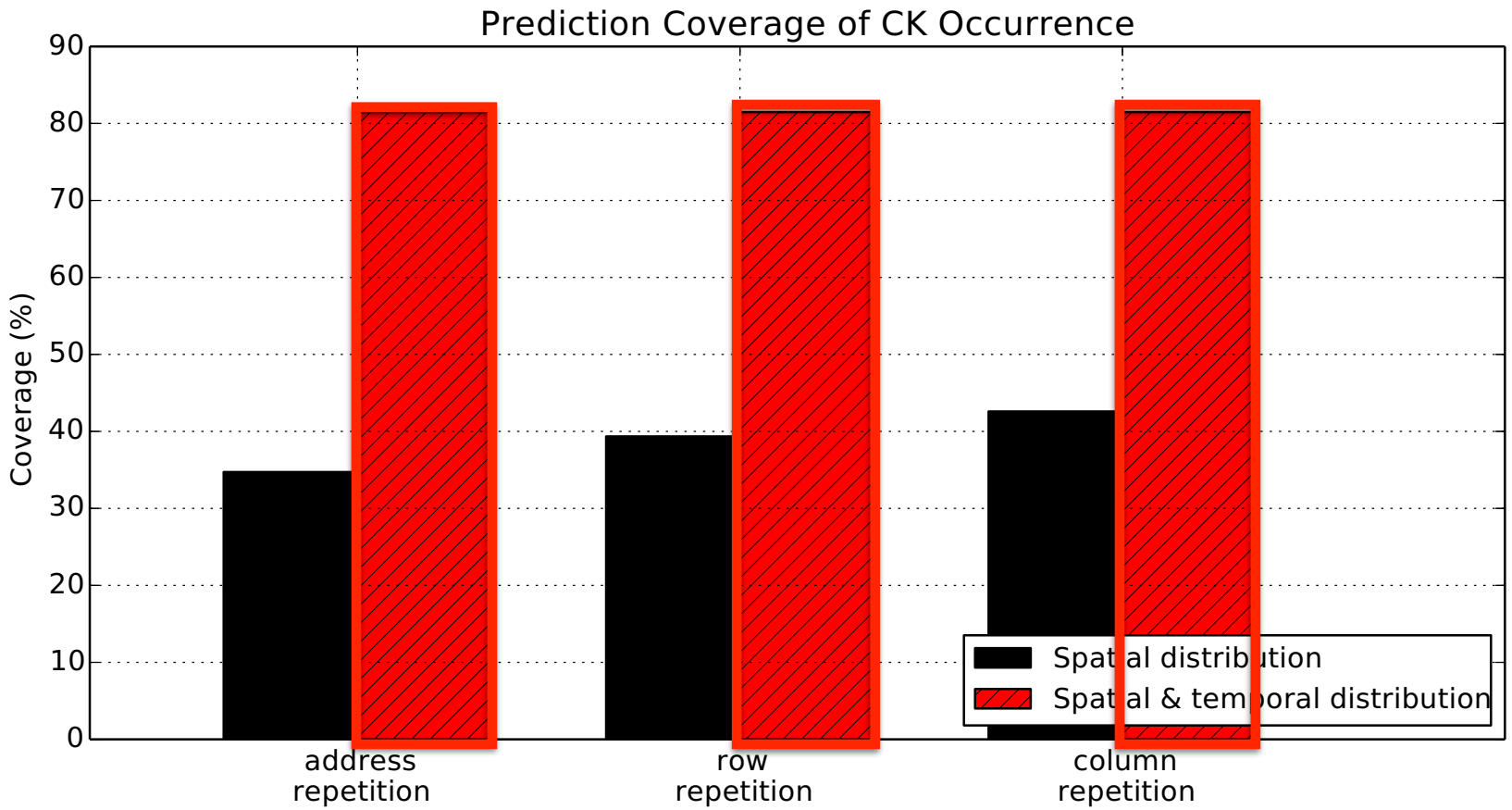



Fig. 5. Coverage of Chipkill prediction based on spatial and temporal distribution of repeated correctable errors for all jobs.


Fig. 6. Example showing correlation of correctable error counts: total error counts for all jobs run on node R04-M1-N01-J17.

TABLE II. TIME ELAPSED BETWEEN FIRST SINGLE-SYMBOL ERROR OBSERVED AND FIRST CHIPKILL CORRECTION.

| $<$ 1s | 1s $\rightarrow$ 1m | 1m $\rightarrow$ 1h | 1h $\rightarrow$ 1 day |
|---|---|---|---|
| 26.21% | 63.57% | 9.51% | 0.69% |

# C. Chipkill and Uncorrected Errors(1/2)

- **In our analysis we focus on relating single-symbol error and Chipkill for two reasons.**

1. **The majority of uncorrectable errors cause multiple failures that corrupt the error reporting, making it difficult to accurately time uncorrectable memory errors and calculate latency.**

2. **Repeated Chipkills are strongly correlated with subsequent failure.**

# C. Chipkill and Uncorrected Errors(2/2)

- The anticipation and avoidance of repeated non-trivial error correction has multiple benefits.

1. It allows us to avoid faulty memory before it is too late to prevent a failure.

2. the repeated activation of a costly and complex error correction can be seen as an efficiency problem in itself.

# III. PROACTIVE MEMORY MANAGEMENT

- **Approach**
  - **monitoring memory degradation**
  - **reallocating memory proactively to avoid faulty memory**
- **Goal**
  - **Dynamically migrate data residing in memory pages**

# A. Design

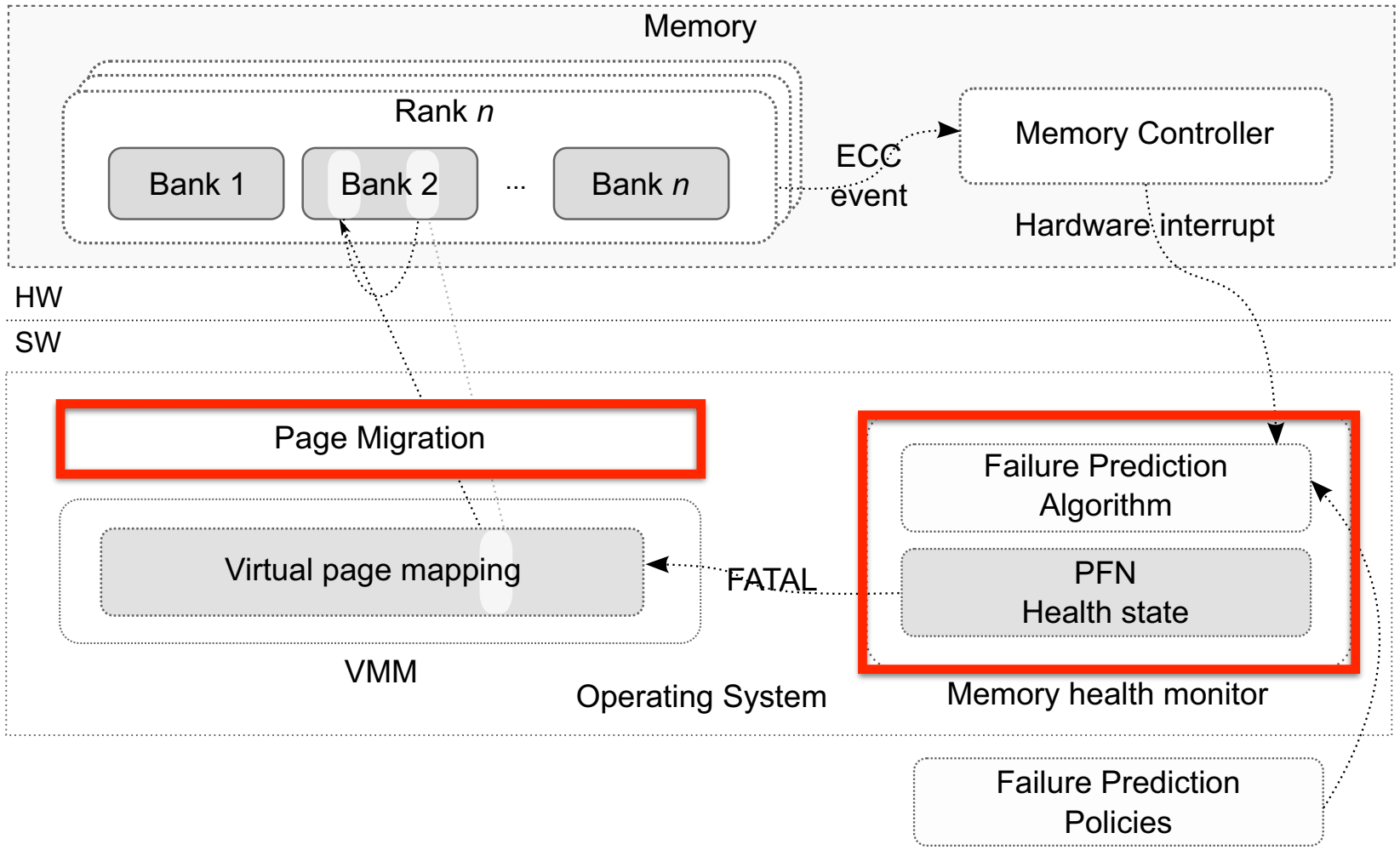



Fig. 7. Memory health monitoring mechanism.

**Require:** physical address, last occurrence of error, error type

1: `addr` ← physical address
2: Get `PFN` for `addr`
3: Search `PFN` in pages hash table (PHT)
4: **if** `PFN` $\notin$ `PHT` **then**
5: Get time and error type
6: Add `PFN` to PHT
7: Add `addr` to address hash table (AHT)
8: **else**
9: Search `addr` in AHT
10: **if** `addr` $\in$ AHT **then**
11: Increment current repetition count
12: Get current time and last occurrence in `PFN`
13: Calculate error rate $E_r$
14: **if** $E_r >$ error threshold **then**
15: Update health state of `PFN` to *fatal*
16: **else**
17: Update health state of `PFN` to *unhealthy*
18: **end if**
19: **else**
20: Update health state of `PFN` to *healthy*
21: Add `addr` to AHT
22: **end if**
23: **end if**

Fig. 8. Algorithm for failure prediction.

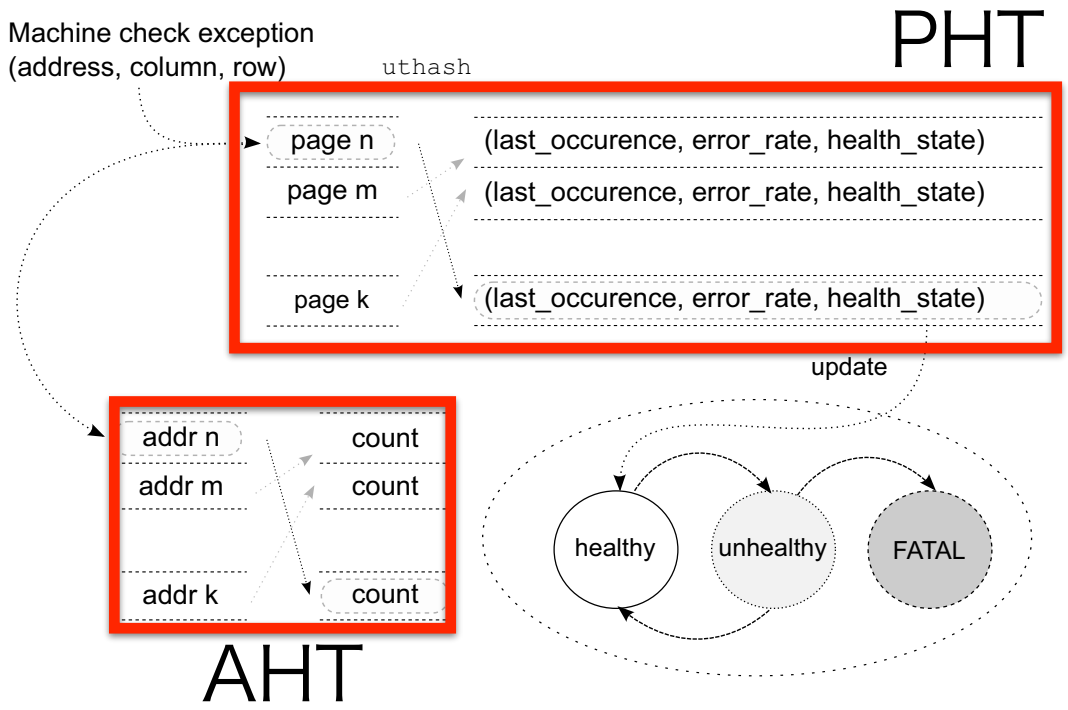



Fig. 9. Mechanism to monitor the health states of memory pages.

# B. Implementation and Prototype

- Target
  - Linux kernel that runs on BG/Q nodes.
  - a BG/Q node
    - processor: 17 core, 4 threads
    - memory: 16GB

# Components of the prototype

1) Interrupt Controller Reconfiguration

2) Interrupt Handler and Health Monitor

3) Page Migration in Linux

4) Error Emulation Through BG/Q Firmware

# 2) Interrupt Handler and Health Monitor

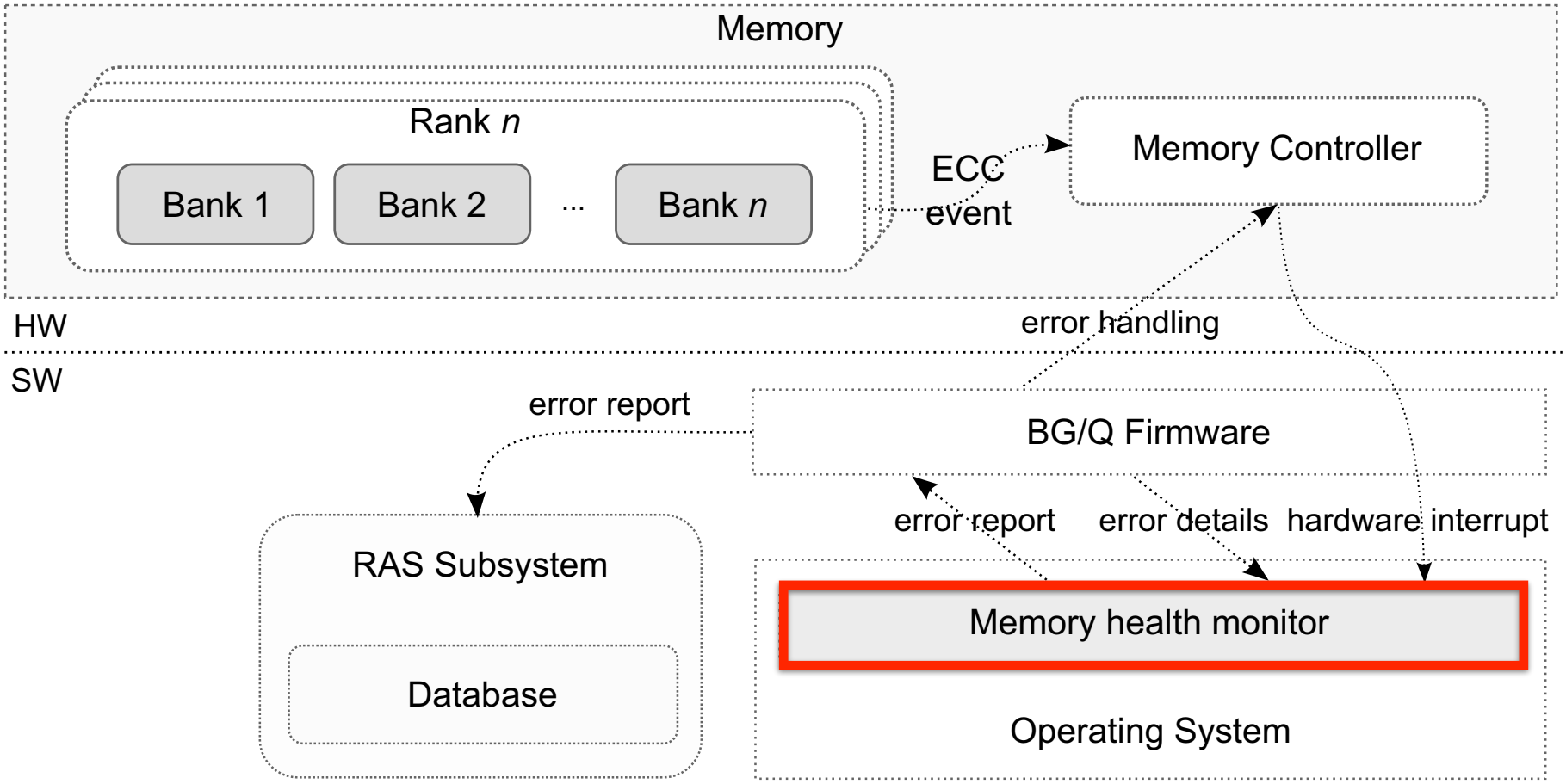



Fig. 10. BG/Q firmware modifications for health monitoring.

# 3) Page Migration in Linux

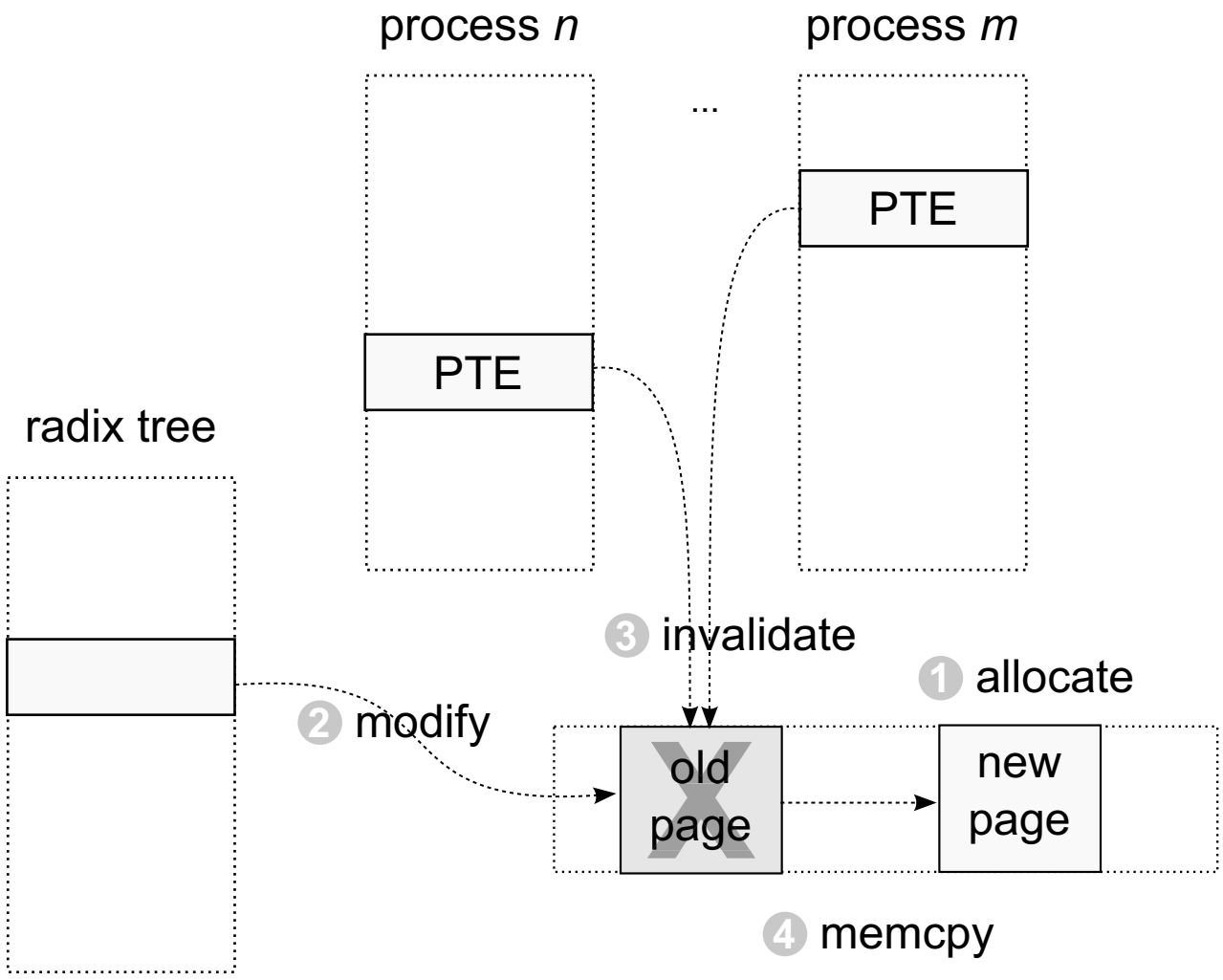



Fig. 11. Page migration in the Linux kernel.

# IV. EXPERIMENTAL EVALUATION

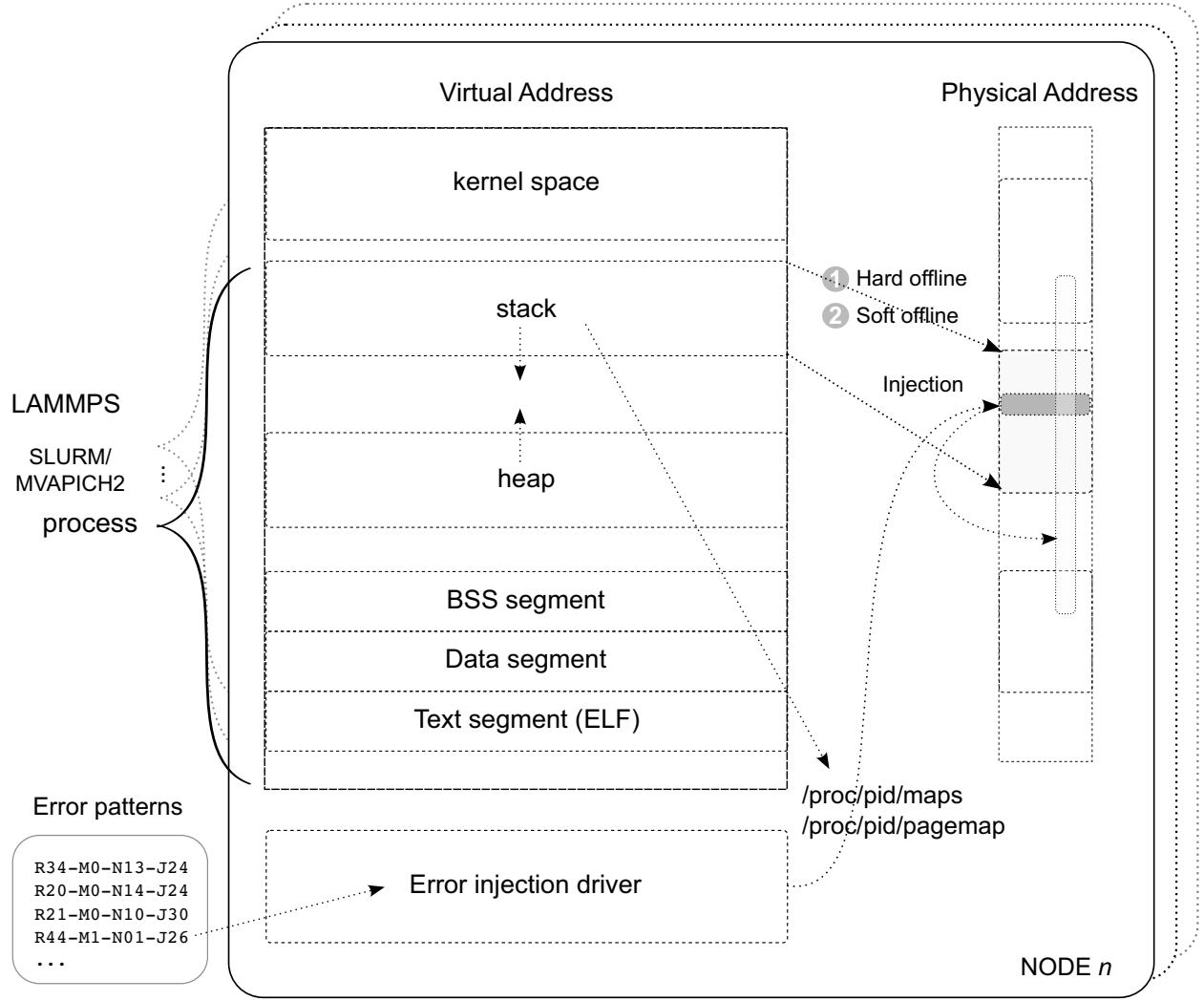
# A. Benchmark and Parallel Execution Environment

- LAMMPS
  - is a classical molecular dynamics code, and an acronym for Large-scale Atomic/Molecular Massively Parallel Simulator
  - version: 1Feb14
- Goal
  - predict parallel efficiency for large systems

# B. Error Injection and Reproduction of Error Patterns

- **Base**
  - **Idealistic case(no error)**
- **Case 1**
  - **Node with unhealthy memory with several and widespread repeated errors, including repeated Chipkill correction.**
- **Case 2**
  - **Node with unhealthy memory with several errors concentrated in few rows.**
- **Case3**
  - **Node with healthy memory, with early signs of degradation. Correctable errors are experienced, but no repetition is observed.**

# C. Survivability




...
1030415; 1681252; 0x03ff4d400; 250; 874; 65535
1681252; 2163075; 0x03ff43c00; 3; 11; 880
2226994; 2765782; 0x03feb6620; 125; 498; 65535
...

# D. Performance Overhead

- **We evaluated the impact of tracking errors and performing page migration on resource utilization and execution time.**

**1) Single-Node Scenario**

**2) Multi-Node Scenario**

**3) Considerations on Memory Locality and Large Page**

# D. Performance Overhead

# 1) Single-Node Scenario

- **a single-node run of LAMMPS**

TABLE III. RESOURCE UTILIZATION OF LAMMPS IN A SHORT SINGLE-NODE (4 OPENMP THREADS) RUN WITH PAGE MIGRATION.

| scenario | real | user | sys | sys incr. (%) |
|---|---|---|---|---|
| baseline | 23m 26.18s | 1h 33m 40s | 0m 3.02s | |
| case 1 | 23m 27.55s | 1h 33m 42ss | 0m 3.63s | **20.20** |
| case 2 | 23m 27.83s | 1h 33m 45s | 0m 3.66s | **21.19** |
| case 3 | 23m 26.22s | 1h 33m 40s | 0m 2.96s | **-1.99** |

TABLE IV. RESOURCE UTILIZATION OF LAMMPS IN A LONG SINGLE-NODE (4 OPENMP THREADS) RUN WITH PAGE MIGRATION.

| scenario | real | user | sys | sys incr. (%) |
|---|---|---|---|---|
| baseline | 3h 55m 05s | 15h 39m 41s | 31.54s | |
| case 1 | 3h 55m 05s | 15h 39m 41s | 32.05s | **1.62** |
| case 2 | 3h 55m 06s | 15h 39m 46s | 30.73s | **-2.57** |
| case 3 | 3h 55m 06s | 15h 39m 47s | 31.56s | **0.06** |

# D. Performance Overhead 2)Multi-Node Scenario

- **a multi-node run of LAMMPS**

TABLE V. SUMMARY OF RESOURCE UTILIZATION ACROSS MULTIPLE NODES: BASELINE AND PAGE MIGRATION COMPARISON FOR LONG RUN OF LAMMPS ON 64 NODES, 6 DIFFERENT RUNS.

| | **real time** | **incr. (%)** | **mean user time** (ticks $10^3$) | **incr. (%)** | **mean sys time** (ticks $10^3$) | **incr. (%)** |
|---|---|---|---|---|---|---|
| baseline | 293m46.131s | | 1736.48 | | 8.424 | |
| avg. with page migration | 292m58.317s | **-0.27** | 1733.74 | **-0.16** | 8.437 | **0.14** |

# E. Effectiveness

- compare the number of errors avoided to the total number of errors

# E. Effectiveness

## 1) Single-Node Scenario

TABLE VI. EFFECTIVENESS AND MEMORY OVERHEAD WITH PAGE MIGRATION: SINGLE-NODE SHORT RUN.

| **scenario** | **SSE** avoid.(%) | **DSE** avoid.(%) | **CK** avoid.(%) | **memory retired** | **% of app. memory** |
|---|---|---|---|---|---|
| case 1 | 27.3 | 23.9 | **71.5** | 512 KB | 2.52 |
| case 2 | 86.69 | 91.97 | **91.87** | 832 KB | 4.10 |
| case 3 | 0 | n/a | n/a | 0 | 0 |

# E. Effectiveness
# 2)Multi-Node Scenario

TABLE VII. EFFECTIVENESS AND MEMORY OVERHEAD WITH PAGE MIGRATION: 64 NODES, 6 DIFFERENT RUNS.

| | **SSE** avoid.(%) | **DSE** avoid.(%) | **CK** avoid.(%) | | **memory retired** | **% of app. memory** |
|---|---|---|---|---|---|---|
| best average | 76.34 | 71.42 | **80.42** | max. | 1152 KB | 0.6 |
| worst average | 71.28 | 63.39 | **74.80** | min. | 64K KB | 0.03 |
| **avg. run** | 76.07 | 68.65 | **76.42** | | | |

# E. Effectiveness

## 3) Memory Availability

Their solution can avoid the majority of memory errors using a very small amount of reserved memory

## 4) Error Rate Threshold and Effectiveness

A lower threshold would potentially provide higher coverage

## 5) Overall Resilience Improvement

the avoidance of 63.43% of memory failures

# VI. CONCLUSION AND FUTURE WORK

- **Conclusion**
  - **They have shown how correctable error information can be used by the OS to avoid repeated memory error and failure.**
  - **They have implemented a prototype.**
  - **They have evaluate the prototype on some environment.**

# VI. CONCLUSION AND FUTURE WORK

- future work
  - the impact of repeated error avoidance on performance and power consumption
  - deploy and evaluate there mechanism in a production system