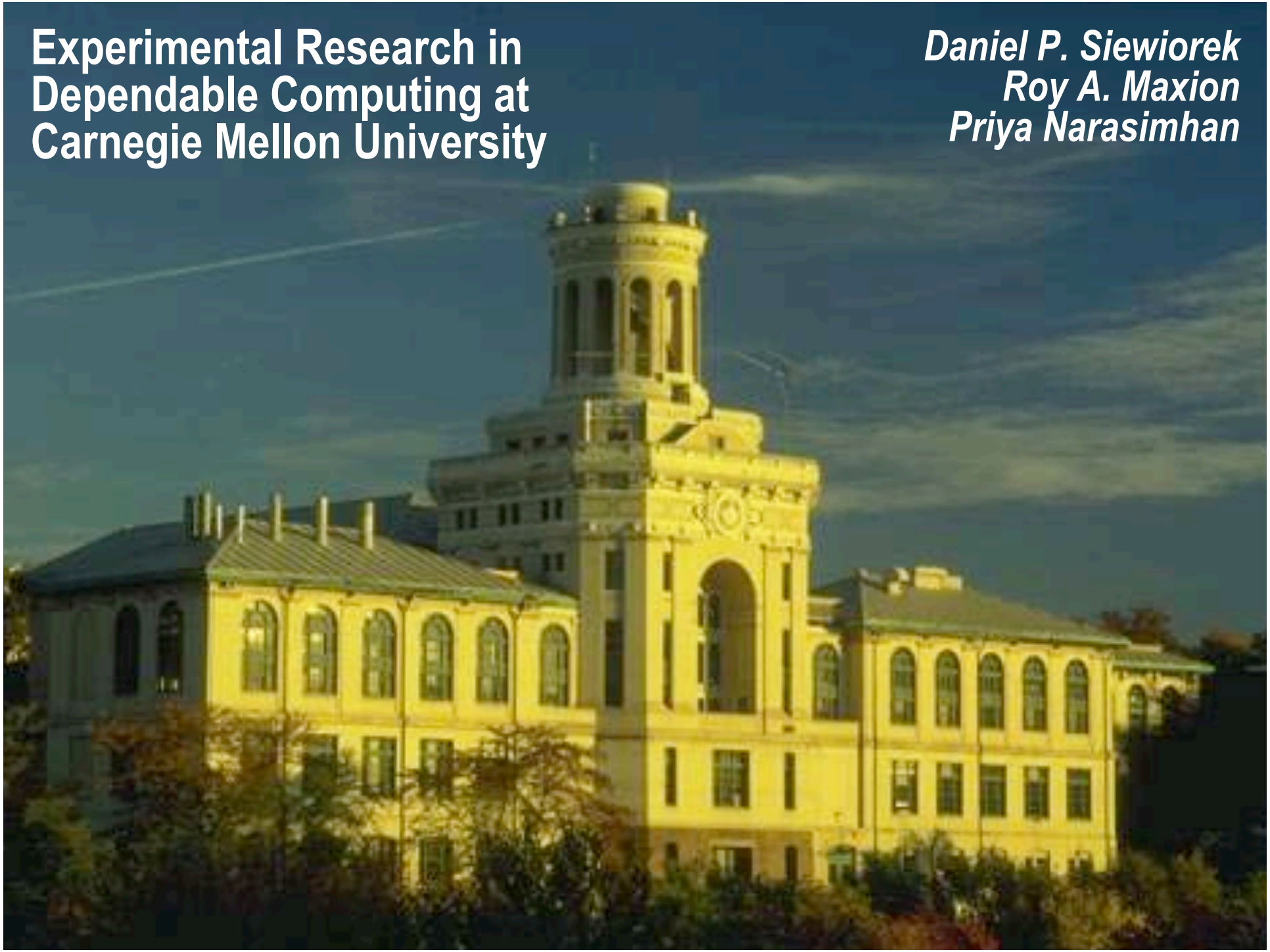
Experimental Research in Dependable Computing at Carnegie Mellon University

Daniel P. Siewiorek
Roy A. Maxion
Priya Narasimhan



Authors

Daniel P. Siewiorek
Joined CMU: 1972

Roy A. Maxion
Joined CMU: 1984

Priya Narasimhan
Joined CMU: 2001

My Background

- **Prior research on dependable enterprise systems**
  - Developed systems that provide "out-of-the-box" reliability to middleware
    - No need to change application or ORB code
  - **Eternal**: Fault-tolerant CORBA/Java support
  - **Immune**: Secure CORBA/Java support
- **Helped to establish the Fault-Tolerant CORBA standard**
  - Served as CTO & VP (Eng.) of startup company to commercialize research
- **Current research/teaching focus at CMU**
  - Continuing research on dependable embedded middleware
    - **MEAD**: Real-time fault-tolerant middleware support
    - **Starfish**: Secure scalable middleware support
  - Teaching courses on
    - Developing real-time fault-tolerant high-performance middleware
    - Embedded systems: Device drivers, interrupts, protocols, real-time, etc.

In the Beginning, There Was ……

- **The Carnegie Plan for higher education (1945)**
  - Emphasis on "learning by doing"
  - One example is the experimental dependability research at CMU
- **Westinghouse Research Corporation in Pittsburgh (1960s)**
  - Research in the use of active redundancy to enhance reliability
  - CMU researchers involved in this effort, leading to a book by Mann (1962)
- **During the next three decades, and continuing into this decade**
  - Several experimental hardware and software systems were designed, implemented and made operational at CMU
    - Each was an opportunity to understand, and advance research in, reliability
    - Each involved significant data-collection and experimentation
  - What have we focused on?
    - Understanding the natural occurrence of faults
    - Mathematical models for fault-prediction (backed by empirical evidence)
    - Raising the level of abstraction of fault models to design dependability better

Chronology & Diversity of CMU Research

	1970's	1980's	1990's	2000's
Monitoring	Crash Dumps (1975)	Error Logs (1980)	Natural Workloads (1990)	Distributed, Asynchronous
Fault Model	Gate Level	Register Transfer	Design, User Errors, Reactive	Attacks, Proactive (2004)
Fault Injection	Stuck-At	Memory Level (1985)	API-Level (1995)	Security (2000), Resource Exhaustion
Abstractions	Stuck-At	Error Logs ⇨ Clustering ⇨ Space/DFT (1986)	Gate ⇨ RT, Message ⇨ Fault feature vector, Memory ⇨ Crash (1995)	Multi-Dimensional
Modeling	Event, Mathematical Distribution/Parameters (1975)	Fault and Workload Interaction (1985)	Event Clustering, Trend Analysis, Prediction (1995)	Machine Learning

Overview of Talk

- **Multiprocessor architectures**
  - C.mmp, Cm*., C.vmp, redundant ECC-based disk-arrays
- **Hard and transient fault distributions**
  - Experimental data collection
- **Trend analysis**
  - Understanding event-logs; monitoring, diagnosis & prediction techniques
- **Robustness testing**
  - Black-box testing using injection of anomalous inputs at interfaces
- **The next decade**
  - Distributed fault-prediction, proactive fault-tolerance, machine learning and adaptation, tunability

Multiprocessor Architectures

- **Gordon Bell headed up a project for C.ai (1969)**
  - Architectures designed for artificial intelligence applications
  - One part consisted of a multiprocessor that evolved into C.mmp
- **DARPA-funded C.mmp project**
  - Started 1971, became operational mid-1975, decommissioned 1980
  - Sixteen PDP-11 processors communicating with 16 memories through a crossbar switch
  - H-shaped configuration – cross-bar switch and memory in the middle, flanked by banks of four processors
  - Natural redundancy in its replicated processors and memory provided opportunities for substantial software error-detection and reconfiguration techniques
- **CMU research on analytical models of reliability and performance in the context of C.mmp**

C.mmp (Computer multi-mini-processor)

Then Came Cm*.

- Conceived, architecturally specified and built at CMU
- Extensively studied with performance and reliability models *during design*
- Reliability intrinsically designed in
- Grew into a 50-processor system (1977) starting from a 10-processor system (1975)
- Even had two independent operating systems

C.vmp (Computer voted multi-processor)

- **Employed off-the-shelf components with little or no modification in order to survive transient and hard faults (1976)**
- **Independent mode and a voting mode; bus-level voter would allow**
  - Unreplicated devices (e.g., console terminal) to broadcast results to all three processors
  - System to divide itself into three independent computers communicating through interfaces
- **Trading off performance for reliability**
  - System could switch dynamically between independent and voting modes
- **Lessons learned from this research**
  - Six times more reliable for transient faults than Cm*.
  - Voter reduced system performance by about 15%
  - Could be used as a "transient-fault meter" by adding statistics board to compare the three buses for (and to record) disagreements

Redundancy in Storage Systems

- **Continuously running systems demand both availability and performance from their storage sub-systems**
- **Redundant disk-arrays: Grouping together a number of smaller disks (rather than using one large disk-drive)**
  - Better performance, but high component-count implies higher failure rates
- **Redundancy approaches explored: Replication and encoding**
  - Error-correcting codes are good for data reliability, but perform poorly in the presence of a disk failure
- **CMU research on ECC-based redundant disk-arrays**
  - Better performance in the presence of disk failures
  - Without significantly affecting performance, cost or reliability

Hard and Transient Faults

- **Data collection from Cm*. to answer questions about hard failures**
  - For each module type, data collected on number of different types of that module, chip count, total hours of utilization and total number of failures
  - Data found to follow exponential distribution, with (MIL Handbook 217) failure-rate taking into account the time rate of change of technology
- **Transient faults**
  - Much harder – by the time fault manifested, traces of nature/location gone
  - Data collection and extensive event-logging of transient faults
    - Four time-sharing systems, an experimental multiprocessor, and an experimental fault-tolerant system
    - Ranged from microprocessors to mainframes
- **Lessons learned**
  - Transient faults were ~20 times more prevalent than hard failures
  - Transient-fault manifestations differed from those of permanent faults

Understanding Error-Logs

- **Born out of a diagnosis and maintenance plan for VAX clusters**
  - Increased number of user-mode diagnostics
  - Online analysis of system error-logs to discover trends and advise the system prior to catastrophic failure
- **CMU research on understanding system event error-logs**
  - Inter-arrival times of errors – probability of crashes decreased with time
    - Weibull function with decreasing failure-rate
  - Modeling relationship between system load and system error-rate
- **Led to trend analysis research**
  - Based on the observation that a hardware module exhibits a period of (potentially) increasing unreliability before final failure

Trend Analysis

- **Developed a model of normal system behavior, and watched for a shift that signifies abnormal behavior**
- **Based on the observation that data from normal system workloads are better suited for pointing out failure mechanisms than specification-based diagnostics are**
  - Normal system workloads tend to stress systems in ways different from specification-based diagnostic programs
- **By discovering normal behavior and trends, it was possible to predict certain hard failures (and even discern hardware/software design-errors) prior to the occurrence of catastrophic failure**
- **Tupling (data-grouping or clustering)**
  - Clusters/groups of event-log entries exhibiting temporal or spatial patterns
  - Single-error events can propagate to cause multiple entries in an event-log

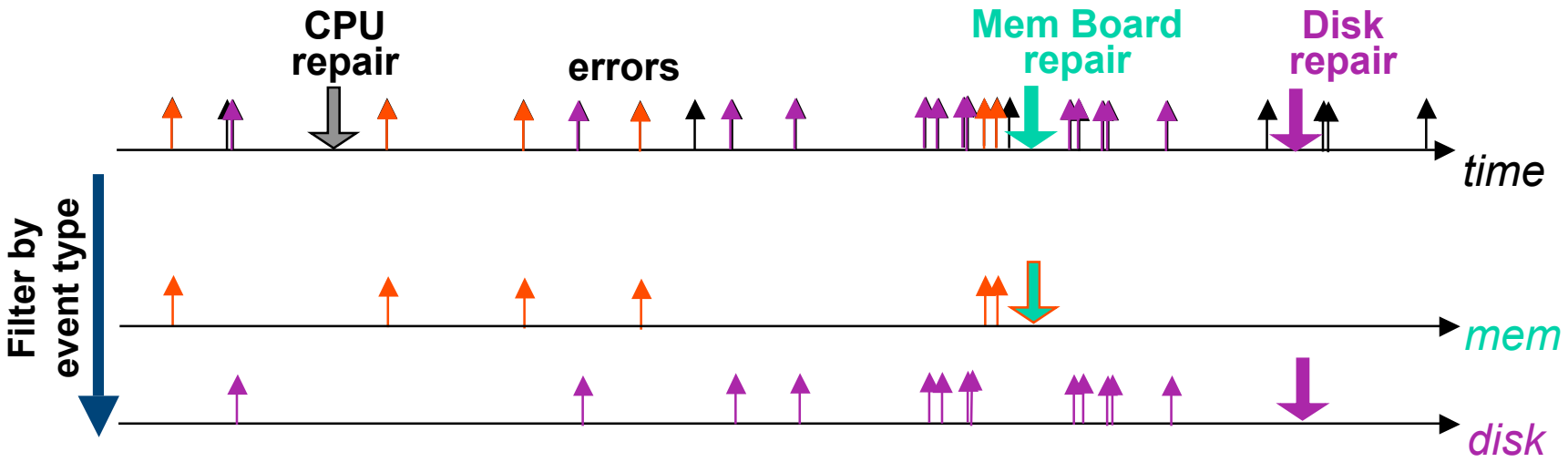
Automated Monitoring and Diagnosis

- **Requires three basic roles/components in a system**
- **Sensors for gathering data**
  - Sensors must be provided to detect, store, and forward performance and error information (*e.g.*, event-log data) to a diagnostic server whose task it is to interpret the information
- **Analyzers for interpreting data**
  - Exercised once the system performance and error data have been accumulated
  - Interpretation done by expert problem-solving modules in the diagnostic server
  - Diagnostic server should have access to profiles of normal system behavior as well as hypotheses about behavior exceptions
- **Effectors for confirming interpretation**
  - Post-analysis, a hypothesis must be confirmed/denied before issuing warnings
  - Effectors stimulated the hypothesized condition in the system.
  - Often exercisers that are downloaded to the suspected portion of the system, and run under special conditions to confirm the fault hypothesis or to narrow its range

Dispersion Frame Technique (DFT)

- Observed periods of increasingly unreliable behavior prior to catastrophic failure

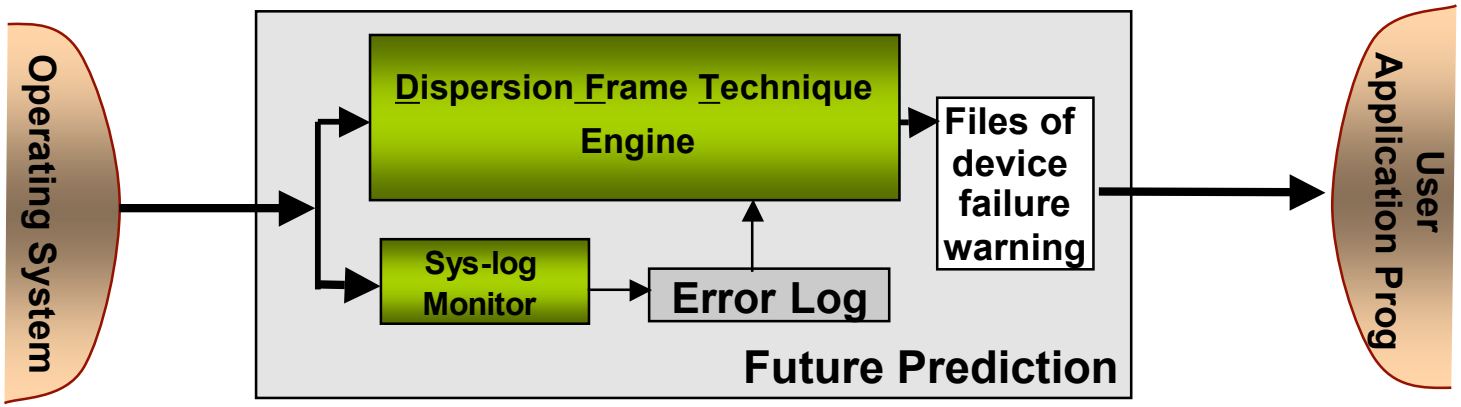
Error entry example: DISK:9/180445/563692570/829000:errmsg:xylg:syc:cmd6:reset failed (drive not ready) blk 0
type *time*



- Based on this observation, the DFT Heuristic was derived, to catch the non-monotonical decrease in error interarrival time (1970's)
  - Simple set of rules that capture various sorts of failure-precursor patterns

DMOD: DFT Engine Implementation

- This module generates device failure warning information (1990's)
  - Sys-log Monitor: Monitors new entries by checking the system event log periodically
  - DFT Engine: Applies DFT heuristic and issues corresponding device-failure warning if the rule(s) is satisfied.



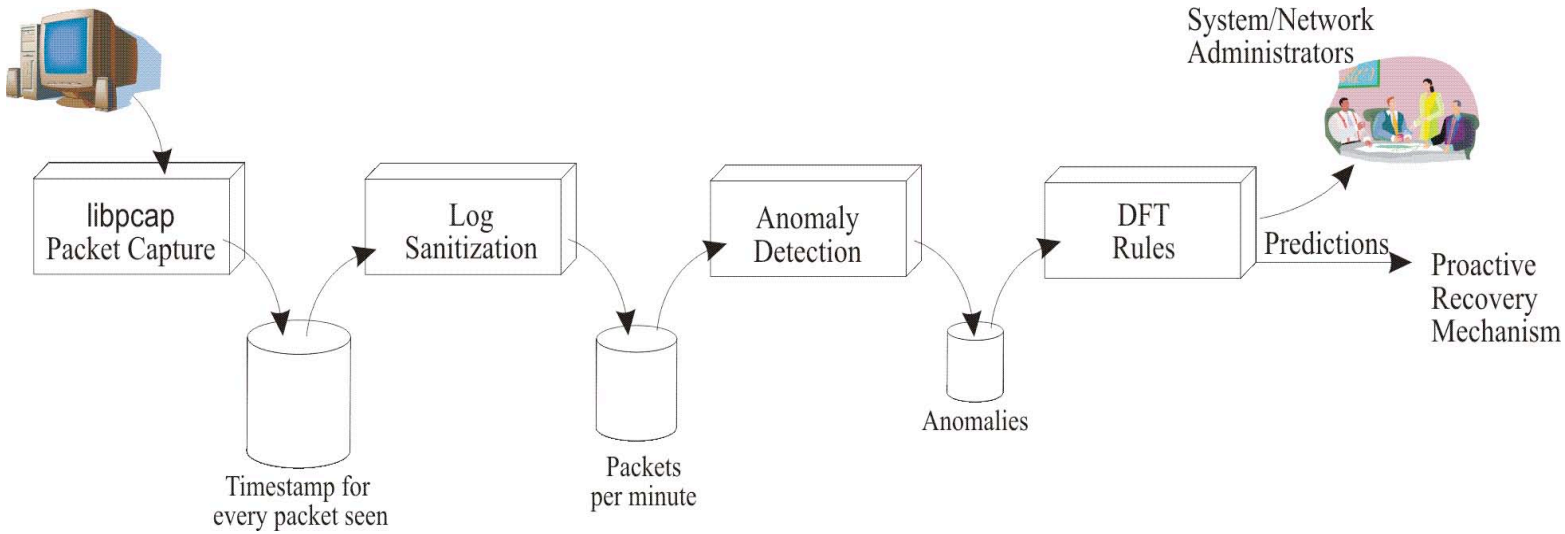
- Successfully used in
  - Cluster of Andrew File Servers at CMU
  - Network-fault prediction to detect anomalous behavior

Network Anomaly-Detection

- **Networks have "soft failures", i.e., temporary loss of bandwidth**
  - Often perceived by users as degraded or anomalous performance
- **Active, online monitoring of the CMU campus Andrew network**
  - Eight network routers, as well as the Computer Science Department's entire Ethernet network, for traffic and diagnostic information
  - Traffic parameters: Transmitted and received packets, network load, and network collisions
  - Diagnostic parameters:CRC errors, packet-alignment errors, router-resource errors due to buffer limitations, router-overrun errors due to throughput limitations
- **Fault feature vector to describe fault-specific anomalous conditions**
  - Effective in detecting network failures over the two-year study
  - Effective in abstracting large amounts of network data (32M points) to only a few events (~200 event-matches) that warranted attention

Fault Prediction Using Network Behavior

- **Detect network anomalies through metrics, e.g., packets/minute**
  - Template of normal network behavior over specific time-period (e.g., day)
  - New patterns of behavior "folded in" carefully to adapt the template
- **Observation: Some impending application-level faults can be signaled through pre-fault patterns of anomalous network behavior**
- **Anomaly-detection + DFT rules = Prediction of some faults (2004)**



Reliability Analysis and Evaluation

- **Two fault-tolerant multiprocessors, FTMP and SIFT, were developed and delivered to the Air-Lab facility at the NASA Langley Research Center**
- **Starting in 1981, CMU performed a series of experiments to validate the fault-free and faulty performance of FTMP and SIFT**
  - Methodology derived from CMU's earlier work on Cm*.
  - Synthetic workload generator (SWG) allowed experimental parameters to vary at run-time
  - SWG drastically reduced the turnaround time for experimentation by eliminating the edit/compile/downlink-load portion of the experimental cycle
  - Avionic workload was developed, and the results of the baseline experiments were reproduced through appropriate settings of the SWG's runtime parameters
  - SWG was modified to include the injection of software faults
- **Methodology has been used to assist the Federal Aviation Administration in the design of the next-generation air traffic control system**

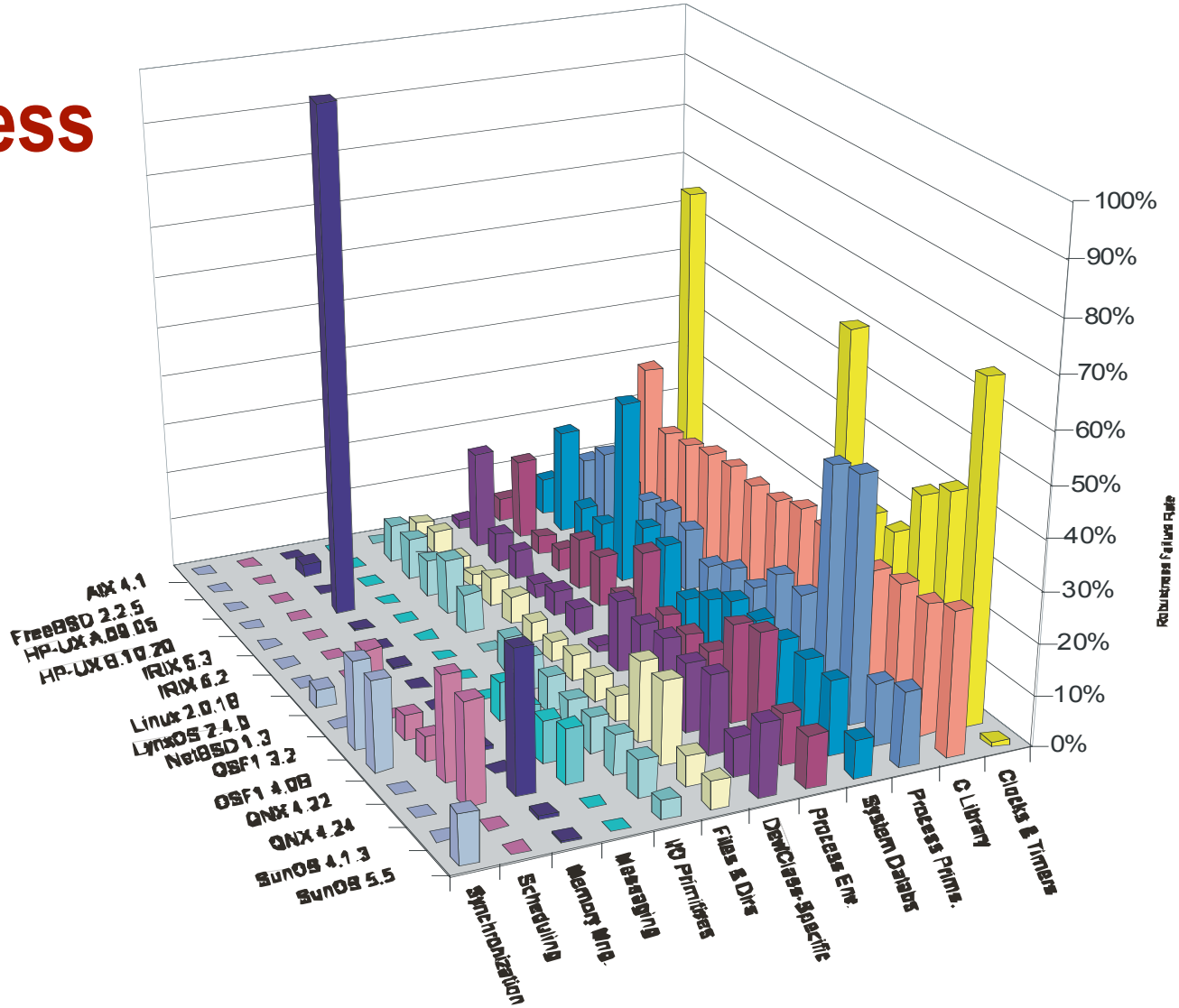
Raising Fault-Model Abstraction

- **CMU research started with exploring effects of gate-level faults on system operation as a basis for fault-models at the program level (1970's)**
  - Simulation models with capabilities for fault injection
  - Workload dependencies modeled and variety of workloads executed
  - Prediction model for fault manifestation based on instruction execution
  - Impact: Reduction of fault space required during fault-injection studies (SWIFI)
- **Next higher-level abstraction: RTL (register-transfer-logic) (ASPHALT) (1980's)**
- **Pipelined functional test program modeling**
- **Device-level modeling (x-diagnosis)**
- **Human/user error modeling (1990's)**
- **Results and lessons learned**
  - Model-based diagnosis, which had been successfully used for gate-level circuits - could be scaled up to system-level circuits
  - Automation of test program execution - a methodology to test circuits in which diagnosis was then done manually - was feasible

Robustness Testing

- Robustness testing of COTS applications (which are usually employed for cost savings) must be cost-effective
- Source code might not always be available
- Ballista: Simple, repeatable way to directly measure software robustness without requiring source code or behavioral specifications
- Each of fifteen different operating systems' respective robustness was measured by automatically testing up to 233 POSIX functions and system calls with exceptional parameter values
- Allowed benchmarking of robustness

Ballista: OS Robustness Evaluation



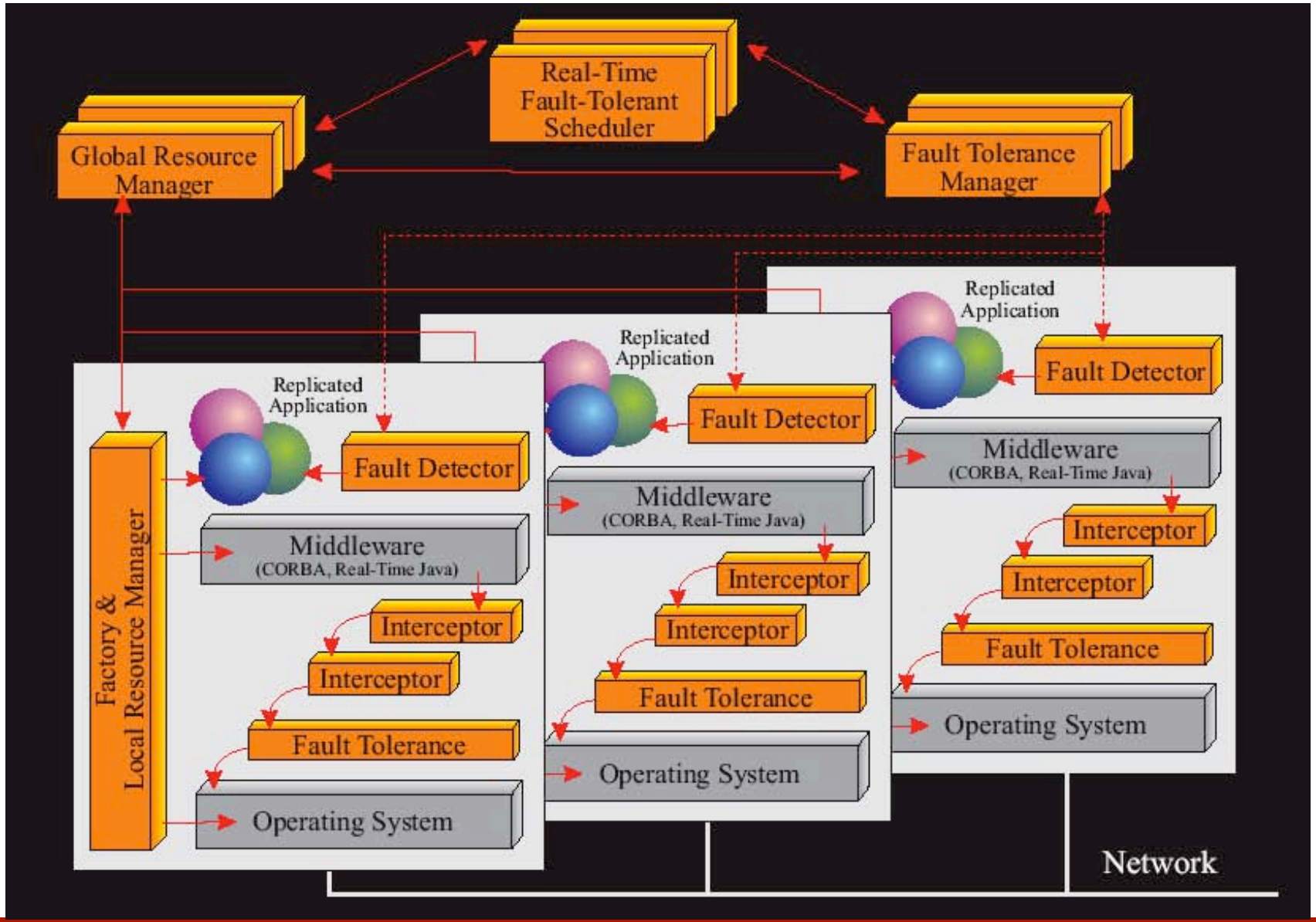
BALLISTA

http://www.ballista.org

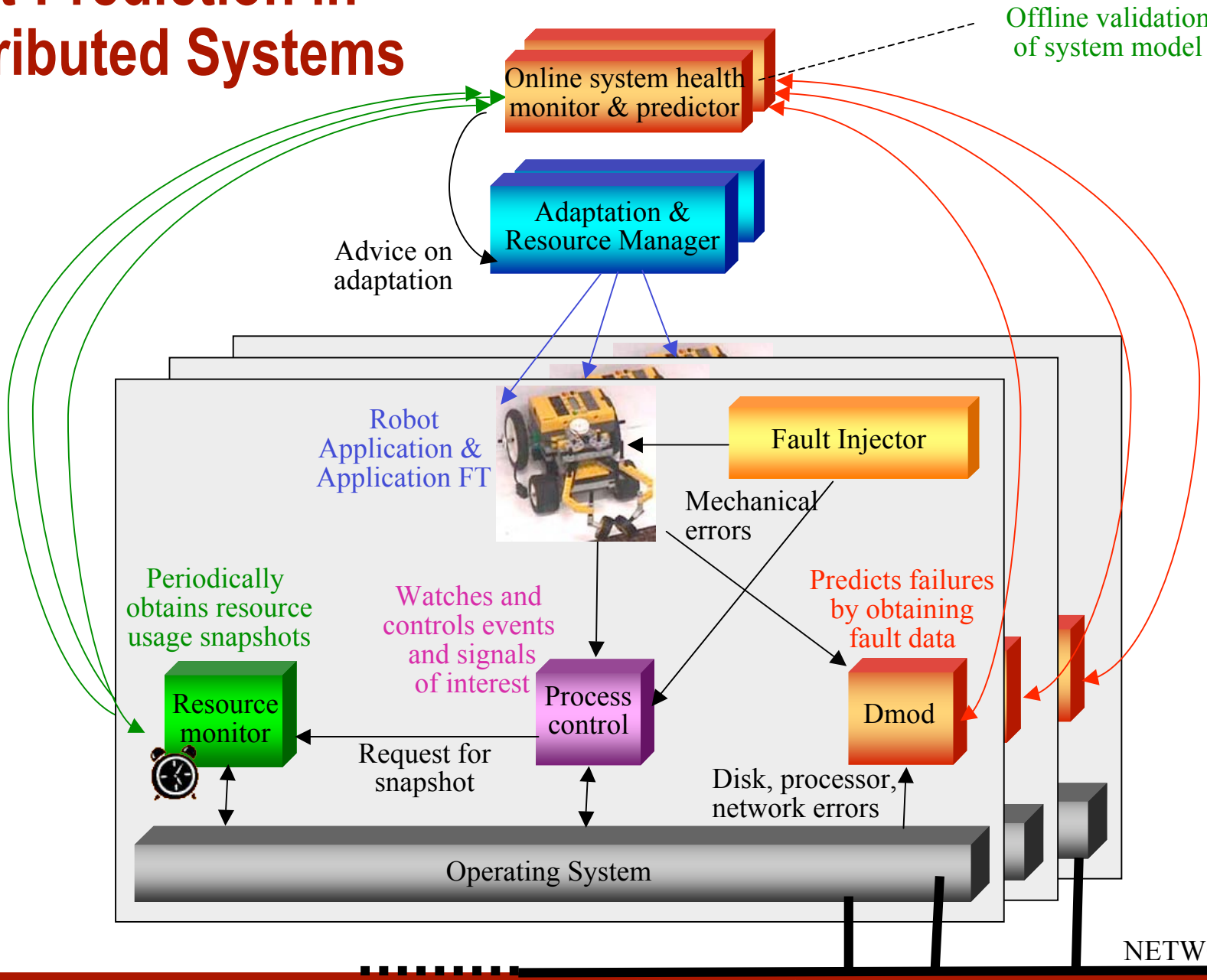
And What About the Next Decade?

- Increased focus on *distributed* systems, rather than single-processor computer systems
- Increased focus on *proactive fault-tolerance,* rather than the classical reactive fault-tolerance
- Increased focus on machine learning with adaptation and reconfiguration techniques for fault-tolerance, rather than static configurations
- Increased focus on achieving *other properties* along with fault-tolerance
- Expanding the fault model to cover
  - Resource-exhaustion, propagating, interacting, system-wide faults

MEAD: Real-time Fault-tolerant Middleware

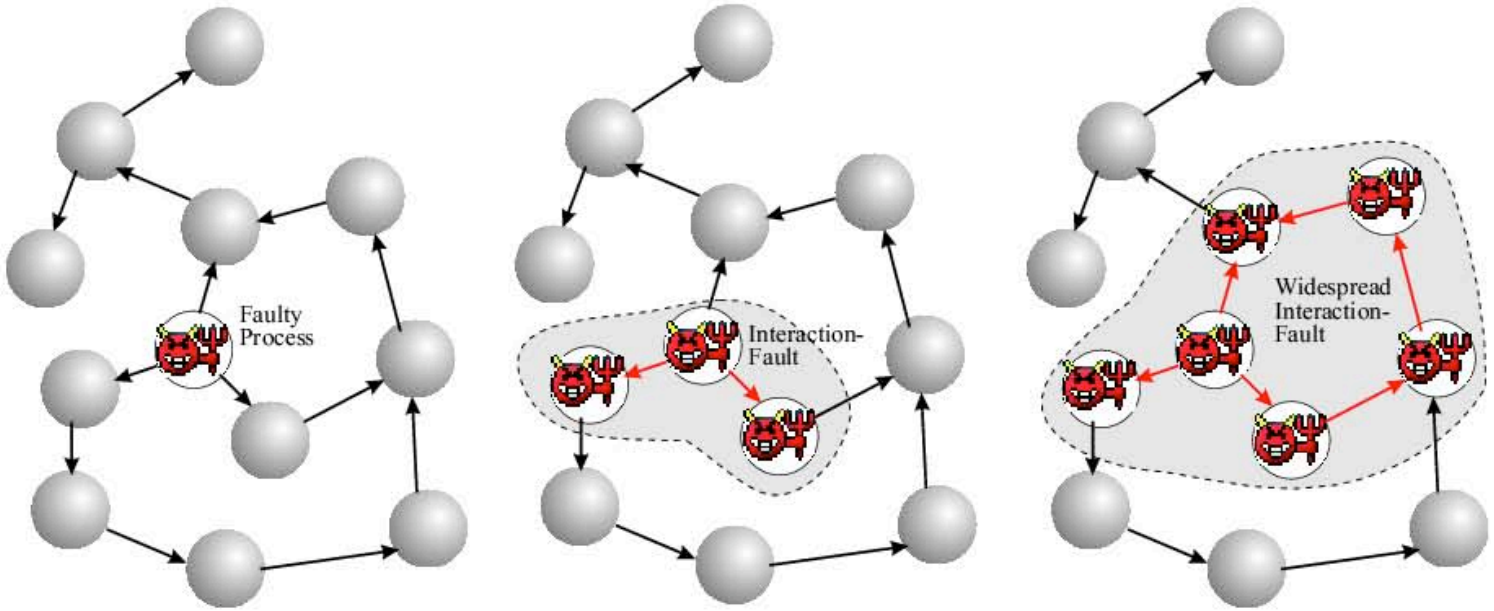


Fault-Prediction in Distributed Systems



Interaction Faults

- **Onset of a fault somewhere in the distributed system, followed by**
  - Propagation of the fault through interactions and dependencies, until the entire system (or significant parts of the system) collapses
    - Live upgrades, network partitions, unchecked exception-handling, virus attacks
- **Developing models to detect & handle interaction-faults**
  - Need to discover, analyze, predict and check fault-spread



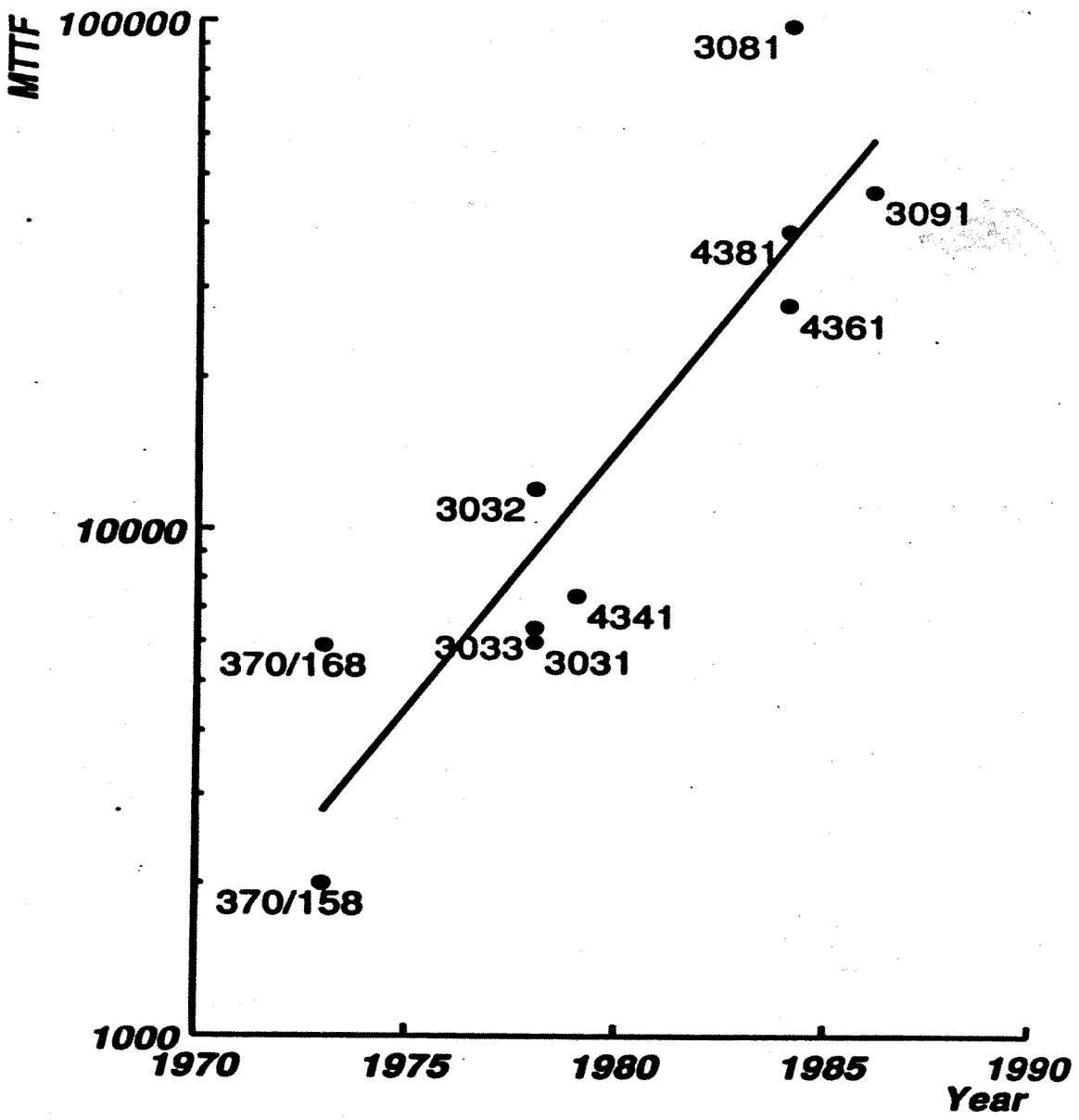
Summary

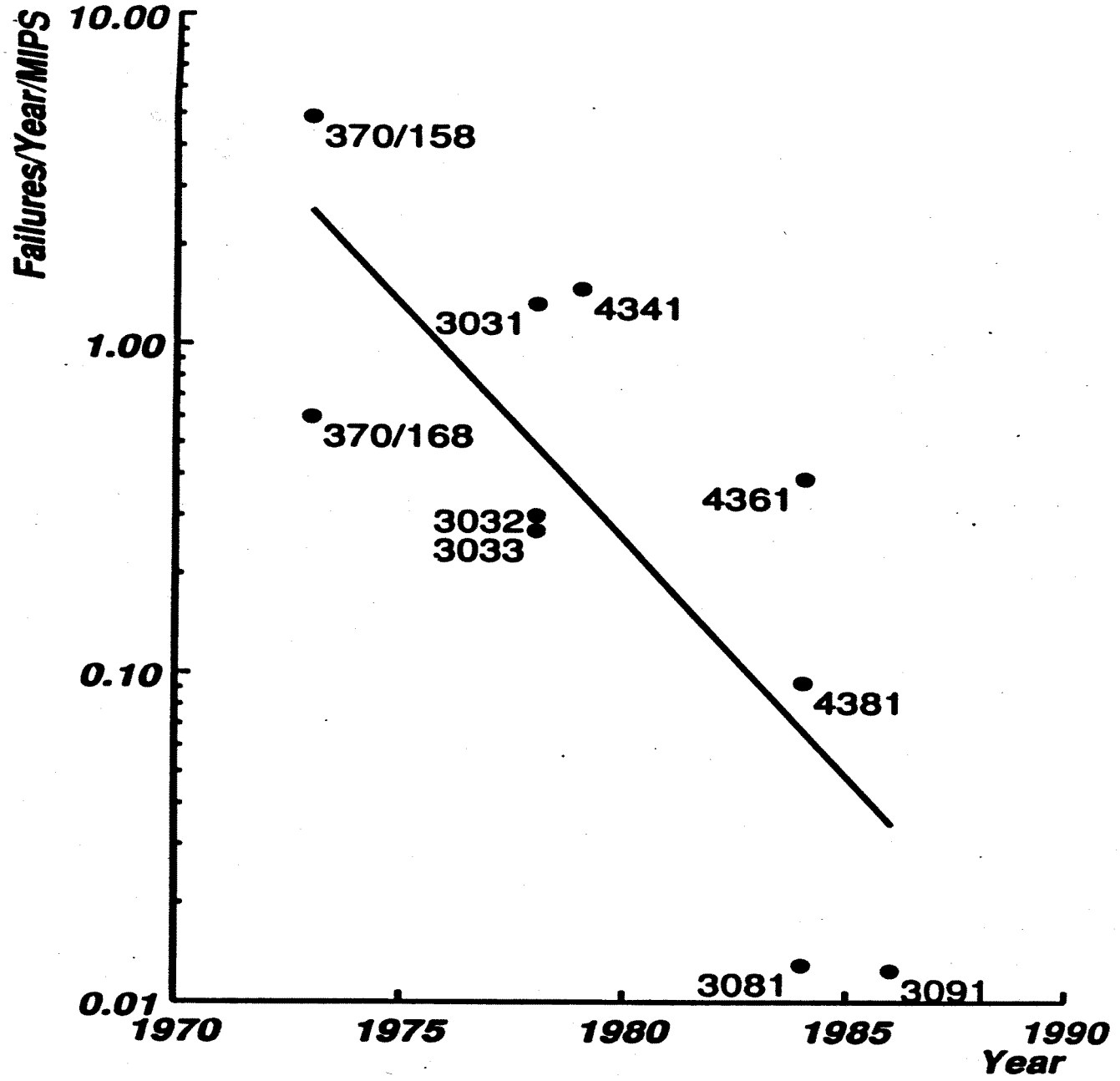
- **CMU research in dependability has spanned three decades, multiple researchers, and is continuing on to this decade**
- **Marked by emphasis on experimentation, empirical evidence and data-collection to substantiate results**
- **Range of research**
  - Microprocessors, robustness testing, distributed systems, elevating fault-model abstractions, trend analysis, anomaly detection
- **Ongoing work**
  - Distributed fault-tolerance
  - Proactive (rather than reactive) fault-tolerance
  - Fault model including interaction and propagating faults

For More Information

Priya Narasimhan
Assistant Professor of ECE and CS
Carnegie Mellon University
Pittsburgh, PA 15213-3890
Tel: +1-412-268-8801
priya@cs.cmu.edu

Extra Slides


MTTF
100000
10000
1000
1970
1975
1980
1985
1990
Year
3081
3091
4381
4361
3032
4341
3033
3031
370/168
370/158


Failures/Year/MIPS
10.00
1.00
0.10
0.01
1970
1975
1980
1985
1990
Year
370/158
3031
4341
370/168
4361
3032
3033
4381
3081
3091

Stages in the development of a system

STAGE	ERROR SOURCES	ERROR DETECTION
Specification & design	Algorithm Design Formal Specification	Simulation Consistency checks
Prototype	Algorithm design Wiring & assembly Timing Component Failure	Stimulus/response Testing
Manufacture	Wiring & assembly Component failure	System testing Diagnostics
Installation	Assembly Component failure	System Testing Diagnostics
Field Operation	Component failure Operator errors Environmental factors	Diagnostics

Sources of Errors

Probability of operational outage from various sources.

	AT&T Switching Systems	Bellcore Commercial	Japanese Commercial Users	Tandem 1985	Tandem 1987	Northern Telecom	Mainframe Users
Hardware	0.20	0.26	0.75*	0.18	0.19	0.19	0.45
Software	0.15	0.30	0.75*	0.26	0.43	0.19	0.20
Procedural error	-	-	-	-	-	0.333	-
Maintenance	-	-	0.75*	0.25	0.13	-	0.05
Operations	0.65	0.44	0.11	0.17	0.13	0.33	0.15
Environment	-	-	0.13	0.14	0.12	0.28	0.15
Power	-	-	-	-	-	0.125	-

* The sum of these three sources was reported as 0.75.

CMU Andrew File Server Study

- **13 SUN II workstations**
  - 68010 processor
  - 4 Fujitsu Eagles

Some Interesting Numbers

- **Permanent outages / total crashes = 0.1**
- **Intermittent faults / permanent failures = 0.1**
  - Thus first symptom appears over 1200 hours prior to repair
- **(Crashes – permanent) / Total faults = 0.255**
- **14/29 failures had three or fewer error log entries**
  - 8/29 had no error log entries

One Set of Experimental Observations

- 21 workstation years' worth of data from CMU Andrew file servers
- Category and number of failures
  - Permanent failures: 29
  - Intermittent faults: 610
  - Transient faults: 446
  - System crashes: 298
- Mean time to specific category of fault
  - Permanent: 6552 hours
  - Intermittent: 58 hours
  - Transient: 354 hours
  - Crash: 689 hours