Software Fault Tolerance via Environmental Diversity

DCCN 2021

September 20, 2021

Prof. Kishor Trivedi
Department of Electrical and Computer Engineering
Duke University, Durham, NC, USA
ktrivedi@duke.edu

www.researchgate.net/project/Software-Fault-Tolerance-via-Environmental-Diversity

www.researchgate.net/project/Date-race-software-failure-MTTF-prediction

Bottom Line Up Front -- BLUF

- Today's complex systems (including SDN, CPS, smartgrid, aircraft, spacecraft, weapon systems, and IoT) contain a huge amount of software → **Software is a major cause of system undependability**

- **Software failures during operation are a fact that we need to learn to deal with**. Traditional method of software fault tolerance, based on design diversity, is expensive and hence does not get used extensively. Software fault tolerance based on ***inexpensive environmental diversity*** **could be exploited**

- The focus in the Software Engineering community so far has been on **software faults**; we need to pay equal attention to **failures** caused by software bugs (faults) and the recovery from these failures during the operational phase

- Focus in the Software Reliability Engineering community so far has been on software reliability; we need to pay attention to **software availability** as well

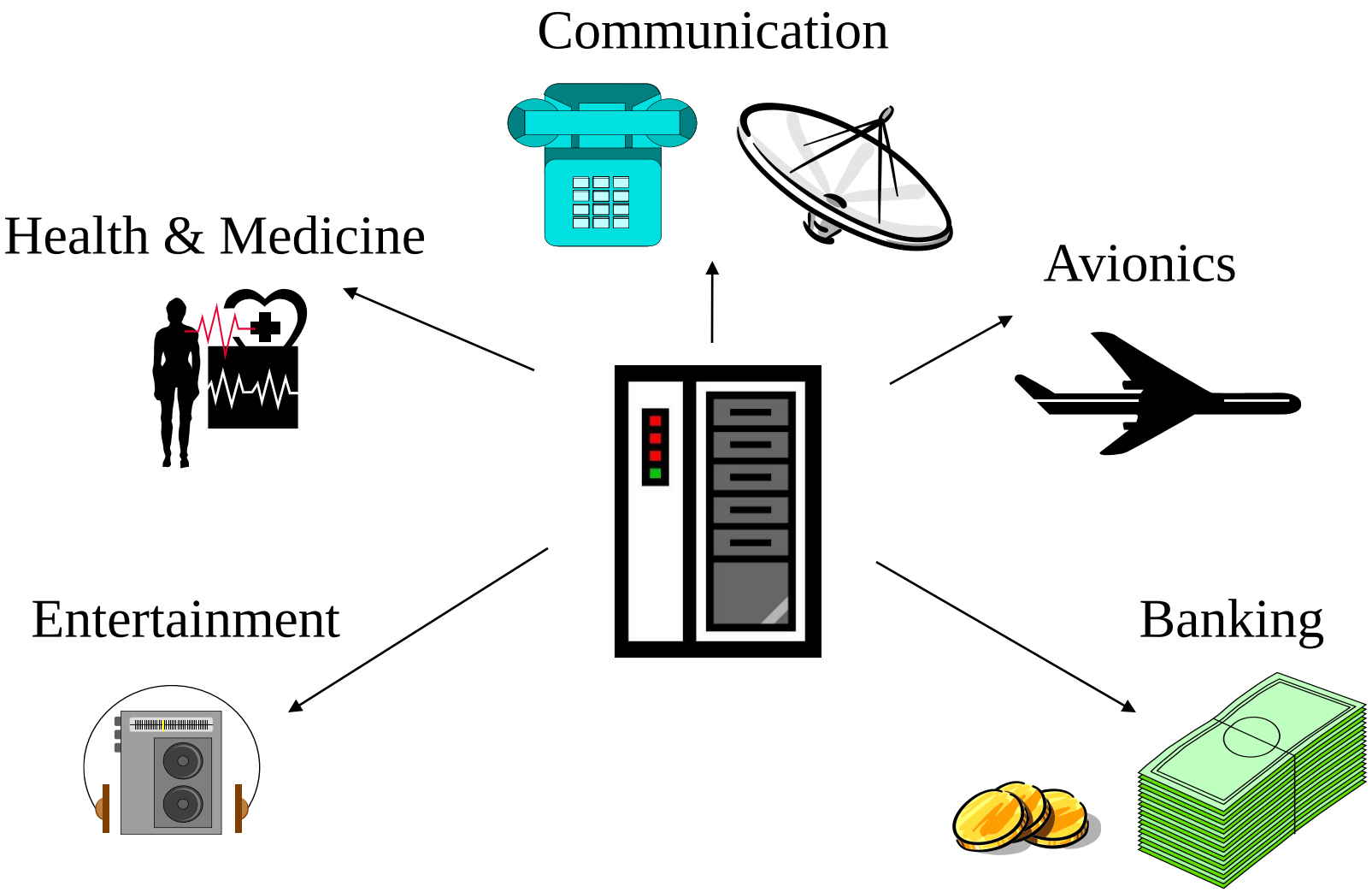
Duke UNIVERSITY

Outline

- Motivation/Definitions
- Real System Examples
- Software Fault Classification
- Environmental Diversity
- Methods of Mitigation
- Conclusions

Duke UNIVERSITY

Our Dependence on Technical Systems → These systems need to be highly reliable



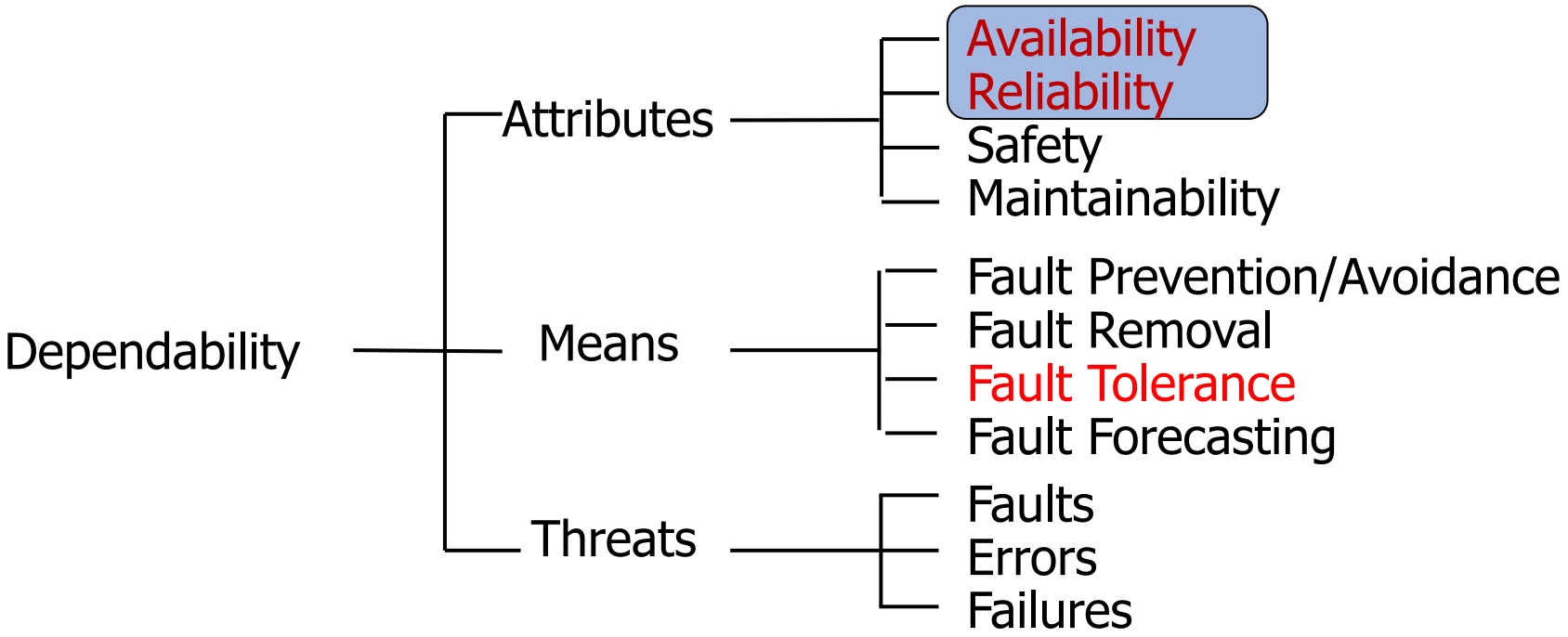
Need for a new term

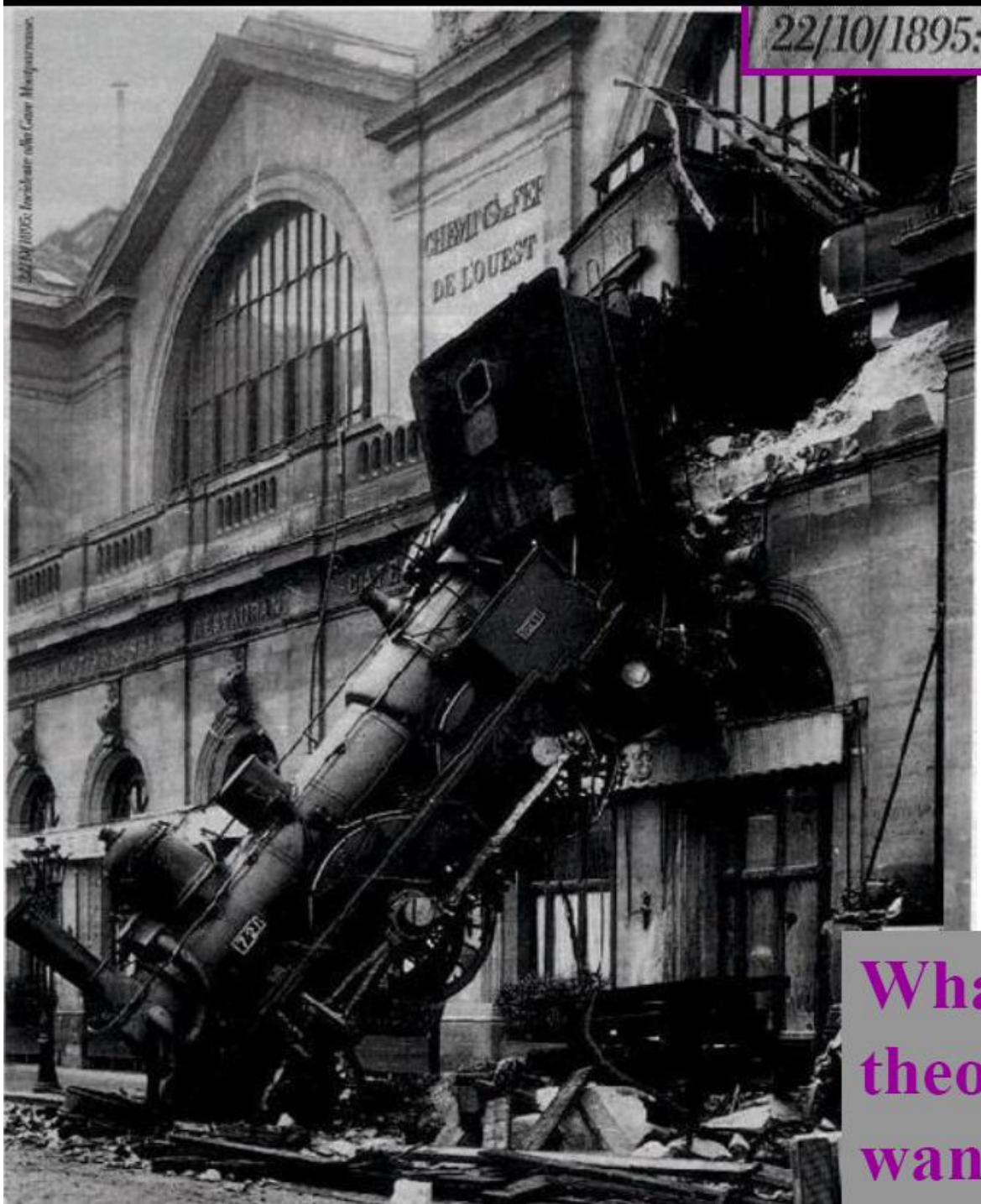
- *Reliability* is often used in a generic sense as an umbrella term.
- *Reliability* is also used as a precisely defined mathematical function.
- To remove the confusion, IFIP WG 10.4 proposed *Dependability* as an umbrella term and *Reliability* is then to be used as a well-defined mathematical function.

Duke UNIVERSITY

Dependability – an Umbrella Term

Dependability: Trustworthiness of a system such that reliance can justifiably be placed on the service it delivers





22/10/1895: Incidente alla Gare Montparnasse.

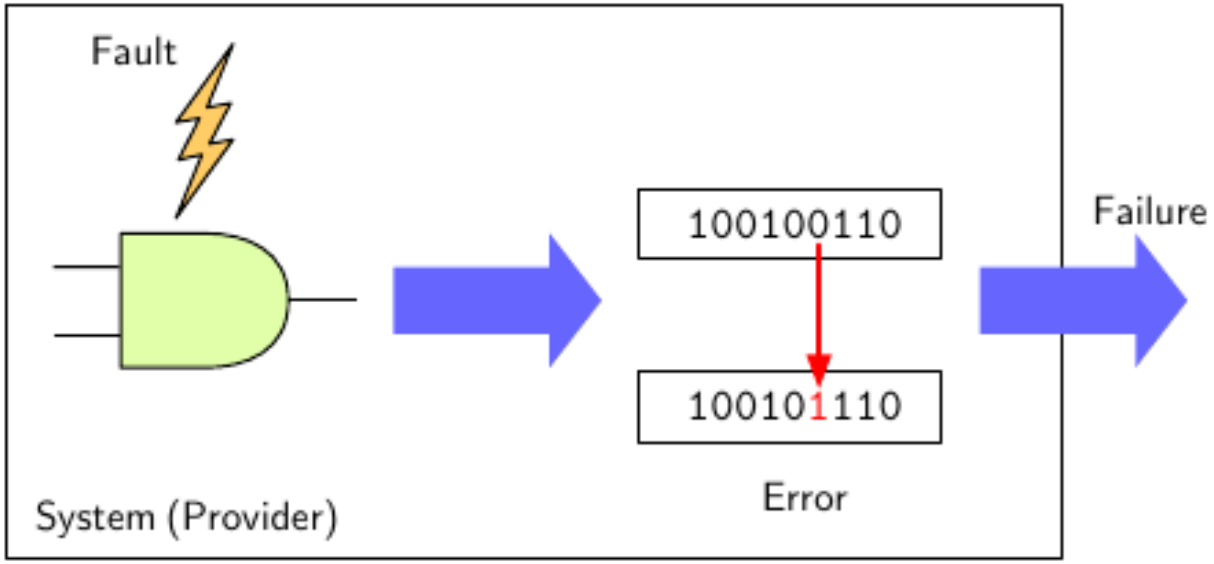
What dependability theory and practice wants to avoid

IFIP Working Group 10.4

- **Failure** occurs when the delivered service no longer complies with the desired output.
- **Error** is that part of the system state which is liable to lead to subsequent failure.
- **Fault** (or **bug**) is adjudged or hypothesized cause of an error.

Faults are the cause of **errors** that may lead to **failures**

Duke UNIVERSITY



- **Fault:** adjudged or hypothesized cause of an error;
- **Error:** part of the system state which is liable to lead to a failure
- **Failure:** deviation of the delivered service from compliance with the specification (e.g., the service is unavailable or it provides a wrong answer).

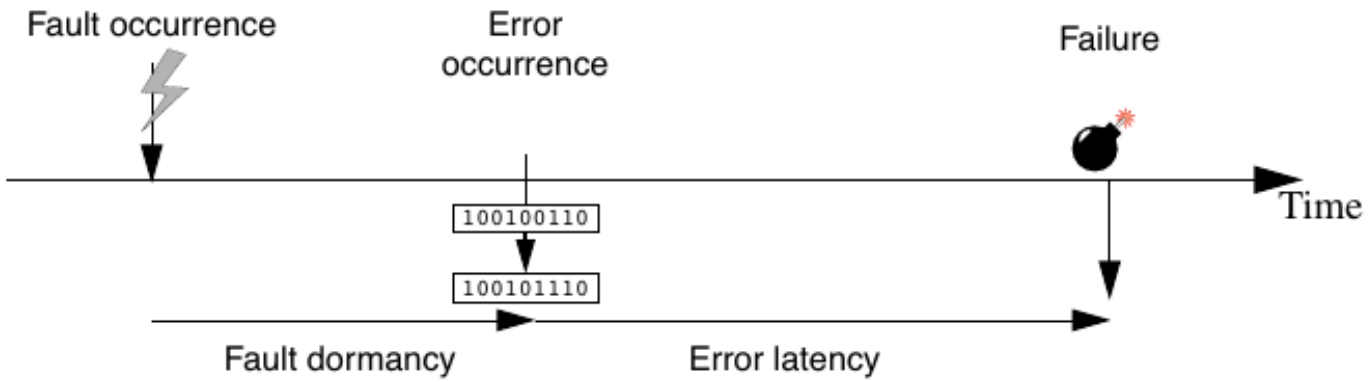


Figure: Fault dormancy and error latency

- For software, the fault dormancy and error latency may be long
- Can be order of years for dormancy but days and weeks for latency
  - Therac-25 introduced in 1982, first error occurred in 1986; thence failure occurred in few seconds
  - https://web.stanford.edu/class/cs240/old/sp2014/readings/therac-25.pdf
- Errors with long latency may be removed by software rejuvenation
  - https://www.gao.gov/products/IMTEC-92-26

Fault Classification

- Physical vs. Design/Manufacturing/Integration vs. Interaction
- Malicious vs. non-malicious
- Node vs. Link
- Hardware vs. Software vs. Human
- Hardware:
  - Permanent, Intermittent, Transient
- Software
  - Bohrbugs, Mandelbugs, Concurrency bugs, Heisenbugs, Aging-related bugs

Failure Classification

- Omission failures (don't get a response)
  - Crash failures
  - Infinite loop
- Value failures (get a response but wrong value)
- Timing failures
  - Early
  - Late (performance or dynamic failures)

Failure Effects

- A failure is classified with respect to the consequences it has for the end-user or end-user application. This is called the failure effect.
- In safety critical system, failures are categorized as:
  - Benign failures vs. catastrophic failures
  - Safe vs. Unsafe failure
- In the context of security, they are classified as:
  - Breach of confidentiality vs. breach of integrity vs. loss of use

Can we trust computers?

Just to mention (very) few cases …

- **2020: Heathrow disruption**
- **2019: Facebook, whatsapp, Instagram outage**
- **2016: Yahoo data breach (credential leaks)**
- **2015: HSBC Payment glitch**
- **2004 Mercedes-Benz - “Sensotronic” braking system**
- **2000 National Cancer Institute, Panama City**

Duke UNIVERSITY

Example Failures from High Tech companies

Feb. 2017 Amazon S3 service outage (almost 6 hours)

Mar. 2015 , Gmail was down for 4 hours and 40 min.

Mar. 2015, Down for 3 hours affecting Europe and US

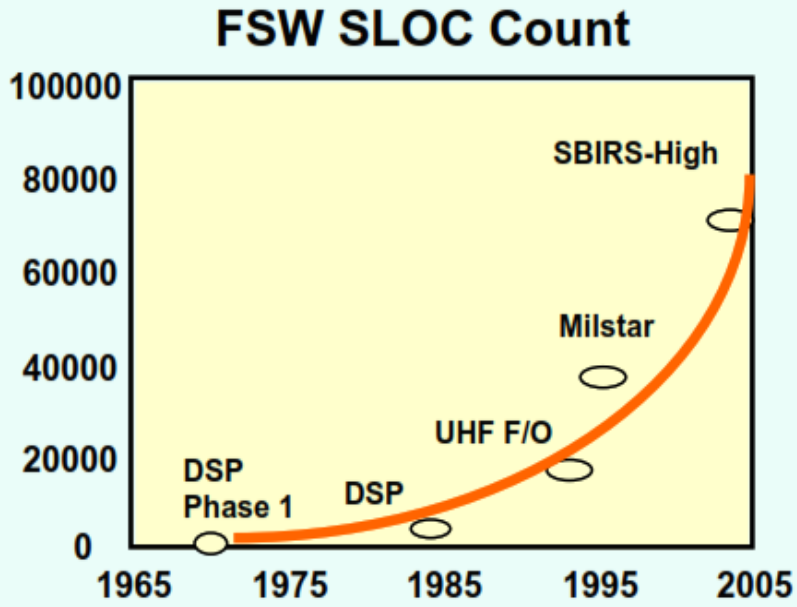
Dec. 2015, Microsoft Office 365 and Azure down for 2 hours

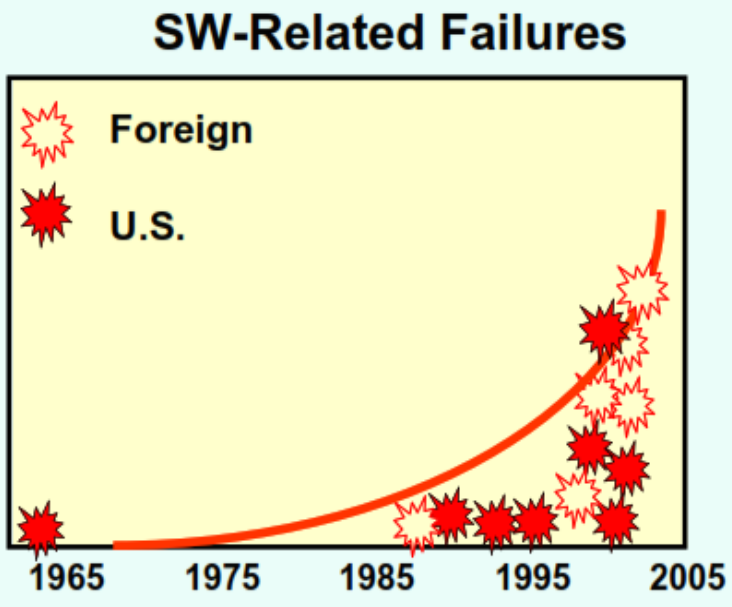
Sept. 2015, AWS DynamoDB down for 4 hours impacting among others Netflix, AirBnB, Tinder

Mar. 2015, Apple ITunes, App Stores long outage: 12 hours

These examples indicate that even the most advanced tech companies are not offering high levels of dependability

Software Increasingly Matters





Over half of failures between 1998 and 2000 involved software

FSW SLOC = Flight Software Source Lines of Codes

1962	Mariner I (Atlas)	1988	Phobos 1
1990	Intelsat 6 (Titan CT2)	1996	Cluster (Ariane 501)
1991	Orbcomm X	1996	SOHO
1994	Clementine	2000	STRV x 2
1999	Milstar 2-1	2000	ICO F1 (Sea Launch)
1999	MCO	2000	QuickBird (Cosmos 3M)
1999	Terriers		
1999	MPL		

NEAR and Phobos 2 not counted

1

THE AEROSPACE CORPORATION

4 March 2003, GSAW Presentation from Paul Cheng, Corporate Risk Assessment & Management Subdivision

Duke UNIVERSITY

Software is a big problem

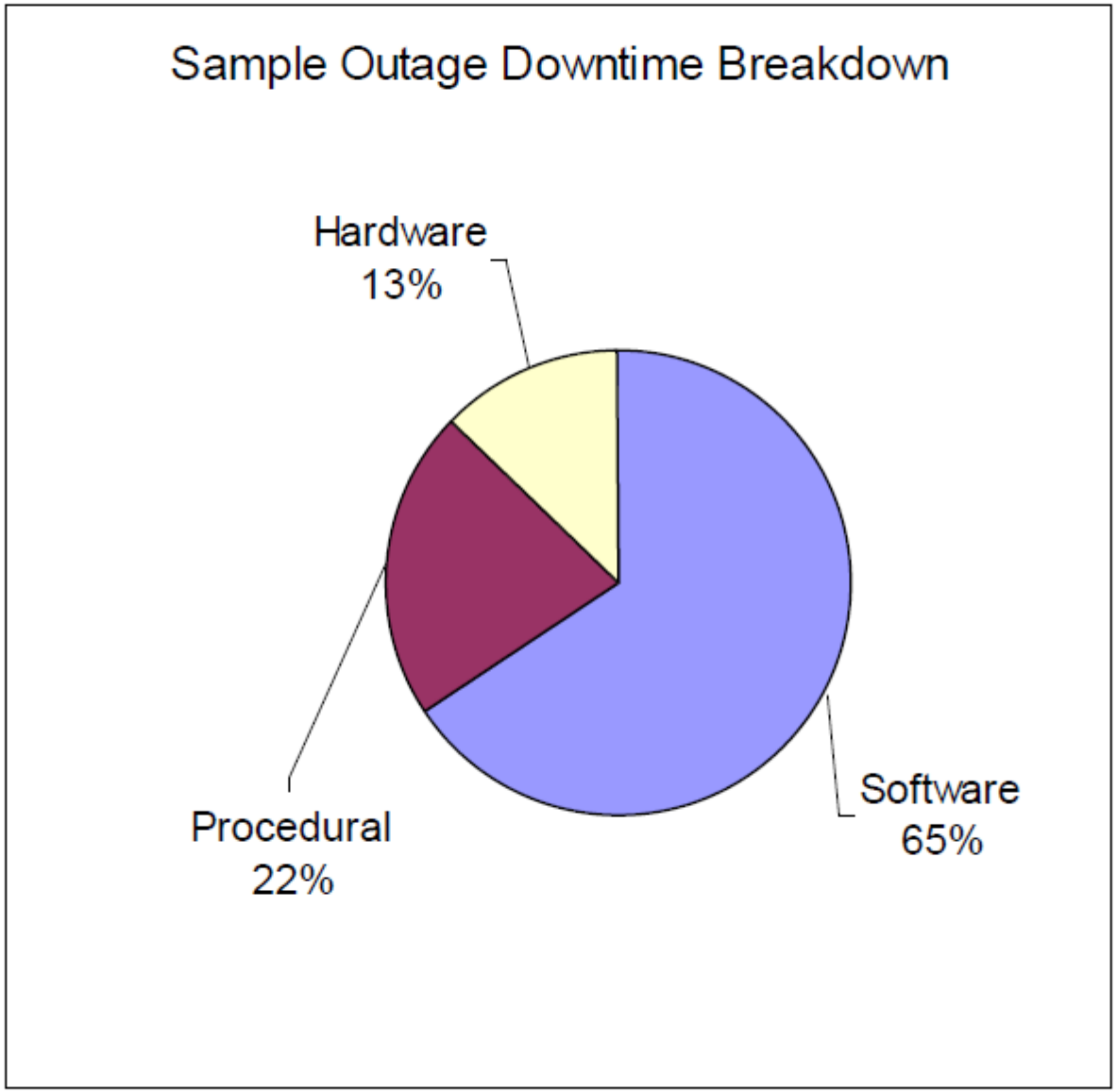
- Hardware fault tolerance, fault management, redundancy management, reliability/availability modeling relatively well developed
- System outages more due to software faults

Key Challenge:

Software reliability is one of the weakest links in system reliability/availability

Duke UNIVERSITY

Impact of Software Field Problems



Failure/downtime due to software bugs

amazon.com	Oct. 2012	Amazon Webservices – 6 hours (Memory leak) Amazon EC2 – 2 hours
Google	Sept. 2011	Google Docs service outage – 1 hour (Memory leak due to a software update)

Failure/downtime due to software bugs

Google	Jul. 2017	Google Cloud Storage service outage (3 hours and 14 min.) - API low-level software bug
Microsoft	Jul. 2017	Jul. 2017 - Microsoft Azure service outage (4 hours) – Load Balancer Software bug

These examples indicate that even the most advanced tech companies are not offering highly reliable software

More Recent Examples

- One Fastly customer triggered internet meltdown – June 9, 2021
  - www.bbc.com/news/technology-57413224
- In Commercial aircrafts (Boeing 737 Max software problem)
  - Ethiopian Airlines Flight, March 2019,
    149 people died
  - Lion Air Flight crash, Oct. 2018,
    189 people died

Failures & Downtime Lead to

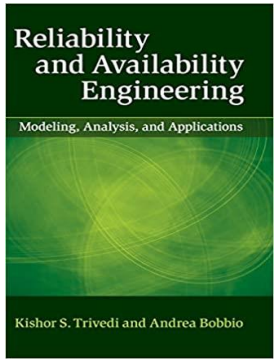
- Loss of Reputation
- Loss of Revenue
- Possible Loss of Mission
- Possible Loss of Life

Need Methods

- That reduce system failures and reduce downtime due to these failures (contributed by hardware, software and humans)

- For System Reliability/Availability assessment and bottleneck detection to help decide the most cost-effective path to improvement of reliability/availability

Ref: Trivedi & Bobbio, Reliability and Availability: Modeling, Analysis, Applications, Cambridge University Press, 2017



Methods to Improve Software Reliability/Availability

- Fault Avoidance or Fault Prevention
- Fault Removal by testing/debugging
- Fault Tolerance or Use of Redundancy
- Fault/Failure Forecasting

Software Reliability: Means

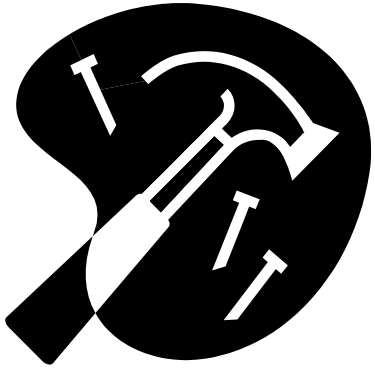
- **Fault prevention or Fault avoidance**
  - Good software engineering practices
  - Good software architecture
  - Use of formal methods
    - UML, SysML, BPML
    - Proof of correctness
    - Model Checking (NuSMV, SMART, SPIN, PRISM)
- Bug free code not yet possible for large scale software systems
- **Yet there is a strong need for failure-free system operation**

Duke UNIVERSITY

Fault removal by testing and debugging

Fault removal

can be carried out during:



- the specification and design phase
- the development phase
- the operational phase
- Test software for as long as possible
  - Use automated testing tools – TestComplete, TestProject
  - Use coverage testing tools – CREST, EvoSuite
  - Use combinatorial testing tools – ACTS, ComTest

Duke UNIVERSITY

Reliable Software

- **Fault removal**
  - Can be carried out during
    - ✓the specification and design phase
    - ✓the development phase
    - ✓the operational phase
  - Failure data may be collected and used to parameterize a software reliability growth model(SRGM) to predict when to stop testing

- Impossible to fully test and verify if software is fault-free

  "Testing shows the presence, not the absence, of bugs" - E. W. Dijkstra

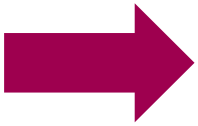
- Software is still delivered with many bugs either because of inadequate budget for testing, very difficult to reproduce/detect/ localize/correct bugs or inadequacy of techniques employed/ known

High Reliability and Availability:

Today's complex systems contain a large amount of software

Software in operation contain a lot of bugs, in spite of best fault avoidance and fault testing/removal techniques

Software failures are a major cause of system undependability



Software fault tolerance is a potential solution to improve software reliability in lieu of virtually impossible fault-free software

Traditional Software fault tolerance

- Classical techniques are based on Design Diversity
- Use of multiple versions (or “variants”) of a software system
- Different versions may execute concurrently or sequentially
- Rationale is that multiple diverse versions will fail differently, i.e., for different inputs/workloads
- Multiple versions are developed from common specifications
- Also helps with respect to intrusion tolerance

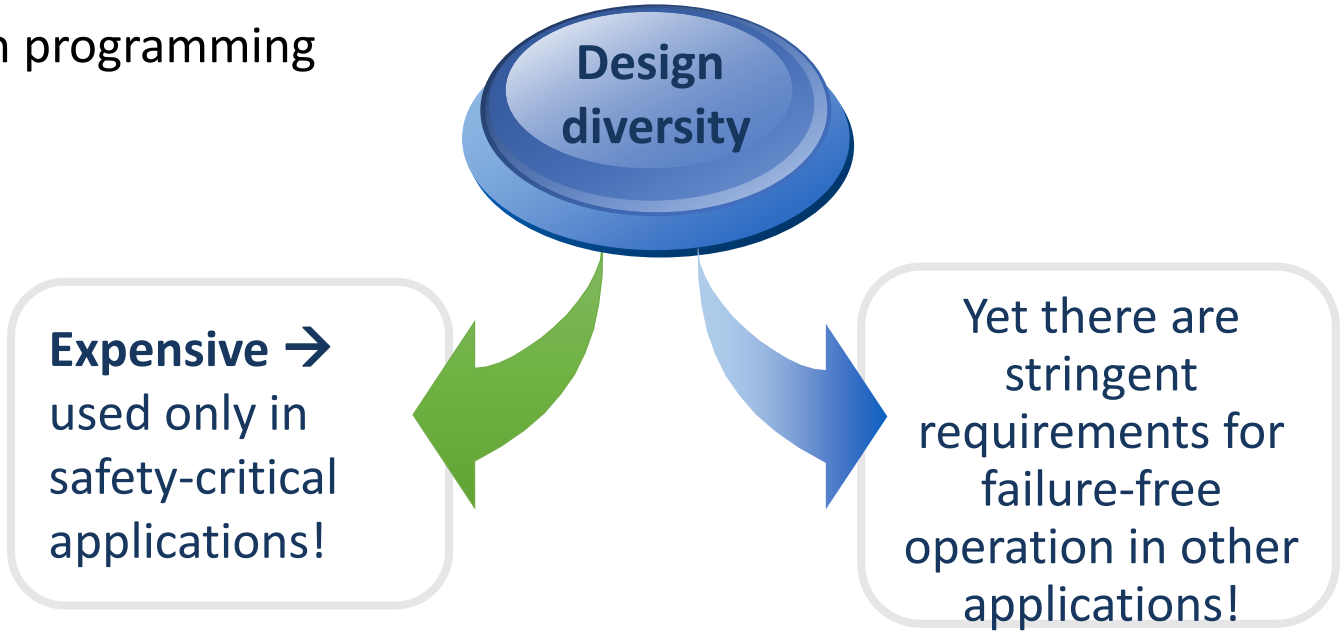
Duke UNIVERSITY

Software Fault Tolerance
Classical Techniques

- Design diversity
  - Recovery block
  - N-version programming
- Key references:
  - System structure for software fault tolerance, Randell, IEEE Trans. Soft. Eng, 1975.
  - Reliability Issues in Computing System Design, Randell, Lee and Treleaven, ACM Computing Surveys, 1978.
  - N-version version programming: a fault-tolerance approach to reliability of software operation, Chen and Avizienis, Proc. FTCS 1995.

Software Fault Tolerance: Classical Techniques

- Design diversity
  - Recovery block
  - N-version programming
  - ……



Challenge: *Affordable* Software Fault Tolerance

A possible answer: Environmental Diversity

Duke UNIVERSITY

Methods to Improve Software Reliability/Availability

- Fault Avoidance or Fault Prevention
  - Good software engineering practices, formal methods
  - Employing highly reliable components/subsystems
- Fault Removal by testing/debugging
  - Software reliability growth models
- Fault Tolerance or Use of Redundancy
  - Design diversity
  - Environmental Diversity
- Fault/Failure Forecasting

Outline

- Motivation
- Real Examples of software fault tolerance
- Software Bug Classification
- Environmental Diversity
- Methods of Mitigation
- Conclusions

Duke UNIVERSITY

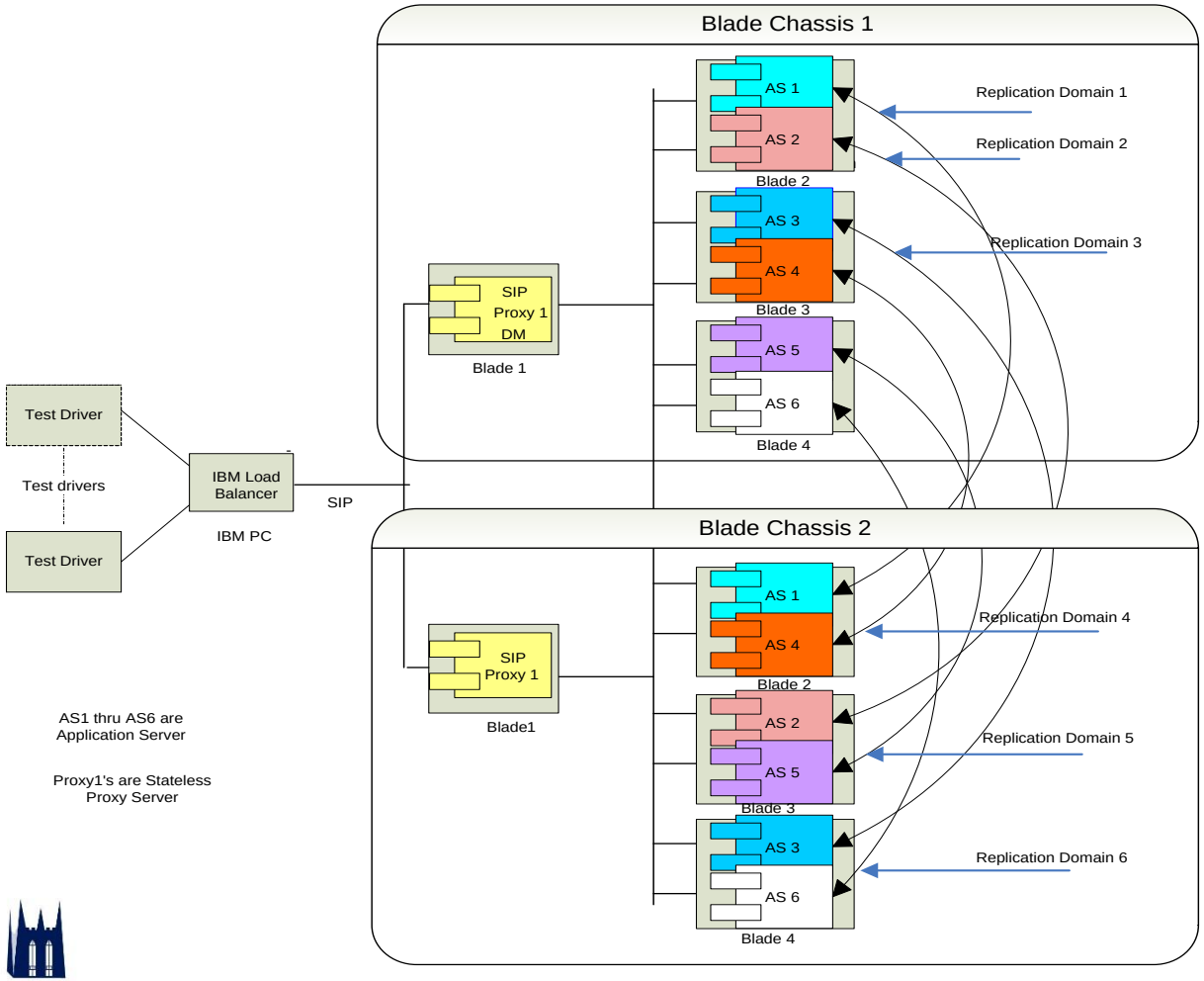
Real System: SIP on WebSphere

IBM Implementation (around 2007)

High-Availability SIP System

- Real System Developed by IBM
- SIP: Session Initiation Protocol
- Hardware platform: IBM Blade Center
- Software platform: IBM WebSphere
- Telco customer asked IBM for models to quantify this product
- IBM asked me to lead the modeling project
  - To quantify system (steady-state) availability
    Ref: Trivedi, Wang, Hunt, Rindos, Smith, Vashaw, "Availability Modeling of SIP Protocol on IBM WebSphere," *PRDC 2008*
  - To quantify a user-oriented metric called DPM
    Ref: Trivedi, Wang & Hunt. "Computing the number of calls dropped due to failures," *ISSRE2010*

High availability SIP Application



More details in papers:

- **PRDC 2008**
- **ISSRE 2010**

High availability SIP Application

- Hardware configuration:
  - Two BladeCenter chassis; 4 blades (nodes) on each chassis (**1 chassis is sufficient from performance perspective**)
- Software configuration:
  - 2 copies of SIP/Proxy servers (**1 sufficient for performance**)
  - 12 copies of WebSphere Application Server (WAS or AS) (**6 copies sufficient for performance**)
  - Each WAS instance forms a redundancy pair (**replication domain**) with WAS installed on another node on a different chassis
- Fault Tolerance:
  - The system has both **hardware redundancy**
  - and **software redundancy.**

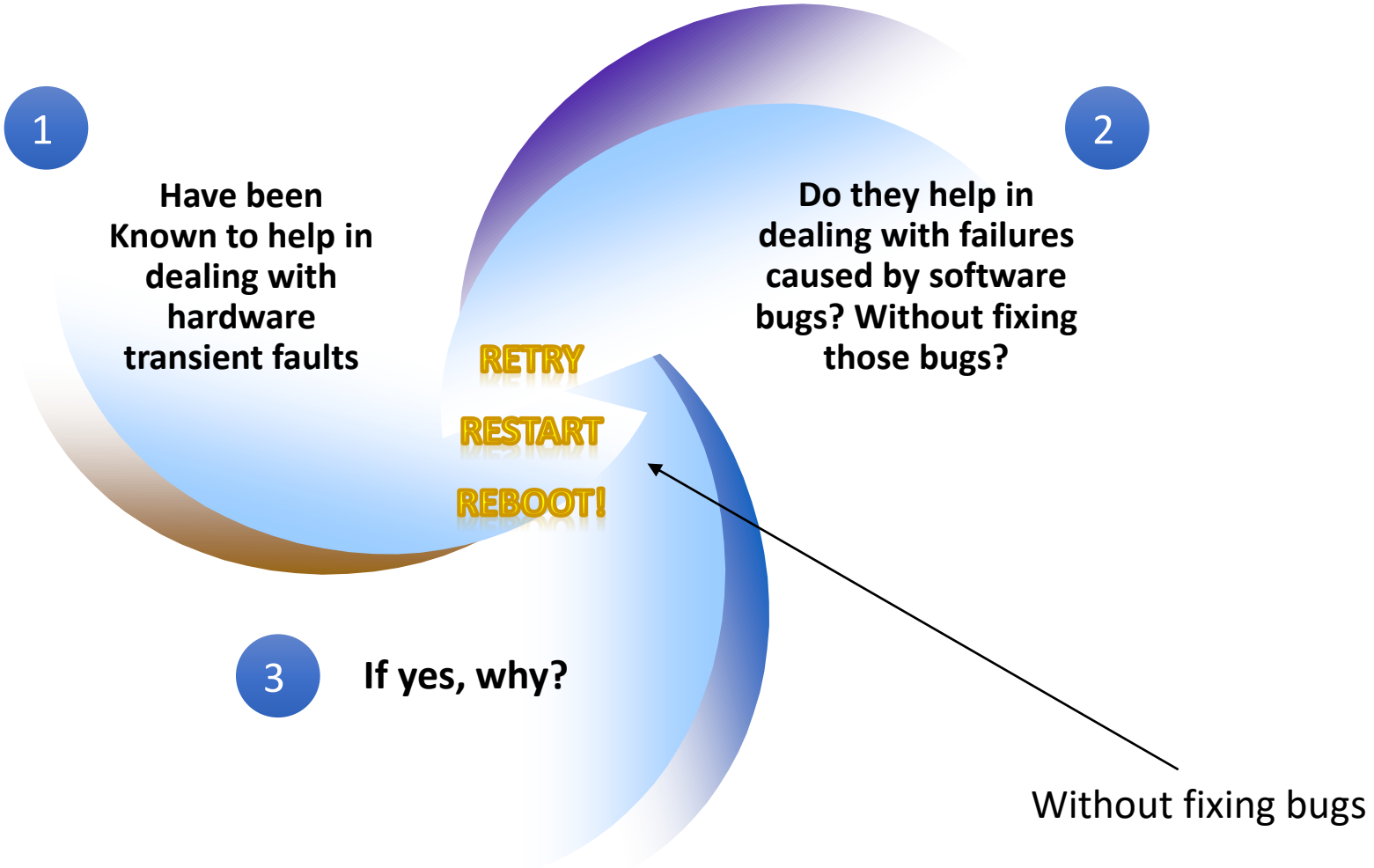
Duke UNIVERSITY

High availability SIP Application

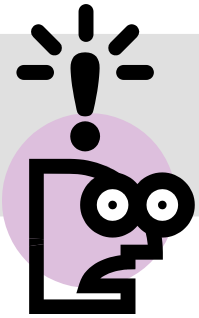
- **Software Redundancy**
  - Identical copies of SIP proxy used as backups (**hot spares**)
  - Identical copies of WebSphere Applications Server (WAS) used as backups (**hot spares**)
  - **Type of software redundancy** – (not design diversity) but replication of identical software copies
  - **Normal recovery after a software failure – uses time redundancy**
    - ✓Restart software, reboot node or fail-over to a software replica; only when all else fails, a “software repair” is invoked

Duke UNIVERSITY

Software Fault Tolerance: New Thinking



Software Fault Tolerance: New Thinking



Does it help?

If yes, why?

Thirty years ago this would be considered crazy!

Failover to an identical software replica
(that is not a diverse version)
They have exactly the same bugs

Bugs are not all equal !

- **Fault triggers** make the difference

- Some bugs are "**trivial**", and failures caused by them can be easily "reproduced" once detected during test
- Other bugs are "**subtle**", and even "reproducing failures caused by them" is challenging
  - *Race conditions*
  - *Memory leaks*
  - *Hardware-software interaction related bugs*
  - ...

  These bugs have a significant impact in terms of software failures and costs

Duke UNIVERSITY

Outline

- Motivation
- Real System Examples
- Software Bug Classification
  - "**Fighting Bugs: Remove, Retry, Replicate and Rejuvenate**," Grottke & Trivedi, *IEEE Computer Magazine,* 2007.
- Environmental Diversity
- Methods of Mitigation
- Conclusions

Duke UNIVERSITY

IFIP Working Group 10.4

- **Failure** occurs when the delivered service no longer complies with the desired output.
- **Error** is that part of the system state which is liable to lead to subsequent failure.
- **Fault** (or **bug**) is adjudged or hypothesized cause of an error.

Faults are the cause of **Errors** that may lead to **Failures**

Duke UNIVERSITY

Need to Classify bug types

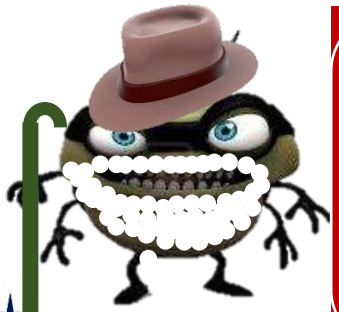
- We submit that a software fault tolerance approach based on **retry**, **restart**, **reboot** or **fail-over** to an identical software replica (not a diverse version) may work because of a significant number of software failures are caused by **Mandelbugs (environment-dependent bugs)** as opposed to the traditional software bugs now known as **Bohrbugs**.

Duke UNIVERSITY

Software fault classification

Bohrbug (BOH) := A fault that is easily isolated and that manifests consistently under a well-defined set of conditions, because its activation and error propagation lack complexity.

Non-Aging related Mandelbug (NAM) := A fault whose activation and/or error propagation are complex. Typically, a Mandelbug is difficult to isolate, and/or the failures caused by a it are not systematically reproducible.



Aging related bug (ARB) := A fault that leads to the accumulation of errors either inside the running application or in its system-context environment, resulting in an increased failure rate and/or degraded performance.

Duke
UNIVERSITY

Mandelbugs

- Besides workload and internal state of the software system, its system-context (or operating) environment participates in determining whether a failure due to such a bug will occur

➢So a fault is a Mandelbug if its manifestation as a failure is subject to the following complexity factors

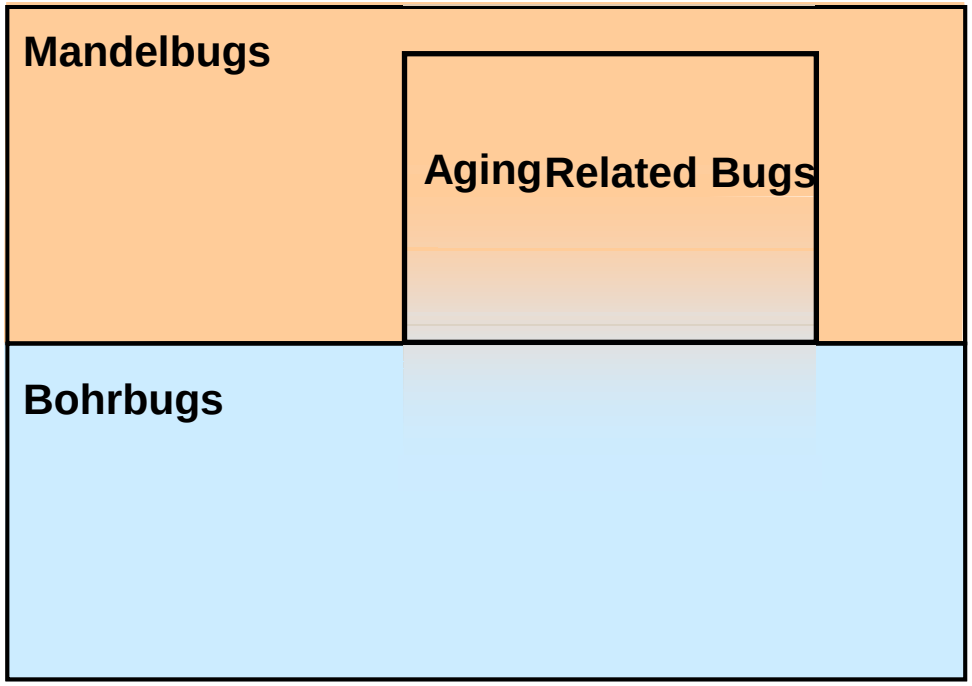
- **Long time lag** between fault activation and failure appearance
- **Operating environment dependence** (OS resources, other applications running concurrently, hardware, network…)
- **Timing** among submitted operations
- **Sequencing or ordering** of operations

➢A failure due to a Mandelbug thus may not recur upon the resubmission of the same workload if the operating *environment* has changed enough

Duke UNIVERSITY

Relationships of the Bug Types

- Bohrbug and Mandelbug are complementary antonyms.
  Aging-related bugs are a subtype of Mandelbugs



Bug Types in Several Systems

- JPL/NASA flight software - **An empirical investigation of fault types in space mission system software**, M. Grottke, A. Nikora, and K. Trivedi. *DSN*, 2010.
- Linux, MySQL, Apache AXIS, HTTPD - **Fault triggers in open-source software: An experience report**, Cotroneo, Grottke, Natella, Pietrantuono, Trivedi. *ISSRE*, 2013.
- Android operating system - **An Empirical investigation of fault triggers in Android operating system**, F. Qin, Z. Zheng, X. Li, Y. Qiao, and K. Trivedi. *PRDC*, 2017.
- Linux - **Fault Triggers in Linux Operating System: From Evolution Perspective**, G. Xiao, Z. Zheng, B. Yin, and K. Trivedi. ISSRE, 2017 (all the bug reports in Linux)

Project	LoC	% BOH	% NAM	% ARB	% UNK
JPL/NASA		61.4	32.1	4.4	2.1
Linux	1.31M	42.2	41.9	8.3	7.6
MySQL	453K	56.6	30.3	7.7	5.4
HTTPD	145K	81.1	10.5	7.0	1.4
AXIS	80K	92.5	3.5	4.0	0.0
Android		65.2	27.0	4.4	3.4
Linux2		55.8	31.7	7.8	4.7

Duke UNIVERSITY

Software Faults and Mitigation Types

- The fault classification is not merely theoretical, it has also practical implications

- Each type of software fault may require different type of approach during development, testing, as well as during operations

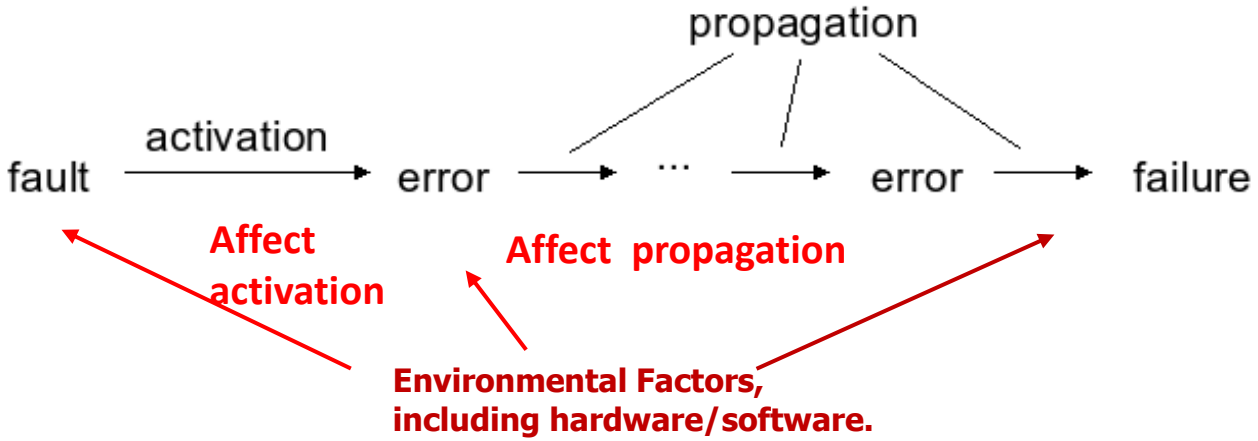
Duke UNIVERSITY

Outline

- Motivation
- Real System Examples
- Software Fault Classification
- Environmental Diversity
- Methods of Mitigation
- Conclusions

Duke UNIVERSITY

Software Fault Tolerance:
Motivation of Environmental Diversity



- **Motivation: For a Mandelbug,** environmental factors could affect **the fault activation and/or error propagation**. Examples are:
  - Data Race/Deadlock, whose fault activation could be affected by other concurrently running processes/threads → after a retry/restart/reboot we may not observe the failure.
  - Memory Leak, whose error propagation could be influenced by the size of available memory → rejuvenation/reconfigure may avoid/postpone the failure.

Software Fault Tolerance: New Thinking

- **Environmental Diversity** as opposed to **Design Diversity**
- Our claim is that this (**retry**, **restart**, **reboot**, **failover to identical software copy**) may well work since failures due to **Mandelbugs** are not negligible (**41.9%** in Linux bug reports). We thus have an affordable software fault tolerance technique that we call **Environmental Diversity**

Duke UNIVERSITY

Software Fault Tolerance:
What is Environmental Diversity?

- The basic idea of Environmental Diversity
  - **Restart an application** (without fixing the bug) **after recovery** and it most likely works -- Why?
  - Because of the environment where the application executed in has changed enough to avoid the fault activation.
- The environment is understood as
  - OS resources, other applications running concurrently and sharing the same resources, interleaving of operations, concurrency, or synchronization.
- This is Fault Tolerance because we do not necessarily fix the fault; fault caused a failure but this failure is dealt with by using time redundancy hence the user may not experience the failure again on retry.

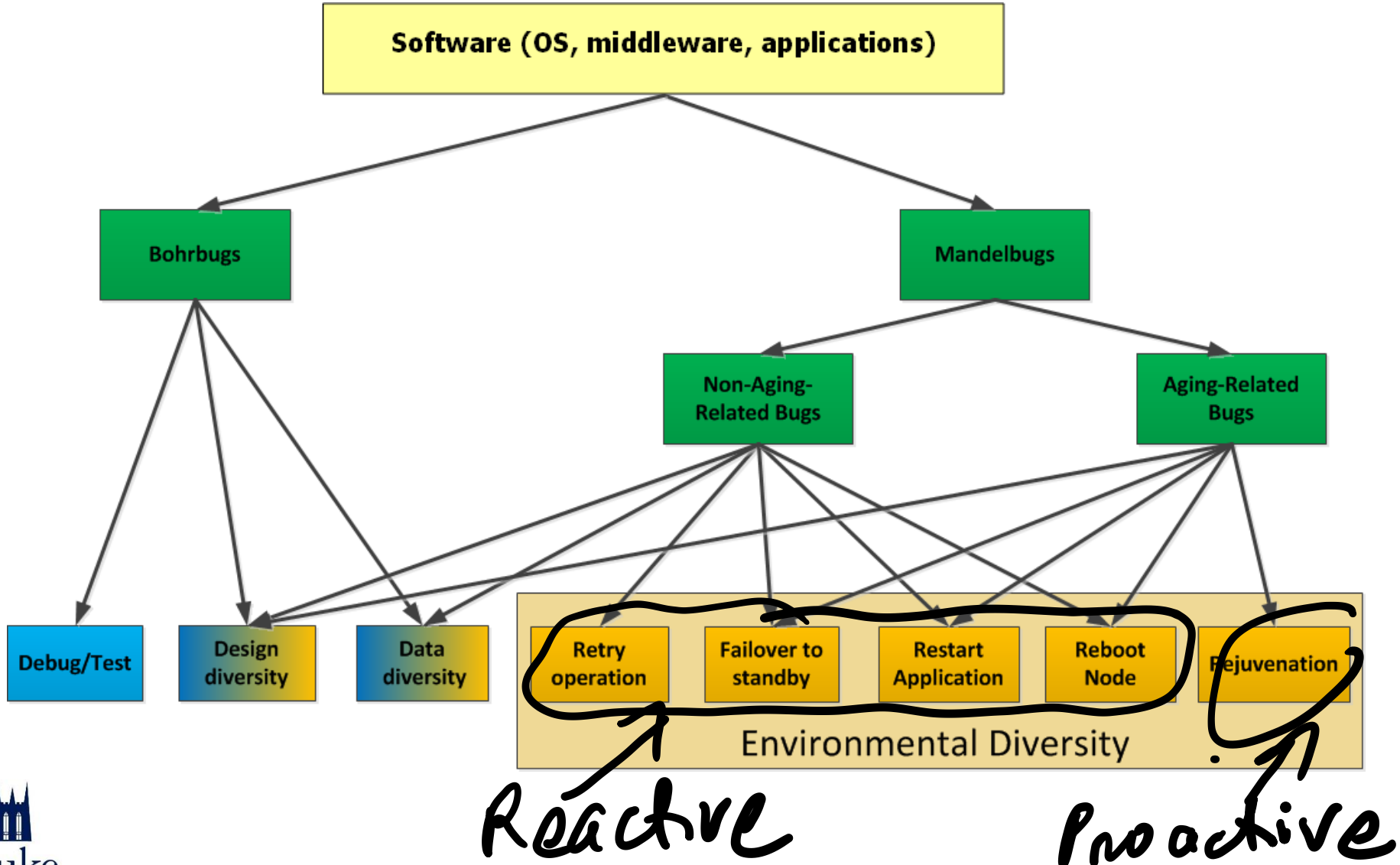
Duke UNIVERSITY

Outline

- Motivation
- Real System Examples
- Software Fault Classification
- Environmental Diversity
- Methods of Mitigation
- Conclusions

Duke UNIVERSITY

Methods of Mitigation



Implications of Mandelbugs

- Can measure/model software ***availability***
- Combined software and hardware availability
- Need:
  - Develop methods of debugging and testing for environment-dependent bugs
  - Methods to determine environmental factors and their effects
  - Run-time control of environmental factors to avoid failure occurrences
  - Optimal recovery sequence after failure occurrence
  - Experimental methods to determine the nature software failure times including use of ALT

Duke
UNIVERSITY

Mandelbug "Reproducibility"

- **(Failures due to) Mandelbugs** are really hard to reproduce
  - Conducted a set of experiments to study the environmental factors that affect the reproducibility of Mandelbugs in MySql
    - disk usage,
    - memory occupancy
    - Concurrency level
  - High usage levels of environmental factors increases significantly failure occurrences due to Mandelbugs
- **Reproducibility of Environment-Dependent Software Failures: An Experience Report**, Cavezza, Pietrantuono, Alonso, Russo, Trivedi, *ISSRE*, 2014.

Duke UNIVERSITY

Outline

- Motivation/Definitions
- Real System Examples
- Software Fault Classification
- Environmental Diversity
- Methods of Mitigation
- Conclusions

Duke UNIVERSITY

Key Points

- Today's complex systems (including SDN, CPS, smartgrid, aircraft, spacecraft, weapon systems, and IoT) contain a huge amount of software → **Software is a major cause of system undependability**

- **Software failures during operation are a fact that we need to learn to deal with**. Traditional method of software fault tolerance, based on design diversity, is expensive and hence does not get used extensively. Software fault tolerance based on ***inexpensive environmental diversity*** **could be exploited**

- The focus in the Software Engineering community so far has been on **software faults**; we need to pay equal attention to **failures** caused by software bugs (faults) and the recovery from these failures during the operational phase

- Focus in the Software Reliability Engineering community so far has been on software reliability; we need to pay attention to **software availability** as well

Duke UNIVERSITY

Methods to Improve Software Reliability/Availability

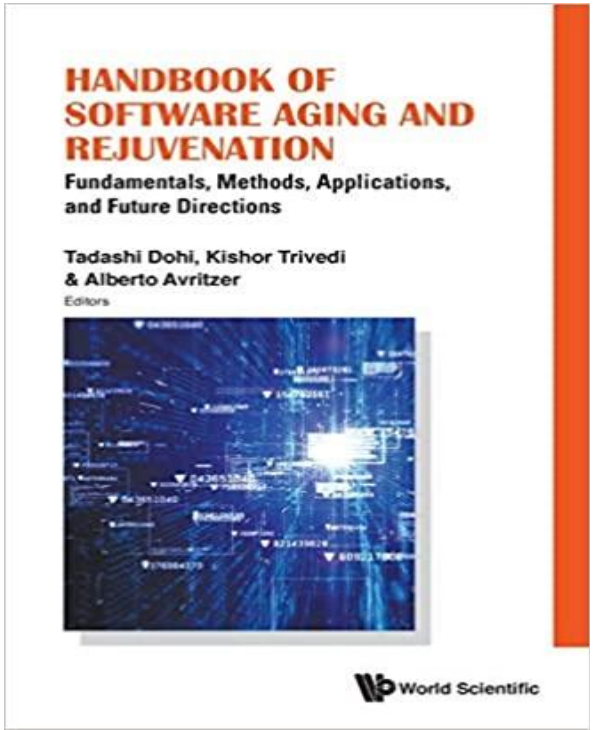
- Fault Avoidance or Fault Prevention
  - Good software engineering practices, use of model checking
  - Good software architecture, Use of FMEA
  - Use of Micro-services
- Fault Removal by testing/debugging
  - Employing good software testing methods, Use of FMEA
  - Software reliability growth models
- Fault Tolerance or Use of Redundancy
  - Design diversity/ Environmental Diversity - Fault Tolerance patterns
- Fault/Failure Forecasting
  - Reliability modeling to Identify bottlenecks
  - Availability modeling to Identify bottlenecks
  - Predict when failures may occur and thence use for preventive maintenance scheduling (software rejuvenation)
  - Identify fault-prone modules to help allocate testing resources

Some More Notes

- Conventional wisdom is that unlike hardware, software does not age, so preventive maintenance does not help in software
- However, since 1995 it has been recognized that software does age and software rejuvenation (preventive maintenance)

  does help improve

  software reliability/availability



Key References

Fault Classification

- **Orthogonal Defect Classification – A Concept for In-process Measurements**, R. Chillarege, et al., IEEE Trans. on Software Engineering, 1992
- **Fighting Bugs: Remove, Retry, Replicate and Rejuvenate**, Grottke, Trivedi, IEEE Computer, 2007
- **An Empirical Investigation of Fault Types in Space Mission System Software**, Grottke, Nikora and Trivedi, Proc. DSN, 2010
- **Analysis of Bugs in Apache Virtual Computing Lab**, Frattini, Ghosh, Cinque, Rindos, and Trivedi, IEEE/IFIP DSN Workshop, 2013
- **An empirical study of fault triggers in the Linux operating system: An evolutionary perspective**, Xiao, Zheng, Yin, Trivedi, IEEE Transactions on Reliability, 2019
- **An empirical investigation of fault triggers in Android operating system**, Qin, Zheng, Li, Qiao, Trivedi, IEEE Transactions on Reliability, 2019

Duke UNIVERSITY

Key References

Environmental Diversity and Methods of Mitigation(1)

- **Performance and Reliability Evaluation of Passive Replication Schemes in Application Level Fault Tolerance**, S. Garg, Y. Huang, C. M. R. Kintala, K. S. Trivedi, and S. Yajnik. Proc. FTCS 1999.
- **Whither generic recovery from Application Faults? A fault study using Open-Source Software**, Chandra S., Chen P. M., Proc. CDSN 2000.
- Vibhu S. Sharma, Kishor S. Trivedi, "**Reliability and Performance of Component Based Software Systems with Restarts, Retries, Reboots and Repairs**", 17th Int. Symp. on Software Reliability Engineering (ISSRE 2006)
- **An Empirical Investigation of Fault Repairs and Mitigations in Space Mission System Software** J. Alonso, M. Grottke, A. Nikora, and K. Trivedi. Proc. DSN 2013.
- **Software fault mitigation and availability assurance techniques**, K. Trivedi, M. Grottke, and E. Andrade. International Journal of System Assurance Engineering and Management, 2011.

Duke UNIVERSITY

Key References

Environmental Diversity and Methods of Mitigation(2)

- **Fault triggers in open-source software: An experience report**, Cotroneo, Grottke, Natella, Pietrantuono, Trivedi, ISSRE 2013
- **Reproducibility of environment-dependent software failures: an experience report**, Cavezza, Pietrantuono, Alonso, Russo, Trivedi , ISSRE 2014
- **Recovery from software failures caused by mandelbugs**, Grottke, Kim, Mansharamani, Nambiar, Natella, & Trivedi, IEEE Trans. on Reliability, 2015
- **Understanding the impacts of influencing factors on time to a datarace software failures**, Qiu, Zheng, Trivedi, Yin, ISSRE 2017
- **An empirical investigation of fault triggers in Android operating system**, Qin, Zheng, Li, Qiao, Trivedi, IEEE Transactions on Reliability, 2019

Duke UNIVERSITY

Key References

Prediction of Fault-Proneness

- **Analysis and Prediction of Mandelbugs in an Industrial Software System,** Carrozza, Cotroneo, Natella, Pietrantuono, Russo, ICST, 2012
- **Studying Aging-Related Bug Prediction Using Cross-Project Models**
- Qin, Zheng, Qiao, Trivedi, IEEE Transactions on Reliability, 2018
- **Supervised Representation Learning Approach for Cross-Project Aging-Related Bug Prediction,** Wan, Zheng, Qin, Qiao, Trivedi, IEEE Int. Symp. on Software Reliability Engineering (ISSRE), 2019

Duke UNIVERSITY