

Reliable Computing I

Lecture 1: Introduction

Instructor: Mehdi Tahoori

INSTITUTE OF COMPUTER ENGINEERING (ITEC) – CHAIR FOR DEPENDABLE NANO COMPUTING (CDNC)



KIT – University of the State of Baden-Wuerttemberg and
National Research Center of the Helmholtz Association

www.kit.edu

Today's Lecture

- Logistics
- Course Outline
- Introduction

Logistics



- Instructor: Mehdi Tahoori
 - Office: Room B2-313.1, Building 07.21
 - Email: mehdi.tahoori@kit.edu
 - Tel: 721-608-47778, Fax: 721-608-43962
 - Office hours: Wednesday 13:00-14:00
 - Secretary: Ms. Iris Schroeder-Piepk
- Lecture:
 - When: Wednesday 14:00-15:30
 - Where: Multimedia HS 301, Building 50.34
- Lecture notes
 - Available online: <http://cdnc.itec.kit.edu>
 - Under (Education → current semester → Reliable computing I)

Logistics (cont)



- Requirements
 - Computer Architecture
- Background on (preferred but not required)
 - Logic Design
 - Algorithms and Programming
 - Operating system (OS)
 - Basic probabilities
- Related Lectures
 - Testing Digital Systems I and II

Reference Books



- D.K. Pradhan, ed., *Fault Tolerant Computer System Design*, Prentice-Hall, 1996
- D.P. Siewiorek and R.S. Swarz, *Reliable Computer Systems - Design and Evaluation*, Digital Press, 1998, 3rd edition.
- I. Koren and C. M. Krishna, *Fault-Tolerant Systems*. Morgan Kaufmann, 2007
- P. K. Lala, *Self-Checking and Fault-Tolerant Digital Design*. Morgan Kaufmann, 2001
- B. W. Johnson, *Design and Analysis of Fault Tolerant Digital Systems*, Addison Wesley, 1989

Course Outline



- Introduction
 - Historical perspectives
 - Motivation for reliable system design
 - Failure sources
- Metrics and definitions
 - Defects, faults, errors, failures
 - Reliability metrics
 - Reliability evaluation
- Hardware reliability techniques
 - Error masking techniques
 - Error detection and recovery techniques
- Software reliability techniques

Course Goals



- Understanding the basic concepts in reliable system design, metrics, evaluation, and requirements
- Introduced to classical fault tolerant techniques
- Being familiar to current challenges in reliable computing
 - Maybe you become interested in doing research in reliable computing ;-)
- Being able to apply some of these concepts to systems
 - Hardware and software

Course Outline (detailed)



- Lecture 1: Introduction to reliable computing
- Lecture 2: Reliability metrics
- Lecture 3: Faults, errors, failures
- Lecture 4: Hardware redundancy
- Lecture 5: Reliability evaluation
- Lecture 6: Information redundancy-1
- Lecture 7: Information redundancy-2
- Lecture 8: Redundant disk arrays
- Lecture 9: Concurrent Error Detection
- Lecture 10: Re-execution
- Lecture 11: Checkpointing and Recovery
- Lecture 12: Software fault tolerance

Today's lecture: Outline

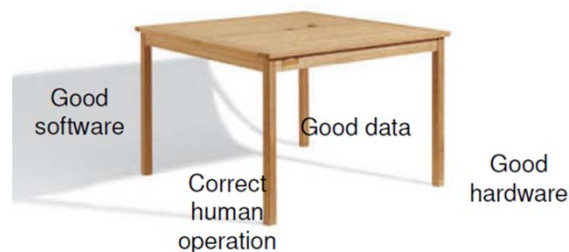


- Need for reliability: real disaster stories!
- Reliability: past, present, future
- Challenges
- Examples of reliable system design

Correct operation in computing



- These are the system components.
- All are needed for proper operation



Motivation



- Fault tolerance has always been around
 - NASA's deep space probes
 - Medical computing devices (e.g., pacemakers)
 - But this had been a niche market until fairly recently
- But now fault tolerance is becoming more important
 - More reliance on computers
- Extreme fault tolerance
 - Avionics, car controllers (e.g., anti-lock brakes), power plants and power delivery network, medical systems, etc.
- High fault tolerance
 - Commercial servers (databases, web servers), file servers, high performance computing (HPC), etc.
- Some fault tolerance
 - Desktops, laptops (really!), smartphones, game consoles, etc.

Why We Need High Reliability?



- High availability systems:
 - Telephone
 - Transaction processing: banks/airlines
- Long life missions:
 - Unscheduled maintenance too costly
 - Long outages, manual reconfiguration OK
 - Critical applications
- Critical applications:
 - Real-time industrial control
 - Flight control
- Ordinary but widespread applications:
 - CDs: encoding
 - Internet: packet retransmission

Stuff Happens



- We wouldn't need fault tolerance otherwise!
- Physical problems
 - Melted wire
 - Toasted chip
- Design flaws
 - Incorrect logic (e.g., Pentium's FDIV, AMD's quad-core TLB bug)
 - Buggy software (e.g., Vista)
- Operator error
 - Incorrect software installation
 - Accidental use of `rm -R *`
- Malicious attacks
 - Security is beyond the scope of this course

eBay Crash



- eBay: giant internet auction house
 - A top 10 internet business
 - Market value of \$22 billion
 - 3.8 million users as of March 1999
 - Bidding allowed 24x7
- June 6, 1999
 - eBay system is unavailable for 22 hours with problems ongoing for several days
 - Stock drops by 6.5%, \$3-5 billion lost revenues
 - Problems blamed on Sun server software
- Shorter downtimes common

Ariane 5 Rocket Crash



- Ariane 5 and its payload destroyed about 40 seconds after liftoff (1996)
- Error due to software bug:
 - Conversion of floating point to 16-bit int
 - Out of range error generated but not handled
- Testing of full system under actual conditions not done due to budget limits
- Estimated cost: 120 million DM

The Therac-25 Failure



- Therac-25 is a linear accelerator used for radiation therapy
- More dependent on software for safety than predecessors (Therac-20, Therac-6)
- Machine reliably treated thousands of patients, but occasionally there were serious accidents, involving major injuries and 1 death (1985-1987).
- Software problems:
 - No locks on shared variables (race conditions).
 - Timing sensitivity in user interface.
 - Wrap-around on counters.

Tele Denmark



- Tele Denmark Internet, ISP
- August 31, 1999
 - Internet service down for 3 hours
 - Truck drove into the power supply cabinet at Tele Denmark
 - Where were the UPSs?
 - Old ones had been disconnected for upgrade
 - New ones were on the truck!

The Cost of Failure



Steve Wozniak's Prius Problems: Apple Co-Founder Thinks Software Is To Blame For Toyota Problem

Huffington Post / AP | First Posted: 4/5/10 | Updated: 5/25/11

React >



Steve Wozniak's Toyota Prius has an acceleration problem, according to the Apple co-founder. And Wozniak believes it has something to do with the vehicle's software.

For its part, Toyota did not include Prius models in its sweeping recall to correct accelerator problems, though [new questions](#) have arisen.

The billionaire tech co-founder has complained about the problem in a [Bloomberg report](#).

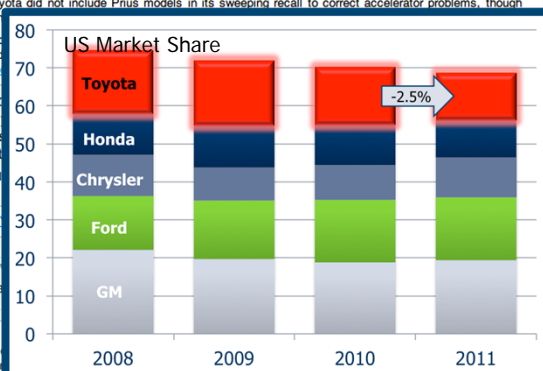
Wozniak's 2007 Prius (150 kilometers per hour) was in the car when he was interviewed yesterday.

Wozniak hasn't responded to his complaints in the past two weeks. "It's scary when it happens," Wozniak, 59, said from San Jose. "I want to listen to me."

Wozniak publicly stated his Prius claim last week. [The New York Times](#) reported that he had said the car was "a piece of junk."

"Toyota has this accelerator problem we've all heard about," Wozniak said in a video posted on [CNET.com](#) (via [Autoblog](#)). "I've been driving it for a while, and it didn't get recalled. This new model has an acceleration problem. And I can repeat it over and over and over again."

U.S. Transportation Secretary Ray LaHood said Wednesday he would lead a review, looking beyond Toyota vehicles, into whether automakers are doing enough to protect consumers. Toyota has said it investigated for electronic problems.



The Cost of Failure

Faulty software may have destroyed Mars orbiter, says NASA

January 18, 2007

(NaturalNews) NASA's Mars Global Surveyor orbiting craft stopped responding to commands in November, the administration announced Wednesday, one day after officials told scientists that the craft may have been in for disaster since faulty software was uploaded to it during the summer.



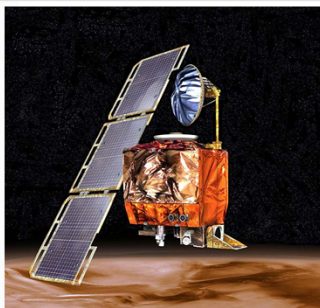
The Cost of Failure

CNN Tech

SEARCH

POWERED BY Google

Home Video NewsPulse U.S. World Politics Justice Entertainment Tech Health Living Travel Opinion iReport Money Sports



Metric mishap caused loss of NASA orbiter

September 30, 1999

Share Twitter Email

Recommend

25 people recommend this. Be the first of your friends.

CNN NASA lost a 125 million Mars orbiter because a Lockheed Martin engineering team used English units of measurement while the agency's team used the more conventional metric system for a key spacecraft operation, according to a review finding released Thursday.

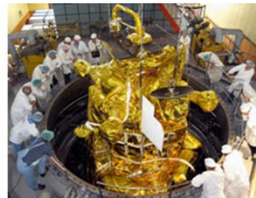
The units mismatch prevented navigation information from transferring between the Mars Climate Orbiter spacecraft team in at Lockheed Martin in Denver and the flight team at NASA's Jet Propulsion Laboratory in Pasadena, California.

Lockheed Martin helped build, develop and operate the spacecraft for NASA. Its engineers provided navigation commands for Climate Orbitsers thrusters in English units although NASA has been using the metric system predominantly since at least 1990.

The Cost of Failure

Weekly Inside Technology News

February 23, 2012



Did Bad Memory Chips Down Russia's Mars Probe?

The failure of Russia's ambitious Phobos-Grunt space mission has been shrouded in confusion and mystery—from the first inklings of trouble after its November launch to inconsistent reports of where it fell to Earth in January, and beyond. The release of the official accident investigation results on 3 February served only to further rumors of fundamental hardware and software design flaws, and of blatant violations of safety standards. The report blames the loss of the probe primarily on memory chips that became fatally damaged by cosmic rays. But observers place the blame squarely on the engineers who put Phobos-Grunt together, noting that if these military-grade chips (which were not radiation hardened) had been proposed for a critical component in a space-probe design for the U.S. space program, they probably would not have been approved for use. Worse still for Russia's space agency are reports suggesting that the chips were counterfeits that had been intentionally misrepresented as offering higher performance than they were actually capable of.

(c) 2013, Mehdi Tahoori

Reliable Computing I: Lecture 1

21

The Cost of Failure

The Pentium Problem

Late last year [1994] there was a major flap in the media about Intel's Pentium (TM) chip.

...

Intel publicly announced that "an error is only likely to occur [about] once in nine billion random floating point divides", and that "an average spreadsheet user could encounter this subtle flaw once in every 27,000 years of use." Critics noted that while hitting a pair of "bad inputs" was unlikely, the Pentium's output for those inputs was wrong every time. Others suggested that some "bad inputs" might occur with disproportionate frequency in common calculations. Many noted that without completely repeating massive calculations on other computers, they could never tell if they had indeed encountered any of the bad inputs. Within a month IBM halted shipment on Pentium-based computers (which comprised only a small percentage of IBM's computer production) and announced that "Common spreadsheet programs, recalculating for 15 minutes a day, could produce Pentium-related errors as often as once every 24 days."

Intel's policy, when it first publicly admitted the problem around November 28 of 1994, was to replace Pentium chips only for those who could explain their need of high accuracy in complex calculations. (Being a math professor seemed to help.) Great public outcry ensued, with Intel the butt of many jokes. By late December Intel capitulated and announced a free replacement Pentium for any owner who asked for one.

...



(c) 2013, Mehdi Tahoori

Reliable Computing I: Lecture 1

22

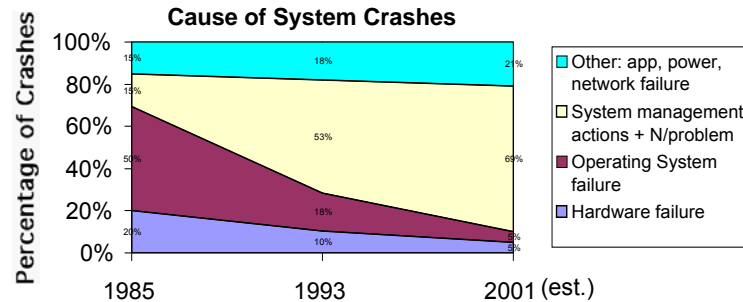
Examples of Computer-related Failures

		FAULTS			FAILURES		Availability / Reliability	Safety	Confidentiality
		Physical	Design	Interaction	Localized	Distributed			
June 1980	False alerts at the North American Air Defense (NORAD) [Ford 85]	✓			✓		✓		
April 1981	First launch of the Space Shuttle postponed [Gaman 81]		✓		✓		✓		
June 1985 - January 1987	Excessive radiotherapy doses (Therac-25) [Leveson & Turner 93]		✓		✓			✓	
November 1988	Internet worm [Spatford 89]		✓	✓		✓	✓		
15 January 1990	9 hours outage of the long-distance phone in the USA [Neumann 95]		✓			✓	✓		
February 1991	Scud missed by a Patriot (Dhahran, Gulf War) [Neumann 95]		✓	✓	✓		✓	✓	
November 1992	Crash of the communication system of the London ambulance service [HA 93]		✓	✓		✓	✓	✓	
26 and 27 June 1993	Authorization denial of credit card operations in France	✓	✓			✓	✓		
4 June 1996	The maiden flight of the Ariane 5 launcher ended in a failure (France)		✓		✓		✓	✓	
July 20, 2008	8-hour outage of Amazon services	✓				✓	✓		

Causes of failures: Tandem

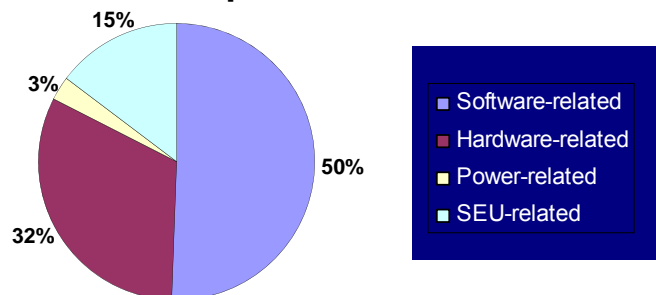
- Dominant manufacturer of fault-tolerant computer systems
 - For ATM networks, banks, stock exchanges, telephone switching centers, and other similar commercial transaction processing applications
 - Now part of HP
- In Gray's '85 survey of Tandem customers
 - 30% were "infantile" failures
 - The rest were broken into (roughly):
 - Administration 42%
 - Software 25%
 - Hardware 18%
 - Environment (power, etc.) 14%
 - Unknown 3%

Causes of failures: Vax



- VAX crashes '85, '93; extrapolated to '01
- System Management includes:
 - Multiple crashes per problem
 - System admin Actions: set params, config, bad app install
- HW/OS 70% in '85 to 28% in '93. In '01, 10%?
 - System admin increasingly important

Causes of failures: enterprise servers



- Field failure analysis of error logs in enterprise servers (2008)
 - 36 months, thousands of live systems, half a billion hours of system operation, servers in various countries

Costs!!!

■ Example of Average Cost per Hour of Downtime

Industry	Application	Average Cost per Hour of Downtime
Financial	Credit Card Sales	\$2,600,000
Media	Pay-per-view	\$1,150,000
On-line commerce	Ebay	\$225,000
Retail	Catalog Sales	\$90,000
Transportation	Airline Reservation	\$89,500

Reliability: increasing concern

■ Historical

- High reliability in computers was needed in critical applications: space missions, telephone switching, process control etc.

■ Contemporary

- Extraordinary dependence on computers: on-line banking, commerce, cars, planes, communications etc.
- Hardware is increasingly more fault-prone
- Software is increasingly more complex
- Things simply will not work without special reliability measures

Why Study Reliable Computing!!!



- **Traditional needs**
 - Long-life applications (e.g., unmanned and manned space missions)
 - Life-critical, short-term applications (e.g., aircraft engine control, fly-by-wire)
 - Defense applications (e.g., aircraft, guidance & control)
 - Nuclear industry
- **Newer critical-computation applications**
 - Health industry
 - Automotive industry
 - Industrial control systems, production lines
 - Banking, reservations, switching, commerce

Why Study Reliable Computing!!! (cont.)



- **Networks**
 - Wired and wireless networked applications
 - Data mining
 - Information processing on the Internet
 - Distributed, networked systems (reliability and security are the major concerns)
 - Intranet - stores, catalog industry (commercial computing)
 - Cloud computing
- **Scientific computing, education**
 - Arrival of supercomputers puts high requirements on reliability
 - Petascale and exascale computing

Need for fault tolerance: Universal & Basic



- Natural objects:
 - Fat deposits in body: survival in famines
 - Clotting of blood: self repair
 - Duplication of eyes: graceful degradation upon failure
- Man-made objects
 - Redundancy in ordinary text
 - Asking for password twice during initial set-up
 - Duplicate tires in trucks
 - Coin op machines: check for bad coins

Reliability issues are not new...



- In the July 1834 issue of the Edinburgh Review, Dr. Dionysius Lardner published the article “Babbage’s calculating engine”, in which he wrote:

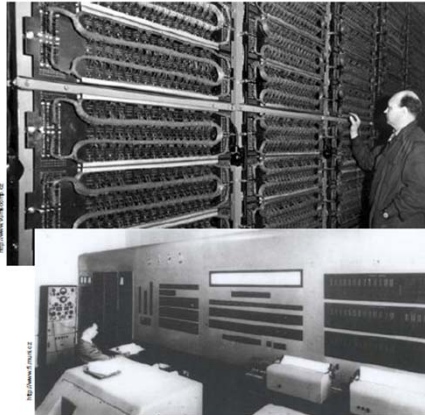


Babbage's Calculating Engine (1833)

“The most certain and effectual check upon errors which arise in the process of computation, is to cause the same computations to be made by separate and independent computers; and this check is rendered still more decisive if they make their computations by different methods.”

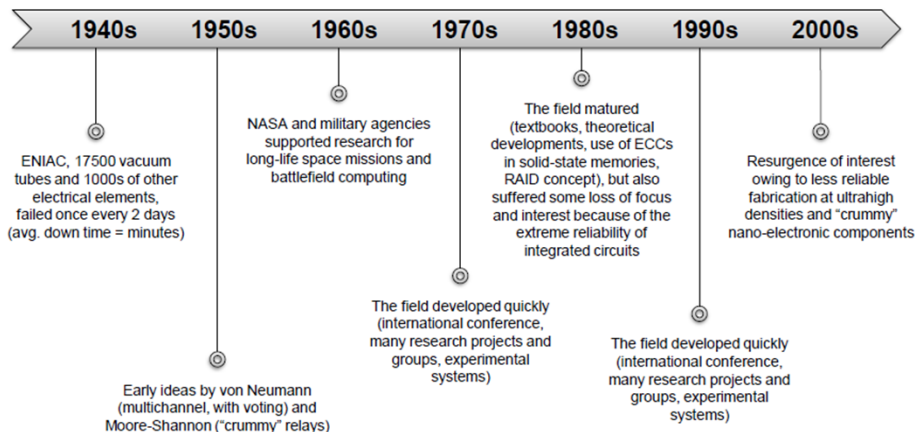
First Fault-Tolerant Computer

■ The SAPO (Samočinný počítač) fault-tolerant computer system



- Designed and built between 1950 and 1956 by **Antonín Svoboda** in Prague
- Three parallel arithmetic units, which decided on the correct result by voting (TMR)
- Electromechanical design:
 - 7000 relays,
 - 400 vacuum tubes
 - magnetic drum memory
- The system burnt down in the year 1960 after a relay failure

Timeline



Only 25 years in between



SUN Server

Server for 6 people
25 MHz
16 MIPS / 2.6 MFLOPS
640 MB hard disk drive
Footprint $\sim 2\text{m}^3$

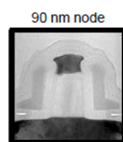
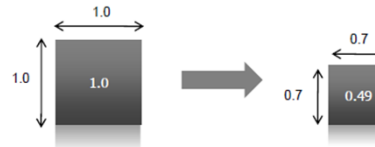
40x
13.5x
50x
2000x

Smartphone

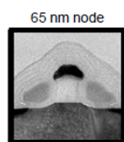
Personal use
1 GHz
35 MFLOPS
32 GB flash memory
Footprint $\sim 0.001\text{m}^3$

Technology scaling

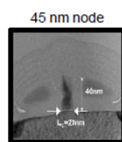
- Decrease feature size from generation to generation
- Typical: scale down dimensions by 30%
- Scaling essentially aims towards:
 - Higher speed
 - Lower power
 - Higher density



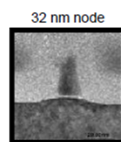
2003



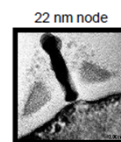
2005



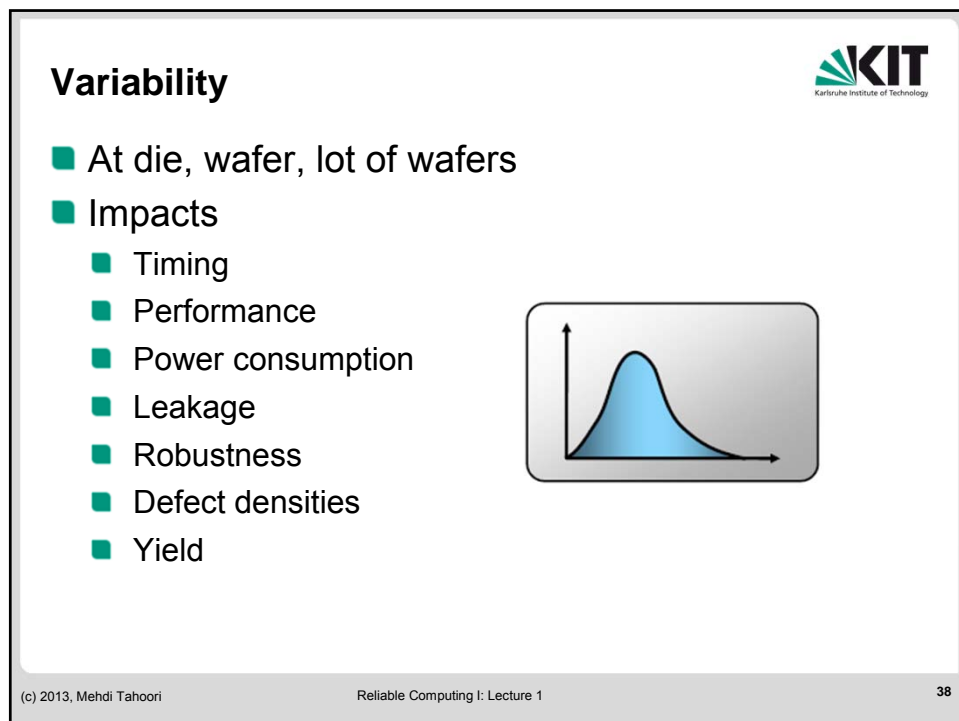
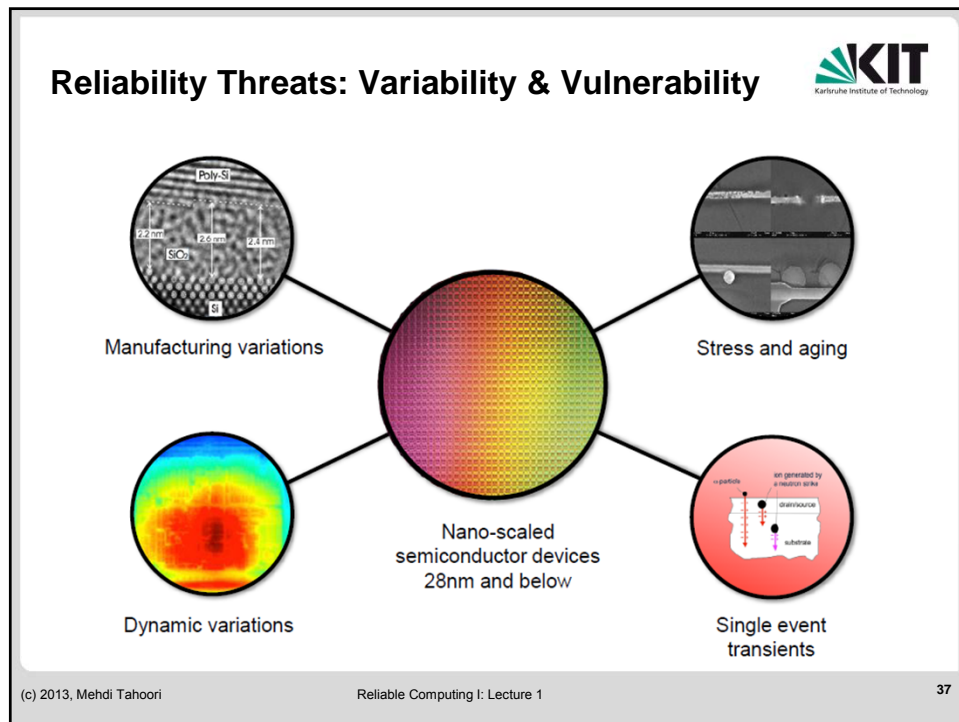
2007



2009



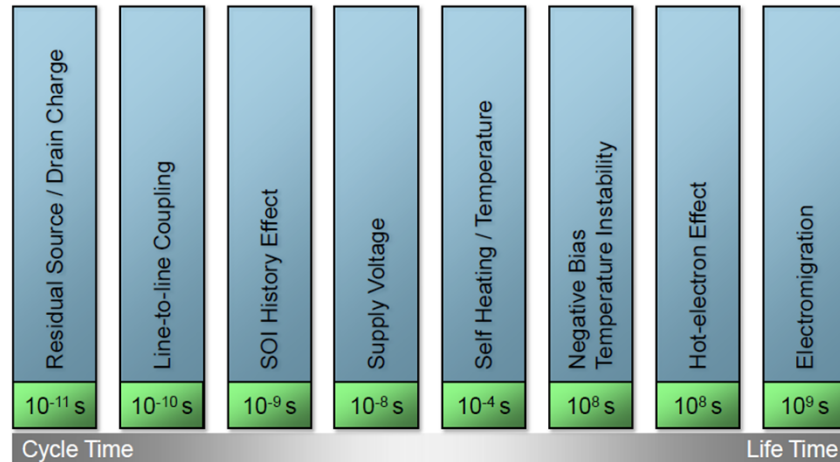
2011



Variability effects over time



■ High temporal variability



(c) 2013, Mehdi Tahoori

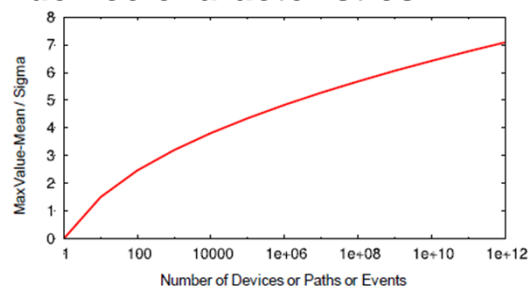
Reliable Computing I: Lecture 1

39

Increasing Device Count per Chip



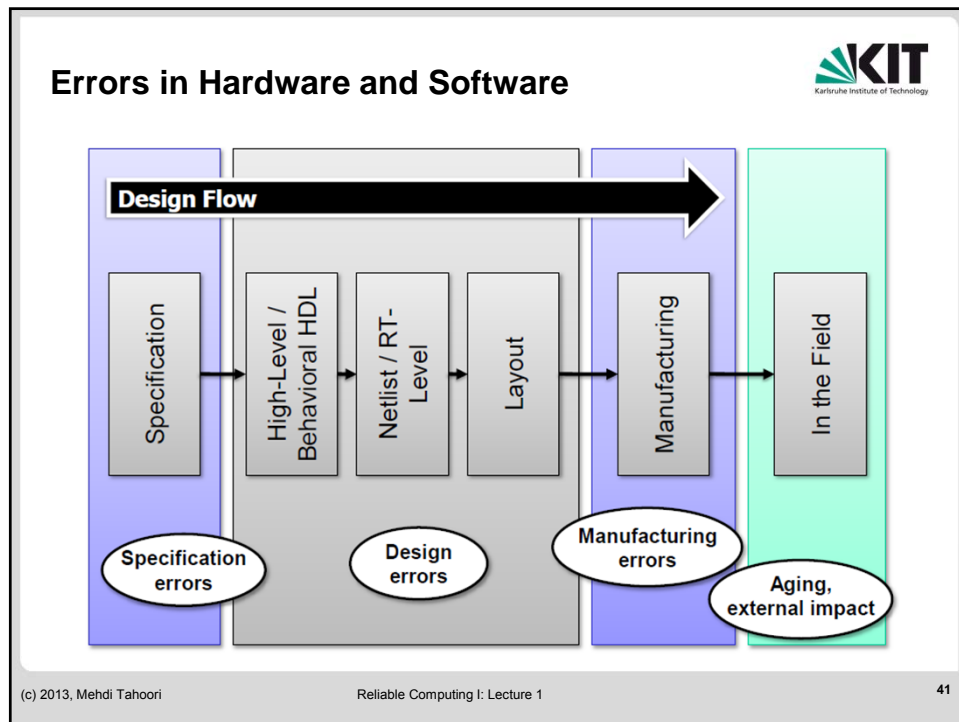
- Increased number of transistors that each chip statistically samples from the device parameter distribution.
- To achieve comparable chip-level yields via margining, we are **forced to accept a larger spread of device characteristics**.




(c) 2013, Mehdi Tahoori

Reliable Computing I: Lecture 1

40



Why Fault Tolerance Isn't Easy



- Fault tolerance can be solved to any arbitrary degree if you're willing to throw resources at the problem
- Resources to sacrifice:
 - System performance
 - Cost
 - **Power**

(c) 2013, Mehdi Tahoori Reliable Computing I: Lecture 1 42

Trying Not To Lose Performance



- There are many FT approaches that sacrifice performance to tolerate faults
- Example 1
 - Periodically stop the system and checkpoint its state to disk
 - If fault occurs, recover state from checkpoint and resume
- Example 2
 - Log all changes made to system state in case recovery is needed
 - During recovery, undo the changes from the log
- Example 3
 - Run two identical systems in parallel
 - Compare their results before using them
- Example 4
 - Run software with lots of assertions and error checking

Performance Issues



- Most important not to degrade performance during fault-free operation
 - This is the common-case → make it fast!
 - **Amdahl's Law**
- Somewhat less important not to degrade performance when a fault occurs
 - This still might not be acceptable in certain situations (e.g., real time systems)

Trying Not To Increase Cost



- There are many FT approaches that sacrifice cost to tolerate faults
- Example 1
 - Replicate the hardware 3 times and vote to determine correct output
- Example 2
 - Mirror the disks (RAID-1) to tolerate disk failures
- Example 3
 - Use multiple independent versions of software to tolerate bugs
 - Called N-version programming

Trying Not To Increase Power



- There are many FT approaches that sacrifice power to tolerate faults
- Examples 1, 2 & 3 (same as previous slide)
 - Replicate the hardware 3 times and vote to determine correct output
 - Mirror the disks (RAID-1) to tolerate disk failures
 - Use multiple independent versions of software to tolerate bugs
- Example 4
 - Add continuously running checking hardware to system
- Example 5
 - Add extra code to check for software faults

Levels of Fault Tolerance



- Fault tolerance can be at many levels in a system:
 - Application software
 - Adding assertions to code
 - Operating system
 - Protecting OS from hanging
 - Entire hardware system
 - Hardware sub-system
 - Circuits and transistors

Example 1: Telephone Switching System



- **Extreme availability**
 - Goal: <3 minutes of downtime per year
 - Goal: <0.01% of calls processed incorrectly
- Uses physical redundancy
- Hardware cost: about 2.5 times cost of equivalent non-redundant system
- Also uses:
 - Error detecting/correcting codes (e.g., parity, CRC)
 - Watchdog timers
 - Many forms of diagnostics
 - Dynamic verification (“sanity program”)

Example 2: IBM Mainframes



- Lots of fault tolerance → high availability
- Note: IBM has produced mainframes since the 1960s, and they've changed their design and enhanced their fault tolerance several times since then
- Redundancy at many levels
 - Redundant units within processor (e.g., register file)
 - Redundant processors
- Diagnostic hardware for isolating faults
- Reliable operating system

Example 3: My (previous) Laptop



- Minimal fault tolerance
- Designed to be cheap and fast ... and obsolete in a few years
- May have parity or ECC on:
 - Some bus lines
 - DRAM
 - Hard disk
- Expected lifetime (as expected by me): 2 years
- Expected Mean Time To Reboot (also by me): 1 week
- Expected Mean Time To Re-install: 6 months
- Expected probability of it successfully coming out of "hibernation" mode: 0

Example 4: Database Software (Oracle, DB2)



- **Lots of fault tolerance**
- Can't afford to corrupt vital data
- Enforces “**ACID**” properties for transactions & data
 - **A**tomicity: transactions are atomic
 - **C**onsistency: only valid data written to database
 - **I**solation: transactions don't interfere with each other
 - **D**urability: data written to database won't be lost
- Implemented with logging and checkpointing
- Writes important data to disks
 - Can even tolerate a fault that occurs while writing to disks