Dependable Systems

Hardware Dependability - Testing

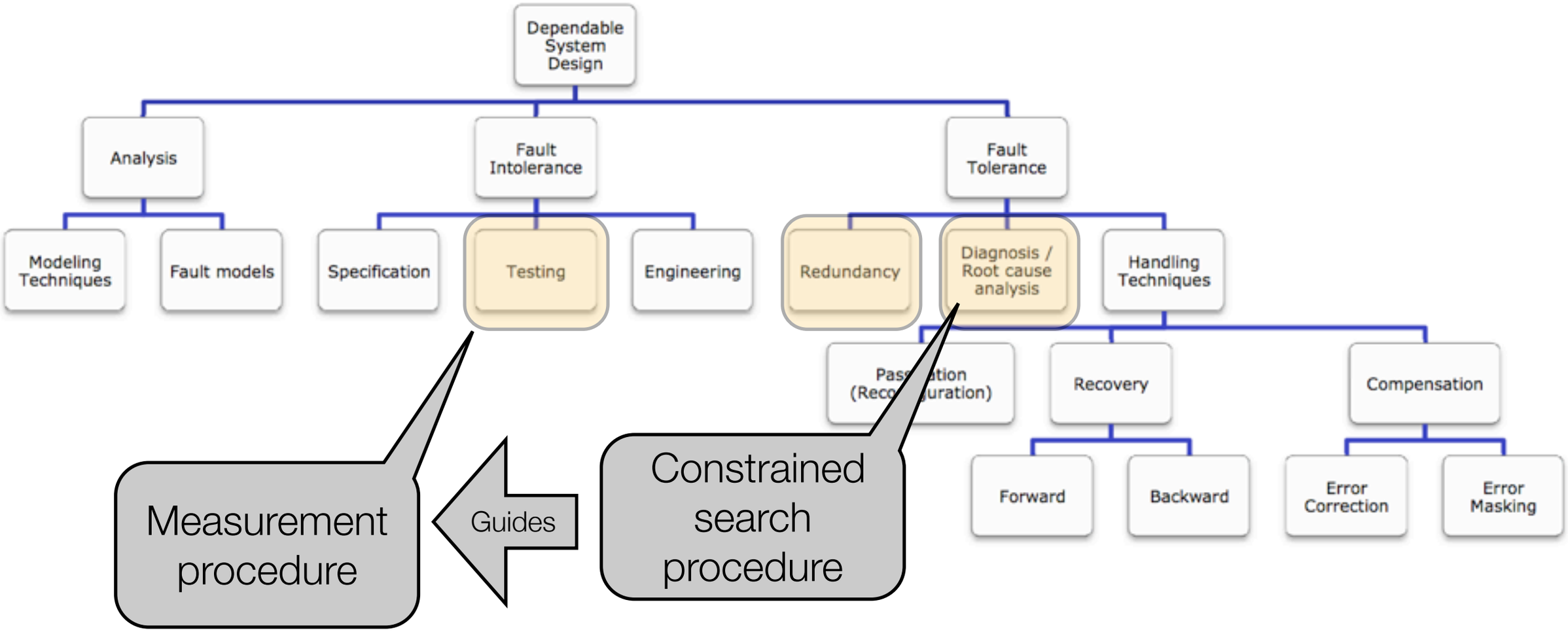
Dr. Peter Tröger

Sources:

Siewiorek, Daniel P.; Swarz, Robert S.:
Reliable Computer Systems. third. Wellesley, MA : A. K. Peters, Ltd., 1998. ,
156881092X

Ziade, Haissam; Ayoubi, Rafic A.; Velazco, Raoul:
A Survey on Fault Injection Techniques. In: Int. Arab J. Inf. Technol. 1 (2004), Nr. 2,
S. 171-186

Dependable System Design (Echtle)



Testing [Sieworek & Swartz]

- Core of specification-based diagnosis is testing - black box approach
- Type of fault is influencing the test procedure - logical fault vs. structural fault
- Development phase influences test procedure - functional vs. acceptance test

		Design Phase	**Production Phase**	**Operational Phase**
Hierarchical Level	System	Design maturity test - Reach specification goals	Maturity test	Synthetic load
	Logic	Simulation	Acceptance test	Built-in test
	Circuit	Simulation	Parametric test, up to stress tests	Margining

Temporal Stage

Hardware Testing

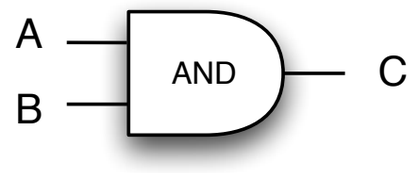
- What makes hardware testing today difficult and expensive ?
  - Large number of faults and fault classes
  - Limited observability and controllability
  - Pattern sensitivity in large density circuits
  - Increasing circuit speeds
  - Increasing system complexity
  - Exponentially growing number of test patterns
  - Incomplete information about a system
    - Companies often do not disclose system description

Hardware Design for Testability

- Testing problem - Test generation vs. test verification
  - Fault simulation as only option for measuring test effectiveness
- Design for testability
  - Accept the risk of shipping a defect product, but support testing in a better way
    - Ad-hoc approaches: Partitioning, extra test points at board level, bus systems
    - Structured approaches: Allow observability for state variables in the system
- Stress testing approaches help in production phase to force infant mortalities
  - Vibration, over-voltage, burn-in, thermal shock, ...
- Largest body of theory exists for logic-level acceptance testing (production phase)
  - Usually, single structural stuck-at faults are assumed
  - Define the „unit under test“, stimulus can be on-chip / off-chip

Processor Testing

- Typical approach is logic testing - over 50.000 papers
- Algorithms for test generation, designed to be programmable
- **D-algorithm** - guarantees to find a test vector if one exists
  - Invented by Roth at IBM in 1966, based on **D notation**
    - Compact way, by a collapsed truth table, to describe how faults propagate through a circuit
    - D - „1“ in the good circuit, „0“ in the faulty
    - D‘ - „0“ in the good circuit, „1“ in the faulty
  - Works for stuck-at faults, bridging faults, logic flaws
- *Primitive D cube of failure (PDF)* - sets of input that will bring the fault to the output



Fault	PDF		
	A	B	C
A stuck-at-0	1	1	D
B stuck-at-0	1	1	D
C stuck-at-0	1	1	D
A stuck-at-1	0	1	D‘
B stuck-at-1	1	0	D‘
C stuck-at-1	X	0	D‘

Processor Testing

- Functional testing (Thatte/Abraham, 1978)
  - Data flow graph model
    - Nodes represent registers or active units, links correspond to data transfer paths in the processor hardware
    - Three classes of instructions (fault model is defined for each class)
      - Sequencing & control
      - Data storage and transfer
        - Example fault model: Any number of lines can be stuck-at-1 or 0
      - Data manipulation
- Functional testing is usually supplemented by additional pseudorandom tests

Memory Testing

- More and more important issue
  - Key component for electronic systems, embedded memory eats most transistors
  - About 30% of the semiconductor market
- Multitude of memory testing approaches
  - Caching problem
    - Memory testing is performed by the processor
    - Disable caches or use appropriate techniques, such as modulo
  - Layout problem
    - Testing procedure must consider organization of RAM hardware
- Testing time vs. fault coverage trade off, additional runtime monitoring

RAM Testing

- Categories: Functional (1‘s and 0‘s),
  DC parametric (signal timing, fall and rise),
  AC parametric (access times,
  setup / hold times and cycle times)
- Exhaustive test
  - Total number of bit configurations is 2^N
  - N! potential address sequences
  - 16 bit RAM
    - 65536 words possible
    - 2.092279e+13 possible sequential writing sequences
    - Total of 1.371196e+18 combinations !

Test	Complexity
MSCAN	4n
Column Bars	4n
Checkerboard	4n
Marching 1‘s / 0‘s	12n
Shifted Diagonal	$4n^{3/2}$
Ping Pong	n^2
Walking 1‘s / 0‘s	$2n^2 + 6n$
Galloping 1‘s & 0‘s	$2n^2 + 8n$

RAM Testing

- **Memory Scan (MSCAN)**
  - For all cells: Write 0, read, write 1, read
  - Can detect any stuck-at fault in the memory, memory data register, or logic
  - Will not detect stuck-at in the memory address register or in the decoder
- **Checkerboard / Volatility Test Pattern**
  - Write 1‘s in all even locations and 0‘s in all odd locations
  - Wait
  - Read and compare all
  - Repeat with complementary pattern
  - Can check for hold time in dynamic memories

RAM Testing

- **Marching 1‘s / 0‘s**
  - Step 1 - For all cells: Write 0
  - Step 2 - For all cells: Read and compare, write 1, read and compare
  - Step 3 - For all cells backward: Read and compare, write 0, read and compare
  - Step 4 - Repeat steps 1 to 3 with complementary pattern
  - Detection of many decoder errors, minimal check on cell interaction
- **Ping Pong**
  - Chose designated test cell, test read / write interaction with all other cells
  - Detect pattern sensity and the interaction between pair cells

RAM Testing

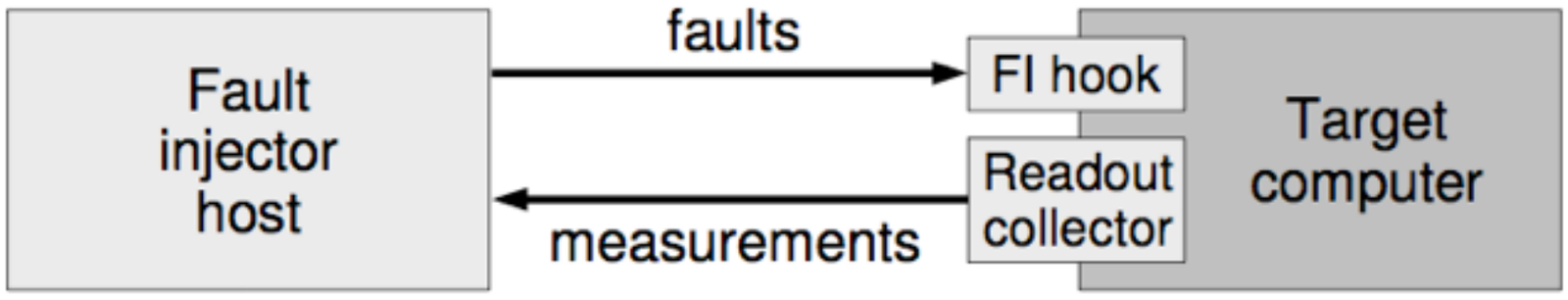
- **Walking 1‘s / 0‘s**
  - Step 1 - For all cells: Write 0
  - Step 2 - For all cells: Write 1 to test cell, read and compare all others, read and compare test cell, write 0 to test cell
    - Checks for independence of cell operations and that decoder addressing works
    - Detects also sense amplifier problems
- **Galloping 1‘s / 0‘s**
  - Same as above, but read 0 test is always followed by read 1 test in test cell
- Complexity Problem: 2 GB RAM = $2*10^9$ Bytes = $16*10^9$ Bits
  - Time effort with checkerboard (4n) and 10ns access time: 10,6 min
  - Reduction by chip-level parallel tests

Stress Test

- Multitude of approaches for hardware stress test
- Example by Hobbs Engineering [Misra]:
  Highly accelerated life tests (HALT), highly accelerated stress screens (HASS)
  - Improve reliability of products by harsh testing in production phase
  - ‚Discovery testing‘, while system modeling normally targets compliance
- Classical HALT
  - Apply monitoring with high coverage
  - Low temperature, high temperature, voltage and frequency margining
  - All axis vibration, product-specific stresses
  - Combine a few at a time, then all
  - Repeat until satisfactory level of robustness is achieved

Fault Injection / Insertion

- Typically low failure rates in standard hardware systems
  - Testing procedures (especially for fault tolerance mechanisms) might take an unacceptable amount of time
- Idea: Accelerate the failure rate by fault injection (in hardware or software)
  - Usually controlled from a second computer
  - FI hook demands trigger for injection process
  - Readout collector obtains data (and verifies injection activity)



Fault Injection Advantages [Ziade et al.]

- Understanding of the effects of real faults and the related system behavior
- Assessment of the efficiency of fault tolerance mechanisms (design faults)
- Encompassing a measurement of the coverage of the fault tolerance mechanism
- Estimate failure coverage and latency
- Explore effects of different workloads on the effectiveness of the FT mechanisms
- Study error propagation and degree of error containment
- Prove fault model correctness and coverage

Fault Injection

- **Hardware fault injection**
  - With contact / invasive (putting unspecified voltages to pins, ...)
  - Without contact (radiation, heat, particle beams, interferences, laser, ...)
- **Simulation-based methods**
  - In high-level models (e.g. VHDL)
- **Emulation-based methods**
  - Use of FPGAs for effective circuit simulation
- **SoftWare-Implemented Fault Injection (SWIFI)**
  - Modification of real system to „emulate“ faults
- **Hybrid approaches**
  - Mix hardware injection with software-based monitoring

Fault Injection

- **FARM** classification approach by Arlat et al.
  - F: Set of faults to be deliberately introduced in the system
  - A: Set of activation trajectories used to exercise the system
  - R: Set of readout values that correspond to the system behavior
  - M: Set of dependability measures obtained through the fault injection
- Input domain: Set of faults F and set of activations A
- Output domain: Set of readouts R and set of measures M
- Single experiment: Select f from F, a from A, and a workload; determine r
- M is determined from a set of experiments called *campaign*
- Each experiment is characterized by <f, a, r>

FARM Assumptions - Set of Faults (F)

- Starts with the choice of a fault model
  - As close as possible to real faults, but still manageable
- Each fault is characterized by
  - *Fault injection time* (time stamp, number of instructions, number of clock cycles)
  - *Fault location* (system component - gate, memory location, register)
  - *Fault mask* (selection of bits to be affected)
- Golden-run experiment used for fault-list generation and collapsing

Fault List Reduction

- Example - fault list reduction for processor SEUs
  - Faults changes a legal instruction code into an illegal instruction code
  - Fault affects the code of an instruction after its last evaluation
  - Fault affects memory cell before a write access, or after the last read access
  - Fault flips the same bit of instruction code as another fault
  - Fault flips the same memory bit as another fault, in between two accesses
  - Fault-injection time or fault location are such that effects do not reach the output
- Experiments show 40% reduction for randomly generated initial fault list

FARM Assumptions - Set of Measures (M)

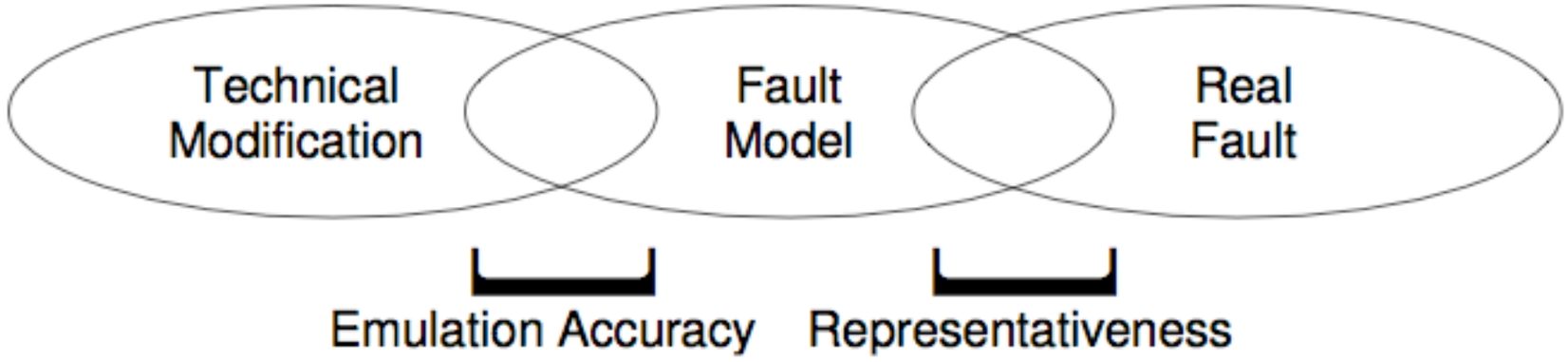
- Determine fault coverage from experiment results
  - *Effect-less faults:* Fault was not activated and was finally removed
  - *Failure:* Fault propagated through the system to the output
  - *Detected fault:* Fault produced an error, was signalled to the user
    - Different fault detection strategies - software-detected, hardware-detected, time-out detected
  - *Latent fault:* Fault remained passive, or became an error that never escalated to the system output
  - *Corrected fault:* System was able to identify an process the error on it‘s own

(HW) Fault Insertion Methods [Sieworek & Swartz]

	Method			
	Software Simulation	Hardware Emulation	Fault Emulation	Physical Insertion
Pro	Access to system at any detail level, fault types and control are unlimited	Representative hardware with favorable access and monitoring	True hardware and software in use	True hardware and software in use
Cons	Simulation time explosion	Implementation and other parameters will change with deployed system	Fault types are limited	Hardware form factor limits access and monitoring points, difficult

Faults vs. Fault Injection

- The set of injected faults usually does not cover the set of targeted real faults
- The fault injection typically can only partly emulate the selected fault model
- The fault model usually covers only parts of the relevant real faults



- Testing with fault injection
  - Problem of repeatability and dependability chain analysis

Hardware-Based Fault Injection

- **With contact** - Injector has direct physical contact with the target system
  - Pin-level active probes for stuck-at and bridging (probe across two pins) faults
  - Socket insertion for stuck-at, open, or more complex logic faults
    - Inverting signal; Logical combination with other pins or previous signals
    - Supports transient / permanent / random / non-random faults
- **Without contact** - External source produces physical phenomenon for disturbances
- Benefits
  - Unique locations, supports high resolution systems, accuracy, fast experiments, broad support for fault classes, support for permanent faults
- Drawbacks
  - High risk of damage, dense packaging issues, low portability and observability, high setup time, expensive

Simulation-Based Fault Injection

- Construction of a simulation model for the hardware under test
- Typically with VHDL: Widespread use, hierarchical description capabilities, structure and behavior description in one place
- VHDL code modification vs. saboteurs insertion
  - Latter mutate existing component descriptions, supported by VHDL tool set
  - Saboteurs can model most faults, including environmental conditions (ESD)
- Benefits
  - Can support all levels (electrical, logical, functional, architectural), not intrusive, broad fault models, without extra hardware, integrated in normal design flow, support for timing-related faults, maximum observability
- Drawbacks
  - Large development effort, long experiments, model accuracy

Emulation-Based Fault Injection

- Cope with time limitations of simulation by synthesizing VHDL descriptions
- Typically, necessary injection modifications are included in the model
- External trigger from connected development machine
- Alternative: Rely on reconfiguration capabilities of the FPGA itself

- Benefits
  - Injection is faster than with simulation, low cost approach
- Drawbacks
  - VHDL must be synthesizable, only good to investigate functional failures, timing failures are not supported, restricted by I/O capabilities of the FPGA

SoftWare-Implemented Fault Injection (SWIFI)

- Use software to emulate hardware faults by representative modifications
- Additional software is expected to be independent from the rest of the system
- Compile-time vs. run-time injection
  - Latter demands trigger: Time-out, exception / trap, code insertion, debug register
- Benefits
  - Can target applications and operating systems, testing the production system, no special-purpose hardware required, no model development needed
- Drawbacks
  - Limitation to assembly instruction level, only locations that are accessible for software, additional code that does not exist with the real fault, limited controllability (e.g. processor pipeline), permanent faults are problematic

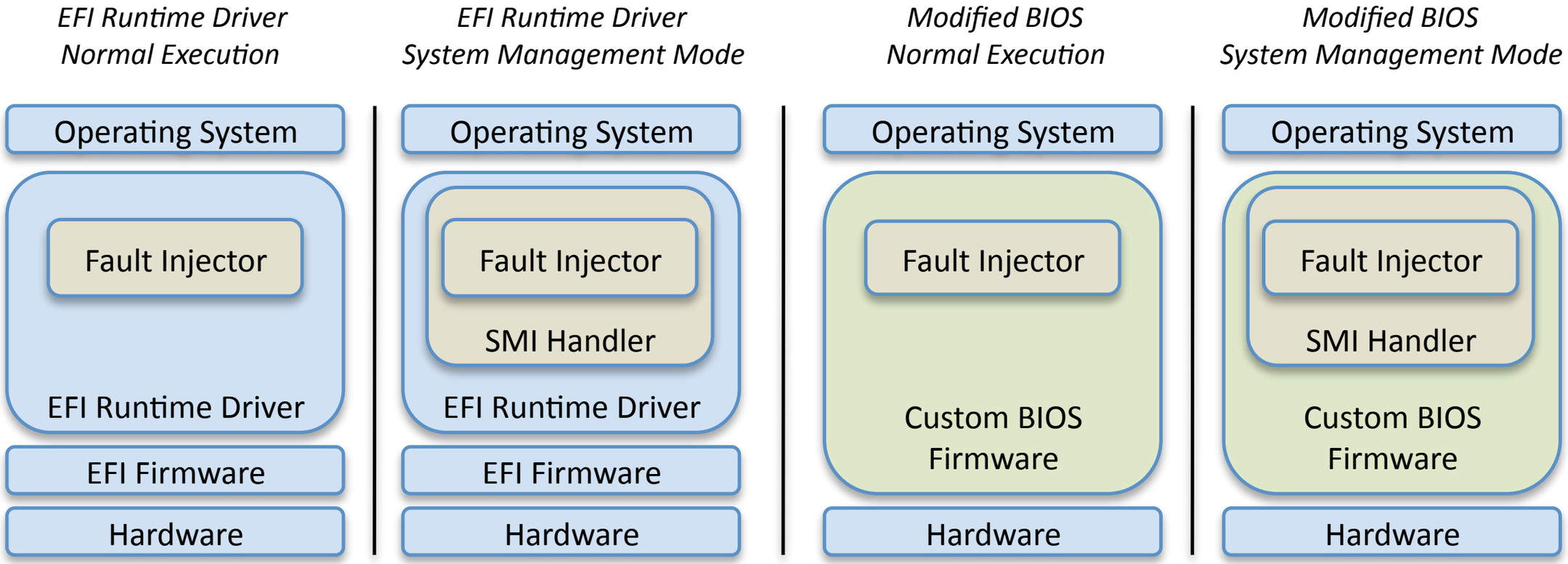
Existing Work

- Some examples
  - FERRARI [Kanawati et al.] - Trap-based triggers
  - FTAPE [Tasi & Iyer] - Kernel module, memory modification
  - FIAT / DOCTOR - Application image manipulation
  - XCeption [Carreira et. al] - Kernel module, processor debug registers
- Typical triggers: Time-out, trap, code insertion, debug register, ...
- Overall property is the need for some non-transparent modification of hardware, operating system, or application

Name	Target Architecture	Fault Trigger	Injector Location
FTAPE	Tandem Integrity S2	random, stress-level monitoring	Operating System
Ferrari	Sun SPARC	Any instruction or timer	Application
FIAT	RT PC	Instruction in the application code	Application
NEXUS	VHDL models, MPC 565	Any instruction, timer	Processor Hardware / Second Machine
XCeption	PowerPC, Pentium	Opcodes, operands, temporal triggers	Processor Hardware / Operating System Driver

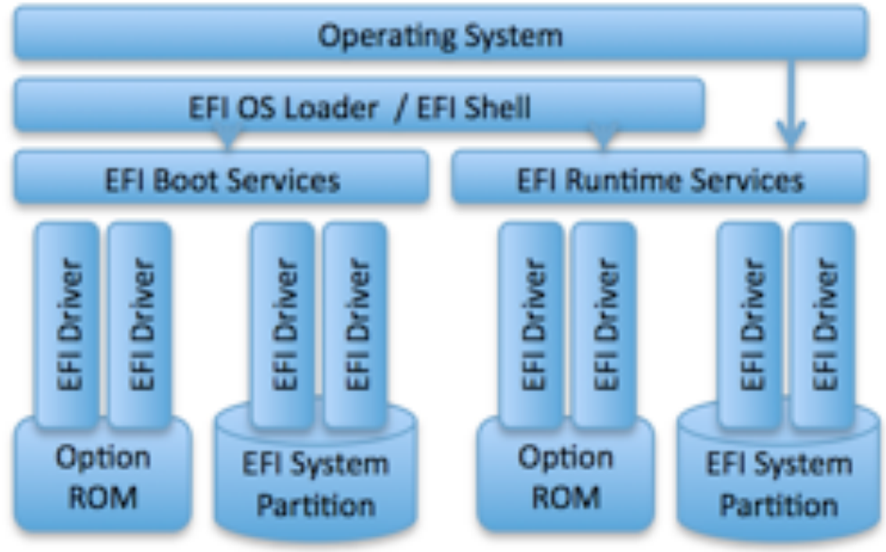
Current Research at OSM / HPI

- Software-Implemented Fault Injection at Firmware Level - a new SWIFI approach
  - New fault locations (instruction pointer, trap controller state, hypervisor, ...)
  - Better portability
  - Advanced fault triggers



X86 Firmware Technologies

- Extensible Firmware Interface (EFI)
  - Initiative by Intel, HP, and others
  - 32-bit replacement for BIOS
  - Driver concept
  - Itanium, Dell, others ...



- System Management Mode (SMM)
  - Most privileged execution mode on X86, 32bit real-mode
  - For regular BIOS activities at runtime
    - power and fan management, error handling,
    - Hardware emulation
  - Non-maskable System Management Interrupt (SMI)
  - System Management RAM for state saving

Fault Location

- Supported fault locations
  - Non-SMM approach == standard kernel approaches
  - SMM approach would bring new possibilities
    - All general purpose registers (incl. EBX)
    - Status register (EFLAGS), instruction pointer
    - Debug registers, trap controller state, MCE registers, ...
    - Caches (model-specific)
- Reduced support for memory injection in SMM mode
- Unique support for hypervisor injection in SMM mode
- I/O fully accessible in both variants

Fault Trigger

- Fault trigger
  - Strongly limited set of events detected and handled by firmware at run-time
  - Hooking of hardware interrupts only with BIOS extension
  - Could move fault trigger to operating system level
    - Brings support for all traditional SWIFI triggers
    - Injection itself still transparent to the operating system
- With some chipset support, SMI can be triggered through
  - Power button, RTC events, serial / USB port activities
  - All NMIs

Status

- EFI runtime driver implementation for fault injector
  - Software project for TianoCore EDK II framework
  - First runtime driver available in open source, difficulties with trigger and driver ‚survival‘
- Use Case: Register utilization as dependability factor
  - Dependency between register utilization and software reliability on transient faults
  - Automated testbed with injection and monitoring support

https://bitbucket.org/troeger/efinject/