Versatile Architecture-Level Fault Injection Framework for Reliability Evaluation: A First Report

Nikos Foutris, **Manolis Kaliorakis**, Sotiris Tselonis, Dimitris Gizopoulos
University of Athens

20th IEEE International On-Line Testing Symposium
Platja d'Aro, Spain, July 7-9, 2014

Clereco
Cross-Layer Early Reliability Evaluation for the Computing cOntinuum

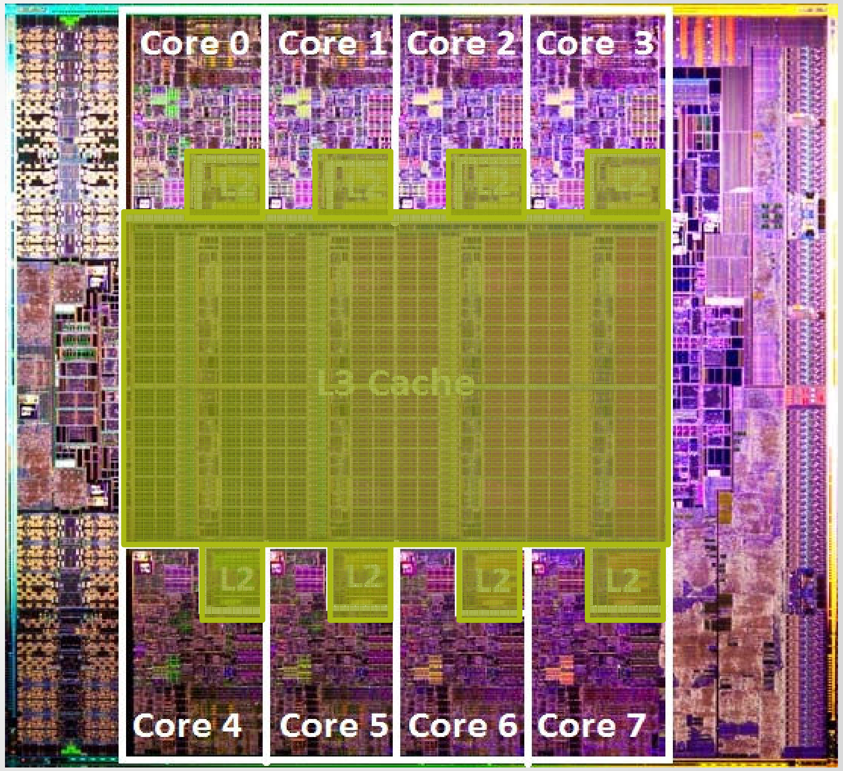
SEVENTH FRAMEWORK PROGRAMME

cal@di

Motivation

- **Technology scaling trends** lead to:
  - higher **performance**
  - roughly **constant power and cost** per chip
  - higher design density
    - **larger storage arrays**

Modern computing systems become **increasingly unreliable**



Intel’s Xeon E5-2600, DL1: 32K, IL1: 32K, L2: 256K, L3: 20M

Motivation (2)

- **Detection and correction techniques are adopted to ensure systems' functionality**:
  - especially on storage arrays (such as **caches**)

Techniques	Detect (Correct)	Area Overhead
Parity	1/64 bits (none)	**1.6%**
SEC-DED	2/64 bits (1/64 bits)	**12.5%**
DEC-TED	3/64 bits (2/64 bits)	**23.4%**
Chipkill	2/8 chips (1/8 chips)	**12.5%**
RAIM	1/5 modules (1/5 modules)	**40.6%**
Mirroring	2/8 chips (1/2 modules)	**125%**

Which is the **most suitable technique** in terms of reliability and area overhead?

Clereco Cross-Layer Early Reliability Evaluation for the Computing cOntinuum cal@di

Motivation (3)

- **Measure microprocessor reliability:**
  - **early enough** in design phase
  - **fast**
  - **accurate**

- **How important is it?**
  - Less design effort and resources
  - Carefully integration of reliability mechanism
  - Reduce time-to-market (TTM)

Outline

- Early Reliability Estimation
- Proposed fault injection framework
- Experimental setup & results
- Conclusion

Background Analysis

Architecture-Level injection is the most suitable for early reliability estimation

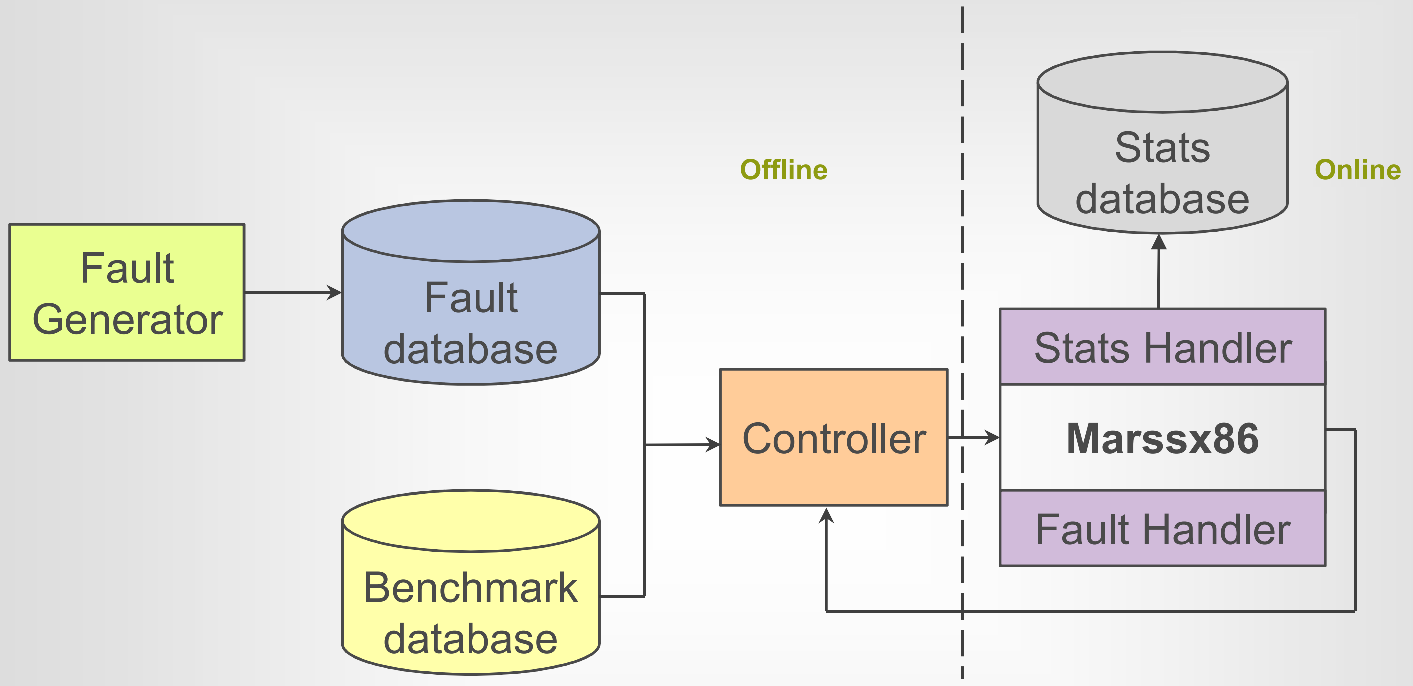
	RT-Level injection	Architecture-Level injection	ACE analysis	Probabilistic models
Simulation Time	High	Medium	Low	None
Fault Model Accuracy	High	Medium	None	None
Estimation Accuracy	High	High	Medium	Medium

- **ACE analysis and Probabilistic models** → conservative lower bound of reliability
- **RTL injection** → accurate, **BUT** suffers from low simulation throughput
- **Architecture-Level injection** → accurate for early reliability estimation of arrays
- **Only performance models** exist early on design phase

Contribution

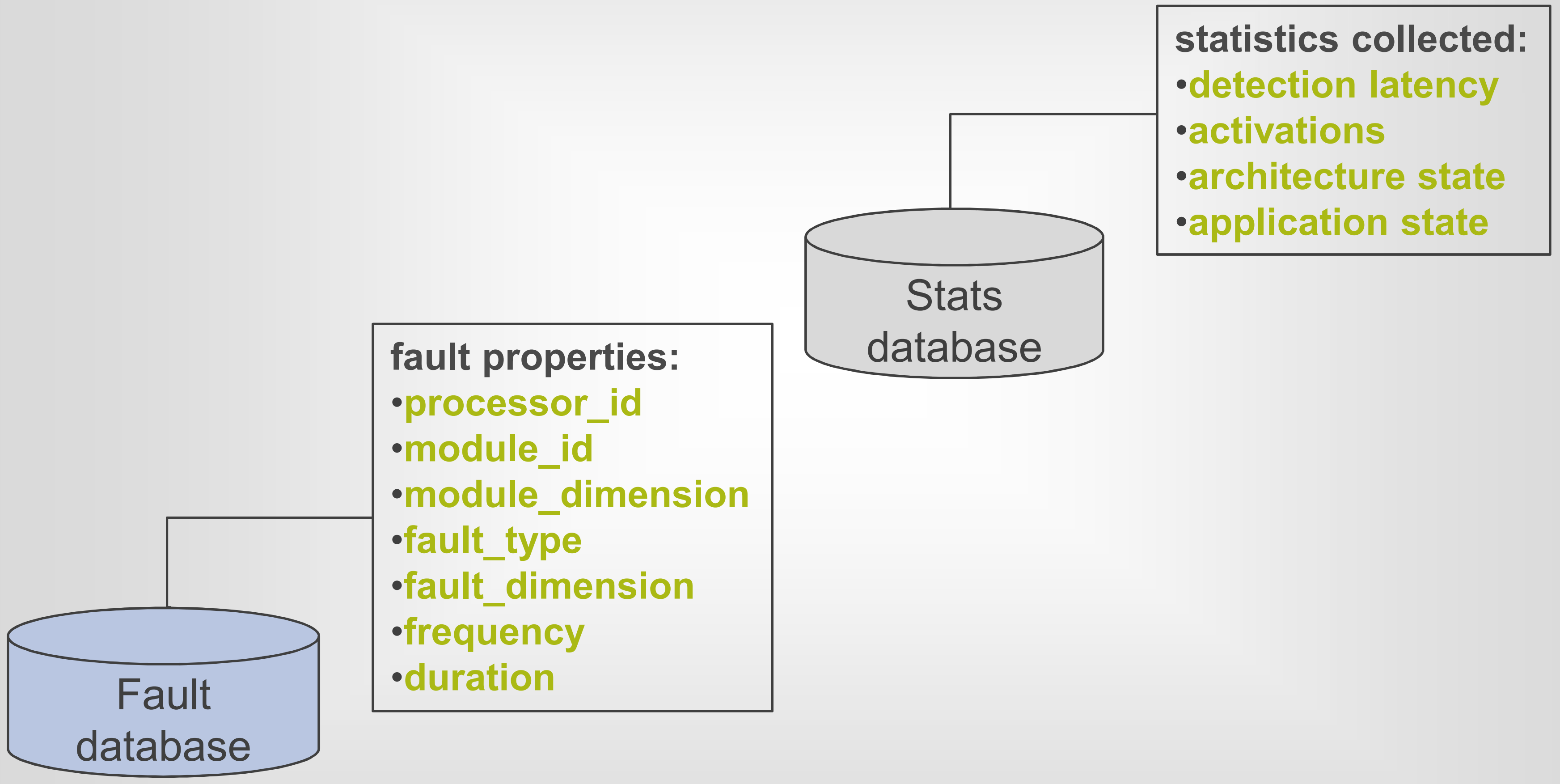
1. Built on **x86 OoO full system** microprocessor model (MARSSx86)
2. We extended **cache memories with data arrays**
3. Models **transient**, **intermittent**, **permanent**, and **multi-bit** faults
4. Traces **fault propagation till higher level of system stack**

Fault injection framework – Block diagram

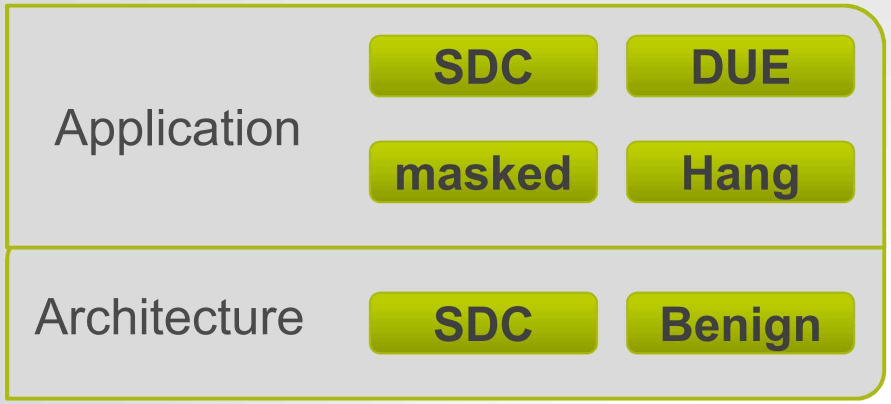


- **Offline part** (**simulation throughput unaffected**):
  - **Fault Generator**: populates the fault mask database
  - **Simulation Controller**: configures MARSSx86 based on user defined parameters, launches the fault injection run and collects the experimental results

Fault injection framework – Internals



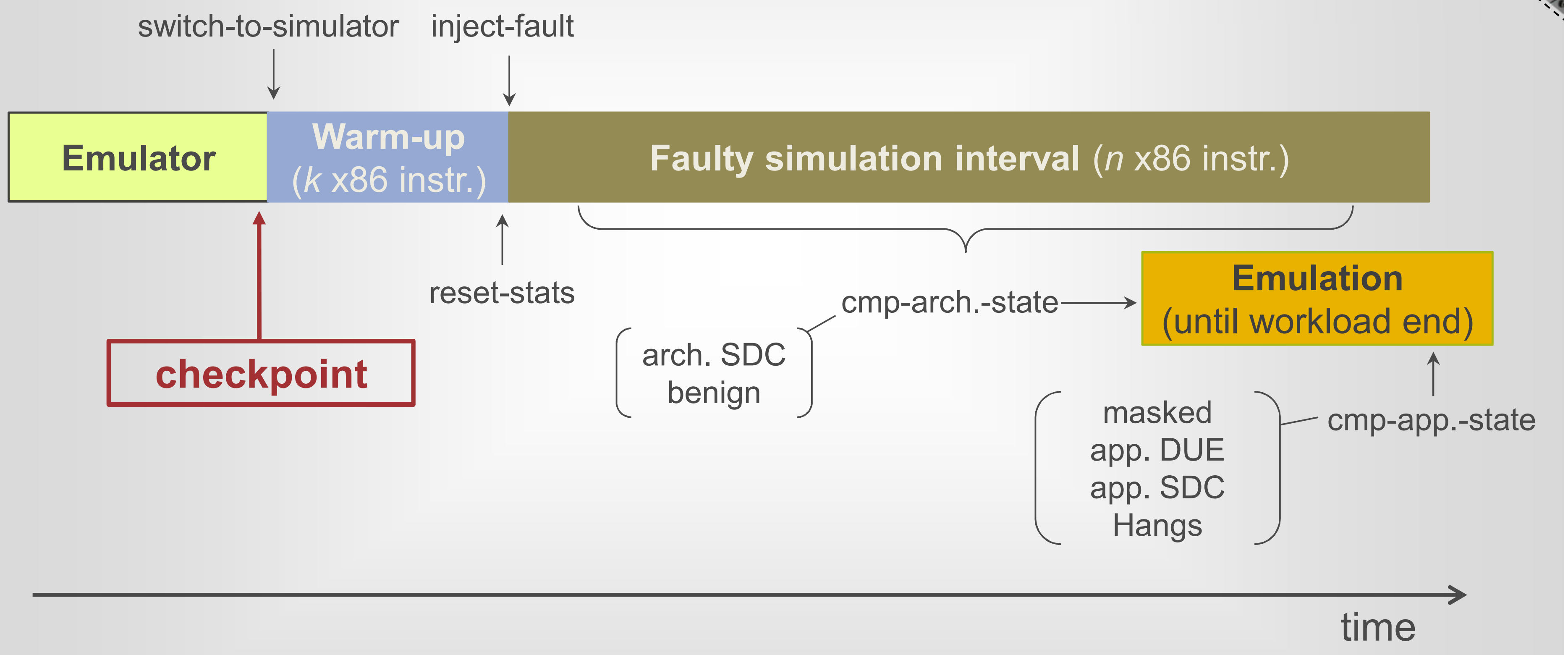
Fault injection framework – Fault classes



- **Architecture-level**:
  - **SDC**: silent data corruption in architectural state (PC and Register File)
  - **Benign**: architecture state of the fault-injected run equals to fault-free execution

- **Application-level**:
  - **SDC**: silent data corruption in application output
  - **DUE (detected unrecoverable error)**: unexpected exception, assertion, deadlock, interrupt, crash. False and True DUE are also included
  - **Masked**: the workload is executed completed without output mismatch
  - **Hangs**: application does not terminate within a reasonable time interval (3x the fault-free execution time)

Fault injection framework – Simulation timeline



Experimental Setup

- **Host machine configuration**:
  - Intel i7-3970X CPU clocked at 3.5 GHz
  - 32 GBytes of RAM
  - Ubuntu 12.04.4 LTS operating system
- **Enhanced model functionalities**:
  - Cache memories are enhanced with data arrays
  - Fault injection handler
  - Statistics handler

x86 microprocessor model

Arrays occupy the largest portion of chip's area

Parameter	Setting
Pipeline depth	24 (max branches in-flight)
Fetch/Issue/Commit	4/4/4 instructions per cycle
Return Address Stack	16 entries
Branch Target Buffer	4KB (4-way set associative, 1K entries)
Combined Predictor	16KB (64K entries, 2 bits per entry, 16 bits BHR) meta predictor table: 64K entries
Issue Queue	16 entries (one per cluster)
Reorder Buffer	128 entries
Functional Units	4 clusters (ALUs: 4 INT, 4 FPU)
L1 instruction cache	64KB (64B line, 512 sets, 2-ways, 2 cycles latency, wt)
L1 data cache	64KB (64B line, 512 sets, 2-ways, 2 cycles latency, wt)
L2 cache	2MB inclusive (64B line, 8-ways, 5 cycles latency, wt)
Main memory	512MB(50 nsec latency)

Memory intensive benchmarks

1. VectorAdd$_1$:

   Adds two arrays consisting of 10,000 integers → Store sum in a hard-disk file at the end of the calculations

2. VectorAdd$_2$:

   Adds two arrays consisting of 10,000 integers → Store sum in a hard-disk file after each individual addition

3. Bubblesort:

   Sorts 1000 integer elements → Store result in a hard-disk file at the end of the calculations

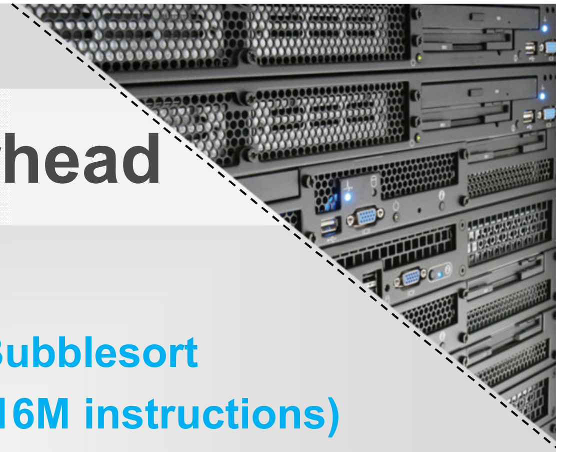
4. mMul:

   Multiplies two arrays (of 100x100 integers) → Store product in a hard-disk file at the end of the calculations

Clereco
Cross-Layer Early Reliability Evaluation for the Computing cOntinuum

cal@di

Experimental results – Framework overhead



		Original Model (MIPS)	Enhanced Model (MIPS)	Slowdown
Throughput of an experiment	**Transient Fault Injections**			
	No fault*	0.0484 (330.4 sec)	0.0477 (335.2 sec)	**1.5%**
	1-fault	-	0.0470 (340.2 sec)	**2.9%**
	5-faults	-	0.0469 (340.9 sec)	**3.1%**
	10-faults	-	0.0468 (341.7 sec)	**3.3%**
	Permanent Fault Injections			
	1-fault	-	0.0467 (342.4 sec)	**3.5%**
	5-faults	-	0.0461 (346.8 sec)	**4.7%**
	10-faults	-	0.0456 (350.6 sec)	**5.8%**

- **Bubblesort (16M instructions)**
- 1, 5, and 10 **transient** and **permanent** faults into the **L1 data cache**
- **Key insights**:
  1. Enhanced model simulation overhead **is up to 5.8%** (1.5% from "No fault" model and 4.3% due to injection framework)
  2. Simulation throughput remains almost **constant**
  3. **Low** overhead in multiple transient faults

"No fault": fault-free simulation (measure overhead induced only from the integration of the data arrays into the cache memory hierarchy)

Experimental results – Fault Injection

- **Single-bit transient faults**
  - randomly injected in L1 data cache (data array)
  - randomly selected clock cycle
- **Fault injection experiments**:

	VectorADD_1	VectorADD_2	mMul	BubbleSort	Total number of injections
#faults	125	664	336	140	**1265**

- **Checkpoint** (after application warm-up)
- **End-to-end execution**
- Compare **application state**

Clereco
Cross-Layer Early Reliability Evaluation for the Computing cOntinuum
cal@di

Reliability evaluation – L1 Data cache

Fault Category	VectorADD$_1$	VectorADD$_2$	mMul	BubbleSort
App. SDC	2.4%	0.6%	47.5%	7.8%
App. DUE	0.8%	0.2%	0.6%	0.0%
Hang	17.4%	0.3%	0.3%	0.7%
Masked	79.4%	98.9%	51.6%	91.5%

- **Application dependent**
  - internal structure of the benchmark affects classification ("App. SDC" of **VectorAdd$_1$** and **mMul**)
  - residency of elements in the L1 cache affects classification (**mMul**)
- **Lower SDC rate** on **VectorADD$_1$**, **VectorADD$_2$**, and **BubbleSort** compared to analytical methods

Conclusion

- **Architecture-level fault injection framework** for early reliability evaluation
- Models **fault propagation till higher level** of system stack
- Supports **all faults types** (transient, intermittent, permanent, multi-bit)
- Simulation overhead **is up to 5.8%**
  - 1.5% from “No fault” model and 4.3% due to injection framework
- Simulation throughput remains almost **constant** regardless of the fault number

Thank You!

Questions?