# Software Fault Tolerance

## 18-849b Dependable Embedded Systems


## Chris Inacio


## February 11, 1999

**Required Reading:** Flame War

**Best Tutorial:** Chapter 2 Software Fault Tolerance

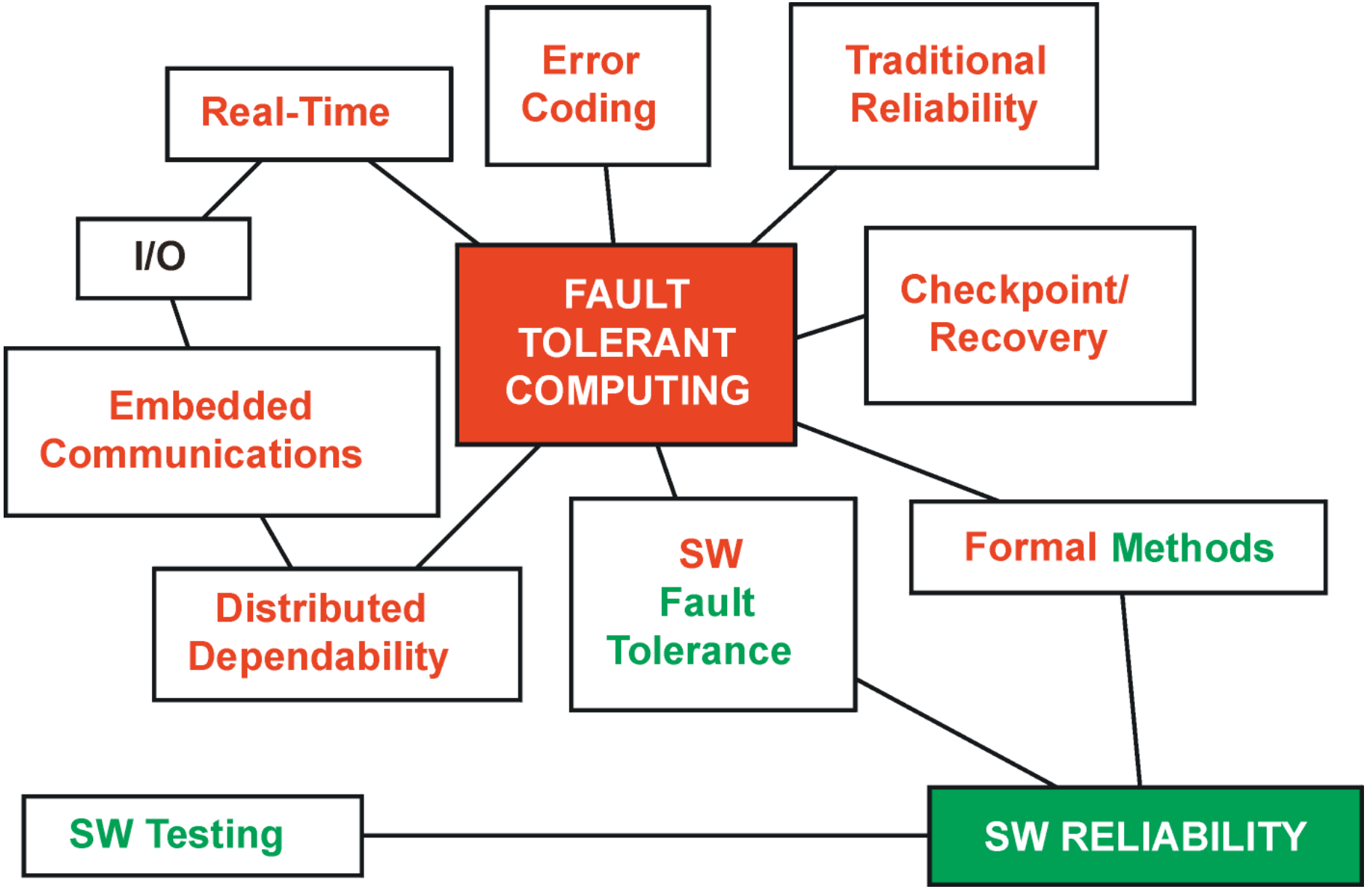
**Authoritative Books:** Software Fault Tolerance, Ed: Lyu


Carnegie Mellon

# Overview: Software Fault Tolerance

- **Introduction**
  - Clusters: Fault Tolerant Computing and Software Reliability
- **Key concepts**
  - Source of errors
  - Based on traditional hardware fault tolerance
  - Very immature field
- **Tools / techniques / metrics**
  - Recovery Blocks, N-Version Programming, Self-Checking Software
  - Metrics and methods in this area are very immature
- **Relationship to other topics**
  - List “surrounding” topics
- **Conclusions & future work**

# YOU ARE HERE MAP



# What is Software Fault Tolerance

- **Fault Tolerance - how to provide, by redundancy, service complying with the specification in spite of faults having occurred or occurring. (Laprie 96)**
  - Software Fault Tolerance - how to provide service complying with the specification in spite of faults
- **Key Concepts**
  - Software Faults are design errors
  - Software Fault Tolerance based on hardware fault tolerance
  - N-version Programming
  - Recovery Blocks
  - Checkpoint and Recovery
  - Design Diversity

# Design Errors

- **Software errors originate from design faults**
  - programmer mistakes
  - misinterpreted specification
- **Hardware errors can originate from design., environment, etc.**
  - Hardware errors mostly from manufacturing
- **Does software have to have bugs?**
- **Does design faults only represent unique problem**

# SW FT based on HW FT

- **Software fails due to  design faults**
- **Hardware fault tolerance based on manufacturing faults**
- **N-version software compares to N-way redundant hardware?**
- **Problems with this approach?**
- **Self-checking software is more common in practice**
  - How fault tolerant is self-checking software?
- **Fault tolerant software is based on multiple versions and design diversity**

# Not Ready For Prime Time

- **Recovery Block is almost anecdotal**
- **N-Version programming is weighed down in disagreement**
- **Can self-checking recover from unexpected faults?**
- **Need new directions and new thoughts**
- **Do any of these method work for non-hardware non-transient faults?**
- **Might solve some multi-processor inconsistency problems**

# Tools / Techniques

- **Tools are non-existent**
- **Methods:**
  - N-Version software
    - multiple implementations of the same specification
    - possibly in different languages
    - pray for non-correlated errors in the software
    - requires design diversity --- pushes problem up to specification level
  - Recovery Blocks
    - put a consistency check at the end of a block
    - make sure the answer “makes sense”
    - retry the block if it doesn’t work!
    - Solves transient failures

# Tools / Techniques

- **Methods Continued**
  - Self-Checking software
    - uses multiple versions to do self checking of results
    - mentioned by Laprie but not described in the literature

- **Failure Detection**
  - Detecting the failure is a challenge
  - Many faults are latent way the fault actually occurs
  - Latent faults show up (a lot) later
  - Can use a watchdog to figure out if the “program” crashed

# Metrics

- **Software fault metrics**
  - Metrics for software errors:
    - best metrics may be from Ballista project
    - other metrics have horrible data sets
  - Same issue with software models
    - models have poor prediction of faults
    - recommended that you only use the lower bound in the model
  - Field data is this area is bad
    - Tandem has some good data about their Non-Stop systems, but limited applicability
    - proves one point: good software is possible

# Relationship To Other Topic Areas

- **Fault Tolerance**
  - This is a subtopic of fault tolerance
  - estimated that 60-90% of current computer errors are from software
- **Ultra Fault Tolerant**
  - Needs Software fault tolerance to work
  - Probably not going to happen soon
- **Hardware Fault Tolerance**
  - currently based on hardware fault tolerance
  - needs to be able to withstand some small amount of hardware faults
  - may need to interact with hardware for hardware fault tolerant
- **Software Methodology plays big role**

# Conclusions & Future Work

- **This area is very immature**
  - Mostly it doesn't work
  - Can solve some transient faults
- **Engineering Tradeoffs**
  - expensive to develop fault tolerant software
  - recovery blocks may be slow since they are serial re-execute
- **The methods proposed so far are based on hardware fault tolerance**
- **Currently there seems to be no method to really guarantee fault tolerance**
- **Does software really have to have bugs?**
  - How do Tandem and Stratus get it "right"?
  - IBM doesn't do too bad either

# Flame War: N-Version Software FT

- **"This sentence is what is cool about this paper" -- it is a flame war!!**
- **N-Version software**
  - replicates N-way redundant hardware
  - requires design diversity, possibly from the spec
  - but software errors are correlated
- **Does N-version software fault tolerance work?**
  - Questionable at best
  - I wouldn't want to bet my life on it!