16
Fault Tolerance

18-540 Distributed Embedded Systems

Philip Koopman

November 8, 2000

Required Reading: **Nelson, Fault-tolerant computing: fundamental concepts**

Carnegie Mellon

Assignments

- **Next lecture read about critical systems.**
- **Project part #5 due Wednesday 11/15**
- **Next homework is #8, due Friday 11/17**

Where Are We Now?

- **Where we've been:**
  - Distributed architecture (1st course section)
  - Distributed networks (2nd course section)
- **Where we're going today:**
  - Making correct, robust systems
    - Today: fault tolerance / reliability
- **Where we're going next:**
  - Critical systems (software, distributed system issues)
  - Validation/certification
  - Design methodology
  - Miscellaneous & advanced topics

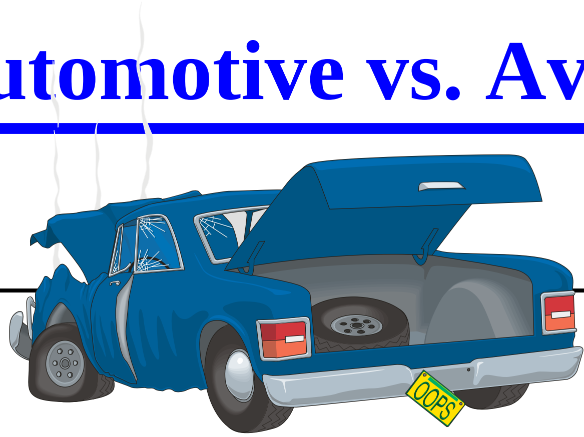
Preview

- **Aerospace approaches don’t necessarily work on consumer products**
  - Automobiles as an example
- **How and why things break**
  - Mechanical
  - Hardware
  - Software
- **Designing systems for failure detection & recovery**
  - Practical limits of fault tolerant design
  - Environment & other sources of problems
  - How to (and not to) design a highly available system

Why Not Build Cars Like Aircraft?

- **We all "know" that flying is safer than driving**
  - (This is only true per mile, not per hour)
- **So, use commercial aircraft techniques to build automated vehicles**
  - Computer-controlled navigation & tactical maneuvers
  - Redundant hardware
  - Near-perfect software
  - High-quality design and components
  - Highly trained professional operators (oops…)

Automotive vs. Aviation Safety



	U.S. Automobiles	U.S. Commercial Aircraft
Deployed Units	~100,000,000	~10,000
Operating hours/year	~30,000 Million	~55 Million
Cost per vehicle	~$20,000	~$65 Million
Mortalities/year	42,000	~350
Accidents/year	21 Million	170
Mortalities / Million Hours	0.71	6.4
Operator Training	Low	High
Redundancy Levels	Brakes only	All flight-critical systems

- Aviation autopilot is probably easier than an automotive autopilot

Why Not Aerospace Approaches For Cars?

- **Based on culture of redundant HW, perfect SW**
- **Too expensive**
  - Component "Pain threshold" for vehicles is at the $.05 level
  - Higher levels of cost OK for Europe if they provide performance value
- **Different operating environment/reaction time**
- **Difficult to enforce maintenance**
  - People run out of gas & engine oil; ignore "idiot lights"
  - Aircraft don't leave gate if something is broken
  - End-of-life wearout -- old vehicles stay on the road
  - Can we ensure same maintenance quality?
- **Poorly trained operators**
  - Yearly driver exam with road test?
  - Required simulator time for accident response?

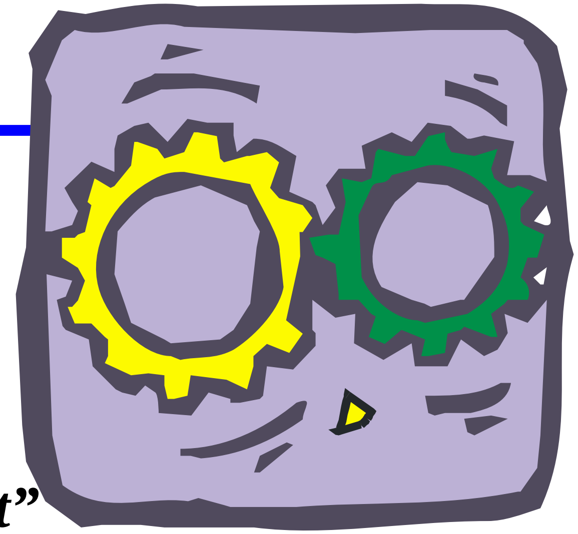
Definitions

- **RELIABILITY -- Aerospace model**
  - Survival probability for given “mission time”
  - Good when repair is difficult

- **AVAILABILITY -- Automotive & general purpose computing model**
  - The fraction of time a system meets its specification
  - Good when continuous service is important

- **DEPENDABILITY**
  - Generalization: system does the right thing at the right time

Generic Sources of Faults



- **Mechanical -- "*wears out*"**
  - Deterioration: wear, fatigue, corrosion
  - Shock: fractures, stiction, overload
- **Electronic Hardware -- "*bad fabrication; wears out*"**
  - Latent manufacturing defects
  - Operating environment: noise, heat, ESD, electro-migration
  - Design defects (*e.g.*, Pentium FDIV bug)
- **Software -- "*bad design*"**
  - Design defects
  - "Code rot" -- accumulated run-time faults
- **People**
  - Takes a whole additional page...

Errors By Development Phase

STAGE	ERROR SOURCES	ERROR DETECTION STRATEGY
Specification & design	**Algorithm Design** **Formal Specification**	**Simulation** **Consistency checks**
Prototype	**Algorithm design** **Wiring & assembly** **Timing** **Component Failure**	**Stimulus/response** **Testing**
Manufacture	**Wiring & assembly** **Component failure**	**System testing** **Diagnostics**
Installation	**Assembly** **Component failure**	**System Testing** **Diagnostics**
Field Operation	**Component failure** **Operator errors** **Environmental factors**	**Diagnostics**

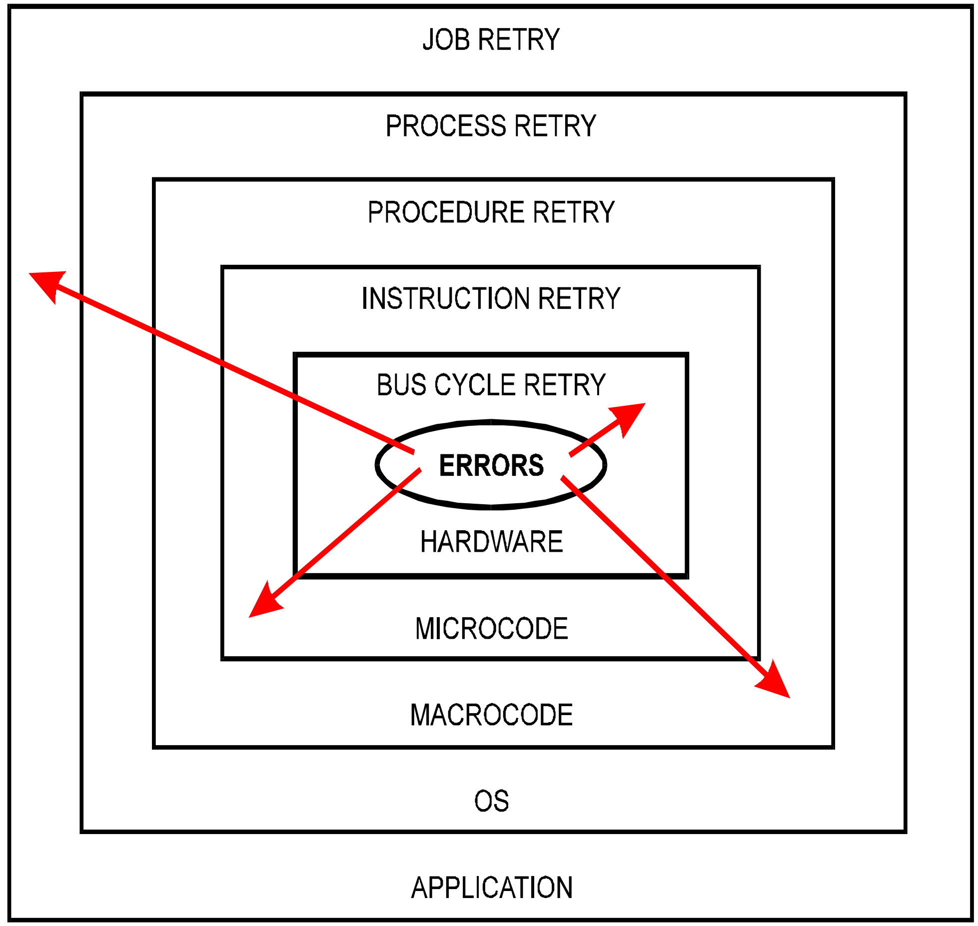
Fault Classification

- **Duration**
  - Transient -- design flaws, environmental factors, *etc.*
  - Intermittent -- recurring events
  - Permanent -- “hard” failures/replace component -- only 10% of problems

- **Extent**
  - Local (independent)
  - Distributed (related)

- **Value**
  - Determinate (stuck-at-high or -low)
  - Indeterminate (varying values)

Error Containment Levels



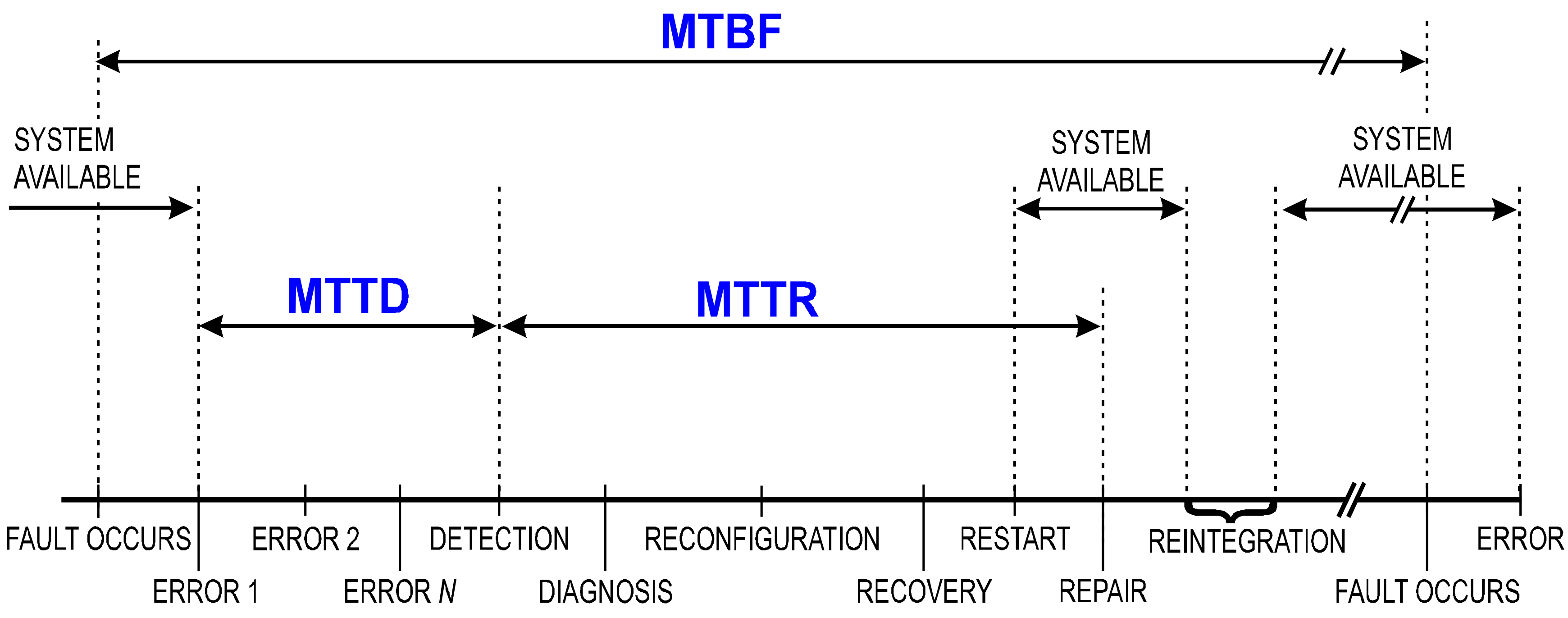
The further out the error propagates, the more state is involved and the more diverse error manifestations becomes, resulting in more complex error recovery.

Basic Steps in Fault Handling

- **Fault Confinement --** contain it before it can spread
- **Fault Detection --** find out about it to prevent acting on bad data
- **Fault Masking --** mask effects
- **Retry --** since most problems are transient, just try again
- **Diagnosis --** figure out what went wrong as prelude to correction
- **Reconfiguration --** work around a defective component
- **Recovery --** resume operation after reconfiguration in degraded mode
- **Restart --** re-initialize (warm restart; cold restart)
- **Repair --** repair defective component
- **Reintegration --** after repair, go from degraded to full operation

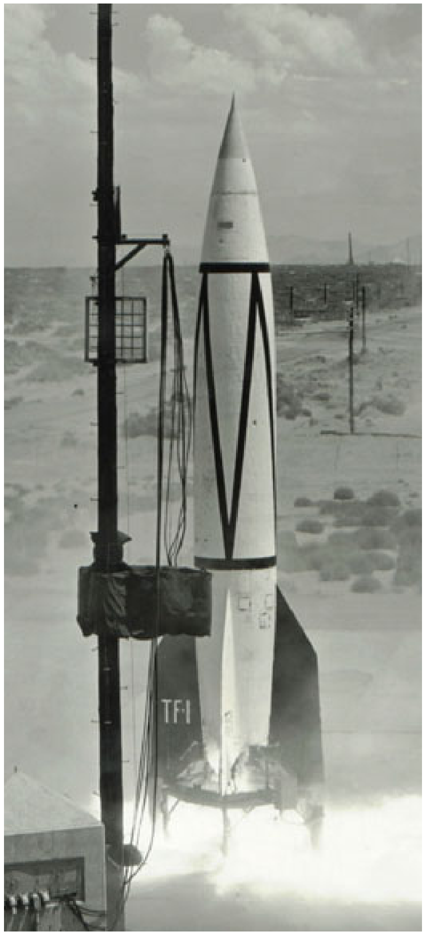
MTBF -- MTTD -- MTTR

$$\text{Availability} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

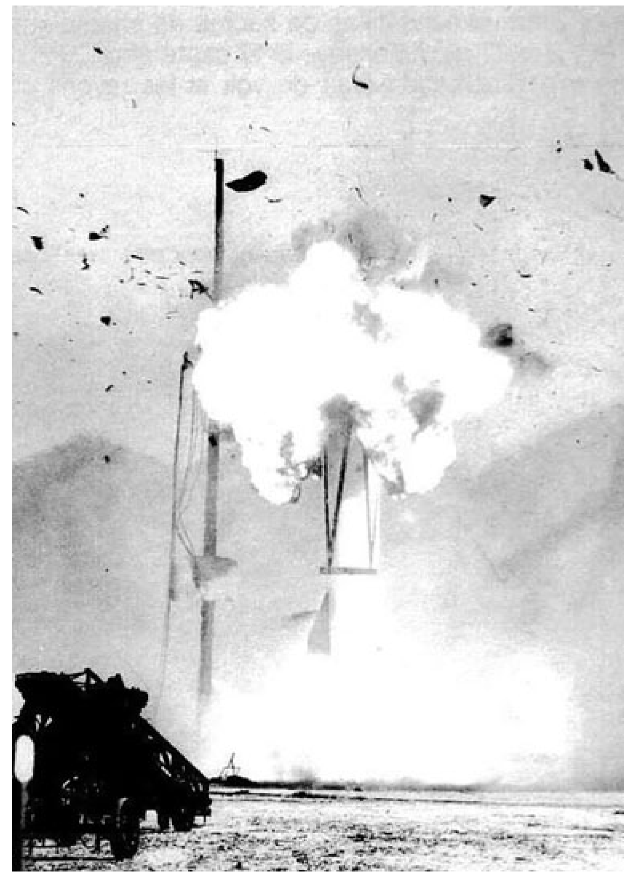


A Scenario for on-line detection and off-line repair. The measures -- MTBF, MTTD, and MTTR are the average times to failure, to detection, and to repair.

A Brief History of Reliability Theory

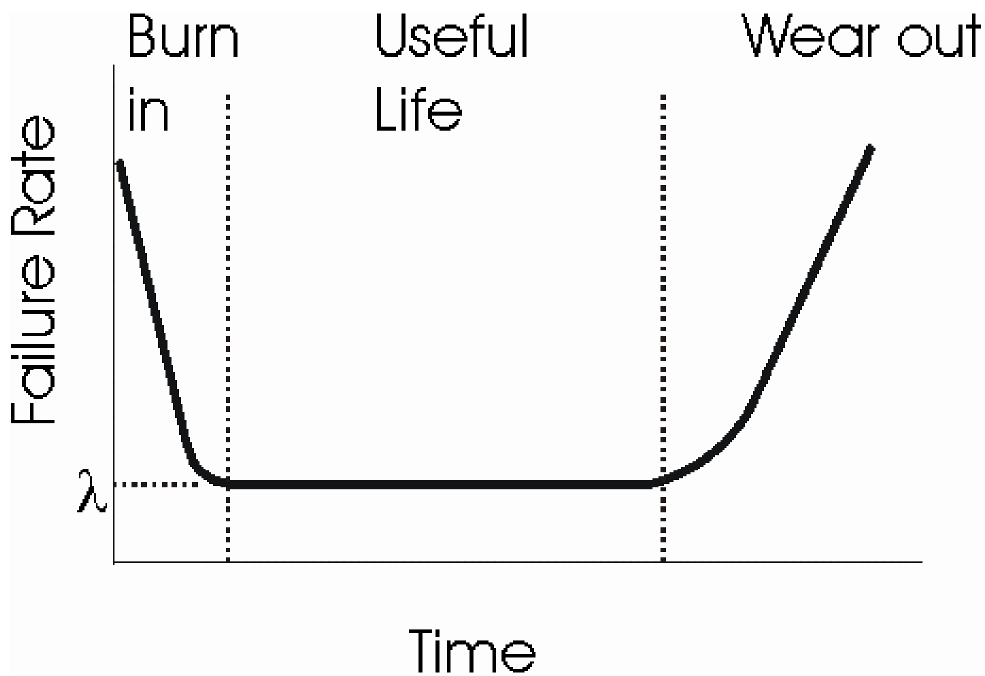


- **Electronics reliability theory was invented in WWII**
  - For V-2 German rocket
  - For Radar/electronics
- **Problem: *<u>Misleading</u>* mechanical analogy:**
  - "Chain is as strong as its weakest link"
    - Example: chain across Hudson River in revolutionary war
  - Assumes failures based only on over-stress and aging effects
  - Works for mechanical components, not electronic components
  - V-2 rockets kept blowing up!



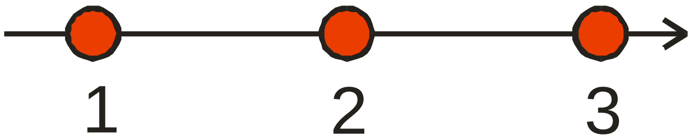
“Modern” Reliability Theory

- **Electronics reality:**
  - Failures are *RANDOM*, with time-varying mean failure rates
  - Even if there is no over-stress, electronic components will fail all the time
    - Result: V2 rocket was unreliable even after improving weak components
  - Solution: move to a probabilistic view of reliability
    - And assume that failure rates are constant during “useful life”
  - Reliability *R(t)* is probability system is working at time *t.*
    - Reliability for N hours = N * l



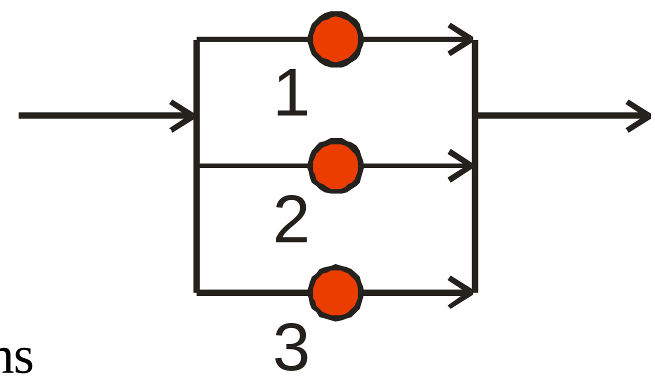
Parallel & Serial Reliability

- **Serial reliability: compute probability of failure-free operation**
  - All components need to operate for system to operate



  - $R(t) = R_1(t) * R_2(t) * R_3(t)$
    - This is probability that all components work

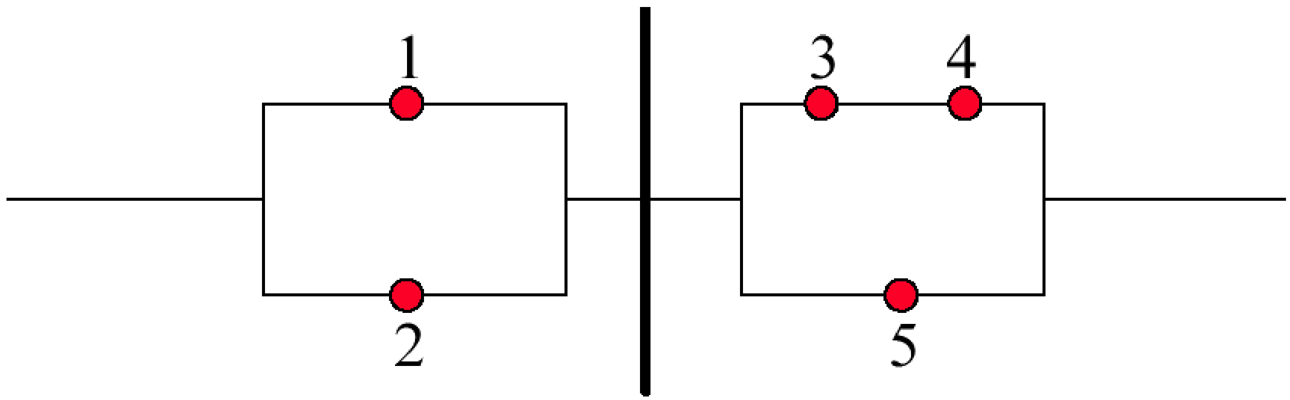
- **Parallel reliability**
  - Simple version -- assume only 1 of N components needs to operate



  - $R(t) = 1 – [(1-R_1(t)) * (1-R_2(t)) * (1-R_3(t))]$
    - This is 1 - Probability that all components fail
  - More complex math used for M of N subsystems
  - There may also be a "voter" that counts for a serial reliability element!

Combination Serial/Parallel Systems

- **Recursively apply parallel/serial equations to subsystems**



Total reliability is the reliability of the first half, in serial with the second half.

Given that R1=.9, R2=.9, R3=.99, R4=.99, R5=.87

$$Rt=[1-(1-.9)(1-.9)][1-(1-.87)(1-(.99*.99))] = .987$$

Uses of Redundancy

- **M of N subsystems need to be working**
  - Assume others “fail fast / fail silent”
  - Example: brakes on a car

- **M of N systems are compared for correctness**
  - Uses special (“failure-proof”) voting circuit; majority rules
  - 2 of 3 is “Triplex Modular Redundancy” (TMR)
    - If any 2 units agree, use that result
    - Any incorrect unit is masked

Post-Modern Reliability Theory

- **Pre-WWII: mechanical reliability / "weakest link"**
- **"Modern" reliability: hardware dominates / "random failures"**

- **But, software matters! ("post-modern" reliability theory)**
  - Several schools of thought; not a mature area yet
  - Still mostly ignores people as a component in the system

1) Assume software never fails
   - Traditional aerospace approach; bring lots of $$$$ and cross your fingers

2) Assume software fails randomly just like electronics
   - May work on large server farms with staggered system reboots
   - Doesn't work with correlated failures -- "packet from Hell" / date rollover

3) Use software diversity analogy to create M of N software redundancy
   - Might work at algorithm level
   - Questionable for general software
   - Pretty clearly does NOT work for operating systems, C libraries, etc.

4) Your Ph.D. thesis topic goes here: ________________

How Often Do Components Break?

- **Failure rates often expressed in failures / million operating hours (“Lambda” l):**

Military Microprocessor	**0.022**
Automotive Microprocessor	**0.12 (1987 data)**
Electric Motor	**2.17**
Lead/Acid battery	**16.9**
Oil Pump	**37.3**
Human: single operator best case	**100 (per Mactions)**
Automotive Wiring Harness (luxury)	**775**
Human: crisis intervention	**300,000 (per Mactions)**

- **We have no clue how we should deal with software field reliability**
  - Best efforts at this point based on usage profile & field experience

Common Hardware Failures

- **Connectors**
  - Especially wiring harnesses that can be yanked
  - Especially if exposed to corrosive environments

- **Power supplies**
  - Especially on/off switches on PCs

“Mainframe” Outage Sources

	AT&T Switching System	Bellcore Commercial	Japanese Commercial Users	Tandem 1985	Tandem 1987	Northern Telecom	Mainframe Users
Hardware	0.20	0.26	0.75*	0.18	0.19	0.19	0.45
Software	0.15	0.30	0.75*	0.26	0.43	0.19	0.20
Maintenance	--	--	0.75*	0.25	0.13	--	0.05
Operations	0.65	0.44	0.11	0.17	0.13	0.33	0.15
Environment	--	--	0.13	0.14	0.12	0.15	0.15
Power	--	--	--	--	--	0.13	--

(* the sum of these sources was 0.75)

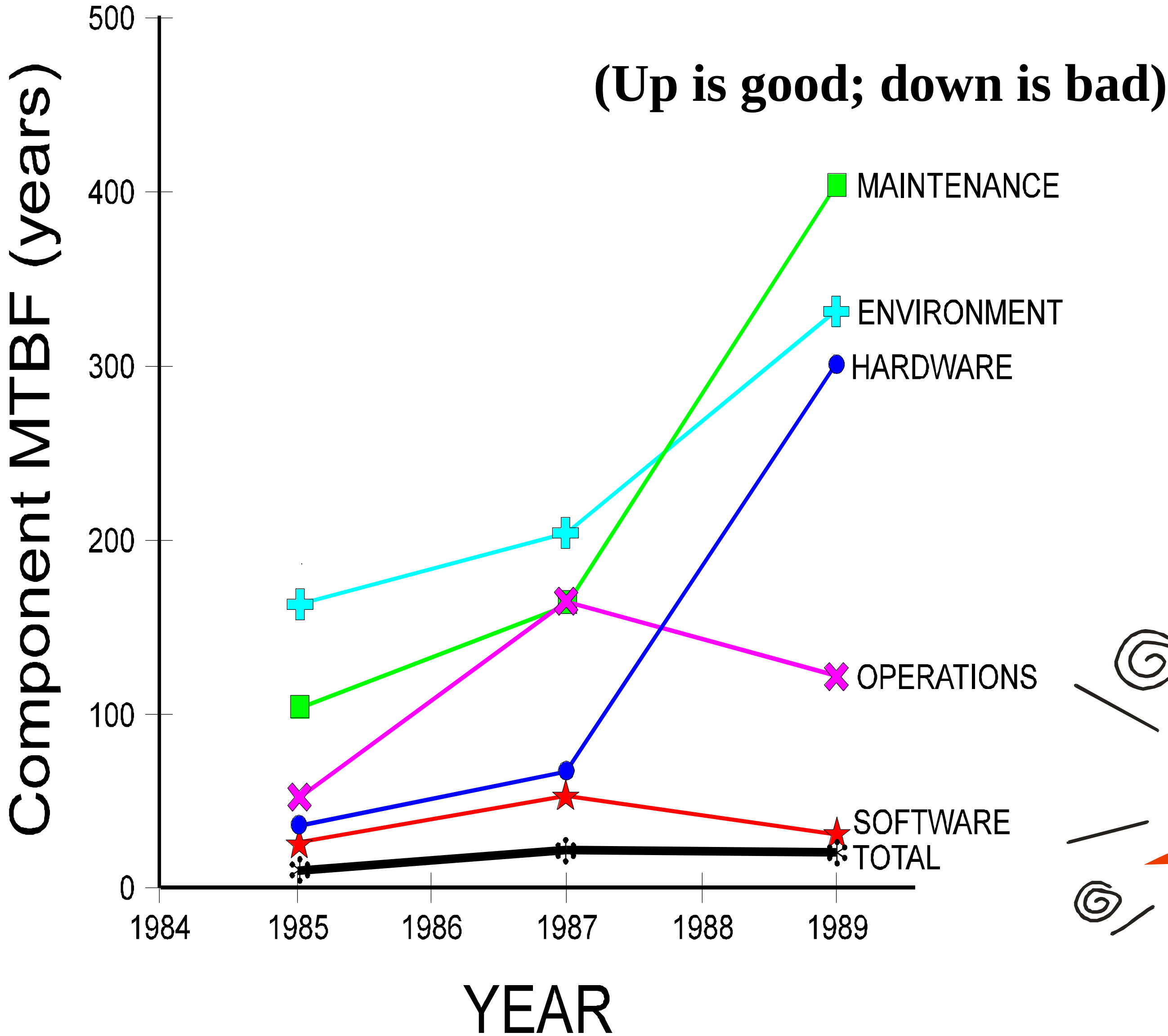
Tandem Environmental Outages

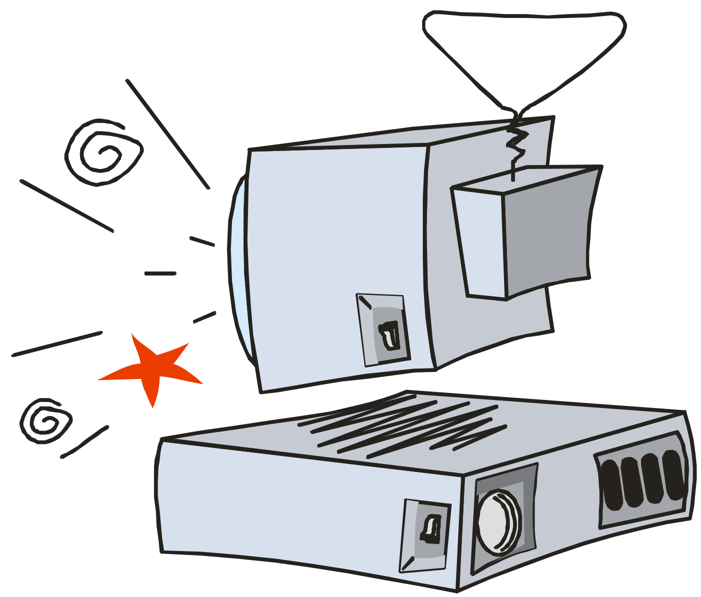
◆ Extended Power Loss	80%
◆ Earthquake	5%
◆ Flood	4%
◆ Fire	3%
◆ Lightning	3%
◆ Halon Activation	2%
◆ Air Conditioning	2%

- **Total MTBF about 20 years**
- **MTBAoG* about 100 years**
  - Roadside highway equipment will be more exposed than this

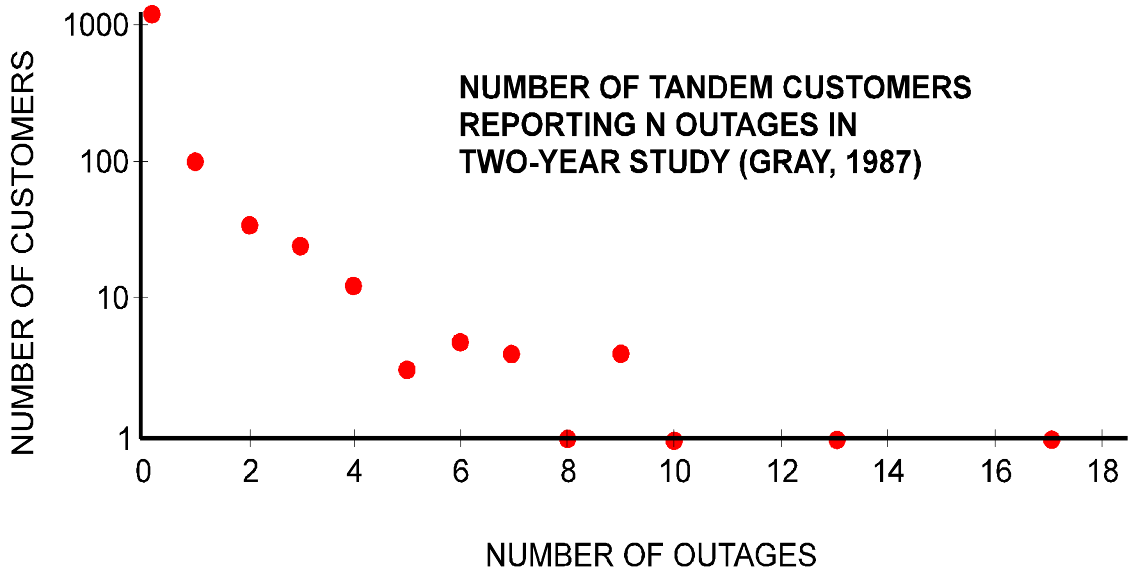
* (AoG= "Act Of God")

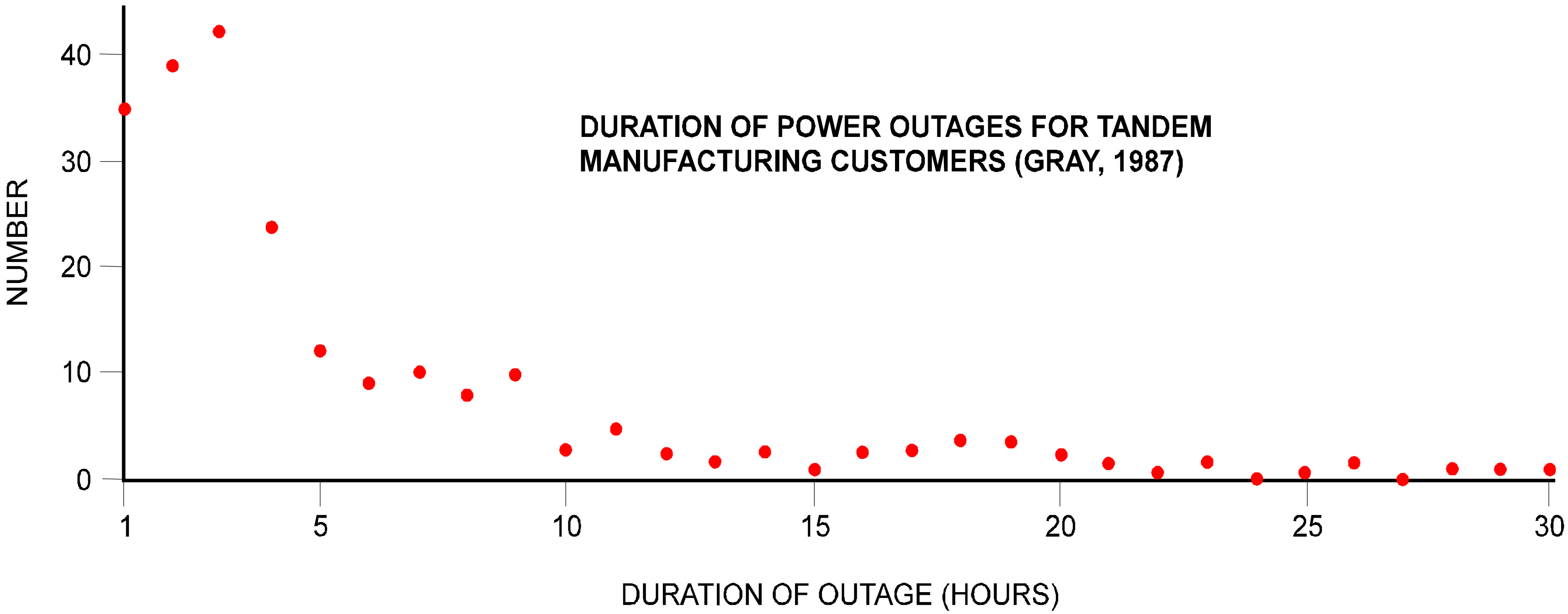
Tandem Causes of System Failures



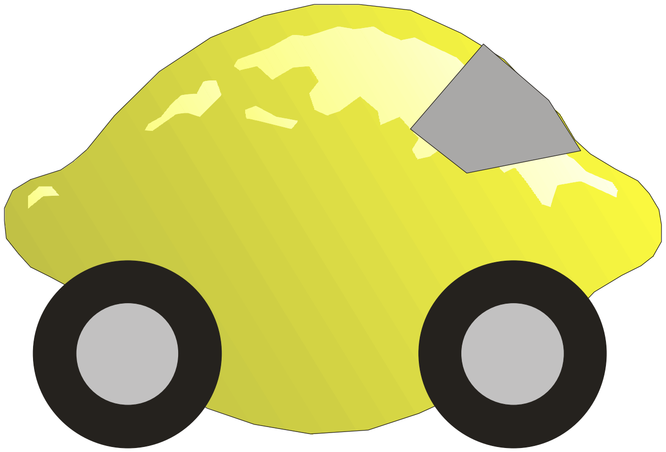


Tandem Outages





Lemons Or Just Statistics?



Poisson distributed failures:

$$p(x) = \frac{(lt)^x}{x!} e^{-lt} \quad x = 0, 1, 2, \ldots$$

Annual failures for 100,000,000 vehicles	Vehicles failing given 10 year MTBF	Vehicles failing given 100 year MTBF
0	90,483,741	99,004,983
1	9,048,374	990,050
2	452,419	4,950
3	15,081	17
4	377	0
5	8	0
6	0	0

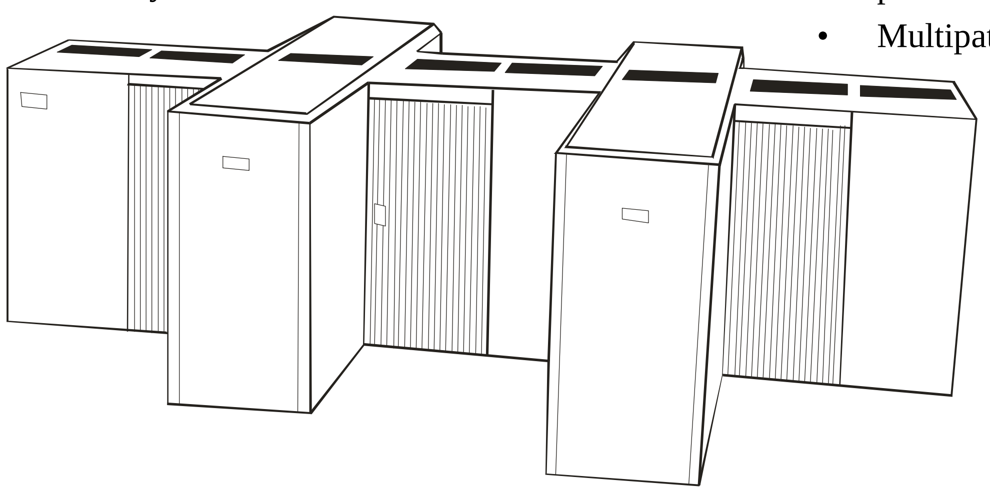
IBM 3090 Fault Tolerance Features

Reliability

- Low intrinsic failure rate technology
- Extensive component burn-in during manufacture
- Dual processor controller that incorporates switchover
- Dual 3370 Direct Access Storage units support switchover
- Multiple consoles for monitoring processor activity and for backup
- LSI Packaging vastly reduces number of circuit connections
- Internal machine power and temperature monitoring
- Chip sparing in memory replaces defective chips automatically

Availability

- Two or tour central processors
- Automatic error detection and correction in central and expanded storage
- Single bit error correction and double bit error detection in central storage
- Double bit error correction and triple bit error detection in expanded storage
- Storage deallocation in 4K-byte increments under system program control
- Ability to vary channels off line in one channel increments
- Instruction retry
- Channel command retry
- Error detection and fault isolation circuits provide improved recovery and serviceability
- Multipath I/O controllers and units



More IBM 3090 Fault Tolerance

- **Data Integrity**
  - Key controlled storage protection (store and fetch)
  - Critical address storage protection
  - Storage error checking and correction
  - Processor cache error handling
  - Parity and other internal error checking
  - Segment protection (S/370 mode)
  - Page protection (S/370 mode)
  - Clear reset of registers and main storage
  - Automatic Remote Support authorization
  - Block multiplexer channel command retry
  - Extensive I/O recovery by hardware and control programs

- **Serviceability**
  - Automatic fault isolation (analysis routines) concurrent with operation
  - Automatic remote support capability - auto call to IBM if authorized by customer
  - Automatic customer engineer and parts dispatching
  - Trade facilities
  - Error logout recording
  - Microcode update distribution via remote support facilities
  - Remote service console capability
  - Automatic validation tests after repair
  - Customer problem analysis facilities

IBM 308X/3090 Detection & Isolation

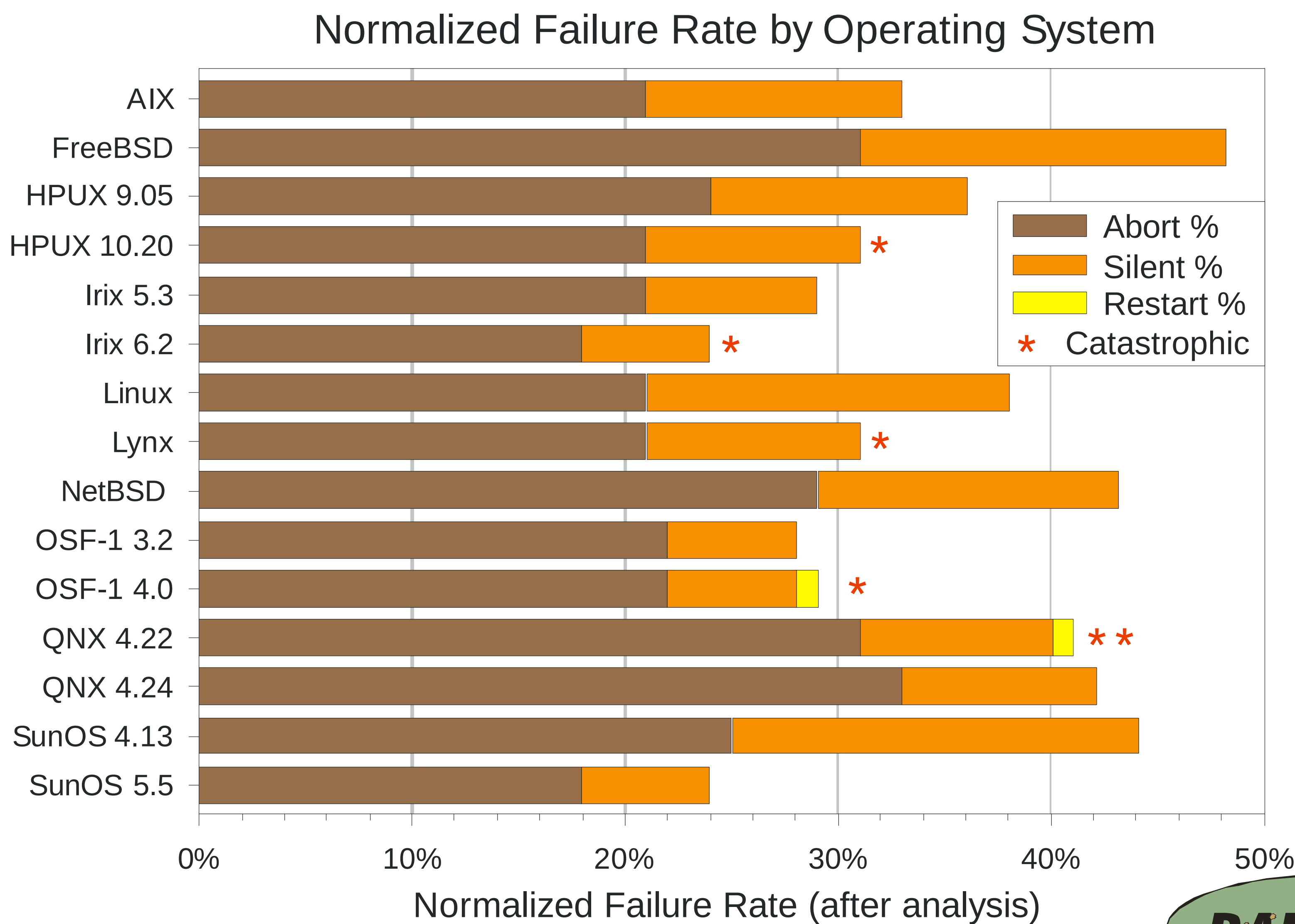
- **Hundreds of Thousands of isolation domains**
- **25% of IBM 3090 circuits for testability -- only covers 90% of all errors**
- **Assumed that only 25% of faults are permanent**
  - If less than two weeks between events, assume same intermittent source
  - Call service if 24 errors in 2 hours
- **(Tandem also has 90% FRU diagnosis accuracy)**

Approximate Consumer PC Hardware ED/FI

this space intentionally blank

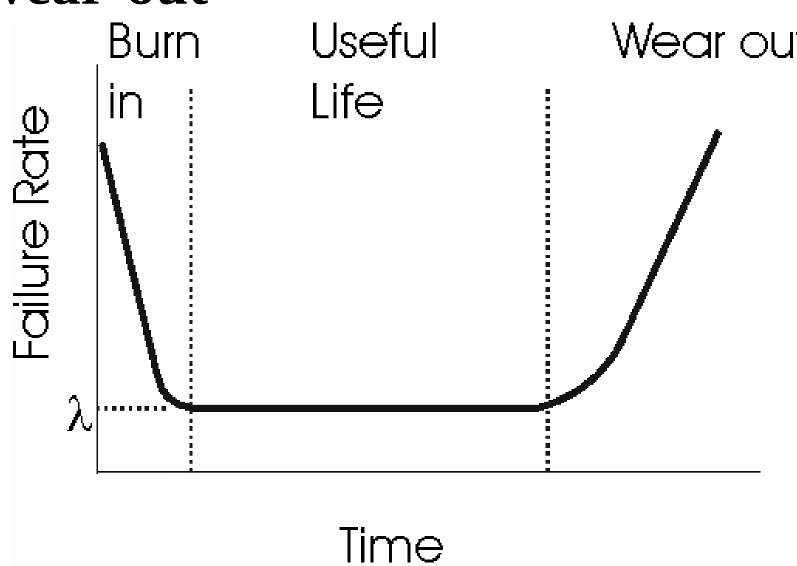
Typical Workstation Software ED/FI

- **SW Defects are inevitable -- what happens then?**



Research Challenges

- **Exploiting redundancy**
  - Hardware redundancy is easy, but that's not the main problem in many cases
  - Software redundancy is hard to ensure
- **Heterogeneous redundancy?**
  - Use "good-enough" techniques in emergencies
    - Car brakes for steering
    - Elevator brake for emergency egress
- **Equipment that reaches end-of-life wear-out**
  - Violates useful life assumptions, but happens in consumer products



- **Software**
  - "Reliability" doesn't even mean the same thing as used by the software community

Conclusions

- **Design reliability into the system, not on top of the system**
  - Take domain constraints into account when choosing approach
- **Historically, goals of 100% unattainable for:**
  - Fault detection/isolation
  - Availability
  - Design correctness
  - Isolation from environmental problems
- **The biggest risk items are people & software**
  - But we’re not very good at understanding software reliability
  - We understand people reliability, but it’s not very good