# Software Robustness and Graceful Degradation in Embedded Systems

May 8, 2000

**Philip Koopman**

koopman@cmu.edu

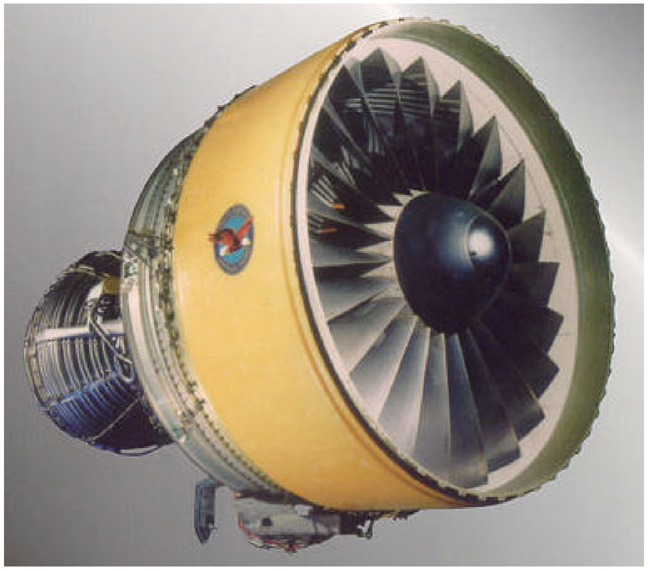
http://www.ices.cmu.edu/koopman

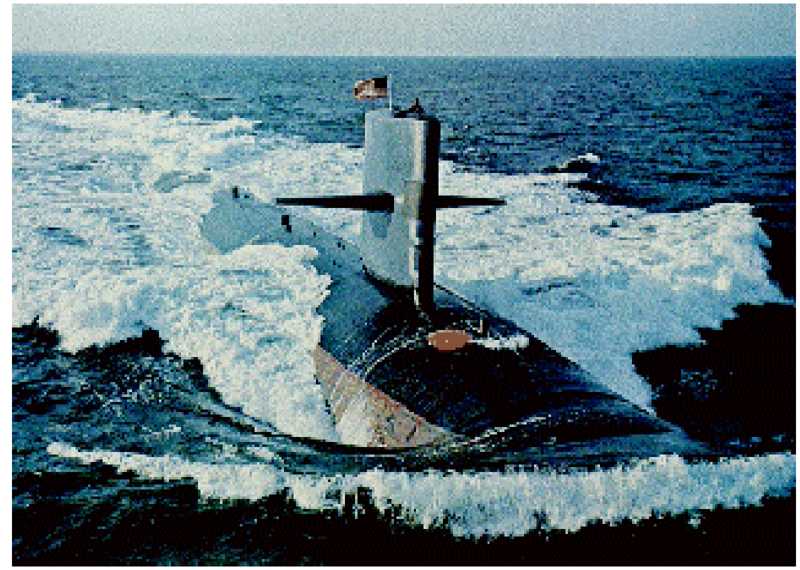
(412) 268-5225

Carnegie Mellon

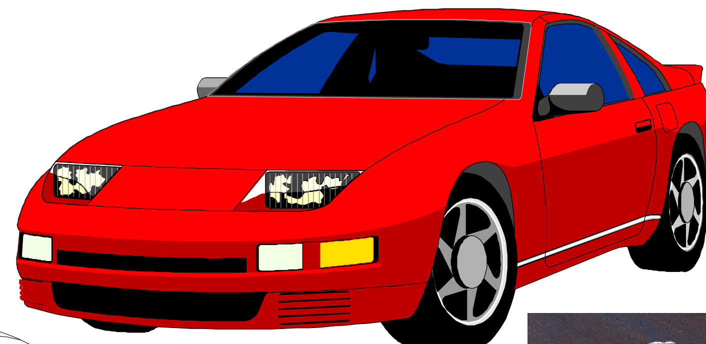
Institute for Complex Engineered Systems

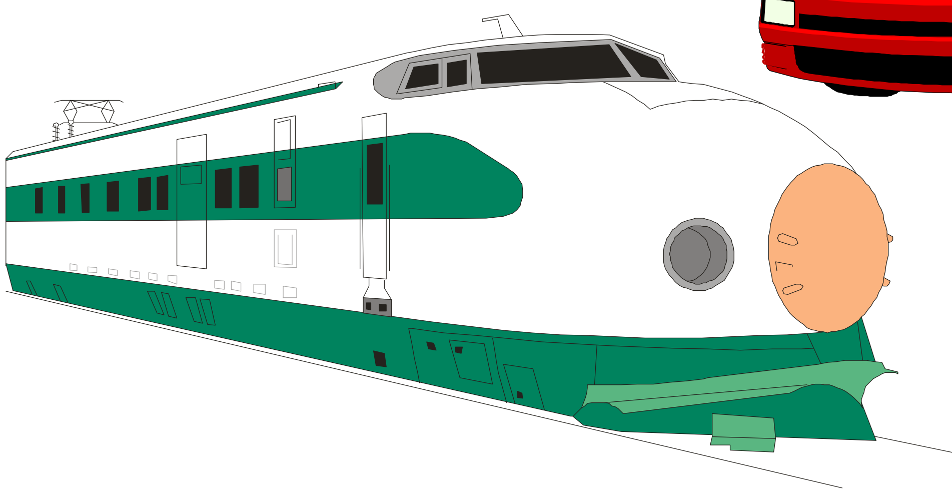
Electrical & Computer ENGINEERING





# Embedded System =
# *Computers Inside a Product*





# Outline of Talk

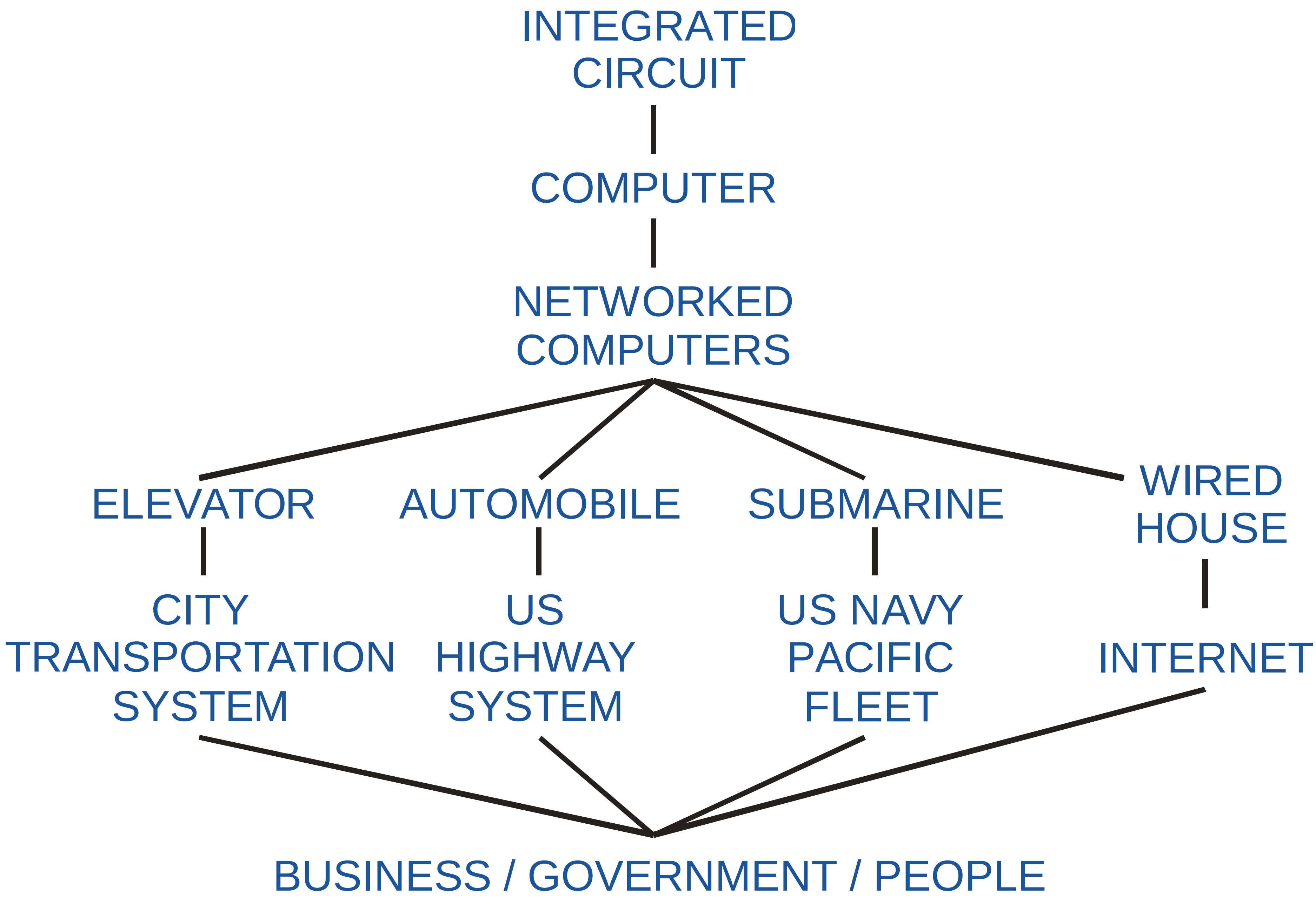
- **A personal trajectory through 4 areas in the embedded systems research space**

- **Previous research areas (what's past is prologue)**
  - #1: CPU design
  - #2: Hardware system synthesis

- **Latest research results (Ballista project)**
  - #3: Software robustness testing
  - *Can software components be made well behaved?*

- **Current research direction (RoSES project)**
  - #4: Graceful degradation/distributed embedded systems
  - Components aren't going to be well behaved –
    *Is automatic reconfiguration a silver bullet?*

# What's an Embedded *System*?

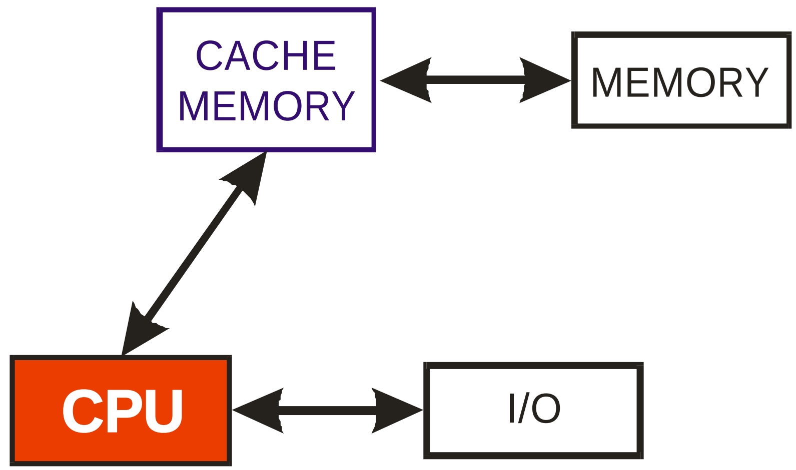


# Classical, General Purpose View of Computing

- **Undergrad – Juniors:**
  - Measured by Performance
  - SPECmarks

# Classical View of Computing (upper division)

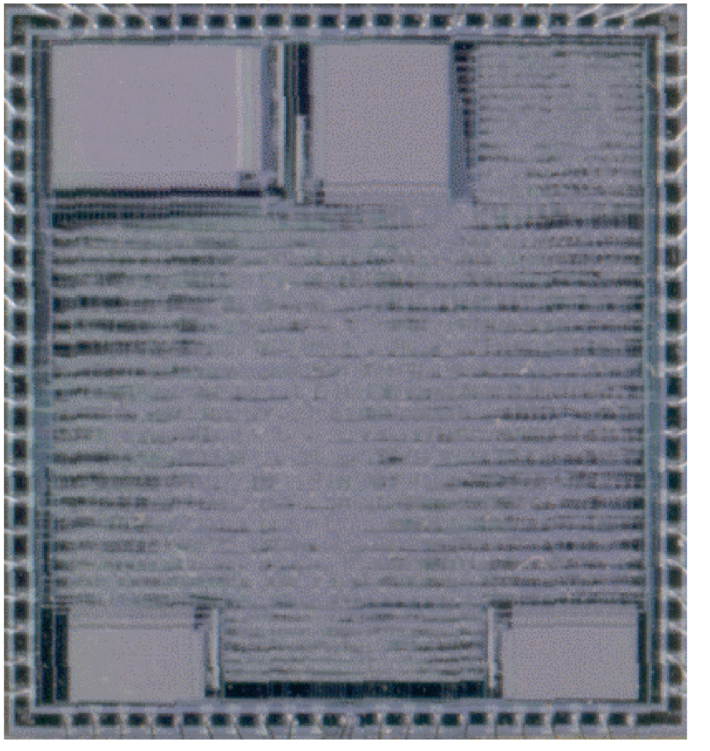
- **Undergrad – Seniors:**
  - Measured by: Performance, Cost
  - Compilers & OS considered too




- **Graduate Level:**
  - Advanced performance techniques
  - Distributed systems   (networks, storage)

# Area #1: CPU design for embedded systems

- **Let's build a CPU that's optimized for embedded systems!**
  - Special-purpose instruction set
  - Special-purpose hardware accelerators
  - Optimized for small memory footprint
  - *etc., etc., etc.*

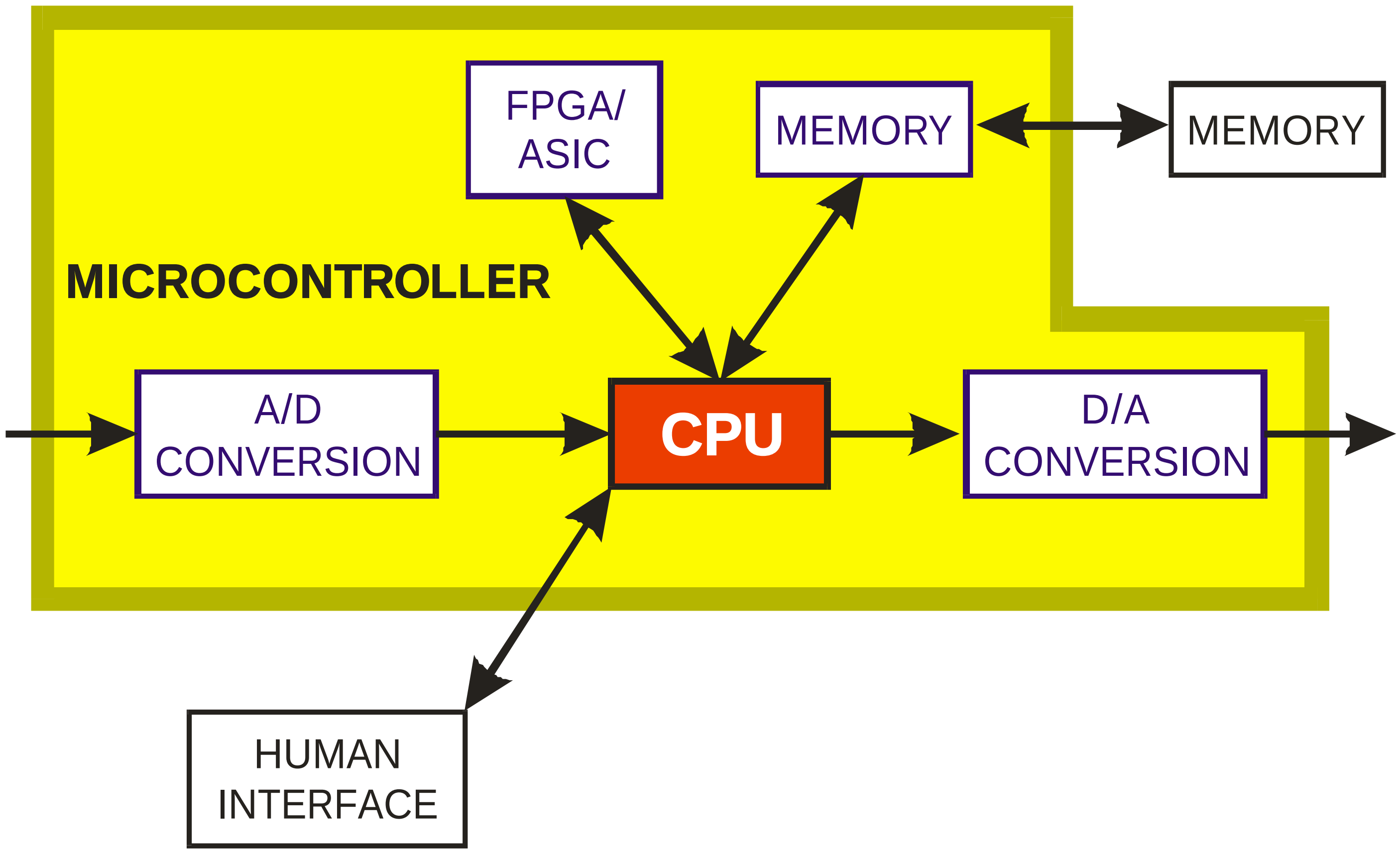


*Harris Semiconductor RTX-4000*

- **Lessons learned:**
  - General purpose processors are *Good Enough* for most things
    - Competing with Intel's technology curve is no fun at all!
  - For better or worse, the desktop market drives the high end
    - Some room left in ultra-high-performance consumer goods (e.g., MPEG players)
  - In the real world, engineers use of-the-shelf components whenever possible

# An Embedded Computer Designer's View

- **CPU + Storage + I/O**
  - Measured by: Cost, I/O connections, Memory Size, Performance



# Area #2: CAD Tool for System Synthesis

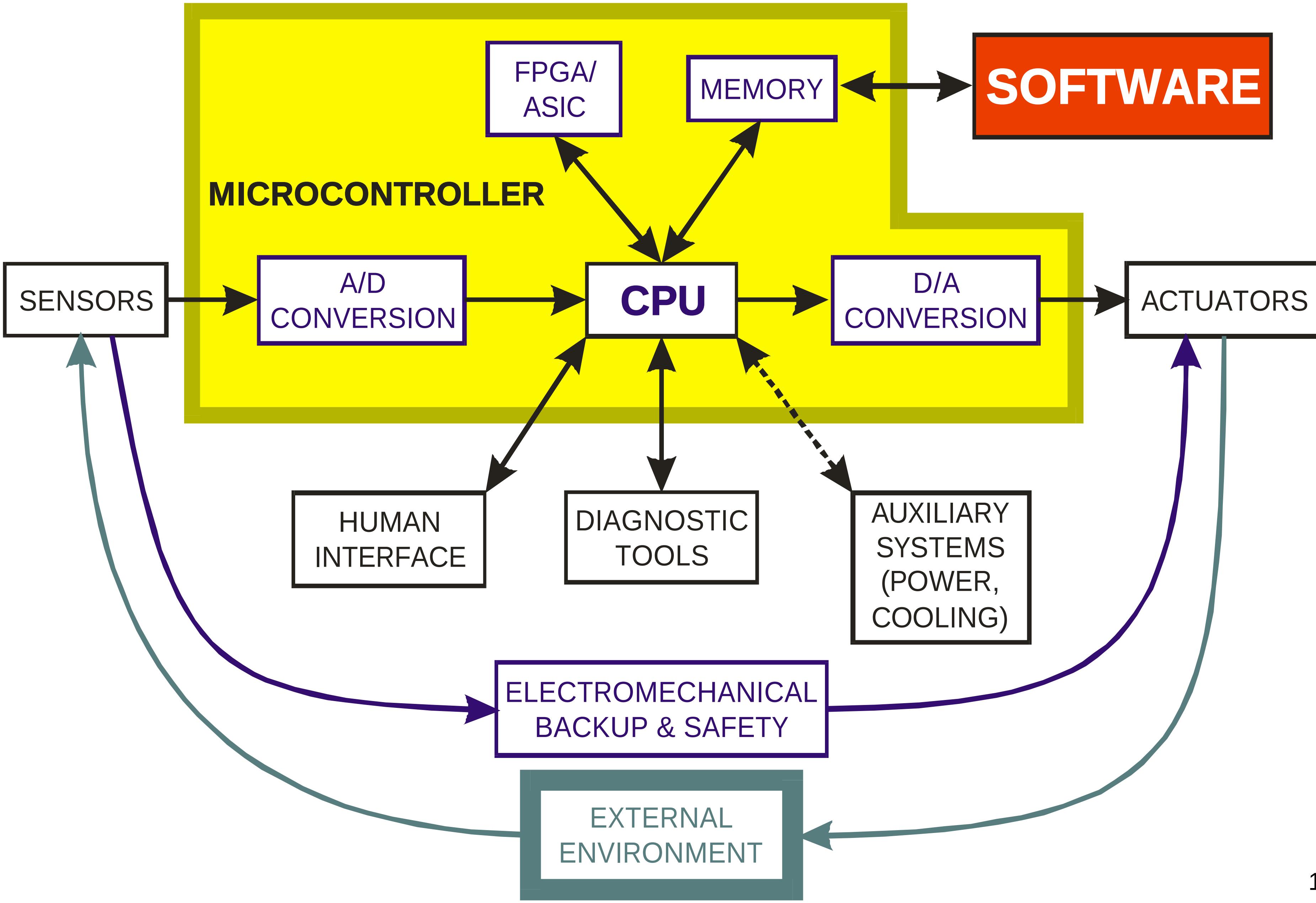
- **Omniview Fidelity – a design-by-composition tool**
  - Most Computer Aided Design (CAD) research is for synthesis, but has limited applicability to industry
  - Fidelity assembles circuit boards by selecting cost-optimized components
  - Experiment: attempt to duplicate hand-optimized design using design-by-composition tool


- **Lessons learned:**
  - Embedded systems need multi-technology tradeoffs
    - Analog, hardware, software, mechanical, …
  - Design tradeoffs for desktop computers are different than embedded tradeoffs

| Desktop | Embedded |
|---|---|
| – Average operating power | Standby power |
| – Component purchase costs | Lifecycle component costs |
| – Similar designs are equivalent | Each change requires certification |
| – Designed by specialists | Designed by generalists |

# An Embedded Control *System* Designer's View

- **Measured by: Cost, Time-to-market, Safety, Functionality & Cost.**

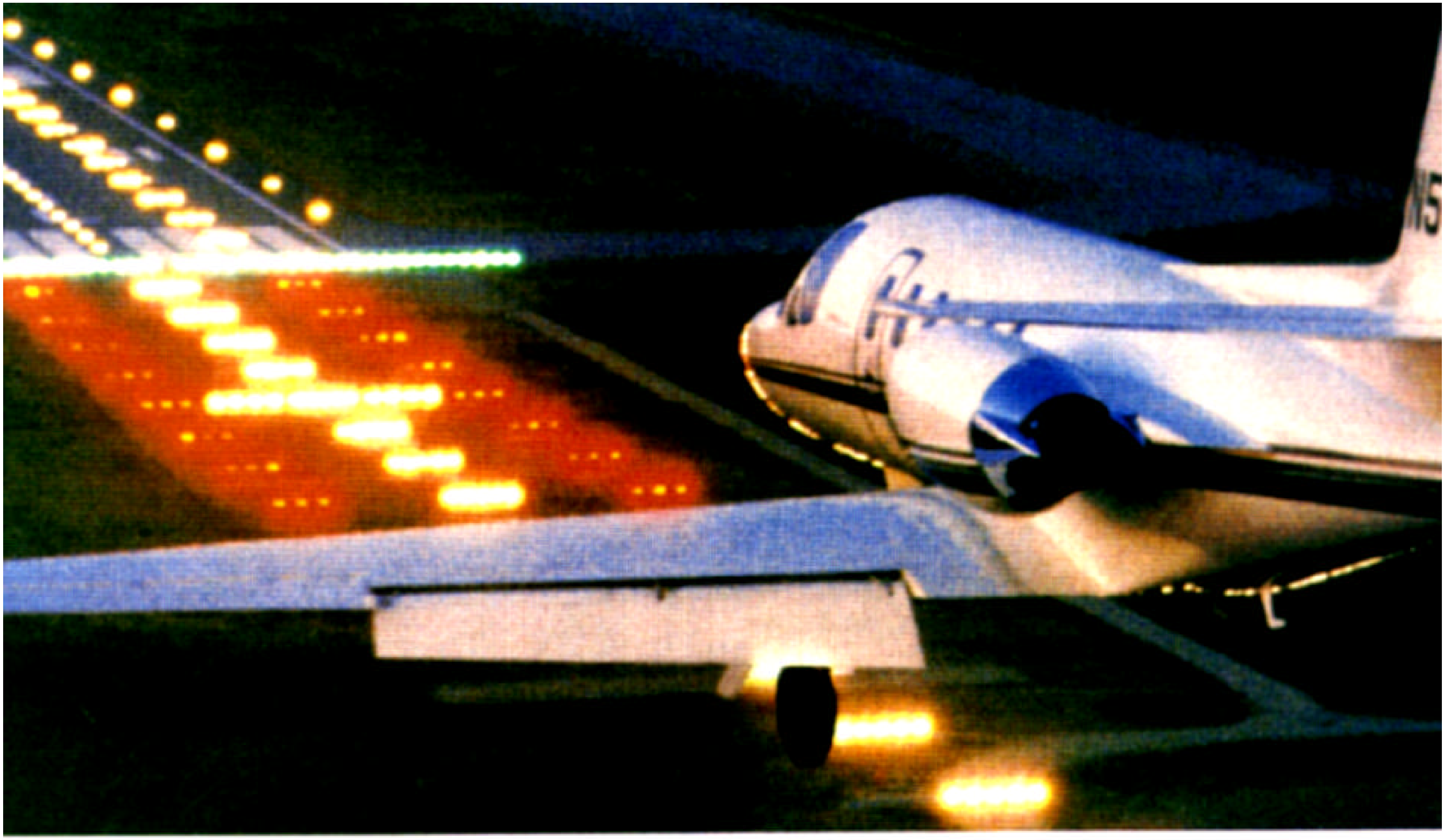


# Area #3: Software Robustness Testing

- **Research motivation:**
  - Low-cost reliability is essential for embedded systems
    - But we really don't know how to do that
  - Component-based systems are becoming prevalent
    - Anecdotally, exception handling *may* be a big source of problems
    - *Idea:* to build robust systems, put together individually robust components
  - *Software* components are probably the most important to study
    - Increasingly, that's where the complexity ends up…

- **Software Dependability ≈ reliability + robustness + safety + security**
  - Software "Reliability" º operates per specification
  - Software Robustness º acts reasonably in exceptional situations

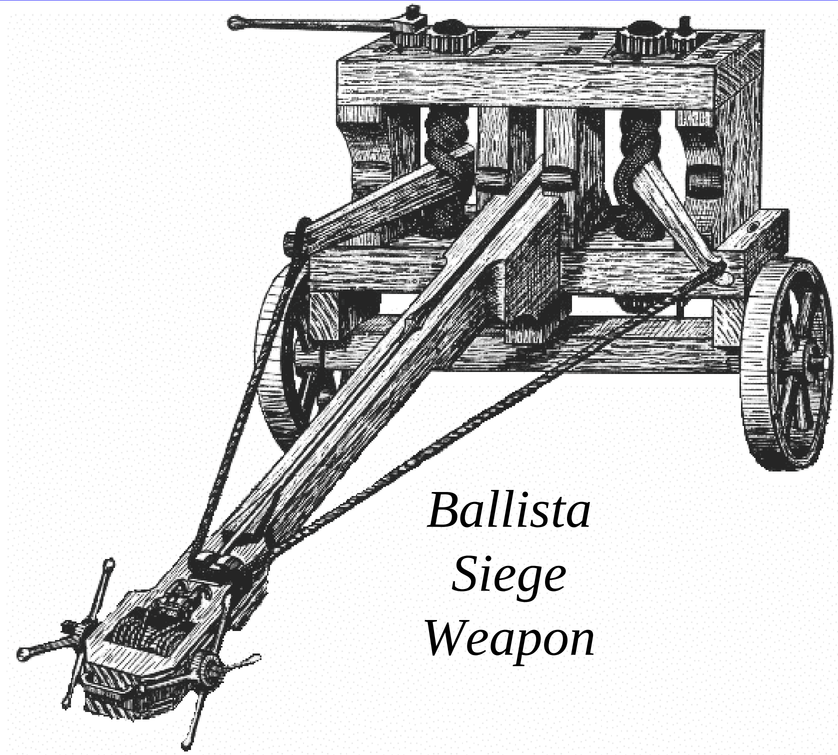
- **Research goal**
  - Find a way to quantify robustness levels



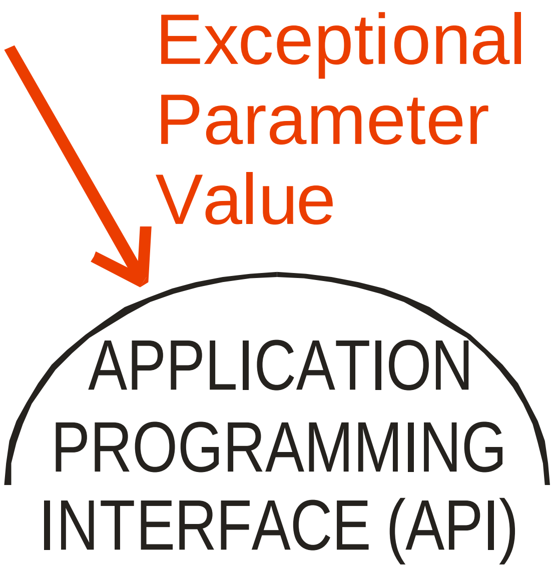
THIS IS A BAD PLACE TO DISCOVER YOUR RTOS IS ONLY 83.3% ROBUST.

# The *Ballista* Robustness Testing Approach

- **Use fault injection techniques**
  - "Ballista" is an ancient siege weapon for hurling big projectiles with good accuracy
  - Traditional fault injection corrupts code in system under test
    - Usual Hypothesis: "If there were a defect in the OS, the system could crash"



*Ballista Siege Weapon*




- **Ballista testing tool does API-level fault injection**
  - Simulates a software defect in something <u>calling</u> the interface
  - Does *NOT* inject a defect in the system under test itself
    - Ballista Hypothesis: "If there were a defect in a user program, the OS could crash"

# Ballista Research Challenges

- **Scalability of testing "oracle"**
  - How do you know if the test got the right answer?
  - Usual method requires knowing expected result of each and every test
  - *Solution:* use a "crash/hang" check instead of functional correctness check

    (similar to idea to "crashme" randomized OS testing, but applicable to any API)

- **Scalability of test cases**
  - Software testing effort is usually proportional to number of functions tested
  - "Scaffolding" set-up code for tests is a large development effort
  - *Solution:* create tests based on number of data types instead of functions

    (similar to idea for Category Partitioning testing method [Ostrand & Balcer '88], but without no per-test analysis required)

- **OS vendors said we'd never find anything**
  - Conventional wisdom is only remaining bugs are obscure, timing-related
  - *Solution:* they were wrong

# Ballista: Scalable Test Generation

*API* `write(int filedes, const void *buffer, size_t nbytes)`

*TESTING OBJECTS*

| FILE DESCRIPTOR TEST OBJECT | MEMORY BUFFER TEST OBJECT | SIZE TEST OBJECT |
|---|---|---|
| FD_CLOSED | BUF_SMALL_1 | SIZE_1 |
| FD_OPEN_READ | BUF_MED_PAGESIZE | SIZE_16 |
| FD_OPEN_WRITE | BUF_LARGE_512MB | SIZE_PAGE |
| FD_DELETED | BUF_XLARGE_1GB | SIZE_PAGEx16 |
| FD_NOEXIST | BUF_HUGE_2GB | SIZE_PAGEx16plus1 |
| FD_EMPTY_FILE | BUF_MAXULONG_SIZE | SIZE_MAXINT |
| FD_PAST_END | BUF_64K | SIZE_MININT |
| FD_BEFORE_BEG | BUF_END_MED | SIZE_ZERO |
| FD_PIPE_IN | BUF_FAR_PAST | SIZE_NEG |
| FD_PIPE_OUT | BUF_ODD_ADDR | |
| FD_PIPE_IN_BLOCK | BUF_FREED | |
| FD_PIPE_OUT_BLOCK | BUF_CODE | |
| FD_TERM | BUF_16 | |
| FD_SHM_READ | BUF_NULL | |
| FD_SHM_RW | BUF_NEG_ONE | |
| FD_MAXINT | | |
| FD_NEG_ONE | | |

*TEST VALUES*

*TEST CASE* `write(FD_OPEN_READ, BUFF_NULL, SIZE_16)`

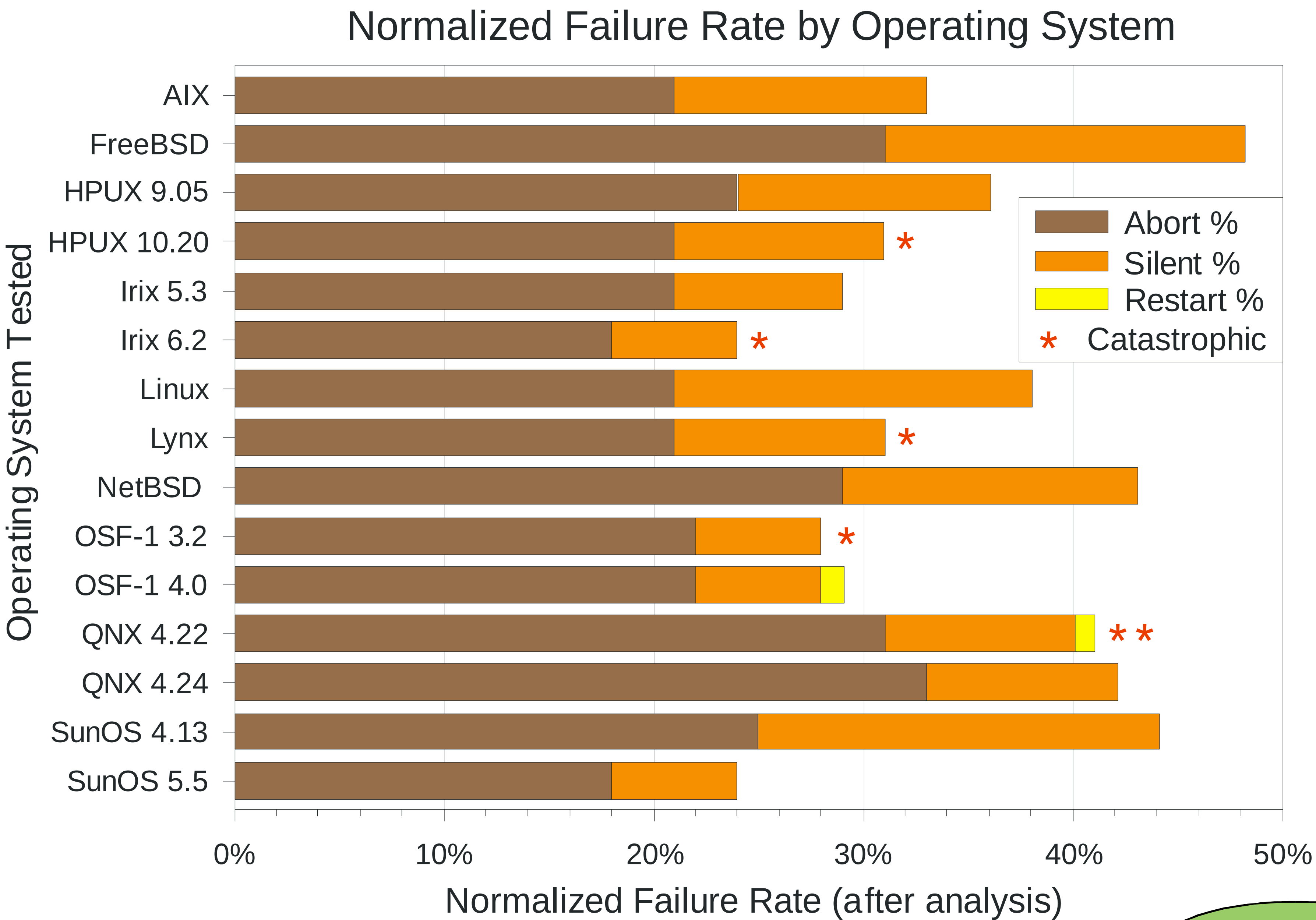
- Ballista combines test values to generate test cases

# CRASH Robustness Testing Result Categories

- **Catastrophic**
  - Computer crashes/panics, requiring a reboot

    *e.g.*, `GetThreadContext(GetCurrentThread(), NULL);`
- **Restart**
  - Benchmark process hangs, requiring restart
- **Abort**
  - Benchmark process aborts (*e.g.*, "core dump")
- **Silent**
  - No error code generated, when one should have been (*e.g.*, de-referencing null pointer produces no error)
- **Hindering**
  - Incorrect error code generated
  - Found via by-hand examinations, not automated yet

# Results for Unix Operating Systems

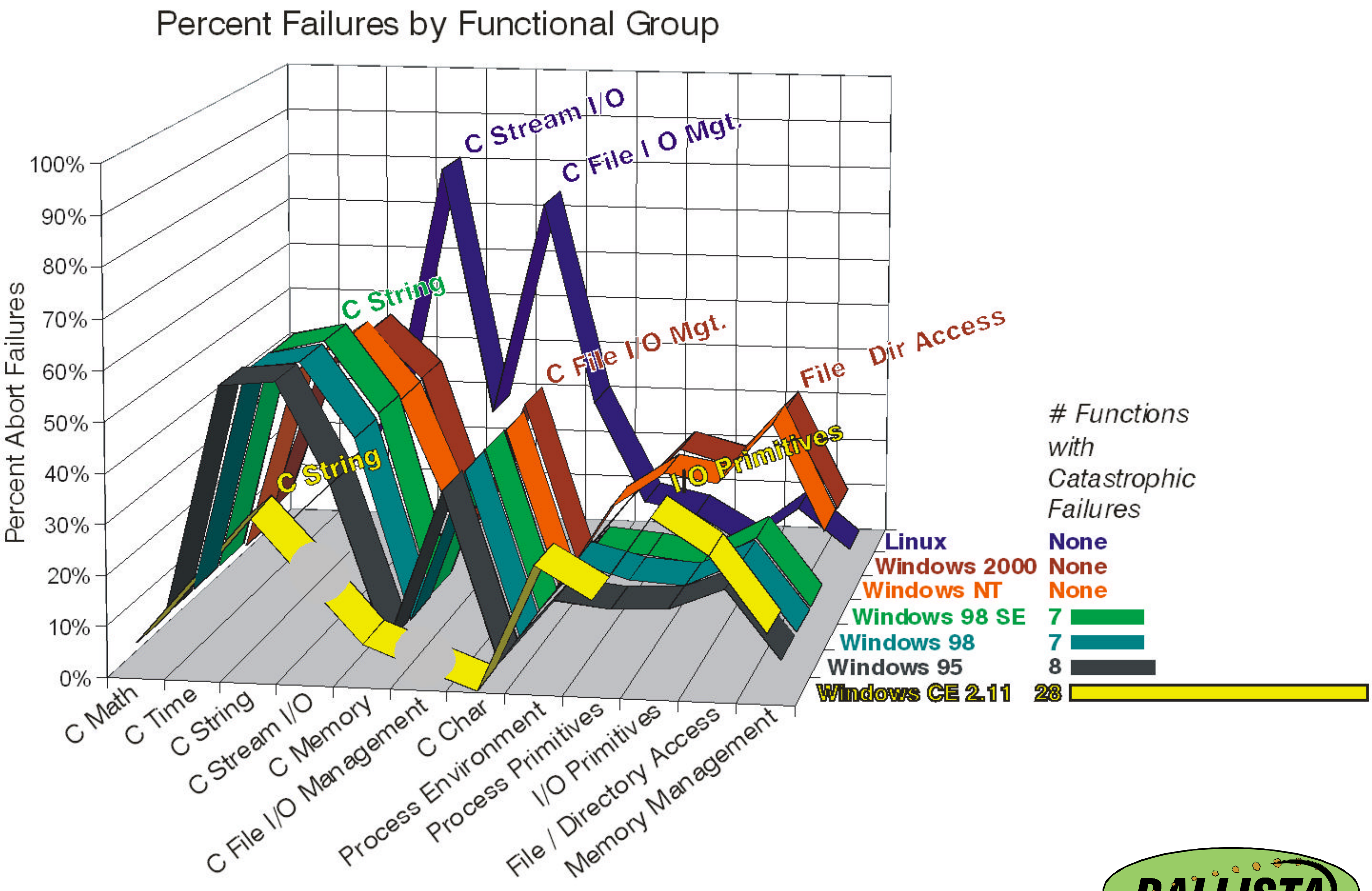


# But What About Windows?

Thinking of running your critical apps on NT?

Isn't there enough world suffering?

# Failure Rates by Function Group




BALLISTA

# Robustness Beyond Operating Systems

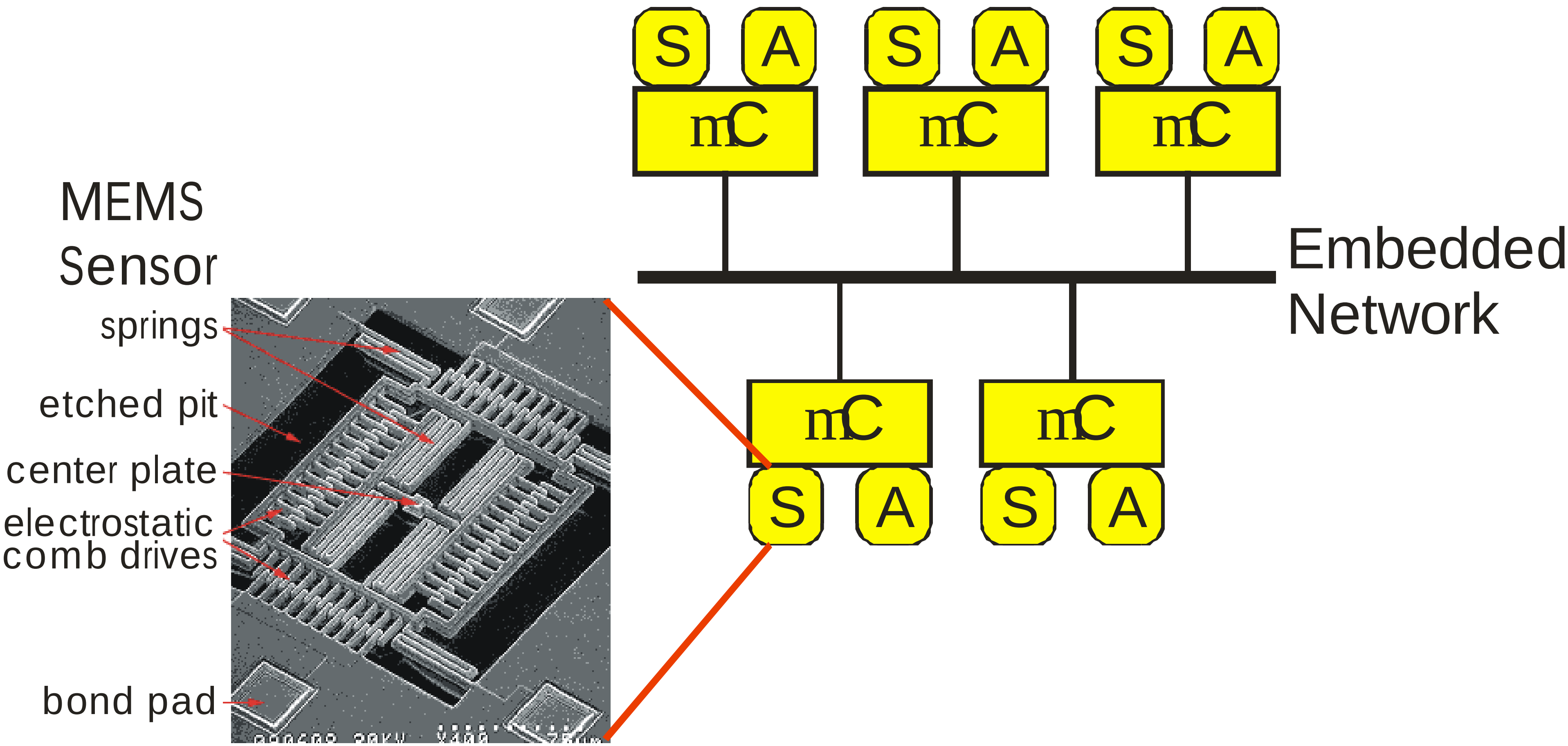
- **Some software is very robust**
  - HLA RTI – DoD distributed simulation backplane

- **Unfortunately, commercial software components tend to be non-robust**
  - Unix systems often displayed a vulnerability to crashing
  - Windows CE had 28 functions that could cause a crash
  - Initial results on several CORBA implementations don’t look promising
  - Initial results on accelerated software aging tests look bad for both Linux and Windows

- **How do you build a robust system from non-robust components?**
  - Multi-version Unix implementations won’t work (we checked)
  - The market isn’t demanding more robust software (yet)
  - *Solution:*(?) build systems that degrade gracefully when components fail

# A Distributed Embedded System Designer's View

- **Highly distributed systems**
  - Measured by: Product family success, life-cycle cost, dependability
  - Fine-grain, system-on-chip nodes form distributed systems
    - Micro-Electrical/Mechanical Systems (MEMS) for I/O
    - Microcontroller & network connection "for free" on same piece of silicon



# Area #4: Graceful Degradation

- **Research motivation:**
  - Future embedded systems will be low cost, but highly distributed
  - In the common case, not all components will be working
    - Degraded or failed component hardware
    - Unreliable/non-robust component software

- **Existing graceful degradation techniques rely upon manual reconfiguration**
  - Works fine for a few processors
  - Doesn’t scale to thousands of processors

- **Research goals**
  - Achieve automatic graceful degradation for a given system architecture
  - Create general principles for designing architectures that gracefully degrade

# RoSES: Robust Self-configuring Embedded Systems

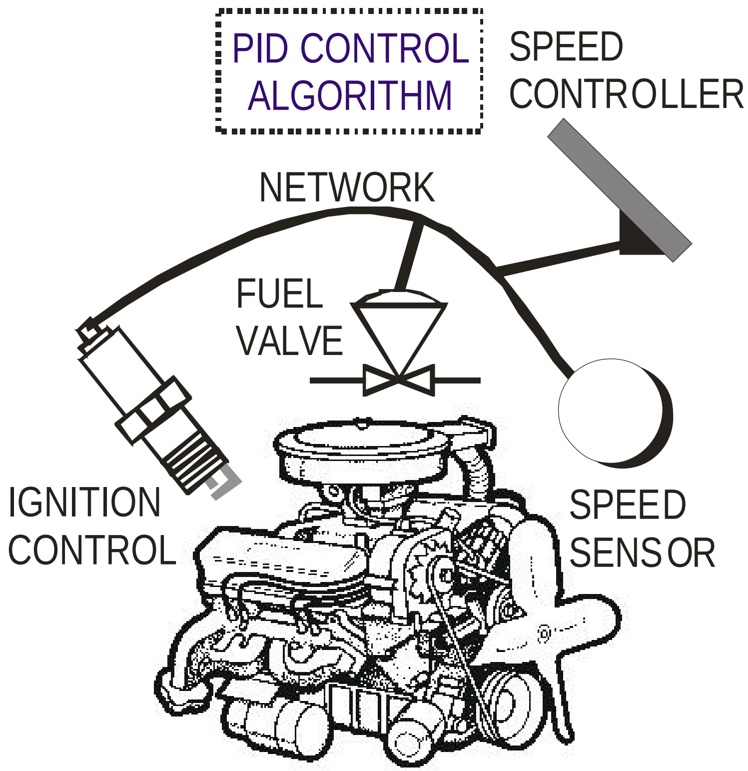
- ***Automatic configuration management* is a unifying capability**
  - Product families can include degradation as well as intentional price/performance tradeoff points

- **Consider component failure as an example:**
  - Component fails –
    triggers reconfiguration for degraded operation
  - Component replaced –
    reconfiguration to integrate repair part
  - New component added –
    reconfiguration to upgrade system

- **That's a lot to attempt all at once…**
  - Static configuration at first
  - On-the-fly configuration as an eventual goal

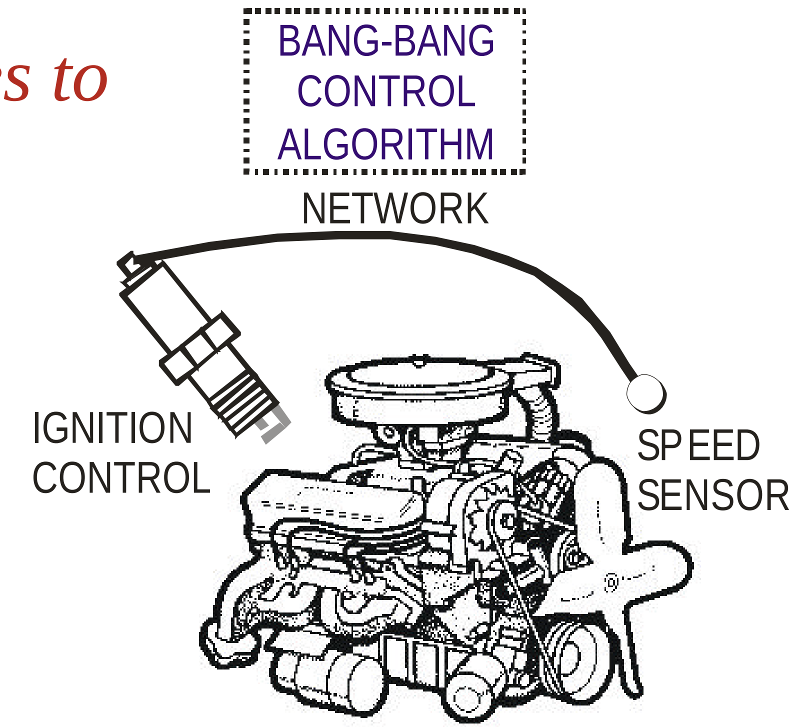
# A Simplistic Example

- **Control of gasoline engine speed**
  - Complicated system controls fuel if valve is installed/operational
  - But, baseline capability is retained in case of failure




Throttled Engine Controller

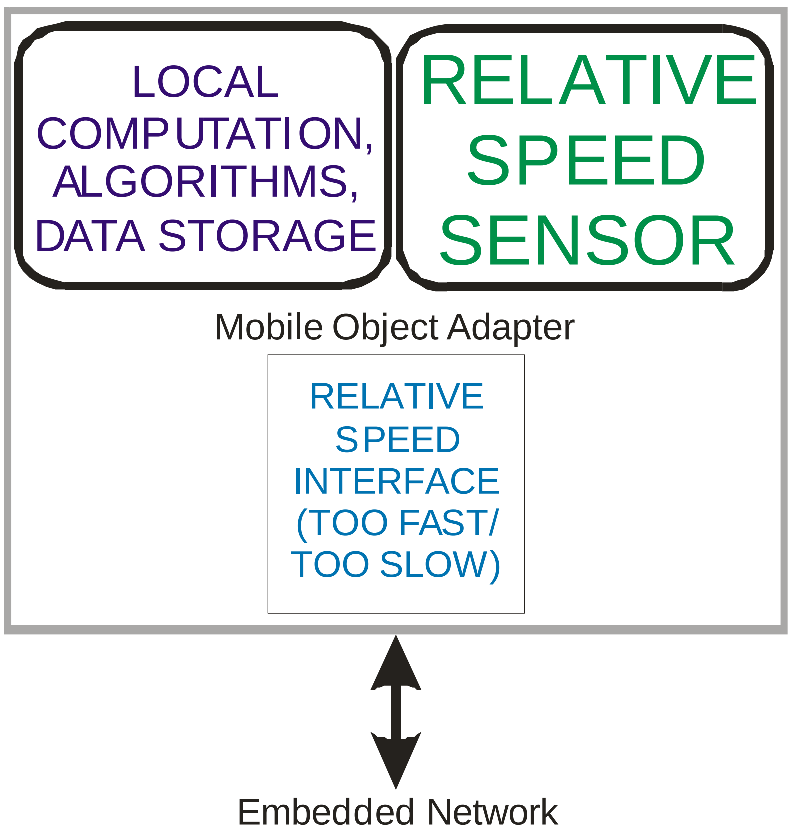
*Degrades to*



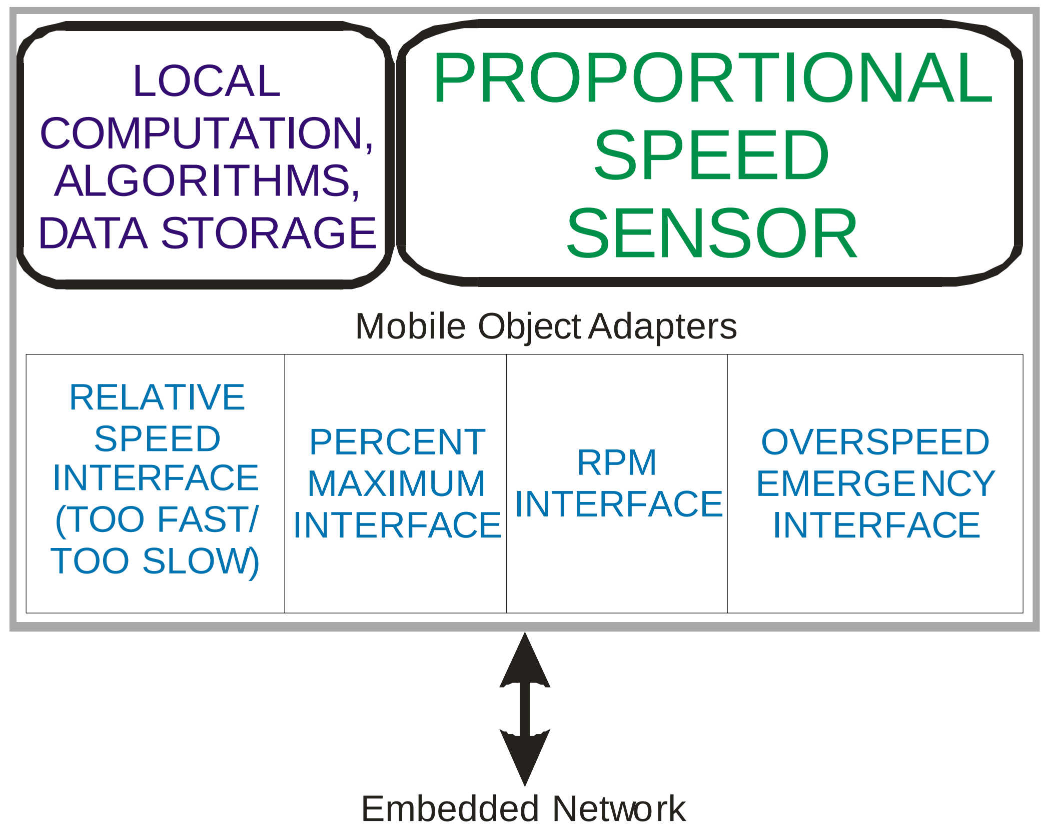

Hit-or-Miss Constant Speed Engine Controller

# Different Sensors / Different Capabilities

- **Similarly, different actuators have different capabilities**
  - *Mobile Object Adapters* translate raw capability into desired interface

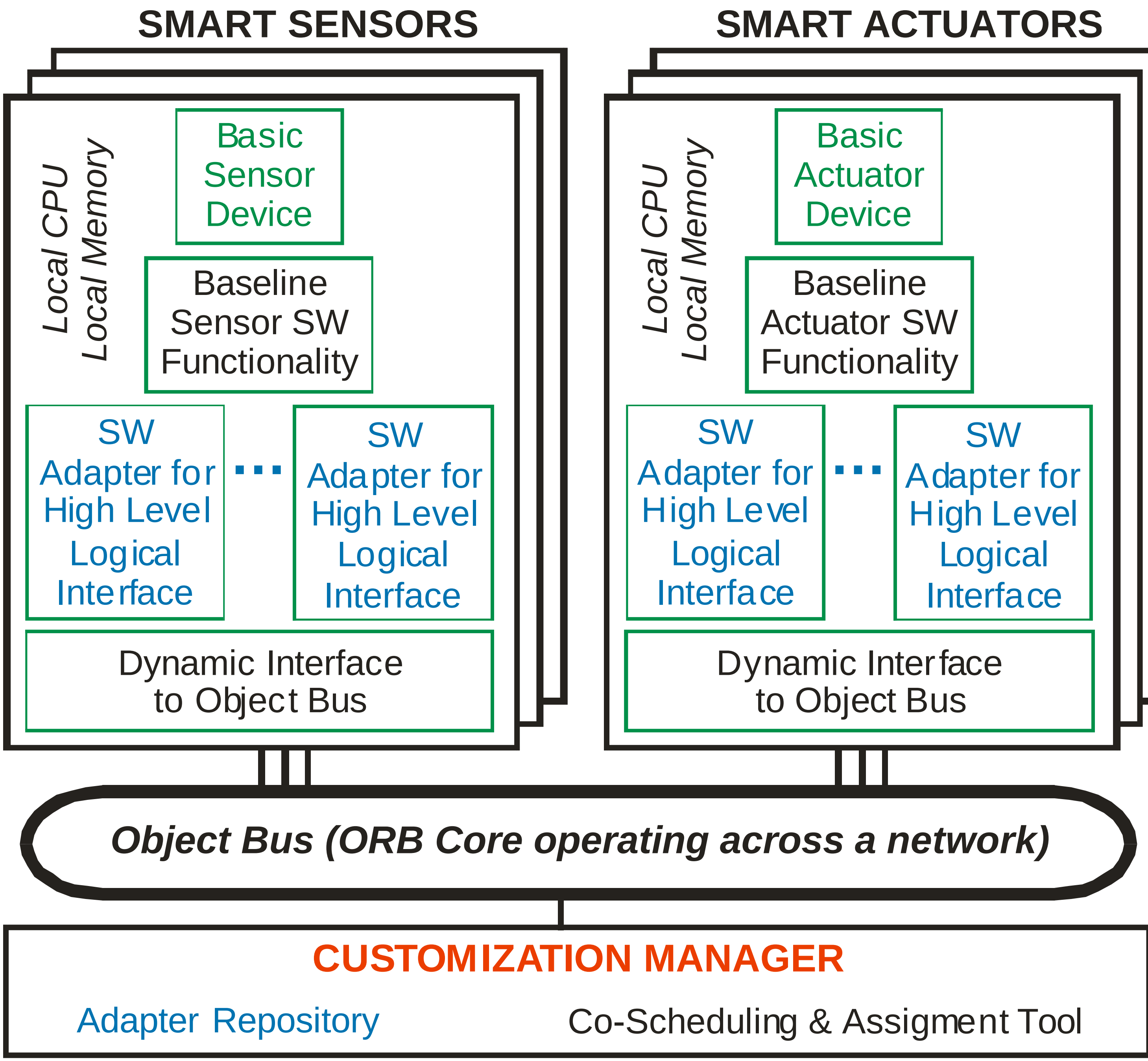



EXAMPLE SIMPLE SPEED SENSOR



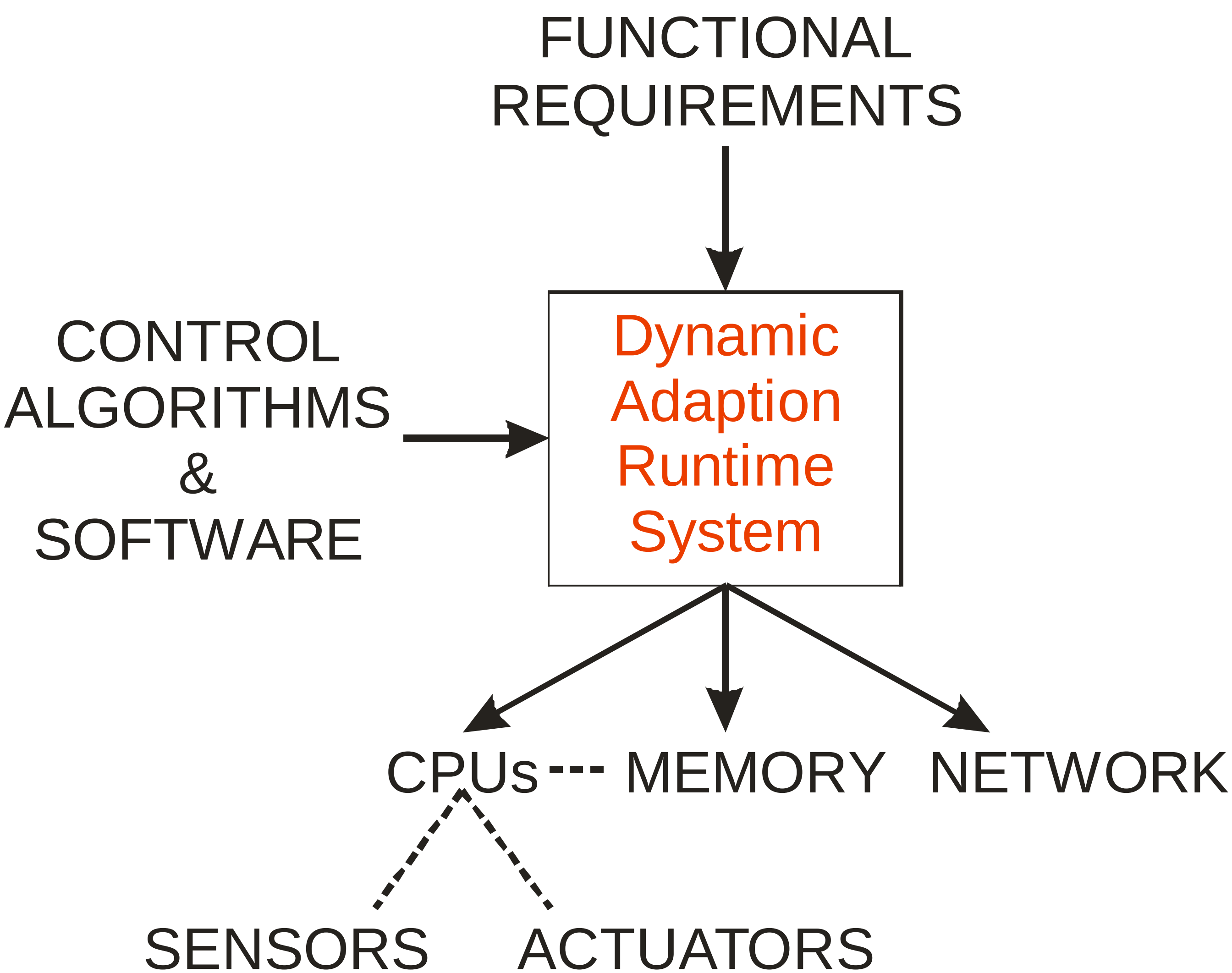

EXAMPLE HIGH-END SPEED SENSOR with MULTIPLE INTERFACES

# Generic RoSES System Architecture



# Functionality To Hardware Mapping

- **One element of RoSES:**
  **Automatic allocation of HW & SW components**
  - Maximize utility of functions within hardware constraints



# Near-Term Research Challenges

- **Mapping functionality onto hardware**
  - Maximize utility of result given constrained resources
- **Achieving real-time operation**
  - Co-schedule CPU, Memory, Network usage to meet real-time deadlines
- **Achieving "plug & play" capabilities**
  - Is CORBA too "fat"? (how about Jini…)
  - Avoid re-inventing distributed object technology if possible!
- **Testbed & demonstration**
  - Generic automotive testbed
  - Apply techniques to multi-sensor vehicle navigation & other functions
- ***Plenty* of long-term research challenges too, of course**

# Other Current Activities

- **Chair of IFIP WG 10.4 SIG on dependability benchmarking**
  - How do you get measures of system dependability that work for real-world conditions?
  - Representatives from universities, industry, government labs

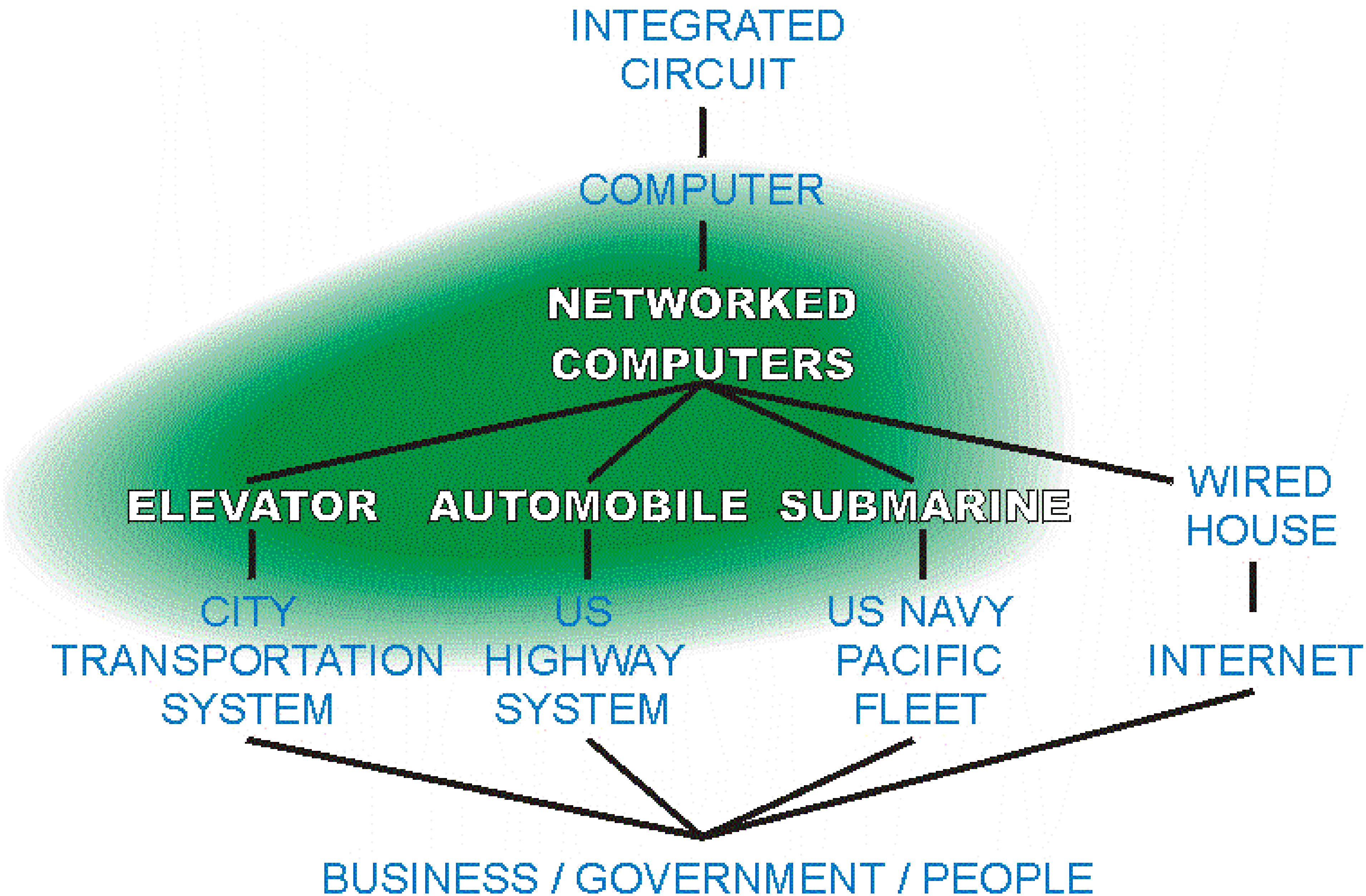
- **Industry-funded research efforts**
  - General Motors, Bosch: graceful degradation of automotive systems
  - Adtranz: dependability analysis of train network protocol
  - Emerson, Microsoft: software robustness of Windows
  - IBM: “bulletproof Linux”
  - ABB: software robustness of embedded systems

# Teaching System Architecture (current status)

- **CMU ECE 18-540: Distributed Embedded Systems**
  - Elevator as an example
  - Includes lightweight software engineering: requirements to validation
  - Material motivated via "war stories"

- **Business issues**
  - How does a particular company make its profits?
  - Non-technical constraints on solutions are a reality
- **Levels of abstraction**
  - Top-down decomposition + Bottom-up synthesis
  - Orthogonal building blocks (when you can find them)
- **Multi-technology tradeoffs**
- **Non-functional requirements**
  - "ilities", safety, cost
- **Life-cycle perspective**
  - Requirements through disposal
  - Selected real-world issues: spare parts, cross-cultural designs, ethics

# Current Research Scope



Phil Koopman
Jiantao Pan
Kanaka Juvva (graduating)
Bill Nace
Kobey DeVale
Ying Shi
John DeVale

Meredith Beveridge
Charles Shelton
Sandeep Tamboli (graduating)

*Not shown (new additions):*
Tridib Chakravarty
Beth Latronico