18-649 Distributed Embedded Systems

Prof. Philip Koopman

Fall 2015

Lecture: Mon/Wed 12:30-2:20 PM

Recitation: Friday 12:30-2:30 PM

(includes required weekly meeting slots)

Required reading as posted at http://www.ece.cmu.edu/~ece649

WAITLIST INFORMATION appears on later slides

Carnegie Mellon

Instructor Background

- **Prof. Phil Koopman**
  - HH A-308
  - ece649-staff@ece.cmu.edu
- **Research:**
  - Embedded system security
  - Embedded system safety & dependability
  - Embedded real-time networking
- **Engineering experiences outside Carnegie Mellon**
  - Expert witness on Toyota Unintended Acceleration cases
  - Embedded CPU designer for Harris Semiconductor
  - Embedded system architect for United Technologies (Otis, UT Automotive, Pratt & Whitney, Carrier, Norden, Sikorsky, …)
  - 140+ design reviews of industry embedded systems
  - Startup company that did embedded CPU design
  - US Navy submarine officer

18-649 Distributed Embedded Systems

- **Based on book, lecture notes, project, and industry reading**

- **Course objectives detailed on web pages**
  - System Engineering
    - Requirements, design, verification/validation, certification, management-lite
  - System Architecture
    - Modeling/Abstraction, Design Methodology, a little UML, Business Issues
  - Embedded Systems
    - Design Issues, scheduling, time, distributed implementations, performance
  - Embedded Networks
    - Protocols, real-time performance, CAN, FlexRay, embedded Internet
  - Critical Systems
    - Analysis, software safety, certification, ethics, testing, graceful degradation
  - Case Studies
    - Elevator as semester-long design project
    - Guest speakers and other discussions as available

Pre-Requisite Knowledge

- **18-213 at CMU is STRICTLY REQUIRED** (15-213, 15-513 are OK)
- **Java programming**
  - Basic use of Unix and/or Windows systems and afs
  - Course project uses Java simulation harness
  - **If you don't know Java, learn it now!** You will need it. Soon.
    - "I'm not good at Java" is **not** an acceptable excuse for slacking in the project
- **Intro to embedded systems (18-348, 18-349, or experience)**
  - Written medium-size **C++ or Java** programs.
  - General familiarity with **development tools** including compilers, linkers, Unix command line, version control tools (Git or other), scripting language (Perl, Python, or other), setting up spreadsheet calculations, ability to edit/create simple html. You will nee all these things here.
  - Familiar with basic **embedded concepts** such as interrupts, determining execution time, debugging, networks, counter/timers, mutexes, D/A, A/D
  - Some experience at **working in teams**, including breaking down tasks, tracking progress, and preparing team presentation (course project is done in teams of 4 students)
  - Intro-level probability theory

Significant Course Project

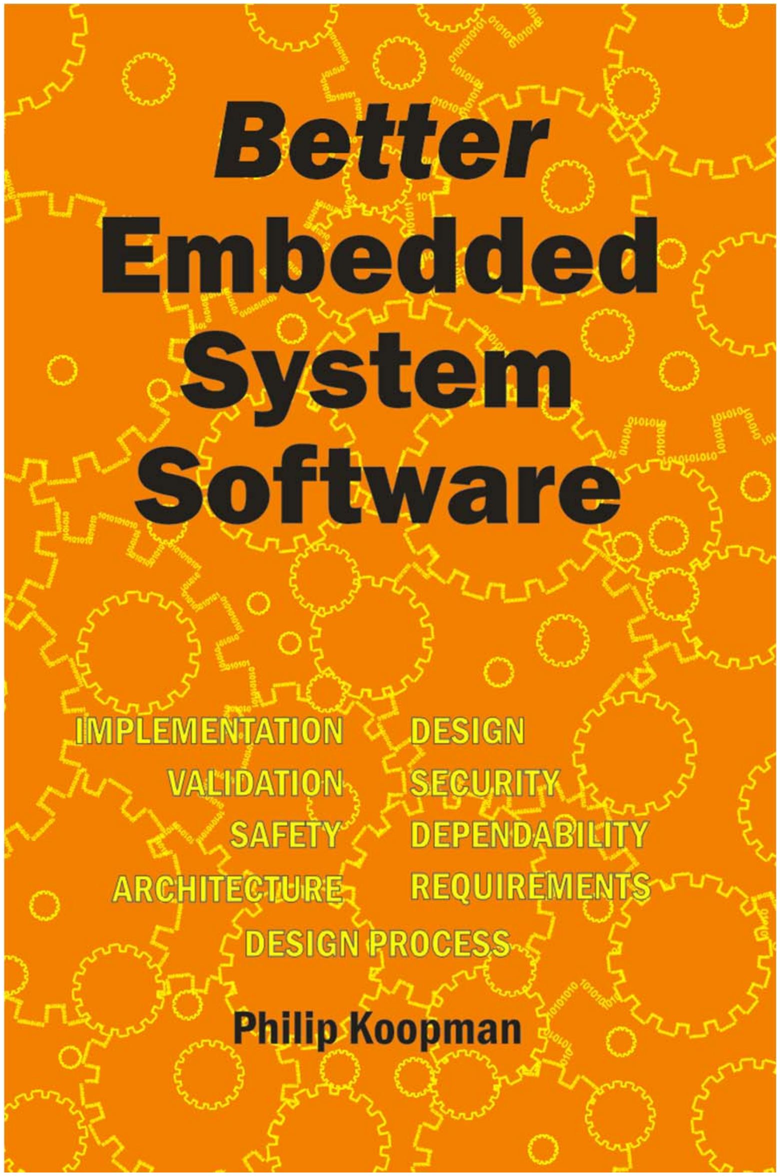
- **Build a simulated elevator implemented as distributed system**
  - Emphasis on good (but lightweight) process and high quality design
    - You will learn how to be better than many industry embedded SW designers
  - Java-based distributed simulation framework
    - Learning how to do simulations is just as important as hacking hardware
    - (You should already know how to hack hardware; not part of this course)
  - **Elevators make a good example system**
    - Real elevators are a lot more complex than they appear
    - Our elevator is based on real elevator experience from Otis and others

- **Project approach**
  - Teams of 3 or 4.
    - Start with teams of 4
    - If there are drops then leave teams of 3 undisturbed as much as possible.
    - Teams assigned next week; you can request specific team members
  - Weekly project phases to spread out work and reduce mortality rate
  - “Simple” running code at mid-term; more complex code at end of semester
  - Focus on industry-grade engineering process; not fancy technology

Text: Better Embedded System Software

- **Each chapter is based on real systems**
  - Real companies, real products, real mistakes
  - Often the reviews were to save failing projects
  - *This is the stuff designers get wrong*

- **Purchase via web**
  - Best price is via “student discount” web page via Paypal
    - $50 with free shipping
    - See pointer on course web page
  - Amazon.com stocks at $89
  - One copy will be on reserve in E&S library



Policy Summary

- **See http://www.ece.cmu.edu/~ece649 for official, detailed versions**
  - *Send all e-mail to the entire course staff:*

    ece649-staff@ece.cmu.edu
  - *Why?* **Because we might be off-line, sometimes for multiple days.**

- **Grading: straight scale A ≥ 90; B ≥ 80; C ≥ 73; R < 73**
  - No "curve" – 89.9 is a "B" … (but you only need 90.00 for a guaranteed "A")

  TESTS:
  - 45 points for in-class tests (two tests, equally weighted); no final exam

  PROJECT:
  - 40 points for project phases (team grade)
    - **Mid-term & Final projects *MUST* pass acceptance tests to pass the course**
  - 8 points for in-class presentations (during semester and end of semester)

  ATTENDANCE:
  - 7 points attendance (weekly survey, meeting attendance, classroom attendance)
    - Attendance at all class events is mandatory
    - 3 free absence points (two or three days of absence based on attendance points)
    - ***Negative points can accumulate without limit***
    - *Having someone else sign you in is cheating; don't do this!*

Assignments, Etc.

- **Lectures are available on line at least the night before class**
  - Handouts provided in classroom
  - Previous year lectures on line now; most won't change too much
- **Required readings**
  - Required reading is testable material; not 100% overlapped with lecture
  - Emphasis on book chapters based on experience from industry reviews
  - Papers representative of what working engineers read to stay current
- **Weekly project milestones**
  - Project reports & materials due generally on Thursday night
  - Group status meetings held on Fridays during or near recitation interval
  - READ the project assignment BEFORE recitation. Ask questions
- **Tests**
  - Were you paying attention in class? Did you actually do the reading?
  - One 8.5" x 11" notes page 2-sided – ***must be in your own handwriting***
  - We'll provide previous-year tests in time to study
  - If you have special needs (e.g., extra test time) TELL US THIS WEEK!

Late Penalties & Other Policies

- **Being on time counts in the real world; it counts here too**
  - Being late for presentations & status meetings incurs penalty in proportion to lateness
  - Project late penalty:
    - Score multiplied by 0.9 if late up to 1 hour, else:

$$grade = MAX\left(score * 0.9^{\left(\lceil \#dayslate + 1 \rceil\right)}, 0.43\right)$$

- **Limited makeup policy:**
  - Makeup exams only under very specific conditions; read policy page
  - Assignments are available well in advance; no extensions if CMU is open.
  - If you have a presentation or it is a test day, catch one bus earlier than you normally do
- **No cheating**
  - *Penalty for first cheating offense is failure ("R" grade) for the entire course.* ***No kidding.***
    - ***Record of cheating could show up on background checks conducted by future employers***
  - ***Reference to other groups or previous semester solutions is expressly forbidden***
  - Keep your eyes on your own paper during tests
  - Tell the truth about what parts of the project you work on
    - If your partner cheats and you take credit for that work product, you are guilty of cheating
  - CMU general policy and details on course web also page apply. Read them!
- **LOOK at the course web page AND the administrative page!**
  - http://www.ece.cmu.edu/~ece649

Classroom Protocol

- **Please arrive on time; lecture begins promptly**
  - Please put extra handouts in pile by door for the few latecomers
    - Handouts on the web – students new to English should read night before
  - If you want to skip a guest speaker, leave ***before*** he/she starts!
  - Attendance is mandatory. We will be taking attendance
    - Students have lost 1, or sometimes 2, letter grades due to poor attendance
    - If you have 15-20 plant trips scheduled, take a different course
  - No noisy food in the classroom (paper wrappers, rustling chip bags)
- **Questions are encouraged**
  - If you don't understand, ask (other students probably want to know too)
- **There is no way to cover everything**
  - Embedded systems is a huge area; this course is really "survival skills for new embedded engineers" (assuming you already know intro stuff)
  - I'm electing to cover fundamentals rather than latest fad topics (little emphasis on internet toaster ovens in this course)
  - There is a "digging deeper" section for each lecture on the web site

Other Notes

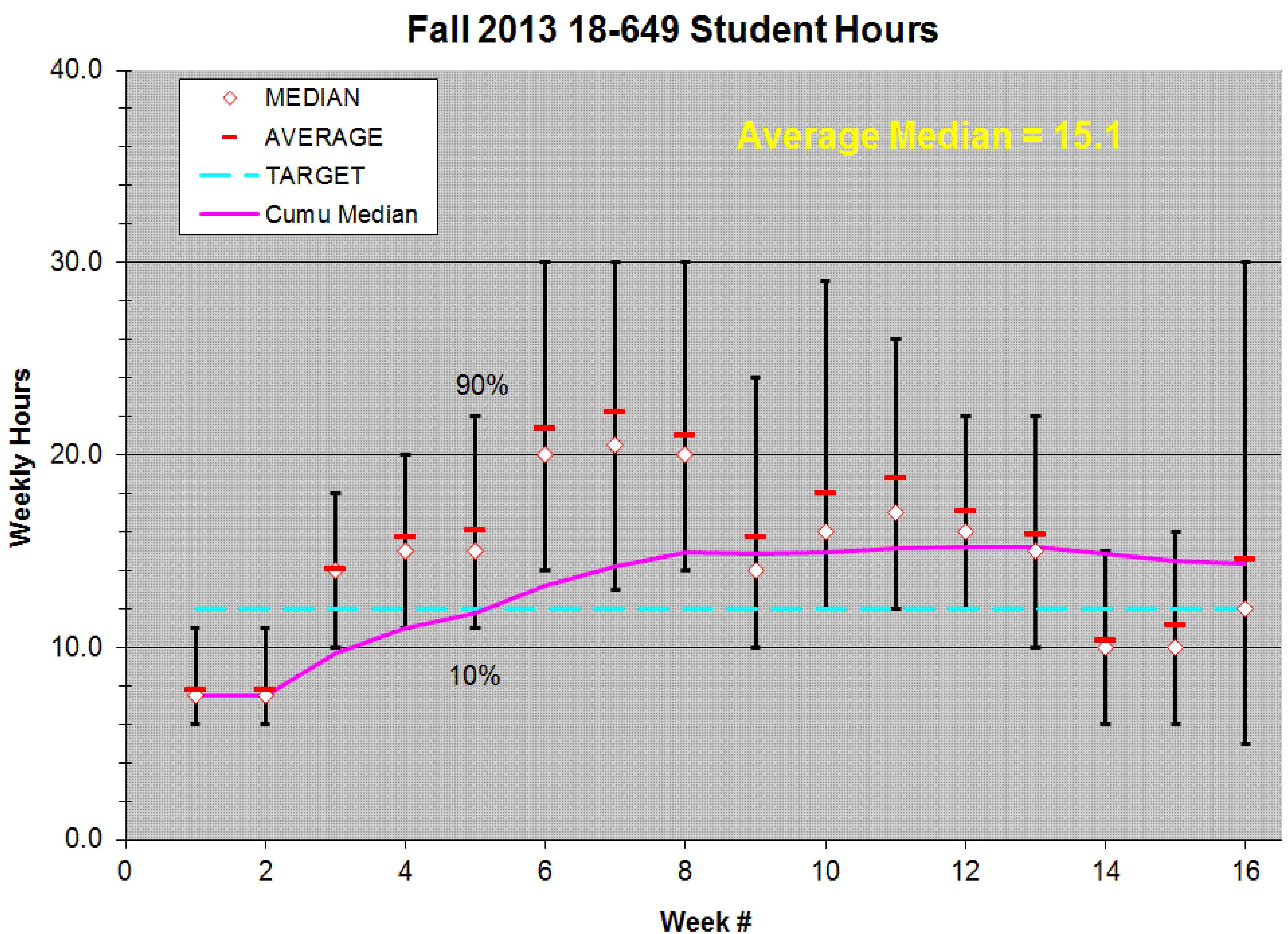
- **Additional policy notes (at the advice of university legal department):**
  - All course material is copyrighted by the instructor
  - You specifically do ***not*** have permission to reprint, publish, upload, or distribute anything (course handouts, tests, notes, book chapters, project materials)
  - You do ***not*** have permission to record or stream any lecture
  - Fair Use permissions inherent in copyright law remain in effect, but do not permit the above
- **End of Semester travel**
  - In-class presentations are **mandatory** (2^{nd} presentation last class week)
  - Final project hand-ins are during first week of final exams
    - You must be physically present for your team's final demo
    - If you want to leave early for winter break it is YOUR responsibility to have all of your obligations fulfilled before you leave
      - » This means successful demo before you leave campus
      - » If you leave before demo or bail out on your team, expect a ***significant*** penalty
    - We do not reschedule presentations due to travel plans
      - » You might be able to arrange a swap, but burden is entirely on you to figure it out

More On Attendance

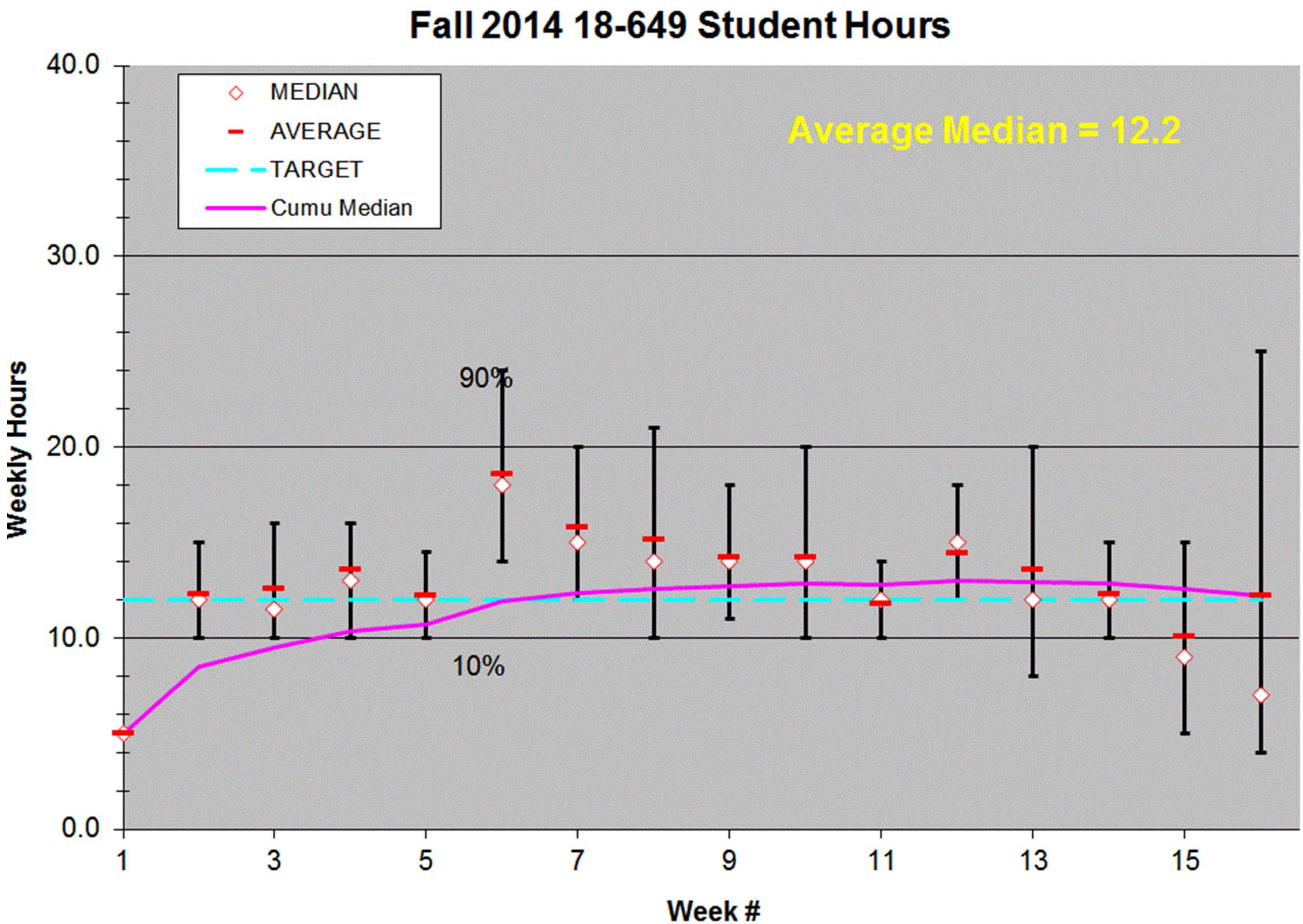
- **Attendance is mandatory** (and attendance will be taken regularly)
  - If you plan to miss 13 lectures due to job hunting… take a different class
    - At least one student failed to graduate due to excessive skipping
    - Signing in for someone else is cheating and will be dealt with severely
- **Why do I take attendance?**
  - Reading the handouts doesn't necessarily give you the big picture
    - Even though the handouts are extensive, they don't have everything
    - The "war stories" put things in perspective
  - You won't ask clarifying questions if you're not in class
    - And (more importantly) you won't hear the questions other students ask
  - Some topics are difficult to structure fair test questions about
    - I'd rather measure exposure to some topics directly (attendance) rather than indirectly (requiring lots of memorization of fine points on slides)
    - This course is as much about ***<u>changing how you think</u>*** as it is about specific facts
  - Poor attendance correlates strongly with poor projects & poor tests
    - Taking attendance encourages the right learning outcome

More On Cheating

- **In past semesters I have failed up to 10% of the class for cheating**
  - Primarily due to copying project information from other groups
  - If I determine you are cheating you will fail the course. **No exceptions.**
    - This might mean you won't graduate
    - This might mean it will be reported on future background checks
    - This might mean you won't get a job/won't be able to start a job
- **I will run the MOSS tool set on projects at the end of the semester**
  - I compare your projects against many years of project hand-ins (code & other)
  - Some students think they can beat MOSS. ***I will know*** if you are doing that
    - Students are usually astonished when they get caught.
    - If you didn't copy, you have nothing to worry about. (Yes, Really!)
- **This is a US graduate program and US rules apply**
  - We use the same project each year to give a much better learning experience
  - "I copied a starting point but worked hard after that" … is still cheating
  - "I just looked at some code without copying" … is still cheating
  - "I was just helping my friend" … is still cheating
  - "In my culture I have to help if someone asks" … is still cheating



Fall 2013 18-649 Student Hours
MEDIAN
AVERAGE
TARGET
Cumu Median
Average Median = 15.1
90%
10%
Weekly Hours
40.0
30.0
20.0
10.0
0.0
0
2
4
6
8
10
12
14
16
Week #



Fall 2014 18-649 Student Hours
MEDIAN
AVERAGE
TARGET
Cumu Median
Average Median = 12.2
90%
10%
Weekly Hours
Week #
40.0
30.0
20.0
10.0
0.0
1
3
5
7
9
11
13
15

WAIT LIST UPDATE

- **As of Thursday: __ enrolled; __ on waitlist**
  - Can handle 64-72 based on TA availability and room size
- **If you want to be enrolled, come to lectures**
  - To the degree the department permits it, I will announce and fill empty class spaces from students physically present in lecture
- **Usually takes two weeks for enrollment to settle down**
  - Most years all grads & seniors eventually get in
  - Many semesters essentially *all* the drop/adds happened at the end of week #2
  - But it all depends on how many students drop
- **If you decide to drop, please send e-mail to us!**
  - The Hub does not send out automatic notification
  - We have to manually check the enrollment list to see if someone dropped
- **This course is not wireless sensor networks / Android**
  - It is, however, about giving you the engineering skills that you need to succeed in the embedded industry
  - If you plan to drop, please let me know today

1
Embedded System Foundations

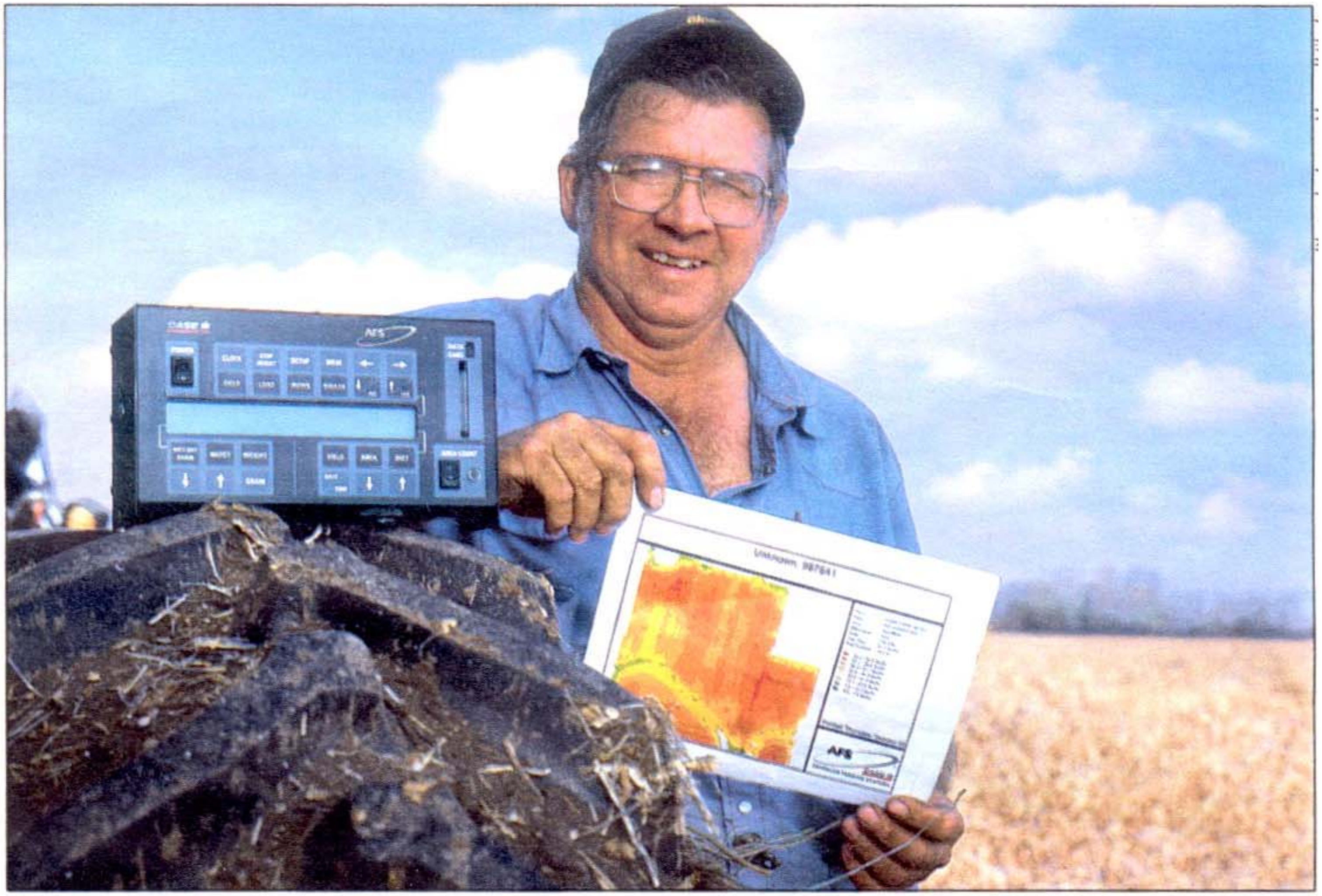
Distributed Embedded Systems

Philip Koopman

August 31, 2015

Carnegie Mellon

"IT SURE WOULD BE MORE WORK WITHOUT COMPUTERS," SAYS A SOYBEAN FARMER WHO RELIES ON HIGH-TECH HELP FOR HARVESTING.



HARVESTING BEANS AND DATA. Ted Sander, 52, a farmer from Moberly, Mo., uses an onboard computer to create maps that show which plots need more fertilizer, herbicide or pesticide.

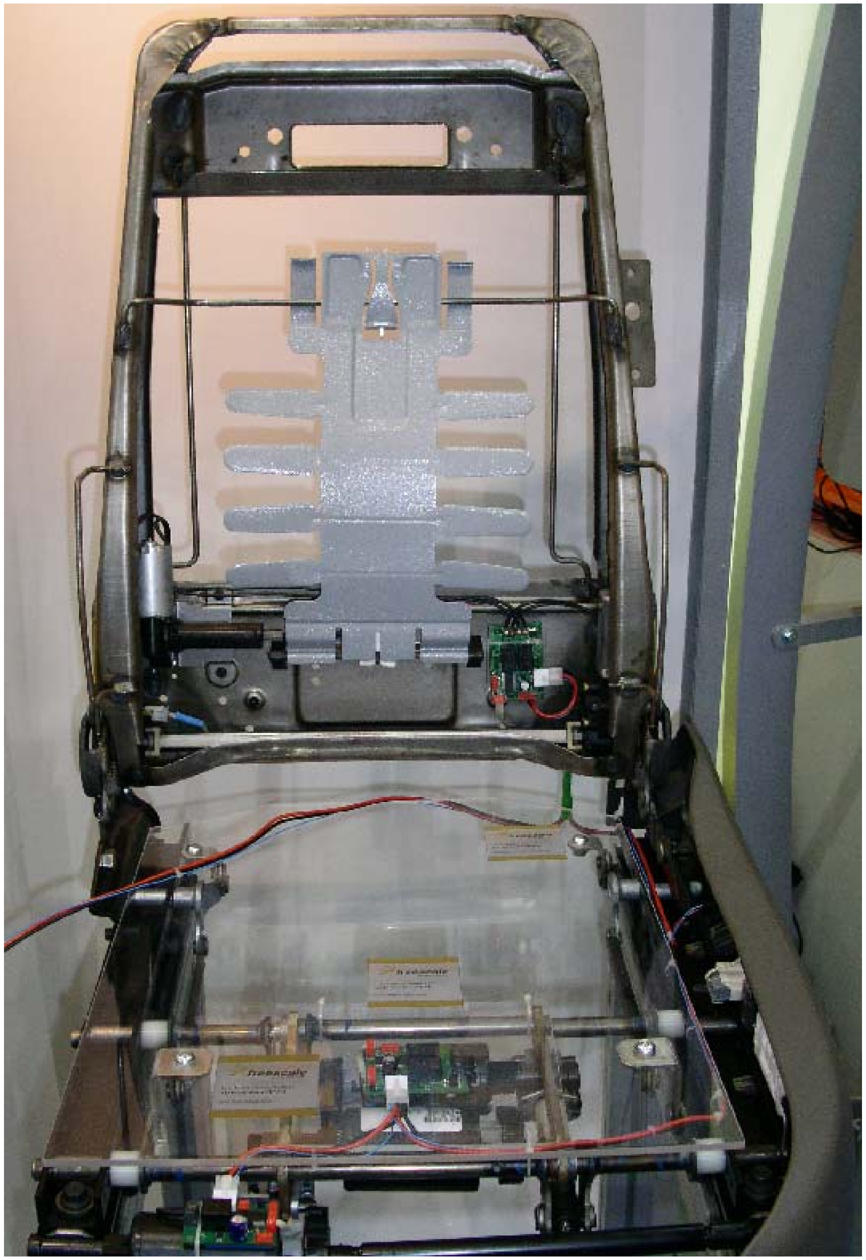
[Smolan]

Small Computers Rule The Marketplace

- **Everything here has a computer – but where are the Pentiums?**
  - And, they all want to be on a network

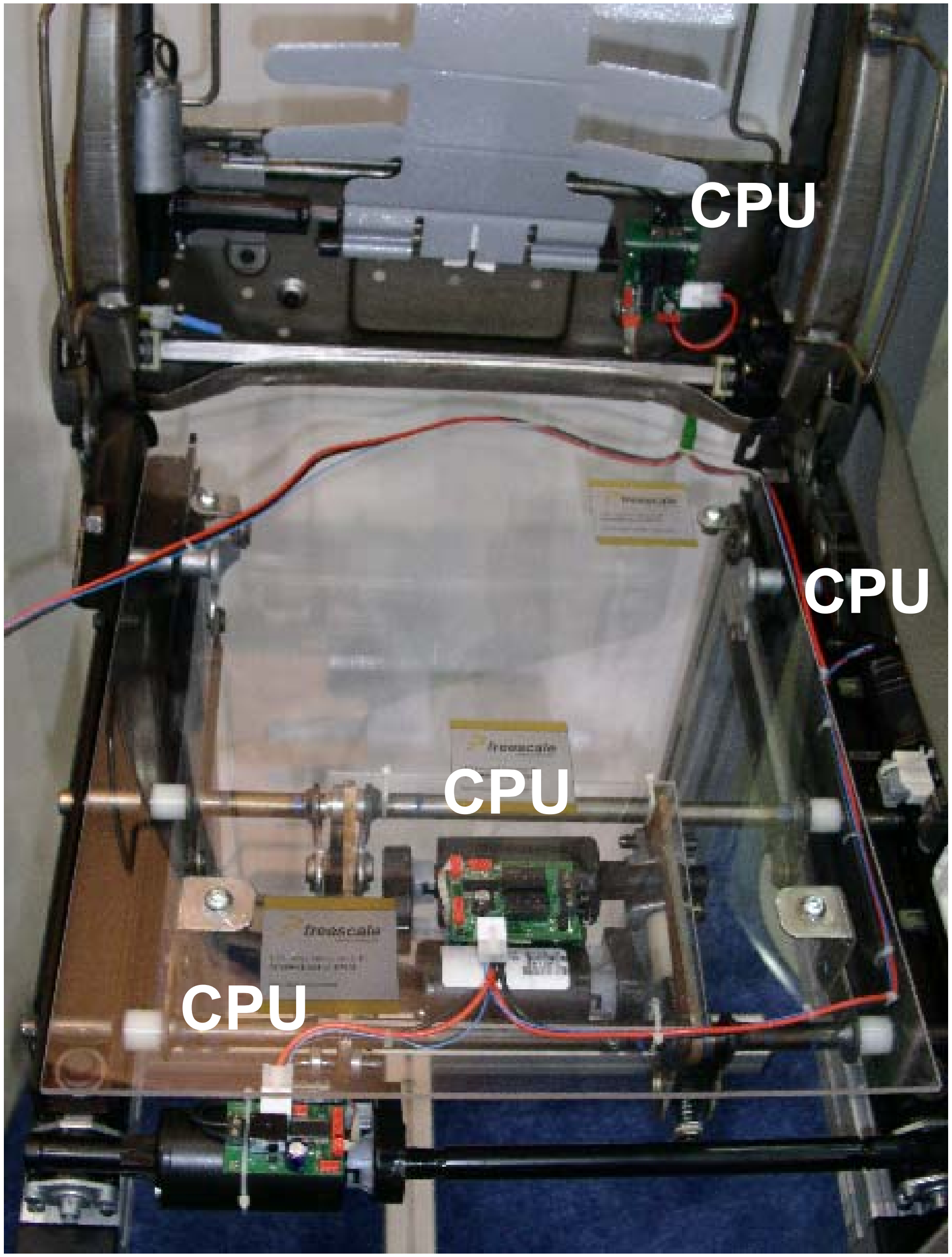
How Many CPUs In A Car Seat?

- **Car seat photo from Convergence 2004**
  - Automotive electronics show



Car Seat Network (no kidding)

- **Low speed LIN network to connect seat motion control nodes**
- **This is a distributed embedded system!**
  - Front-back motion
  - Seat tilt motion
  - Lumbar support
  - Control button interface



How Many CPUs In A Car? How Many Pentiums?

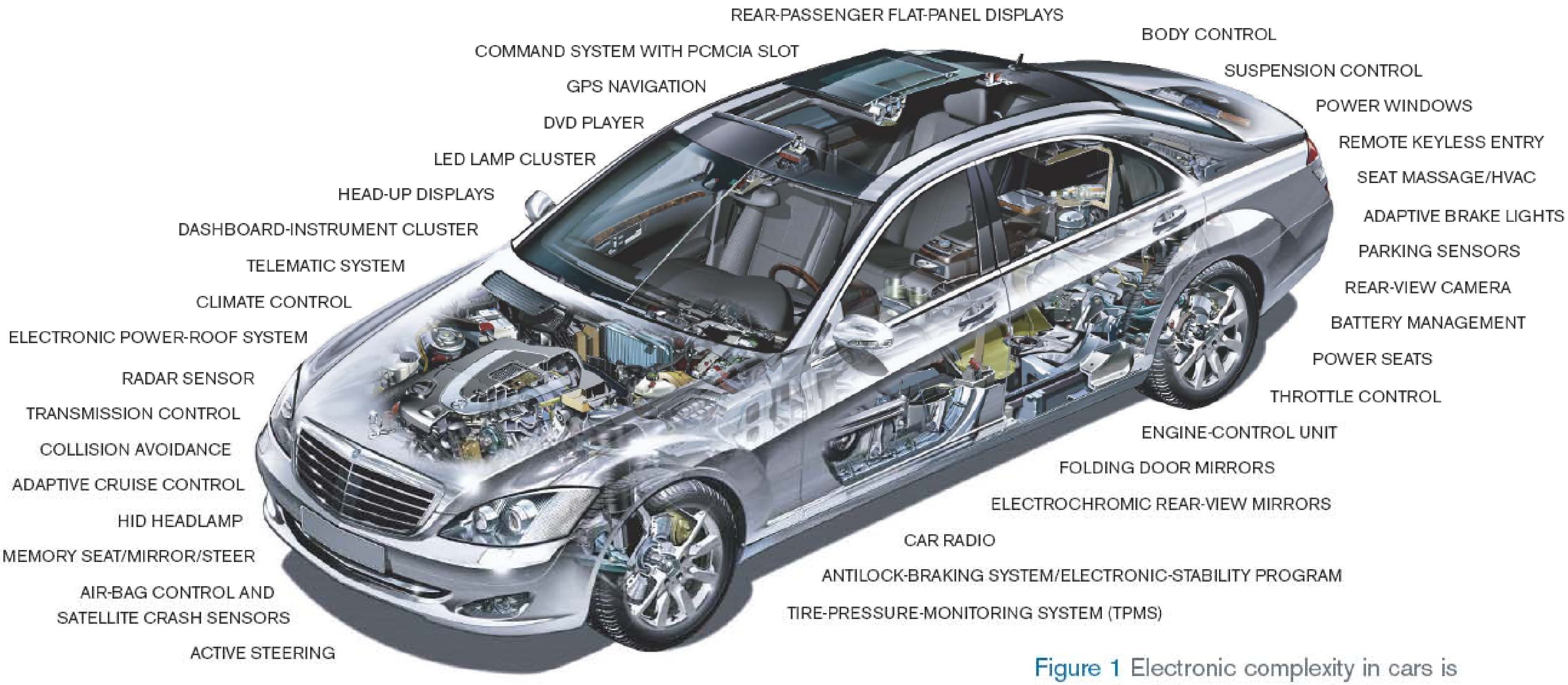
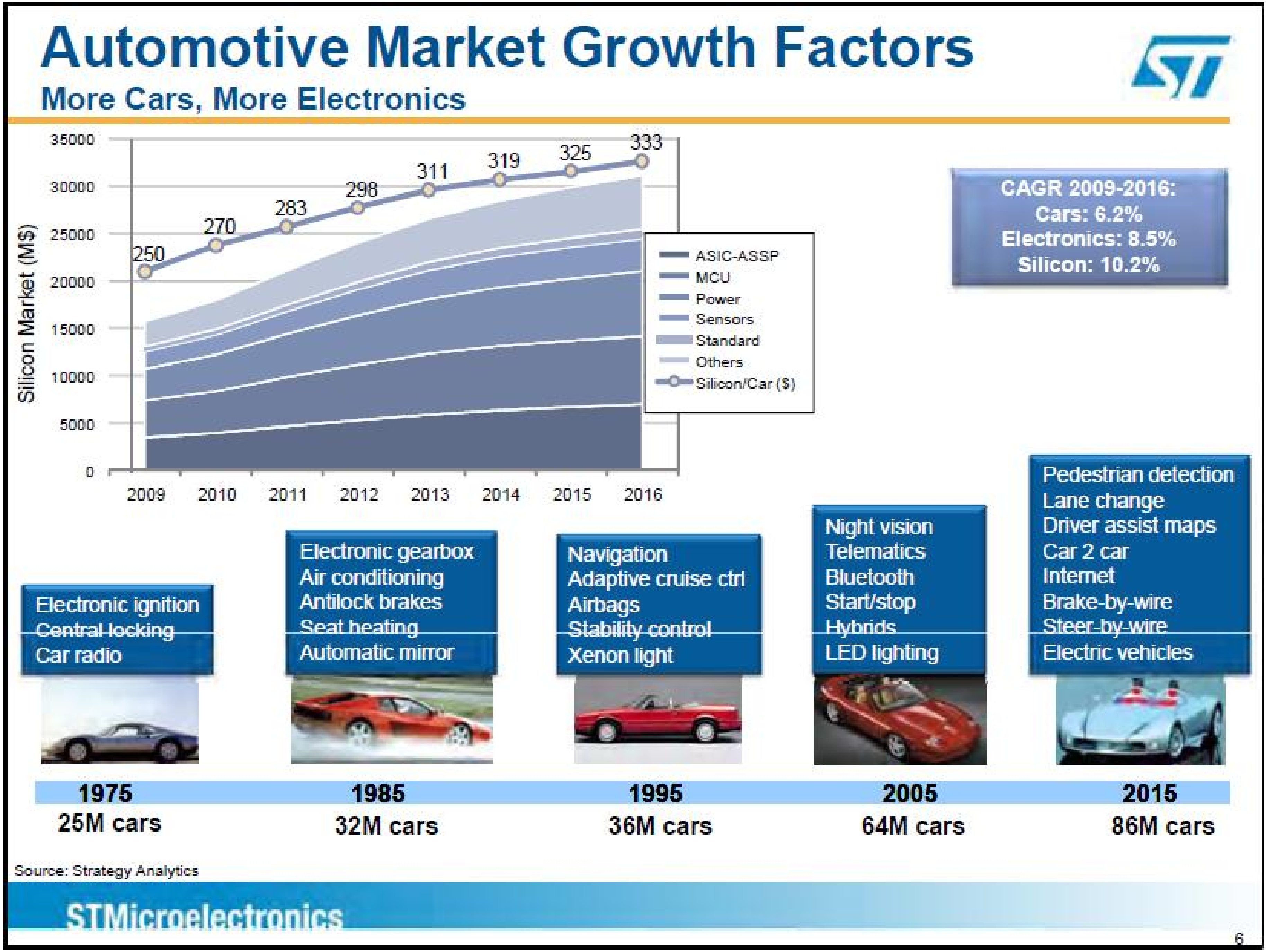
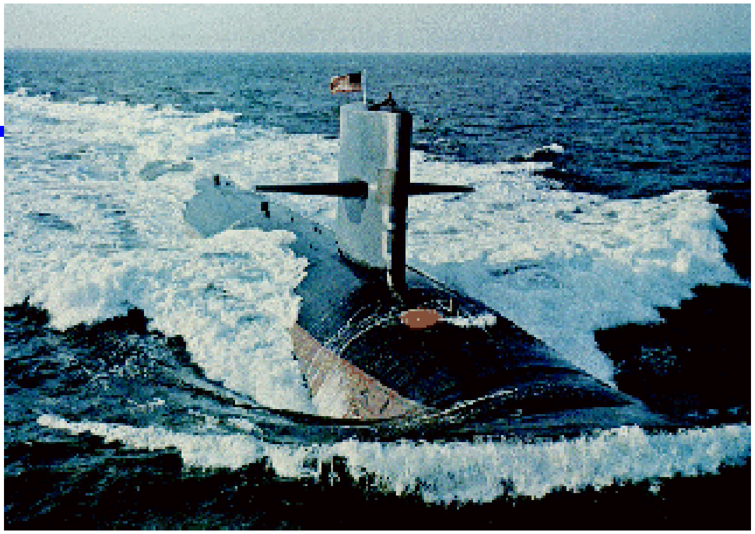


Figure 1 Electronic complexity in cars is increasing. New Mercedes S-Class cars employ at least 70 networked ECUs (electronic control units); 10 years ago, most cars had three ECUs (photo courtesy of DaimlerChrylser; source: Gartner Research, November 2005).

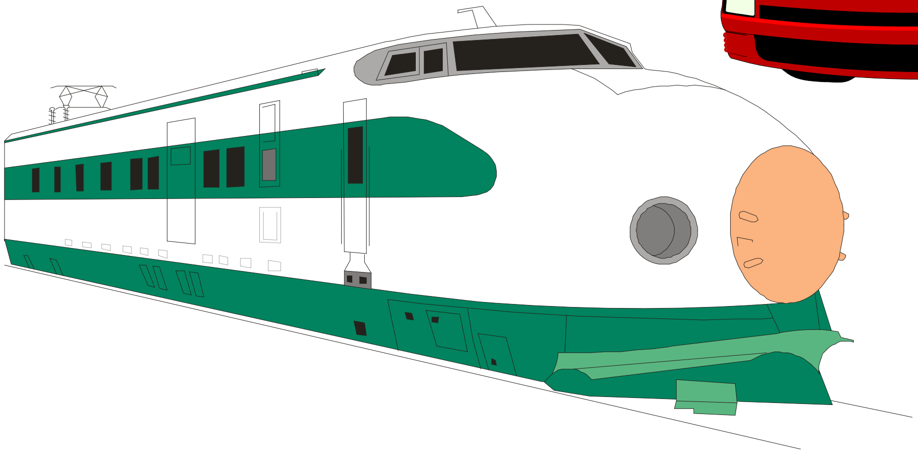
[Santarini06]

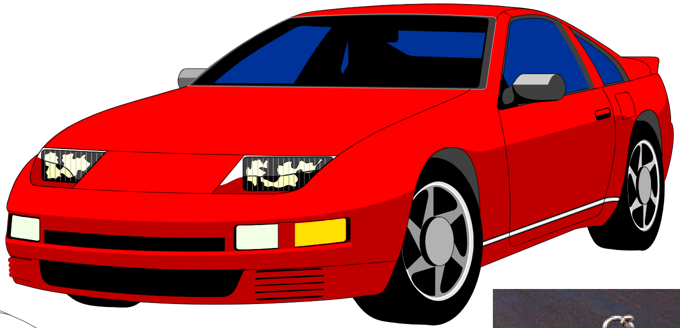


http://www.sec.gov/Archives/edgar/data/932787/10/000094787110000679/ss93129_6k.htm



Embedded System =
Computers Inside a Product





Definition of an Embedded Computer

- **Computer purchased as part of some other piece of equipment**
  - Typically dedicated software (may be user-customizable)
  - Often replaces previously electromechanical components
  - Often no “real” keyboard
  - Often limited display or no general-purpose display device
- **But, every system is unique – there are always exceptions**
- **Course scope focuses on distributed embedded systems, and not other embedded areas such as:**
  - Military systems: Radar, Sonar, Command & Control
  - Consumer electronics: set-top boxes, digital cameras
  - Telecommunications/DSP: cell phones, central office switches
  - Robotics
  - *However,* the engineering methods we teach are useful to those areas as well

Why Does This Course Have The Content It Has?

- **Based on experience from ~150 design reviews**
  - All sorts of embedded projects
- **Most common development teams and environments:**
  - Engineering domain experts: mechanical, electrical, auto, HVAC,…
  - Smallish team sizes: 1 to 25 developers
  - Embedded languages: C, C++, assembly, a little Java; no custom ICs
  - Small to medium projects: 1000-1M lines of code
  - Medium size production runs: 1,000-20,000 units
  - Product cost: $20 - $20,000
  - Old-school process models: Waterfall, Vee
  - Small systems had no RTOS, bigger systems had one
  - Senior designers in US; common to have China, India team members

- **But, encountered at least one of almost everything**
  - All-China team, all-Italy team, 100K+ units/year, 10 units/yr, agile methods, …

Design Review Approach

- **General approach:**
  on-site high level review of product

- **Pre-visit review of available documents (if any)**
  - Issue logs created before visit
  - Agenda tailored to best guess of risk areas (both reviewer & host opinions)

- **On-site review for 1 or 2 days**
  - Walk through issue logs
  - Discuss obvious risk areas
  - Use a risk screening checklist to hunt for additional risks
    - 100+ questions, but usually subsetted at discretion of reviewer to save time
    - Marked as:
      "red" / "yellow" / "green" / "not applicable" grades
    - Checklist evolved over time; early reviews did not use it

I. ☐ Implementation:

I.1. ☐R ☐Y ☐G ☐N ☐O Coding Standard |

I.2. ☐R ☐Y ☐G ☐N ☐O Language Use | |

I.3. ☐R ☐Y ☐G ☐N ☐O Static Code Analysi

I.4. ☐R ☐Y ☐G ☐N ☐O Design Margin | |

I.5. ☐R ☐Y ☐G ☐N ☐O Debugging and Perf Measurement |

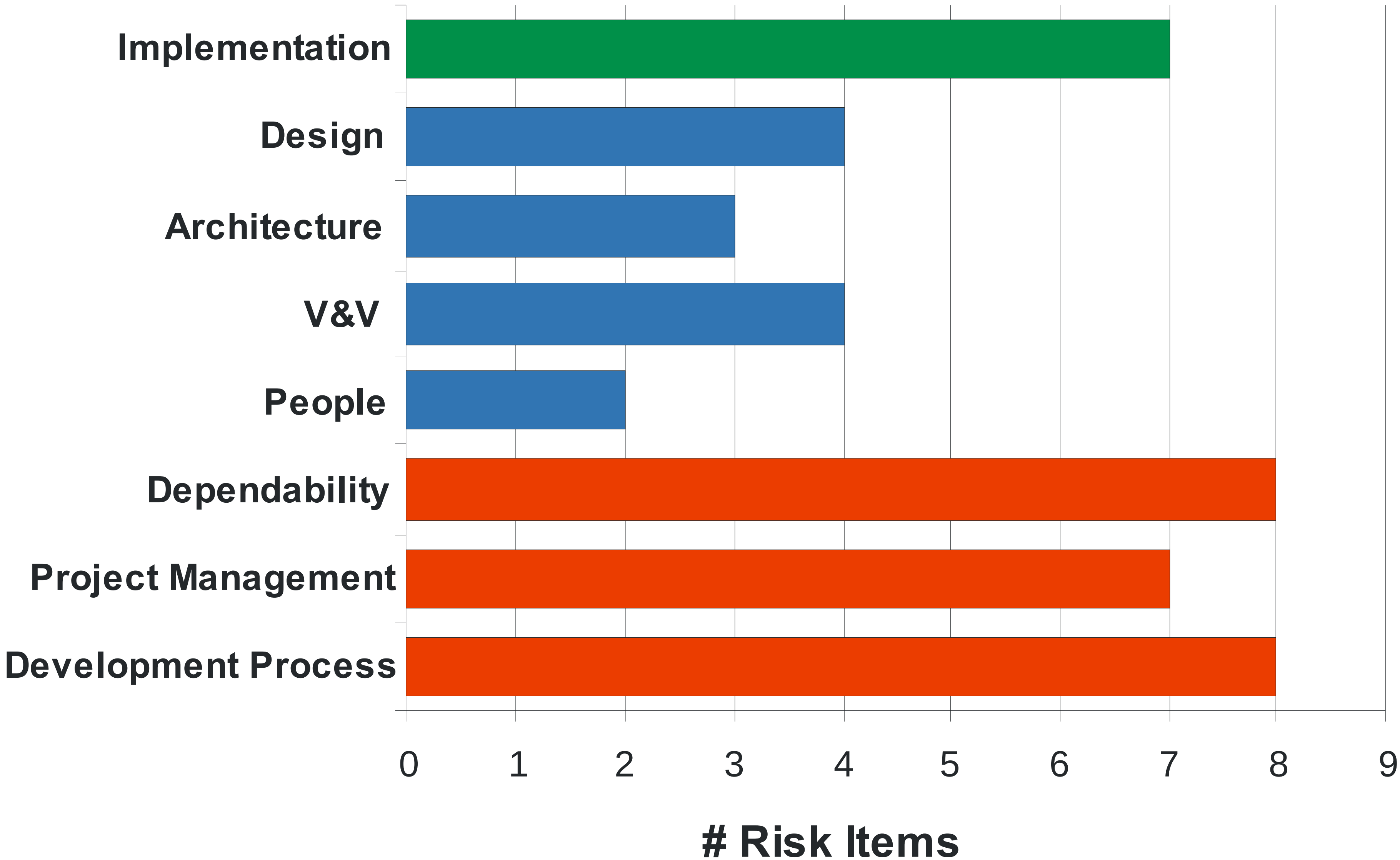
I.6. ☐R ☐Y ☐G ☐N ☐O Non-Volatile Memo

- **Review report**
  - Most important part of written report: red flag issues and how to fix them

Technical Risks

- **Most developers had little or no formal computer education**
  - Usually there was a senior developer who had learned the hard way
  - They were generally capable engineers … give them a book and they will learn
- **I expected to find lots of technical issues**
  - And yes, they were some, such as ignoring compiler warnings, but…
  - Not all that many rookie technical mistakes
  - Mostly problems with *complexity* or *advanced embedded knowledge*
    - E.g., Poor modularity
    - E.g., Ad hoc real time scheduling approaches
- **In general, technical problems:**
  - Corresponded with common holes in intro embedded textbooks
    - (Based on an informal survey of about 25 intro embedded texts)
  - Mostly were things that were hard to find in simple testing
    - In other words, most projects got the basic functionality right
    - And, most engineers can figure out embedded basics from a book

Risks In Management, Dependability & Process



Only about 1/6 of risk areas are problems with the code itself.

The 43 Red Flag Areas

1. Informal development process **Development Process**
2. Not enough paper
3. No written requirements
4. Requirements with poor measurability
5. Requirements omit extra-functional aspects
6. High requirements churn
7. No SQA function
8. No lessons learned mechanism

---

9. No defined software architecture **Architecture**
10. No network message dictionary
11. Poor code modularity

---

12. Design skipped or created after coding **Design**
13. Flowcharts used instead of statecharts
14. No real time schedule analysis
15. No methodical user interface approach

The 43 Red Flag Areas – Part 2

16.	Inconsistent coding style	Implementation
17.	Resources too full	
18.	Too much assembly language	
19.	Too many global variables	
20.	Ignoring compiler warnings	
21.	Inadequate concurrency management	
22.	Use of home-made RTOS	
23.	No peer reviews	Verif. & Validation
24.	No test plan	
25.	No defect tracking	
26.	No stress testing	
27.	Not enough attention on: reliability/availability	Dependability
28.	Not enough attention on: security	
29.	Not enough attention on: safety	
30.	No/incorrect use watchdog timers	

The 43 Red Flag Areas – Part 3

31. Not enough attention on: system reset
32. No run-time fault instrumentation
33. No software update plan
34. No IP protection plan

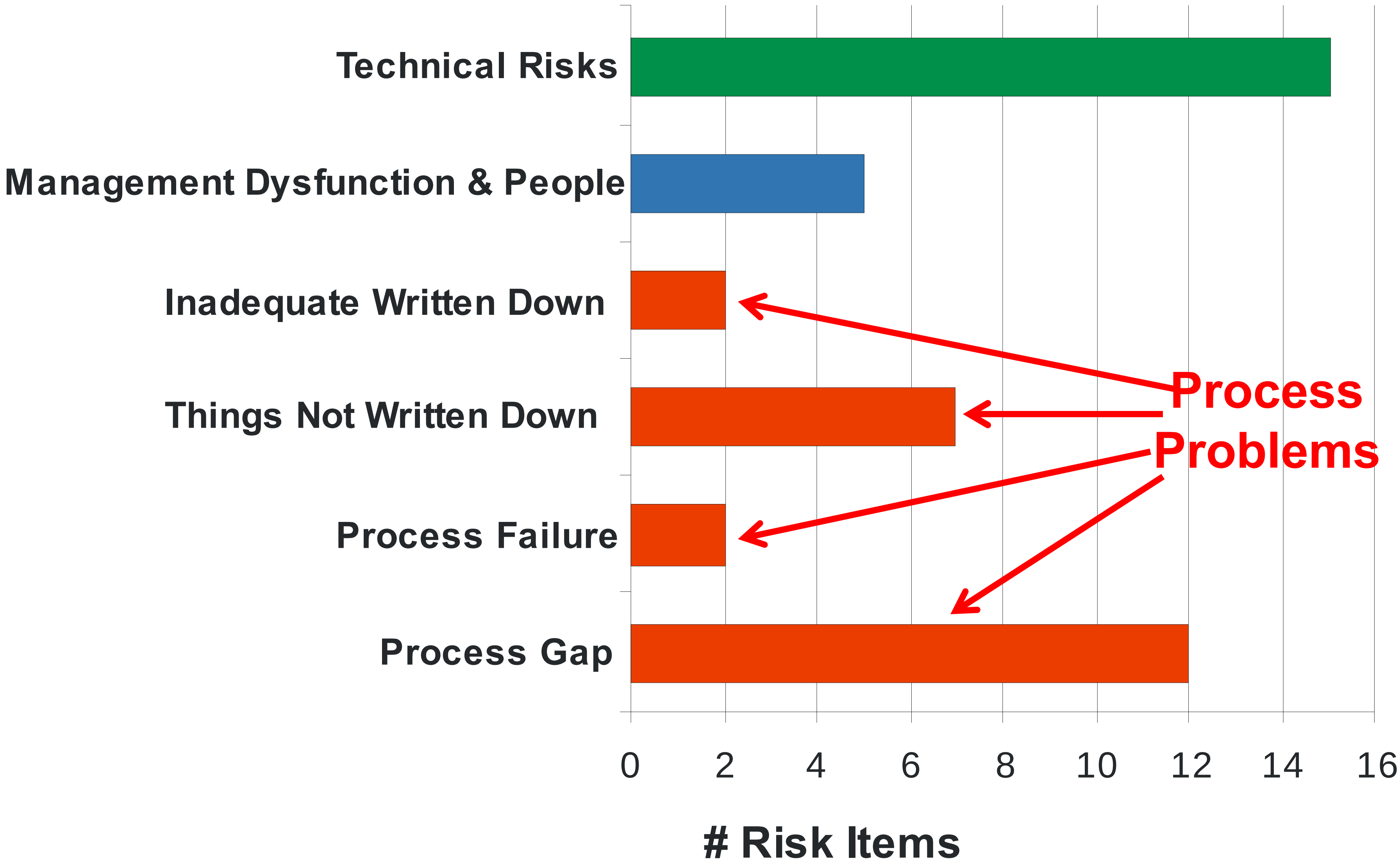
Project Management

35. No version control
36. No version management plan
37. Use of cheap tools instead of good tools
38. Schedule not taken seriously
39. Managers act as if software is free
40. Risks from external tools and components
41. Disaster recovery not tested

People

42. High turnover and developer overload
43. No training for managing outsource relationships

Most Risks Are Technical Risks or Process Gaps



Only about 1/3 of risk areas are technical

The Big Problems Are Process Gaps

- **Process Gaps – things developers didn't attempt to do**
  - E.g., no SQA function, no SW update plan, no security plan
  - In some cases they didn't appreciate importance of these activities
  - In other cases it never occurred to them that these things were relevant
- **Missing paper – things developers didn't write down**
  - (These are a special type of process gap)
  - E.g., no written requirements, no software architecture, no design
- **Process *failures* weren't that common**
  - Relatively few "tried and failed"
  - **Mostly "didn't try at all" and "didn't know they should be trying"**
  - In other words, developers didn't even know they should worry about the areas that were presenting the biggest risks
- **The technical risks are what you'd expect – advanced embedded stuff**
  - Concurrency, scheduling, and so on, not "how does an A/D work"

The 18-649 Approach

- **Experience a well defined process with medium-light weight paperwork**
  - Includes all phases of real projects through beta test
  - Really, most industry practices for big projects has more paperwork
  - Even (good) embedded agile teams have at least as much paperwork as we use!

- **Exposure to basic techniques that will work in most projects**
  - UML-lite approach with many examples gets teams on the right track
  - Knowing how to use a simulator helps with system design choices
  - Testing frameworks make it easier to do thorough testing
  - Version control (you are on your own to pick one, but pick one and use it!)

- **Point of the project is to give you a realistic design experience**
  - Fancy elevator functions are fun and you can do that
  - **But it is more important to design a rock-solid elevator than a fancy one**
  - What you experience in this course is what many industry companies try to achieve (but often only *after* they have had to clean up a software disaster)

Where Are We Now?

- **Part 1 of this course: embedded system design**

- **Where we're going today:**
  - General discussion of embedded system foundations
  - Fundamental concepts & definitions
    - Time constants
    - What makes something distributed
  - Topics that matter in embedded systems

- **Where we're going next:**
  - Elevators as an example embedded application
  - Methodical design ("hacking code" doesn't cut it in embedded industry)

  - Part 2 of course: embedded networking
  - Part 3 of course: dependable & safe system design

Preview

(Some of this is familiar from 18-348/18-349; many students here did their undergrad elsewhere, so this lecture has some gap-filling material)

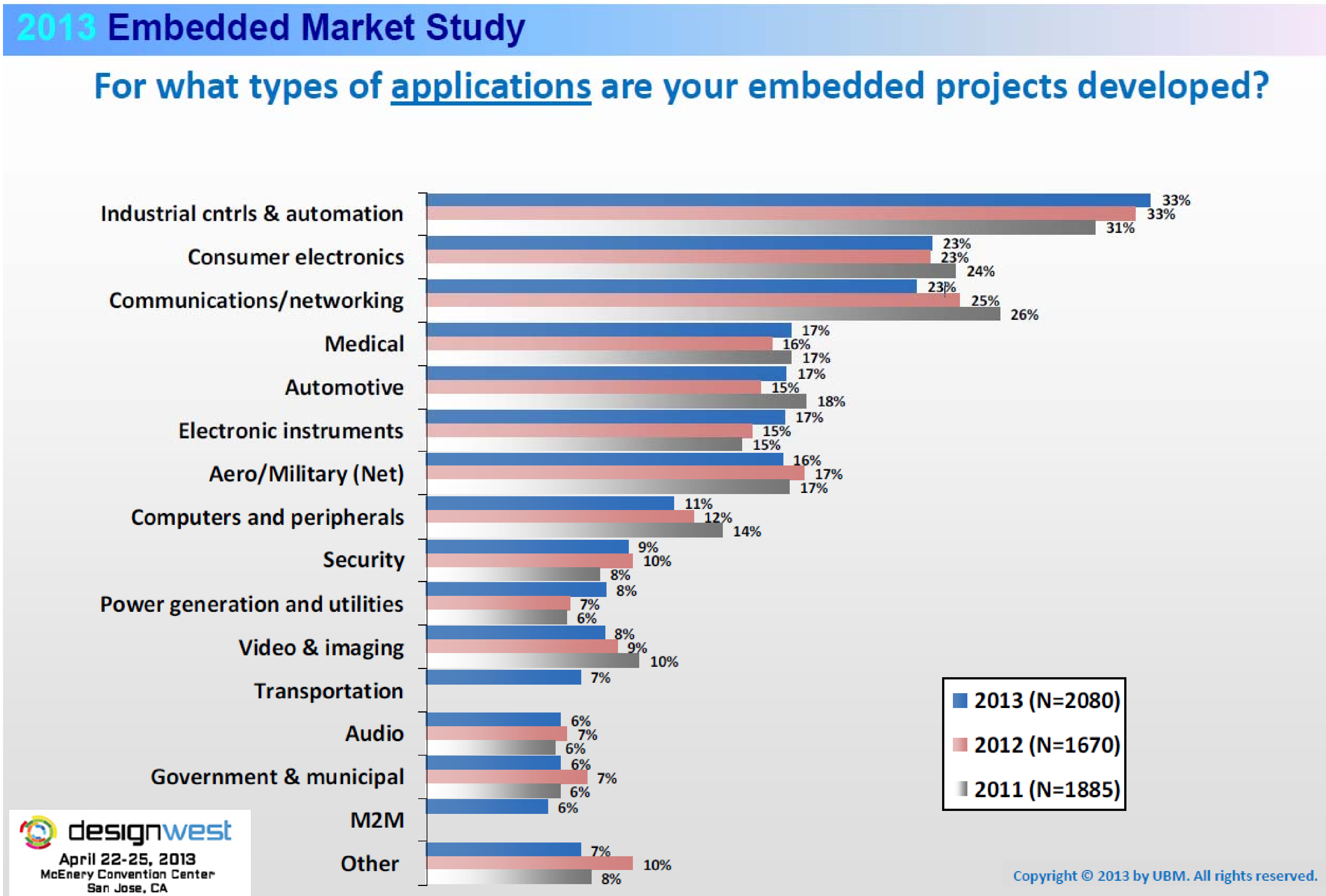
- **Embedded computing overview**
  - What's an embedded computer
  - General types of embedded computing
- **Overview of areas covered by this course**
  - Automotive "x-by-wire" is a useful example application for discussion
- **Control loop issues**
  - System latencies & time constants

Required Reading For This Lecture

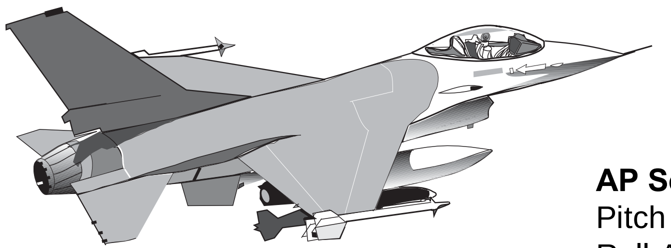
- **Ebert & Jones: Embedded SW: Facts, Figures, Future**
- **Text Ch. 2: Written Development Plan**
  - Warmup thinking for course project – there is more to product development than hacking code
  - This semester we're going to walk you through an end-to-end project
  - I know many of you are skeptical about the need for "documentation"
    - Most students think that this stuff is useful by the end of the course. Some don't.
- **Text Ch. 3: How Much Paper is Enough?**
  - Course project will emphasize methodical but relatively light-weight paperwork
- **Questions on readings will be included in the tests (up to 20% of test) :**
  - Main points of each assigned reading
  - They are often (but *not solely*) the boxed text in the book
  - They are often the main points of papers being read
  - They are *not* 100% the same as the lecture notes

There Are Many Application Areas

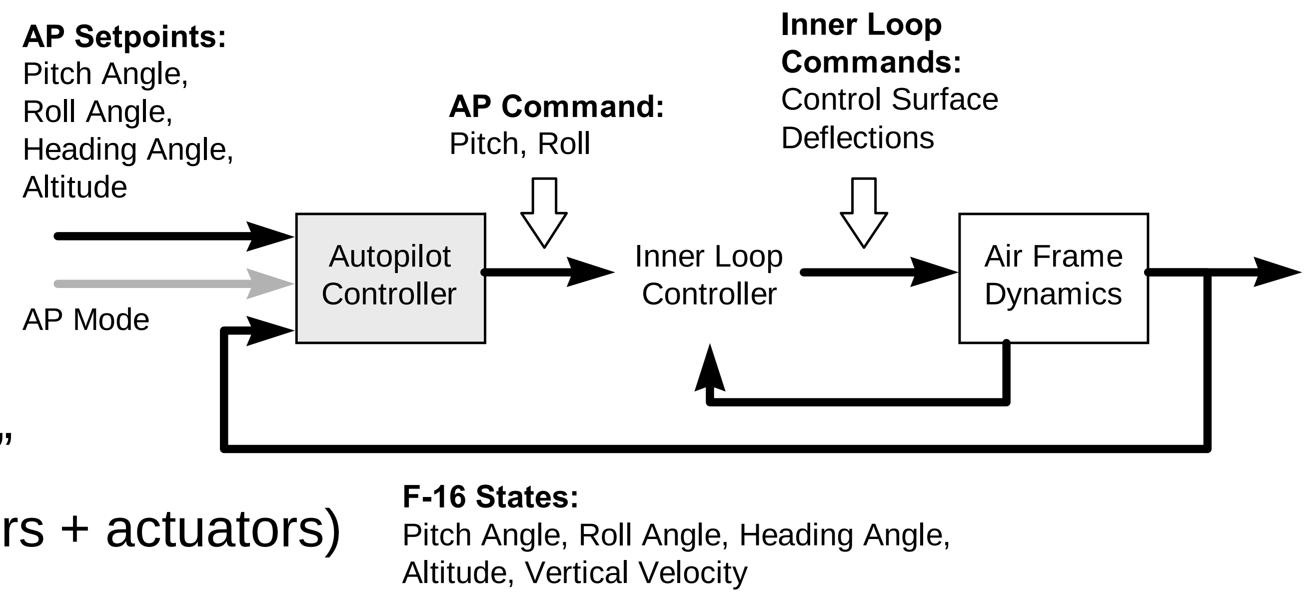
- [UBM 2013]



Elements of Embedded Systems



E.g., flight control systems



[Krogh]

- **System =**
  - Computer
  - + Controlled "plant"
  - + Plant I/O (sensors + actuators)
  - + Operator
  - + Operator interface
  - + Physical environment
- **Design engineer's job:**
  - Make it work correctly (functionality + real-time) and safely
  - Make it meet real world constraints (size, power, weight)
  - Optimize for: cost, performance, convenience, *etc.*

Common Types of Embedded System Functions

- **Control Laws**
  - PID control, other control approaches
  - Fuzzy logic
- **Sequencing logic**
  - Finite state machines
  - Switching modes between control laws
- **Signal processing**
  - Multimedia data compression
  - Digital filtering
- **Application-specific interfacing**
  - Buttons, bells, lights,…
  - High-speed I/O
- **Fault response**
  - Detection & reconfiguration
  - Diagnosis

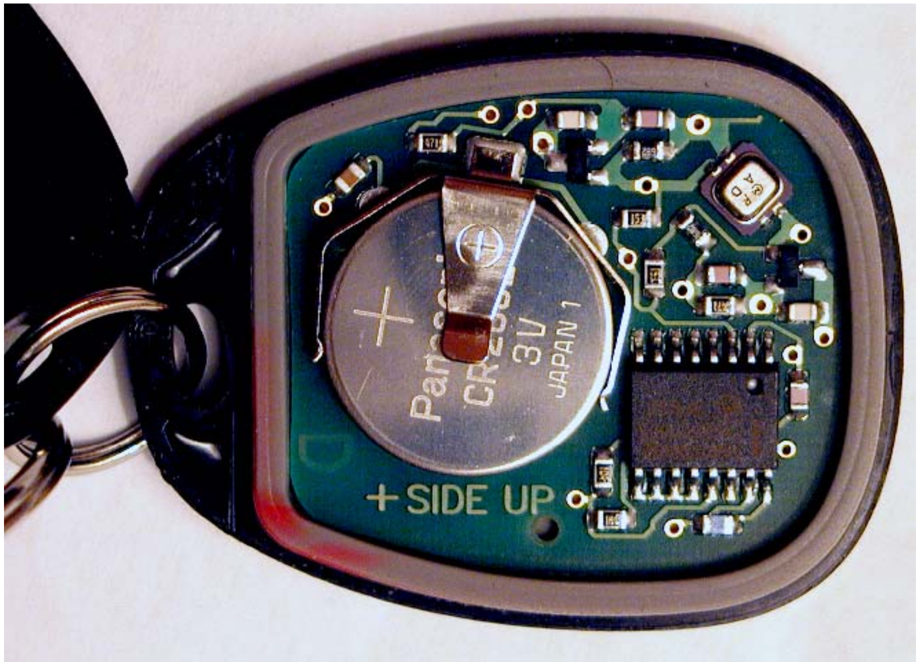
PW-4000 FADEC
(Full Authority Digital
Engine Controller)

[P&W]

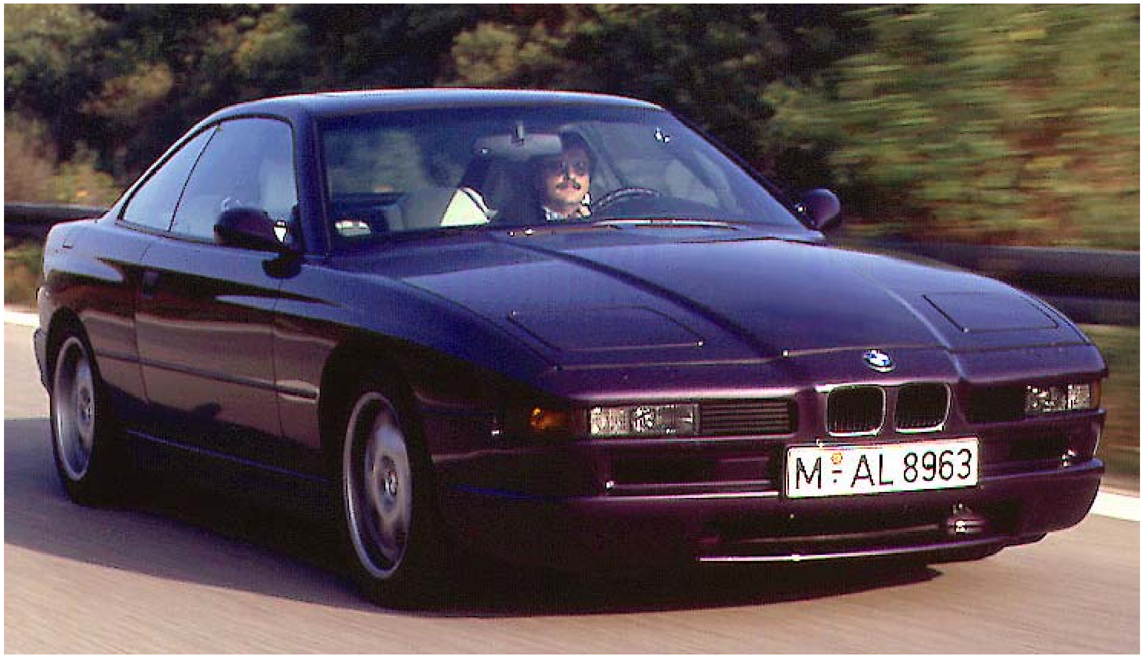
Typical Embedded System Constraints

- **Small Size, Low Weight**
  - Hand-held electronics
  - Transportation applications -- weight costs money
- **Low Power**
  - Battery power for 8+ hours (laptops often last only 2 hours)
  - Limited cooling may limit power even if AC power available
- **Harsh environment**
  - Power fluctuations, RF interference, lightning
  - Heat, vibration, shock
  - Water, corrosion, physical abuse
- **Safety-critical operation**
  - Must function correctly
  - Must *not* function *in*correctly
- **Extreme cost sensitivity**
  - $.05 adds up over 1,000,000 units

Lear Encrypted Remote Entry Unit

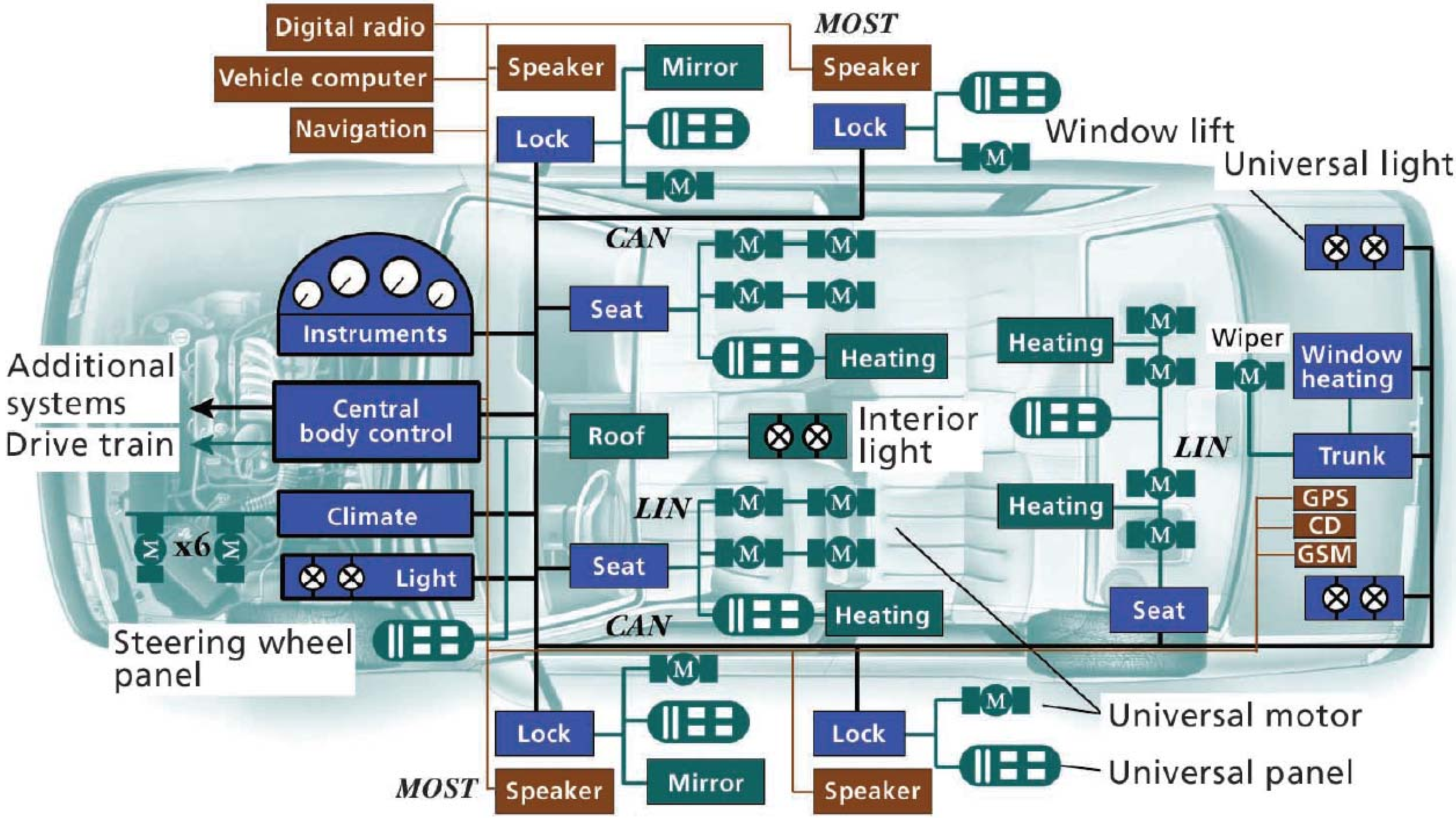


A Customer View



- **Reduced Cost**
- **Increased Functionality**
- **Improved Performance**
- **Increased Overall Dependability**
  - (Debatable, but can be true)

An Engineering View

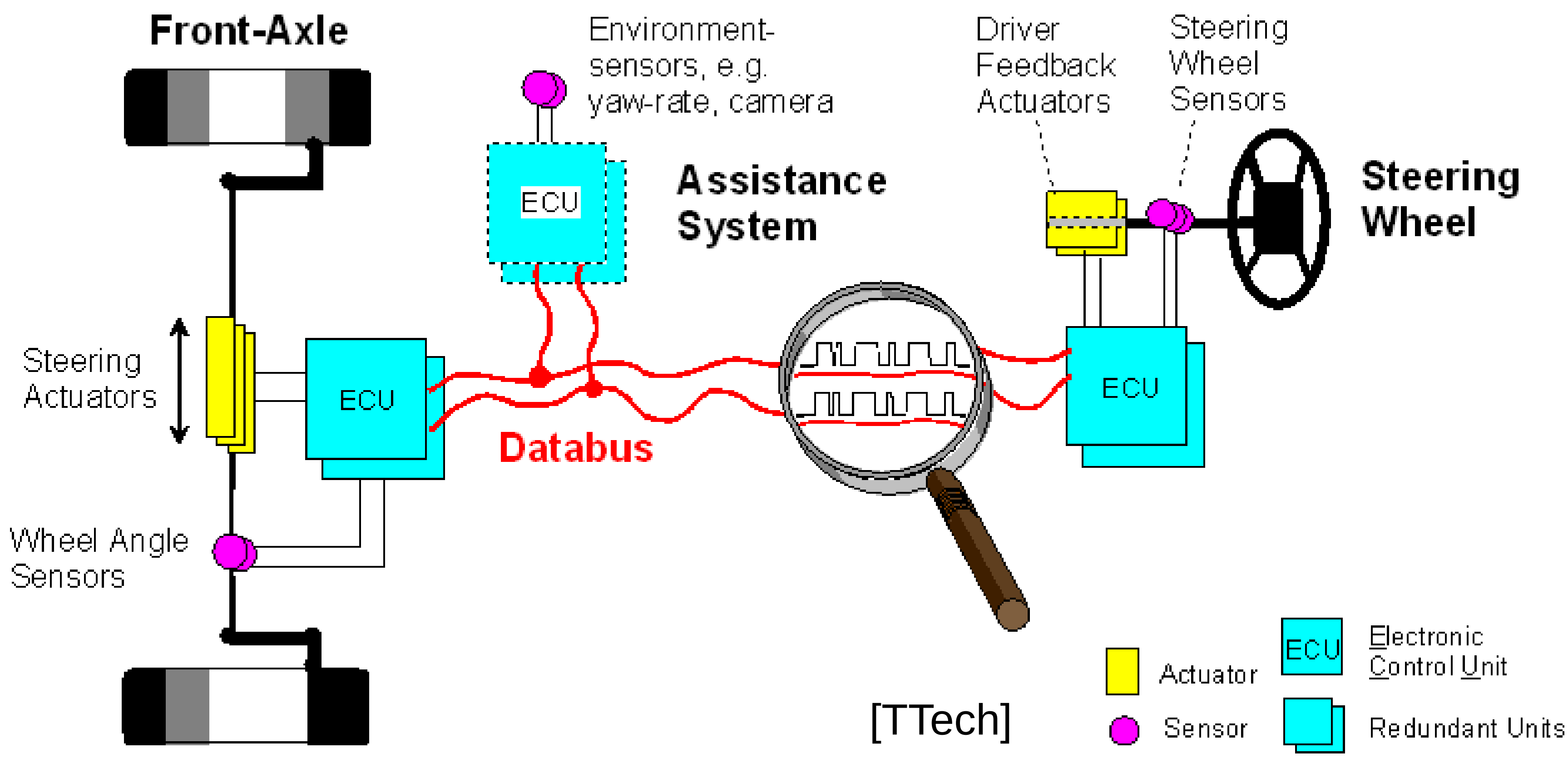


CAN Controller area network
GPS Global Positioning System
GSM Global System for Mobile Communications
LIN Local interconnect network
MOST Media-oriented systems transport

[Leen02]

X-by-Wire As Topic Motivation

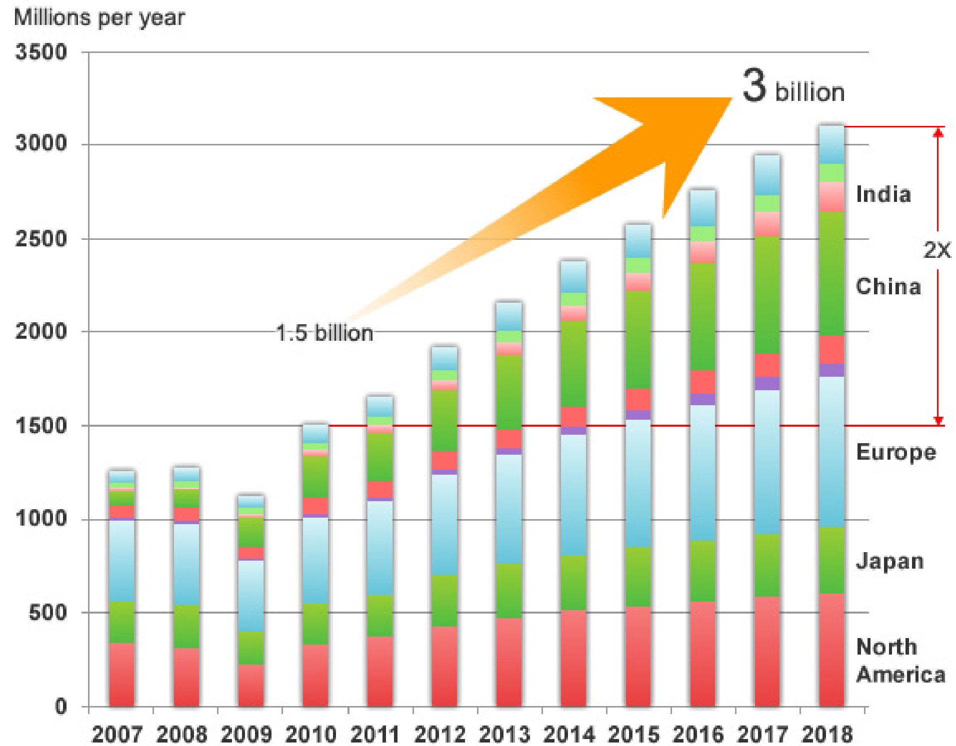
- **X-by-Wire is perhaps the ultimate automotive computer technology**
  - All embedded computers in automobile will probably interface to it
  - Has the most stringent requirements
  - This course looks at what it takes to do X-by-Wire (and others)



[TTech]

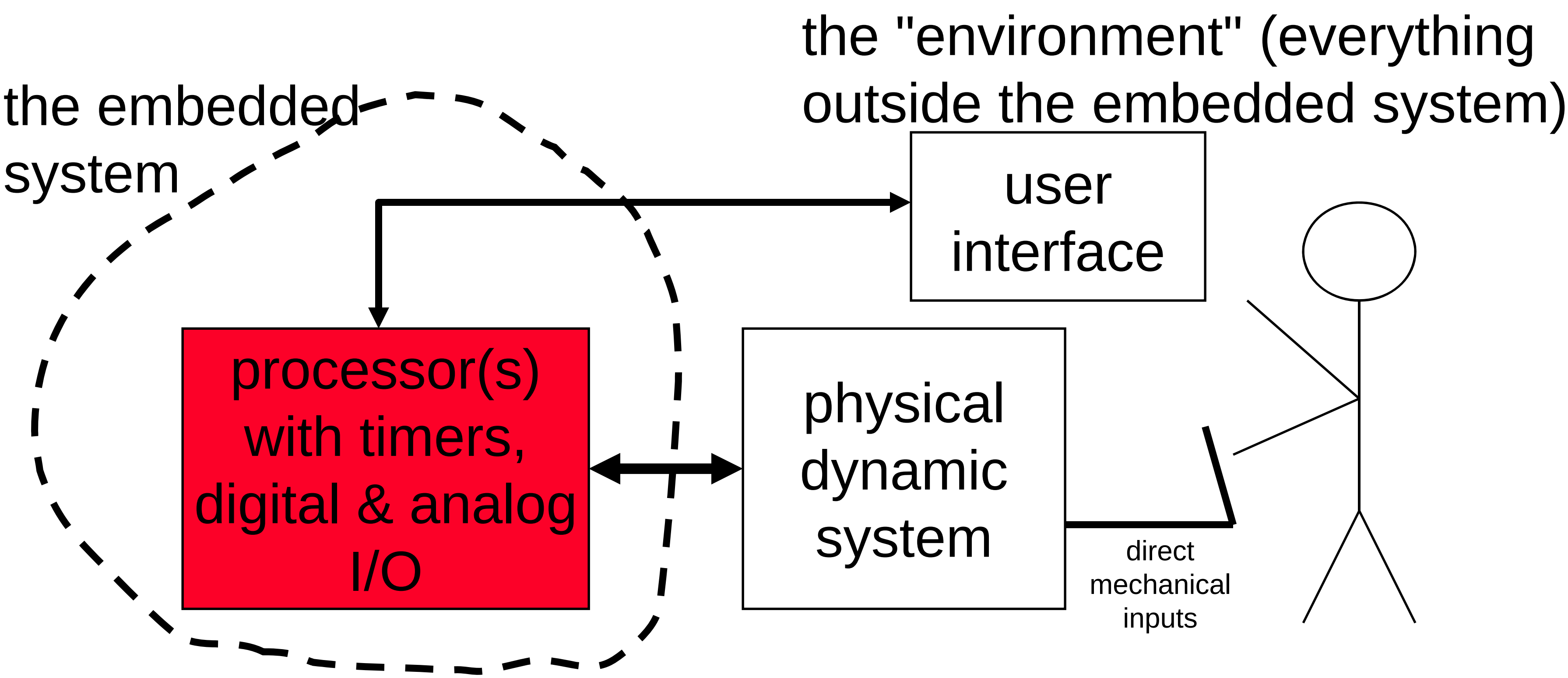
World Automotive Electronics Market

- **Electronics already a big part of vehicle cost**
  - Perhaps $1500 of OEM cost (estimates vary)
  - Expected to increase annually to perhaps 25% o vehicle cost
- **X-by-Wire projected to be a key technology**
  - Throttle-by-wire is already common
  - Brake-by-wire is being introduced
  - Self-driving cars are putting pressure on increasing X-by-wire



Projected worldwide sales of automotive MCUs (by volume)
[Renesas 2013]

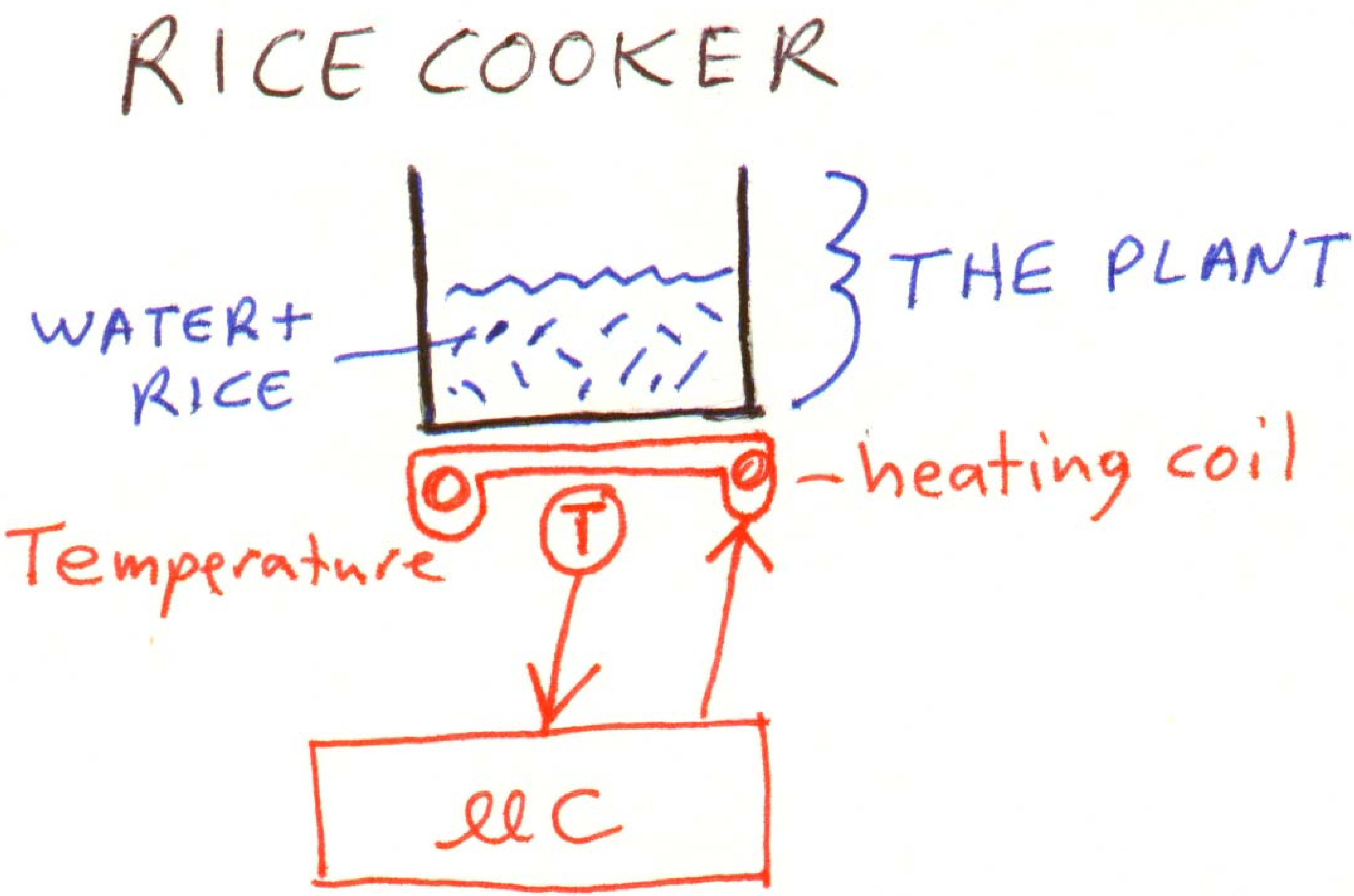
What's "Real" in Real-Time Embedded Systems?



The (real) environment determines the constraints on:

- **sampling rates**
- **computation time**
- **jitter (random variations in timing)**

Example Simple Control System



Control Loop Elements

Symbol	Meaning	Notes
d^{object}	Object Delay	Actuator to sensor lag through the object
d^{rise}	"Rise" Time	Time constant of system
d^{sample}	Sampling Period	$d^{sample} < (d^{rise} / 10)$
$d^{computer}$	Computation Delay	$d^{computer}$ ƒ d^{sample}
$G\,d^{computer}$	Jitter of Computation Delay	$G\,d^{computer} << d^{computer}$
$d^{deadtime}$	Dead Time (control loop total delay)	$d^{deadtime} =$ $d^{sample} + d^{computer} + d^{object}$ (worst case)

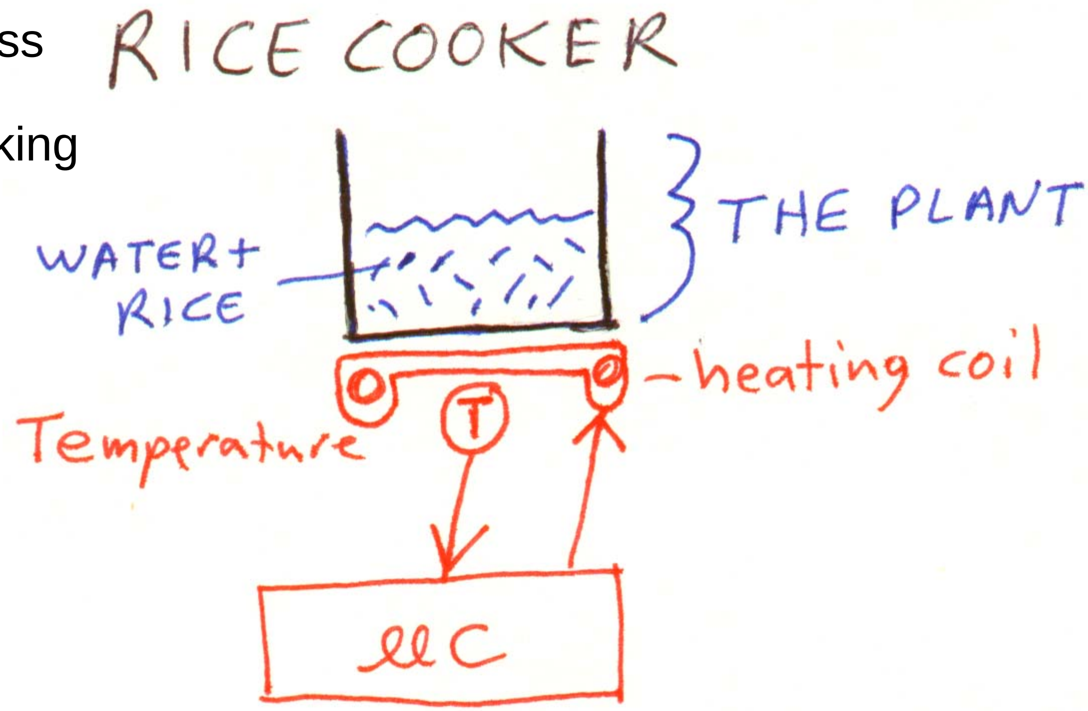
- **In rice cooker example:**
  - "Object" = water + rice + cooking pot
  - "Sensor" = temperature sensor
  - "Actuator" = heating coil
  - Rise time in this case is how long it takes to heat up by a desired temperature step size (e.g., 3 degrees). In some systems could be "fall" time instead.

Control Timing Element Definitions

- **d^{object} controlled object delay**
  - Delay from applying control force to first observed response
  - Due to inertial lag of physical plant (speed of thermal wavefront in rice cooker)
- **d^{rise} rise time of step response**
  - Physical time constant of system (thermal mass of rice+water+pot)
- **d^{sample} sampling period**
  - How often temperature sensor is read (should be **> 10x faster than d^{rise})**
  - **Want to run control loop 10 times faster than system "time constant"**
- **$d^{computer}$ computer delay**
  - Time to compute new actuator command point (sensor $\rightarrow$ heater on or off)
- **$\Delta d^{computer}$ jitter of computer delay**
  - Variations in computer delay (*e.g.*, cache misses, competing tasks)
- **$d^{deadtime}$ dead time**
  - End-to-end latency from observation to action (lower = more stable)
  - Worst case is: wait for next sample; compute; wait for object delay
  - If Dead Time is too large, system will be unstable even for fast sampling"

Rice Cooker Example

- d^{object} **=> heating coil to T sensor**
  - Guess 5 seconds
- d^{rise} **=> say increase by 3 degrees**
  - Varies depending on water mass
  - Varies depending on desired temperature stability while cooking
  - Guess 10 seconds
- d^{sample} **= 10 seconds / 10 = 1 second**
- $d^{computer} <= d^{sample}$
  - $d^{computer} <= 1$ second
  - Say it's 900 msec
- $\Delta d^{computer} << d^{computer}$
  - Let's say 100 msec jitter << 1 second
  - Really what you need is $(d^{computer} + \Delta d^{computer}) < d^{sample}$
- $d^{deadtime} = d^{sample} + d^{computer} + d^{object}$
  - $d^{deadtime} = 1 + 1 + 5 = 7$ seconds
  - Slow computer is fine (d^{object} is the limiting factor to performance)
  - Want good control algorithm to avoid temperature overshoots



Computers Creep Into Applications

- **Usually adding computers is an incremental process**
  - Diagnostic equipment
  - Add-on accessories/peripheral equipment
  - Routine tasks (data logging)
  - Suggestions to operator & “smart” alarms
  - Servo loop closures
  - Complete automation with human operator
  - Autonomous operation

- **There has to be a business reason to use computers**
  - Cars adopted them for emission controls
  - Elevators use them to do fancy dispatching and load management
  - Aircraft engines use them for weight and fuel efficiency
  - Dishwashers use them to provide hi-tech look & advanced features

Historical Example: Cars

- **Almost 1 million lines of code in some car**
  - 70+ CPUs in a luxury car
- **Engine controller**
  - Hard real time (ignition cycle)
  - Fail safe
  - 32-bit CPU, resource adequate
- **Transmission controller**
  - Soft real time (shift points)
  - Fail safe
  - 8-bit to 32-bit CPU, resource marginal
- **Anti-lock Braking System (ABS)**
  - Firm real time (pulses brake pedal
  - Fail operational (for brakes);
    Fail safe (for electronics)
  - 8-bit CPU, resource constrained
- **Trend: drive-by-wire; autonomy**

1970 Mustang

1996 Mustang

(Purple bundles connect to computers;
Note large alternator to supply electricity

2006 Mustang Photo

- http://www.mustang50magazine.com/featuredvehicles/m5lp_1001_2006_mustang_gt/photo_07.html

Actual Student Comments

18-649 Distributed Embedded Systems, selected 2008-2013 anonymous feedback from course evaluations:

- “This course helped me to experience real industrial-like [project], and practical matters. What I learned from this course will definitely help me in my future as an engineer.”
- “Having a well-established course plan and rigid schedule, but at the same time providing flexibility to group members are signs of a well-balanced teaching style. Course material is a balance of technical and nontechnical content with a emphasis on pragmatism. There is no course that prepares students for the real world [better] than this one.
- “Extremely useful course, especially for interviews.”
- “Strength: Takes you through all the stupid things that a embedded company wants you to do. Weakness: I don't think how relevant it is to do those stupid things.”
- “This class had the some of the best lectures I've had in college. It was truly informative and taught well. My only complaint would be with the project.”
- “This course did a good job teaching and demonstrating design through the project”
- “Near- Industrial work experience is what I value in this course”
- “If the point of the class is to make us hate redoing documentation, thereby making us try harder to get it right the first time, then mission accomplished.”
- “Best structured course ever.”
- “A great course on software engineering. Gave me a good perspective on how to write good systems, which is different from most software course that only teach how to pick good algorithms.”

Lecture Review

- **System includes many pieces, including the user**
  - Latency & time constants are critical for stable control loops
  - Various time constant definitions & how to estimate them are the underpinnings of timing for embedded control systems
- **Distributed embedded systems require knowledge in many areas:**
  - Embedded computing
  - Distributed systems
  - Embedded real-time networking
  - End-to-end real time scheduling
  - Dependability (including security)
  - Safety
- **Test might include the following topics**
  - Know and apply Control Timing Element Definitions to an example
  - Recognize some of the Typical Embedded System Constraints
  - Know the course policy on cheating, including penalty
  - Topics from required reading