CPEN 432
Real-Time Systems Design

Bader Alahmad
University of British Columbia

Course information

- Course website: cpen432.github.io
- Lectures will be delivered through zoom and will be recorded
  - Links to zoom lectures on course Canvas page: https://canvas.ubc.ca/courses/63247
- We will use **Piazza** for all course-related discussions
  - **Piazza signup link:**
    https://piazza.com/ubc.ca/winterterm22021/cpen4322012020w
    Sign up access code: **A6FG90**
  - **Piazza course page:**
    https://piazza.com/ubc.ca/winterterm22021/cpen4322012020w/home
- Instructor's office hours
  - **Time:** TBD. Location: **Online**
  - Office hours start the week of **Jan 18**
- Teaching Assistant:
  - **TBD**; Office hours: **TBD**, Location: **TBD**

Reading material

- **Textbook:** *Giorgio C. Buttazzo, Hard Real-Time Computing Systems, 3rd ed.*
  - Electronic version available for download to UBC students at UBC library's webpage

- Recommended material
  - Real-Time Systems, Jane Liu, Prentice Hall, 2000
  - An Introduction to Real-Time Systems, Raymond Buhr & Donald Bailey, Prentice Hall, 1998
  - Programming with POSIX Threads, David R. Butenhof, Addison Wesley, 1997
  - Computers as Components, Wayne Wolf, Morgan Kaufman, 2005
  - Real-Time Systems and Programming Languages, Alan Burns & Andy Wellings, Addison Wesley, 2009

Course organization

- **Programming assignments [50%]**
  - Assignments to be completed in groups of ***at most 4***
    - Will use **GitLab**
- **Written assignments [30%]**
  - Individual work
  - Must be typeset, preferably **LaTeX** (recommended: **ShareLatex)**
  - Submission through **Gradescope**
- **Final Examination (take home) [20%]**
  - In the same spirit of the written assignments
  - You might be asked about project material

Expected background

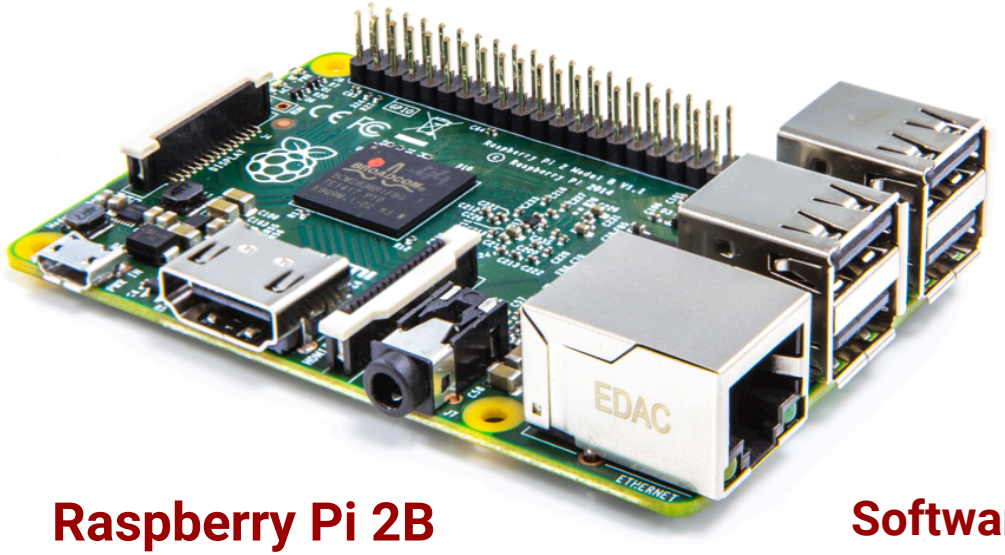
- **Understanding of operating systems**
  - CPEN 331 → Concepts + Implementation
- **A fair amount of experience with C programming**
  - Java, Python, etc., are **not** enough, you need "**pointers**"
- **Basic knowledge of assembly-language programming**
  - Concepts, not the ISA details
- **Algorithms design and analysis**
  - Greedy, DP, …
  - Proving correctness, optimality, running-time analysis
  - Basic complexity: NP-Completeness, pseudo-polynomial time, etc
- **Some probability**

Useful Habits

- Regular reading
- **Early start on assignments**
- Learn from manuals
  - Programming assignments will involve hard-core embedded programming
  - You will be provided with all resources
    - But you will have to read them on your own!
- *Will **not** lecture on embedded programming!*

Programming Assignments - Hardware

Raspberry Pi 2B



TTL-to-USB cable

Raspberry Pi 2B

900 MHz quad-core
ARM Cortex-A7
1 GB RAM
SDHC slot for Flash
Broadcom VideoCore IV
Released Feb 2015

Software Toolchain

- ARM GCC
- picocom
- cppcheck (MISRA C code compliance)
- gcov/lcov (code/testing coverage metrics)
- SciTools' Understand 6.0 (code complexity metrics)
- industry-standard RTOS

Programming Assignments

1. **Bare-metal Programming:** Peripherals and Memory Mapped IO
   - UART, ARM timer, printing, code profiling and assembly optimization
   - **Release date:** Jan 15, **Submission deadline:** Jan 30

2. **More bare-metal**
   - Interrupt handling in ARM + Timer interrupts
   - Applications
   - Embedded SW Engineering Activities (requirements, static analysis, verification, etc)
   - **Release date:** Feb 1, **Submission deadline:** Feb 25

3. **Real-time OS**
   - Port a real-time operating system to RPi
   - Implement periodic-task real-time scheduling algorithms and online admission control
   - Support real-time resource access control
   - **Release date:** Feb 26, **Submission deadline:** Mar 20

4. **Aperiodic tasks + Runtime Monitoring + Formal Verification**
   - **Release date:** Mar 21, **Submission deadline:** Apr 12

5. **Bonus project?**

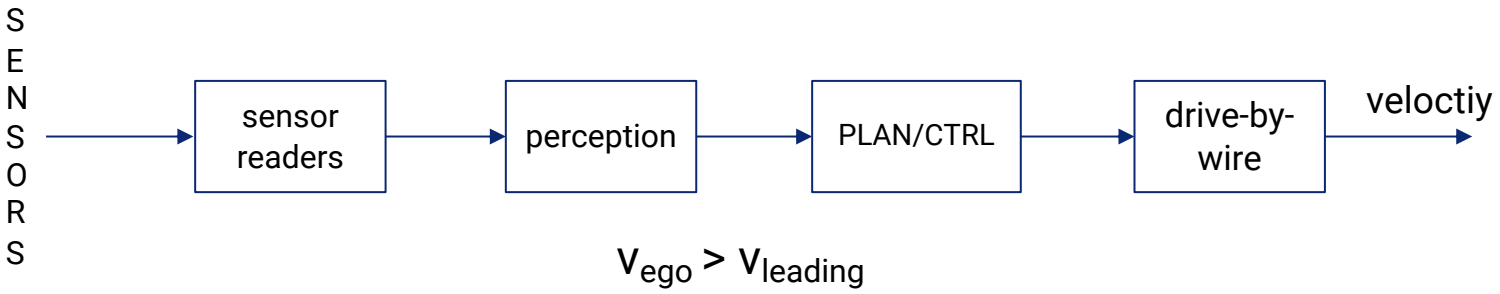
Programming Assignments

- Groups of 3 or 4
- Groups must be formed by **Thursday Jan. 14, 11:59 p.m.**
- Provide the Gitlab usernames of **all group members** in a private Piazza post to instructors

First programming assignment will be out Friday Jan. 15

Programming Assignments – Safety Critical SW

- Assume code will run in a **safety critical system**
  - **Example:** Use a fully autonomous vehicle as the intended "assumed" context
- **Ask:** if my SW fails or performs inadequately in manner that is not mitigated, would such failures contribute to the occurrence of hazards that could lead to harm?
  - Harm: accidents that can take people lives



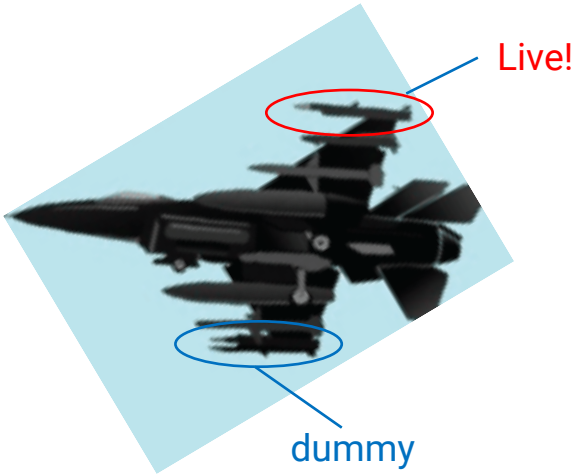
t_1: distance safe
keep ego speed as is

t_2: distance unsafe
Perception does not update view of world

Programming Assignments – Safety Critical SW

Note: Dangerous situations might occur even if SW behaves reliably and as intended/specified

Missile carrier



- there was an antenna between the dummy missile and the target
- "smart" software was designed to substitute a different missile if the one that was commanded to be fired was not in a good position
- software decided to fire a live missile located in a different (better) position instead

Reliability != Safety

Nancy Leveson. Are You Sure Your Software Will Not Kill Anyone?

Embedded Software Risks – Top 10 Warning Signs

- No written requirements or requirements with poor measurability/verifiability
- Process not well-defined or process steps skipped during schedule crunches
- SW development is simply “coding” + “testing”
- Poor traceability from product test to requirements
- Bugs due to poor coding style and complexity
- Bugs in SW fault detection/recovery [recall the “lean and mean” checker]
- No Security plan; no Safety Plan
- Tester:Developer ratio far from about 1:1
- More than about 5–10% of bugs are found in product test
- Fewer than 50% of defects are found by peer review

Avoiding the Top 43 Embedded Software Risks, Philip Koopman

Programming Assignments – Safety Critical SW

- Safety-critical code needs to be developed to the highest quality
- To achieve this, will perform activities that include:
  - Develop unambiguous, verifiable, and measurable **requirements**
  - **Tracing** of requirements to design, source code, and testing
  - Effective **code reviews** → *perspective-based* code review
  - Reduced code **complexity** → McCabe Cyclomatic Complexity (MCC) <= 10
    - Will use SciTools' **`Understand 6.0`** tool
    - Can obtain a student license through https://licensing.scitools.com/student

Programming Assignments – Safety Critical SW (continued)

- Code SHALL comply with **MISRA C:2012** rules
  - **Example:** [Rule 2.2] <Required> *There shall be no dead code*
  - **Example: [**Rule 18.5] <Advisory> *Declarations should contain no more than two levels of pointer nesting* → e.g., `int ** arr[10];` 😨
  - Will use a combination of static analysis tools and manual code reviews to check compliance
    - MISRA C addon to the **`cppcheck`** tool
    - Some bugs cannot be automated through static analysis tools (undecidable)
    - **Q:** Do we care more about tool false positives or false negatives?

Programming Assignments – Safety Critical SW (continued)

- **Tests** should achieve high requirement and structural **coverage**
  - Will develop requirements-based tests, fault injection tests, etc
  - Use code coverage tools to measure how much of the code structure your tests cover
    - Tools: **`lcov/gcov`**
  - **Required:** 100% requirement coverage
  - **Required:** 100% line and branch coverage

Written Assignments

- About 4-5?
- Cover material discussed in class
- Individual work!
  - *You may discuss with colleagues, but **final write-up should be your own***
- Must be **typeset** → handwritten submissions will not be graded
- Submission on **Gradescope**
  - You need to subscribe to the course's Gradescope
    - https://www.gradescope.com/courses/220789
    - Entry code: **BPB6YN**

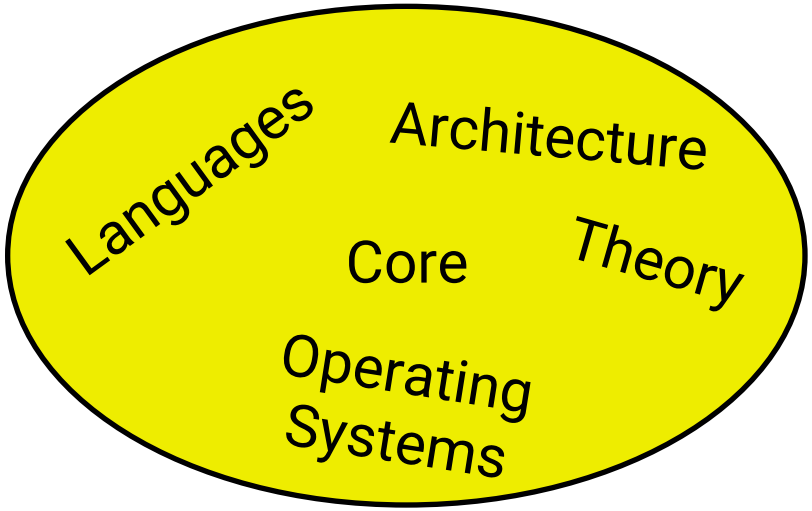
Tutorials

- Every **Wednesday** 9 a.m. – 11:00 a.m.
- Discuss material related to embedded systems, projects
- **Project checkpointing**, discuss progress, and project demos
- Answer questions related to written assignments, class material, projects, etc

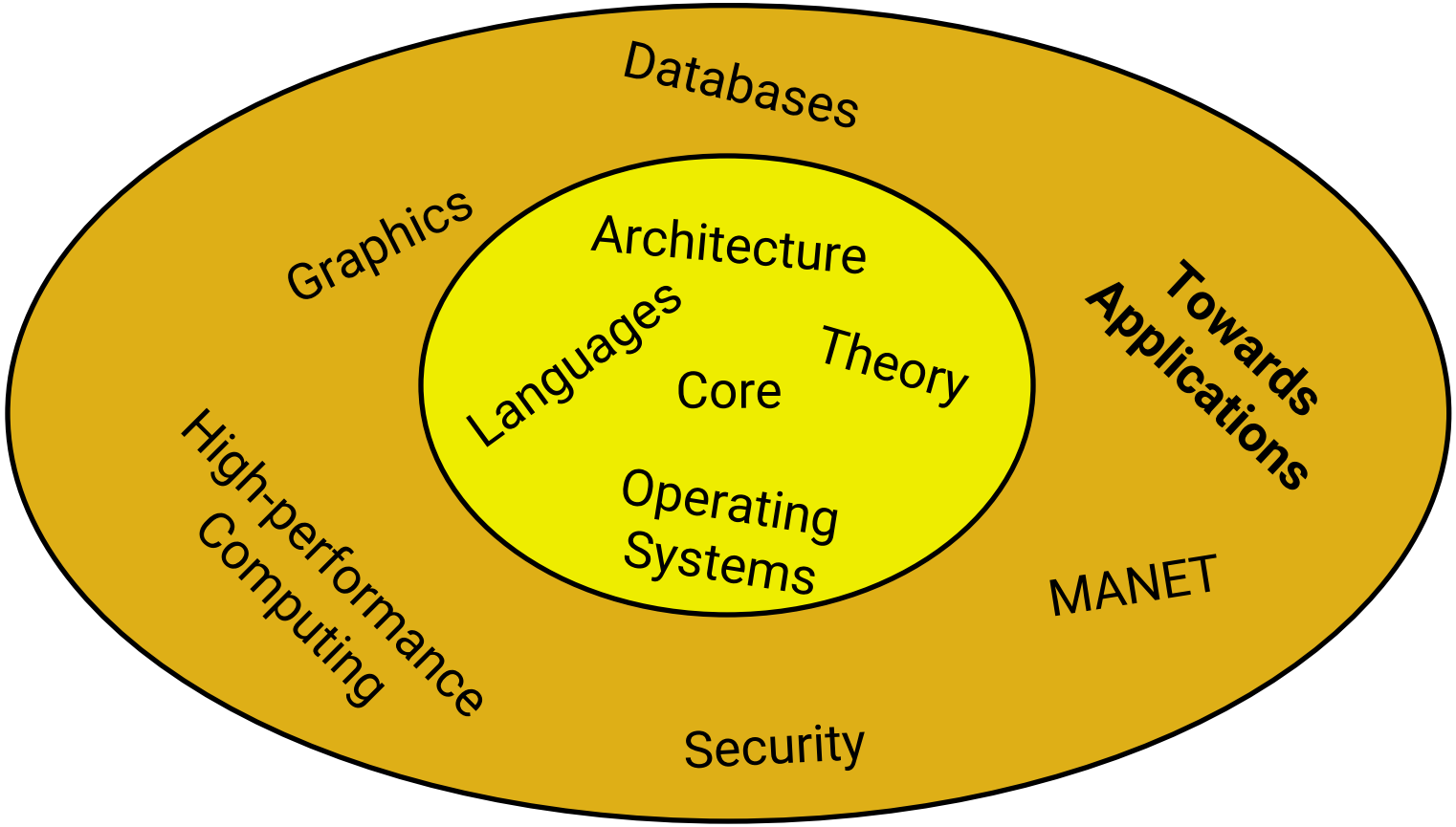
First tutorial: Wednesday Jan. 20

Where are computer systems going?

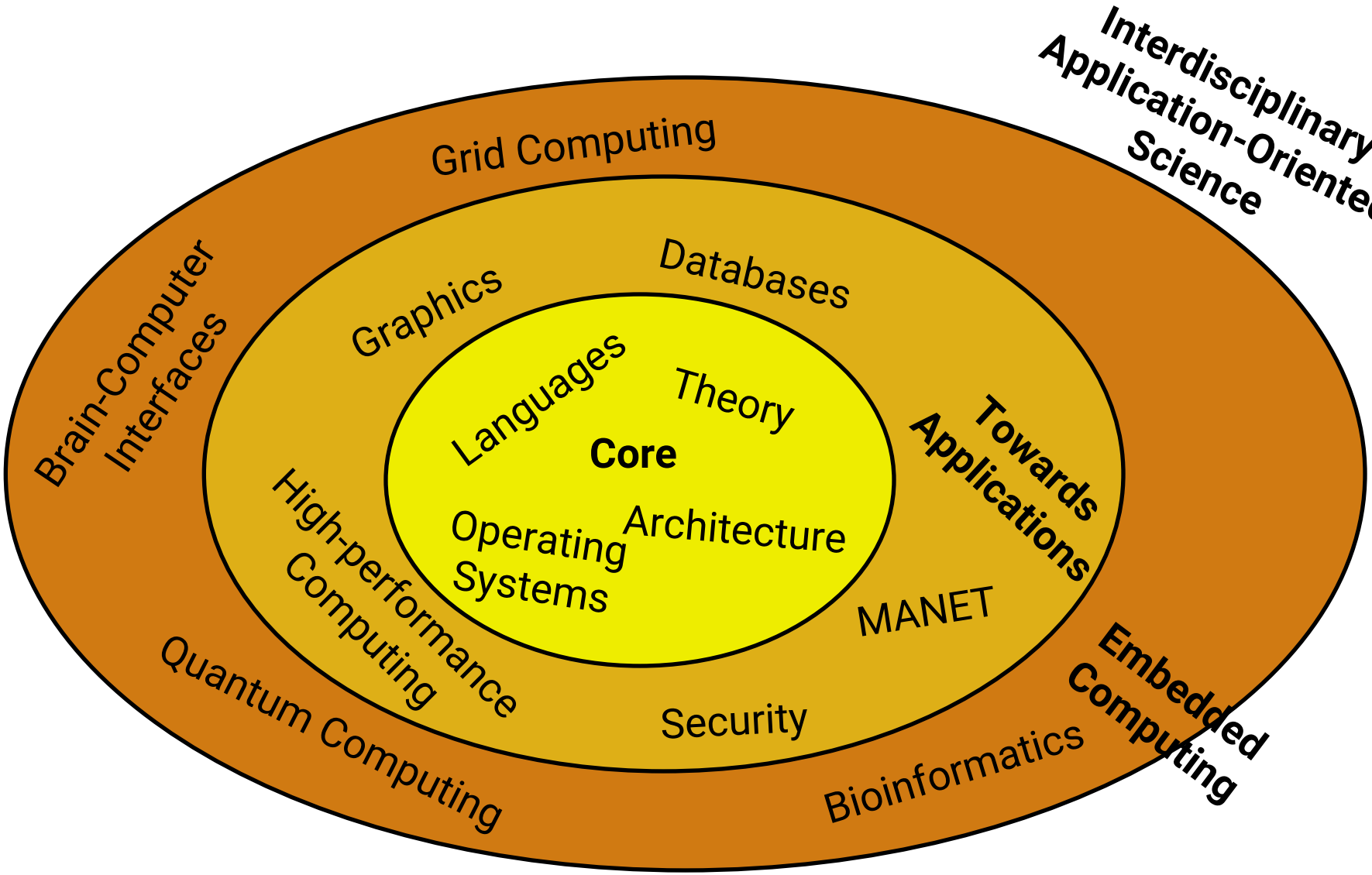
The beginning



Where are computer systems going?



Where are computer systems going?



What are embedded systems?

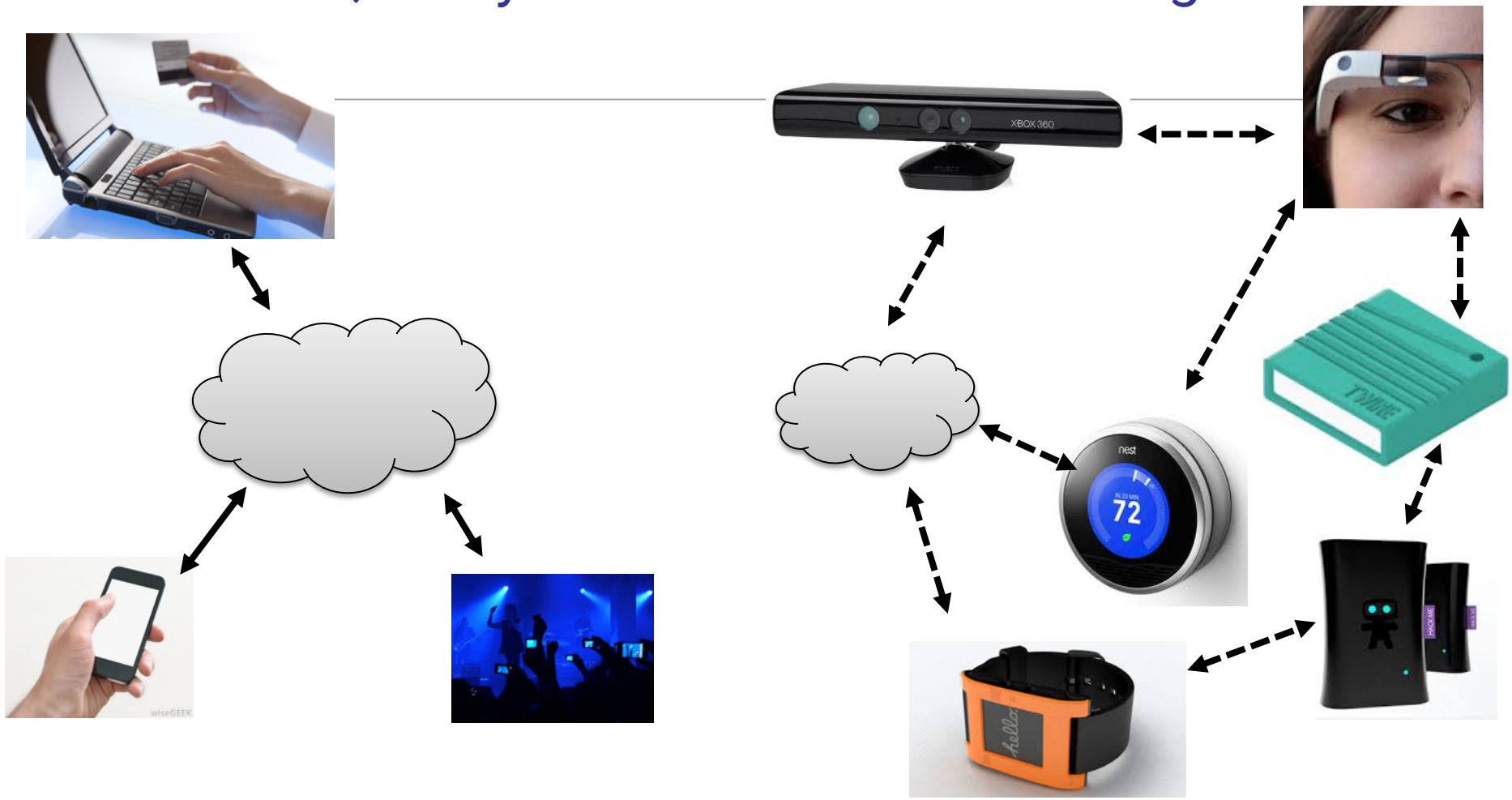
- Computer system hidden (embedded) in other systems
  - Often interacts with the physical environment
- Purpose built
- End users see “smart” device rather then computer
- Special-purpose vs. general-purpose computing

Courtesy of Anthony Rowe

Embedded, Everywhere

Courtesy of Anthony Rowe

Embedded, Everywhere – Internet of Things



People Connecting
Through Machines

Machines Connecting to Machines
and the Physical Environment

Courtesy of Anthony Rowe

Embedded in Your Daily Life

- How many micro-controllers are around you?
- Bathroom scale with digital read out
- Iron that turns itself off automatically
- Electronic toothbrush (with ~3000 lines of code)
- Cooking range
- Laundry machine and dryer
- Toaster
- Microwave
- Home-security
- TV, cable-box, AV system
- Game console
- Thermostat
- Cars, Toys, Medical Devices…

Courtesy of Anthony Rowe

Technology Trends

- Multi-core embedded with SoC
- Better, cheaper, lower power sensors
- Better communication
  - Bluetooth Low-Energy (BTLE)
  - 802.15.4 (focus: low-cost, low-speed ubiquitous communication between devices)
  - 802.11 AC
- Energy Harvesting

Courtesy of Anthony Rowe

Why is embedded different?

Typical Embedded System Challenges (1-2)

- **Small Size, Low Weight**
  - Handheld 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

Courtesy of Anthony Rowe

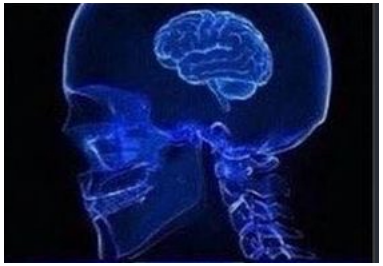
Typical Embedded System Challenges (2-2)

- **Harsh environment**
  - Heat, vibration, shock
  - Power fluctuations, RF interference, lightning
  - Water, corrosion, physical abuse
- **Safety-critical operation**
  - Must function correctly
  - Must not function incorrectly
- **Extreme cost sensitivity**
  - $0.05 adds up over 1,000,000 units

Courtesy of Anthony Rowe

CPU: An All Too Common View of Computing

- Measured by: Performance

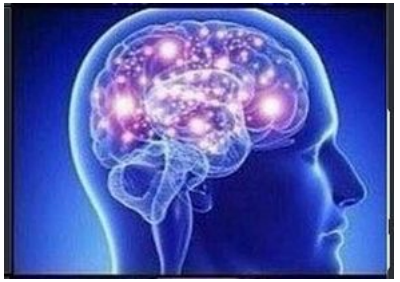


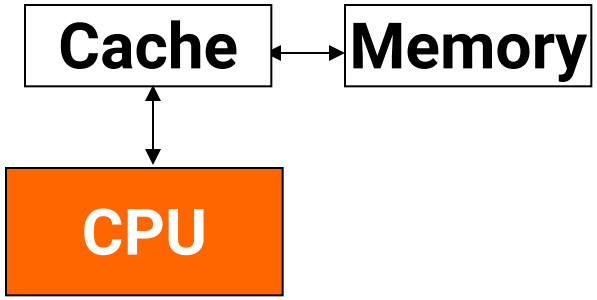
CPU

Courtesy of Anthony Rowe

An **Advanced Computer Engineer's** View

- Measured by: Performance
  - Compilers matter too...

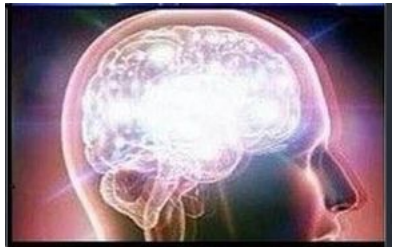


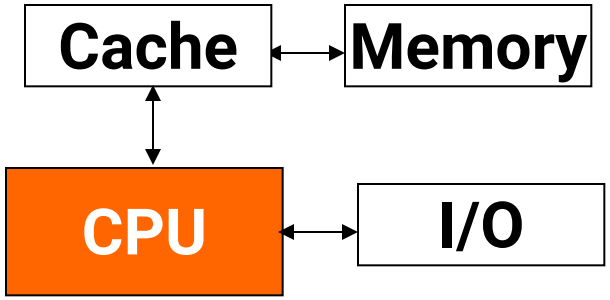


Courtesy of Anthony Rowe

An **Enlightened Computer Engineer's** View

- Measured by: Performance, Cost
  - Compilers & OS matter

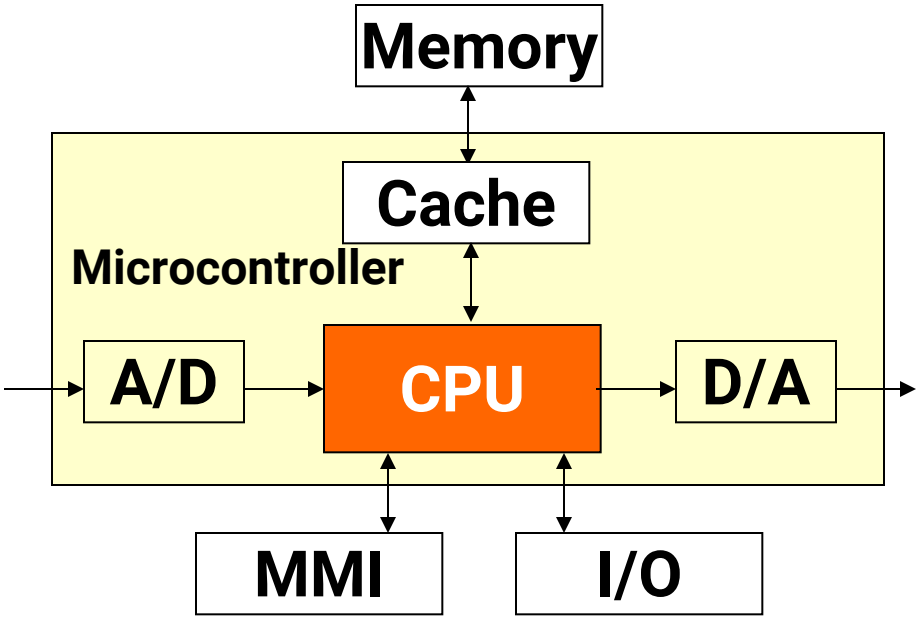




Courtesy of Anthony Rowe

An **Embedded Computer Designer's** View

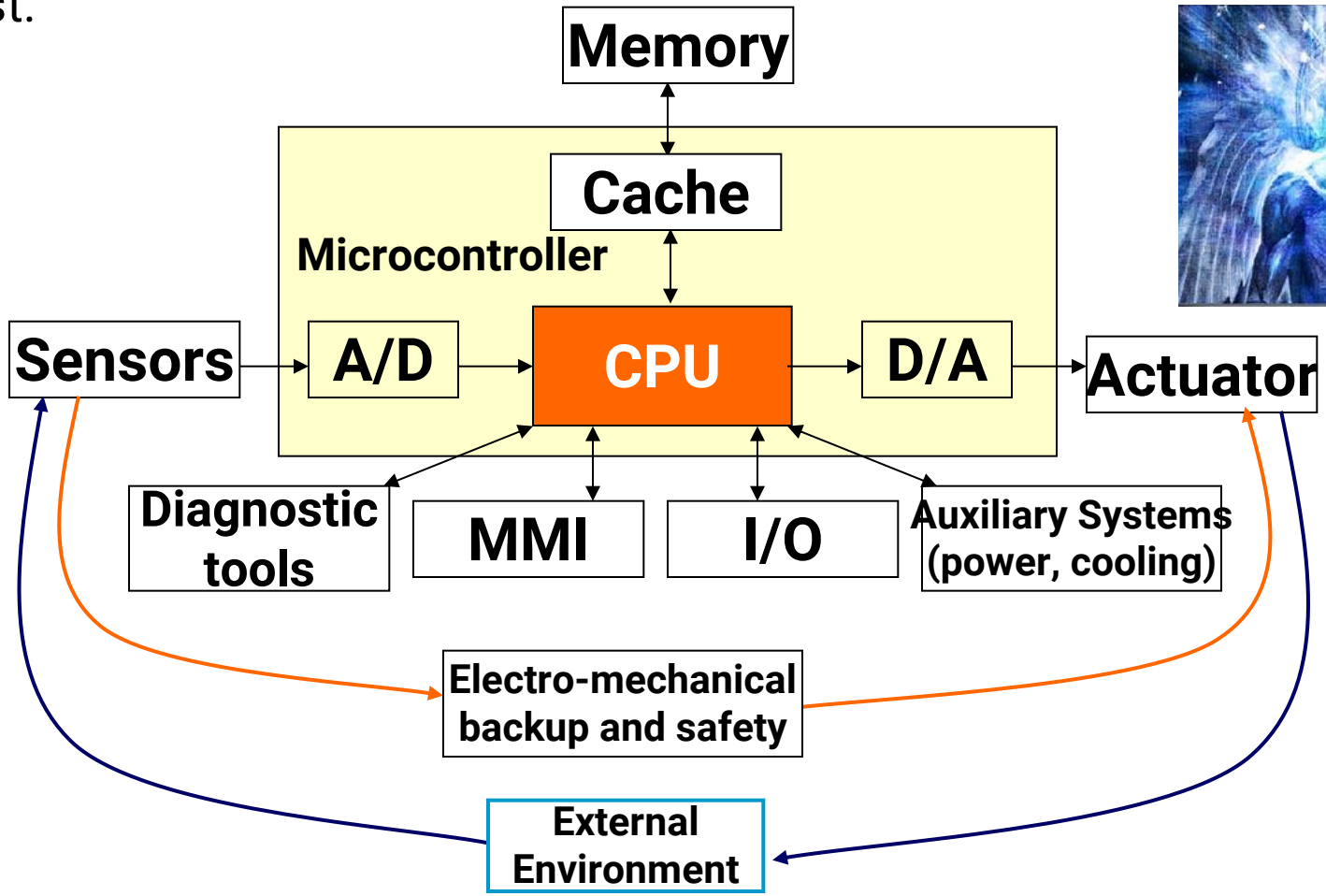
- Measured by: Cost, I/O connections, Memory Size, Performance



Courtesy of Anthony Rowe

An Embedded **Application Designer's** View

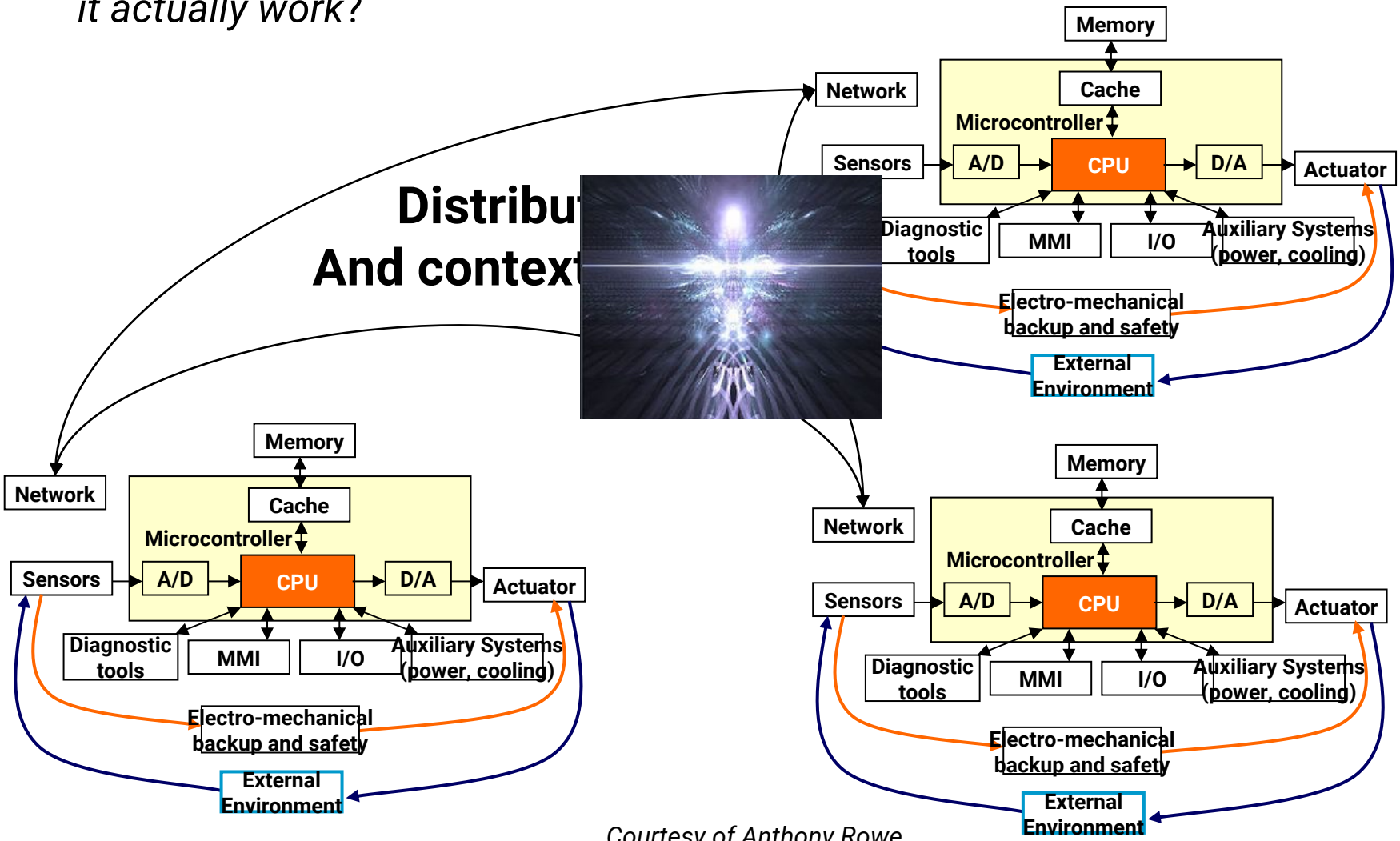
- Measured by: Cost, Time-to-market, Cost, Functionality, Cost & Cost.



Courtesy of Anthony Rowe

Modern Embedded Systems View

- Measured by: cost, performance, time-to-market, cost, cost, & *does it actually work*?



Courtesy of Anthony Rowe

Embedded Computers *Rule* the Marketplace

- **In 2003: Embedded processors were 98% of all processors manufactured**
  - ~80 Million PCs vs. ~3 **Billion** Embedded CPUs annually
  - Embedded market growing; PC market *mostly* saturated
- **Domain Experts Needed…**
  - **General Computing**
    - Set-top boxes, video game consoles, ATM, …
  - **Control Systems**
    - Airplane, Heating and Cooling System
  - **Signal Processing**
    - Radar, Sonar, Video Compression, Human-Brain interface
  - **Communication**
    - Internet, Wireless Communication, VoIP…

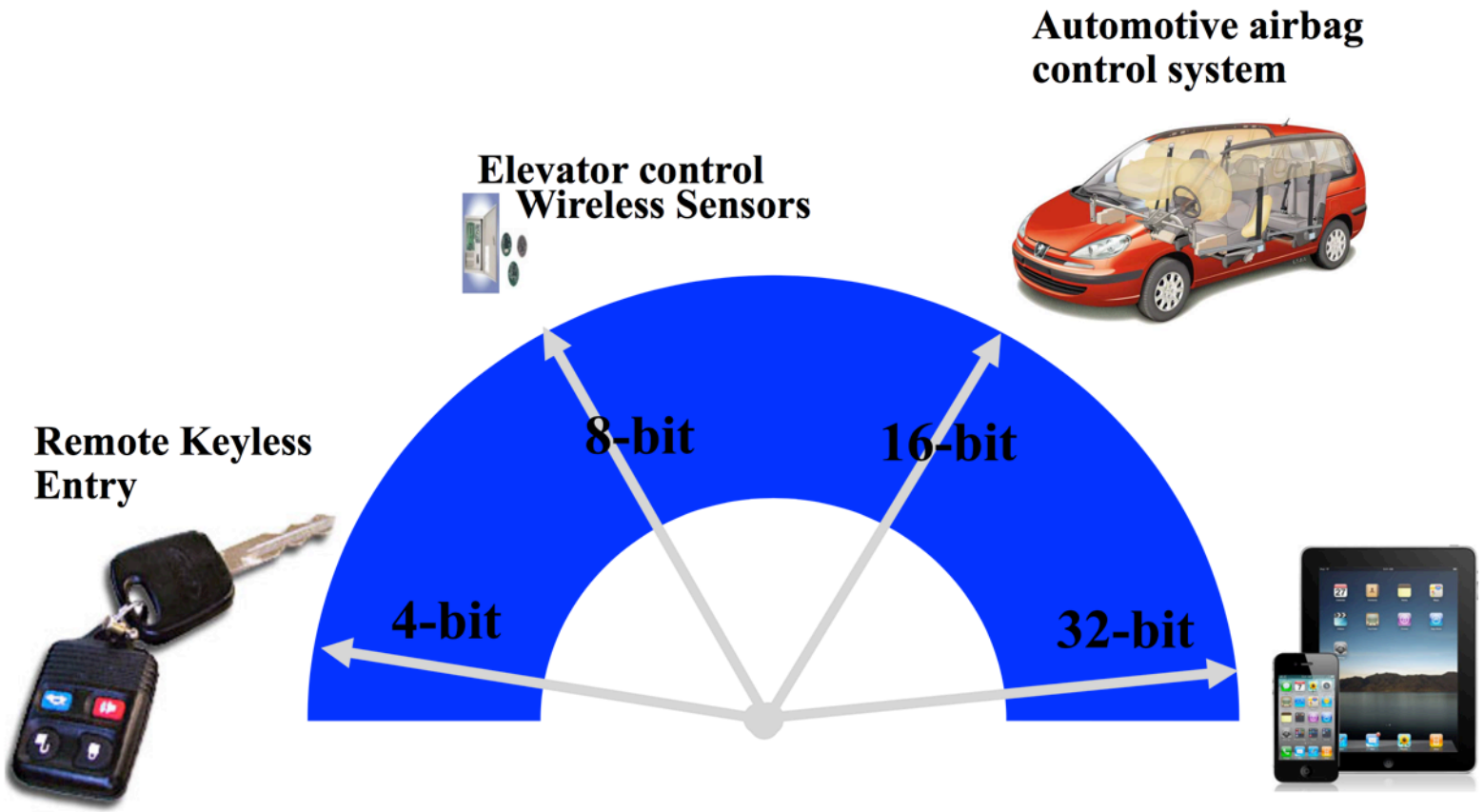
Courtesy of Anthony Rowe

Embedded Systems Careers

Courtesy of Anthony Rowe

Misconceptions about embedded systems (1)

Embedded systems = low end microcontrollers



Courtesy of Anthony Rowe

Misconceptions (2)

Embedded systems = old topic

- Always new and exciting developments that track technology
  - New sensors / actuators
  - More powerful chips
  - New communication mechanisms
- Embedded systems + Internet = Internet of Things
  - Massively hot topic right now!

Courtesy of Anthony Rowe

Are these misconceptions?

- Embedded system programing = programming in assembly to optimize the code for space, time, etc.
- Compilers are typically better then humans at generating the best code
- Code portability issues → some device-driver dependent code written in assembly, but most app code is written in higher-level languages

Courtesy of Anthony Rowe

So, what does “real-time” mean?

Fast?

Predictable?

Reliable?

All of the above?

Other?

Why predictability?

Example: Going to the airport
Which route would you choose?

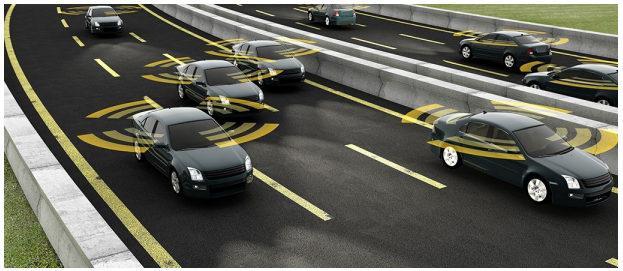
Route 1: **15** min ($1 Toll)
Route 2: 5-45 min, with **15 min average** (free)

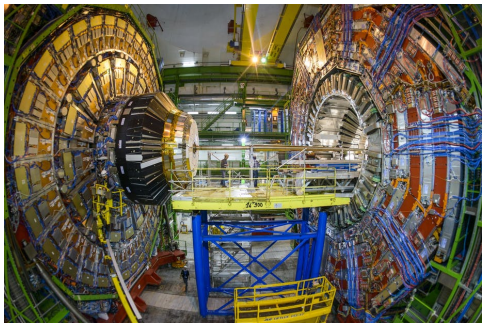
You pay for predictability

What are real-time systems?

Study of systems where the correctness of computation depends on the timing of the results

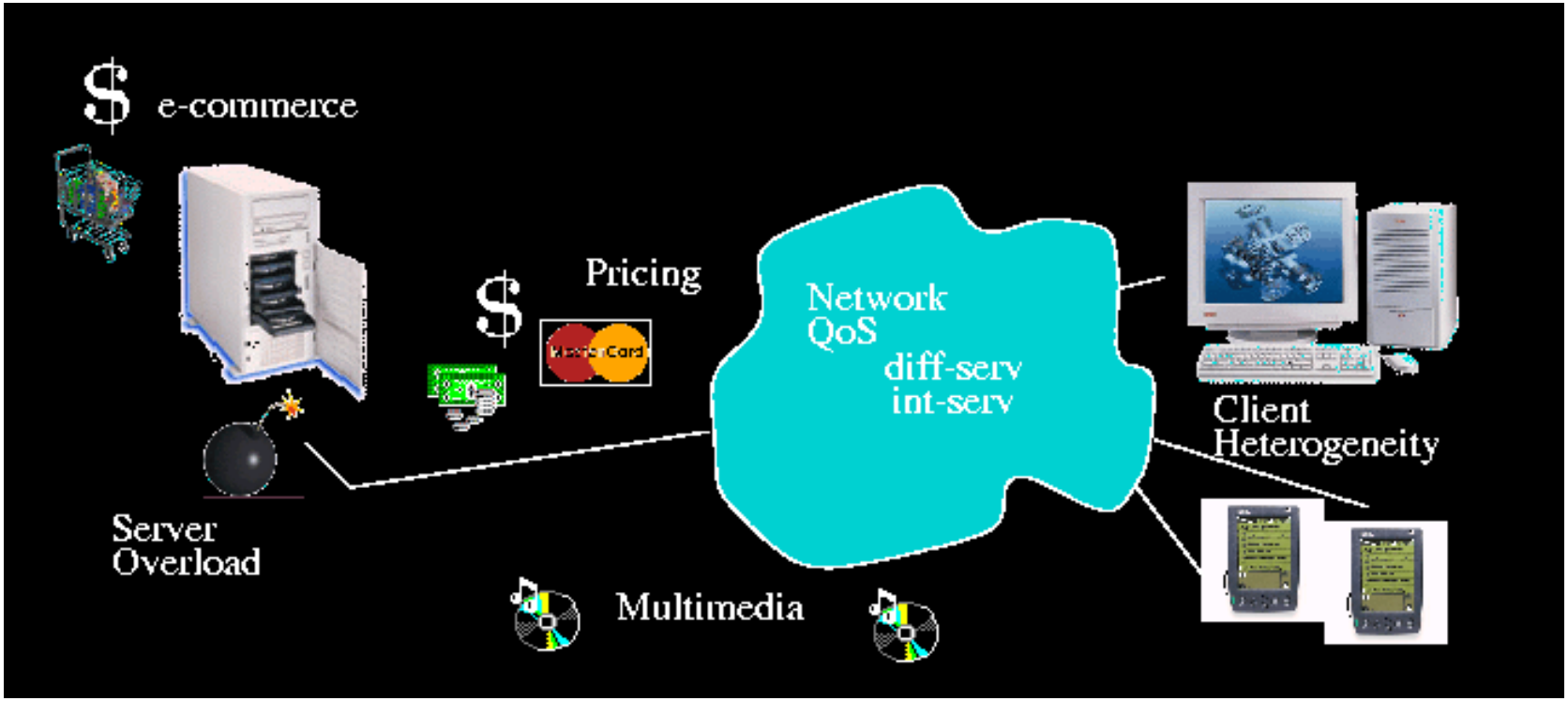
What are real-time systems?





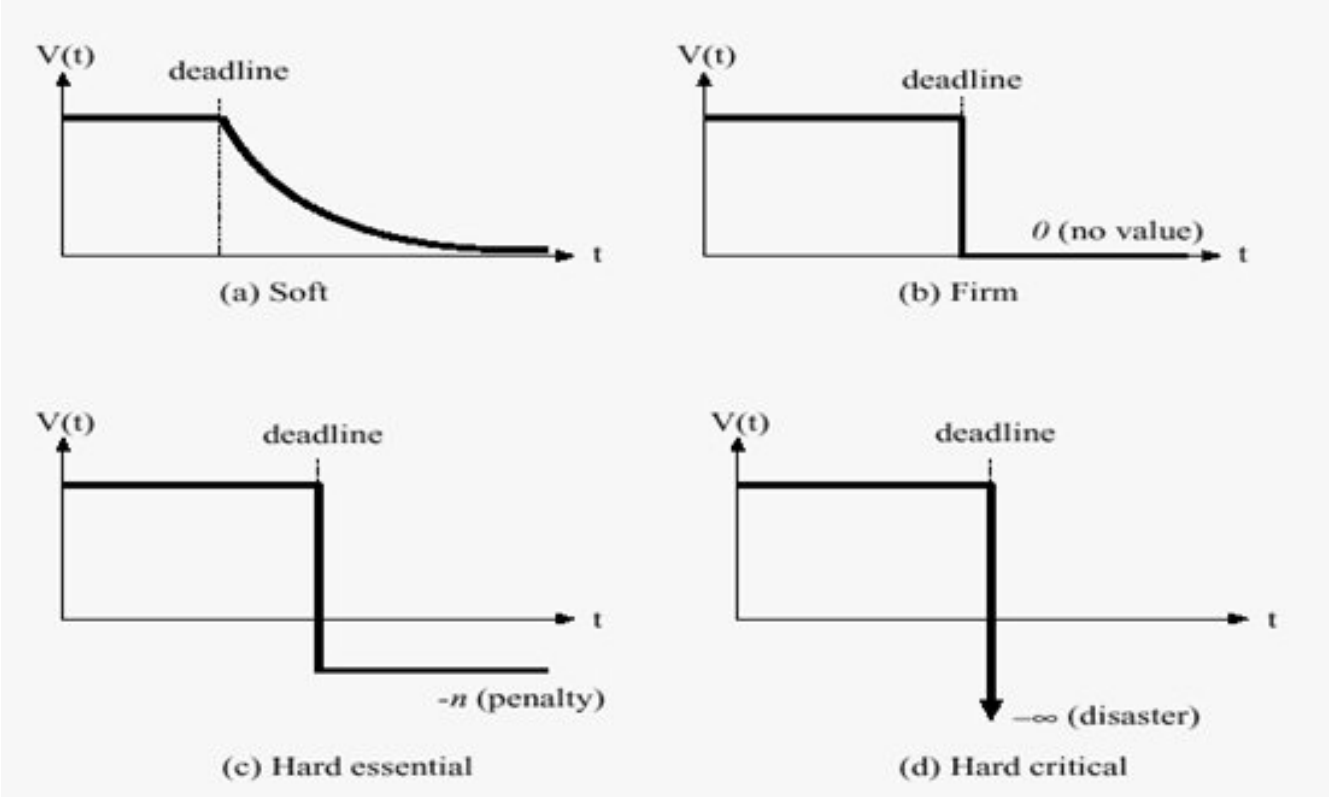
Classic applications (Hard real-time)

More real-time systems (soft)



Applications of the 90s

Various levels of “impact” of missing deadlines



V(t): Value of execution (utility)

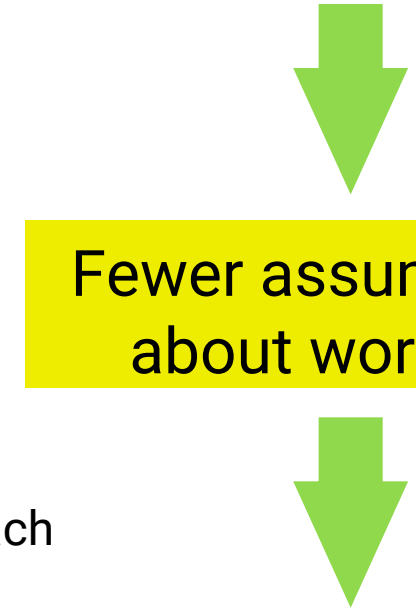
Real-time + Embedded

- Many Hard real-time systems are embedded devices
  - Specific hardware
  - Customized and limited software
  - Simplified OS/ **RTOS** .. or No OS
  - Guarantees are provided through simplicity, precise definition, good design and ***overprovisioning***

- Real-time systems more concerned with ***predictability*** rather than absolute response times
  - Providing faster processors will convert a PC (Soft RTS) to a faster PC (Soft RTS), not a Hard RTS.

The real-time computing roadmap

- **Scheduling!**
- Milestone 1 (1960s): Cyclic executive scheduling
  - Fixed task set
  - Known arrivals
  - Known resources
- Milestone 2 (1970s): Rate monotonic analysis
  - Tasks could be added or removed online
- Milestone 3 (1980s): The Spring scheduling approach
- Milestone 4 (1990s): Quality of service
- Milestone 5 (2000s): Feedback-controlled scheduling



The changing scope of real-time computing

- Classical view
- **Definition:** correctness of computation depends on the time at which results are generated
- Applications: embedded control systems
  - Perfect knowledge of resources
  - No conflicts of interest
- Assumptions: **hard real-time**
  - Predictability instituted by associating every task with a ***deadline***
  - Meeting deadlines is very important for correctness
  - Deadlines are inflexible

What are real-time systems?

Systems where a substantial fraction of the design effort goes into making sure that *deadlines are met*

New assumptions

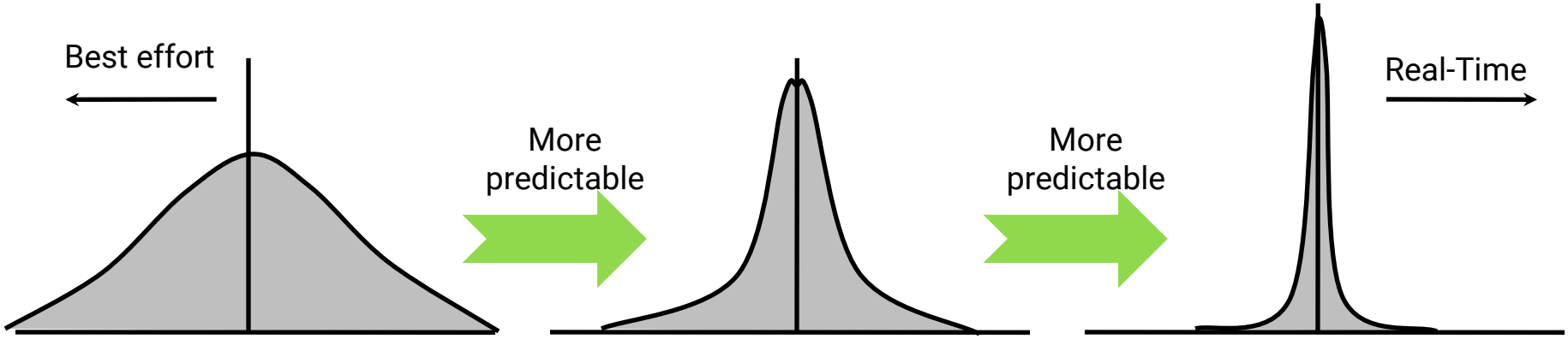
- Unknown resource requirements
- Elastic time constraints
  - Flexible periods (if any)
  - Flexible deadlines
- Adaptation capability
  - Changing algorithm version
  - Adaptive data quality
- **Steady-state** versus **transient metrics**

Response time is an important parameter

The time between the presentation of a set of inputs to a system (stimulus) and the realization of the required behavior (response), including the availability of all associated outputs

New real-time definitions?

- Systems with low variance in performance measurements



- Systems with guaranteed metrics in which time is either in the numerator or in the denominator
  - Response time = Service time + Queuing time
  - **Utilization = Used resource units/time**
  - Service throughput = Produced data units/time

What we will cover in this course

- Basic concepts (tasks, threads, blocking, priorities, importance, resource partitioning, QoS, etc.)
- Estimating (worst-case) execution times
- Periodic versus aperiodic task models and their implications
- Optimality results in real-time scheduling
- Fixed-priority and dynamic-priority aperiodic-task servers
- Blocking, synchronization, and resource access protocols
- Overload, quality of service, and system design for graceful degradation
- Hardware/software co-design for real-time embedded systems
- Reliability and fault tolerance