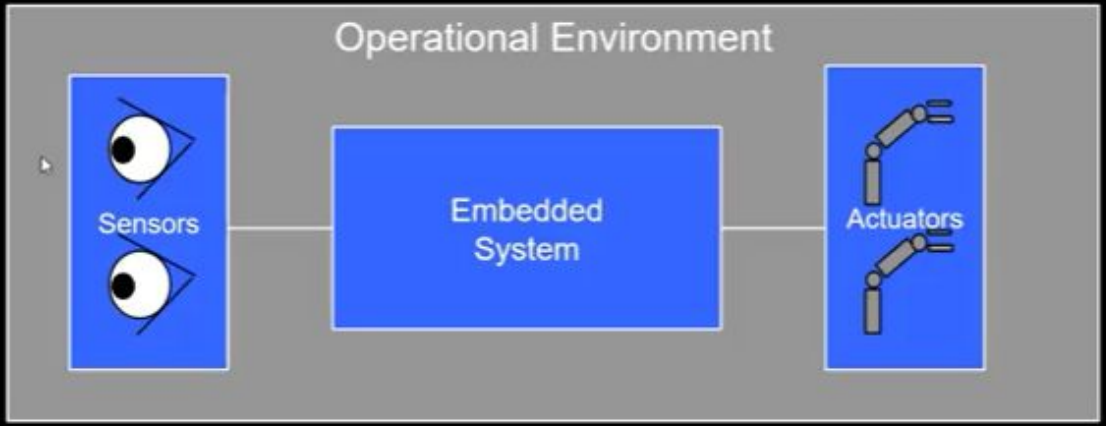
Real Time Embedded System (RTES)

Embedded System

- **To be Embedded is …**
- **A Compute Node That Provides Specific Services by Processing Inputs and Producing Required Responses**
  - **Provides Specific Services Rather than General Purpose Computing**
  - **Often Limited or No Direct Connection to User Input/Output (e.g. Antilock brakes)**
  - **Contained within a Larger System as a Sub-system (flight control on aircraft)**



- To be Real-Time is …

- Services that must be provide upon request prior to a deadline
  - A deadline is a constraint for processing
  - If no deadline constraint exists for a system, it is not real-time
  - Most real-time systems handle requests on a periodic basis
  - Some real-time systems require deterministic response before a deadline
  - Some required predictable response (some missed deadlines are allowed)

- Impact of a missed deadline as specified by design constraint
  - Loss of property, life, financial, or system itself - HRT (Hard Real-Time)
  - Loss of service, quality of that service, and annoyance - SRT (Soft Real-Time)

- No deadline constraint - BE (Best Effort)
  - Desktop systems are interactive, low latency response, but no response constraint
  - May be responsive, but in a best effort fashion (some waiting… hourglass)

- **Many Examples of Real-Time Embedded Systems**
  - Real Time – Must Respond to Requests for Service by a Deadline relative to request
  - Failure to Respond Prior to Deadline Results in a System Failure
  - Request Rate for Service Driven by Real-World Events
  - Controls Processes and Delivers Deadline Driven Services
  - Specific Examples (with deadline constraints)
    - Medical Devices (e.g. Pulse Oximeter)
    - Unmanned Aerial Systems
    - Anti-Lock Braking
    - Streaming Media (Video and Audio)
    - Process Control
    - Aircraft Flight Control
    - Robotic Systems

Why are RT Embedded Systems a Challenge?

1. Correct results on time – Deadlines (constraint)
2. Multi-service Concurrency - Multi-threaded Software
3. Multiple interfaces to service concurrently
4. Function and Service Allocation – CPU Software, HW Co-processor (e.g. FGPA, MPEG decoder), or Software Co-processing (e.g. DSP, GP-GPU)
5. Management of CPU, IO, and Memory Resources
6. Modern architecture – high throughput, less deterministic
7. Sensors / Actuators (control Physical Process)
8. Networks (Latency and QoS)
9. Persistent Memory Devices (e.g. Flash, EEPROM)
10. Memory Hierarchy (Register, Cache, RAM, Flash)

- Simplifying Real-Time Systems and Ensuring Success

1. RT Service and CPU Resource Management Policies (e.g. Rate Monotonic, EDF)
2. RT Theory and Best Practices (Theory, Analysis, Models, System, Verification)
3. RMA Resource Theory (proof of feasibility of design)
4. Performance Prediction and Measurement (predicted to actual comparison)
5. When to Allocate Services/Functions to HW, FW, or SW
6. Multi-threaded RTOS and Real-Time Linux Systems
7. RTOS or POSIX RT Mechanisms (e.g. message queue, signal, semaphore)
8. Analysis and Debug Tools
9. I/O Device Interfaces and Driver
10. Abstracted Non-Volatile Memory File systems

- Rate Monotonic Concepts and Principles
  - Best Effort - Non-RT processing (Windows, desktop Linux, AOS, iOS, etc.)
  - Soft Real-Time - predictable response (Netflix, SmartTV, MPEG video, digital video games, etc.)
  - Hard Real-Time - Commercial aviation FCS (Flight Control System), Anti-lock braking, Heart-lung machine, Air Traffic Control, etc.
- Scheduling of Services (Concurrency)
  - Math Models - Rate Monotonic Analysis (RMA) of fixed priority, Adaptive Analysis of dynamic priority
  - Feasibility - determined by manual analysis or mathematical feasibility tests (algorithms) - can concurrent services meet deadlines (constraint)
  - Specification of RT constraints - S_i, T_i, C_i, D_i
  - Schedulers and Hardware vs. Software RT Services