Real-Time Software In FRC Robotics

Keith Buchanan (FRC 1296)

Principal Engineer

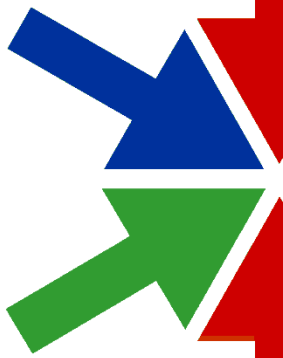
Mike Anderson (FRC 116)

Chief Technical Officer

The PTR Group, Inc.

www.theptrgroup.com

keith@theptrgroup.com



About The PTR Group, Inc.

- Founded in 2000, 35+ Engineers
- Embedded Systems Specialists
  - VxWorks™, Linux, Android and more
  - Hardware and software design
- Sponsors Six FRC teams in 2014

FRC 63	-- Red Barons	McDowell HS, Erie, PA
FRC 116	-- Epsilon Delta	Herndon HS, Herndon VA
FRC 401	-- Hokie Guard	Virginia Tech School of Education & Montgomery Co Public Schools, Christiansburg, VA
FRC 620	-- Warbots	James Madison HS, Vienna, VA
FRC 1296	-- Full Metal Jackets	Rockwall HS, Rockwall, TX
FRC 2537	-- Space RAIDers	Altholton HS, Columbia, MD

Who Am I?

- FIRST Team 1296, Full Metal Jackets, Rockwall HS TX
  - Head Mentor, 10 years
- Electrical Engineer (BSEE- Maryland Terps)
  - 30+ years in Embedded Systems
  - Principal Engineer with The PTR Group
- Big Projects Using VxWorks
  - Control Systems of Navy Prototype Optical Interferometer(NPOI)
  - Navigation System, Aircraft Carriers
  - AEGIS Gun Weapons System Networks
  - Corporate RTOS/VxWorks/Linux Training For The PTR Group, Wind River & MontaVista

Understanding Temporal Requirements

- Most humans understand time with respect to *Wall Clock Time*
  - Something a human can measure
- Unfortunately, humans are terrible at being able to detect the passage of time
  - 1 μs vs 100ms are both instantaneous to a human

Source: stanford.edu

What Is Real-Time?

- Real-time means computing with a deadline
- A late answer is a wrong answer
- Fast enough to maintain control
- Determinism
  - Always works the same way every time
- Software for the real world

Source: redorbit.com

Soft vs. Hard Real Time

- In soft real time, you don't want to miss a deadline, but the deadlines are "squishy"
  - By definition if there's a human in the loop, you've got soft real time
    - The dashboard is a good example
- In hard real time, if you miss the deadline then life, limb or unrecoverable data is lost
  - Examples include air bag deployment, avionics, etc.

Synchronous Vs. Event-driven

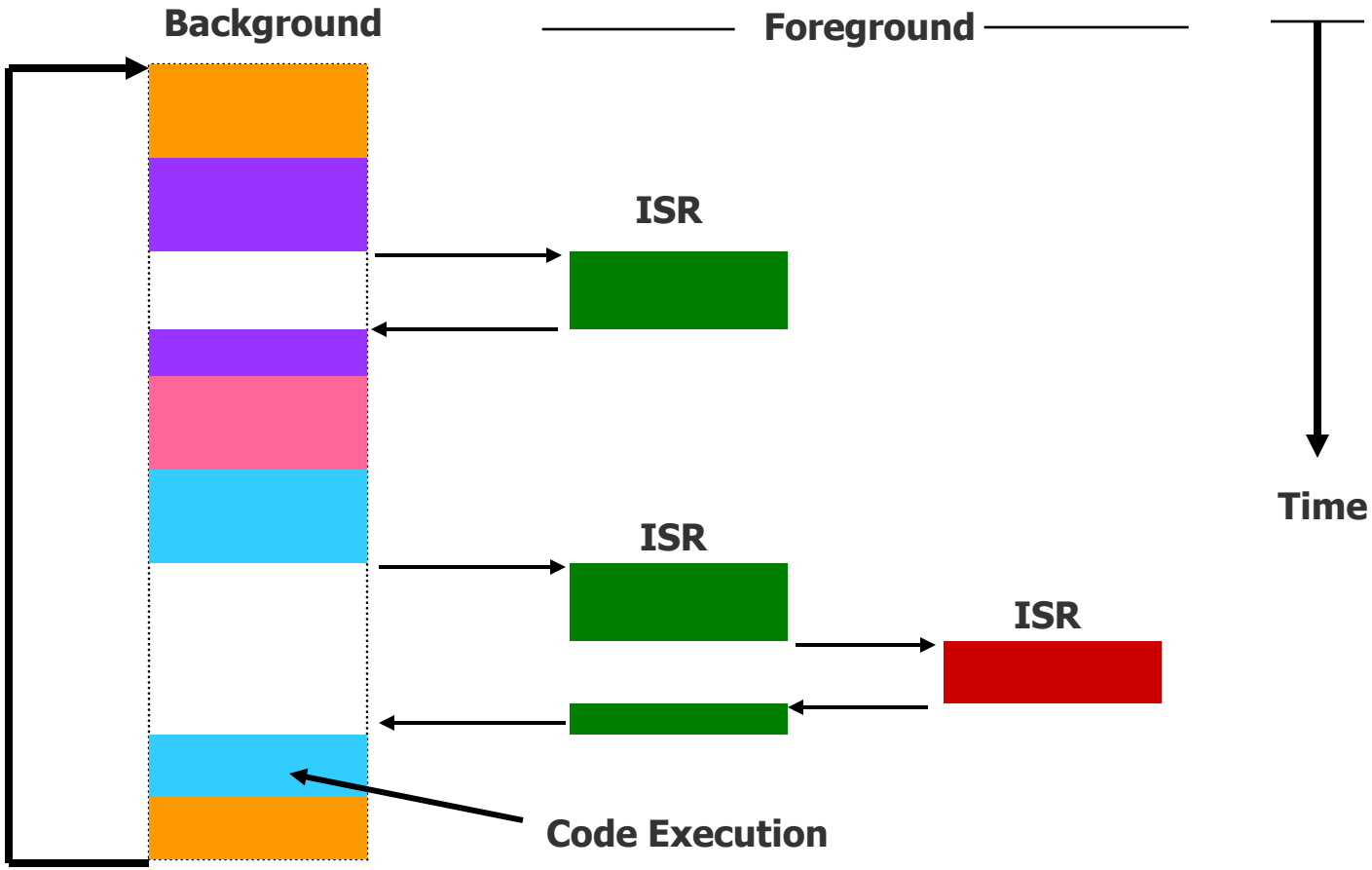
- Synchronous systems perform tasks on a set schedule
  - The outside world may be polled
  - Conflicting schedules can be difficult to resolve
- Asynchronous systems are event-driven
  - They do action based on outside stimuli
- Most systems are a combination of the two

Real-Time Design Approaches

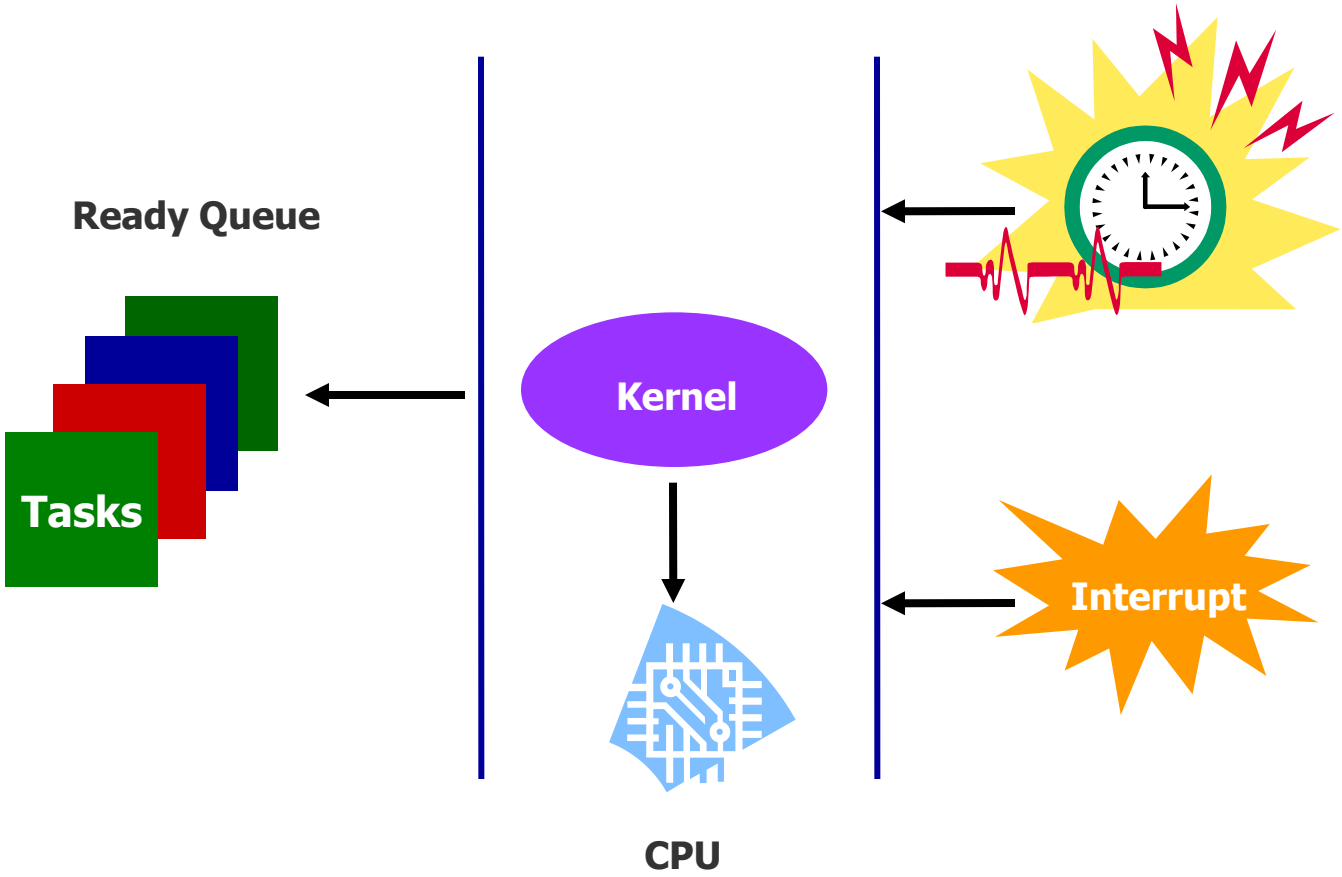
- Two Primary Techniques
  - Super-Loops
    - One program running in a loop + ISRs
  - Multi-Tasking
    - Cooperative multi-tasking
      - Delays, System Calls, Voluntary
    - Preemptive multi-tasking
      - System Timer, Forced Context Switch
    - VxWorks™ & Linux (Next Year)

The Juggler 1991

Super-Loop Execution



Multi-Tasking Example



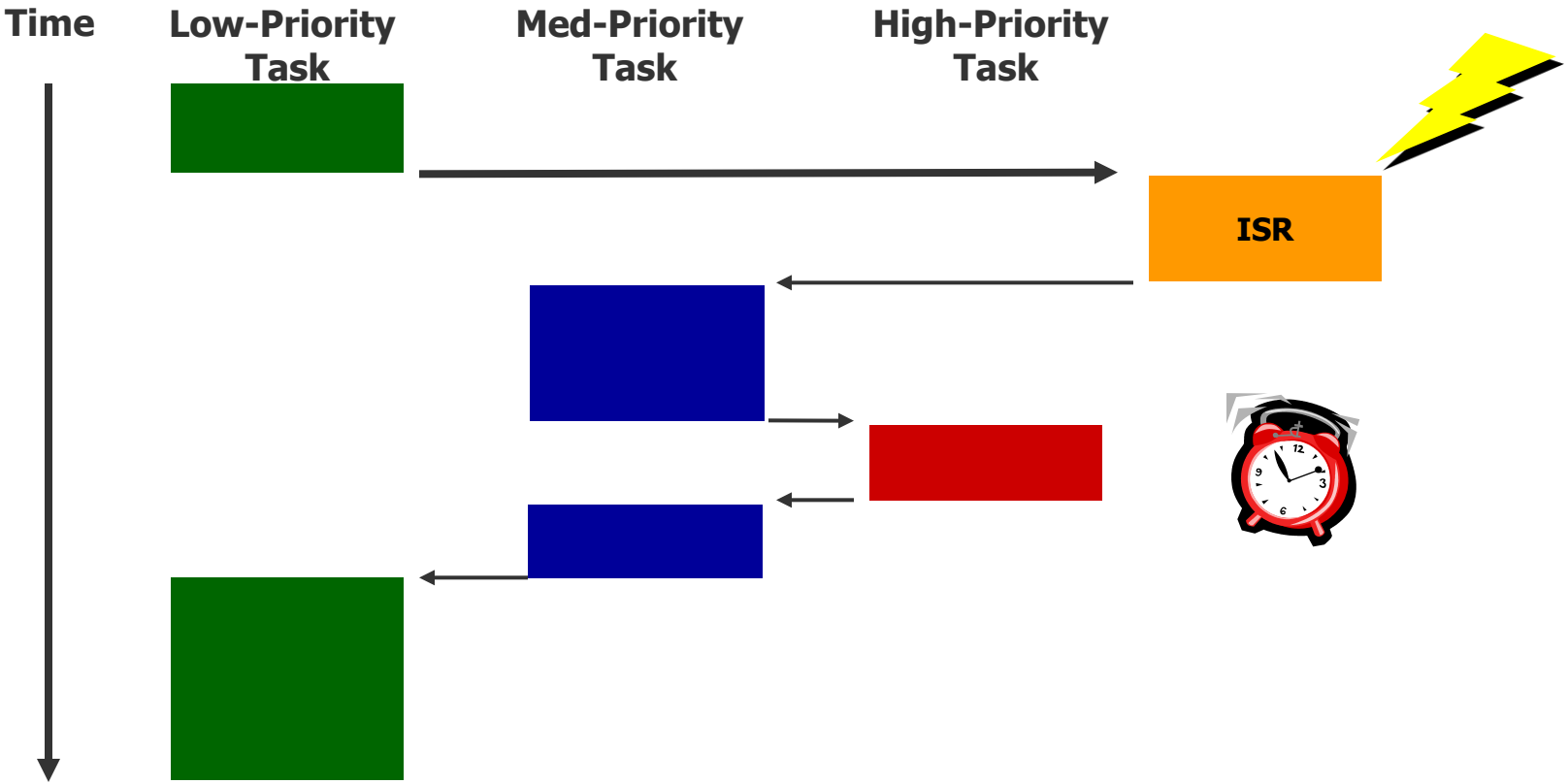
What is a Task?

- A task (also called a thread) is a single thread of control in the system
- While it runs, a task thinks it has the CPU all to itself
- We will typically divide the work to be done into a series of cooperating tasks
- A task consists of its TCB, its stack and the code, may (RTPs in VxWorks, Processes in Linux) or may not (tasks/threads) include a memory context

Preemptive, Priority-Based Scheduling

- As with real life, certain control threads have an inherently higher priority
  - Air bag deployment takes precedence over GPS
- With this approach, the operating system guarantees that the highest priority thread always wins
  - O/S forces a context switch to the highest thread ready to run
- Default in VxWorks™, "Real Time Priorities" (non-zero) in Linux

Preemptive, Priority-based Example



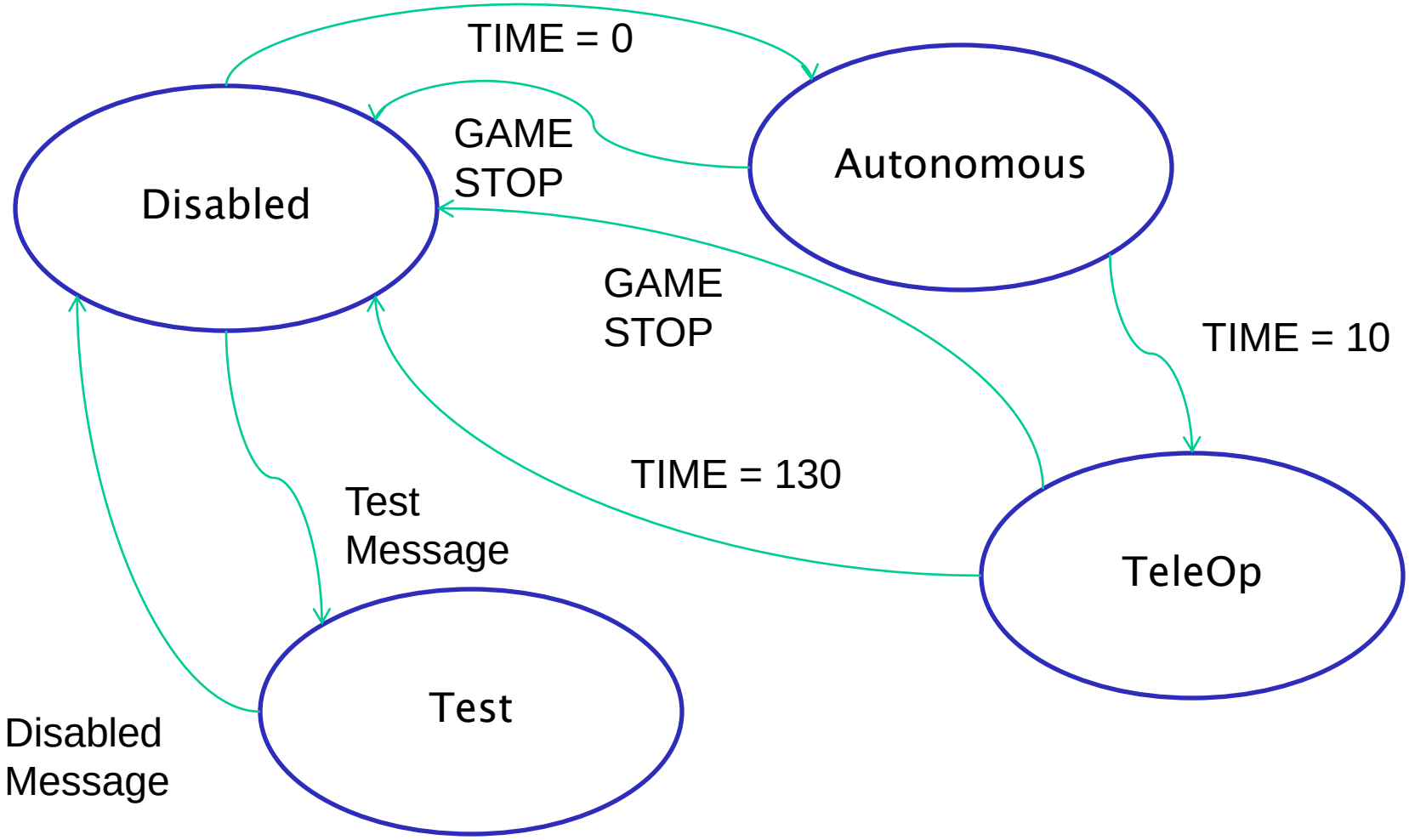
Decomposition of Your Tasks

- Before we can develop a real-time control system, we need to identify the major tasks and assign a notional priority to them
  - Mobility
  - Sensors
  - End effectors
  - User interaction
- One approach is to develop *Finite state machines* that represent the subsystems

Finite State Machines

- In general, a state machine is any device that stores the status of something at a given time and can operate on input to change the status and/or cause an action or output to take place for any given change
  - It's finite because there are only so many states that are possible
- In their book *Real-time Object-oriented Modeling*, Bran Selic & Garth Gullekson view a state machine as:
  - A set of states
  - A description of the initial state
  - A set of input events
  - A set of output events
  - A function that maps states and input to output
  - A function that maps states and inputs to states
    - Called a *state transition* function

Example Robot Mobility States

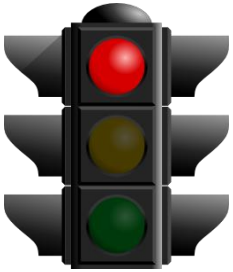


Task/Thread Control Mechanisms

- Using polling loops works in some cases, but has potential efficiency problems
  - How long does it take before you get back to the top of the loop to do something important there?
- By using multi-threading, we enable code to run outside of the main control loop
  - Scheduler keeps everything separate and guarantees highest priority thread always wins
- However, we need to coordinate threads to avoid race conditions
  - Binary semaphores/synchronizers, message queues, mutual exclusion semaphores, sockets, etc.
  - Known collectively as inter-process communications (IPC) mechanisms

What is a Semaphore?

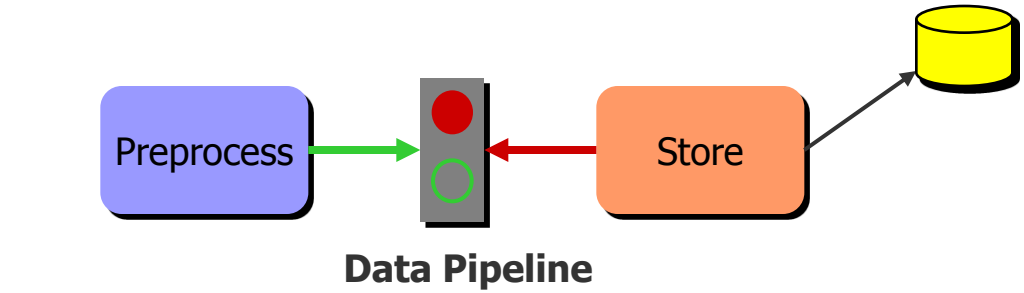
- Conceived by Edsger Wybe Dijkstra in the 1960s, the semaphore can be thought of as a traffic signal in an operating system
  - In Hardware: Read-Modify-Write (RMW), Test And Set (TAS), Atomics
- Binary semaphores do not have a concept of ownership
  - Light turns green and you can go, but you don't own the light
  - Used primarily for synchronization
    - E.g., wait here until this event happens
    - "Blocking" code



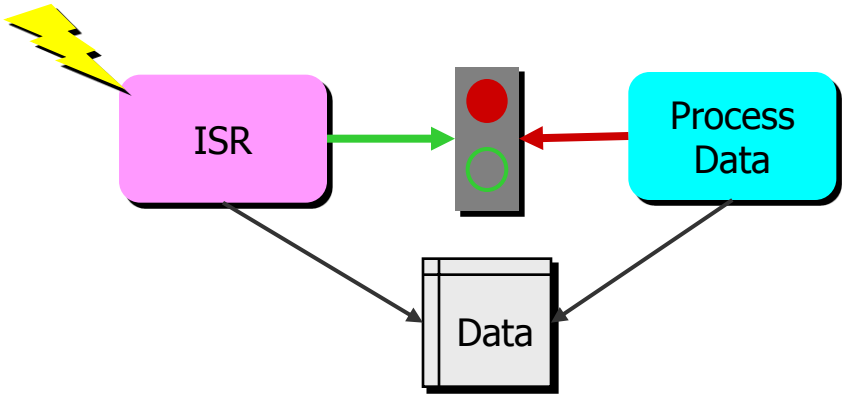
Source: clker.com

Source: dot.gov

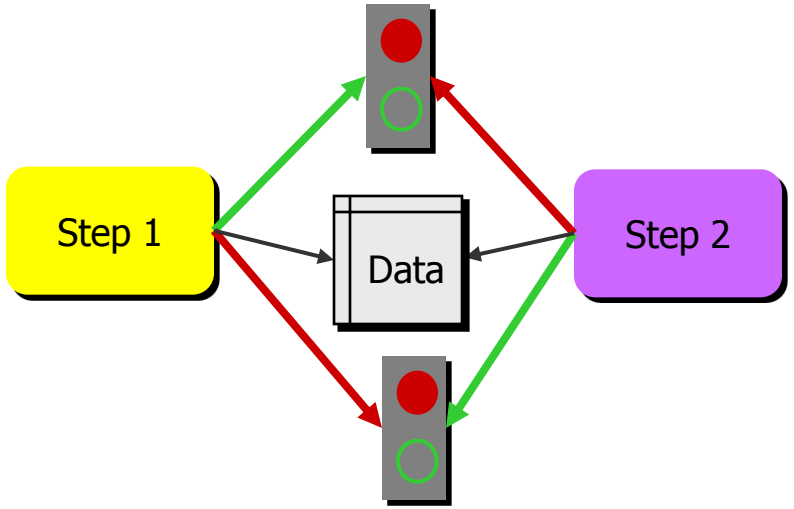
Synchronization Scenarios



Data Pipeline



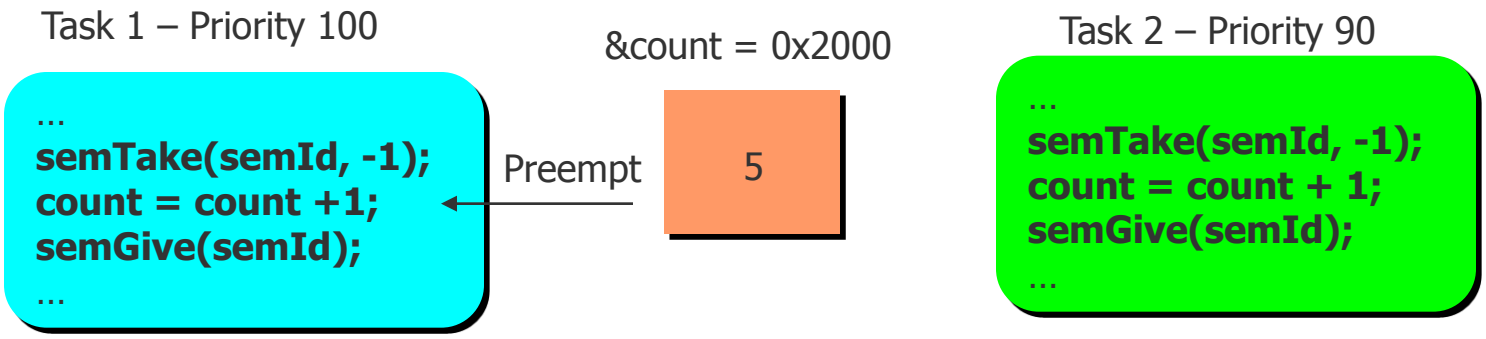
Event Driven Data



Handshake Protocol

Mutual Exclusion Semaphore

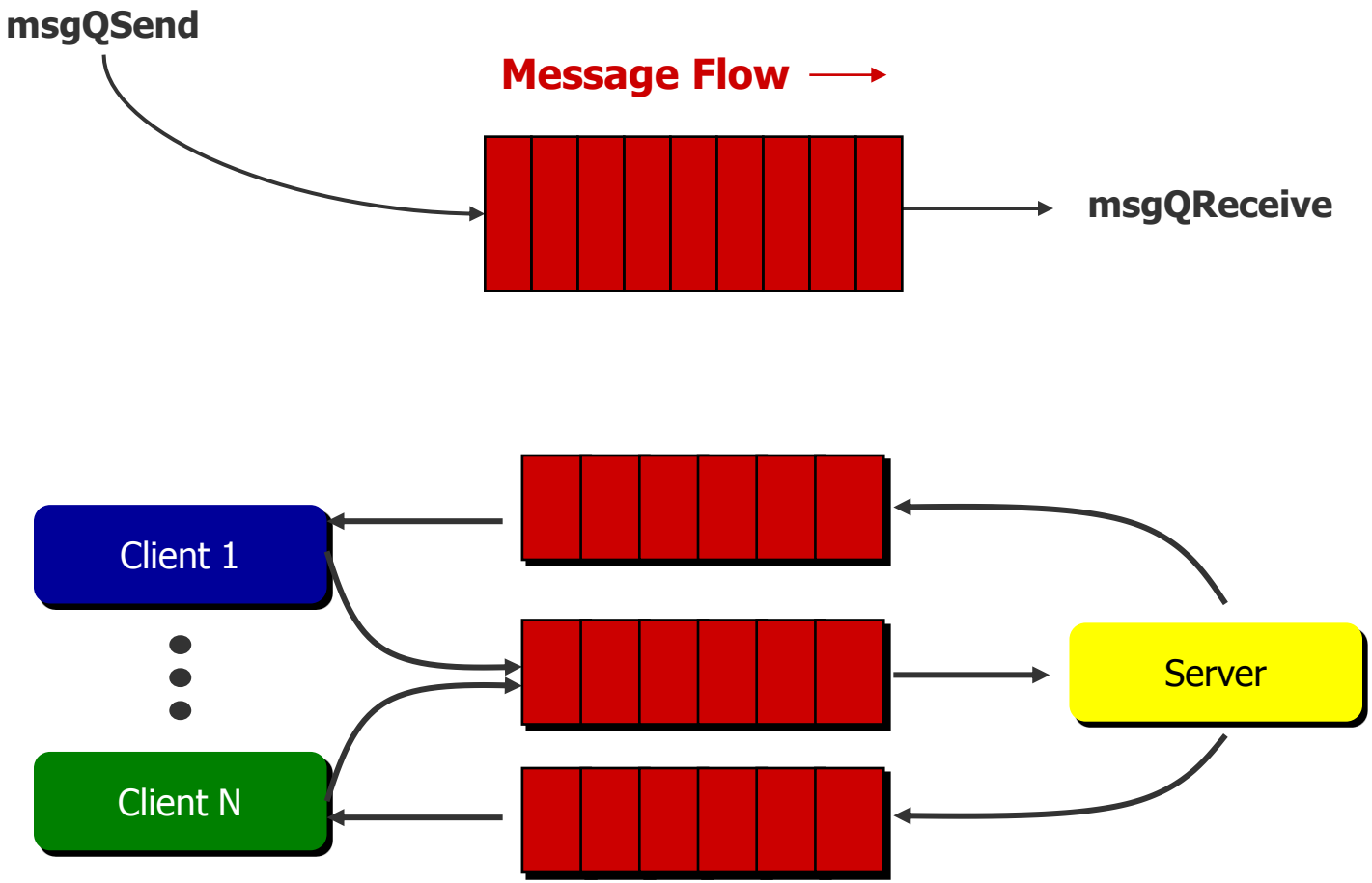
- Mutual exclusion semaphores (a.k.a., mutexes) are used to protect resources
  - Concept of ownership
  - Recursive access
  - Avoid race conditions in multi-core and multi-tasking use cases



Message Queues and Sockets

- These IPC constructs have you copy the data into the construct itself
  - The waiting code is blocked until the data shows up in the construct
- Good way to de-couple the source and sink of data when they run at different rates
- These act like buffers with a finite depth and *automagic* handshaking
- IP sockets can be used to go off board to a camera, Arduino or operator interface via the wireless LAN interface

Message Queue Operation



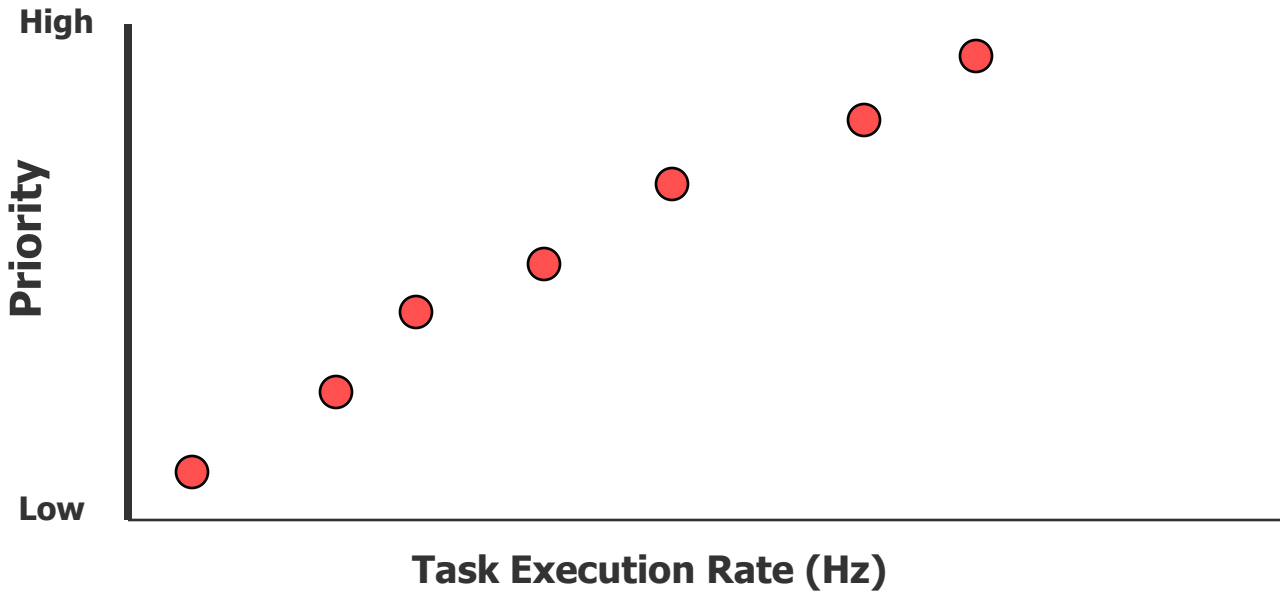
General Rules of Thumb for R-T

- Never sleep/delay in the main loop
  - If you must delay, use a separate timer and check the time in the loop
  - Better yet, move the delay into a separate thread and use IPCs to trigger the execution of the thread
- Don't allocate/free memory at run time
  - This includes creating and destroying objects, be careful with C++ and Java (control how and when objects are created/destroyed)
  - Allocating/freeing memory locks interrupts and can cause you to miss deadlines
- Don't create/delete threads at run time
  - This allocates/frees memory and perturbs the scheduler
- Avoid formatted output
  - Debug messages take time!
  - Some times buffered, some times not
- Engineering effort, not like other environments
  - Temporal requirements
  - Mathematical analysis
  - Rigorous design, code and test (just like ME and EE designs)

Rate Monotonic Analysis

- RMA is a temporal analysis method
  - Will my tasks get the time they need?
- Prerequisites:
  - Pre-emptive priority-based scheduling
  - Run scheduler any time ready queue might change
  - Faster Rate = Greater priority
- Rate Monotonic Scheduling not directly supported in VxWorks
  - But you can do the analysis and assign priorities manually
  - Assume worst case period for aperiodic tasks
  - Think of ISRs as “super” priority tasks
  - Do simple analysis in a spreadsheet (OK for FRC) or buy dedicated tool

Rate Monotonic Analysis

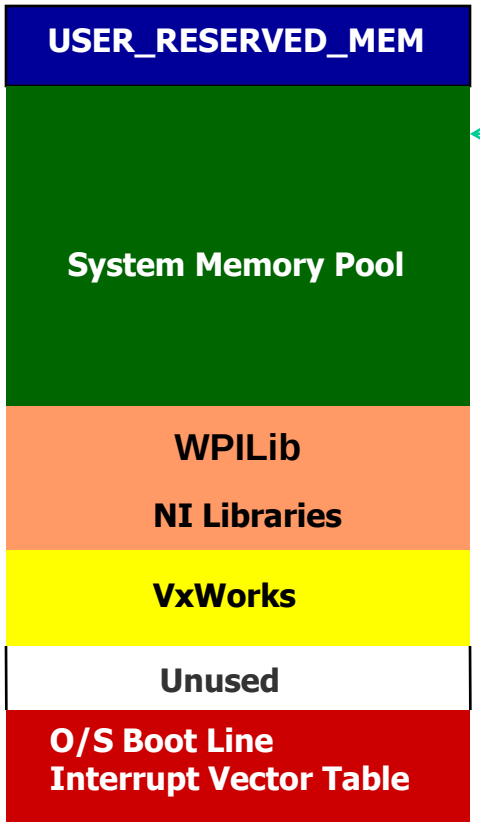


$$\sum_i \frac{E_i}{T_i} \leq n(2^{1/n} - 1)$$

E_i = max execution time of task i
T_i = execution period of task i

This formula means that total utilization must be < 70%!

What is Running On the Robot?



Application
(Loaded by OS During Boot Cycle)

Application
(Loaded With the Debugger)

Summary

- Well, we're out of time and we've only scratched the surface
- Next year's control system brings a lot of new opportunities for advanced control software
  - However, dual cores opens possibility for major lock ups due to race conditions
- NI's R-T Linux has most of the same capabilities as VxWorks™ on the current control system
  - The learning curve is significant, but the rewards will be worth it ☺