Principles of Cyber-Physical Systems

Introduction

Instructor: Lanshun Nie

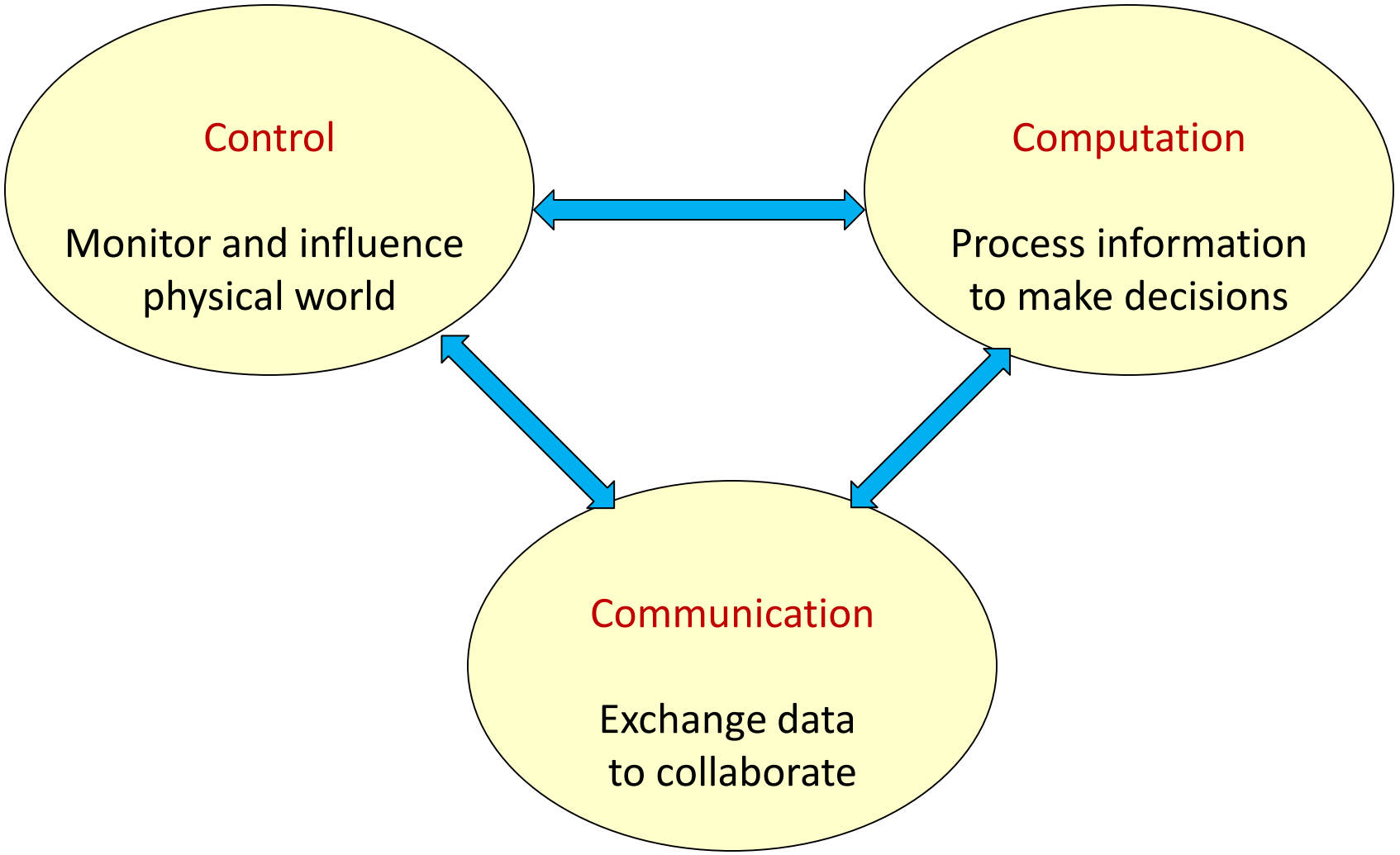
Embedded Software Systems Everywhere!

Software Inside

From Desktops to Cyber-Physical Systems

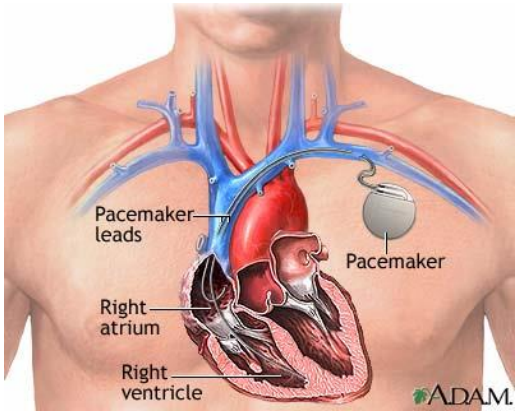
- Traditional computers: Stand-alone device running software applications (e.g. data processing)
- Traditional controllers: Devices interacting with physical world via sensors and actuators (e.g. thermostat)
- Embedded Systems
  - Special-purpose system with integrated microcontroller/software
  - Cameras, watches, washing machines…

Cyber-Physical Systems

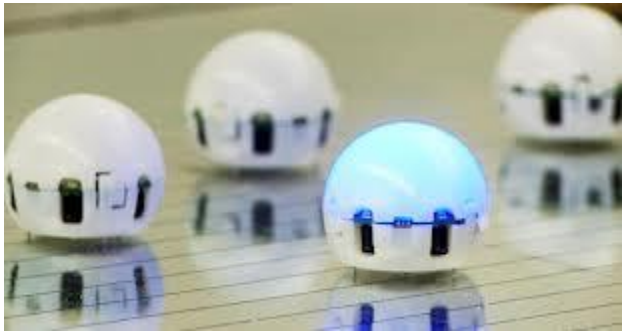


Cyber-Physical Systems

Driverless Cars



Medical devices



Coordinating robots

Design of Cyber-Physical Systems

Systems that integrate control, computation, and communication can do

Cool things,

And useful things…

Lots of promise and potential: medicine, transportation, energy, …

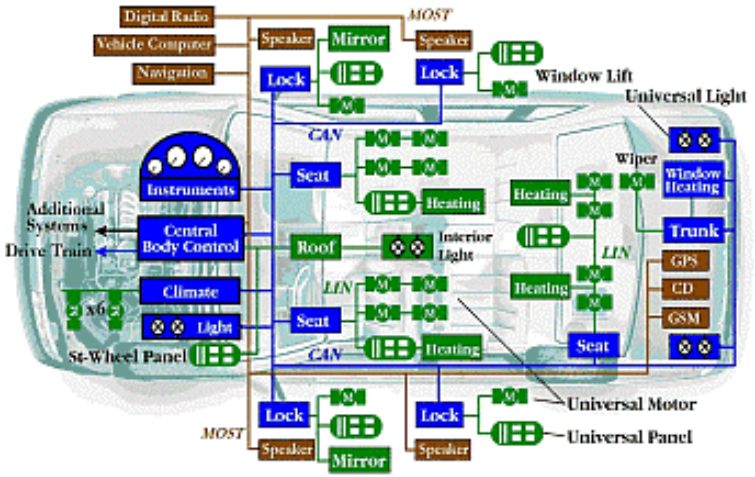
So what's the challenge?

Ariane 5 Explosion

“It took the European Space Agency 10 years and $7 billion to produce Ariane 5. All it took to explode that rocket less than a minute into its maiden voyage last June, scattering fiery rubble across the mangrove swamps of French Guiana, was a small computer program trying to stuff a 64-bit number into a 16-bit space”

A bug and a crash, J. Gleick, New York Times, Dec 1996

Prius Brake Problems Blamed on Software Glitches



“Toyota officials described the problem as a "disconnect" in the vehicle's complex anti-lock brake system (ABS) that causes less than a one-second lag. With the delay, a vehicle going 60 mph will have traveled nearly another 90 feet before the brakes begin to take hold”

CNN Feb 4, 2010

Software: The Achilles' Heel

Software everywhere means bugs everywhere

2002 study by NIST:

Software bugs cost US economy $60 billion annually (0.6% of GDP)

Lack of trust in software as techr

Would you use an autonom toring and drug-delivery device?

Grand challenge for computer science:

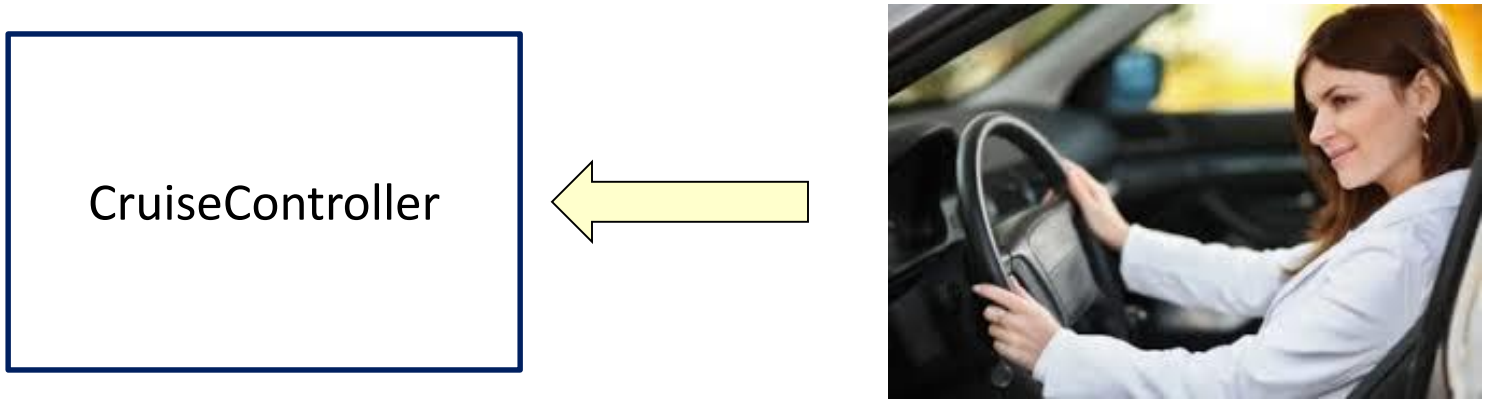
Technology for designing reliable cyber-physical systems

Let's Design a Cruise Controller

What's the goal of a cruise controller?

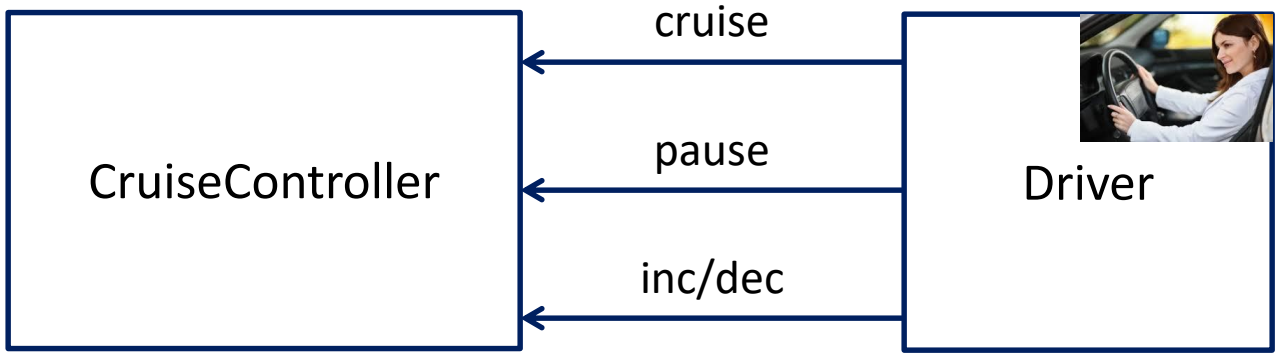
Automatically adjust the speed of the car so that it matches the speed desired by the driver

Block Diagrams of High-Level Design



How does this component interact with the rest of the world ?

Interfaces for Components: Inputs and Outputs



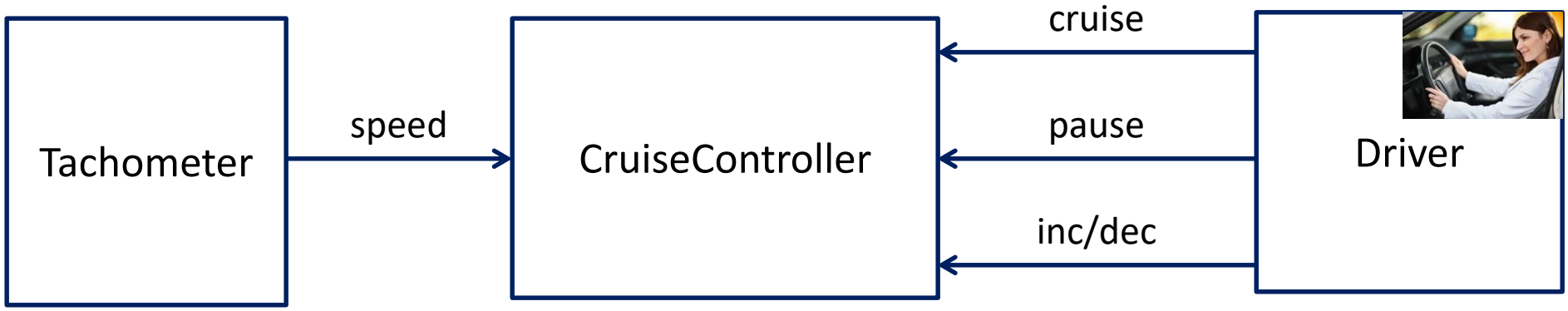
Driver interacts with the system using 4 buttons:

Cruise button to turn the cruise on or off

Pause button to suspend/restart its operation

Inc and Dec buttons to increment or decrement desired speed

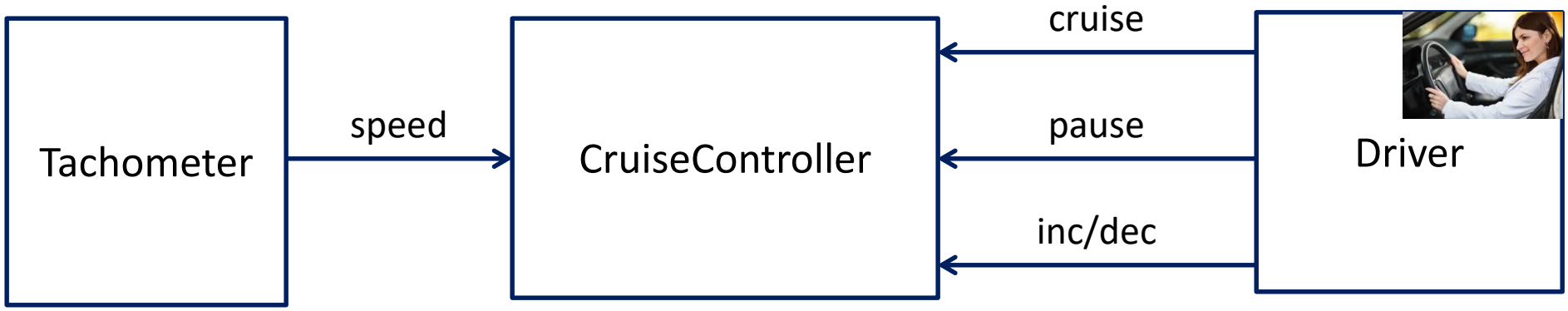
Interfaces for Components: Inputs and Outputs



What other information does the cruise-controller need ?

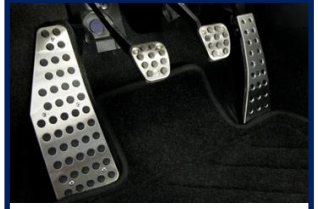
And who supplies it?

Interfaces for Components: Inputs and Outputs



What should be the outputs of the cruisecontroller?

And who needs these outputs?



Throttle

Force

Tachometer

speed

CruiseController

cruise

pause

inc/dec

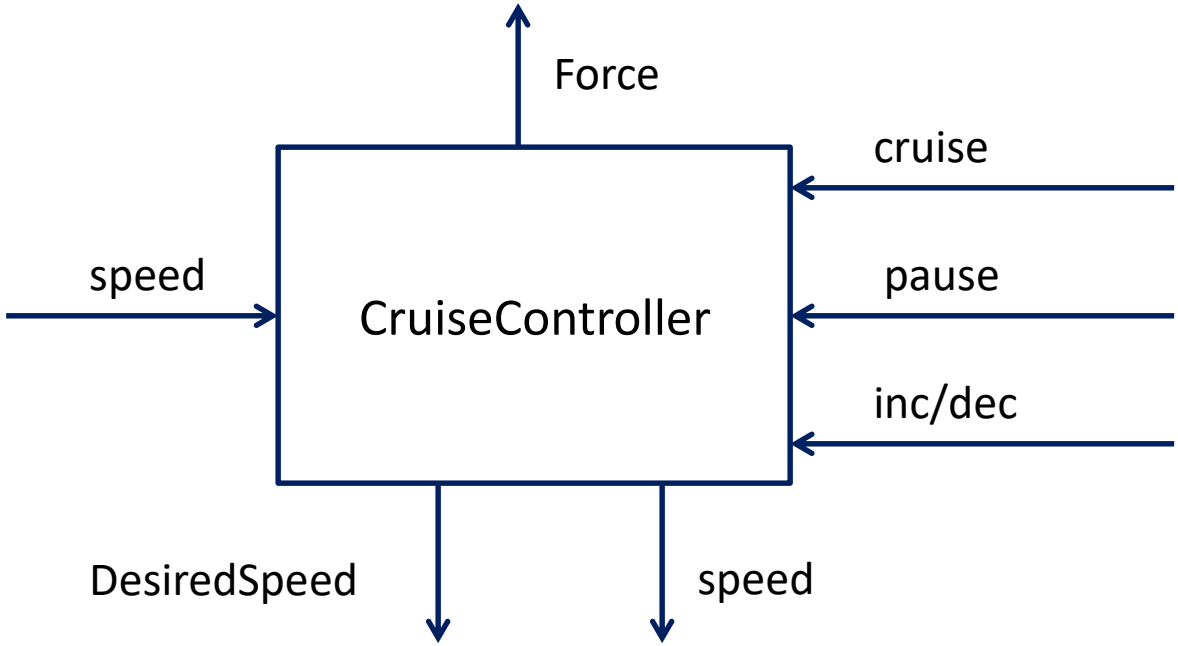
Driver

DesiredSpeed

speed

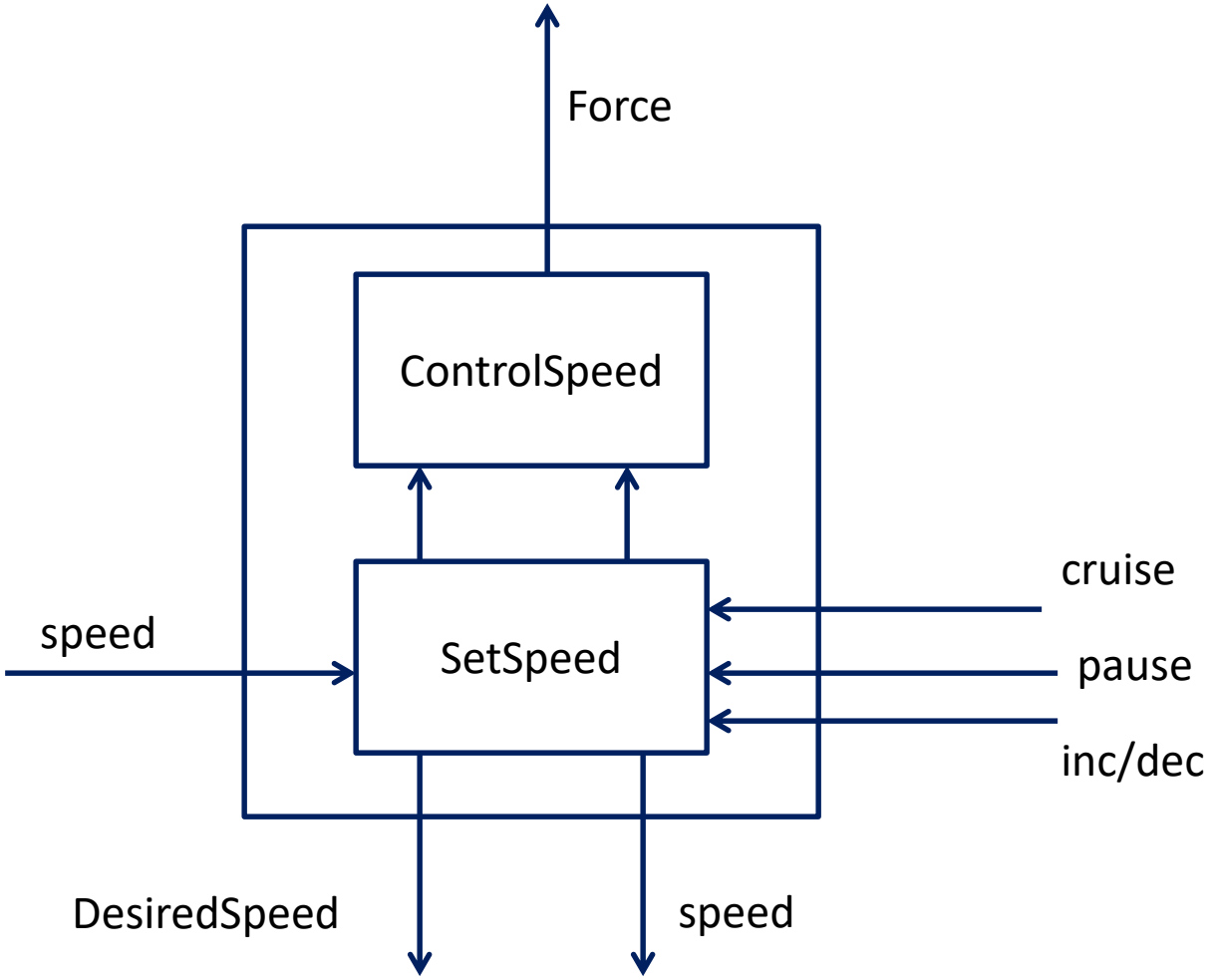
Display

Compositional Design

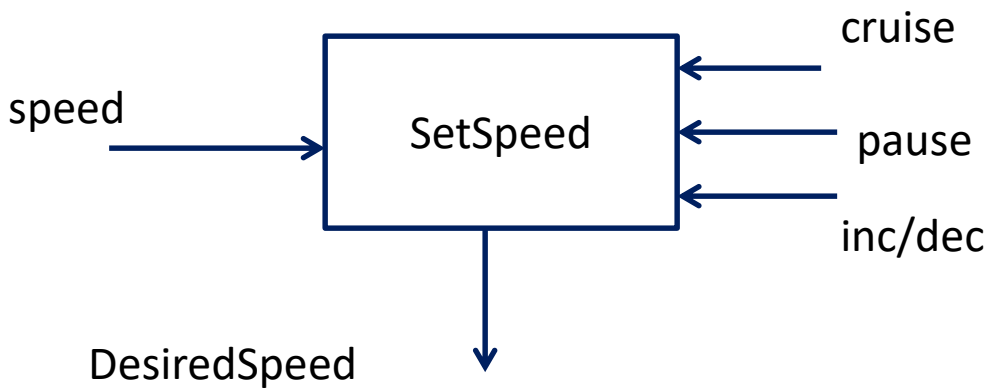


How to break up the computation of the cruise controller into subtasks?

Decomposing the Cruise Controller

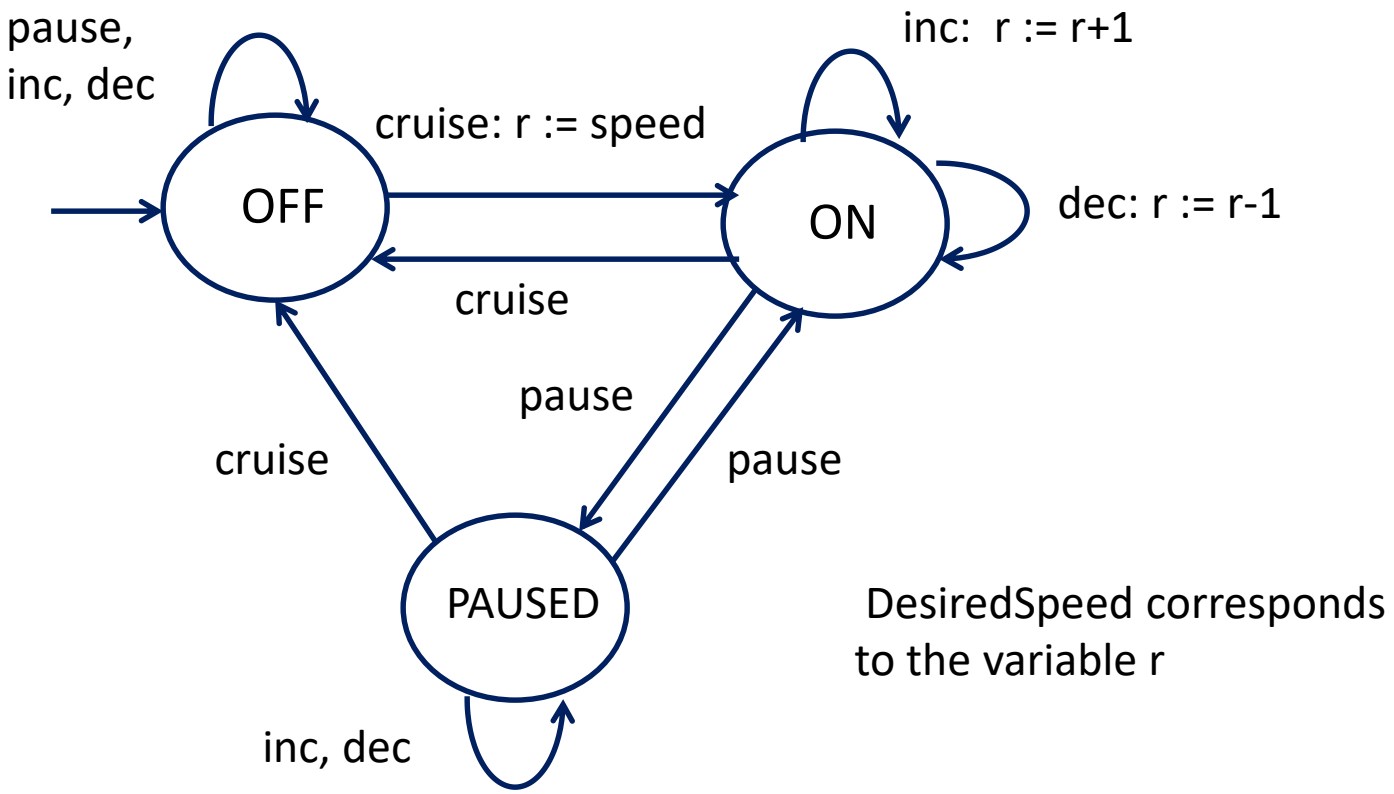


Designing SetSpeed Component

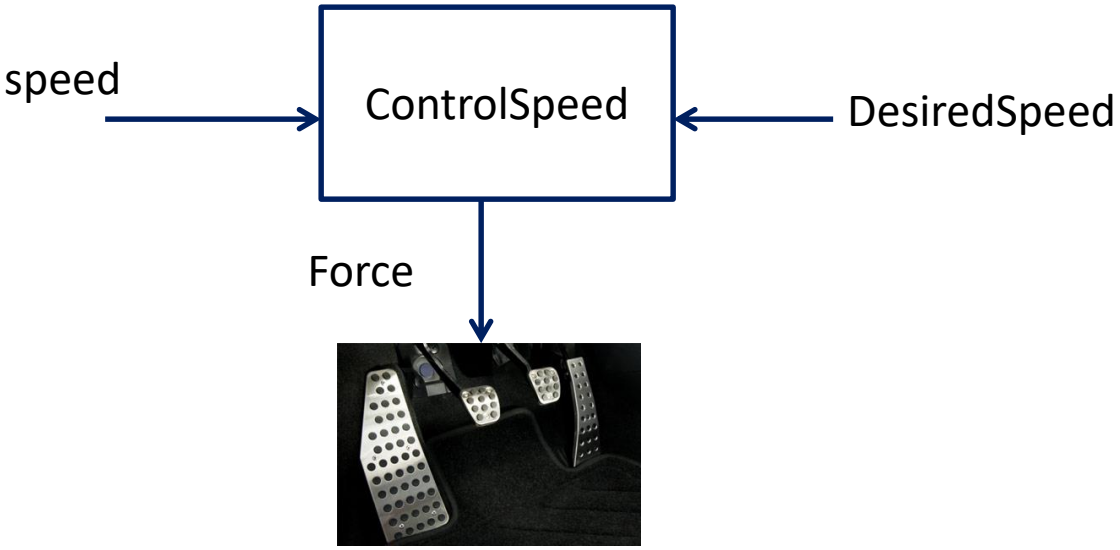


Goal: Compute the desired cruising speed in response to the commands from the driver

Designing SetSpeed: State Machines

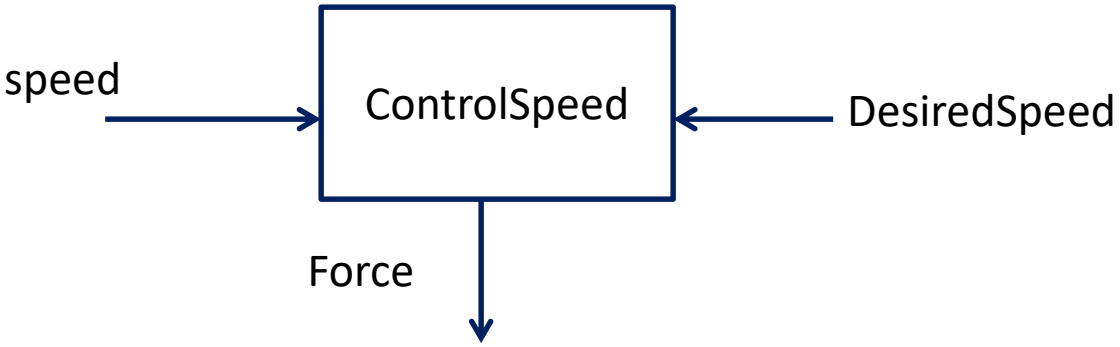


Designing ControlSpeed Component



Goal: Determine the force to be applied to throttle so that speed becomes equal to DesiredSpeed

Capturing Requirements



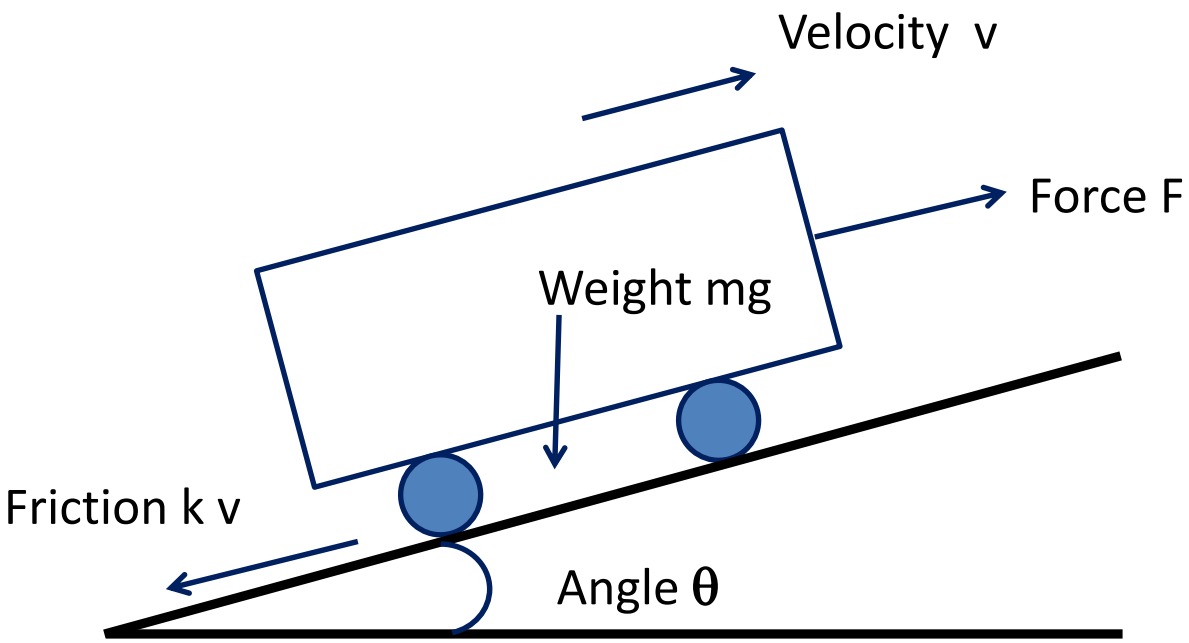
Requirements: Mathematically precise description of what a system is supposed to do.

Writing requirements is key to ensuring reliability of systems

Requirement 1: Actual speed eventually converges to desired speed

Requirement 2: Speed of the car stays “stable”

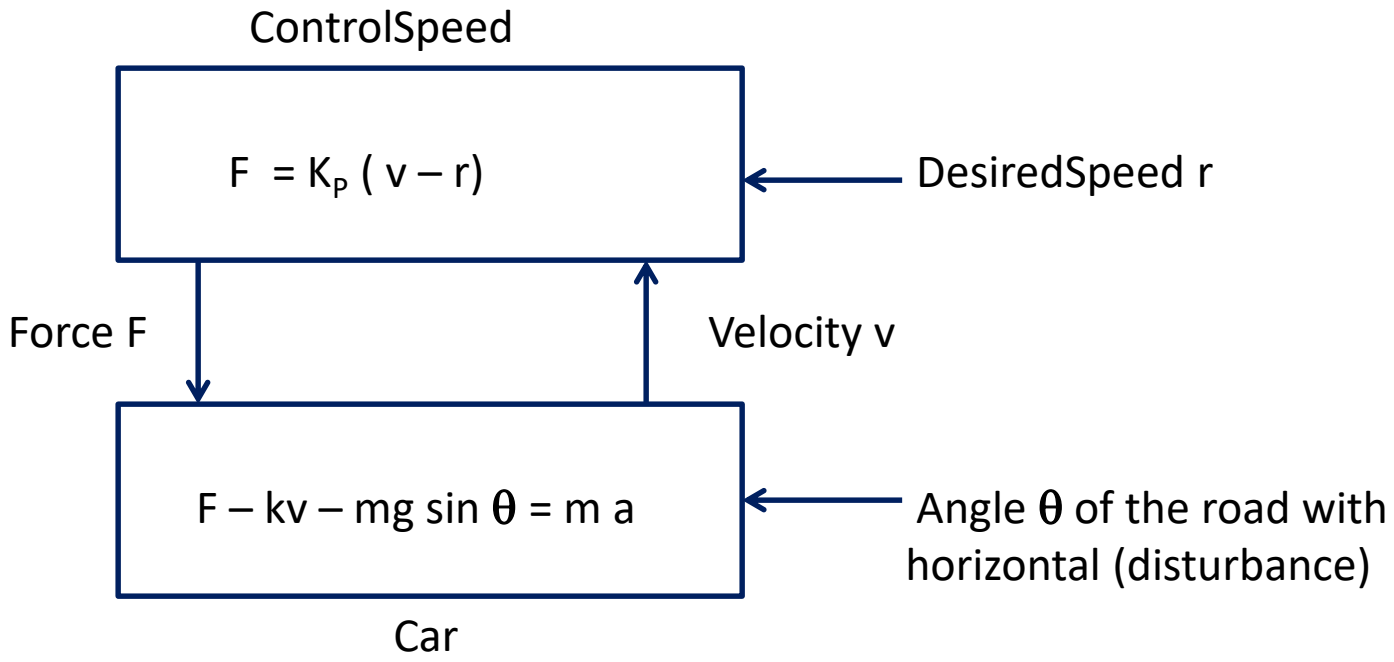
A bit of Physics: Modeling a car



Newton's law of motion gives

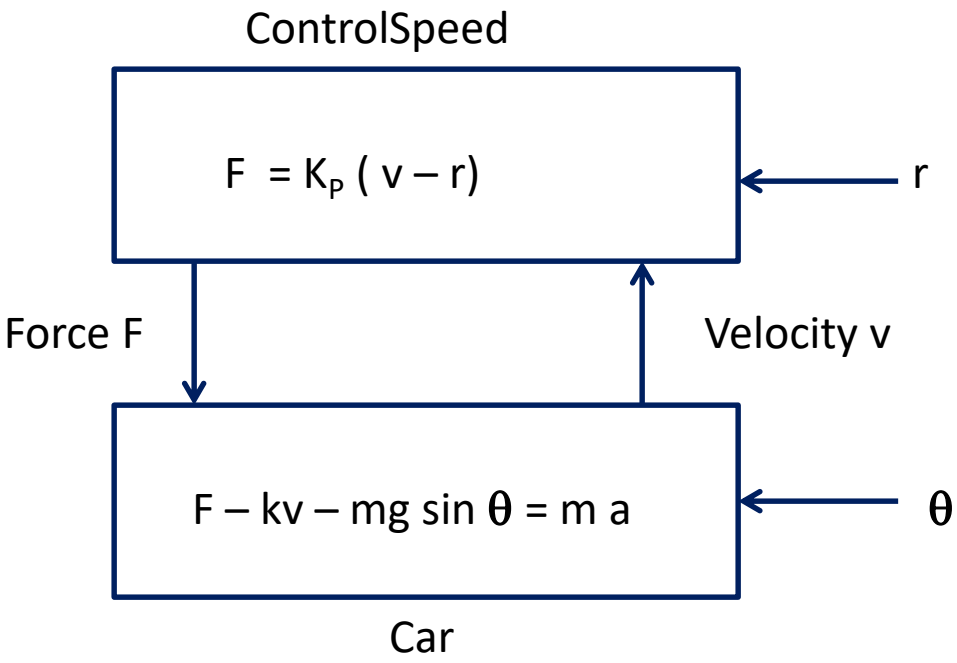
$$F - kv - mg \sin\theta = m\,a$$

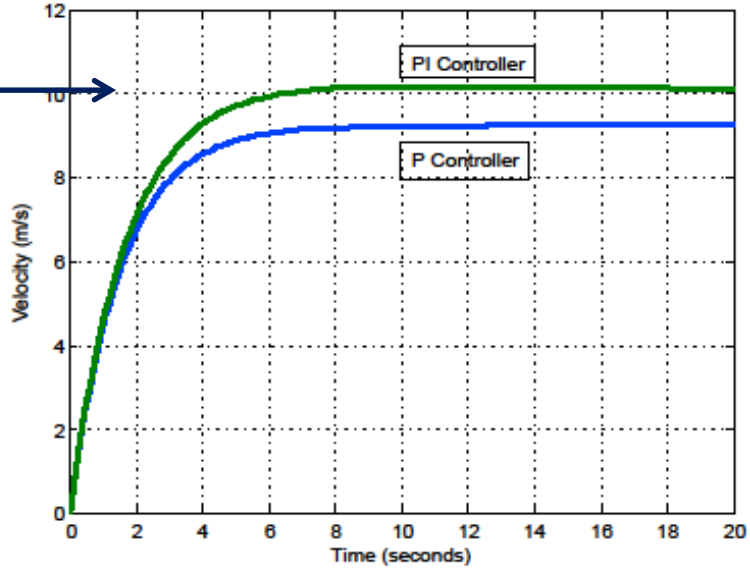
ControlSpeed Component



Control Theory: Mathematical techniques to compute force (F) as a function of velocity (v) and desired speed (r)

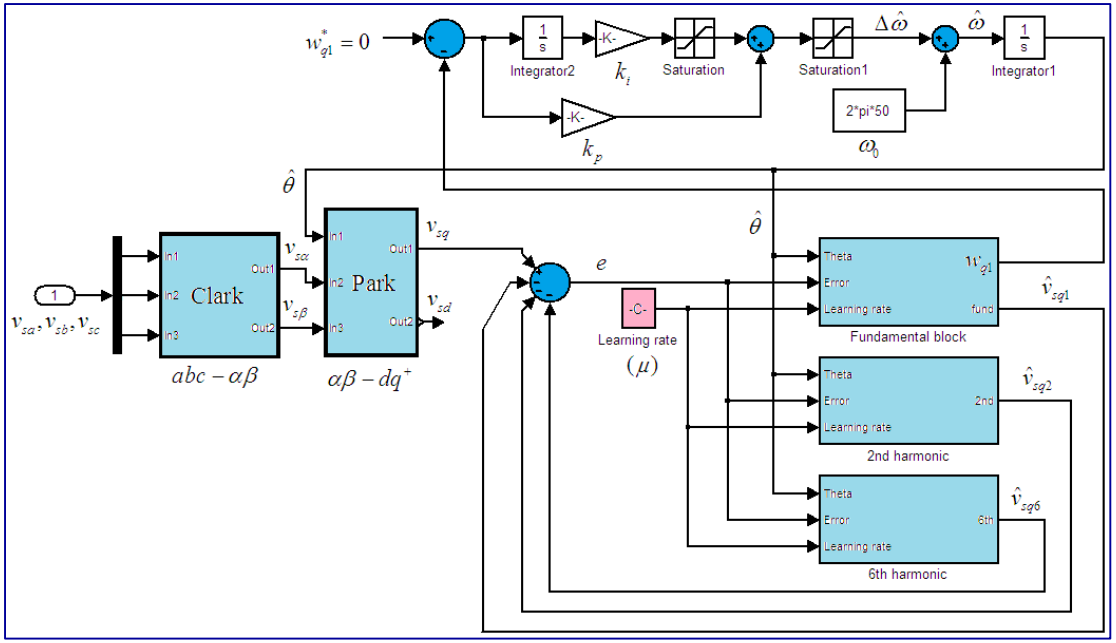
Does our controller work ?





Verification Tools: Allow you to check if system model indeed works as expected, that is, satisfies requirements

Model-based design != Coding

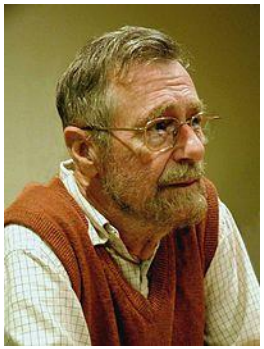



```python
#make sure that we are allowed to recurse many times
import sys
sys.setrecursionlimit(20000)

def probOfStreak(numCoins, minHeads, headProb, saved=None):
    #Computes the probability (i.e. S[n,k]) of getting a run of minHeads
    #(i.e. K) or more heads in a row out of numCoins
    #(i.e. N) independent coin tosses where the probability of getting a
    #heads each time is headProb (i.e. p) and the
    #probability of a tails is 1-headProb (i.e. q).
    #We will be using the recursion:
    #S[N,K] = p^K + sum_{j=1,K} p^(j-1) (1-p) S[N-j,K]
    #As well as the base cases   S[0,k] = 0  and  S[N,K] = 0 for K>N

    #if it's our first call, allocate a hash table to store saved values
    if saved == None: saved = {}

    #get a unique identifier for the value that they want to compute
    ID = (numCoins, minHeads, headProb)

    #if it has been computed before, just return the precomputed value.
    #there is no point in wasting time computing the same thing again.
    if ID in saved: return saved[ID]
    else: #if it's never been computed before
        #handle the base case where we have no coins or where we have
        #more heads we are looking for than we have coins
        if minHeads > numCoins or numCoins <= 0:
            result = 0
        else:
            #use our recursive relationship to compute S[n,k] by
            #breaking it into a sum of terms involving S[n-j,k] for 1<=j<=k
            result = headProb**minHeads  #S[n,k] = p^k + ...
            #S[n,k] = ... + sum_{j=1,k} p^(j-1) (1-p) S[n-j,k]
            for firstTail in xrange(1, minHeads+1):
                pr = probOfStreak(numCoins-firstTail, minHeads, headProb, saved)
                result += (headProb**(firstTail-1))*(1-headProb)*pr
        #save the resulting value so that we can use it later, if need be
        saved[ID] = result

        #return the computed value
        return result
```

Design using high-level block diagrams and state machines gets automatically compiled into low-level code !

Models not only of system being designed, but also of its environment

Verification != Simulation/Testing



Program testing can be used to show the presence of bugs, but never their absence!

Edsger W. Dijkstra

Formal Verification

- Goal: Establish that model satisfies requirements under all possible scenarios
- First challenge: Need formal definitions of "model" and "requirement" to make the problem mathematically precise
- Second challenge: Need verification techniques and tools

Course Topics

- Goal: Introduction to principles of design, specification, analysis and implementation of CPS
- Disciplines
  - Model-based design
  - Concurrency theory
  - Distributed algorithms
  - Formal specification
  - Verification techniques and tools
  - Real-time systems
  - Hybrid systems
- Emphasis on mathematical concepts

Theme 1: Formal Models

- Mathematical abstractions to describe system designs
- Modeling formalisms
  - Synchronous models
  - Asynchronous models
  - Continuous-time dynamical systems
  - Timed models
  - Hybrid systems
- Modeling concepts
  - Syntax vs semantics
  - Composition
  - Input/output interfaces
  - Nondeterminism, fairness, …

Theme 2: Specification and Analysis

- Formal techniques to ensure correctness at design time
- Requirements
  - Safety (invariants, monitors)
  - Liveness (temporal logic, automata over infinite sequences)
  - Stability
  - Schedulability
- Analysis techniques
  - Deductive: Inductive invariants and ranking functions
  - Enumerative and symbolic search for state-space exploration
  - Model checking
  - Linear-algebra-based analysis of dynamical systems
  - Verification of timed and hybrid systems

Theme 3: Model-based Design

- Design and analysis of illustrative computing problems
- Design methodology
  - Structured modeling (bottom-up, top-down)
  - Requirements-based design and design-space exploration
- Case studies
  - Distributed coordination: mutual exclusion, consensus, leader election
  - Communication: Reliable transmission, synchronization
  - Control design: PID, cruise controller
  - CPS: Pacemaker, obstacle avoidance for robots, multi-hop control network