# Cyber-Physical Systems

## Revision

UNIVERSITY AT ALBANY
State University of New York

IECE 553/453– Fall 2019

Prof. Dola Saha

# Final Examination

- Dec 16
- 8AM-10AM
- ES 019
- Closed books, closed Notes
- Syllabus : As discussed in class and lab throughout the semester
- Separate questions for undergrads and grads

# What did you learn?

# Application Domains – major societal impact

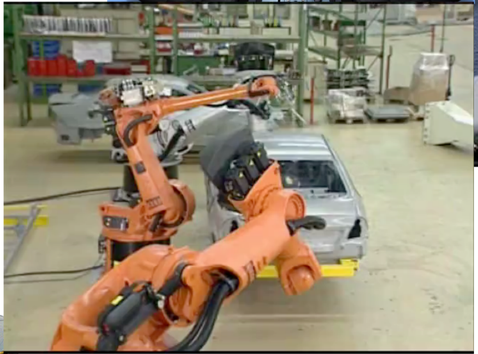
- Agriculture, Aeronautics, Building design, Civil infrastructure, energy, environmental quality, healthcare and personalized medicine, Manufacturing, and transportation.

# CPS

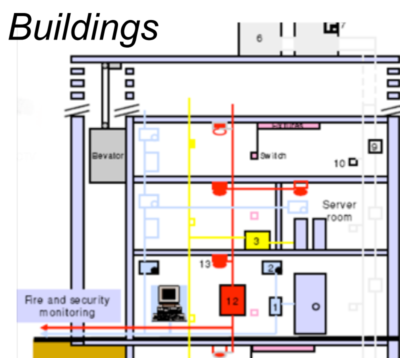
- Cyber + Physical
- Computation + Dynamics + Communication
- Security + Safety

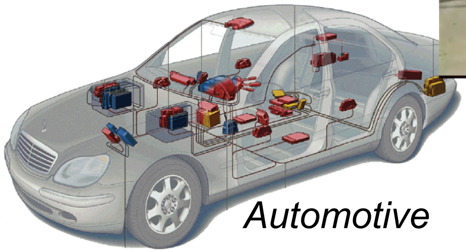






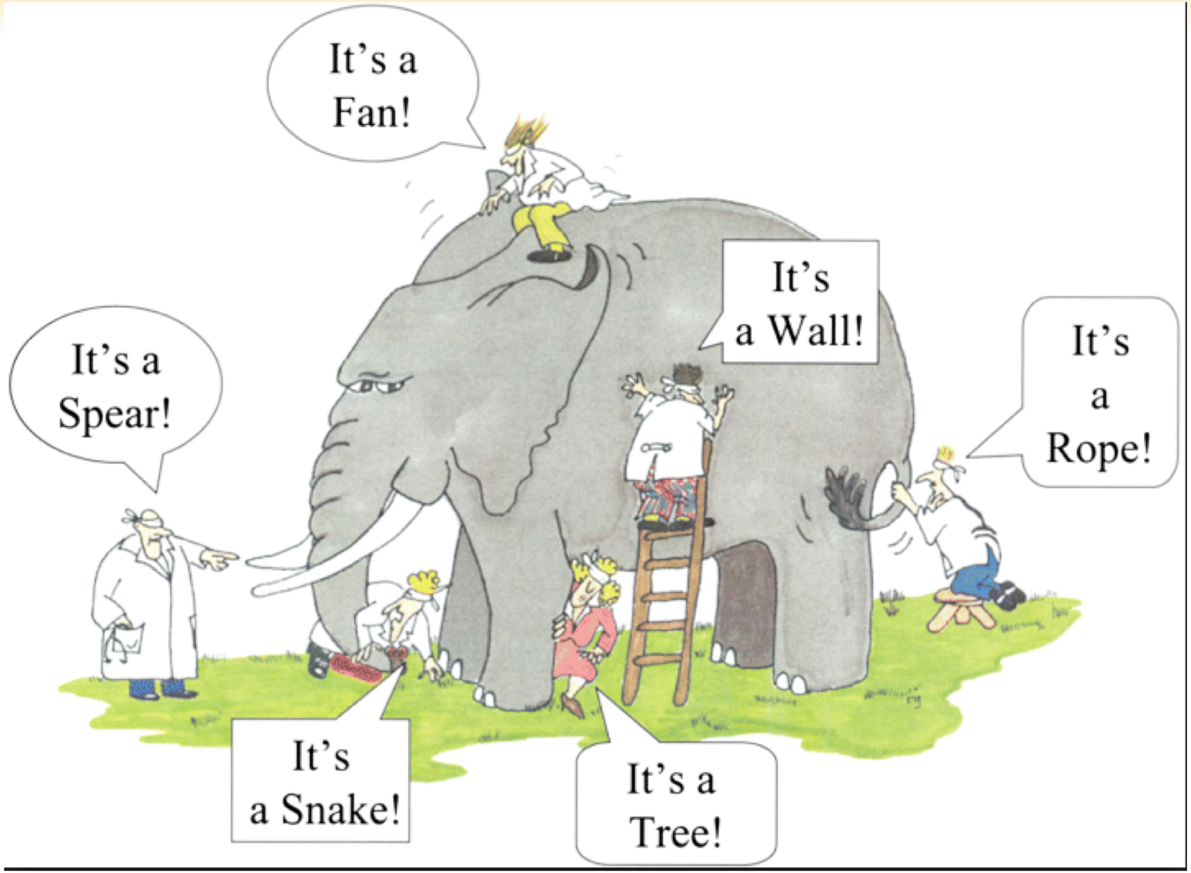
Manufacturing



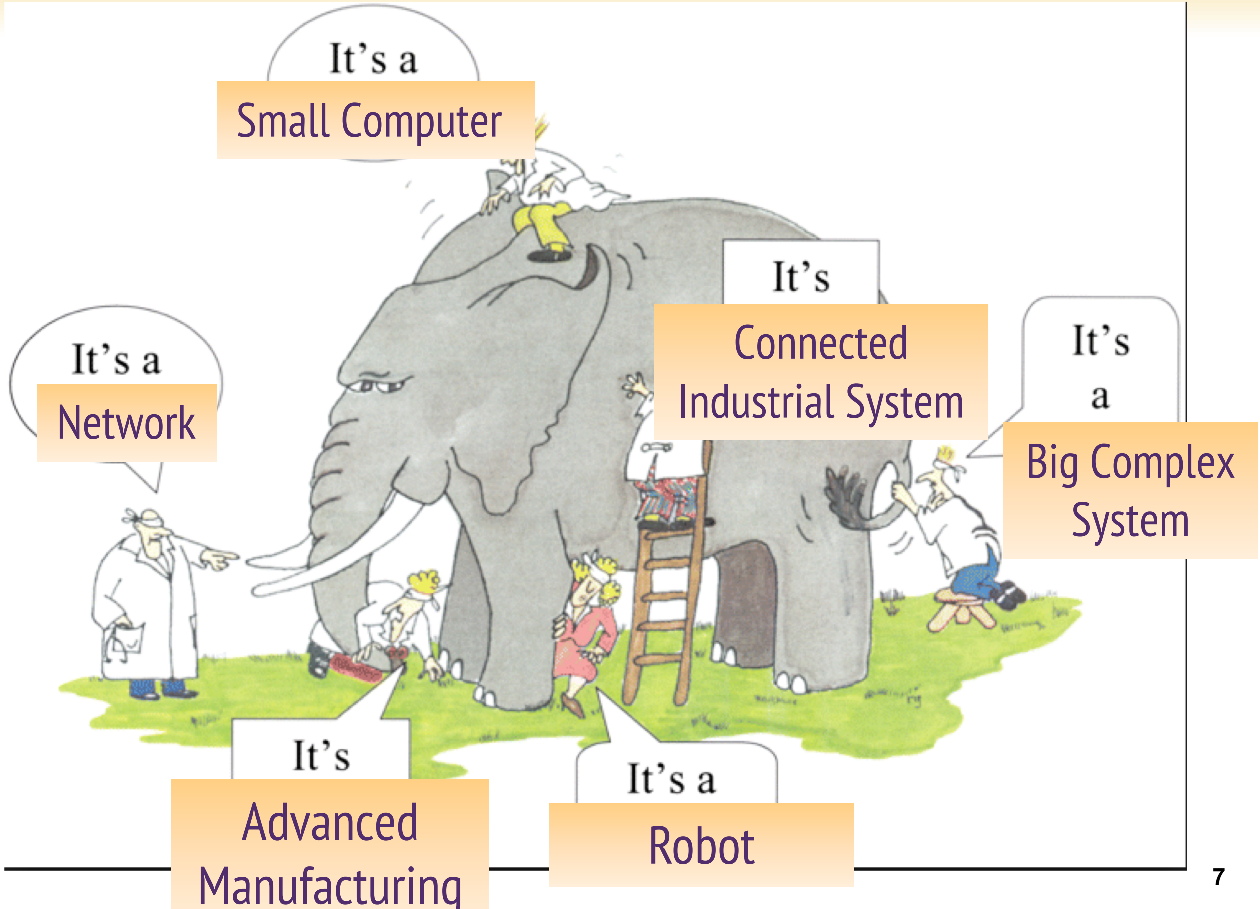



Automotive

# Challenges of Working in a Multidisciplinary Area



# Challenges of Working in a Multidisciplinary Area




UNIVERSITY AT ALBANY
State University of New York

# What is this course about?

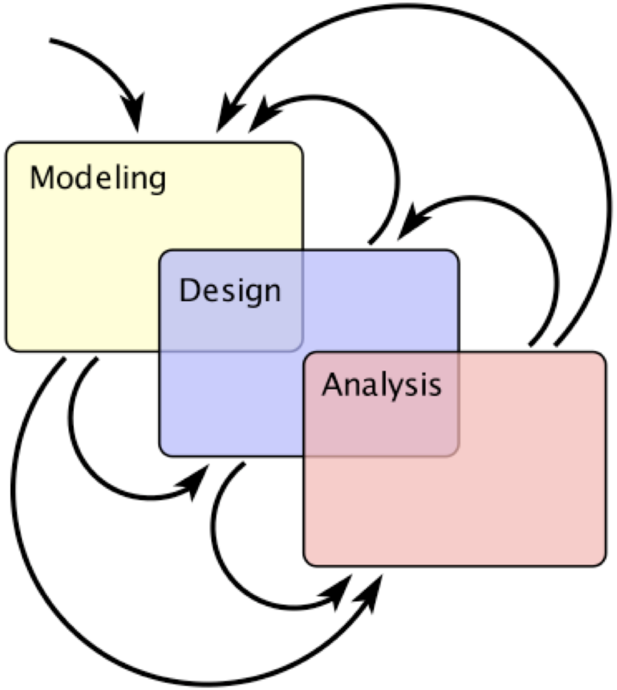
- A scientific structured approach to designing and implementing embedded systems
- Not just hacking and implementing
- Focus on model-based system design, on embedded hardware and software

# Model, Design & Analysis

➢ ***Modeling*** is the process of gaining a deeper understanding of a system through imitation. Models specify **what** a system does.

➢ ***Design*** is the structured creation of artifacts. It specifies **how** a system does what it does. This includes optimization.

➢ ***Analysis*** is the process of gaining a deeper understanding of a system through dissection. It specifies **why** a system does what it does (or fails to do what a model says it should do).



# What is a sensor? An actuator?

- A sensor is a device that measures a physical quantity
- → Input / "Read from physical world"

- An actuator is a device that modifies a physical quantity
- → Output / "Write to physical world"

# Sensor Model

- Linear and Affine Functions

$$f\big(x(t)\big) = ax(t)$$

$$f\big(x(t)\big) = ax(t) + b$$

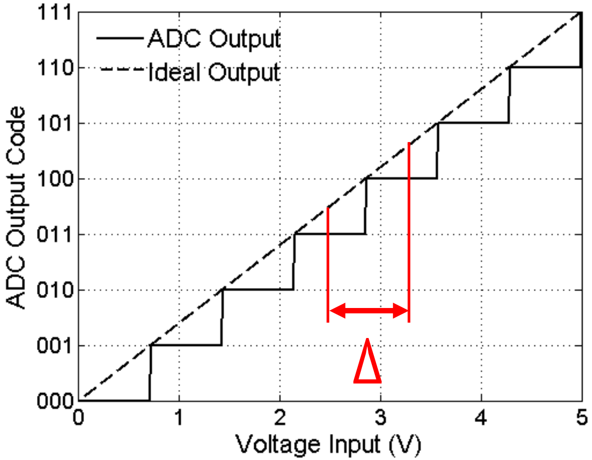
- Affine Sensor Model

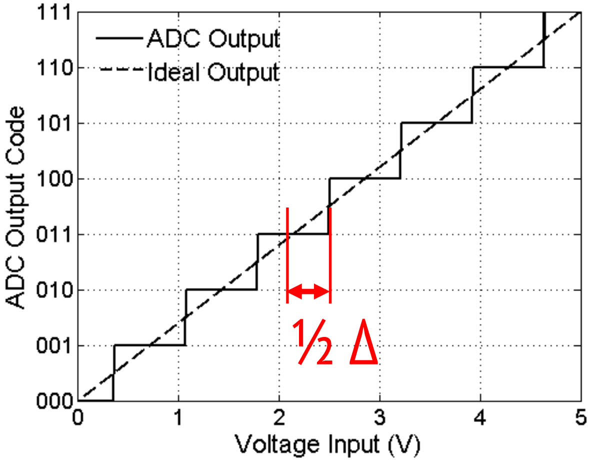
$$f\big(x(t)\big) = ax(t) + b + n$$

- Sensitivity (a), Bias (b) and Noise (n)
  - Sensitivity specifies the degree to which the measurement changes when the physical quantity changes

# Resolution

- Resolution is determined by number of bits (in binary) to represent an analog input.
- Example of two quantization methods (N = 3)




$$\text{Digital Result} = \text{floor}\left(2^3 \times \frac{V}{V_{REF}}\right)$$

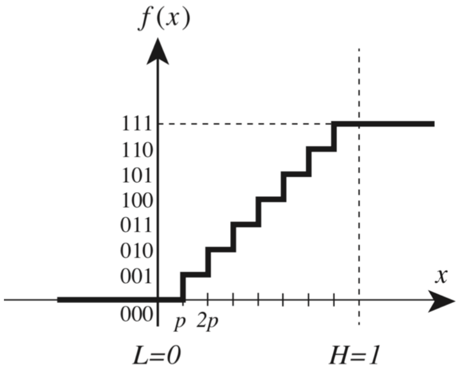
Max quantization error = $\Delta = V_{REF}/2^3$

$$\text{Digital Result} = \text{round}\left(2^3 \times \frac{V}{V_{REF}}\right)$$

Max quantization error = $\pm\frac{1}{2}\Delta = \pm V_{REF}/2^4$

$$\text{round}(x) = \text{floor}(x + 0.5)$$

# Range and Dynamic Range




- Range

$$f(x(t)) = \begin{cases} ax(t) + b & \text{if } L \leq x(t) \leq H \\ aH + b & \text{if } x(t) > H \\ aL + b & \text{if } x(t) < L, \end{cases}$$

- Dynamic Range

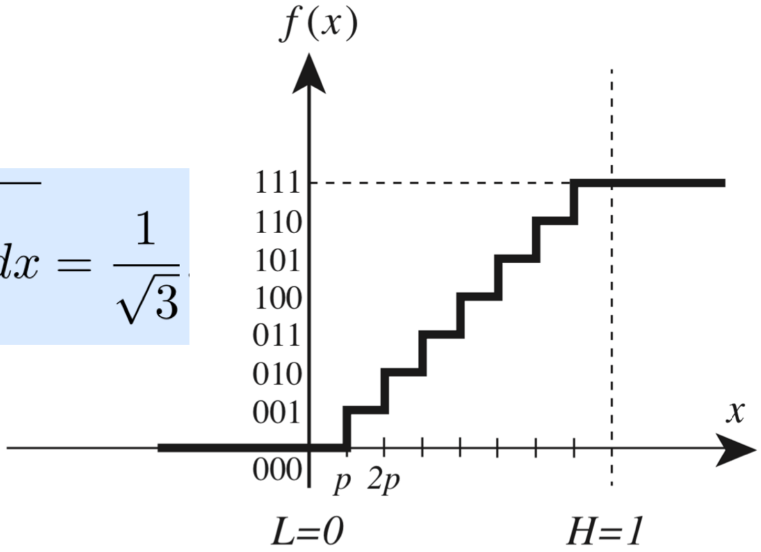
$$D = \frac{H - L}{p}, \qquad D_{dB} = 20 \log_{10} \left( \frac{H - L}{p} \right)$$

# Noise modeled as statistical property

- x(t) is a random variable with uniform distribution ranging from 0 to 1

$$X = \sqrt{\int_0^1 x^2 dx} = \frac{1}{\sqrt{3}}$$

- n(t) = f(x(t)) − x(t)
  - ranges from −1/8 to 0

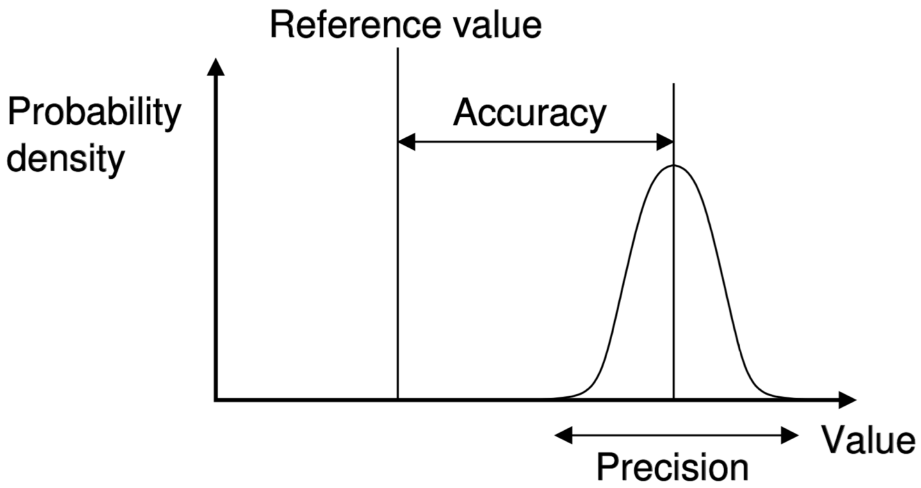



$$N = \sqrt{\int_{-1/8}^{0} 8n^2 dn} = \sqrt{\frac{1}{3 \cdot 64}} = \frac{1}{8\sqrt{3}}$$

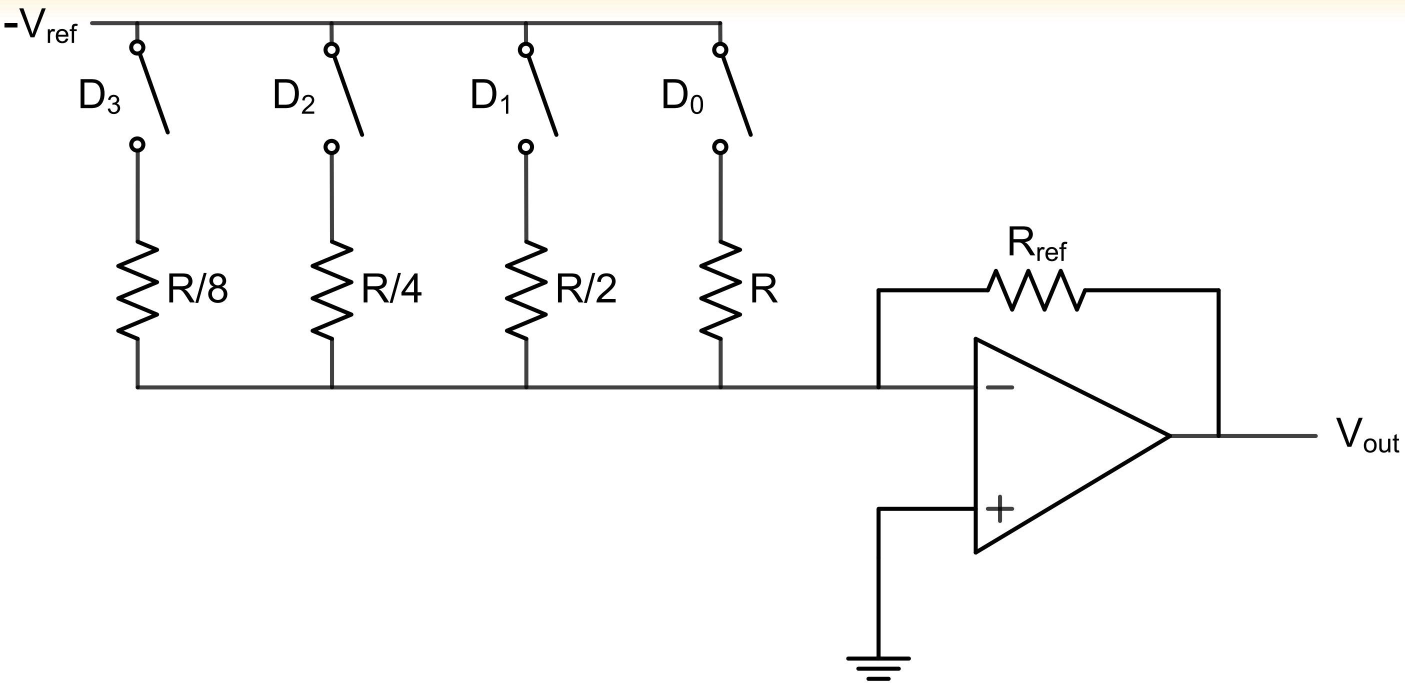
$$SNR_{dB} = 20\log_{10}\left(\frac{X}{N}\right) = 20\log_{10}(8) \approx 18dB$$

# Precision and Accuracy

- Precision: how close the two measured values can be
- Accuracy: how close is the measured value to the true value



# Binary-weighted Resistor DAC




$$V_{out} = V_{ref} \times \frac{R_{ref}}{R} \times (D_3 \times 2^3 + D_2 \times 2^2 + D_1 \times 2 + D_0)$$

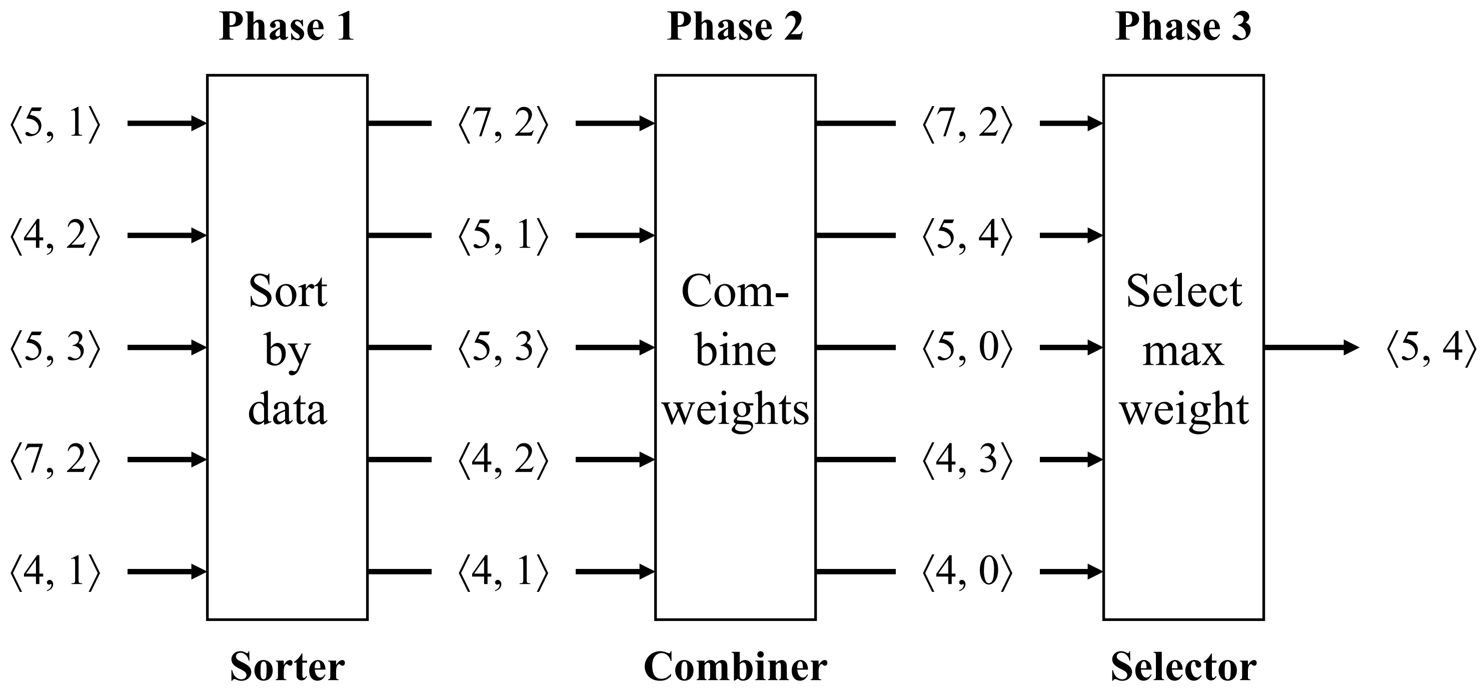
# Sensor Fusion: Marzullo's Algorithm

- Axiom: if sensor is non-faulty, its interval contains the true value
- Observation: true value must be in overlap of non-faulty intervals
- Consensus (fused) Interval to tolerate f faults in n: Choose interval that contains all overlaps of n – f; i.e., from least value contained in n – f intervals to largest value contained in n – f

# Weighted Plurality Voting Units

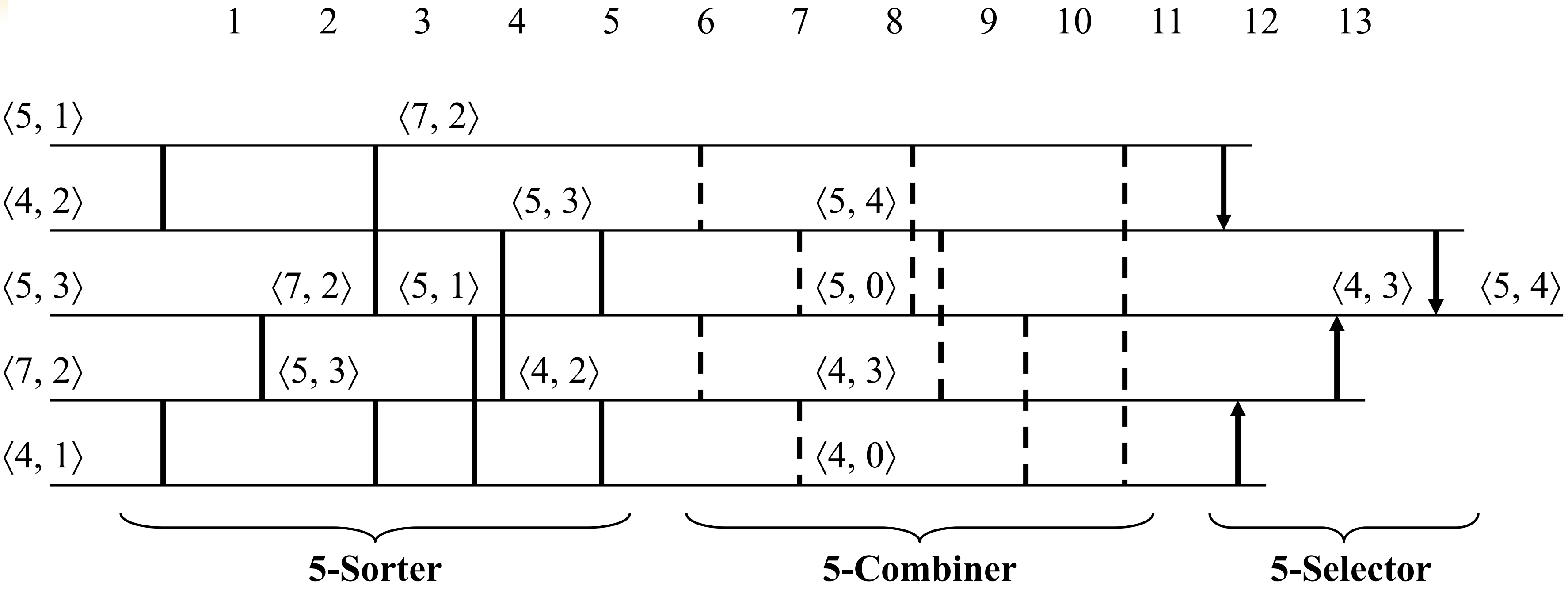
Inputs: Data-weight pairs

Output: Data with maximal support and its associated tally




Source: B. Parhami, *IEEE Trans. Reliability*, Vol. 40, No. 3, pp. 380-394, August 1991

# Stages of delay

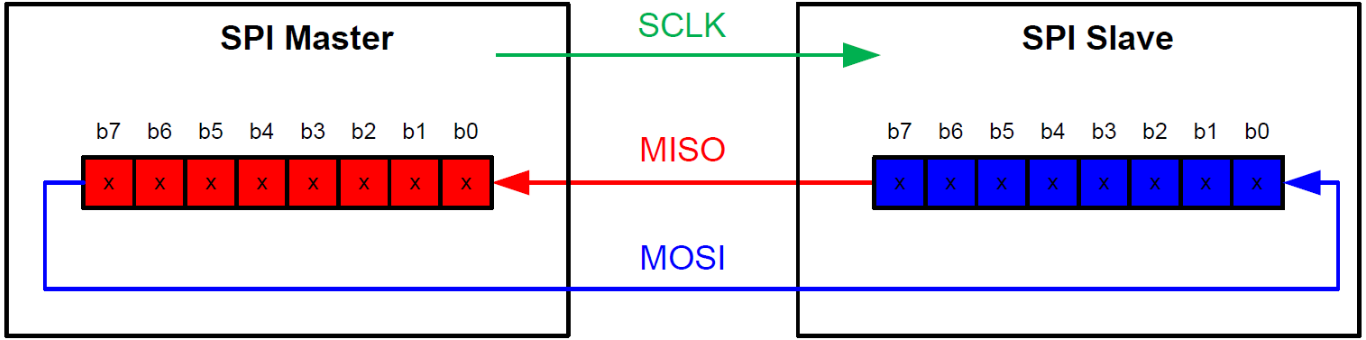



The first two phases (sorting and combining) can be merged, producing a 2-phase design – fewer, more complex cells (lead to tradeoff)

# Peripheral I/O

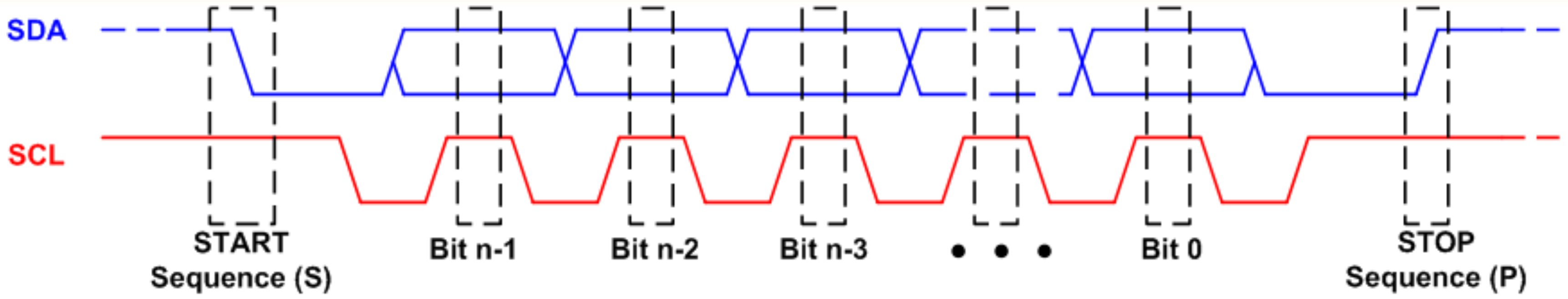
- GPIO
- SPI
- I2C
- UART
- USB
- CAN

# Serial Peripheral Interface (SPI)

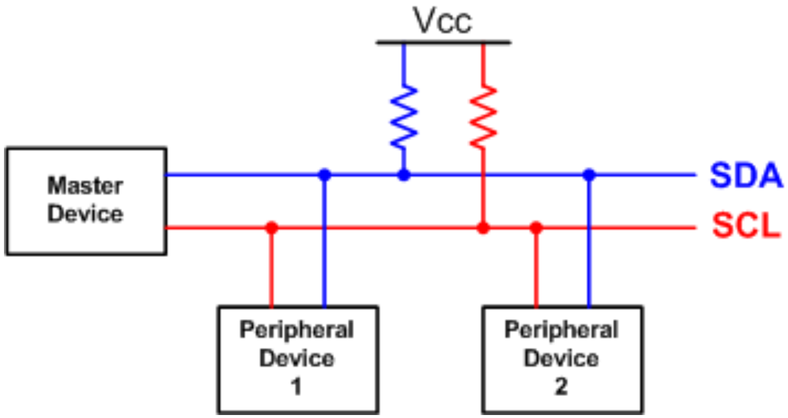



- Master has to provide clock to slave
- Synchronous exchange: for each clock pulse, a bit is shifted out and another bit is shifted in at the same time. This process stops when all bits are swapped.
- Only master can start the data transfer

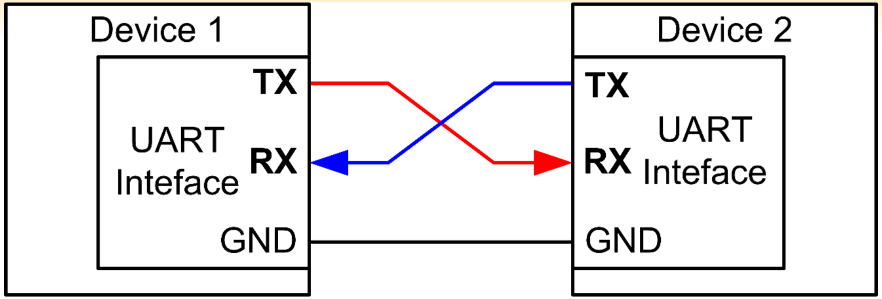
# Inter-Integrated Circuit (I2C)




- A **START** condition is a high-to-low transition on SDA when SCL is high.
- A **STOP** condition is a low to high transition on SDA when SCL is high.
- The address and the data bytes are sent most significant bit first.
- Master generates the clock signal and sends it to the slave during data transfer



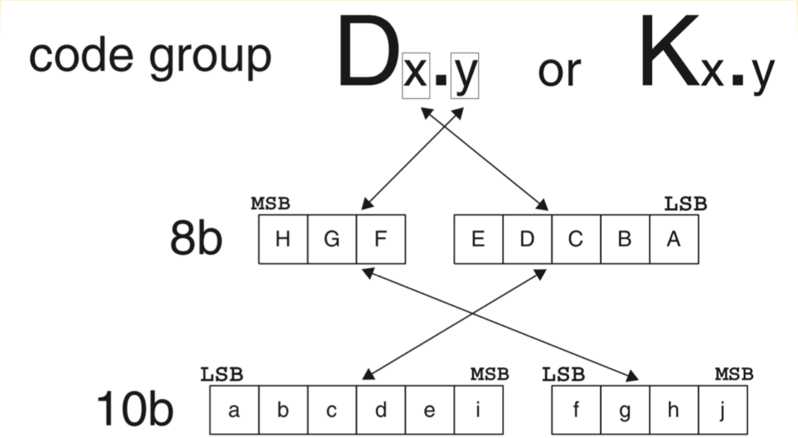
# Universal Asynchronous Receiver and Transmitter (UART)




- Universal
  - Programmable format, speed, etc.
- Asynchronous
  - Sender provides no clock signal to receivers
- Half Duplex
- Any node can initiate communication
- Two lanes are independent of each other

# 8b/10b Encoding

- ensure sufficient data transitions for clock recovery
- A DC-balanced serial data stream
  - it has almost same number of 0s and 1s for a given length of data stream.
  - DC-balance is important for certain media as it avoids a charge being built up in the media.



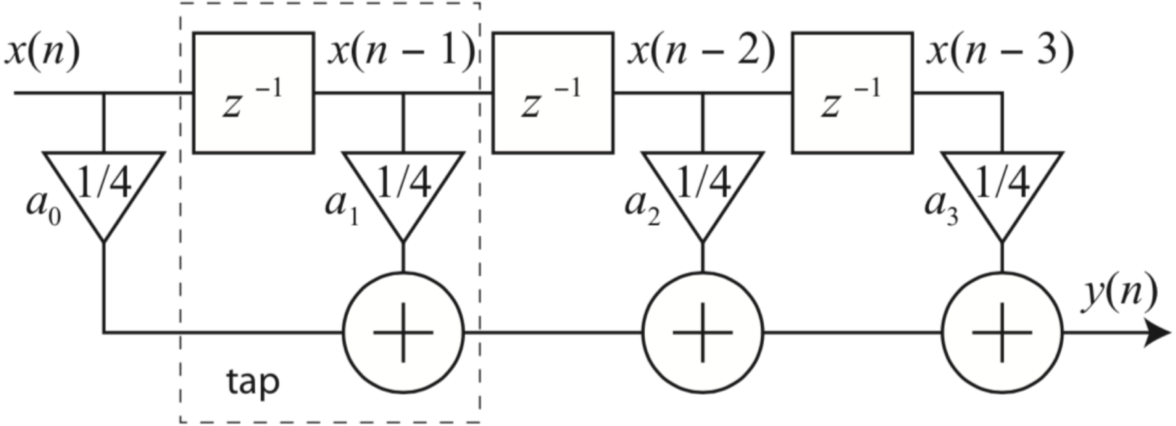

| 3b Binary (HGF) | 4b Binary (fghi) |
|---|---|
| 000 | 0100 or 1011 |
| 001 | 1001 |
| 010 | 0101 |
| 011 | 0011 or 1100 |
| 100 | 0010 or 1101 |
| 101 | 1010 |
| 110 | 0110 |
| 111 | 0001 or 1110 or 1000 or 0111 |

# FIR Filter Implementation

- $z^{-1}$ is unit delay
- Suppose N = 4 and $a_0 = a_1 = a_2 = a_3 = 1/4$.
- Then for all $n \in N$,

$$y(n) = (x(n) + x(n - 1) + x(n - 2) + x(n - 3))/4 .$$

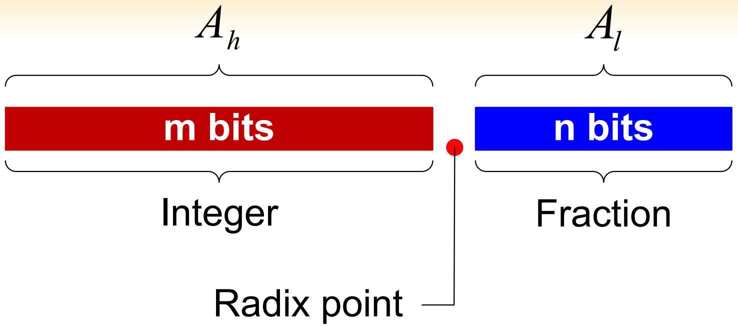
- Multiply-Accumulate



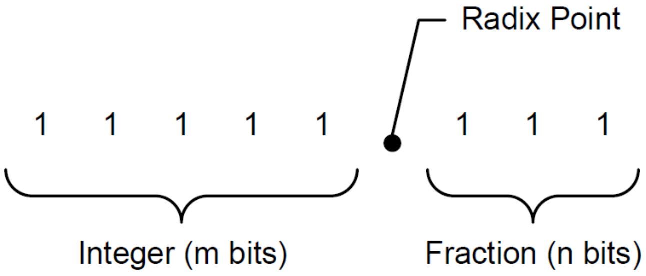

Tapped delay line implementation of the FIR filter

# Fixed Point Numbers

- $\mathbf{01101.101_2}$
- $= 1\times2^3 + 1\times2^2 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-3}$
- $= 13.625$




$$\boldsymbol{f = A_h + A_l\times2^{-n}}$$




$$10101.101_2 = A_h + A_l\times2^{-3}$$

$$= 21 + 5\times2^{-3}$$

$$= 21.625$$

# Notation

Velocity

$$\dot{\mathbf{x}}\colon \mathbb{R} \to \mathbb{R}^3$$

is the derivative, $\forall\, t \in \mathbb{R}$,

$$\dot{\mathbf{x}}(t) = \frac{d}{dt}\mathbf{x}(t)$$

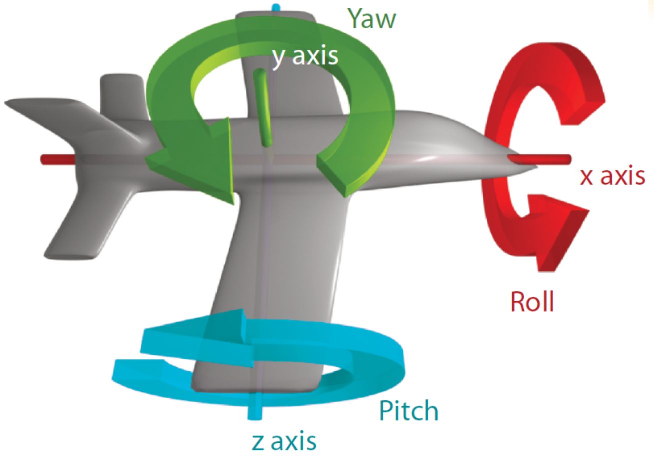
Acceleration $\ddot{\mathbf{x}}\colon \mathbb{R} \to \mathbb{R}^3$ is the second derivative,

$$\ddot{\mathbf{x}} = \frac{d^2}{dt^2}\mathbf{x}$$

Force on an object is $\mathbf{F}\colon \mathbb{R} \to \mathbb{R}^3$.

# Orientation

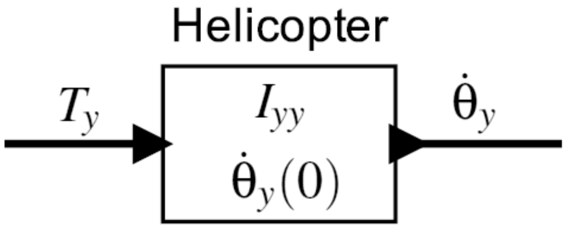
- Orientation: $\theta : \mathbb{R} \rightarrow \mathbb{R}^3$
- Angular velocity: $\dot{\theta} : \mathbb{R} \rightarrow \mathbb{R}^3$
- Angular acceleration: $\ddot{\theta} : \mathbb{R} \rightarrow \mathbb{R}^3$
- Torque: $\mathbf{T} : \mathbb{R} \rightarrow \mathbb{R}^3$




$$\theta(t) = \begin{bmatrix} \dot{\theta}_x(t) \\ \dot{\theta}_y(t) \\ \dot{\theta}_z(t) \end{bmatrix} = \begin{bmatrix} \text{roll} \\ \text{yaw} \\ \text{pitch} \end{bmatrix}$$

# Actor Model of the Helicopter

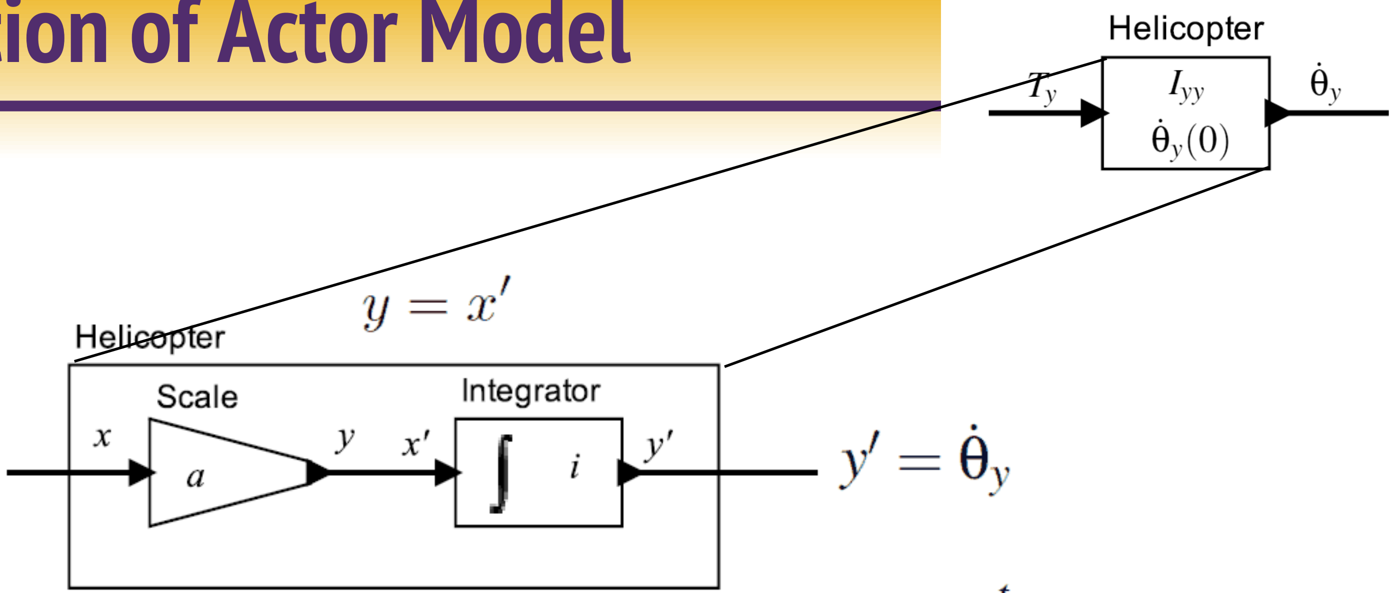
- Input is the net torque of the tail rotor and the top rotor. Output is the angular velocity around the y-axis.




- Parameters of the model are shown in the box. The input and output relation is given by the equation to the right.

$$\dot{\theta}_y(t) = \dot{\theta}_y(0) + \frac{1}{I_{yy}} \int_0^t T_y(\tau) d\tau$$

# Composition of Actor Model




$$y = x'$$

$$x = T_y$$

$$y' = \dot{\theta}_y$$

$$\forall\, t \in \mathbb{R}, \quad y(t) = ax(t)$$

$$y'(t) = i + \int_0^t x'(\tau)d\tau$$

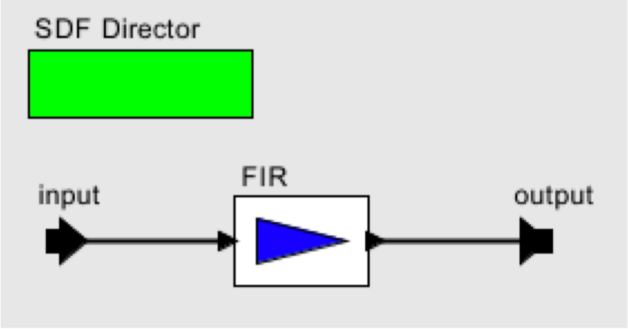
$$y = ax$$

$$i = \dot{\theta}_y(0)$$

$$a = 1/I_{yy}$$

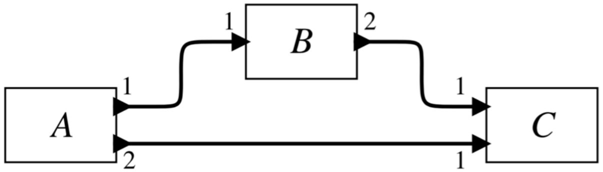
# Synchronous Dataflow (SDF)

- Specialized model for dataflow
- All actors consume input tokens, perform their computation and produce outputs in one atomic operation
- Flow of control is known (predictable at compile time)
- Statically scheduled domain
- Useful for synchronous signal processing systems
- Homogeneous SDF: one token is usually produced for every iteration



# Solving the Balance Equation

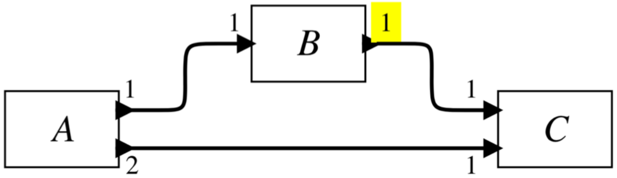
- Every connection between actors results in a balance equation
- The model defines a system of equations, and the goal is to find the least positive integer solution




$$\begin{aligned} q_A &= q_B \\ 2q_B &= q_C \\ 2q_A &= q_C \end{aligned}$$

- The least positive integer solution to these equations is
  - $q_A = q_B = 1$, and $q_C = 2$
- The schedule {A, B, C, C} can be repeated forever to get an unbounded execution with bounded buffers

# Inconsistent SDF



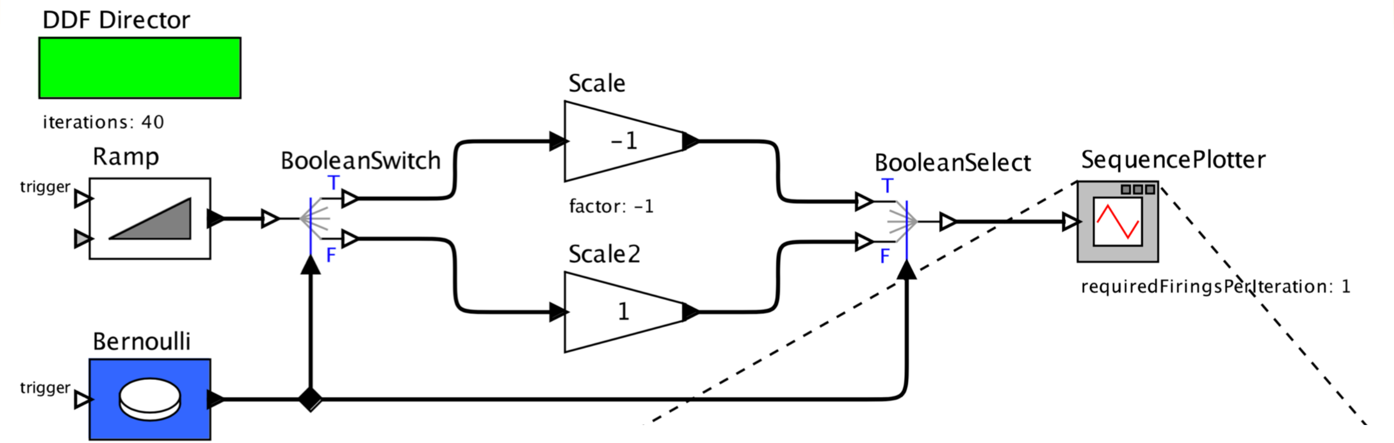

$$q_A = q_B = q_C = 0$$

- An SDF model that has a non-zero solution to the balance equations is said to be consistent.
- If the only solution is zero, then it is inconsistent.
- An inconsistent model has no unbounded execution with bounded buffers.

# Dynamic Dataflow (DDF)

- SDF cannot express conditional firing: an actor fires only if a token has a particular value
- DDF: Firing Rule is required to be satisfied for firing
- Number of tokens produced can vary
- Example DDF Actor: Select
- Similar to Go To in Imperative Programming

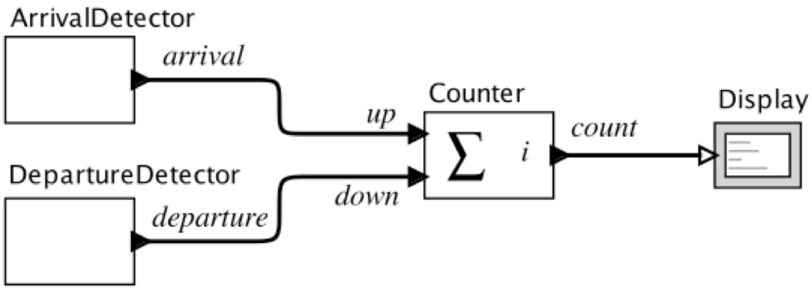
# Example DDF (Conditional Firing)




When Bernoulli produces true, the output of the Ramp actor is multiplied by −1

# Discrete Systems

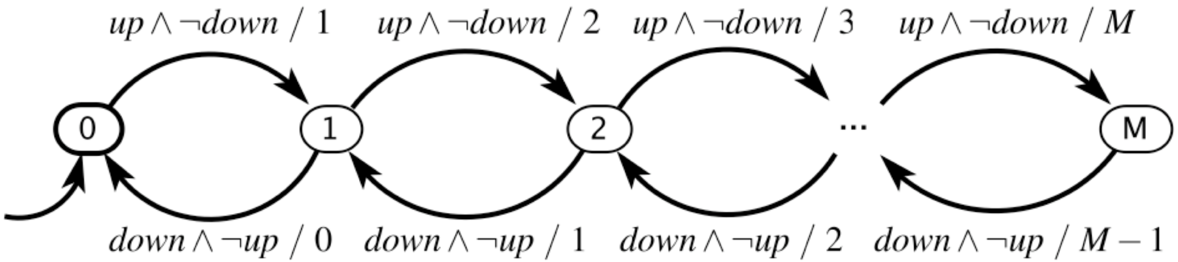
- Example: count the number of cars that enter and leave a parking garage:




- Pure signal: $up\colon \mathbb{R} \to \{absent, present\}$
- Discrete actor: $Counter\colon (\mathbb{R} \to \{absent, present\})^P \to (\mathbb{R} \to \{absent\} \cup \mathbb{N})$

$P = \{up, down\}$

# Garage Counter Mathematical Model




Formally: $(States, Inputs, Outputs, update, initialState)$, where

- $States = \{0, 1, \cdots, M\}$
- $Inputs = (\{\text{up}, \text{down}\} \to \{absent, present\}$
- $Outputs = (\{count\} \to \{absent\} \cup \mathbb{N})$
- $update : States \times Inputs \to States \times Outputs$
- $initialState = 0$

Transition Function

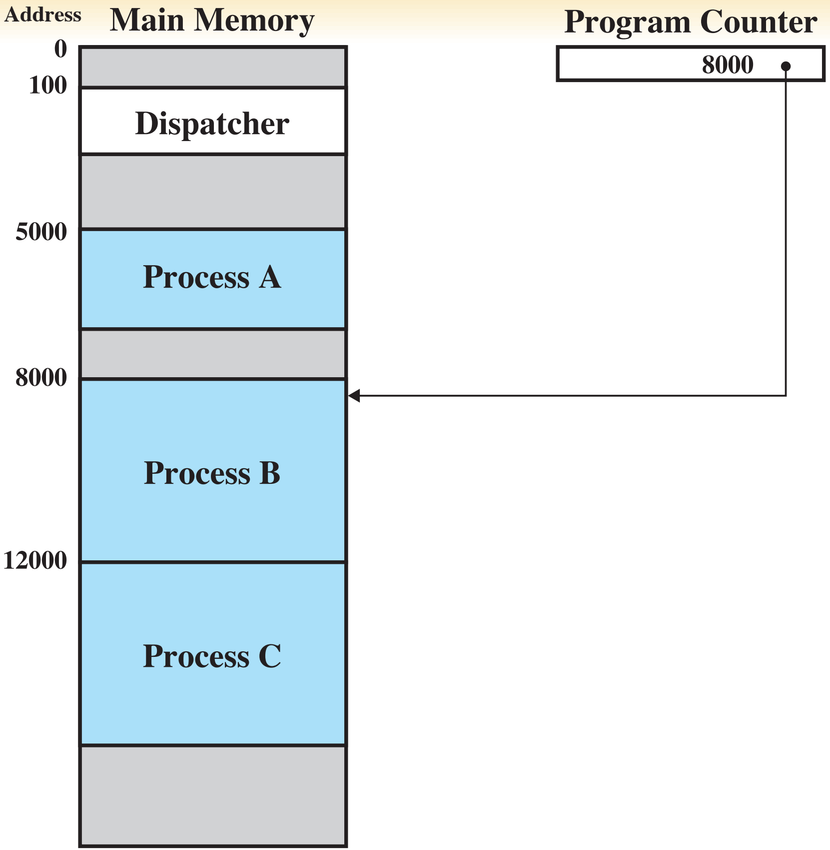
$$(s(n+1), y(n)) = update(s(n), x(n))$$

The update function is given by

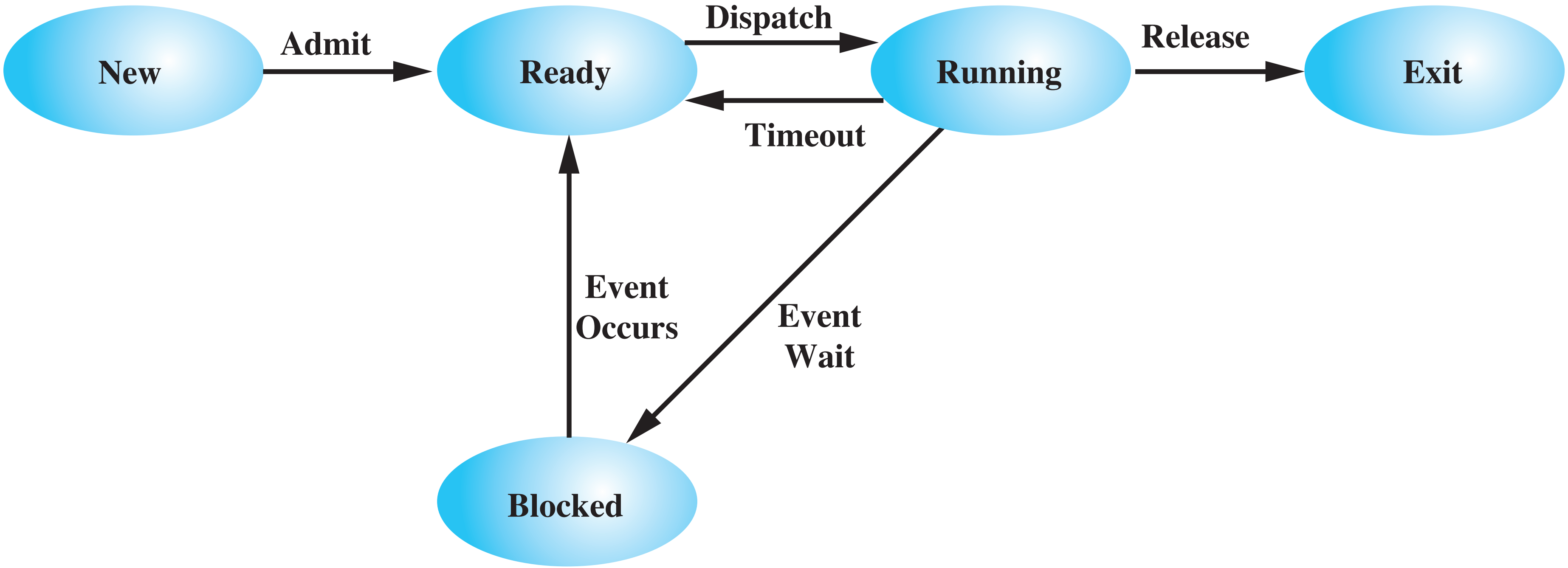
$$update(s,i) = \begin{cases} (s+1, s+1) & \text{if } s < M \\ & \wedge\, i(up) = present \\ & \wedge\, i(down) = absent \\ (s-1, s-1) & \text{if } s > 0 \\ & \wedge\, i(up) = absent \\ & \wedge\, i(down) = present \\ (s, absent) & \text{otherwise} \end{cases}$$

# Process Execution

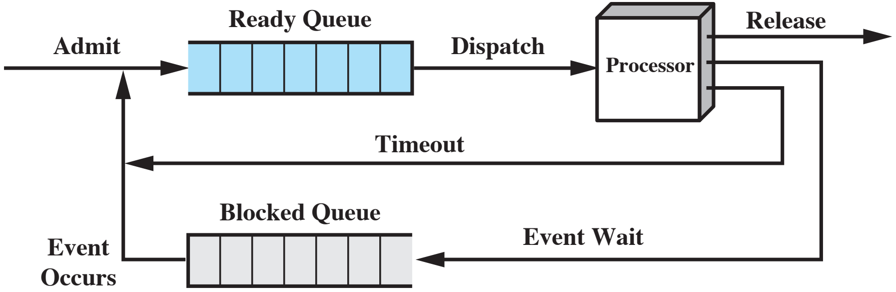
- Memory layout of three processes
- Dispatcher program: switches processor from one process to another

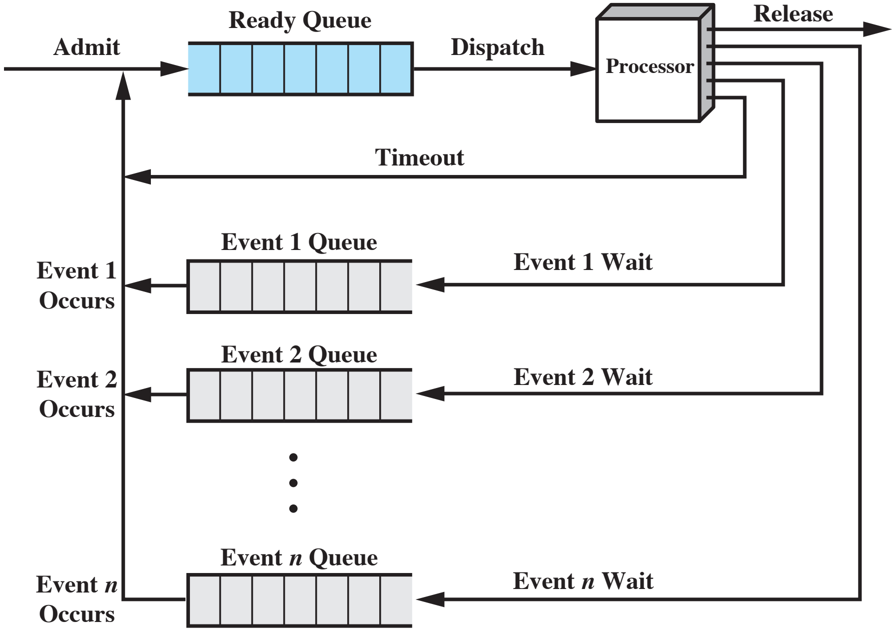


# Five State Process Model



# Queuing Model

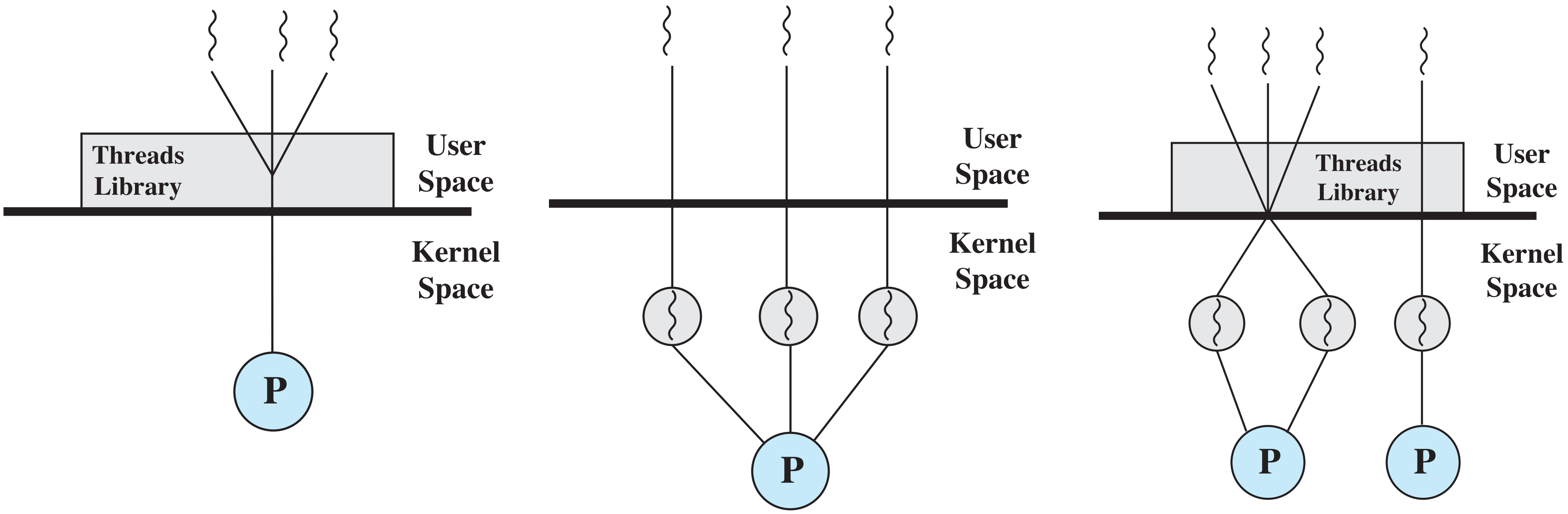



Single Blocked Queue

Multiple Blocked Queue

# User and Kernel level Threads




(a) Pure user-level

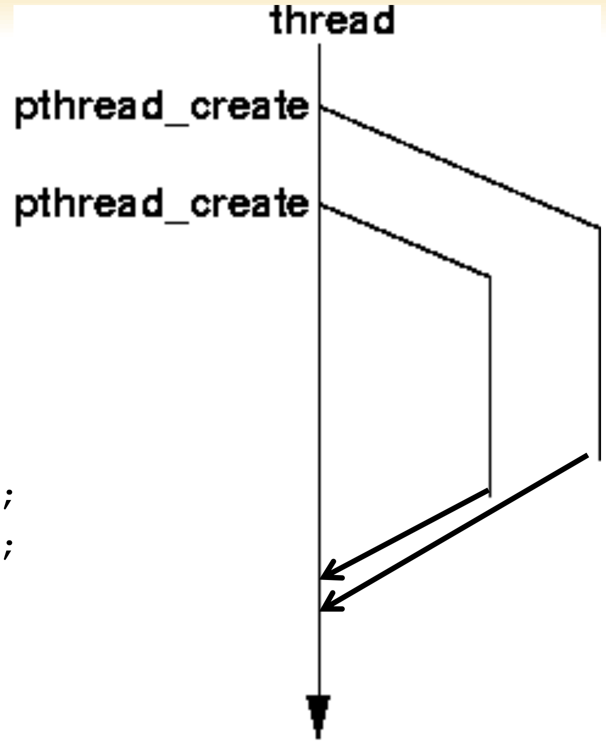
(b) Pure kernel-level

(c) Combined

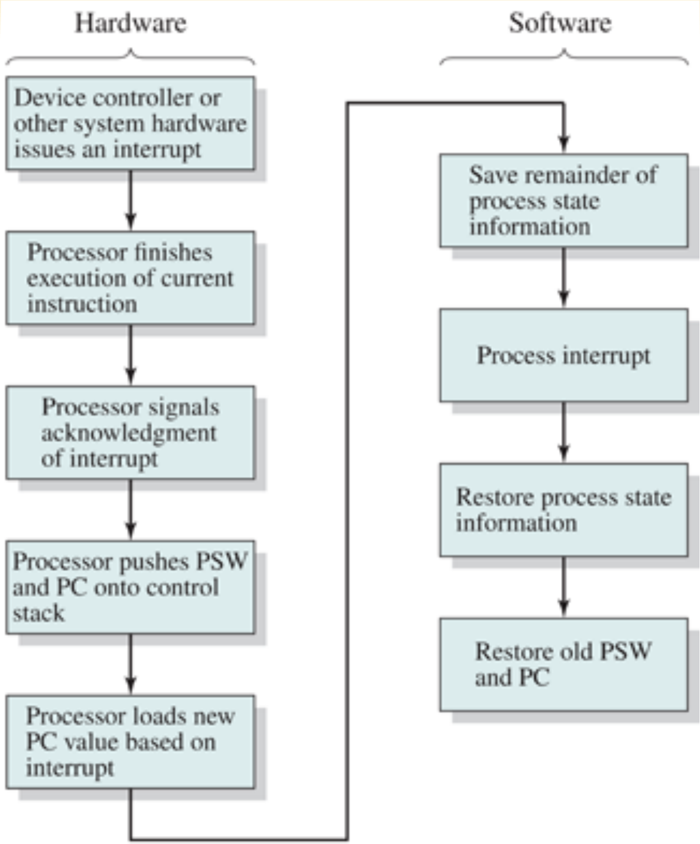
# Create Thread and Join

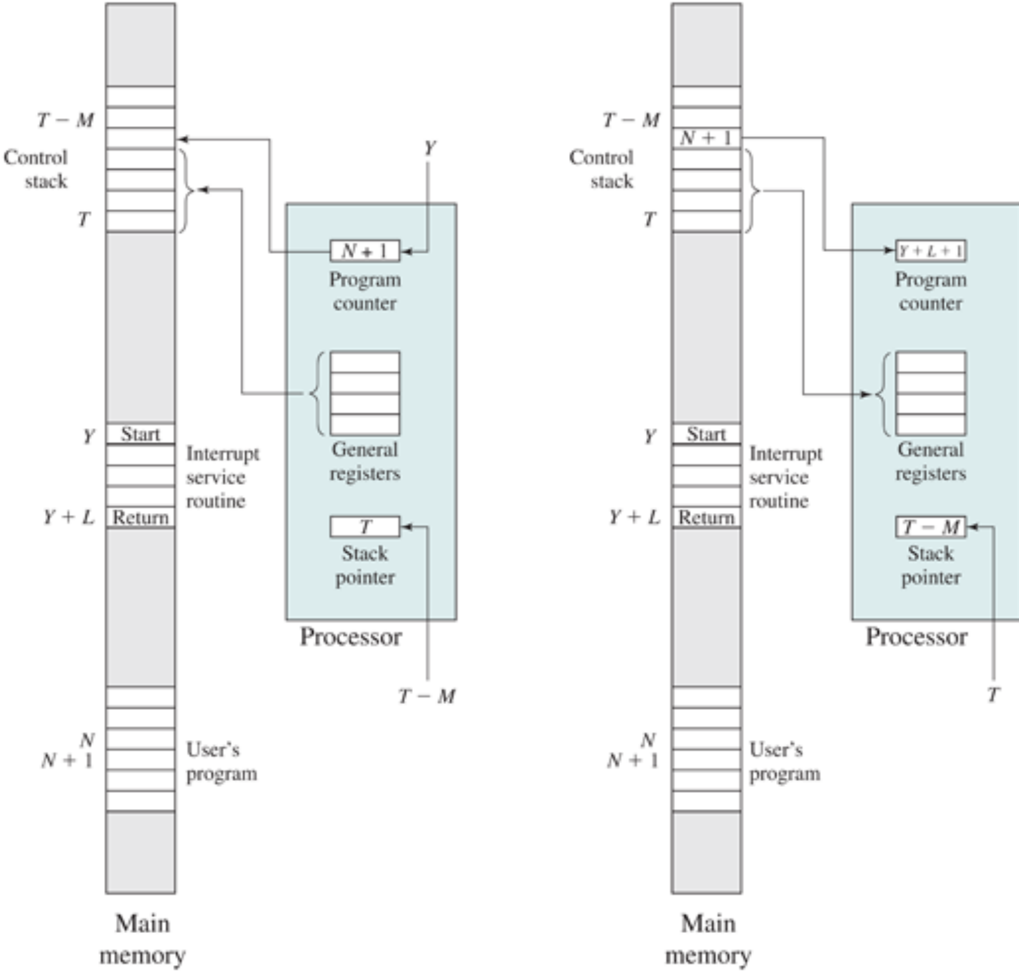
```
#include <pthread.h>
#include <stdio.h>
void *PrintHello(void * id){
   printf("Hello from thread %d\n", id);
}

void main (){
   pthread_t thread0, thread1;
   pthread_create(&thread0, NULL, PrintHello, (void *) 0);
   pthread_create(&thread1, NULL, PrintHello, (void *) 1);
   pthread_join(thread0, NULL);
   pthread_join(thread1, NULL);
}
```



# Interrupt Processing



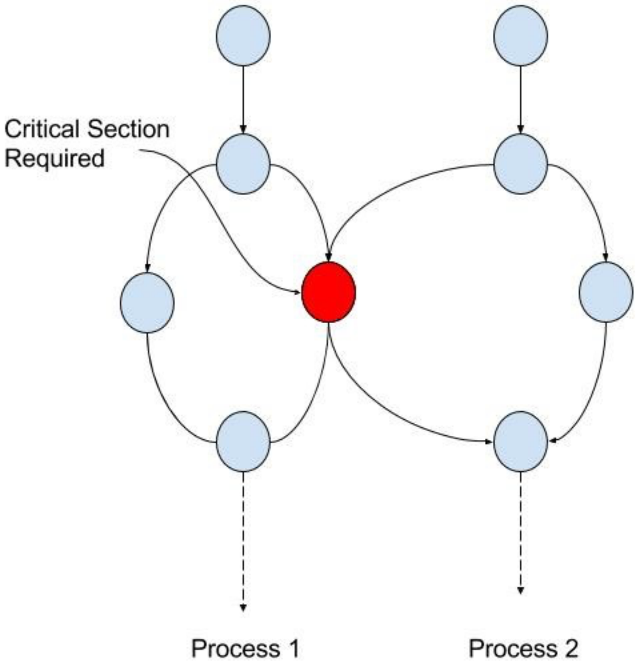




(a) Interrupt occurs after instruction at location N

(b) Return from interrupt

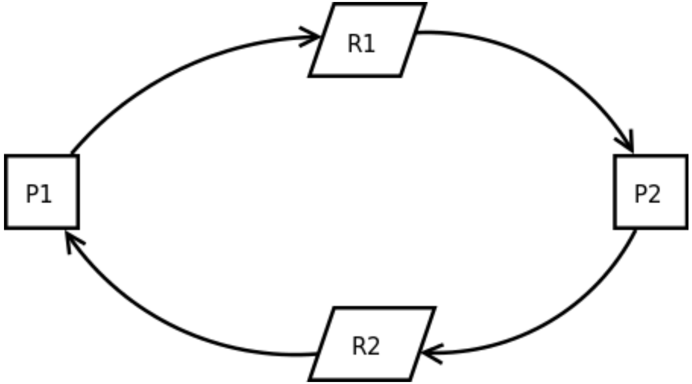
# Mutual Exclusion - Mutex

- Prevents Race Condition
- Enables resource sharing
- Critical section is performed by a single process or thread
- One thread blocks a critical section by using locking technique (mutex)
- Other threads have to wait to get their turn to enter into the section.



# Deadlock

- The permanent blocking of a set of processes that either compete for system resources or communicate with each other
- A set of processes is deadlocked when each process in the set is blocked awaiting an event that can only be triggered by another blocked process in the set

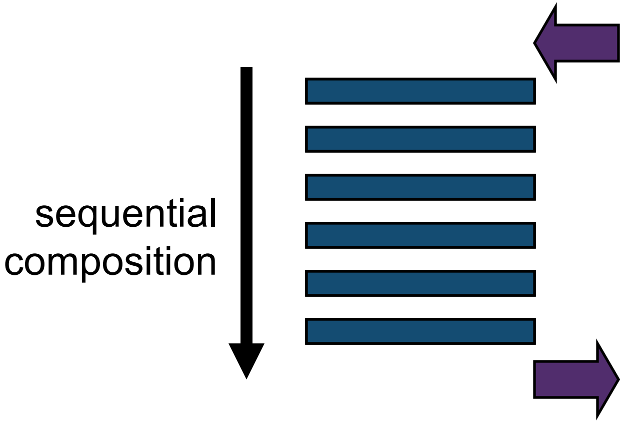



- Example: addListener() and update()

# Problems with the Foundations of Threads

- A model of computation:
  - Bits: B = {0, 1}
  - Set of finite sequences of bits: B*
  - Computation: f : B* → B*
  - Composition of computations: f • f '
  - Programs specify compositions of computations
- Threads augment this model to admit concurrency.
- But this model does not admit concurrency gracefully.

# Basic Sequential Computation



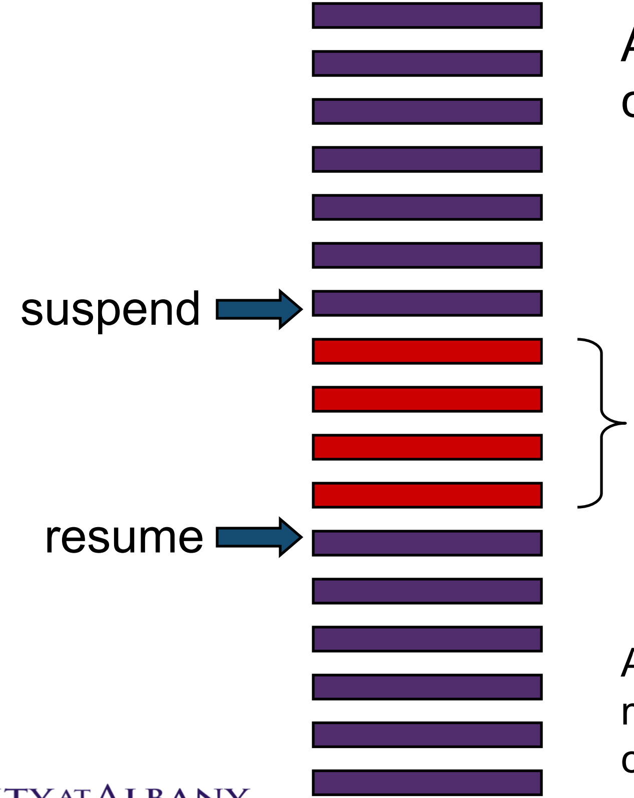

initial state: $b_0 \in B^*$

$$b_n = f_n(b_{n-1})$$

final state: $b_N$

*Formally, composition of computations is function composition.*

# When There are Threads, Everything Changes




A program no longer computes a function.

$$b_n = f_n ( b_{n-1} )$$

another thread can change the state

$$b'_n = f_n ( b'_{n-1} )$$

Apparently, programmers find this model appealing because nothing has changed in the *syntax*.

# Scheduling Policies

- First Come First Serve
- Round Robin
- Shortest Process Next
- Shortest Remaining Time Next
- Highest Response Ratio Next
- Feedback Scheduler
- Fair Share Scheduler

# How to predict execution time in SPN ?

$$S_{n+1} = \frac{1}{n}\sum_{i=1}^{n} T_i$$

$T_i$ =processor execution time for the *i*th instance of this process (total execution time for batch job; processor burst time for interactive job),

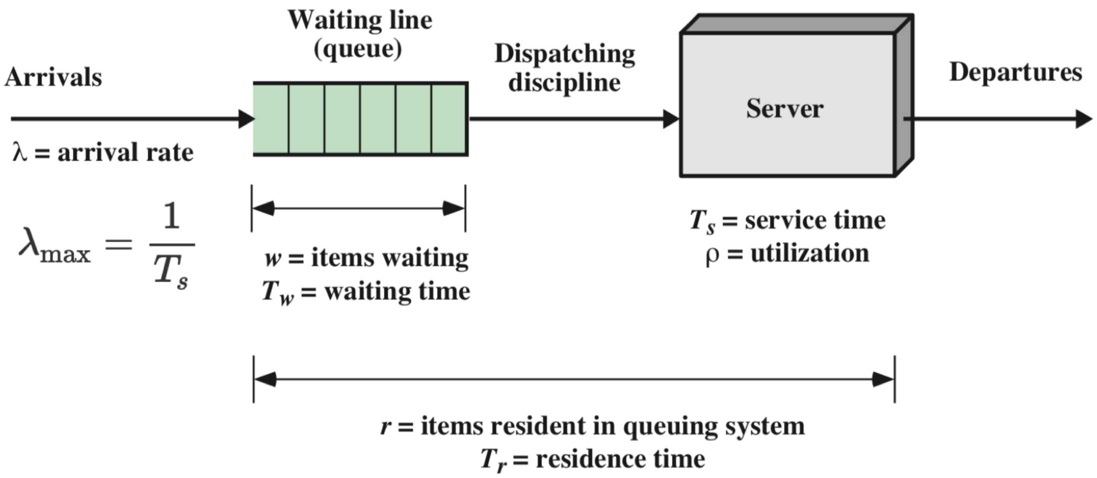
$S_i$ =predicted value for the *i*th instance, and

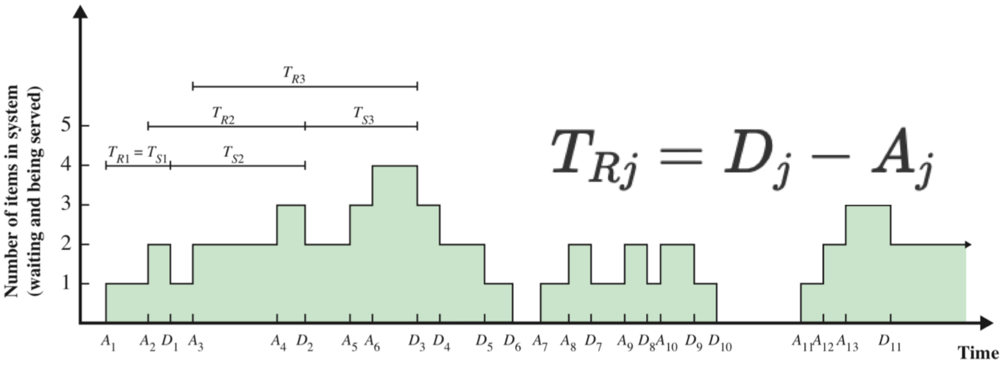
$S_1$ =predicted value for first instance; not calculated.

- Store the Sum $S_{n+1} = \frac{1}{n} T_n + \frac{n-1}{n} S_n$
- Higher weight to recent instances $S_{n+1} = \alpha T_n + (1-\alpha) S_n$
- The older the observation, the less it is counted in to the average.

$$S_{n+1} = \alpha T_n + (1-\alpha)\alpha T_{n-1} + \ldots + (1-\alpha)^i \alpha T_{n-i} + \ldots + (1-\alpha)^n S_1$$

# Queuing Analysis




$$T_{Rj} = D_j - A_j$$

$$T_{Rn+1} = T_{Sn+1} + \mathrm{MAX}\left[0,\ D_n - A_{n+1}\right]$$

For item *i*:

$A_i$ = Arrival time
$D_i$ = Departure time
$T_{Ri}$ = Residence time
$T_{Si}$ = Service time

UNIVERSITY AT ALBANY
State University of New York

# Characteristics of Various Scheduling Policies

| | FCFS | Round Robin | SPN | SRT | HRRN | Feedback |
|---|---|---|---|---|---|---|
| **Selection Function** | max[$w$] | constant | min[$s$] | $\min[s-e]$ | $\max\left(\frac{w+s}{s}\right)$ | (see text) |
| **Decision Mode** | Non-preemptive | Preemptive (at time quantum) | Non-preemptive | Preemptive (at arrival) | Non-preemptive | Preemptive (at time quantum) |
| **Throughput** | Not emphasized | May be low if quantum is too small | High | High | High | Not emphasized |
| **Response Time** | May be high, especially if there is a large variance in process execution times | Provides good response time for short processes | Provides good response time for short processes | Provides good response time | Provides good response time | Not emphasized |
| **Overhead** | Minimum | Minimum | Can be high | Can be high | Can be high | Can be high |
| **Effect on Processes** | Penalizes short processes; penalizes I/O-bound processes | Fair treatment | Penalizes long processes | Penalizes long processes | Good balance | May favor I/O-bound processes |
| **Starvation** | No | No | Possible | Possible | No | Possible |

# Characteristics of Real Time Systems

Real-time operating systems have requirements in five general areas:

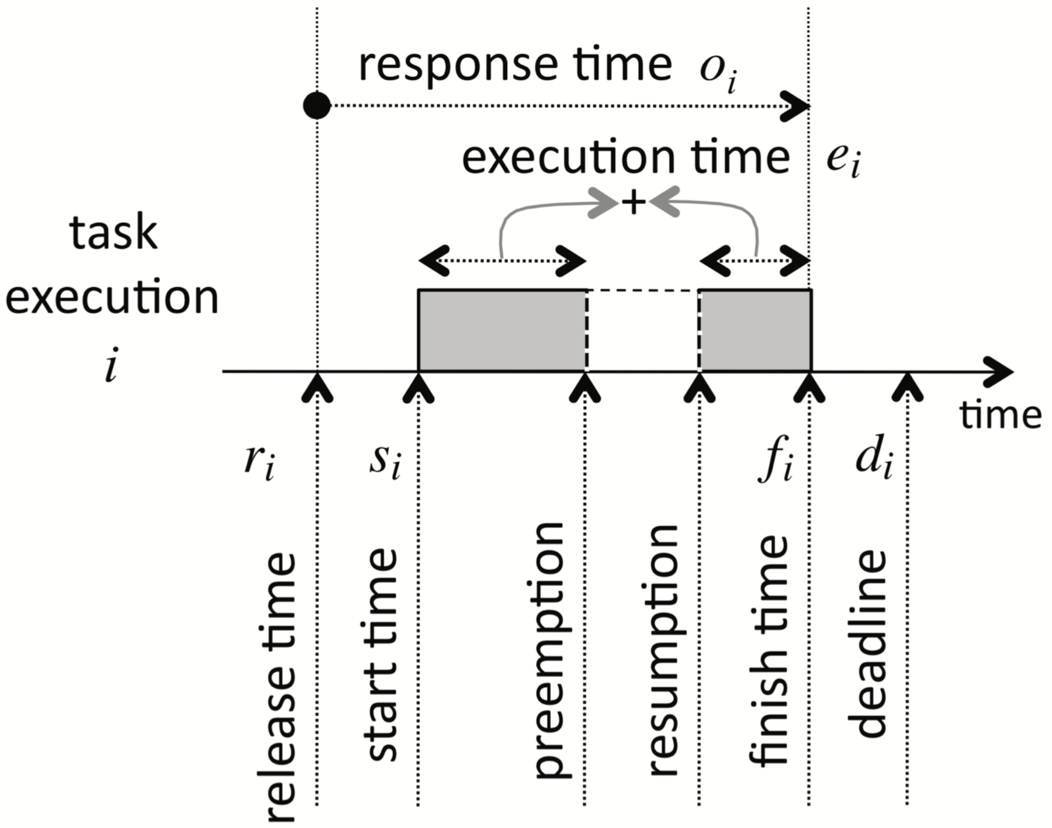
- Determinism
- Responsiveness
- User control
- Reliability
- Fail-soft operation

# Task Model

$$s_i \geq r_i$$

$$f_i \geq s_i$$

$$o_i = f_i - r_i$$



# Criteria or Metrices

- Processor Utilization $\mu$
- Maximum Lateness

$$L_{\max} = \max_{i \in T}(f_i - d_i)$$

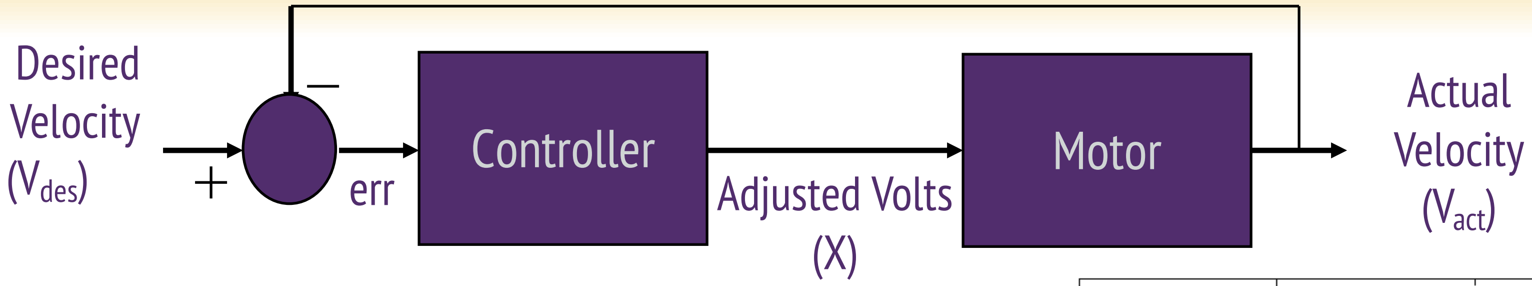
- Total Completion Time or Makespan

$$M = \max_{i \in T} f_i - \min_{i \in T} r_i$$

- Average Response Time

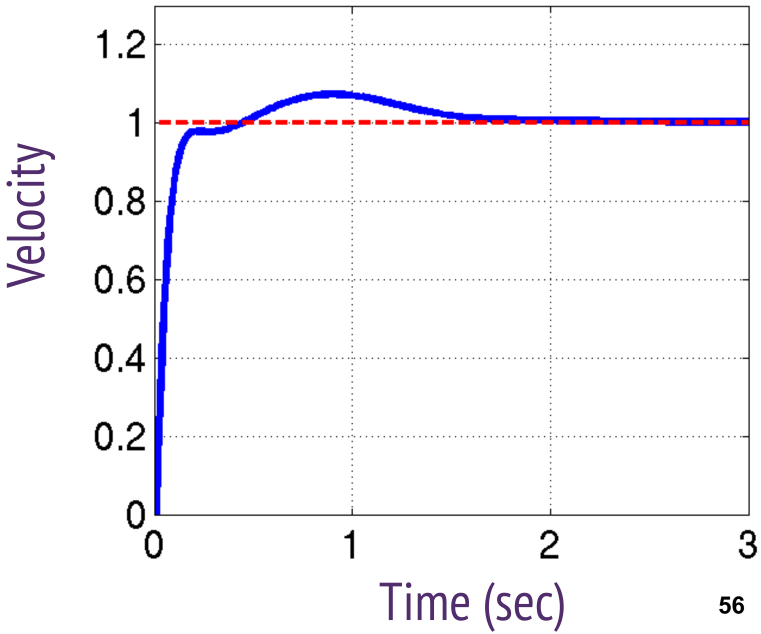
$$\overline{t_r} = \frac{1}{n}\sum_{i=1}^{n}(f_i - a_i)$$

# Step Response with PID Controller




- Combined benefits of PI and PD

$$X = V_{des} + K_P e(t) + K_I \int e(t)\ dt - K_D \frac{de(t)}{dt}$$




UNIVERSITY AT ALBANY
State University of New York

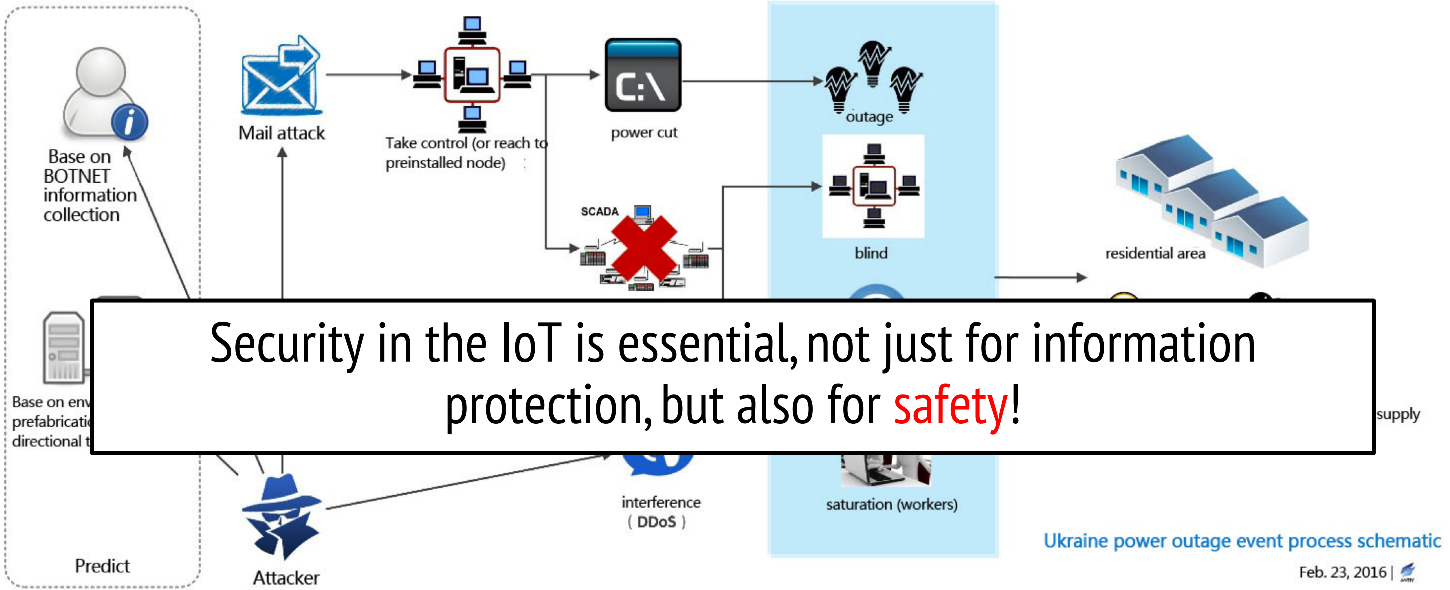
# PID Controller Pseudocode

```
% Precompute controller coefficients
bi=ki*h
ad=Tf/(Tf+h)
bd=kd/(Tf+h)
br=h/Tt

% Control algorithm - main loop
while (running) {
  r=adin(ch1)                  % read setpoint from ch1
  y=adin(ch2)                  % read process variable from ch2
  P=kp*(b*r-y)                 % compute proportional part
  D=ad*D-bd*(y-yold)           % update derivative part
  v=P+I+D                      % compute temporary output
  u=sat(v,ulow,uhigh)          % simulate actuator saturation
  daout(ch1)                   % set analog output ch1
  I=I+bi*(r-y)+br*(u-v)        % update integral
  yold=y                       % update old process output
  sleep(h)                     % wait until next update interval
}
```

# Security Threats in the IoT

- Cyber attack on the Ukrainian power grid
- Power outage caused by hackers




Security in the IoT is essential, not just for information protection, but also for safety!

Source: Comprehensive Analysis Report on Ukraine Power System Attacks
March 16, 2016 By Antiy Lab

# Properties and Threat Models

- Secrecy/Confidentiality
  - Can secret data be leaked to an attacker?
- Integrity
  - Can the system be modified by the attacker?
- Authenticity
  - Who is the system communicating/interacting with?
- Availability
  - Is the system always able to perform its function?
- Need to think about Threat (attacker) Models

# Polyalphabetic Cipher

| | |
|---|---|
| Plaintext letter: | a b c d e f g h i j k l m n o p q r s t u v w x y z |
| $C_1(k = 5)$: | f g h i j k l m n o p q r s t u v w x y z a b c d e |
| $C_2(k = 19)$: | t u v w x y z a b c d e f g h i j k l m n o p q r s |

- n substitution ciphers, $C_1, C_2, \ldots, C_n$
- cycling pattern:
  - e.g., n=4 [$C_1$-$C_4$], k=key length=5: $C_1,C_3,C_4,C_3,C_2$; $C_1,C_3,C_4,C_3,C_2$; ..
- for each new plaintext symbol, use subsequent substitution pattern in cyclic pattern
  - dog: d from $C_1$, o from $C_3$, g from $C_4$

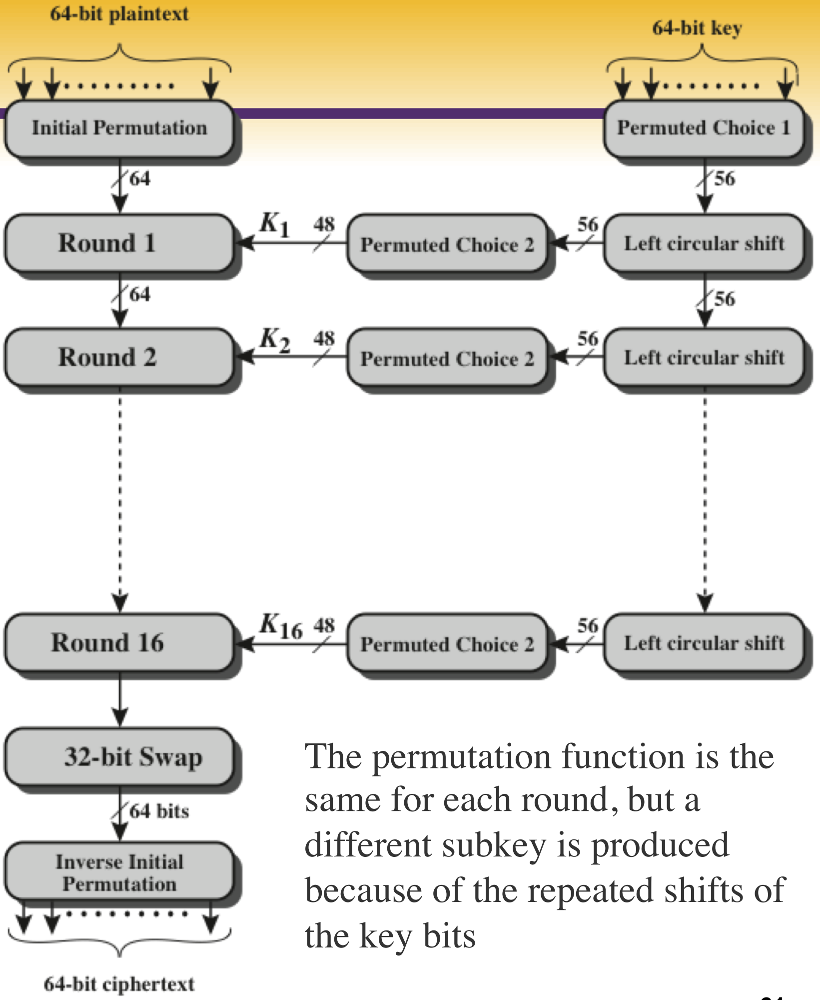
  *Encryption key:* n substitution ciphers, and cyclic pattern
  - key need not be just n-bit pattern

# Symmetric key crypto: DES

- initial permutation (on 64 bits)
- 16 identical "rounds" of function application
  - each using different 48 bits of key
  - a subkey ($K_i$) is produced by the combination of a left circular shift and a permutation
  - rightmost 32 bits are moved to leftmost 32 bits
- final permutation (on 64 bits)

Kaufman, Schneier, 1995

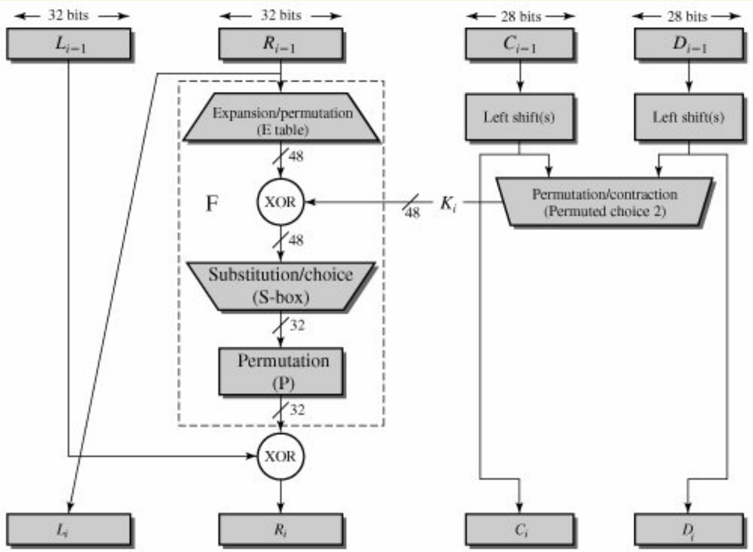
With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher




The permutation function is the same for each round, but a different subkey is produced because of the repeated shifts of the key bits

# Each round of DES

- $K_i$ is 48 bits, R input is 32 bits.
- R is first expanded to 48 bits
  - a table defines a permutation plus an expansion that involves duplication of 16 of the R bits
- Resulting 48 bits are XORed with Ki
- This 48-bit result passes through a substitution function (S box) that produces a 32-bit output
- This is permuted




$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \times F(R_{i-1}, K_i)$$

# RSA: Creating public/private key pair

1. choose two large prime numbers $p, q$. (e.g., 1024 bits each)

2. compute $n = pq$, $z = (p-1)(q-1)$

3. choose $e$ (with $e<n$) that has no common factors with z ($e, z$ are "relatively prime").

4. choose $d$ such that $ed-1$ is exactly divisible by $z$. (in other words: $ed \bmod z = 1$ ).

5. *public* key is $\underbrace{(n,e)}_{K_B^+}$. *private* key is $\underbrace{(n,d)}_{K_B^-}$.

# RSA: encryption, decryption

0. given $(n,e)$ and $(n,d)$ as computed above

1. to encrypt message $m$ $(<n)$, compute

   $c = m^e \bmod n$

2. to decrypt received bit pattern, $c$, compute

   $m = c^d \bmod n$

$$m = (\underbrace{m^e \bmod n}_{c})^d \bmod n$$

# RSA example:

Bob chooses *p=5, q=7*. Then *n=35, z=24*.

*e=5* (so *e, z* relatively prime).

*d=29* (so *ed-1* exactly divisible by z).

encrypting 8-bit messages.

**encrypt:**

| bit pattern | $m$ | $m^e$ | $c = m^e \bmod n$ |
|---|---|---|---|
| 00001000 | 12 | 24832 | 17 |

**decrypt:**

| $c$ | $c^d$ | $m = c^d \bmod n$ |
|---|---|---|
| 17 | 481968572106750915091411825223071697 | 12 |