ECE 445L – Embedded System Design Lab

Quiz 1 review

Spring 2025

Quiz 1

- Quiz will be in person. Closed book, closed notes.
- You may have no electronic devices
- You may bring a double-sided 8.5” x 11.0” crib-sheet
  - Handwritten
  - Do not print any software (it will not help)
- **You will NOT need a calculator**.

Quiz 1 Topics (covers Weeks 1 – 4)

- Key concepts
- Some True/False (read carefully)
- Fixed Point arithmetic
- ARM Cortex-M4 architecture
- SW & HW debugging
- Interrupts, real time, jitter, NVICs
- FIFO analysis, CPU bound, I/O bound, Little's Thm
- Critical sections
- Sampling, PMF, Nyquist, CLI
- Data Acquisition Systems
- LED, switch, buzzer, MOSFET, capacitor, inductor
- OSI, IP, DNS, TCP, UDP, sockets, MQTT

HW stuff from previous courses that you should already know

- **Resistor**
  - What is the equation?
  - When will it explode?
- **Capacitor**
  - What is the equation?
  - DC versus AC response?
- **Inductor**
  - What is the equation?
  - DC versus AC response?
- **Diode, BJT, MOSFET**
  - I/V curves
- **Power**
  - Voltage $\times$ Current
  - $\text{Curent}^2 \times$ Resistance
  - Voltage^2 / Resistance
- **Energy**
  - Power $\times$ Time
- **Time**
  - Resistance $\times$ Capacitance
- **Battery Capacity**
  - mA-Hour
  - What are the limitations?

Number Representations

- Integers
  - Fixed-width (8, 16, 32, 64) integer number
- Reals (rational $\Rightarrow \frac{\textbf{Integer}}{\textbf{Integer}} \Rightarrow \frac{\mathbf{23}}{\mathbf{4}} = 5.75$)
  - Fixed-point number $\equiv$ **I** • Δ
    - Store **I**, but Δ is fixed
    - Decimal fixed-point (**Δ=**10^m) = **I** • 10^m
    - Binary fixed-point (**Δ=**2^m) = **I** • 2^m
  - Floating-point number = **I** • B^E
    - Store both **I** and **E** (only B=2 is fixed)

What is 22/7?
What is 24/17?

Fixed-point numbers

- **Why:**
  express values with non-integer values
  no floating-point hardware support

- **When:**
  range of values is known
  range of values is small

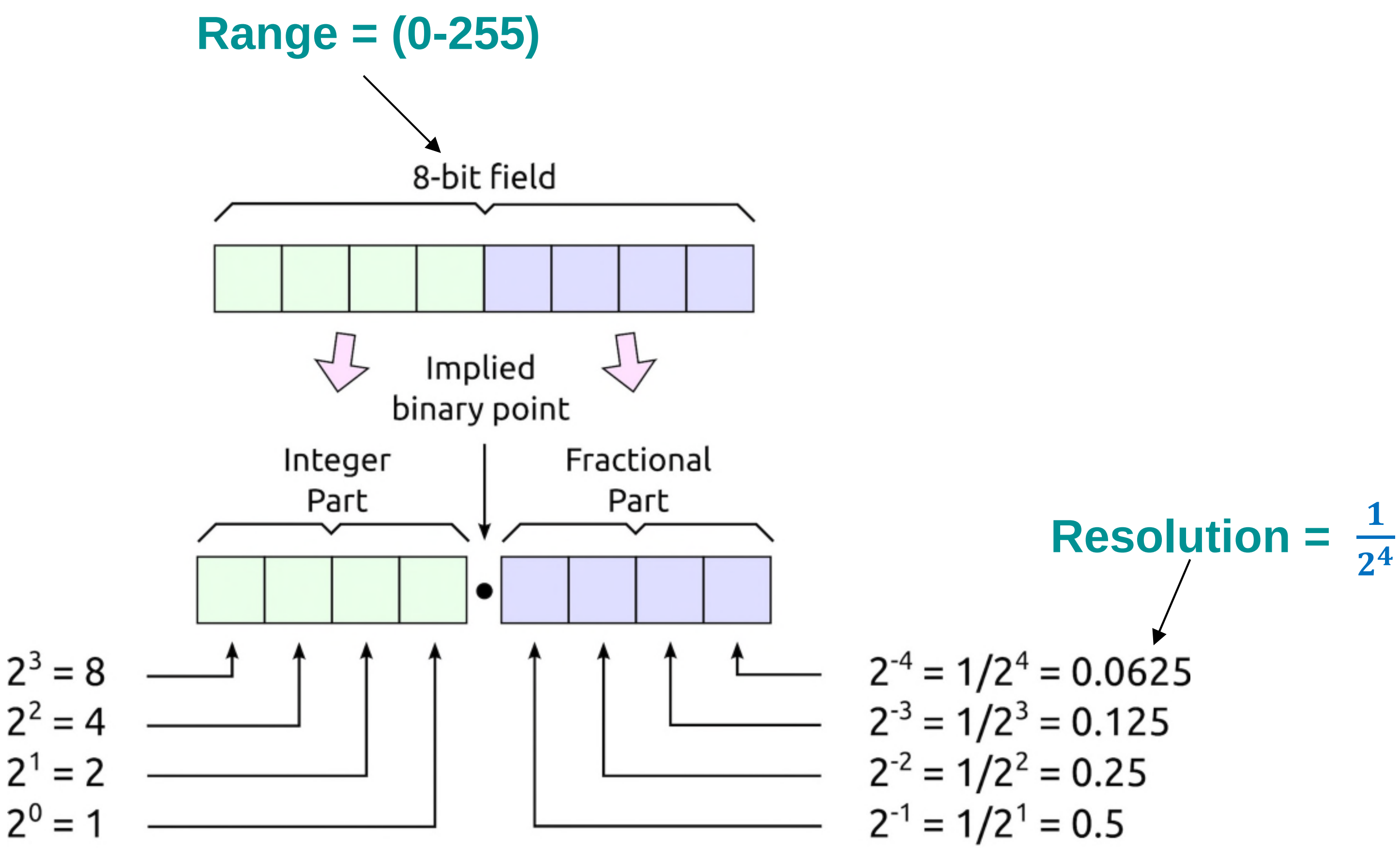
- **How:**
  1) Variable integer, called I.
     - may be signed or unsigned
     - may be 8, 12, 16, 24 or 32 bits (range/precision)

  2) Fixed constant, called Δ (resolution)
     - value is fixed, and cannot be changed
     - not stored in memory
     - specify this fixed constant using comments

Fixed-point numbers

- **How to design a fixed-point number?**
  - Integer can be signed or unsigned
  - **Range** is the number of distinguishable values that can be represented.
    - determined by the number of bits used to store the variable integer, *e.g.*, 8-bits, 12-bits 16-bits, 24-bits or 32-bits
  - **Resolution** is the smallest difference in value that can be represented.
    - equal to the fixed constant (Δ).
    - defines units

Fixed-point numbers



8-bit binary 4.4 fixed-point representation

Example: Fixed-point numbers

- **Create a voltmeter**
  - N = ADC output, 0 to 4095
  - V_{in} = 3.3V * N/4096 = 0.000805664 * N
  - Let Δ = 0.001V, V_{in} = I * 0.001V
  - Solve I in terms of N
  - I = 0.805664 * N

- **Representations of 0.805664**
  - I = (805664*N)/100000
  - I = (3300*N)>>12

- **Calibration coefficients *A B***
  - I = *A*+ (*B**N)>>12

Example: Fixed-point numbers

- **Temperature Measurement System**
  - Analog to digital converter (ADC)
  - 12-bit range (4096 alternatives)
    - digital output varies 0 to 4095.
  - analog input range is 0 to +3.3 V,
  - resolution = range/precision less than 1 mV

- **Measurement System**
  - range is 10℃ to 40℃
  - 12-bit range
  - resolution less than 0.01℃

Example: Fixed-point numbers

- **Voltage representation**
  - o V_{in} = 3.3 * N/4096 = 0.000805664 * N

- **Temperature representation**
  - o let T = 10 + (30℃ * Vin/3.3V) = 10 + 9.09(℃ /V) * V_{in}
  - o T = I * 0.01℃
  - o then I = 1000+3000*N/4096 where Δ is 0.01℃

- **Let *A* and *B* be calibration coefficients**
  - o I = *A*+(*B**N)>>12

Fixed-point numbers

- Issues (Δ=0.01℃)
  - **overflow**

    `I = 1000+(3000*N)/4096;`

    reduce integer size

    promote to higher range/precision

    `I = 1000+(3000*(uint32_t)N)/4096`

  - **dropout (underflow)**

    `I = 1000+3000*(N/4096);`

Fixed-point numbers

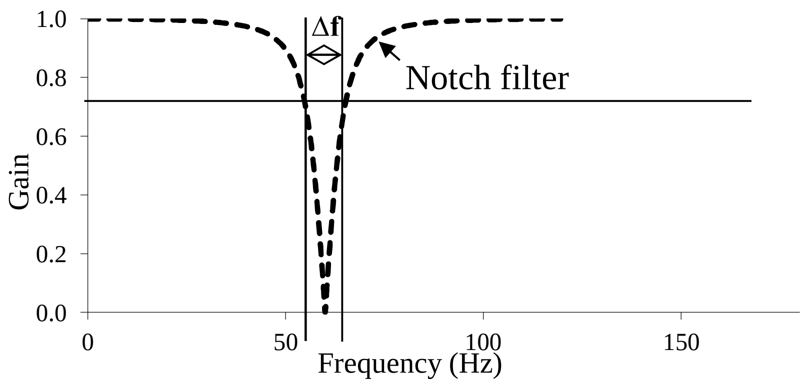
- C optimization
  - `I = 1000+(3000*N)/4096;`
  - `I = 1000+(3000*N)>>12;`
- Rounding
  - `I = 1000+(3000*N + 2048)>>12;`
- Assembly Optimization

```
Convert
  MOV r1,#3000
  MUL r0,r1,r0
  ADD r0,r0,#2048
  MOV r1,#1000
  ADD r0,r1,r0,LSR #12
  BX  lr
```

```
uint32_t Convert(uint32_t adc){
 return 1000+(3000*adc+2048)/4096;
}
```

What does this do?
MLA R1, R2, R3, R4

Example: Digital Notch Filter



f_s=480Hz, f_c=60Hz,
Q=f_c/Δf = 6

- Consider this digital filter calculation:

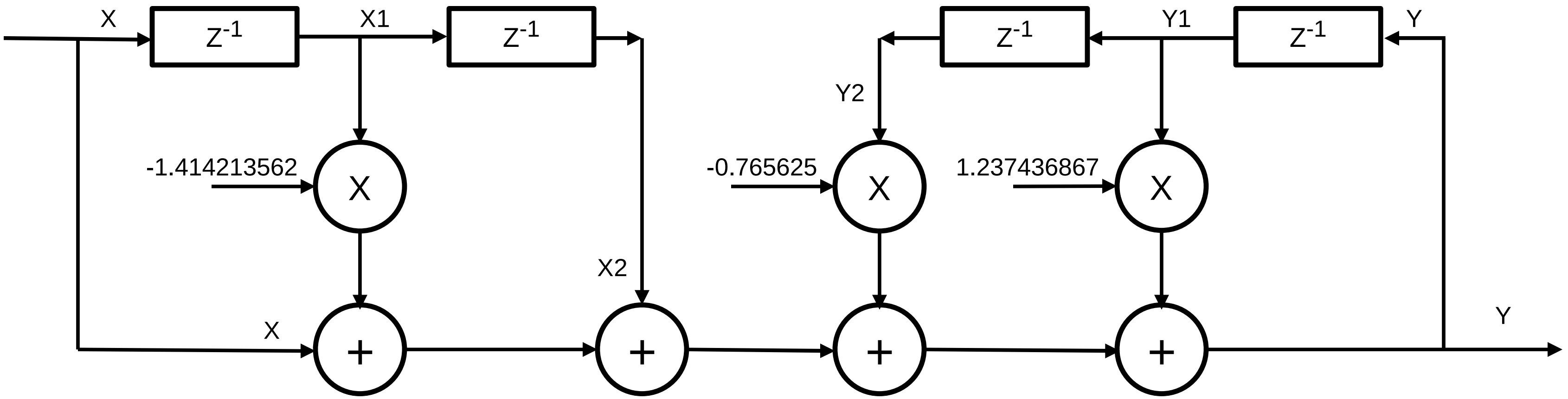
```
y = x - 1.414213562*x1 + x2
    + 1.237436867*y1 - 0.765625*y2;
```

- A fixed-point implementation is:

```
y = x + x2
   +( -724*x1 +634*y1 -392*y2)/512;
```

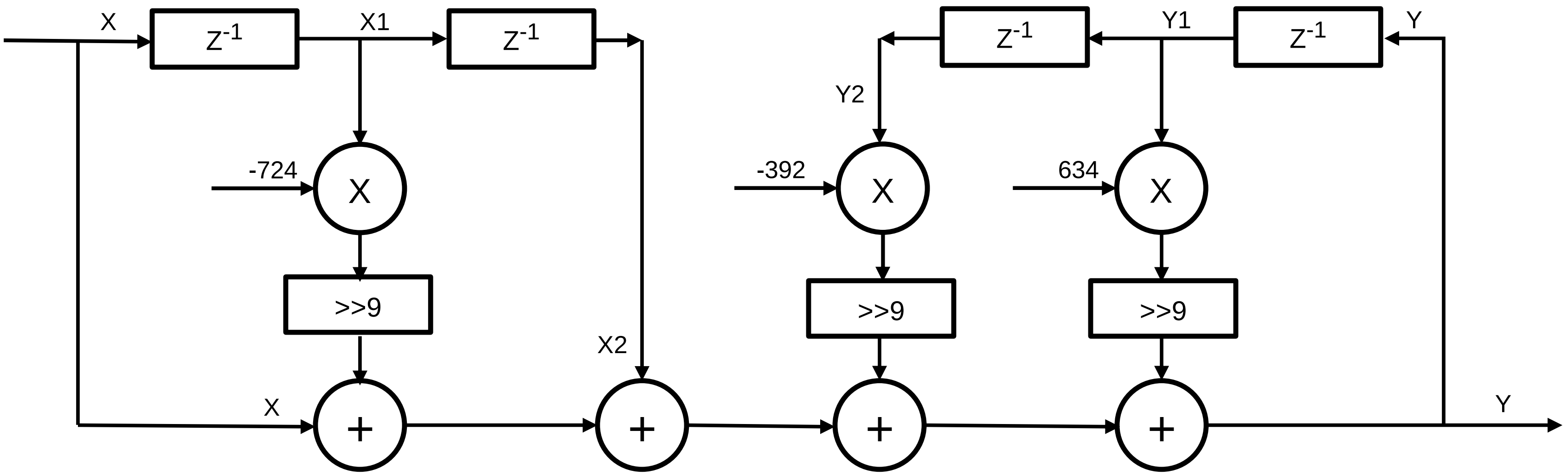
Floating point solution →

```
y = x - 1.414213562*x1 + x2
    + 1.237436867*y1 - 0.765625*y2;
```



Fixed point solution →

```
y = x + x2
    +( -724*x1 +634*y1 -392*y2)/512;
```



Fixed-Point Numbers

- Fixed-point numbers are generally stored in “In.Qm” format (sometimes referred as Qn.m format)

- n = number of bits in integer part.
- m = number of bits in fractional part.

- Example: I8.Q16

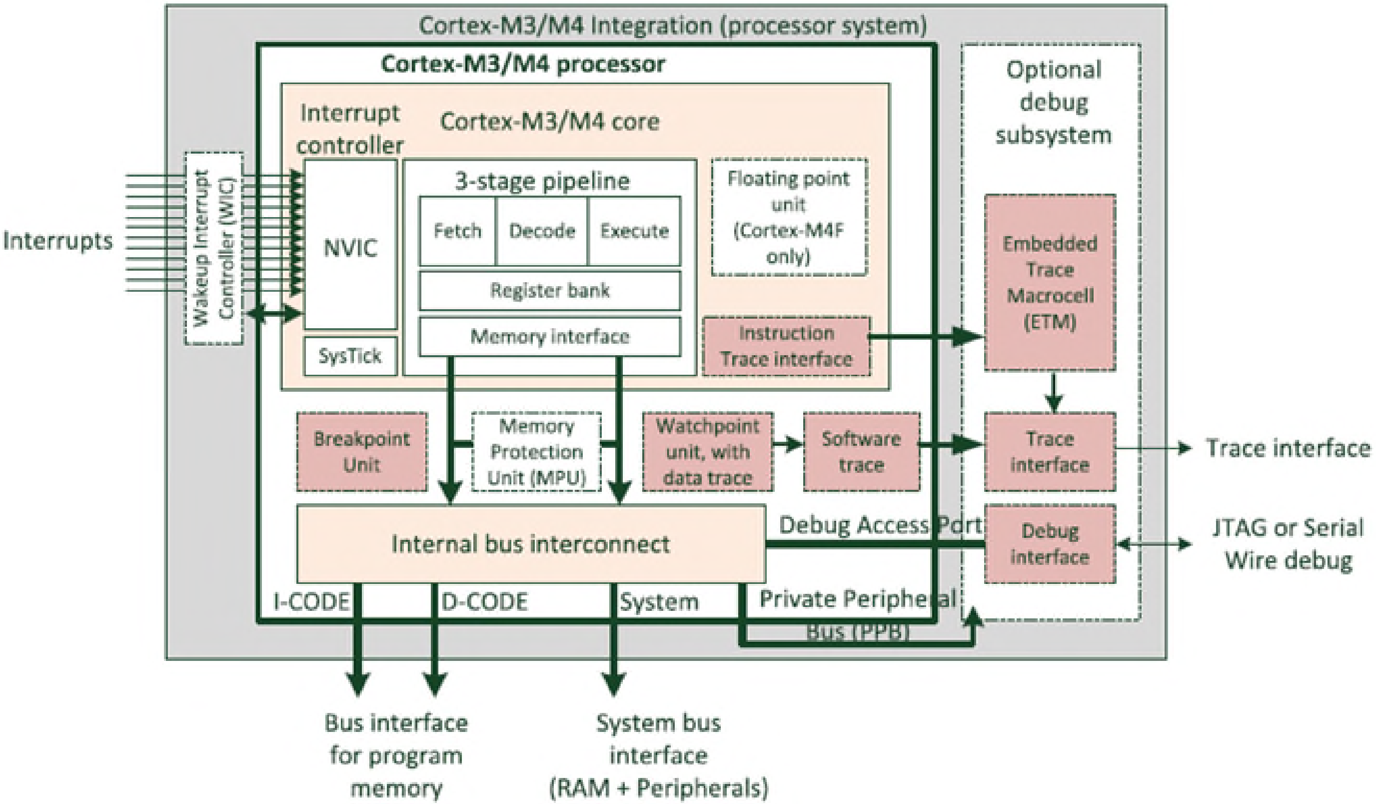
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}	2^{-16}
0	0	1	0	1	1	1	0	.	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

= 32 + 8 + 4 + 2 . 1/2 + 1/4 + 1/16

= 46 . 8125

https://baseconvert.com

Cortex-M Processor Architecture



RISC Harvard Architecture

Few opcodes

Fixed length opcode

Few addressing modes

Execute in 1 or 2 cycles

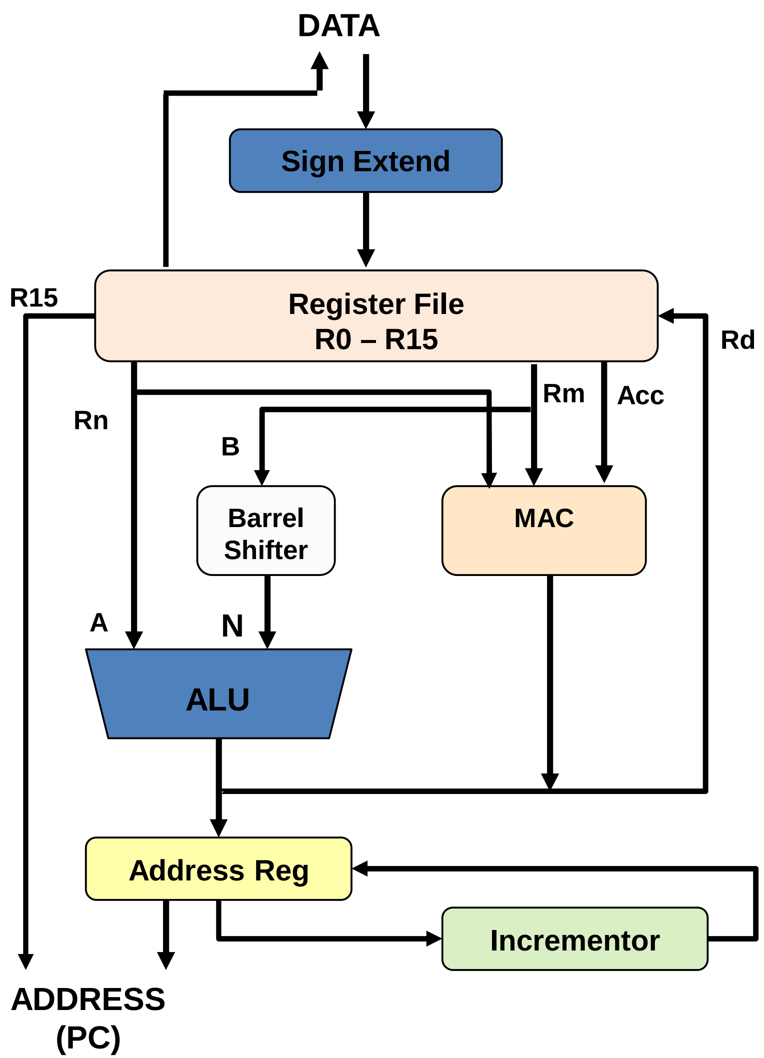
Only load and store can access memory

No one instruction can both read and write memory

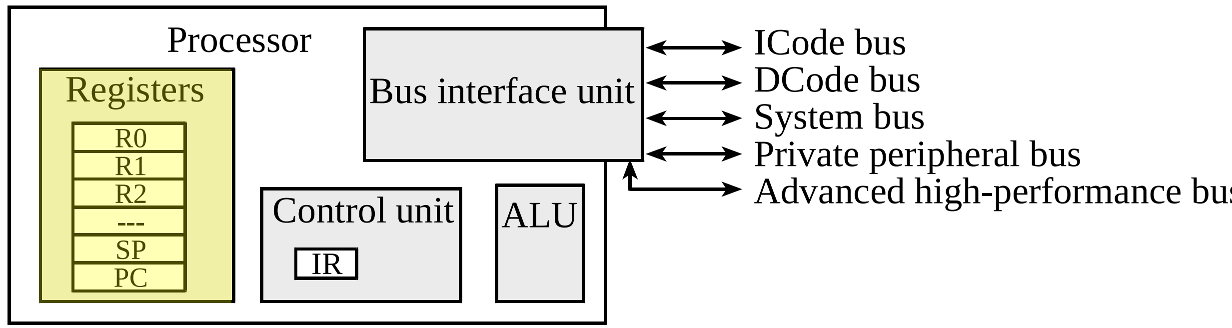
Many general-purpose registers

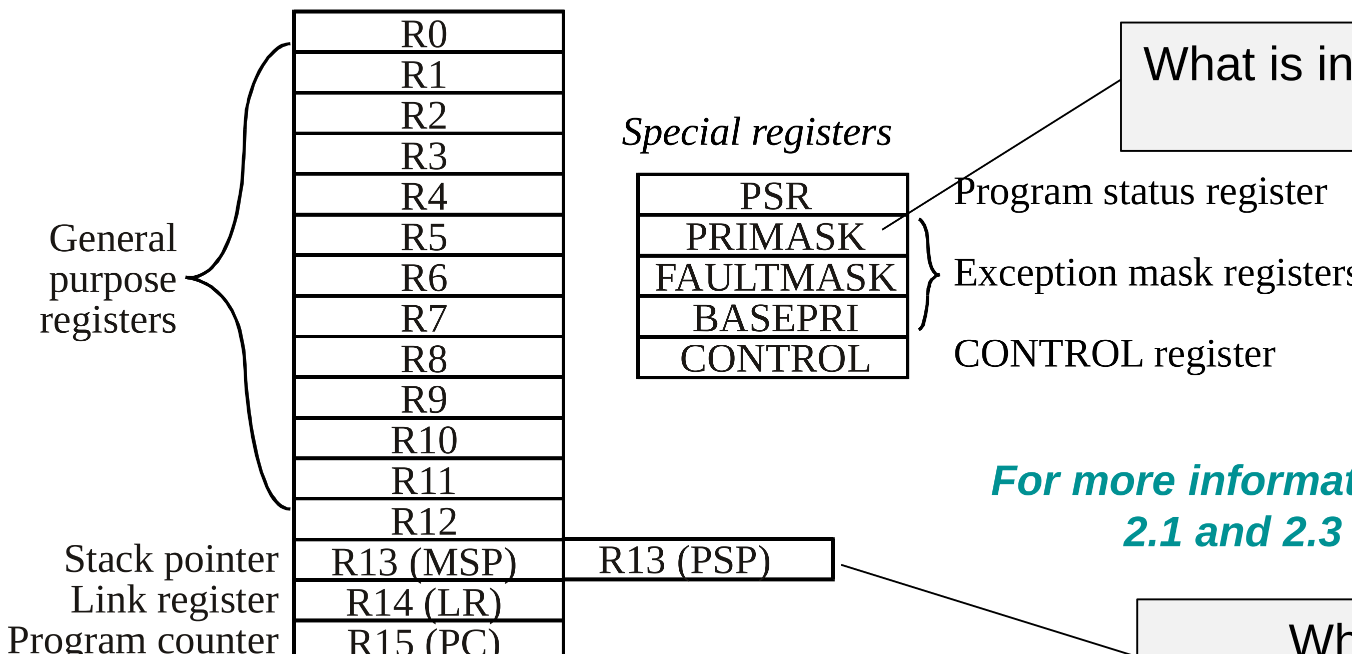
For more information read Sections 2.1 and 2.3 in the book

Cortex-M4 Datapath



Cortex-M Registers



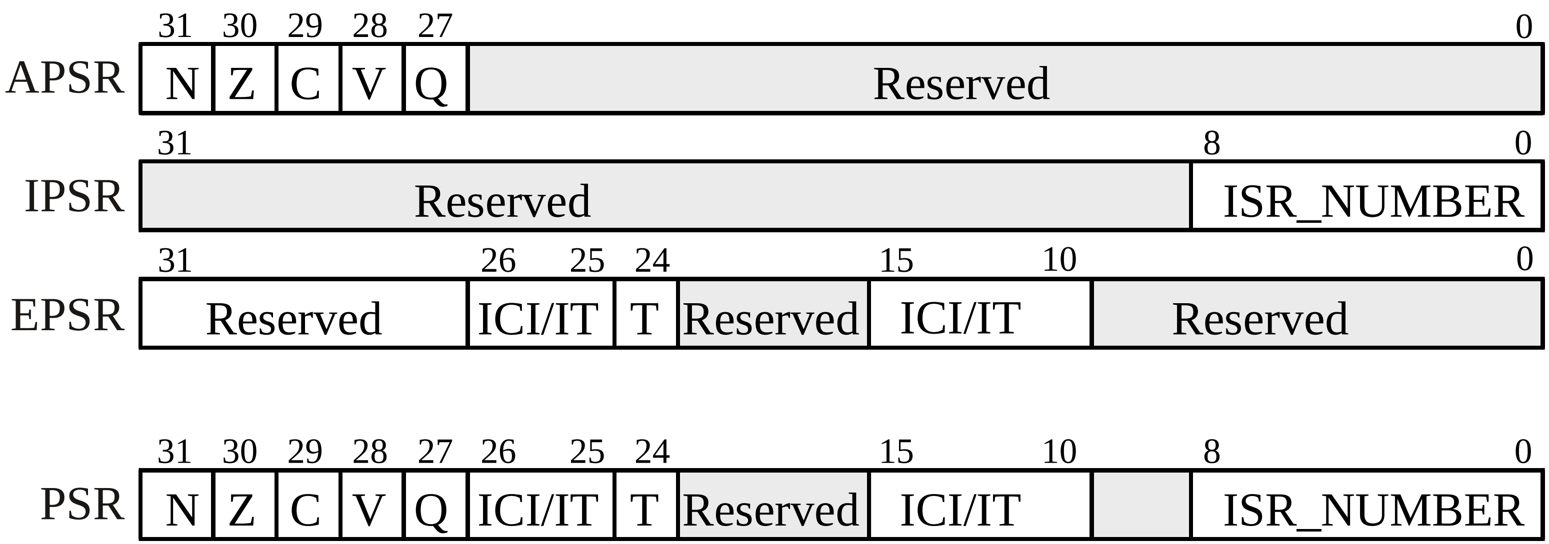


What is in PRIMASK?

Why two?

For more information read Sections 2.1 and 2.3 in the book

Cortex-M Status Registers



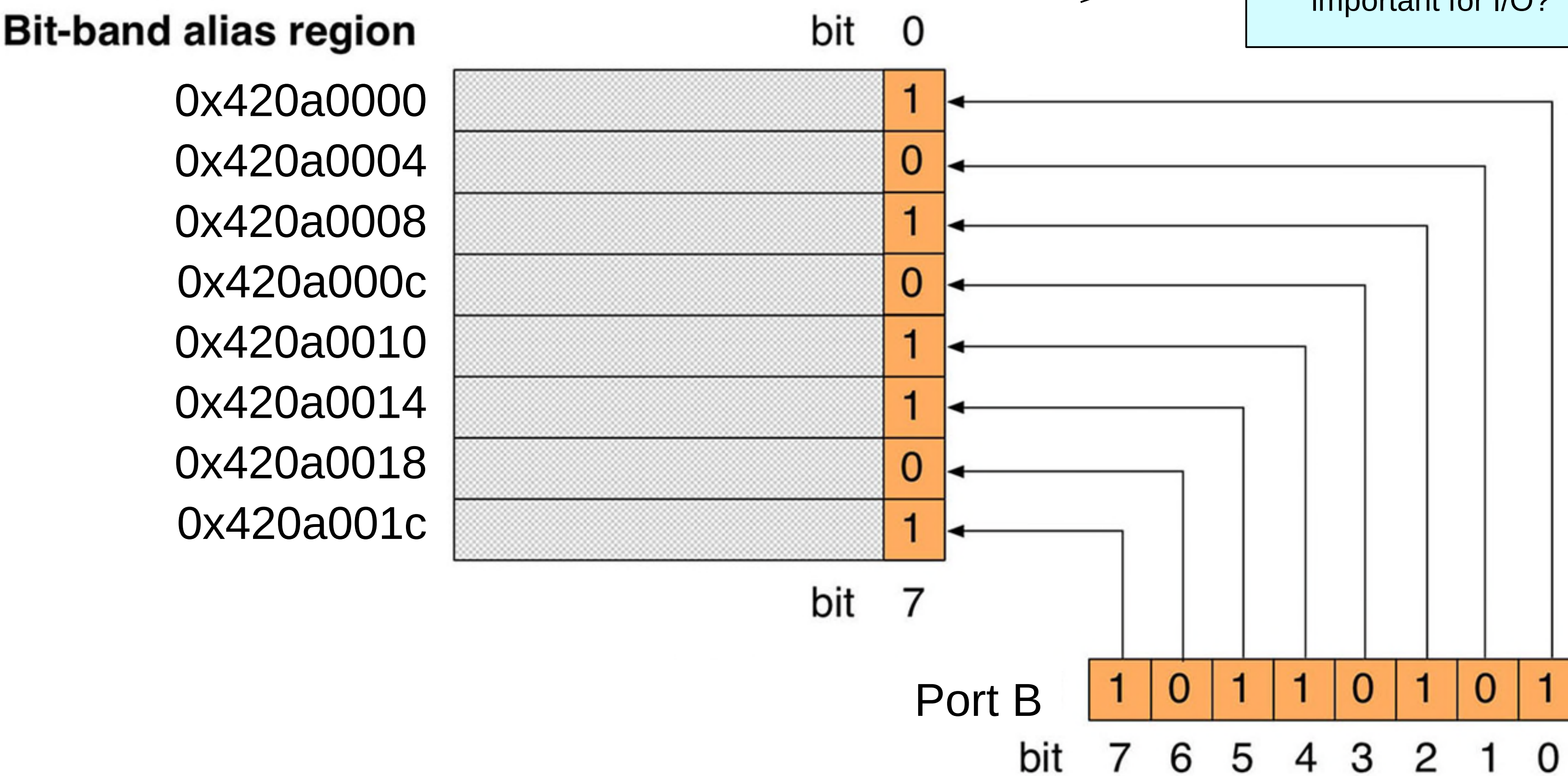
APSR contains the current state of the condition flags from previous instruction executions

IPSR contains the exception type number of the current Interrupt Service Routine (ISR)

EPSR contains the Thumb state bit and the execution state bits for the If-Then (`IT`) instruction

I/O Bit Banding Example

Why is bit-banding important for I/O?



Bit banding on all Cortex M4, bit specific addressing on just TM4C

Thread

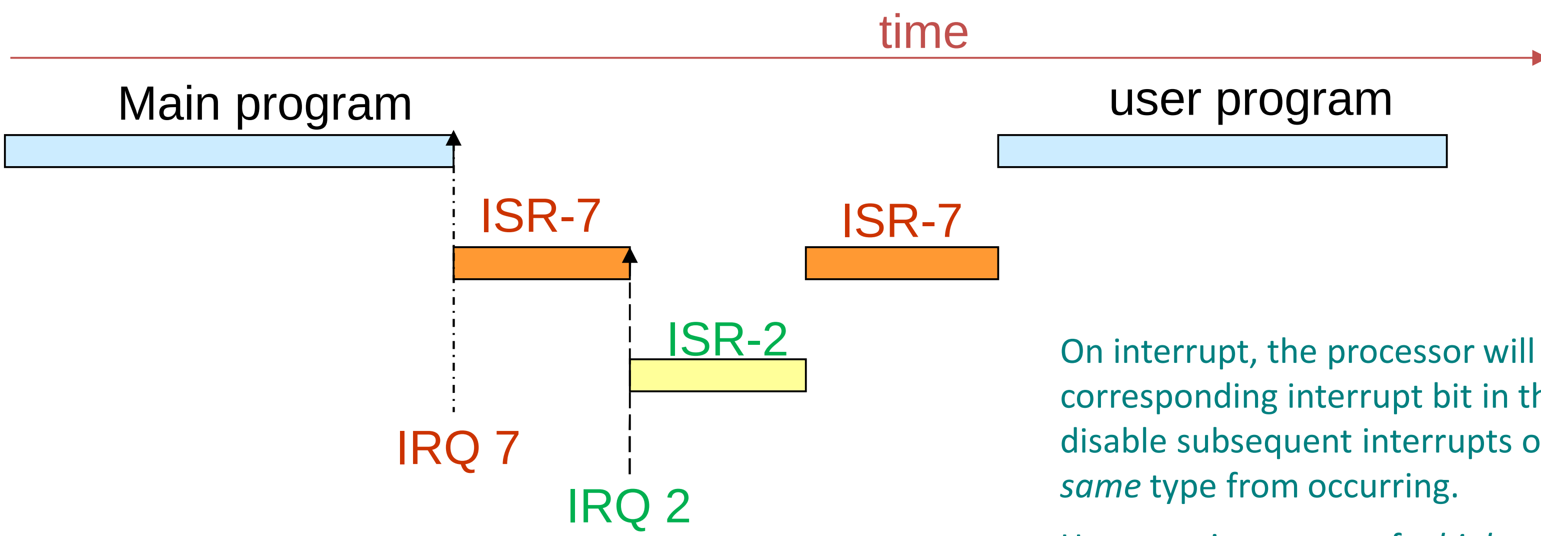
- A **thread** is defined as the path of action of software as it executes. If an interrupt occurs, then:
  - a **background thread** interrupt service routine (ISR) is called.
  - a new background thread is created for each interrupt request.
  - threads share global variables
  - local variables and registers used in the interrupt service routine are unique

Exceptions vs. Interrupts

- **Exceptions** are fault conditions that occur during the execution of a program.
  - Memory management, instruction fetch, unaligned memory access, divide by zero, etc.
  - **Synchronous** to the flow of instructions.

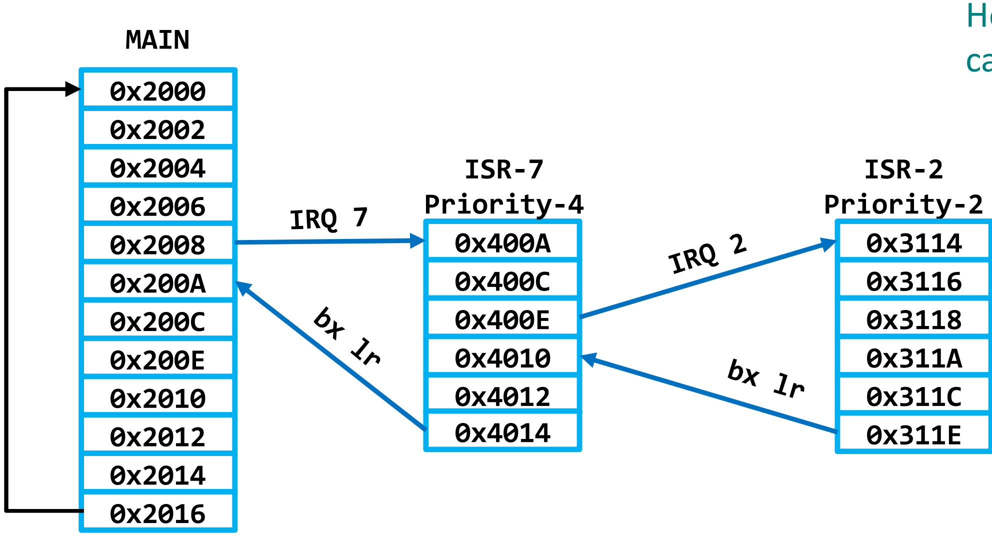
- **Interrupts** are events that cause the program flow to change.
  - Timers, GPIO, ADC, PWM, etc
  - **Asynchronous** to the flow of instructions

Nested ISR Background Threads



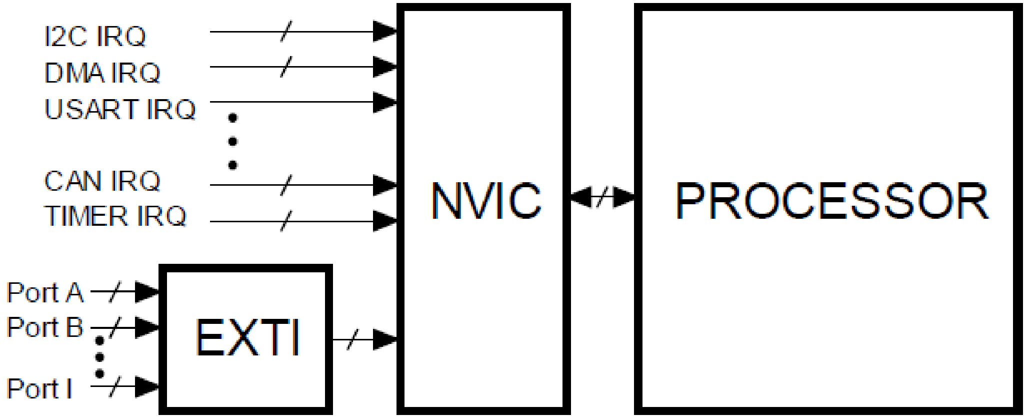
On interrupt, the processor will set the corresponding interrupt bit in the CPSR to disable subsequent interrupts of the *same* type from occurring.

However, interrupts of a *higher priority* can still occur.



Interrupts

- Cortex-M *Nested Vector Interrupt Controller* *(NVIC)*.
  - Interrupt Priorities
    - Active Status
  - Enable and Clear Enable registers
  - Set-Pending and Clear-Pending registers



For more information read Sections 4.7-4.9 in the book

NVIC Device Enable

- A device must be enabled in the NVIC and its priority set
- BASEPRI register sets priority of interrupts that are permitted to occur
  - if BASEPRI = 3, interrupts with priority
    - 0 – 2 can occur, suspending this interrupt
    - 3-7 will be postponed until this interrupt is finished

Arming (enabling) Device Interrupts

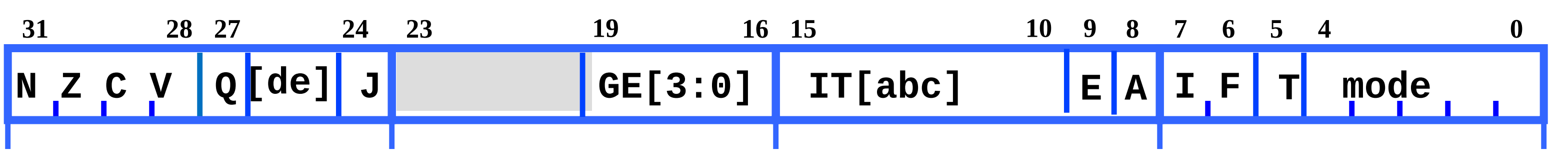
- Each potential interrupt source has a separate "arming" bit that enables interrupts
  - Set the "arm" bits for those devices from which it wishes to accept interrupts,
  - Deactivate "arm" bits in those devices from which interrupts are not allowed

Flag

- Each potential interrupt source has a separate **flag** bit.
  - hardware sets the flag when it wishes to request an interrupt
  - software clears the flag in the ISR to signify it is processing the request

Interrupt Enable/Disable

- **Interrupt Disable** bit, I, (bit 0 of PRIMASK) which is in the program status register.
  - enable all armed interrupts by setting I=0
  - disable all interrupts by setting I=1
  - Setting I=1 does not dismiss the interrupt requests, rather it postpones them.



Interrupt Enable/Disable

```
;*********** EnableInterrupts ***************
; disable interrupts
; inputs:  none
; outputs: none
EnableInterrupts
        CPSIE  I
        BX     LR

;*********** DisableInterrupts ***************
; disable interrupts
; inputs:  none
; outputs: none
DisableInterrupts
        CPSID  I
        BX     LR
```

In startup.s

Interrupt Requirements

1) Enable device in the NVIC

2) Initialization software will set the arm bit

    individual control bit for each possible flag that can interrupt

3) When it is convenient, the software will enable, **I=0**

    allow all interrupts now

4) Hardware action (busy to done) sets a flag

    e.g., new input data ready, output device idle, periodic, alarm

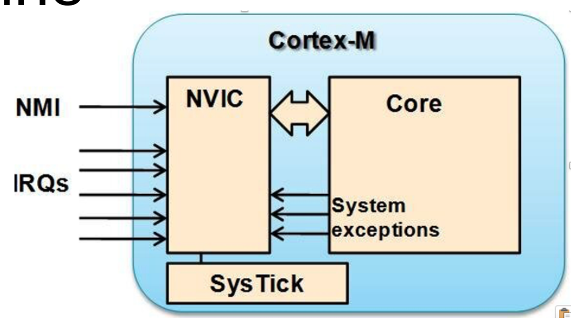
Interrupts

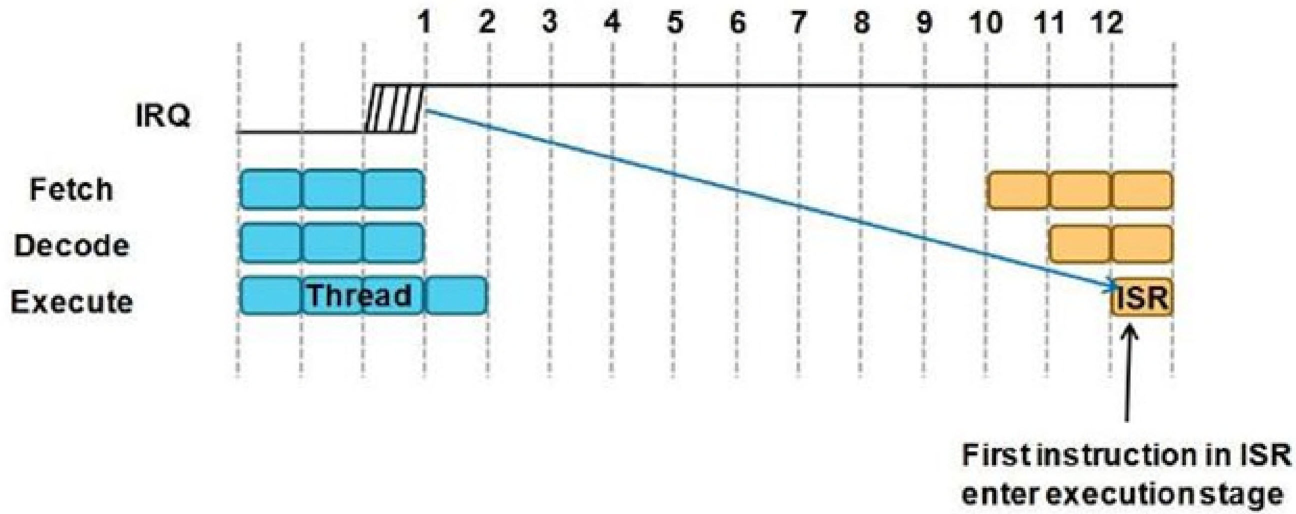
- An **interrupt** is the automatic transfer of software execution in response to hardware that is **asynchronous** with current software execution.
  - External I/O device (like a keyboard or printer) or
  - An internal event (like a periodic timer, ADC, etc.)
  - Occurs when the hardware needs service (busy to done state transition)

Show the two ISRs in Lab2.c

Interrupt Processing

- **All interrupting systems must have:**
  - the ability for the hardware to request action from computer
  - the ability for the computer to determine the source of the request
  - the ability for the computer to acknowledge the interrupt



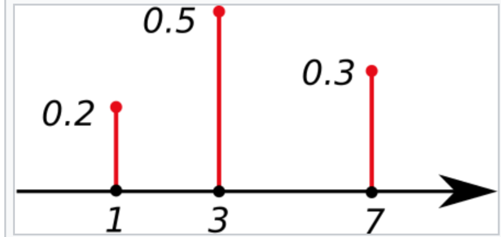


Latency

- Input device
  - Latency is the time between new input data ready and the software reading the data
- Output device
  - Latency is the time between output device idle and the software giving the device new data to output.
- Periodic events (ADC, DAC, control system).
  - Latency is the time between when it is supposed to run and when it is actually run.
  - Time jitter

Jitter

- **Real-time** system
  - Sampling jitter
  - Multicycle instructions

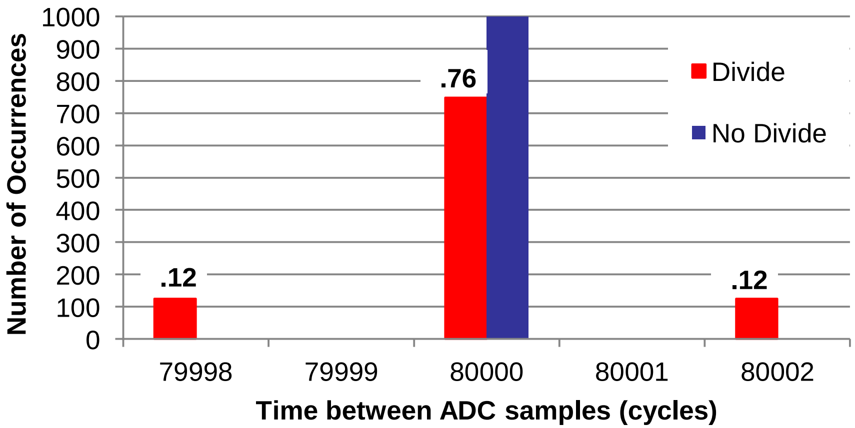


The graph of a probability mass function. All the values of this function must be non-negative and sum up to 1.

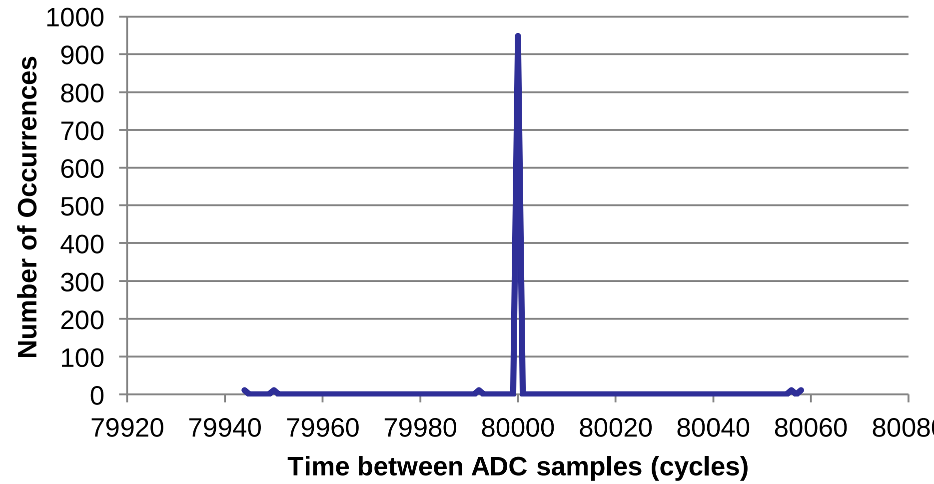
https://en.wikipedia.org/wiki/Probability_mass_function

In **probability** and statistics, a **probability mass function** (pmf) is a **function** that gives the **probability** that a discrete random variable is exactly equal to some value.

One Interrupt



Two (or more) interrupts



Performance measures

- **Hardware** or **device latency** is the time between when an I/O device is given a command, and the time when command is completed
- **Bandwidth** is the maximum data flow or capacity of a channel
  - bandwidth can be limited by the I/O device or software
  - can be reported as an overall average or a short-term max
- **Throughput** measures how much data was transmitted into the channel

Bandwidth Limits

- **I/O bound** is defined as
  - Bandwidth is limited by speed of I/O device
  - Making the I/O device faster will increase bandwidth
  - Making the software run faster will not increase bandwidth
  - Software often waits for the I/O device

Bandwidth Limits

- **CPU bound** is defined as
  - Bandwidth is limited by speed of executing software
  - Software does not have to wait for the I/O device
  - Making the I/O device faster will not increase bandwidth
  - Making the software run faster will increase bandwidth

For more information read Sections 5.1, 5.2 in the book

Debugging

- **Types**
  - performance debugging (timing)
  - functional debugging (data)
- **Goal of debugging**
  - maintain and improve software
  - remedy faults or to correct errors in a program
  - role of a debugger is to support this endeavor
- **The debugging process**
  - testing,
  - stabilizing,
  - localizing, and
  - correcting errors.

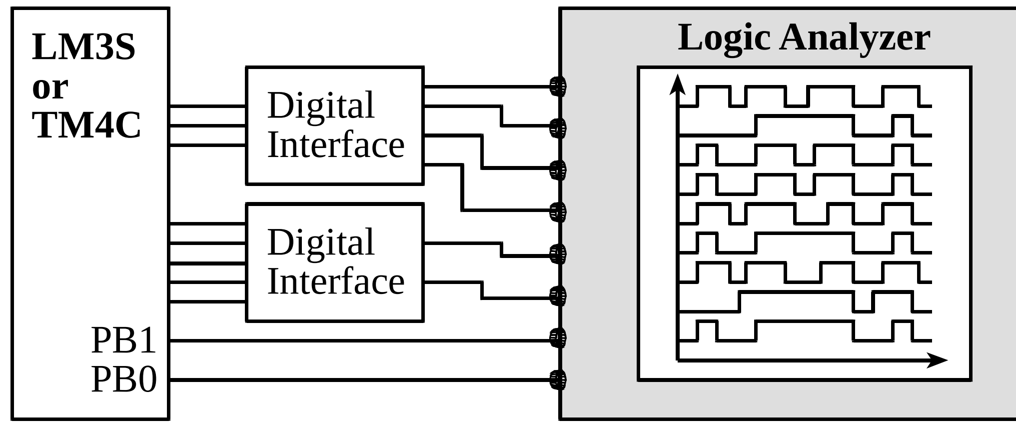
For more information read Section 3.9 in the book

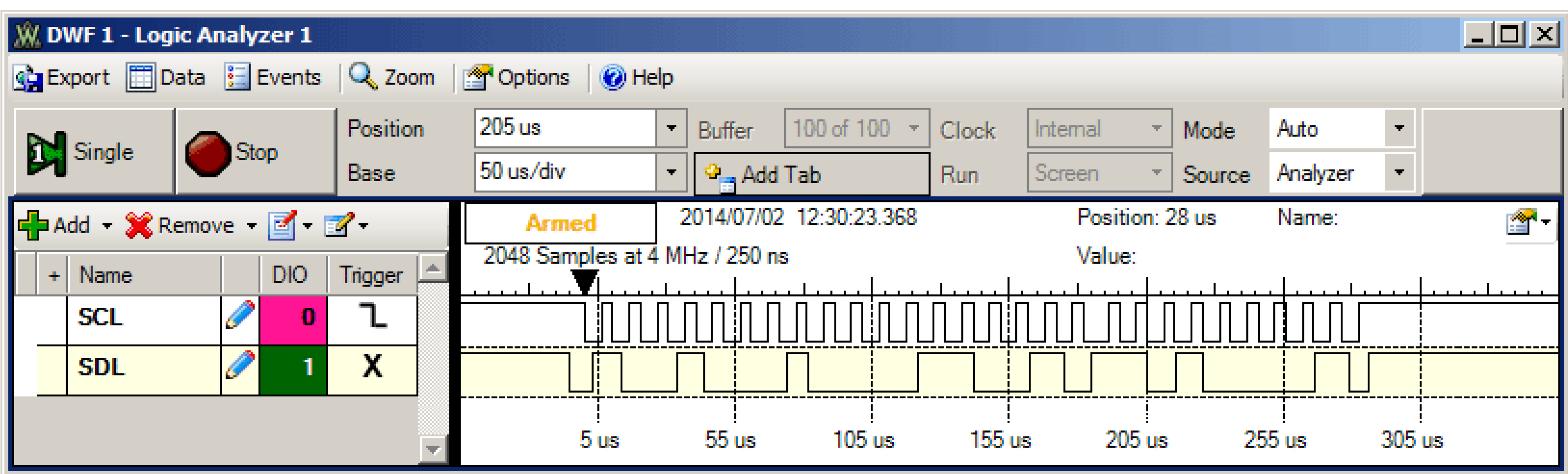
Manual Methods

- **Desk-checking**
  - Hand execute the program and think about it a lot
    - Write down intermediate results
  - Then execute program and compare
    - What you thought it should do
    - What is it doing?
- **Dumps**
  - save important data into an array, look at it later
- **Print statements**
  - print important during execution

Hardware debugging tools

- **Logic analyzer**
  - Multiple channel, digital, storage scope
  - Flexible method of triggering





Software Debugging Tools

- **A debugging instrument**
  - software that is added to the program for the purpose of debugging, e.g., print statement
  - instrument added using editor, assembler & loader

```
int f(void)
{
    int x = 5;
    x = x + 1;
    assert(x > 1);
}
```

- **Assertions**
  - Programmers can use assertions to help specify programs and to reason about program correctness.

C.A.R. Hoare, An axiomatic basis for computer programming, *Communications of the ACM, 1969.*

Software Debugging Tools

- **Choose one of the following techniques**
  - Place all instruments in the first column of your code, they are easy to see
  - Define instruments with specific pattern in their names
  - Use instruments that test a run time global flag
  - **Leave a permanent copy of the debugging code**
    - will cause it to suffer runtime overhead when activated
    - simplifies "on-site" customer support.
  - Use conditional compilation (or conditional assembly)
    - Easy to remove all instruments
  - **But, Leave the instruments**
    - Because this is the way it was proven to work
    - May need more testing

Intrusiveness

Degree of perturbation caused by the debugging itself (Heisenberg)

How much the debugging slows down execution

- **Nonintrusive**
  - Characteristic or quality of a debugger
  - Allows system to operate as if debugger did not exist
  - e.g., logic analyzer, oscilloscope, ICE
- **Minimally intrusive**
  - Negligible effect on the system being debugged
  - e.g., dumps (ScanPoint) and monitors, JTAG, assertions
- **Highly intrusive**
  - print statements, breakpoints and single-stepping

Nyquist Sampling Theorem

A band-limited signal of finite energy, which has no frequency components higher than W Hertz, may be completely recovered from a knowledge of its samples taken at a rate greater than 2W samples per second.

- sampling jitter
- sample precision

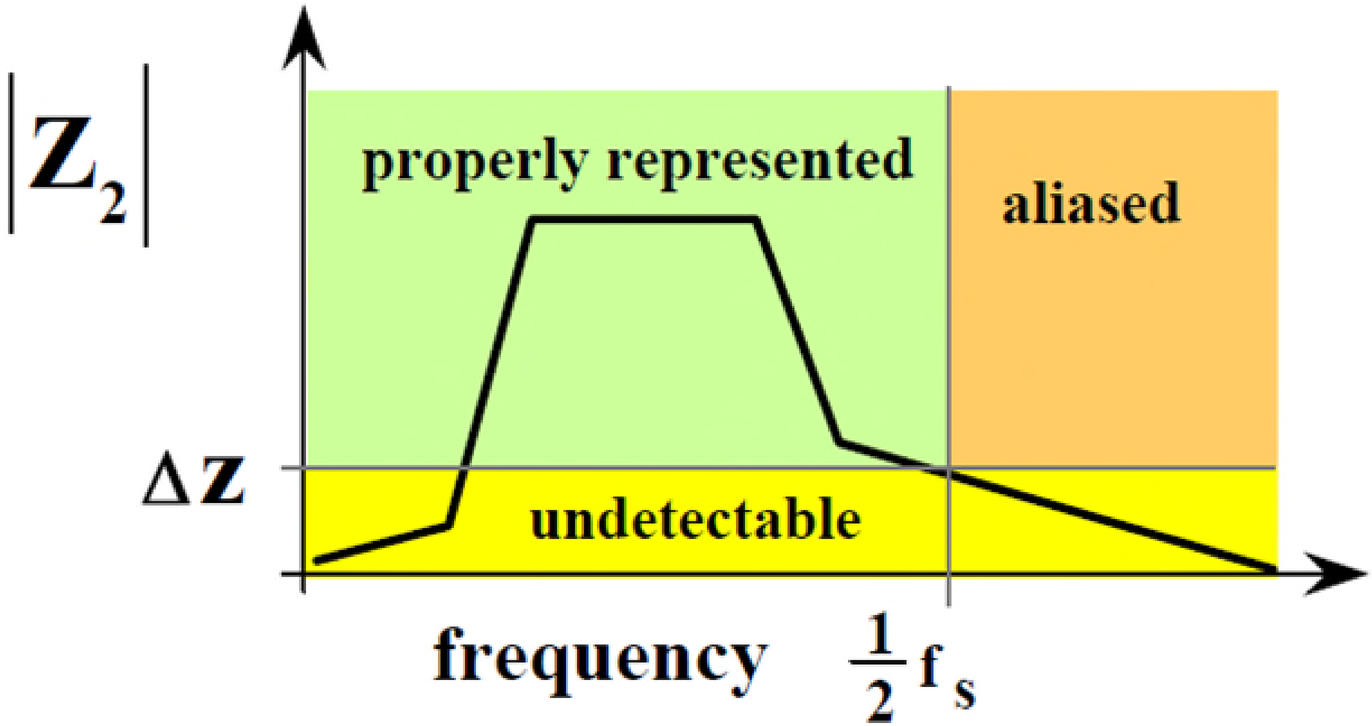
Harry Nyquist

Valvano Postulate

If f_{max} is the largest frequency component of the analog signal, then you must sample more than ten times f_{max} in order for the reconstructed digital samples to look like the original signal when plotted on a voltage versus time graph.

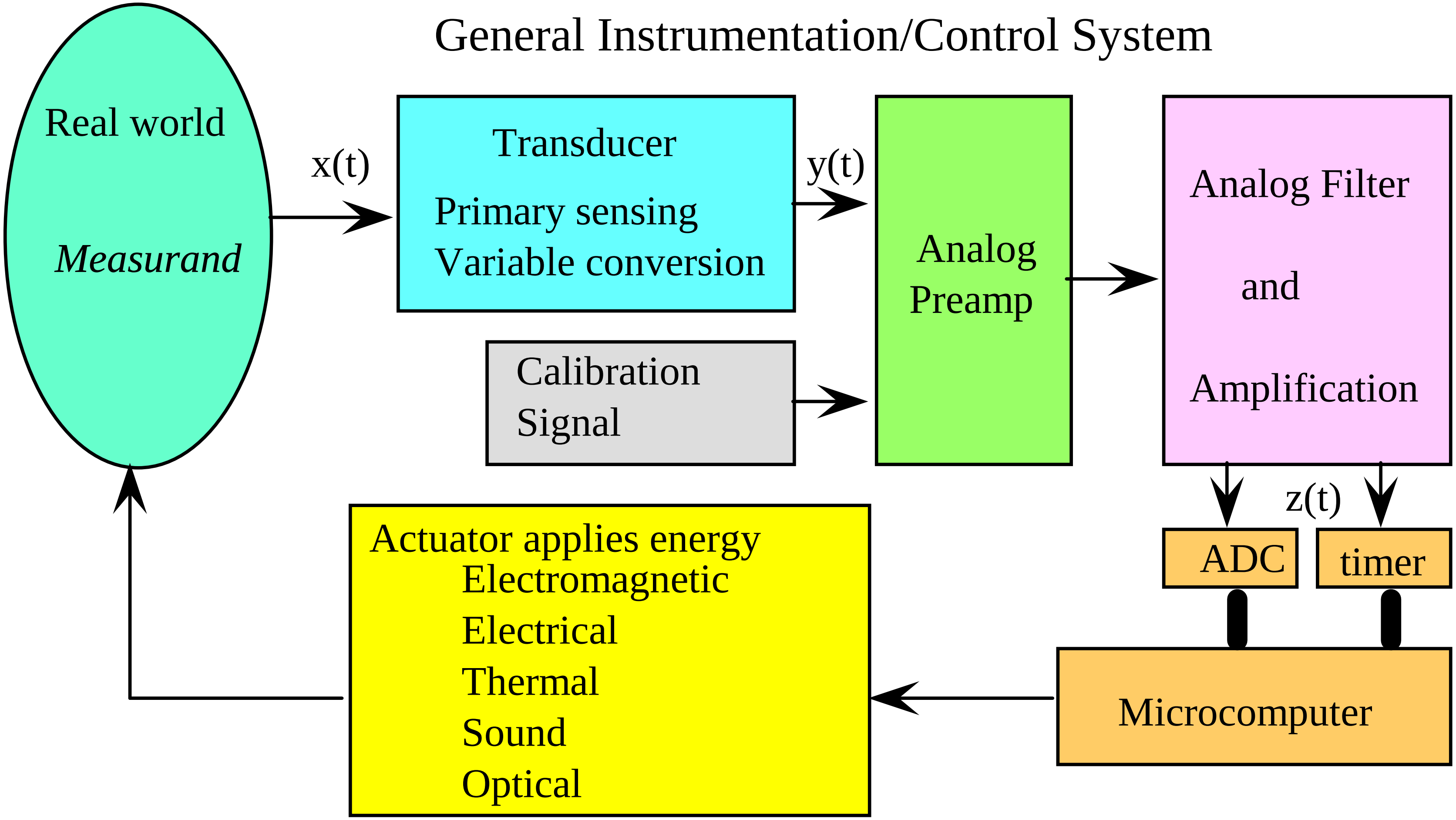
http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C14_Interactives.htm

Sampling Window



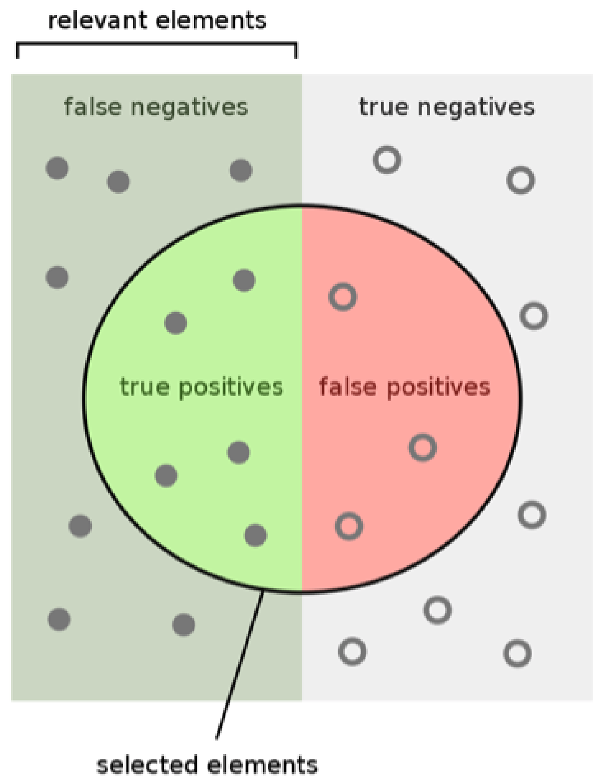
To prevent aliasing => no measurable signal above ½**fs**.

Data Acquisition System

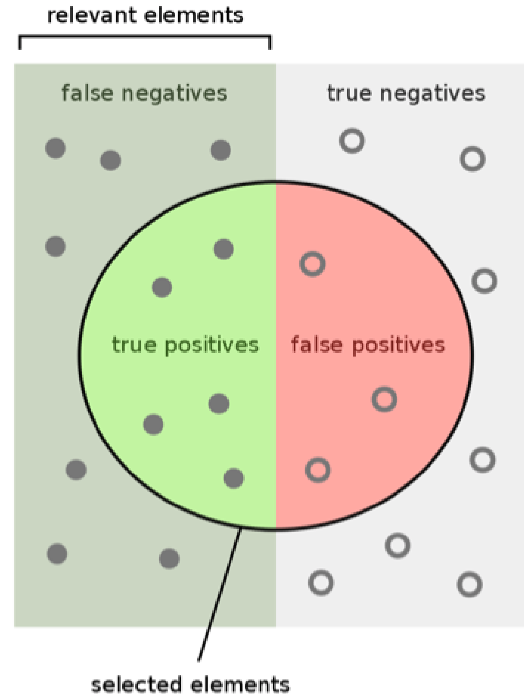


Qualitative Data Acquisition System Parameters

- true positive (TP)
  - *event being monitored occurs and system detects it*
- false positive (FP)
  - *event being monitored does not occur but system reports it*
- false negative (FN)
  - *event being monitored occurs and system fails to detect it*



Qualitative Data Acquisition System Parameters



- Prevalence = (TP + FN) / (TP + TN + FP + FN)
- Sensitivity = TP / (TP + FN)
- Specificity = TN / (TN + FP)
- Positive Predictive Value = TP / (TP + FP)
- Negative Predictive Value = TN / (TN + FN)

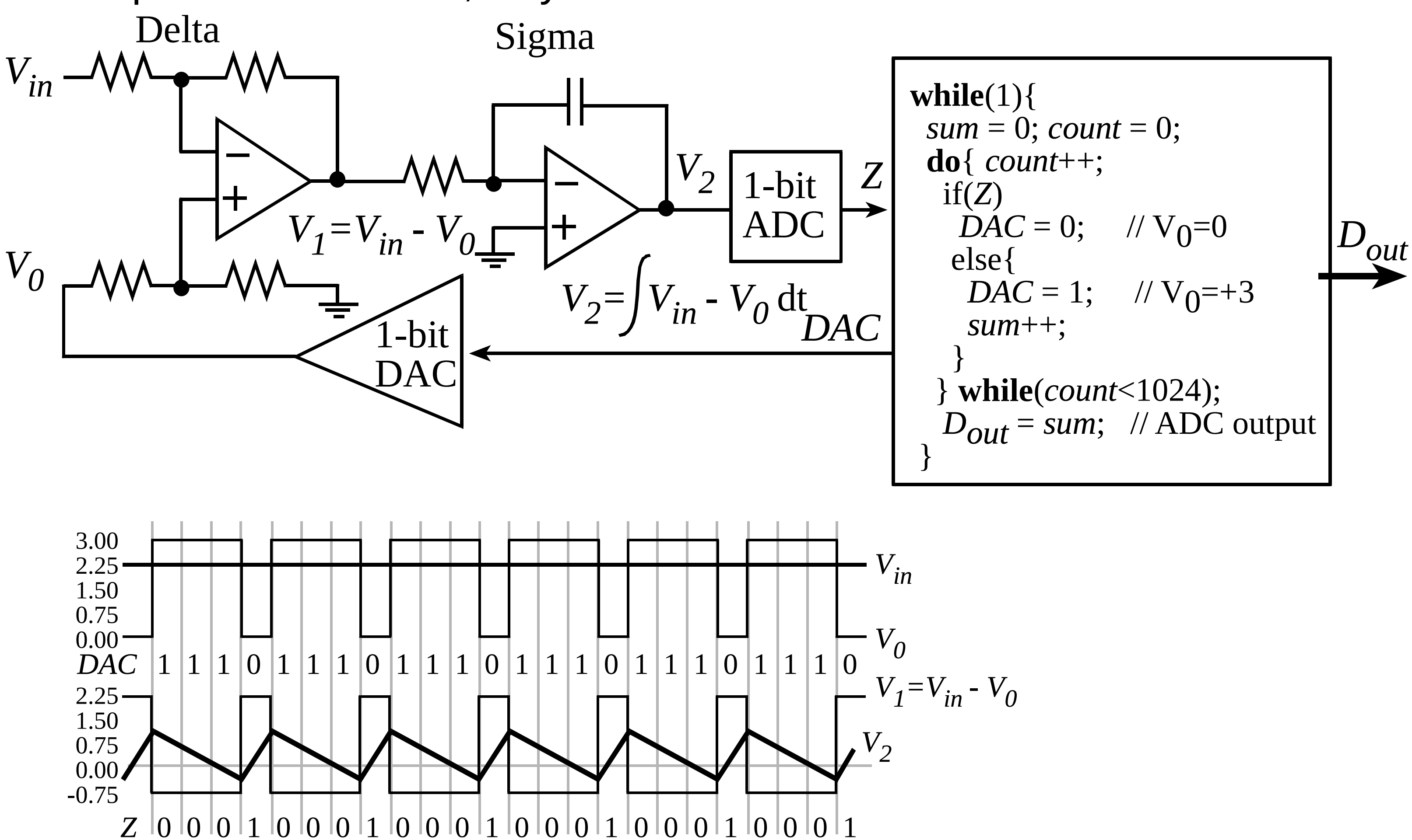
Data Acquisition System

- **Quantitative DAS parameters**
  - range (r_x)
  - resolution (Δx)
  - coefficient of variation (σ/μ)
  - precision (n_x in alternatives)
  - frequencies of interest (f_{min} to f_{max})
  - repeatability (σ of repeated measurements, same conditions)
  - reproducibility (σ of repeated measurements, different conditions)

Analog to Digital Methods

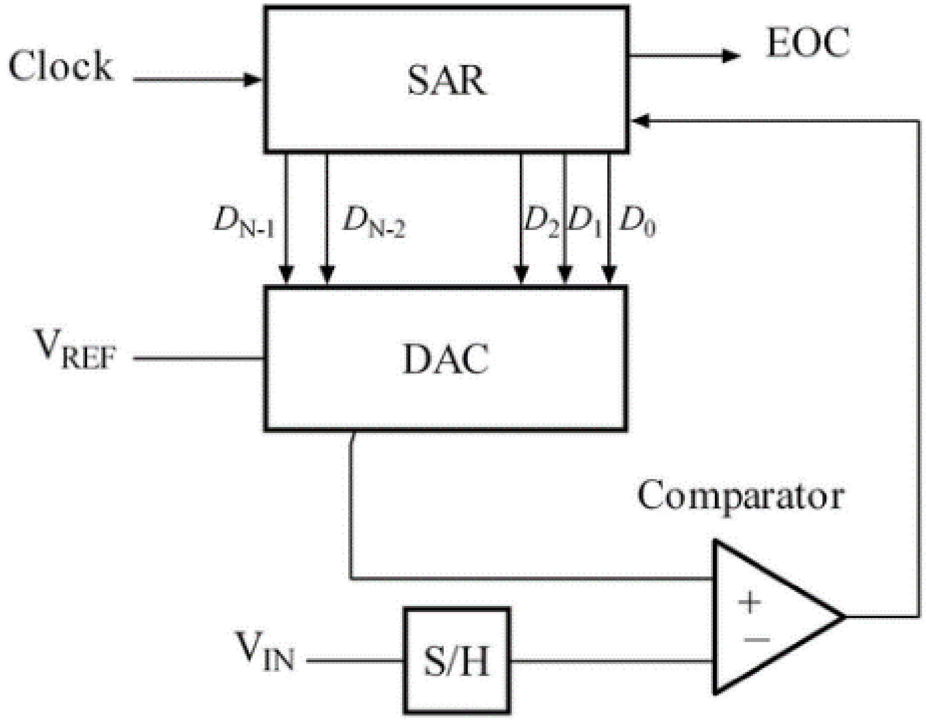
- **Sigma delta**
  - Explain how it works, why we would use it

[∑∆ Tutorial]



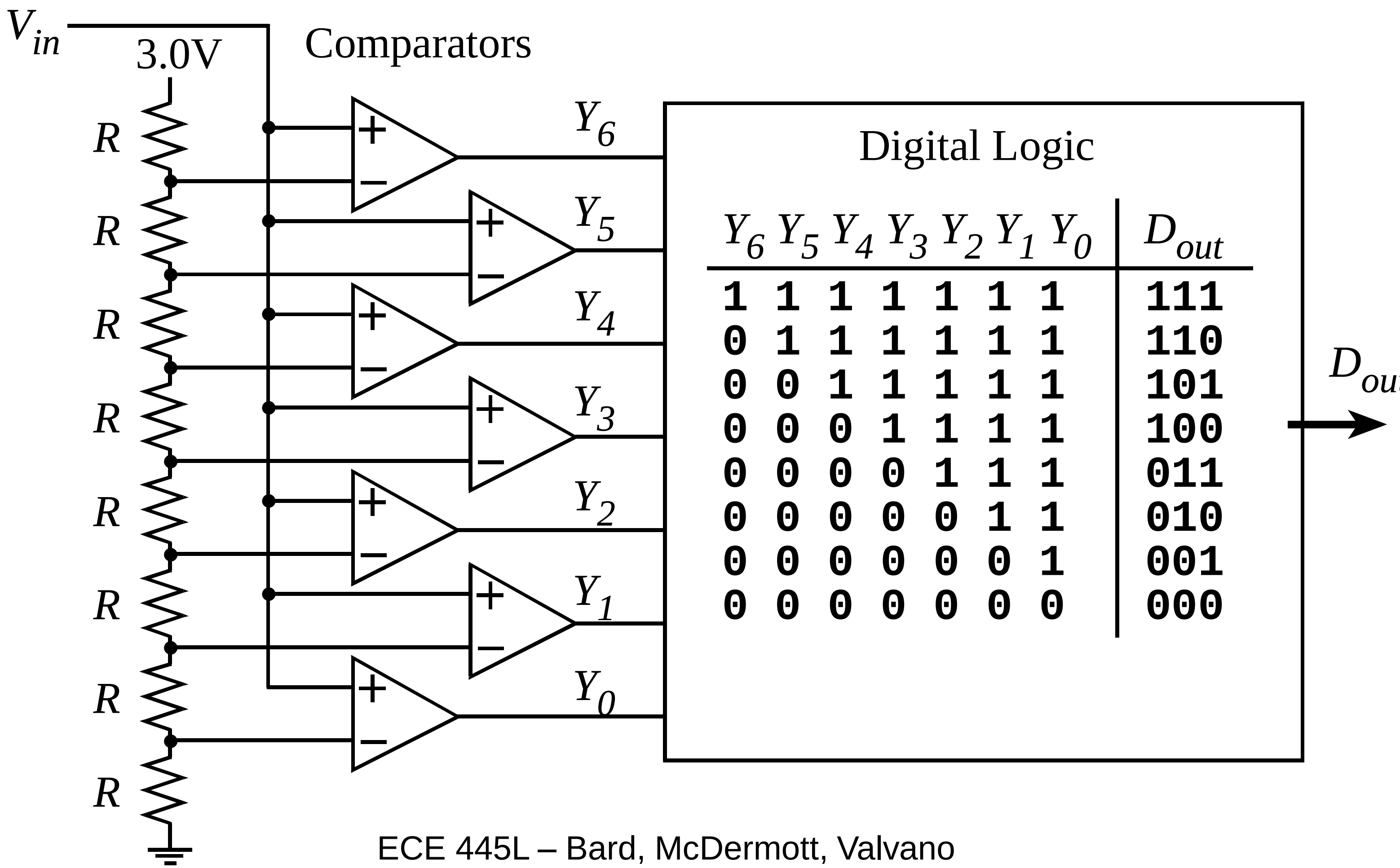
Analog to Digital Conversion

- Successive Approximation ADC
  - V_{IN} is approximated as a static value in the *sample and hold* circuit
  - the *successive approximation register* is a counter that increments each *clock* as long as it is enabled by the *comparator*
  - the output of the SAR is fed to a DAC that generates a voltage for comparison with V_{IN}
  - when the ouput of the DAC = V_{IN} the value of SAR is the digital representation of V_{IN}



Analog to Digital Methods

- **Flash ADC Converter**
  - Explain how it works, why we would use it



Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	D_{out}
1	1	1	1	1	1	1	111
0	1	1	1	1	1	1	110
0	0	1	1	1	1	1	101
0	0	0	1	1	1	1	100
0	0	0	0	1	1	1	011
0	0	0	0	0	1	1	010
0	0	0	0	0	0	1	001
0	0	0	0	0	0	0	000

Time Jitter

- Sampling rate f_s, $\Delta t = 1/f_s$
- Definition of time-jitter:
  - Measure t_i the time the task is actually run
  - Calculate $\delta_i = t_i - t_{i-1}$
  - Jitter is $\max\delta_i - \min\delta_i$
  - Sampling accuracy is $\max|\delta_i - \Delta t|$
- Real time systems with periodic tasks, must have an upper bound, **k**,
  - $-\mathbf{k} \leq \delta_i \leq +\mathbf{k}$ for all **i**

Delayed Service

- Consequences
  - Nyquist's theorem no longer holds
    - requires constant sampling interval
  - data acquisition and control systems operate using incorrect calculated values
    - consider derivative $dx/dt = ((x(t)-x(t-\Delta t))/\Delta t$
  - errors in signal generation
    - sound is distorted
    - picture is blurry

$$\delta V = \delta t * dV/dt$$

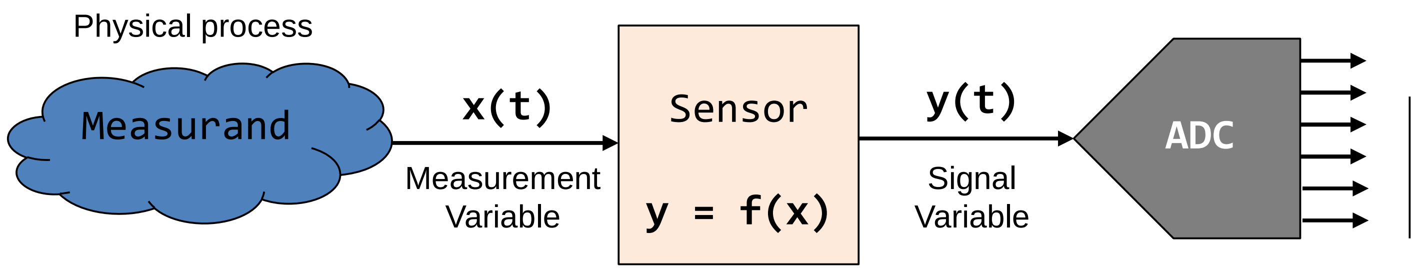
Error

Jitter

Slew rate

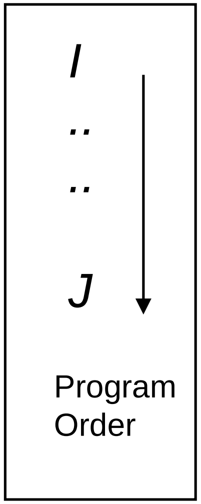
Analog-to-Digital Converter Resolution

- Observable **x(t)** is sensed via transducer as signal **y(t)**
  - assume a relation, $\mathbf{y = f(x)}$
  - range of **x** is $\mathbf{r_x}$ and range of **y** is $\mathbf{r_y}$
  - precision of **x** and **y** is $\mathbf{n_x}$ and $\mathbf{n_y}$ respectively
  - resolution of **x** and **y** is $\Delta\mathbf{x}$ and $\Delta\mathbf{y}$ respectively and $\Delta\mathbf{x} = \mathbf{r_x/n_x}$



Data Hazard Classification

Given two instructions ***I*****,** ***J*****,** with ***I*** occurring before ***J*** in an instruction stream (program execution order):



<u>RAW (read after write):</u> *A true data dependence violation*

J tried to read a source before *I* writes to it,
so *J* incorrectly gets the old value.

<u>WAW (write after write):</u> *A name dependence violation*

J tries to write an operand before it is written by *I*
The writes end up being performed in the wrong order.

<u>WAR (write after read):</u> *A name dependence violation*

J tries to write to a destination before it is read by *I*,
so *I* incorrectly gets the new value.

RAR (read after read): Not a hazard.

Critical Section

- Shared global
- Non-atomic access
- At least one write

```
int main(void){
  while(1){
    GPIO_PORTF_DATA_R ^= 0x02;
  }
}
Loop LDR   r0,[pc,#32]
     LDR   r0,[r0,#0x00]
     EOR   r0,r0,#0x02
     LDR   r1,[pc,#20]
     STR   r0,[r1,#0x3FC]
     B     Loop
```

```
void ISR1(void){
  GPIO_PORTF_DATA_R ^= 0x04;
  Stuff1();
}
ISR1 ...
     LDR   r1,[pc,#56]
     LDR   r0,[r1,#0x00]
     EOR   r0,r0,#0x04
     LDR   r1,[pc,#52]
     STR   r0,[r1,#0x3FC]
     ...
```

Solution: two ways to remove sharing

Critical Section

- Shared global
- Non-atomic access
- At least one write

Lab 3

```
int main(void){
...
  while(1){

    Display(Hour);
    Display(Minute);

  }
}
```

```
void ISR1(void){
  Second++;
  if(Second==60){
    Second = 0;
    Minute++;
    if(Minute == 60){
      Minute = 0;
      Hour++;
    }
  }
}
```

Critical Section

- Shared global
- Non-atomic access
- At least one write

Lab 3

```
int main(void){
...
  while(1){uint32_t myH,myM;
    Disable_Interrupts();
    myH = Hour;
    myM = Minute;
    Enable_Interrupts();
    Display(myH);
    Display(myM);
  }
}
```

```
void ISR1(void){
  Second++;
  if(Second==60){
    Second = 0;
    Minute++;
    if(Minute == 60){
      Minute = 0;
      Hour++;
    }
  }
}
```

Solution: make atomic

Critical Section

- Shared global
- Non-atomic access
- At least one write

Lab 3

```
uint32_t Time; // shared
int main(void){
...
  while(1){uint32_t myT;
    myT = Time;
    Display(myT/100);
    Display(myT%100);
  }
}
```

```
void ISR1(void){
  Second++;
  if(Second==60){
    Second = 0;
    Minute++;
    if(Minute == 60){
      Minute = 0;
      Hour++;
    }
  } Time = 100*Hour+Minute;
}
```

Solution: make atomic

What is UTC?
https://en.wikipedia.org/wiki/Coordinated_Universal_Time

Need Reentrant Code

- Function call by two threads
- Recursion

Not reentrant!
Why?

```
int PutFifo(int32_t data){

  if(((PutI+1)%SIZE) == GetI){

    return 0;
  }
  FIFO[PutI] = data;
  PutI = (PutI+1)%SIZE;

  return 1;
}
```

```
void GPIOPortA_Handler (void){
  PutFifo(GPIO_PORTA_DATA_R);
}
```

```
void GPIOPortB_Handler (void){
  PutFifo(GPIO_PORTB_DATA_R);
}
```

Wrong: Use local variables
Correct: Interrupt priority
Correct: make atomic

Reentrant Code

- Function call by two threads

```
int PutFifo(int32_t data){
  DisableInterrupts();
  if(((PutI+1)%SIZE) == GetI){
    EnableInterrupts();
    return 0;
  }
  FIFO[PutI] = data;
  PutI = (PutI+1)%SIZE;
  EnableInterrupts();
  return 1;
}
```

```
void GPIOPortA_Handler (void){
  PutFifo(GPIO_PORTA_DATA_R);
}
```

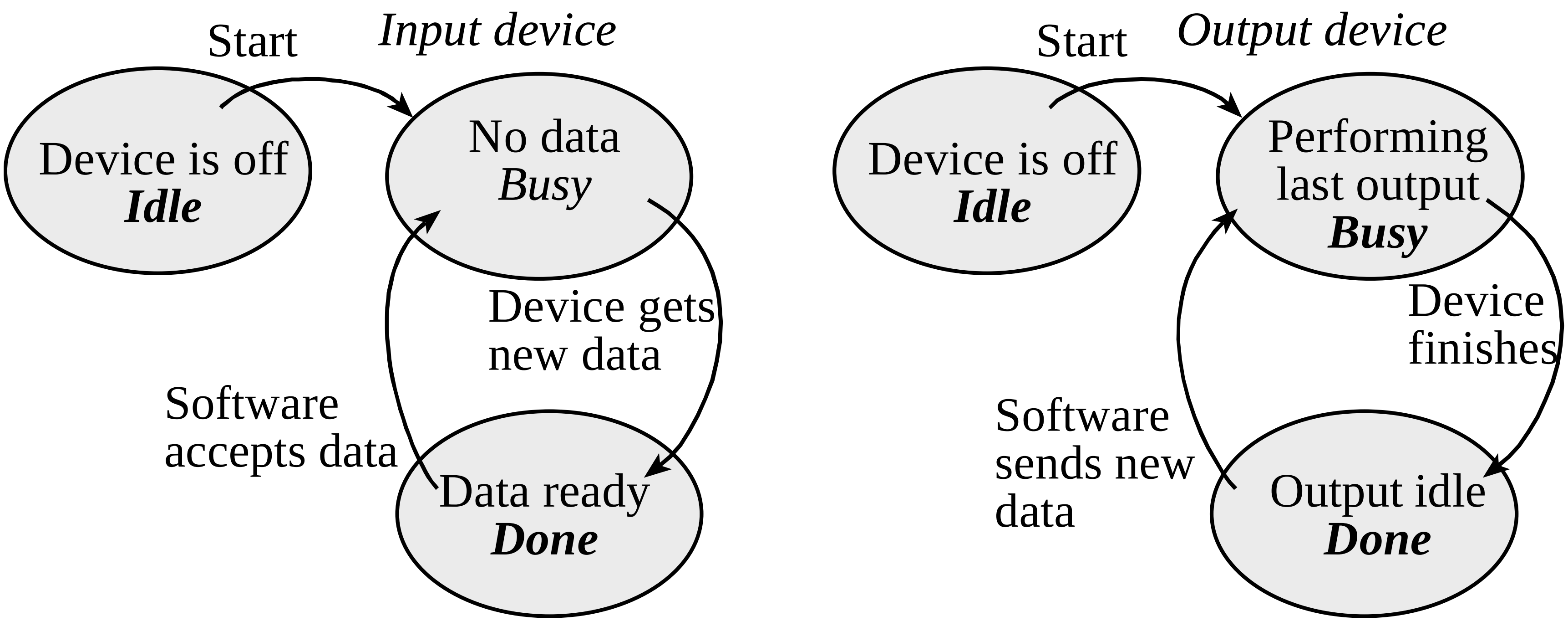
```
void GPIOPortB_Handler (void){
  PutFifo(GPIO_PORTB_DATA_R);
}
```

Correct: Atomic access to Globals

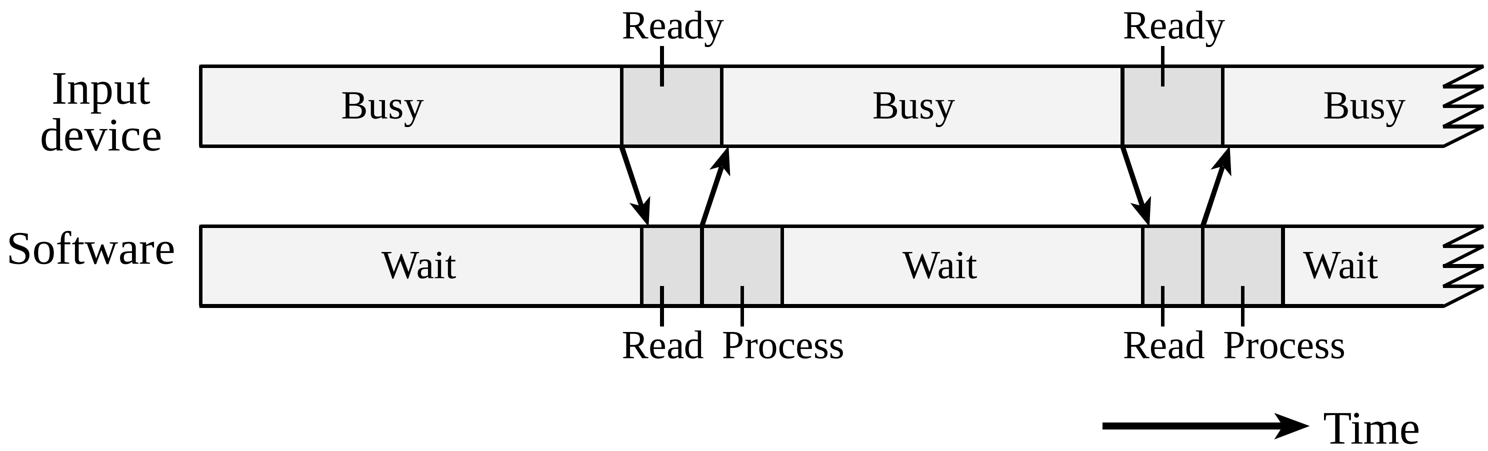
Input/Output Synchronization

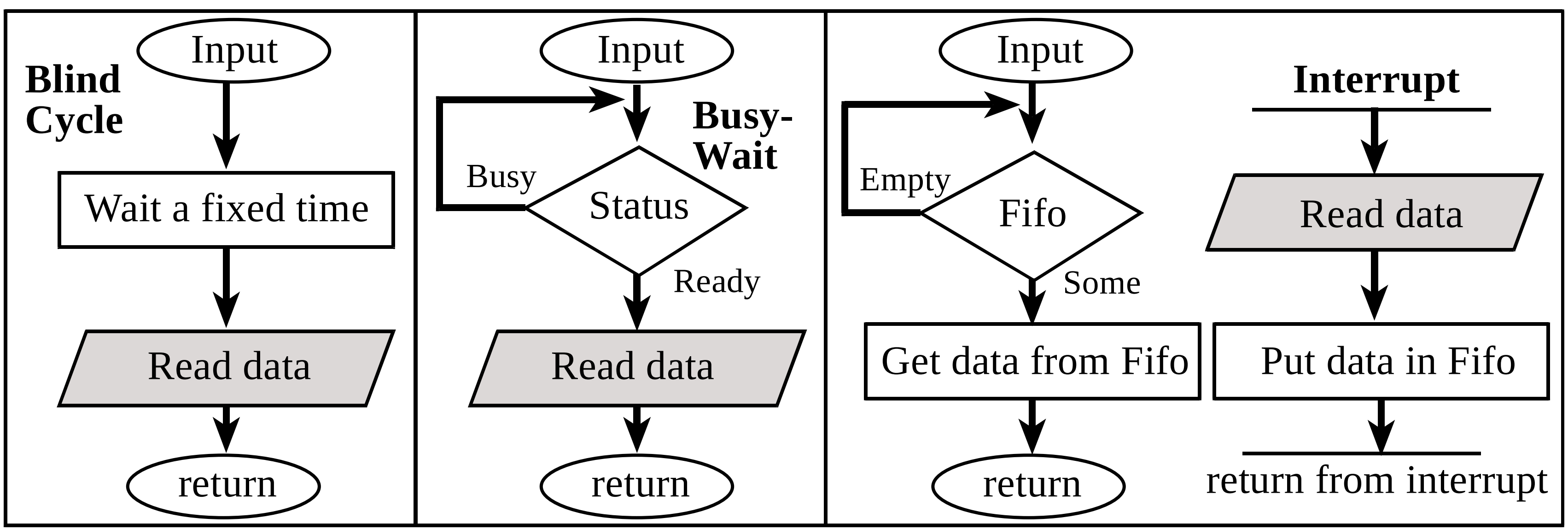
- **Timing Mismatch**
  - Processor ~ MHz
  - Peripheral ~ kHz or Hz
  - Asynchronous
- **Respond to events**
  - Periodic tasks: ADC, DAC, control systems
  - Aperiodic tasks: input, output, alarms

I/O SYNCHRONIZATION

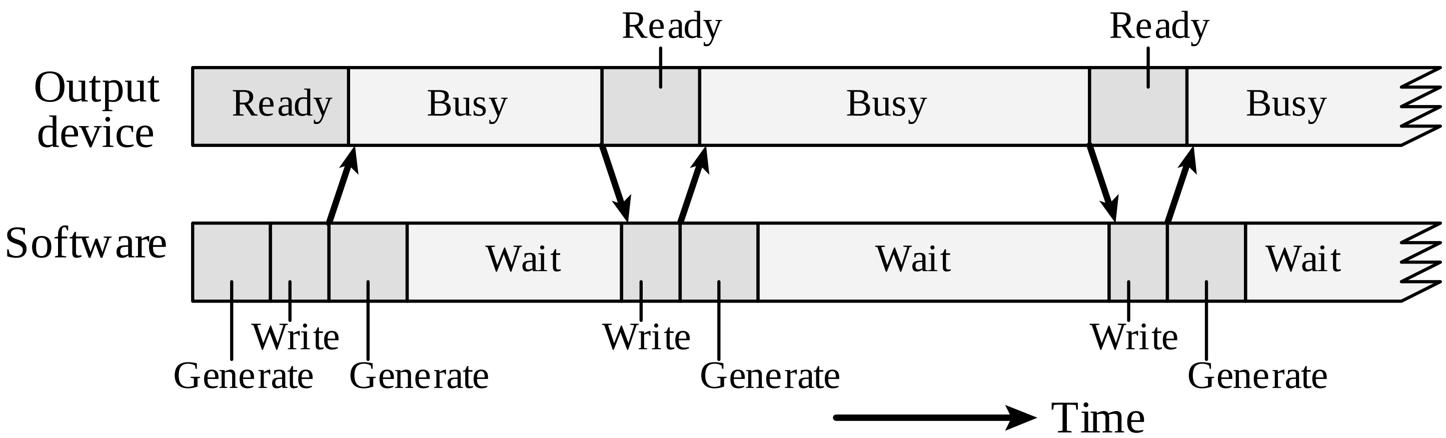


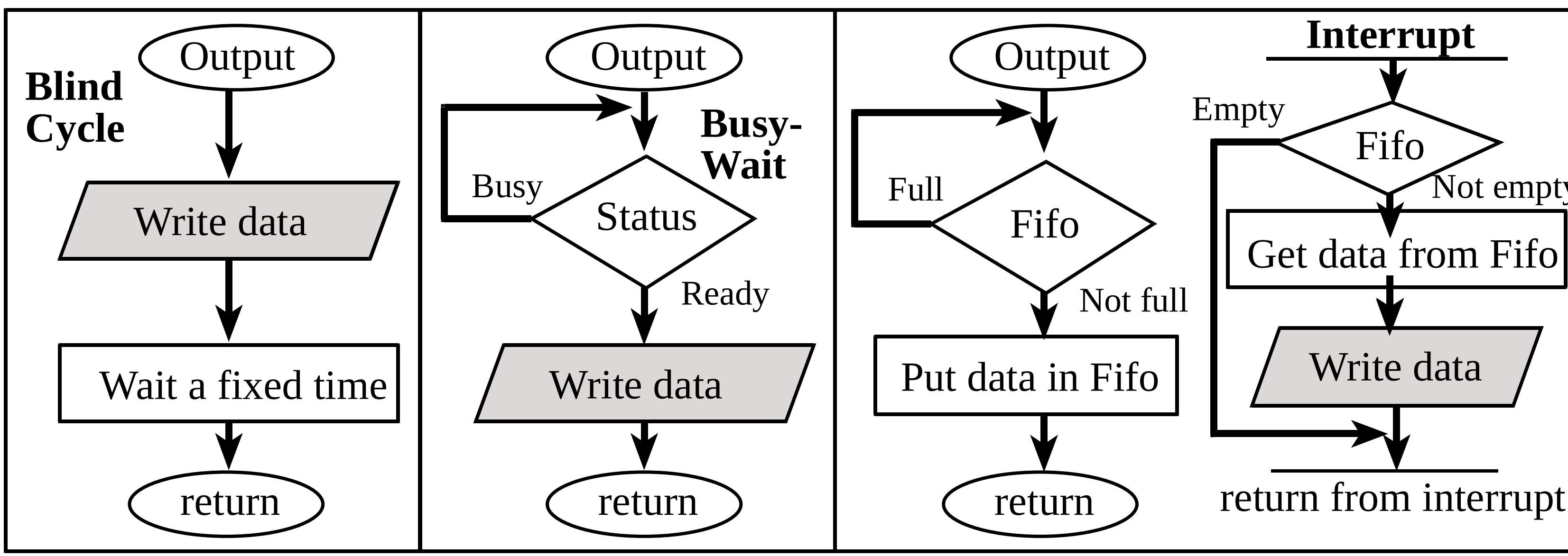
INPUT SYNCHRONIZATION





OUTPUT SYNCHRONIZATION





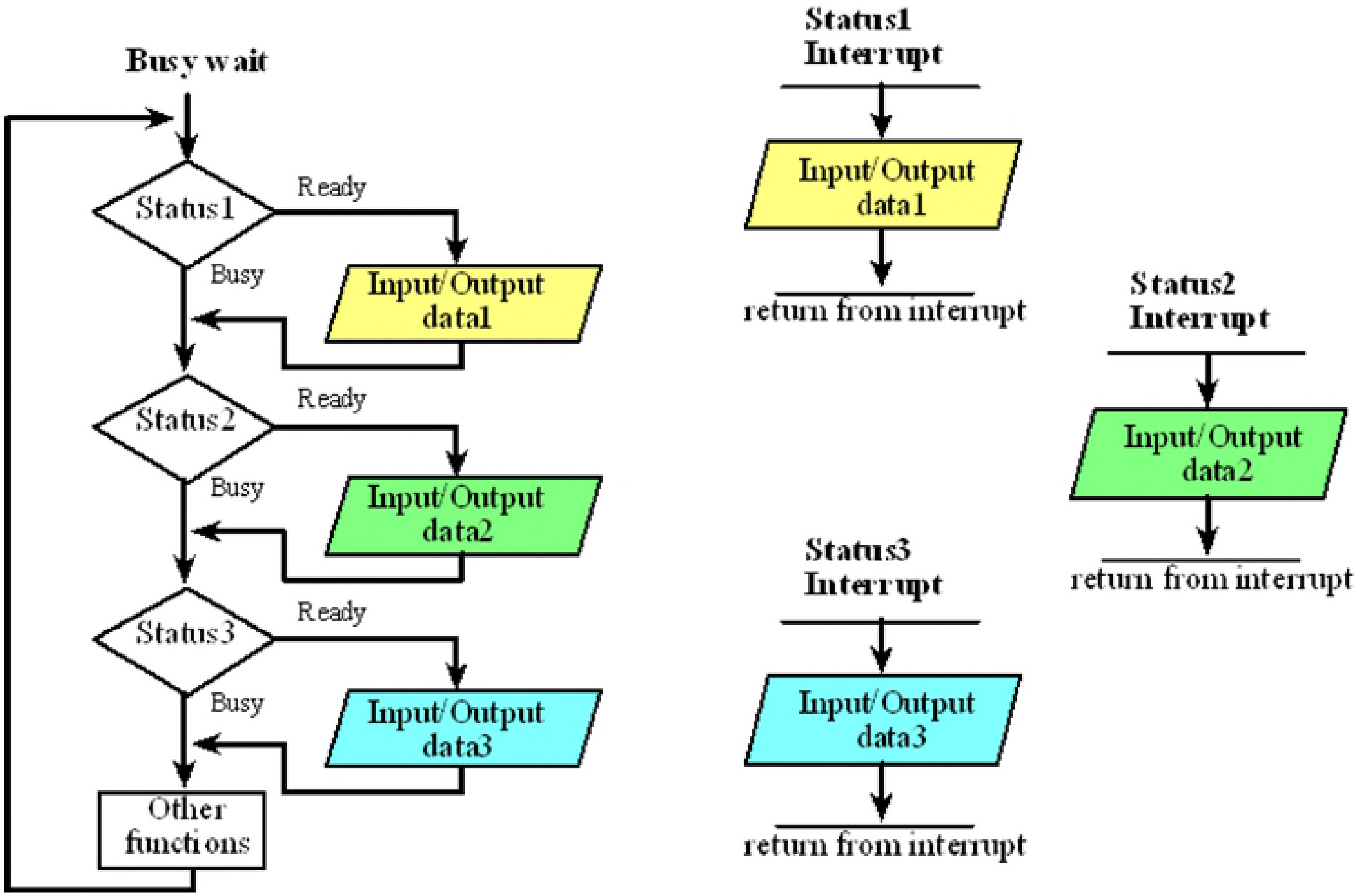
Busy-Wait Conditions

- Predictable
- Simple I/O
- Fixed load
- Dedicated, single thread
- Single process
- Nothing else to do (nothing else you can do)

Multiple Devices

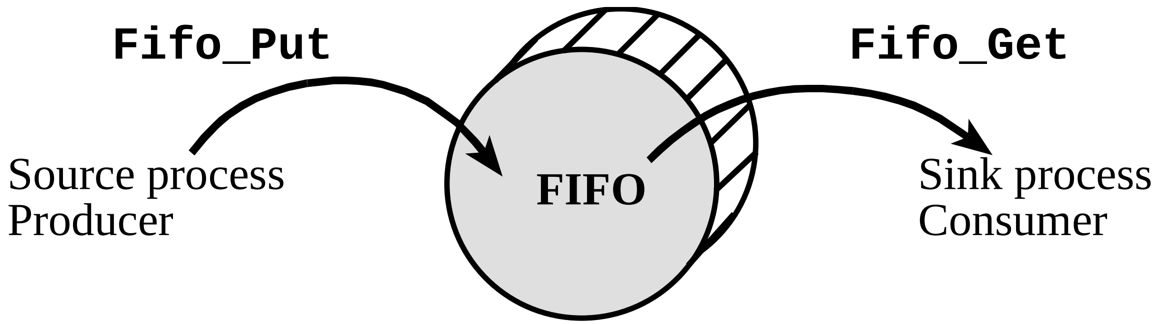
busy-wait synchronization

Bandwidth can be improved by establishing concurrent I/O operations



Little's Theorem[1]

$$N = \lambda R$$



- λ is the average arrival rate in packets per second (pps)
- R is the average response time of a packet
  - Waiting time in the FIFO plus
  - Time to be processed by the consumer
- N is the average number of packets in the system
  - N-1 stored in the FIFO plus
  - 1 being processed by the consumer

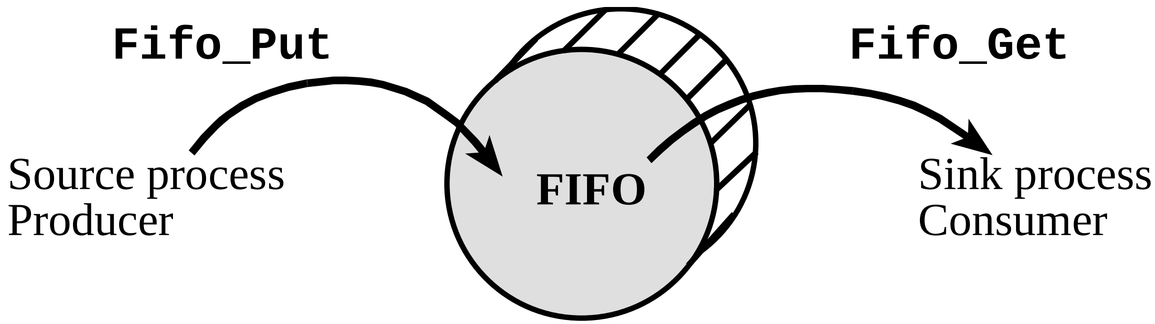
How is this different from real time?

How do you prove Little's Thm?

[1] https://en.wikipedia.org/wiki/Little%27s_law

Using Little's Theorem

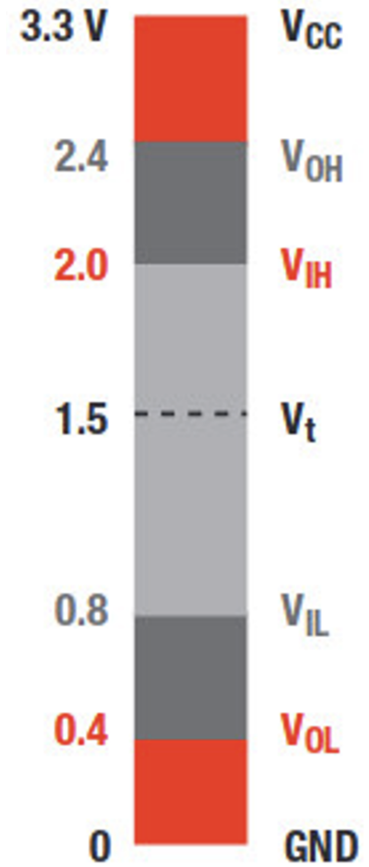
$$N = \lambda R$$



- S be the average service time of a packet
- C is the average service rate (C=1/S)
- Stable if average arrival rate < average service rate
  - $\lambda < C$
- If stable (FIFO never fills), we estimate response time
  - Measure average arrival rate λ
  - Measure average FIFO size M, N=M+1
  - Calculate $R = N/\lambda$

Digital Logic Families

- **Logic voltage level definitions**
  - V_{IL} ⟹ voltage below which an input is considered logic low
  - V_{IH} ⟹ voltage above which an input is considered logic high
  - V_{OH} ⟹ output voltage for a logic high (current less than I_{OH})
  - V_{OL} ⟹ output voltage for a logic low (current less than I_{OL})

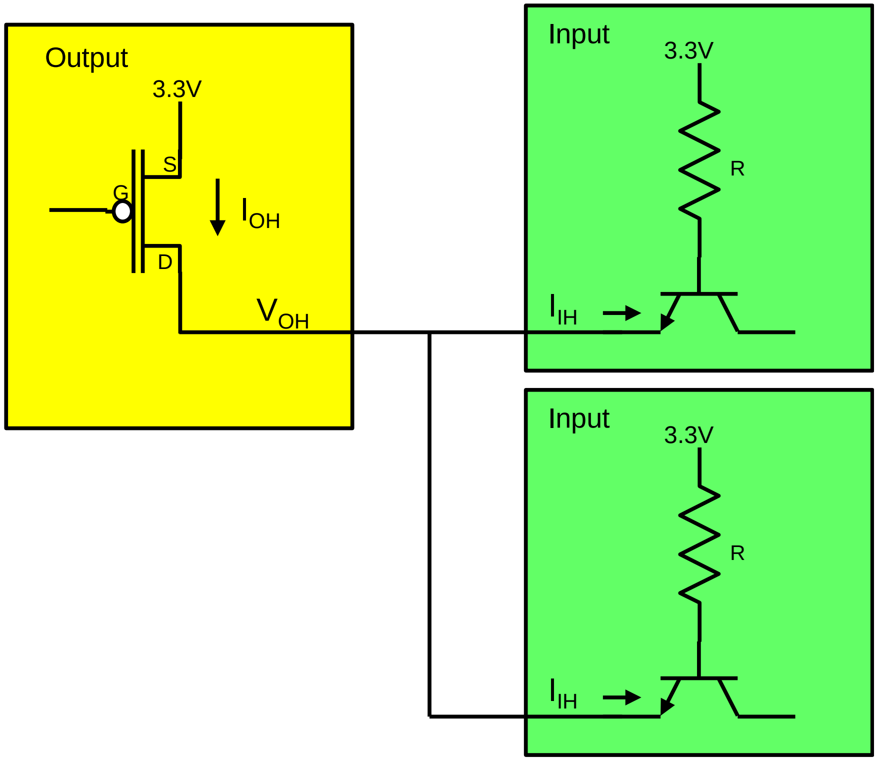


$V_{OL} < V_{IL}$ $V_{OH} > V_{IH}$

Digital Logic (MOS-TTL(BJT))

Output High State

- The device providing (driving) the output is capable of sourcing a maximum current to that output
  - I_{OH}
- Each input device receiving (sinking) the output as an input requires a maximum current:
  - I_{IH}

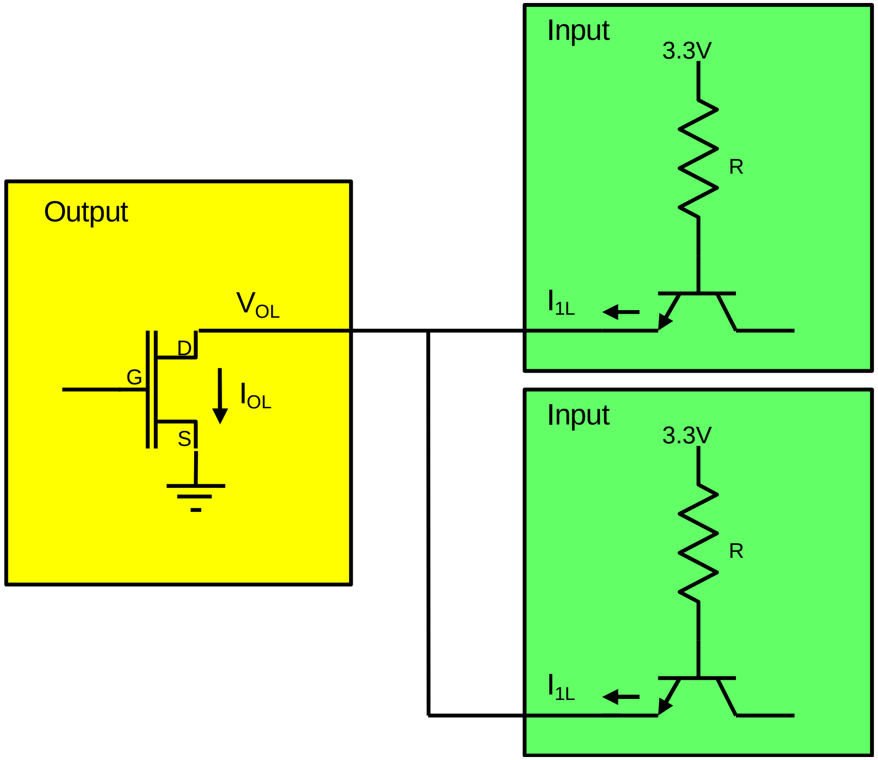


Family	*Example*	I_{OH}	I_{OL}	I_{IH}	I_{IL}
Standard TTL	7404	0.4 mA	16 mA	40 µA	1.6 mA

Digital Logic (MOS-TTL(BJT))

Output Low State

- The device providing (driving) the output is capable of receiving (sinking) a maximum current through the output
  - I_{OL}
- Each input device sources a maximum current:
  - I_{IL}

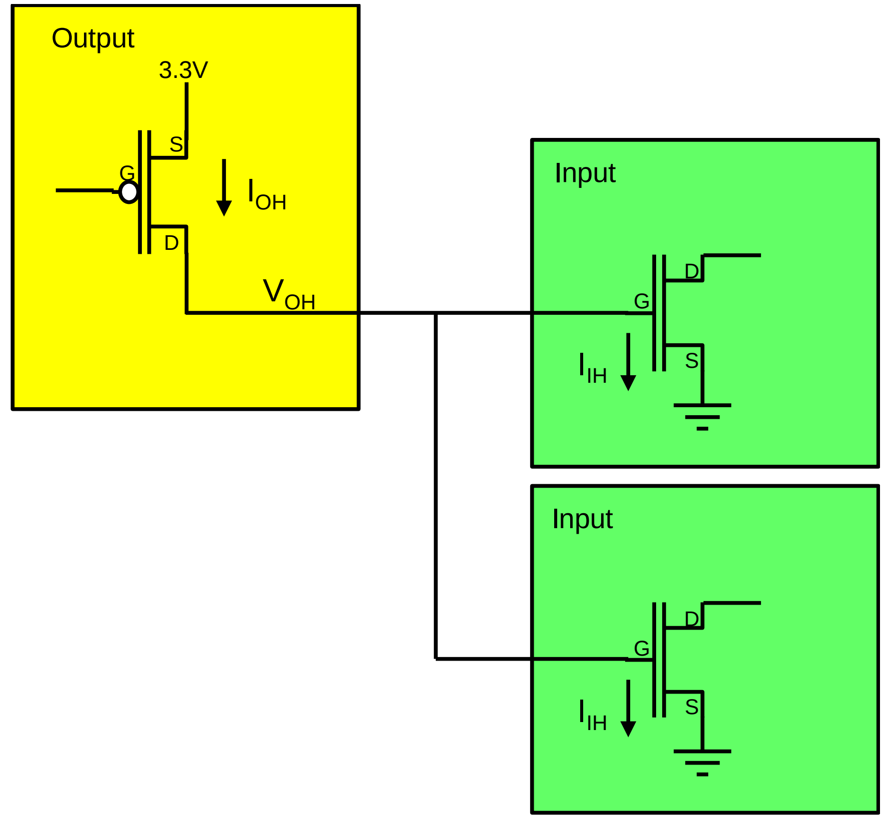


Family	*Example*	I_{OH}	I_{OL}	I_{IH}	I_{IL}
Standard TTL	7404	0.4 mA	16 mA	40 µA	1.6 mA

Digital Logic (MOS-MOS)

Output High State

- The device providing (driving) the output is capable of sourcing a maximum current to that output
  - I_{OH}
- Each input device receiving (sinking) the output as an input requires a maximum current:
  - I_{IH}

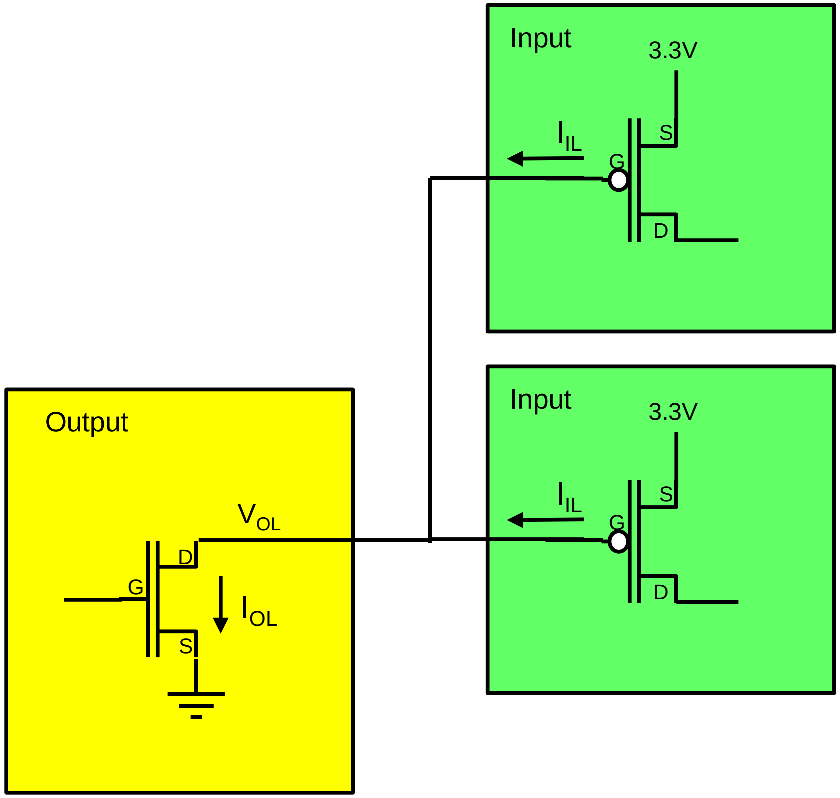


Family	*Example*	I_{OH}	I_{OL}	I_{IH}	I_{IL}
High Speed CMOS	74HC04	4 mA	4 mA	1 µA	1 µA
Adv High Speed CMOS	74AHC04	4 mA	4 mA	1 µA	1 µA

Digital Logic (MOS-MOS)

Output Low State

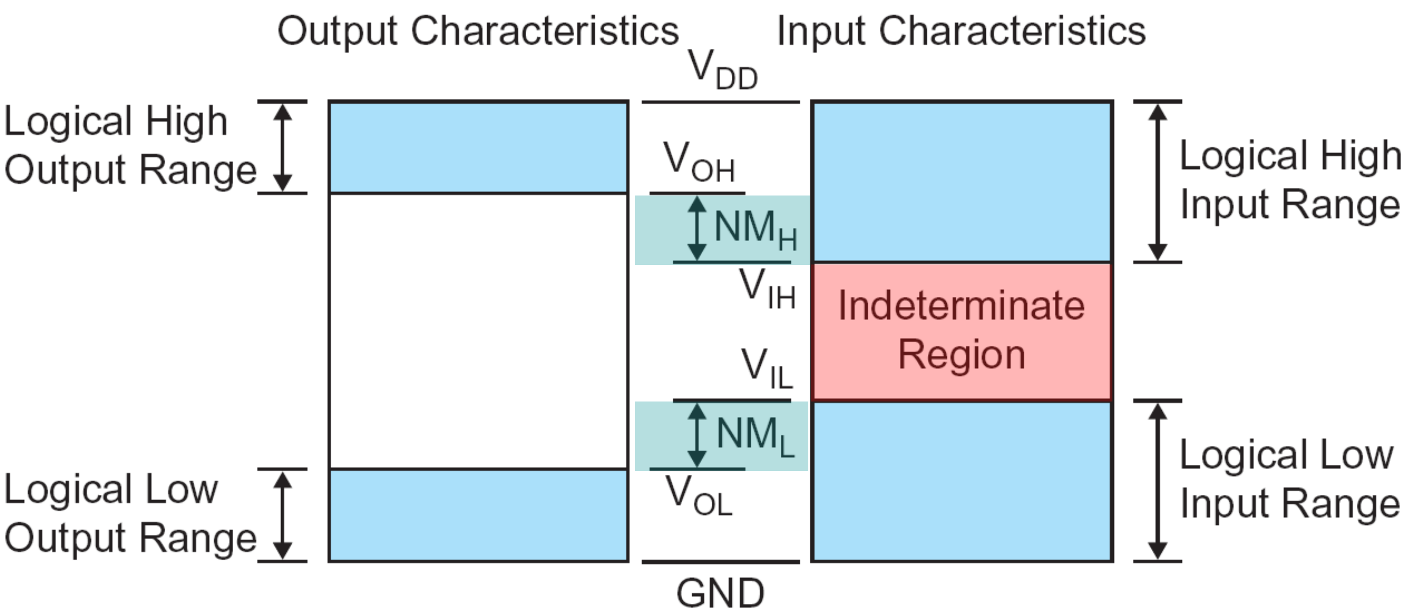
- The device providing (driving) the output is capable of receiving (sinking) a maximum current through the output
  - I_{OL}
- Each input device sources a maximum current:
  - I_{IL}

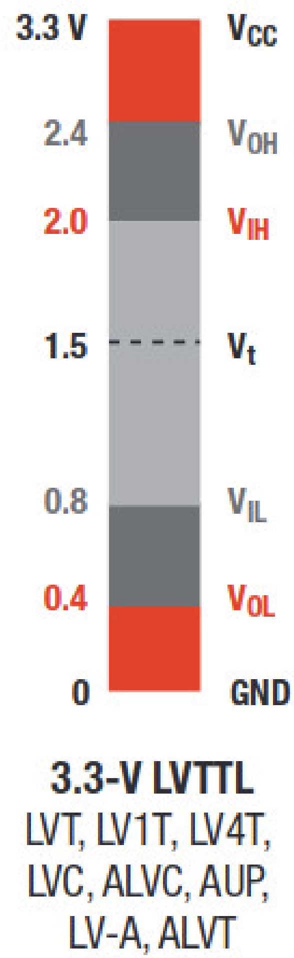


Family	*Example*	I_{OH}	I_{OL}	I_{IH}	I_{IL}
High Speed CMOS	74HC04	4 mA	4 mA	1 µA	1 µA
Adv High Speed CMOS	74AHC04	4 mA	4 mA	1 µA	1 µA

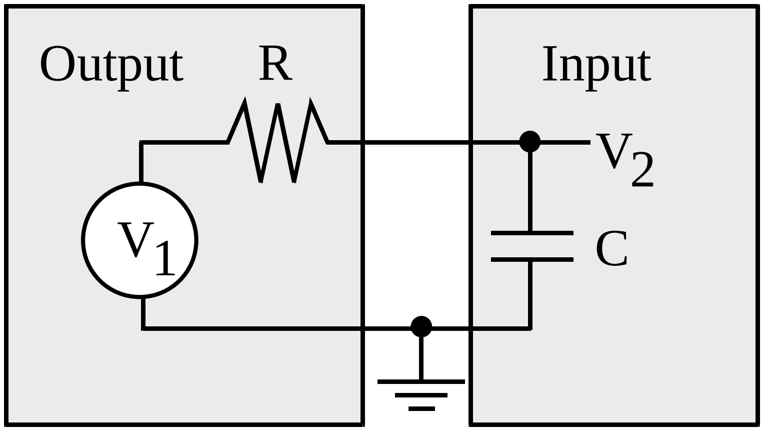
Noise Margins

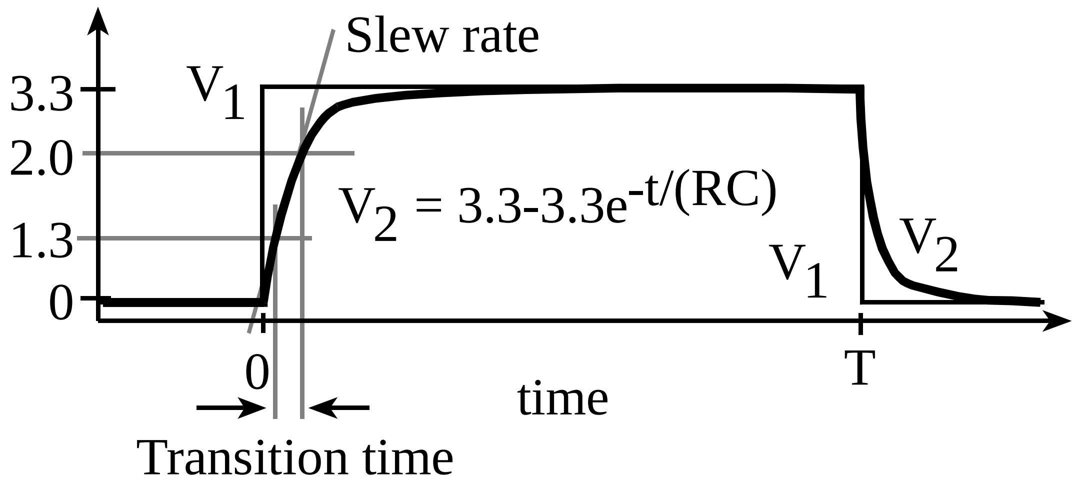
- **How much noise can a gate input see before it does not recognize the in**





Capacitance





- Make it run faster
  - Decrease R,C
  - Increase I, P
- Make it less noisy
  - Decrease slew rate

Capacitance loading is an important factor when interfacing CMOS devices

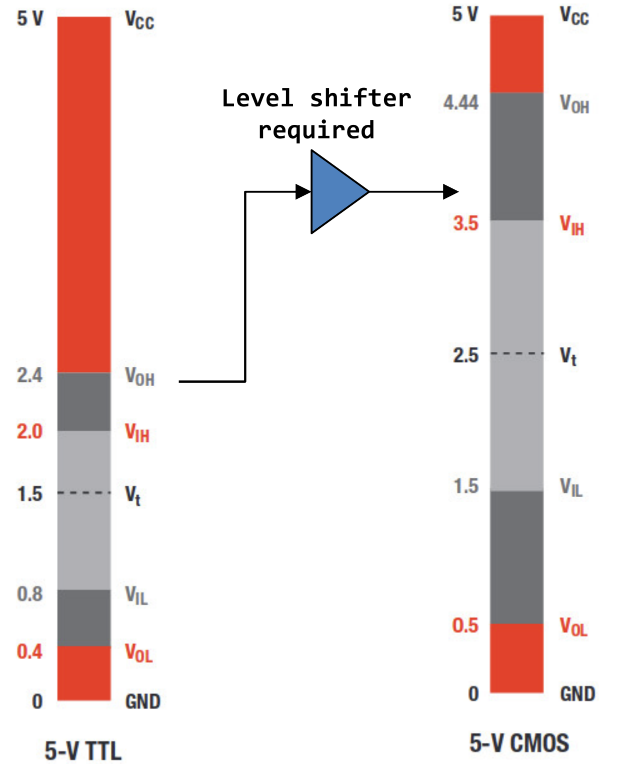
Take Aways

- **R - Resistive loads require current amplification**
- **L - Inductive loads produce back EMF**
  - Snubber diode(flyback diode, clamp diode, etc)
- **C - Capacitive loads will slow down dV/dt**
  - Time constant, $\tau \approx R*C$
- **Make it go faster by**
  - Decreasing R and C
  - Increasing I

Speed is directly proportional to power

Summary

- **Logic Families**
- **Bipolar Transistors**
  - Saturated Mode
  - Current gain h_{fe}
  - Activation V_{be}
  - Output V_{ce} I_{ce}
- **Inductance**
  - Can blow up your laptop
- **Capacitance**
  - Slows down signals
  - Reduces EM emissions

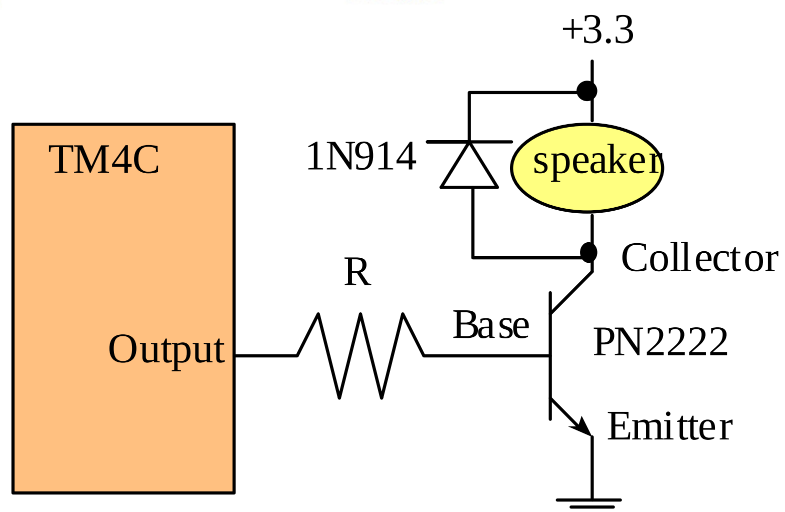


$V_{OH} > V_{IH}$

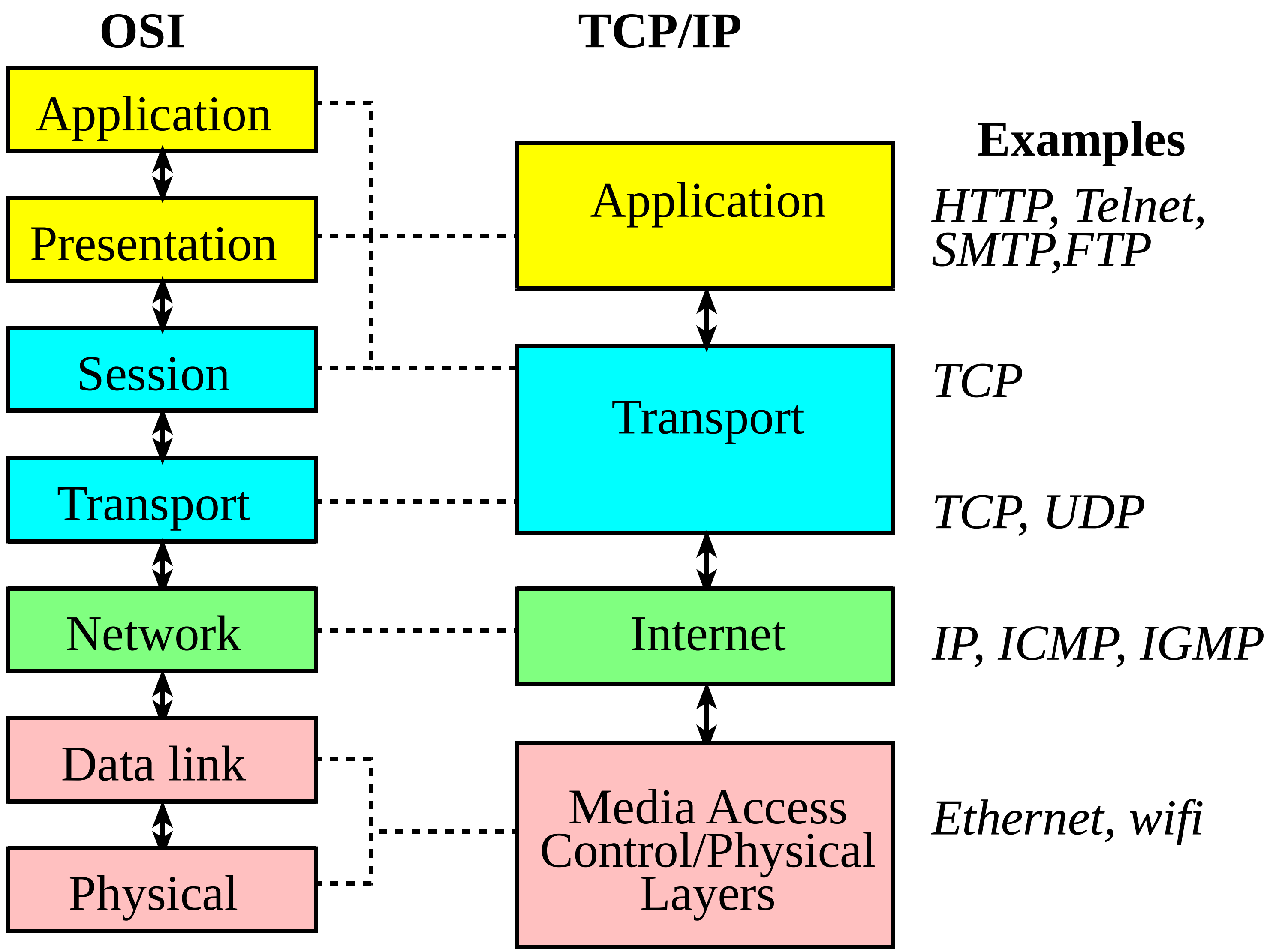
$I_{OH} > I_{IH}$

$V_{OL} < V_{IL}$

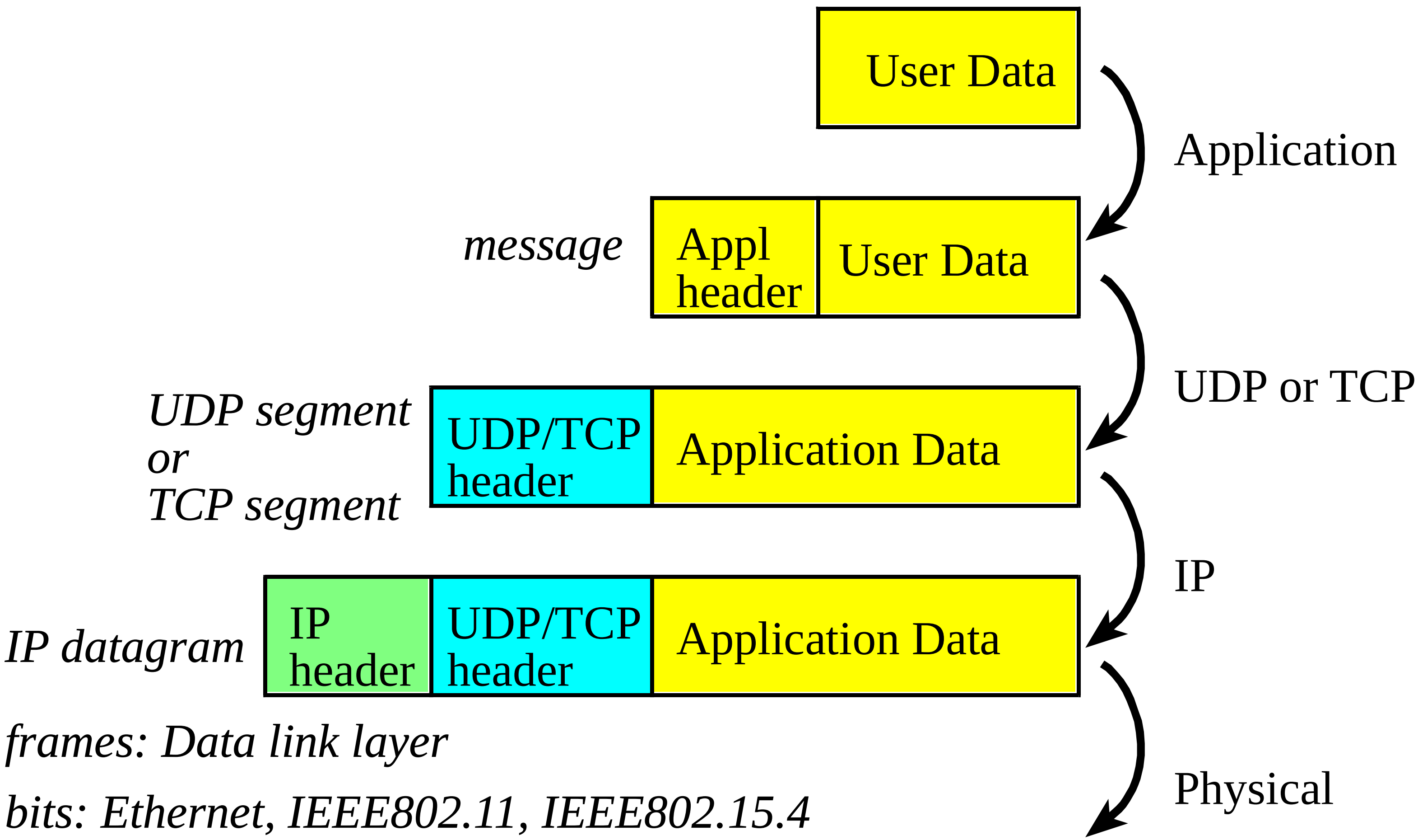
$I_{OL} > I_{IL}$



Layers


OSI
TCP/IP
Application
Presentation
Session
Transport
Network
Data link
Physical
Examples
Application
HTTP, Telnet, SMTP,FTP
Transport
TCP
TCP, UDP
Internet
IP, ICMP, IGMP
Media Access Control/Physical Layers
Ethernet, wifi

Layered Message Protocol

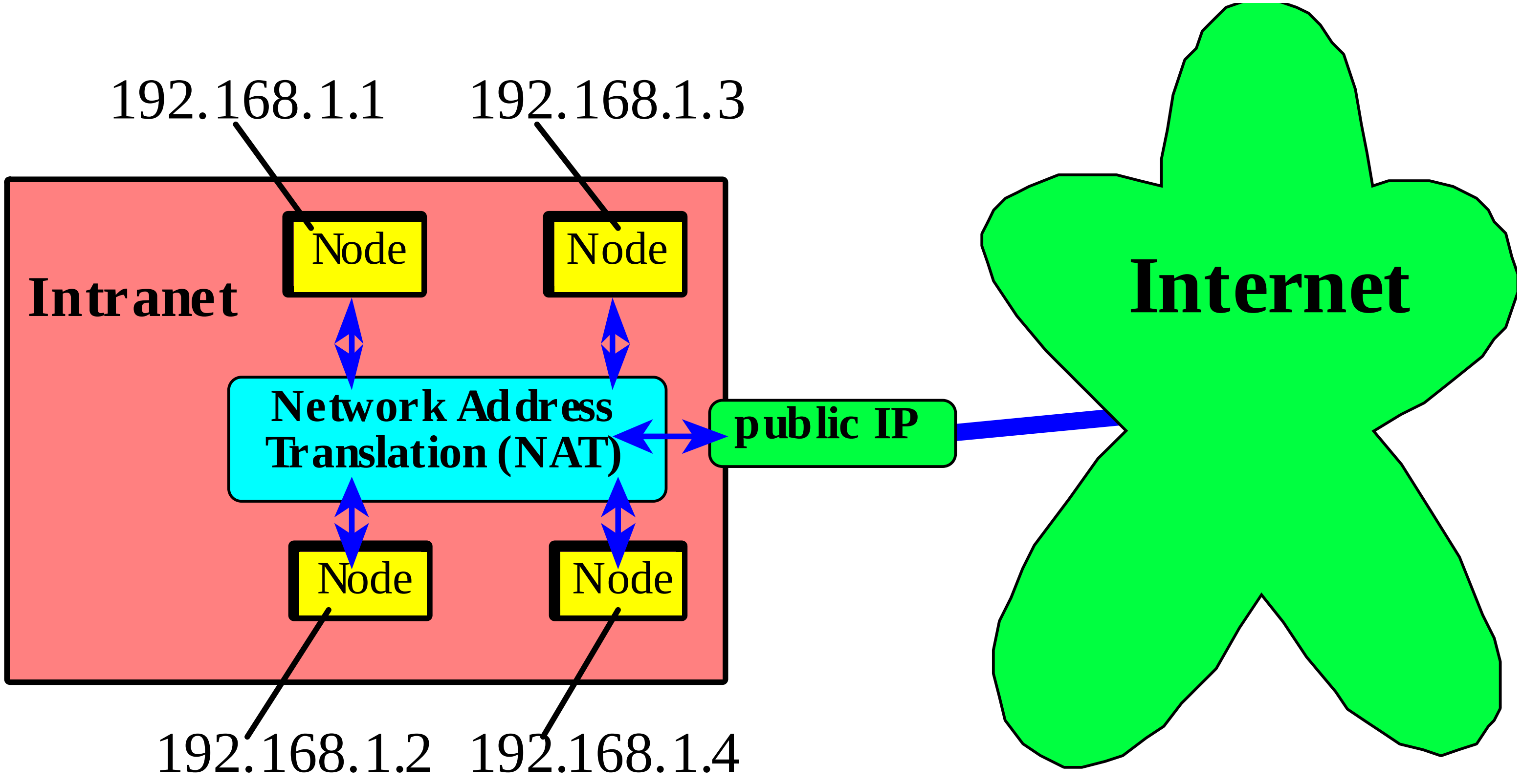


Sequence of events

- Connect to AP
- DNS
- Open a Socket
- Send TCP
- Receive TCP
- Close Socket

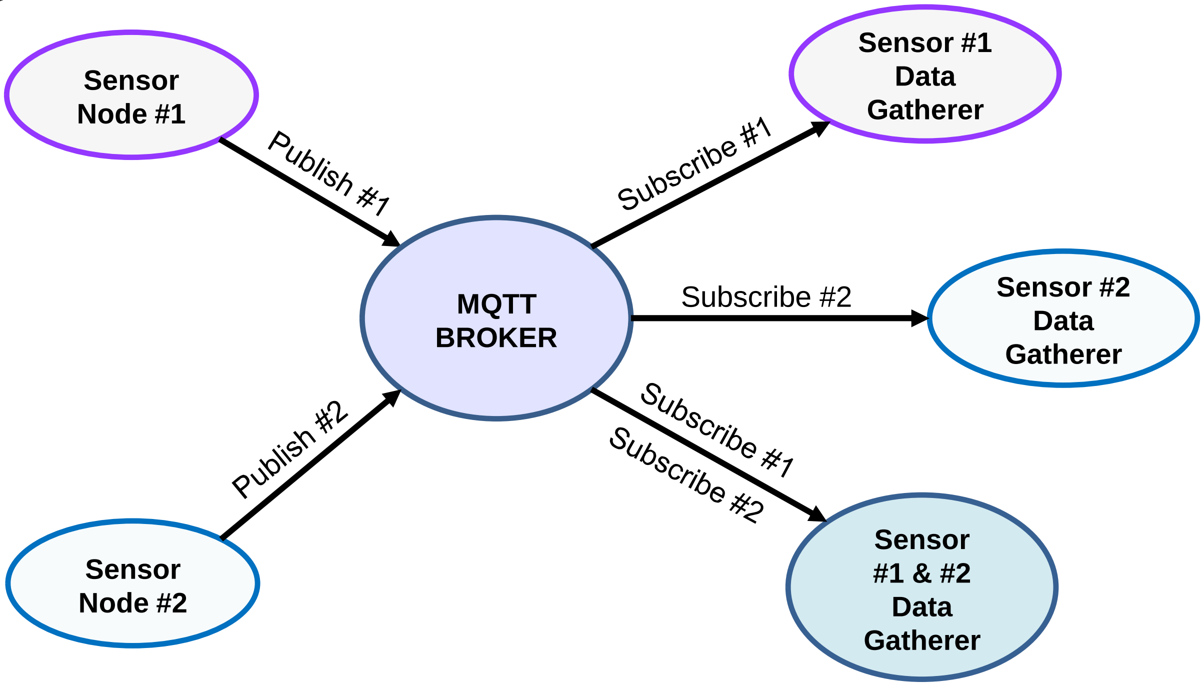
IP Addresses, IPv4

Domain Name Service (DNS),



MQTT

- MQTT is a Publisher-Subscriber Protocol
  - A publisher publishes messages on a topic and a subscriber must subscribe to that topic to view the message.
- MQTT **requires the use of a central Broker** as shown in the diagram below:



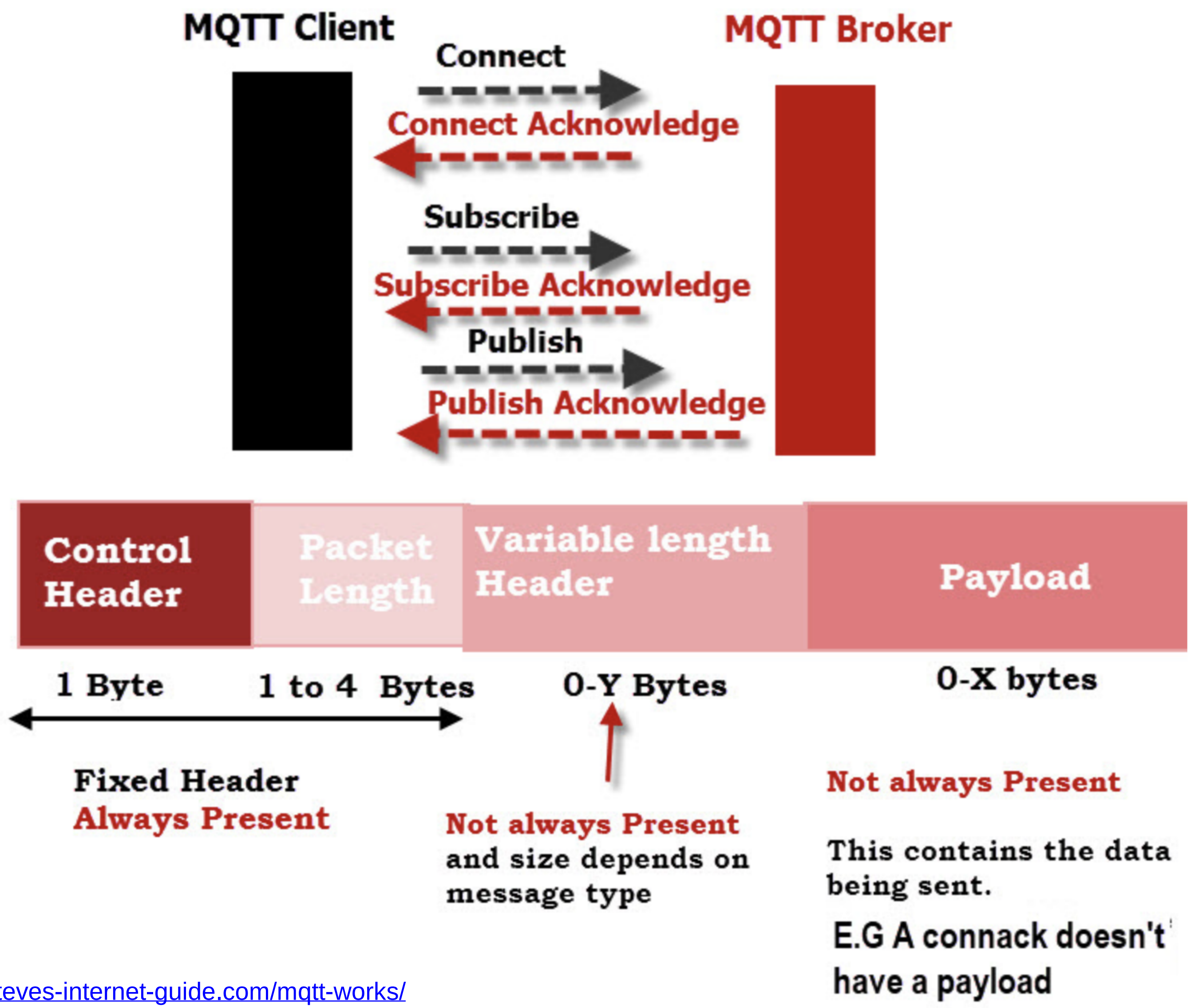
http://www.steves-internet-guide.com/mqtt-works/

MQTT Details

- Subscribers do not have addresses like in email systems, and messages are not sent directly to subscribers .
- Messages are published to a broker for a particular topic.
- The job of an MQTT broker is to filter messages based on the topic, and then distribute them to subscribers.
- A client can receive these messages by subscribing to that topic on the <u>same</u> broker.
- There is no direct connection between a publisher and subscriber.
- All clients can publish (broadcast) and subscribe (receive).
- MQTT brokers do not normally store messages

http://www.steves-internet-guide.com/mqtt-works/

MQTT Protocol



http://www.steves-internet-guide.com/mqtt-works/

MQTT over TCP

- MQTT uses TCP/IP to connect to the broker.
- TCP is a connection orientated protocol with error correction and guarantees that packets are received in order.
- You can consider a TCP/IP connection to be like a telephone connection.
  - Once a telephone connection is established you can talk over it until one party hangs up.
- Most MQTT clients will connect to the broker and remain connected even if they aren’t sending data.
- Connections are acknowledged by the broker using a CONNACK message about once a minute

http://www.steves-internet-guide.com/mqtt-works/

MQTT over WebSocket

- WebSocket is a computer communications protocol, providing full-duplex communication channels over a single TCP/IP connection.
- It is closely associated with HTTP as it uses HTTP for the initial connection establishment..
- The client and server connect using HTTP and then negotiate a connection upgrade to WebSocket, the connection then switches from http to WebSocket.
- The client and server can now exchange full duplex binary data over the connection.

http://www.steves-internet-guide.com/mqtt-websockets/