



Internet of Things

Senior Design Project Course

Introduction – Part 1

Lecturer: Avesta Sasan

University of California Davis
Fall 2023

Required Reading:

- F. Samie, L. Bauer and J. Henkel, "**IoT technologies for embedded computing: A survey**," *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Pittsburgh, PA, 2016, pp. 1-10.
 - <http://ieeexplore.ieee.org/document/7750968/>
- No Review is required for this paper!

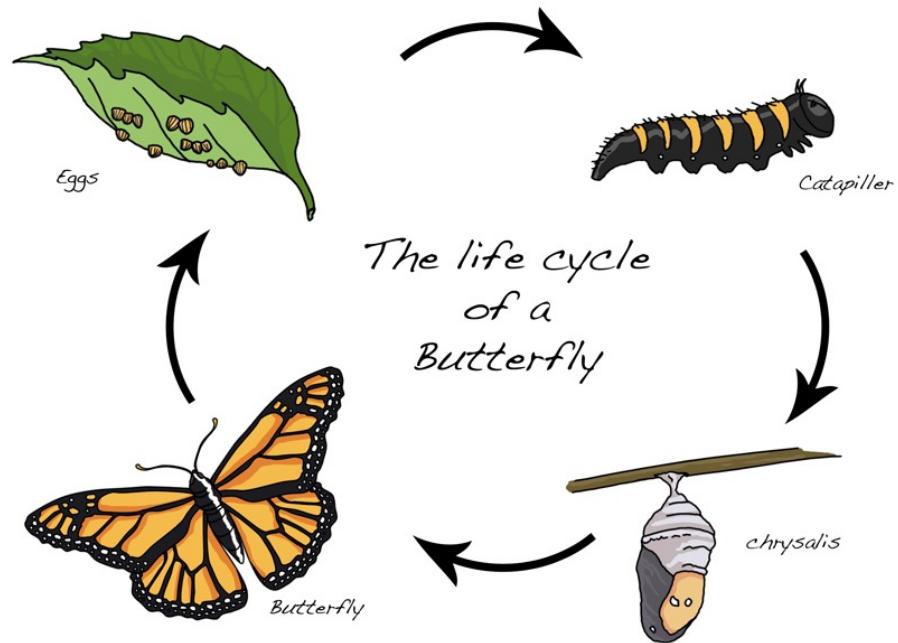
Flow of Data in Internet of Things (Review)



Image source: <http://www.cchc.cl/informacion-a-la-comunidad/industria-de-la-construccion/personaje/>

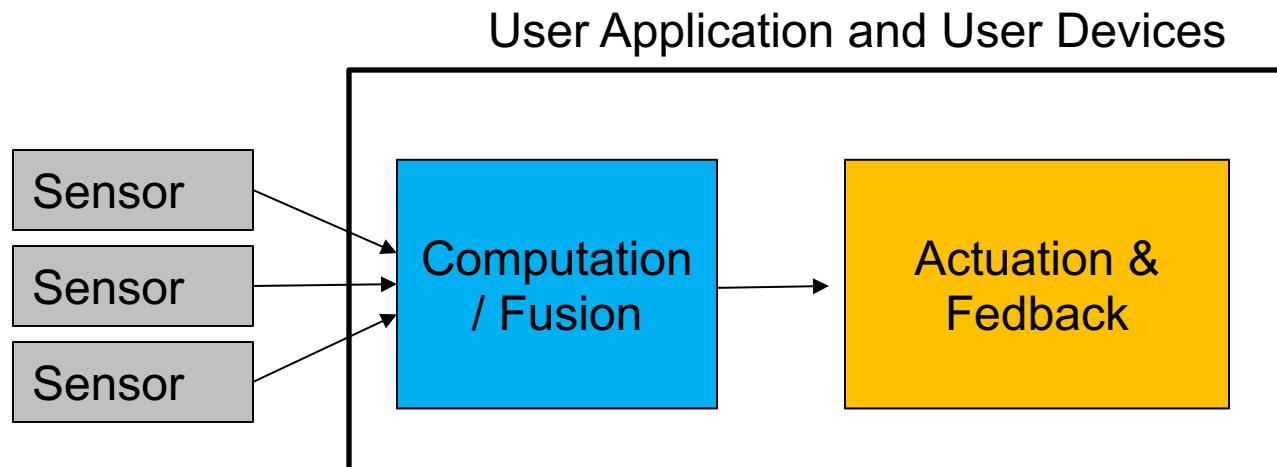
Life Cycle of Data at Edge

- The edge solutions could be
 - Memoryless & Short Cycle
 - With memory & Short Cycle
 - With memory and Long Cycle
- What does it mean?



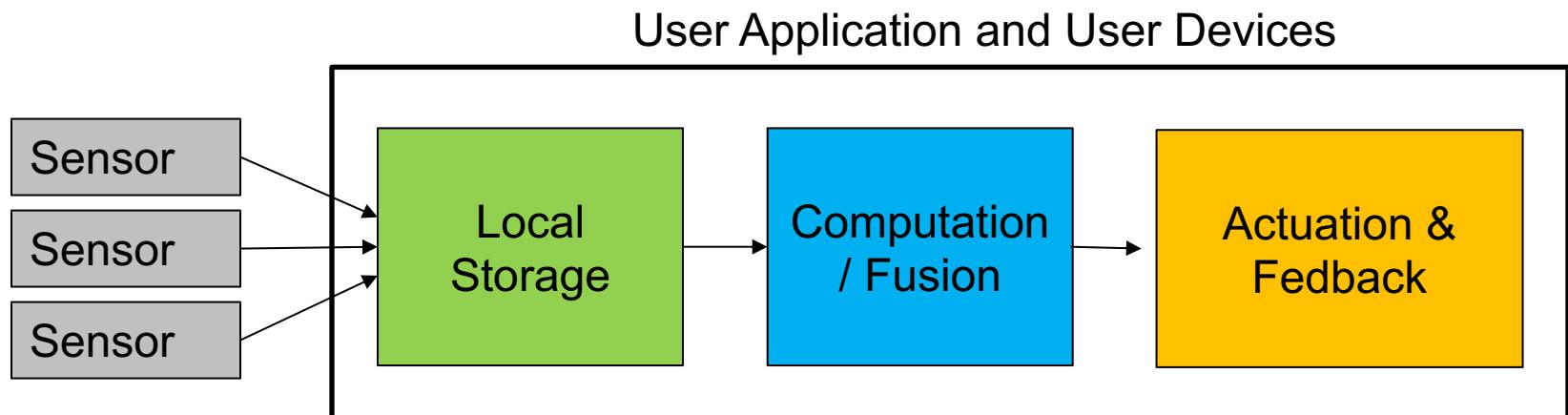
Memoryless and Short Cycled

- Data is sensed
- Sensed data is fused and is subjected to some form of computation
- Based on the computation result, an action is actuated!
- Data is then disregarded!



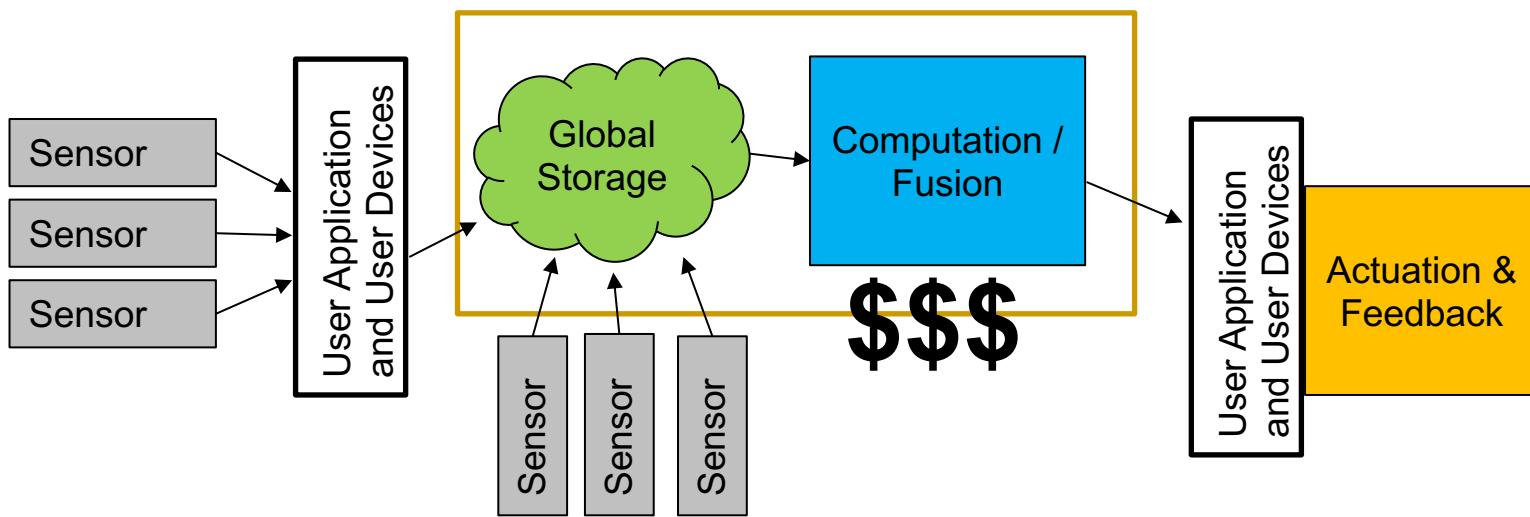
With Memory and Short Cycle

- Data is sensed in regular/irregular intervals
- Sensed Data is stored locally
- Sensed data is fused and is subjected to some form of computation
- Based on the computation result, an action is actuated!



Long Cycle (Most Desired)

- Local sensing, processing and actuation happens as before!
- Processed or sample data is communicated to the cloud service(s).
- Sensed data from many sources are aggregated, fused and analyzed to extract knowledge.
 - Events are detected, trends are detected, predictions are made
- Feedback is sent to selected edge nodes, improving services
- Knowledge and history is stored for improving or enabling future services.



IoT Big Data

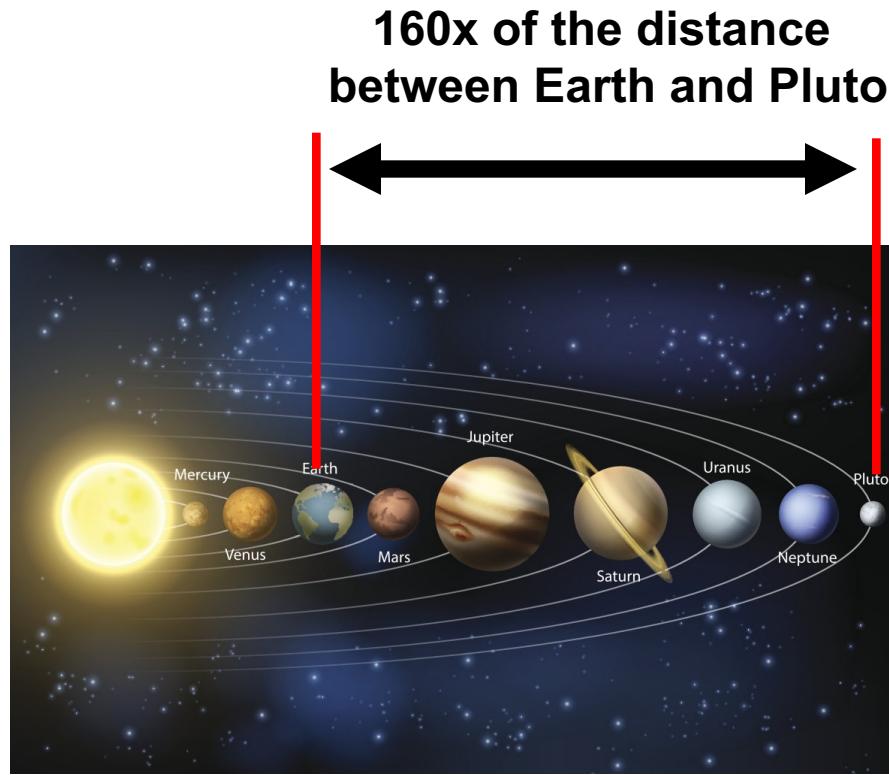
- **A Poor man definition:** An enormous amount of data, including streams of data, audio, or video, will be generated from IoT devices
- Computation and processing of data should be migrated to different underlying computing layers in the IoT chain
- Available layers are **Device, Gateway, Fog, Cloud**
- When deciding where computation take place, one should consider:
 - **System objectives**
 - (e.g. real-time requirement, energy efficiency, etc.)
 - **System specifications**
 - (e.g. energy consumption for data processing and data transmission on IoT device, communication bandwidth, transmission delay, etc.)

Information on Cloud!

4000
Exabyte



=

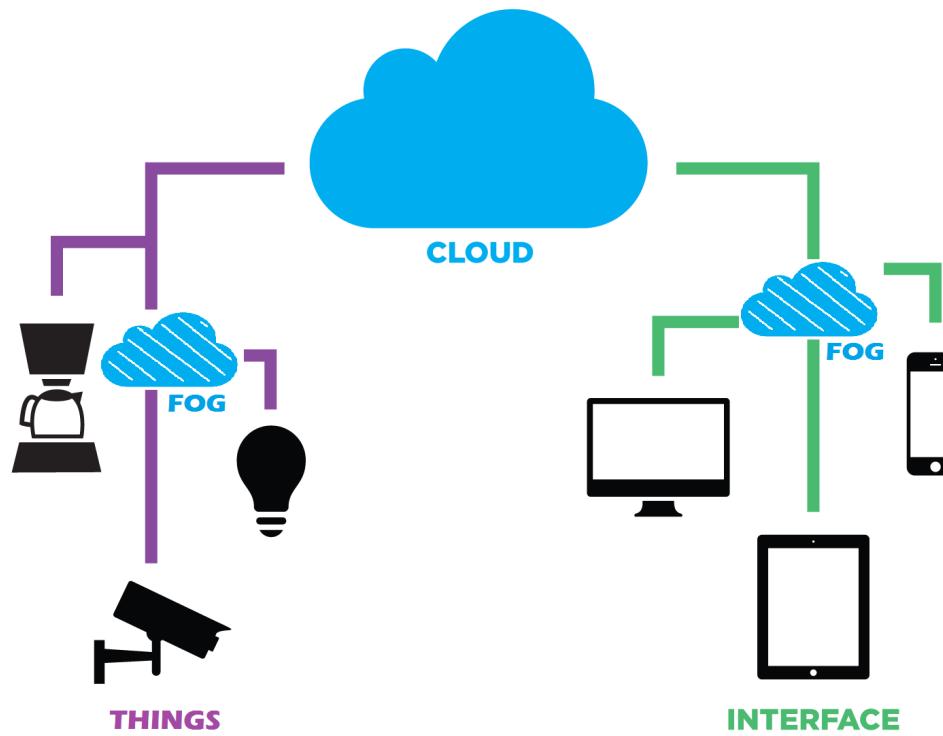


X



Fog Computing

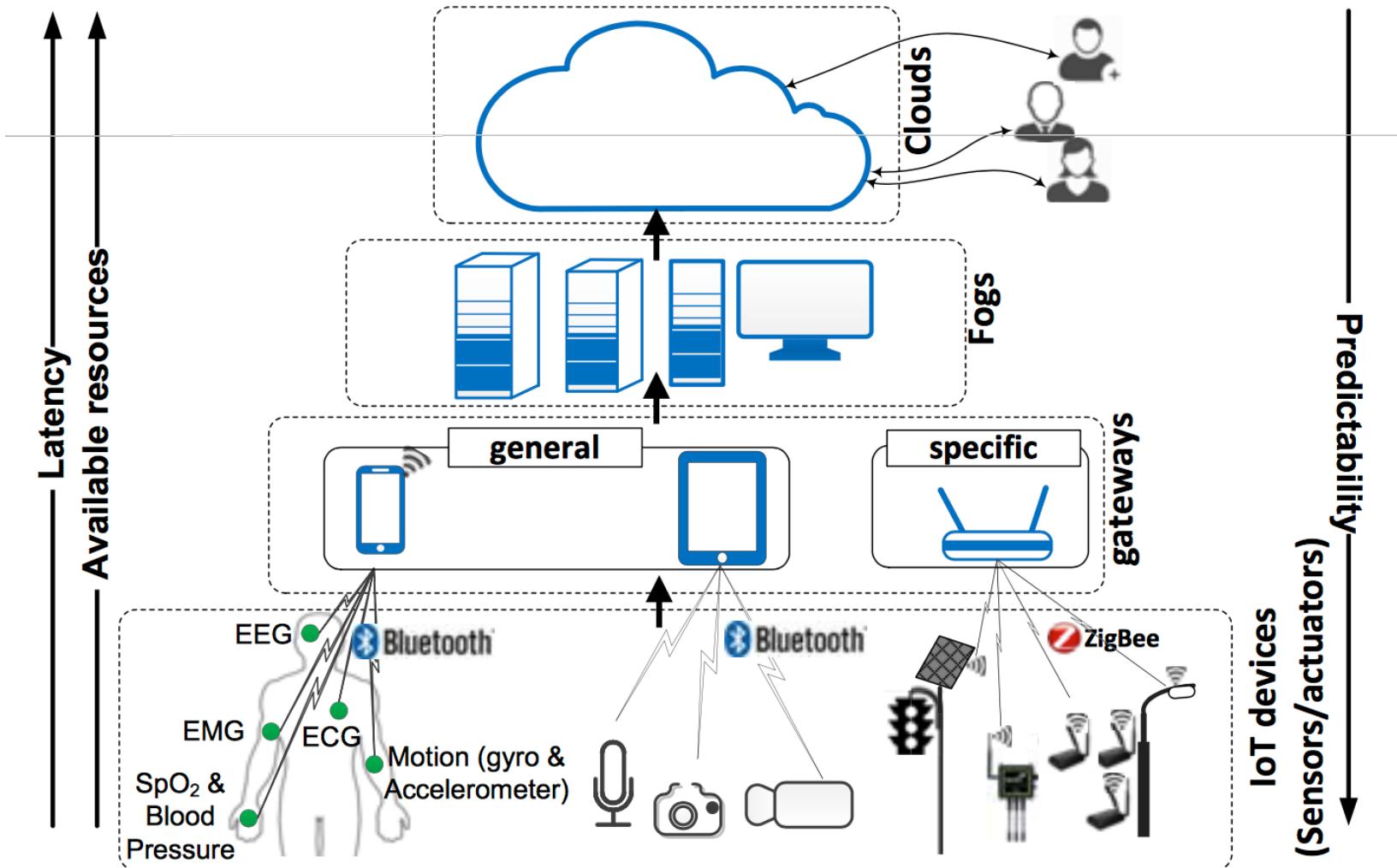
- Number of connected devices increases exponentially
- Amount of data being generated grows exponentially
- Using smart devices, as apposed to a primitive sensor
- Handling the interpretation logic rather than requiring a trip to the cloud



Fog vs Cloud Computing

- **Cloud Computing**
 - Sending data to the cloud for processing
 - A group of computers and servers interconnected through the web to form a network
- **Fog Computing**
 - A middle layer between the cloud and the hardware to enable more efficient data processing, analysis and storage
 - a distributed infrastructure
 - Reduces the amount of data which needs to be transported to the cloud
 - Make the data (to be transferred) more meaningful!
- **Fog is the cloud close to the ground**

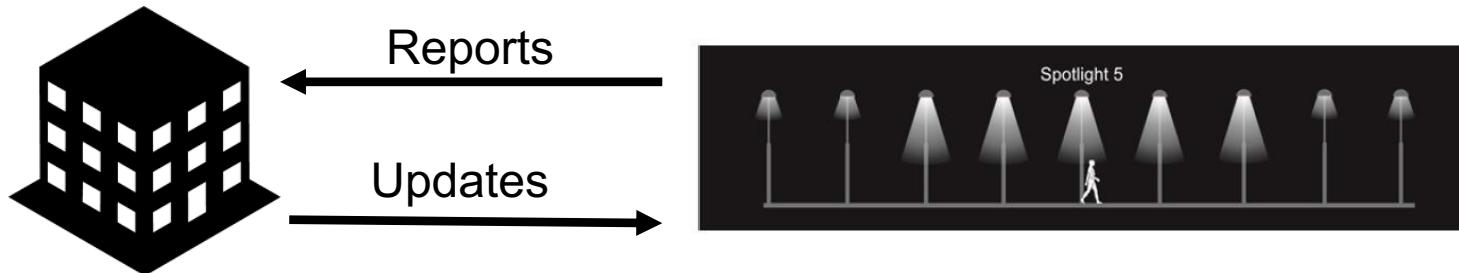
IoT Chain Computation Layers



F. Samie, L. Bauer and J. Henkel, "IoT technologies for embedded computing: A survey," *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Pittsburgh, PA, 2016, pp. 1-10.

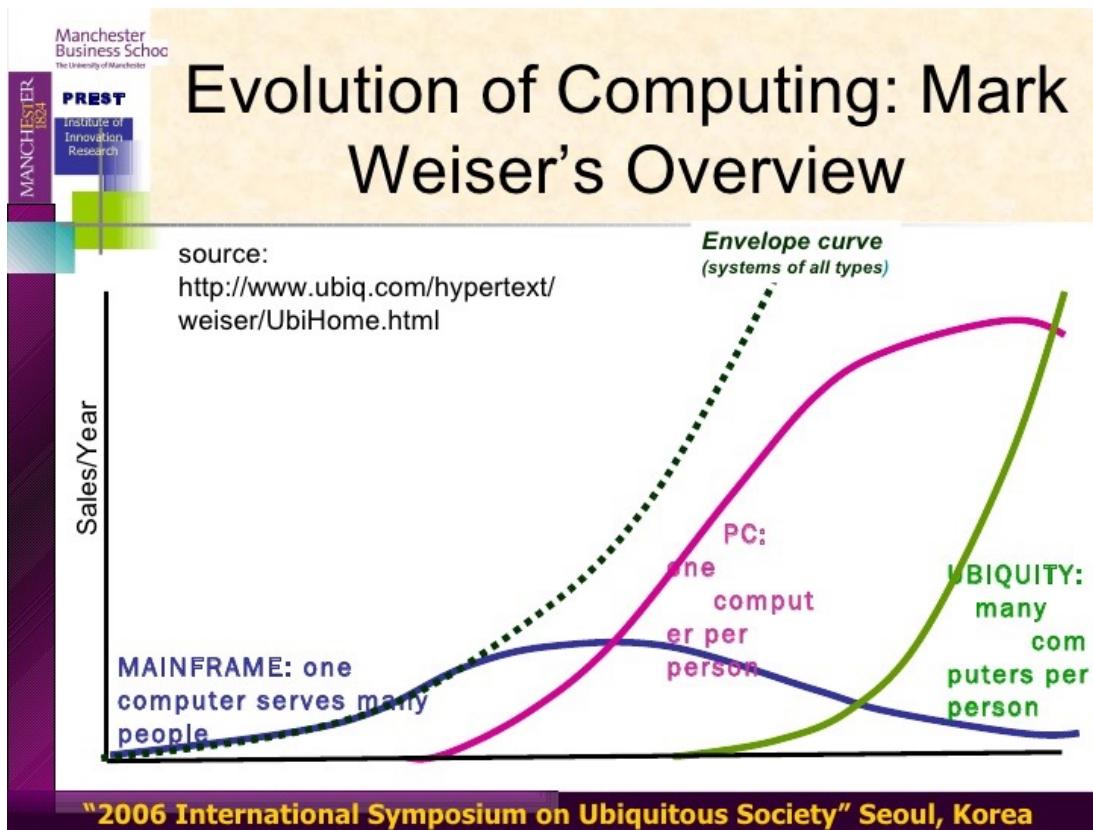
Fog Computing Example

- Smart lighting system
 - operates based on movement.
 - Sense → process data → movement detected → lights on!
 - Sense → process data → no movement → lights off!
 - data & resulting decisions are best processed at the edge
 - Energy efficiency reports and on/off logs are reported to the managing service/company
 - Reported data provides a “bigger picture” to the reporting system in the cloud
 - Knowledge is extracted
 - Managing service updates the processing nodes for better service or higher efficiency.



Ubiquitous (Pervasive) Computing

- **Mainframe:** many people share one computer
- **Personal computer:** one person uses one computer
- **Ubiquitous computing:** many computers serve each person.



Ambient Intelligence (Aml)

- Ambient intelligence paradigm builds upon pervasive computing
- Electronic environments that are sensitive and responsive to the presence of people.
- Devices work in concert to support people in carrying out their everyday life activities, tasks and rituals in an easy, natural way
- Using information and intelligence that is hidden in the network connecting these devices
- The technology disappears into our surroundings until only the user interface remains perceivable by users

Ambient Intelligence

- Characterized by systems and technologies that are:
 - **Embedded:** many networked devices are integrated into the environment
 - **Context aware:** devices can recognize you and your situational context
 - **Personalized:** devices can be tailored to your needs
 - **Adaptive:** can change in response to you
 - **Anticipatory:** can anticipate your desires without conscious mediation



F. Samie, L. Bauer and J. Henkel, "IoT technologies for embedded computing: A survey," *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Pittsburgh, PA, 2016, pp. 1-10.

Relation between #Services and #Devices

- The relation between the # of devices and # of service and applications
 - **One-to-One:** One IoT device per service
 - E.g. healthcare monitoring that captures real time bio-signals
 - **One-to-many:** One IoT device provides multiple services.
 - E.g. smart watch with multiple sensors to track user's physical activity, heart rate, location
 - **Many-to-one:** physically distributed devices providing a single service.
 - E.g. distributed smart cameras for video surveillance
 - Should be optimized with respect to
 - high communication rate between devices
 - large amount of redundancy
 - **Many-to-many:** multiple devices shared between multiple applications and services
 - E.g. Smart Citizen where multiple IoT embedded devices are geographically distributed to gather information for the applications that report temperature, humidity, noise level, and air pollution

Where to Compute?

- **Device centric:**
 - microcontroller in an IoT device can be exploited to perform the computation.
 - challenges are
 - scarce resources on IoT devices
 - runtime decision
- **Gateway centric**
 - gateway devices which are used to settle the heterogeneity between different networks and Internet usually have more computational power
 - This scheme has been used for medical and healthcare monitoring application
 - challenge
 - guarantee the availability and deadline constraints

Where to Compute?

■ **Fog centric**

- Fogs provide more computational power compared to IoT embedded and gateway devices and have less latency compared to the cloud servers

■ **Cloud centric**

- massive data storage volume, huge processing resources
- challenges
 - Size of data (big data)
 - scalability
 - high energy cost
 - latency
 - bandwidth
 - availability

IoT Companies

■ **SoC era (2000-2016)**

- larger but fewer companies
- integration of many components
- Chip design
- Focus on developing new IP
- More general purpose
 - Example: Snapdragon → Qualcomm

■ **IoT era (2014 – 202X)**

- Smaller size but larger number of companies
- Integration of few necessary components
- System design
- Focus on usage of existing IP
- Specific purpose



Internet of Things

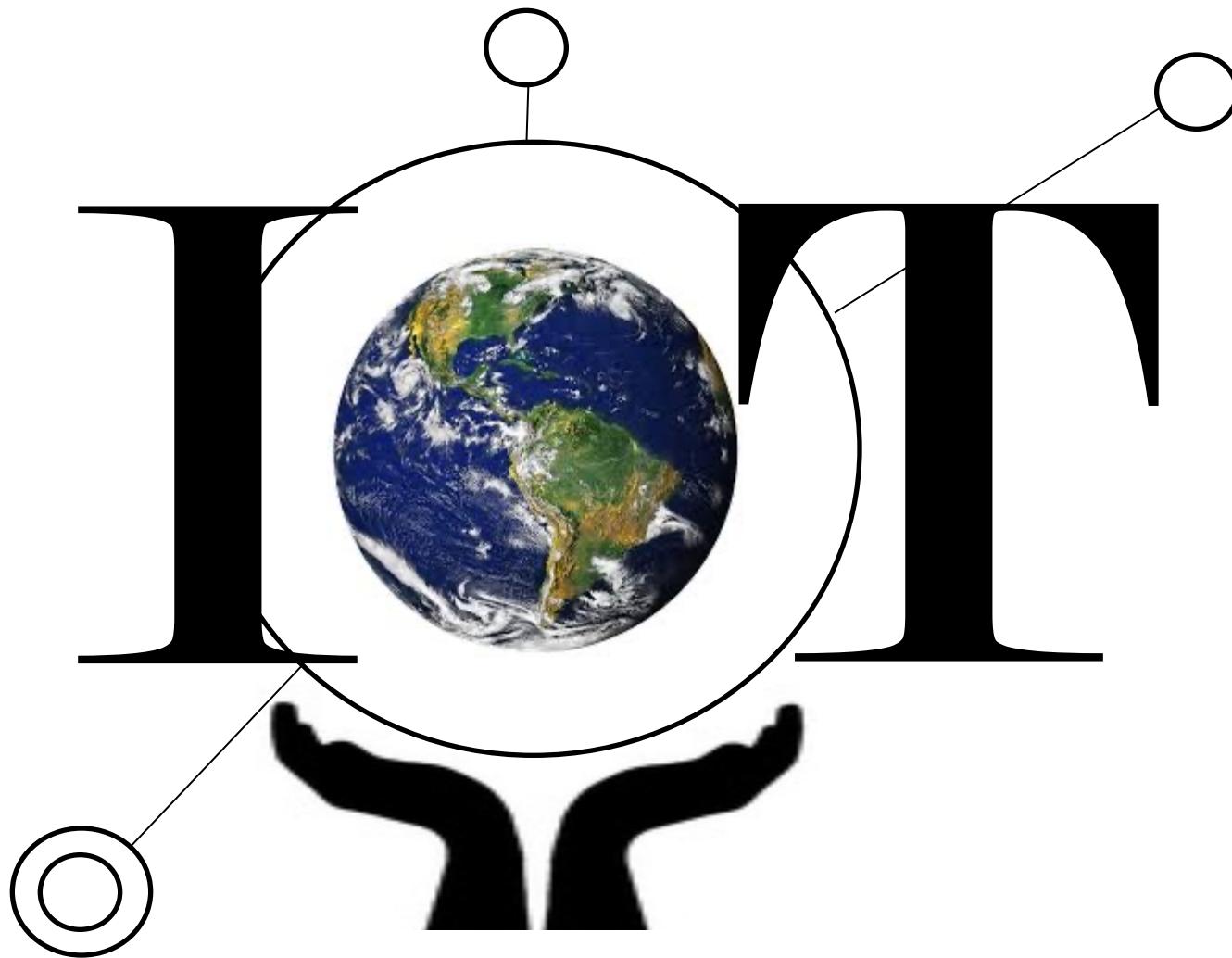
Senior Design Project Course

Cloud  *IOT*

Lecturer: Avesta Sasan

University of California Davis

Lets Get Started:



Focus of Today's Lecture: (Review)

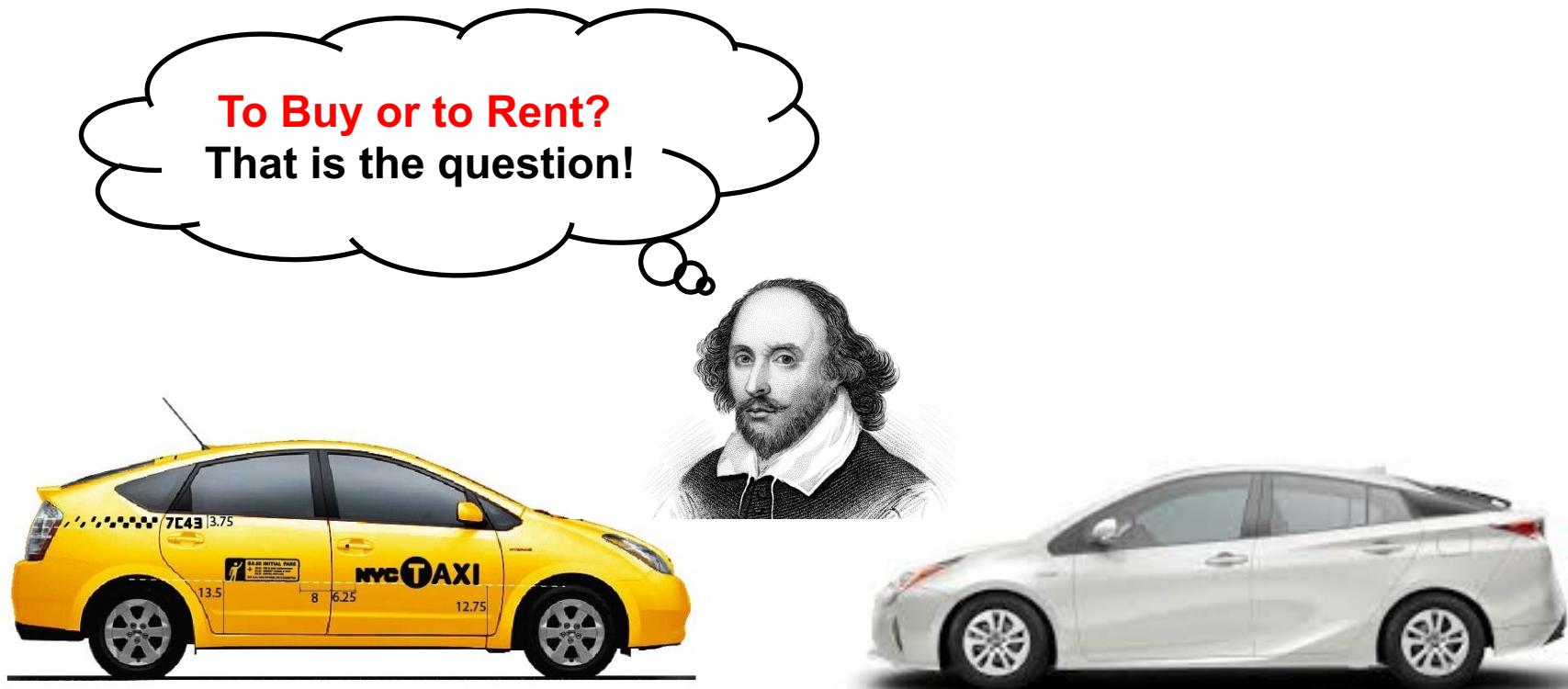


Image source: <http://www.cchc.cl/informacion-a-la-comunidad/industria-de-la-construccion/personaje/>

What is Cloud?

■ IBM definition of Cloud:

- ❑ the delivery of on-demand computing resources—everything from applications to data centers—over the internet on a pay-for-use basis.



Properties of Cloud:

■ Properties of Cloud (IBM definition)

- **Elastic resources:** Scale up or down quickly and easily to meet demand



- **Metered service:** Only pay for what you use



- **IT support:** All the IT resources you need with self-service access



Cloud vs Classical Computing

Classical Computing

- Buy & Own
 - Hardware, System Software, Applications often to meet peak needs.
- Install, Configure, Test, Verify, Evaluate
- Manage
- Finally, use it
- \$\$\$\$\$....\$(High CapEx)

Every few (18) months?



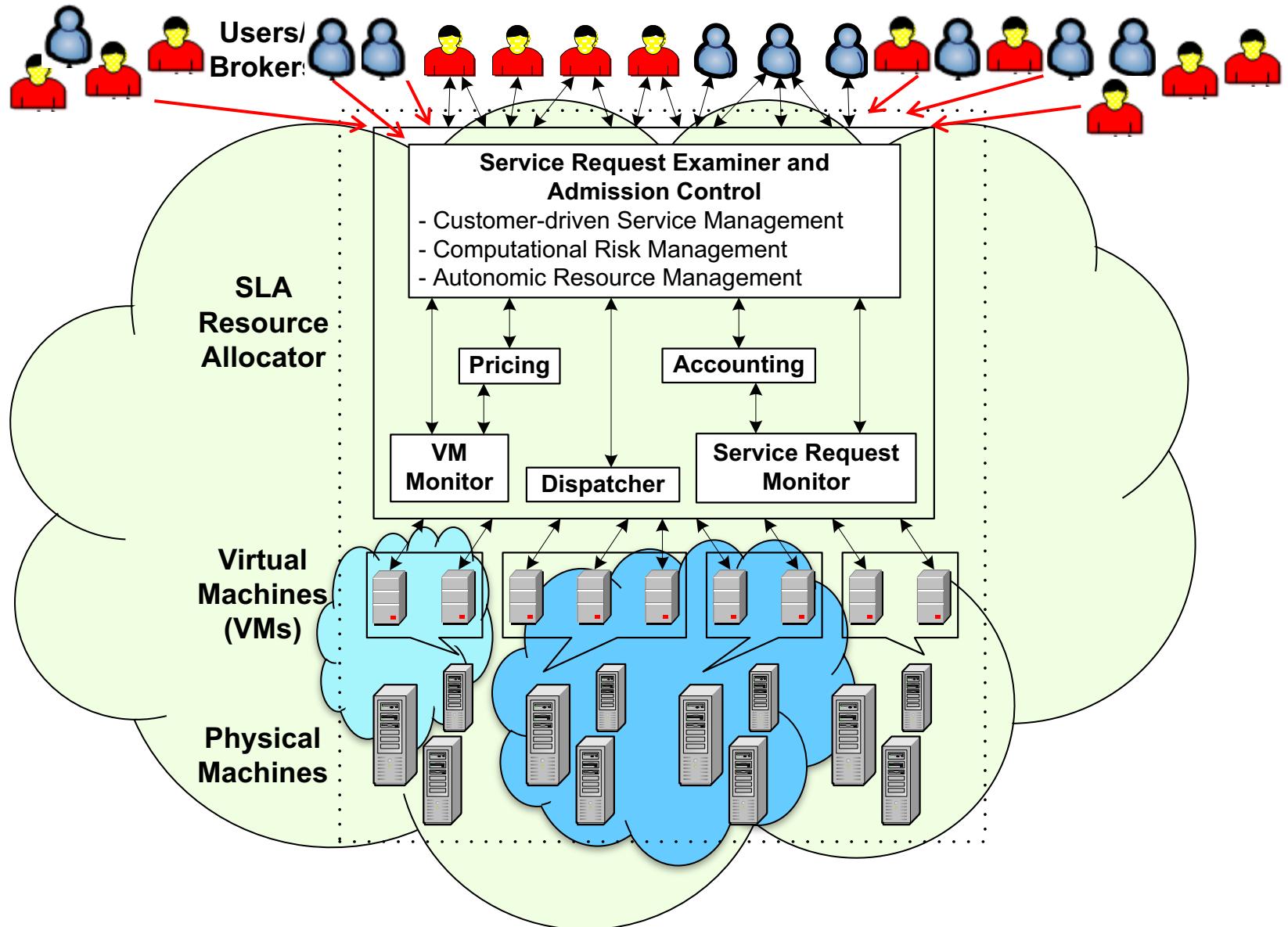
Cloud Computing

- Subscribe
- Use

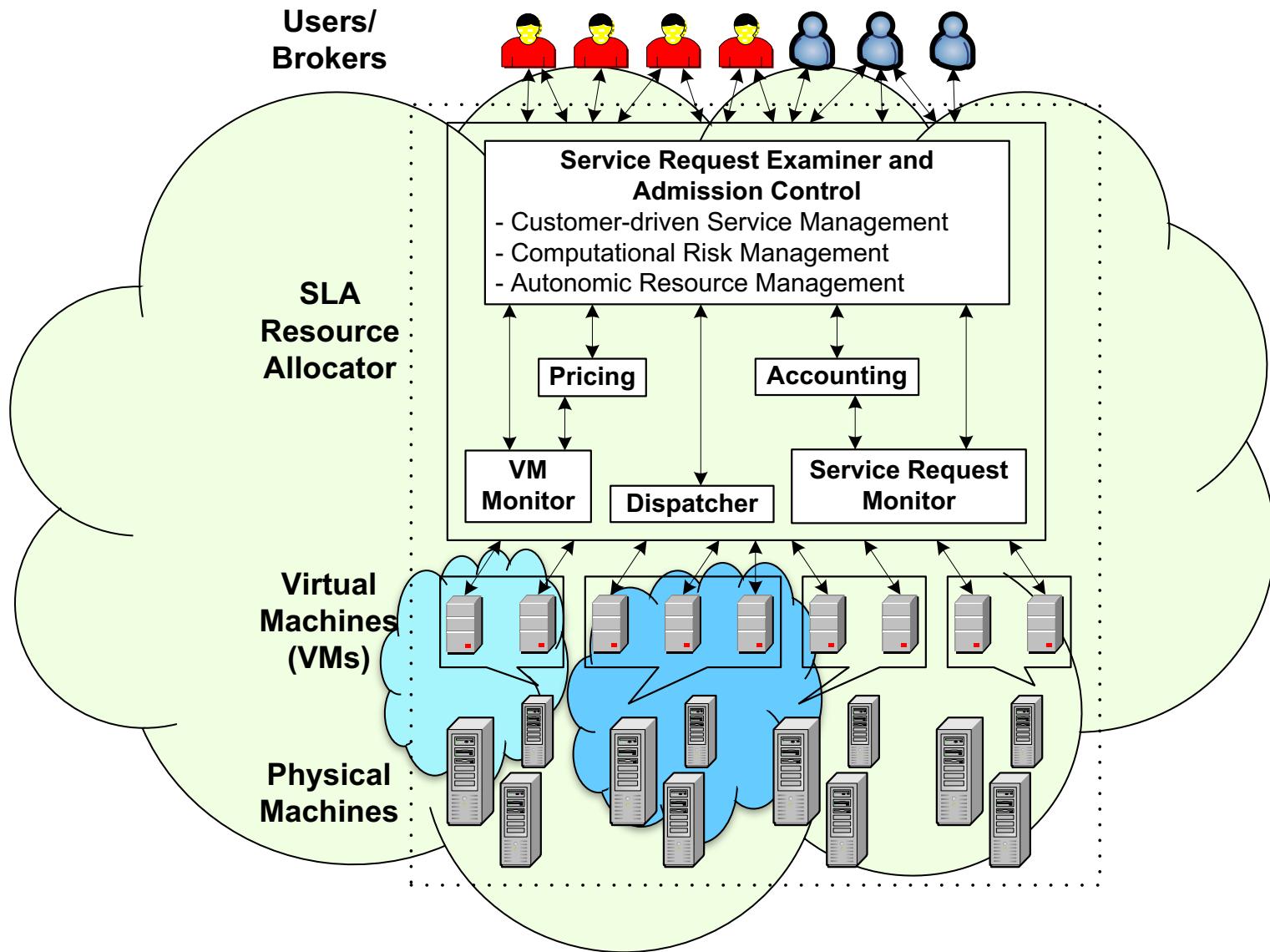


- Pay for what you use!

Cloud to expand (lease more resources) with increase in demand



Cloud to shrink (unlease resources) with decrease in demand

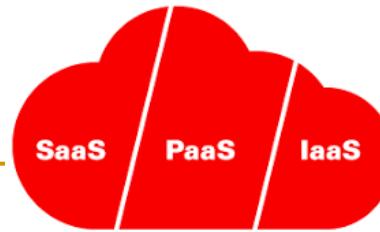


Cloud Servers

Cloud servers are powerful computers housed in big data centers or **server farms**



Cloud Services



- **SaaS (Software as a Service)**
 - Provides off-the-shelf applications offered over the internet
- **PaaS (Platform as a Service)**
 - allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure
 - makes the development, testing, and deployment of applications quick, simple, and cost-effective.
- **IaaS (Infrastructure as a Service)**
 - self-service models for accessing, monitoring, and managing remote datacenter infrastructures, such as compute (virtualized or bare metal), storage, networking, and networking services (e.g. firewalls)

Software as a Service (SaaS)

■ What is SaaS: (IBM definition)

- The applications/software runs on distant computers (in the cloud).
- The distant computing resources are owned and operated by others.
- The connection and service request is made through internet (usually by a web browser).

■ SaaS Benefits:

- **No Installation:** Applications are installed and ready to use (just need to signup)
- **Access freedom:** Access the service from any computer connected to the internet.
- **Protected Data:** data is not lost due to computer breakdown, it is always in the cloud.
- **Scalable service:** the computing resources scale up or down based on user needs.

SaaS Examples

- **Microsoft Office 365**
 - Create, edit and share content from any PC, Mac, iOS, Android or Windows device, real-time and collaborative
- **Box**
 - Securely share large files, preview content prior to downloading, real-time notifications when edits are made.
- **Google Apps**
 - Custom professional email (eliminating email attachments), shared calendars and video meetings, document storage
- **DocuSign**
 - Electronic signature technology
- **Sharelatex**
 - Create latex document online, and work simultaneously.
- **Cisco WebEx**
- **Dropbox**

Platform as a service (PaaS)

■ **What is PaaS (IBM definition)**

- Complete environment: A cloud-based environment with everything required to support the complete lifecycle of building and delivering web-based (cloud) applications
- Without the cost and complexity of buying and managing the underlying hardware, software, provisioning, and hosting.

■ **PaaS Benefits:**

- Develop applications and get to market faster
- Deploy new web applications to the cloud in minutes
- Reduce complexity with middleware as a service

PaaS Examples:

- AWS Elastic Beanstalk
- Windows Azure
- Heroku
- Force.com
- Google App Engine
- Apache Stratos
- RED HAT OPENSHIFT

Infrastructure as a service (IaaS)

■ **What is IaaS (IBM definition)**

- Provides companies with computing resources including servers, networking, storage, and data center space on a pay-per-use basis.

■ **Benefits of IaaS:**

- No need to invest in your own hardware
- Infrastructure scales on demand to support dynamic workloads
- Flexible, innovative services available on demand

IaaS Examples:

- Amazon Web Services (AWS)
- Cisco Metapod
- Microsoft Azure
- Google Compute Engine (GCE)
- Joyent
- Rackspace

Cloud Services

My Software Package

My Application

Data

Runtime

Middleware

Operating Sys

Virtualization

Servers

Storage

Networking

IaaS

PaaS

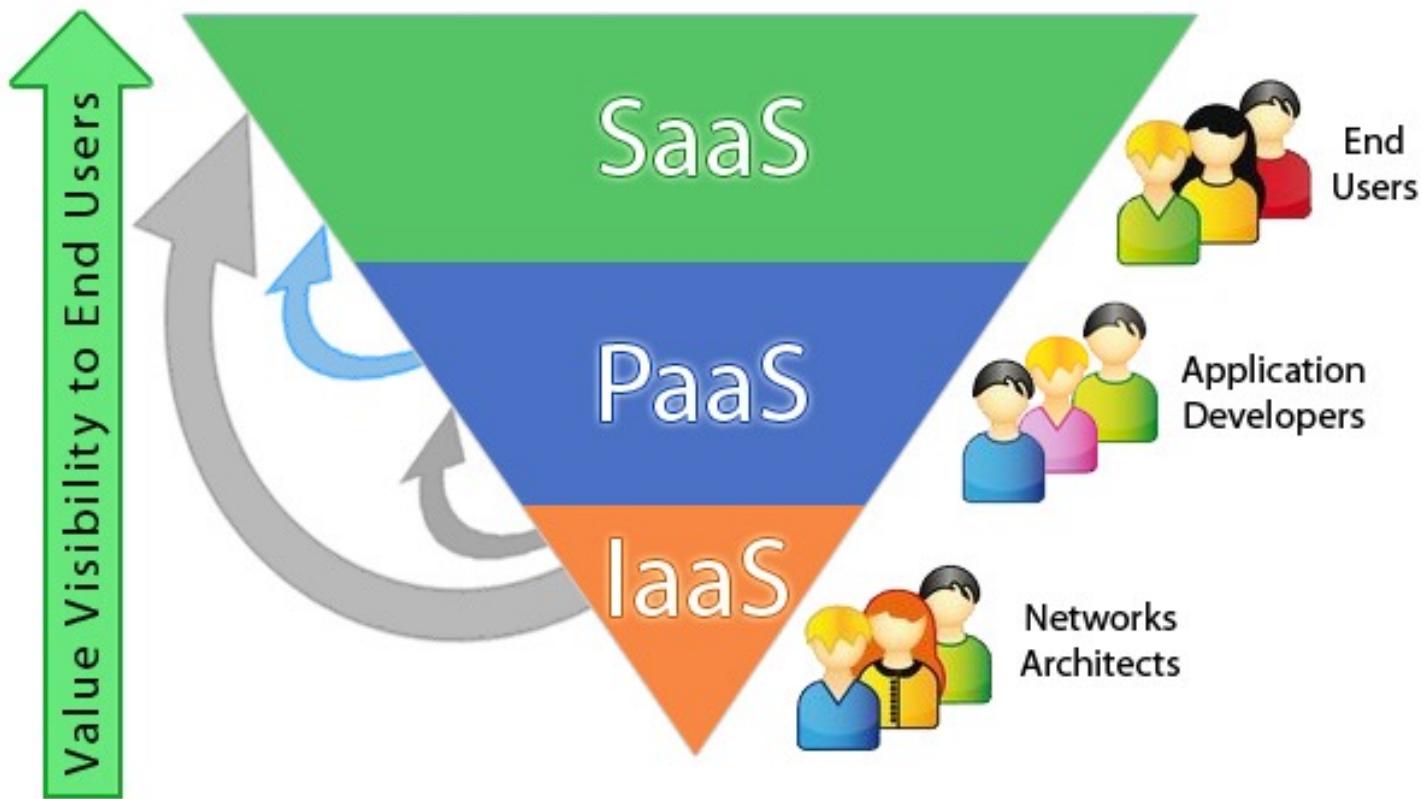
SaaS

Legends :

Managed by Me

Managed by the Vendor

Cloud Services



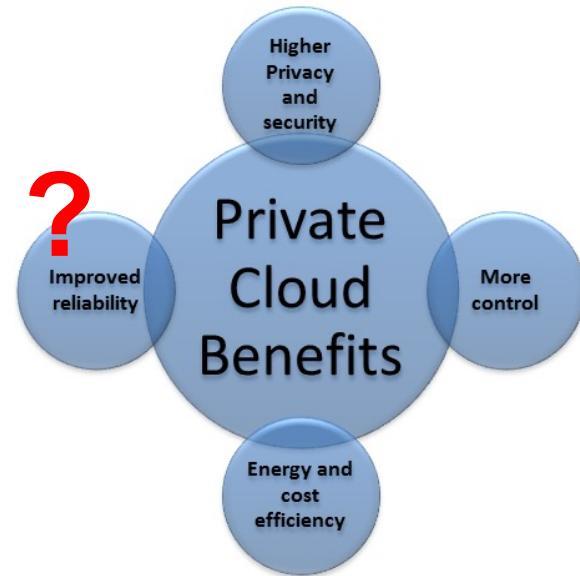
Public Cloud

- Owned and operated by companies that offer rapid access over a public network to affordable computing resources.
 - Microsoft, Amazon, Google, IBM, EMC,
- Users don't need to purchase hardware, software, or supporting infrastructure, which is owned and managed by providers.
- Access to SaaS, IaaS, PaaS.



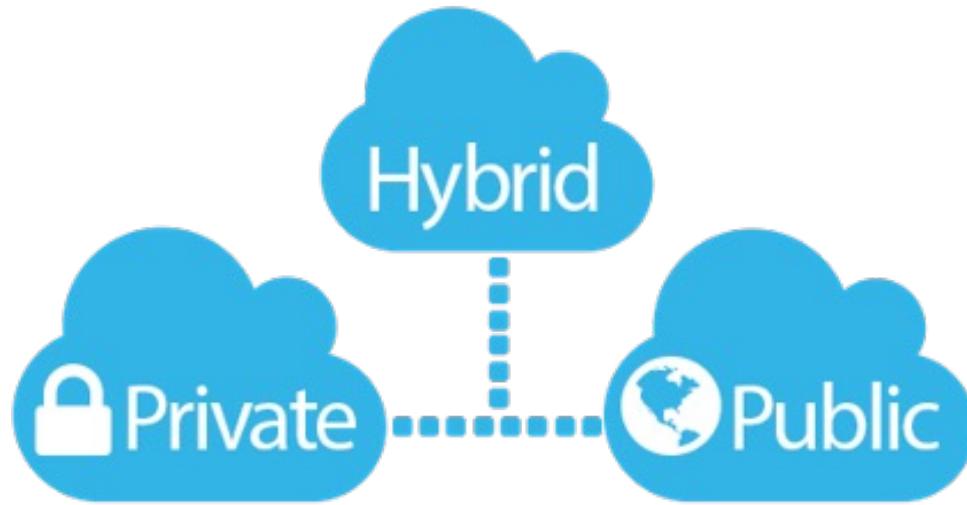
Private Cloud

- Infrastructure operated solely for a **single organization**
- Could be managed internally or by a third party
- Could be hosted either internally or externally.
- Takes advantage of cloud's efficiencies, while providing more control of resources and steering clear of multi-tenancy.
- **Sophisticated security** and governance designed **for a company's specific requirements** (security and privacy are two big reasons for having private clouds).



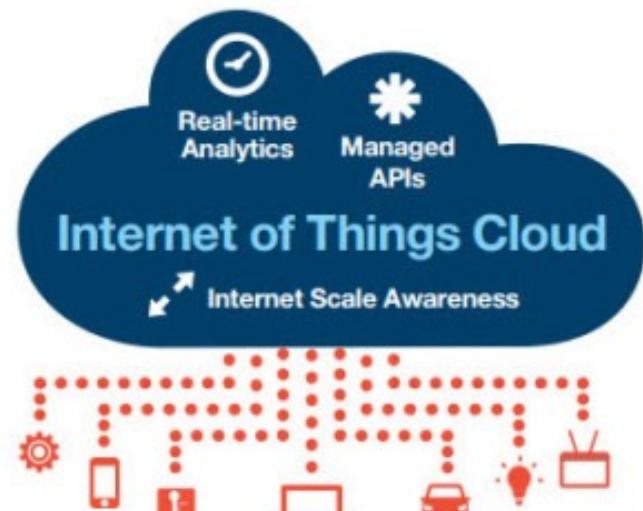
Hybrid Cloud

- Uses a private cloud foundation combined with the **strategic integration and use of public cloud services.**
- Allows companies to keep the critical applications and sensitive data in a traditional data center environment or private cloud
- Enables taking advantage of public cloud resources like SaaS, for the latest applications, and IaaS, for elastic virtual resources
- Facilitates portability of data, apps and services and more choices for deployment models



IoT Meets Cloud

- **IoT is characterized by having many edge devices with limited**
 - ❑ Storage
 - ❑ Processing capacity and Performance
 - ❑ Reliability
 - ❑ Security
 - ❑ Various privacy issues.
- **Cloud is characterized by:**
 - ❑ Large storage
 - ❑ Unlimited processing power and Performance
 - ❑ To a good extent reliable
 - ❑ (security is not yet there!)
- Most of IoT issues are at least partially solved in cloud



IoT Marries the Cloud!

- **Cloud can offer an effective solution to**

- IoT service management
 - IoT service composition
 - Running applications/services that require large storage, and high performance to do
 - Predictive analysis
 - Data mining
 - ...

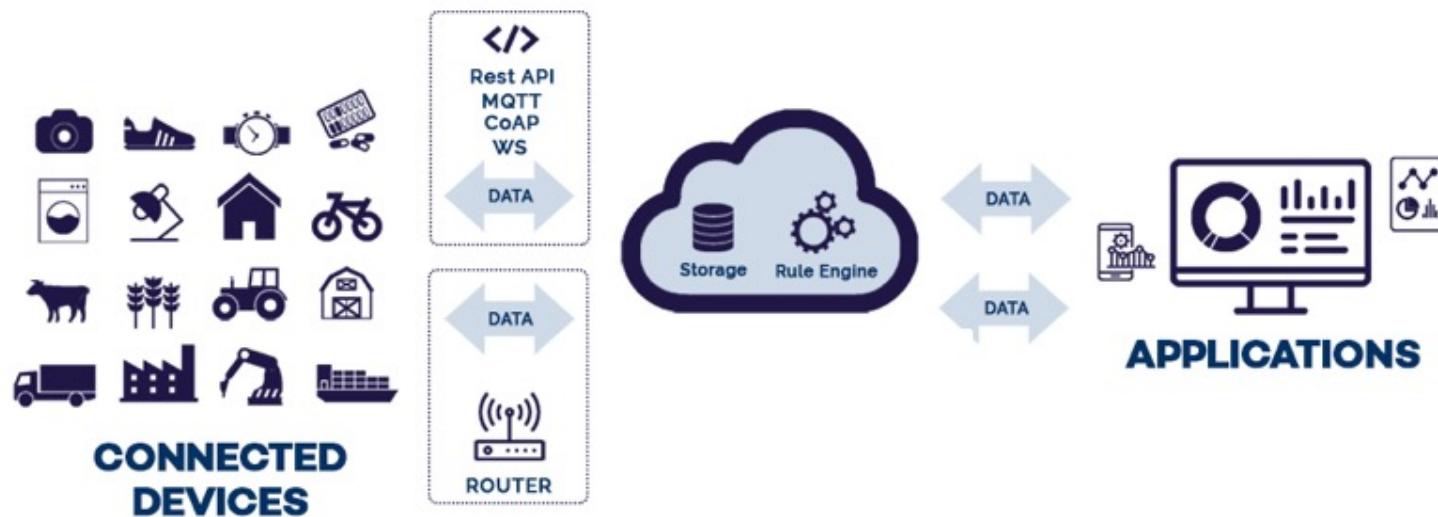


- **IoT allows the Cloud to**

- extending its scope to deal with real world things which are
 - distributed
 - dynamic
 - deliver new services in a large number of real life scenarios.

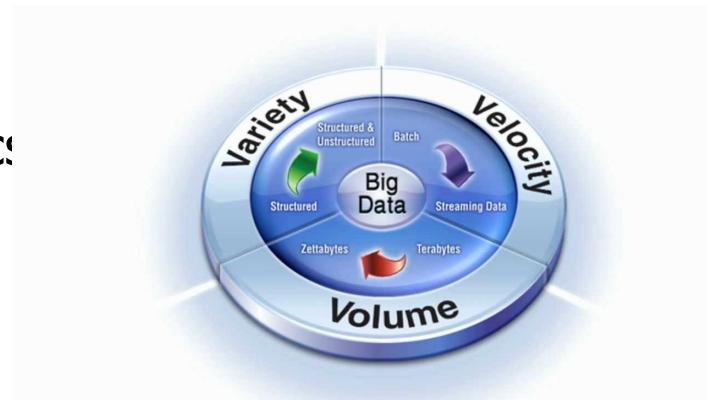
Cloud as a Middle Layer

- The Cloud acts as intermediate layer between the things and the applications
- It hides all the complexity and the functionalities necessary to implement the application/services.



Cloud Storage

- IoT generates a huge amount of data
- Generated Data has all three characteristics:
 - **Volume** (very large)
 - **Variety** (many types of data)
 - **Velocity** (rate of data generation)
- Cloud helps with needed storage services including:
 - Collecting, accessing, archiving and sharing data
- Generated data could be
 - **Unstructured**
 - **Structured**
 - **Semi Structured**

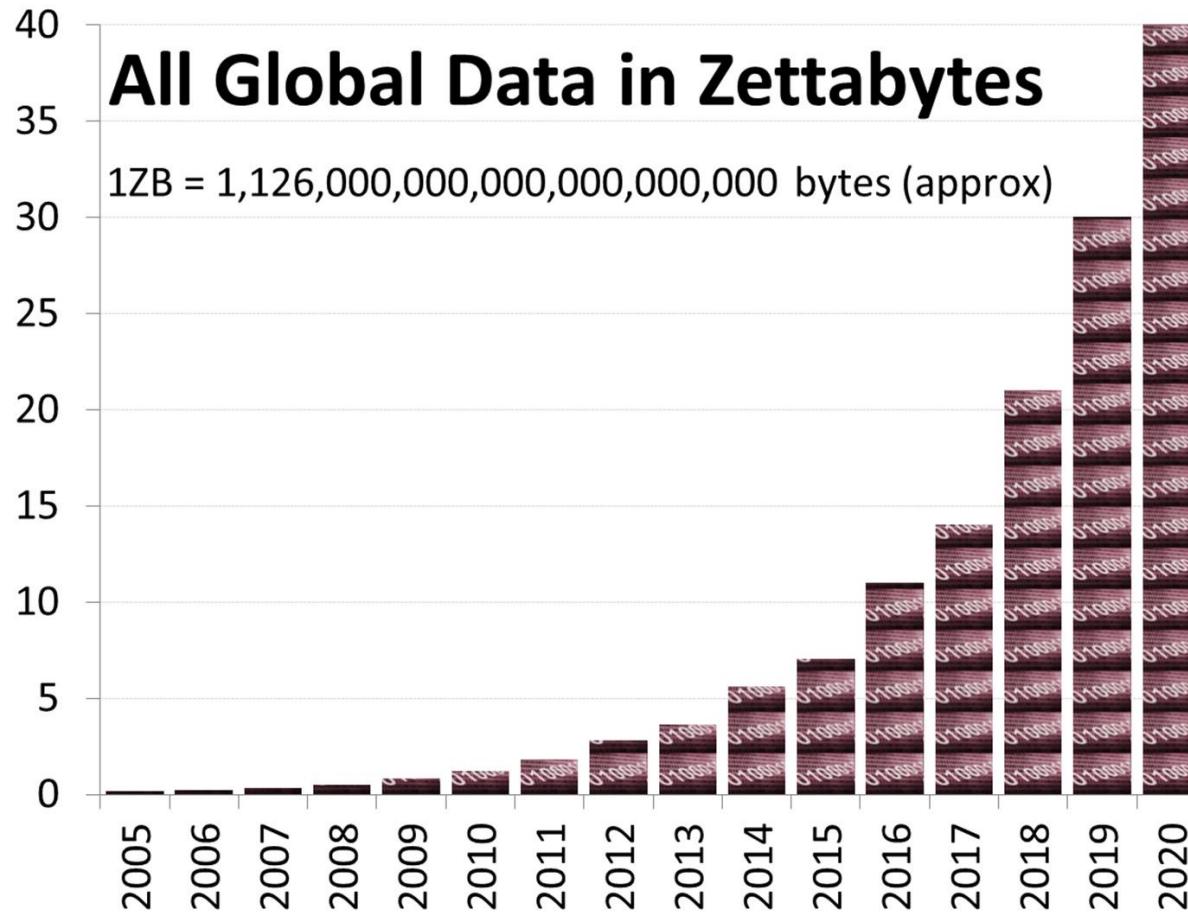


Types of Data

- **Structured Data:**
 - Data that resides in a fixed field within a record or a file.
 - Example: data contained in relational databases and spreadsheets.
- **Unstructured Data:**
 - Refers to information which does not have a pre-defined structure.
 - Example: Free text on web, audio, videos, pdf file, text document etc.
- **Semi-structured Data:**
 - A form of structured data that does not conform with the formal structure of data models associated with relational databases or other forms of data tables.
 - Example: XML

Scale Of Unstructured Data growth

About 85% is unstructured data



Cloud Processing

- Cloud **computational resources** are plenty.
 - Many CPUs, GPUs, TPUs, FPGAs, Custom Accelerators, Vector Processing Machines,
- Cloud could **help with running intensive tasks** such as:
 - Data-driven decision making
 - Prediction algorithms
 - real-time processing (on-the-fly)
 - scalable, real-time, collaborative, sensor-centric applications to manage complex events
 - Data mining
 - ...
- Non of which are possible on IoT devices as
 - These services run after aggregations of data collected from many devices
 - Large scale computational resources are required.

Children of IoT and Cloud Marriage!

- The **IoT+Cloud** enables new scenarios for smart services and applications
 - **SNaaS** (*Sensing as a Service*):
providing ubiquitous access to sensor data
 - **SAaaS** (*Sensing and Actuation as a Service*).
enabling automatic control logics implemented in the Cloud
 - **SEaaS** (*Sensor Event as a Service*).
dispatching messaging services triggered by sensor events
 - **SENaaS** (*Sensor as a Service*).
enabling ubiquitous management of remote sensors

Children of IoT and Cloud Marriage!

- The **IoT+Cloud** enables new scenarios for smart services and applications (Continued ..)
 - **DaaS** (*Data as a Service*)
providing ubiquitous access to any kind of data
 - **EaaS** (*Ethernet as a Service*)
providing ubiquitous layer-2 connectivity to remote devices
 - **IPMaaS** (*Identity and Policy Management as a Service*).
enabling ubiquitous access to policy and identity management functionalities
 - **VSaaS** (*Video Surveillance as a Service*).
providing ubiquitous access to recorded video and implementing complex analyses in the Cloud.

Top 8 Cloud Companies

Companies	Major Cloud Offerings	User Groups
Amazon , Seattle 1994	Amazon Web Services, a half-dozen infrastructure as a services (IaaS) including the EC2 for computing capacity, and the S3 for on-demand storage capacity.	Over 10 thousands of businesses, and individual users, including the New York Times, Wash Post, and Eli Lilly.
Enomaly Toronto 2004	Elastic Computing Platform integrates enterprise datacenters with commercial cloud offerings, manages both internal and external resources, and VM migration among the datacenters	Customers include Business Objects, France Telecom, NBC, Deutsche Bank, Best Buy, etc.
Google , Mountain View, 1998	GAE offers a PaaS plus office productivity tools including the gmail, calendaring, docs and a web site creation tool Postini, and some security protection services.	Lots of small businesses, enterprises and colleges including Arizona State Univ. and Northwestern Univ.
GoGrid , San Francisco 2008	Offers web-based storage and deploys Windows- and Linux-based virtual servers onto the cloud, with preinstalled software from Apache, PHP, Microsoft SQL and MySQL.	Mostly start-ups, Web 2.0 and SaaS companies, plus a few big names like SAP and Novell
Microsoft , Seattle. 1975	Azure offers Windows-as-a-service platform consisting of the OS and developer services that can be used to build and enhance web-hosted applications	Epicor, S3Edge and Micro Focus are among the early customers using Azure to develop cloud applications
NetSuite , San Mateo 1998	A business software suite including e-commerce, CRM, accounting and ERP tools.	Business customers including Puck Coffee, Wrigleyville Sports and Isuzu.
Rackspace , San Antonio, 1998	Mosso cloud offers a platform for building Web sites; Cloud Files for a storage service; and Cloud Servers, an EC2-like service that provides access to virtualized server instances.	Web developers and SaaS providers such as Zapproved, which uses Mosso to deliver an online productivity tool.~
Saleforce .com San Francisco 1999	CRM tools including salesforce automation, analytics, marketing and social networking tools. The Force.com offers a PaaS for building web apps. on Salesforce infrastructure	Half million customers in financial services, communications and media, energy, healthcare and retailing.

Cloud Vendors

- 3Leaf Systems
- 3PAR
- 3Tera
- 10Gen
- Adaptivity
- Agathon Group
- Akamai
- Amazon EC2
- Apache Hadoop
- Appirio
- Appistry
- AppNexus
- Apprenda
- Appzero
- Aptana
- Arjuna
- Asankya
- AT&T
- Bluewolf
- Boomi
- Box-Net
- Booz Allen Hamilton
- CA
- Callidus Software
- Cassatt
- Cisco
- Citrix
- Cloud9 Analytics
- CloudBerry Lab
- Cloudera
- Cloudscale
- Cloudswitch
- Cloudworks
- Coghead
- CohesiveFT
- Cordys
- Cumulux
- Dataline
- Dell
- Desktoptwo
- ElasticHosts
- Elastic Compute Cloud
- Elasta
- EMC
- Engine Yard
- ENKI
- Enomaly
- Enomalism
- Eucalyptus
- eVapt
- EyeOS
- FlexiScale
- Force.com
- Fortress ITX
- G.ho.st
- GigaSpaces
- GoGrid/ServPath
- Google
- gOS
- Grid Dynamics
- Hadoop
- Heroku
- Hosting.com
- HP
- Hyperic
- IBM
- iCloud
- IMOD
- Intel
- Interoute
- iTricity
- Joyent
- JumpBox
- Juniper Networks
- Kaavo
- Kadient
- Keynote Systems
- Layered Technologies
- LinkedIn
- LongJump
- Meeza
- Mezeo Software
- Microsoft
- Morgan Stanley
- MorphExchange
- Netsuite
- newScale
- Ning
- Nirvanix
- Novell
- OpenNebula
- OpSource
- Oracle
- OTOY
- Parallels
- ParaScale
- Penguin Computing
- Platform Computing
- Q-layer
- Qrimp
- Quantivo
- Quickbase
- Rackspace
- Red Hat
- Reservoir
- Rhomobile
- RightScale
- Rollbase
- rPath
- S3
- SalesForce.com
- Savvis
- ServePath/GoGrid
- SIMtone
- Skytap
- SLA@SOI
- SmugMug
- SOASTA
- Strikelron IronCloud
- Sun
- Terremark
- The GridLayer
- ThinkGrid
- Unisys
- Univa UD
- vCloud
- Vertica
- Virtual Workspaces
- VMware
- WorkXpress
- Yahoo!
- Zetta
- Zimory
- Zoho
- Zuora





Internet of Things

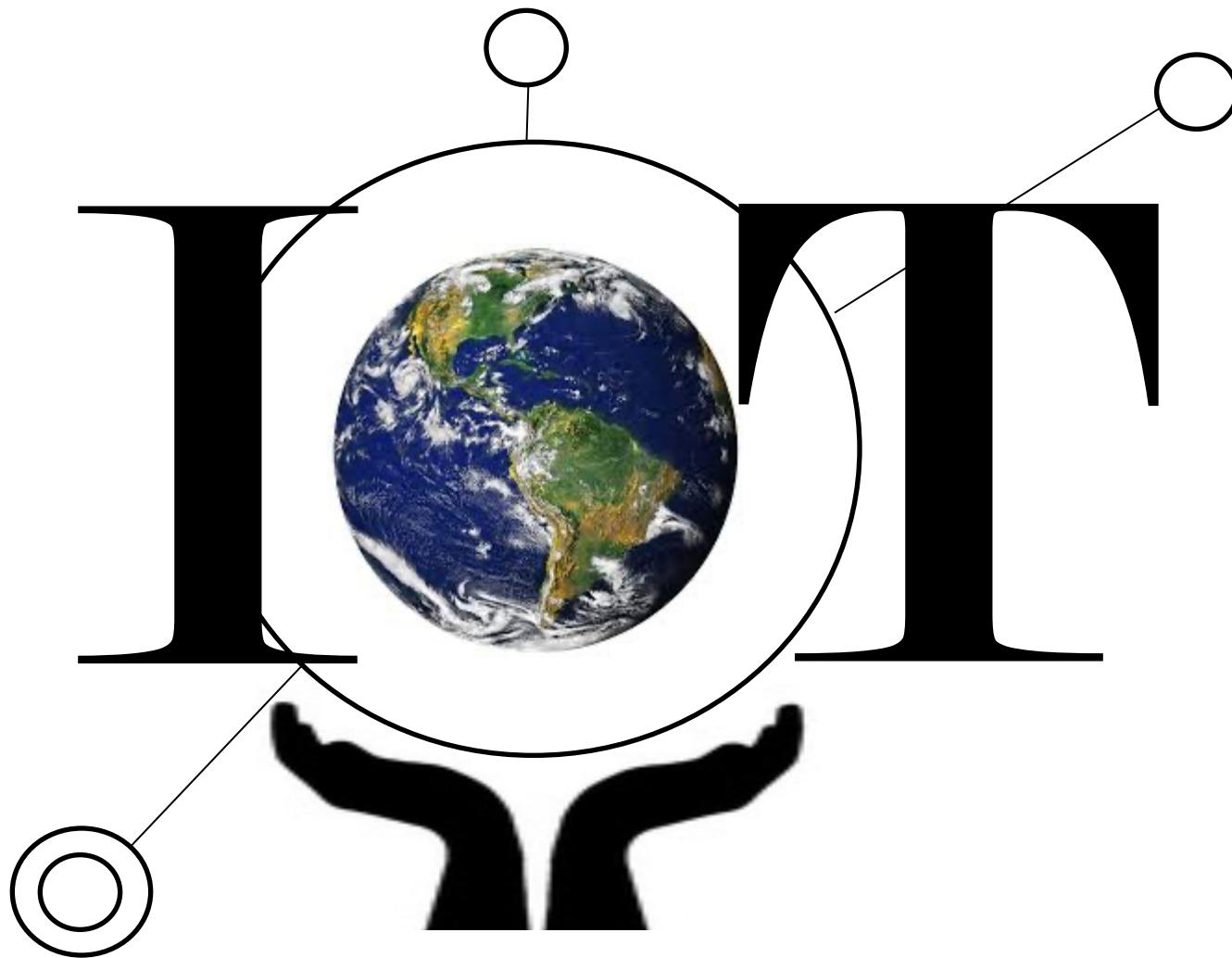
Senior Design Project Course

IP Communication

Lecturer: Avesta Sasan

University of California Davis

Lets Get Started:



Focus of Today's Lecture: (Review)

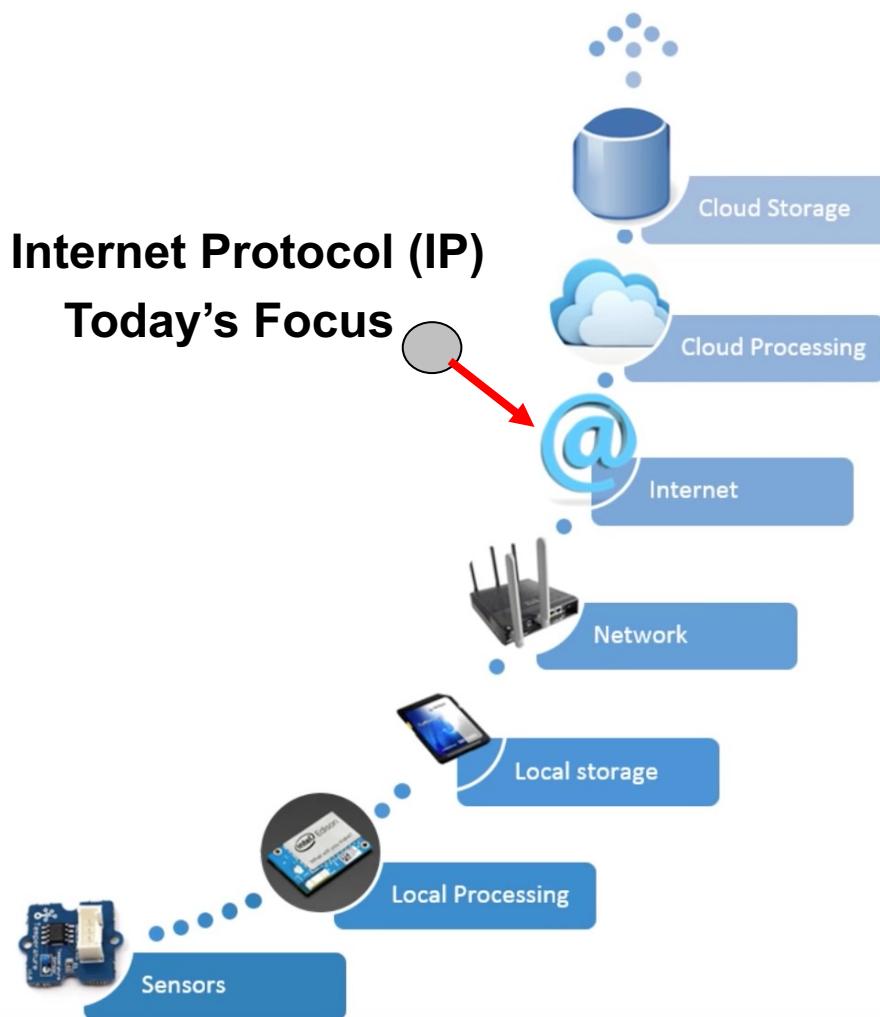


Image source: <http://www.cchc.cl/informacion-a-la-comunidad/industria-de-la-construccion/personaje/>

Why Using Gateway? (Review)

1. Lowering Power

- ❑ Sensor send data to a gateway in short range, gate way send the data to cloud.

2. Supporting varying to/from sensor communication protocols

- ❑ Each sensor may have a different protocol. Gateway translate it to IP

3. Filtering the data

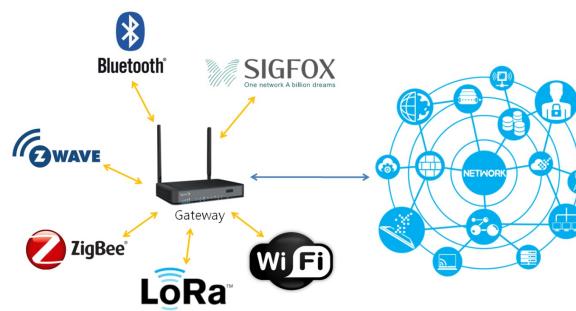
- ❑ Usually small fraction of data is usable. Filtering could be done at gateways.

4. Reducing latency

- ❑ Many IoT devices too small to do the processing themseleves, and it take too long to wait for cloud. Gateway remedy this.

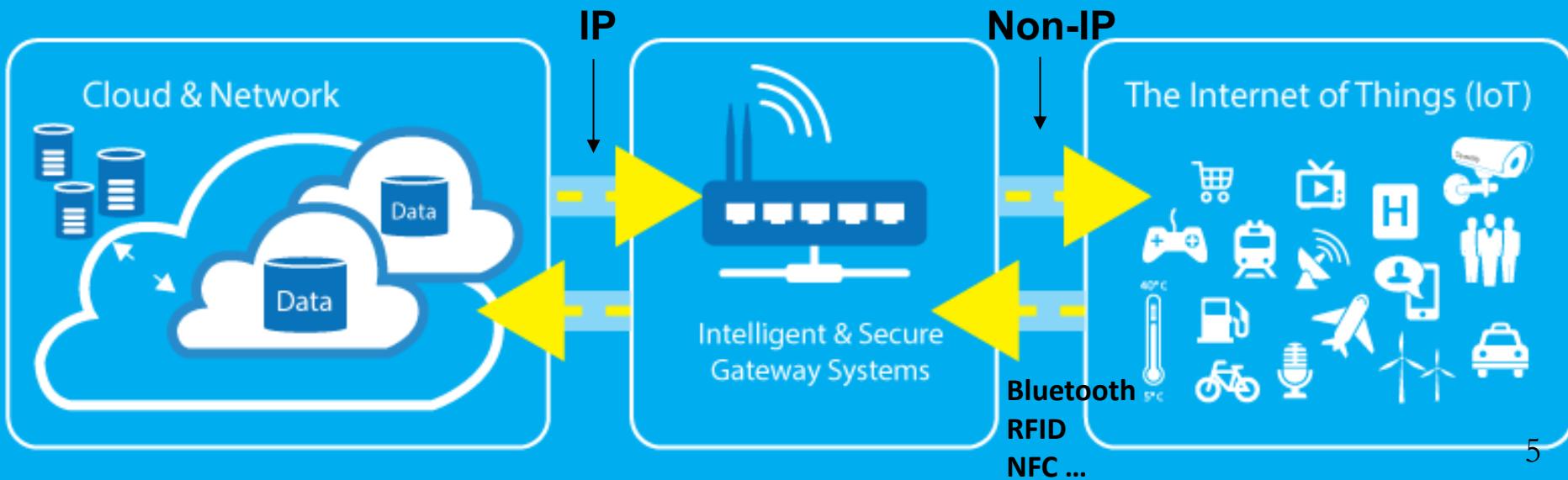
5. Improving security

- ❑ Can afford to make data transmission through gateway more secure.
- ❑ Prevent too many lightly secured sensors to be connected to internet.



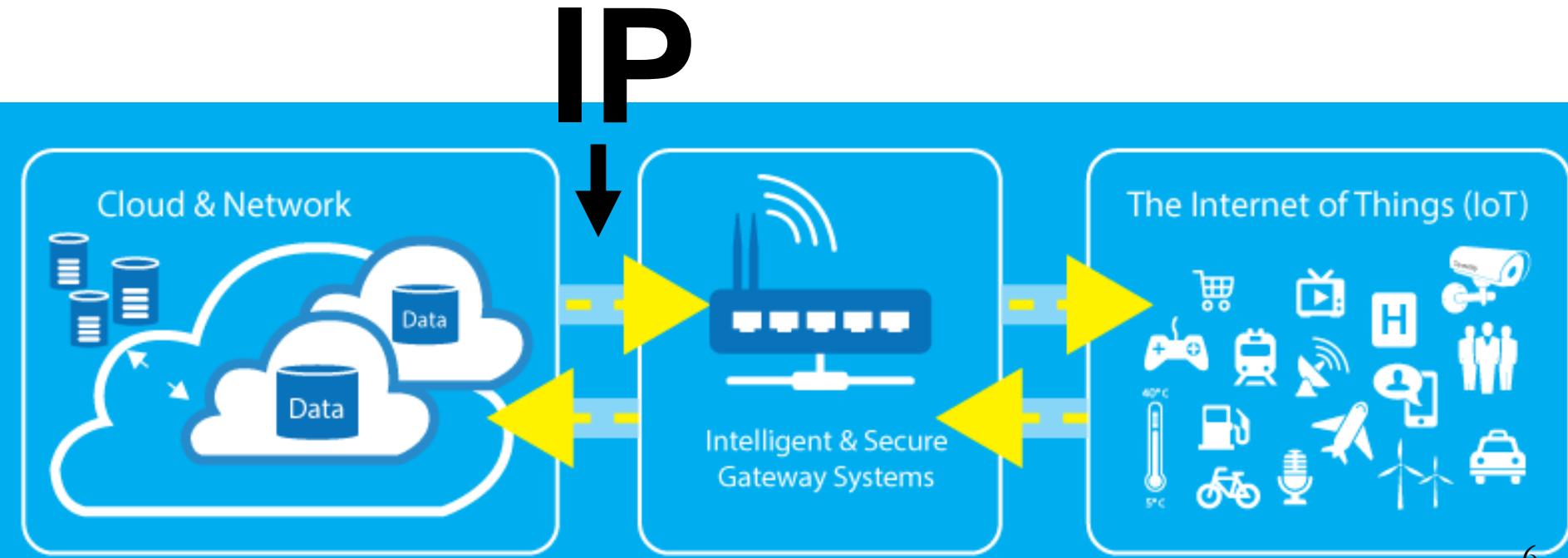
Gateway & Protocol Translation (Review)

- IoT devices can connect to Internet using Internet Protocol (IP) stack.
 - **Problem:**
 - IP stack is very **complex**.
 - Demands a large power and memory from the connecting devices.
- **Gateway removes the need for direct connection to internet.**
 - IoT devices can also connect locally through non-IP networks
 - Consume less power and offer larger mobility, and connect to the Internet via a smart gateway.



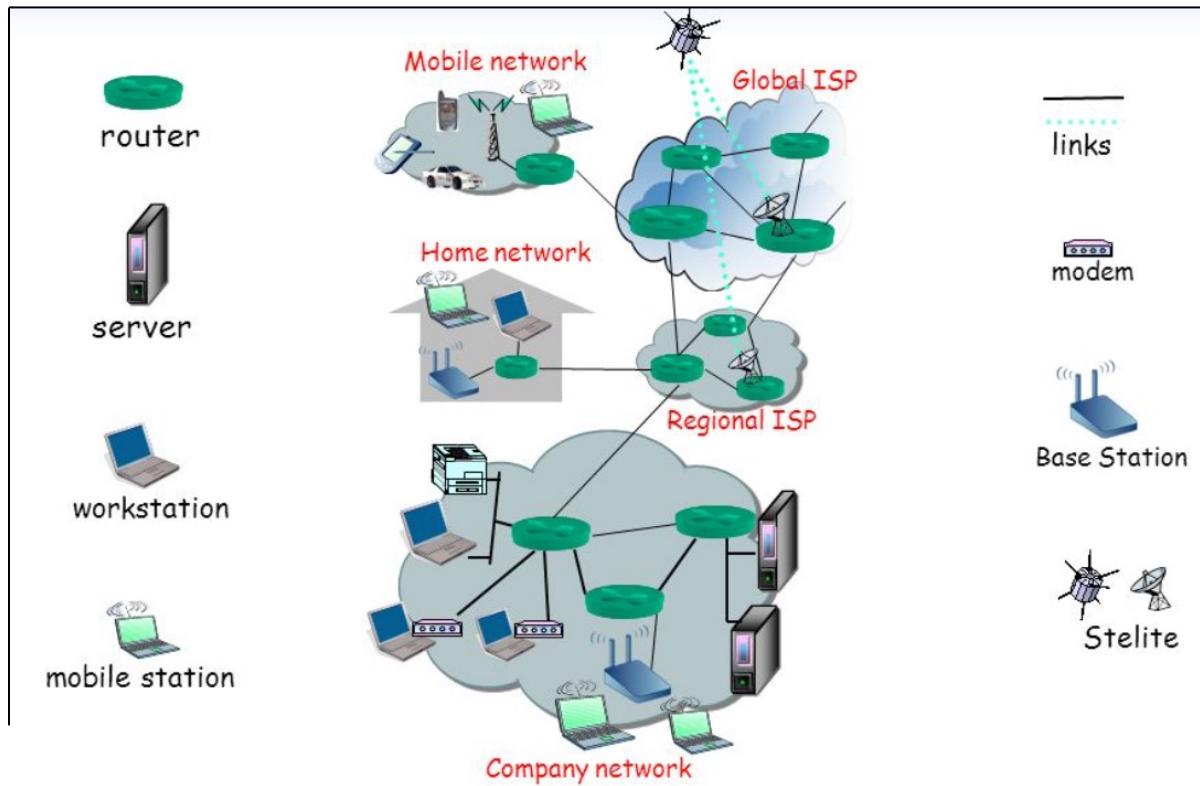
Internet Protocol

- Internet Protocol (**IP**) widely used to send/receive data over **Internet**.
- There are courses that are entirely dedicated to TCP/IP communication
- In this lecture, we are going to just briefly review this technology



Internet

- **Definition:** A set of *interconnected networks*
- Underlying networks can be completely different
- (TCP/)IP is what links them



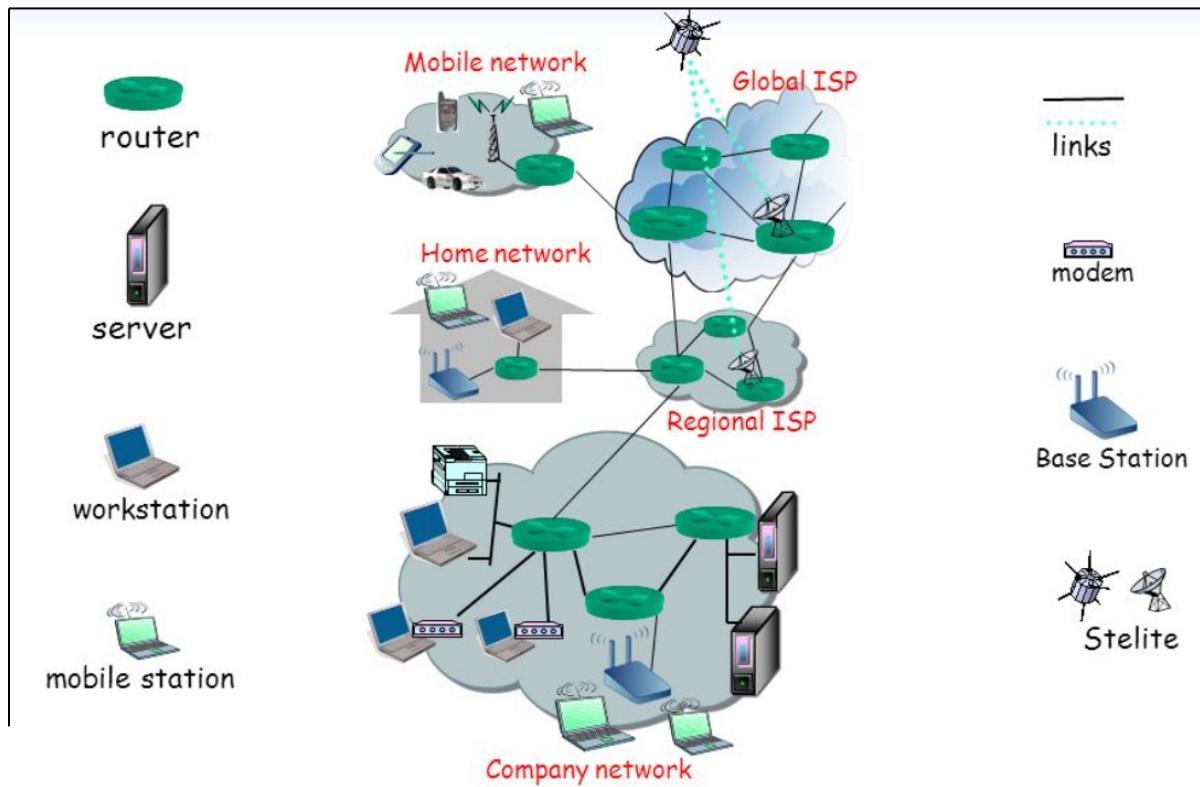
Internet

- Internet is a **distributed system** (no central control)
- Billions of connected devices
- Host: end systems that run network applications (protocols)
- Communication links
 - Fiber, copper, radio, satellite
 - Bandwidth matters the most. Delay, jitter etc. are also important
- **IP** is the glue that connects all these devices
- Analogous to sending mail using the postal service



Routers:

- **Routers:** devices on multiple networks that pass traffic between them
 - Individual networks pass traffic from one router or endpoint to another
 - TCP/IP hides the details as much as possible

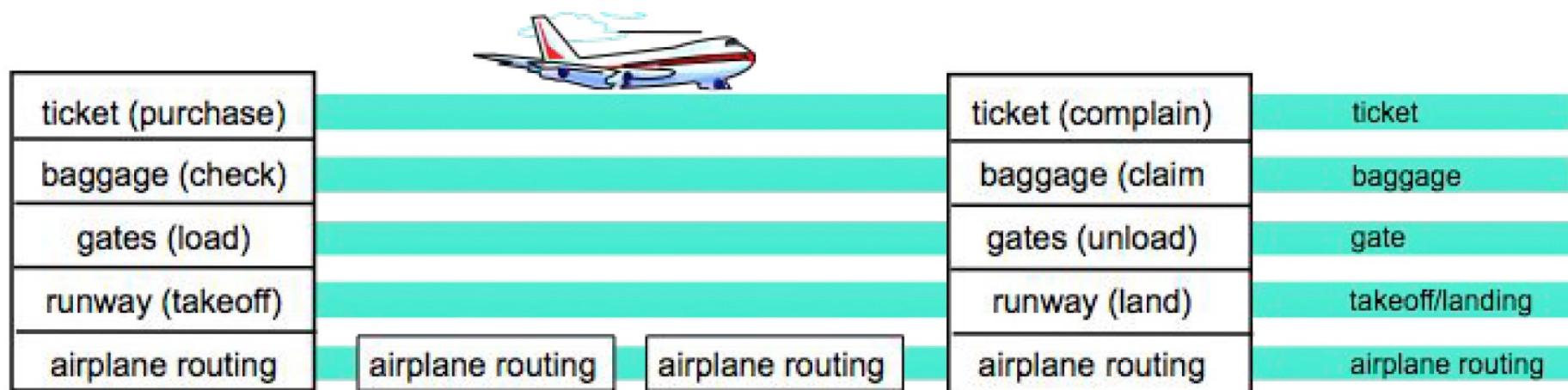


Protocol Layering

- Networks are complex! Too many pieces
 - Hosts, Routers
 - Links of various media
 - Applications
 - Protocols
 - Hardware, software
- Layers Implement service abstractions
 - Each relying on services provided by layer below

Protocol Layering Example

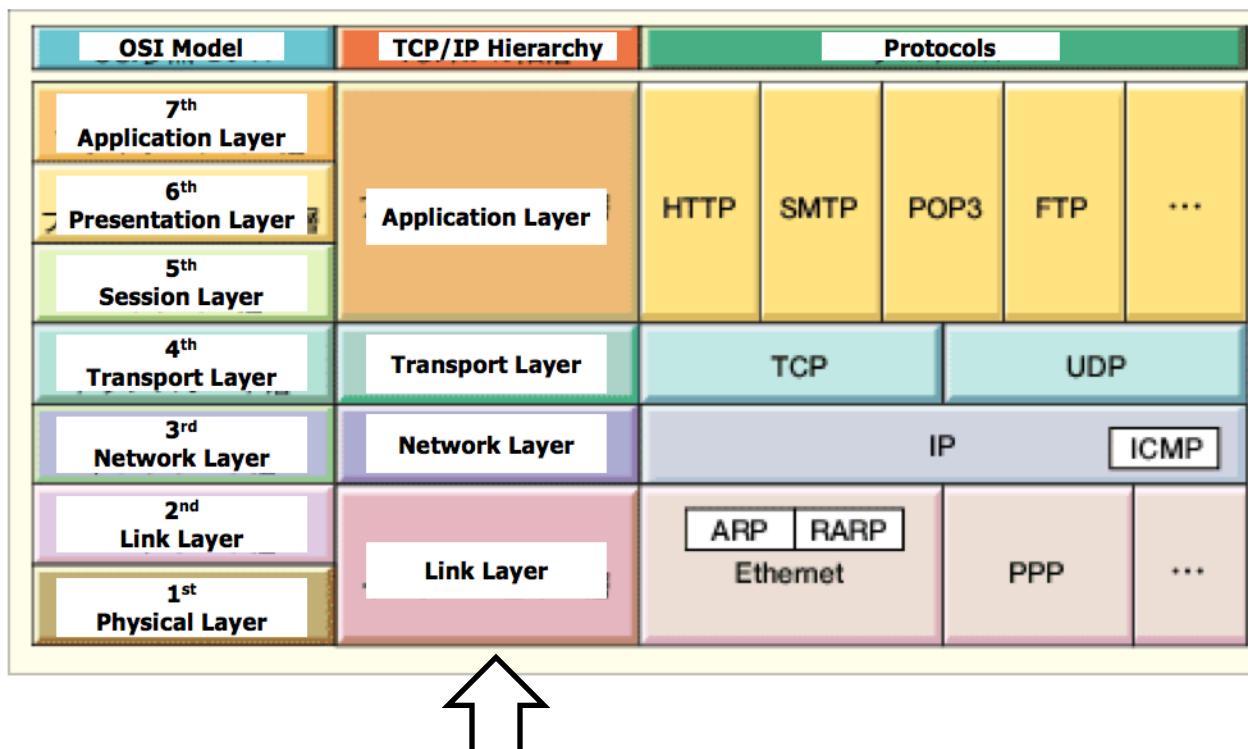
- Layers Implement service abstractions
 - Each relying on services provided by layer below



TCP/IP Network Model

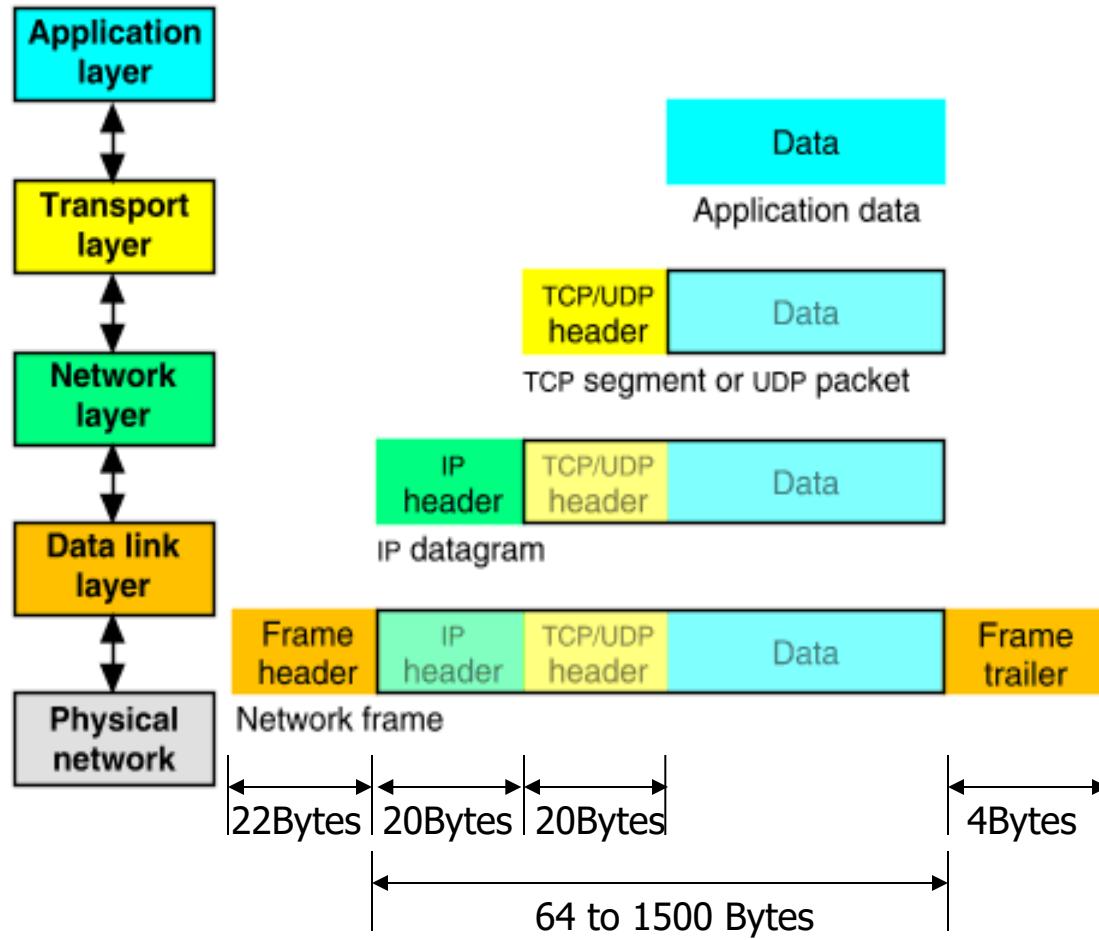
■ A layered Protocol:

- ❑ Layer 1 : **Link** : includes device driver and network interface card
- ❑ Layer 2 : **Network** : handles the movement of packets, e.g., Routing
- ❑ Layer 3 : **Transport** : provides a reliable flow of data between two hosts
- ❑ Layer 4 : **Application** : handles the details of the particular application



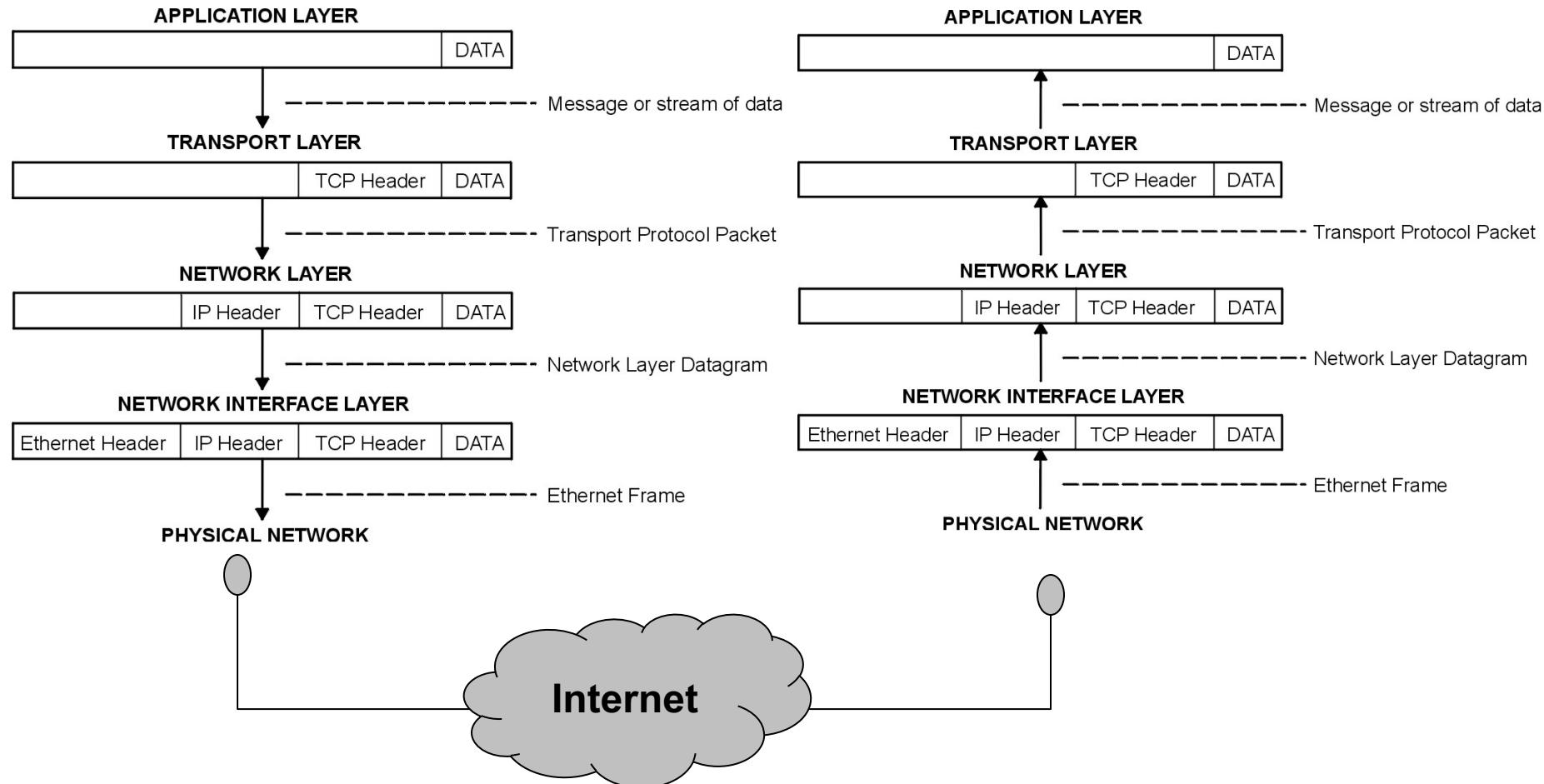
Packet Encapsulation

- The data is sent down the protocol stack
- Each layer adds to the data the required header



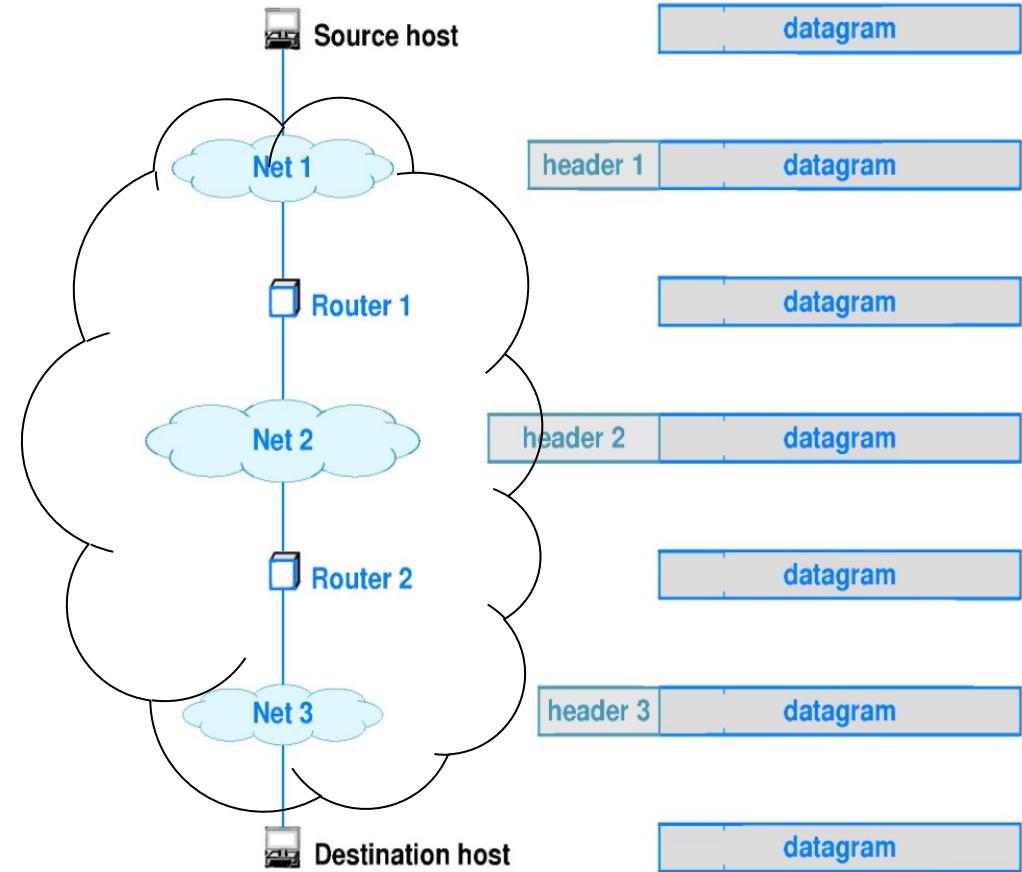
Packet Encapsulation

- Encapsulation and De-capsulation of data during a transfer!



Link Layer (header and trailer information)

- When a datagram is sent across a physical network, the receiver on the other side, removes the header from the encapsulating frame and ***discards*** it.
- If the IP datagram is forwarded along further, the router places (only) the IP datagram into a new frame suitable to the next network it must cross.



Internet Protocol

- Responsible for end to end transmission
- Sends data in individual packets
- Maximum size of packet is determined by the networks
 - Fragmented if too large
- Unreliable
 - Packets might be lost, corrupted, duplicated, delivered out of order
- Currently internet widely uses IPv4
 - 4 bytes for address
 - e.g. 163.1.125.98
 - Each device normally gets one (or more)
 - In theory there are about **4 billion (2^{32})** addresses available

2^{32} Devices could be addressed

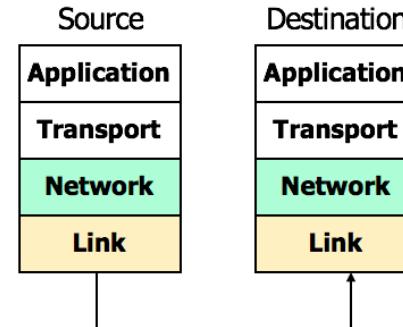


- But the **number of devices in IoT** is much larger.
 - That is why **IPv6** is being introduced. (128 bit addressing scheme).
 - How many unique addresses we could generate using IPv6

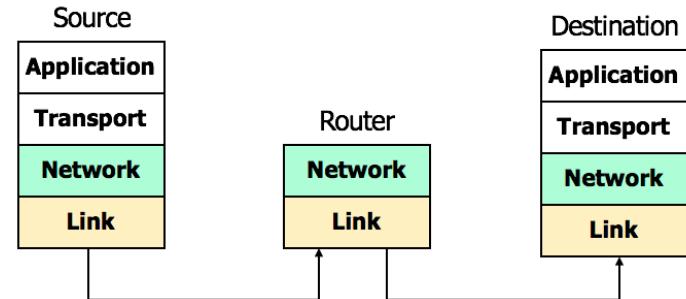
Routing

- How does a device know where to send a packet?
 - All devices need to know what IP addresses are on directly attached networks

- If the destination is on a local network
 - send it directly there

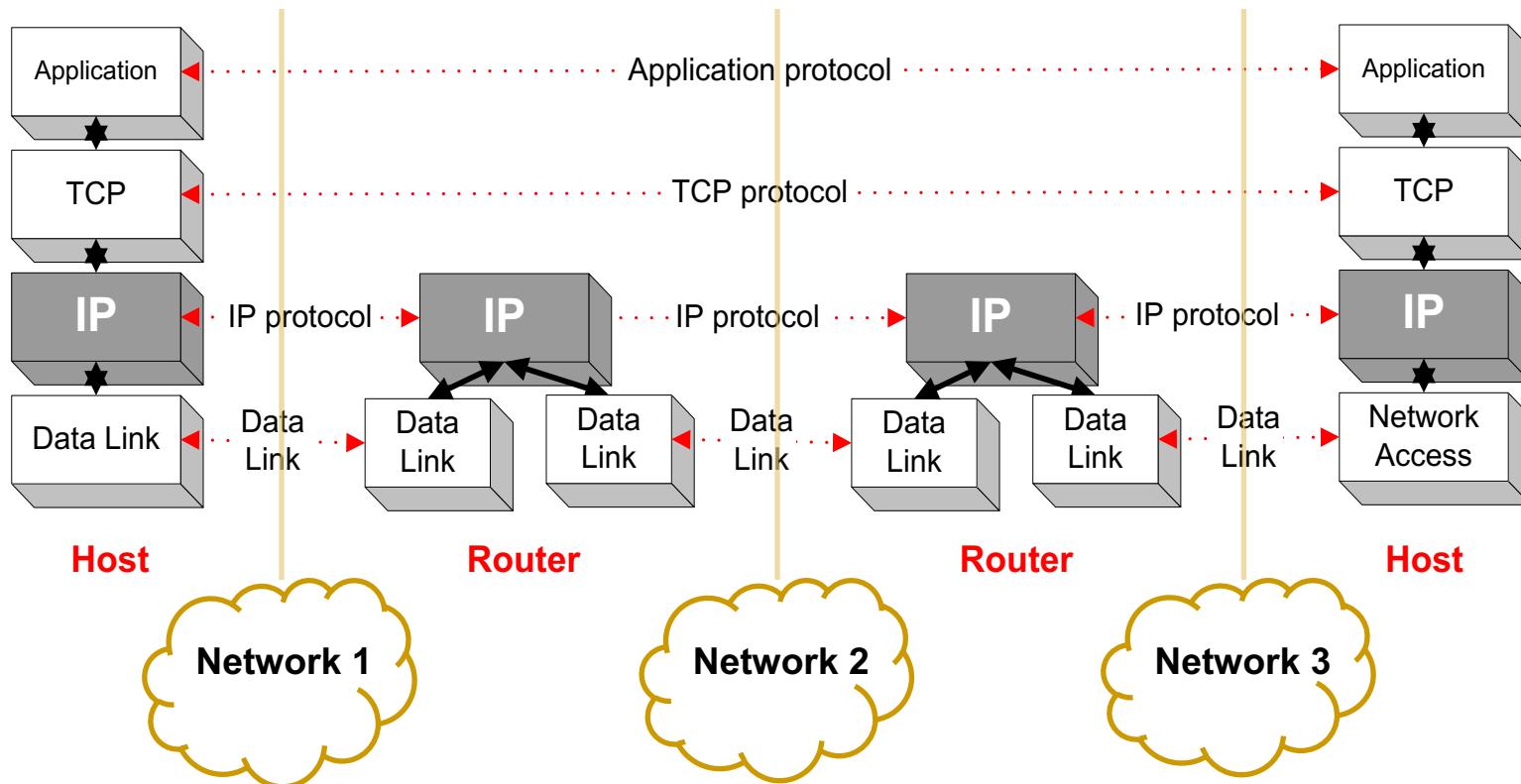


- If the destination address isn't local
 - Most non-router devices just send everything to a single local router
 - Routers need to know which network corresponds to each possible IP address



IP Routing

- IP is the highest layer protocol which is implemented routers.



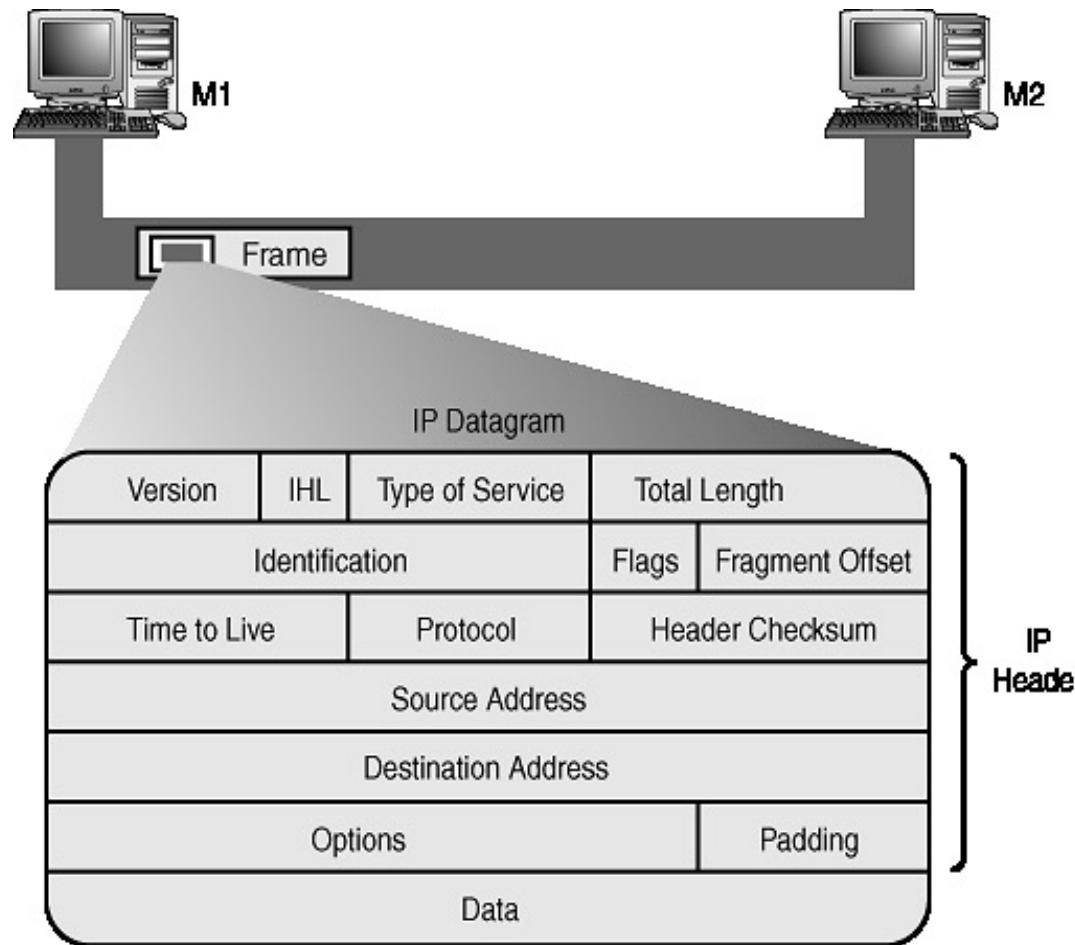
Who Allocates The IP addresses?

- IP address allocation is controlled centrally by **ICANN** (**I**nternet **C**orporation **A**ssigned **N**ames and **N**umbers)
- Fairly strict rules on further delegation to avoid wastage
 - Have to demonstrate actual need for them
 - <https://en.wikipedia.org/wiki/ICANN>
- Organizations that got in early have bigger allocations than they really need



IP Datagram

- The structure of IP datagram is as following:



UDP (User Datagram Protocol)

- A thin layer on top of IP
- Adds packet length + checksum
 - Guard against corrupted packets
- Also source and destination *ports*
 - Ports are used to associate a packet with a specific application at each end
- Still unreliable:
 - Duplication, loss, out-of-order receiving of packets are possible

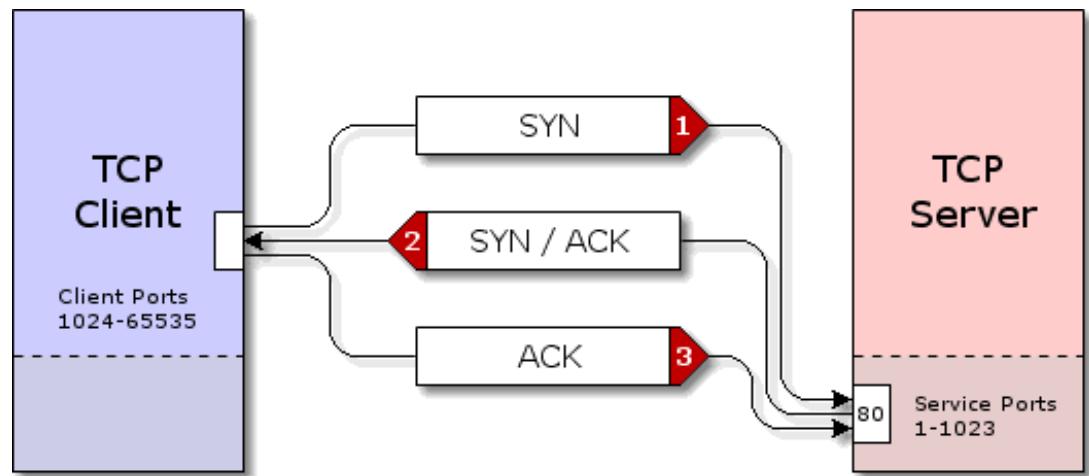
Field	Purpose
Source Port	16-bit port number identifying originating application
Destination Port	16-bit port number identifying destination application
Length	Length of UDP datagram (UDP header + data)
Checksum	Checksum of IP pseudo header, UDP header, and data

TCP (Transmission Control Protocol)

- Reliable alternative of UDP with more overhead
- A tick (in oppose to thin in UDP) layer on top of IP
- Reliable, *connection-oriented, stream* delivery
 - Data is guaranteed to arrive, and in the correct order without duplications
 - Or the connection will be dropped
- **Imposes significant overheads**

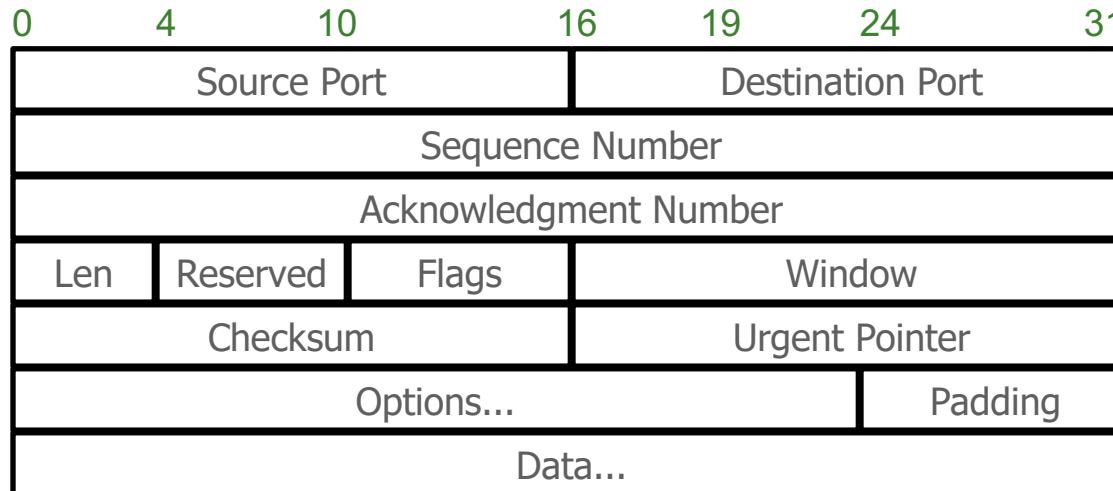
TCP Implementation

- Connections are established using a *three-way handshake*
 - [Click here to learn how three-way handshake works?](#)
- Data is divided up into packets by the operating system
- Packets are numbered, and received packets are acknowledged
- Connections are explicitly closed
 - (or may abnormally terminate)



TCP Header (just for your information!)

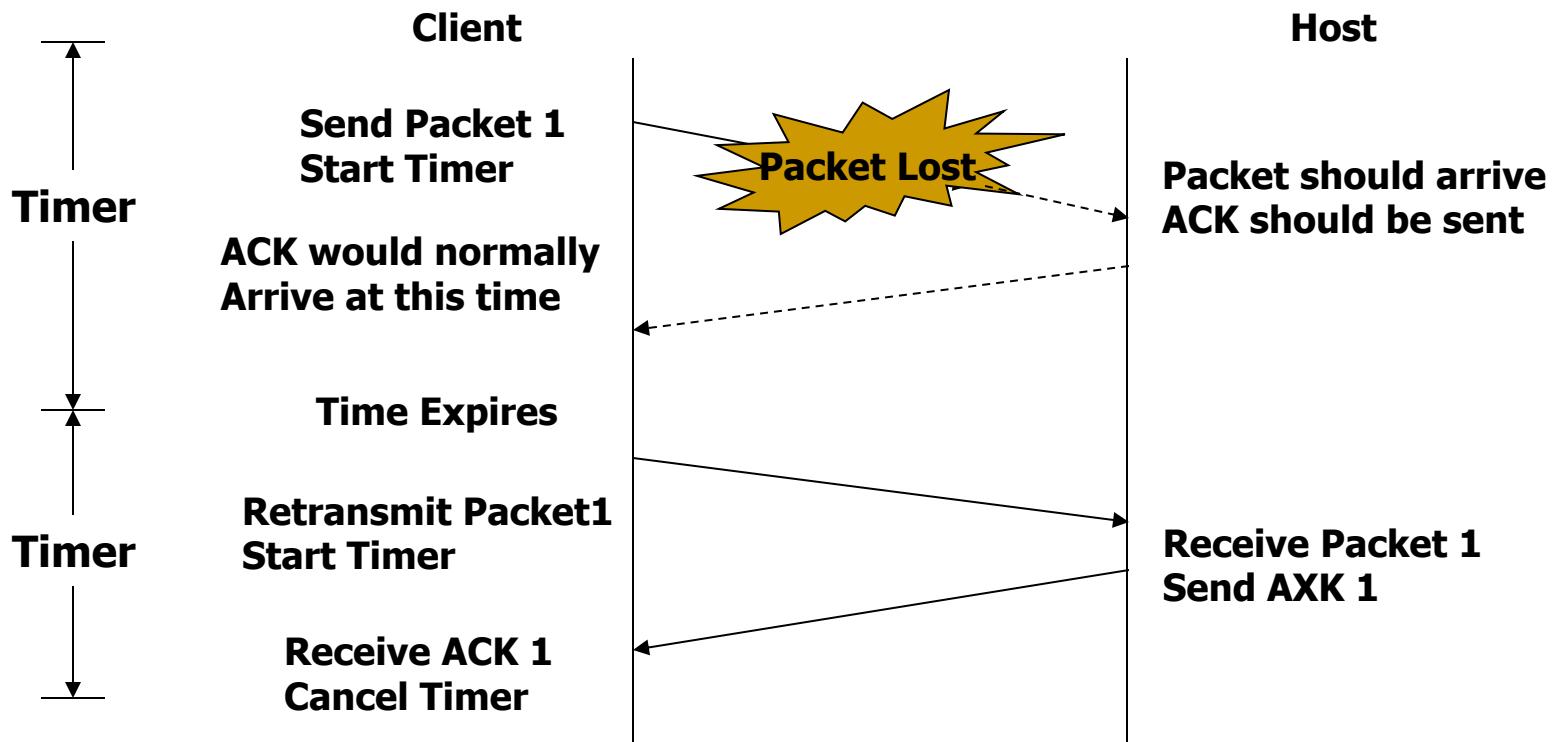
- Compared to UDP, TCP protocol need a larger header size!



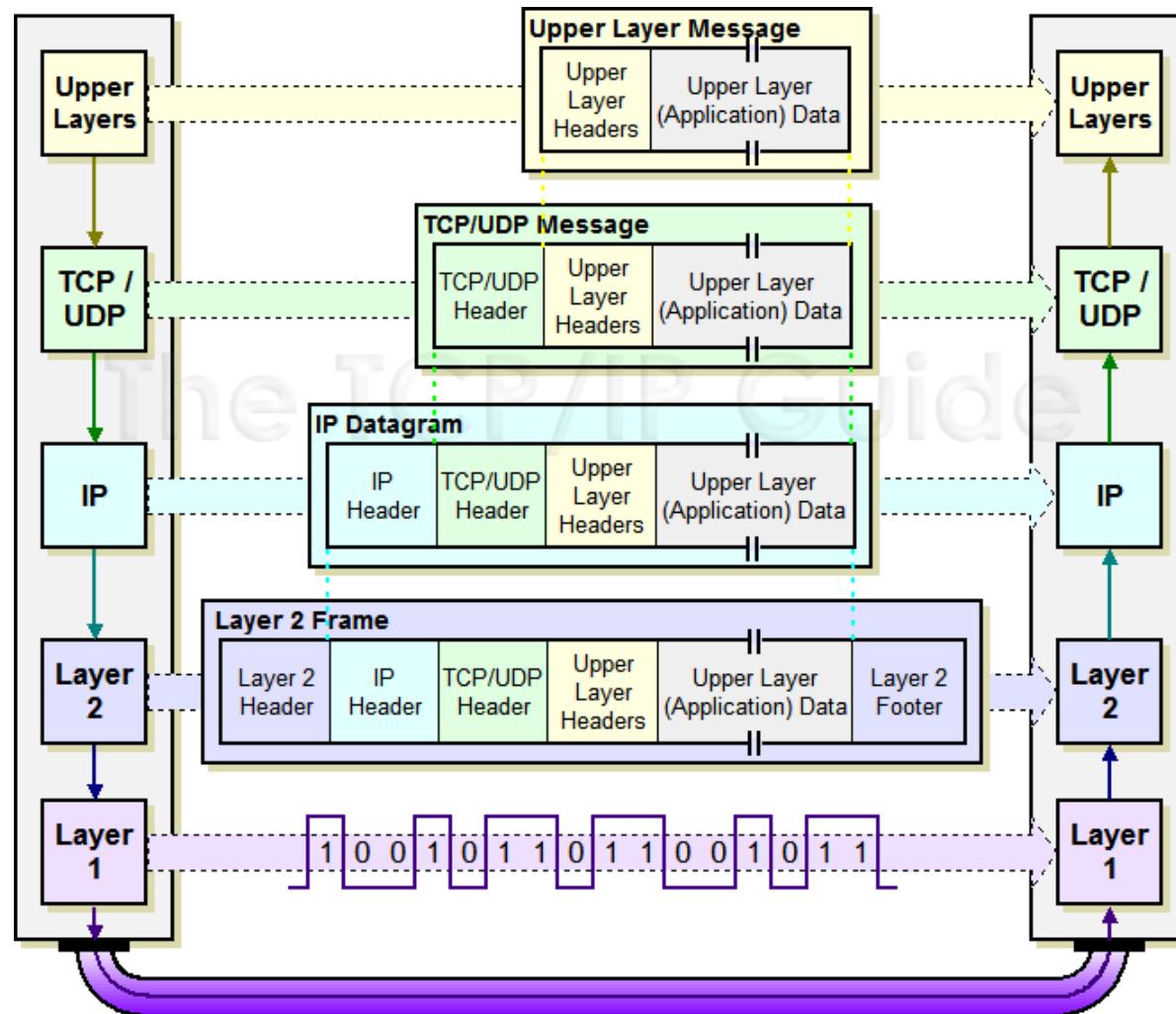
Field	Purpose
Source Port	Identifies originating application
Destination Port	Identifies destination application
Sequence Number	Sequence number of first octet in the segment
Acknowledgment #	Sequence number of the next expected octet (if ACK flag set)
Len	Length of TCP header in 4 octet units
Flags	TCP flags: SYN, FIN, RST, PSH, ACK, URG
Window	Number of octets from ACK that sender will accept
Checksum	Checksum of IP pseudo-header + TCP header + data
Urgent Pointer	Pointer to end of “urgent data”
Options	Special TCP options such as MSS and Window Scale

TCP Data Transfer

- No packet loss in TCP!
 - If acknowledge is not received (within specified time), re-transmit again!

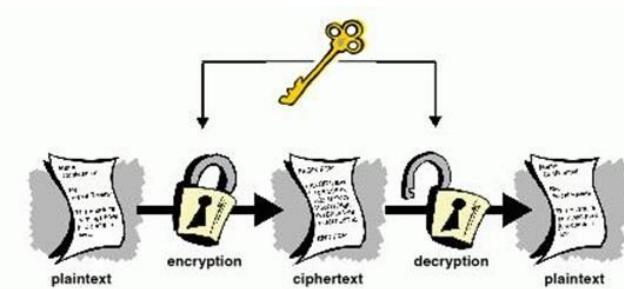


Lets Summarize:



Network Security

- Network security is the study of
 - How bad guys can **attack** networks
 - How to **defend** against such attacks
 - How to **design** systems that are immune to attacks
- Internet was **not** designed with security in mind
 - The protocols and system works best under mutual trust
 - Security is embedded in every layer







Internet of Things

Senior Design Project Course

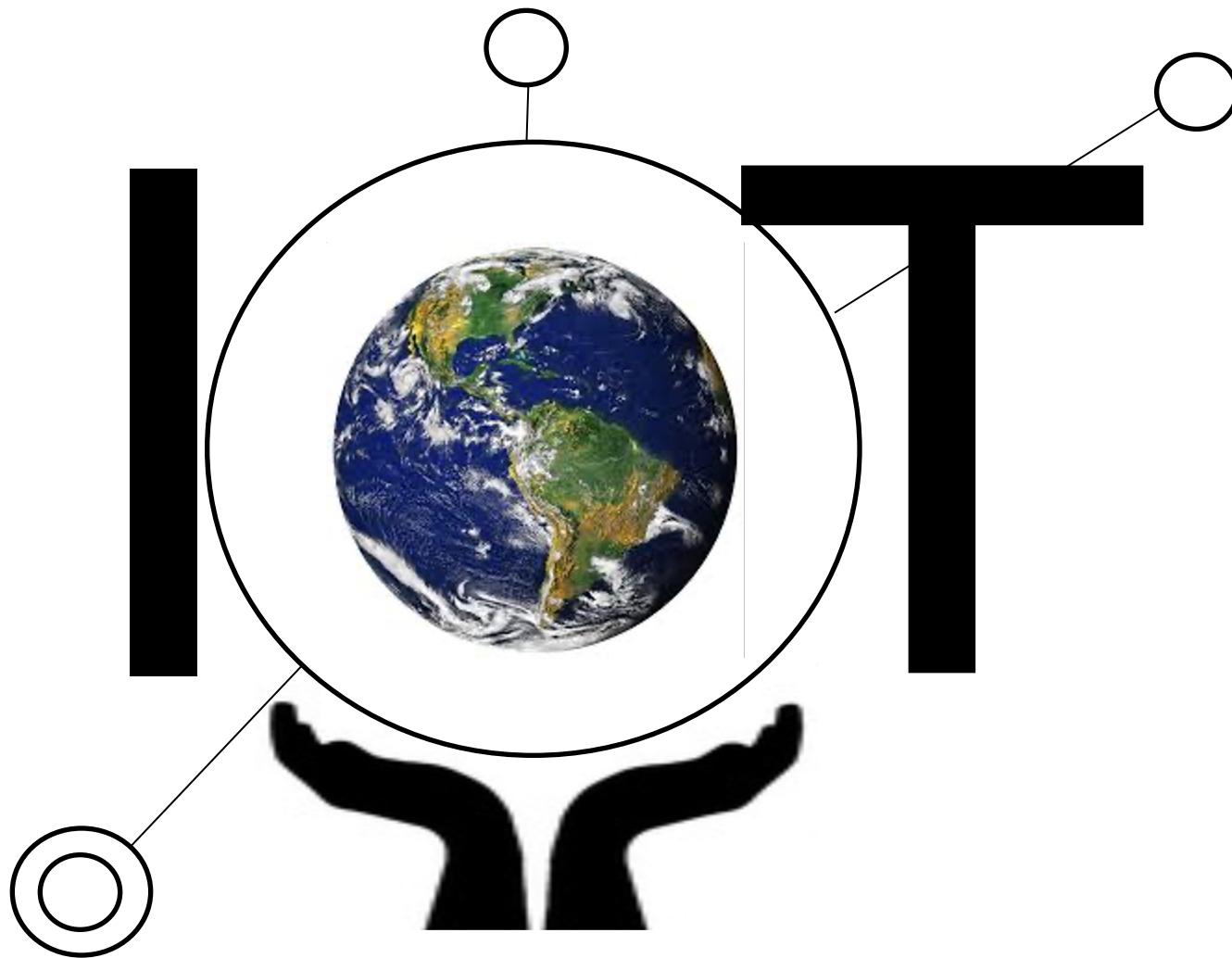
Communication – Part 2

MCU to Gateway

Lecturer: Avesta Sasan

University of California Davis

Lets Get Started:



Focus of Today's Lecture: (Review)

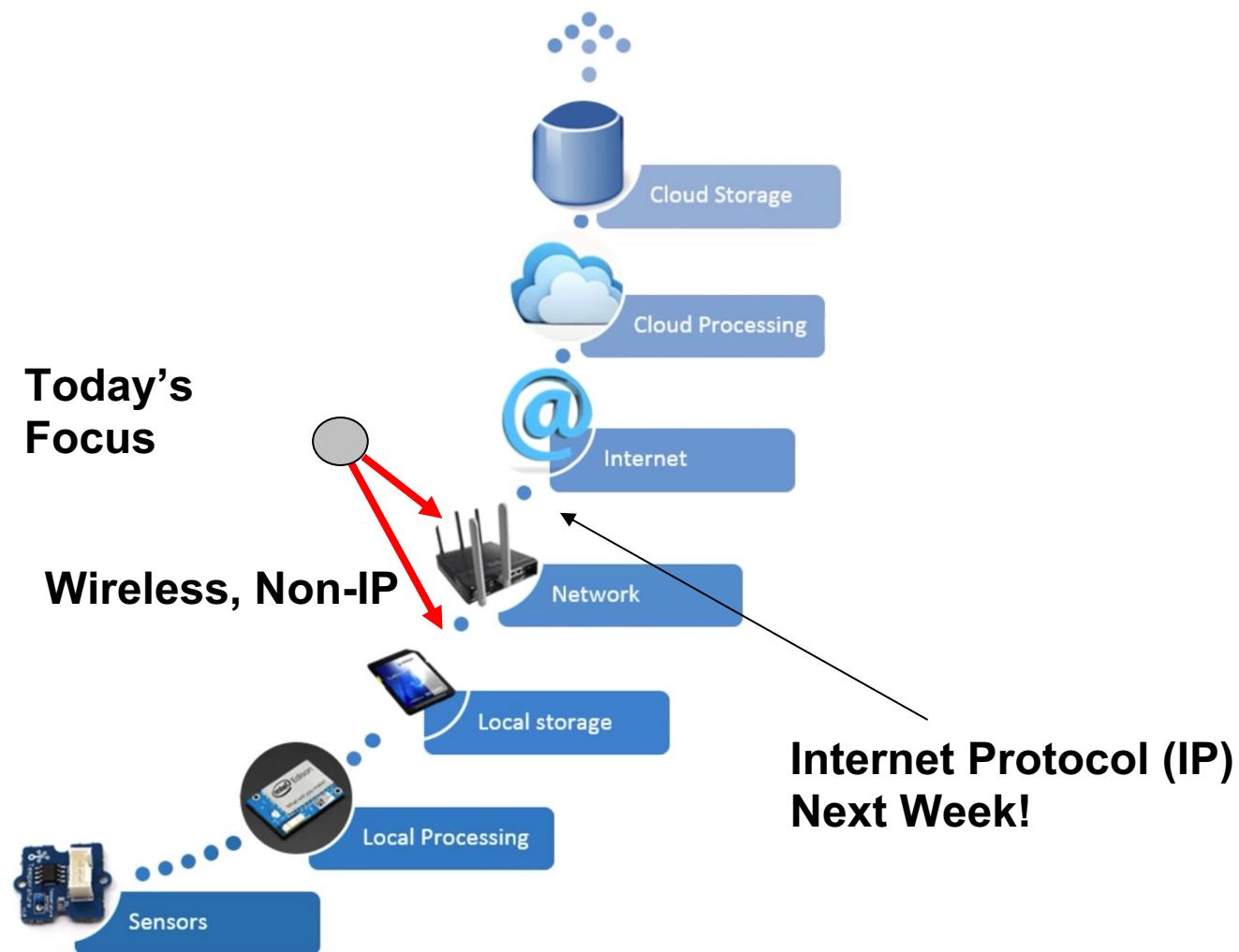


Image source: <http://www.cchc.cl/informacion-a-la-comunidad/industria-de-la-construccion/personaje/>

Why Using Gateway? (Review)

1. Lowering Power

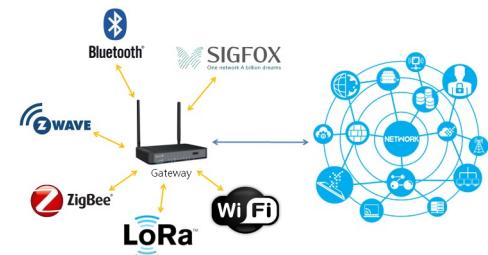
- ❑ Sensor sends the data to a gateway in short range (requires lower power)
- ❑ gateway send the data to cloud.

2. Supporting varying to/from sensor communication protocols

- ❑ Each sensor may have a different protocol.
- ❑ Gateway translates it to IP

3. Filtering the data

- ❑ Usually small fraction of data is usable.
- ❑ Filtering could be done at gateways.
(more resources than MCU, reduce communication size, reduce cloud computation load)



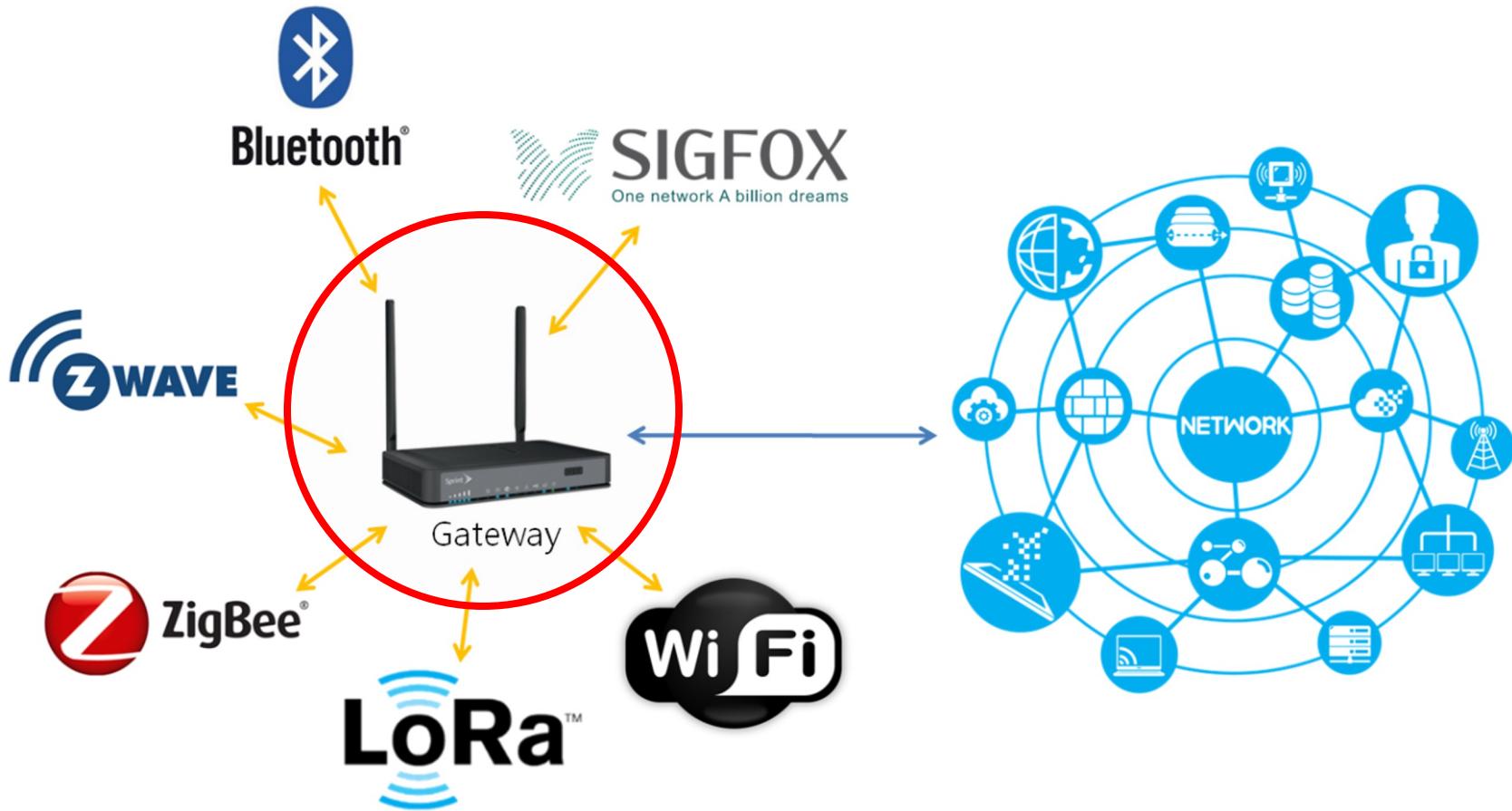
4. Reducing latency

- ❑ Many IoT devices are too small to do the processing themselves, and it takes too long to wait for cloud. Gateway (an intermediate computation layer) remedies this.

5. Improving security

- ❑ Can afford to make data transmission through gateway more secure.
- ❑ Prevent too many lightly secured sensors from being connected to the internet.

Gateway Supporting Various Protocols



Wireless Comm. To/From Gateways

- Wireless communication technologies used for:
 - Connecting the IoT device as local networks
 - Connecting these local networks (or individual IoT devices) to the Internet

- Some of popular wireless technologies are:
 -  □ Near Field Communication (**NFC**) 
 -  □ **Bluetooth** 
 -  □ **ZigBee** 
 -  □ Wireless Fidelity (**WiFi**)
 -  □ **Cellular network**
 -  □ Low Power Wide Area Network (**LPWAN**)
 - ...

[1] F. Samie, L. Bauer and J. Henkel, "IoT technologies for embedded computing: A survey," *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Pittsburgh, PA, 2016, pp. 1-10.

ZigBee



ZigBee®

ZigBee

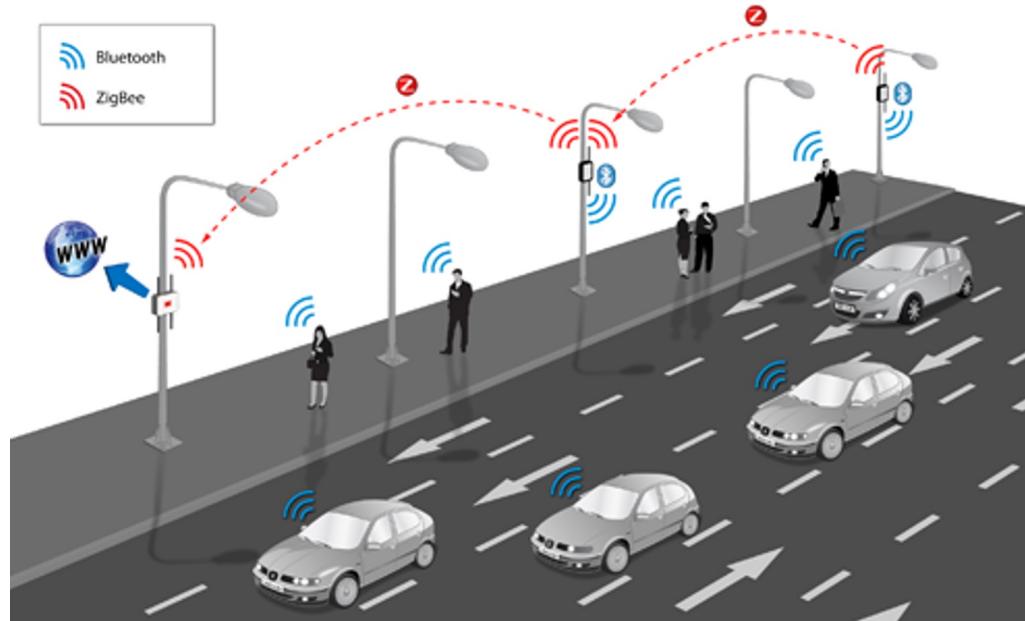
- Specification for a suite of high level communication protocols for Personal Area Network (PAN).

- ZigBee

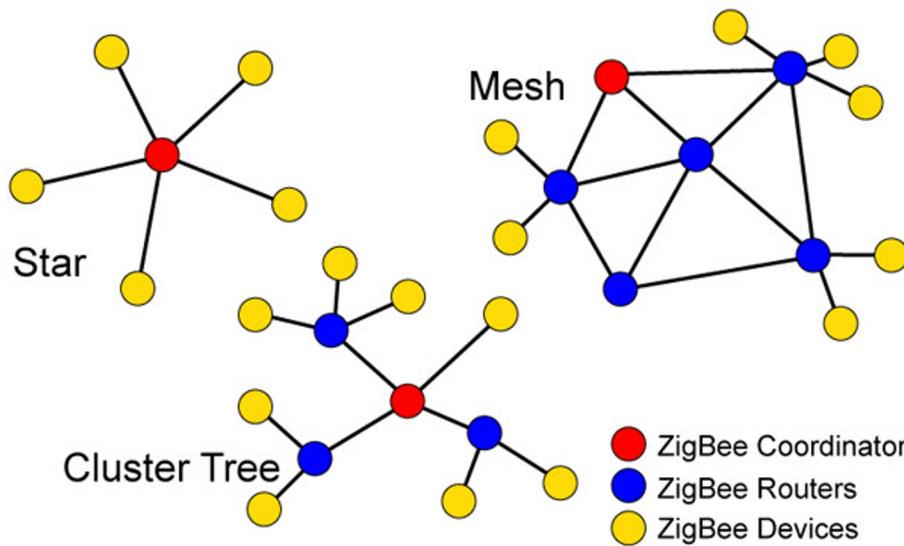
- Introduced in 1998
 - Standardized in 2003
 - Revised in 2006

- Characteristics:

- small-size
 - low-cost
 - low-power
 - **wide transmission range**, depending on the output power
 - Its low power consumption limits transmission distances to 10–100 meters



- Supports different network topologies (e.g. mesh, star, tree).
- **ZigBee coordinator** initiates and manages the devices
- **Mesh and Tree** networks can transmit data over long distances by passing data through a network of intermediate devices to reach further distant.



ZigBee

- Specification for a suite of **high level communication protocols** for Personal Area Network (**PAN**) used in **low data rate** applications that require **long battery life** and **secure networking**
- ZigBee networks are secured by 128 bit symmetric encryption keys
- has a defined rate of 250 kbit/s, best suited for intermittent data transmissions from a sensor or input device
- Faces market barriers
 - **Compete with BLE** that provides higher bandwidth at a lower energy consumption



Zigbee and Arduino

- **Xbee Shield**
- **Wireless SD shield**

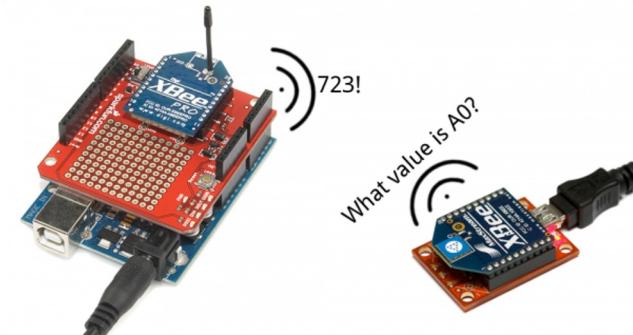
```
#include <SoftwareSerial.h>
// XBee's DOUT (TX) is connected to pin 2 (Arduino's Software RX)
// XBee's DIN (RX) is connected to pin 3 (Arduino's Software TX)
SoftwareSerial XBee(2, 3); // RX, TX

void setup()
{
    // Set up both ports at 9600 baud. This value is most important
    // for the XBee. Make sure the baud rate matches the config
    // setting of your XBee.
    XBee.begin(9600);
    Serial.begin(9600);
}
```

```
void loop()
{
    if (Serial.available())
    { // If data comes in from serial monitor, send it out to XBee
        XBee.write(Serial.read());
    }
    if (XBee.available())
    { // If data comes in from XBee, send it out to serial monitor
        Serial.write(XBee.read());
    }
}
```

CLICK
HERE
↓

More information



WiFi



Wireless Fidelity (WiFi)

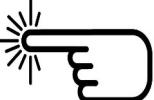
- **Wi-Fi** or **WiFi** is a technology for wireless local area networking (WLAN) based on IEEE 802.11 standard.
- Widely adopted
 - Internet access
 - City-wide Wi-Fi
 - Campus-wide Wi-Fi, ...
- Range of about **20 meters** (66 feet) indoors and a greater range outdoors.

Wireless Fidelity (WiFi)

- Most commonly uses the 2.4 gigahertz (12 cm) UHF and 5GHz (6cm) SHF ISM radio bands.
- Flavors:
 - Conventional WiFi (IEEE 802.11 b/g/n)
 - High bandwidth
 - High energy consumption
 - Unsuitable for ultra-low-power IoT devices
 - Low-power WiFi (802.11 ah) or HaLow
 - Extend the range of transmission
 - less data rate
 - Lower power consumption
 - Less interference with existing wireless networks



WiFi and Arduino

- How to connect to a WEP encrypted network named "yourNetwork" with a hex key of "ABBADEAF01", and a key index of 0
- Reference: [Arduino WiFi Shield](#) 

```
#include <WiFi.h>

char ssid[] = "yourNetwork";      // your network SSID (name)
char key[] = "ABBADEAF01";        // your network key
int keyIndex = 0;                //your network key Index number
int status = WL_IDLE_STATUS;     // the Wifi radio's status

void setup() {
  // initialize serial:
  Serial.begin(9600);

  // attempt to connect using WEP encryption:
  Serial.println("Attempting to connect to WEP network...");
  status = WiFi.begin(ssid, keyIndex, key);

  // if you're not connected, stop here:
  if (status != WL_CONNECTED) {
    Serial.println("Couldn't get a wifi connection");
    while(true);
  }
  // if you are connected, print out info about the connection:
  else {
    Serial.println("Connected to network");
  }
}

void loop() {
  // do nothing
}
```

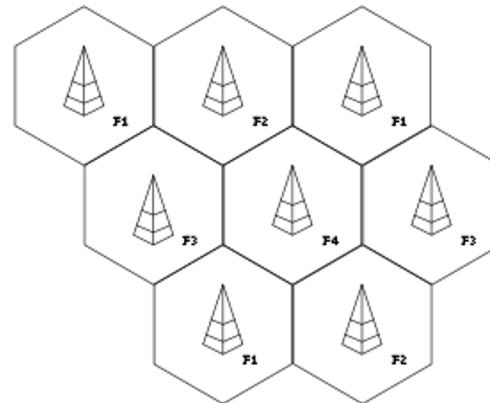


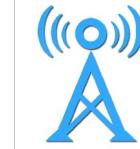
Cellular Network



Cellular Network

- A **cellular network** or **mobile network** is a communication network where the last link is wireless
 - E.g., 3G, LTE, 4G, and 5G
- The network is **distributed over** land areas called **cells**
- Each cells use a **different set of frequencies** from neighboring cells
 - to avoid interference
 - to provide guaranteed service quality within each cell





Cellular Network

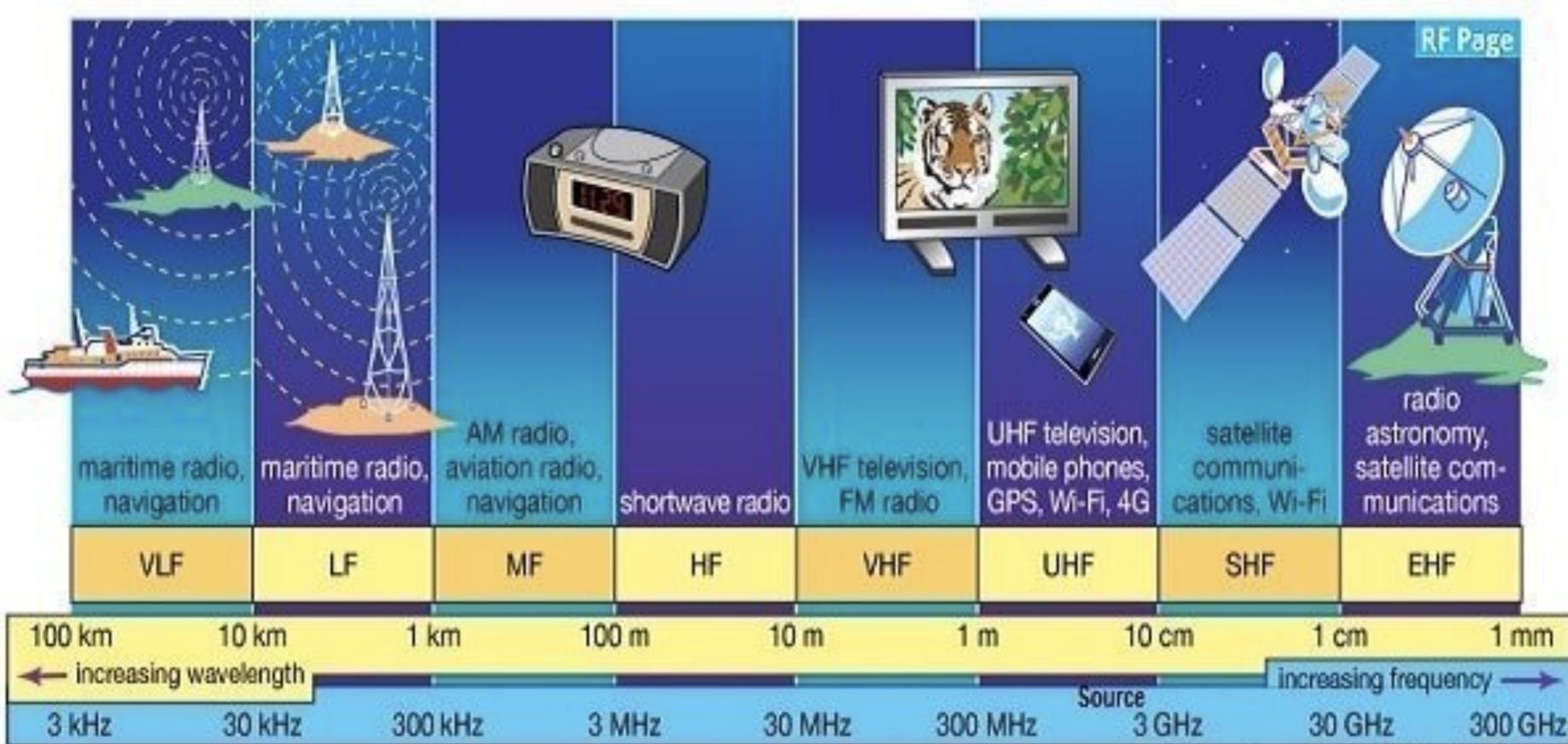
- Cellular networks offer a number of desirable features
 - **More capacity** than a single large transmitter
 - the same frequency can be used for multiple links as long as they are in different cells
 - Mobile devices use **less power** than with a single transmitter or satellite
 - the cell towers are closer
 - **Larger coverage** area than a single terrestrial transmitter
 - additional cell towers can be added indefinitely and are not limited by the horizon
- When it come to IoT
 - Provide reliable high-speed connectivity to the Internet
 - **BUT** It has high power consumption
 - **BUT** It is not suitable for M2M

Low Power Wide Area Network (LPWAN)

- Suited for **low power** applications with **very long range** transmission
- Support up to **10 Km distance** between end-nodes and gateway
- **Very low data rate** (<1 Kbps)
- Operates in **sub-GHz bands**
 - No globally available band for LPWAN in sub-GHz
- Main technologies:
 - SigFox
 - LoRaWAN
 - Weightless



Radio Frequency bands



Source: Encyclopaedia Britannica, Inc.

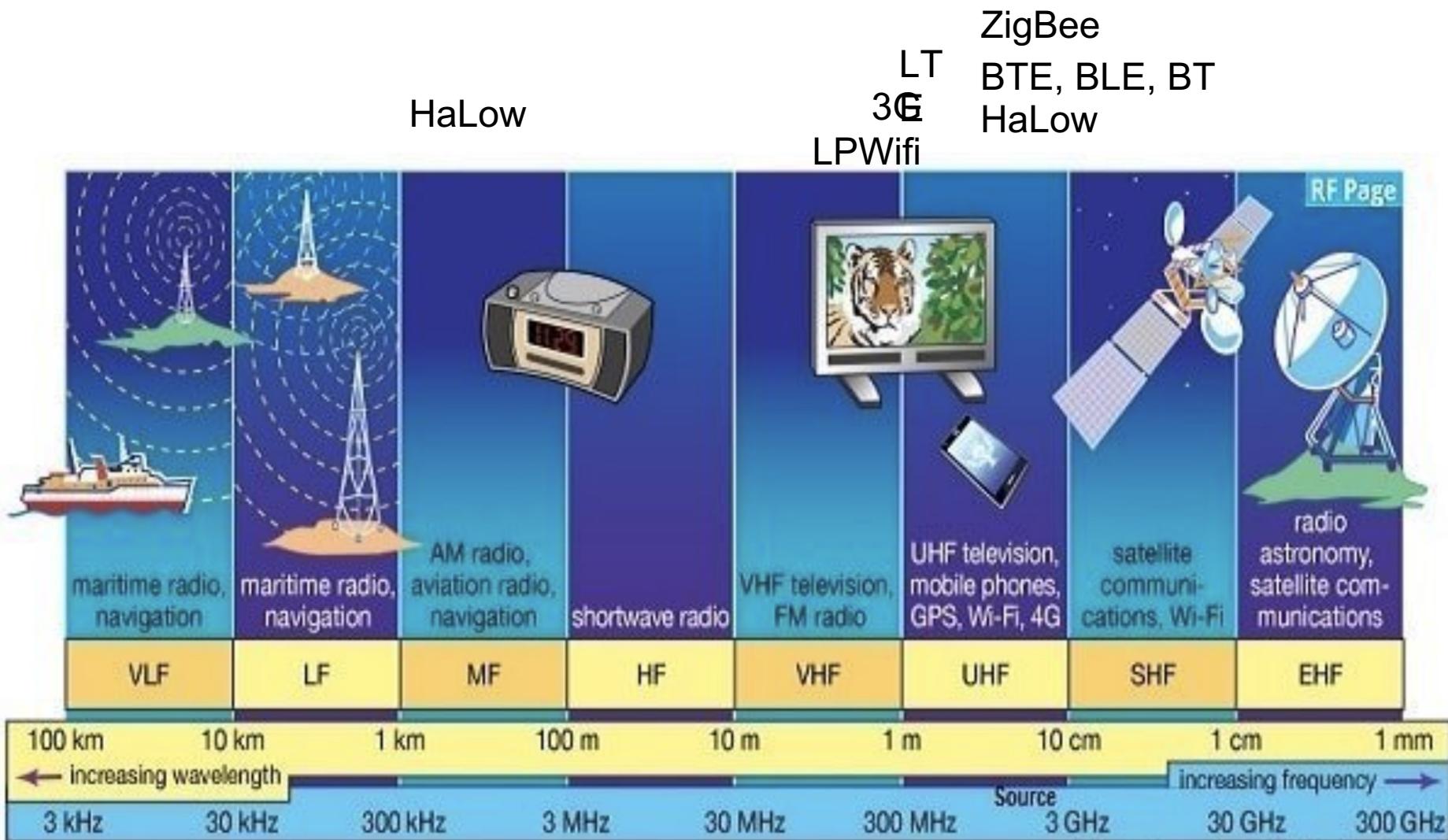
Summary of Comm. Technologies

- The communication technology should be chosen based on
 - Area** constrain (Area to implement the protocol, Area used for antenna)
 - Power** constrain (Range of broadcast)
 - Performance** constrain (Requirement for data transmission rate)
 - Noise** and interference
 - E.g. Bluetooth, WiFi, and ZigBee may face interference due to the coexistence of other devices working at the same frequency

		NFC	Bluetooth	BLE	BT v5	ZigBee	HaLow 802.11 b/g/n	LP WiFi 802.11ah	LPWAN	Cellular network		
Range	indoor	<0.2 m	1–100 m	~100 m	<300 m	<20 m	<70 m	<700 m	<10 Km	3G	LTE	
	outdoor					<1500 m	<230 m	<1000 m				
Bit rate [Mbps]	0.424	1–3	1	2	0.25	>1	0.15–40	<0.05	0.17	75–300		
Throughput [Mbps]	0.22	1.5	0.30	1.5	0.15	2–50	>0.1	<0.05	NA			
freq. [GHz]	0.014	2.4–2.5	2.4–2.5	2.4	2.4	2.4/5	0.9	sub-GHz	0.8–1.9	2.1		
Network topology	p2p	scatternet	star, scatternet	NA	star, tree, mesh	star	star	star	NA			

F. Samie, L. Bauer and J. Henkel, "IoT technologies for embedded computing: A survey," *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Pittsburgh, PA, 2016, pp. 1-10.

Radio Frequency bands



Source: Encyclopaedia Britannica, Inc.

Timing of Data Transmission:

■ **Continuous**

- Send or receive data continuously
- E.g. real-time systems such as health monitoring.

■ **Sporadic**

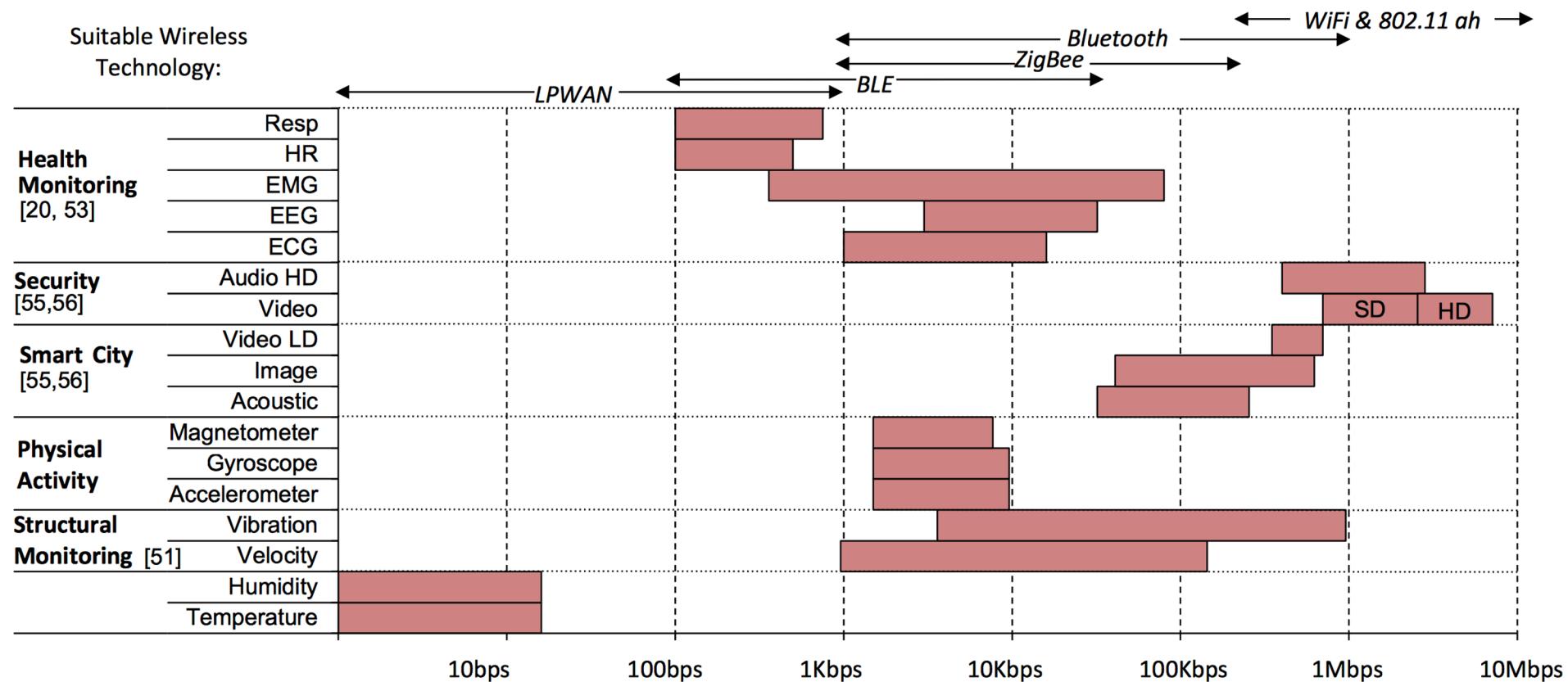
- device collects and stores the data
- transmits data whenever the connection is available

■ **On-demand**

- **User driven**
 - IoT device can be requested by the operator to send the collected data
- **Event driven**
 - Communication is done once a specific event happens

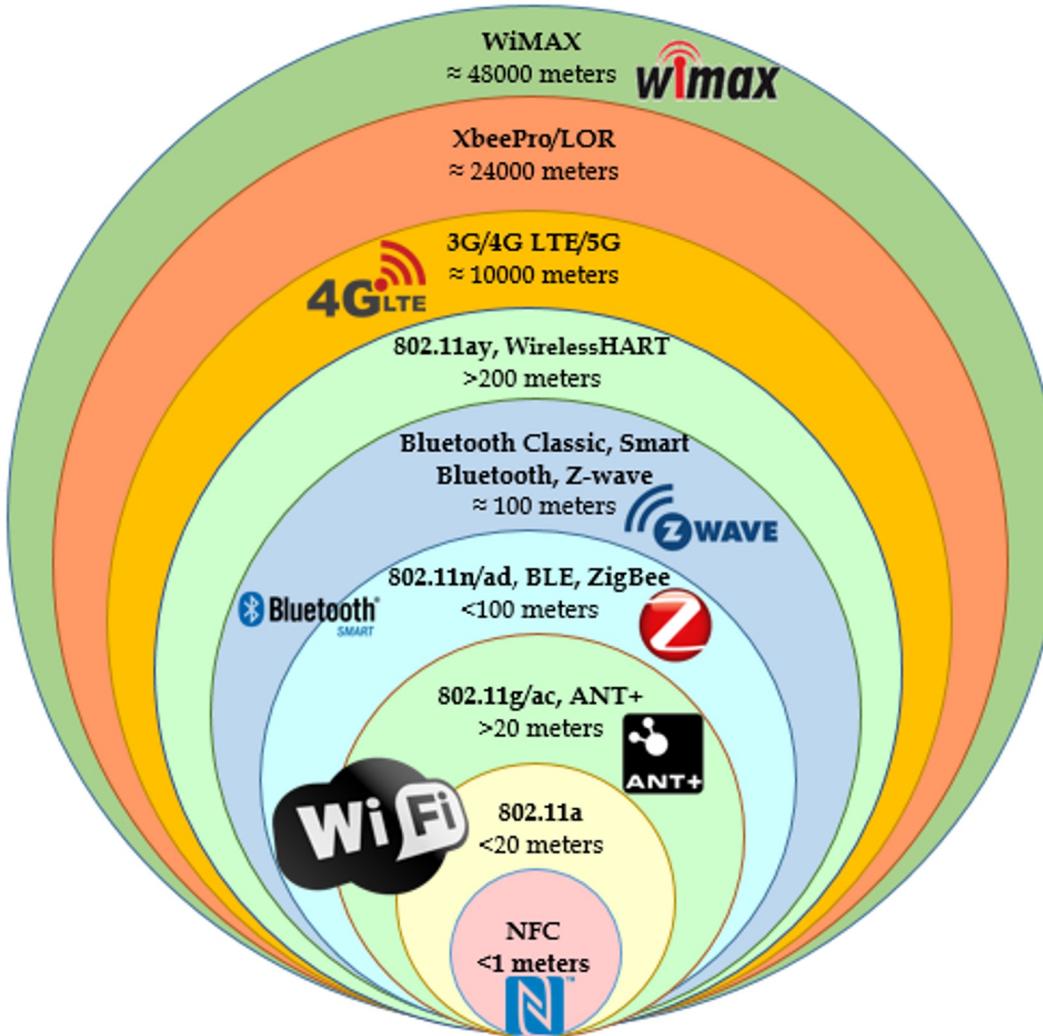
Does Data Rate Decide the Protocol?

- Which communication protocol for which sensing rates?



F. Samie, L. Bauer and J. Henkel, "IoT technologies for embedded computing: A survey," *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Pittsburgh, PA, 2016, pp. 1-10.

Does Communication Range Decide the Protocol?



THE END



Internet of Things

Senior Design Project Course

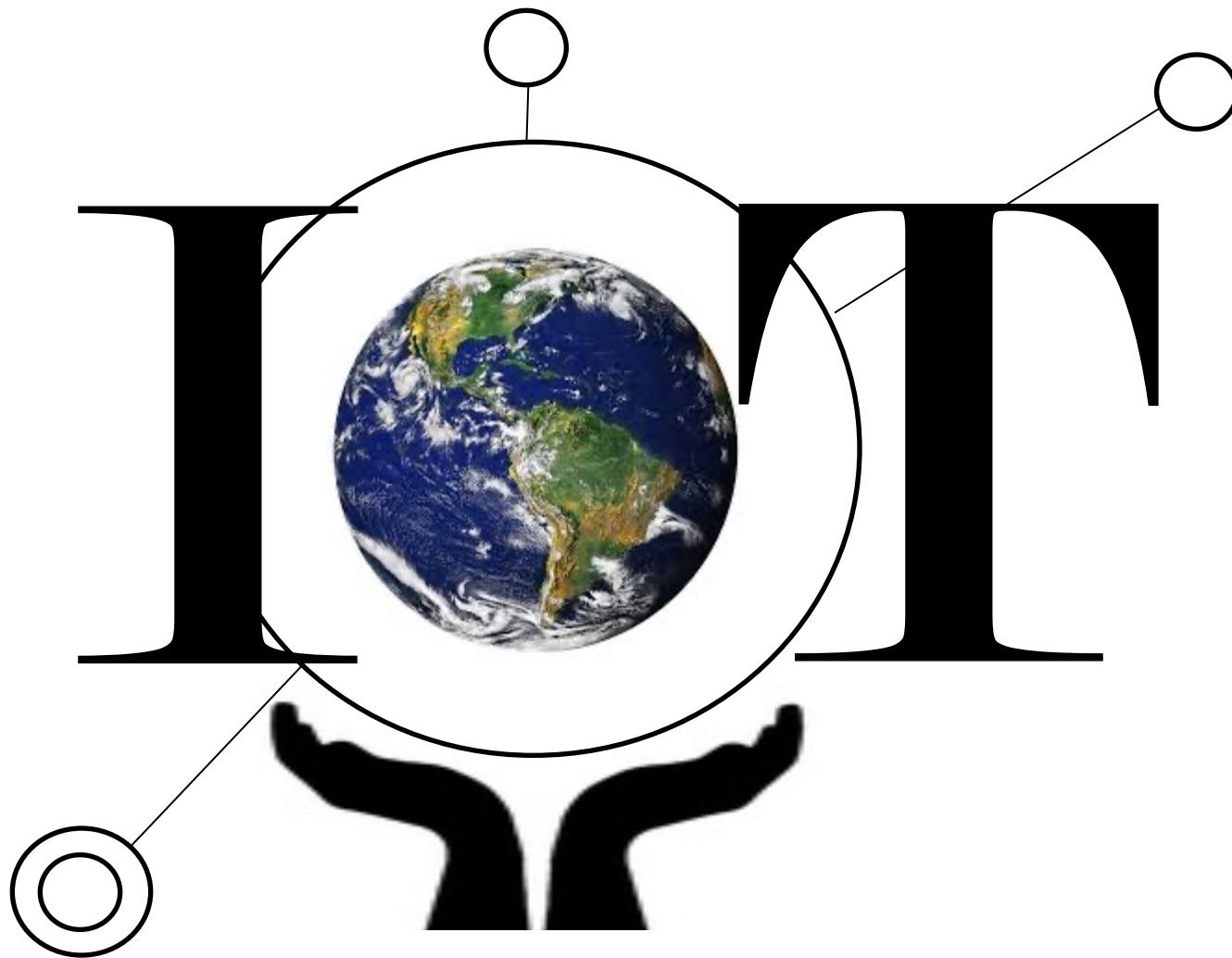
Communication - Part 1

MCU to Gateway

Lecturer: Avesta Sasan

University of California Davis

Lets Get Started:



Focus of Today's Lecture:

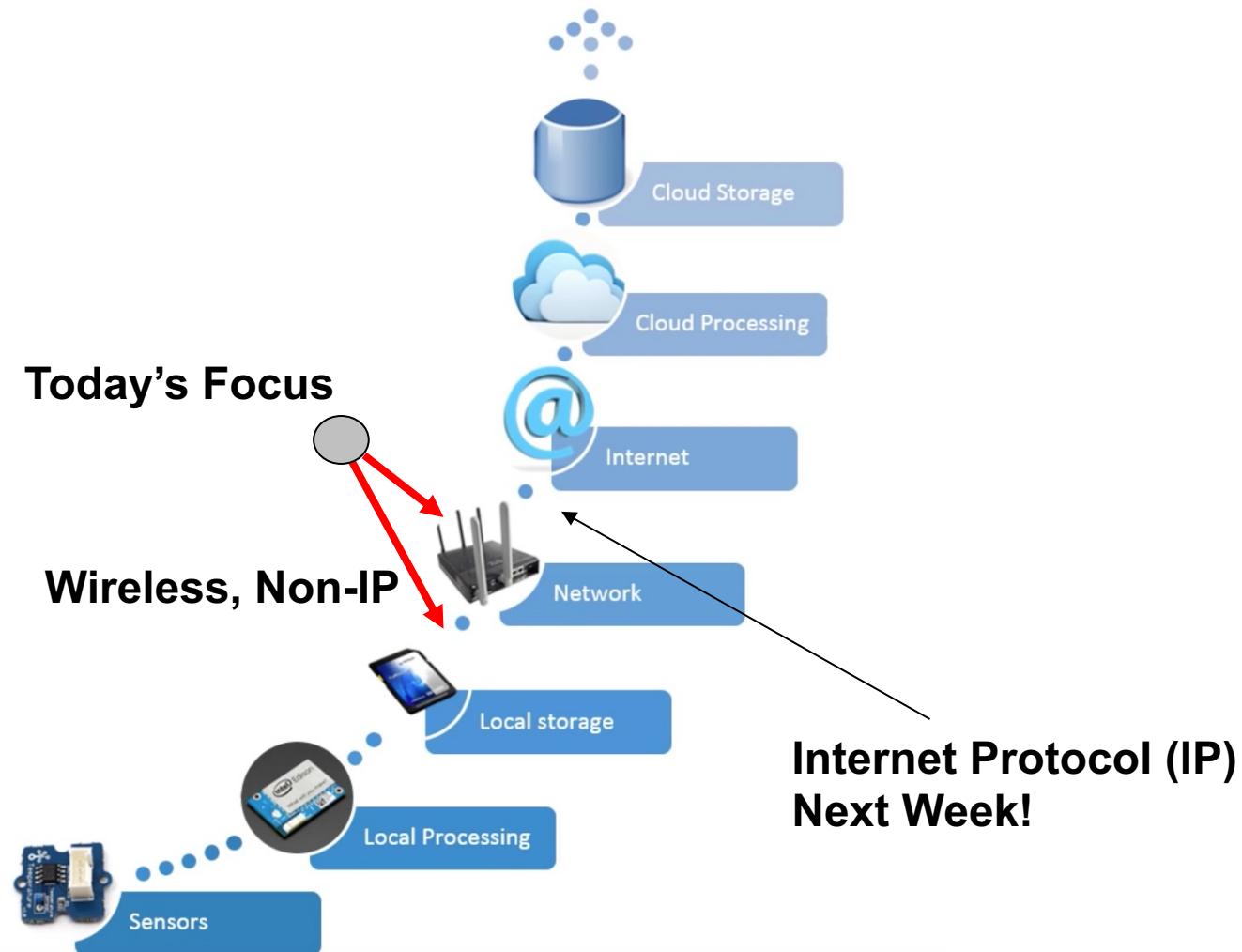


Image source: <http://www.cchc.cl/informacion-a-la-comunidad/industria-de-la-construccion/personaje/>

Why Using Gateway?

1. Lowering Power

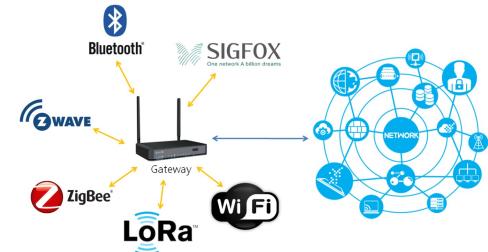
- ❑ Sensor sends the data to a gateway in short range (requires lower power)
- ❑ gateway send the data to cloud.

2. Supporting varying to/from sensor communication protocols

- ❑ Each sensor may have a different protocol.
- ❑ Gateway translates it to IP

3. Filtering the data

- ❑ Usually small fraction of data is usable.
- ❑ Filtering could be done at gateways.
(more resources than MCU, reduce communication size, reduce cloud computation load)



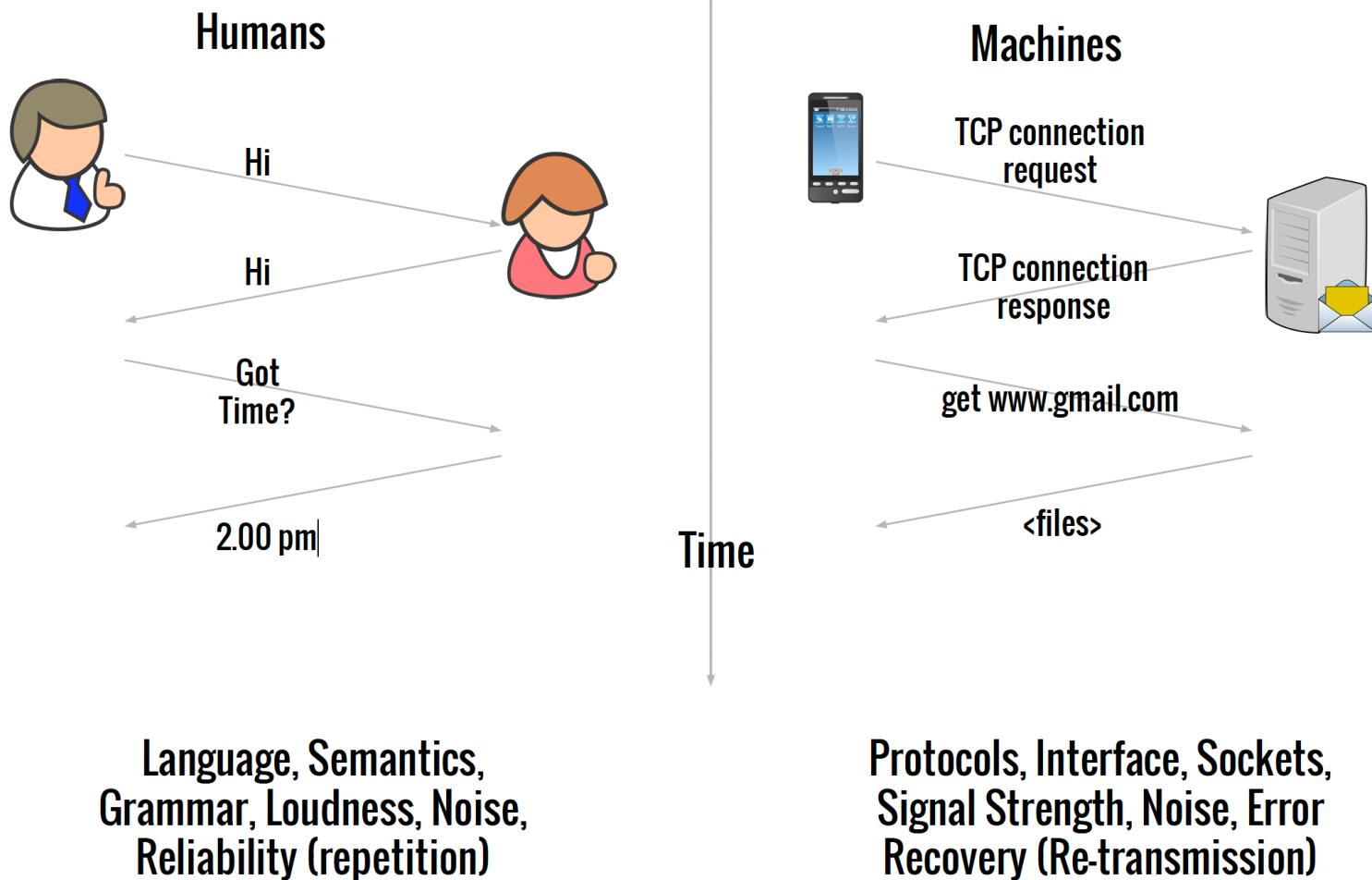
4. Reducing latency

- ❑ Many IoT devices are too small to do the processing themselves, and it takes too long to wait for cloud. Gateway (an intermediate computation layer) remedy this.

5. Improving security

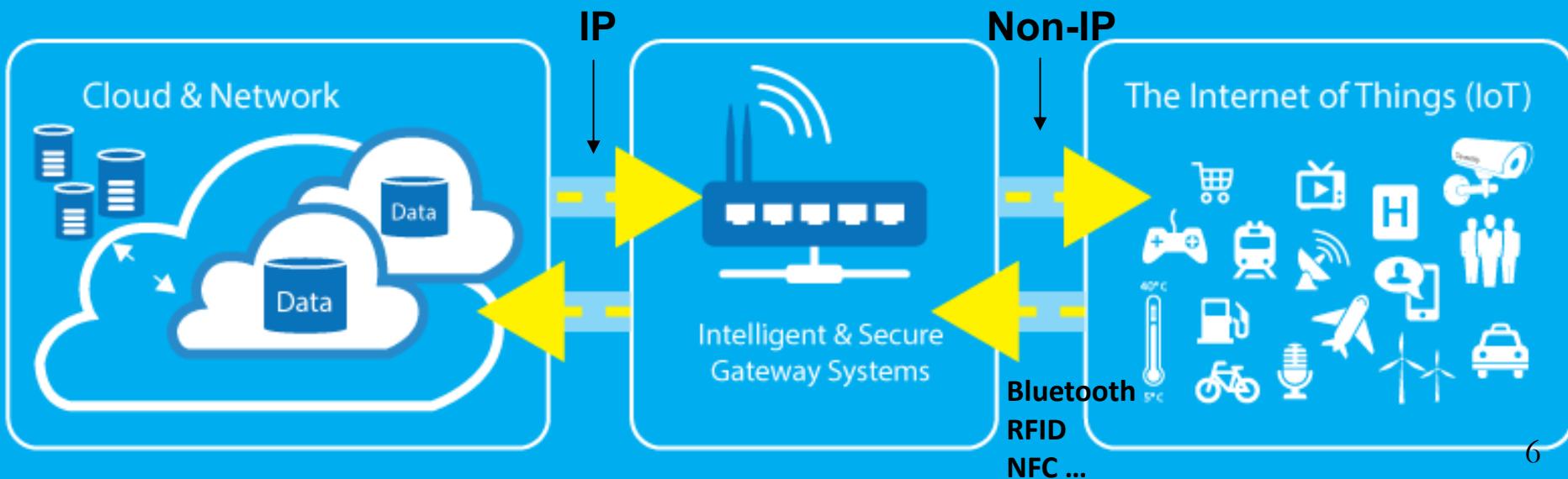
- ❑ Can afford to make data transmission through gateway more secure.
- ❑ Prevent too many lightly secured sensors to be connected to internet.

What is a Protocol



Gateway and Protocol Translation

- IoT devices can connect to Internet using **Internet Protocol (IP)**
Problem:
 - IP stack is very **complex** and demands a large amount of **power** and **memory** from the connecting devices.
 - **Wired**, hence **mobility** is limited!
- **Gateway removes the need for direct connection to internet.**
 - IoT devices can also connect locally through non-IP networks, which consume less power and offer larger mobility, and connect to the Internet via a smart gateway. It also enhances mobility of IoT devices.



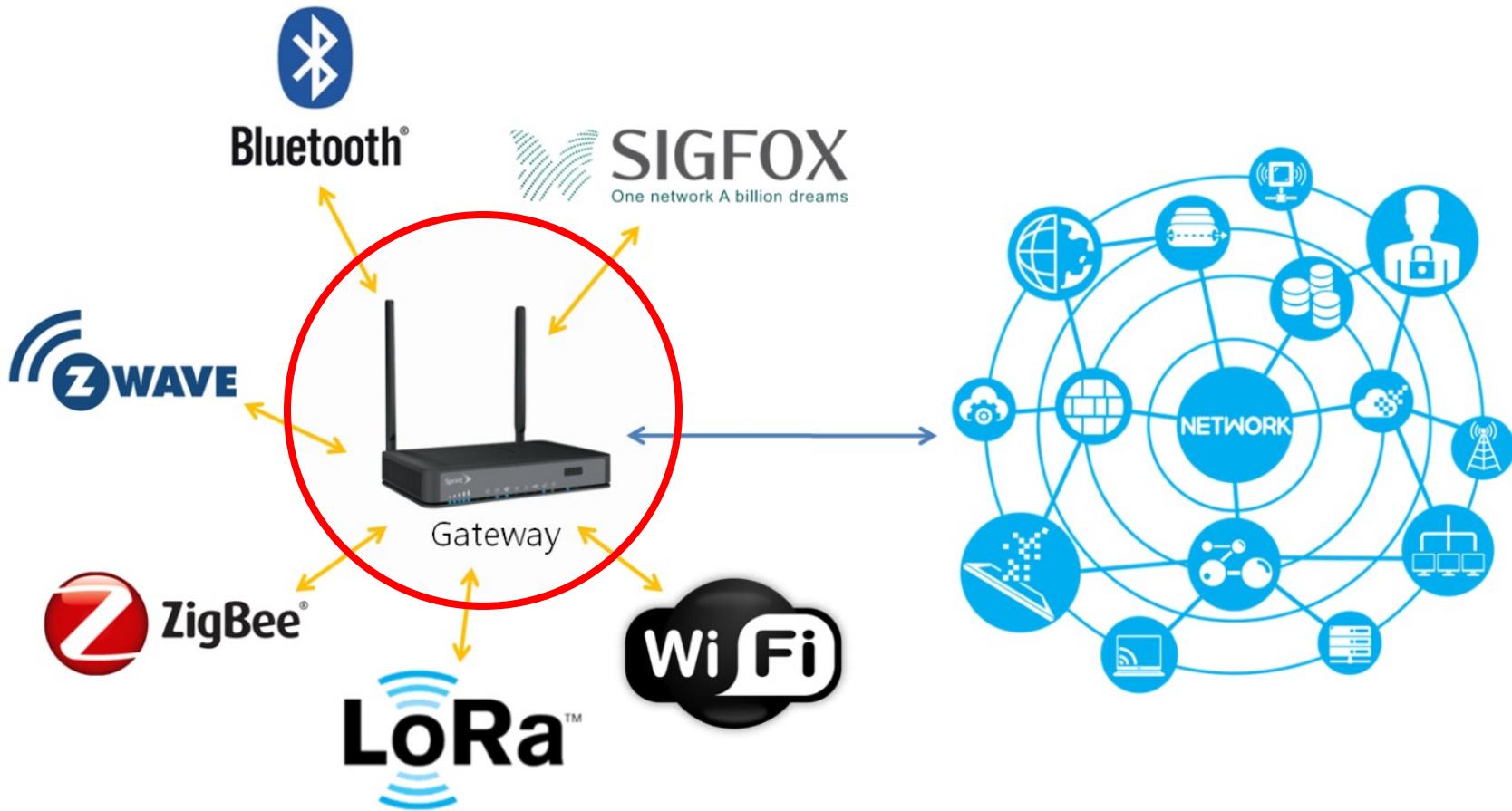
Protocol Interfacing IoT Objects

- Now lets look at the communication between gateway and IoT devices.

Non-IP



Gateway Supporting Various Protocols



Wireless Comm. To/From Gateways

- Wireless communication technologies used for:
 - Connecting the IoT device as local networks
 - Connecting these local networks (or individual IoT devices) to the Internet

- Some of popular wireless technologies are:
 -  □ Near Field Communication (**NFC**)
 -  □ **Bluetooth**
 -  □ **ZigBee**
 -  □ Wireless Fidelity (**WiFi**)
 -  □ **Cellular network**
 -  □ Low Power Wide Area Network (**LPWAN**)
 - ...

[1] F. Samie, L. Bauer and J. Henkel, "IoT technologies for embedded computing: A survey," *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Pittsburgh, PA, 2016, pp. 1-10.

NFC



Near Field Communication (NFC)

- **Short-range** (4~20 cm)
- **Low-speed** connection with simple setup
 - supported data rates: 106, 212 or 424 kbit/s.
 - operates at 13.56 MHz on ISO/IEC 18000-3 air interface.
- Can be used **to bootstrap** more capable wireless connections
- Has a tag that can contain small amount of data
 - can be read-only
 - can be re-writable
- Popular current usage:
 - contactless payment systems
 - sharing contacts, photos, videos or files



NFC



Near Field Communication (NFC)

- NFC devices are **Full-Duplex**
 - able to receive and transmit data at the same time.
- **Modes of operation:**
 - **Passive**
 - the initiator device provides a carrier field.
 - the target device answers by modulating the existing field.
 - the target device may draw its operating power from the initiator-provided electromagnetic field, making the target device a transponder.
 - **Active**
 - Both initiator and target device communicate by alternately generating their own fields.
 - A device deactivates its RF field while it is waiting for data.
 - both devices typically have power supplies.

NFC Security Problems!

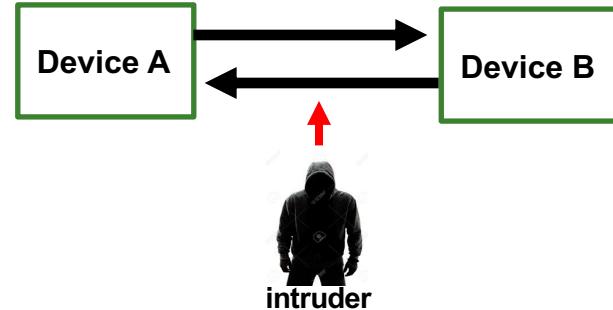
Eavesdrop:

/'ēvz, dräp/

To secretly listen to a conversation

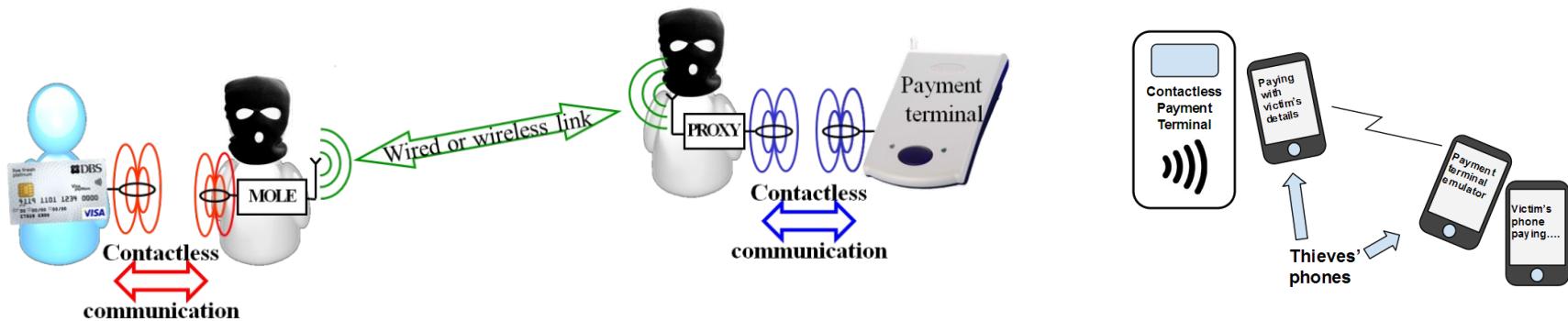
No protection against eavesdropping

- Can be vulnerable to data modifications
- An attacker can typically eavesdrop within
 - 10 m for active devices
 - 1 m for passive devices
- Can use higher-layer cryptographic protocols (e.g. SSL) to establish secure channel



Relay attacks are feasible

- The adversary forwards the request of the reader to the victim
- Relays its answer to the reader in real time pretending to be the owner of the victim's smart card

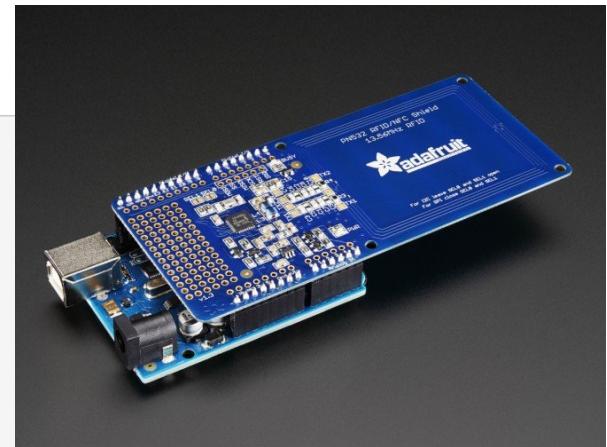


Using NFC with Arduino

- Read and write on NFC tag using Arduino
- You can add NFC capabilities by adding a shield to Arduino:
 - Adafruit PN532 RFID/NFC Shield

```
#include <Wire.h>
#include <PN532_I2C.h>
#include <PN532.h> // The following files are included in the libraries Installed
#include <NfcAdapter.h>

PN532_I2C pn532_i2c(Wire);
NfcAdapter nfc = NfcAdapter(pn532_i2c); // Indicates the Shield you are using
```



```
void setup(void) {
    Serial.begin(9600);
    Serial.println("NFC TAG READER"); // Header used when using the serial monitor
    nfc.begin();
}
```

Using NFC with Arduino

```
void loop(void) {
    Serial.println("\nScan your NFC tag on the NFC Shield\n"); // Command so that you an others
    will know what to do

    if (nfc.tagPresent())
    {
        NfcTag tag = nfc.read();
        Serial.println(tag.getTagType());
        Serial.print("UID: ");Serial.println(tag.getUidString()); // Retrieves the Unique Identific
        ation from your tag

        if (tag.hasNdefMessage()) // If your tag has a message
        {

            NdefMessage message = tag.getNdefMessage();
            Serial.print("\nThis Message in this Tag is ");
            Serial.print(message.getRecordCount());
            Serial.print(" NFC Tag Record");
            if (message.getRecordCount() != 1) {
                Serial.print("s");
            }
            Serial.println(".");

            // If you have more than 1 Message then it wil cycle through them
            int recordCount = message.getRecordCount();
            for (int i = 0; i < recordCount; i++)
            {
                Serial.print("\nNDEF Record ");Serial.println(i+1);
                NdefRecord record = message.getRecord(i);

                int payloadLength = record.getPayloadLength();
                byte payload[payloadLength];
                record.getPayload(payload);

                String payloadAsString = ""; // Processes the message as a string vs as a HEX value
                for (int c = 0; c < payloadLength; c++) {
                    payloadAsString += (char)payload[c];
                }
                Serial.print(" Information (as String): ");
                Serial.println(payloadAsString);

                String uid = record.getId();
                if (uid != "") {
                    Serial.print(" ID: ");Serial.println(uid); // Prints the Unique Identification of th
                    e NFC Tag
                }
            }
        }
        delay(10000);
    }
}
```

Bluetooth



Bluetooth

- Technology designed and marketed by the Bluetooth Special Interest Group (SIG)
- A wireless technology standard for exchange of data over **short distances**.
- Uses short-wavelength Ultra High Frequency (**UHF**) radio waves in the industrial, scientific and medical (**ISM**) band from 2.4 to 2.485 GHz.
 - ISM is globally unlicensed (but not unregulated) band.
- Invented by telecom vendor Ericsson in 1994
- managed by the Bluetooth Special Interest Group (SIG)
- Bluetooth is a packet-based protocol.
- Master slave structure
- Multiple flavors:
 - Classic Bluetooth
 - Bluetooth Low Energy (BLE)
 - Bluetooth 5.0 (BT v5)
 - ...



Bluetooth Flavors

- **Classic Bluetooth**
 - High throughput and bandwidth
 - Suitable for data stream applications
 - Limitations
 - Limited number of nodes in the network (up to seven slaves)

- **Bluetooth Low Energy (BLE)**
 - A.K.A. Bluetooth smart
 - Originally introduced under the name **Wibree** by Nokia in 2006
 - Short-range, low bandwidth, and low latency
 - Lower power consumption
 - Lower setup time
 - Supports unlimited number of nodes



Bluetooth Classic vs BLE

■ Choose wisely!

	Bluetooth Classic	Bluetooth Low Energy
Range (theoretical)	100 m	> 100 m
Power consumption	1 W	0.01 to 0.5 W
Peak current consumption	<30 mA	<15 mA
Data rate	1-3 Mbit/s	1 Mbit/s
Radio Frequencies	2.4 GHz	2.4 GHz
Focus	Wireless protocol for short range data exchange	Low power consumption – periodic exchange of small amounts of data
Use Cases	Wireless speakers, headsets	Wearable devices, smart pay systems

Bluetooth Flavors

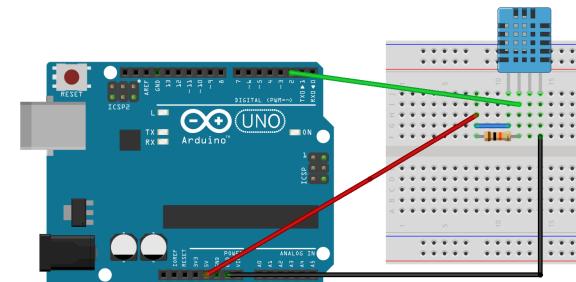
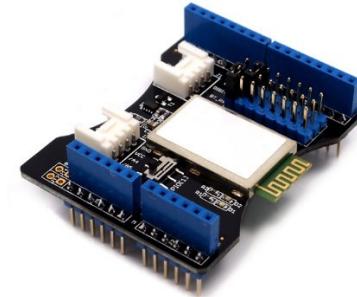
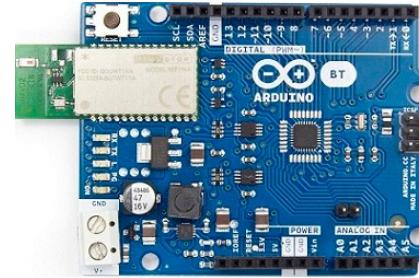
■ Bluetooth 5.0 (BT v5)

- Focus on IoT
- The Bluetooth SIG officially unveiled Bluetooth 5 on June 16, 2016
 - Iphone 8, Samsung Galaxy S8, Xpheria XZ (sony) first to lunch with BT v5.0
- Doubles the speed of low energy connections (2MBps)
- Quadruple the range
 - Forward Error Correction (FEC)
- Eight-fold increase in data broadcasting
- Today: BT v5.3 (as of July 2021)



Bluetooth with Arduino

- Three connectivity options:
 - **Arduino BT**: an MCU with embedded Bluetooth capabilities.
 - Add a **Bluetooth shield**.
 - Connect a standalone BLE module
 - Connect using UART, I2C, SPI, ...
 - (1) Example code (control Arduino with B)
 - (2) Example code





Internet of Things

Senior Design Project Course

Processing - Part 2

Lecturer: Avesta Sasan

University of California Davis

Focus of Today's Lecture: MCU

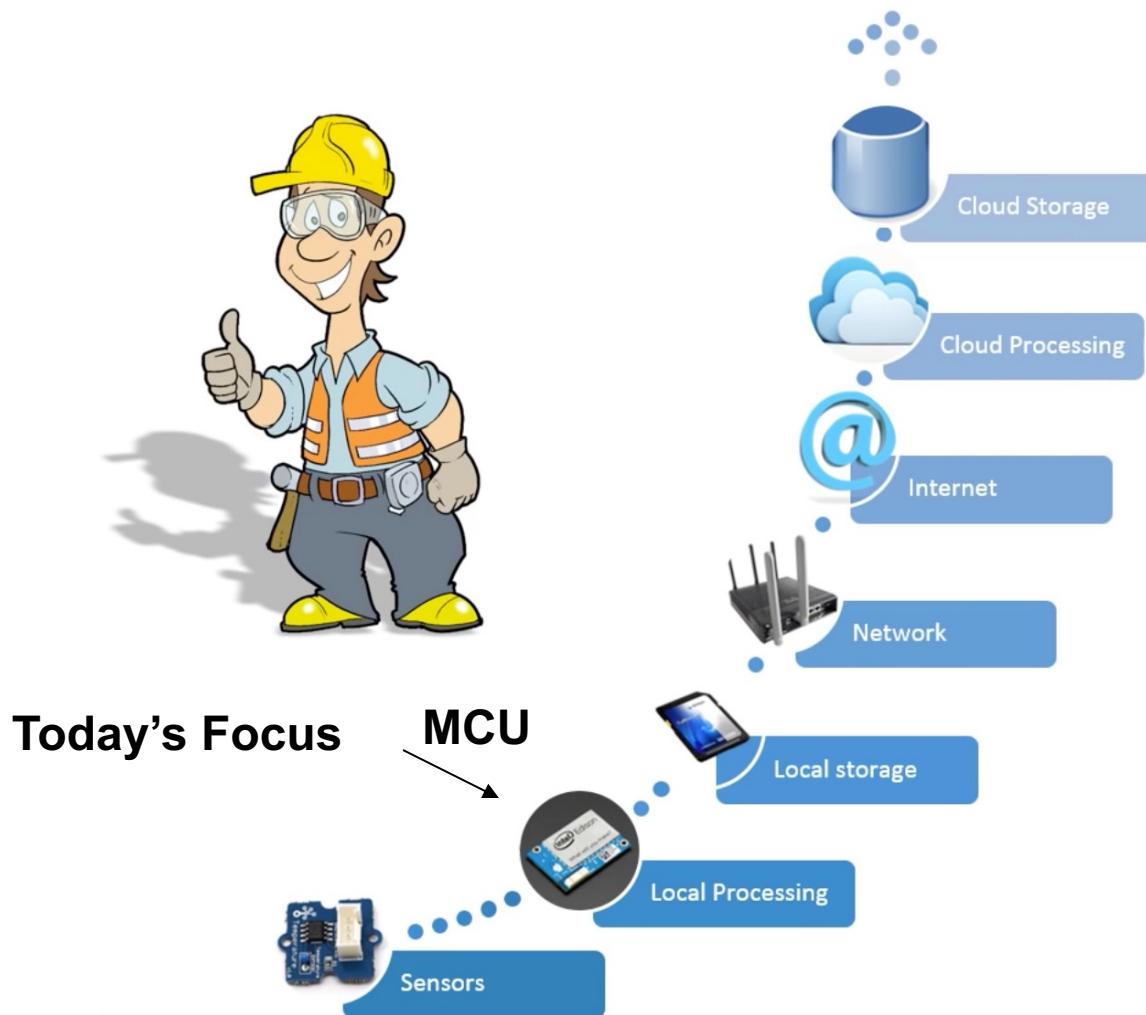
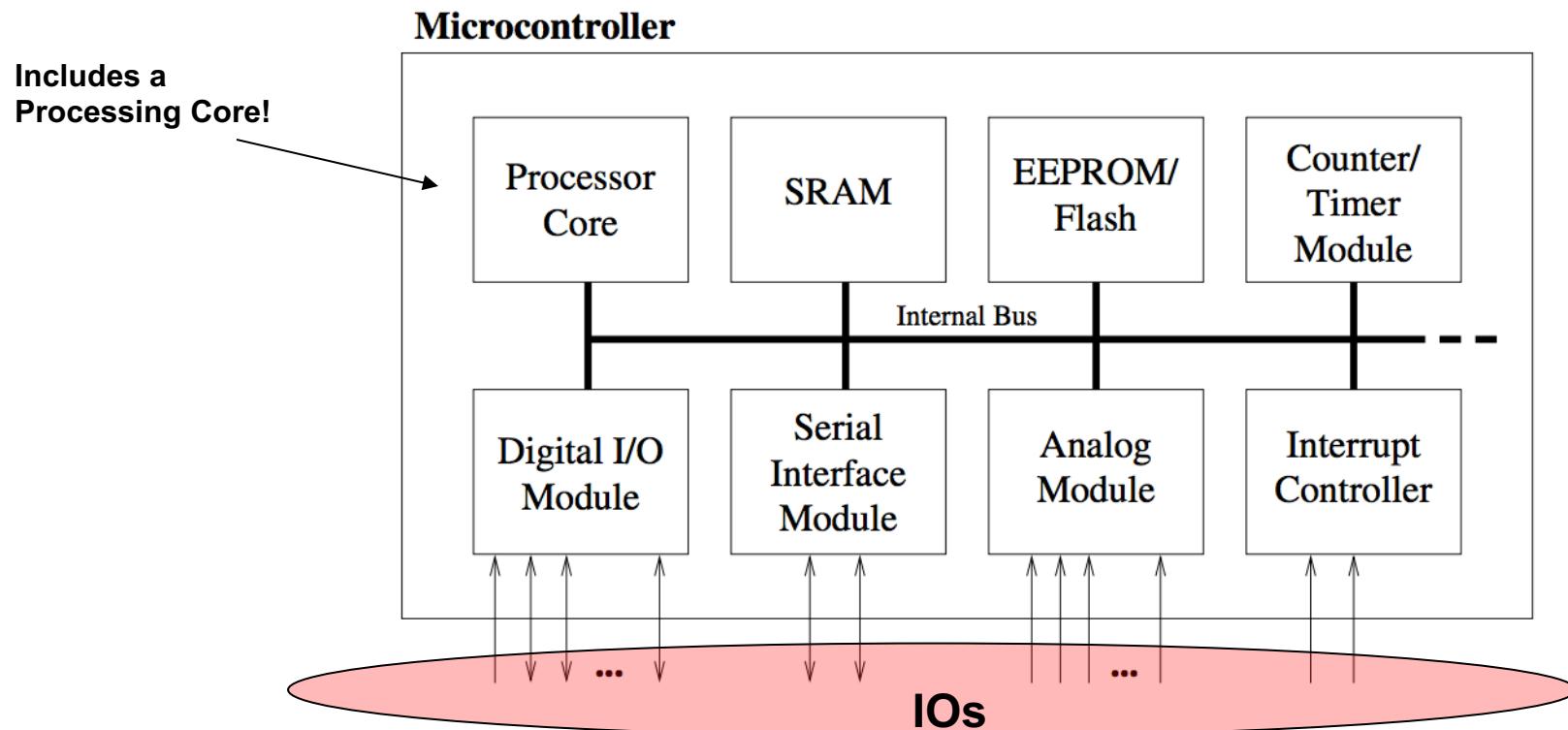


Image source: <http://www.cchc.cl/informacion-a-la-comunidad/industria-de-la-construccion/personaje/>

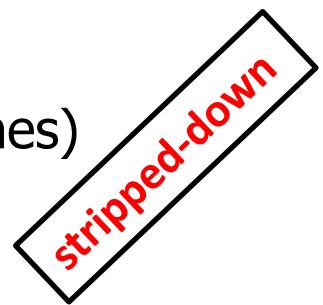
Microcontroller Basic Design (Review)

- All components are connected via an **internal bus**.
- All components are **integrated on one chip**.
- **Communicate** to outside world **via IOs**.



What is not Inside a MCU? (Review)

- No Cache!
- No MMU (maybe you see this in larger microcontrollers)
- No complicated pipeline (single or simple multicycle pipelines)
- No disk
- No FP ALU
-

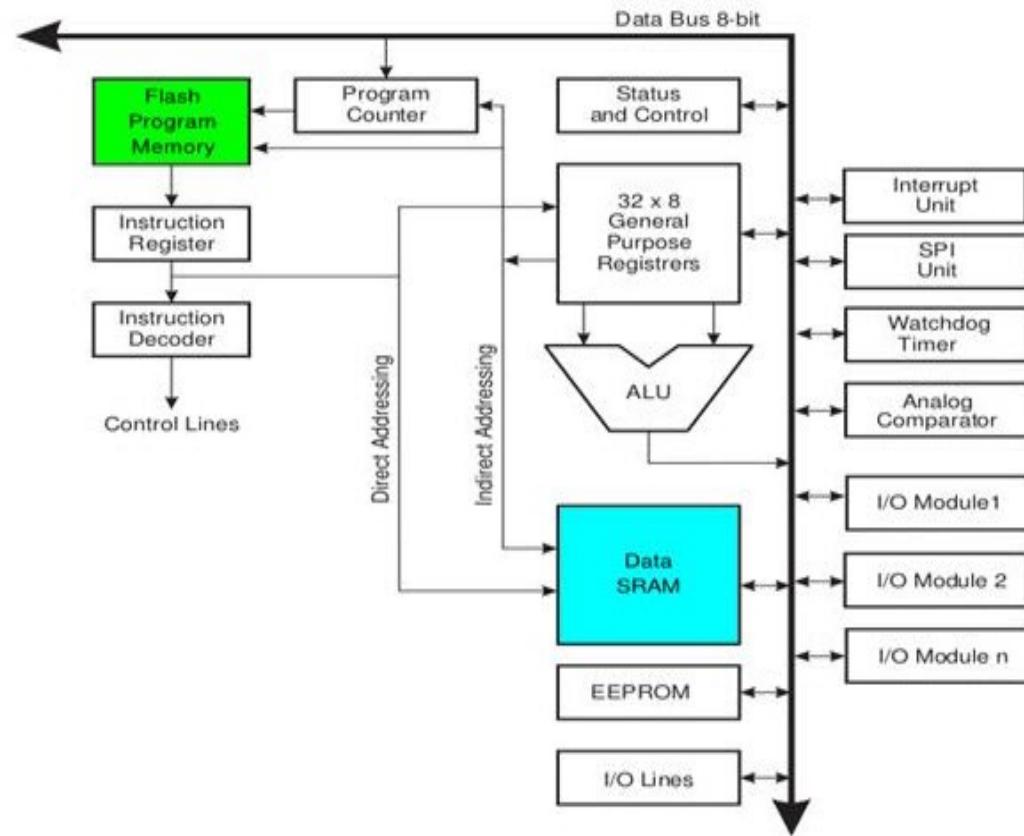


A microcontroller is a (stripped-down) processor which is equipped with memory, timers, (parallel) I/O pins and other on-chip peripherals.

Example: Arduino Processor: (Review)

- Uses the Harvard architecture
 - The program code and program data have separate memories
- Single level pipeline to execute the instructions in order
- 32 x 8 bit general purpose registers
- Single clock cycle access time
- Single cycle ALU operation

Simple



Example: IBM Power5 (Review)

- 2 cores, out-of-order execution
- 100-entry instruction window in each core
- 8-wide instruction fetch, issue, execute
- Large, local+global hybrid branch predictor
- 1.5MB, 8-way L2 cache
- Aggressive stream based prefetching

Complex

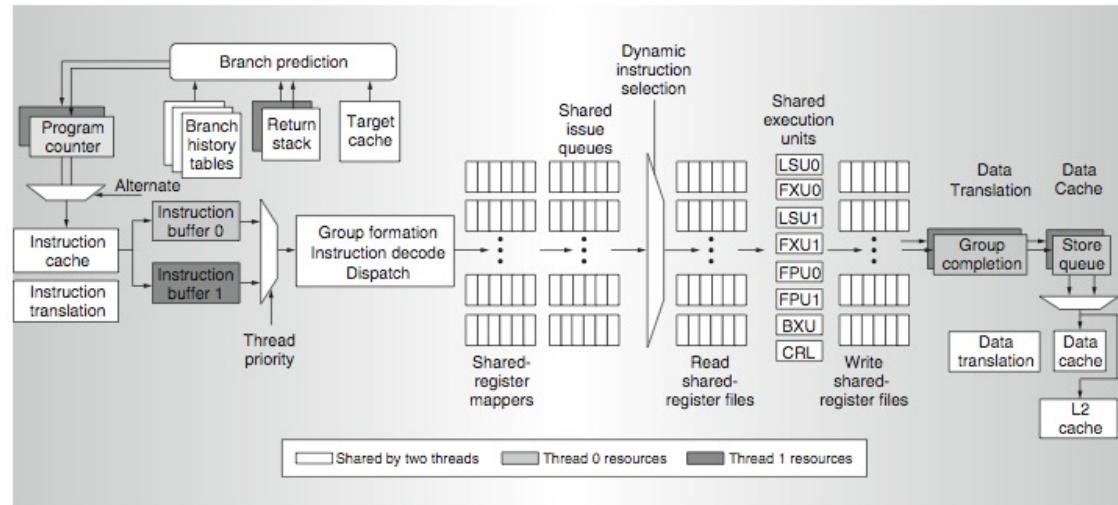
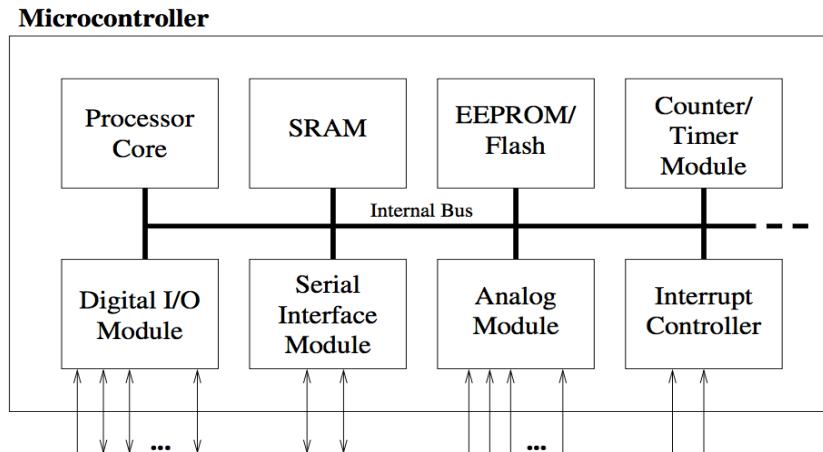
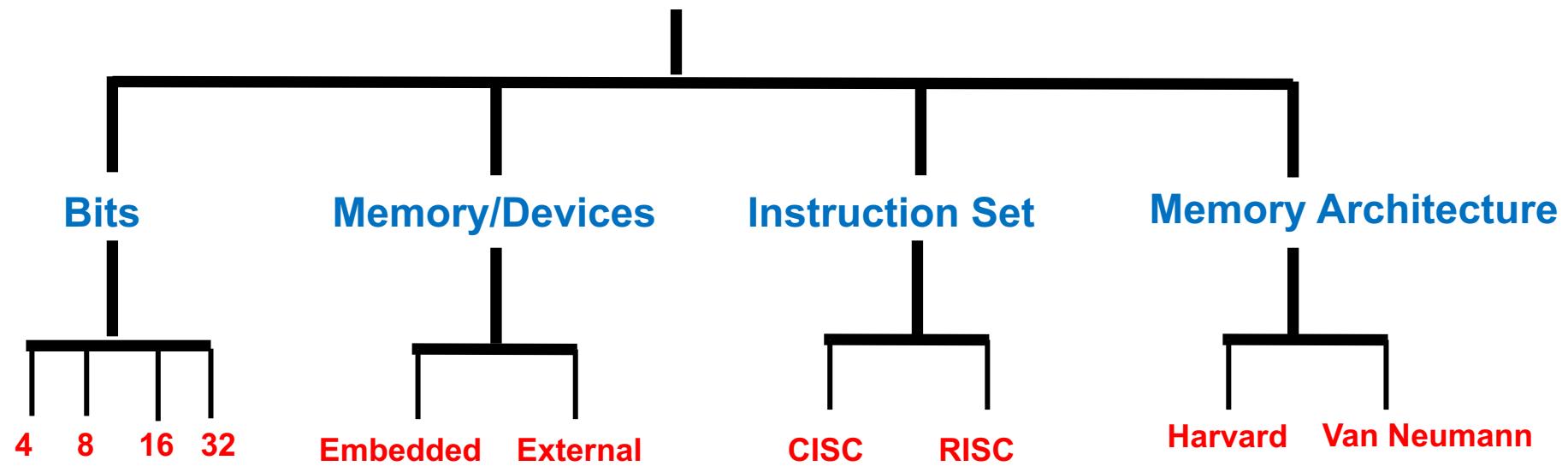


Figure 4. Power5 instruction data flow (BXU = branch execution unit and CRL = condition register logical execution unit).

Microcontroller Classification (Review)



Microcontrollers



Microcontroller vs. Microprocessor (Review)

Microprocessor

- CPU is stand-alone, RAM, ROM, I/O, timer are separate
- designer can decide on the amount of ROM, RAM and I/O ports.
- Expensive
- Versatility
- General-purpose
- High processing power
- High power consumption
- Instruction sets focus on processing-intensive operations
- Typically 32/64 – bit
- Typically deeply pipelined (5-20 stages)

Microcontroller

- CPU, RAM, ROM, I/O and timer are all on a single chip
- fixed amount of on-chip ROM, RAM, I/O ports
- For applications in which cost, power and space are critical
- Single (or limited) purpose (control-oriented)
- Low processing power
- Low power consumption
- Bit-level operations
- Instruction sets focus on control and bit-level operations
- Typically 8-16 bit
- Typically single-cycle/two-stage pipeline

Example:

- A MPU in a GP architecture, running at **600 MHz** has an average **CPI** (number of **Clock** needed **Per Instruction**) **of 1.2** and a average power consumption of **400 mW**. It costs \$100.
- A processor in a MCU running at **12 MHz** with a **two cycle datapath** has a power consumption of **2.4 mW**. It cost \$0.96.
 - What is the associated CPI?
- Calculate their respective MIPS (**M**illions of **I**nstructions processed **P**er **S**econd)
 - **MPU:** $600,000,000 \frac{clk}{s} * \frac{1}{1.2} \frac{Inst}{clk} = 500,000,000 \frac{Inst}{s} = \textcolor{red}{500MIPS}$
 - **MCU:** $12,000,000 \frac{clk}{s} * \frac{1}{2} \frac{Inst}{clk} = 6,000,000 \frac{Inst}{s} = 6MIPS$

Example:

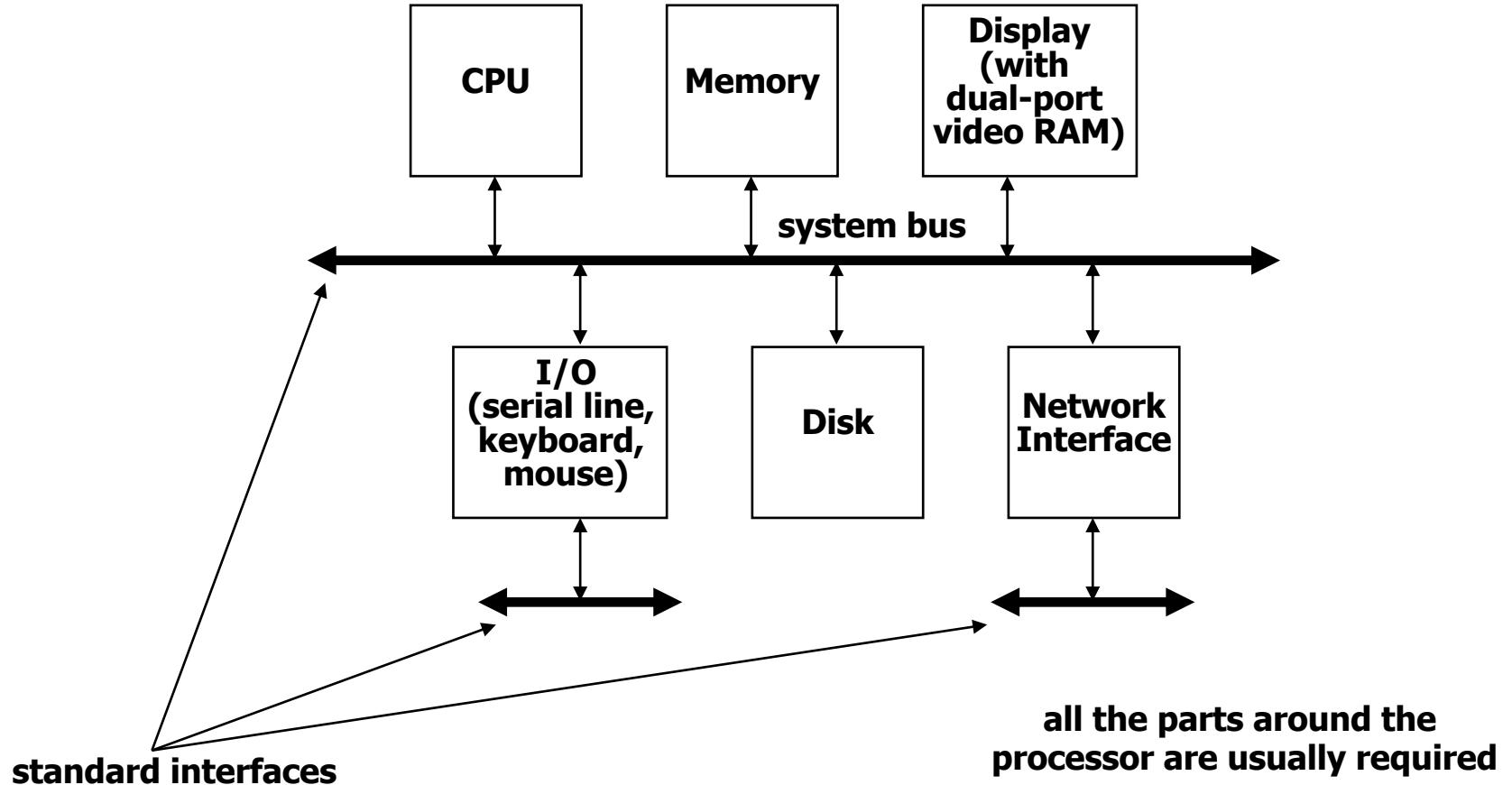
- Which one is more efficient in MIPS/mW?

- **MPU:** $\frac{500 \text{MIPS}}{400 \text{mW}} = 1.25 \text{MIPS/mW} = 1.25 \frac{\text{million instruction}}{\frac{s}{\frac{J}{s}}} = 1.25 \frac{\text{million instruction}}{j}$
- **MCU:** $\frac{6 \text{MIPS}}{2.4 \text{mW}} = 2.5 \text{MIPS/mW} = 2.5 \frac{\text{million instruction}}{\frac{s}{\frac{J}{s}}} = 2.5 \frac{\text{million instruction}}{j}$

- Which is more efficient in MIPS/\$?

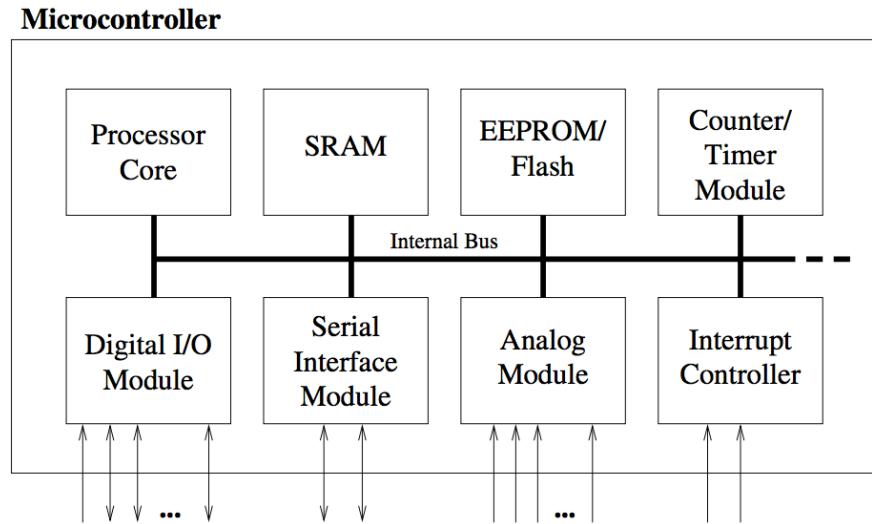
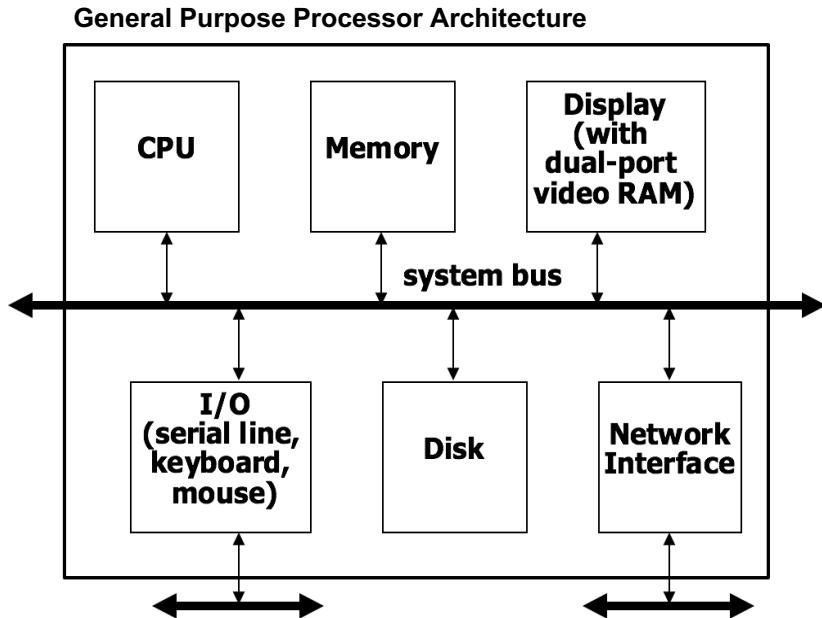
- **MPU:** $\frac{500 \text{MIPS}}{\$100} = 5 \text{MIPS/\$} = 5 \frac{\text{million instruction}}{j.s}$
- **MCU:** $\frac{6 \text{MIPS}}{\$0.48} = 12.5 \text{MIPS/\$} = 12.5 \frac{\text{million instruction}}{j.s}$

Typical General-Purpose Architecture



GPP vs MCU

- Complexity of which processing core (CPU) is higher?
- What is different about the way MCU and GPP communicate to outside?



Which is Used in IoT?

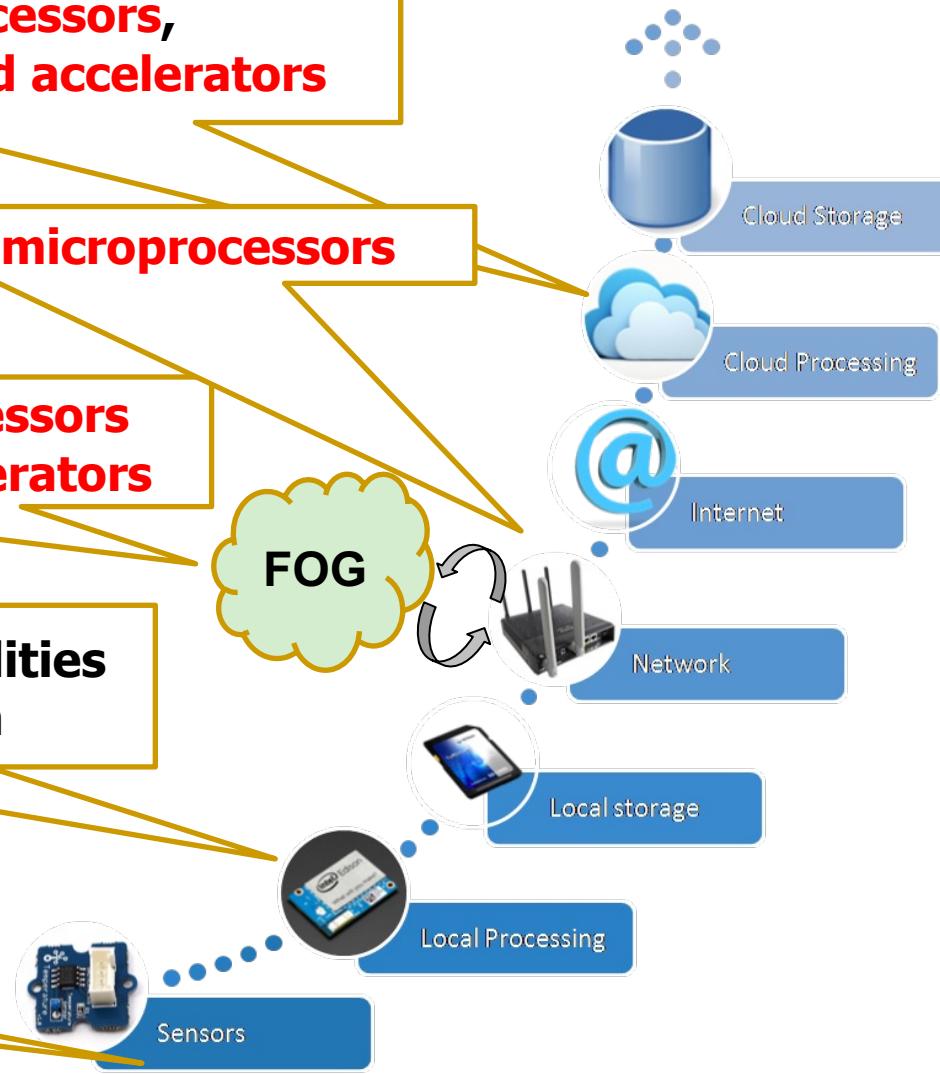
Many **high-end microprocessors**,
assisted by **all kinds of high-end accelerators**

One or few mid-strength microprocessors

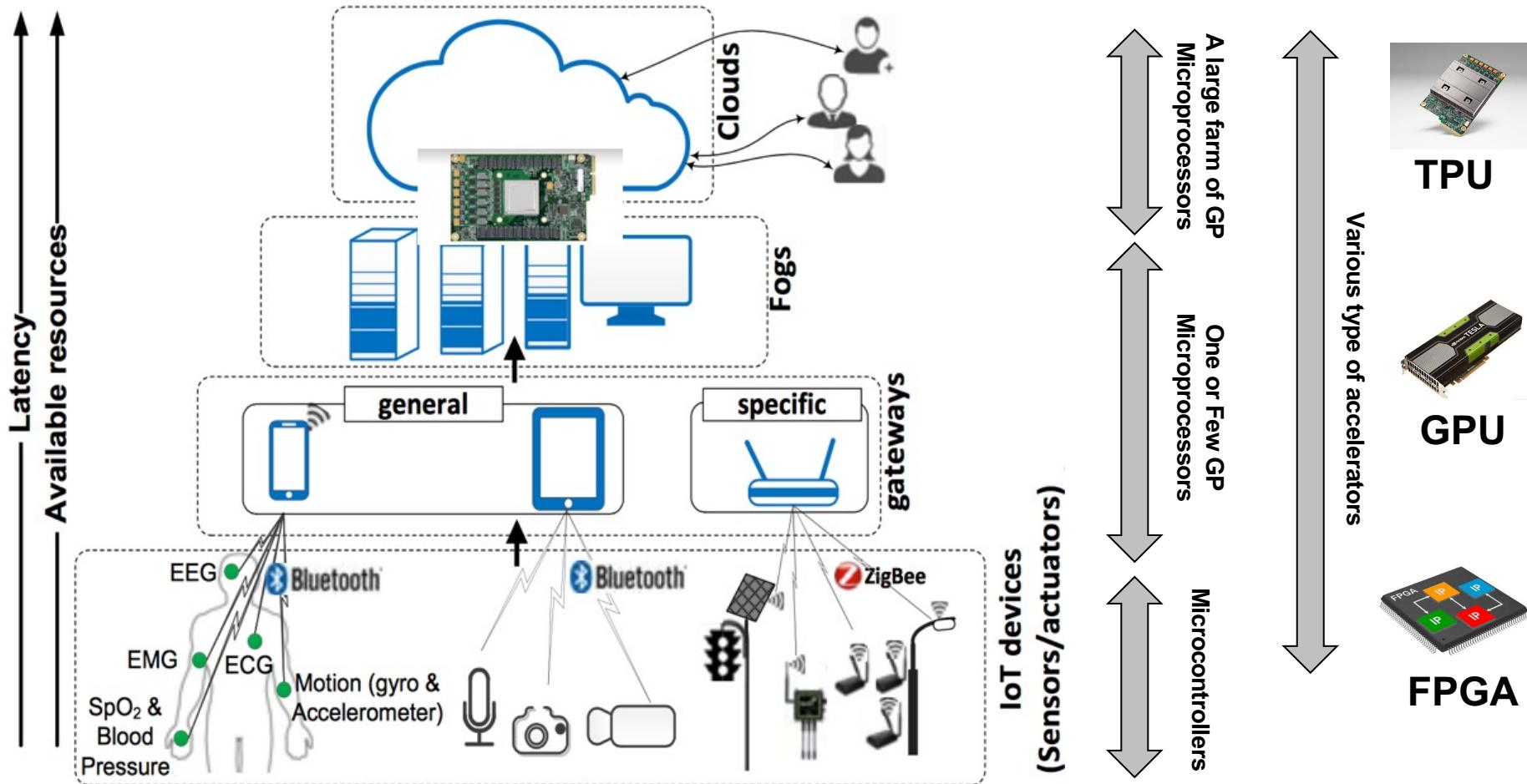
One or few semi-strong microprocessors
maybe assisted by one or few **accelerators**

Microcontrollers with various abilities
depending on the application

Very limited function ICs or
controllers in MEMs

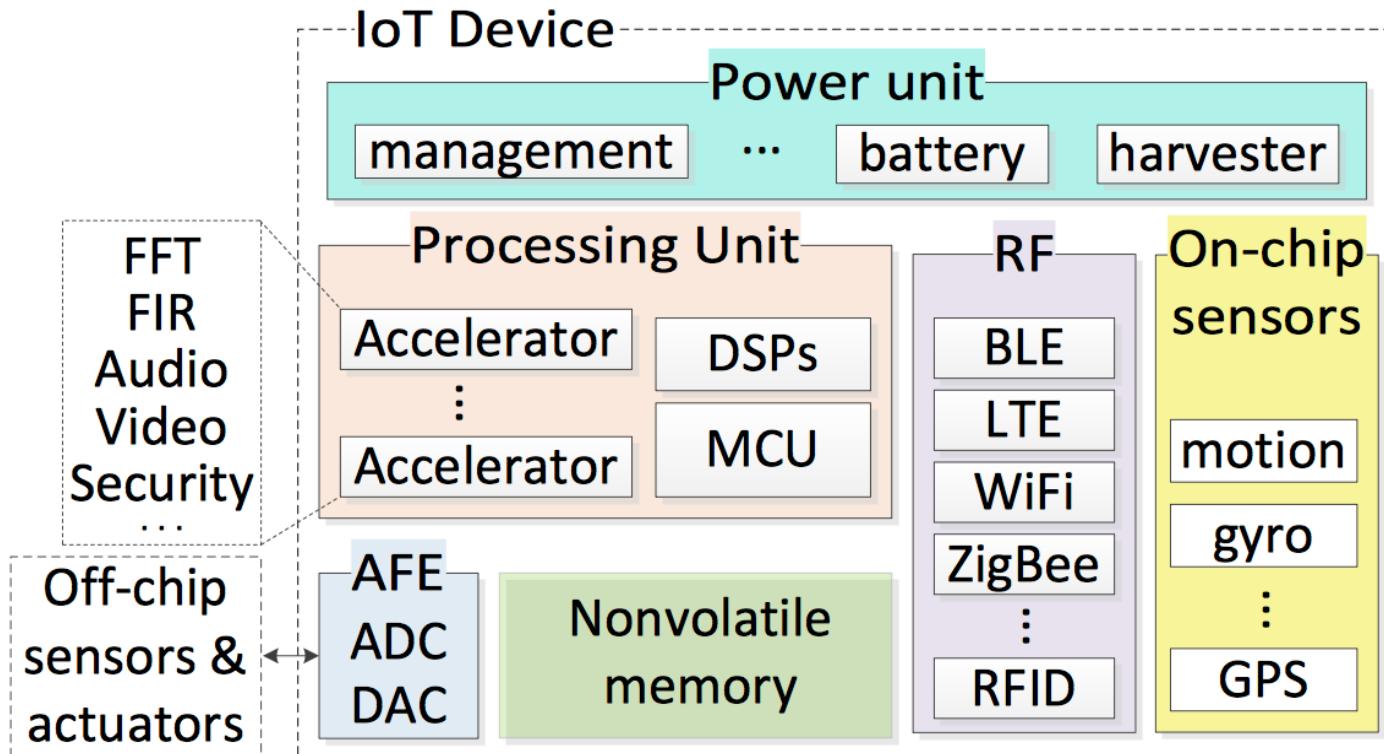


IoT Chain Computation Layers



F. Samie, L. Bauer and J. Henkel, "IoT technologies for embedded computing: A survey," *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Pittsburgh, PA, 2016, pp. 1-10.

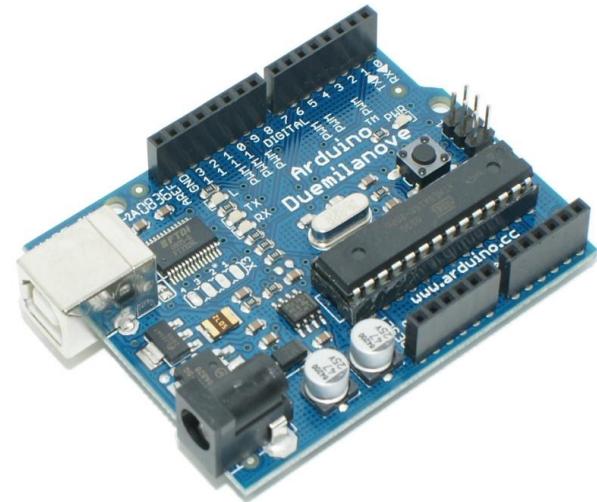
General Arch. of an IoT Device:



Lets look at some examples!

Arduino Microcontroller

- Inexpensive (\$6 - \$50 depending on package!)
- Small size
- Easily Programmable
- Easily connectable
- Open source with big developer community
- Simple to use software
- Easy to augment the functionality
 - Wire directly into the pins on the Arduino board
 - Stack chips called “**shields**” on top of the base unit.
- <https://www.arduino.cc>



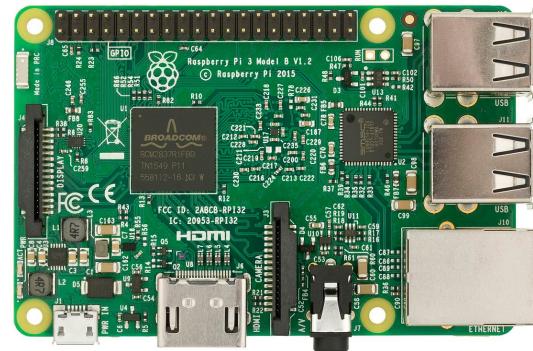
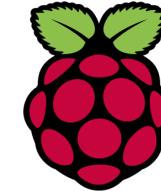
Arduino Ethernet Shield

- Extends the Microcontroller functionality:
 - Connect your Arduino board to the internet.
- Open source
- Simple to use software
- You can **keep stacking** the shields!

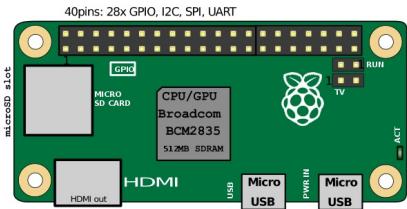


Raspberry Pi

- It is a computer
- It runs Linux
- More software oriented programming
- Embeds a full Networking System
- It is born in the United Kingdom to promote teaching of basic computer science.
- <https://www.raspberrypi.org>

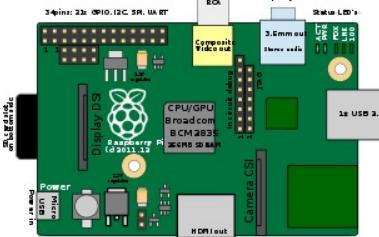


Pi Zero



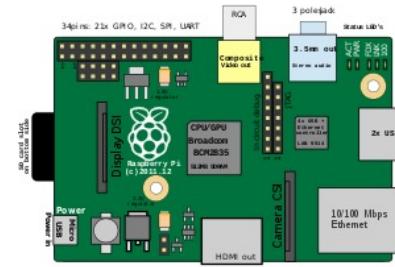
2013

Model A



2015

Model B



Raspberry Pi vs Arduino

Hardware



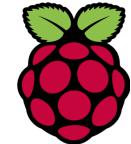
A microcontroller motherboard

run one program at a time,
over and over again

begins executing code when
turned on and stops when you
pull the plug

much easier to connect
analog sensors

Software and Networking system



A general-purpose computer

Can run multiple programs

Need 5V supply to remain on,
and is shut down via a
software process

Built-in Ethernet port

requires software to effectively
interface with other devices

Good for Sensors

<https://www.arduino.cc>



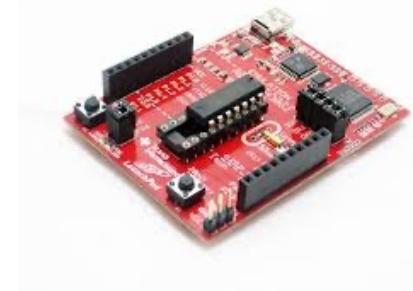
Arduino
\$25
ATmega328

<http://chipkit.net>



ChipKIT
\$30
PIC

<http://www.ti.com/lscds/ti/tools-software/launchpads/launchpads.page>

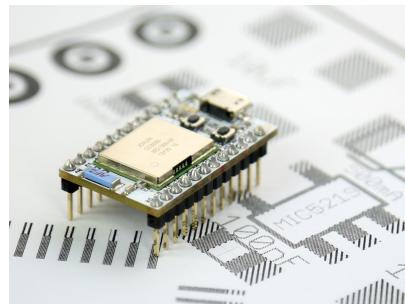


LaunchPad
\$4
MSP430

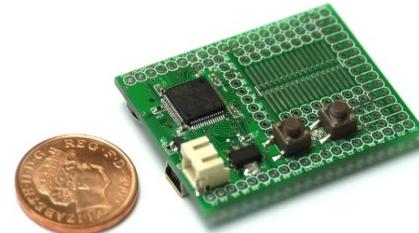
Good for Sensing & Processing



STM32
\$30
ARM Cortex M0,
M3, M4



Particle
\$35
ARM
WiFi Internet



Espruino
\$30
ARM
Javascript

Good for Processing & Networking



Raspberry Pi

\$35

900 MHz ARM CPU

250 MHz GPU

1 GB RAM

Compute Module



Intel® Galileo

\$50

400 MHz Quark x86

256 MB RAM



Intel® Edison

\$70

1 GHz Dual Core Atom x86

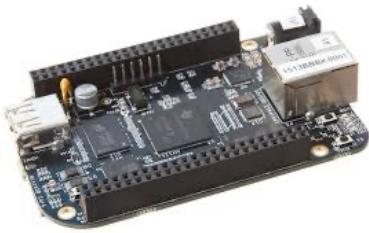
1 GB RAM

WiFi

BLE

4 GB Flash

Good for Processing and Network



Beaglebone Black
\$45
1 GHz ARM, GPU
512 MB RAM
4 GB Flash



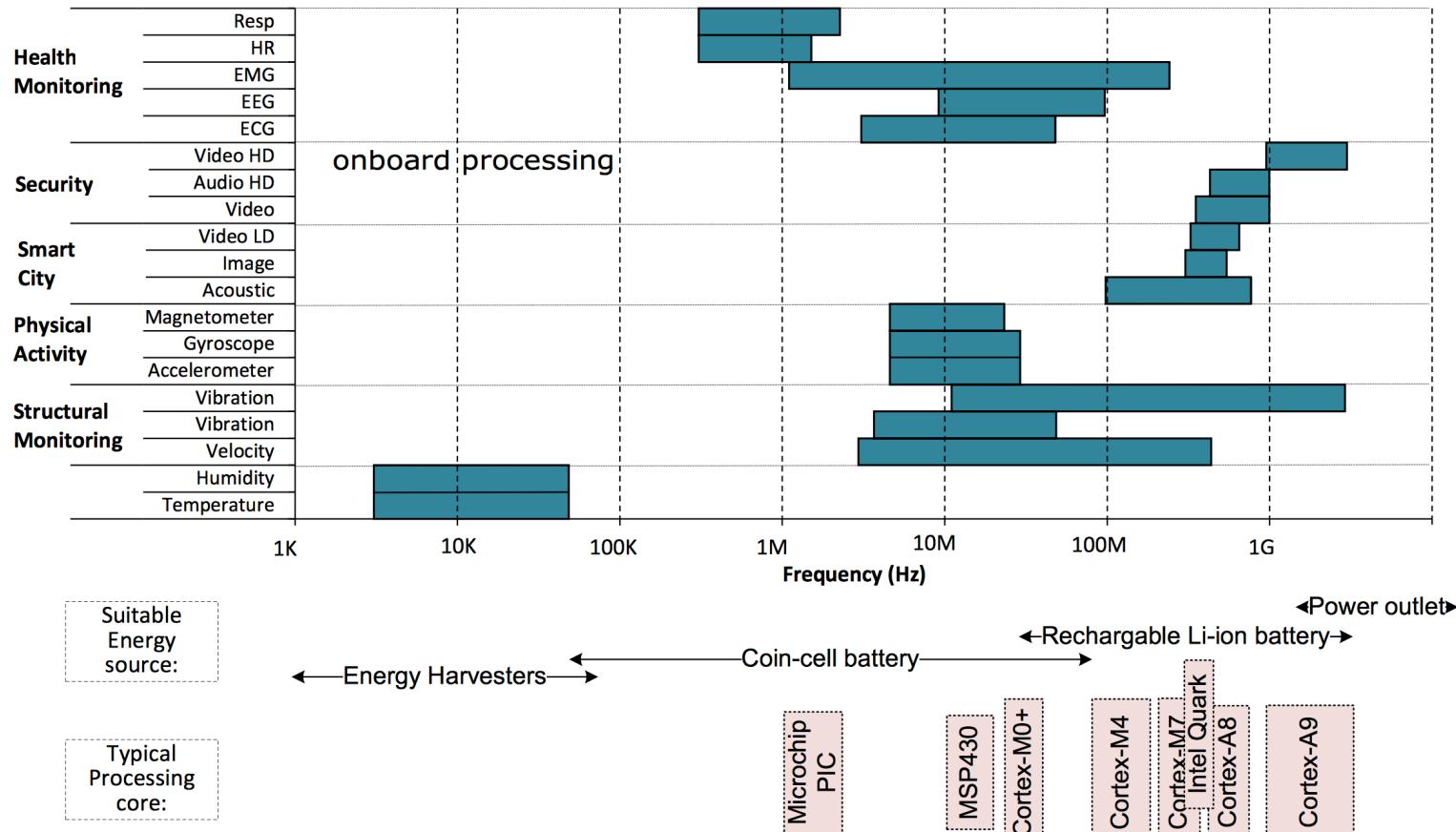
UDOO Neo
\$50
i.MX 6 Solo ARM, GPU
ARM M4
512 MB or 1 GB RAM



Parallella
\$99
1 GHz Dual Core Zynq ARM
16 or 64 Epiphany CPUs

Processing Sensor Data

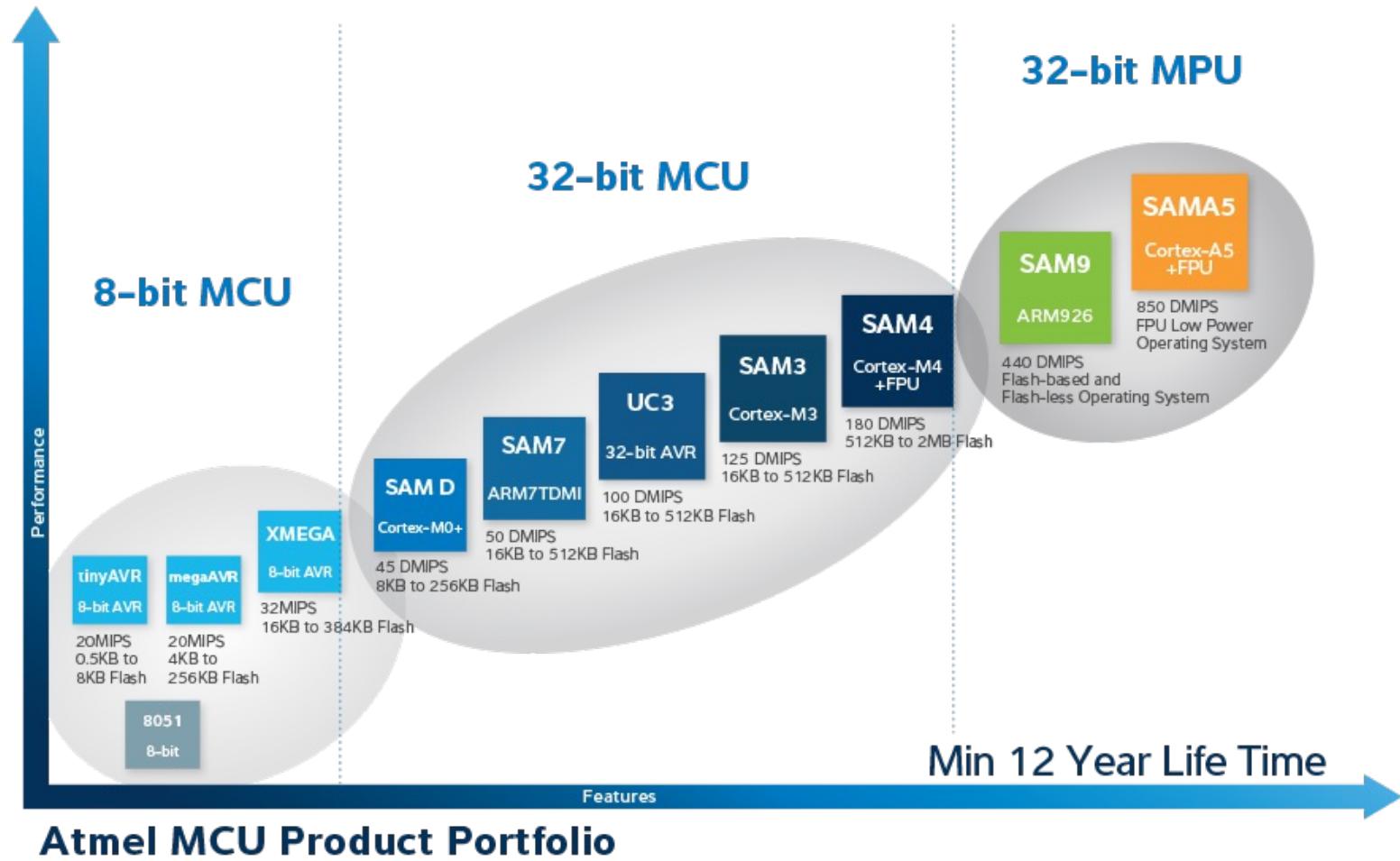
- The number of cycles (i.e. required frequency) to fully process the IoT sensors



F. Samie, L. Bauer and J. Henkel, "IoT technologies for embedded computing: A survey," *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Pittsburgh, PA, 2016, pp. 1-10.

Wide Range of MCU Choices

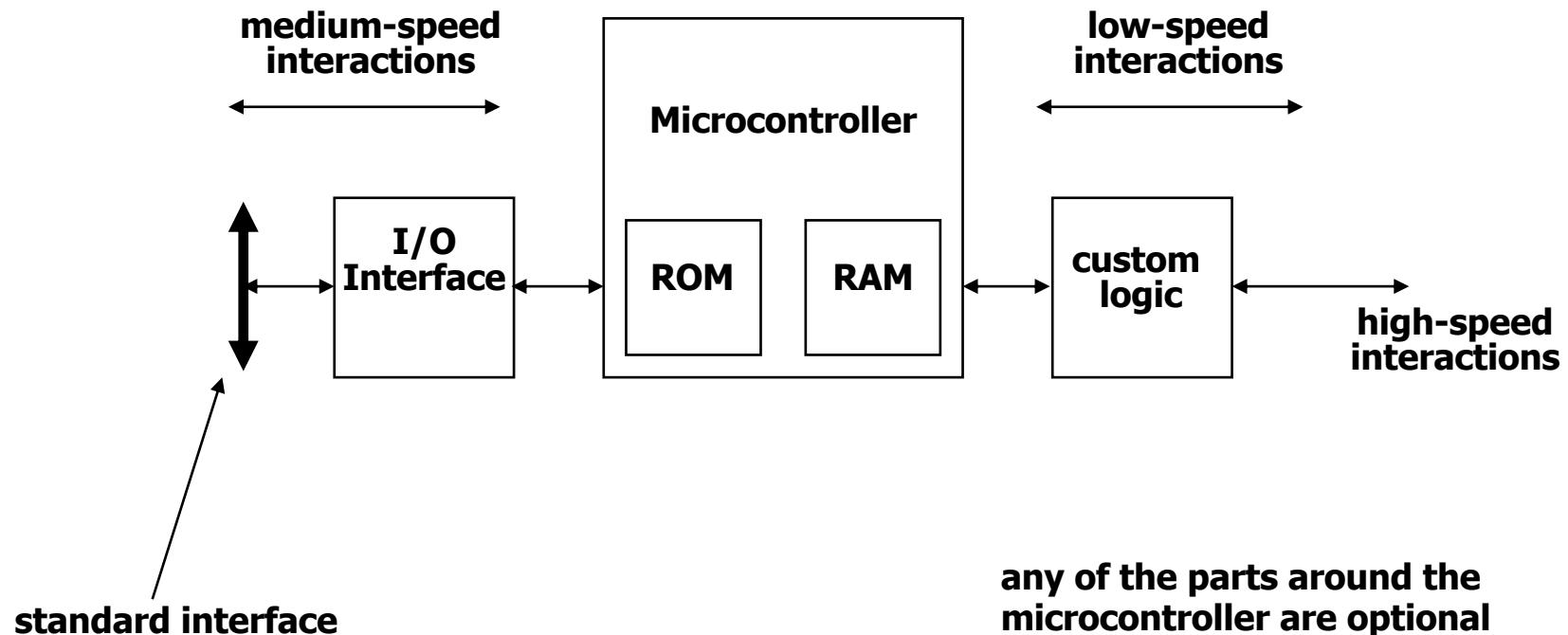
- As an example, see how many MCUs are offered by Atmel



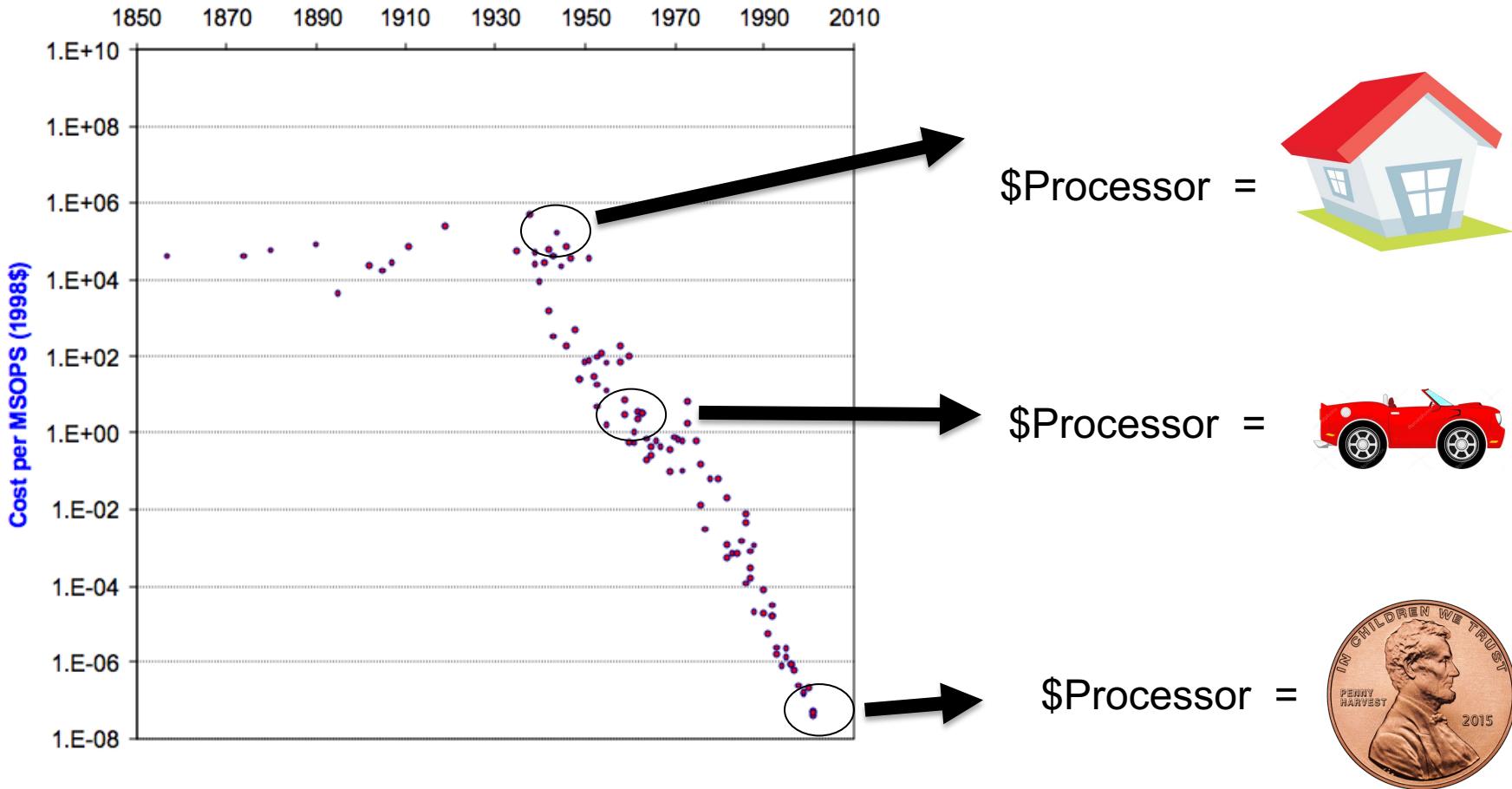
The End!



Typical task-specific architecture



Cost of Processing Drops Quickly!



Similar trend happened to sensors!

This allows us to put few sensors and a processor in any and every object!



Internet of Things

Senior Design Project Course

Processing - Part 1

Lecturer: Avesta Sasan

University of California Davis

Focus of Today's Lecture: MCU

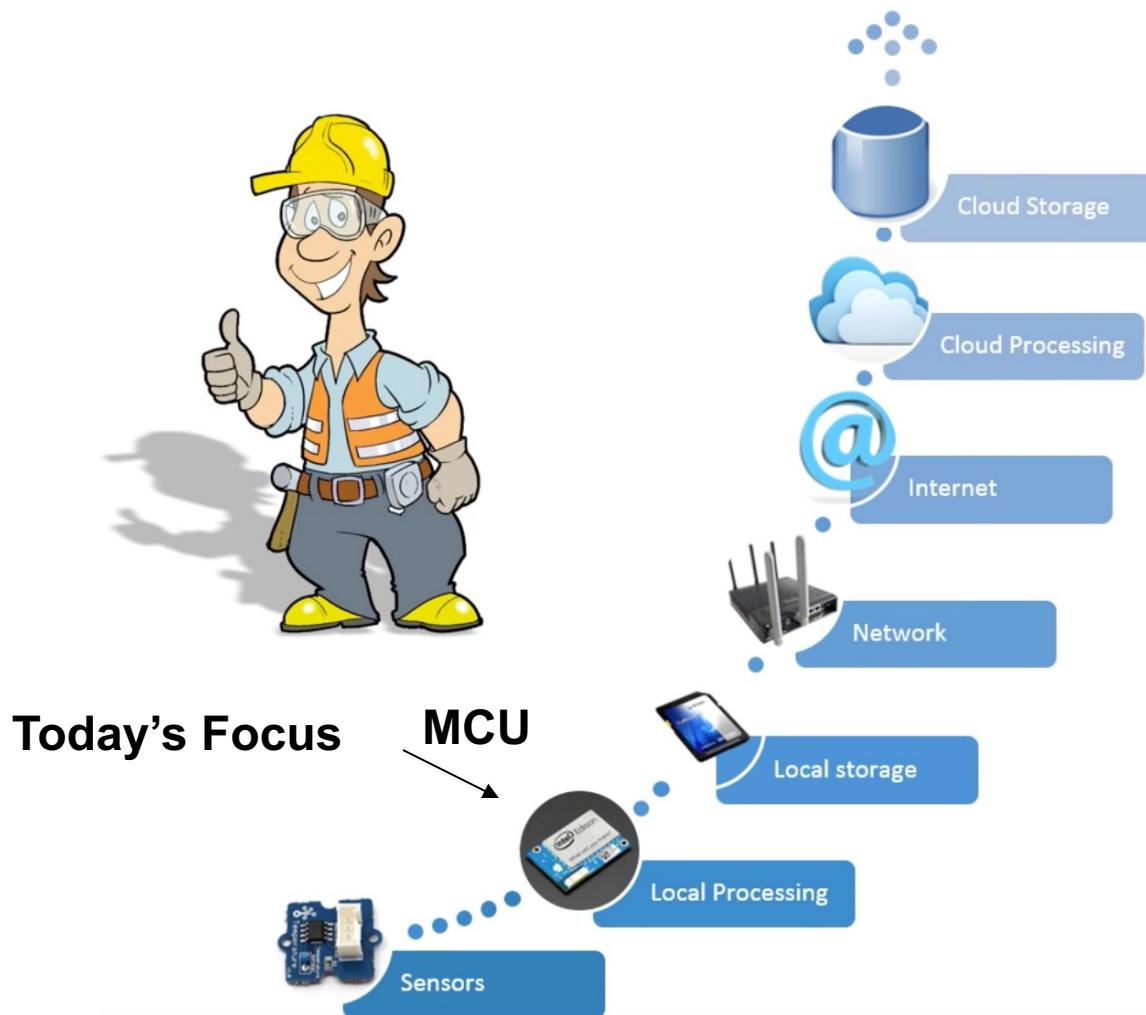
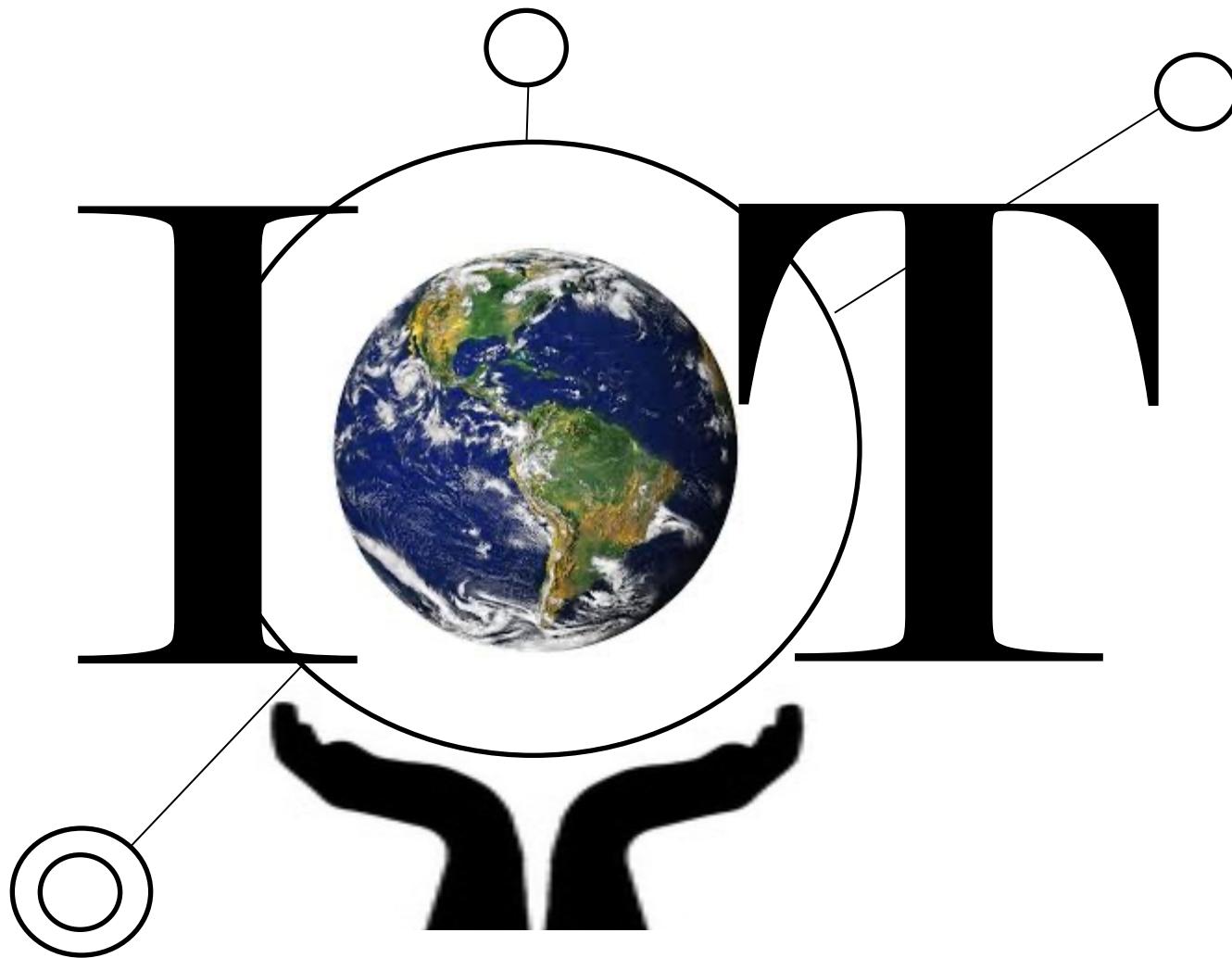


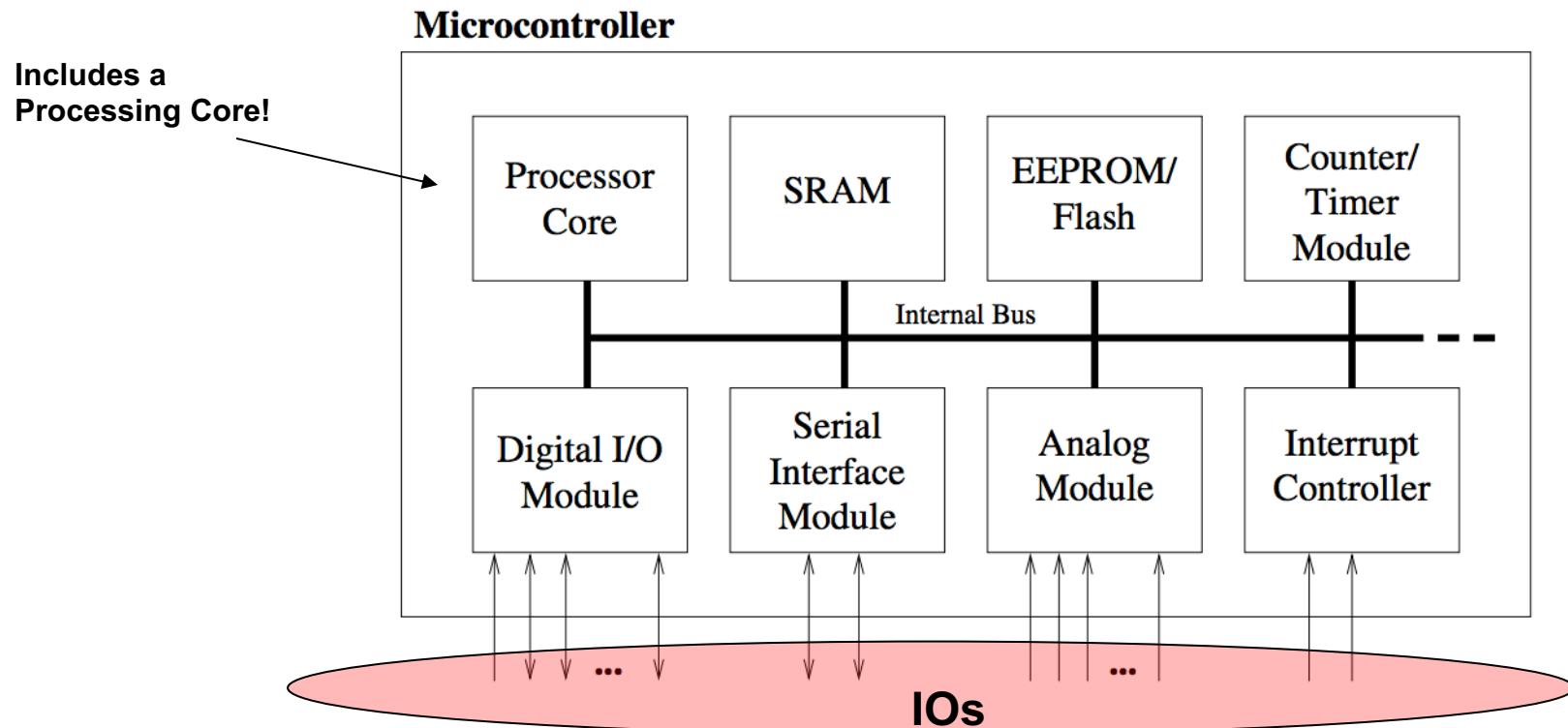
Image source: <http://www.cchc.cl/informacion-a-la-comunidad/industria-de-la-construccion/personaje/>

Lets Get Started:



Microcontroller Basic Design

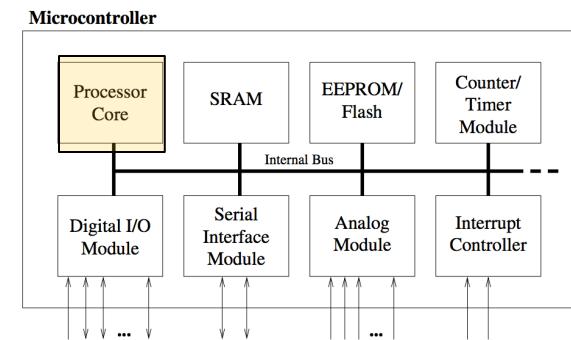
- All components are connected via an **internal bus**.
- All components are **integrated on one chip**.
- **Communicate** to outside world **via IOs**.



Inside a Microcontroller!

■ Processor Core:

- The CPU of the controller.
- Contains the arithmetic logic unit (**ALU**), the **control unit**, and the **registers** (stack pointer, program counter, accumulator register, register file, . . .).



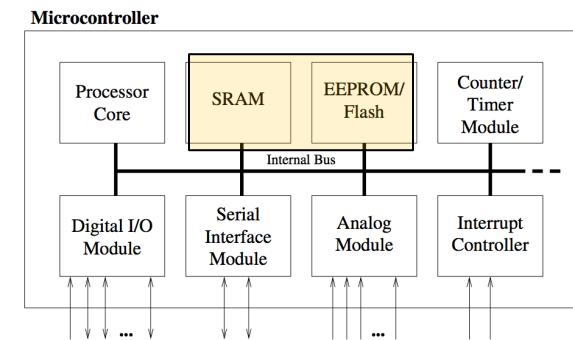
Inside a Microcontroller!

■ Processor Core:

- The CPU of the controller.
- Contains the arithmetic logic unit (**ALU**), the **control unit**, and the **registers** (stack pointer, program counter, accumulator register, register file, . . .).

■ Memory:

- May be split into program memory and data memory.
- RAM, ROM, SRAM, FLASH/EEPROM



Inside a Microcontroller!

■ Processor Core:

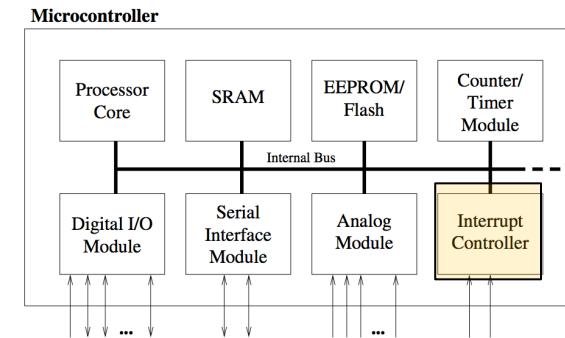
- The CPU of the controller.
- Contains the arithmetic logic unit (**ALU**), the **control unit**, and the **registers** (stack pointer, program counter, accumulator register, register file, . . .).

■ Memory:

- May be split into program memory and data memory.
- RAM, ROM, SRAM, FLASH/EEPROM

■ Interrupt Controller:

- Interrupting the normal program flow in case of external or internal events.
- When combined with sleep modes, they help to conserve power.



Inside a Microcontroller!

■ Processor Core:

- The CPU of the controller.
- Contains the arithmetic logic unit (**ALU**), the **control unit**, and the **registers** (stack pointer, program counter, accumulator register, register file, . . .).

■ Memory:

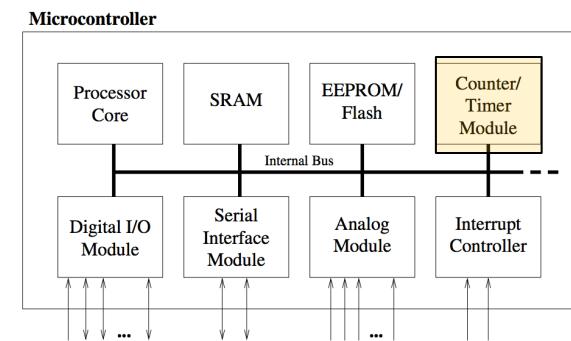
- May be split into program memory and data memory.
- RAM, ROM, SRAM, FLASH/EEPROM

■ Interrupt Controller:

- Interrupting the normal program flow in case of external or internal events.
- When combined with sleep modes, they help to conserve power.

■ Timer/Counter:

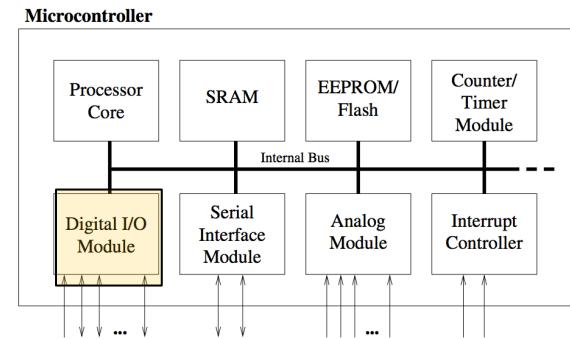
- Used to timestamp events, measure intervals, or count events
- 1-3 Timer/Counters per MCU
- Some also contain Pulse Width Modulation (PWM).



Inside a Microcontroller!

■ Digital I/O:

- Parallel digital I/O ports
 - from 3-4 to over 90



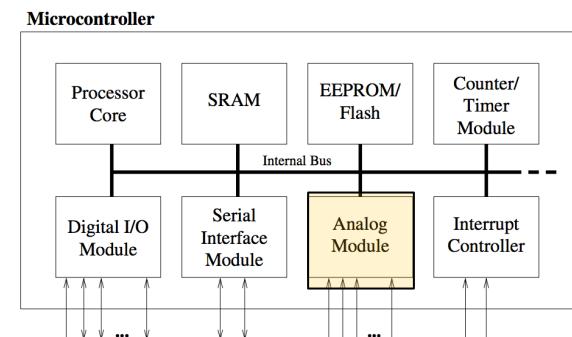
Inside a Microcontroller!

■ Digital I/O:

- Parallel digital I/O ports
 - from 3-4 to over 90

■ Analog I/O:

- 2-16 ports
- Digital/Analog converters. (DAC, ADC)



Inside a Microcontroller!

■ Digital I/O:

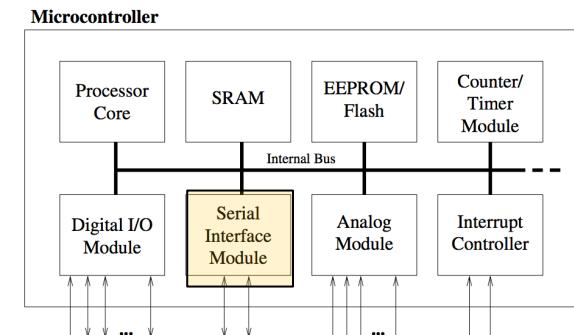
- Parallel digital I/O ports
 - from 3-4 to over 90

■ Analog I/O:

- 2-16 ports
- Digital/Analog converters. (DAC, ADC)

■ Interfaces:

- For communication with the development PC in general
- Most controllers offer several and varied interfaces like SPI and I2C.
- Larger microcontrollers may also contain PCI, USB, or Ethernet interfaces.



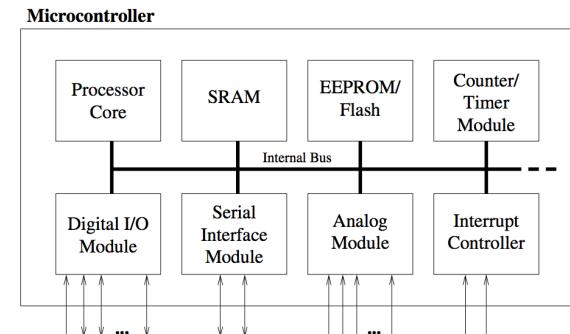
Inside a Microcontroller!

■ Watchdog Timer:

- Used to reset the controller in case of software “crashes”.
- Timer set to count down, if reaches 0, restart the microcontroller
- To prevent restart, the software has to reset the watchdog.

■ Debugging Unit:

- Some include additional hardware to allow remote debugging so there is no need to download special debugging software.



What is not Inside a MCU?

- No Cache!
- No MMU (maybe you see this in larger microcontrollers)
- No complicated pipeline (single or simple multicycle pipelines)
- No disk
- No FP ALU
-



stripped-down



A microcontroller is a (stripped-down) processor which is equipped with memory, timers, (parallel) I/O pins and other on-chip peripherals.

Lets Review Related Terms:

■ **Microprocessor:**

- A normal CPU (Central Processing Unit).
- Communicate with external devices by data bus
 - Peripheral devices (memory, floppy controller, USB controller, timer, . . .) are connected to the bus.
- Only contains data, address pins and a couple of control pins.
- Can not operate stand-alone
 - At the very least it requires some memory and an output device



■ **Mixed-Signal Controller:**

- This is a microcontroller which can process both digital and analog signals.

■ **Real-Time System:**

- Reaction to an event has to occur within a specified time.
- MCUs are very often used in Real Time Systems.

Lets Review Related Terms:

■ **Embedded System:**

- A microcontroller within a larger mechanical or electrical system
- A complete device often including hardware and mechanical parts.

■ **Embedded Processor vs Embedded Controller**

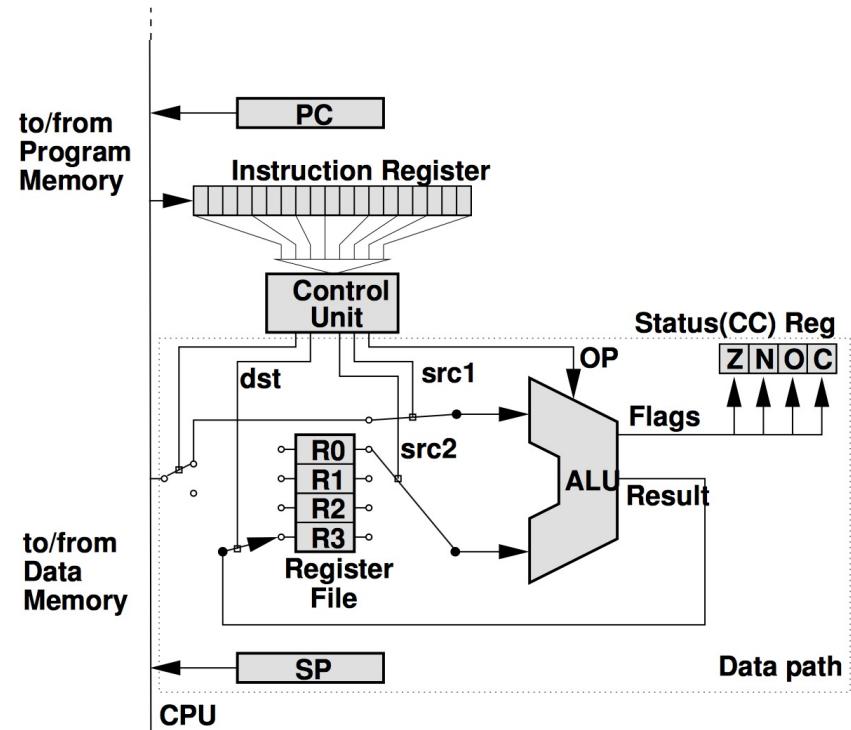
- This term often occurs in association with embedded systems!
- “embedded processor” is used for high-end devices (32 bits).
- “embedded controller” is traditionally used for low-end devices (4, 8, 16 bits).

■ **Digital Signal Processor (DSP):**

- To process signals.
- An important area of use is telecommunications.
- Designed for fast addition and multiplication
- Many vendors combine a controller with a DSP on one chip
 - e.g. Motorola’s DSP56800.

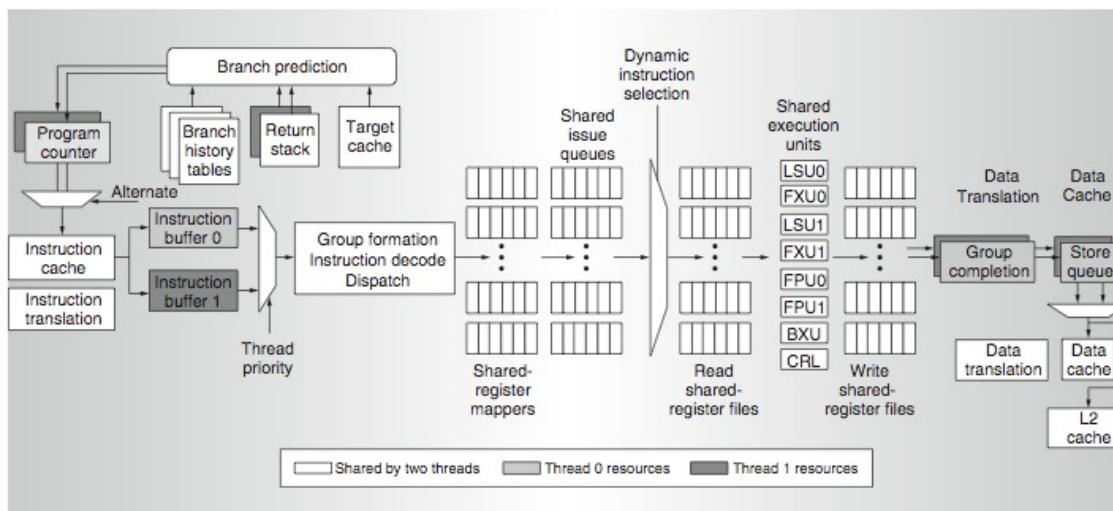
Basic CPU architecture

- A very basic CPU architecture:
- **PC:** Program Counter: keep track of executed program.
- **Instruction Register:** keeps the incoming instruction for execution.
- **SP:** Stack Pointer
- **Control Unit:** decode the instruction and generate the control signals.
- **ALU:** Arithmetic Logic Unit: execute arithmetic operation on data
- **RF:** Register File: hold the data input to or output from ALU.
- **Status Reg:** Keep the status of the current instruction executed by ALU
 - **Z:** Zero
 - **N:** Negative
 - **O:** Overflow
 - **C:** Carry

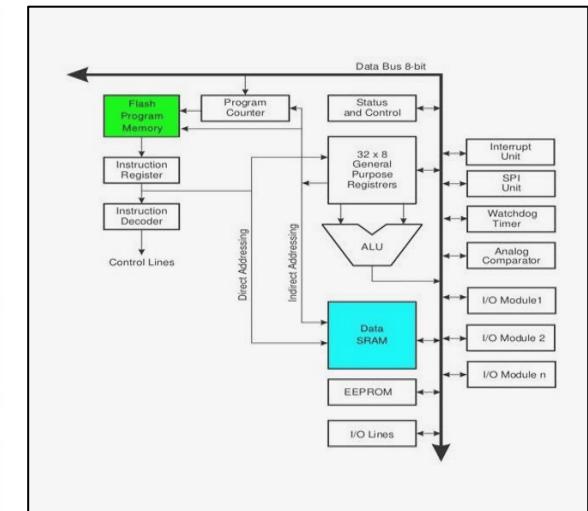


Processor Complexity Varies Widely!

- Processor in MCU and in General Purpose Computers are very different in terms of capabilities.
 - MCU processor → Simple → light workload → low power
 - GP processor → Complex → Can handle anything → high power



IBM Power5 Microprocessor

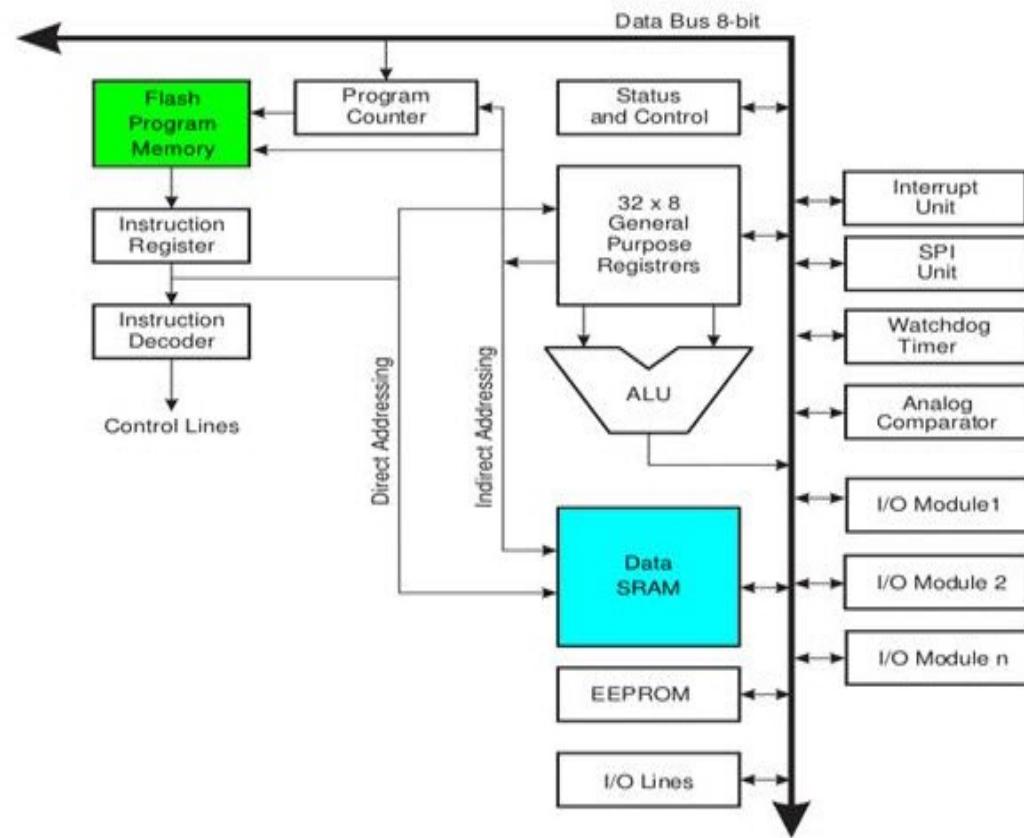


Arduino Microprocessor

Example: Arduino Processor:

- Uses the Harvard architecture
 - The program code and program data have separate memories
- Single level pipeline to execute the instructions in order
- 32 x 8 bit general purpose registers
- Single clock cycle access time
- Single cycle ALU operation

Simple



Example: IBM Power5

- 2 cores, out-of-order execution
- 100-entry instruction window in each core
- 8-wide instruction fetch, issue, execute
- Large, local+global hybrid branch predictor
- 1.5MB, 8-way L2 cache
- Aggressive stream based prefetching

Complex

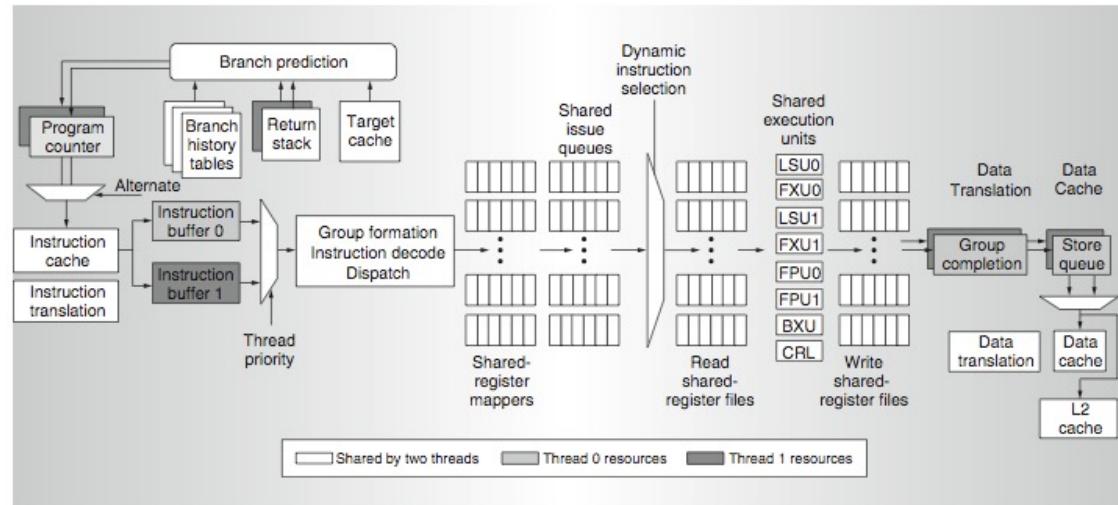
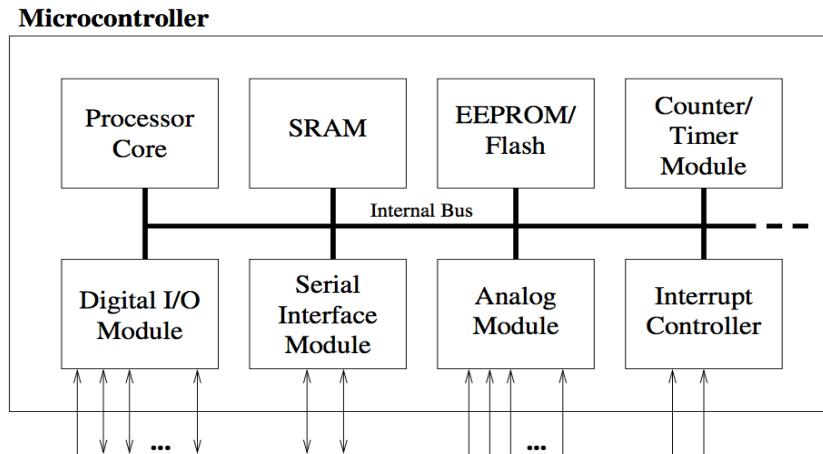
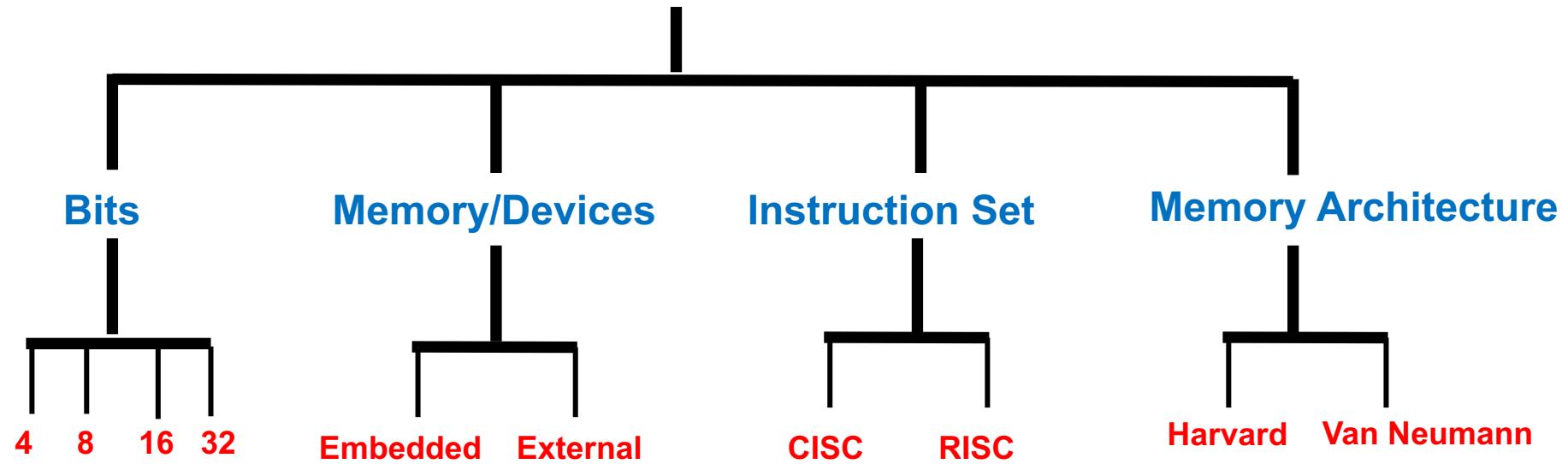


Figure 4. Power5 instruction data flow (BXU = branch execution unit and CRL = condition register logical execution unit).

Microcontroller Classification



Microcontrollers



Microcontroller Data Width

- Data width is the size of internal bus and size of ALU inputs.
- The larger the data width
 - Greater the precision
 - Higher the performance
 - Higher the complexity
- Note that workload should match processor complexity.
 - Using over or under capable MCU result in larger energy consumption for execution of task.
- Examples of MCU with different data widths:
 - **8 bit microprocessors:** Intel 8031/8051, PIC1x and Motorola MC68HC11 families
 - **16 bit microprocessors:** extended 8051XA, PIC2x, Intel 8096 and Motorola MC68HC12 families
 - **32 bit microprocessors:** Intel/Atmel 251 family, PIC3x

Processor Architecture (RISC vs CISC)

■ **RISC: Reduced Instruction Set Architecture**

- Has simple, hard-wired instructions
- Instructions take one or a few clock cycles to execute.
- Few instructions
- Few addressing modes.
- The instruction set is rather simple.
- Execution of instructions is very fast, but instructions are simple

■ **CISC: Complex Instruction Set Architecture**

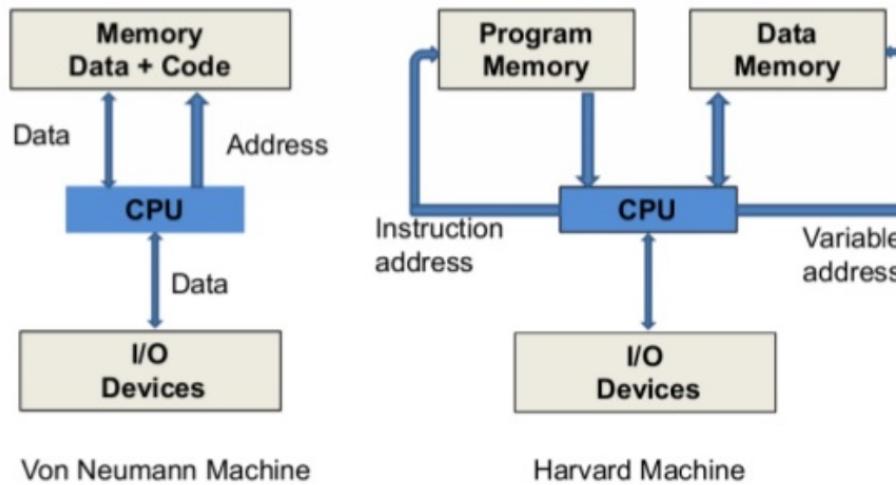
- Has complex micro-coded instructions
- Instructions can take many clock cycles to execute.
- Powerful instructions and addressing modes.
- In comparison to RISC, CISC takes longer to execute its instructions, but the instruction set is more powerful.

Von Neumann vs Harvard Architecture

- If memory and data are in the same memory or not?

- **Von Neumann Architecture:**

- program and data are stored together and are accessed through the same bus.
 - program and data accesses may conflict

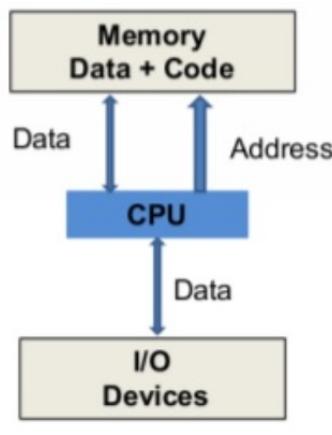


- **Harvard Architecture:**

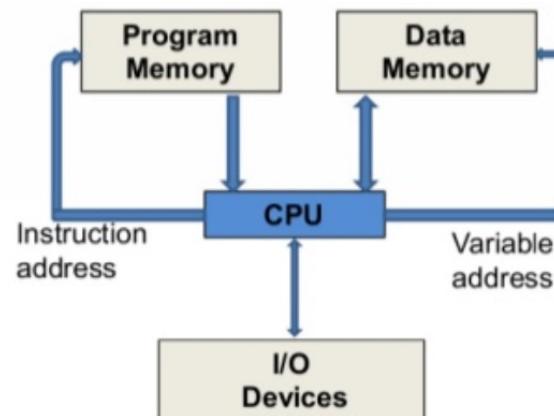
- program and data are in separate memories which are accessed via separate buses.
 - code accesses do not conflict with data accesses
 - improves system performance.

Tradeoff!

- **Harvard Architecture:** requires more hardware,
 - two busses and either two memory chips or a dual-ported memory (a memory chip which allows two independent accesses at the same time).
- **Von Neumann Architecture:**
 - conflict in simultaneous need to access instruction and data causes *bottleneck, leading to unwelcome delays.*
 - *A.K.A Princeton Architecture*



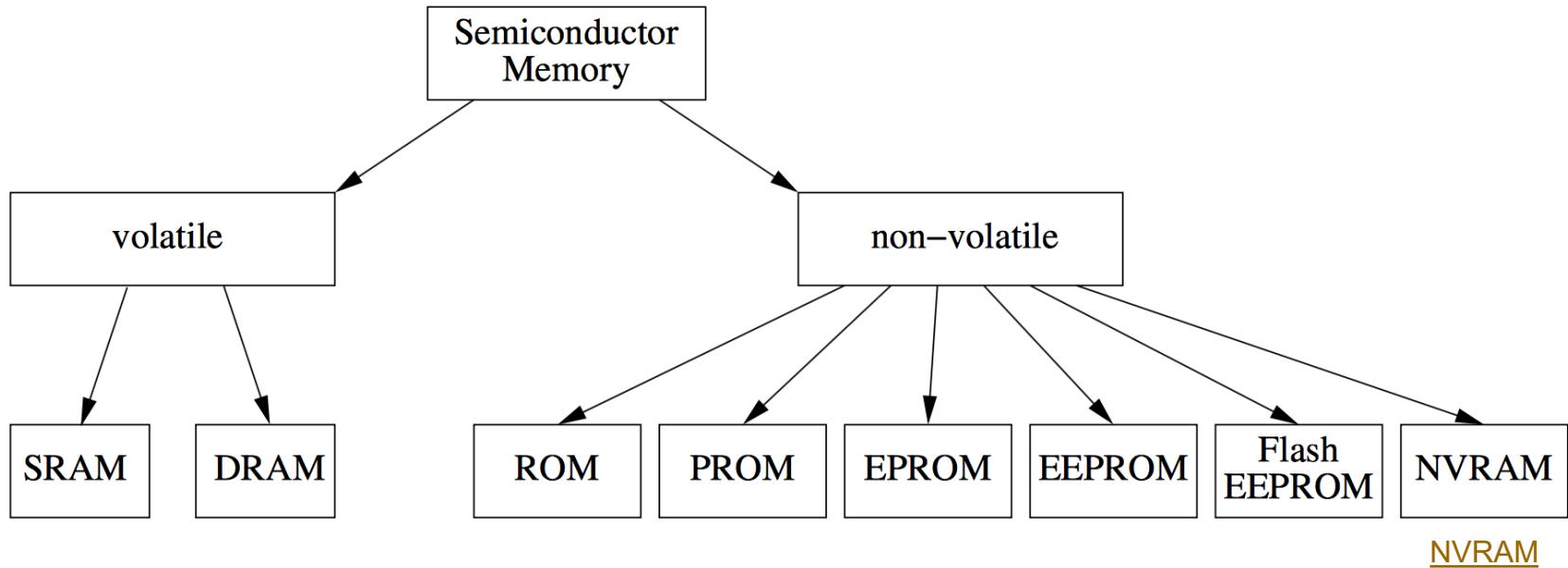
Von Neumann Machine



Harvard Machine

Memory

- Semiconductor memories could be categorized into:
 - **Volatile:** loses value if supply is disconnected
 - **Nonvolatile:** keeps the value if supply is disconnected



Microcontroller vs. Microprocessor

Microprocessor

- CPU is stand-alone, RAM, ROM, I/O, timer are separate
- designer can decide on the amount of ROM, RAM and I/O ports.
- Expensive
- Versatility
- General-purpose
- High processing power
- High power consumption
- Instruction sets focus on processing-intensive operations
- Typically 32/64 – bit
- Typically deeply pipelined (5-20 stages)

Microcontroller

- CPU, RAM, ROM, I/O and timer are all on a single chip
- fixed amount of on-chip ROM, RAM, I/O ports
- For applications in which cost, power and space are critical
- Single (or limited) purpose (control-oriented)
- Low processing power
- Low power consumption
- Bit-level operations
- Instruction sets focus on control and bit-level operations
- Typically 8-16 bit
- Typically single-cycle/two-stage pipeline



Internet of Things

Senior Design Project Course

Signals and Systems

Lecturer: Avesta Sasan

University of California Davis

Signals And Systems



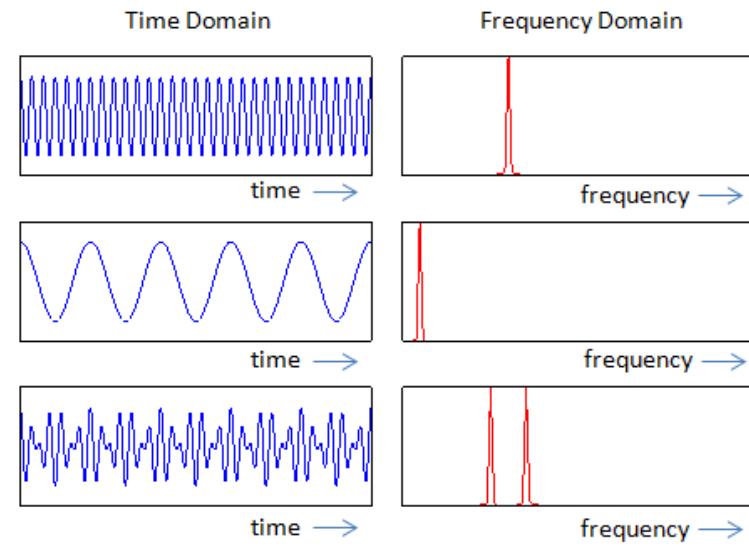
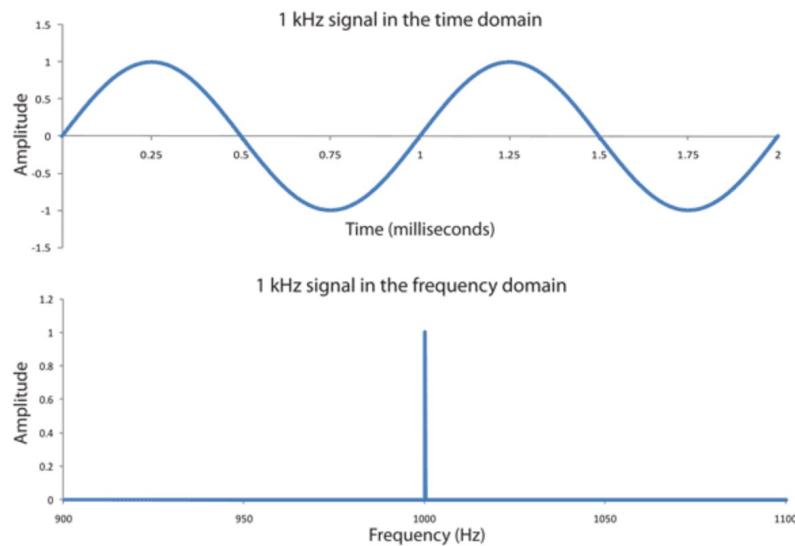
Signals and Systems

- In this part of lecture, we learn the following
 - Spectrum
 - Analog vs Digital
 - Time sampling
 - Quantization
 - Pulse Width Modulation

- How is it related to IoT?
 - Physical world events are continuous, while MCU operate on digital signals.
 - For reading and interpreting continuous input data from sensors
 - For generating continuous output data for actuators

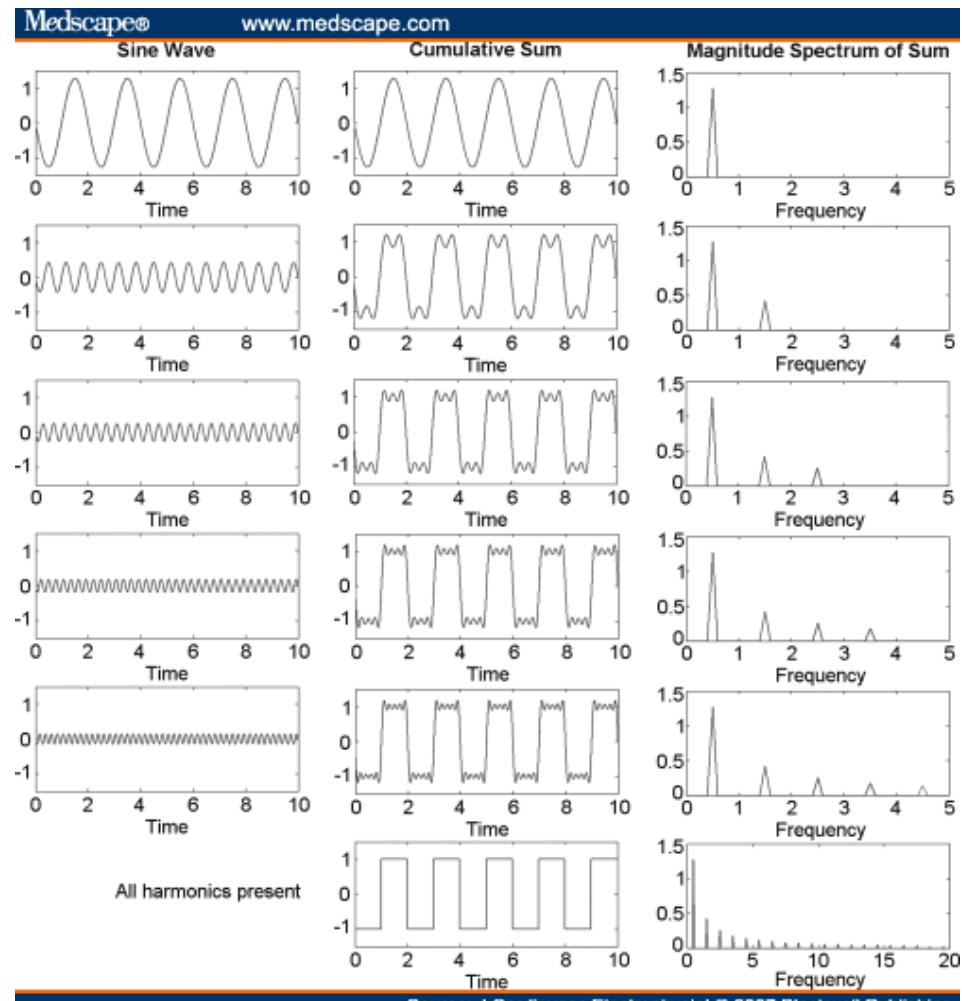
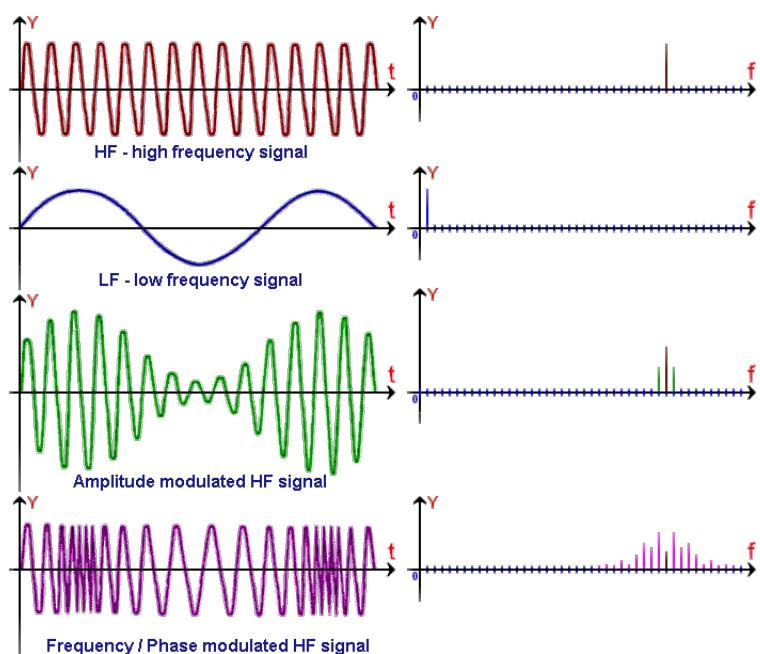
What Is a Signal

- A function that carries information in **Time** and **Frequency**



What Is a Spectrum?

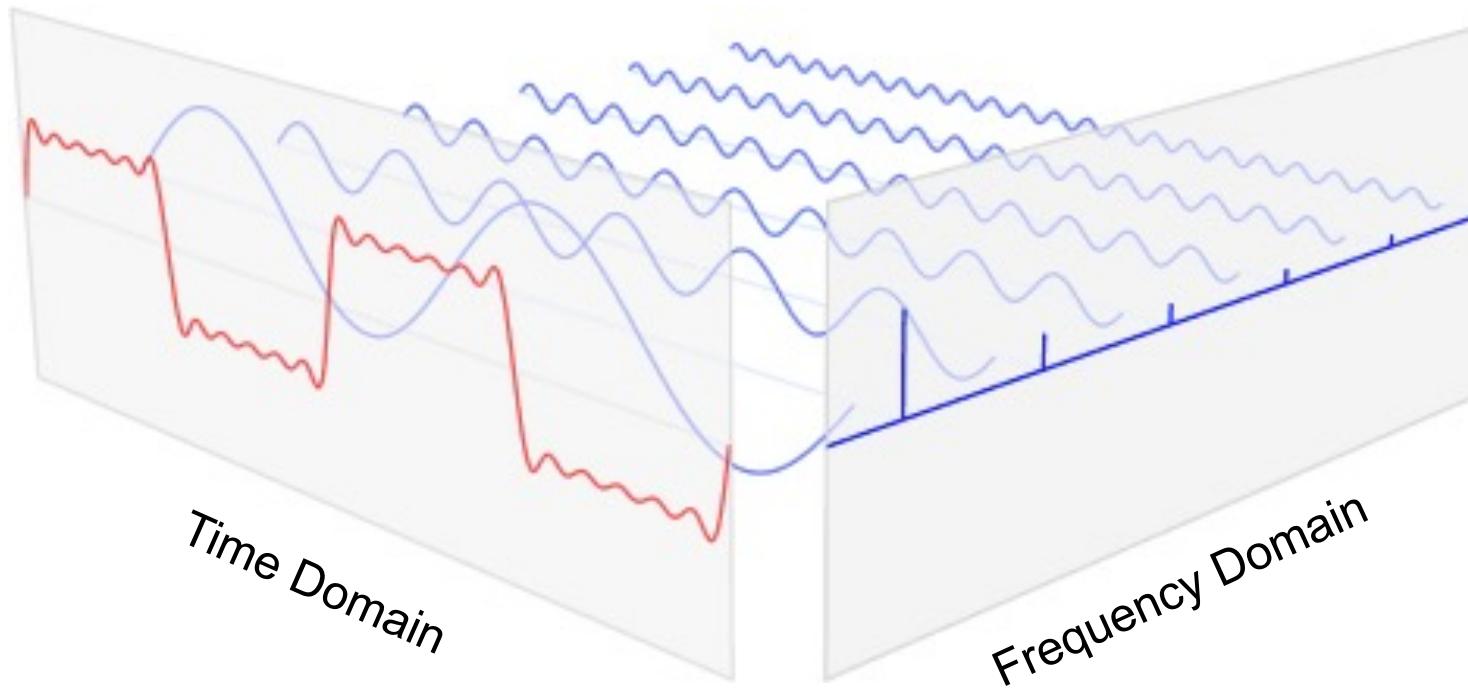
- Spectrum is the **representation of a signal in frequency domain**.



Source: J Cardiovasc Electrophysiol © 2007 Blackwell Publishing

Visualizing The Spectrum!

- Visualizing the relationship between time and frequency domain.



Analog vs Digital Signals

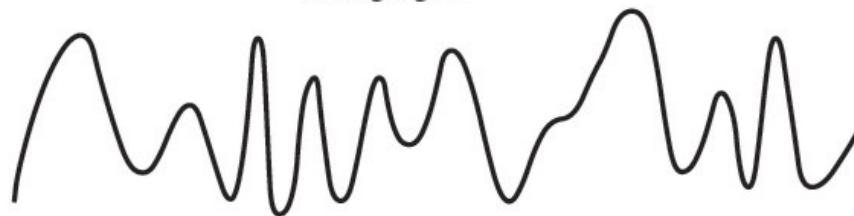
- Analog

- Continuous
 - Usually output of sensors
 - ADC → analog to digital converter

- Digital

- Discrete/sampled
 - Ones and zeros
 - What we process
 - DAC → digital to analog converter

analog signal

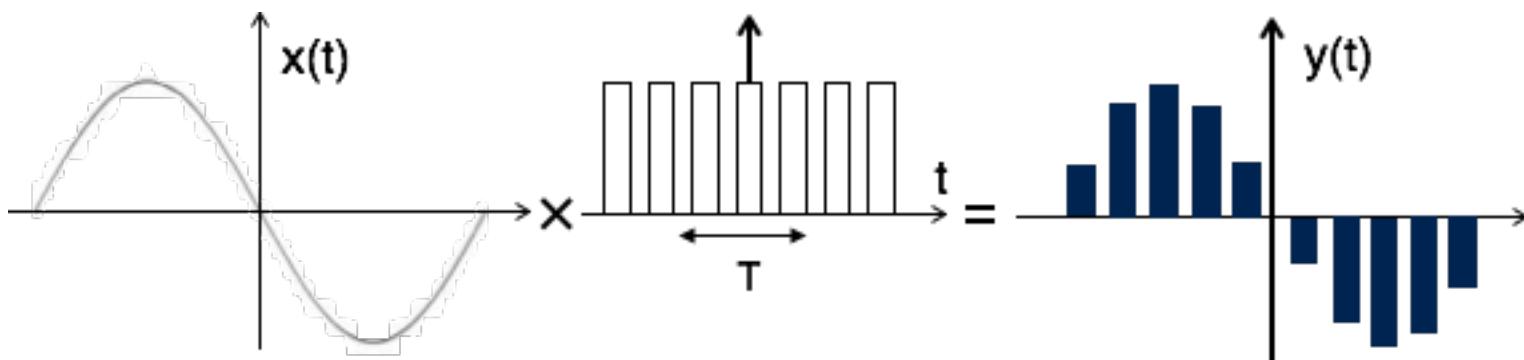
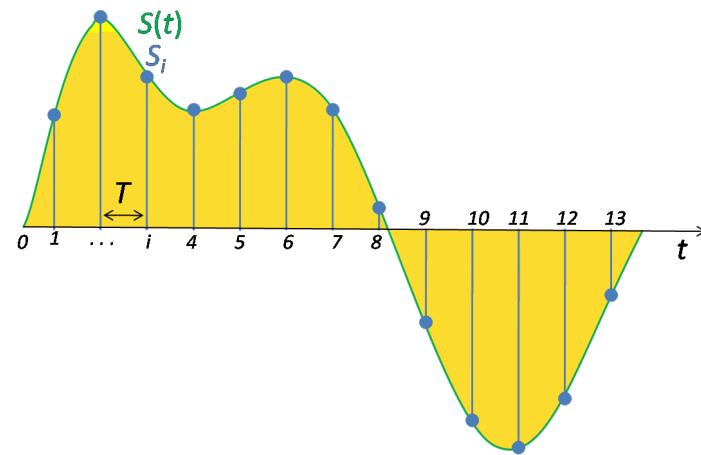


digital signal



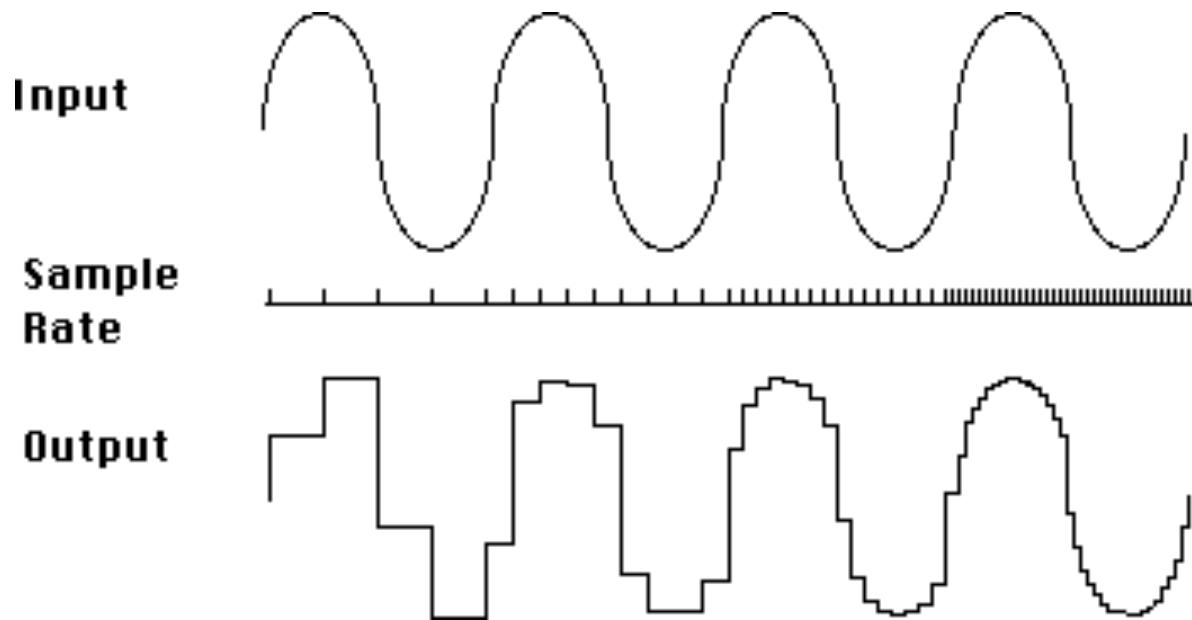
Time Sampling (discrete-time signals)

- F = frequency of sampling
- T = sampling period
- $F = 1/T$



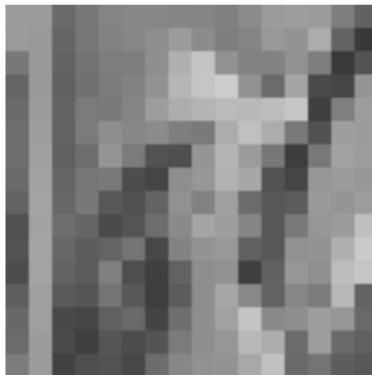
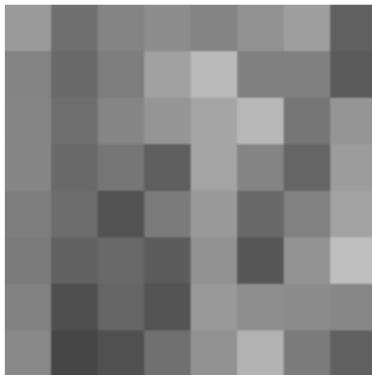
Discrete Time Signals

- Higher sampling rate
 - (+) Higher accuracy
 - (-) Higher power consumption



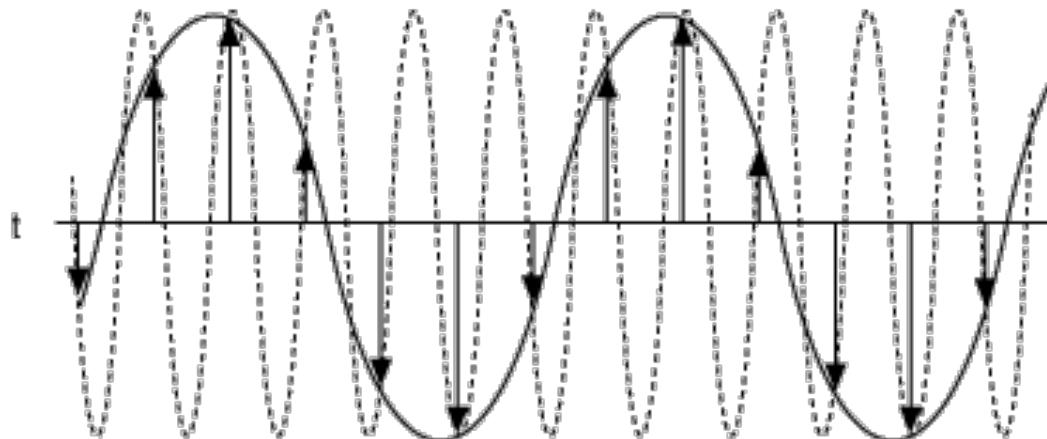
Sampling Two Dimensional Signals

- Pictures are two-dimensional spatial signals
- Videos are three-dimensional spatio-temporal signals
- Below sampling of picture Lena with different spatial sampling rates
 - 8×8 , 16×16 , 32×32 , and 128×128 samples (from left to right)
 - Each sample is represented with $n = 8$ bits
 - Each square represents average of luminance values it covers



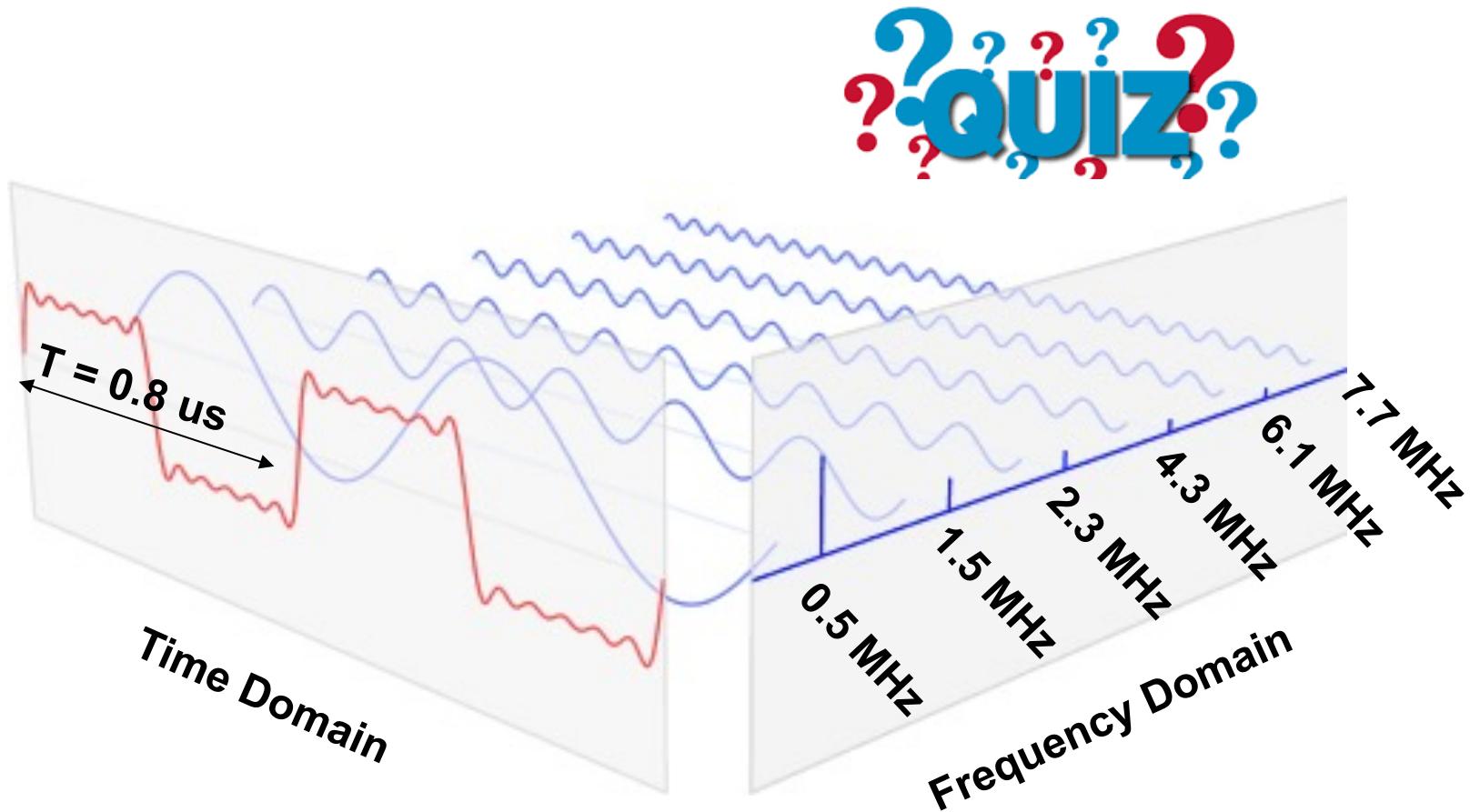
Nyquist-Shannon Theorem

- If a function $x(t)$ contains no frequencies higher than *B Hertz*, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart.
 - If the sampling frequency is less than $1/(2B)$ a wrong wave form could be sampled (An example is shown below)
- **Rule:**
- The sampling frequency F_s should be equal or greater than $2B$
 - $F_s >= 2B$



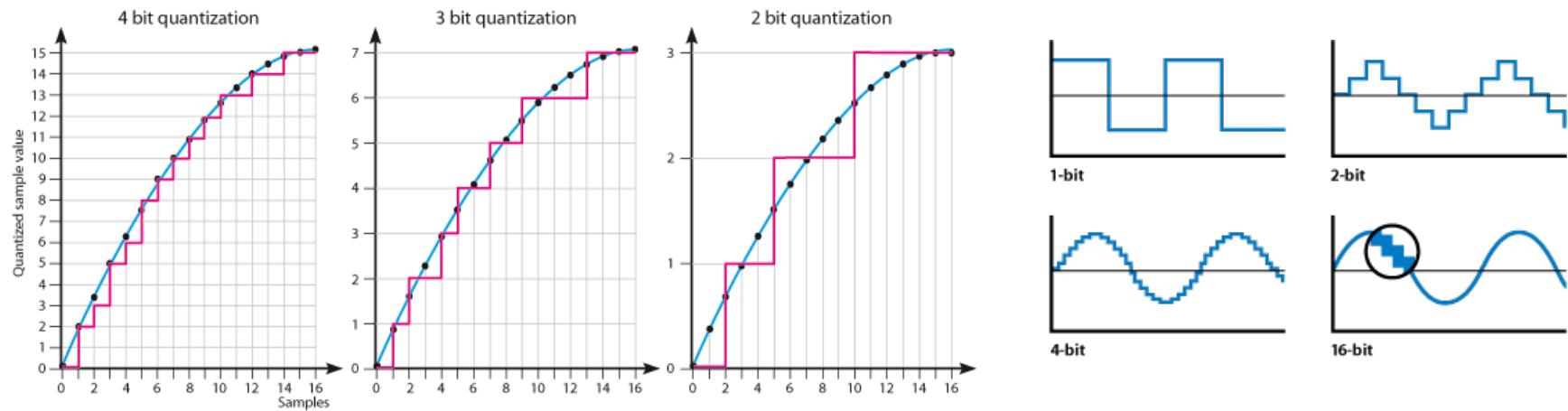
Quiz:

- What should our sampling Period (T) be?



Signal Quantization

- **Quantization** is the process of constraining an input from a continuous or otherwise-large set of values (such as the real numbers) to a discrete set.
- Number of quantization bits defines the accuracy of signal estimation.
 - More quantization bits → (+) higher accuracy → (-) but more data to process and communicate
 - Less quantization bits → (-) less accuracy → (-) but less data to process and communicate
 - E.g. representing a gray color with 8 bit → 256 shades of gray (high quality image)
representing a gray color with 4 bits → 16 shades of gray (low quality image)



Two Dimensional Signal Quantization

- Below quantization of picture Lena with different bits/sample
 - $k = 1, 2, 4, and 8 bits/sample (from left to right)$
 - The spatial sampling rate is fixed to 128×128



Signal Quantization

- A parrot image quantized
 - **W** = width of image in pixels (lets consider)
 - **H** = height of image in pixels
 - **3** channels of Red, Green and Blue (RGB) are needed.
 - Lets consider $W \times H = 544 \times 372$ (WebTV image size)
- Size of image
 - 3-bit RGB = $W \times H \times 3 \times 1 = 544 \times 372 \times 3 \times 1 = 607104 = \sim 75.8\text{KBytes}$
 - 6-bit RGB = $W \times H \times 3 \times 2 = 544 \times 372 \times 3 \times 2 = 1214208 = \sim 151.8\text{ KBytes}$
 - 24bit RGB = $W \times H \times 3 \times 8 = 544 \times 372 \times 3 \times 8 = 4856832 = \sim 607.1\text{ Kbytes}$

3-bit RGB

$2^{1 \times 3}$ colors



6-bit RGB

$2^{2 \times 3}$ colors



9-bit RGB

$2^{3 \times 3}$ colors



12-bit RGB

$2^{4 \times 3}$ colors



15-bit RGB

$2^{5 \times 3}$ colors



18-bit RGB

$2^{6 \times 3}$ colors



24-bit RGB

$2^{8 \times 3}$ colors



Signal Quantization

- Lets consider we can communicate 1MByte/S and for each MByte/S we consume 1mW.
 - 3-bit RGB → =~ 75.8ms to transfer, 75.8uW to communicate → consume low power.
 - 6-bit RGB → =~ 151.8ms to transfer, 151.8uW to communicate → consume twice the power.
 - 24bit RGB → =~ 607.1ms to transfer, 607.1uW to communicate → consume 8 times the power.

3-bit RGB

$2^{1 \times 3}$ colors



6-bit RGB

$2^{2 \times 3}$ colors



9-bit RGB

$2^{3 \times 3}$ colors



12-bit RGB

$2^{4 \times 3}$ colors



15-bit RGB

$2^{5 \times 3}$ colors



18-bit RGB

$2^{6 \times 3}$ colors



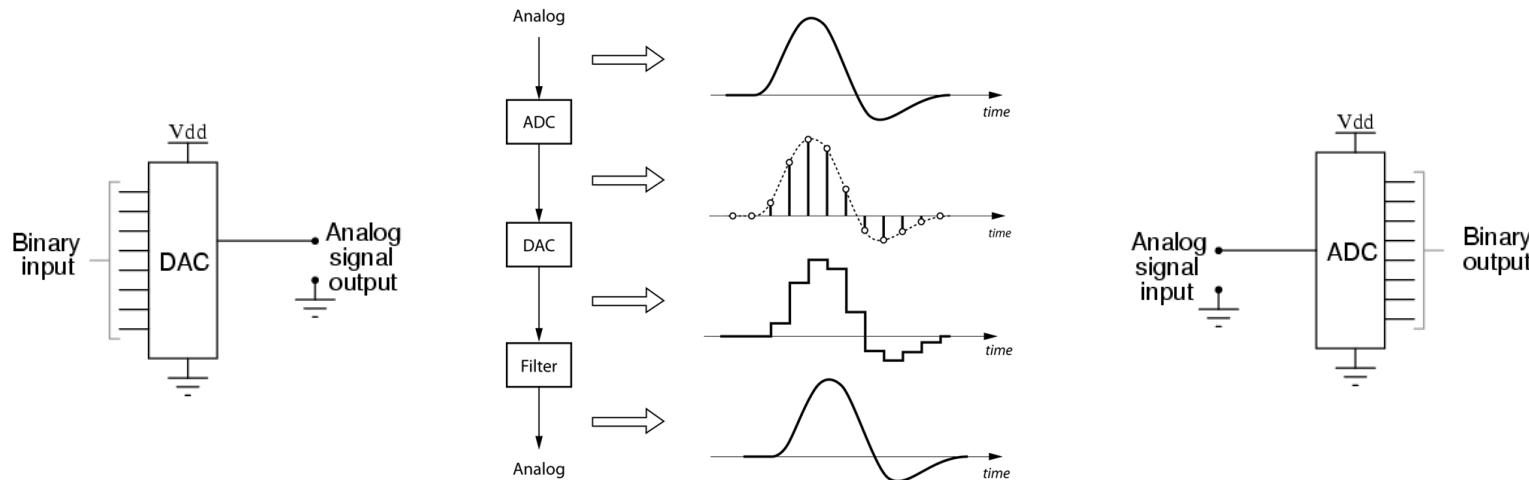
24-bit RGB

$2^{8 \times 3}$ colors

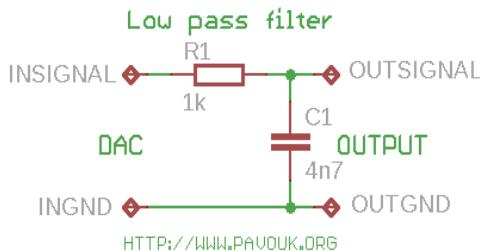


Signal Conversation

- Digital to Analog converter
 - **DAC, D/A, D–A, D2A, or D-to-A**
 - A device that converts a digital signal into an analog signal
- Analog to Digital converter
 - **ADC, A/D, A–D, A2D, or A-to-D**
 - A device that converts an analog signal into a digital signal



Generating an analog signal from a discrete signal require a reconstruction *filter* to **smooth** the discrete steps.



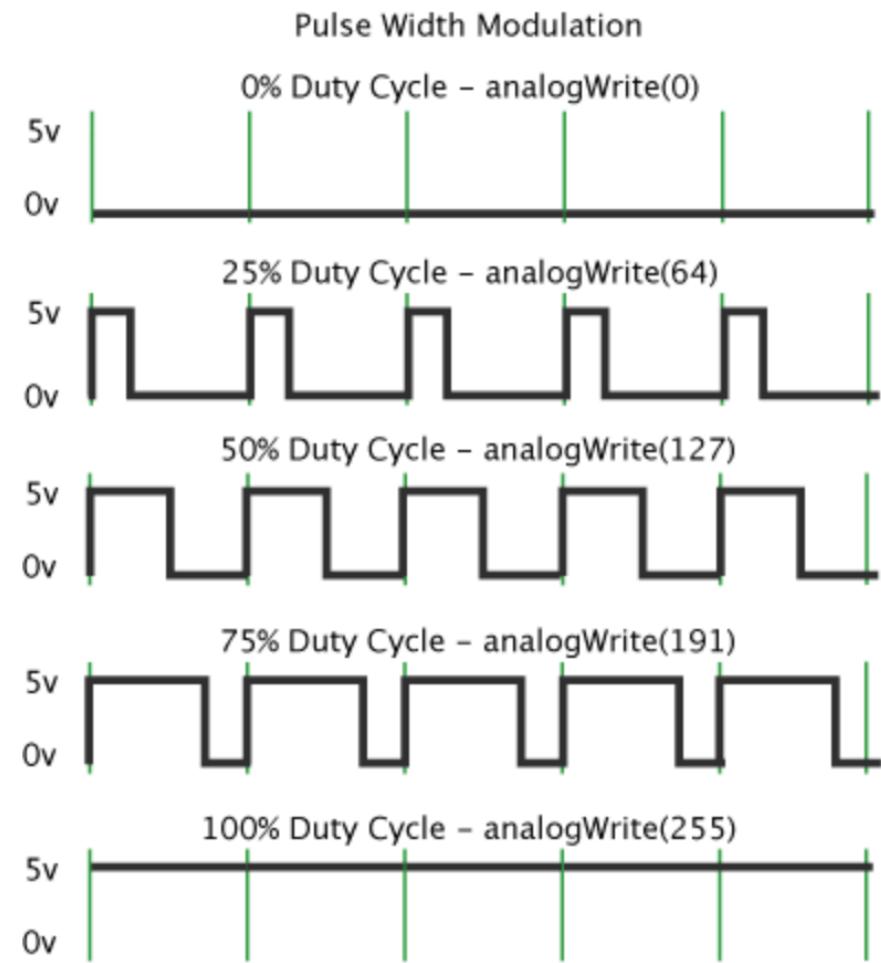
Your Assignment for the next class

- **Can we sample a signal by lower sampling frequency as determined by Nyquist-Shannon Theorem?**
 - Hint: Maybe by using compress sensing?

- **Your assignment:** Research, Read online articles, and write a 1-page report on how compress sensing works and what are its limitations!
 - **Due date: Nov 11th (upload to canvas)!**

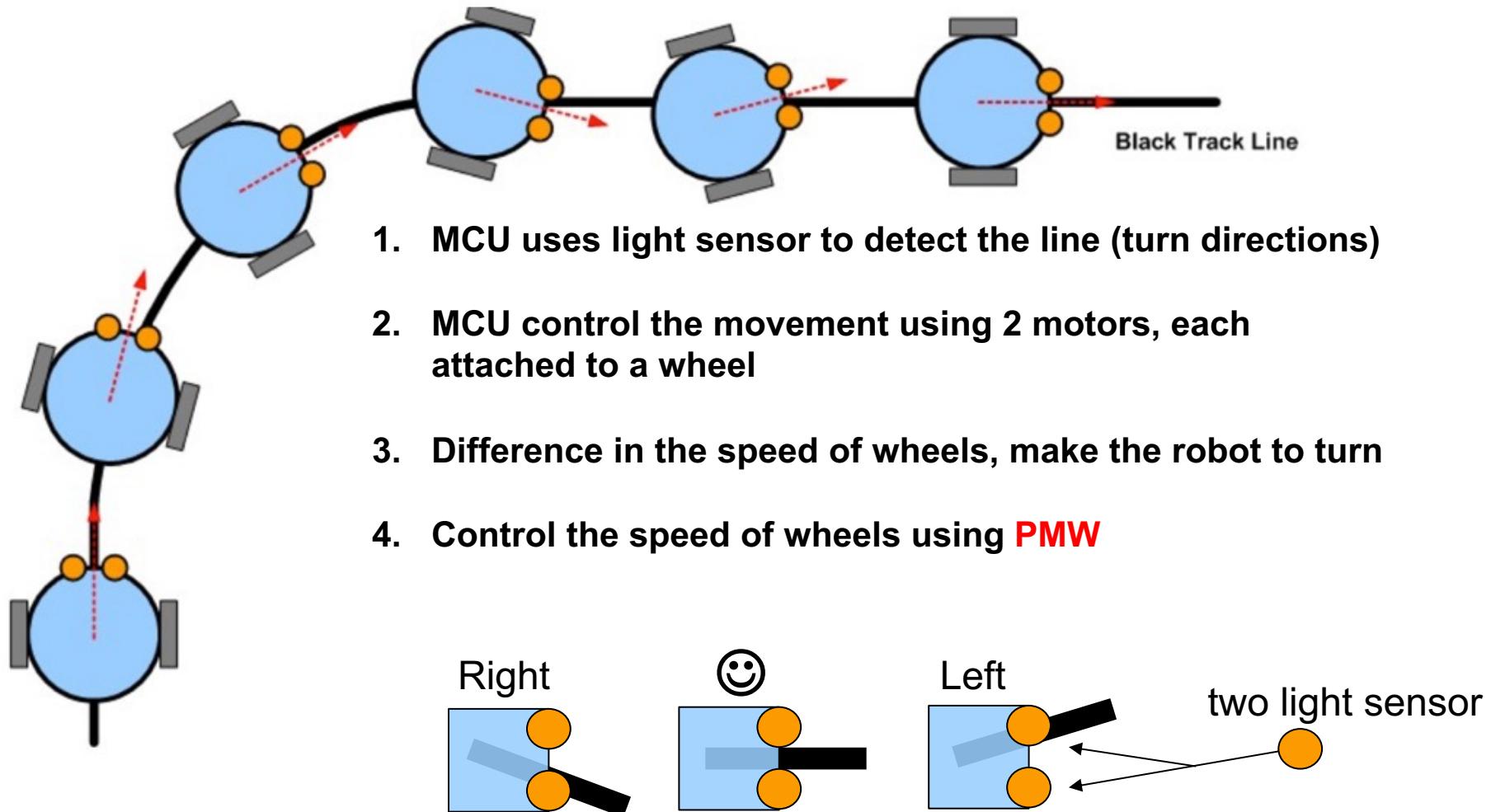
Pulse Width Modulation (PWM)

- A technique for getting analog results with digital means
- **Pulse Width:** The duration of "on time"
- **PWM:** The duration of High (1) and Low (0) Voltage is controlled.
- If you repeat this on-off pattern fast enough the result is as if the signal is a steady voltage between 0 and 5v.
 - With an LED, for example, you can control the brightness of the LED
 - With a motor, you can control the speed of motor.



A Cute Example!

Line Tracking Navigation of The Line Follower Robot (LFR)



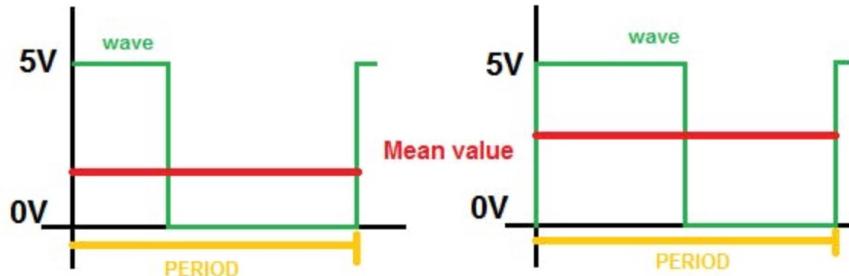
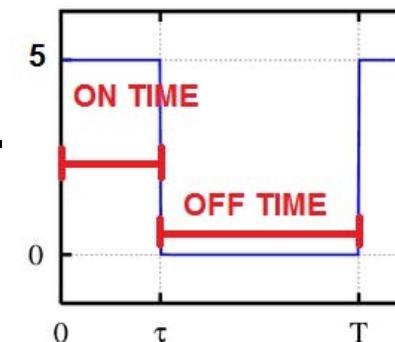
What If MCU doesn't have a DAC!!!!!!

- Use an external DAC!
- Can we build a simple one? 😊
- **Our Goal:** To translate numeric values into analog signals using a MCU (Arduino in our case!)
 - have output voltages variable from 0 to 5V by setting only a variable.
- What do you need:
 - Operational Amplifier (you can use almost every operational amplifier, just remember to check that it is rail-to-rail).
 - 22uF capacitor
 - 3.3kOhm resistor.



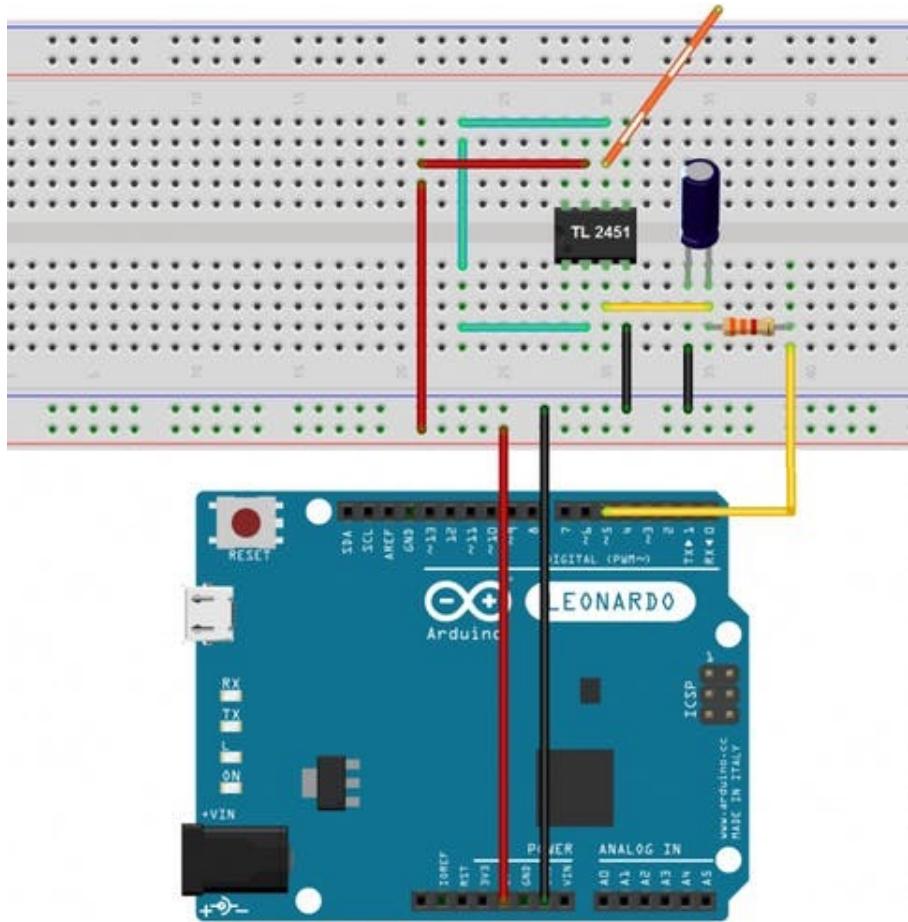
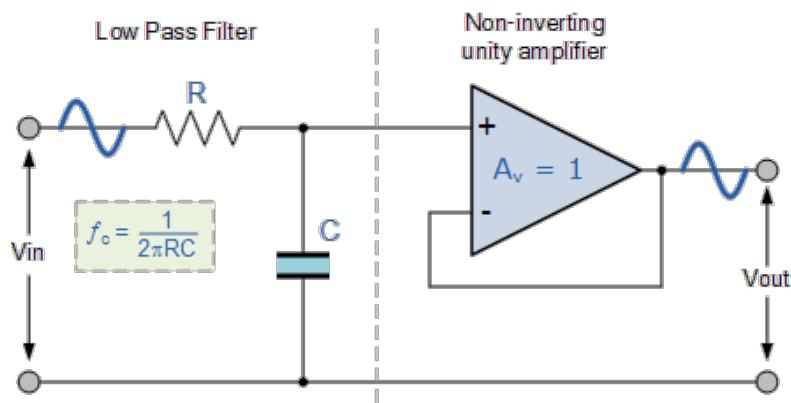
Using PWM and Duty Cycle

- **analogWrite()** function allow you to set the output duty cycle
 - vary between 0% and 100% by choosing values with the second parameter you pass to the function (0 - 255)
 - ☺ This means Analog pin has built in PWM
 - `analogWrite(pin,127)` is 50% duty cycle (default).
- **Mean or DC = ON TIME/(OFF TIME + ON TIME)**



- You can extract the DC value of the signal by using a **low-pass filter** that allows only the DC signal to pass and blocks nearly all others signals.

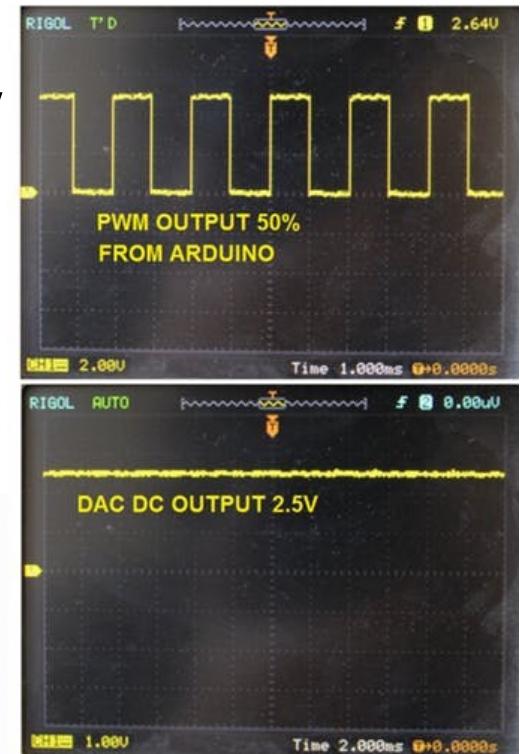
Build Your Filter Circuit



Using the Circuit

- **analogWrite(5,127)** is 50% duty cycle on pin 5.
- The DC voltage is then **50%** of supply voltage (5V in this case!) which is 2.5V

```
void setup() {  
    // initialize the digital pin as an output.  
    pinMode(5, OUTPUT);  
}  
  
void loop() {  
    analogWrite(5,127); //Set the PWM at 50 % on digital pin 5  
}
```

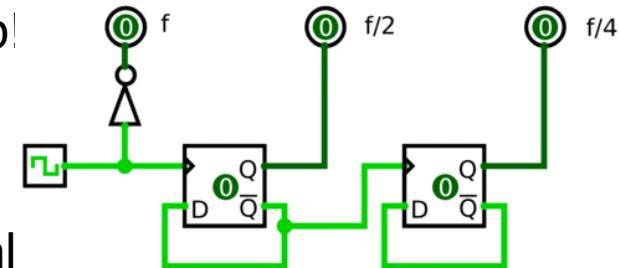


Know Your MCU DAC and ADC

- The conversion of A2D and D2A takes time.
 - Depending on the type of ADC or DAC converter used, you have a cap on how fast you can convert!
 - Depending on frequency of MCU clock you have a cap!
 - Depending on clock **divider used**, you have a cap!

- Example:
 - If **analogRead()** in Arduino takes 100us for signal conversion → 10,000 samples per second.
 - Can you sample a sound signals 1KHz, 4KHz, 10KHz, 14KHz?

- If you need higher sampling rate in Arduino, there are ways (**out of scope of this class**)
 - Such as playing with clock divider variable (presale)
 - Using ADC free running code which is based on Interrupts
 - If interested to know more [click here!](#)





Internet of Things

Senior Design Project Course

***Digital Communication
protocols***

Part 2

Lecturer: Avesta Sasan

University of California Davis

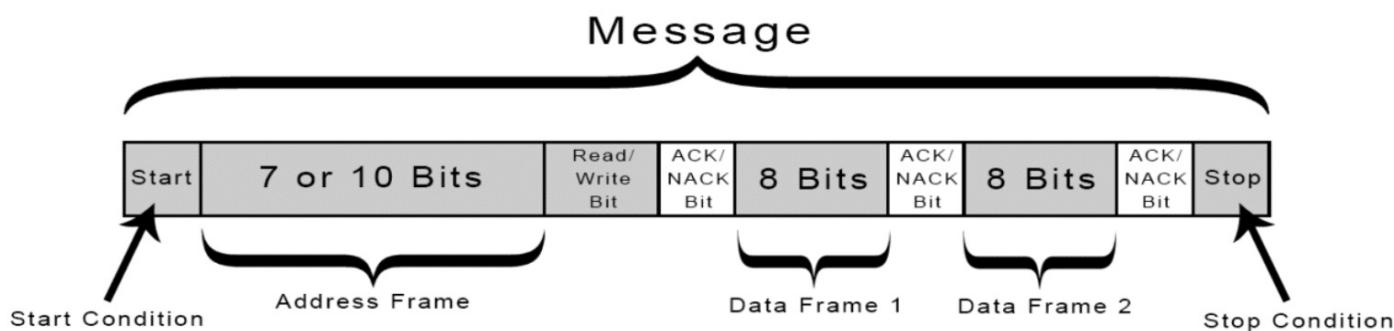
Digital Sensors (Review)

- To read digital sensors you need to use the sensor's **interface protocols**
- Some common interface protocols are:
 - Universal Asynchronous Receiver/Transmitter (**UART**)
 - Asynchronous
 - https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter
 - Serial Peripheral Interface (**SPI**)
 - Synchronous
 - Single Master Multiple Slaves.https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
 - Inter-Integrated Circuit (**I2C**)
 - Synchronous
 - Multi Master Multiple Slaves. <https://en.wikipedia.org/wiki/I2C>



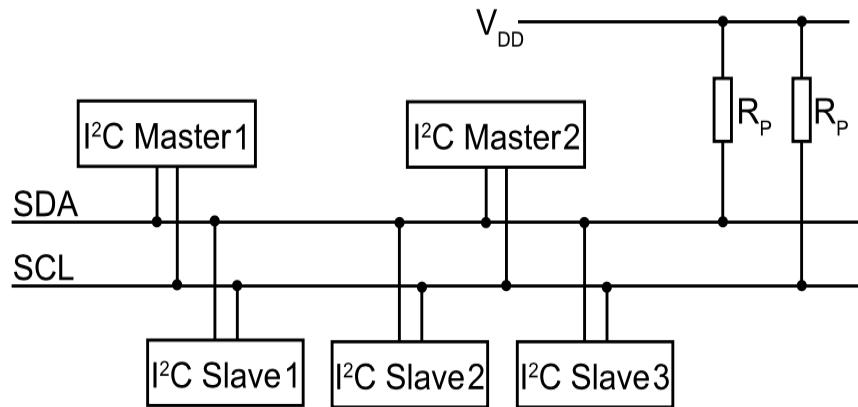
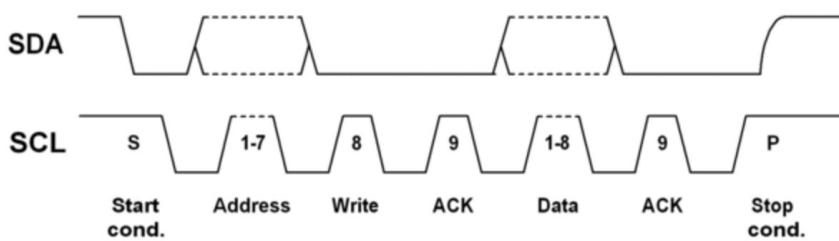
I2C Device Addressing

- I2C transfer data in form of messages.
- Messages are broken down into segments shown in the image below
 - **Address Frame:** A 7 or 10 bit sequence unique to each slave that identifies the slave when the master wants to talk to it.
 - **Start Condition:** The SDA line switches from a high voltage level to a low voltage level *before* the SCL line switches from high to low.
 - **Stop Condition:** The SDA line switches from a low voltage level to a high voltage level *after* the SCL line switches from low to high.
 - **Read/Write Bit:** A single bit specifying whether the master is sending data to the slave (low voltage level) or requesting data from it (high voltage level).
 - **ACK/NACK Bit:** Each frame in a message is followed by an acknowledge/no-acknowledge bit. If an address frame or data frame was successfully received, an ACK bit is returned to the sender from the receiving device.



I²C Device Addressing

- The **first byte** of an I²C transfer always contains the **slave address** and the **data direction**.
- Usually the address is transferred with the **MSB first**.

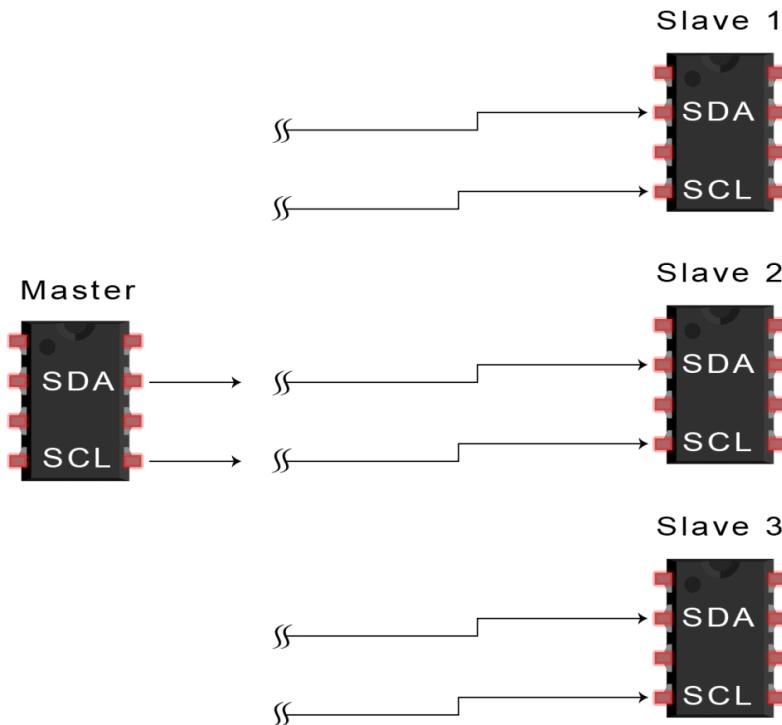


- Following web page is a good source to learn how to use I²C:
 - <https://www.i2c-bus.org/addressing/>

I2C Device Addressing

- The master sends the start condition to every connected slave:
- **How:** Switching the SDA line from a high voltage level to a low voltage level *before* switching the SCL line from high to low:

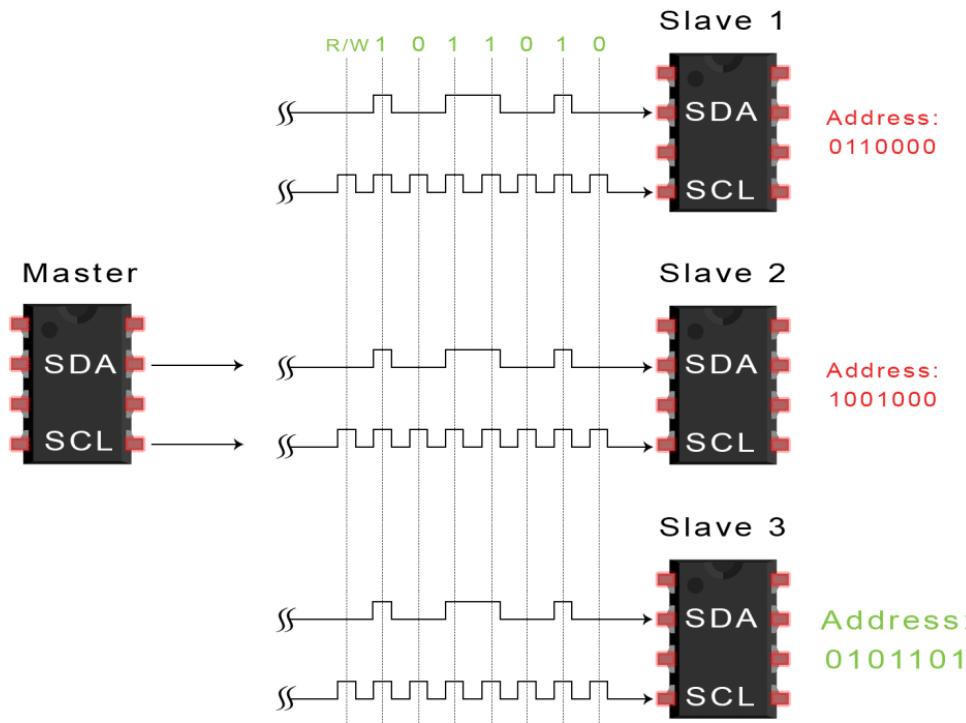
1



I2C Device Addressing

- The master sends each slave the 7 or 10 bit address of the slave it wants to communicate with, along with the read/write bit:

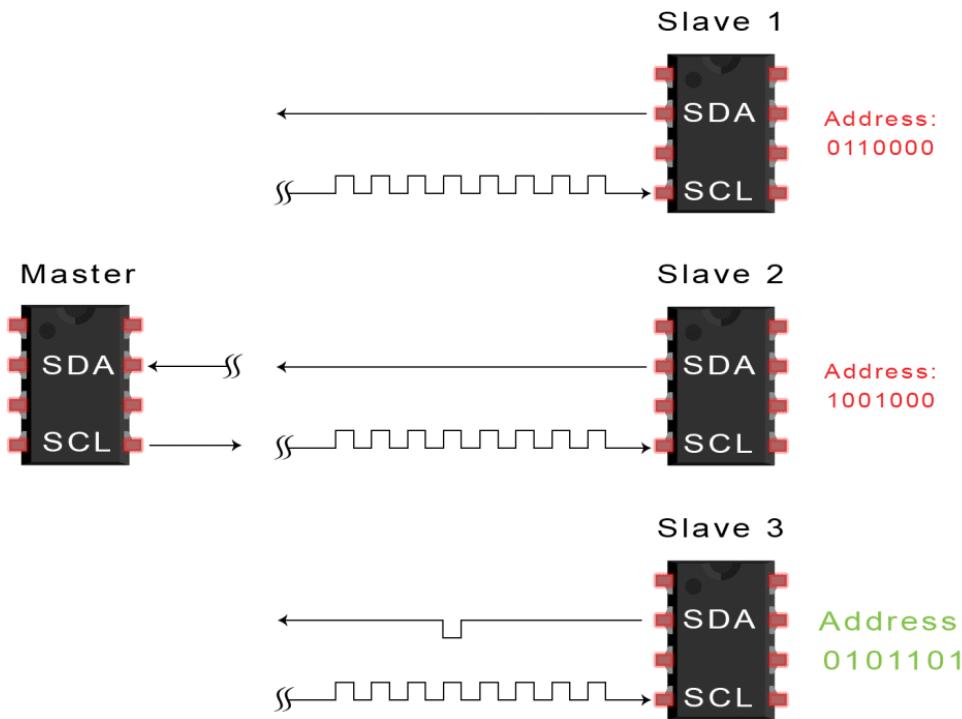
2



I2C Device Addressing

- Each slave compares the address sent from the master to its own address.
 - If matches, the slave returns an ACK bit by pulling the SDA line low for one bit.
 - If does not match the slave's own address, the slave leaves the SDA line high.

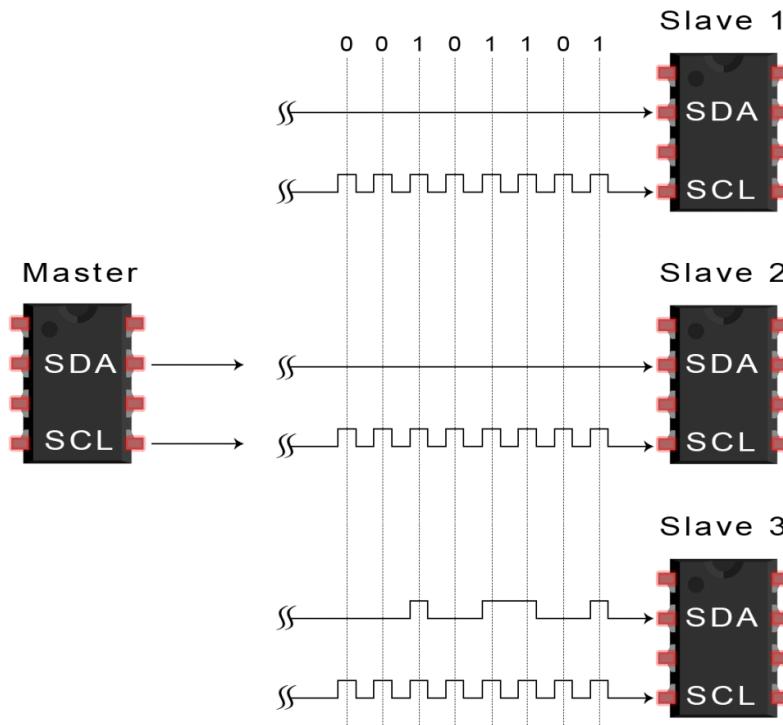
3



I2C Device Addressing

- The master sends or receives the data frame:
- Can you tell if the data is being sent from the master to slave or from slave to the master in this example?

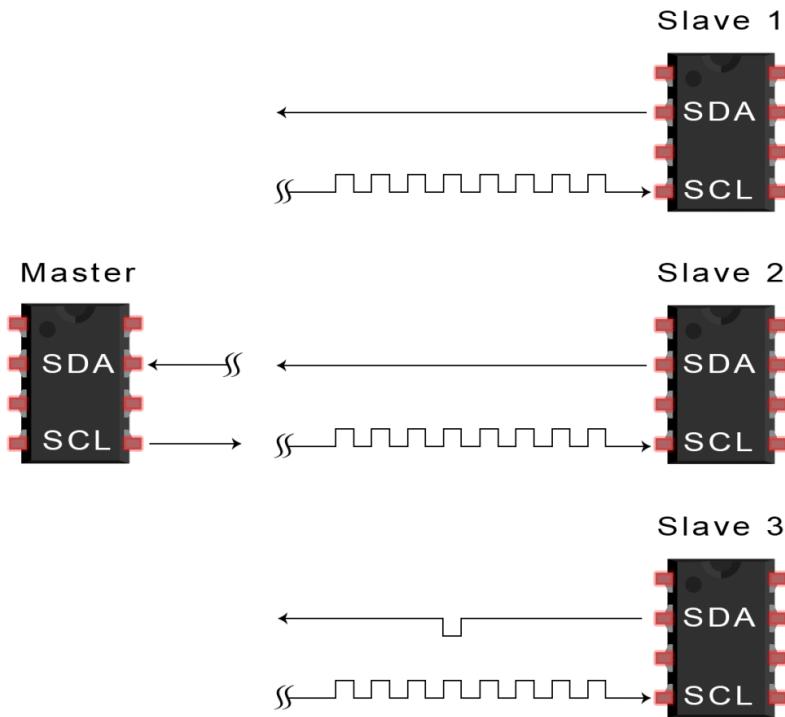
4



I2C Device Addressing

- After each data frame has been transferred, the receiving device returns another ACK bit to the sender to acknowledge successful receipt of the frame:

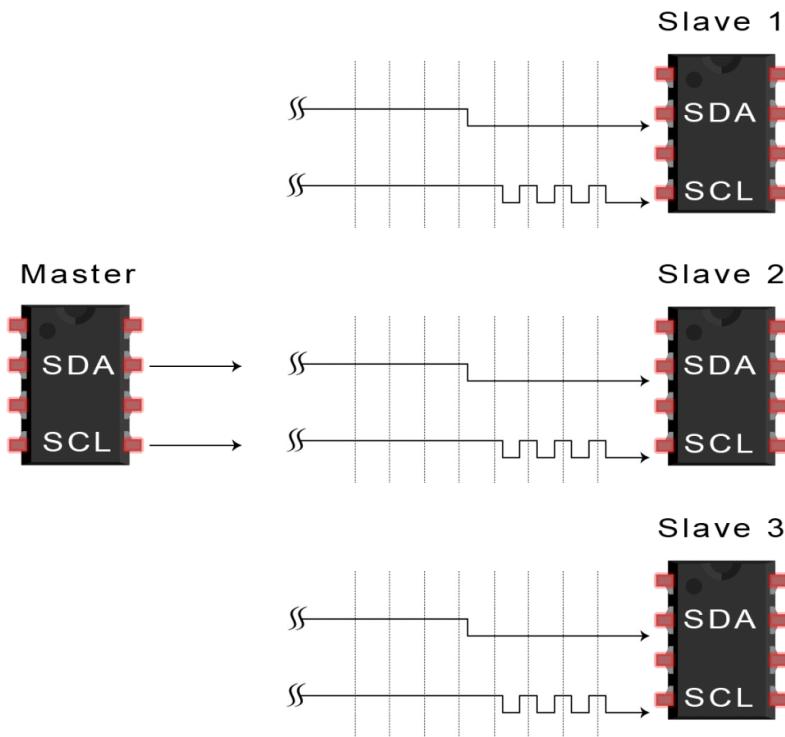
5



I2C Device Addressing

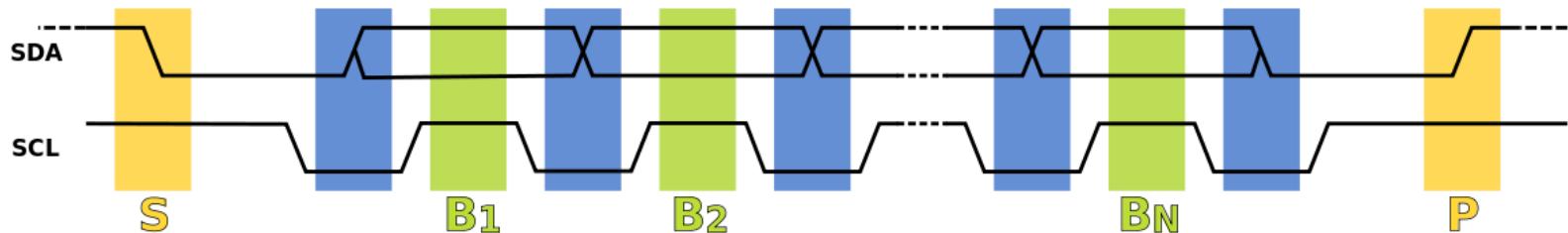
- To stop the data transmission, the master sends a stop condition to the slave by switching SCL high before switching SDA high:

6



When I2C Slaves Look for Their Device Address?

- During the data transfer SDA changes while SCL is LOW
 - After Falling edge of SCL
 - Before Rising edge of SCL
- Data transfer is initiated with **SDA being pulled low while SCL stays high.**
 - This condition **does not happen again during the data transfer!**
 - After this condition, devices look for the next 7 bits for address, if address doesn't match they ignore the data transfer and look for stop bit!
- A *stop* bit (P) is signaled when **SDA is pulled high while SCL is high.**
 - **This condition doesn't happen during the data transfer.**



Power and Energy Comparison

- Comparing UART, SPI and I²C Energy and Power consumption:
- The communication protocol is implemented in both HW and SW
- The data rate is set at 15.8 Kbps.

Parameters	UART		SPI		I ² C				
	1 byte	9 bytes	RX selected	9 bytes	1 byte	9 bytes	1 byte	9 bytes	
<i>Interface implementation in hardware</i>									
Power consumption with inactive interface, mW	6.06	6.06	4.66	4.66	4.66	4.66	6.13	6.13	6.13
<i>Communication:</i>									
Required time, ms	0.68	5.78	0.52	4.6	0.52	4.6	1.26	5.91	1.26
Required energy, μJ	7.31	53.31	2.5	23.53	2.42	22.52	8.11	31.71	8.01
Normalized energy ^a , μJ	7.27	53.01	2.48	23.33	2.4	22.32	8.19	31.99	8.08
<i>Interface implementation in software</i>									
Power consumption with inactive interface, mW	11.04	11.04	11.17	11.17	11.17	11.17	11.23	11.23	11.23
<i>Communication:</i>									
Required time, ms	0.64	5.72	0.51	4.62	0.51	4.61	1.23	5.85	1.23
Required energy, μJ	9.00	80.06	7.26	65.25	7.26	65.25	17.45	82.84	17.44
Normalized energy ^a , μJ	9.01	80.16	7.28	65.42	7.28	65.42	17.45	82.84	17.44

Source: <http://ieeexplore.ieee.org/document/6208716/>

Implement Protocol in HW or SW?

Parameters	UART		SPI		I^2C	
	1 byte	9 bytes	RX selected	RX unselected	RX selected	RX unselected
<i>Interface implementation in hardware</i>						
Power consumption with inactive interface, mW	6.06	6.06	4.66	4.66	4.66	6.13
<i>Communication:</i>						
Required time, ms	0.68	5.78	0.52	4.6	0.52	1.26
Required energy, μJ	7.31	53.31	2.5	23.53	2.42	22.52
Normalized energy ^a , μJ	7.27	53.01	2.48	23.33	2.4	22.32
<i>Interface implementation in software</i>						
Power consumption with inactive interface, mW	11.04	11.04	11.17	11.17	11.17	11.23
<i>Communication:</i>						
Required time, ms	0.64	5.72	0.51	4.62	0.51	1.23
Required energy, μJ	9.00	80.06	7.26	65.25	7.26	65.25
Normalized energy ^a , μJ	9.01	80.16	7.28	65.42	7.28	65.42

- Software implementation consumes more power than hardware implementation!
 - SW implementation, should be executed with general purpose processing logic, hence more operations are needed!
 - Customized HW is always more efficient!
 - HW could use tricks, such as going to sleep mode, using interrupts, ...

- **Conclusion:** If low power solutions is desired, look for Microprocessor that has the HW implementation for the desired communication protocol.

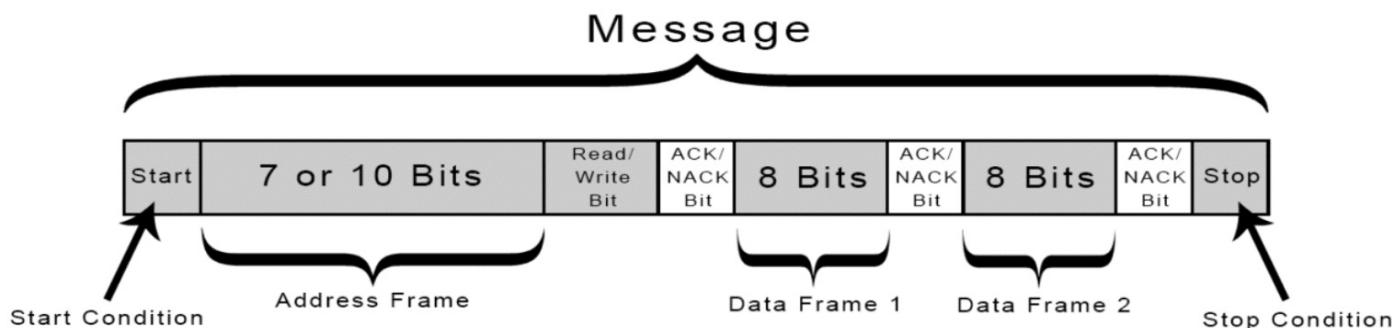
Power and Energy Comparison

Parameters	UART		SPI		I^2C	
	1 byte	9 bytes	RX selected	RX unselected	1 byte	9 bytes
<i>Interface implementation in hardware</i>						
Power consumption with inactive interface, mW	6.06	6.06	4.66	4.66	4.66	6.13
<i>Communication:</i>						
Required time, ms	0.68	5.78	0.52	4.6	0.52	4.6
Required energy, μJ	7.31	53.31	2.5	23.53	2.42	22.52
Normalized energy ^a , μJ	7.27	53.01	2.48	23.33	2.4	22.32
<i>Interface implementation in software</i>						
Power consumption with inactive interface, mW	11.04	11.04	11.17	11.17	11.17	11.23
<i>Communication:</i>						
Required time, ms	0.64	5.72	0.51	4.62	0.51	4.61
Required energy, μJ	9.00	80.06	7.26	65.25	7.26	65.25
Normalized energy ^a , μJ	9.01	80.16	7.28	65.42	7.28	65.42

- Compared to UART, I2C is less efficient for smaller number of bytes, but become more efficient as number of bytes increases.
- **Reason:** I2C need to send an extra byte for device identification.
 - For 1-Byte packet overhead is 100% when a single byte is transferred
 - Overhead is a much smaller % when number of transfer bytes is large!
 - For 10 bytes and larger I2C is more efficient than UART
- **Conclusion:** For power efficiency know your sensor behavior (data packet size) to choose the best protocol.

Question:

- Sensor will be sending 28 bytes of data to the microcontroller. The UART uses a data size of 8bit. How many clock cycles are needed for UART communication? How many clock cycles are needed for I2C communication? Which is more efficient? Repeat the same question if sensor need to send 7 bytes of data?



Power and Energy Comparison

Parameters	UART		SPI				I^2C					
	1 byte	9 bytes	RX selected	9 bytes	RX unselected	1 byte	9 bytes	RX selected	9 bytes	RX unselected	1 byte	9 bytes
<i>Interface implementation in hardware</i>												
Power consumption with inactive interface, mW	6.06	6.06	4.66	4.66	4.66	4.66	6.13	6.13	6.13	6.13	6.13	6.13
<i>Communication:</i>												
Required time, ms	0.68	5.78	0.52	4.6	0.52	4.6	1.26	5.91	1.26	5.91		
Required energy, μJ	7.31	53.31	2.5	23.53	2.42	22.52	8.11	31.71	8.01	30.23		
Normalized energy ^a , μJ	7.27	53.01	2.48	23.33	2.4	22.32	8.19	31.99	8.08	30.5		
<i>Interface implementation in software</i>												
Power consumption with inactive interface, mW	11.04	11.04	11.17	11.17	11.17	11.17	11.23	11.23	11.23	11.23	11.23	11.23
<i>Communication:</i>												
Required time, ms	0.64	5.72	0.51	4.62	0.51	4.61	1.23	5.85	1.23	5.86		
Required energy, μJ	9.00	80.06	7.26	65.25	7.26	65.25	17.45	82.84	17.44	83.6		
Normalized energy ^a , μJ	9.01	80.16	7.28	65.42	7.28	65.42	17.45	82.84	17.44	83.6		

- SPI is more energy efficient than I2C, but it come at a cost:
 - SPI doesn't support automatic device authentication (**Scalability**)
 - I2C identify the device by the first byte is transmits
 - SPI require wiring up the device correctly and SS (or alternatively using CS)
 - SPI doesn't support multiple Masters! (**functionality**)
 - SPI has larger wiring overhead.
- **Conclusion:** Consider the number of sensors and masters you want to connect to a device, need for adding additional sensors in the future, and the power consumption constraints to decide between I2C and SPI



Internet of Things

Senior Design Project Course

***Digital Communication
protocols***

Part 1

Lecturer: Avesta Sasan

University of California Davis

Focus of Today's Lecture

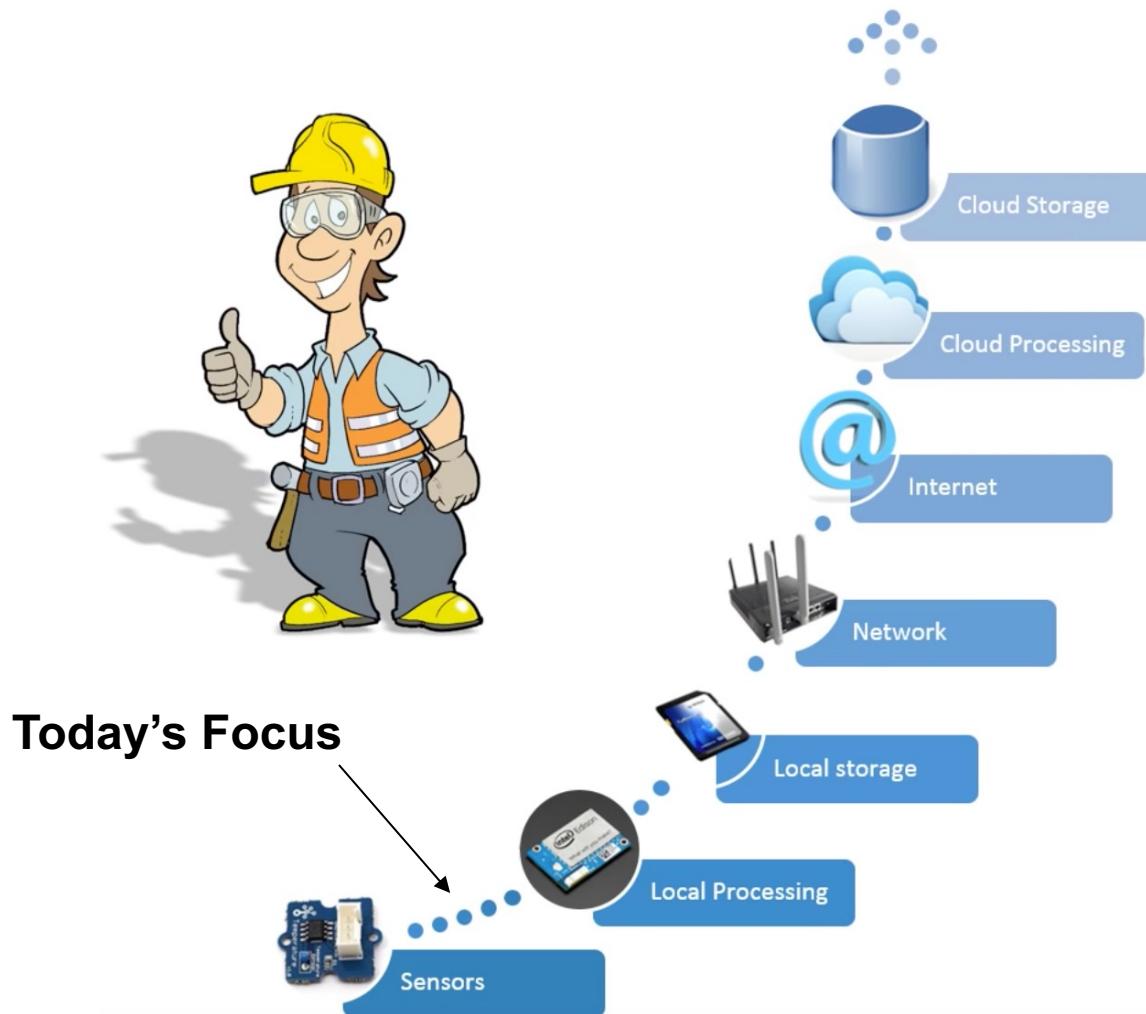
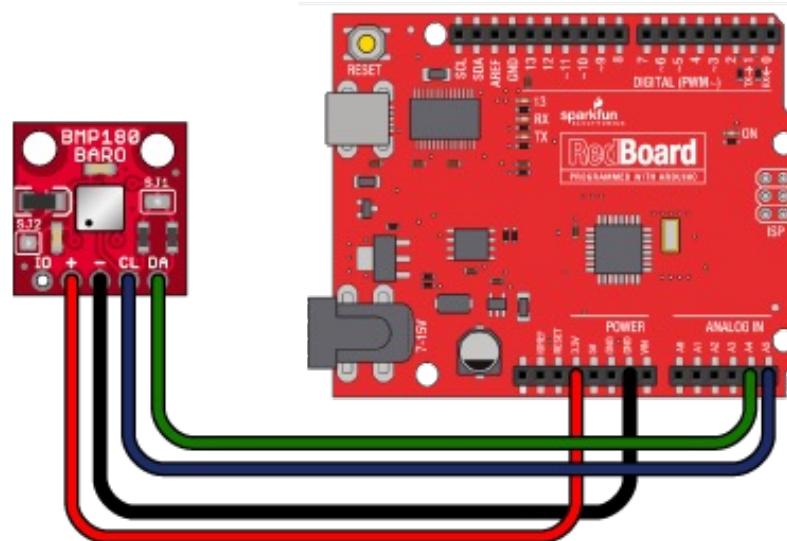


Image source: <http://www.cchc.cl/informacion-a-la-comunidad/industria-de-la-construccion/personaje/>

Communication with Digital Sensors

- To read digital sensors you need to use the sensor's **interface protocols**
- Interface Protocols could be **Synchronous or Asynchronous.**
- Interface Protocol could be **Serial or Parallel.**



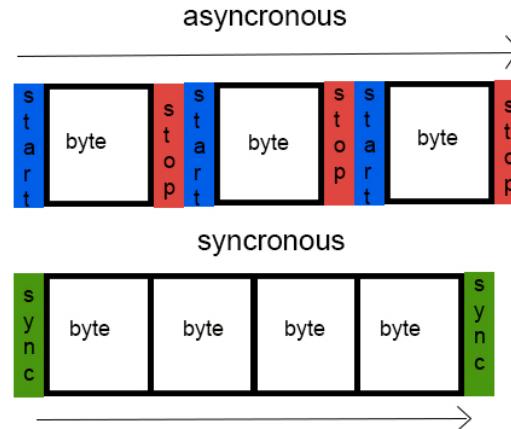
Synchronous vs Asynchronous

■ Synchronous

- ❑ Uses Clock!
- ❑ Pairs its data line(s) with a clock signal.
- ❑ Clock is shared among all devices
- ❑ Faster serial transfer
- ❑ Straight forward implementation
- ❑ Examples: **SPI, I2C**

■ Asynchronous

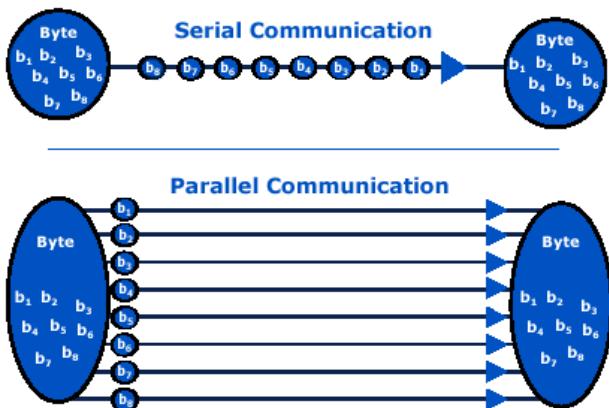
- ❑ Doesn't need a clock
- ❑ Need lower number of wires (no clock!)
- ❑ Usually slower transfer
- ❑ Implementation is more difficult
- ❑ Examples: **TTL, RS32**



Parallel vs Serial

■ Parallel Interface

- Transfer multiple bits at a time.
- Require a bus (group of wires)
- Higher performance
- Require too many IO pins



■ Serial Interface

- Transfer a single bit at a time.
- Require a single wire
- Higher overhead
- Lower performance
- Require single or only a few wires

How about a hybrid approach?

Which one is easier to implement?

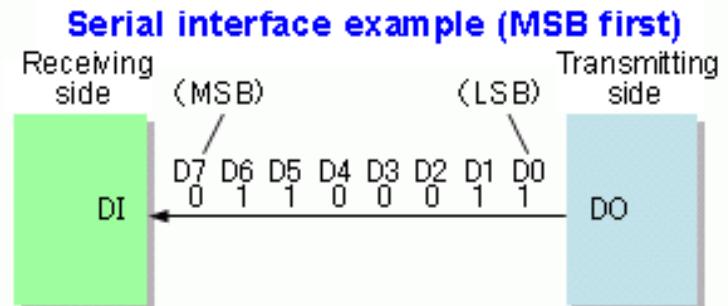
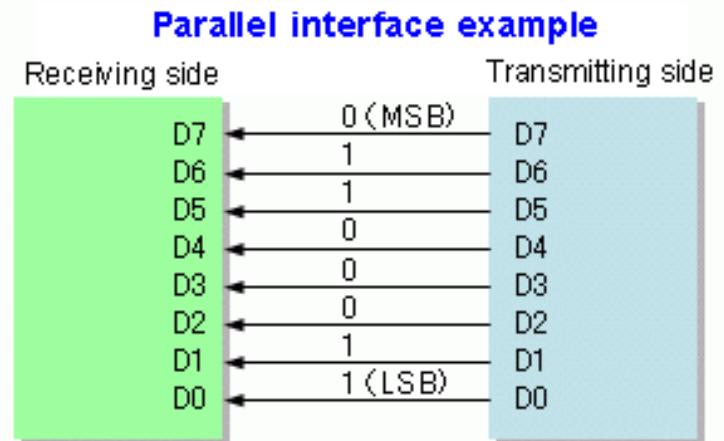
Which one is more natural to use?

Serial Communication

- To exchange information, we always need a well defined protocol.
- Many to/from sensor communication protocols exist, in this lecture we will cover a few!

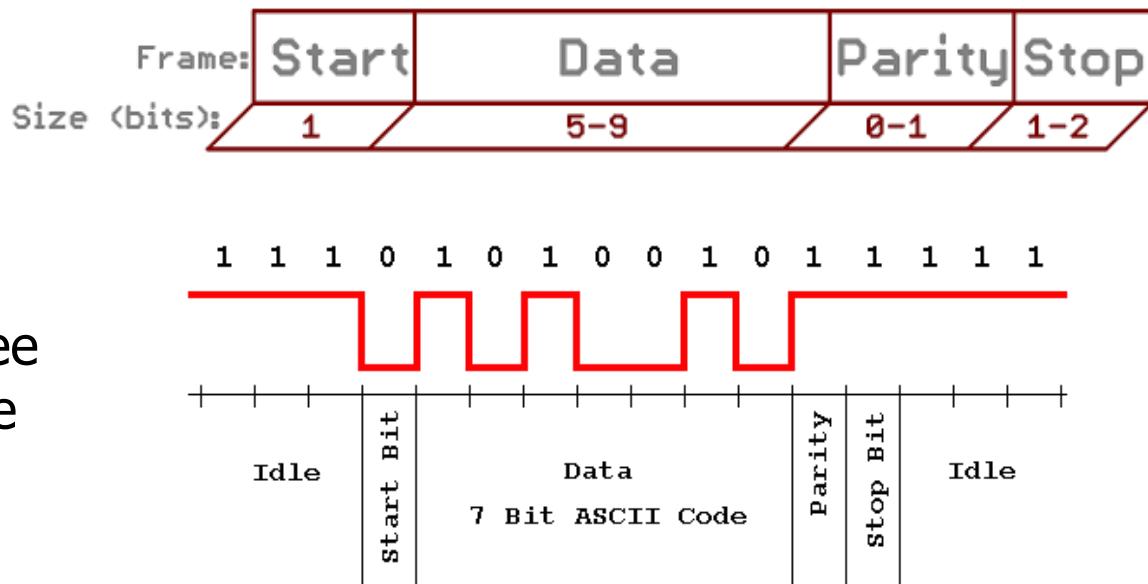


- Sensor communication could be
 - **Parallel Interface:** communicating multiple bits at the same time
 - **Serial Interface:** communicating one bit at a time



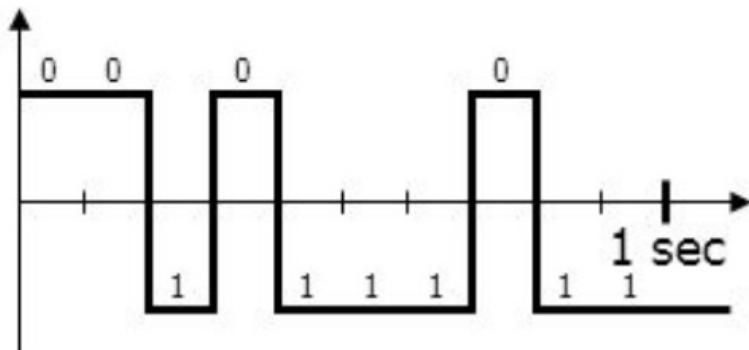
Asynchronous Comm. Protocol

- To implement the Asynchronous communication protocol, we need to build our data packets using a predefined (agreed upon) format.
- A common way to build this data packet is by using the following sections on a data transfer packet:
 - Start bit
 - Data bits
 - Parity Bit
 - Stop Bit(s)
- We also need to agree on the speed that the data is transferred!

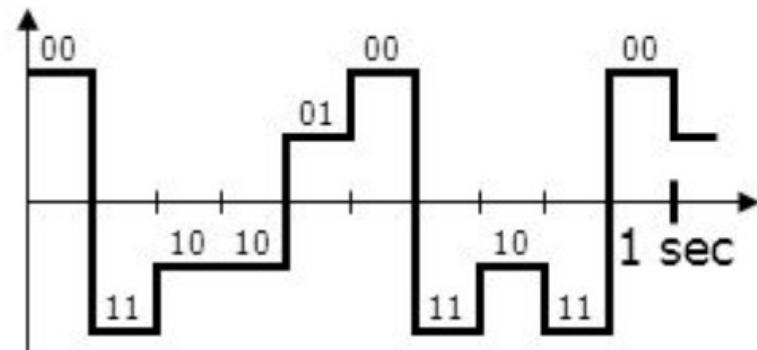


Baud Rate

- **Baud rate:** How many times a signal changes per second.
- **Bit Rate:** How many bits can be sent per second.
- Bit rate is controlled by Baud rate and the number of signal levels.
- **Can you find the relationship between Baud rate and Bitrate below?**
- **If we only have signals with two level, what is the relationship?**
- Baud rate is measured by the unit: **bits per second (bps)**
- Standard Baud rates are 200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200bps.



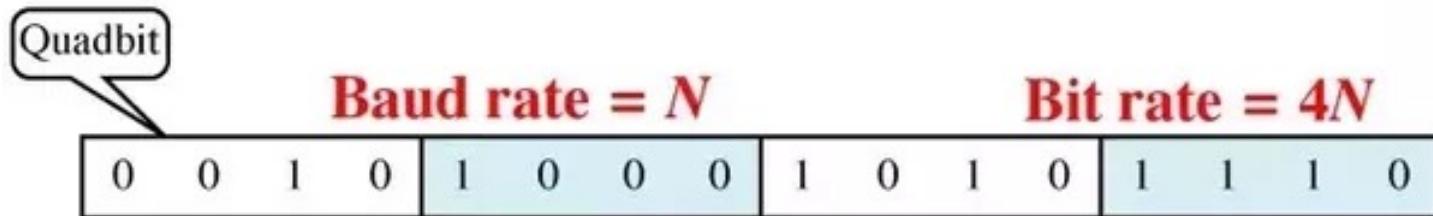
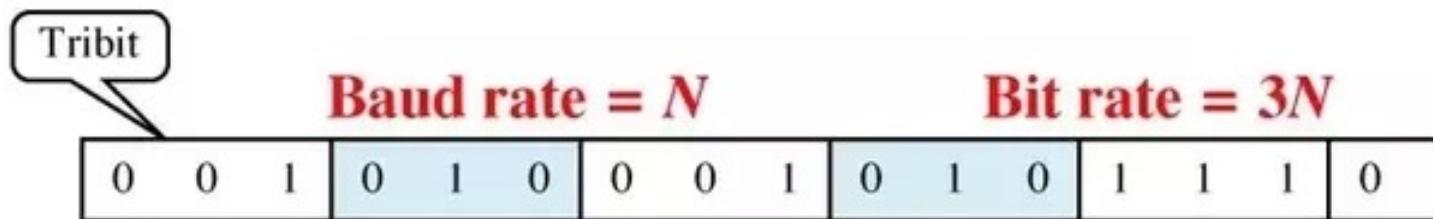
Baud = 10
Bit rate = 10 bps



Baud = 10
Bit rate = 20 bps

Baud Rate vs Bit Rate

Bit Rate and Baud Rate



Packaging the Data

- Each packet of data is called a **frame**.
- **Start Bit:** A synchronization bit that mark the beginning of the data,
 - Start bit is always one bit.
 - Indicate the start of a transition by going from 1 to 0.
- **End bit:** Synchronization bit(s) to mark the end of the data.
 - One or two bits (depending on the implementation)
 - Indicates the end of a transition by pulling the line to 1



Packaging the Data

- **Data bits:** contain the information of interest!
 - It is also called a **chunk**
 - Data could be set to a value between 5 to 9 bits
 - **8 bits is a natural choice (size of a Byte), but why use other sizes?**
 - We also have to agree if data is **Big Endian** or **Little Endian**?



Big Endian vs Little Endian

- The order that bits are transferred! MSB first or LSB first:
- **Exercise:** A Microprocessor has received the following byte (**01100100**) of information. What value it represent if it is Big Endian? What if it was Small Endian?

ANIMATED RESPONSE FOR BIG ENDIAN

ANIMATED RESPONSE FOR LITTLE ENDIAN

Packaging the Data

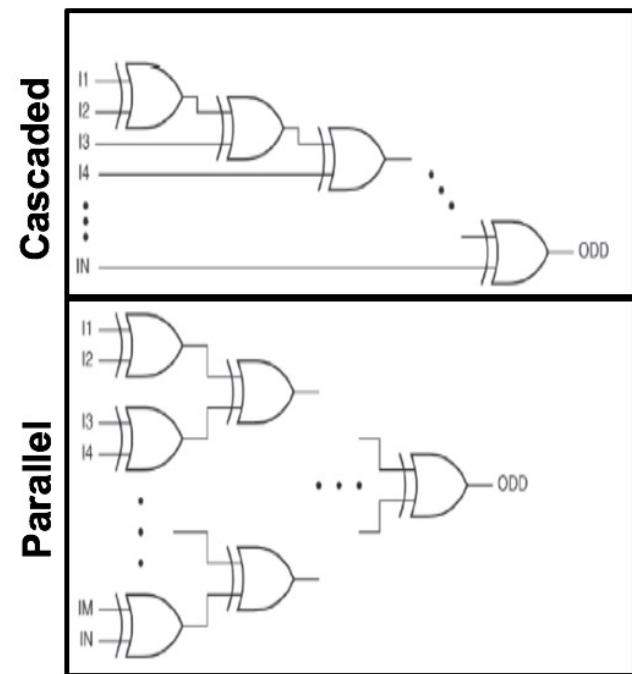
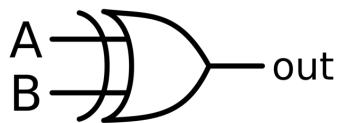
- Parity bit (**optional**): Very simple way of error checking
 - Add up the bits, if even number then it is 0, if odd number it is 1. (or vice versa)
 - Can detect one bit flip error. (can it detect two)?
 - Slow down the data transfer (need error coding encoder and decoder)
 - Improves reliability

Original Data	Even Parity	Odd Parity
00000000	0	1
01011011	1	0
01010101	0	1
11111111	0	1
10000000	1	0
01001001	1	0

- Now you know everything that you need to know to build a packet.
- **Exercise:** Build an **optimized** format for a frame that uses **little endian** format. It is used to read sensor information that are between values [0, 232]. It uses **1 bit even Parity** and we intend to send **value 208** from sensor to the receiver, and use a **single bit for end signal**.

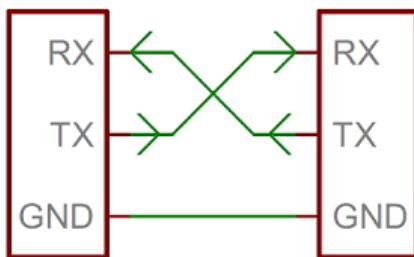
Parity Bit Generator Circuit

- Simplest way: Use XORs
- Could cascade the XORs or form a XOR tree
 - What is the tradeoff?
- **Question:** How do you change the parity between **ODD** and **EVEN**?
- **Answer:** XOR the results with value one or zero to change parity mode!

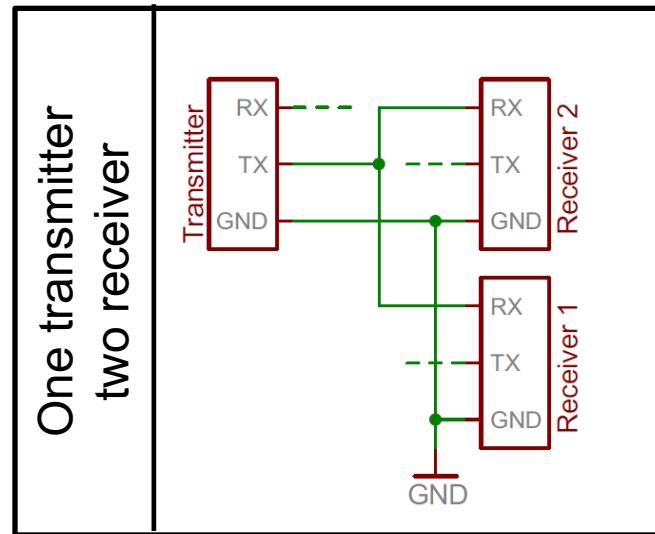
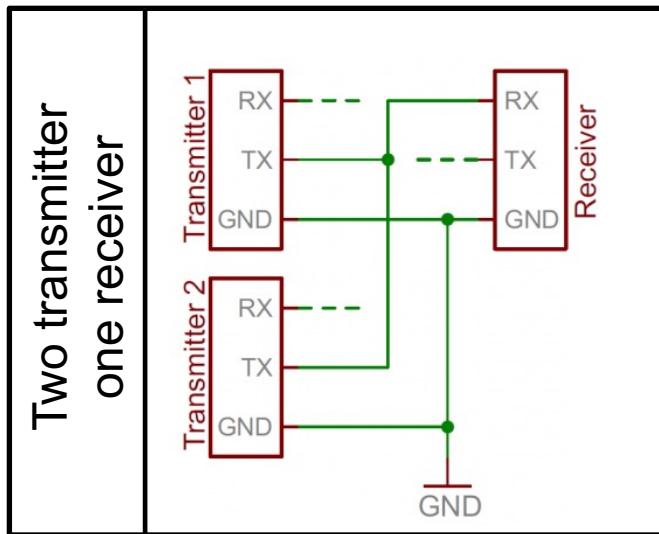


How to Connect Asynch Devices?

- TX to RX



- **Question:** How do we setup the circuit if we have multiple transmitters and/or multiple receivers?



- **Question:** Why do you need a common GND?

More Details!

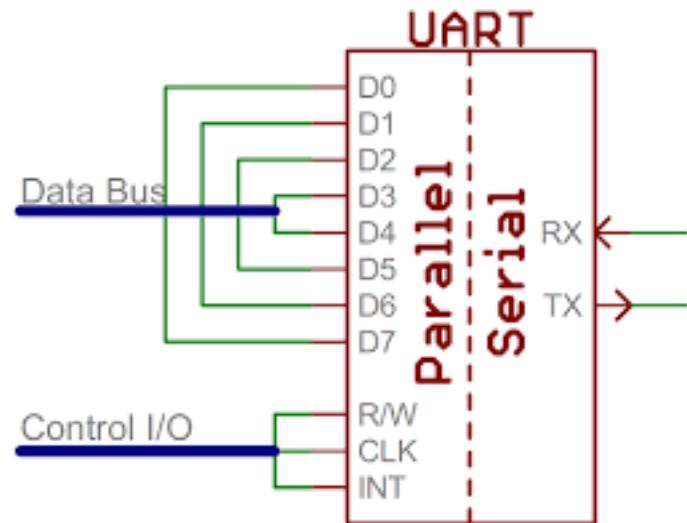
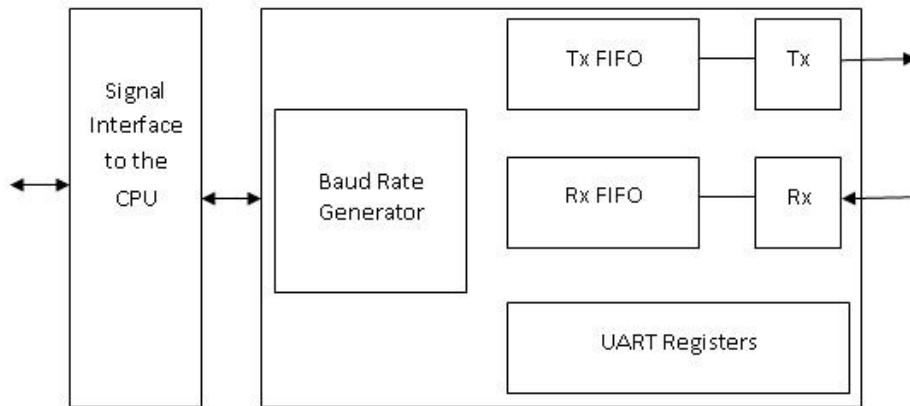
- Serial Asynchronous communication could be
 - **Full-duplex**: both devices can send and receive data simultaneously.
 - **Half-duplex**: devices should take turn in transmitting data.



- Make sure the voltages match up!

UART

- Universal Asynchronous Receiver/Transmitter (**UART**)
- Implements the serial communication!
- Act as intermediate between parallel and serial data
- Many microcontrollers have the UART block internally, but you can also have it as a standalone block.
- Refer to the following link to learn how to use UART functions in Arduino:
 - <https://www.arduino.cc/en/Reference/Serial>



Asynchronous Communication Problems

- No precise control over when data is sent/received
- **Tx** and **Rx** should agree prior to data transfer on Baud rate
- **What if clocks are slightly off?**
 - Extra start and stop bits to somewhat take care of this problem, but it is communication overhead!
- Hardware required to compose or decode the data frame is more complex than parallel communication protocols.
- **Synchronous communication solve all this problems!**
 - **As an example we see how SPI protocol work, and slightly cover I2C.**

SPI: A Synchronous Comm. Protocol

- Separate data and clock wires!
- Data is read on rising or falling edge of clock
- Clock and data are send at the same time
 - Hence, no need to pre-arranged baud rate
- Receiving hardware can be as simple as a shift register
- Clock is always generated by Master
 - But both master and slave can receive and transfer data
- Always one master, but could have multiple slaves!
 - I2C protocol solves this problem!

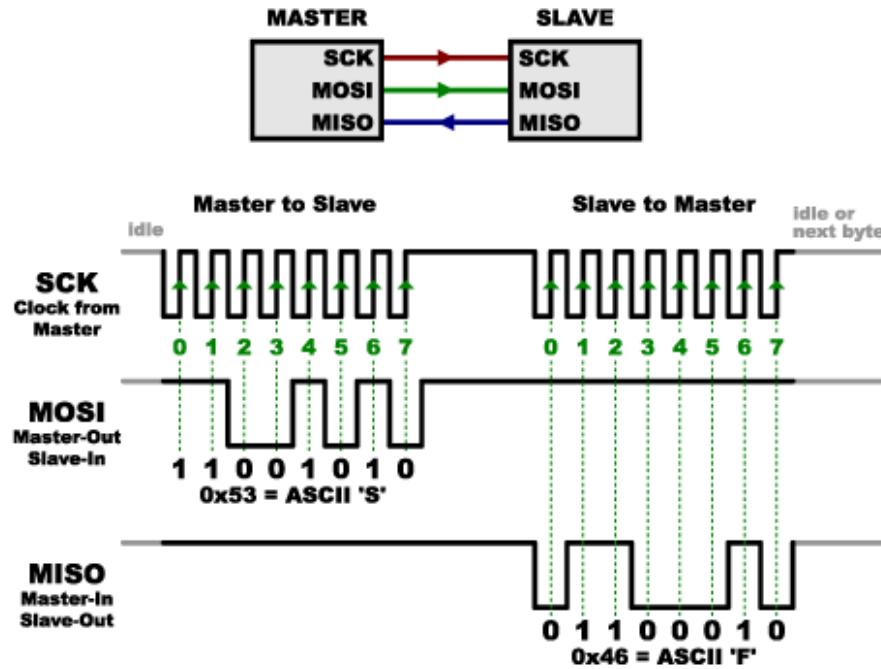
SPI: Data Exchange!

■ **MOSI:** Master Out Slave In

- Line to send data from Master to Slave

■ **MISO:** Master In Slave Out

- Line to send data from Slave to Master
- The number of cycles is **pre-arranged**, generated by Master, during which slave toggle the MISO



SPI: Data Exchange

- Pre-arranging the number of needed clock cycles could be achieved by

- **Knowing the Command behavior in advance!**

Using known command to communicate with sensors whose responses are known and are always fixed in length!



- **Data length is communicated:**

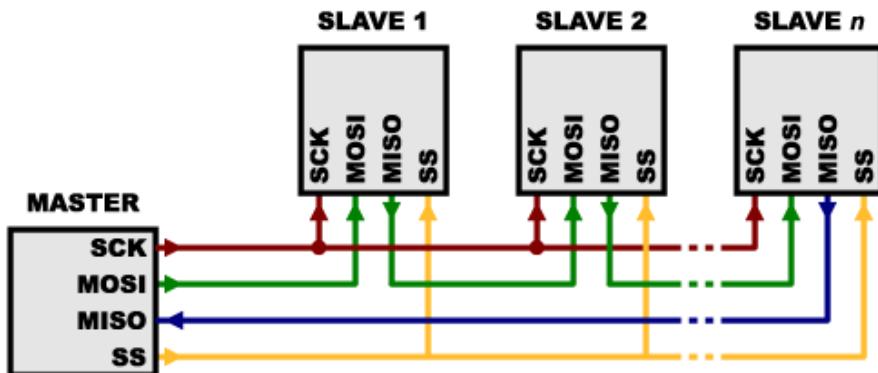
This is a two step process!

1. The sensor send a fixed response (1 or 2 bytes), by which it specifies the length of data.
2. The Master generates the number of needed clock cycles to transfer the specified data from Slave to Master.

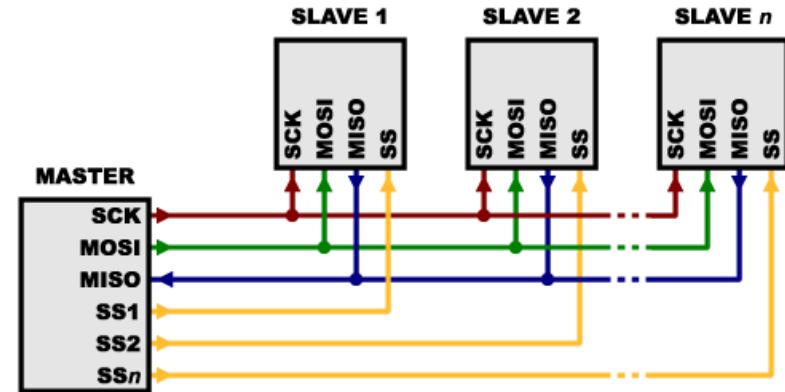


SPI: Slave Select

- SPI has **one Master**, but could have **multiple slaves**
 - How do we select the slave?
- Slave Select (SS) or Chip Select (CS) is used to select the slave node that SPI communicate with.
- Two way to connect multiple Slaves:
 - **Direct:** Data is directly transferred to the target.
 - **Daisy Chain:** Data flow through many slaves to reach target.



Daisy Chain Connection



Direct Connection

Using SPI on Arduino Microprocessor

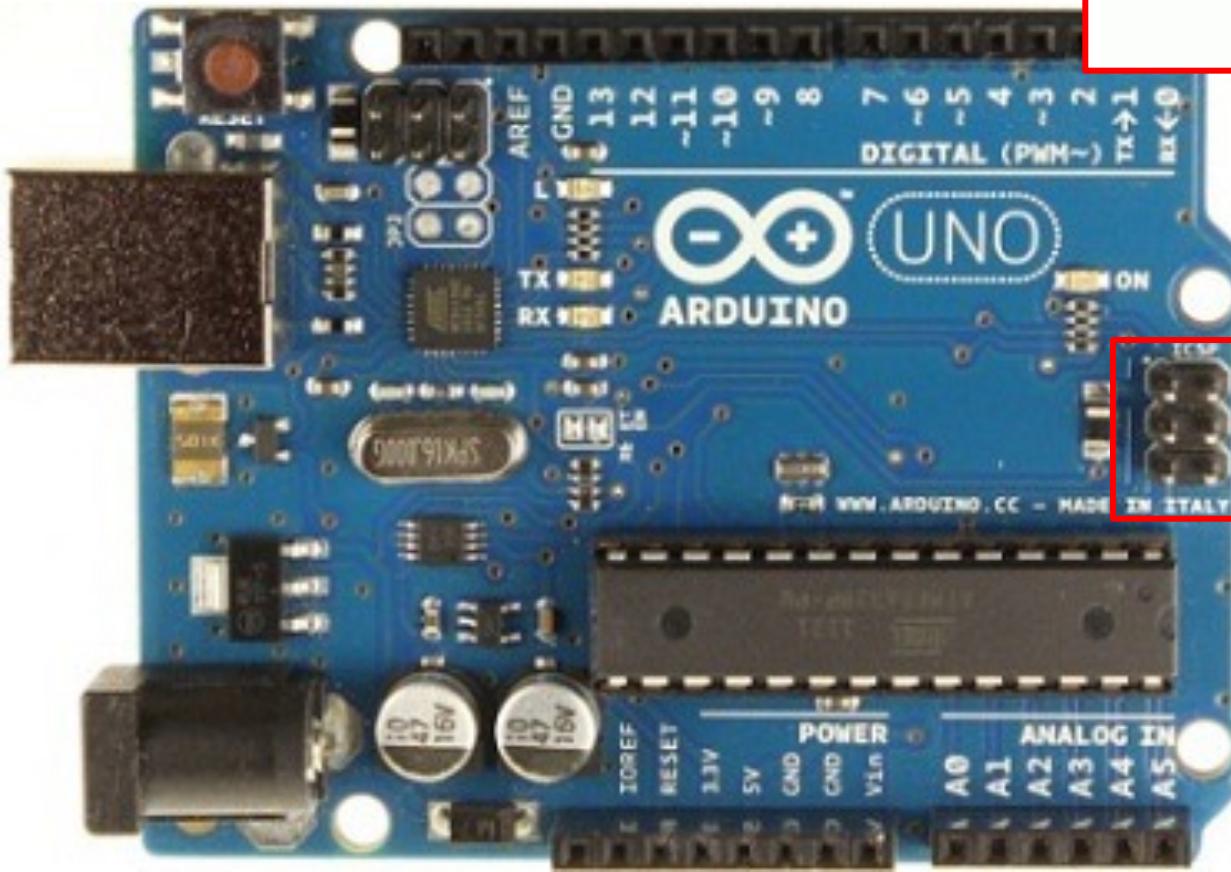
- Some Microprocessors have **build in SPI peripherals**
 - SPI is simple, so you could also write your SPI protocol
 - http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus
 - <https://www.arduino.cc/en/Reference/SPI>
 - Example Arduino Microprocessor:
 - Has SPI peripheral support and the associated library
 - SPI could be configured using library defined function:
 - **setBitOrder()**: sets the data transfer endian (Big vs Little).
 - <https://www.arduino.cc/en/Reference/SPISetBitOrder>
 - **setDataMode()**: sets whether clock is idle at high or low.
 - <https://www.arduino.cc/en/Reference/SPISetDataMode>
 - **setClockDivider()**: sets the clock frequency.
 - <https://www.arduino.cc/en/Reference/SPISetClockDivider>
 - **Transfer()**: performs chip select, data transfer and chip deselect
 - <https://www.arduino.cc/en/Reference/SPITransfe>
 - Read more: <https://www.arduino.cc/en/Reference/DueExtendedSPI>
-

SPI on Arduino (Power of Library!)

```
void setup(){
    // initialize the bus for a device on pin 4
    SPI.begin(4);
    // initialize the bus for a device on pin 10
    SPI.begin(10);
}
```

```
void setup(){
    // initialize the bus for the device on pin 4
    SPI.begin(4);
    // Set clock divider on pin 4 to 21
    SPI.setClockDivider(4, 21);
    // initialize the bus for the device on pin 10
    SPI.begin(10);
    // Set clock divider on pin 10 to 84
    SPI.setClockDivider(10, 84);
}
```

MISO, MOSI and other pins for SPI



1 - MISO 2 - +Vcc
3 - SCK 4 - MOSI
5 - Reset 6 - Gnd
ICSP

SPI on Arduino (Power of Library!)

- **SPI.transfer(pin, value)** command in the library take care of chip select, data transfer, and chip deselect.
- For example:

```
void loop(){
    byte response = SPI.transfer(4, 0xFF);
}
```

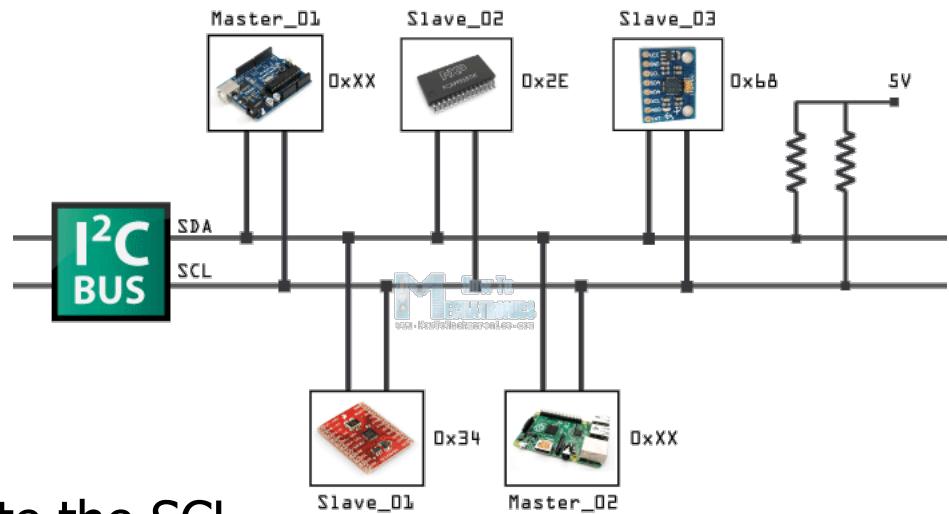
- **Select device** by setting pin 4 to LOW
- **Send 0xFF** through the SPI bus and return the byte received
- **Deselect device** by setting pin 4 to HIGH

I2C: A Synchronous Comm. Protocol

- Requires only two wires

- **SDA:** Data Signal

- **SCL:** Clock Signal



- Can have multiple Master

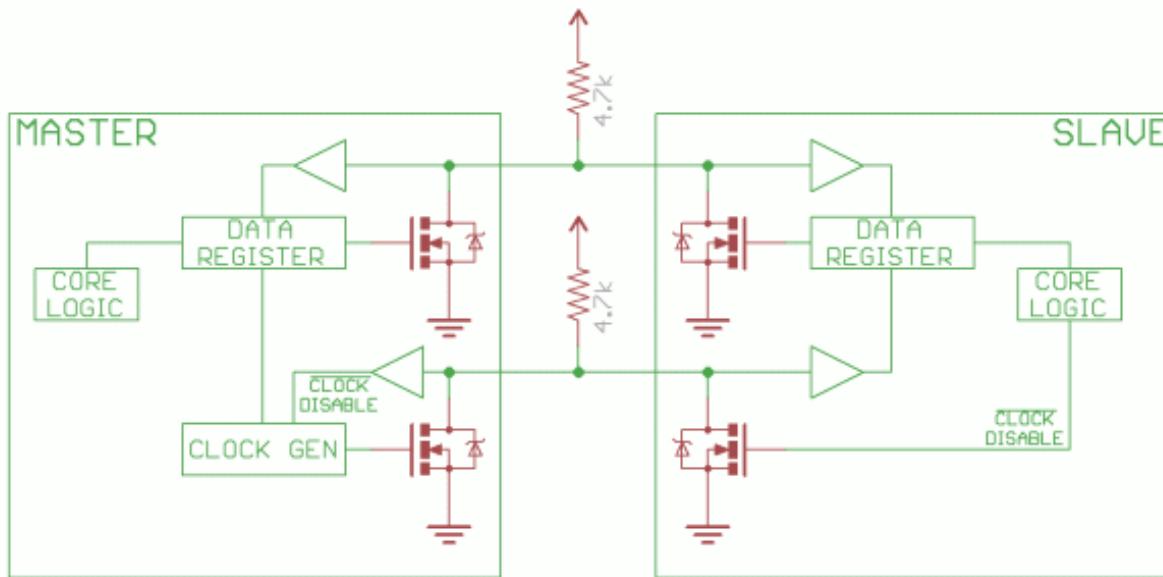
- The **current** Master generate the SCL

I²C

- Slaves can not generate the clock, but can force the clock down to prevent data transfer (Clock Stretching) to consume or generate data.
 - <http://howtomechatronics.com/tutorials/arduino/how-i2c-communication-works-and-how-to-use-it-with-arduino/>

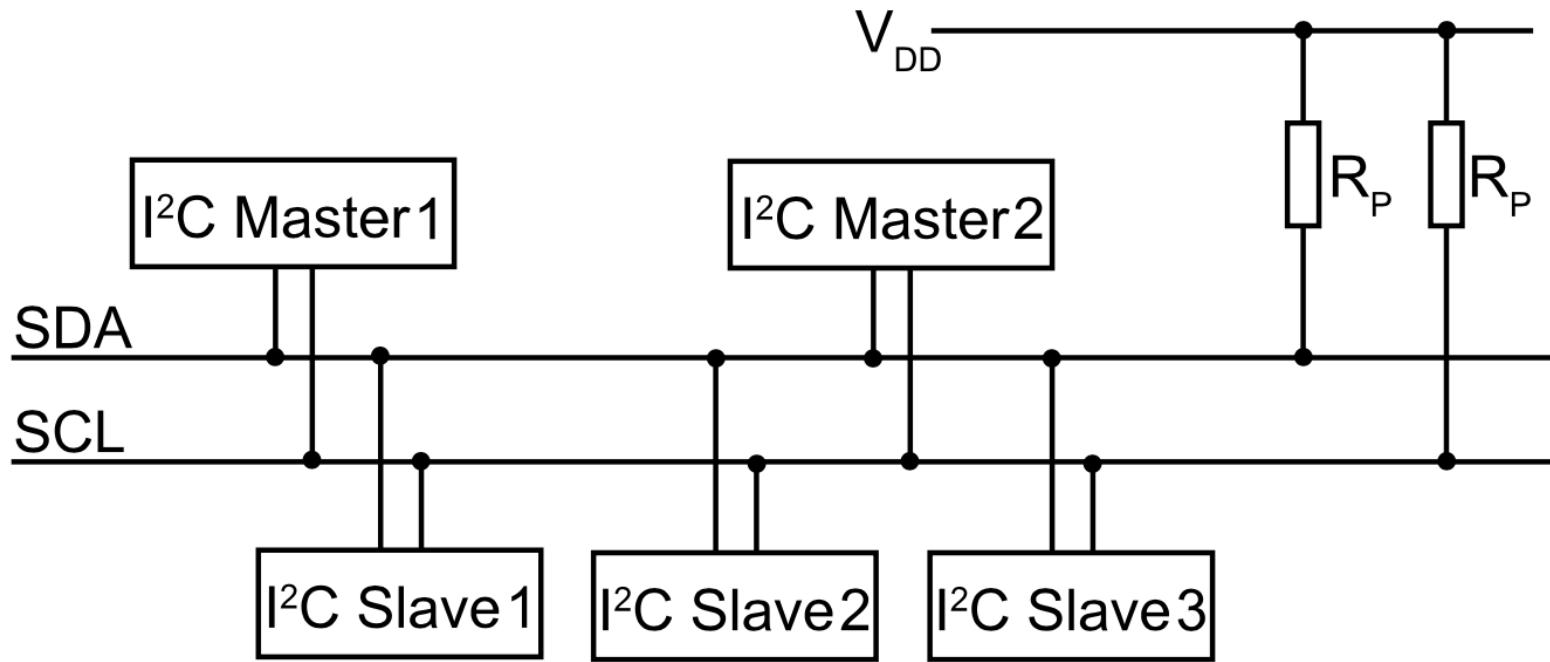
More Details on I2C

- Bus drivers are open drain: Can pull the bus down, but can't pull it up.
 - Prevents **bus contention** (two components, one trying to pull the bus down and one trying to pull up!)
- Each signal line has a pull up resistor to pull up the wire.
 - Larger resistors for short distances (2-3 meters), lower resistors for long distances (> 3 meters)



I²C: How to Connect?

- As easy as connecting the SDA and SCL pins to the existing wires.



SPI vs I2C

I2C	SPI
Speed limit varies from 100kbps, 400kbps, 1mbps, 3.4mbps depending on i2c version.	More than 1mbps, 10mbps till 100mbps can be achieved.
Half duplex synchronous protocol	Full Duplex synchronous protocol
Support Multi master configuration	Multi master configuration is not possible
Acknowledgement at each transfer	No Acknowledgement
Require Two Pins only SDA, SCL	Require separate MISO, MOSI, CLK & CS signal for each slave.
Addition of new device on the bus is easy	Addition of new device on the bus is not much easy a I2C
More Overhead (due to acknowledgement, start, stop)	Less Overhead



Internet of Things

Senior Design Project Course

Sensing - Part 2

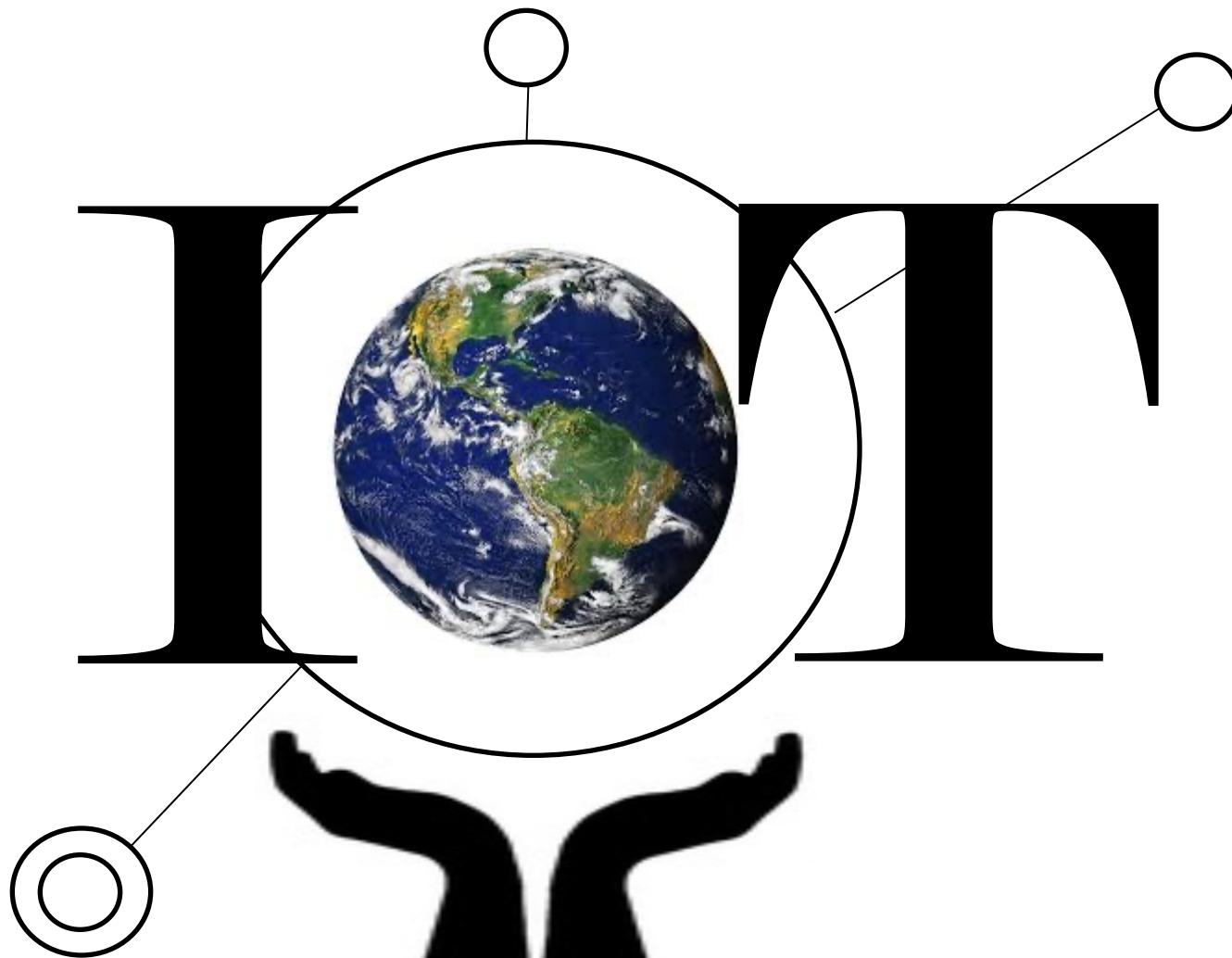
Lecturer: Avesta Sasan

University of California Davis

An Example of Senior Project (speaker diarization)

- In a room multiple people are speaking simultaneously. We want to design an IoT system for voice recognition and noise cancellation.
- Use multiple microphones around the room (4)
 - Sensing
- Capture the data and communicate the data to a mobile phone, or a computer using a wireless protocol.
 - communication
- Use machine learning algorithm to generate multiple audio files, one for each person in the room, such that other conversations are cancelled.
 - Fog Computing
 - <https://github.com/parthe/Speaker-Diarization-toolkit-MATLAB>
- Or use a cloud service to do this for you
 - <https://cloud.google.com/speech-to-text/docs/multiple-voices>
 - Cloud Computing
- Convert the speech to text ...
- And other cool things that you can do on top of this!

Lets Get Started:



Flow of Data in Internet of Things (Review)



Today's Focus



Image source: <http://www.cchc.cl/informacion-a-la-comunidad/industria-de-la-construccion/personaje/>

Sensors (Review)

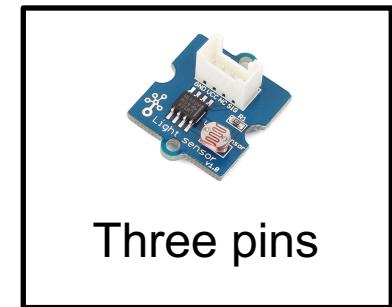
- Measure values
- Send raw data
- For IoT devices it is desired that sensors consume very Low power



Sensor Types (Review!)

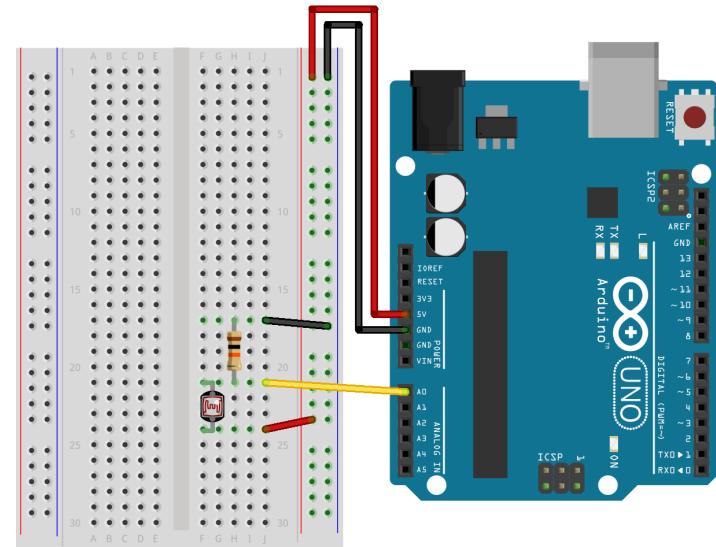
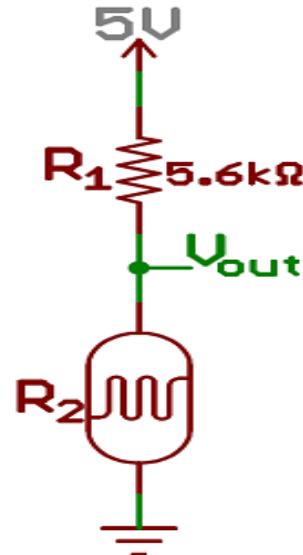
■ Analog

- 2 or 3 pins
- Use pin functions
- Following article show how analog sensor could be read by Arduino processor:
 - <https://www.arduino.cc/en/Tutorial/AnalogInput>



Measuring Analog

- In most cases we build a Voltage Divider
- We measure the voltage in V_{out}
- Rule of Thumb :
 - To minimize readout errors read many values and average them



Made with Fritzing.org

$$V_{out} = \frac{R_2}{(R_1 + R_2)} \cdot V_{supply}$$

Digital Sensors

■ Digital

- Use some **digital protocol**
- **Use libraries**
- Following article show how digital sensors could be read by Arduino processor:
 - <https://www.arduino.cc/en/Tutorial/DigitalReadSerial>

■ Examples of digital sensors:

- Digital Accelerometers
- Digital Temperature Sensor
- Digital Gyro
- Camera



Digital Temperature Sensor DS1620

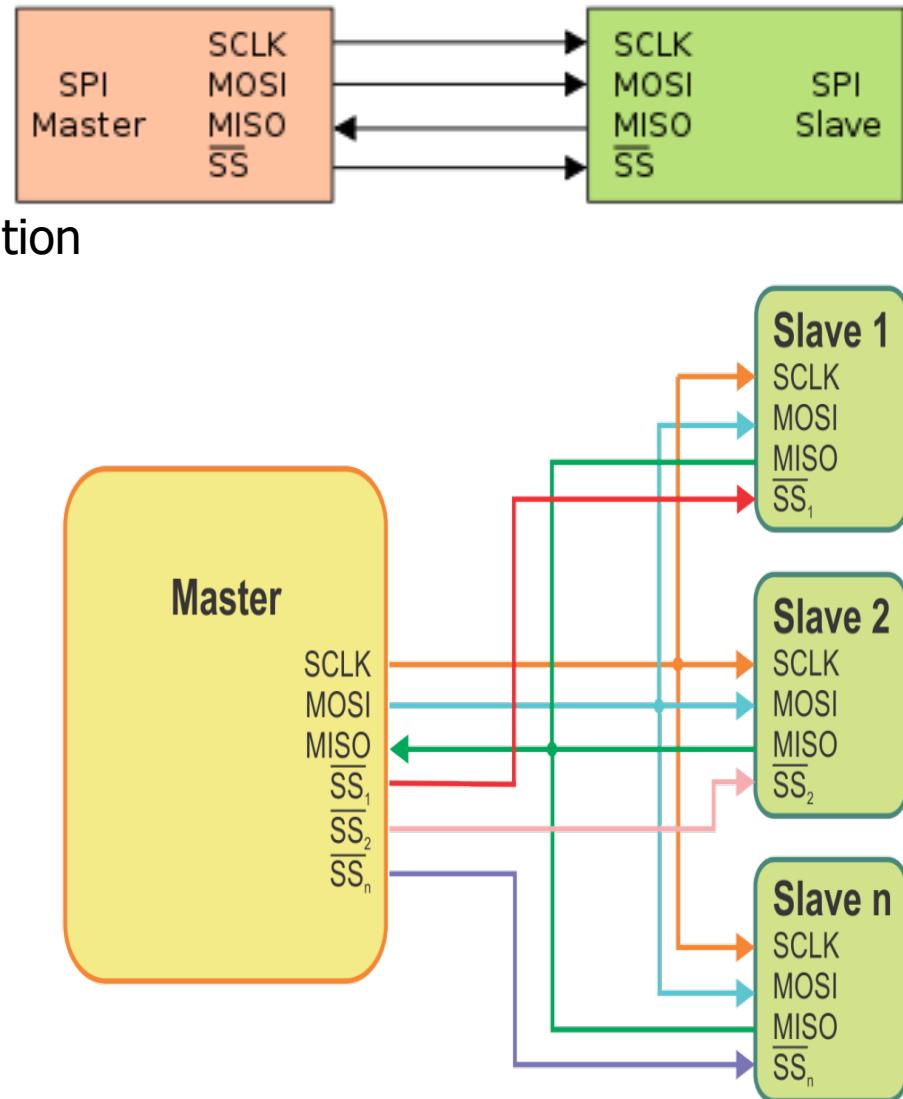
Digital Sensors

- To read digital sensors you need to use the **sensor's interface protocols**
- Most common interface protocols are:
 - Serial Peripheral Interface (**SPI**)
 - Is an interface bus commonly used to send data between microcontrollers and small peripherals such as shift registers, sensors, and SD cards. It uses separate clock and data lines, along with a select line to choose the device you wish to talk to
 - https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus 
 - Inter-Integrated Circuit (**I2C**)
 - is a multi-master, multi-slave, packet switched, single-ended, serial computer bus used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication
 - <https://en.wikipedia.org/wiki/I2C> 

SPI

We will cover the SPI protocol in more details in the next lecture!

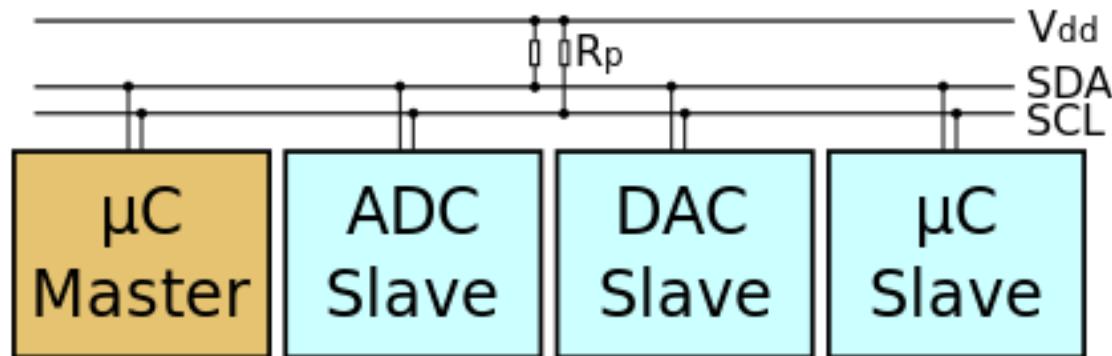
- A Master/Slave protocol
 - **One Master**
 - Several slaves
 - Master always initiates communication
- Wires
 - MOSI – Master Out Slave In
 - MISO – Master In Slave Out
 - SCLK – SPI Clock
 - SS_n – Slave Select
- **Speeds**
 - Up to 10MBs



I2C

We will cover the I2C protocol in more details in the next lecture!

- A Master/Slave protocol
 - **One or more masters**
 - Several slaves
 - Master always initiates communication
 - Each device has an address
- Wires
 - SDA – Serial Data Line
 - SCL – Serial Clock Line
- **Speeds**
 - Standard 100 Kbit
 - Up to 3.4 Mbit



Reading Digital Input (an example!):

■ **Objective:**

- Read the input from a push button (regulated by SR latch to produce digital output) and control an LED.

■ **What do we need:**

- Arduino or Genuino Board (our microprocessor)
- SR debounced push button
- built-in LED on pin 13 (already available on this board!)

Example Continued!

- Busy waiting on the digital pin and using digital write to change the LED state!

```
int ledPin = 13; // LED connected to digital pin 13
int inPin = 7;   // pushbutton connected to digital pin 7
int val = 0;     // variable to store the read value

void setup()
{
  pinMode(ledPin, OUTPUT);      // sets the digital pin 13 as output
  pinMode(inPin, INPUT);       // sets the digital pin 7 as input
}

void loop()
{
  val = digitalRead(inPin);    // read the input pin
  digitalWrite(ledPin, val);   // sets the LED to the button's value
}
```

- If you like to use more complicated digital reads involving SPI libraries, look at the following example:

- <https://www.arduino.cc/en/Reference/SPISettings>
 - https://www.youtube.com/watch?v=YE0wnom_7As



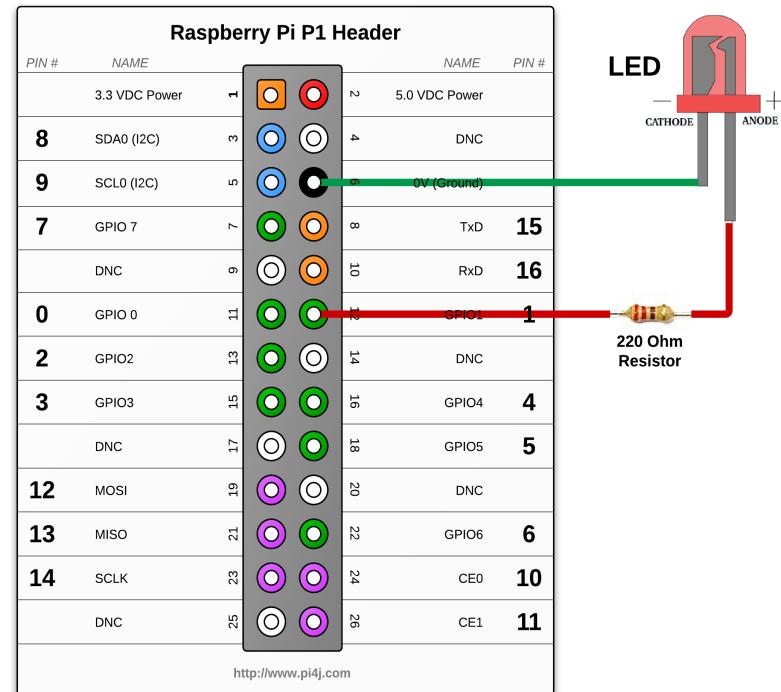
We Sometimes Need to Actuate!

- **Actuators** do something in the physical world
- Examples of actuators are:
 - Engines
 - Lights
 - Displays
 - Speakers
 - ...



LED

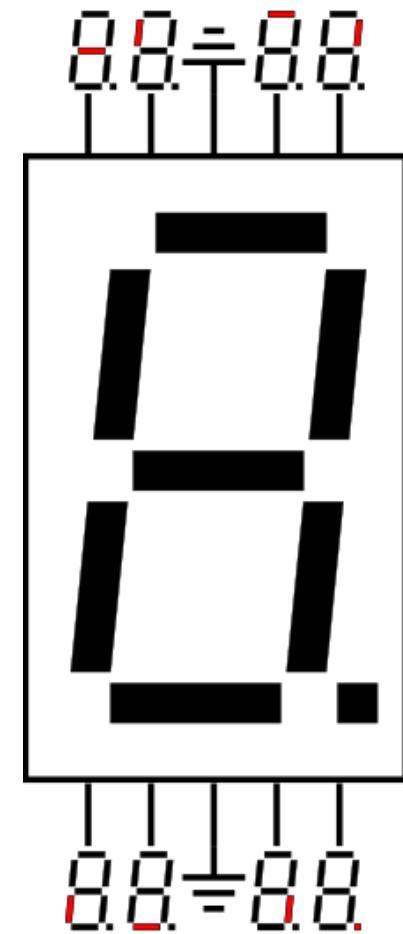
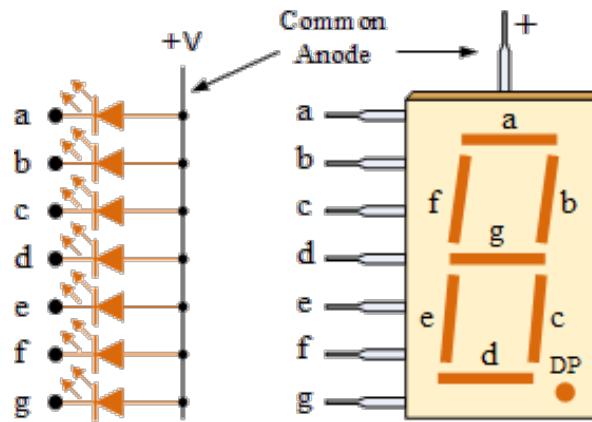
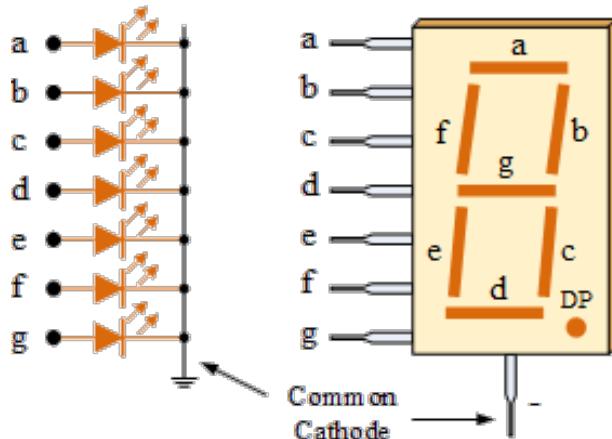
- Light Emitting Diode (**LED**):
 - Diode
 - Two legs
 - Anode (+)
 - Cathode (-)
 - Start lighting up if it has more than 0.6V
 - Infinite resistance up to 0.6 V
 - 0 resistance when it lights up



7 Segment Display

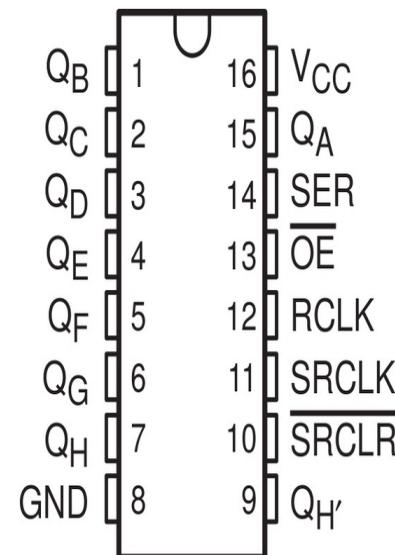
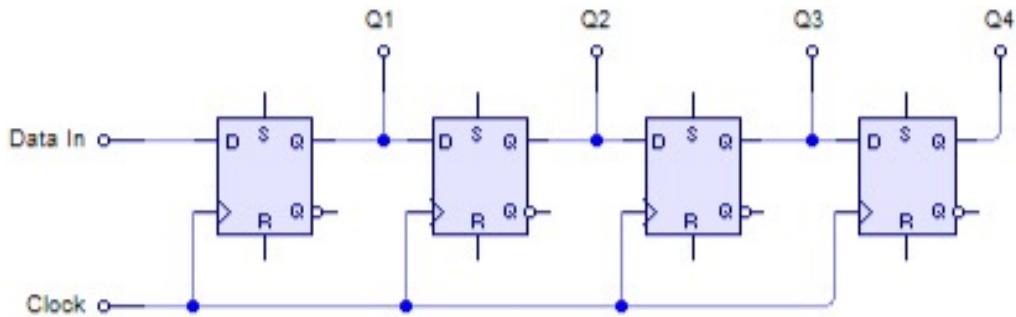
■ Seven Segment LEDs

- Common Cathode
 - Connect cathodes to 0, apply 1 to anode
- Common Anode
 - Connect anodes to 1, apply 0 to cathode
- Use multiplexing to reduce the number of pins if multiple seven segment LEDs are needed!



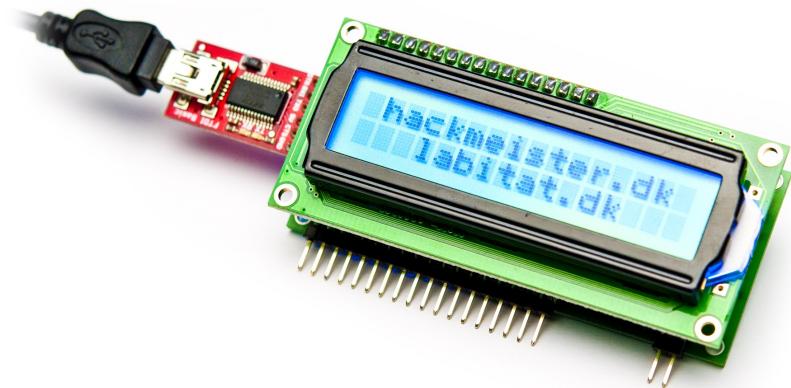
Shift Register

- Serial to Parallel Register
- Pins are limited
 - Use expanders
 - Shift register
- $Q_A \dots Q_H$ – data stored
- OE – enable (if 0)
- SEN – Serial input
- SRCLK – Serial clock
- RCLK – register clock (outputs)
- SRCLR – clear
- $Q_{H'}$ – shift output bit



LCD

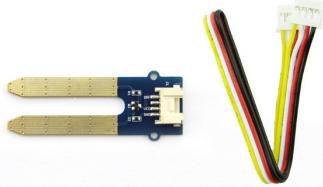
- Liquid Crystal Display (**LCD**)
 - 16 pins
 - Two data protocols
 - 4 bit
 - 8 bit
 - Microcontroller
 - I2C version (with an adapter)



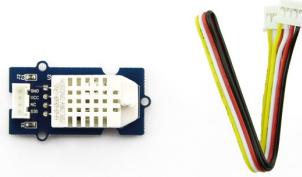
More Sensors



Barometer Sensor



Moisture Sensor



Temp & Humidity sensor



Dust Sensor



Air Quality Sensor



Gas Sensor



Sound Sensor

Buy Them Chip! Build a Cool Project!

- Very chip to buy from Amazon or other vendors to build your cool projects.

- Arduino Raspberry pi Sensor kit



- SunFounder 37 Modules Sensor Kit

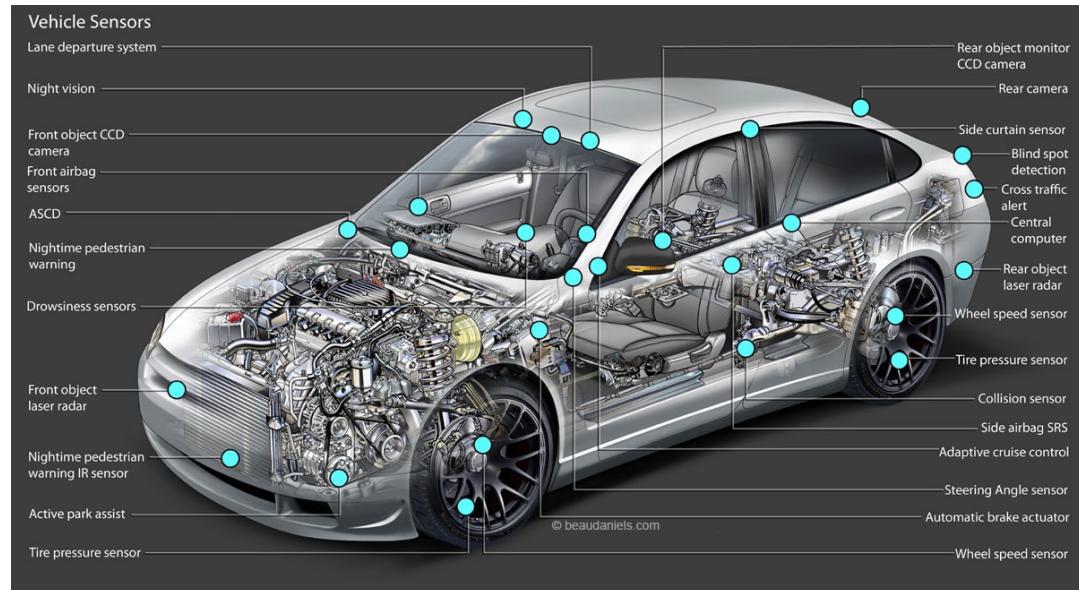


- Sensor Kit For Arduino UNO R3 MEGA NANO Raspberry Pi



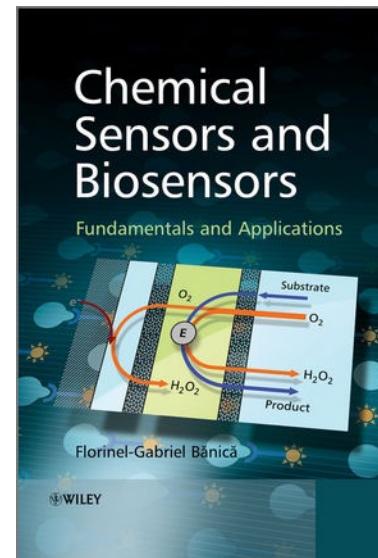
More Sensors (Automotive, Transportation)

- [Air flow meter](#)
- [Air-fuel ratio meter](#)
- [AFR sensor](#)
- [Blind spot monitor](#)
- [Crankshaft position sensor](#)
- [Curb feeler](#)
- [Defect detector](#)
- [Engine coolant temperature sensor](#)
- [Hall effect sensor](#)
- [Knock sensor](#)
- [MAP sensor](#)
- [Mass flow sensor](#)
- [Oxygen sensor](#)
- [Parking sensors](#)
- [Radar gun](#)
- [Speedometer](#)
- [Speed sensor](#)
- [Throttle position sensor](#)
- [Tire-pressure monitoring sensor](#)
- [Torque sensor](#)
- [Transmission fluid temperature sensor](#)
- [Turbine speed sensor](#)
- [Variable reluctance sensor](#)
- [Vehicle speed sensor](#)
- [Water sensor](#)
- [Wheel speed sensor](#)



Chemical Sensors

- [Breathalyzer](#)
- [Carbon dioxide sensor](#)
- [Carbon monoxide detector](#)
- [Catalytic bead sensor](#)
- [Chemical field-effect transistor](#)
- [Chemiresistor](#)
- [Electrochemical gas sensor](#)
- [Electronic nose](#)
- [Electrolyte-insulator-semiconductor sensor](#)
- [Fluorescent chloride sensors](#)
- [Holographic sensor](#)
- [Hydrocarbon dew point analyzer](#)
- [Hydrogen sensor](#)
- [Hydrogen sulfide sensor](#)
- [Infrared point sensor](#)
- [Ion-selective electrode](#)
- [Nondispersive infrared sensor](#)
- [Microwave chemistry sensor](#)
- [Nitrogen oxide sensor](#)
- [Olfactometer](#)
- [Optode](#)
- [Oxygen sensor](#)
- [Ozone monitor](#)
- [Pellistor](#)
- [pH glass electrode](#)
- [Potentiometric sensor](#)
- [Redox electrode](#)
- [Smoke detector](#)
- [Zinc oxide nanorod sensor](#)



If you like to know more:

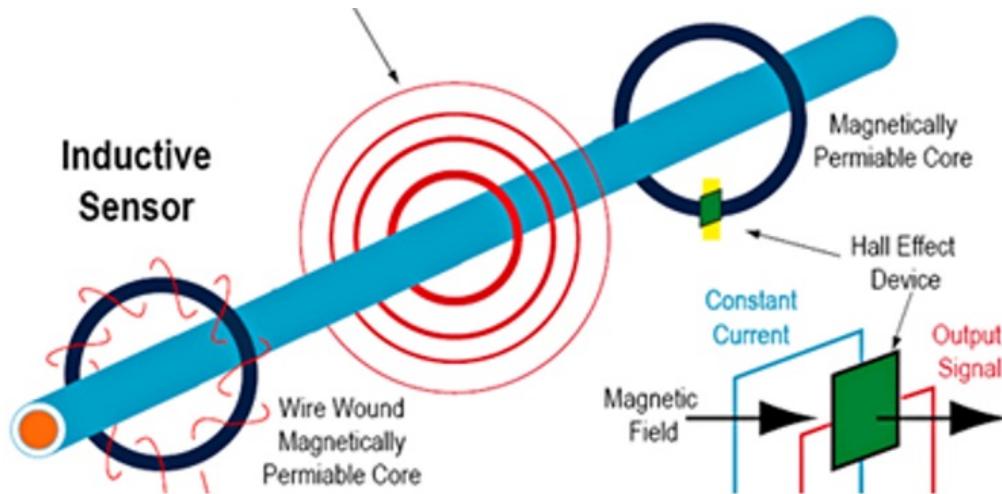
<http://www.wiley.com/WileyCDA/WileyTitle/productCd-0470710675.html>

Electric & Magnetic Sensors

- [Current sensor](#)
- [Daly detector](#)
- [Electroscope](#)
- [Electron multiplier](#)
- [Faraday cup](#)
- [Galvanometer](#)
- [Hall effect sensor](#)
- [Hall probe](#)
- [Magnetic anomaly detector](#)
- [Magnetometer](#)
- [Magnetoresistance](#)
- [MEMS magnetic field sensor](#)
- [Metal detector](#)
- [Planar Hall sensor](#)
- [Radio direction finder](#)
- [Voltage detector](#)
- etc.

If you like to know more:

- **Electric and Magnetic Sensors and Actuators**
- http://digital-library.theiet.org/content/books/10.1049/sbcs502e_ch5
- **Sensor Magazine:**
 - <http://www.sensorsmag.com/topic/electric-magnetic>



Environmental Sensors

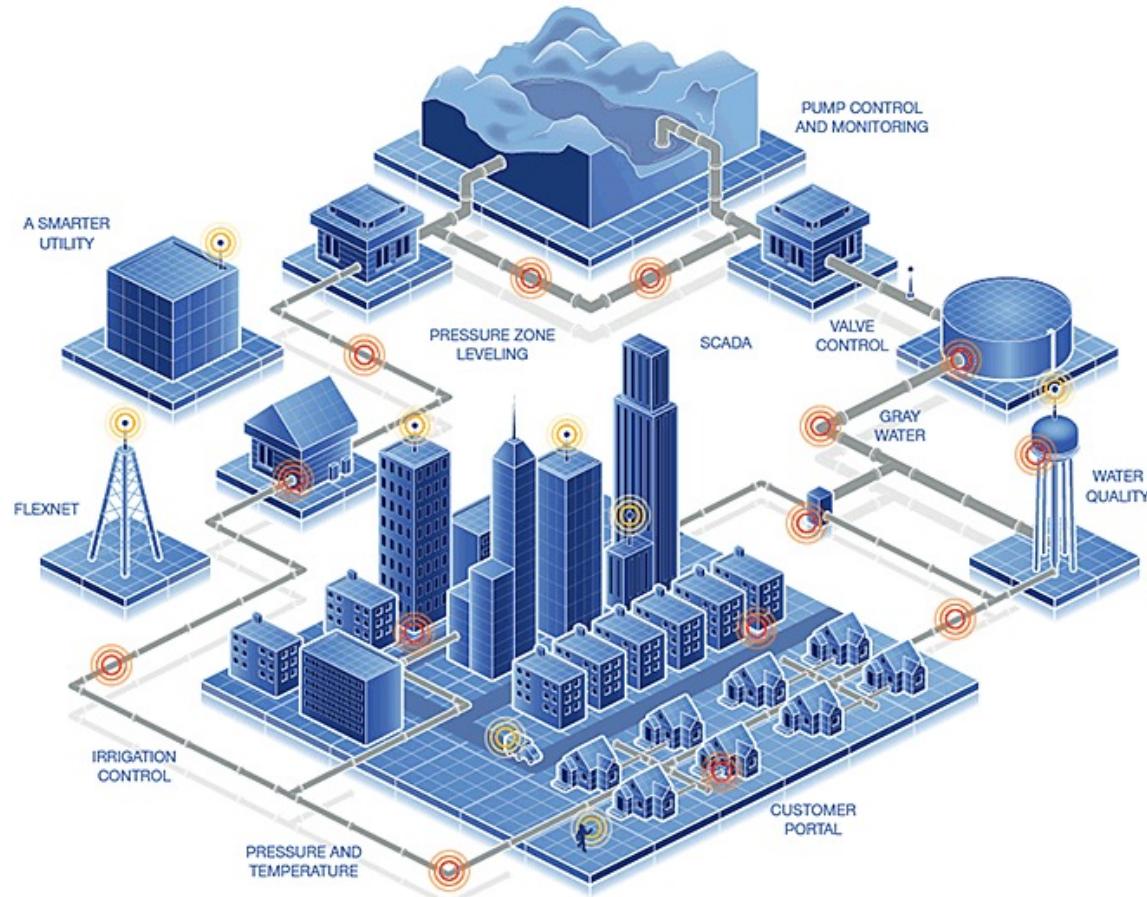
- [Actinometer](#)
- [Air pollution sensor](#)
- [Bedwetting alarm](#)
- [Ceilometer](#)
- [Dew warning](#)
- [Electrochemical gas sensor](#)
- [Fish counter](#)
- [Frequency domain sensor](#)
- [Gas detector](#)
- [Hook gauge evaporimeter](#)
- [Humistor](#)
- [Hygrometer](#)
- [Leaf sensor](#)
- [Lysimeter](#)
- [Pyranometer](#)
- [Pyrgeometer](#)
- [Psychrometer](#)
- [Rain gauge](#)
- [Rain sensor](#)
- [Seismometers](#)
- [SNOTEL](#)
- [Snow gauge](#)
- [Soil moisture sensor](#)
- [Stream gauge](#)
- [Tide gauge](#)



Interested to purchase some of this:
<http://www.trossenrobotics.com/c/arduino-environmental-sensors.aspx>

Flow and Fluid Velocity Sensors

- [Air flow meter](#)
- [Anemometer](#)
- [Flow sensor](#)
- [Gas meter](#)
- [Mass flow sensor](#)
- [Water meter](#)



Ionizing Radiation & Subatomic Particles Sensors

- [Cloud chamber](#)
- [Geiger counter](#)
- [Geiger-Muller tube](#)
- [Ionisation chamber](#)
- [Neutron detection](#)
- [Proportional counter](#)
- [Scintillation counter](#)
- [Semiconductor detector](#)
- [Thermoluminescent dosimeter](#)

Navigation Sensor

- [Air speed indicator](#)
- [Altimeter](#)
- [Attitude indicator](#)
- [Depth gauge](#)
- [Fluxgate compass](#)
- [Gyroscope](#)
- [Inertial navigation system](#)
- [Inertial reference unit](#)
- [Magnetic compass](#)
- [MHD sensor](#)
- [Ring laser gyroscope](#)
- [Turn coordinator](#)
- [TiaLinx sensor](#)
- [Variometer](#)
- [Vibrating structure gyroscope](#)
- [Yaw rate sensor](#)



Position, Angle, Displacement, ... Sensors

- [Auxanometer](#)
- [Capacitive displacement sensor](#)
- [Capacitive sensing](#)
- [Flex sensor](#)
- [Free fall sensor](#)
- [Gravimeter](#)
- [Gyroscopic sensor](#)
- [Impact sensor](#)
- [Inclinometer](#)
- [Integrated circuit piezoelectric sensor](#)
- [Laser rangefinder](#)
- [Laser surface velocimeter](#)
- [LIDAR](#)
- [Linear encoder](#)
- [Linear variable differential transformer \(LVDT\)](#)
- [Liquid capacitive inclinometers](#)
- [Odometer](#)
- [Photoelectric sensor](#)
- [Piezocapacitive sensor](#)
- [Piezoelectric accelerometer](#)
- [Position sensor](#)
- [Position sensitive device](#)
- [Rate sensor](#)
- [Rotary encoder](#)
- [Rotary variable differential transformer](#)
- [Selsyn](#)
- [Shock detector](#)
- [Shock data logger](#)
- [Stretch sensor](#)
- [Tilt sensor](#)
- [Tachometer](#)
- [Ultrasonic thickness gauge](#)
- [Variable reluctance sensor](#)
- [Velocity receiver](#)

Optical, Light, Imaging & Photon Sensors

- [Charge-coupled device](#)
- [CMOS sensor](#)
- [Colorimeter](#)
- [Contact image sensor](#)
- [Electro-optical sensor](#)
- [Flame detector](#)
- [Infra-red sensor](#)
- [Kinetic inductance detector](#)
- [LED as light sensor](#)
- [Light-addressable potentiometric sensor](#)
- [Nichols radiometer](#)
- [Fiber optic sensors](#)
- [Optical position sensor](#)
- [Thermopile laser sensors](#)
- [Photodetector](#)
- [Photodiode](#)
- [Photomultiplier tubes](#)
- [Phototransistor](#)
- [Photoelectric sensor](#)
- [Photoionization detector](#)
- [Photomultiplier](#)
- [Photoresistor](#)
- [Photoswitch](#)
- [Phototube](#)
- [Scintillometer](#)
- [Shack-Hartmann](#)
- [Single-photon avalanche diode](#)
- [Superconducting nanowire single-photon detector](#)
- [Transition edge sensor](#)
- [Visible light photon counter](#)
- [Wavefront sensor](#)



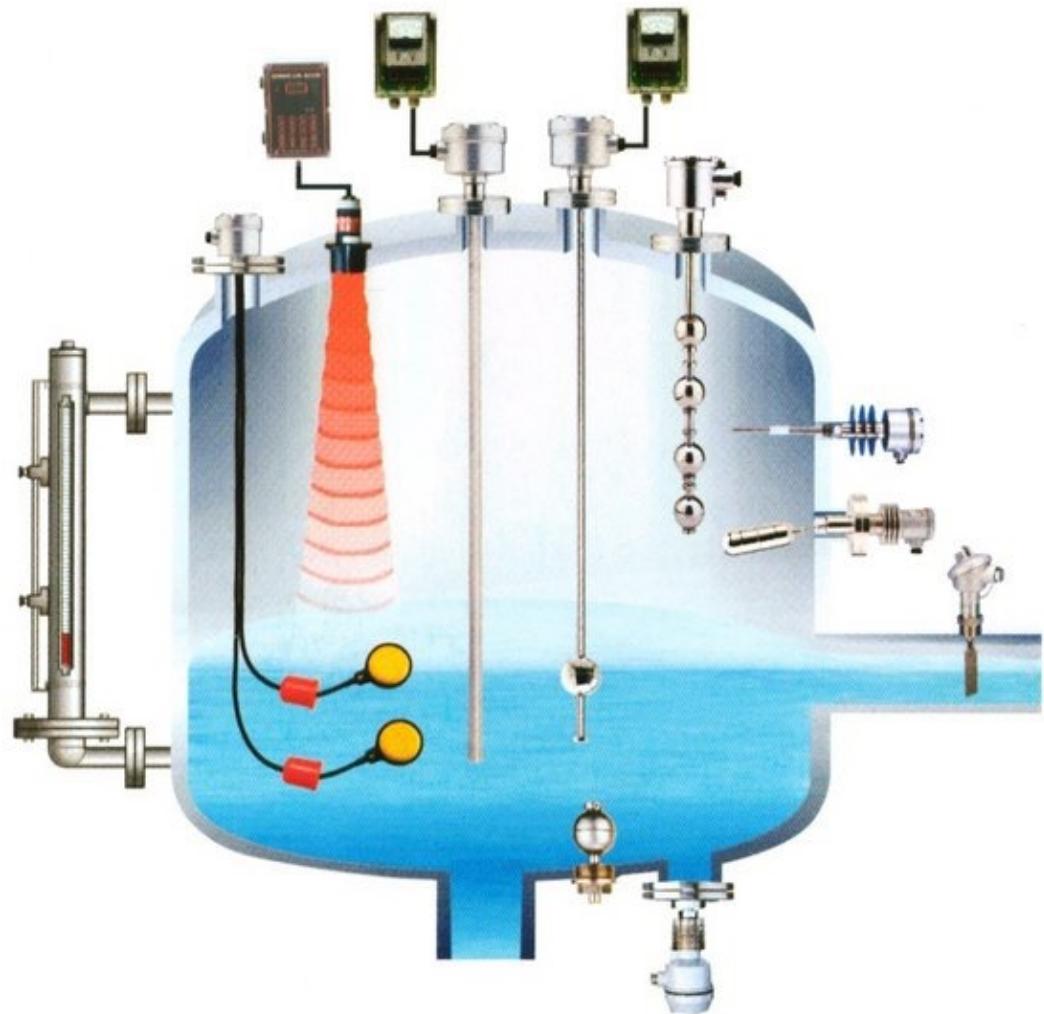
Pressure Sensor

- [Barograph](#)
- [Barometer](#)
- [Boost gauge](#)
- [Bourdon gauge](#)
- [Hot filament ionization gauge](#)
- [Ionization gauge](#)
- [McLeod gauge](#)
- [Oscillating U-tube](#)
- [Permanent Downhole Gauge](#)
- [Piezometer](#)
- [Pirani gauge](#)
- [Pressure sensor](#)
- [Pressure gauge](#)
- [Tactile sensor](#)
- [Time pressure gauge](#)



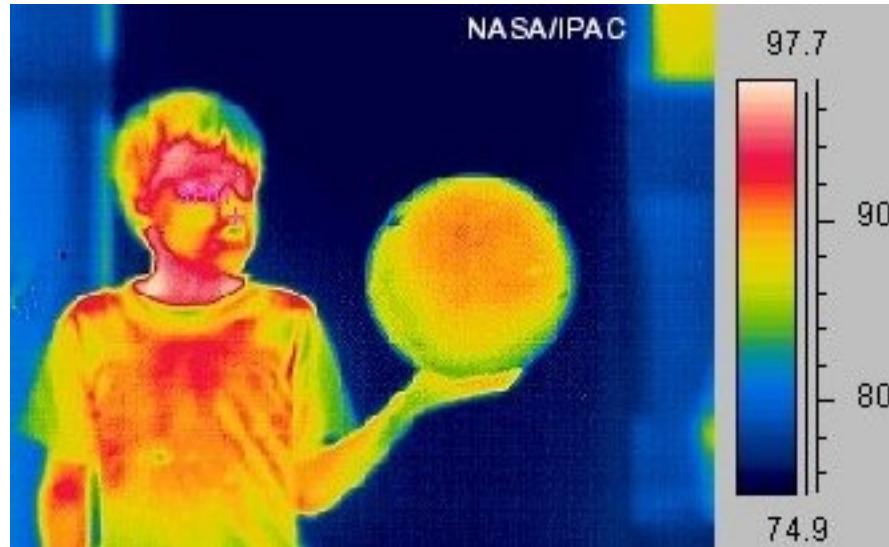
Force, Density & Level Sensors

- [Bhangmeter](#)
- [Hydrometer](#)
- [Force gauge and Force Sensor](#)
- [Level sensor](#)
- [Load cell](#)
- [Magnetic level gauge](#)
- [Nuclear density gauge](#)
- [Piezocapacitive pressure sensor](#)
- [Piezoelectric sensor](#)
- [Strain gauge](#)
- [Torque sensor](#)
- [Viscometer](#)



Thermal, Heat & Temperature Sensor

- [Bolometer](#)
- [Bimetallic strip](#)
- [Calorimeter](#)
- [Exhaust gas temperature gauge](#)
- [Flame detection](#)
- [Gardon gauge](#)
- [Golay cell](#)
- [Heat flux sensor](#)
- [Infrared thermometer](#)
- [Microbolometer](#)
- [Microwave radiometer](#)
- [Net radiometer](#)
- [Quartz thermometer](#)
- [Resistance thermometer](#)
- [Silicon bandgap temperature sensor](#)
- [Special sensor microwave/imager](#)
- [Temperature gauge](#)
- [Thermistor](#)
- [Thermocouple](#)
- [Thermometer](#)
- [Pyrometer](#)

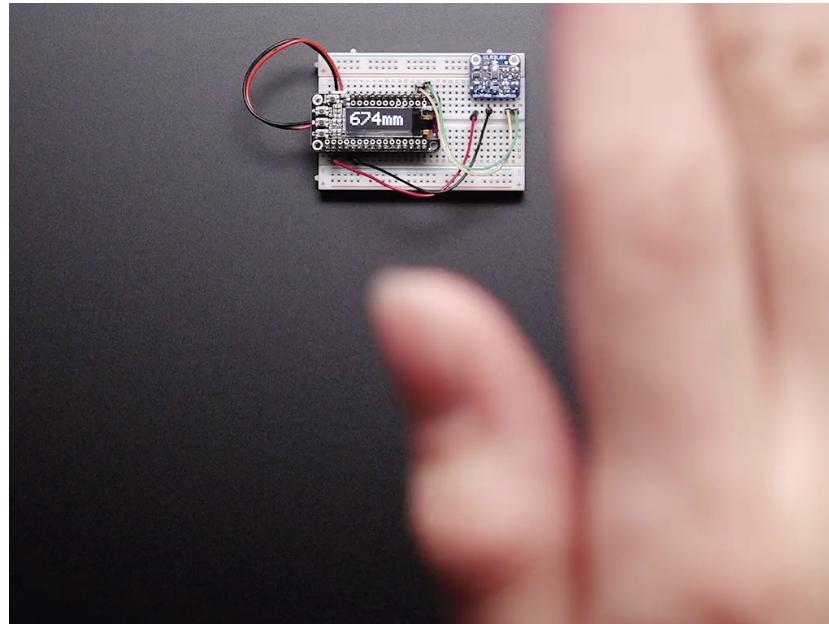


Proximity & Presence Sensor

- [Alarm sensor](#)
- [Doppler radar](#)
- [Motion detector](#)
- [Occupancy sensor](#)
- [Proximity sensor](#)
- [Passive infrared sensor](#)
- [Reed switch](#)
- [Stud finder](#)
- [Triangulation sensor](#)
- [Touch switch](#)
- [Wired glove](#)

Video Source:

https://www.adafruit.com/product/3317?gclid=EAIaIQobChMIs-iJ1OGM1gIVBKxpCh1G7AZLEAQYAiABEqL9DPD_BwE



More Sensors . . .

List goes on!

But wait!

Can we do better?

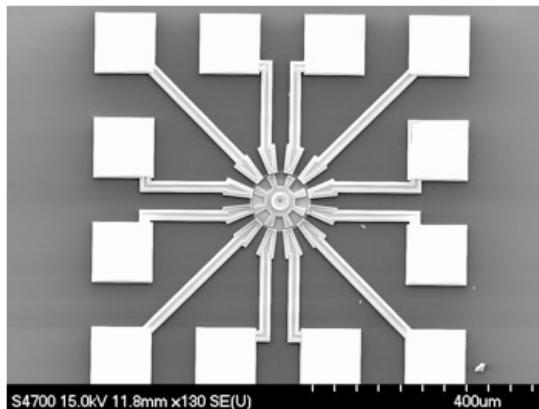
- Lower power?
- Lower area?
- Lower latency?
- Higher reliability?
- Higher performance?

MEM Sensors

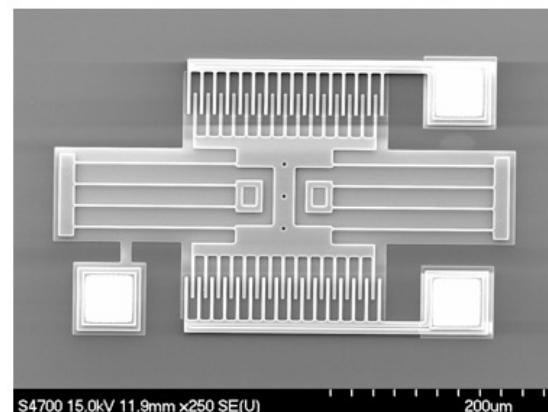
- **Micro Electro Mechanical or MEM Sensors:**
 - Miniaturized mechanical and electro-mechanical elements
 - Is the technology of microscopic devices, particularly those with moving parts
 - There are at least some elements having some sort of mechanical functionality
- It merges at the nano-scale into nanoelectromechanical systems (NEMS)
- Components between 1 and 100 Micrometers in size
- MEM sensors could be integrated within CMOS, making it possible to have on die sensors
 - Extremely small
 - Extremely sensitive
 - Extremely low power

Microelectromechanical (MEM) Sensors

- Use same fabrication process as CMOS
 - Very accurate
 - Very small
 - Very sensitive
 - Very inexpensive
- heterogeneous integration (CMOS + MEM + NEM) create many new and exciting opportunities

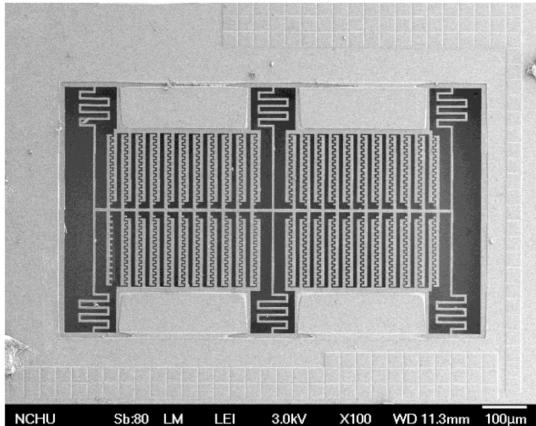


electro-statically-actuated micro-motor

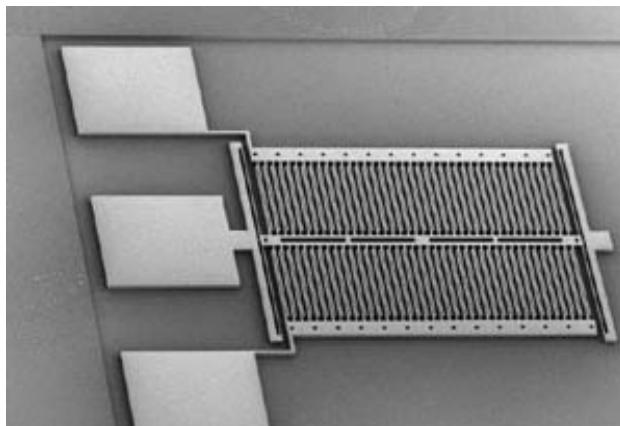


micromachined resonator

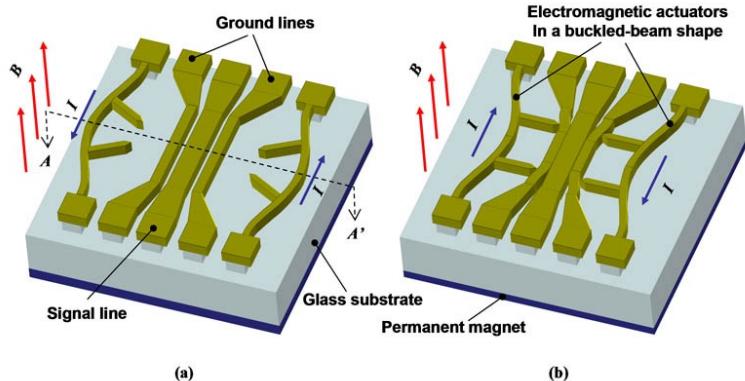
MEM Device Examples



magnetic sensor



accelerometer

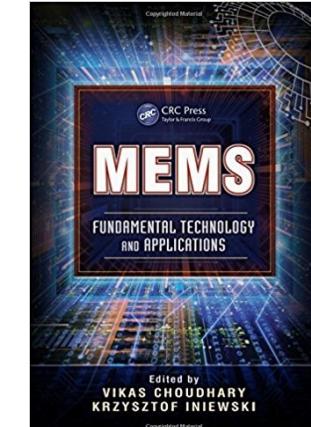
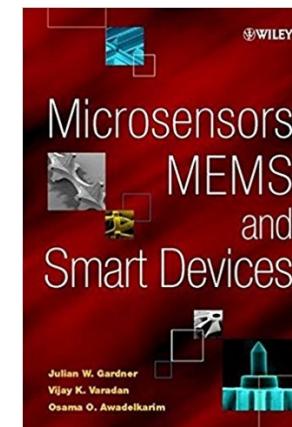
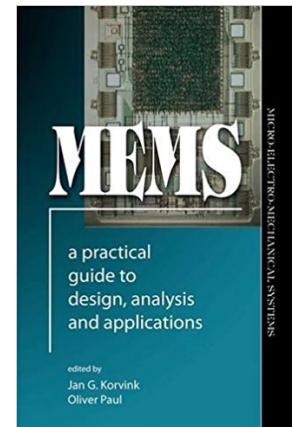
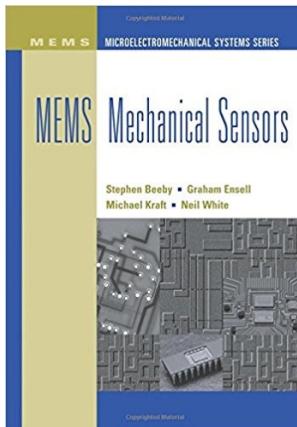
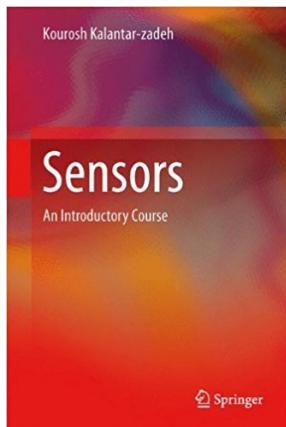


A quick read for interested students:
<https://www.memsnet.org/mems/fabrication.html>

MEMS switch

How MEM Sensors Work?

- Lets watch the following video:
- How MEMS Accelerometer Gyroscope Magnetometer Work:
 - <https://www.youtube.com/watch?v=eqZgxR6eRjo>
- How MEMs are made:
 - <https://www.youtube.com/watch?v=EALXTht-stg>



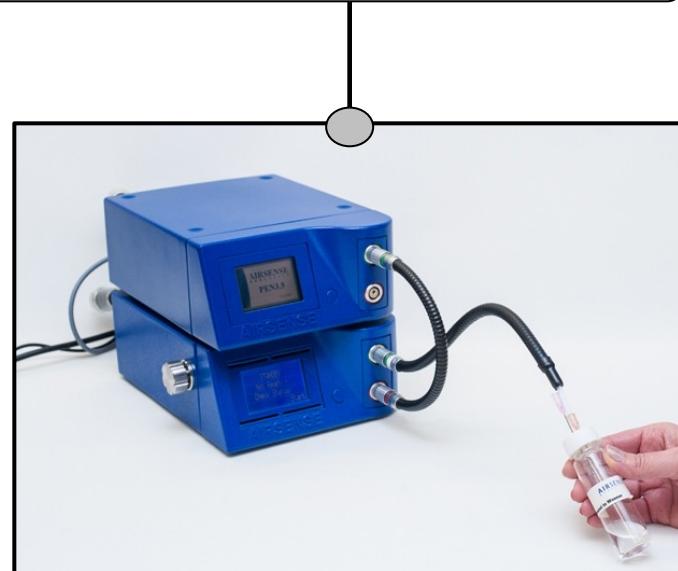
Emerging Sensors

- Many new sensor technologies are becoming available.
- May be quite expensive now, but in a few years, you can use them at very low cost for building new and exciting IoT solutions

Example 1: Electronic Nose

A sensing device intended to detect odors or flavors

- Technology exist today, however, miniaturization and cost reduction will make it available for use in IoT devices.
- **Applications:**
 - Quality Control
 - Health (detect harmful bacteria), lung cancer, viral infection,
 - Improving sense of smell through implants
 - Brain cancer cells
 - Security and Crime Prevention
 - Replacing police dogs in airports for bomb or drug detection
 - Environmental Monitoring



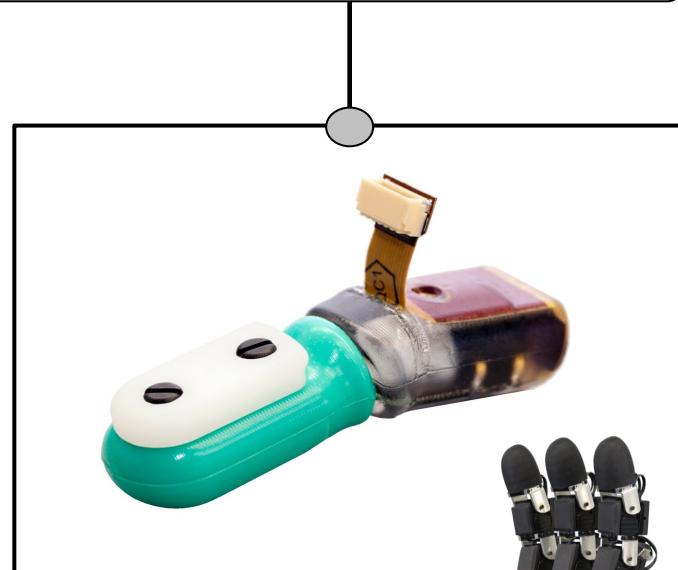
Emerging Sensors

- Many new sensor technologies are becoming available.
- May be quite expensive now, but in a few years, you can use them at very low cost for building new and exciting IoT solutions

Example 2: Tactile Sensors

Measures information arising from physical interaction with environment (mimic human biological cutaneous touch)

- Technology exist today, however, miniaturization and cost reduction will make it available for use in IoT devices.
- **Applications:**
 - Robotic
 - Computer hardware
 - Security



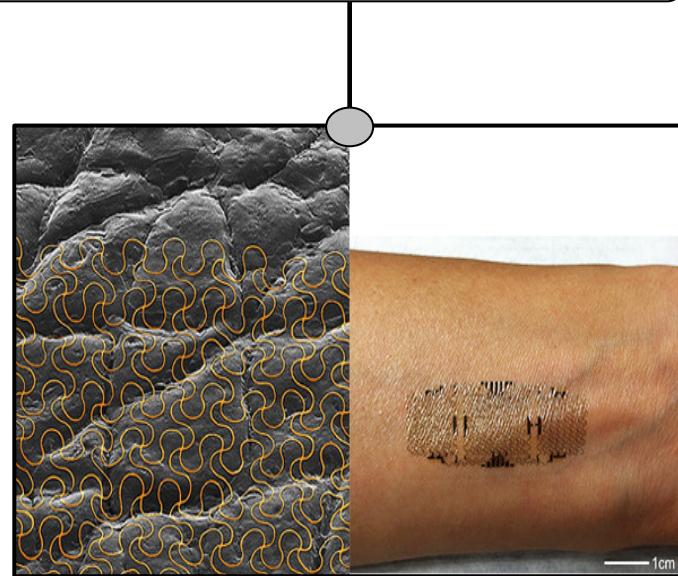
Emerging Sensors

- Many new sensor technologies are becoming available.
- May be quite expensive now, but in a few years, you can use them at very low cost for building new and exciting IoT solutions

Example 3: Printed Sensors

Sensors printed on flexible substrate such as human skin

- Technology exist today, however, miniaturization and cost reduction will make it available for use in IoT devices.
- **Applications:**
 - Human-machine interface
 - Environmental sensing





Internet of Things

Senior Design Project Course

Sensing - Part 1

Lecturer: Avesta Sasan

University of California Davis

Flow of Data in Internet of Things (Review)



Image source: <http://www.cchc.cl/informacion-a-la-comunidad/industria-de-la-construccion/personaje/>

Where to Compute? (Review!)

■ **Device centric:**

- microcontroller in an IoT device can be exploited to perform the computation.
- challenges are
 - scarce resources on IoT devices
 - runtime decision

■ **Gateway centric**

- gate- way devices which are used to settle the heterogeneity between different networks and Internet usually have more computational power
 - This scheme has been used for medical and healthcare monitoring application
 - challenge
 - guarantee the availability and deadline constraints

Where to Compute? (Review!)

■ **Fog centric**

- Fogs provide more computational power compared to IoT embedded and gateway devices and have less latency compared to the cloud servers

■ **Cloud centric**

- massive data storage volume, huge processing resources
- challenges
 - Size of data (big data)
 - scalability
 - high energy cost
 - latency
 - bandwidth
 - availability

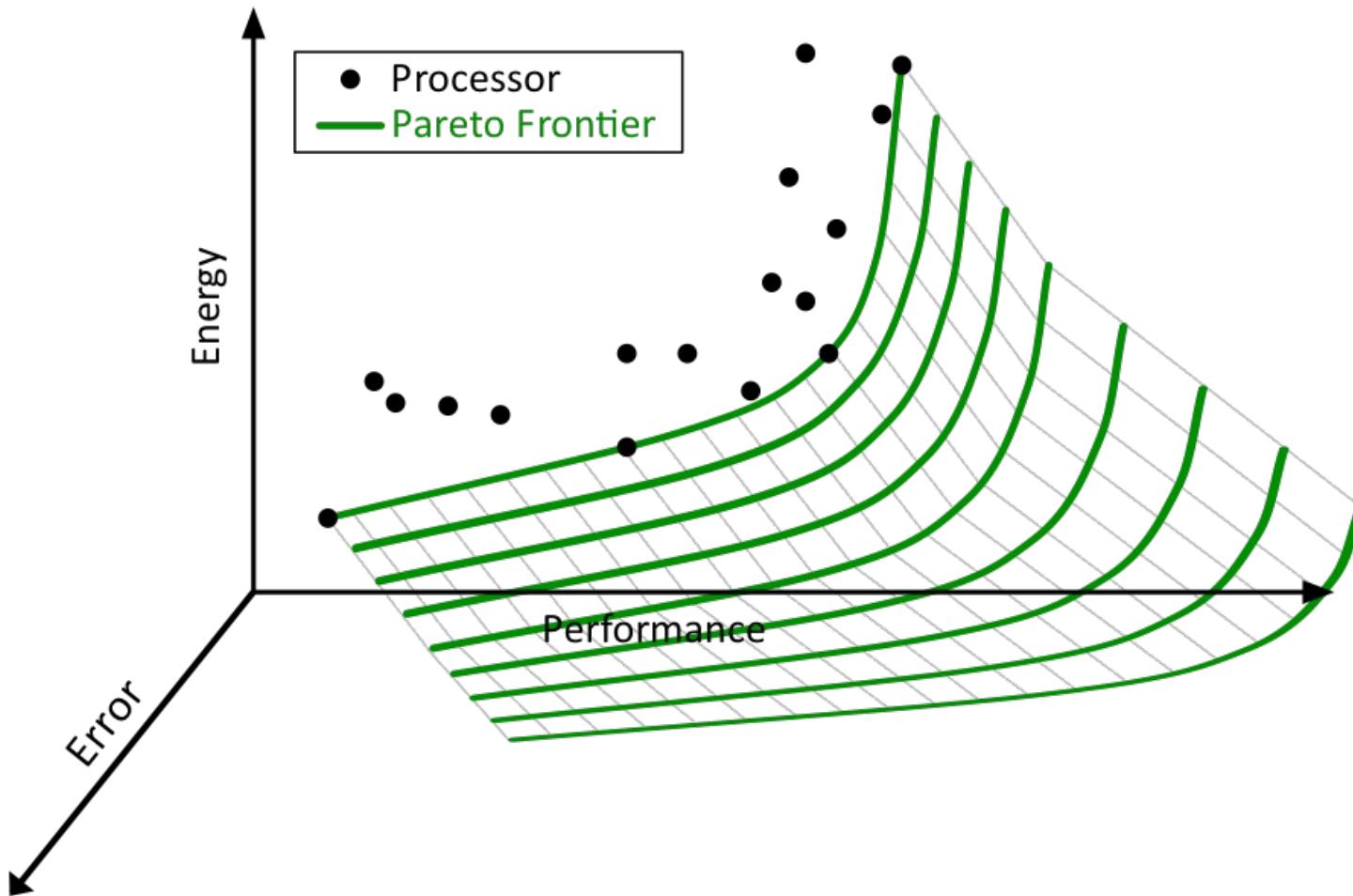
Approximate Computing

- 100% accuracy is not always required!
 - Example: Multimedia
 - 8 bits represent 256 shades of gray
 - Human can only recognize 27 shades of gray
- **Definition:** a computation which returns a possibly inaccurate result rather than a guaranteed accurate result, for a situation where an approximate result is sufficient for a purpose.
- leverages inherent resilience of applications
- relaxes the requirement of exact equivalence between the specification and implementation [1]
- Advantages: Power, Performance, Area (PPA)
- We will cover this in more details!



[1] F. Samie, L. Bauer and J. Henkel, "IoT technologies for embedded computing: A survey," *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Pittsburgh, PA, 2016, pp. 1-10.

Approximate computing design space



Approximate computing in IoT

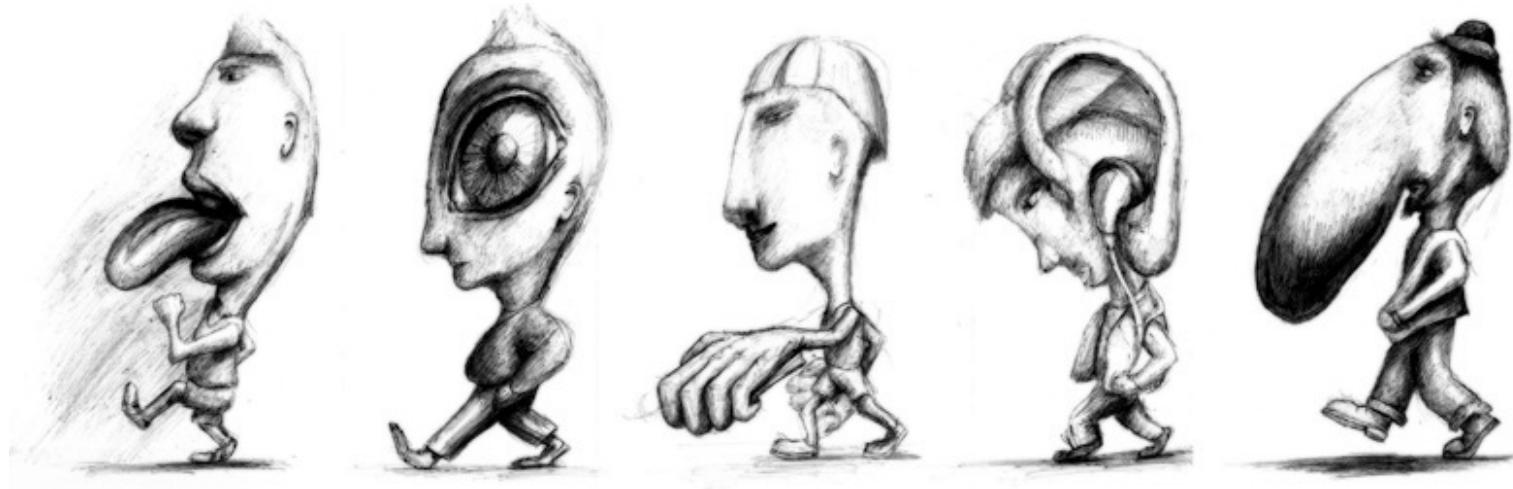
- **Data acquisition**
 - reducing the quality of input
 - to reduce the energy consumption or delay
- **Data processing**
 - designing inexact hardware units
 - Approximate adders, multipliers, DCT, FFT
 - designing inexact software
 - Loop skipping, proliferation, stage skipping
- **Data Storage:**
 - Use tolerable error to
 - reduce the size of memory
 - reduce number of memory accesses
 - reduce energy consumption
 - Improve performance

Approximate Computing to improve the PPA is a Good Topic for your Research project!

[1] F. Samie, L. Bauer and J. Henkel, "IoT technologies for embedded computing: A survey," *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Pittsburgh, PA, 2016, pp. 1-10.



Sensing



A handwritten signature in black ink, likely belonging to the artist or designer of the illustration.

Sensors

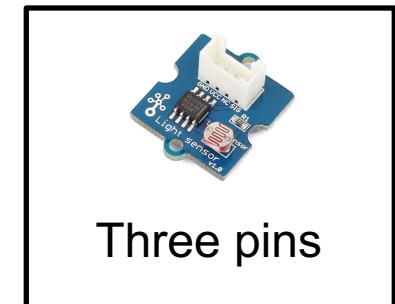
- Measure values
- Send raw data
- For IoT devices it is desired that sensors consume very Low power



Sensor Types

■ Analog

- 2 or 3 pins
- Use pin functions
- Following article show how analog sensor could be read by Arduino processor:
 - <https://www.arduino.cc/en/Tutorial/AnalogInput>

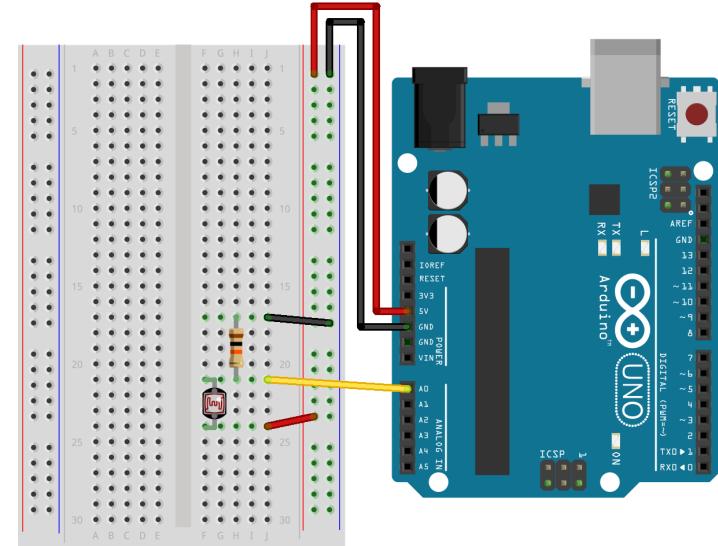
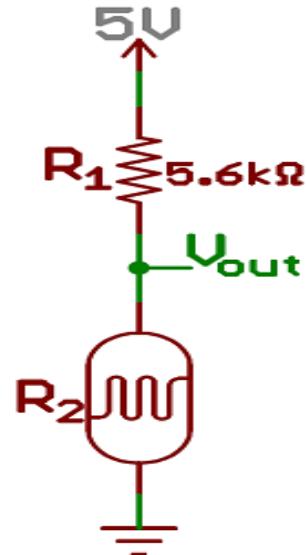


■ Digital

- Use some digital protocol
- Use libraries
- Following article show how digital sensors could be read by Arduino processor:
 - <https://www.arduino.cc/en/Tutorial/DigitalReadSerial>

Measuring Analog

- In most cases we build a Voltage Divider
- We measure the voltage in V_{out}
- Rule of Thumb :
 - To minimize readout errors read many values and average them



Made with  Fritzing.org

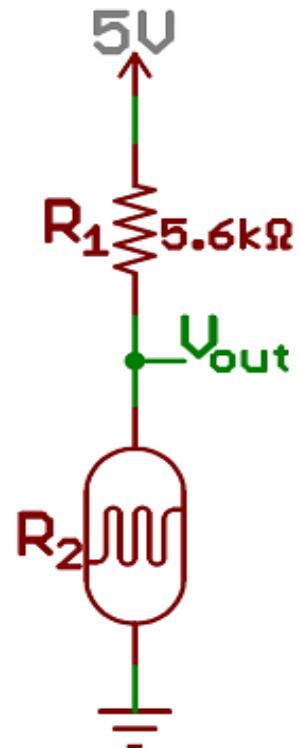
$$V_{out} = \frac{R_2}{(R_1 + R_2)} \cdot V_{supply}$$

Measuring Analog (photoresistor example)

- A resistor divider to allow the **high impedance Analog input** to measure the voltage.

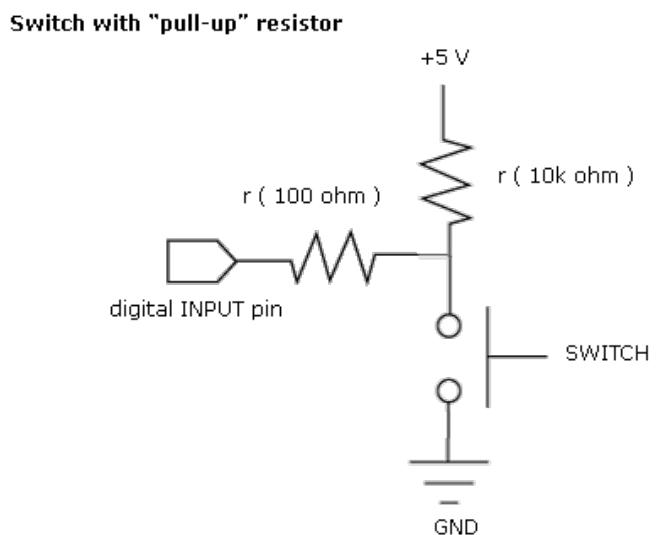
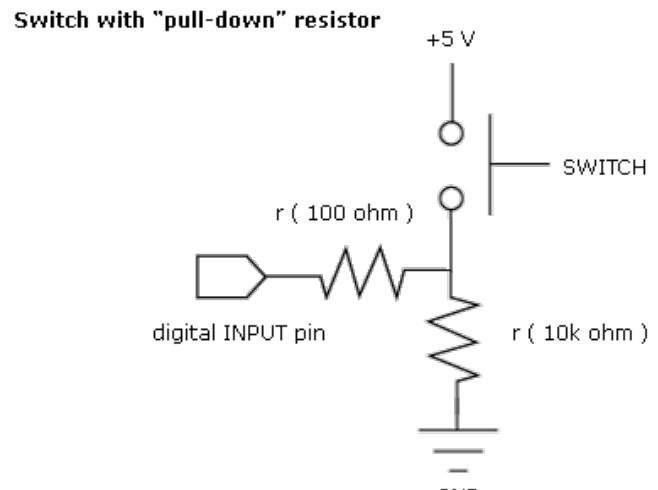
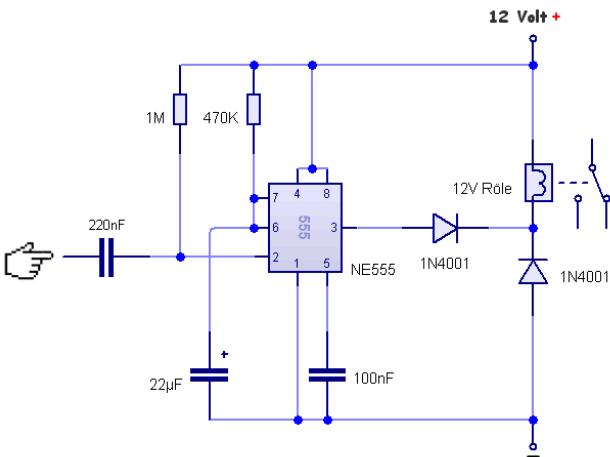
Why high impedance?

- V_{out} do not draw any current
 - By Ohm's law the voltage measured on the other end of a resistor connected to 5V is always 5V, regardless the resistor's value.
- To get a voltage proportional to the photoresistor value, a voltage divider (resistor divider) is necessary.
- This circuit uses a variable resistor, a fixed resistor and the measurement point is in the middle of the resistors.



Button is a Sensor!

- Button is an analog sensor!
- We measure **change in resistance**:
 - Infinite → if released
 - 0 → if pressed



Button Debounce

When you push the switch, it initially makes contact with the other metal part, but just in a brief split of a microsecond. Then it makes contact a little longer, and then again a little longer. In the end the switch is fully closed. **The switch is bouncing between in-contact, and not in-contact.**

- **Problem:** When the button is pressed
 - Signal is bouncing, Fast reading results in 0s and 1s
- **Solutions:**
 - Read more values and average them (analogue solution)
 - Use a debouncer (RC based or SR based) → **converting it to a digital input!**

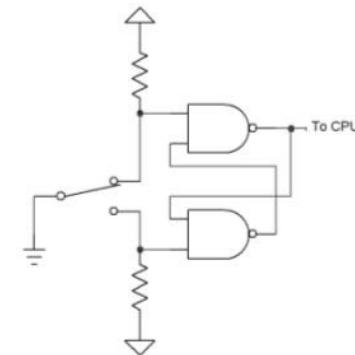
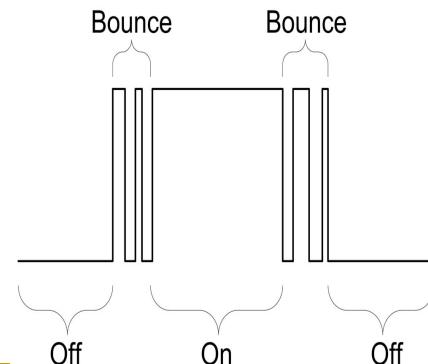
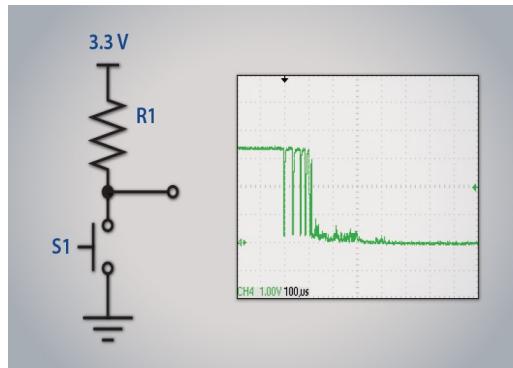


Figure 1: The SR debouncer

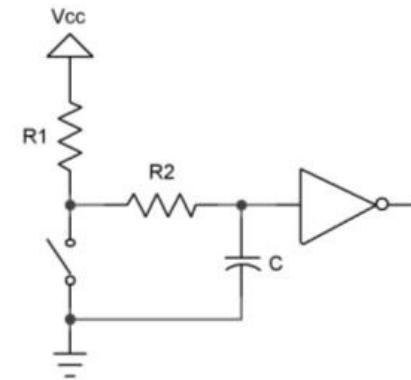
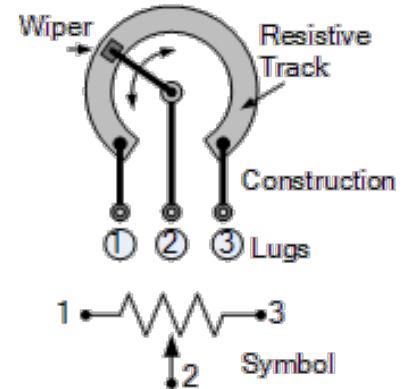
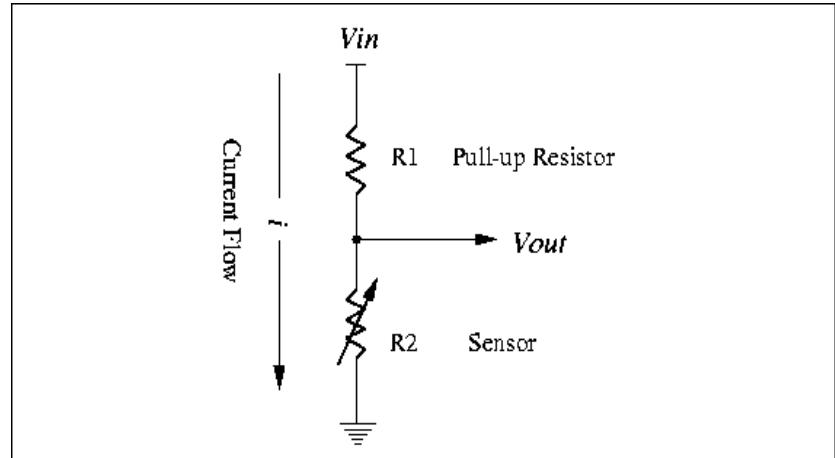
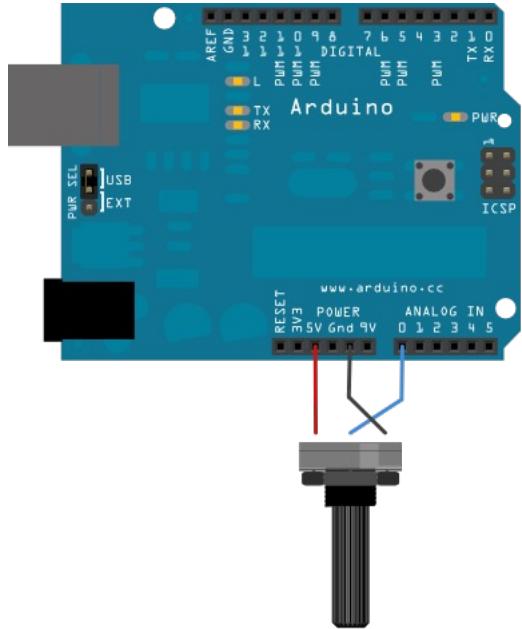


Figure 2: An RC debouncer

Potentiometer

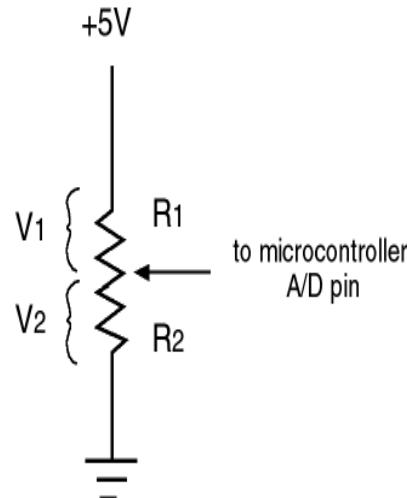
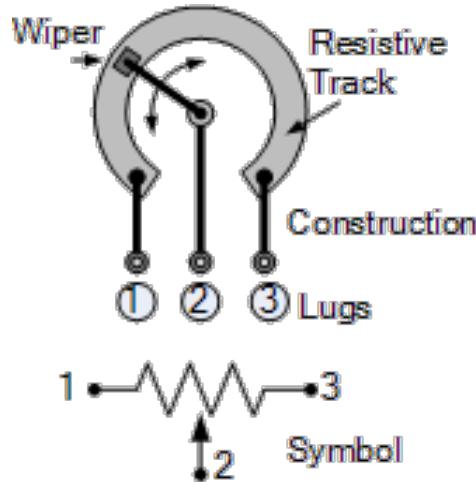
A **potentiometer**, informally a **pot**, is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider.



Is this an analogue or a digital sensor?

Potentiometer is a Voltage Divider

- Variable resistance
- Connect the three pins
 - One pin to the supply voltage Vdd
 - The middle pin the **analog high impedance input**
 - One pin to the ground Vss



Potentiometer as Voltage Divider

Photoresistor

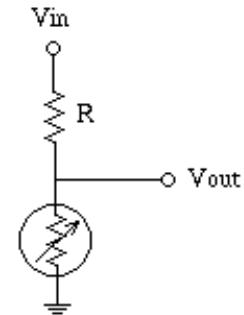
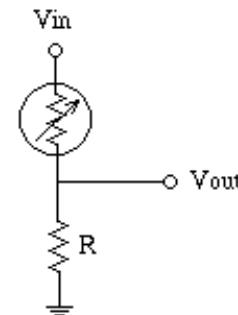
Is a light-controlled variable resistor. The resistance of a Photoresistor decreases with increasing incident light intensity

- Photo Resistor
 - 2 pins
 - R inverse proportional with the light

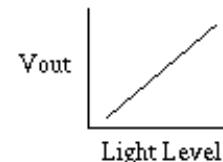


Using Photoresistors

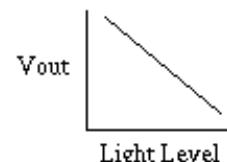
(The symbols with the circles are the photoresistors.)



This circuit gives an output voltage that increases with the light level.

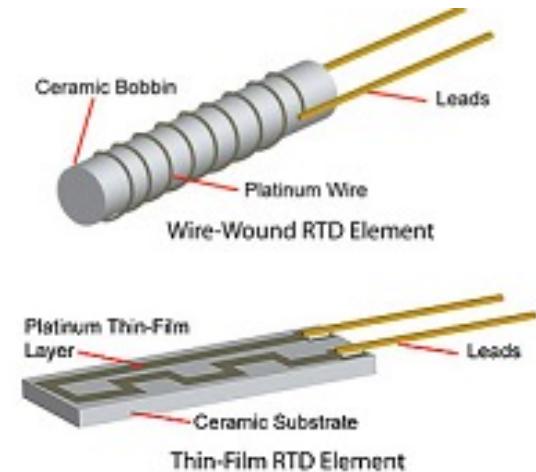
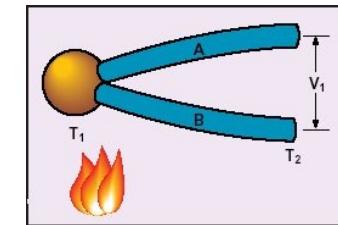
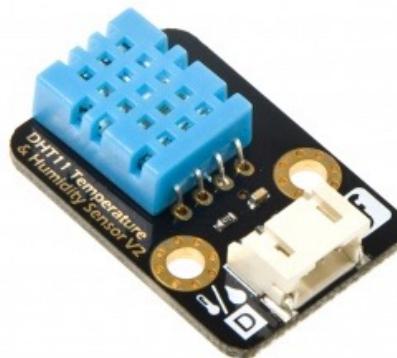
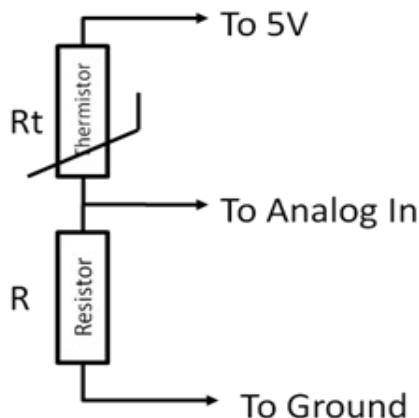


This circuit gives an output voltage that decreases with the light level.



Temperature Sensor

- Measure the change in temperature
- There are few different classes of temp sensors:
 - Thermocouples
 - Resistance temperature detectors
 - Temperature-transducer ICs
 - **Thermistors**

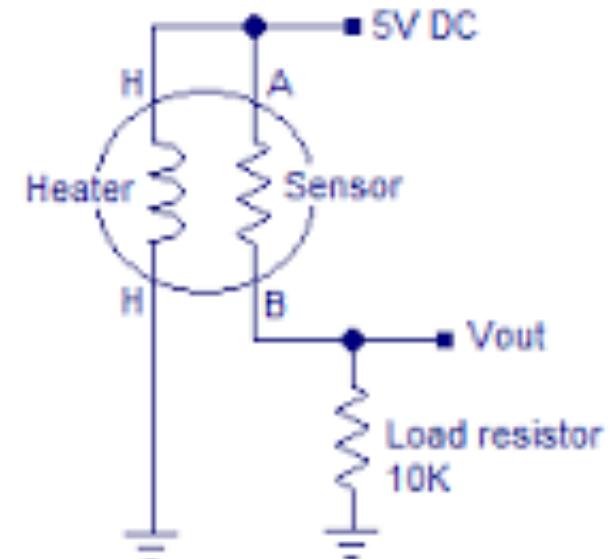
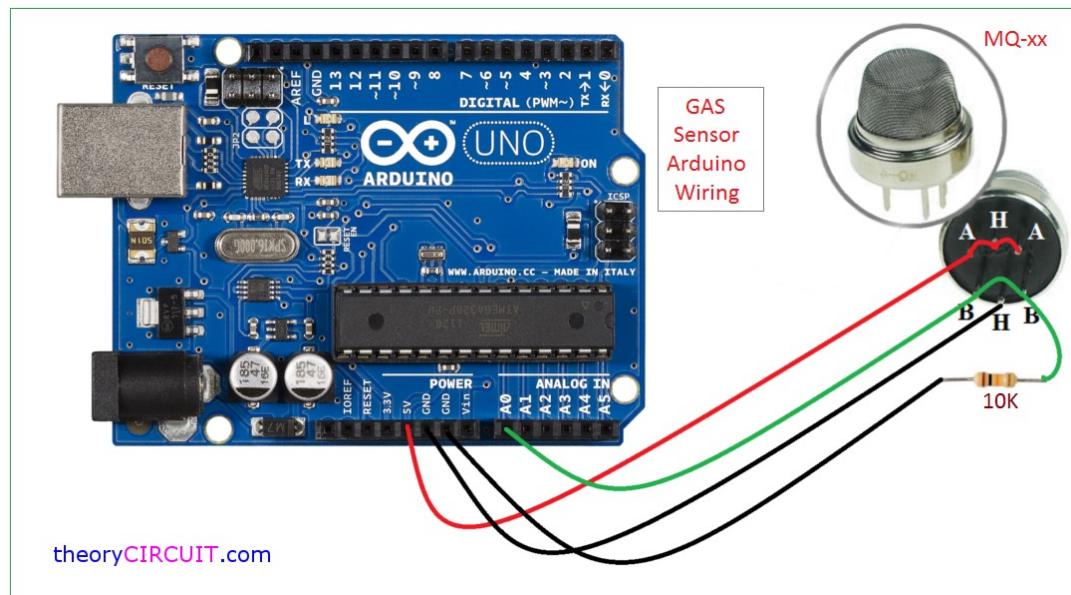


Thermistors are thermally sensitive resistors whose prime function is to exhibit a large, predictable and precise change in electrical resistance when subjected to a corresponding change in body temperature.

Gas Sensor

Gas Sensor

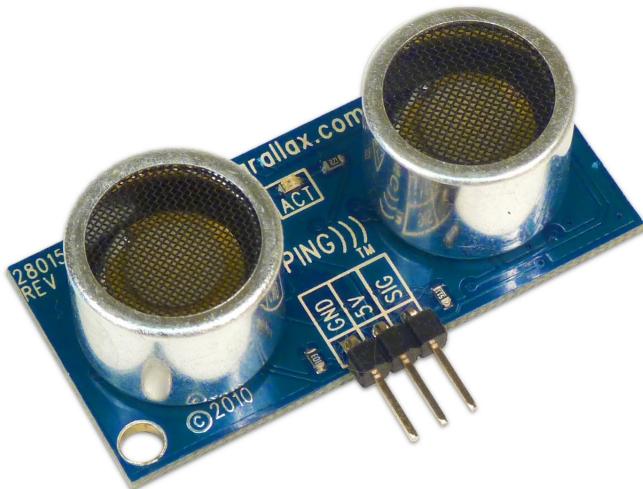
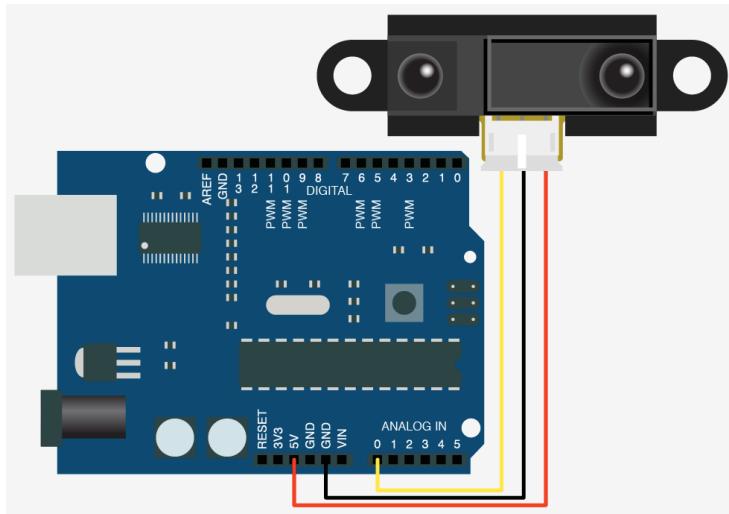
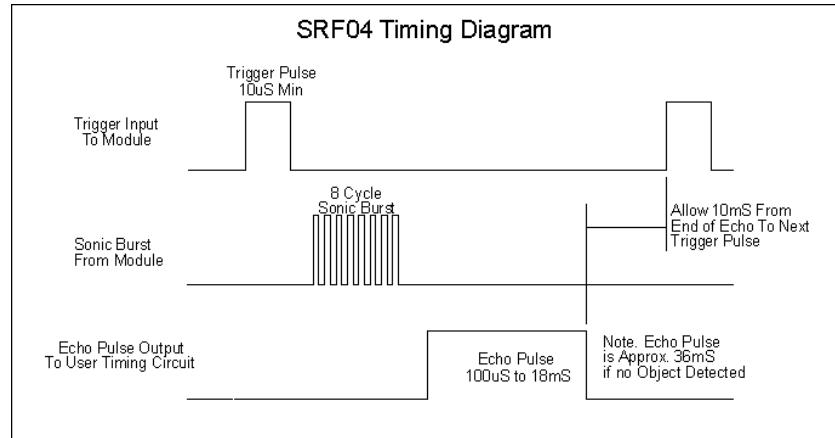
- Three pins
 - Vcc
 - Signal
 - Ground



MQ2 gas sensor connection diagram

Distance Sensor

- SRF04 sensor
 - Ultrasonic technology
 - Sends a pulse
 - Real time system
 - Works on microcontrollers
- Interested to know how it works:
 - <https://www.robot-electronics.co.uk/htm/srf04tech.htm>



Reading Analogue Input (an example!):

■ **Objective:**

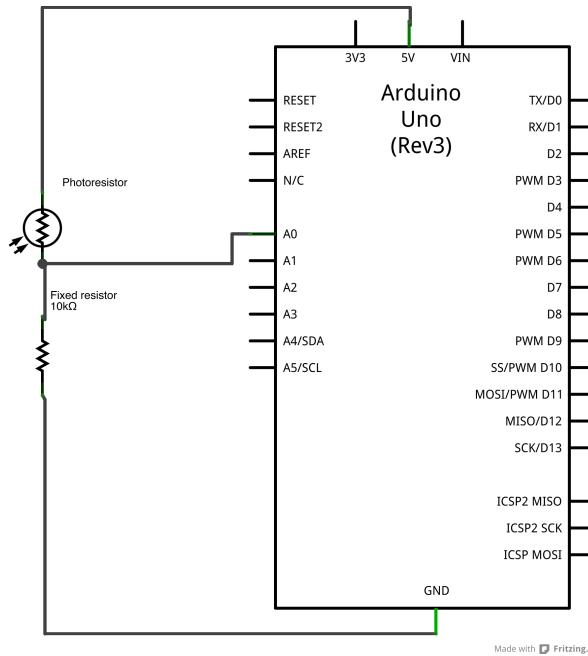
- Use a variable resistor (a potentiometer or a photoresistor), we read its value using one analog input of an Arduino or Genuino board and we change the blink rate of the built-in LED accordingly.

■ **What do we need:**

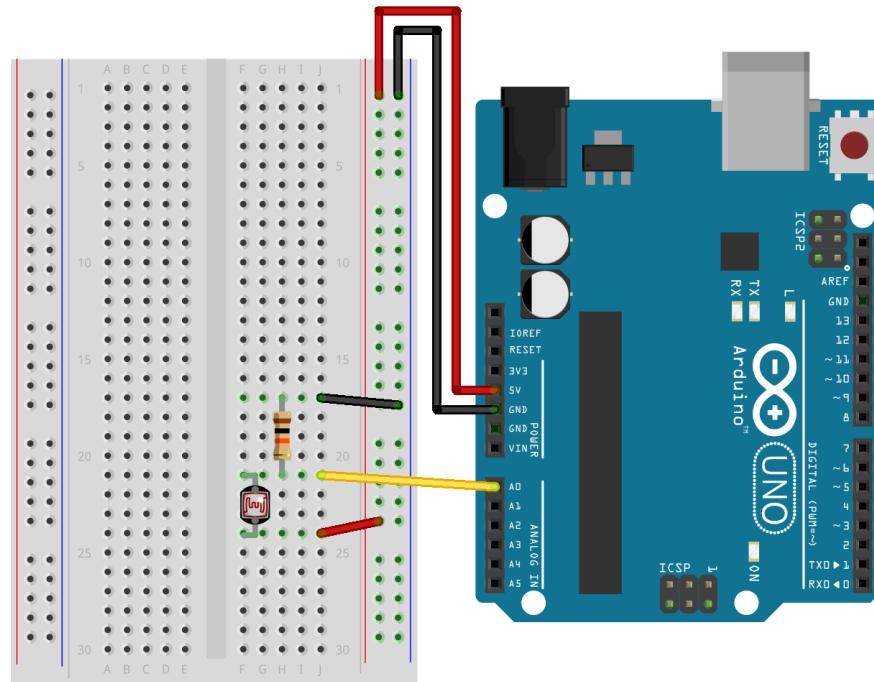
- Arduino or Genuino Board (our microprocessor)
- 10K ohm photoresistor and 10K ohm resistor
- built-in LED on pin 13 (already available on this board!)

Example Continued:

- Build the voltage divider and connect the mid point to the analogue sensor!



Made with Fritzing.org



Made with Fritzing.org

$$V_{out} = \frac{R_2}{(R_1+R_2)} \cdot V_{supply} \mid V_{supply} = 5V$$

Example Continued:

■ Microprocessor commands:

□ analogRead():

- converts the input voltage range, 0 to 5 volts, to a digital value between 0 and 1023
- **Which circuit does this conversion?**

Click here to get
more details!



□ **delay(variable):**

- processor busy wait for the duration specified by variable!

Click here to get
more details!

□ digitalWrite():

- writes a HIGH (1,5V) or LOW (0,0V) to a digital pin.

□ Digital pin (13) in this board is connected to the on board LED.

Example Continued

■ Program the microprocessor!

This example code is in the public domain.

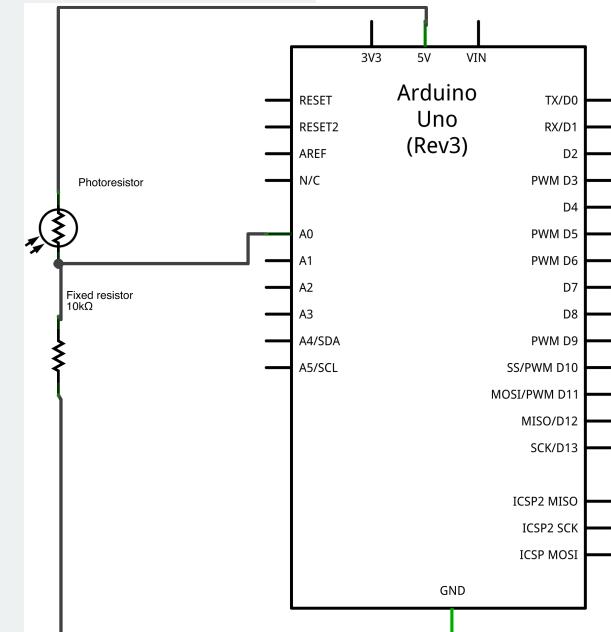
<http://www.arduino.cc/en/Tutorial/AnalogInput>

*/

```
int sensorPin = A0;      // select the input pin for the potentiometer
int ledPin = 13;          // select the pin for the LED
int sensorValue = 0;      // variable to store the value coming from the sensor

void setup() {
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
  // turn the ledPin on
  digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue);
  // turn the ledPin off:
  digitalWrite(ledPin, LOW);
  // stop the program for for <sensorValue> milliseconds:
  delay(sensorValue);
}
```



Made with Fritzing.org



Internet of Things

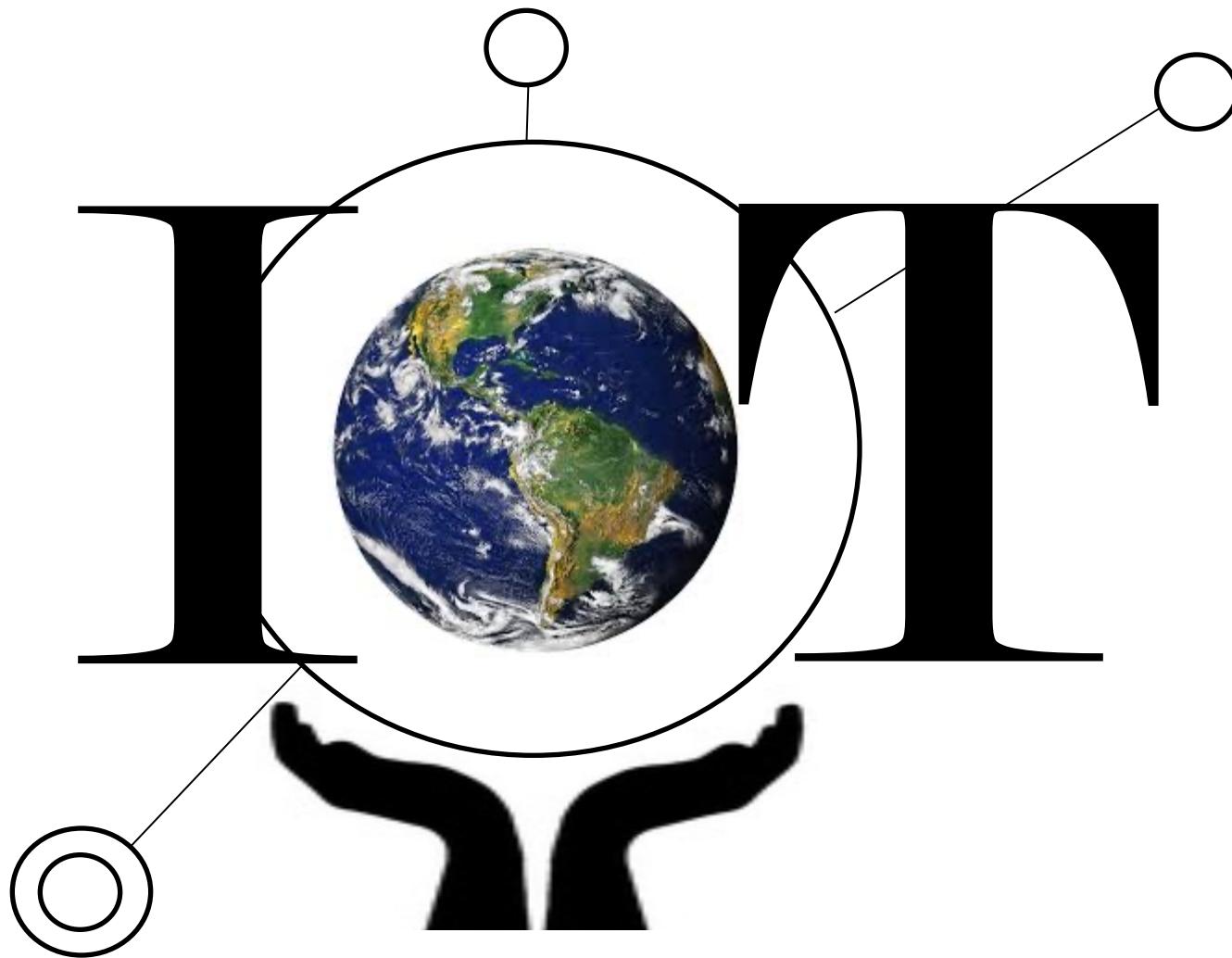
Senior Design Project Course

Introduction - Part 2

Lecturer: Avesta Sasan

University of California Davis
Fall 2023

Let's Get Started:



Required Reading:

- F. Samie, L. Bauer and J. Henkel, "**IoT technologies for embedded computing: A survey**," *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Pittsburgh, PA, 2016, pp. 1-10.
 - <http://ieeexplore.ieee.org/document/7750968/>

Let's Get Started!

The IoT was probably coined by **Kevin Ashton**, co-founder of Auto-ID Center at the Massachusetts Institute of Technology (MIT), as the title of a presentation in 1999

Internet Of Things

- Name Coined by Kevin Ashton 1999
 - Originally for RFID
 - Lipstick inventory problem
- Apps moving from the smartphone to a world of connected devices
- **Internet of Things aka IoT**



If RFID is an IoT..... Then???

Related Terms and Technologies:

Web of Things (WoT)

ambient intelligence (Aml)

ubiquitous computing

Big Data

Internet of Everything (IoE)

pervasive computing

Internet of Me (IoM)

Internet of Things(IoT)

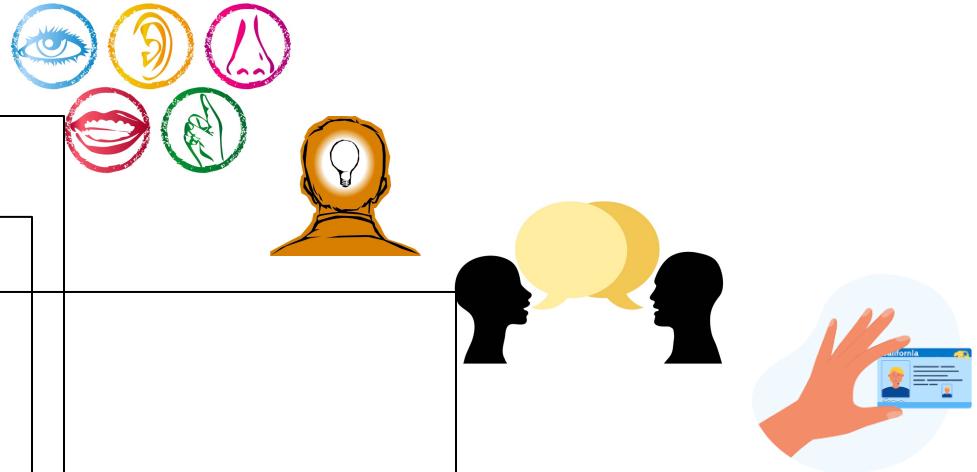
Smarter Planet

Cyber Physical Systems (CPS)

Industrial Internet of Things (IIoT)

What is IoT

- An IoT smart object is characterized by having the following 4 properties:
 - To be able to **Sense**
 - To be able to **Compute**
 - To be able to **Communicate**
 - To be **Uniquely Identified**
- IoT is a system of **Smart Objects** with **unique identifiers** and the **ability to transfer** data over a network without requiring human-to-human or human-to-computer interaction.



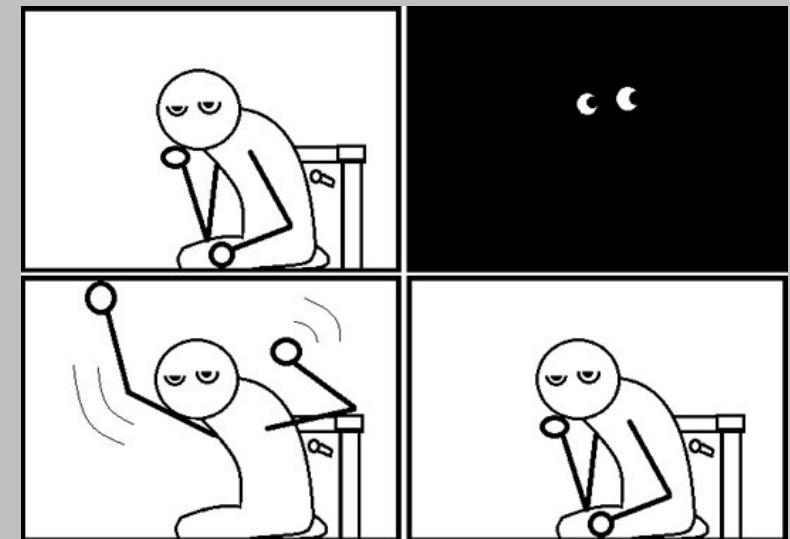
Sensors

- **Measure values** in digital or analogue form
- With little or no processing they **send the raw data**
- Best if they consume little **power**



Quiz:

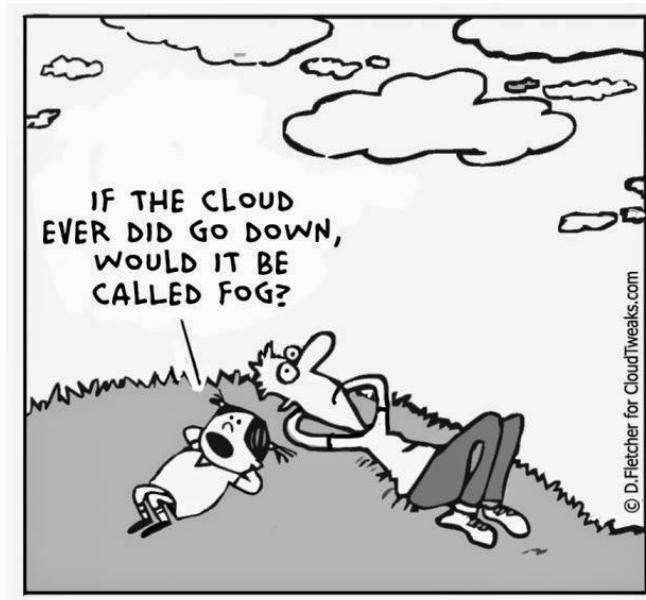
Is a motion sensor an IOT Object?
What is missing?



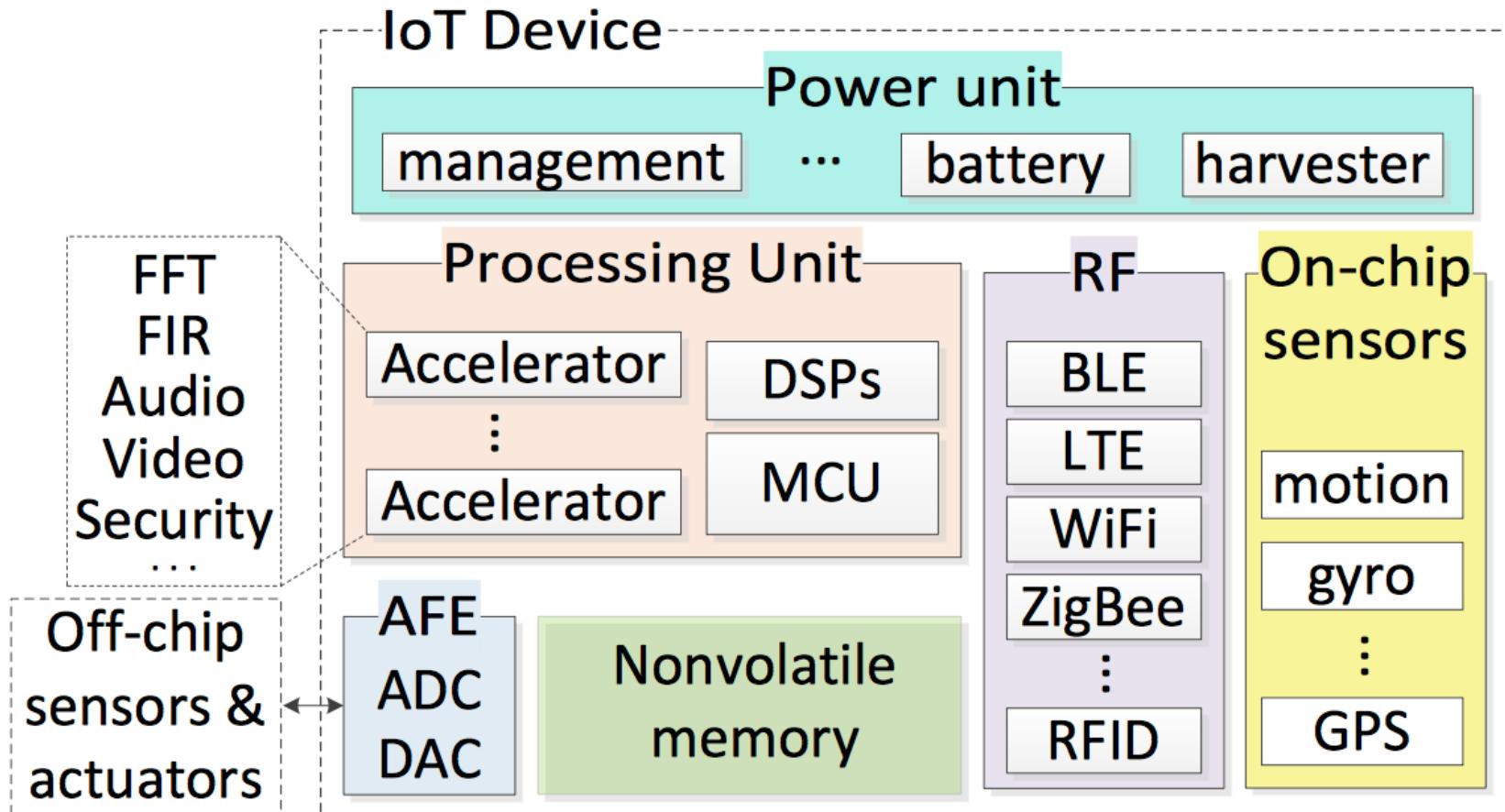
Motion Sensor controlling the light!

Local Processing and Local Storage

- Receive data from sensors
- Process the data (Edge/Fog Computing)
 - Make some decision locally
 - Store some of data locally
- Communicate raw or processed data to the Cloud.



General architecture of an IoT device



Flow of Data in Internet of Things



Image source: <http://www.cchc.cl/informacion-a-la-comunidad/industria-de-la-construccion/personaje/>

IoT makes Autonomous!

Passive



Active



Automation



Why?

- We like to collect data (information is power)
- We want to control things
- We want to automate processes
- We want to make things faster
- We want to enhance availability of services
- We want to make products cheaper
- etc....

What is Connected in IoT?

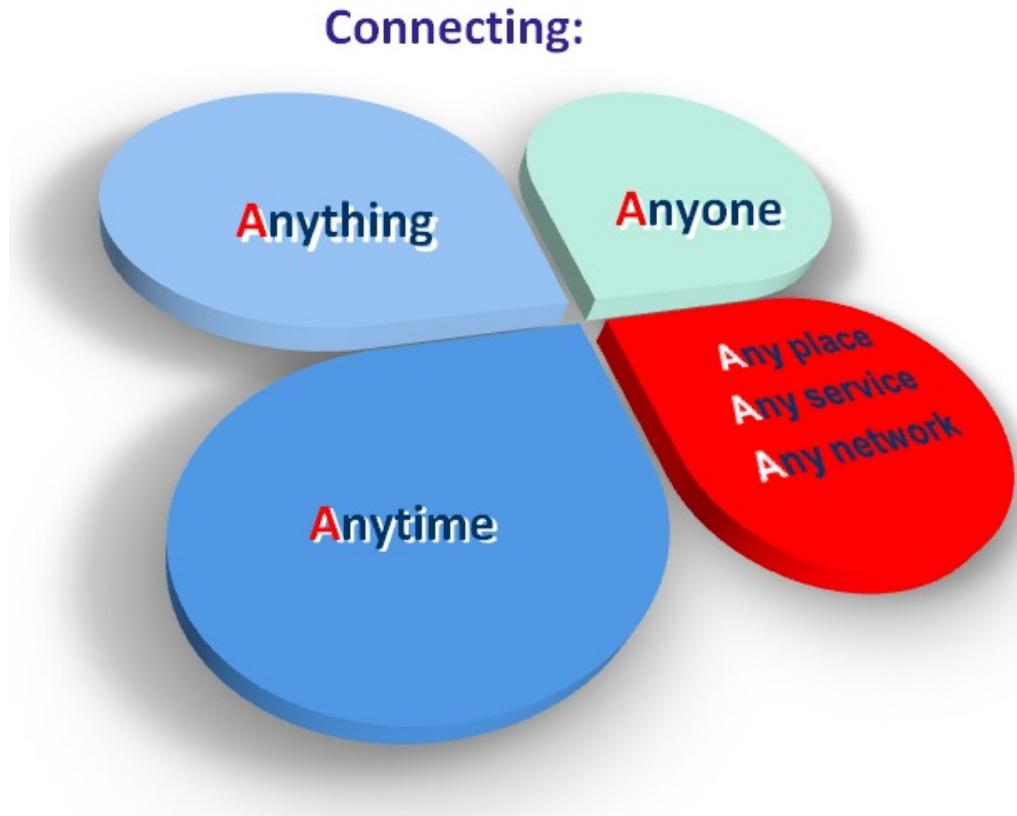
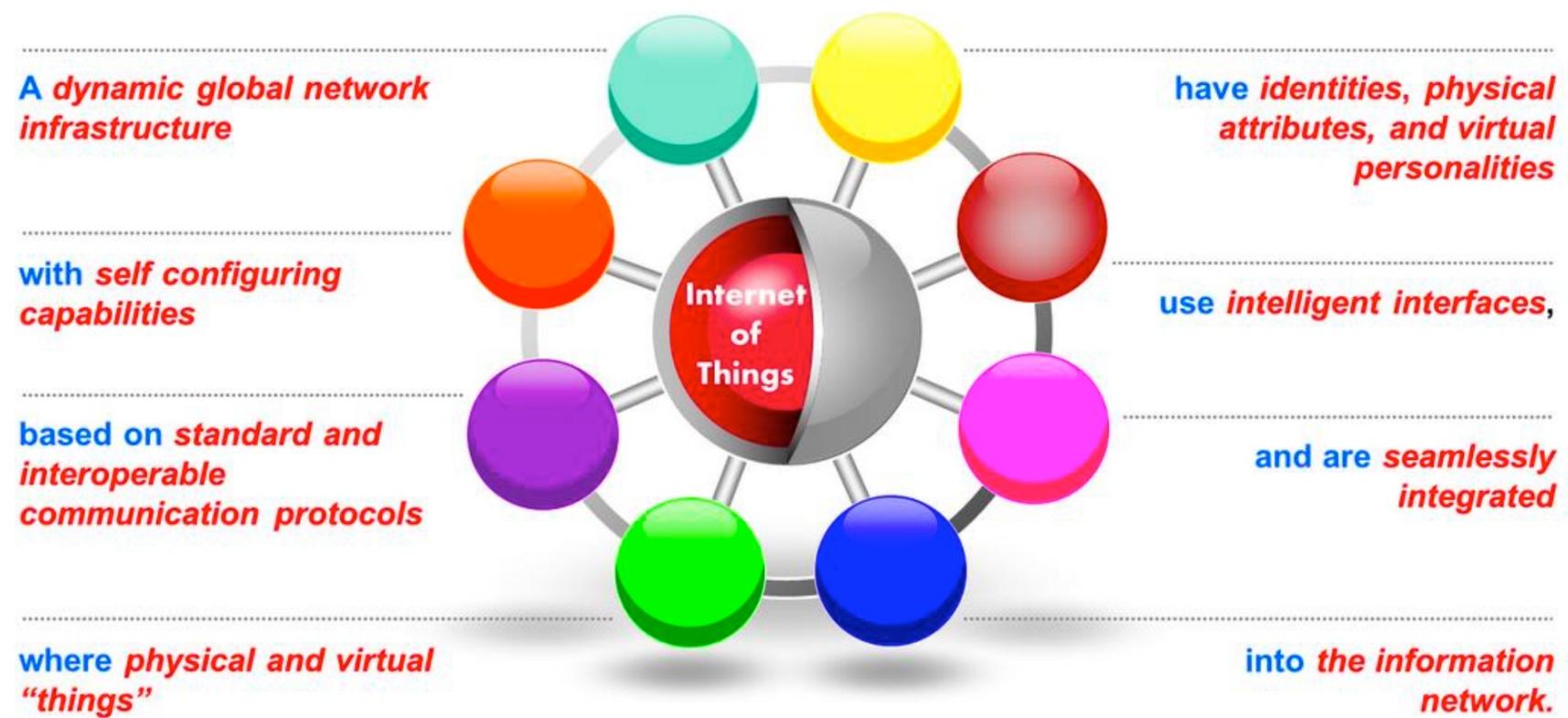


Image source: http://www.internet-of-things-research.eu/about_iot.htm

IERS definition of IoT?

The European Research Cluster on the Internet of Things (IERC) definition of the IoT is:



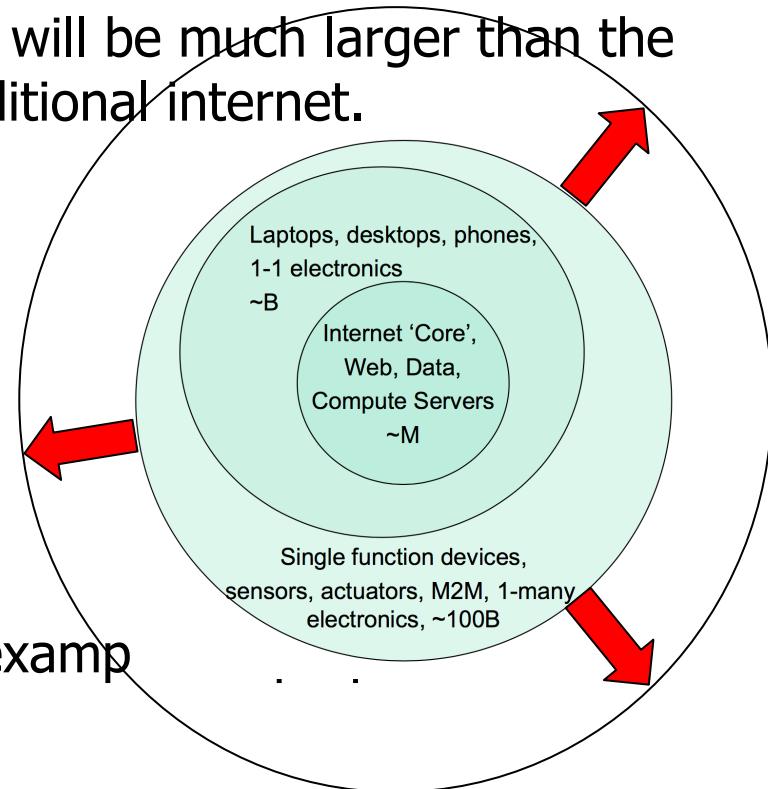
How IoT differs from Traditional Internet?

Topic	Traditional Internet	The Internet of Things (IoT)
Who creates content?	Human	Machine
How is the content consumed?	By request	By pushing information and triggering actions
How is the content combined?	Using explicitly defined links	Through explicitly defined operators
What is the value?	Answer questions	Action and timely information
What was done so far?	Both content creation (HTML) and content consumption (search engines)	Mainly content creation

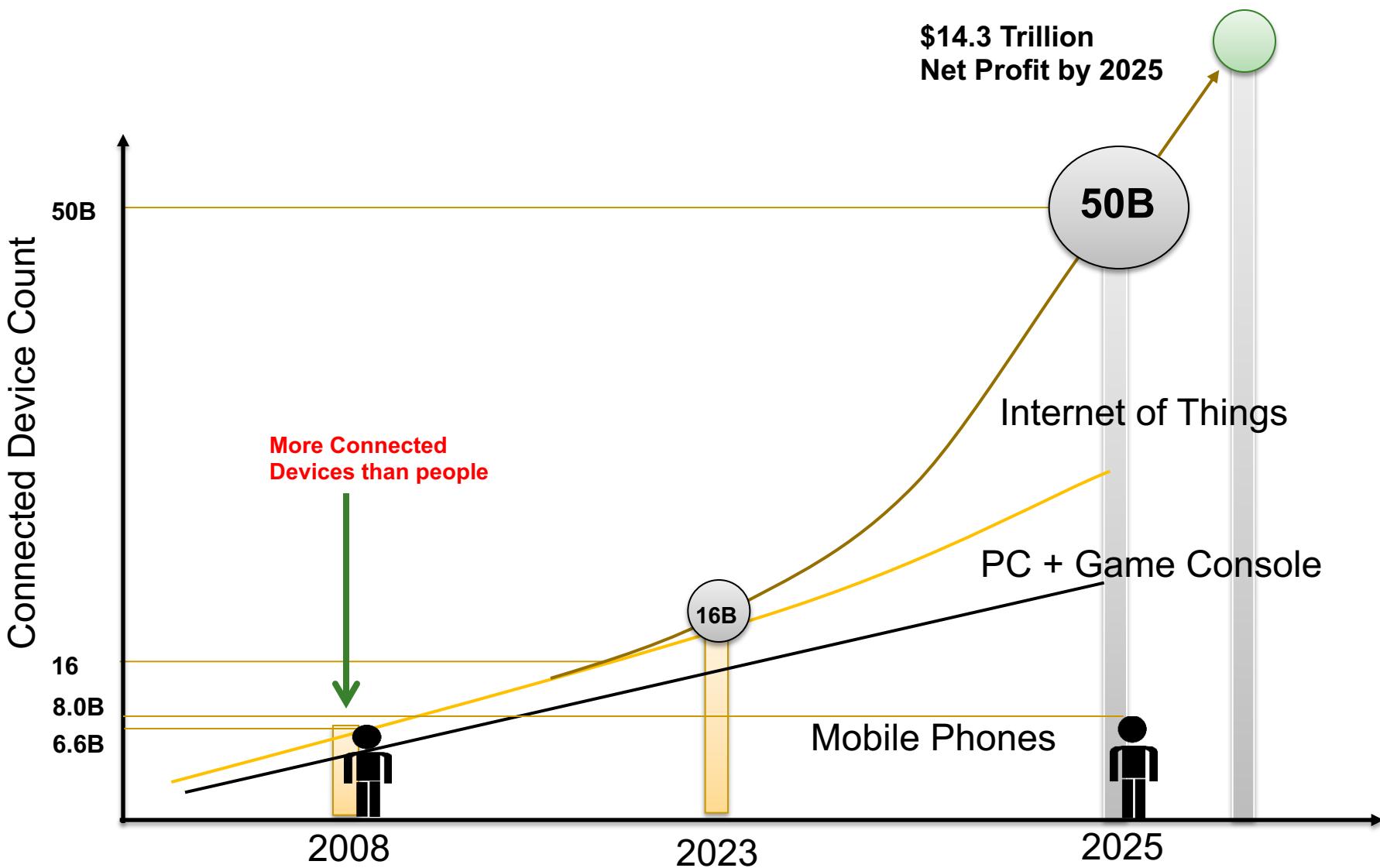
Source: <https://www.rtinsights.com/differences-between-the-iot-and-traditional-internet/>

IoT Is far larger than Internet

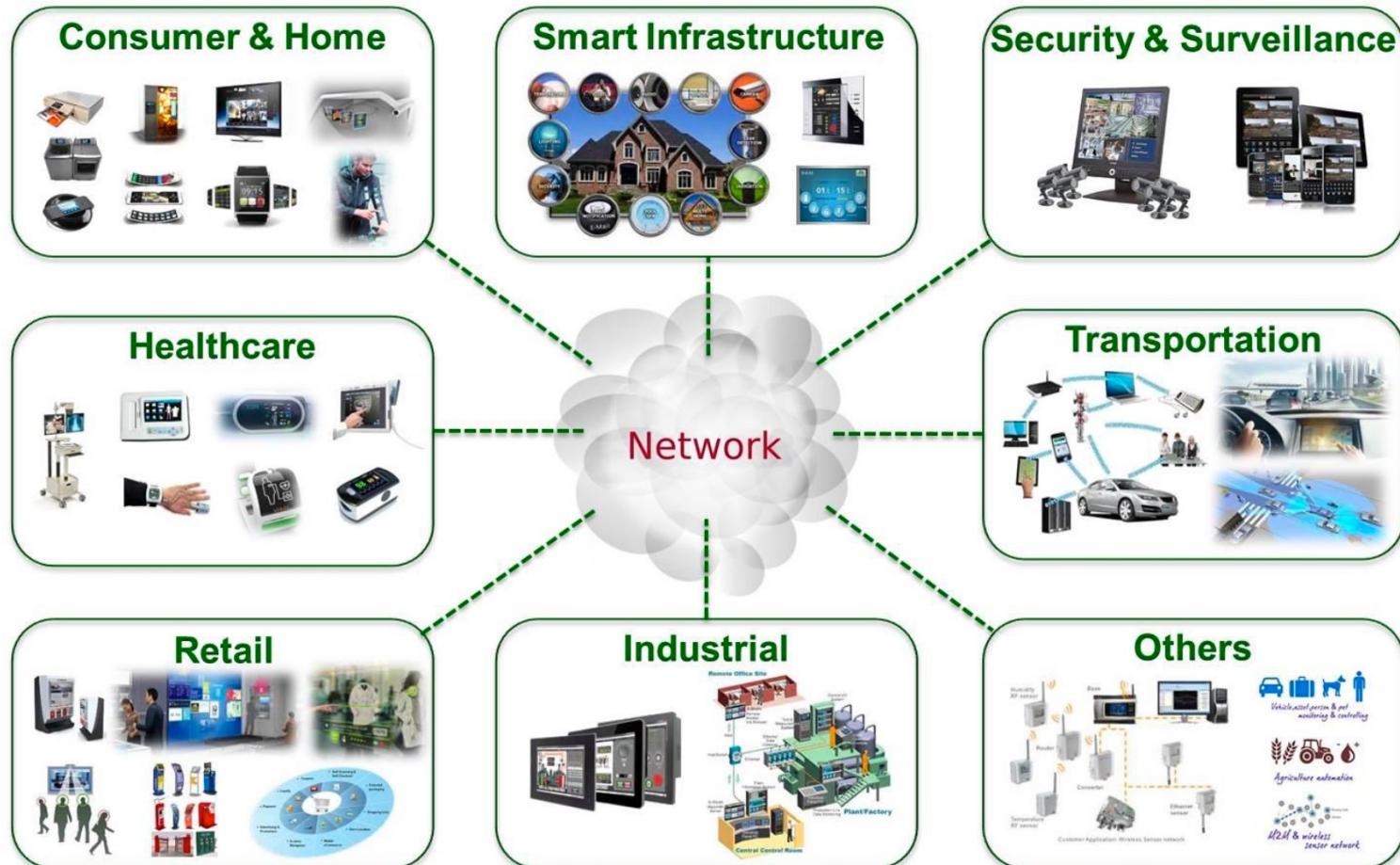
- Number of connected objects in the IoT will be much larger than the number of connected devices in the traditional internet.
- Objects in IoT:
 - Have limited functions
 - Are not useful by themselves
 - Are connected to low capacity networks
 - Gen/Receive Limited data per device
 - Do not have proper UI
 - Are In large numbers >> 1/human
 - Interact with the real world
- Devices connected to the Internet (for example phone)
 - Support wider range of functions
 - Are useful devices for the user
 - Are connected to WWW
 - Could Gen/Receive large amount of data
 - Have a proper UI (ex. OSX in apple phones)
 - Are in proportional number to human users ($\leq 1/\text{human}$)
 - Interact with human users



Growth of Connected Devices



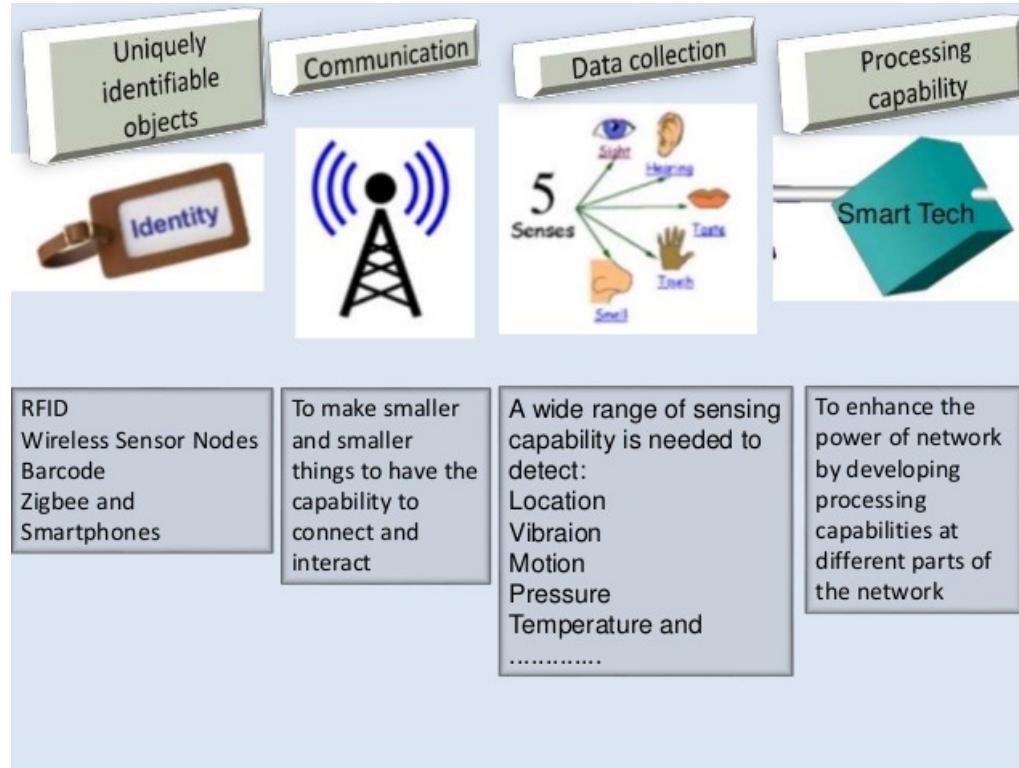
Many Application Domains



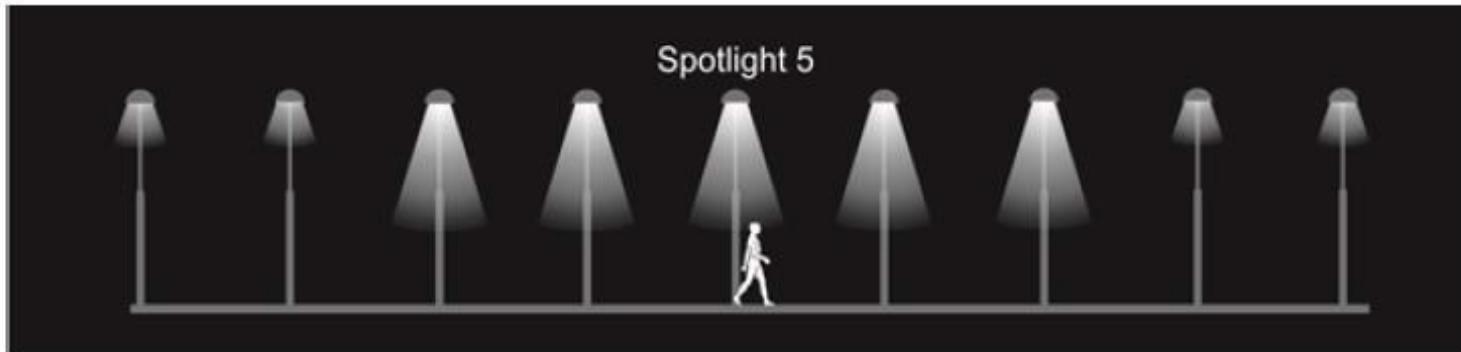
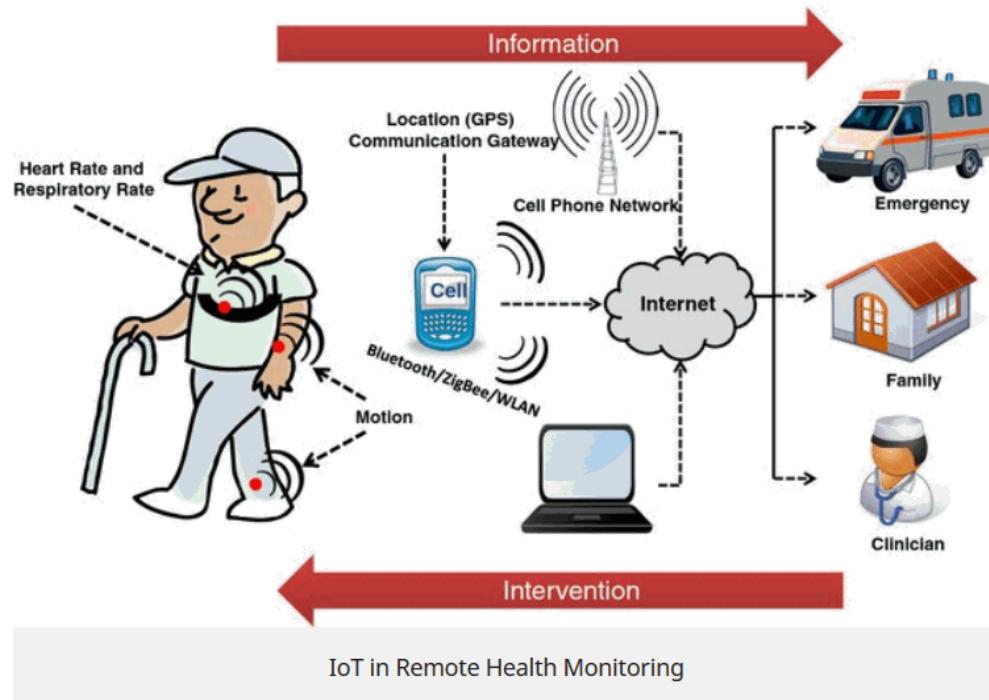
Vivante and the Vivante logo are trademarks of Vivante Corporation. All other product, image or service names in this presentation are the property of their respective owners. © 2013 Vivante Corporation

Things

- Goods
- Objects
- Machines
- Appliances
- Buildings
- Vehicles
- Animals
- **People**
- Plans
- Soil
- Etc....



Us as Objects or Hosting Objects!



Smart Devices on Our Bodies!

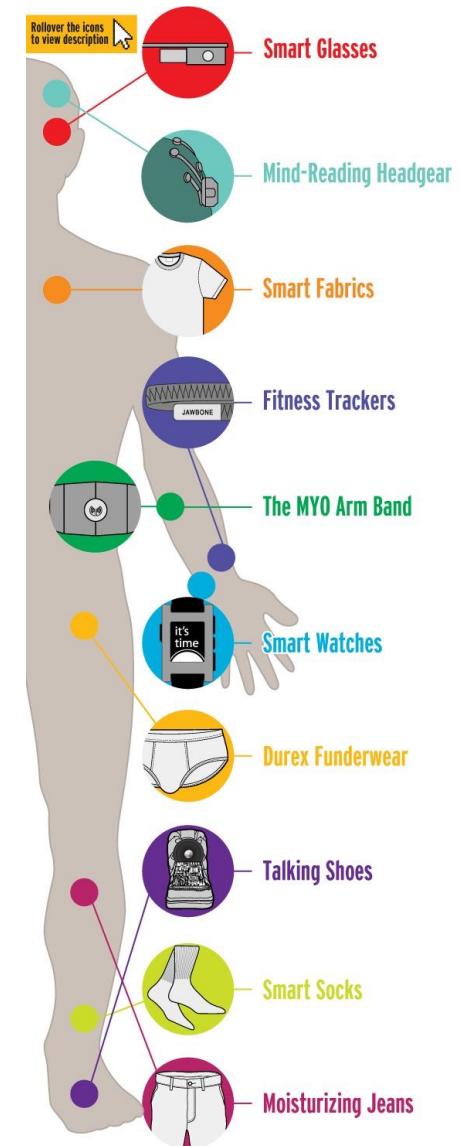
- Technology on our bodies, for
 - Convenience
 - Health
 - ...



- Open and read the following links:

<https://googleblog.blogspot.com/2014/01/introducing-our-smart-contact-lens.html>

<https://www.getqardio.com/qardiocore-wearable-ecg-ekg-monitor-iphone/>



Smart Devices in Our Bodies

- Health, Monitoring, Management

WIRELESS IMPLANTABLE MEDICAL DEVICES

Deep Brain
Neurostimulators



Cochlear Implants



A **cochlear implant** is an electronic medical device that replaces the function of the damaged inner ear.

Gastric
Stimulators



This helps decrease nausea and vomiting in some patients with gastroparesis.

Foot Drop
Implants



Foot drop implants helps stroke victims suffering from **foot** paralysis

Cardiac Defibrillators/
Pacemakers

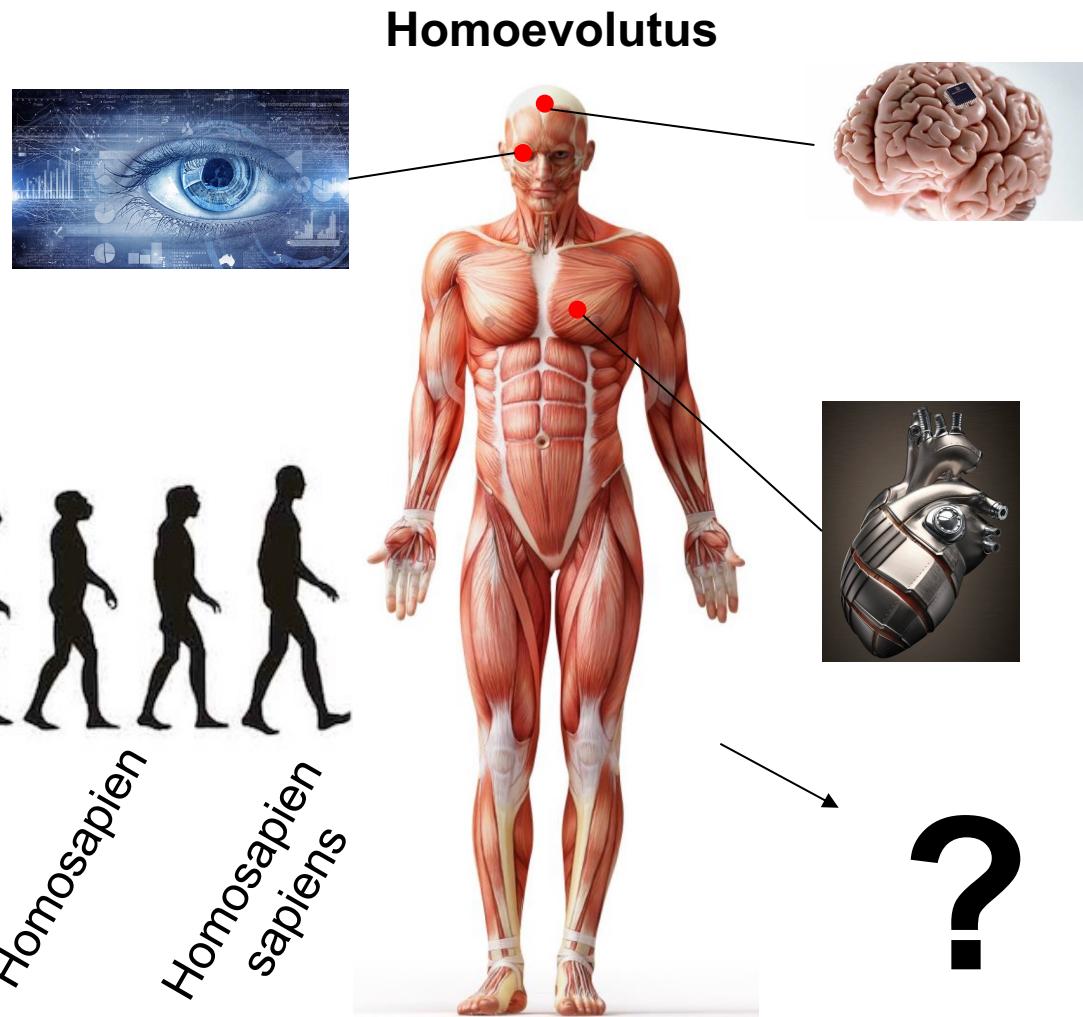
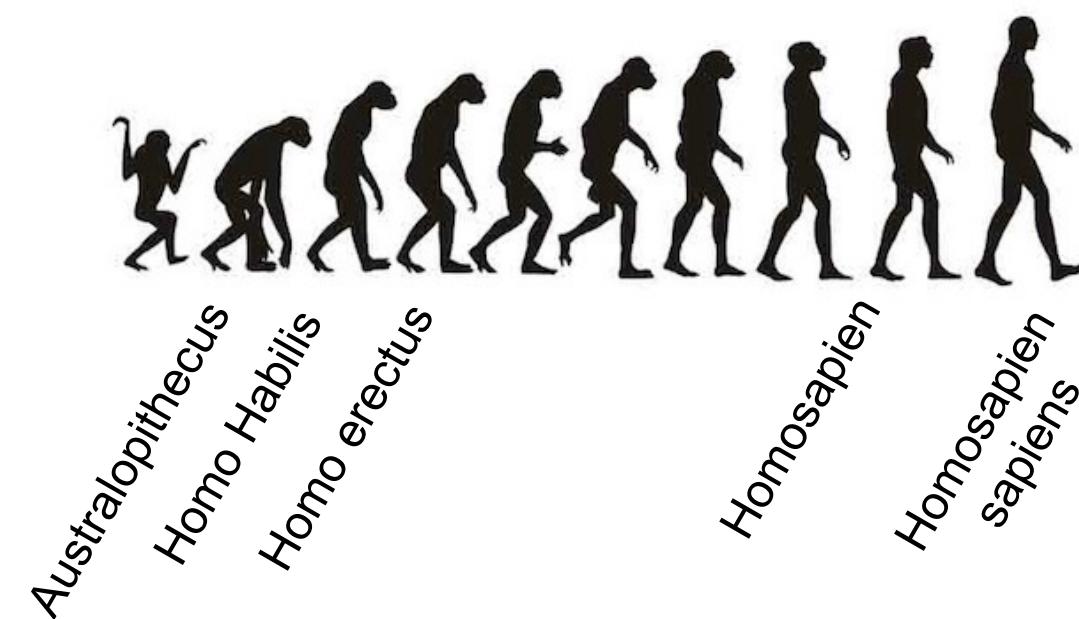


Insulin Pumps



Smart Devices to Upgrade Our Bodies

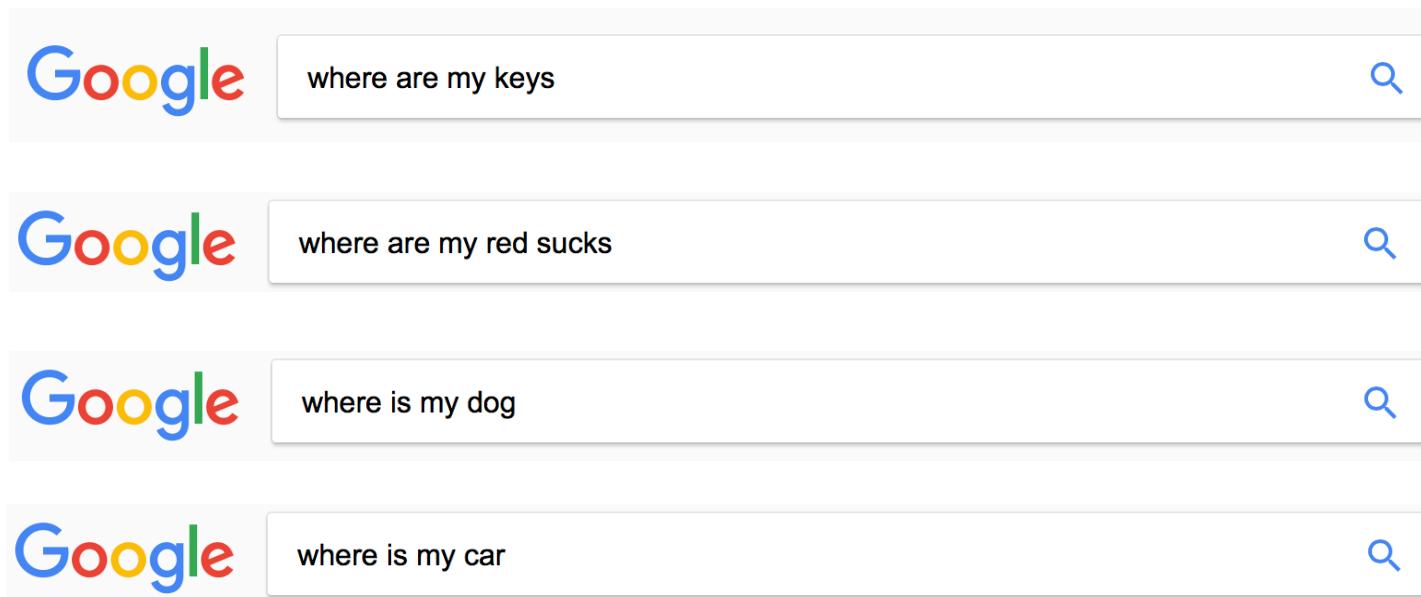
- Improve
- Upgrade
-



Interesting video: <https://www.youtube.com/watch?v=01hbkh4hXEk>

Reality Search Engines

- Things are searchable, reachable and identifiable, therefore search engines could be extended into searching physical world for things!

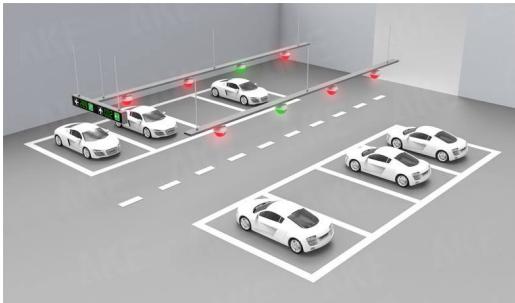


Managing Things, Building Services

Not Smart!



Some Smart!

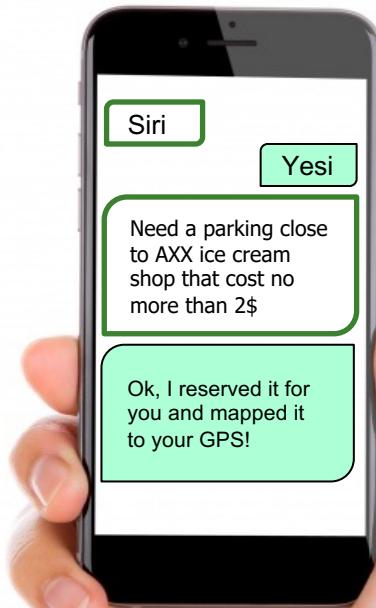


Siri

Yes!

Need a parking close
to AXX ice cream
shop that cost no
more than 2\$

Ok, I reserved it for
you and mapped it
to your GPS!



Smart!



Control Things

- Control **yourself**
 - e.g. control the lighting of your house as you desire using a voice command!
- Or Give the control to **grid**
 - e.g. charge my electric car when cost of electricity is the lowest!



Privacy is a Concern!!!

