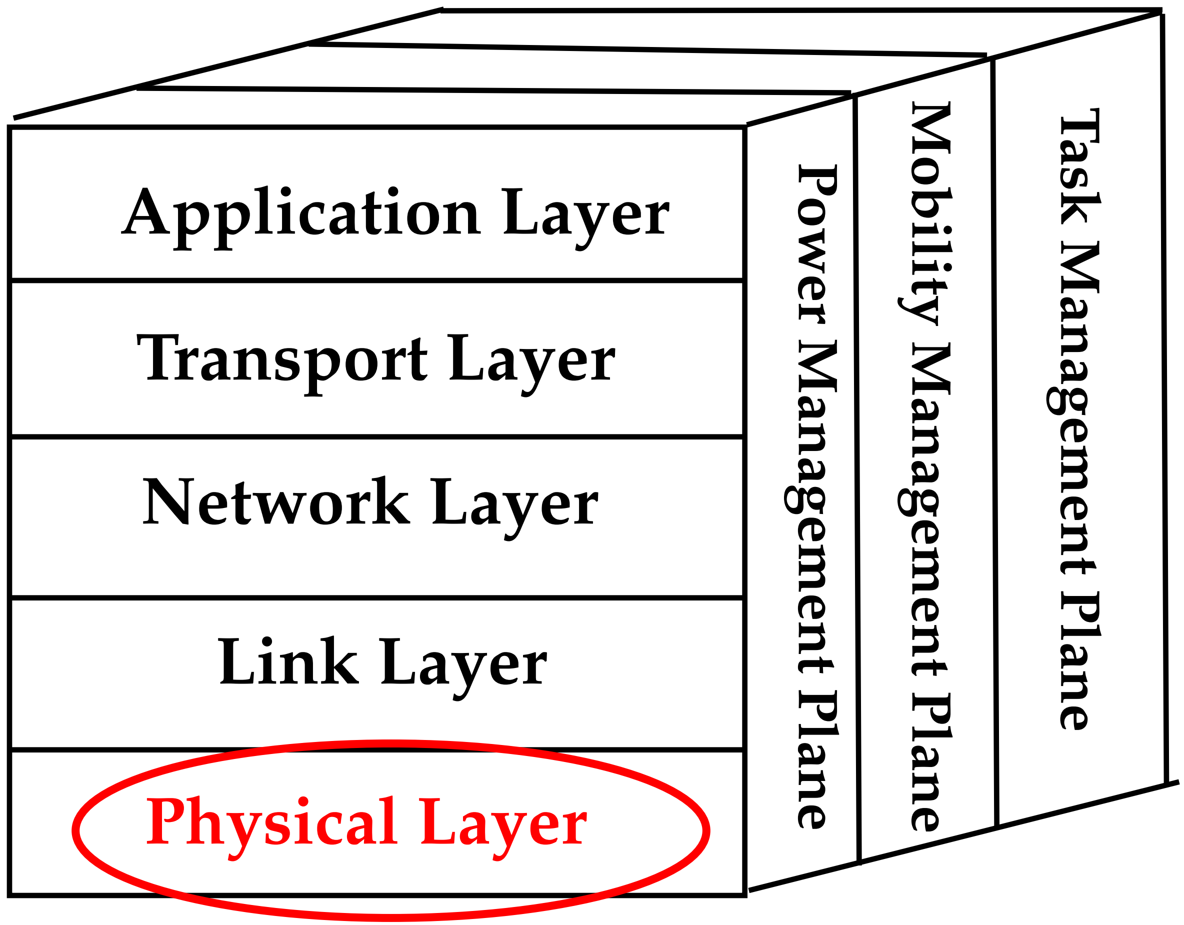
Institute for the Wireless Internet of Things at Northeastern University

EECE 5155
Wireless Sensor Networks (and The Internet of Things)

Prof. Francesco Restuccia
Email: f.restuccia@northeastern.edu

Protocol Stack



Target of this class

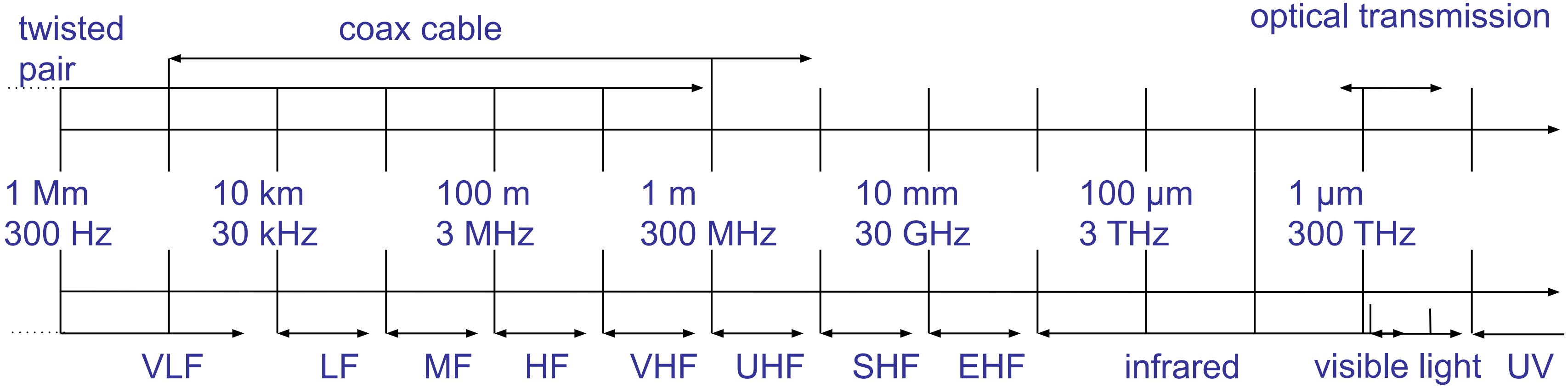
- Basic understanding of the peculiarities of wireless communications
  - ''Wireless channel" as abstraction of these properties – e.g., bit error patterns
  - Focus on **radio-frequency (RF)** communications

- Impact of different factors on communication performance
  - Frequency band
  - Transmission power
  - Modulation scheme

- Understanding of energy consumption for radio communications

Radio spectrum for communication

- Which part of the electromagnetic spectrum is used for communication?
  - **Not all frequencies are equally suitable for all tasks** – e.g., wall penetration, different atmospheric attenuation (oxygen resonances, …)



- VLF = Very Low Frequency
- LF = Low Frequency
- MF = Medium Frequency
- HF = High Frequency
- VHF = Very High Frequency

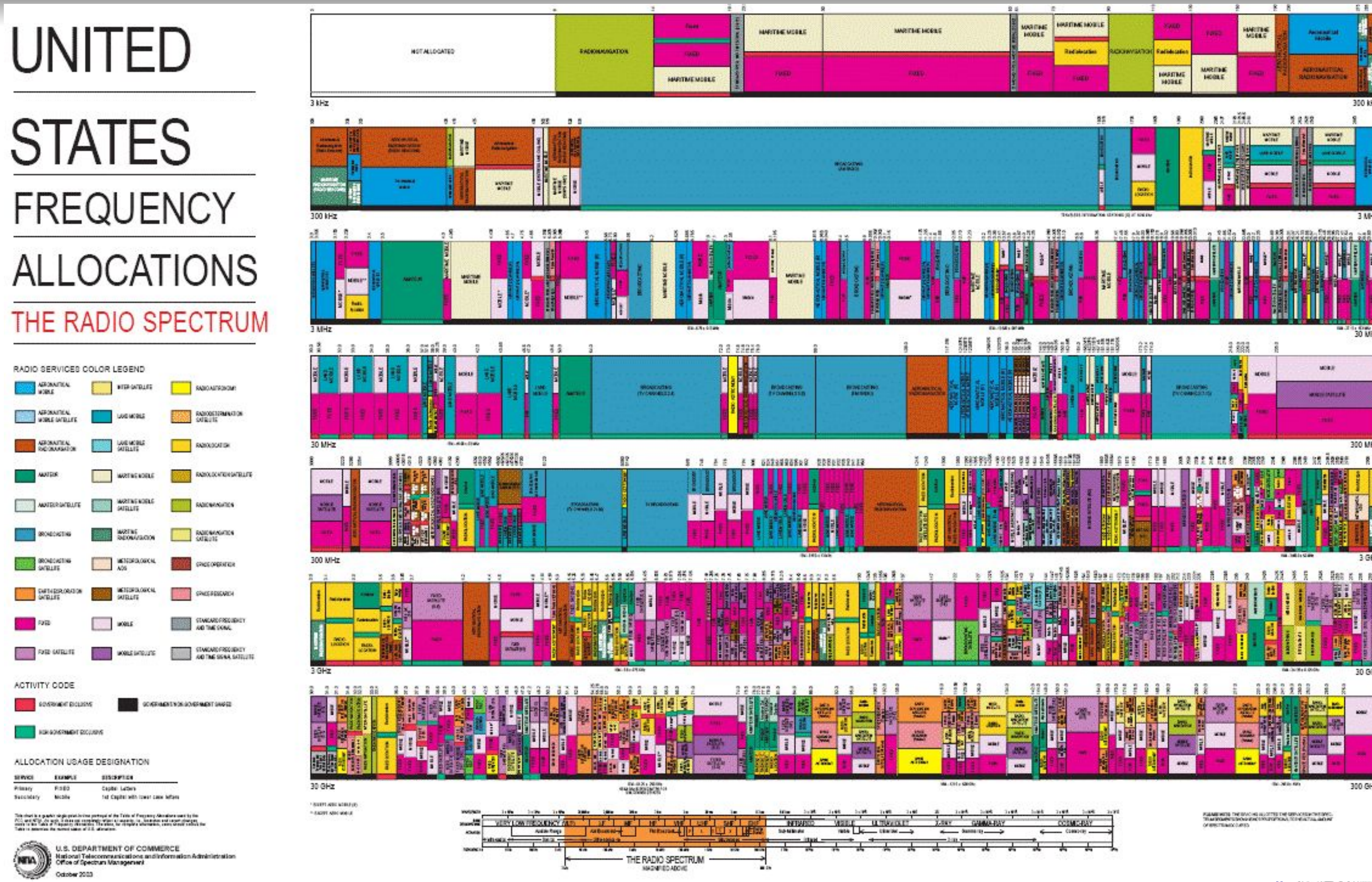
UHF = Ultra High Frequency
SHF = Super High Frequency
EHF = Extra High Frequency
UV = Ultraviolet Light

Frequency Allocation

- Some spectrum bands are allocated to specific uses
  - Cellular phones, analog television/radio broadcasting, DVB-T, radar, emergency services, radio astronomy, …
- ISM bands (Industrial, scientific, medical) can be used without a license
- **Why do we use different bands for transmission?**

ISM bands	
Frequency	
13,553-13,567 MHz	
26,957 – 27,283 MHz	
40,66 – 40,70 MHz	
433 – 464 MHz	Europe
900 – 928 MHz	Americas
2,4 – 2,5 GHz	802.11b/g, Bluetooth
5,725 – 5,875 GHz	802.11a
24 – 24,25 GHz	

US Frequency Allocation



Modulation: Transmitting Data With Radio Waves

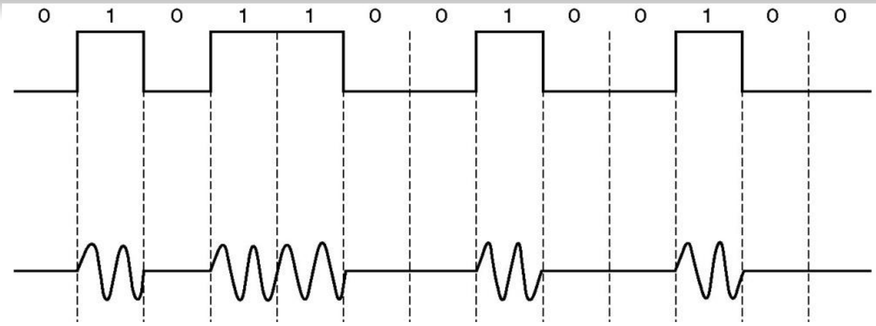
- Assumption: Transmitter can send a radio wave, receiver can detect whether such a wave is present and also its parameters
- Parameters of a wave = sine function:

$$s(t) = A(t) \sin(2\pi f(t) t + \phi(t))$$

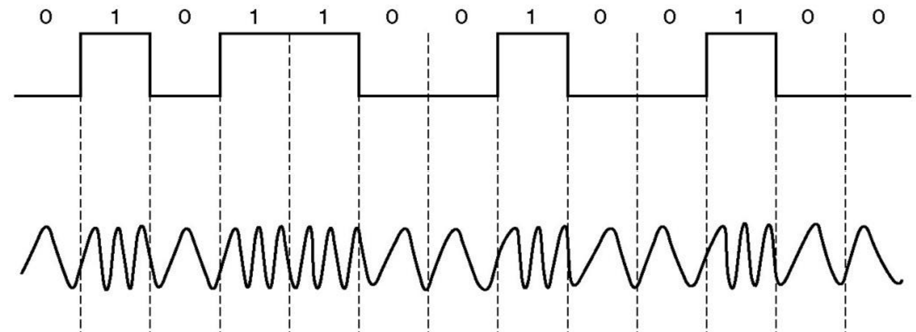
  – Parameters: amplitude *A(t)*, frequency *f(t)*, phase *φ(t)*
- Manipulating these three parameters allows the sender to encode data; receiver reconstructs data from signal
- Simplification: Receiver "sees" the same signal that the sender generated – **not true**, see later!

Modulation Examples

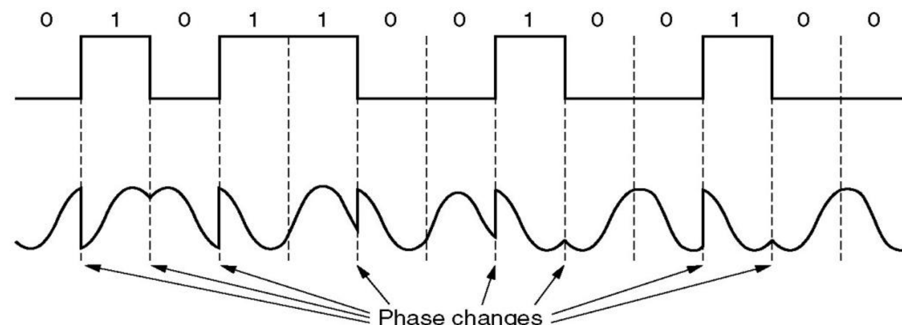
- **Amplitude Shift Keying**



- **Frequency Shift Keying**

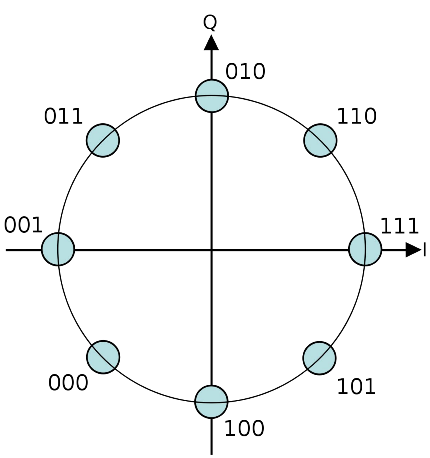


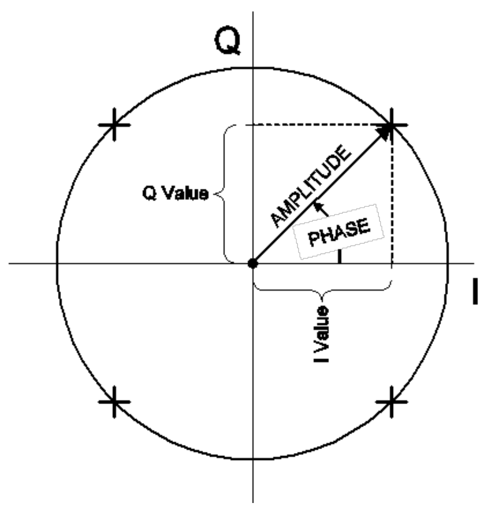
- **Phase Shift Keying**



Modulation and Keying - IQ Data

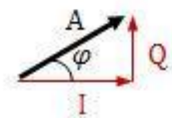
- How to manipulate a given signal parameter?
  - Set the parameter to an arbitrary value: ***analog modulation***
  - Choose parameter values from a finite set of values: ***digital keying***
  - Focus on digital keying
  - Precisely varying the phase of a high-frequency carrier sine wave in a hardware circuit according to an input message signal is difficult. Circuit would be expensive and difficult to design/build
  - I/Q Data is used





$$\cos(\alpha+\beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)$$

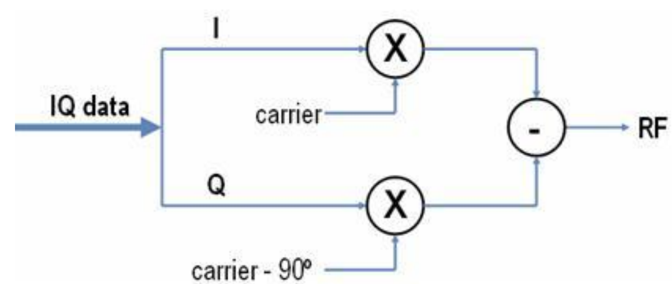
$$A\cos(2\pi f_c t + \varphi) = A\cos(2\pi f_c t)\cos(\varphi) - A\sin(2\pi f_c t)\sin(\varphi)$$



$$I = A\cos(\varphi)$$

$$Q = A\sin(\varphi)$$

$$A\cos(2\pi f_c t + \varphi) = I\cos(2\pi f_c t) - Q\sin(2\pi f_c t)$$

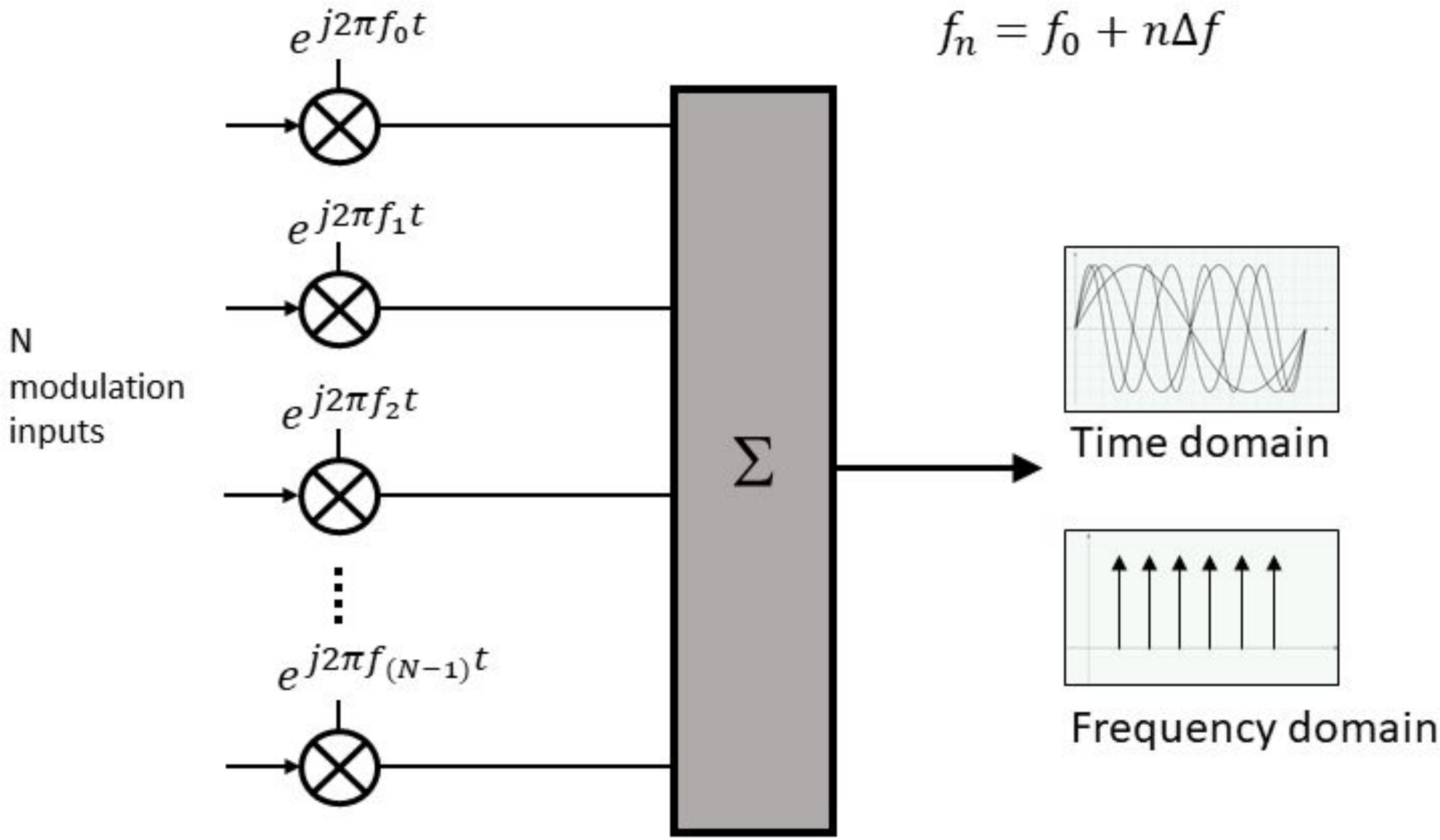


Source: TI.com

Receiver: Demodulation

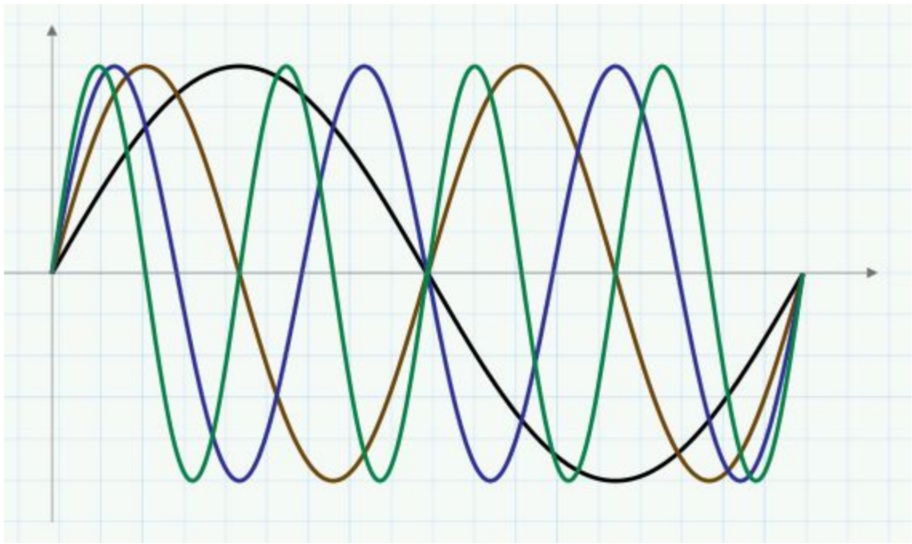
- The receiver looks at the received waveform and matches it with the data bit that caused the transmitter to generate this waveform
  - Necessary: one-to-one mapping between data and waveform
  - Because of channel imperfections, this is at best possible for **digital signals**, but not for analog signals
- Problems caused by
  - **Carrier synchronization**: frequency can vary between sender and receiver (drift, temperature changes, aging, …)
  - **Bit synchronization** (actually: symbol synchronization): When does symbol representing a certain bit start/end?
  - **Frame synchronization:** When does a packet start/end?
  - Biggest problem: Received signal is ***not*** the transmitted signal!

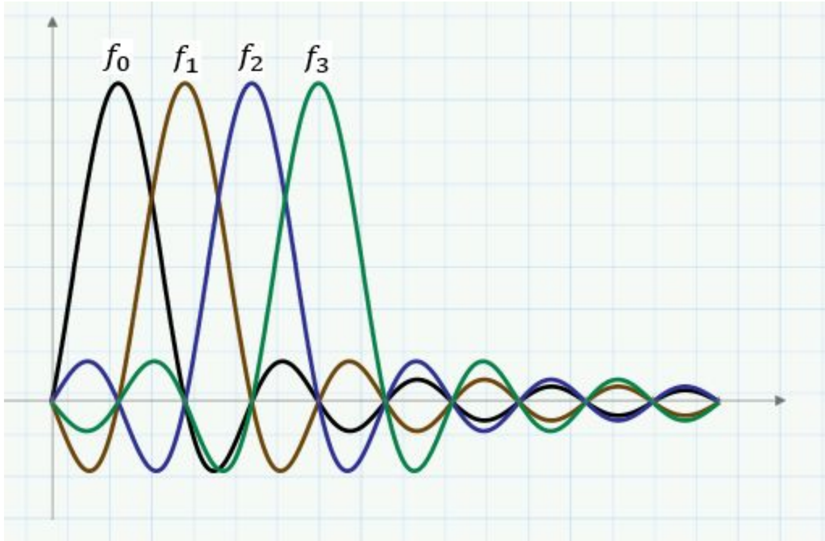
Multi-carrier transmission: OFDM



www.5gtechnologyworld.com

Multi-carrier transmission: OFDM (1)

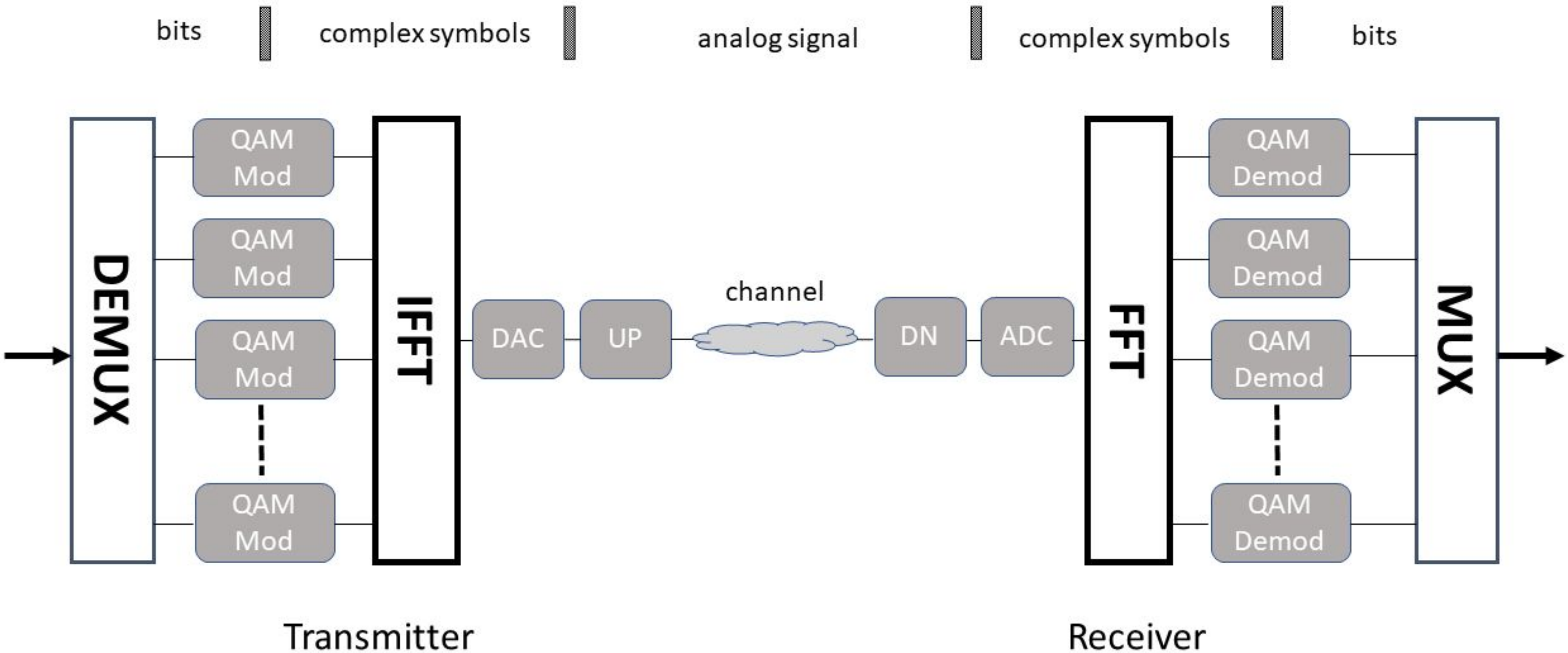




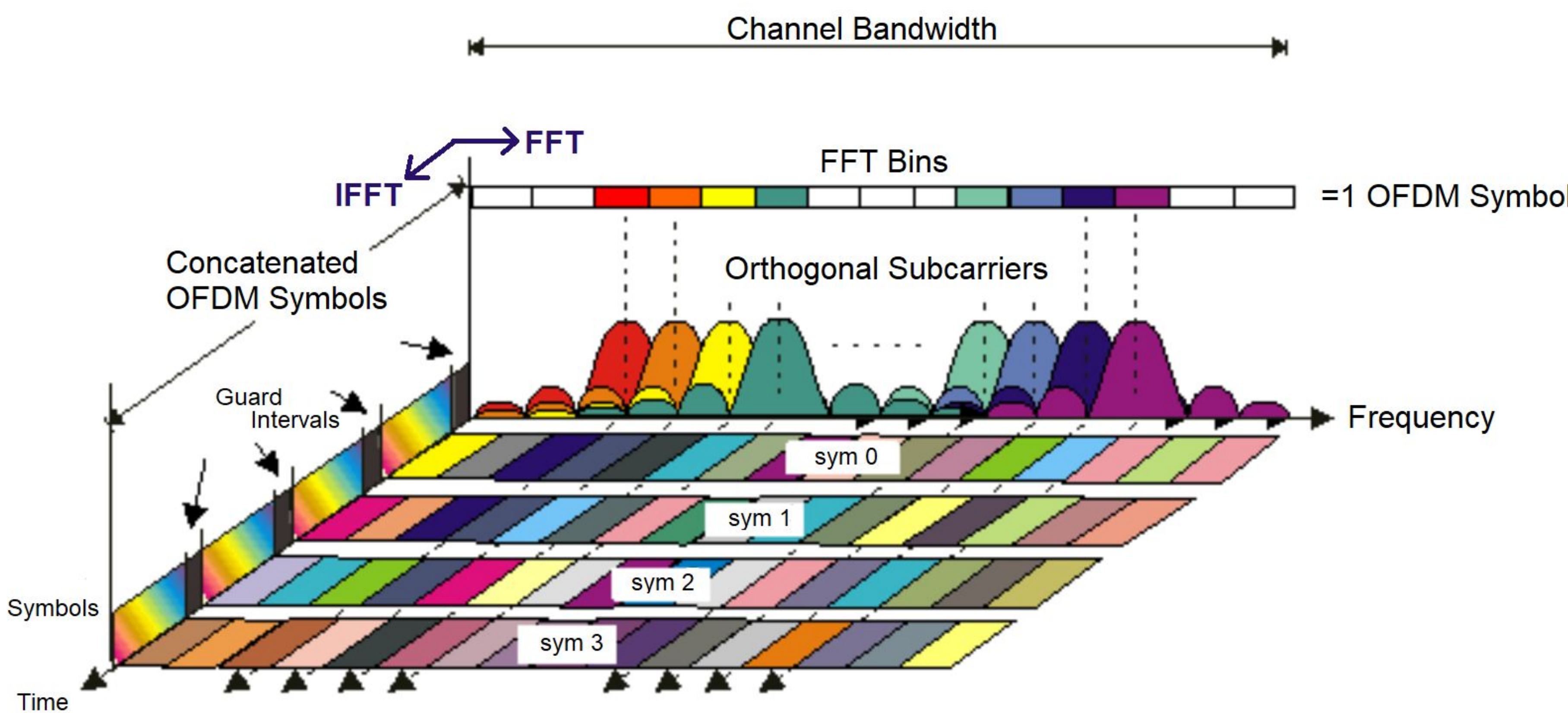
- Time domain
- Frequency domain

What's the advantage of OFDM?

Multi-carrier transmission: OFDM (2)



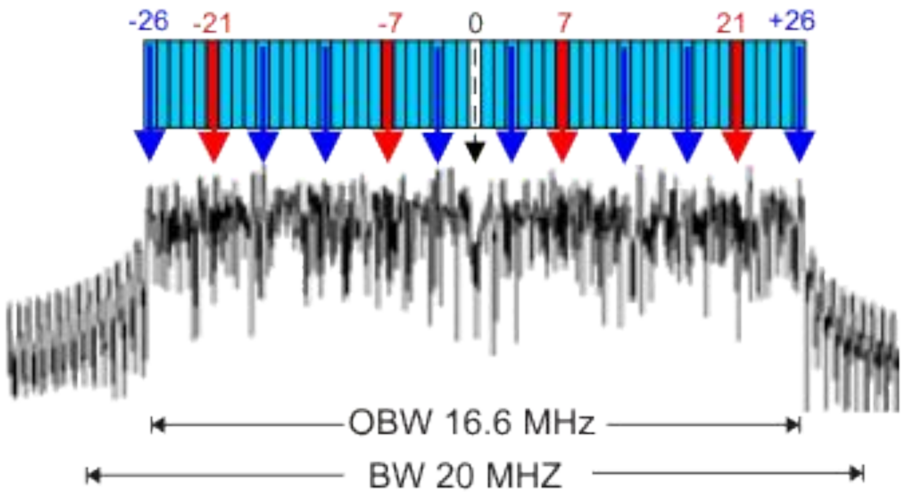
Time-Frequency Visualization



Real-World Example: PHY layer of Wi-Fi

Source: Keysight.com

802.11a OFDM PHY Parameters	
BW	20 MHZ
OBW	16.6 MHZ
Subcarrer Spacing	312.5 Khz (20MHz/64 Pt FFT)
Information Rate	**6**/9/**12**/18/**24**/36/48/54 Mbits/s
Modulation	BPSK, QPSK, 16QAM, 64QAM
Coding Rate	1/2, 2/3, 3/4
Total Subcarriers	52 (Freq Index -26 to +26)
Data Subcarriers	48
Pilot Subcarriers*	4 (-21, -7, +7, +21) *Always BPSK
DC Subcarrier	Null (0 subcarrier)

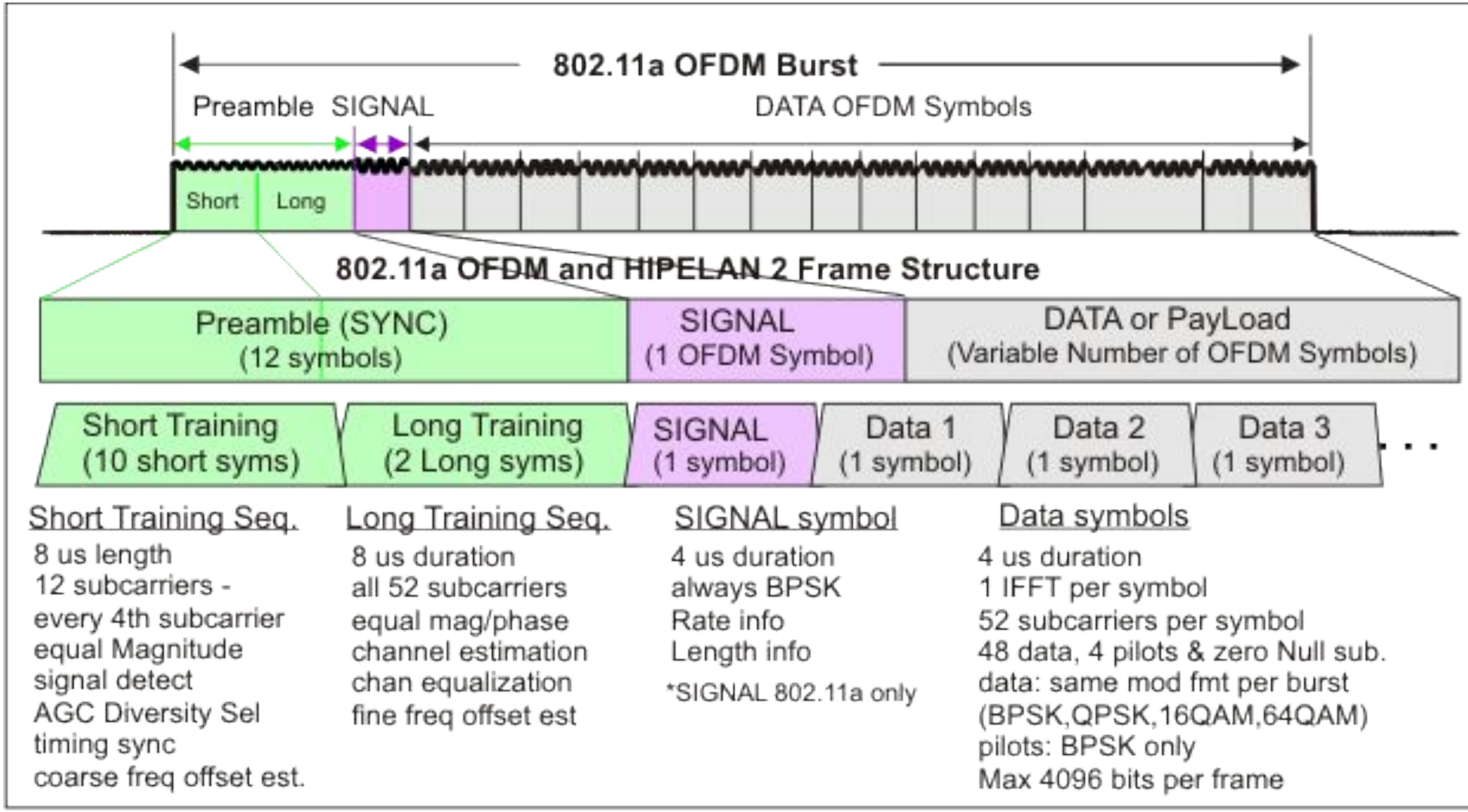


One Subcarrier = 1 constellation point
1 OFDM symbol = 52 subcarriers
1 OFDM Burst = one or more OFDM symbols

802.11a OFDM Physical Parameters

- OFDM transmission scheme
- Symbols are encoded in subcarriers

Real-World Example: PHY layer of Wi-Fi



802.11a and HIPERLAN/2 Frame Structure

Source: Keysight.com

Real-World Example: PHY layer of Wi-Fi

802.11a Timing Related Parameters

Parameter	Value
Total subcarriers N_{ST}	52
Data subcarriers N_{SD}	48
Pilot subcarriers N_{SP}	4 (subcarriers -21, 7, 7, 21)
Subcarrier Frequency Spacing F_{SP}	312.5 KHz (20MHz/64)
Symbol Interval Time T_{SYM}	4 us (T_{GI} +T_{FFT})
Data Interval Time T_{DATA}	3.2 us (1/F_{SP})
Guard Interval (GI) Time T_{GI}	0.8 us (T_{FFT}/4)
IFFT/FFT Period T_{FFT}	3.2 us (1/F_{SP})
SIGNAL Symbol TIme T_{SIGNAL}	4 us (T_{GI} +T_{FFT}
Preamble $T_{PREAMBLE}$	16 us (T_{SHORT} +T_{LONG})
Short Training Sequence T_{SHORT}	8 us (10xT_{FFT}/4)
Long Training Sequence T_{LONG}	8 us (T_{GI2} + 2xT_{FFT})
Training symbol GI T_{GI2}	1.6 us (T_{FFT}/2)
FFT sample size	64 point

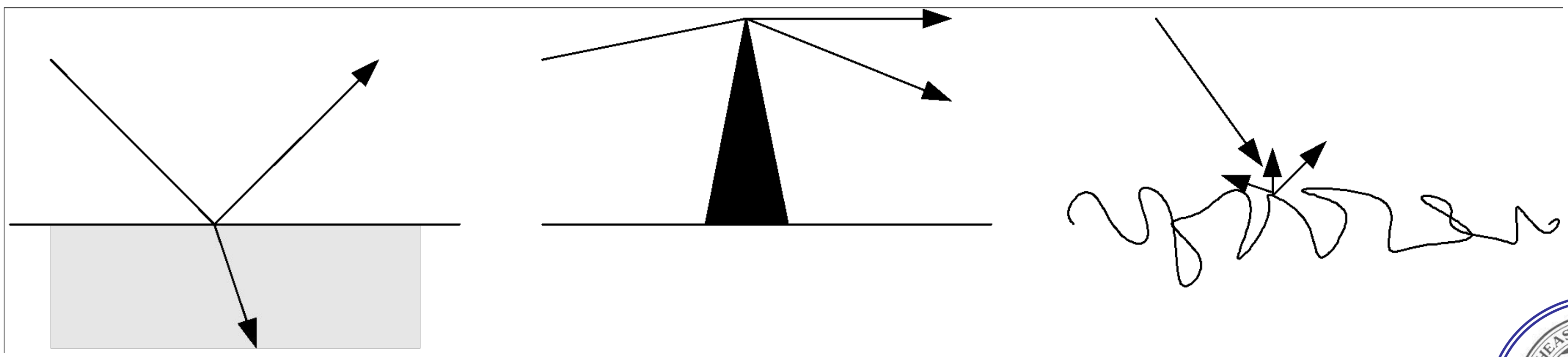
Source: Keysight.com

Yes, but why do we need all this?

Attenuation and Distortion

- Wireless propagation causes two main effects
  - Distortion – Waveform of received signal is different from transmitted
  - Attenuation – Energy is distributed to larger areas with increasing distance

- Sources of distortion
  - Reflection/refraction – bounce of a surface; enter material
  - Diffraction – start "new wave" from a sharp edge
  - Scattering – multiple reflections at rough surfaces (e.g., tree leaves)
  - Doppler fading – shift in frequencies (loss of center)



Attenuation: Path Loss

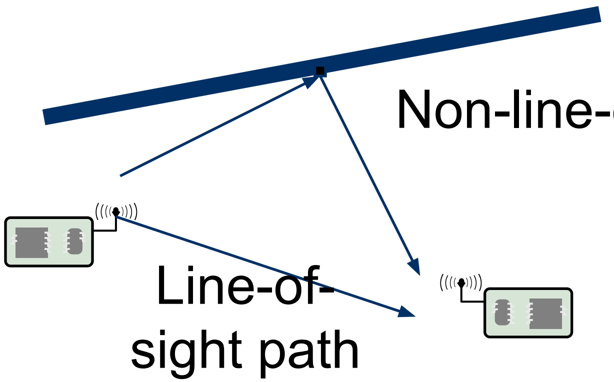
- Captured by ***Friis free-space equation***
  - Describes signal strength at distance *d* relative to some reference distance $d_0 < d$ for which strength is known
  - d_0 is ***far-field distance,*** depends on antenna technology

P_r = Power at the receiving antenna
P_t = Output power of transmitting antenna
G_t = Gain of the transmitting antenna
G_r = Gain of the receiving antenna
λ = Wavelength

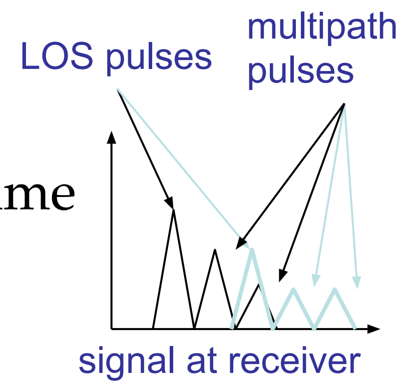
$$
\begin{aligned}
P_{\text{recv}}(d) &= \frac{P_{\text{tx}} \cdot G_t \cdot G_r \cdot \lambda^2}{(4\pi)^2 \cdot d^2 \cdot L} \\
&= \frac{P_{\text{tx}} \cdot G_t \cdot G_r \cdot \lambda^2}{(4\pi)^2 \cdot d_0^2 \cdot L} \cdot \left(\frac{d_0}{d}\right)^2 = P_{\text{recv}}(d_0) \cdot \left(\frac{d_0}{d}\right)^2
\end{aligned}
$$

Distortion Effects: Non-line-of-sight Paths

- ➢ Because of reflection, scattering, …, radio communication is not limited to direct line of sight communication
  - – Effects depend strongly on frequency, thus different behavior at higher frequencies

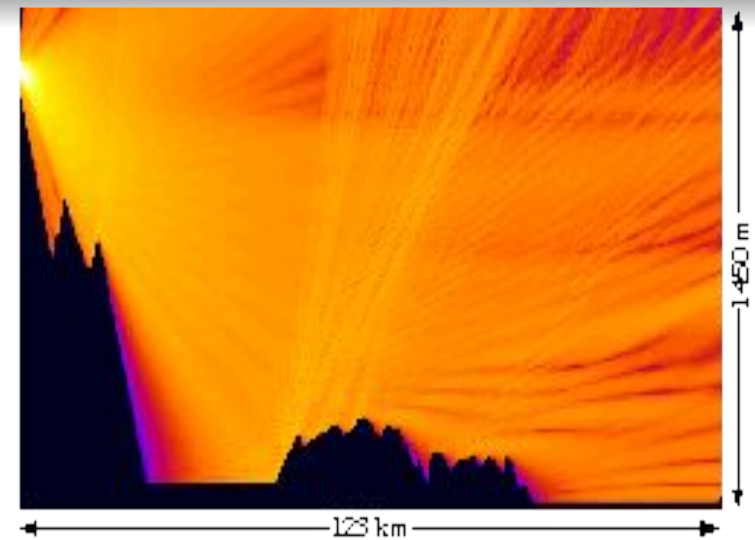


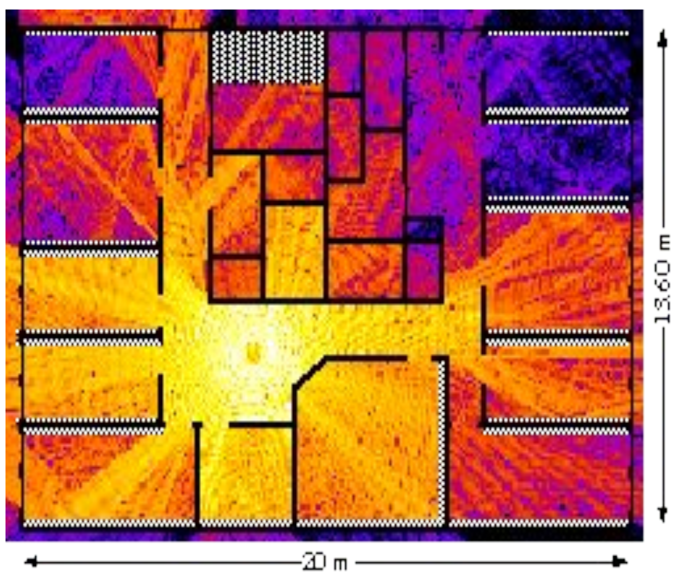
- ➢ Different paths have different lengths = propagation time
  - – Results in ***delay spread*** of the wireless channel
  - – Closely related to frequency-selective fading properties of the channel
  - – With movement: ***fast fading***

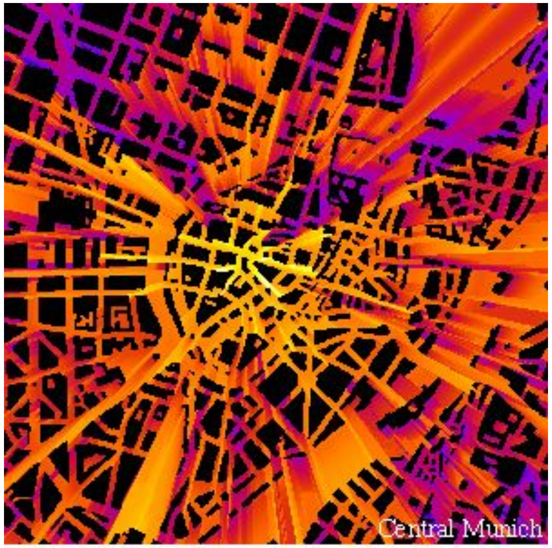


Wireless signal strength in a multi-path environment

- Brighter color = stronger signal
- Obviously, simple (quadratic) free space attenuation formula is not sufficient to capture these effects







© Jochen Schiller, FU Berlin

Attenuation

- To take into account stronger attenuation we use a larger exponent $2 < \gamma < 5$
  - γ is the ***path-loss exponent***

$$P_{\mathsf{recv}}(d) = P_{\mathsf{recv}}(d_0) \cdot \left(\frac{d_0}{d}\right)^{\gamma}$$

  - Rewrite in logarithmic form (in dB):

$$\mathsf{PL}(d)[\mathsf{dB}] = \mathsf{PL}(d_0)[\mathsf{dB}] + 10\gamma \log_{10}\left(\frac{d}{d_0}\right)$$

- Take obstacles into account by a random variation
  - Add a Gaussian random variable with 0 mean, variance σ^2 to dB representation
  - Equivalent to multiplying with a lognormal distributed r.v. in metric units - ***lognormal fading***

$$\mathsf{PL}(d)[\mathsf{dB}] = \mathsf{PL}(d_0)[\mathsf{dB}] + 10\gamma \log_{10}\left(\frac{d}{d_0}\right) + X_\sigma[\mathsf{dB}]$$

Noise and Interference

- Received signal is further affected by
  - *Noise*
    - Temperature-dependent effects in receiver electronics
    - Typical model: an additive Gaussian variable, mean 0, no correlation in time
  - *Interference*
    - Co-channel interference: another sender uses the same spectrum
    - Adjacent-channel interference: another sender uses some other part of the radio spectrum, but receiver filters are not good enough to fully suppress it
- Effect: Received signal is distorted by channel, corrupted by noise and interference
  - What is the result on the received bits?

Symbols and Bit Errors

- Extracting symbols out of a distorted/corrupted waveform causes errors
- Depends on strength of the received signal compared to the corruption
- ***Signal to Interference plus noise ratio (SINR)***

$$\mathsf{SINR} = 10 \log_{10} \left(\frac{P_{\mathsf{recv}}}{N_0 + \sum_{i=1}^{k} I_i} \right)$$

- SINR allows to compute ***bit error rate*** (***BER***) for a given modulation
  - Also depends on data rate (# bits/symbol) of modulation
  - E.g., for simple BPSK, data rate corresponding to bandwidth:

$$\mathsf{BER}(\mathsf{SINR}) = 0.5 e^{-\frac{E_b}{N_0}}$$

$$E_b/N_0 = \mathsf{SINR} \cdot \frac{1}{R}$$

Channel Models - Analog

- Stochastically capture the behavior of a wireless channel
- In networking research, often used to model SINR
- Signal models
  - Transmission power and attenuation are constant
  - ***Additive White Gaussian Noise*** model
  - No line-of-sight path, many indirect paths:
    - Amplitude of resulting signal has a ***Rayleigh*** distribution (***Rayleigh fading)*** - used to model mobile scenarios
  - One dominant line-of-sight plus many indirect paths
    - Signal has a ***Rice*** distribution (***Rice fading***)

WSN-specific channel models

- Typical WSN properties
  - Small transmission range
  - Implies small delay spread (nanoseconds, compared to micro/milliseconds for symbol duration)
  - Low to negligible inter-symbol interference
    - Coherence bandwidth often > 50 MHz
- Some example measurements
  - γ path loss exponent
  - Shadowing variance σ^2
  - Reference path loss at 1 m

Location	Average of γ	Average of σ^2 [dB]	Range of PL(1m)[dB]
Engineering Building	1.9	5.7	$[-50.5, -39.0]$
Apartment Hallway	2.0	8.0	$[-38.2, -35.0]$
Parking Structure	3.0	7.9	$[-36.0, -32.7]$
One-sided Corridor	1.9	8.0	$[-44.2, -33.5]$
One-sided patio	3.2	3.7	$[-39.0, -34.2]$
Concrete canyon	2.7	10.2	$[-48.7, -44.0]$
Plant fence	4.9	9.4	$[-38.2, -34.5]$
Small boulders	3.5	12.8	$[-41.5, -37.2]$
Sandy flat beach	4.2	4.0	$[-40.8, -37.5]$
Dense bamboo	5.0	11.6	$[-38.2, -35.2]$
Dry tall underbrush	3.6	8.4	$[-36.4, -33.2]$

Wireless channel quality – summary

- Wireless channels are substantially worse than wired channels
  - In throughput, bit error characteristics, energy consumption, …
- Wireless channels are extremely diverse
  - There is no such thing as **THE typical wireless channel**
- Various schemes for quality improvement exist
  - Some of them geared towards high-performance wireless communication – not necessarily suitable for WSN, ok for MANET
    - Diversity, equalization, …
  - Some of them general-purpose (ARQ, FEC)
  - **Energy issues need to be taken into account!**

Some transceiver design considerations

- Strive for **good power efficiency at low transmission power**
  - Some amplifiers are optimized for efficiency at high output power
  - To radiate 1 mW, typical designs need 30-100 mW to operate the transmitter
    - WSN nodes: **20 mW** (mica motes)
  - Receiver can use as much or more power as transmitter at these power levels
    - Sleep state is important
- Startup energy/time penalty can be high
  - Examples take 0.5 ms and 60 mW to wake up
- Exploit communication/computation tradeoffs
  - Might pay off to invest in rather complicated coding/compression schemes

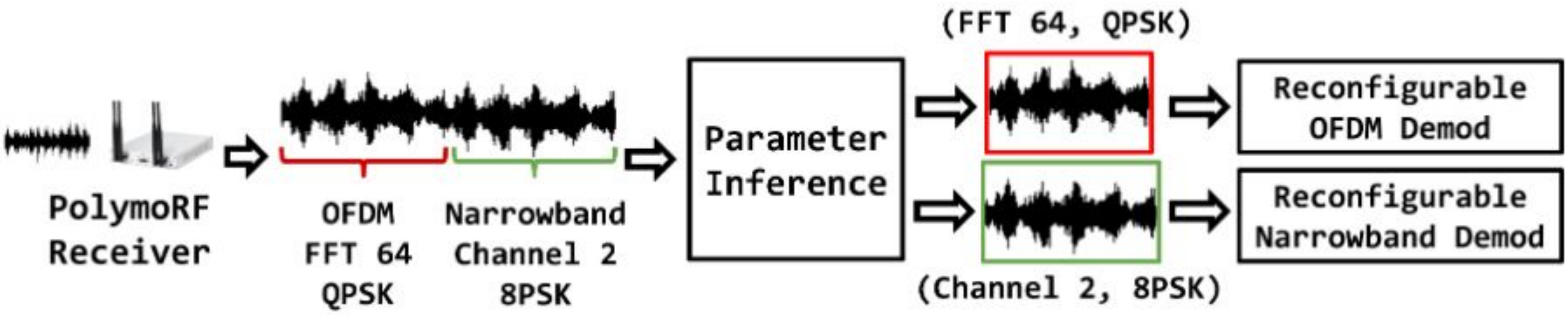
Choice of modulation

- One exemplary design point: **which modulation to use**?
  - **Consider:** required data rate, available symbol rate, implementation complexity, required BER, channel characteristics, …
  - **Tradeoffs:** the faster one sends, the longer one can sleep
    - However, power consumption can depend on modulation scheme
  - **Tradeoffs:** symbol rate (high?) versus data rate (low)
    - Use m-ary transmission to get a transmission over with ASAP
    - But: startup costs can easily void any time saving effects

- Adapt modulation choice to operation conditions
  - Akin to dynamic voltage scaling, introduce ***Dynamic Modulation Scaling***

Automatic Modulation Recognition

F. Restuccia and T. Melodia, *"PolymoRF: Polymorphic Wireless Receivers Through Physical-Layer Deep Learning,"* **Proceedings of ACM International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (ACM MobiHoc)**, October 2020.



Why do you think we need AMR?

Think-Share!

Summary

- Wireless radio communication introduces many uncertainties and vagaries into a communication system
- Handling the unavoidable errors will be a major challenge for the communication protocols
- Dealing with limited bandwidth in an energy-efficient manner is the main challenge
- MANET and WSN are pretty similar here
  - Main differences are in required data rates and resulting transceiver complexities (higher bandwidth, spread spectrum techniques)

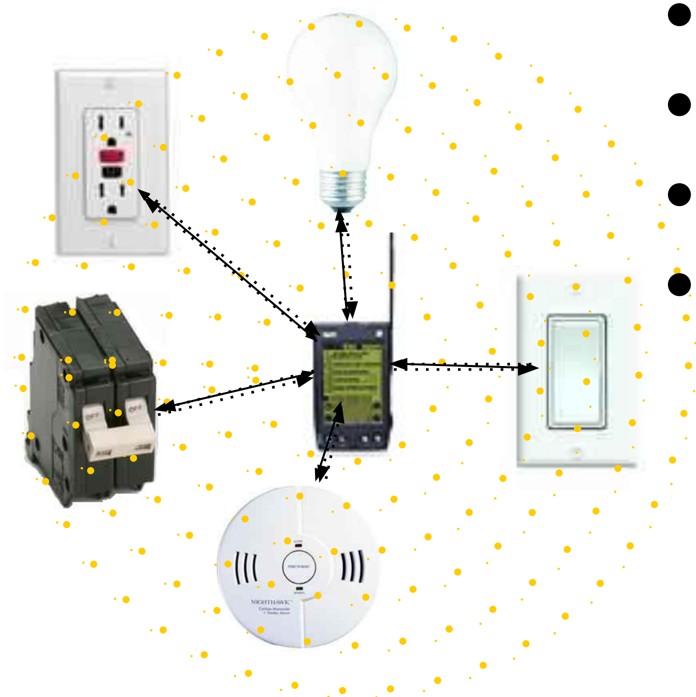
Institute for the Wireless Internet of Things at Northeastern University

EECE 5155
Wireless Sensor Networks
(and The Internet of Things)

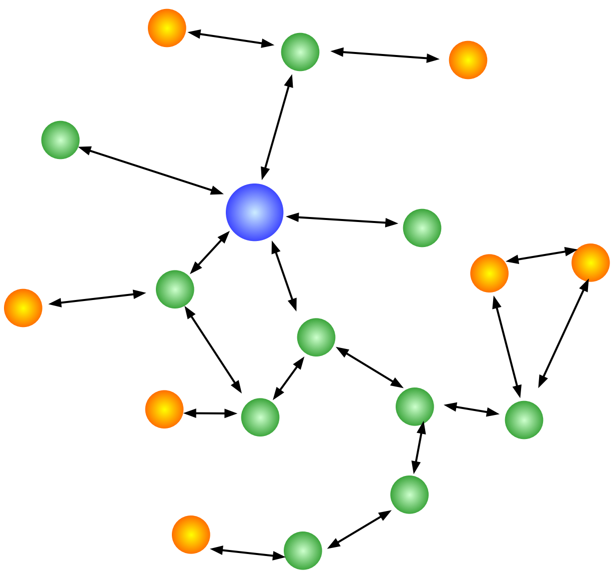
Prof. Francesco Restuccia
Email: f.restuccia@northeastern.edu

IEEE 802.15.4
PHY LAYER

IEEE 802.15.4 Application Space



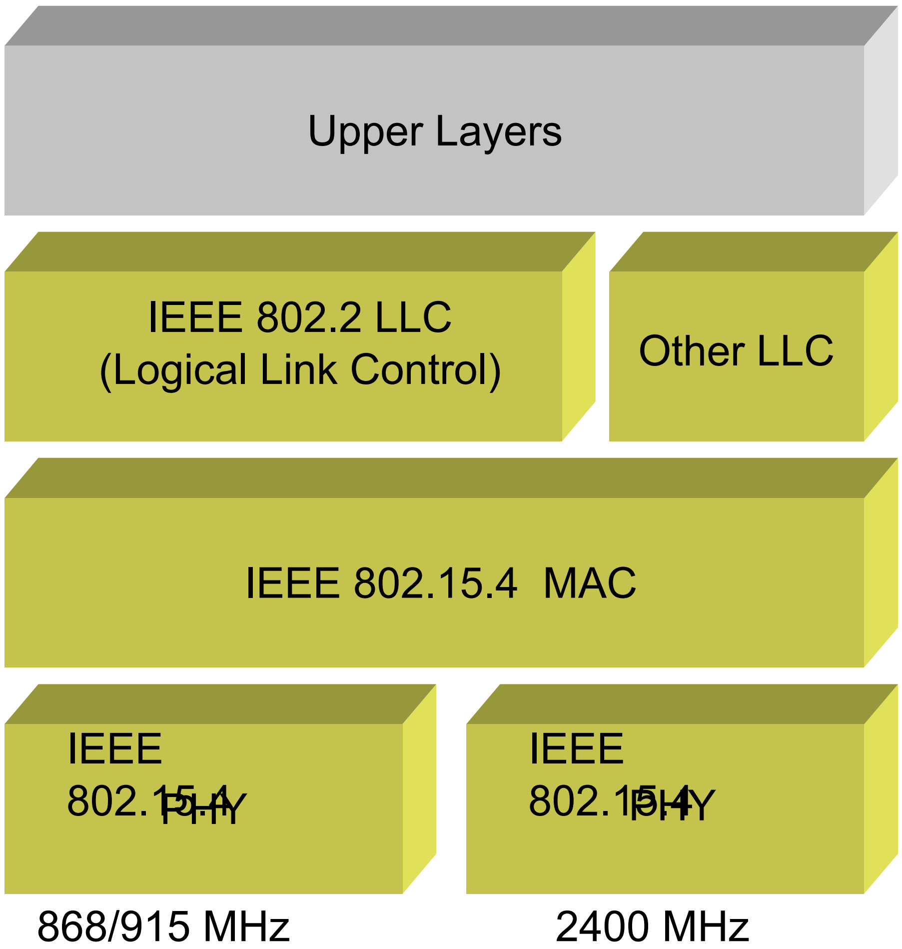
- Home Networking
- Automotive Networks
- Industrial Networks
- Interactive Toys
- Remote Metering



Differences between IEEE 802.15.4 & ZigBee

- IEEE 802.15.4
  - PHYsical Layer (PHY)
    - Radio portion, transmitter and receiver
  - Media Access Control (MAC) Layer
    - Radio controller, data to next device
- ZigBee
  - Network Layer
  - Application Support Layer

IEEE 802.15.4 Architecture



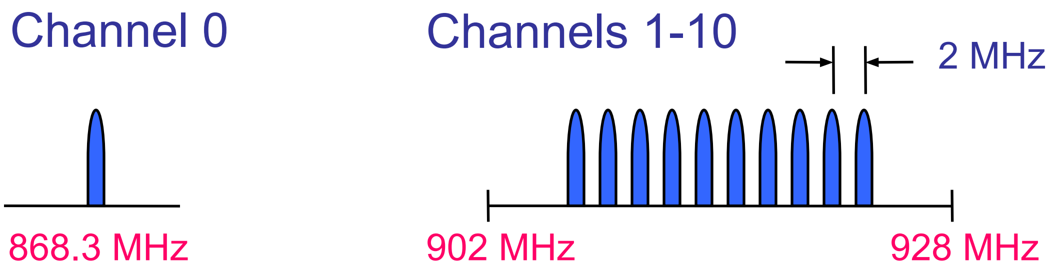
802.15.4 General Characteristics

- ➢ Data rates of 250 kbit/s, 40 kbit/s and 20 kbit/s
- ➢ Star or Peer-to-Peer operation
- ➢ Support for low latency devices
- ➢ CSMA-CA channel access **(Homework 1!)**
- ➢ Dynamic device addressing
- ➢ Fully handshaked protocol for transfer reliability
- ➢ **Low power consumption**
- ➢ Frequency Bands of Operation, either:
  - ✓ 16 channels in the 2.4GHz ISM band;
  - ✓ Or 10 channels in the 915MHz ISM band
    and 1 channel in the European 868MHz band.

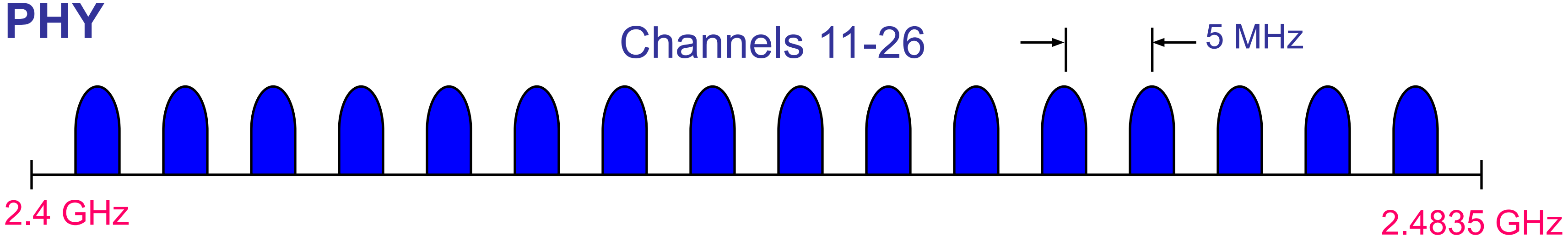
IEEE 802.15.4 PHY Overview

Operating Frequency Bands

868MHz / 915MHz PHY



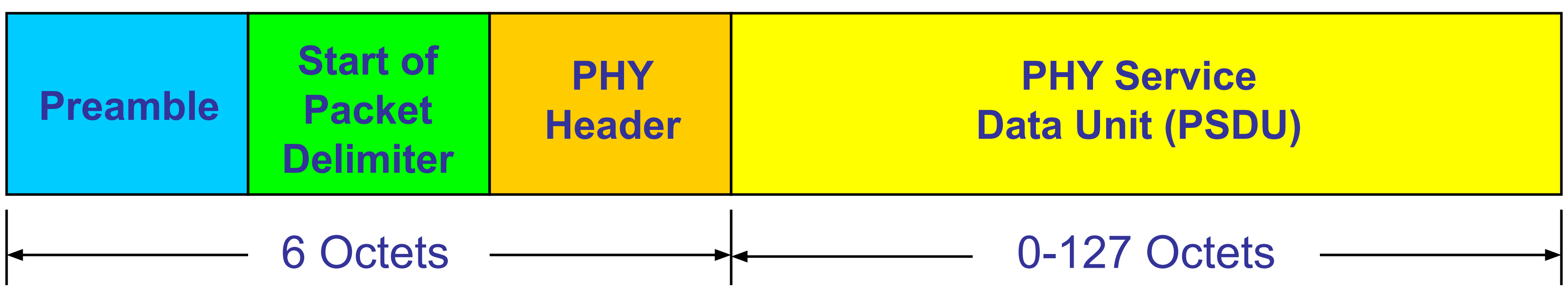
2.4 GHz PHY



Packet Structure

PHY Packet Fields

- Preamble (32 bits) – synchronization
- Start of Packet Delimiter (8 bits)
- PHY Header (8 bits) – PSDU length
- PSDU (0 to 1016 bits) – Data field



IEEE 802.15.4 PHY Overview

Modulation/Spreading

- **2.4 GHz PHY**
  - 250 kb/s (4 bits/symbol, 62.5 ksymbols/s)
  - Data modulation is 16-ary orthogonal modulation
  - 16 symbols are orthogonal set of 32-chip PN codes
  - Chip modulation is O-QPSK at 2.0 Mchips/s

- **868MHz/915MHz PHY**
  - Symbol Rate
    - 868 MHz Band: 20 kb/s (1 bit/symbol, 20 ksymbols/s)
    - 915 MHz Band: 40 kb/s (1 bit/symbol, 40 ksymbols/s)
  - Data modulation is BPSK
  - Spreading code is a 15-chip m-sequence
  - Chip modulation is BPSK at
    - 868 MHz Band: 300 kchips/s
    - 915 MHz Band: 600 kchips/s

IEEE 802.15.4 PHY Overview

Common Parameters

Transmit Power
- Capable of at least .5 mW

Transmit Center Frequency Tolerance
- ± 40 ppm

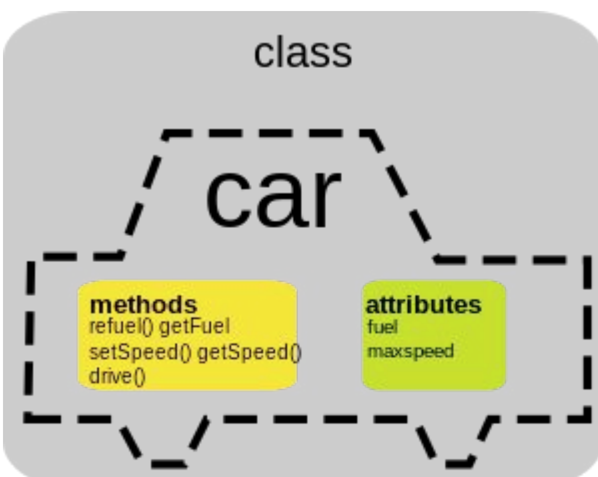
Receiver Sensitivity (Packet Error Rate <1%)
- ≤-85 dBm @ 2.4 GHz band
- ≤-92 dBm @ 868/915 MHz band

Rx Signal Strength Indication Measurements
- Packet strength indication
- Clear channel assessment
- Dynamic channel selection

Principles of Object-Oriented Programming using C++

What is OOP?

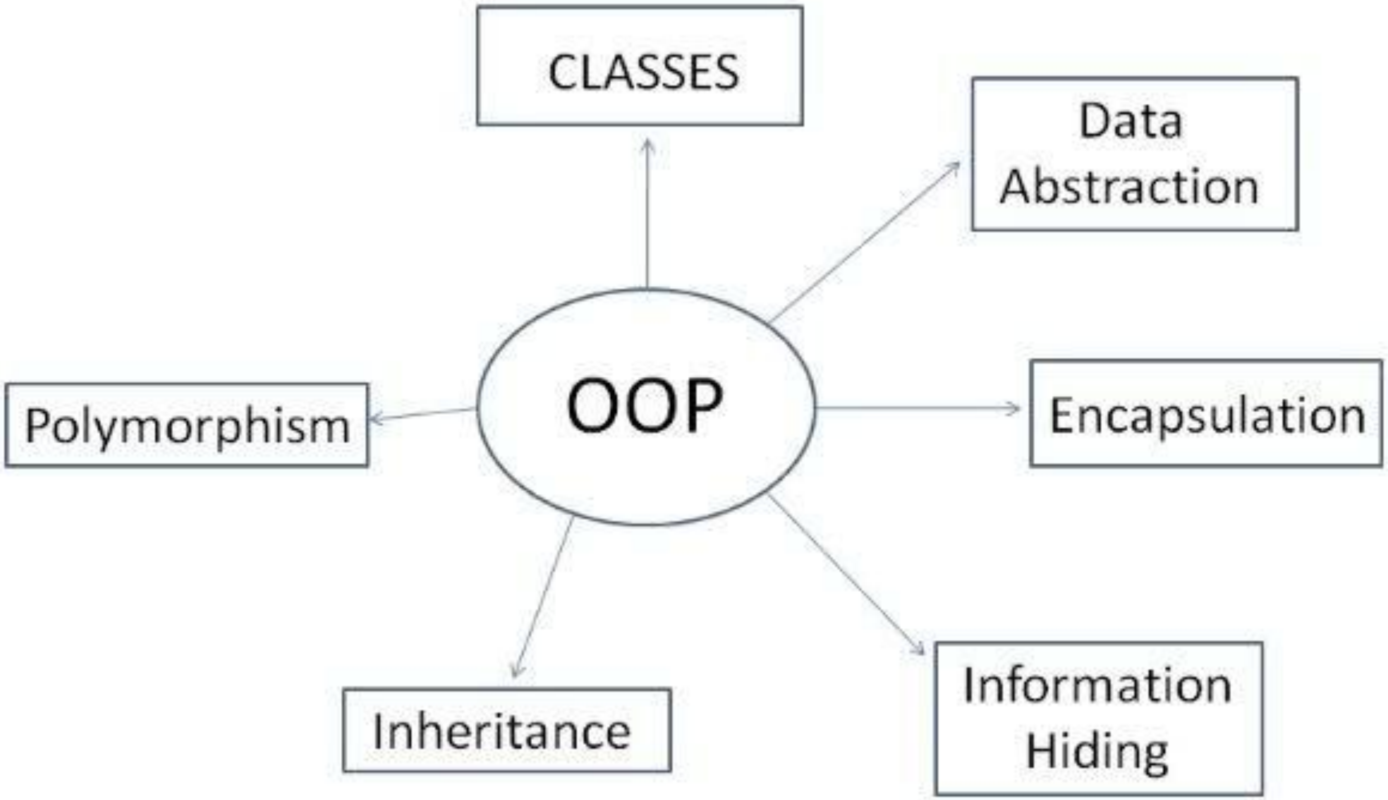
- The prime purpose of C++ programming was to add **object orientation** to the C programming language
- The core of the pure object-oriented programming is to create an **object**, in code, that has certain **properties** and **methods**.
- While designing C++ modules, we try to see whole world in the form of objects (EXAMPLES?)



keyword   user-defined name

```
class ClassName
{  Access specifier:          //can be private,public or protected
   Data members;              // Variables to be used
   Member Functions() { }     //Methods to access data members
};                            // Class name ends with a semicolon
```

Basic Concepts of OOP



Classes in C++

```
#include <iostream>

using namespace std;

class Box {
   public:
      double length;   // Length of a box
      double breadth;  // Breadth of a box
      double height;   // Height of a box
};

int main() {
   Box Box1;        // Declare Box1 of type Box
   Box Box2;        // Declare Box2 of type Box
   double volume = 0.0;     // Store the volume of a box here

   // box 1 specification
   Box1.height = 5.0;
   Box1.length = 6.0;
   Box1.breadth = 7.0;

   // box 2 specification
   Box2.height = 10.0;
   Box2.length = 12.0;
   Box2.breadth = 13.0;

   // volume of box 1
   volume = Box1.height * Box1.length * Box1.breadth;
   cout << "Volume of Box1 : " << volume <<endl;

   // volume of box 2
   volume = Box2.height * Box2.length * Box2.breadth;
   cout << "Volume of Box2 : " << volume <<endl;
   return 0;
}
```

```
Volume of Box1 : 210
Volume of Box2 : 1560
```

Can you see the problem in this piece of code?

Abstraction / Information Hiding

- Providing only **essential information** to the outside world and **hiding their background details**
- For example, a database system hides certain details of how data is stored and created and maintained
- Similarly, C++ classes provides different methods to the outside world **without giving internal details**

- WHY?

- **No need to know!** (e.g., a user **doesn't care** about the internal structure of a car)
- Internal implementation of functionalities **might change**!

Abstraction / Information Hiding (2)

```
#include <iostream>
using namespace std;

class Adder {
   public:
      // constructor
      Adder(int i = 0) {
         total = i;
      }

      // interface to outside world
      void addNum(int number) {
         total += number;
      }

      // interface to outside world
      int getTotal() {
         return total;
      };

   private:
      // hidden data from outside world
      int total;
};

int main() {
   Adder a;

   a.addNum(10);
   a.addNum(20);
   a.addNum(30);

   cout << "Total " << a.getTotal() <<endl;
   return 0;
}
```

```
Total 60
```

- The public members *addNum()* and *getTotal()* are the interfaces to the outside world and a user needs to know them to use the class
- The private member *total* is something that the user **doesn't need to know about**, but is needed for the class to operate properly

Encapsulation

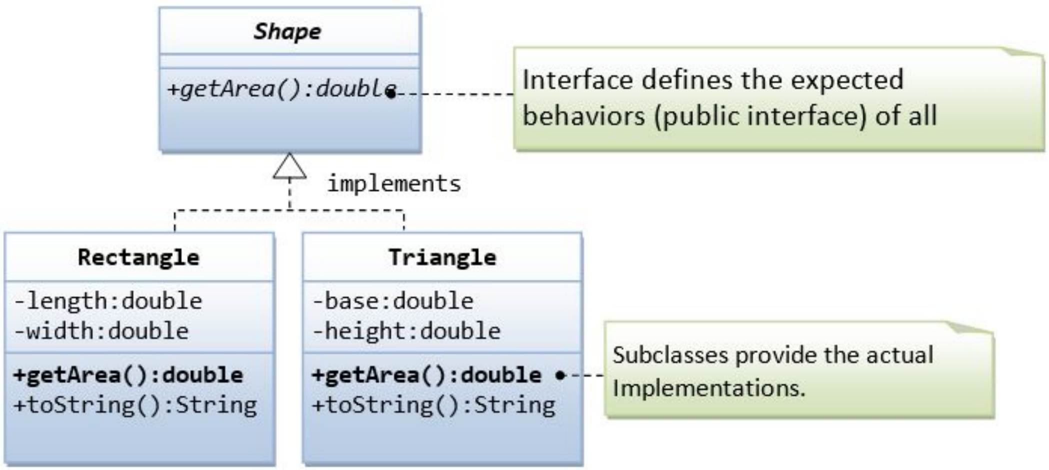
- Encapsulation is placing the **data and the functions that work on that data in the same place**
- We don't access data directly, but only through **methods**

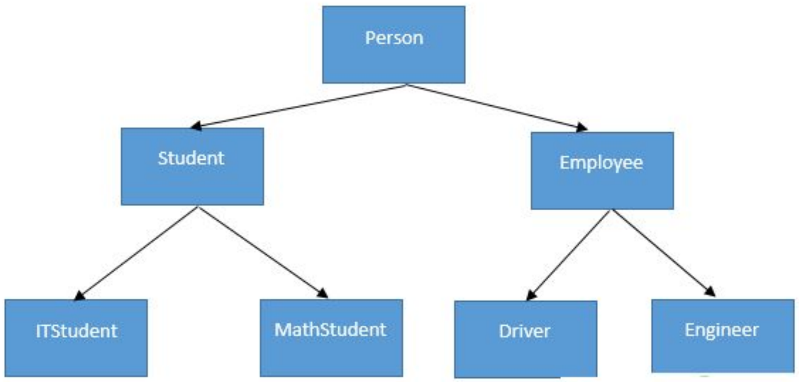
- WHY?

- More order and control (which implies, **less bugs!**)

Inheritance

- Mechanism in which one object acquires **all the properties and behaviors** of the parent object





- Create classes that are built upon existing classes
- Maintain the same behavior w/ different implementation (i.e., an **interface**),
- **Reuse code** and to independently **extend** original software via **public** classes

Inheritance (2)

```
#include <iostream>

using namespace std;

// Base class
class Shape {
   public:
      void setWidth(int w) {
         width = w;
      }
      void setHeight(int h) {
         height = h;
      }

   protected:
      int width;
      int height;
};

// Derived class
class Rectangle: public Shape {
   public:
      int getArea() {
         return (width * height);
      }
};

int main(void) {
   Rectangle Rect;

   Rect.setWidth(5);
   Rect.setHeight(7);

   // Print the area of the object.
   cout << "Total area: " << Rect.getArea() << endl;

   return 0;
}
```

```
Total area: 35
```

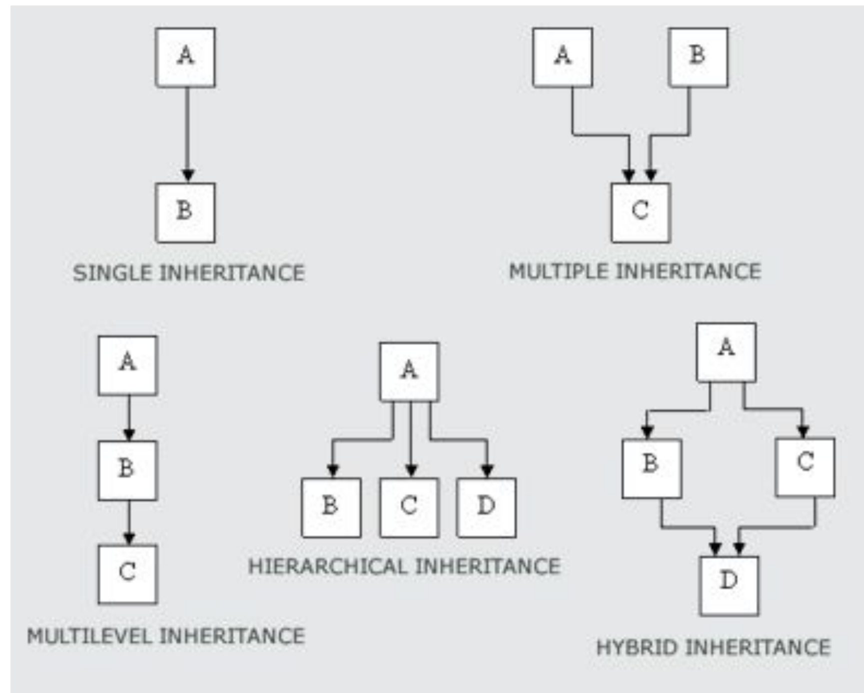
Inheritance (3)

Base class visibility	Derived class visibility		
	Public derivation	**Private derivation**	**Protected derivation**
• Private → • Protected → • Public →	• Not inherited • Protected • Public	• Not inherited • Private • Private	• Not inherited • Protected • Protected

```
class base
{
    private:
        int x;
    protected:
        int y;
    public:
        int z;
};

class public_derived : public base
{
    ... .. ...
};

class protected_derived : protected base
{
    ... .. ...
};

class private_derived : private base
{
    ... .. ...
};

int main()
{
   public_derived b;
   protected_derived c;
   private_derived d;
}
```

Accessible

Accessible



```cpp
#include <iostream>
using namespace std;

class Shape {
   protected:
      int width, height;

   public:
      Shape( int a = 0, int b = 0){
         width = a;
         height = b;
      }
      int area() {
         cout << "Parent class area :" <<endl;
         return 0;
      }
};
class Rectangle: public Shape {
   public:
      Rectangle( int a = 0, int b = 0):Shape(a, b) { }

      int area () {
         cout << "Rectangle class area :" <<endl;
         return (width * height);
      }
};

class Triangle: public Shape {
   public:
      Triangle( int a = 0, int b = 0):Shape(a, b) { }

      int area () {
         cout << "Triangle class area :" <<endl;
         return (width * height / 2);
      }
};
```

```cpp
// Main function for the program
int main() {
   Shape *shape;
   Rectangle rec(10,7);
   Triangle  tri(10,5);

   // store the address of Rectangle
   shape = &rec;

   // call rectangle area.
   shape->area();

   // store the address of Triangle
   shape = &tri;

   // call triangle area.
   shape->area();

   return 0;
}
```

```
Parent class area :
Parent class area :
```

Static vs Dynamic Binding

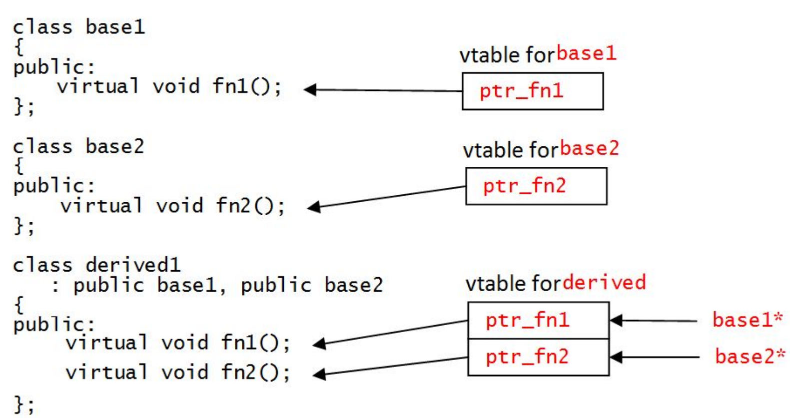
- Call of the function *area()* is being **set once** by the **compiler** as the version defined in the base class
- This is called **early (or static) binding** as the *area()* function is set at compile time
- We need **late (or dynamic) binding,** so that *area()* is decided at **execution time!**

Let’s Fix the Problem...

```
class Shape {
   protected:
      int width, height;

   public:
      Shape( int a = 0, int b = 0) {
         width = a;
         height = b;
      }
      virtual int area() {
         cout << "Parent class area :" <<endl;
         return 0;
      }
};
```

```
Rectangle class area
Triangle class area
```

- Defining in a base class a virtual function, with another version in a derived class, signals to the compiler that we **don't want static binding** for this function
- What we do want is the <u>selection of the function to be called at any given point in the program to be based on the kind of object for which it is called</u>
- This is called **dynamic linkage**, or **late binding**



Pure Virtual Functions

- Defined in a derived class to **suit the objects of that class**
- **No meaningful definition** can be given for the function in the base class

```
class Shape {
   protected:
      int width, height;

   public:
      Shape(int a = 0, int b = 0) {
         width = a;
         height = b;
      }

      // pure virtual function
      virtual int area() = 0;
};
```

- We call a class w/ pure virtual functions an **abstract class**
- We cannot create instances of abstract classes
- They provide a “skeleton” of what the object is, so every derived class has to implement and conform to the specs

Virtual Functions

+

Inheritance

=

Polymorphism!

(can somebody define it now?)

Polymorphism:

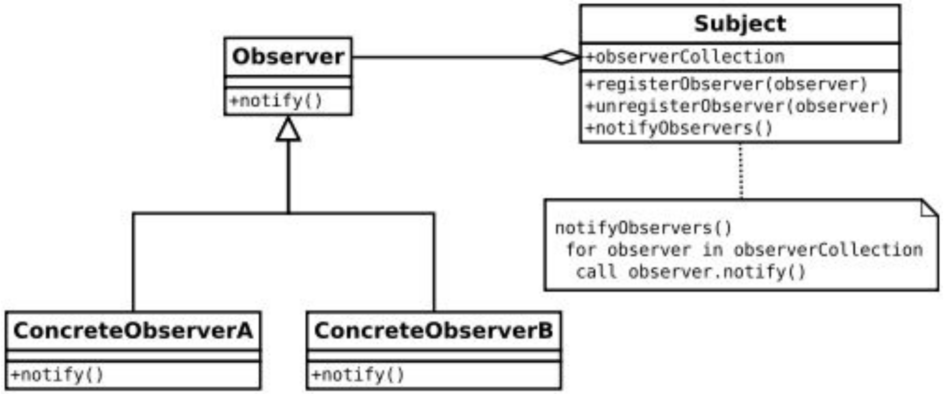
Ability of a reference variable to change behavior according to what instance variable it is holding

Examples of Design Patterns that use Polymorphism

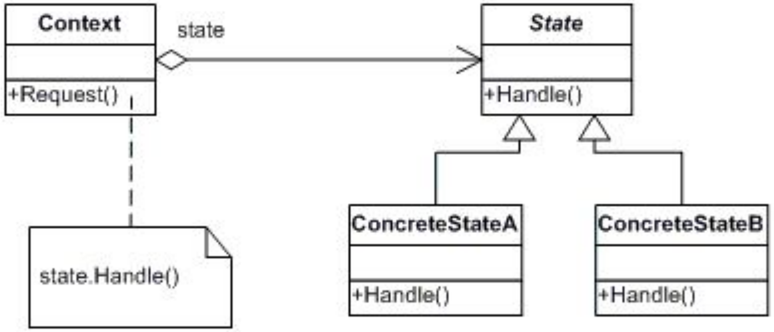
Factory Design Pattern

```
class Computer
{
public:
    virtual void Run() = 0;
    virtual void Stop() = 0;

    virtual ~Computer() {}; /* without this, you do not call Laptop or Desktop destructor in this example! */
};
class Laptop: public Computer
{
public:
    void Run() override {mHibernating = false;};
    void Stop() override {mHibernating = true;};
    virtual ~Laptop() {}; /* because we have virtual functions, we need virtual destructor */
private:
    bool mHibernating; // Whether or not the machine is hibernating
};
class Desktop: public Computer
{
public:
    void Run() override {mOn = true;};
    void Stop() override {mOn = false;};
    virtual ~Desktop() {};
private:
    bool mOn; // Whether or not the machine has been turned
};
```

```
class ComputerFactory
{
public:
    static Computer *NewComputer(const std::string &description)
    {
        if(description == "laptop")
            return new Laptop;
        if(description == "desktop")
            return new Desktop;
        return NULL;
    }
};
```



Observer Design Pattern (Code Example)



State Design Pattern (Code Example)