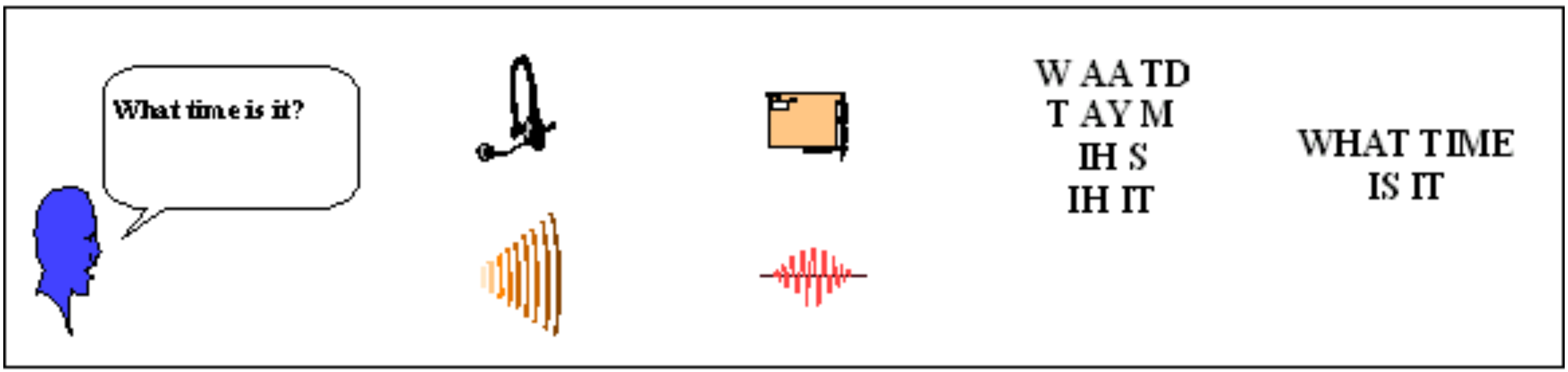
Overview of
Embedded Digital Signal Processing

Embedded Digital Signal Processing (DSP)

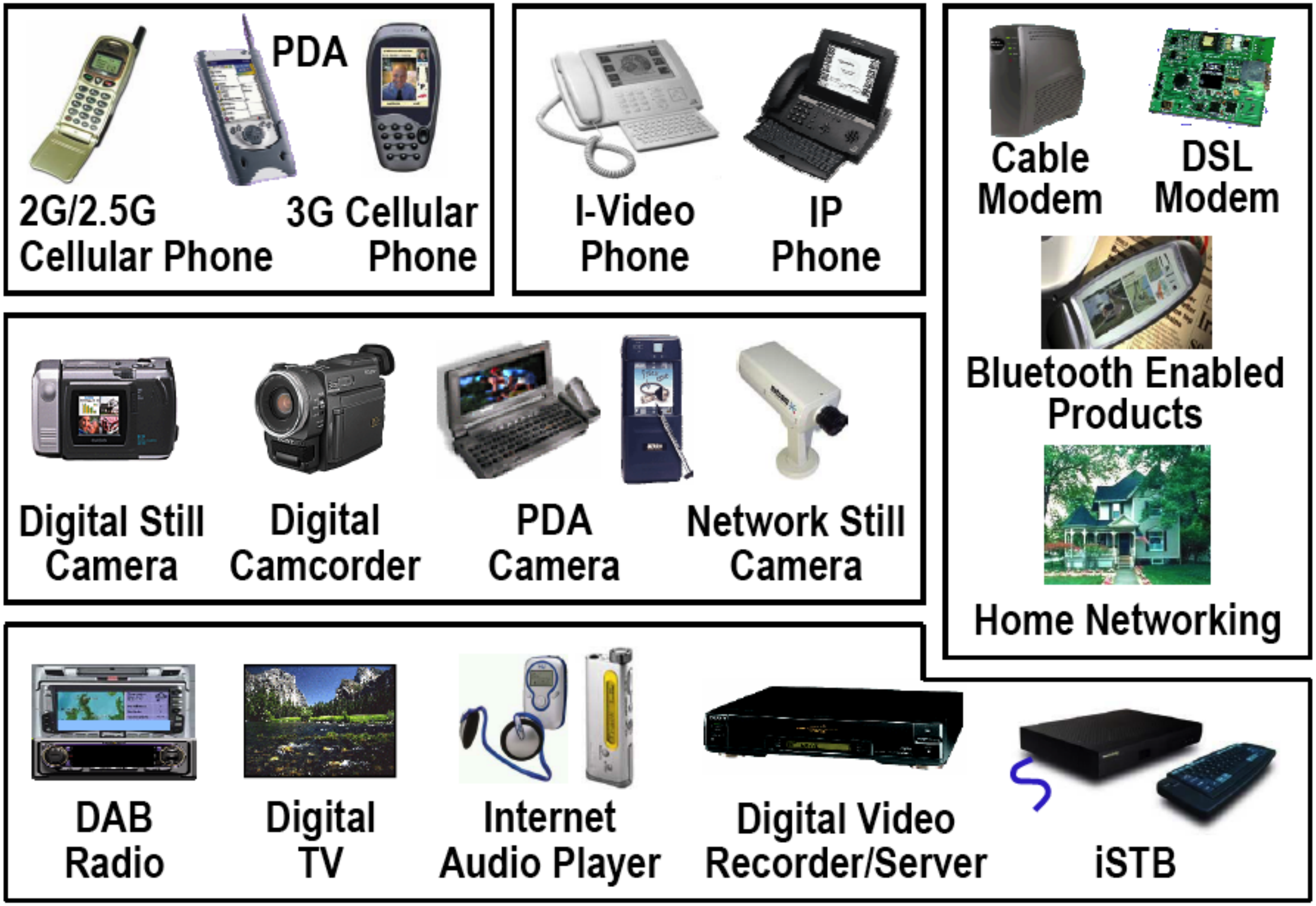
- **"Signal"**: physical quantity that carries information
- **"Processing"**: series of steps to achieve a particular end
- **"Digital"**: done by computers, microprocessors, or logic circuits
- "**Embedded**": part of a complete device (hardware), often with real-time constraints

Example: Speech Recognition using DSP



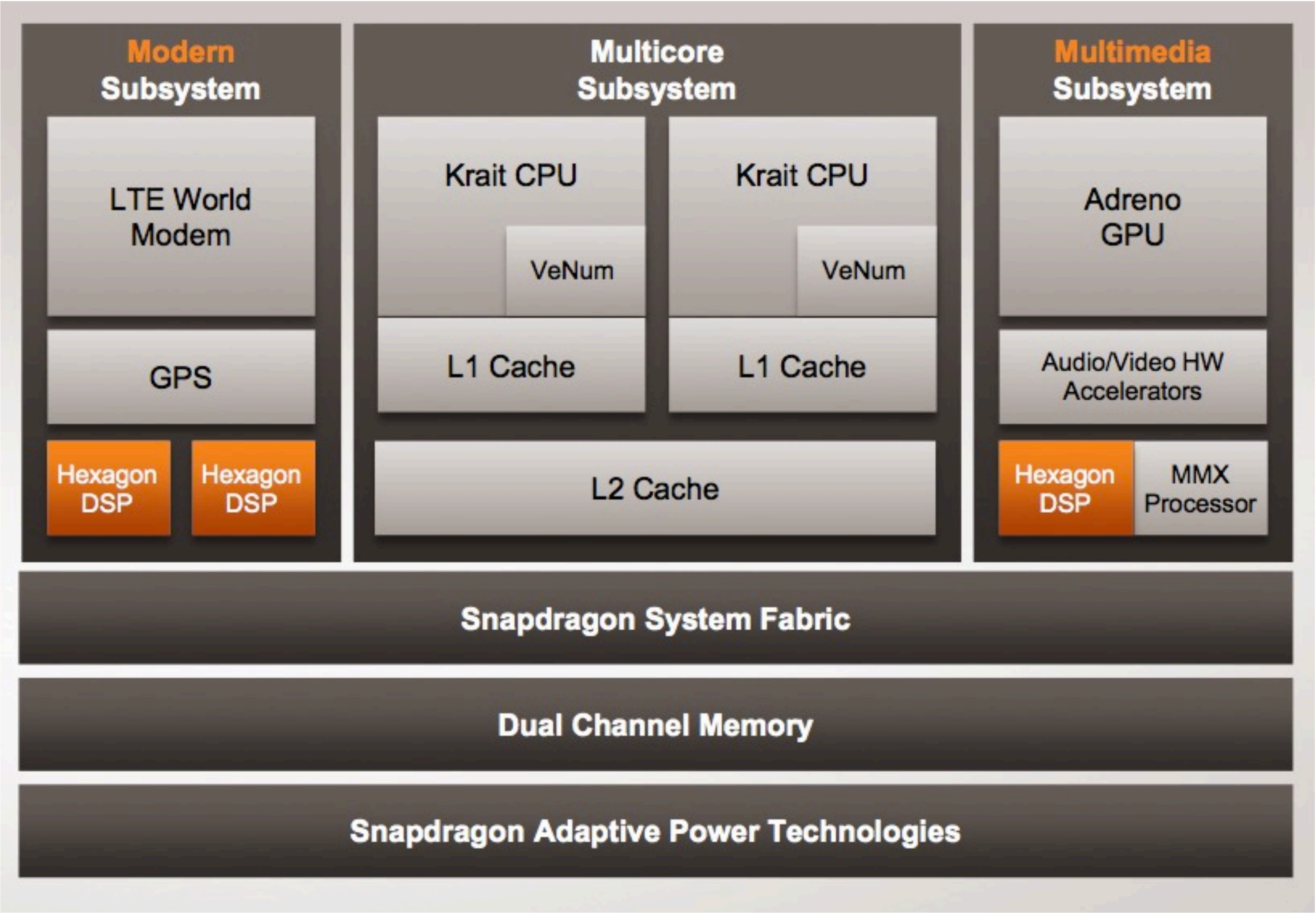
USER	MICROPHONE	SOUND CARD	SPEECH RECOGNITION ENGINE	SPEECH-AWARE APPLICATION
User speaks into the microphone.	Microphone captures sound waves and generates electrical impulses.	Sound card converts acoustical signal to digital signal.	Speech recognition engine coverts digital signal to phonemes, then words.	Application processes words as text input.

DSP Appliances



Smart Phones

Example Smartphone Chip



Digital Cameras

Original

After DSP

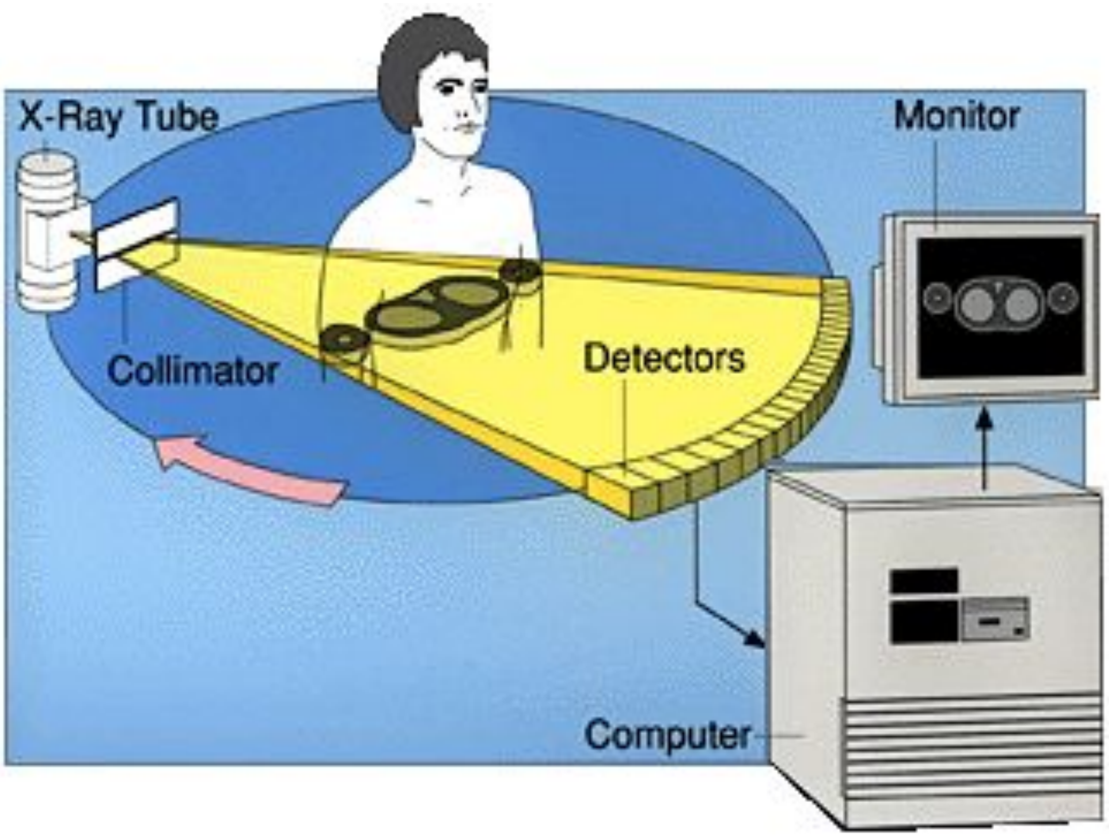



www.dxo.com

Multimedia Compression

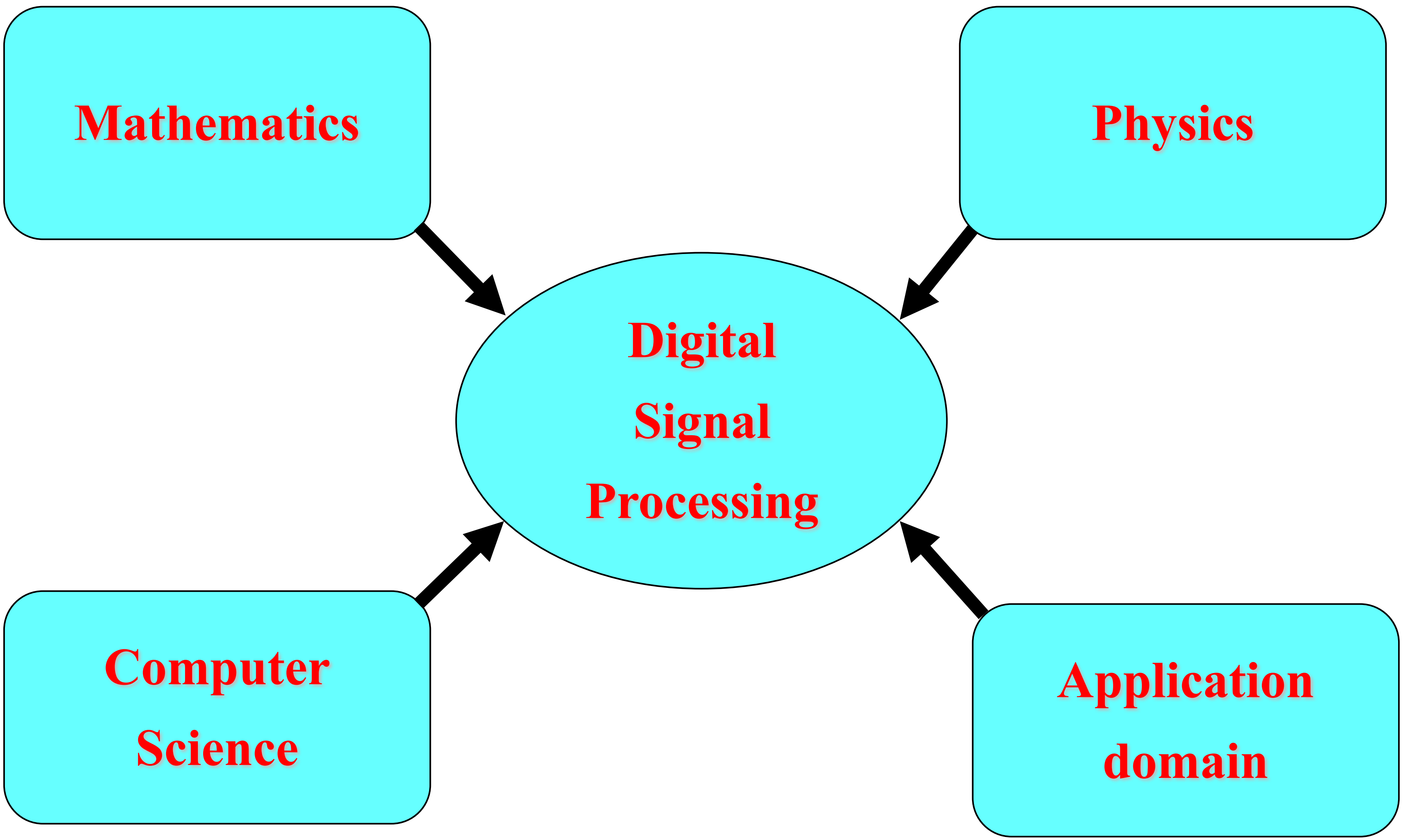
- Provide the crucial technology for:
  - WWW with multimedia content (e.g. audio, image, and video)
  - DVD
  - Digital cameras, camera phones
  - MP3, iPod

Medical Imaging:
Ultrasound (US), Computer Tomography (CT), Magnetic Resonance Imaging (MRI), …



www.imaginis.com/ct-scan

Background for DSP



Best Practices in Developing DSP Software: Systematic Debugging

- First, develop and test DSP algorithms in high-level languages (Python, MATLAB)
  - Use test signals
  - Examine intermediate signal outputs
    - Sample values
    - Signal blocks
    - Visualize signals in time, in frequency domains
  - Quantify algorithm performance (over datasets, need ground truth)
    - Signal-to-noise ratio
    - Recognition accuracy

- Then, port tested algorithms into embedded platform (Android)
- Sometimes, need to go back and refine algorithms in Python

Practical Considerations

- Reducing power is *critical* for mobile real-time devices
  - Battery drain is #1 reason for users to turn off an app
- Ways to save power
  - 16-bit fixed point, not floating point
  - Low clock speed/voltage through parallelism
  - Simple, low-power microprocessor architecture
  - Program in low-level languages
  - Use hardware accelerators, or dedicated computing units

ECE 420 Overview

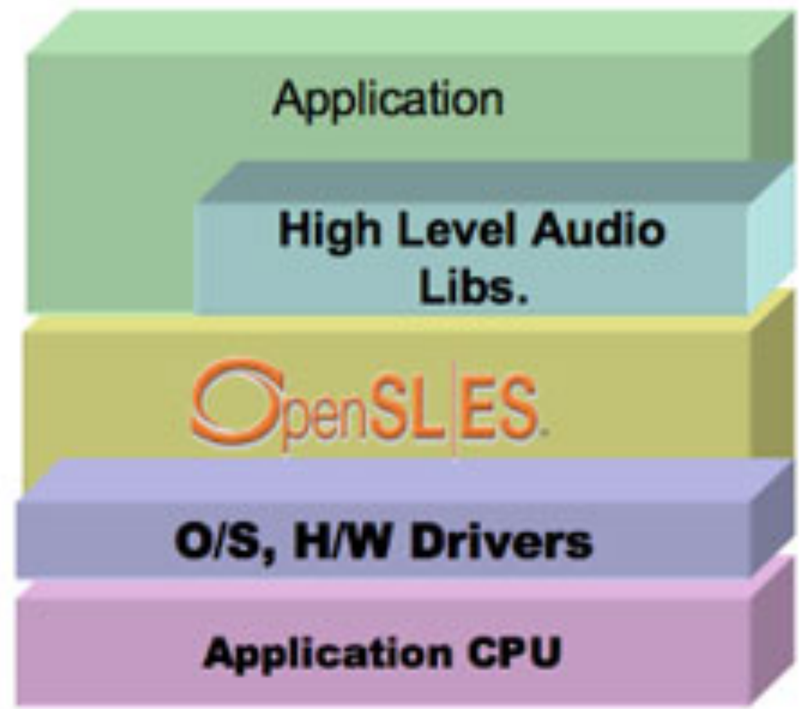
- First half: Structured Labs (7)
  - Embedded DSP development framework
    - High-level (Python) → Embedded (Android with Java/C)
  - Different signal modalities and interfaces: IMU, audio, visual
  - Basic DSP algorithms
    - Digital filtering
    - Spectral analysis
    - Auto-correlation analysis: pitch detection/correction
    - Image and multidimensional signal processing
- Second half: Individual Projects
  - Start with an Assigned Project Lab (in Python; 2 weeks)
  - Design Review → Plan for Deliverables
  - Milestones (3)
  - Final Project Demo and Presentation → Report

Next Lab: Digital Filter

Audio A/D and D/A in Android

- We will use OpenSL ES (Sound Library Embedded System)

OpenSL ES – H/W implementation



OpenSL ES – S/W implementation

Filter Design:
Mapping Analog to Digital Frequencies

If we sample an analog signal $x_a(t)$ to obtain a digital signal $x_d[n] = x_a(nT)$ using the sampling frequency $f_s = 1/T$, then their Fourier transforms are related by:

$$X_d(\omega) = \frac{1}{T} \sum_{k=-\infty}^{\infty} X_a\left(\frac{\omega - 2k\pi}{T}\right).$$

Hence, assuming no aliasing (i.e. $X_a(\Omega) = 0$ for $|\Omega| \leq \pi/T$) then an analog frequency $\Omega = 2\pi f$ (where $|\Omega| \leq \pi/T$) is mapped to a digital frequency

$$\omega = \Omega T = \frac{2\pi f}{f_s}.$$

In particular, the Nyquist frequency $f = f_s/2$ is mapped to $\omega = \pi$.

Digital Filter Implementation

Given a digital filter

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + \ldots + b_K z^{-K}}{1 + a_1 z^{-1} + \ldots + a_L z^{-L}},$$

then the filtering by $H(z)$:

$$x[n] \longrightarrow \boxed{H(z)} \longrightarrow y[n]$$

can be implemented for each n as:

$$y[n] = (b_0\, x[n] + b_1\, x[n-1] + \ldots + b_K\, x[n-K]) - (a_1\, y[n-1] + \ldots + a_L\, y[n-L]).$$