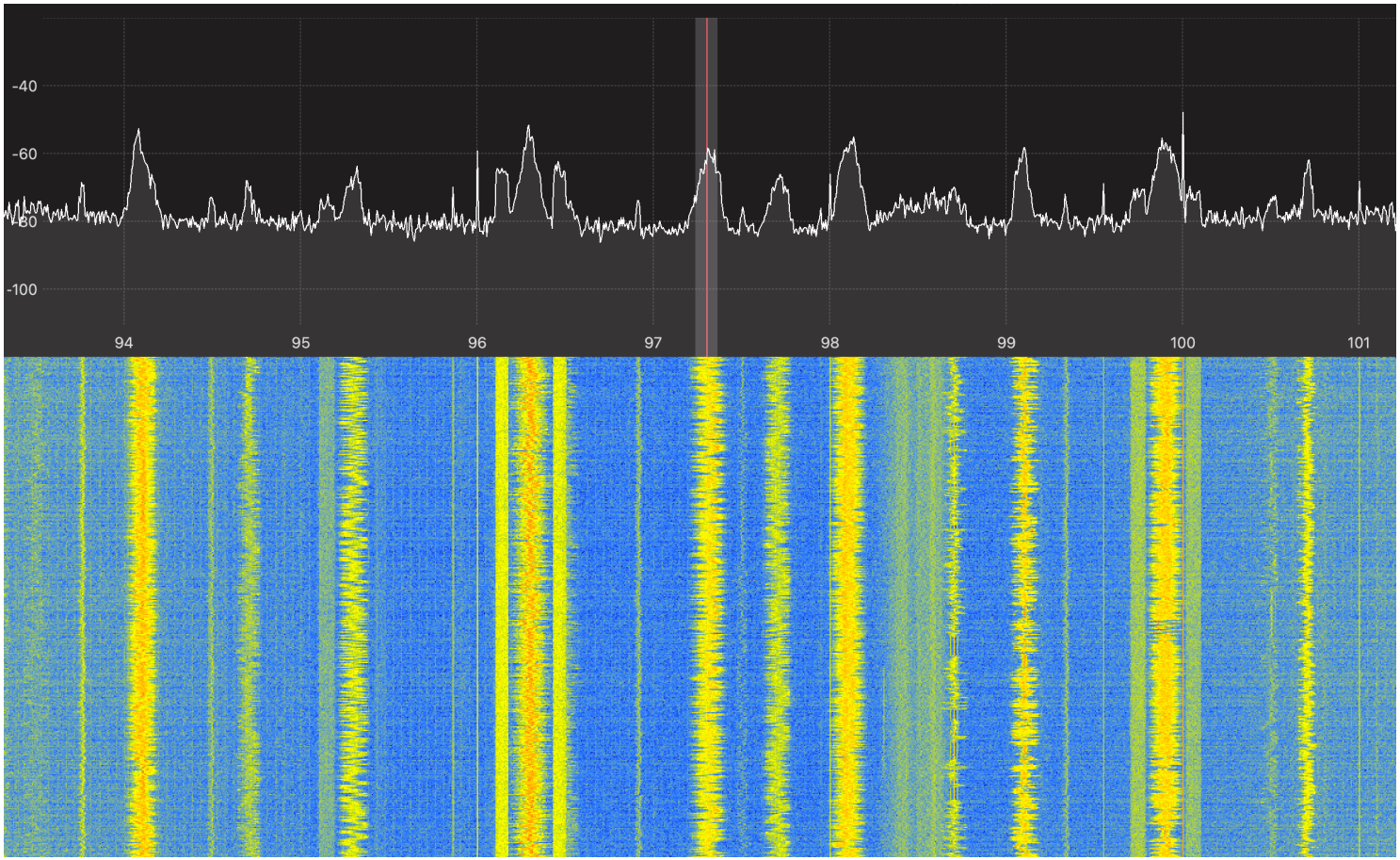
Project Overview



Project Overview (modeling)

Python SciPy/NumPy

Convolution

Fourier transform

I/Q samples

Multirate DSP

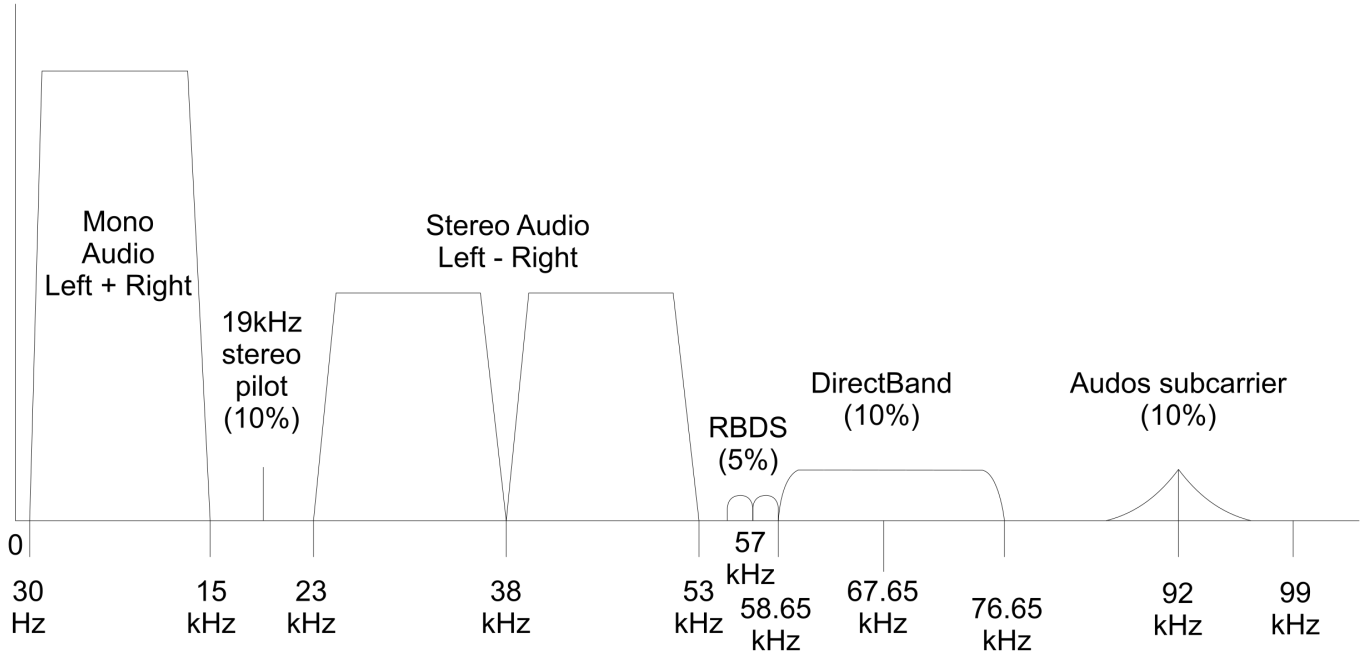
FM demodulation

Clock/data recovery

Frame synchronization

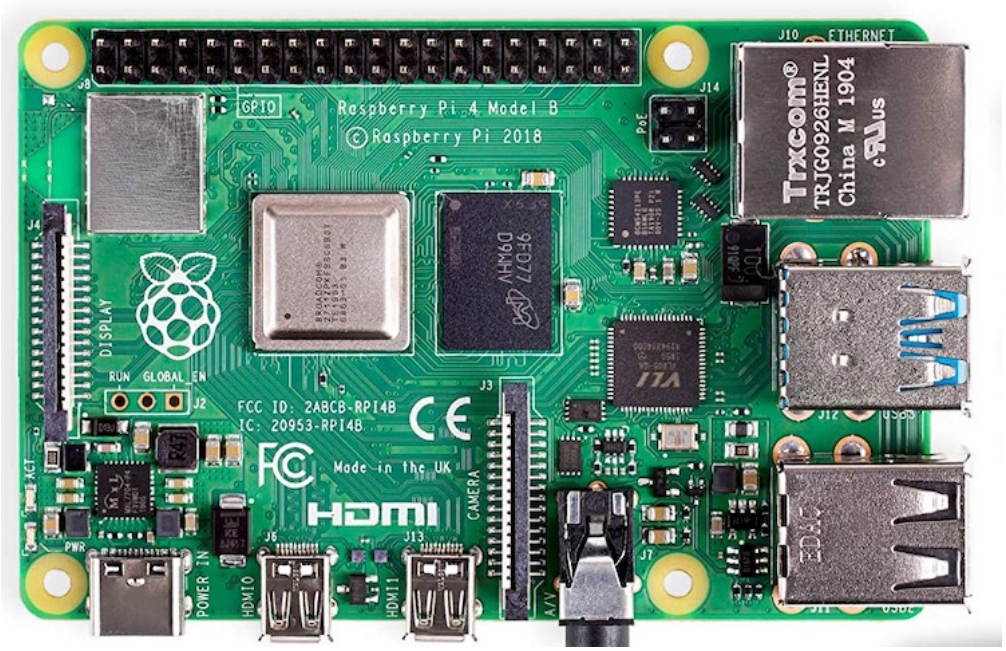
ISI

PLL



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

Project Overview (implementation)



- C++ STL
- Unix pipes
- Queues
- Semaphores
- Mutex
- Performance profiling
- Performance optimization
- Multi-threading
- Multi-core
- Real-time

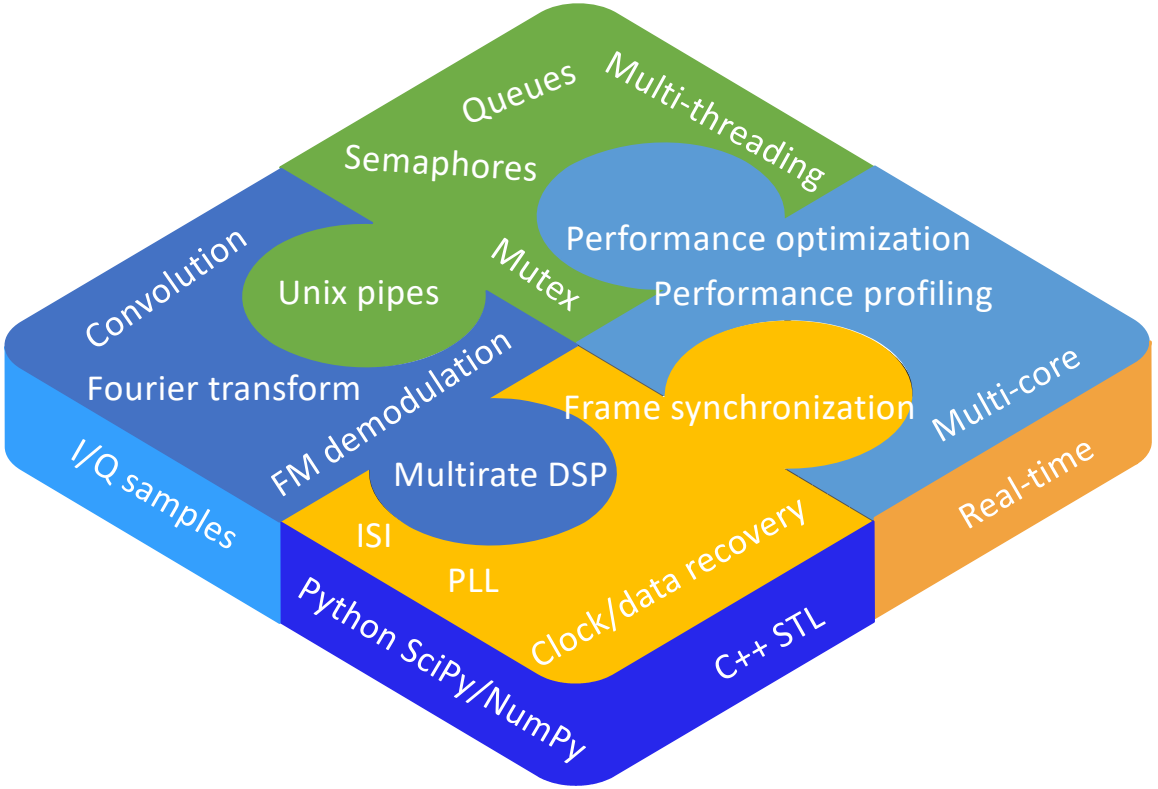
https://www.amazon.ca/gp/product/B01GDN1T4S/

Project Overview (constraints)

Settings for group 99	Mode 0	Mode 1	Mode 2	Mode 3
RF Fs (KSamples/sec)	2400	1152	2400	2880
IF Fs (KSamples/sec)	240	384	240	360
Audio Fs (KSamples/sec)	48	32	44.1	44.1

Settings for group 99	Mode 0	Mode 2
Samples per symbol (SPS)	24	33

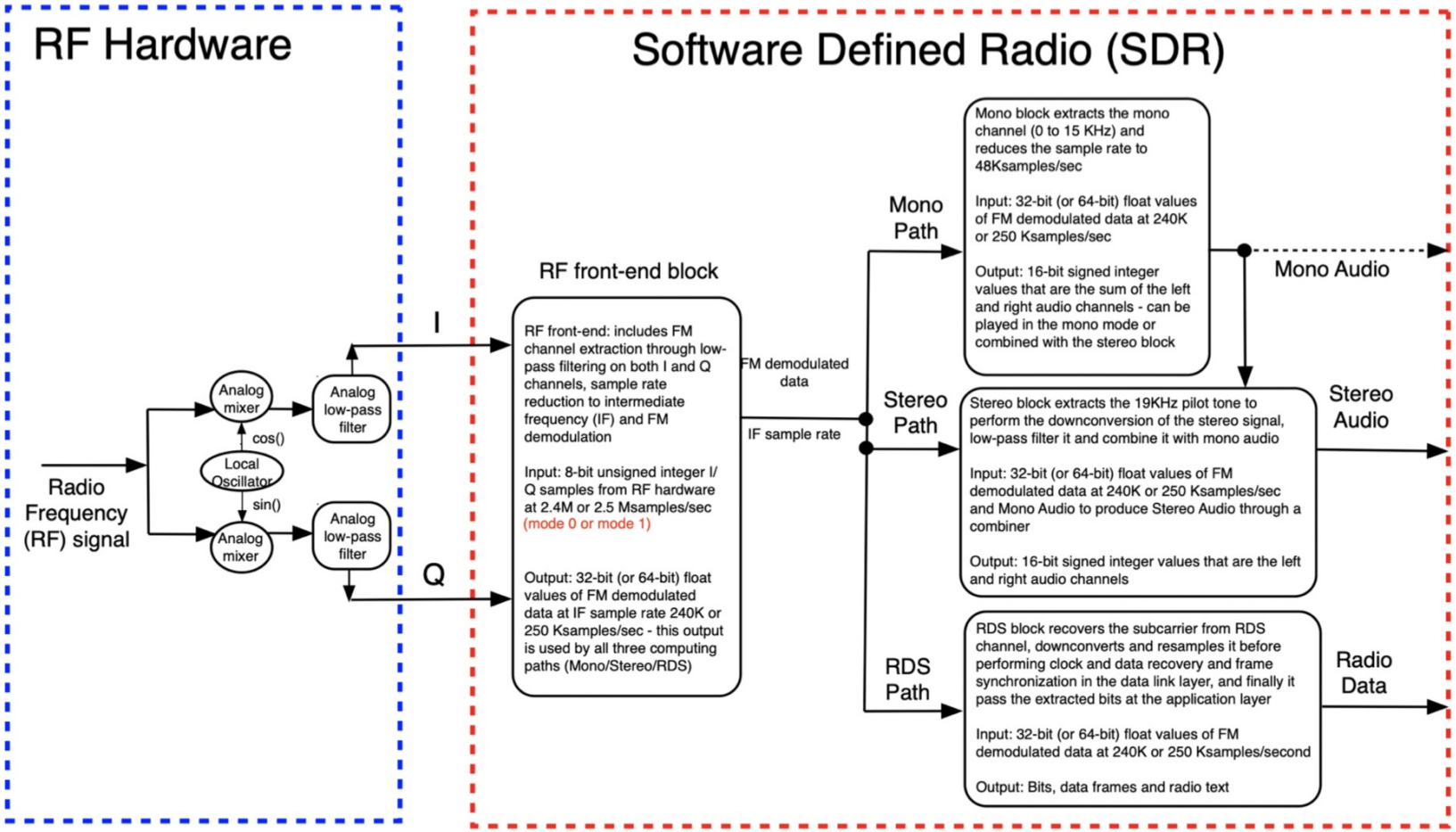
Project Overview (integration)



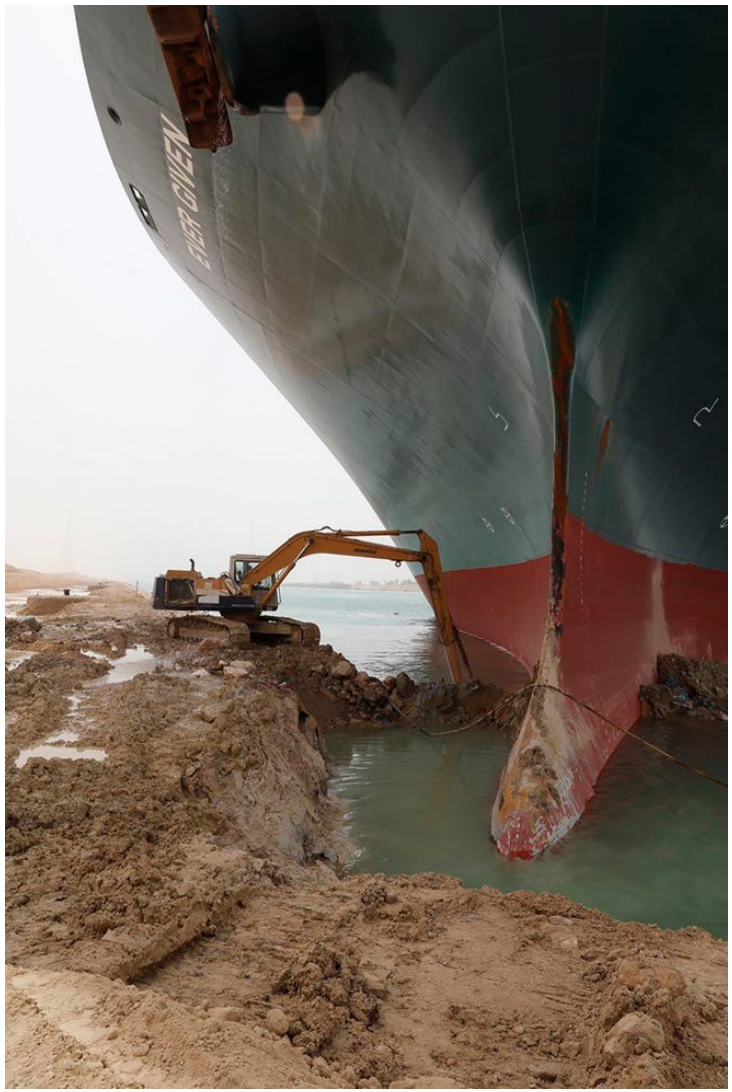
Project Complexity (appearance)

Suez Canal Authority via AP

Managing Complexity (big picture)



Managing Complexity (ad-hoc)

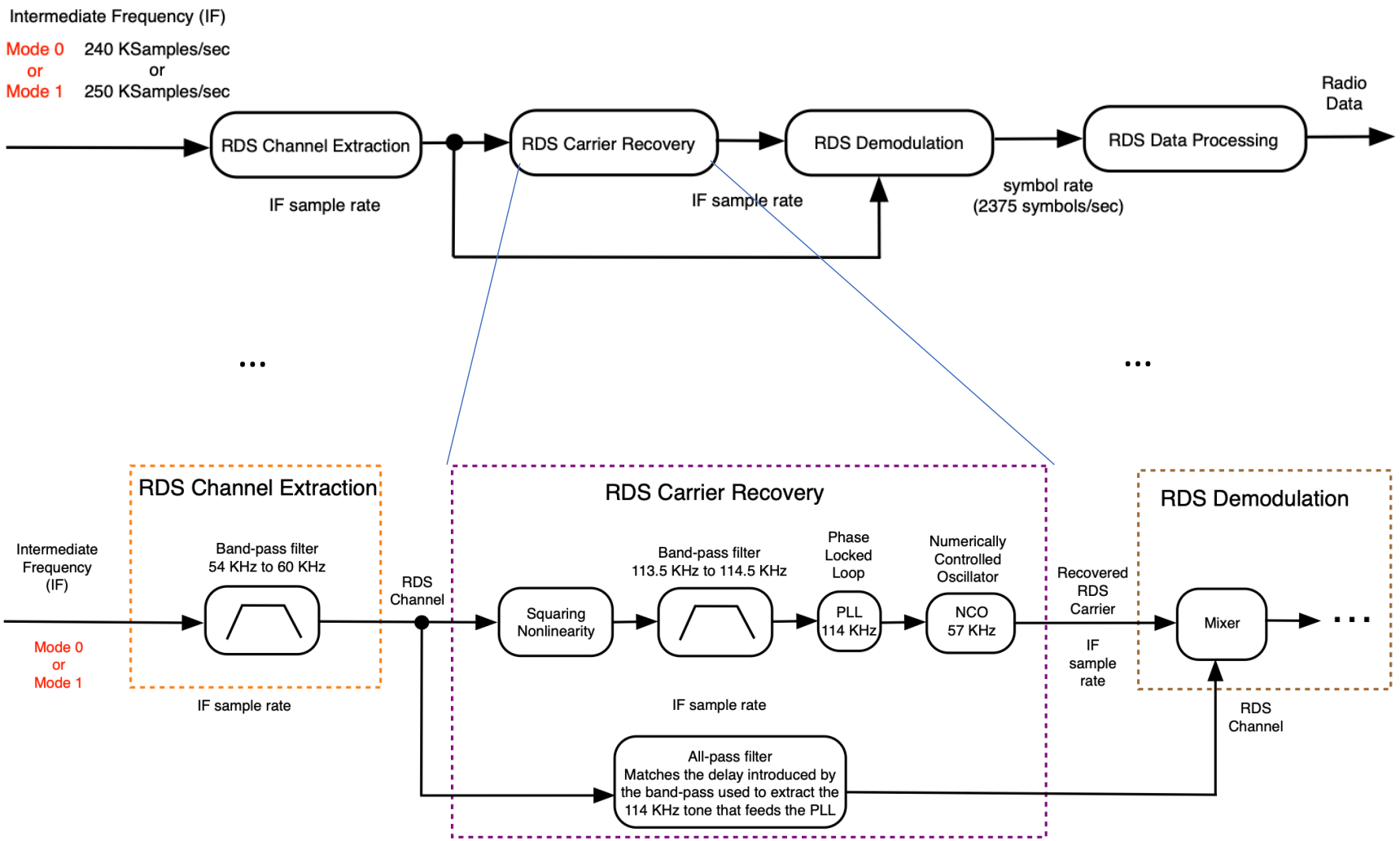


Suez Canal Authority via AP

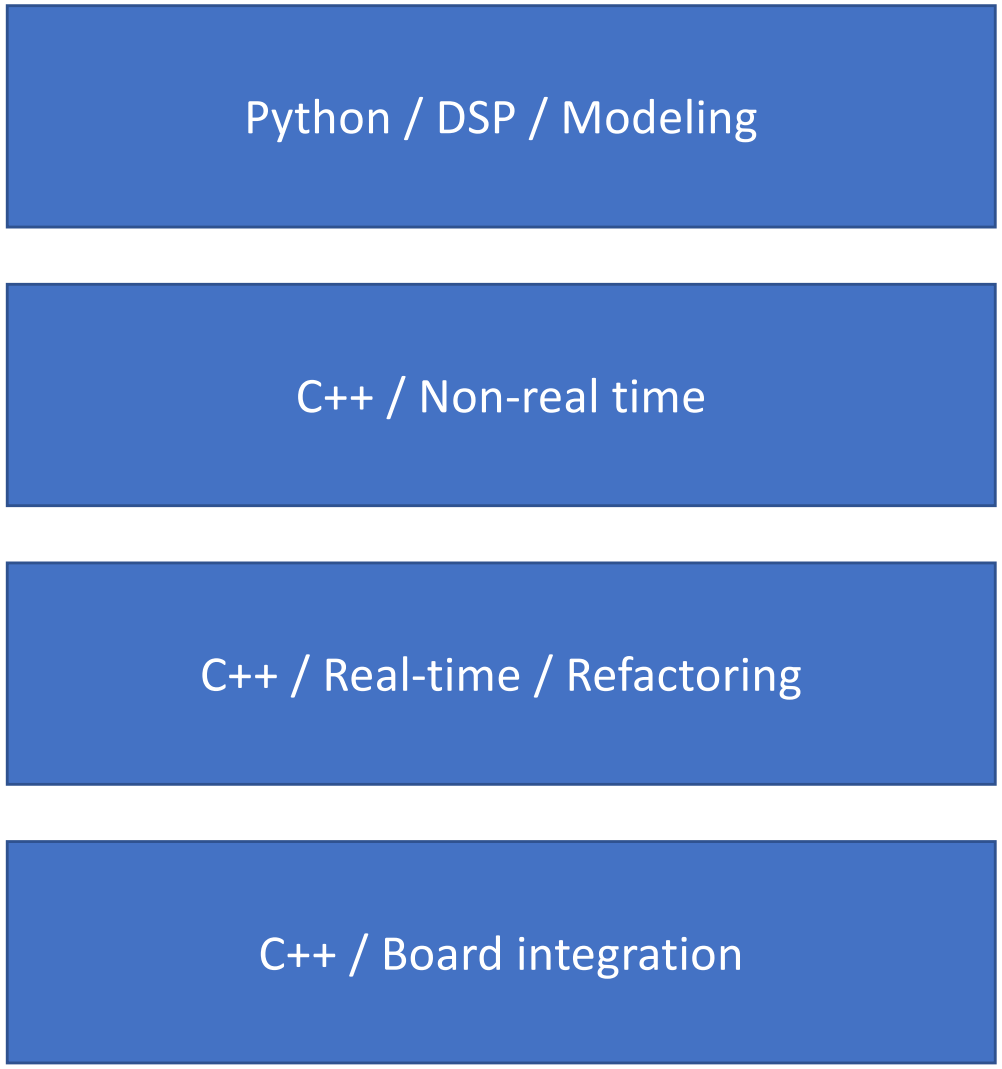
Managing Complexity (timeline)

- Week of Feb 10 - wrap-up lab 4, lab interviews, receive kits, custom constraints
- Week of Feb 17 - recess - "acclimatize" with spec and refactor lab 3 to C++
- Week of Feb 24 - mono path - resampling and using UNIX pipes for streaming
- Week of Mar 3 - stereo path - career recovery, PLLs + status update
- Week of Mar 10 - threading - from basics to synchronization and queues
- Week of Mar 17 - RDS path - emphasize only what is different from the above
- Week of Mar 24 - source code due + discuss report expectations
- Week of Mar 31 - report due + in-class presentations + in-lab examinations
- Week of Apr 7 - bye bye

Managing Complexity (hierarchy)



Managing Complexity (steps)



Dealing with levels of complexity in different steps

- Python model for mode 0 is mandatory for both stereo and RDS before C++ code is attempted

- Unit tests for downsampler and resampler are mandatory

- Unit tests recommended for any new function that was not integrated yet – these unit tests need to be submitted, and a case for correctness needs to be made, if incomplete work is to be assessed

Managing Complexity (plan)

Suez Canal Authority via AP

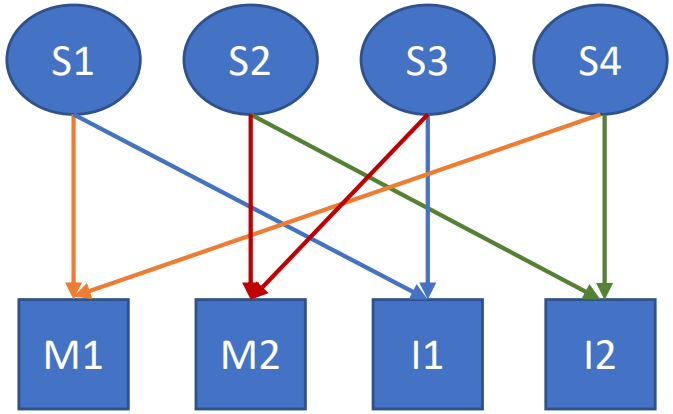
Managing Complexity (teamwork)

http://www.hopa.ro/

Knowledge Building (responsibility)

- Mono path in mode 0
  - every single group member must be involved and understand the details of both modelling (Python) and implementation (C++)

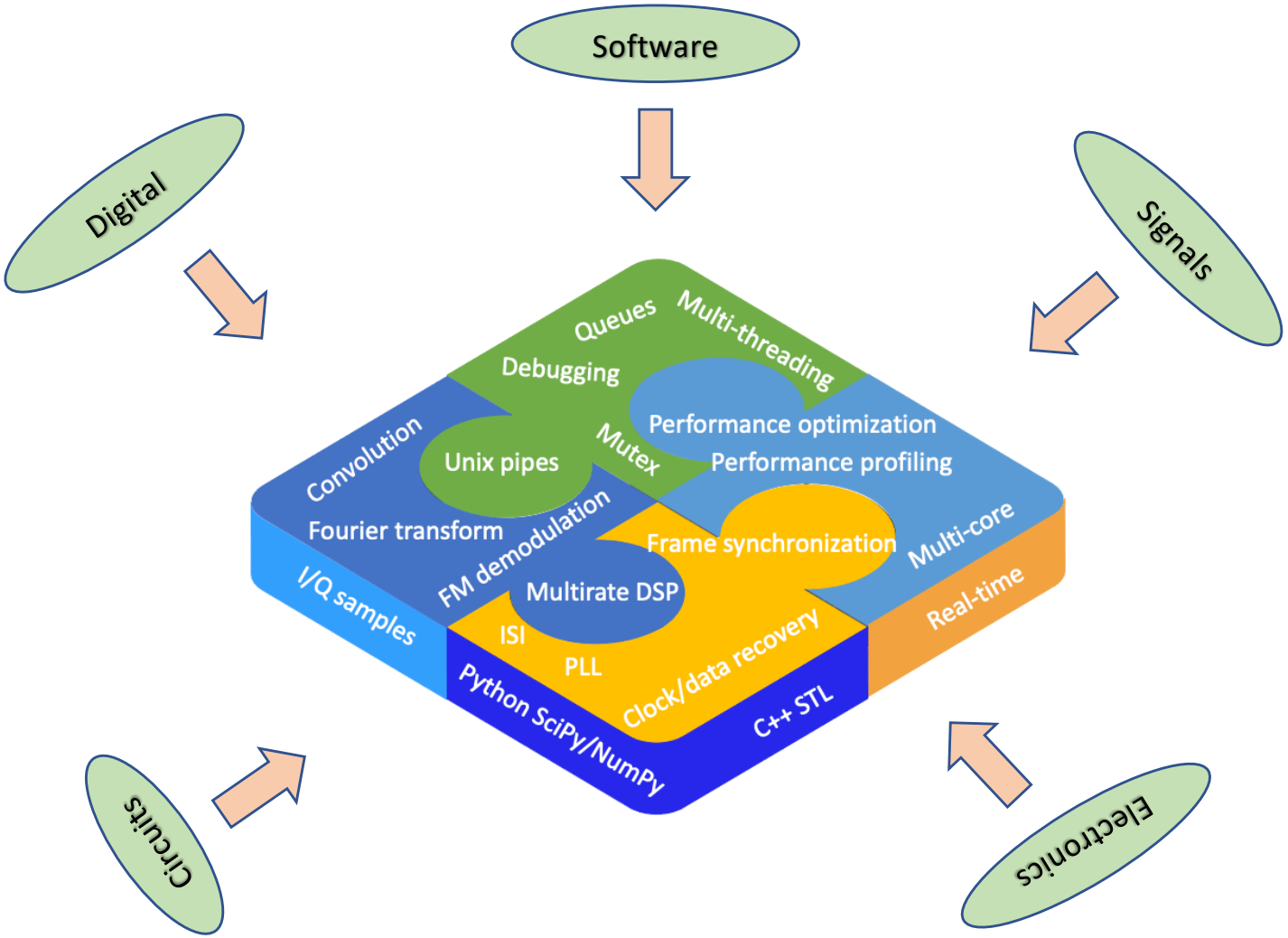
- All the other paths/modes
  - concurrent engineering
  - assume responsibility (non-trivial tasks)
  - at least one modelling (M) task
  - at least one implementation (I) task
  - (implied) fair division of labour
  - **type of tasks** and **level of understanding** will both influence evaluation



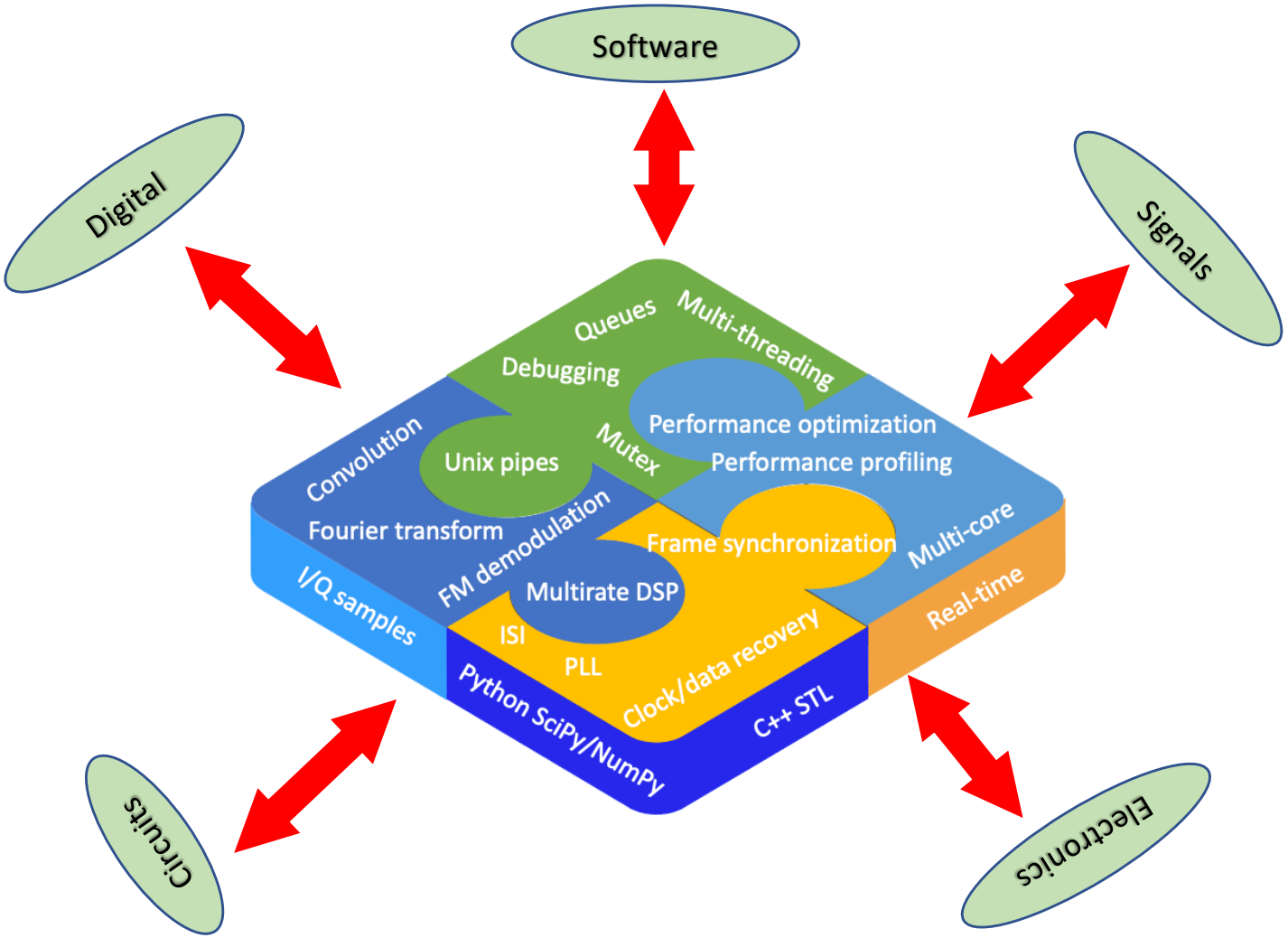
Knowledge Building (avoid roles)

https://www.linkedin.com/pulse/decoding-roles-choosing-between-coder-programmer-your

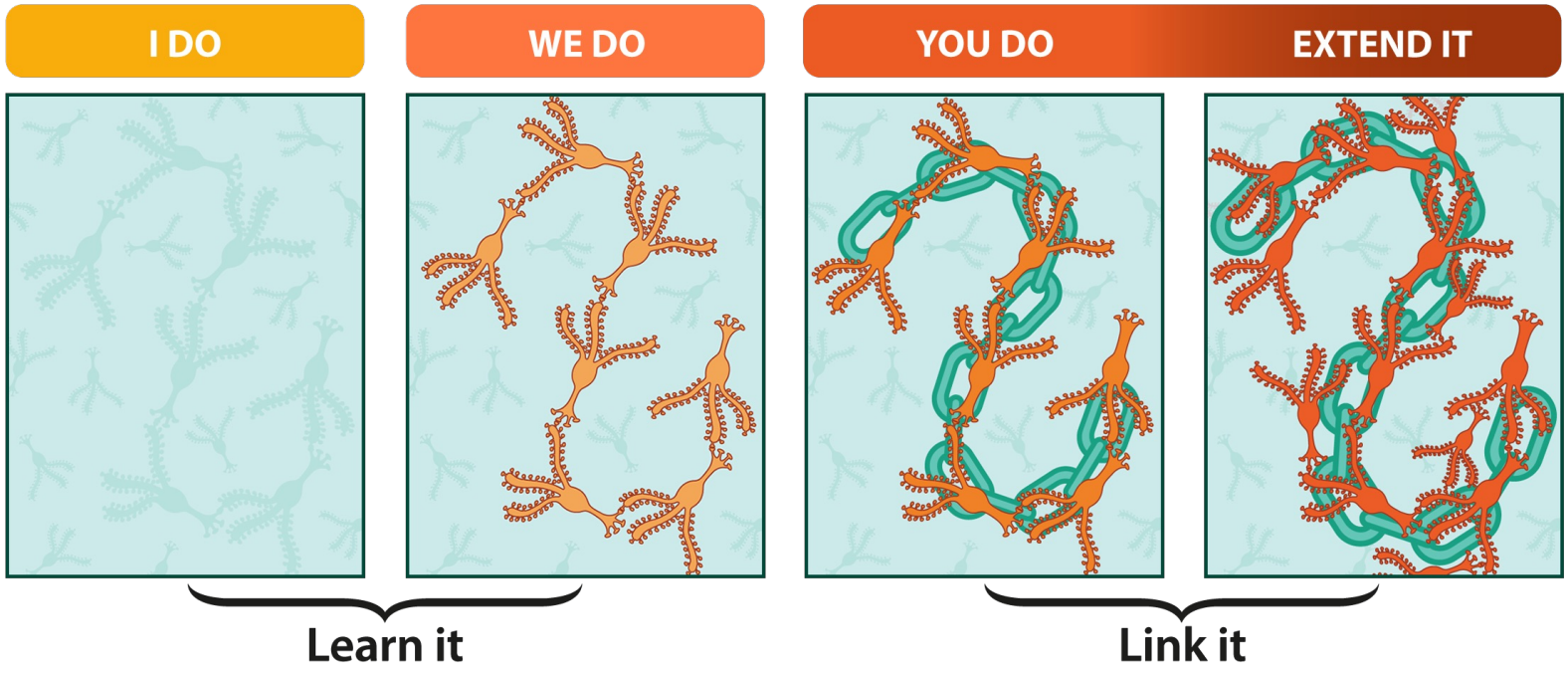
Knowledge Integration



(Future) Learning Stimulation



Computer Systems Integration Project



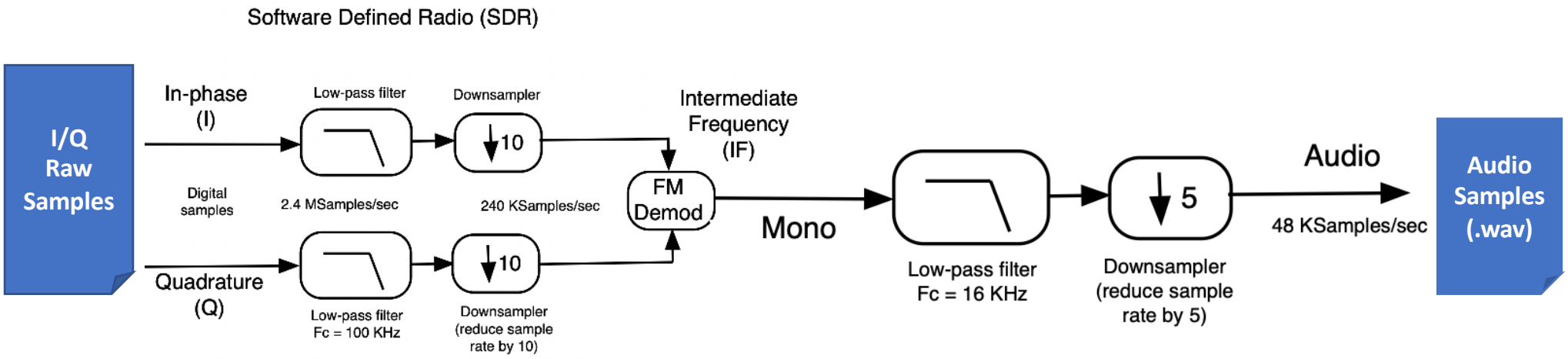
https://barbaraoakley.com/books/uncommon-sense-teaching/

Project – Getting Organized

When?	What?	Who?	Where?
Mar 4, 5, 6	Status update	One group member from each group	Lab
Mar 17-24	Extended lab office hours	Teaching team	Lab
Mar 24	Last day for clarifications	Teaching team	Lab
Mar 27 at 7 p.m.	Source code deadline	All groups	GitHub
Mar 31 Apr 4	In-class presentations	All groups	Classroom
Apr 1, 2, 3	Project cross-examinations	All groups	Lab
Apr 4 at 7 p.m.	Report deadline	All groups	GitHub

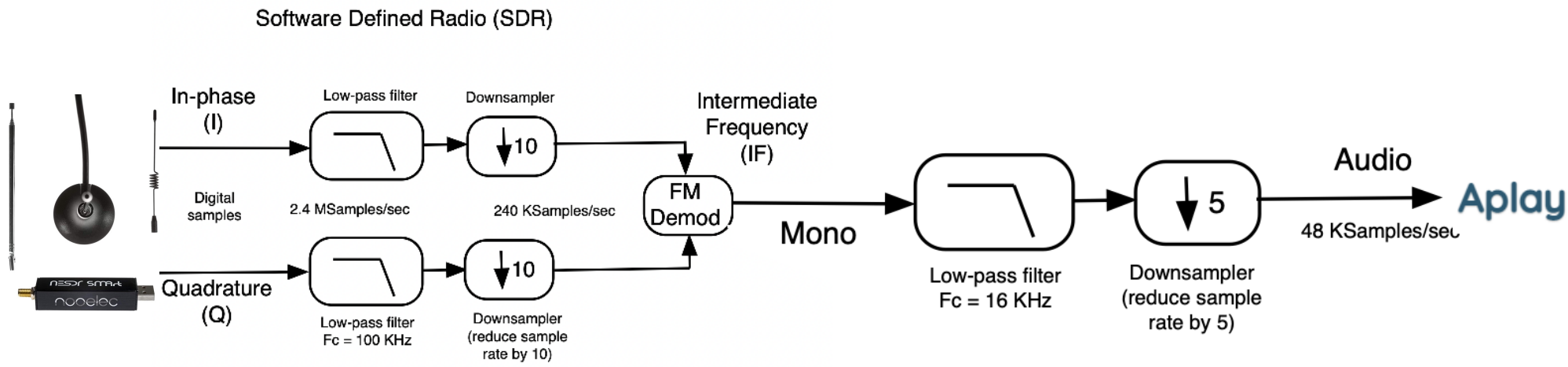
Project – Practical Aspects

- Lab 3 - recorded I/Q samples processed into audio file (.wav)
  - Not applicable to real-life radio



Project – Practical Aspects

- Project - real-time I/Q samples from RF dongle to audio player
  - Must validate lab 3 signal flow graph using the real-life setup



Project – Practical Aspects

- Record 8-bit IQ samples from RF dongle

```
rtl_sdr -f 99.9M -s 2.4M - > my_samples_u8.raw
```

- Use the recorded file by redirecting its contents to `stdout`, and feed it into the `stdin` of your software through Unix pipes

```
cat my_samples_u8.raw | ./my_proj | aplay -c 1 -f S16_LE -r 48000
```

- If software is too slow (initially), break the pipe in two stages:

```
cat my_samples_u8.raw | ./my_proj > my_audio.bin
cat my_audio.bin | aplay -c 1 -f S16_LE -r 48000
```

- Command line arguments of the program **not** shown above

Project – Practical Aspects

- How to read input data from Unix pipe into your C++ program

```
cat my_samples_u8.raw | ./my_proj
```

```
for (unsigned int block_id=0; ; block_id++) {
    std::vector<float> block_data(BLOCK_SIZE);
    readStdinBlockData(BLOCK_SIZE, block_id, block_data);
    if ((std::cin.rdstate()) != 0) {
        std::cerr << "End of input stream reached" << std::endl;
        exit(1);
    }
    std::cerr << "Read block " << block_id << std::endl;
}
```

Read one block of data

adjust float to **real**

Read from stdin

Unix pipe drives input

```
// data in the redirected input is interpreted as bytes (no endianness concerns)
void readStdinBlockData(unsigned int num_samples, \
    unsigned int block_id,
    std::vector<float> &block_data)
{
    std::vector<char> raw_data(num_samples);
    std::cin.read(reinterpret_cast<char*>(&raw_data[0]), num_samples*sizeof(char));
    for (int k=0; k<(int)num_samples; k++) {
        // automatically normalizes the data to the range -1 to +1
        block_data[k] = float(((unsigned char)raw_data[k]-128)/128.0);
    }
}
```

Project – Practical Aspects

- How to write output data to Unix pipe from your C++ program

```
./my_proj | aplay -c 1 -f S16_LE -r 48000
```

```
// processed_data holds the audio output (after all the filtering/resampling/...
std::vector<float> processed_data(BLOCK_SIZE);
// ...
short int sample;
for (unsigned int k=0; k<processed_data.size(); k++) {
    if (std::isnan(processed_data[k])) sample = 0;
    // the scaling below assumes output data is a float between -2 and +2
    else sample = static_cast<short int>(processed_data[k] * 16384);
    // note to above: processed data can be normalized to the -1 to +1 range
    // by following the moving average of the recent blocks of data
    // and adjusting the scale factor (i.e., 16384 above) accordingly
    fwrite(&sample, 1, sizeof(short int), stdout);
}
```

- Sample by sample is slow (not practical)

Project – Practical Aspects

- How to write output data to Unix pipe from your C++ program

```
./my_proj | aplay -c 1 -f S16_LE -r 48000
```

```
// processed_data holds the audio output (after all the filtering/resampling/...
std::vector<float> processed_data(BLOCK_SIZE);
// ...
std::vector<short int> audio_data(BLOCK_SIZE);
for (unsigned int k=0; k<processed_data.size(); k++) {
    if (std::isnan(processed_data[k])) audio_data[k] = 0;
    // prepare a block of audio data to be redirected to stdout at once
    else audio_data[k] = static_cast<short int>(processed_data[k] * 16384);
}
// a block write is approx an order of magnitude faster than writing each sample
// (assuming input streams are processed in blocks of hundreds of kilobytes)
fwrite(&audio_data[0], sizeof(short int), audio_data.size(), stdout);
```

- Write to `stdout` a full block of audio samples (fast and efficient)

Project – Practical Aspects

- How to configure your project to handle different operation modes through terminal?

```
./my_proj
```

argc = 1
argv[0] = "./my_proj"

```
./my_proj 2
```

argc = 2
argv[0] = "./my_proj"
argv[1] = "2"

argc = 3
…
Take-home exercise
…

```
int main(int argc, char* argv[]) {

    int mode = 0;

    if (argc < 2) {
        std::cerr << "Operating in default mode 0" << std::endl;
    } else if (argc == 2) {
        mode = atoi(argv[1]);
        if (mode > 3) {
            std::cerr << "Wrong mode " << mode << std::endl;
            exit(1);
        }
    } else {
        std::cerr << "Usage: " << argv[0] << std::endl;
        std::cerr << "or " << std::endl;
        std::cerr << "Usage: " << argv[0] << " <mode>" << std::endl;
        std::cerr << "\t\t <mode> is a value from 0 to 3" << std::endl;
        exit(1);
    }

    std::cerr << "Operating in mode " << mode << std::endl;

    // ...
}
```

Project – Practical Aspects

- For Unix pipes to work as described, your software is assumed
  - To print user information using only `std::cerr`
  - To output only raw audio samples through `stdout (std::cout)`
- Input samples are 8-bit unsigned integers
  - Default setting for `rtl_sdr` (no need to change anything)
- Output samples are 16-bit signed integers (little endian)
  - Represent audio samples as short integers in your program and provide the format argument to `aplay` as follows `-f S16_LE`
- For the stereo audio
  - Interleave the data from left/right channels before sending it to `stdout`
  - `aplay` must be aware that there are two audio channels and needs argument `-c 2`