EECS 570 Programming Assignment 1

January 16th, 2026

About Me

Jonah Rosenblum

- Education
  - [Redacted] years as PhD student in CSE
  - Undergrad + masters at Michigan
- Research
  - System & Architecture Security
  - Applications of formal methods
- Fun facts
  - I took 570 in [redacted]
  - This is my third time teaching 570
  - Amateur rock climber

Announcements

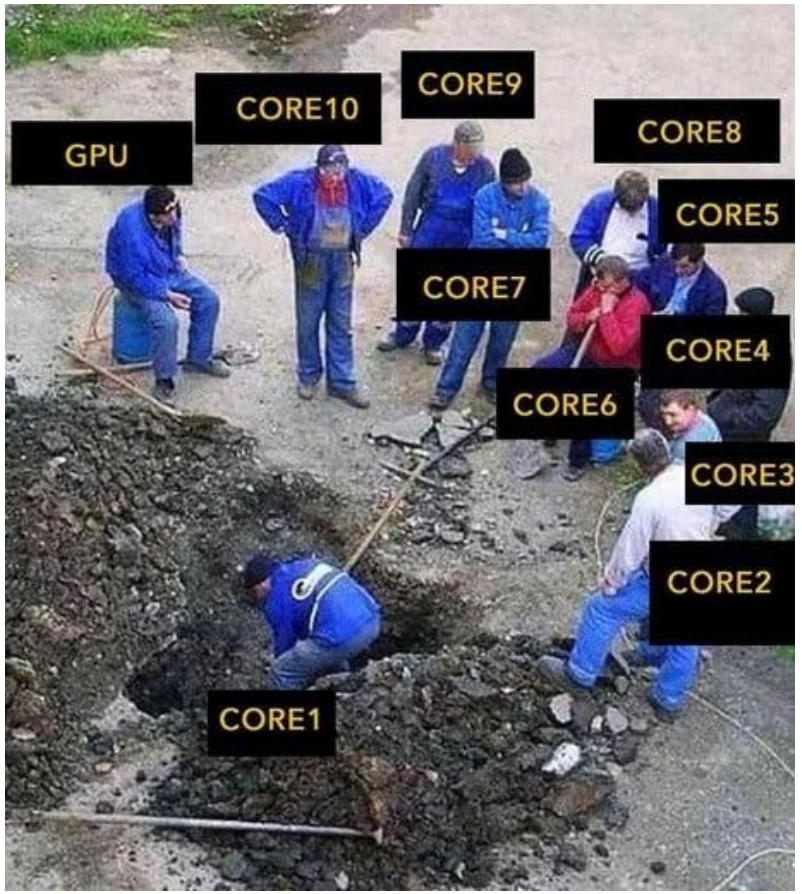
- Final Project
  - Piazza - search for teammates
  - Next Friday: Project introduction discussion / handout
- PA1
  - Due 09/29/2026 on Canvas
  - Please pay attention to submission format!
- Office Hours
  - Tue & Thu: 10:30 – 11:30 AM, 4844 BBB + Zoom
- Quizzes
  - Due each Monday

PA1 Overview

- Parallel Computation
  - Introduction
  - Medical Imaging using Ultrasound

- Introduction to POSIX Threads
  - Thread Creation and Joining
  - Synchronization Primitives

Parallel Computation



Parallel Computation Refresher

- Break up large problem into many small problems
  - Work on each sub-problem in parallel
  - Coordinate workers

- Question: parallelize sum of array
  - How would you implement this?
  - [3, 1, 7, 5, 2, 4, 6, 8]
  - (3 + 1), (7 + 5), (2 + 4), (6 + 8)
    → **[4, 12, 6, 14]**
  - (4 + 12), (6 + 14)
    → **[16, 20]**
  - (16 + 20)
    → **36**

Parallel Computation Performance

- Question: how much faster is this approach?
  - Turns an O(n) problem into a O(log n)*
  - * Given sufficient parallelism, where n is # of sequential steps

- Not always faster in practice
  - I need to divide the array
  - I need to send the data to each worker
  - I need to collect the result from each worker
  - **For every iteration**
  - The larger “n” is the greater the speedup!

Parallel Computation with a Twist

- Can we do better than log(n)?
- Not all problems are parallelizable
  - Dependencies may force sequential computation

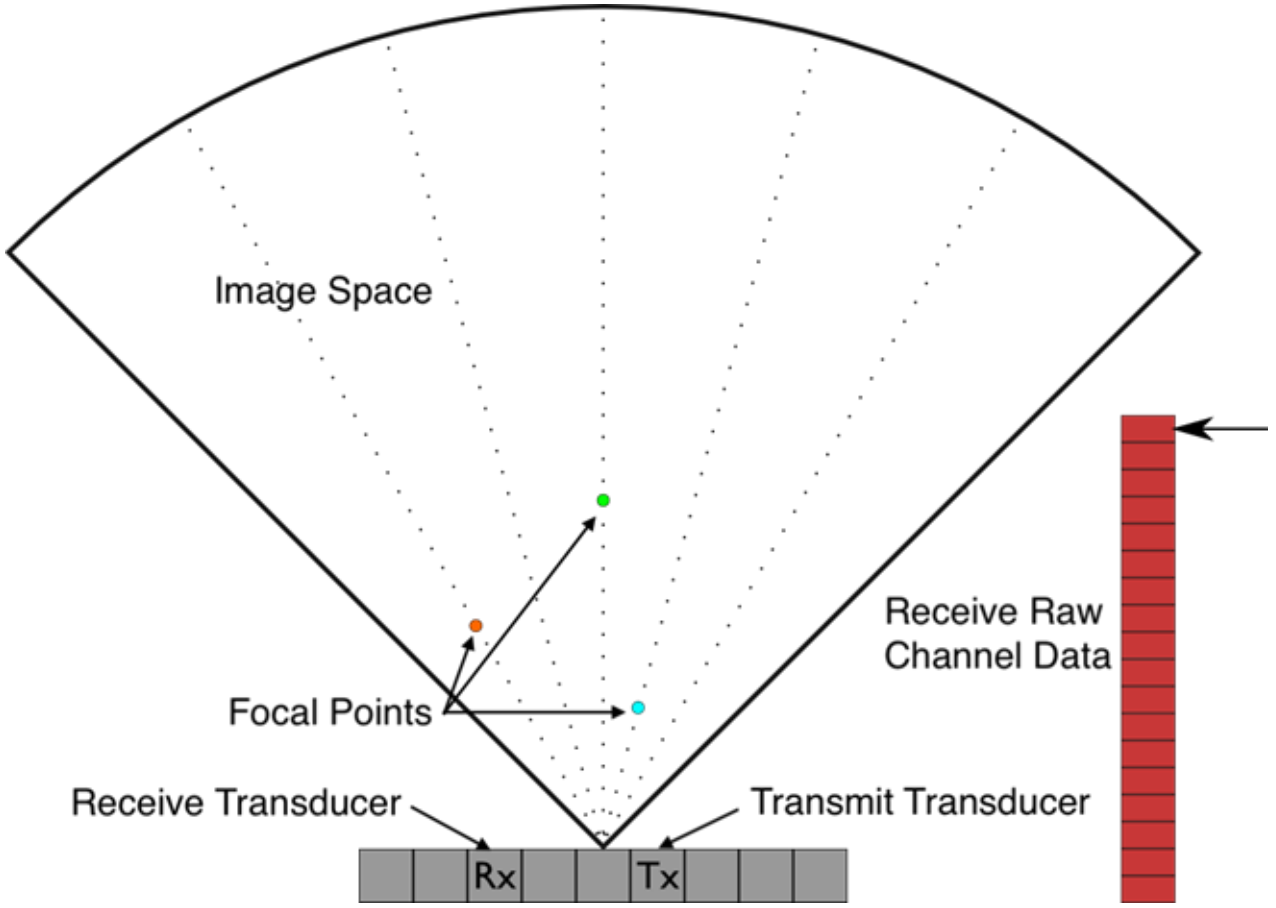
PA1 Intro

Portable Medical Imaging Devices

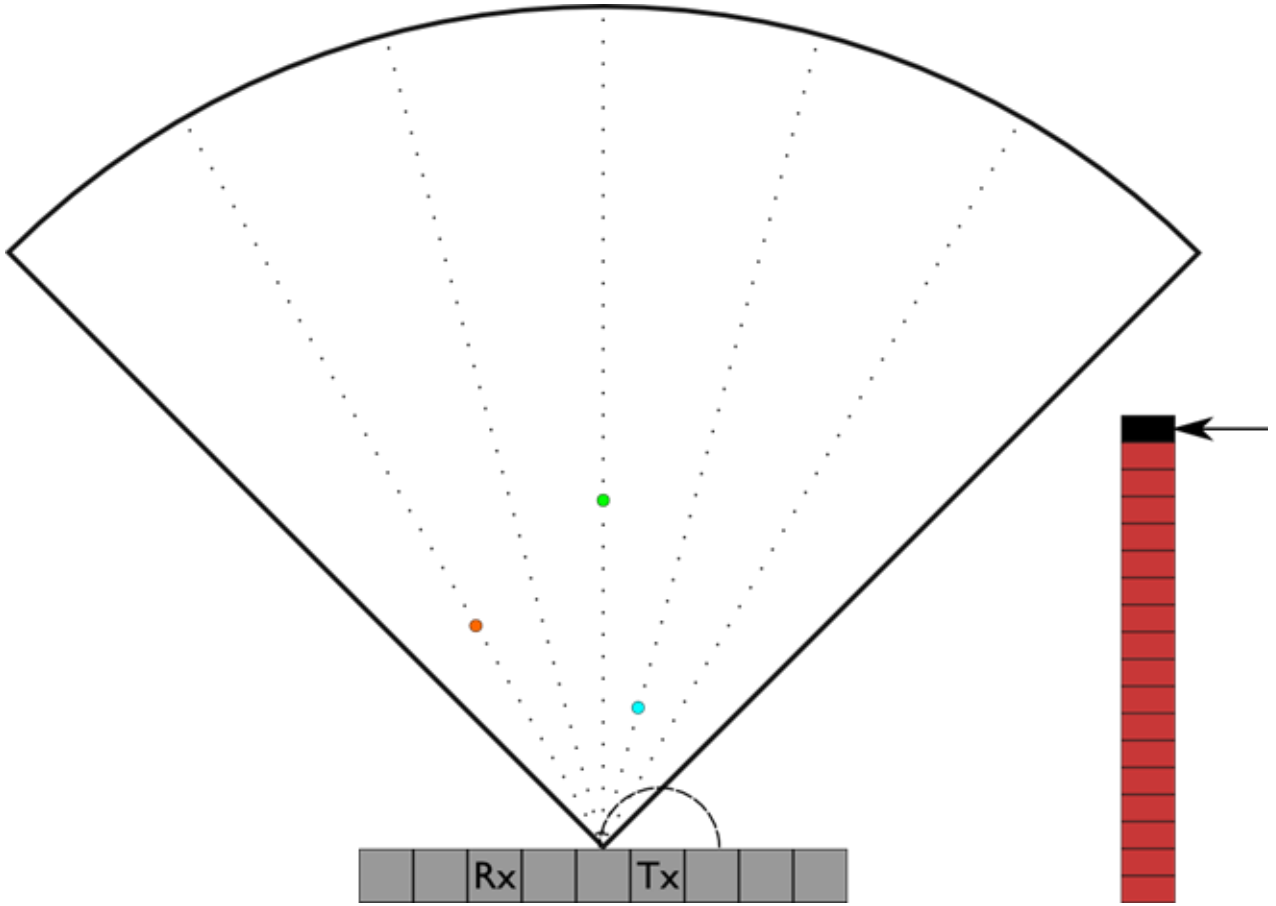
- Medical Imaging moving towards portability
  - MEDICS (X-Ray CT) [Dasika ’10]
  - Handheld 2D Ultrasound [Fuller ’09]
- Not just a matter of convenience
  - Improved patient health [Gunnarsson ’00, Weinreb ’08]
  - Access in developing countries
- Why ultrasound?
  - Low transmit power [Nelson ’10]
  - No danger or side-effects

This is an old course, but the lessons carry over to modern applications

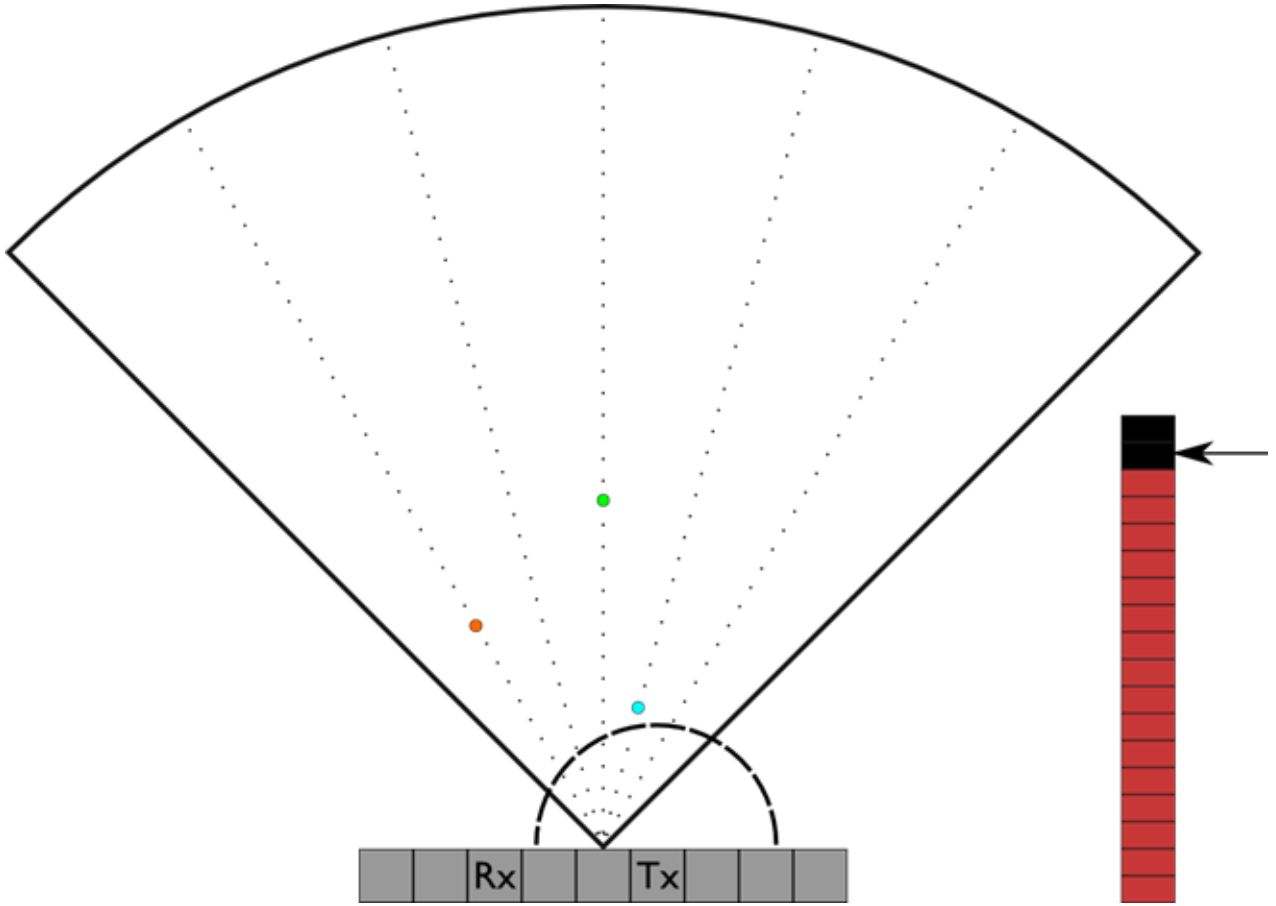
Ultrasound: Transmission and Reception



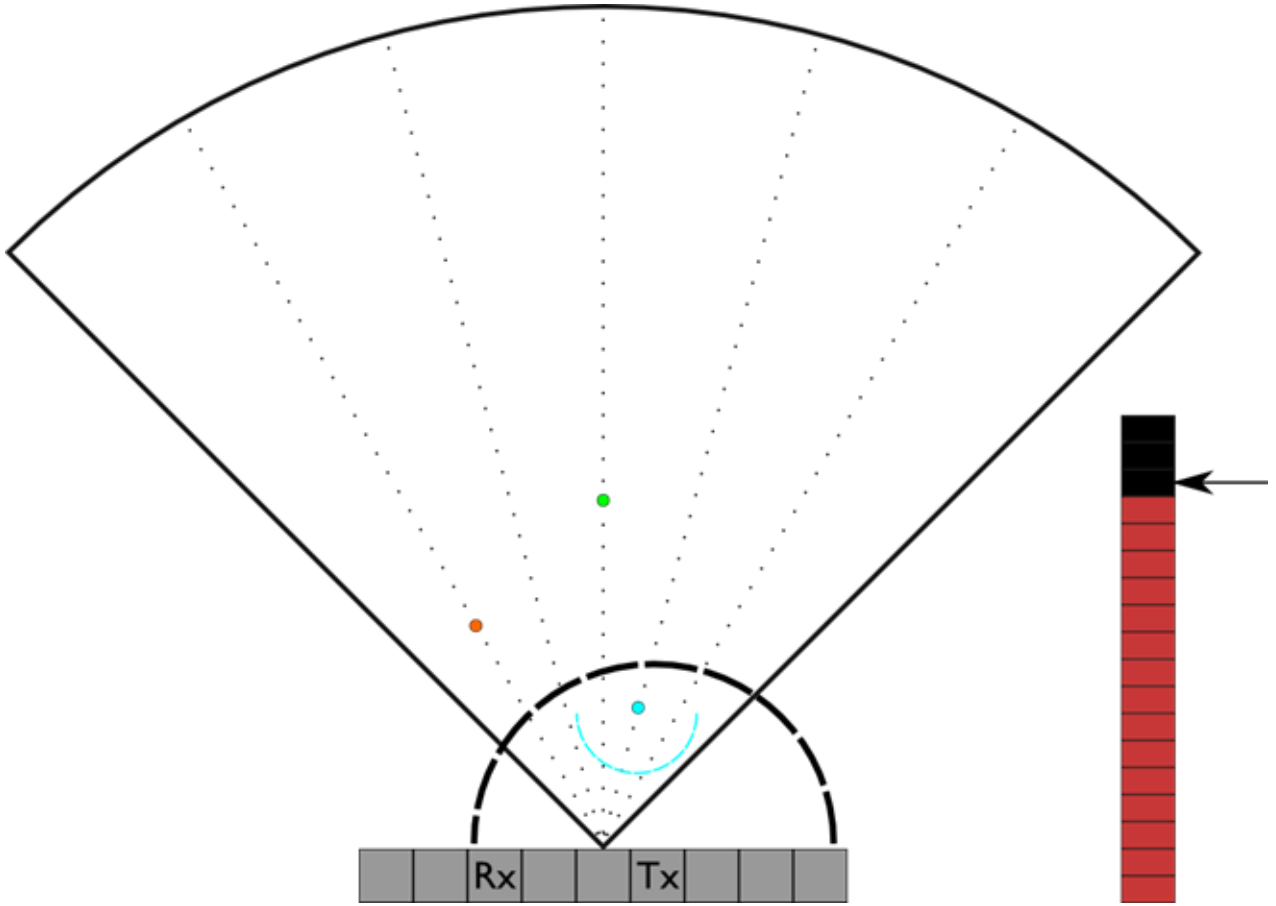
Ultrasound: Transmission and Reception



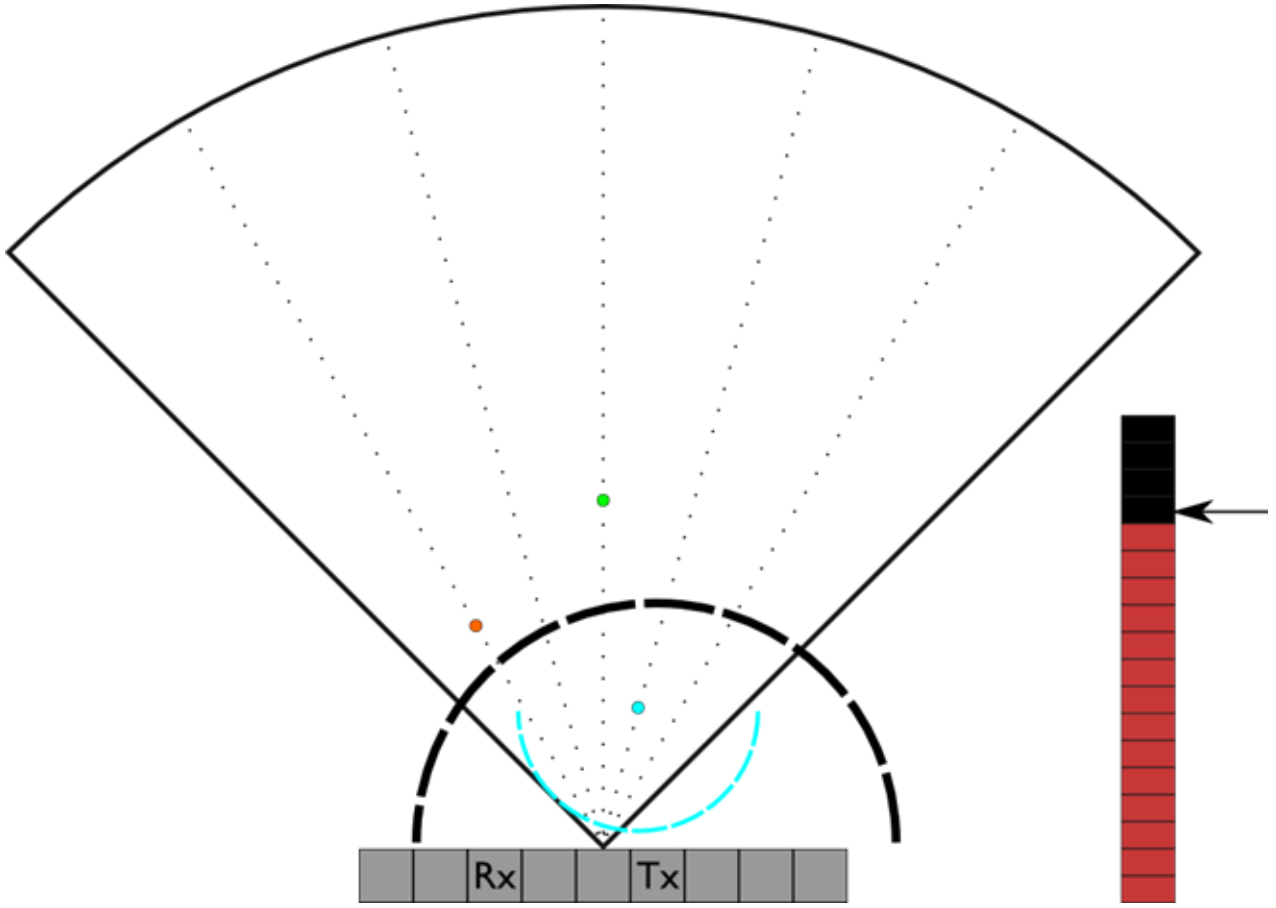
Ultrasound: Transmission and Reception



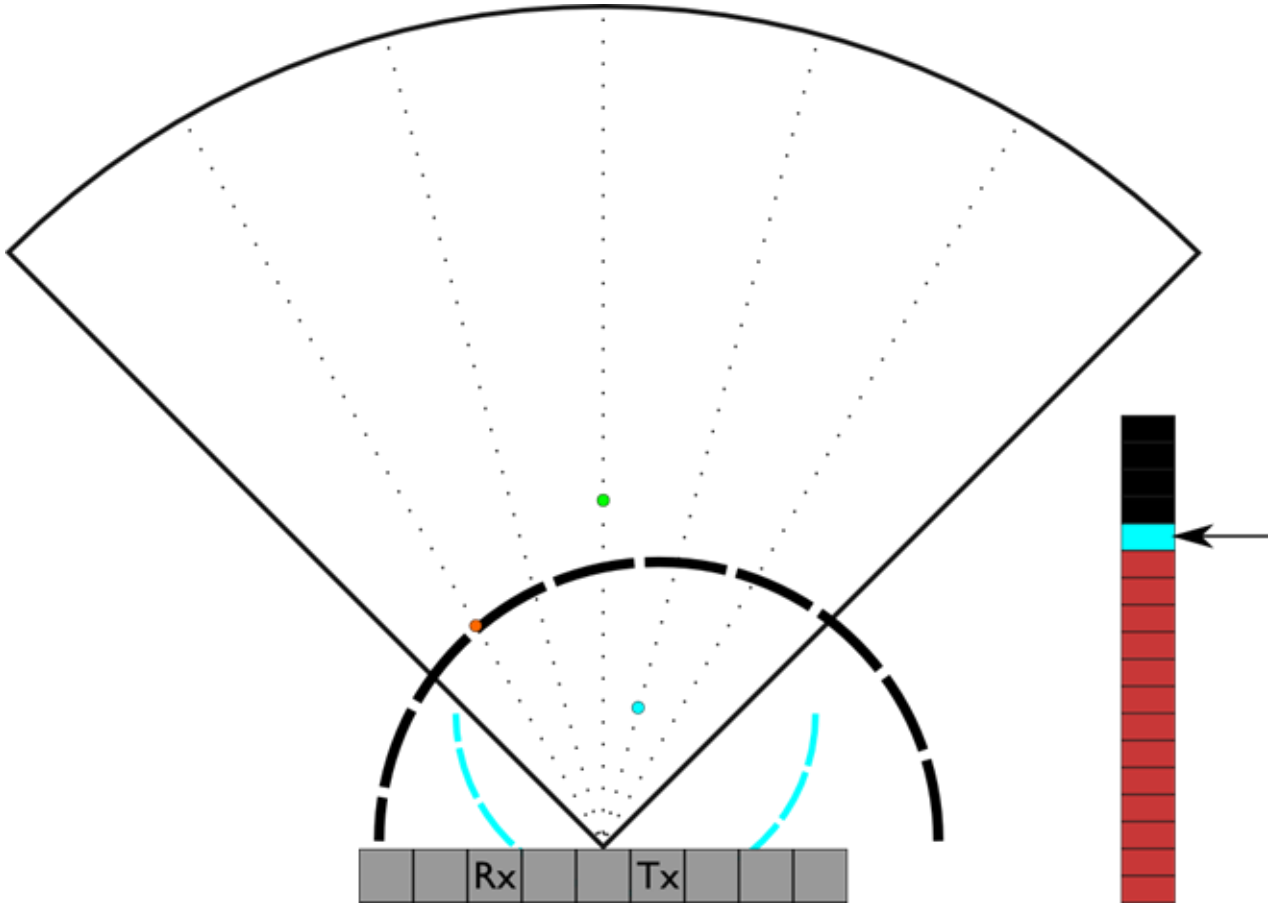
Ultrasound: Transmission and Reception



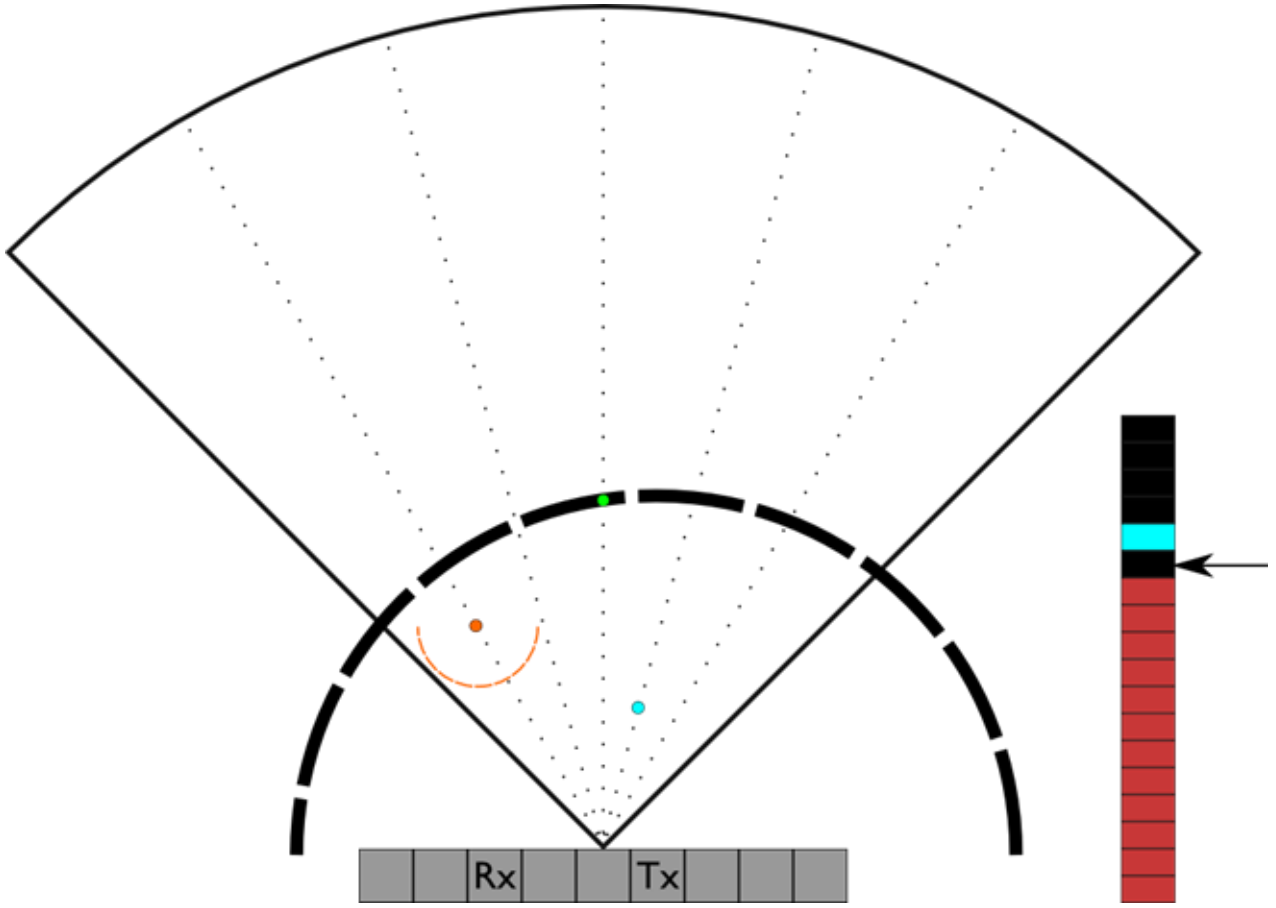
Ultrasound: Transmission and Reception



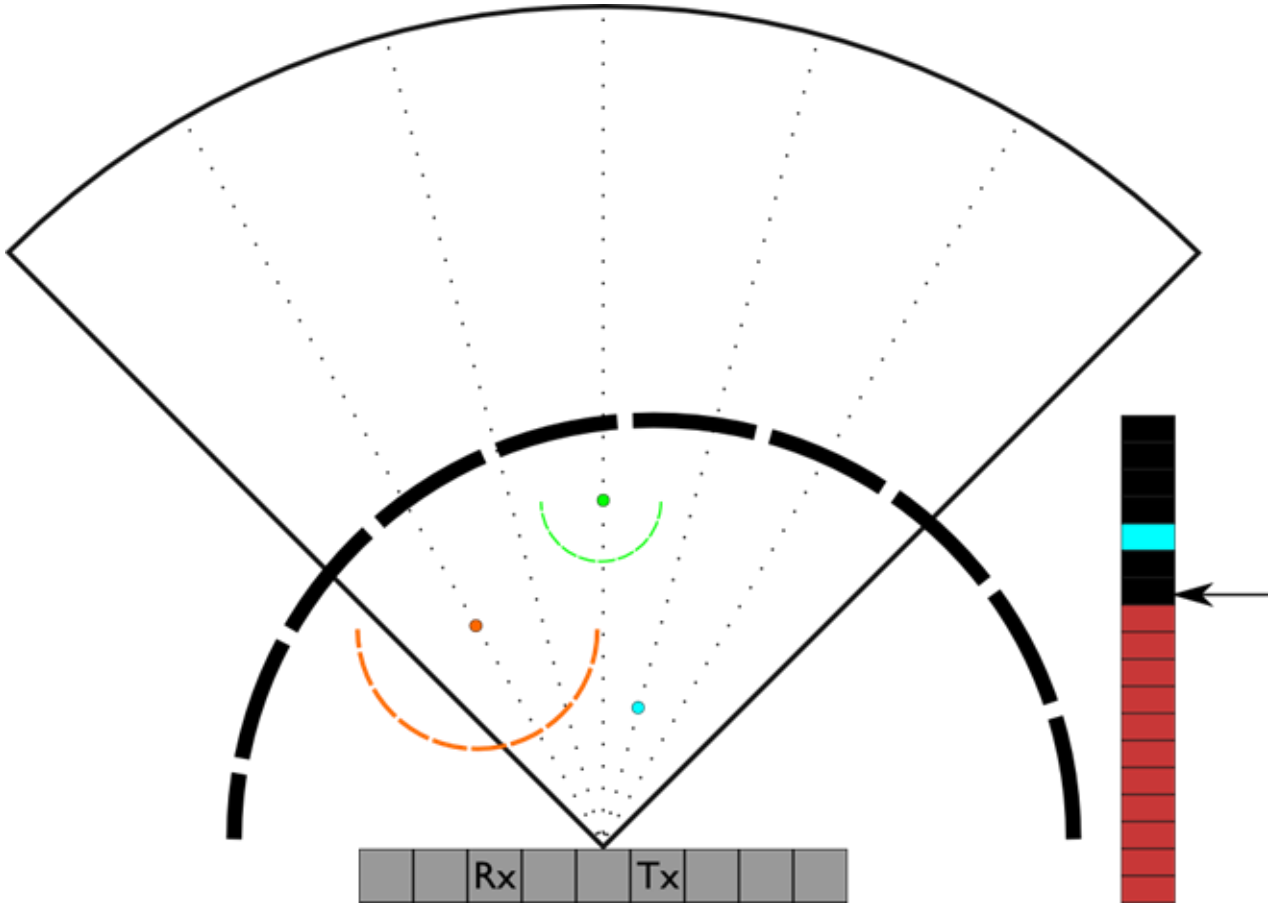
Ultrasound: Transmission and Reception



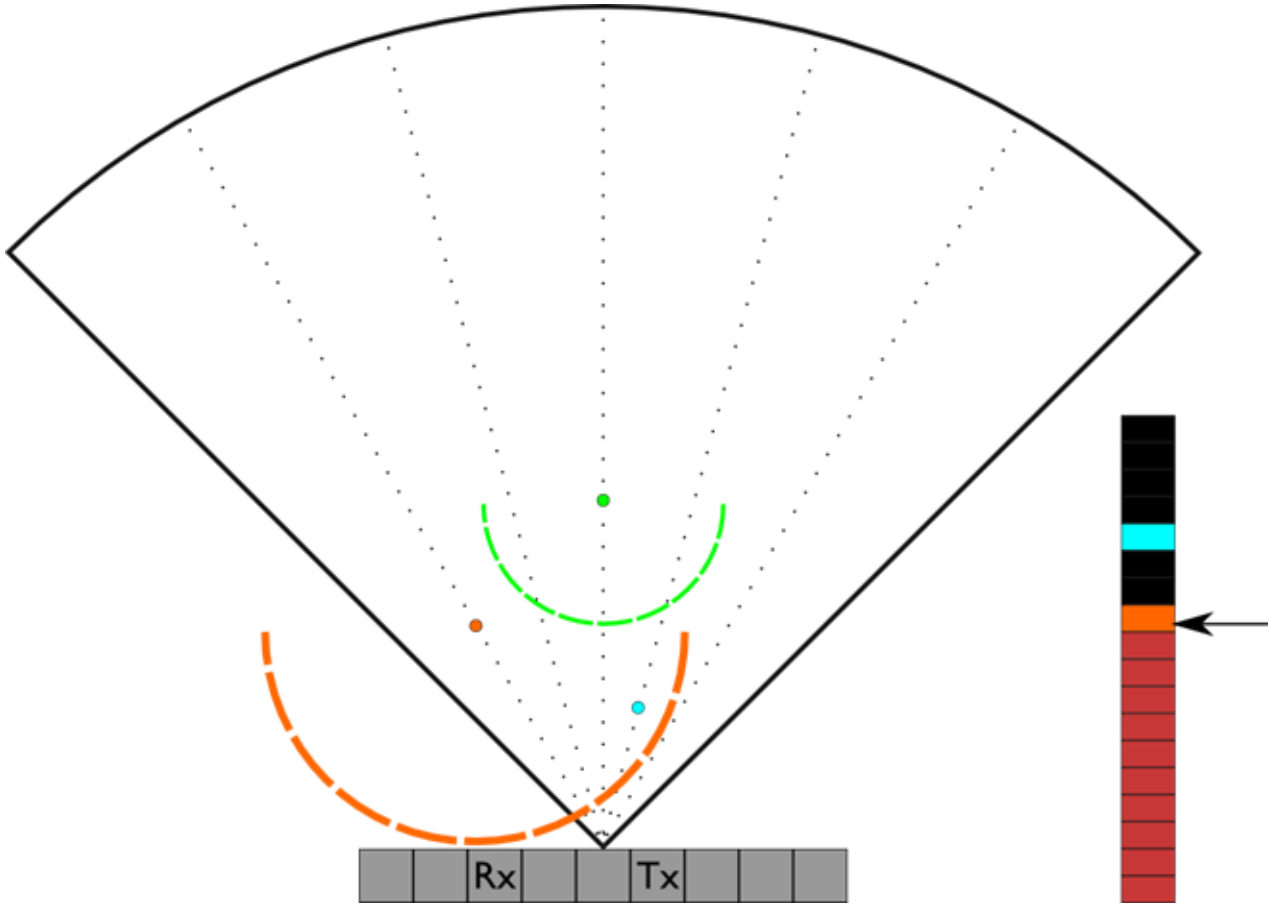
Ultrasound: Transmission and Reception



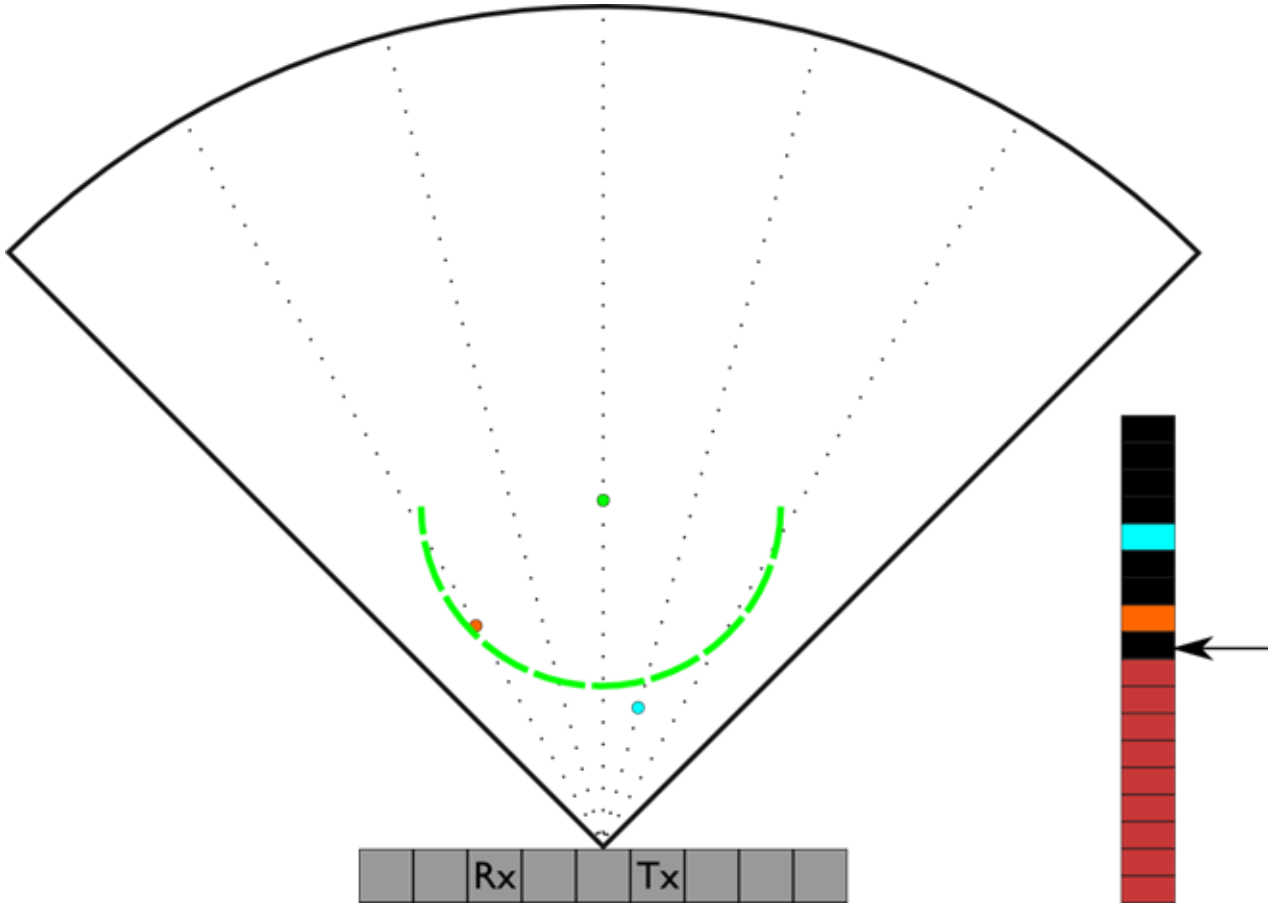
Ultrasound: Transmission and Reception



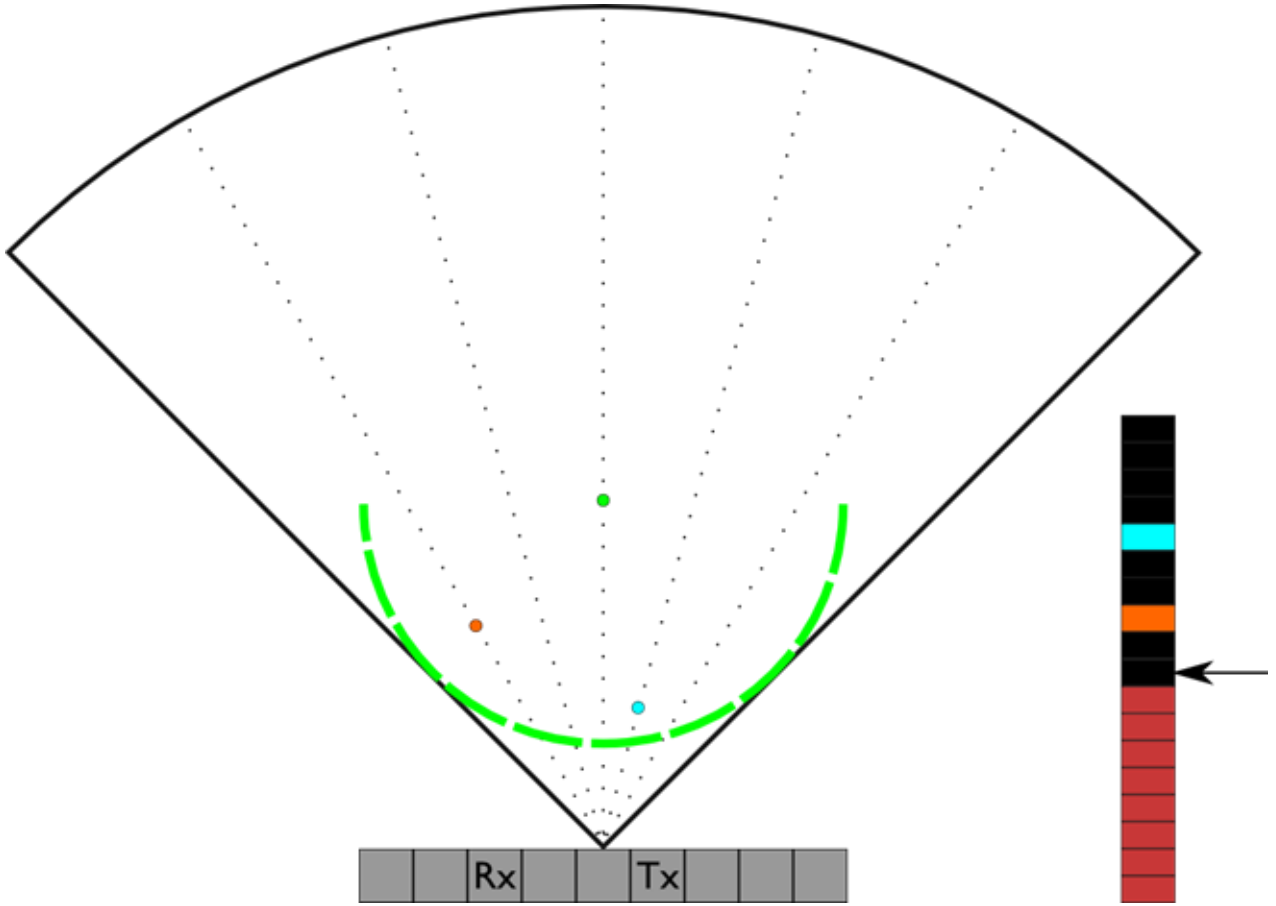
Ultrasound: Transmission and Reception



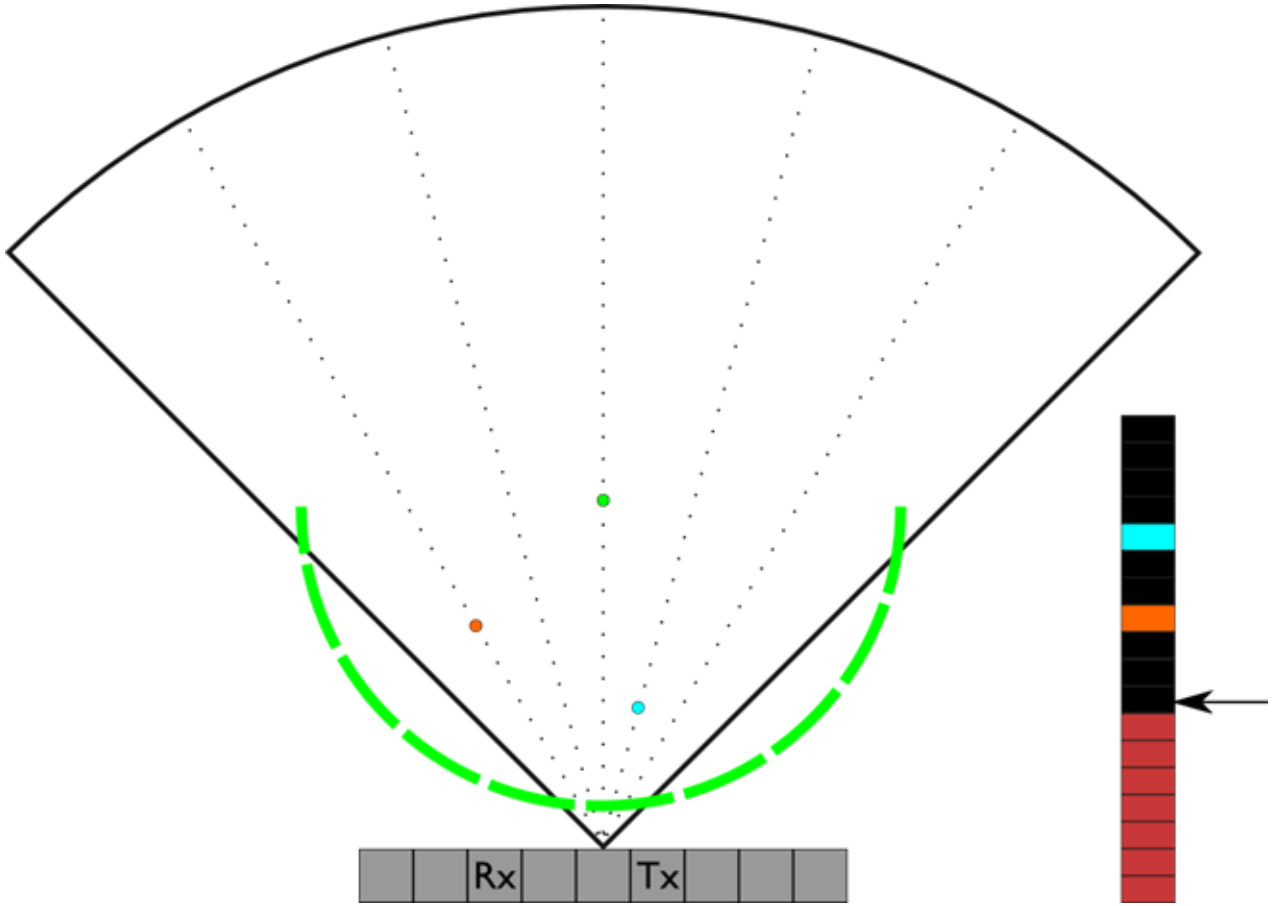
Ultrasound: Transmission and Reception



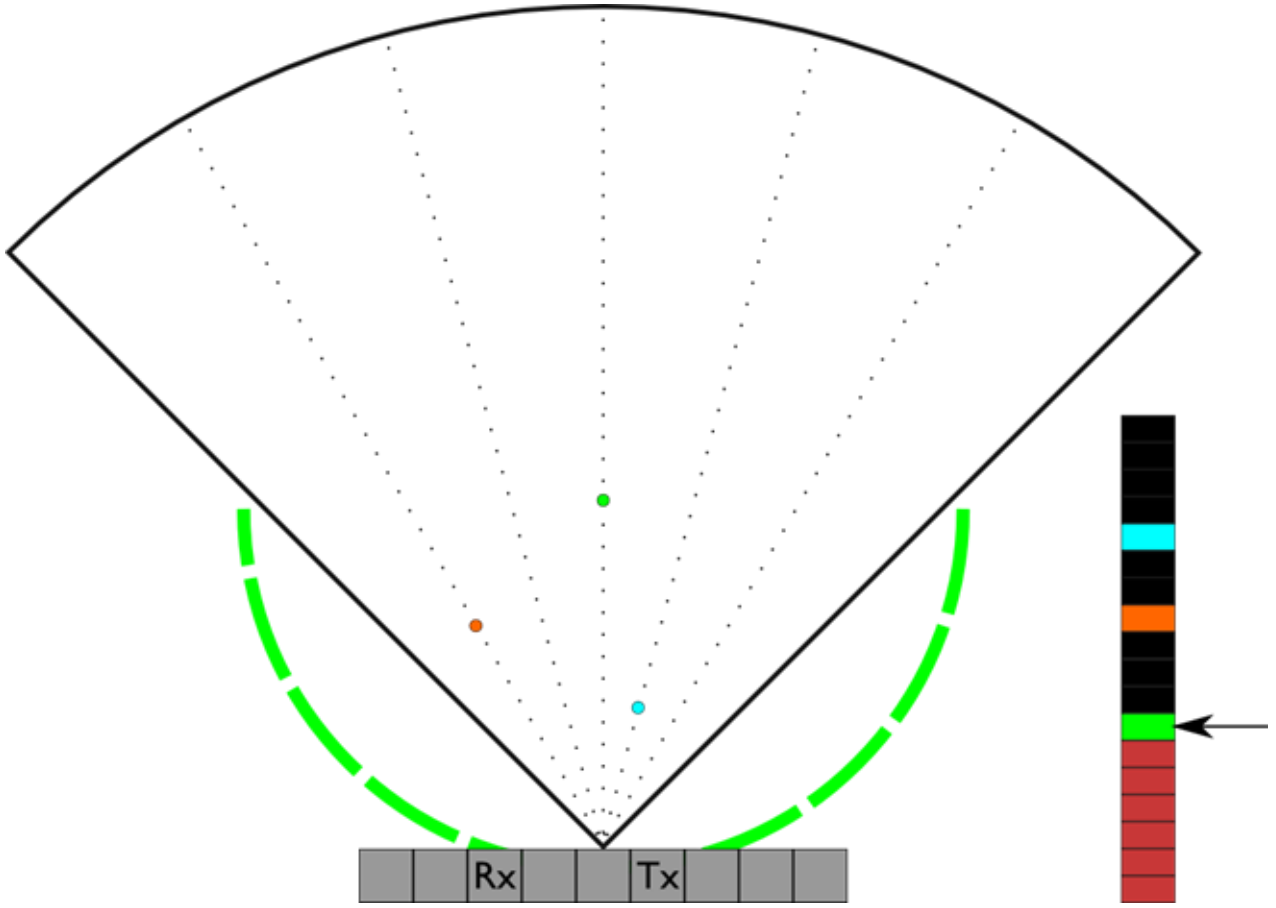
Ultrasound: Transmission and Reception



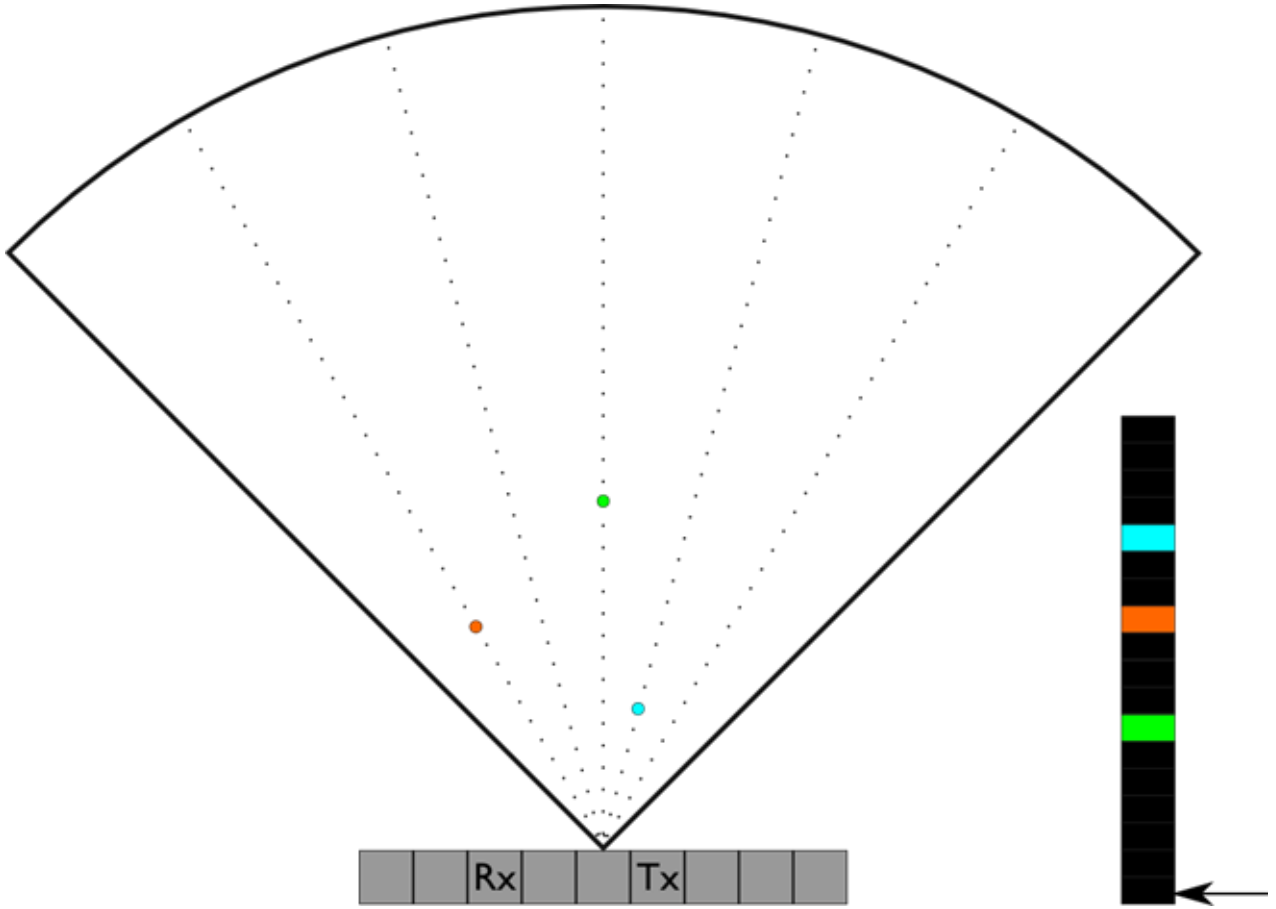
Ultrasound: Transmission and Reception



Ultrasound: Transmission and Reception



Ultrasound: Transmission and Reception



Each transducer stores an array of raw received data

Ultrasound: Transmission and Reception

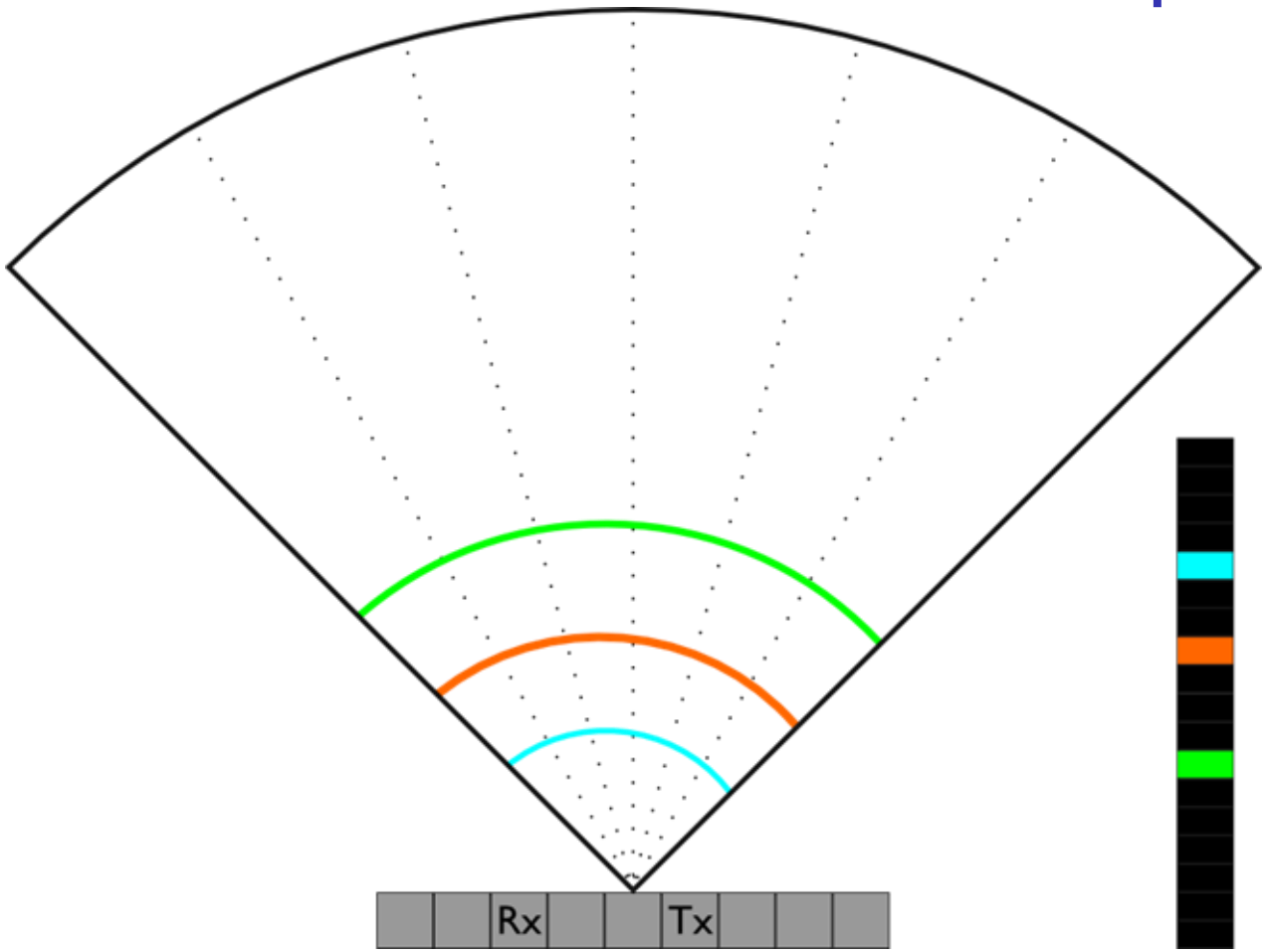
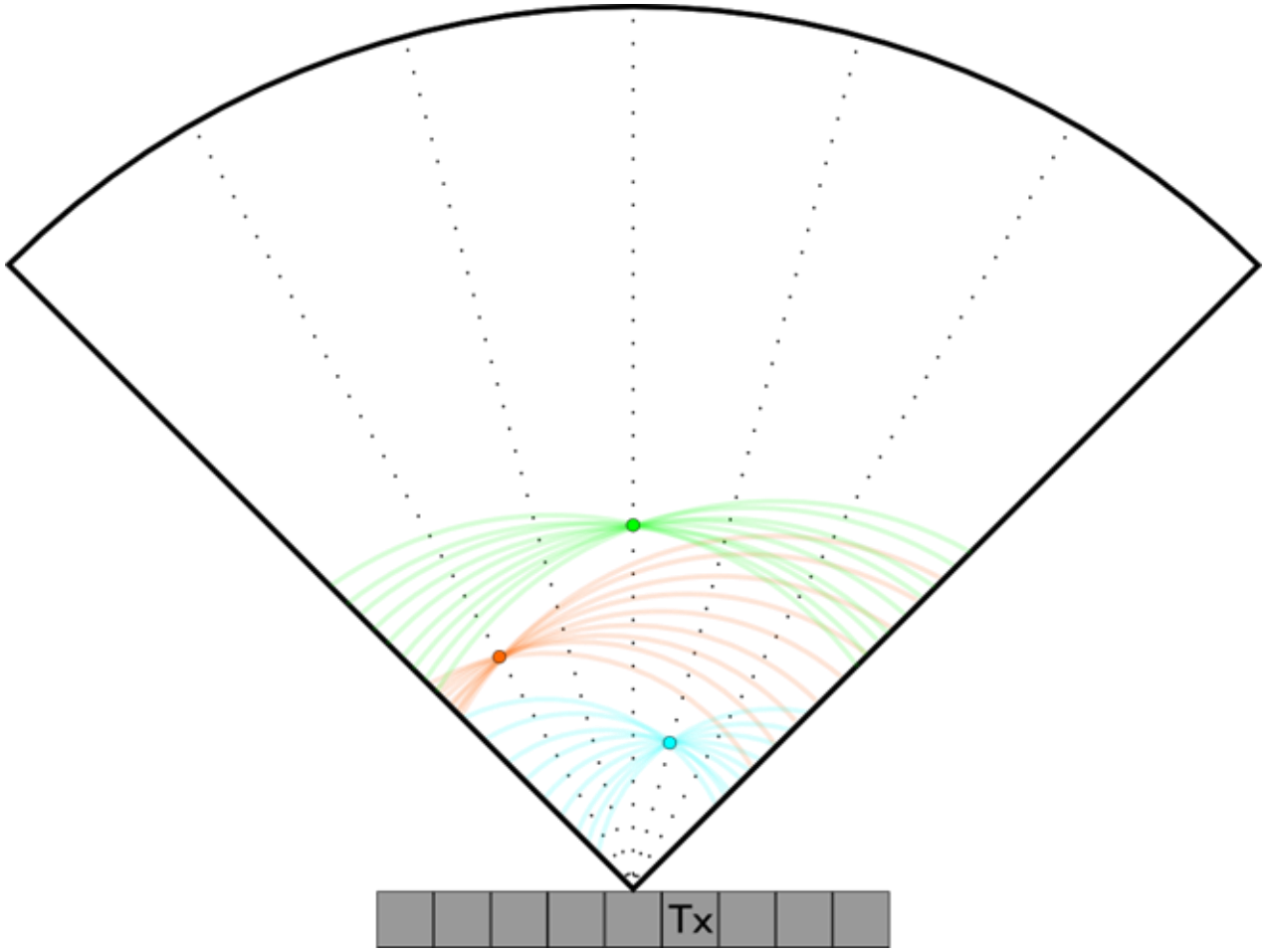


Image reconstructed from data based on round-trip delay

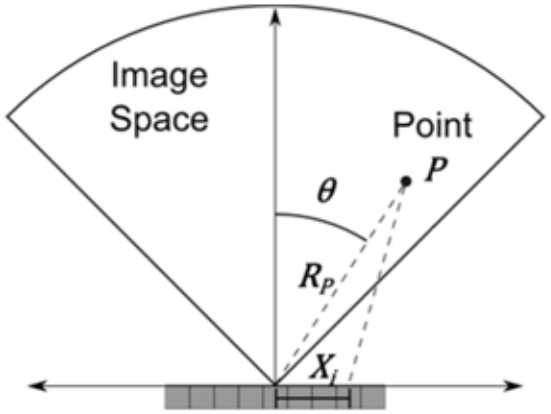
Ultrasound: Transmission and Reception



Images from each transducer combined to produce the full frame

Delay Index Calculation

Iterate through all index points for each transducer and calculate delay index τ_P



$$\tau_P = \frac{f_s}{c}\left(R_p + \sqrt{R_P^2 + X_i^2 - 2R_P X_i \sin\theta}\right)$$

Often done with lookup tables (LUTs)
50 GB LUT required for target 3D system

Parallelism Opportunities

```
/* Now compute reflected distance, find index values, add to image */
for (it_rx = 0; it_rx < trans_x * trans_y; it_rx++) {

    image_pos = image; // Reset image pointer back to beginning
    point = 0; // Reset

    // Iterate over entire image space
    for (it_t = 0; it_t < sls_t; it_t++) {
        for (it_p = 0; it_p < sls_p; it_p++) {
            for (it_r = 0; it_r < pts_r; it_r++) {

                x_comp = rx_x[it_rx] - point_x[point];
                x_comp = x_comp * x_comp;
                y_comp = rx_y[it_rx] - point_y[point];
                y_comp = y_comp * y_comp;
                z_comp = rx_z - point_z[point];
                z_comp = z_comp * z_comp;

                dist = dist_tx[point++] + (float)sqrt(x_comp + y_comp + z_comp);
                index = (int)(dist/idx_const + filter_delay + 0.5);
                *image_pos++ += rx_data[index+offset];
            }
        }
    }
    offset += data_len;
}
```

Pthreads Tutorial

- What is a thread?

Pthreads Tutorial

- What is a thread?
  - Independently executable stream of instructions
  - Schedulable unit of execution for the operating system

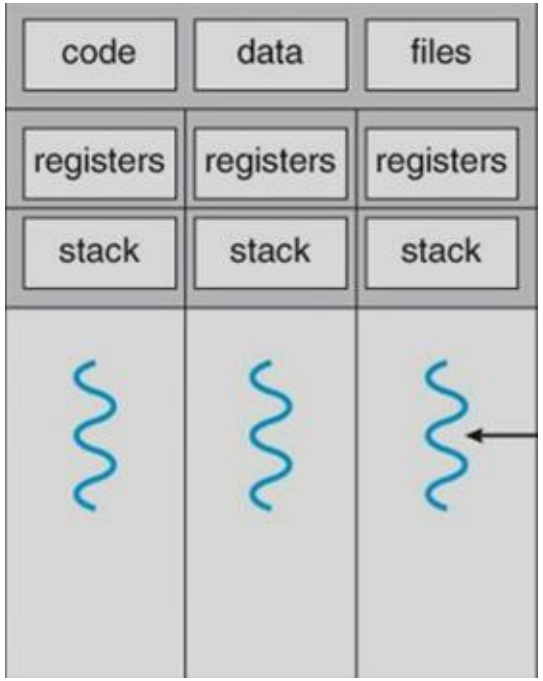
Pthreads Tutorial

- What is a thread?
  - Independently executable stream of instructions
  - Schedulable unit of execution for the operating system
- Pthreads - the POSIX threading interface

Pthreads Tutorial

- What is a thread?
  - Independently executable stream of instructions
  - Schedulable unit of execution for the operating system
- Pthreads - the POSIX threading interface
  - Provides system calls to *create* and *synchronize* threads
  - Communication happens through **shared memory**
    - Specifically using ***pointers*** to shared data

Pthreads Tutorial



[Posix Threads Tutorial]

Optimization Strategies- SIMD

- Utilize vector instructions
- Handle “extra” elements / iterations with scalar code

```
#include <immintrin.h>
__m256 a_vec = _mm256_load_ps(&array[i]);      // Load 8 floats
__m256 b_vec = _mm256_load_ps(&array[i+8]);    // Load next 8 floats
__m256 result = _mm256_sub_ps(a_vec, b_vec);   // Subtract 8 pairs
__m256 squared = _mm256_mul_ps(result, result); // Square 8 values
```

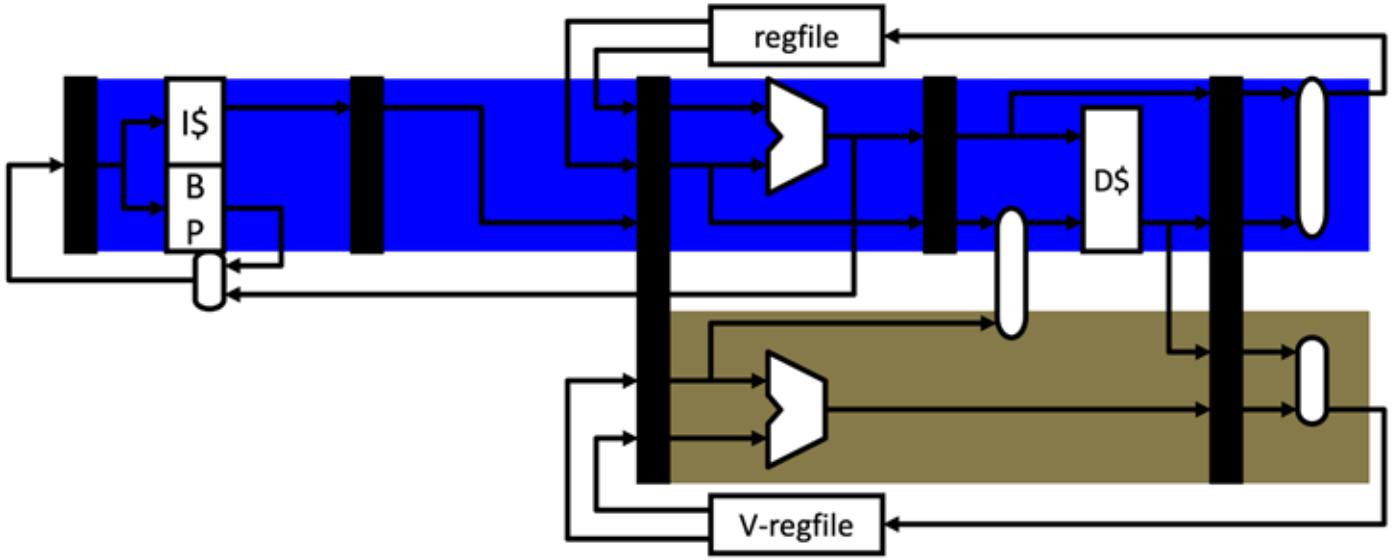
SIMD / Vector Instructions

Concept

- Data-level Parallelism
- **S**ingle **I**nstruction **M**ultiple-**D**ata (**SIMD**)

Implementation

- Vector data types
- Vector registers
- Vector datapath
- **Vector instructions**



Example SIMD instructions

Vector length 4 example

- Load vector: `ldf.v [X+r1]->v1`

  `ldf [X+r1+0]->v1`$_0$
  `ldf [X+r1+1]->v1`$_1$
  `ldf [X+r1+2]->v1`$_2$
  `ldf [X+r1+3]->v1`$_3$

- Add two vectors: `addf.vv v1,v2->v3`

  `addf v1`$_i$`,v2`$_i$`->v3`$_i$ `(where i is 0,1,2,3)`

- Add vector to scalar: `addf.vs v1,f2,v3`

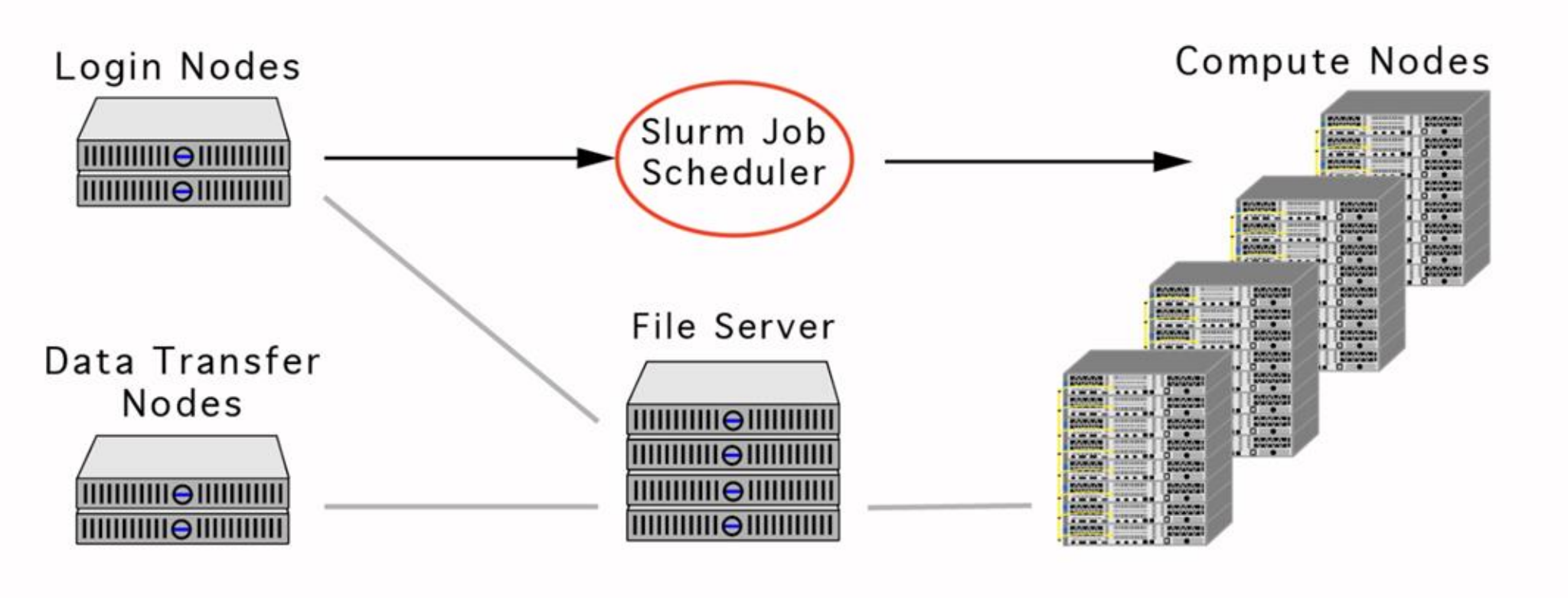
  `addf v1`$_i$`,f2->v3`$_i$ `(where i is 0,1,2,3)`

Setup

- PA1 Handout
- **If you have not requested a Great Lakes login please do so here**
- Submission Requirements
  - 1 paragraph report
  - Optimized beamform.c

```
/scratch/eecs570w26s001_class_root/eecs570w26s001_class/shared_data/PA1
[[ttwomey@gl-login1 PA1]$ tree
.
├── code
│   ├── beamform.c
│   ├── solution_check.c
│   └── submit.sh
├── inputs
│   ├── beamforming_input_16.bin
│   ├── beamforming_input_32.bin
│   └── beamforming_input_64.bin
└── solutions
    ├── beamforming_solution_16.bin
    ├── beamforming_solution_32.bin
    └── beamforming_solution_64.bin
```

Great Lakes


Login Nodes
Slurm Job Scheduler
Compute Nodes
File Server
Data Transfer Nodes

Testing

- **Do not change compilation flags**
- Take advantage of submit.sh
  - Provides exclusive node access
  - #SBATCH --cpus-per-task=36
- Local development (x86) useful for debugging and quick feedback

Note on Grading

From Assignment Description:

- 40% - Correct output (RMS error < 1e-16 per solution_check)
- 50% - Speedup > 8x relative to sequential on Great Lakes
- **10% - Relative performance compared to the best submission**

  your_speedup / max_speedup