ACCELERATED HARRIS CORNER DETECTOR

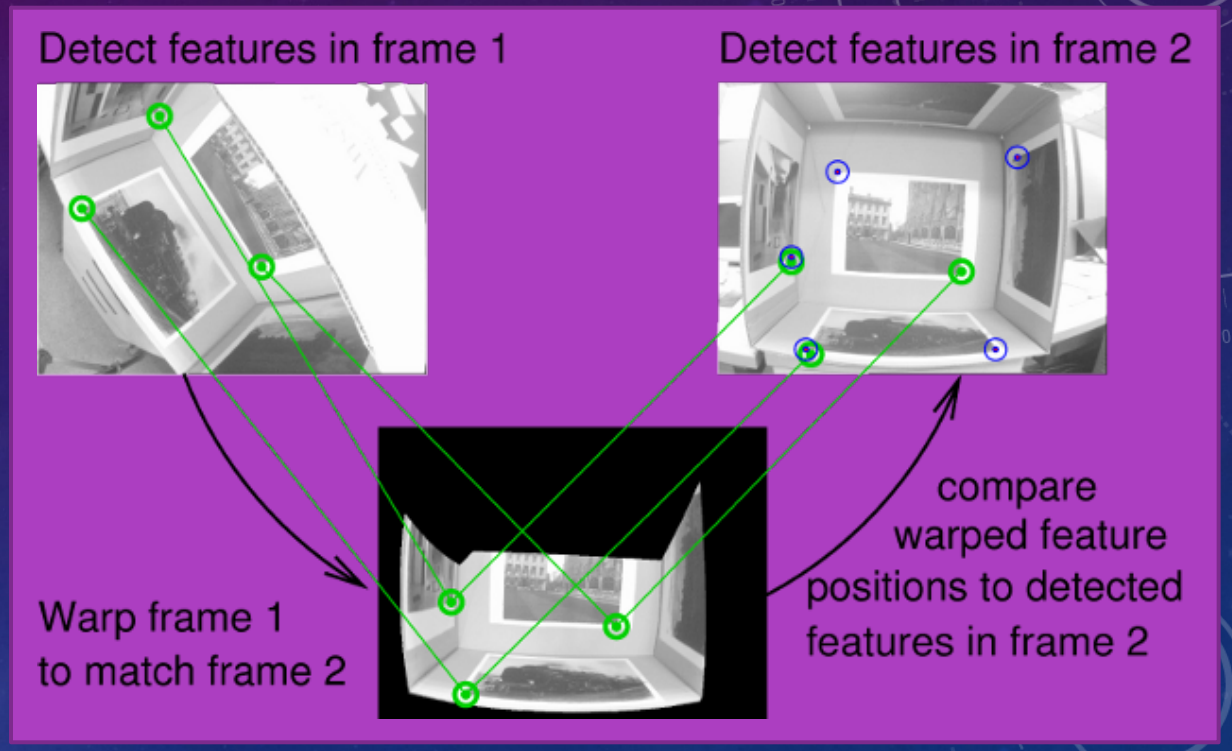
ANDREW TU

EECE5640

SPRING 2019

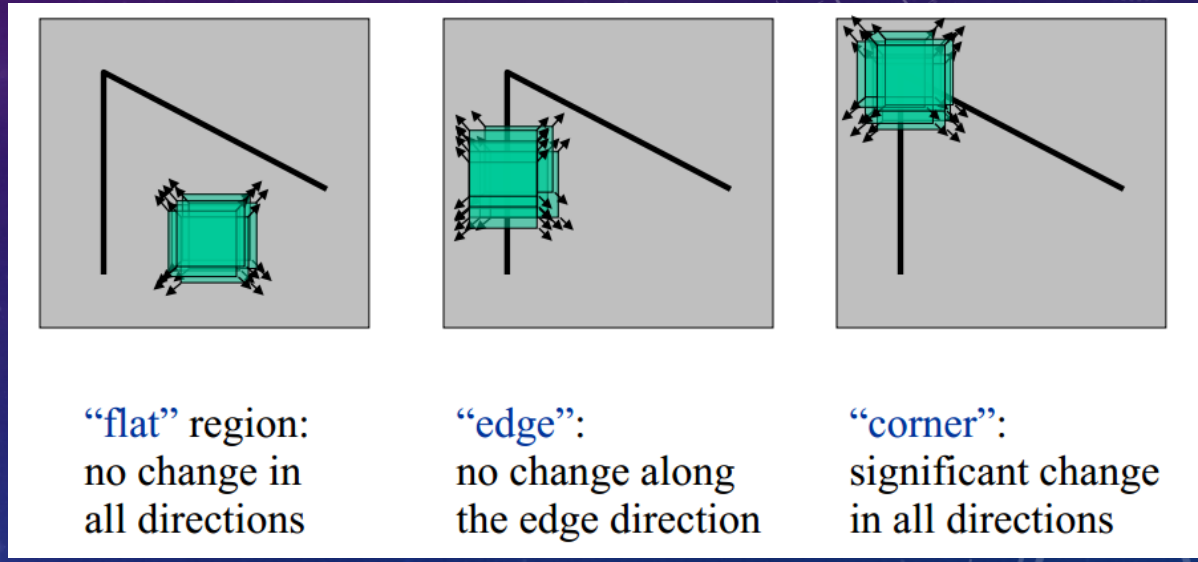
CORNER DETECTORS

- Corners make good features to track
  - Rotation Invariant
  - Translation Invariant
- Useful for matching point between two images



HARRIS CORNER DETECTOR

Find large changes in intensity in BOTH x and y direction



HARRIS CORNER DETECTOR

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

$$R = \det M - k(trace\, M)^2$$

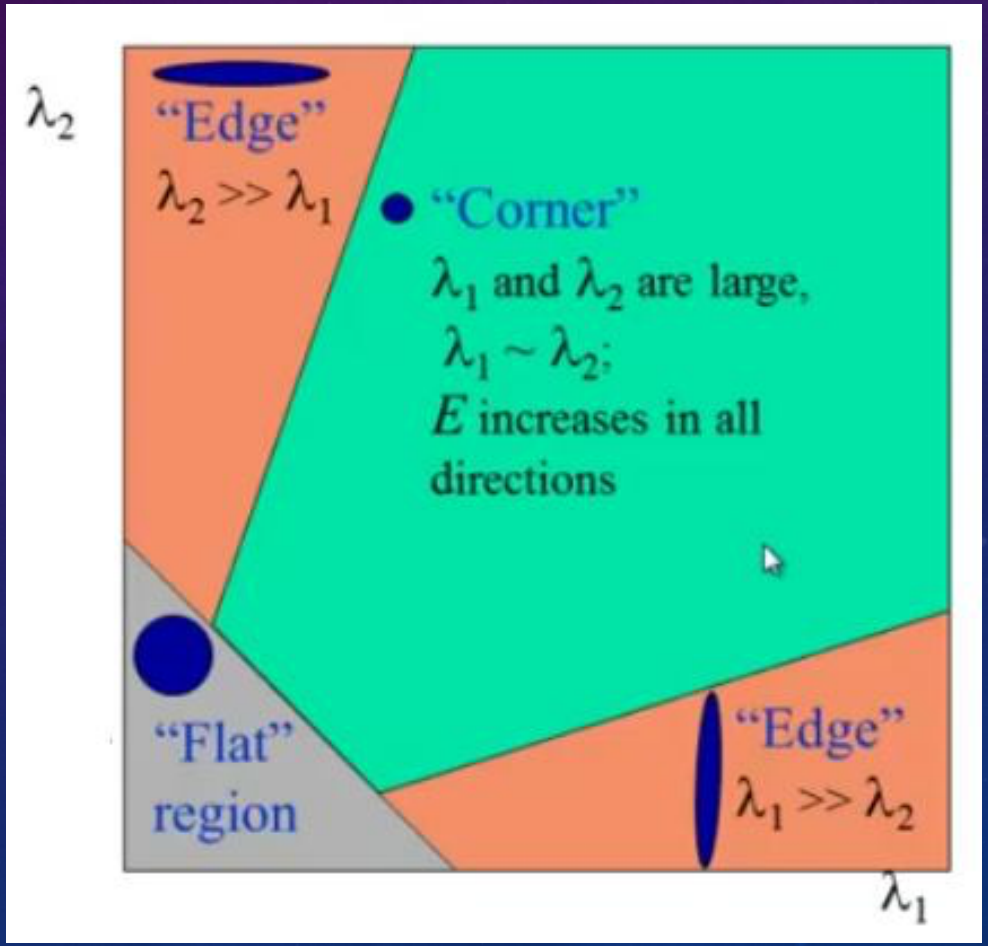
$$\det M = \lambda_1 \lambda_2$$

$$trace\, M = \lambda_1 + \lambda_2$$

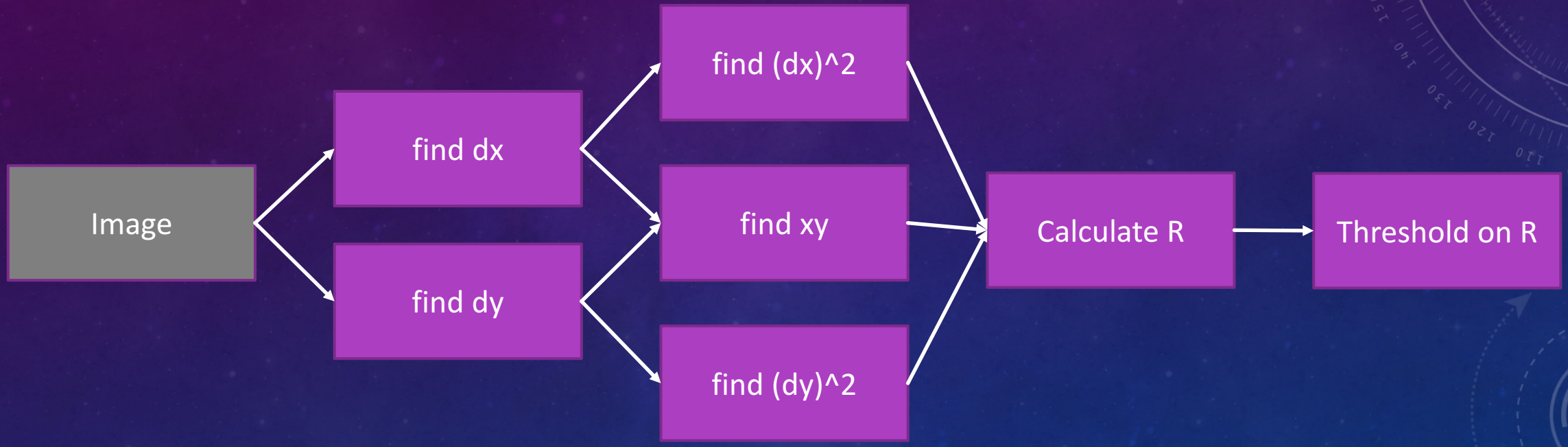
$\boldsymbol{\lambda}_1 = I_x^2$

$\boldsymbol{\lambda}_2 = I_y^2$

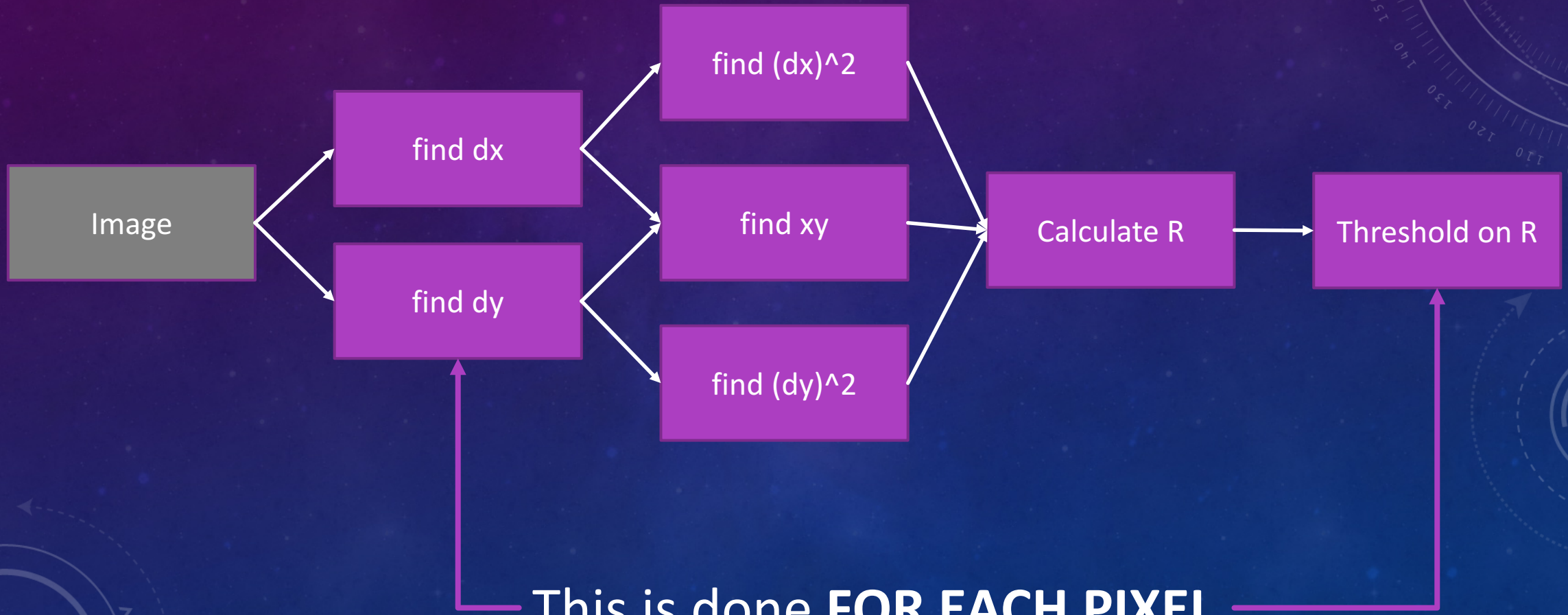
HARRIS CORNER DETECTOR



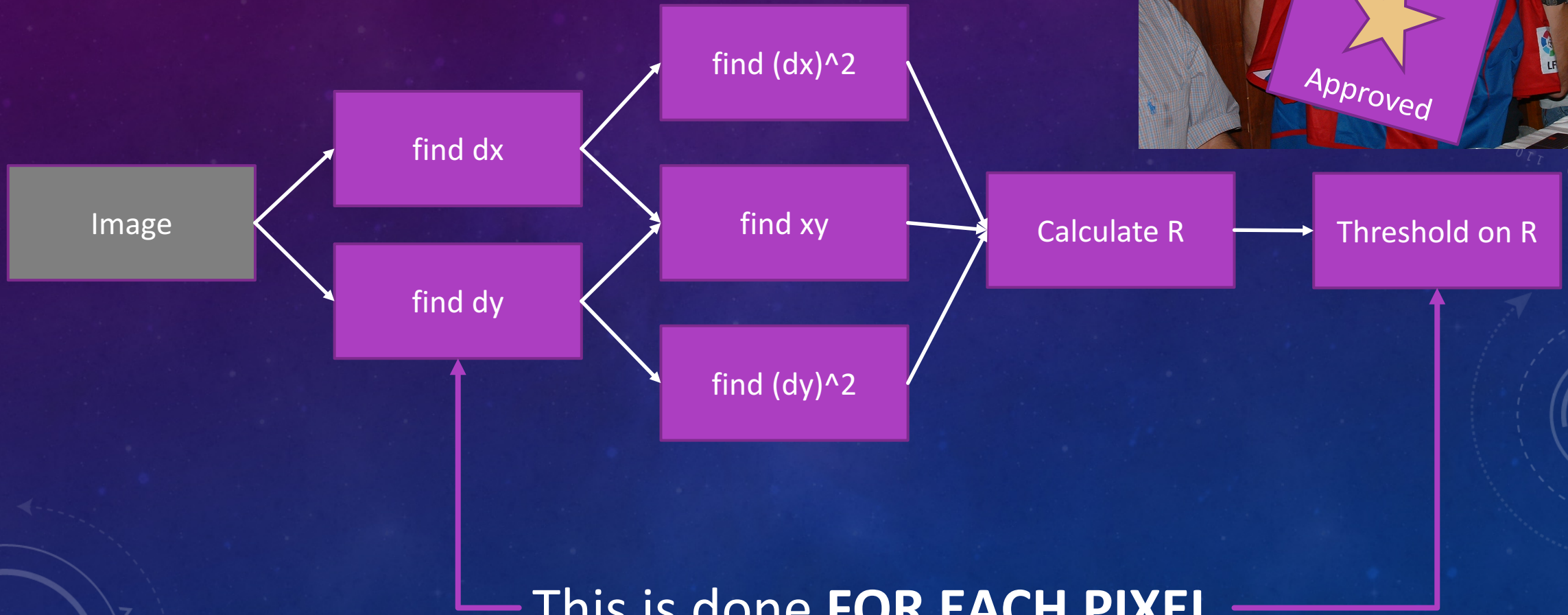
HARRIS CORNER DETECTOR PIPELINE



HARRIS CORNER DETECTOR PIPELINE



HARRIS CORNER DETECTOR PIPELINE



This is done **FOR EACH PIXEL**

PROJECT GOALS

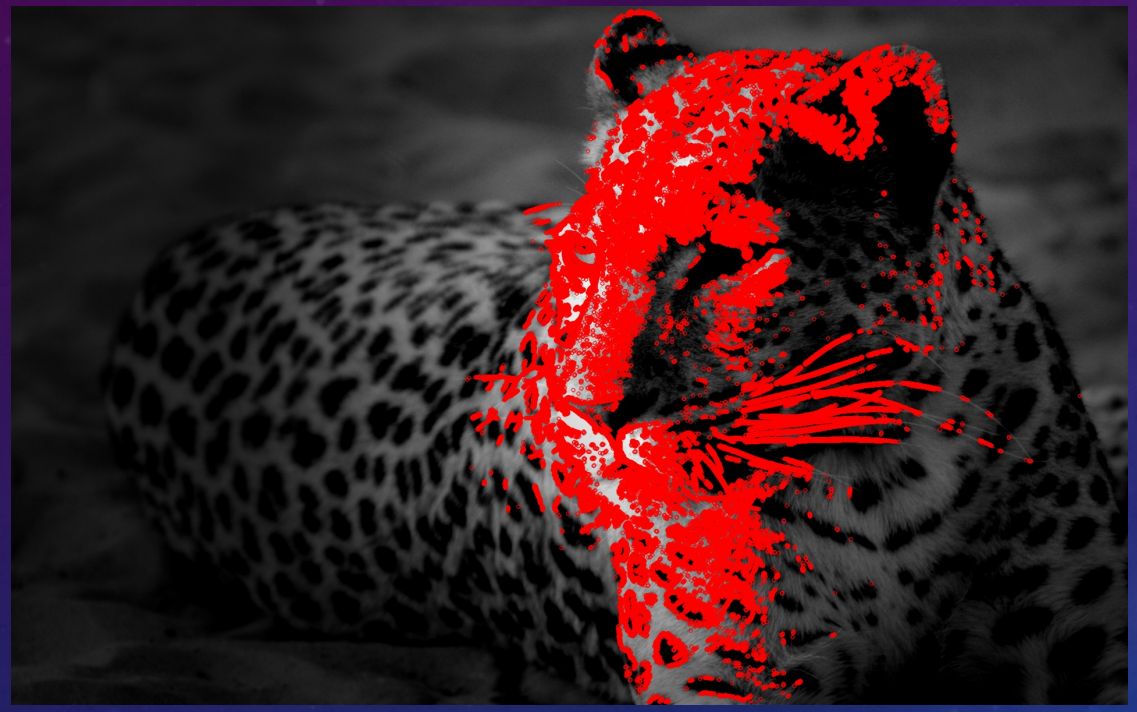
Implementation Goals

1. Implement Harris Corner in C++
2. Accelerate with OpenMP
3. Accelerate with CUDA

Analysis

1. Compare and explain timing results
2. Look into further CUDA Optimizations

PRELIMINARY IMAGE RESULTS

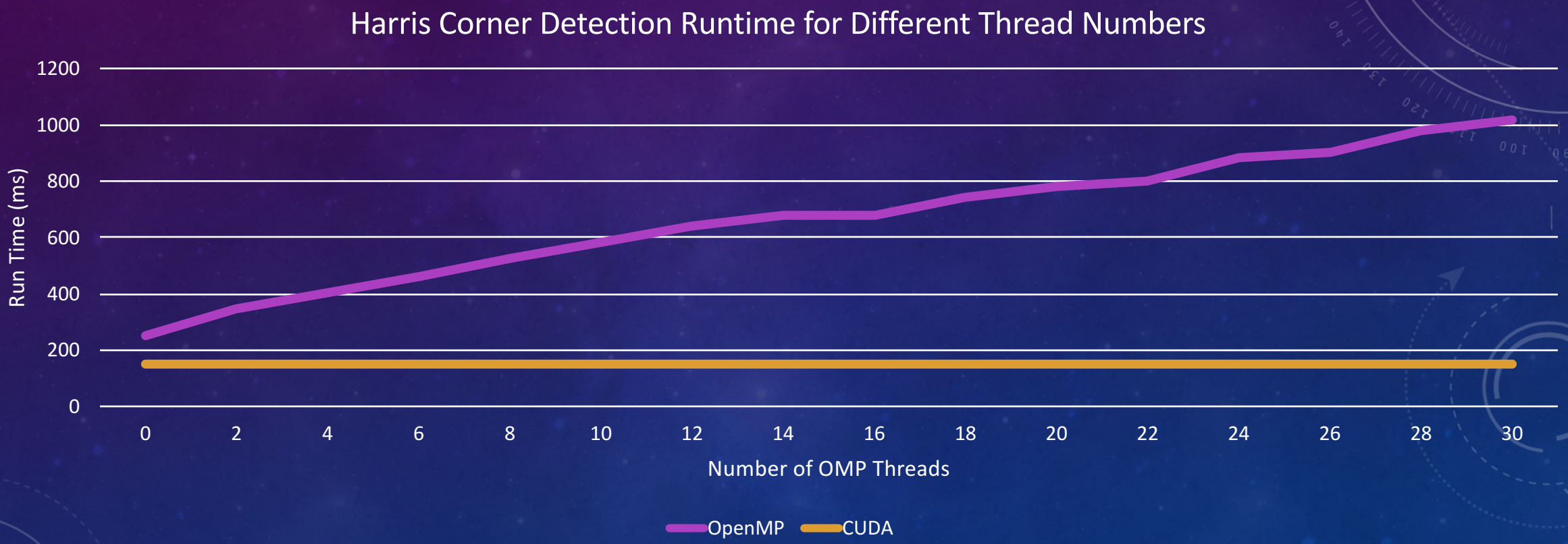


C++ Implementation

(No Non-Max Supression or Gaussian Smoothing)

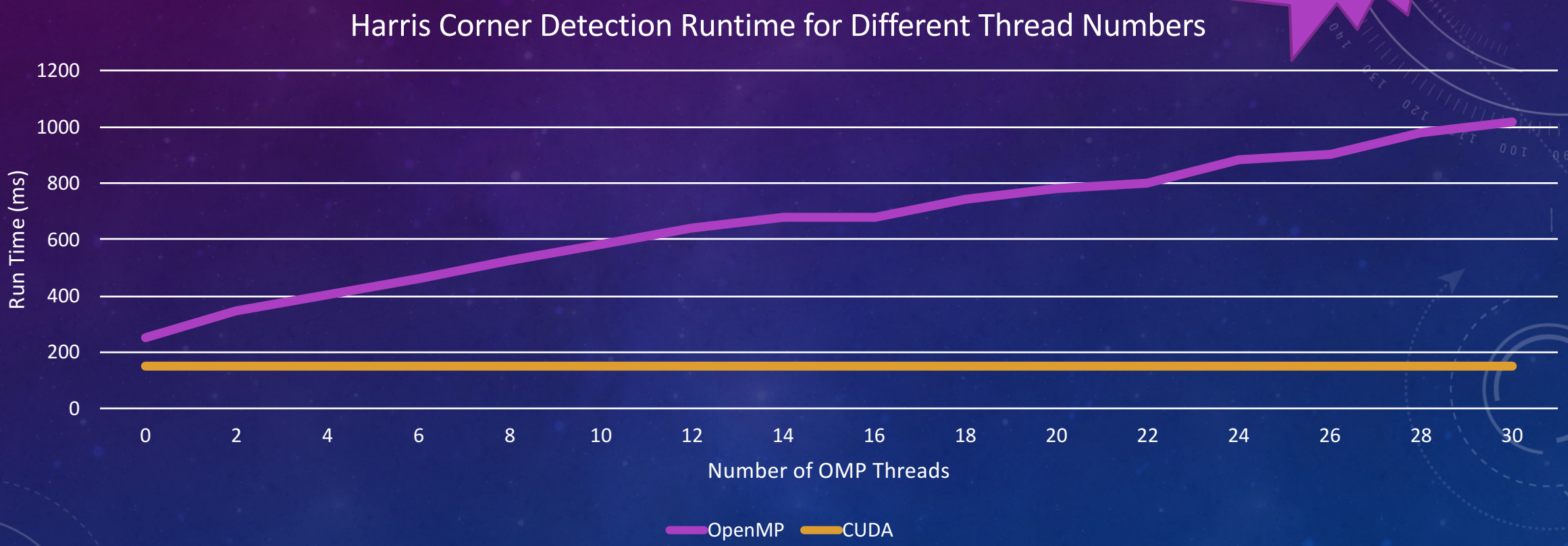
OpenCV “Ground Truth”

PRELIMINARY TIMING RESULTS



PRELIMINARY TIMING RESULTS

????

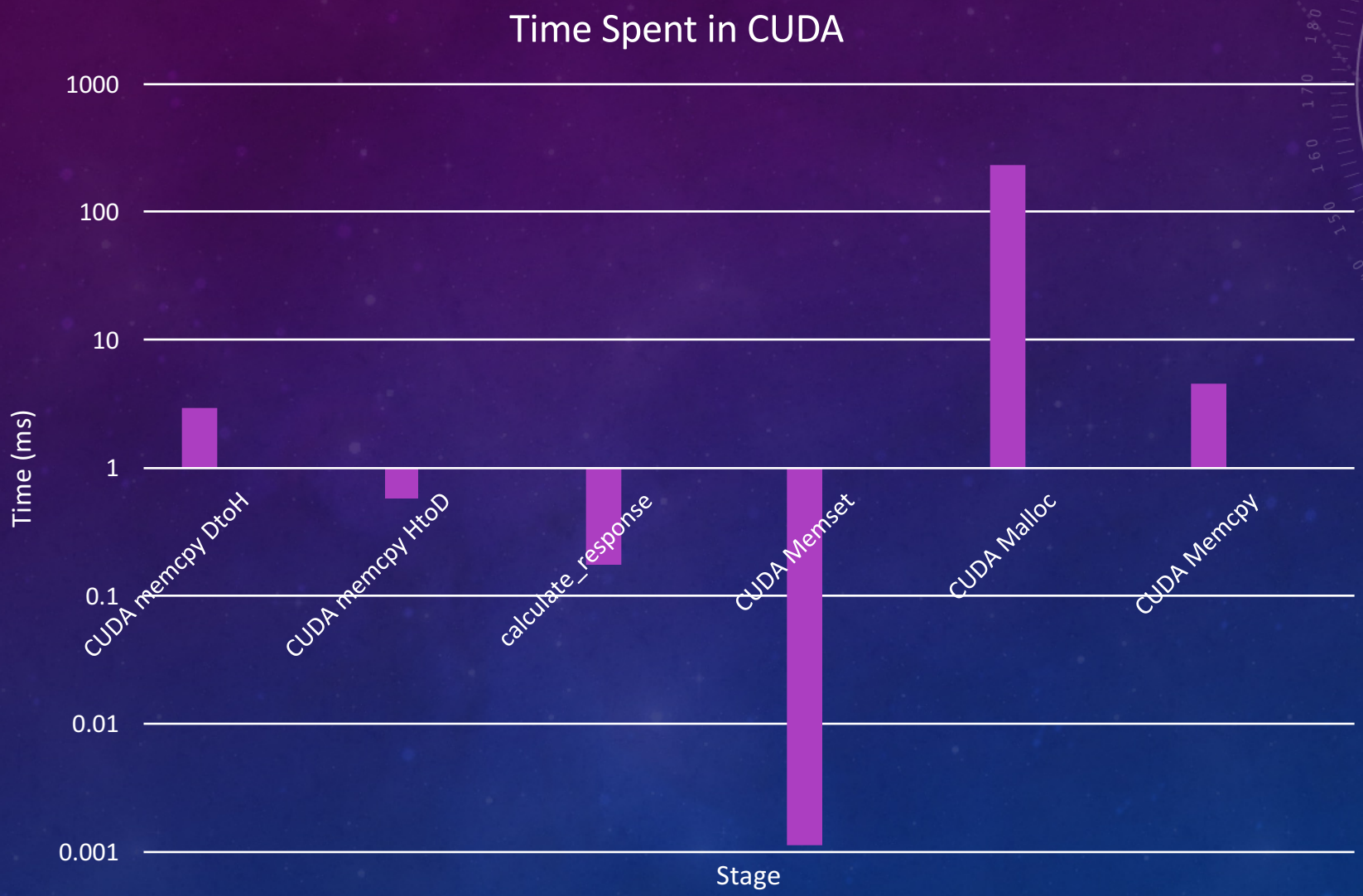


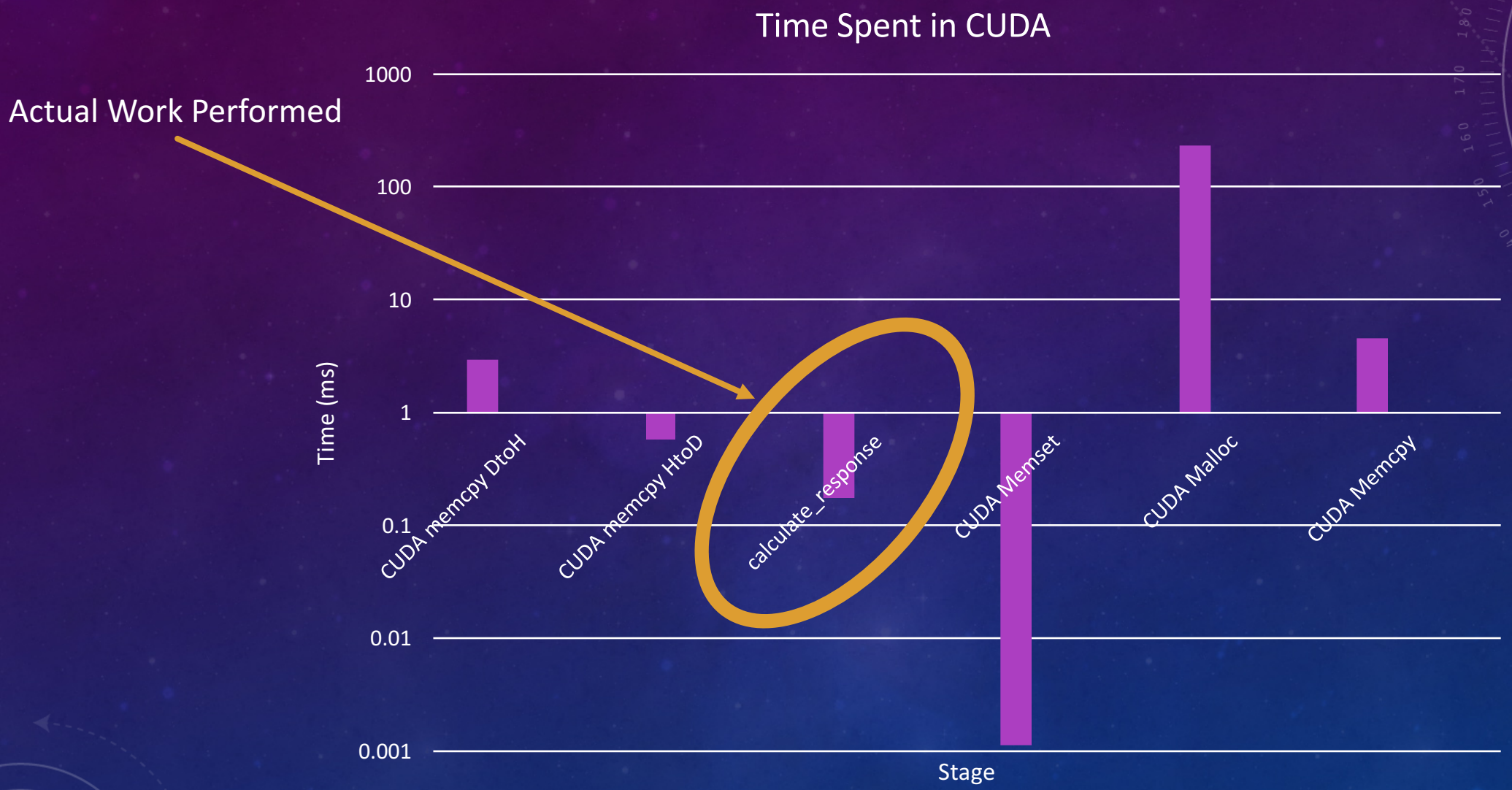
NAÏVE CUDA IMPLEMENTATION

- Performs the entire pipeline except thresholding in a single memory transfer on the GPU
- One thread per pixel (maximize throughput)
- All data is read in and out of GPU Global Memory
- Thresholding is done on the CPU (serially)
- Does not perform gaussian smoothing or non-max supression

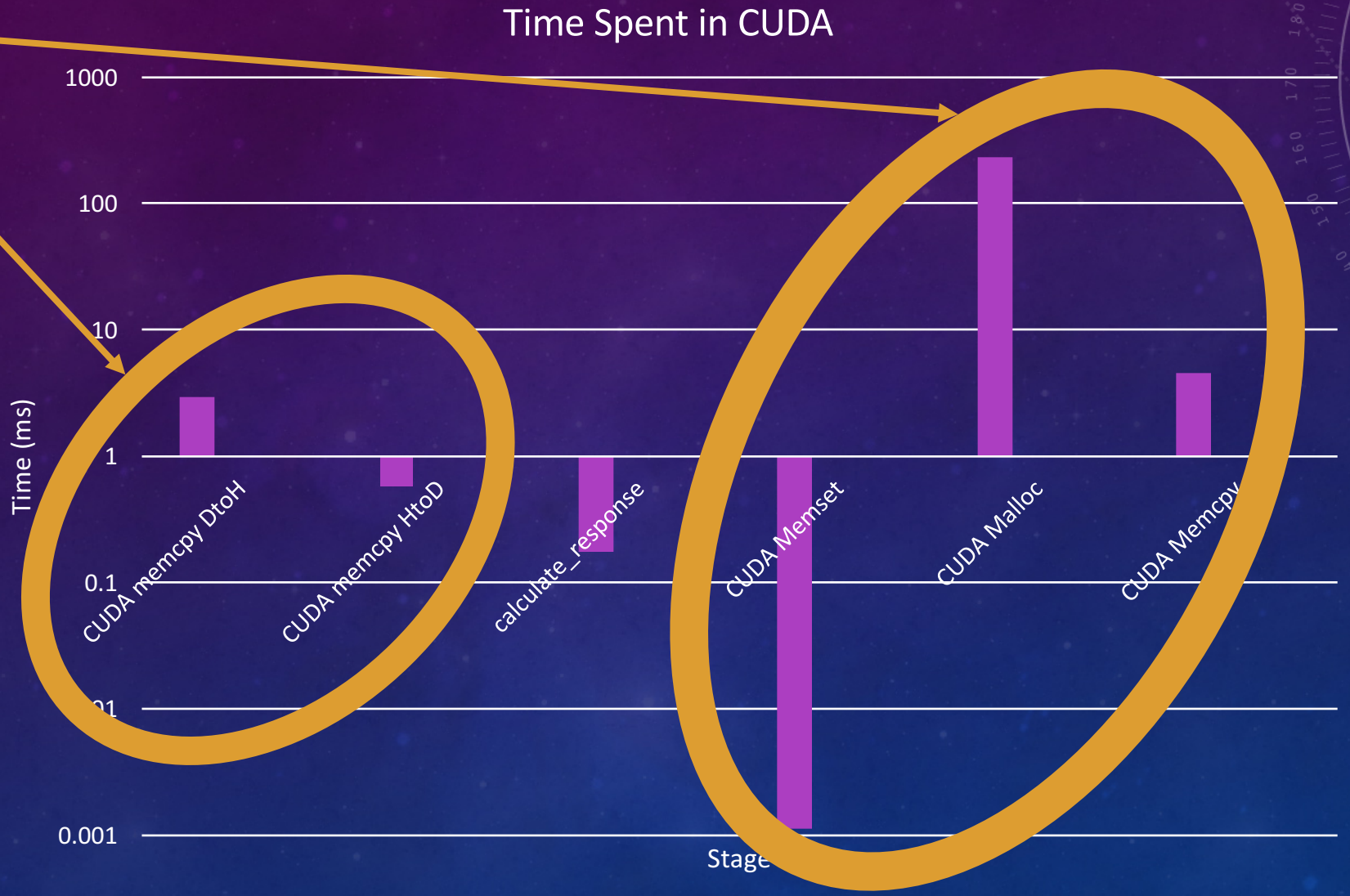
CUDA TIMING RESULTS

```
==3419== Profiling application: ./release/main input/leopard.jpg test.jpg
==3419== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   79.63%  2.9945ms         1  2.9945ms  2.9945ms  2.9945ms  [CUDA memcpy DtoH]
                   15.60%  586.57us         1  586.57us  586.57us  586.57us  [CUDA memcpy HtoD]
                    4.74%  178.40us         1  178.40us  178.40us  178.40us  calculate_response(unsigned
char*, float*, unsigned int, unsigned int)
                    0.03%  1.1520us         1  1.1520us  1.1520us  1.1520us  [CUDA memset]
      API calls:   96.95%  231.19ms         2  115.59ms  225.08us  230.96ms  cudaMalloc
                    1.88%  4.4859ms         2  2.2429ms  928.38us  3.5575ms  cudaMemcpy
                    0.54%  1.2868ms         2  643.42us  223.79us  1.0631ms  cudaFree
                    0.27%  643.56us        94  6.8460us     352ns  260.15us  cuDeviceGetAttribute
                    0.12%  296.21us         2  148.10us  125.57us  170.64us  cudaDeviceSynchronize
                    0.09%  204.68us         1  204.68us  204.68us  204.68us  cuDeviceTotalMem
                    0.08%  193.80us         1  193.80us  193.80us  193.80us  cudaLaunch
                    0.03%  74.930us         1  74.930us  74.930us  74.930us  cudaMemset
                    0.03%  70.885us         1  70.885us  70.885us  70.885us  cuDeviceGetName
                    0.00%  9.0930us         3  3.0310us     580ns  6.8550us  cuDeviceGetCount
                    0.00%  5.2110us         4  1.3020us     258ns  3.5890us  cudaSetupArgument
                    0.00%  3.5600us         2  1.7800us     550ns  3.0100us  cuDeviceGet
                    0.00%  2.0400us         1  2.0400us  2.0400us  2.0400us  cudaConfigureCall
                    0.00%     828ns         1     828ns     828ns     828ns  cudaPeekAtLastError
```


Time Spent in CUDA
1000
100
10
1
0.1
0.01
0.001
Time (ms)
CUDA memcpy DtoH
CUDA memcpy HtoD
calculate_response
CUDA Memset
CUDA Malloc
CUDA Memcpy
Stage


Time Spent in CUDA
Actual Work Performed
1000
100
10
1
0.1
0.01
0.001
Time (ms)
CUDA memcpy DtoH
CUDA memcpy HtoD
calculate_response
CUDA Memset
CUDA Malloc
CUDA Memcpy
Stage

Overhead Costs!



FUTURE WORK

- Look at smarter ways of dividing up work on GPU
- Explore slow down in OpenMP
- Asynchronous transfers & convolutions
- Use of Constant + Texture Memory
- Expand Harris Corner Implementation for more robust execution