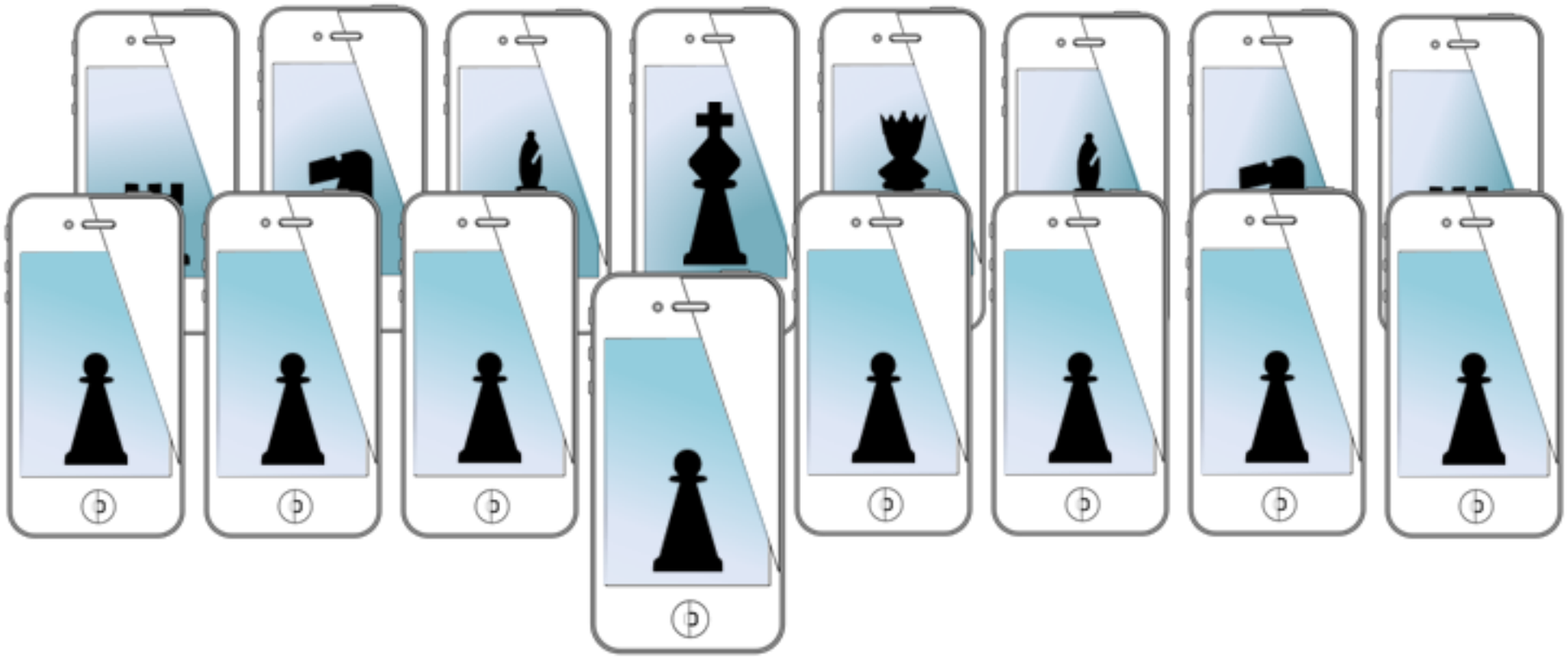
MOBILE SENSING LEARNING & CONTROL



CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture twelve: computer vision

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University

course logistics

- A2 grades will be up today
- A3 is due at 6PM!
- A4 is due the Friday after spring break (~3 weeks)

Module A

Create an iOS application using the CoreImage template that:

- Reads and displays images from the camera in real time
- Highlights multiple faces in the scene
- Highlights eye and mouth position
  - hint: could use another filter for highlights
- (extra credit, up to 0.25 points) displays if the user is smiling or blinking (and with which eye)

Verify the functionality of the application to the instructor during lab time or office hours (or scheduled via email).

Module B

Create an iOS application using the iOpenCV template that:

- Uses video of the user's finger (with flash on) to sense a single dimension stream indicating the "redness" of the finger
- Uses the redness to measure the heart rate of the individual (coarse estimate)
- (optional, NOT extra credit) Display an estimate of the PPG signal

Verify the functionality of the application to the instructor during lab time or office hours (or scheduled via email).

agenda

- video processing
- computer vision
  - face detection
  - OpenCV in iOS

updating filter parameters

- can be done on the fly, without performance loss

```
radius = 100;
self.center = [CIVector vectorWithX:self.view.bounds.size.height/2.0-radius/2
                                  Y:self.view.bounds.size.width/2.0+radius/2];
```

init

```
CIFilter *filter = [CIFilter filterWithName:@"CIPinchDistortion" keysAndValues:
                                            @"inputRadius", @(radius),
                                            @"inputCenter", self.center,
                                            @"inputScale", @0.5,
                                            nil];
```

apply

```
__weak typeof(self) weakSelf = self;
[self.captureManager setProcessBlock:^(CIImage *cameraImage){

    [filter setValue:weakSelf.center forKey:@"inputCenter"];
    [filter setValue:cameraImage forKey:kCIInputImageKey];
    cameraImage = filter.outputImage;
    return cameraImage;

}];
```

updating filter parameters

- update from the UI

setup when users drags

get drag location

```
- (IBAction)panGestureRecognized:(UIPanGestureRecognizer *)sender {
    CGPoint pt = [sender locationInView:self.captureManager.videoPreviewView];
    self.center = [CIVector vectorWithX:pt.x-radius/2
                                  Y:self.captureManager.videoPreviewView.bounds.size.height-pt.y+radius/2];
}
```

update center

VideoAnalgesic Mirrors

filter param demo

- LookinLive with distortion

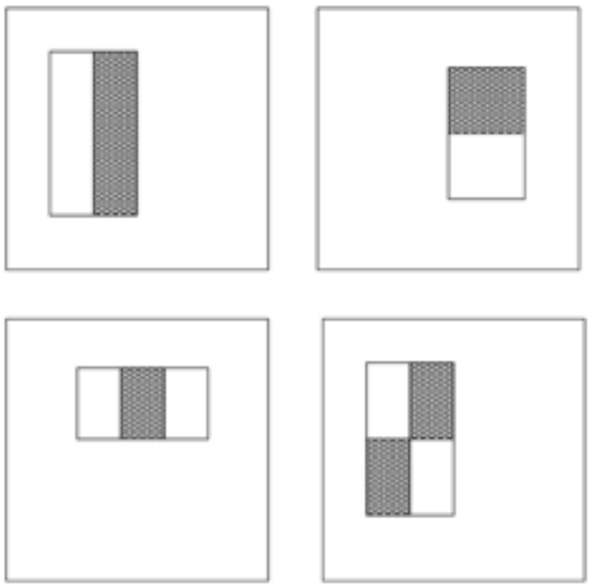
face detection

- is a face in the picture and where?
- algorithm is probably hardware accelerated variant of Viola Jones
- essentially, a “matching” filter is applied
- only happens in one orientation
- but multiple scales (which takes “some” time)



an intuition

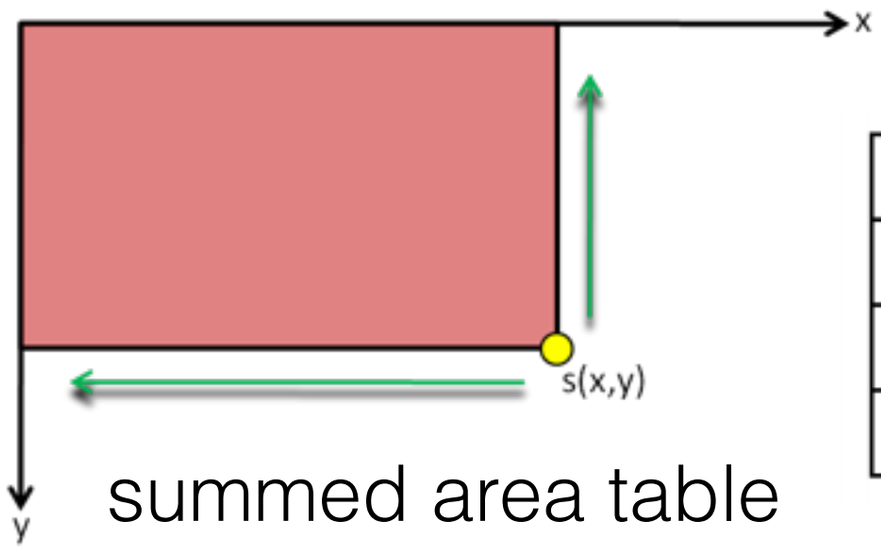
- face detection with “rectangle” features



feature value =
sum of pixels in white area -
sum of pixels in black area

“best” dark and light rectangles already chosen for face detection!

sum of any rectangle= C-D-B+A



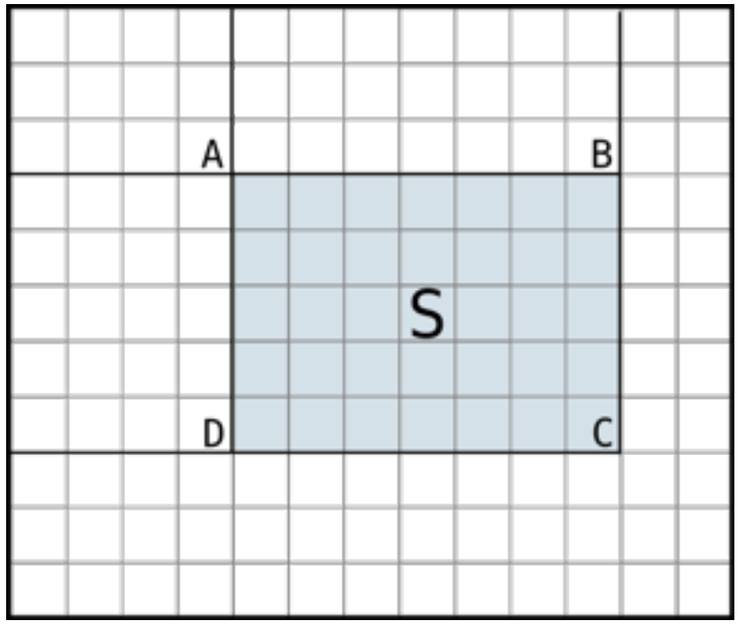
summed area table

4	1	2	2
0	4	1	3
3	1	0	4
2	1	3	2

original

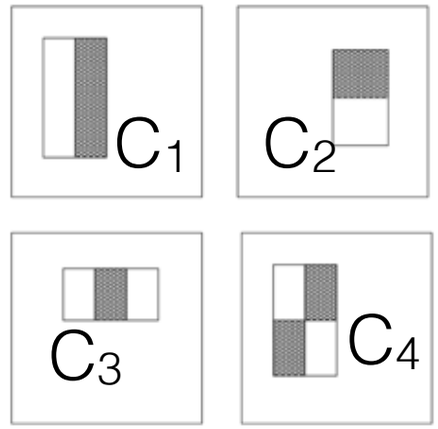
4	5	7	9
4	9	12	17
7	13	16	25
9	16	22	33

summed



learning

- train a bunch of “classifiers” with lots of examples



$$\underbrace{C_t(x)}_{\text{classifier}} = \underbrace{1, \text{ if } f_t > \theta_t}_{\text{feature above thresh}}$$

combine output of classifiers

cascade these

$$\underbrace{C(x)}_{\text{ensemble}} = \underbrace{1, \text{ if } \sum_{t=0}^{T-1} \alpha_t C_t(x) > \frac{1}{2} \sum_{t=0}^{T-1} \alpha_t}_{\text{learned weights}}$$

learning

- tough to train
  - need examples in various lighting and illumination
  - different poses
  - different genders, races, and scales
  - what made this easier?
- easy to use once trained
  - just getting integral image
  - then getting relevant “features”
  - multiply with learned weights!
- iOS already has done the training for you

face detection iOS

- similar pipeline to applying a filter
- use an instance of the CIDetector class
- specify where the processing should

specify options

currently only faces can be detected

```
NSDictionary *opts = @{ CIDetectorAccuracy : CIDetectorAccuracyLow};
CIDetector *detector = [CIDetector detectorOfType:CIDetectorTypeFace
                                          context:self.captureManager.ciContext
                                          options:opts];

opts = @{ CIDetectorImageOrientation : [VideoAnalgesic
    ciOrientationFromDeviceOrientation:[UIApplication sharedApplication].statusBarOrientation]};
```

context

for each face

```
  NSArray *features = [detector featuresInImage:cameraImage options:opts];

  for (CIFaceFeature *f in features)
  {
      NSLog(@"%@",NSStringFromCGRect(f.bounds));
  }
```

do this

options specific to "run"

face demonstration

- ThroughTheLookingGlass

face detection

- many tracking mechanisms are supported
- eye location
- mouth location
- smile detection
- blink / wink detection for each eye
- all use a variant of the Haar Wavelet method

Module A

Create an iOS application using the CoreImage template that:

- Reads and displays images from the camera in real time
- Highlights multiple faces in the scene
- Highlights eye and mouth position
    - hint: could use another filter for highlights
- (extra credit, up to 0.25 points) displays if the user is smiling or blinking (and with which eye)

Verify the functionality of the application to the instructor during lab time or office hours (or scheduled via email).

computer vision

- face detection is just the beginning
  - could use tracking method for any object
- could also do “recognition”
  - typically done with eigen-faces or fisher-faces
  - would take (slightly) too much time in this class to implement
- more than just tracking
  - edge detection
  - finding lines and shapes
  - color space transformations
- extract “knowledge” from a scene

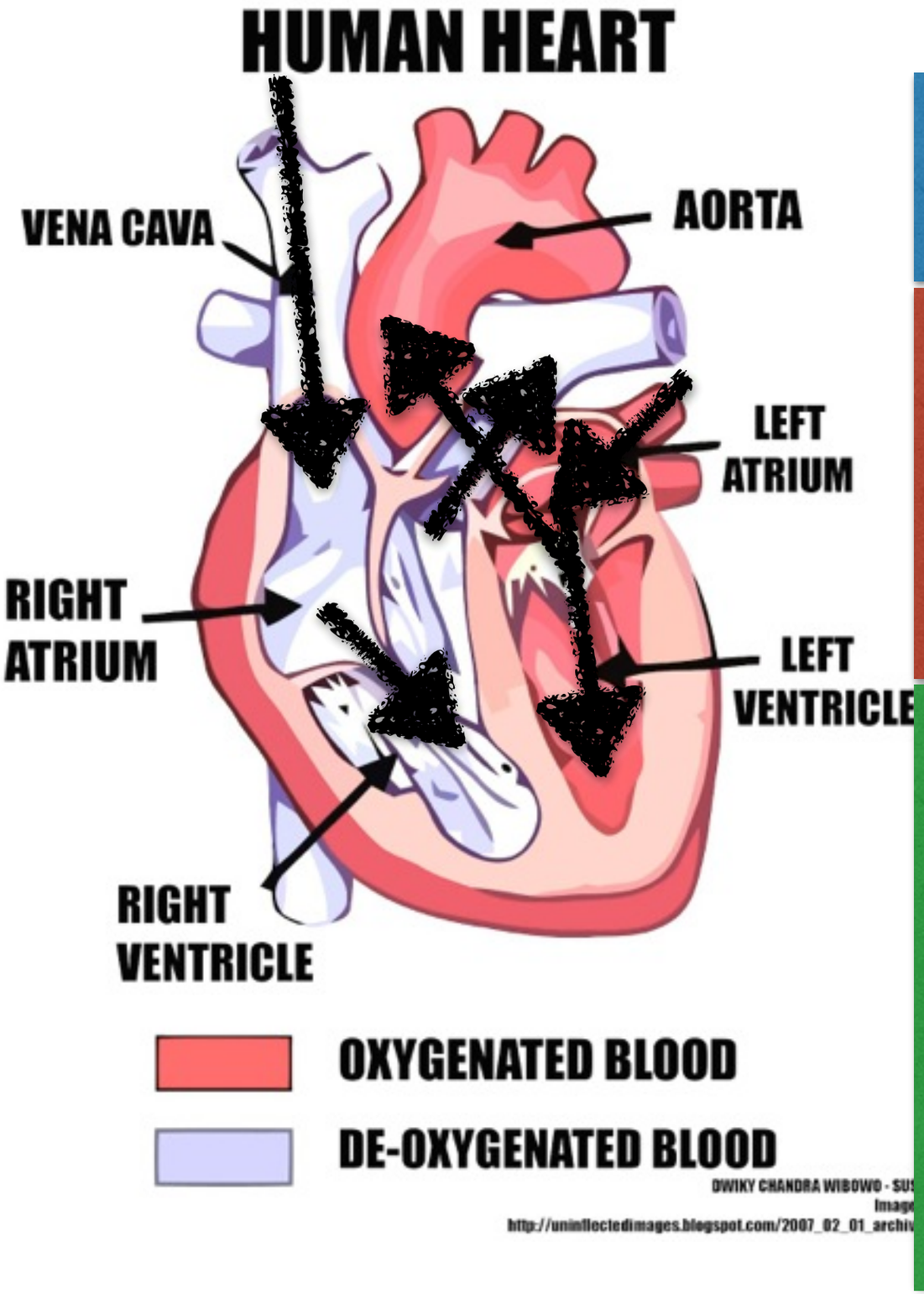
computer vision in iOS

- mobile camera is a rich medium for:
  - enhancement
  - interaction
  - augmented re
    - gaming
    - tracking
  - health

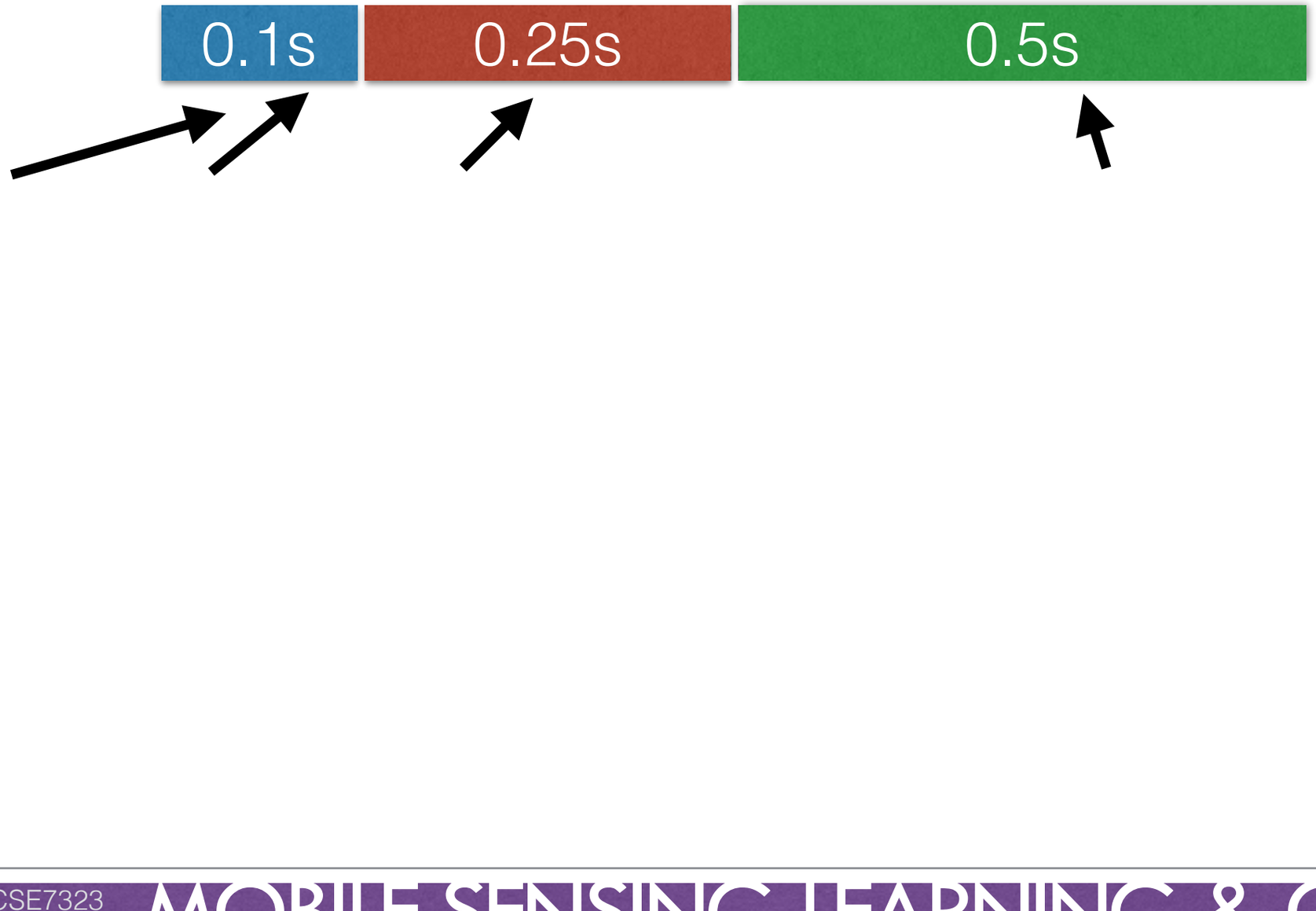
health?

- detecting heart rate from the camera
- what is the function of the heart?
  - pump oxygenated blood from lungs to the rest of the body
  - bring back de-oxygenated blood
- a pump maintains pressure and flow
  - no pump works continuously
  - series of pressure buildup, release, buildup , release
  - cycles in the heart is the **heart beat**

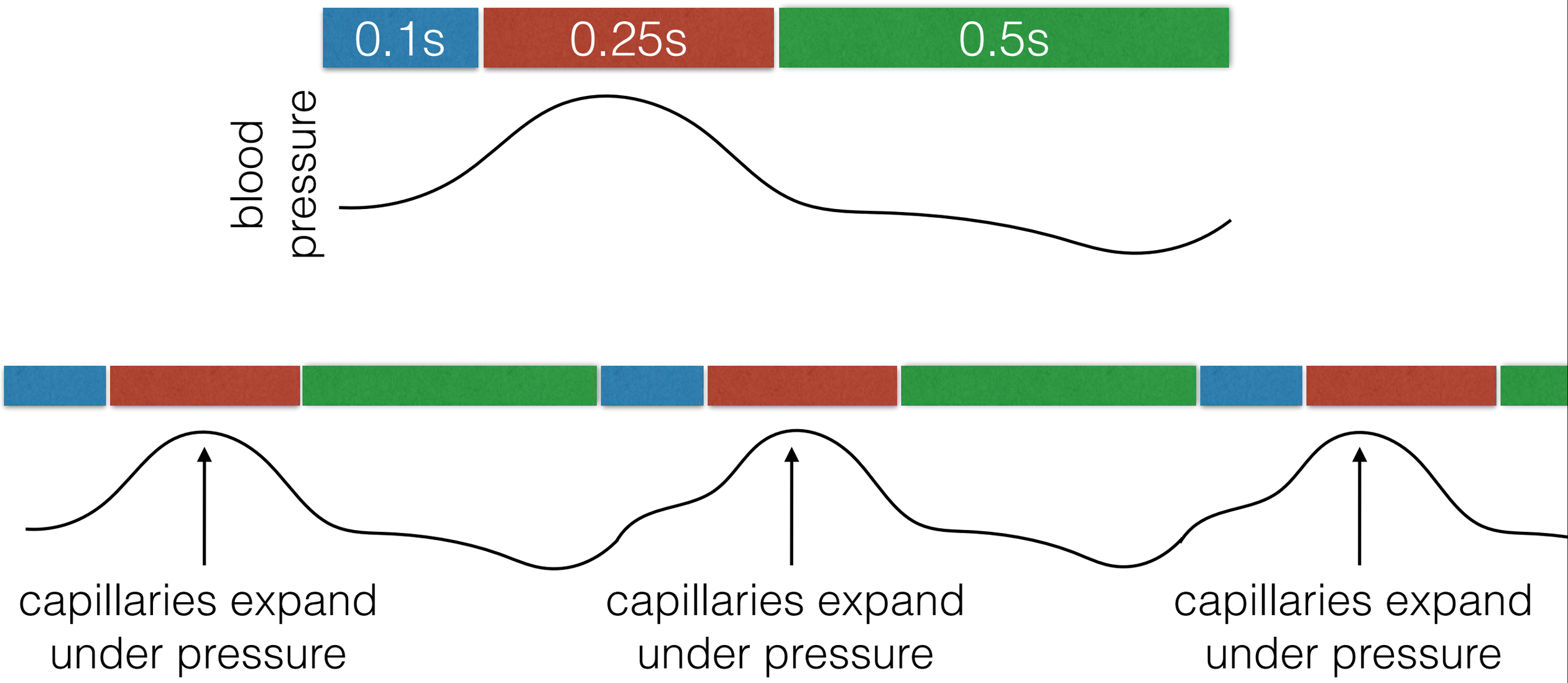
the cardiac cycle



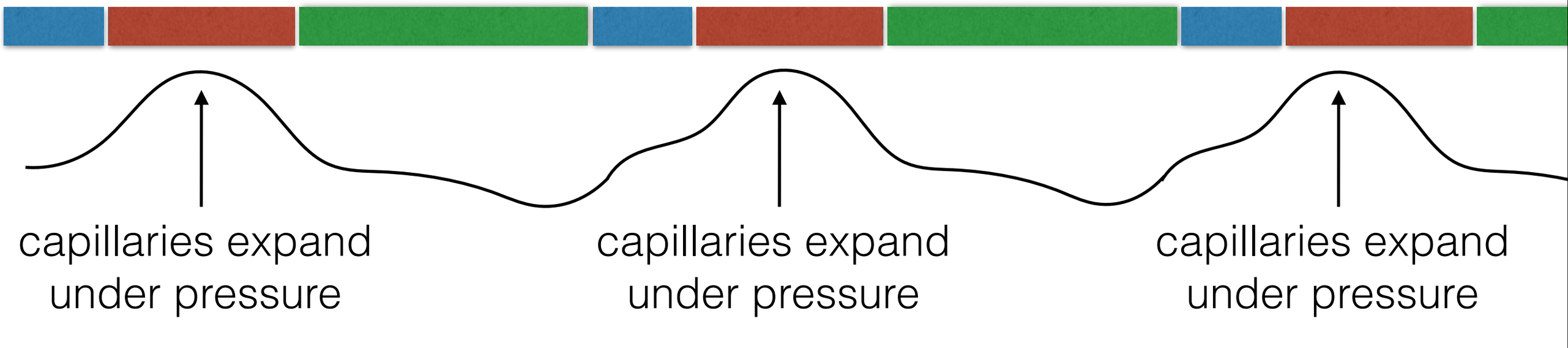
- aortic valve and vena cava valve open
- blood enters ventricles
- pump of heart ejects blood into arteries and out to lungs/body
- heart relaxes, letting blood flow into atria and ventricles


0.1s
0.25s
0.5s

a signal from the heart



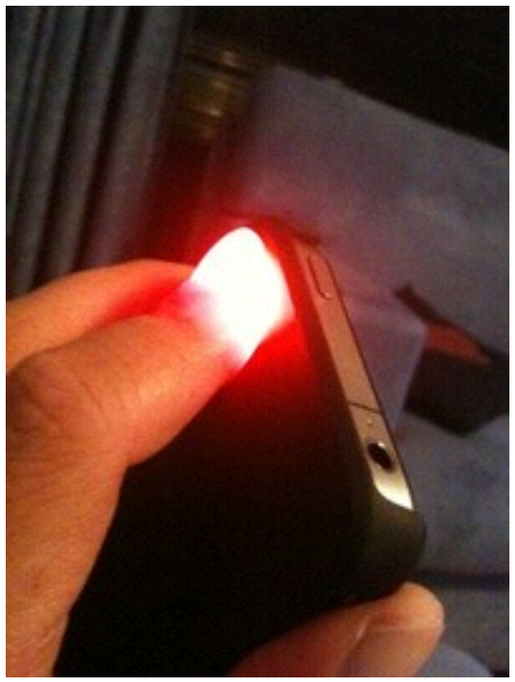
a signal from the heart

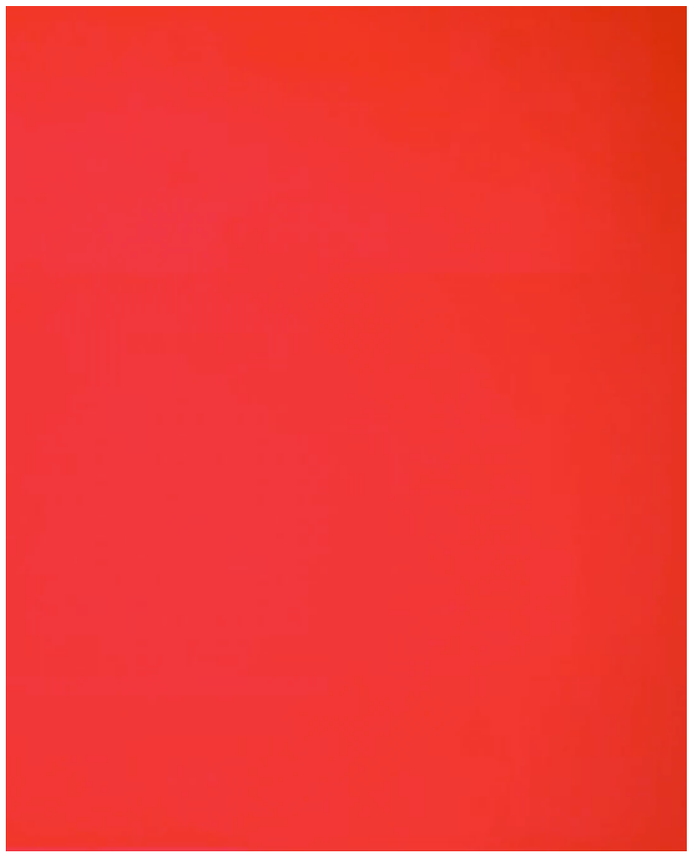


- capillary expansion means more blood under skin
  - shift in redness from oxygenated blood
  - shift in blueness from deoxygenated blood
  - more blood molecules for light to reflect from

a signal from the heart

- hold finger over
  - camera
  - torch (always on flash)





photoplethysmography (PPG)

caveats

- do not press too hard on camera
- vasoconstriction and vasodilation
- bigger surface areas are better
- don’t move around too much
- the heart is not the only organ that increases pressure
  - what else could cause the capillaries to expand/contract?
- what method might you use to measure PPG from the camera?

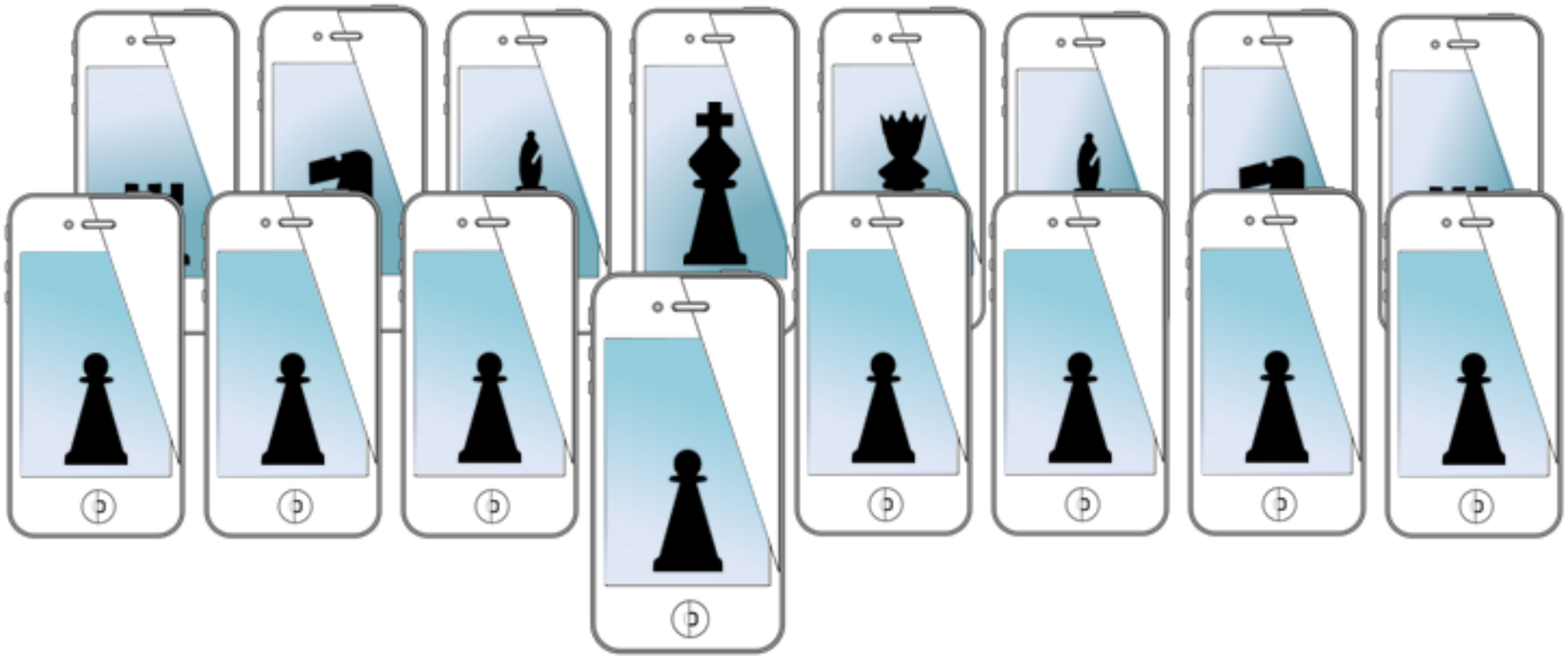
OpenCV in iOS

- open computer vision library
- released by intel
- many common functions are implemented
- written in c++, but has many wrappers
  - EMGU for .NET (c#,VC++,etc.), pycv2 for python, Java API for Android, and many, many more
- some hardware accelerations on iOS
  - not as many as core image
  - expect slightly slower processing, but still fast!

for next time…

- more computer vision with OpenCV
  - fun operations in imaging

MOBILE SENSING LEARNING & CONTROL



CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture twelve: computer vision

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University