SLAC-WP-102

X-Ray Detector Simulations

Adrienne Tiña

Office of Science, Science Undergraduate Laboratory Internship (SULI) Program

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internship (SULI) program, under Contract No. DE-AC02-76SF00515.

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

Oral Presentation Abstract:
X-Ray Detector Simulations

The free-electron laser at LCLS produces X-Rays that are used in several facilities. This light source is so bright and quick that we are capable of producing movies of objects like proteins. But making these movies would not be possible without a device that can detect the X-Rays and produce images. We need X-Ray cameras. The challenges LCLS faces include the X-Rays' high repetition rate of 120 Hz, short pulses that can reach 200 femto-seconds, and extreme peak brightness. We need detectors that are compatible with this light source, but before they can be used in the facilities, they must first be characterized. My project was to do just that, by making a computer simulation program. My presentation discusses the individual detectors I simulated, the details of my program, and how my project will help determine which detector is most useful for a specific experiment.

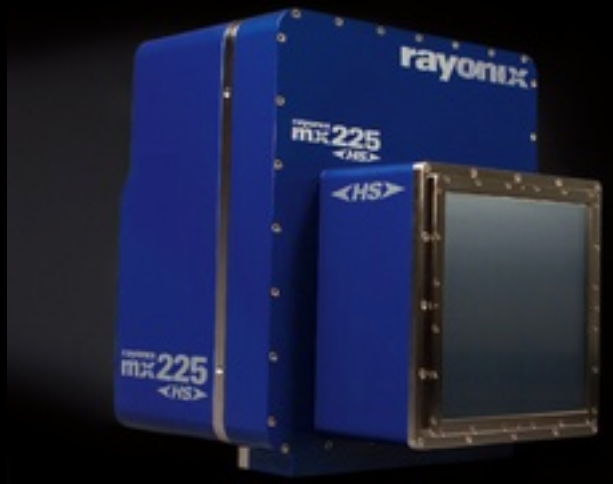
Adrienne Tiña – SULI Program

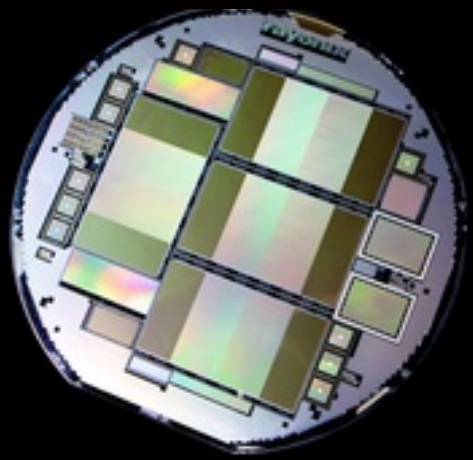
Background

- LCLS FEL
- Challenge
  - Need new detectors to be characterized
  - Computer Simulation

Detectors

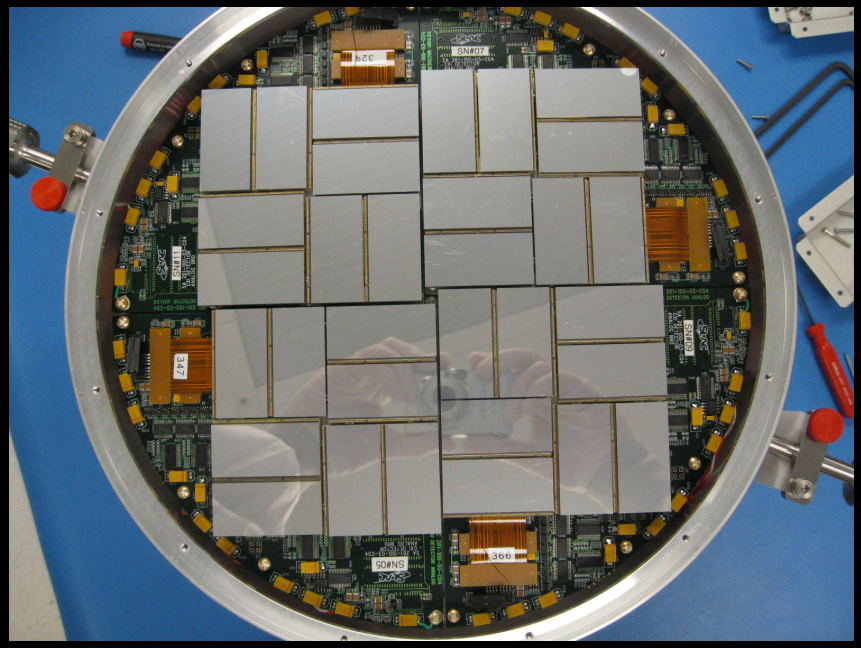
- Indirect
  - Scintillator, fiber-optic plate, CCD
  - Advantages
  - Rayonix MX170HS





Detectors

- Direct
  - CSPAD
  - ePix10k



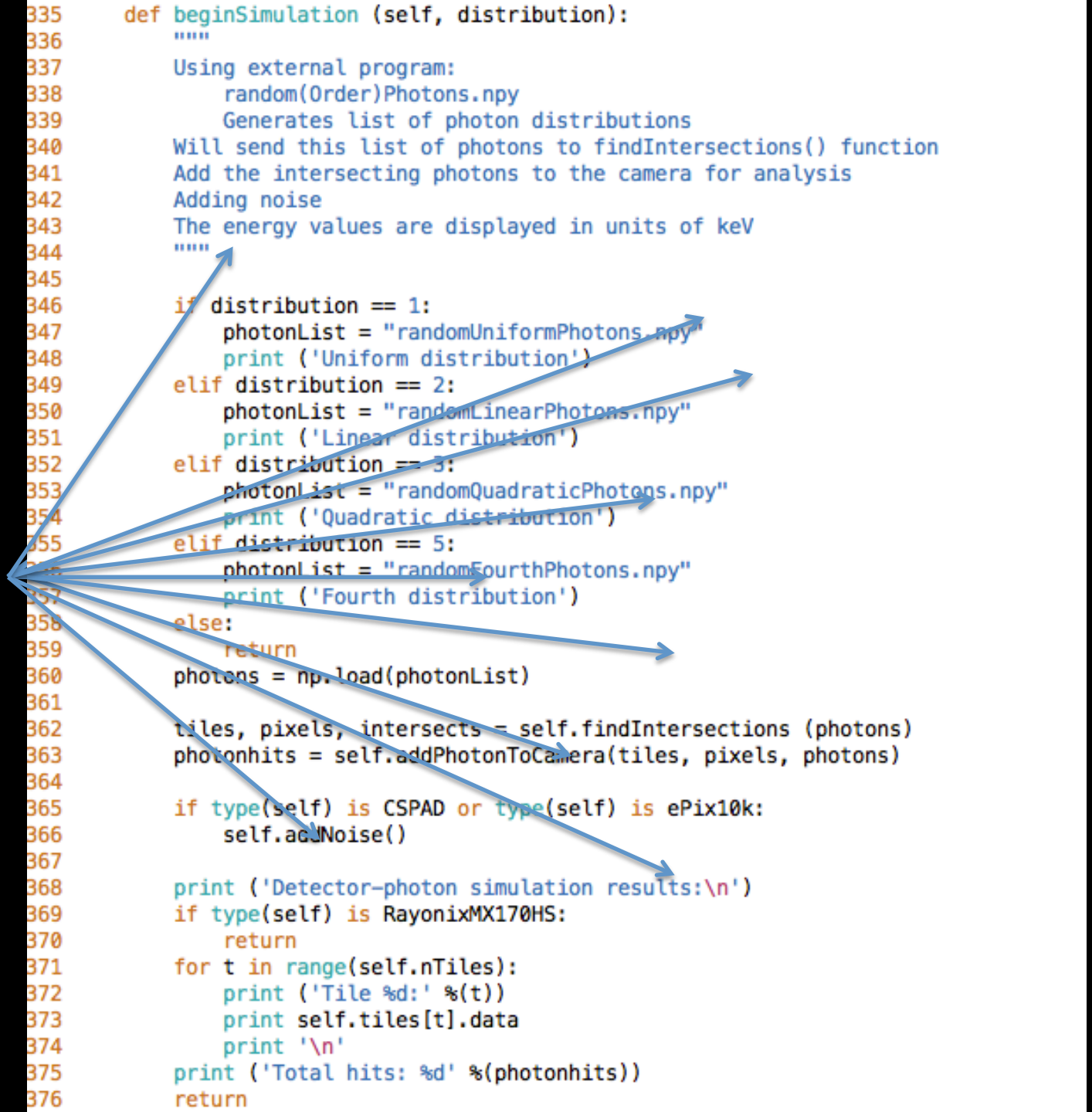
User interface

- Which specific detector
- X-ray photon intensity
- Order of photon distribution
- Noise values
- Etc.

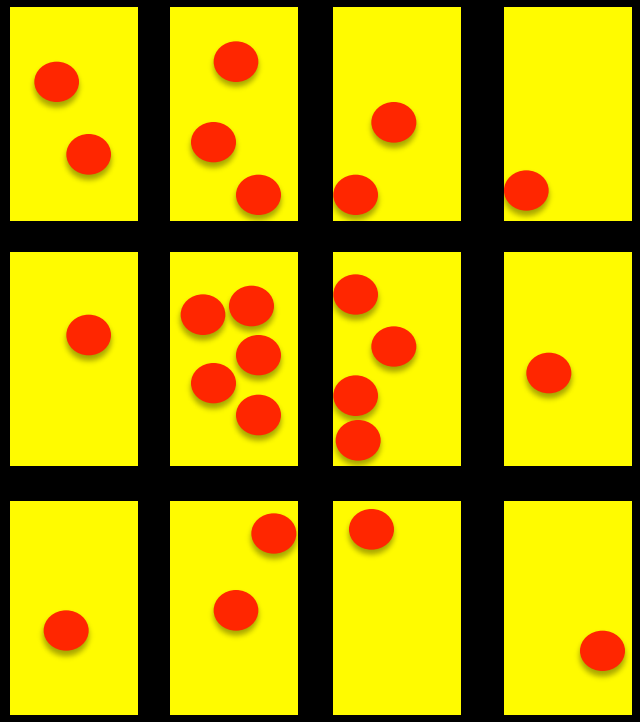
```
113 class Detector(object):
114
115     def __init__ (self, nTiles):
116
117         """
118         Parameters
119         ----------
120         nTiles : this variable will give us the size of the detector
121             and also possibly simplify searching
122
123
124         Attributes
125         ----------
126         tiles : this very important list is made of the Tile class
127                 it represents the detector grid made of several tiles
128
129         NoiseArray : this 2D array contains the same amount of pixels
130                 in each tile, w/ corresp. noise values
131
132         rows : this parameter specifies the amount of rows of pixels
133             per tile which will be stored in the Tile class
134
135         cols : this parameter specifies the amount of cols of pixels
136             per tile which will be stored in the Tile class
137
138         sizea : this specifies the width of the detector's pixels
139
140         sizeb : this specifies the length of the detector's pixels
141
142         rate : this is the Frame rate of the detector
143
144         _ps : array of position vectors (p-vectors) that represent the
145                 orientation of a tile in space
146
147         _ss : array of vectors that point along the slow-scan direction,
148             defining a single pixel in the direction of the vector
149
150         _fs : array of vectors that point along the fast-scan direction,
151             defining a single pixel in the direction of the vector
152
153         _shapes : defines grid dimensions, i.e. the number of pixels
154                 in the fast/slow direction
155
156         _ns : array of normal vectors that help define the plane of
157                 the corresponding tile at the same index
```

```
44 class Tile(object):
45
46     def __init__ (self, rows, cols):
47
48         """
49         Attributes
50         ----------
51         rows : the specified amount of rows of pixels in tile,
52                 specific to detector type
53
54         cols : the specified amount of columns of pixels in tile,
55                 specific to detector type
56
57         xLoc : the x-coordinate of the tile
58
59         yLoc : the y-coordinate of the tile
60
61         zLoc : the z-coordinate of the tile
62
63         theta : the first Euler angle of the tile
64
65         phi : the second Euler angle of the tile
66
67         The x,y,z coordinates and the two Euler angles will help determine
68             the plane that the tile covers
69         """
70
71         self.rows    = rows
72         self.cols    = cols
73         self.xLoc    = 0
74         self.yLoc    = 0
75         self.zLoc    = 0
76         self.theta   = 0
77         self.phi     = 0
78
79         ## pixel, data, gain, and gainSetting arrays represent pixel-specific info within tile
80         self.data           = np.zeros((self.cols, self.rows))
81         self.gain           = np.zeros((self.cols, self.rows))
82         self.gainSetting    = np.zeros((self.cols, self.rows))
83         self.NoiseArray     = np.zeros((self.cols, self.rows))
84         self.statusArray    = np.zeros((self.cols, self.rows))
```

```
267     def makeGrid (self, x, y, z, angle1, angle2, gain):
268         #initializing array of default tile values based on given initial values
269         index = 0
270         xmod = 0
271         ymod = 0
272         for i in range (self.nTiles):
273             # making 5x5 grid of tiles
274             if xmod == 5:
275                 xmod = 0
276                 x += 15
277             if ymod == 5:
278                 ymod = 0
279                 y = 0
280 
281             # Storing new values from user into Tile's variable placeholders
282             self.tiles[index].xLoc = x
283             self.tiles[index].yLoc = y
284             self.tiles[index].zLoc = z
285             self.tiles[index].theta = angle1
286             self.tiles[index].phi = angle2
287 
288 
289             p = (x, y, z)
290             self._ps.append(p)
291             print ('p:'), p
292             if index == 0:
293                 self._ns.append(p)
294 
295             y += 15
296             index += 1
297             xmod += 1
298             ymod += 1
299 
300         self.computeVectors()
```

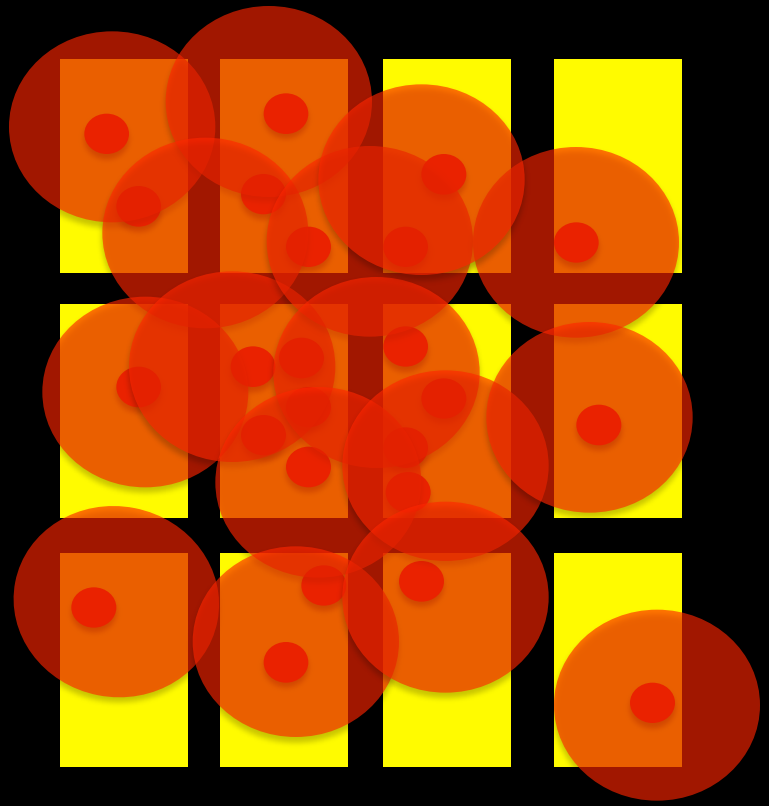


```
    def beginSimulation (self, distribution):
        """
        Using external program:
            random(Order)Photons.npy
            Generates list of photon distributions
        Will send this list of photons to findIntersections() function
        Add the intersecting photons to the camera for analysis
        Adding noise
        The energy values are displayed in units of keV
        """

        if distribution == 1:
            photonList = "randomUniformPhotons.npy"
            print ('Uniform distribution')
        elif distribution == 2:
            photonList = "randomLinearPhotons.npy"
            print ('Linear distribution')
        elif distribution == 3:
            photonList = "randomQuadraticPhotons.npy"
            print ('Quadratic distribution')
        elif distribution == 5:
            photonList = "randomFourthPhotons.npy"
            print ('Fourth distribution')
        else:
            return
        photons = np.load(photonList)

        tiles, pixels, intersects = self.findIntersections (photons)
        photonhits = self.addPhotonToCamera(tiles, pixels, photons)

        if type(self) is CSPAD or type(self) is ePix10k:
            self.addNoise()

        print ('Detector-photon simulation results:\n')
        if type(self) is RayonixMX170HS:
            return
        for t in range(self.nTiles):
            print ('Tile %d:' %(t))
            print self.tiles[t].data
            print '\n'
        print ('Total hits: %d' %(photonhits))
        return
```

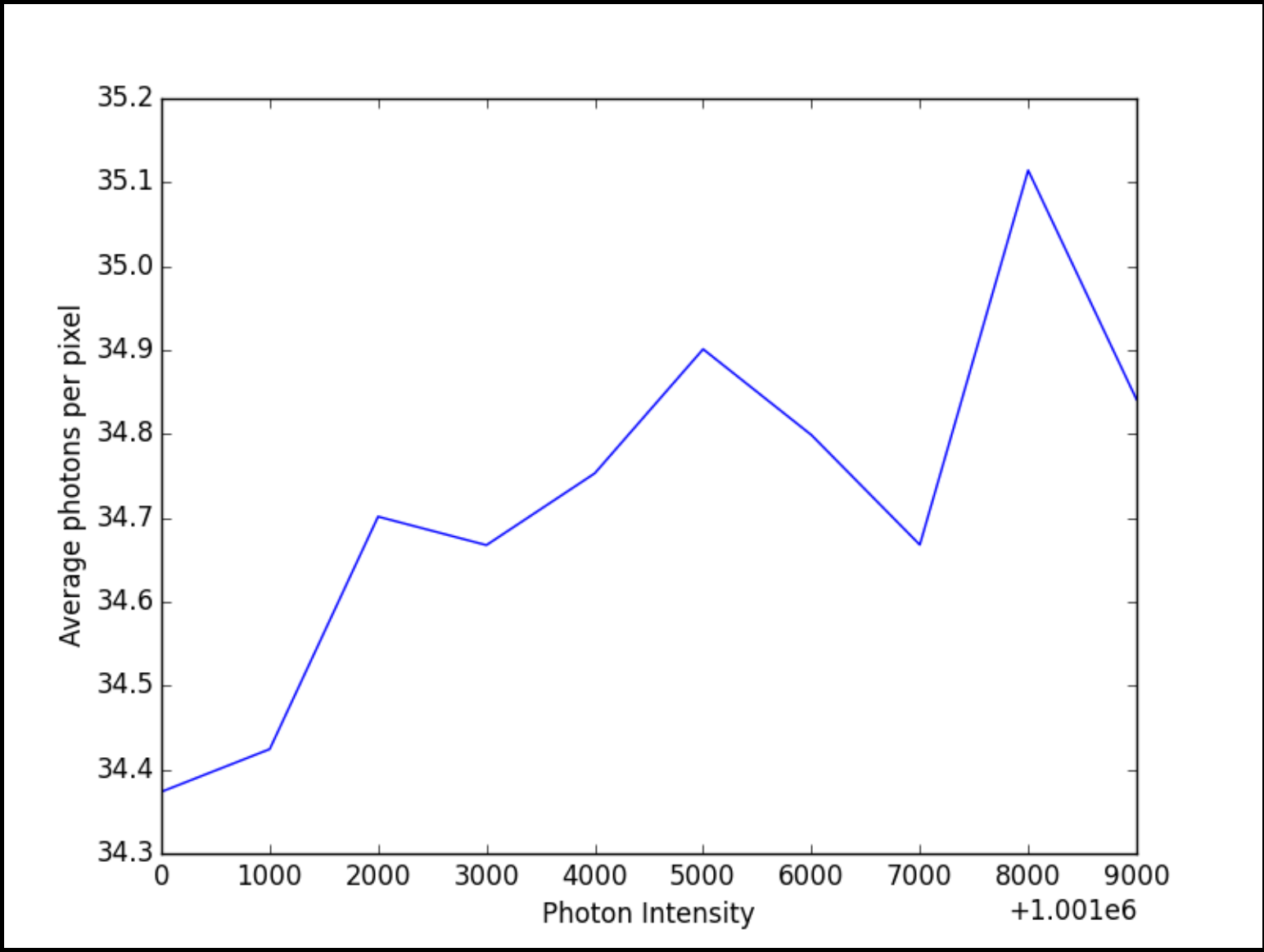


```
Tile 24:
[[ 10.   5.   2.   8.   5.   3.  10.   7.   4.   0.]
 [  8.   7.   6.   3.  11.   8.   9.   6.   4.   0.]
 [  4.   5.   7.   9.  10.   5.   7.   5.   7.   0.]
 [  9.   5.  10.  11.   7.   5.   5.   3.   5.   0.]
 [  5.   3.   1.   8.   4.   5.   9.   5.   8.   0.]
 [  5.   5.   5.   4.   8.   9.   9.   4.   8.   0.]
 [  7.   7.   6.   8.   6.   4.   7.   5.  10.   0.]
 [  5.   8.   7.   5.   5.   7.   6.   8.   3.   0.]
 [  4.   3.   3.   4.  12.   7.   6.  12.   7.   0.]
 [  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.]]
```

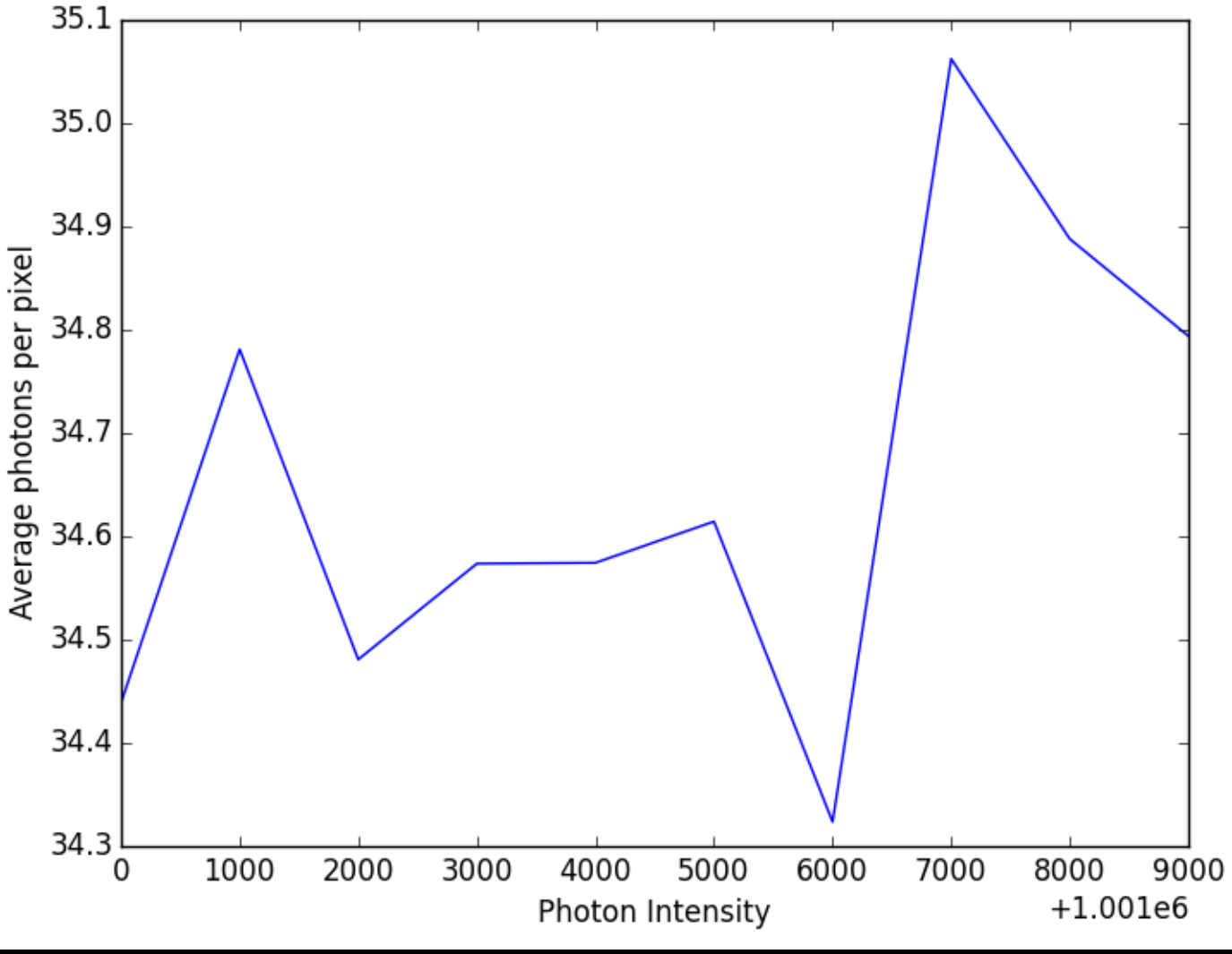
```
Tile 0:
[[ 726.  717.  707.  712.  639.  651.  571.  542.  497.    0.]
 [ 753.  727.  708.  717.  671.  603.  521.  545.  517.    0.]
 [ 695.  651.  619.  659.  630.  625.  559.  512.  495.    0.]
 [ 646.  653.  645.  609.  538.  511.  531.  487.  506.    0.]
 [ 594.  557.  546.  558.  494.  465.  499.  447.  426.    0.]
 [ 489.  549.  467.  465.  446.  448.  436.  412.  382.    0.]
 [ 450.  420.  413.  438.  392.  458.  389.  367.  356.    0.]
 [ 366.  376.  370.  382.  362.  341.  317.  343.  310.    0.]
 [ 324.  353.  345.  357.  328.  340.  296.  276.  267.    0.]
 [   0.    0.    0.    0.    0.    0.    0.    0.    0.    0.]]
```



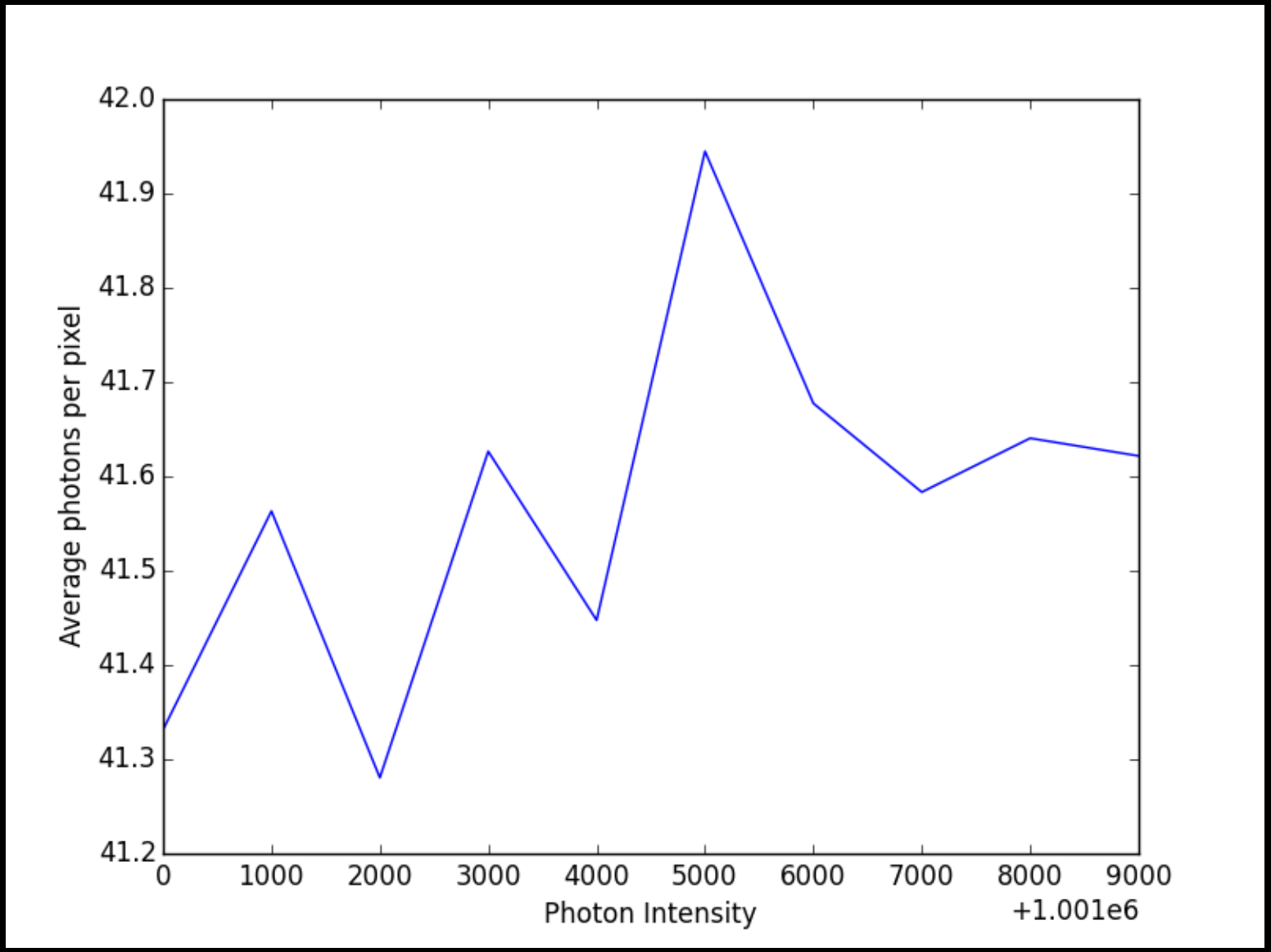
CSPAD


35.2
35.1
35.0
34.9
34.8
34.7
34.6
34.5
34.4
34.3
Average photons per pixel
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
Photon Intensity
+1.001e6

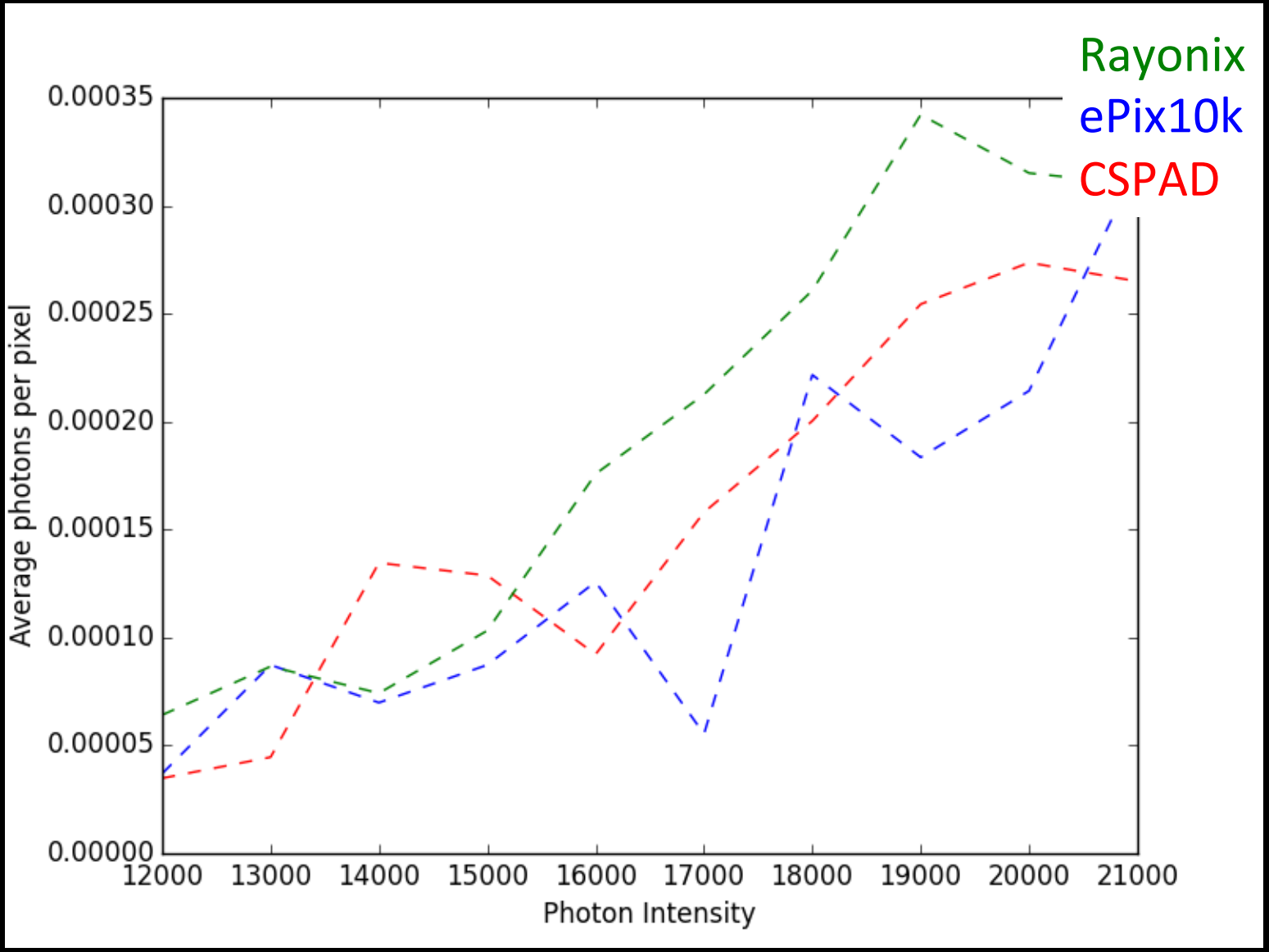
ePix10k


35.1
35.0
34.9
34.8
34.7
34.6
34.5
34.4
34.3
Average photons per pixel
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
Photon Intensity
+1.001e6

Rayonix


42.0
41.9
41.8
41.7
41.6
41.5
41.4
41.3
41.2
Average photons per pixel
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
Photon Intensity
+1.001e6

3 Cameras


Rayonix
ePix10k
CSPAD
0.00035
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
Average photons per pixel
12000
13000
14000
15000
16000
17000
18000
19000
20000
21000
Photon Intensity

Conclusion

- Characterize detectors by simulating behavior
- Understand how they work, what they output, costs and benefits
  - Active area
  - SNR tradeoff
- **Which detector is most useful for specific experiment**
- Program applicable in future

Acknowledgements

- Gabriella Carini
- Philip Hart
- Thomas Joseph Lane
- Enrique Cuellar
- SULI Program

U.S. DEPARTMENT OF ENERGY | Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY