Lawrence Berkeley National Laboratory

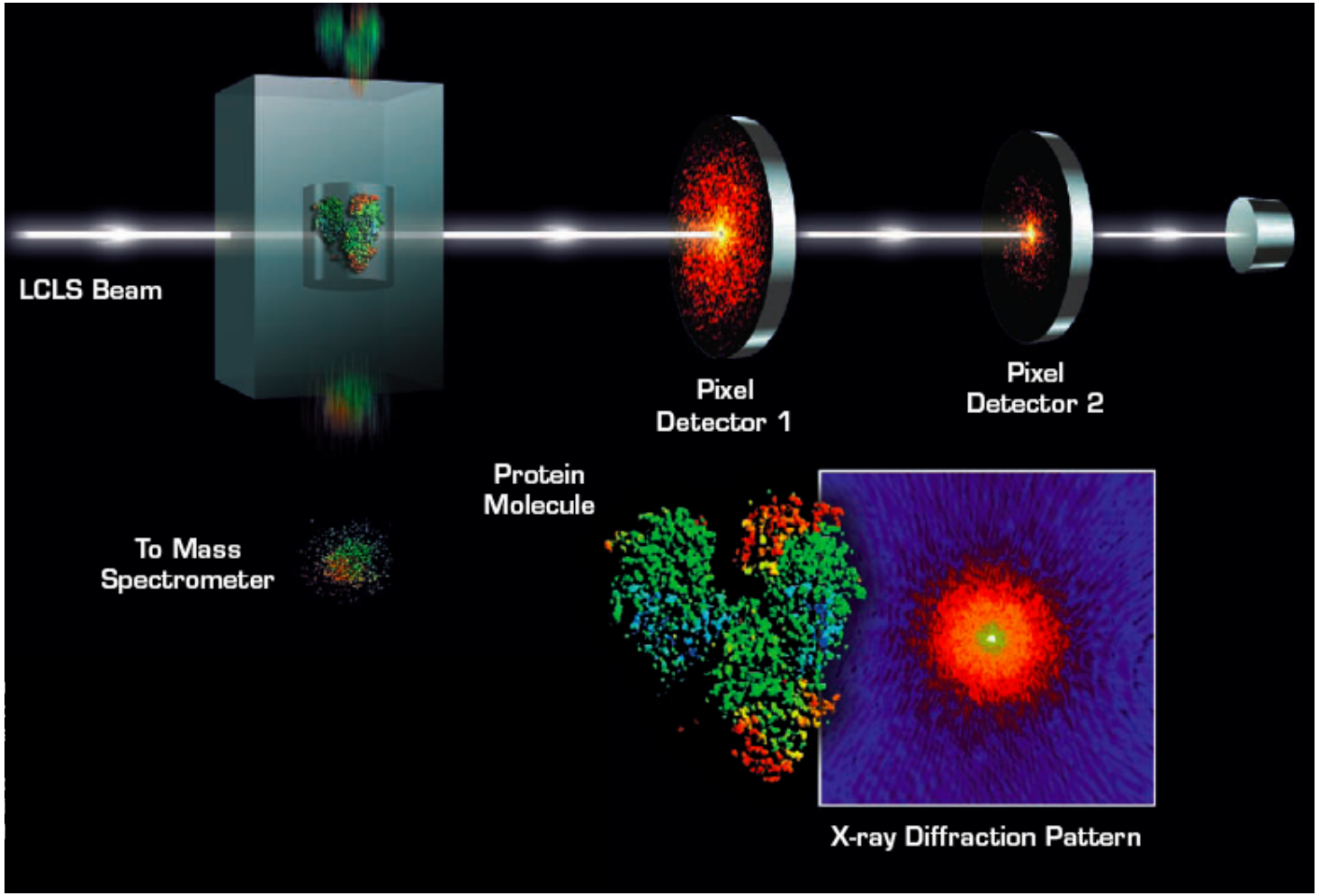
CUDA accelerated X-ray Imaging

Filipe Maia
NERSC

Stefano Marchesini
ALS

2011-01-26

Single Particle X-ray Diffraction



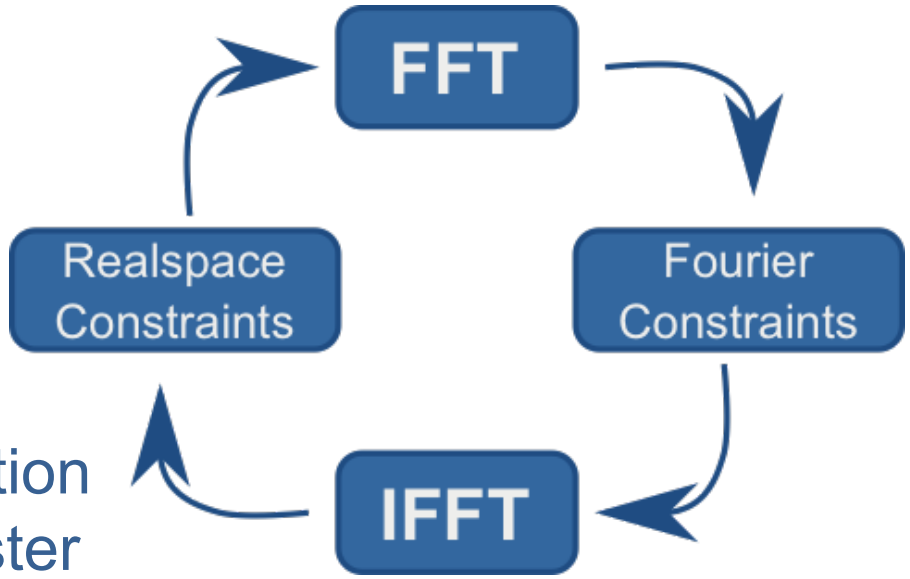
BERKELEY LAB

Single Particle X-ray Diffraction

- Recover an object from its diffraction pattern.
- Iterative procedure
- Requires user interaction to find best parameters

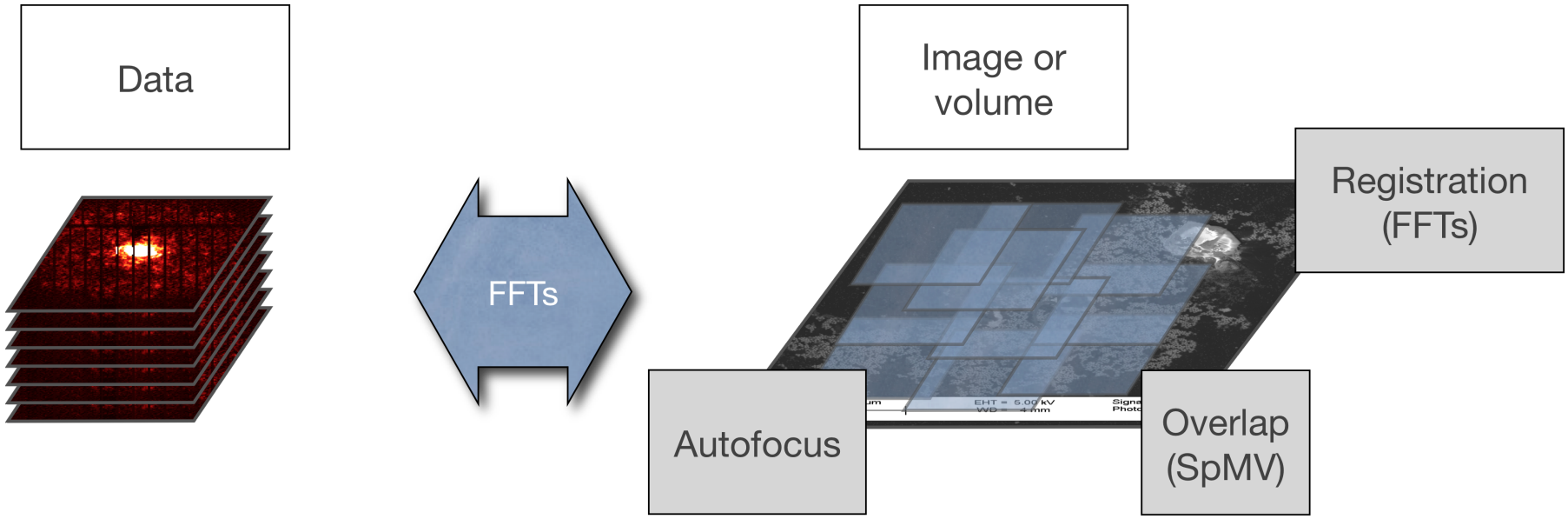
$$I(x) = |\mathcal{F}\rho(r)|^2$$

$$I(x) \rightarrow \rho(r)$$



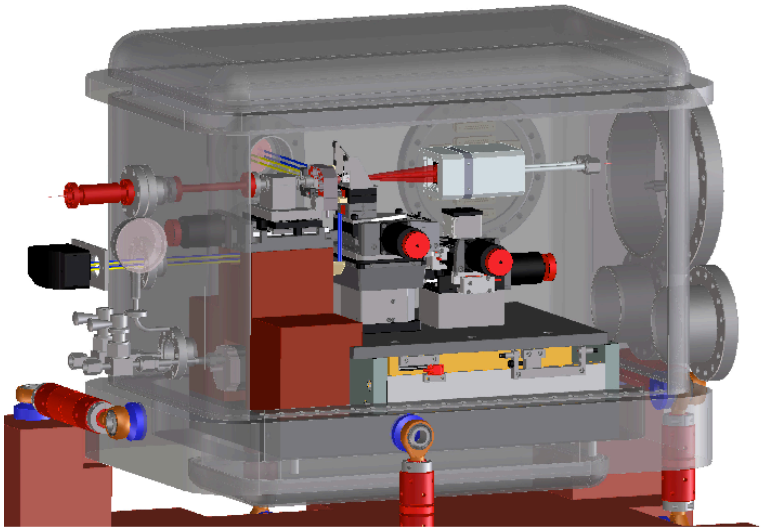
- CUDA made reconstruction an order of magnitude faster

Ptychography – Realtime X-ray "microscope"

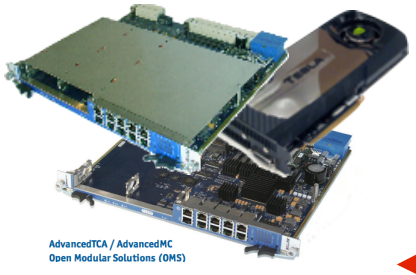


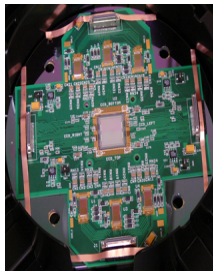
- Fast auto-focus possible with 1 GPU+fast motors.
- Algorithms tolerate 100 nm vibrations/accuracy ($200k saving)
- 50 MB/s enable real time high resolution imaging

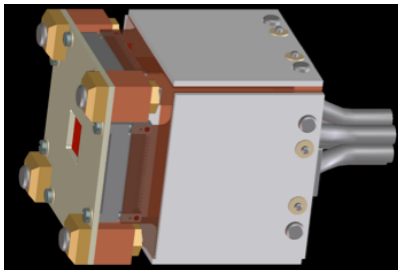
Nanosurveyor - Ptychography



Implement dedicated infrastructure at ALS

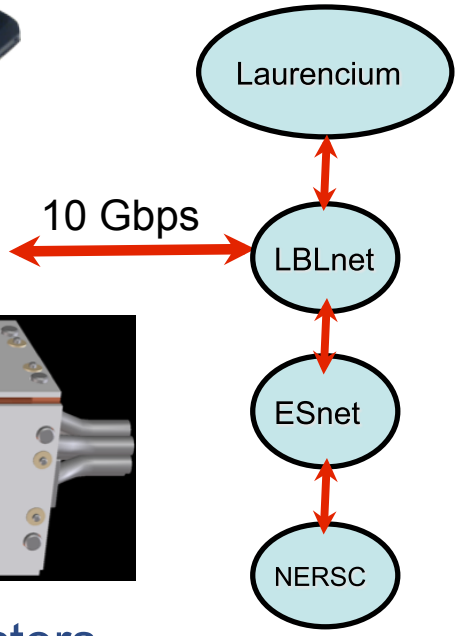






ARRA Funded detectors

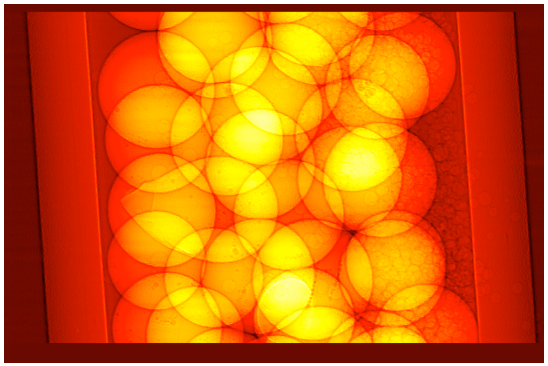
HPC infrastructure at LBL



BERKELEY LAB

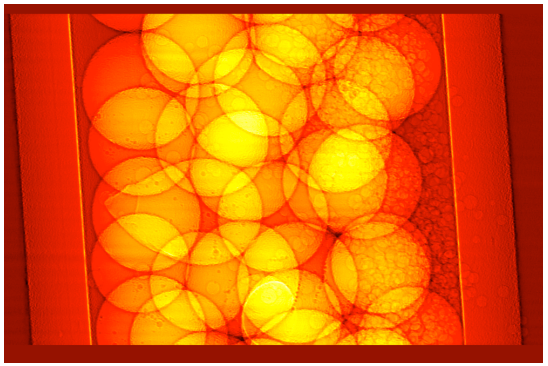
Phase Contrast X-ray Tomography

BL 8.3.2



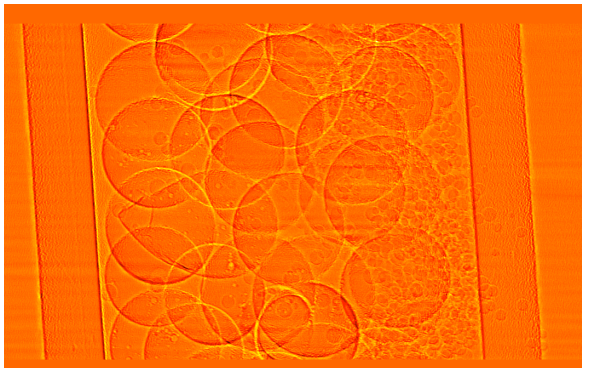
short distance

-



long distance

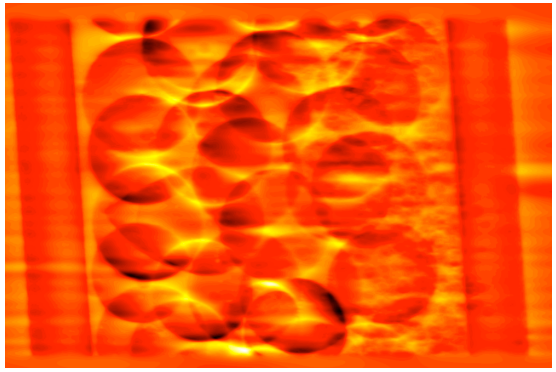
=



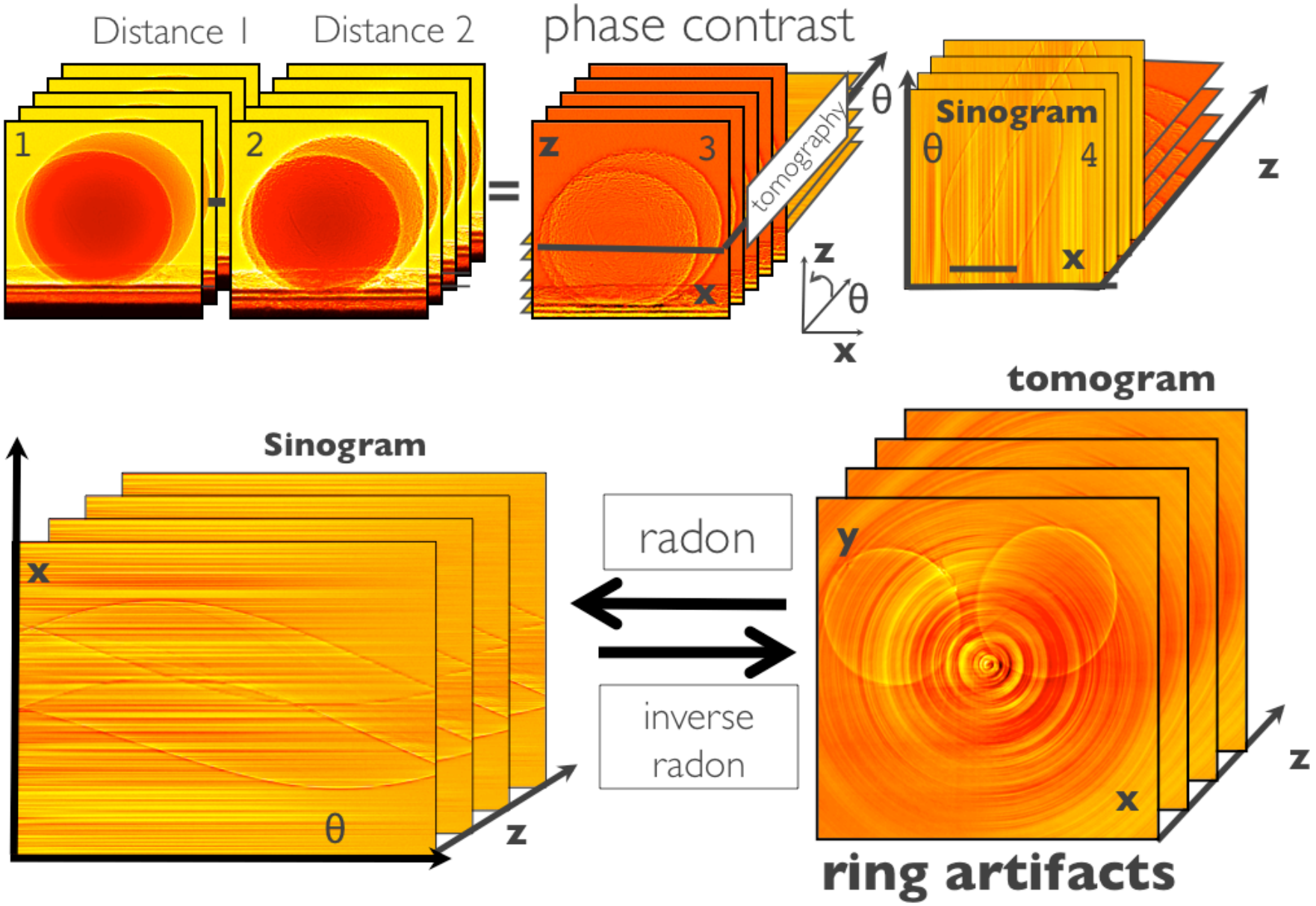
phase contrast

$$\frac{d}{dz}I(x,y) \approx -k^{-1}\nabla_{\perp}\bullet\left(I(x,y)\nabla_{\perp}\phi(x,y)\right)$$

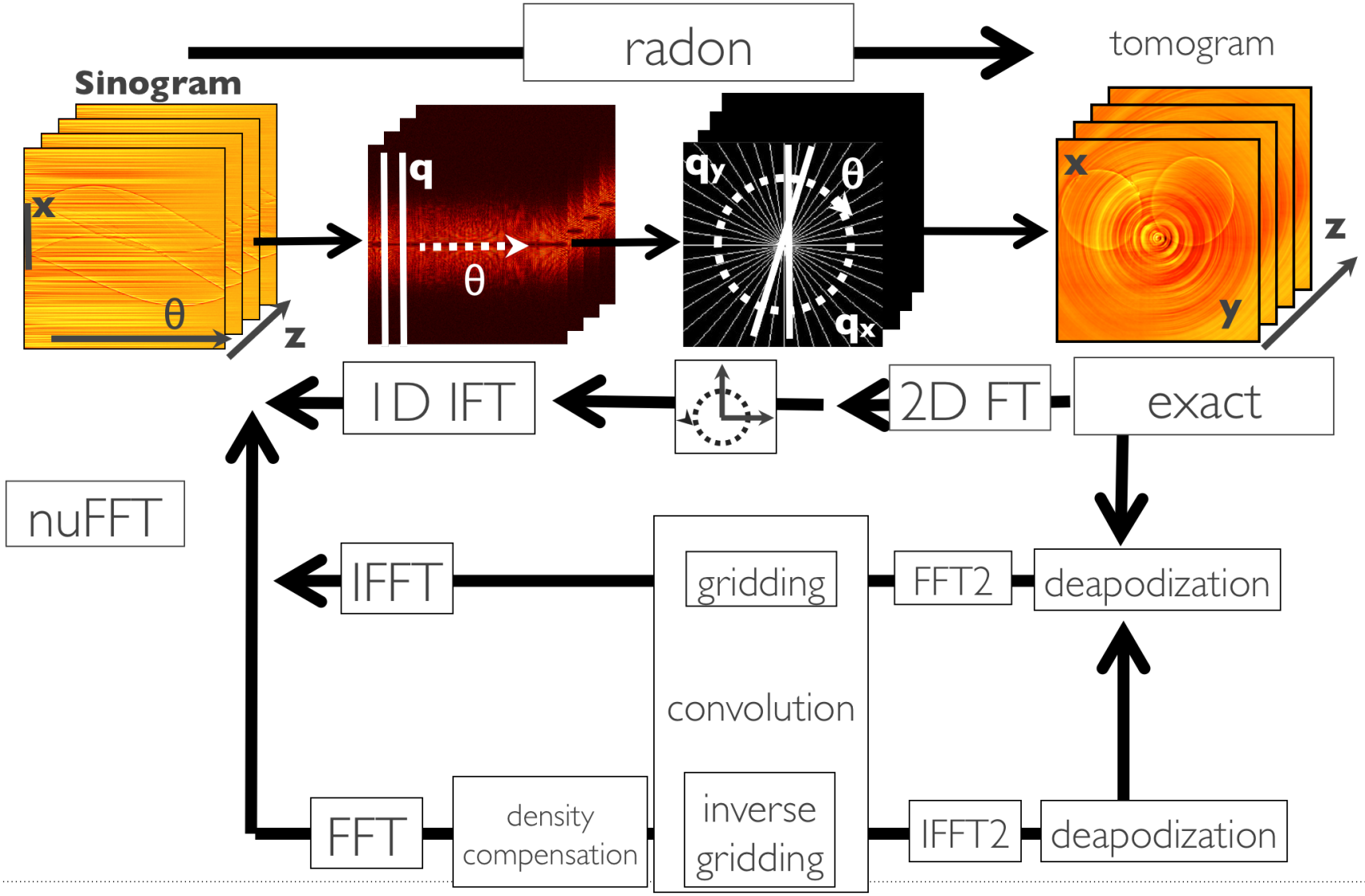
Transport of intensity equation



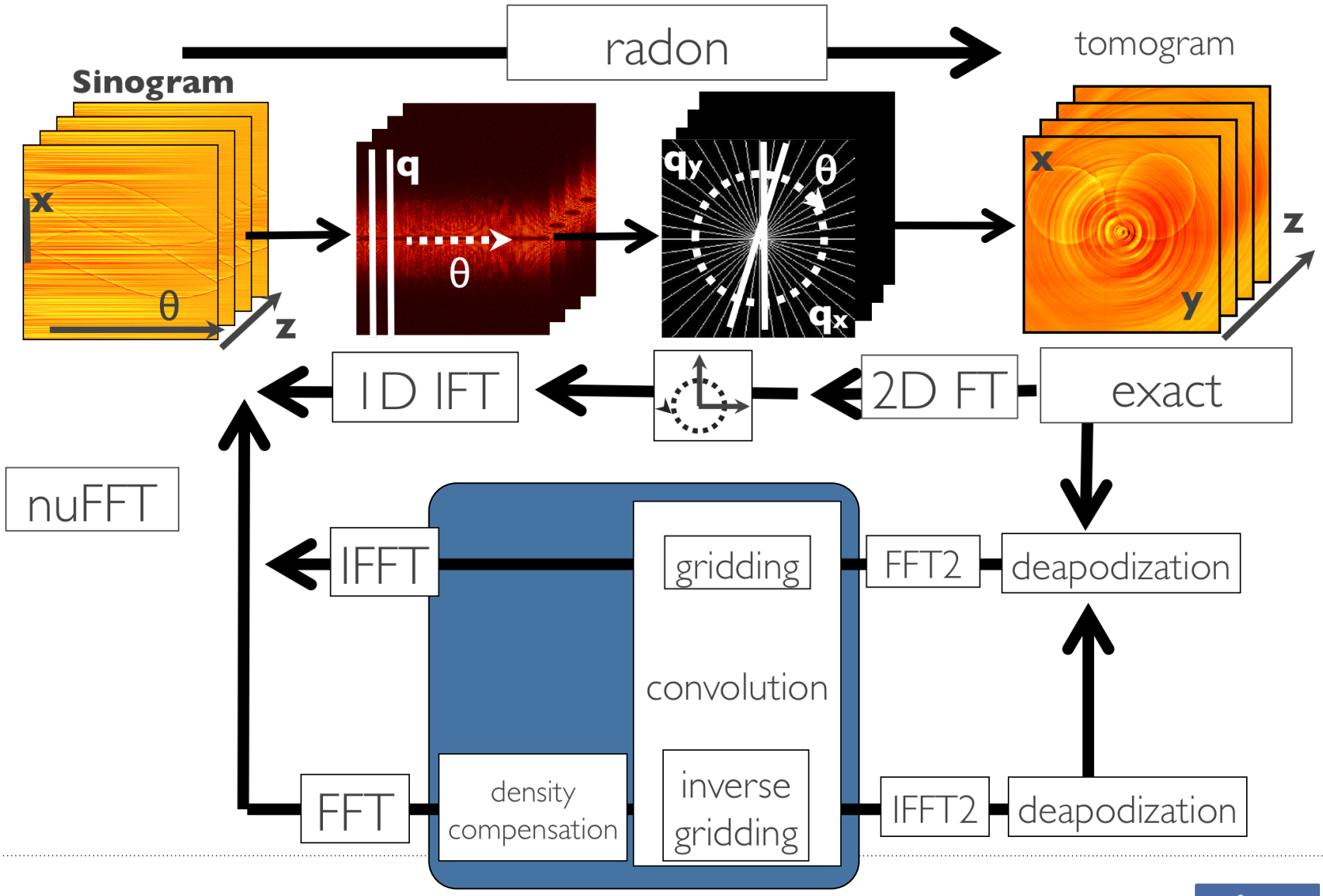
reconstructed phase


Distance 1
Distance 2
phase contrast
1
2
z
3
tomography
θ
x
z
θ
x
Sinogram
θ
4
x
z
tomogram
Sinogram
x
θ
z
radon
inverse
radon
y
x
z
ring artifacts

Need fast Radon transform

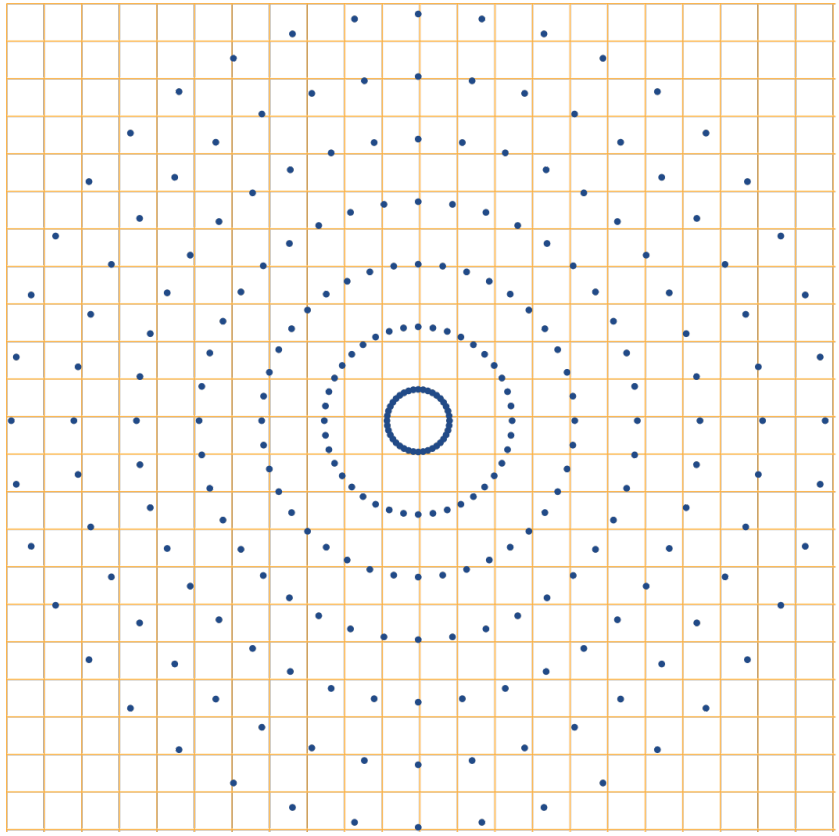


Need fast Radon transform



Inverse Gridding

- Convert from Cartesian to polar samples

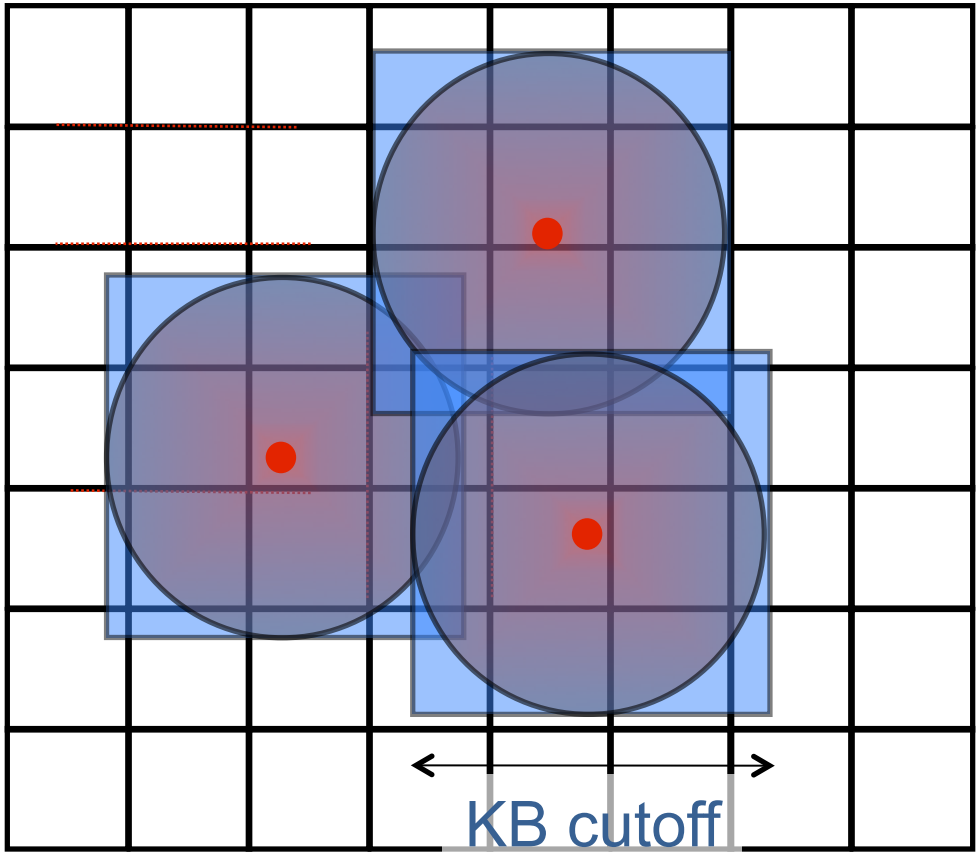


Input Samples

Output Samples

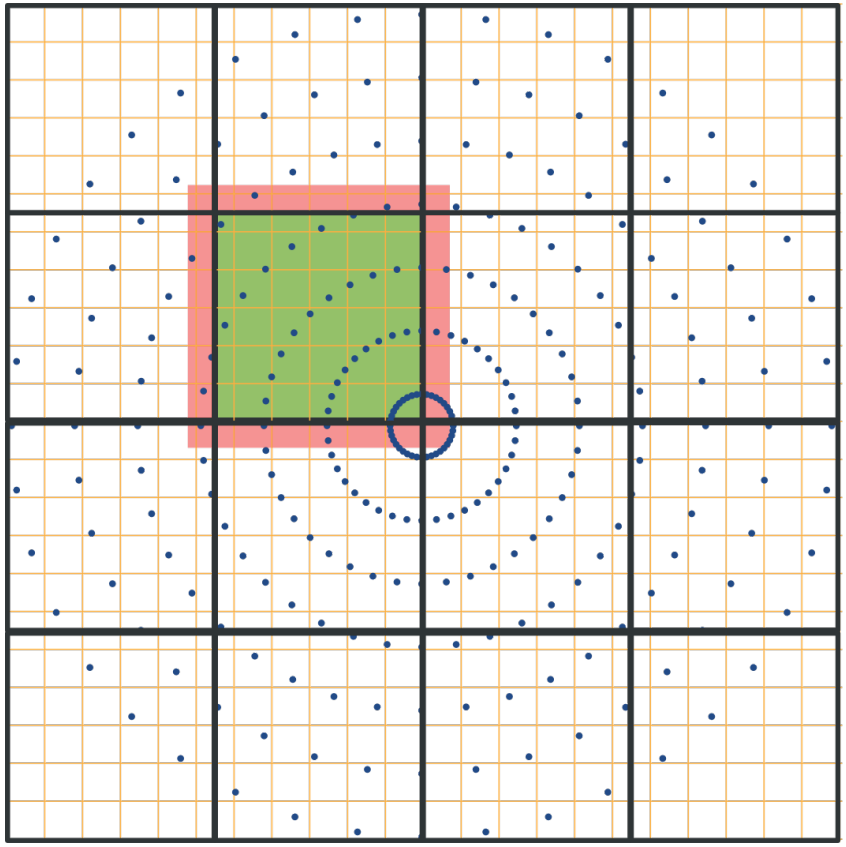
Inverse Gridding

- We will use a Kaiser Bessel function for the sampling.



Inverse Gridding – GPU Strategy

- Divide the input in equal area regions.



Halo Around Each Block

Area Assigned To One Block

Input Samples

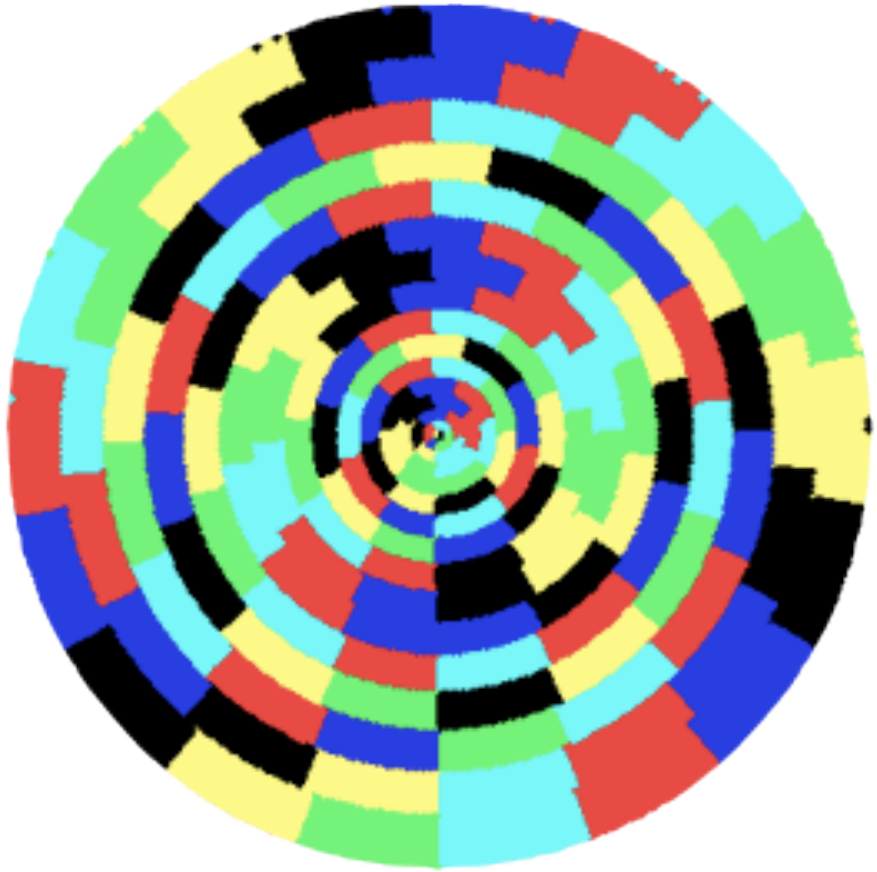
Output Samples

Inverse Gridding – GPU Strategy

```
for each grid point{
     grid point cache = grid point
}

for each sample point in region{
     for each grid point cache within KB radius of sample point{
          distance = sample point position –
                     grid point cache position
          sample += grid point cache * KB weight(distance)
     }
}
```

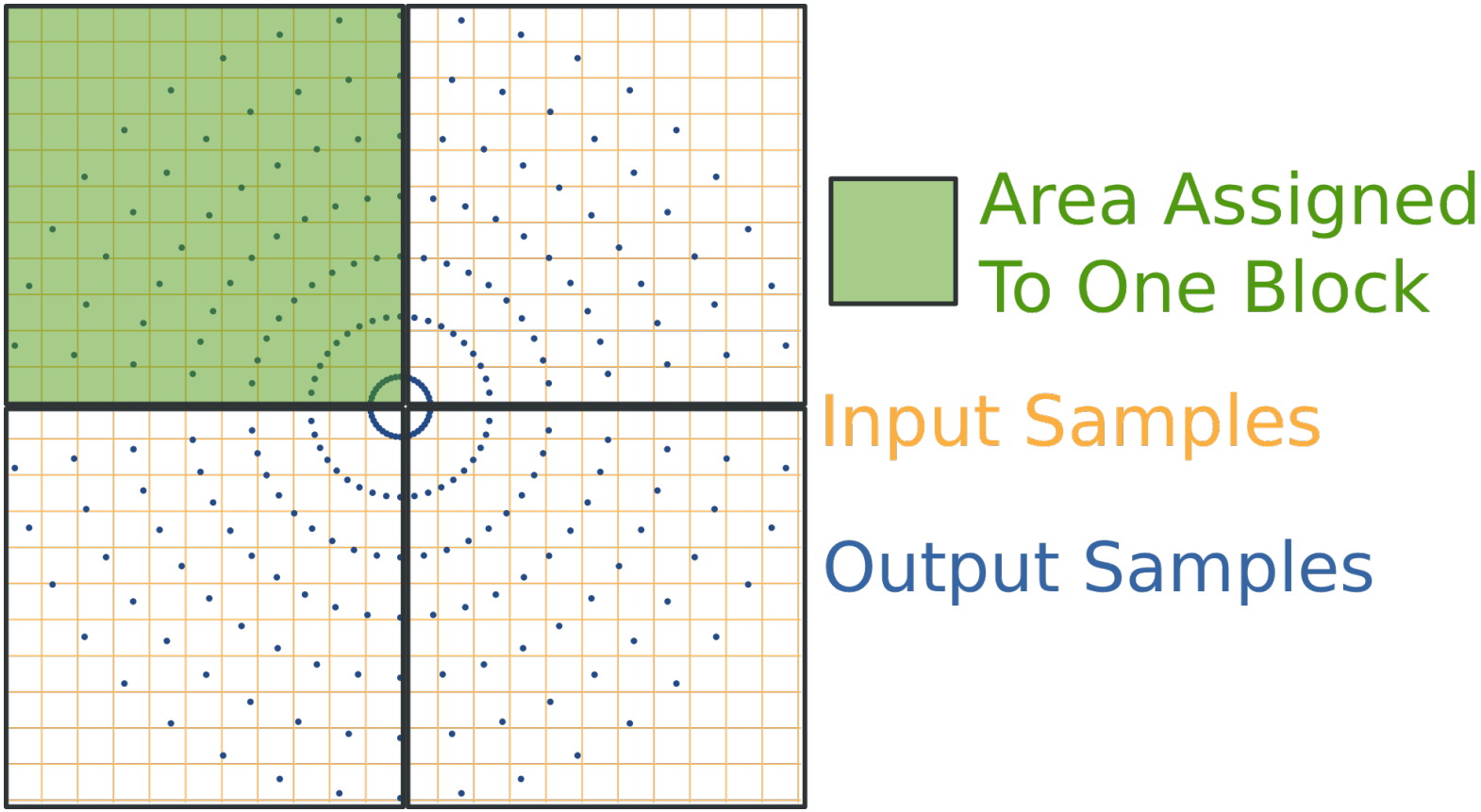
Inverse Gridding – Load Imbalance

- More balanced regions!



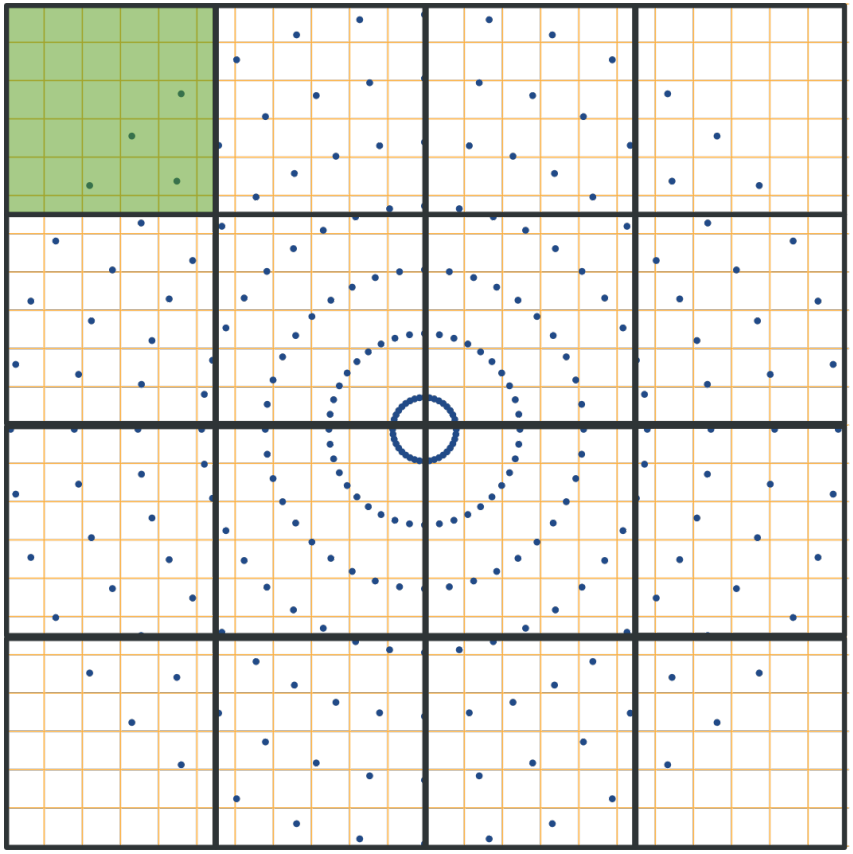
Inverse Gridding – GPU Strategy 2.0

- Recursively bisect the regions of highest workload (quadtree).



Inverse Gridding – GPU Strategy 2.0

- Recursively bisect the regions of highest workload (quadtree).



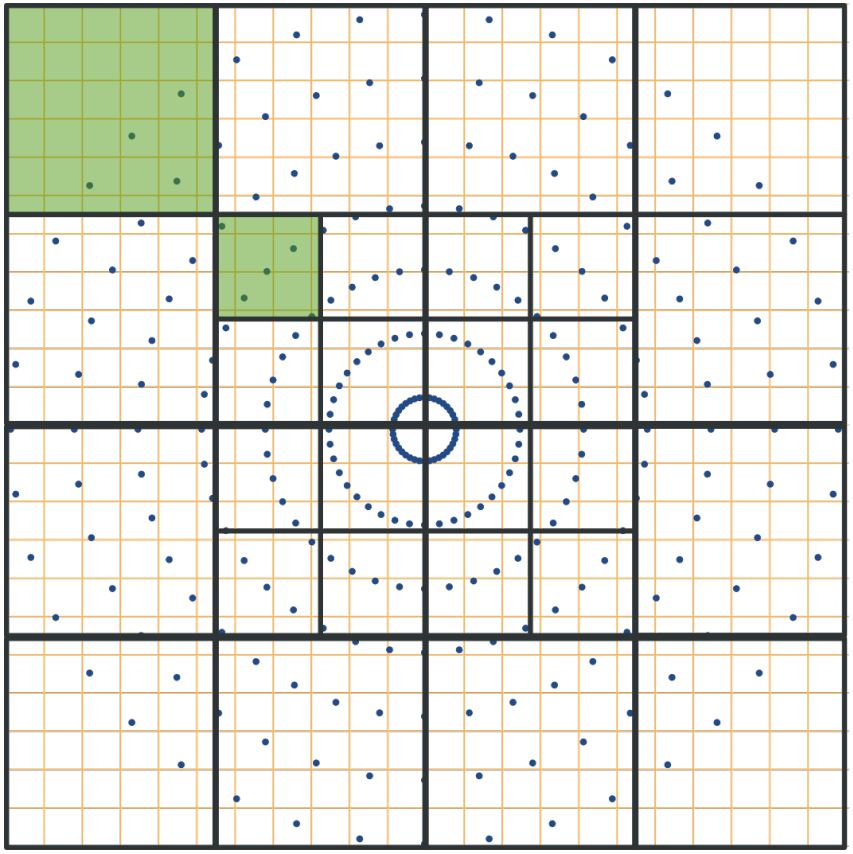
Area Assigned To One Block

Input Samples

Output Samples

Inverse Gridding – GPU Strategy 2.0

- Recursively bisect the regions of highest workload (quadtree).



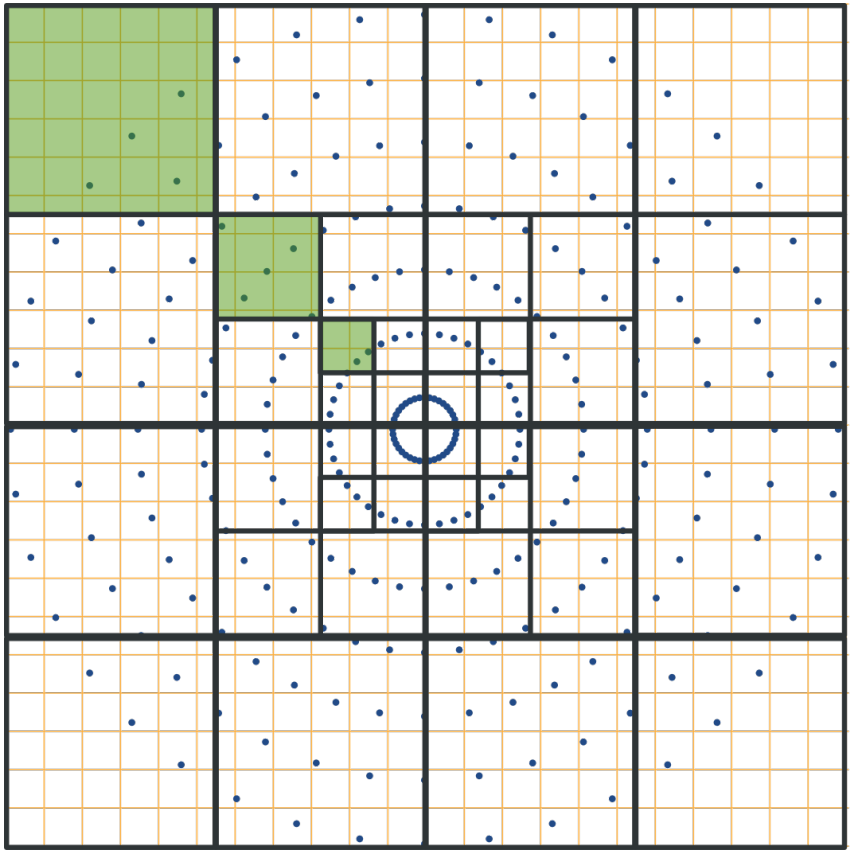
Area Assigned To One Block

Input Samples

Output Samples

Inverse Gridding – GPU Strategy 2.0

- Recursively bisect the regions of highest workload (quadtree).



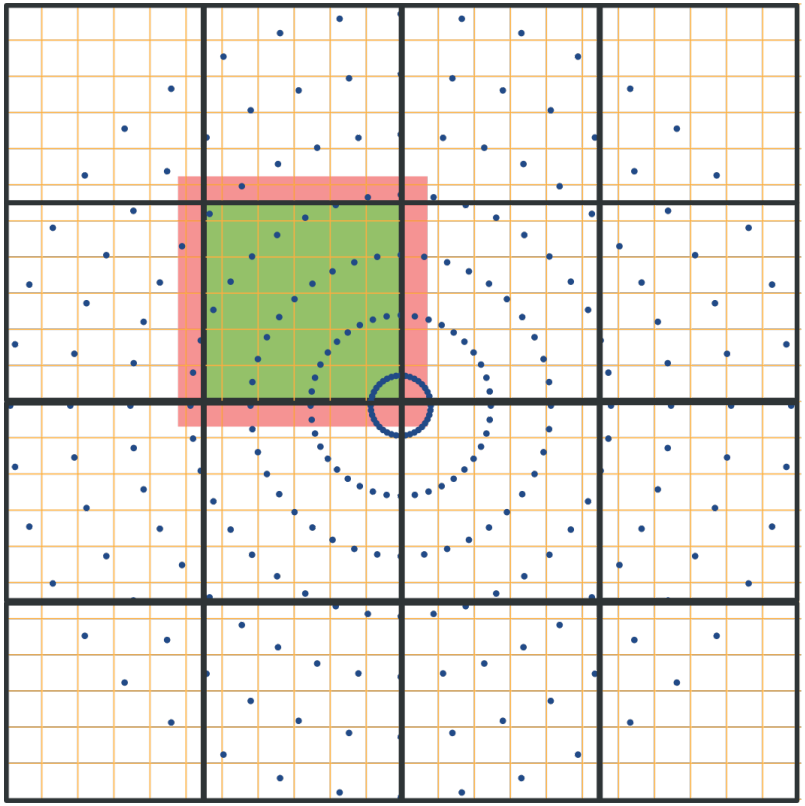
Area Assigned To One Block

Input Samples

Output Samples

Gridding

- More complicated
- We can't predict the location of the input samples



Halo Around Each Block

Area Assigned To One Block

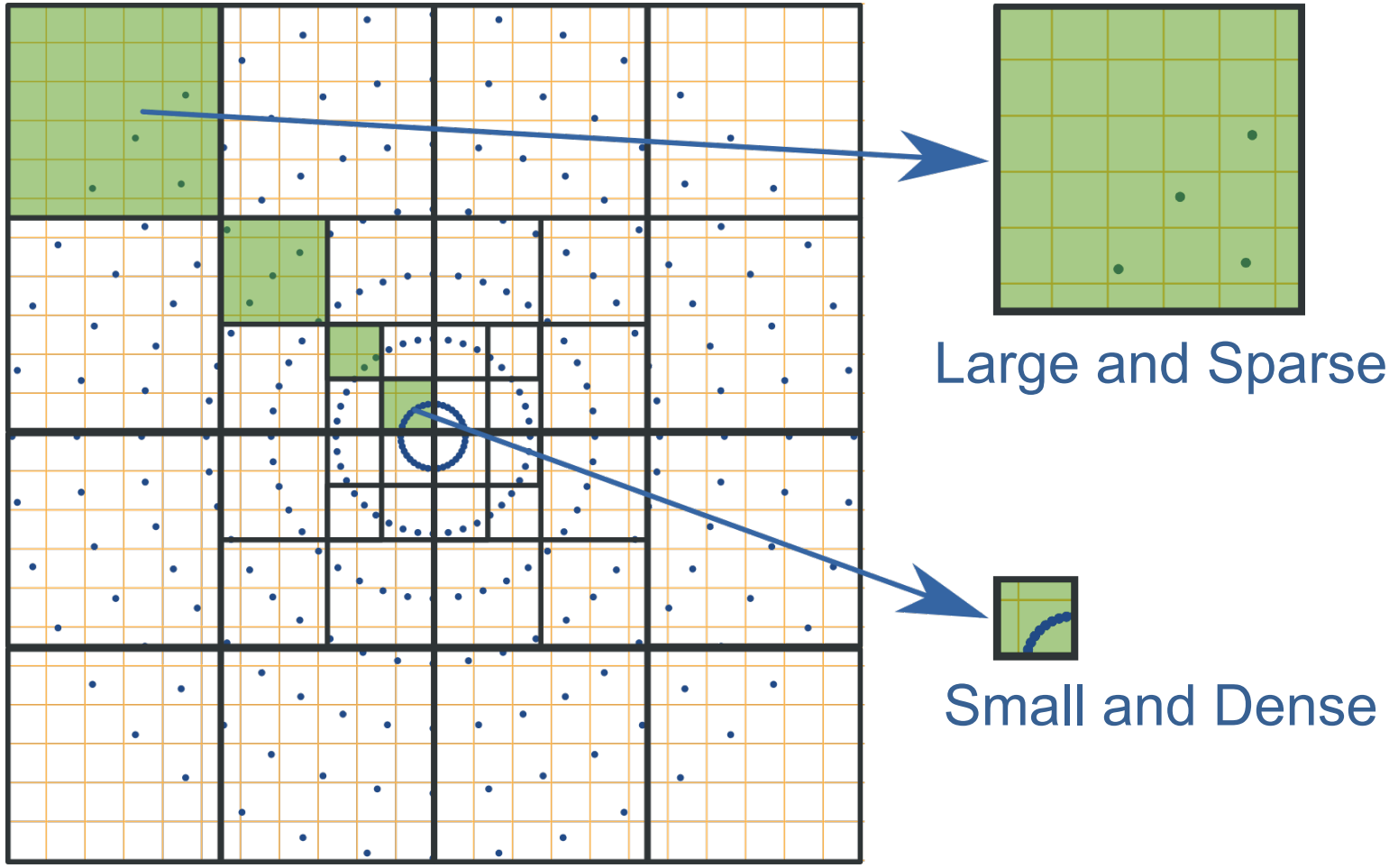
Input Samples

Output Samples

Gridding

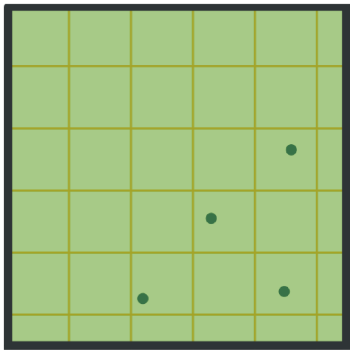
```
for each polar point within Halo of region{
     polar point cache = polar point
     polar position cache = polar position
}

for each grid point in region{
     for each polar point cache{
          distance = grid point position - polar position cache
          if(distance <= KB radius){
               grid sample += polar point cache * KB weight(distance)
          }
     }
}
```

Gridding – Two area types



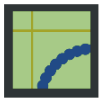
Gridding – Two area types

- Lots of grid points and not much work per grid point
- **Strategy 1** – One thread per grid point



Large and Sparse

- Very few grid points to calculate and lots of work per grid point
- **Strategy 2** – All the threads calculate each grid point (requires a reduction)



Small and Dense

```
for each polar point within Halo of region{
      polar point cache = polar point
      polar position cache = polar position
}
if(sparse region){
      for each grid point in region in blockSize steps{
            for each polar point cache{
                  distance = grid point position - polar position cache
                  if(distance <= KB radius){
                        grid sample += polar point cache * KB weight(distance)
                  }
            }
      }
}else{
      for each grid point in region{
            for each polar point cache in blockSize steps{
                  distance = grid point position - polar position cache
                  if(distance <= KB radius){
                        grid sample += polar point cache * KB weight(distance)
                  }
            }
            if(threadId == 0){
                  grid sample = block reduce(grid sample)
            }
      }
}
```

Around 50x speedup compared to 1 core

Conclusions

- GPUs can provide large computing power for interactive tasks.
- They are often more power efficient.
- Require careful programming to take full advantage of.
- The performance improvement provided can make new algorithms feasible.

Acknowledgements

- American Recovery Act

- SciDAC DOE

BERKELEY LAB

Thank You

U.S. DEPARTMENT OF
ENERGY

UNIVERSITY OF
CALIFORNIA