Cluster Computing: SMP Assignment

Martin Rehr

eScience Center
Niels Bohr Institute
University of Copenhagen

Copenhagen, February the 25th, 2013

CT-Reconstruction

Using Symmetric Multi Processors (SMP)

Outline

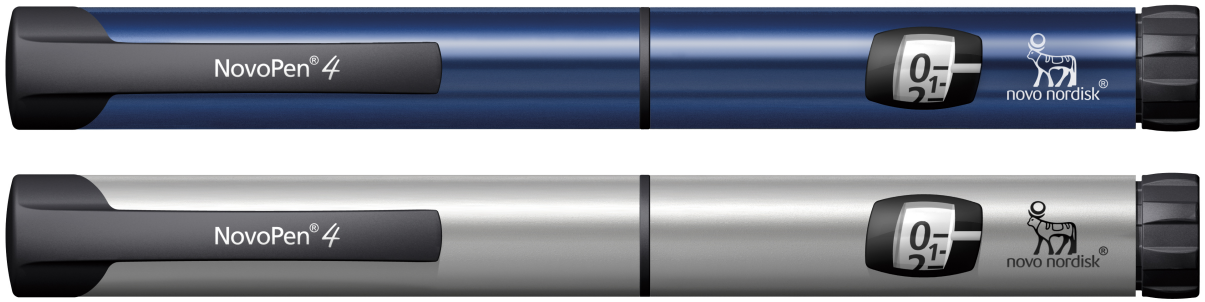
CT Scanning

We go to the hospital, right ?!?

CT Scanning: Veterinary

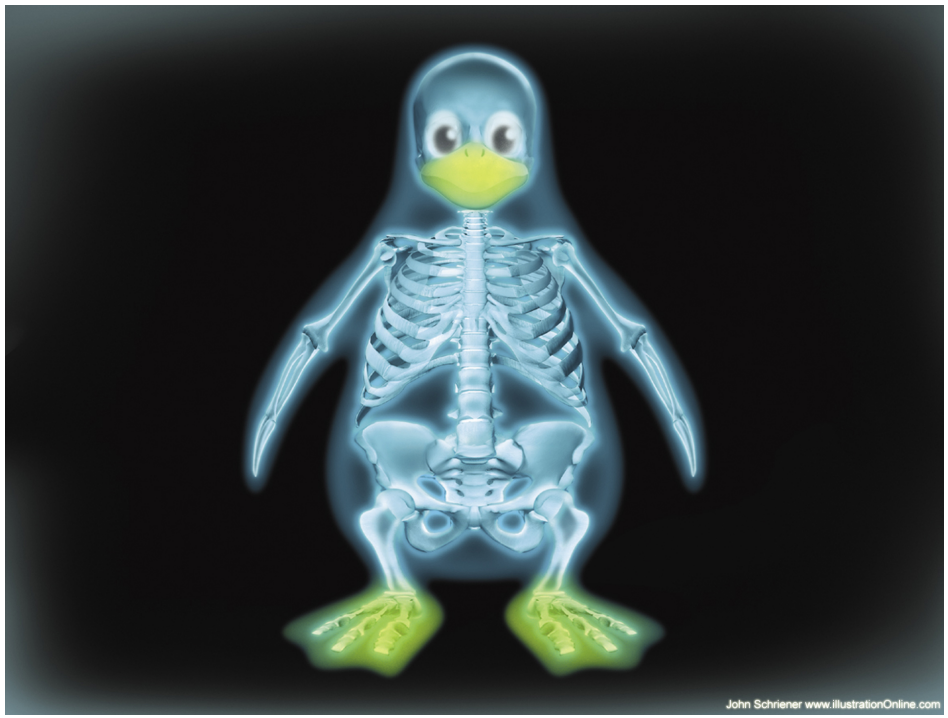
CT Scanning: Danish Crown

CT Scanning: NOVO



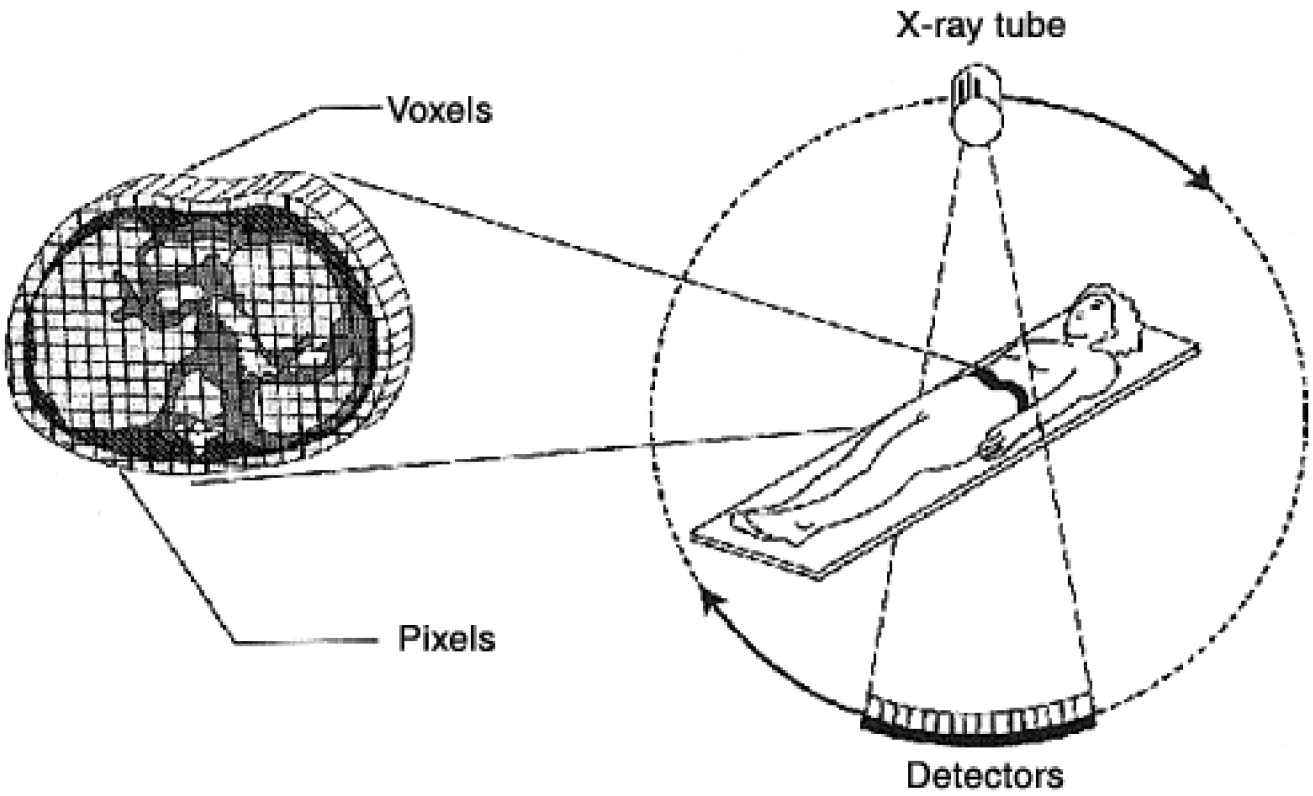
CT Scanning: Grundfos

CT Scanning: NBI XRay

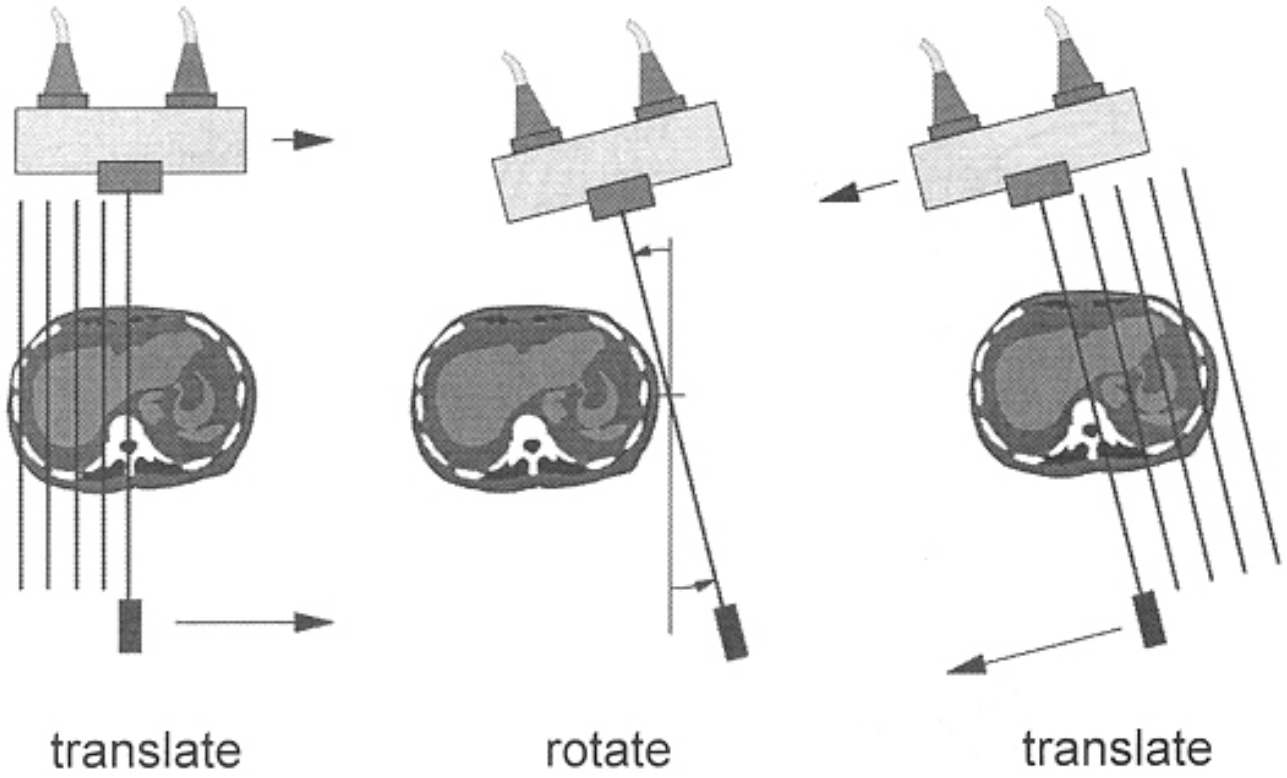


Computed Tomography

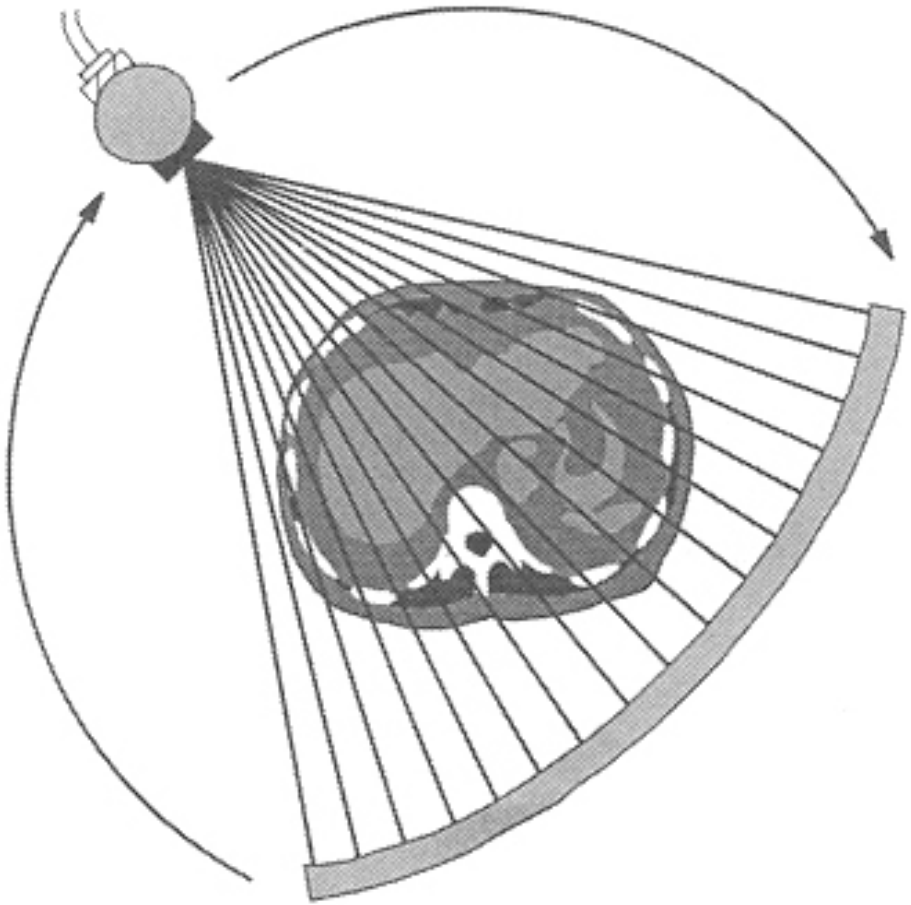
Overview



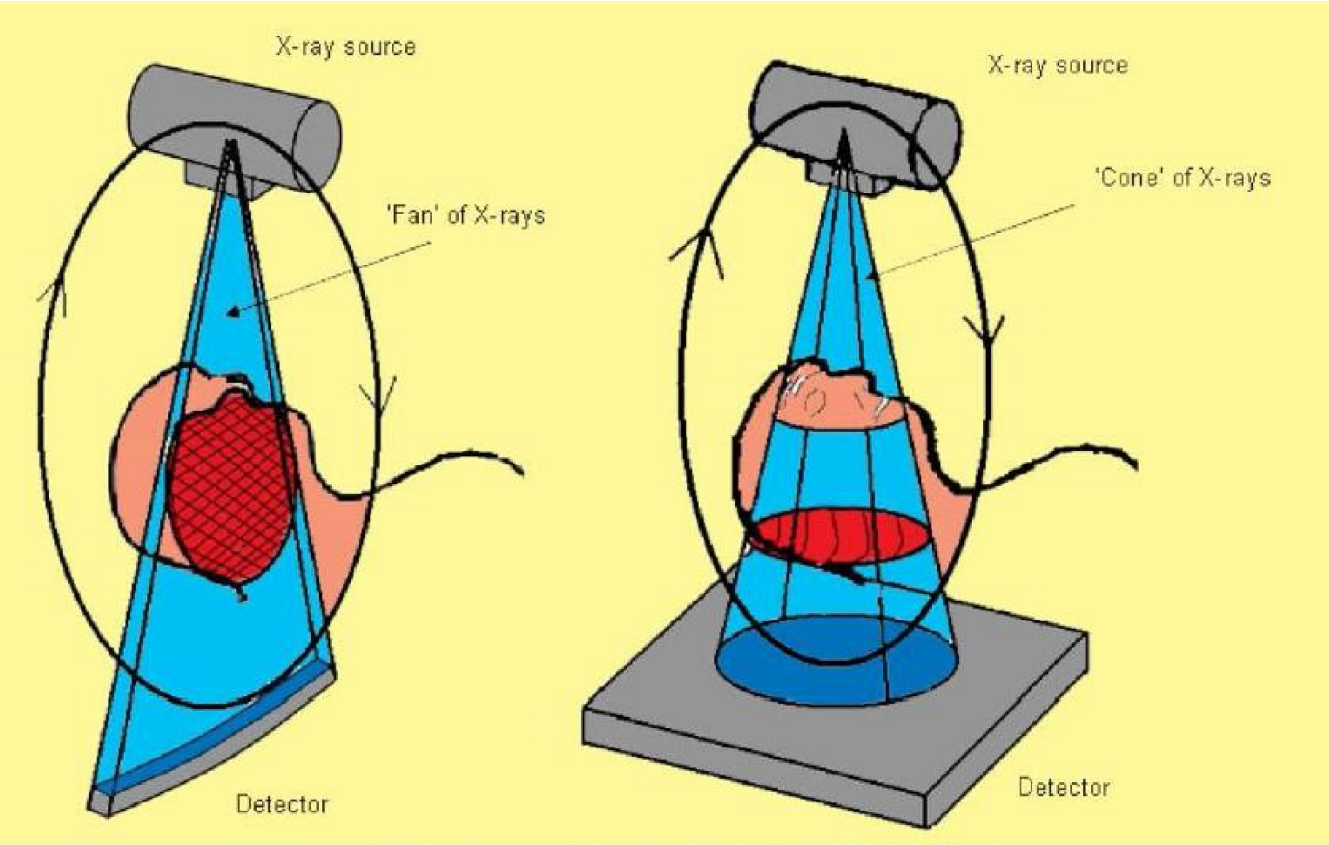
Parallel Beam



Fan Beam

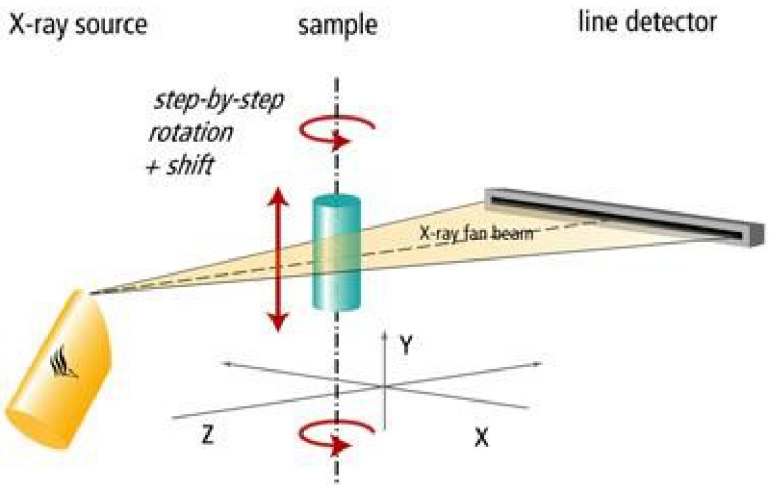


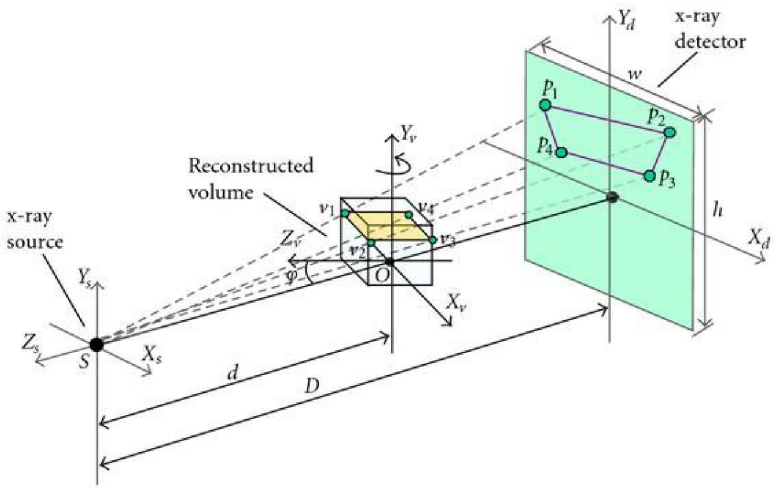
Fan vs. Cone Beam



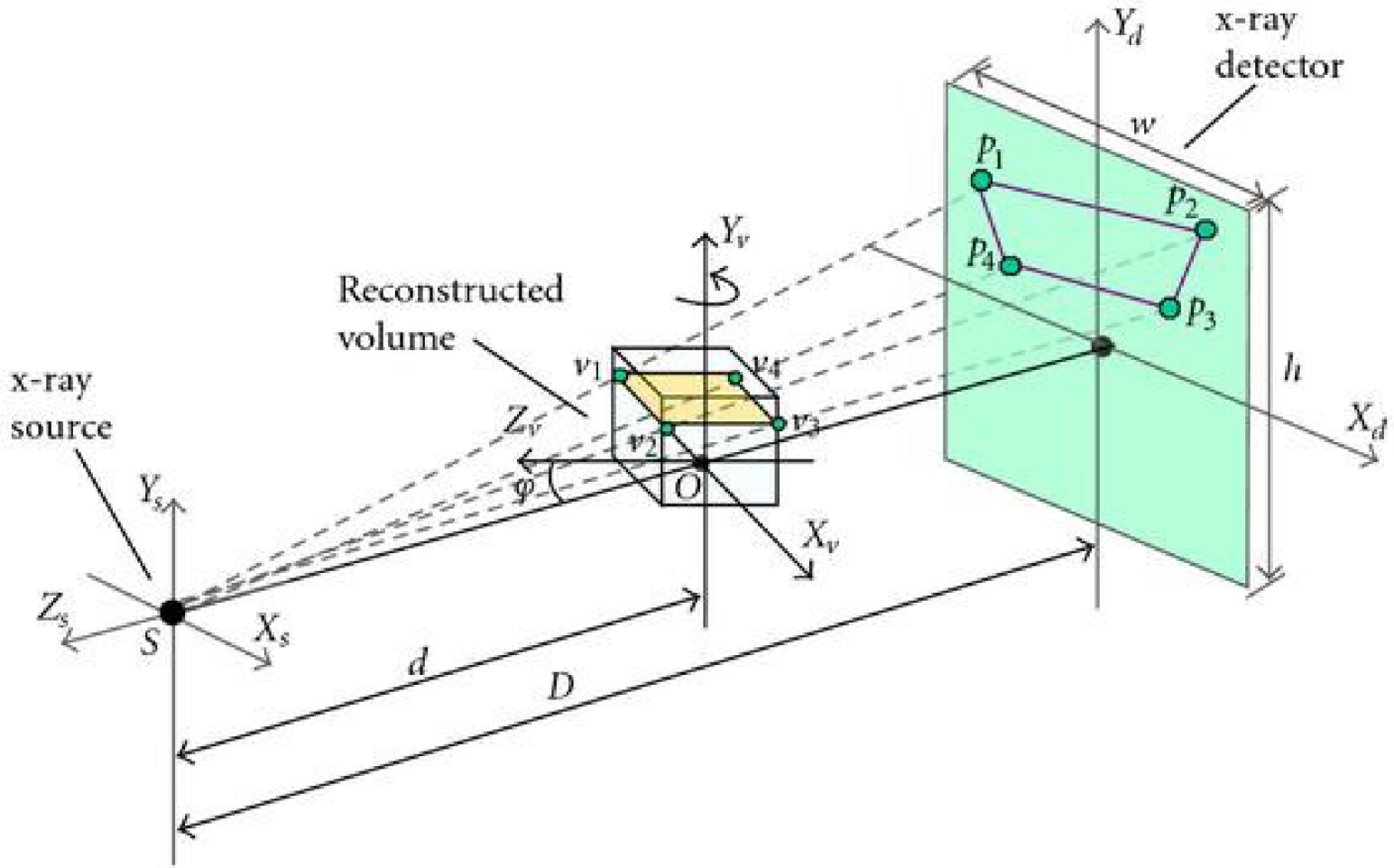
© J Can Dent Assoc 2006; 72(1); 75-80

Fan vs. Cone Beam



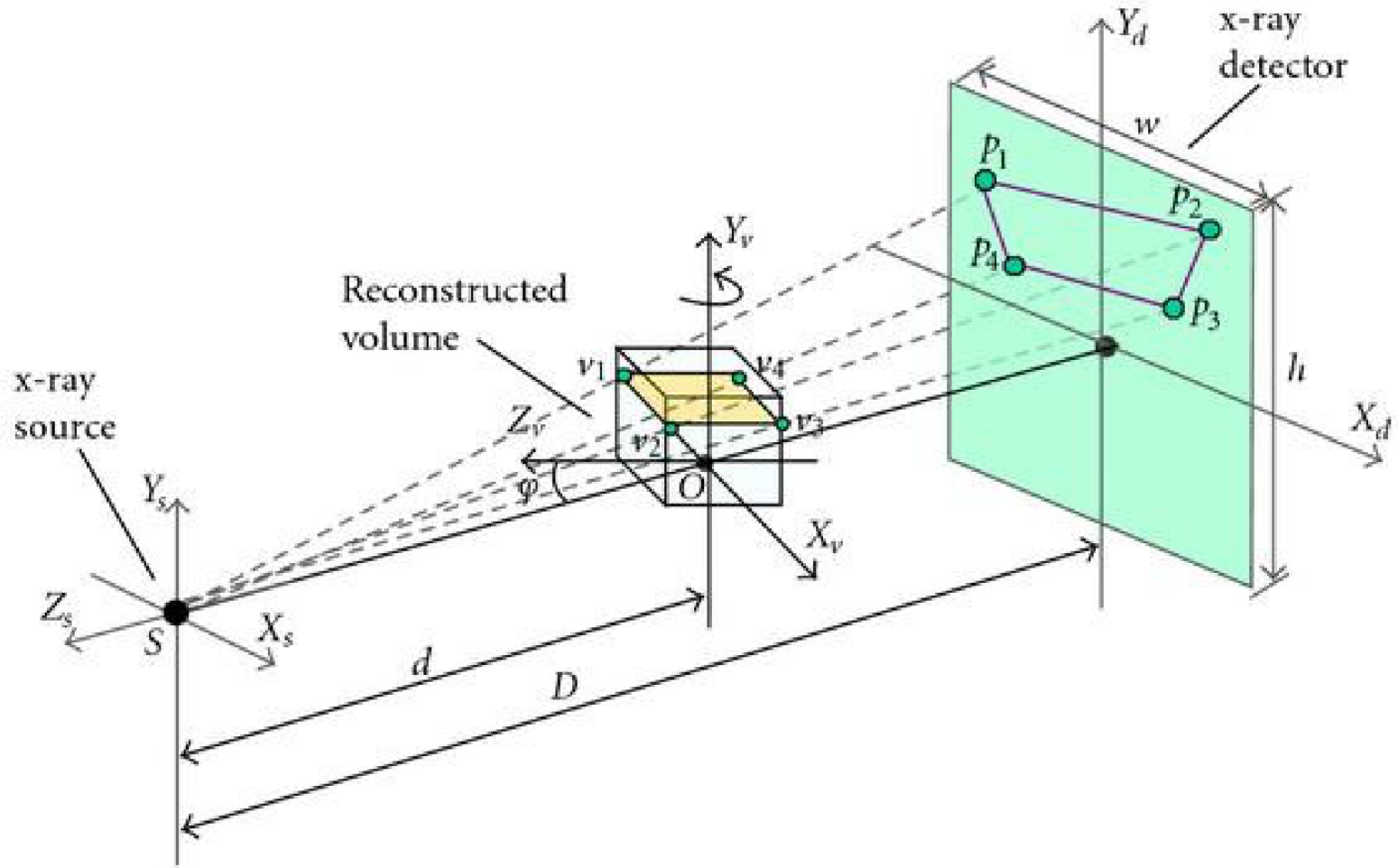


Cone Beam

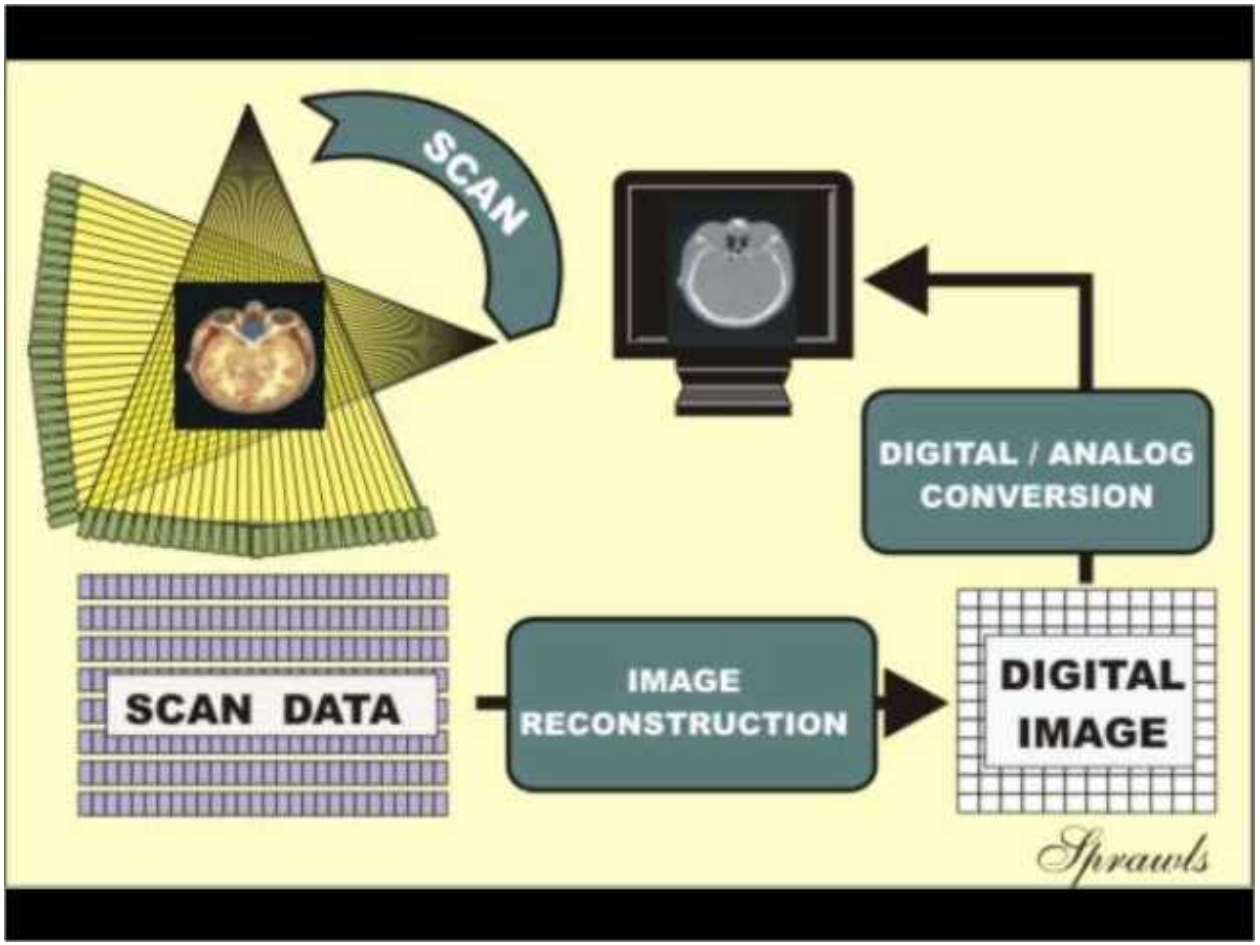

x-ray
source
Reconstructed
volume
x-ray
detector
Y_d
w
P_1
P_2
P_3
P_4
h
X_d
Y_v
v_1
v_4
v_2
v_3
Z_v
O
φ
X_v
Y_s
Z_s
S
X_s
d
D

CT Reconstruction

CT Reconstruction



CT Reconstruction



CT Reconstruction

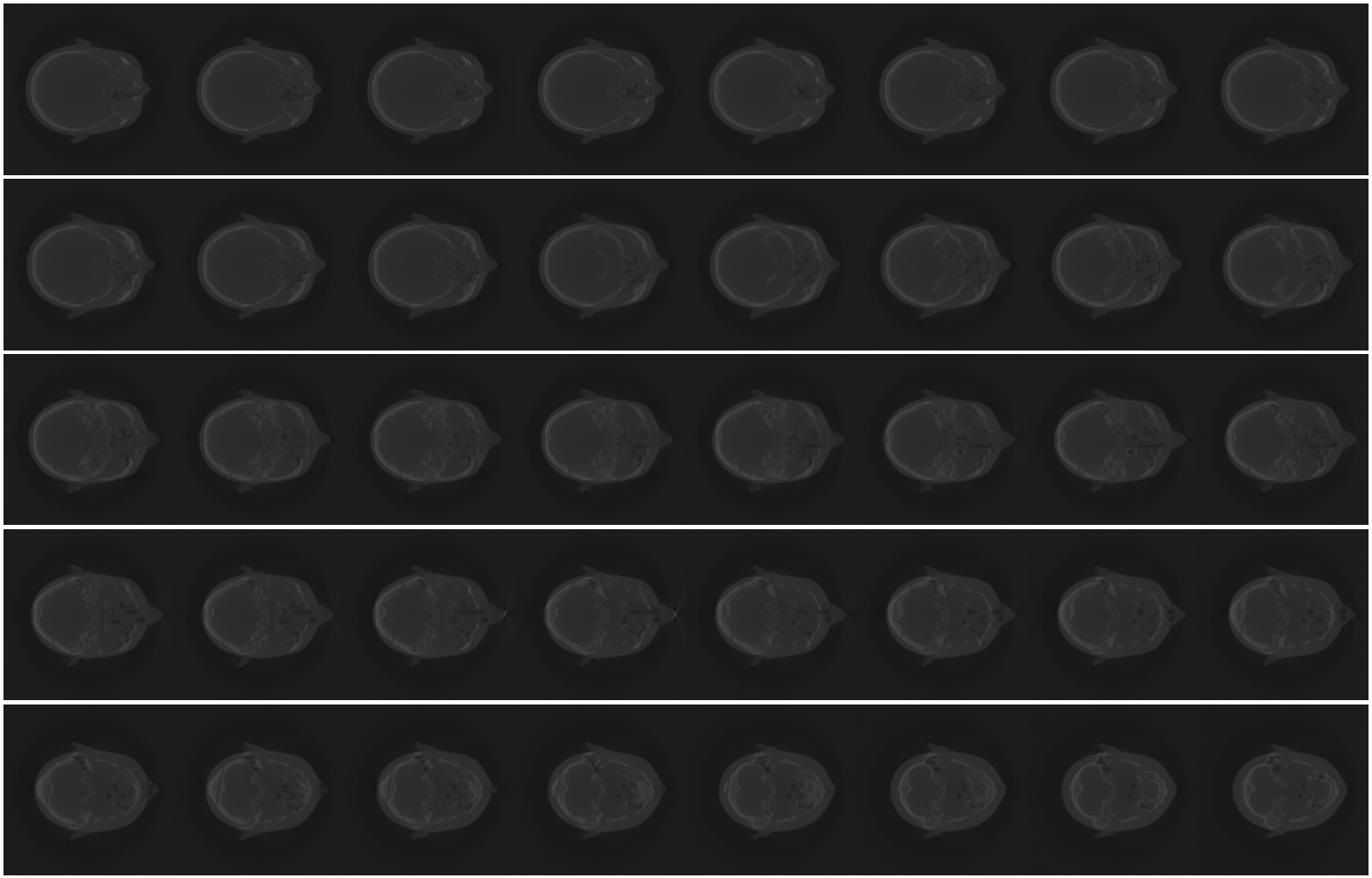
CT Reconstruction

CT Reconstruction

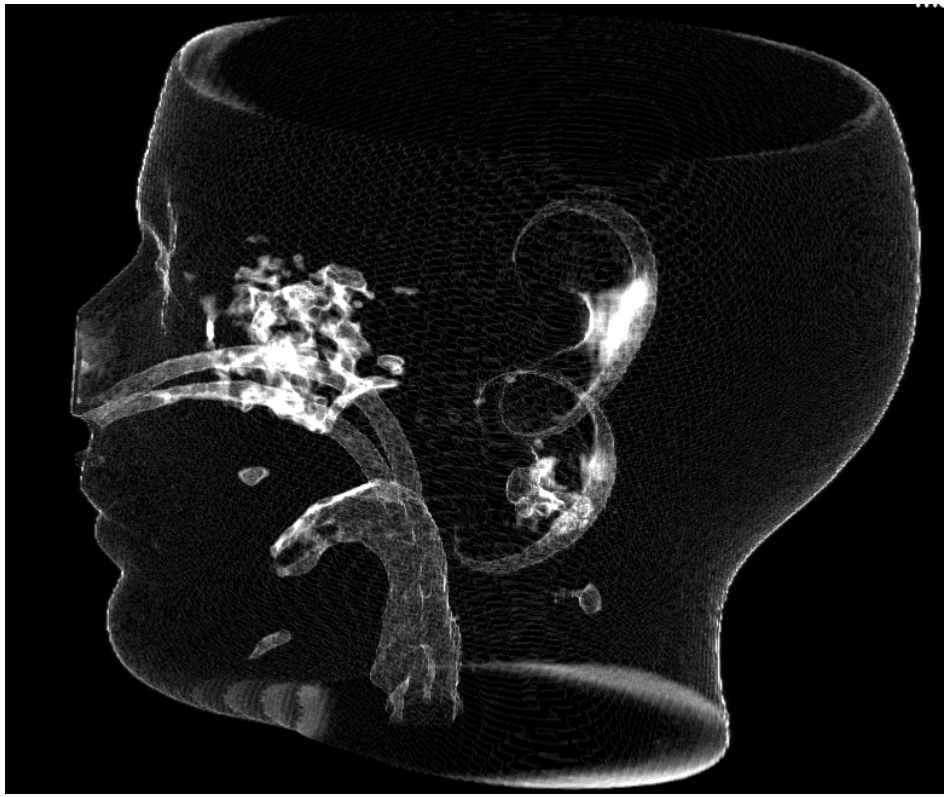
CT Reconstruction

CT Reconstruction

CT Reconstruction



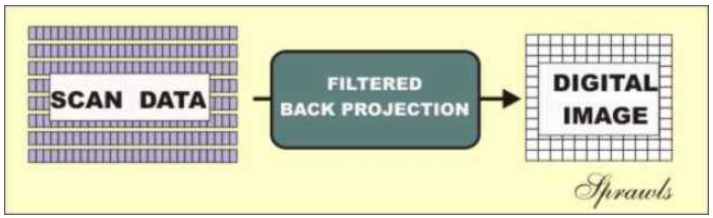
CT Reconstruction



Feldkamp, Davis, and Kress (FDK)

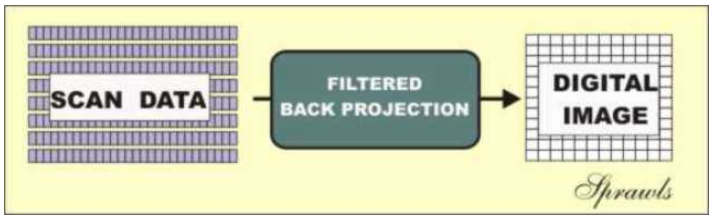
Cone Beam Reconstruction

- Projection Weighting
- Projection Filtering
- Back Projection
- Volume Weighting



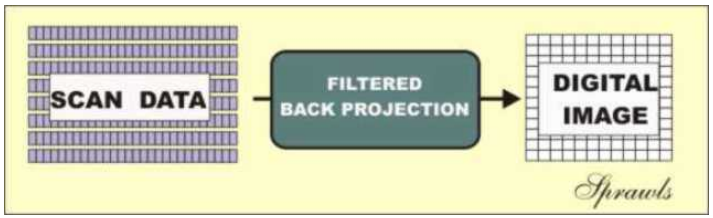
Cone Beam Reconstruction

- Projection Weighting
- Projection Filtering
- Back Projection
- Volume Weighting



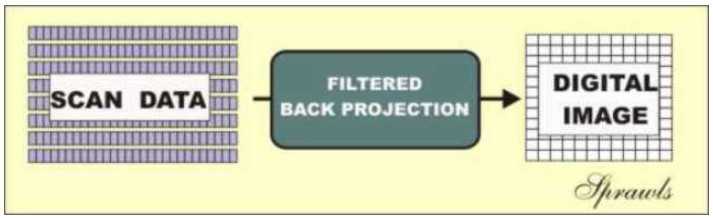
Cone Beam Reconstruction

- Projection Weighting
- Projection Filtering
- Back Projection
- Volume Weighting



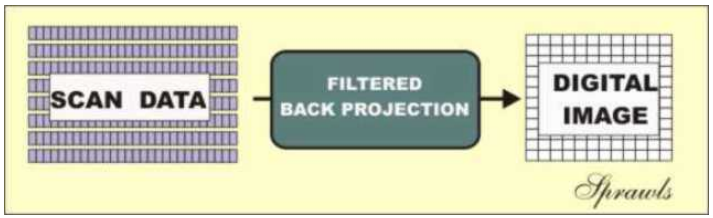
Cone Beam Reconstruction

- Projection Weighting
- Projection Filtering
- Back Projection
- Volume Weighting

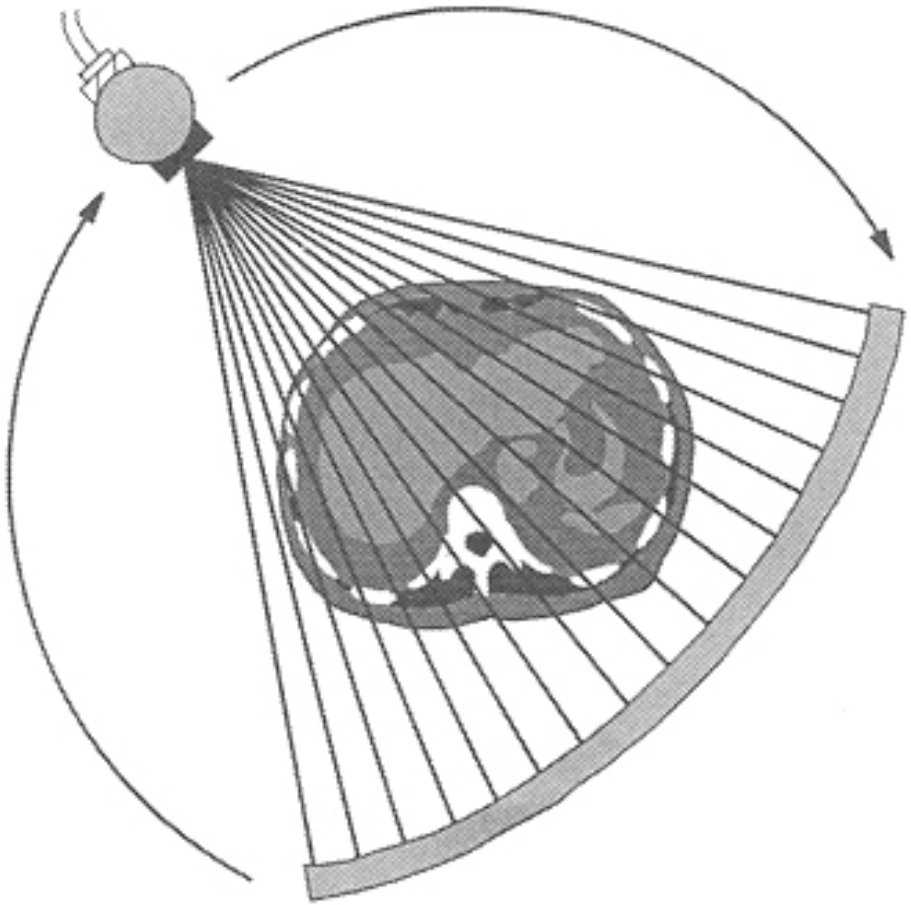


Cone Beam Reconstruction

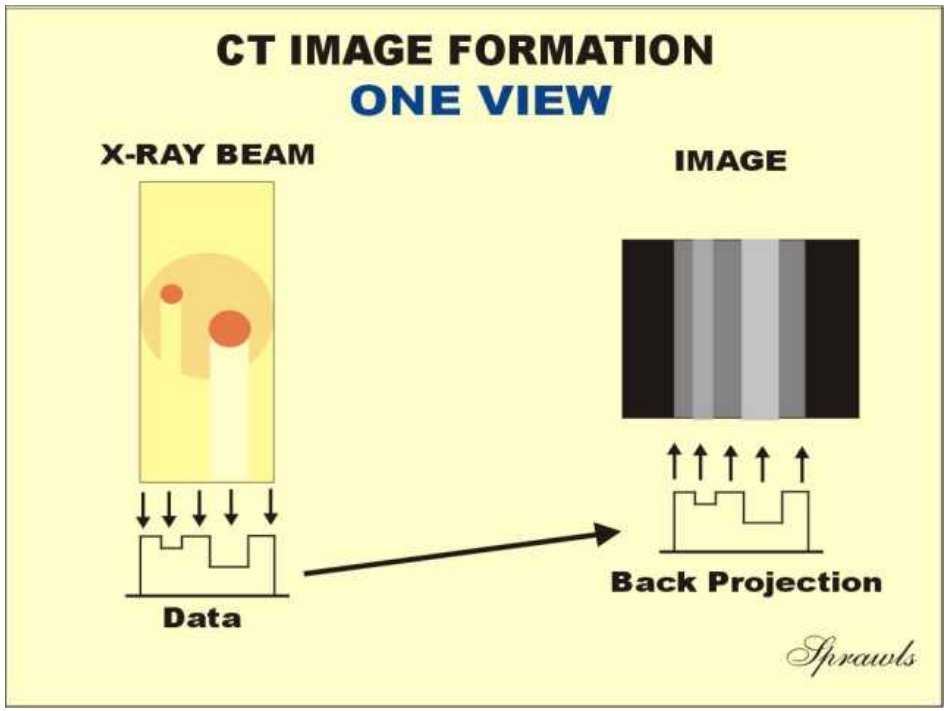
- Projection Weighting
- Projection Filtering
- Back Projection
- Volume Weighting



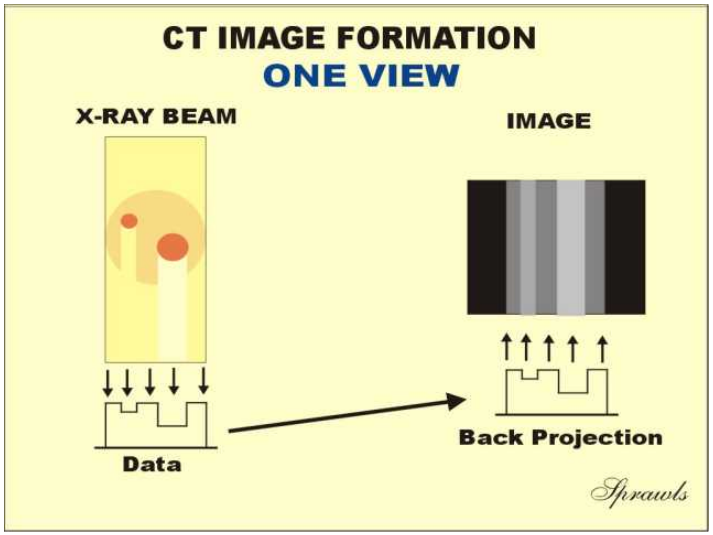
Projection Weighting

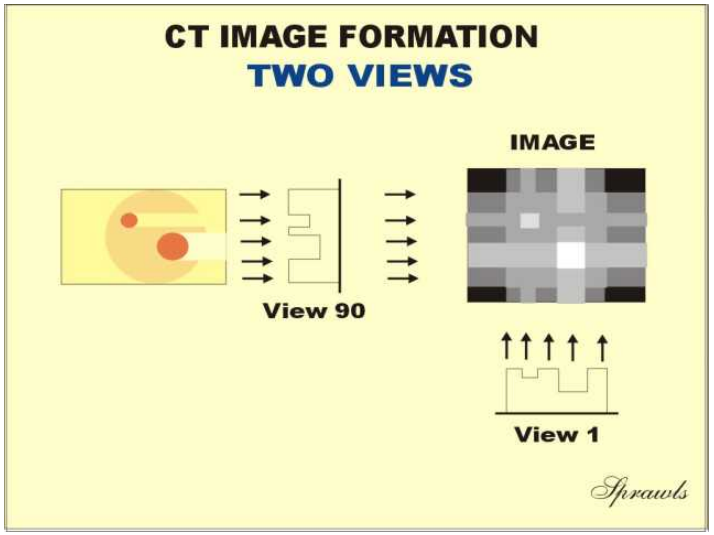


Back Projection

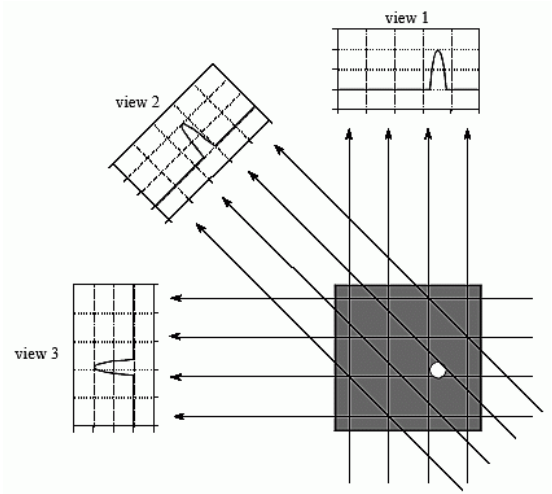


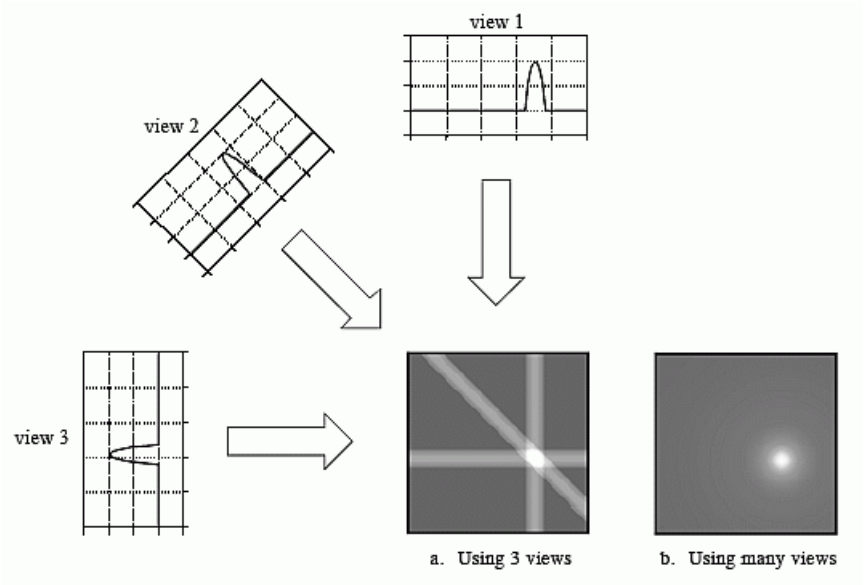
Back Projection



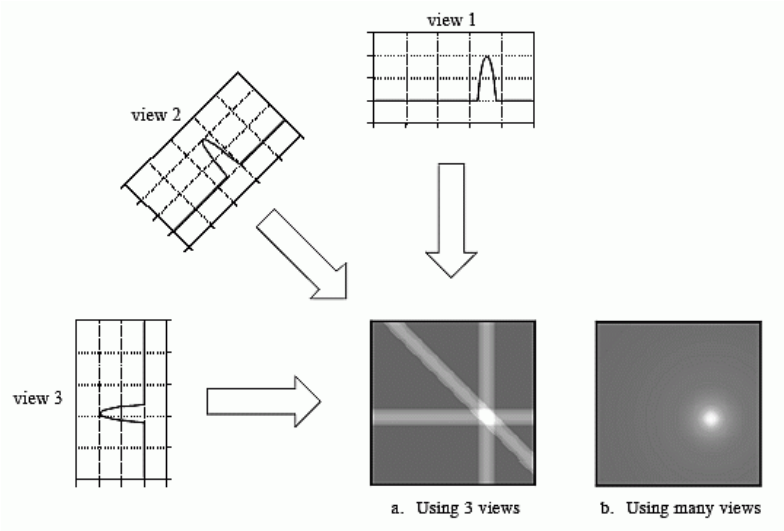


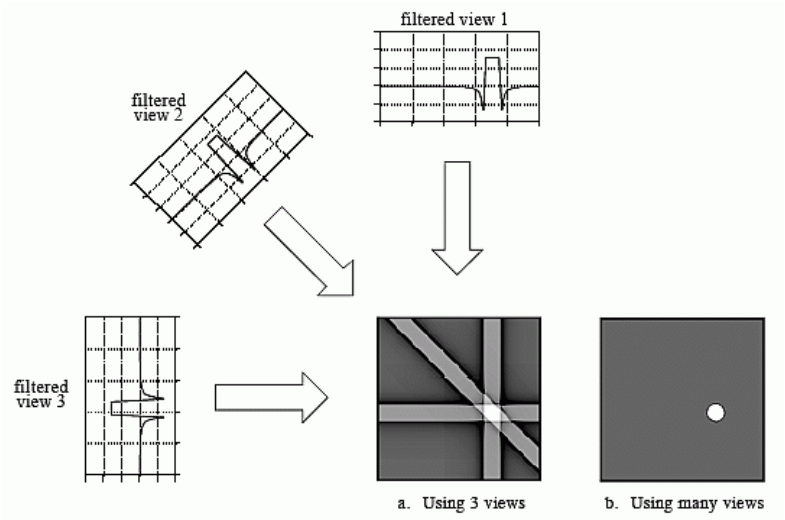
Back Projection



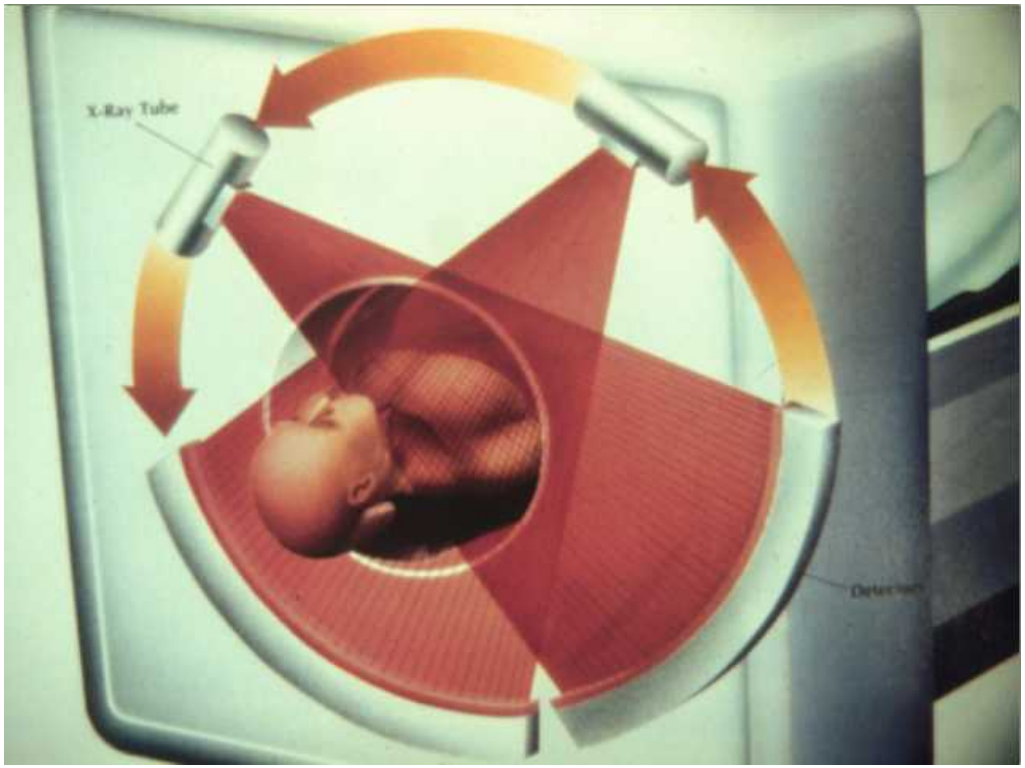


Filtered Back Projection





Volume Weighting



Assignment

Make a SMP version of FDK

Assignment files

- fdk.py
  - Main file used for benchmarks
- fdkvisual.py
  - Main file used for visualization
- fdkload.py
  - Module used for loading data
- fdkcore.py
  - Core reconstruction kernel to be modified

Assignment files

- fdk.py
  - Main file used for benchmarks
- fdkvisual.py
  - Main file used for visualization
- fdkload.py
  - Module used for loading data
- fdkcore.py
  - Core reconstruction kernel to be modified

Assignment files

- fdk.py
  - Main file used for benchmarks
- fdkvisual.py
  - Main file used for visualization
- fdkload.py
  - Module used for loading data
- fdkcore.py
  - Core reconstruction kernel to be modified

Assignment files

- fdk.py
  - Main file used for benchmarks
- fdkvisual.py
  - Main file used for visualization
- fdkload.py
  - Module used for loading data
- fdkcore.py
  - Core reconstruction kernel to be modified

Assignment files

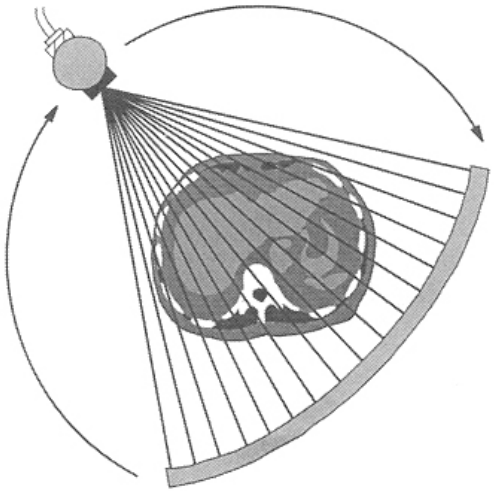
- fdk.py
  - Main file used for benchmarks
- fdkvisual.py
  - Main file used for visualization
- fdkload.py
  - Module used for loading data
- fdkcore.py
  - Core reconstruction kernel to be modified

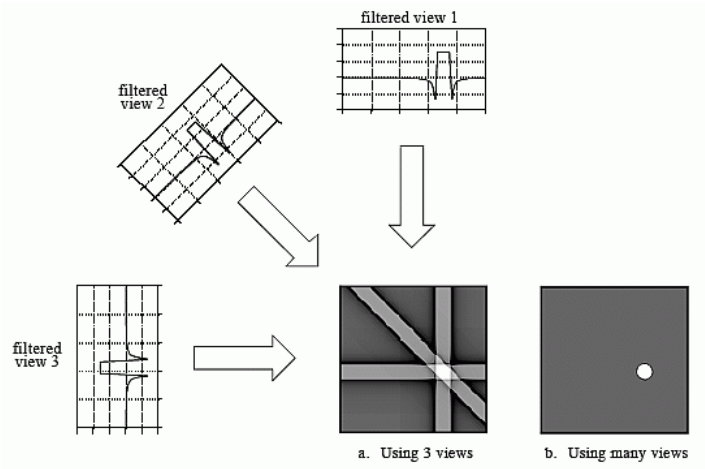
FDK core kernel

```
1  for p in xrange(nr_projections):
2     for z in xrange(z_voxels):
3        combined_matrix[2, :] = z_voxel_coords[z]
4
5        vol_det_map = dot(transform_matrix[p], combined_matrix)
6
7        map_cols = rint(divide(vol_det_map[0, :],
8                               vol_det_map[2,:])).astype(int32)
9
10       map_rows = rint(divide(vol_det_map[1, :],
11                              vol_det_map[2,:])).astype(int32)
12
13       mask = (map_cols >= 0) & (map_rows >= 0) & \
14              (map_cols < detector_columns) & \
15              (map_rows < detector_rows)
16
17       proj_indexs = map_cols * mask + \
18                     map_rows * mask * detector_columns
19
20       recon_volume[z].flat += flat_proj_data[proj_indexs] * \
21                               volume_weight[p] * mask
```

FDK predefined data

- projections.bin
  - Matrix with weighted and filtered 2D X-Ray images





FDK predefined data

- combined.bin
  - Matrix with all combinations of all X,Y coordinates for the 3D

	Voxel 0	**Voxel 1**	**Voxel . . .**	**Voxel X*Y**
X	val	val	. . .	val
Y	val	val	. . .	val
Z	-	-	-	-
Dummy	1	1	1	1

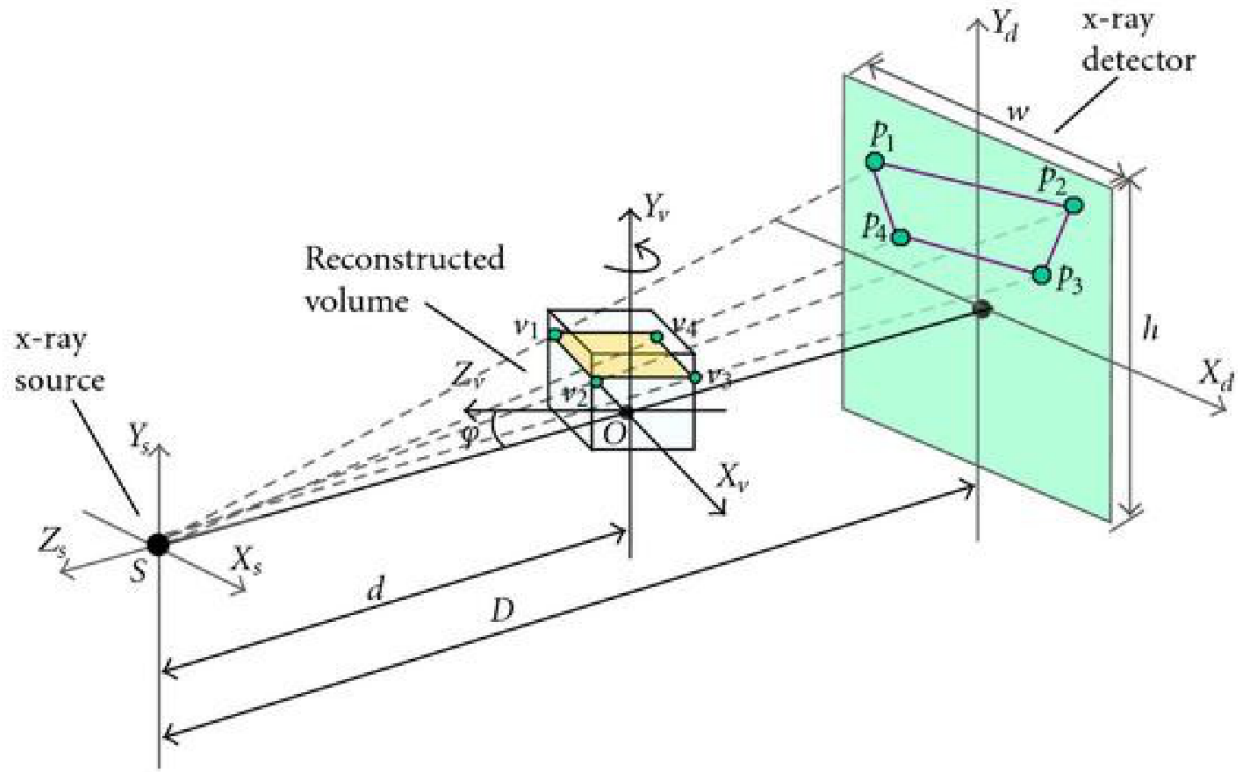
FDK predefined data

- z_voxel_coords.bin
  - Matrix with Z coordinates for the 3D volume

	Voxel 0	**Voxel 1**	**Voxel . . .**	**Voxel X*Y**
Z	val	val	. . .	val

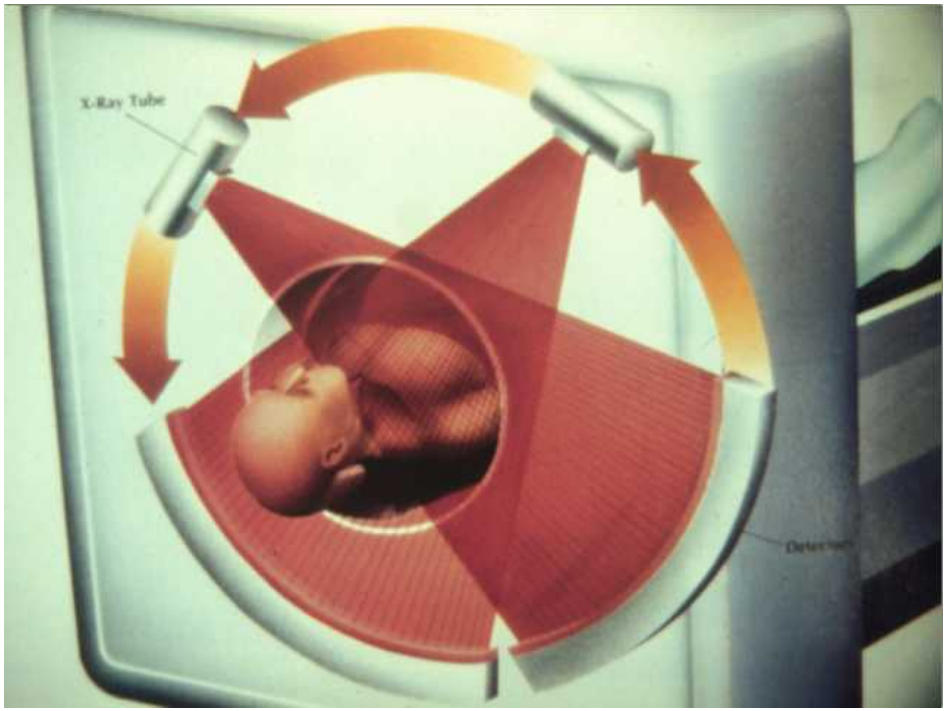
FDK predefined data

- transform.bin
  - Matrix used to map 3D coordinates to 2D coordinates



FDK predefined data

- volumeweight.bin
  - Post weight to compensate for the cone effect of the X-Ray beam

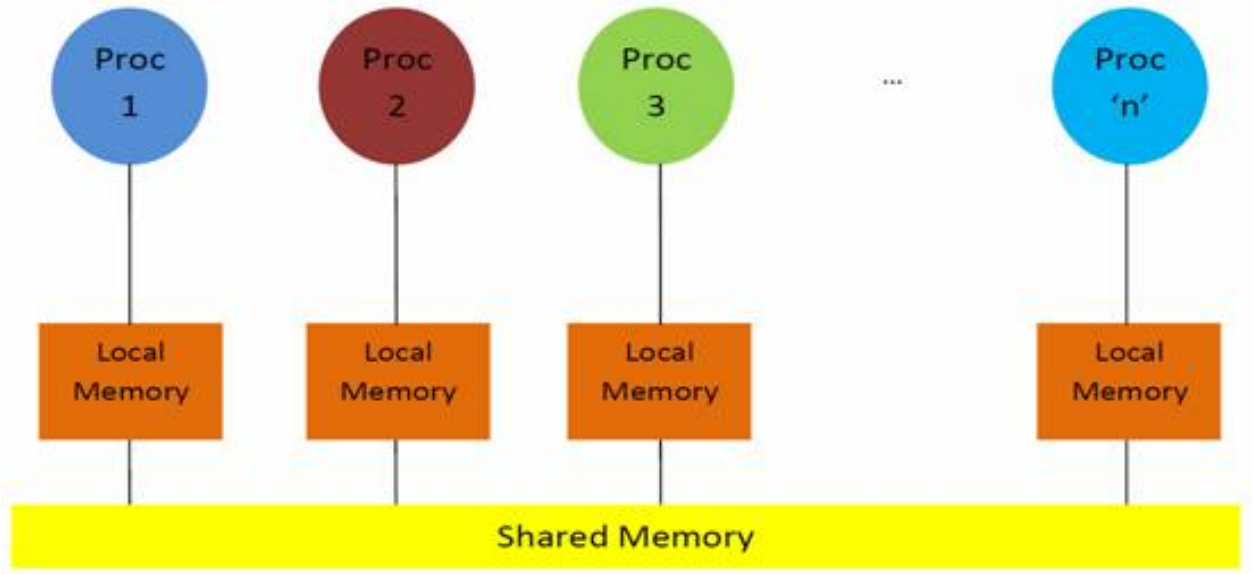


FDK core kernel

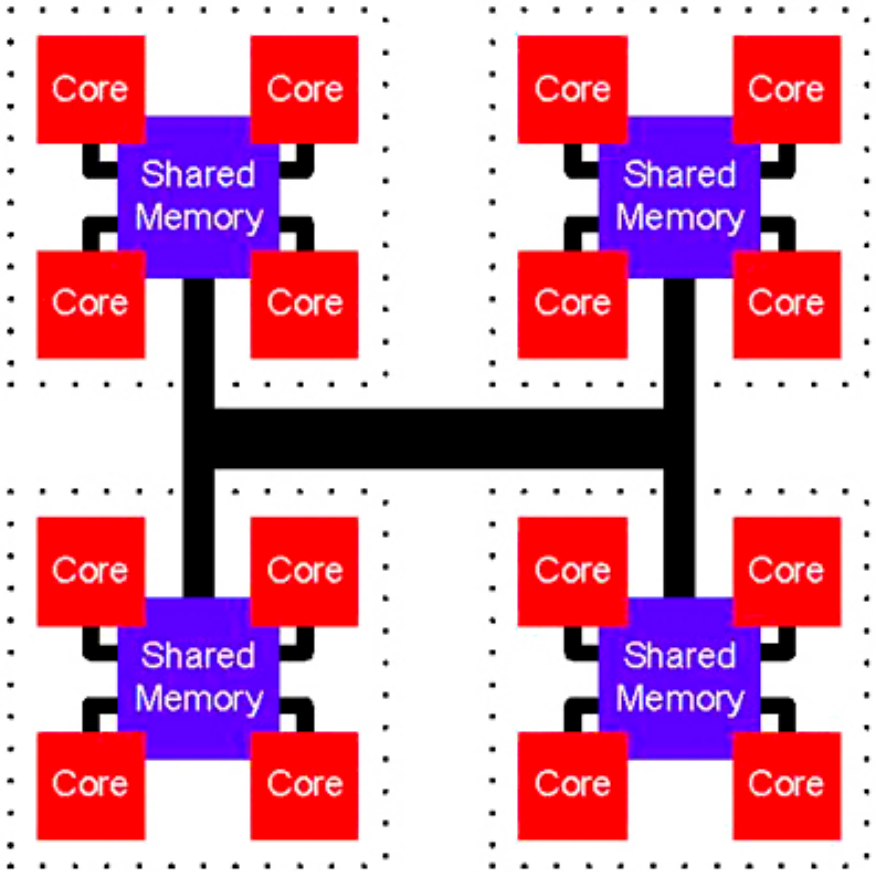
```
1  for p in xrange(nr_projections):
2     for z in xrange(z_voxels):
3        combined_matrix[2, :] = z_voxel_coords[z]
4
5        vol_det_map = dot(transform_matrix[p], combined_matrix)
6
7        map_cols = rint(divide(vol_det_map[0, :],
8                               vol_det_map[2,:])).astype(int32)
9
10       map_rows = rint(divide(vol_det_map[1, :],
11                              vol_det_map[2,:])).astype(int32)
12
13       mask = (map_cols >= 0) & (map_rows >= 0) & \
14              (map_cols < detector_columns) & \
15              (map_rows < detector_rows)
16
17       proj_indexs = map_cols * mask + \
18                     map_rows * mask * detector_columns
19
20       recon_volume[z].flat += flat_proj_data[proj_indexs] * \
21                               volume_weight[p] * mask
```

Programming SMP systems

Architecture overview



Architecture overview



Python Multiprocessing

```
1  from multiprocessing import Process
2  import numpy
3  import shmarray
4
5  def worker(id, result):
6      ``''''thread worker function''''''
7      result[id,:] = numpy.random.random(10)
8
9  def main():
10     workers = 4
11
12     data = shmarray.create((workers, 10), numpy.float)
13
14     processes = [Process(target=worker, args=(i, data)) \
15                  for i in xrange(workers)]
16
17     for p in processes: p.start()
18     for p in processes: p.join()
19
20 if __name__ == `__main__':
21     main()
```

Python Multiprocessing

```
1  from multiprocessing import Process, Lock
2  import numpy
3  import shmarray
4
5  def worker(id, result, lock):
6      ``''''thread worker function''''''
7      lock.acquire(True)
8      result[:] += numpy.random.random(10)
9      lock.release()
10
11 def main():
12     lock = Lock()
13     workers = 4
14
15     data = shmarray.create((10), numpy.float)
16
17     processes = [Process(target=worker, args=(i, data, lock)) \
18                  for i in xrange(workers)]
19
20     for p in processes: p.start()
21     for p in processes: p.join()
22
23 if __name__ == `__main__':
24     main()
```

Python Multiprocessing

```
1 def worker(id, result):
2     `''''thread worker function''''''
3         result[id,:] = numpy.random.random(10)
```

```
1 def worker(id, result, lock):
2     ``''''thread worker function''''''
3     lock.acquire(True)
4     result[:] += numpy.random.random(10)
5     lock.release()
```

The Assignment

- SMP: Simpler programming model than OpenCL
- But expect a lot smaller speedup

Recon Execution time 256x256x256	
Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz:	488.8565s
NVIDIA GTX-680:	1.8033s
NVIDIA GTX-680:	1.0250s

The Assignment

- SMP: Simpler programming model than OpenCL
- But expect a lot smaller speedup

Recon Execution time 256x256x256	
Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz:	488.8565s
NVIDIA GTX-680:	1.8033s
NVIDIA GTX-680:	1.0250s

The Assignment

- SMP: Simpler programming model than OpenCL
- But expect a lot smaller speedup

Recon Execution time 256x256x256

Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz:	488.8565s
NVIDIA GTX-680:	1.8033s
NVIDIA GTX-680:	1.0250s

The Assignment

- SMP: Simpler programming model than OpenCL
- But expect a lot smaller speedup

Recon Execution time 256x256x256

Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz: 488.8565s

NVIDIA GTX-680: 1.8033s

NVIDIA GTX-680: 1.0250s

The Assignment

- SMP: Simpler programming model than OpenCL
- But expect a lot smaller speedup

Recon Execution time 256x256x256

Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz:	488.8565s
NVIDIA GTX-680:	1.8033s
NVIDIA GTX-680:	1.0250s

The Assignment

- SMP: Simpler programming model than OpenCL
- But expect a lot smaller speedup

Recon Execution time 256x256x256

Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz:	488.8565s
NVIDIA GTX-680:	1.8033s
NVIDIA GTX-680:	1.0250s

The Assignment

To be released the 25th February at 15.00

Remember:

This is a SMP assignment

The Assignment

To be released the 25th February at 15.00

Remember:

This is a SMP assignment

The Assignment

To be released the 25th February at 15.00

Remember:

This is a SMP assignment