Nuts & Bolts of Advanced Imaging

The Image Reconstruction Pipeline

Michael S. Hansen, PhD

Magnetic Resonance Technology Program

National Institutes of Health, NHLBI

NIH National Heart, Lung, and Blood Institute

Declaration of Financial Interests or Relationships

Speaker Name: Michael S. Hansen

I have the following financial interest or relationship to disclose with regard to the subject matter of this presentation:

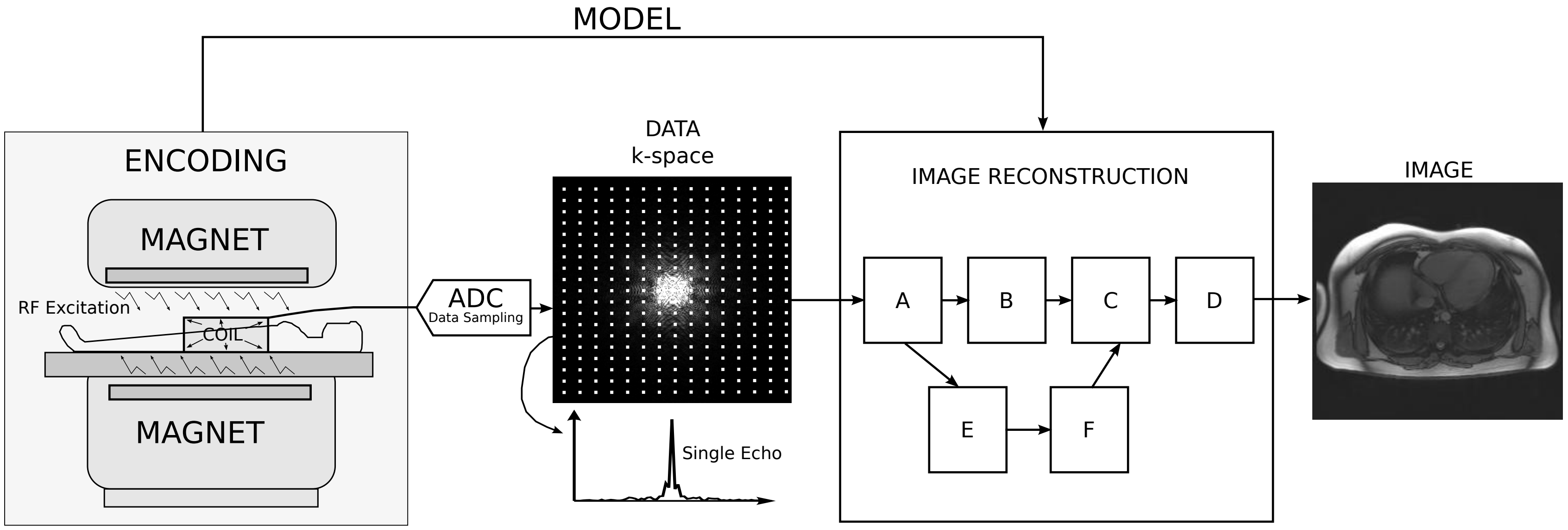
Company Name: Siemens Medical Solutions
Type of Relationship: Research Agreement

Outline

- What is a reconstruction pipeline
- Common pipeline elements:
  - Noise adjust, filtering, accumulation, FFT
- A Simple Reconstruction Pipeline Example
  - Cartesian Parallel Imaging
- Examples of pipeline architectures
  - Open Source
  - Vendors

PART 1

The role of the image reconstruction process



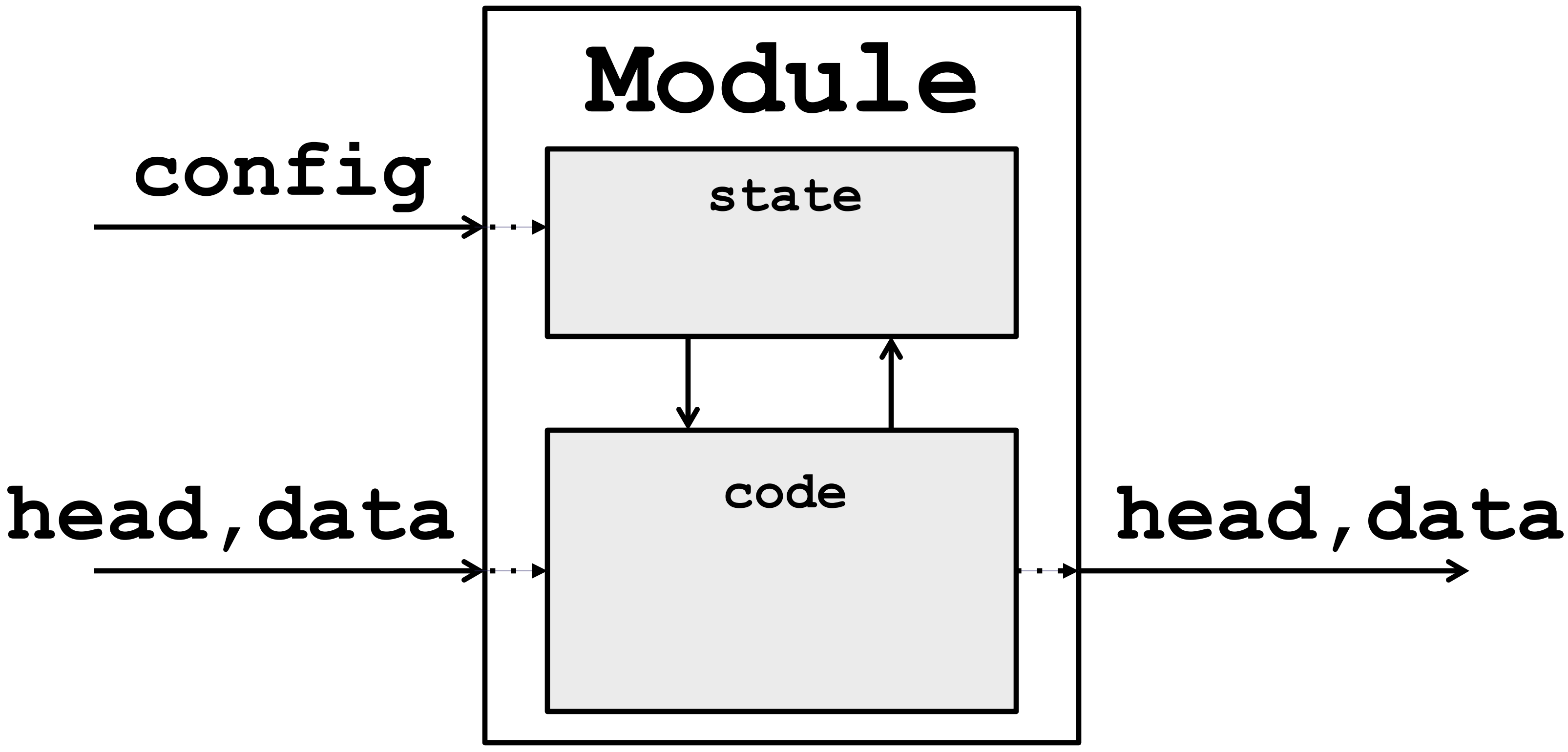
Reconstruction Pseudo Code

```
function reconstruct(datafile):
```

Problems with the `reconstruct` function

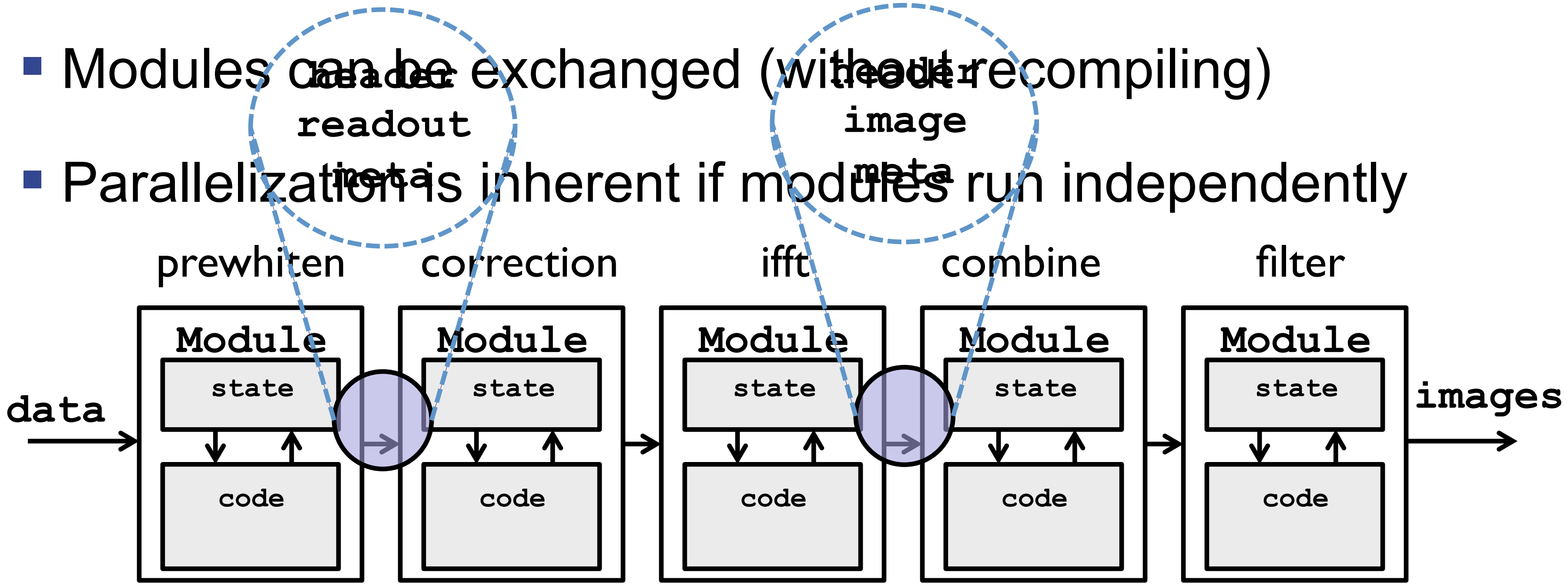
- Reconstruction does not start until all data is stored
- Parallelization requires low level management
- Changes to reconstruction software requires editing of source code function (and recompilation of code)
- Encourages bad programming practices:
  - Poorly defined data structures (interfaces)
  - Duplication of code

A modular pipeline

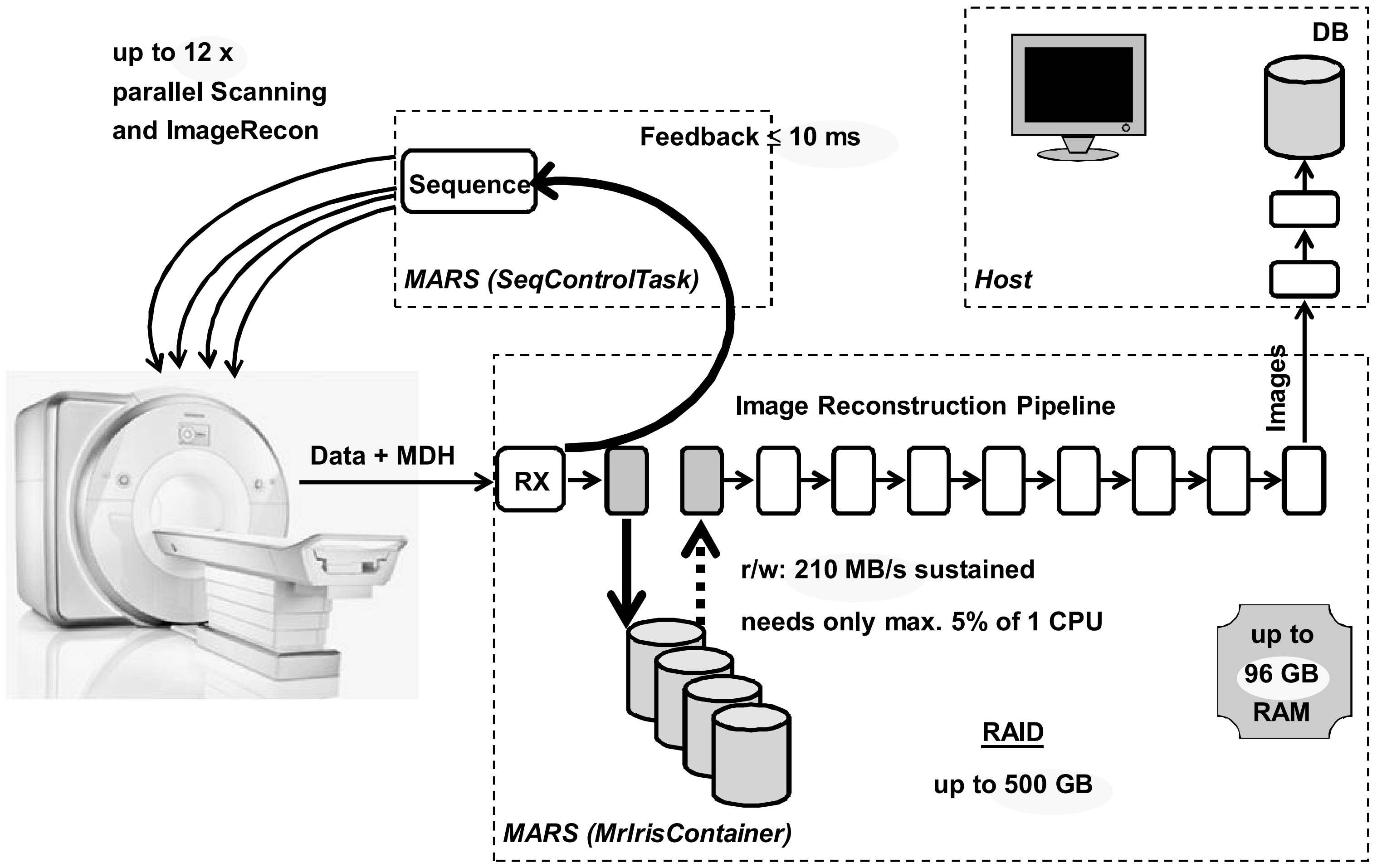


A modular pipeline

- Modular reconstruction design
- Well defined interfaces and data structures
- Processing can start when first readout is acquired
- Modules can be exchanged (without recompiling)
- Parallelization is inherent if modules run independently

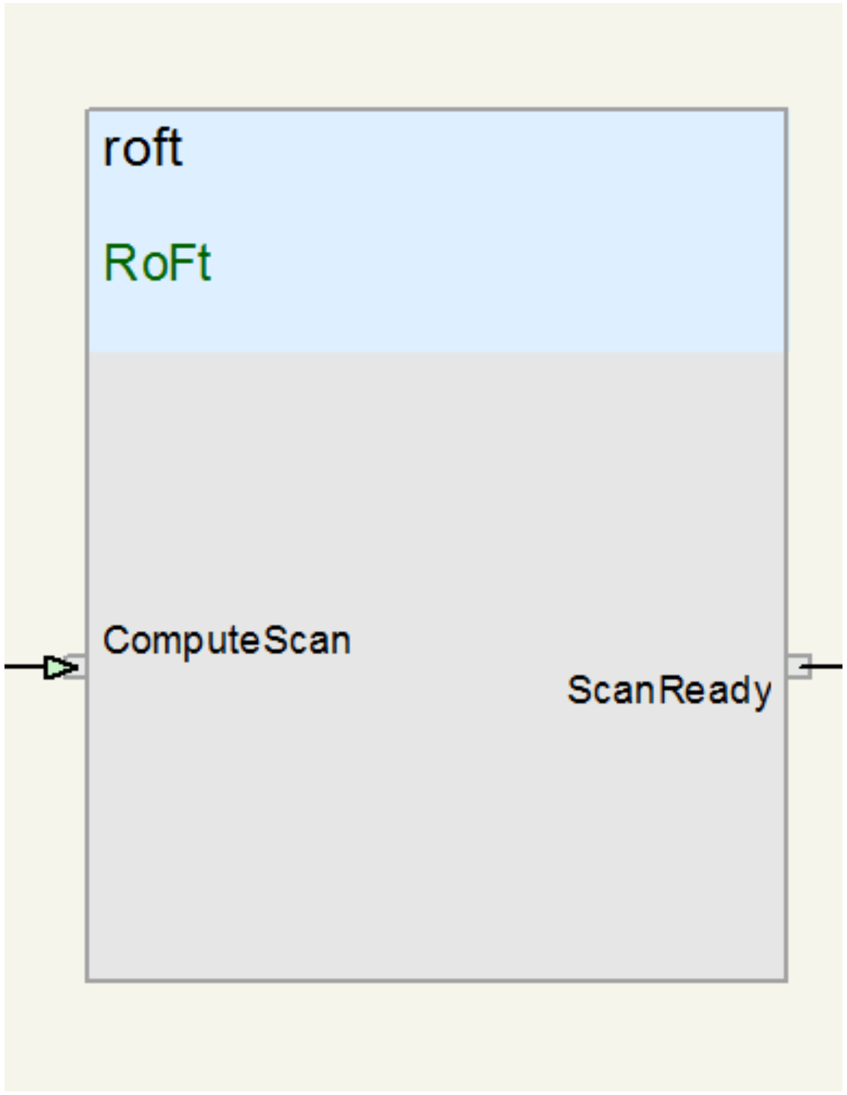


Siemens Reconstruction Pipeline



Functor

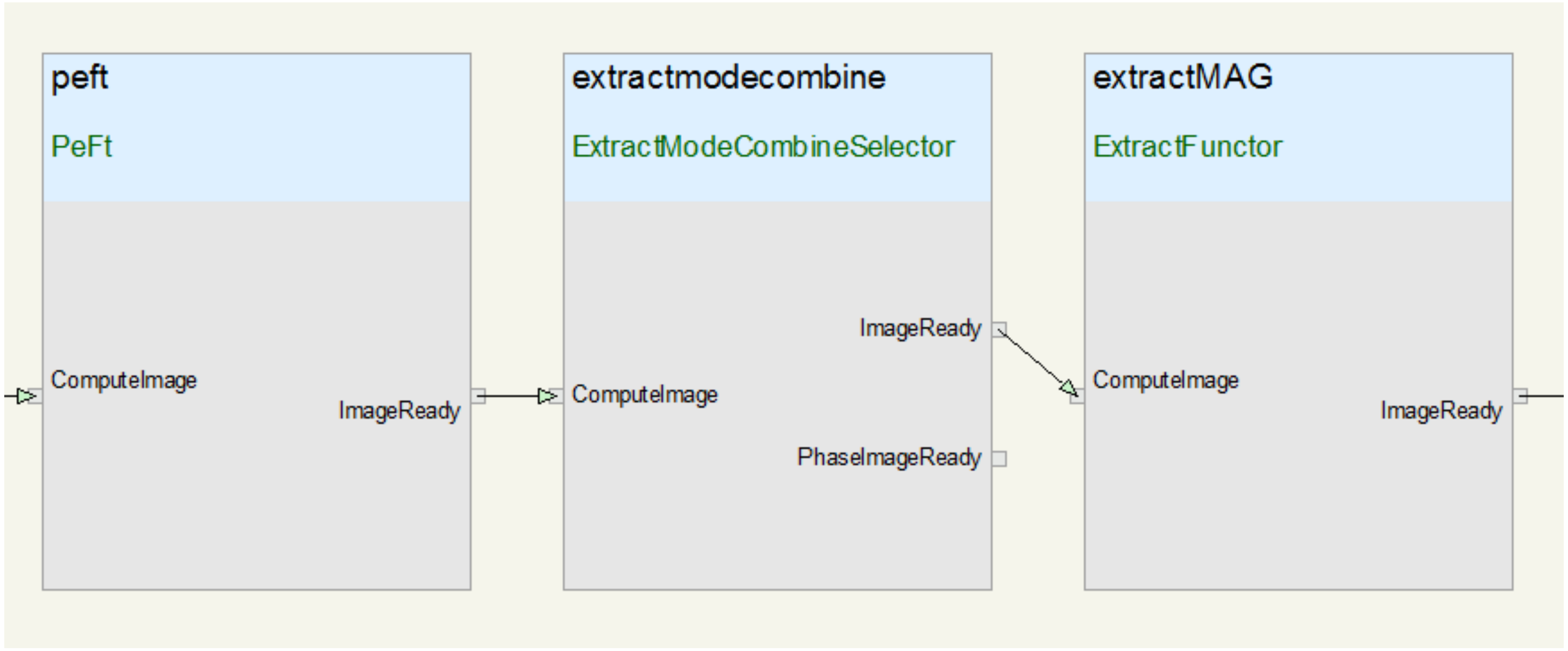
- Modules are called functors.
- Data structures are well defined on interfaces.
- Modules are interchangeable.
- Configured at run time.



SIEMENS

Functor Chain

The user interfaces with the underlying framework through specification of a file known as the Ice program. This program specifies the functor chain of operations and configuration parameters, based upon the type of pulse sequence and the desired data output (integer images, complex raw data).



The Basics of a Reconstruction Application

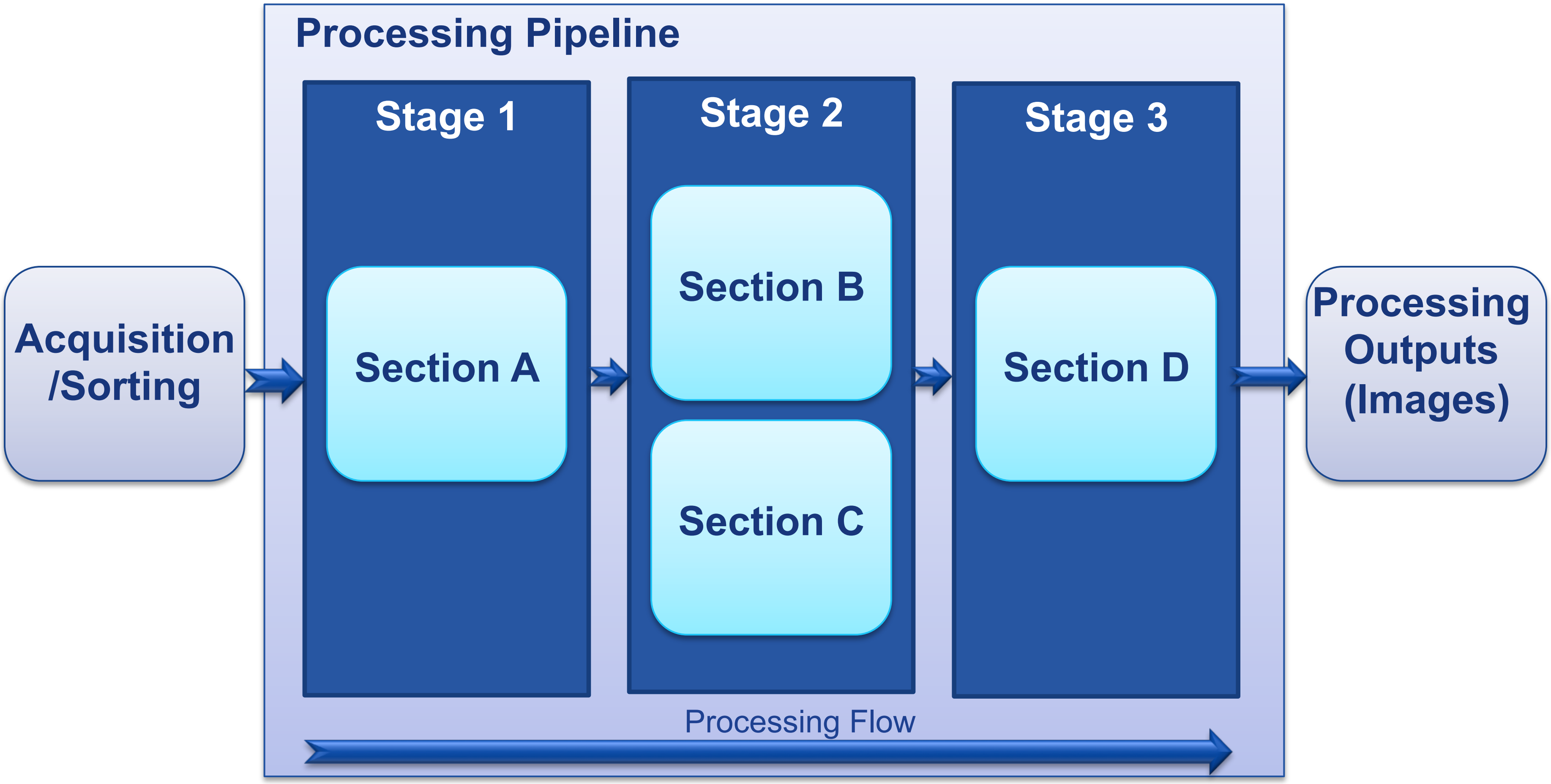
Data Sorting and Organization

- Meaningful data structure based on application or desired processing
- Raw acquisition data can be: scanner, raw files, data streams

Processing Pipeline

- Group of **Processing Sections**: Modules that typically “do math” and have custom inputs/output
- The pipeline builds and wires the **Sections** in **stages** that define the processing **flow**
- Pipeline building defines:
  - Processing order
  - Section dependencies
  - Distribution of processing

Example


Processing Pipeline
Stage 1
Stage 2
Stage 3
Acquisition /Sorting
Section A
Section B
Section C
Section D
Processing Outputs (Images)
Processing Flow

Programming Window: The Orchestra SDK

Collection of modular and reusable **product** recon algorithms

- MATLAB functions
- C++ classes and functions

DICOM Toolbox

- Create, read, and write compliant files
- Store/stream images to networked peers

Multi-Platform

- Linux, Mac, Windows

Raw file readers

- APIs for accessing data and parameters

GE example pipelines

- Cartesian, EPI, Spectroscopy

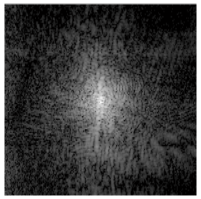
Data Stream

Measurement

RAW MR Signal

Reconstruction

Preprocessing

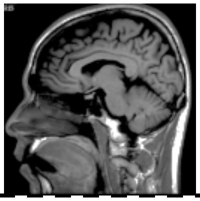


RAW Data: Samples (Complex Floating Point)

FFT

CPX Data: Image (Complex Floating Point)

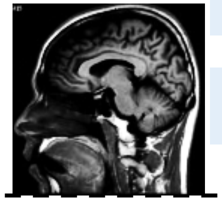
Image Production & Scaling



REC Data: Image (Integer 8,12 bit)

Viewing

Rescaling



Display Grey Levels (after Windowing)

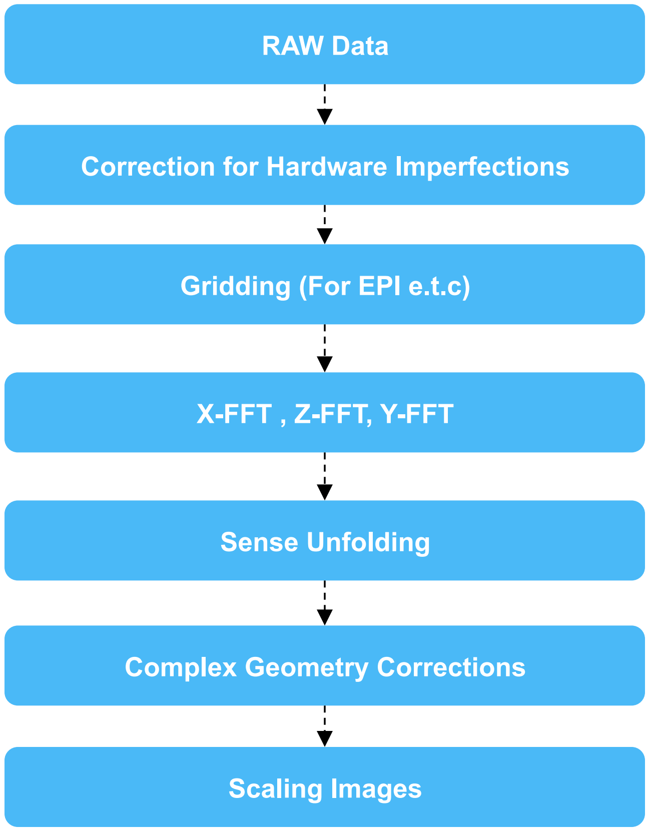
Introduction to Recon 2.0

Nodes, Reconstruction Graph

Reconstruction is a sequence of steps for transforming data received from the previous step and passing it onto the next step.

Nodes The individual steps performed in the reconstruction pipeline are referred to as Nodes.

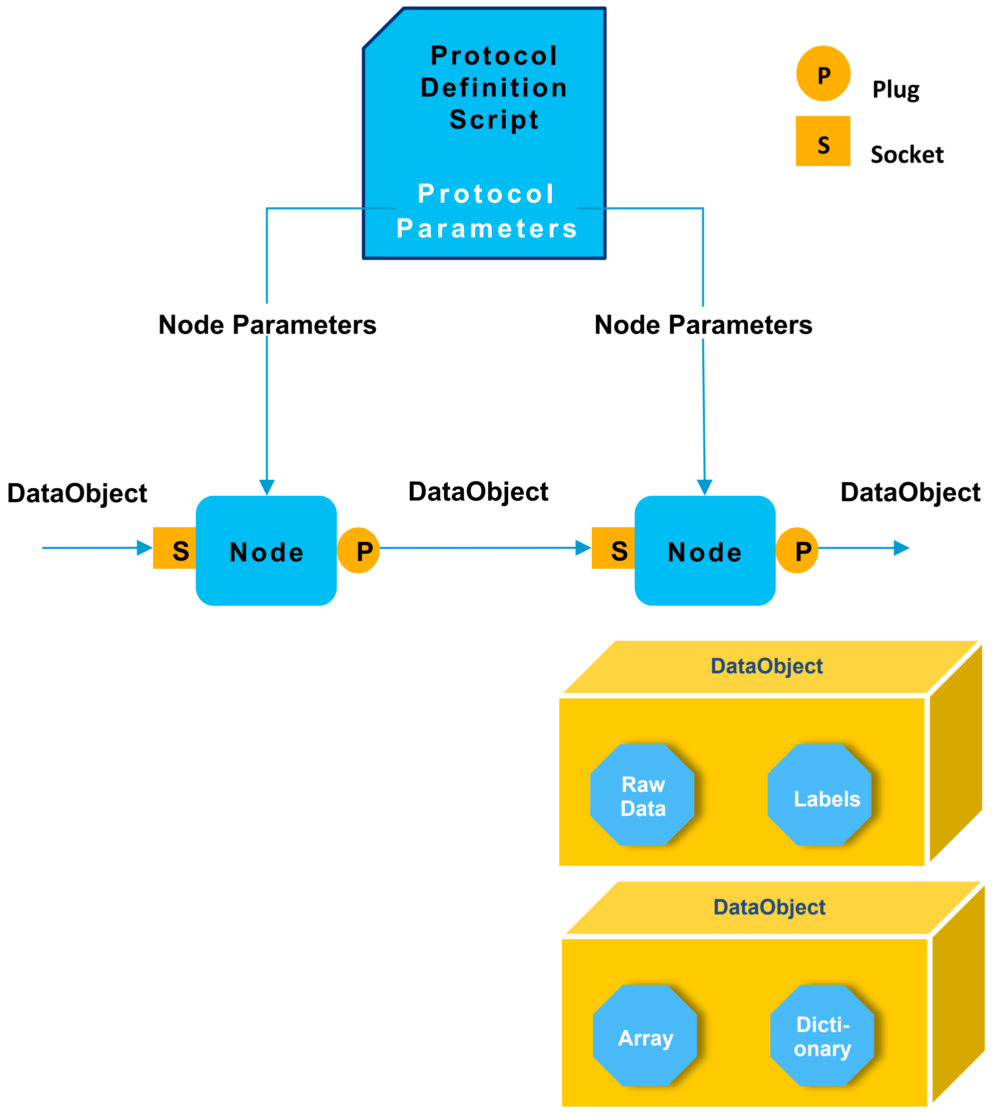
Reconstruction Graph The pipeline connecting the nodes to each other is called as the reconstruction graph.



- ready-to-use platform for complete MR image reconstruction
- raw data processed via independent, easy to adapt processing modules called “nodes”

Introduction to Recon 2.0

Framework Architecture



PHILIPS

Summary

- Modern reconstruction software is often implemented as streaming pipeline architectures
  - Modularity
  - Performance
- Modules have well defined data interfaces and are interchangeable to some extend
- Modules can often be assembled at run-time to form different recon programs (without recompilation)

What to expect in Part 2

- We will play with a simple recon pipeline:
  - Built in Python
  - Basic Parallel Imaging reconstruction
- Open Raw Data Standard
  - ISMRMRD
- Open Source Pipeline Environments
  - Gadgetron
  - GPI
  - Codeare

Nuts & Bolts of Advanced Imaging

The Image Reconstruction Pipeline

Michael S. Hansen, PhD

Magnetic Resonance Technology Program

National Institutes of Health, NHLBI

NIH National Heart, Lung, and Blood Institute

Declaration of Financial Interests or Relationships

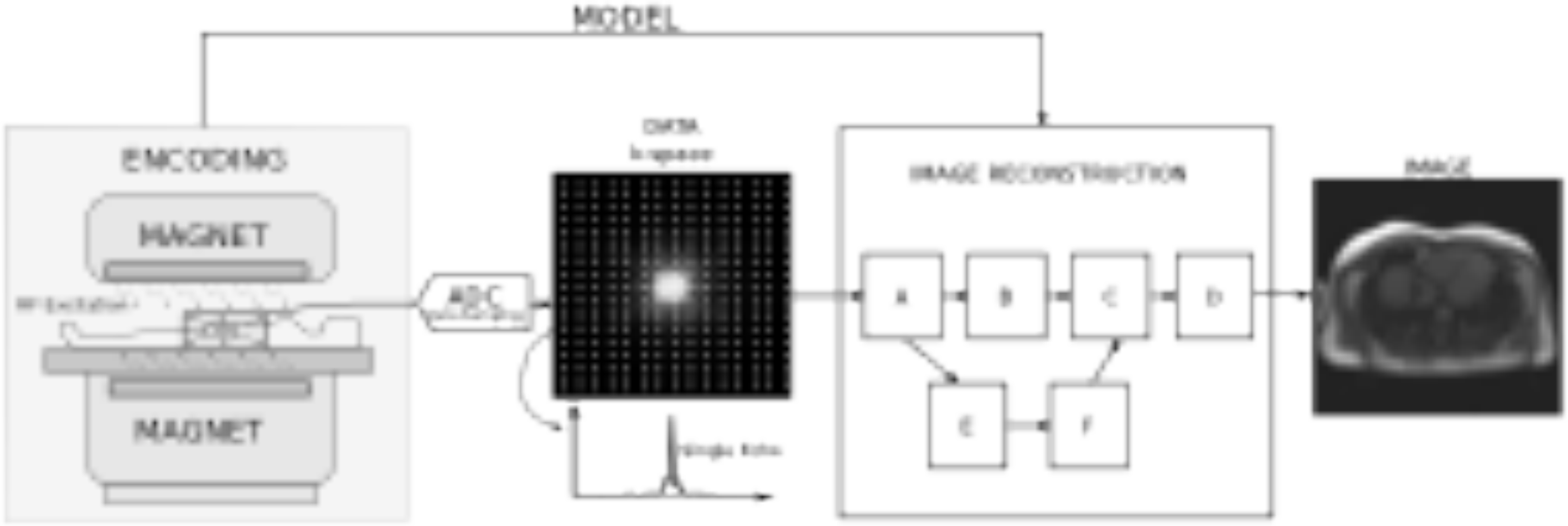
Speaker Name: Michael S. Hansen

I have the following financial interest or relationship to disclose with regard to the subject matter of this presentation:

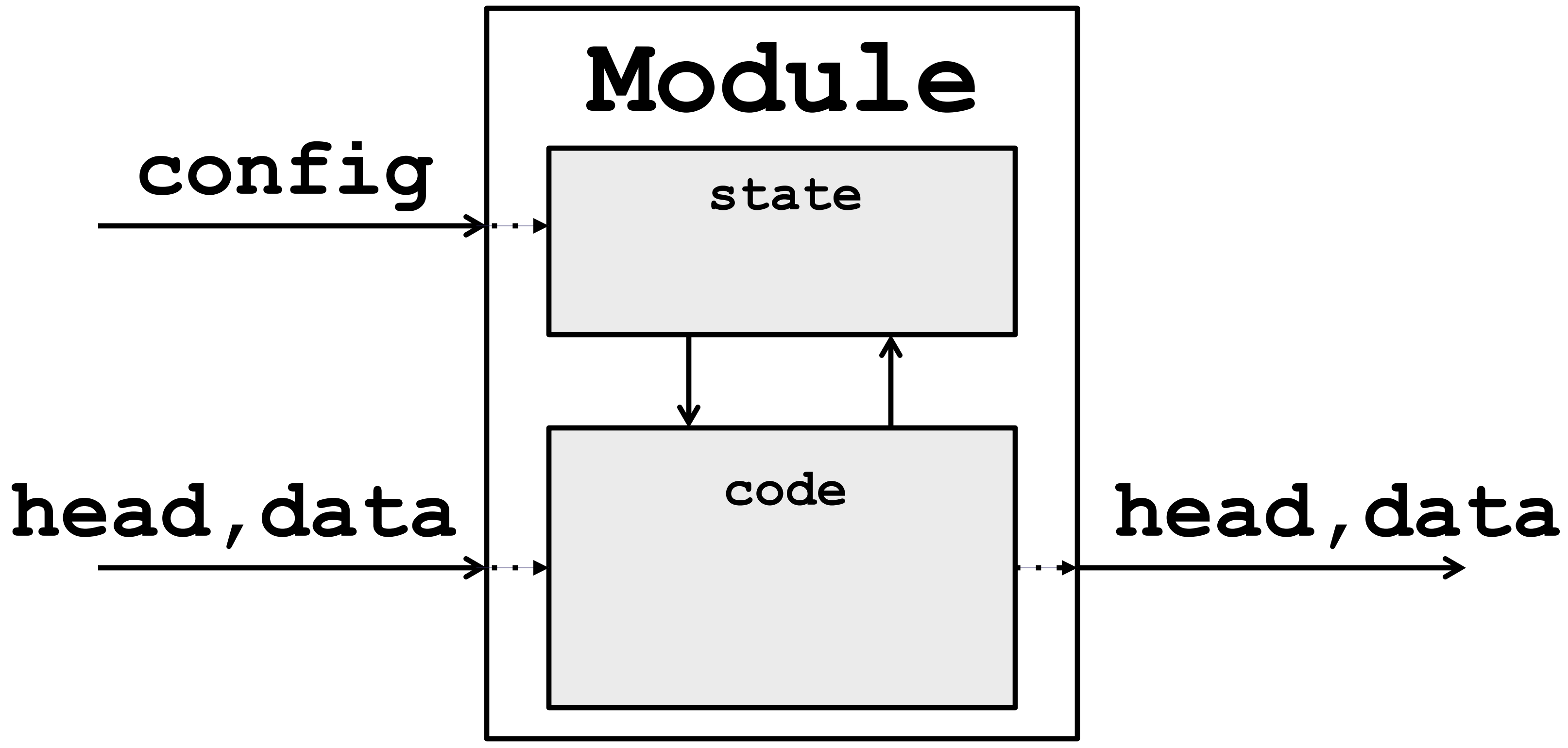
Company Name: Siemens Medical Solutions
Type of Relationship: Research Agreement

PART 2

The role of the image reconstruction process

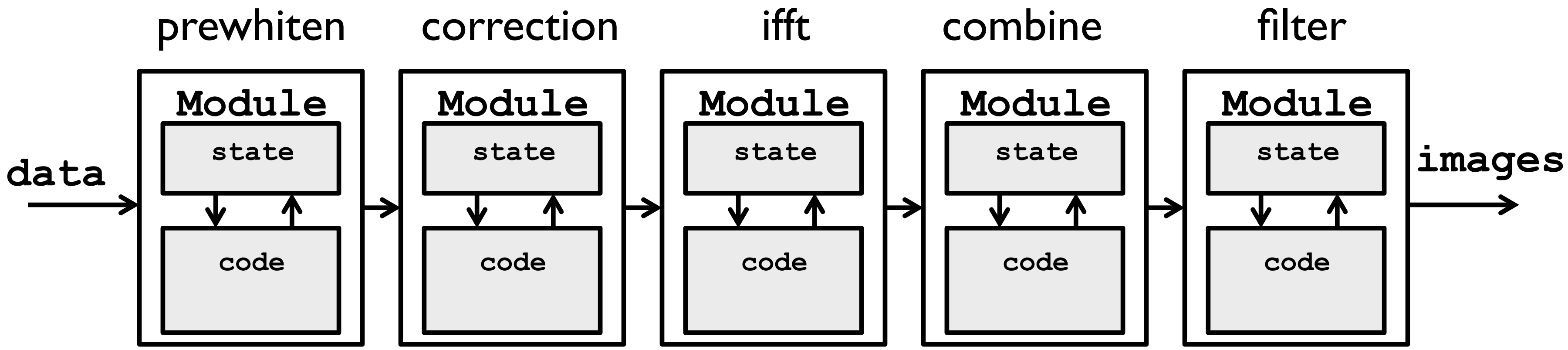


A modular pipeline

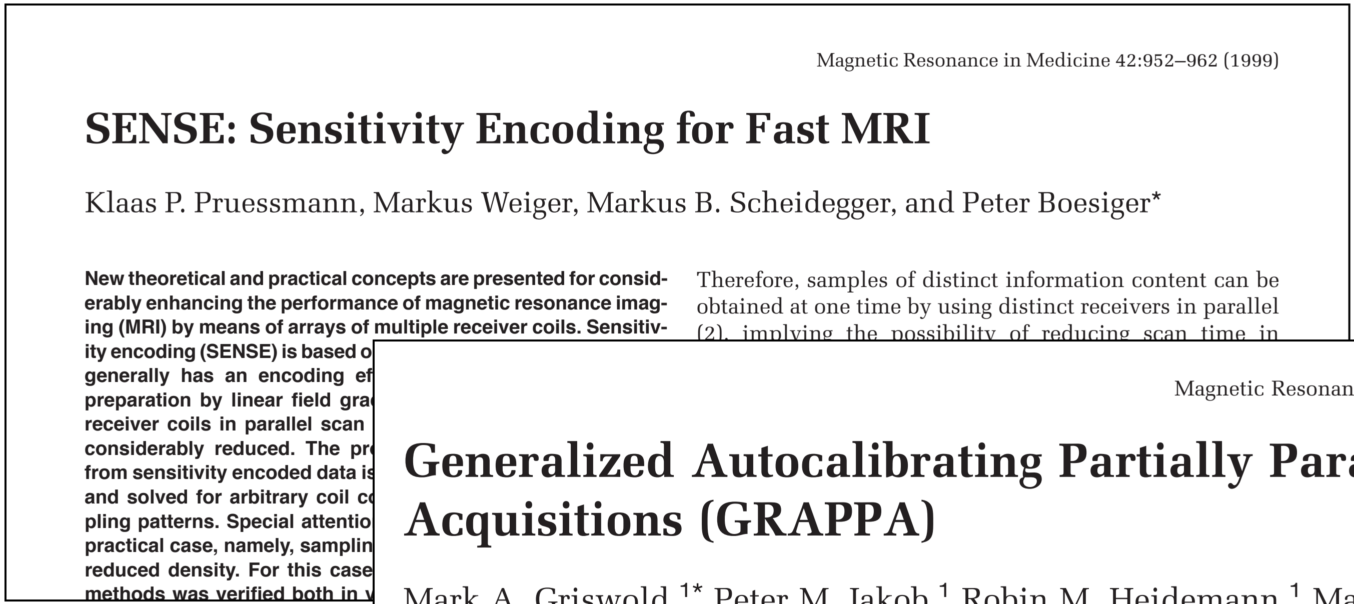


A modular pipeline

- Modular reconstruction design
- Well defined interfaces and data structures
- Processing can start when first readout is acquired
- Modules can be exchanged (without recompiling)
- Parallelization is inherent if modules run independently



Implement 2D Cartesian Parallel Imaging



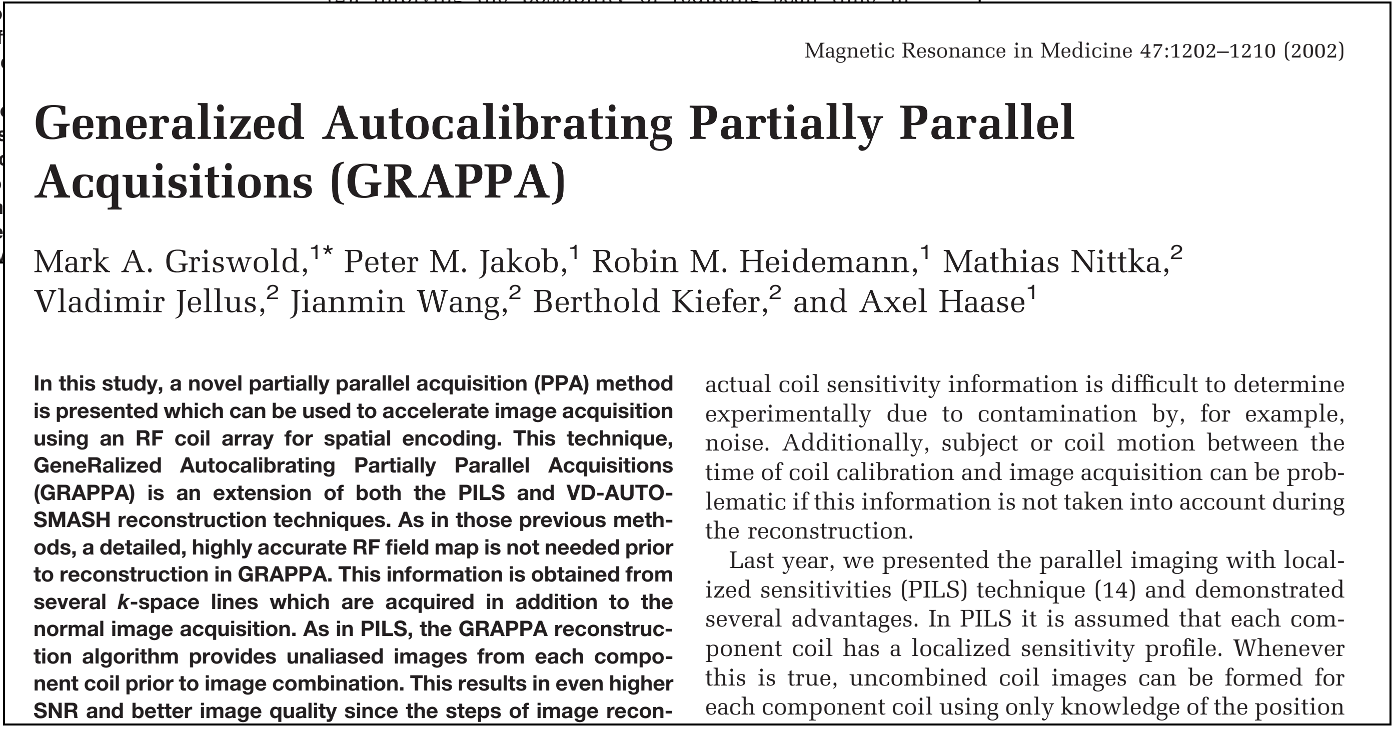
Magnetic Resonance in Medicine 42:952–962 (1999)

SENSE: Sensitivity Encoding for Fast MRI

Klaas P. Pruessmann, Markus Weiger, Markus B. Scheidegger, and Peter Boesiger*

New theoretical and practical concepts are presented for considerably enhancing the performance of magnetic resonance imaging (MRI) by means of arrays of multiple receiver coils. Sensitivity encoding (SENSE) is based o
generally has an encoding ef
preparation by linear field gra
receiver coils in parallel scan
considerably reduced. The pr
from sensitivity encoded data is
and solved for arbitrary coil c
pling patterns. Special attentio
practical case, namely, samplin
reduced density. For this case
methods was verified both in v

Therefore, samples of distinct information content can be obtained at one time by using distinct receivers in parallel (2), implying the possibility of reducing scan time in



Magnetic Resonance in Medicine 47:1202–1210 (2002)

Generalized Autocalibrating Partially Parallel Acquisitions (GRAPPA)

Mark A. Griswold,[1*] Peter M. Jakob,[1] Robin M. Heidemann,[1] Mathias Nittka,[2] Vladimir Jellus,[2] Jianmin Wang,[2] Berthold Kiefer,[2] and Axel Haase[1]

In this study, a novel partially parallel acquisition (PPA) method is presented which can be used to accelerate image acquisition using an RF coil array for spatial encoding. This technique, GeneRalized Autocalibrating Partially Parallel Acquisitions (GRAPPA) is an extension of both the PILS and VD-AUTO-SMASH reconstruction techniques. As in those previous methods, a detailed, highly accurate RF field map is not needed prior to reconstruction in GRAPPA. This information is obtained from several *k*-space lines which are acquired in addition to the normal image acquisition. As in PILS, the GRAPPA reconstruction algorithm provides unaliased images from each component coil prior to image combination. This results in even higher SNR and better image quality since the steps of image recon-

actual coil sensitivity information is difficult to determine experimentally due to contamination by, for example, noise. Additionally, subject or coil motion between the time of coil calibration and image acquisition can be problematic if this information is not taken into account during the reconstruction.

Last year, we presented the parallel imaging with localized sensitivities (PILS) technique (14) and demonstrated several advantages. In PILS it is assumed that each component coil has a localized sensitivity profile. Whenever this is true, uncombined coil images can be formed for each component coil using only knowledge of the position

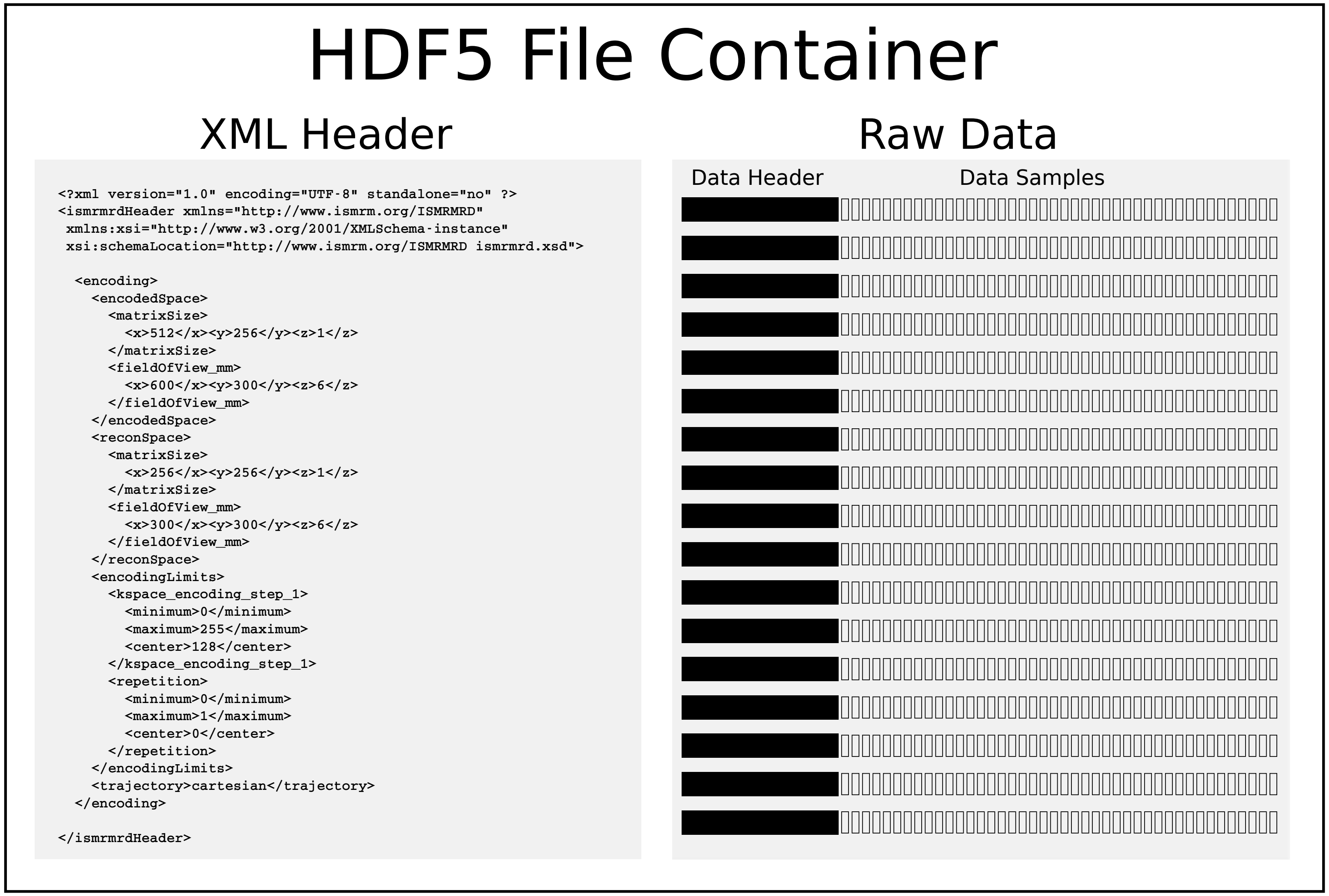
Cartesian Parallel Imaging - Ingredients

1. Data

2. Software

Details and code: http://hansenms.github.io/sunrise

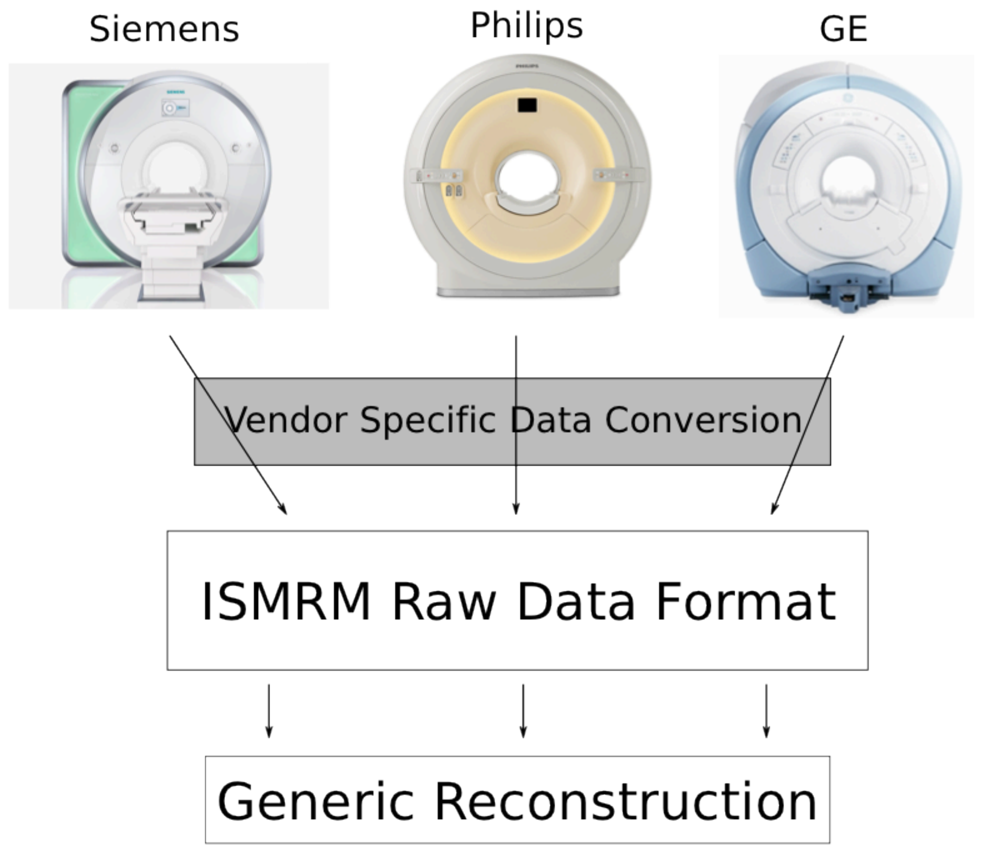
ISMRM Raw Data Format



```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<ismrmrdHeader xmlns="http://www.ismrm.org/ISMRMRD"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.ismrm.org/ISMRMRD ismrmrd.xsd">

  <encoding>
    <encodedSpace>
      <matrixSize>
        <x>512</x><y>256</y><z>1</z>
      </matrixSize>
      <fieldOfView_mm>
        <x>600</x><y>300</y><z>6</z>
      </fieldOfView_mm>
    </encodedSpace>
    <reconSpace>
      <matrixSize>
        <x>256</x><y>256</y><z>1</z>
      </matrixSize>
      <fieldOfView_mm>
        <x>300</x><y>300</y><z>6</z>
      </fieldOfView_mm>
    </reconSpace>
    <encodingLimits>
      <kspace_encoding_step_1>
        <minimum>0</minimum>
        <maximum>255</maximum>
        <center>128</center>
      </kspace_encoding_step_1>
      <repetition>
        <minimum>0</minimum>
        <maximum>1</maximum>
        <center>0</center>
      </repetition>
    </encodingLimits>
    <trajectory>cartesian</trajectory>
  </encoding>

</ismrmrdHeader>
```

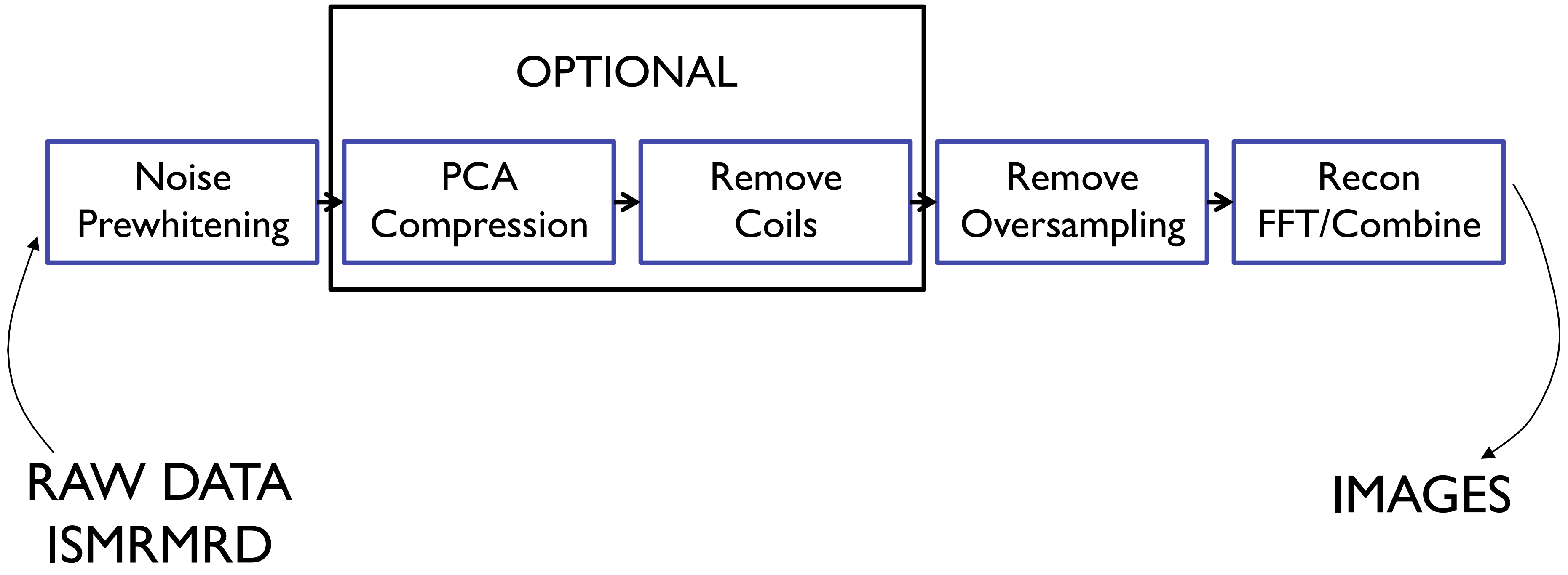
http://ismrmrd.github.io

ISMRM Raw Data Format

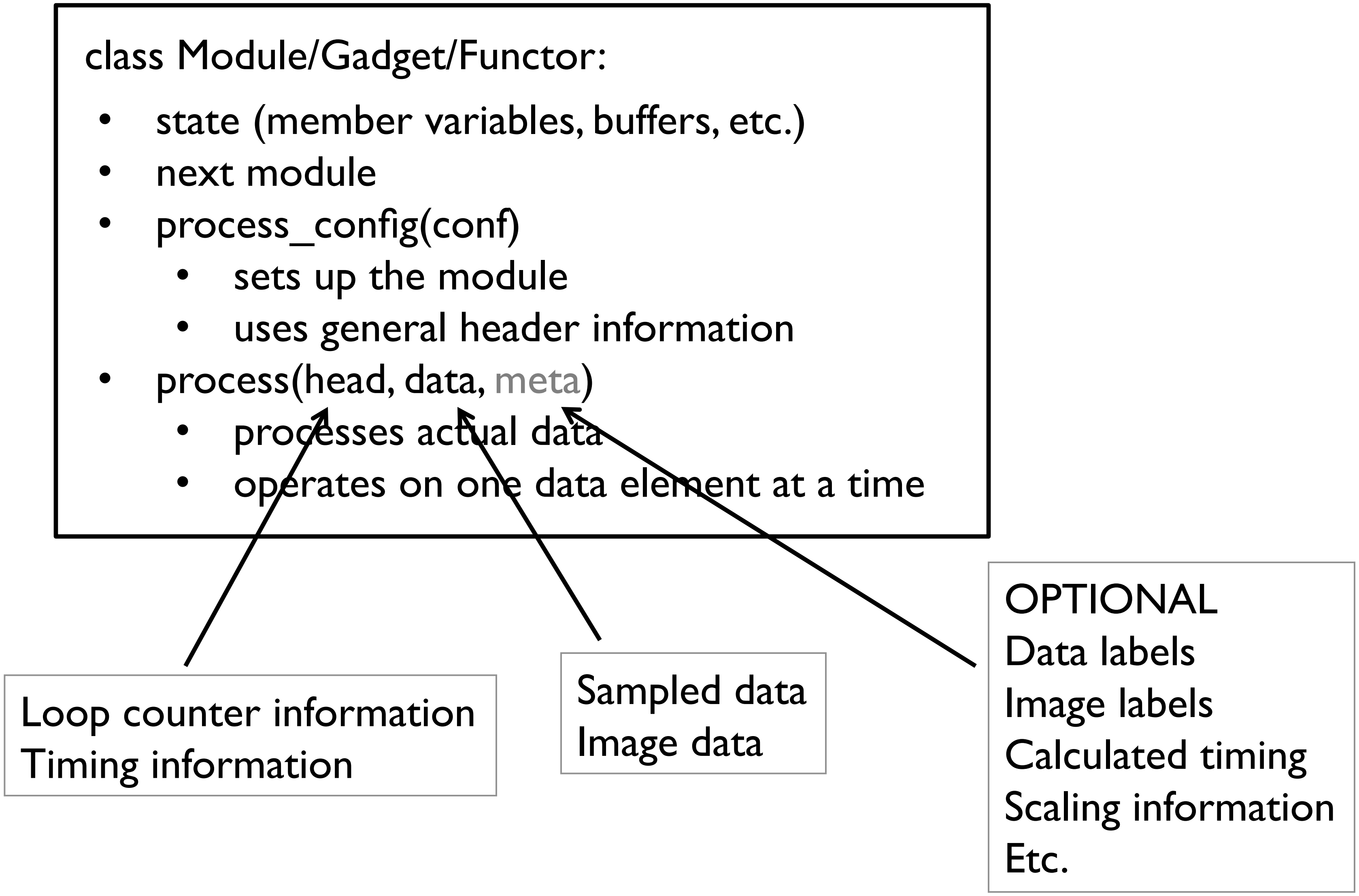


Info on converters: michael.hansen@nih.gov

Reconstruction Pipeline – TSENSE, TPAT, TGRAPPA



Pipeline Module (aka Gadget, Node, Functor, etc.)



class Module/Gadget/Functor:

- state (member variables, buffers, etc.)
- next module
- process_config(conf)
  - sets up the module
  - uses general header information
- process(head, data, meta)
  - processes actual data
  - operates on one data element at a time

Loop counter information
Timing information

Sampled data
Image data

OPTIONAL
Data labels
Image labels
Calculated timing
Scaling information
Etc.

Some example Python code

```
import ismrmrd
import ismrmrd.xsd
from gadgetron import gadget_chain_wait
from gadgetron import gadget_chain_config
from tpat_snr_scale import RemOS, NoiseAdj, PCA, CoilReduce, Recon
```

SET UP MODULES

```
def define_gadget_chain():
    g2 = Recon()
    g1 = RemOS(next_gadget=g2)
    g0 = CoilReduce(next_gadget=g1)
    gb = PCA(next_gadget=g0)
    ga = NoiseAdj(next_gadget=gb)
    return ga

g_python = define_gadget_chain()
```

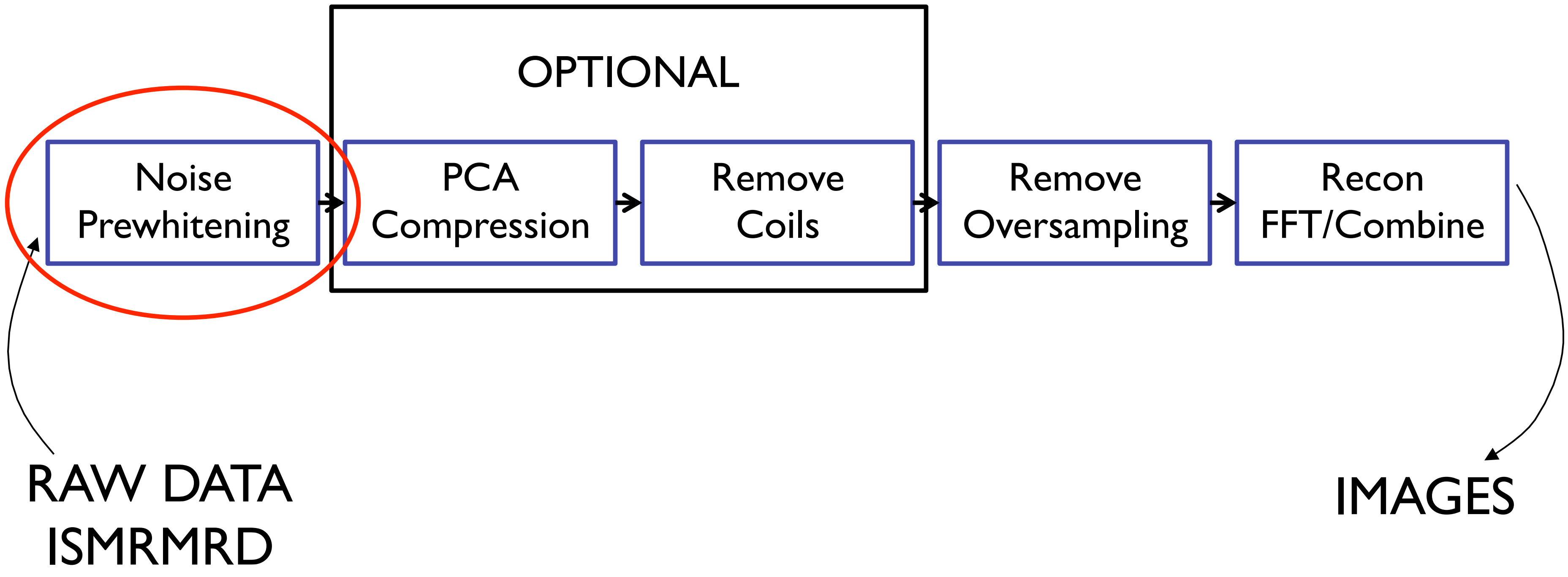
OPEN DATASET

```
dset = ismrmrd.Dataset(filename, 'dataset', create_if_needed=False)
```

RECONSTRUCT

```
# Send in data
#First ISMRMRD XML header
gadget_chain_config(g_python,dset.read_xml_header())

# Loop through the rest of the acquisitions and stuff
for acqnum in range(0,dset.number_of_acquisitions()):
    acq = dset.read_acquisition(acqnum)
    g_python.process(acq.getHead(),acq.data.astype('complex64'))

# Wait for recon to finish
gadget_chain_wait(g_python)
```

Reconstruction Pipeline – TSENSE, TPAT, TGRAPPA



Noise in Parallel Imaging

Idealized Experiment:

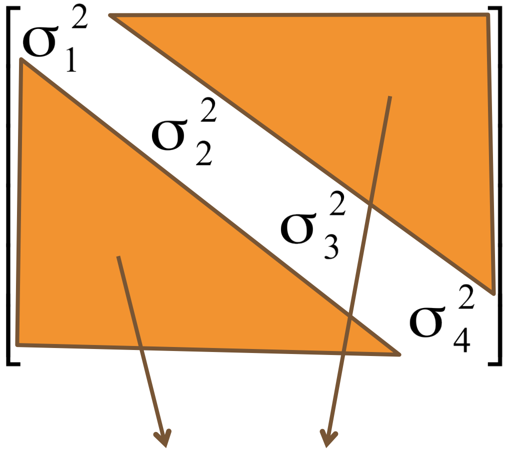
$$\mathbf{s} = \mathbf{E}\boldsymbol{\rho}$$

In practice, we are affected by noise

$$\mathbf{s} = \mathbf{E}\boldsymbol{\rho} + \boldsymbol{\eta}$$

Noise covariance matrix

$$\Psi_{\Upsilon,\Upsilon'} = \langle \eta_{\Upsilon}, \eta_{\Upsilon'} \rangle$$



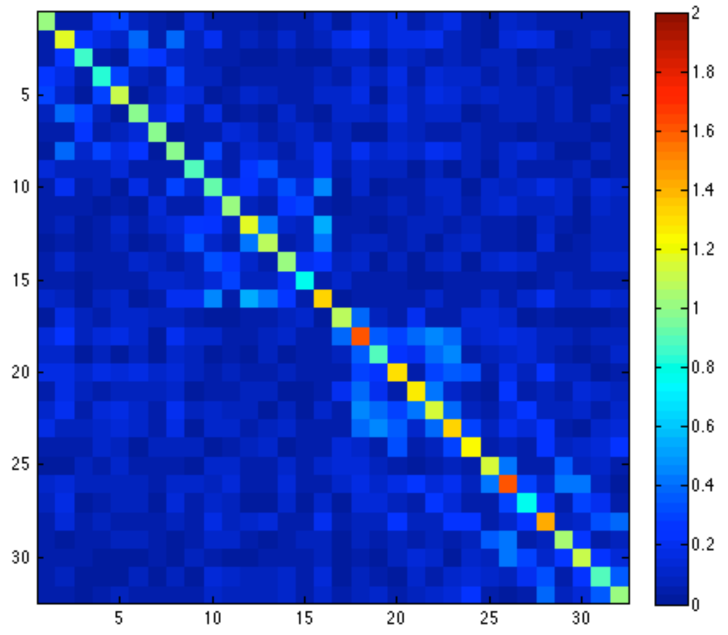
Noise correlation

We can measure this noise covariance:

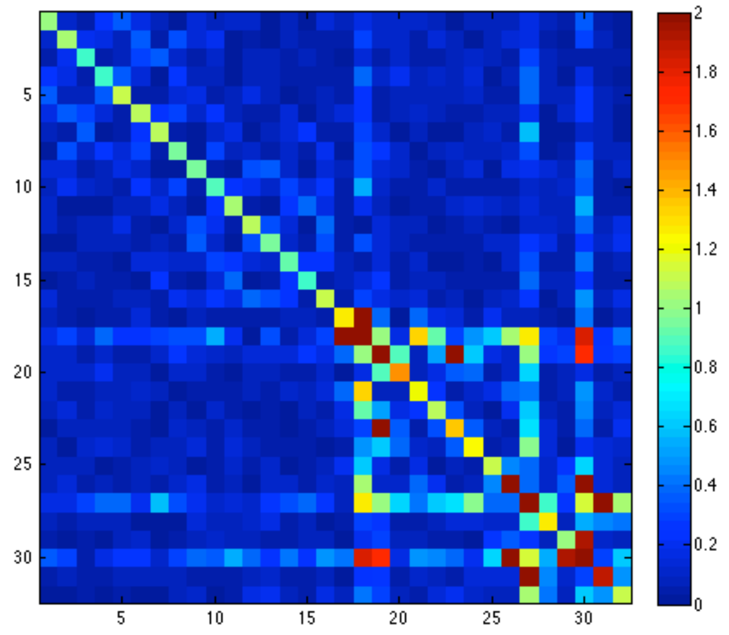
```
# Python
% eta: [Ncoils, Nsamples]
Psi = (1/(M-1))*np.asmatrix(eta)*np.asmatrix(eta).H
```

Psi Examples – 32 Channel Coil

“Normal Coil”



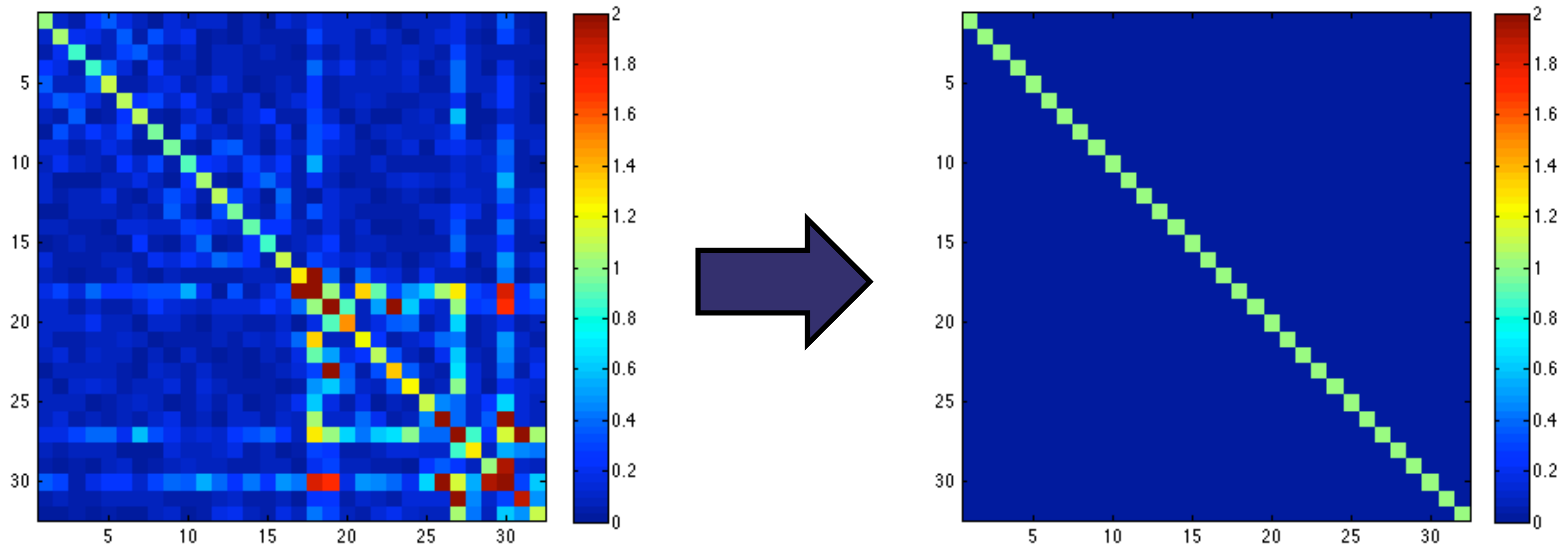
“Broken Coil”



Examination of the noise covariance matrix is an important QA tool. Reveals broken elements, faulty pre-amps, etc.

Noise Pre-Whitening

We would like to apply an operation such that we have unit variance in all channels:



Noise Pre-Whitening

More generally, we want to weight the equations with the "inverse square root" of the noise covariance, if

$$\mathbf{\Psi} = \mathbf{L}\mathbf{L}^{\mathrm{H}}$$

We will solve:

$$\mathbf{L}^{-1}\mathbf{A}\mathbf{x} = \mathbf{L}^{-1}\mathbf{b}$$

Or:

$$\mathbf{x} = \left(\mathbf{A}^{\mathrm{H}}\mathbf{\Psi}^{-1}\mathbf{A}\right)^{-1}\mathbf{A}^{\mathrm{H}}\mathbf{\Psi}^{-1}\mathbf{b}$$

In practice, we simply generate "pre-whitened" input data before recon

Noise Prewhitening - Python Code

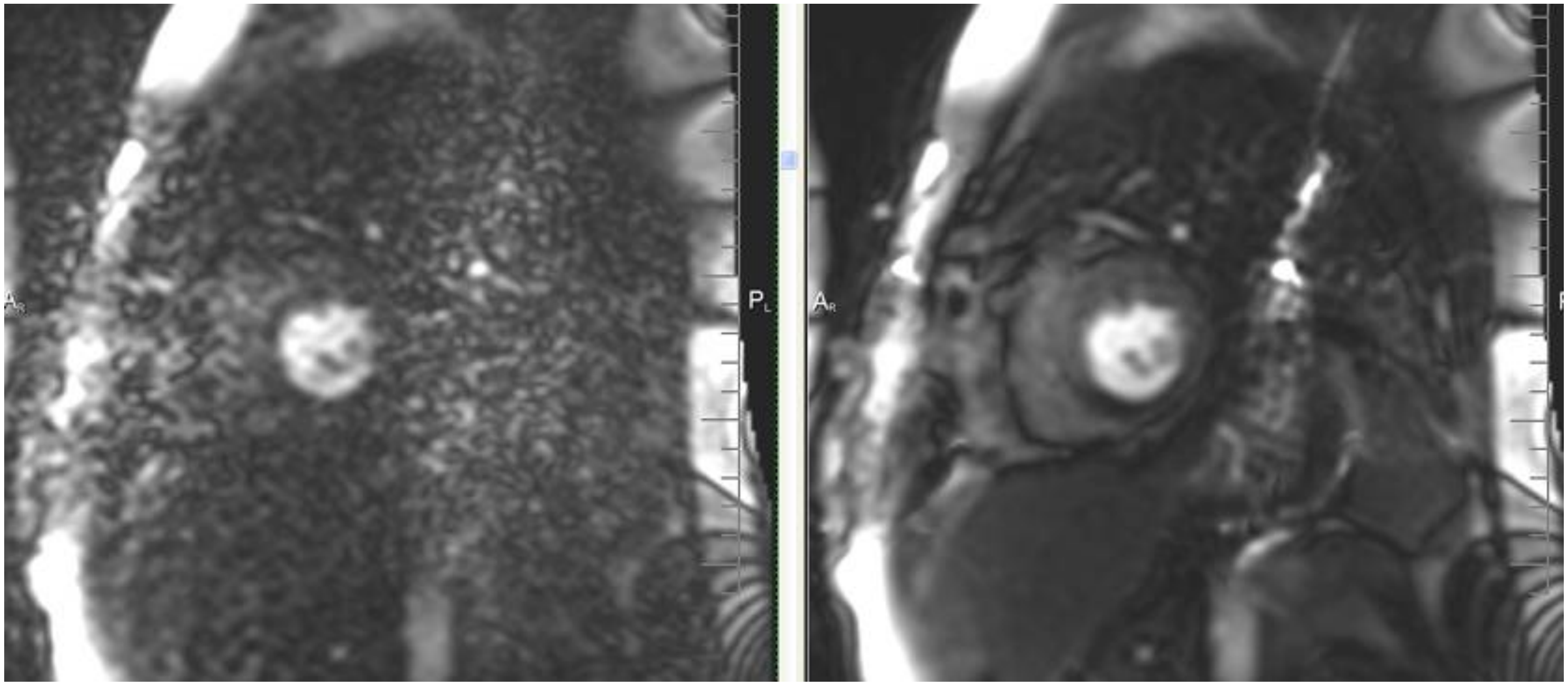
```python
import numpy as np

def calculate_prewhitening(noise, scale_factor=1.0):
    '''Calculates the noise prewhitening matrix

    :param noise: Input noise data (array or matrix), ``[coil, nsamples]``
    :scale_factor: Applied on the noise covariance matrix. Used to
                   adjust for effective noise bandwith and difference in
                   sampling rate between noise calibration and actual measurement:
                   scale_factor = (T_acq_dwell/T_noise_dwell)*NoiseReceiverBandwidthRatio

    :returns w: Prewhitening matrix, ``[coil, coil]``, w*data is prewhitened
    '''

    noise_int = noise.reshape((noise.shape[0],noise.size/noise.shape[0]))
    M = float(noise_int.shape[1])
    dmtx = (1/(M-1))*np.asmatrix(noise_int)*np.asmatrix(noise_int).H
    dmtx = np.linalg.inv(np.linalg.cholesky(dmtx));
    dmtx = dmtx*np.sqrt(2)*np.sqrt(scale_factor);
    return dmtx

def apply_prewhitening(data,dmtx):
    '''Apply the noise prewhitening matrix

    :param noise: Input noise data (array or matrix), ``[coil, ...]``
    :param dmtx: Input noise prewhitening matrix

    :returns w_data: Prewhitened data, ``[coil, ...]``,
    '''

    s = data.shape
    return np.asarray(np.asmatrix(dmtx)*np.asmatrix(data.reshape(data.shape[0],data.size/data.shape[0]))).reshape(s)
```

Noise Pre-Whitening – In vivo example

In vivo stress perfusion case where broken coil element resulted in non-diagnostic images.

Without pre-whitening

With pre-whitening



Example provided by Peter Kellman, NIH

Noise Adjust Module

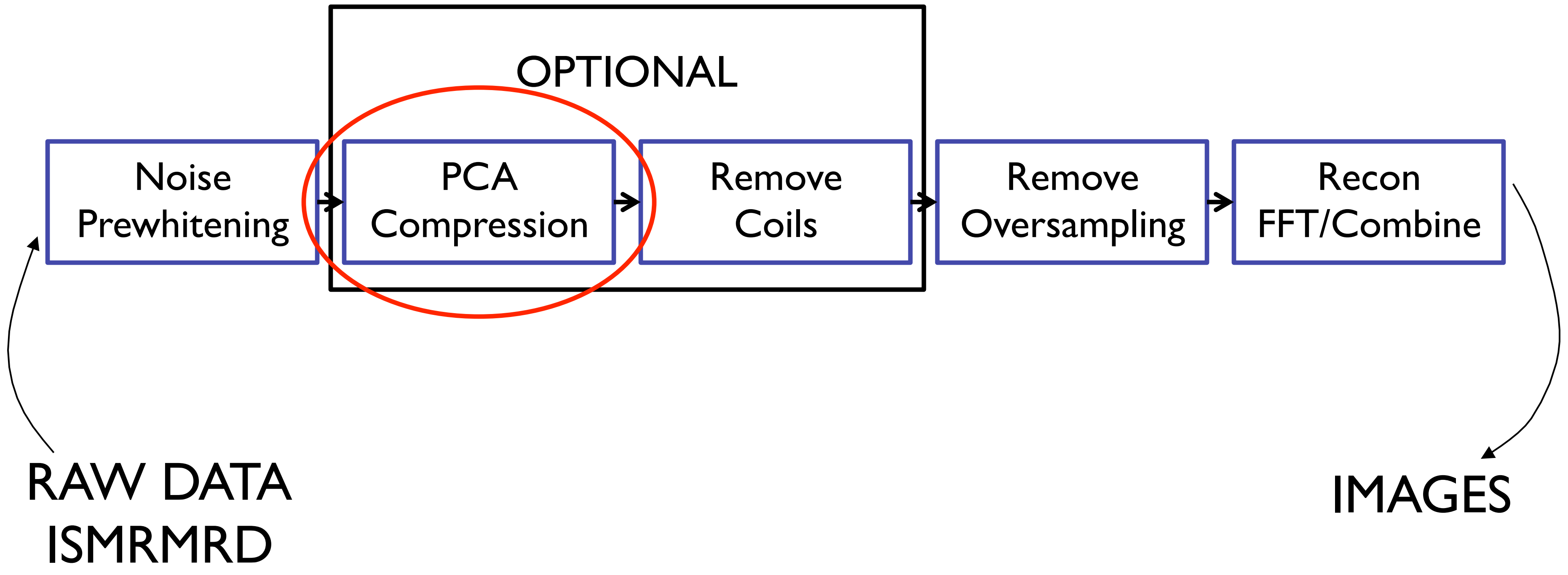
```
class NoiseAdj(Gadget):
    def __init__(self, next_gadget = None):
        Gadget.__init__(self, next_gadget)
        self.noise_data = list()
        self.noise_dmtx = None
    def process(self,acq,data,*args):
        if acq.isFlagSet(ismrmrd.ACQ_IS_NOISE_MEASUREMENT):
            self.noise_data.append((acq,data))
        else:
            if len(self.noise_data):
                profiles = len(self.noise_data)
                channels = self.noise_data[0][1].shape[0]
                samples_per_profile = self.noise_data[0][1].shape[1]
                noise = np.zeros((channels,profiles*samples_per_profile),dtype=np.complex64)
                counter = 0
                for p in self.noise_data:
                    noise[:,counter*samples_per_profile:(counter*samples_per_profile+samples_per_profile)] = p[1]
                    counter = counter + 1

                scale = (acq.sample_time_us/self.noise_data[0][0].sample_time_us)*0.79
                self.noise_dmtx = coils.calculate_prewhitening(noise,scale_factor=scale)

                #Test the noise adjust
                d = self.noise_data[0][1]
                d2 = coils.apply_prewhitening(d, self.noise_dmtx)
                self.noise_data = list()

            if self.noise_dmtx is not None:
                data2 = coils.apply_prewhitening(data, self.noise_dmtx)
            else:
                data2 = data

            self.put_next(acq,data2)
        return 0
```

BUFFERING

CALCULATE PREWHITENER

APPLY

Reconstruction Pipeline – TSENSE, TPAT, TGRAPPA



PCA Module

```
class PCA(Gadget):
    def __init__(self, next_gadget=None):
        Gadget.__init__(self, next_gadget)
        self.calib_data = list()
        self.pca_mtx = None
        self.max_calib_profiles = 100
        self.samples_to_use = 16
        self.buffering = True

    def process(self,acq,data,*args):
        if self.buffering:
            self.calib_data.append((acq,data))
```

BUFFERING

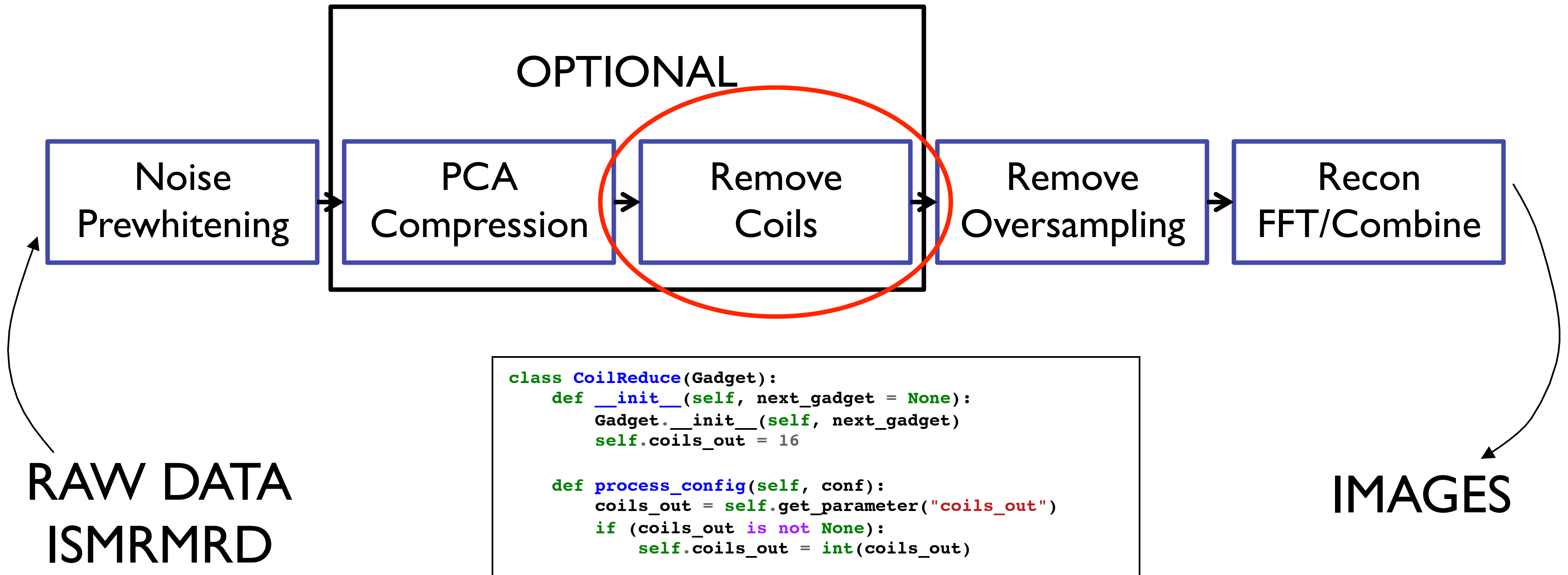
```
            if (len(self.calib_data)>=self.max_calib_profiles or acq.isFlagSet(ismrmrd.ACQ_LAST_IN_SLICE)):
                #We are done buffering calculate pca transformation
                # ...book keeping code (removed to save space, refer to original code)

                A = np.zeros((total_samples,channels), dtype=np.complex64)
                counter = 0
                for p in self.calib_data:
                    d = p[1][:, acq.center_sample-(samp_to_use>>1):acq.center_sample+(samp_to_use>>1)]
                    A[counter*samp_to_use:counter*samp_to_use+samp_to_use,:] = np.transpose(d)
                    counter = counter+1

                m = np.mean(A,0)
                A_m = A - m.reshape((1,m.shape[0]))
                U, s, V = np.linalg.svd(A_m, full_matrices=False)
                self.pca_mtx = V

                for p in self.calib_data:
                    data2 = np.dot(self.pca_mtx,p[1])
                    self.put_next(p[0],data2)

                self.buffering = False
                self.calib_data = list()
                return 0
```

CALCULATE PCA COEFFICIENTS

```
        else:
            if self.pca_mtx is not None:
                data2 = np.dot(self.pca_mtx,data)
                self.put_next(acq,data2,*args)
            else:
                self.put_next(acq,data,*args)

        return 0
```

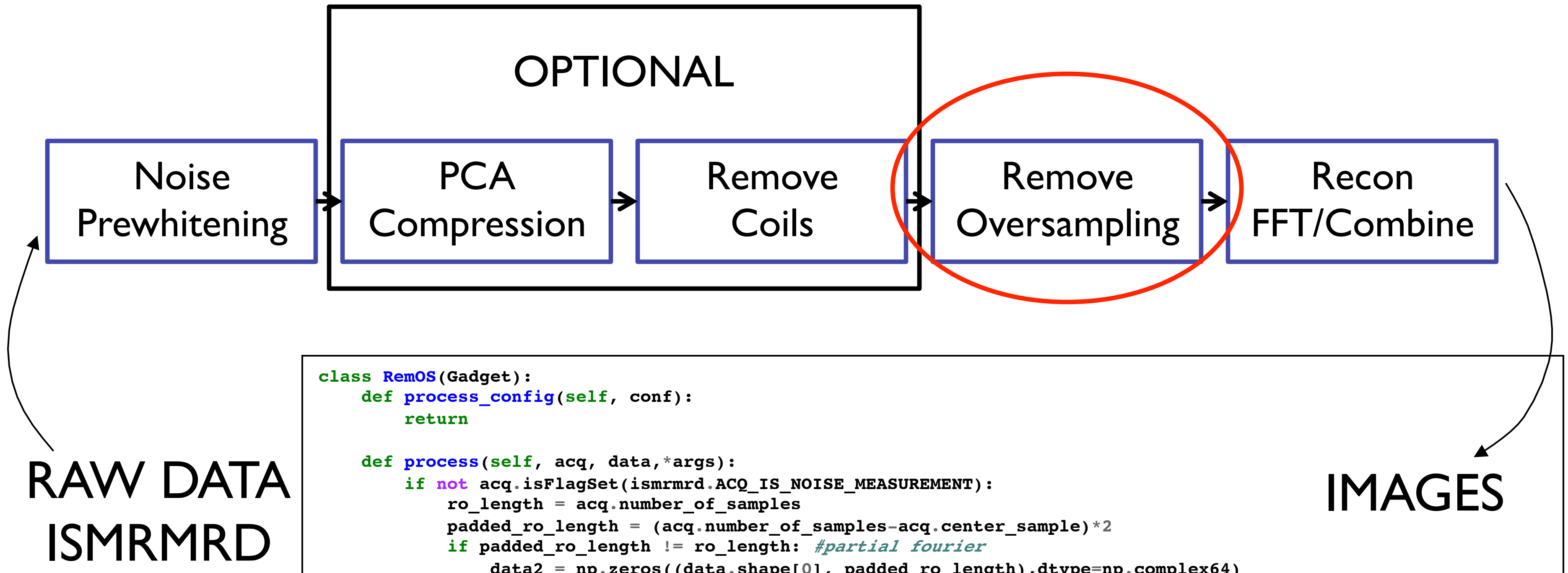
APPLY

Reconstruction Pipeline – TSENSE, TPAT, TGRAPPA



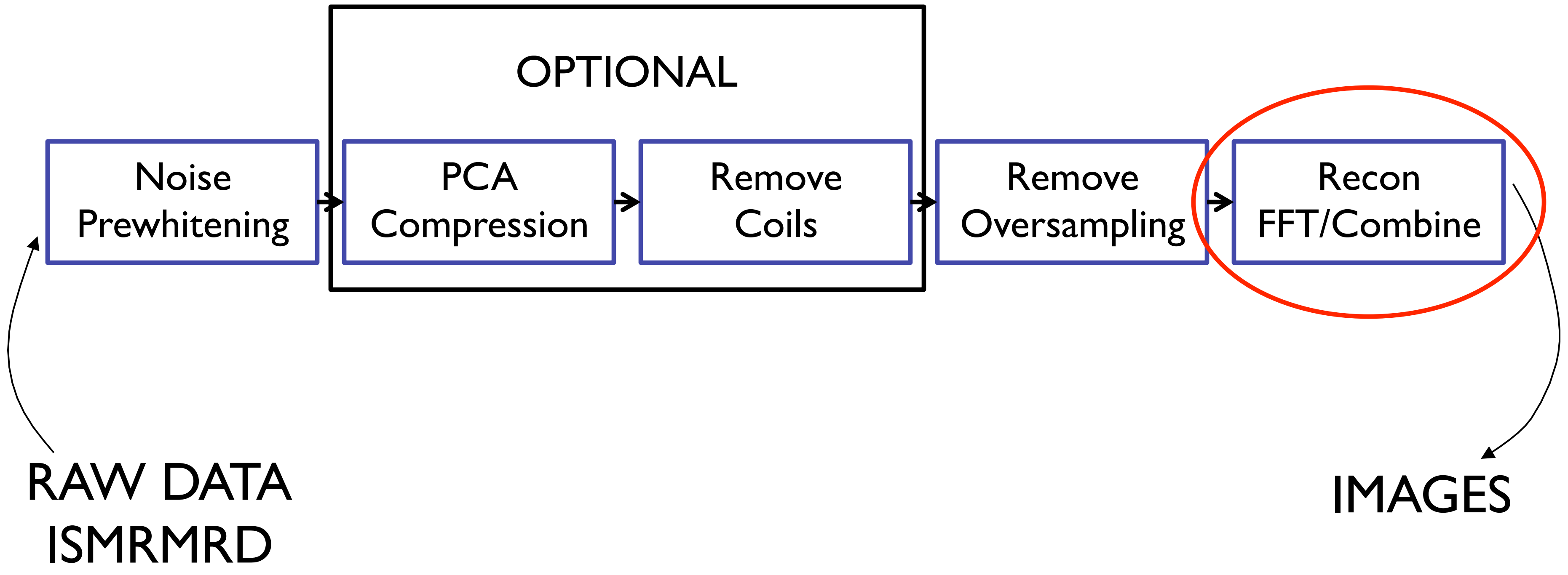
```
class CoilReduce(Gadget):
    def __init__(self, next_gadget = None):
        Gadget.__init__(self, next_gadget)
        self.coils_out = 16

    def process_config(self, conf):
        coils_out = self.get_parameter("coils_out")
        if (coils_out is not None):
            self.coils_out = int(coils_out)

    def process(self, acq, data, *args):
        if acq.active_channels > self.coils_out:
            data2 = data[0:self.coils_out,:]
            acq.active_channels = self.coils_out
        else:
            data2 = data

        self.put_next(acq,data2,*args)
        return 0
```

Reconstruction Pipeline – TSENSE, TPAT, TGRAPPA

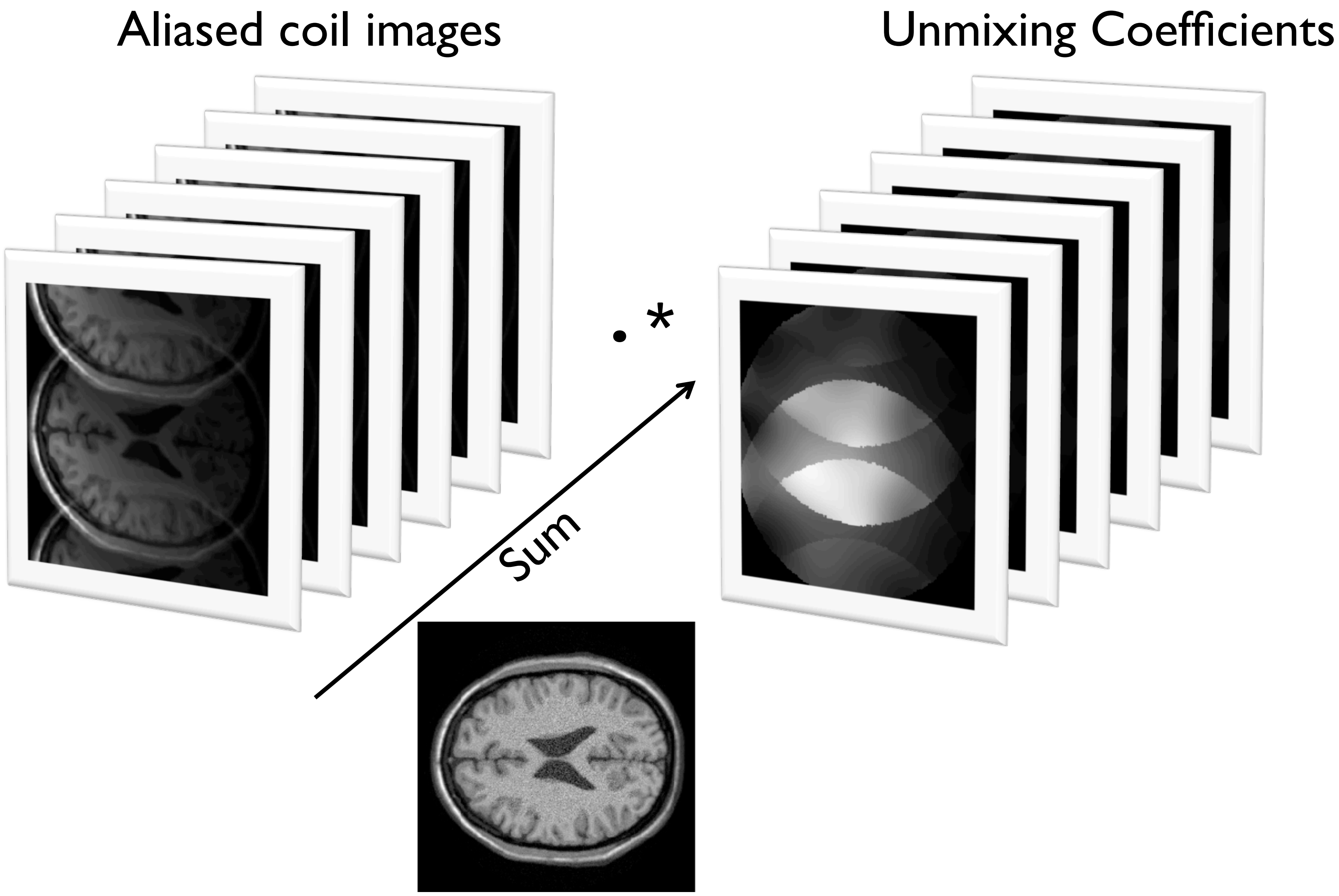


```
class RemOS(Gadget):
    def process_config(self, conf):
        return

    def process(self, acq, data,*args):
        if not acq.isFlagSet(ismrmrd.ACQ_IS_NOISE_MEASUREMENT):
            ro_length = acq.number_of_samples
            padded_ro_length = (acq.number_of_samples-acq.center_sample)*2
            if padded_ro_length != ro_length: #partial fourier
                data2 = np.zeros((data.shape[0], padded_ro_length),dtype=np.complex64)
                offset = (padded_ro_length>>1)  - acq.center_sample
                data2[:,0+offset:offset+ro_length] = data
            else:
                data2 = data

            data2=transform.transform_kspace_to_image(data2,dim=(1,))
            data2=data2[:,(padded_ro_length>>2):(padded_ro_length>>2)+(padded_ro_length>>1)]
            data2=transform.transform_image_to_kspace(data2,dim=(1,)) * np.sqrt(float(padded_ro_length)/ro_length)
            acq.center_sample = padded_ro_length>>2
            acq.number_of_samples = data2.shape[1]
            self.put_next(acq,data2,*args)
        return 0
```

Reconstruction Pipeline – TSENSE, TPAT, TGRAPPA



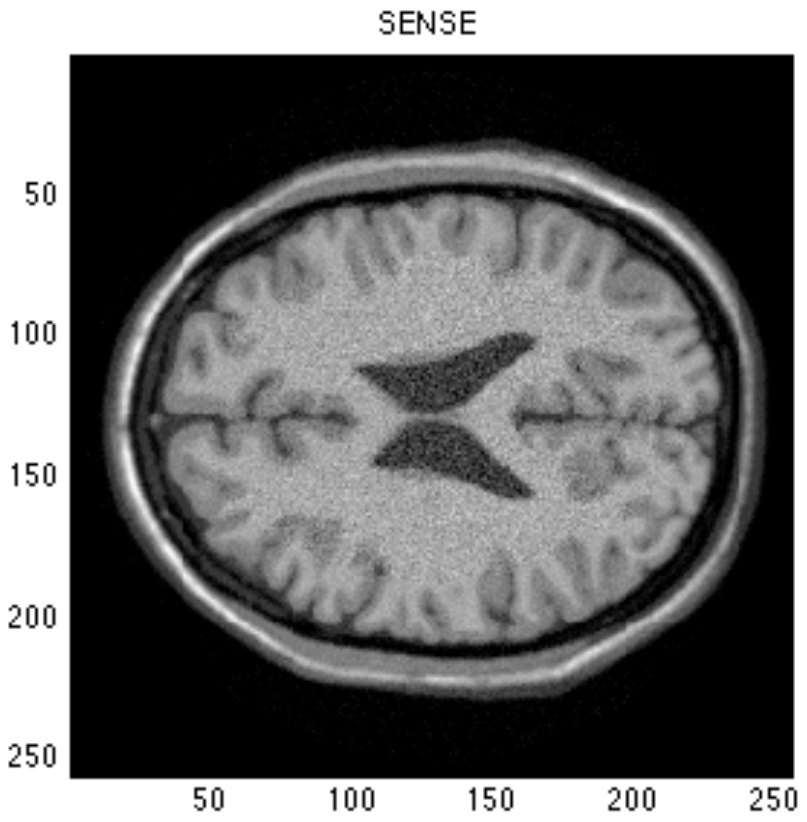
SENSE – Image Synthesis with Unmixing Coefficients

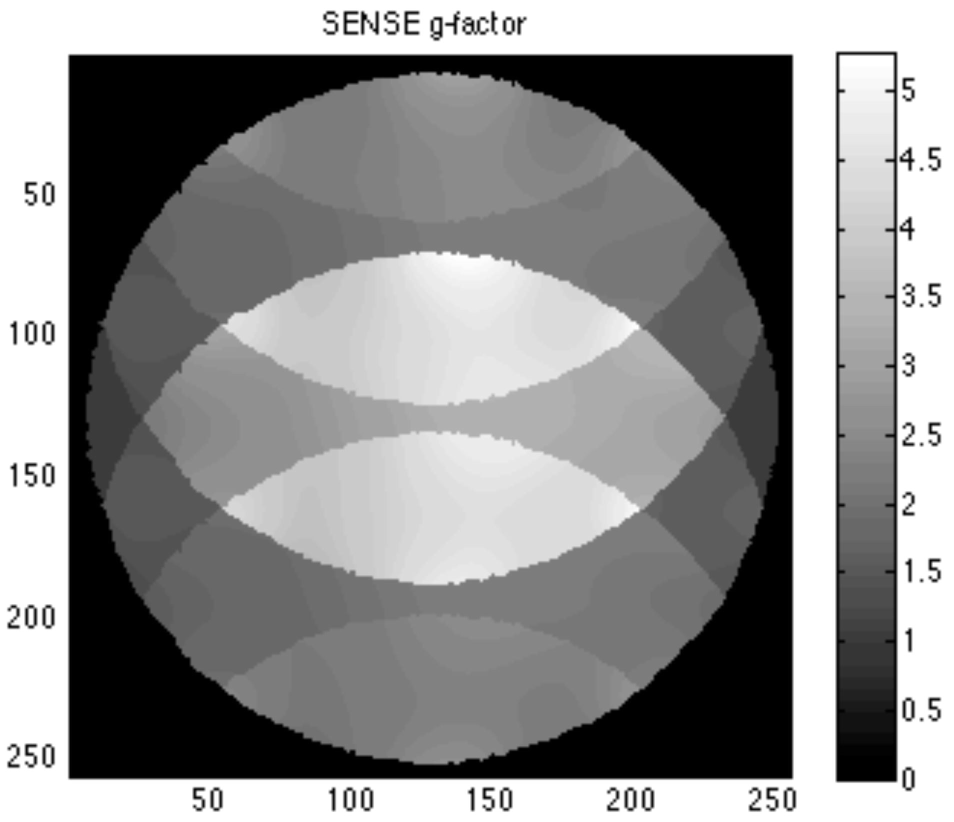


SENSE – Simple Rate 4 Example

$$\tilde{\rho}(x_1) = \sum_{i=0}^{N_c} u_i a_i$$

$$g(x_1) = \sqrt{\sum_{i=0}^{N_c} |u_i|^2} \sqrt{\sum_{i=0}^{N_c} |S_i|^2}$$





Recon Module

```
class Recon(Gadget):
    def __init__(self, next_gadget=None):
        Gadget.__init__(self, next_gadget)
        self.header = None
        self.enc = None
        self.acc_factor = None
        self.buffer = None
        self.samp_mask = None
        self.header_proto = None
        self.calib_buffer = list()
        self.unmix = None
        self.gmap = None
        self.calib_frames = 0
        self.method = 'grappa'

    def process_config(self, cfg):
        self.header = ismrmrd.xsd.CreateFromDocument(cfg)
        self.enc = self.header.encoding[0]

        #Parallel imaging factor
        self.acc_factor = self.enc.parallelImaging.accelerationFactor.kspace_encoding_step_1

        reps = self.enc.encodingLimits.repetition.maximum+1
        phs = self.enc.encodingLimits.phase.maximum+1
        if reps > phs:
            self.calib_frames = reps
        else:
            self.calib_frames = phs

        if self.calib_frames < self.acc_factor:
            self.calib_frames = self.acc_factor

        #Frames should be a multiple of the acceleration factor
        self.frames = math.floor(self.calib_frames/self.acc_factor)*self.acc_factor

        pmri_method =  self.get_parameter('pmri_method')
        if pmri_method == 'grappa' or pmri_method == 'sense':
            self.method = pmri_method

    def process(self, acq, data,*args):
        #....
```

```python
def process_config(self, cfg):
  #...

def process(self, acq, data,*args):

  if self.buffer is None:
    # Matrix size
    # ...initialize bufffer (code removed, see original source)

  #Now put data in buffer
  line_offset = self.buffer.shape[1]/2 - self.enc.encodingLimits.kspace_encoding_step_1.
  self.buffer[:,acq.idx.kspace_encode_step_1+line_offset,:] = data
  self.samp_mask[acq.idx.kspace_encode_step_1+line_offset,:] = 1
```

BUFFERING

```python
  #If last scan in buffer, do FFT and fill image header
  if acq.isFlagSet(ismrmrd.ACQ_LAST_IN_ENCODE_STEP1) or acq.isFlagSet(ismrmrd.ACQ_LAST_IN_SLICE):
     #... Set up image header (code removed, see original source)

    #We have not yet calculated unmixing coefficients
    if self.unmix is None:
      self.calib_buffer.append((img_head,self.buffer.copy()))
      self.buffer[:] = 0
```

BUFFERING

```python
      self.samp_mask[:] = 0

      if len(self.calib_buffer) >= self.calib_frames:
        coil_images = transform.transform_kspace_to_image(cal_data,dim=(1,2))
        (csm,rho) = coils.calculate_csm_walsh(coil_images)

        if self.method == 'grappa':
          self.unmix, self.gmap = grappa.calculate_grappa_unmixing(cal_data,
                                                                   self.acc_factor,
                                                                   kernel_size=(4,5),
                                                                   csm=csm)
        elif self.method == 'sense':
          self.unmix, self.gmap = sense.calculate_sense_unmixing(self.acc_factor, csm)
        else:
          raise Exception('Unknown parallel imaging method: ' + str(self.method))
```

CALIBRATION

```python
        for c in self.calib_buffer:
          recon = transform.transform_kspace_to_image(c[1],dim=(1,2))*np.sqrt(scale)
          recon = np.squeeze(np.sum(recon * self.unmix,0))
          self.put_next(c[0], recon,*args)

      return 0

    if self.unmix is None:
      raise Exception("We should never reach this point without unmixing coefficients")

    recon = transform.transform_kspace_to_image(self.buffer,dim=(1,2))*np.sqrt(scale)
    recon = np.squeeze(np.sum(recon * self.unmix,0))
    self.buffer[:] = 0
```

RECON

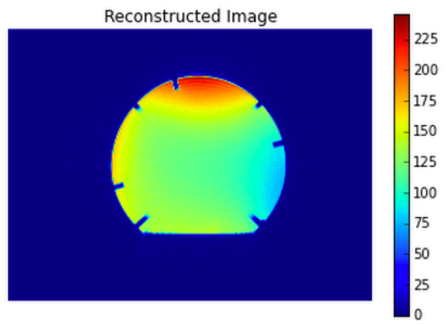
```python
    self.samp_mask[:] = 0
    self.put_next(img_head,recon,*args)
  return 0
```

Running the Python reconstruction

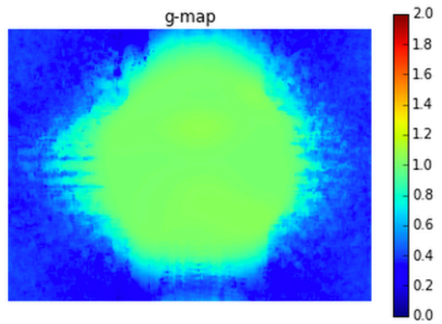
```
# Send in data
#First ISMRMRD XML header
gadget_chain_config(g_python,dset.read_xml_header())

# Loop through the rest of the acquisitions and stuff
for acqnum in range(0,dset.number_of_acquisitions()):
    acq = dset.read_acquisition(acqnum)
    g_python.process(acq.getHead(),acq.data.astype('complex64'))

# Wait for recon to finish
gadget_chain_wait(g_python)
```

```
res_python = get_last_gadget(g_python).get_results()
gmap = get_last_gadget(g_python).gmap
```

```
show.imshow(abs(np.squeeze(res_python[0][1])),colorbar=True,titles=['Reconstructed Image'])
```



```
show.imshow(abs(np.squeeze(gmap)),colorbar=True,scale=(0,2.0),titles=['g-map'])
```



Open Source Reconstruction Frameworks

COMPUTER PROCESSING AND MODELING - Full Papers

Magnetic Resonance in Medicine 69:1768–1776 (2013)

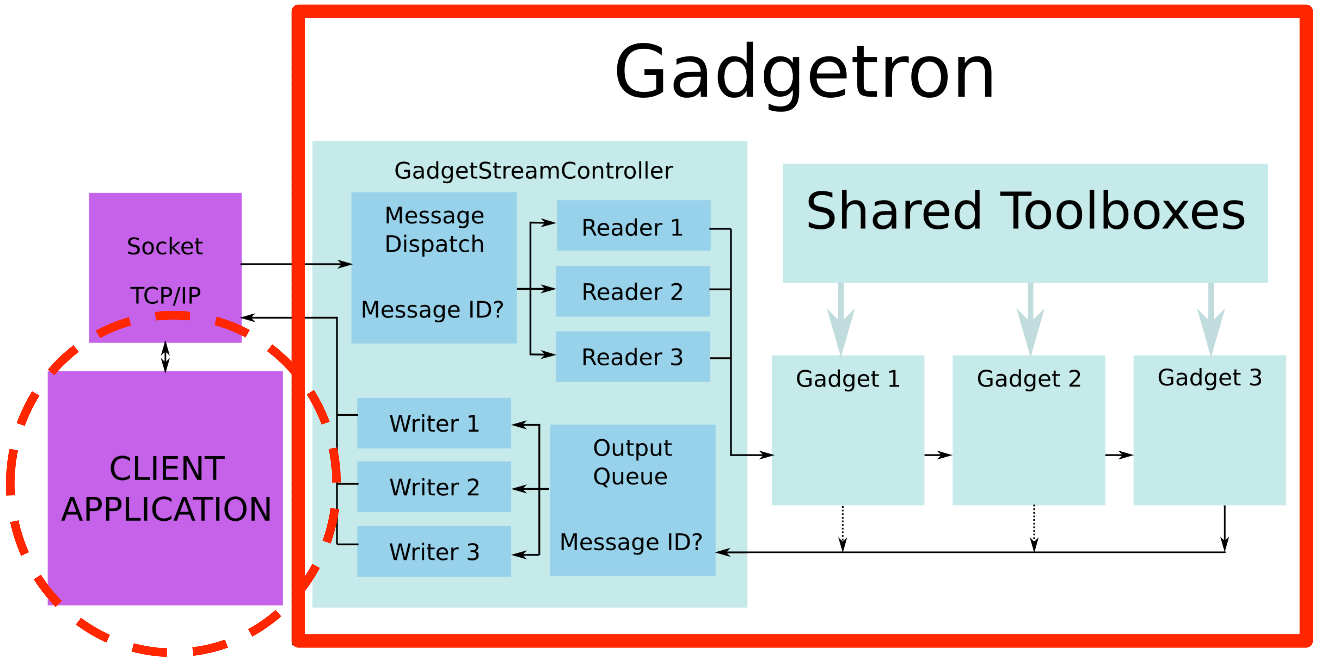
Gadgetron: An Open Source Framework for Medical Image Reconstruction

Michael Schacht Hansen[1]* and Thomas Sangild Sørensen[2,3]

This work presents a new open source framework for medical image reconstruction called the "Gadgetron." The framework implements a flexible system for creating streaming data processing pipelines where data pass through a series of modules or "Gadgets" from raw data to reconstructed images. The data processing pipeline is configured dynamically at run-time based on an extensible markup language configuration description. The framework promotes reuse and sharing of reconstruction modules and new Gadgets can be added to the Gadgetron framework

way of sharing the algorithms; they may rely on a great deal of accessory code, some of which could be vendor specific or even contain vendor-provided code that cannot be shared. Regardless of the reasons, it undermines the scientific process that readers and reviewers are prevented from reproducing the results of reconstruction research articles. It is exceedingly difficult for other researchers to evaluate how a given algorithm might perform given a different type of data or how it might interact with other algorithms. As a

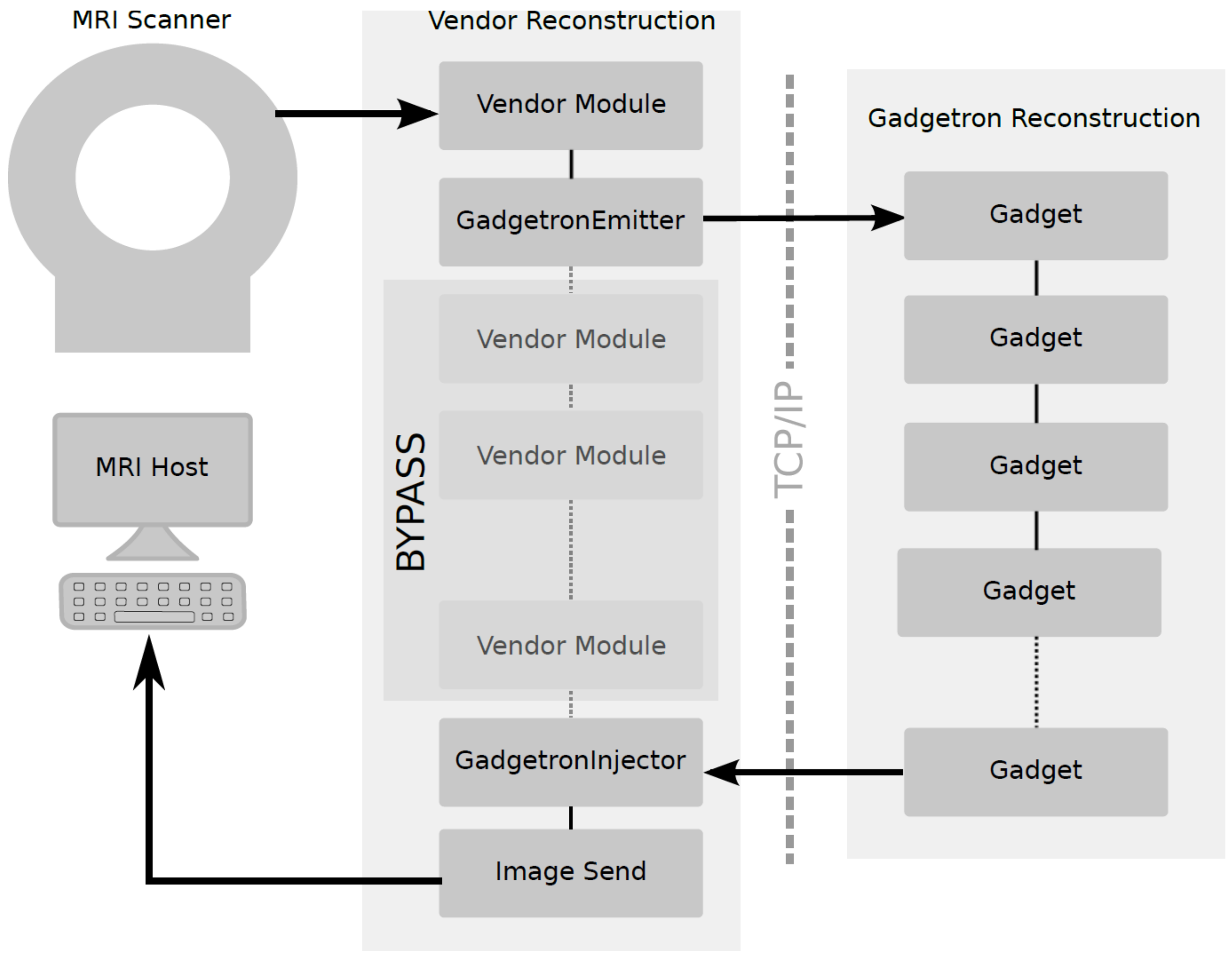
Gadgetron Architecture



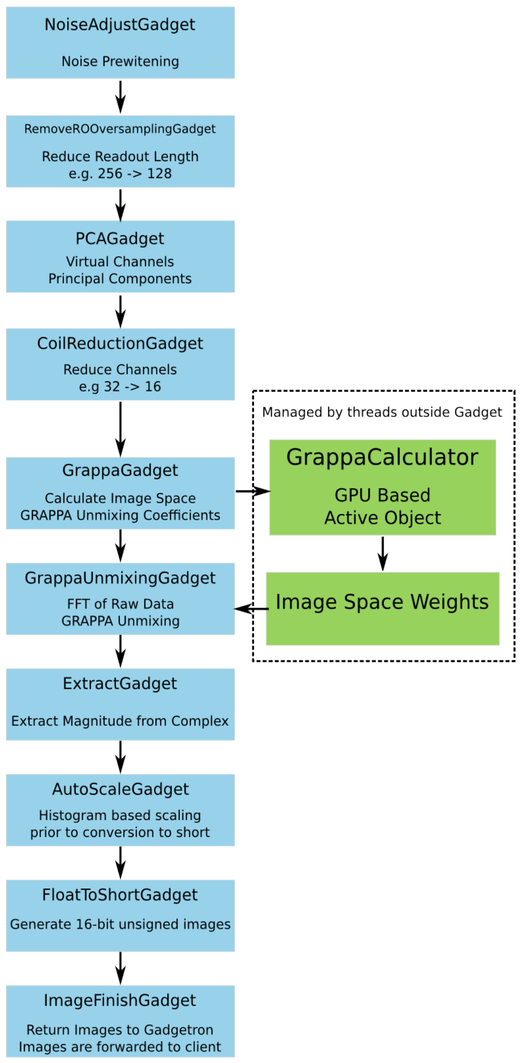
Configured dynamically at run time
XML Configuration

Scanner Reconstruction

Generic Scanner Integration

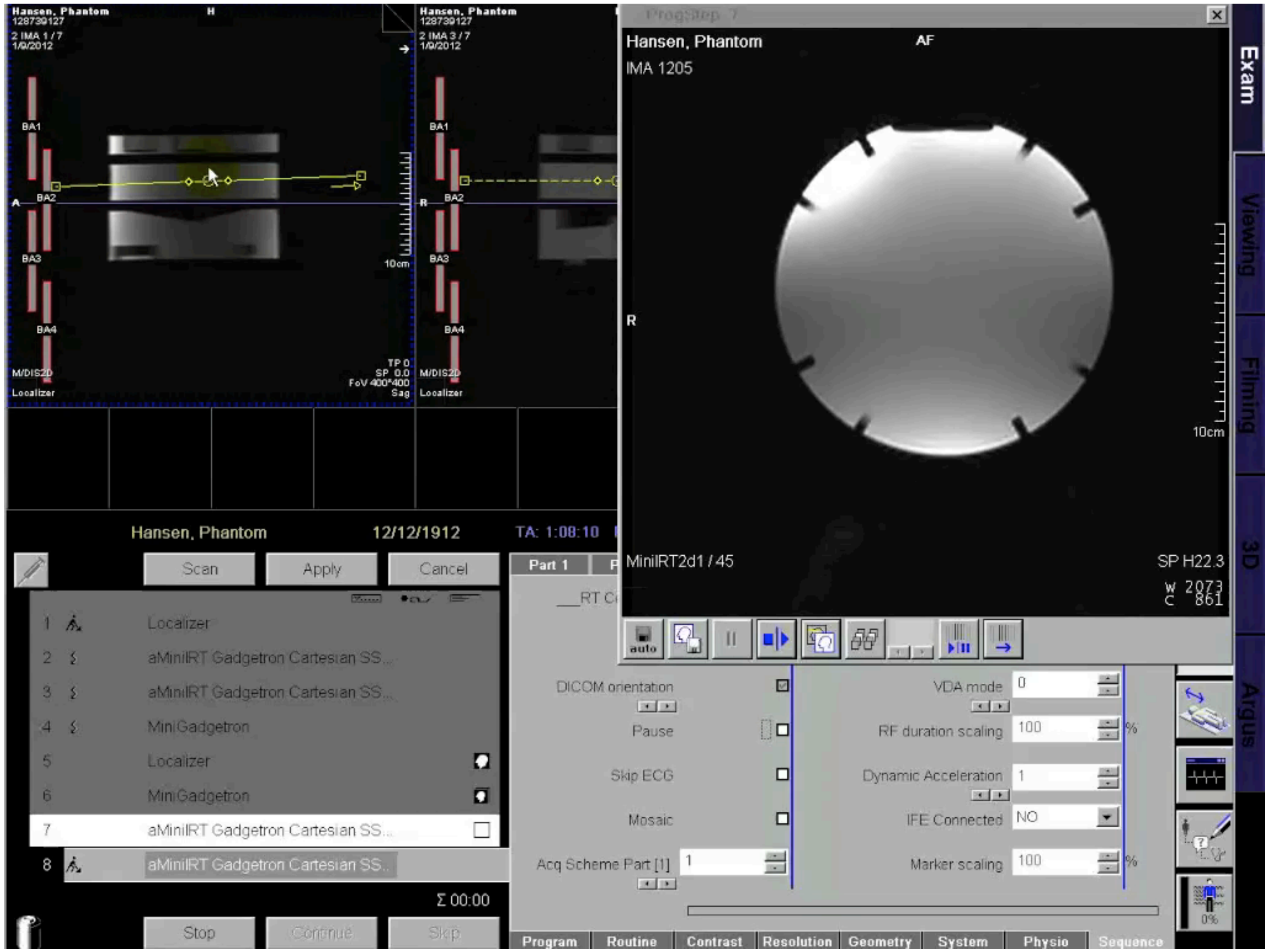


Example - Cartesian GRAPPA



- High-throughput GRAPPA
- Designed for multi-slice 2D real-time imaging (interventional)
- GRAPPA coefficients calculated on GPU

Cartesian GRAPPA



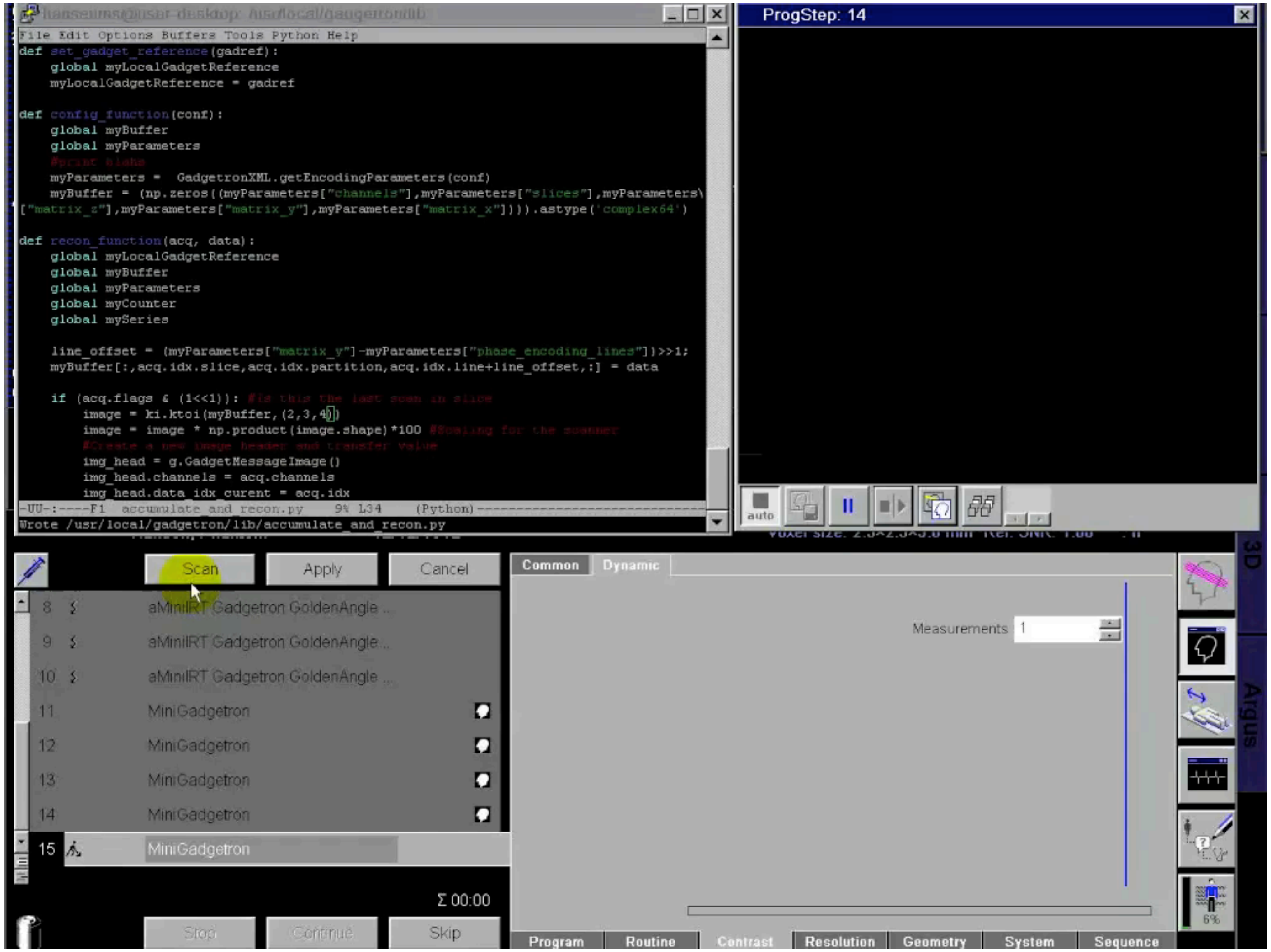
Hansen, Phantom 12/12/1912 TA: 1:08:10

1	Localizer
2	aMiniIRT Gadgetron Cartesian SS...
3	aMiniIRT Gadgetron Cartesian SS...
4	MiniGadgetron
5	Localizer
6	MiniGadgetron
7	aMiniIRT Gadgetron Cartesian SS...
8	aMiniIRT Gadgetron Cartesian SS...

Σ 00:00

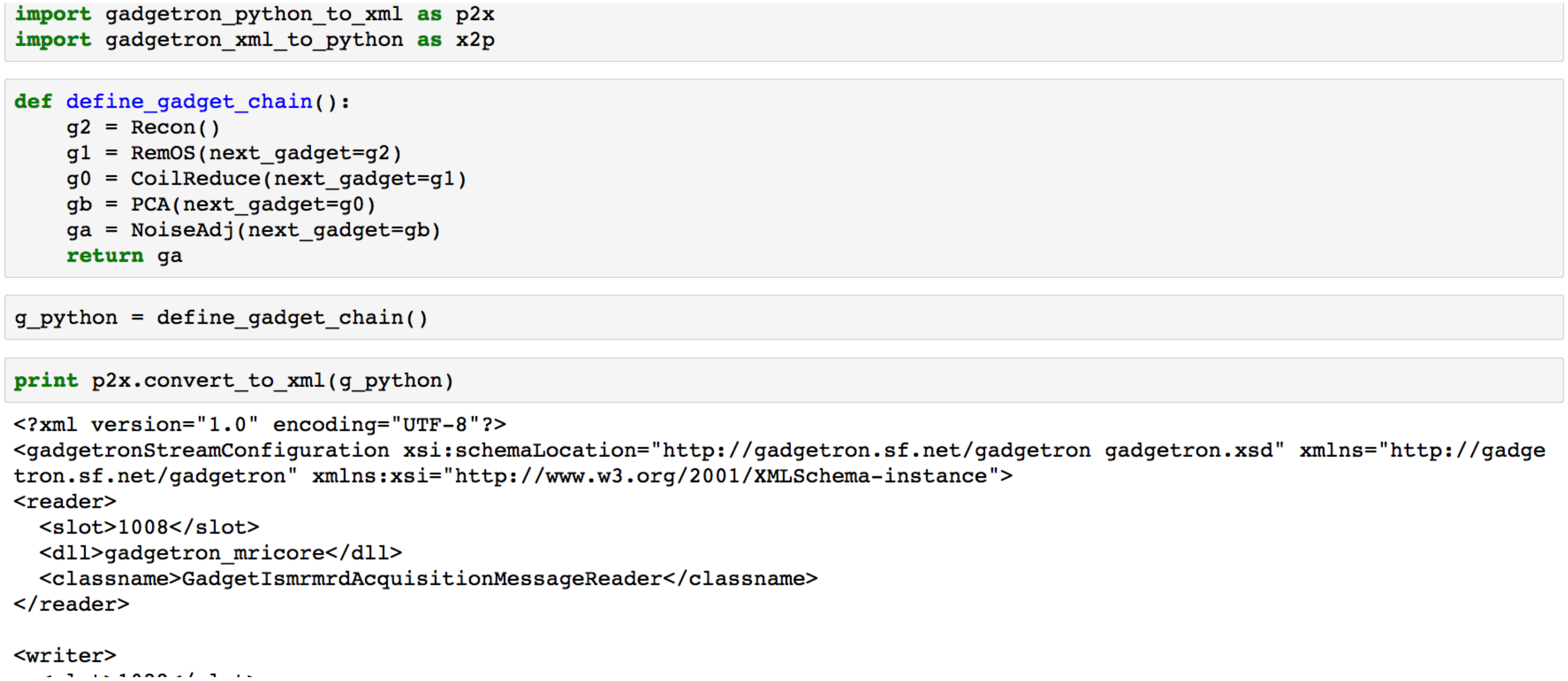
DICOM orientation	☑	VDA mode	0
Pause	☐	RF duration scaling	100 %
Skip ECG	☐	Dynamic Acceleration	1
Mosaic	☐	IFE Connected	NO
Acq Scheme Part [1]	1	Marker scaling	100 %

Online Python Editing



```
def set_gadget_reference(gadref):
    global myLocalGadgetReference
    myLocalGadgetReference = gadref

def config_function(conf):
    global myBuffer
    global myParameters
    #print blahs
    myParameters =  GadgetronXML.getEncodingParameters(conf)
    myBuffer = (np.zeros((myParameters["channels"],myParameters["slices"],myParameters\
["matrix_z"],myParameters["matrix_y"],myParameters["matrix_x"]))).astype('complex64')

def recon_function(acq, data):
    global myLocalGadgetReference
    global myBuffer
    global myParameters
    global myCounter
    global mySeries

    line_offset = (myParameters["matrix_y"]-myParameters["phase_encoding_lines"])>>1;
    myBuffer[:,acq.idx.slice,acq.idx.partition,acq.idx.line+line_offset,:] = data

    if (acq.flags & (1<<1)): #is this the last scan in slice
        image = ki.ktoi(myBuffer,(2,3,4))
        image = image * np.product(image.shape)*100 #Scaling for the scanner
        #Create a new image header and transfer value
        img_head = g.GadgetMessageImage()
        img_head.channels = acq.channels
        img_head.data_idx_curent = acq.idx
-UU-:----F1  accumulate_and_recon.py    9% L34    (Python)--------------------------------
Wrote /usr/local/gadgetron/lib/accumulate_and_recon.py
```

Converting Python chains to Gadgetron



```python
import gadgetron_python_to_xml as p2x
import gadgetron_xml_to_python as x2p
```

```python
def define_gadget_chain():
    g2 = Recon()
    g1 = RemOS(next_gadget=g2)
    g0 = CoilReduce(next_gadget=g1)
    gb = PCA(next_gadget=g0)
    ga = NoiseAdj(next_gadget=gb)
    return ga
```

```python
g_python = define_gadget_chain()
```

```python
print p2x.convert_to_xml(g_python)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<gadgetronStreamConfiguration xsi:schemaLocation="http://gadgetron.sf.net/gadgetron gadgetron.xsd" xmlns="http://gadge
tron.sf.net/gadgetron" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<reader>
  <slot>1008</slot>
  <dll>gadgetron_mricore</dll>
  <classname>GadgetIsmrmrdAcquisitionMessageReader</classname>
</reader>

<writer>
```

Converting Gadgetron XML to Python

```
print x2p.convert_xml(os.environ['GADGETRON_HOME'] + '/share/gadgetron/config/default.xml')
```

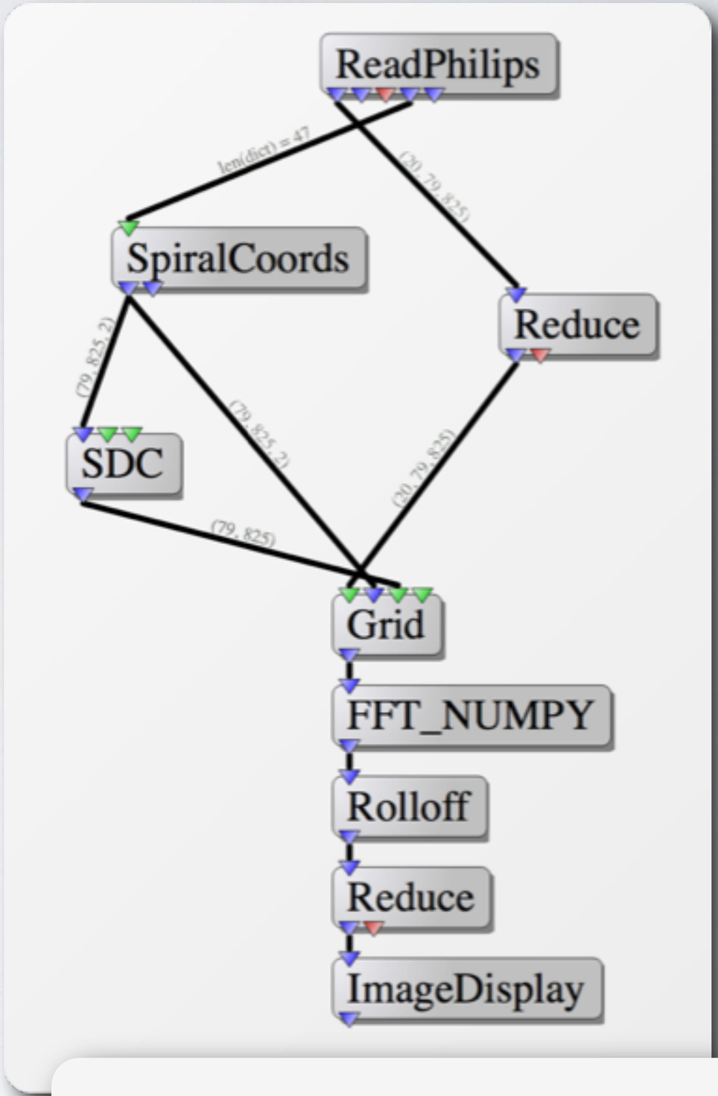
```
# Automatically generated Python representation of /home/hansenms/local/share/gadgetron/config/default.xml

from gadgetron import WrapperGadget

def define_gadget_chain():
    g2 = WrapperGadget("gadgetron_mricore", "ImageFinishGadget", gadgetname="ImageFinish", next_gadget=None)
    g2.prepend_gadget("gadgetron_mricore", "ExtractGadget", gadgetname="Extract")
    g2.prepend_gadget("gadgetron_mricore", "ImageArraySplitGadget", gadgetname="ImageArraySplit")
    g2.prepend_gadget("gadgetron_mricore", "SimpleReconGadget", gadgetname="SimpleRecon")
    g2.prepend_gadget("gadgetron_mricore", "BucketToBufferGadget", gadgetname="Buff")
    g2.set_parameter("Buff", "N_dimension", "")
    g2.set_parameter("Buff", "S_dimension", "")
    g2.set_parameter("Buff", "split_slices", "true")
    g2.prepend_gadget("gadgetron_mricore", "AcquisitionAccumulateTriggerGadget", gadgetname="AccTrig")
    g2.set_parameter("AccTrig", "trigger_dimension", "repetition")
    g2.set_parameter("AccTrig", "sorting_dimension", "slice")
    g2.prepend_gadget("gadgetron_mricore", "RemoveROOversamplingGadget", gadgetname="RemoveROOversampling")
    return g2
```

GPI

A Graphical Development Environment for Scientific Algorithms

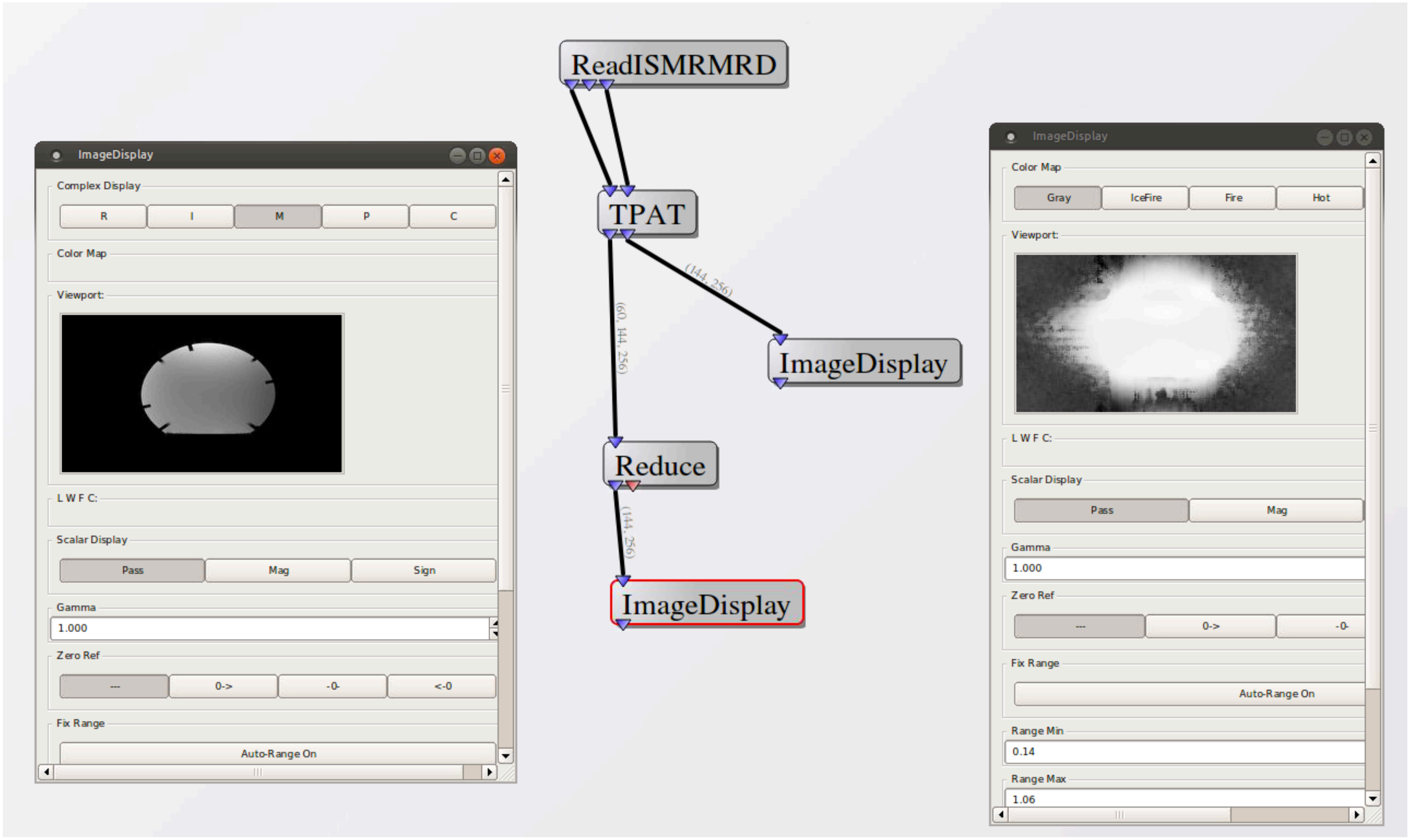
Introduction

- Modular
  - Side by Side Comparisons
  - Ease of Reuse
  - Data & Algorithm Analysis
- Reconstruction, Simulations, Pulse Sequence Development



Spiral Reconstruction

ISMRMRD Support in GPI



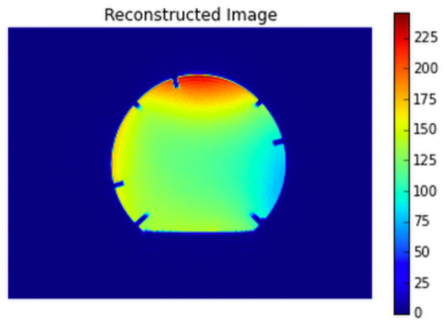
https://github.com/hansenms/gpi_ismrmrd

Running the Python reconstruction

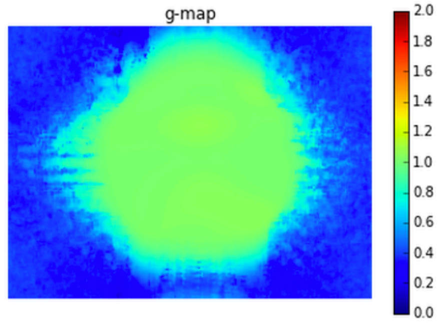
```
# Send in data
#First ISMRMRD XML header
gadget_chain_config(g_python,dset.read_xml_header())

# Loop through the rest of the acquisitions and stuff
for acqnum in range(0,dset.number_of_acquisitions()):
    acq = dset.read_acquisition(acqnum)
    g_python.process(acq.getHead(),acq.data.astype('complex64'))

# Wait for recon to finish
gadget_chain_wait(g_python)
```

```
res_python = get_last_gadget(g_python).get_results()
gmap = get_last_gadget(g_python).gmap
```

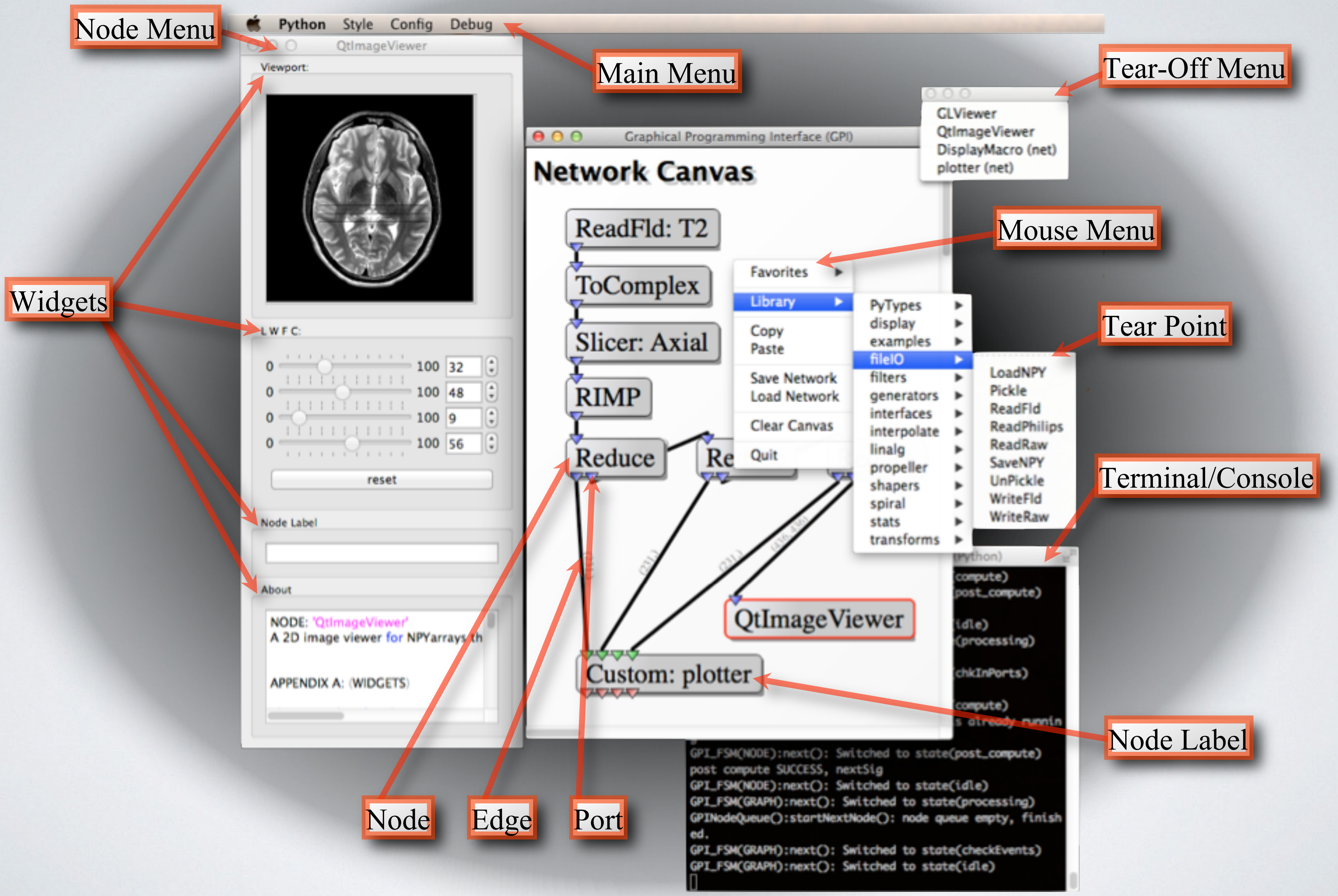
```
show.imshow(abs(np.squeeze(res_python[0][1])),colorbar=True,titles=['Reconstructed Image'])
```



```
show.imshow(abs(np.squeeze(gmap)),colorbar=True,scale=(0,2.0),titles=['g-map'])
```



Graphical interface



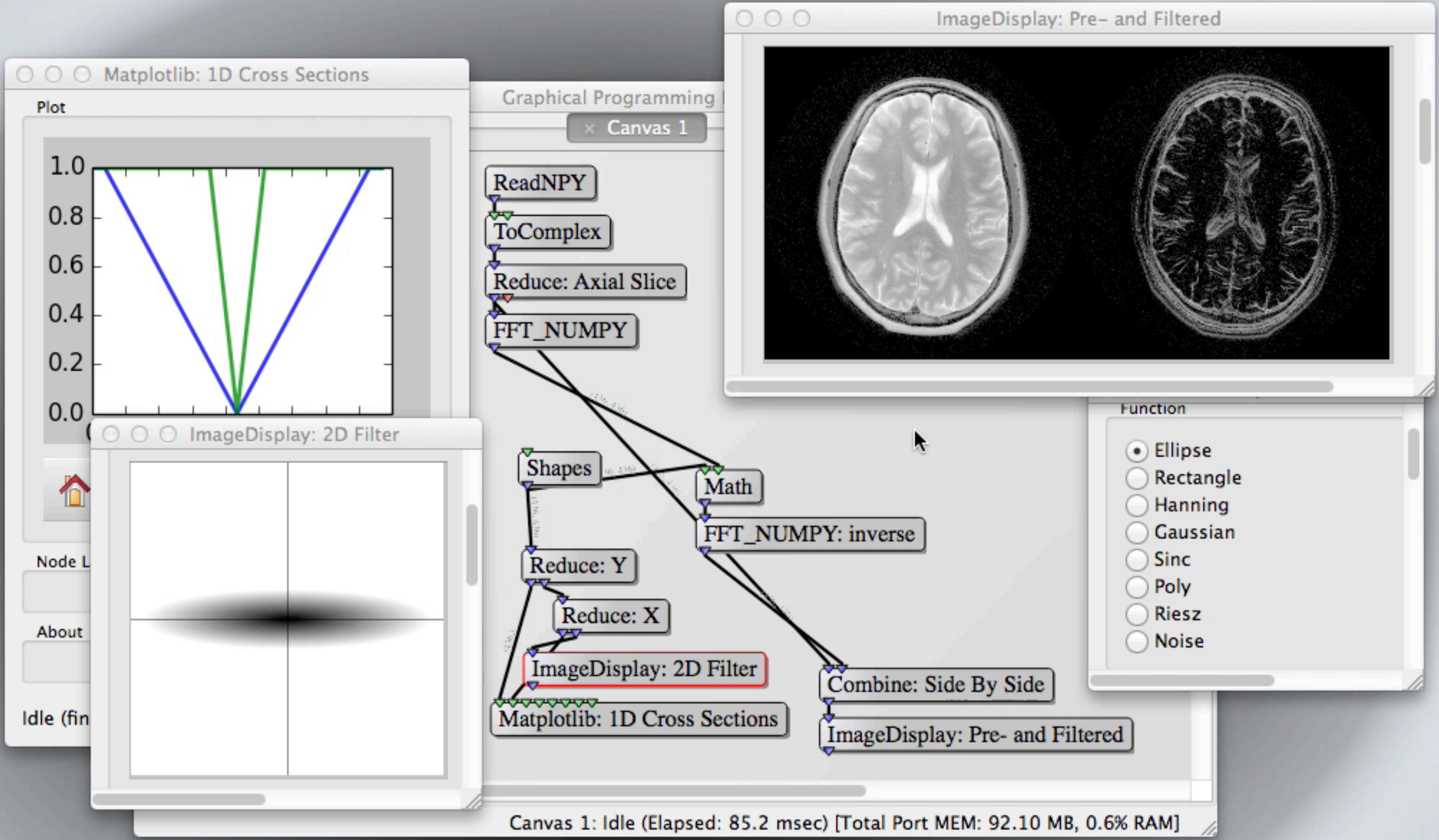
0	100	32	
0	100	48	
0	100	9	
0	100	56	

NODE: 'QtImageViewer'
A 2D image viewer for NPYarrays th

APPENDIX A: (WIDGETS)

```
compute)
post_compute)
idle)
(processing)
(chkInPorts)
compute)
s already runnin
GPI_FSM(NODE):next(): Switched to state(post_compute)
post compute SUCCESS, nextSig
GPI_FSM(NODE):next(): Switched to state(idle)
GPI_FSM(GRAPH):next(): Switched to state(processing)
GPINodeQueue():startNextNode(): node queue empty, finish
ed.
GPI_FSM(GRAPH):next(): Switched to state(checkEvents)
GPI_FSM(GRAPH):next(): Switched to state(idle)
```

K-space filter

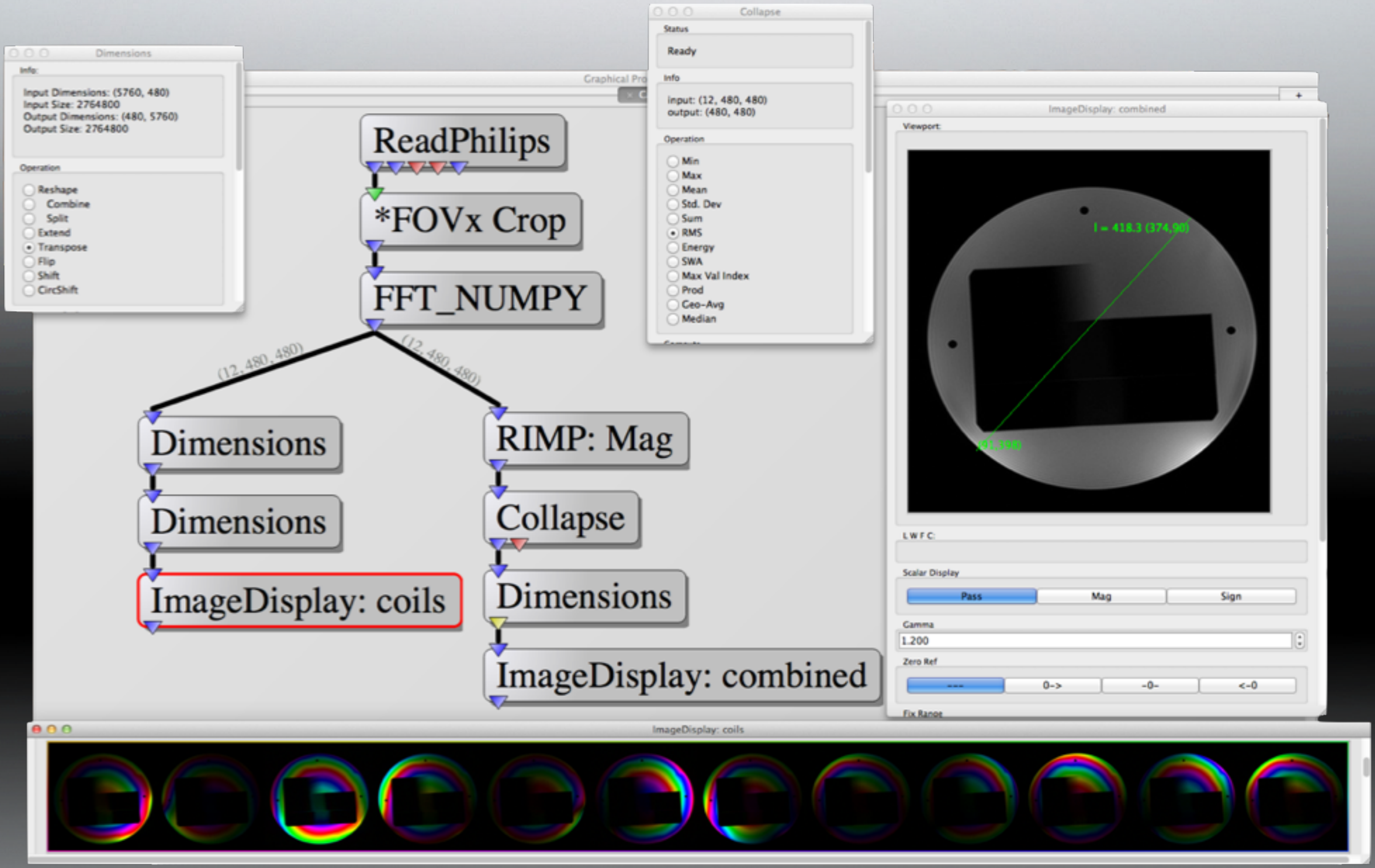


Function

(● Ellipse) Rectangle Hanning Gaussian Sinc Poly Riesz Noise

Canvas 1: Idle (Elapsed: 85.2 msec) [Total Port MEM: 92.10 MB, 0.6% RAM]

coil combination



Dimensions

Info:

Input Dimensions: (5760, 480)
Input Size: 2764800
Output Dimensions: (480, 5760)
Output Size: 2764800

Operation

Reshape
Combine
Split
Extend
[✓ Transpose]
Flip
Shift
CircShift

Collapse

Status

Ready

Info

input: (12, 480, 480)
output: (480, 480)

Operation

Min
Max
Mean
Std. Dev
Sum
[✓ RMS]
Energy
SWA
Max Val Index
Prod
Geo-Avg
Median

ImageDisplay: combined

Gamma: 1.200

codeare

Common Data Exchange And Reconstruction
www.codeare.org

NYU Langone MEDICAL CENTER

Realisation

Algorithm library and client/server application

- N-dimensional Data structure called “**Matrix**”
- **Algorithm dictionary** for arithmetic, linear algebra, Fourier transformation, statistics, optimisation, file IO, …
- Near **textbook high level language** for implementation

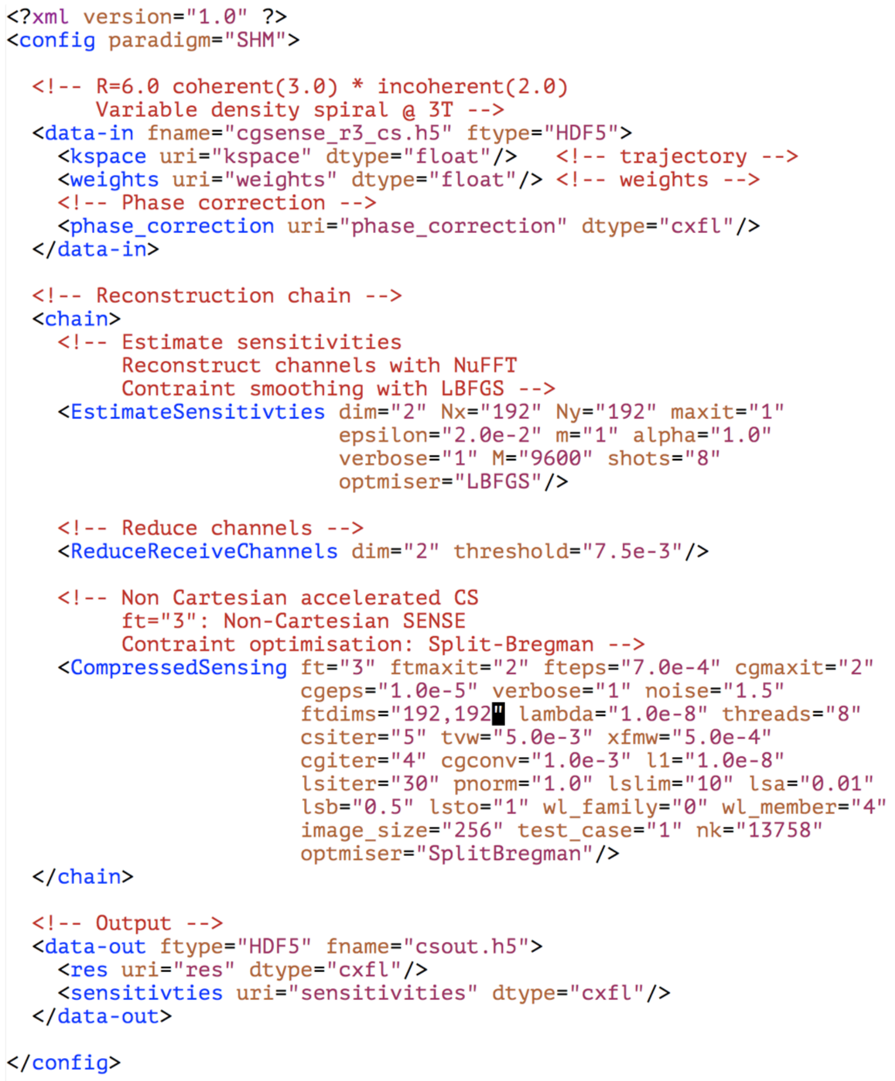
```
Matrix<cf> A = phantom<cf>(256), B,
           cxnoise = randn<cf>(256);
DFT<f> F;
B = F / (F*A + cxnoise);
print (B, ‘B.png’, ‘r600’);
fopen (f, ‘B.mat’, WRITE);
fwrite (f, conj(B));
fclose (f);
```

- Usage
  - C++ **library**
  - **stand alone application**
  - **client server application** for realtime scanner feedback

Usage

Reconstruction chain

- Reconstruction strategies often involve multiple reusable steps
- Example on write is part of the package to demonstrate how this is achieved in codeare
- Real-day example involves the estimation of receive sensitivties, reduction of coils and subsequent CS reconstruction



```xml
<?xml version="1.0" ?>
<config paradigm="SHM">

  <!-- R=6.0 coherent(3.0) * incoherent(2.0)
       Variable density spiral @ 3T -->
  <data-in fname="cgsense_r3_cs.h5" ftype="HDF5">
    <kspace uri="kspace" dtype="float"/>   <!-- trajectory -->
    <weights uri="weights" dtype="float"/> <!-- weights -->
    <!-- Phase correction -->
    <phase_correction uri="phase_correction" dtype="cxfl"/>
  </data-in>

  <!-- Reconstruction chain -->
  <chain>
    <!-- Estimate sensitivities
         Reconstruct channels with NuFFT
         Contraint smoothing with LBFGS -->
    <EstimateSensitivties dim="2" Nx="192" Ny="192" maxit="1"
                          epsilon="2.0e-2" m="1" alpha="1.0"
                          verbose="1" M="9600" shots="8"
                          optmiser="LBFGS"/>

    <!-- Reduce channels -->
    <ReduceReceiveChannels dim="2" threshold="7.5e-3"/>

    <!-- Non Cartesian accelerated CS
         ft="3": Non-Cartesian SENSE
         Contraint optimisation: Split-Bregman -->
    <CompressedSensing ft="3" ftmaxit="2" fteps="7.0e-4" cgmaxit="2"
                       cgeps="1.0e-5" verbose="1" noise="1.5"
                       ftdims="192,192" lambda="1.0e-8" threads="8"
                       csiter="5" tvw="5.0e-3" xfmw="5.0e-4"
                       cgiter="4" cgconv="1.0e-3" l1="1.0e-8"
                       lsiter="30" pnorm="1.0" lslim="10" lsa="0.01"
                       lsb="0.5" lsto="1" wl_family="0" wl_member="4"
                       image_size="256" test_case="1" nk="13758"
                       optmiser="SplitBregman"/>
  </chain>

  <!-- Output -->
  <data-out ftype="HDF5" fname="csout.h5">
    <res uri="res" dtype="cxfl"/>
    <sensitivties uri="sensitivities" dtype="cxfl"/>
  </data-out>

</config>
```

Critical Components of Pipeline Processing

- Modularity
  - Reusable code
- Event driven processing
- Multi-threading
- Standardized raw data representation
  - Meaningful data labels

THANK YOU

- Jeff Voskuil, GE Healthcare
- Amol Pednekar, Philips Healthcare
- Wes Gilson, Siemens Healthcare
- Nick Zwart, GPI, Barrow Neurological Institute, Phoenix AZ
- Kaveh Vahedipour, NYU
- Souheil Inati, National Institutes of Health
- Joe Naegele, National Institutes of Health
- Hui Xue, National Institutes of Health

Open Source Resources

- Course Materials: http://hansenms.github.io/sunrise
- ISMRMRD:
  - http://ismrmrd.github.io
  - http://github.com/ismrmrd/ismrmrd-python
  - http://github.com/ismrmrd/ismrmrd-python-tools
- GADGETRON:
  - http://gadgetron.github.io
- GPI:
  - http://gpilab.com
  - http://github.com/hansenms/gpi_ismrmrd
- CODEARE:
  - http://www.codeare.org