GADGETRON

Michael S. Hansen, PhD
Magnetic Resonance Technology Program
National Institutes of Health - NHLBI

National Heart, Lung,
and Blood Institute

QUESTIONS/COMMENTS

EMAIL: michael.hansen@nih.gov

Twitter: @ReconstructThis

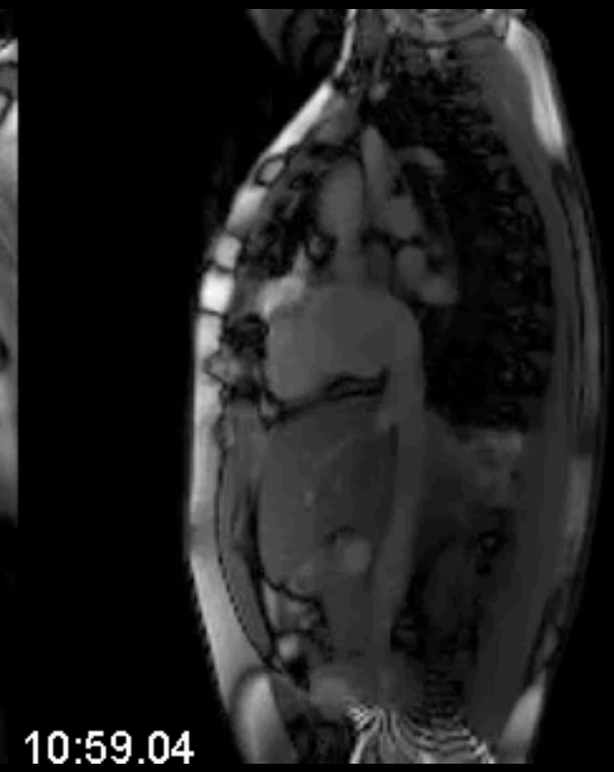
National Heart, Lung,
and Blood Institute

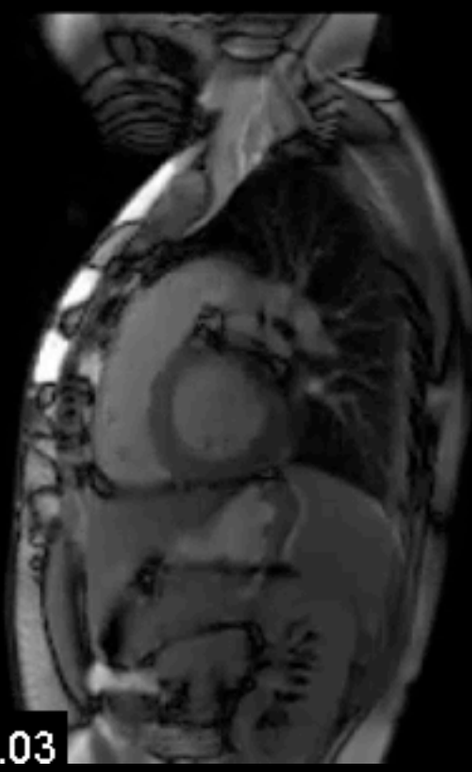
Outline

- Gadgetron Motivation
- Design and Implementation
  - Streaming Architecture
  - Data Formats
- Examples
- Questions

Diagnostic MR Right Heart Cath

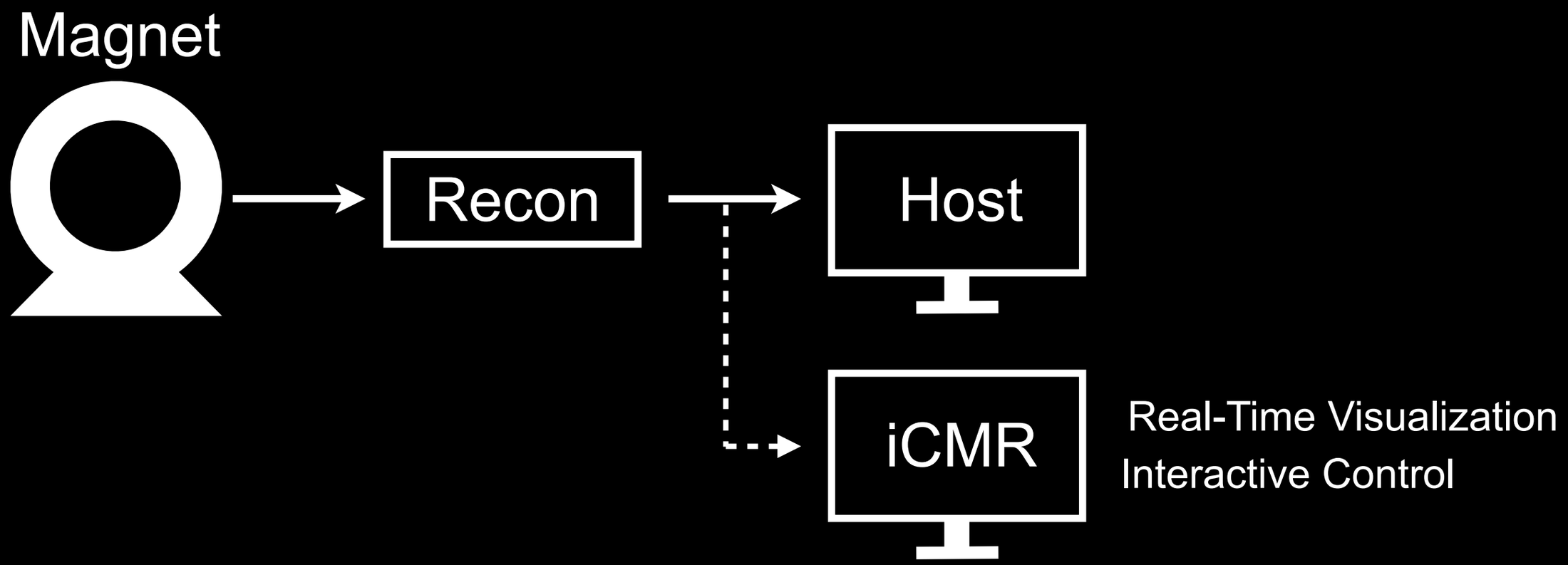
First case at Children's National Medical Center, Washington, DC



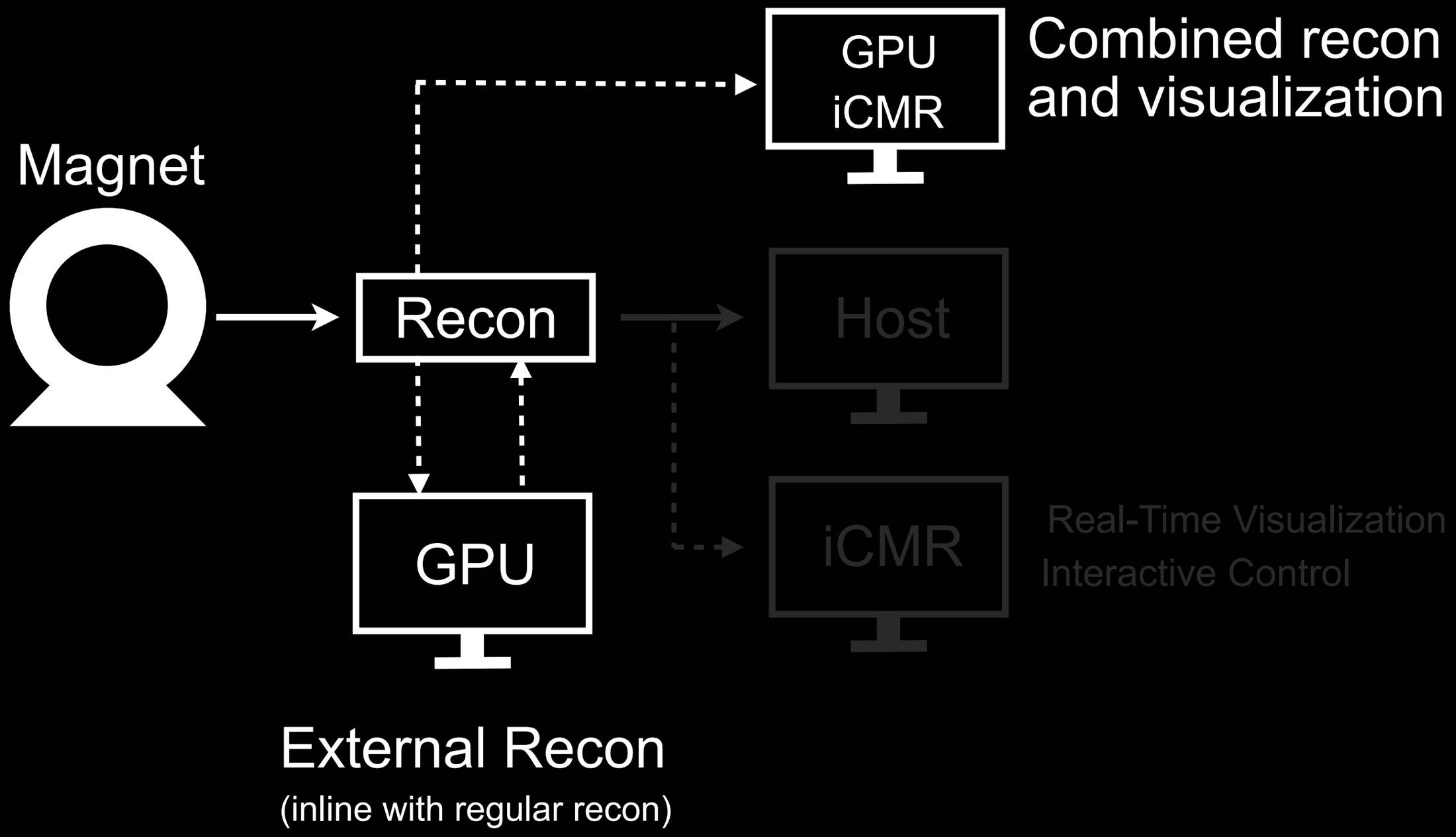


March 18, 2015

Integrating External Computing Resources



Integrating External Computing Resources



Gadgetron

FULL PAPER

Magnetic Resonance in Medicine 000:000–000 (2012)

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 through a plugin-like architecture without recompiling the basic framework infrastructure. Gadgets are typically implemented in C/C++, but the framework includes wrapper Gadgets that allow the user to implement new modules in the Python scripting language for rapid prototyping. In addition to the streaming framework infrastructure, the Gadgetron comes with a set of dedicated toolboxes in shared libraries for medical image reconstruction. This includes generic toolboxes for data-parallel (e.g., GPU-based) execution of compute-intensive components. The basic framework architecture is independent of medical imaging modality, but this article focuses on its application to Cartesian and non-Cartesian parallel magnetic resonance imaging. Magn Reson Med 000:000–000, 2012. © 2012 Wiley Periodicals, Inc.

Key words: image reconstruction; open source; GPU

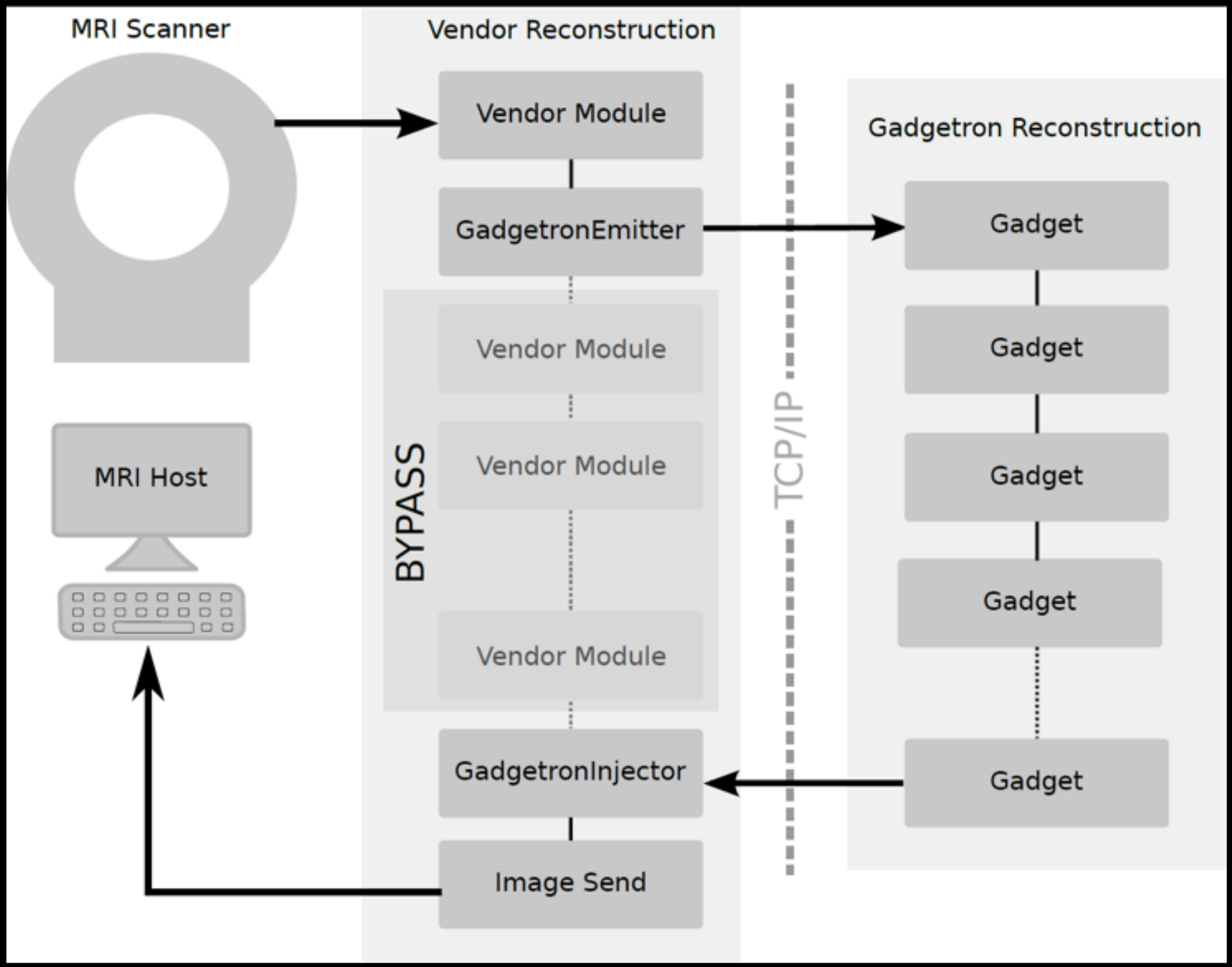
INTRODUCTION

Image reconstruction software is an integral part of all

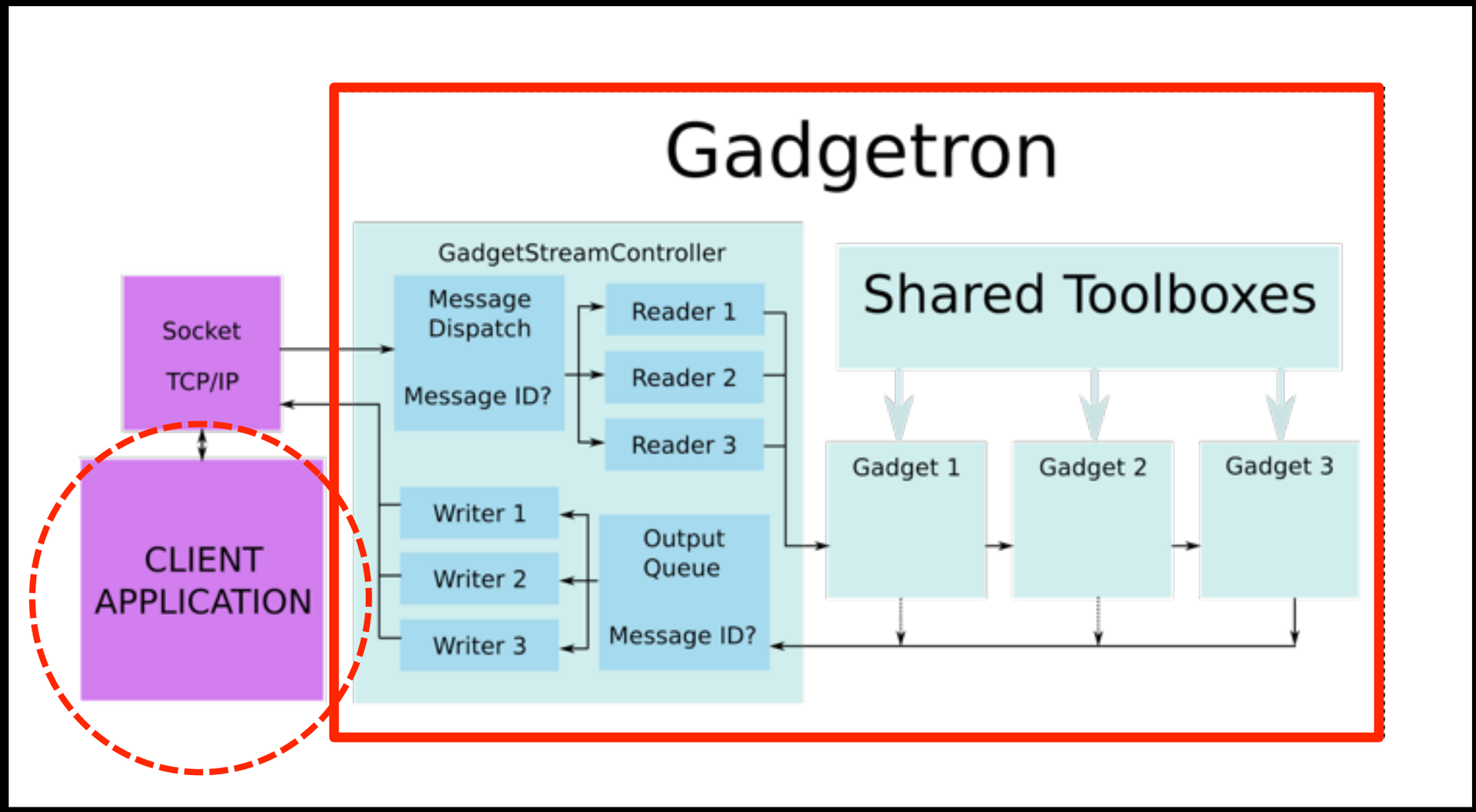
deal of accessory code, some of which could be vendor specific or even contain vendor-provided code that cannot be shared. Regardless of the reasons, it undermines the scientific process that readers and reviewers are prevented from reproducing the results of reconstruction research articles. It is exceedingly 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 result, researchers who wish to build on previous work by other groups often have to reimplement previous work for comparison purposes, as there is no available common platform for sharing medical image reconstruction algorithms. Given the complexities of new reconstruction algorithms, it may be difficult or indeed impossible reimplement the methods described in research articles. This problem has been pointed out by other researchers, e.g. in the report by Buckheit and Donoho (6).

The proof-of-concept implementations that form the basis of many publications have further limitations. They are often unsuitable for deployment in a clinical (or clinical research) environment where they could be tested in broader patient populations. More specifically, the reconstruction algorithm development is often done in "offline" applications that are difficult to integrate with medical devices in such a way that the reconstruction can be performed "online." Consequently, proper eval-

Generic Scanner Integration



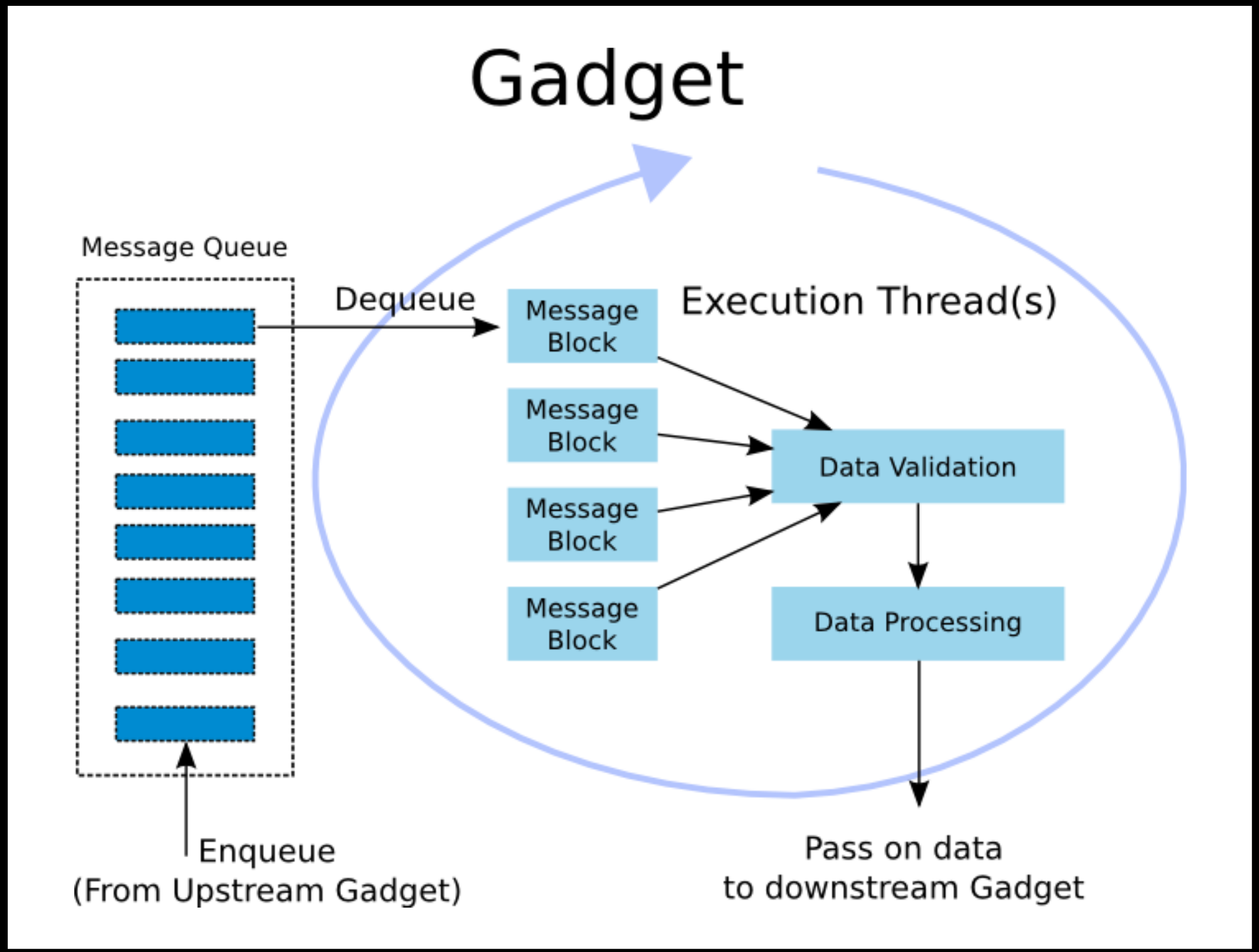
Gadgetron Architecture



Scanner Reconstruction

Configured dynamically at run time
XML Configuration

Gadgetron Architecture

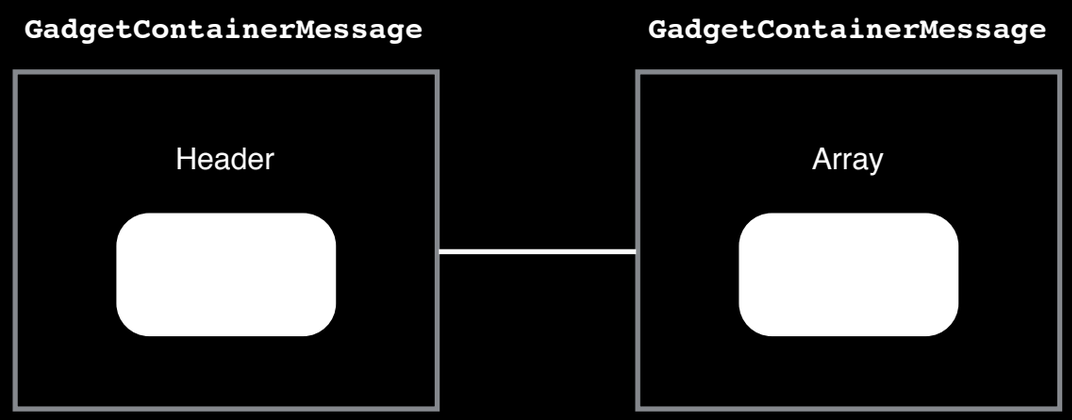


Gadgets

```
class Gadget : public ACE_Task<ACE_MT_SYNCH>
{

public:
   virtual int svc(void)
   {
      //Pick up package from queue

      //Call process
      if (this->process(m) == -1) {
         //Handle error
      }
      return 0;
   }

   //More function (left out for simplicity)

protected:
   virtual int process(ACE_Message_Block * m) = 0;

   virtual int process_config(ACE_Message_Block * m) {
      return 0;
   }

};
```

Container Messages

The Gadgetron uses a generic container capable of storing any object




```
GadgetContainerMessage<MyClass>* m =
  new GadgetContainerMessage<MyClass>();

MyClass* mc = m->getObjectPtr();

//Do something with mc

m->release(); //Delete the message block and containing data
```

Gadgets and Container Messages

```
int process(ACE_Message_Block* mb)
{

  GadgetContainerMessage<MyClass>* m =
    AsContainerMessage<MyClass>(mb);

  if (m) {
    MyClass* mc = m->getObjectPtr();

    //Do something with mc

  } else {
    //Something went wrong, deal with error
    return -1;
  }

  mb->release();

  return 0;
}
```

Gadgets and Container Messages

```
int process(ACE_Message_Block* mb)
{

  GadgetContainerMessage<MyClass>* m1 =
    AsContainerMessage<MyClass>(mb);

  GadgetContainerMessage<MyOtherClass>* m2 =
    AsContainerMessage<MyOtherClass>(mb->cont());

  if (m1 && m2) {
    MyClass* mc = m1->getObjectPtr();
    MyOtherClass* moc = m2->getObjectPtr();

    //Do something with mc

  } else {
    //Something went wrong, deal with error
    return -1;
  }

  mb->release(); //This deletes both message blocks

  return 0;
}
```

Gadgets Accepting Predefined Data

```
template <class P1, class P2> class Gadget2 : public Gadget
{
protected:
   int process(ACE_Message_Block* mb)
   {
     //Do type checking
   }

   virtual int process(GadgetContainerMessage<P1>* m1,
     GadgetContainerMessage<P2>* m2) = 0;
};
```

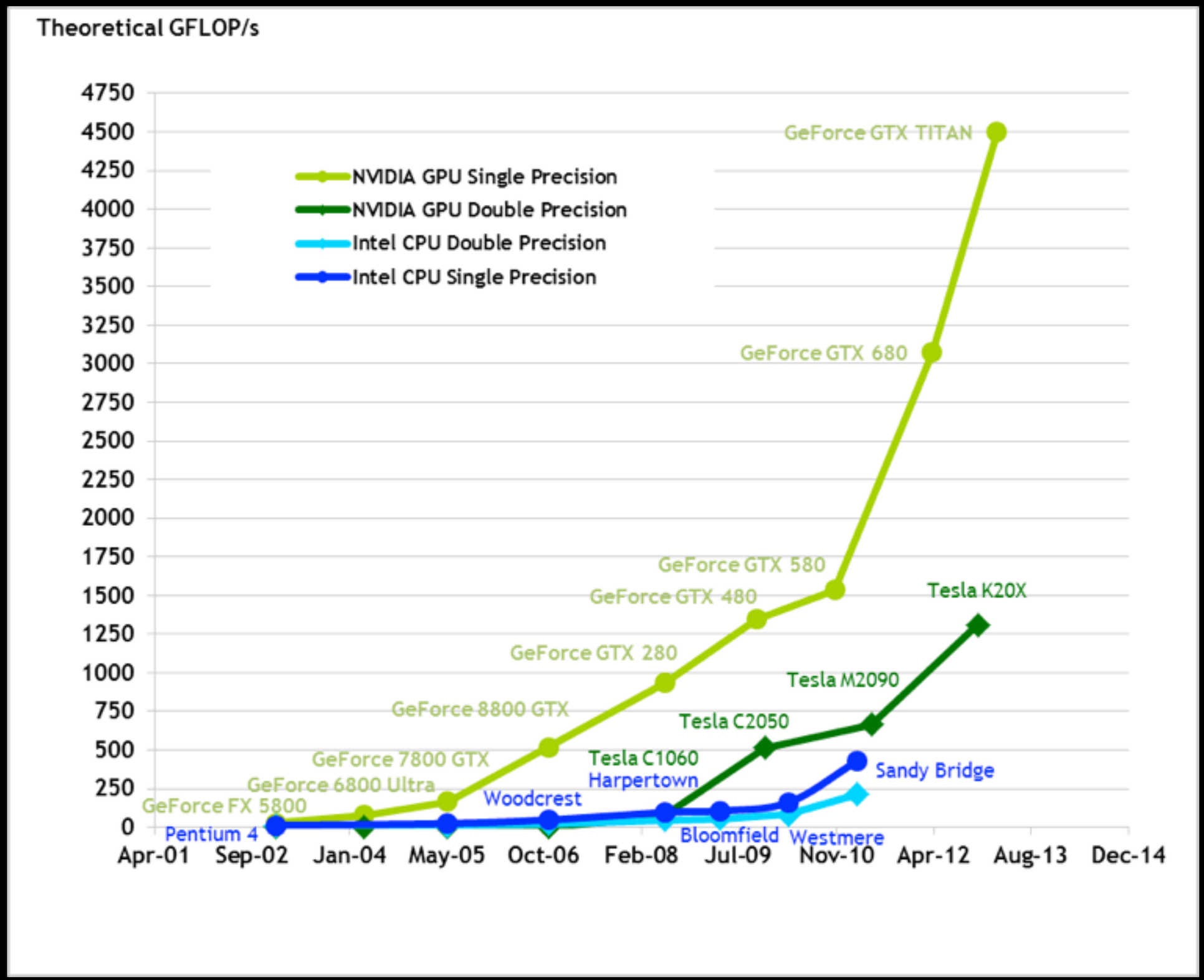
Gadgets - FFT Example

```
#include "gadgetroncore_export.h"
#include "Gadget.h"
#include "ismrmrd.h"
#include "hoNDArray.h"
#include <complex>

class EXPORTGADGETSMRICORE FFTGadget :
public Gadget2<ISMRMRD::ImageHeader, hoNDArray< std::complex<float> > >
{
 public:
  GADGET_DECLARE(FFTGadget)

 protected:
  virtual int process(
     GadgetContainerMessage< ISMRMRD::ImageHeader >* m1,
     GadgetContainerMessage< hoNDArray< std::complex<float> > >* m2);
};
```

Gadgets - FFT Example

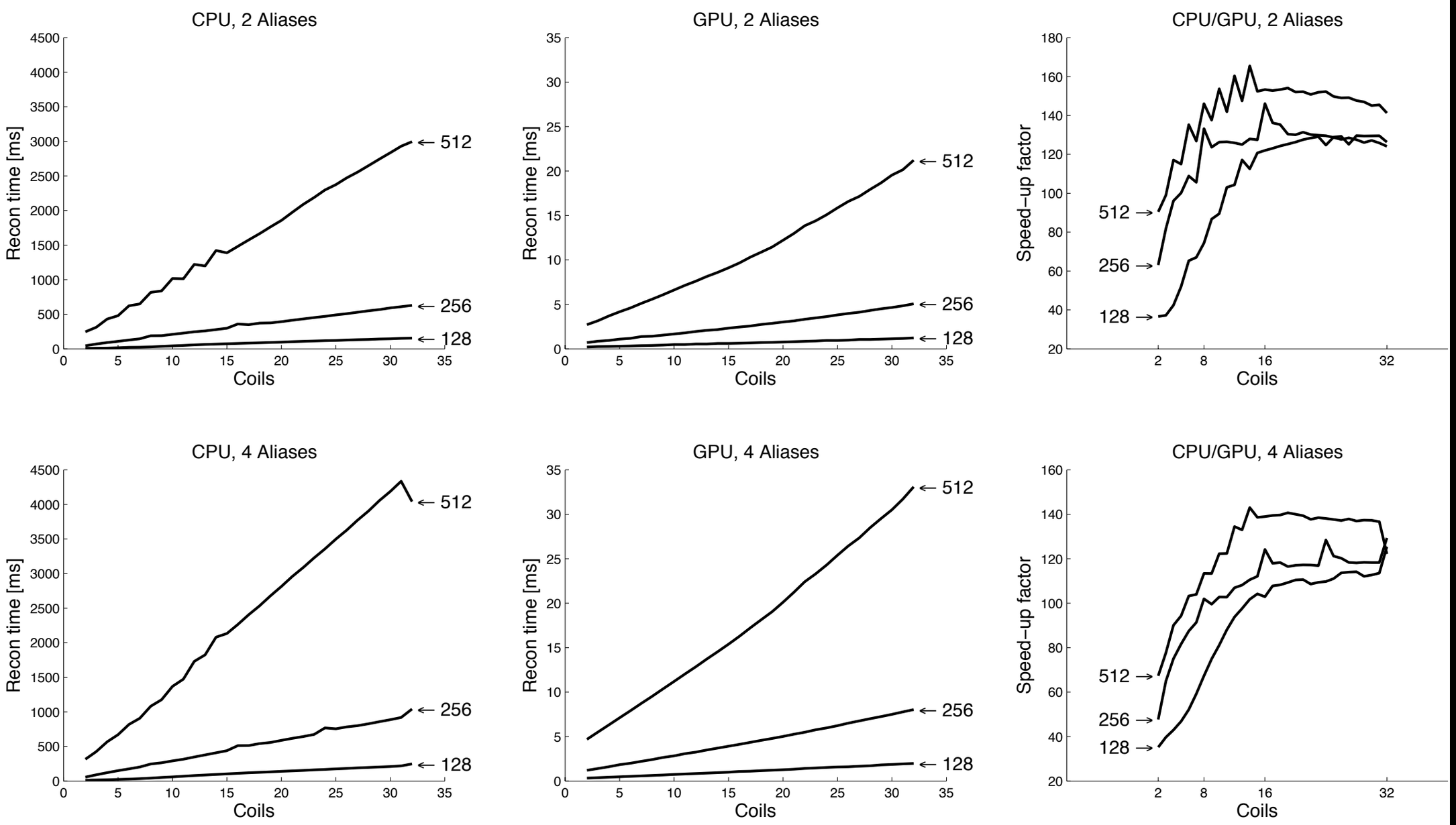
```
#include "FFTGadget.h"
#include "FFT.h"

int FFTGadget::process(
    GadgetContainerMessage< ISMRMRD::ImageHeader >* m1,
    GadgetContainerMessage< hoNDArray< std::complex<float> > >* m2)
{
  hoNDFFT<float>::instance()->ifft(m2->getObjectPtr(),0);
  hoNDFFT<float>::instance()->ifft(m2->getObjectPtr(),1);
  hoNDFFT<float>::instance()->ifft(m2->getObjectPtr(),2);

  if (this->next()->putq(m1) < 0) {
     return GADGET_FAIL;
  }

  return GADGET_OK;
}

GADGET_FACTORY_DECLARE(FFTGadget)
```

GPU vs CPU Speed



*NVIDIA CUDA Programming Guide, v. 5

SENSE inversion speed GPU vs. GPU



GPU: GeForce 8800 Ultra (NVIDIA, USA), 128 cores, with 768MB RAM, core clock speed of 675MHz.
CPU: 64-bit (Intel Xeon) Linux workstation with 4GB RAM, clock frequency of 3200 MHz.

Hansen et al., Magn Reson Med. 2008 Mar;59(3):463-8.

GPU Gridding Implementation

		Conv	FFT	Deap	Total
256x256 samples	**CPU**	1.8 e^{-1}	1.8 e^{-2}	1.8 e^{-3}	2.0 e^{-1}
	GPU	3.5 e^{-3}	8.9 e^{-3}	4.5 e e^{-4}	1.4 e e^{-2}
	Factor	**51**	**2.0**	**4.0**	**14**
512x512 samples	**CPU**	1.2 e^{0}	7.4 e^{-2}	7.3 e^{-3}	1.3 e^{0}
	GPU	1.3 e^{-2}	4.5 e^{-2}	2.3 e^{-3}	6.4 e^{-2}
	Factor	**92**	**1.6**	**3.2**	**20**

Sorensen et al., IEEE Trans Med Imaging. 2008 Apr;27(4):538-47.

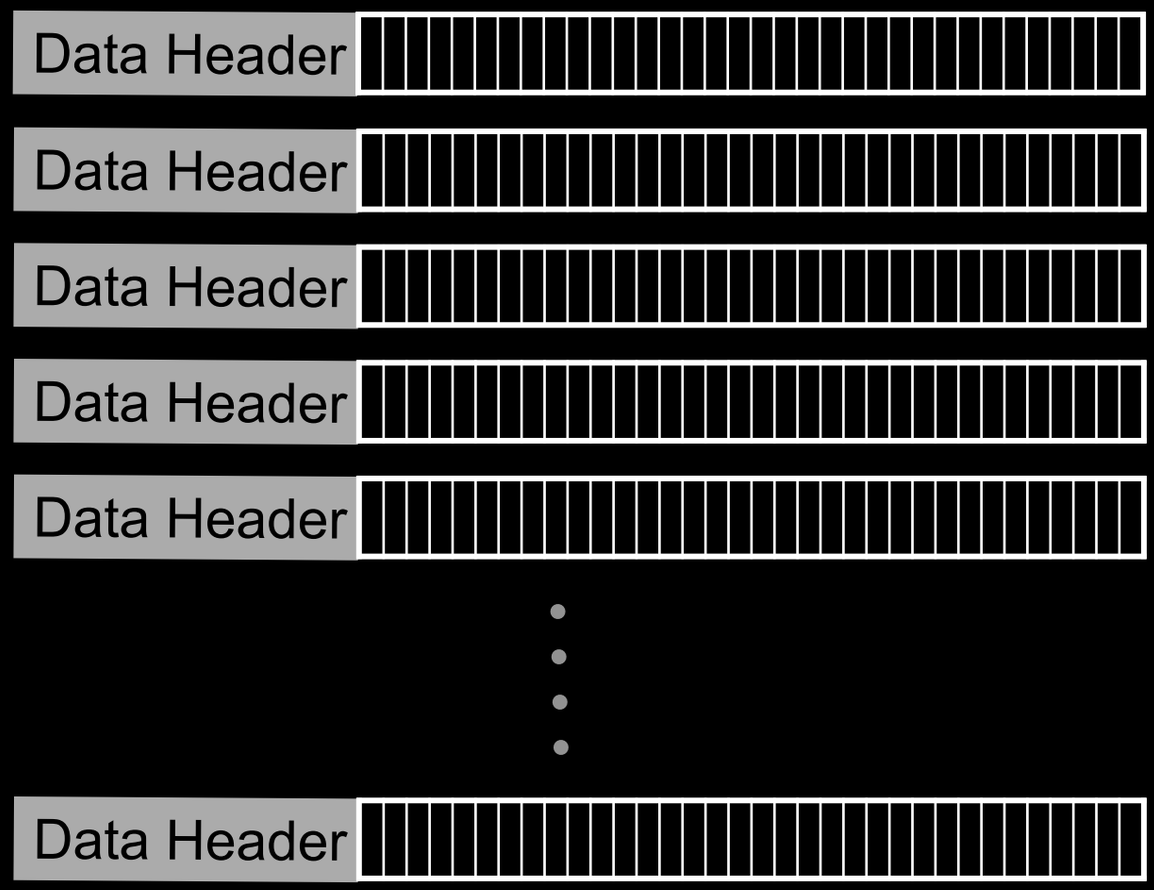
ISMRM Raw Data - Standard Raw Data

HDF5 File Container

XML Header

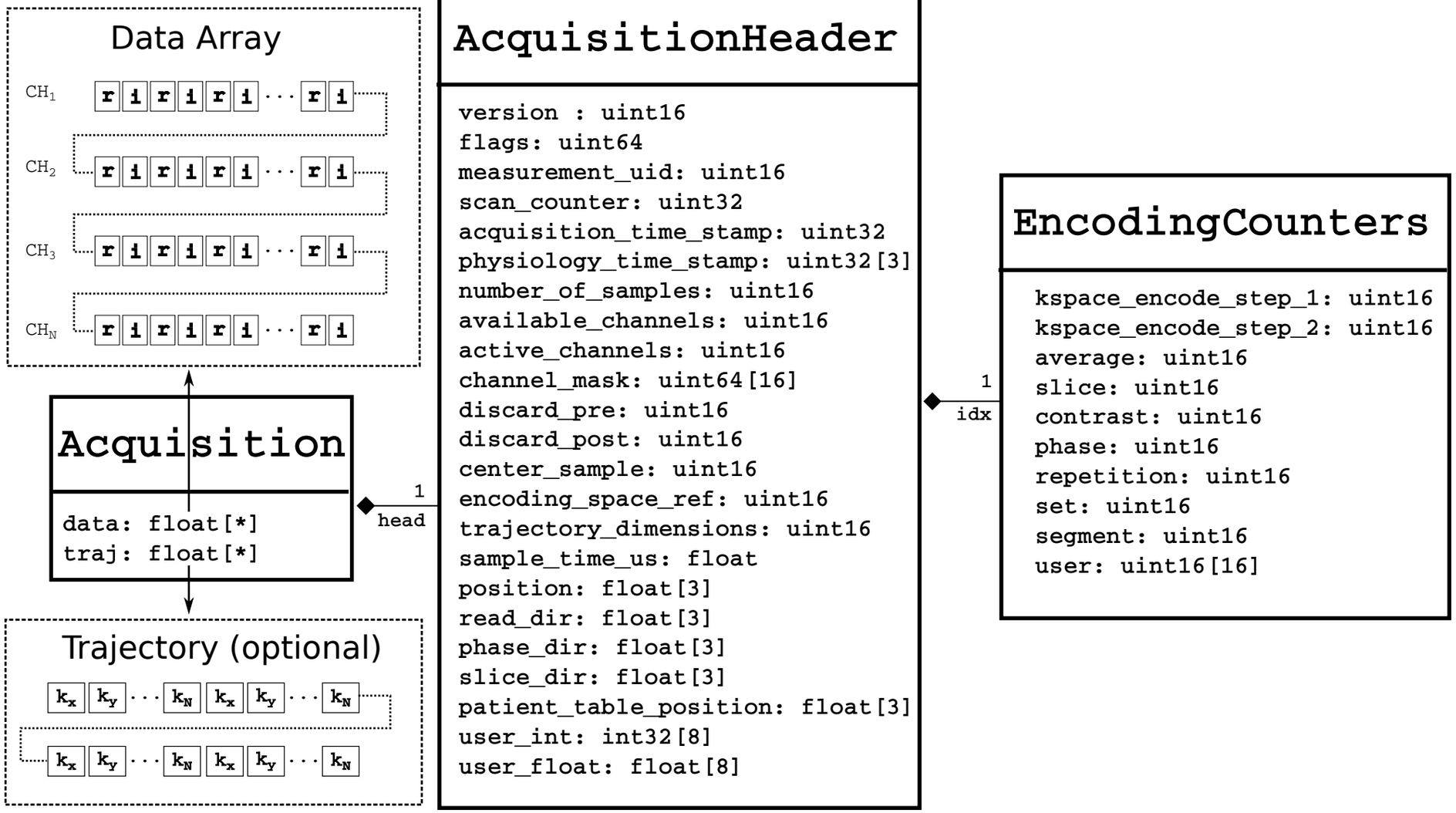
```
<?xml version="1.0"?>
<ismrmrdHeader xmlns="http://www.ismrm.org/ISMRMRD"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xsi:schemaLocation="http://www.ismrm.org/ISMRMRD ismrmrd.xsd">
   <experimentalConditions>
    <H1resonanceFrequency_Hz>63642459</H1resonanceFrequency_Hz>
  </experimentalConditions>
  <encoding>
    <trajectory>cartesian</trajectory>
    <encodedSpace>
      <matrixSize>
        <x>256</x><y>140</y><z>80</z>
      </matrixSize>
      <fieldOfView_mm>
        <x>600</x><y>328.153125</y><z>160</z>
      </fieldOfView_mm>
    </encodedSpace>
    <reconSpace>
      <matrixSize>
    <x>128</x><y>116</y><z>64</z>
      </matrixSize>
      <fieldOfView_mm>
        <x>300</x><y>271.875</y><z>128</z>
      </fieldOfView_mm>
    </reconSpace>
    <encodingLimits>
      <kspace_encoding_step_1>
        <minimum>0</minimum>
        <maximum>83</maximum>
        <center>28</center>
      </kspace_encoding_step_1>
     </encodingLimits>
  </encoding>
</ismrmrdHeader>
```

Samples

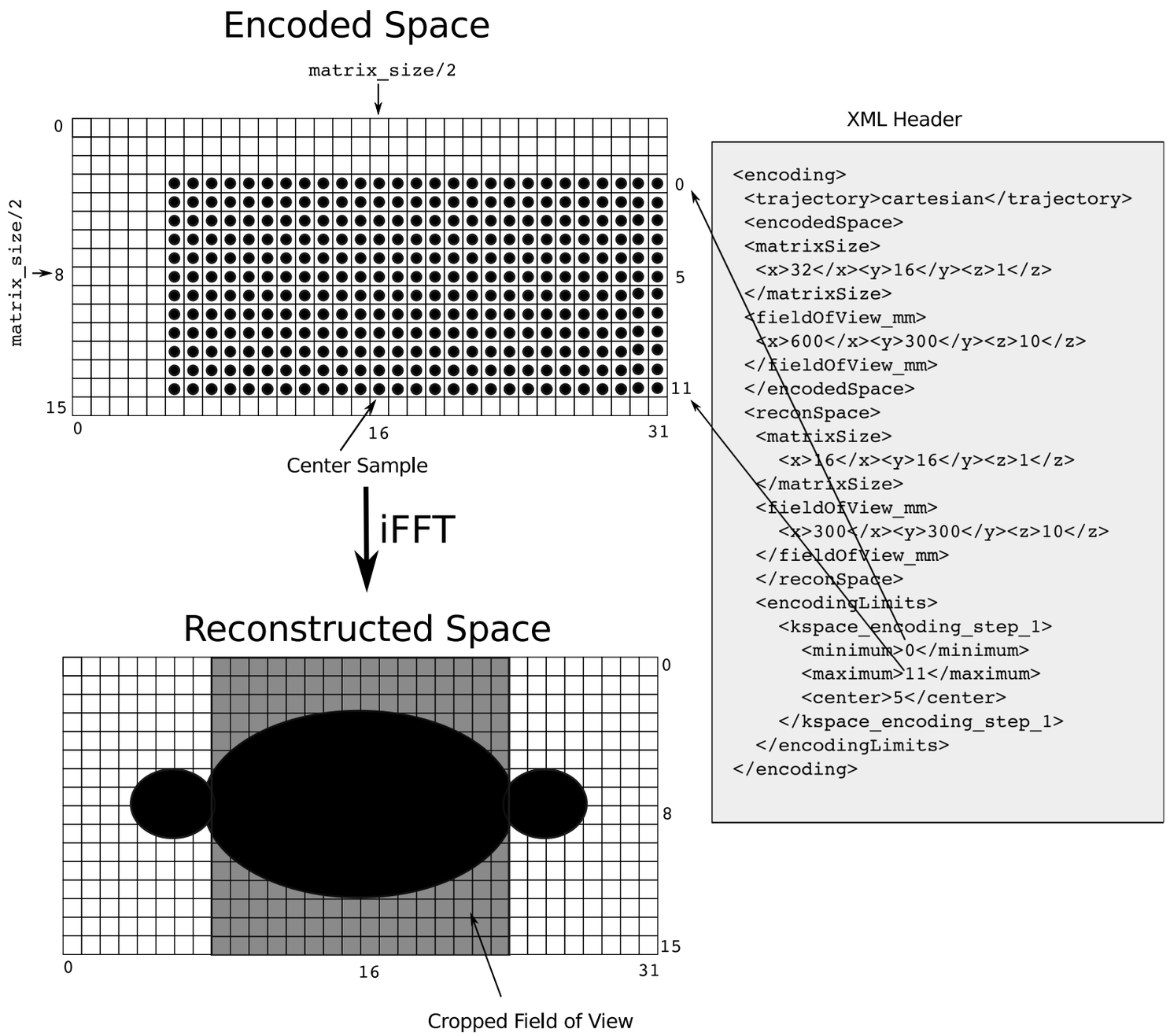


http://ismrmrd.github.io

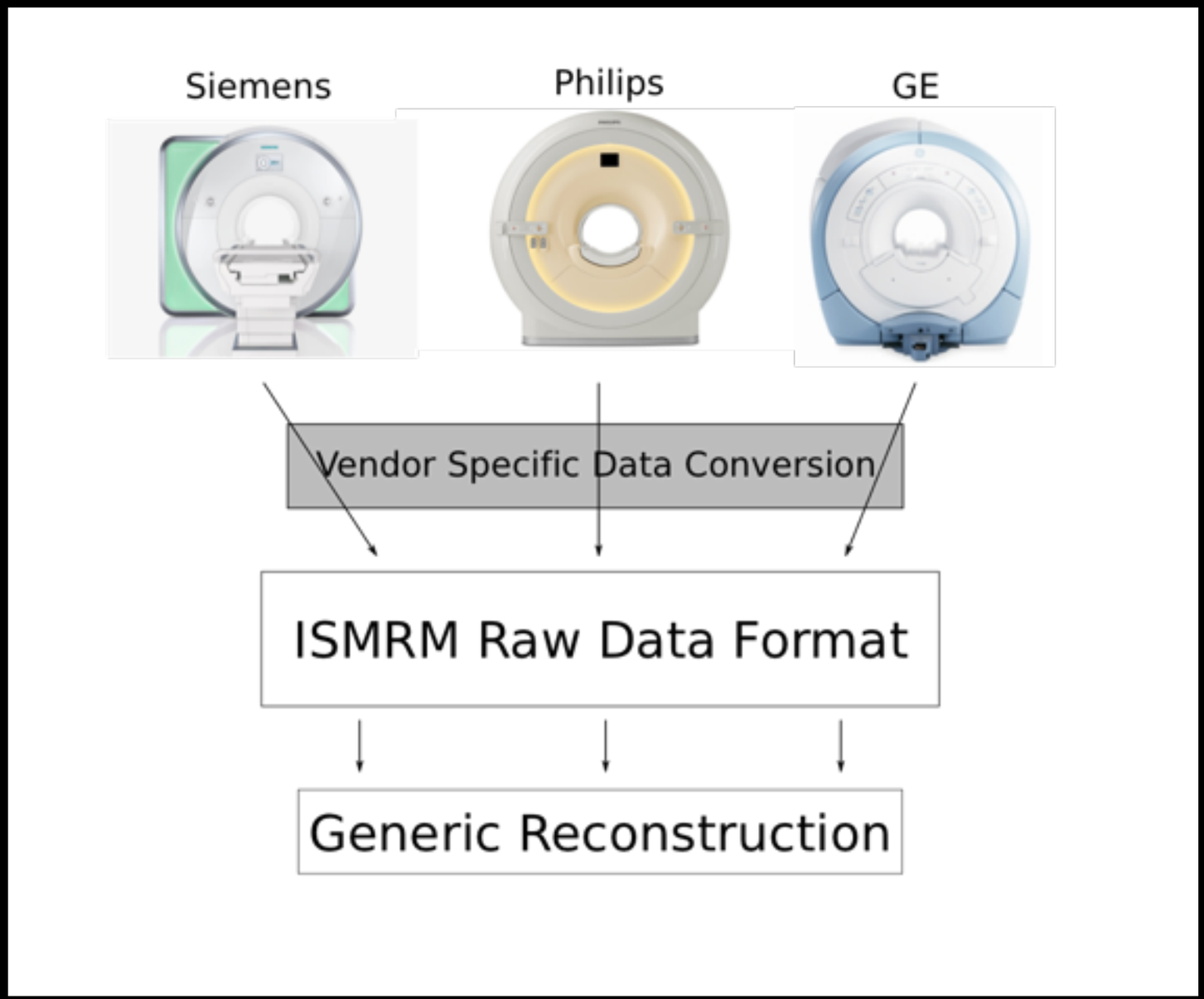
ISMRMRD


Data Array
CH1
CH2
CH3
CHN
r i r i r i ··· r i
Acquisition
data: float[*]
traj: float[*]
1
head
Trajectory (optional)
kx ky ··· kN kx ky ··· kN
AcquisitionHeader
version : uint16
flags: uint64
measurement_uid: uint16
scan_counter: uint32
acquisition_time_stamp: uint32
physiology_time_stamp: uint32[3]
number_of_samples: uint16
available_channels: uint16
active_channels: uint16
channel_mask: uint64[16]
discard_pre: uint16
discard_post: uint16
center_sample: uint16
encoding_space_ref: uint16
trajectory_dimensions: uint16
sample_time_us: float
position: float[3]
read_dir: float[3]
phase_dir: float[3]
slice_dir: float[3]
patient_table_position: float[3]
user_int: int32[8]
user_float: float[8]
1
idx
EncodingCounters
kspace_encode_step_1: uint16
kspace_encode_step_2: uint16
average: uint16
slice: uint16
contrast: uint16
phase: uint16
repetition: uint16
set: uint16
segment: uint16
user: uint16[16]

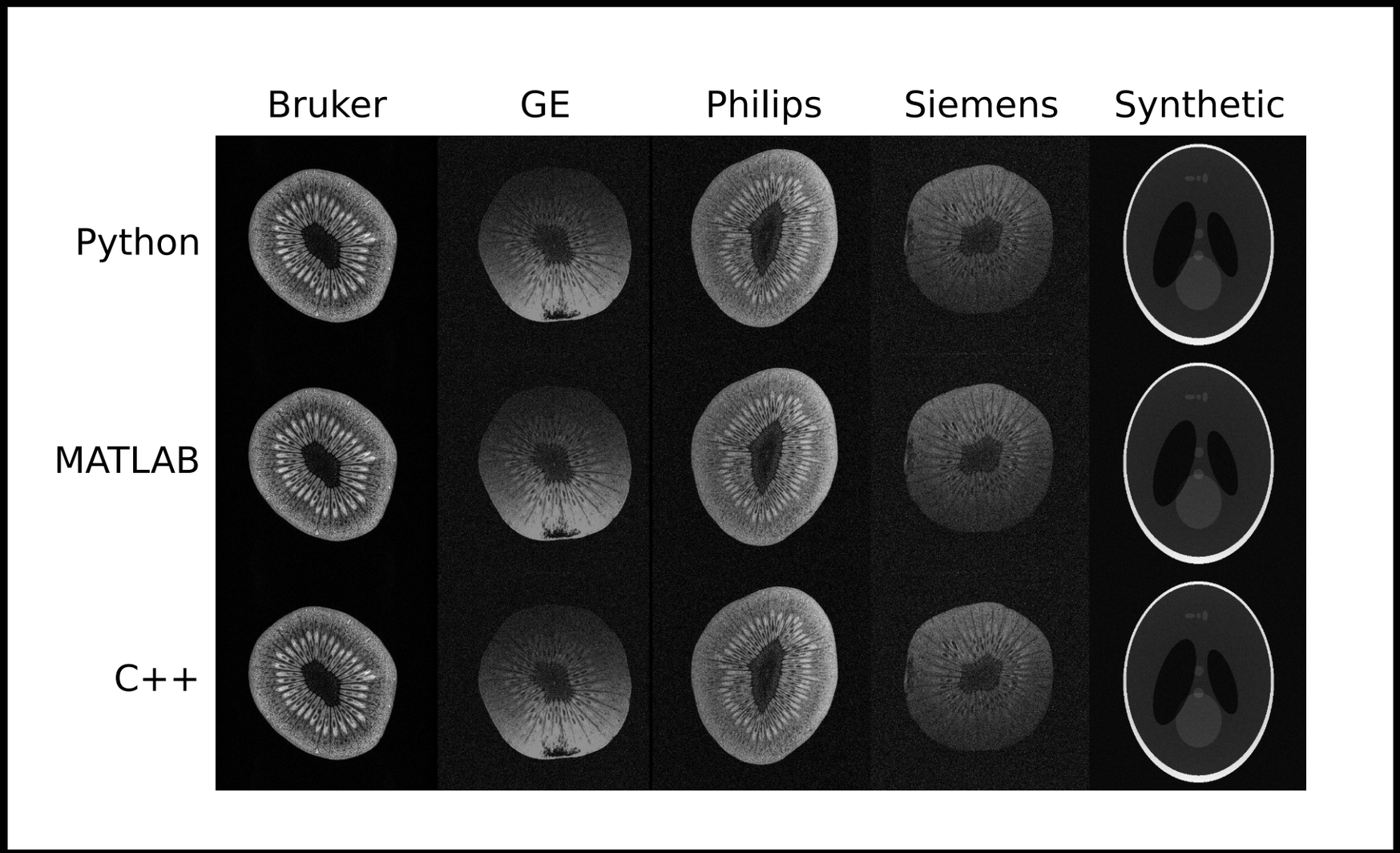
ISMRMRD


Encoded Space
matrix_size/2
matrix_size/2
0
8
15
0
16
31
Center Sample
iFFT
Reconstructed Space
0
8
15
0
16
31
Cropped Field of View
XML Header
0
5
11
<encoding>
<trajectory>cartesian</trajectory>
<encodedSpace>
<matrixSize>
<x>32</x><y>16</y><z>1</z>
</matrixSize>
<fieldOfView_mm>
<x>600</x><y>300</y><z>10</z>
</fieldOfView_mm>
</encodedSpace>
<reconSpace>
<matrixSize>
<x>16</x><y>16</y><z>1</z>
</matrixSize>
<fieldOfView_mm>
<x>300</x><y>300</y><z>10</z>
</fieldOfView_mm>
</reconSpace>
<encodingLimits>
<kspace_encoding_step_1>
<minimum>0</minimum>
<maximum>11</maximum>
5
</kspace_encoding_step_1>
</encodingLimits>
</encoding>

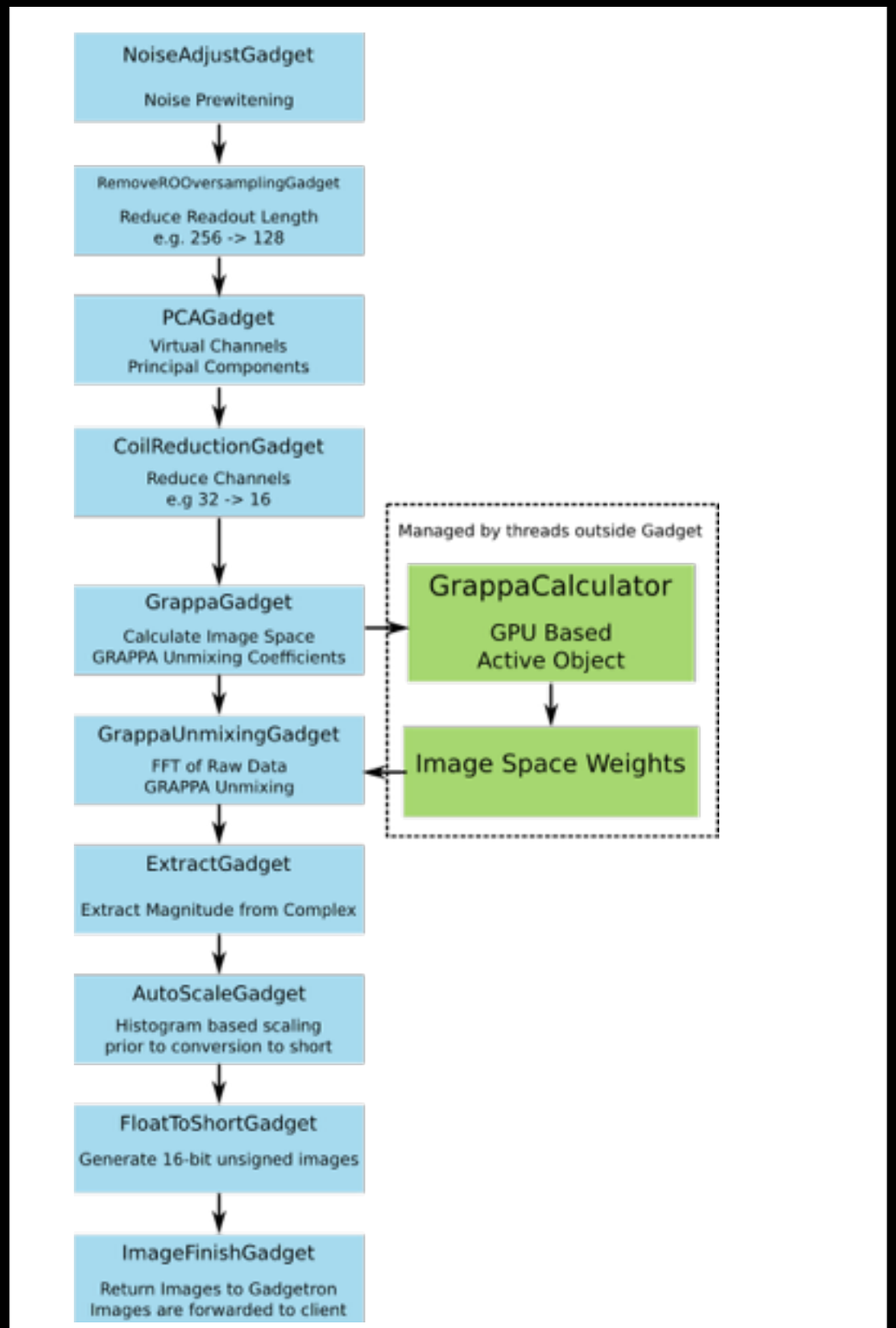
Conversion from vendor data



Multi Vendor Example

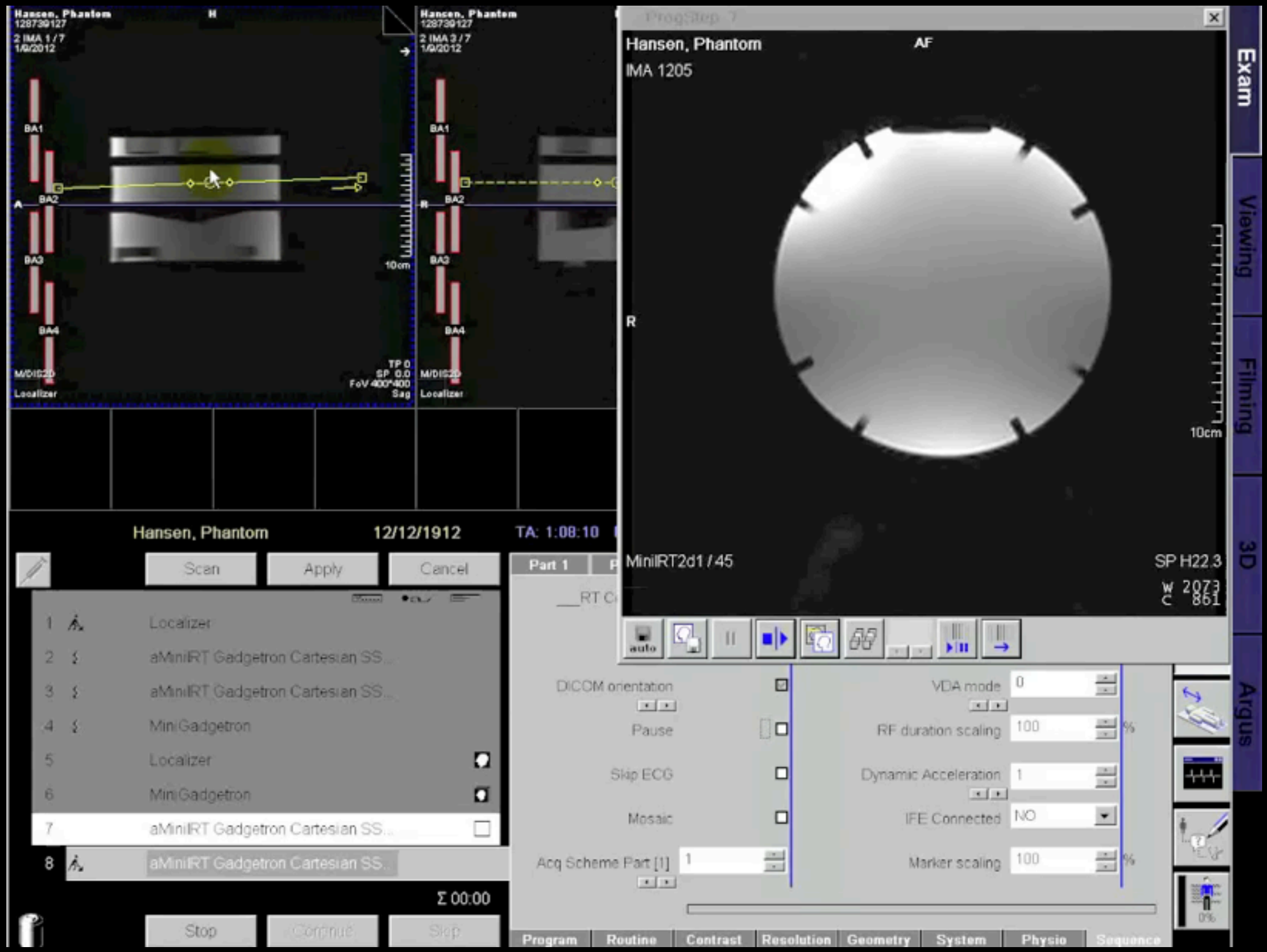


Example - Cartesian GRAPPA

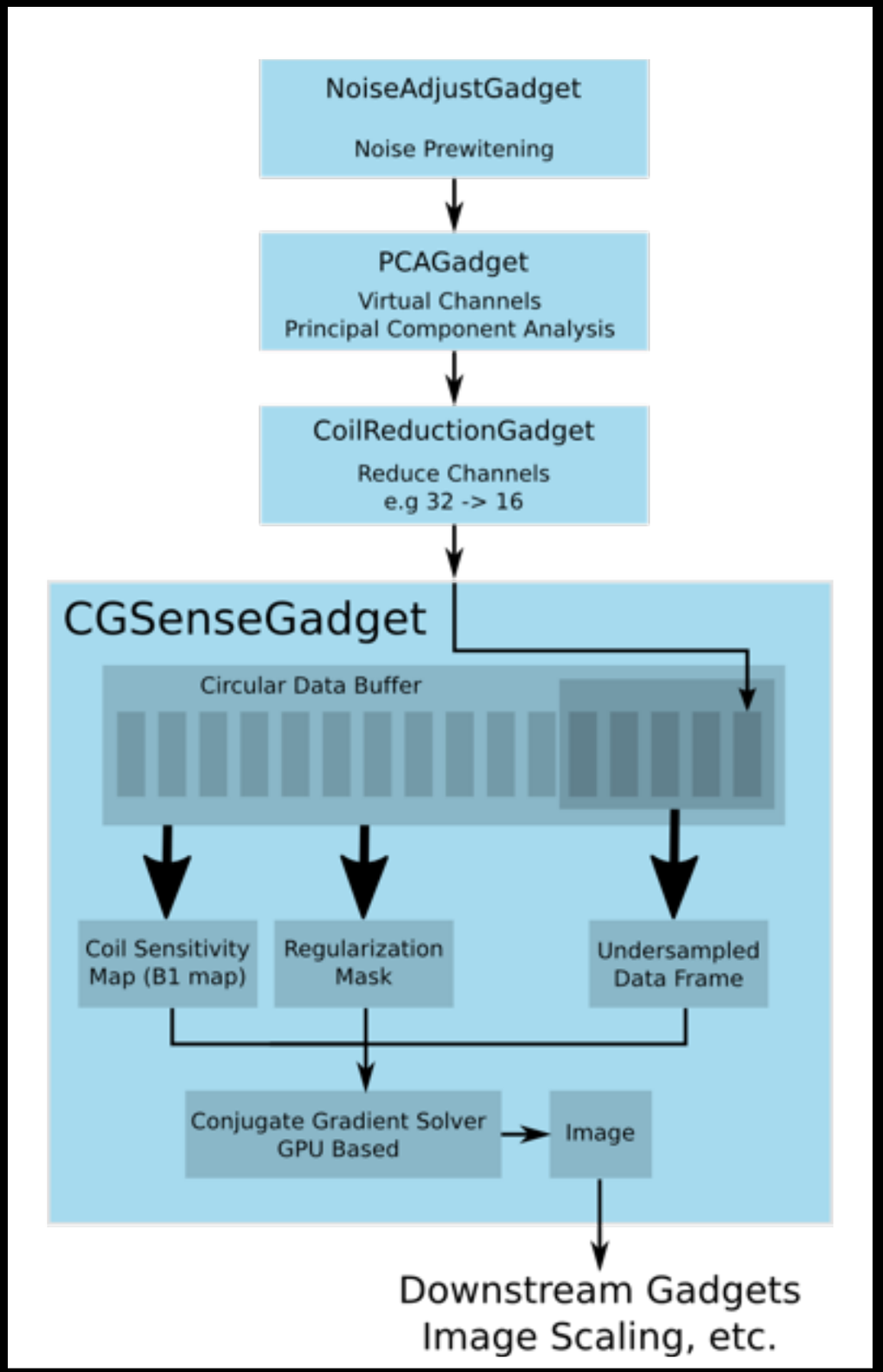


- High-throughput GRAPPA
- Designed for multi-slice 2D real-time imaging (interventional)
- GRAPPA coefficients calculated on GPU

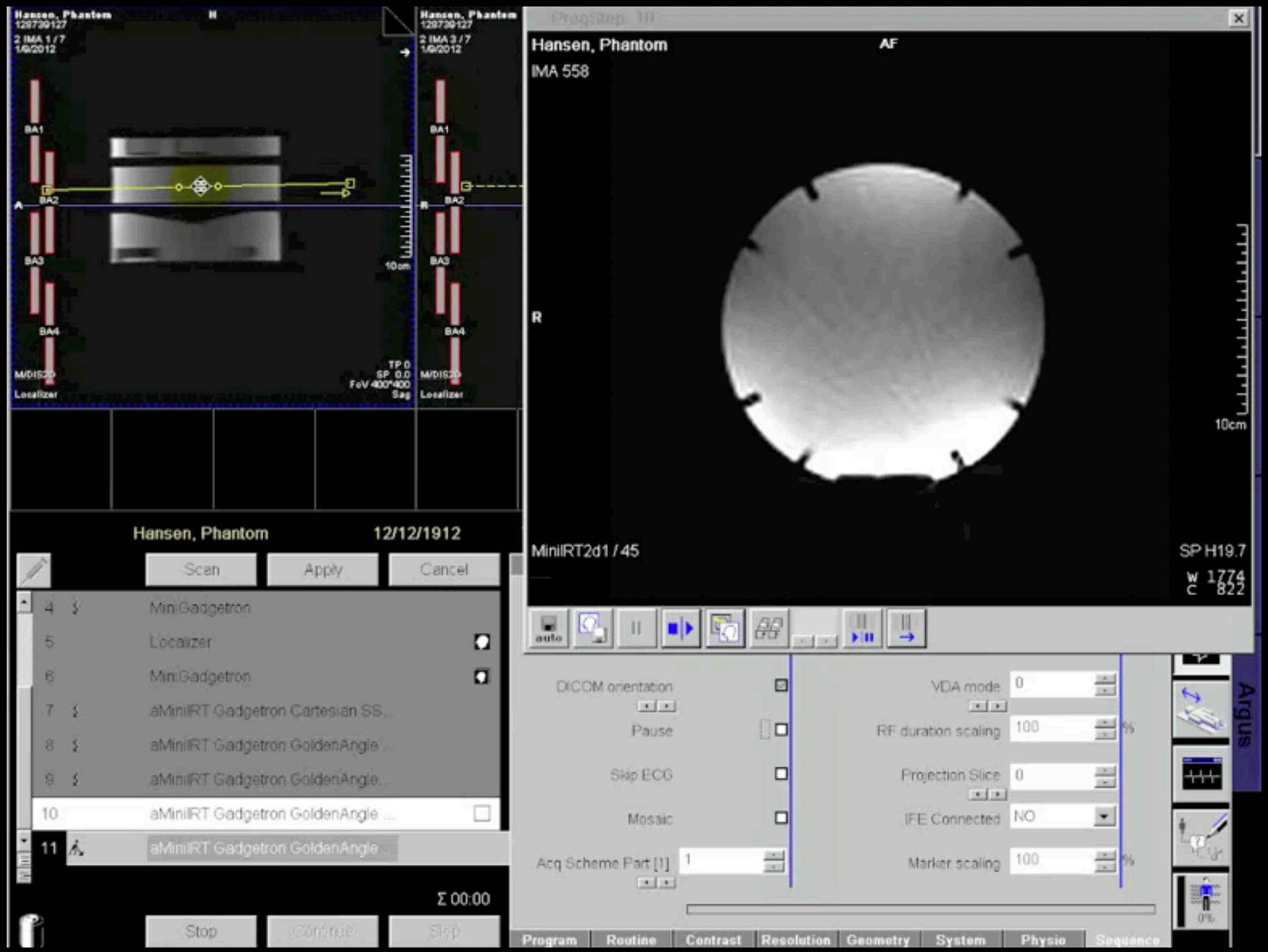
Cartesian GRAPPA



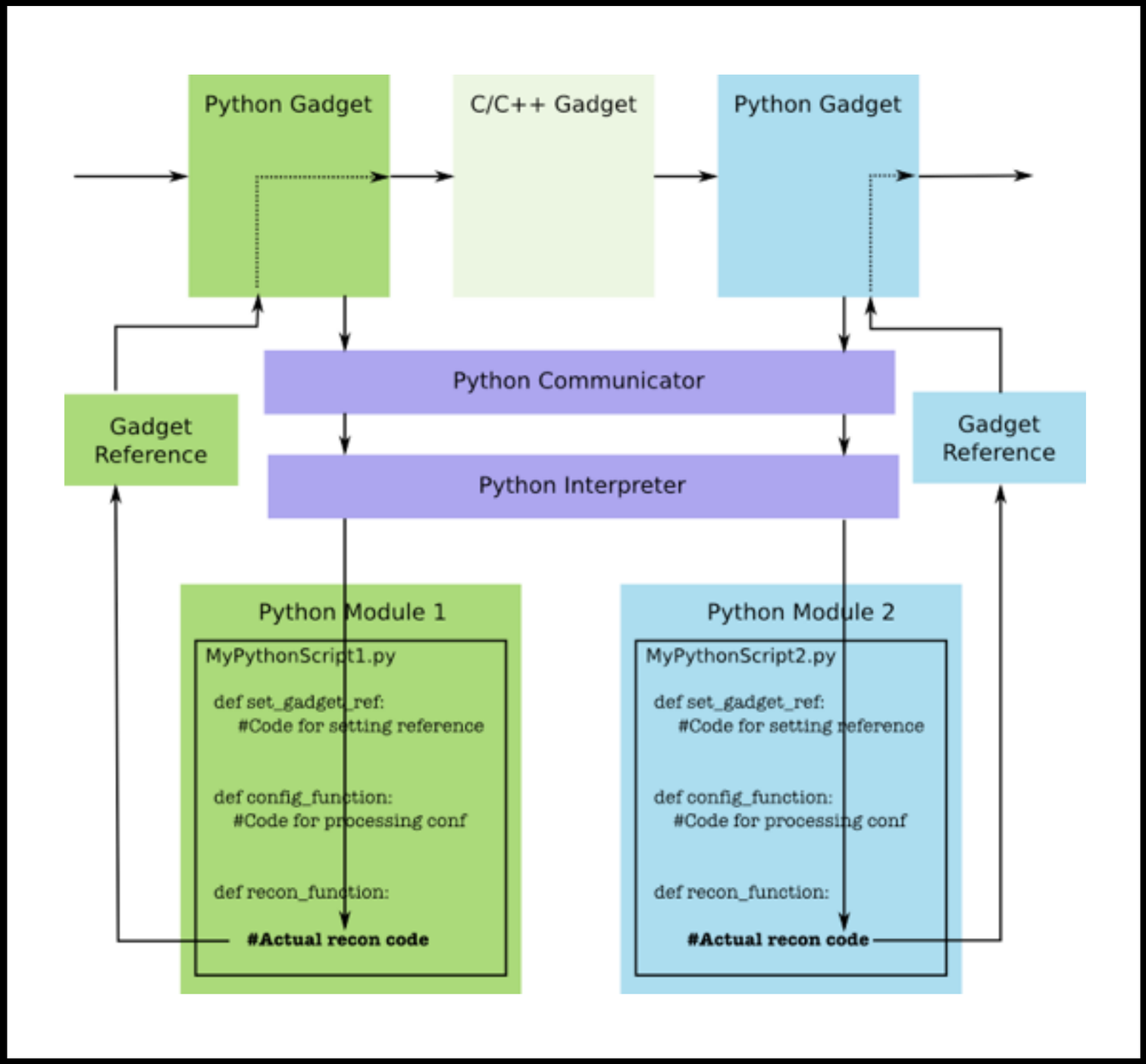
Non-Cartesian SENSE



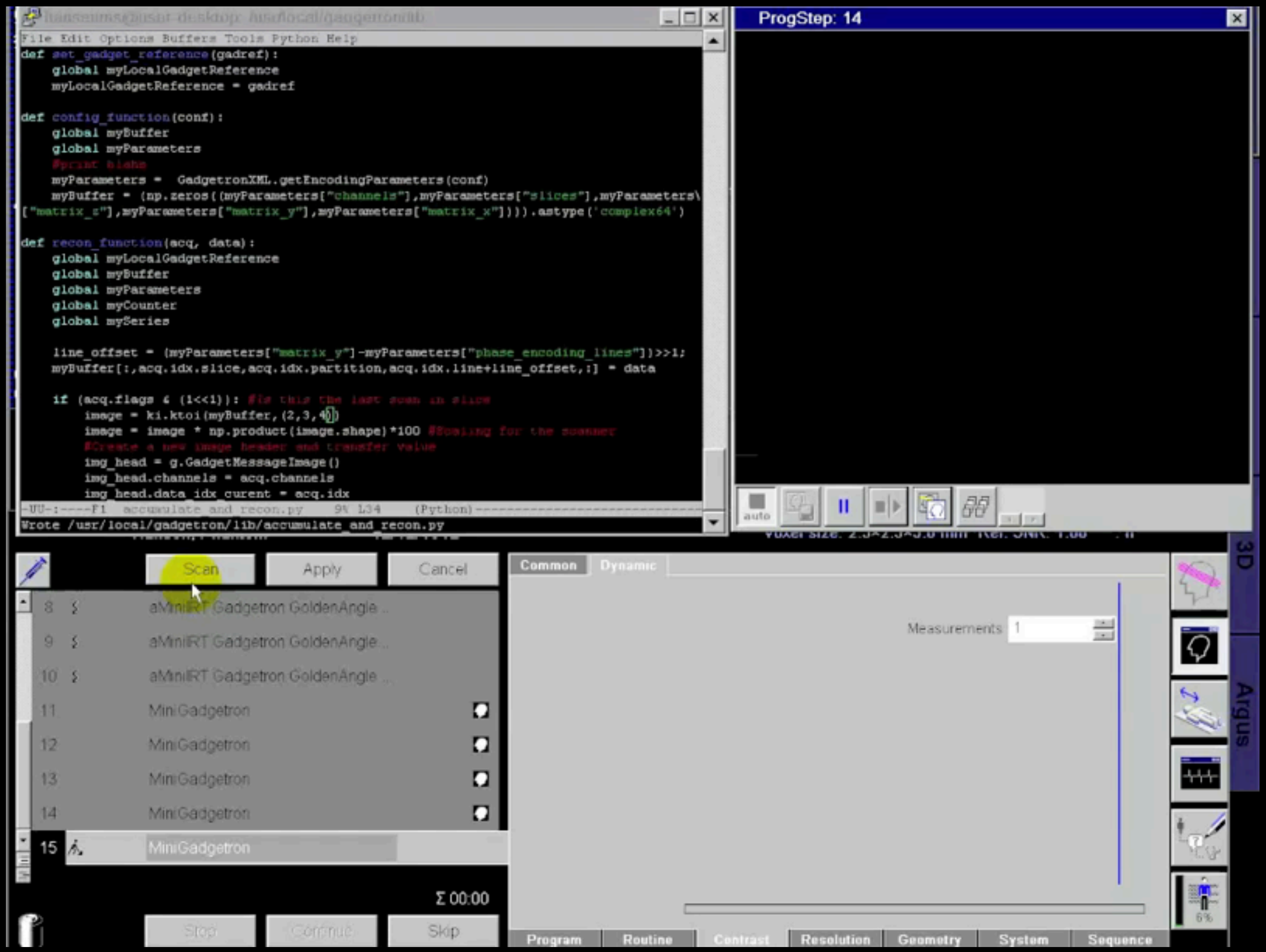
Golden Angle Radial



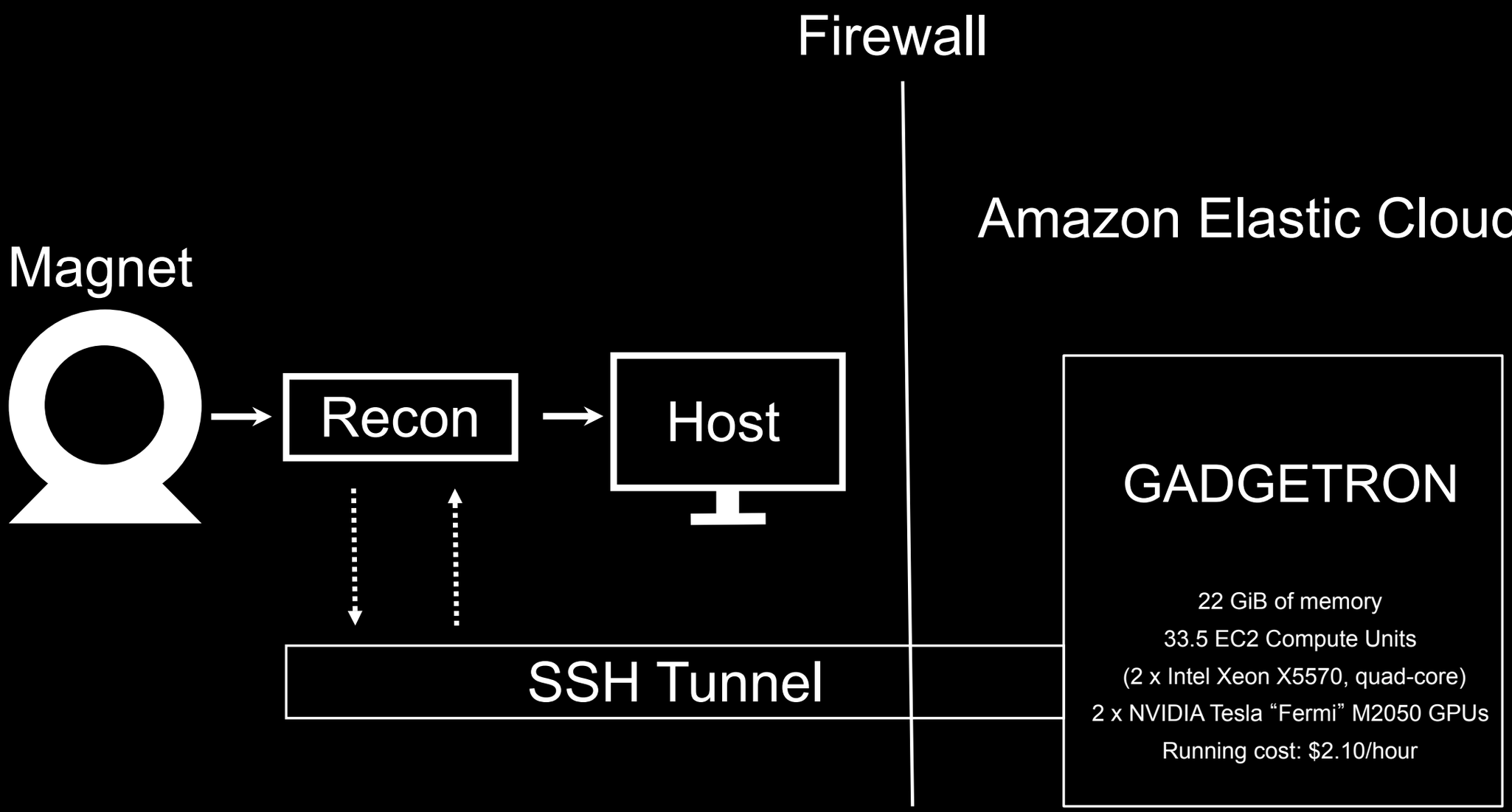
Python Support



Online Python Editing

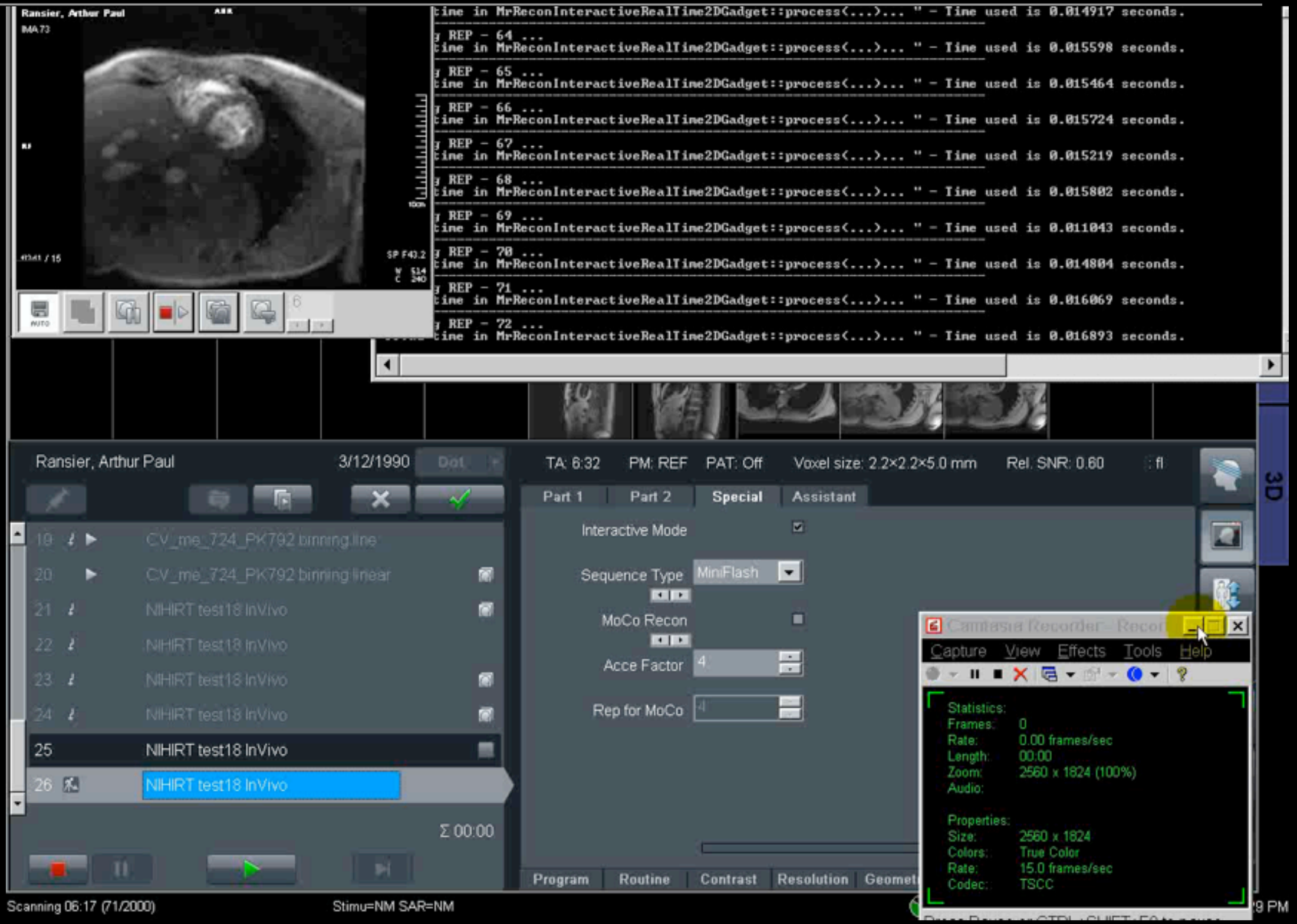


Gadgetron in the Cloud

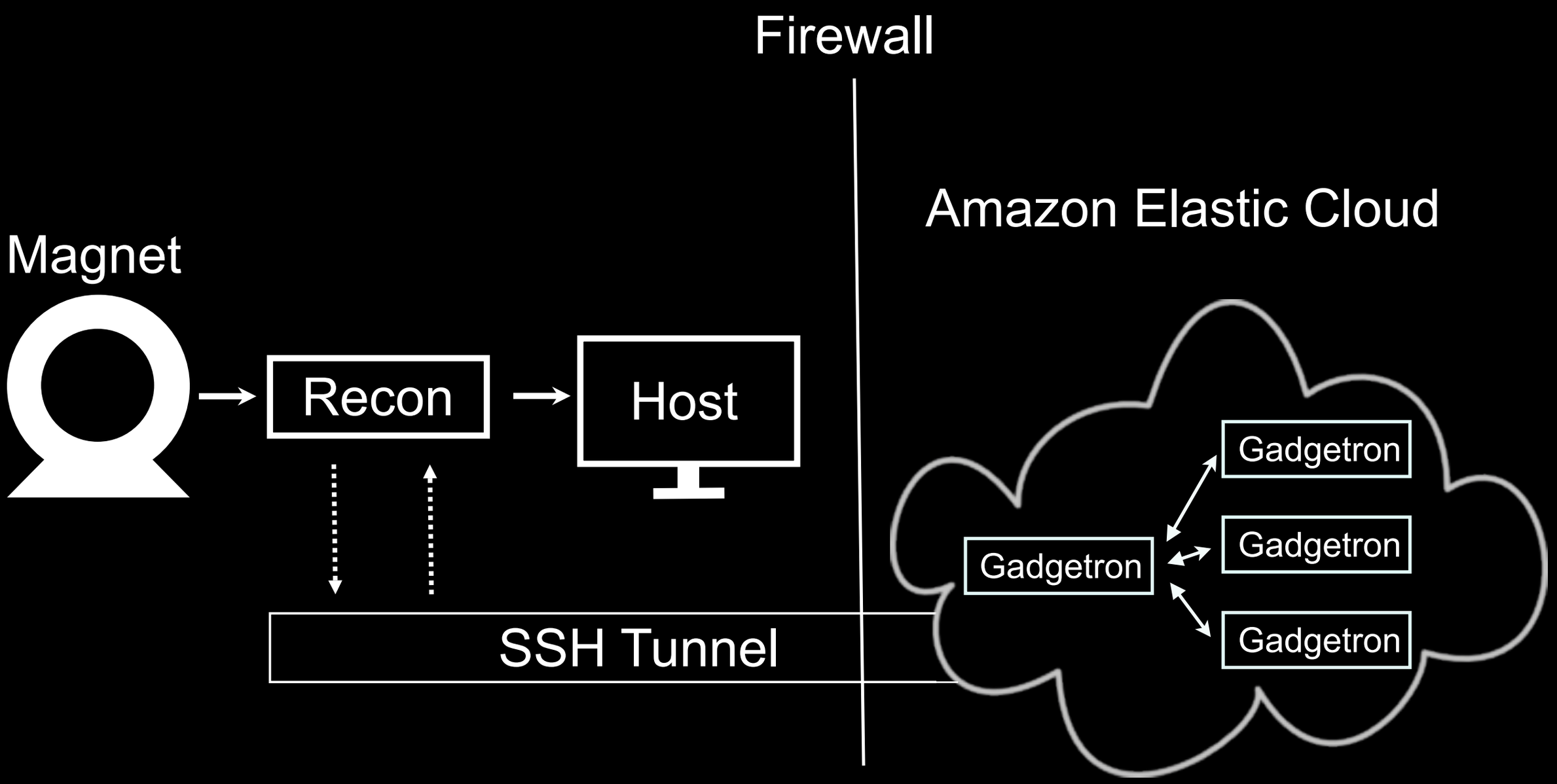


Enables easy deployment of Gadgetron at multiple locations without installing and managing hardware

Reconstruction in the Cloud

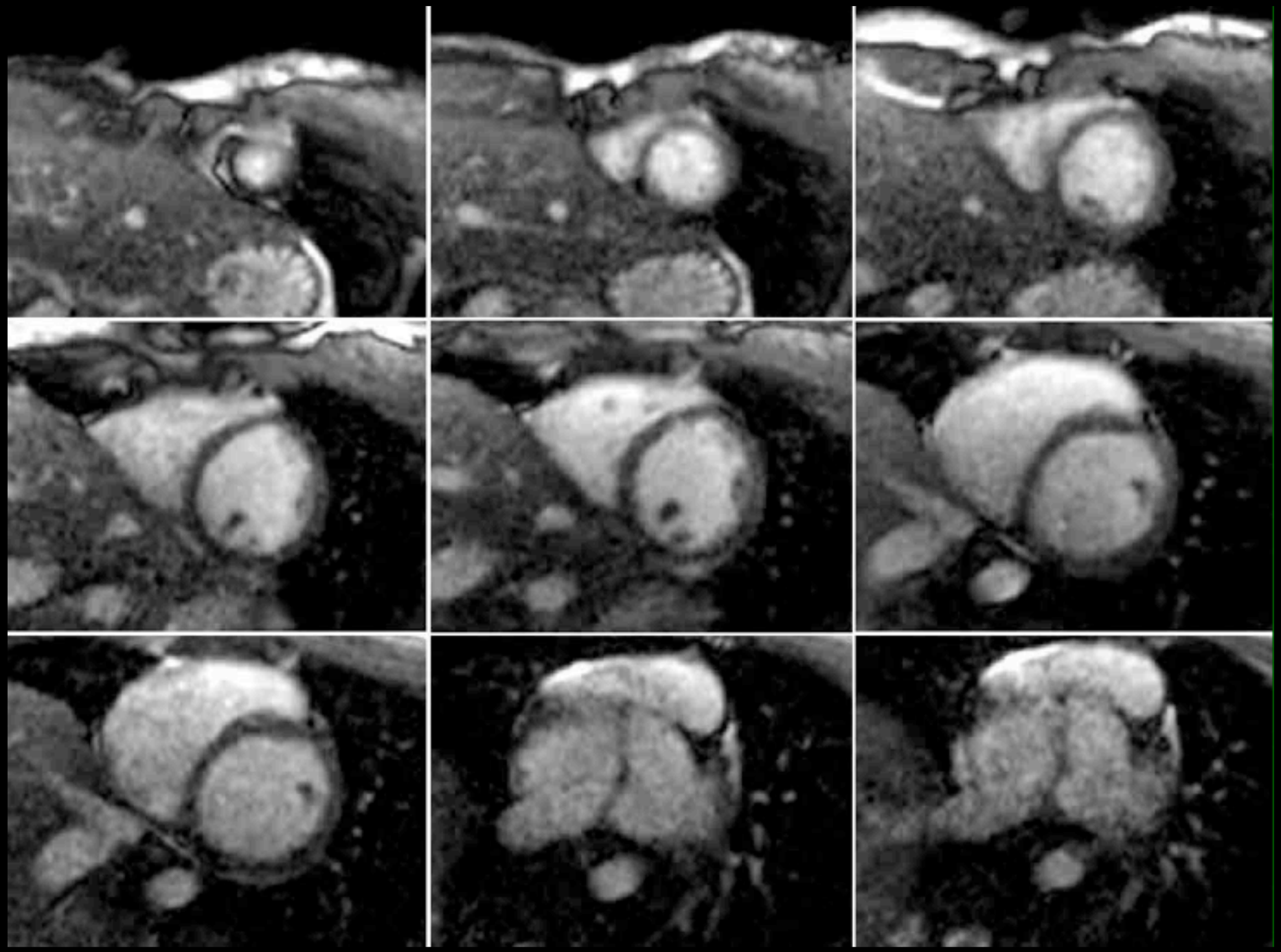


Gadgetron in the Cloud



Enables easy deployment of Gadgetron at multiple locations without installing and managing hardware

Cloud Based Gadgetron Reconstruction

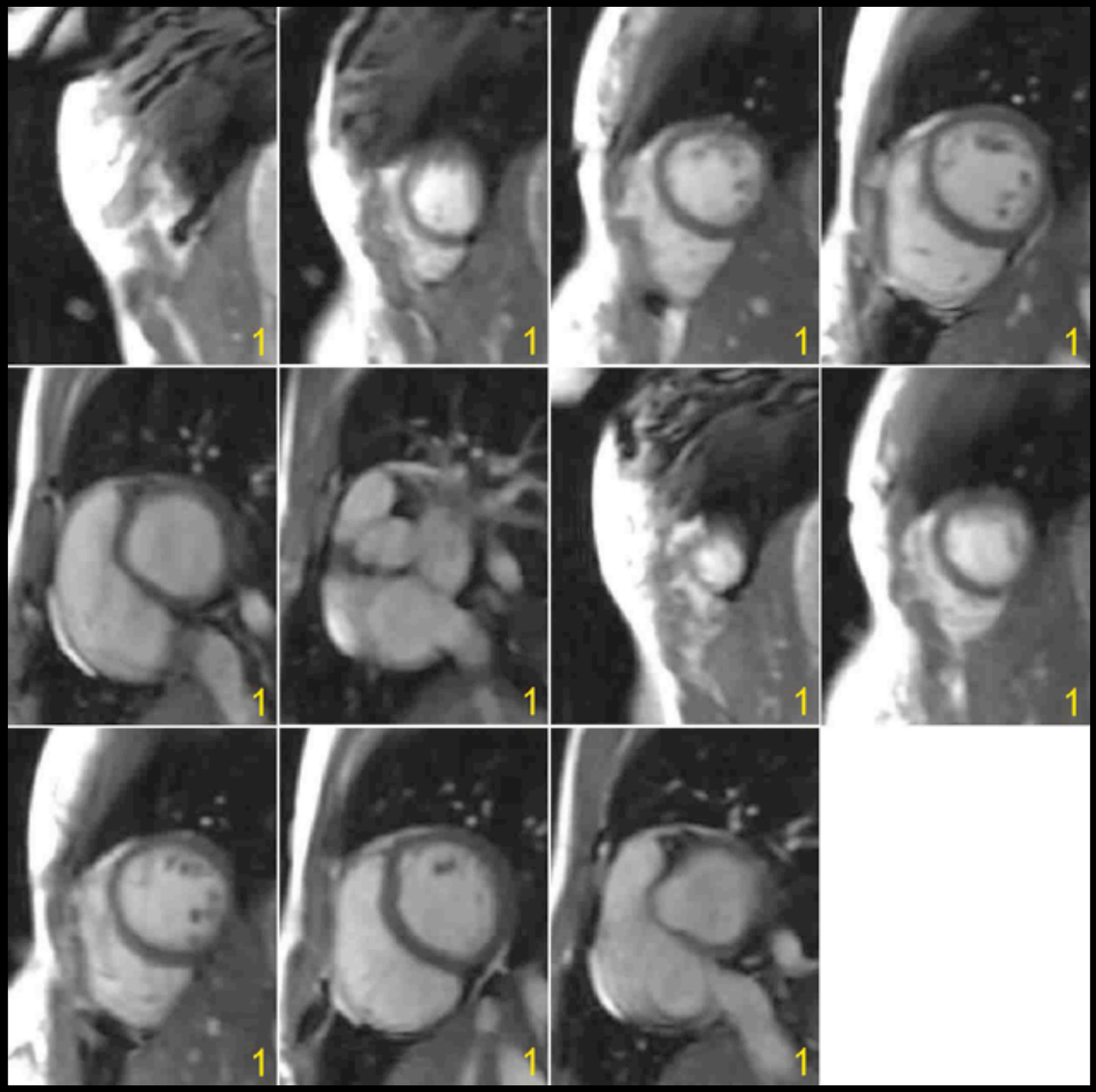


192x100 matrix
Rate 5
~50ms/frame

Acquired and reconstructed in <1min

Hui Xue, NHLBI, NIH

Real-Time Non-Linear



1.5T, FISP

Real-time cine

30ms temporal resolution

192x100 matrix

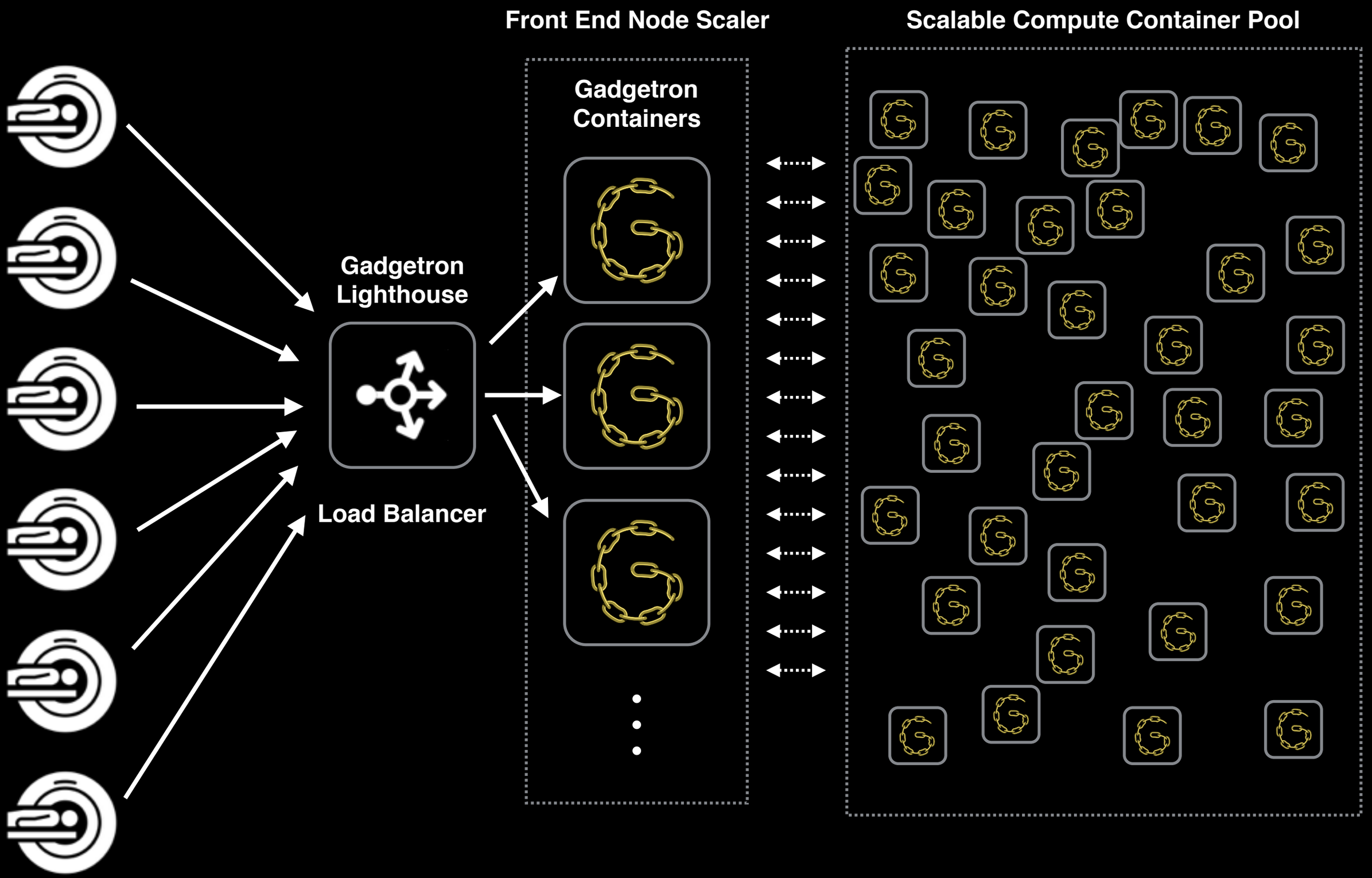
1.5s acquisition per slice, 11 slices

Entire LV is covered in <30s, complete free-breathing acquisition

Reconstruction is performed on a Gadgetron based cloud with 7 nodes

Computation is completed within ~1min after the end of acquisition

Gadgetron Cloud Strategy



Thank You

Andrew Arai

Bob Balaban

Adrienne Campbell-Washburn

Russell Cross

Anthony Faranesh

David Hansen

Souheil Inati

Peter Kellman

Robert Lederman

Joseph Naegele

Kendall O’Brien

Laura Olivieri

Kanishka Ratnayaka

Toby Rogers

Thomas Sorensen

Hui Xue

National Heart, Lung, and Blood Institute

michael.hansen@nih.gov

http://gadgetron.github.io

http://ismrmrd.github.io

National Heart, Lung,
and Blood Institute