Basics of Pulseq

Maxim Zaitsev

Division of Medical Physics, Dept. of Radiology,
University Medical Center Freiburg, Germany

Nov 15, 2023

What is *Pulseq*?

- *Pulseq* is a language to describe MR pulse sequences

- *Pulseq* sequences are fixed successions of RF and gradient pulses and ADC events

- *Pulseq* is the software to generate such pulse sequence descriptions

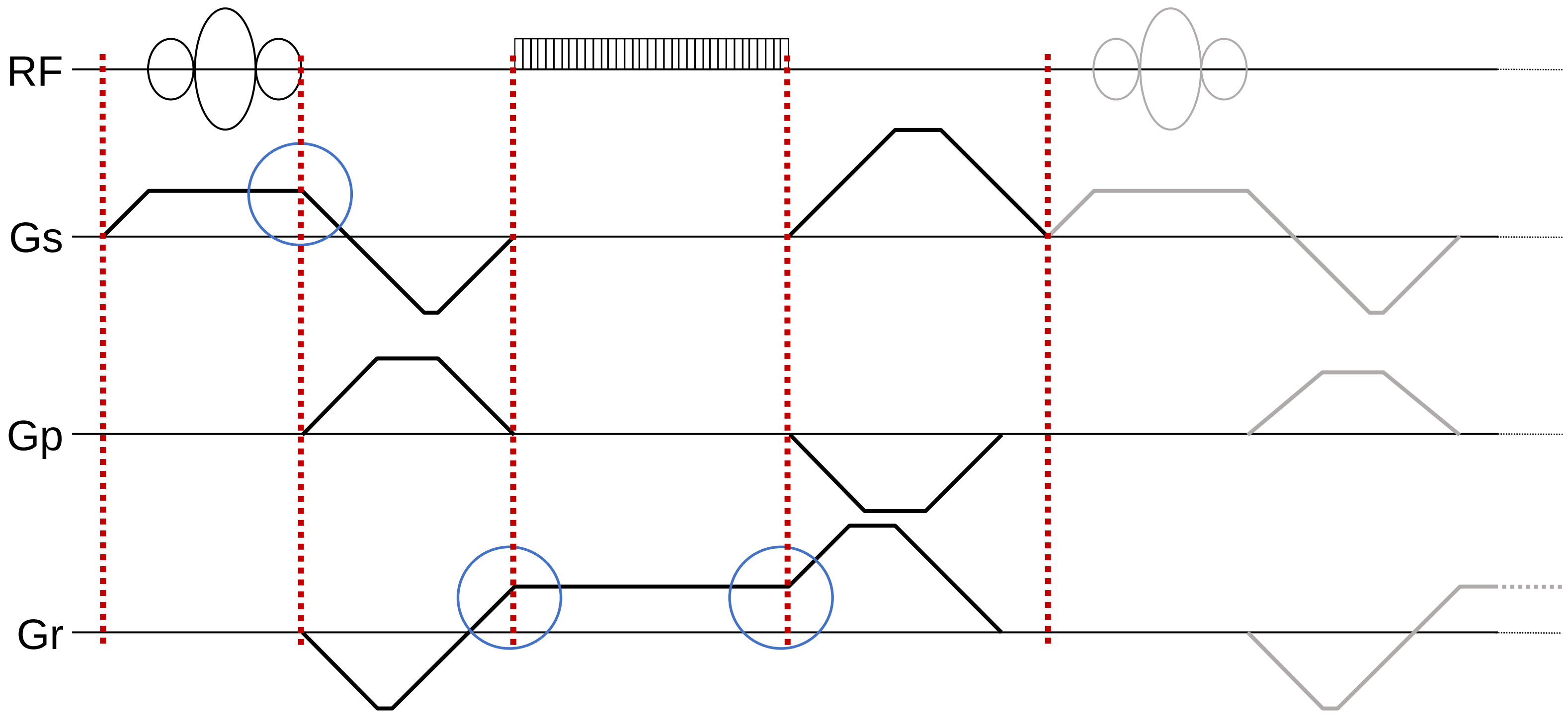
- *Pulseq* scripts can re-generate *Pulseq* sequences to accommodate user input

Pulseq ecosystem includes sequences, software to generate them and software and hardware to consume them

Pulseq Philosophy

- Minimize effort for implementation and support on hardware
  - Lean sequence-to-hardware interface
- Remove the initial threshold in sequence programming
  - Turn simple things really simple
- Make researcher-oriented features easily accessible
  - Arbitrary gradients, arbitrary RF, flexible reordering, X-nuclei, …
- Prevent typical sources of (human) errors
  - Avoid timing errors with “overlapping” gradients
  - Make data flag and counter setting optional/unnecessary
- **Promote open-source thinking, sharing and exchange!**

Pulse sequence definition in *Pulseq*



- Block 1:
  gradient and RF
- Block 2:
  only gradients
- Block 3:
  gradient and ADC
- Block 4:
  only gradients
- Block 5:
  gradient and RF …

- Sequence is a concatenation of non-overlapping blocks
- Gradients do not have to start or end at 0 at the block boundaries

see http://pulseq.github.io/specification.pdf for more details

Pulseq block concept in detail

- Each block may contain following events:
  - One optional gradient pulse per axis
  - One optional RF pulse
  - One optional ADC event
- Individual events may define own start delays
- All events in the block overlap in time
- Duration of the block is defined by the longest event
  - Matlab/Python toolboxes use “dummy” delay objects to make blocks longer
- Explicit sequence description
  - No loops, no dependent parameters

How to design a sequence in Pulseq

conceptual design steps

- Step 1: spilt the time axis into blocks
- Step 2: assign events to the blocks

practical implementation steps

- Step 3: create/calculate all events
- Step 4: populate the blocks and add them to the sequence

validation steps

- Step 5: check timing, verify k-space trajectory, check hardware and PNS limits, mechanical resonances, etc…

High-level programming environments

- Matlab *Pulseq* toolbox
- Python *PyPulseq* toolbox

- Further options
  - TOPPE is primarily targeted at GE but can import and export *pulseq* files (Jon-Fredrik Nielsen will talk about it today)
  - GammaStar can export *pulseq* files
  - JEMRIS Bloch simulator can export *pulseq* files
  - CoreMRI Bloch simulator can export *pulseq* files
  - …

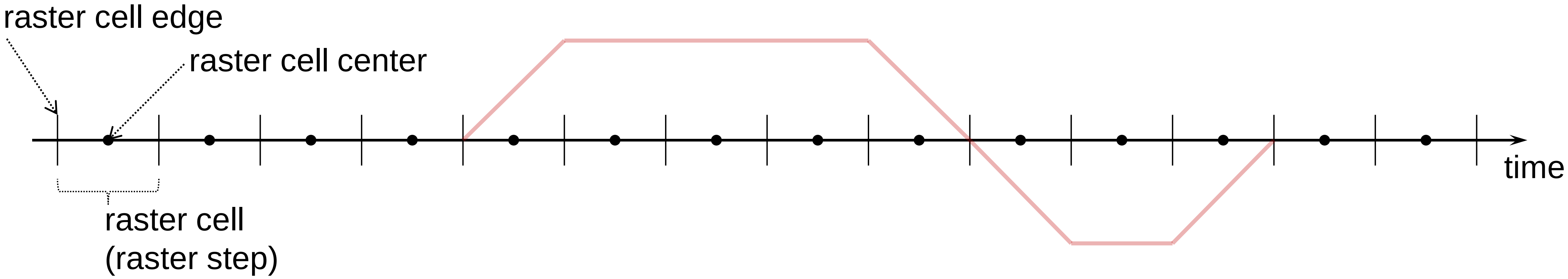
Matlab *Pulseq* workflow

```
system = mr.opts('MaxGrad',30,'GradUnit','mT/m',...
    'MaxSlew',170,'SlewUnit','T/m/s');
seq=mr.Sequence(system);

fov = 220e-3; Nx=64; Ny=64; TE = 10e-3; TR = 20e-3;

[rf, gz] = mr.makeSincPulse(15*pi/180,system,'Duration',4e-3,...
    'SliceThickness',5e-3,'apodization',0.5,'timeBwProduct',4);

gx = mr.makeTrapezoid('x',system,'FlatArea',Nx/fov,'FlatTime',6.4e-3);
adc = mr.makeAdc(Nx,'Duration',gx.flatTime,'Delay',gx.riseTime);
gxPre = mr.makeTrapezoid('x',system,'Area',-gx.area/2,'Duration',2e-3);
gzReph = mr.makeTrapezoid('z',system,'Area',-gz.area/2,'Duration',2e-3);
phaseAreas = ((0:Ny-1)-Ny/2)*1/fov;

delayTE = TE - mr.calcDuration(gxPre) - mr.calcDuration(rf)/2 ...
    - mr.calcDuration(gx)/2;
delayTR = TR - mr.calcDuration(gxPre) - mr.calcDuration(rf) ...
    - mr.calcDuration(gx) - delayTE;
delay1 = mr.makeDelay(delayTE);
delay2 = mr.makeDelay(delayTR);

for i=1:Ny
    seq.addBlock(rf,gz);
    gyPre = mr.makeTrapezoid('y',system,'Area',phaseAreas(i),...
                            'Duration',2e-3);
    seq.addBlock(gxPre,gyPre,gzReph);
    seq.addBlock(delay1);
    seq.addBlock(gx,adc);
    seq.addBlock(delay2)
end

seq.write('gre.seq')
```

a runnable gradient echo sequence code (similar to Siemens' example miniFlash)

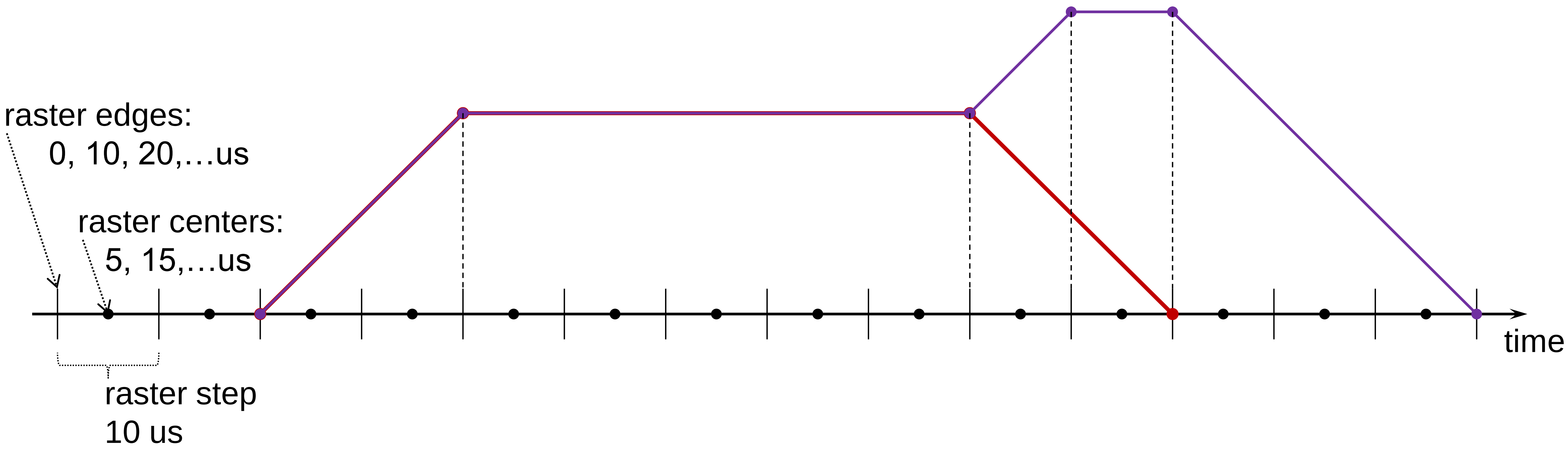
- Define the system properties
- Define high-level parameters (convenience)
- Define pulses and ADC objects used in the sequence
- Calculate the delays and reordering tables
- Loop and define sequence blocks
- Duration of each block is defined by the duration of the longest event
- *Copy '*.seq' to the scanner and run it!*

Shapes and Raster Times *Pulseq*



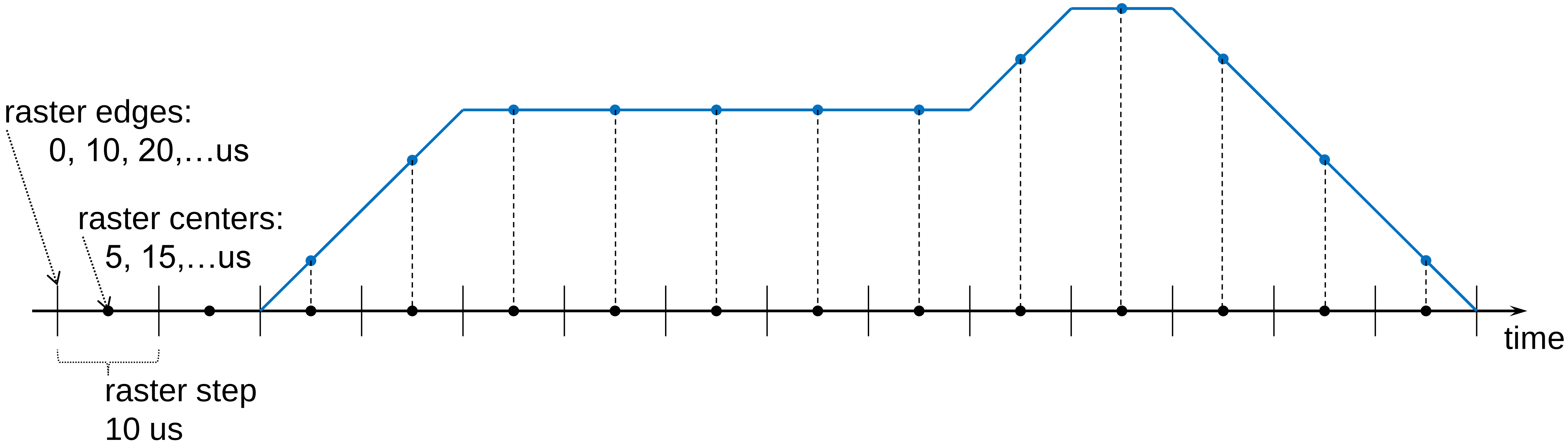
- Precise control of arbitrary gradient and RF waveforms
- Pulseq defines four types of raster times
  - adcRasterTime, rfRasterTime, gradRasterTime, blockDurationRaster
- Raster 'thinking' is probably one of the most demanding concepts in the practical pulse sequence programming
- Raster cells, edges and centers

Gradient Raster and Shapes



- Gradient raster (10 us on Siemens)
- Trapezoid and extended trapezoids: vertices on raster edges

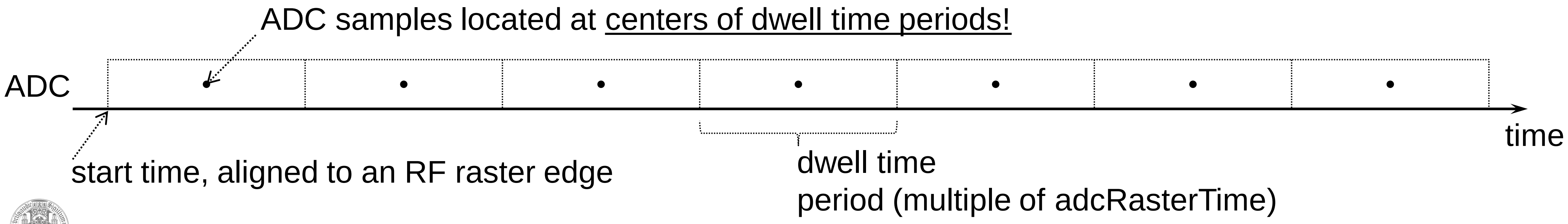
Gradient Raster and Shapes



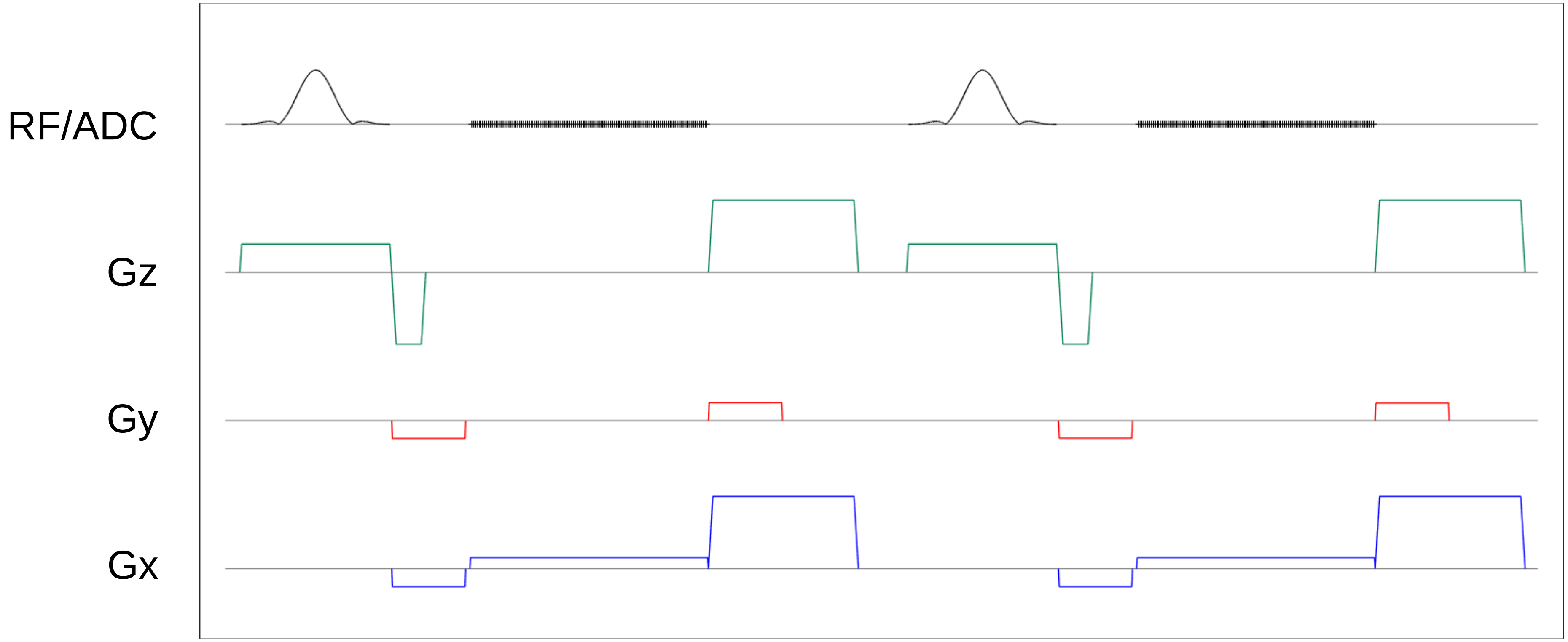
- Gradient raster (10 us on Siemens)
- Trapezoid and extended trapezoids: nodes on raster edges
- Sampled (arbitrary) gradients: <u>samples on raster centers</u>

RF and ADC Raster Times

- RF objects can be either regularly sampled or defined by vertices
  - mr.makeSincPulse() defines a regularly sampled pulse
  - mr.makeBlockPulse() uses a shape with two points: (0,1) and (dur,1)
- rfRasterTime on Siemens: 1us
  - Dwell time for regularly-sampled pulses: multiple of rfRasterTime
- ADC start time must be aligned to rfRasterTime
  - ADC dwell time: multiple of adcRasterTime (100ns on Siemens)

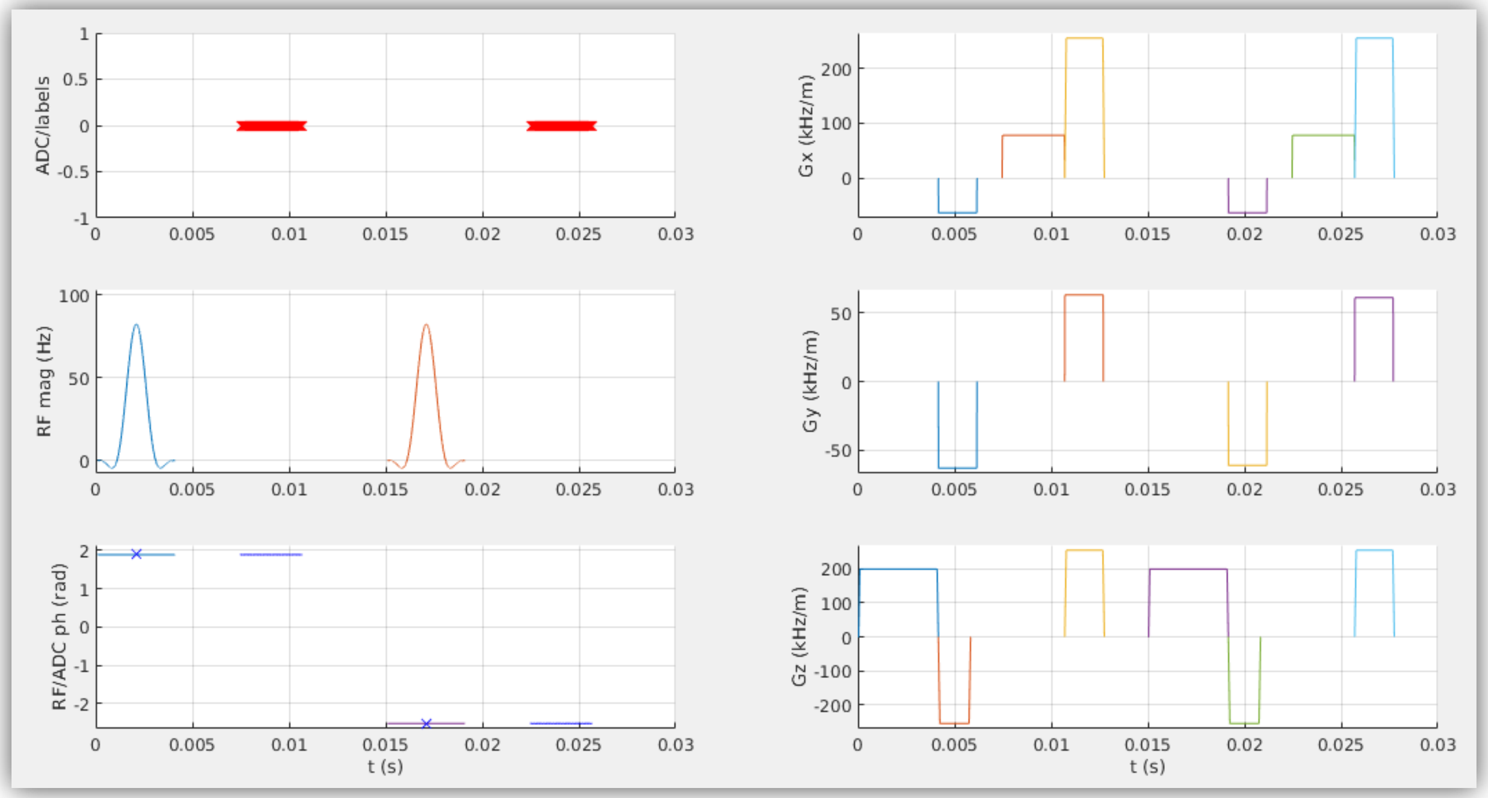


Example 1: simple gradient echo



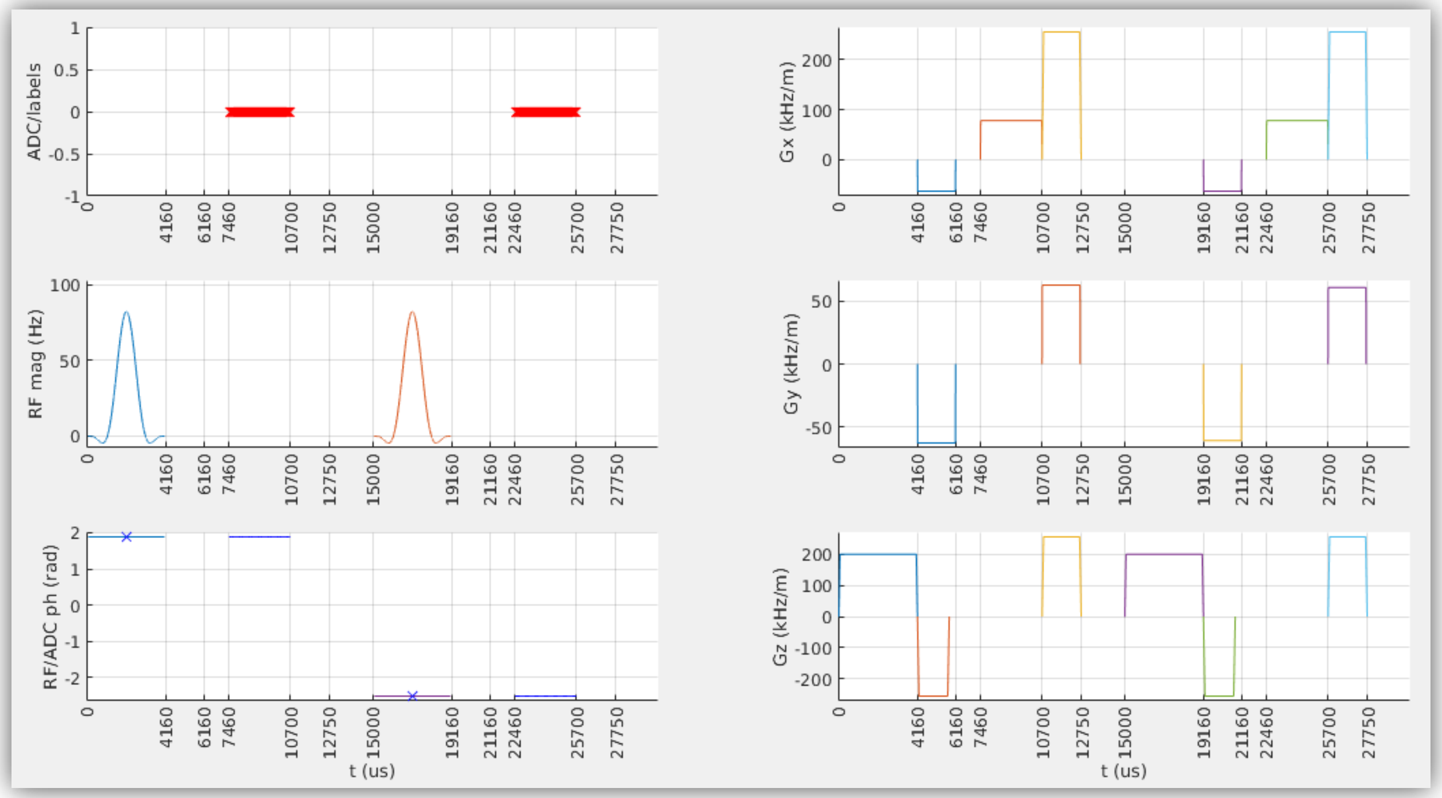
- No overlapping gradient ramps on different axes
- Events are clearly separated

Example 1: basic sequence display options



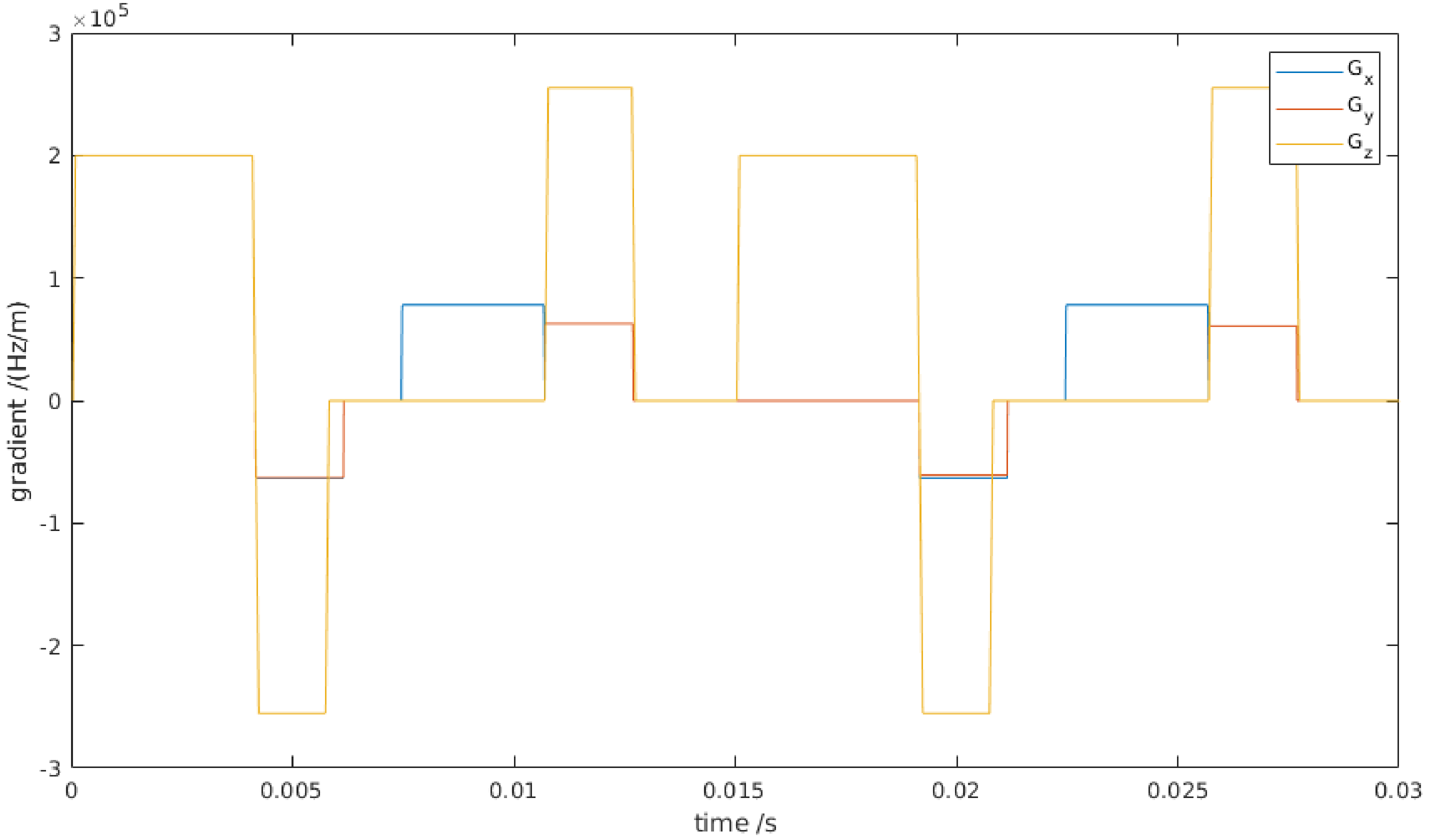
- Pulseq 6-panel plot: seq.plot()
  - ADC, RF magnitude, RF phase, Gx, Gy, Gz
  - Each event plotted in its own color

Example 1: display block structure



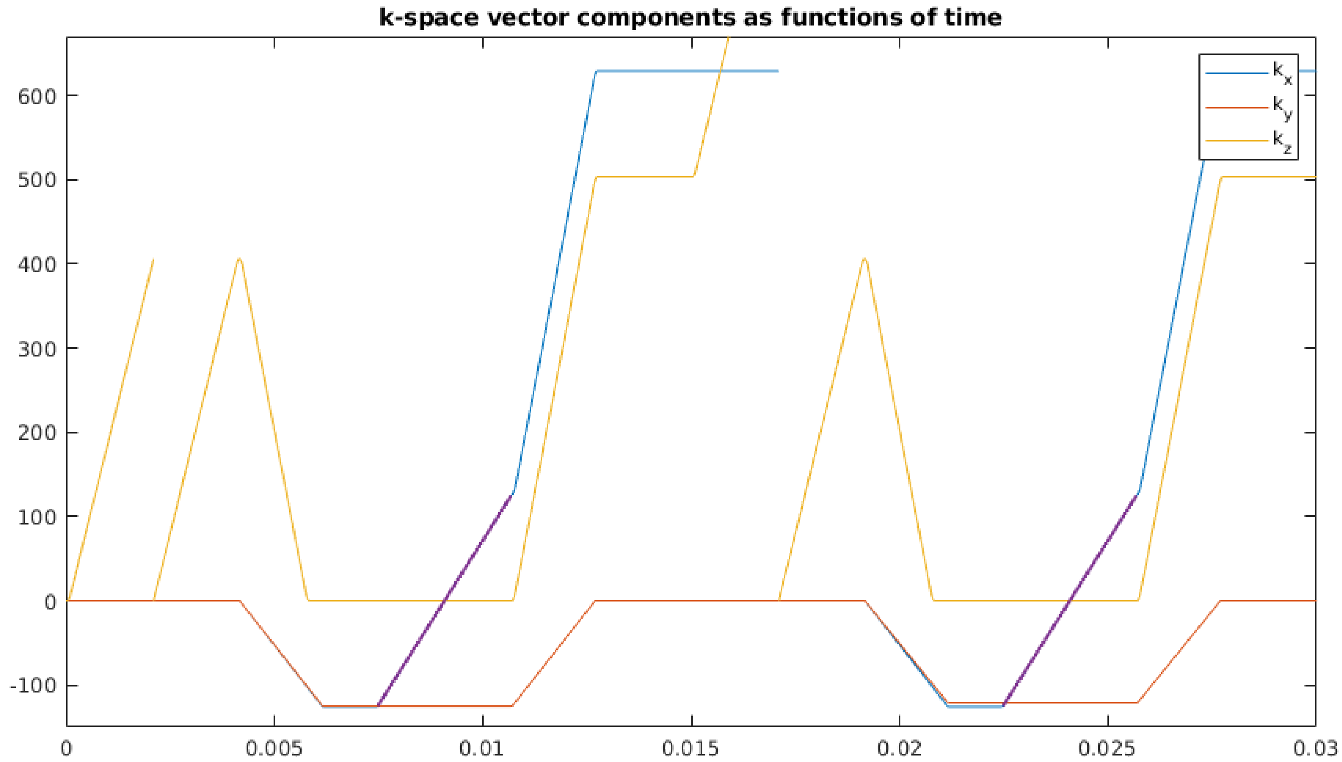
- Advantageous to separate PE & PR gradients into different blocks
- To visualize block structure:
  - seq.plot('showBlocks',true,'timeDisp','us');

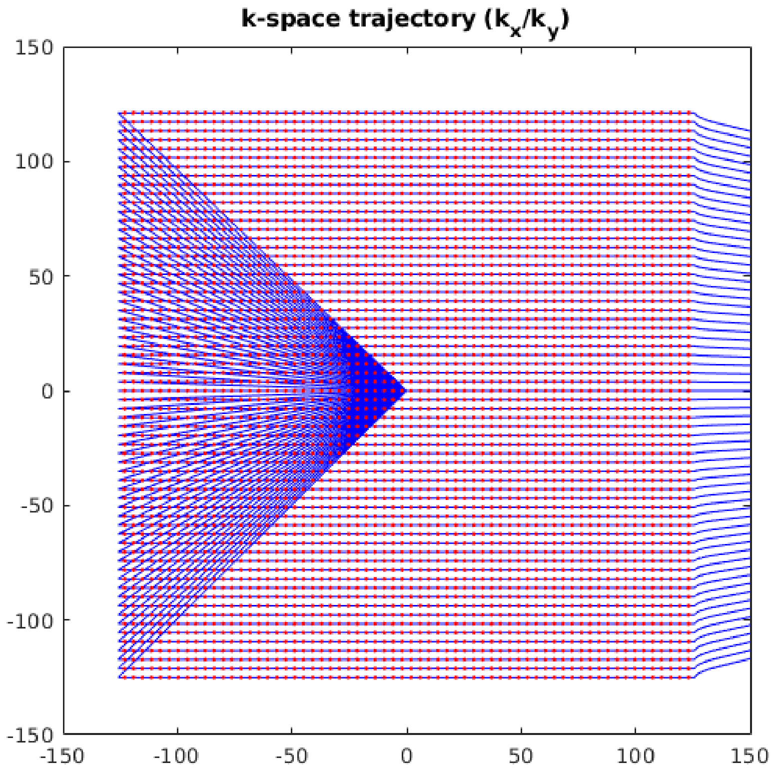
Plot gradient waveforms



- Plot entire waveforms for all axes
    - Native gradient unit in *Pulseq*: Hz/m

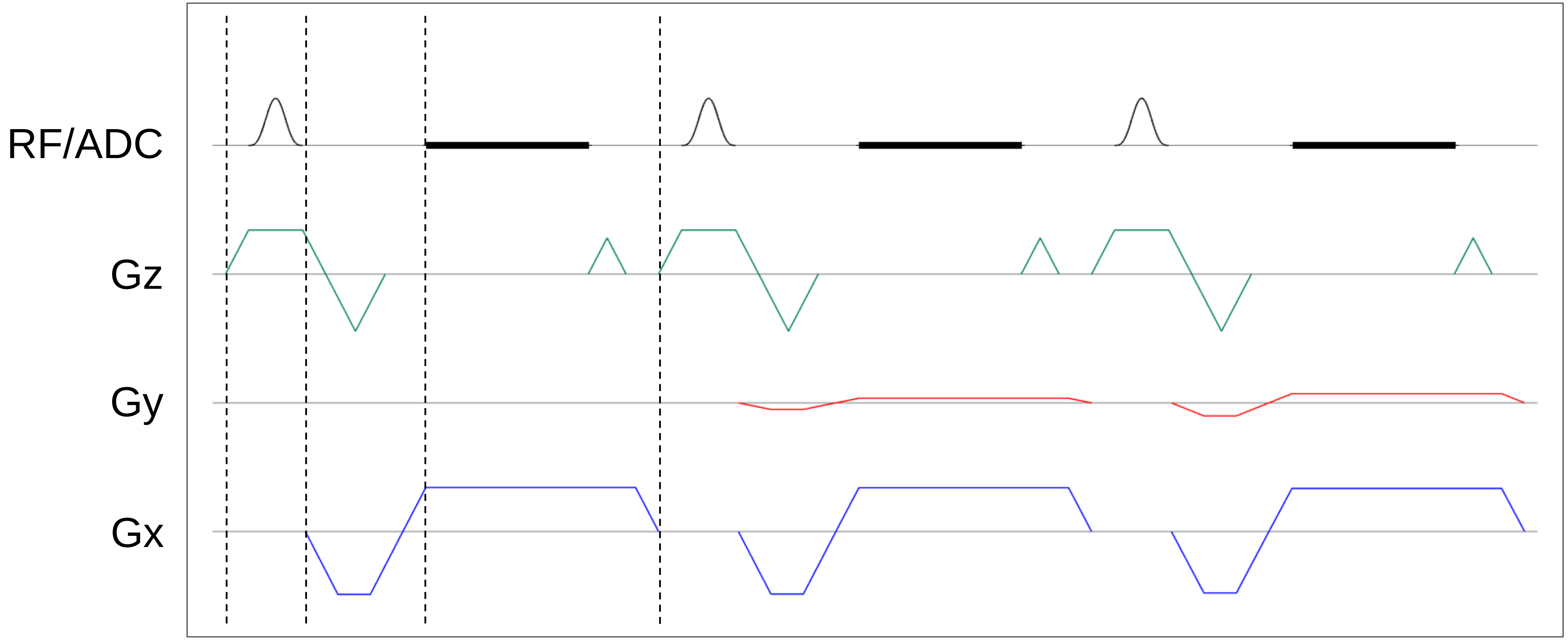
Basic sequence display options: k-space





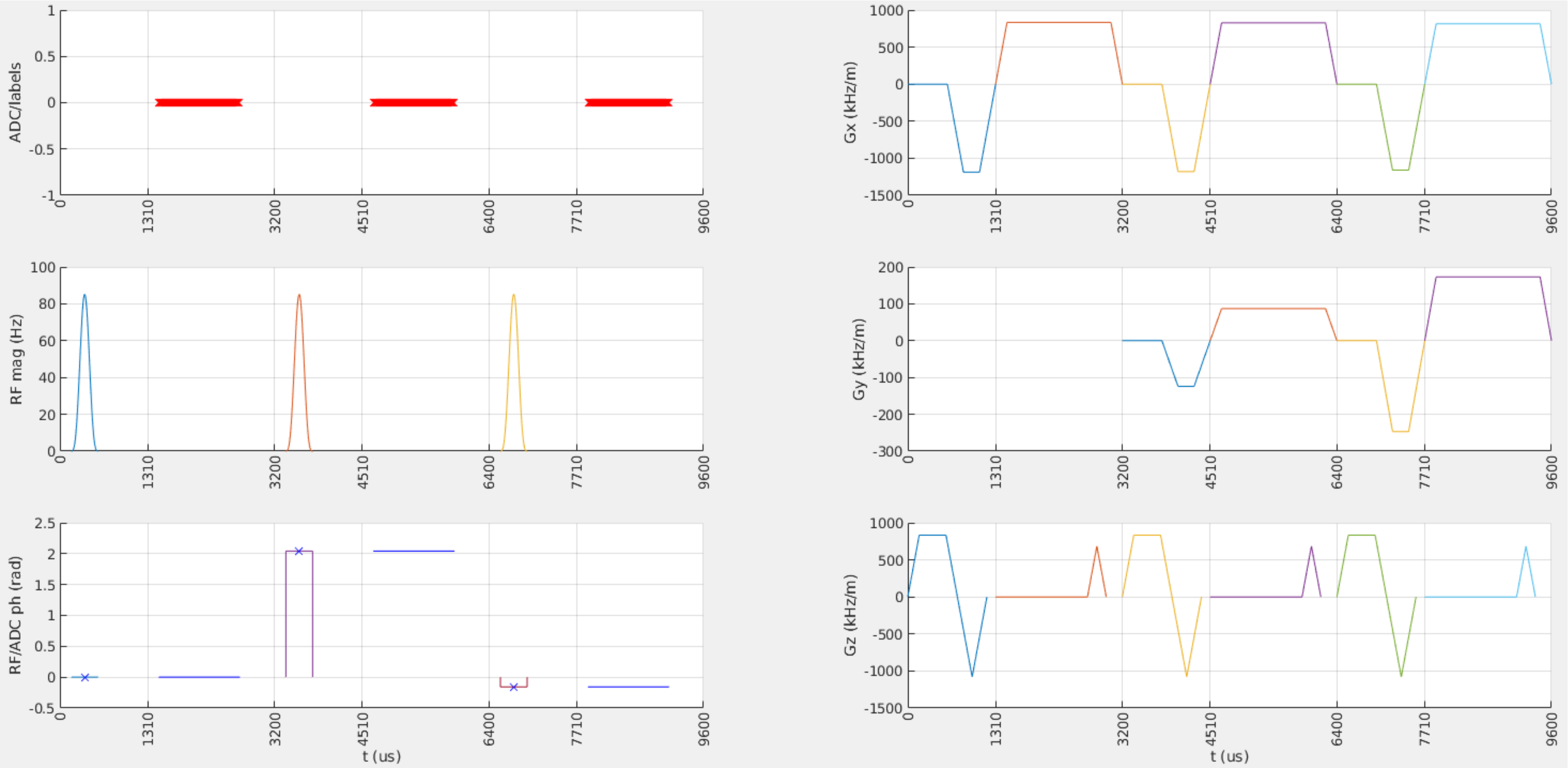
- Plot k-space time evolution or 2D (or even 3D) trajectories
  - Native k-space unit in *Pulseq*: m^{-1}

Example 2: fast radial gradient echo



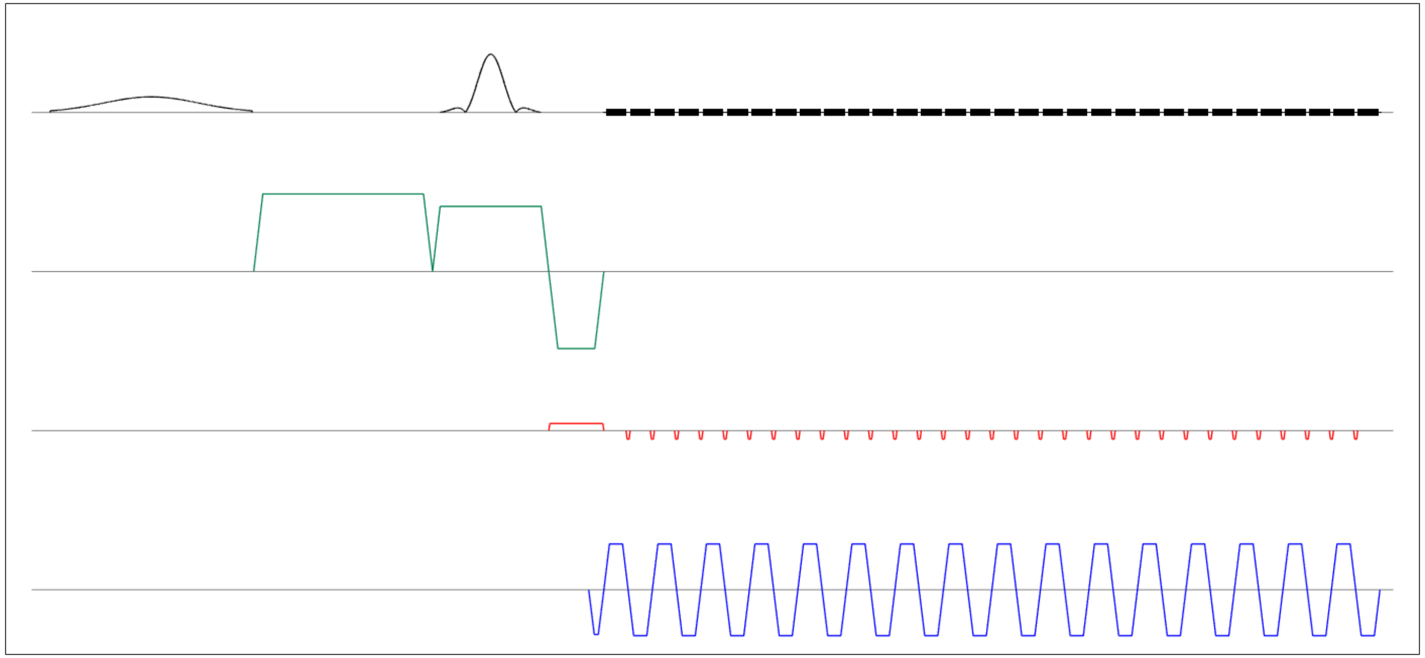
- Block separation is less obvious
  - Merging all into one block is possible, but this would make Z spoiler a part of a shaped gradient….

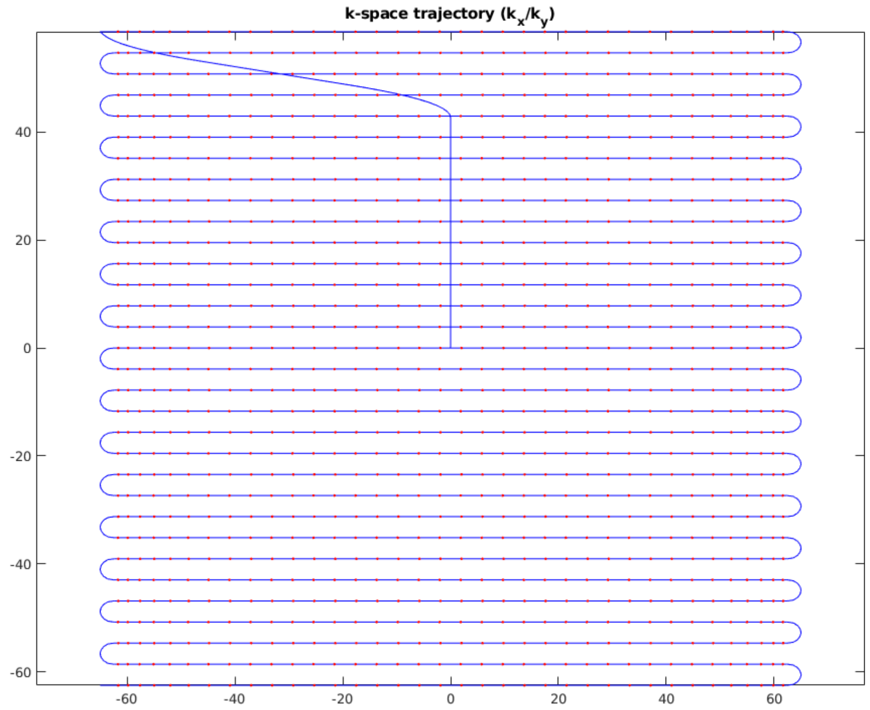
Fast radial gradient echo block structure



- This is one of many possible solutions
  - Many options work, but your Pulseq file size may vary

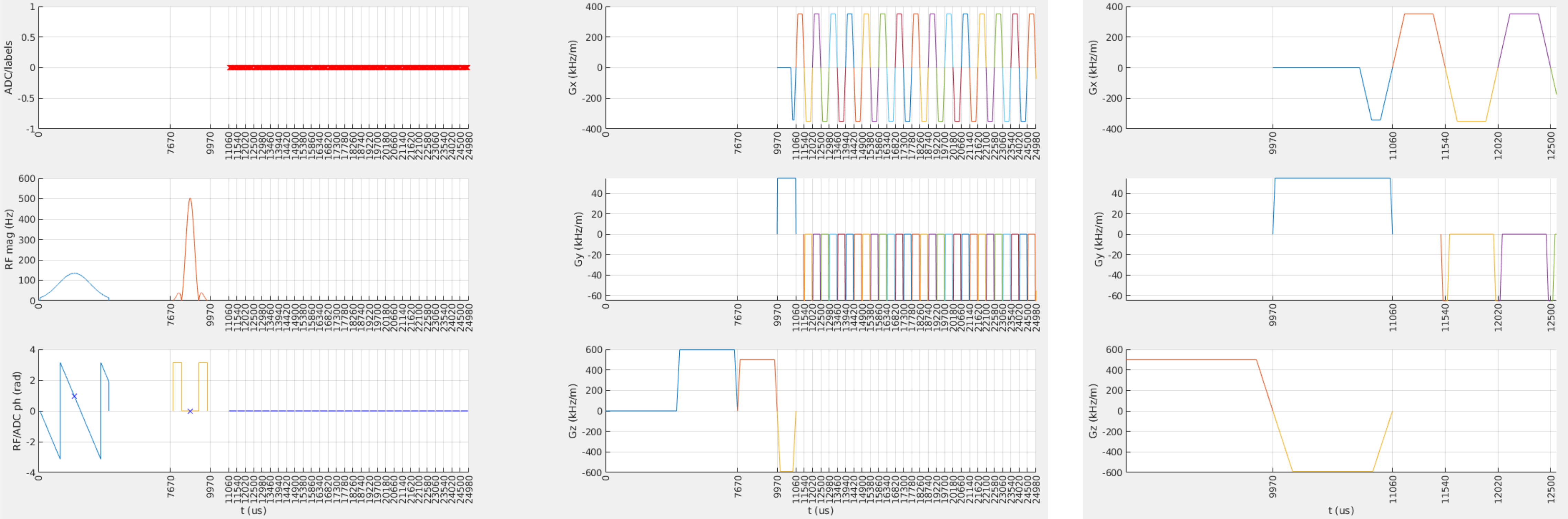
Example 3: echo planar imaging (EPI)





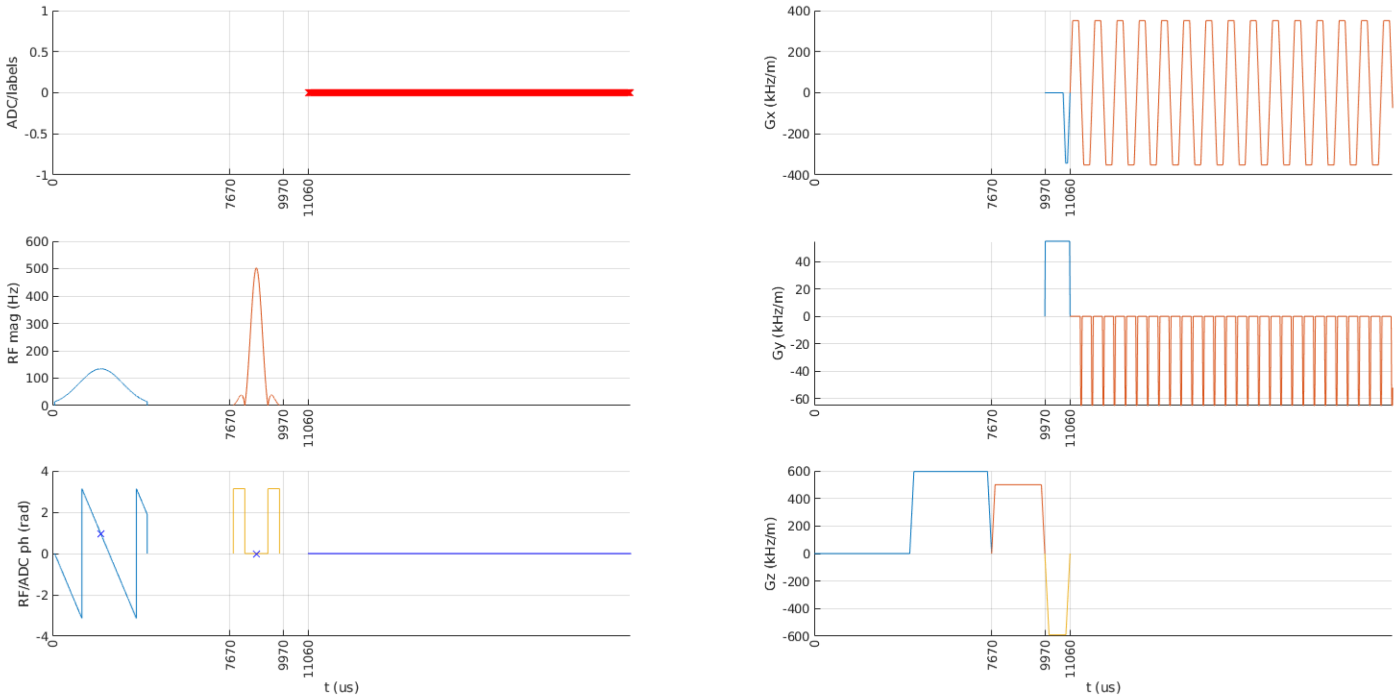
- Two RF pulses need to be in separate blocks
- Each ADC event needs to be in a separate block
  - Challenge with blips, readout ramp and optimal sampling window

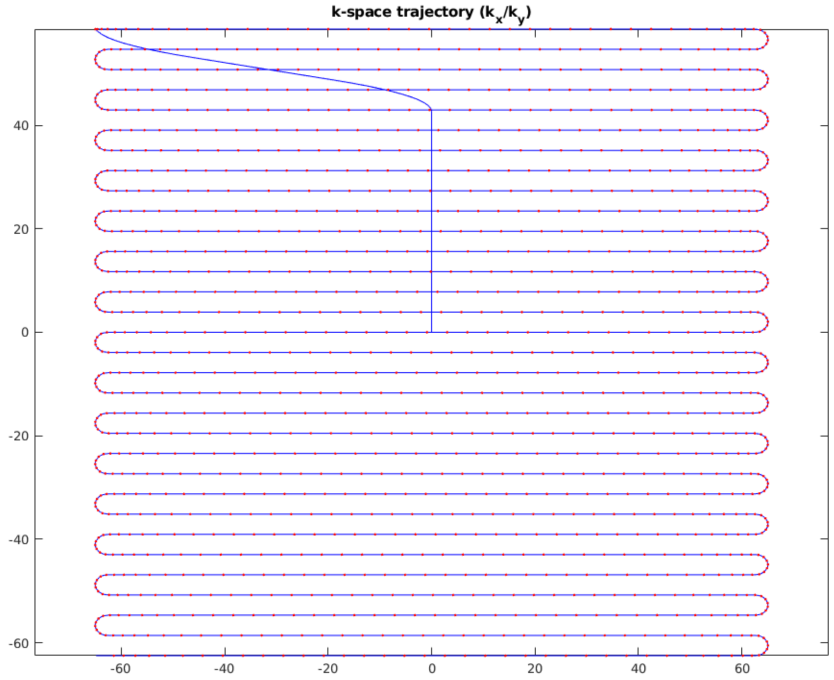
EPI block structure



- One possible solution:
  - Keep readout gradient as trapezoid
  - Convert blips to shapes and split them at the center

Alternative EPI block structure





- Put the entire readout into one block
  - Gx and Gy are now both shaped gradients
  - ADC is sampling continuously (need to crop some points in the recon)

Pulseq blocks summary

- Block concept takes some time to get used to
- Blocks help to organize events and eliminate timing errors
- There is a lot of flexibility
  - Different strategies possible
- Some interpreters expose additional limitations
  - Explicit and implicit delays, number of ADCs per TR, etc…
  - You will hear more about this in the next talks

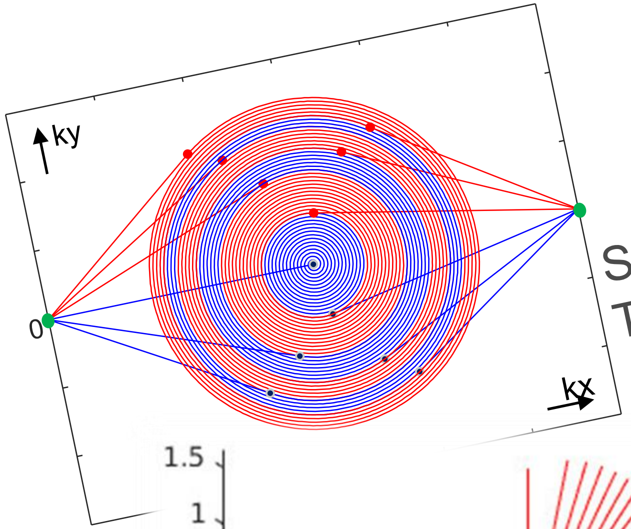
- **Blocks make it easier for the interpreter to play things out**

Pulseq objects & blocks

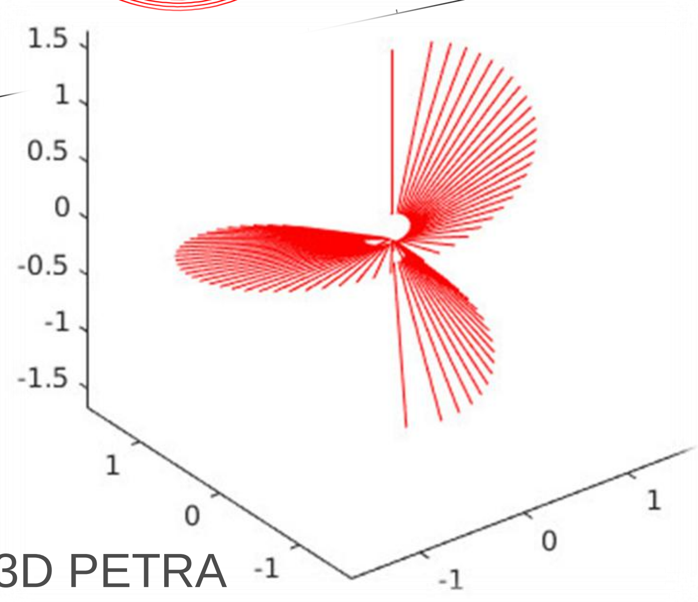
- *Pulseq* objects ‘live’ inside *Pulseq* blocks
  - Gradient & RF pulses
  - ADC objects
  - Delays (dummy objects)
  - Extension objects
    (data labels, cardiac trigger directives, trigger pulses)
- More in the next presentation by:
  Jon-Fredrik Nielsen “Working with Pulseq objects”
- How to assemble sequences from blocks and objects?
  Qingping Chen “Tutorials of some basic sequences”

Cross-platform sequences with *Pulseq*

- MR physics-oriented workflow
  - Write your sequences from scratch
  - Non-Cartesian readouts, user-defined gradient shapes and custom RF pulses
  - Advanced visualization and analysis tools
  - Automatic k-space calculation
- Pulseq files play out on many scanners
  - Siemens & GE : works
  - Philips : stay with us till Day 3
- Upcoming New release v1.4.2 (Dec 2023)



Spiral TSE



UNIVERSITÄTS KLINIKUM FREIBURG

Acknowledgements:

Berkin Bilgic
Frank Zijlstra
Jon-Fredrik Nielsen
Moritz Zaiss
Qiang Liu
Sebastian Littin
Borjan Gagoski
Imam Shaik
Juergen Hennig
Naveen Murthy
Qingping Chen
Will Grissom
Douglas Noll
Jeff Fessler
Mojtaba Shafiekhani
Niklas Wehkamp
Scott Peltier
Yogesh Rathi

THANK YOU FOR YOUR ATTENTION!

Pulseq MATLAB

pypulseq Python

TOPPE

Supported by NIH U24-NS120056 (Nielsen, Zaitsev) and R01-EB032378 (Rathi, Bilgic)

NIH National Institutes of Health