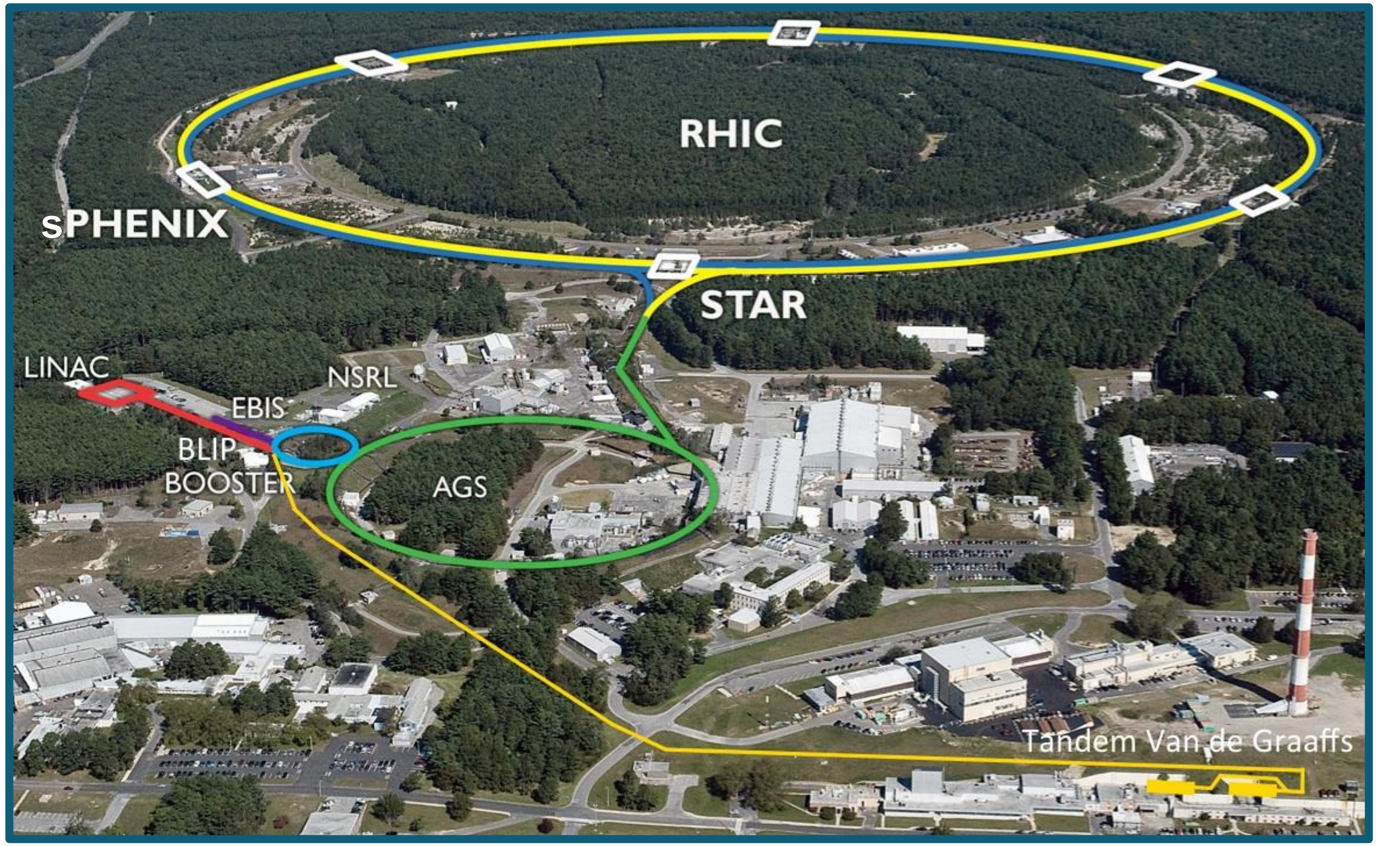
A Physics-Based Simulator to Facilitate Reinforcement Learning in the RHIC Accelerator Complex

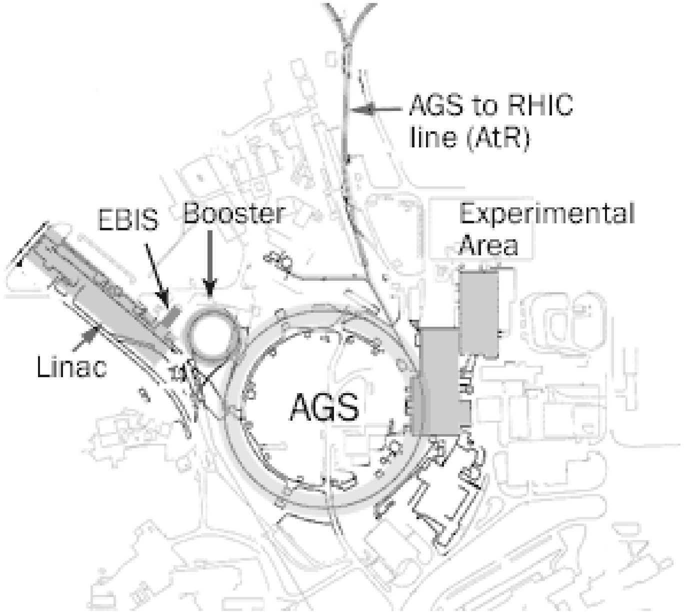
L.K. Nguyen, K.A. Brown, M.R. Costanzo, Y. Gao, M. Harvey, J.P. Jamilkowski, J.T. Morris, V. Schoefer

October 13th, 2023

@BrookhavenLab

RHIC Accelerator Complex at BNL





- Bunches merge in Booster for injection into the Alternating Gradient Synchrotron (AGS)
- Bunches merge in AGS for injection into the Relativistic Heavy Ion Collider (RHIC)

Brookhaven National Laboratory

Motivation

Good bunch merging is crucial for operations but not trivial to achieve. Machine learning (ML) provides a promising tool for improving bunch merges. **BUT:**

- **Real machine time for ML development can be very hard to come by.**
  Booster & AGS are part of the accelerator chain for multiple programs, and they are often in operational use when not supplying RHIC with beam.
- **Real machine time for ML development is expensive and has opportunity costs.**
  Downtime is generally allotted to maintenance and/or needed repairs. Meanwhile, part of the ML development cycle is purely software-related (e.g., debugging).

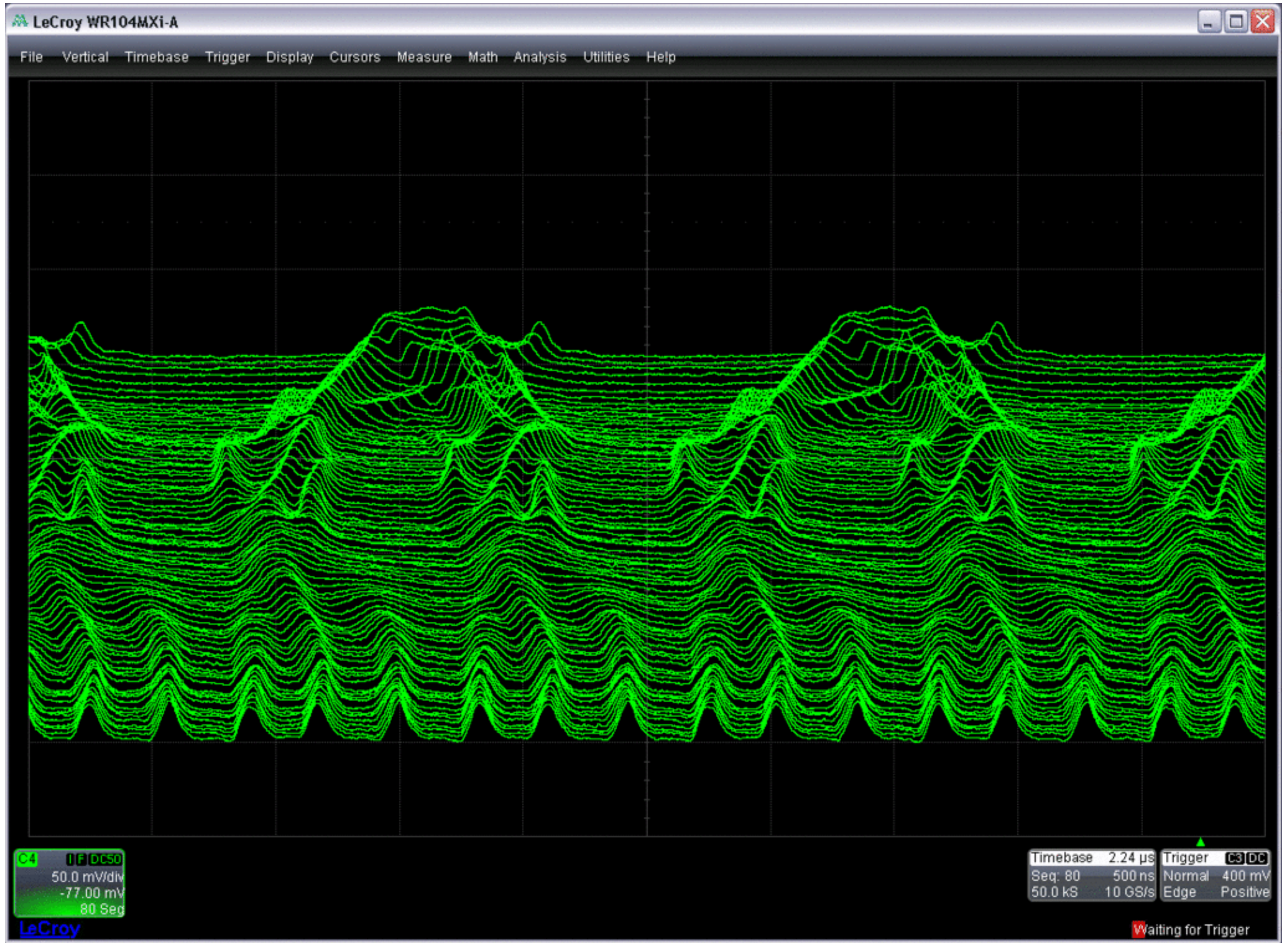
Some ML approaches, such as reinforcement learning (RL), do not learn machine parameters and are therefore amenable to other development paths.

Replace accelerator, diagnostics, and controls with simulator*

* In this case, written in Python

Real Environment

- To diagnose a merge, a wall current monitor (WCM) generates a voltage vs. time signal. Signal traces are subsequently stacked on a scope to create a mountain range plot.
- Each trace is separated in time from the one before it by a certain number of accelerator periods (referred to herein as *N turns*).
- Scope has usual settable properties (e.g., timebase, trigger, etc.).

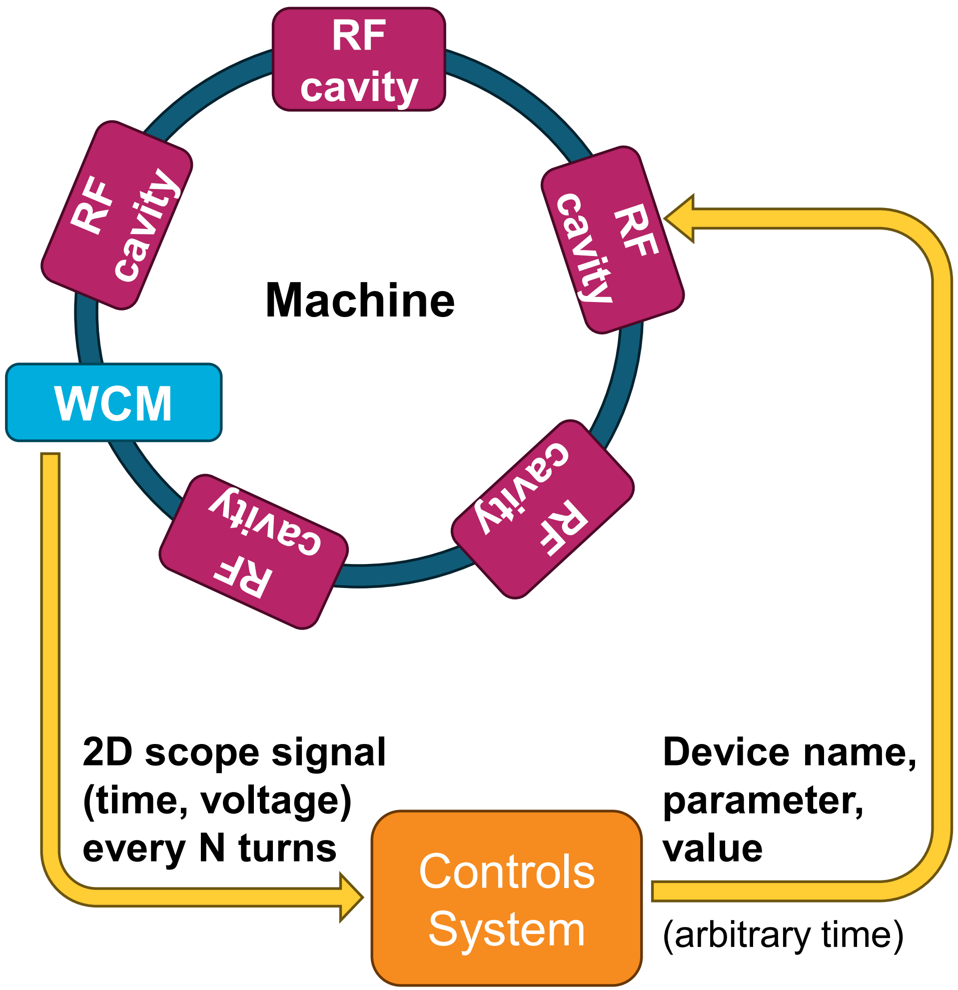


Real mountain range data showing an overall 6-to-1 merge (6-to-2 merge followed by 2-to-1 merge) in Booster

Brookhaven National Laboratory

Real Environment, cont.

- For the merge, RF gymnastics are performed via different RF harmonics—but **not** necessarily different physical cavities.
- Booster & AGS differ in number of physical cavities and can differ in harmonics and merge pattern. They naturally differ in energy, slip factor, and other beam/accelerator qualities.
- Voltage and phase are the available knobs for a given RF harmonic.
- Knobs are controlled by specifying device name, parameter, and new value.



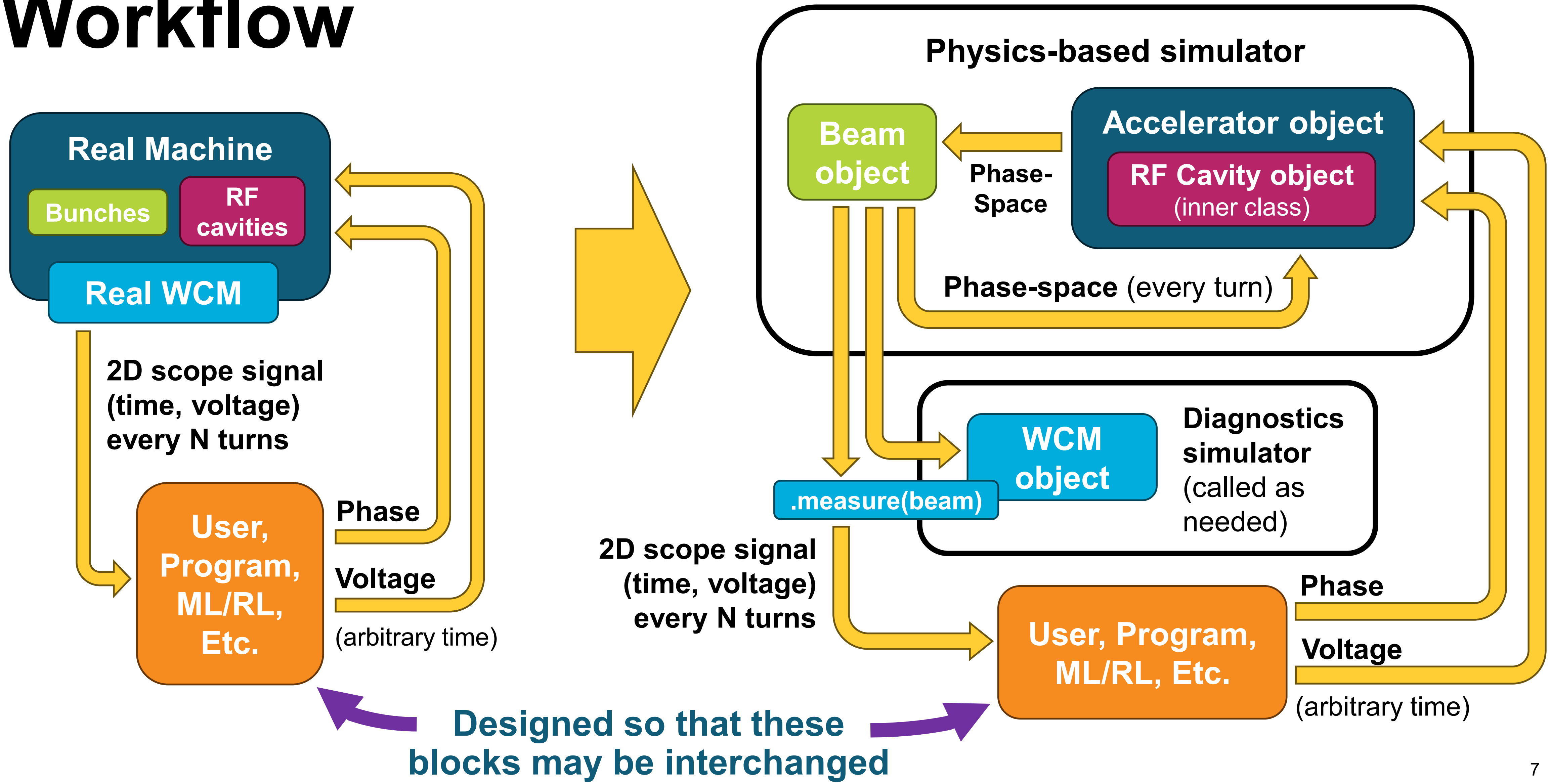
Cartoon representation of accelerator with WCM, RF cavities (arbitrary number), and input/output

Simulator & Code Overview

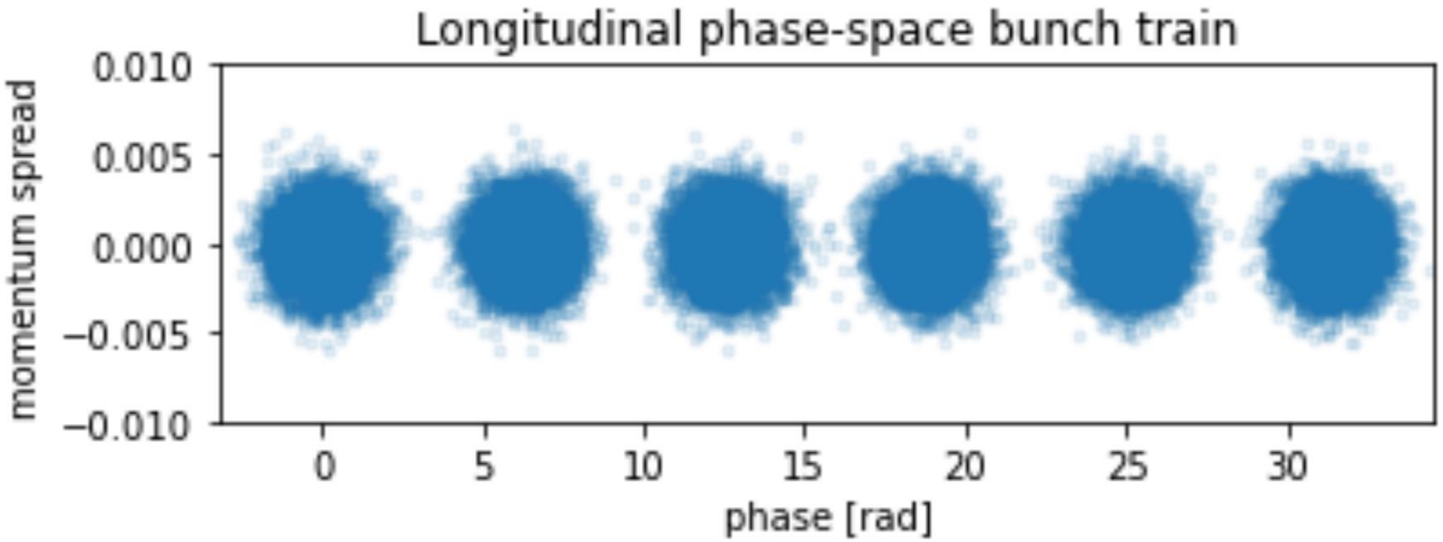
(Note that implemented variables and functions constitute an ever-growing list as the simulator evolves)

Bunch Merge Simulator Workflow

Object-Oriented Programming



Beam Object



Example of six 80-nanosecond bunches constituting one full orbit (i.e., 12π rads in phase) at 400 kHz revolution frequency. Each bunch is a 2D Gaussian distribution of 10,000 particles (variable).

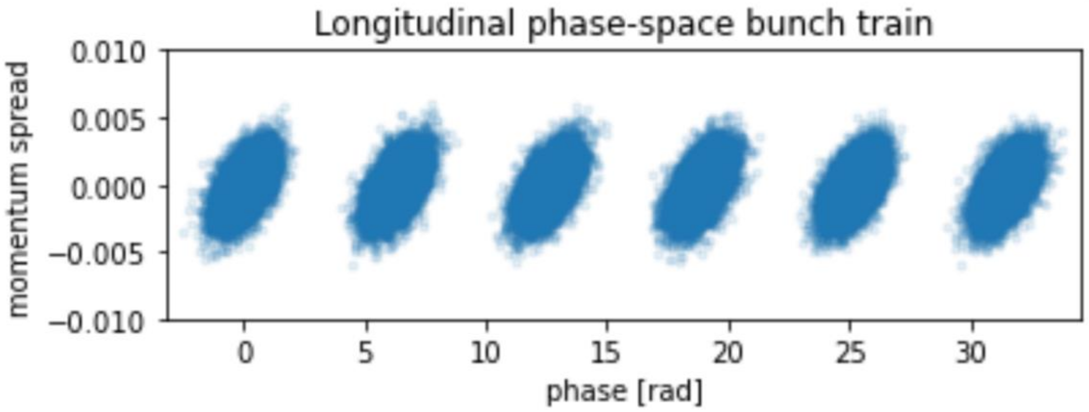
Covariance and bunch variation settings can be combined arbitrarily. Phase resolution and σ of Gaussian filter affect phase-space projection for time signal (see later slide).

Instantiation Arguments

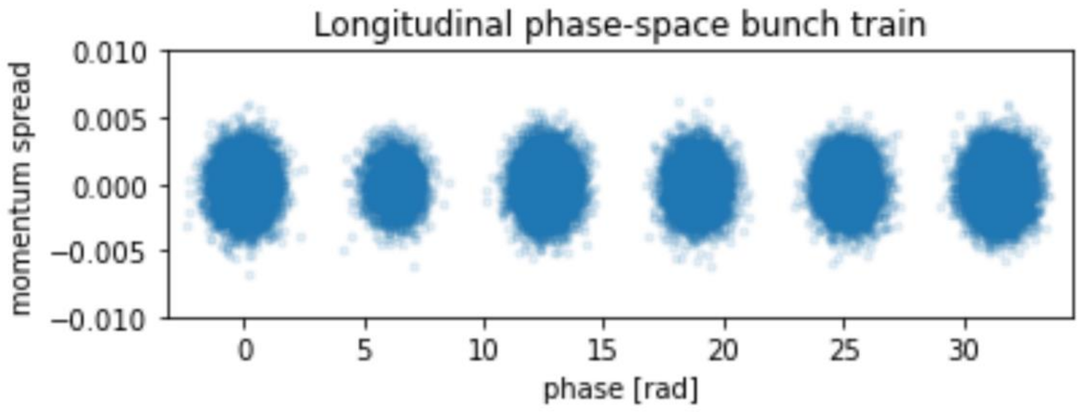
- Particles per bunch
- Bunches in orbit
- Signal strength
- Bunch length (ns)
- Momentum spread
- Covariance
- Particle number variation
- Bunch length variation
- Bunch timing variation
- Phase resolution
- Gaussian filter σ

Main Methods

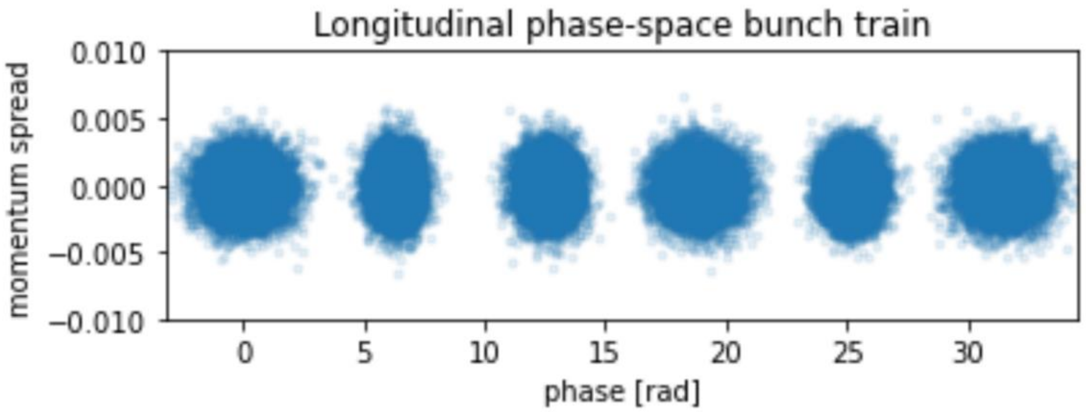
```
.collapse()
.beamLoss(loss)
.phaseShift(shift)
```



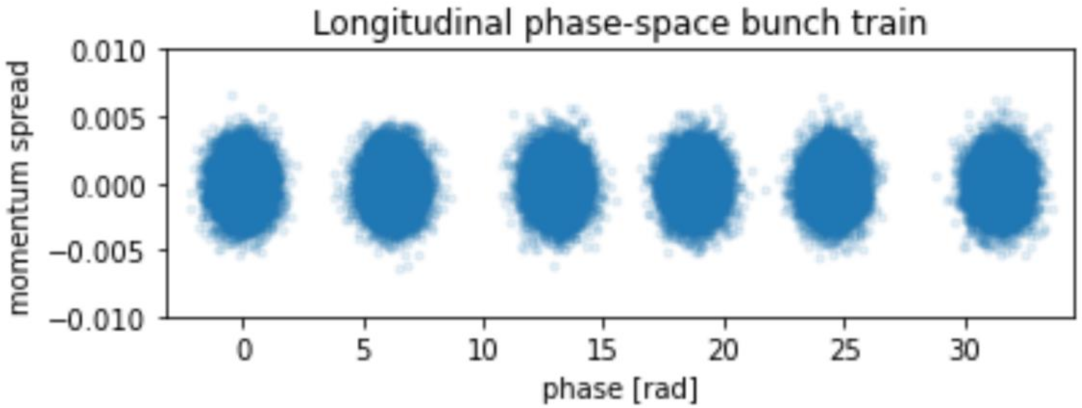
Covariance



Particle number variation



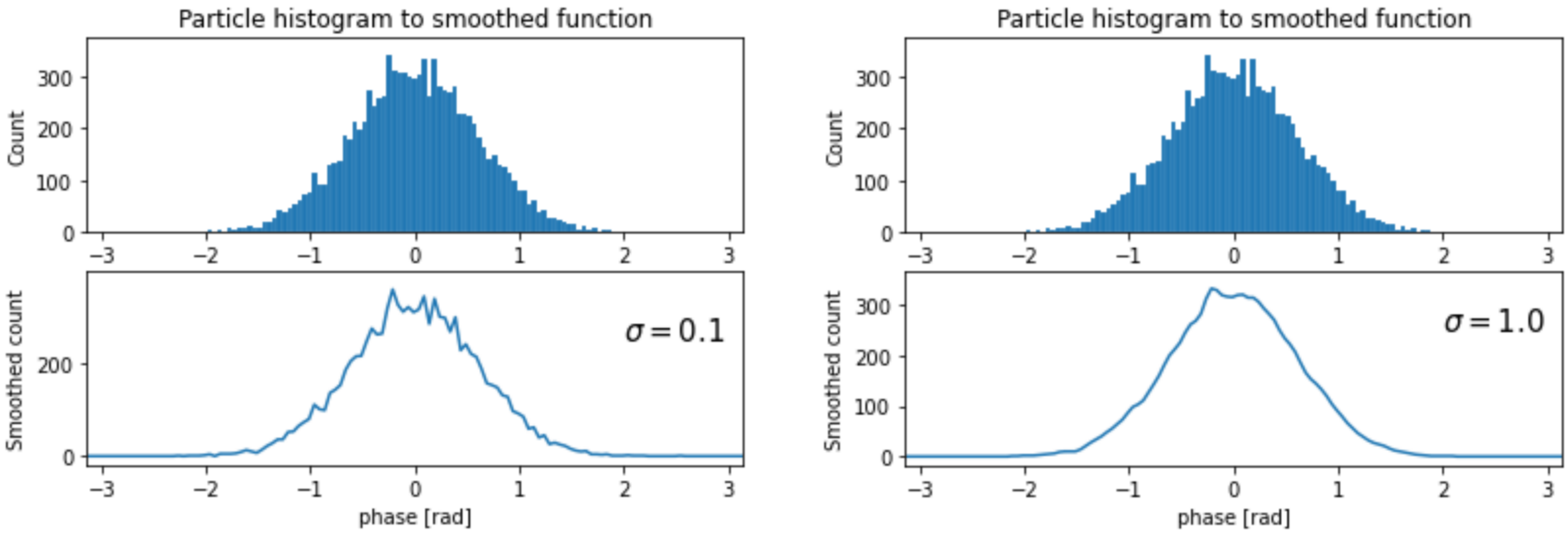
Bunch length variation



Bunch timing variation

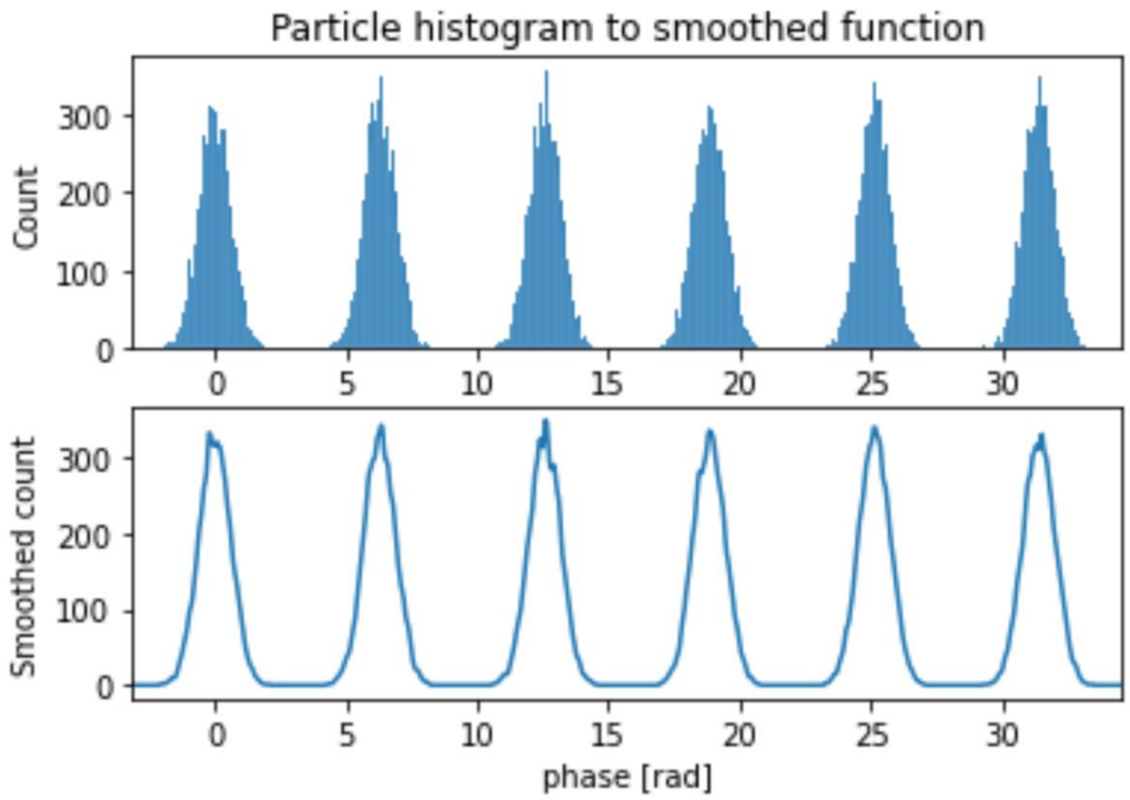
Phase-space projection

Phase-space projection is the first step in going from 2D phase space to a voltage vs. time signal. The Beam().collapse() method creates a particle histogram (phase resolution for bins can be set) that is smoothed via a Gaussian filter and then scaled via the signal strength parameter. Phase is converted to time via the revolution frequency.



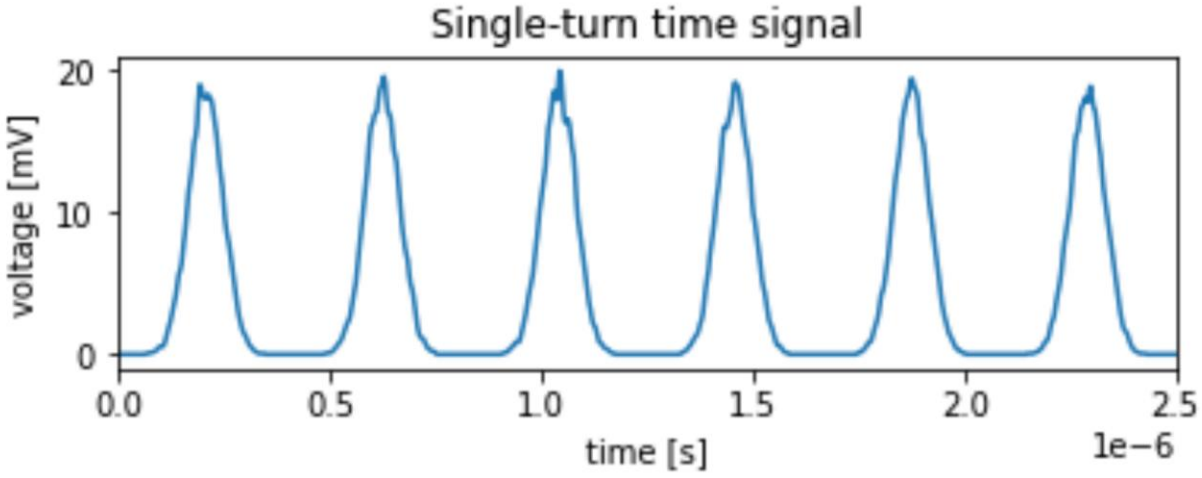
Close-up of one bunch projected for different Gaussian filter sigmas

Phase to time signal



All bunches as phase signal

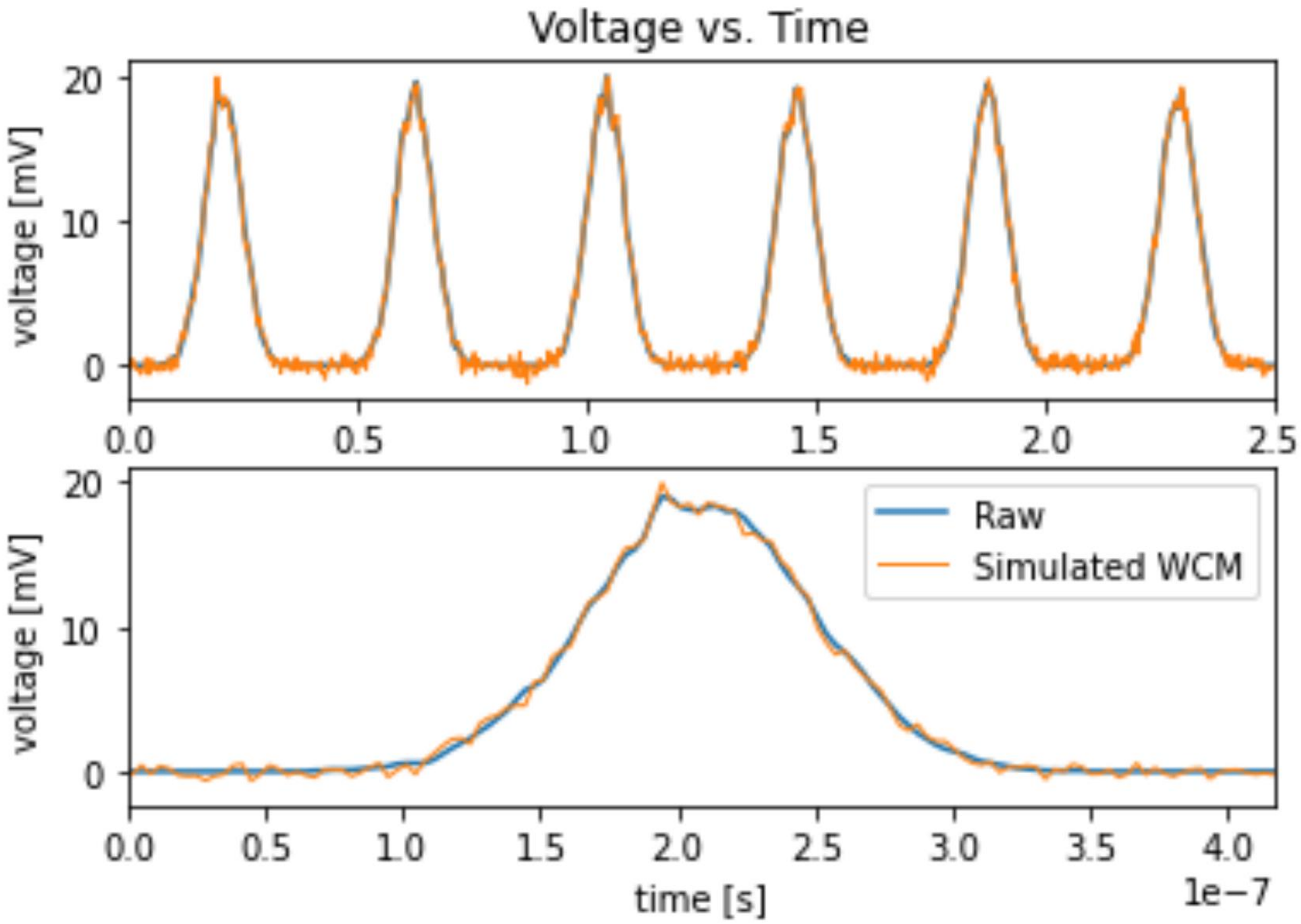
- Note how signal strength parameter (20 mV in example) is used for y-axis



All bunches as time signal

- 400 kHz revolution frequency → 2.5 μs for one turn

WallCurrentMonitor Object



Imparts acquisition noise and other simulated scope properties. Accumulates traces for mountain range plot.

Instantiation Arguments

- Sampling rate
- Number of samples per acquisition
- Turns between acquisitions
- Number of acquisitions
- Trigger delay
- Bit resolution of scope
- Acquisition noise

Main Attributes

```
.acquisition
.accumulator
```

Main Methods

```
.measure(beam)
.display()
.clear()
```

Available Outputs

Simulator provides four 2D NumPy arrays as Beam object attributes:

Phase as x-axis, momentum spread as y-axis:
`phase, delta = beam.phase_space`

This is the signal used by the Accelerator object

Phase as x-axis, smoothed bunch count as y-axis:
`phase, phase_train = beam.phase_signal`

Intermediate step to generating time signal

Time as x-axis, phase signal amplitude scaled to voltage:
`time, voltage = beam.raw_time_signal`

Passed to WCM object for simulated diagnostics

Time as x-axis, simulated WCM output:
`time, voltage = beam.measured_signal`

Serves as traces and the primary output for ML

Note that `beam.raw_time_signal` and `beam.time_signal` will be identical if no additional acquisition noise or settings are added

Physics Simulation and the Accelerator Object

Evolution of Beam object is handled by Accelerator object and takes place in phase space according to longitudinal phase-space mapping equations:

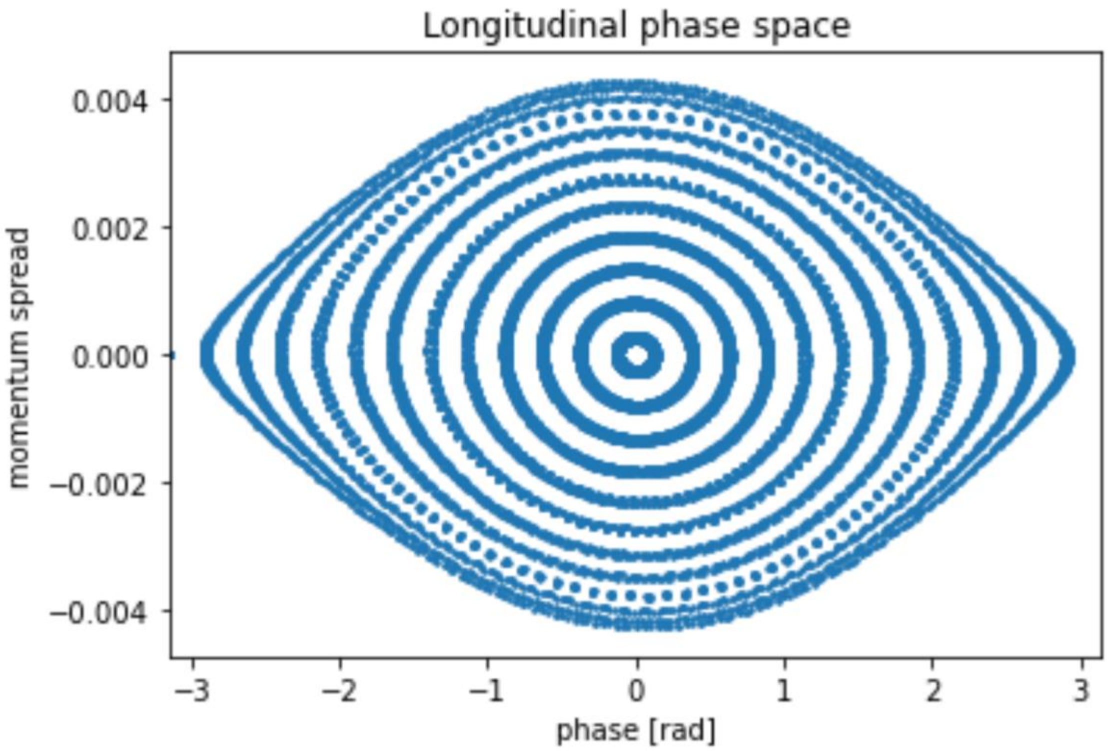
$$\delta_{n+1} = \delta_n + \sum_{i}^{len(h)} \frac{ZeV_i}{\beta^2 E}\left(\sin\varphi_{n,i} - \sin\varphi_{s,i}\right)$$

$$\varphi_{n+1} = \varphi_n + 2\pi h_N \eta \delta_{n+1}$$

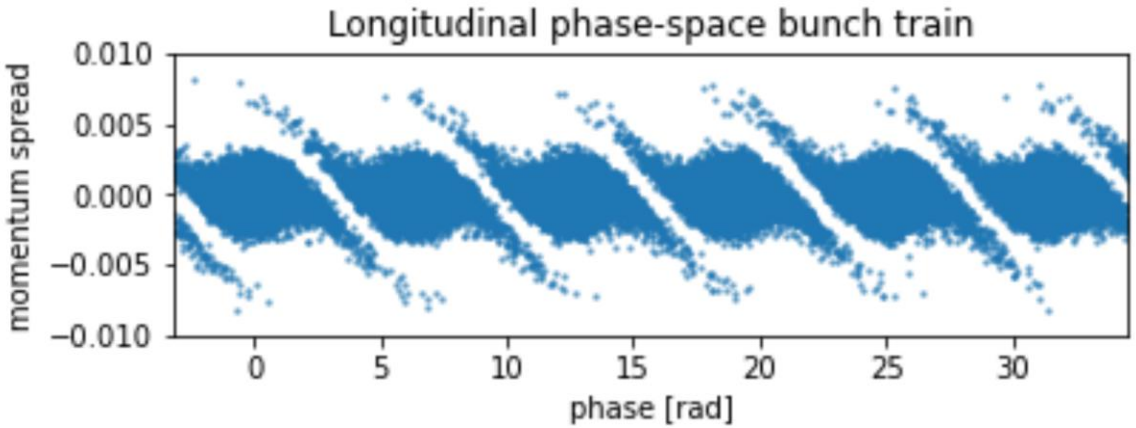
h = **h_list**, V = **v_list**, φ = **ph_list**

- h **is harmonic of *revolution* frequency**
- h_N **is base harmonic used for accounting**
- **Lump-element model means** φ_s **for each harmonic can be approximated by** φ_i **(i.e., set phase offset)**

Sanity checks:



On-momentum phase-space ellipses with constant voltage



Phase wrapping (full orbit tracking)

Accelerator & RF Cavity Object

Accelerator object

RF Cavity object
(inner class)

```
machine = Accelerator(...,h_list,v_list,
                      ph_list,v_names,ph_names,...)
```

- `machine.cavity.harmonics = h_list` (e.g., `[1,2,3,6]`)
- `machine.cavity.voltages = v_list` (in kV, e.g., `[0,0,0,2.5]`)
- `machine.cavity.phases = ph_list` (in rads, e.g., `[0,0,0,0]`)

Instantiation Arguments

- Species
- Particle rest energy
- Charge number Z
- Machine name
- Energy
- Slip factor
- Revolution frequency
- Merge harmonics
- Initial RF voltages
- Voltage device names
- Initial RF phases
- Phase device names
- Voltage noise
- Phase noise

- **`v_names` and `ph_names` contain device+parameter names per Controls System syntax (see next slide)**

Main Attributes

- `.turn`
- `.line`
- **`.action`**
- **`.cavity.harmonics`**
- **`.cavity.voltages`**
- **`.cavity.phases`**
- **`.cavity.params`**
- **`.cavity.v_increment`**
- **`.cavity.ph_increment`**

- **`.cavity.voltages` and `.cavity.phases` change during execution according to `.cavity.v_increment` and `.cavity.ph_increment` , respectively**
    - **`.action` signals new line of increments**

Main Methods

- **`.loadCommands(commands)`**
- `.unloadCommands()`
- **`.simulate(beam,turns)`**
- `.cavity.getValue(parameter)`
- `.cavity.setValue(parameter,value)`
- `.reset(attribute)`

Device/Controls Architecture

- `machine.cavity.setValue(setting,value)` is used to change voltage or phase
- `setting` is a list of the form: `[‘Device name’, ‘Device parameter’]`
- List is mapped to `.voltages` or `.phases` accordingly and changed to `value` for use in subsequent iterations of simulator

Example:

```
v_names = [['Cavity1', 'h1_VoltageSetpoint'],
           ['Cavity2', 'h2_VoltageSetpoint'],
           ['Cavity3', 'h3_VoltageSetpoint'],
           ['Cavity4', 'h6_VoltageSetpoint']]

ph_names = [['RFSupply1', 'h1_PhaseOffset'],
            ['RFSupply2', 'h2_PhaseOffset'],
            ['RFSupply3', 'h3_PhaseOffset'],
            ['RFSupply4', 'h6_PhaseOffset']]
```

Index in `h_list` will correspond to row in `v_names` and `ph_names`

For merge via scripted program example:

```
# Specify each command as a row in the following format:
# [Time in ms, ['Device name', 'Parameter’],value]
# Voltage in kV, phase in rads

merge_commands = [
    [0, ['Cavity4', 'h6_VoltageSetpoint'], 2.5],
    [8, ['Cavity4', 'h6_VoltageSetpoint'], 1.25],
    [8, ['Cavity3', 'h3_VoltageSetpoint'], 0.8335],
    [16, ['Cavity4', 'h6_VoltageSetpoint'], 0],
    [16, ['Cavity3', 'h3_VoltageSetpoint'], 1.25],
    ...
    [48, ['Cavity3', 'h3_VoltageSetpoint'], 0.35],
    [48, ['Cavity1', 'h1_VoltageSetpoint'], 1.46045],
    [60, ['Cavity3', 'h3_VoltageSetpoint'], 0],
    [60, ['Cavity2', 'h2_VoltageSetpoint'], 0.730225],
    [72, ['Cavity2', 'h2_VoltageSetpoint'], 0]
    ]
```

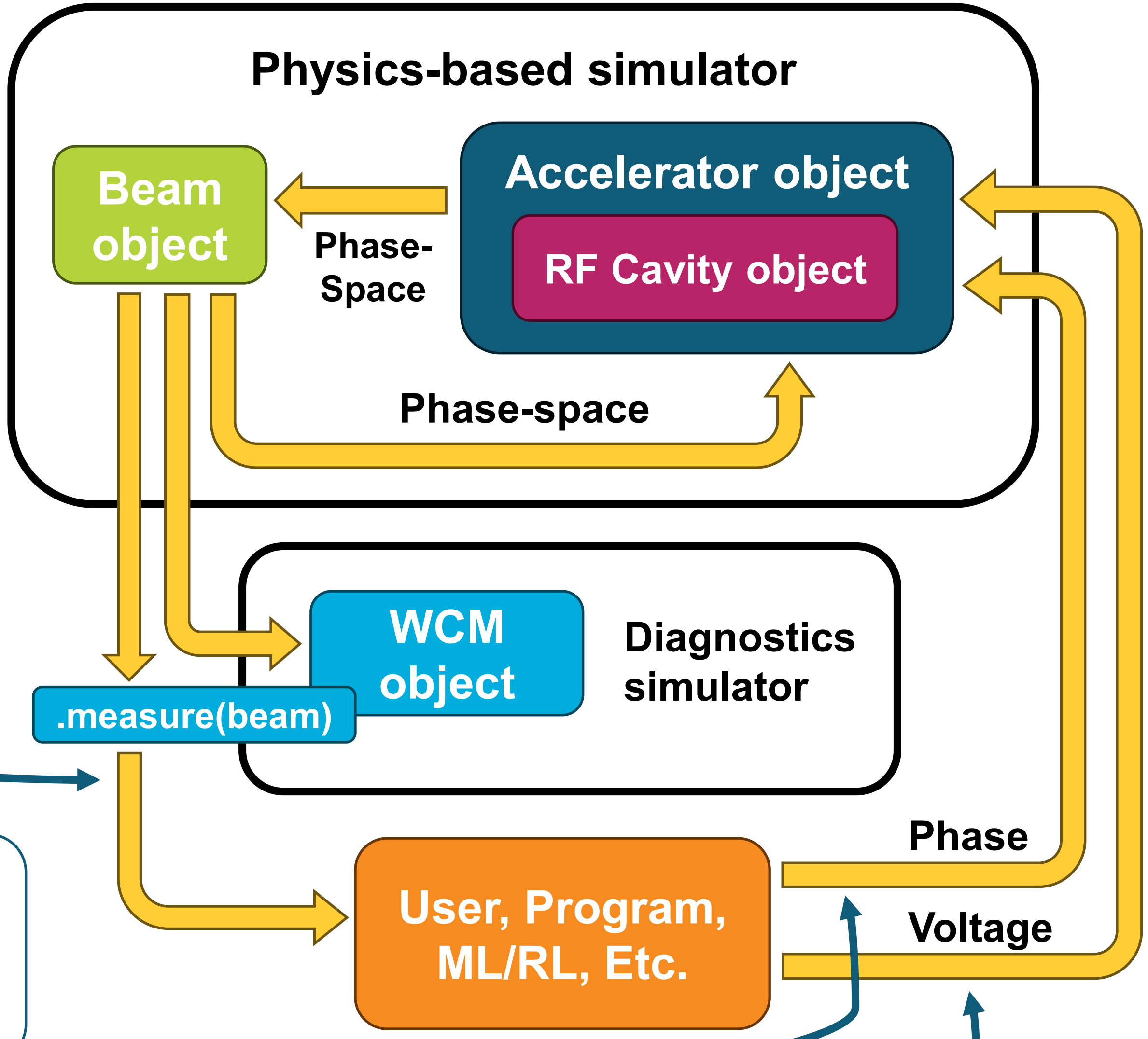
`Time` is converted to turn number and unique turn numbers create a new line in `.action`

Data Structure

OUTPUT from simulator

2D array:
[time, voltage]

```
array([[1.66002656e-09, 4.98007968e-09, 8.30013280e-09,
        1.16201859e-08, 1.49402390e-08, 1.82602922e-08,
        ...
        2.48837981e-06, 2.49169987e-06, 2.49501992e-06,
        2.49833997e-06],
       [7.66208925e-01,  7.87920809e-01,  4.26911847e-01,
        2.21427015e-01, -8.54739956e-02,  4.55331427e-01,
        ...
        8.50578511e-01,  7.87545251e-01,  5.94906358e-02,
        9.82494844e-01]])
```



INPUTS to simulator

List with time (ms) into merge to reach new setpoint, device name, device parameter, and new setpoint value

```
[8, ['RFSupply4', 'h6_PhaseOffset'], 0.05]
[16, ['Cavity4', 'h6_VoltageSetpoint'], 1.25]
```

Execution

INSTANTIATION

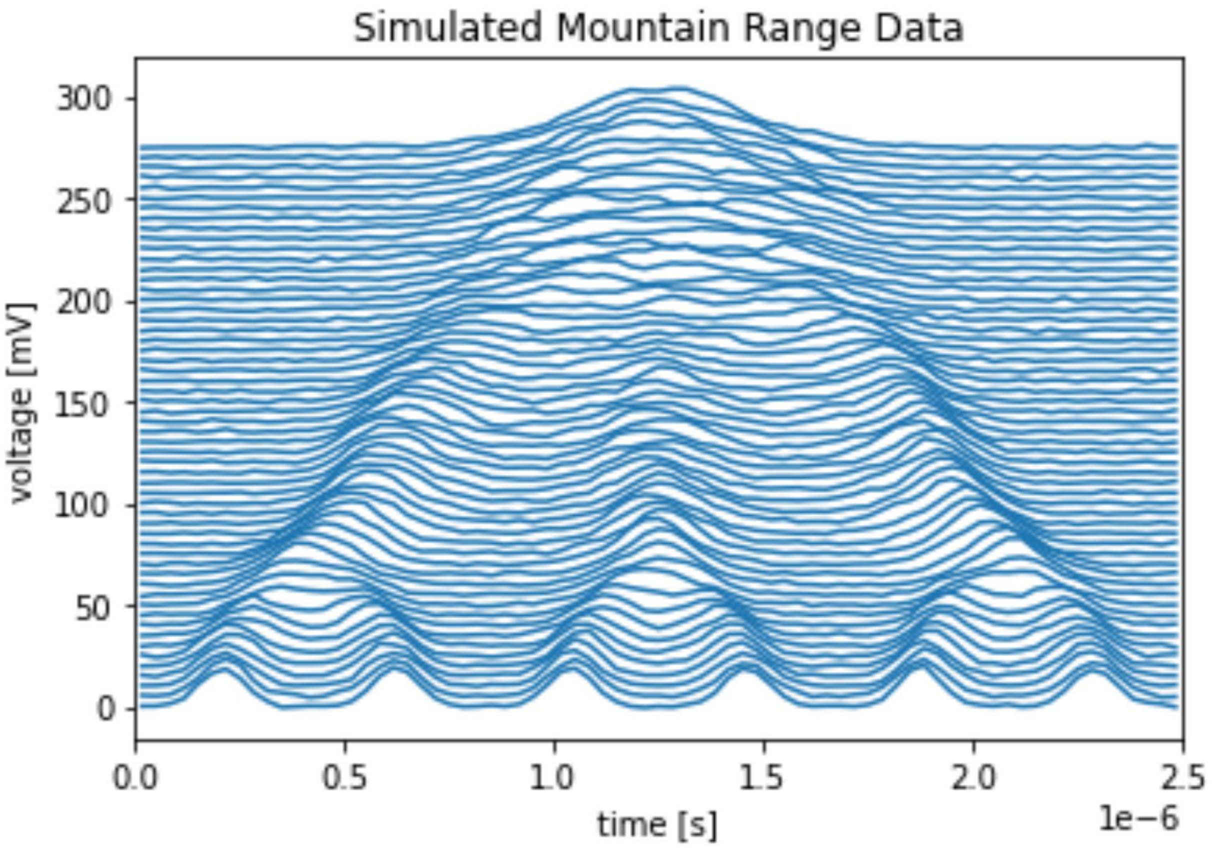
```
wcm = WallCurrentMonitor(N_turns,N_samples,N_length,wcm_noise,
                         trigger_delay,bit_res)

beam = Beam(N_bunches,N_particles,cov,bunch_amp_noise,sigma_noise,
            sigma_ph,phase_jitter,signal_strength,b_num,gauss_sigma)

machine = Accelerator(machine_name,species,E_0,Z,E,eta,h_list,
                      v_list,ph_list,v_names,ph_names,v_noise,
                      rf_ph_noise)
```

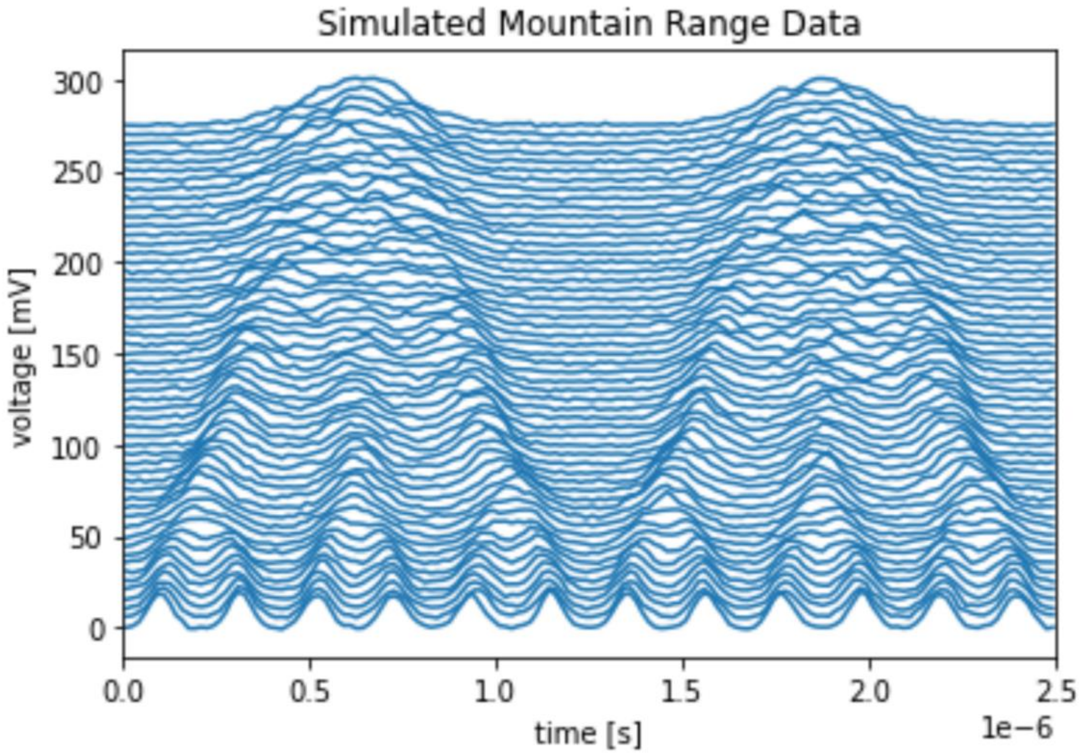
SIMULATION
(programmed merge)

```
machine.loadCommands(merge_commands)

wcm.measure(beam)   # Log initial beam

for i in range(N_measurements):
    machine.simulate(beam, wcm.turns)
    wcm.measure(beam)

wcm.display()   # Mountain range plot
```

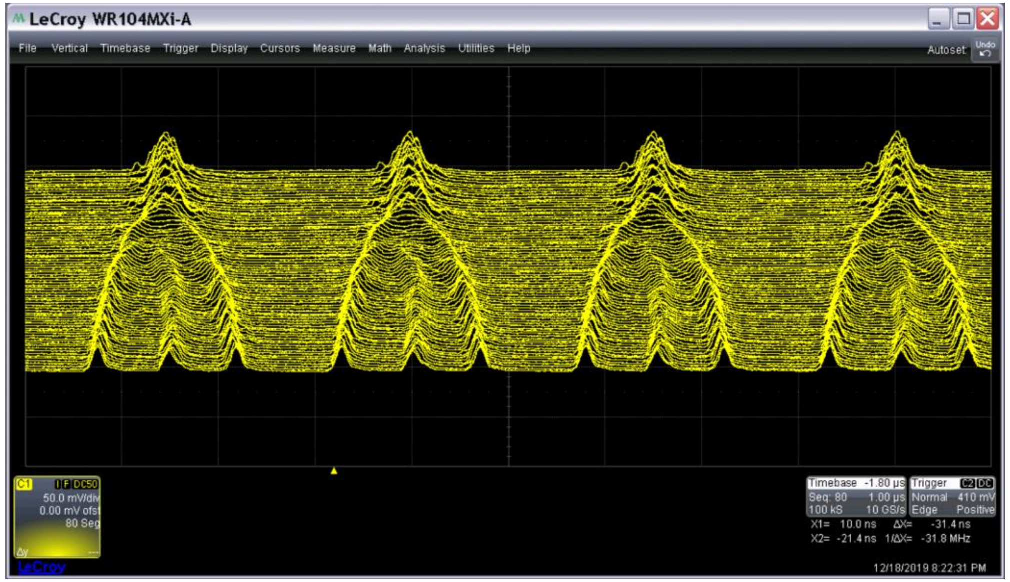
Brookhaven National Laboratory

Example Results





c.f. real mountain range data for a 3-to-1 merge in Booster during Run 20:



Conclusion & Future Work

- We have created a physics-based simulator in Python that mimics our bunch merge environment and diagnostics by combining longitudinal phase-space mapping and phase-space projection for time signal replication
- A diagnostics simulator is included in the framework
- Due to its authentic data and Controls structure, as well as options for injecting noise, the simulator can be used for RL development for improving bunch merges
  - Other envisaged uses include training, troubleshooting, and other systems investigations
- The simulation environment will be expanded to accept console input
  - This is expected to improve RL development/performance since the algorithm would not need to wait until the very end of the merge for feedback
- It is likely that the simulate method (or a version of it) will be rewritten to perform the computationally intensive phase-space mapping portion in C for full-scale simulations (>>10,000 particles per bunch) when GPUs are insufficient/unavailable

Thank you!

Questions?