Brookhaven National Laboratory

Jefferson Lab

U.S. DEPARTMENT OF ENERGY

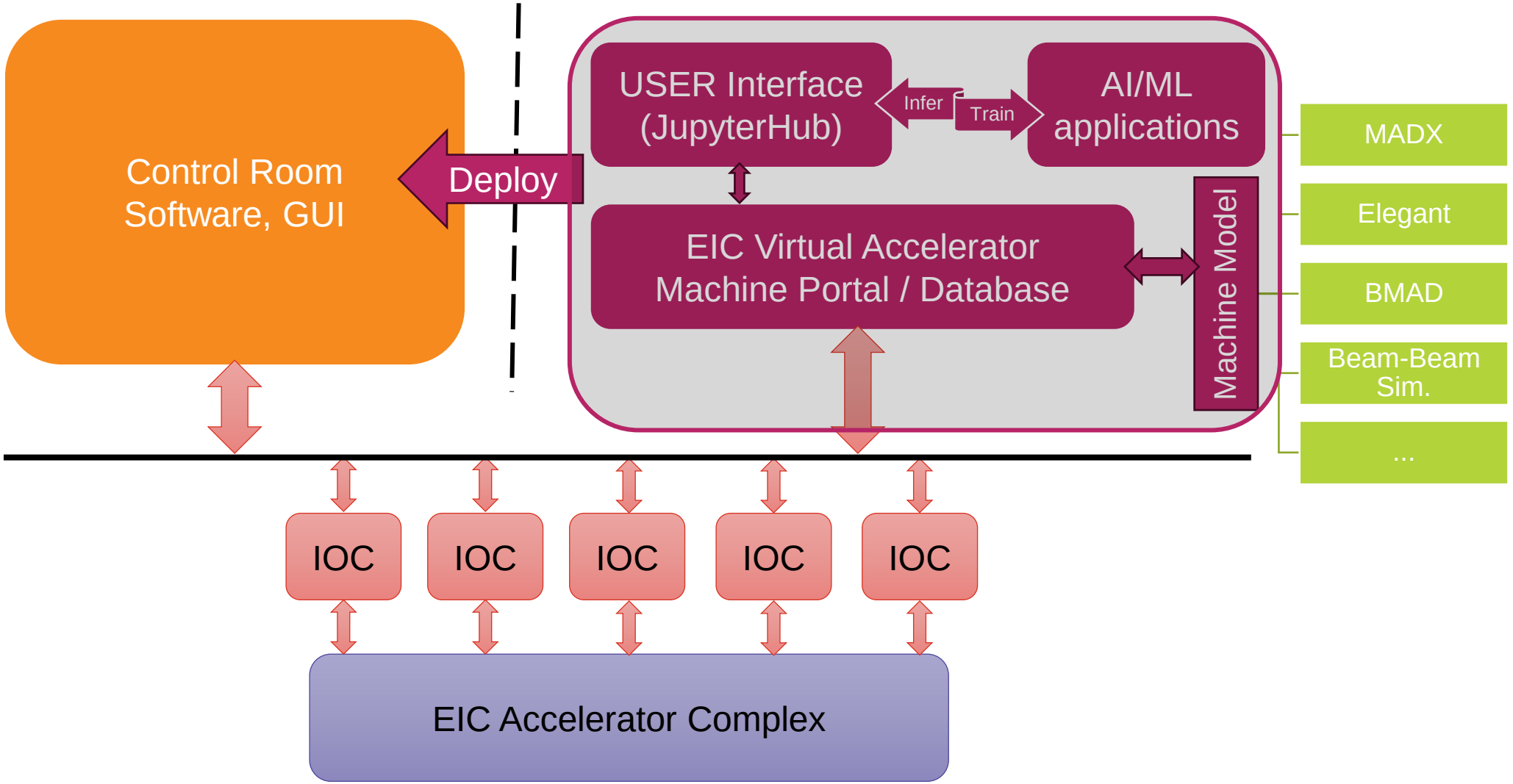
EIC International Accelerator Collaboration Commissioning Tool Workgroup

EIC Virtual Accelerator Updates

Yue Hao

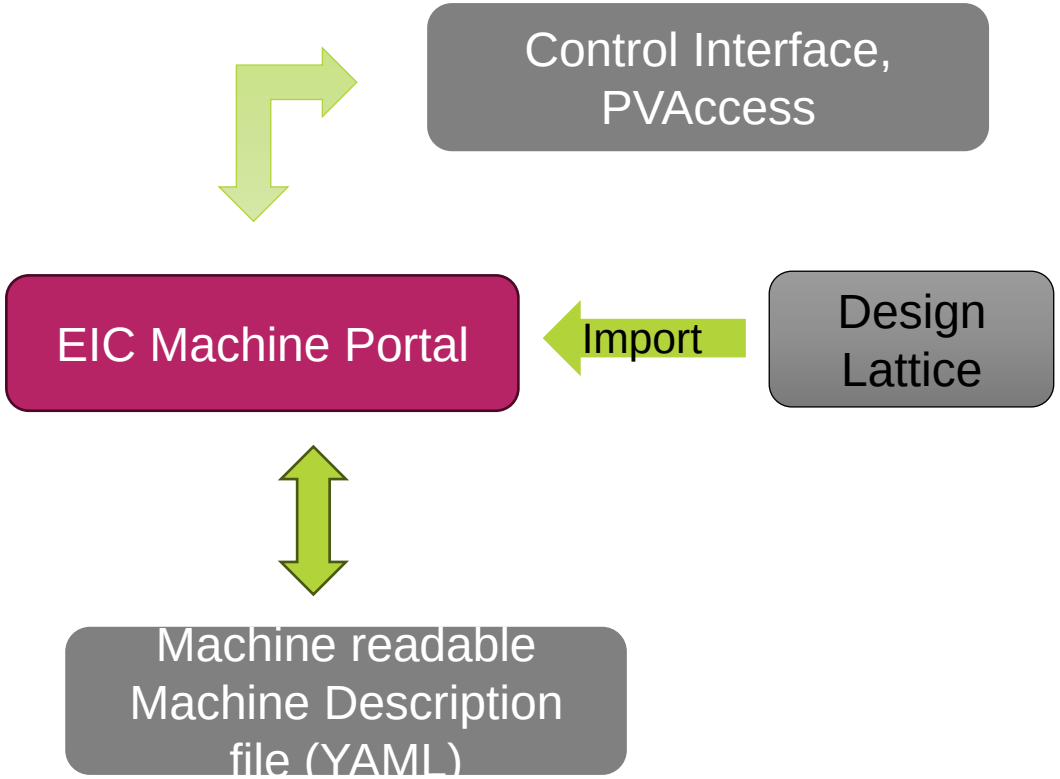
Electron-Ion Collider

Plans for EIC VA

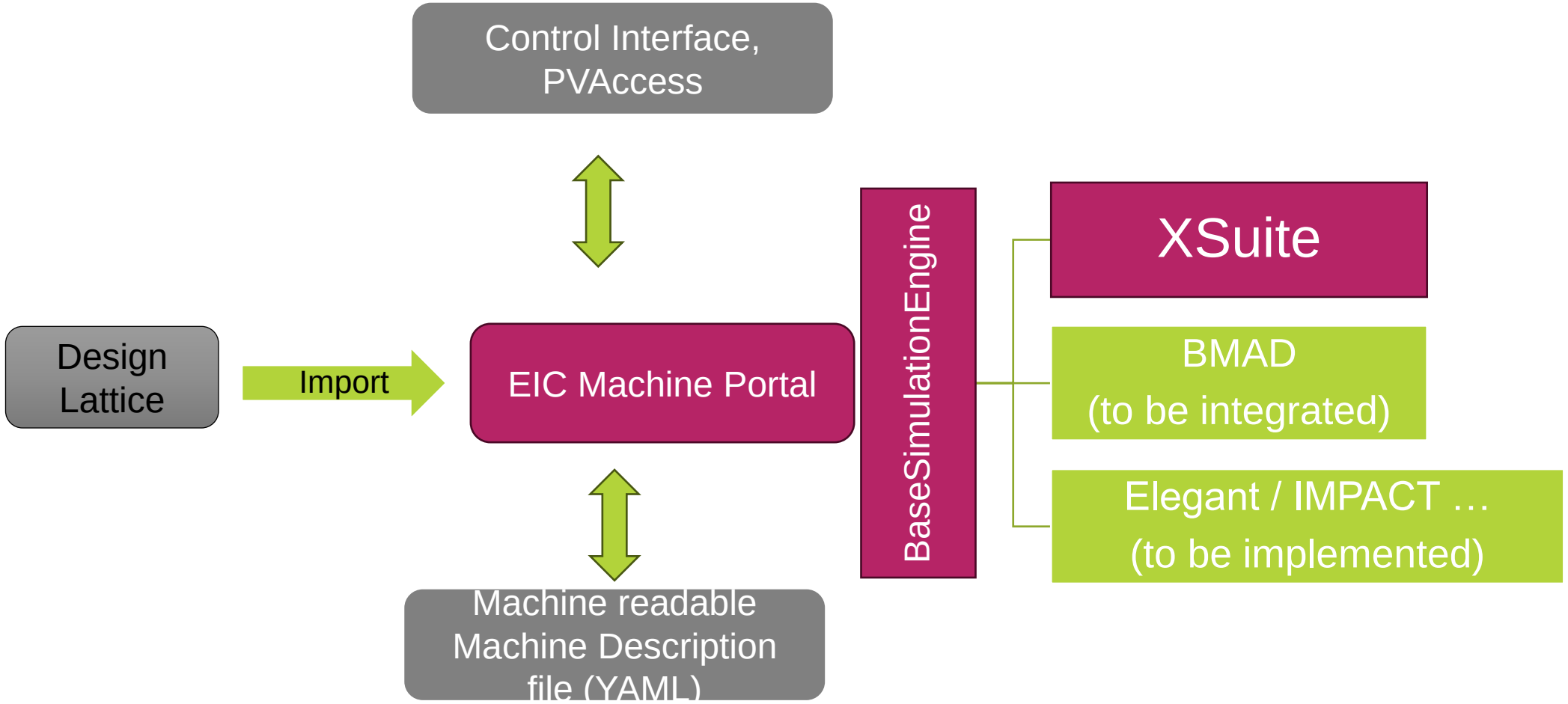


EIC VA updates I

- Package name: EICVIBE
  EIC Virtual **B**eam **E**nvironment
- Current effort : Machine Portal
  - Contains machine description that related to beam model
  - Machine description adopt PALS concept
  - Initial description imported from lattice design
- EIC element, lattice use subset of PALS
  - Export and Load from Human readable file
  - YAML is chosen
- EICVIBE, manage (create, save and load) lattice configurations and select proper simulation/modeling mode for the selected configuration.

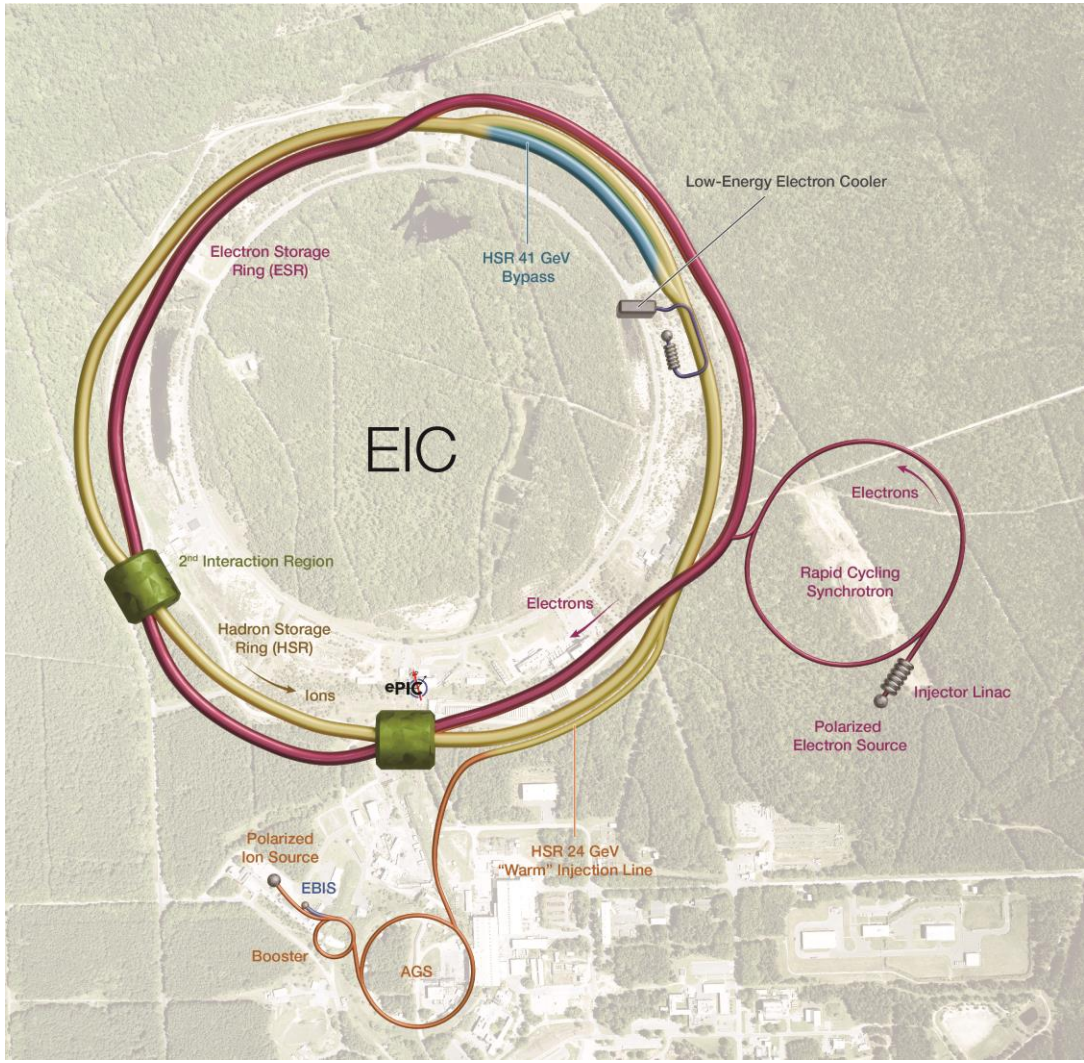


EIC VA updates II



EICVIBE simulation modes

- EIC has complicated operation modes
  - EIC electron line,
    - source → linac → BAR → RCS → ESR
    - source → linac → RCS → ESR
  - EIC proton line,
    - Existing RHIC injector → HSR
  - Many transfer lines between them
  - Need to support flexible configuration in EICVIBE
- 3 running modes
  - Single pass mode (linac, transfer line, injection, extraction)
  - Ring mode (closed orbit, optics, beam dynamics)
  - Ramping mode (RCS)



Ring mode

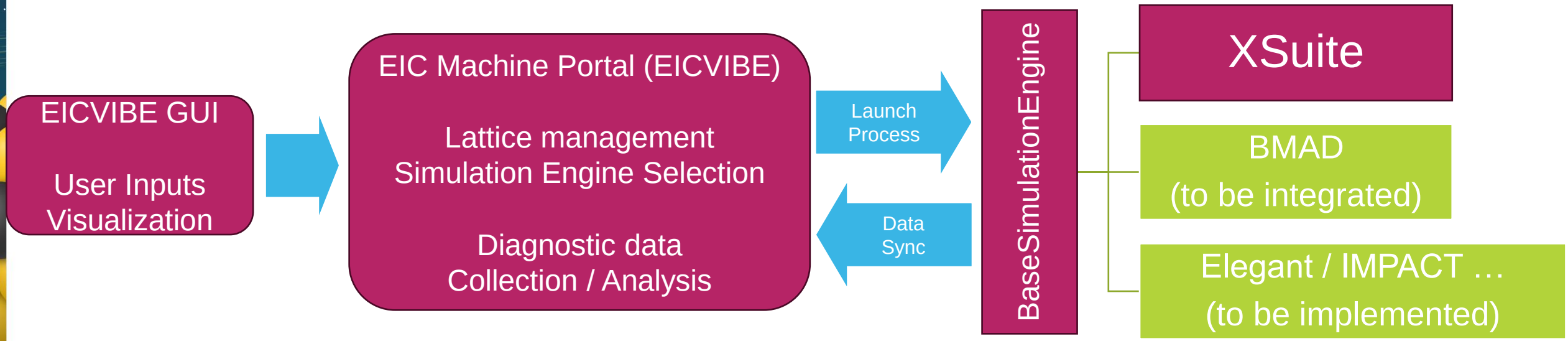
- No simulation can match the real machine synchronously yet.
- Instead, each simulation is launched from a ‘snapshot’ of current state at the launching time with N turns.
- Various simulation engines connects to the EICVIBE asynchronously.
- Shared memory for retrieving diagnostic data generated by the simulation engines without interfere the simulation itself.
- Different engines has various simulation time
  - From near real time to ~ minutes (need good surrogates for self-interactions)
- Model goal
  - Closed orbit correction,
  - Optics / crab crossing tuning
  - Prediction of Beam dynamics online optimization, Beam-Beam

Simulation Engines

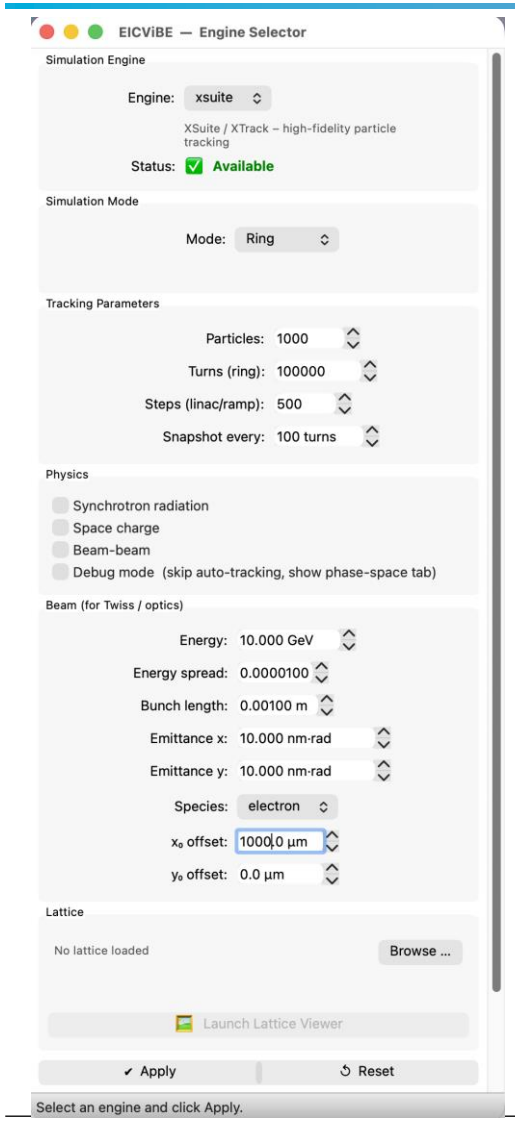
- The interface to all engines is based on BaseSimulationEngine.
  - Engine-agnostic integration with multi-engine support
  - Convert the EICVIBE lattice to engine specific interface
  - With start/stop/pause (if needed) control
  - Support all running modes, running in asynchronous loop
  - Generate and manage diagnostic data, expose structured output to the EICVIBE core
  - Maintain internal buffer for inter-engine communications (in future)
- Currently, only XSUITESimulationEngine is implemented.
- Each simulation engine runs as a separate subprocess, Managed via ZMQ IPC control channel.
- The diagnostic data is currently shared with EICVIBE core and human interfaces by shared memory for zero-copy (limited to local machine).
  - Will need to expand to ZMQ for support remote access.

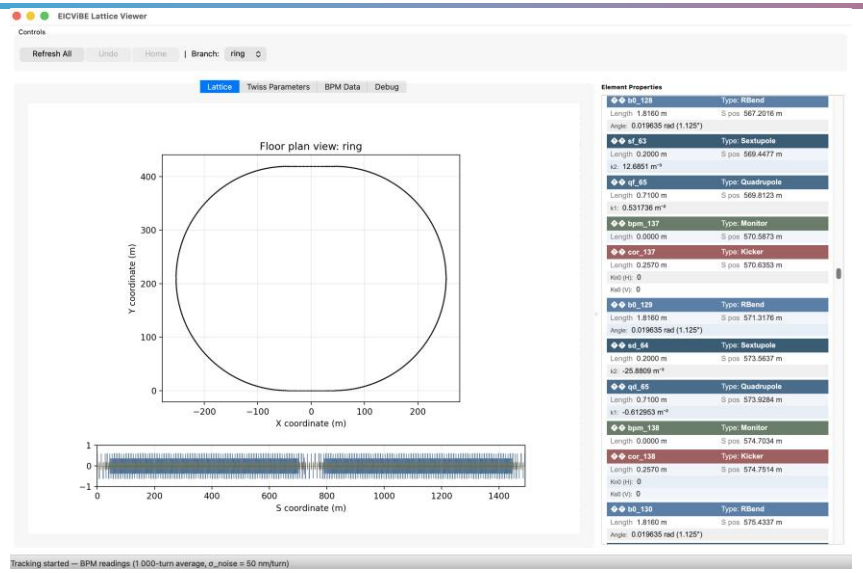
Visualization

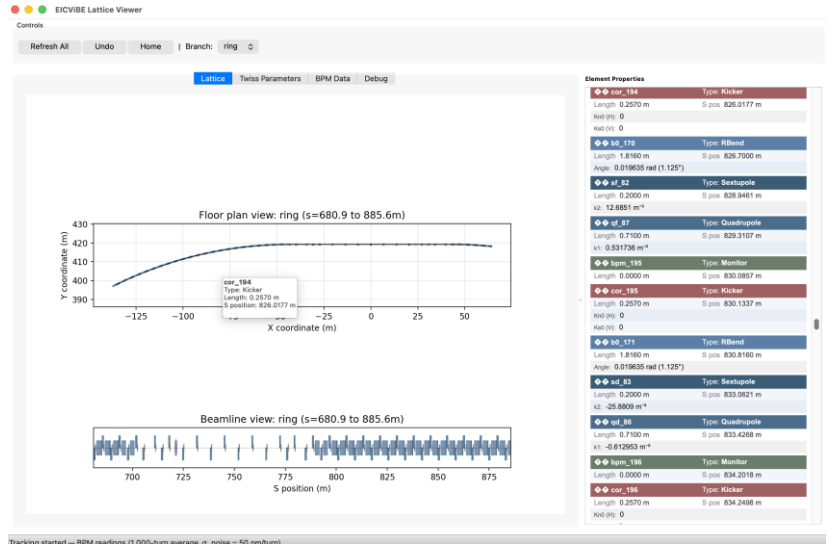
- Initially, only the Jupyter notebook is planned for user interface. Hard to benchmark all functionality
- A simple GUI is made for convenience, with PyQT5+matplotlib
- Later integrated with Control system.
- May also serve as convenient tool during construction?

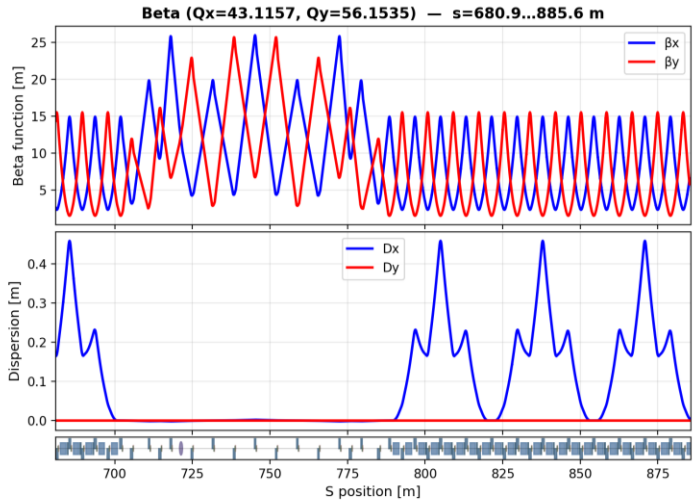


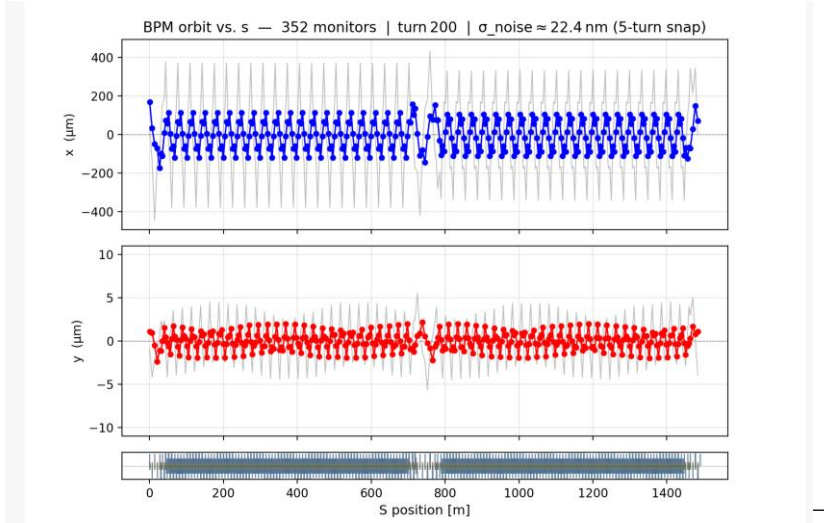
Examples with RCS lattice (Ring mode)











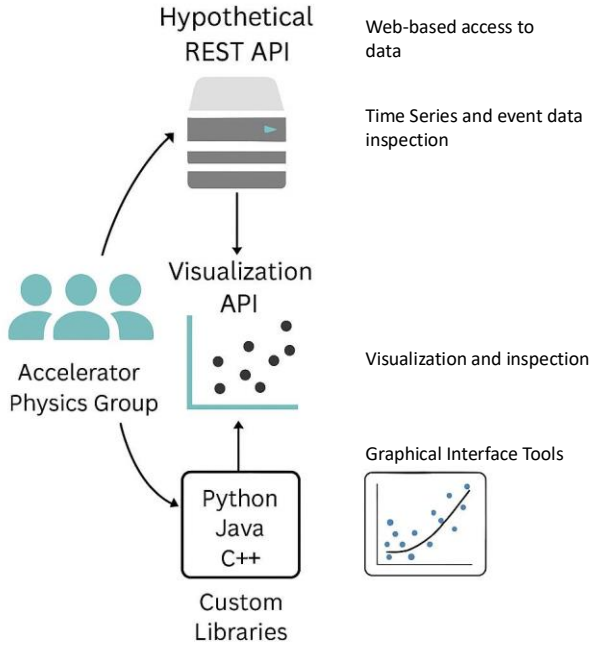
Improvement needed

- Create different machine configurations in EICVIBE
- Allow engine to run on remote machine, need to implement remote diagnostic data exchange.
  - One local engine for near-real-time predictions
  - Multiple remote engines for modeling in various time-interval, different dynamics and pre-defined fidelities.
- Add algorithm/interface to combine the results from different engines.
- Find possible work around of XSUITE engine recompiling each time the lattice is modified at EICVIBE
  - For RCS example, 3454 elements need ~60 seconds to compile.

Control integration

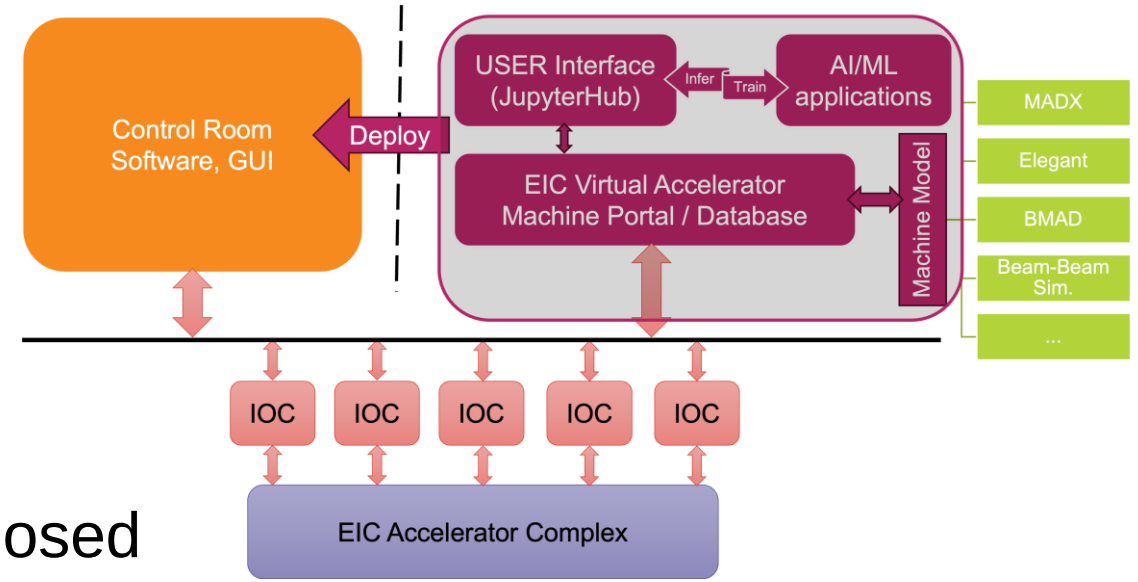
Interface Requirement	Access Method - "ICD"
The system shall expose a **web-based API for database access**, supporting queries and data retrieval.	**HTTP REST API** (GET/POST), JSON format
Time-series and event data will be available for plotting and analysis via easy-to-use tools.	**HTTP REST API for data**, with optional custom format support - eg. CSV/JSON output
Secure access will be available to **read and update control system parameters**.	**HTTP REST API** (GET/POST/PUT), secure endpoints for PV manipulation; Native API access for PV operations in supported languages; JSON-formatted requests or similar
APIs will be documented and work with **graphical tools used to build custom applications**	**HTTP REST API and native async APIs** for GUI event loop compatibility; asynchronous communication via web-sockets, signal-slot, callback, async/wait
The system will support standard plotting libraries to help **visualize data**.	**HTTP REST API** for historical data and WebSocket for streaming updates; **plotting tools** such as PyQtGraph, Matplotlib, QtChart/Charts, Swing/JavaFX charts.
The control system shall provide and maintain documentation, tooling, and libraries to support the development of custom physics applications that utilize core control interfaces	REST API, native bindings (C++, Python, Java), developer documentation
The system shall provide language-specific APIs (e.g. python, java, c++) that expose high-level functions for retrieving and manipulating data, abstracting away transport mechanisms such as HTTP.	Language specific abstraction layer (C++, Python, Java), developer documentation

Accessing Control System Data via Direct REST API or Custom Libraries



Reconsideration of AI integration

- One year ago, we proposed to have AI/ML application to be a separate layer
- Benefit from recent Genesis discussions
  - Migrating to digital twin
  - AI reasoning
  - Data readiness
  - Uncertainty Quantification



- More integrated AI components can be proposed
  - Surrogate model for expensive physics process
  - AI agent over EICVIBE api for reasoning and auto-workflow
  - Integrate optimizer (gradient-based, Bayesian)
  - AI process of diagnostic data from simulation and machine
  - ML ready data structure