



Design of an Application Programming Interface for the Program to Optimize Simulated Trajectories II

Anthony Williams and Rafael Lugo
NASA Langley Research Center

Jim Hoffman
Analytical Mechanics Associates

*2024 AIAA SciTech Forum, January 8-12 2024,
Orlando, FL*





Background



- **Program to Optimize Simulated Trajectories II (POST2) is a widely used, workhorse trajectory simulation tool that has been used to solve a variety of atmospheric ascent and entry problems**
- **Development of POST2 began in the 1970s, and has undergone continuous development and updates at NASA Langley Research Center (LaRC) since the conversion from all Fortran to C with C++ in the late 2000s**
- **Significant efforts have led to upgrading POST2 to incorporate modern programming paradigms, such as thread safe computations and increased modularity and flexibility**

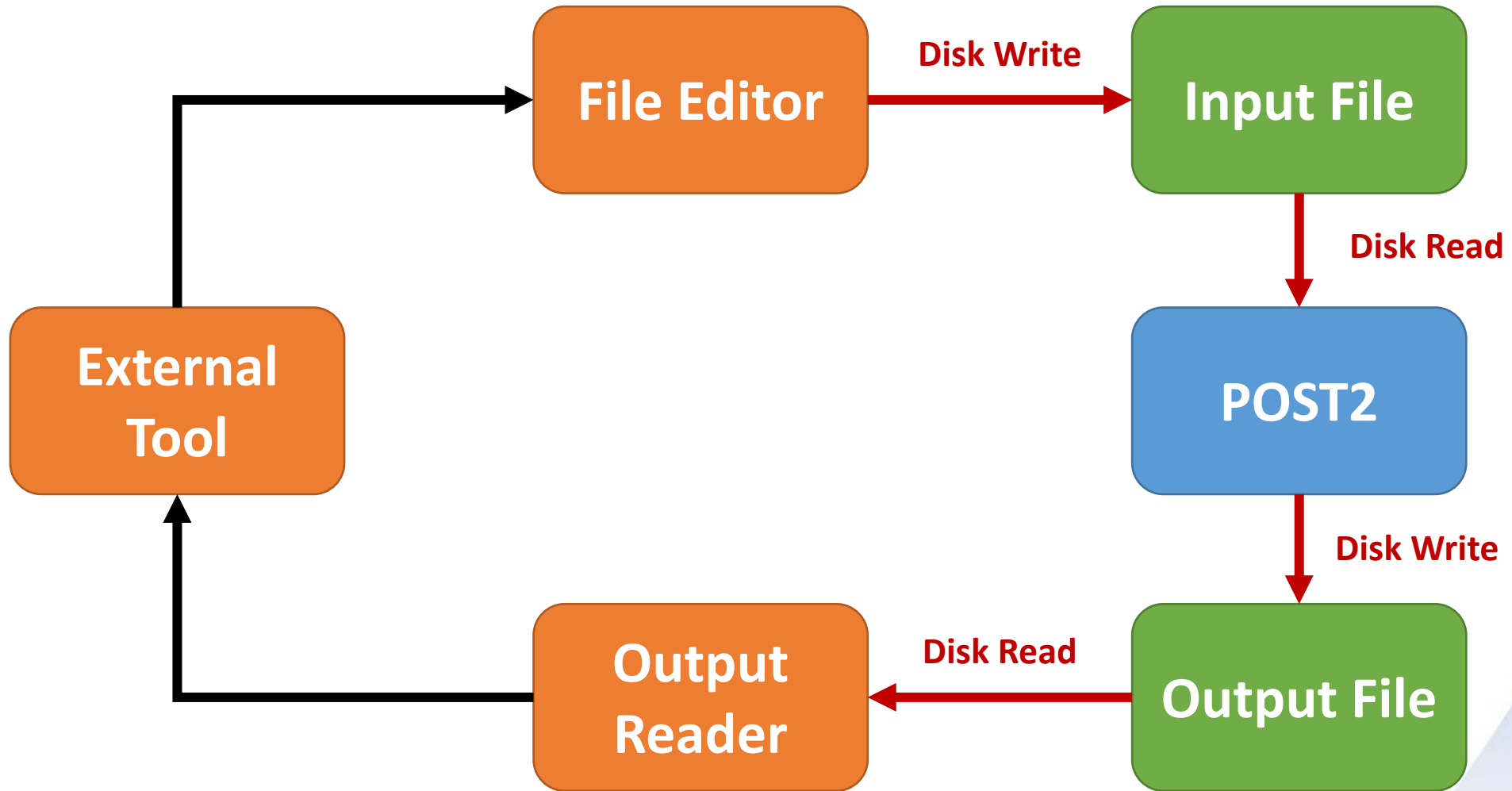


Motivation

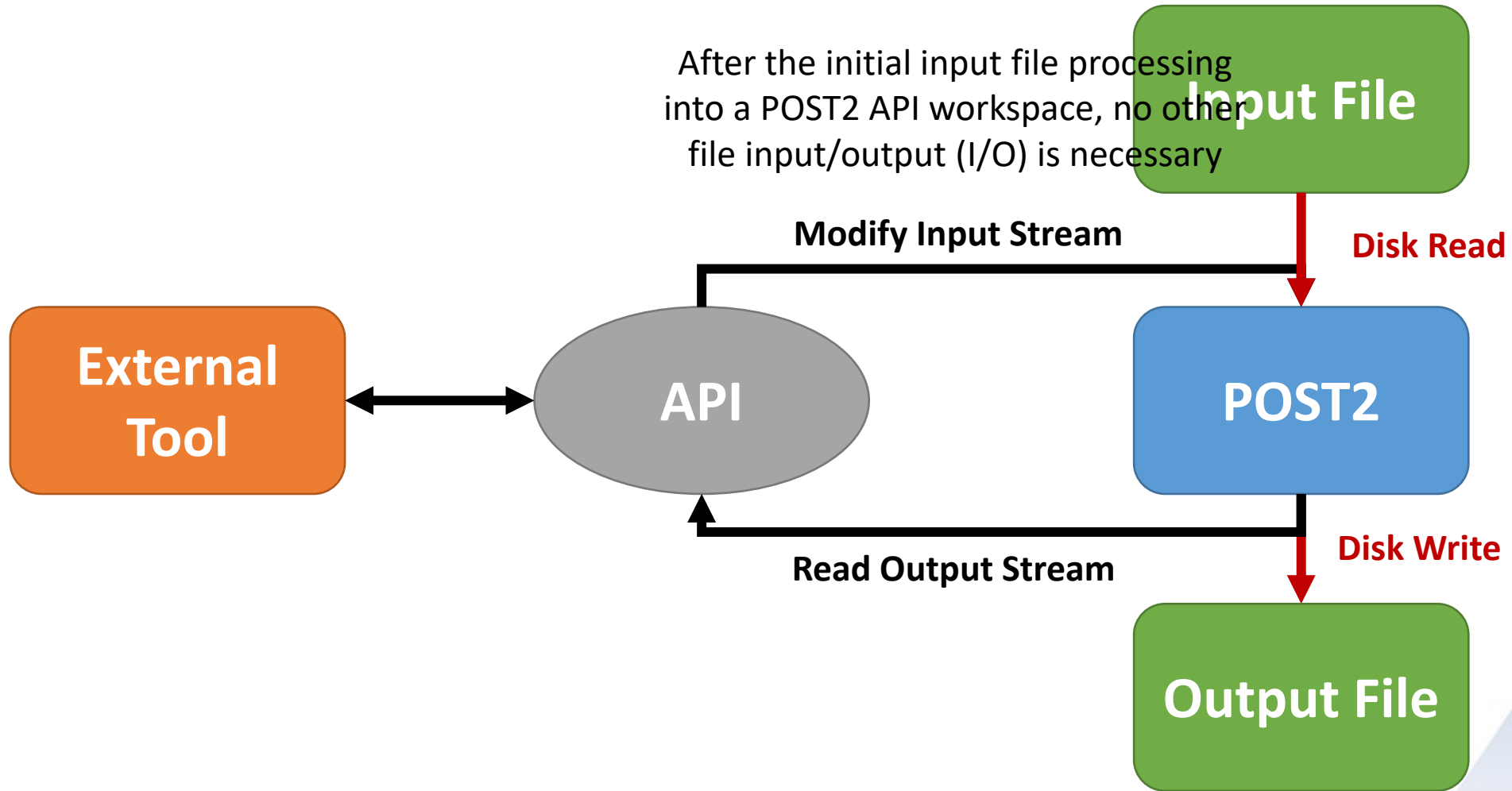


- **Meeting current and future NASA mission requirements may require multiple specialized computational tools to interact to properly assess a system**
 - Leverage the strength of each tool to maximize overall performance
 - Ex: Couple an interplanetary trajectory design tool with POST2 to allow for optimal end-to-end trajectory solutions
- **Problem: how can information be passed between applications in an accurate and efficient manner?**
- **Solution: an Application Programming Interface (API) allows tools to communicate information through pre-defined methods**

Existing Interfacing Method



Leveraging API Method





Outline



- **Design Considerations**
- **User Control**
 - Memory hierarchy
 - Simulation Settings
 - Data Output
- **Parallelism with the API**
- **Example Use Cases of the API**
- **Summary**



API Design Considerations



➤ Limited file I/O

- For accuracy purposes, no rounding or truncation
- For efficiency purposes, by passing data in memory

➤ No code duplication

- Ensure execution matches that of stand alone POST2, by utilizing existing code

➤ Enabling parallelism

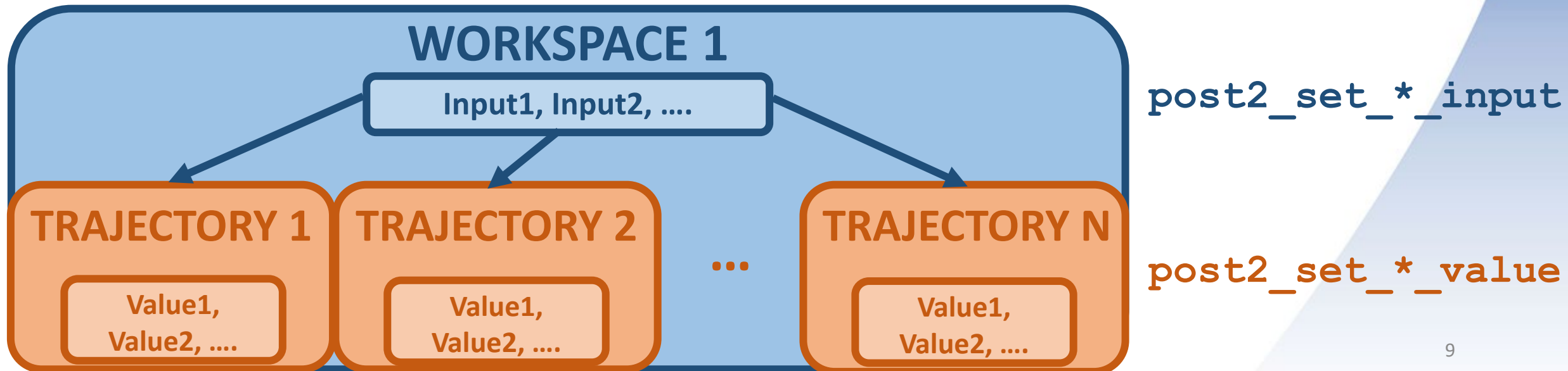
- Where possible, make data independent, allowing parallel computations when utilizing the API

- A **workspace** contains all the information on simulation setup that is defined in a POST2 input file
 - Number of available **workspaces** limited only by available memory on machine
- **Workspaces** contain one or more **trajectories** that will each follow the simulation setup for that **workspace**
 - **Trajectory** contains data for one or more vehicles to be simulated
 - Number of available **trajectories** within a **workspace** limited only by available memory on machine



User Controlled Simulation Settings

- **Setting a *workspace input* is akin to editing an input file and reprocessing it**
 - No file I/O needed, can modify the POST2 memory directly
 - All *trajectories* within *workspace* are affected by *workspace inputs*
 - Ex: set ending event for all *trajectories* to be vertical descent instead of touchdown
- **Setting a *trajectory value* affects only that specific *trajectory***
 - Ex: set integration time step for each *trajectory* to analyze affect time step size has on resultant landing location





User Controlled Data Output

User has ability to set what data should be captured and at what cadence to capture it

Set

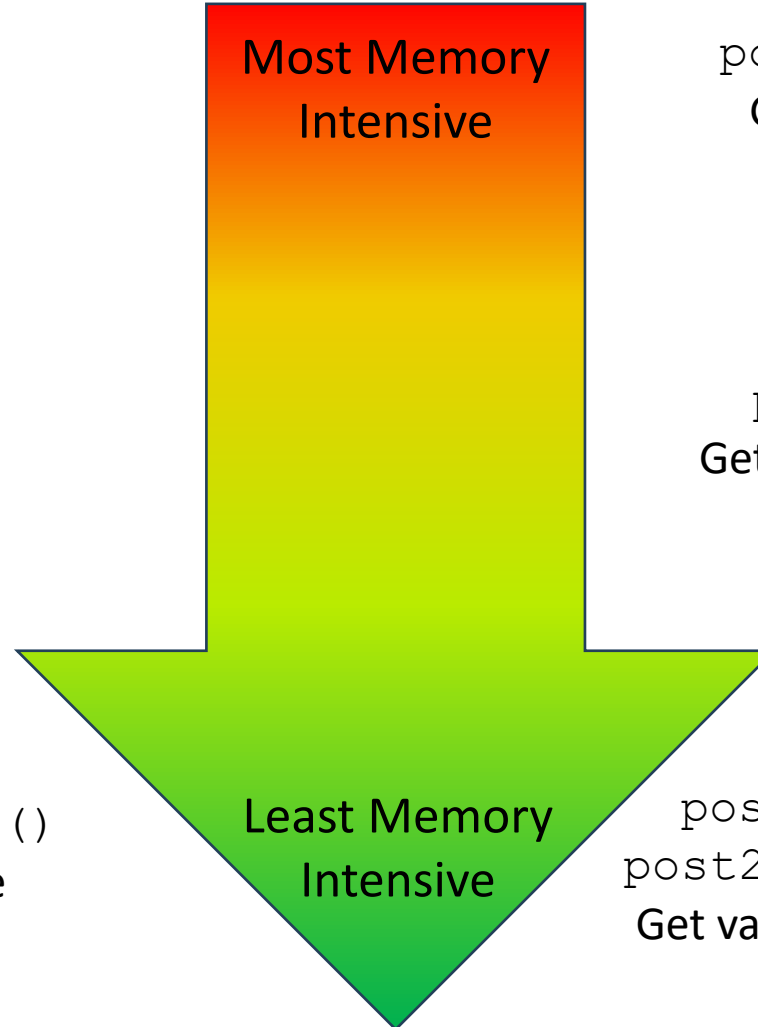
Get

`post2_set_output_at_intervals()`
Capture data at multiple of sim dt

`post2_get_value_time_history()`
Get entire time history of requested data

`post2_set_output_at_events()`
Capture data only at event boundaries

`post2_get_value_at_event()`
Get value of data at specific event, by name
or by number



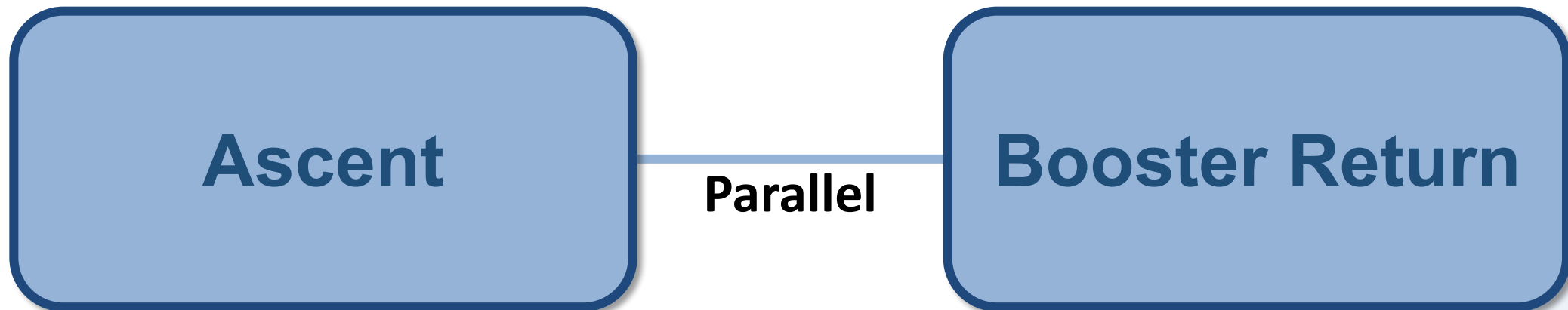
`post2_set_output_at_final_state()`
Capture data only at simulation's final state

`post2_get_value_at_timestep()` //
`post2_get_value_at_final_timestep()`
Get value of data at specific time step // final time
step



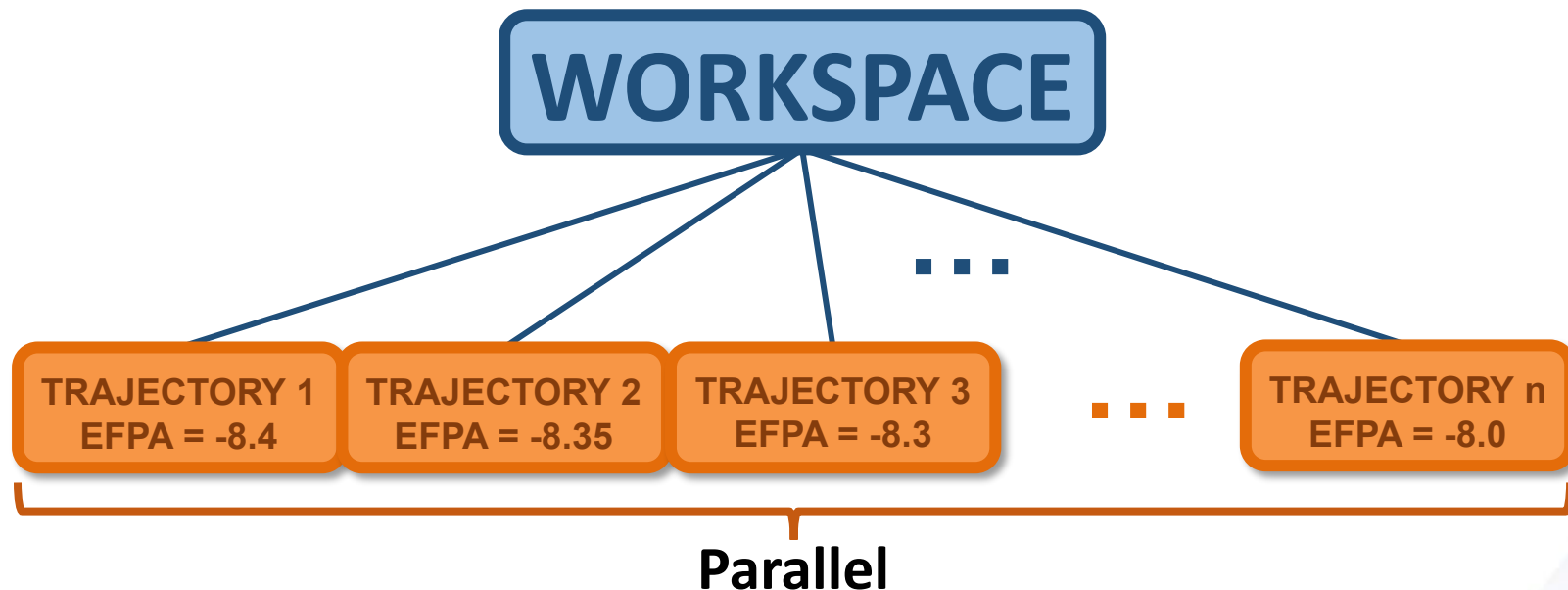
Parallelism in the API: Workspaces

- **Workspaces** can be simulated in parallel in the API
- Each **workspace** could represent a segment of flight
- Could allow for trajectory collocation to be leveraged if **workspaces** are constructed properly



Parallelism in the API: Trajectories

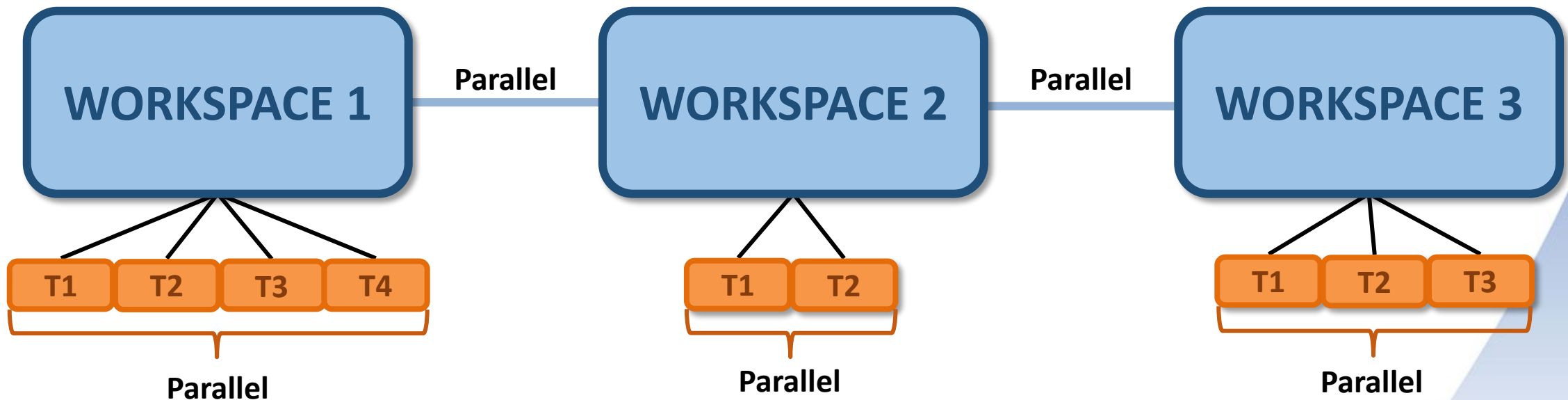
- Within each **workspace**, **trajectories** can be simulated in parallel
- Allows for rapid design space exploration (parametric sweeps)
- Can compare models side-by-side, where only simulation difference is a particular model



EFPA = Entry Flight
Path Angle

Parallelism in the API: Multi-level

- **Workspaces** can operate in parallel, that each have **trajectories** simulated in parallel within them





Example Use Cases of the API

- **Since the POST2 API is written in C, many languages and frameworks have existing methods that can interface with the POST2 API**
 - Allows for user to operate in the environment they are most productive in
- **Additionally, all inputs and outputs from API methods are primitive C data types, to allow for easier interoperability**
- **Two common platforms used in engineering will be highlighted: Python and MATLAB**



Example Use Cases of the API: Python



- **C Foreign Function Interface (CFFI) package was used to create callable POST2 API module in Python**
 - Provides a means to operate entirely in a Python process
- **Leverage visualization and scientific computing libraries (such as `matplotlib`, `numpy`, etc.)**
- **Can also leverage multi-threading capabilities within Python**



Example Use Cases of the API: Python



- Common use case might be a parametric sweep
- Single **workspace**, multiple **trajectories** sweeping through entry flight path angle (EFPA)





Example Use Cases of the API: MATLAB



- `calllib` used in MATLAB to provide method to call POST2 API functions
- While POST2 has flexible built-in optimizers, no one optimizer is one-size-fits-all
 - External optimizers can be leveraged via the API, such as `fmincon` in MATLAB
- Visualize feedback on data provided by the optimizer



Example Use Cases of the API: MATLAB

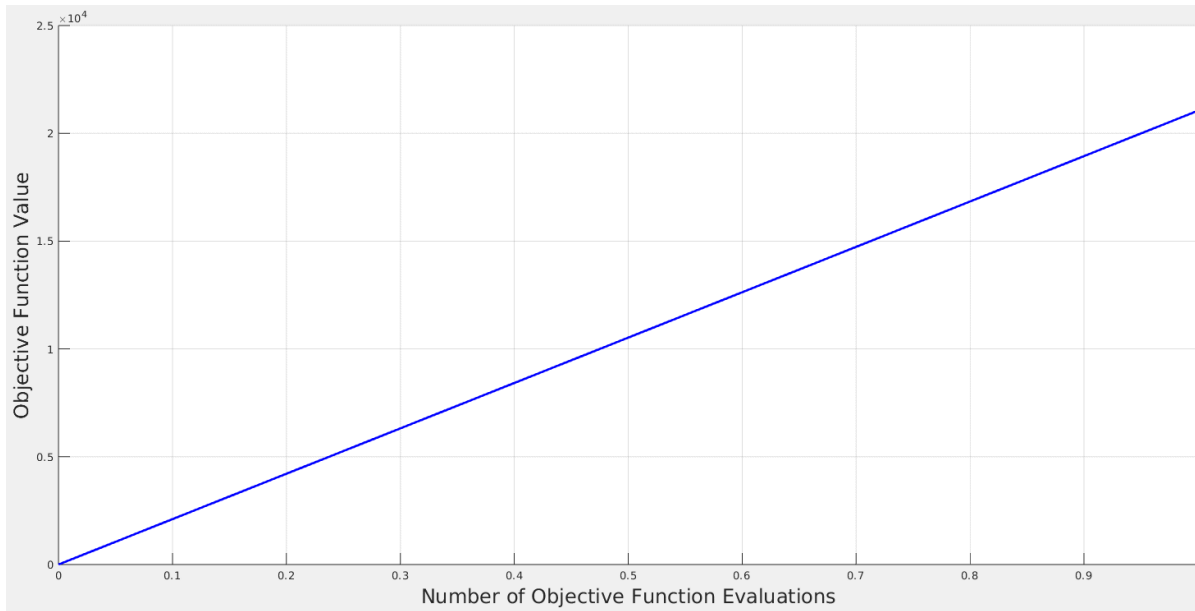


- Targeting problem for powered descent solved using `fmincon`
- Controls: engine throttle, time of flight, and flight path angle
- Constraints: vertical and horizontal velocity, and geodetic altitude

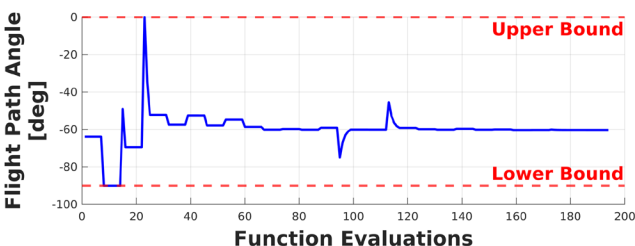
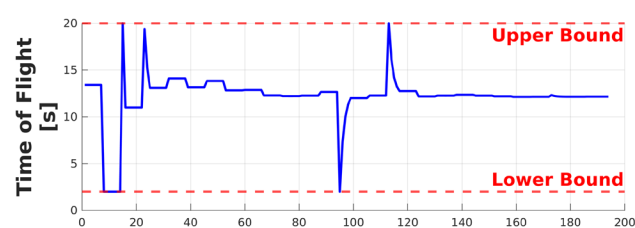
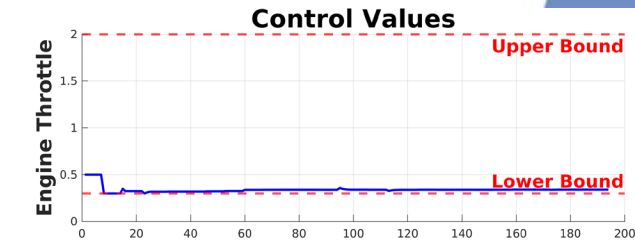
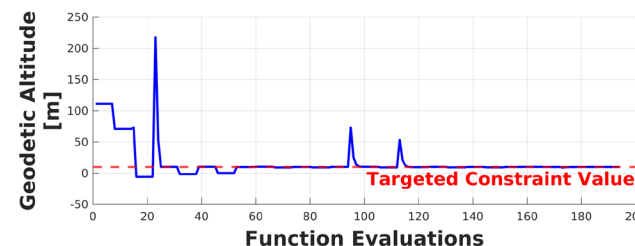
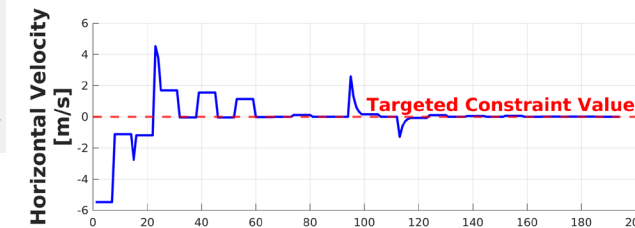
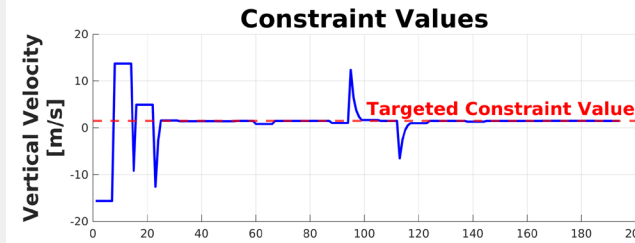
Example Use Cases of the API: MATLAB

- Targeting problem for powered descent solved using `fmincon`
- Controls: engine throttle, time of flight, and flight path angle
- Constraints: vertical and horizontal velocity, and geodetic altitude

Post-convergence feedback on constraints and controls



Real-time feedback of optimization solution convergence



- **An application programming interface was developed for a widely used flight mechanics tool, POST2**
- **Allows for interoperability with external applications, with Python and MATLAB demonstrated, to maximize performance of tools by leveraging the strength of each**
 - Any language, framework, or application that can interface with C is able to be coupled with the POST2 API
- **API will be part of next POST2 release,**
<https://www.nasa.gov/post2/requesting>





Backup

