

Conceptual Aircraft Design in OpenMDAO

Eytan Adler

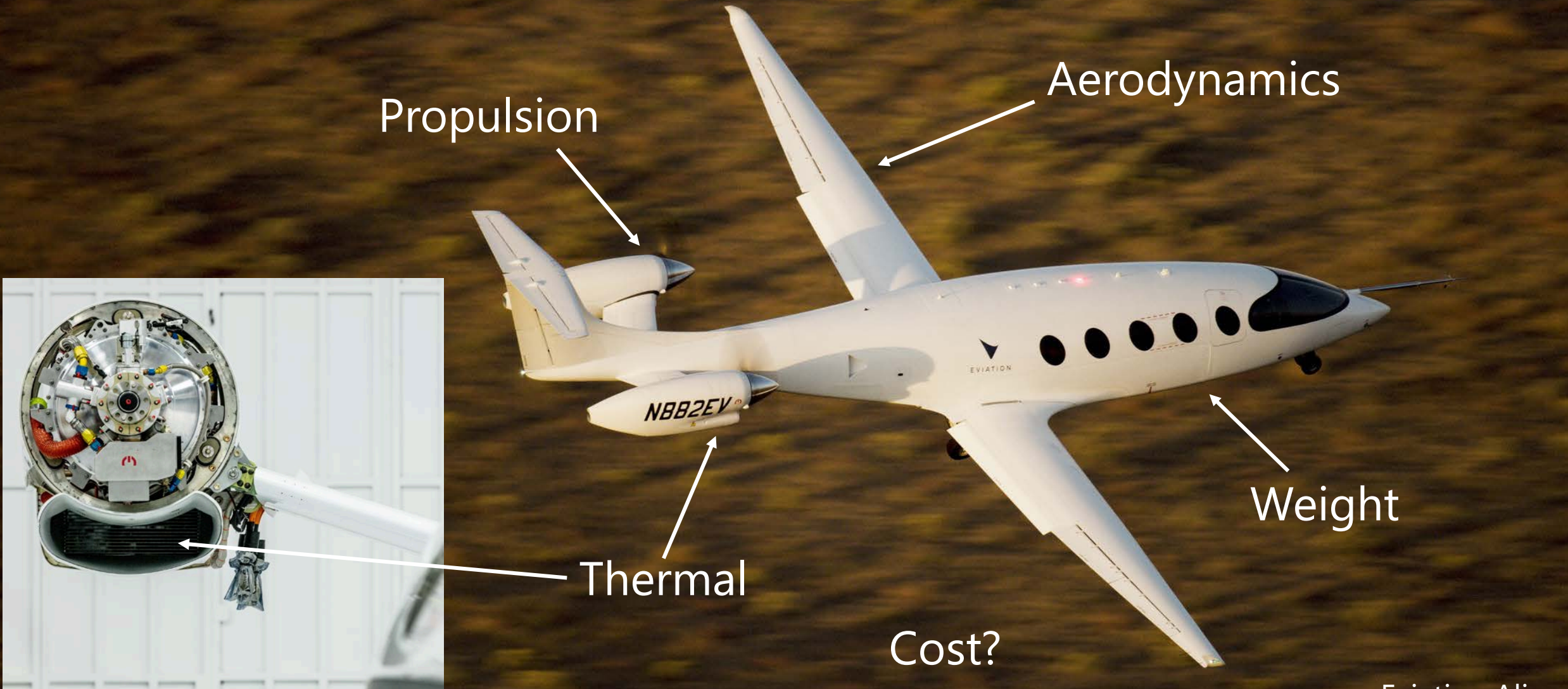
2022 OpenMDAO Workshop

NASA Glenn

October 24–25th



Conceptual design is multidisciplinary



Eviation Alice

OpenConcept

Open-source conceptual aircraft design tool built on OpenMDAO

The screenshot shows the GitHub repository for 'mdolab/openconcept'. The repository is public and has 21 stars and 24 forks. The main branch is 'main'. The repository description is 'OpenConcept: A toolkit for conceptual MDAO of aircraft with unconventional propulsion architectures'. The repository includes a README, a LICENSE, and a setup.py file. The README is titled 'OpenConcept - A conceptual design toolkit with efficient gradients implemented in the OpenMDAO framework' and lists the authors as Benjamin J. Brelje and Eytan J. Adler. The README also includes a table of badges for build status, code coverage, documentation status, Python version, and download statistics. The README text describes OpenConcept as a new toolkit for the conceptual design of aircraft, built on top of NASA Glenn's OpenMDAO framework. It mentions that OpenConcept is capable of modeling a wide range of propulsion systems, including detailed thermal management systems. The README includes a link to a paper and a reference to a specific example file. Below the README text is a diagram of a parallel hybrid turbofan engine system. The diagram shows a ducted heat exchanger, a variable exit duct, a pump, a chiller, a battery, a bypass, and a parallel hybrid turbofan. The turbofan is labeled with 'Electric motor' and 'Fault protection'. The diagram is a schematic representation of the engine's thermal and mechanical components and their interactions.

mdolab / openconcept Public

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 6 tags

Go to file Add file Code

About

OpenConcept: A toolkit for conceptual MDAO of aircraft with unconventional propulsion architectures

Readme MIT license 21 stars 4 watching 24 forks

Releases 9

OpenConcept 1.0.0 Latest on Aug 11 + 8 releases

Packages

No packages published Publish your first package

Contributors 5

Languages

Python 100.0%

eytanadler Changed image links in readme to absolute URLs ✓ 8a7661f on Aug 11 157 commits

File	Description	Time
.github	Docs overhaul and refactor to organize code (#41)	2 months ago
doc	Formatting and linting (#42)	2 months ago
openconcept	Bumped version to 1.0.0	2 months ago
.coveragerc	Another attempt at coverage testing	4 years ago
.git-blame-ignore-revs	Added ignore blame for formatting changes	2 months ago
.gitignore	Formatting and linting (#42)	2 months ago
.readthedocs.yml	Formatting and linting (#42)	2 months ago
LICENSE	Initial commit	4 years ago
pytest.ini	Ignore all pytest warnings related to deprecating the numpy matrix cl...	3 years ago
readme.md	Changed image links in readme to absolute URLs	2 months ago
setup.py	Formatting and linting (#42)	2 months ago

readme.md

OpenConcept - A conceptual design toolkit with efficient gradients implemented in the OpenMDAO framework

Authors: Benjamin J. Brelje and Eytan J. Adler

Build passing codecov 81% docs passing pypi v1.0.0 downloads 46/month

OpenConcept is a new toolkit for the conceptual design of aircraft. OpenConcept was developed in order to model and optimize aircraft with electric propulsion at low computational cost. The tools are built on top of NASA Glenn's [OpenMDAO](#) framework, which in turn is written in Python.

OpenConcept is capable of modeling a wide range of propulsion systems, including detailed thermal management systems. The following figure (from [this paper](#)) shows one such system that is modeled in the `N3_HybridSingleAisle_Refrig.py` example.

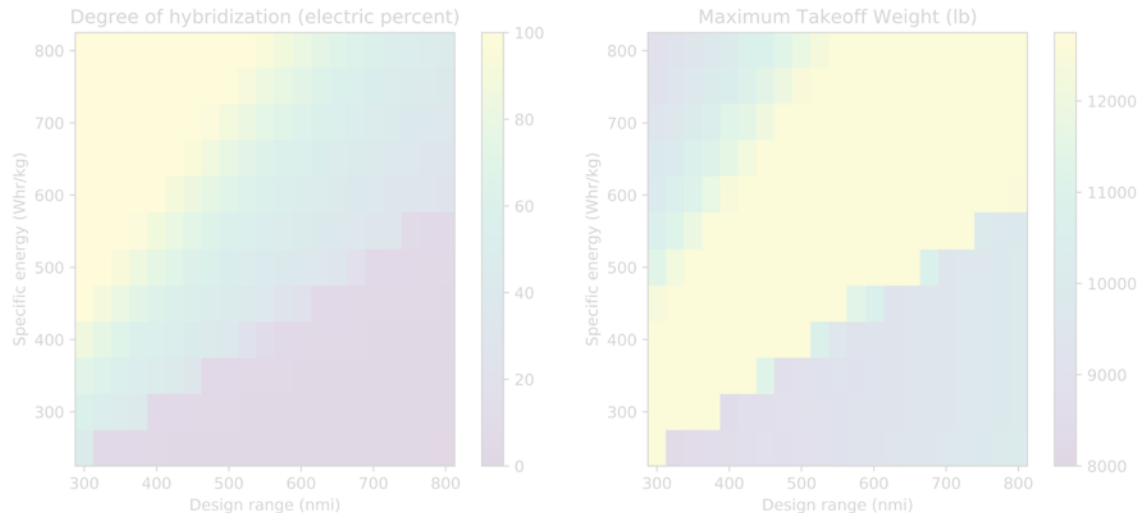
The diagram illustrates a parallel hybrid turbofan engine system. It features a ducted heat exchanger, a variable exit duct, a pump, a chiller, a battery, a bypass, and a parallel hybrid turbofan. The turbofan is labeled with 'Electric motor' and 'Fault protection'. The diagram shows the flow of air and the integration of the battery and chiller into the engine's thermal management system.

Who is "we"?



Ben Brelje

Hybrid electric aircraft design optimization



minimize: fuel burn + 0.01MTOW

by varying:

MTOW

S_{ref}

d_{prop}

$W_{battery}$

P_{motor} (rated)

$P_{turboshaft}$ (rated)

$P_{generator}$ (rated)

H_E (degree of hybridization w.r.t energy)

subject to scalar constraints:

$$R_{TOW} = W_{TO} - W_{fuel} - W_{empty} - W_{payload} - W_{batt} \geq 0$$

$$R_{batt} = E_{batt,max} - E_{batt,used} \geq 0$$

$$R_{vol} = W_{fuel,max} - W_{fuel} \geq 0$$

$$BFL \leq 4452 \text{ ft (no worse than baseline)}$$

$$V_{stall} \leq 81.6 \text{ kt (no worse than baseline)}$$

and vector constraints:

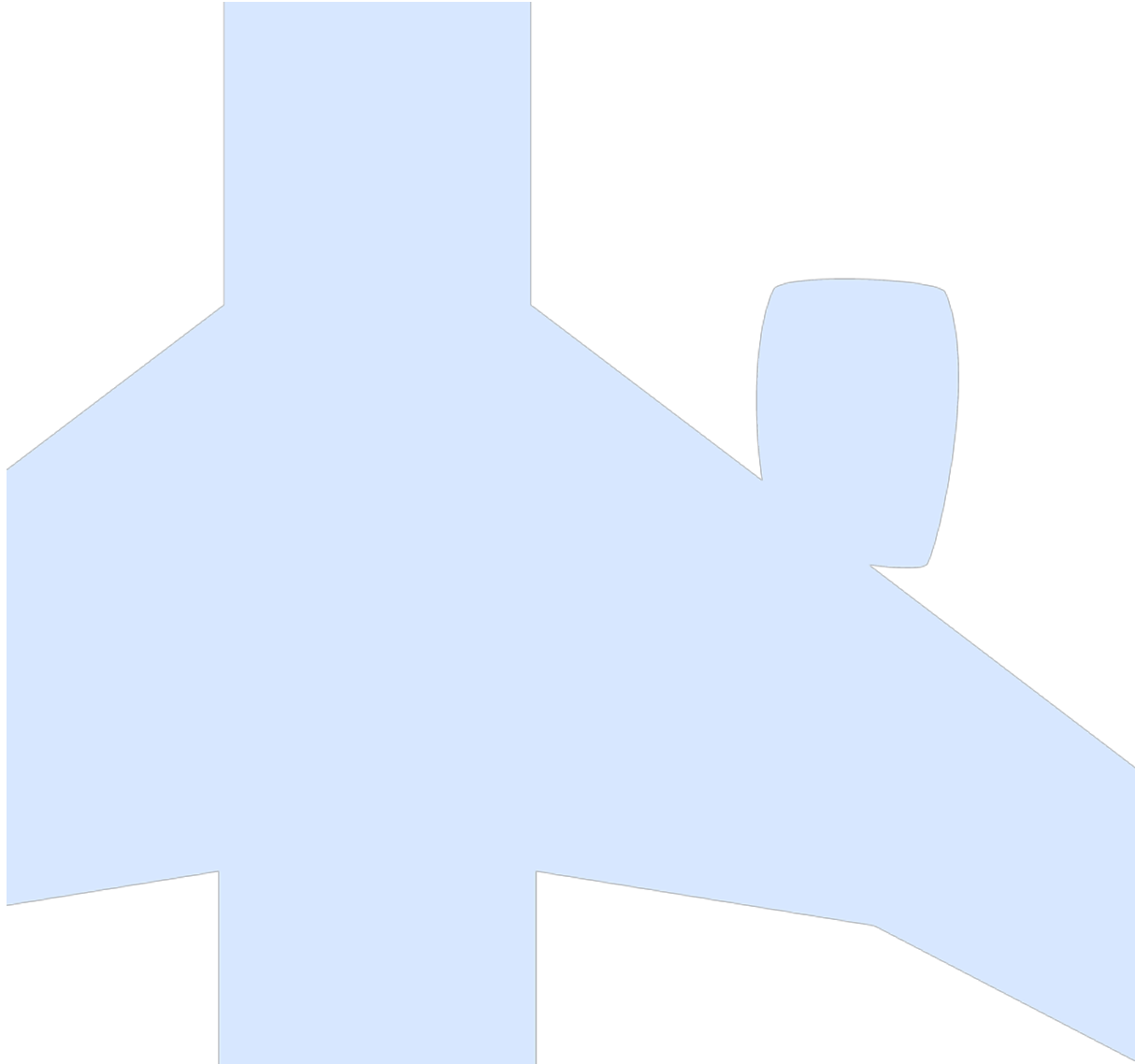
$$\vec{P}_{motor} \leq 1.05 P_{motor} \text{ (rated)}$$

$$\vec{P}_{turboshaft} \leq P_{turboshaft} \text{ (rated)}$$

$$\vec{P}_{generator} \leq P_{generator} \text{ (rated)}$$

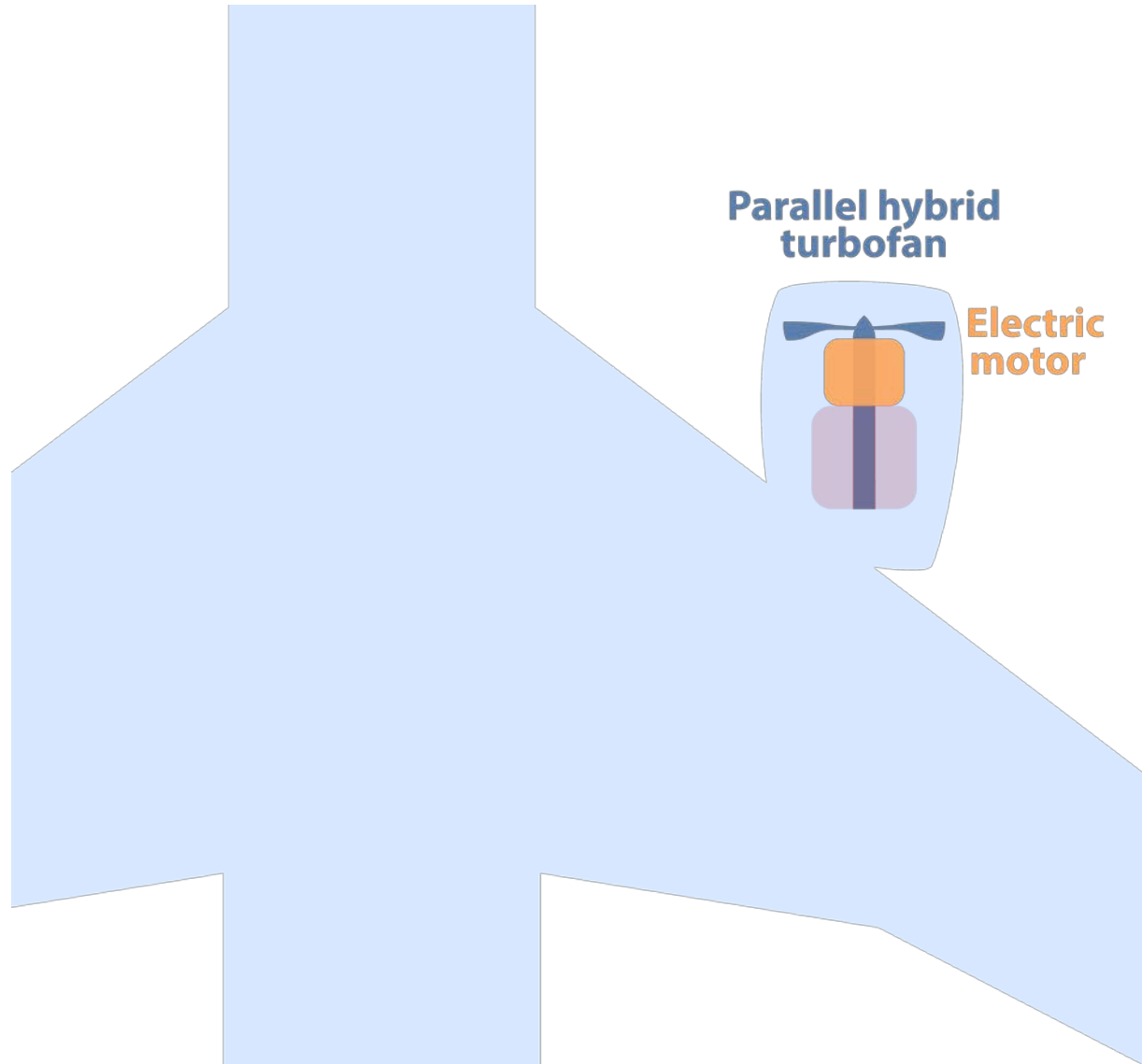
$$\vec{P}_{battery} \leq W_{battery} \cdot pb$$

But we can do a lot more...



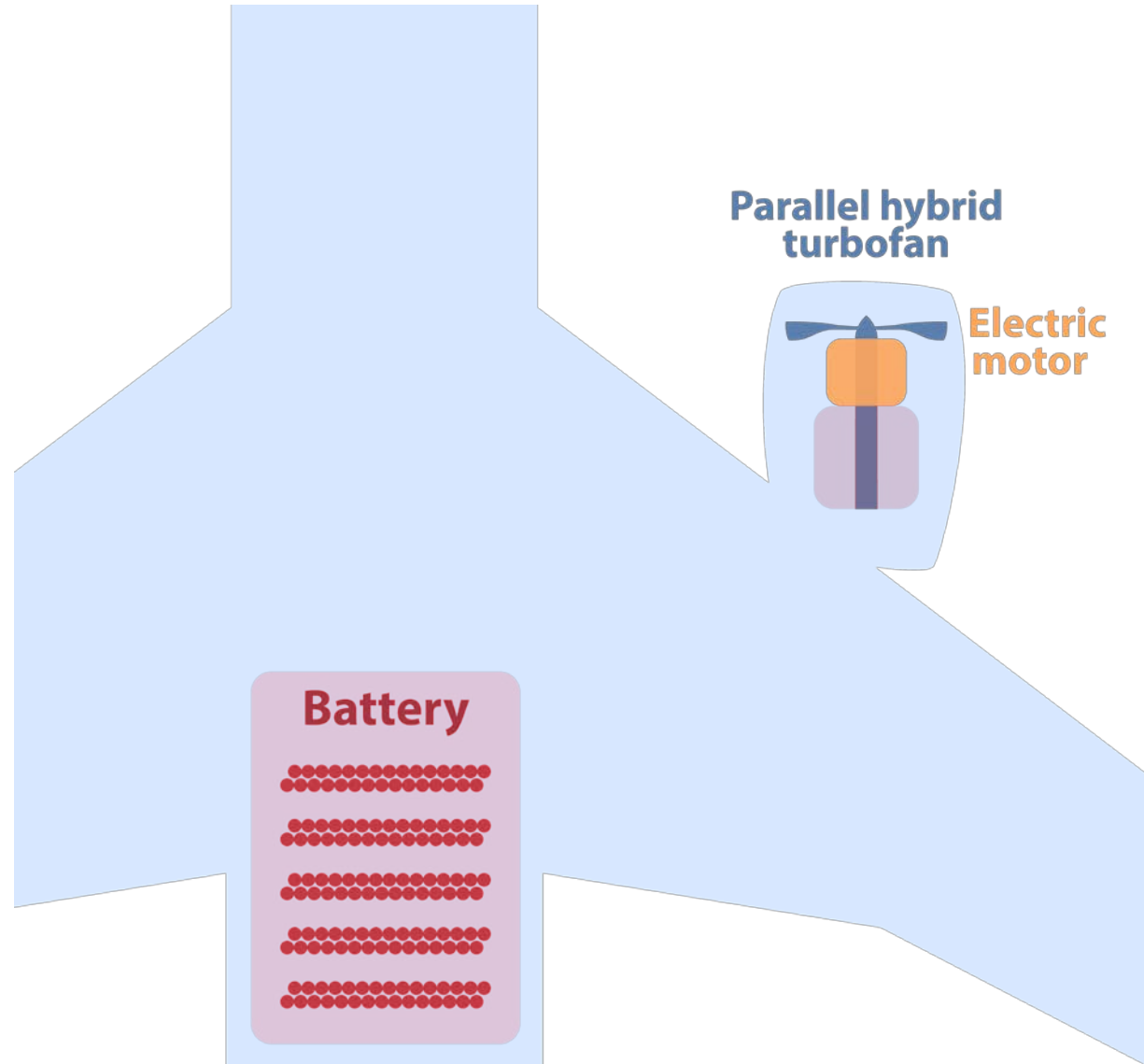
Adler, Brelje, and Martins, "Thermal Management System Optimization for a Parallel Hybrid Aircraft Considering Mission Fuel Burn", 2022.

Start with a parallel hybrid turbofan



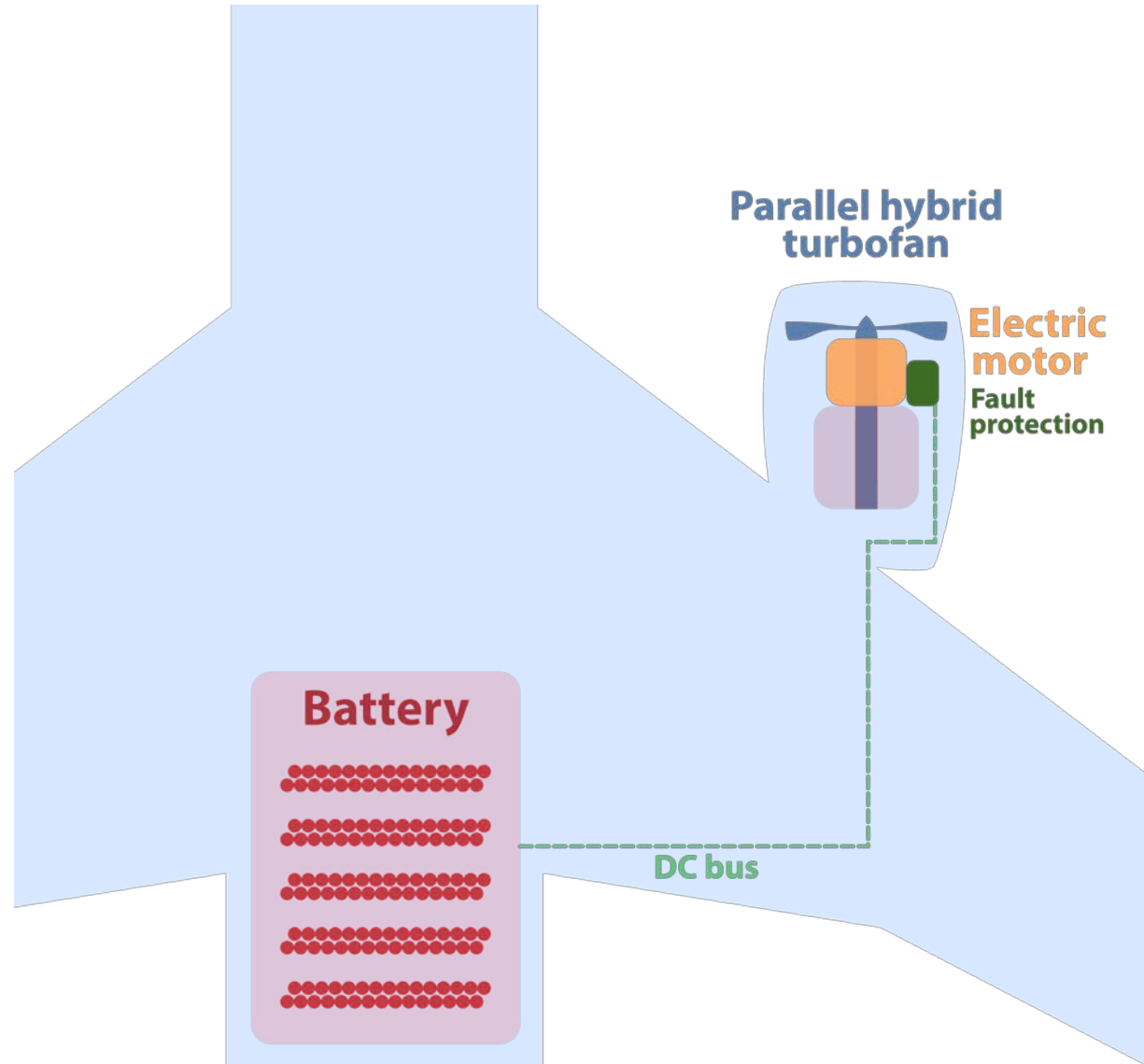
Adler, Brelje, and Martins, "Thermal Management System Optimization for a Parallel Hybrid Aircraft Considering Mission Fuel Burn", 2022.

Battery powers the electric motor



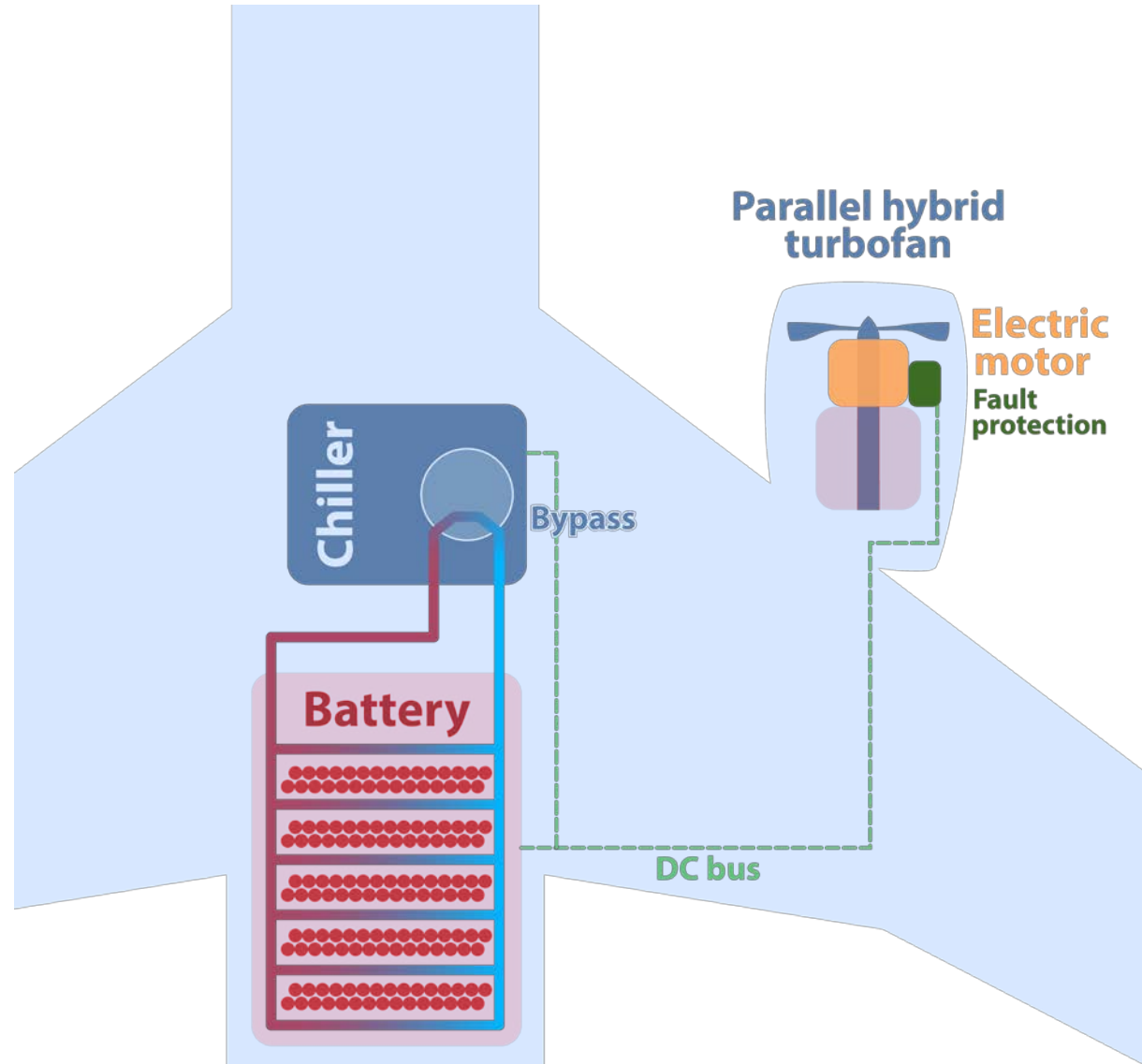
Adler, Brelje, and Martins, "Thermal Management System Optimization for a Parallel Hybrid Aircraft Considering Mission Fuel Burn", 2022.

Battery powers the electric motor



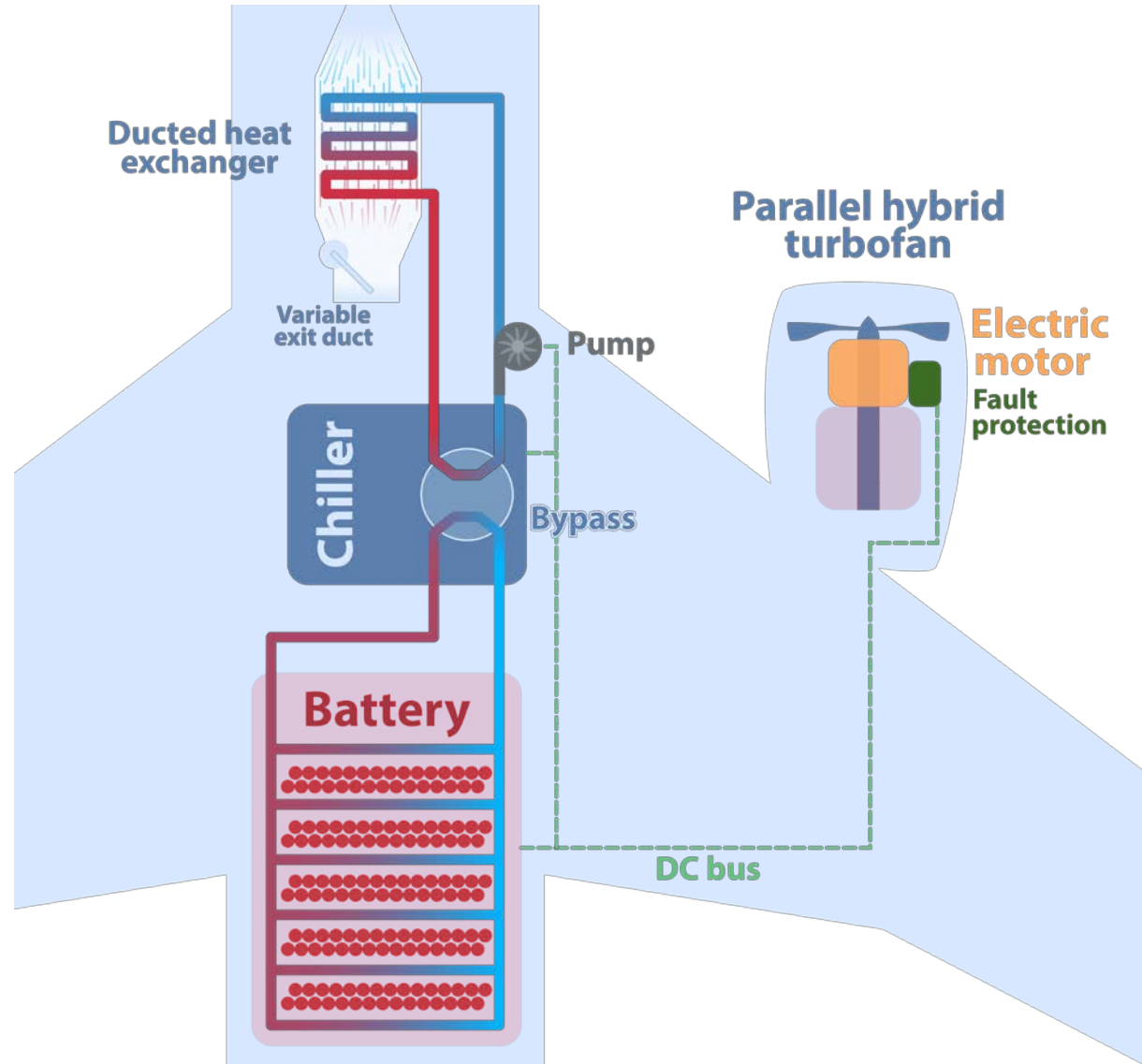
Adler, Brelje, and Martins, "Thermal Management System Optimization for a Parallel Hybrid Aircraft Considering Mission Fuel Burn", 2022.

Cool the battery with a refrigerator



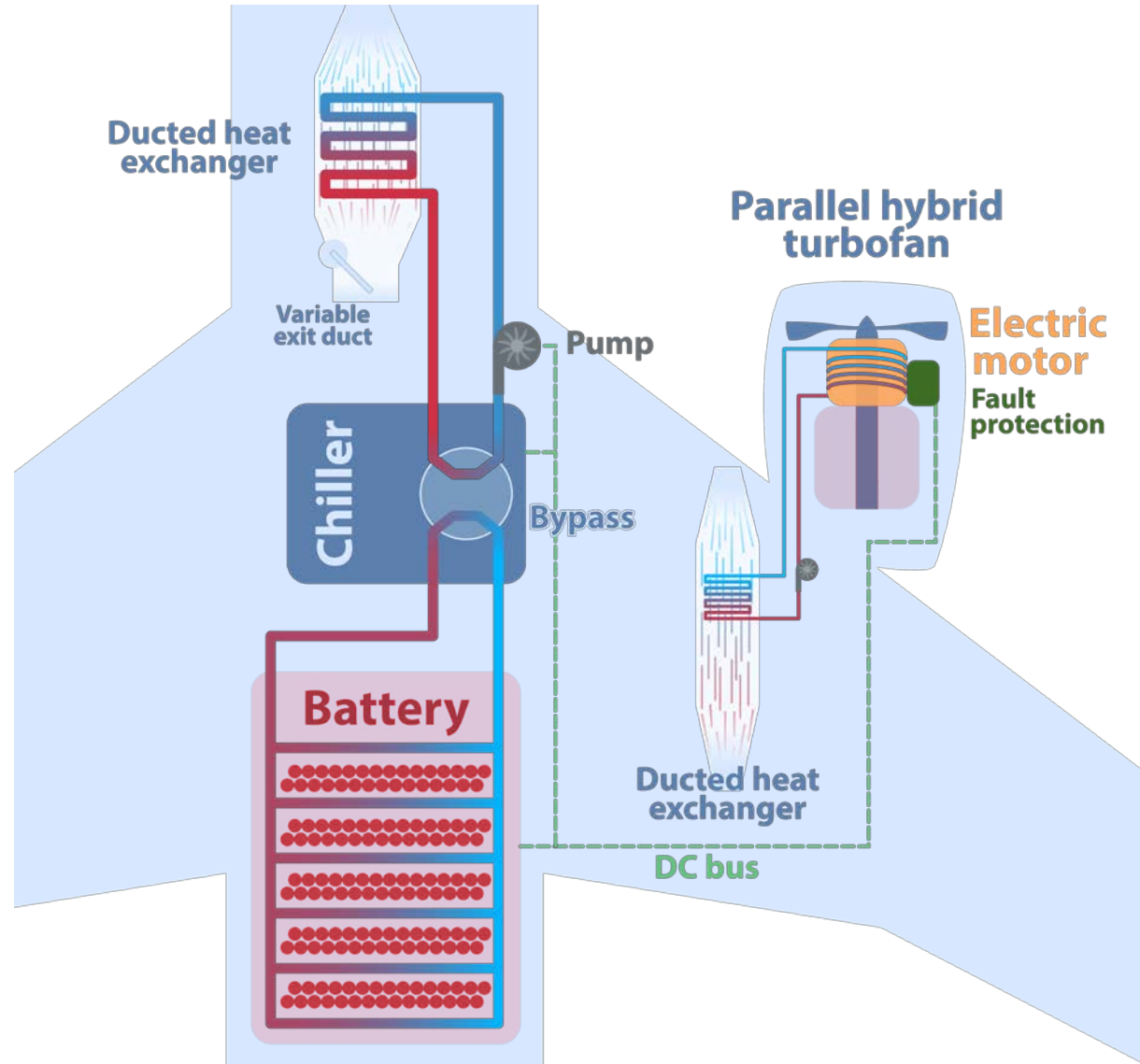
Adler, Brelje, and Martins, "Thermal Management System Optimization for a Parallel Hybrid Aircraft Considering Mission Fuel Burn", 2022.

Refrigerator dumps heat into freestream



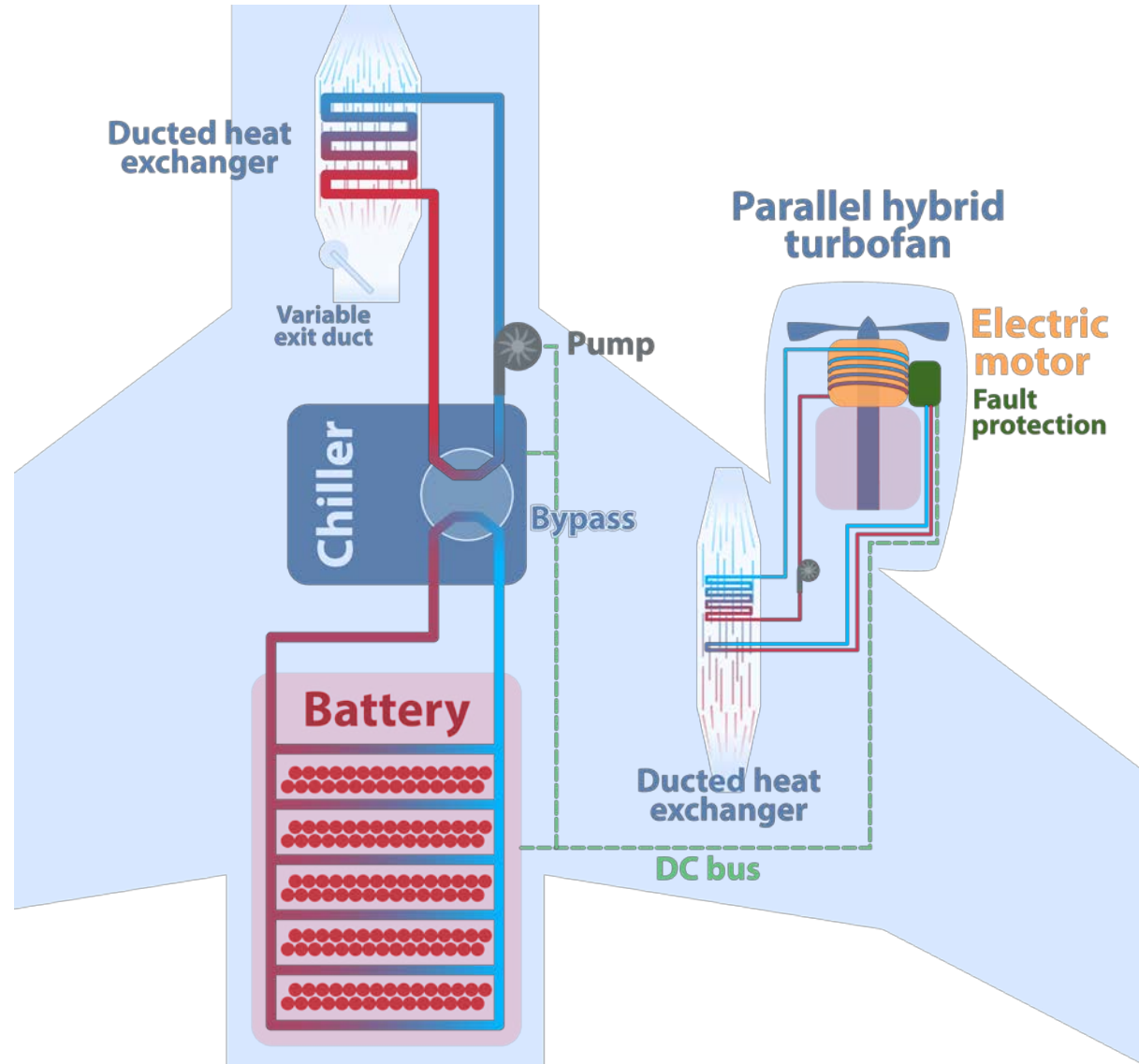
Adler, Brelje, and Martins, "Thermal Management System Optimization for a Parallel Hybrid Aircraft Considering Mission Fuel Burn", 2022.

Same for the electric motor



Adler, Brelje, and Martins, "Thermal Management System Optimization for a Parallel Hybrid Aircraft Considering Mission Fuel Burn", 2022.

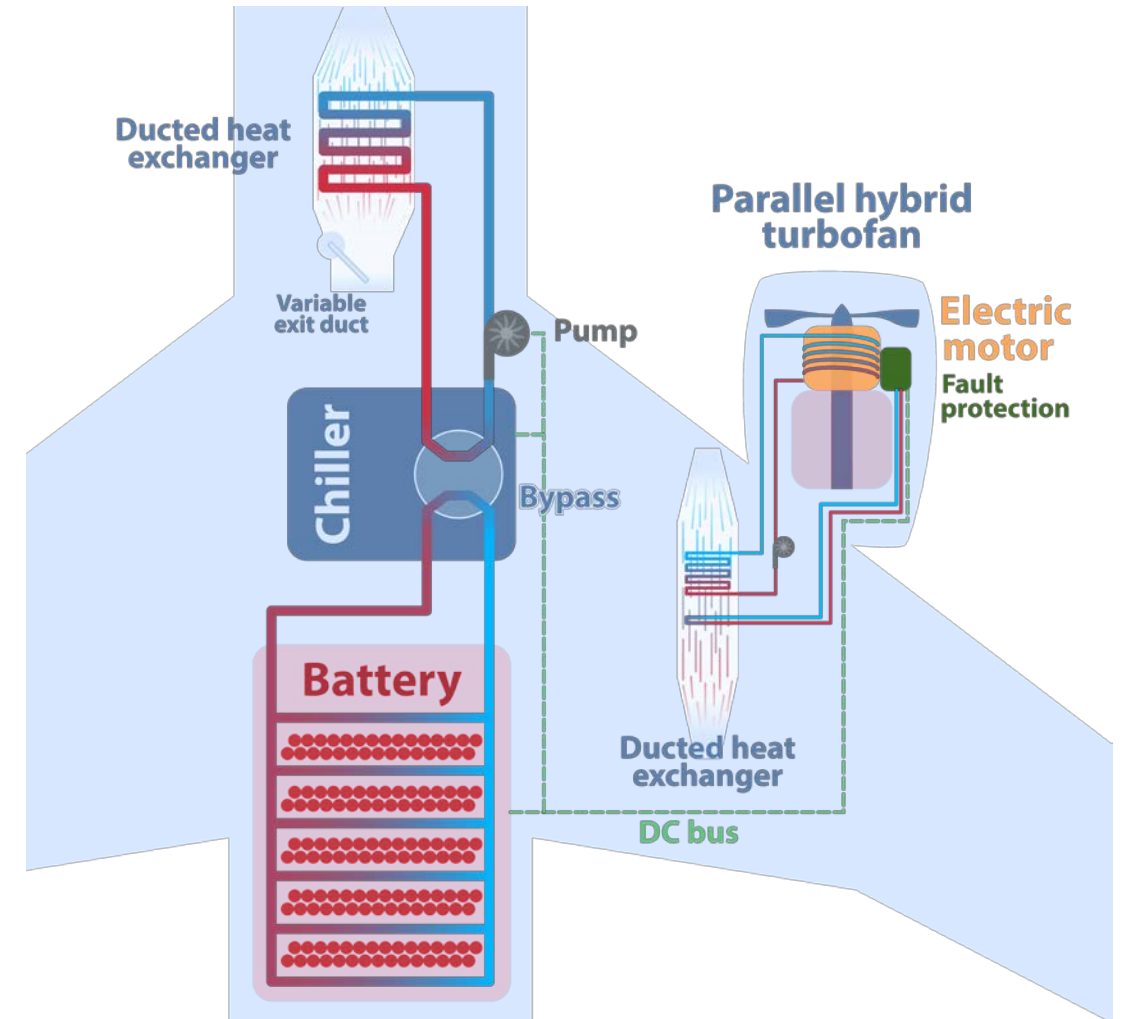
...and the fault protection



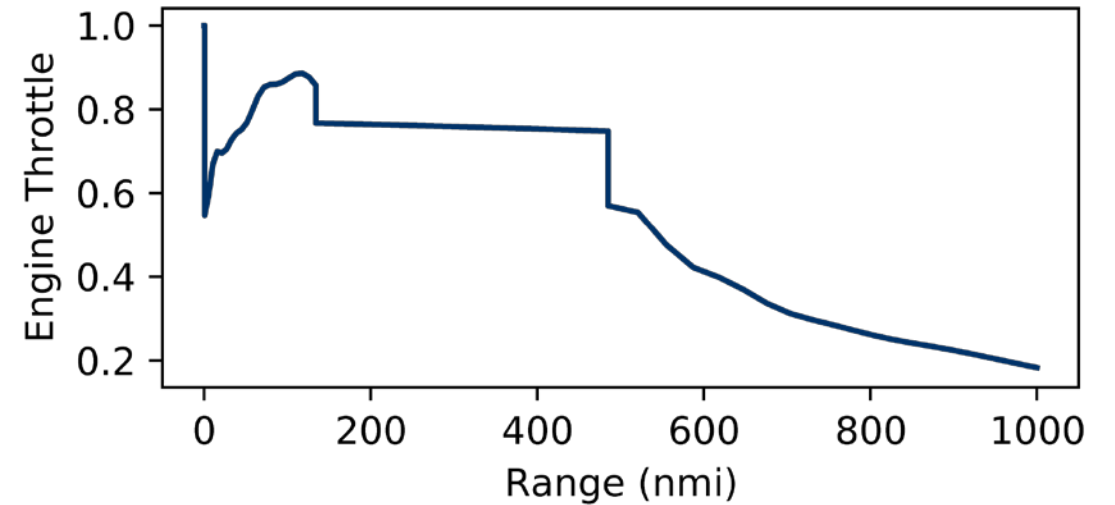
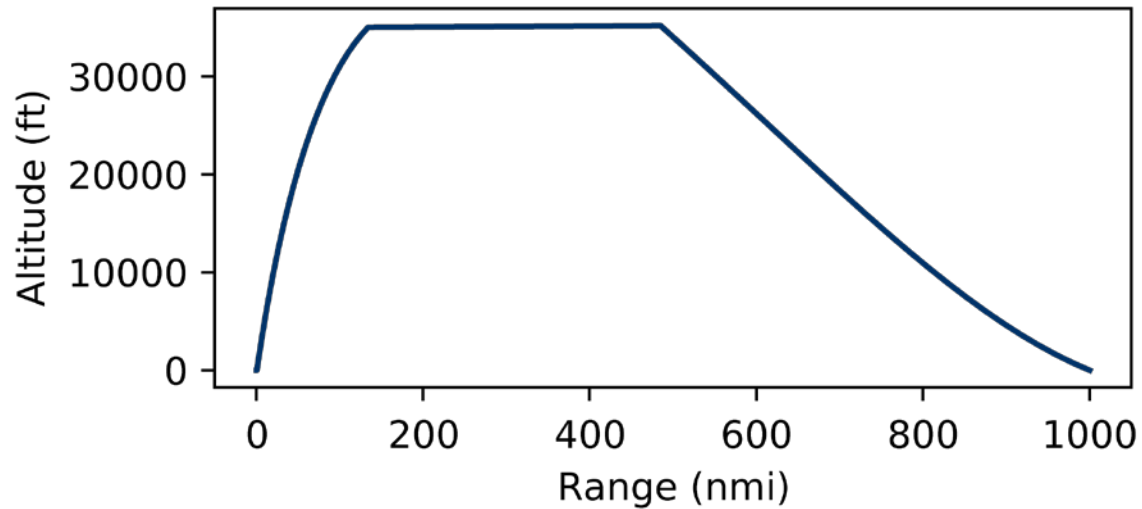
Adler, Brelje, and Martins, "Thermal Management System Optimization for a Parallel Hybrid Aircraft Considering Mission Fuel Burn", 2022.

And there is still more complexity

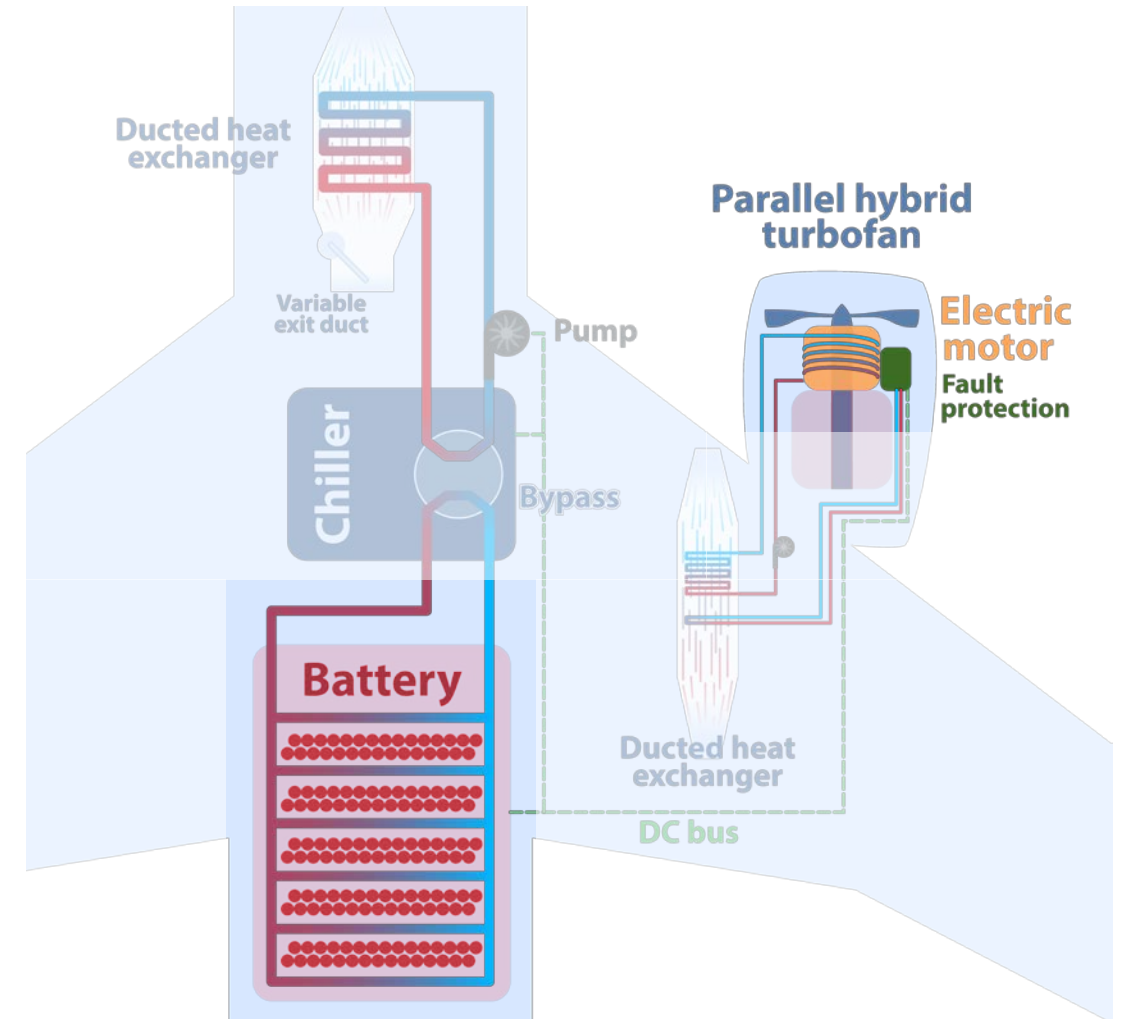
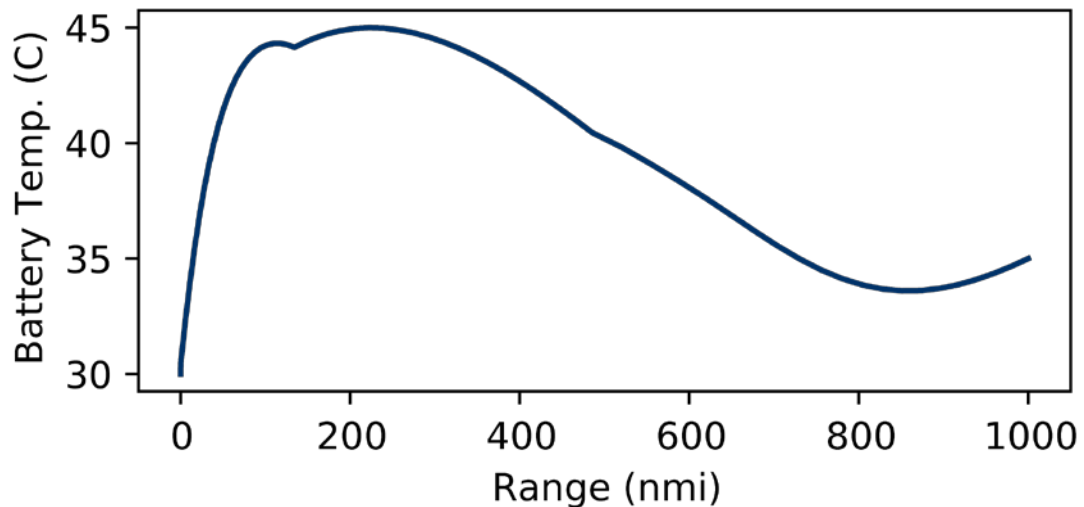
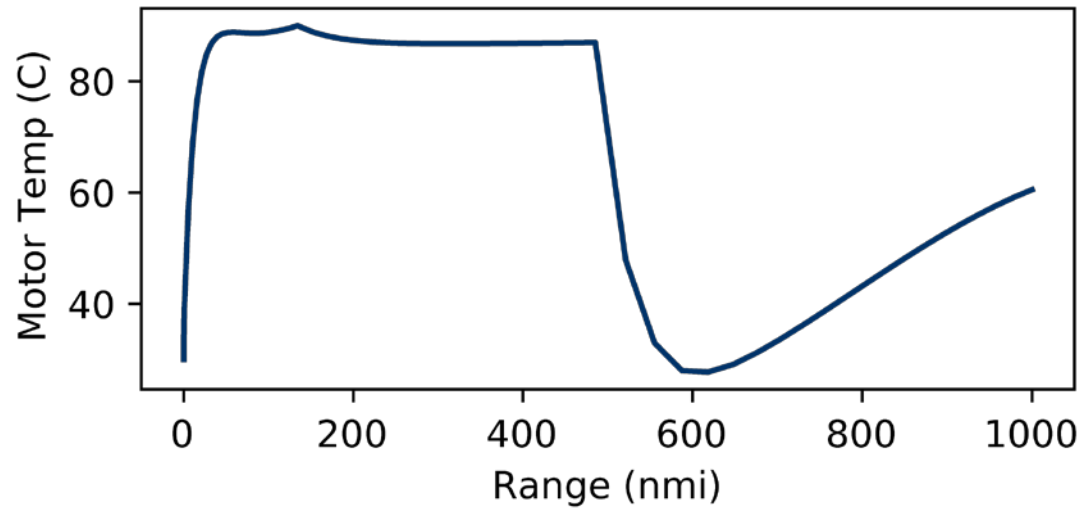
- All in a mission analysis
- Battery and motor can accumulate heat (not assuming steady state)
- Chiller bypass, variable exit duct, and engine hybrid fraction can be controlled during the mission



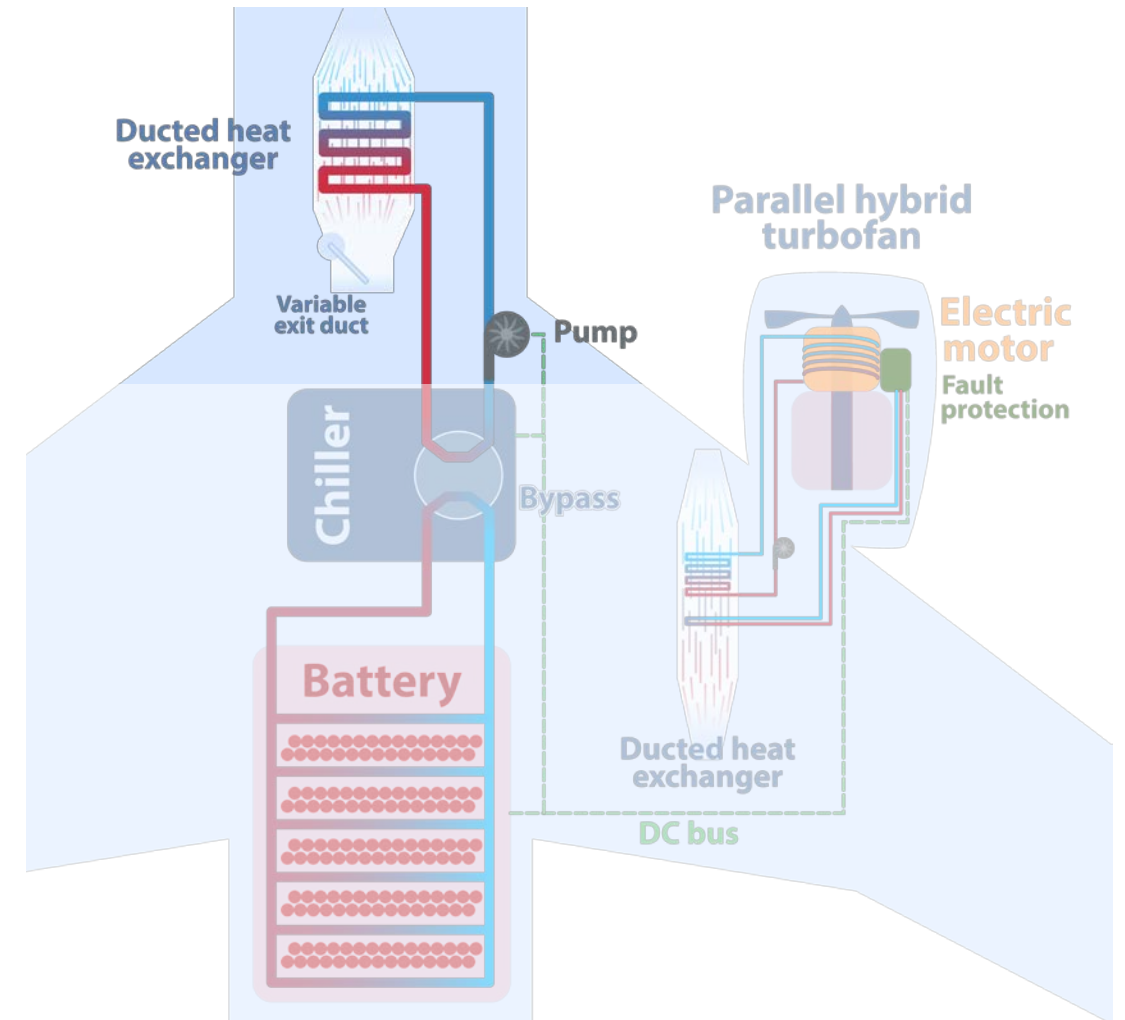
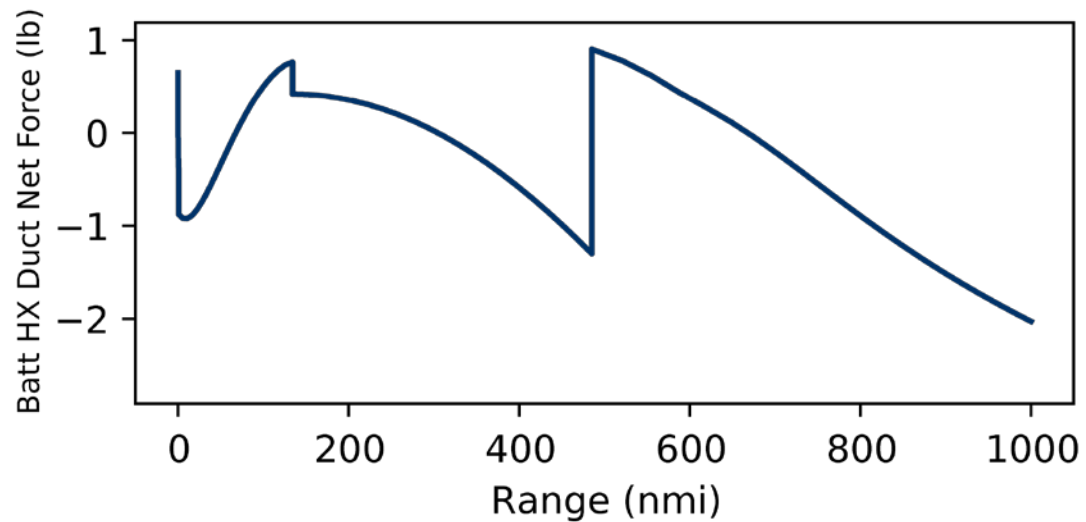
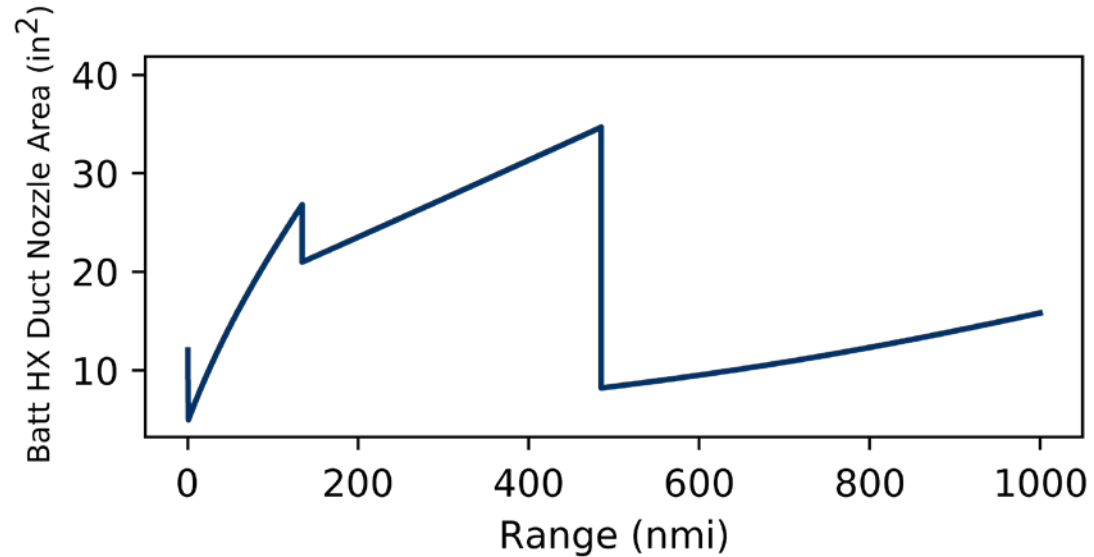
After running a mission, we can...



...analyze component temperatures



...and optimize duct area in time







Code flexibility

Mission analysis

Lessons learned



Code flexibility

Mission analysis

Lessons learned

Flexible



Easy to use

- Use on a wide range of problems
- Steeper learning curve
- Longer case setup time

- Well-defined interfaces
- Easier to learn
- Simpler setup



Aircraft model only has a few requirements

```
class Aircraft(om.Group):
    def setup(self):
        self.add_subsystem("aero", AerodynamicModel()) # compute drag using lift coefficient
        self.add_subsystem("prop", PropulsionModel()) # compute thrust using throttle

        intfuel = self.add_subsystem(
            "intfuel",
            Integrator(num_nodes=21, method="simpson", diff_units="s", time_setup="duration"),
        )
        intfuel.add_integrand("fuel_used", rate_name="fuel_flow", units="kg")
        self.connect("prop.fuel_flow", "intfuel.fuel_flow")

        self.add_subsystem("weight", WeightModel()) # compute weight
```

- Computes drag, thrust, and weight from C_L and throttle
- Atmospheric and flight conditions available as inputs

Must compute drag and thrust

```
class Aircraft(om.Group):
    def setup(self):
        self.add_subsystem("aero", AerodynamicModel()) # compute drag using lift coefficient
        self.add_subsystem("prop", PropulsionModel()) # compute thrust using throttle

        intfuel = self.add_subsystem(
            "intfuel",
            Integrator(num_nodes=21, method="simpson", diff_units="s", time_setup="duration"),
        )
        intfuel.add_integrand("fuel_used", rate_name="fuel_flow", units="kg")
        self.connect("prop.fuel_flow", "intfuel.fuel_flow")

        self.add_subsystem("weight", WeightModel()) # compute weight
```

- Computes drag, thrust, and weight from C_L and throttle
- Atmospheric and flight conditions available as inputs

Must compute weight

```
class Aircraft(om.Group):
    def setup(self):
        self.add_subsystem("aero", AerodynamicModel()) # compute drag using lift coefficient
        self.add_subsystem("prop", PropulsionModel()) # compute thrust using throttle

        intfuel = self.add_subsystem(
            "intfuel",
            Integrator(num_nodes=21, method="simpson", diff_units="s", time_setup="duration"),
        )
        intfuel.add_integrand("fuel_used", rate_name="fuel_flow", units="kg")
        self.connect("prop.fuel_flow", "intfuel.fuel_flow")

        self.add_subsystem("weight", WeightModel()) # compute weight
```

- Computes drag, thrust, and weight from C_L and throttle
- Atmospheric and flight conditions available as inputs

Can use OpenConcept's integrator for fuel

```
class Aircraft(om.Group):
    def setup(self):
        self.add_subsystem("aero", AerodynamicModel()) # compute drag using lift coefficient
        self.add_subsystem("prop", PropulsionModel()) # compute thrust using throttle

        intfuel = self.add_subsystem(
            "intfuel",
            Integrator(num_nodes=21, method="simpson", diff_units="s", time_setup="duration"),
        )
        intfuel.add_integrand("fuel_used", rate_name="fuel_flow", units="kg")
        self.connect("prop.fuel_flow", "intfuel.fuel_flow")

        self.add_subsystem("weight", WeightModel()) # compute weight
```

- Computes drag, thrust, and weight from C_L and throttle
- Atmospheric and flight conditions available as inputs

Mission analysis Group is the top level

```
class Mission(om.Group):
    def setup(self):
        # Define variables from airplane data file
        ac_vars = self.add_subsystem("ac_vars", DictIndepVarComp(acdata), promotes_outputs=["*"])
        ac_vars.add_output_from_dict("ac|aero|polar|e")
        ac_vars.add_output_from_dict("ac|aero|polar|CD0")
        ac_vars.add_output_from_dict("ac|geom|wing|S_ref")
        ac_vars.add_output_from_dict("ac|geom|wing|AR")

        # Run a full mission analysis including takeoff, climb, cruise, and descent
        self.add_subsystem(
            "mission_analysis",
            FullMissionAnalysis(num_nodes=21, aircraft_model=Aircraft),
            promotes_inputs=["*"],
            promotes_outputs=["*"],
        )
```


Pull some variables from a data file

```
class Mission(om.Group):
    def setup(self):
        # Define variables from airplane data file
        ac_vars = self.add_subsystem("ac_vars", DictIndepVarComp(acdata), promotes_outputs=["*"])
        ac_vars.add_output_from_dict("ac|aero|polar|e")
        ac_vars.add_output_from_dict("ac|aero|polar|CD0")
        ac_vars.add_output_from_dict("ac|geom|wing|S_ref")
        ac_vars.add_output_from_dict("ac|geom|wing|AR")

        # Run a full mission analysis including takeoff, climb, cruise, and descent
        self.add_subsystem(
            "mission_analysis",
            FullMissionAnalysis(num_nodes=21, aircraft_model=Aircraft),
            promotes_inputs=["*"],
            promotes_outputs=["*"],
        )
```

Add mission analysis group

```
class Mission(om.Group):
    def setup(self):
        # Define variables from airplane data file
        ac_vars = self.add_subsystem("ac_vars", DictIndepVarComp(acdata), promotes_outputs=["*"])
        ac_vars.add_output_from_dict("ac|aero|polar|e")
        ac_vars.add_output_from_dict("ac|aero|polar|CD0")
        ac_vars.add_output_from_dict("ac|geom|wing|S_ref")
        ac_vars.add_output_from_dict("ac|geom|wing|AR")

        # Run a full mission analysis including takeoff, climb, cruise, and descent
        self.add_subsystem(
            "mission_analysis",
            FullMissionAnalysis(num_nodes=21, aircraft_model=Aircraft),
            promotes_inputs=["*"],
            promotes_outputs=["*"],
        )
```



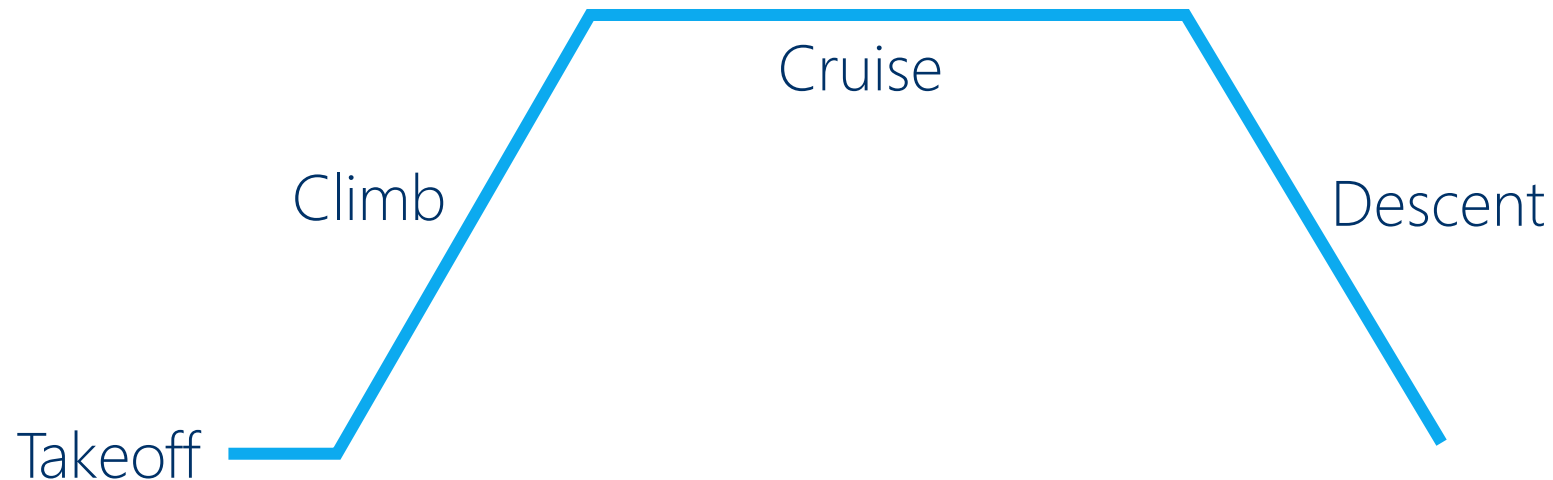
Code flexibility

Mission analysis

Lessons learned

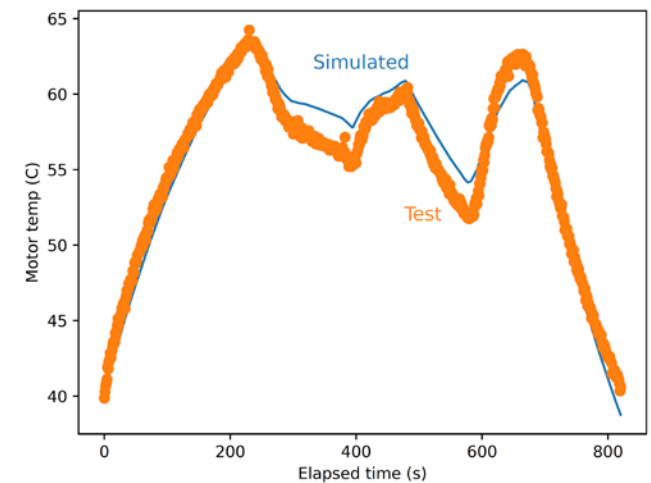
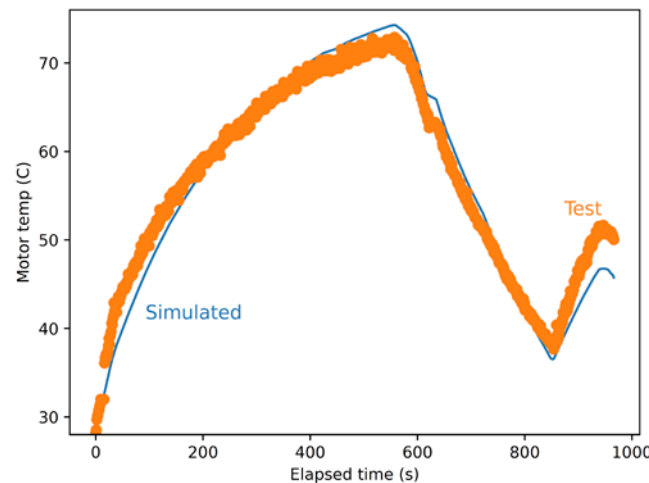
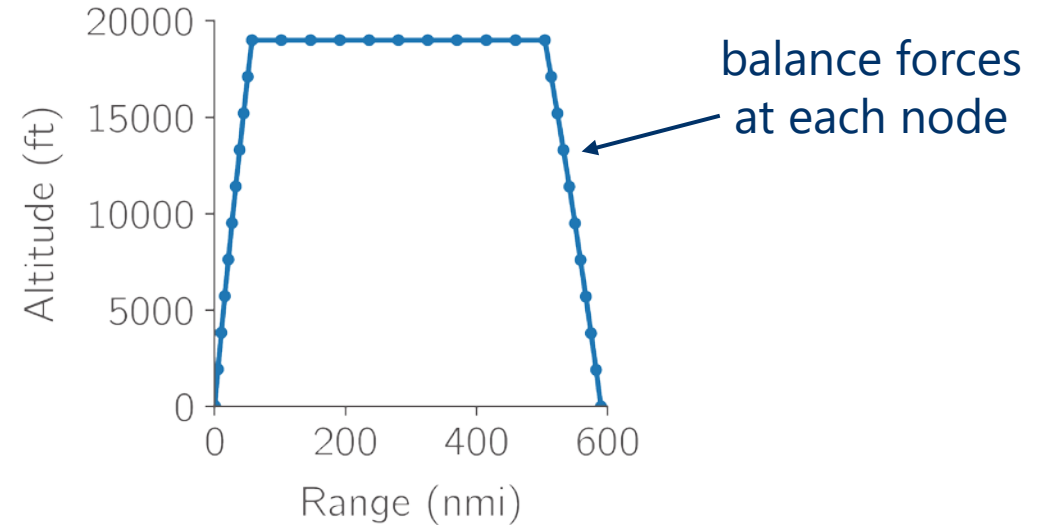
What is mission analysis?

- Simulate aircraft flying mission while satisfying physics
- Determine fuel burn
- Analyze component temperatures, hydrogen boil off, etc.

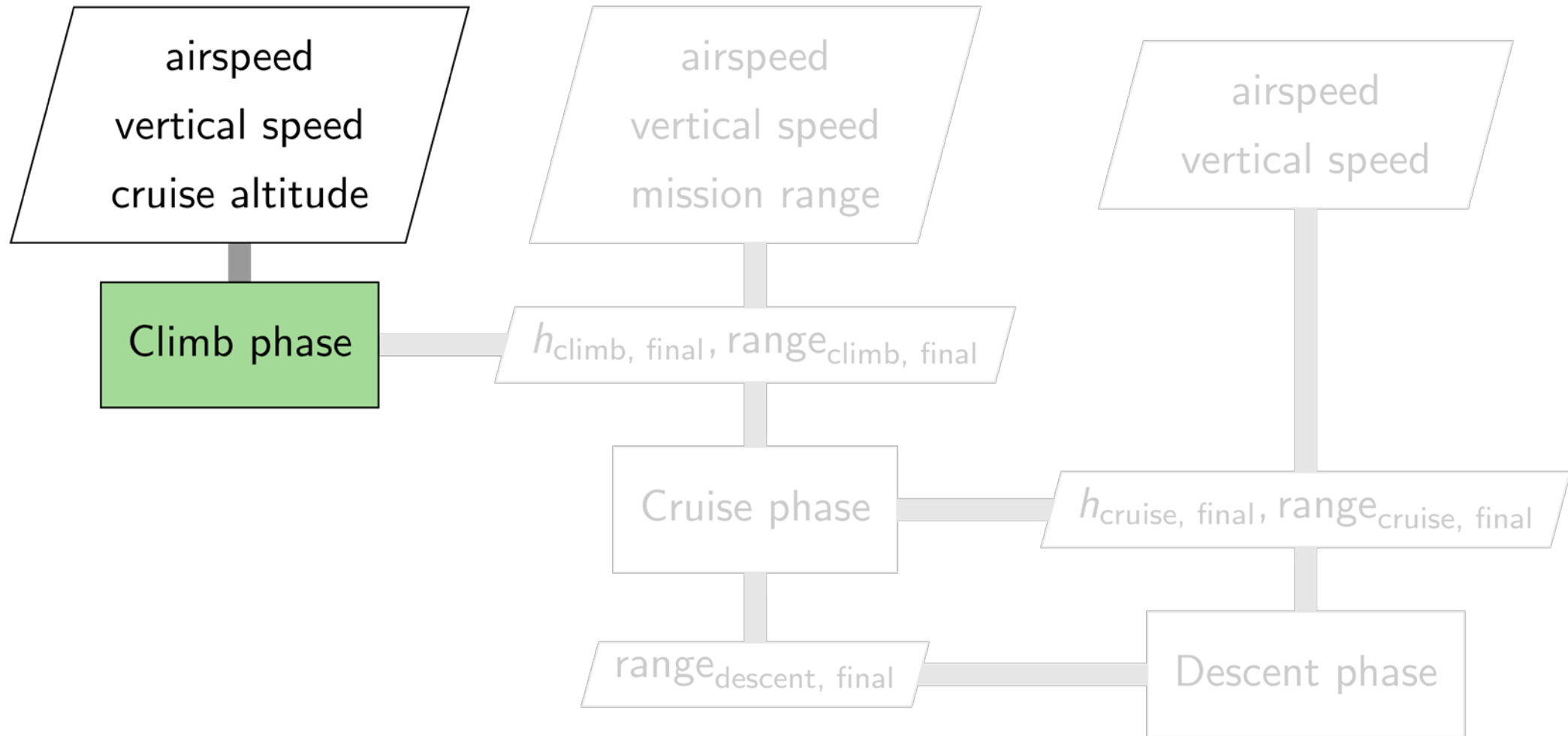


We assume steady flight at integration points

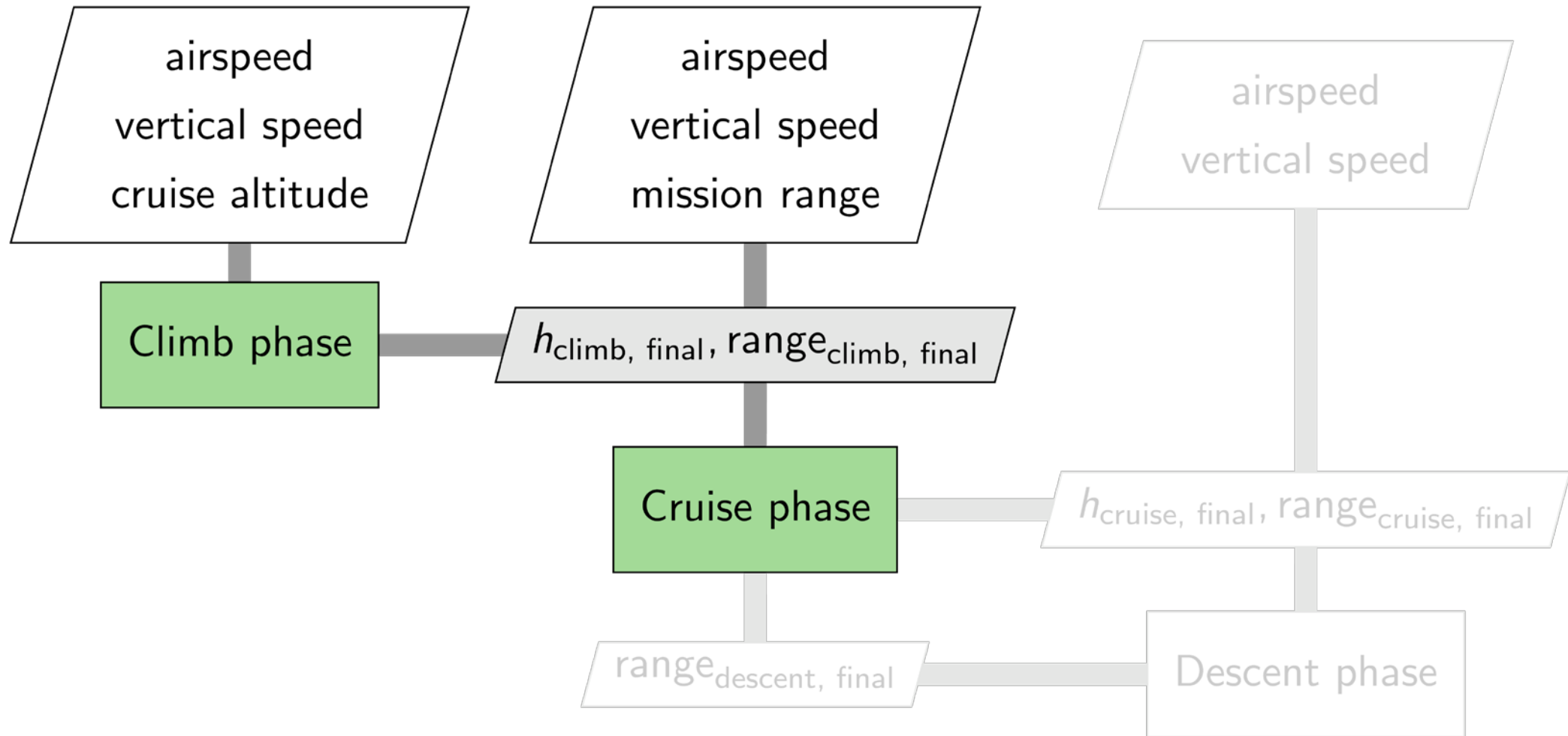
- This avoids working directly with the equations of motion
- Validated against real world data from Pipistrel



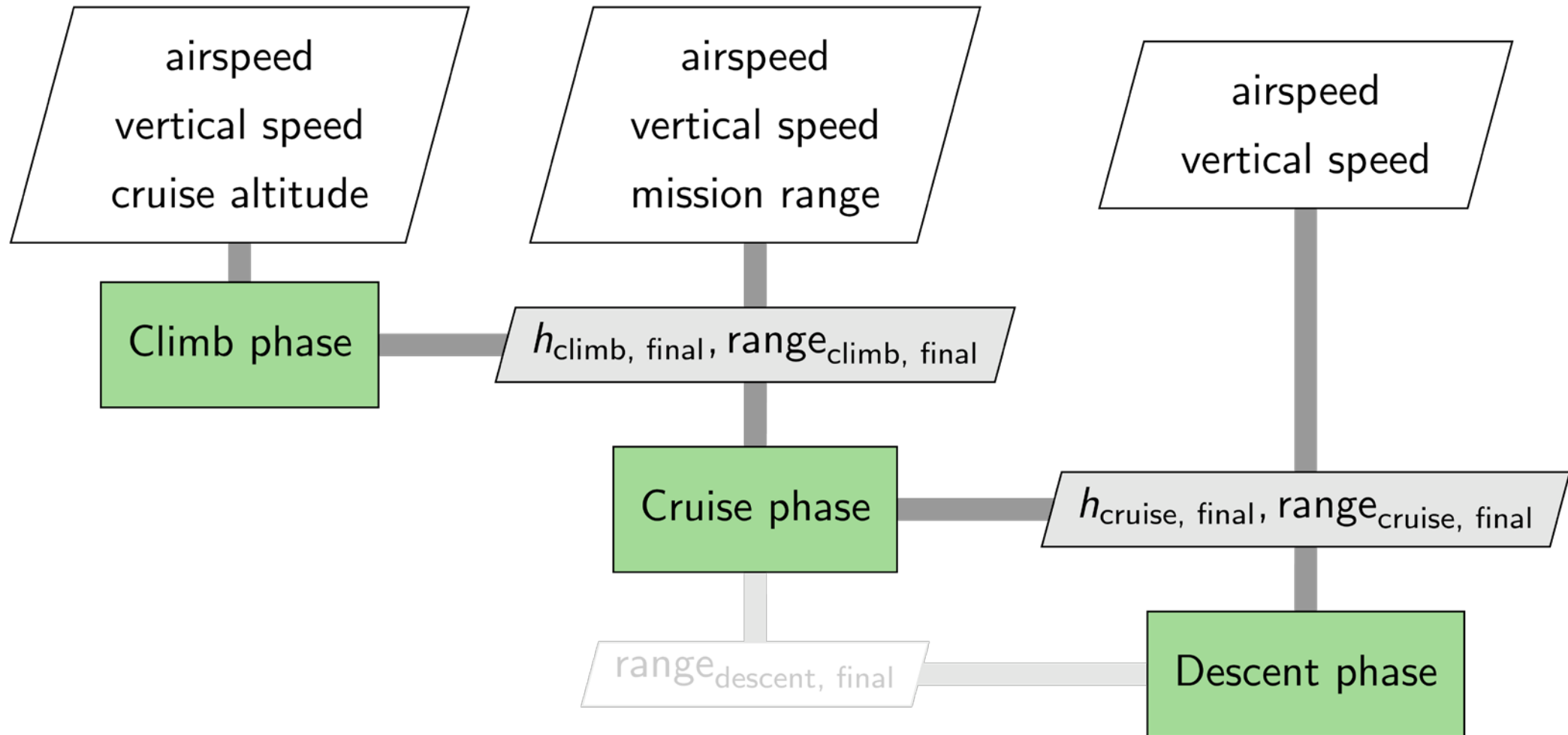
Basic mission setup



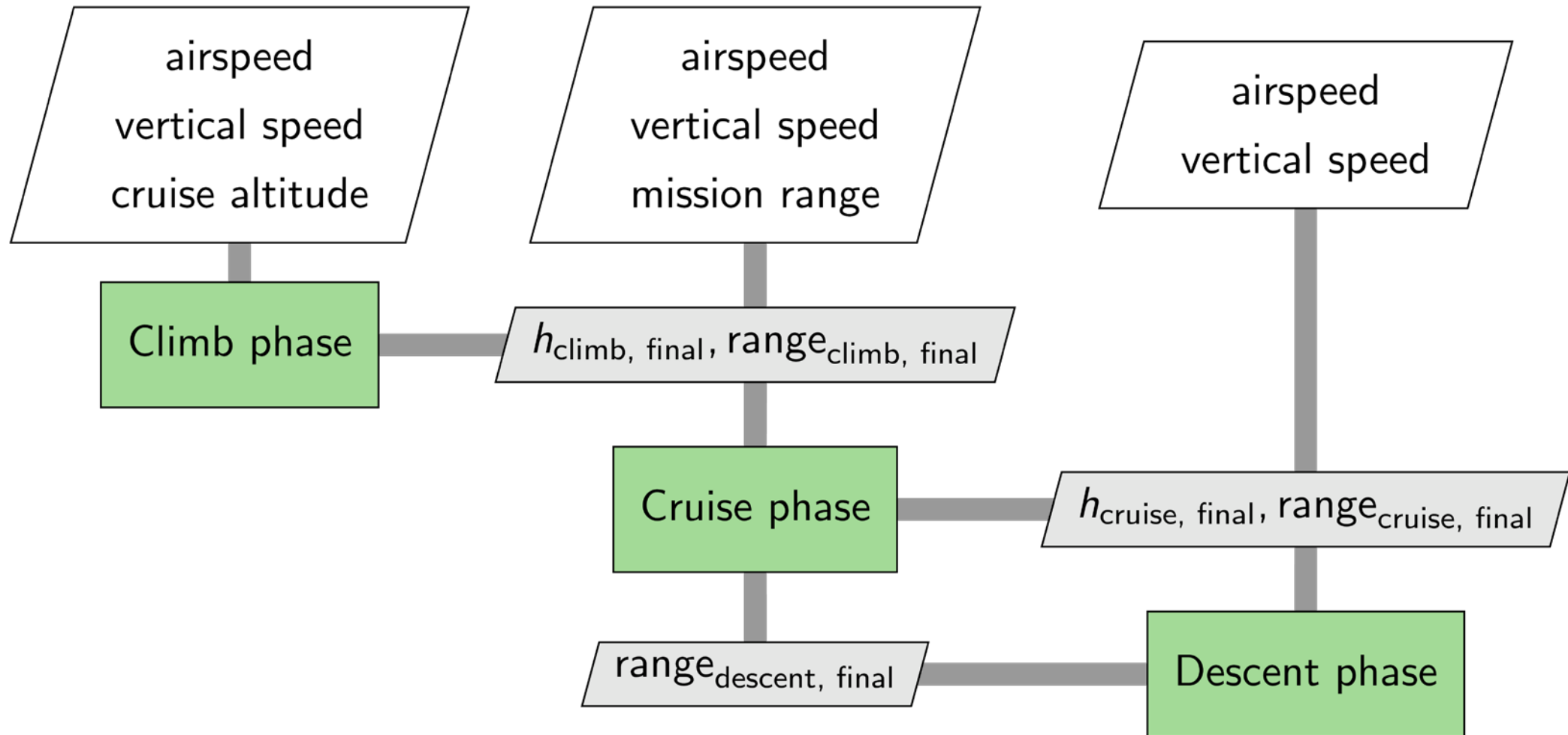
Basic mission setup



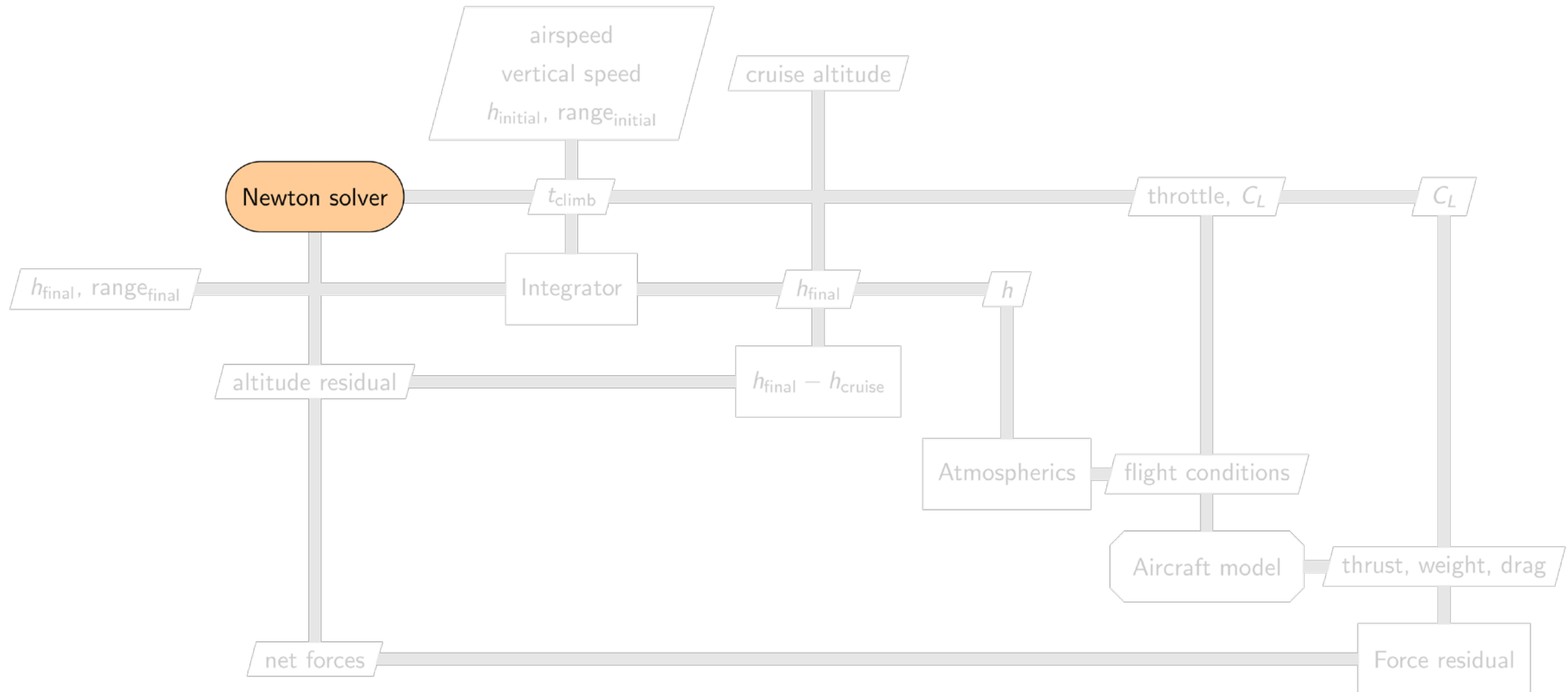
Basic mission setup



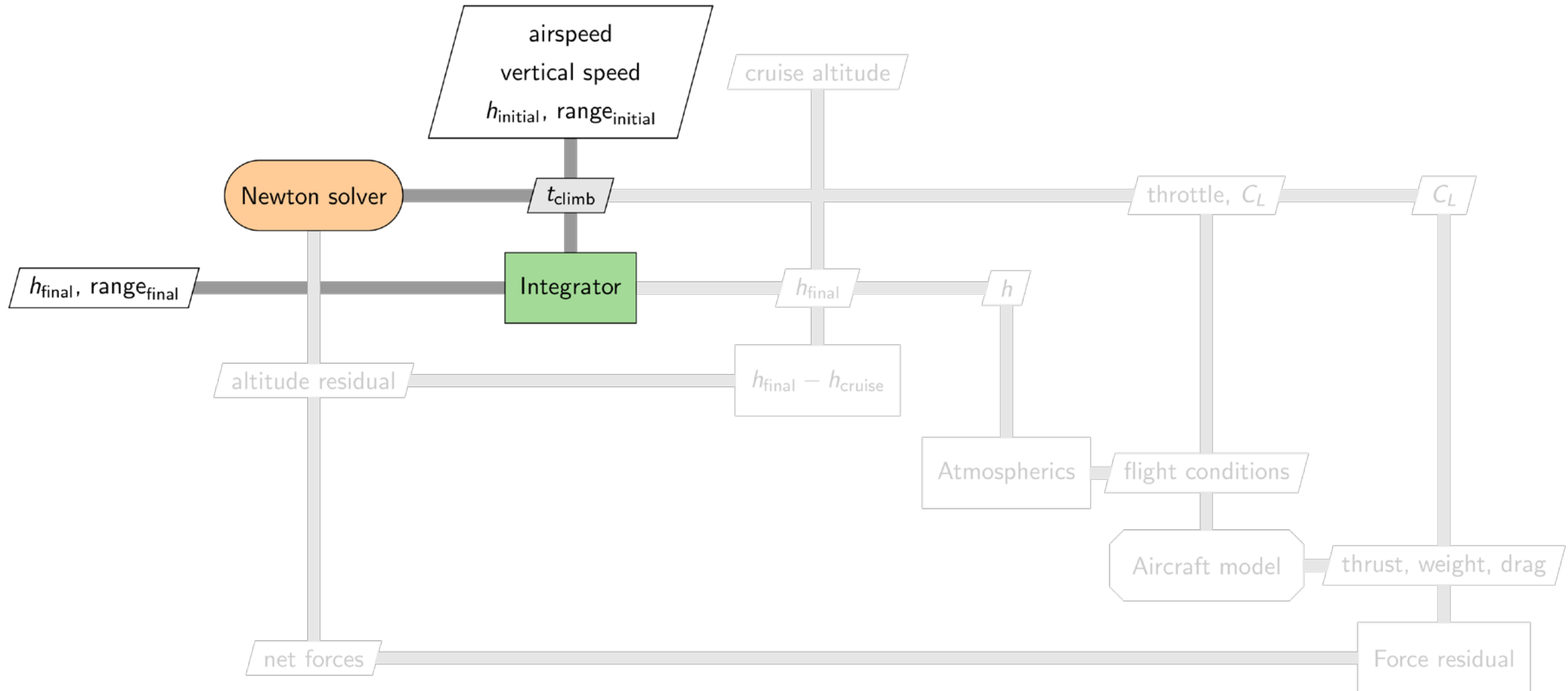
Basic mission setup



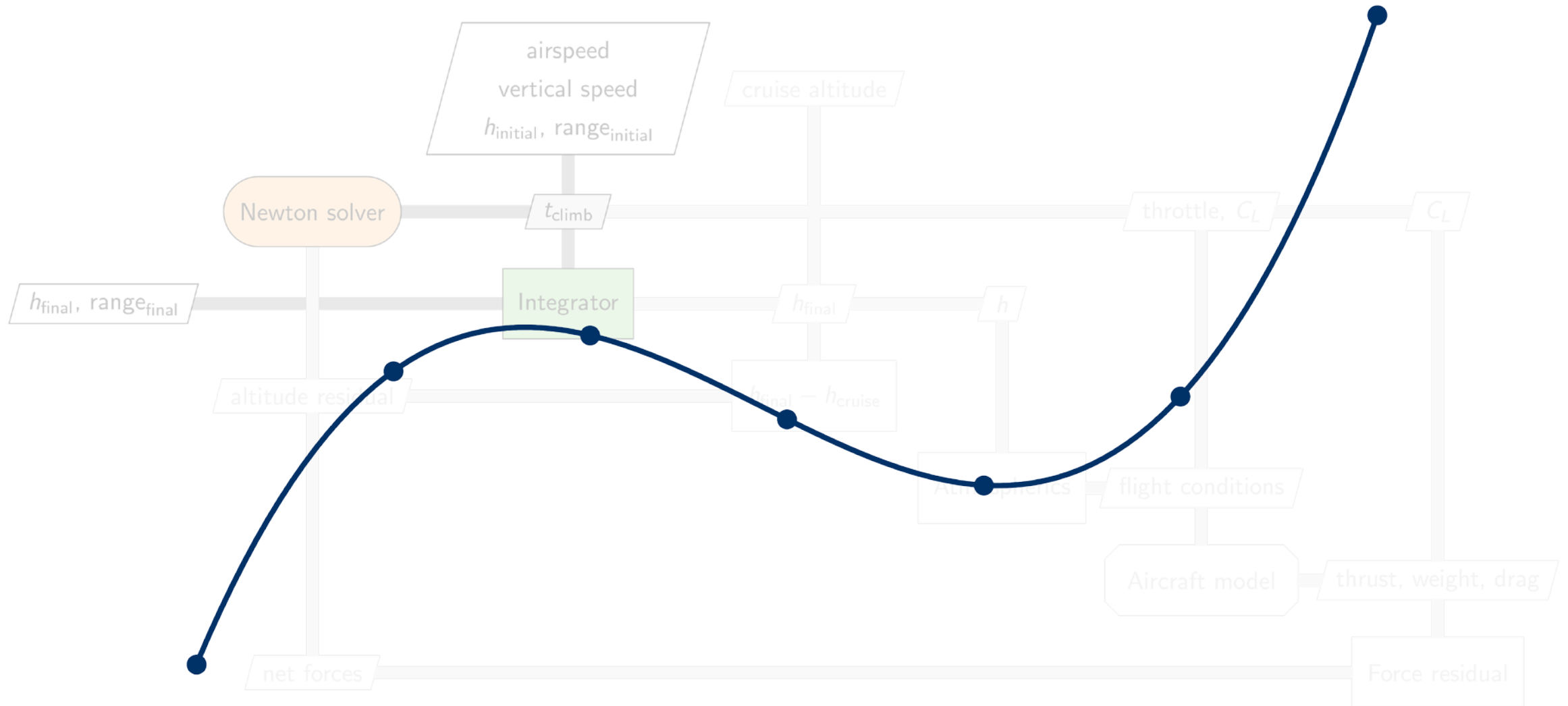
Solving an individual flight phase (climb)



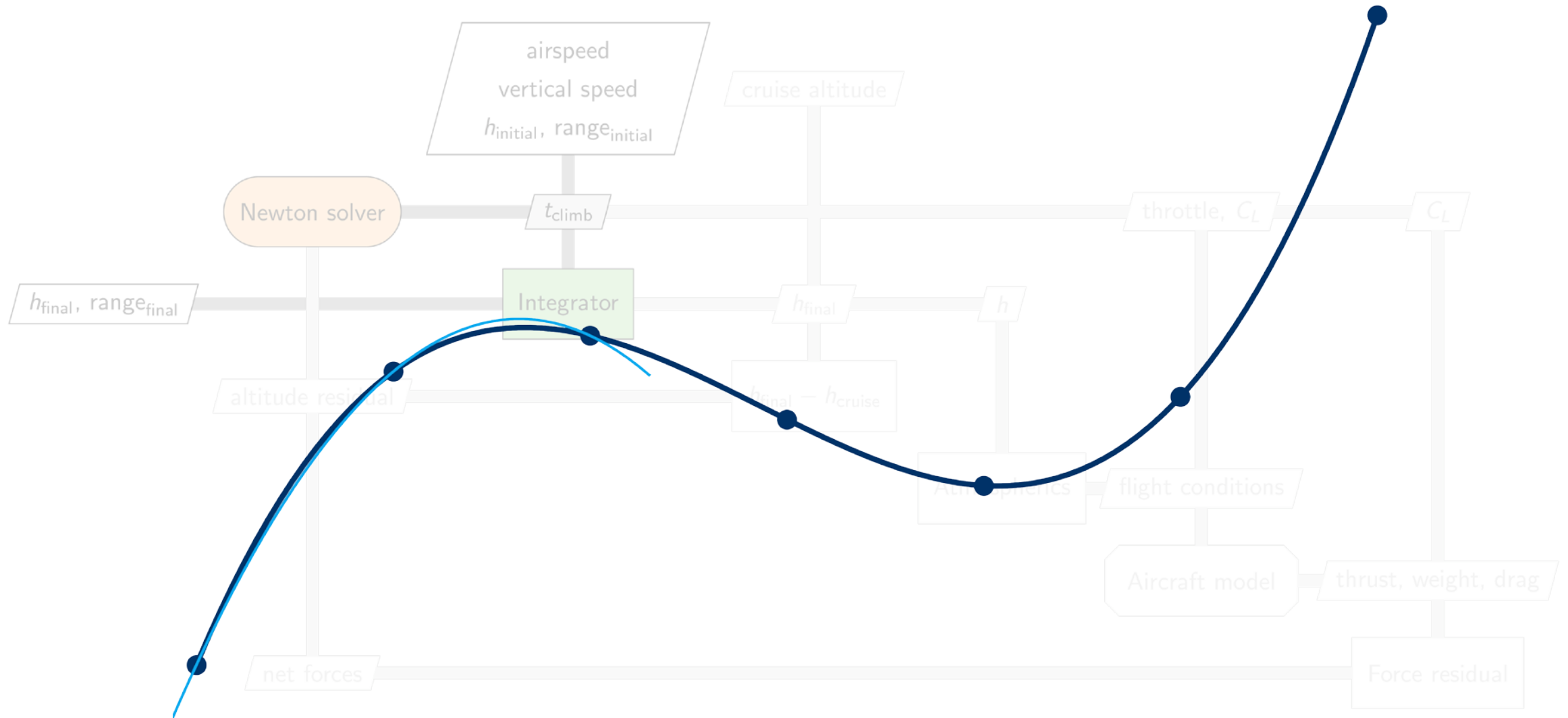
Integrator solves for altitude and range



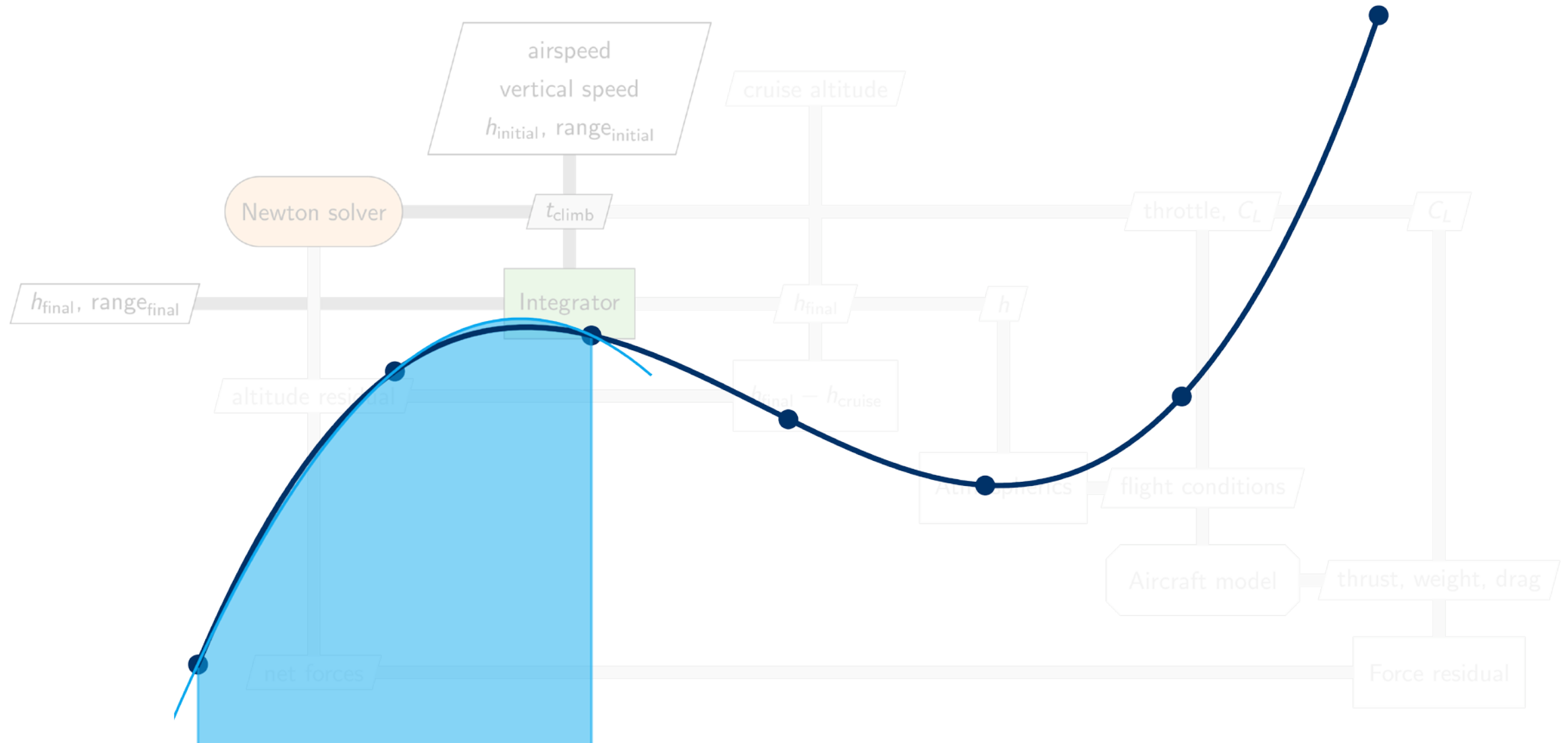
Integrator uses Simpson's rule



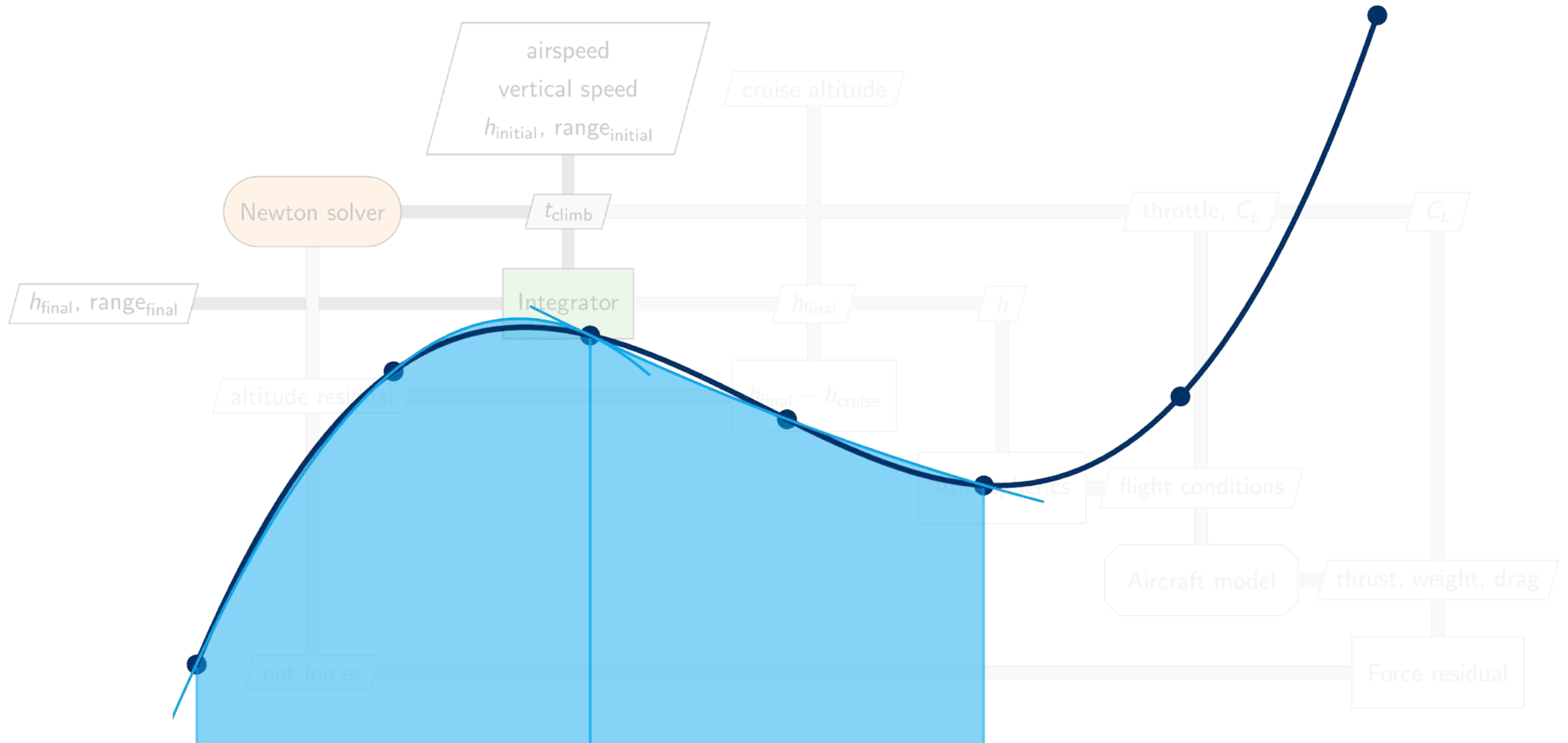
Integrator uses Simpson's rule



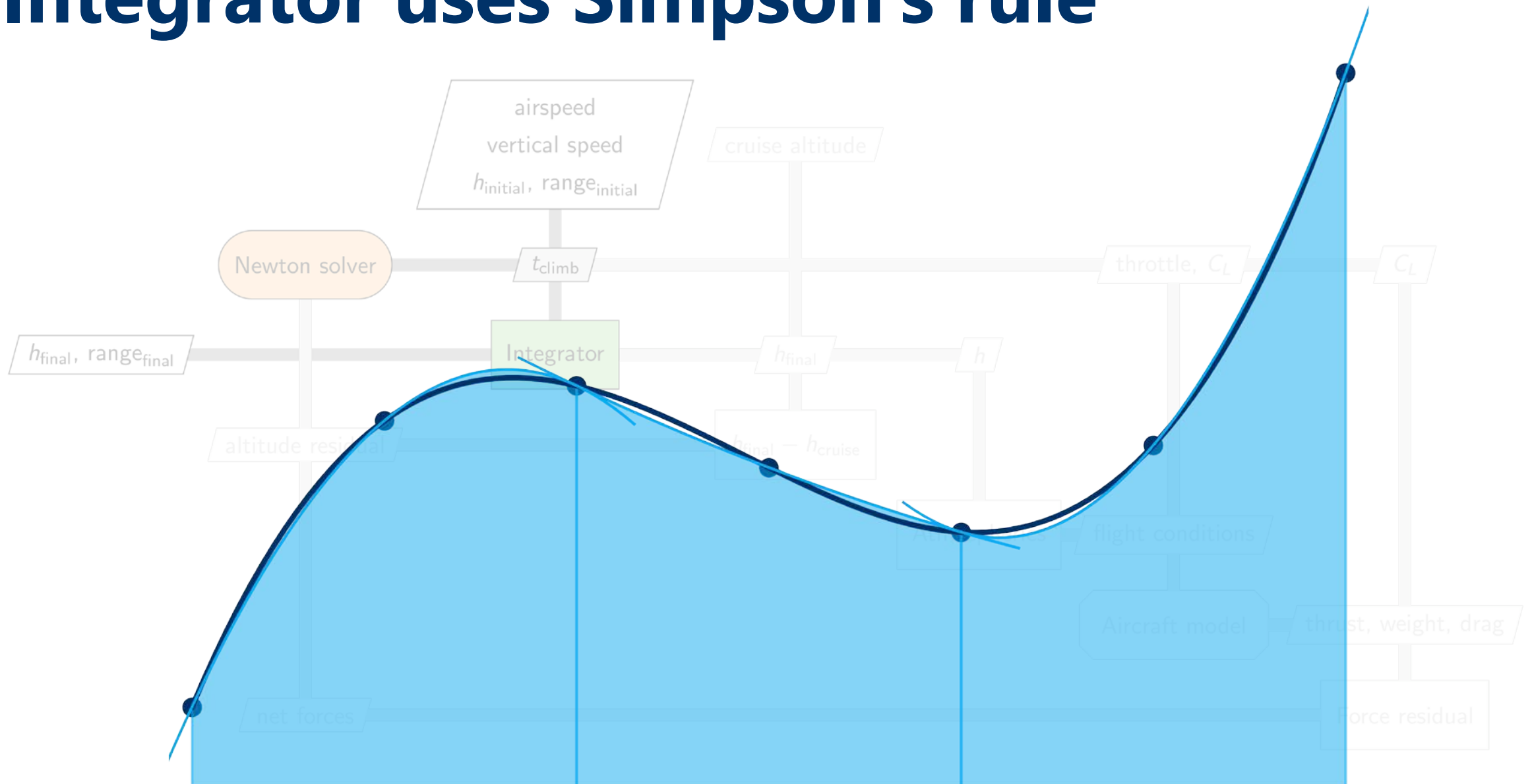
Integrator uses Simpson's rule



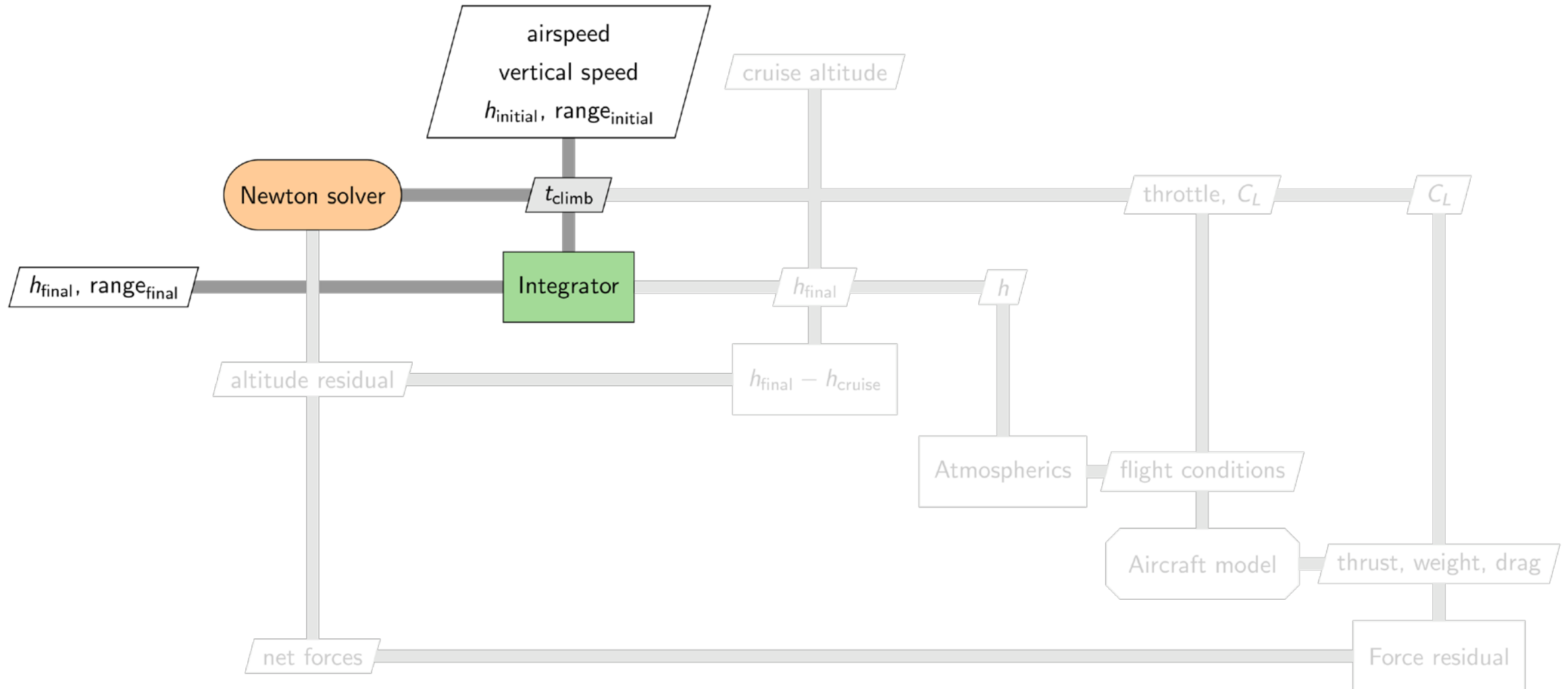
Integrator uses Simpson's rule



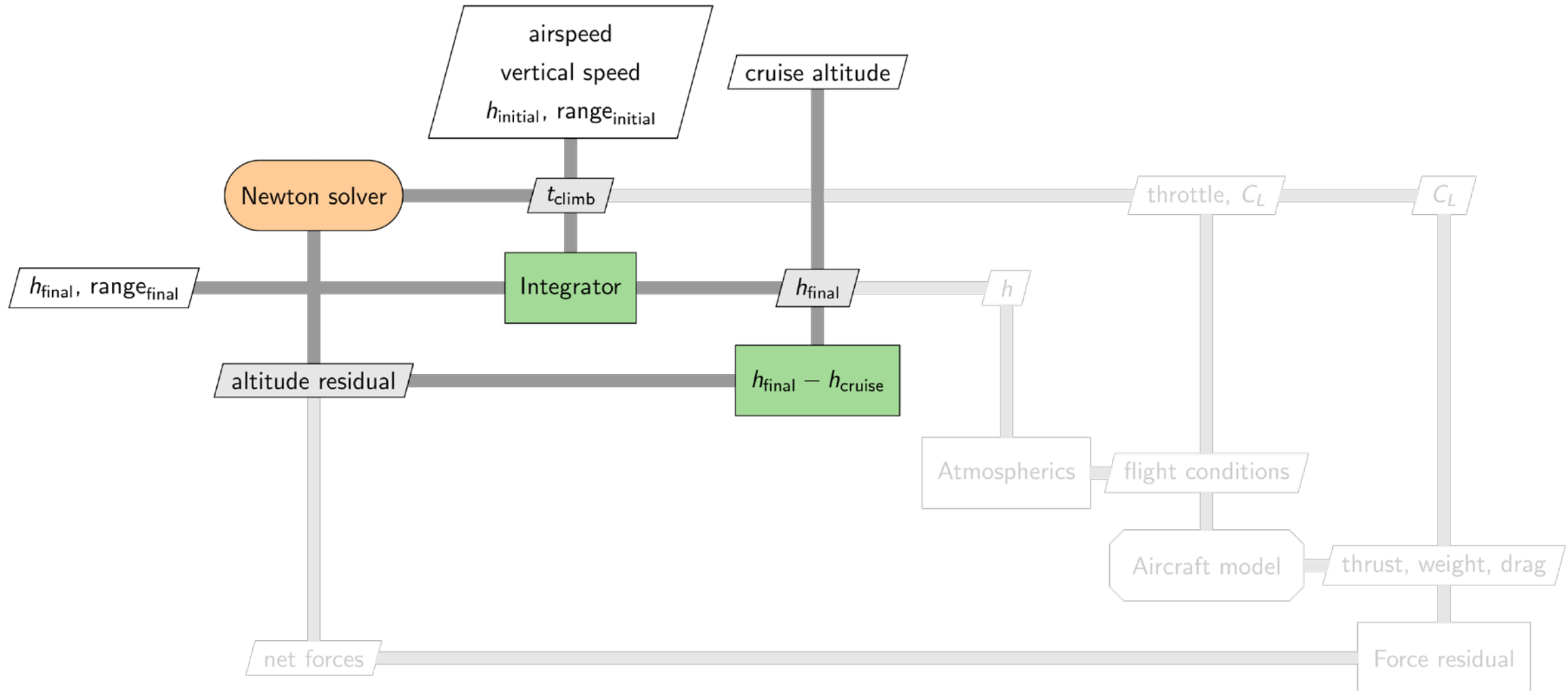
Integrator uses Simpson's rule



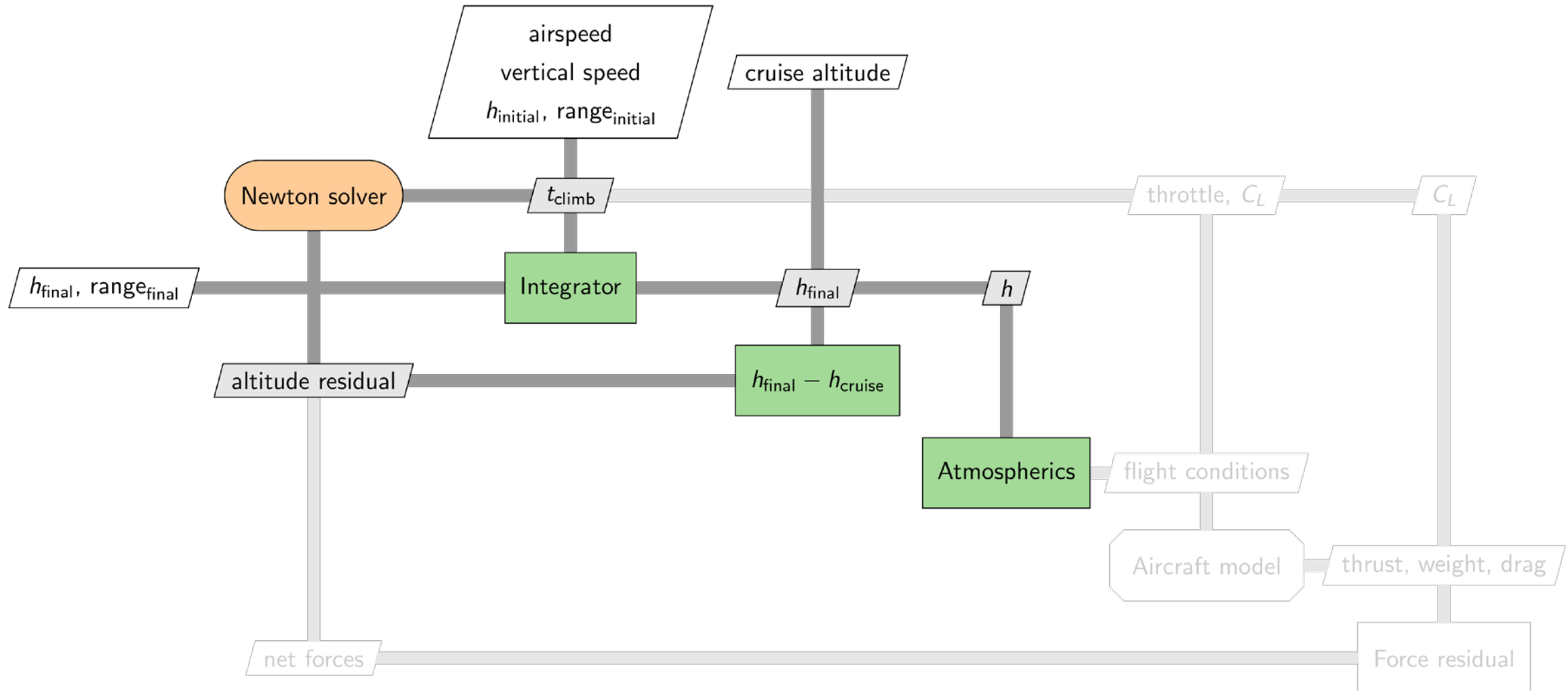
Integrator uses Simpson's rule



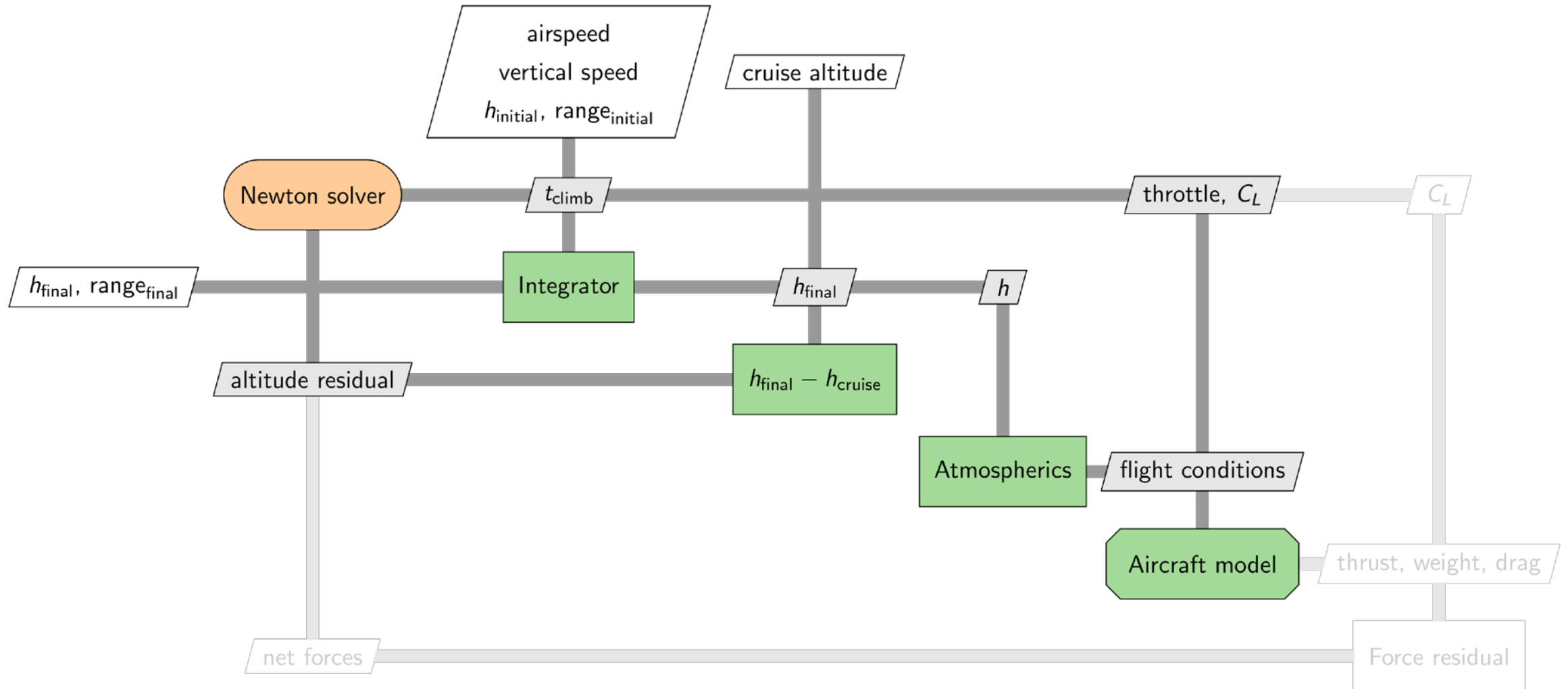
Set phase duration so it ends at cruise altitude



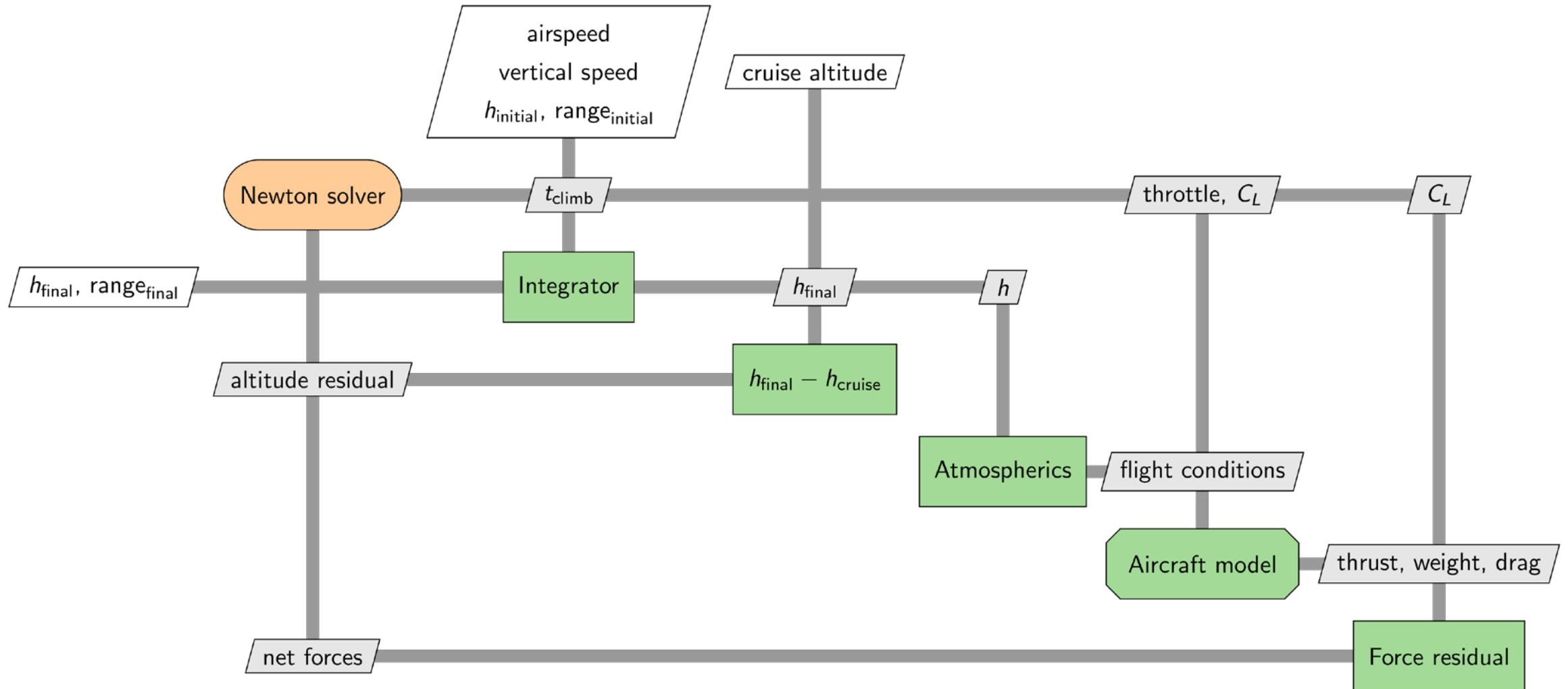
Atmospherics computes flight conditions



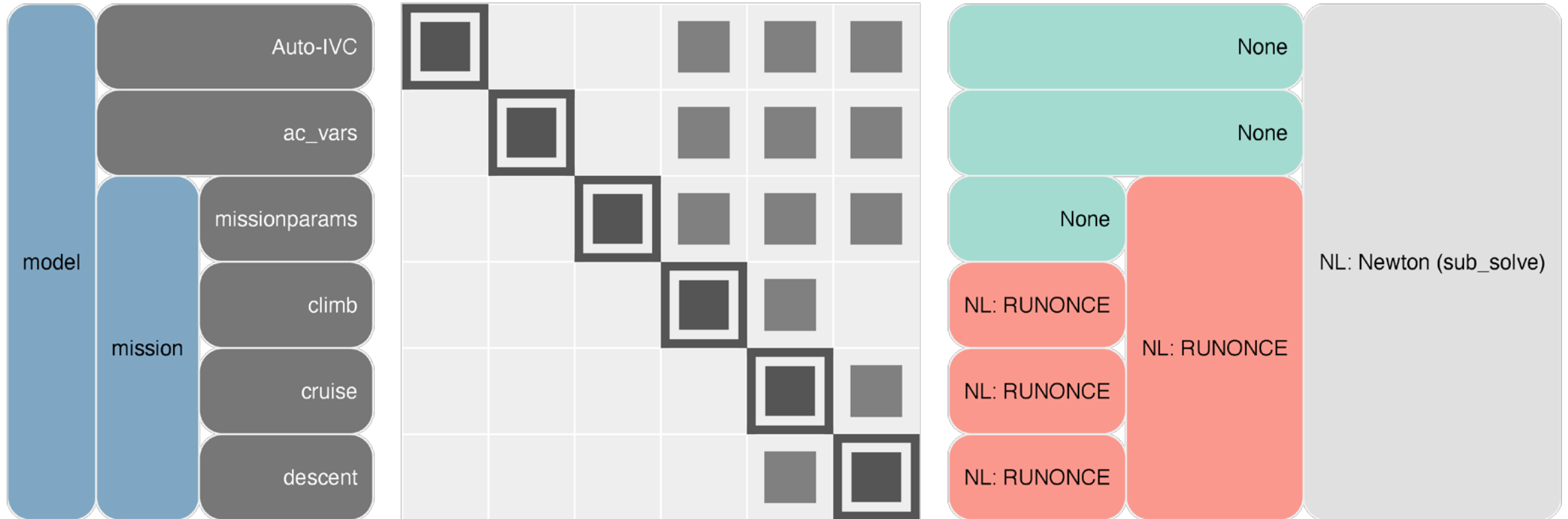
Aircraft model computes forces



Net forces driven to zero by Newton solver



We usually use a single Newton solver



The integrator is also used elsewhere

- Integrate fuel flow to compute fuel burn

```
intfuel = self.add_subsystem(  
    "intfuel",  
    Integrator(num_nodes=21, method="simpson", diff_units="s", time_setup="duration"),  
)  
intfuel.add_integrand("fuel_used", rate_name="fuel_flow", units="kg")
```

- Integrate heat flows to compute component temperature



A diagram showing a thermal component represented by a red rounded rectangle. Inside the rectangle is the differential equation $\frac{dT}{dt} = \frac{q_{in} - q_{out}}{mc}$. To the right of the rectangle, there are two horizontal arrows: the top one points left towards the rectangle and is labeled q_{in} , and the bottom one points right away from the rectangle and is labeled q_{out} .

- Brelje magic™ automatically finds states within aircraft model to integrate and links them across mission phases

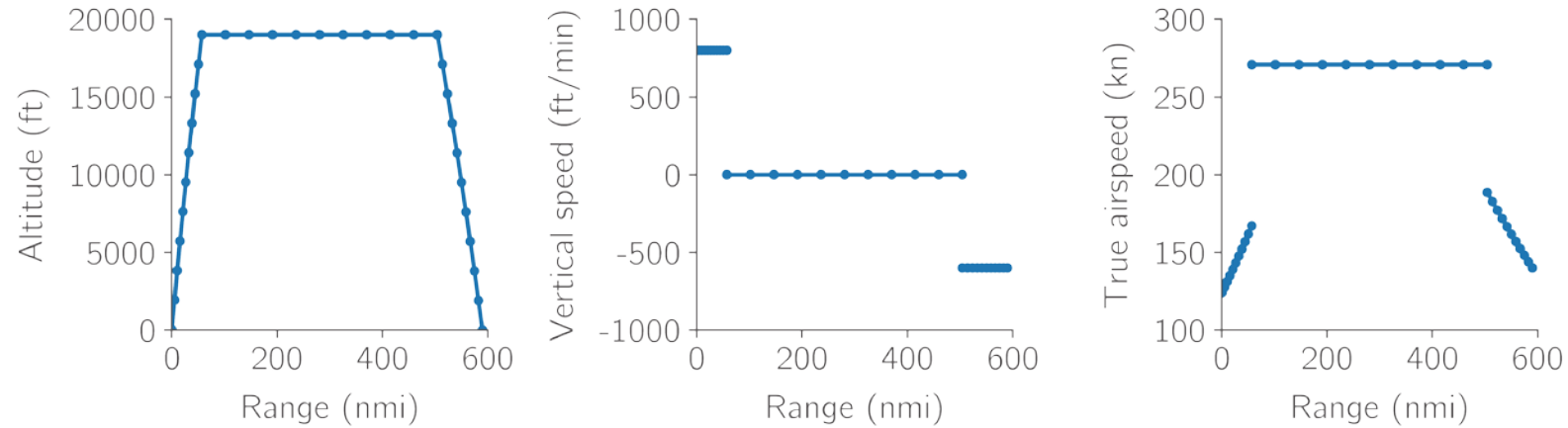


Code flexibility

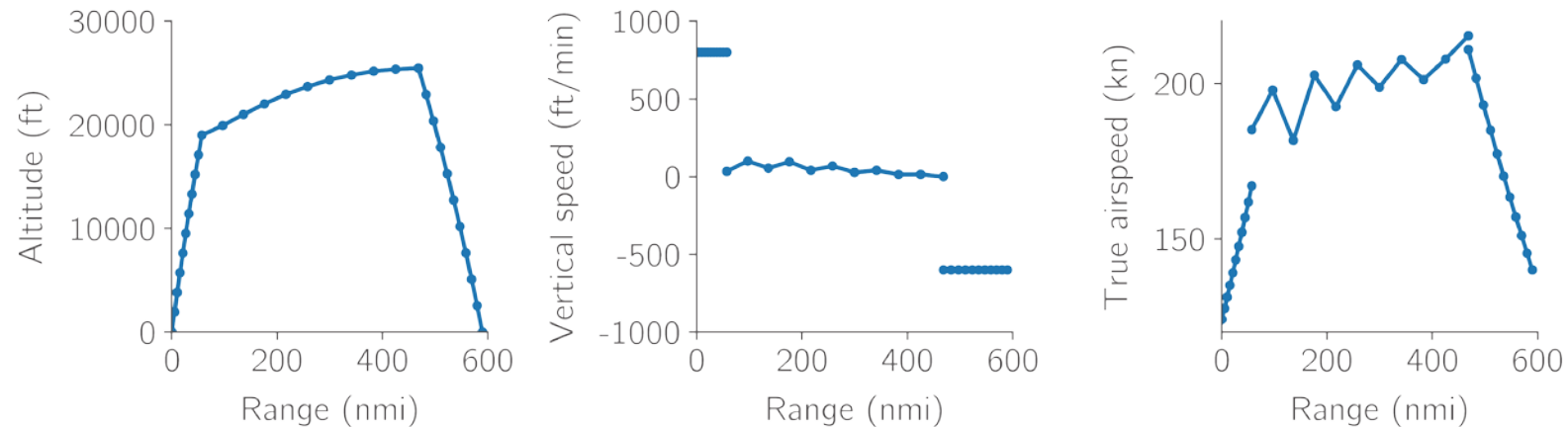
Mission analysis

Lessons learned

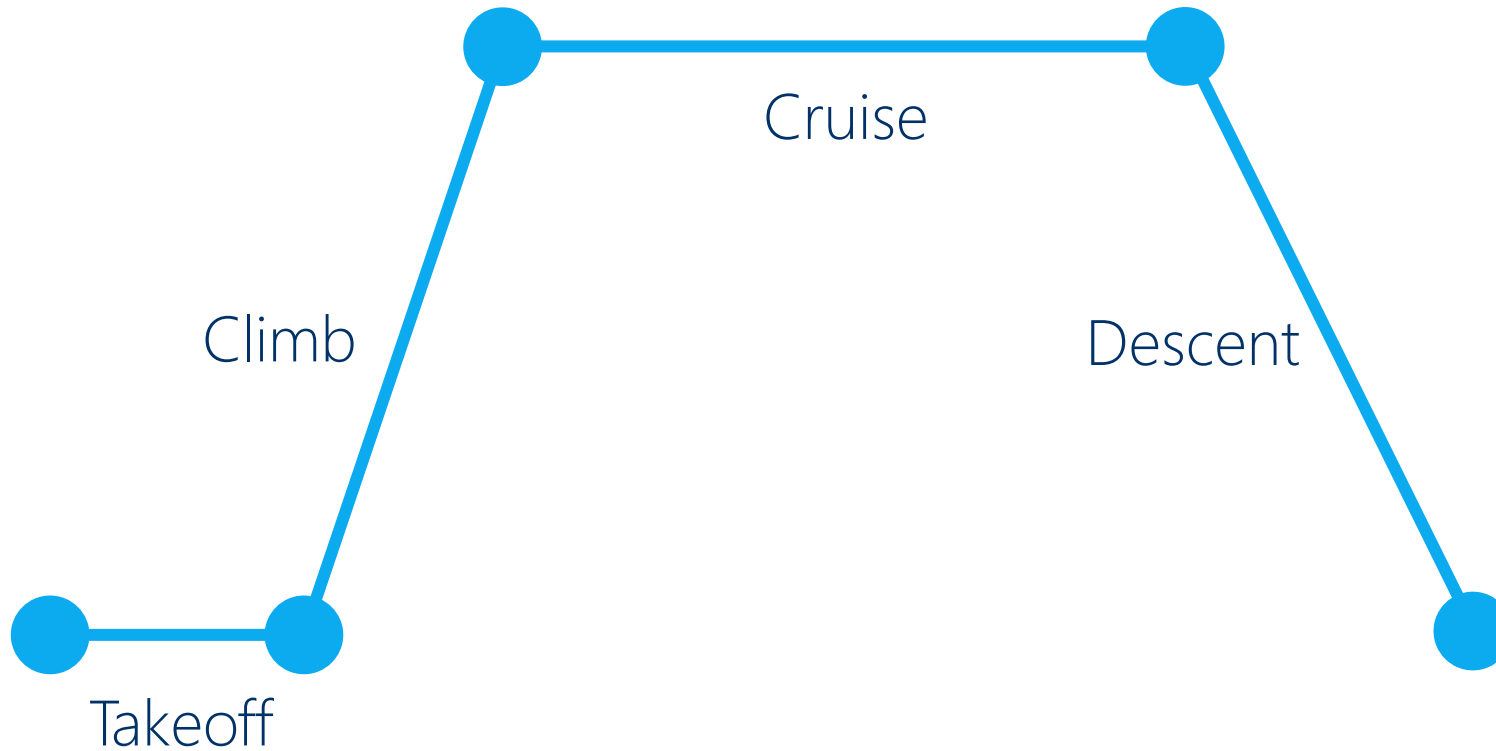
Not designed for trajectory optimization



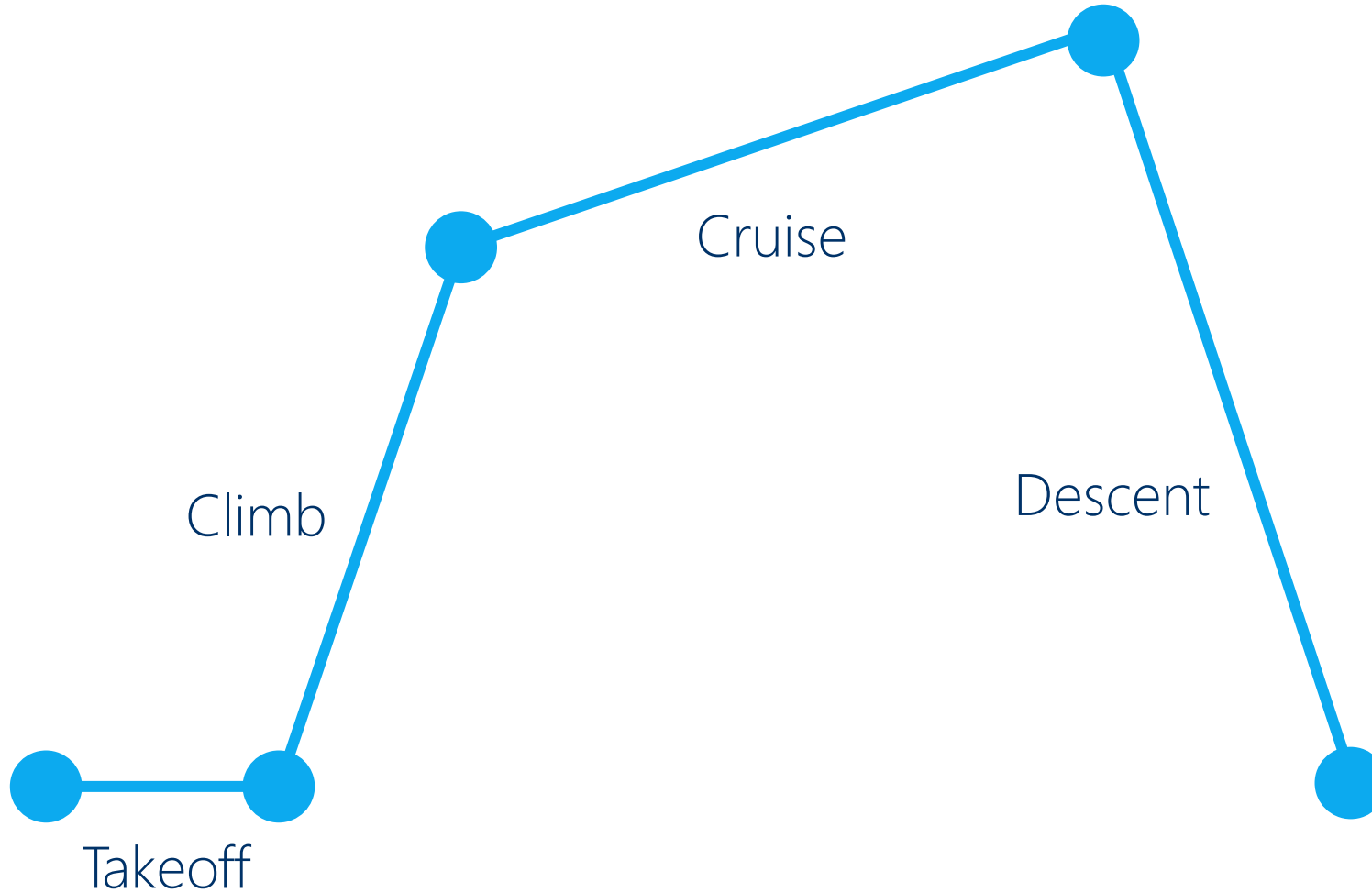
**Optimize cruise airspeed
and vertical speed**



We linearly interpolate within phases



We linearly interpolate within phases



Vectorization for efficiency

```
class SimpleMotor(om.ExplicitComponent):
    def initialize(self):
        self.options.declare("num_nodes", default=11)
        self.options.declare("efficiency", default=0.95)

    def setup(self):
        nn = self.options["num_nodes"]

        self.add_input("throttle", shape=(nn,))
        self.add_input("elec_power_rating", units="W")

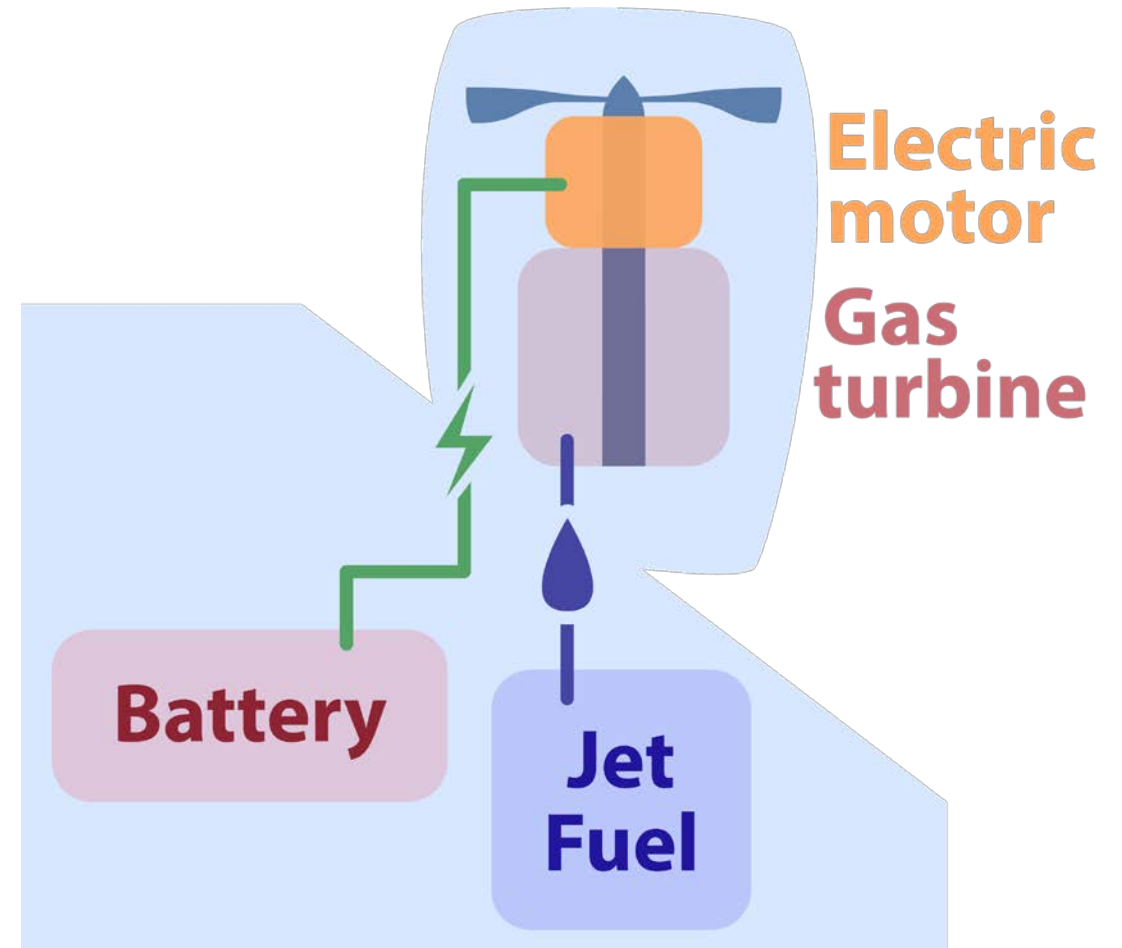
        self.add_output("shaft_power_out", units="W", shape=(nn,))
        self.add_output("elec_load", units="W", shape=(nn,))

        # declare sparse partials here

    def compute(self, inputs, outputs):
        outputs["shaft_power_out"] = inputs["throttle"] * inputs["elec_power_rating"] * self.options["efficiency"]
        outputs["elec_load"] = inputs["throttle"] * inputs["elec_power_rating"]
```

Propulsion modeling with surrogates

- Detailed turbomachinery and propulsion models use offline surrogates (often pyCycle)
- Avoids challenges with robustness and cost



What is best for mission analysis?

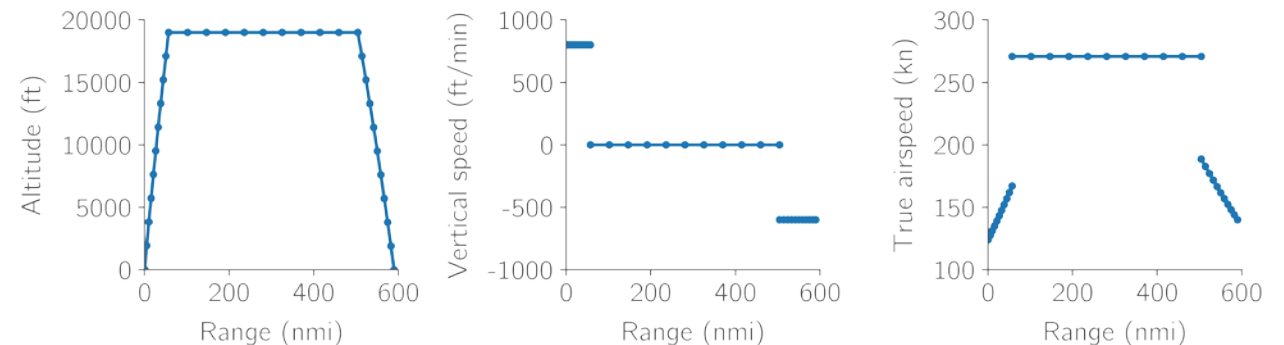
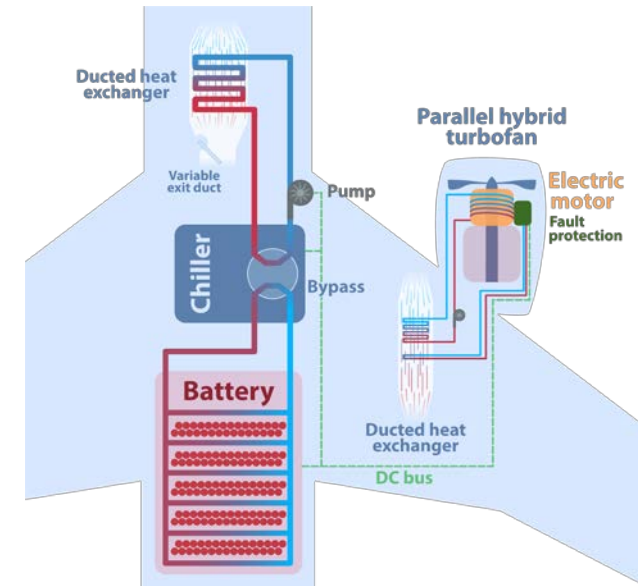
- OpenConcept's approach
 - Fast
 - Robust-ish
 - Physics valid once Newton solver converges (no optimization required)
- Trajectory optimization-style approach
 - Not as robust
 - More general representation of mission profile
 - Better for mission profile optimization

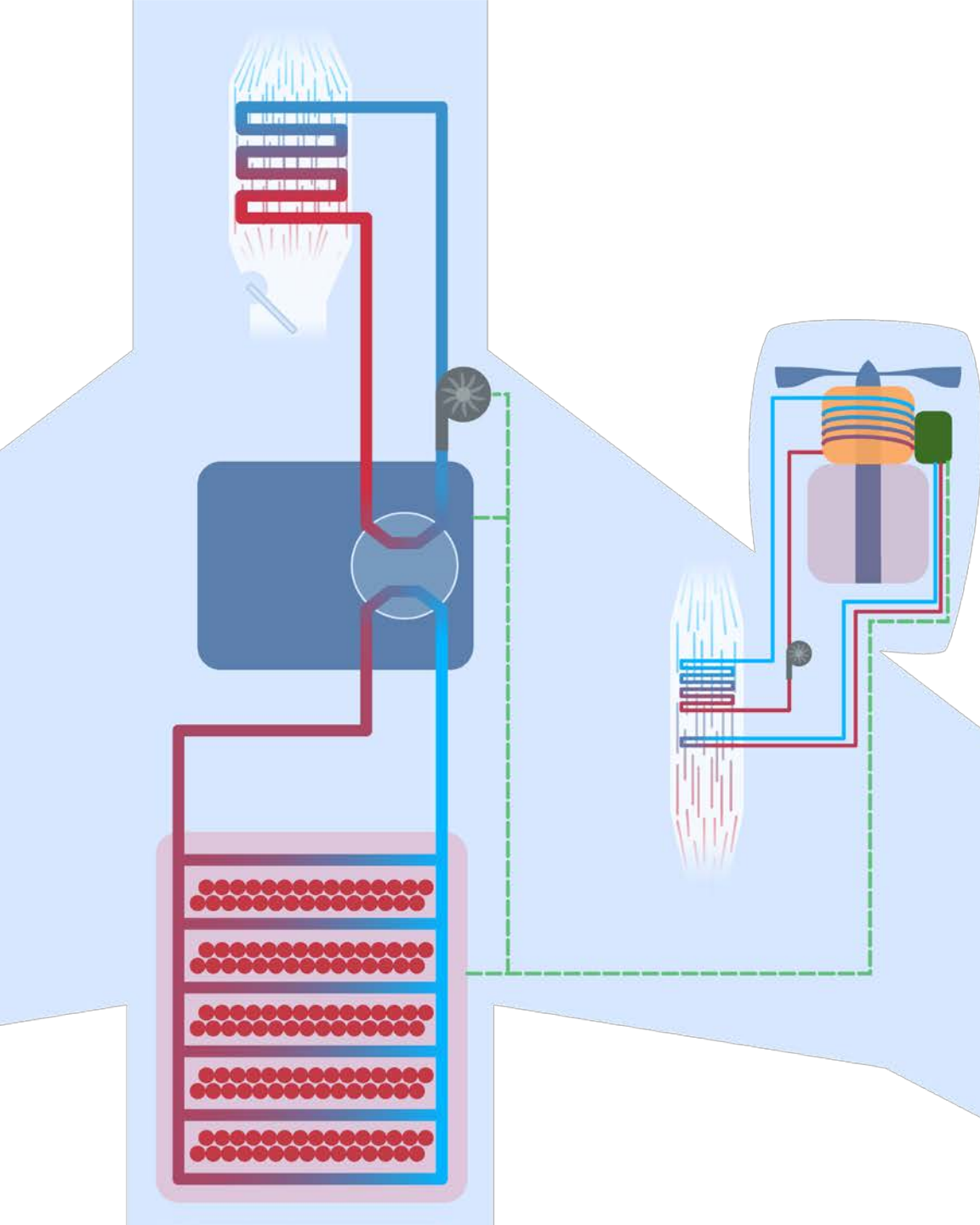
We developed a tool for efficient conceptual aircraft design with OpenMDAO

Rapidly optimize aircraft architectures

Steady flight mission phases with predefined profile

Hybrid mission analysis approach for the best of both?





github.com/mdolab/openconcept

