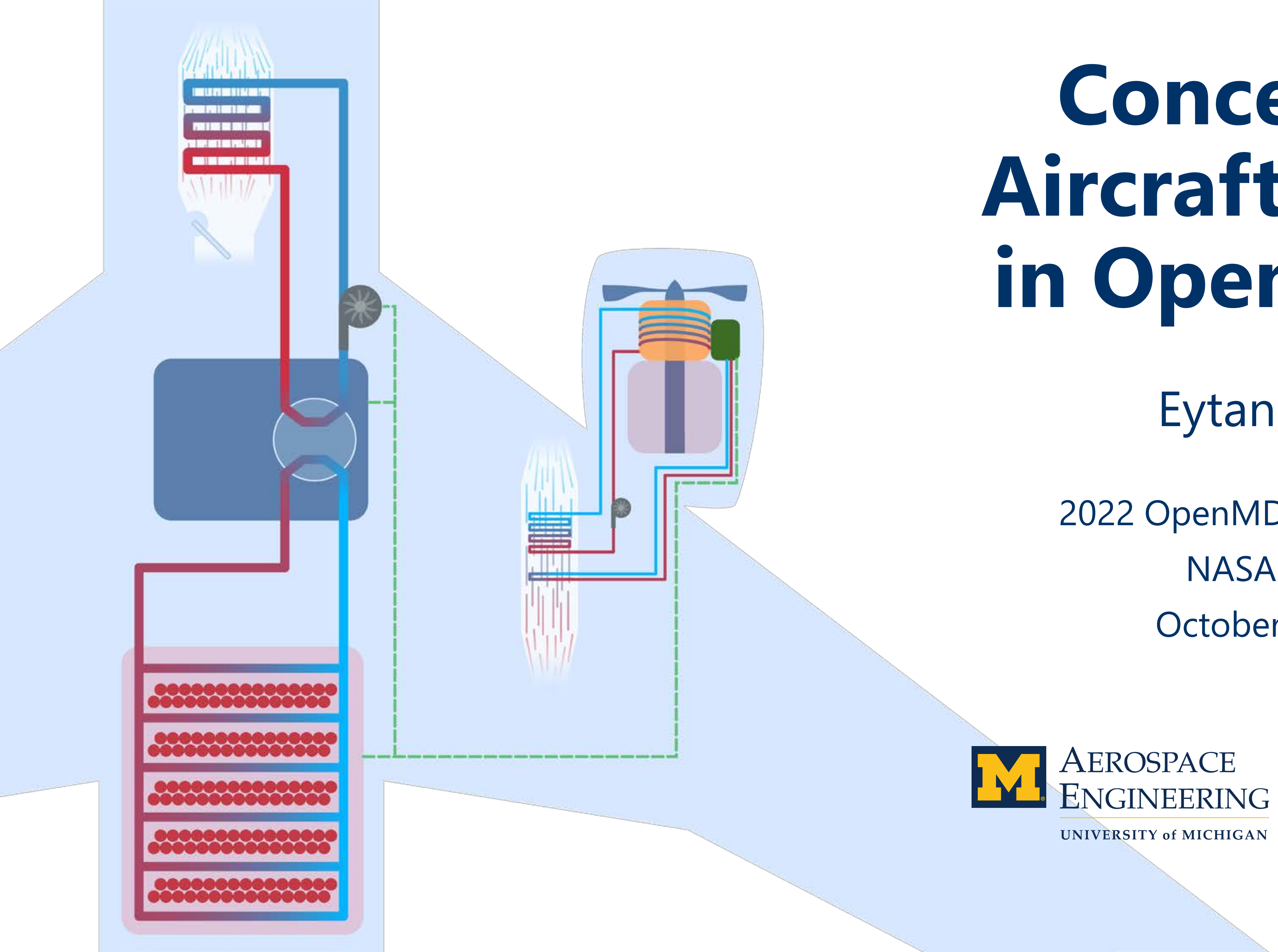


Conceptual Aircraft Design in OpenMDAO

Eytan Adler

2022 OpenMDAO Workshop

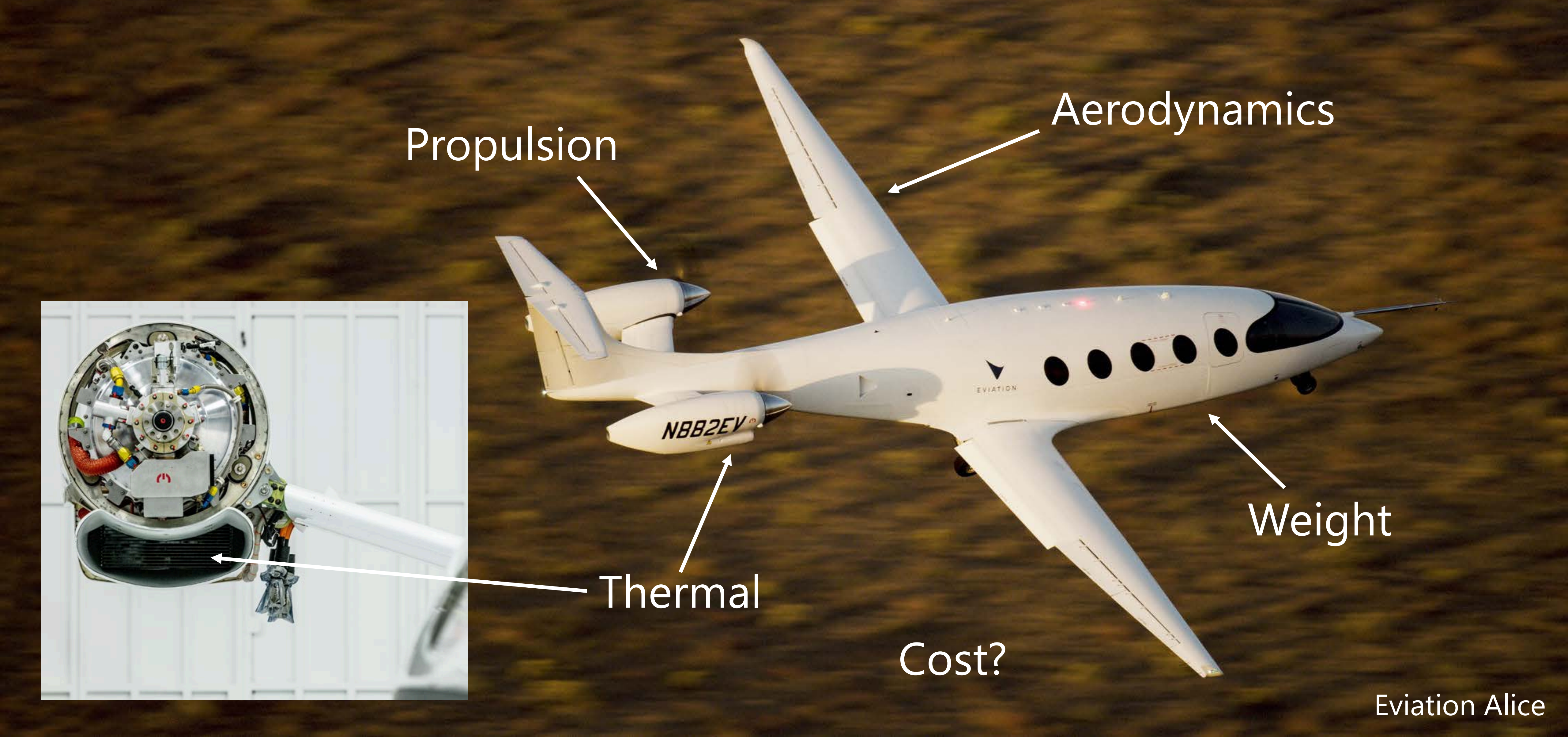
NASA Glenn

October 24–25th

Aerospace Engineering
University of Michigan

MDO Lab

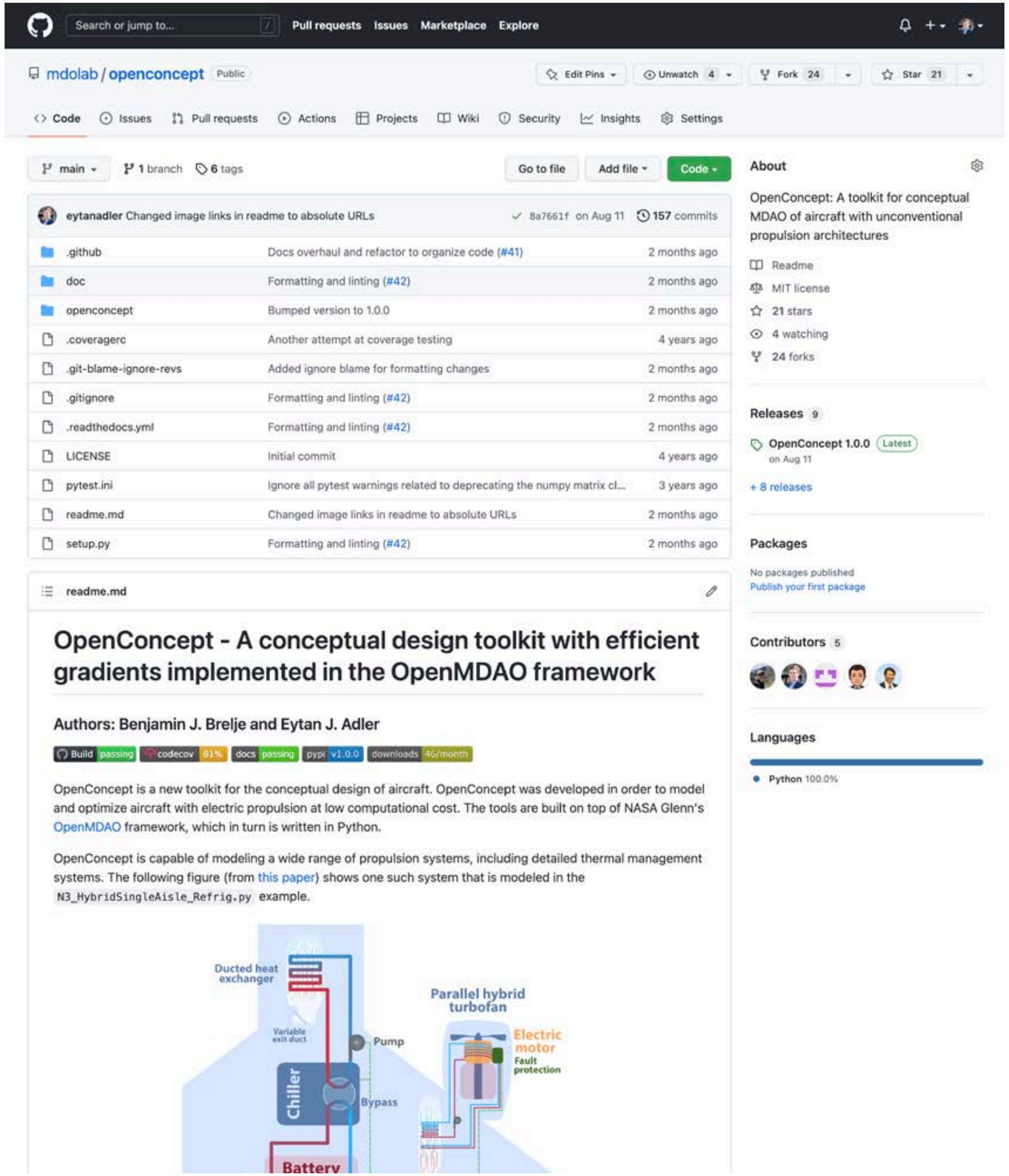
Conceptual design is multidisciplinary



Eviation Alice

OpenConcept

Open-source conceptual aircraft design tool built on OpenMDAO



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)
- $P_{\text{turboshaft}}$ (rated)
- $P_{\text{generator}}$ (rated)
- H_E (degree of hybridization w.r.t energy)

subject to scalar constraints:

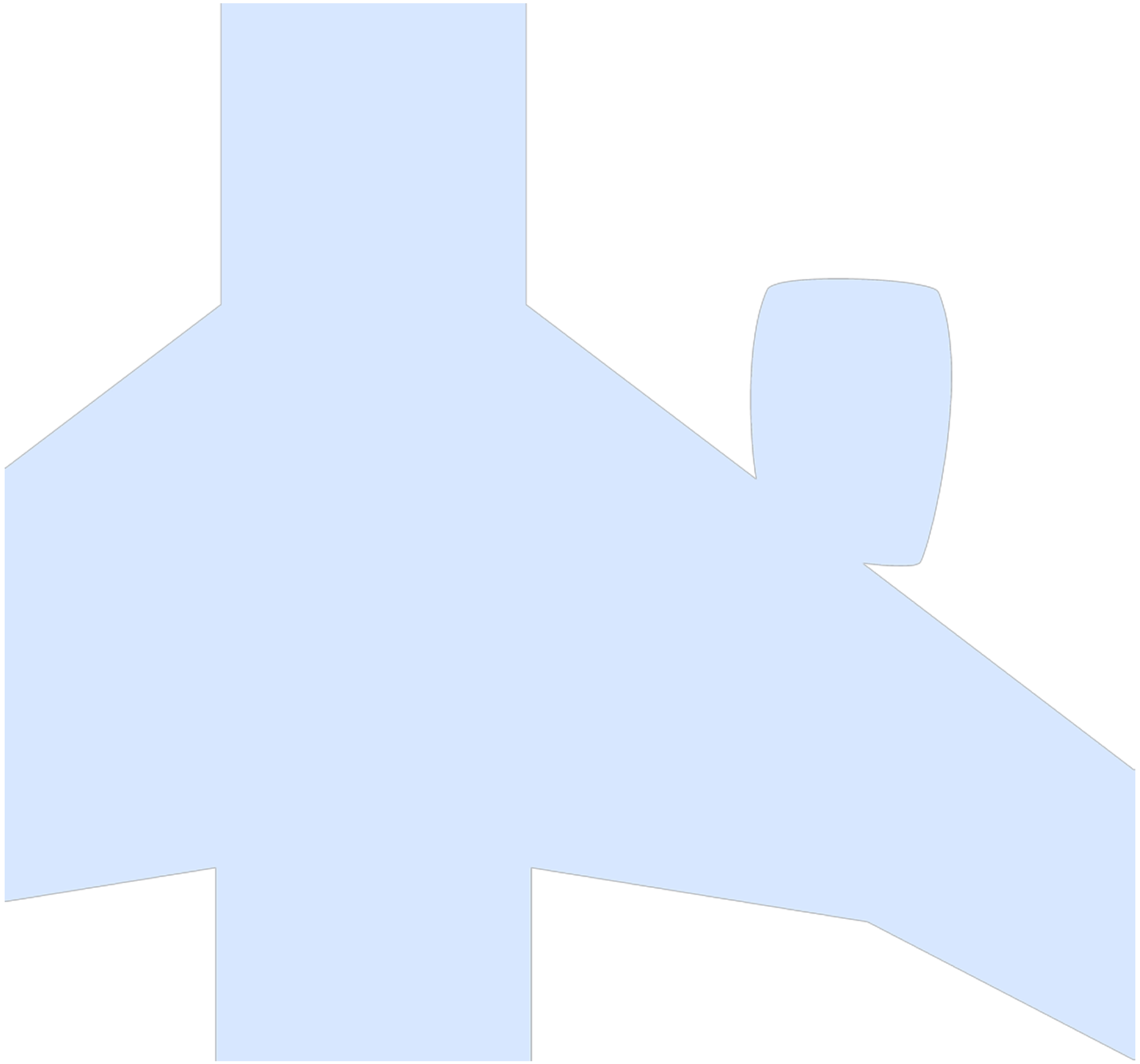
$$R_{\text{TOW}} = W_{\text{TO}} - W_{\text{fuel}} - W_{\text{empty}} - W_{\text{payload}} - W_{\text{batt}} \geq 0$$
$$R_{\text{batt}} = E_{\text{batt,max}} - E_{\text{batt,used}} \geq 0$$
$$R_{\text{vol}} = W_{\text{fuel,max}} - W_{\text{fuel}} \geq 0$$
$$\text{BFL} \leq 4452 \text{ ft (no worse than baseline)}$$
$$V_{\text{stall}} \leq 81.6 \text{ kt (no worse than baseline)}$$

and vector constraints:

$$\vec{P}_{\text{motor}} \leq 1.05 P_{\text{motor}} \text{ (rated)}$$
$$\vec{P}_{\text{turboshaft}} \leq P_{\text{turboshaft}} \text{ (rated)}$$
$$\vec{P}_{\text{generator}} \leq P_{\text{generator}} \text{ (rated)}$$
$$\vec{P}_{\text{battery}} \leq W_{\text{battery}} \cdot p_b$$

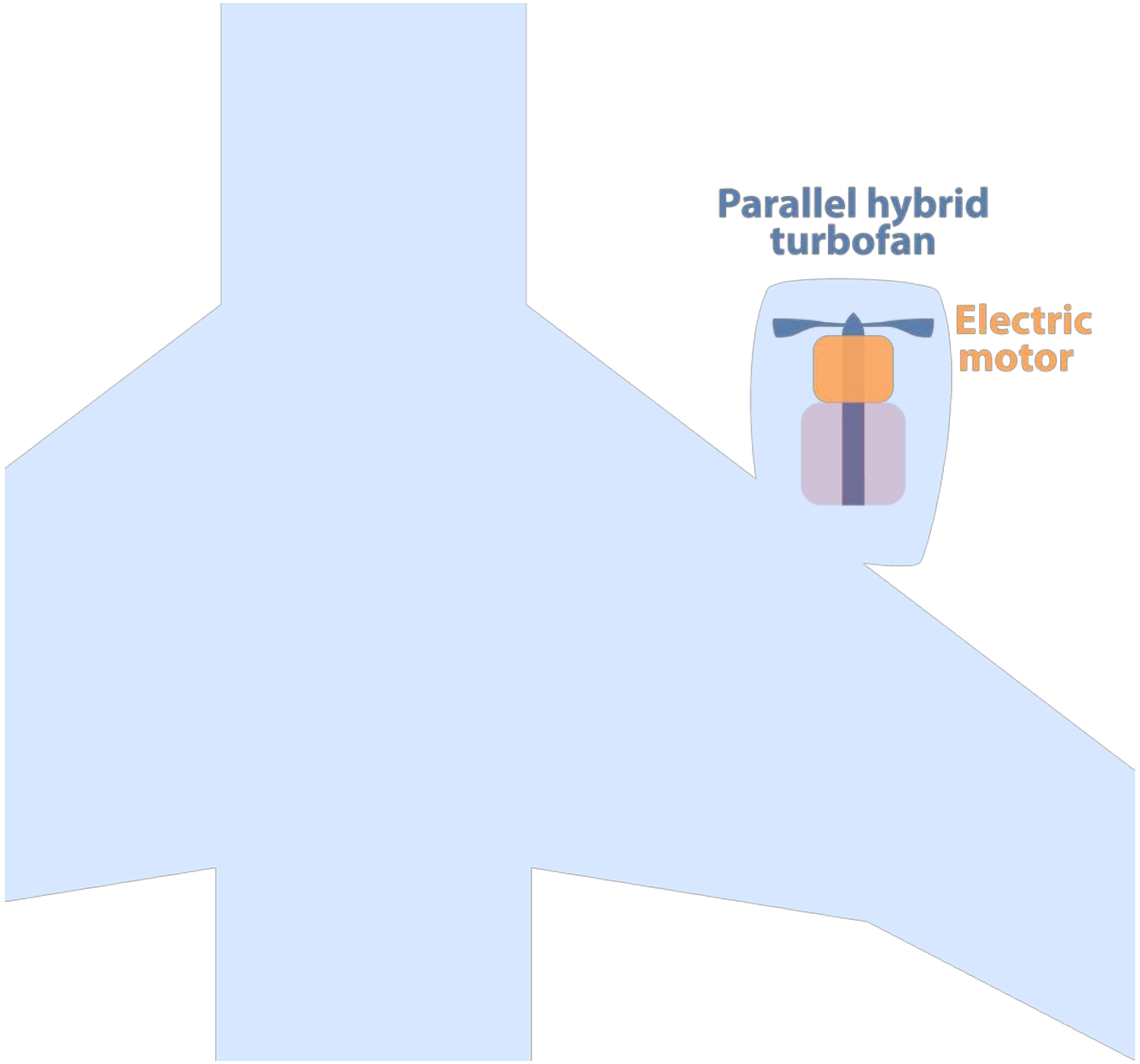
Brelje and Martins, “Development of a Conceptual Design Model for Aircraft Electric Propulsion with Efficient Gradients”, 2018.

But we can do a lot more…



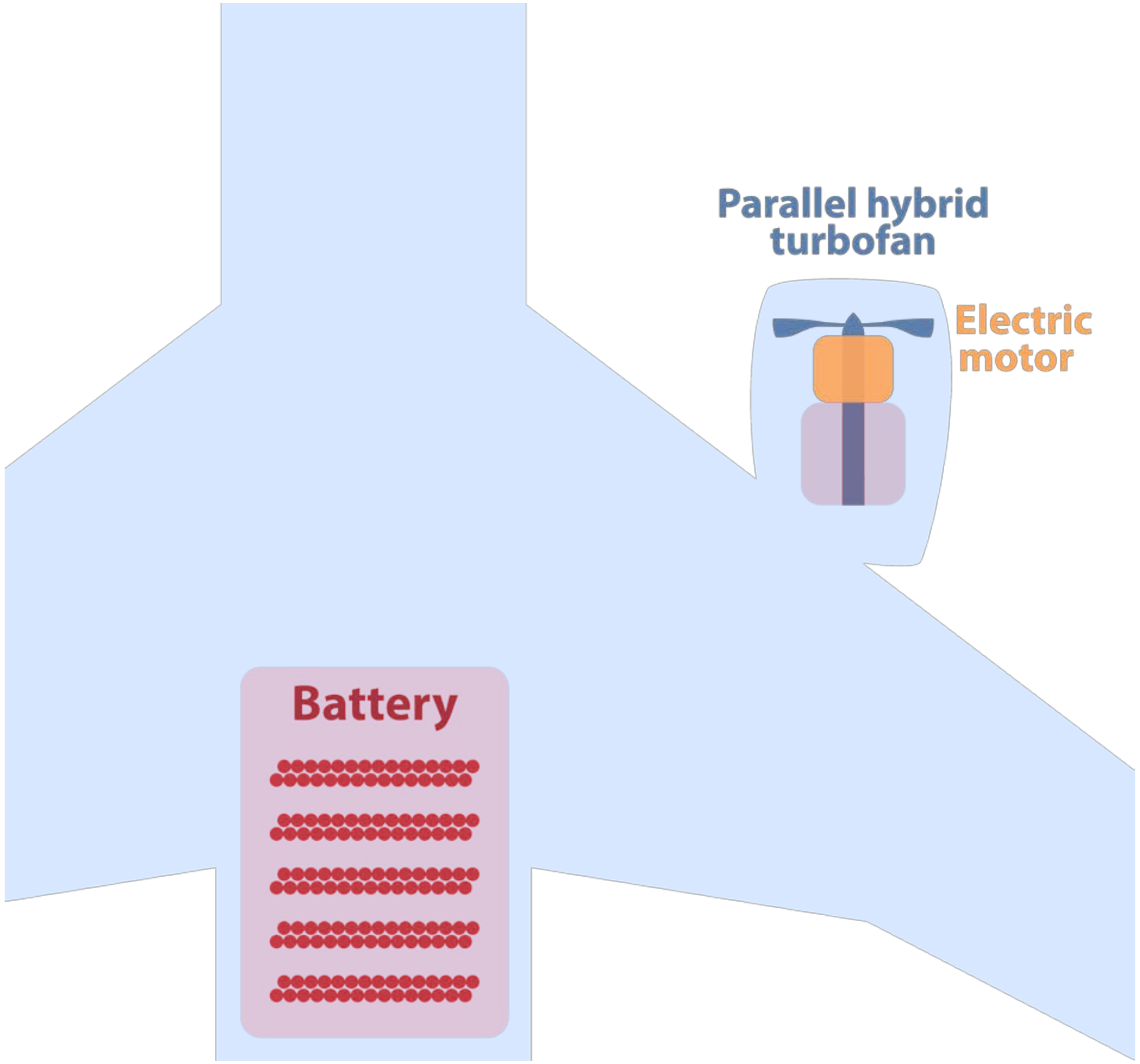
Adler, Brelje, and Martins, "Thermal Management System Optimization for a Parallel Hybrid Aircraft Considering Mission Fuel Burn", 2022.

Start with a parallel hybrid turbofan



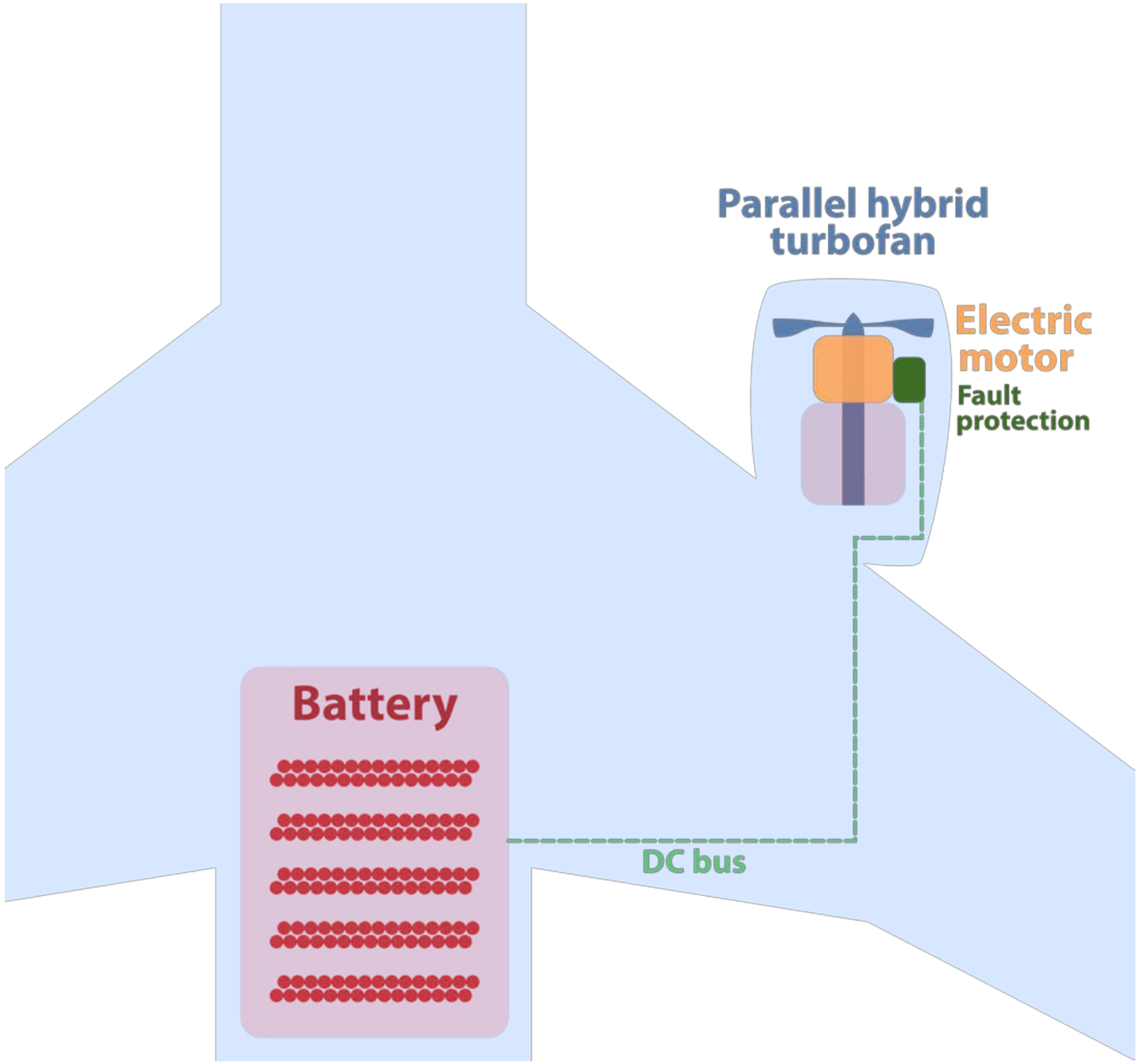
Adler, Brelje, and Martins, "Thermal Management System Optimization for a Parallel Hybrid Aircraft Considering Mission Fuel Burn", 2022.

Battery powers the electric motor



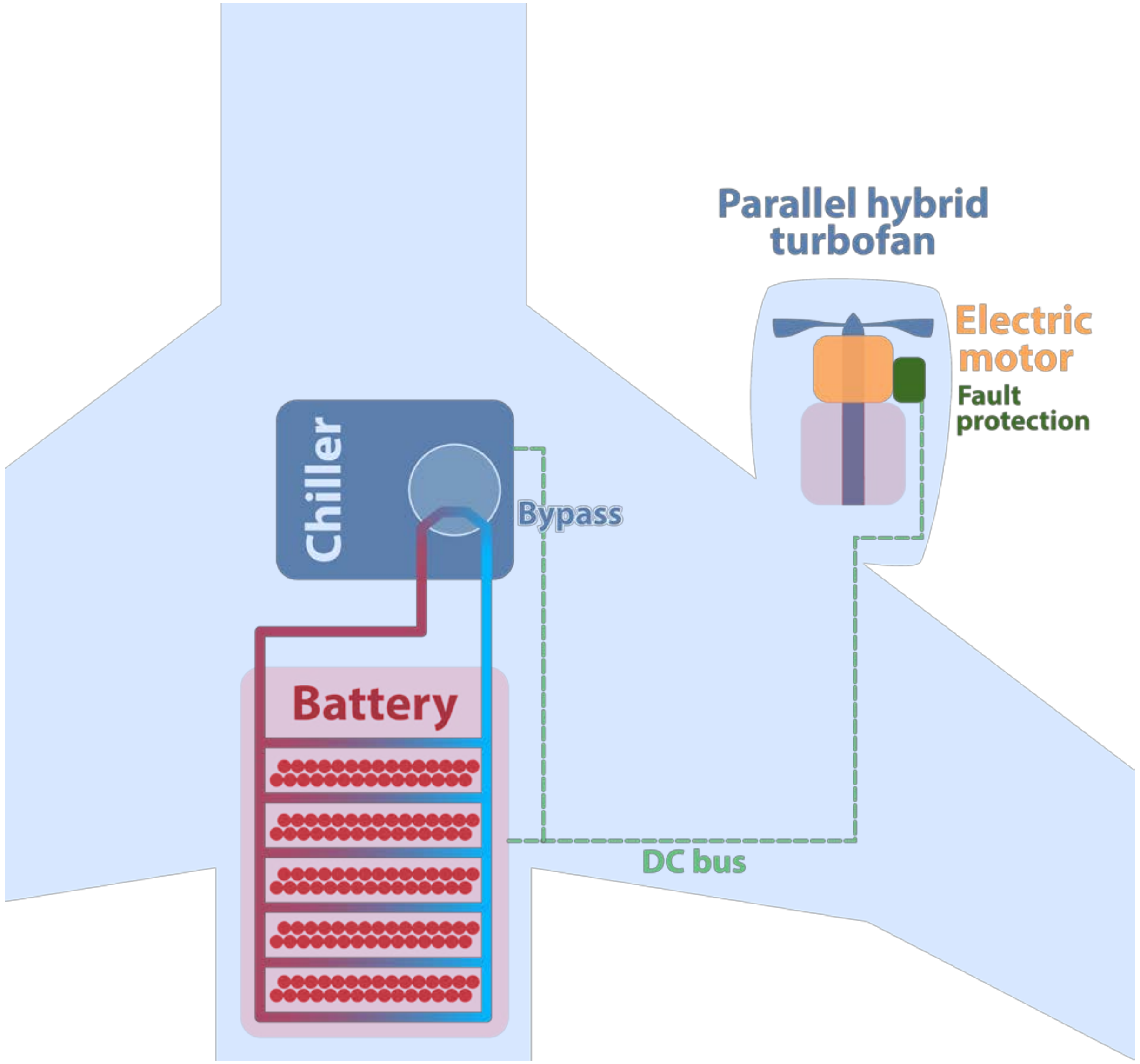
Adler, Brelje, and Martins, "Thermal Management System Optimization for a Parallel Hybrid Aircraft Considering Mission Fuel Burn", 2022.

Battery powers the electric motor



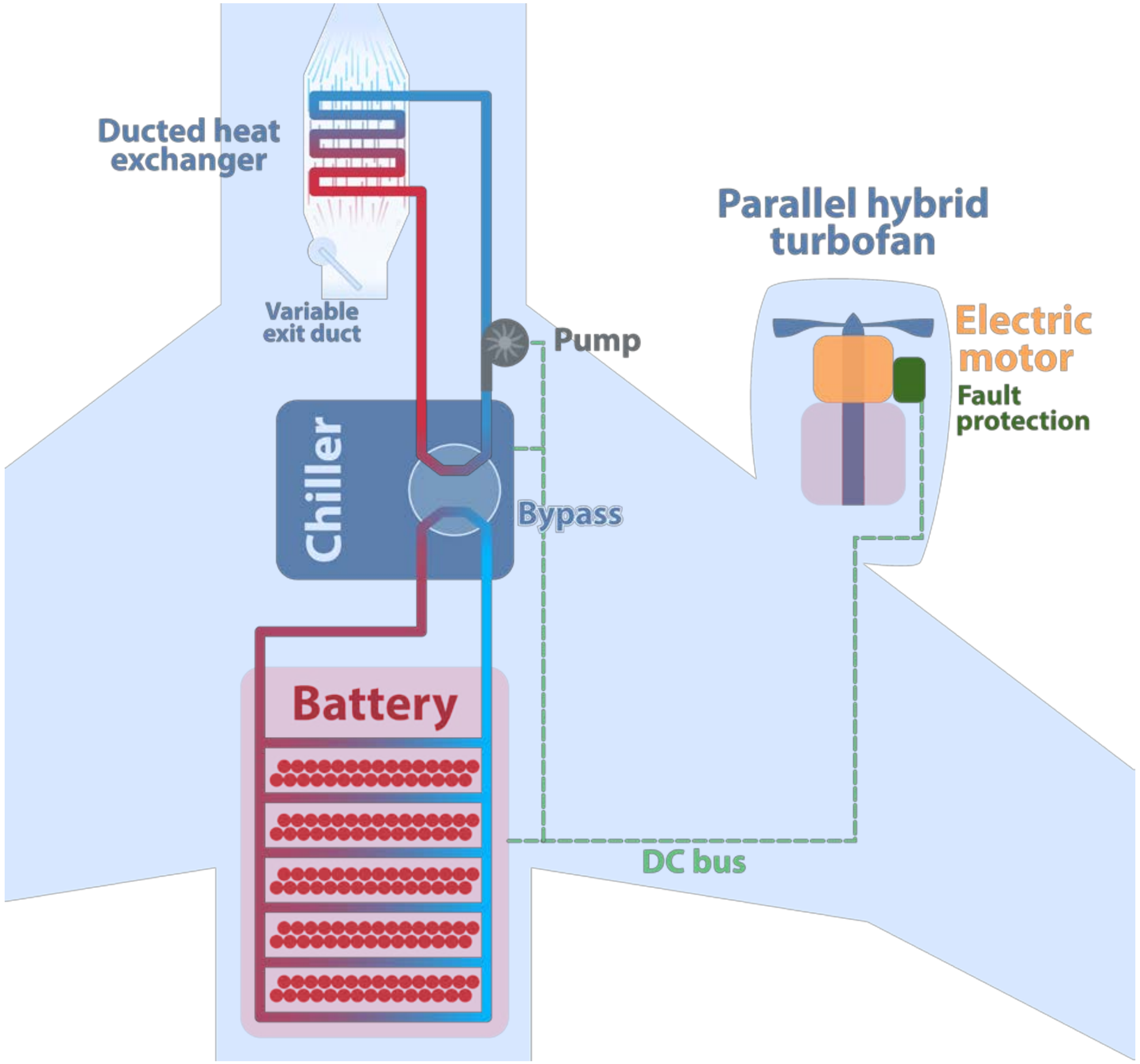
Adler, Brelje, and Martins, "Thermal Management System Optimization for a Parallel Hybrid Aircraft Considering Mission Fuel Burn", 2022.

Cool the battery with a refrigerator



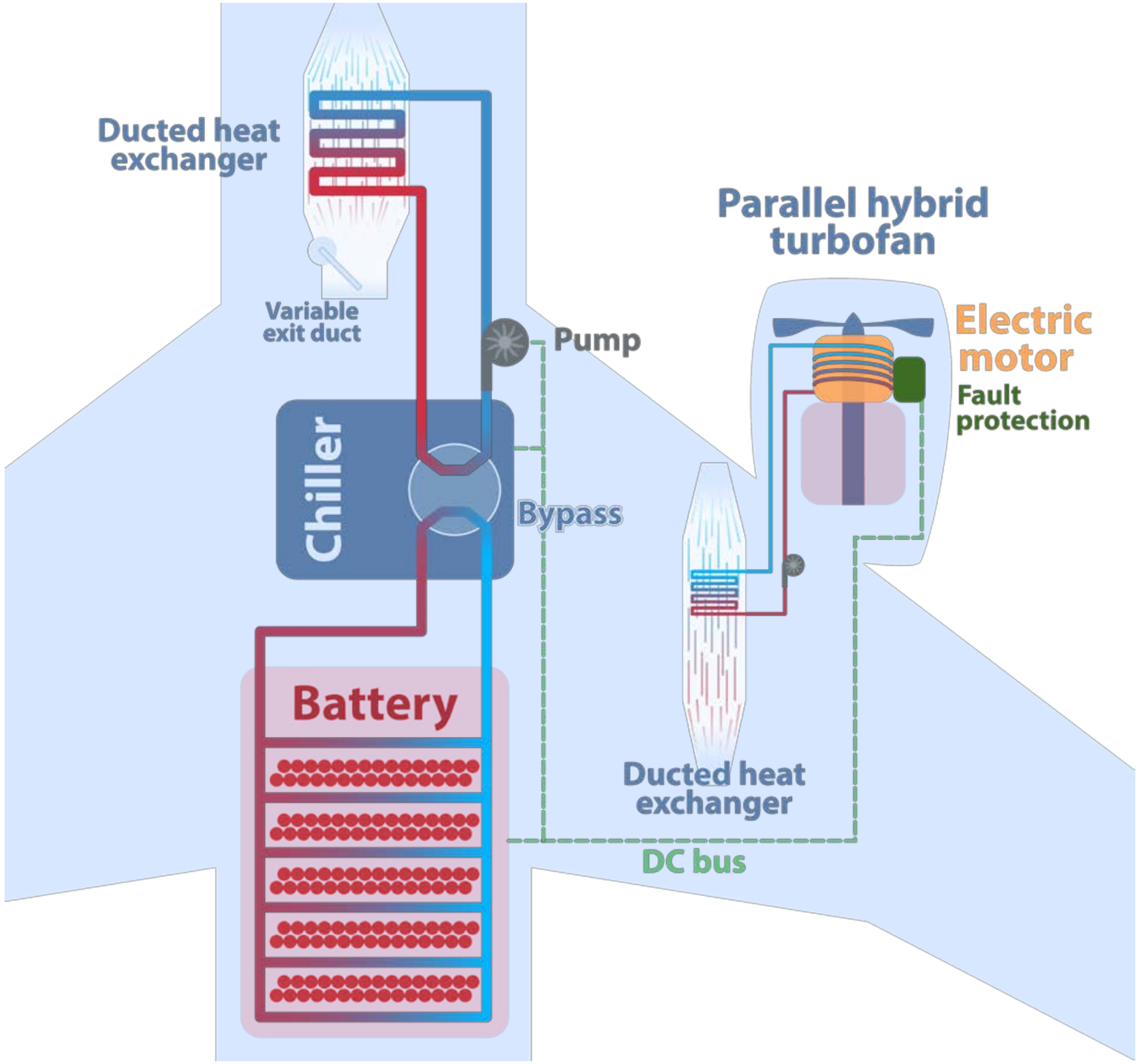
Adler, Brelje, and Martins, "Thermal Management System Optimization for a Parallel Hybrid Aircraft Considering Mission Fuel Burn", 2022.

Refrigerator dumps heat into freestream



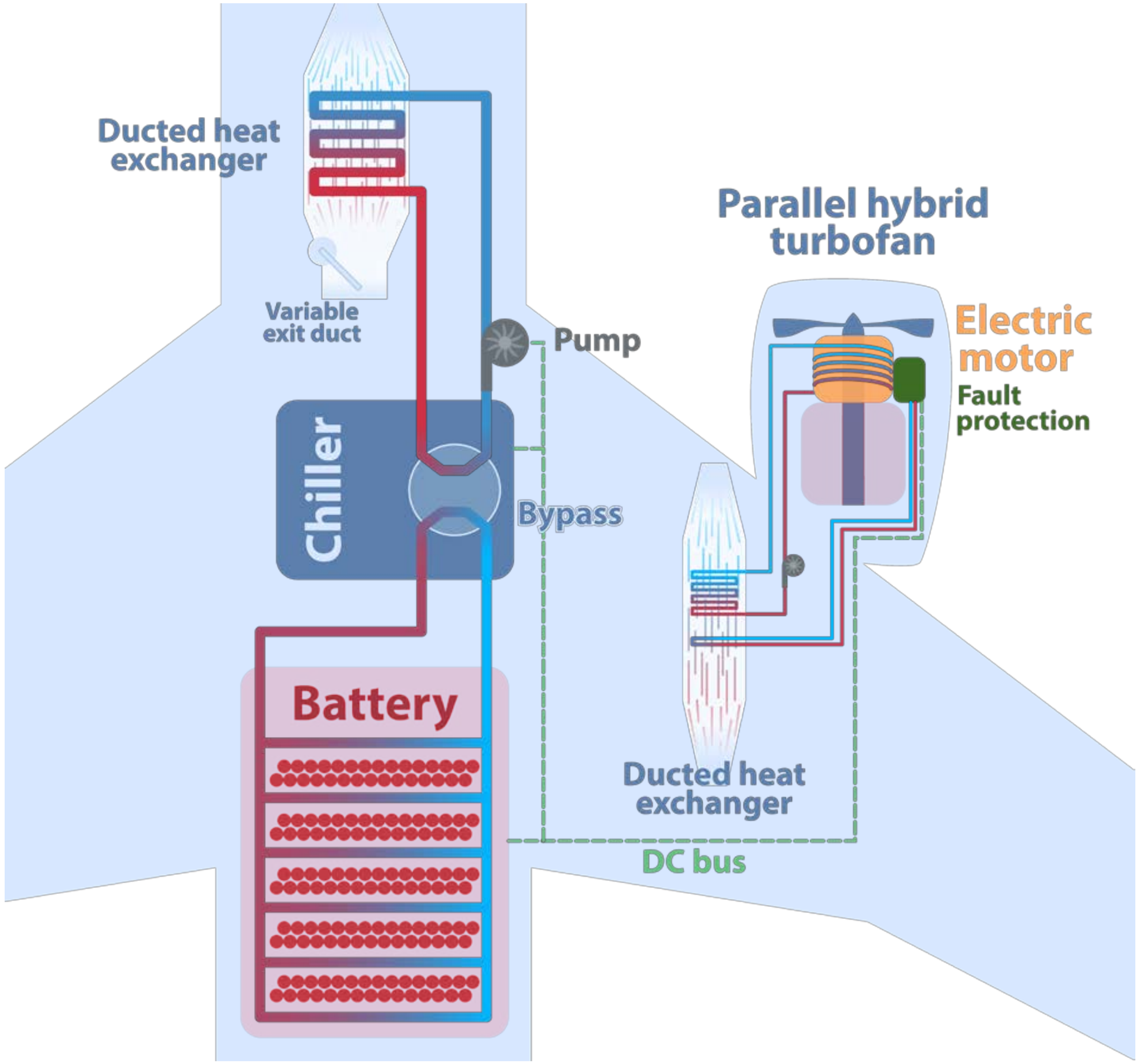
Adler, Brelje, and Martins, "Thermal Management System Optimization for a Parallel Hybrid Aircraft Considering Mission Fuel Burn", 2022.

Same for the electric motor



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

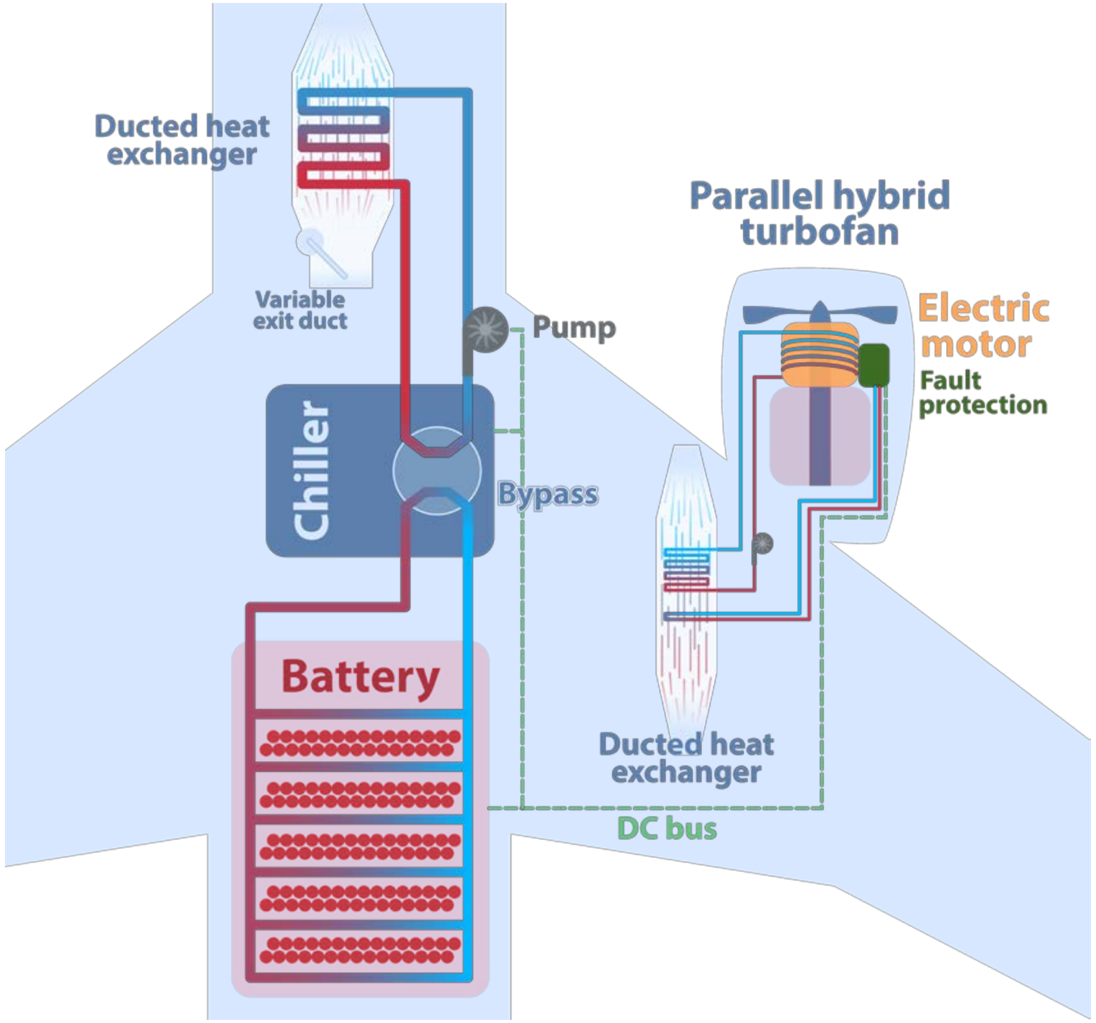
…and the fault protection



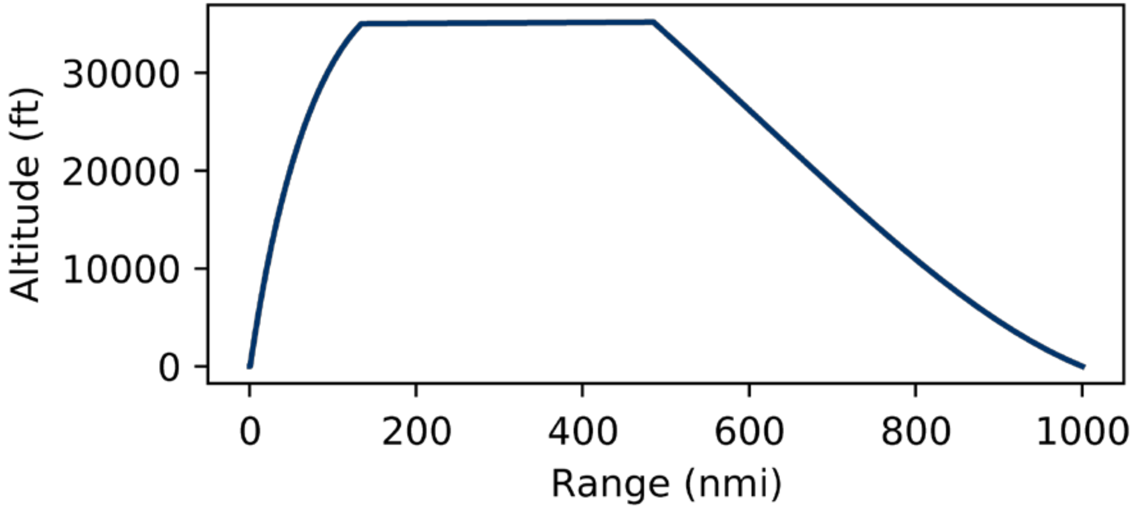
Adler, Brelje, and Martins, "Thermal Management System Optimization for a Parallel Hybrid Aircraft Considering Mission Fuel Burn", 2022.

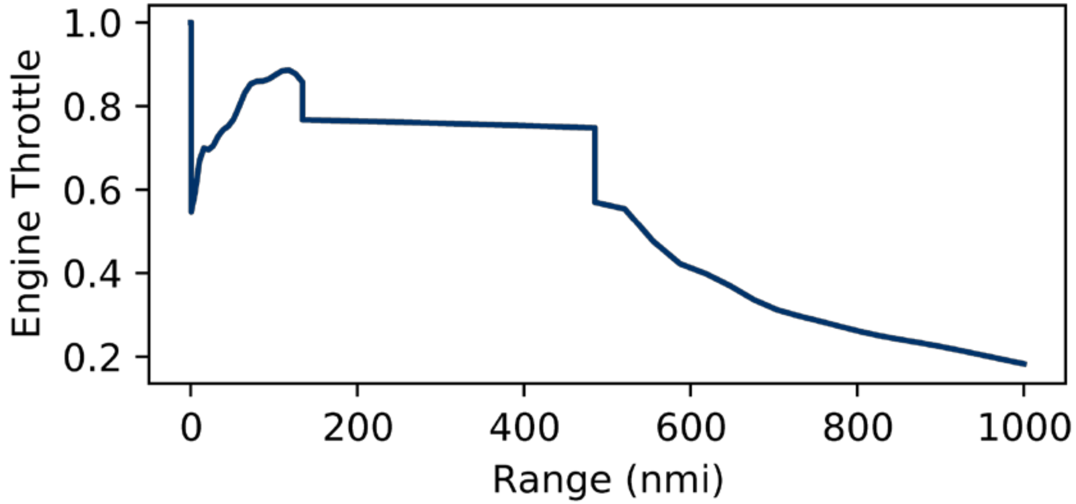
And there is still more complexity

- 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

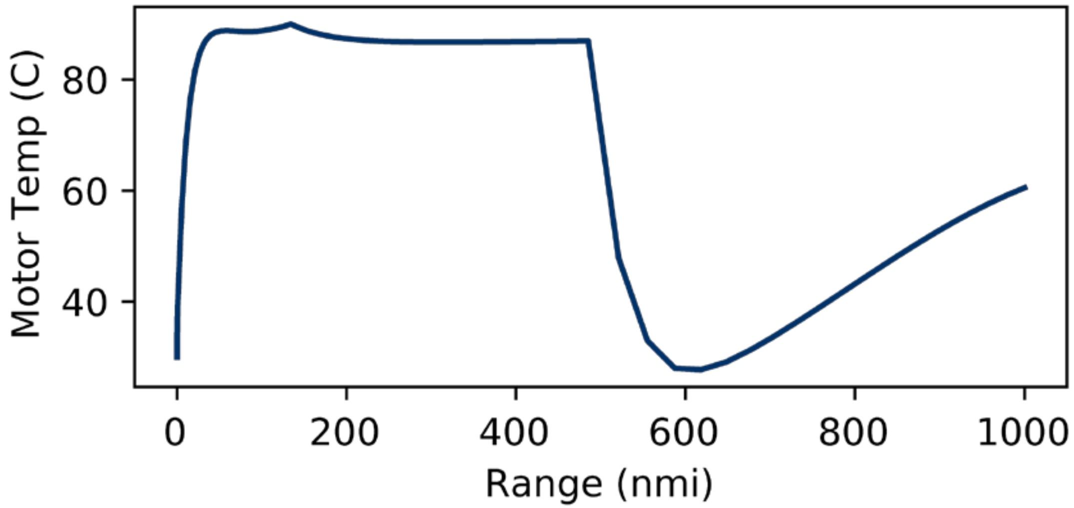


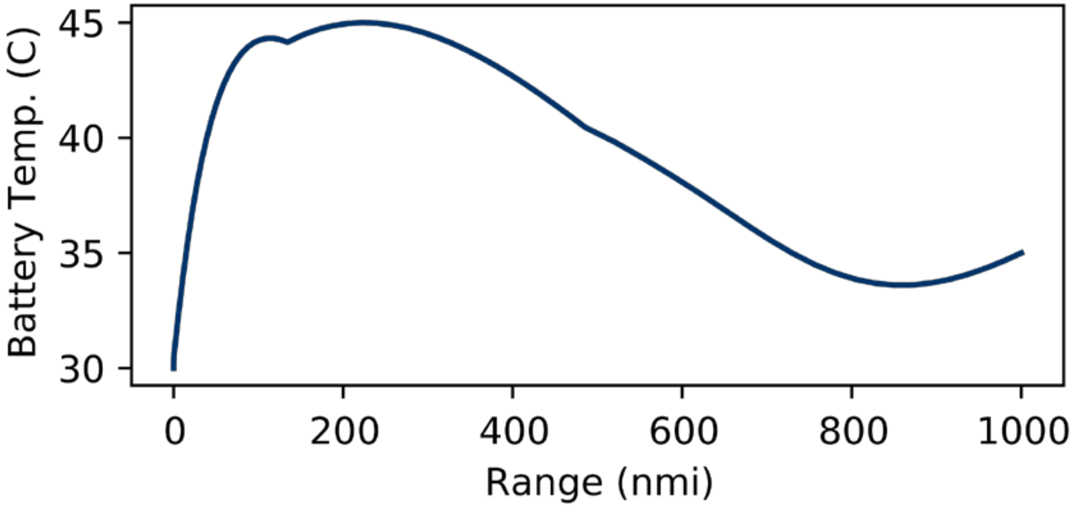
After running a mission, we can…

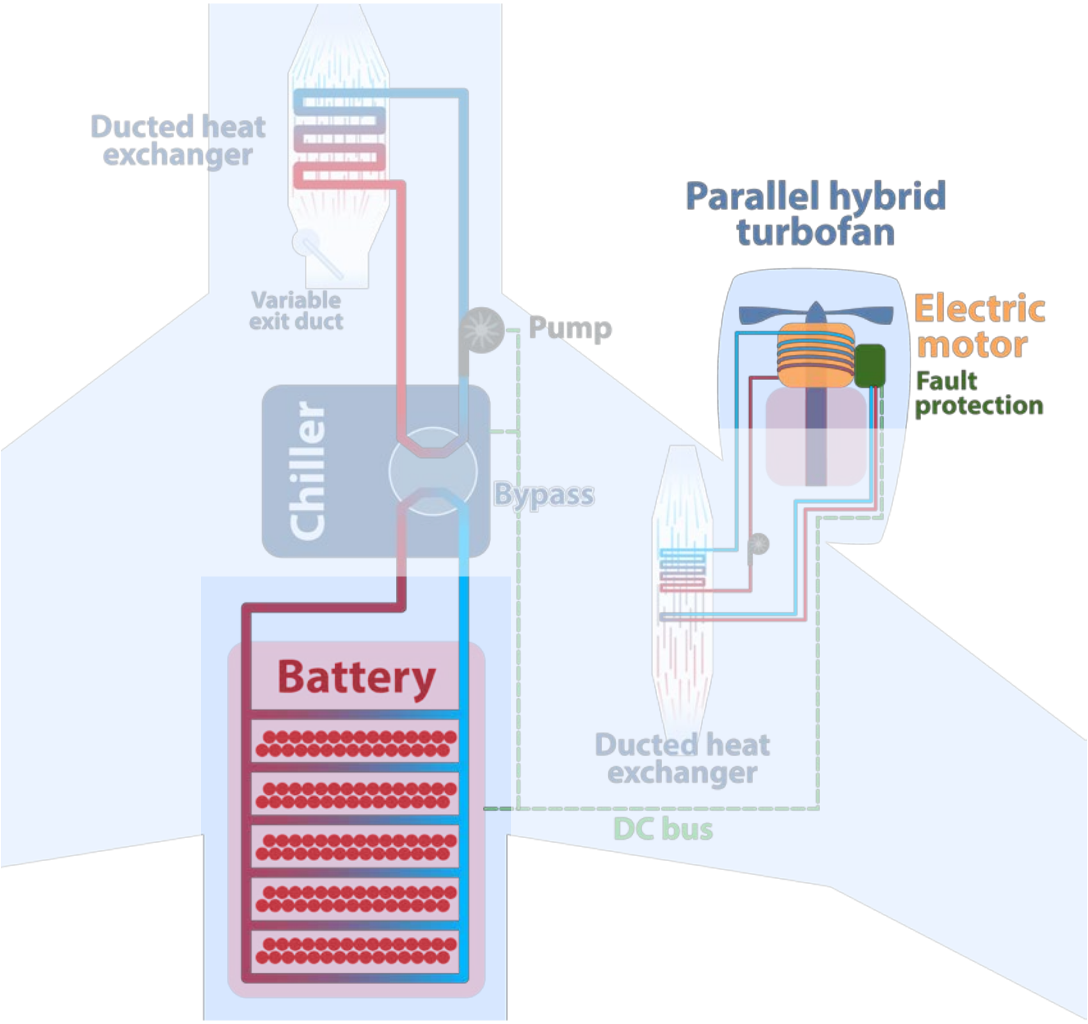




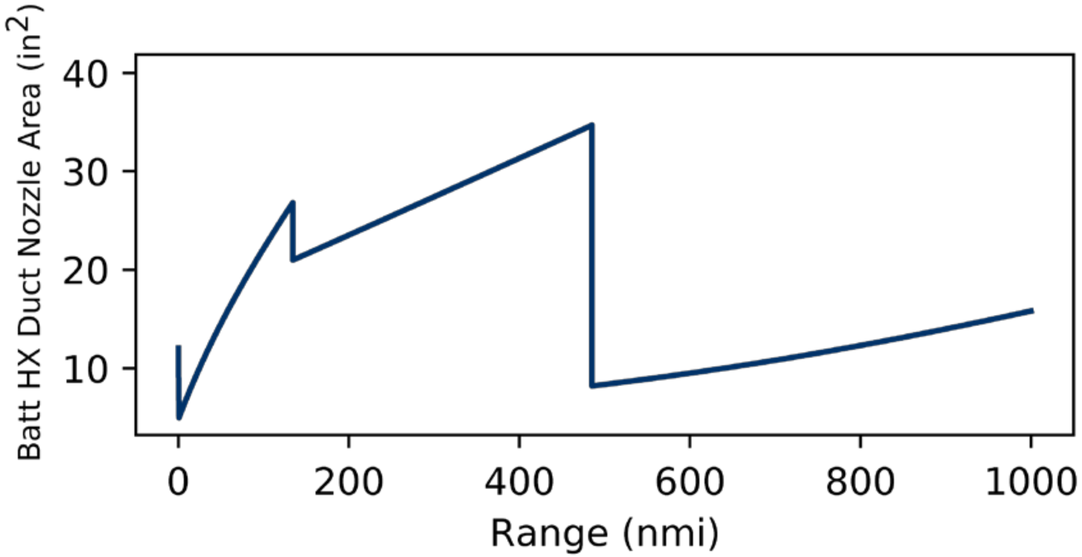
…analyze component temperatures

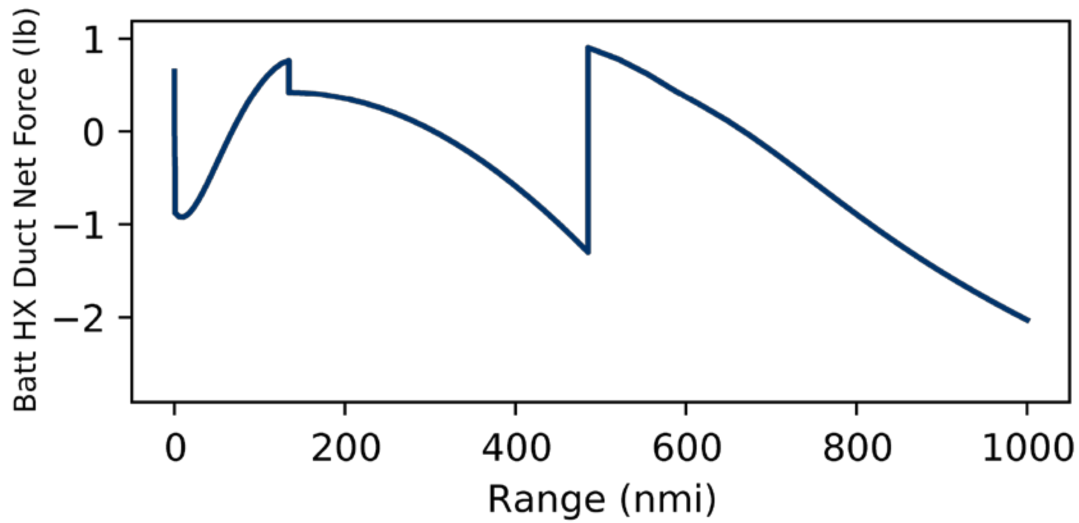


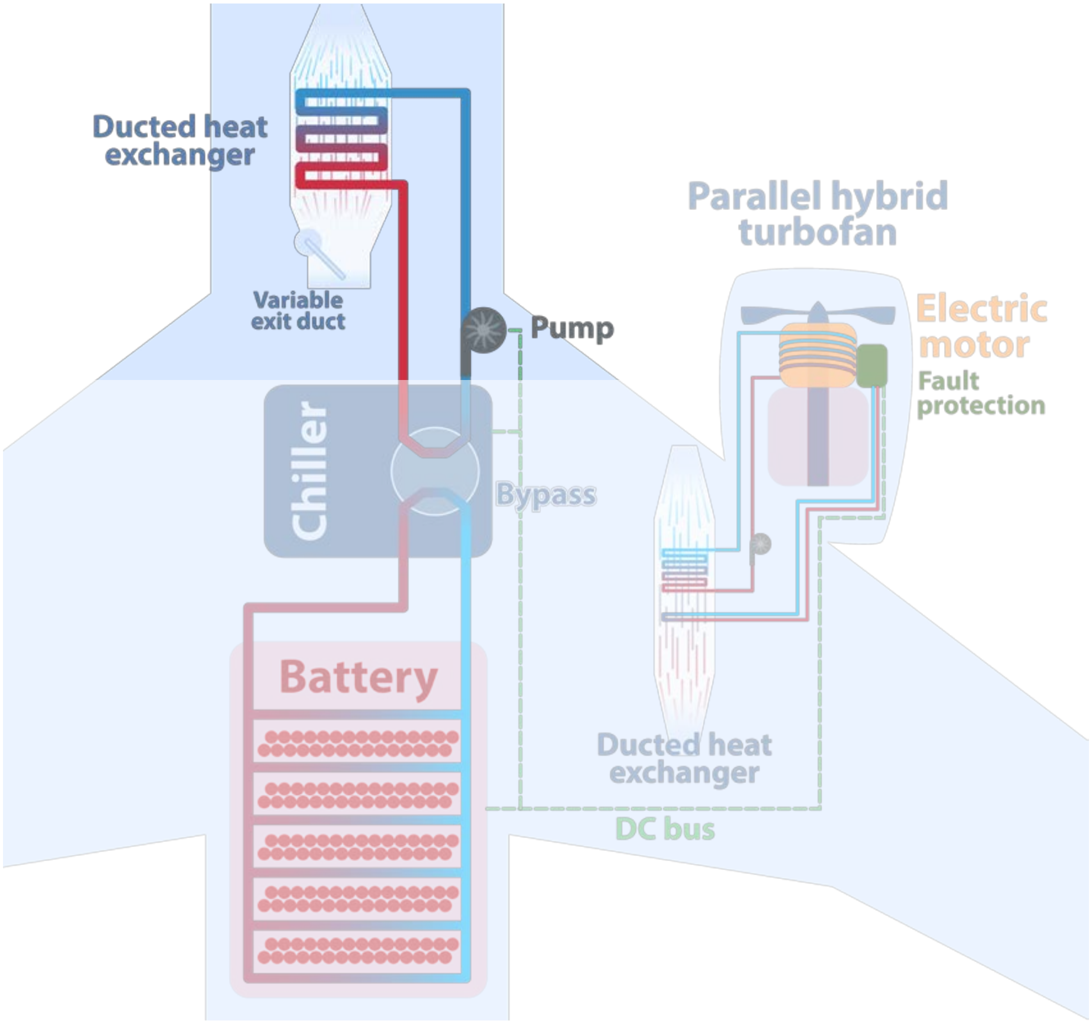




...and optimize duct area in time







Code flexibility

Mission analysis

Lessons learned

Code flexibility

Mission analysis

Lessons learned

Flexible ↔ **Easy to use**

Flexible

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

Easy to use

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

Flexible
Easy to use
OpenConcept

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

```python
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=["*"],
        )
```

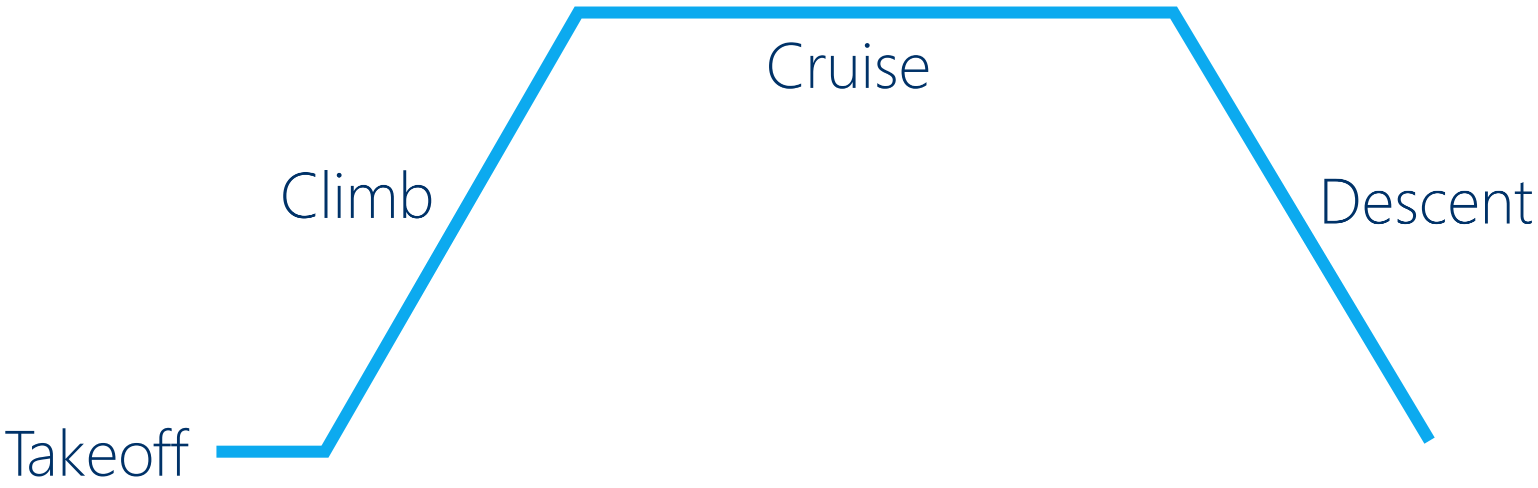
Code flexibility

Mission analysis

Lessons learned

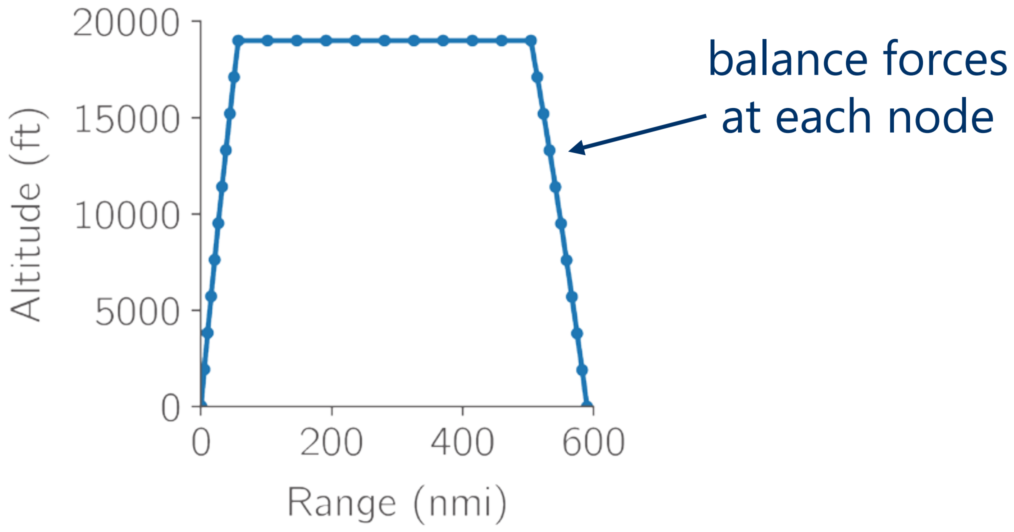
What is mission analysis?

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

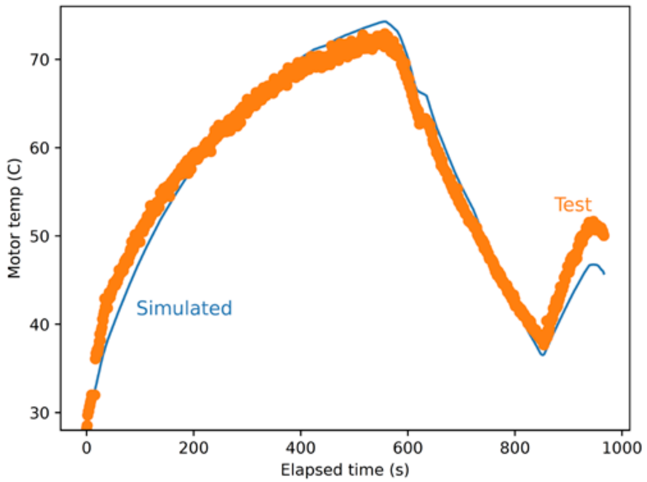


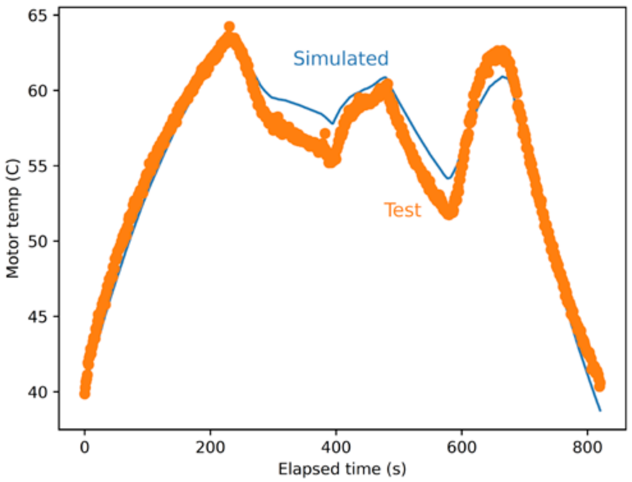
We assume steady flight at integration points

- This avoids working directly with the equations of motion

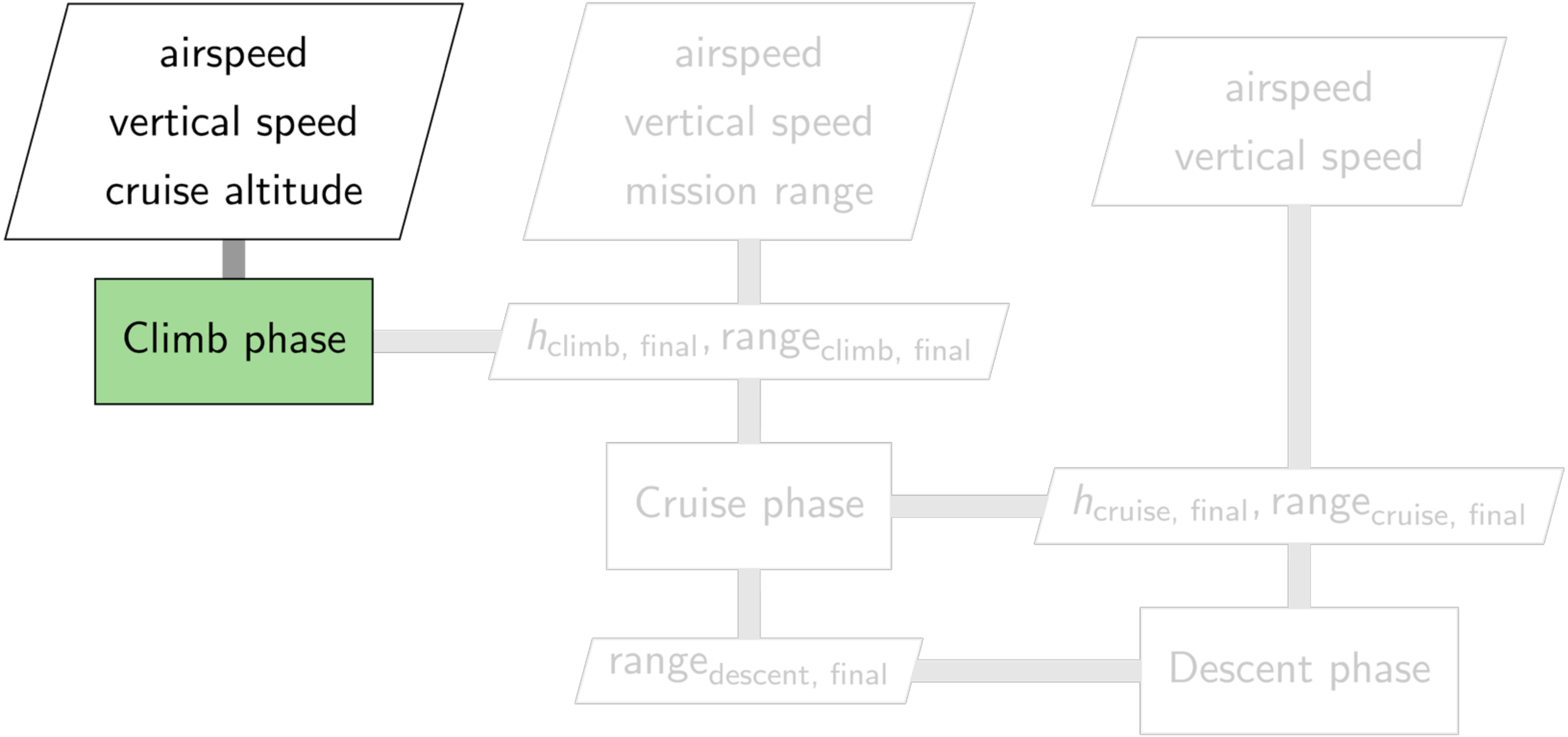


- Validated against real world data from Pipistrel

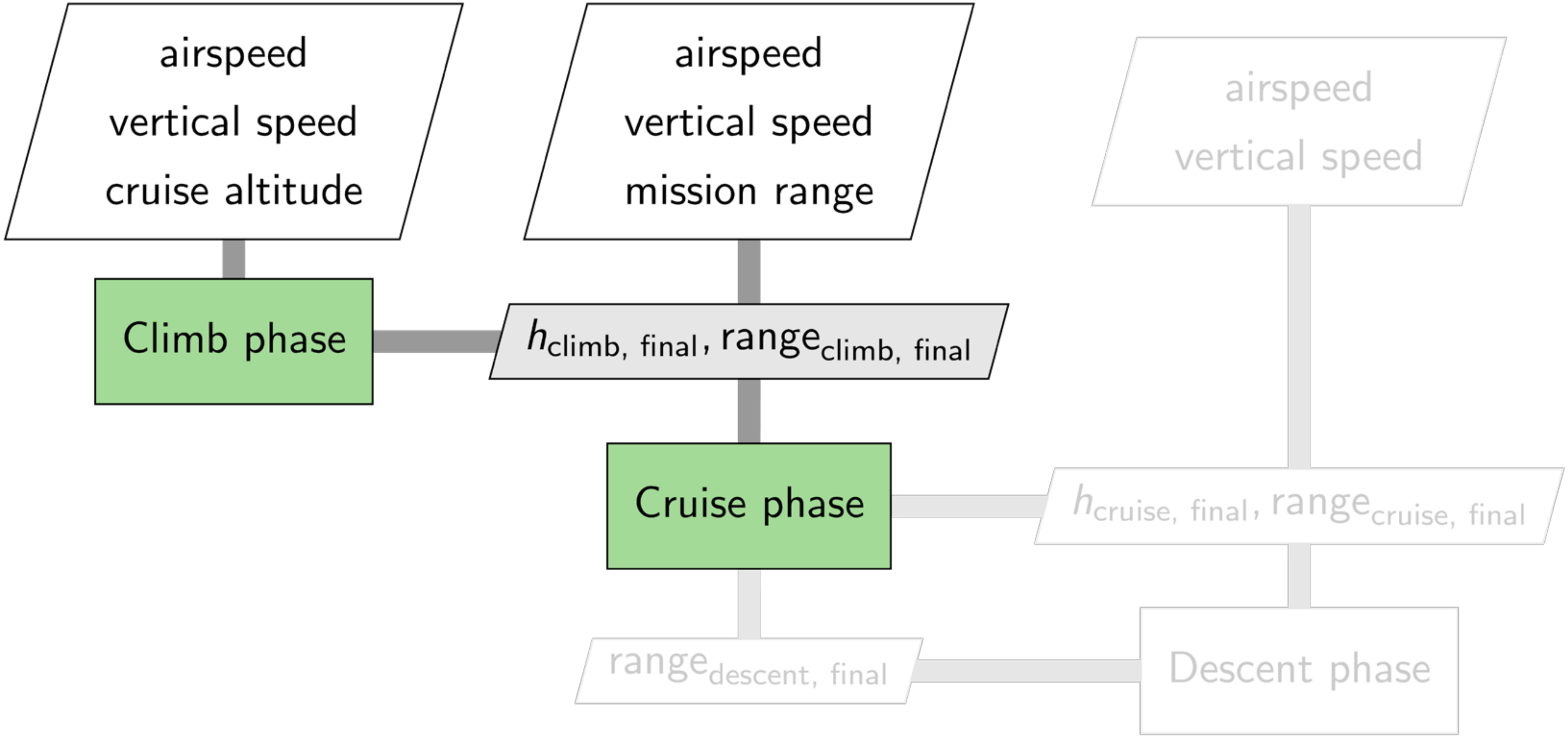




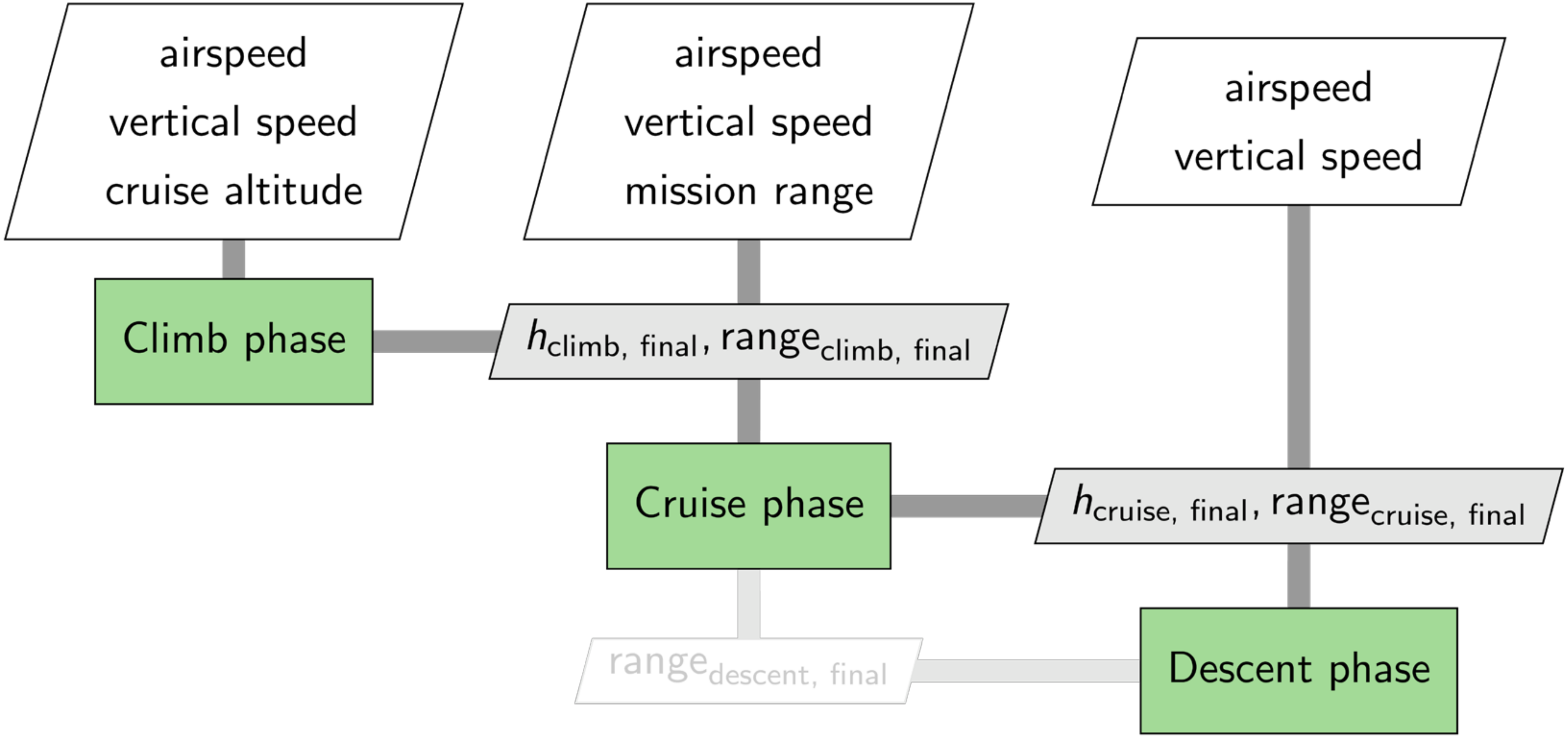
Basic mission setup



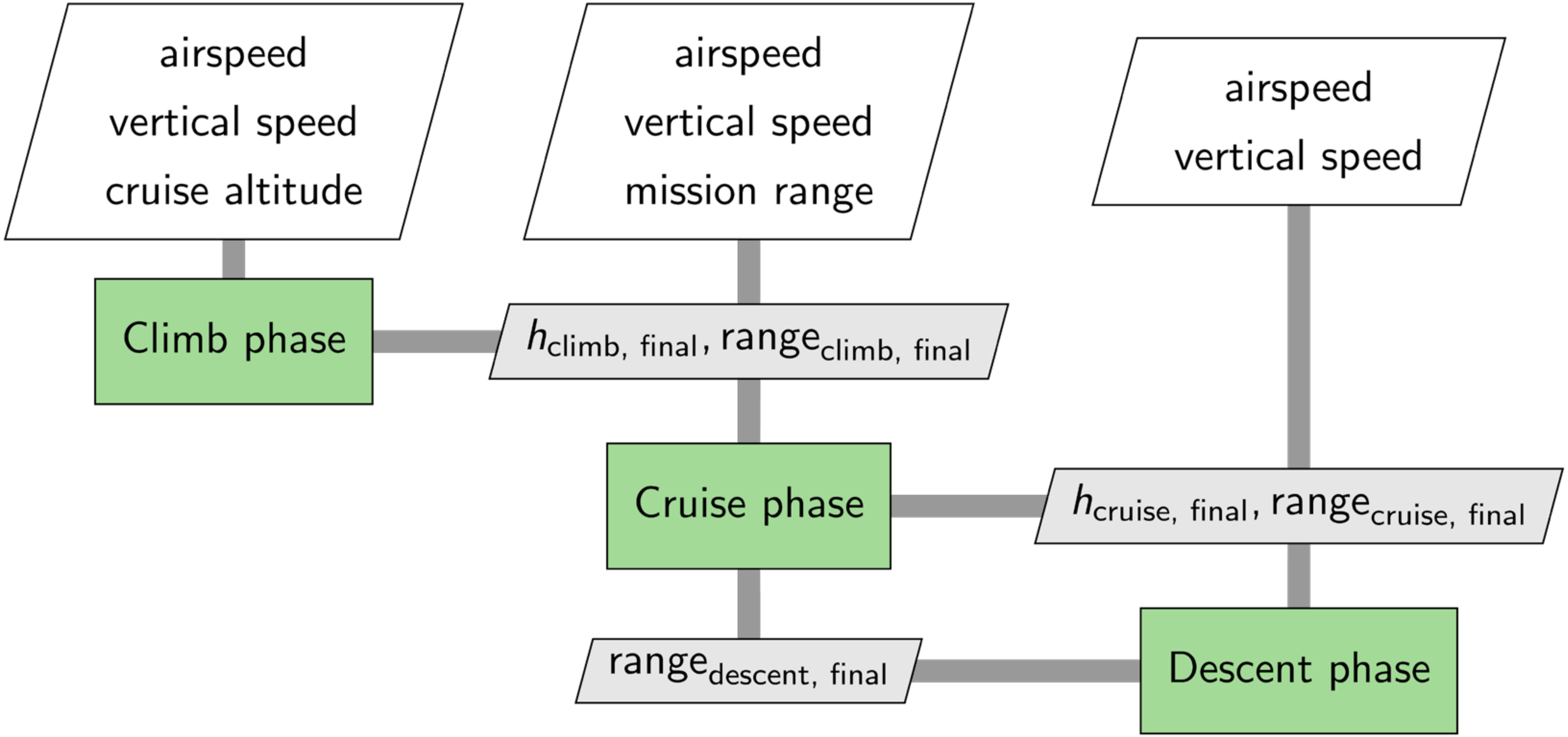
Basic mission setup



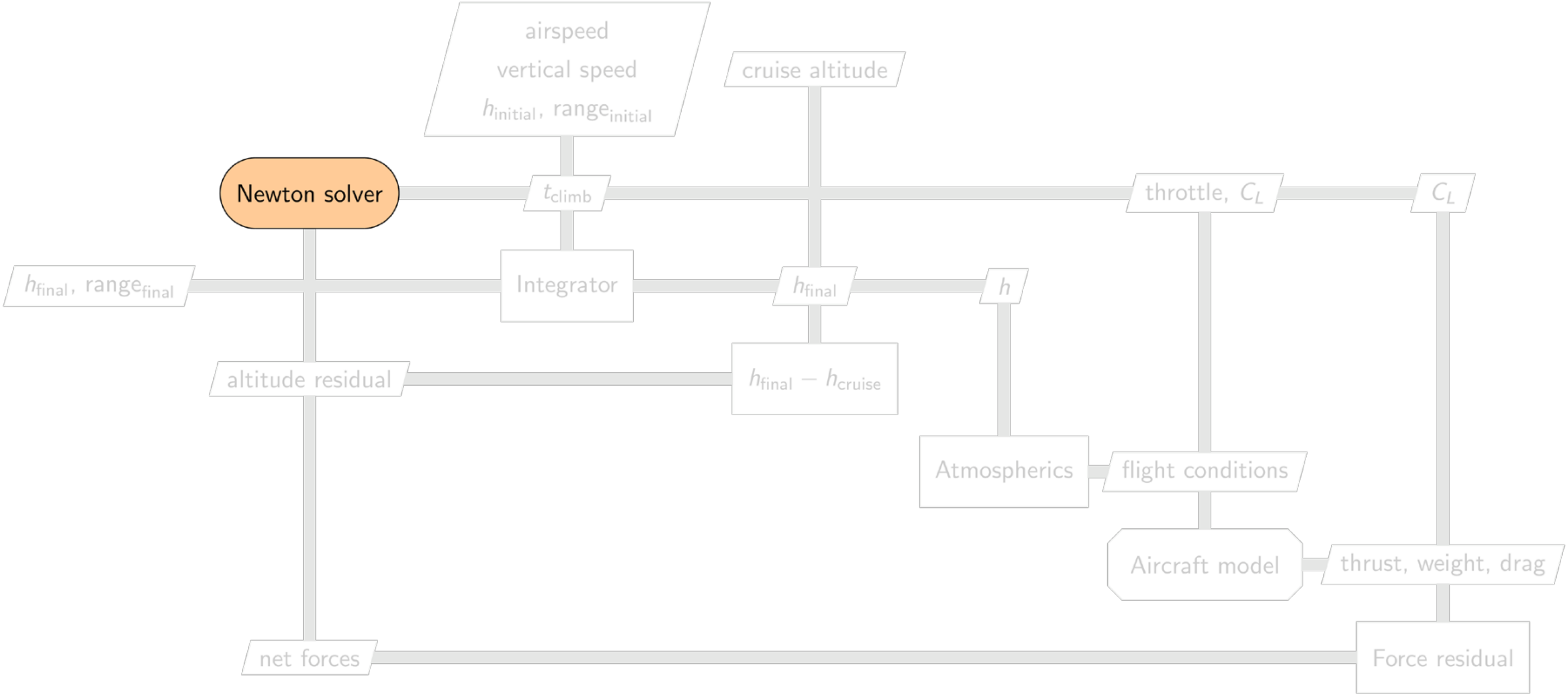
Basic mission setup



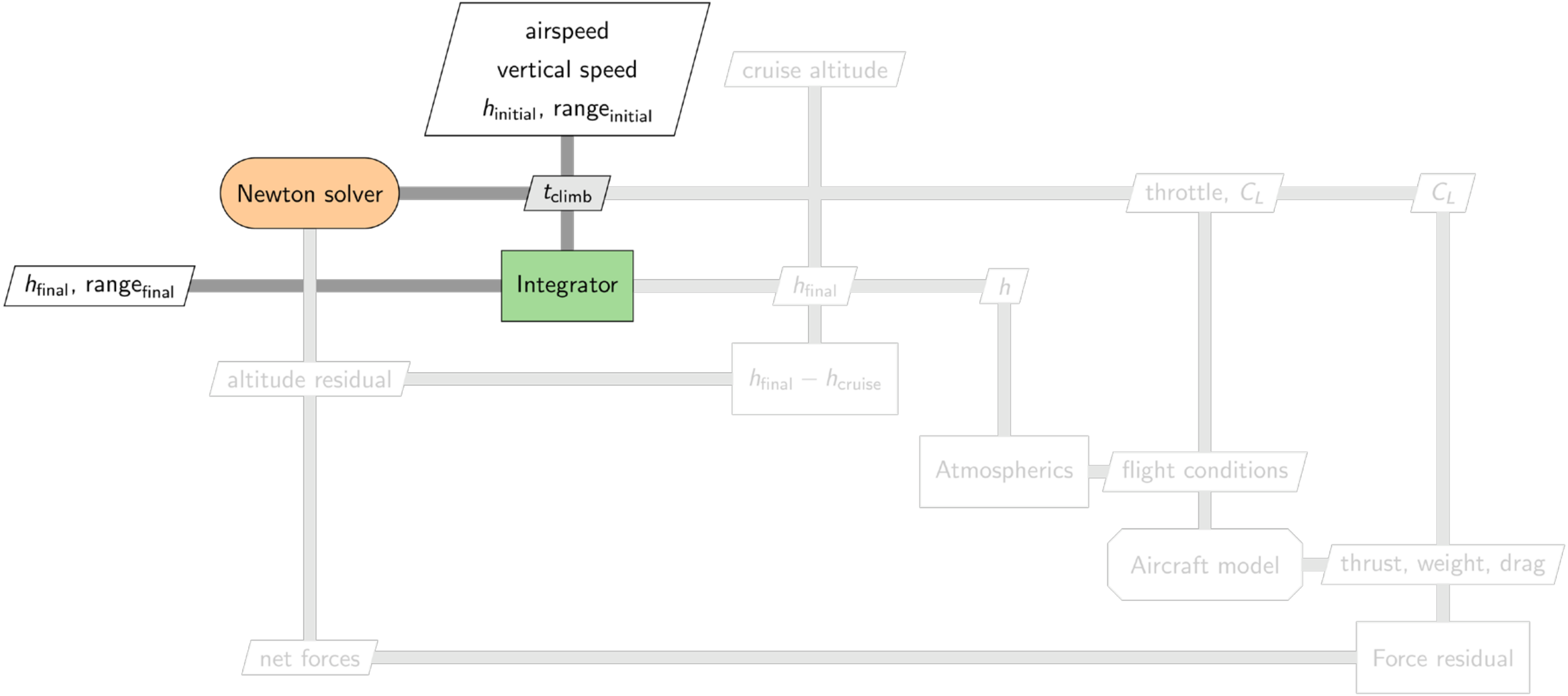
Basic mission setup



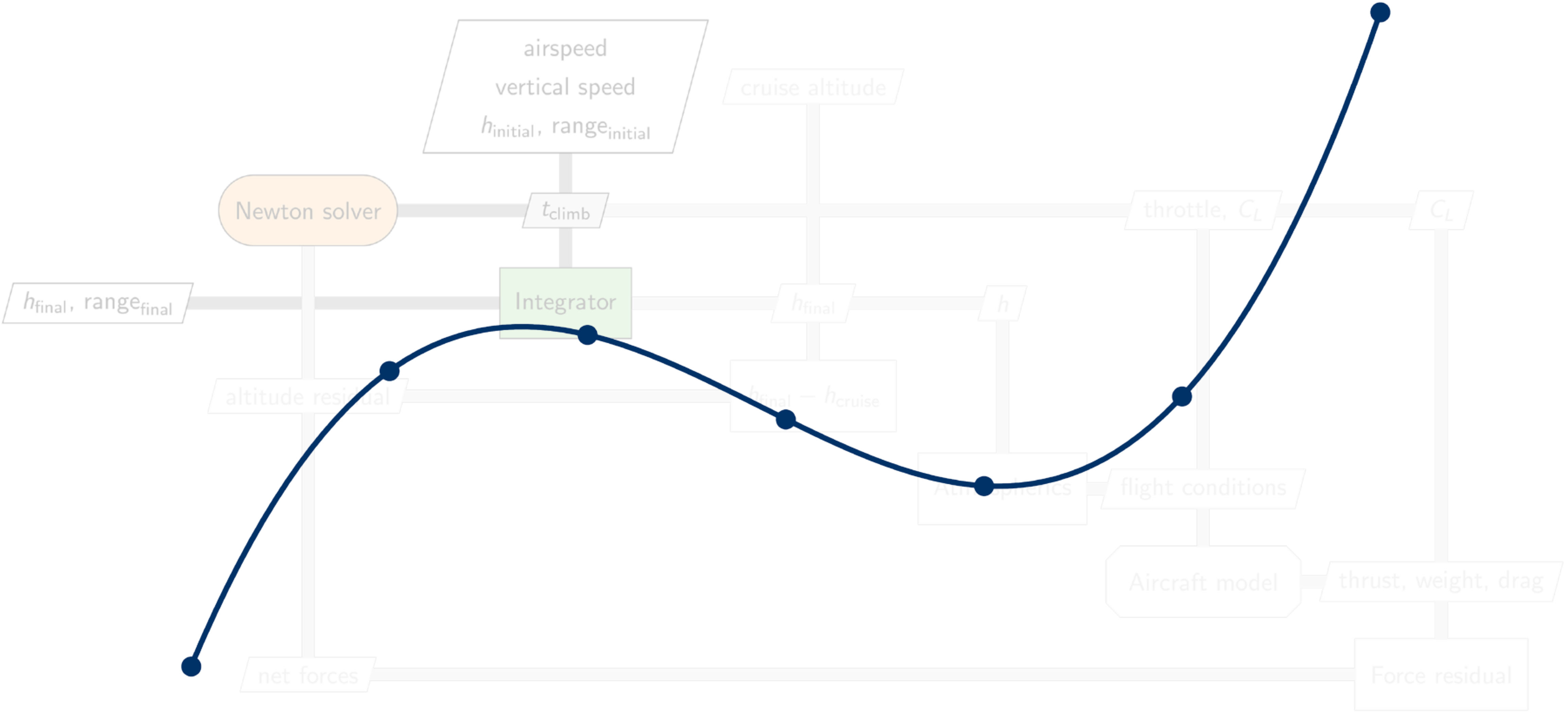
Solving an individual flight phase (climb)



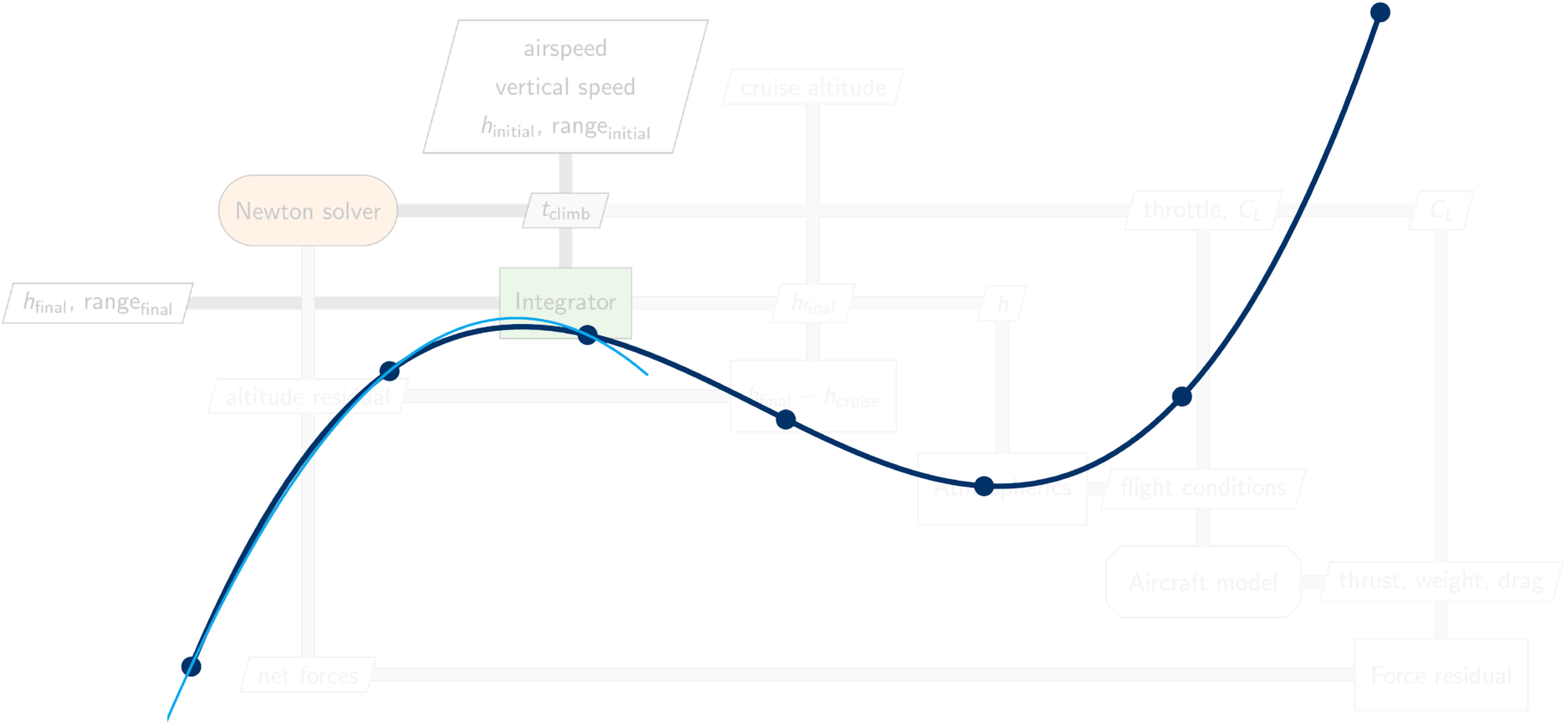
Integrator solves for altitude and range



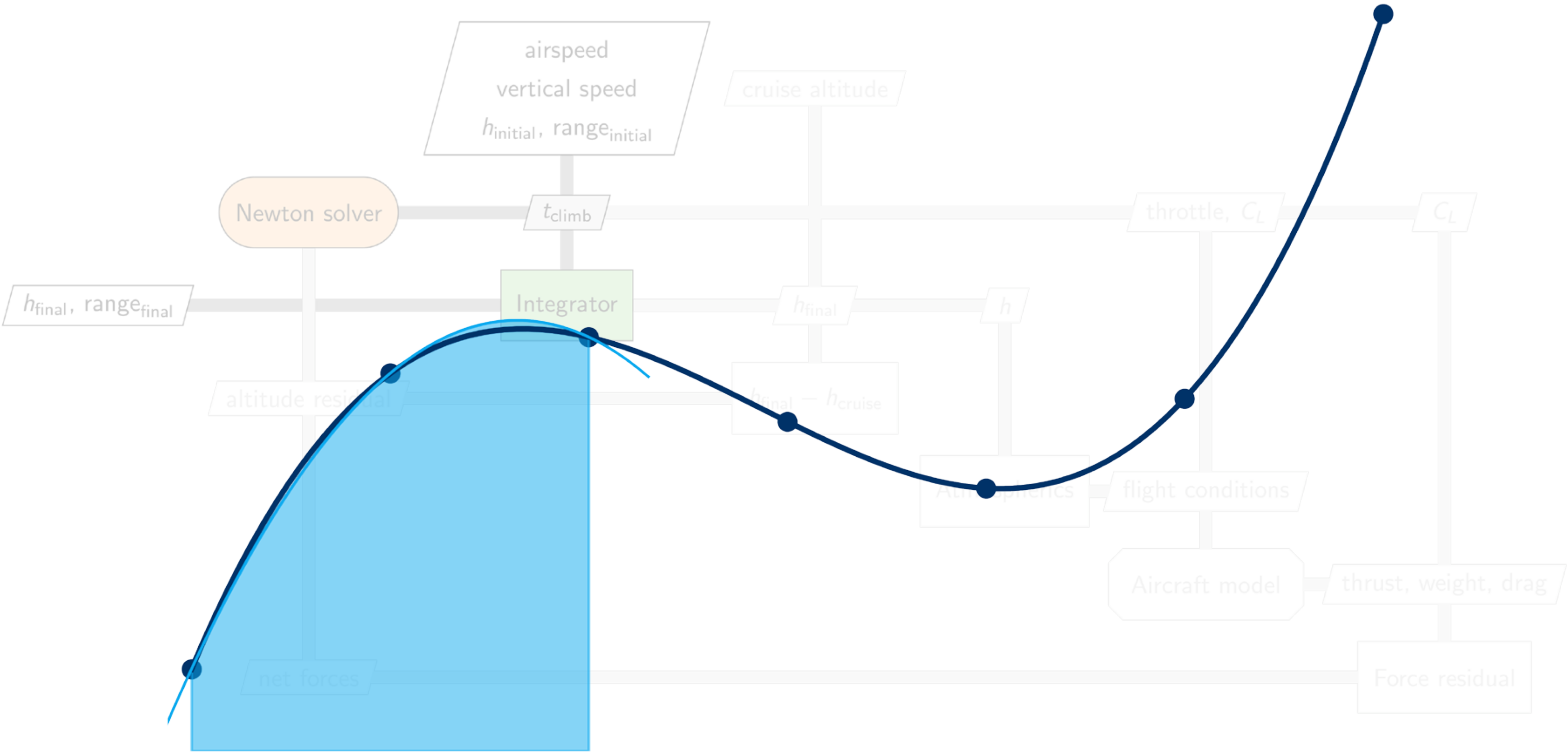
Integrator uses Simpson's rule



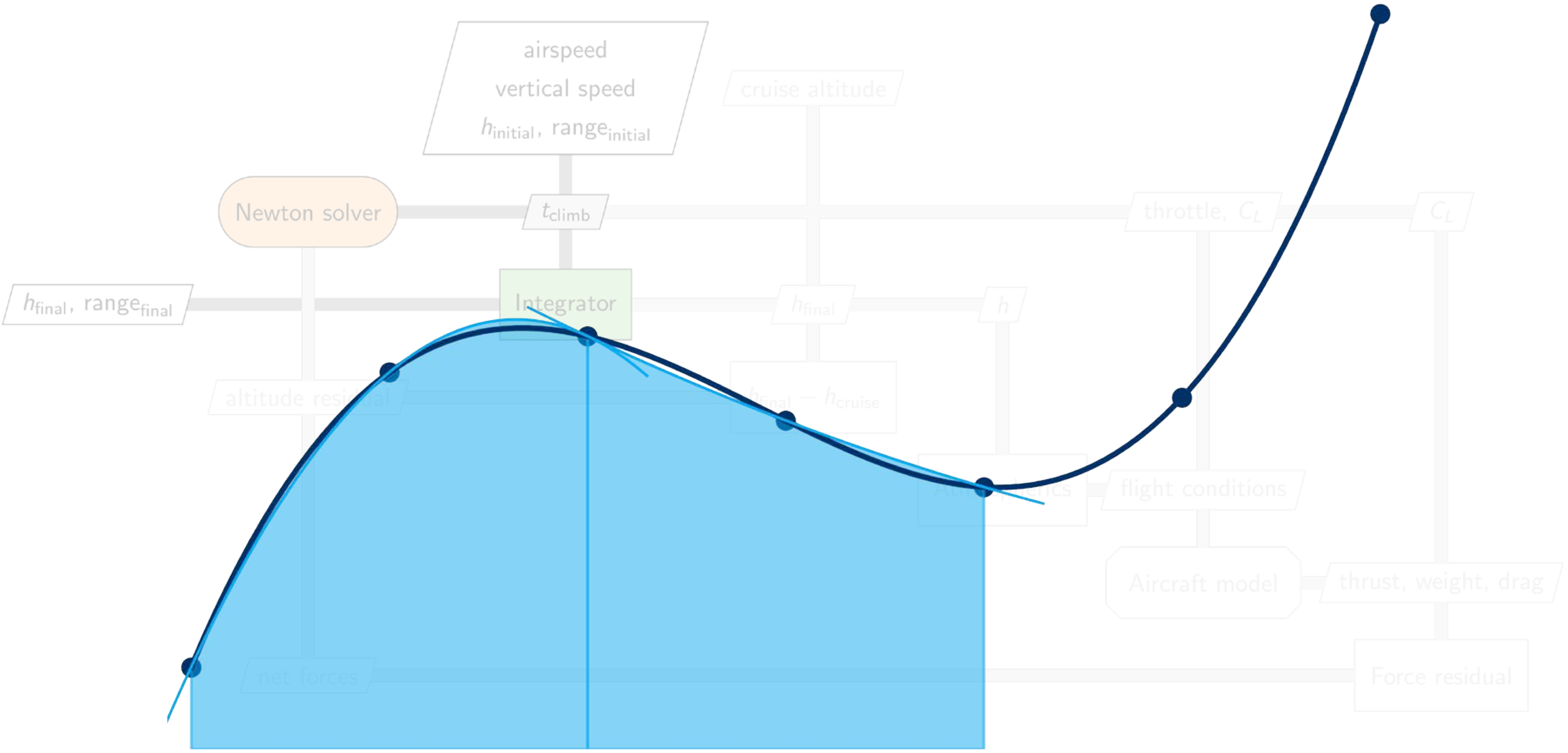
Integrator uses Simpson's rule



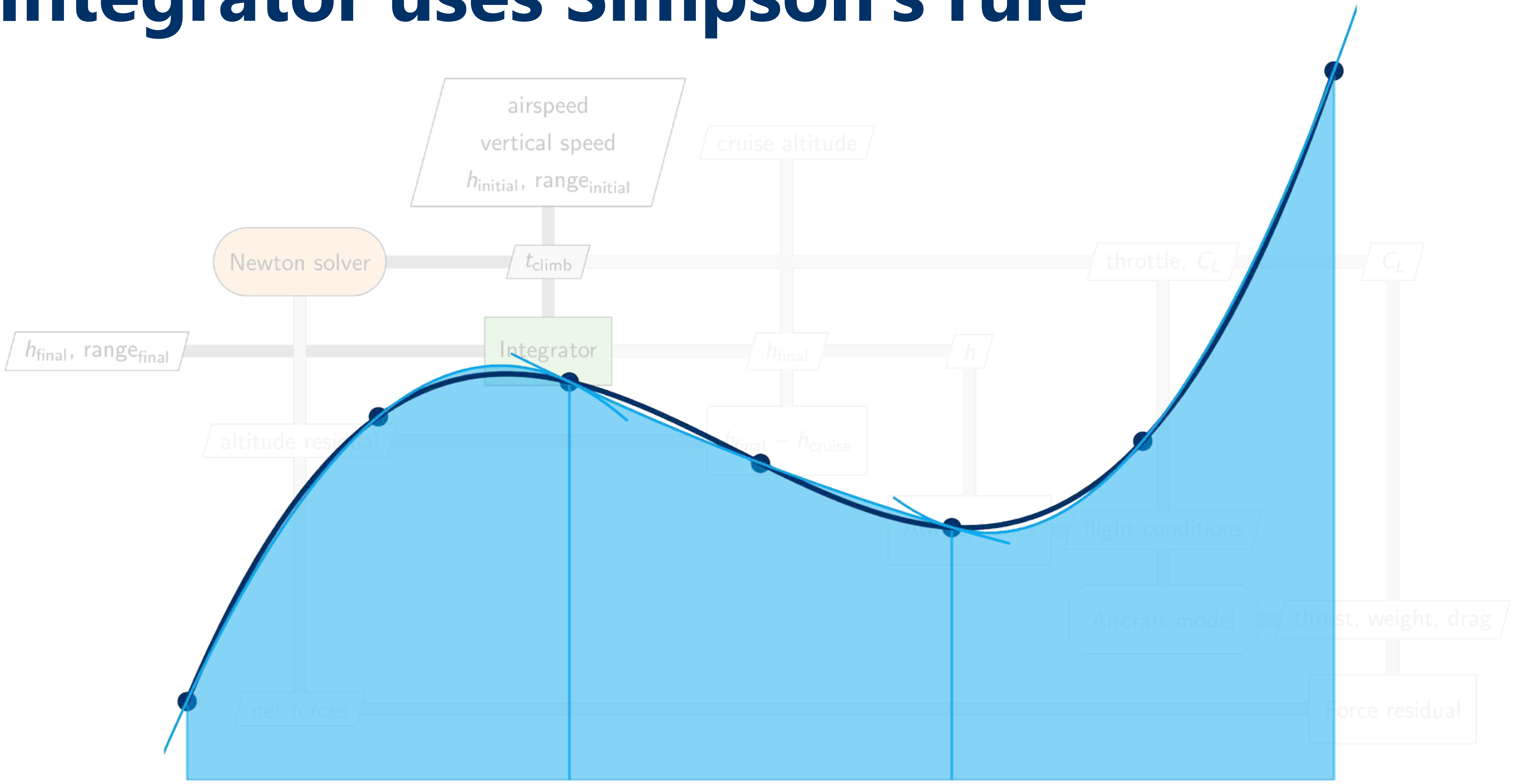
Integrator uses Simpson's rule



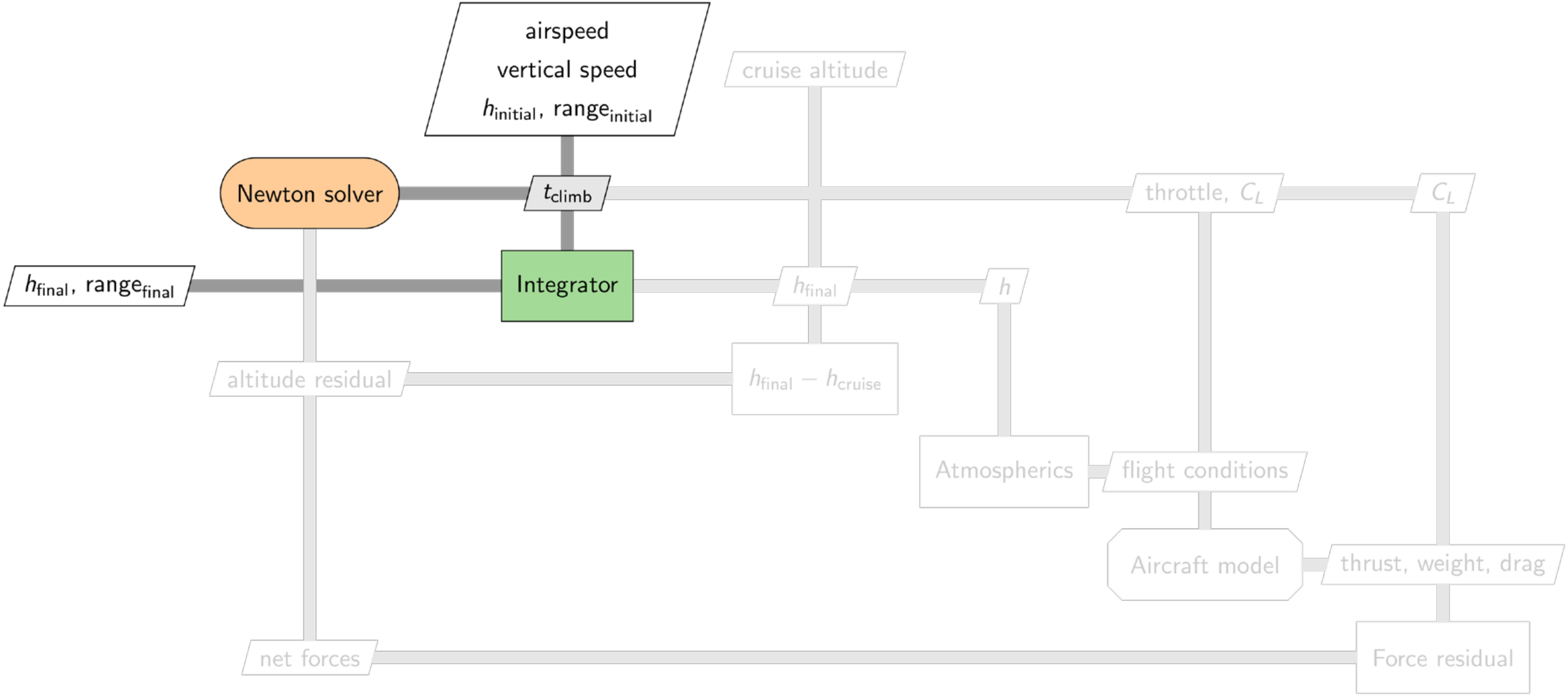
Integrator uses Simpson's rule



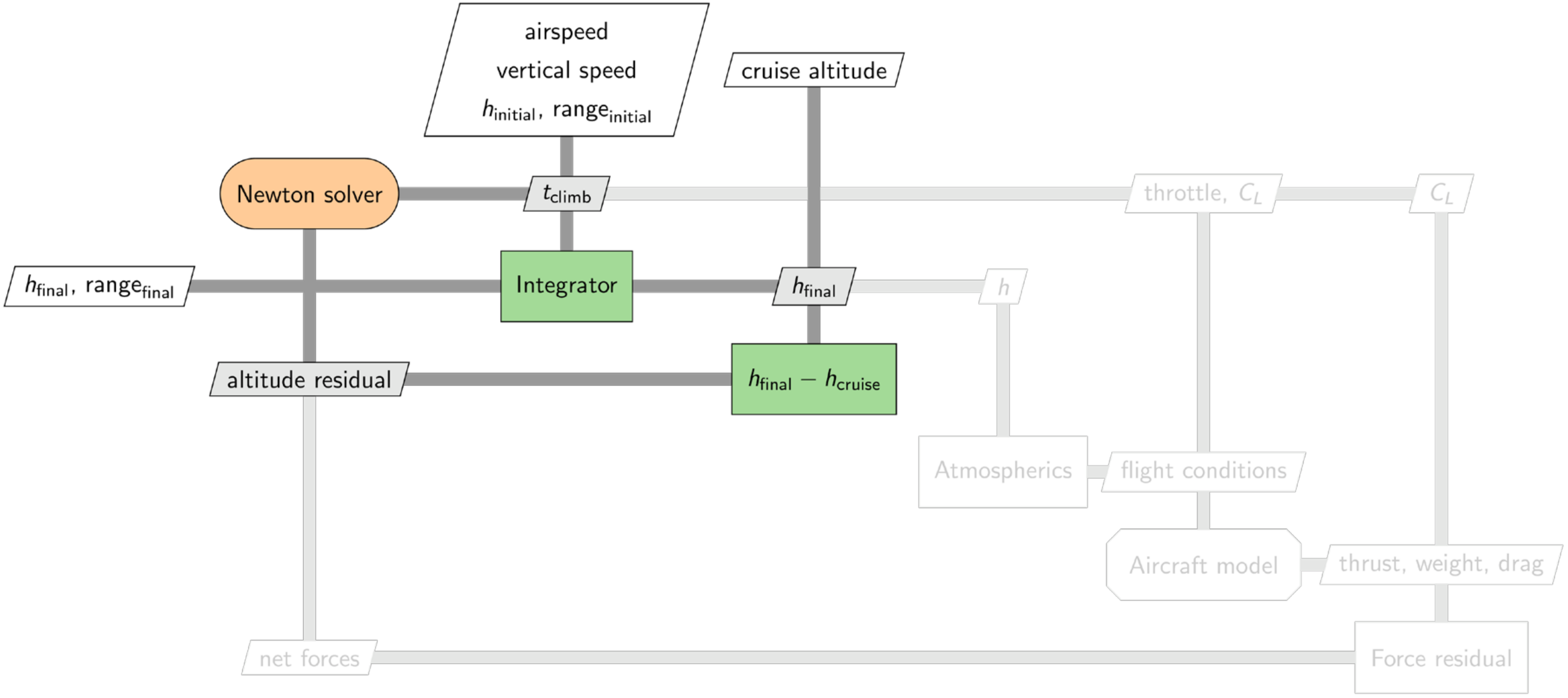
Integrator uses Simpson's rule



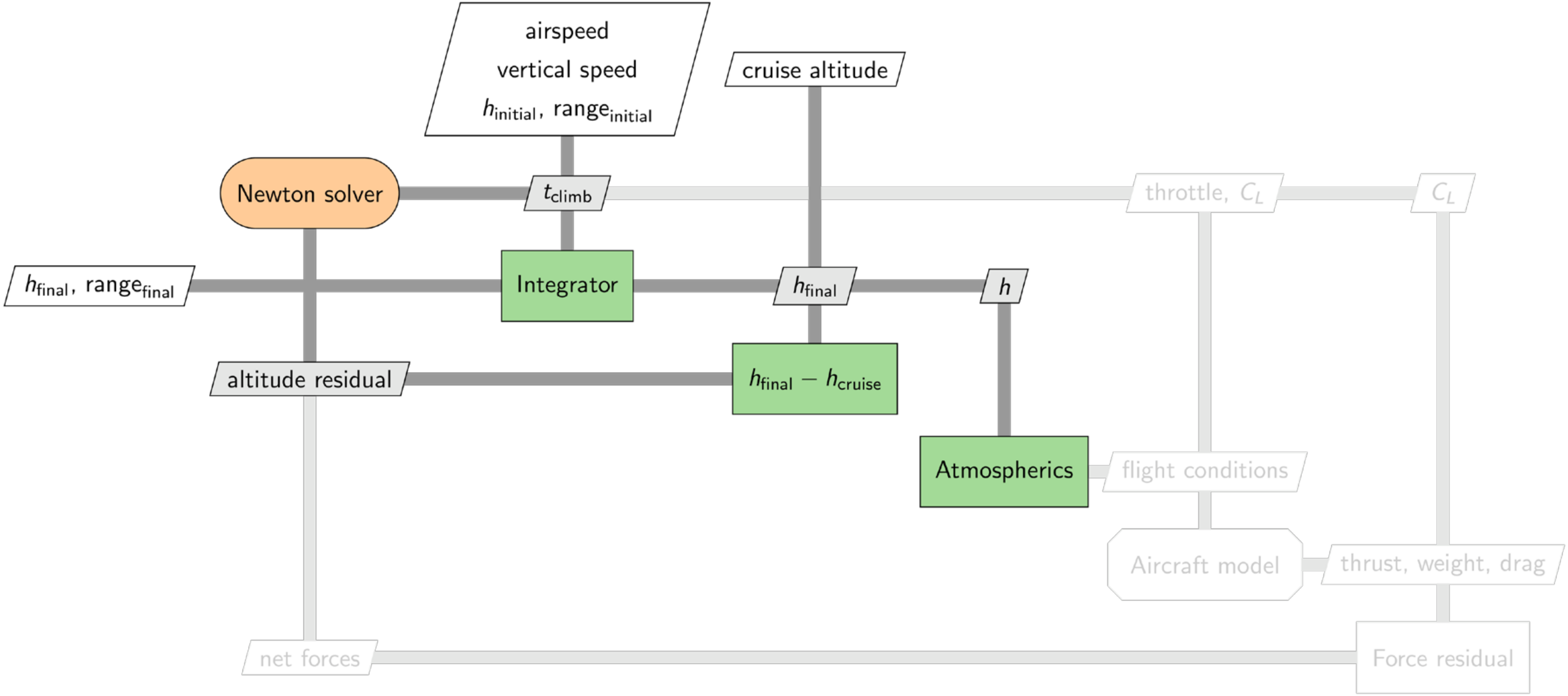
Integrator uses Simpson’s rule



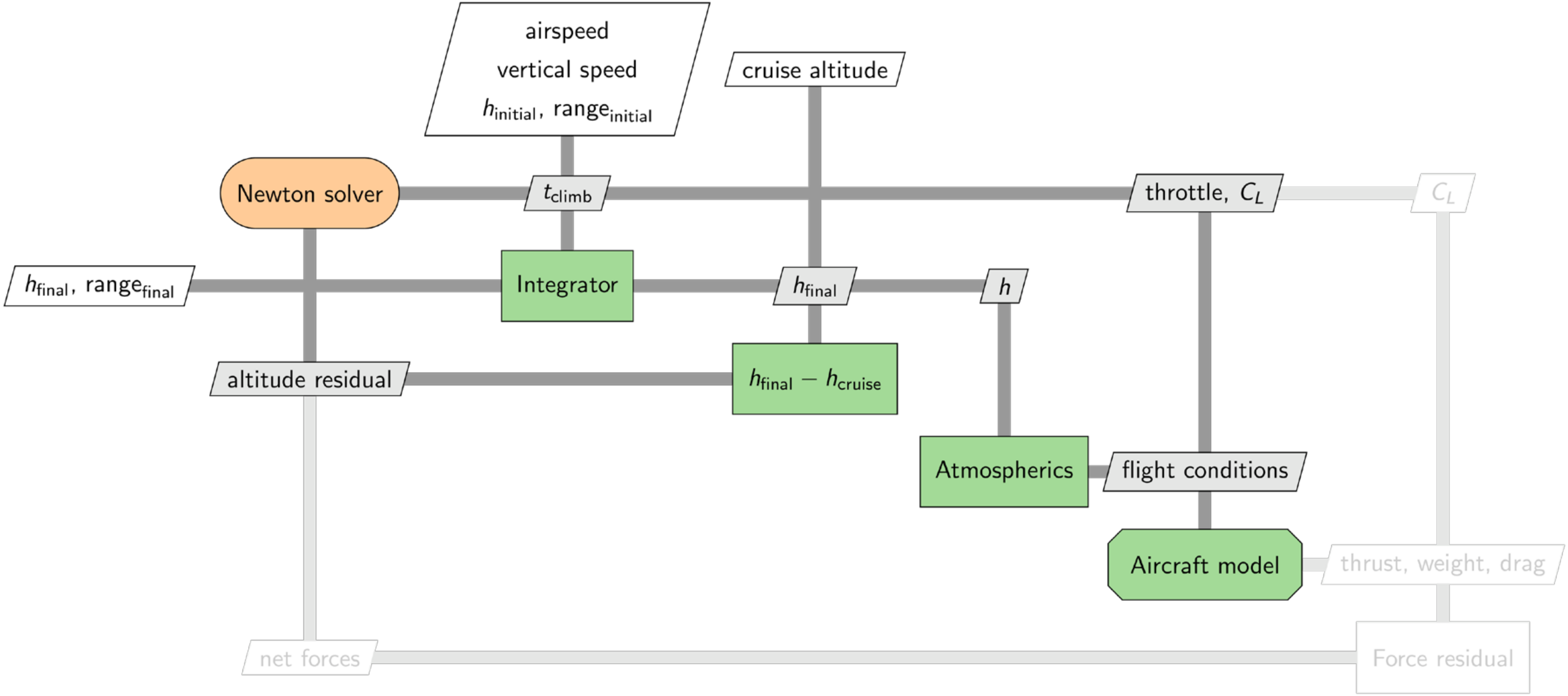
Set phase duration so it ends at cruise altitude



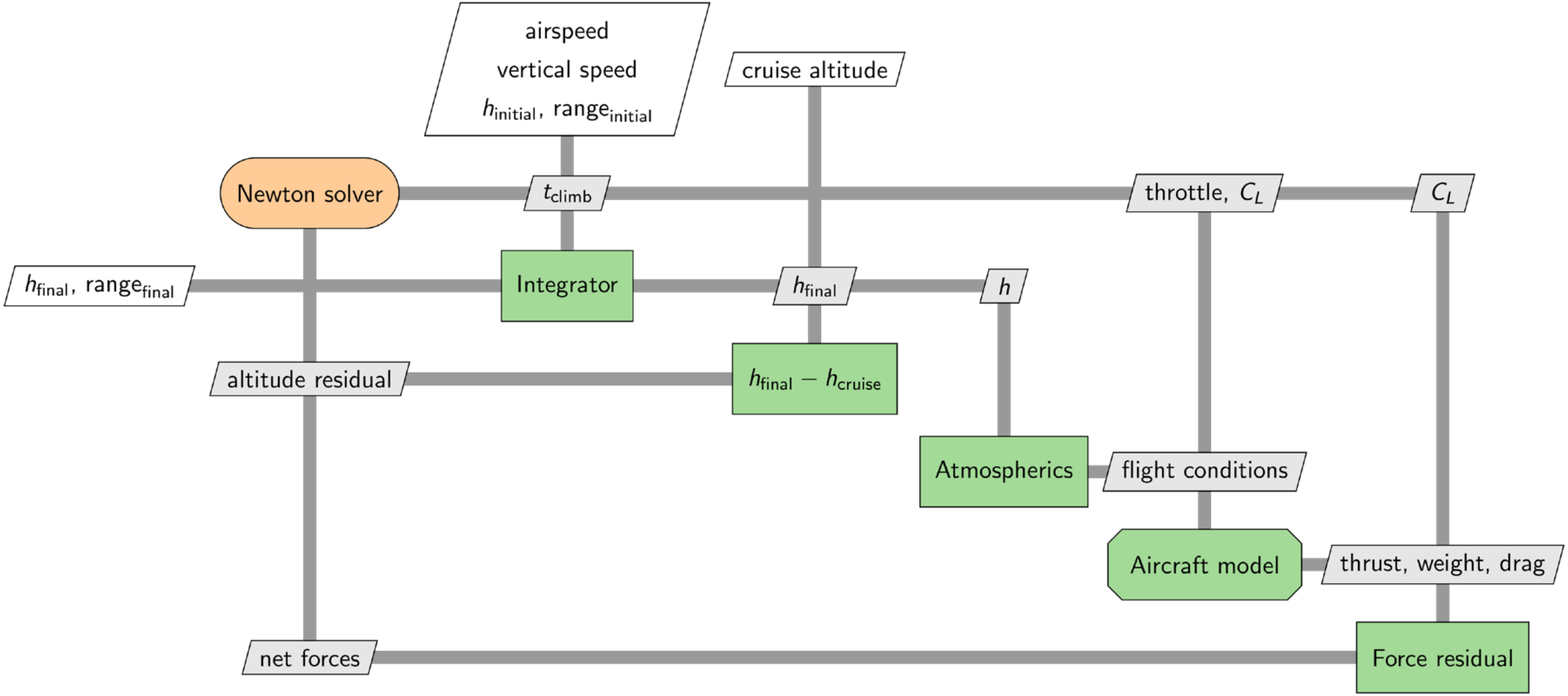
Atmospherics computes flight conditions



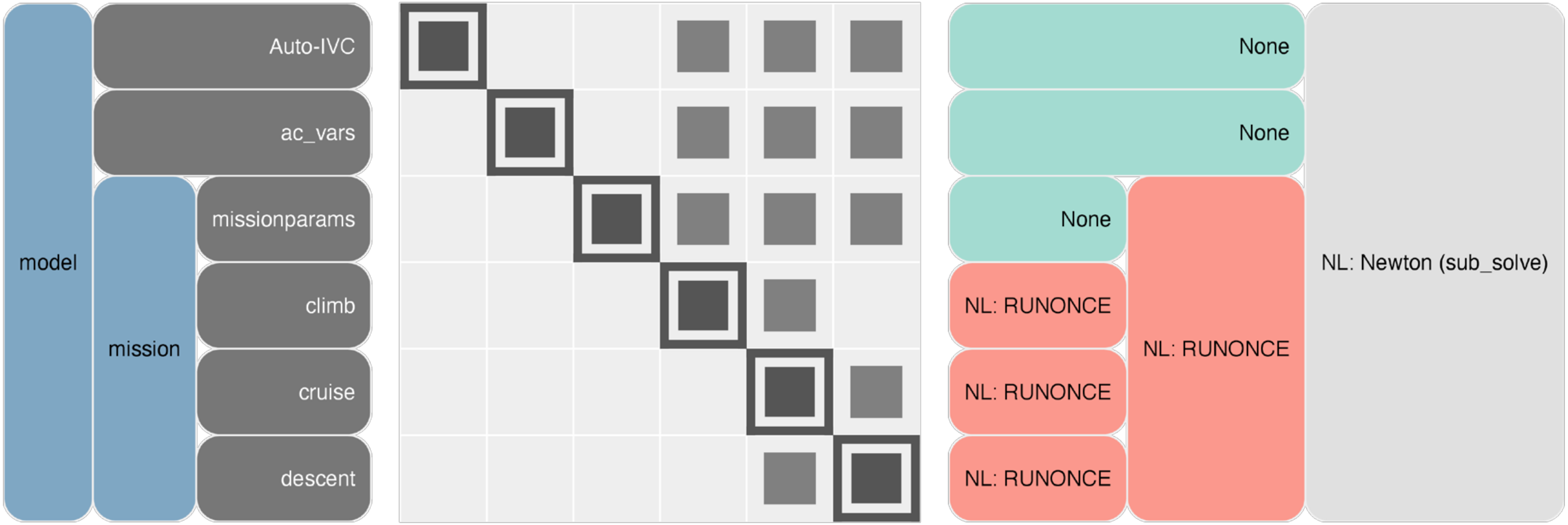
Aircraft model computes forces



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

$$\frac{dT}{dt} = \frac{q_{in} - q_{out}}{mc}$$

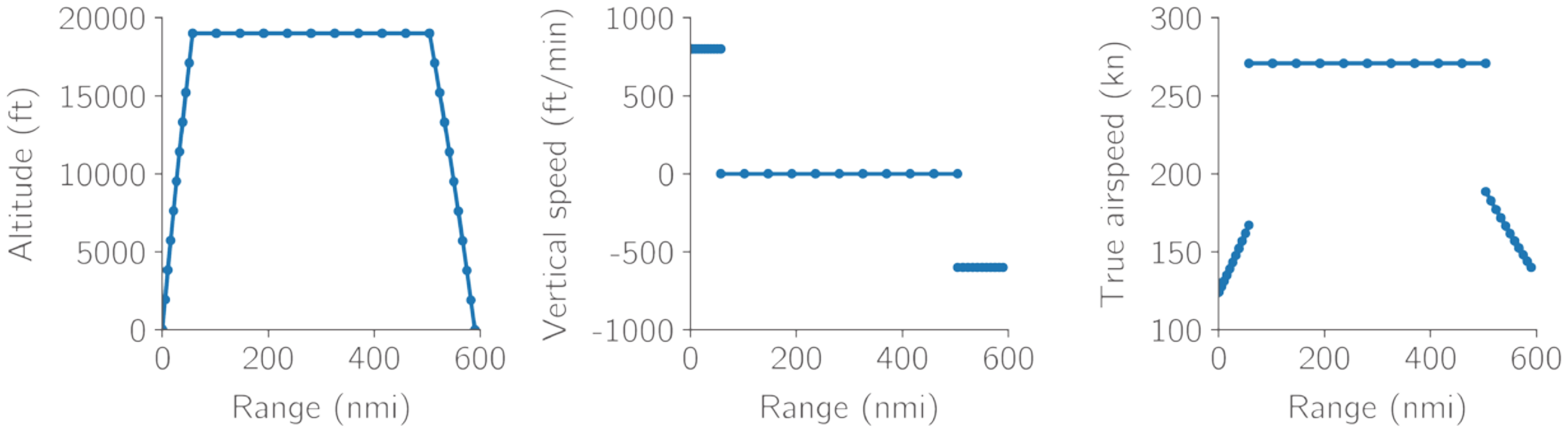
- Brelje magic™ automatically finds states within aircraft model to integrate and links them across mission phases

Code flexibility

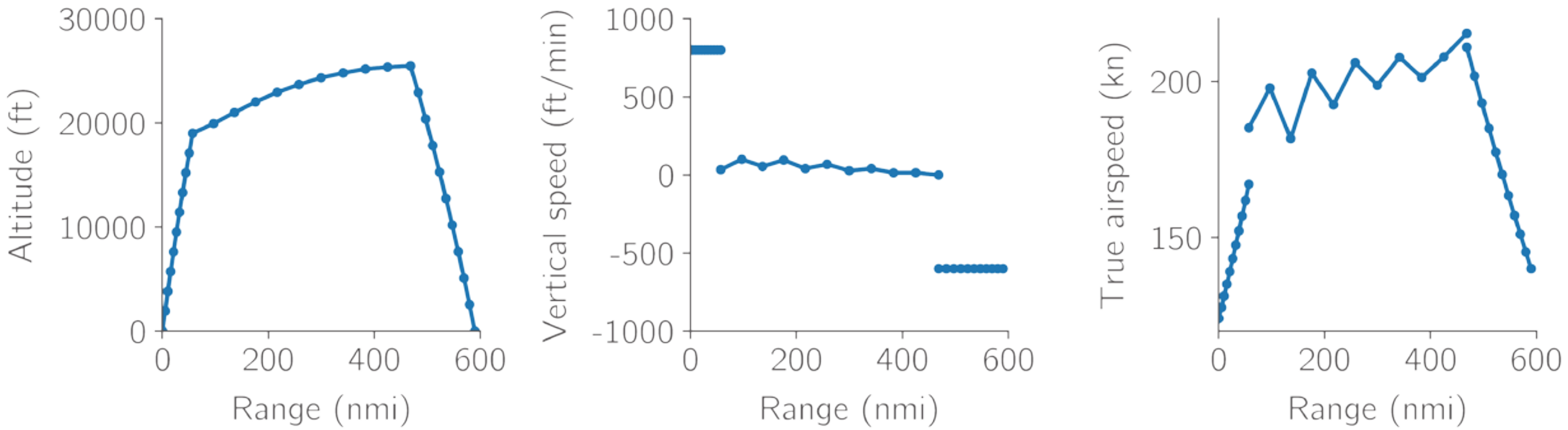
Mission analysis

Lessons learned

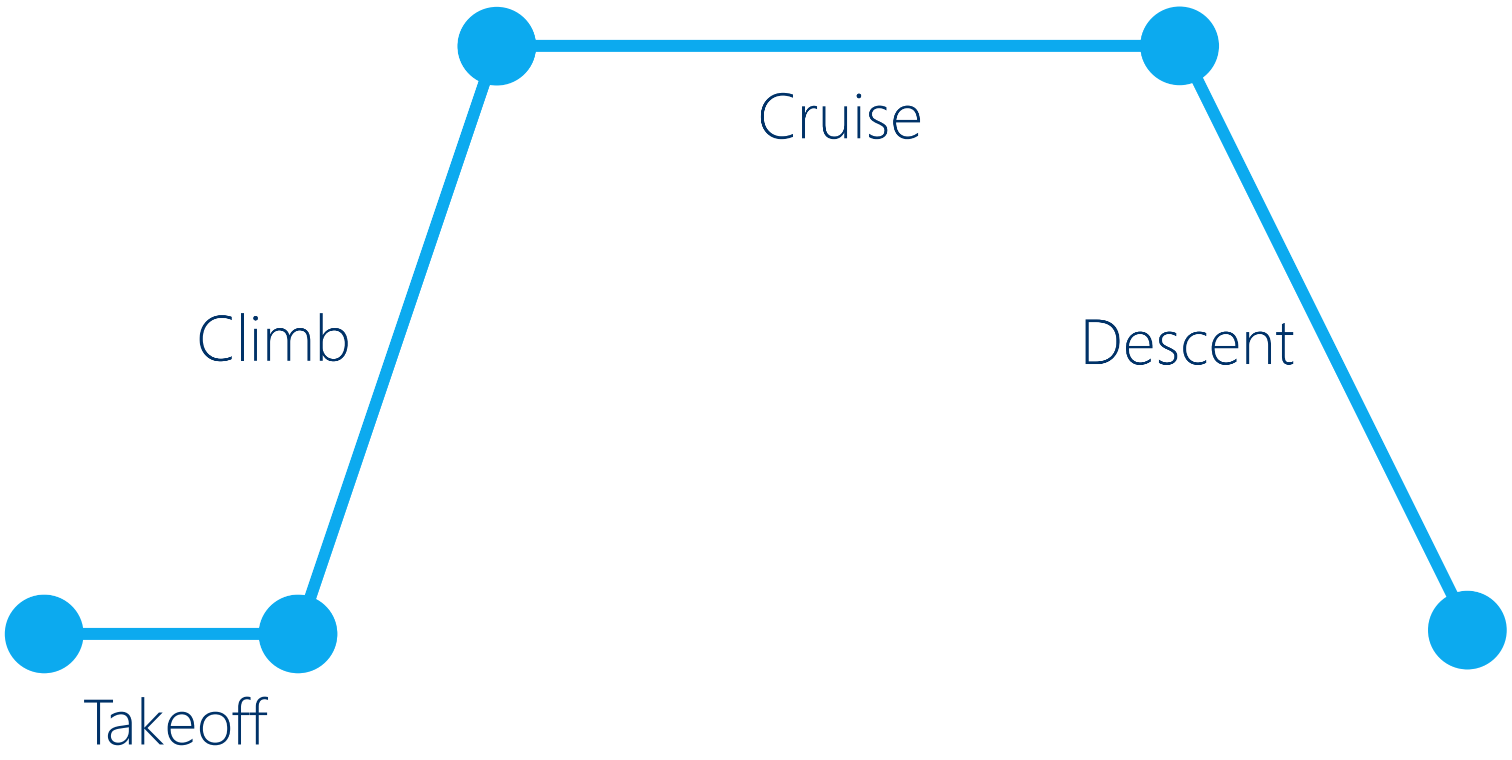
Not designed for trajectory optimization



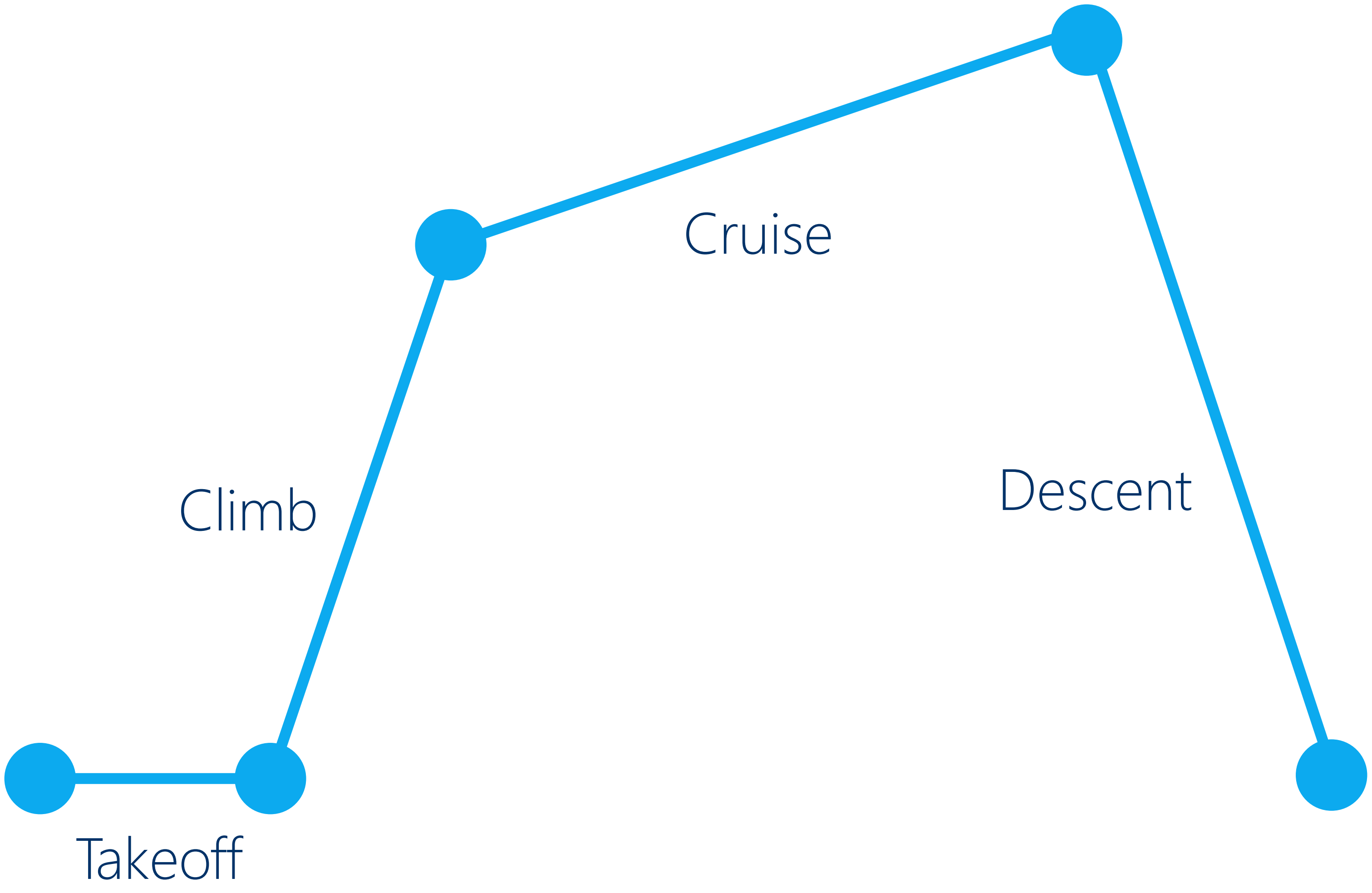
Optimize cruise airspeed and vertical speed



We linearly interpolate within phases



We linearly interpolate within phases

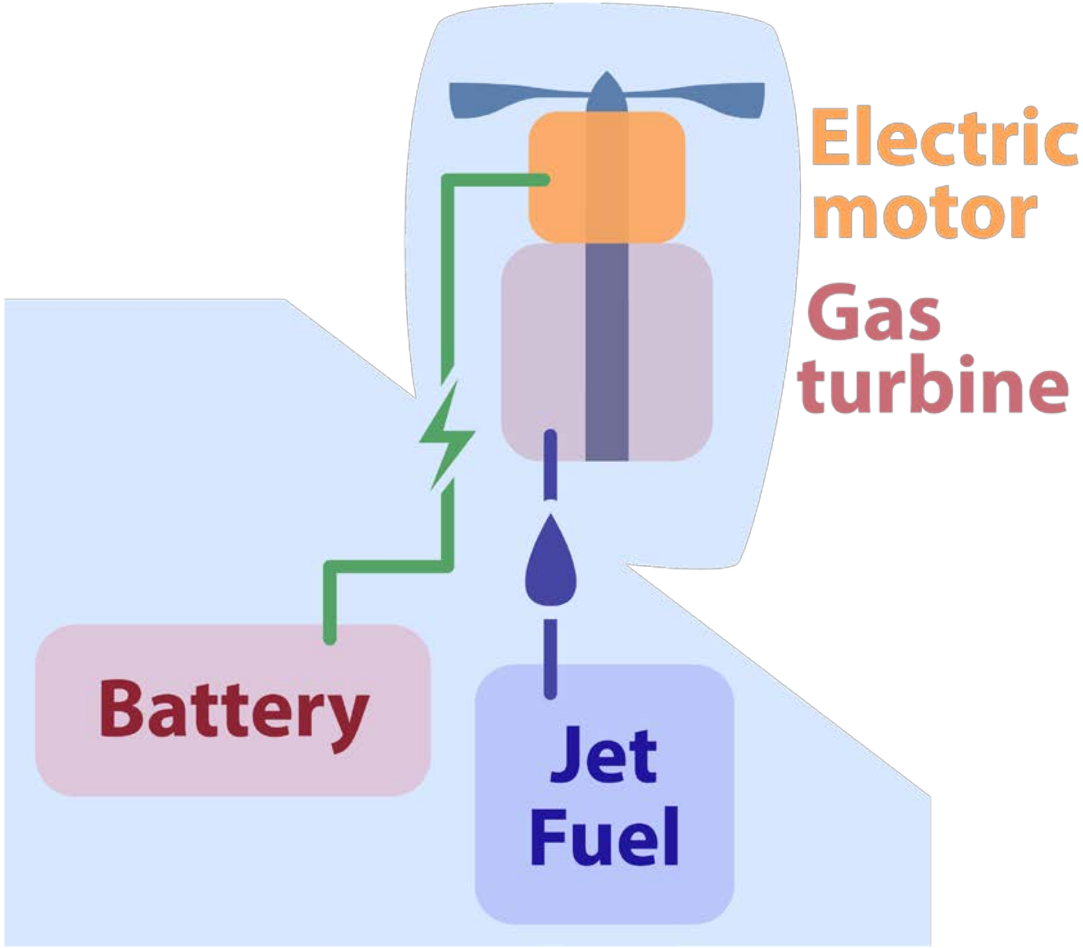


Vectorization for efficiency

```python
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?

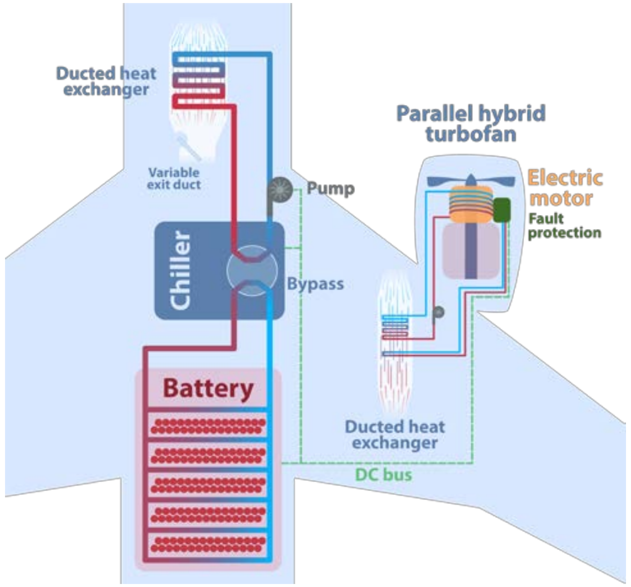
- OpenConcept's approach
  - Fast
  - Robust-ish
  - Physics valid once Newton solver converges (no optimization required)
- Trajectory optimization-style approach
  - Not as robust
  - More general representation of mission profile
  - Better for mission profile optimization

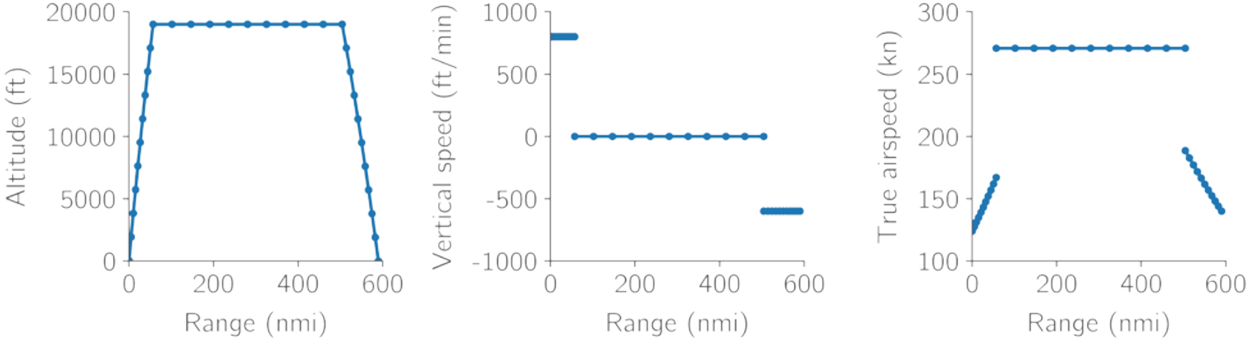
We developed a tool for efficient conceptual aircraft design with OpenMDAO

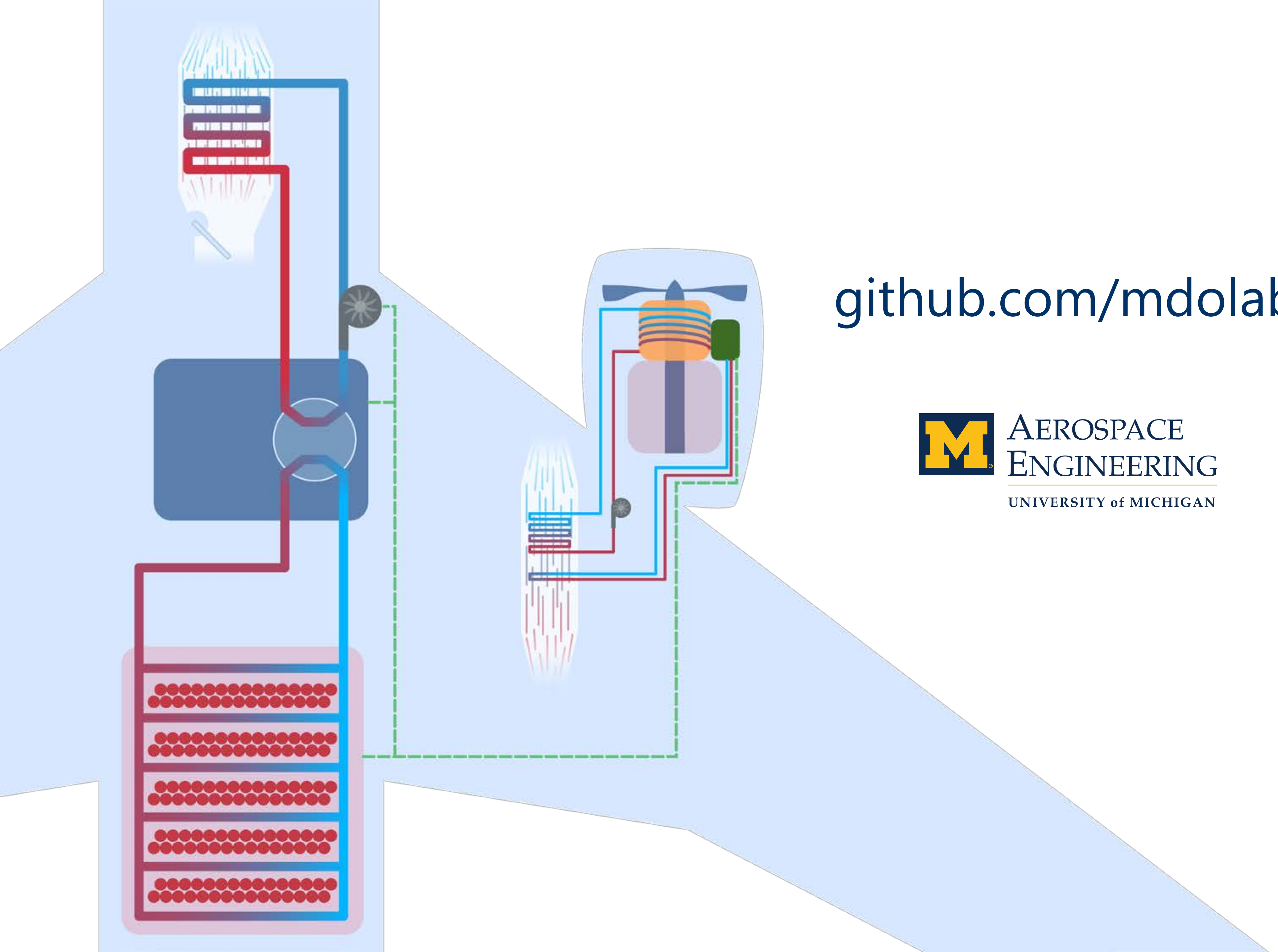
Rapidly optimize aircraft architectures

Steady flight mission phases with predefined profile

Hybrid mission analysis approach for the best of both?







github.com/mdolab/openconcept

AEROSPACE ENGINEERING
UNIVERSITY of MICHIGAN

MDO Lab