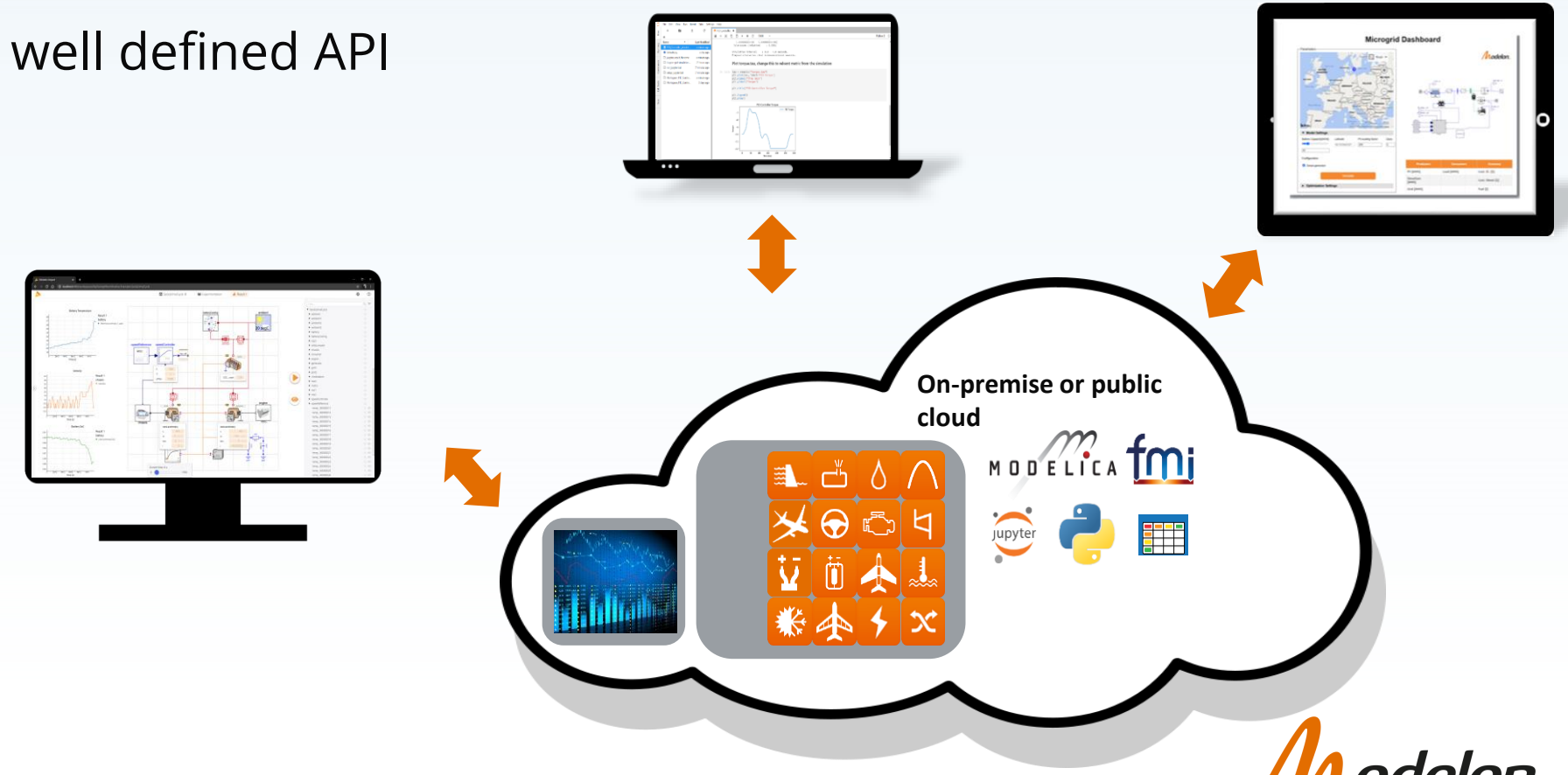
Modelon Impact
Open Innovation

Presented by Modelon

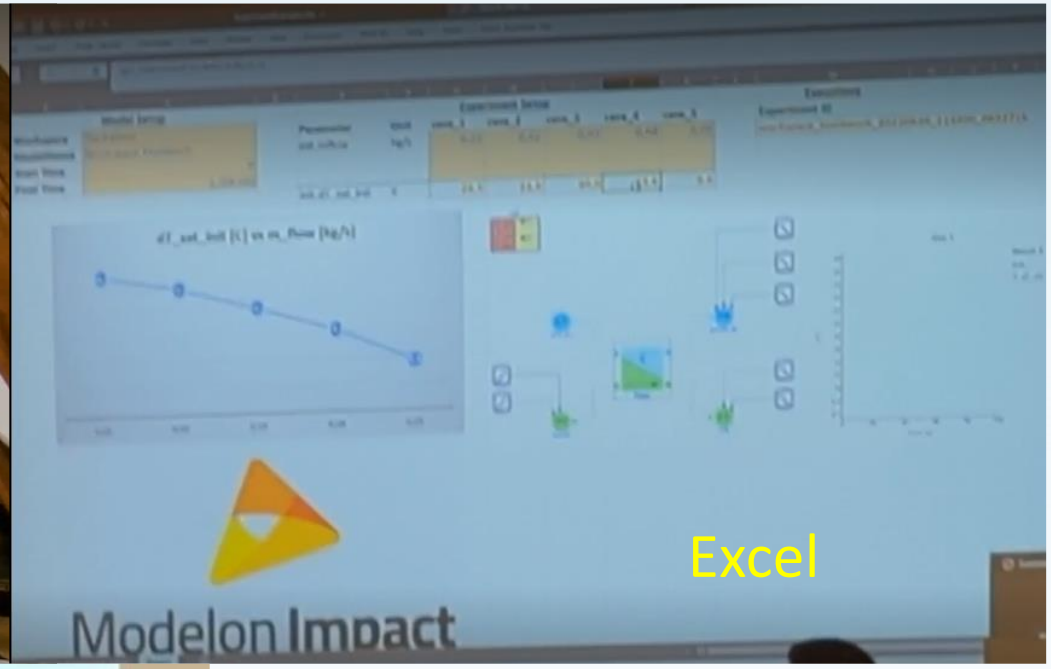
Open Architecture

- Openness through well defined API



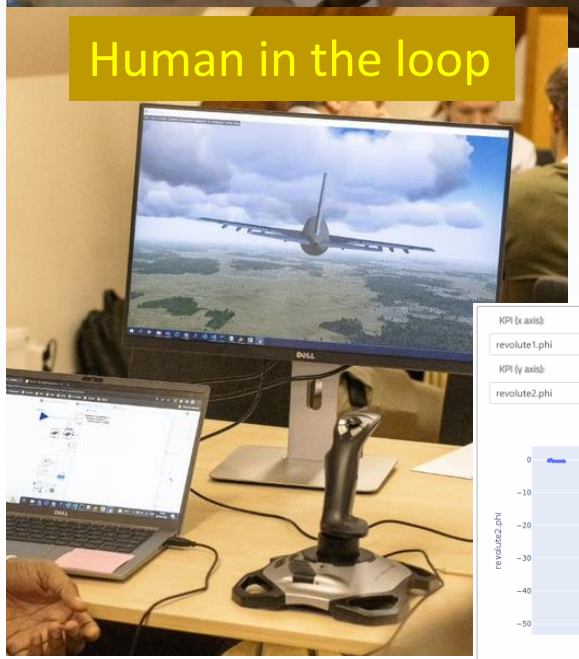
Growth Hackathon

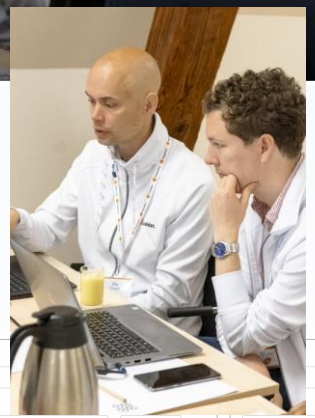
Node RED



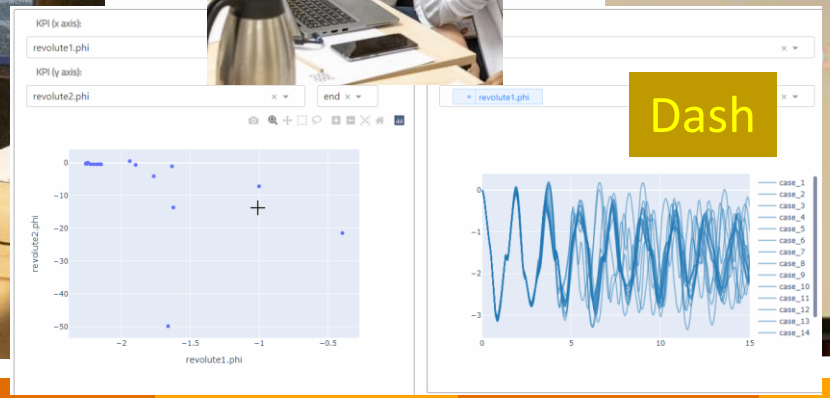
Excel

Human in the loop





Oculus Quest



Dash

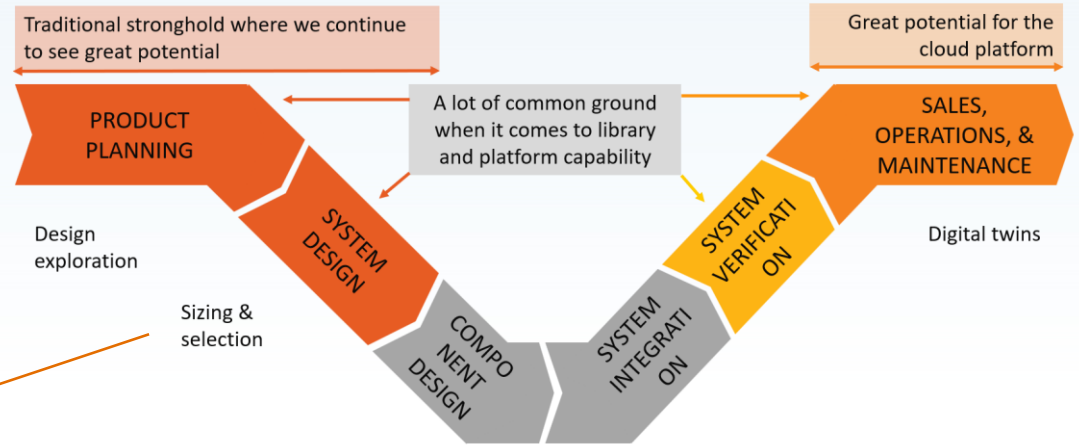
Winners

Directions

- Impact today:
  - Modeling in browser
  - Simulation in browser
  - JupyterLab
  - Custom functions
  - Custom web apps

- Declarative experiment workflow definitions
- Massive parallel simulations
- Post processing, visualization and decision support

How do we prioritize and focus?



Demos

- Modelon Impact Excel Add-In
- Dashboarding with Dash
- Digital Twin with NodeRED

Modelon Impact Excel Add-In

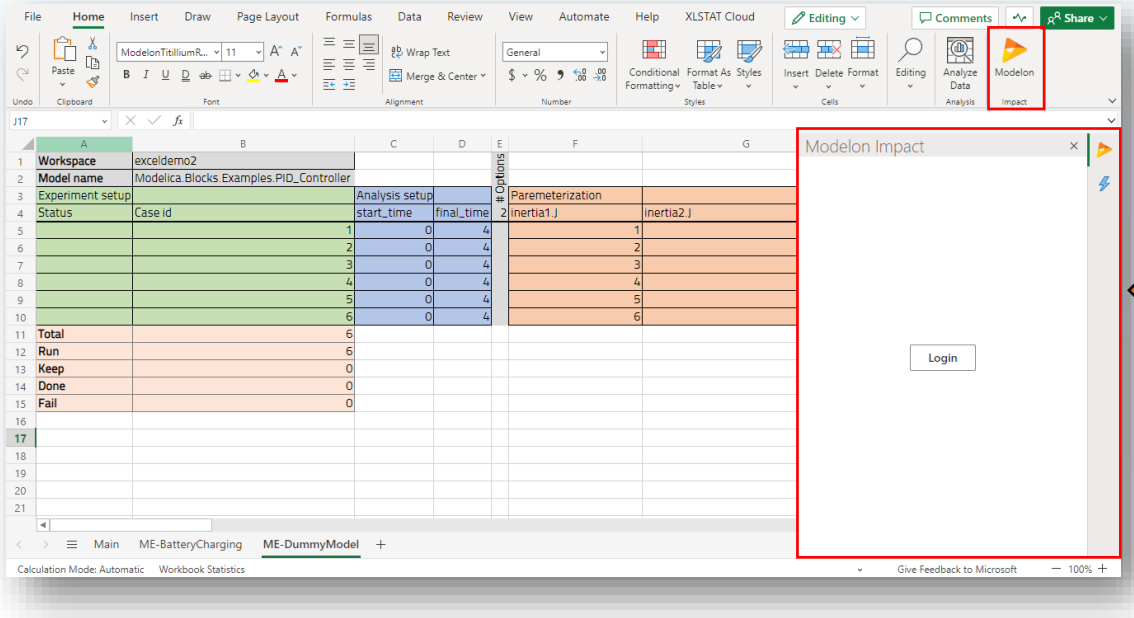
- Engineers (and Others) in Most Companies work daily in Excel
- Data from MI Simulations will likely at one point end up in an Excel Spreadsheet
- Integrate the usage of both applications!

Building Blocks

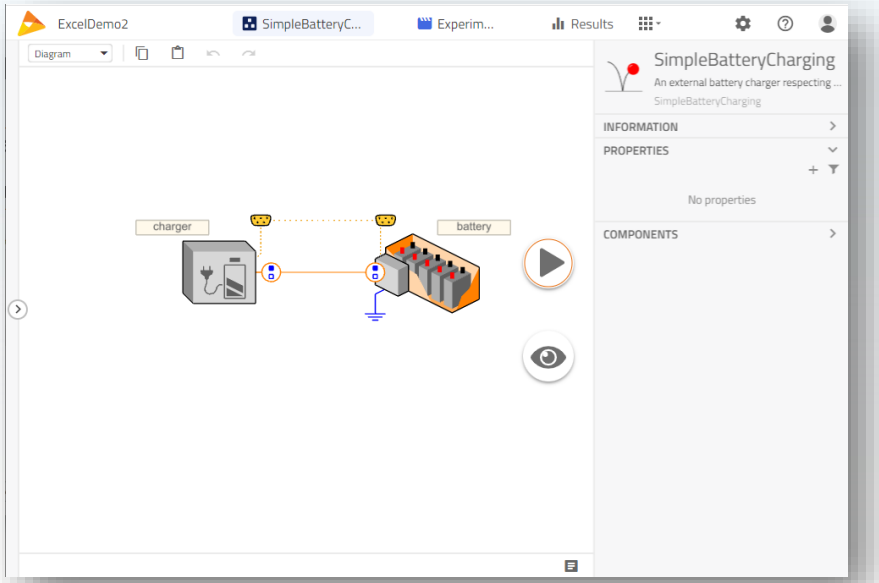
Office JavaScript API library

Modelon Impact Client for JavaScript

Modelon Impact API



Excel



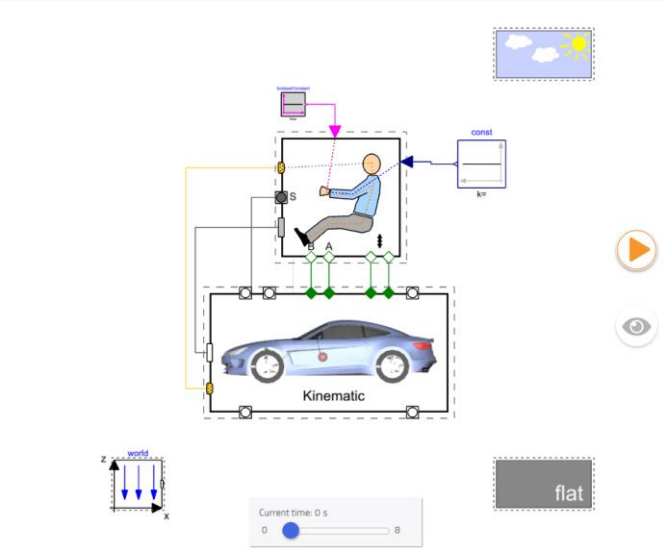
Modelon Impact

Demo

Modelon Impact Excel Add-In

Dashboards with Dash

Modelon Impact



https://impact.modelon.cloud

jupyter python

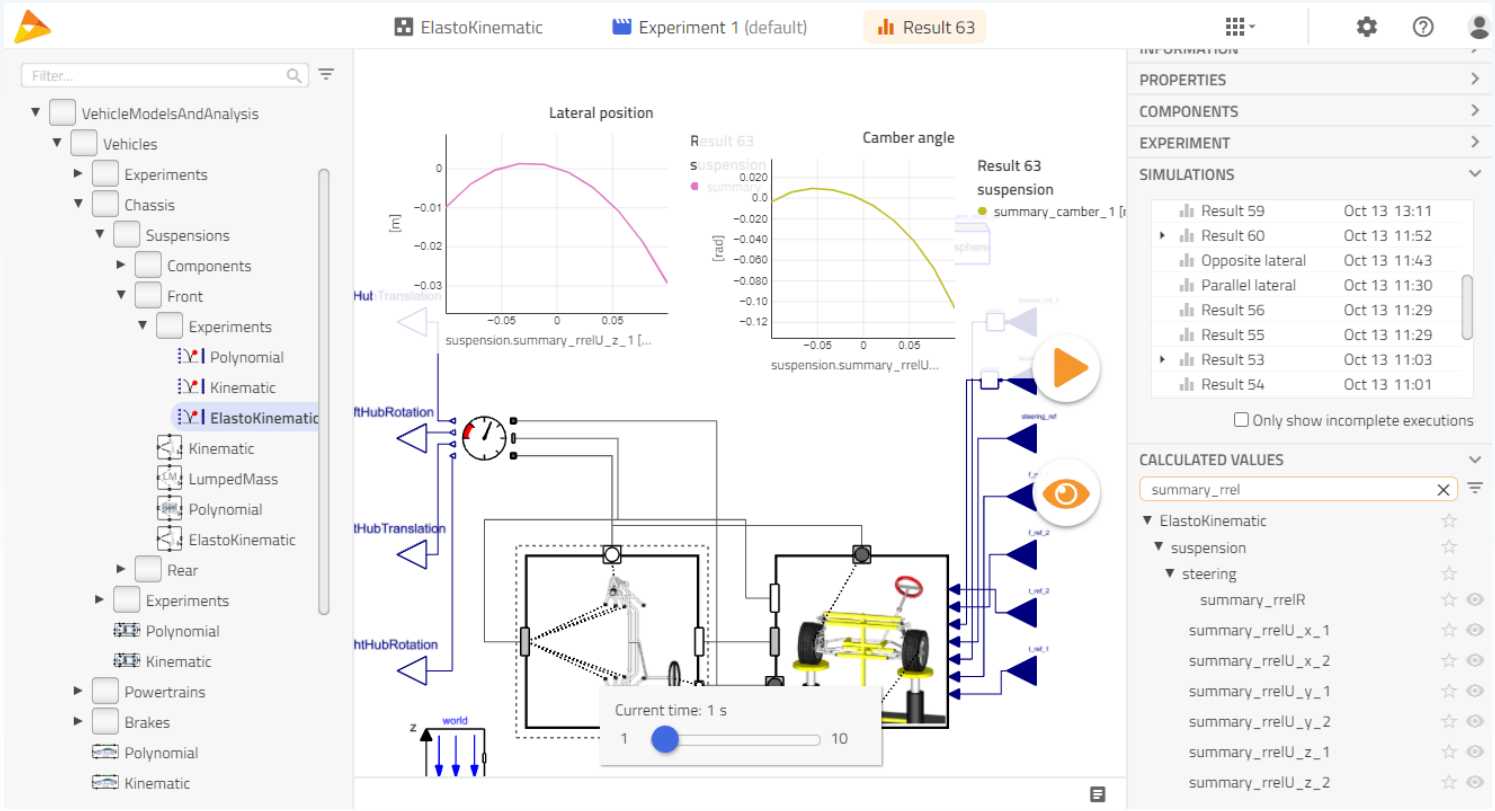
https://modelon-impact-client.readthedocs.io

plotly | Dash

https://dash.plotly.com

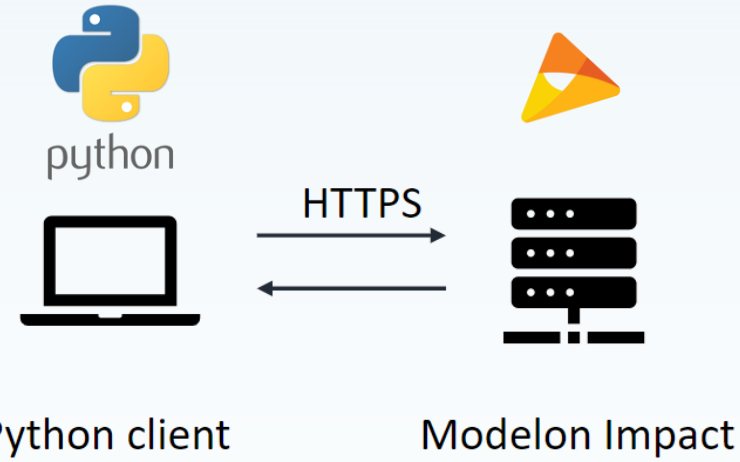
Modelon

Build and prepare models in Impact



- Put together your models
- Set up parameterizations
- Set up measurements and KPI outputs

Impact Python Client



Impact Python Client handles communication with the server, including authentication

Simple example of simulating a model and plotting results:

```
from modelon.impact.client import Client
client = Client(url=<impact-domain>)

workspace = client.create_workspace(<workspace name>)
model = workspace.get_model("Modelica.Blocks.Examples.PID_Controller")
dynamic = workspace.get_custom_function('dynamic')

experiment_definition = model.new_experiment_definition(dynamic)

experiment = workspace.execute(experiment_definition).wait()

import plotly.graph_objects as go

fig = go.Figure()

for case in experiment.get_cases():
   if case.is_successful():
      result = case.get_trajectories()
      fig.add_trace(go.Scatter(x=result['time'], y=result['inertia1.phi']))

fig.show()
```

Python based framework for building web apps

Access to the full range of plot options from Plotly

Low threshold to building interactive visualizations

https://dash.plotly.com

Simple app example with a figure:

```
from dash import Dash, html, dcc
import plotly.express as px
import pandas as pd

app = Dash(__name__) # assume you have a "long-form" data frame
# see https://plotly.com/python/px-arguments/ for more options

df = pd.DataFrame({ "Fruit": ["Apples", "Oranges", "Bananas",
"Apples", "Oranges", "Bananas"], "Amount": [4, 1, 2, 2, 4, 5],
"City": ["SF", "SF", "SF", "Montreal", "Montreal", "Montreal"]
})

fig = px.bar(df, x="Fruit", y="Amount", color="City",
barmode="group")

app.layout = html.Div(children=[ html.H1(children='Hello Dash'),
html.Div(children=''' Dash: A web application framework for your
data. '''), dcc.Graph( id='example-graph', figure=fig ) ])

if __name__ == '__main__': app.run_server(debug=True)
```


Modelon

Example 1: Calibration dashboard

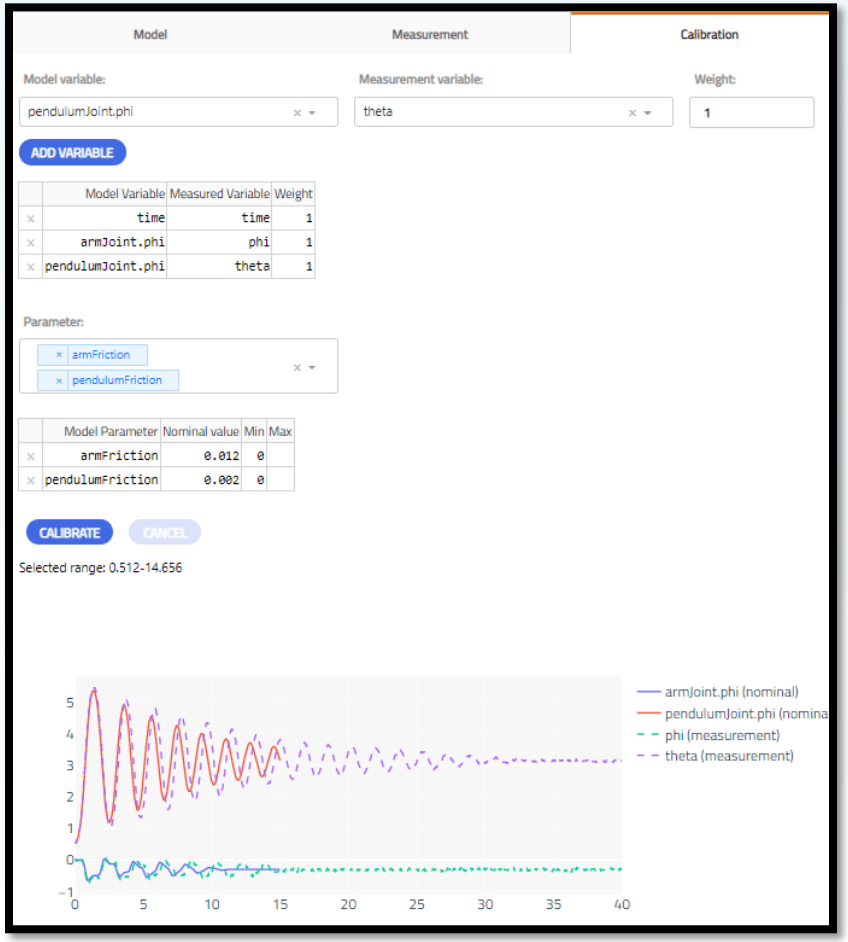
Select model, inspect results

Upload measurement data, inspect signals

Select variables to calibrate

Select parameters to use for calibration

Calibrate model to measurements

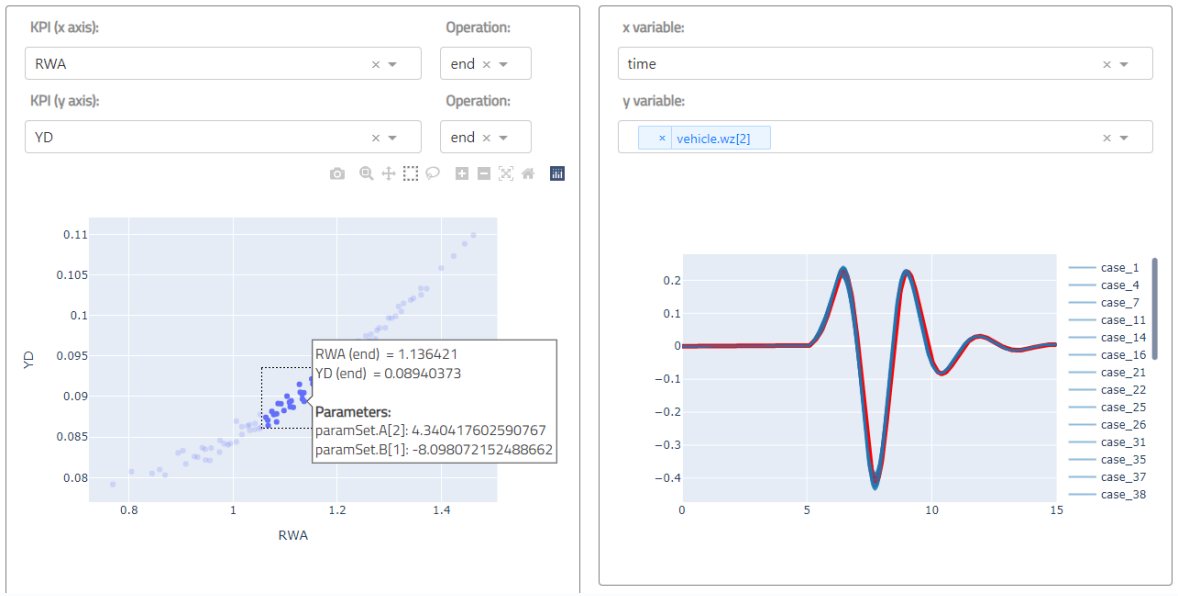


Example 2: Multi-execution dashboard

Select model, inspect results

Set up distributions for selected variables, decide number of cases

Visualize results, both scatter plot and time traces



Demo

Digital Twin in Modelon Impact

Digital Twin – From Design to Operation

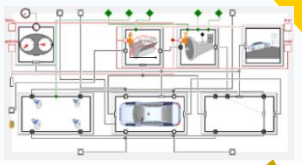
Engineering Design → *System Operation*

Virtual World

Requirements & Performance Targets

Finished Product

System Design



Platform Based Design

Implementation

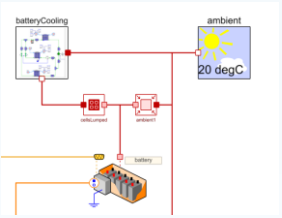
System Integration & **Validation**

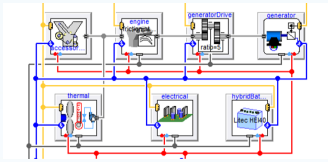
Module Design

Module Integration & **Verification**

Component Design

Component **Verification**





Battery life& SOH twin

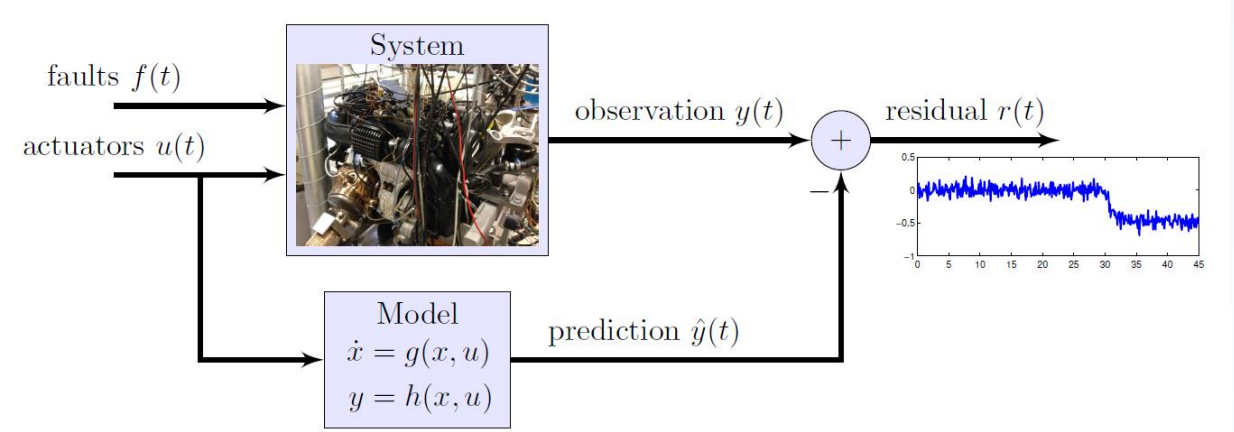
Powertrain fault detection twin

Real World

Modelon

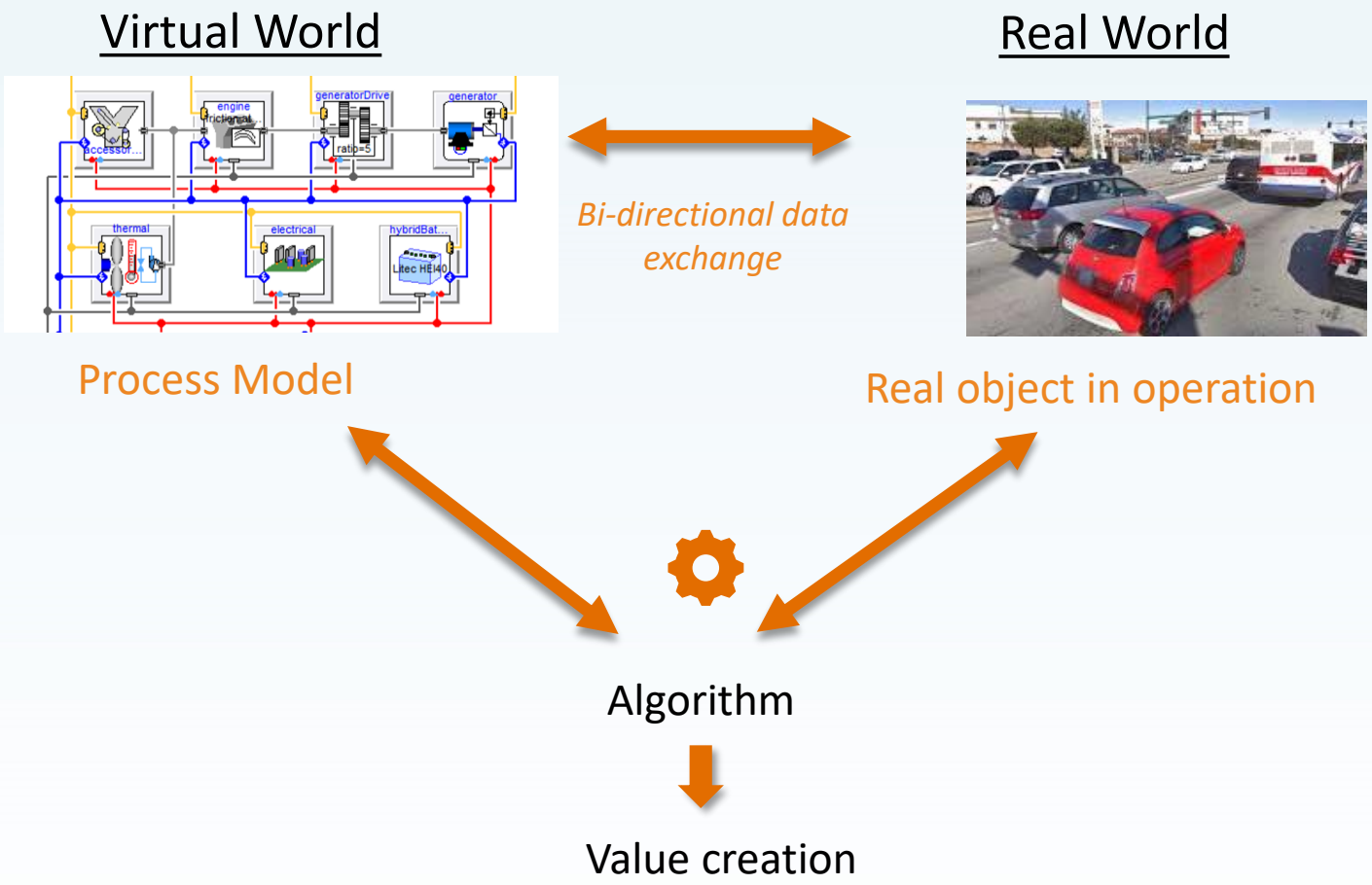
Examples of application

- Equipment health monitoring
- Predictive maintenance
- Fault detection & isolation
- Soft sensing
- Operating decision support
- Real-time optimal control



Platform requirements

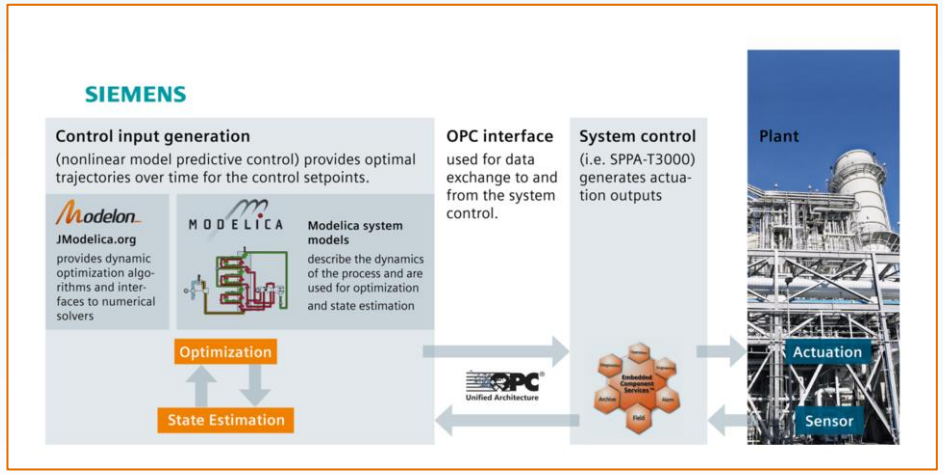
- A connected simulation platform for bi-directional data exchange
- Plant models & algorithms that can be executed at real-time
- Versatile modelling platform to describe various aspects of a process
  - Short and long-term
  - Different physical domains
  - Low or hi-fidelity depending on the computation



Previous digital twins based on Modelon components

Modelon’s back-end has already been used in Digital Twin context

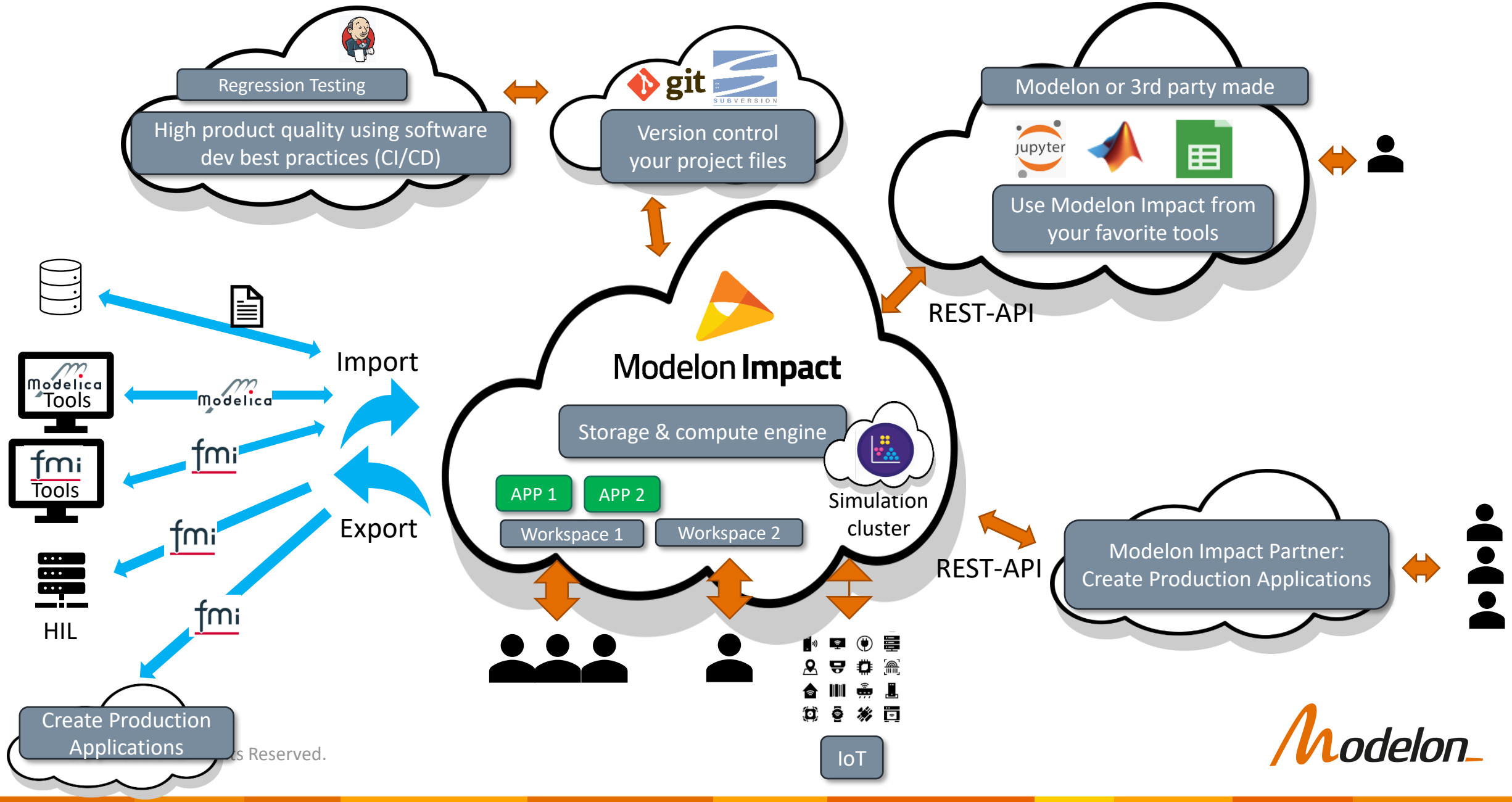
Plant FMU imported into data management system



- Optimization routines to solve for state estimation and optimal control
- OPC communication with control system

Modelon

MODELON IMPACT ECOSYSTEM

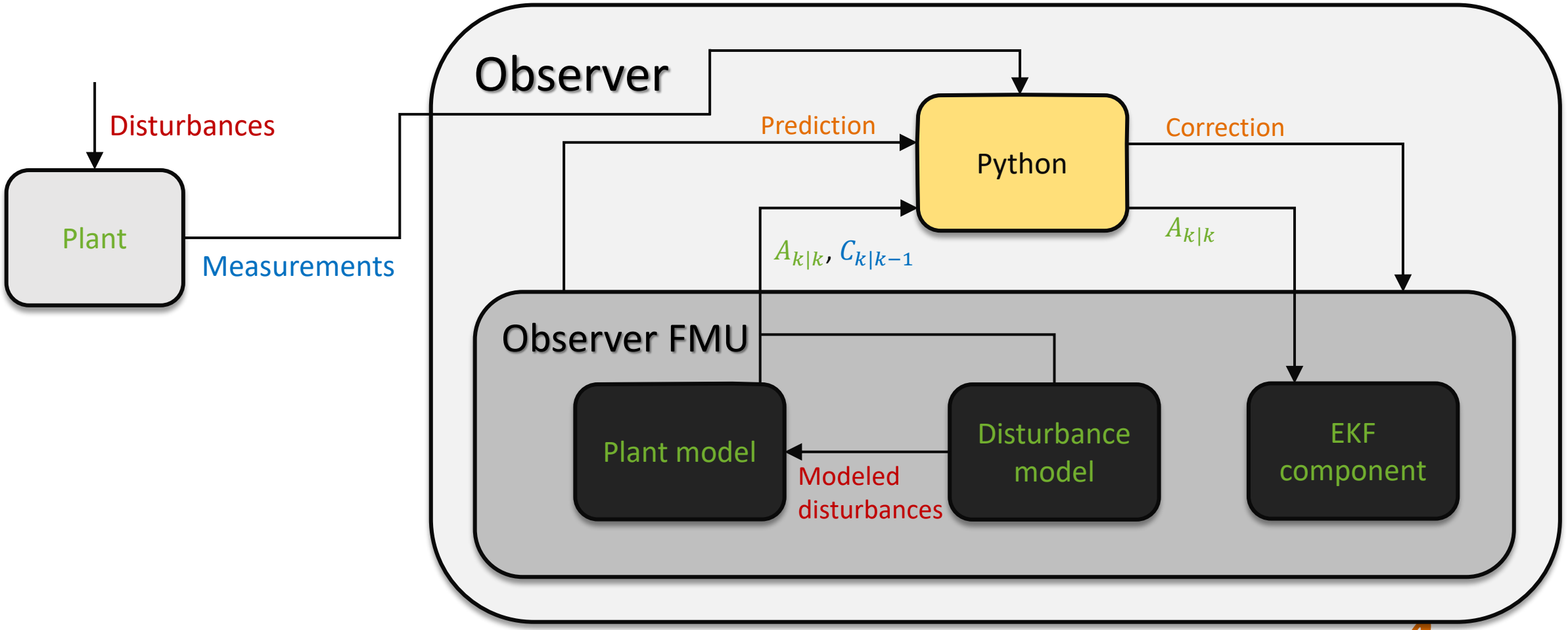


Digital Twin – proof of concept

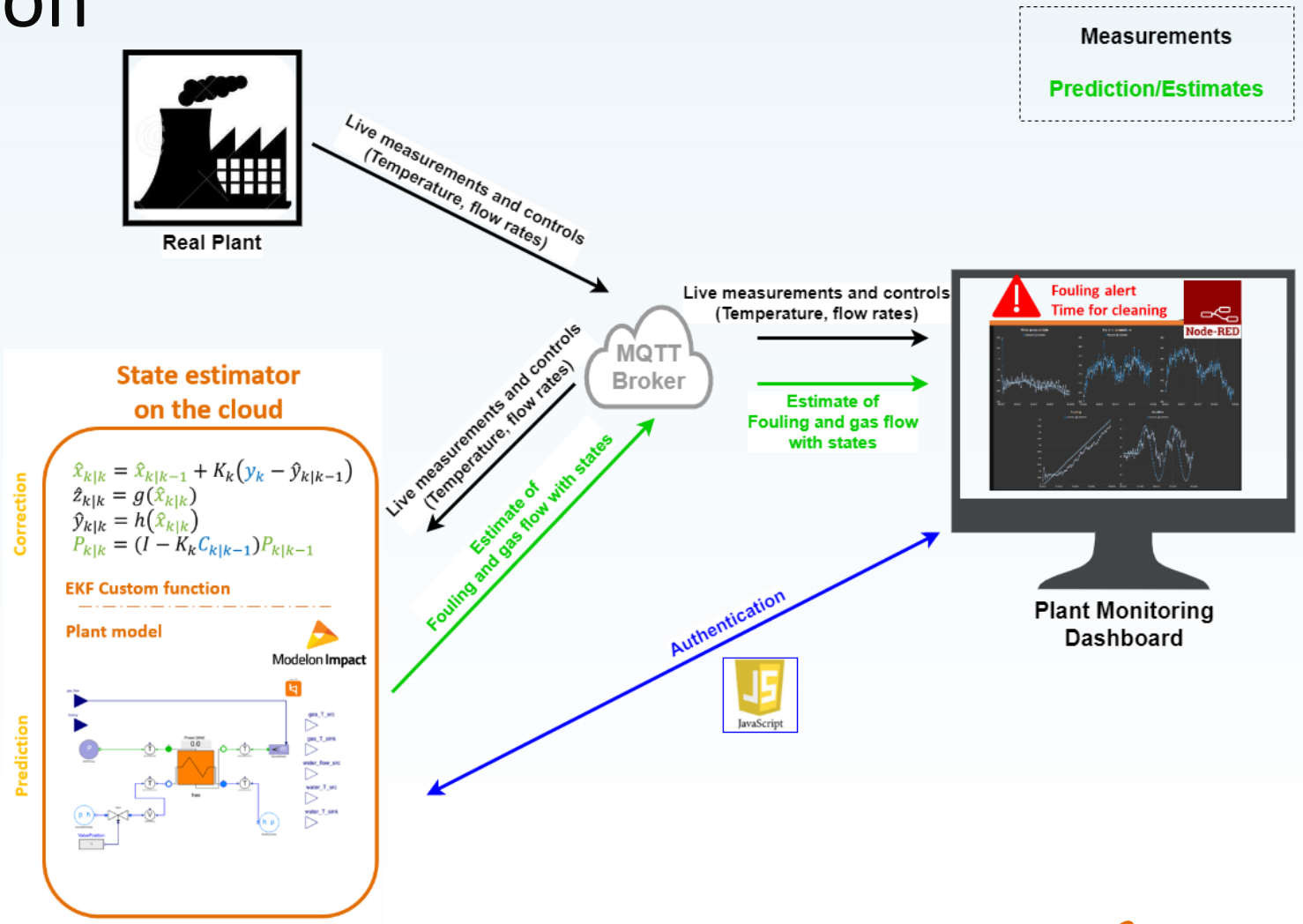
Goal: use Modelon Impact + ecosystem to build a digital twin that estimates the performance of a heat exchanger

- Bi-directional data exchange: MQTT
- Physical model on the cloud: Modelon Impact and libraries
- State estimation on the cloud: Custom function in Python
- Local dashboard for visualization: Node-red and Java-script API

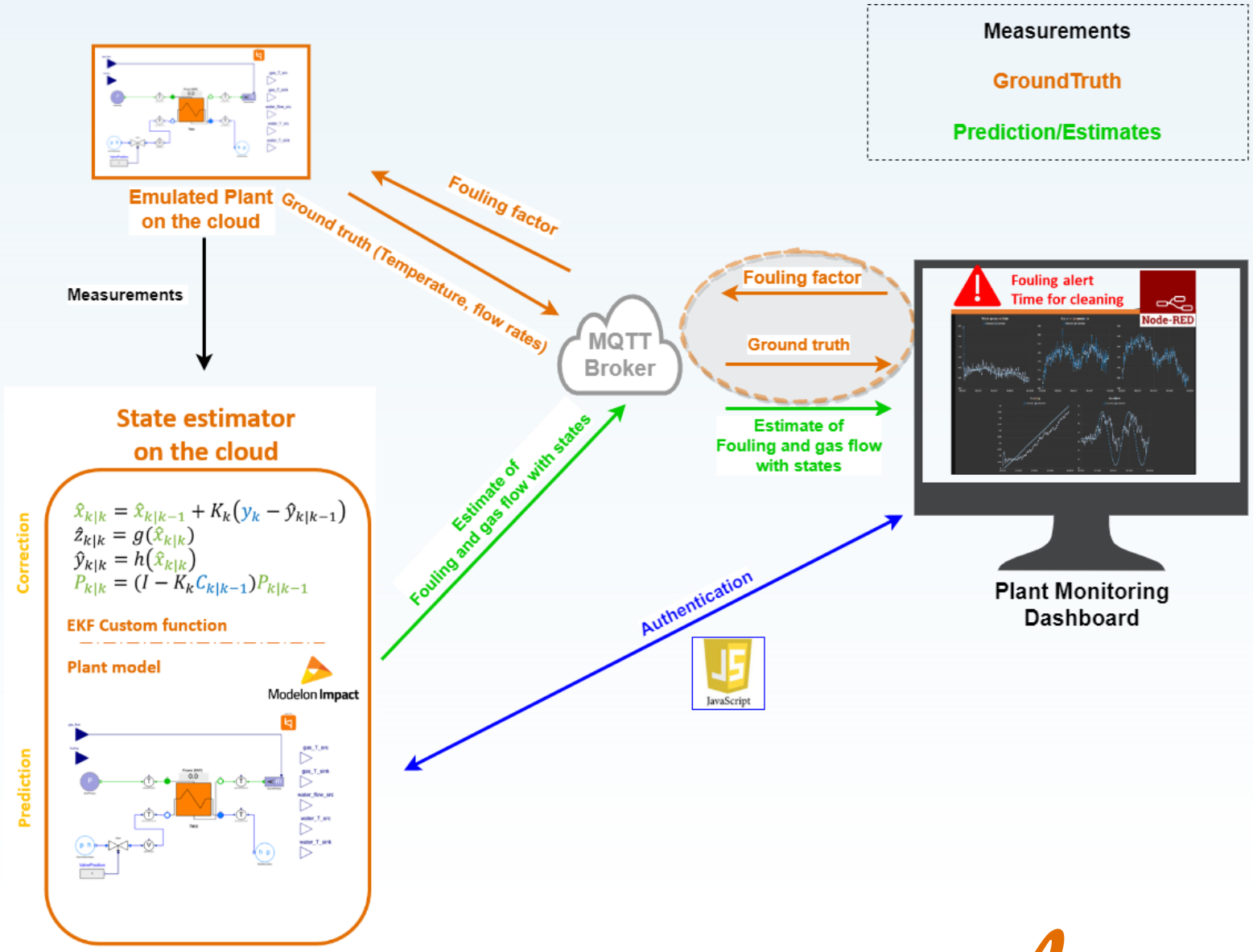
Extended Kalman filter



Target implementation



Current status



Accurate Simulations. Better Decisions.