Optimal Sizing and Operations Of Energy Systems Using GBOML

Introduction Course

February 2023

Bardhyl Miftari
Guillaume Derval
Damien Ernst

LIÈGE université

Table of Content

Sizing and Operations Problems

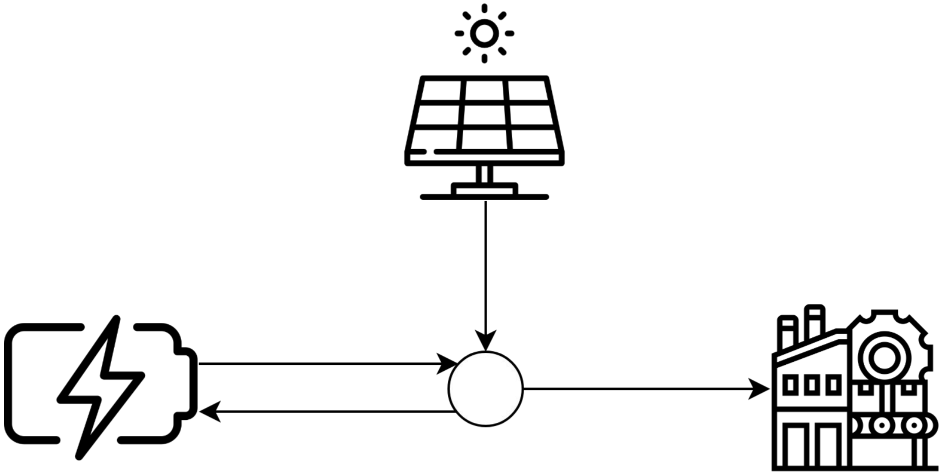
- Sizing Decisions:
  - Dimensioning the system
  - e.g, What are the investments to make?
  - e.g, What are the capacity needed?
- Operating Decisions:
  - Timestep per timestep control of entities
  - e.g, When to (dis)charge a battery?
- Objective: Criteria to optimize

Sizing and Operations Problems

- Sizing Decisions:
  - Dimensioning the system
  - e.g, What are the investments to make?
  - e.g, What are the capacities needed?
  - e.g, How many trams are needed?
- Operating Decisions:
  - Timestep per timestep control of entities
  - e.g, When to (dis)charge a battery?
  - e.g, Where and when to send the trams?
- Objective: Criteria to optimize
  - e.g, Minimize the traffic
- Example: Liege Tram

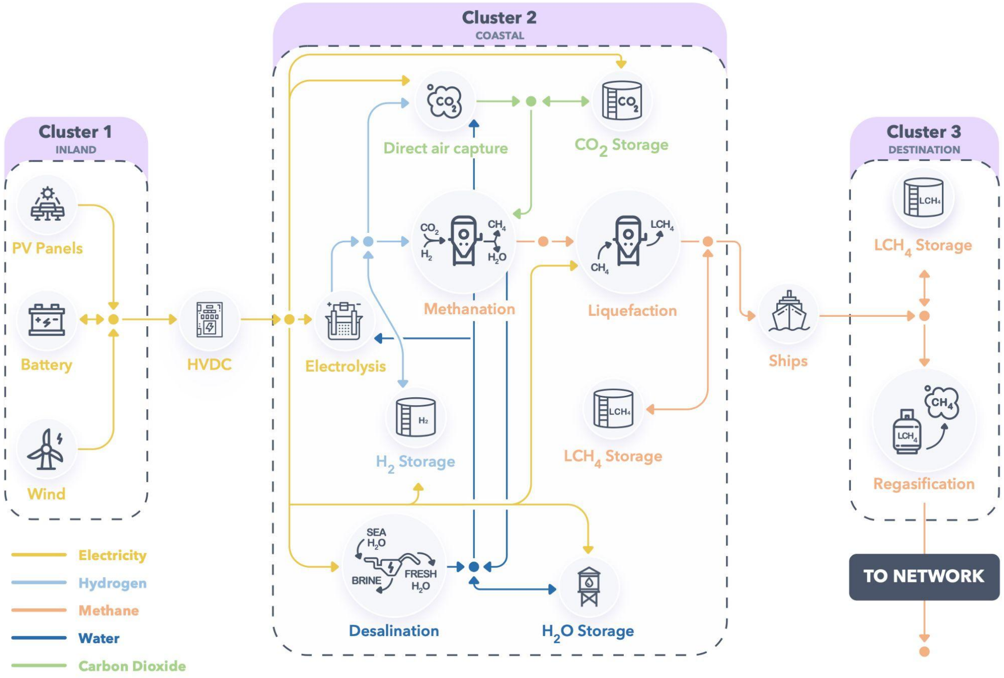
Examples in Energy: Microgrid

- Question:
  - Should we invest in PVs and batteries?
- Sizing:
  - Invest in PV panels and a battery
- Operating:
  - Control the battery
- Objective:
  - Minimize the overall cost



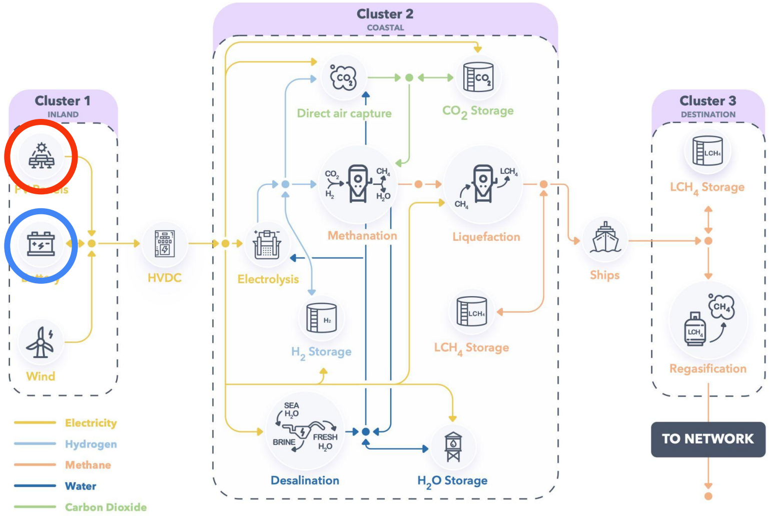
Examples in Energy: Remote Renewable Energy Hub[1]

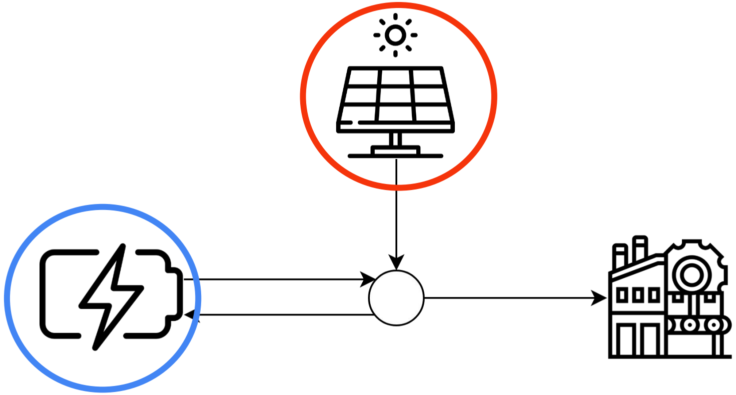
- Question:
  - What are the financial costs of producing energy in remote areas and bringing it back?
- Sizing:
  - Invest in various technologies
- Operating:
  - Control the technologies
- Objective:
  - Minimize the overall cost



Energy Planning and Control: Properties

- Recurring blocks of technologies
  - Same amongst several problems
  - Different topologies
- An optimization horizon





Energy Planning and Control: Properties

- Recurring blocks of technologies
  - Same amongst several problems
  - Different topologies
- An optimization horizon

➡ Usually solved with Mixed Integer Linear Programming

Introduction to Mathematical Optimization[2]

$$\text{minimize } f(x)$$
$$\text{such that } x \in \mathcal{X}$$

- Generic Problem:
  - An optimization function f
  - A feasible set $\mathcal{X}$

- How do we determine the feasible set?
- What resolution time to expect?

Introduction to Mathematical Optimization

- A few types of optimization problems:
  - Non-Linear Optimization (≈ hundreds of variables)

$$\min\ f(x)$$
$$\text{s.t. } g(x) \leq 0$$

  - Integer Linear Optimization (≈ thousands of variables)

$$\min\ c^T x$$
$$\text{s.t. } Ax \leq 0,\ x \in \mathbf{Z}^n$$

  - Linear Optimization (≈ millions of variables)

$$\min\ c^T x$$
$$\text{s.t. } Ax \leq 0,\ x \in \mathbf{R}^n$$

Introduction to Mathematical Optimization

- A few types of optimization problems:
  - Non-Linear Optimization

$$\min \; f(x)$$
$$\text{s.t.} \; g(x) \leq 0$$

  - Integer Linear Optimization

$$\min \; c^T x$$
$$\text{s.t.} \; Ax \leq 0, \; x \in \mathbf{Z}^n$$

  - Linear Optimization

$$\min \; c^T x$$
$$\text{s.t.} \; Ax \leq 0, \; x \in \mathbf{R}^n$$

Mixed-Integer Linear Programming (MILP)

Mixed-Integer Linear Optimization in Energy

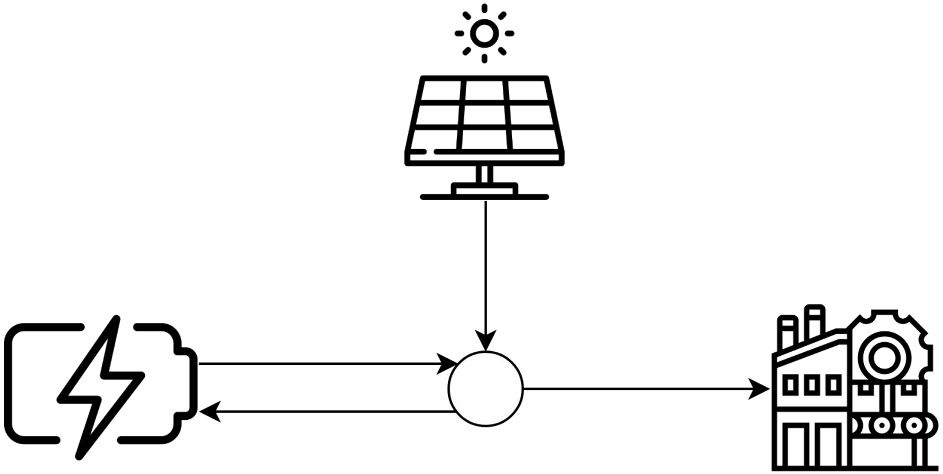
- Formulation allows to deal with relatively big problem instances

- Many energy problems have an exact MILP formulation
  - Reformulate non-linearities with piece-wise affine functions

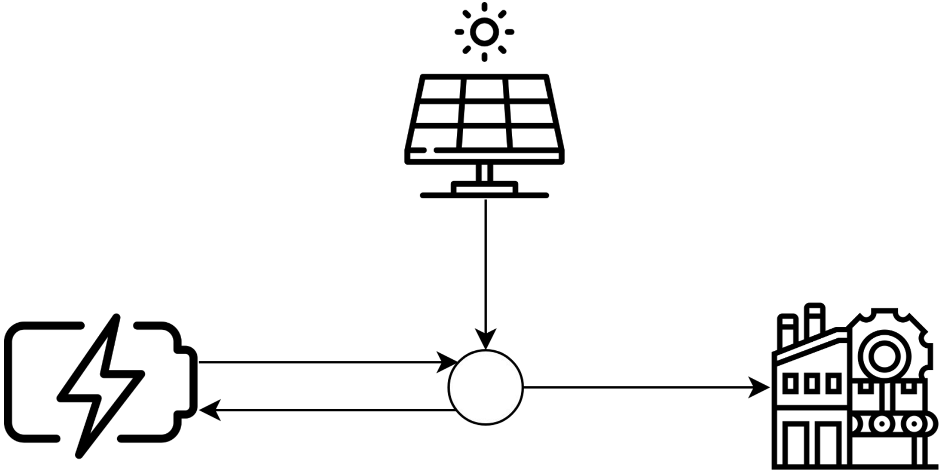
- Typical assumptions:
  - Perfect foresight and knowledge
  - Central planning and operation

Examples in Energy: Microgrid

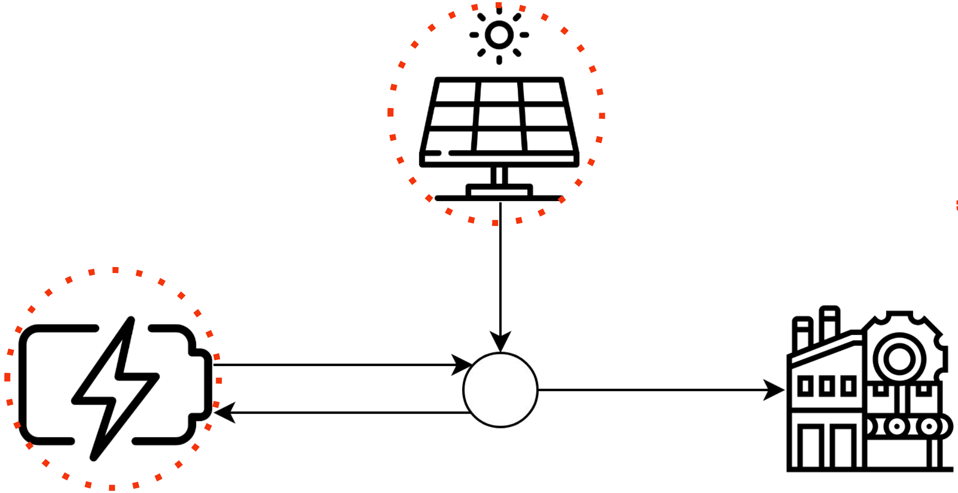
- Question:
  - Should we invest in PVs and batteries?
- Sizing:
  - Invest in PV panels and a battery
- Operating:
  - Control the battery
- Objective:
  - Minimize the overall cost



Optimization over a certain time period



Optimization over a certain time period



$$
\begin{aligned}
\min \;& [\text{investment costs}] + [\text{operation costs}] \\
\text{s.t.} \;& [\text{system design constraints}] \\
& [\text{operating constraints of subsystem}]
\end{aligned}
$$

$$
\begin{aligned}
& [\text{coupling constraints}] \\
& [\text{regulatory constraints}]
\end{aligned}
$$

For example, we may want to *design* and *operate* a system that minimizes the overall bill of a factory

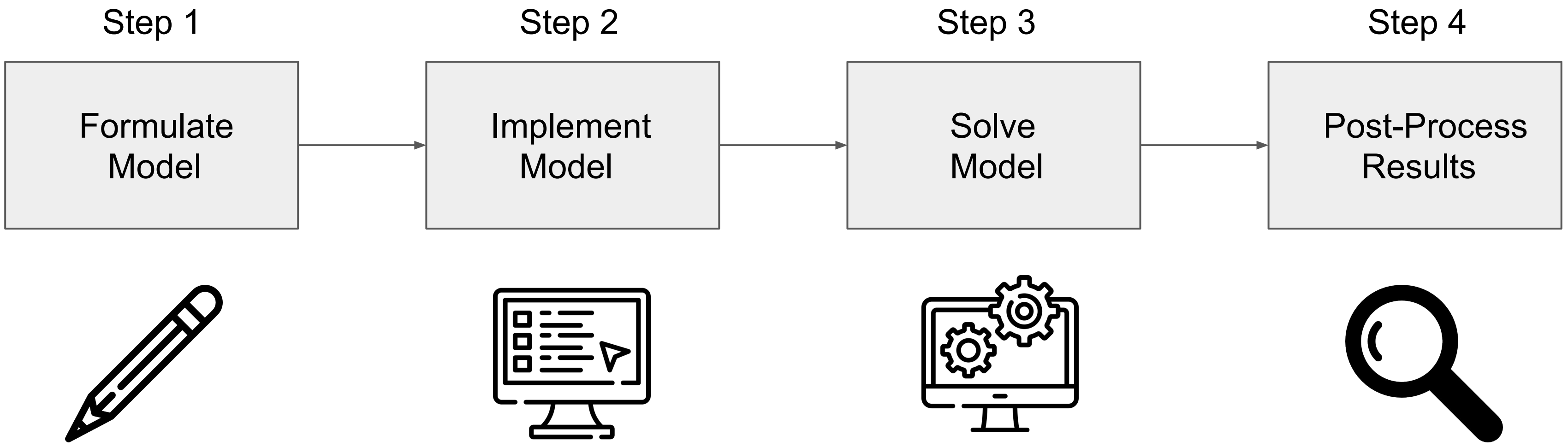
$$\min \sum_{\text{subsystems}} \left([\text{investment costs}] + [\text{operating costs}] \right)$$

$$\begin{aligned} \text{s.t. } & [\text{system design constraints}], \text{ for every subsystem} \\ & [\text{operating constraints}], \text{ for every subsystem} \\ & [\text{coupling constraints between subsystems}] \\ & [\text{regulatory and environmental constraints}] \end{aligned}$$

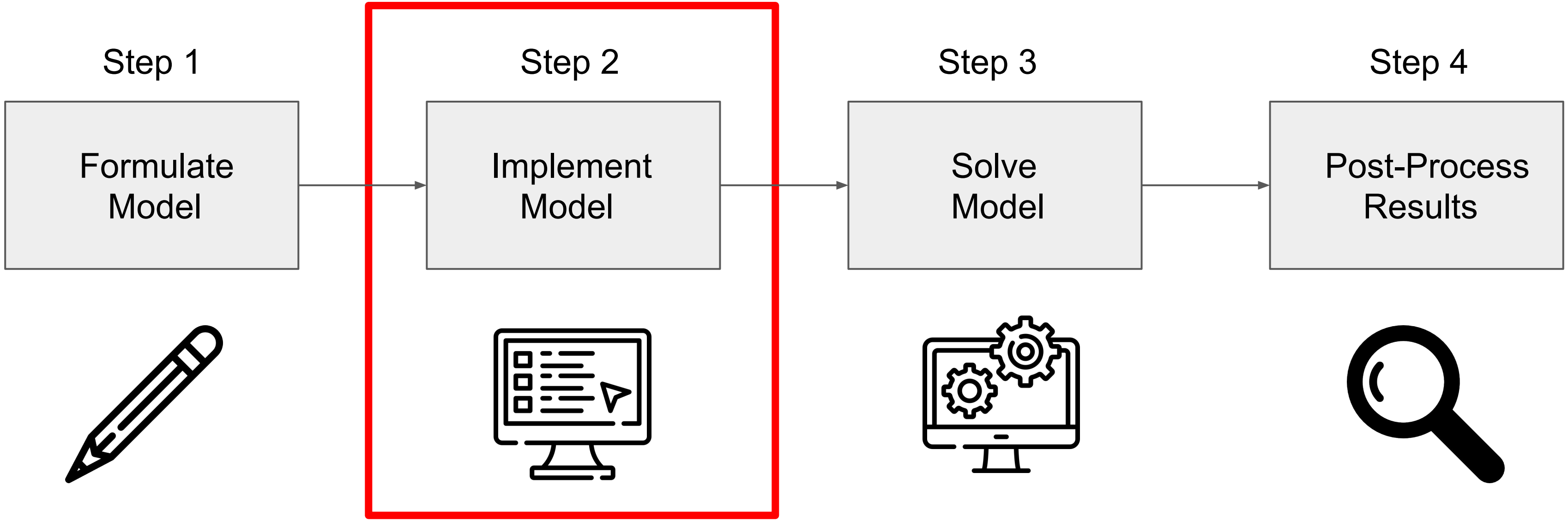
This structure can be represented via a hypergraph abstraction augmented with some concept of time-indexing

$$
\begin{aligned}
\min \quad & \textstyle\sum_{n \in \mathcal{N}} \left[f_0^n(X^n, Z^n) + \sum_{t \in \mathcal{T}} f^n(X^n, Z^n, t) \right] \\
\text{s.t.} \quad & h_k^n(X^n, Z^n, t) = 0, \ \forall t \in \mathcal{T}_k^n, \ k = 1, \dots K^n, \ \forall n \in \mathcal{N} \\
& g_k^n(X^n, Z^n, t) \leq 0, \ \forall t \in \bar{\mathcal{T}}_k^n, \ k = 1, \dots \bar{K}^n, \ \forall n \in \mathcal{N} \\
& H^e(Z^e) = 0, \ \forall e \in \mathcal{E} \\
& G^e(Z^e) \leq 0, \ \forall e \in \mathcal{E} \\
& X^n \in \mathcal{X}^n, Z^n \in \mathcal{Z}^n, \ \forall n \in \mathcal{N}.
\end{aligned}
$$

Working with optimization models involves at least four basic steps



We will focus on the second step: model encoding and implementation



Two classes of tools are available to implement models

1. Algebraic Modeling Languages (AMLs):

- **Formulation** close to **mathematical notation** (e.g., index-based notation)
- Very **expressive** (e.g., can represent any mixed-integer nonlinear program)
- Often interface with **multiple solvers**
- Sometimes **open source**
- Examples :

Two classes of tools are available to implement models

2. Object-Oriented Modeling Environments (OOMEs):

- Focus on **one** particular **application** (e.g., generation expansion planning)
- Usually make use of **predefined components** that can be “imported”
- Typically have advanced **data processing** capabilities tailored to the application
- Often **open source**
- Examples :

PyPSA PLEXOS Dispa-SET Power system modelling

Calliope BALMOREL Energy system model

Each approach has drawbacks

AMLs typically fail to **expose** or **exploit** block **structure**, although this may be used to:

- simplify model encoding
- enable model re-use
- speed up model generation
- facilitate the use of structure-exploiting algorithms

OOMEs, usually:

- Lack **expressiveness**
- Often **cumbersome** to add **new components**
- Usually **rely** on **AMLs** and inherit some of their **shortcomings**

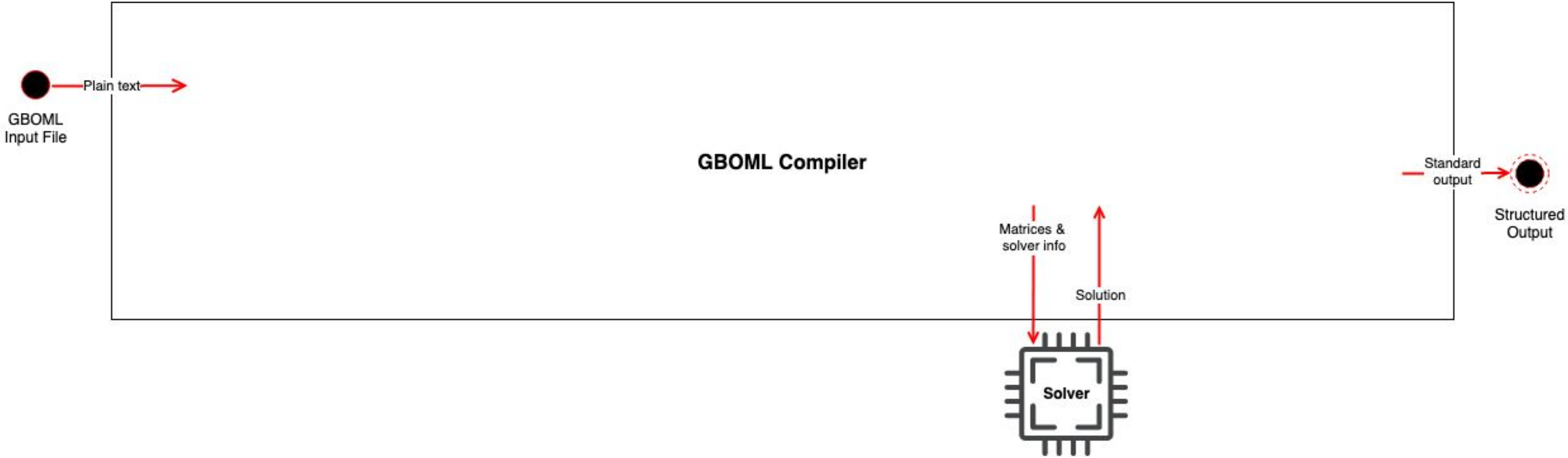
GBOML combines the strengths of AMLs and OOMEs[3]

- **open-source** and **stand-alone**
- any Mixed-Integer Linear Program (MILP) can be represented
- **hierarchical block structure** can be exposed and exploited
- syntax is close to **mathematical notation**
- **time-indexed models** can be encoded easily
- **re-using** and **combining** model components is straightforward
- interfaces with **various solvers** are available

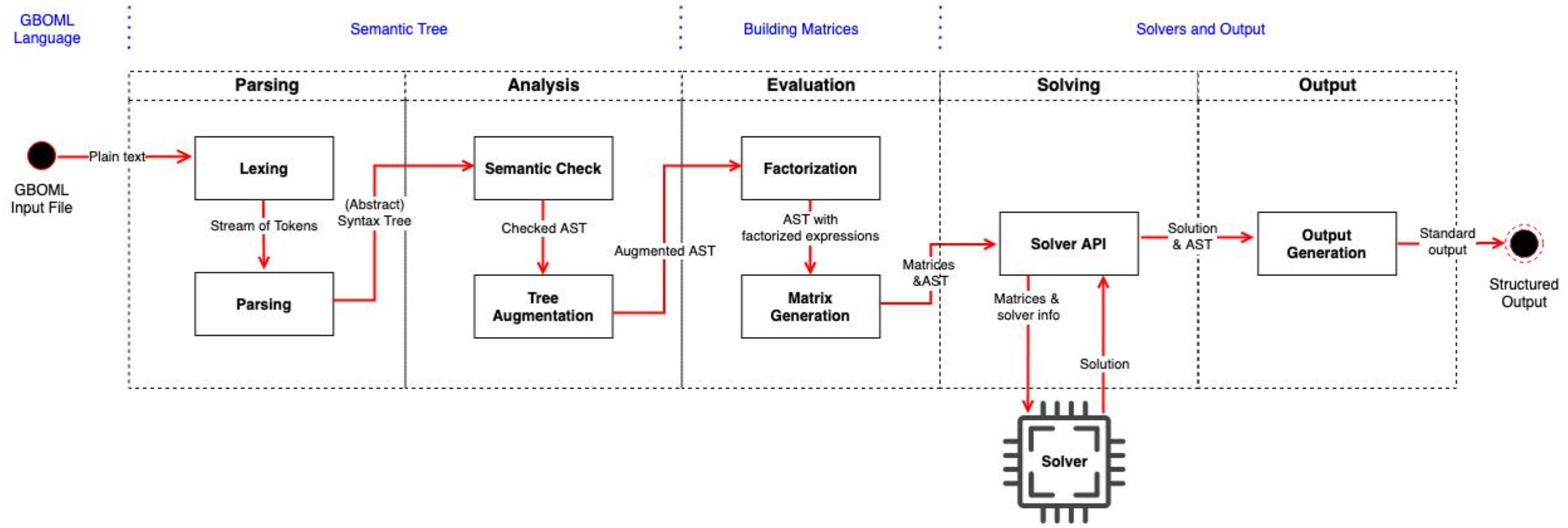
GBOML Compiler[4]

- Software developed in Python:
  - Has very **few dependencies** (PLY, NumPy, SciPy)
  - Provides two methods to **encode** models (**text file** and Python **API**)
  - Interfaces with **several solvers** (Gurobi, CPLEX, Xpress, Cbc/Clp, HiGHS and DSP)
  - Produces plain **.csv** or structured **.json** outputs
- Model structure is exploited on multiple levels:
  - Model **encoding** via dedicated language constructs
  - Model **generation** via parallelism and multiprocessing
  - **Solving** via structure-exploiting solvers such as DSP
- Fully documented - Clear issue handling

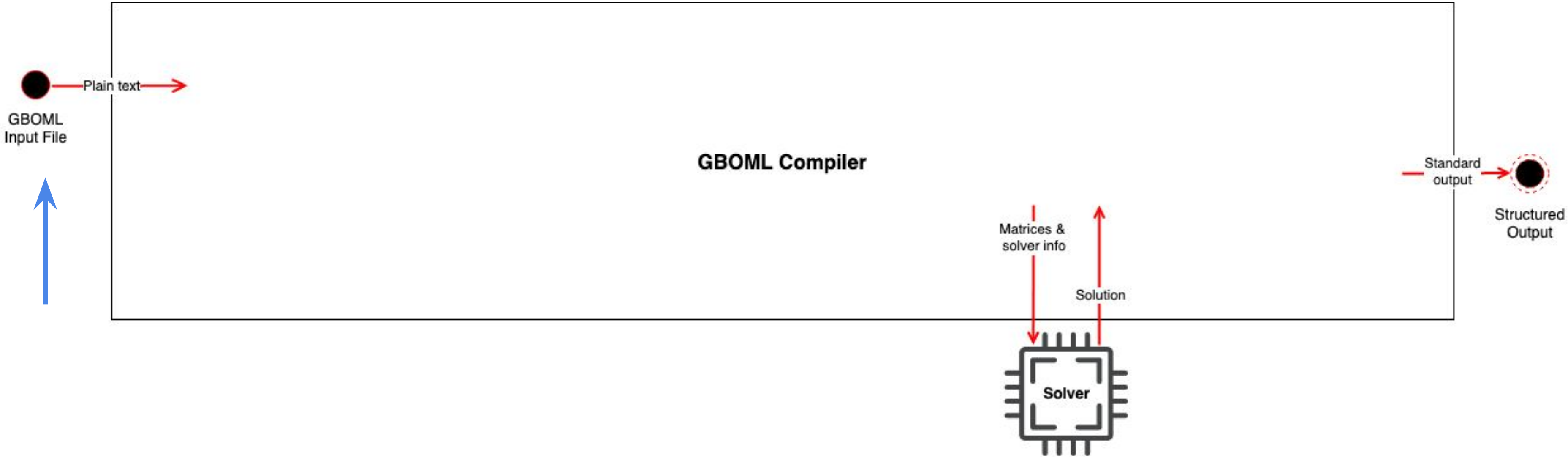
Simplified GBOML Workflow



Full GBOML Workflow



GBOML: The Language



GBOML: The Language

```
#TIMEHORIZON ...

#NODE <node identifier>
#PARAMETERS
// parameter definitions
#VARIABLES
// variable definitions
#CONSTRAINTS
// constraint definitions
#OBJECTIVES
// objective definitions

#NODE
...

#HYPEREDGE <identifier>
#PARAMETERS
// parameter definitions
#CONSTRAINTS
// constraint definitions
```

Example: PV Panels

```
#NODE SOLAR_PV
    #PARAMETERS
        capex = 600; // annualised capital expenditure per unit capacity
        capacity_factor = import "gboml/examples/microgrid/pv_gen.csv"; // normalised generation profile
    #VARIABLES
        internal: capacity; // capacity of solar PV plant
        external: power[T]; // power output of solar PV plant
    #CONSTRAINTS
        capacity >= 0;
        power[t] >= 0;
        power[t] <= capacity_factor[mod(t,24)] * capacity;
    #OBJECTIVES
        min: capex * capacity;
```

Re-use

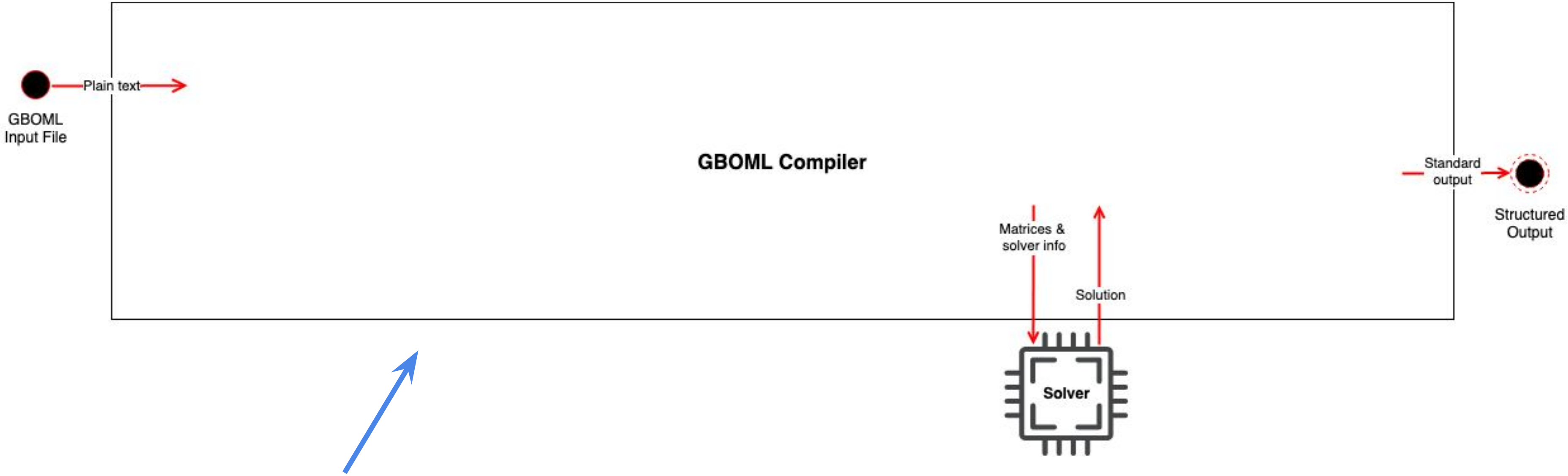
- Let us consider file1.txt

```
#NODE DEMAND
#PARAMETERS
demand = import "demand.csv";
#VARIABLES
external: consumption[T];
#CONSTRAINTS
consumption[t] == demand[mod(t, 24)];
```

- To import that node

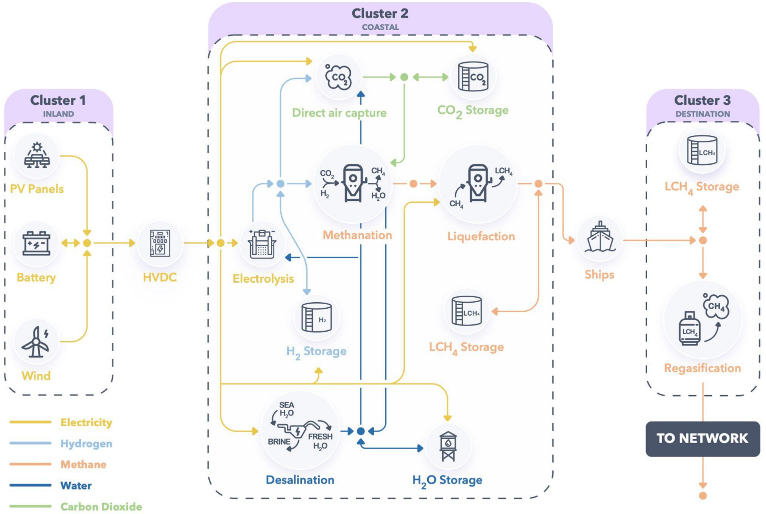
```
#NODE DEMAND = import DEMAND from "file1.txt";
```

GBOML: Compiler Performance



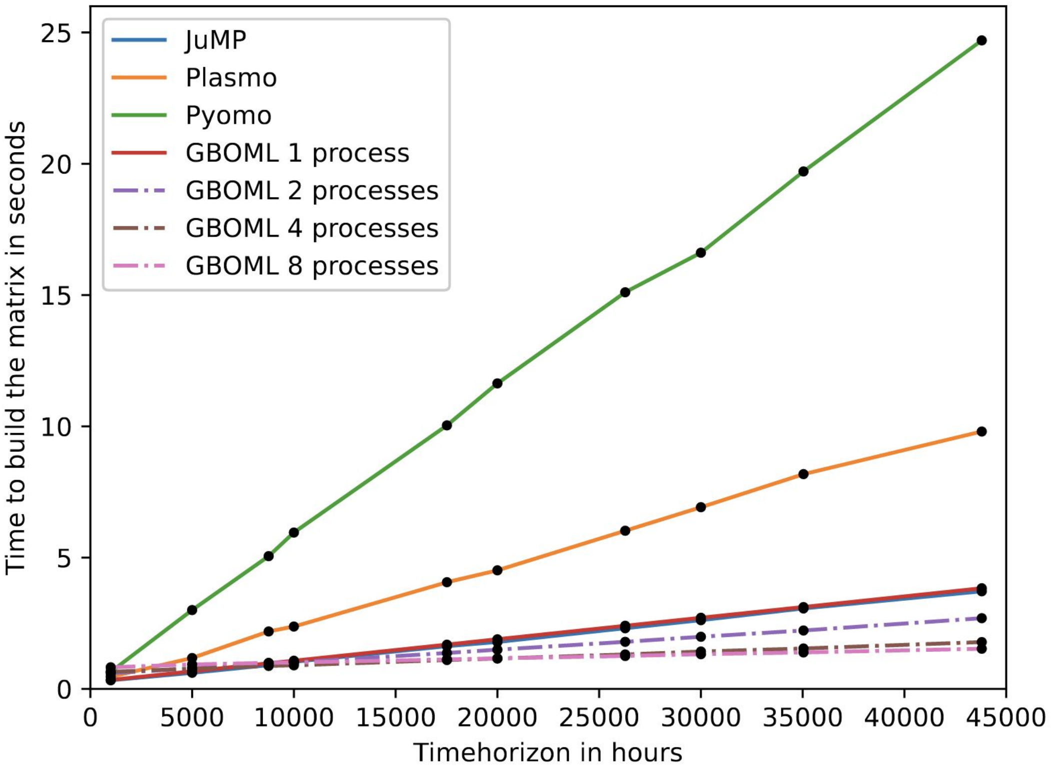
GBOML: Compiler Performance[5]

- Compare GBOML - JuMP - Plasmo - Pyomo
  - Remote Renewable Energy Hub
  - Time to build the model
  - Peak RAM usage

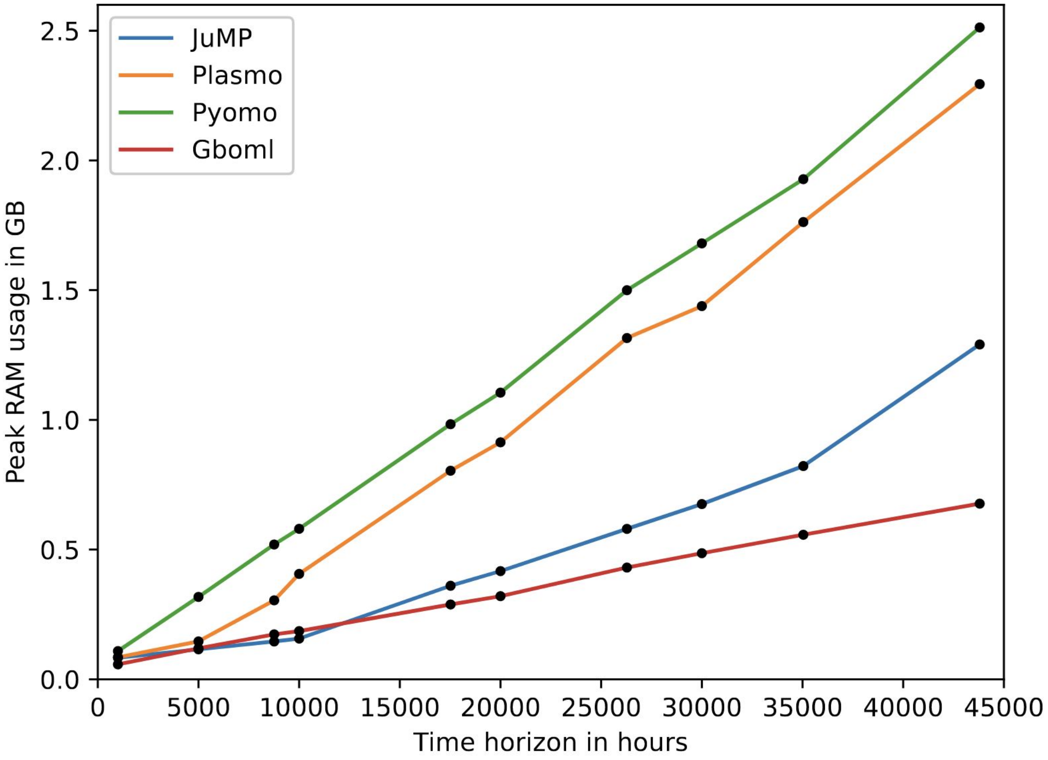


- Exploiting problem structure in resolution
  - MIPLIB Noswot
  - Gurobi - DSP

Results: Time to generate the model[5]

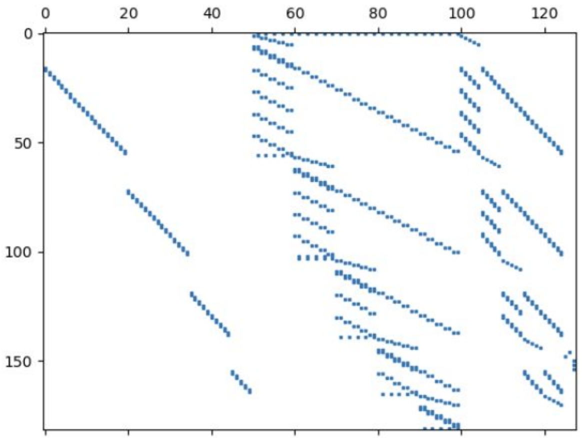


Results: Peak RAM usage[5]



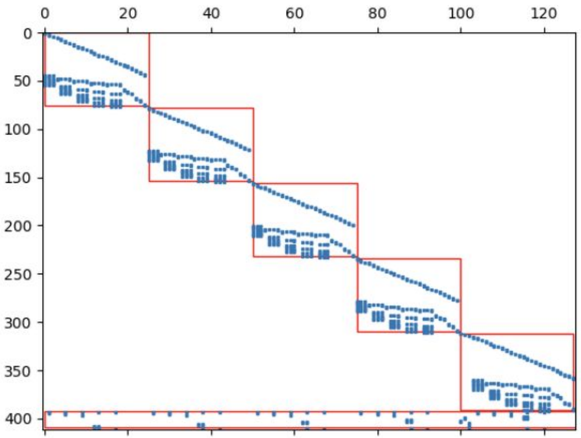
Structure exploiting methods[5]

- “MIPLIB noswot” problem



(a) Original Representation

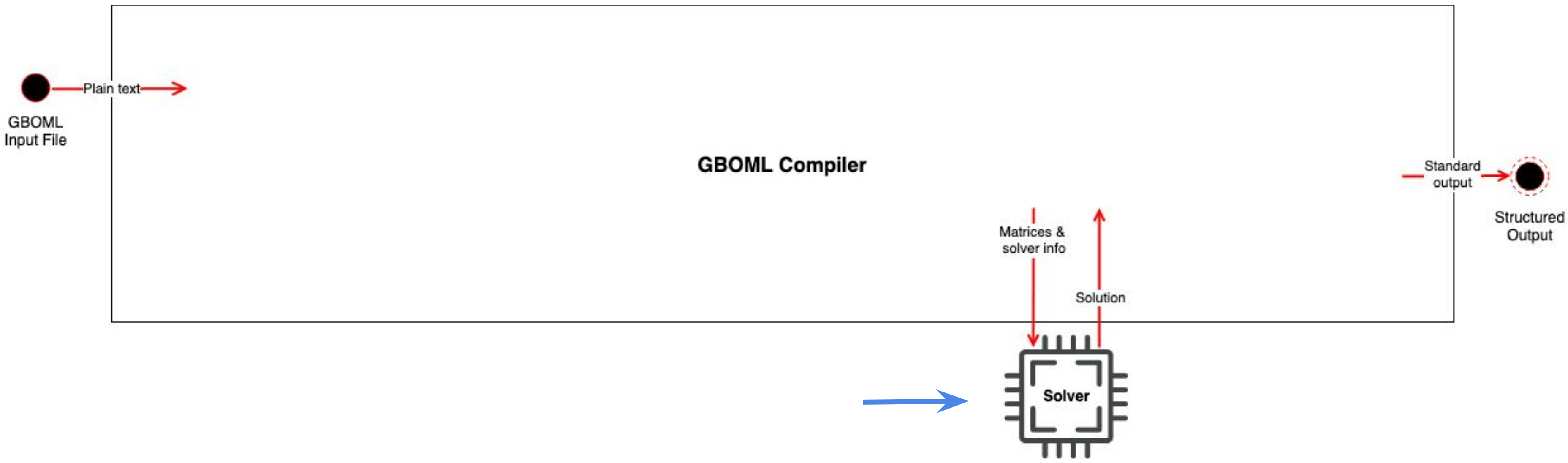
Resolution Gurobi: ≃25s
(no structure is considered)



(b) GBOML representation

Resolution DSP: ≃2.2s
(structure taken into account)

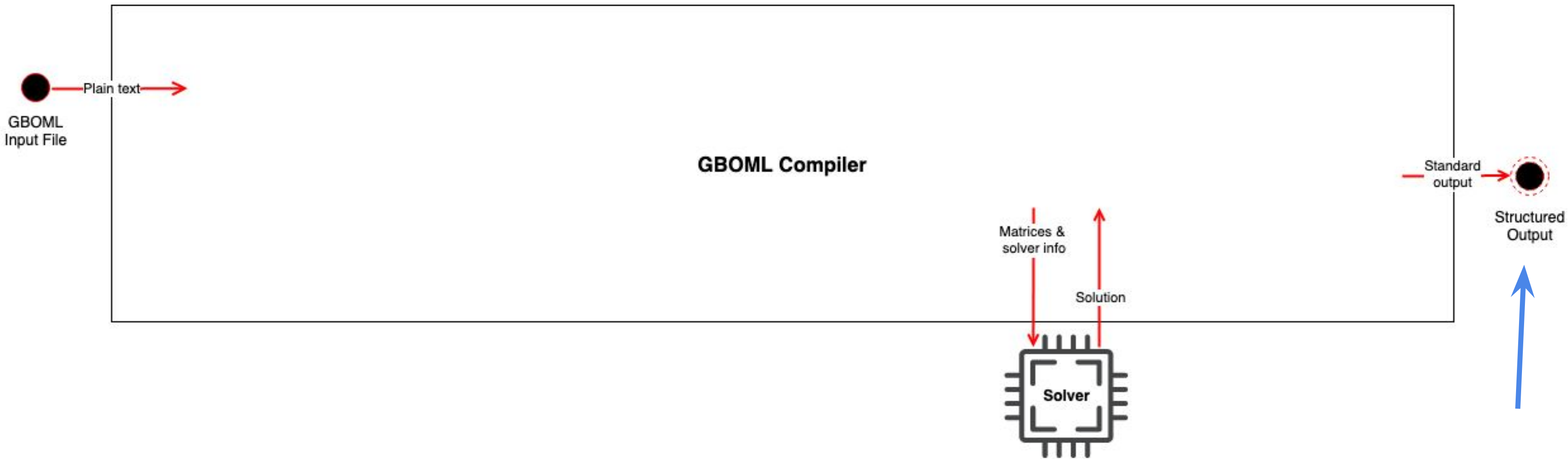
Simplified GBOML Workflow



MILP Solvers

- Open-source:
  - CBC
  - HiGHS
  - SCIP (GBOML is yet to interface with it)
- Commercial:
  - Fico Xpress
  - Gurobi
  - IBM Cplex
- Meta-Solver:
  - DSP

Simplified GBOML Workflow



GBOML Output

- Standardized output either CSV or JSON:

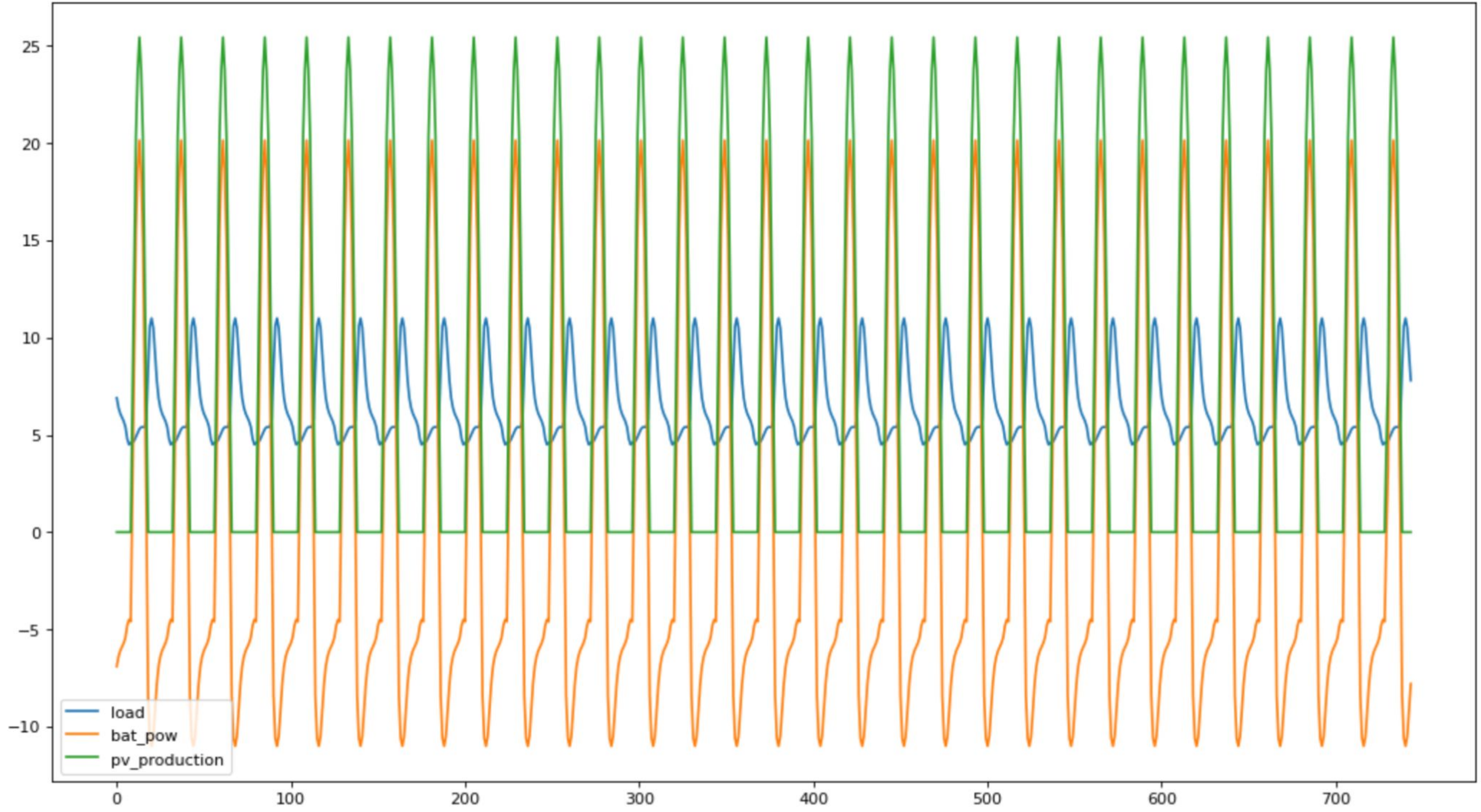

```json
{
    "version": "0.1.3",
    "model": {
        "horizon": 10,
        "number_nodes": 1,
        "global_parameters": {},
        "nodes": {
            "H": {
                "number_parameters": 1,
                "number_variables": 1,
                "number_constraints": 1,
                "number_expanded_constraints": 10,
                "number_objectives": 1,
                "number_expanded_objectives": 10,
                "parameters": {
                    "b": [
                        4
                    ]
                },
                "variables": [
                    "x"
                ]
            }
        },
        "hyperedges": {}
    },
    "solver": {
        "name": "linprog",
        "status": true
    },
```

GBOML Output

- Standardized output either CSV or JSON:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	DISTRIBUTION.operating_cost	0.34500000000000003	0.32000000000000006	0.305	0.29500000000000004	0.28500000000000003	0.27	0.24	0.22500000000000003	0.22999999999999998	0.22999999999999998	0.23500000000000004	0.24500000000000005	0.255
2	DISTRIBUTION.power_import	6.9	6.400000000000001	6.1	5.9	5.700000000000001	5.4	4.8	4.5	4.6	4.6	4.7	4.9	5.1
3	DISTRIBUTION.unnamed_objective	2817.4100000000003												
4	DEMAND.consumption	6.9	6.400000000000001	6.1	5.9	5.700000000000001	5.4	4.8	4.5	4.6	4.6	4.7	4.9	5.1
5	BATTERY.capacity	-0.0												
6	BATTERY.investment_cost	0.0												
7	BATTERY.energy	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	BATTERY.charge	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	BATTERY.discharge	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	BATTERY.unnamed_objective	0.0												
11	SOLAR_PV.capacity	-0.0												
12	SOLAR_PV.investment_cost	0.0												
13	SOLAR_PV.electricity	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14	SOLAR_PV.investment	0.0												

GBOML Output for Microgrid



Conclusion - GBOML

- A modelling tool for supply chain management and energy system planning and sizing
- Allows easy model re-use and combination
- Exploits the structure
  - Encoding via the language
  - Internally in the model representation and parallelization
  - Interfacing with structure exploiting methods
- Performance on a large problem (remote hub):
  - Better peak RAM usage than JuMP & Plasmo
  - Similar times to JuMP, faster than others
  - With parallelization, faster than all

Tutorial Session

https://colab.research.google.com/drive/15jmzQPLIfSILNCcK6fEv2UnvVbT8s6yL?usp=sharing

Acknowledgements

- We would like to thank:
  - SPF Economie for their financial support through the INTEGRATION project

  - Mathias Berger for his work on GBOML and a previous version of this presentation
  - Amina Benzerga for her feedback on this course

References:

[1] Mathias Berger et al., “Remote Renewable Hubs for Carbon-Neutral Synthetic Fuel Production”, in Frontiers in Energy Research 9 (2021), p.200. DOI 10.3389/fenrg.2021.671279. https://www.frontiersin.org/article/10.3389/fenrg.2021.671279

[2] D. Bertsimas, J. Tsistsiklis. Introduction to linear optimization, Dynamic Ideas, 1997.

[3] Mathias Berger et al., “Graph-Based Optimization Modelling Language: A Tutorial”, https://orbi.uliege.be/handle/2268/256705, 2021

[4] Bardhyl Miftari et al., GBOML: Graph-Based Optimization Modeling Language, https://joss.theoj.org/papers/10.21105/joss.04158, 2022

[5] Bardhyl Miftari et al., GBOML: a Structure-exploiting Optimization Modeling Language in Python, https://orbi.uliege.be/handle/2268/296930, 2022

[6] Bardhyl Miftari et al., “GBOML repository”, https://gitlab.uliege.be/smart_grids/public/gboml, 2021-23

Learn more:

Group publication: http://blogs.ulg.ac.be/damien-ernst