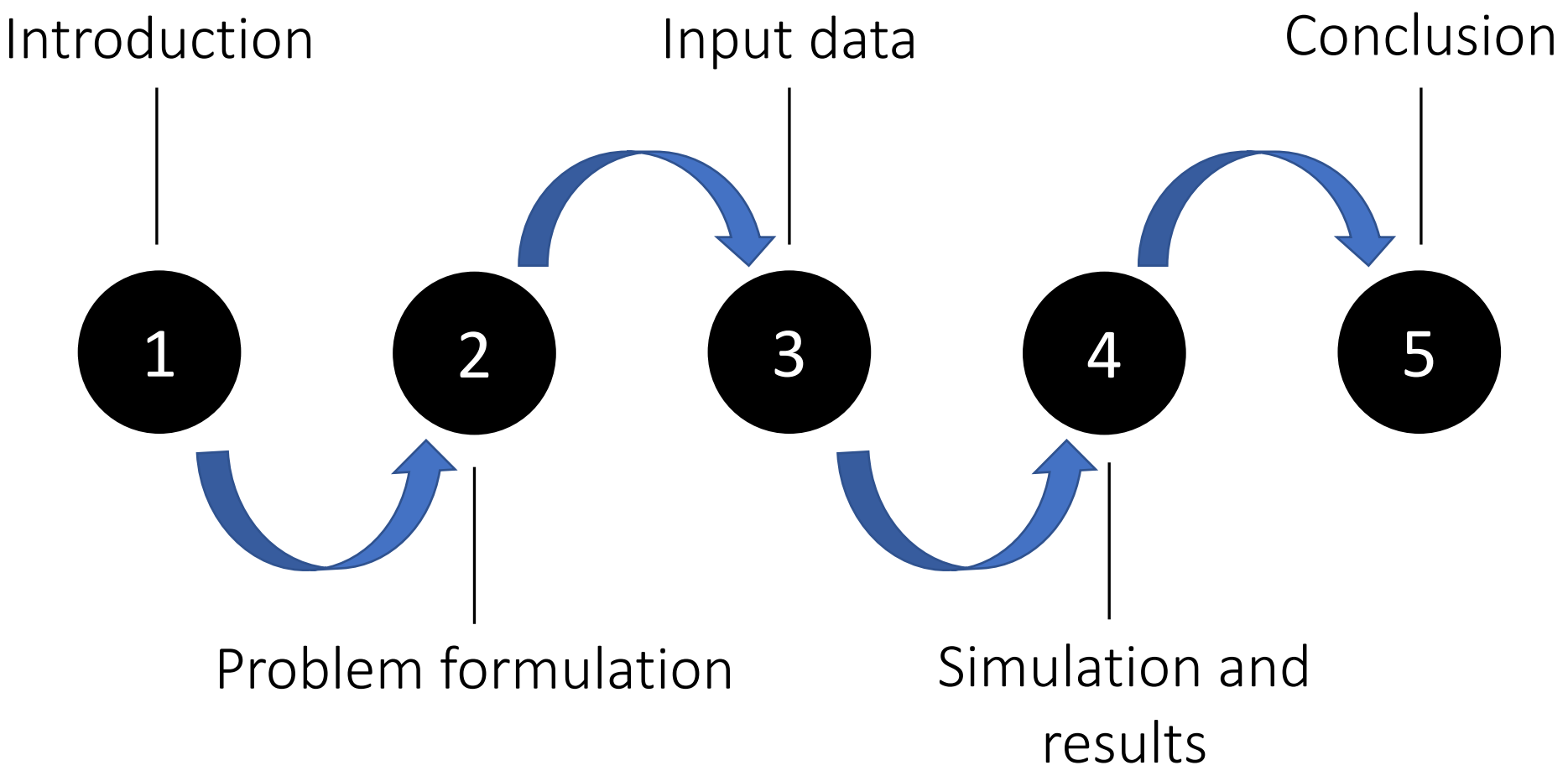
Optimal Scheduling of Vehicle-to-Grid Energy and Ancillary Services

InnoEnergy
Knowledge Innovation Community

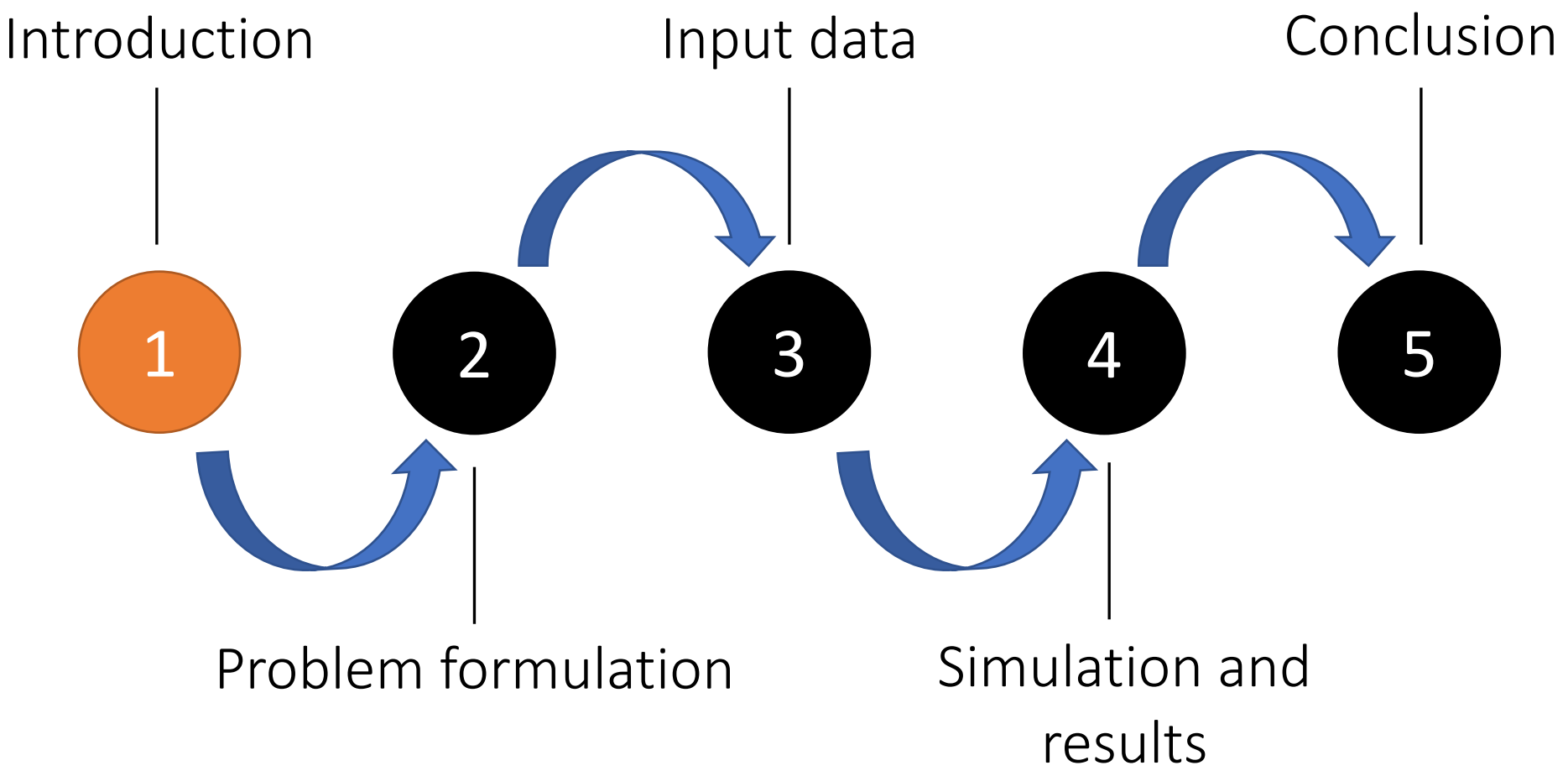
TÉCNICO LISBOA

July 2nd, 2020
Andrea Bertolini

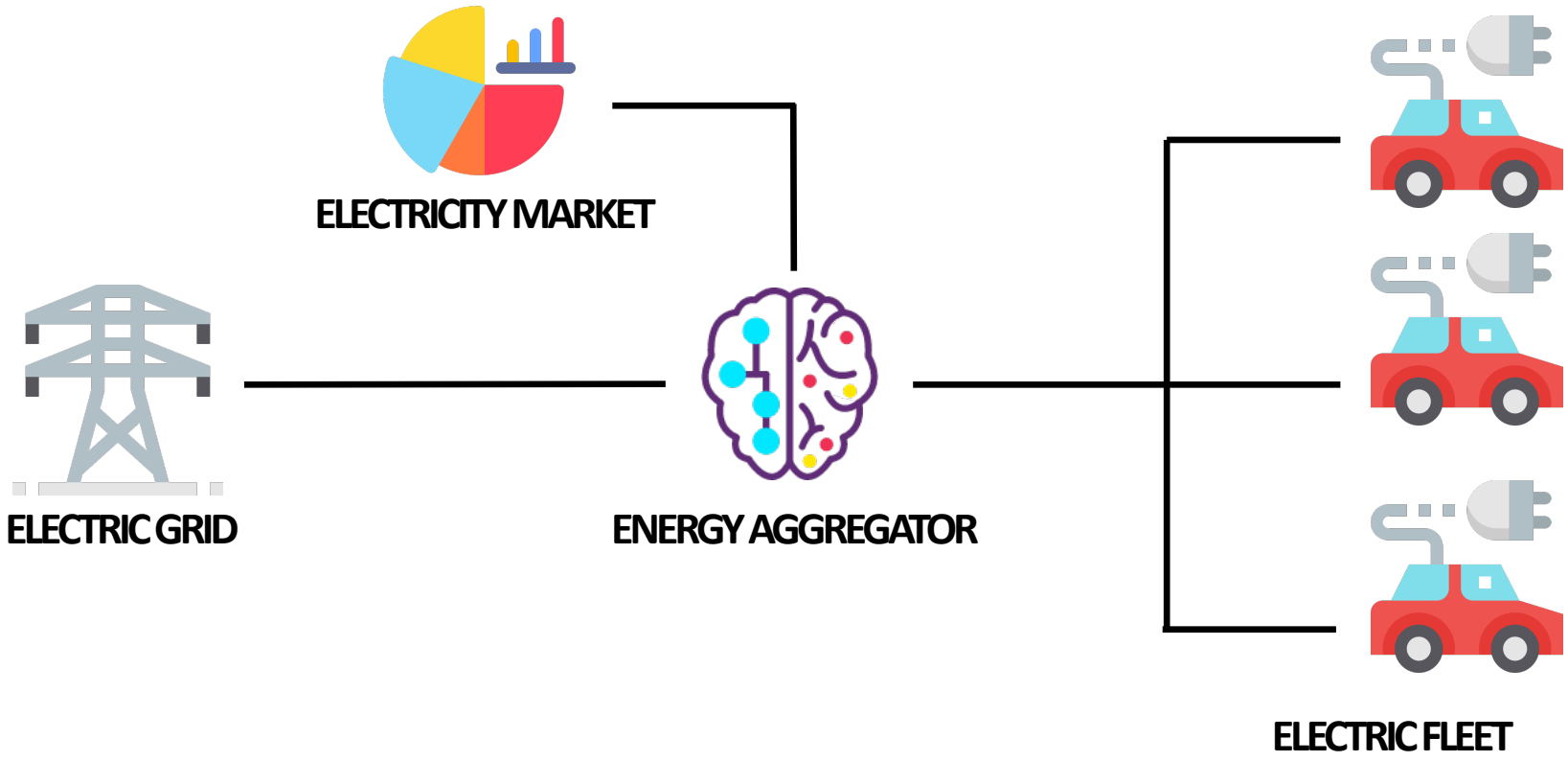
Agenda


Introduction
Input data
Conclusion
1
2
3
4
5
Problem formulation
Simulation and results

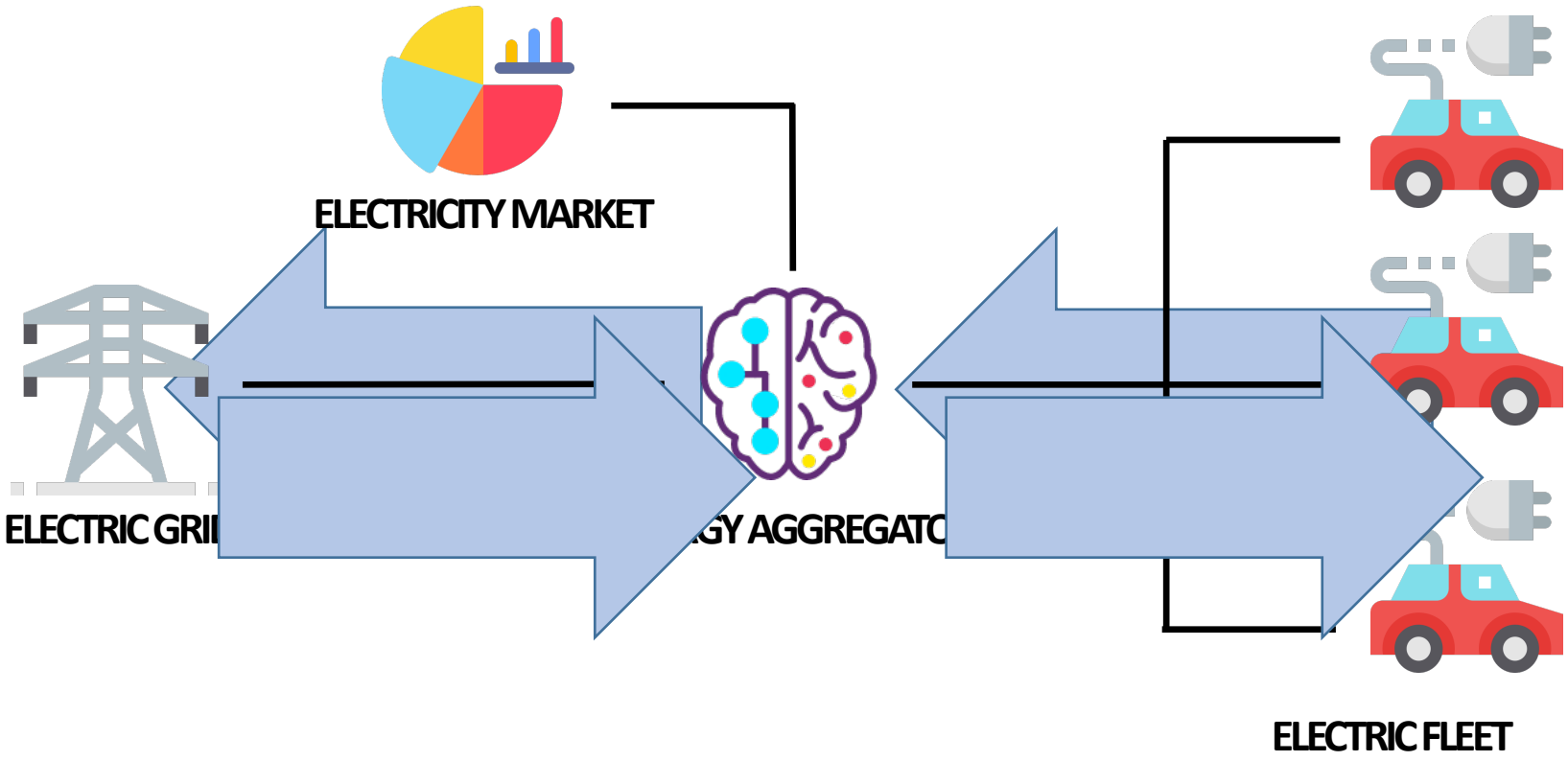
Agenda


Introduction
Input data
Conclusion
1
2
3
4
5
Problem formulation
Simulation and results

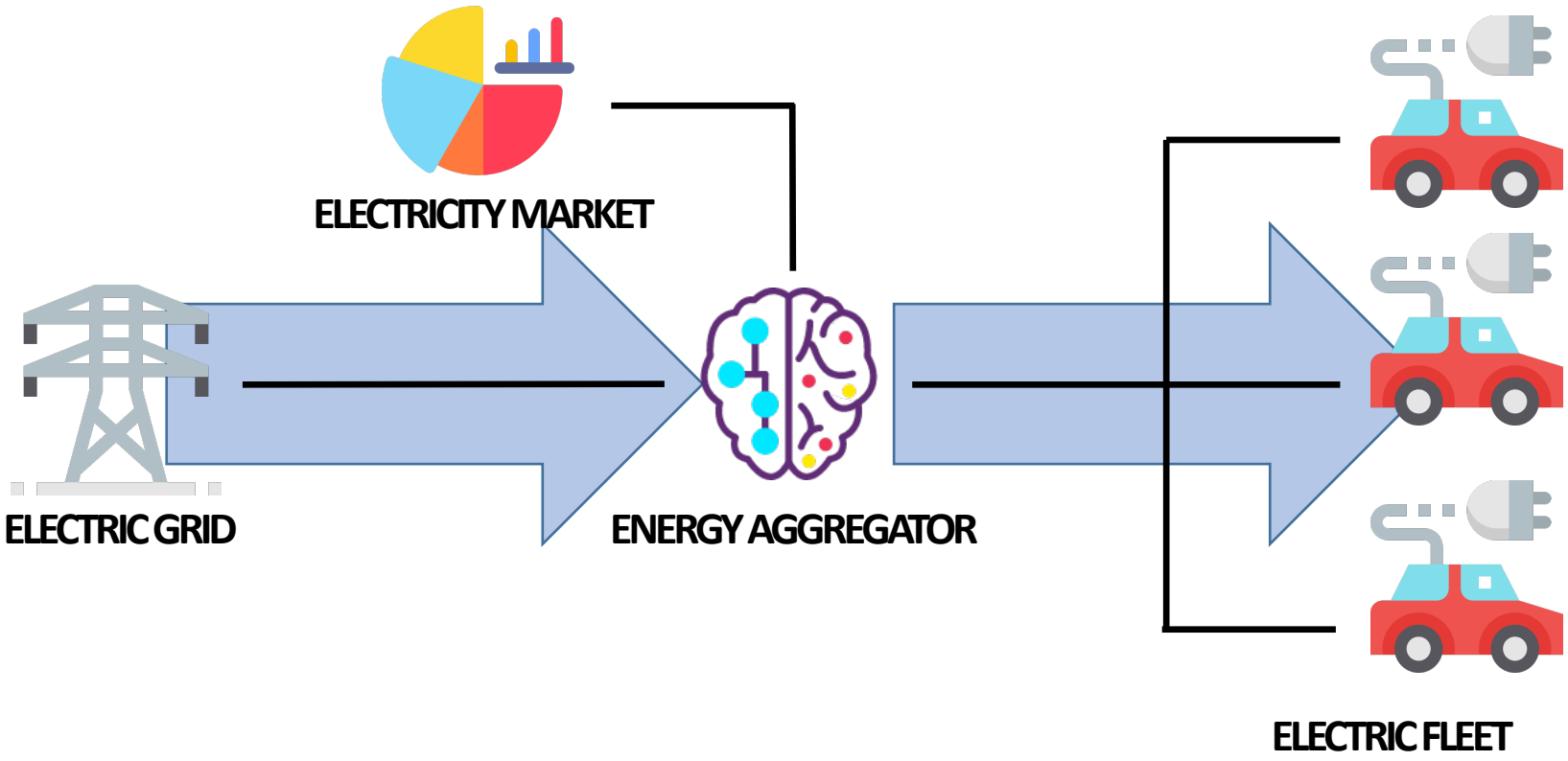
Introduction



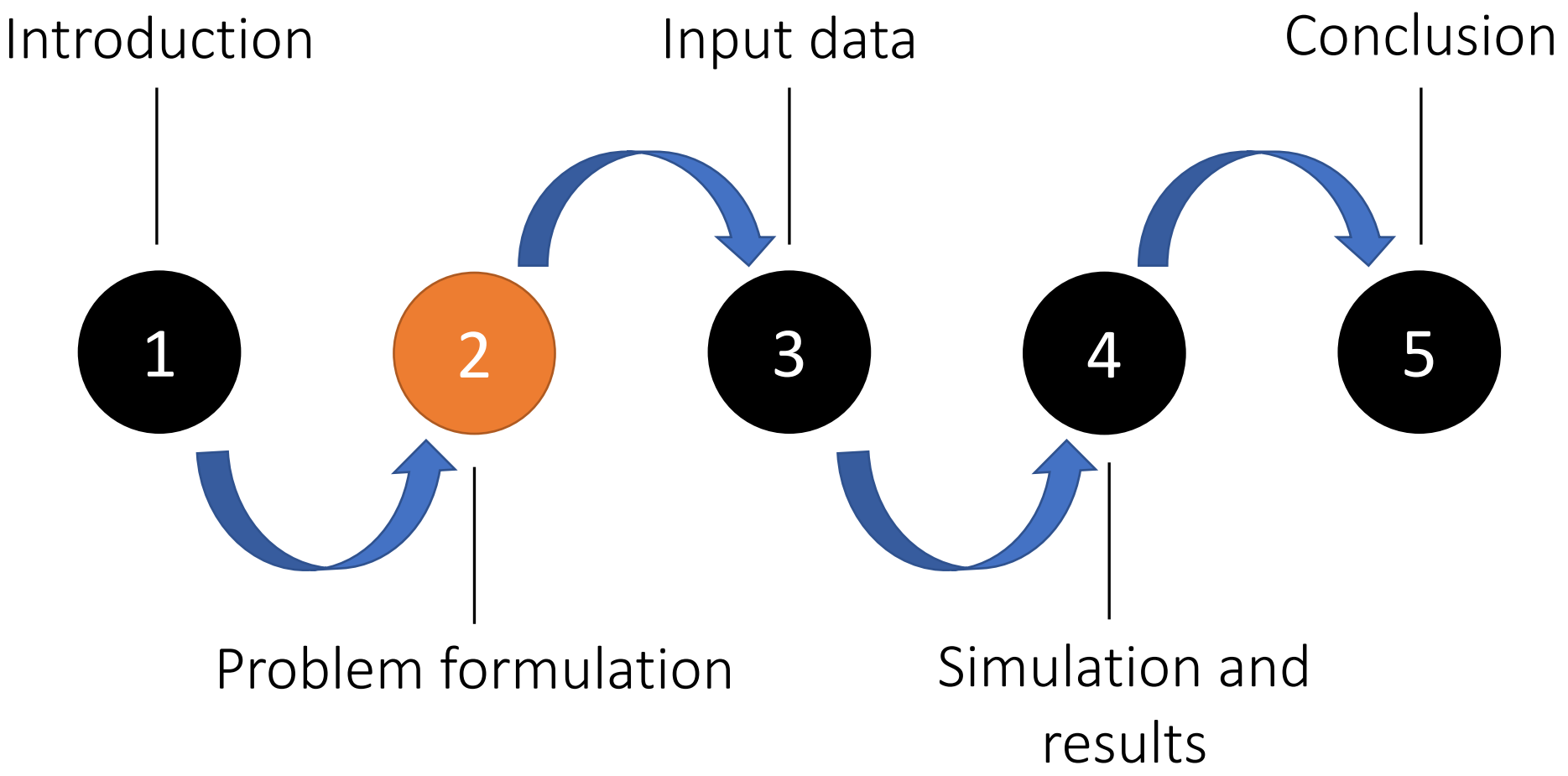
Introduction - V2G



Introduction - Smart Charging



Agenda


Introduction
Input data
Conclusion
1
2
3
4
5
Problem formulation
Simulation and results

Software package & work-flow

Python programming language and **Pyomo** library

Run the script and **solve** the problem (GLPK)

Energy and ancillary services prices collected in **excel spreadsheets**

Converted all input data of the model in a **DAT file**

Analysis of the **results** from the optimization

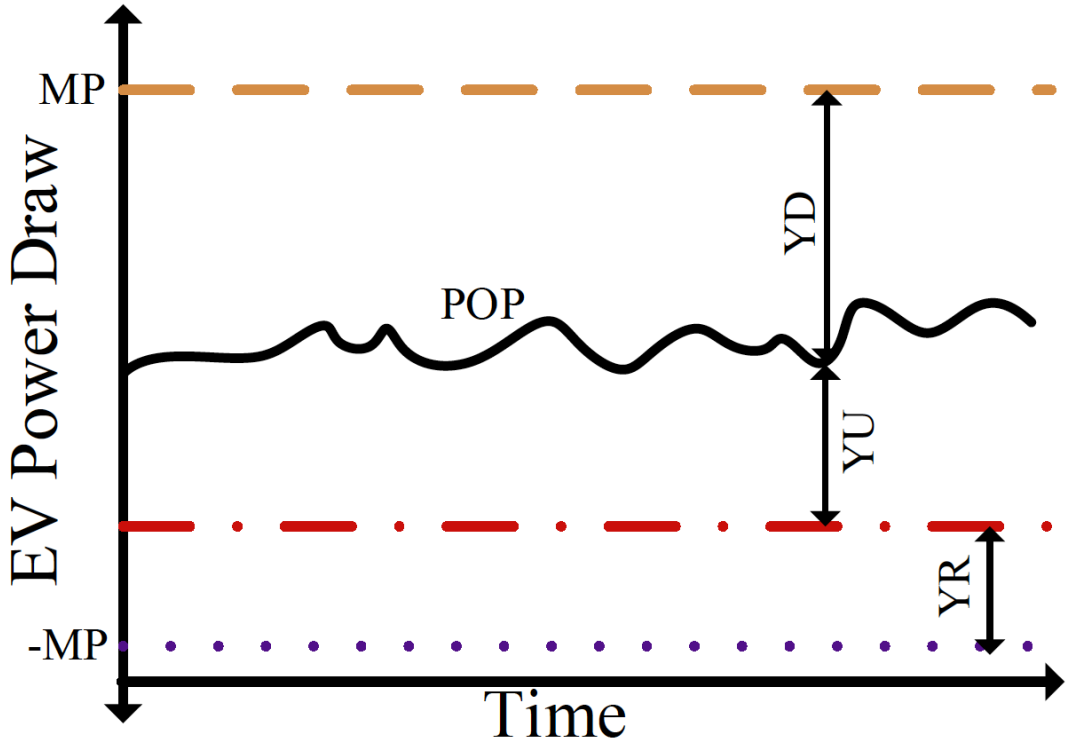
Non è possibile visualizzare l'immagine

Creation of the model and definition of the parameters

Parameter	Description	
Arrival	Daily arrival time of every EV	DRIVING PROFILES
T_Trip	Daily departure time of every EV	
EVPer	Expected percentage of the EVs remaining to perform V2G	
Dep	Probability of unexpected departure of the EV	
Comp	Compensation factor to account for unplanned departures	
SOC	Initial state of charge of the EV	
Ef	Efficiency of the EV's battery charger	EVs and CHARGERS
Mc	Maximum charge capacity of the EV	
DC	Degradation cost of the battery for discharging	
MP	Maximum possible power draw of the EV (if not plugged, this value is 0)	
Price_RD	Price of regulation down	PRICE SIGNALS
Price_RU	Price of regulation up	
P	Electricity price	

```
#CREATE MODEL
model = AbstractModel()

#DEFINE PARAMETERS
model.time = Param(within=NonNegativeIntegers)
model.cars = Param(within=NonNegativeIntegers)
model.days = Param(within=NonNegativeIntegers)

model.Arrival = Param(model.I, model.J, within=NonNegativeIntegers)
model.T_Trip = Param(model.I, model.J, within=NonNegativeIntegers)

model.EVPer = Param(model.J, model.T, within = NonNegativeReals)
model.Dep = Param(model.I, model.J, model.T, within = NonNegativeReals)
model.Comp = Param(model.I, model.J, model.T, within = NonNegativeReals)

model.SOC = Param(model.I, model.J, within = NonNegativeReals)


#Technical specifications of cars and chargers
model.Ef = Param(model.I, within = NonNegativeReals)
model.Mc = Param(model.I, within = NonNegativeReals)

model.DC = Param(model.I, within = NonNegativeReals)

model.MP = Param(model.I, model.J, model.T)


#Prices (electricity + ancilarry services)
model.Price_RD = Param(model.J, model.T, within = NonNegativeReals)
model.Price_RU = Param(model.J, model.T, within = NonNegativeReals)
model.P = Param(model.J, model.T, within = NonNegativeReals)

#DEFINE ITERATORS
model.T = RangeSet(1, model.time)
model.I = RangeSet(1, model.cars)
model.J = RangeSet(1, model.days)
```

Decision variables




```
#DATA STRUCTURES - DECISION VARIABLES
POP_dict = {}
for i in range(1, number_EVs+1):
    for j in range(1, num_days+1):
        for t in range(1, final_time+1):
            POP_dict[i, j, t] = 0
            #print(i, j, t, POP_dict[i,j,t])

MxAP_dict = {}
for i in range(1, number_EVs+1):
    for j in range(1, num_days+1):
        for t in range(1, final_time+1):
            MxAP_dict[i, j, t] = 0
            #print(i, j, t, MxAP_dict[i,j,t])

MnAP_dict = {}
for i in range(1, number_EVs+1):
    for j in range(1, num_days+1):
        for t in range(1, final_time+1):
            MnAP_dict[i, j, t] = 0
            #print(i, j, t, MnAP_dict[i,j,t])

Deg_dict = {}
for i in range(1, number_EVs+1):
    for j in range(1, num_days+1):
        for t in range(1, final_time+1):
            Deg_dict[i, j, t] = 0
            #print(i, j, t, Deg_dict[i,j,t])


#DEFINE DECISION VARIABLES
model.POP = Var(model.I, model.J, model.T, domain=Reals, initialize = POP_dict)
model.MxAP = Var(model.I, model.J, model.T, domain=NonNegativeReals, initialize
model.MnAP = Var(model.I, model.J, model.T, domain=NonNegativeReals, initialize
model.Deg = Var(model.I, model.J, model.T, domain=NonNegativeReals, initialize
```

Objective function

Objective function's equations term	Definition
$\sum_t (P_{RU}(t) \cdot R_U(t) + P_{RD}(t) \cdot R_D(t) + P_{RR}(t) \cdot R_R(t))$	Revenues from selling regulation and responsive services
$+ Mk \sum_i \sum_t (E[FP_i(t)])$	Fixed rate on the energy delivered to the EV
$+ \sum_i \sum_t (E[FP_i(t)] \cdot P(t))$ if $E[FP_i(t)] \leq 0$	Revenues from selling energy to the grid

INCOME

$\sum_i \sum_t (E[FP_i(t)] \cdot P(t))$	Wholesale cost of energy delivered to the EVs
$+ \sum_i \sum_t \left(\frac{DC_i \cdot E[FP_i^-(t)]}{Ef_i} \right)$	Battery degradation associated from discharging

COSTS

```
#OBJECTIVE FUNCTION
def income(model):
    return sum(sum((model.Price_RD[j,t] * sum(model.MxAP[i,j,t] for i in model.I) + model.Price_RU[j,t]
                    * sum(model.MnAP[i,j,t] for i in model.I)) * model.EVPer[j,t] for t in model.T) for j in model.J)
                    + Mk*sum(sum((model.MxAP[i,j,t] * ExD + model.POP[i,j,t] - model.MnAP[i,j,t] * ExU)
                    * model.EVPer[j,t] for t in model.T) for j in model.J) for i in model.I)

def costs(model):
    return sum(sum(sum((model.MxAP[i,j,t] * ExD + model.POP[i,j,t] - model.MnAP[i,j,t] * ExU)
                       * model.EVPer[j,t] * model.P[j,t] for t in model.T) for j in model.J) for i in model.I)
                       + sum(sum(sum(model.Deg[i,j,t] for t in model.T) for j in model.J) for i in model.I)

#Final equation
def profit(model):
    return income(model) - costs(model)
model.obj = Objective(rule = profit, sense = maximize)
```

Operational constraints

```python
#OPERATIONAL CONSTRAINTS
def fifteen_rule(model, i):
    for j in range(1, num_days+1):
        for k in range(model.Arrival[i,j], model.T_Trip[i,j]):
            return sum((model.MxAP[i,j,t] * ExD + model.POP[i,j,t] - model.MnAP[i,j,t] * ExU) * model.Comp[i,j,t]
                       + (model.Deg[i,j,t]/model.DC[i])*(1-model.Ef[i]*model.Ef[i])/model.Ef[i]
                       for t in range(model.Arrival[i,j], model.T_Trip[i,j])[:k-model.Arrival[i,j]+1]) * model.Ef[i]
                       + model.SOC[i,j] <= model.Mc[i]
model.fifteen = Constraint(model.I, rule=fifteen_rule)


def sixteen_rule(model, i):
    for j in range(1, num_days+1):
        for k in range(model.Arrival[i,j], model.T_Trip[i,j]):
            return sum((model.MxAP[i,j,t] * ExD + model.POP[i,j,t] - model.MnAP[i,j,t] * ExU) * model.Comp[i,j,t]
                       + (model.Deg[i,j,t]/model.DC[i])*(1-model.Ef[i]*model.Ef[i])/model.Ef[i]
                       for t in range(model.Arrival[i,j], model.T_Trip[i,j])[:k-model.Arrival[i,j]+1]) * model.Ef[i]
                       + model.SOC[i,j] >= 0
model.sixteen = Constraint(model.I, rule=sixteen_rule)


def seventeen_rule(model, i):
    for j in range(1, num_days+1):
        return sum((model.MxAP[i,j,t] * ExD + model.POP[i,j,t] - model.MnAP[i,j,t] * ExU) * model.Comp[i,j,t]
                   + (model.Deg[i,j,t]/model.DC[i])*(1-model.Ef[i]*model.Ef[i])/model.Ef[i]
                   for t in range(model.Arrival[i,j], model.T_Trip[i,j])) + model.SOC[i,j] >= 0.99 * model.Mc[i]
model.seventeen = Constraint(model.I, rule=seventeen_rule)


def eighteen_rule(model, i):
    for j in range(1, num_days+1):
        return (model.MxAP[i,j,model.Arrival[i,j]] + model.POP[i,j,model.Arrival[i,j]])
        * model.Comp[i,j,model.Arrival[i,j]] * model.Ef[i] + model.SOC[i,j] <= model.Mc[i]
model.eighteen = Constraint(model.I, rule=eighteen_rule)


def nineteen_rule(model, i):
    for j in range(1, num_days+1):
        return (model.POP[i,j,model.Arrival[i,j]] - model.MnAP[i,j,model.Arrival[i,j]])
        + (model.Deg[i,j,model.Arrival[i,j]]/model.DC[i])*(1-model.Ef[i]*model.Ef[i])/model.Ef[i] *
        model.Comp[i,j,model.Arrival[i,j]] * model.Ef[i] + model.SOC[i,j] >= 0
model.nineteen = Constraint(model.I, rule=nineteen_rule)
```

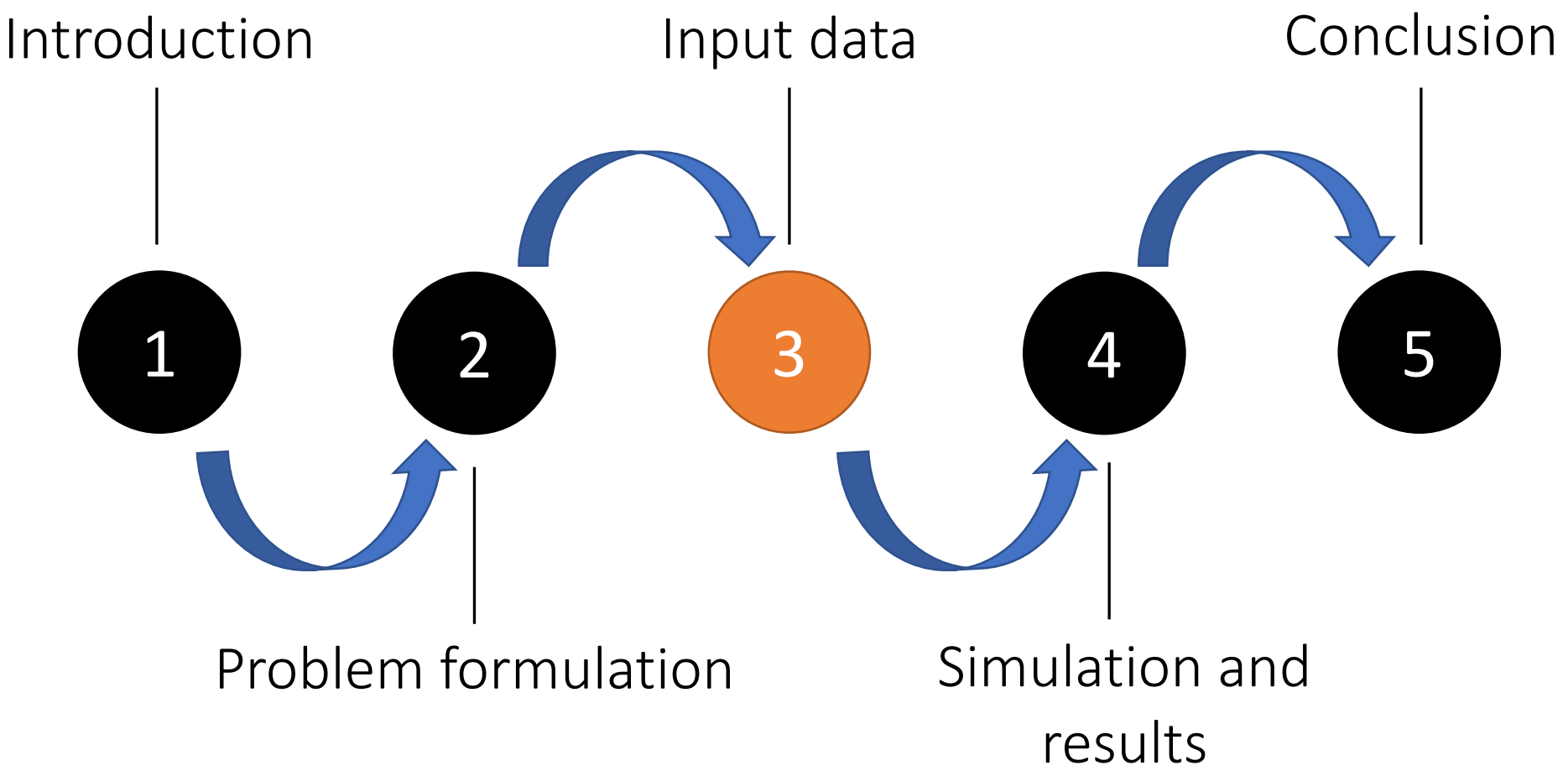
Operational constraint	Description
$\left(\sum_{t=1}^{time}(E[FP_i(t)]\cdot Comp_i(t)+\rho_i(t))\cdot Ef_i + SOC_{I,i} - Trip_i(time)\right) \leq M_{Ci}$ $\left(\sum_{t=1}^{time}(E[FP_i(t)]\cdot Comp_i(t)+\rho_i(t))\cdot Ef_i + SOC_{I,i} - Trip_i(time)\right) \geq 0$ $\forall i, time$ $\left(\sum_{t=1}^{T}(E[FP_i(t)]\cdot Comp_i(t)+\rho_i(t))\cdot Ef_i + SOC_{I,i} - Trip_i(T)\right) \geq 0.99 M_{Ci}$	The energy level of the battery is bounded between 0 and its maximum capacity at every moment of the daily scheduling and it is set to be at least 99% of its maximum capacity at the end of the daily scheduling
$(MxAP_i(1)+POP_i(1))\cdot Comp_i(1)\cdot Ef_i + SOC_{I,i} \leq M_{Ci}$ $\left((POP_i(1) - MnAP_i(1) - RsRP_i(1) + \rho_i(1)) \cdot Comp_i(1)\cdot Ef_i + SOC_{I,i}\right) \geq 0$	Conditions on the arrival times of the electric vehicles
$(MxAP_i(t) + POP_i(t))\cdot Comp_i(t) \leq MP_i(t)\ \forall i$ $MnAP_i(t) \leq POP_i(t) + MP_i(t)\ \forall i$	Ancillary services capacities cannot exceed the admissible limits of the maximum possible power draw of the charger
$MxAP_i(t) \geq 0\ \forall i$ $MnAP_i(t) \geq 0\ \forall i$ $POP_i(t) \geq -MP_i(t)\ \forall i$ $Deg_i(t) \geq 0\ \forall i$	Bounds of the decision variables
$Deg_i(t) \geq DC_i \cdot E[FP_i^-(t)] \cdot \frac{Comp_i(t)}{Ef_i}\ \forall i$	Condition on the degradation of the battery

Operational constraints

```python
def twenty_one_rule(model, i, j, t):
    return (model.MxAP[i,j,t] + model.POP[i,j,t]) * model.Comp[i,j,t] <= model.MP[i,j,t]
model.twenty_one = Constraint(model.I, model.J, model.T, rule=twenty_one_rule)


def twenty_two_rule(model, i, j, t):
    return model.MnAP[i,j,t] <= model.POP[i,j,t] + model.MP[i,j,t]
model.twenty_two = Constraint(model.I, model.J, model.T, rule=twenty_two_rule)



def twenty_four_rule(model, i, j, t):
    return model.MxAP[i,j,t] >= 0
model.twenty_four = Constraint(model.I, model.J, model.T, rule=twenty_four_rule)


def twenty_five_rule(model, i, j, t):
    return model.MnAP[i,j,t] >= 0
model.twenty_five = Constraint(model.I, model.J, model.T, rule=twenty_five_rule)



def twenty_seven_rule(model, i, j, t):
    return model.POP[i,j,t] >= -model.MP[i,j,t]
model.twenty_seven = Constraint(model.I, model.J, model.T, rule=twenty_seven_rule)


def twenty_eight_rule(model, i, j, t):
    return model.Deg[i,j,t] >= 0
model.twenty_eight = Constraint(model.I, model.J, model.T, rule=twenty_eight_rule)


def twenty_nine_rule(model, i, j, t):
    return model.Deg[i,j,t] >= model.DC[i] * (model.POP[i,j,t] - model.MnAP[i,j,t] * ExU)
                               * model.Comp[i,j,t]/model.Ef[i]
model.twenty_nine = Constraint(model.I, model.J, model.T, rule=twenty_nine_rule)
```

Operational constraint	Description
$\left(\sum_{t=1}^{time}(E[FP_i(t)]\cdot Comp_i(t)+\rho_i(t))\cdot Ef_i + SOC_{I,i} - Trip_i(time)\right) \leq M_{Ci}$ $\left(\sum_{t=1}^{time}(E[FP_i(t)]\cdot Comp_i(t)+\rho_i(t))\cdot Ef_i + SOC_{I,i} - Trip_i(time)\right) \geq 0$ $\forall i, time$ $\left(\sum_{t=1}^{T}(E[FP_i(t)]\cdot Comp_i(t)+\rho_i(t))\cdot Ef_i + SOC_{I,i} - Trip_i(T)\right) \geq 0.99 M_{Ci}$	The energy level of the battery is bounded between 0 and its maximum capacity at every moment of the daily scheduling and it is set to be at least 99% of its maximum capacity at the end of the daily scheduling
$(MxAP_i(1)+POP_i(1))\cdot Comp_i(1)\cdot Ef_i + SOC_{I,i} \leq M_{Ci}$ $\left((POP_i(1) - MnAP_i(1) - RsRP_i(1) + \rho_i(1)) \cdot Comp_i(1)\cdot Ef_i + SOC_{I,i}\right) \geq 0$	Conditions on the arrival times of the electric vehicles
$(MxAP_i(t) + POP_i(t))\cdot Comp_i(t) \leq MP_i(t)\ \forall i$ $MnAP_i(t) \leq POP_i(t) + MP_i(t)\ \forall i$	Ancillary services capacities cannot exceed the admissible limits of the maximum possible power draw of the charger
$MxAP_i(t) \geq 0\ \forall i$ $MnAP_i(t) \geq 0\ \forall i$ $POP_i(t) \geq -MP_i(t)\ \forall i$ $Deg_i(t) \geq 0\ \forall i$	Bounds of the decision variables
$Deg_i(t) \geq DC_i \cdot E[FP_i^{-}(t)] \cdot \frac{Comp_i(t)}{Ef_i}\ \forall i$	Condition on the degradation of the battery

Agenda


Introduction
Input data
Conclusion
1
2
3
4
5
Problem formulation
Simulation and results

Driving profiles

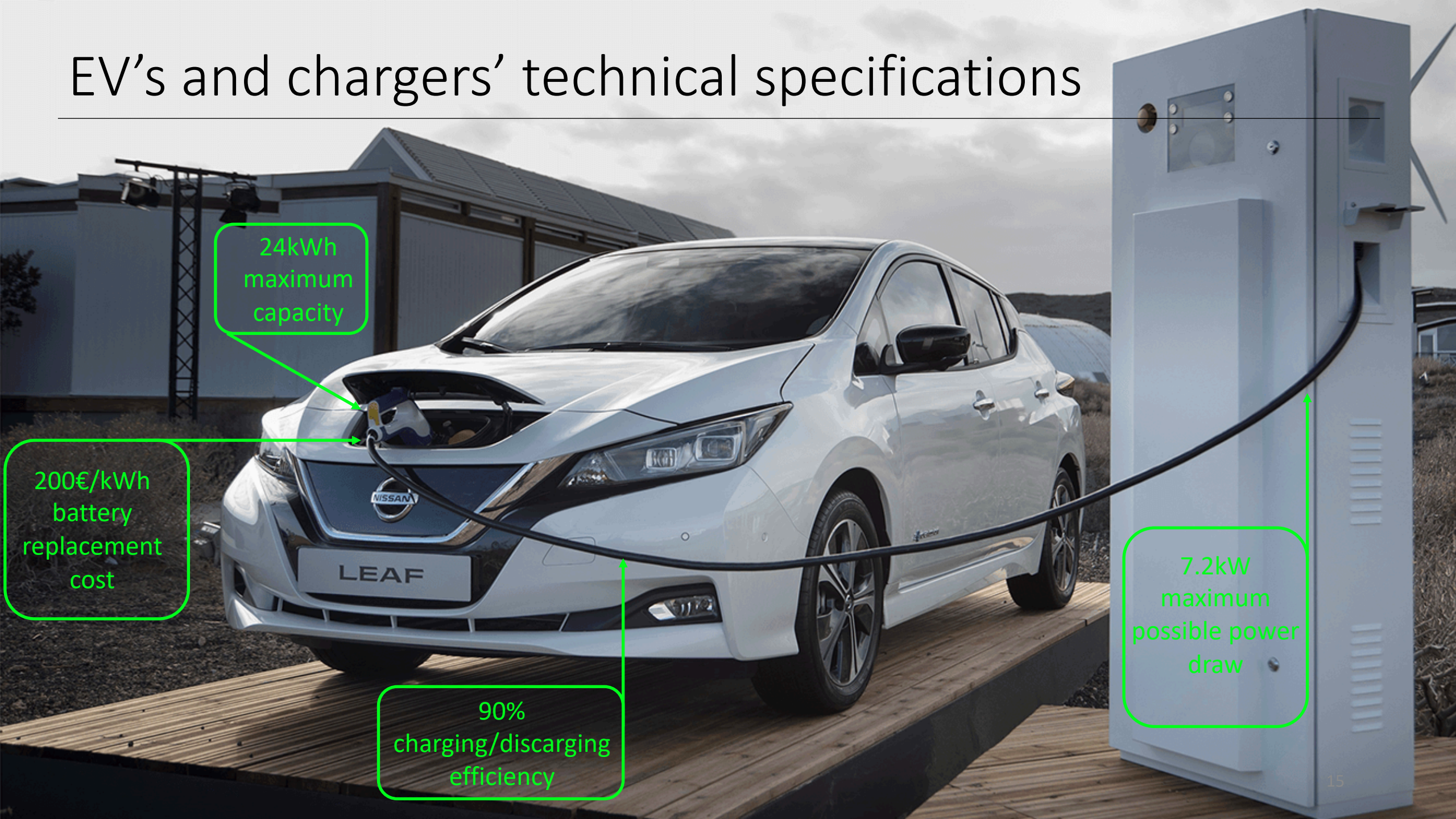
Hypothetical group of **100** EVs

People arrive at work every morning between **6AM** and **9AM** and come back home every evening between **4PM** and **7PM**

Battery's SOC in the morning is **randomly generated**

	6AM	7AM	8AM	9AM	10AM	11AM	12AM	1PM	2PM	3PM	4PM	5PM	6PM	7PM
EV 1														
EV 2														
EV 3														
⋮														
EV 100														

EV's and chargers' technical specifications



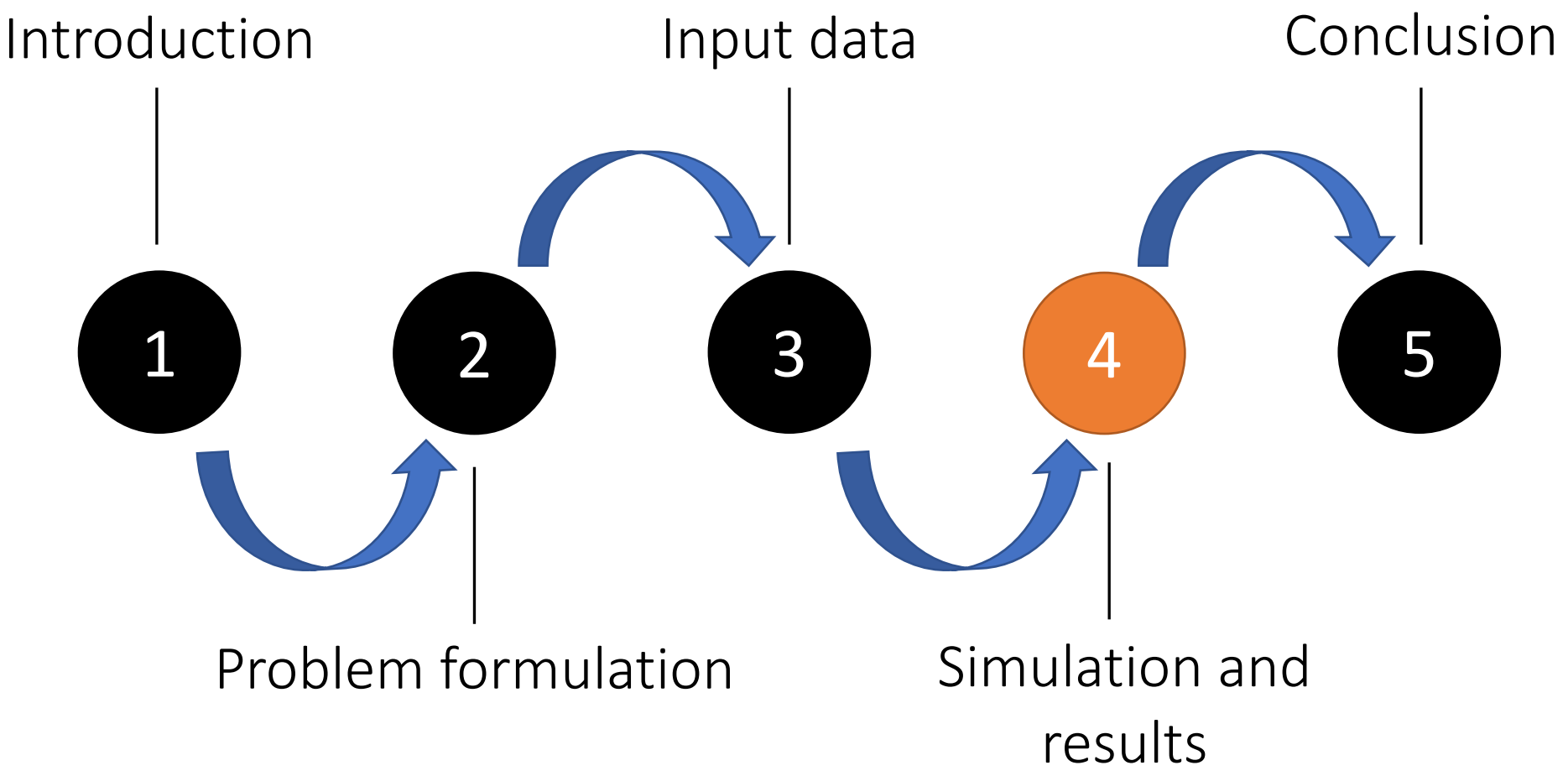
Price signals

Ancillary services and electricity prices found in **Terna public archives** and **GME repository**

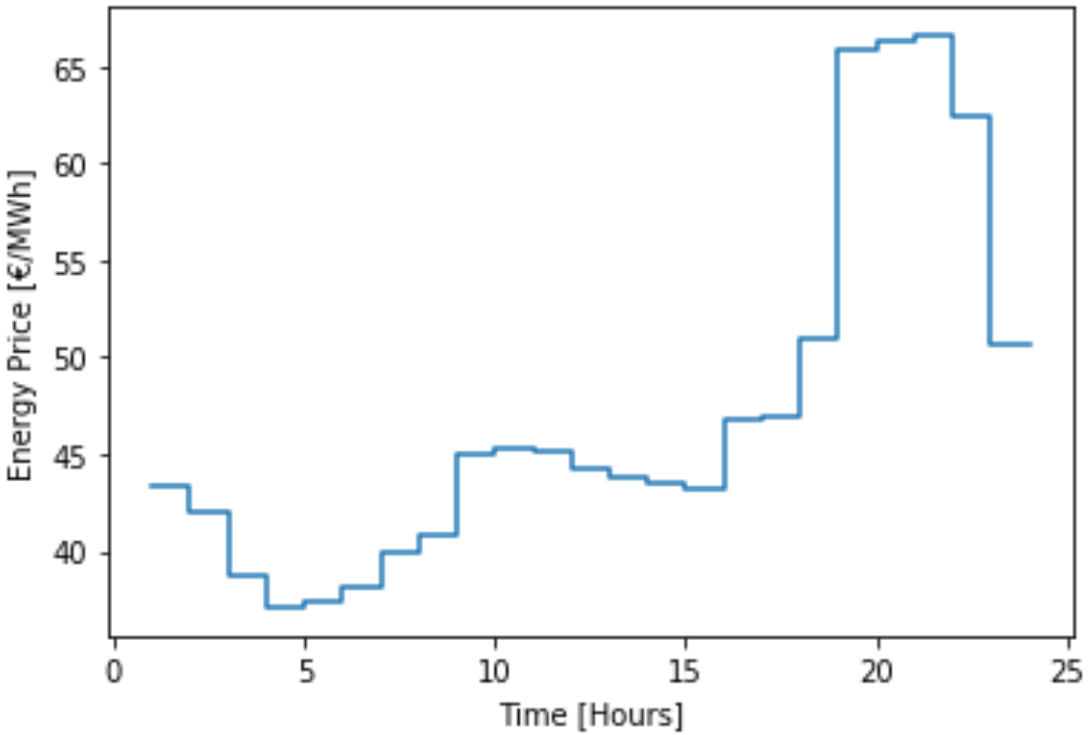
Simulation period of **three months**, from August 2019 to October 2019

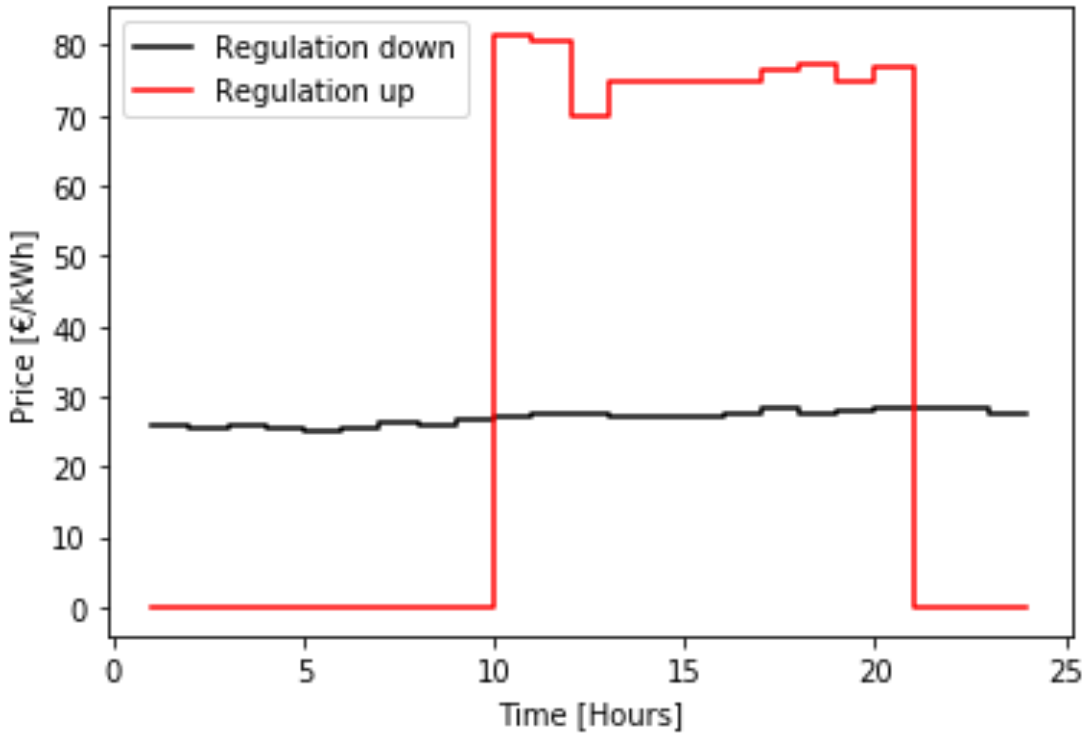
Aggregator rate of energy charged to the consumer has fixed at **€0.01/kWh**

Agenda

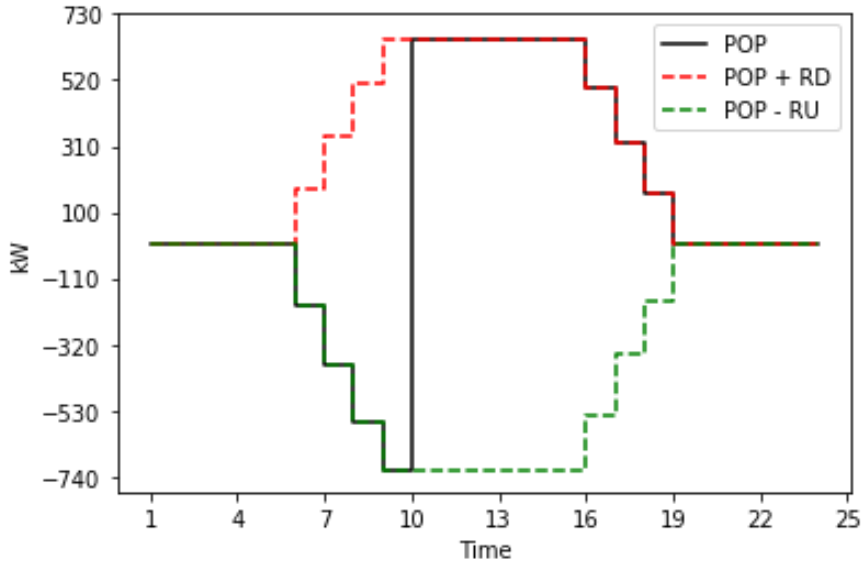

Introduction
Input data
Conclusion
1
2
3
4
5
Problem formulation
Simulation and results

Energy and Ancillary Services Prices



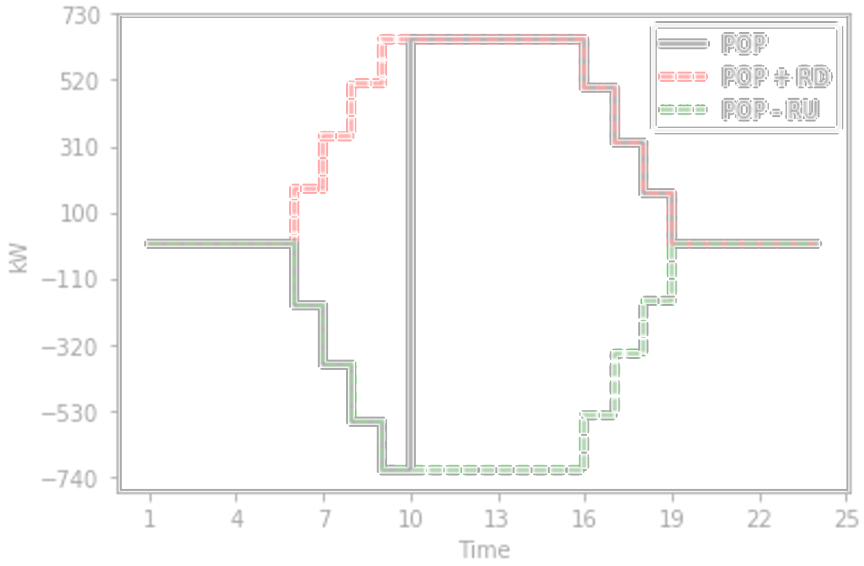


V2G and Smart Charging algorithms



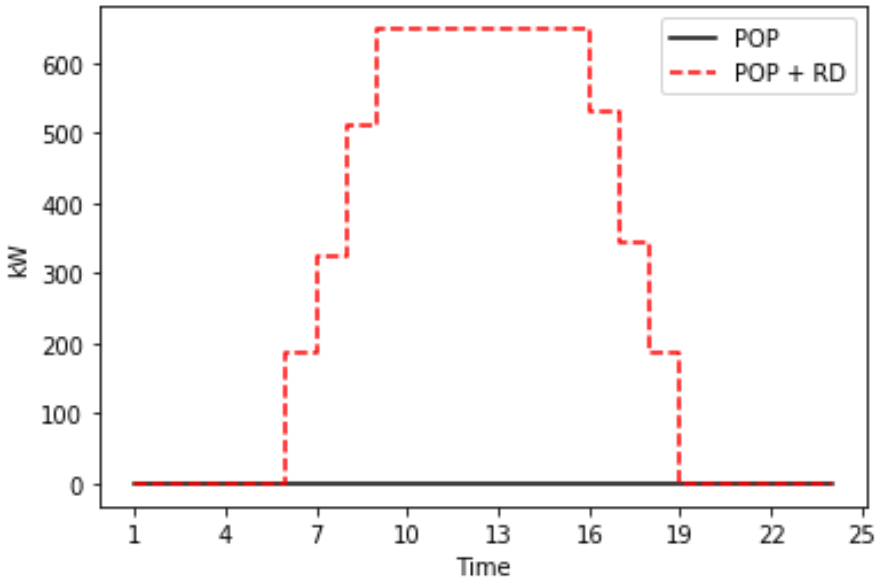
- Vehicle discharge until **10AM** and charge during the second part of the day
- Charging process always happens at the **maximum** of his capability
- The variables stay at zero **before the arrival** of the EVs and **after their departure**

V2G and Smart Charging algorithms

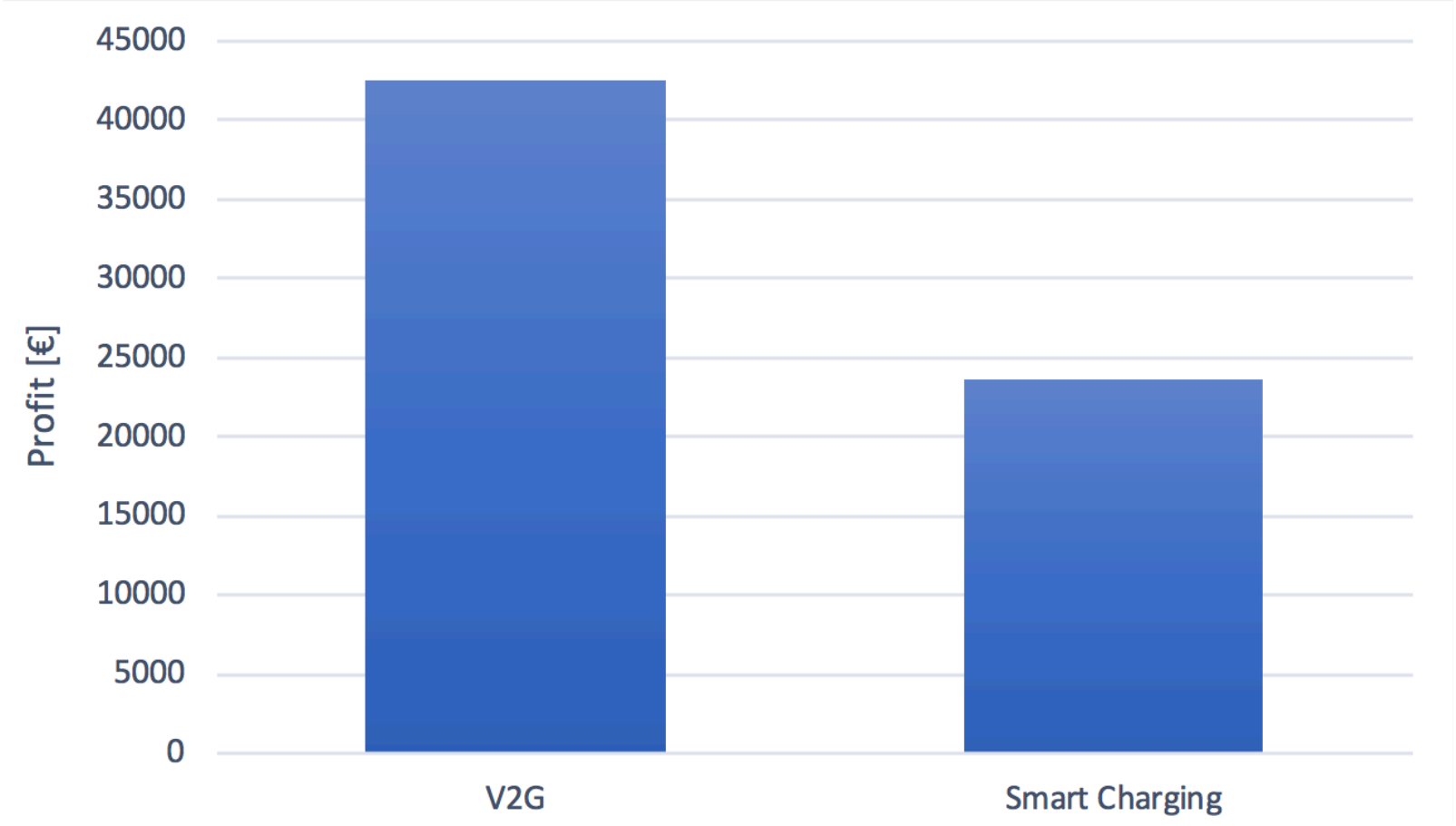


- Vehicle discharge until **10AM** and charge during the second part of the day
- Charging process always happens at the **maximum** of his capability
- The variables stay at zero **before the arrival** of the EVs and **after their departure**

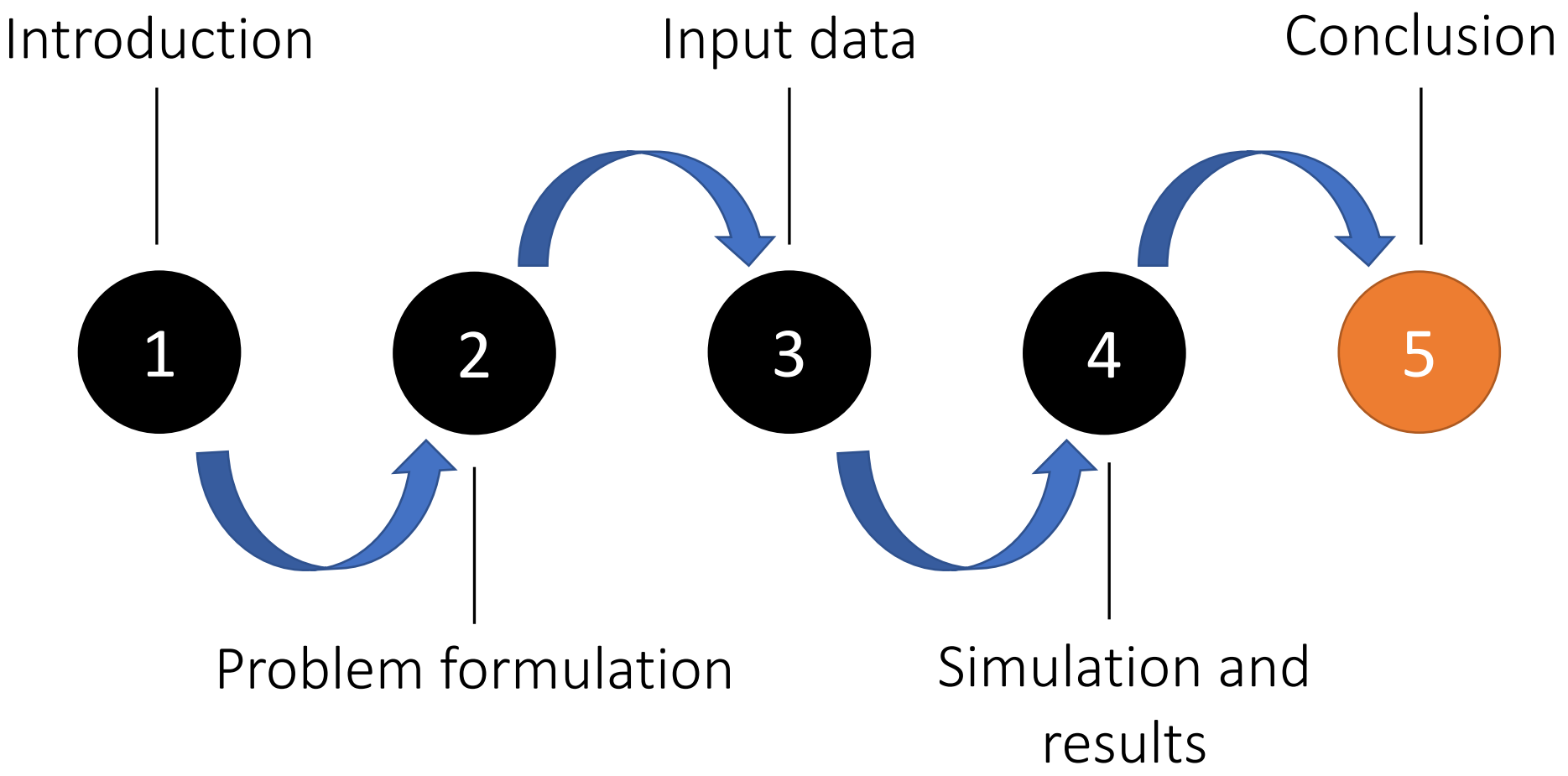
- Decision variables related to **regulation up** (and its prices) and **degradation of the battery** are not taken into account
- Charging of the fleet **doesn't need to be altered** during that day
- Capacity of regulation down **increases** as people arrive to connect their vehicles.



Profits comparison



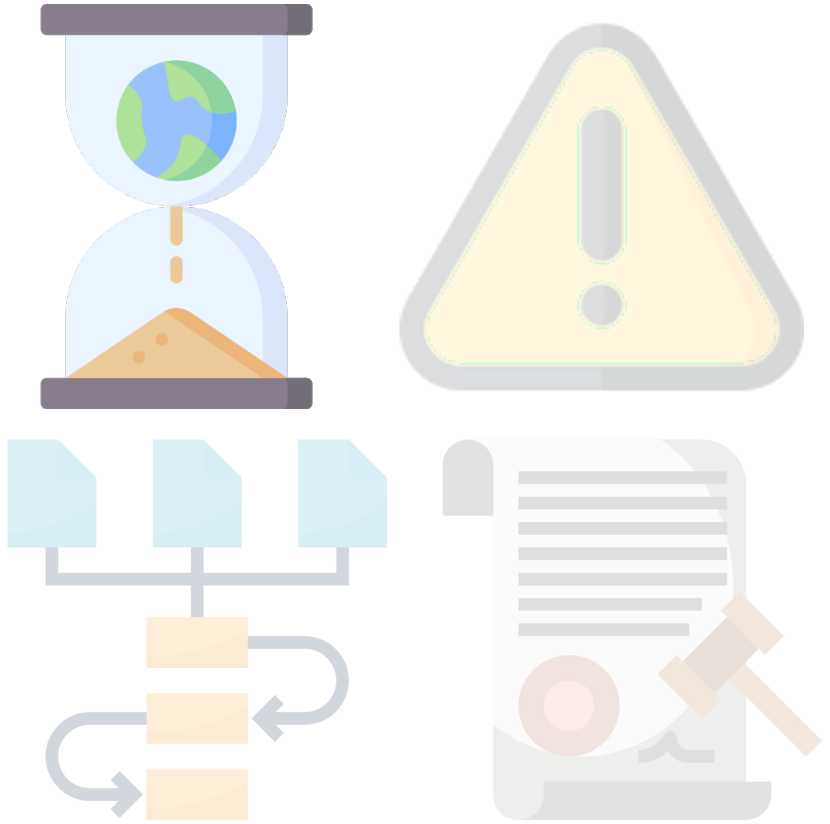
Agenda


Introduction
Input data
Conclusion
1
2
3
4
5
Problem formulation
Simulation and results

Conclusion

Conclusion

EXPONENTIAL EXECUTION TIME



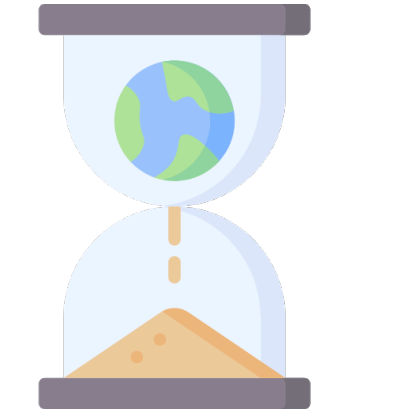
Conclusion

EXPONENTIAL EXECUTION TIME

PROBLEMS WITH THE SMART CHARGING OPTIMIZATION

Conclusion

EXPONENTIAL EXECUTION TIME

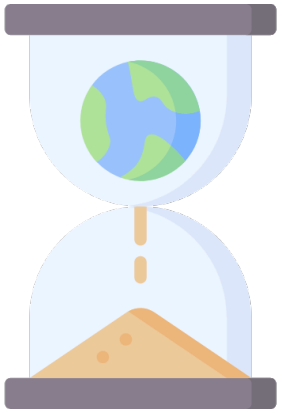


PROBLEMS WITH THE SMART CHARGING OPTIMIZATION

ELECTRICITY MARKET NEEDS TO BE MODELED PROPERLY

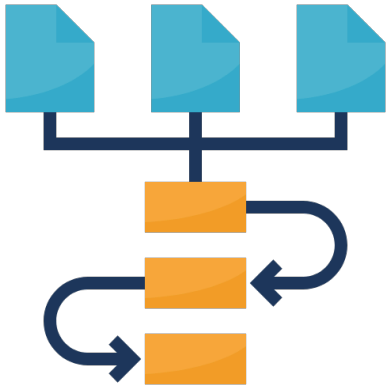
Conclusion

EXPONENTIAL EXECUTION TIME



PROBLEMS WITH THE SMART CHARGING OPTIMIZATION

REAL-TIME ANCILLARY SERVICES ALGORITHM



ELECTRICITY MARKET NEEDS TO BE MODELED PROPERLY

THANK YOU!
Q&A

References

- Eric Sortomme and Mohamed A. El-Sharkawi, *Optimal Scheduling of Vehicle-to-Grid Energy and Ancillary Services,* IEEE Transactions on smart grid, VOL. 3, NO. 1, (2012)