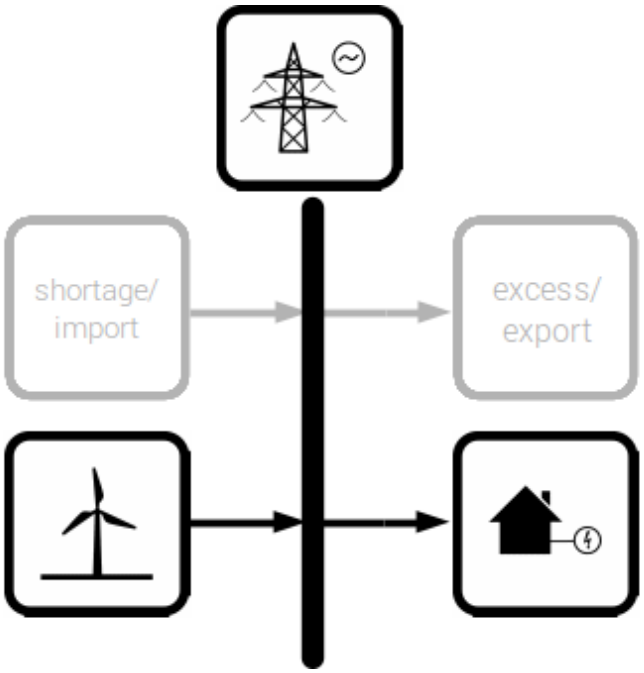
sector_coupling

May 14, 2017

1 Multisectoral energy system with oemof

1.1 Create a simple energy system



Simple energy system

1.1.1 Initialize energy system

```
In [1]: from oemof.solph import EnergySystem
        import pandas as pd

        # initialize energy system
        energysystem = EnergySystem(timeindex=pd.date_range('1/1/2016',
                                                            periods=168,
                                                            freq='H'))
```

1.1.2 Import input data

In [2]:
```
# import example data with scaled demands and feedin timeseries of renewables
# as dataframe
data = pd.read_csv("example_data.csv", sep=",")
#print(data.demand_el[0:10])
#print(data.keys())
```

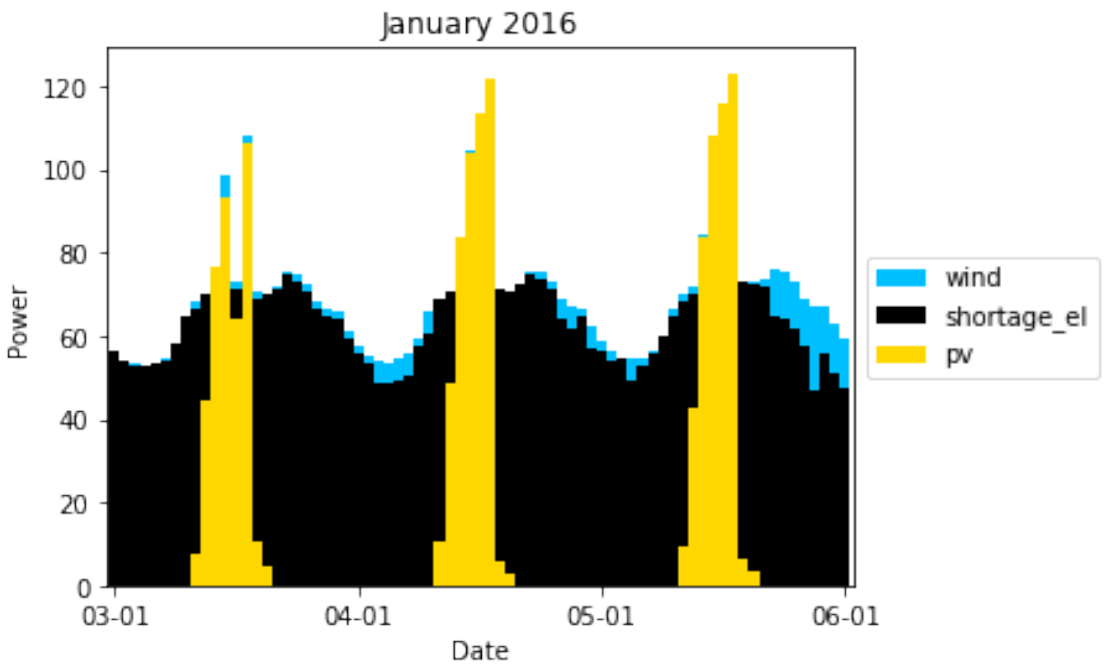
1.1.3 Add entities to energy system

In [3]:
```
from oemof.solph import Bus, Flow, Sink, Source, LinearTransformer

### BUS
# create electricity bus
b_el = Bus(label="b_el")

# add excess sink to help avoid infeasible problems
Sink(label="excess_el",
     inputs={b_el: Flow()})
Source(label="shortage_el",
       outputs={b_el: Flow(variable_costs=1000)})

### DEMAND
# add electricity demand
Sink(label="demand_el",
     inputs={b_el: Flow(nominal_value=85,
                        actual_value=data['demand_el'],
                        fixed=True)})

### SUPPLY
# add wind and pv feedin
Source(label="wind",
       outputs={b_el: Flow(actual_value=data['wind'],
                           nominal_value=60,
                           fixed=True)});
Source(label="pv",
       outputs={b_el: Flow(actual_value=data['pv'],
                           nominal_value=200,
                           fixed=True)});
```

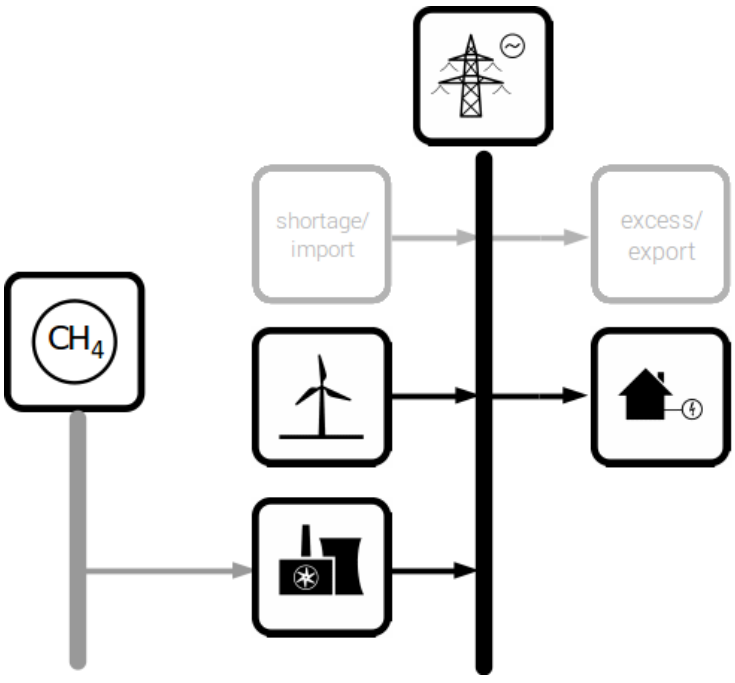
1.1.4 Optimize energy system and plot results

In [4]:
```
from oemof.solph import OperationalModel

import oemof.outputlib
import matplotlib.pyplot as plt

def optimize(energysystem):
```

```
    ### optimize
    # create operational model
    om = OperationalModel(es=energysystem)

    # solve using the cbc solver
    om.solve(solver='cbc',
             solve_kwargs={'tee': False})

    # save LP-file
    om.write('sector_coupling.lp', io_options={'symbolic_solver_labels': True})

    # write back results from optimization object to energysystem
    om.results();

def plot(energysystem, bus_label, bus_type):
    # define colors
    cdict = {'wind': '#00bfff', 'pv': '#ffd700', 'pp_gas': '#8b1a1a',
             'pp_chp_extraction': '#838b8b', 'excess_el': '#8b7355',
             'shortage_el': '#000000', 'heater_rod': 'darkblue',
             'pp_chp': 'green', 'demand_el': 'lightgreen', 'demand_th': '#ce4aff',
             'heat_pump': 'red', 'leaving_bev': 'darkred', 'bev_storage': 'orange'}

    # create multiindex dataframe with result values
    esplot = oemof.outputlib.DataFramePlot(energy_system=energysystem)

    # select input results of electrical bus (i.e. power delivered by plants)
    esplot.slice_unstacked(bus_label=bus_label, type=bus_type,
                           date_from='2016-01-03 00:00:00',
                           date_to='2016-01-06 00:00:00')

    # set colorlist for esplot
    colorlist = esplot.color_from_dict(cdict)

    # set plot attributes
    esplot.plot(color=colorlist, title="January 2016", stacked=True, width=1,
                kind='bar')
    esplot.ax.set_ylabel('Power')
    esplot.ax.set_xlabel('Date')
    esplot.set_datetime_ticks(tick_distance=24, date_format='%d-%m')
    esplot.outside_legend(reverse=True)

    plt.show()

optimize(energysystem)
plot(energysystem, "b_el", "to_bus")
```



1.2 Adding the gas sector



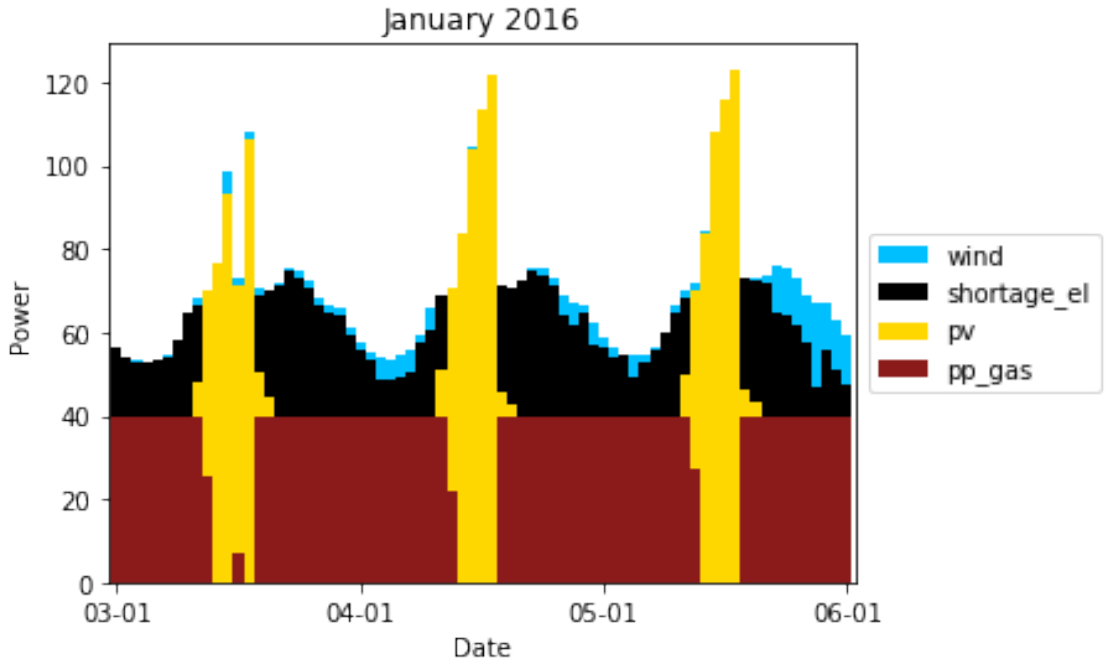
Energy system with gas power plant

In order to add a gas power plant, a gas ressource bus is needed. The gas power plant connects

the gas and electricity busses and thereby couples the gas and electricity sector.

```
In [5]: # add gas bus
        b_gas = Bus(label="b_gas",
                    balanced=False)

        # add gas power plant
        LinearTransformer(label="pp_gas",
                          inputs={b_gas: Flow(summed_max_flow=200)},
                          outputs={b_el: Flow(nominal_value=40,
                                              variable_costs=40)},
                          conversion_factors={b_el: 0.50});
```

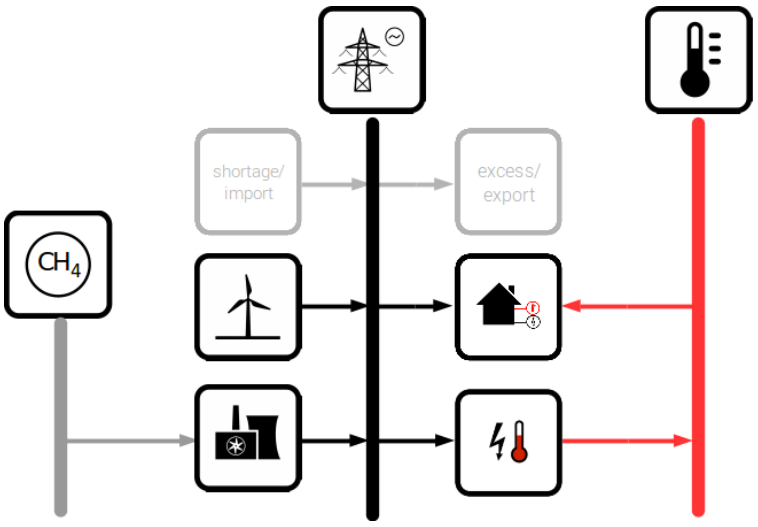
```
In [6]: optimize(energysystem)
        plot(energysystem, "b_el", "to_bus")
```



1.3 Adding the heat sector

The heat sector is added and coupled to the electricity sector similarly to the gas sector. The same component, the LinearTransformer, is used to couple the two sectors. Only through its parametrisation it becomes a heater rod or a heat pump.

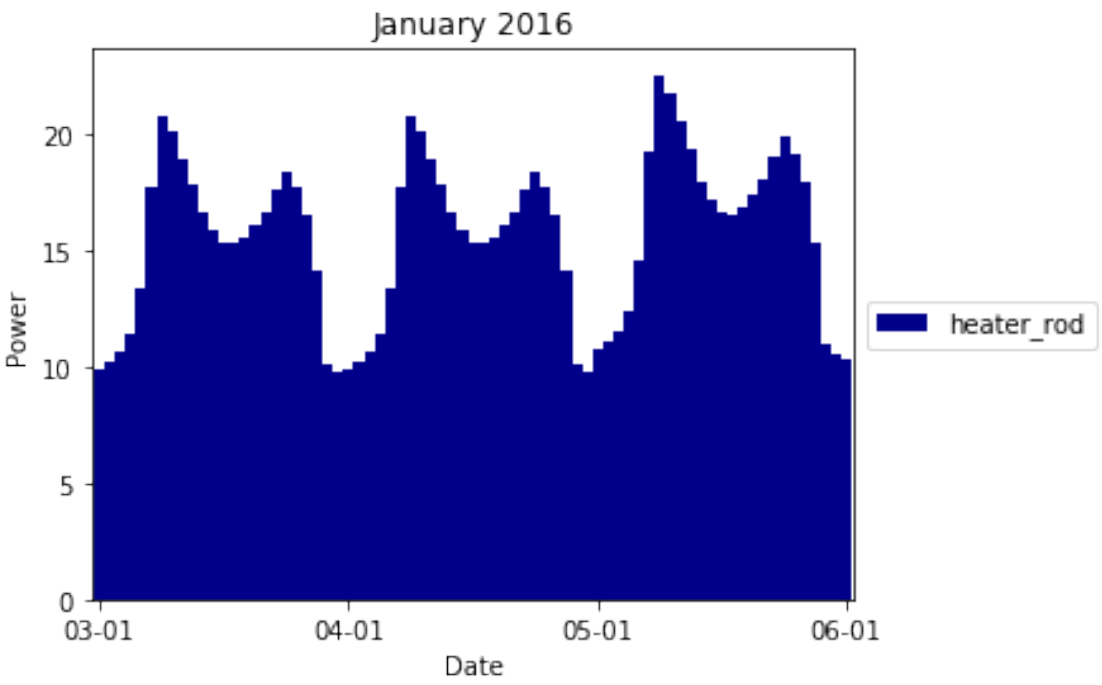
```
In [7]: # add heat bus
        b_heat = Bus(label="b_heat",
                     balanced=True)
```



Energy system with heat sector

```
        # add heat demand
        Sink(label="demand_th",
             inputs={b_heat: Flow(nominal_value=60,
                                  actual_value=data['demand_th'],
                                  fixed=True)})

        # add heater rod
        LinearTransformer(label="heater_rod",
                          inputs={b_el: Flow()},
                          outputs={b_heat: Flow(variable_costs=10)},
                          conversion_factors={b_heat: 0.98});
```

```
In [8]: optimize(energysystem)
        plot(energysystem, "b_heat", "to_bus")
```

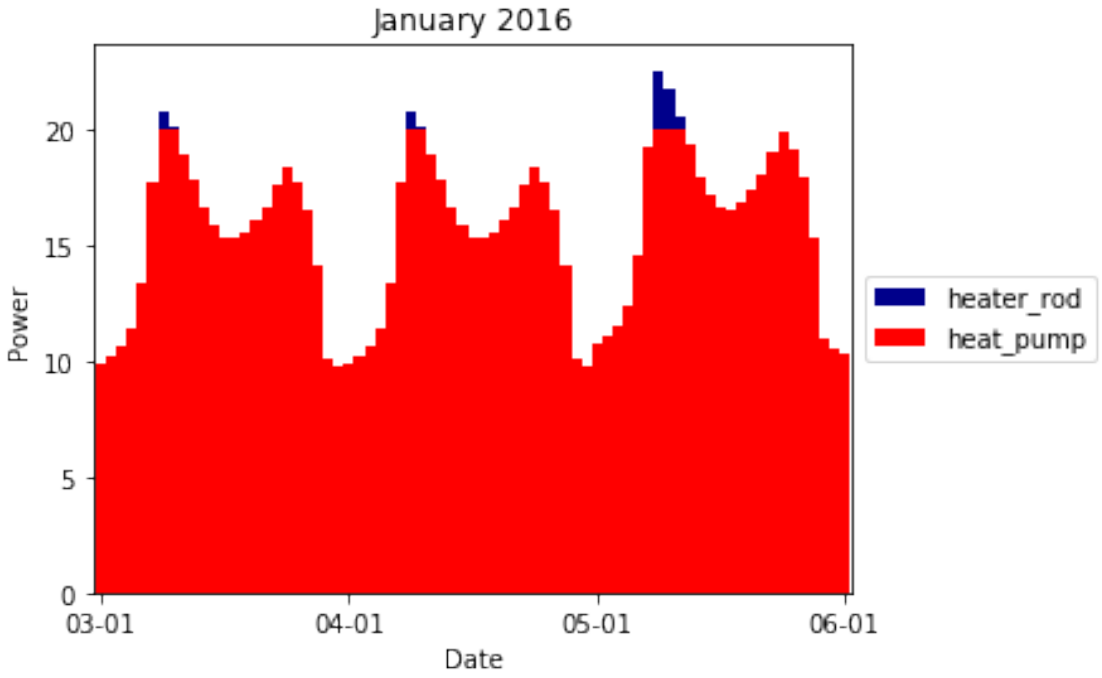


1.3.1 Adding a heat pump

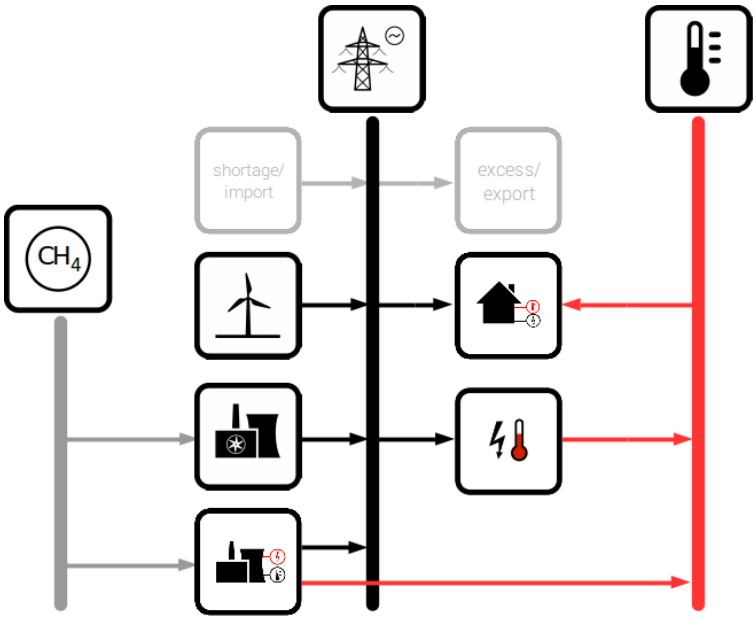
There are different ways to model a heat pump. Here the approach of precalculating a COP and using this as a conversion factor for the LinearTransformer is used. Another approach is to use the LinearN1Transformer that has two inputs - electricity and heat from a heat source. See the solph example "simple_dispatch".

```
In [9]: # COP can be calculated beforehand, assuming the heat reservoir temperature
        # is infinite random timeseries for COP
        import numpy as np
        COP = np.random.uniform(low=3.0, high=5.0, size=(168,))

        # add heater rod
        #LinearTransformer(label="heater_rod",
        #                  inputs={b_el: Flow()},
        #                  outputs={b_heat: Flow(variable_costs=10)},
        #                  conversion_factors={b_heat: 0.98});

        # add heat pump
        LinearTransformer(label="heat_pump",
                          inputs={b_el: Flow()},
                          outputs={b_heat: Flow(nominal_value=20,
                                                variable_costs=10)},
                          conversion_factors={b_heat: COP});
```

```
In [10]: optimize(energysystem)
         plot(energysystem, "b_heat", "to_bus")
```



1.3.2 Adding a combined heat and power plant



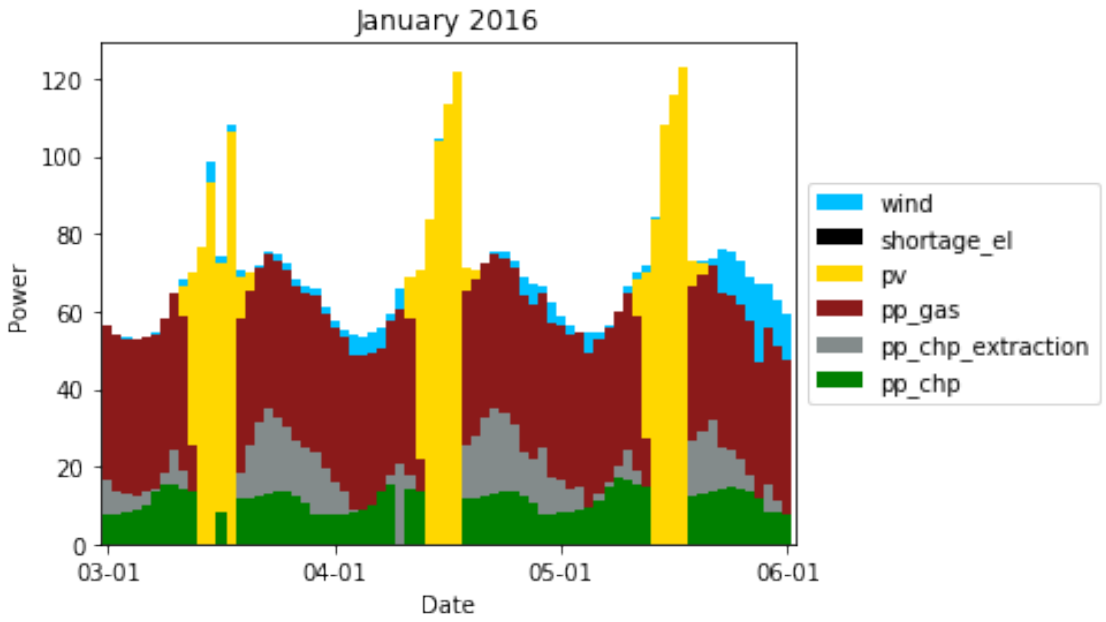
Energy system with CHP

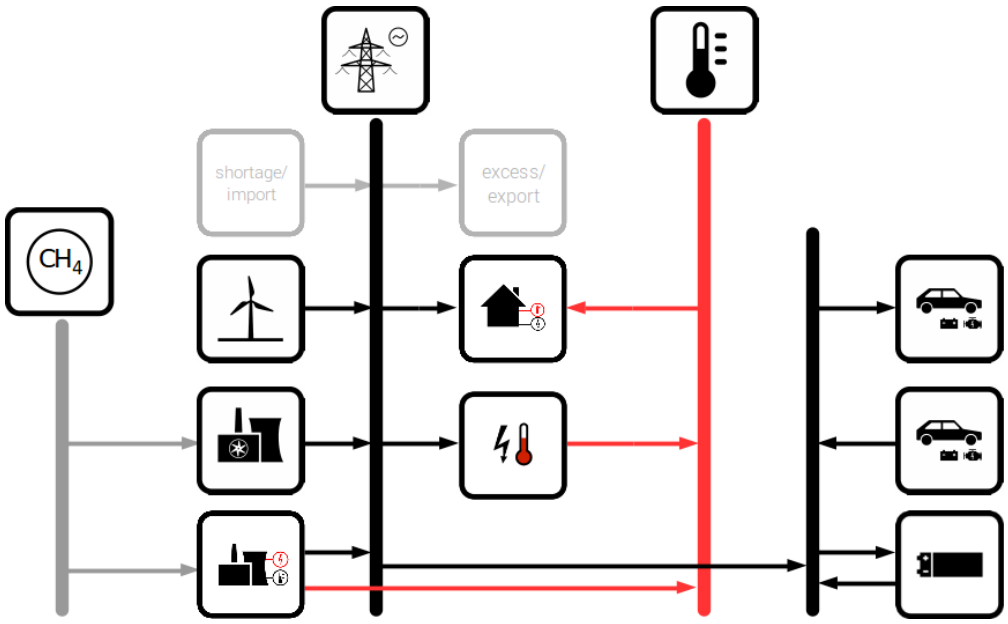
The combined heat and power plant couples the gas, electricity and heat sector.

```
In [11]: # add CHP with fixed ratio of heat and power (back-pressure turbine)
         LinearTransformer(label='pp_chp',
                           inputs={b_gas: Flow()},
                           outputs={b_el: Flow(nominal_value=30,
                                               variable_costs=42),
                                    b_heat: Flow(nominal_value=40)},
                           conversion_factors={b_el: 0.3,
                                               b_heat: 0.4});
```

```
In [12]: from oemof.solph import VariableFractionTransformer

         # add CHP with variable ratio of heat and power (extraction turbine)
         VariableFractionTransformer(label='pp_chp_extraction',
                                     inputs={b_gas: Flow()},
                                     outputs={b_el: Flow(nominal_value=30,
                                                         variable_costs=42),
                                              b_heat: Flow(nominal_value=40)},
                                     conversion_factors={b_el: 0.3,
                                                         b_heat: 0.4},
                                     conversion_factor_single_flow={b_el: 0.5});
```

```
In [13]: optimize(energysystem)
         plot(energysystem, "b_el", "to_bus")
```





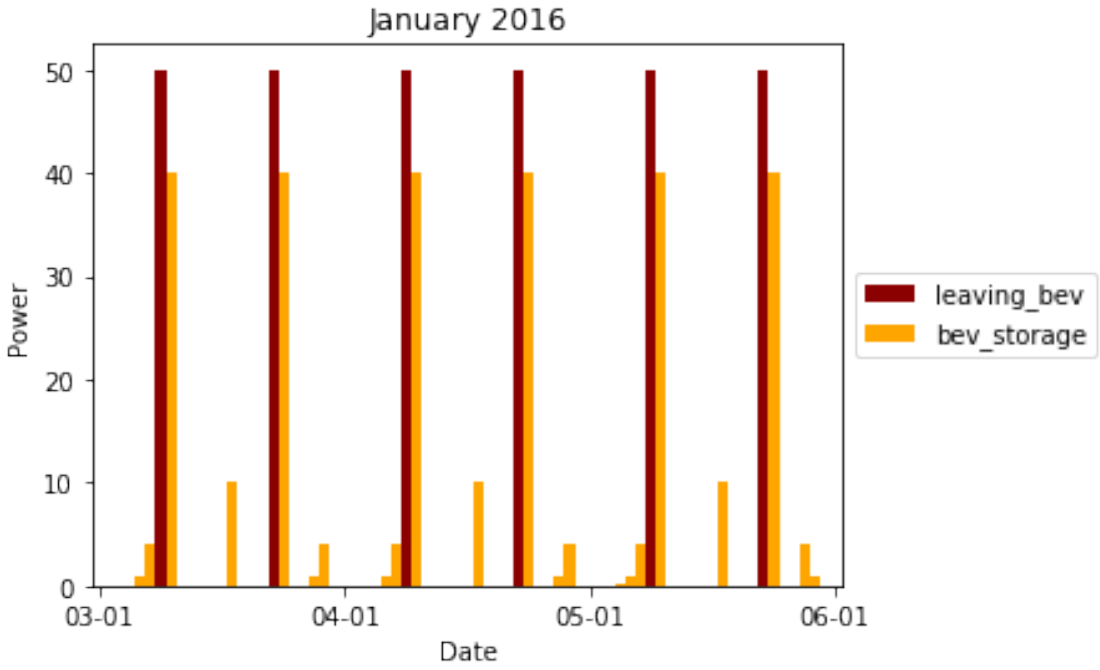
Energy system with mobility sector

1.4 Adding the mobility sector

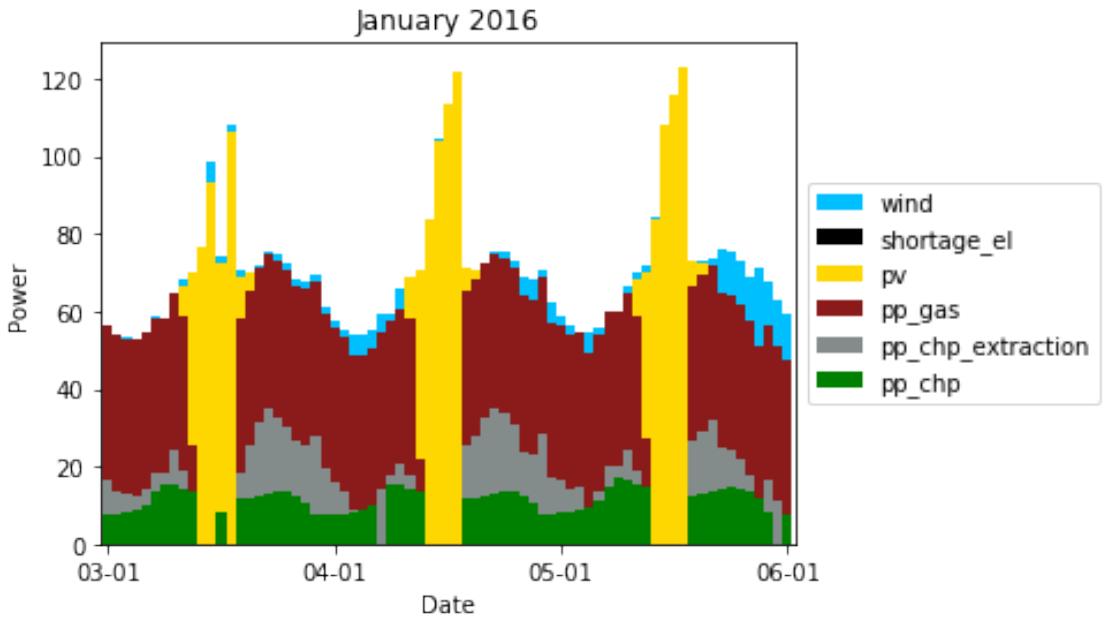
```
In [14]: from oemof.solph import Storage

         charging_power = 20
         bev_battery_cap = 50

         # add mobility bus
         b_bev = Bus(label="b_bev",
                     balanced=True)

         # add transformer to transport electricity from grid to mobility sector
         LinearTransformer(label="transport_el_bev",
                           inputs={b_el: Flow()},
                           outputs={b_bev: Flow(variable_costs=10,
                                                nominal_value=charging_power,
                                                max=data['bev_charging_power'])},
                           conversion_factors={b_bev: 1.0})

         # add BEV storage
         Storage(label='bev_storage',
                 inputs={b_bev: Flow()},
                 outputs={b_bev: Flow()},
                 nominal_capacity=bev_battery_cap,
                 capacity_min=data['bev_cap_min'],
                 capacity_max=data['bev_cap_max'],
                 capacity_loss=0.00,
                 initial_capacity=None,
```

```
                inflow_conversion_factor=1.0,
                outflow_conversion_factor=1.0,
                nominal_input_capacity_ratio=1.0,
                nominal_output_capacity_ratio=1.0,
                fixed_costs=35)

        # add sink for leaving vehicles
        Sink(label="leaving_bev",
             inputs={b_bev: Flow(nominal_value=bev_battery_cap,
                                 actual_value=data['bev_sink'],
                                 fixed=True)})

        # add source for returning vehicles
        Source(label="returning_bev",
               outputs={b_bev: Flow(nominal_value=bev_battery_cap,
                                    actual_value=data['bev_source'],
                                    fixed=True)});
```

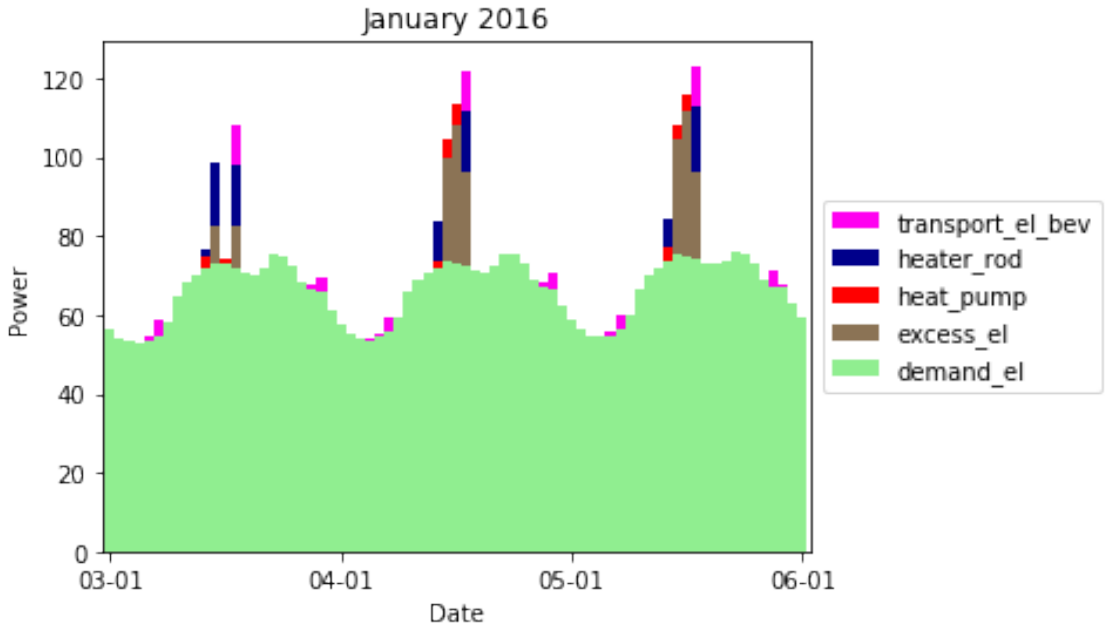
In [15]:
```
optimize(energysystem)
plot(energysystem, "b_bev", "from_bus")
```



In [16]:
```
plot(energysystem, "b_el", "to_bus")
```



In [17]: plot(energysystem, "b_el", "from_bus")



In [ ]: