

# An approach to cellular modelling with oemof-solph

# What is cellular modelling?

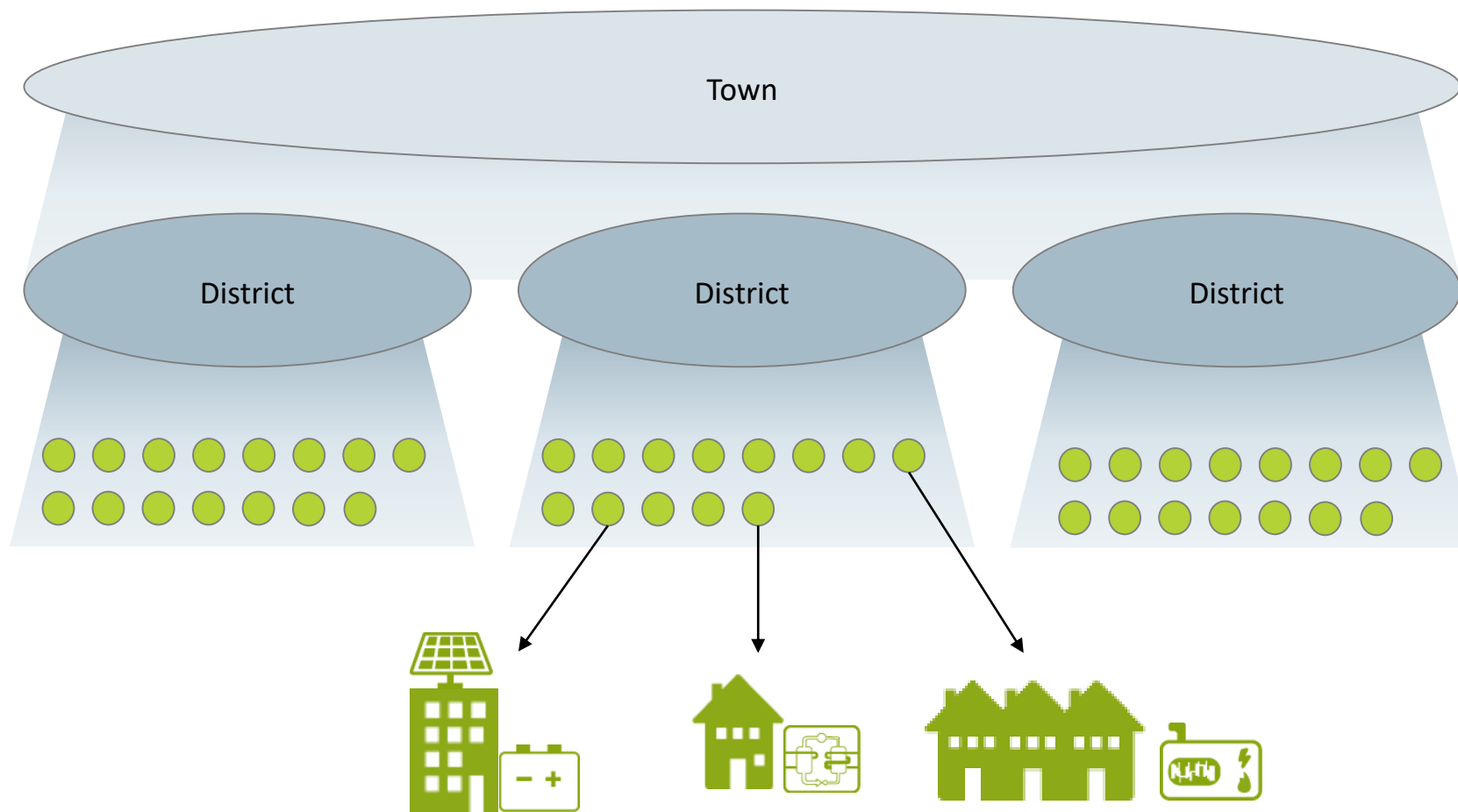


# Cellular modelling aims to depict cellular energy systems

# What is a cellular energy system?

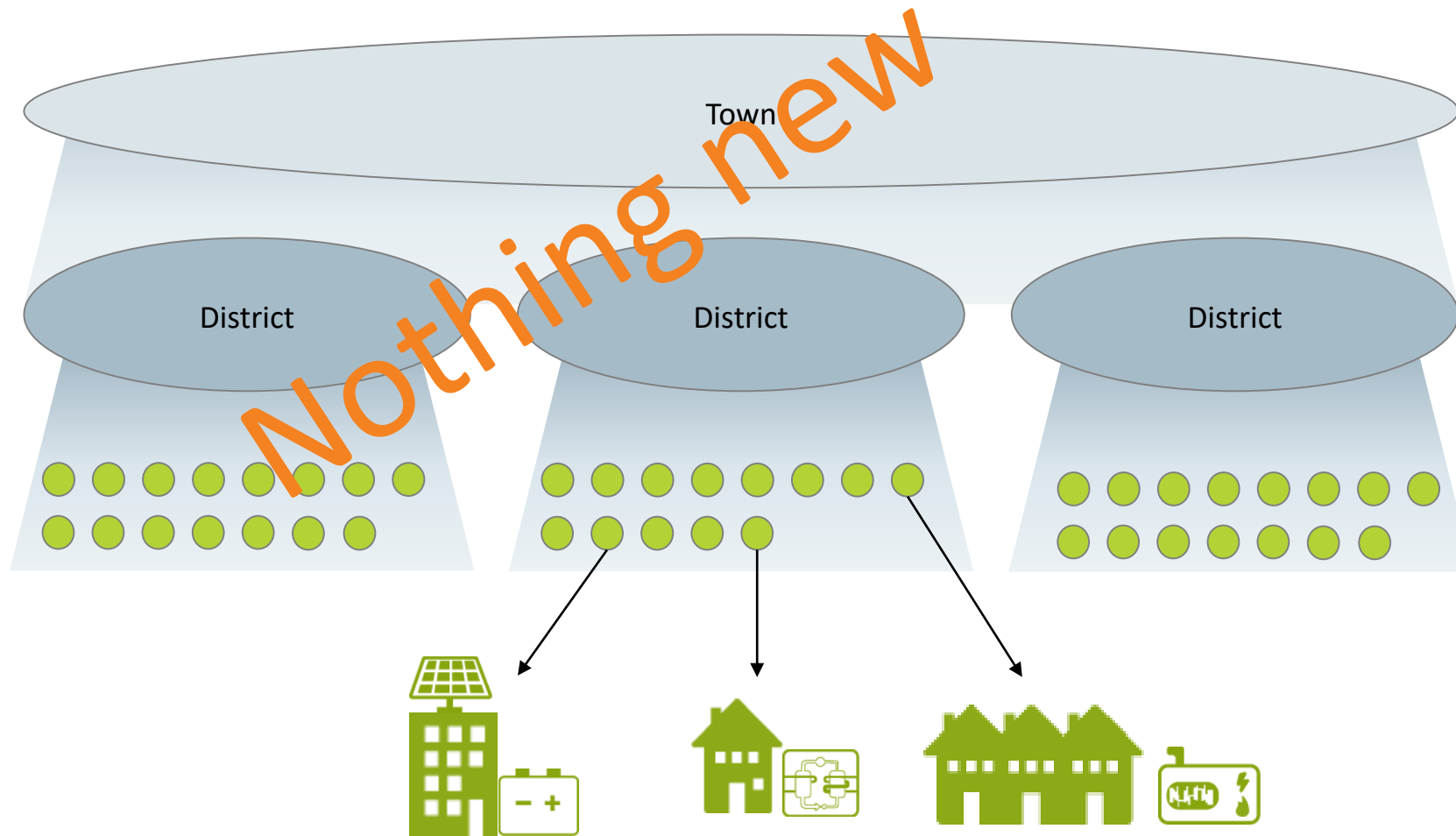
# Cellular Energy Systems

## An example



# Cellular Energy Systems

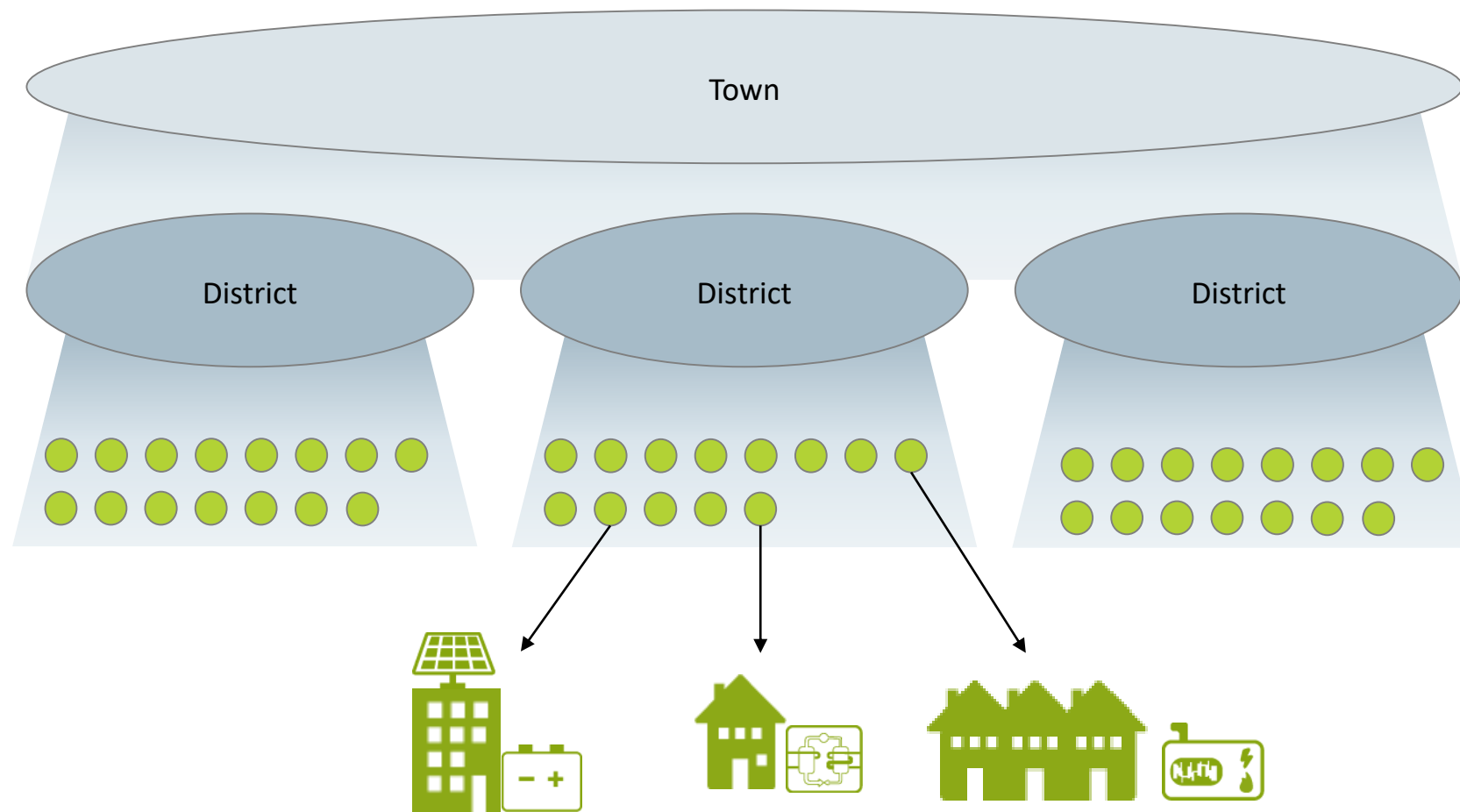
An example



Public information

# Cellular Energy Systems

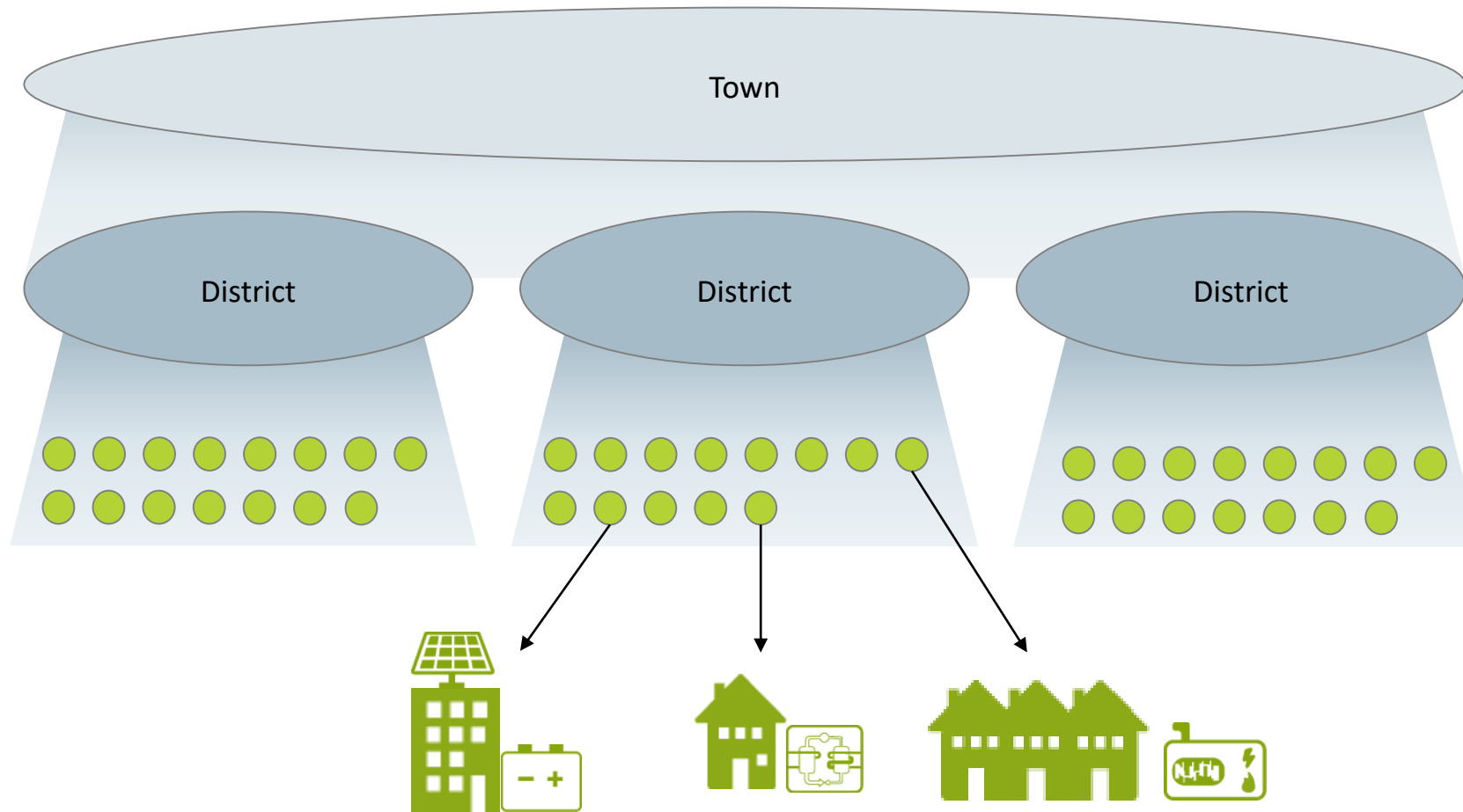
## An example



➔ Energycell

# Cellular Energy Systems

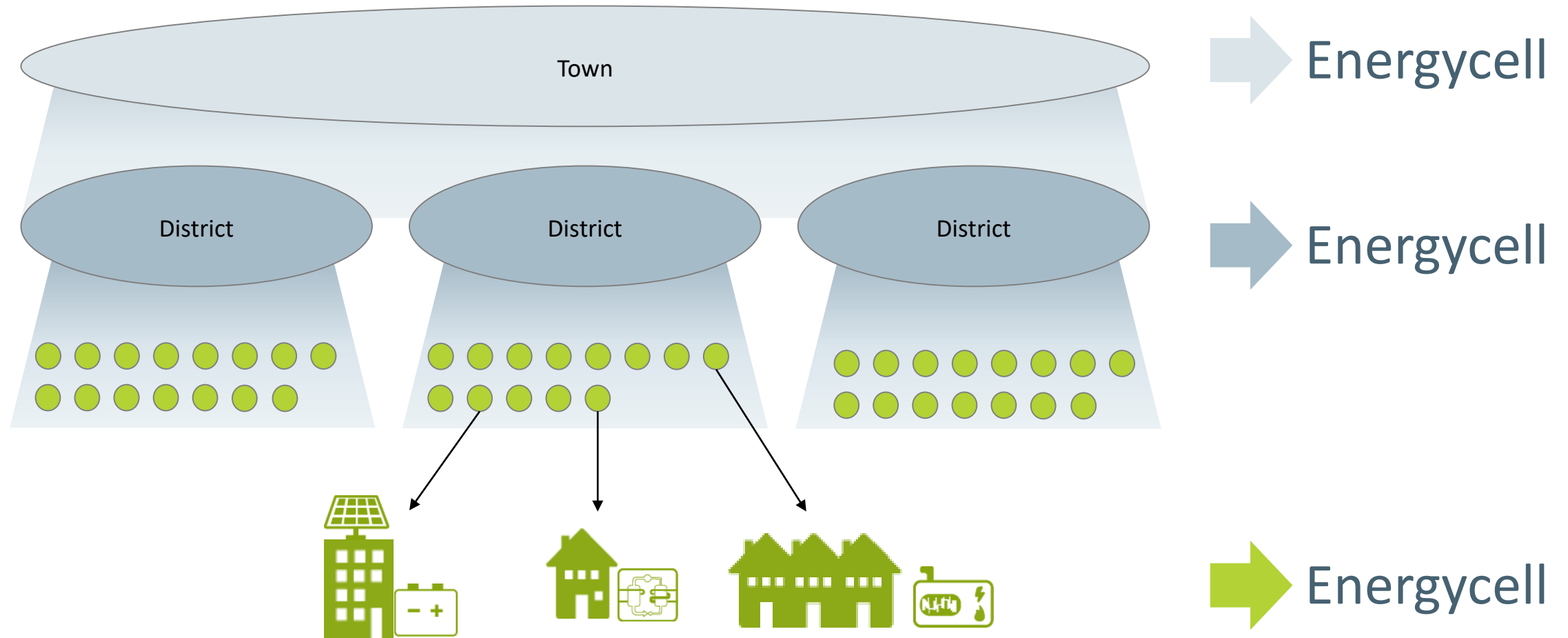
## An example



Energy Hub?  
➔ ~~Energy~~cell

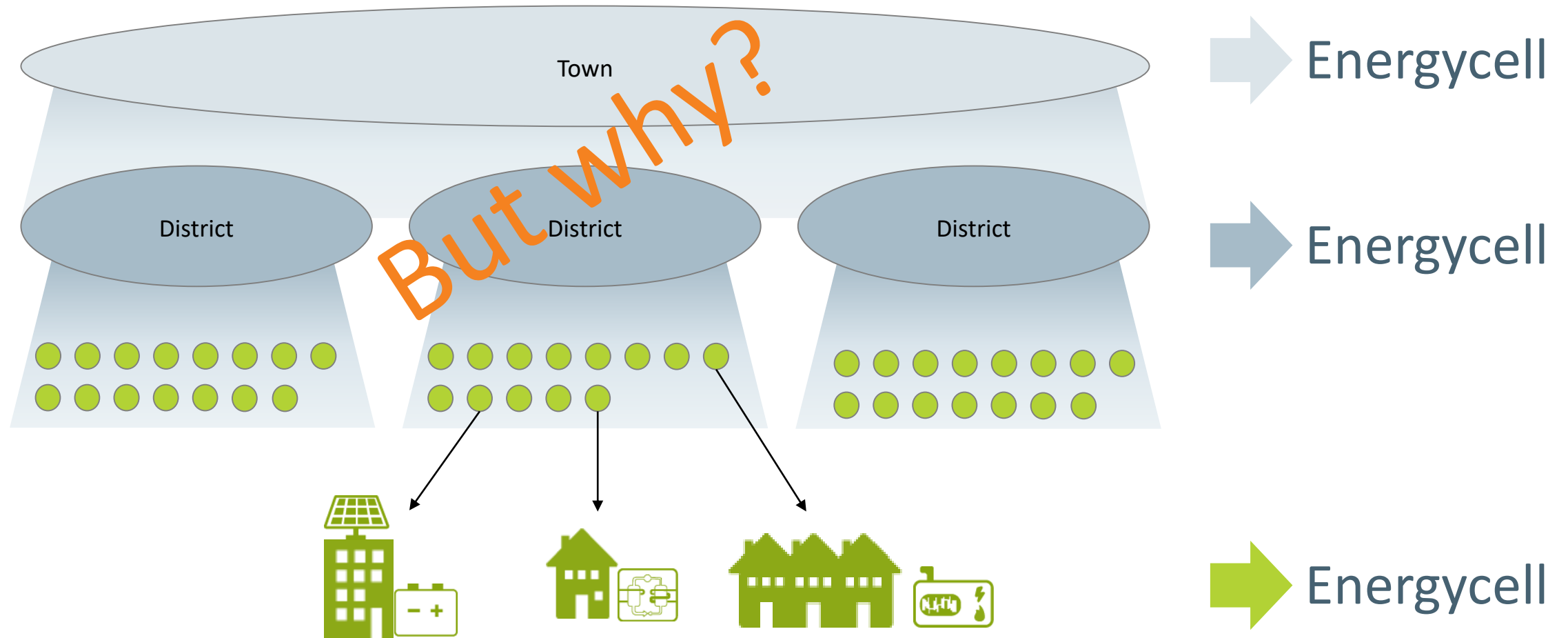
# Cellular Energy Systems

An example



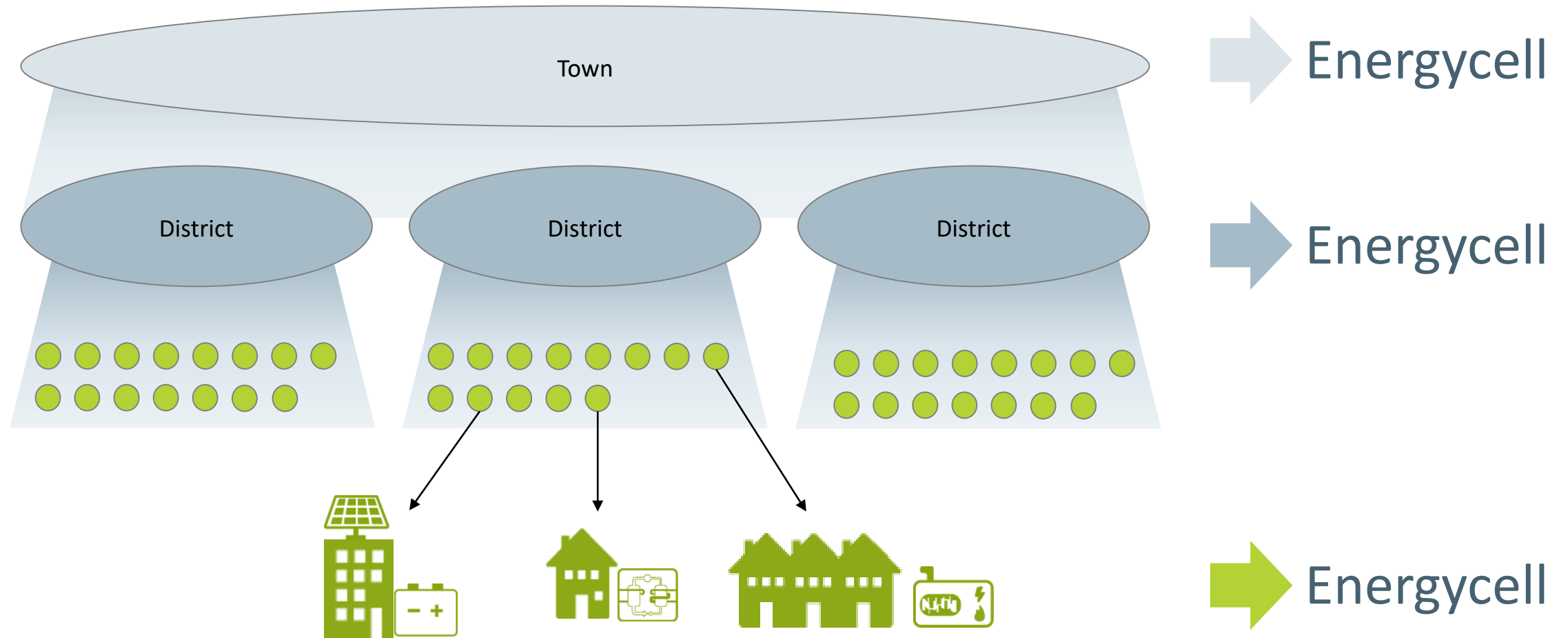
# Cellular Energy Systems

An example



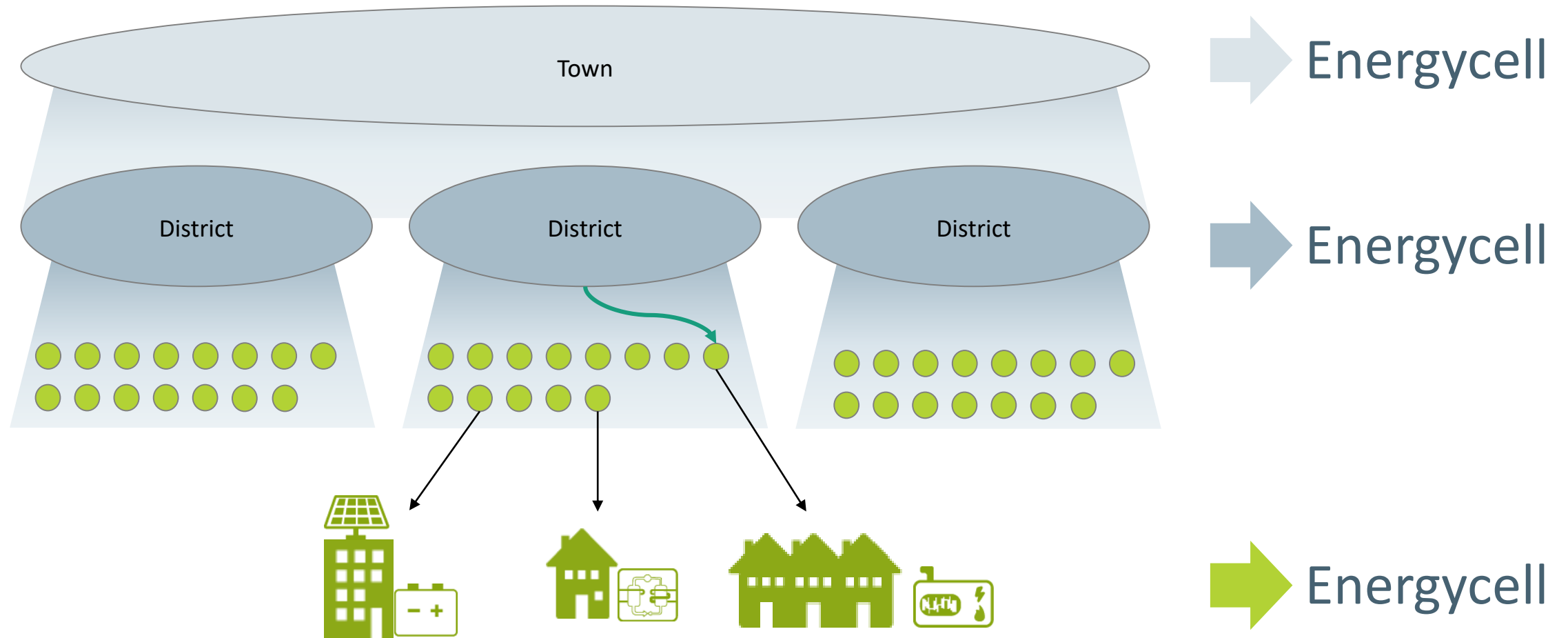
# Cellular Energy Systems

An example



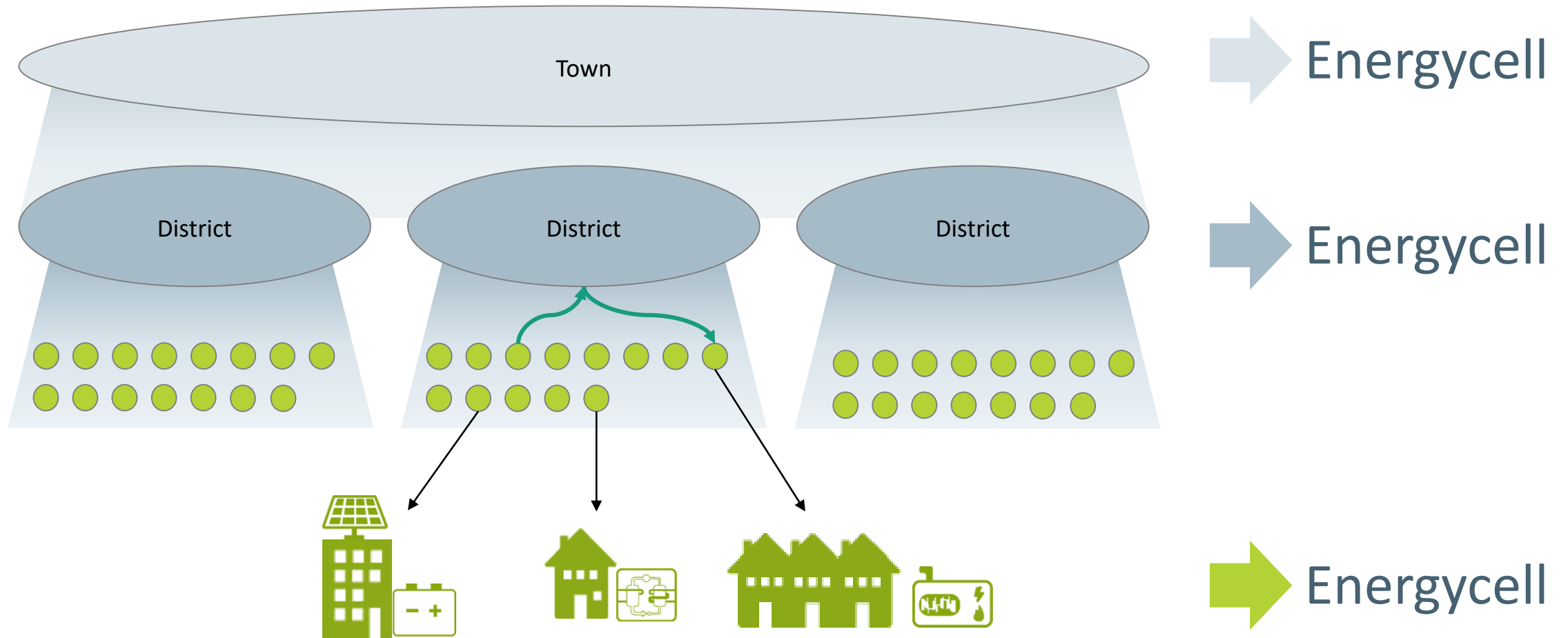
# Cellular Energy Systems

An example



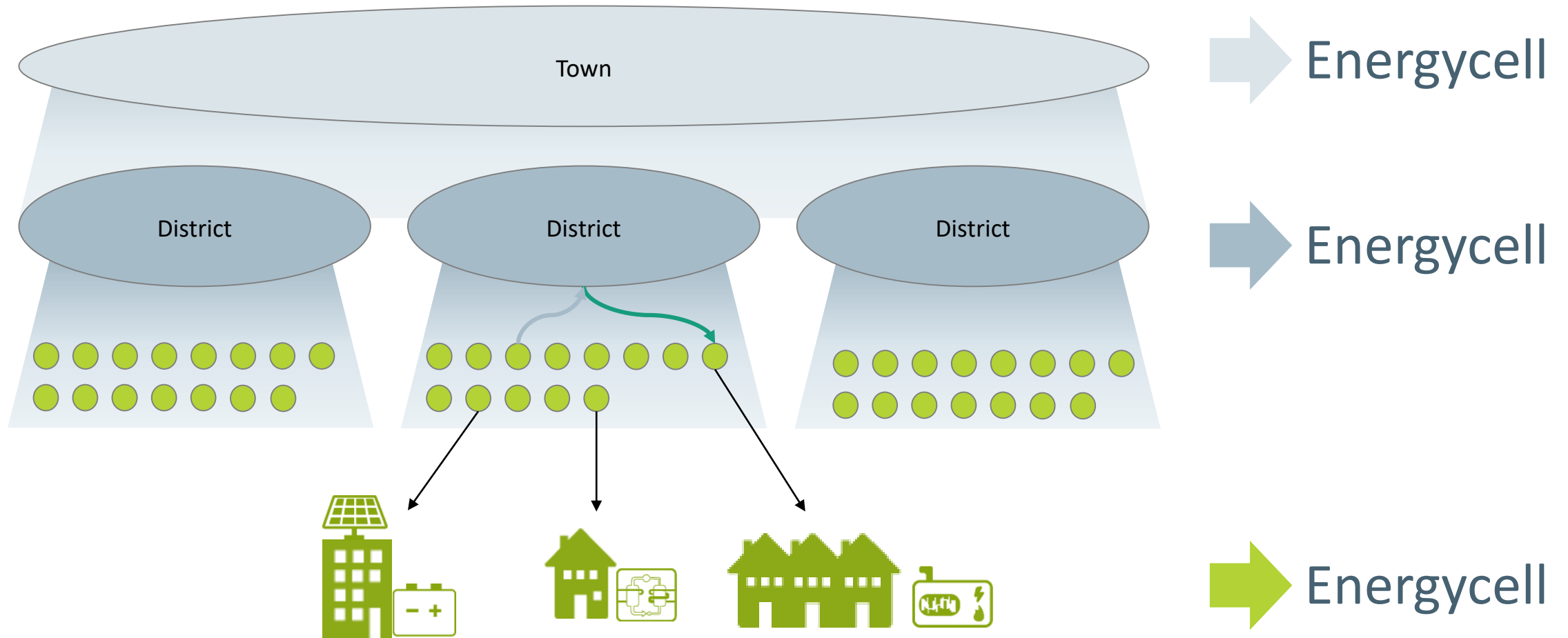
# Cellular Energy Systems

An example



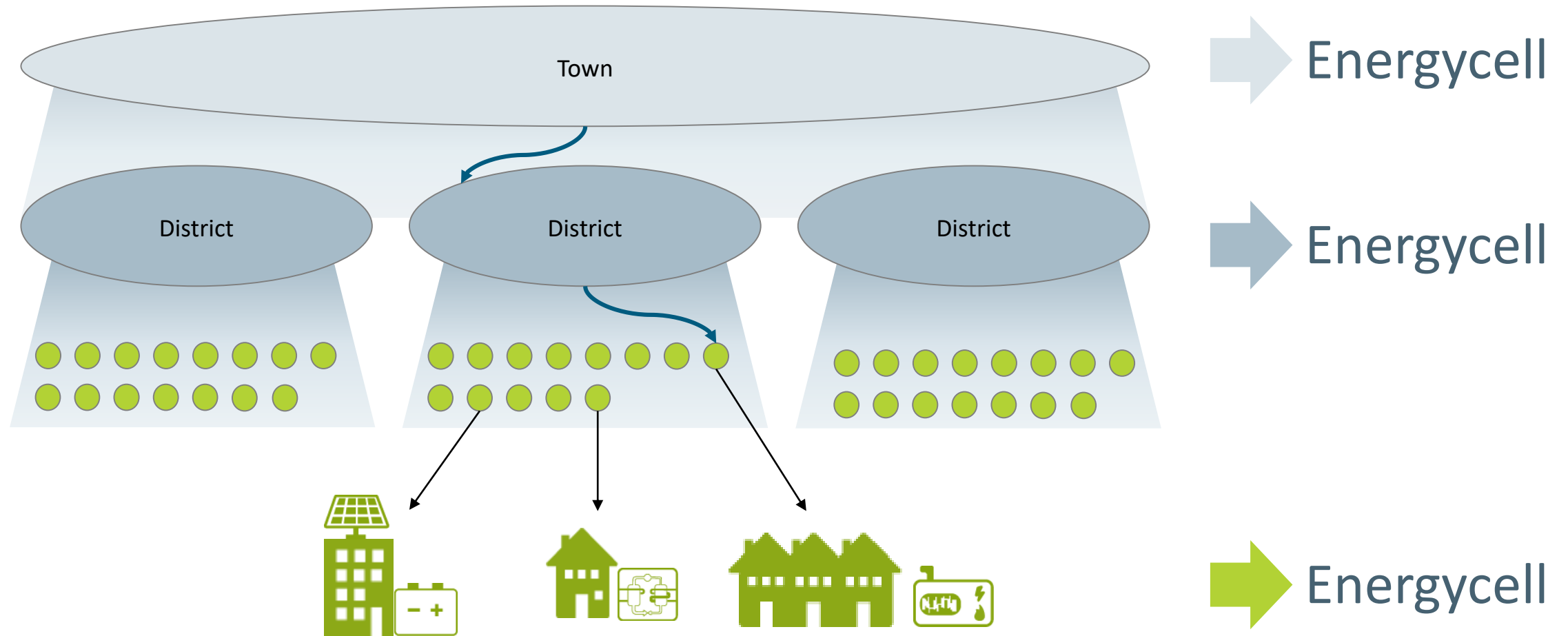
# Cellular Energy Systems

## An example



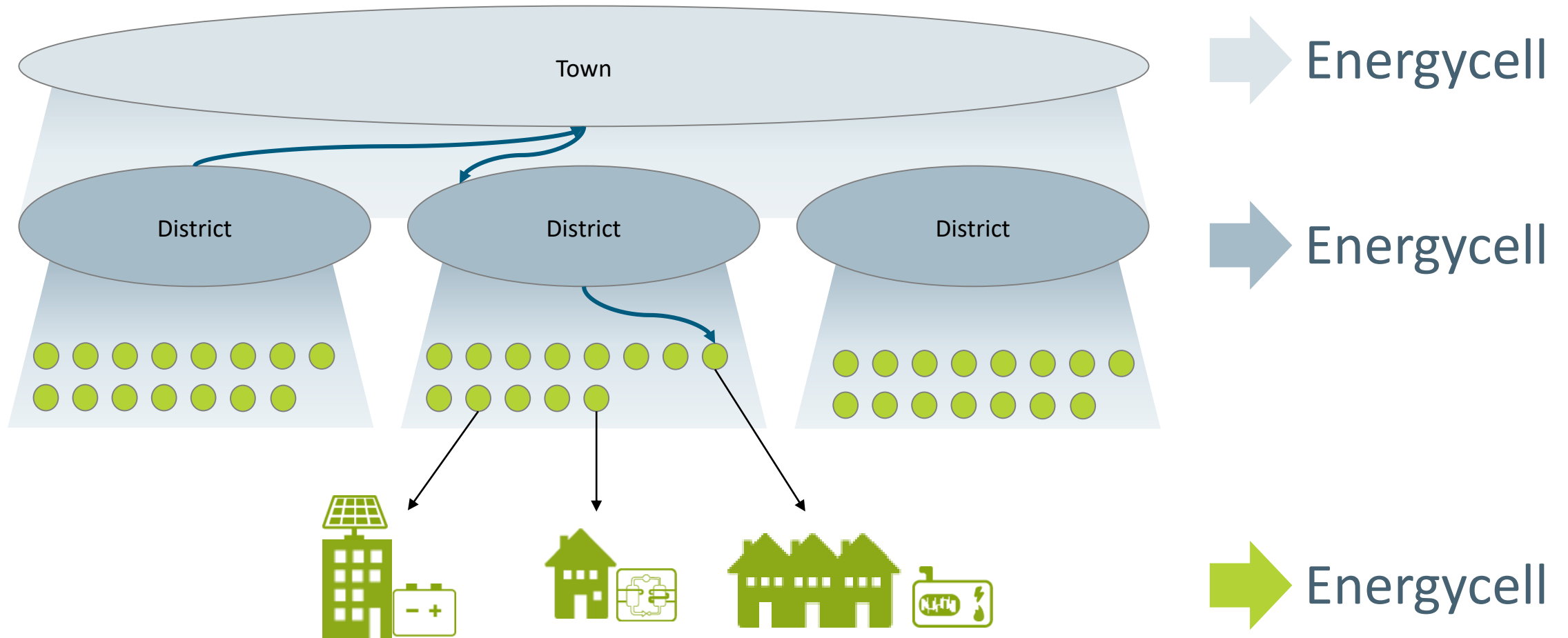
# Cellular Energy Systems

An example



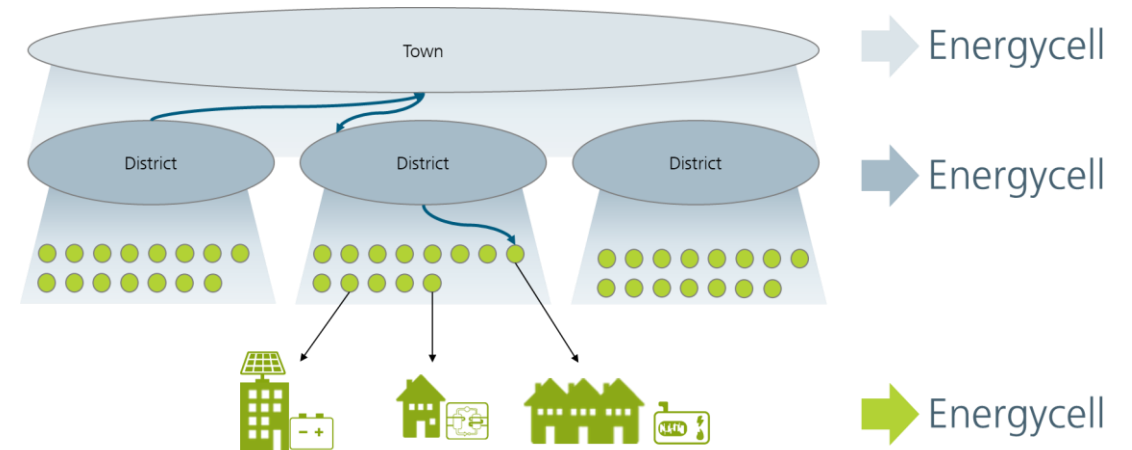
# Cellular Energy Systems

## An example





# The exchange of information and energy is always handled on the lowest level possible!

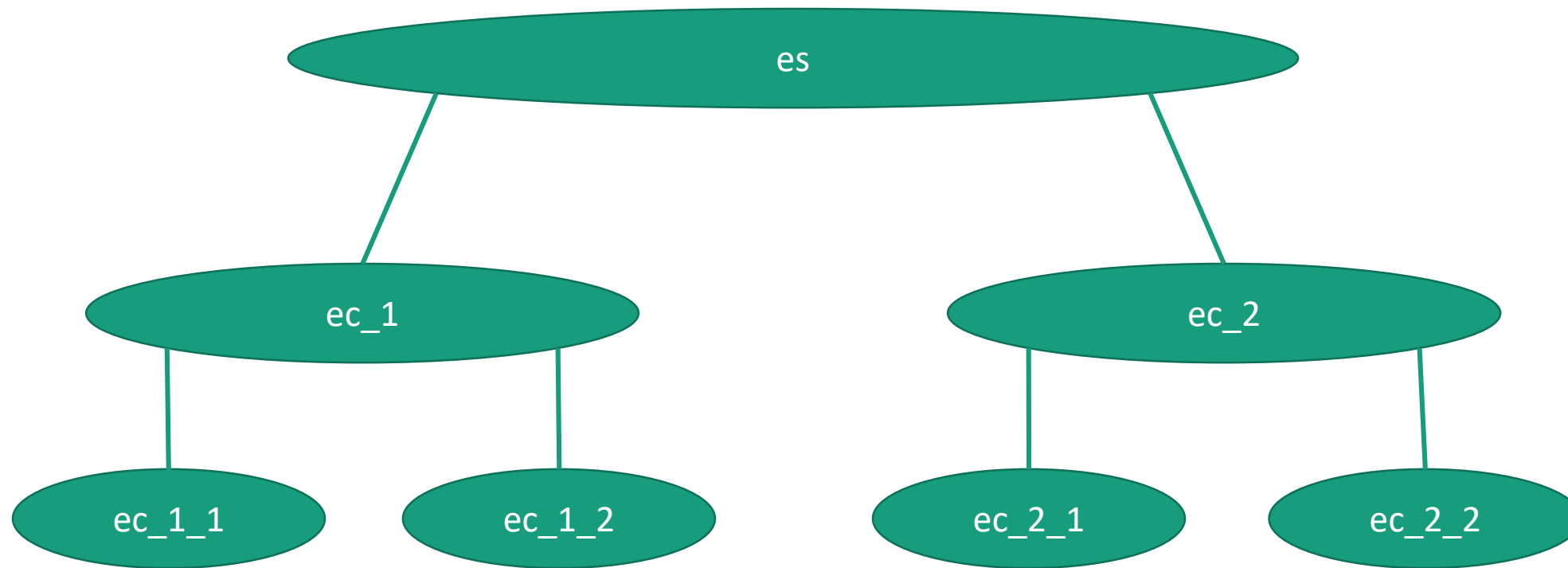


# How can this be modelled?

# Modelling approach

## Network layout

---



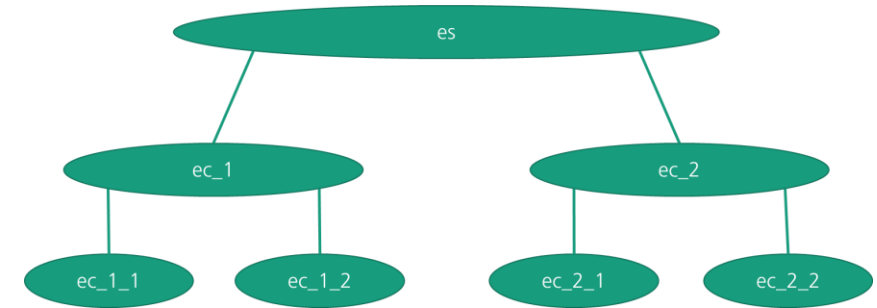
# Modelling approach

## Cell creation

```
# create the energy cells
es = EnergyCell(label="es", timeindex=daterange, infer_last_interval=False)
ec_1 = EnergyCell(label="ec1", timeindex=daterange, infer_last_interval=False)
ec_2 = EnergyCell(label="ec2", timeindex=daterange, infer_last_interval=False)
ec_1_1 = EnergyCell(
    label="ec1_1", timeindex=daterange, infer_last_interval=False
)
ec_1_2 = EnergyCell(
    label="ec1_2", timeindex=daterange, infer_last_interval=False
)
ec_2_1 = EnergyCell(
    label="ec2_1", timeindex=daterange, infer_last_interval=False
)
ec_2_2 = EnergyCell( ←
    label="ec2_2", timeindex=daterange, infer_last_interval=False
)
...
```

adding sources, sinks, buses, transformers to each cell...

```
...
ec_1.add(source_el_ec_1)
ec_2.add(source_el_ec_2)
ec_1_1.add(source_el_ec_1_1)
ec_1_2.add(source_el_ec_1_2)
ec_2_1.add(source_el_ec_2_1)
ec_2_2.add(source_el_ec_2_2)
```



EnergyCell is direct heir of the EnergySystem class  
(no new functionality as of yet)

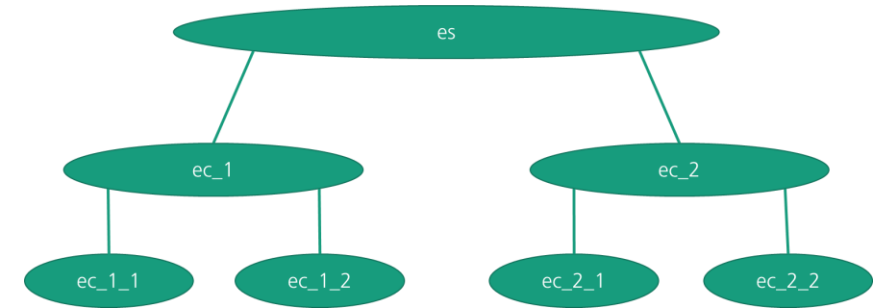
# Modelling approach

## Cell creation

```
# create the energy cells
es = EnergyCell(label="es", timeindex=daterange, infer_last_interval=False)
ec_1 = EnergyCell(label="ec1", timeindex=daterange, infer_last_interval=False)
ec_2 = EnergyCell(label="ec2", timeindex=daterange, infer_last_interval=False)
ec_1_1 = EnergyCell(
    label="ec1_1", timeindex=daterange, infer_last_interval=False
)
ec_1_2 = EnergyCell(
    label="ec1_2", timeindex=daterange, infer_last_interval=False
)
ec_2_1 = EnergyCell(
    label="ec2_1", timeindex=daterange, infer_last_interval=False
)
ec_2_2 = EnergyCell( ←
    label="ec2_2", timeindex=daterange, infer_last_interval=False
)
...
```

adding sources, sinks, buses, transformers to each cell...

```
...
ec_1.add(source_el_ec_1)
ec_2.add(source_el_ec_2)
ec_1_1.add(source_el_ec_1_1)
ec_1_2.add(source_el_ec_1_2)
ec_2_1.add(source_el_ec_2_1)
ec_2_2.add(source_el_ec_2_2)
```



EnergyCell is direct heir of the EnergySystem class  
(no new functionality as of yet)

How are the connections realized?

# Modelling approach

## Connection establishment

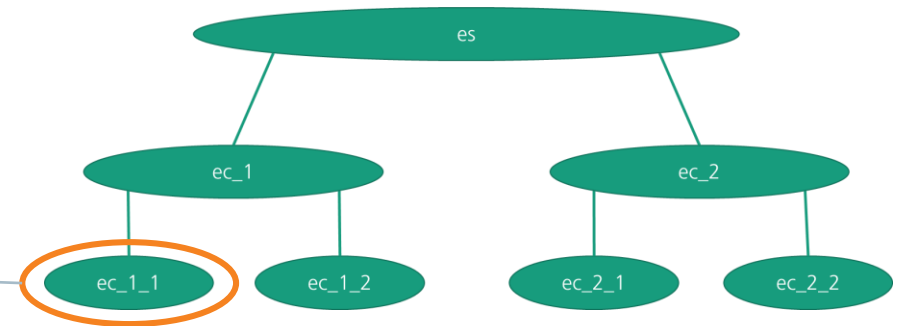
```
# relax the max constraint to see different results
```

```
connector_el_ec_1_1 = buses.Bus(  
    label="connector_el_ec_1_1",  
    inputs={  
        bus_el_ec_1: flows.Flow(),  
        bus_el_ec_1_1: flows.Flow(),  
    },  
    outputs={  
        bus_el_ec_1: flows.Flow(  
            investment=Investment(ep_costs=5, maximum=10, existing=20)  
        ),  
        bus_el_ec_1_1: flows.Flow(),  
    },  
)
```

Two inputs:

Flow from outside the cell (the bus of the „parent“ cell) to the connector

Flow from inside the cell (own bus) to the connector



# Modelling approach

## Connection establishment

```
# relax the max constraint to see different results
```

```
connector_el_ec_1_1 = buses.Bus(  
    label="connector_el_ec_1_1",  
    inputs={  
        bus_el_ec_1: flows.Flow(),  
        bus_el_ec_1_1: flows.Flow(),  
    },  
    outputs={  
        bus_el_ec_1: flows.Flow(  
            investment=Investment(ep_costs=5, maximum=10, existing=20)  
        ),  
        bus_el_ec_1_1: flows.Flow(),  
    },  
)
```

### Two inputs:

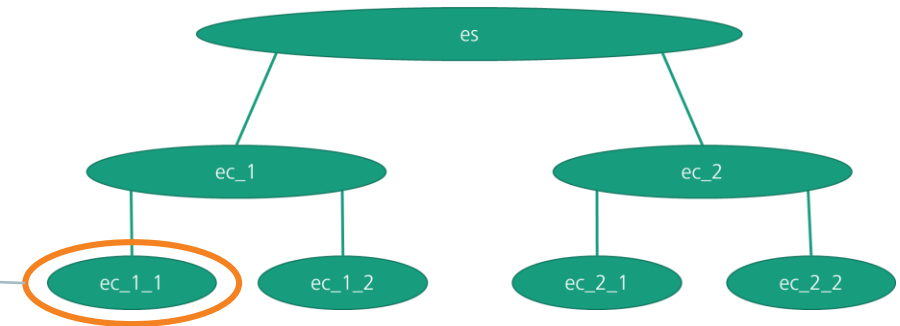
Flow from outside the cell (the bus of the „parent“ cell) to the connector

Flow from inside the cell (own bus) to the connector

### Two outputs:

Flow from the connector to the outside (bus of the „parent“ cell)

Flow from the connector to the inside of the cell (own bus)



# Modelling approach

## Connection establishment

```
# relax the max constraint to see different results
```

```
connector_el_ec_1_1 = buses.Bus(  
    label="connector_el_ec_1_1",  
    inputs={  
        bus_el_ec_1: flows.Flow(),  
        bus_el_ec_1_1: flows.Flow(),  
    },  
    outputs={  
        bus_el_ec_1: flows.Flow(  
            investment=Investment(ep_costs=5, maximum=10, existing=20)  
        ),  
        bus_el_ec_1_1: flows.Flow(),  
    },  
)
```

```
ec_1.add(connector_el_ec_1)  
ec_2.add(connector_el_ec_2)  
ec_1_1.add(connector_el_ec_1_1)  
ec_1_2.add(connector_el_ec_1_2)  
ec_2_1.add(connector_el_ec_2_1)  
ec_2_2.add(connector_el_ec_2_2)
```

### Two inputs:

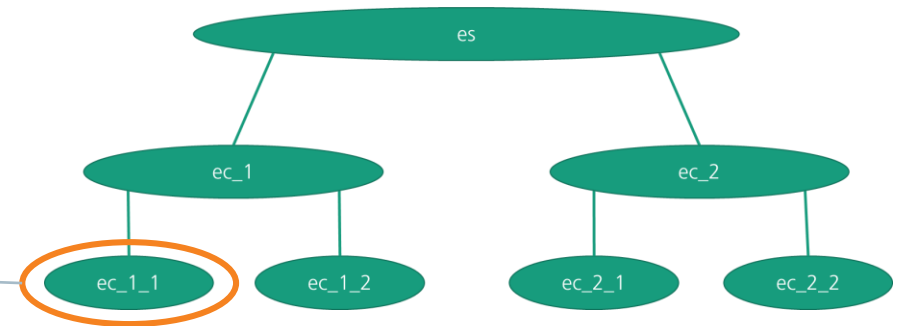
Flow from outside the cell (the bus of the „parent“ cell) to the connector

Flow from inside the cell (own bus) to the connector

### Two outputs:

Flow from the connector to the outside (bus of the „parent“ cell)

Flow from the connector to the inside of the cell (own bus)



# Modelling approach

## Connection establishment

```
# relax the max constraint to see different results
```

```
connector_el_ec_1_1 = buses.Bus(  
    label="connector_el_ec_1_1",  
    inputs={  
        bus_el_ec_1: flows.Flow(),  
        bus_el_ec_1_1: flows.Flow(),  
    },  
    outputs={  
        bus_el_ec_1: flows.Flow(  
            investment=Investment(ep_costs=5, maximum=10, existing=20)  
        ),  
        bus_el_ec_1_1: flows.Flow(),  
    },  
)
```

### Two inputs:

Flow from outside the cell (the bus of the „parent“ cell) to the connector

Flow from inside the cell (own bus) to the connector

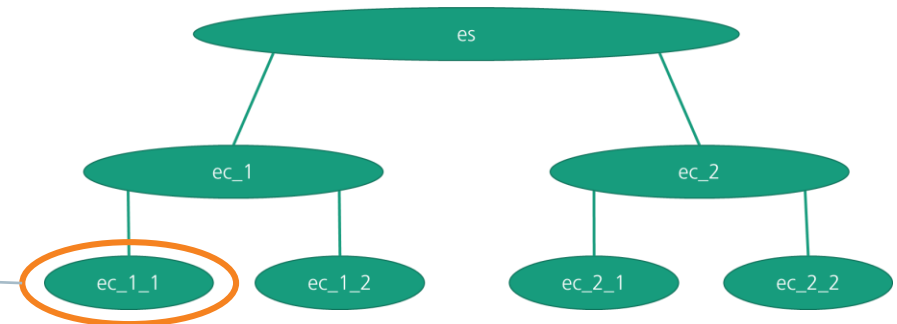
### Two outputs:

Flow from the connector to the outside (bus of the „parent“ cell)

Flow from the connector to the inside of the cell (own bus)

```
# the connectors are all part of the overarching es
```

```
es.add(  
    connector_el_ec_1,  
    connector_el_ec_2,  
    connector_el_ec_1_1,  
    connector_el_ec_1_2,  
    connector_el_ec_2_1,  
    connector_el_ec_2_2,  
)
```



# Modelling approach

## Connection establishment

```
# relax the max constraint to see different results
```

```
connector_el_ec_1_1 = buses.Bus(  
    label="connector_el_ec_1_1",  
    inputs={  
        bus_el_ec_1: flows.Flow(),  
        bus_el_ec_1_1: flows.Flow(),  
    },  
    outputs={  
        bus_el_ec_1: flows.Flow(  
            investment=Investment(ep_costs=5, maximum=10, existing=20)  
        ),  
        bus_el_ec_1_1: flows.Flow(),  
    },  
)
```

```
# the connectors are all part of the overarching es
```

```
es.add(  
    connector_el_ec_1,  
    connector_el_ec_2,  
    connector_el_ec_1_1,  
    connector_el_ec_1_2,  
    connector_el_ec_2_1,  
    connector_el_ec_2_2,  
)
```

### Two inputs:

Flow from outside the cell (the bus of the „parent“ cell) to the connector

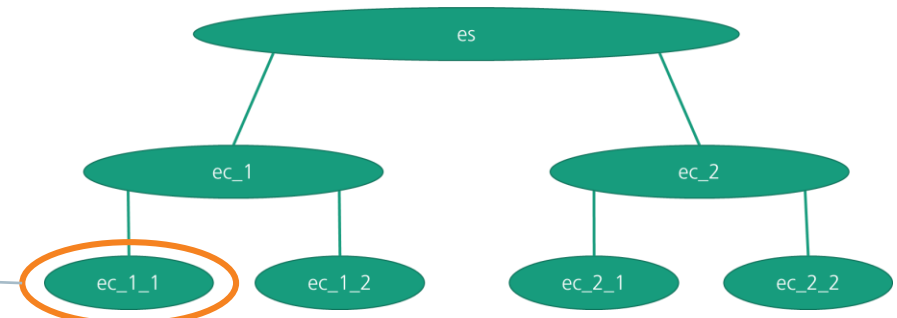
Flow from inside the cell (own bus) to the connector

### Two outputs:

Flow from the connector to the outside (bus of the „parent“ cell)

Flow from the connector to the inside of the cell (own bus)

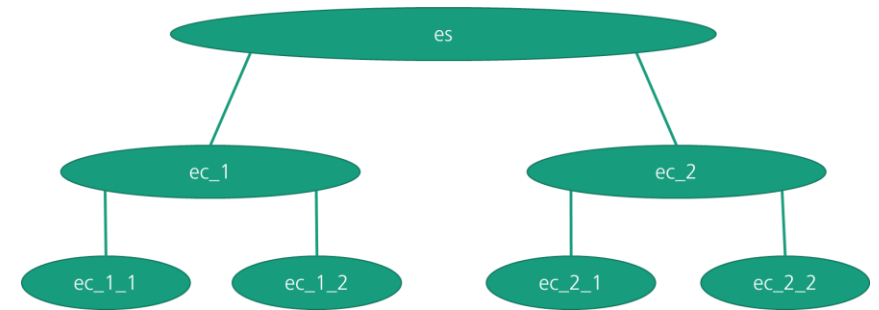
How are the cells passed to the model?



# Modelling approach

## Model generation

---



```
cmodel = Model(energysystem={es: [ec_1, ec_2, ec_1_1, ec_1_2, ec_2_1, ec_2_2]})
```

# What are the benefits?

# Benefits

---

- Modelling of cellular structures with lower abstraction level
- Possibly: faster solution times

# Benefits

---

- Modelling of cellular structures with lower abstraction level
- Possibly: faster solution times

*Why, what, how?*

# Benefits

---

- Modelling of cellular structures with lower abstraction level
- Possibly: faster solution times
- Outlook: Distributed Optimization with Dantzig-Wolfe Decomposition
  - Dividing the whole EnergySystem into multiple Sub-Problem (SP) and a Main Problem (MP)
  - Parallelized solving of the SP with reporting to the MP
    - Iterative generation of design and operation proposals by the SP, steered by the MP
    - Selection of the best design proposal for each SP by the MP
- The cellular structure begs to be solved distributedly (IMHO)

# Open Questions?

# Open Questions

---

- EnergyCell has no functionality gain over EnergySystem
  - Keep it, toss it, add functionality?
- EnergyCells are passed as dict to the model
  - Other data format?
- What kind of functionality would you appreciate?

## Pull-Request:

[github.com/oemof/oemof-solph/pull/912](https://github.com/oemof/oemof-solph/pull/912)

**Lennart Schürmann**

[lennart.schuermann@umsicht.fraunhofer.de](mailto:lennart.schuermann@umsicht.fraunhofer.de) 

[github.com/lensum](https://github.com/lensum) 

[researchgate.net/profile/Lennart-Schuermann](https://researchgate.net/profile/Lennart-Schuermann) 

[orcid.org/0000-0002-5624-232X](https://orcid.org/0000-0002-5624-232X) 