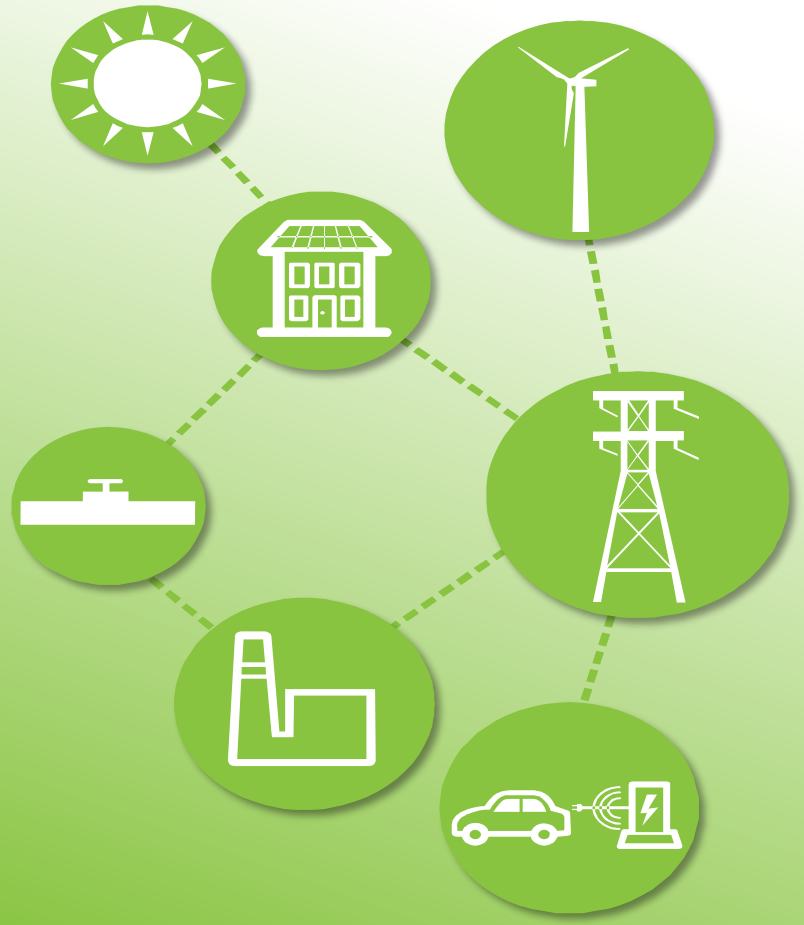
Spine



Bringing together tools, data, & models

Jody Dillon,

Energy Reform Ltd.

Workshop on the Development of an Open Modelling Platform for Electrification and Deep Decarbonisation Studies

February 22nd 2019

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N. 774629.

Spine: Open source toolbox for modelling integrated energy systems

- Project part funded by the Horizon 2020 program of the European Union
- LCE-05-2017 - Tools and technologies for coordination and integration of the European energy system
- 4 year project commenced October 2017 with a €3.7m budget
- 5 Partners

www.spine-model.eu

spine_info@vtt.fi

Spine

Motivation and Challenges

- Increased electrification creates more convergence between energy sectors leading to interesting opportunities which need to be studied
- Many existing tools are sectoral based and many multi-sectoral models have low levels of operational detail
- In most cases, the problem we want to solve is dictated by the tools we are using and have access to
- Studying these opportunities is highly data intensive and a coherent framework for data management/sharing is required

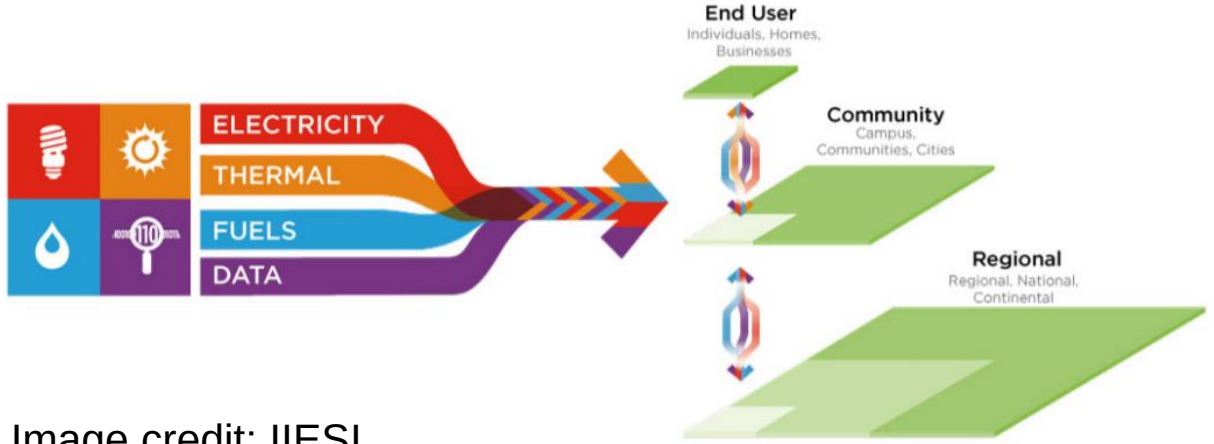
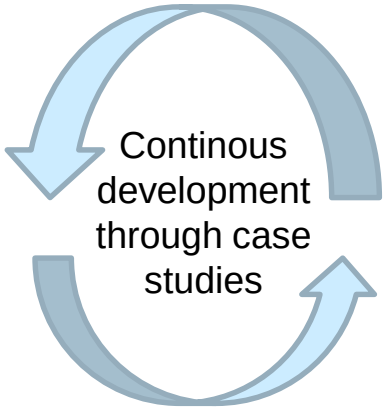


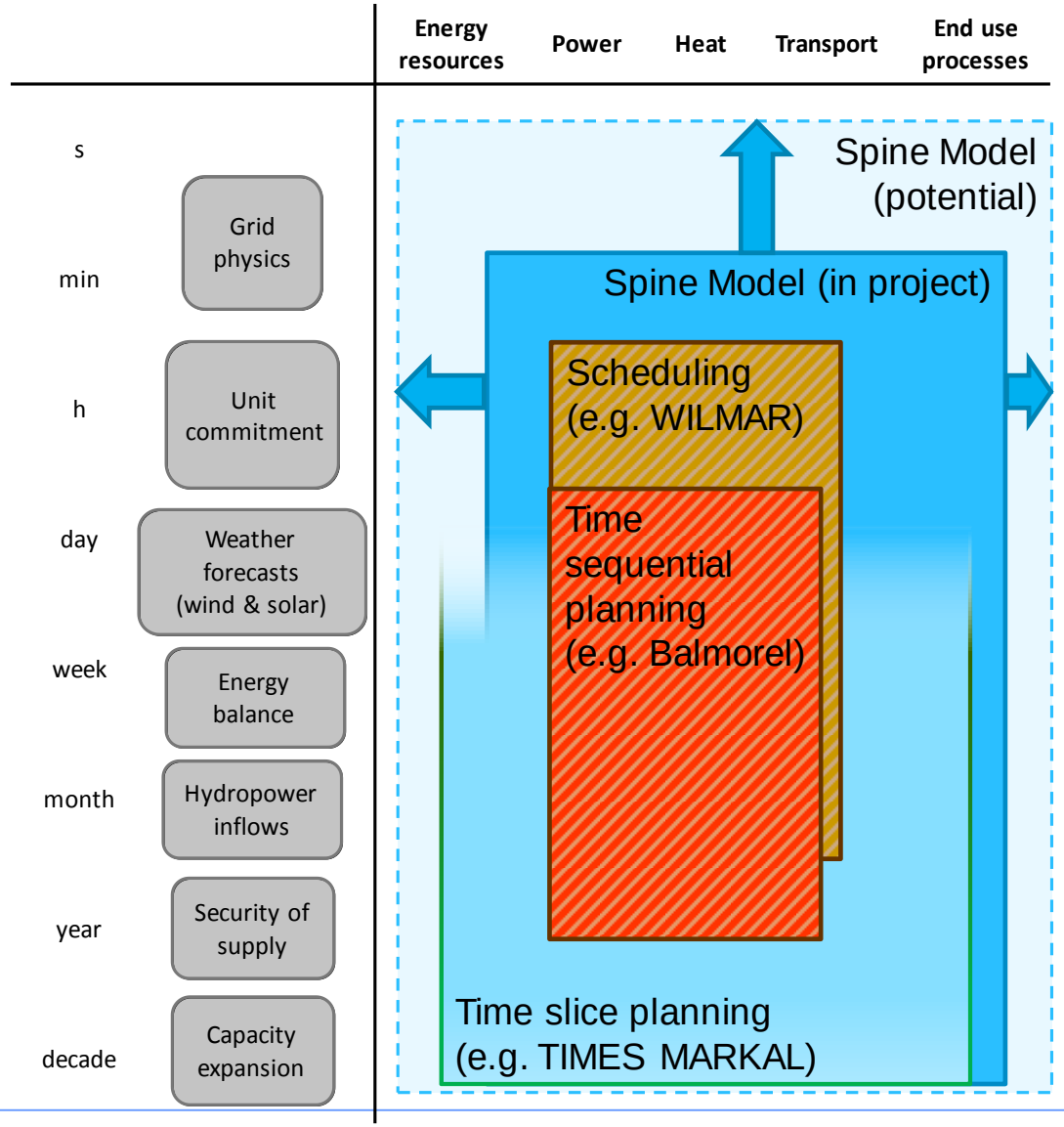
Image credit: IIESI

Project Scope and Objectives

Scientific Objective: Develop an open, flexible end-to-end energy system planning tool (SPINE toolbox and SPINE model)



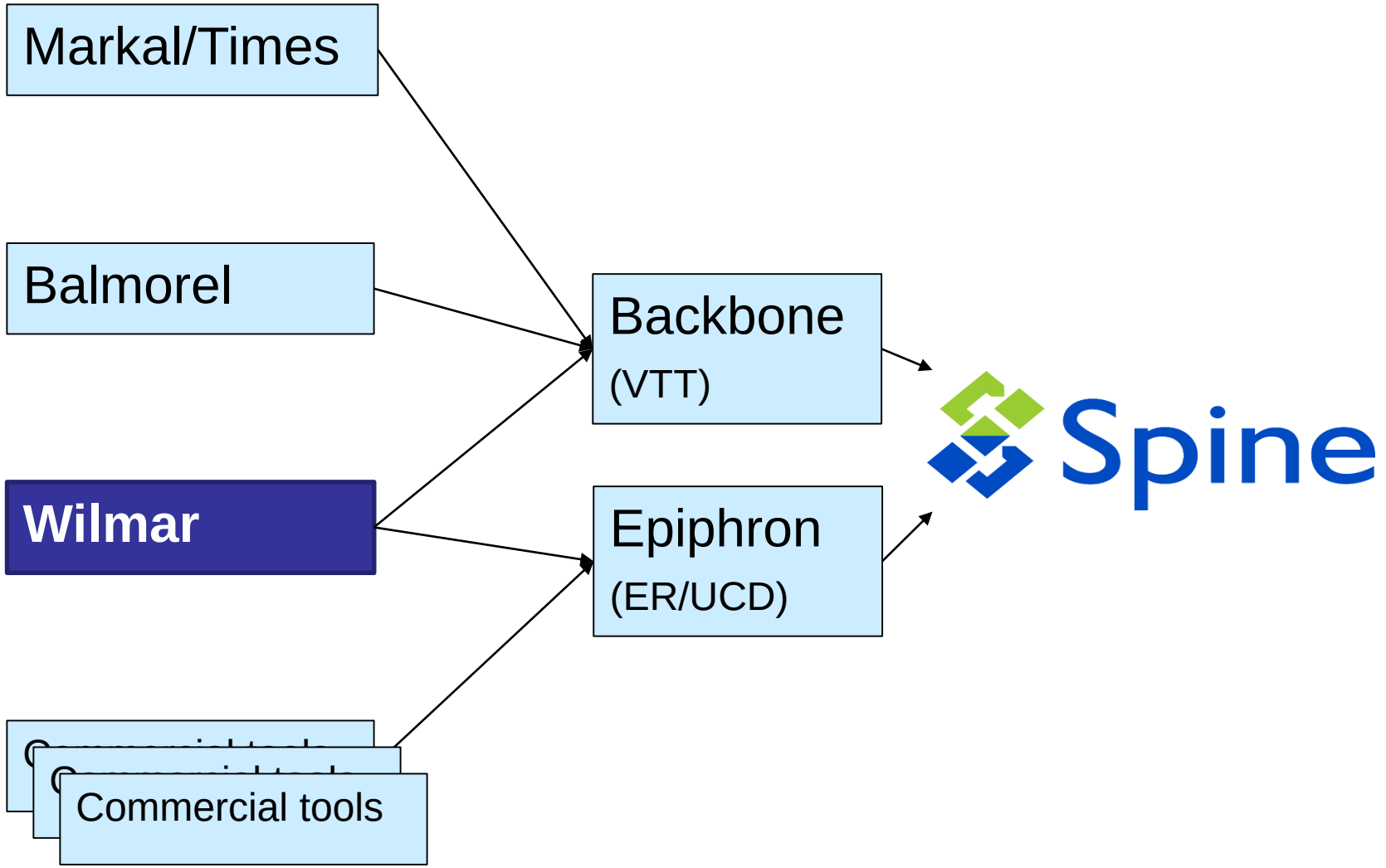
Deployment in 13 case studies: advance model capabilities, verify model behaviour, perform studies, involve stakeholders



Not "Just another model"

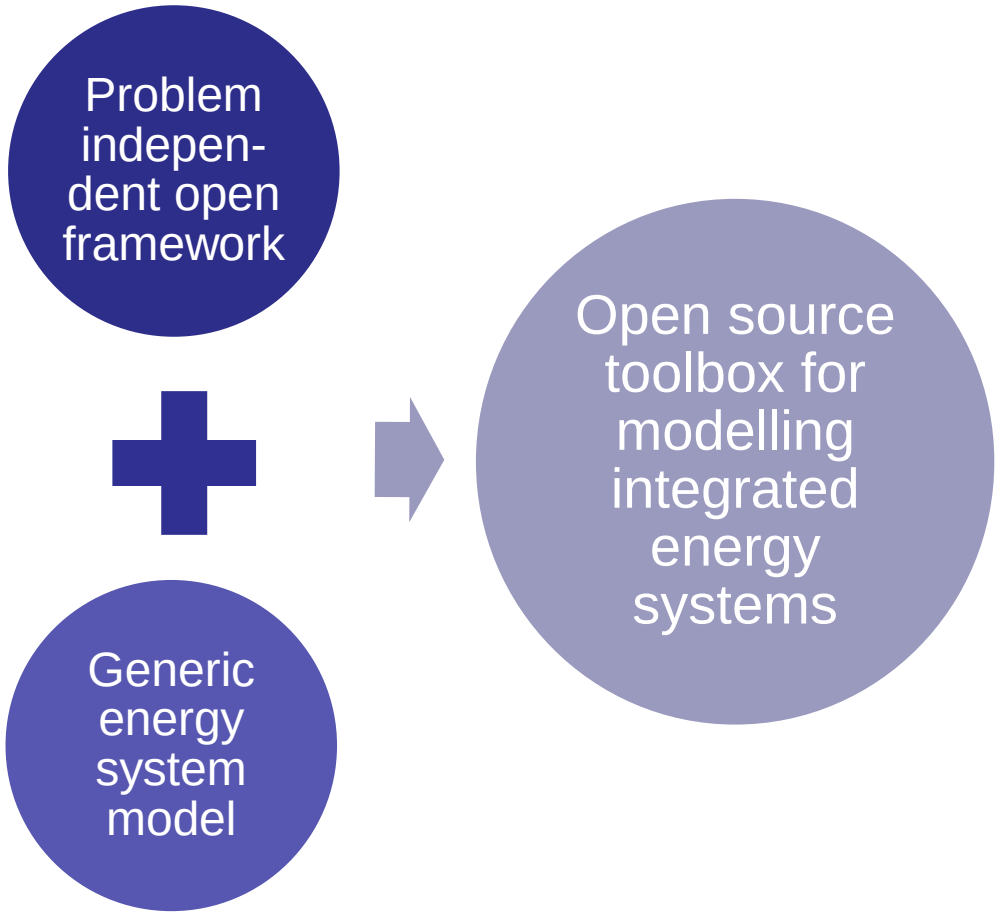
Spine

Spine Heritage

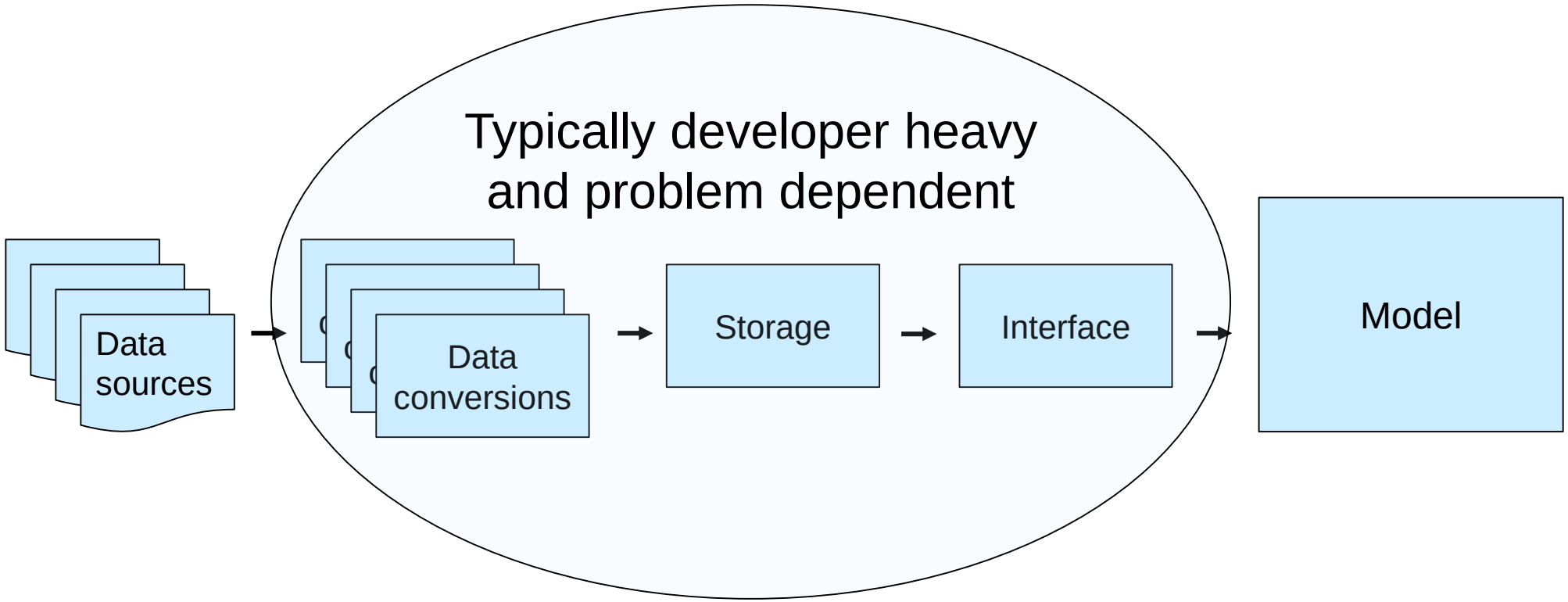


Key Design Objectives

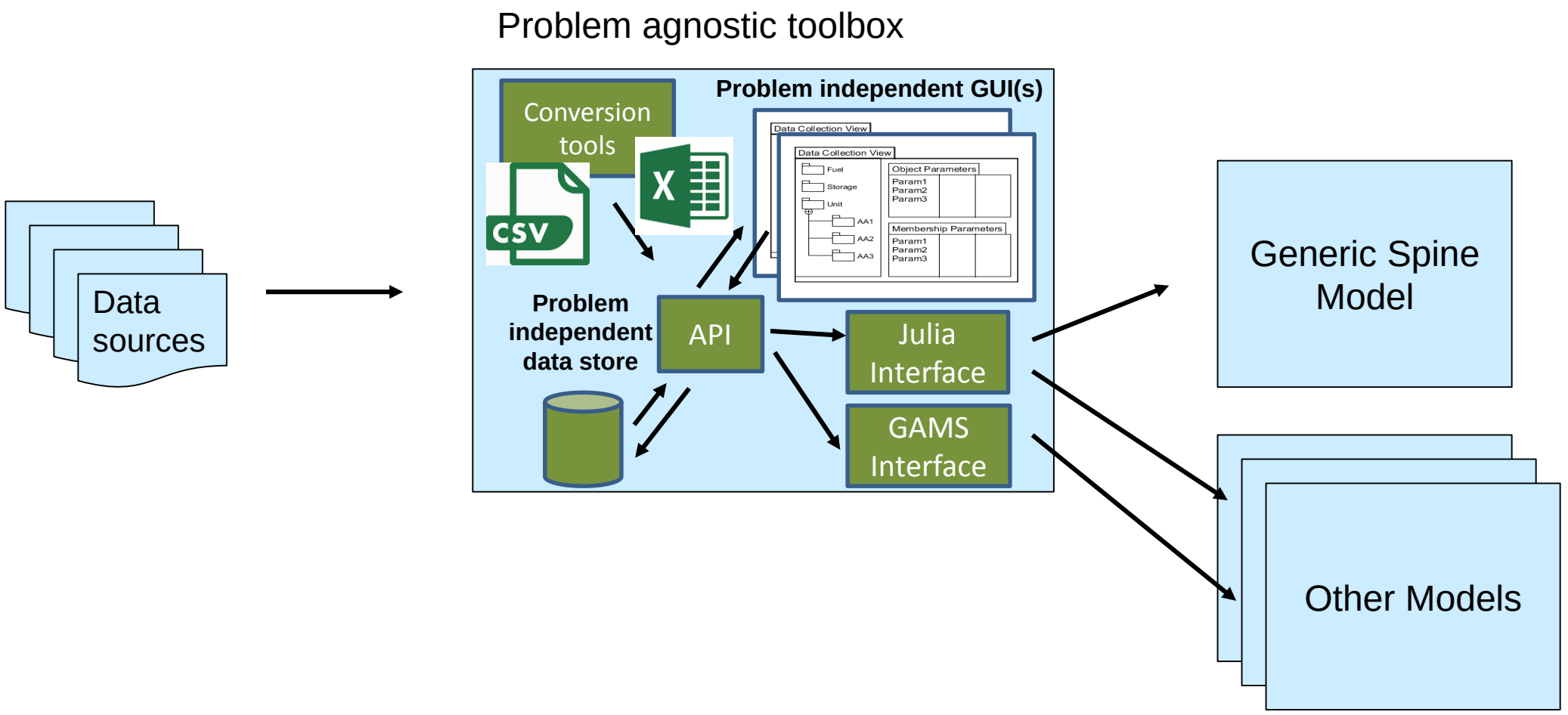
- We assume from the start that we don't know the precise problem a user may want to solve
- Develop a problem independent data management infrastructure that facilitates faster model creation and subsequent extension and development
- Key framework components should not require maintenance or development if the problem changes
- Facilitate integration of data and models within a single open framework
- Develop a generic energy system model capable of addressing the energy system integration challenges and opportunities
- Allow models and associated data to be defined in a flexible and coherent way



Typical Model/Study Structure

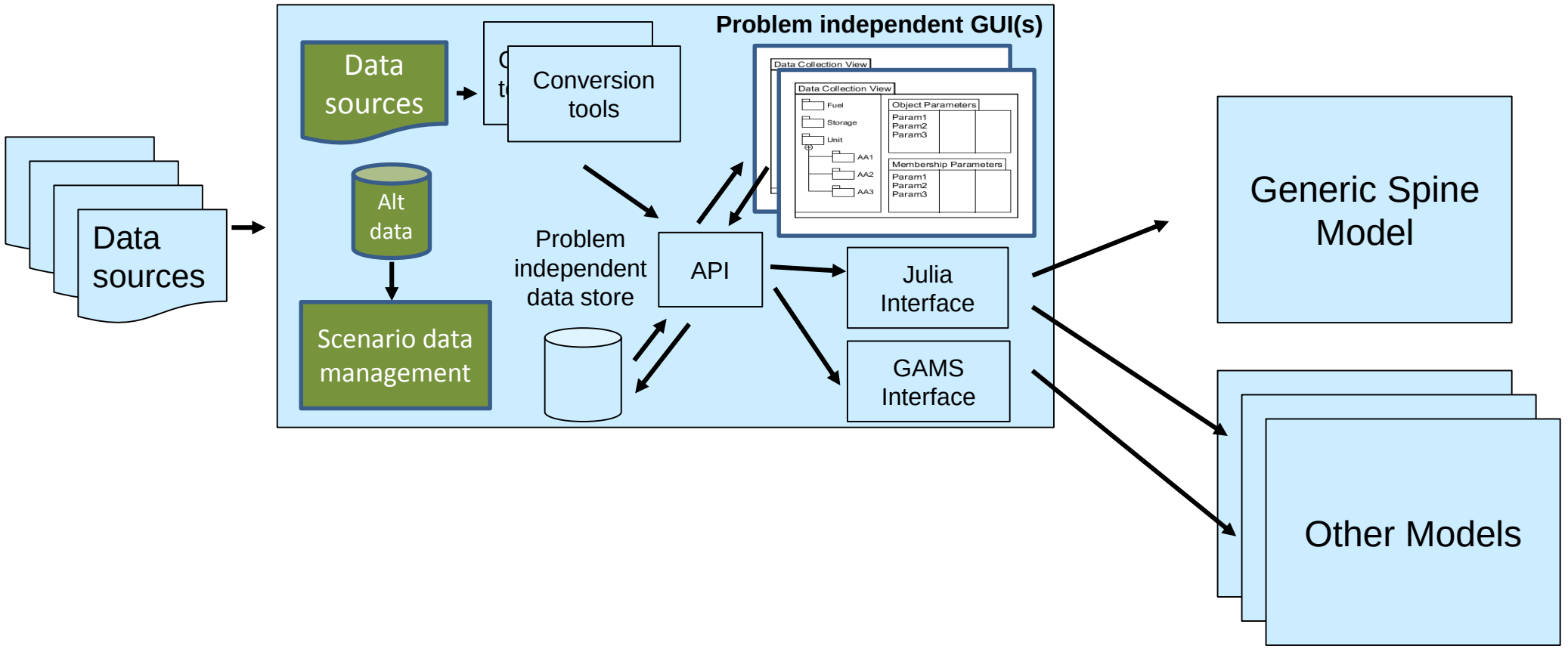


Spine Approach



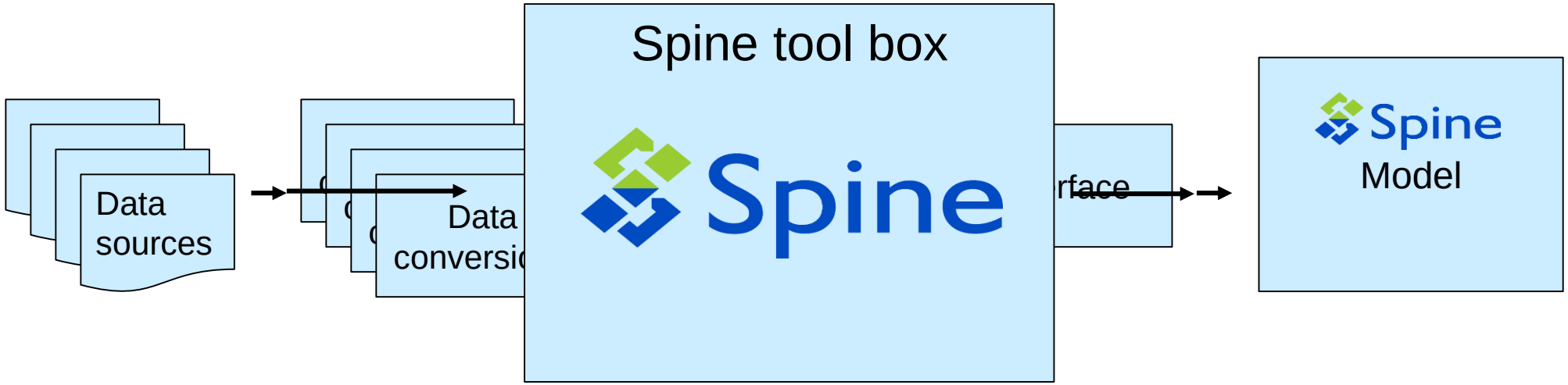
Spine

Spine Approach



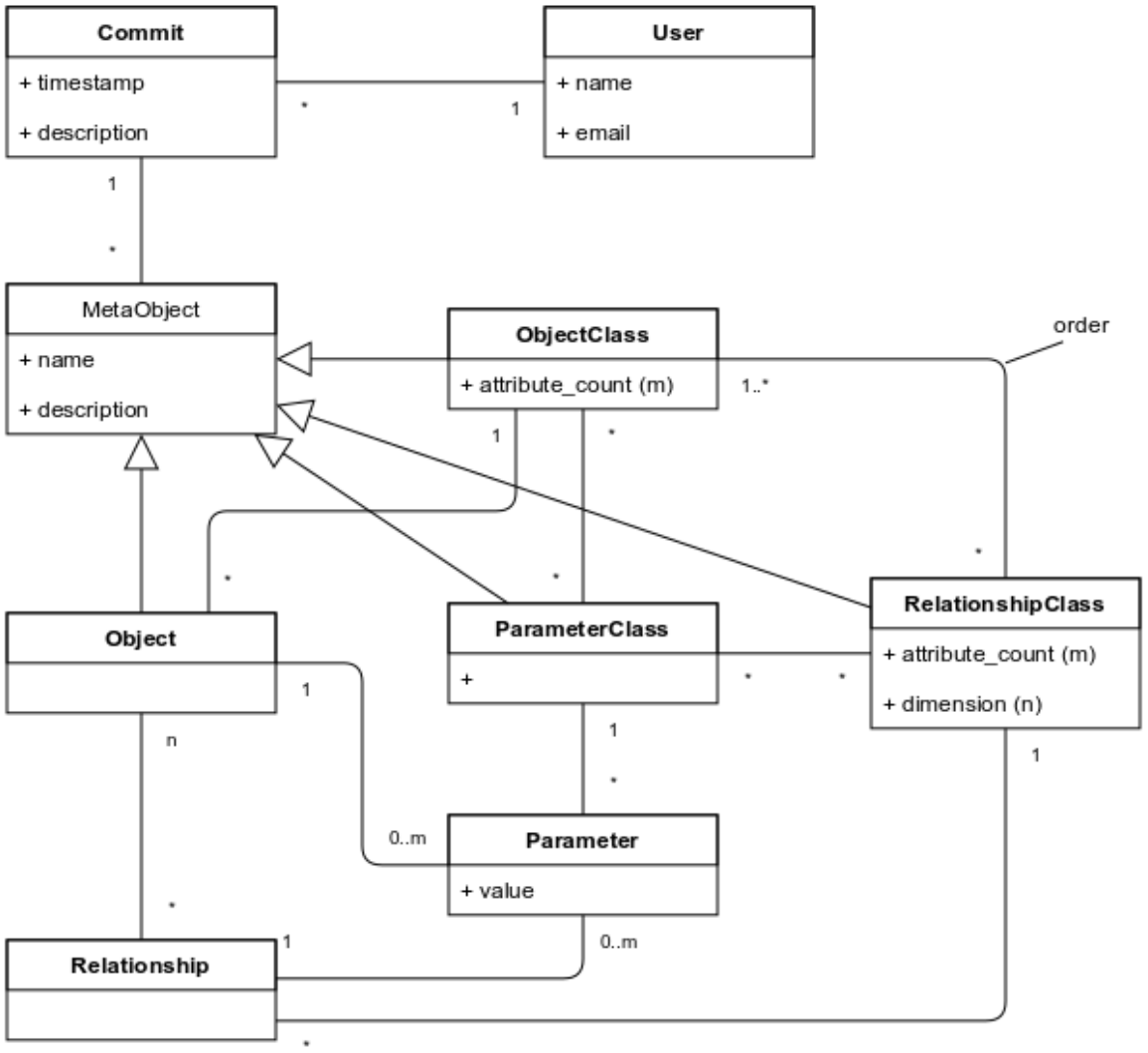
Spine

Alternative Approach



Problem independent data structure

- Entity–attribute–value structure with classes and relationships (EAV/CR)
- Fully flexible (generic) creation of:
  - Object classes
  - Relationship classes (n-dimensional)
  - Attributes and parameters
- Flexibility to group objects
- Database format independency (SQLAlchemy)



Spine

Data side features

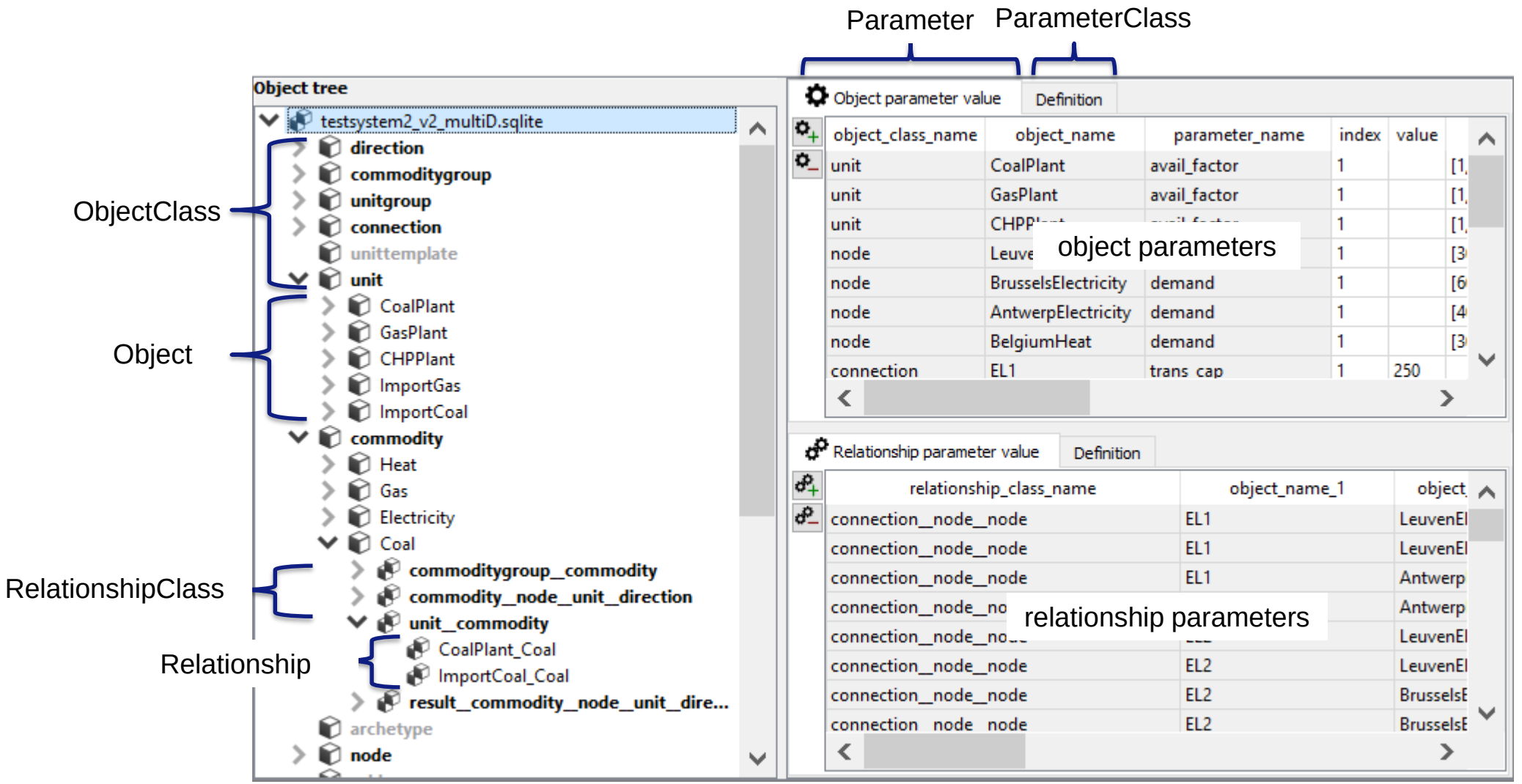
Data Features

- Transactional data: Rollback / Commit
- Static parameter values
- Time series parameters
- Time patterned values
- Symbols and expressions
- Flexible JSON field
  - E.g. piece-wise linear functions
  - Custom data types
- Parameter classes and enumerated parameters
- Entity archetypes

Current Import/Export Tools

- Excel import/export
  - Data template generation
  - Automatic data model creation
- Data package import with meta-data

Spine's generic data store: toolbox tree view



Enabling efficient model development: Problem independent data passthrough

Enabled by Spine database API and Julia metaprogramming

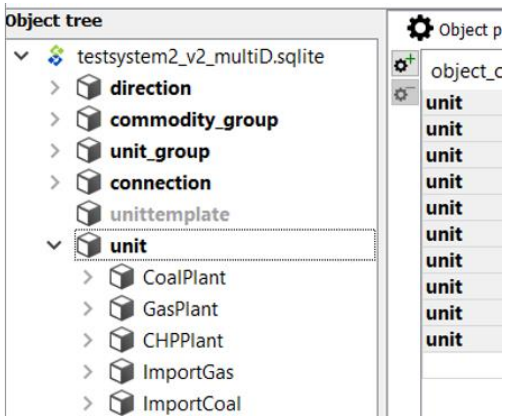
Model requires new entity type (model index), or data parameter

Add data via GUI

Reference entities/parameters by name in Julia code

e.g. need to add a new entity called “unit”

Right click -> add new object class



Reference new object class by name in code

```
function max_flow(m::Model, flow)
    @constraint(
        m,
        [
            u in unit(),
            c in commodity();
            max_flow(unit=u, commodity=c)
        ],
        flow[c, u, t] <= max_flow(unit = u
    )
end
```


Spine

Spine Julia interface example

Accessing the spine data model in Julia directly by object class and parameter name

Import the SpineModel.jl package:

In [25]:
```
using SpineModel
```

Specify location of datastore:

In [26]:
```
db_url = "sqlite:///c:\\workspace\\spine\\spinetoolbox\\models\\testsystem2_v2_multiD.sqlite"
```

Out[26]:
```
"sqlite:///c:\\workspace\\spine\\spinetoolbox\\models\\testsystem2_v2_multiD.sqlite"
```

Do the magic!

In [27]:
```
JuMP_all_out(db_url)
```

The data has been loaded into memory and we now have "convenience functions" to access all object classes, parameters and relationships and can reference these by name. For example:

Enumerate all objects of type "unit":

In [28]:
```
for u in unit()
    println(u)
end
```

```
CoalPlant: 1
GasPlant: 1
CHPPlant: 1
ImportGas: 1
ImportCoal: 1
```

Spine

Spine Julia interface example

Accessing the spine data model in Julia directly by object_class and parameter name

What about parameters?

In [29]:
```
for u in unit()
    println(join([u,": ", number_of_units(unit=u)]))
end
```

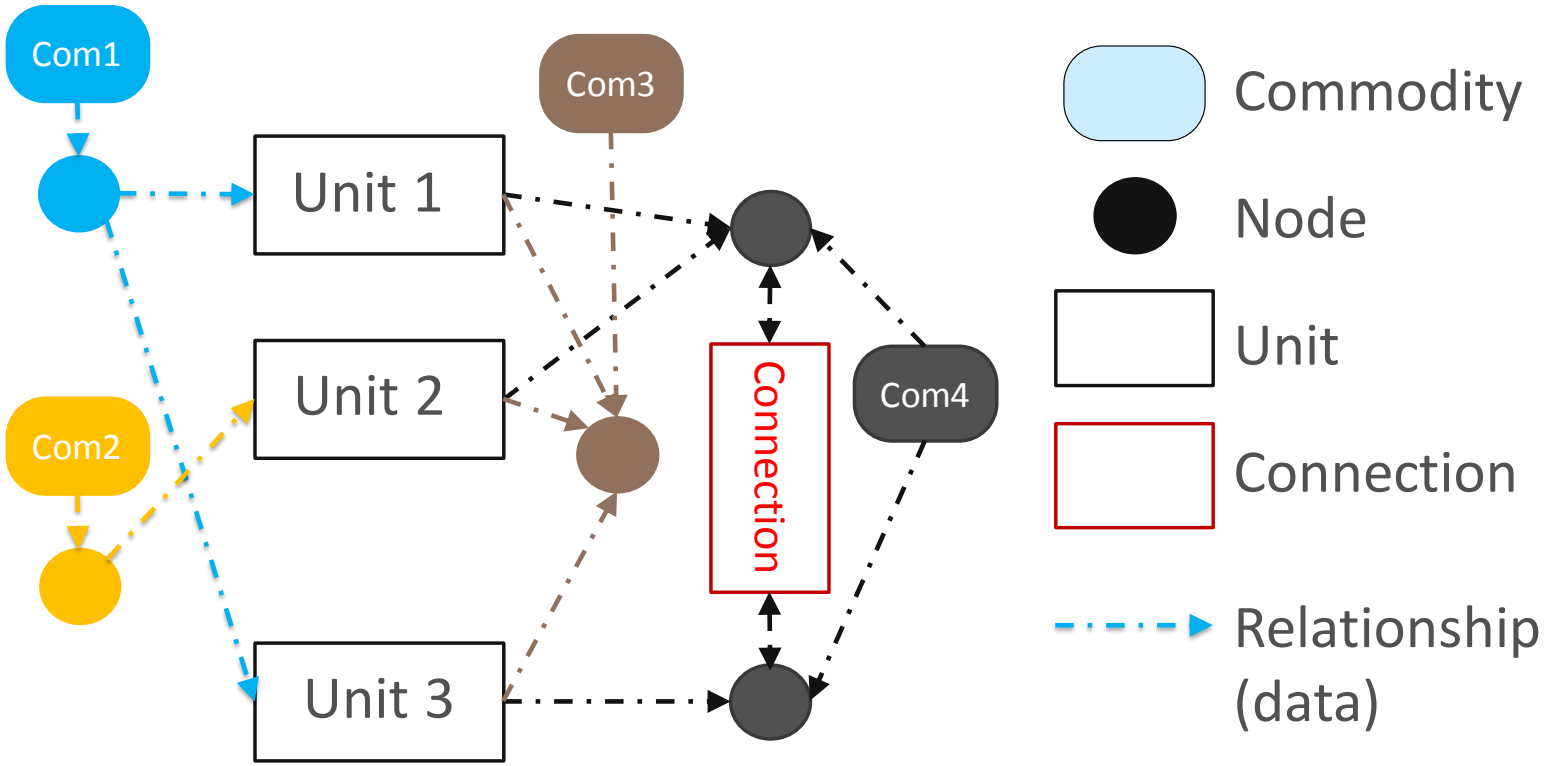
```
CoalPlant: 1
GasPlant: 1
CHPPlant: 1
ImportGas: 1
ImportCoal: 1
```

After calling JuMP_all_out() once, we get access to our entire database by referencing data objects by name without writing a single line of code.

```
@constraint(
    m,
    + flow[c, n, u, d, t]
    <=
    + avail_factor(unit=u, t=t)
        * unit_capacity(unit=u, commodity=c)
            * number_of_units(unit=u)
                * unit_conv_cap_to_flow(unit=u, commodity=c)
)
```


Spine

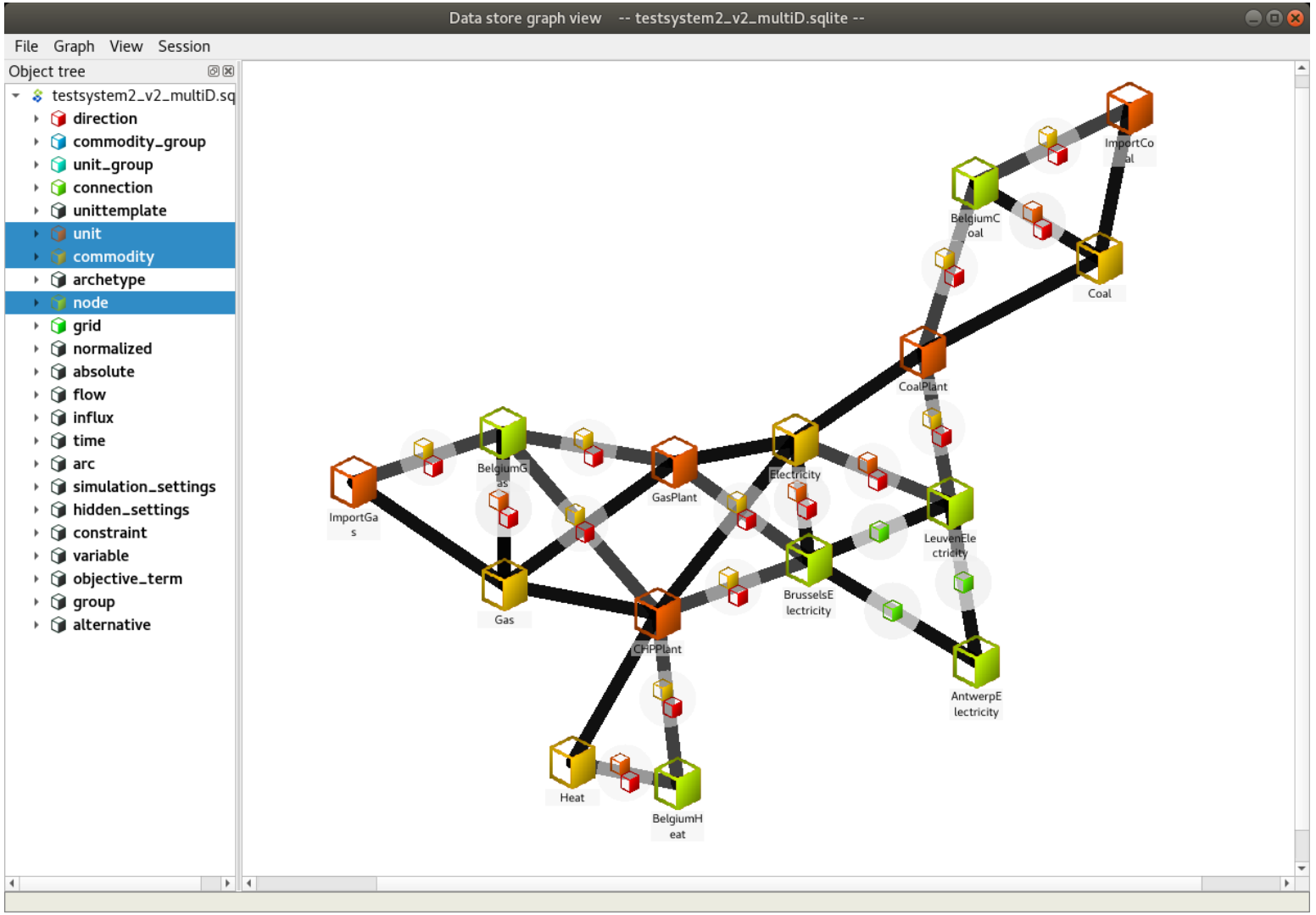
Generic Spine Model



- Arbitrary number of generic energy system networks represented within the generic data structure using commodities, nodes, units and connections
- Entities are related to each other using data
- Generic constraints implement features common to all energy networks, (e.g. conversion efficiency, balance)
- Commodity-specific network physics will be layered on top of generic constraints

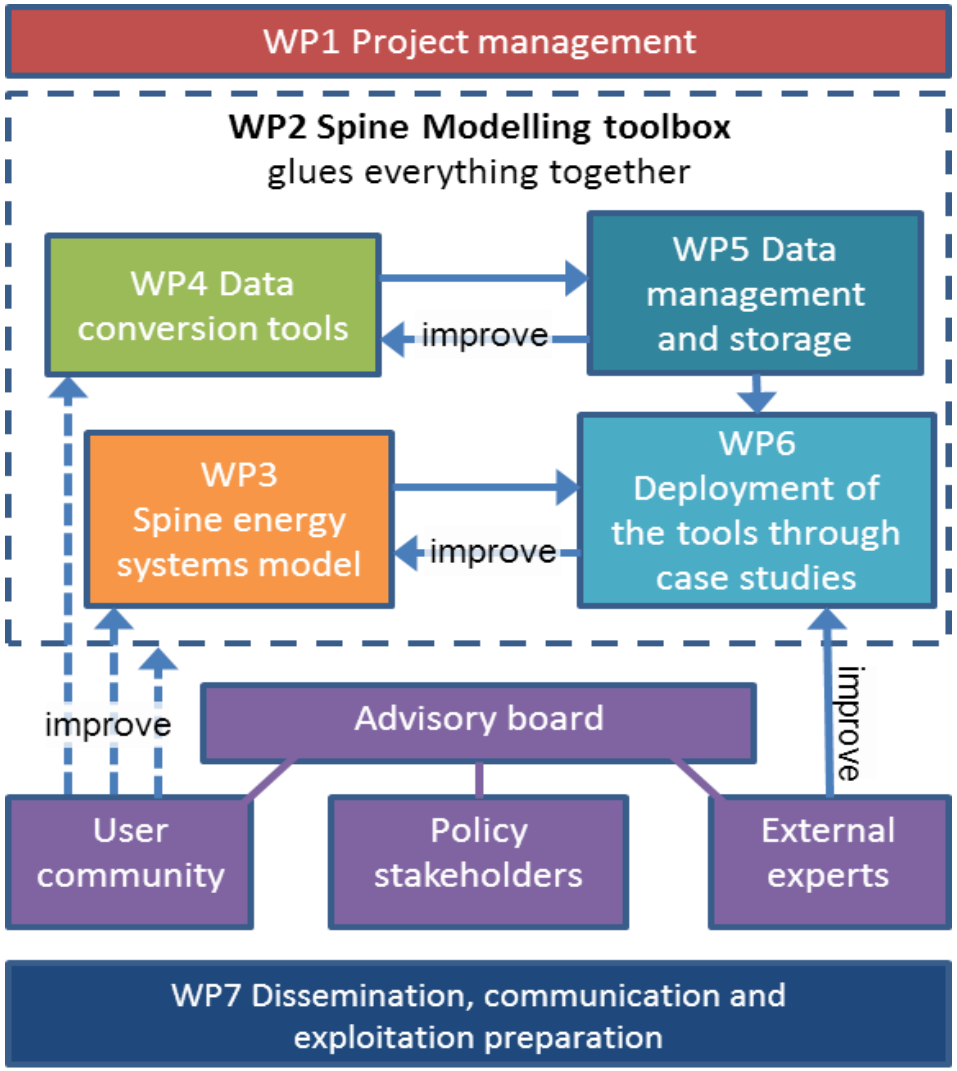
Spine

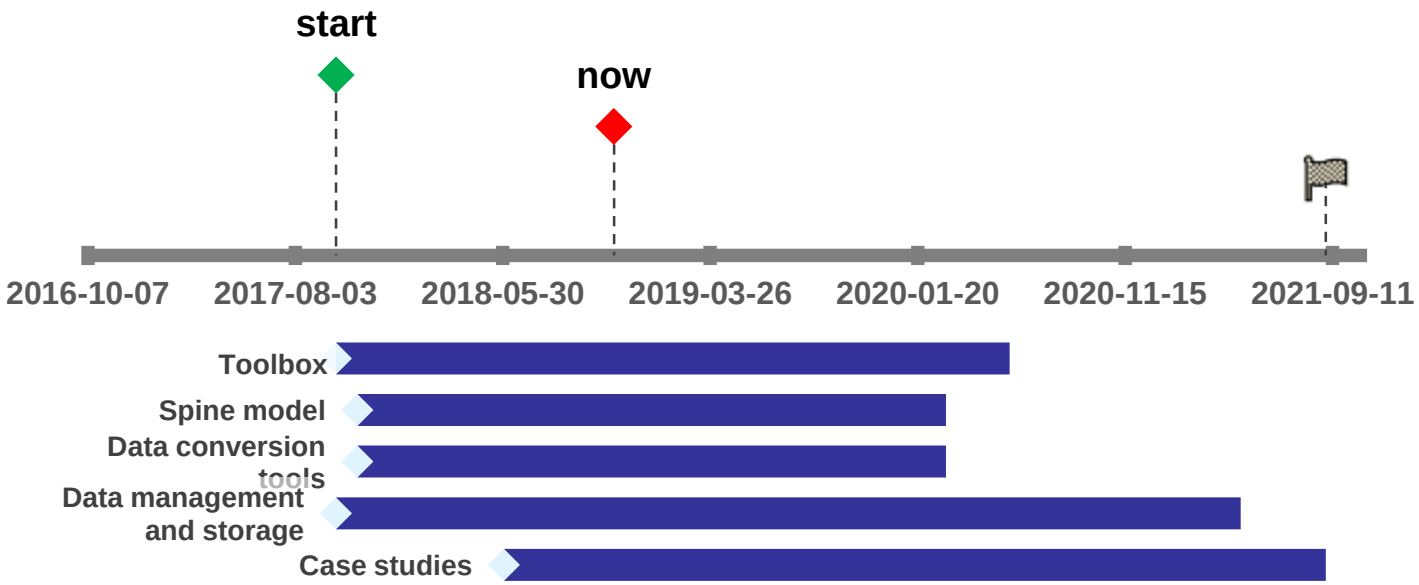
Planned Model Features



- Flexible, hierarchical representation of time, enabling commodities to be tracked at different resolutions in different stages
- Multi-stage approach facilitating flexible looping, rolling and stage variable stochastic and temporal resolution
- Feature and method based selection to control alternative constraint formulation
- Generic approach to decomposition and parallelisation

Spine Project Structure and Status





Collaboration

Objective: Build an active community of users, contributors and collaborators

Current and future collaborations

- NREL
  - Access to powersimulations.jl and powersystems.jl through the SpineToolbox, e.g. for calculation of Power Transmission Distribition Factors for DC load flow analysis
  - Translation of powersystems.jl datasets to Spine format
  - Implementation of the RTS-GMLC (https://github.com/GridMod/RTS-GMLC) reliability test system in Spine format
- Univ. of French West Indies: Spine-LARGE Project, "Using Spine to study wind and solar power integration in the Haitian electricity system”
- EnergyVille, KU Leuven: ELDEST, “Energy Policy Decision Support Toolbox”. Aims to develop an energy modeling toolbox to support energy policy-making.

Version 0.2 of the Spine Toolbox has been released on 17th January 2019

Follow and/or participate:

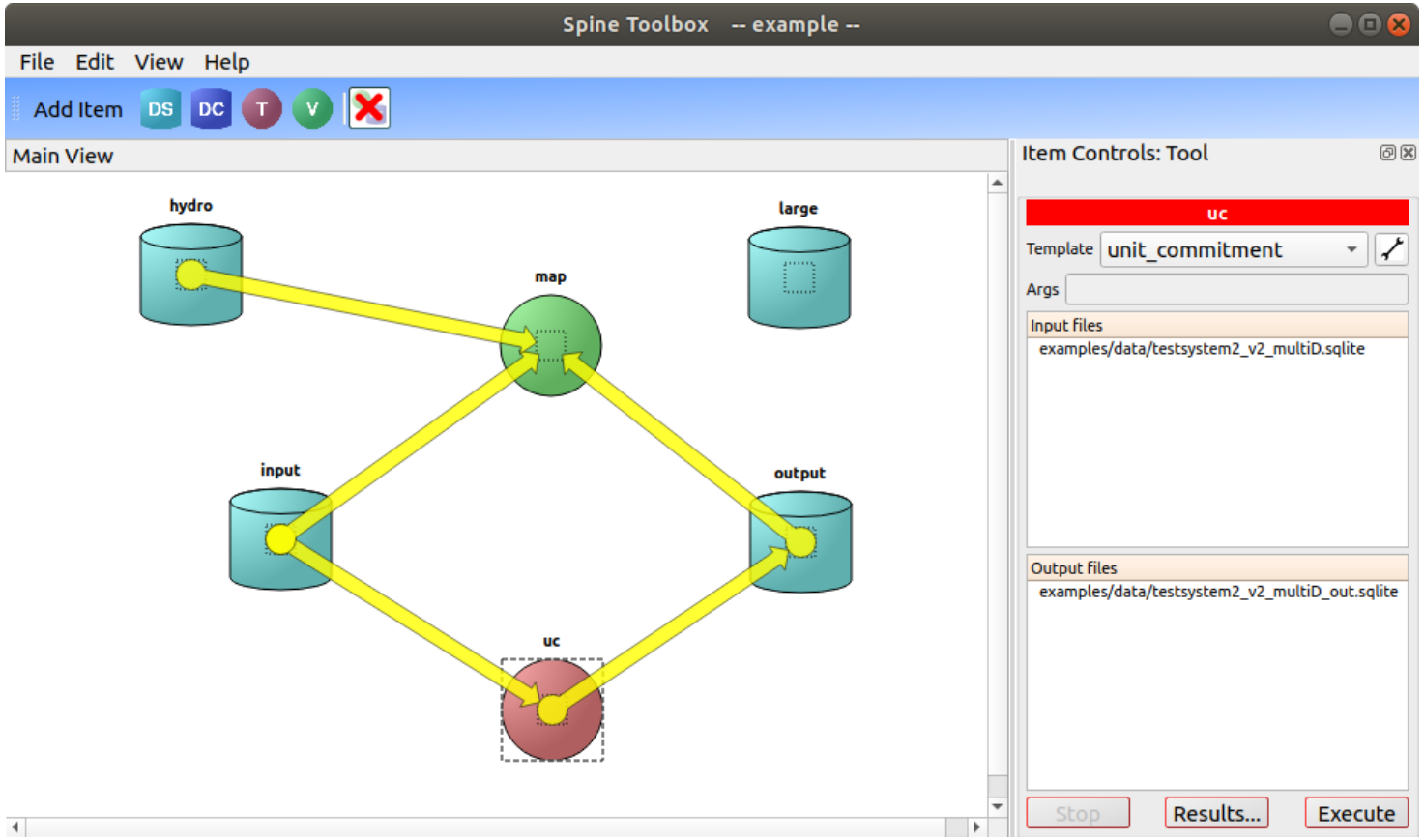
- Github: https://github.com/Spine-project
- Web: http://www.spine-model.org/
- Email: spine_info@vtt.fi

Possible participation use cases:

- **Spine model**: Exploiting the Spine Model (later)
  - Use and develop data sets
  - Use and develop the model
- **Model integration**: Exploiting open data interfaces and Spine "front-end"
  - Access/use/develop other models using the Spine toolbox
  - Integrate/translate existing models within the Spine framework – double win
- **Data**: Exploiting open, data management framework:
  - Providing/using/developing data sets

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N. 774629.

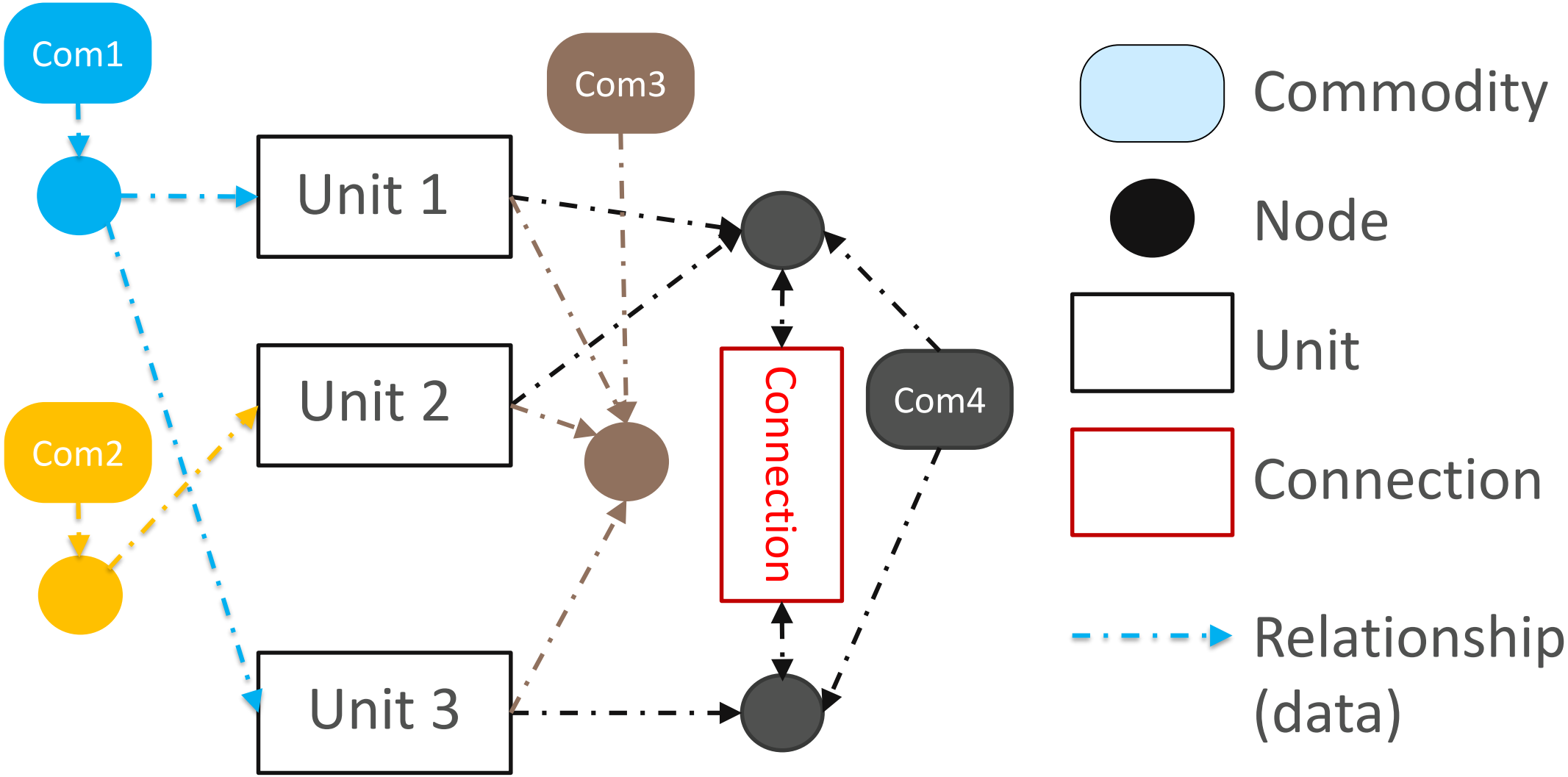
Spine Toolbox: Main View



- GUI visualizing data stores, tools (models/scripts), views (viewing scripts)
- Drawing of connections to link data stores to tools or views
- API allows connecting to different types of databases (e.g., MySQL, SQLite, etc.)

Spine

Generic Spine Model



Spine