Open LCA Software Advances

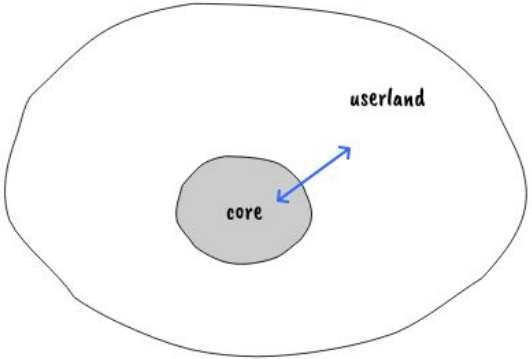
Michael Srocka, GreenDelta
Brightcon 2022

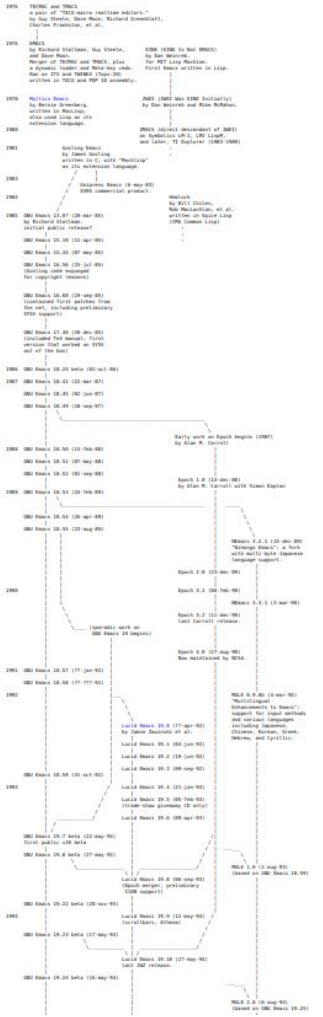
(openLCA software advances)

see openLCA.org

Vision for LCA tooling, maybe ...

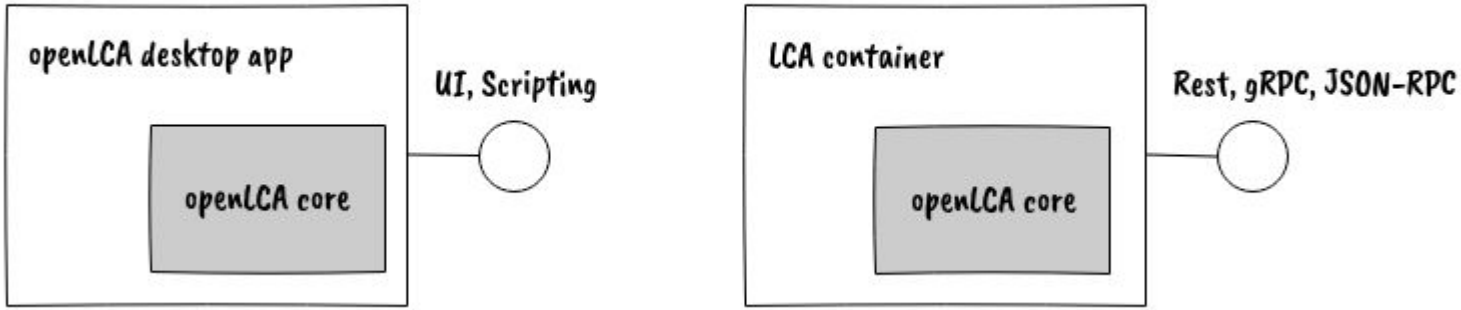
- ... a bit like Emacs:
  - provide an open **core** that just works
  - with a communication concept, the "buffer"
  - build creative things on top of it, in **userland**
  - move best things back into the core
  - iterate





https://www.jwz.org/doc/emacs-timeline.html

openLCA: desktop & web-services



Data model & communication protocol

Data model and communication protocol

- Java API: access the internals
  - Java, Kotlin, Scala, ...
  - Jython interpreter in openLCA
- external interface: openLCA schema

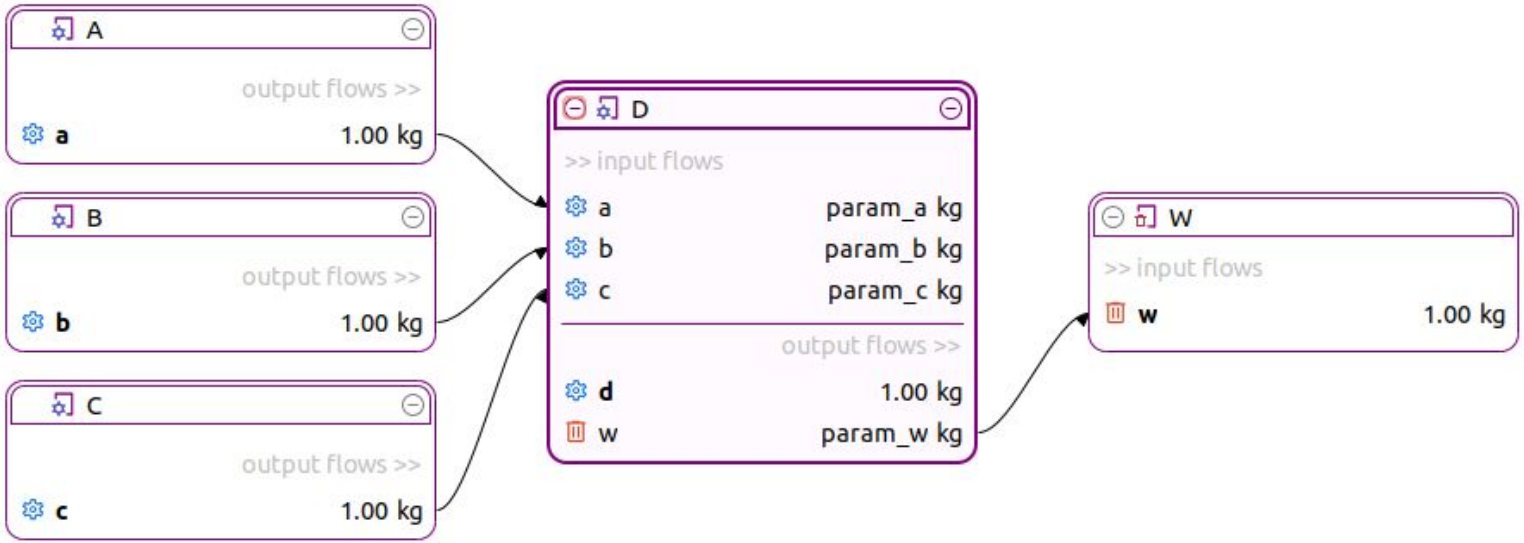
```
mass = db.getForName(FlowProperty, 'Mass')
brick = db.getForName(Flow, 'brick, at plant')
wall = db.insert(Flow.product('Wall element', mass))
process = Process.of('Wall something', wall)
process.input(brick, 1000) # kg
db.insert(process)

App.open(process)
```

openLCA schema

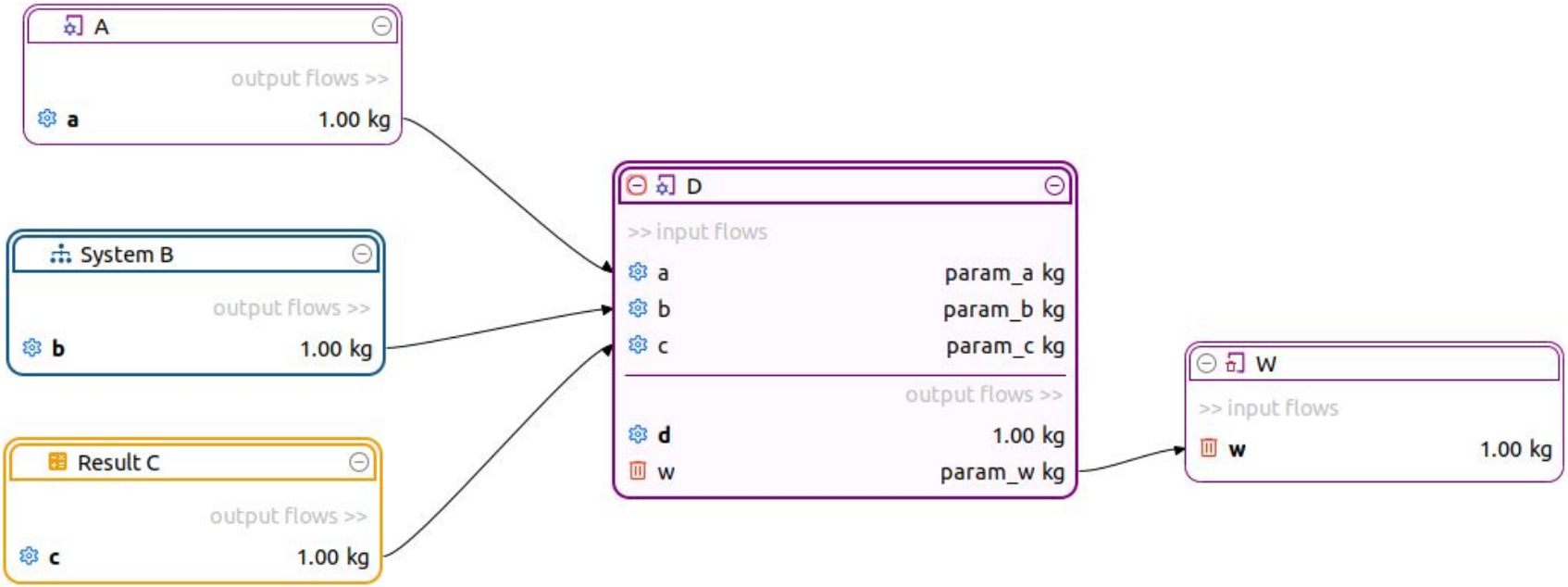
- Version 2: github.com/GreenDelta/olca-schema
- a meta-format spec for:
  - Json, Protocol Buffers, Python classes ...
- used for:
  - file based data exchange
  - JSON-RPC, gRPC, Rest APIs
  - Git based data exchange
  - ...

```
cd olca-schema/osch && \
  go build && \
  ./osch all

# or

pip install olca-schema
```

Example: product systems



Product systems



Results

- stored LCI results, LCIA results only; or both
- e.g. imported from EPDs, or used to build EPDs
- ILCD+EPD & OpenEPD support

https://english.lem.pl/gallery/mroz-drawings/category/37-cyberiad-1972-wl

Parameter redefinitions

- redefinitions applied per calculations ...
  - scopes: global, processes, LCIA categories

Parameters

Context	Parameter	Amount
global	param_a	0.1
global	param_b	0.2
D	param_c	0.3
D	param_w	0.4

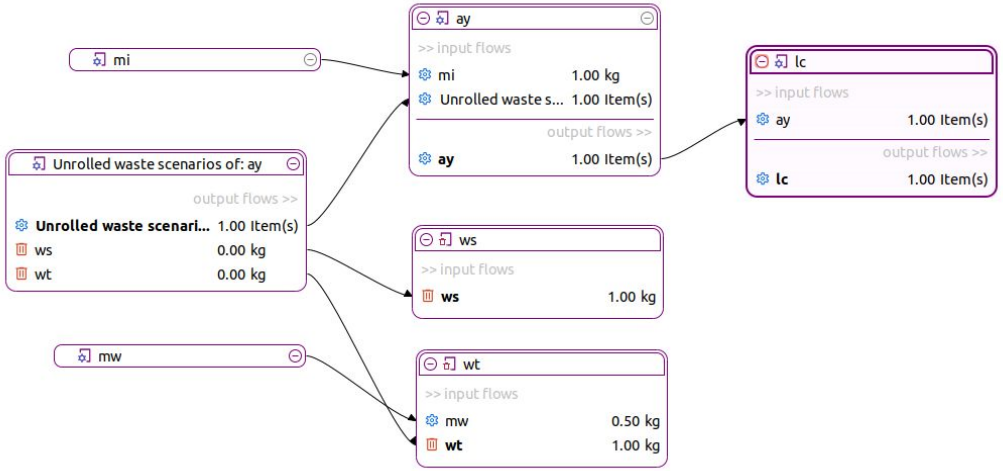
Contribution	Process	Required amount	Result
100.00%	D	1.00000 kg	2.00000 kg
20.00%	W	0.40000 kg	0.40000 kg
15.00%	Result C	0.30000 kg	0.30000 kg
10.00%	System B	0.20000 kg	0.20000 kg
05.00%	A	0.10000 kg	0.10000 kg

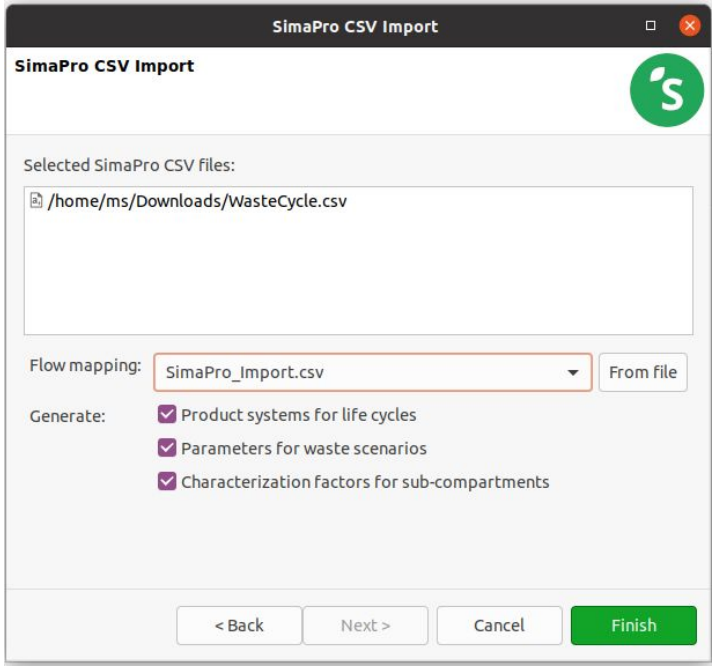
https://english.lem.pl/gallery/mroz-drawings/category/37-cyberiad-1972-wl

Parameter redefinitions: stateless services

- each parameter can be redefined in a calculation request
- application in memory only
- can be stored in product systems
- sub-systems:
  - redef. of outer system overwrites redef. of sub-system
  - but not when redef. in sub-system is protected

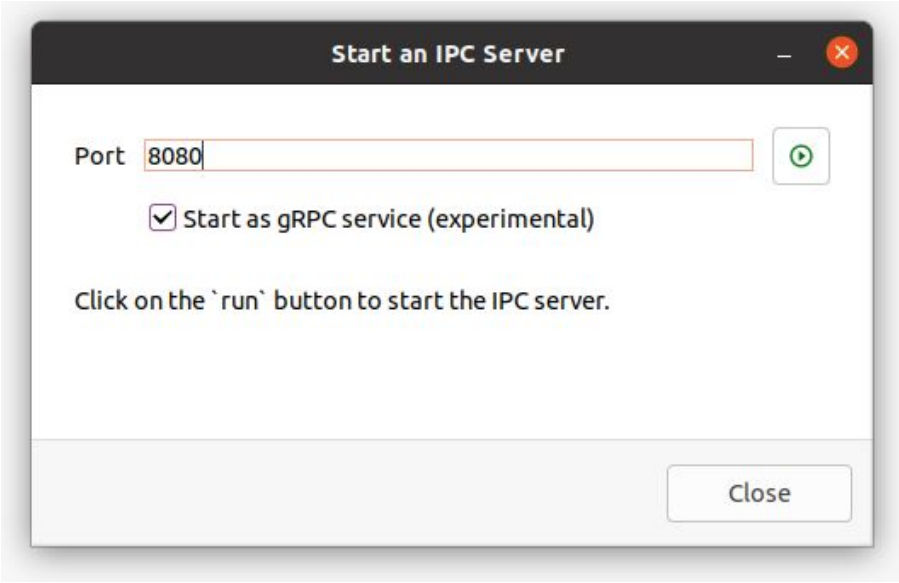
Protected parameters redefs.





openLCA schema - next

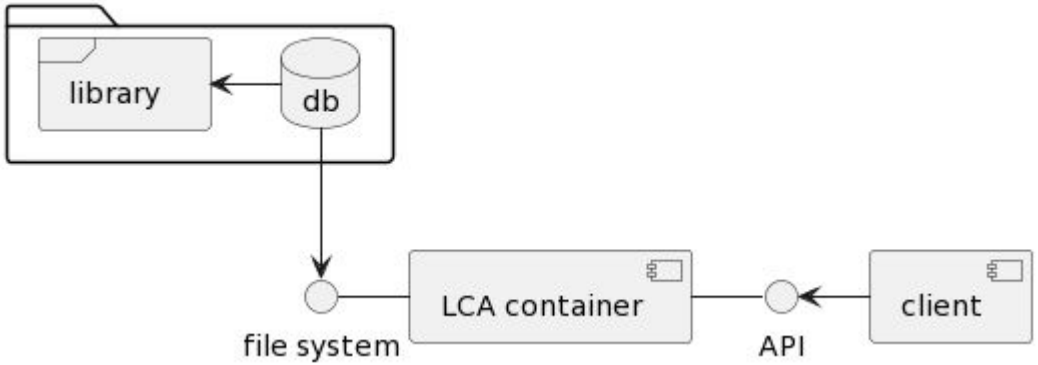
- spec for calculation setups and results
  - simple results (LCI, LCIA, LCC, SLCIA, ...)
  - contribution results
  - upstream trees
  - graphs for Sankey diagrams
  - ...
- extensions
- for service integrations



Service integrations

LCA containers

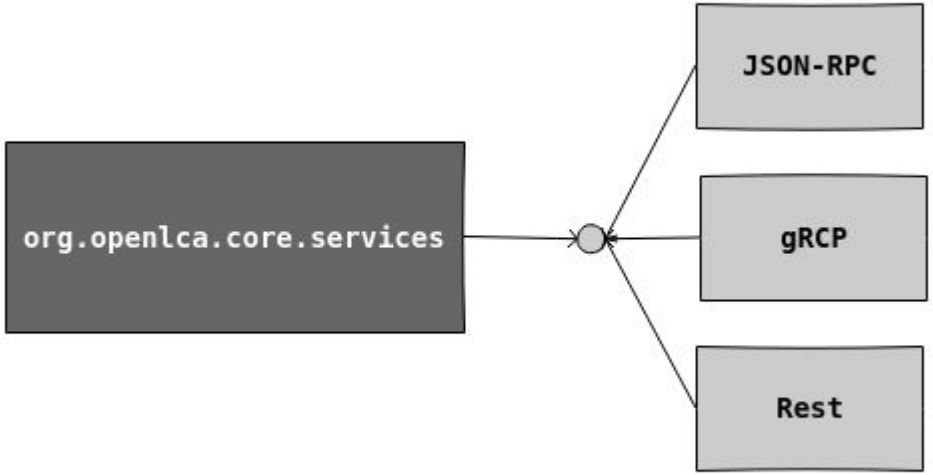
- container: server + service backend
- mount file system: database + possible libraries
- expose API



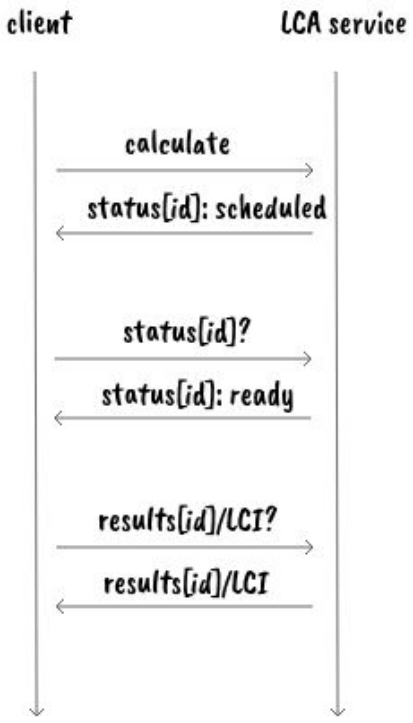
- works with every openLCA database and multiple service frontends
- model pipelines

Service backend

- in development: a backend for multiple service frontends
  - calculation queue, result polling, JSON de/serialization, ...



Calculation queue



Client APIs

```
system = client.get_descriptor(ProductSystem, uid)
setup = ipc.CalculationSetup(
    product_system=system,
    amount=500,
    unit=units.unit_ref('g'),
    parameters=[
        # ...
    ])
result = client.calculate(setup)
# ...
client.dispose(result)
```

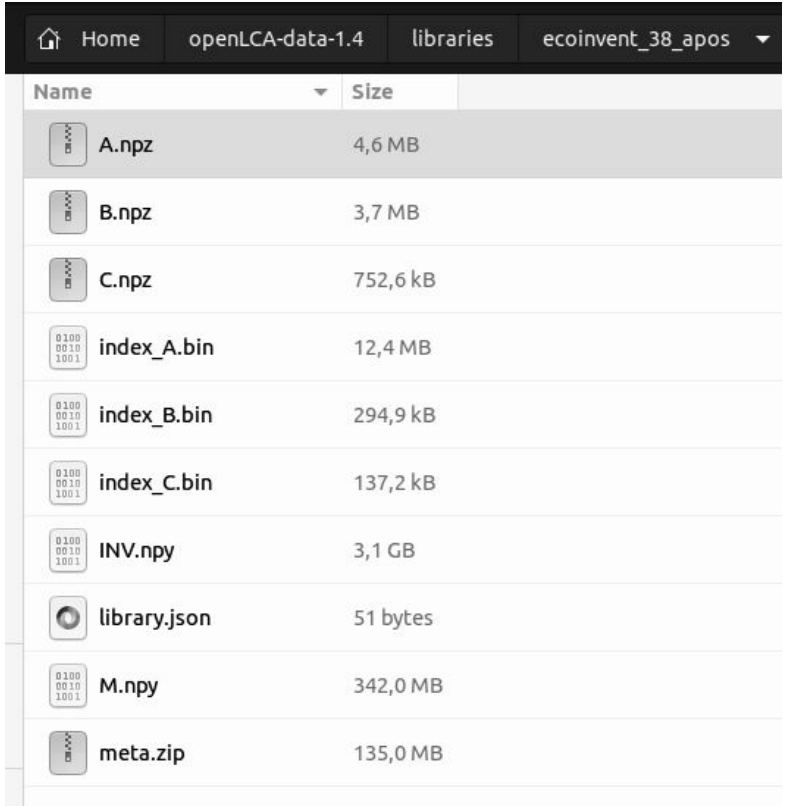
Libraries

Libraries

- a vertical core extension
  - data packages with pre-processed matrices
- improvements
  - calculation speed & memory usage
  - graphical editing
  - data sharing
  - ...

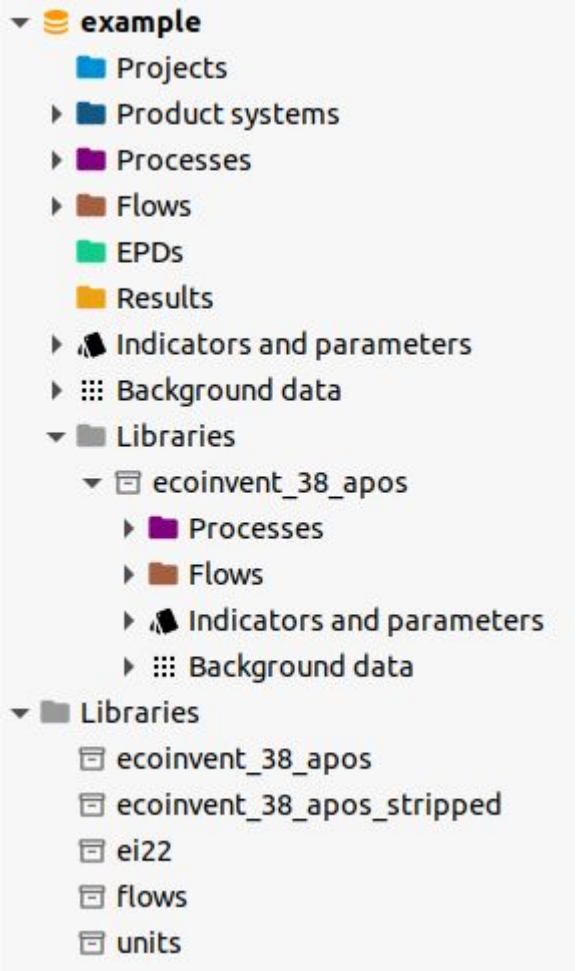
Libraries

- read-only package format
- based on open standards:
  - matrices: NPY, NPZ
  - indices: CSV, Protocol Buffers
  - meta-data: openLCA schema



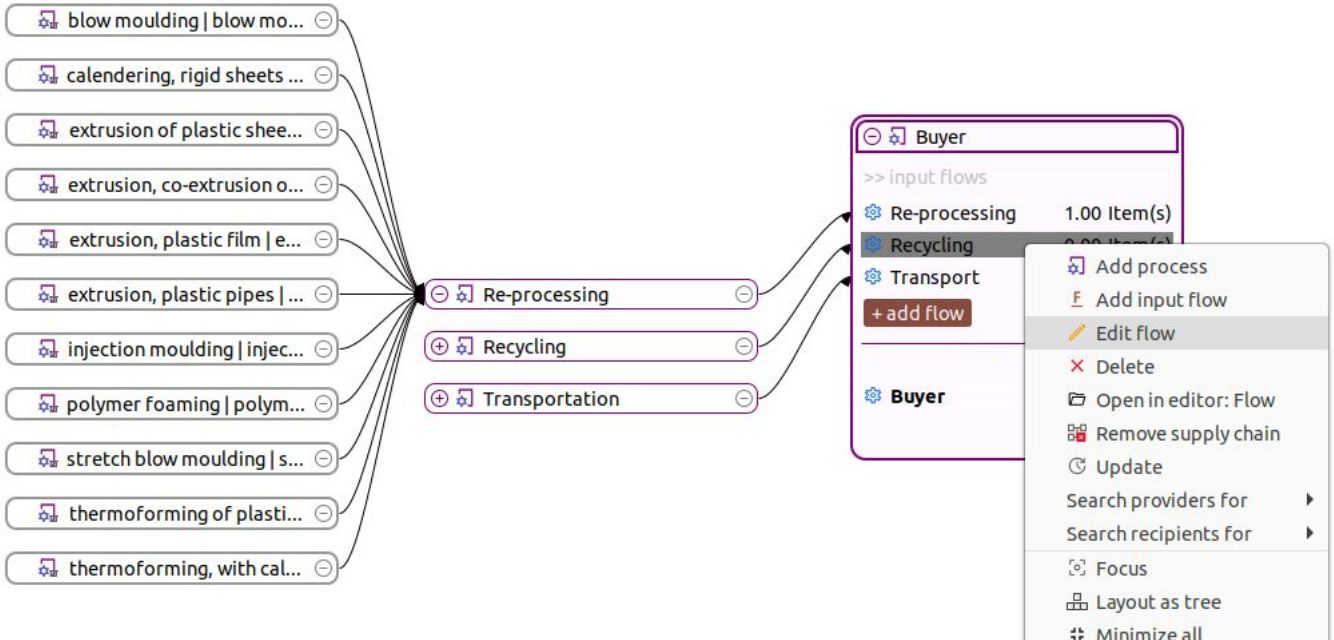
openLCA integration

- works with sparse and dense systems
- library objects usable like any other object
- support recursive dependencies for reference data (e.g. units, flow lists)



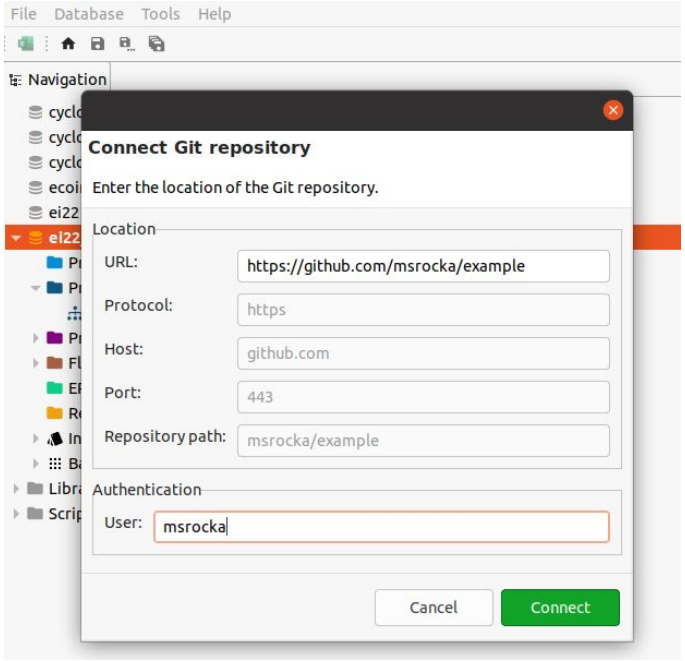
Graphical editing

- draw your LCA modell; libraries hide details
- but you get all details in the results



Data sharing

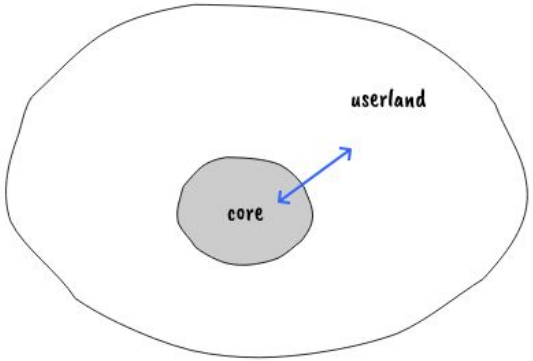
- supported in: openLCA Collaboration Server 2.0
- builds on top of openLCA schema & Git
- openLCA db <-> sync with local repo <-> sync with remote
- with libraries: only foreground system is shared



Conclusions

Conclusions

- "dogfooding": building new things in userland first
- use open standards: Json, Npy, ... SQLite?
- make the data model extendible
  (like key-value-pairs in Brightway)



Thank you!

srocka@greendelta.com