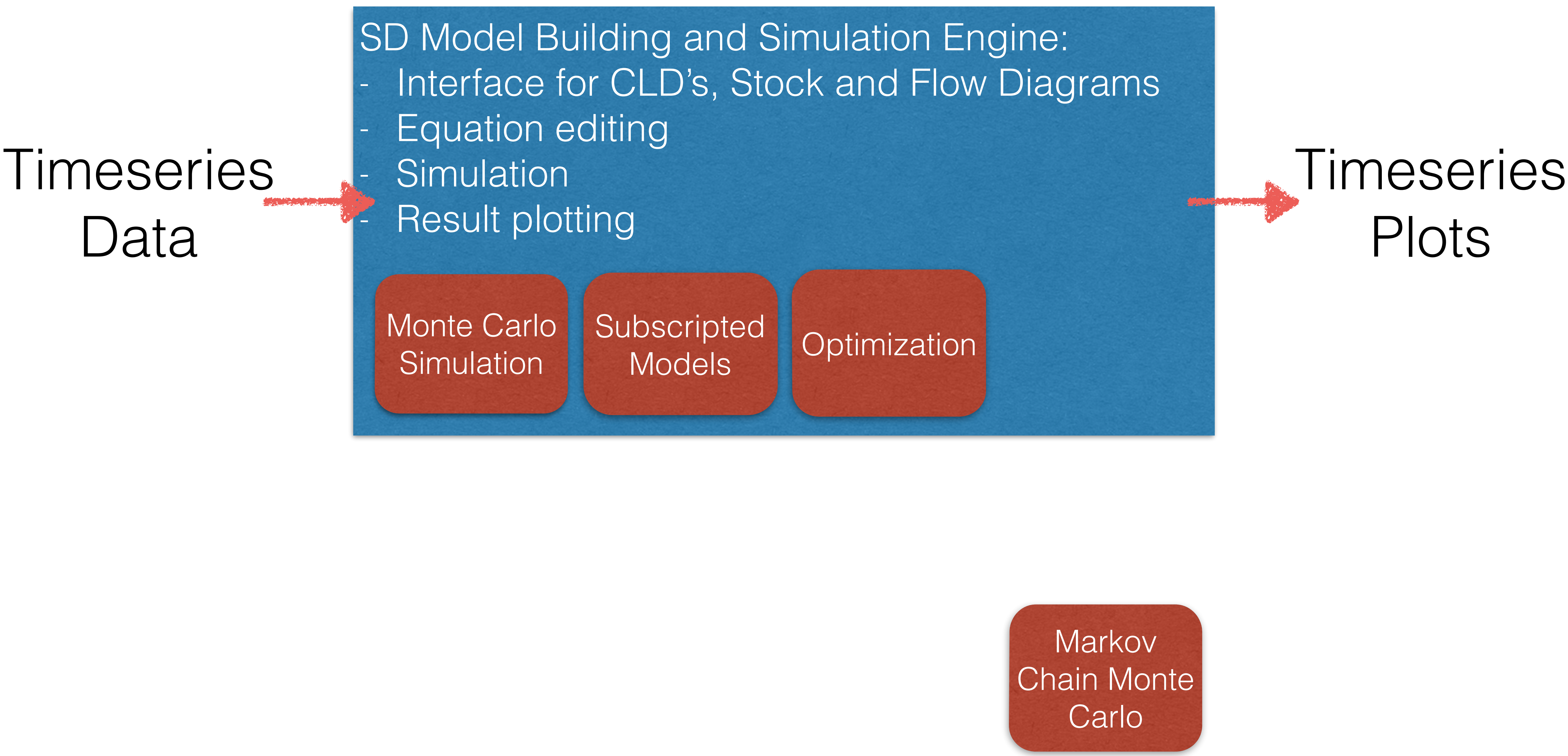
pysd:
System Dynamics Models in Python

purpose, capabilities, examples, structure, development plan

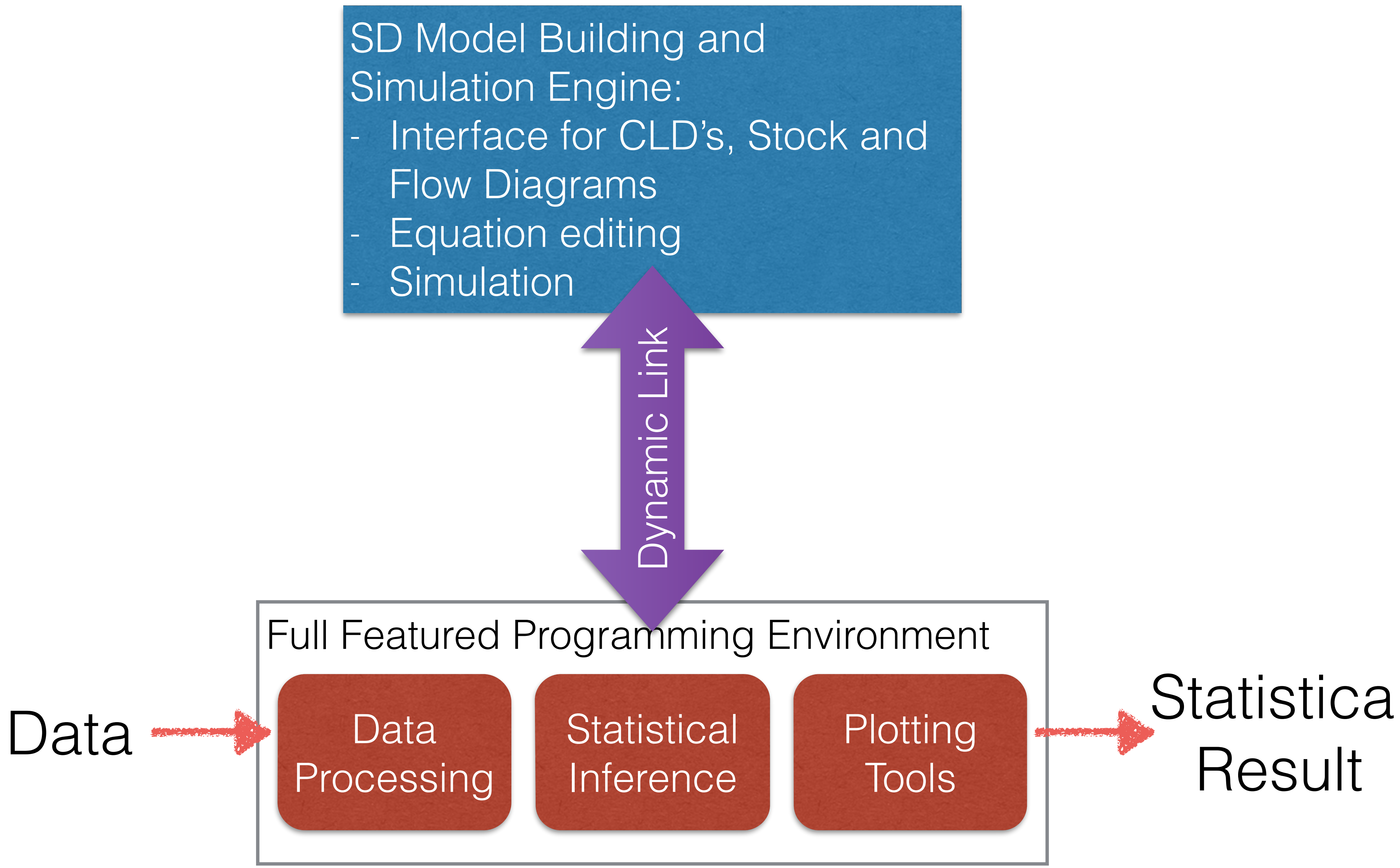
Motivation

- More powerful analysis tools
- Better ways to structure multi-part models
- Tighter integration with “big data” (including non-timeseries data)
- Integration with other simulation and analysis methods
- Better communication of analysis method and results (replicability!)

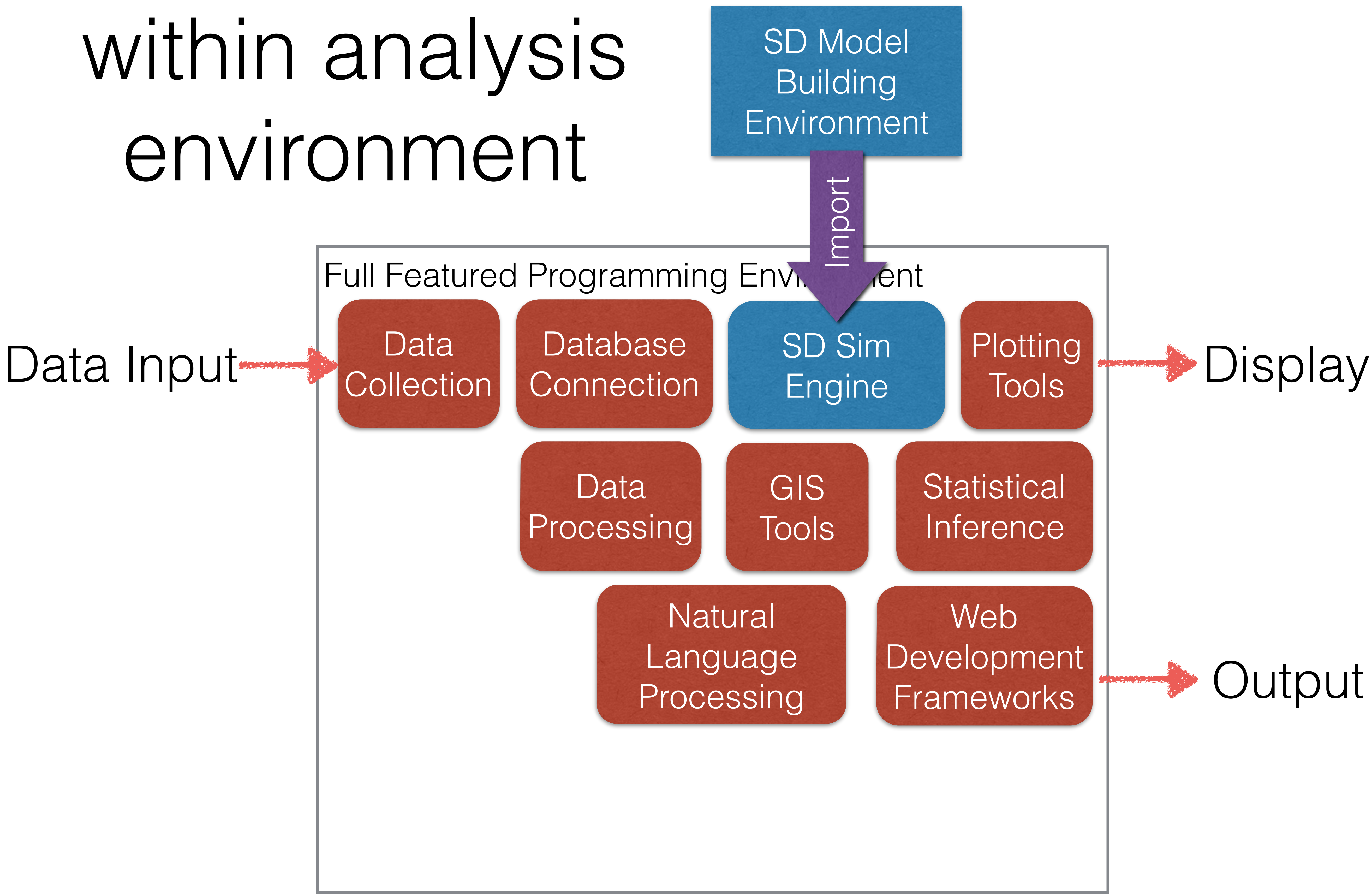
Traditional: Bring analysis capabilities into SD software



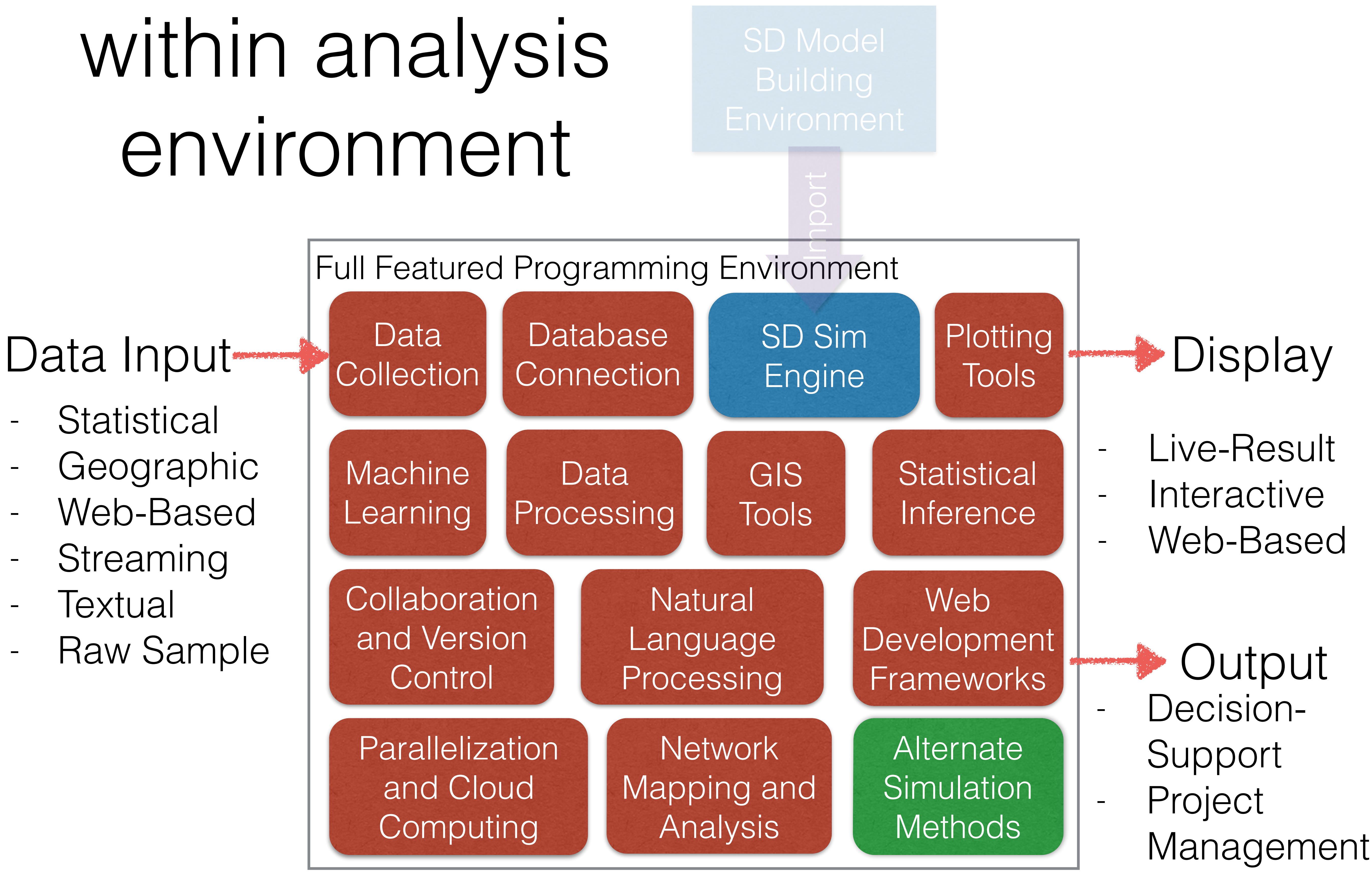
State of the art: Interface SD software with external analysis tools



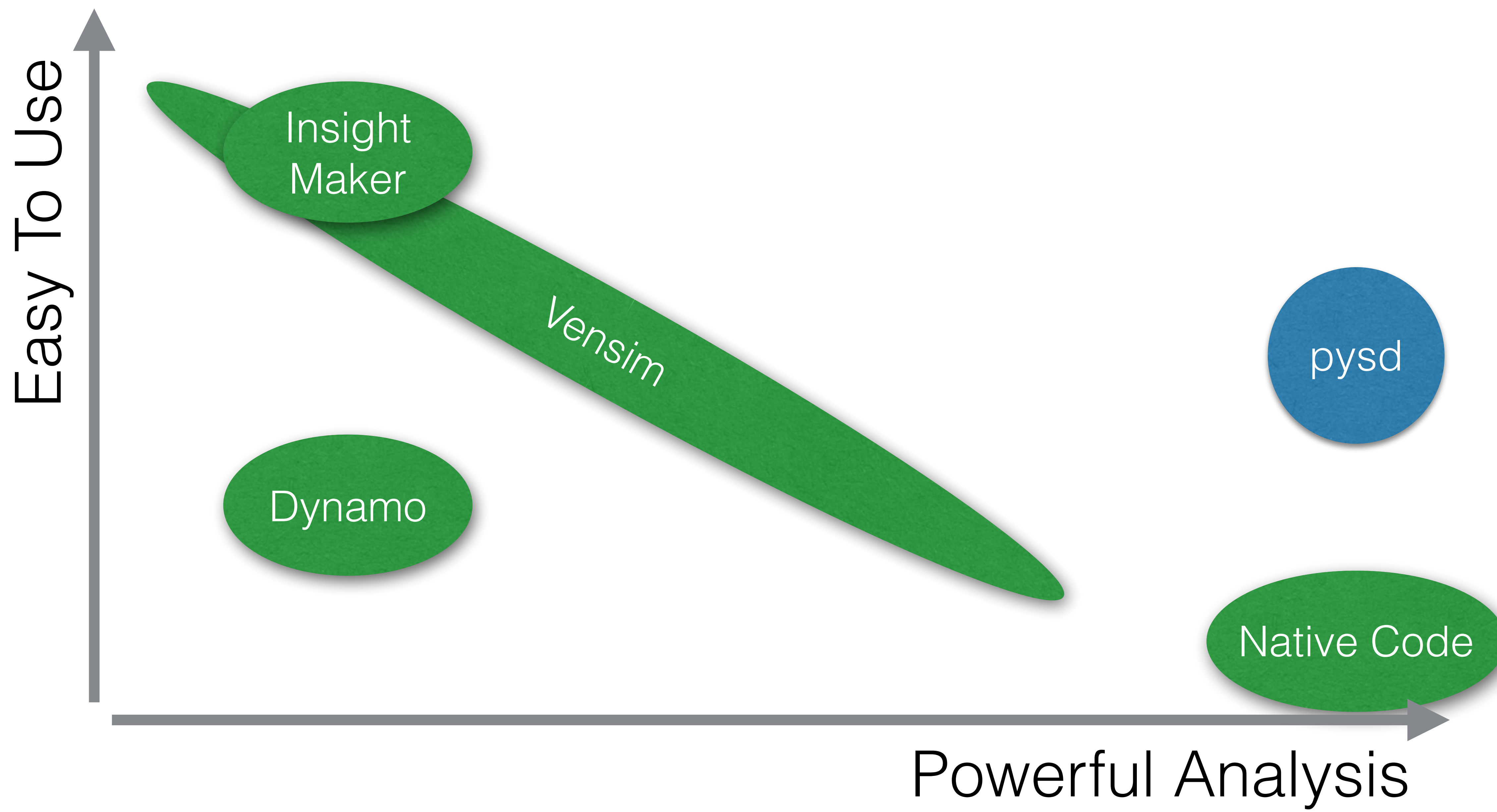
pysd: Simulate within analysis environment



pysd: Simulate within analysis environment



An incomplete map of the SD software space

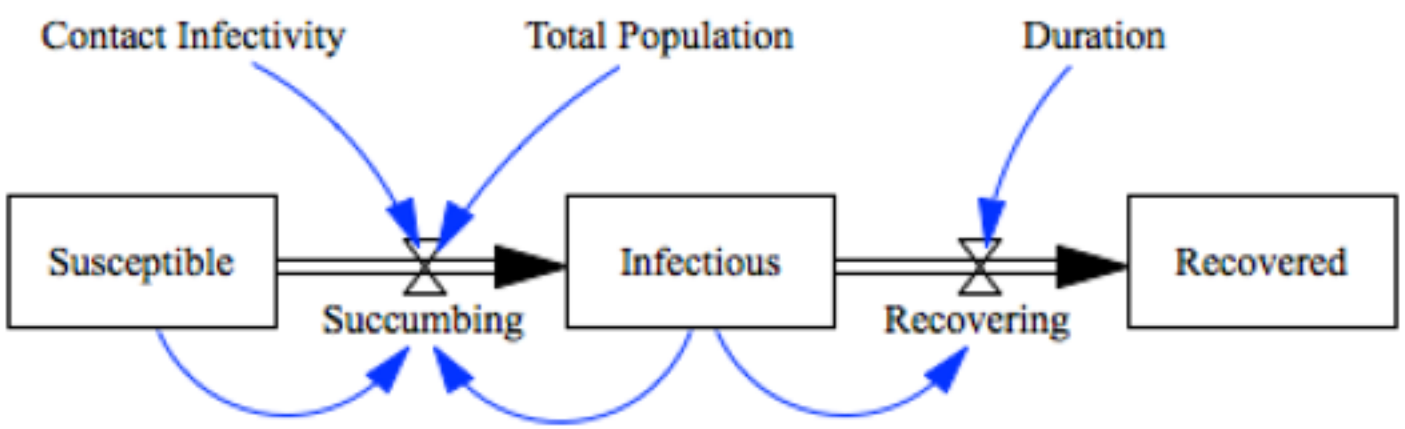


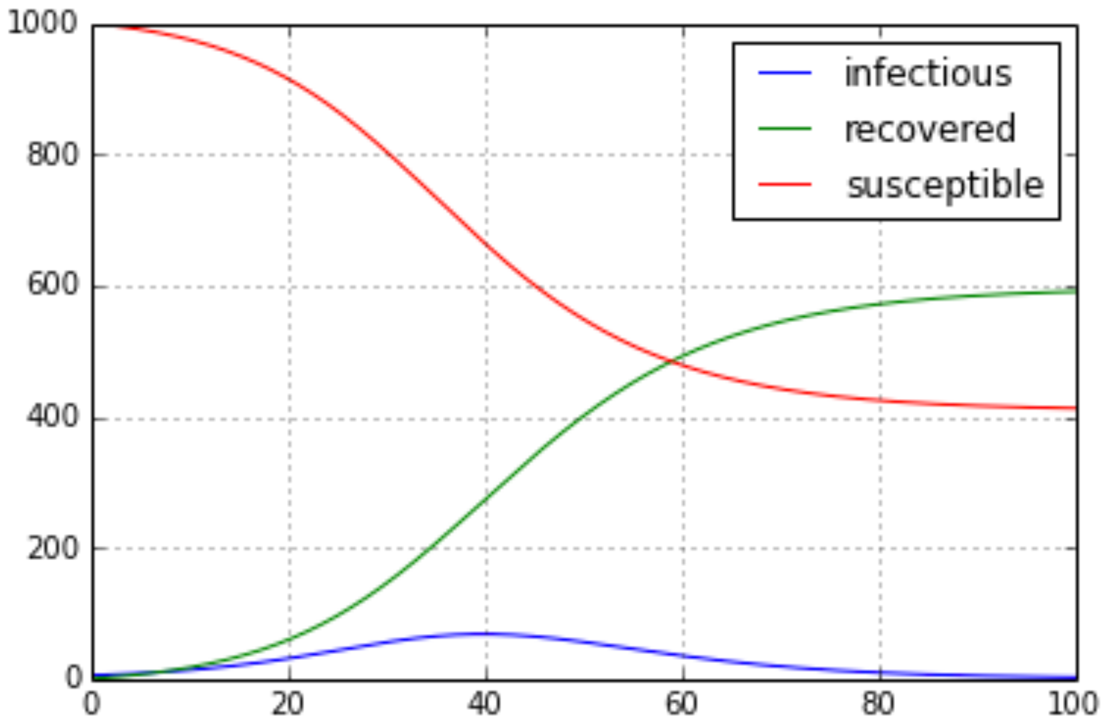
Status

- Functioning prototype available at: https://github.com/JamesPHoughton/pysd
- Supports subset of XMILE, Vensim commands
- Install through python package exchange:
  >> pip install pysd

Basic Usage

```
import pysd
model = pysd.read_vensim('SIR.mdl')
model.run().plot()
```





Mechanics of Importing

SIR.mdl:

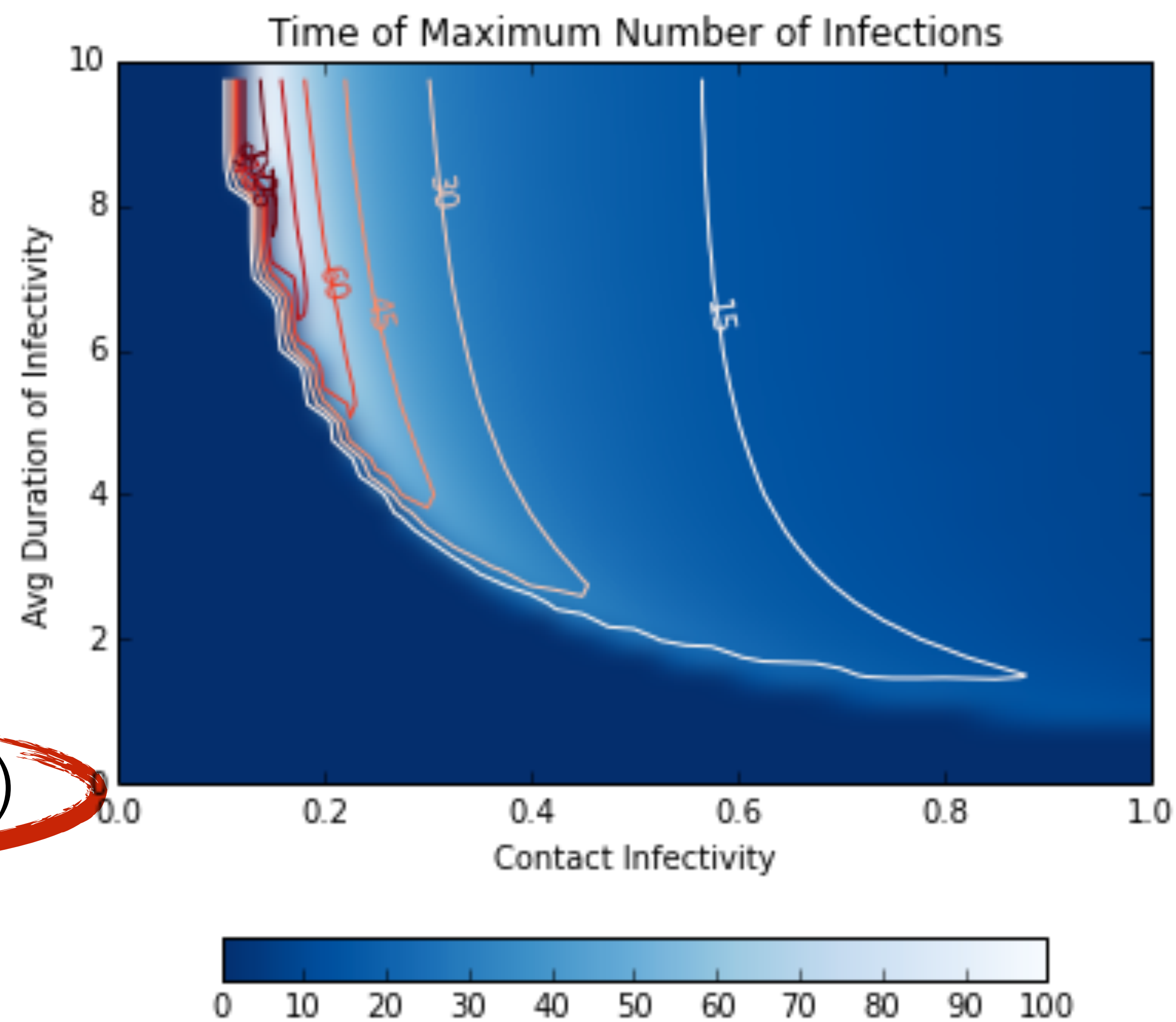
```
Contact Infectivity=
	0.3
	~	Persons/Persons/Day
	~		|

Duration=
	5
	~	Days
	~		|

Infectious= INTEG (
	Succumbing-Recovering,
		5)
	~	Persons
	~		|

Recovered= INTEG (
	Recovering,
		0)
	~	Persons
	~		|

Recovering=
	Infectious/Duration
	~	Persons/Day
	~		|
```

Import

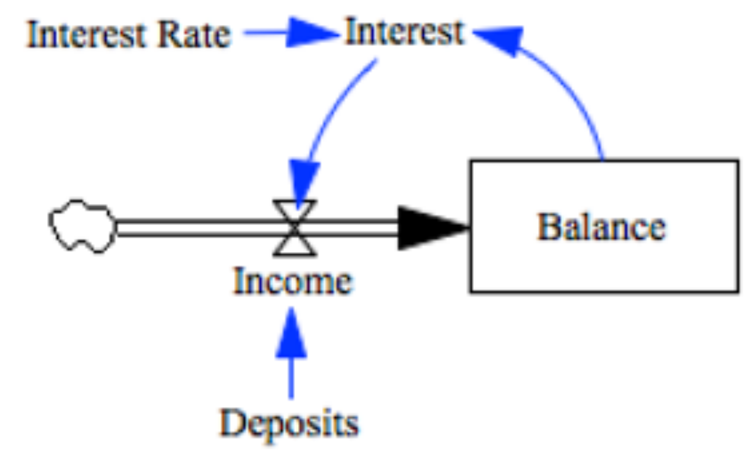
```python
def model_function(stocks, t):
    infectious, recovered, susceptible, = stocks

    total_population = 1000
    duration = 5
    recovering = infectious/duration
    contact_infectivity = 0.3
    succumbing = susceptible*infectious/
                      total_population*
                      contact_infectivity
    dinfectious_dt = succumbing-recovering
    dsusceptible_dt = -succumbing
    drecovered_dt = recovering

    return [dinfectious_dt, drecovered_dt,
            dsusceptible_dt]

odeint(model_function, initial_values, tseries)
```

Model Exploration

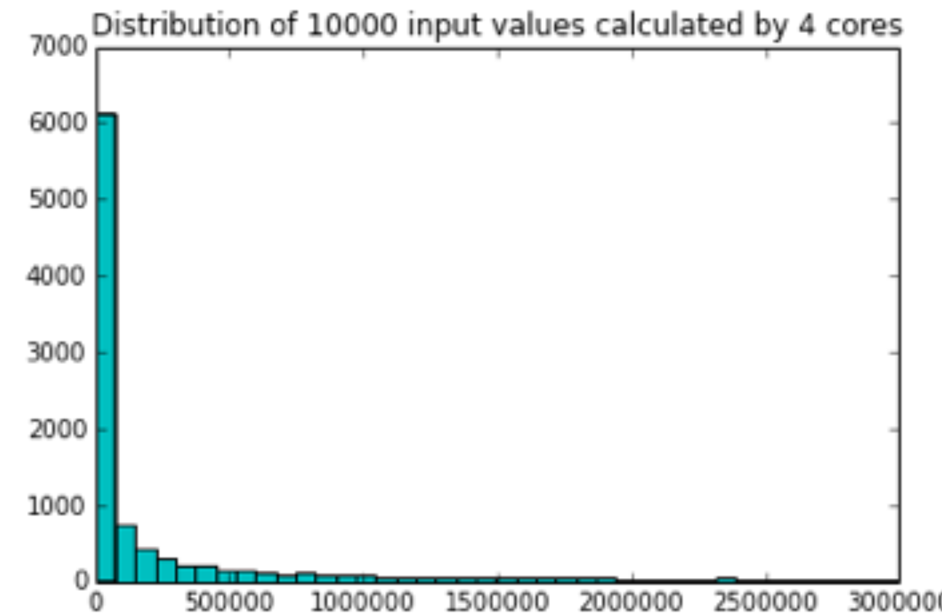
```
ci = np.arange(0,1,.025)
d = np.arange(.5,10,.25)
ci_grid, d_grid = np.meshgrid(ci, d)

def tmax(ci, d):
    ps = {'contact_infectivity':ci,
          'duration':d}
    stocks = model.run(params=ps)
    return stocks['infectious'].idxmax()

vtmax = np.vectorize(tmax)
tmax_grid = vtmax(ci_grid, d_grid)
```



Parallelization




```
model = pysd.read_vensim('bank_balance.mdl')
interest_rate = np.random.uniform(high=.1, size=10000)

def final(rate):
    ps={'interest_rate':rate}
    stocks = model.run(params=ps)
    return stocks['balance'].iloc[-1]
```



Series Monte Carlo:

```
[final(rate) for rate in interest_rate]
```

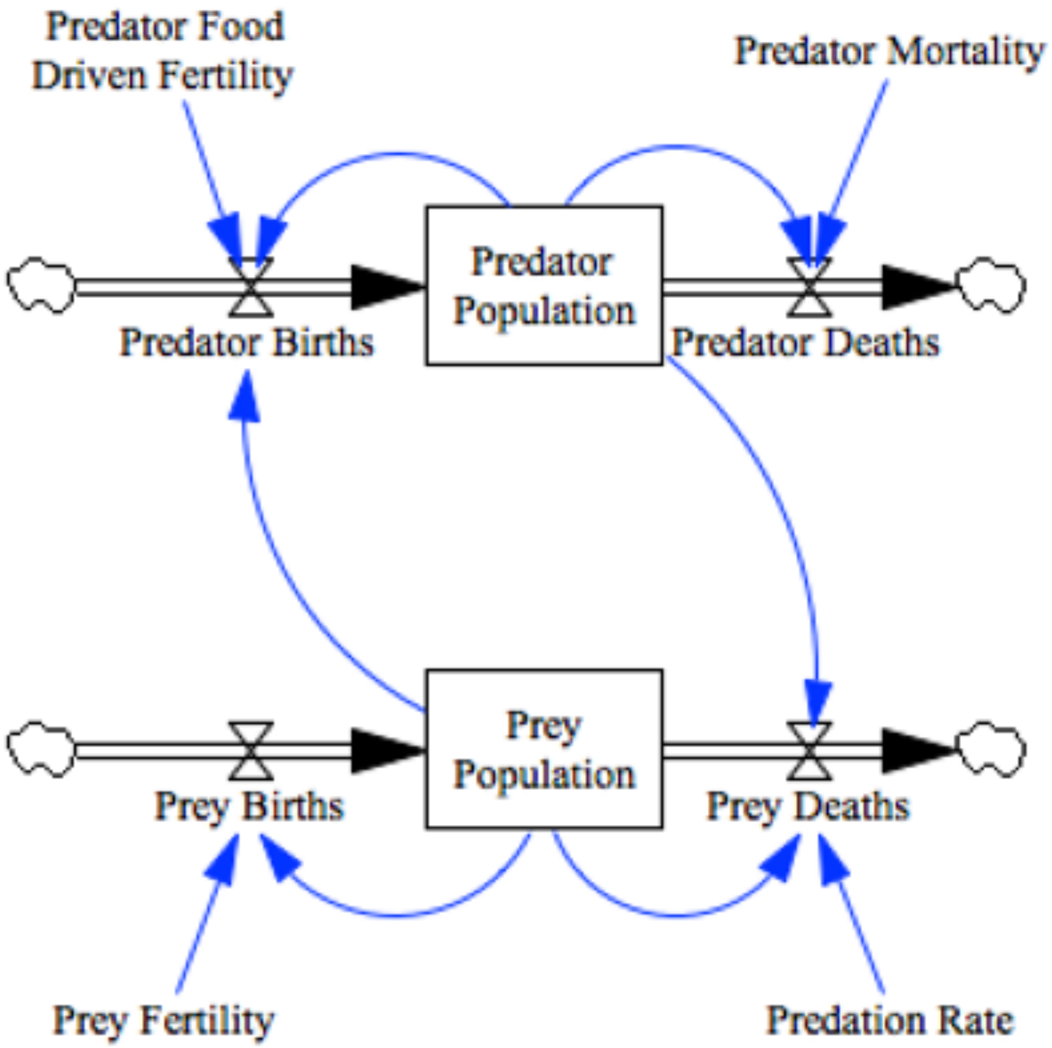
Execution Time: 11s

Parallel Monte Carlo:

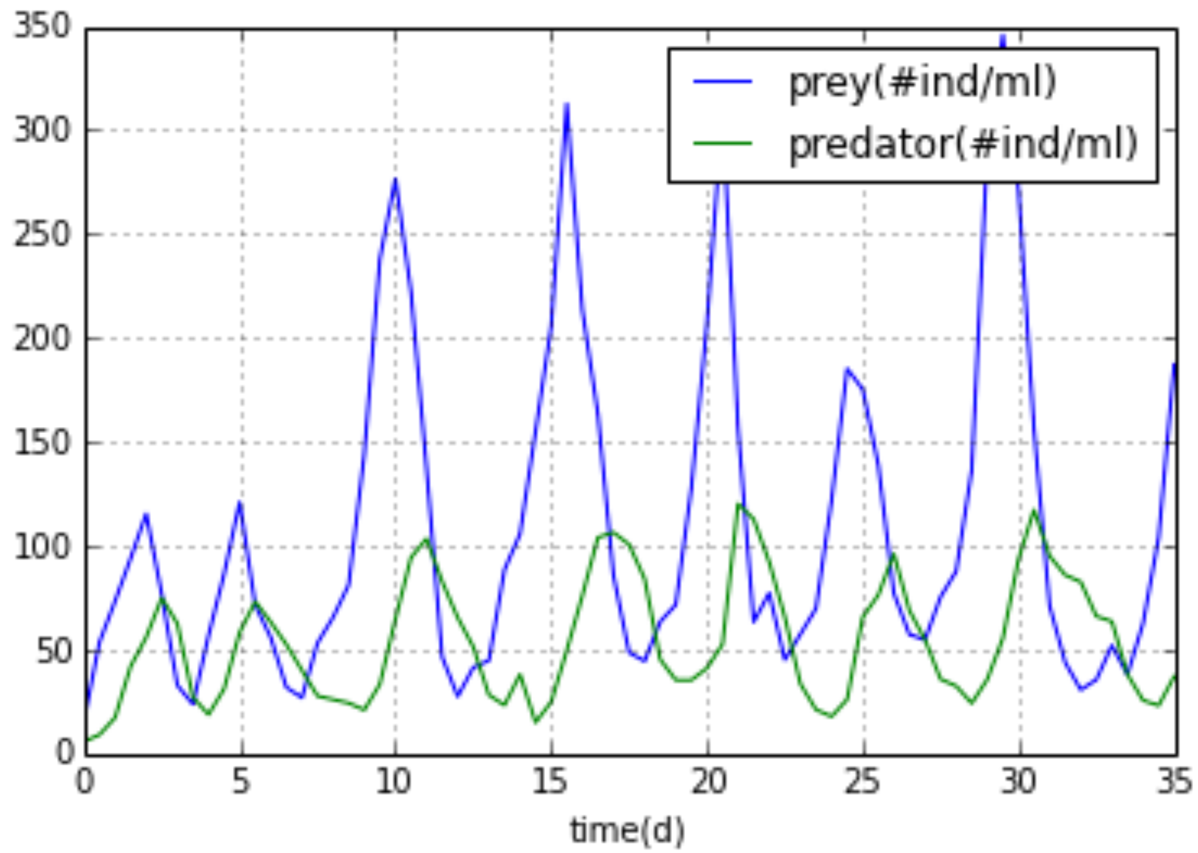
```
from IPython.parallel import Client
dview = Client().cli[:]
dview.push(model)
dview.map(final, interest_rate)
```

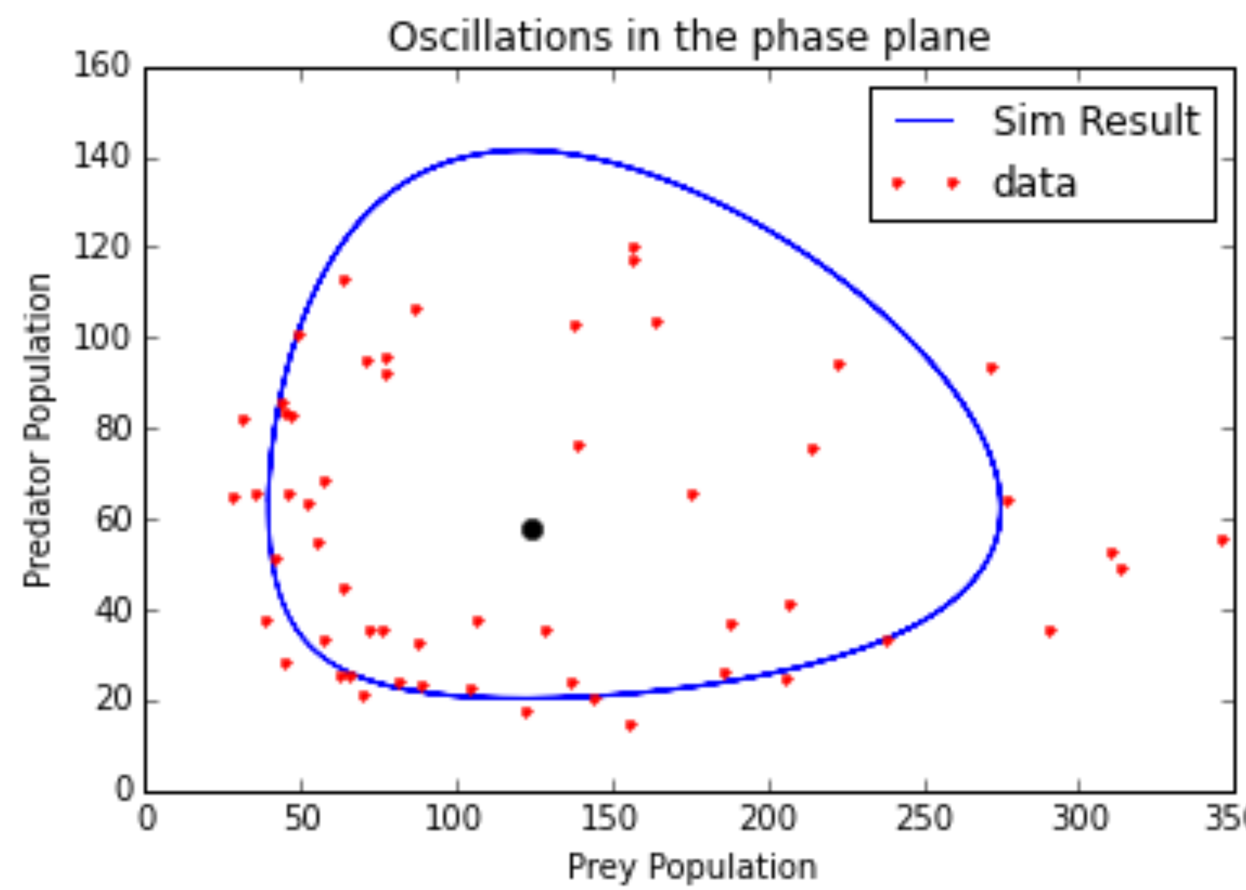
Execution Time: 2.9s

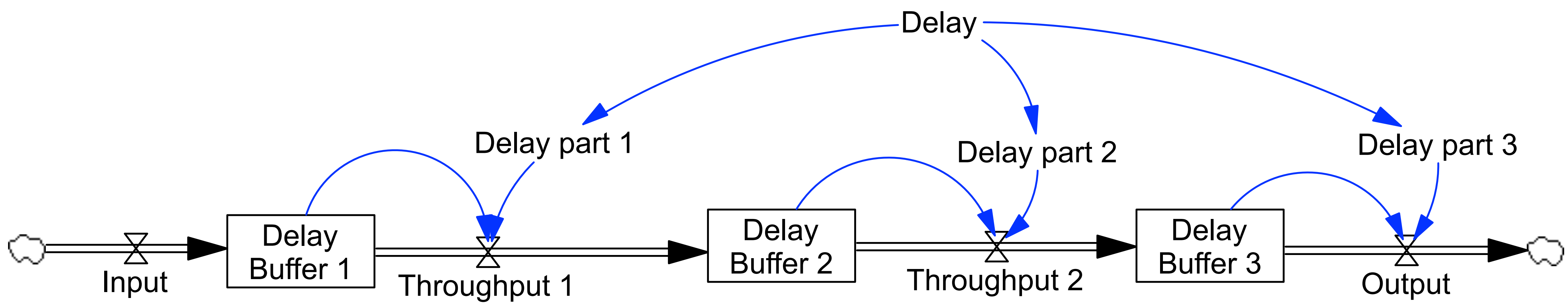
Non-Timeseries Optimization




```
def error(parameter_list):
    ...
    errors = sim_result - data_transform
    phase_plot_error = (errors**2).sum()
    ...

scipy.optimize.minimize(error, x0=[.005, 1, 1, .002],
                        method='L-BFGS-B',
                        bounds=[(0,10), (0,None),
                                (0,10), (0,None)])
```





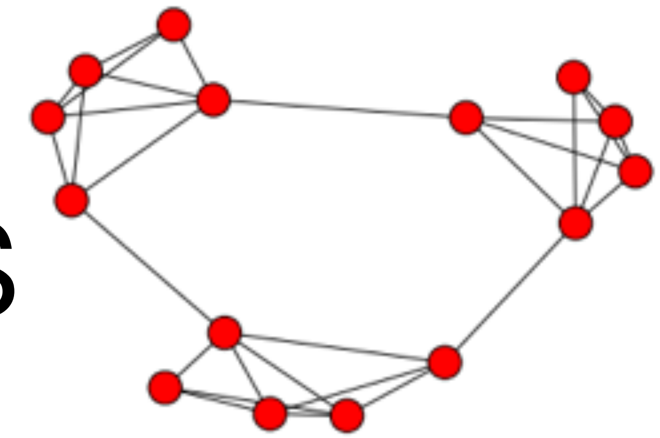


Realtime Interaction

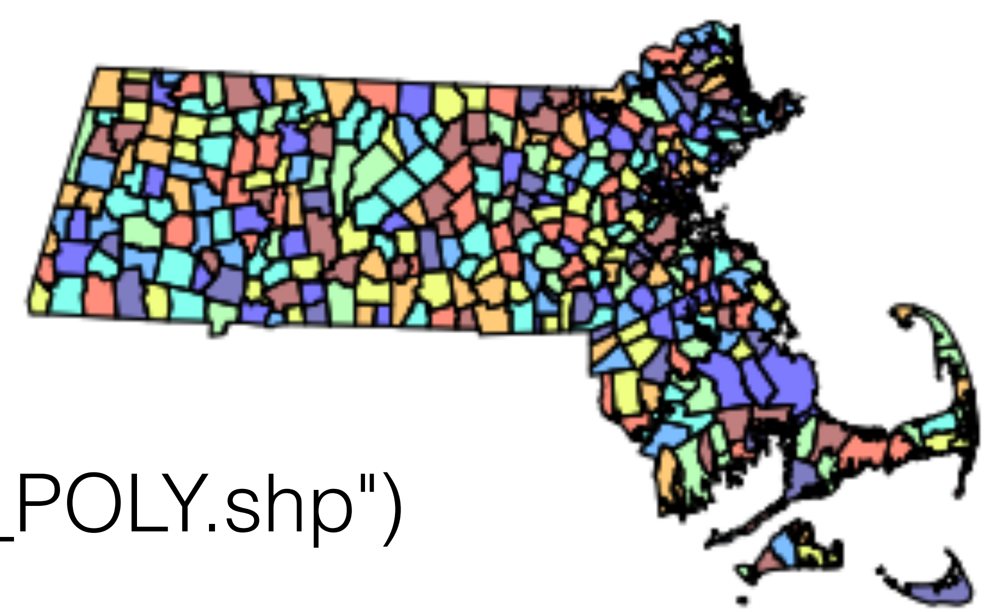
Maybe a demo?

```
def animate(t):
    #draw a live result
    …
    #run the simulation forward
    global stocks
    model.tstart = x[-1]
    model.tstop = x[-1]+1./fps
    model.initial_values = stocks
    ps={'input':input_val,
           'delay':adjustment_delay})
    stocks = model.run(params=ps).iloc[-1]
    …
    #collect user input
    …

anim = animation.FuncAnimation(fig, animate,
                                              frames=seconds*fps)
```

Network/Patch Models

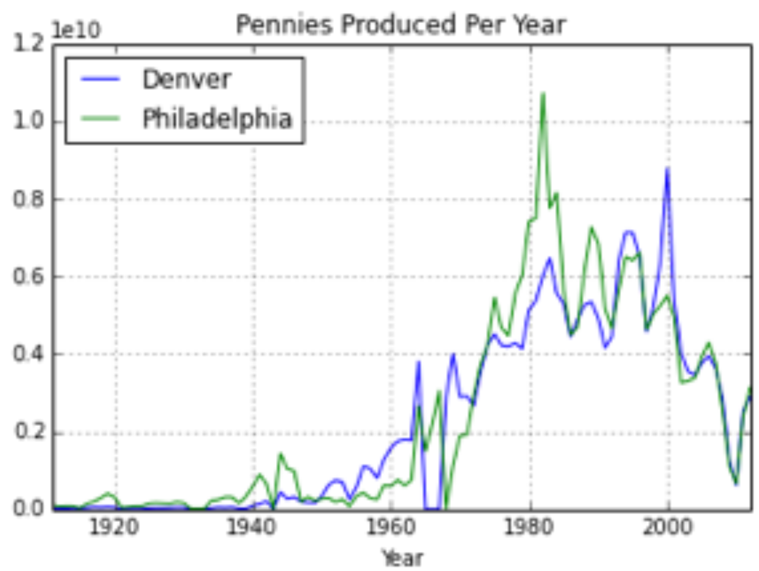


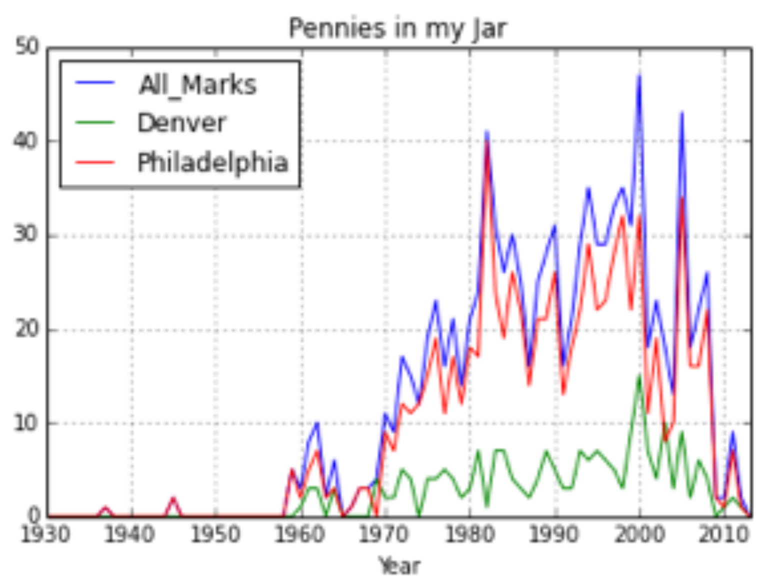
```
import networkx as nx
g = nx.graph(…)

for node in g.nodes_iter():
    g[node]['model'] = pysd.read_vensim('Patch SIR.mdl’)

for t in np.arange(model.tstart, model.tstop, model.dt):
    for node in g.nodes_iter():
        ps = {‘cross_infectivity’: nx.edgelist(node)… }
        g[node]['model'].run(params=ps)
```

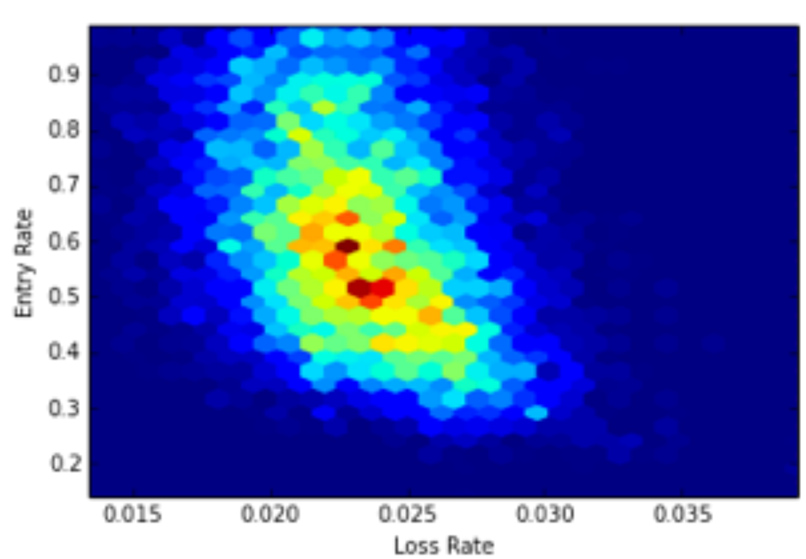


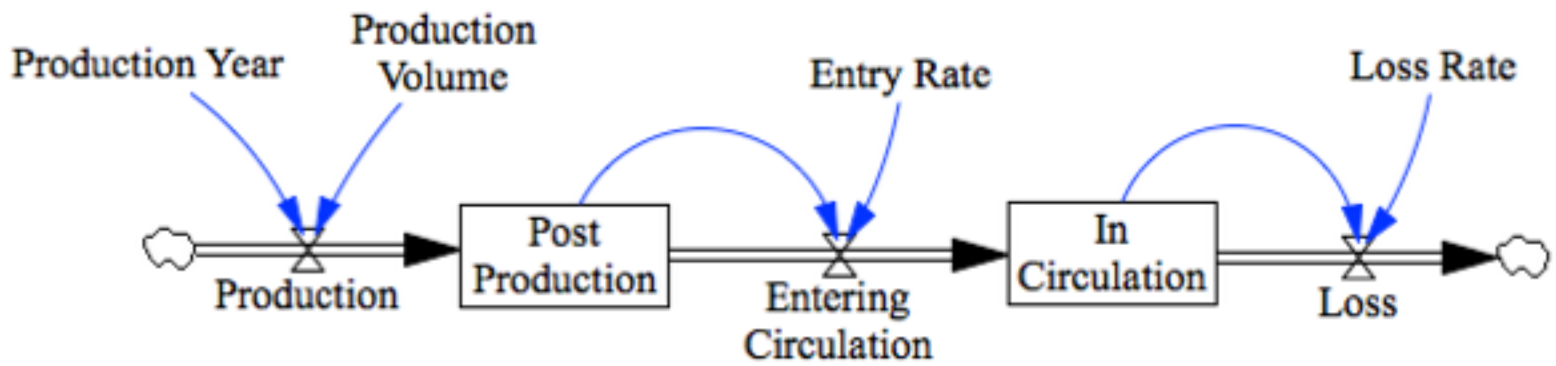
```
import geopandas as gpd
towns = gpd.read_file("TOWNSSURVEY_POLY.shp")
```

Subscripted MCMC










```
models = [[year, pysd.read_vensim('penny_jar.mdl')]
            for year in range(1930,2014)]

entry_rate = mc.Uniform('entry_rate', lower=0,
                                     upper=.99, value=.08)
loss_rate = mc.Uniform('loss_rate', lower=0,
                                   upper=.3, value=.025)

@mc.stochastic(trace=True, observed=True)
def circulation(…)
    population = models['model'].run(…)…
    …
    return log_prob

mcmc = mc.MCMC(…)
mcmc.sample(20000)
```

To be demonstrated:

- Fitting models to streaming data
- Machine learning regressions in place of lookups
- Reversible jump MCMC for model selection
- Web-based interactive result display
- Runtime connection to ABM, etc…
- Adaptive sampling of parameter spaces
- Integration with decision models
- …

To be Developed:

- cython/theano speedups
- Step function with memory
- Smart parameter modification
- Additional XMILE/Vensim translation ability