Discrete-materials/discrete-facility nuclear fuel cycle systems analysis with GENIUS

Kyle Oliver

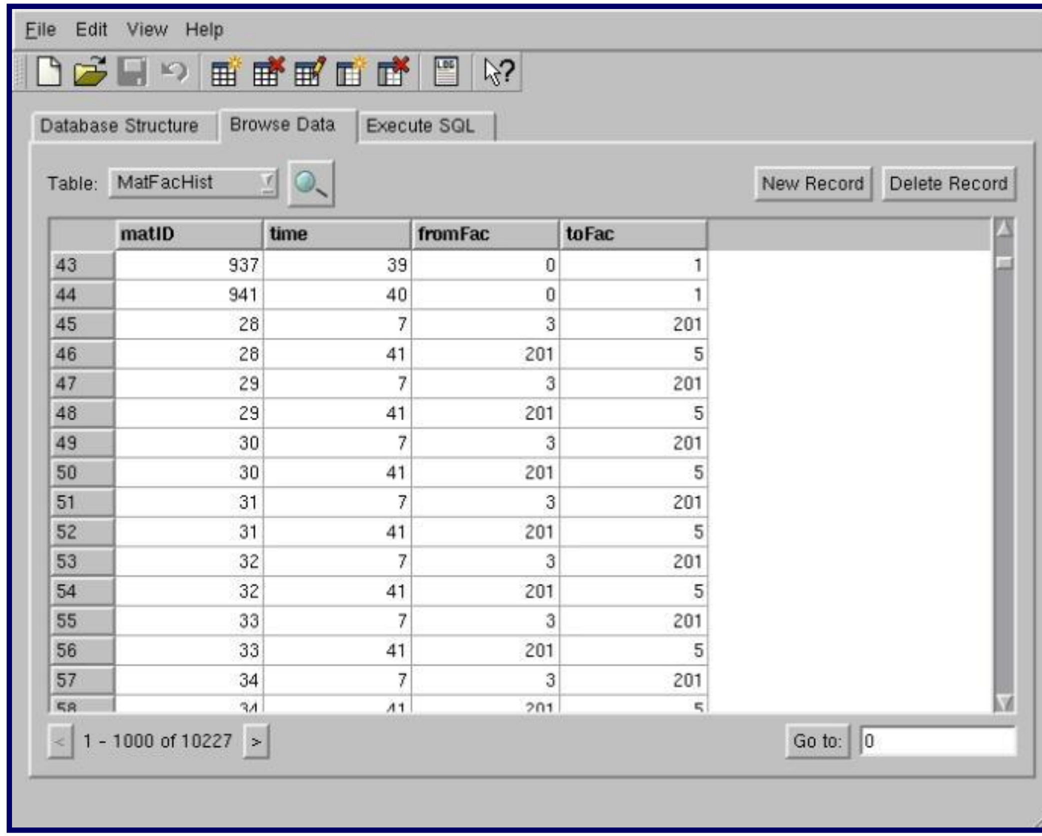
Computational
Nuclear Engineering
Research Group
(CNERG)

Paul Wilson

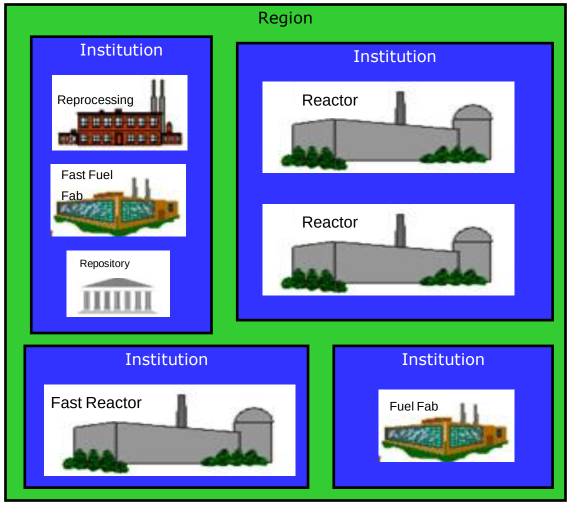
Advisor

UW-Madison
Engineering Physics

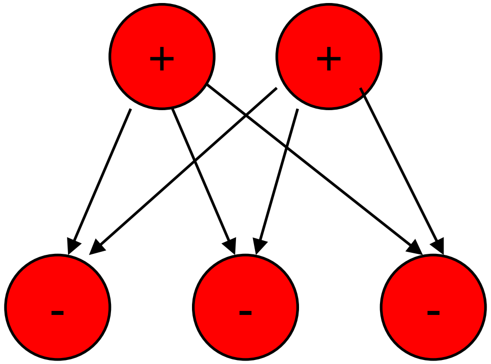
Nov. 17, 2008



A highly discretized fuel cycle code poses new opportunities, challenges.

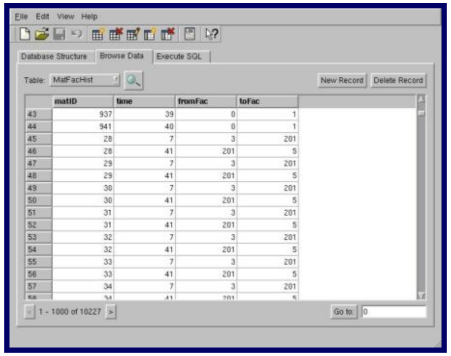


Scenarios: **GENIUS data model allows rich, realistic scenario specification.**



Capabilities: **GENIUS currently supports once-through fuel cycles; closed cycles to follow shortly.**

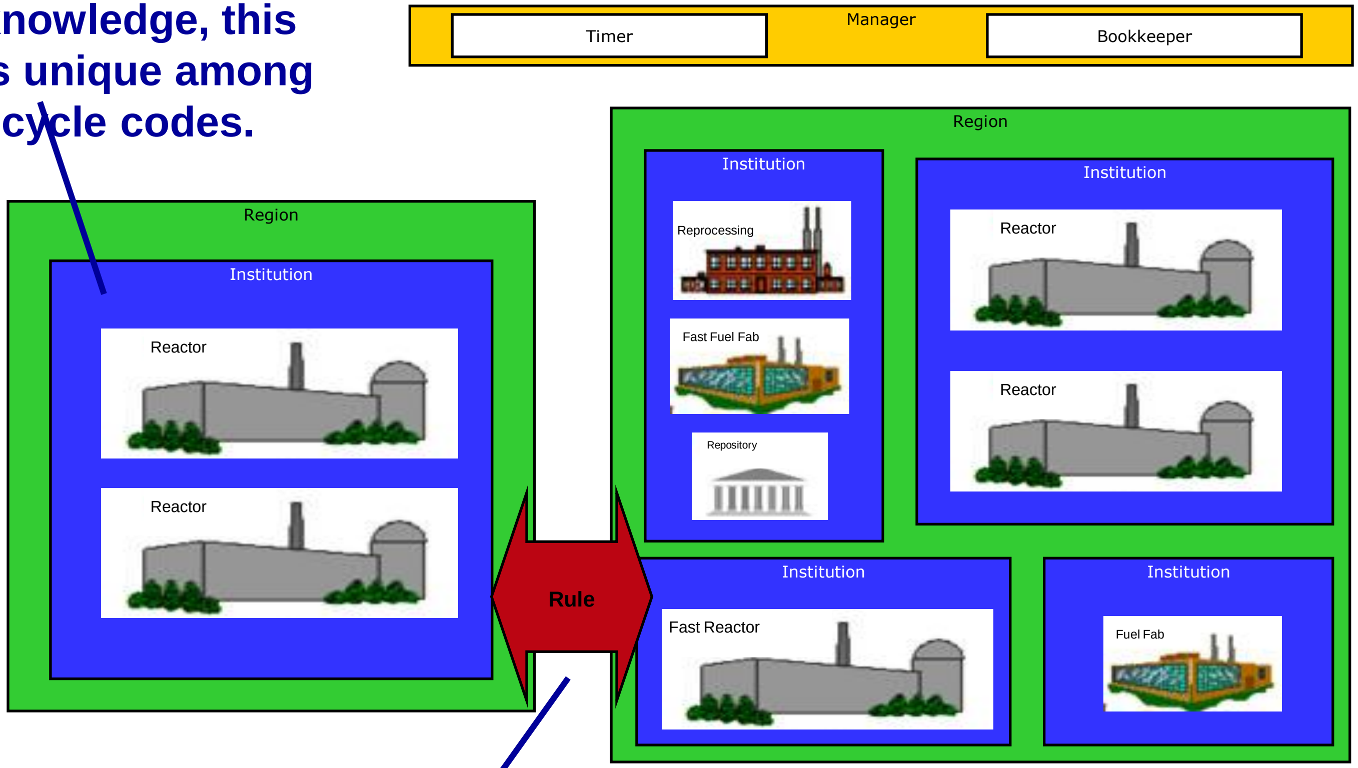
Infrastructure: **GENIUS makes extensive use of existing computing tools and libraries, especially for optimization.**



	matID	time	fromFac	toFac
43	937	39	0	1
44	941	40	0	1
45	28	7	3	201
46	28	41	201	5
47	29	7	3	201
48	29	41	201	5
49	30	7	3	201
50	30	41	201	5
51	31	7	3	201
52	31	41	201	5
53	32	7	3	201
54	32	41	201	5
55	33	7	3	201
56	33	41	201	5
57	34	7	3	201

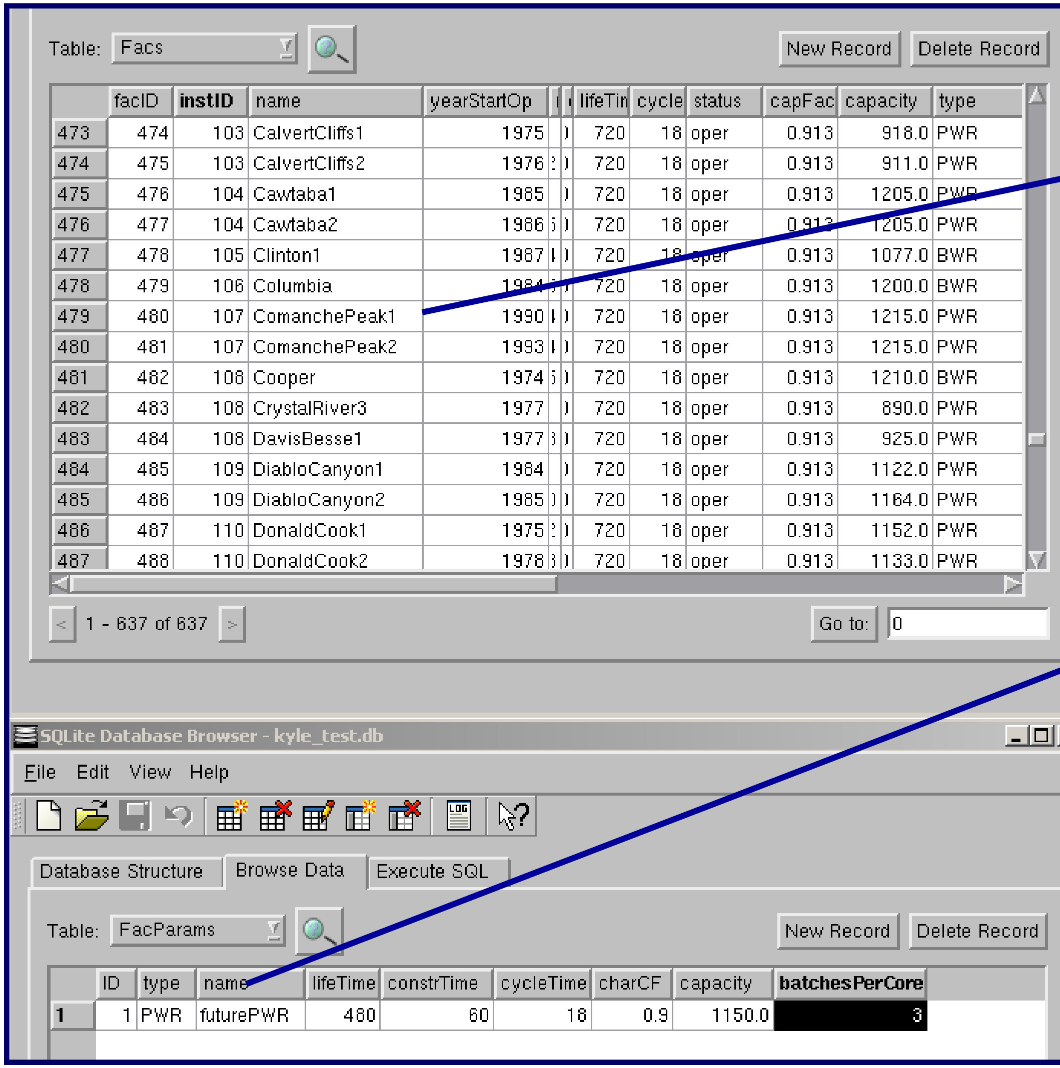
Users specify nuclear facilities, the institutions that own them, and the regions they operate in.

Institutions represent companies and government entities. To our knowledge, this modeling layer is unique among comparable fuel cycle codes.



User-specified rules might state whether one fuel cycle state can send materials to another, or define some special fuel-trade contract.

Complete scenario input file includes facility deployment "initial condition" and future build plan.



	facID	instID	name	yearStartOp	lifeTim	cycle	status	capFac	capacity	type
473	474	103	CalvertCliffs1	1975	720	18	oper	0.913	918.0	PWR
474	475	103	CalvertCliffs2	1976	720	18	oper	0.913	911.0	PWR
475	476	104	Cawtaba1	1985	720	18	oper	0.913	1205.0	PWR
476	477	104	Cawtaba2	1986	720	18	oper	0.913	1205.0	PWR
477	478	105	Clinton1	1987	720	18	oper	0.913	1077.0	BWR
478	479	106	Columbia	1984	720	18	oper	0.913	1200.0	BWR
479	480	107	ComanchePeak1	1990	720	18	oper	0.913	1215.0	PWR
480	481	107	ComanchePeak2	1993	720	18	oper	0.913	1215.0	PWR
481	482	108	Cooper	1974	720	18	oper	0.913	1210.0	BWR
482	483	108	CrystalRiver3	1977	720	18	oper	0.913	890.0	PWR
483	484	108	DavisBesse1	1977	720	18	oper	0.913	925.0	PWR
484	485	109	DiabloCanyon1	1984	720	18	oper	0.913	1122.0	PWR
485	486	109	DiabloCanyon2	1985	720	18	oper	0.913	1164.0	PWR
486	487	110	DonaldCook1	1975	720	18	oper	0.913	1152.0	PWR
487	488	110	DonaldCook2	1978	720	18	oper	0.913	1133.0	PWR

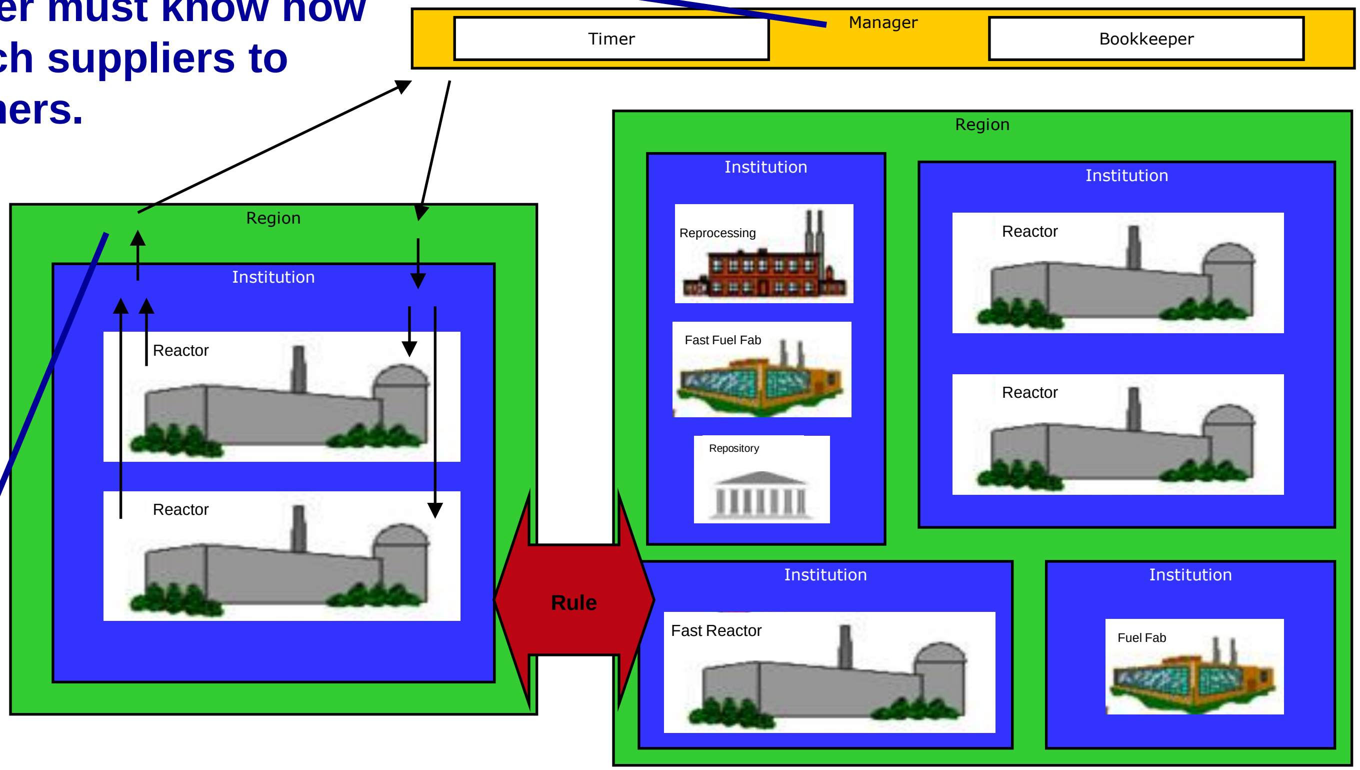
	ID	type	name	lifeTime	constrTime	cycleTime	charCF	capacity	batchesPerCore
1	1	PWR	futurePWR	480	60	18	0.9	1150.0	3

Existing and planned facilities can be listed individually according to some nuclear facilities database

Generic future facilities get built according to a user-specified timetable

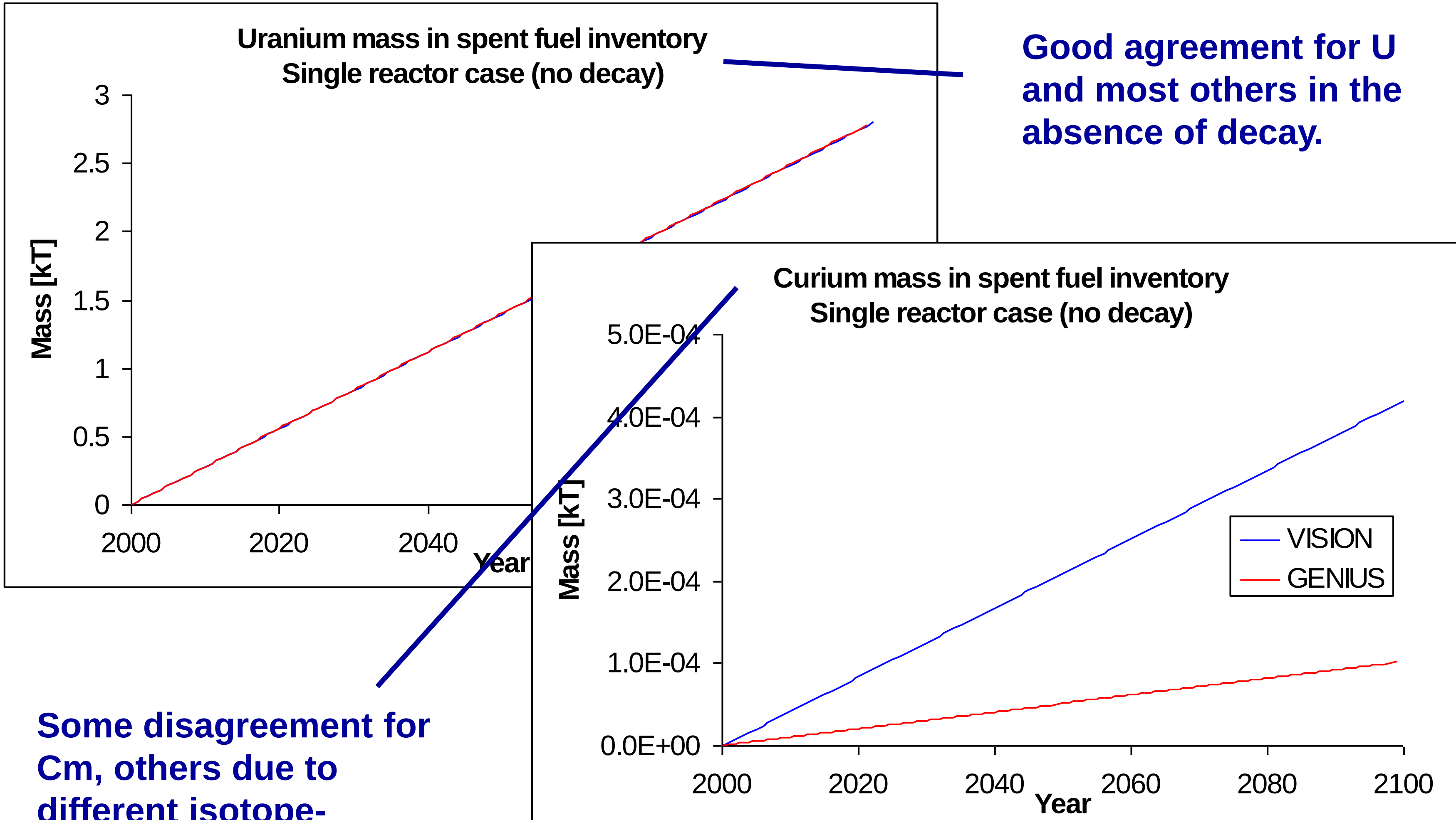
Currently, simulation manager uses simple greedy algorithm to match "once-through" commodities.

For each commodity, the manager must know how to match suppliers to customers.



Requests and offers travel up to the manager at the beginning of each time step. After matching, instructions get sent back down to facilities.

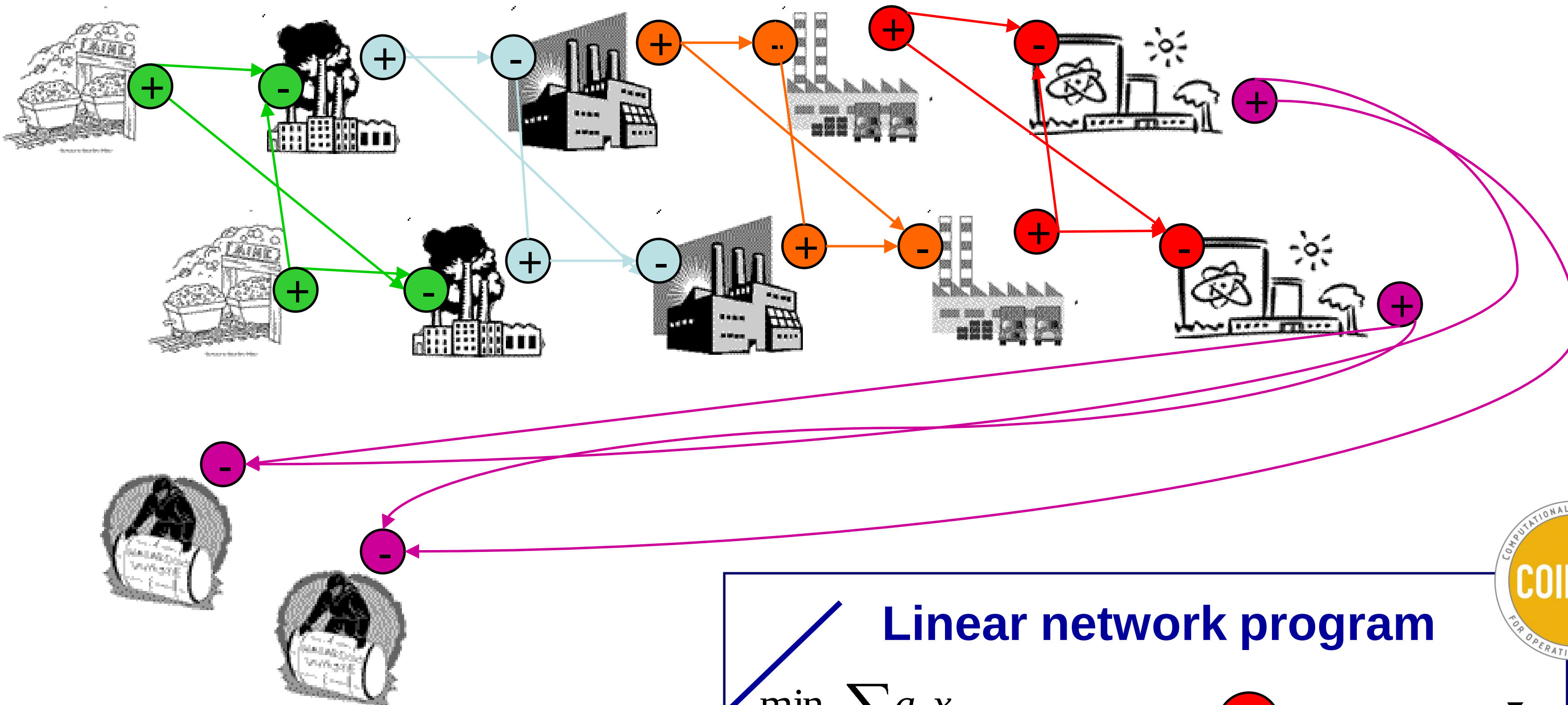
GENIUS mass flows and power production can be benchmarked against other codes (e.g., VISION)



Good agreement for U and most others in the absence of decay.

Some disagreement for Cm, others due to different isotope-tracking conventions.

We can improve on existing greedy algorithm for commodity matching using network optimization…



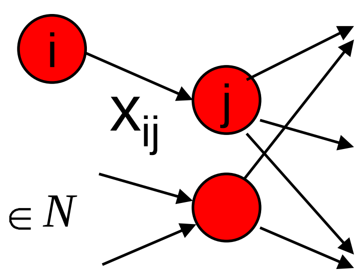
Match suppliers and customers for each commodity by solving a standard network flow problem.

Linear network program

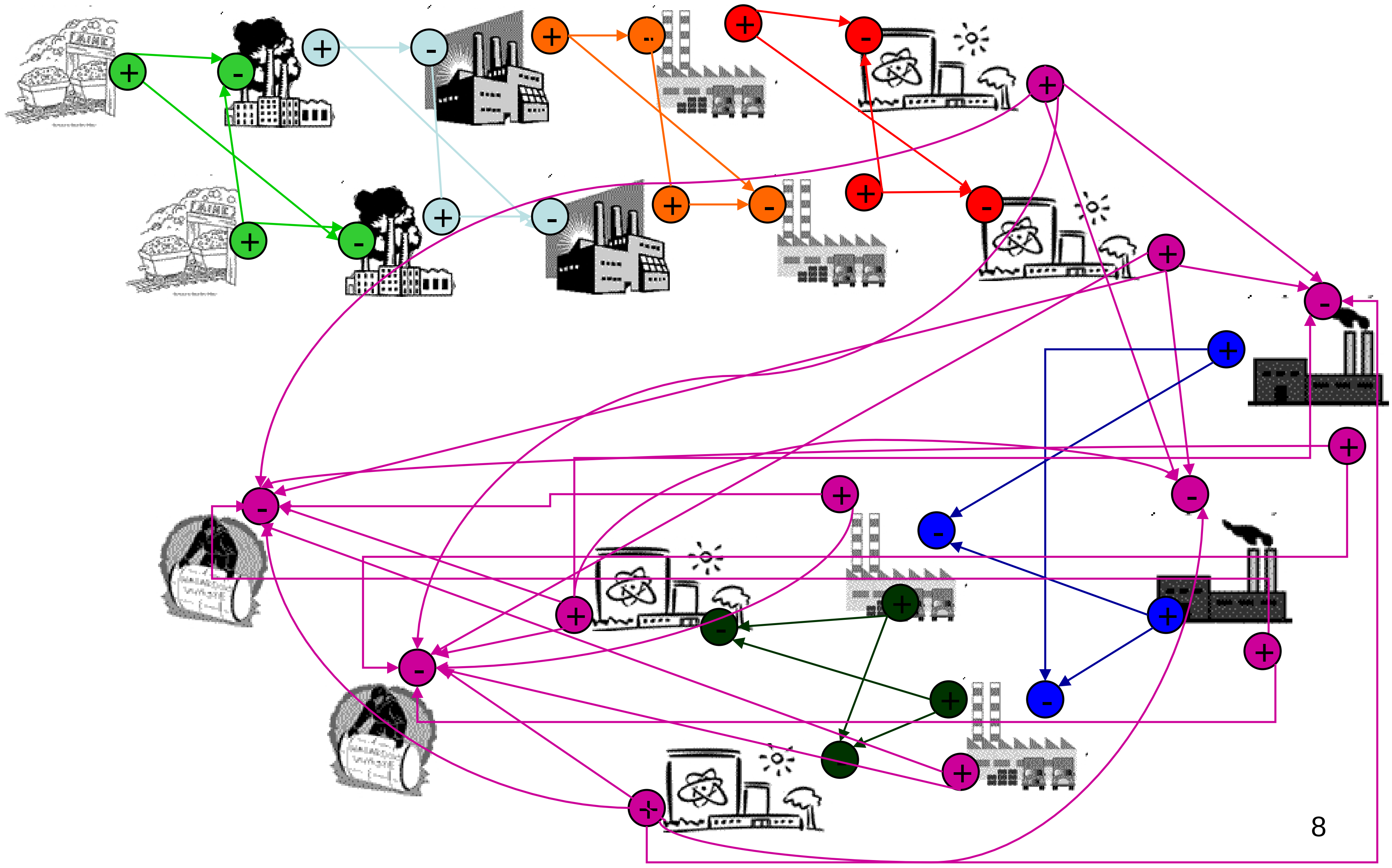
$$\min \sum_{(i,j)\in A} a_{ij} x_{ij}$$

$$s.t. x_{ij} \in [b_{ij}, c_{ij}], \forall (i,j) \in A$$

$$\sum_{\{j|(i,j)\in A\}} x_{ij} - \sum_{\{j|(j,i)\in A\}} x_{ji} = s_i, \forall i \in N$$



…but the networks get complicated under scenarios that include reprocessing.



GENIUS will also be able to be called iteratively by optimizers to identify promising fuel cycle designs.

DAKOTA

Iteration tool perturbs input scenario and re-executes the code, iterating to convergence.

Execution of main GENIUS code measures effectiveness of proposed fuel cycle subject to some global objective function (e.g., levelized cost of required electricity).

Open-source scientific computing tools improve code performance, development time, installation.

SQLite databases log facility and material histories and serve as input and output files.

Python and matplotlib provide functionality for custom pre- and post-processing modules.

Doxygen automatically generates C++ code documentation.

COIN-OR's Clp linear program solver optimizes material routing.

GNU's Autotools system streamlines platform-specific installation.

DAKOTA's iterators will be used (?) to optimize facility deployment.