Representing California's Water System with an Open Source Model: PyVIN

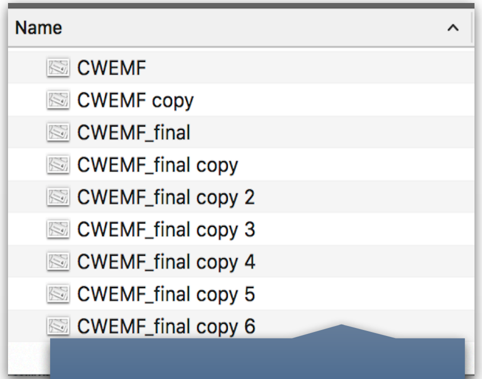
Mustafa Dogan

Center for Watershed Sciences, UC Davis

CWEMF 2017

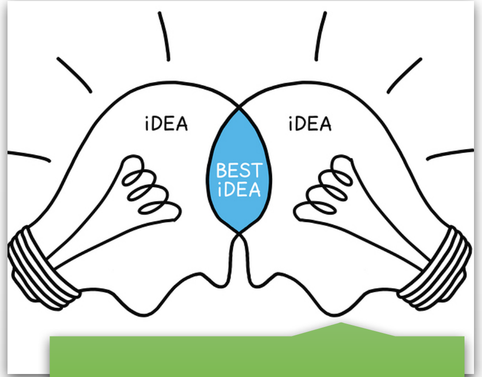
Goals

data transparency

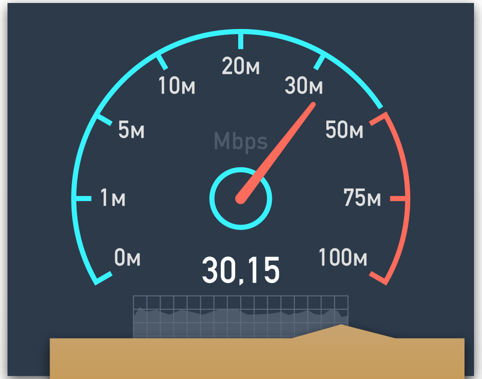


version control

documentation

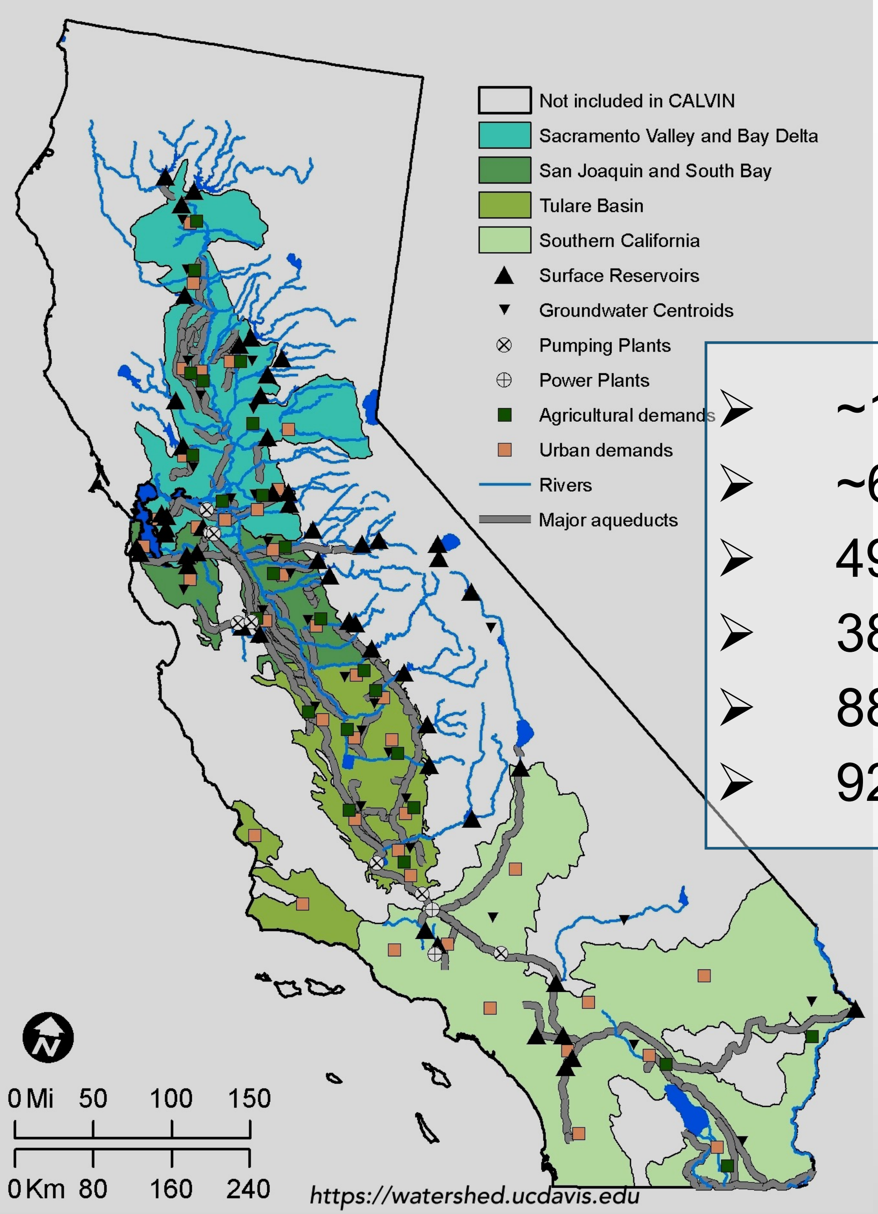


collaboration



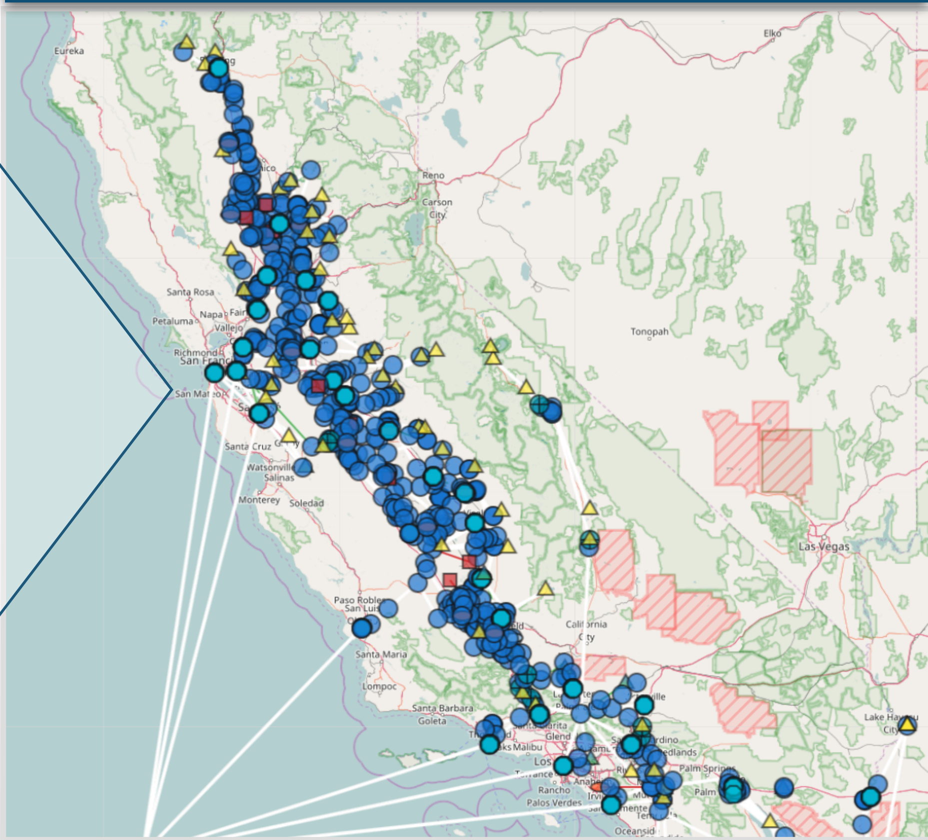
run speed

CALVIN



- ~1250 nodes
- ~600 conveyance links
- 49 surface reservoirs
- 38 groundwater reservoirs
- 88% of CA's irrigated acreage
- 92% of CA's urban population

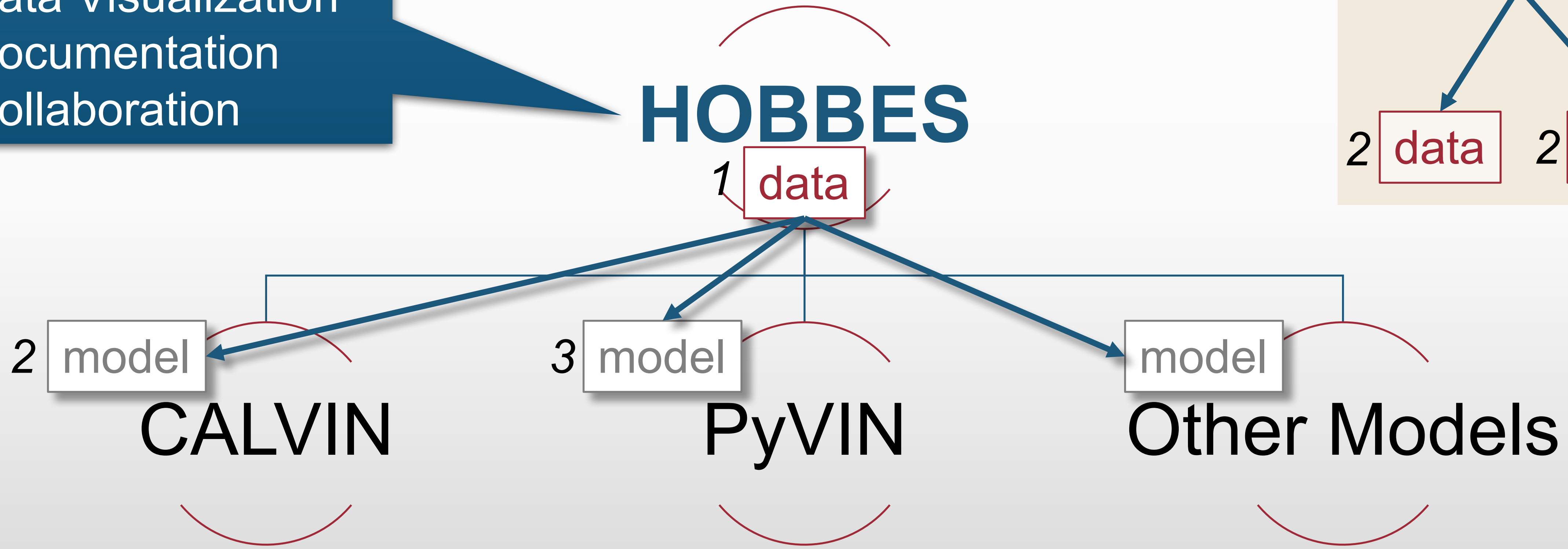
HOBBES Database



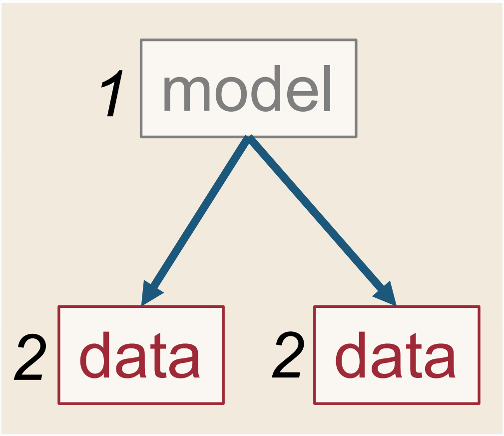
https://cwn.casil.ucdavis.edu

HOBBES Database

- Online Database
- Version Control
- Data Visualization
- Documentation
- Collaboration

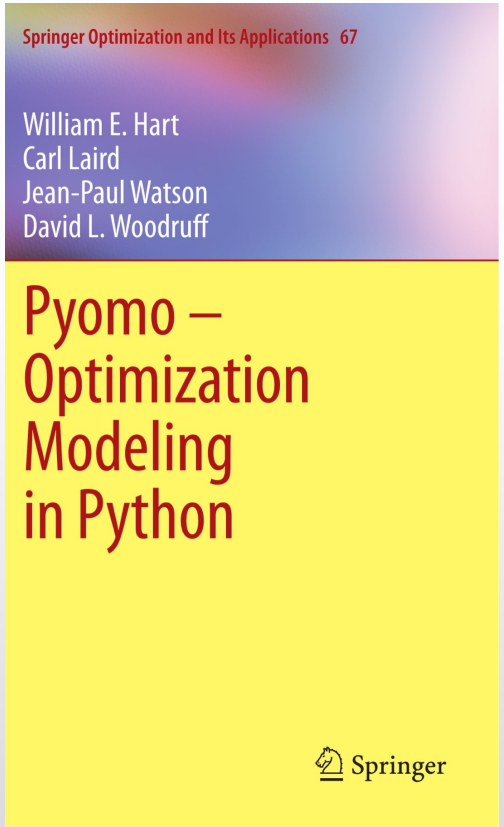


Classical Approach



Modeling Environment

PYOMO

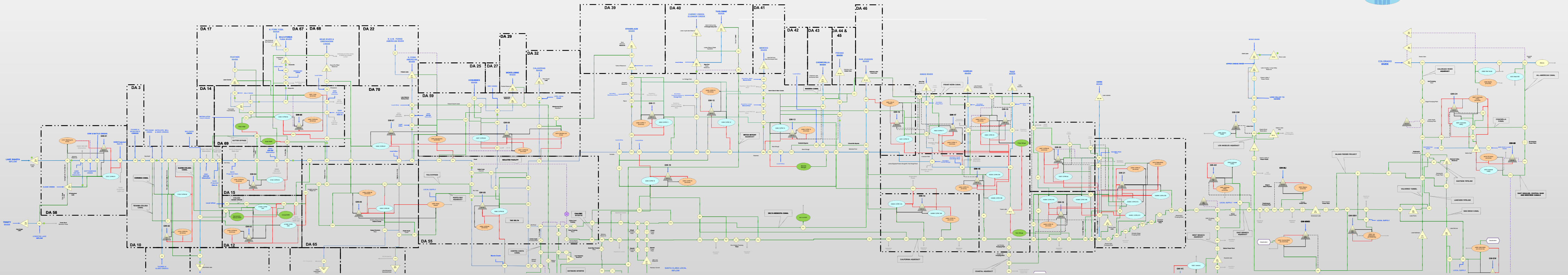


http://www.pyomo.org/documentation

- Pyomo is a Python-based, open-source optimization modeling language
- Algebraic language similar to GAMS, AMPL
- Easy to install: `”conda install -c cachemeorg pyomo”`
- User defined solvers:
  - `CPLEX`[1]
  - `GUROBI`[1]
  - `CBC`[2]
  - `GLPK`[2]

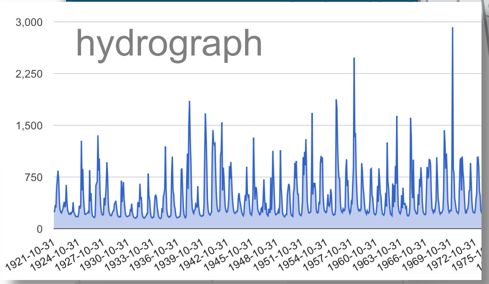
([1] free for academic purposes only, [2] open source)

CALVIN v.1	PyVIN v.2
Large-scale hydroeconomic model	
Optimize water allocation to agricultural and urban users	
Minimize statewide water scarcity and operating costs	
HEC-PRM and VBA based	Pyomo and Python based
Less flexible (limited to HEC-PRM)	More flexible (full LP)
Solver runtime: ~16 hr (depending on initial solution)	Solver runtime: ~1 min (depending on solver)
Requires 32 bit Windows PC	Any computer
HEC-DSS database	Open source: data and source code



HOBBES Matrix Creator

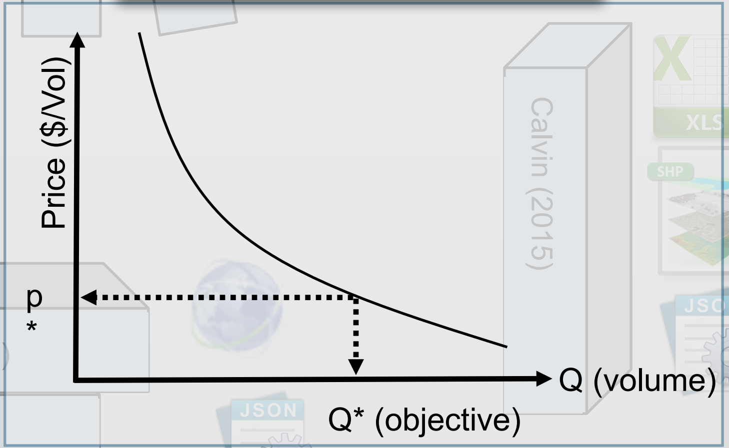
Time-series



Date	D94-SR-WHI
10/31/1921	203.00
11/30/1921	116.74
12/31/1921	53.68
01/31/1922	0.00
02/28/1922	0.00
03/31/1922	0.00
04/30/1922	0.00
05/31/1922	0.00
06/30/1922	0.00
07/31/1922	15.92
08/31/1922	0.00
09/30/1922	0.00
10/31/1922	0.00

+

Demand Curves



Data Matrix (PyVIN input)

i	j	k	cost	amplitude	lower_bound	upper_bound
HU402A.1965-05-31	CVPM14AG.1965-05-31	0	-1760	1	0	32.83
HU402A.1965-05-31	CVPM14AG.1965-05-31	1	-1619	1	0	2.98
HU402A.1965-05-31	CVPM14AG.1965-05-31	2	-1450	1	0	2.98
HU402A.1965-05-31	CVPM14AG.1965-05-31	3	-1199	1	0	2.99
HU402A.1965-05-31	CVPM14AG.1965-05-31	4	-927.5	1	0	2.98
HU402A.1965-05-31	CVPM14AG.1965-05-31	5	-782.2	1	0	2.99
HU402A.1965-05-31	CVPM14AG.1965-05-31	6	-720.7	1	0	2.98
HU402A.1965-05-31	CVPM14AG.1965-05-31	7	-614.8	1	0	2.99
HU402A.1965-05-31	CVPM14AG.1965-05-31	8	-500.4	1	0	2.98
HU402A.1965-05-31	CVPM14AG.1965-05-31	9	-224.2	1	0	2.98
HU402A.1965-06-30	CVPM14AG.1965-06-30	0	-1764	1	0	53.70
HU402A.1965-06-30	CVPM14AG.1965-06-30	1	-1618	1	0	4.88
HU402A.1965-06-30	CVPM14AG.1965-06-30	2	-1446	1	0	4.89
HU402A.1965-06-30	CVPM14AG.1965-06-30	3	-1202	1	0	4.88
HU402A.1965-06-30	CVPM14AG.1965-06-30	4	-926.6	1	0	4.88
HU402A.1965-06-30	CVPM14AG.1965-06-30	5	-784	1	0	4.88
HU402A.1965-06-30	CVPM14AG.1965-06-30	6	-719.9	1	0	4.88
HU402A.1965-06-30	CVPM14AG.1965-06-30	7	-615	1	0	4.89
HU402A.1965-06-30	CVPM14AG.1965-06-30	8	-499.9	1	0	4.88
HU402A.1965-06-30	CVPM14AG.1965-06-30	9	-223.9	1	0	4.88
HU402A.1965-07-31	CVPM14AG.1965-07-31	0	-1763	1	0	54.33

PyVIN Model Structure

1 Linear Programming Model with Gains & Losses

objective
$$\text{minimize } Z = \sum_i \sum_j \sum_k c_{ijk} \cdot X_{ijk}$$

mass balance
$$\sum_i \sum_k X_{ijk} = \sum_i \sum_k a_{ijk} \cdot X_{ijk} + b_j$$

upper bound
$$X_{ijk} \leq u_{ijk}$$

lower bound
$$X_{ijk} \geq l_{ijk}$$

where
Z: total cost
X: flow on the arc
c: unit cost (or penalty)
b: external flow
a: amplitude
l: lower bound
u: upper bound

2 Abstract Model in Pyomo

Model Structure + Data = PyVIN

Separate data and model structure

3 Freely Available Solvers

- CPLEX
- CBC
- GUROBI
- GLPK

User installed and defined solvers
No dependency on any solver

PyVIN Run Process

HOBBES
- Input Data
  - Network matrix

Pyomo
- Model Run
  - Specify data file and solver

Python
- Postprocess Results
  - Analyze results stored in `*.csv` files

HOBBES
- Data Visualization
  - Time-series plots & animation

```
model = AbstractModel()

# Nodes in the network
model.N = Set()

# Network arcs
model.k = Set()
model.A = Set(within=model.N*model.N*model.k)

# Source node
model.source = Param(within=model.N)
# Sink node
model.sink = Param(within=model.N)
# Flow capacity limits
model.u = Param(model.A)
# Flow lower bound
model.l = Param(model.A)
# Link amplitude (gain/loss)
model.a = Param(model.A)
# Link cost
model.c = Param(model.A)

# The flow over each arc
model.X = Var(model.A, within=Reals)

# Minimize total cost
def total_rule(model):
    return sum(model.c[i,j,k]*model.X[i,j,k] for (i,j,k) in model.A)
model.total = Objective(rule=total_rule, sense=minimize)

# Enforce an upper bound limit on the flow across each arc
def limit_rule_upper(model, i, j, k):
    return model.X[i,j,k] <= model.u[i,j,k]
model.limit_upper = Constraint(model.A, rule=limit_rule_upper)

# Enforce a lower bound limit on the flow across each arc
def limit_rule_lower(model, i, j, k):
    return model.X[i,j,k] >= model.l[i,j,k]
model.limit_lower = Constraint(model.A, rule=limit_rule_lower)
```

PyVIN

Input

```python
model = AbstractModel()

# Nodes in the network
model.N = Set()

# Network arcs
model.k = Set()
model.A = Set(within=model.N*model.N*model.k)

# Source node
model.source = Param(within=model.N)
# Sink node
model.sink = Param(within=model.N)
# Flow capacity limits
model.u = Param(model.A)
# Flow lower bound
model.l = Param(model.A)
# Link amplitude (gain/loss)
model.a = Param(model.A)
# Link cost
model.c = Param(model.A)

# The flow over each arc
model.X = Var(model.A, within=Reals)

# Minimize total cost
def total_rule(model):
    return sum(model.c[i,j,k]*model.X[i,j,k] for (i,j,k) in model.A)
model.total = Objective(rule=total_rule, sense=minimize)

# Enforce an upp
def limit_rule_u
    return model
model.limit_uppe

# Enforce a lowe
def limit_rule_l
```

pyvin.py

i	j	k	cost	amplitude	lower_bound	upper_bound
A101.2002-10-31	HU101.2002-	0	0	1	0	1E+12
A101.2002-11-30	HU101.2002-	0	0	1	0	1E+12
A101.2002-12-31	HU101.2002-	0	0	1	0	1E+12
A101.2003-01-31	HU101.2003-	0	0	1	0	1E+12
A101.2003-02-28	HU101.2003-	0	0	1	0	1E+12
A101.2003-03-31	HU101.2003-	0	0	1	0	1E+12
A101.2003-04-30	HU101.2003-	0	0	1	0	1E+12
A101.2003-05-31	HU101.2003-	0	0	1	0	1E+12
A101.2003-06-30	HU101.2003-	0	0	1	0	1E+12
A101.2003-07-31	HU101.2003-	0	0	1	0	1E+12
A101.2003-08-31	HU101.2003-	0	0	1	0	1E+12
A101.2003-09-30	HU101.2003-	0	0	1	0	1E+12
A102.2002-10-31	HU102.2002-	0	0	1	0	1E+12
A102.2002-11-30	HU102.2002-	0	0	1	0	1E+12
A102.2002-12-31	HU102.2002-	0	0	1	0	1E+12
A102.2003-01-31	HU102.2003-	0	0	1	0	1E+12
A102.2003-02-28	HU102.2003-	0	0	1	0	1E+12
A102.2003-03-31	HU102.2003-	0	0	1	0	1E+12
A102.2003-04-30	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
A102.2003-05-31						

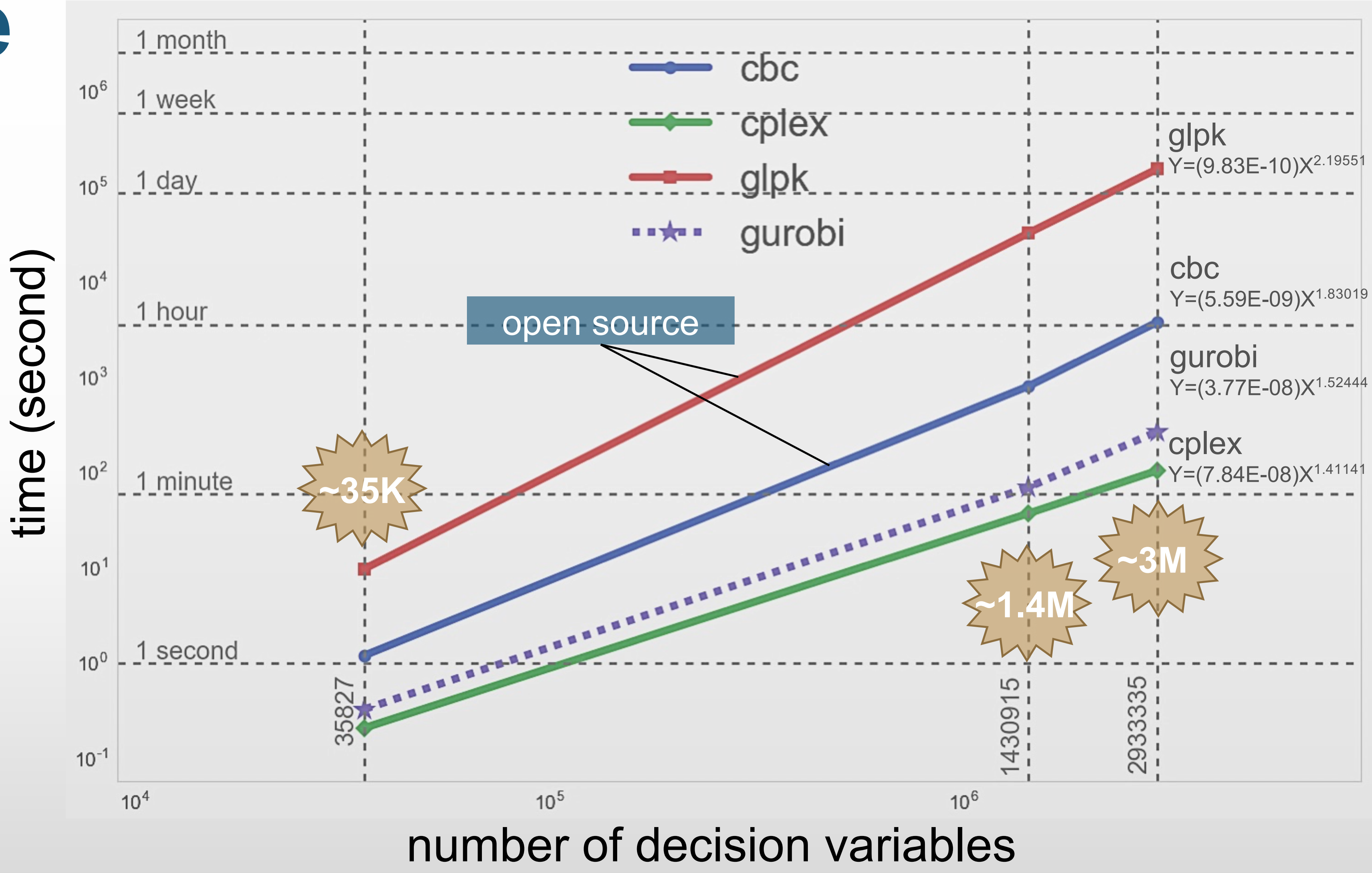
data.dat

Output

- water deliveries to users — `flow.csv`
- surface and groundwater storage — `storage.csv`
- cost of shortage to users and WTP for additional water — `shortage cost.csv`
- shadow prices of environmental flows — `dual - lower bound.csv`
- benefits of capacity expansion — `dual - upper bound.csv`

Solver Time

- ➢ Runs are performed on UC Davis College of Engineering's HPC1 (High Performance Computer)
- ➢ `cbc`, `cplex` and `gurobi` are run in parallel, and `glpk` is run in serial

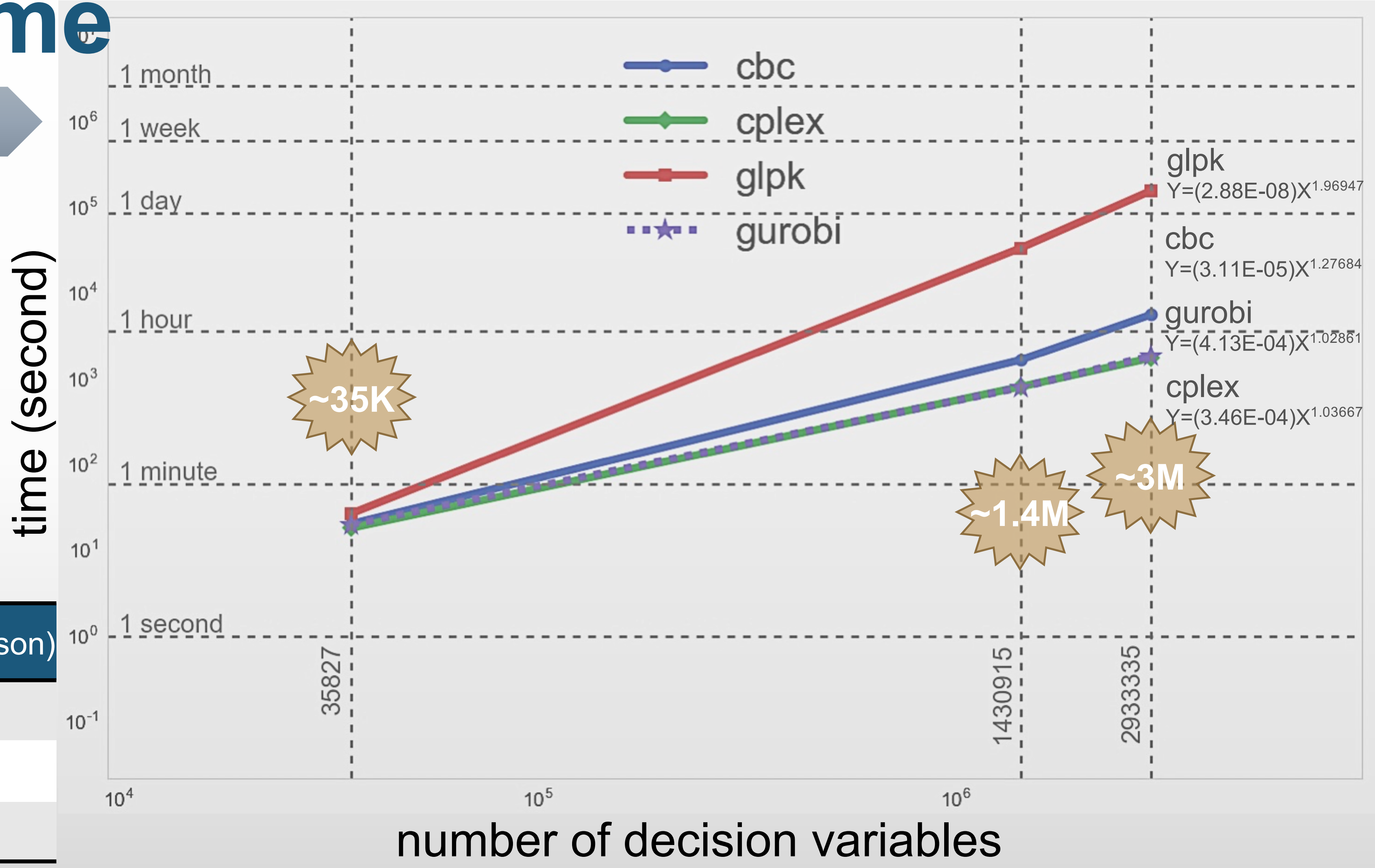


Total Runtime

Pyomo builds the model → Solver finds a solution → Pyomo writes results

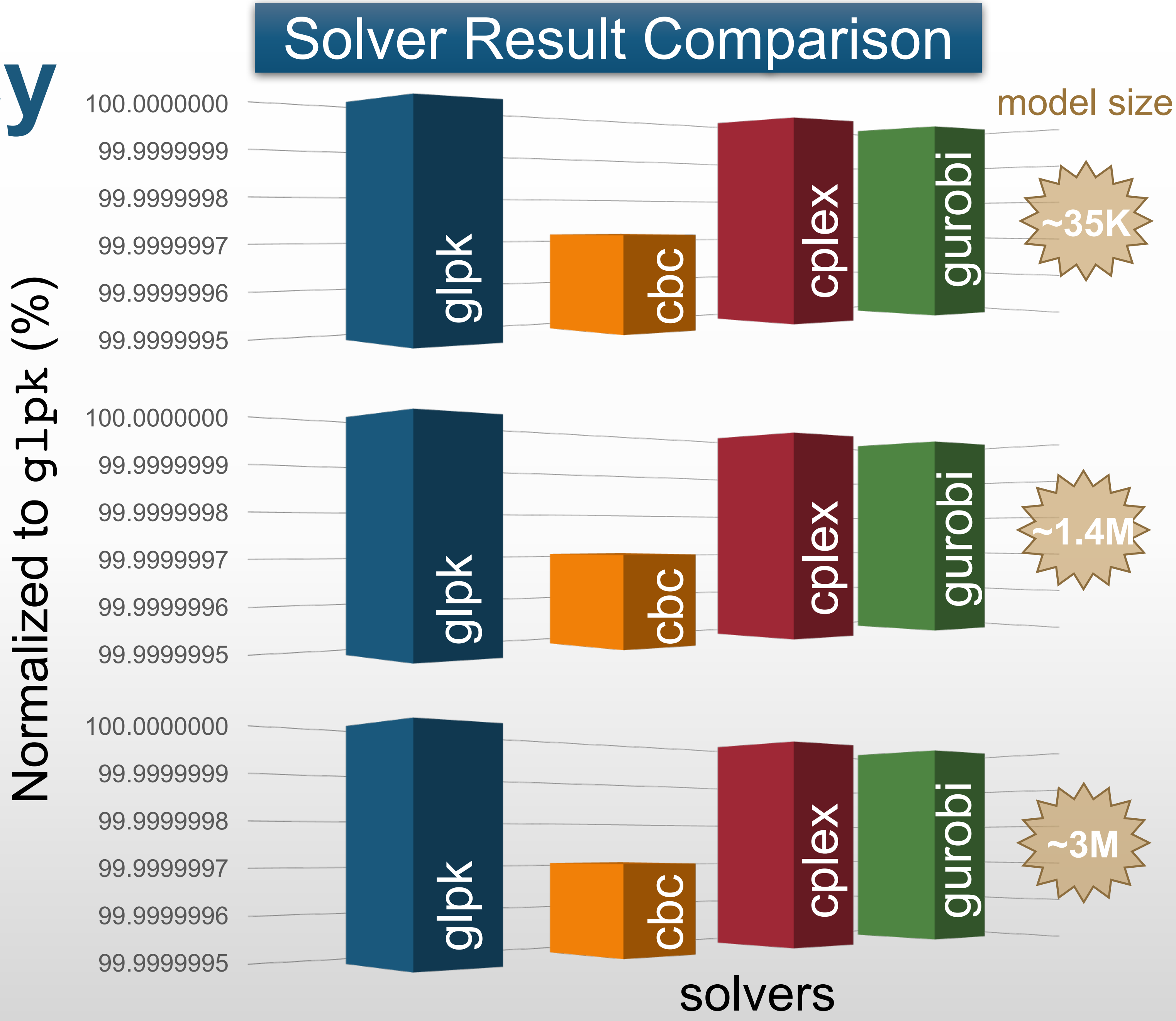
- Differences get bigger as model size increases
- `cplex` and `gurobi` have similar total runtime

# of decision variables	output file size (json)
35,827	~7.5 MB
1,430,915	~300 MB
2,933,335	~602 MB



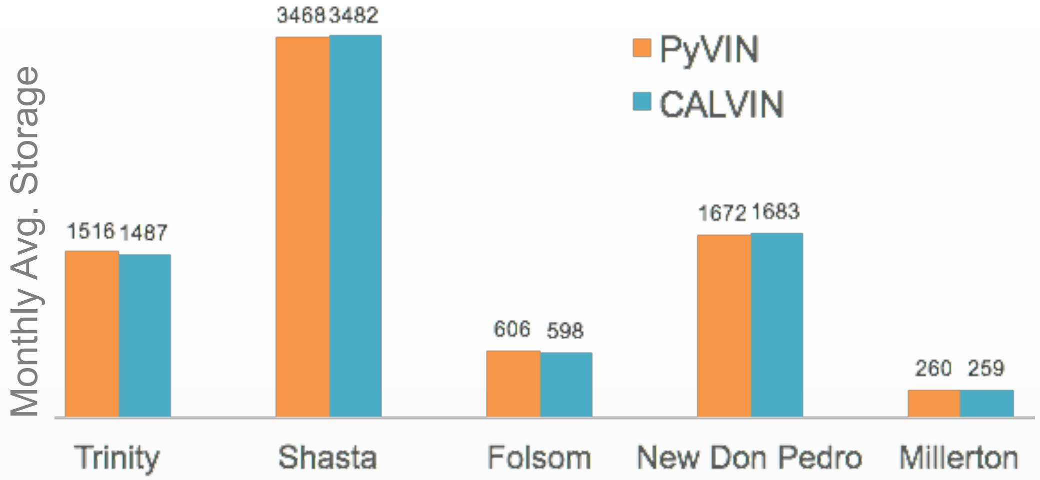
Solver Accuracy

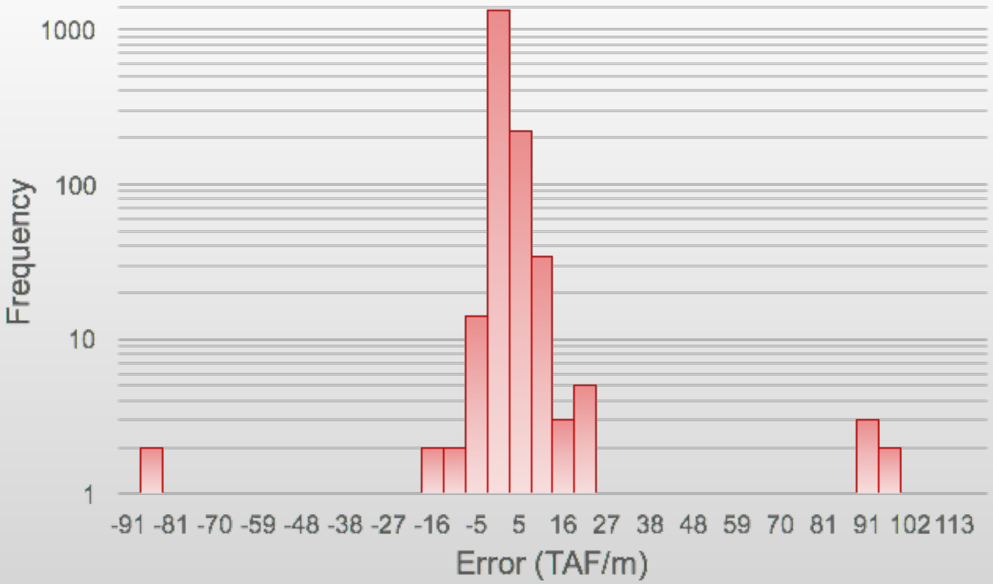
- Objective value normalized to `glpk` results
- `cbc` results slightly lower
- Overall consistent solver results



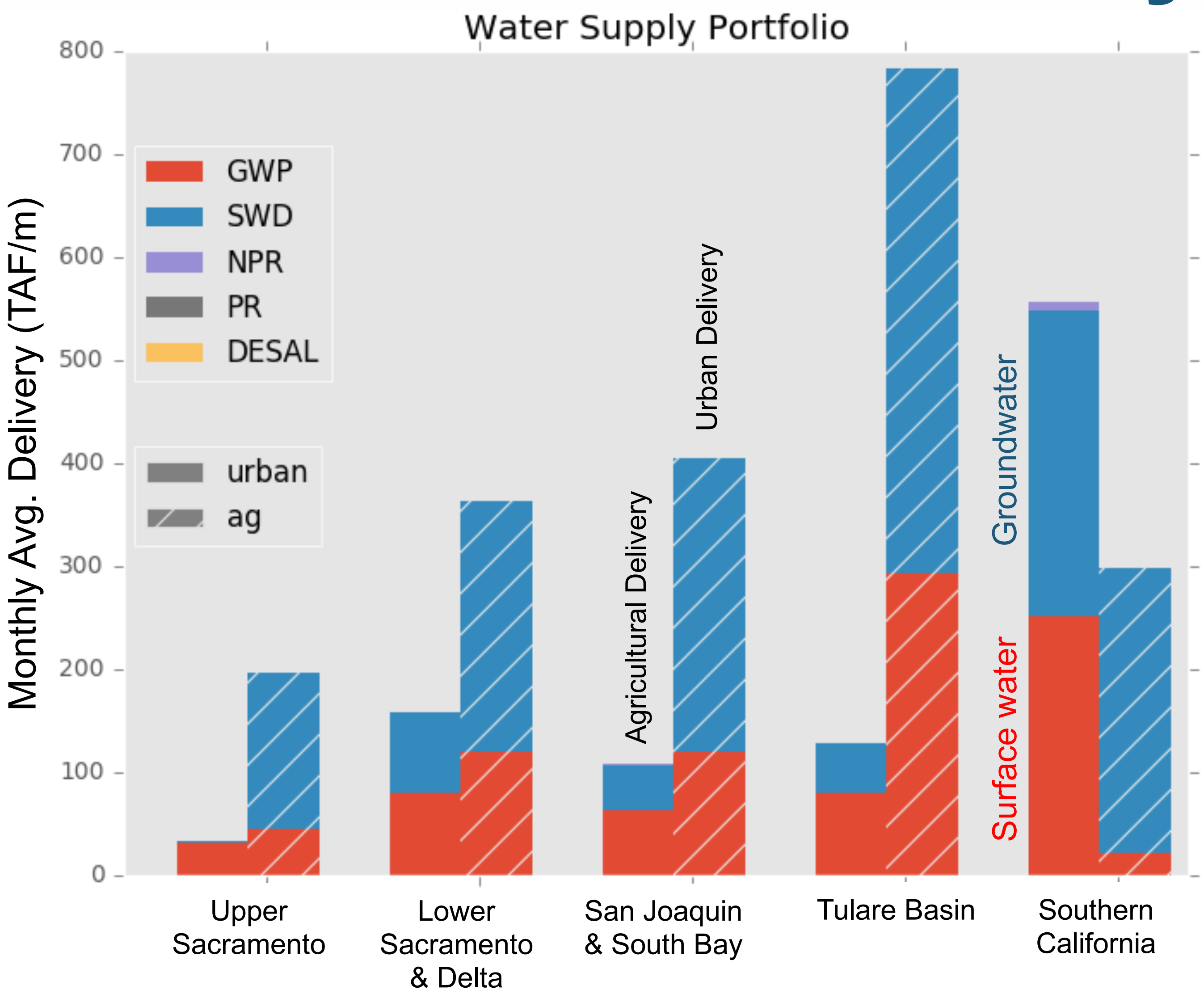
CALVIN to PyVIN

- Similar results but faster runtime
  (16 hours vs. 1 minute)
- Easy-to-understand model: only ~20 lines of code
  (solver takes care of the rest)

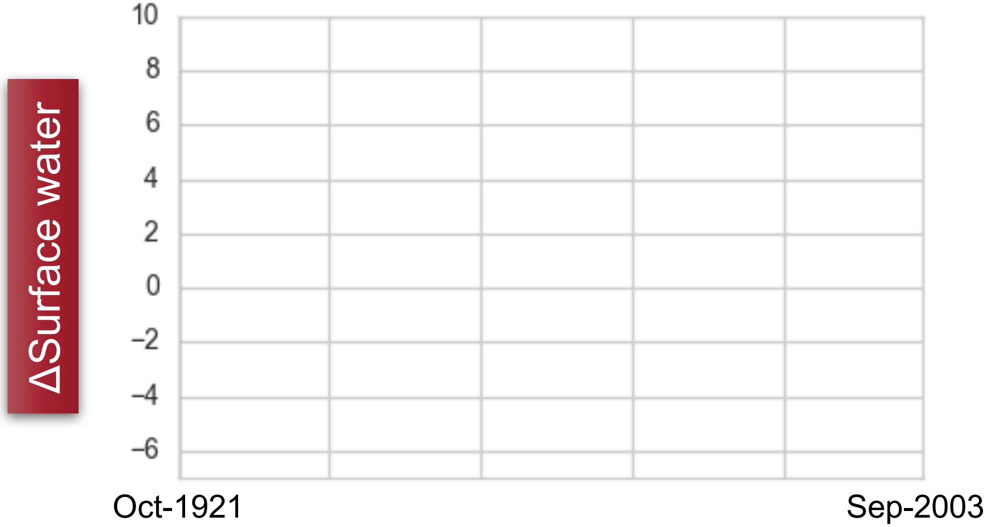




Preliminary Results

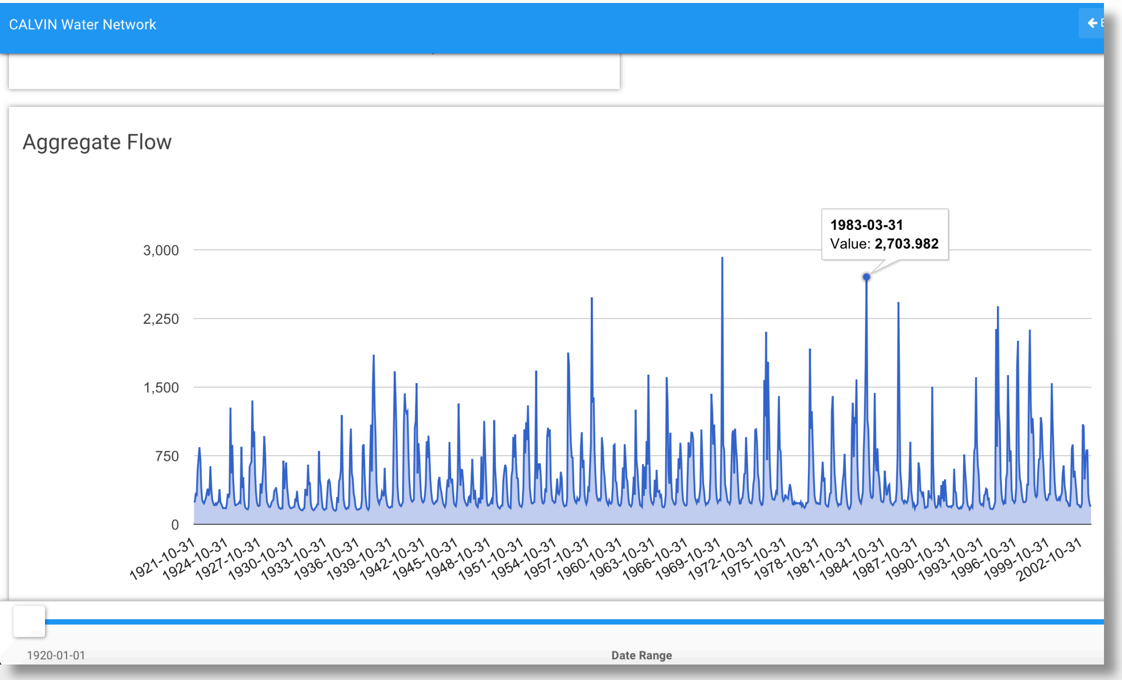


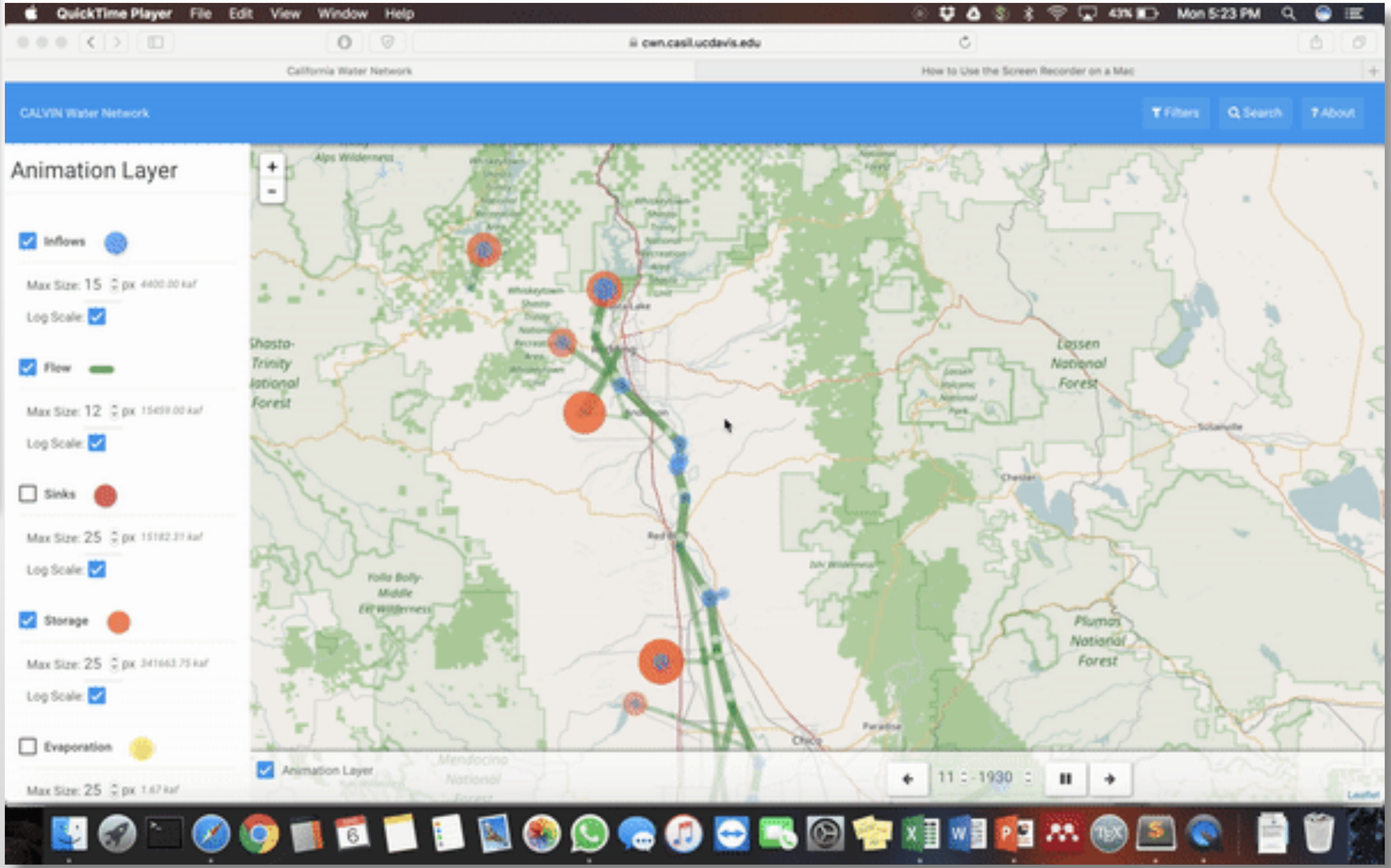
Monthly Change in Overall Storage (MAF)



Data Visualization

- Animation tool
- Inflow, flow, storage, evaporation





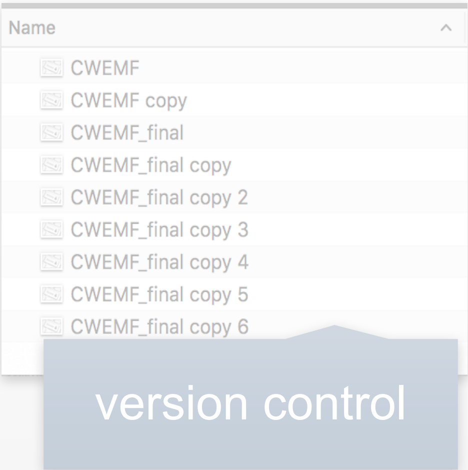
- Graph flow and storage results
- Select time-period

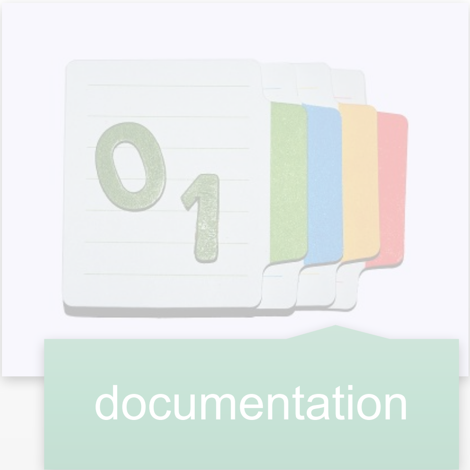
Limitations & Improvements

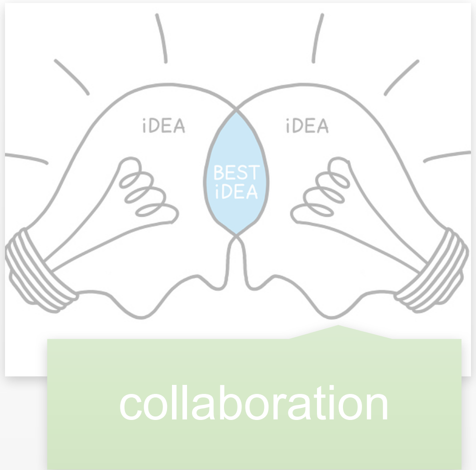
- Perfect foresight
  - Run year-by-year to prevent perfect hydrologic foresight
- Groundwater representation
  - Implement SGMA constraints on groundwater basins
- Water rights
  - Represent water rights in system operations

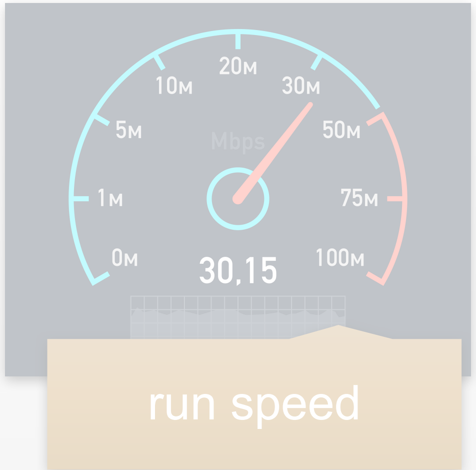
Conclusions

- Free and publicly available data and model
- Version control and tracking changes via GitHub
- Online documents, theses, journal papers
- Communication between modelers & models
- Fast, state-of-the-art solvers; cplex, gurobi and cbc









Useful links

- CALVIN: https://calvin.ucdavis.edu/node
- HOBBES: https://hobbes.ucdavis.edu/node & https://cwn.casil.ucdavis.edu
- PyVIN: https://github.com/msdogan/pyvin
- Network data: https://github.com/ucd-cws/calvin-network-data
- Network tools: https://github.com/ucd-cws/calvin-network-tools

Developers:
Mustafa Dogan, Max Fefer, Ellie White, Jon Herman, Justin Merz
Quinn Hart, Josue Medellin-Azuara and Jay Lund
contact: msdogan@ucdavis.edu