



Spring 2017 PyVIN Shortcourse & Hands-on Tutorial

Mustafa Dogan

Center for Watershed Sciences, UC Davis

Spring 2017



Agenda

- CALVIN theory & model introduction
 - **break**
 - HOBES database
 - Data flow overview
 - **Break (lunch)**
- PyVIN updates & model architecture
 - **break**
 - First PyVIN run
 - Postprocessing results

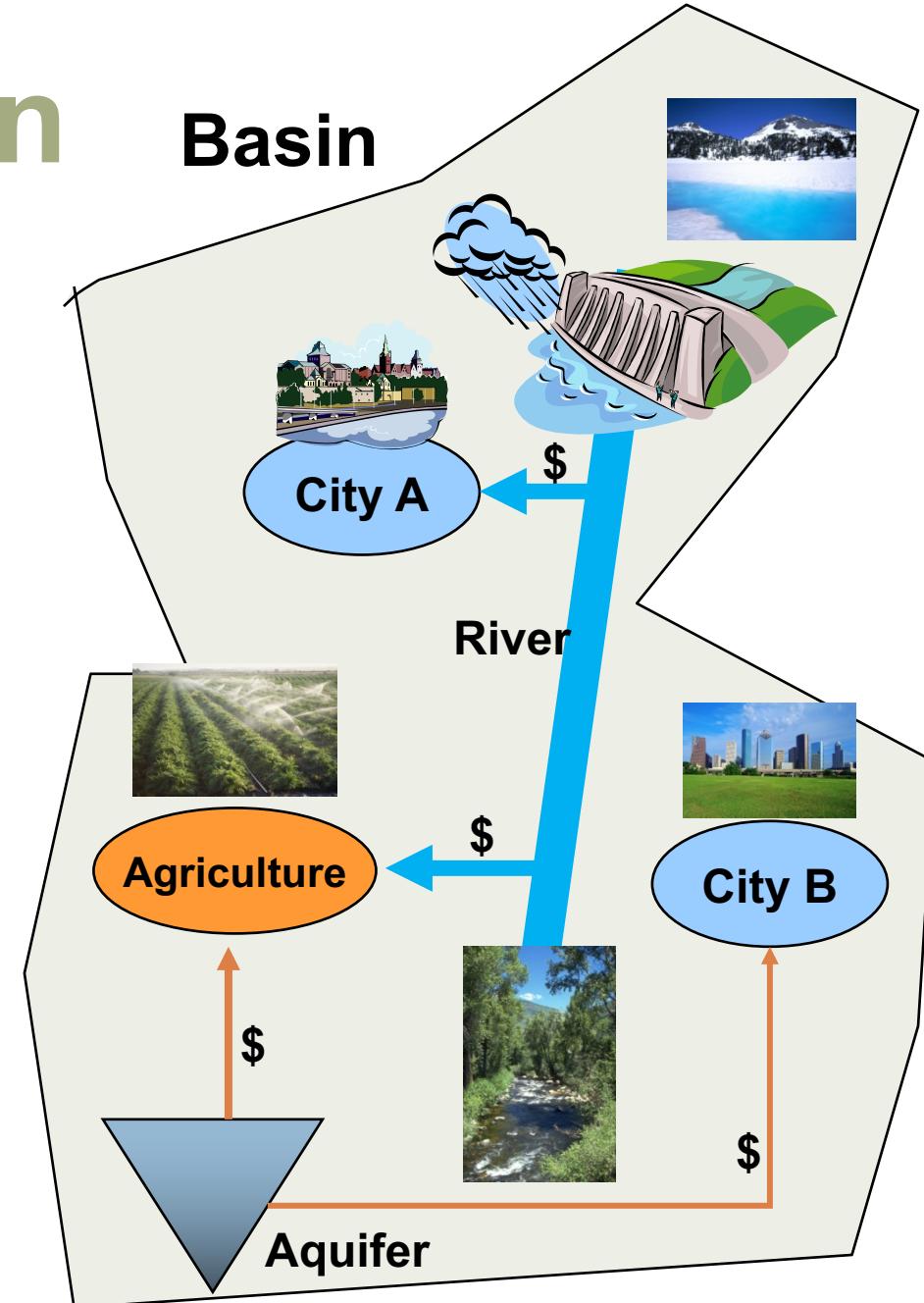
CALVIN Theory & Model Introduction

CALifornia Value Integrated Network

- Hydroeconomic optimization model
- Represents California's water operations
- Optimizes water deliveries to users
- Surface and groundwater representation
- Historical monthly hydrology (from Oct 1921 to Sep 2003)
- 2050 agricultural and urban demands

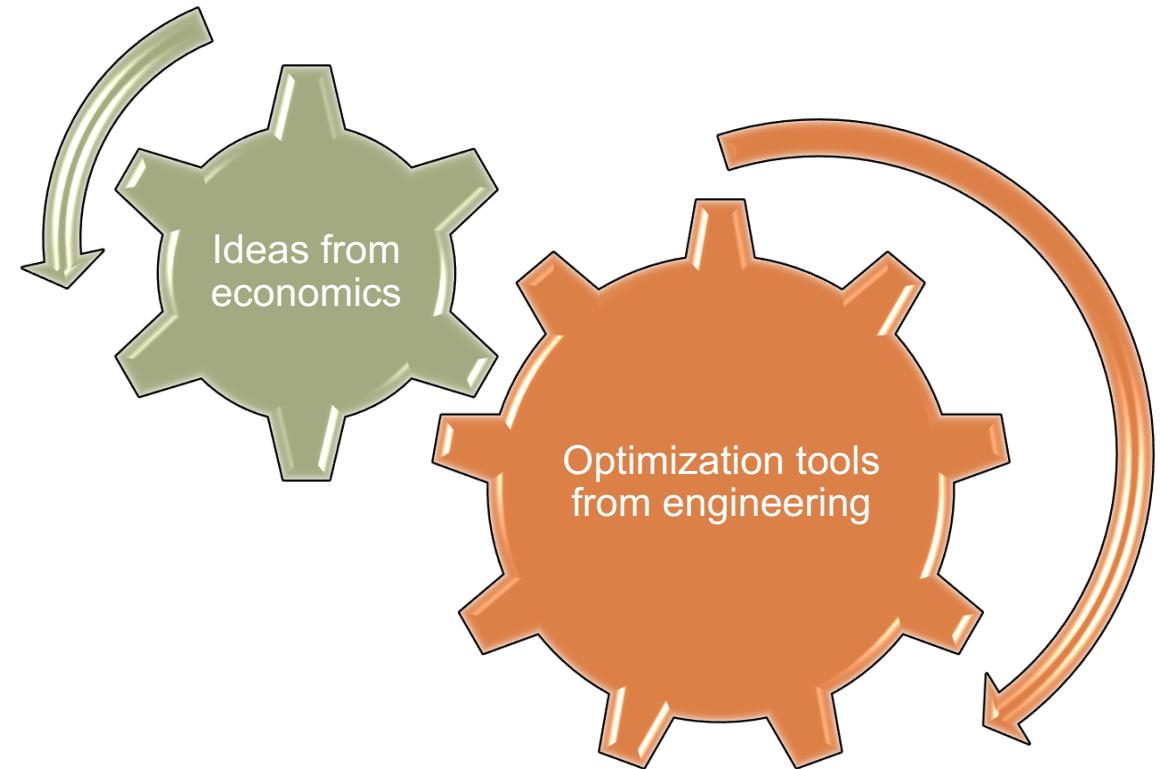
Water Allocation

- Economic values for agricultural, urban and hydropower uses
- Constraints for capacity limitations and regulatory requirements
- 82-year hydrology and operations
- Monthly time-step



CALVIN

- Forces quantitative understanding of integrated water and economic system
- Provides insights into regional and statewide water planning and management

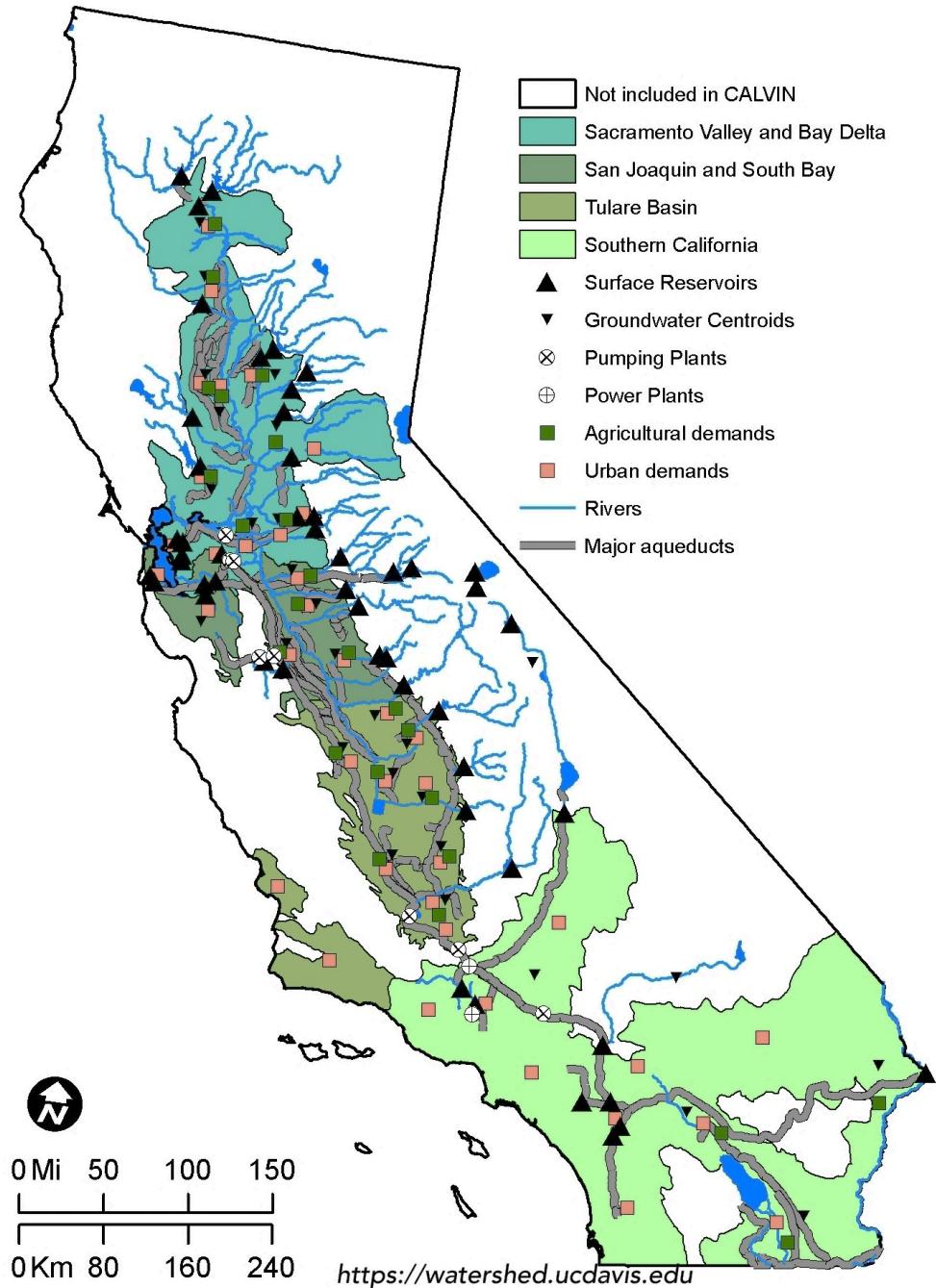


Motivation

- Make better sense of integrated system and operations
- Seek ways to improve system management
- Quantifying user willingness to pay for additional water
- Find insights into changes in physical capacities and policies

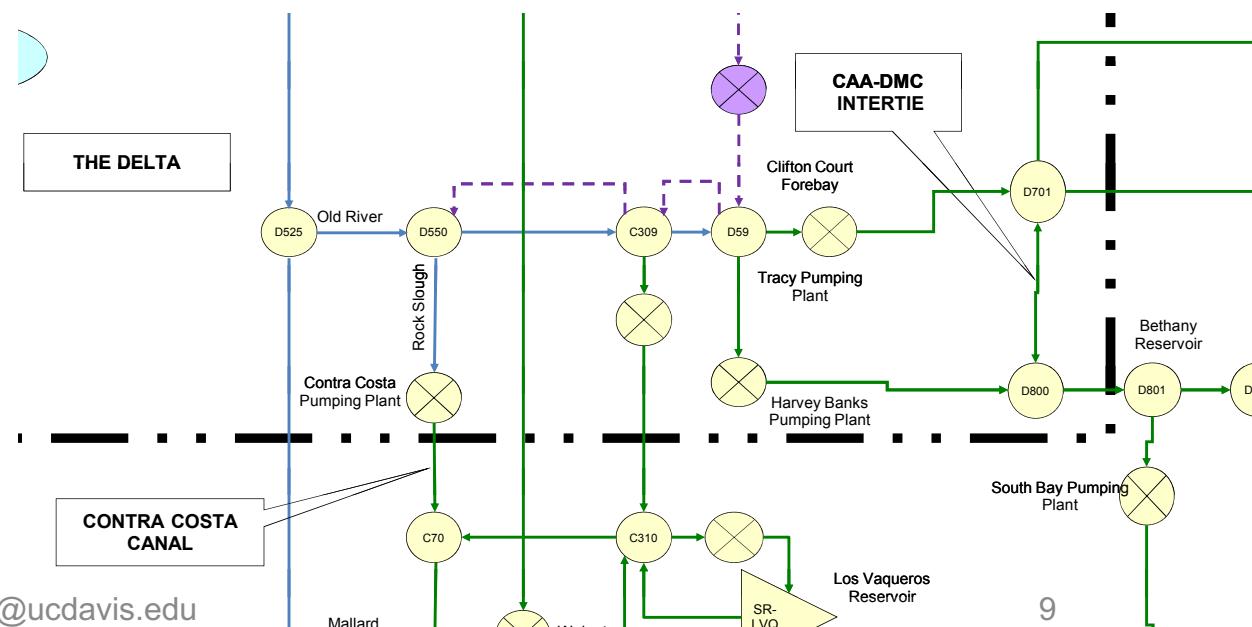
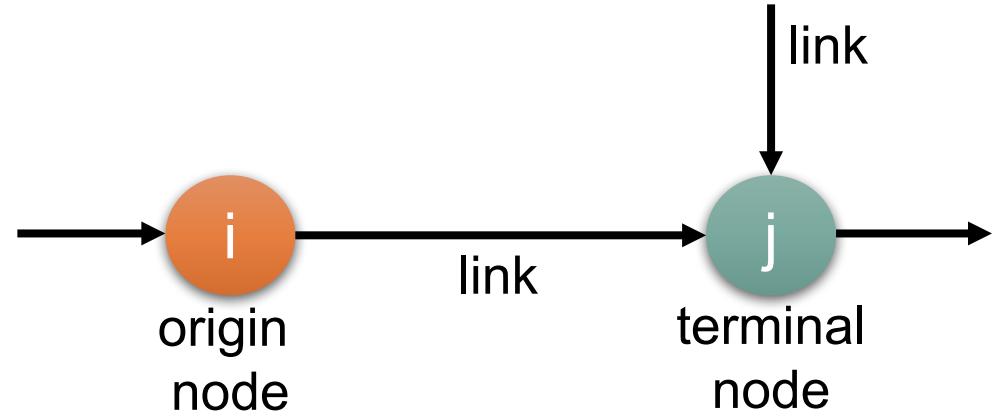
Model Coverage

- Most agricultural activities and urban population
- Major conveyance systems (CAA, DMC, CRA)
- CVP and SWP operations
- Surface and groundwater reservoirs



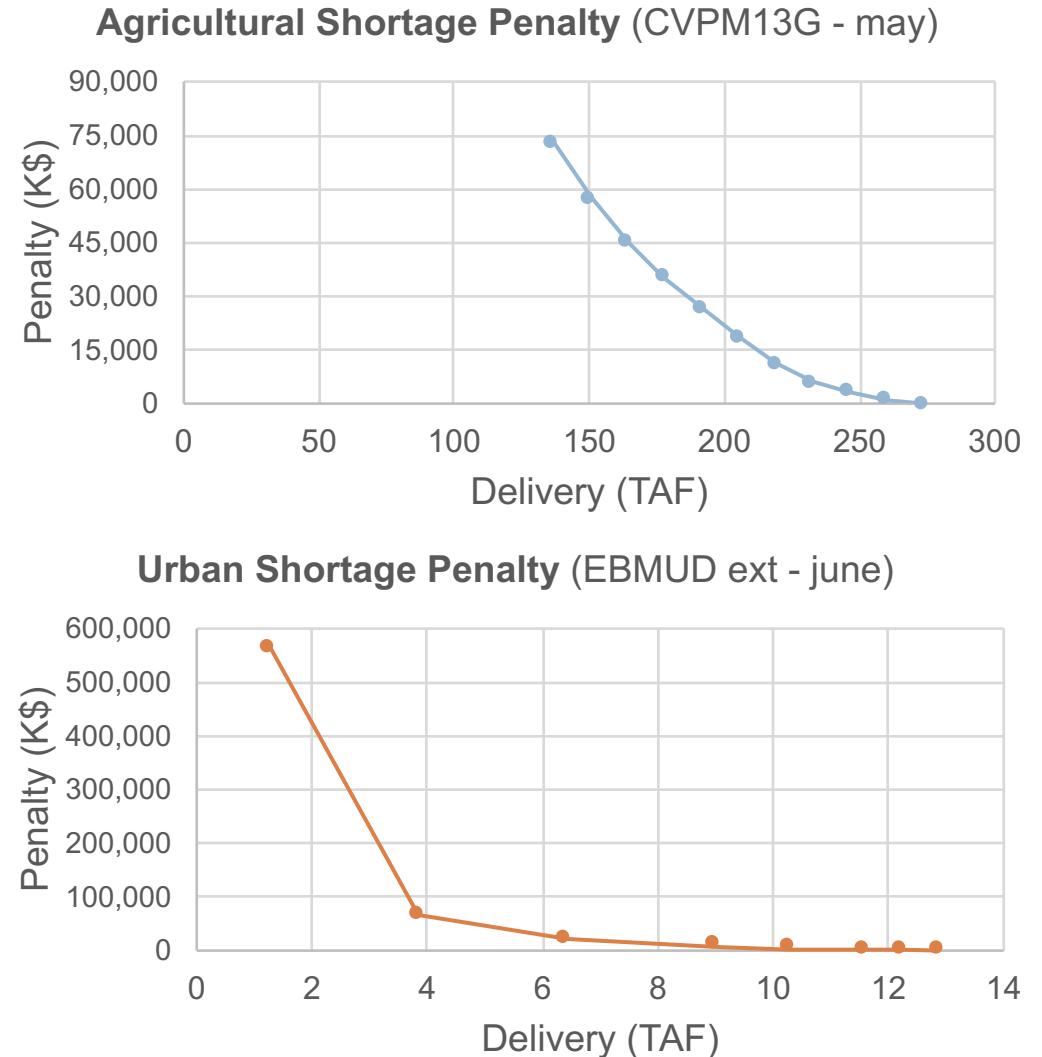
Network-Flow Structure

- Nodes
 - Reservoirs, plants, demand areas
- Links
 - Capacity (upper bound), minimum flow (lower bound), loss (amplitude), cost
- CALVIN network schematic



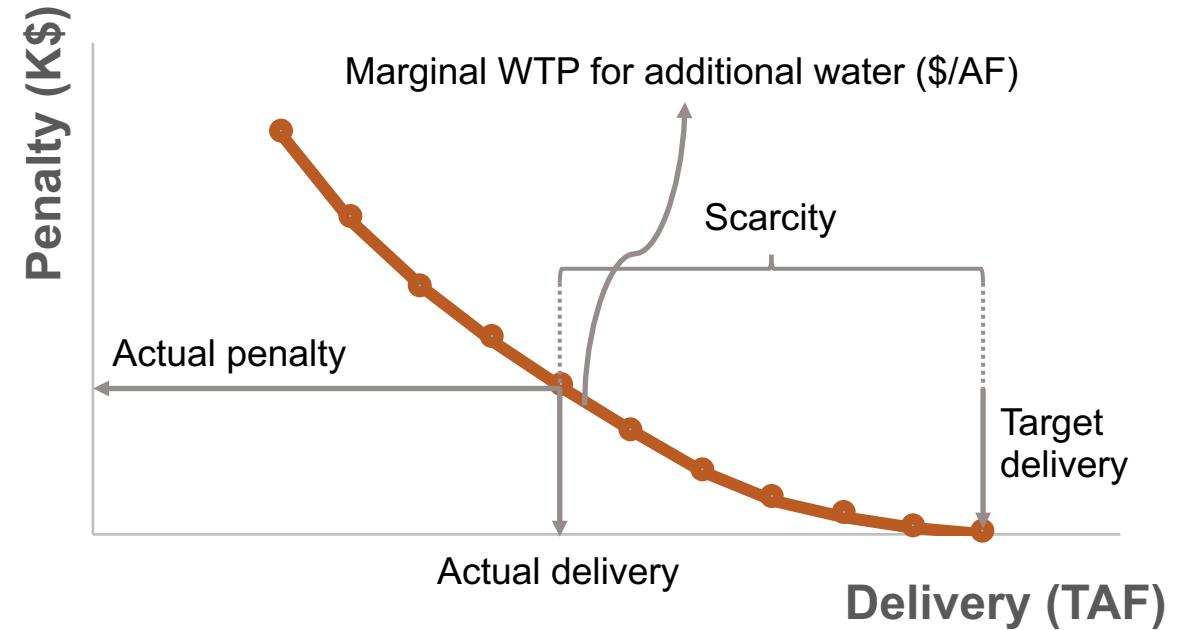
Water Users: Demand Areas

- Users
 - Agricultural
 - Urban
 - Environmental
- Economic representation for ag and urban
 - Penalty for not delivering water
 - Urban users with higher WTP
- Constrained (fixed) deliveries for environment (wildlife refuges)



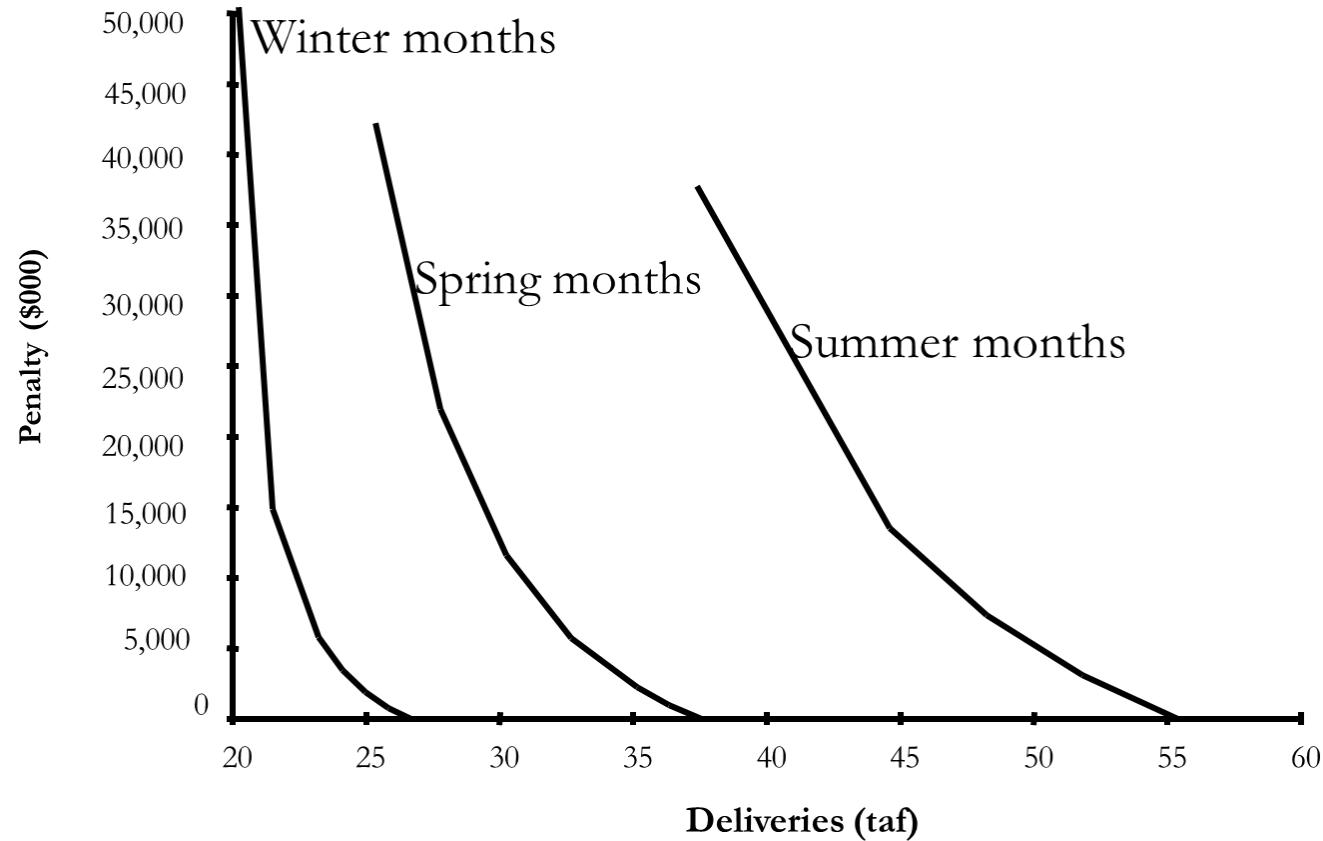
Demand Area Penalty Curves

- Penalty for not delivering water (and economic loss)
- No penalty at the target delivery
- Piecewise linear curves (linear programming model)
- User's willingness to pay (WTP) for water
- Convex: the more delivery the less penalty



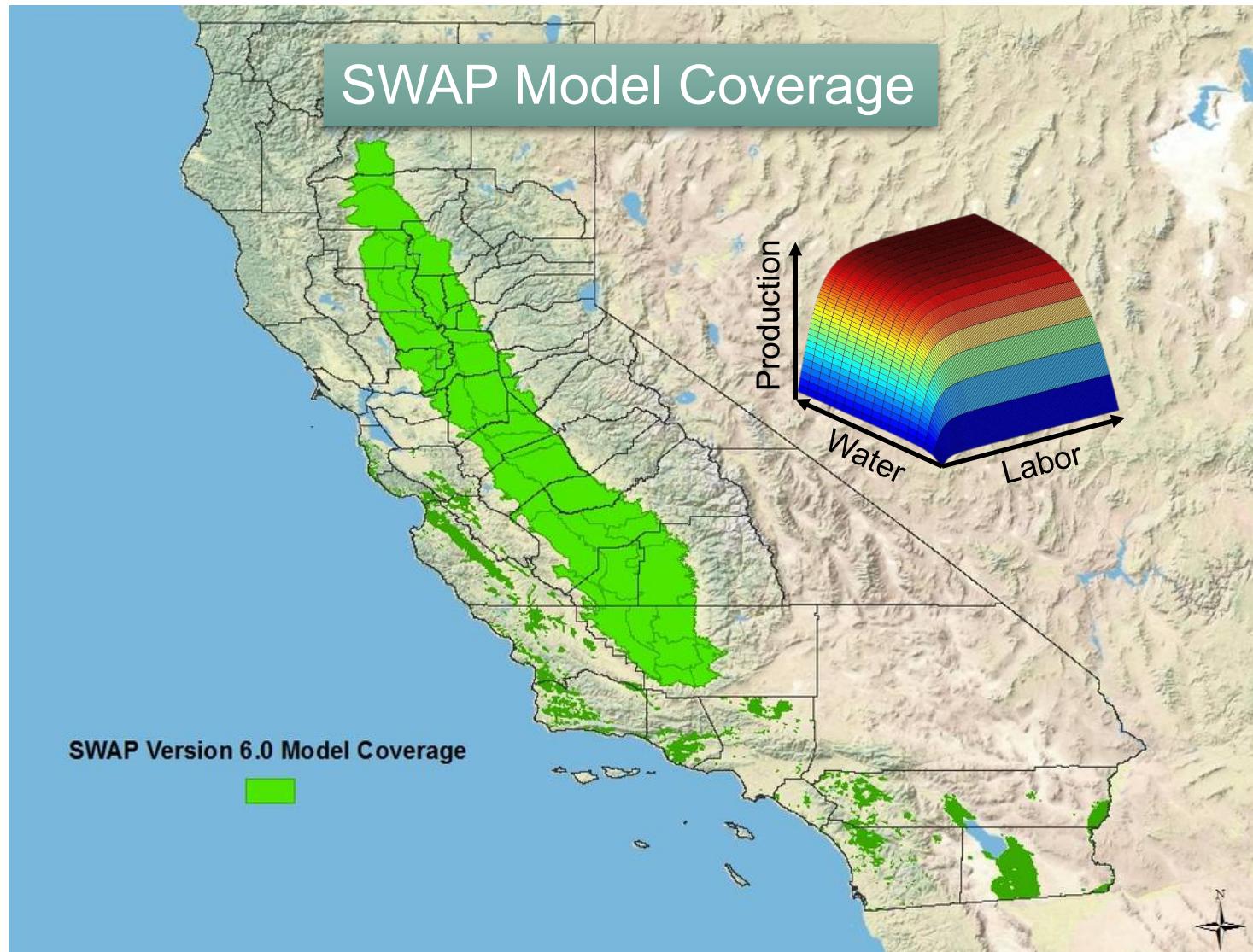
Monthly Varying Demand Curves

- Demand curves vary by month (no annual variation)
- Higher target demand in summer months
- Less price elasticity (steeper slope) in winter months



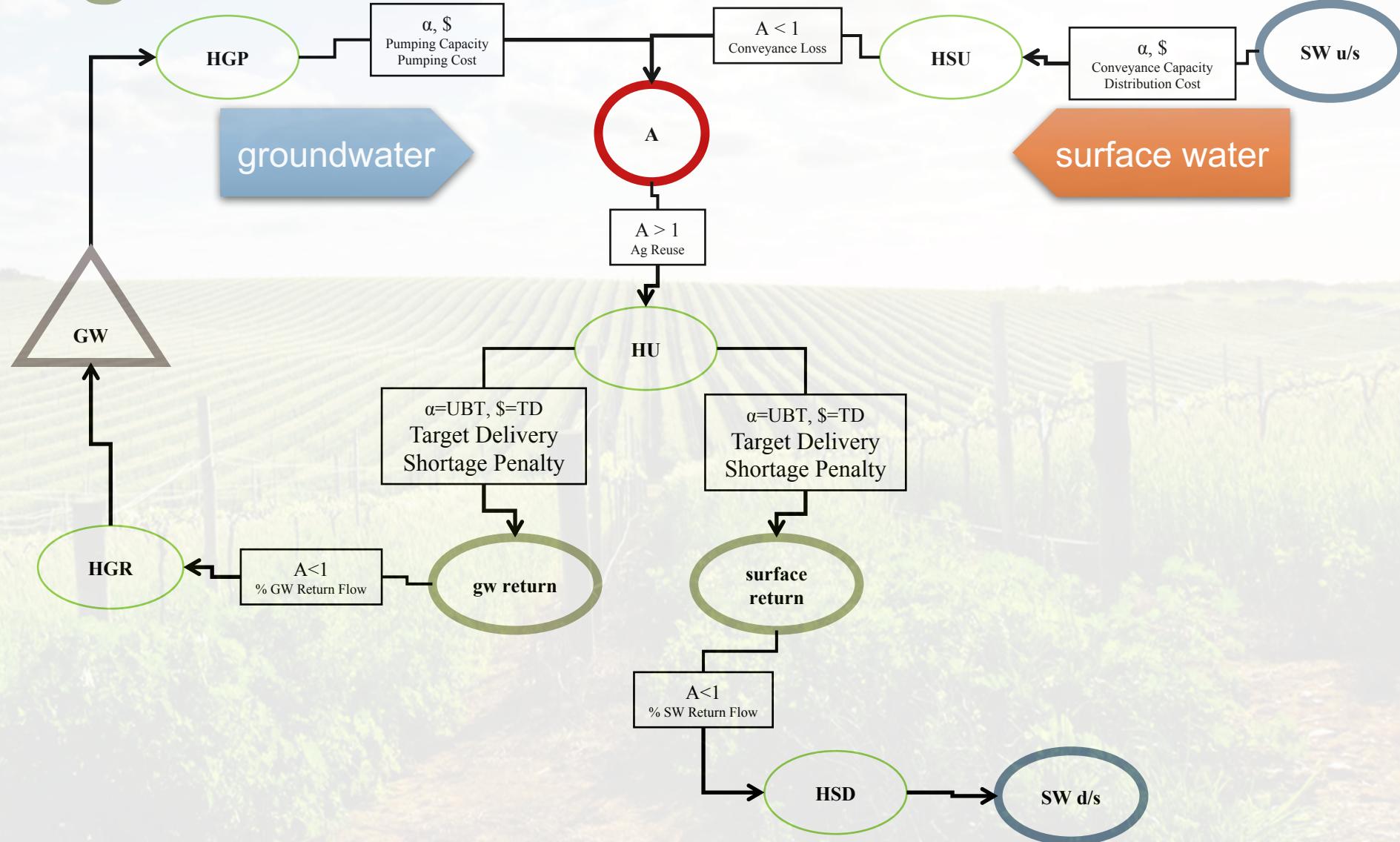
Agricultural Representation

- Results from Statewide Agricultural Production (SWAP) model
- SWAP simulates farmers decision in California
- Profit maximization model
- 35 aggregated subbasins
- SWAP results are used to create ag penalties in CALVIN (& PyVIN)



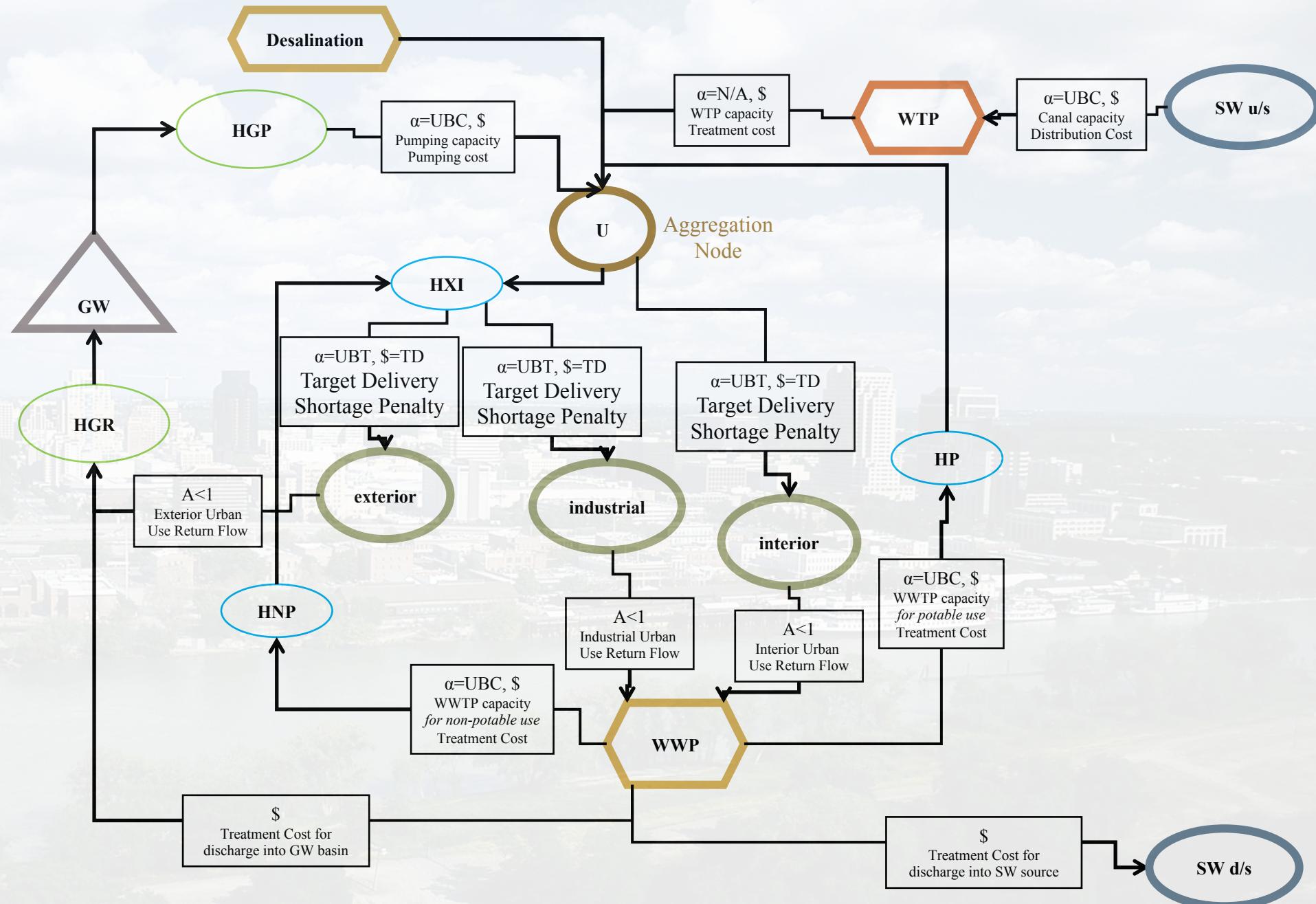
Agricultural Demand

- Supply sources:
 - Groundwater
 - Surface water
- Split into two parts depending on return flows



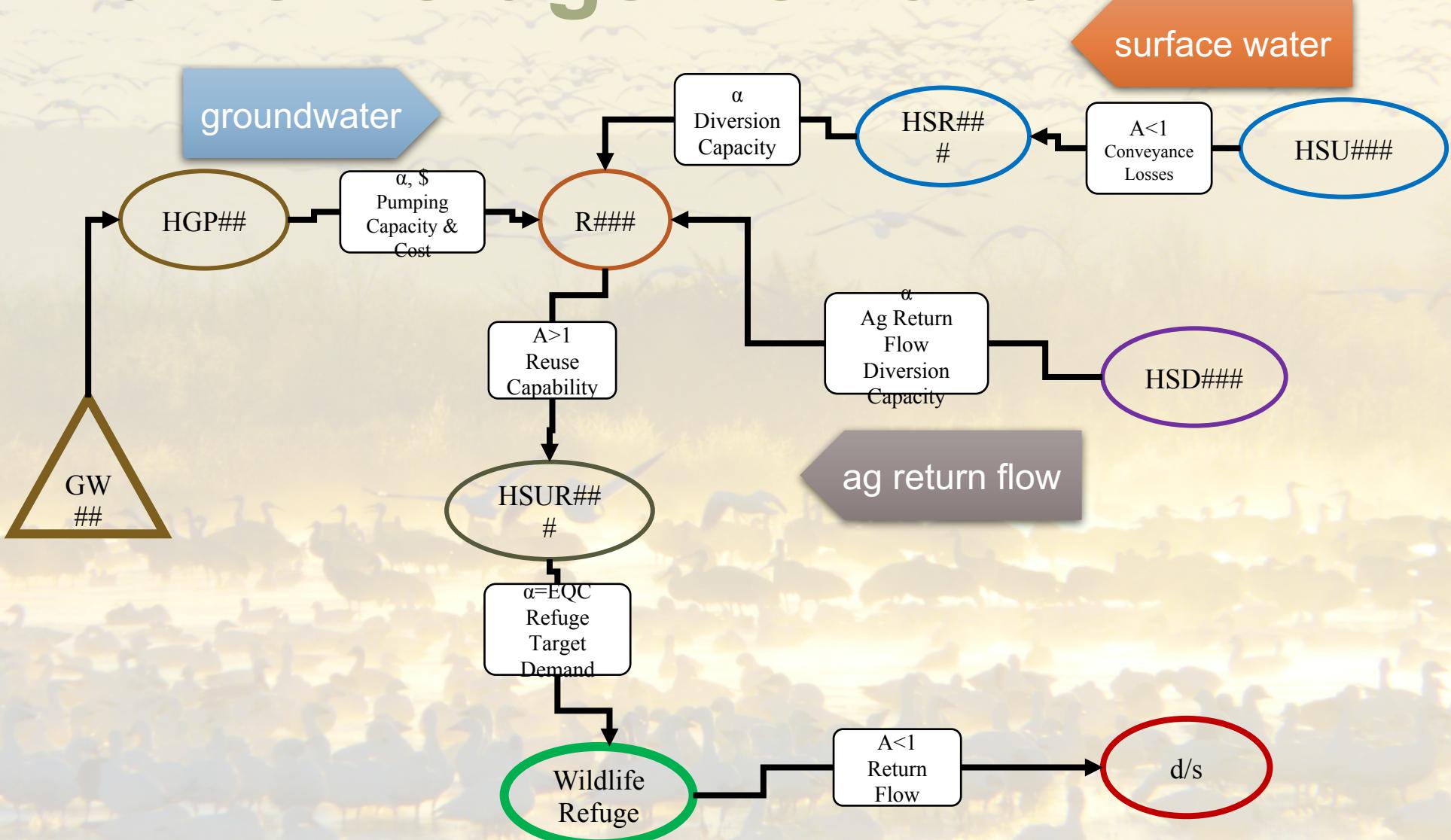
Urban Demand

- Supply sources:
 - Groundwater
 - Surface water
- Split into three parts
 - Exterior
 - Interior
 - Industrial



Wildlife Refuge Demand

- Supply sources:
 - Groundwater
 - Surface water
 - Ag return flow
- Constrained (fixed) flow deliveries



Advantages

- Integrated model (surface and groundwater)
- Unique combination of economics and engineering principles
- Well-documented model (reports, articles, website and theses)

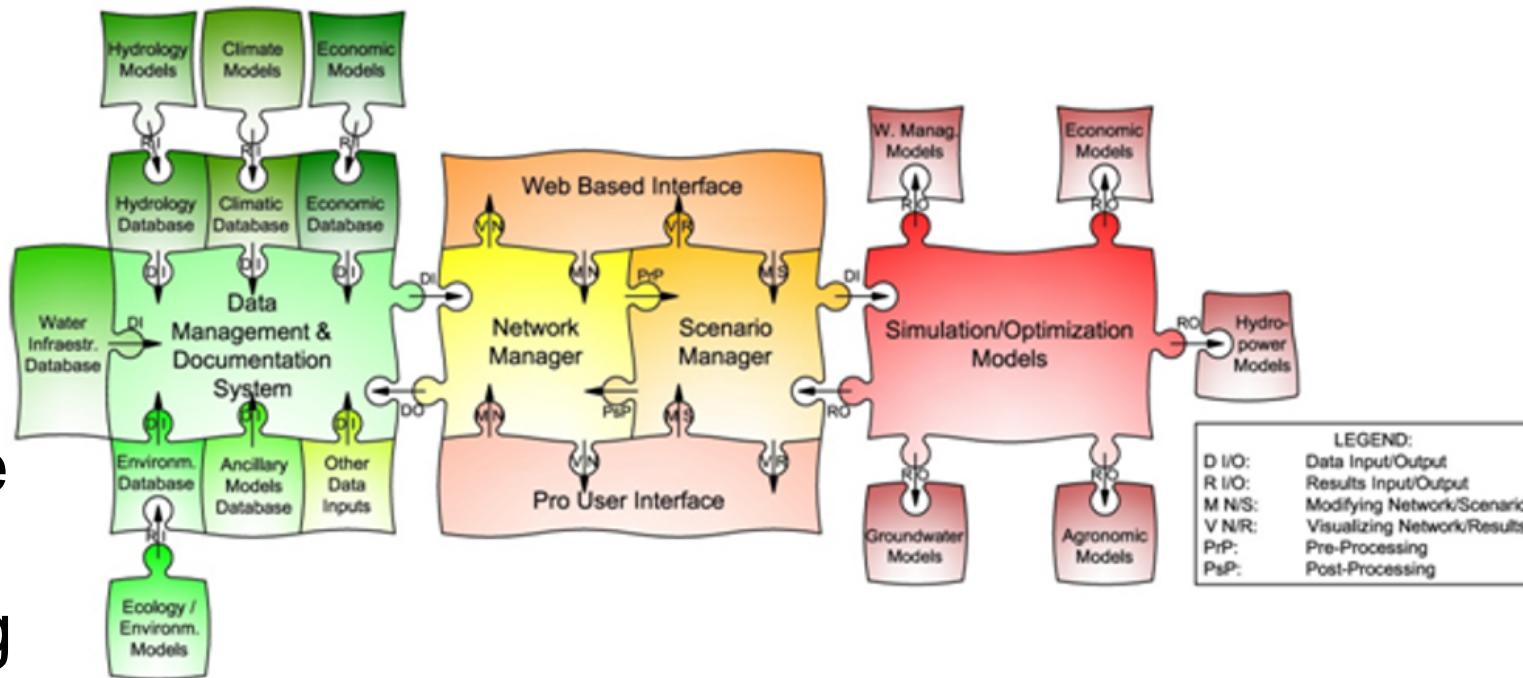
Limitations

- Perfect hydrologic foresight
 - The model knows all hydrologic events in its operating period
 - Scarcity costs are less than actual
- Simplified reservoir and Delta operations
- Piecewise linearization
- Simplified hydropower representation

HOBES Database

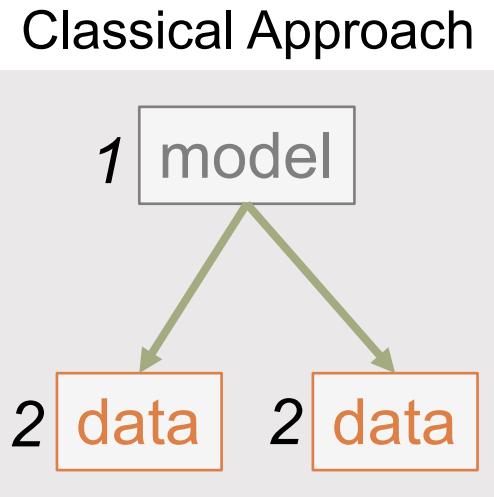
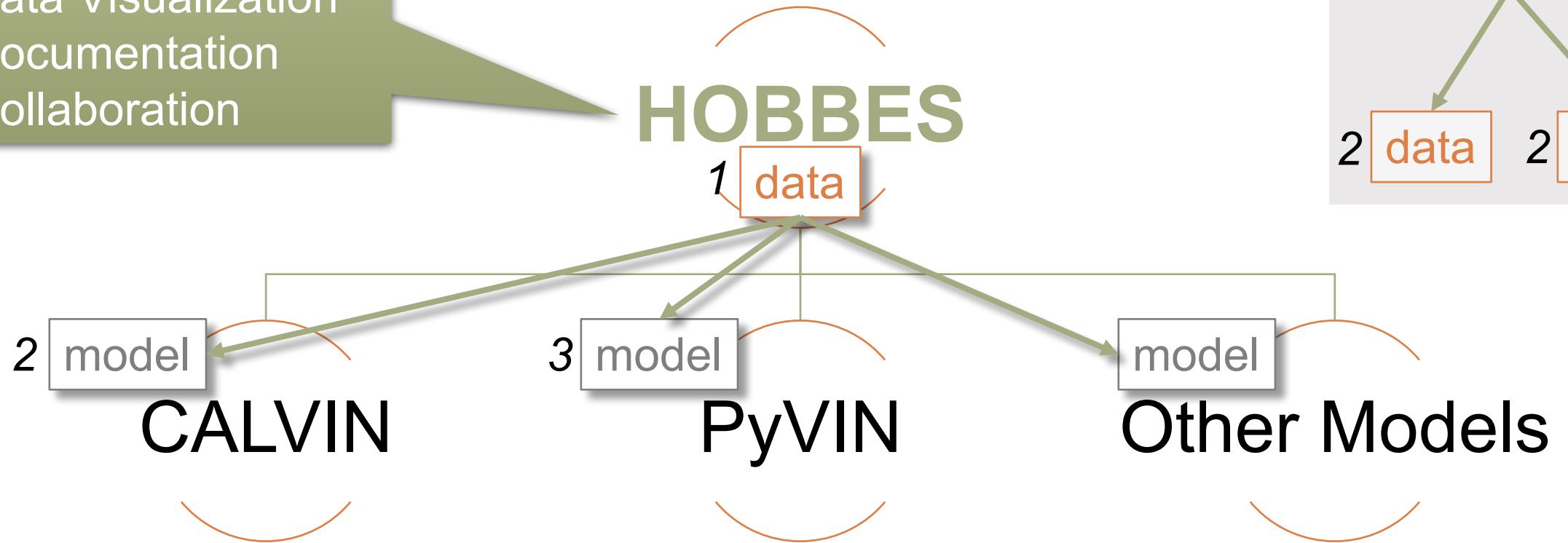
HOBBISS

- A bottom up approach to improve and organize data
- Venue for modelers to create an open, organized and documented water representation
- Geocoded elements can be converted to networks and solved by multiple modeling platforms (PyVIN)



Modeling with HOBBES

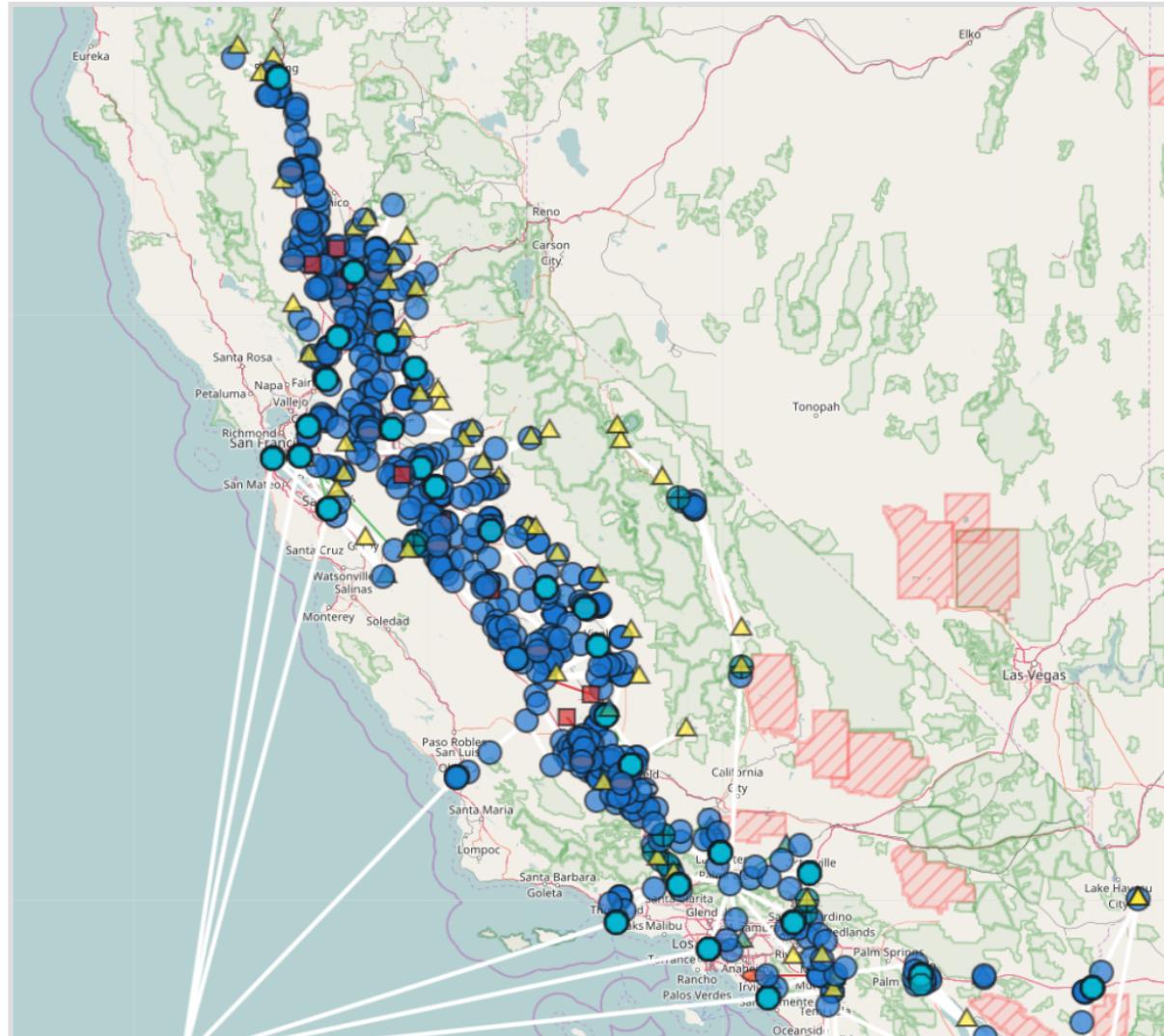
- Online Database
- Version Control
- Data Visualization
- Documentation
- Collaboration



HOBBES Website

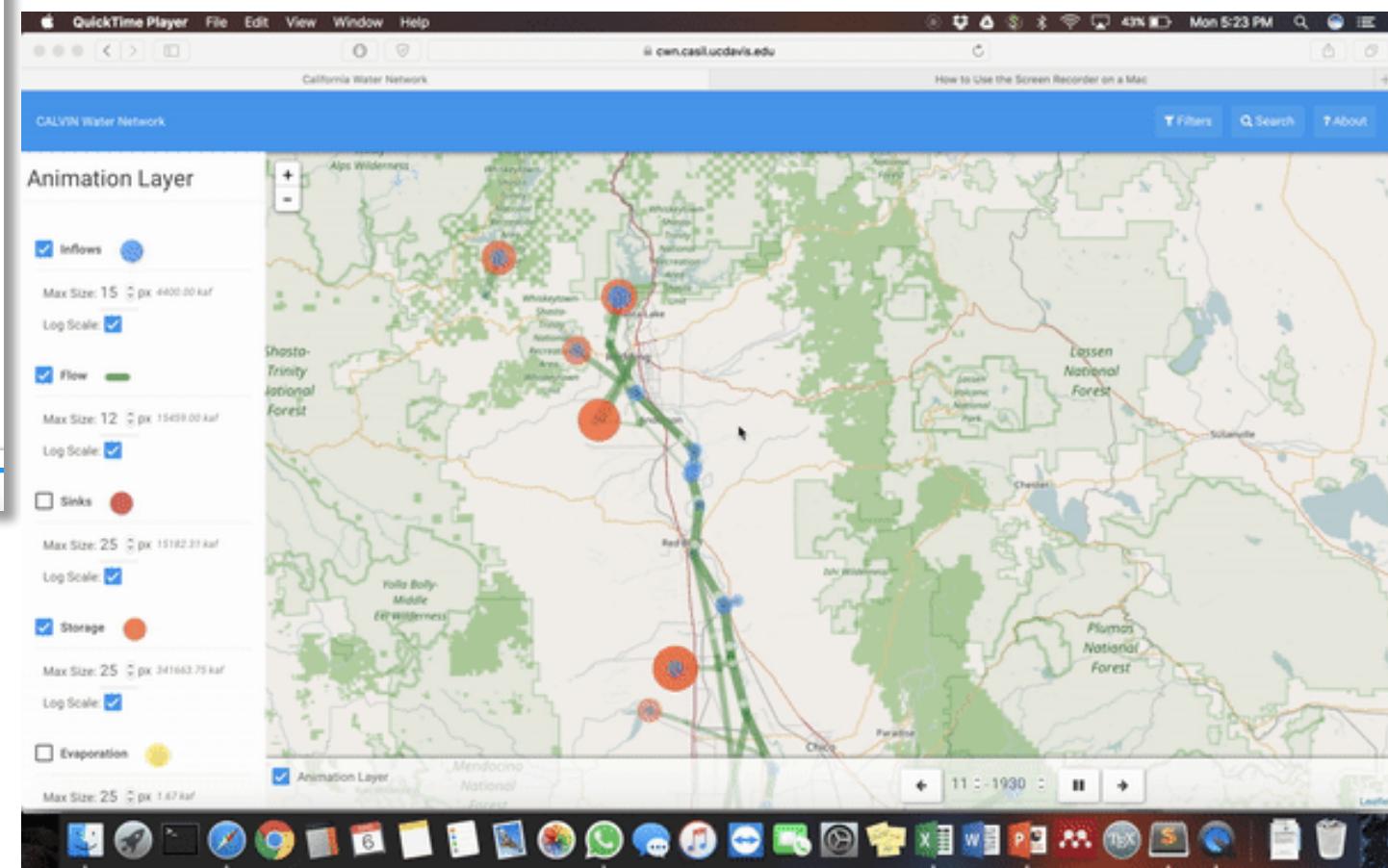
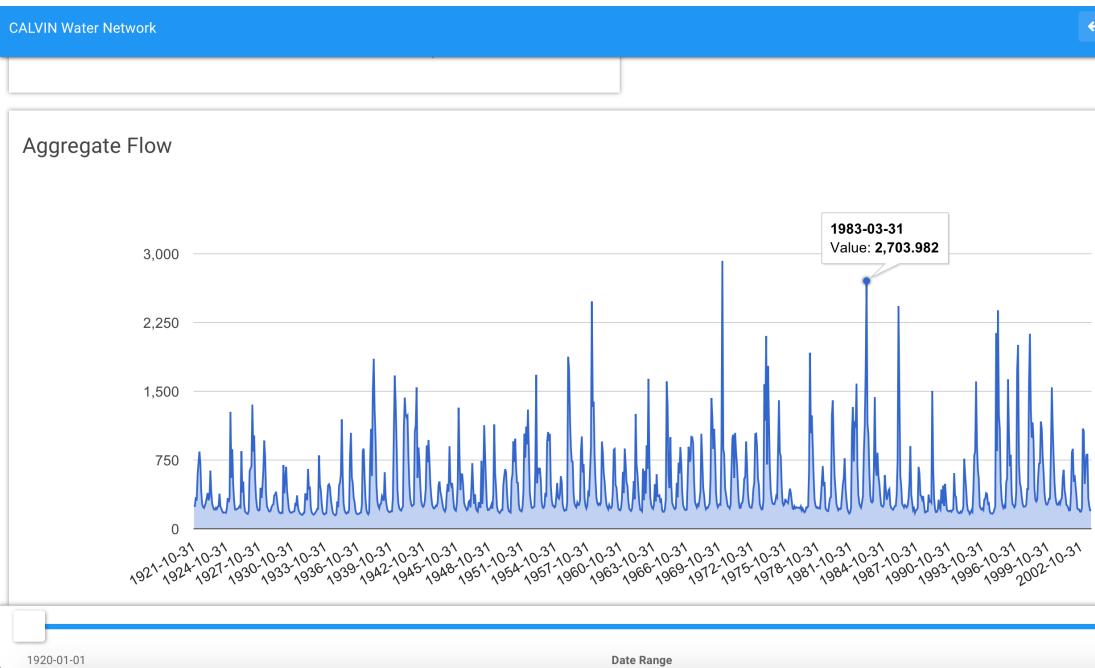
<https://cwn.casil.ucdavis.edu>

- California water network
- Geocoded network elements
- Layered based on hydrologic regions and subregions
- Data visualization
- Animation tool



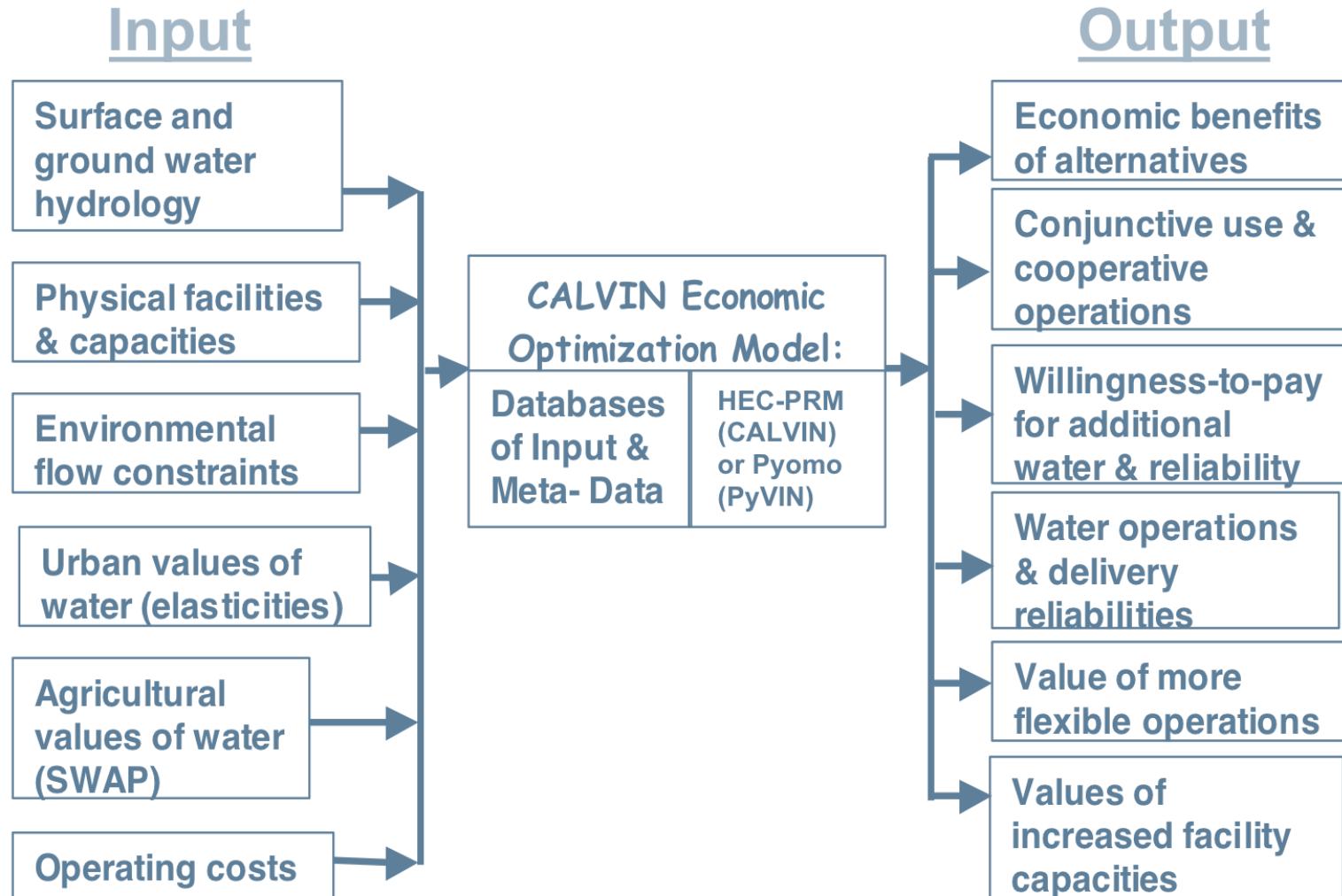
Data Visualization

- Animation tool
- Inflow, flow, storage, evaporation



- Graph flow and storage results
- Select time-period

Data Flow Overview



Hydrology

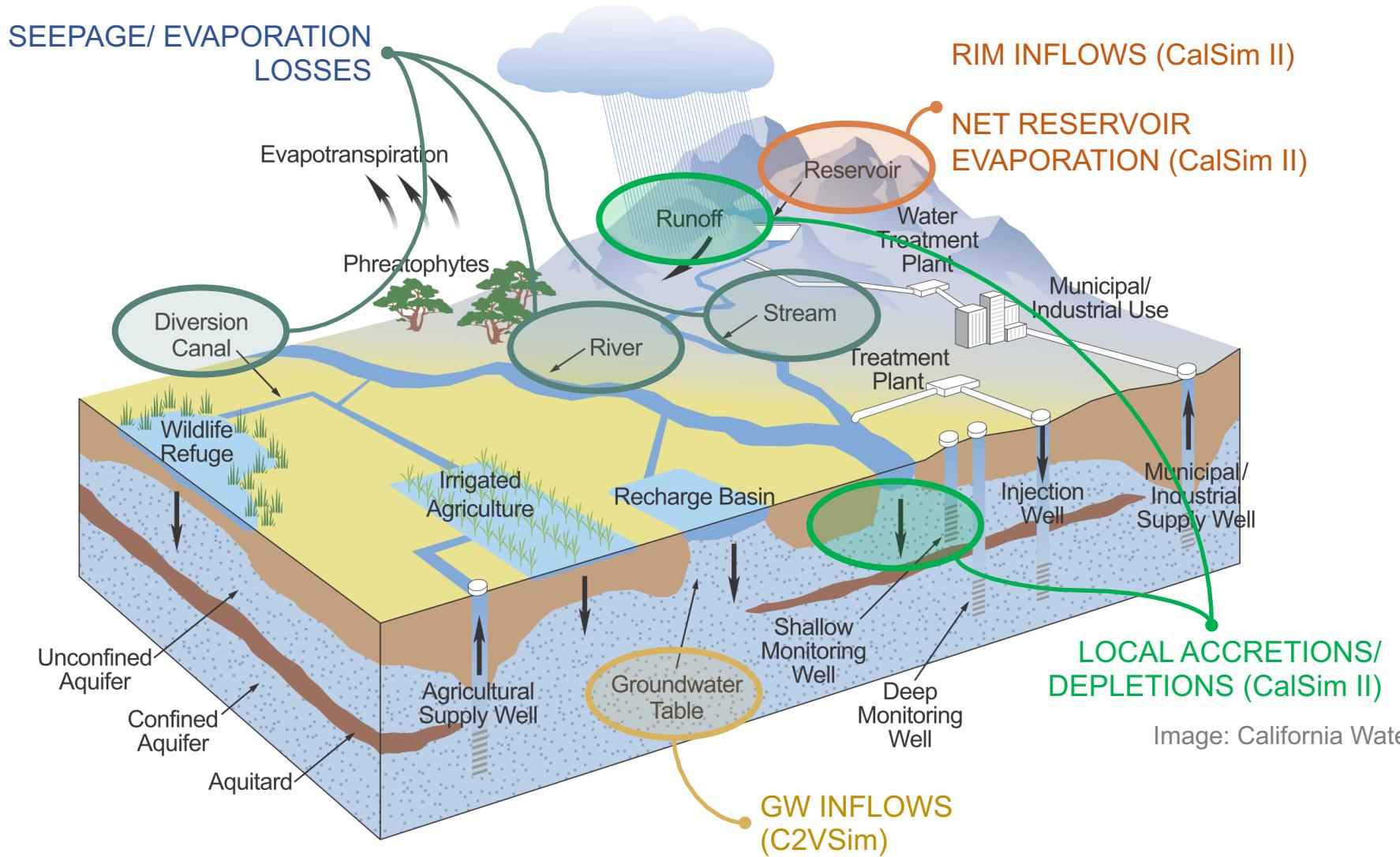


Image: California Water Plan Update 2013

Operating Cost

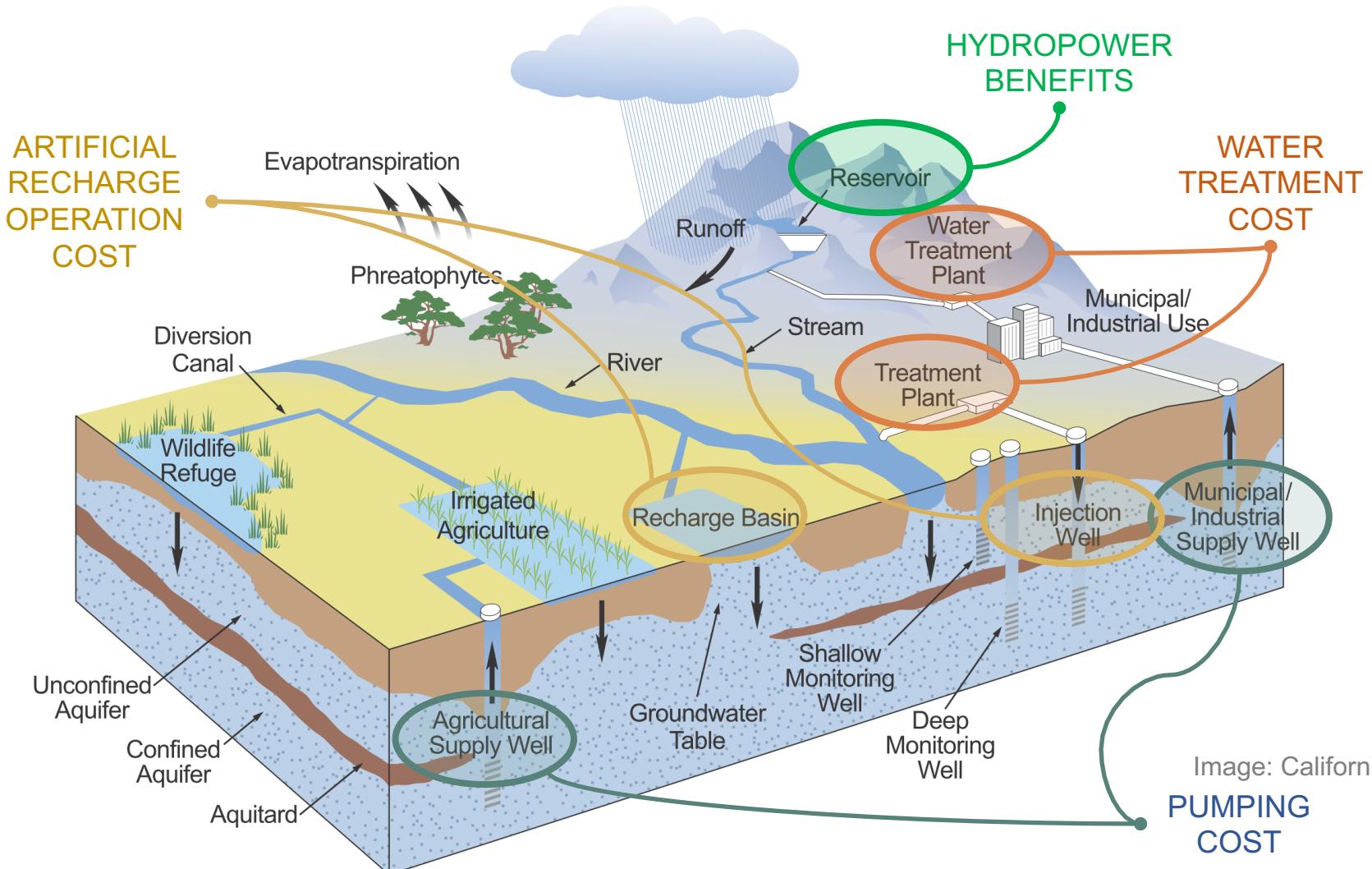
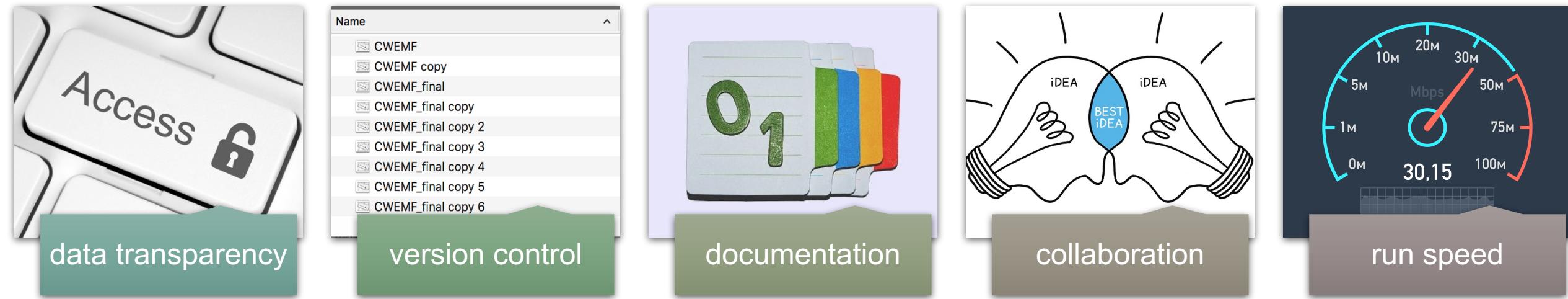


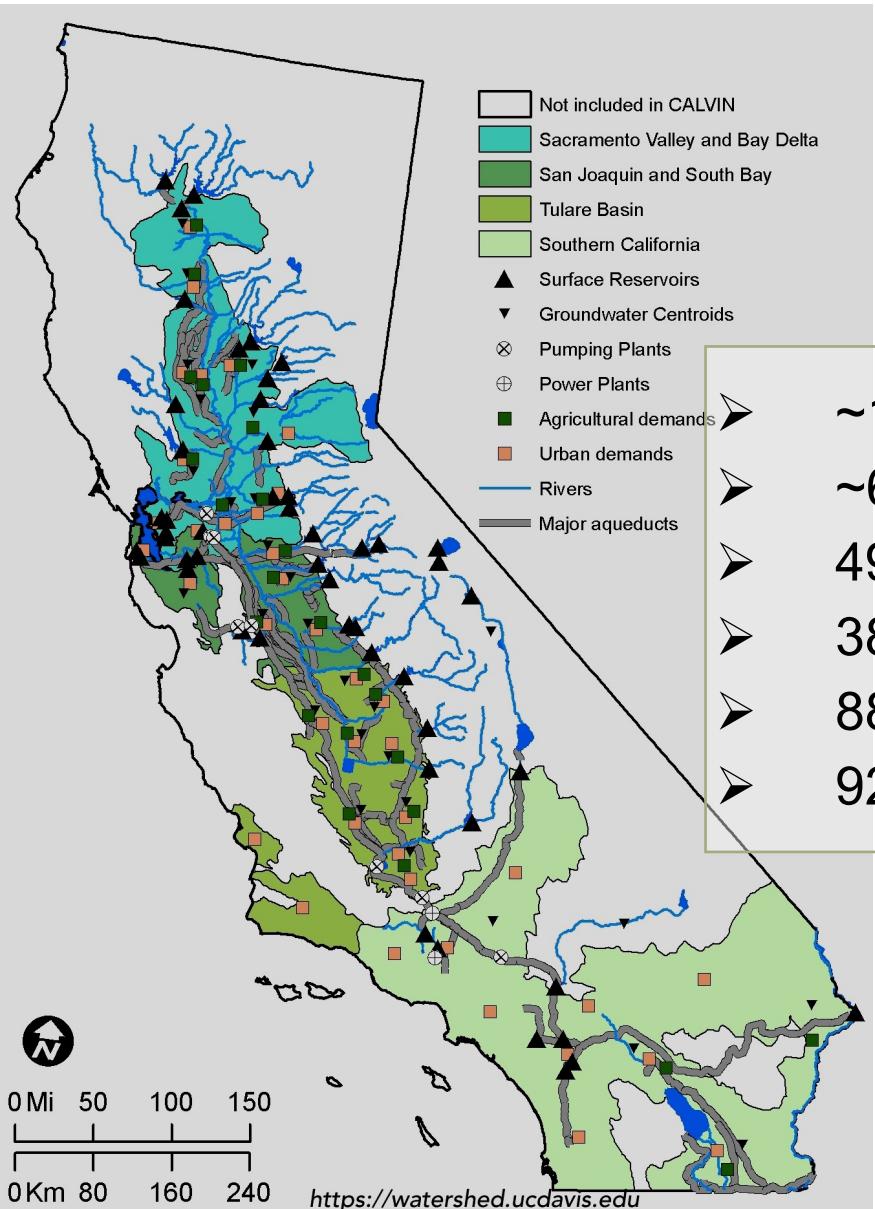
Image: California Water Plan Update 2013
PUMPING COST

PyVIN Model

Goals

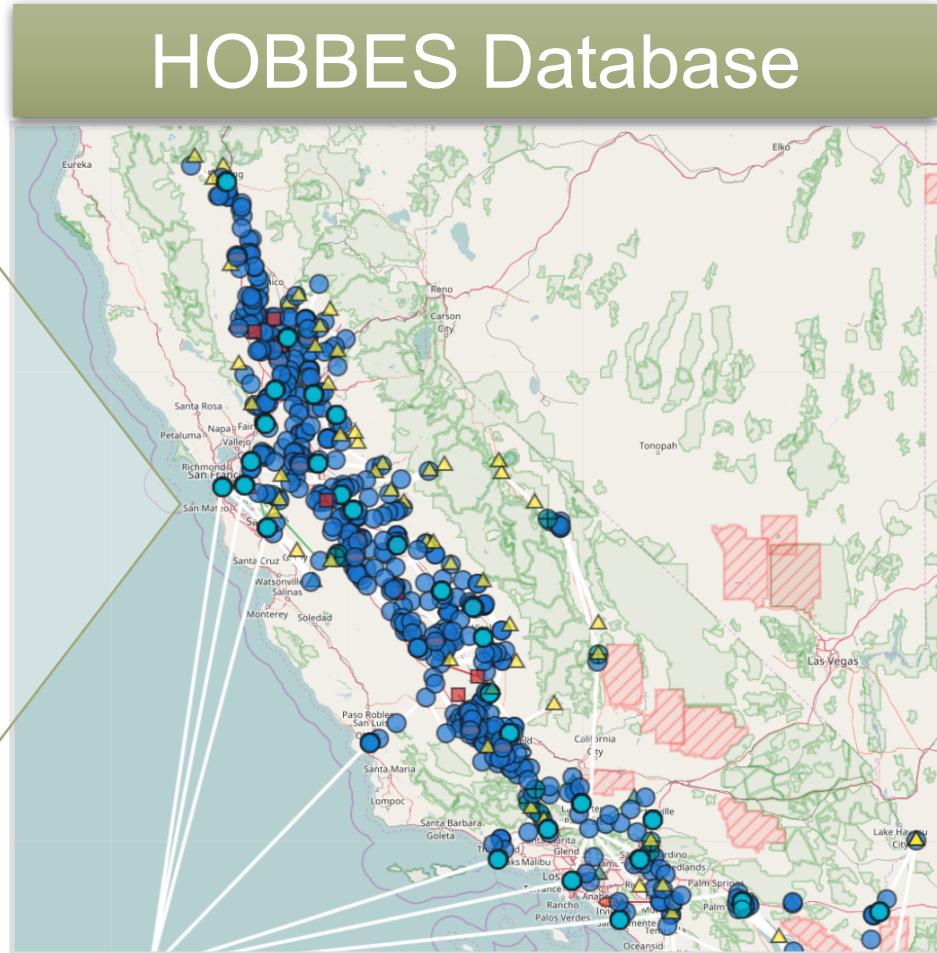


CALVIN



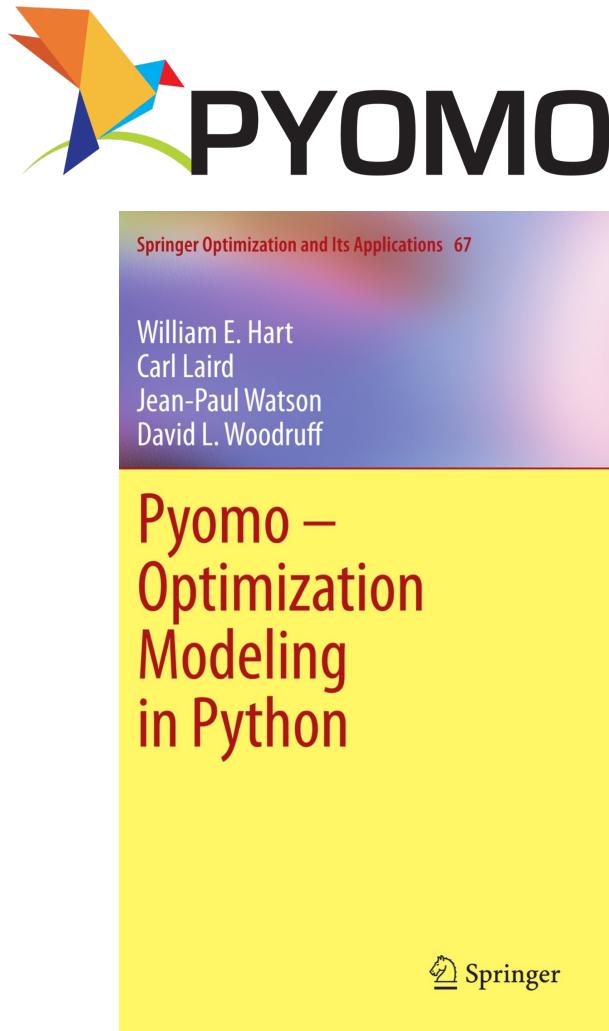
- ~1250 nodes
- ~600 conveyance links
- 49 surface reservoirs
- 38 groundwater reservoirs
- 88% of CA's irrigated acreage
- 92% of CA's urban population

HOBES Database



<https://cwn.casil.ucdavis.edu>

Modeling Environment



<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 cacheorg pyomo"
- User defined solvers:
 - CPLEX¹
 - GUROBI¹
 - CBC²
 - GLPK²

(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

Less flexible
(limited to HEC-PRM)

Solver runtime: ~16 hr
(depending on initial solution)

Requires 32 bit Windows PC

HEC-DSS database

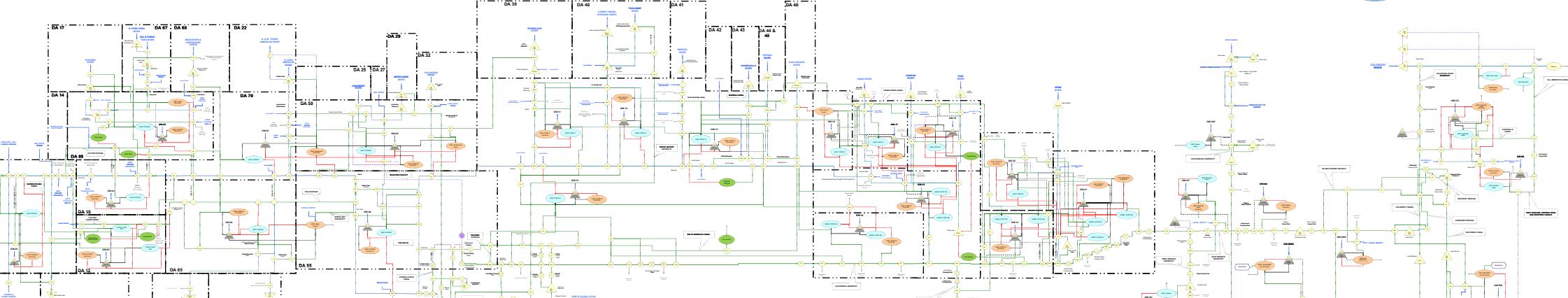
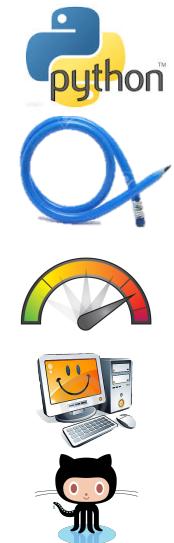
Pyomo and Python based

More flexible
(full LP)

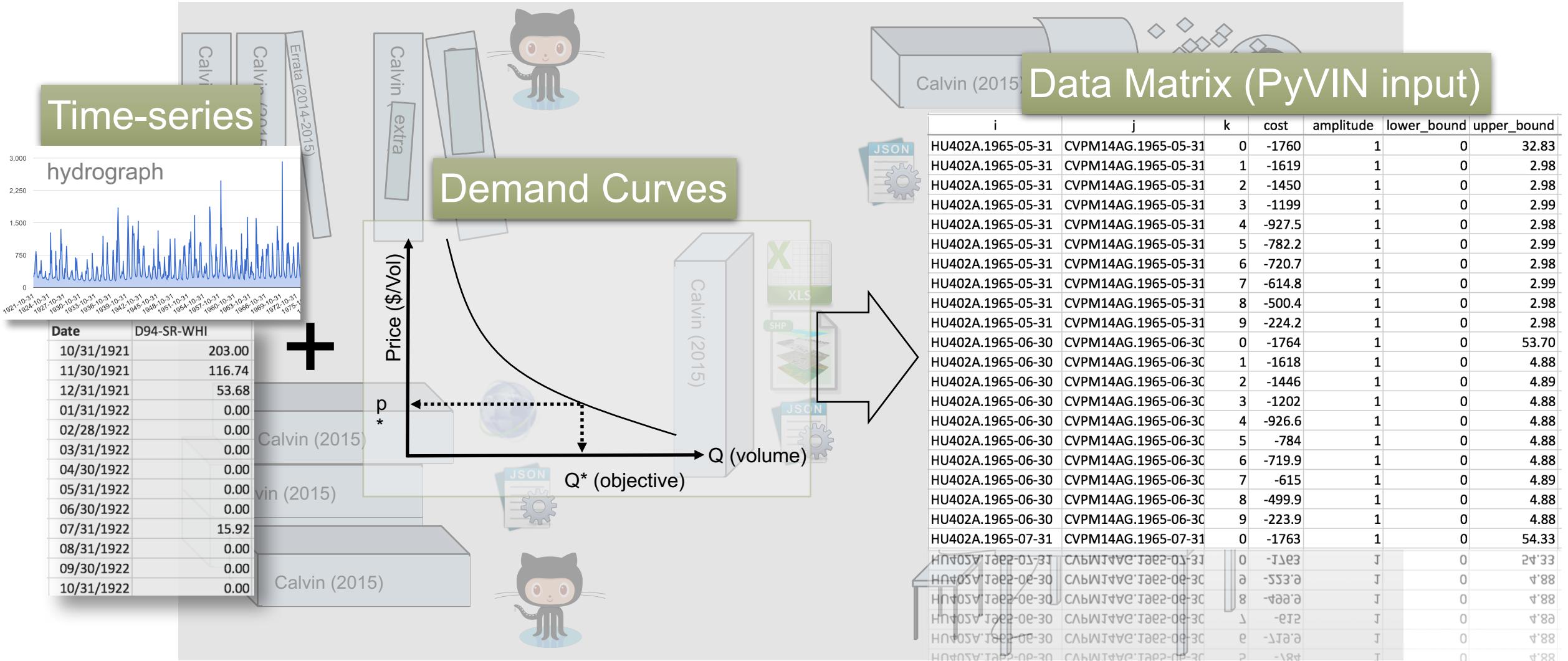
Solver runtime: ~1 min
(depending on solver)

Any computer

Open source: data and source code



HOBBISS Matrix Creator



PyVIN Model Structure

1

Linear Programming Model with Gains & Losses

objective

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

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

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

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

3

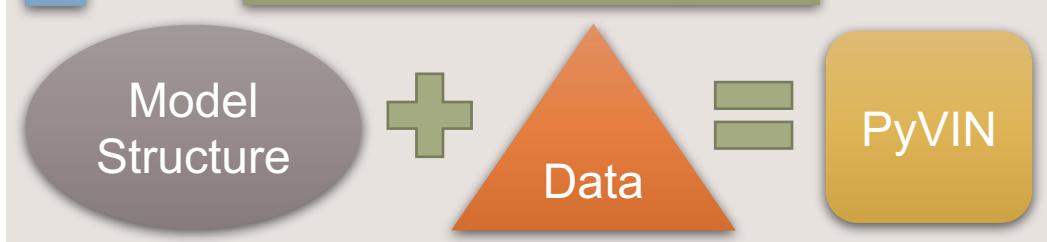
Freely Available Solvers

- CPLEX
- CBC
- GUROBI
- GLPK

User installed and defined solvers
No dependency on any solver

2

Abstract Model in Pyomo



Separate data and model structure

PyVIN Run Process



- Input Data
 - Network matrix



- Model Run
 - Specify data file and solver



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



- 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)

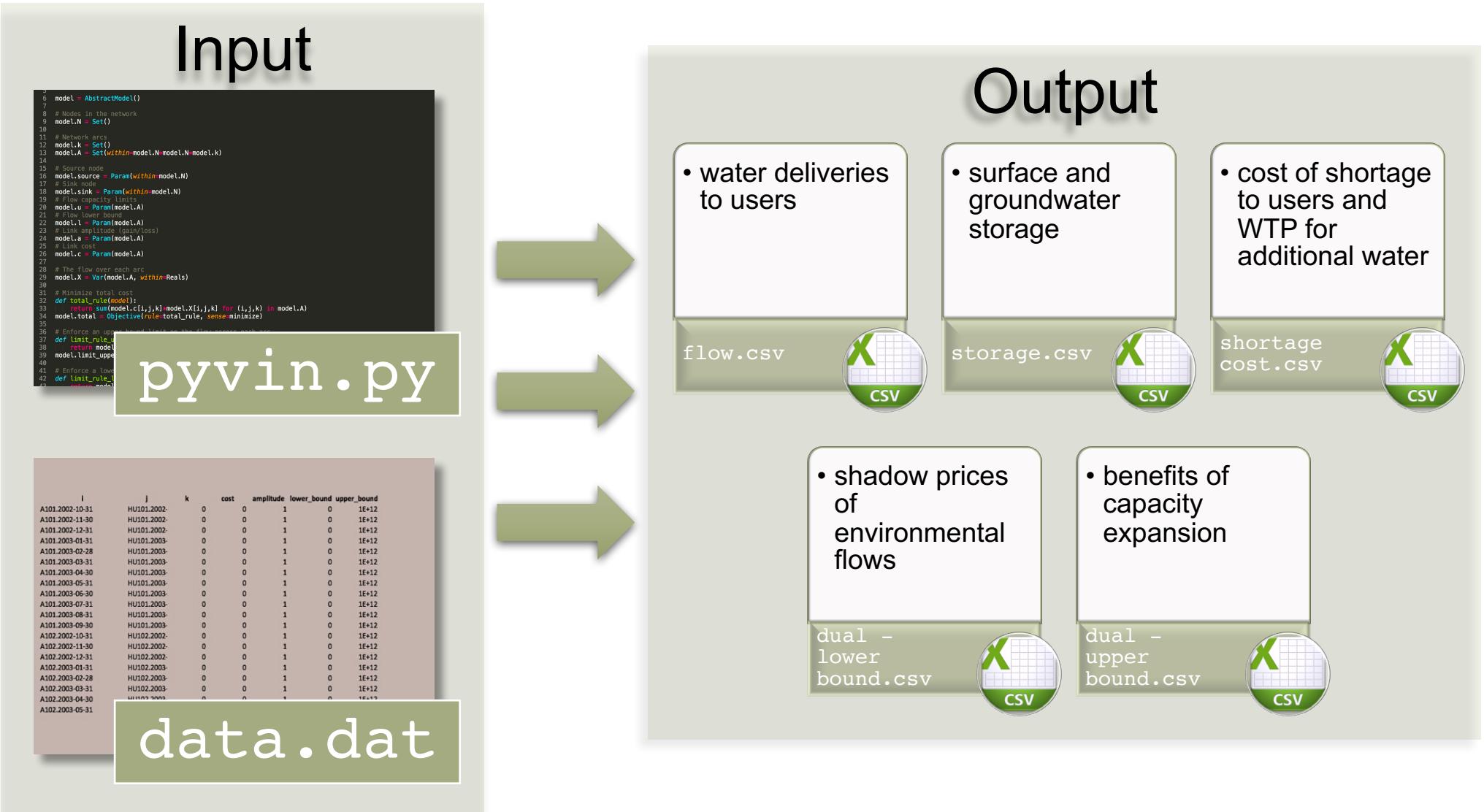
# The flow over each arc
model.X = Var(model.A, within=Reals)

# 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)

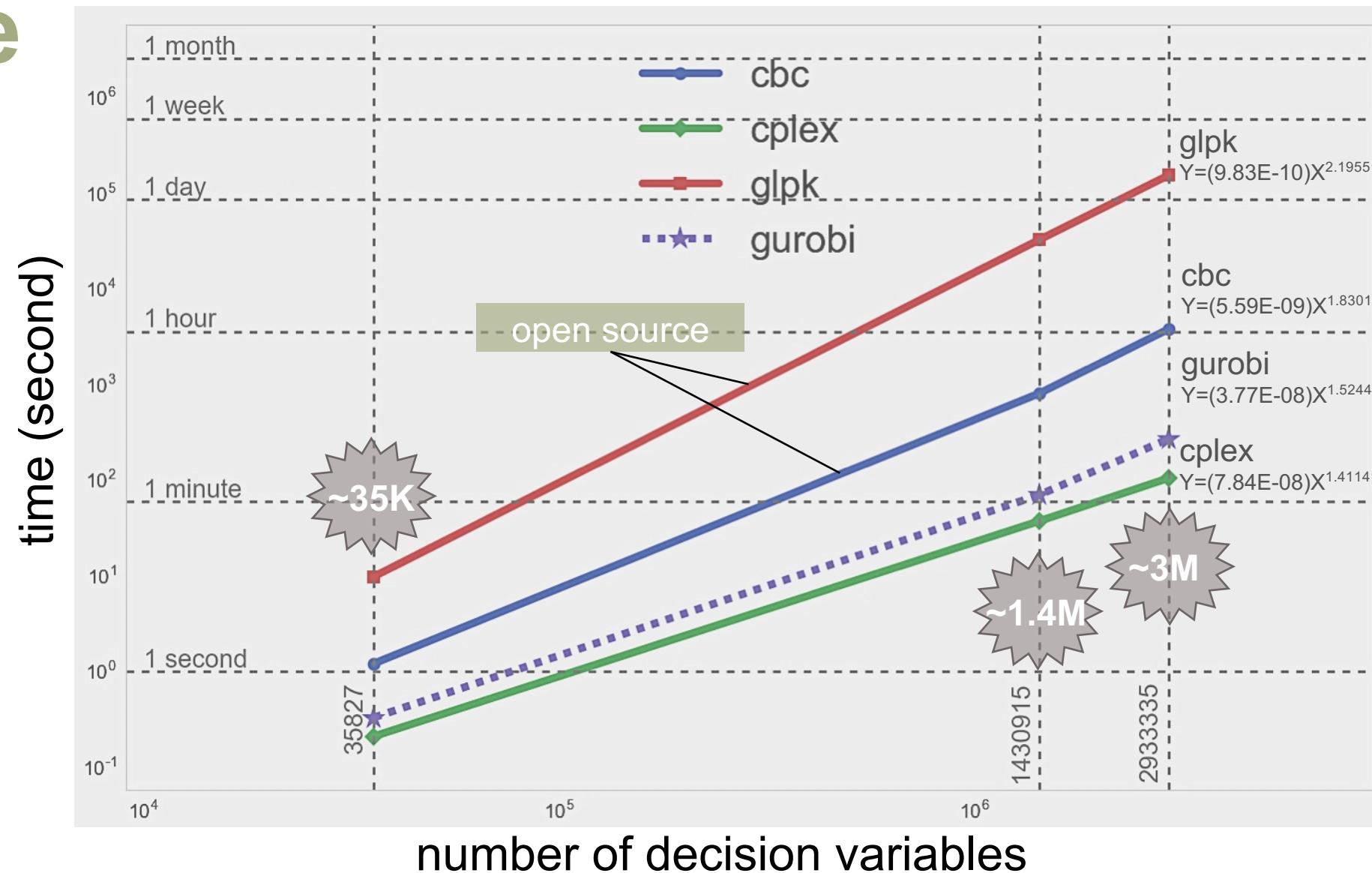
# Enforce a lower bound limit on the flow across each arc
def limit_rule_lower(model, i, j, k):
    return model.X[i,j,k] >= model.l[i,j,k]
model.limit_lower = Constraint(model.A, rule=limit_rule_lower)
```

PyVIN



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

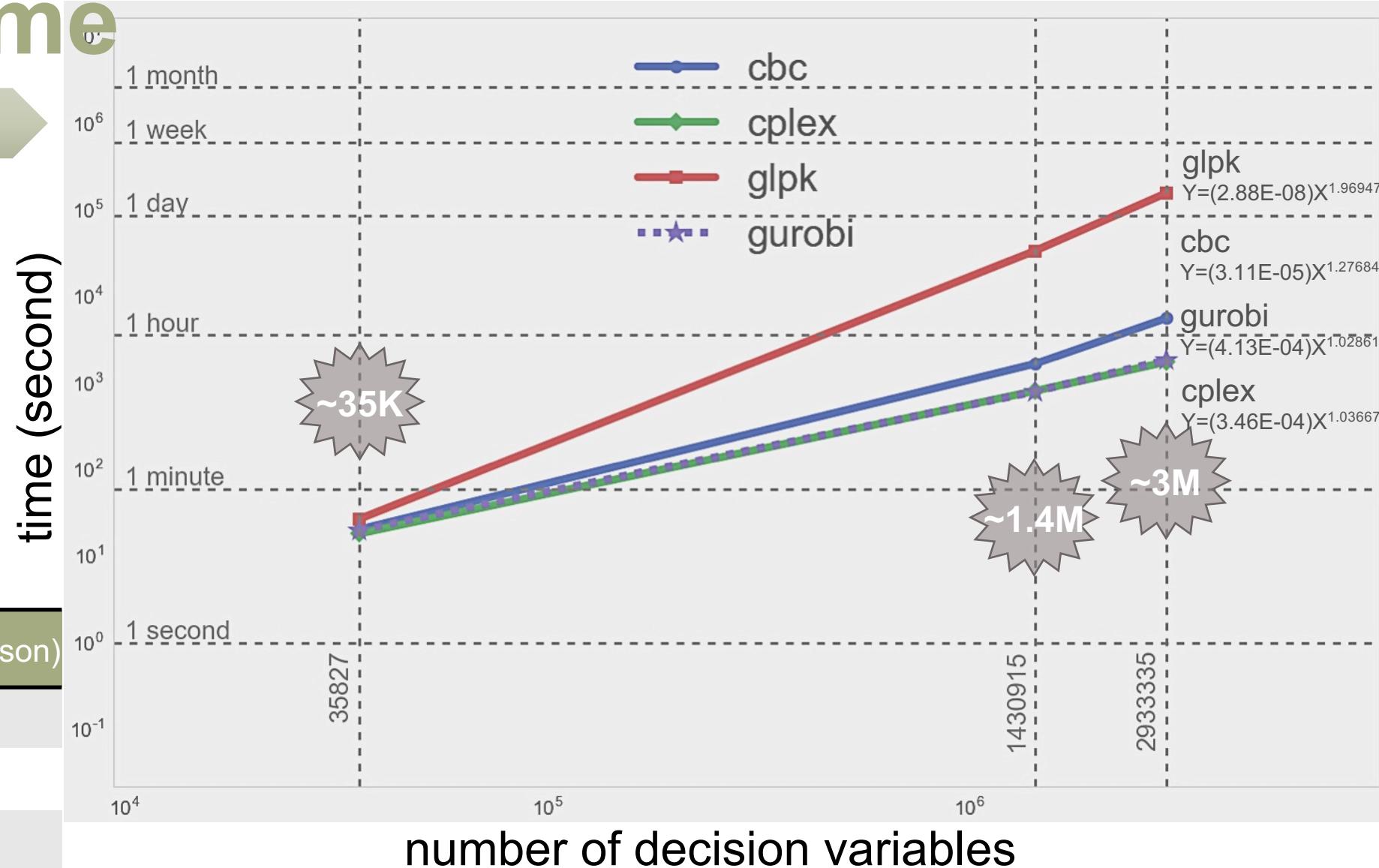


Total Runtime



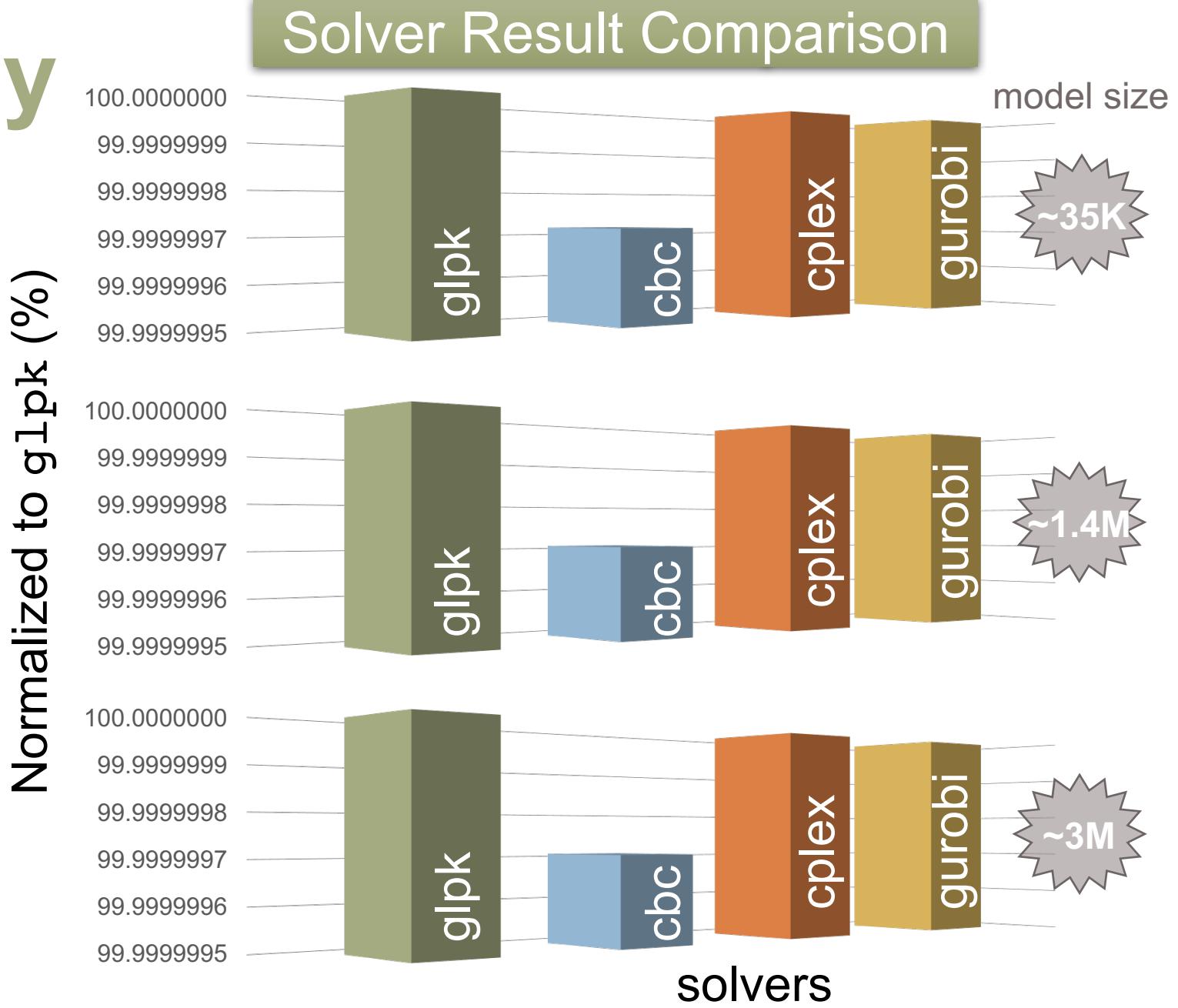
- Differences get bigger as model size increases
- cplex and gurobi have similar total runtime

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

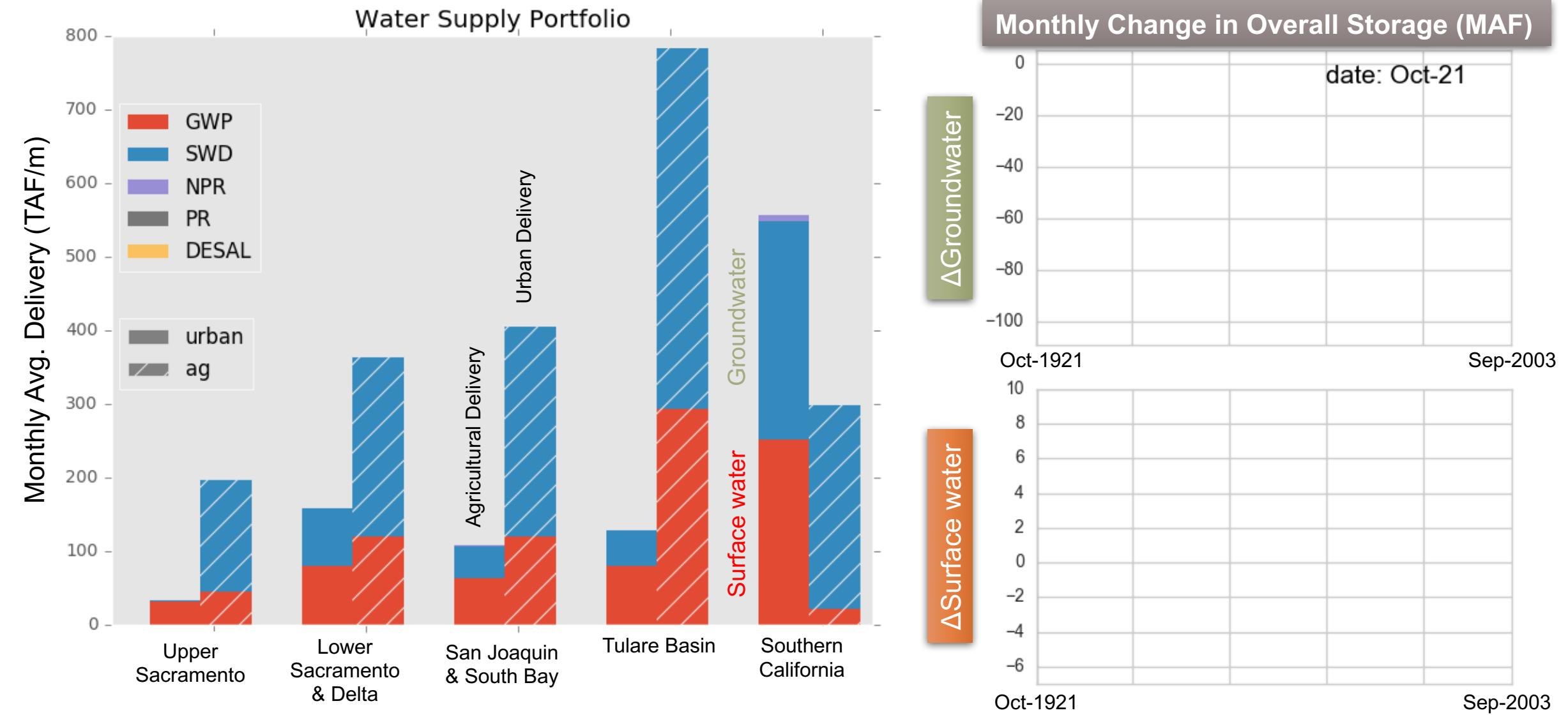


Solver Accuracy

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



Preliminary Results



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

Software Installation & Your 1st PyVIN run

Structure File: pyvin.py

```

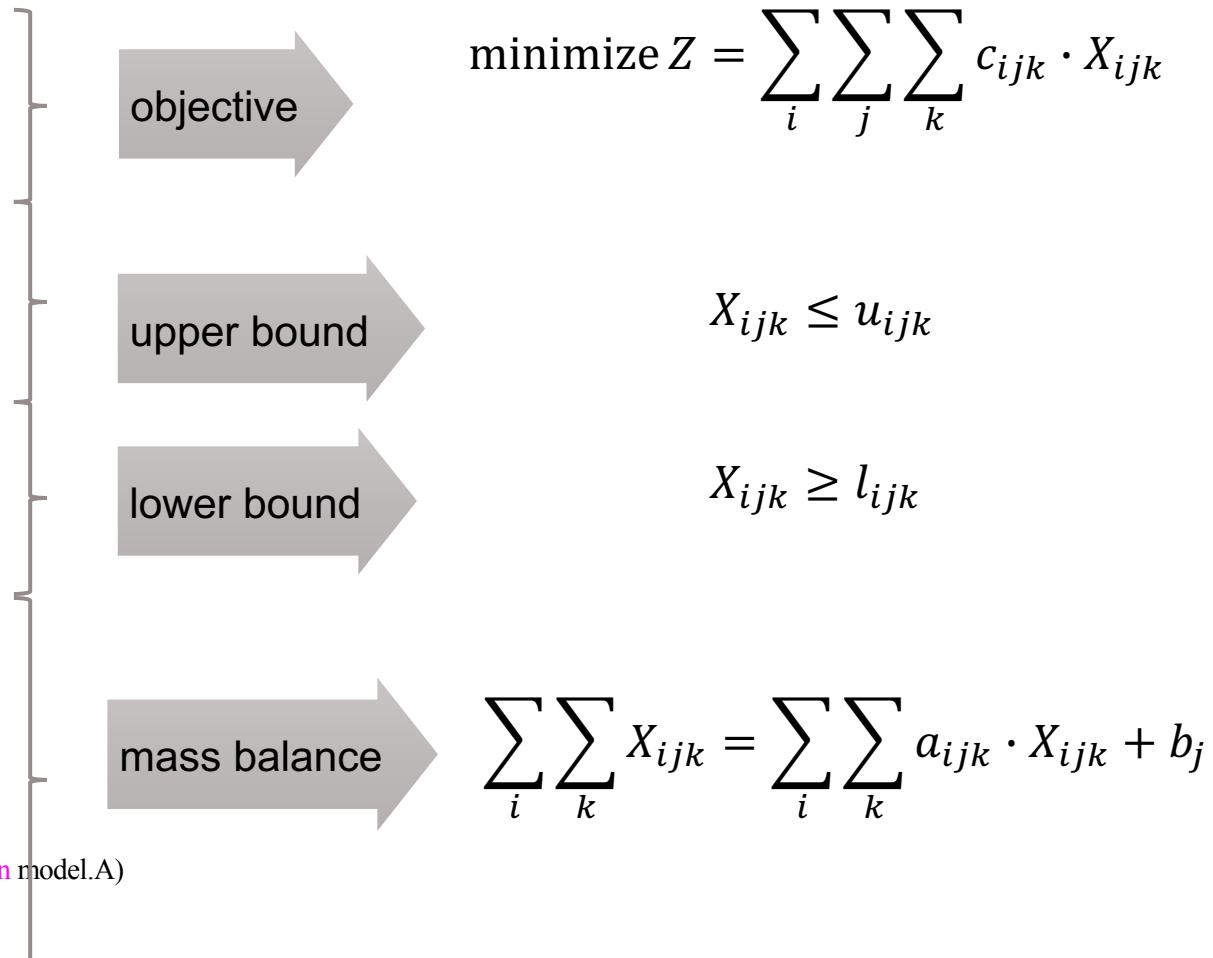
# Minimize total cost
def total_rule(model):
    return sum(model.c[i,j,k]*model.X[i,j,k] for (i,j,k) in model.A)
model.total = Objective(rule=total_rule, sense=minimize)

# 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)

# Enforce flow through each node (mass balance)
def flow_rule(model, node):
    if node in [value(model.source), value(model.sink)]:
        model.c = Param(model.A)
        # The flow over each arc
        return Constraint.Skip
    outflow = sum(model.X[i,j,k]/model.a[i,j,k] for i,j,k in model.A) inflow = sum(model.X[i,j,k] for i,j,k in model.A)
    return inflow == outflow
model.flow = Constraint(model.N, rule=flow_rule)

```



Data File: data.dat

```
set N :=  
INITIAL SR SHA.1983-10-31 SR SHA.1983-11-30  
FINAL  
INFLOW.1983-10-31  
INFLOW.1983-11-30;  
  
set k := 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15;  
  
param source := SOURCE; param sink := SINK;  
  
param: A: c a l u :=  
SOURCE INITIAL 0 0 1 0 10000000  
INITIAL SR SHA.1983-10-31 0 0 1 3686.84 3686.84  
SR SHA.1983-10-31 SR SHA.1983-11-30 0 -7.003 0.997  
SR SHA.1983-10-31 SR SHA.1983-11-30 1 -2.974 0.997  
SR SHA.1983-10-31 SR SHA.1983-11-30 2 -1.466 0.997  
SR SHA.1983-11-30 SR SHA.1983-12-31 0 -6.972 0.999  
SR SHA.1983-11-30 SR SHA.1983-12-31 1 -3.056 0.999  
SR SHA.1983-11-30 SR SHA.1983-12-31 2 -1.479 0.999
```

List of nodes

Number of piecewise pieces

Identify SOURCE and SINK (no mass balance calculation)

- Links with parameters
- TAB separated
- Order:
- i j k cost amplitude lower_bound upper_bound

Historical Case Example Run

- Go to folder “/CALVIN-PyVIN Shortcourse/Run Folder/base case/”
- Take a look at `pyvin.py` (right click, don’t run) and `data.dat`
- Open command line and change directory to run folder
`cd \path\to\folder`
- Run the model command:

```
pyomo solve --solver=glpk --solver-suffix=dual pyvin.py  
data.dat -- stream-solver --report-timing --json
```

Postprocessing Results

Running Postprocessors

- Copy all postprocessors to run folder (where `results.json` is generated)
- Run `postprocess.py`
- Run `aggregate_regions.py`
- Run `supply_portfolio_hatchedbarplot.py`
- Should you have any problem with running postprocessors, results are saved under "output" folder.