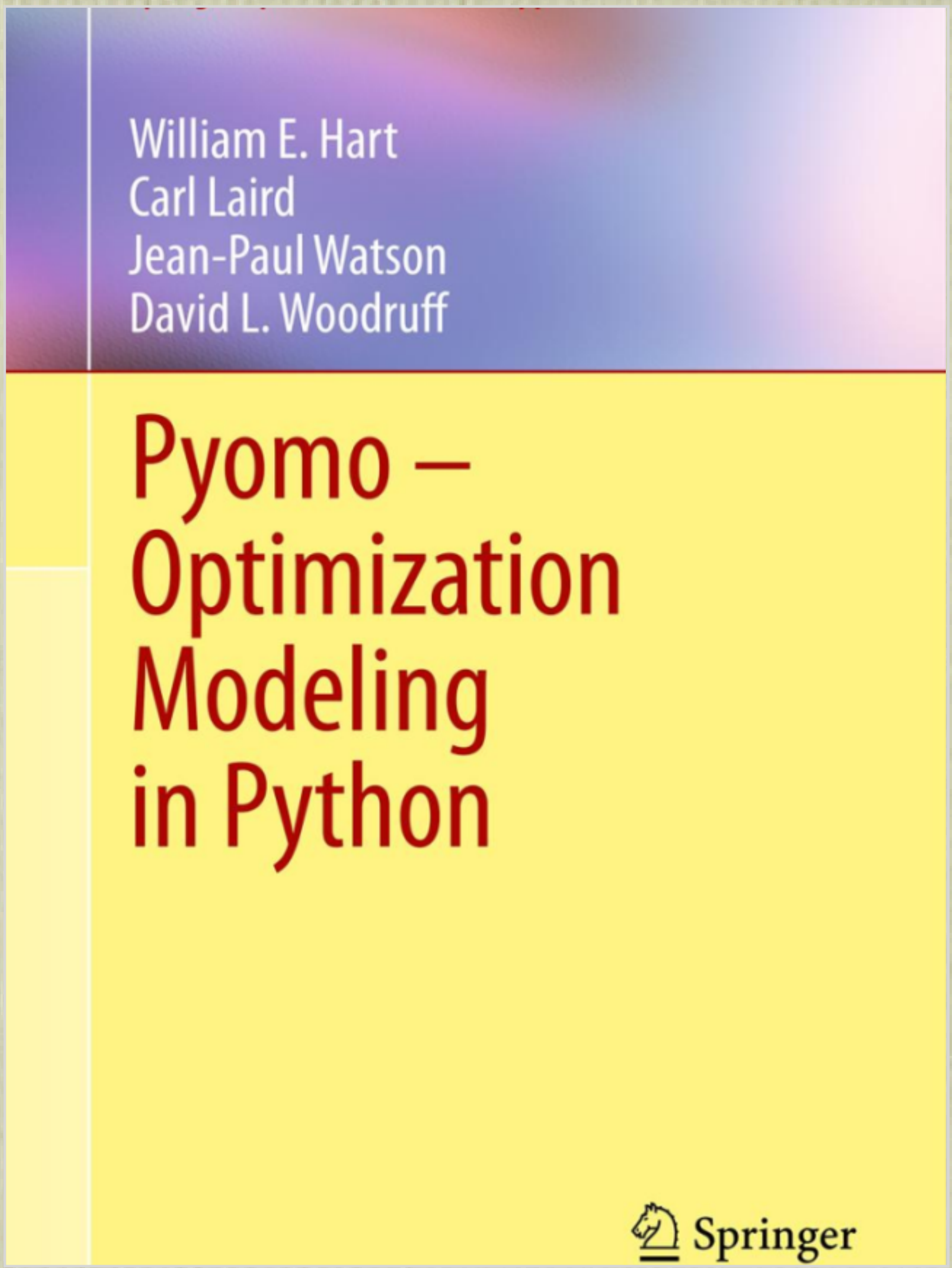


Carl D. Laird, Assistant Professor
Chemical Engineering, Texas A&M University

William E. Hart, Jean-Paul Watson, John D. Siirola
Sandia National Laboratories, Albuquerque, NM

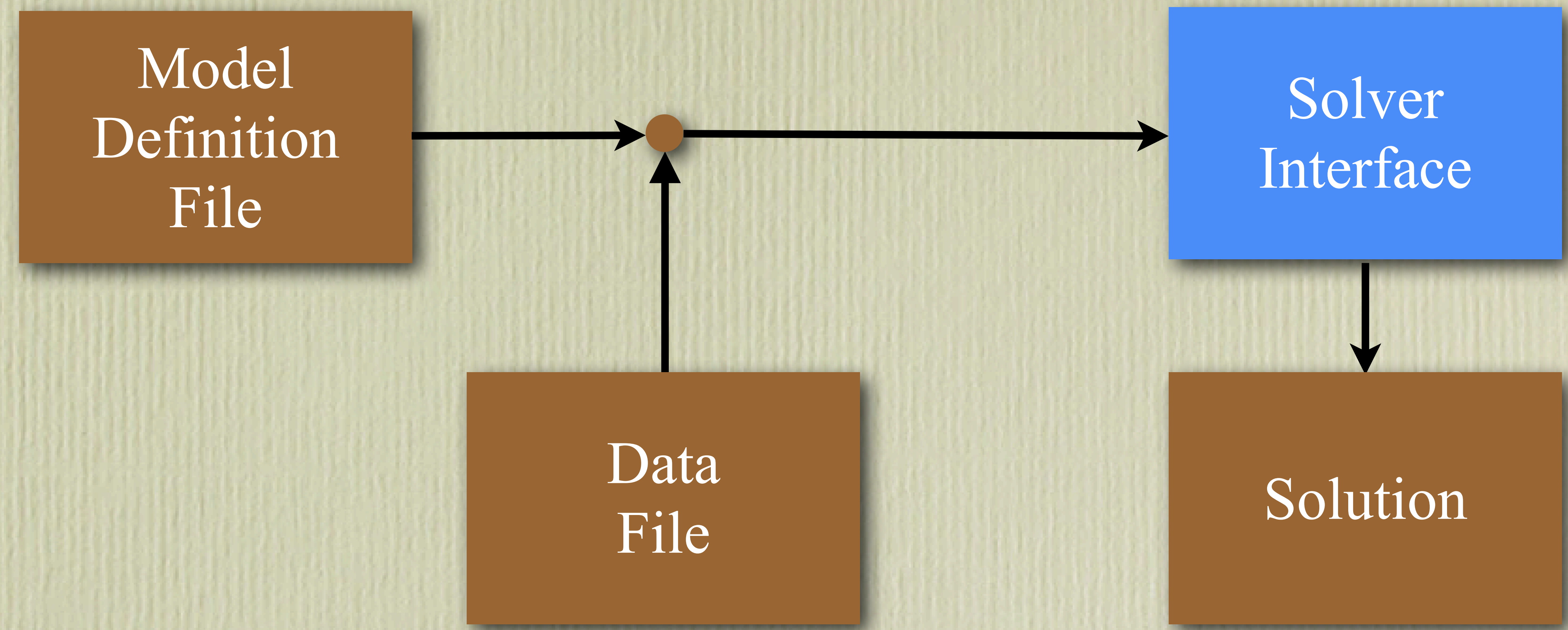
David L. Woodruff, Professor
Business Management, University of California, Davis

Artie McFerrin Department of
CHEMICAL ENGINEERING | TEXAS A&M ENGINEERING

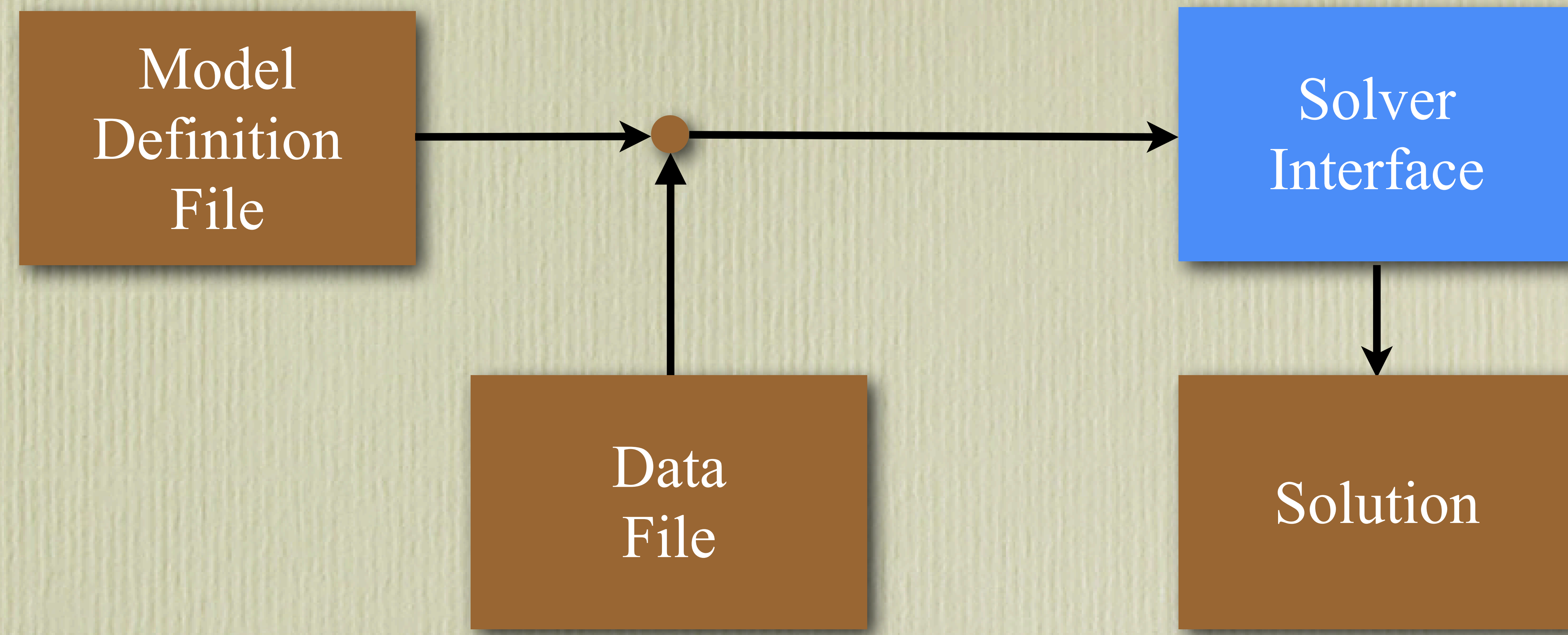
Pyomo - Python Optimization Modeling Objects

- Algebraic equation-based modeling language for optimization
  - e.g AMPL, GAMS, AIMMS
  - acausal, equation-based modeling
  - currently no support for differential equations
  - initially driven by large-scale MILP
- Designed by Math Programmers for Math Programmers
  - open-source, extensible alternative to existing tools
  - used to enable research and engineering solutions
- I work on algorithms and applications
  - I am a user of modeling languages, ... right?

Typical Algebraic Modeling Language



Typical Algebraic Modeling Language

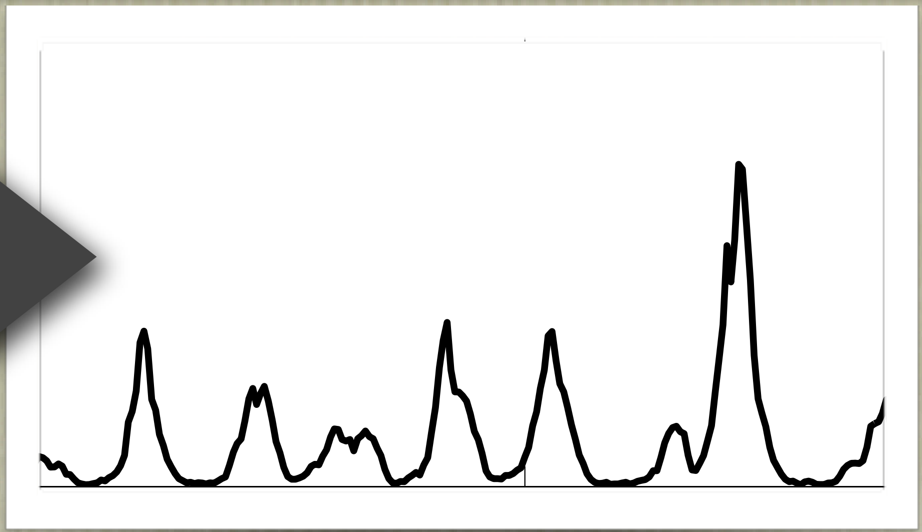


- Provide powerful, high-level problem specification
- Familiar math programming constructs (Sets, expressions)
- Very limited programming / scripting capability
  - model transformations? language extensions?
  - plotting? functions? numerical libraries?

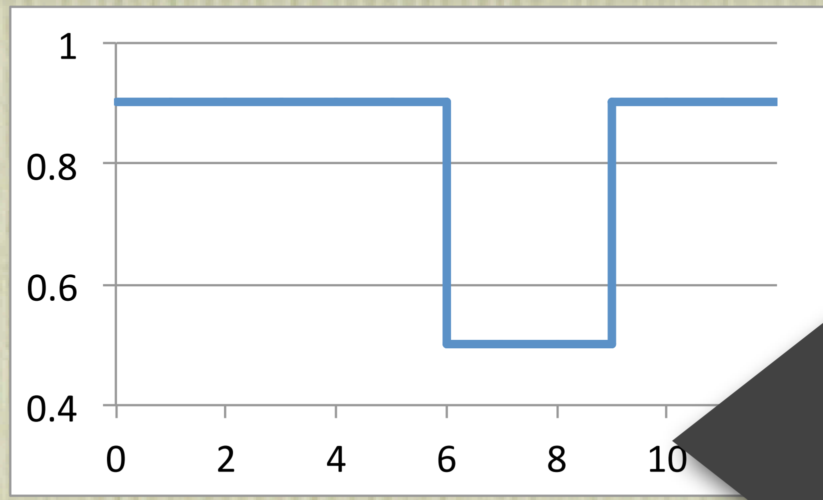
Seasonal Drivers in Infectious Disease Spread

Seasonal Drivers?

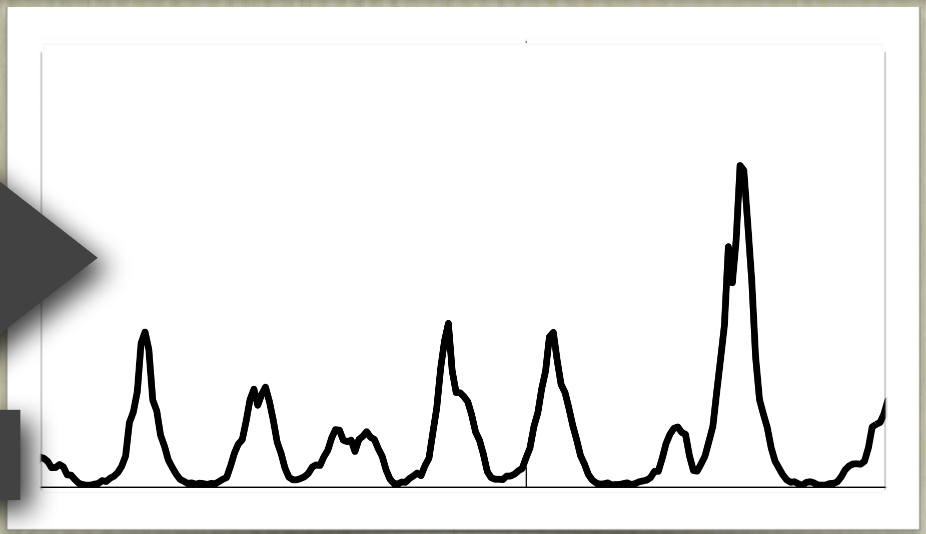
Nonlinear Discrete-Time Disease Model



Seasonal Drivers in Infectious Disease Spread



Nonlinear Discrete-Time Disease Model

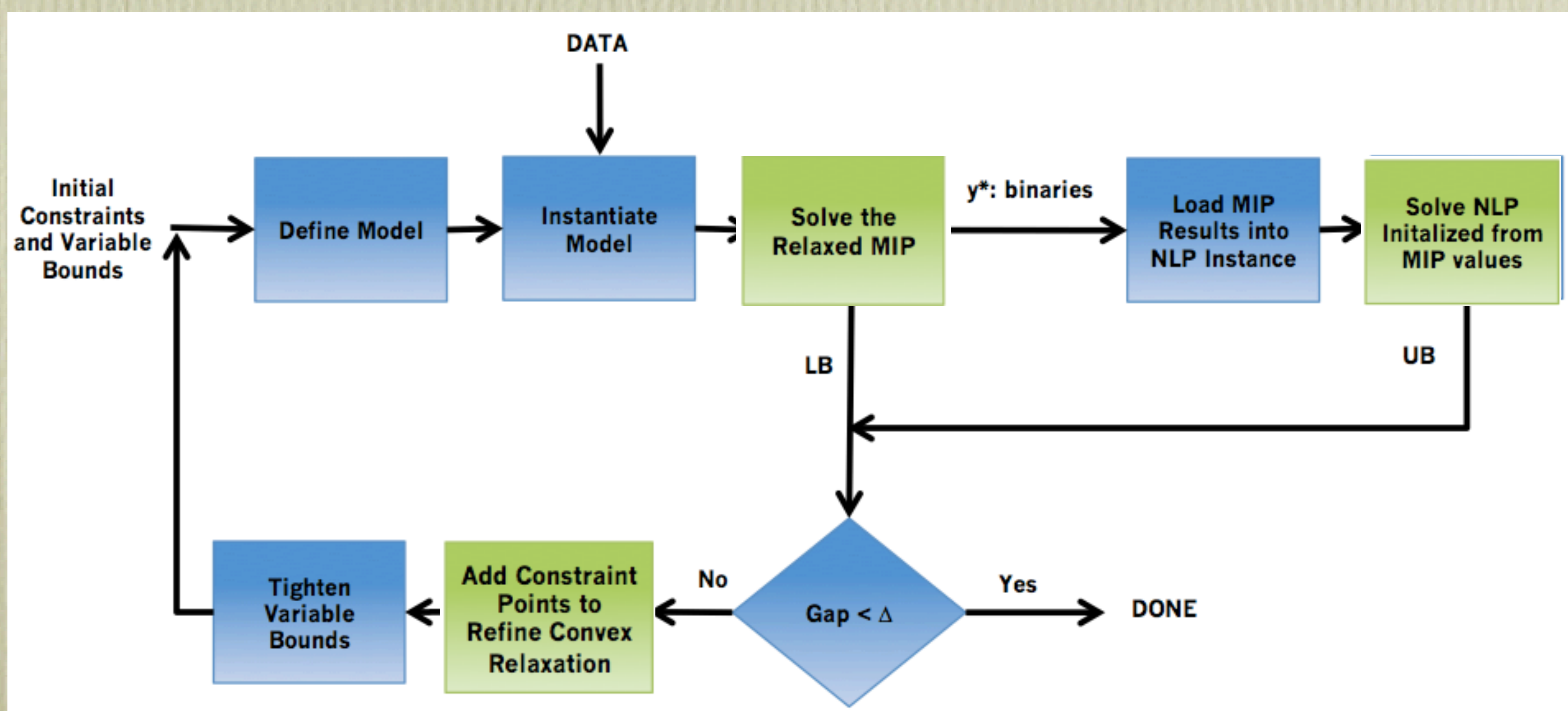


Seasonal Drivers in Infectious Disease Spread

Large Mixed Integer Non-Linear Programming Problem

Seasonal Drivers in Infectious Disease Spread

Large Mixed Integer Non-Linear Programming Problem



Parallel Decomposition in Interior-Point Methods

$$\min_x \; f(x) \quad \text{s.t.} \; c(x) = 0, \; x \geq 0$$

$$\min_x \; f(x) - \mu \cdot \sum_i ln(x_i) \quad \text{s.t.} \; c(x) = 0$$

$$\begin{aligned} \nabla f(x) + \nabla c(x)^T \cdot \lambda - z &= 0 \\ c(x) &= 0 \\ X \cdot z &= \mu e \\ (x > 0, z &> 0) \end{aligned}$$

$$z = \mu X^{-1} e$$

$$\begin{aligned} \nabla f(x) + \nabla c(x)^T \lambda - \mu X^{-1} e &= 0 \\ c(x) &= 0 \\ (x &> 0) \end{aligned}$$

$$\begin{bmatrix} W_k + \Sigma_k + \delta_w I & \nabla c(x_k) \\ \nabla c(x_k)^T & -\delta_c I \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta \lambda \end{pmatrix} = - \begin{bmatrix} \nabla \varphi_\mu(x_k) + \nabla c(x_k)^T \lambda_k \\ c(x_k) \end{bmatrix}$$

$$\left(W_k = \nabla^2_{xx} \mathcal{L} = \nabla^2_{xx} f(x_k) + \nabla^2_{xx} c(x_k) \lambda \right)$$

$$(\delta_w, \delta_c \geq 0) \; \left(\Sigma_k = Z_k X_k^{-1} \right)$$

Parallel Decomposition in Interior-Point Methods

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & x \geq 0 \end{aligned}$$

$$\begin{aligned} \min_{x} \quad & f(x) - \mu \cdot \sum_i ln(x_i) \\ \text{s.t.} \quad & c(x) = 0 \end{aligned}$$

$$\begin{aligned} \nabla f(x) + \nabla c(x)^T \lambda - \mu X^{-1} e &= 0 \\ c(x) &= 0 \\ (x &> 0) \end{aligned}$$

$$z = \mu X^{-1} e$$

$$\begin{aligned} \nabla f(x) + \nabla c(x)^T \cdot \lambda - z &= 0 \\ c(x) &= 0 \\ X \cdot z &= \mu e \\ (x > 0, z &> 0) \end{aligned}$$

$$\begin{bmatrix} W_k + \Sigma_k + \delta_w I & \nabla c(x_k) \\ \nabla c(x_k)^T & -\delta_c I \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta \lambda \end{pmatrix} = - \begin{bmatrix} \nabla \varphi_\mu(x_k) + \nabla c(x_k)^T \lambda_k \\ c(x_k) \end{bmatrix}$$

$$\left(W_k = \nabla^2_{xx} \mathcal{L} = \nabla^2_{xx} f(x_k) + \nabla^2_{xx} c(x_k) \lambda\right)$$

$$(\delta_w, \delta_c \geq 0) \ \left(\Sigma_k = Z_k X_k^{-1}\right)$$

Parallel Decomposition in Interior-Point Methods

$$\min_{x_q, y} \sum_{q \in \mathcal{Q}} f_q(x_q)$$

$$\text{s.t.} \quad c_q(x_q) = 0$$

$$x_q^L \le x_q \le x_q^U \qquad \forall \; q \in \mathcal{Q}$$

$$L_q^x x_q - L_q^y y = 0,$$

- Nonlinear Stochastic Optimization
- Large-scale Parameter Estimation
- Design Under Uncertainty
- Spatially Decomposable Problems
- Very large-scale NLP Problems
  - Highly Structured

Parallel Decomposition in Interior-Point Methods

$$\min_{x_q, y} \sum_{q \in \mathcal{Q}} f_q(x_q)$$

$$\text{s.t.} \quad c_q(x_q) = 0$$

$$x_q^L \le x_q \le x_q^U \qquad \forall \;\; q \in \mathcal{Q}$$

$$L_q^x x_q - L_q^y y = 0,$$

- Nonlinear Stochastic Optimization
- Large-scale Parameter Estimation
- Design Under Uncertainty
- Spatially Decomposable Problems
- Very large-scale NLP Problems
  - Highly Structured

$$\begin{bmatrix} K_1 & & & & A_1 \\ & K_2 & & & A_2 \\ & & \ddots & & \vdots \\ & & & K_{n_q} & A_{n_Q} \\ A_1^T & A_2^T & \cdots & A_{n_Q}^T & D_y \end{bmatrix} \begin{bmatrix} \Delta_1 \\ \Delta_2 \\ \vdots \\ \Delta_{n_q} \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{n_q} \\ r_y \end{bmatrix}$$

Parallel Decomposition in Interior-Point Methods

$$\min_{x_q, y} \sum_{q \in \mathcal{Q}} f_q(x_q)$$

$$\begin{aligned} \text{s.t.} \quad & c_q(x_q) = 0 \\ & x_q^L \le x_q \le x_q^U \qquad \forall \; q \in \mathcal{Q} \\ & L_q^x x_q - L_q^y y = 0, \end{aligned}$$

$$\begin{bmatrix} K_1 & & & & A_1 \\ & K_2 & & & A_2 \\ & & \ddots & & \vdots \\ & & & K_{n_q} & A_{n_Q} \\ A_1^T & A_2^T & \cdots & A_{n_Q}^T & D_y \end{bmatrix} \begin{bmatrix} \Delta_1 \\ \Delta_2 \\ \vdots \\ \Delta_{n_q} \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{n_q} \\ r_y \end{bmatrix}$$

Parallel solution of structured linear system

Parallel Decomposition in Interior-Point Methods

$$\min_{x_q, y} \sum_{q \in \mathcal{Q}} f_q(x_q)$$

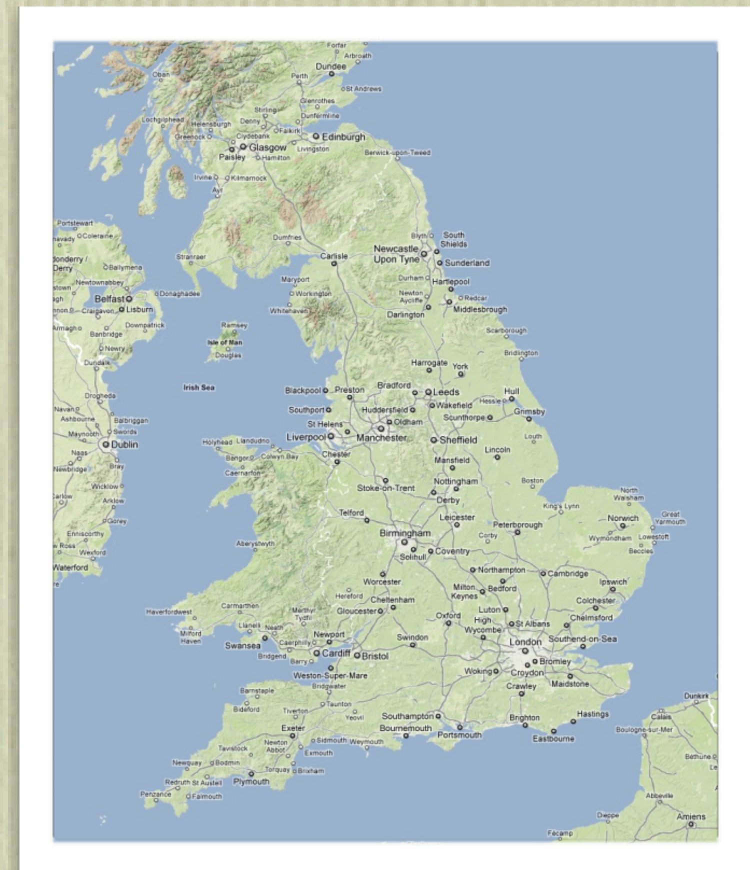
$$\begin{aligned} \text{s.t.} \quad & c_q(x_q) = 0 \\ & x_q^L \le x_q \le x_q^U \qquad \forall \; q \in \mathcal{Q} \\ & L_q^x x_q - L_q^y y = 0, \end{aligned}$$

Parallel construction/evaluation of equations, J, H

Parallel solution of structured linear system

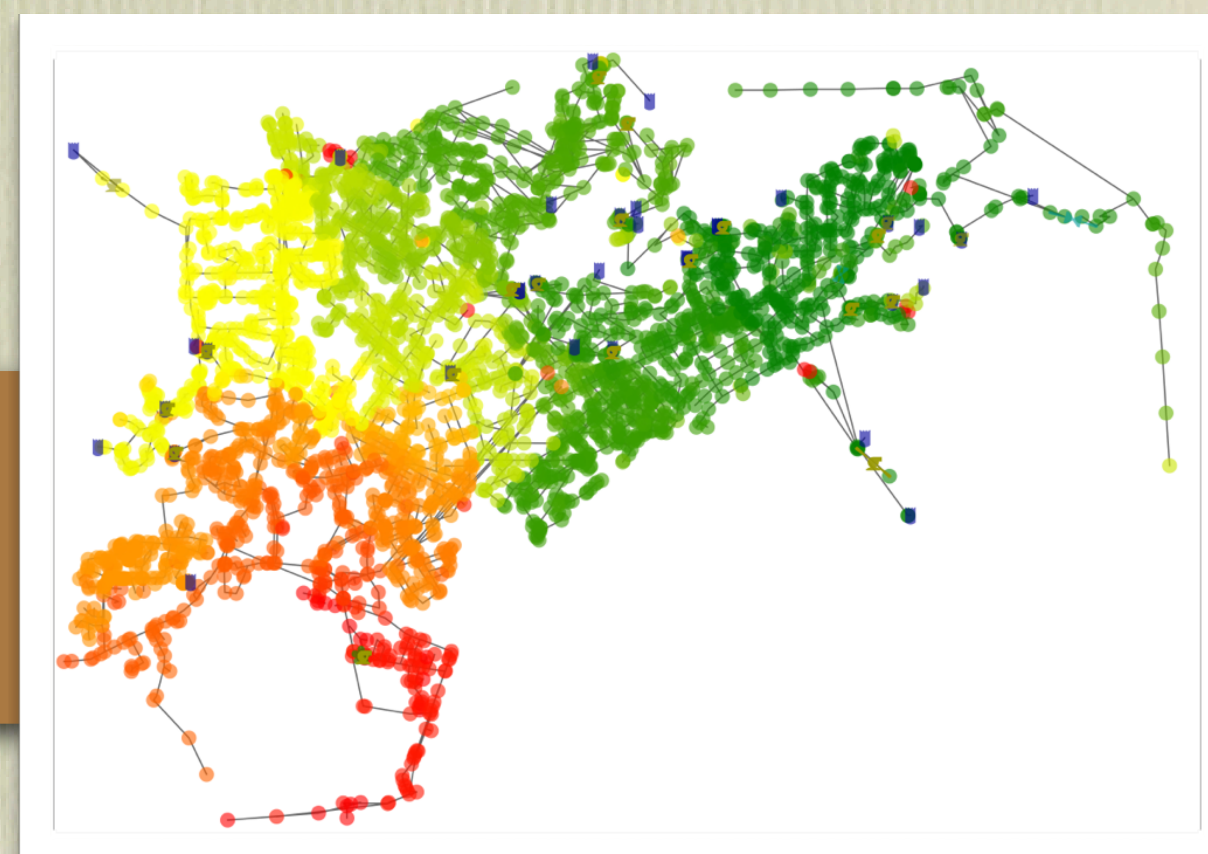
$$\begin{bmatrix} K_1 & & & & A_1 \\ & K_2 & & & A_2 \\ & & \ddots & & \vdots \\ & & & K_{n_q} & A_{n_Q} \\ A_1^T & A_2^T & \cdots & A_{n_Q}^T & D_y \end{bmatrix} \begin{bmatrix} \Delta_1 \\ \Delta_2 \\ \vdots \\ \Delta_{n_q} \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{n_q} \\ r_y \end{bmatrix}$$

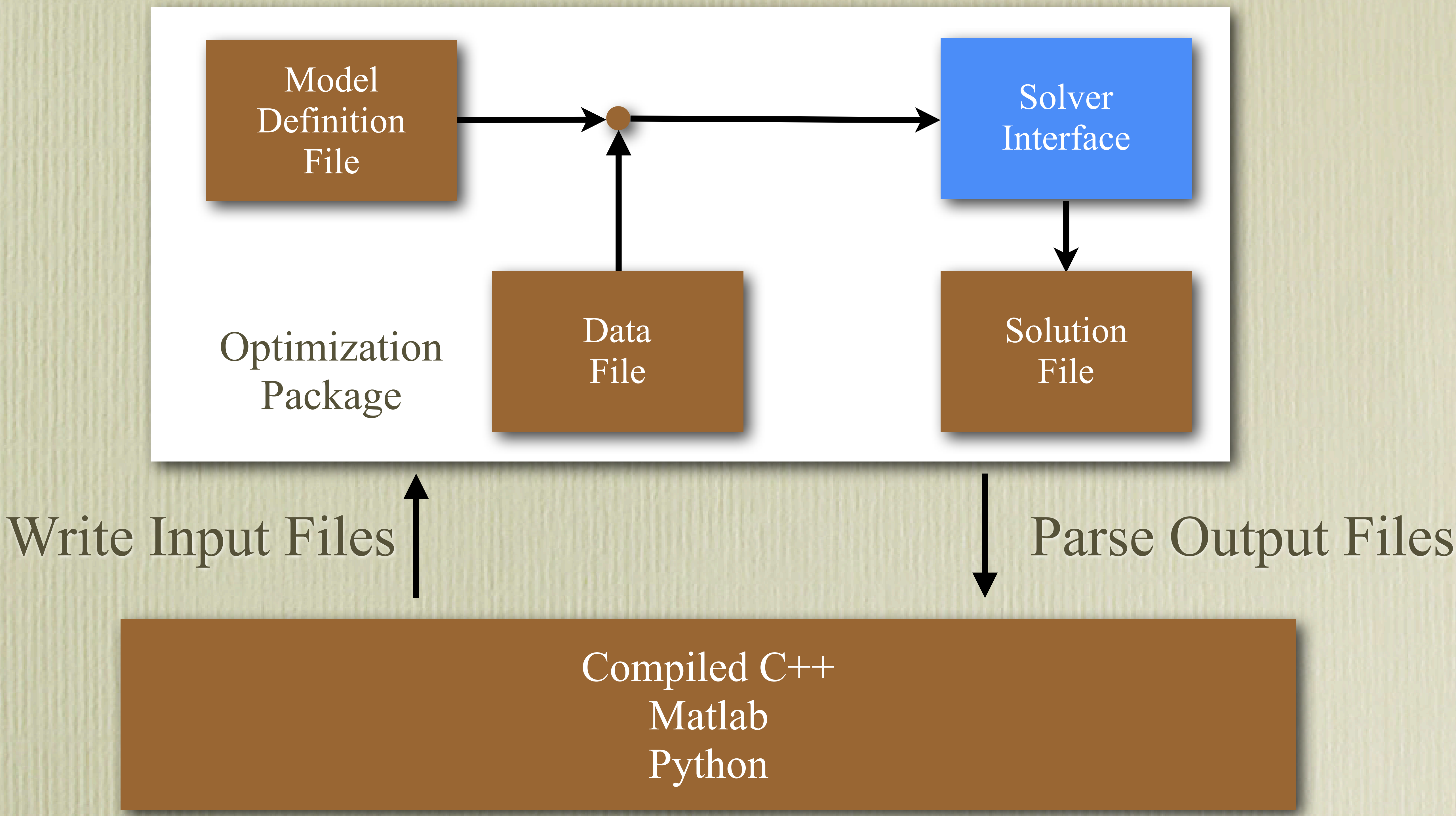
Other Examples of Applications



Parallel Parameter Estimation for Spatial Transportation Affecting Disease Spread

Optimal Response to Water Contamination Events





Fragile tool chain

Two Choices

1. Design new language
   - modeling, scripting syntax
   - compiler tools

2. Use programming language
   - develop components in another language
   - import types/functionality

Two Choices

1. Design new language
   - modeling, scripting syntax
   - compiler tools

2. Use programming language
   - develop components in another language
   - import types/functionality

- Selected to develop in Python (Choice 2)
  - tired of writing parsers
  - not language experts
  - existing tools are not actively updated
  - not responsible for full language functionality and packages
  - want full-featured language and user-extensibility (for “free”)

Requirements

- Powerful
  - full support for standard math programming constructs (LP, MILP, NLP, MINLP, ...)
  - full-featured programming environment (model interrogation, scripting, functions, classes, standard & numerical libraries)
  - extensive solver integration - "out-of-the-box"
- Open
  - licensed under BSD (i.e. really open-source)
  - reduce barriers to adoption, ease of collaboration
  - transparency
- Flexible
  - extensible by users, contributors, not only by us
  - portable (Windows, Linux, OS X)
- Easy
  - language constructs familiar to math programmers - Abstract Models
  - scripting / programming capability well-defined
  - substantial documentation

Why Python?

- License
  - open-source
- Language Features
  - familiar, lean syntax, rich set of existing data types, object-oriented, exceptions, dynamic loading, ...
- Support and stability
  - highly stable, well-supported
- Documentation
  - extensive online documentation, several books
- Libraries
  - significant external libraries, numerical & scientific packages
- Portability
  - widely available on many platforms

Simple Modeling Example: Knapsack

$\mathcal{S}$	:	set of items (set)
v_i	:	value of item i (param)
w_i	:	weight of item i (param)
W_{max}	:	maximum weight (param)
x_i	:	binary indicator (var)

$$\max \sum_{i \in \mathcal{S}} v_i \cdot x_i$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{S}} w_i \cdot x_i \leq W_{max}$$

$$x_i \in \{0, 1\} \quad \forall\, i \in \mathcal{S}$$

$\mathcal{S}$: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator

$$\max \sum_{i \in \mathcal{S}} v_i \cdot x_i$$

$$\text{s.t.} \sum_{i \in \mathcal{S}} w_i \cdot x_i \leq W_m$$

$$x_i \in \{0, 1\}$$

```
from coopr.pyomo import *

model       = AbstractModel()
model.ITEMS = Set()
model.v     = Param( model.ITEMS, within=PositiveReals )
model.w     = Param( model.ITEMS, within=PositiveReals )
model.W_max = Param( within=PositiveReals )
model.x     = Var( model.ITEMS, within=Binary )

def value_rule(model):
    return sum( model.v[i]*model.x[i] for i in model.ITEMS )
model.value = Objective( sense=maximize )

def weight_rule(model):
   return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
        <= model.W_max
model.weight = Constraint( )
```

Knapsack Problem: Abstract Model

$\mathcal{S}$: set of items

v_i: value of items

w_i: weight of items

W_m: maximum weight

x_i: binary indicator

$$\max \sum_{i \in \mathcal{S}} v_i \cdot x_i$$

$$\text{s.t.} \sum_{i \in \mathcal{S}} w_i \cdot x_i \leq W_m$$

$$x_i \in \{0, 1\}$$

```
from coopr.pyomo import *

model       = AbstractModel()
model.ITEMS = Set()
model.v     = Param( model.ITEMS, within=PositiveReals )
model.w     = Param( model.ITEMS, within=PositiveReals )
model.W_max = Param( within=PositiveReals )
model.x     = Var( model.ITEMS, within=Binary )

def value_rule(model):
    return sum( model.v[i]*model.x[i] for i in model.ITEMS )
model.value = Objective( sense=maximize )

def weight_rule(model):
   return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
        <= model.W_max
model.weight = Constraint( )
```

Knapsack Problem: Abstract Model

$\mathcal{S}$: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator

$$\max \sum_{i \in \mathcal{S}} v_i \cdot x_i$$

$$\text{s.t.} \sum_{i \in \mathcal{S}} w_i \cdot x_i \leq W_m$$

$$x_i \in \{0, 1\}$$

```
from coopr.pyomo import *

model       = AbstractModel()
model.ITEMS = Set()
model.v     = Param( model.ITEMS, within=PositiveReals )
model.w     = Param( model.ITEMS, within=PositiveReals )
model.W_max = Param( within=PositiveReals )
model.x     = Var( model.ITEMS, within=Binary )

def value_rule(model):
    return sum( model.v[i]*model.x[i] for i in model.ITEMS )
model.value = Objective( sense=maximize )

def weight_rule(model):
   return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
        <= model.W_max
model.weight = Constraint( )
```

Knapsack Problem: Abstract Model

$\mathcal{S}$: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator

$$\max \sum_{i \in \mathcal{S}} v_i \cdot x_i$$

$$\text{s.t.} \sum_{i \in \mathcal{S}} w_i \cdot x_i \leq W_m$$

$$x_i \in \{0, 1\}$$

```
from coopr.pyomo import *

model       = AbstractModel()
model.ITEMS = Set()
model.v     = Param( model.ITEMS, within=PositiveReals )
model.w     = Param( model.ITEMS, within=PositiveReals )
model.W_max = Param( within=PositiveReals )
model.x     = Var( model.ITEMS, within=Binary )

def value_rule(model):
    return sum( model.v[i]*model.x[i] for i in model.ITEMS )
model.value = Objective( sense=maximize )

def weight_rule(model):
   return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
        <= model.W_max
model.weight = Constraint( )
```

Knapsack Problem: Abstract Model

$\mathcal{S}$: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator

$$\max \sum_{i \in \mathcal{S}} v_i \cdot x_i$$

$$\text{s.t.} \sum_{i \in \mathcal{S}} w_i \cdot x_i \leq W_m$$

$$x_i \in \{0, 1\}$$

```
from coopr.pyomo import *

model         = AbstractModel()
model.ITEMS = Set()
model.v       = Param( model.ITEMS, within=PositiveReals )
model.w       = Param( model.ITEMS, within=PositiveReals )
model.W_max = Param( within=PositiveReals )
model.x       = Var( model.ITEMS, within=Binary )

def value_rule(model):
    return sum( model.v[i]*model.x[i] for i in model.ITEMS )
model.value = Objective( sense=maximize )

def weight_rule(model):
   return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
        <= model.W_max
model.weight = Constraint( )
```

Knapsack Problem: Abstract Model

$\mathcal{S}$: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator

$$\max \sum_{i \in \mathcal{S}} v_i \cdot x_i$$

$$\text{s.t.} \sum_{i \in \mathcal{S}} w_i \cdot x_i \leq W_m$$

$$x_i \in \{0, 1\}$$

```
from coopr.pyomo import *

model       = AbstractModel()
model.ITEMS = Set()
model.v     = Param( model.ITEMS, within=PositiveReals )
model.w     = Param( model.ITEMS, within=PositiveReals )
model.W_max = Param( within=PositiveReals )
model.x     = Var( model.ITEMS, within=Binary )

def value_rule(model):
    return sum( model.v[i]*model.x[i] for i in model.ITEMS )
model.value = Objective( sense=maximize )

def weight_rule(model):
   return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
        <= model.W_max
model.weight = Constraint( )
```

Knapsack Problem: Abstract Model

$\mathcal{S}$: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator

$$\max \sum_{i \in \mathcal{S}} v_i \cdot x_i$$

$$\text{s.t.} \sum_{i \in \mathcal{S}} w_i \cdot x_i \leq W_m$$

$$x_i \in \{0, 1\}$$

```
from coopr.pyomo import *

model       = AbstractModel()
model.ITEMS = Set()
model.v     = Param( model.ITEMS, within=PositiveReals )
model.w     = Param( model.ITEMS, within=PositiveReals )
model.W_max = Param( within=PositiveReals )
model.x     = Var( model.ITEMS, within=Binary )

def value_rule(model):
    return sum( model.v[i]*model.x[i] for i in model.ITEMS )
model.value = Objective( sense=maximize )

def weight_rule(model):
   return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
        <= model.W_max
model.weight = Constraint( )
```

Knapsack Problem: Abstract Model

$\mathcal{S}$: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator

$$\max \sum_{i \in \mathcal{S}} v_i \cdot x_i$$

$$\text{s.t.} \sum_{i \in \mathcal{S}} w_i \cdot x_i \leq W_m$$

$$x_i \in \{0, 1\}$$

```
from coopr.pyomo import *

model       = AbstractModel()
model.ITEMS = Set()
model.v     = Param( model.ITEMS, within=PositiveReals )
model.w     = Param( model.ITEMS, within=PositiveReals )
model.W_max = Param( within=PositiveReals )
model.x     = Var( model.ITEMS, within=Binary )

def value_rule(model):
    return sum( model.v[i]*model.x[i] for i in model.ITEMS )
model.value = Objective( sense=maximize )

def weight_rule(model):
   return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
        <= model.W_max
model.weight = Constraint( )
```

Knapsack Problem: Abstract Model

$\mathcal{S}$: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator

$$\max \sum_{i \in \mathcal{S}} v_i \cdot x_i$$

$$\text{s.t.} \sum_{i \in \mathcal{S}} w_i \cdot x_i \leq W_m$$

$$x_i \in \{0, 1\}$$

```
from coopr.pyomo import *

model        = AbstractModel()
model.ITEMS = Set()
model.v      = Param( model.ITEMS, within=PositiveReals )
model.w      = Param( model.ITEMS, within=PositiveReals )
model.W_max = Param( within=PositiveReals )
model.x      = Var( model.ITEMS, within=Binary )

def value_rule(model):
    return sum( model.v[i]*model.x[i] for i in model.ITEMS )
model.value = Objective( sense=maximize )

def weight_rule(model):
   return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
        <= model.W_max
model.weight = Constraint( )
```

Knapsack Problem: Abstract Model

$\mathcal{S}$: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator

$$\max \sum_{i \in \mathcal{S}} v_i \cdot x_i$$

$$\text{s.t.} \sum_{i \in \mathcal{S}} w_i \cdot x_i \leq W_m$$

$$x_i \in \{0, 1\}$$

```
from coopr.pyomo import *

model       = AbstractModel()
model.ITEMS = Set()
model.v     = Param( model.ITEMS, within=PositiveReals )
model.w     = Param( model.ITEMS, within=PositiveReals )
model.W_max = Param( within=PositiveReals )
model.x     = Var( model.ITEMS, within=Binary )

def value_rule(model):
    return sum( model.v[i]*model.x[i] for i in model.ITEMS )
model.value = Objective( sense=maximize )

def weight_rule(model):
   return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
        <= model.W_max
model.weight = Constraint( )
```

Knapsack Problem: Abstract Model

Model is completely abstract - there is no data

$\mathcal{S}$: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator

$$\max \sum_{i \in \mathcal{S}} v_i \cdot x_i$$

$$\text{s.t.} \sum_{i \in \mathcal{S}} w_i \cdot x_i \leq W_m$$

$$x_i \in \{0, 1\}$$

```
from coopr.pyomo import *

model       = AbstractModel()
model.ITEMS = Set()
model.v     = Param( model.ITEMS, within=PositiveReals )
model.w     = Param( model.ITEMS, within=PositiveReals )
model.W_max = Param( within=PositiveReals )
model.x     = Var( model.ITEMS, within=Binary )

def value_rule(model):
    return sum( model.v[i]*model.x[i] for i in model.ITEMS )
model.value = Objective( sense=maximize )

def weight_rule(model):
   return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
        <= model.W_max
model.weight = Constraint( )
```

Knapsack Problem: Abstract Model

Knapsack Problem: Abstract Model

Model is completely abstract - there is no data

$\mathcal{S}$: set of items
v_i: value of items
w_i: weight of items
W_m: maximum weight
x_i: binary indicator

$$\max \sum_{i \in \mathcal{S}} v_i \cdot x_i$$

$$\text{s.t.} \sum_{i \in \mathcal{S}} w_i \cdot x_i \leq W_m$$

$$x_i \in \{0, 1\}$$

```
from coopr.pyomo import *

model       = AbstractModel()
model.ITEMS = Set()
model.v     = Param( model.ITEMS, within=PositiveReals )
model.w     = Param( model.ITEMS, within=PositiveReals )
model.W_max = Param( within=PositiveReals )
model.x     = Var( model.ITEMS, within=Binary )

def value_rule(model):
    return sum( model.v[i]*model.x[i] for i in model.ITEMS )
model.value = Objective( sense=maximize )

def weight_rule(model):
   return sum(model.w[i]*model.x[i] for i in model.ITEMS) \
        <= model.W_max
model.weight = Constraint( )
```

```
> pyomo --solver=glpk knapsack.py akesson_art.dat
```

```
from coopr.pyomo import *

v = {'hammer':8, 'wrench':3, 'screwdriver':6, 'towel':11}
w = {'hammer':5, 'wrench':7, 'screwdriver':4, 'towel':3}
W_max = 14

model       = ConcreteModel()
model.ITEMS = Set( initialize=v.keys() )
model.x     = Var( model.ITEMS, within=Binary )

model.value = Objective(
  expr = sum( v[i]*model.x[i] for i in model.ITEMS ),
  sense = maximize )

model.weight = Constraint(
  expr = sum( w[i]*model.x[i] for i in model.ITEMS ) <= W_max )
```

Knapsack Problem: Concrete Model

```
from coopr.pyomo import *

v = {'hammer':8, 'wrench':3, 'screwdriver':6, 'towel':11}
w = {'hammer':5, 'wrench':7, 'screwdriver':4, 'towel':3}
W_max = 14

model       = ConcreteModel()
model.ITEMS = Set( initialize=v.keys() )
model.x     = Var( model.ITEMS, within=Binary )

model.value = Objective(
  expr = sum( v[i]*model.x[i] for i in model.ITEMS ),
  sense = maximize )

model.weight = Constraint(
  expr = sum( w[i]*model.x[i] for i in model.ITEMS ) <= W_max )
```

Knapsack Problem: Concrete Model

```
from coopr.pyomo import *

v = {'hammer':8, 'wrench':3, 'screwdriver':6, 'towel':11}
w = {'hammer':5, 'wrench':7, 'screwdriver':4, 'towel':3}
W_max = 14

model       = ConcreteModel()
model.ITEMS = Set( initialize=v.keys() )
model.x     = Var( model.ITEMS, within=Binary )

model.value = Objective(
  expr = sum( v[i]*model.x[i] for i in model.ITEMS ),
  sense = maximize )

model.weight = Constraint(
  expr = sum( w[i]*model.x[i] for i in model.ITEMS ) <= W_max )
```

Knapsack Problem: Concrete Model

```
from coopr.pyomo import *

v = {'hammer':8, 'wrench':3, 'screwdriver':6, 'towel':11}
w = {'hammer':5, 'wrench':7, 'screwdriver':4, 'towel':3}
W_max = 14

model       = ConcreteModel()
model.ITEMS = Set( initialize=v.keys() )
model.x     = Var( model.ITEMS, within=Binary )

model.value = Objective(
  expr = sum( v[i]*model.x[i] for i in model.ITEMS ),
  sense = maximize )

model.weight = Constraint(
  expr = sum( w[i]*model.x[i] for i in model.ITEMS ) <= W_max )
```

Knapsack Problem: Concrete Model

```python
from coopr.pyomo import *

v = {'hammer':8, 'wrench':3, 'screwdriver':6, 'towel':11}
w = {'hammer':5, 'wrench':7, 'screwdriver':4, 'towel':3}
W_max = 14

model       = ConcreteModel()
model.ITEMS = Set( initialize=v.keys() )
model.x     = Var( model.ITEMS, within=Binary )

model.value = Objective(
  expr = sum( v[i]*model.x[i] for i in model.ITEMS ),
  sense = maximize )

model.weight = Constraint(
  expr = sum( w[i]*model.x[i] for i in model.ITEMS ) <= W_max )
```

Knapsack Problem: Concrete Model

```
from coopr.pyomo import *

v = {'hammer':8, 'wrench':3, 'screwdriver':6, 'towel':11}
w = {'hammer':5, 'wrench':7, 'screwdriver':4, 'towel':3}
W_max = 14

model       = ConcreteModel()
model.ITEMS = Set( initialize=v.keys() )
model.x     = Var( model.ITEMS, within=Binary )

model.value = Objective(
  expr = sum( v[i]*model.x[i] for i in model.ITEMS ),
  sense = maximize )

model.weight = Constraint(
  expr = sum( w[i]*model.x[i] for i in model.ITEMS ) <= W_max )
```

Scripting

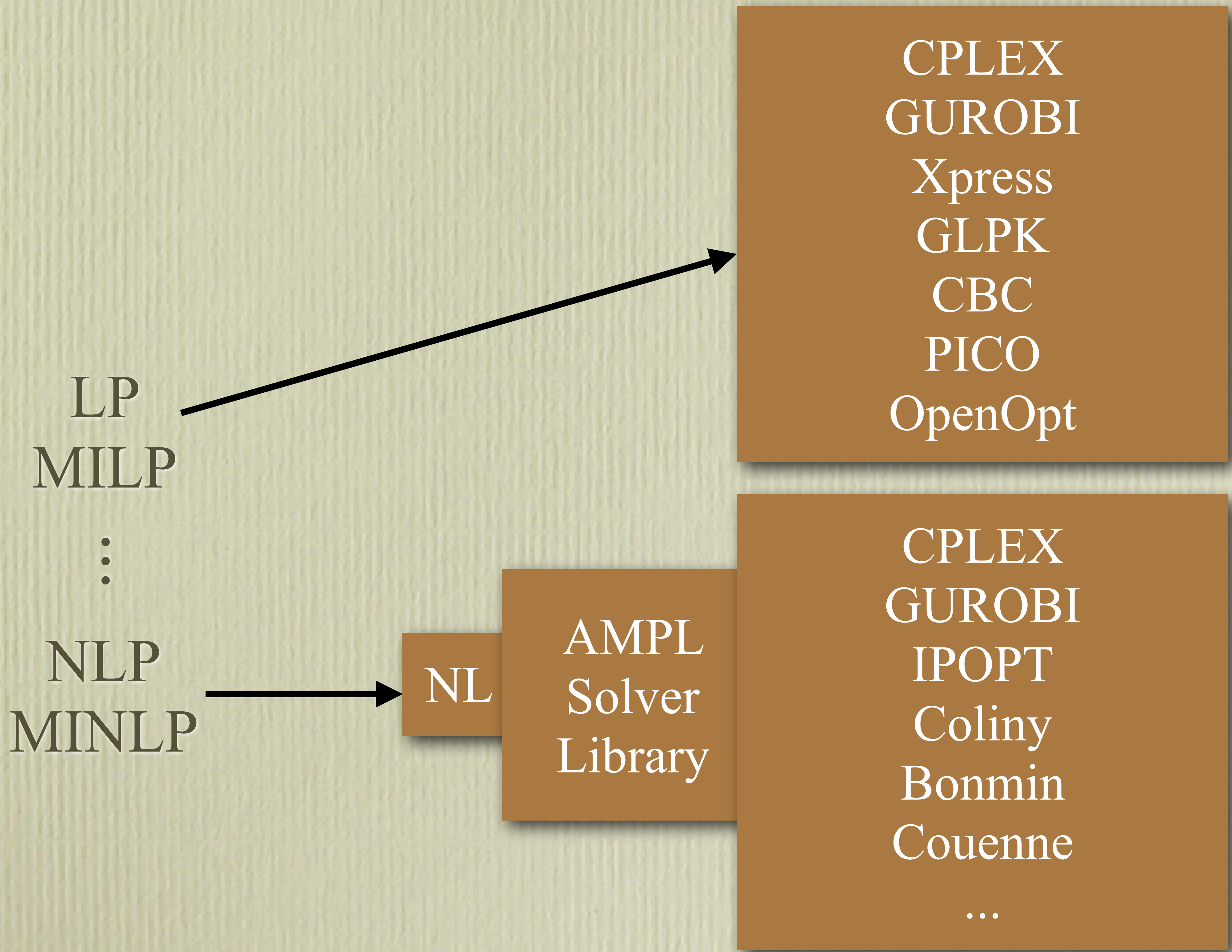
Knapsack Problem: Concrete Model

Solver Interfaces

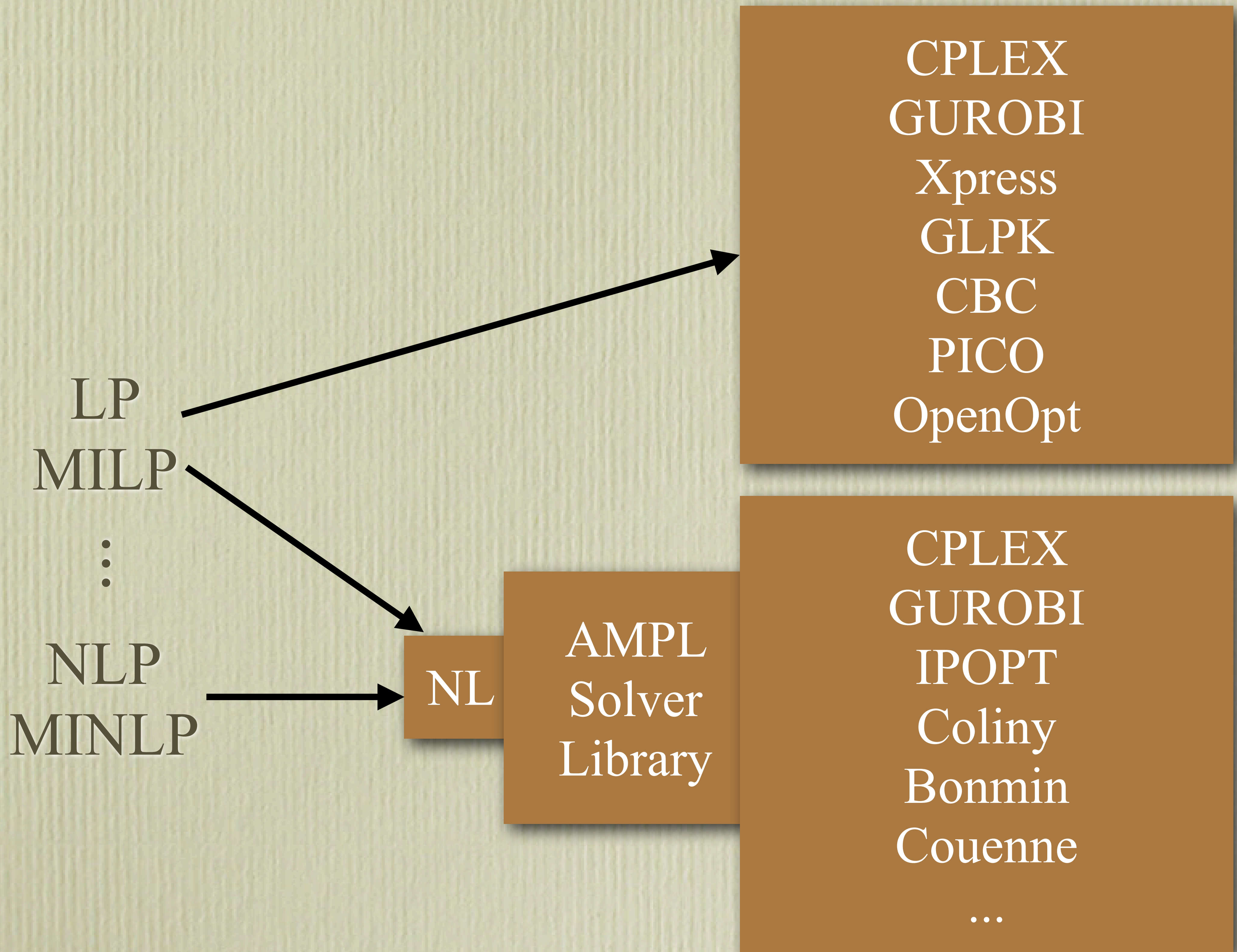
CPLEX
GUROBI
Xpress
GLPK
CBC
PICO
OpenOpt

LP
MILP
⋮
NLP
MINLP

Solver Interfaces



Solver Interfaces



Other Pyomo Features

- Advanced scripting capability
  - functions, OO, model interrogation & transformation
- Extensive set operations, tuples, multi-dimensional
- Load data from different sources
  - AMPL dat files, CSV files, Excel, databases
- Support for custom workflow with plugins
  - e.g. preprocess, create_modeldata, save_instance
- And more with extensions...

Summary

- Pyomo is an equation-based, algebraic modeling language for optimization
- Pyomo is an object-oriented framework for building optimization-based applications
- Based on Python
  - simple syntax for modeling
  - full-featured language
- Significant solver integration
- Open-source and Extensible
  - PySP: Stochastic Programming Framework
  - PH: Progressive Hedging Framework
  - Generalized Disjunctive Programming Capability
  - Blocks - Connectors
  - Piecewise-linear Constructs

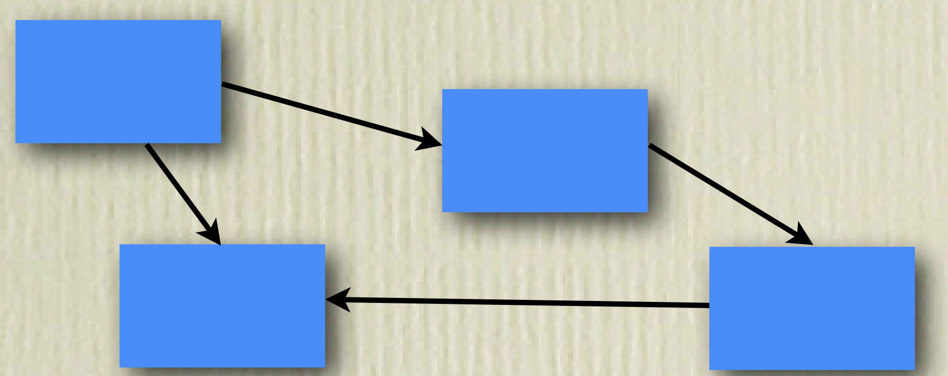
Some Closing Comments

- Performance?
  - Python is slow... but not that slow
  - Time dominated by solution, not construction
  - Compiled code for solver/AD

$$
\begin{aligned}
\max \quad & \sum_{i \in \mathcal{S}} v_i \cdot x_i \\
\text{s.t.} \quad & \sum_{i \in \mathcal{S}} w_i \cdot x_i \leq W_m \\
& x_i \in \{0, 1\}
\end{aligned}
$$

- Flat Model Specification
  - Abstract models
  - Computer scientists

- Object-Oriented Modeling
  - Concrete models
  - Programmatic creation
  - Engineers



- Karl Åström's Comment: Don't just do what you did before with new technology

Aknowledgments (Development Community)

- Sandia National Laboratories
  - Bill Hart
  - Jean-Paul Watson
  - John Siirola
  - David Hart
  - Tom Brounstein
- University of California, Davis
  - Prof. David L. Woodruff
  - Prof. Roger Wets
- Texas A&M University
  - Prof. Carl D. Laird
  - Daniel Word
  - James Young
  - Gabe Hackebeil
- Texas Tech University
  - Zev Friedman
- Rose Hulman Institute
  - Tim Ekl
  - William & Mary
  - Patrick Steele
- North Carolina State
  - Kevin Hunter

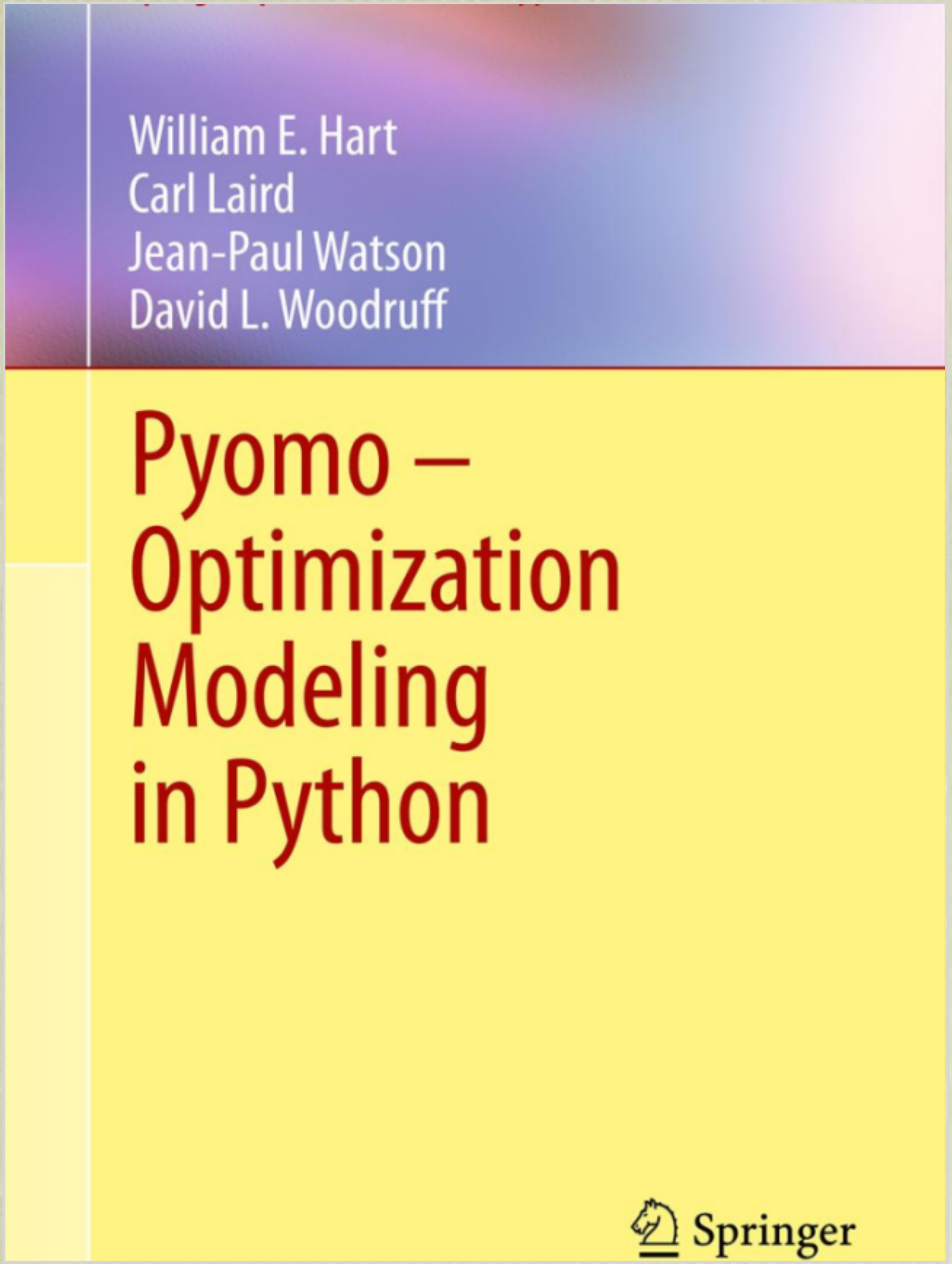
Plus our many users, including:
- University of California, Davis
- Texas A&M University
- University of Texas
- Rose-Hulman Institute of Technology
- University of Southern California
- George Mason University
- Iowa State University
- N.C. State University
- University of Washington
- Naval Postgraduate School
- Universidad de Santiago de Chile
- University of Pisa
- Lawrence Livermore National Lab
- Los Alamos National Lab

Learn More

- Project Homepage
  http://software.sandia.gov/coopr

- The Book

- Pyomo and PySP papers



Pyomo: Modeling and Solving Mathematical Programs in Python (Vol. 3, No. 3, 2011)
PySP: Modeling and Solving Stochastic Programs in Python (Vol. 4, No. 2, 2012)