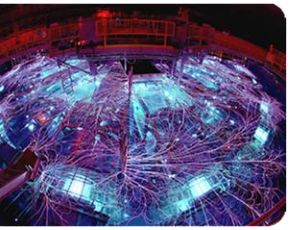
SAND2018-2485PE



Power Systems, Process Systems, and Equation-Oriented Optimization

John D. Siirola

Discrete Math & Optimization (1464)
Center for Computing Research
Sandia National Laboratories
Albuquerque, NM USA

Auburn University
7 March 2018

Sandia National Laboratories

Exceptional service in the national interest

U.S. DEPARTMENT OF ENERGY | NNSA National Nuclear Security Administration | CCR Center for Computing Research

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under Contract DE-NA0003525.

Acknowledgements

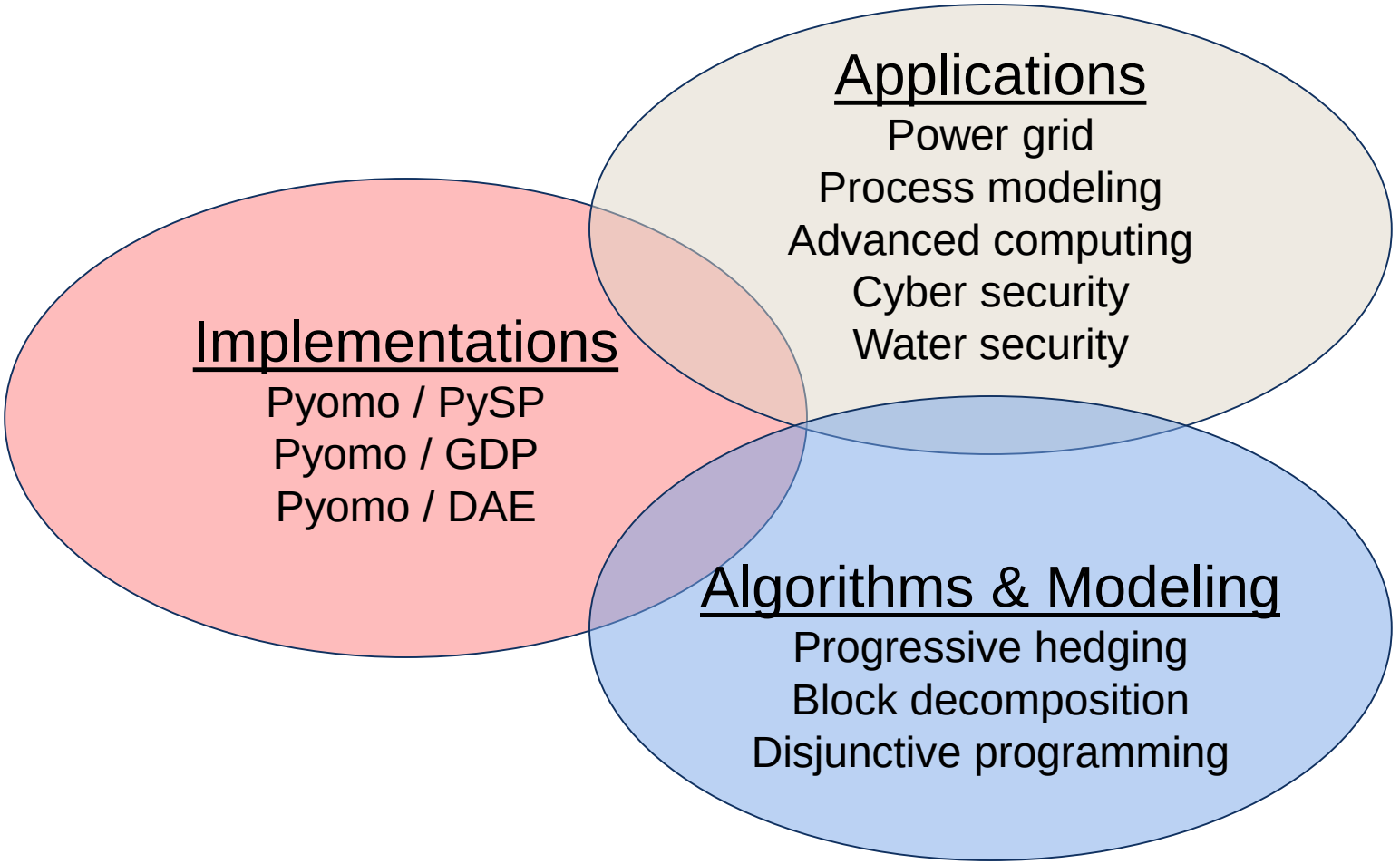
- Sandia National Laboratories
  - William E. Hart
  - Carl D. Laird
  - Bethany L. Nicholson
  - Jean-Paul Watson
  - Francisco D. Munoz (Universidad Adolfo Ibáñez)
- National Energy Technology Laboratory
  - David C. Miller
  - Anthony Burgard
  - Andrew Lee
  - John Eslick
- University of California, Davis
  - Prof. David L. Woodruff
- University of Michigan
  - Gabe Hackebeil
- Carnegie Mellon University
  - Qi Chen
  - Prof. Ignacio Grossmann
  - Prof. Larry Biegler

Portions of this work were conducted as part of the Institute for the Design of Advanced Energy Systems (IDAES) with funding from the Office of Fossil Energy, Cross-Cutting Research, U.S. Department of Energy

Operations Research at Sandia

- Sandia National Laboratories
  - Science & Technology Division
    - Center for Computing Research
      - Discrete Math and Optimization
- Sandia is a Federally-Funded Research and Development Center operated for the U.S. Department of Energy
  - Multimission national security laboratory
  - Historical focus on engineering applications
- Science & Technology Division is home to the Labs' *Research Foundations*
  - Pursue fundamental research driven by mission needs
- Discrete Math & Optimization Department
  - Conducts fundamental and applied OR/CS/Analytics research
  - Focus on algorithms, modeling approaches, and scalable tools
  - Partners with universities, other OR groups in Sandia "mission areas"

3 threads to our research (and this talk)



Operating the electric power grid

- Conceptually simple

$$\sum demand = \sum generation - \sum losses \quad \forall\, t \in T$$

  - This is just a single(*) component process flow network with fixed demands and controllable supplies

- In practice, this is complicated by
  - No (significant) storage
  - Dynamic constraints (ramp rates)
  - Transmission limitations
  - Security (reliability) requirements
  - Market constraints

(*) Actually, it is a 2-component system (real and reactive power) with "conversion" at every node, but for the purposes of this talk we will follow industry's lead and allow a small angle assumption to only work with the "DC" optimal power flow model.

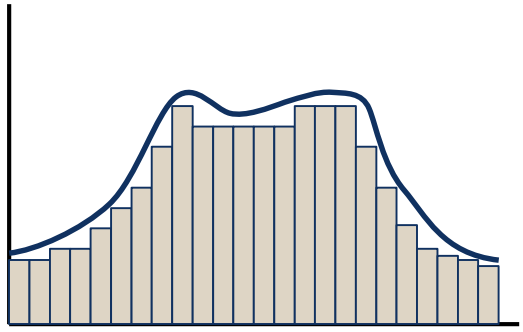
An ISO's view of operating the grid (1)

- [Unit Commitment]: Plan a day ahead (1600h)
  - Demand forecast
  - Generator bids
  - Produce:
    - Hourly (on/off) schedules for all participants
    - Hourly interconnect schedules
    - Hourly DAEM Locational Marginal Prices (LMPs)

"Day Ahead Market" (DAM)

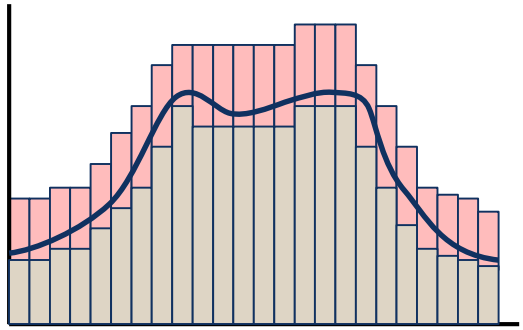
"Day Ahead Energy Market" (DAEM)

- So what's the problem?
  - *No one believes the forecast.*
  - *Things go wrong*



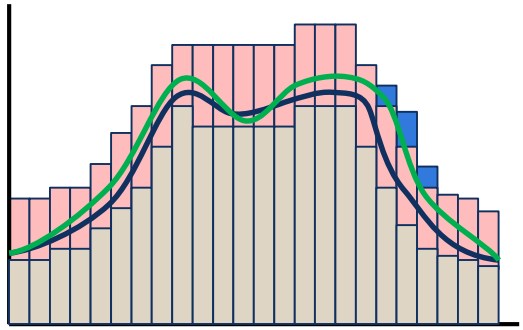
An ISO's view of operating the grid (2)

- [Reliability Unit Commitment]: Modify the plan
  - Reserve Adequacy Analysis – allocate reserves to meet load
    - Standard: 10% reserve requirement
  - Contingency analysis
    - *N-1*: survive loss of any (1) generator or (non-radial) line
  - Produces:
    - Additional commitments (DAEM respected)
    - Updated generator dispatch points



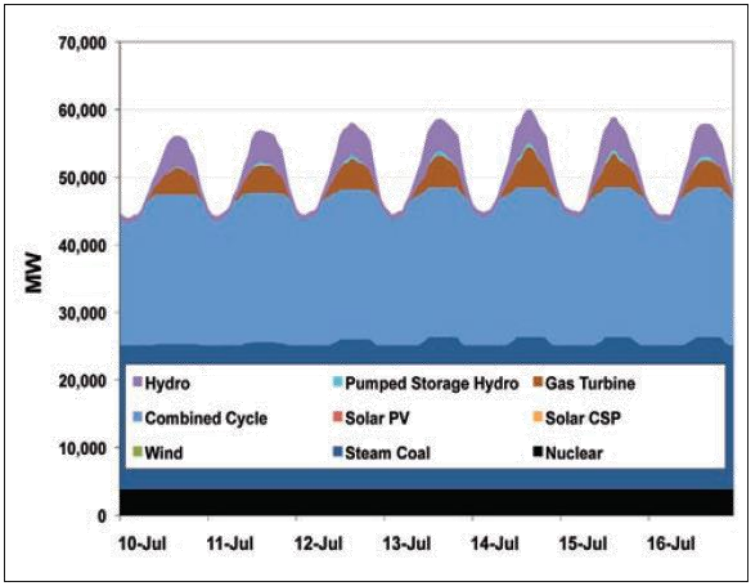
An ISO's view of operating the grid (3)

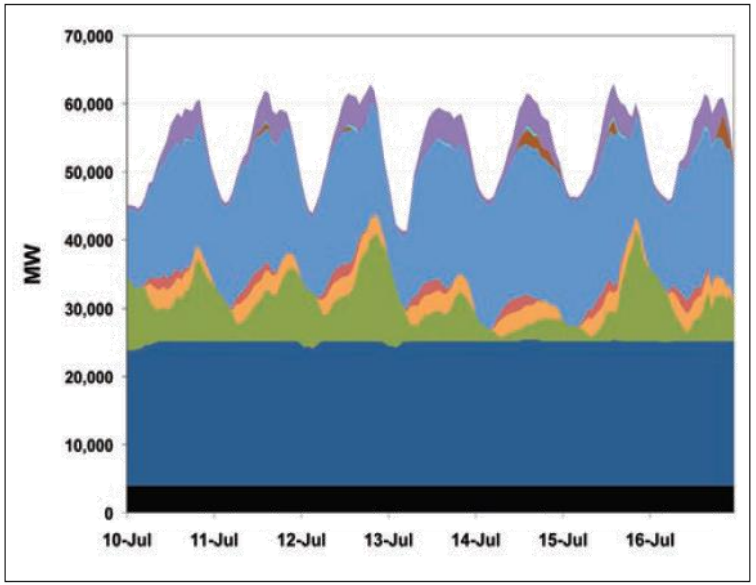
- [Economic Dispatch]: operate the grid / serve actual load
  - H – 1h: Look-ahead economic dispatch
  - H: Hourly economic dispatch
  - H+5*n*: 5-minute economic dispatch
  - Produces:
    - Updated dispatch points (generator output levels)
    - Additional commitments (fast-start units)



Given the lights are on, why care?

- Nondispatchable generation (renewables)
  - Frequently treated as “must-take” resources
    - Appears as “negative demand”
  - Less predictable than consumer demand
    - Increased reserve requirement

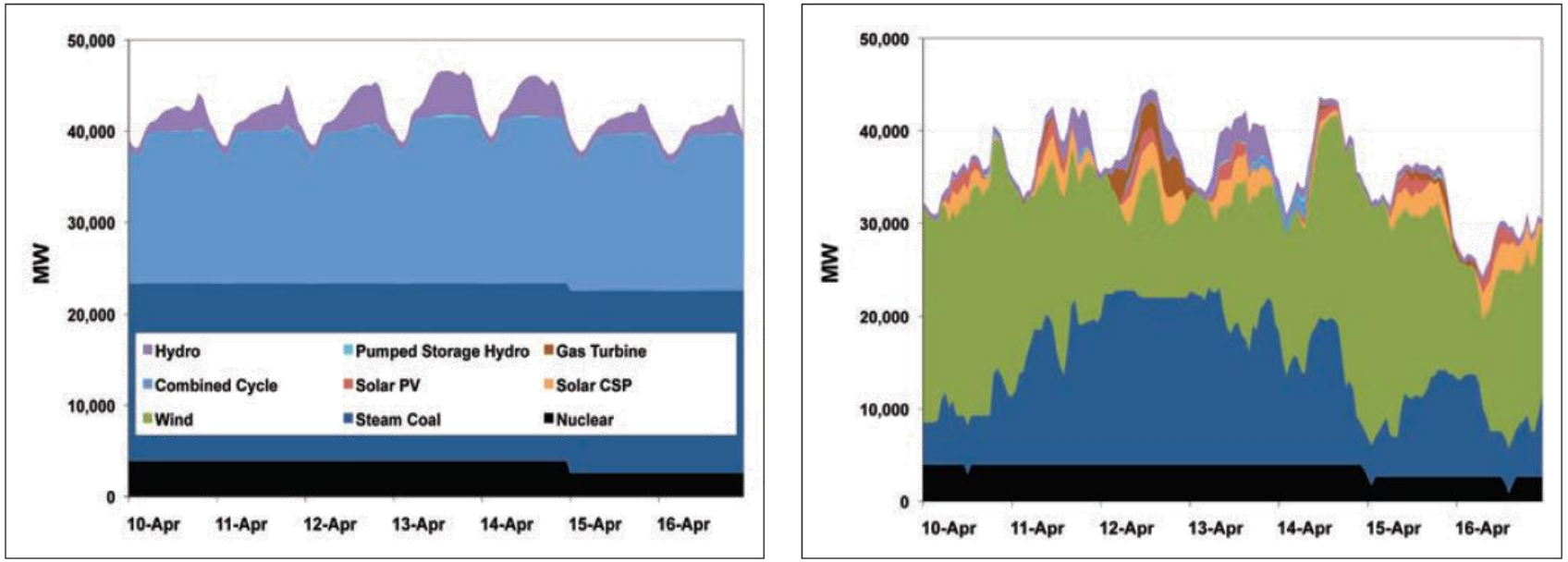




0% and 30% renewable penetration for an “easy week” in July.
Reproduced from NREL 2010 *Western Wind and Solar Integration Study*

Given the lights are on, why care?

- Nondispatchable generation (renewables)
  - Increased ramping of base-load generation
    - Results in increased O&M costs and higher forced outage rates
- Can we reduce cost by explicitly addressing uncertainty?



0% and 30% renewable penetration for an “challenging week” in April.
Reproduced from NREL 2010 *Western Wind and Solar Integration Study*

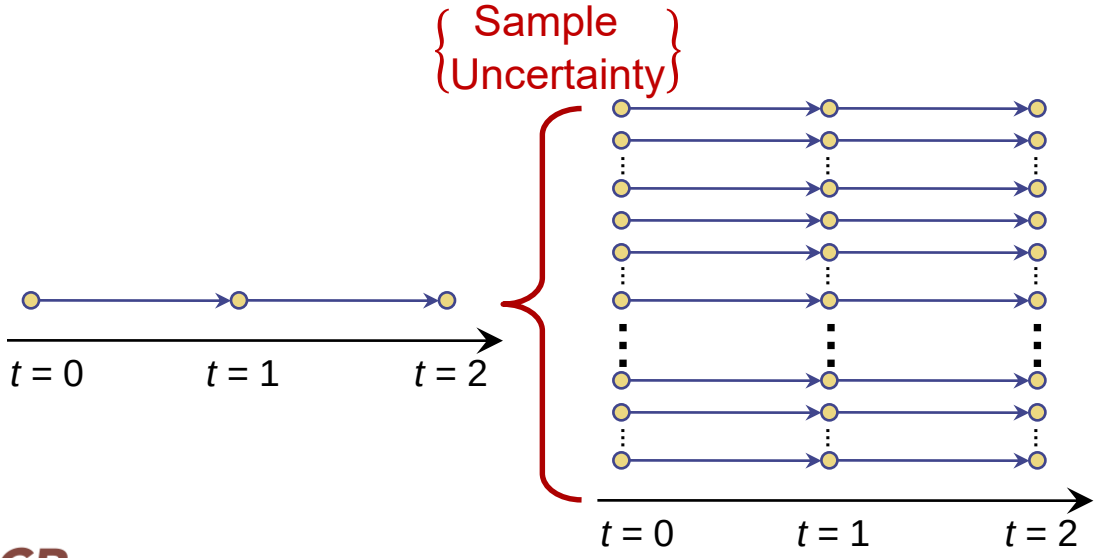
Stochastic Unit Commitment

Optimization under Uncertainty

- Many options for capturing uncertainty
  - Sampling / Surrogate methods / Robust optimization / (Approximate) dynamic programming / Stochastic programming
  - *We focus on stochastic programming*
    - Capture problem uncertainty as a set of possible scenarios
    - Solve to select a single answer that optimizes across all scenarios

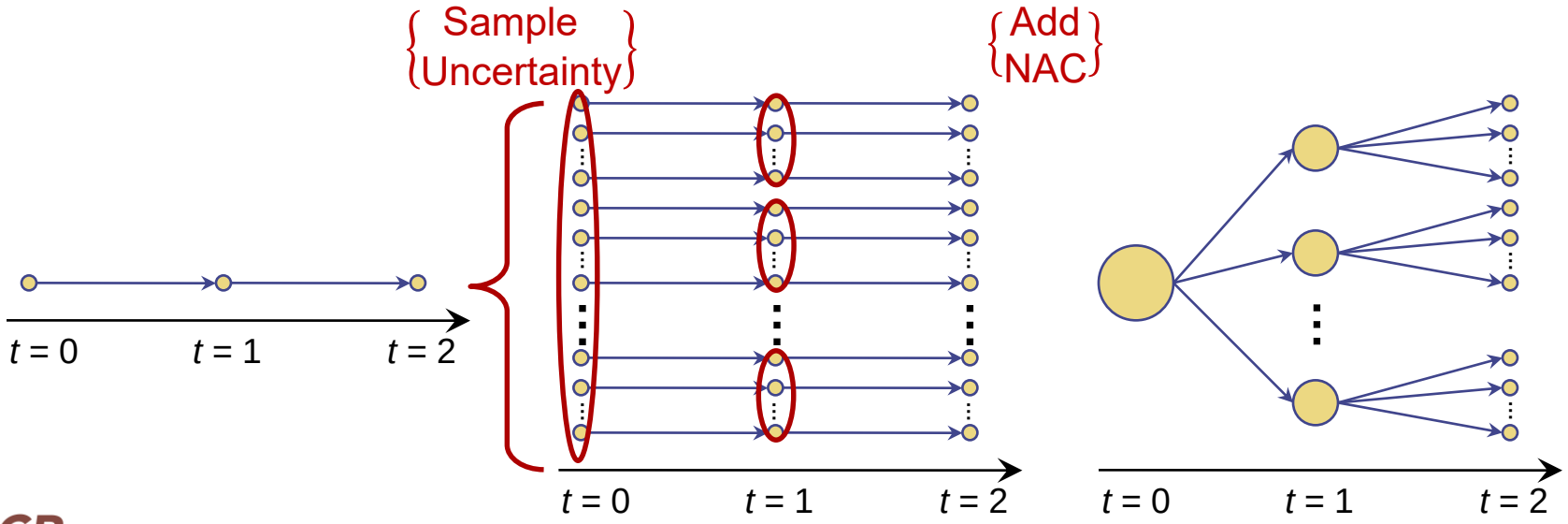
Optimization under Uncertainty

- Many options for capturing uncertainty
  - Sampling / Surrogate methods / Robust optimization / (Approximate) dynamic programming / Stochastic programming
  - *We focus on stochastic programming*
    - Capture problem uncertainty as a set of possible scenarios
    - Solve to select a single answer that optimizes across all scenarios



Optimization under Uncertainty

- Many options for capturing uncertainty
  - Sampling / Surrogate methods / Robust optimization / (Approximate) dynamic programming / Stochastic programming
  - *We focus on stochastic programming*
    - Capture problem uncertainty as a set of possible scenarios
    - Solve to select a single answer that optimizes across all scenarios



Pyomo: Equation-oriented modeling in Pyrhon

Sandia National Laboratories

- Pyomo is…
  - A library of classes for expressing optimization models in Python?
  - A collection of tools for importing and exporting data?
  - A set of interfaces to numerous LP, MIP, NLP, and MINLP solvers?
  - An executable that takes a model & data, sends it to a solver, and reports the final solution?
  - A collection of routines for manipulating optimization models?
  - A collection of optimization algorithms implemented in Python?
  - A software environment for teaching optimization?
  - An open-source community for applied OR research?
- But doesn't this sound a lot like AMPL, GAMS, AIMMS, gPROMS, …?
  - Why did we "reinvent the wheel"?

CCR Center for Computing Research

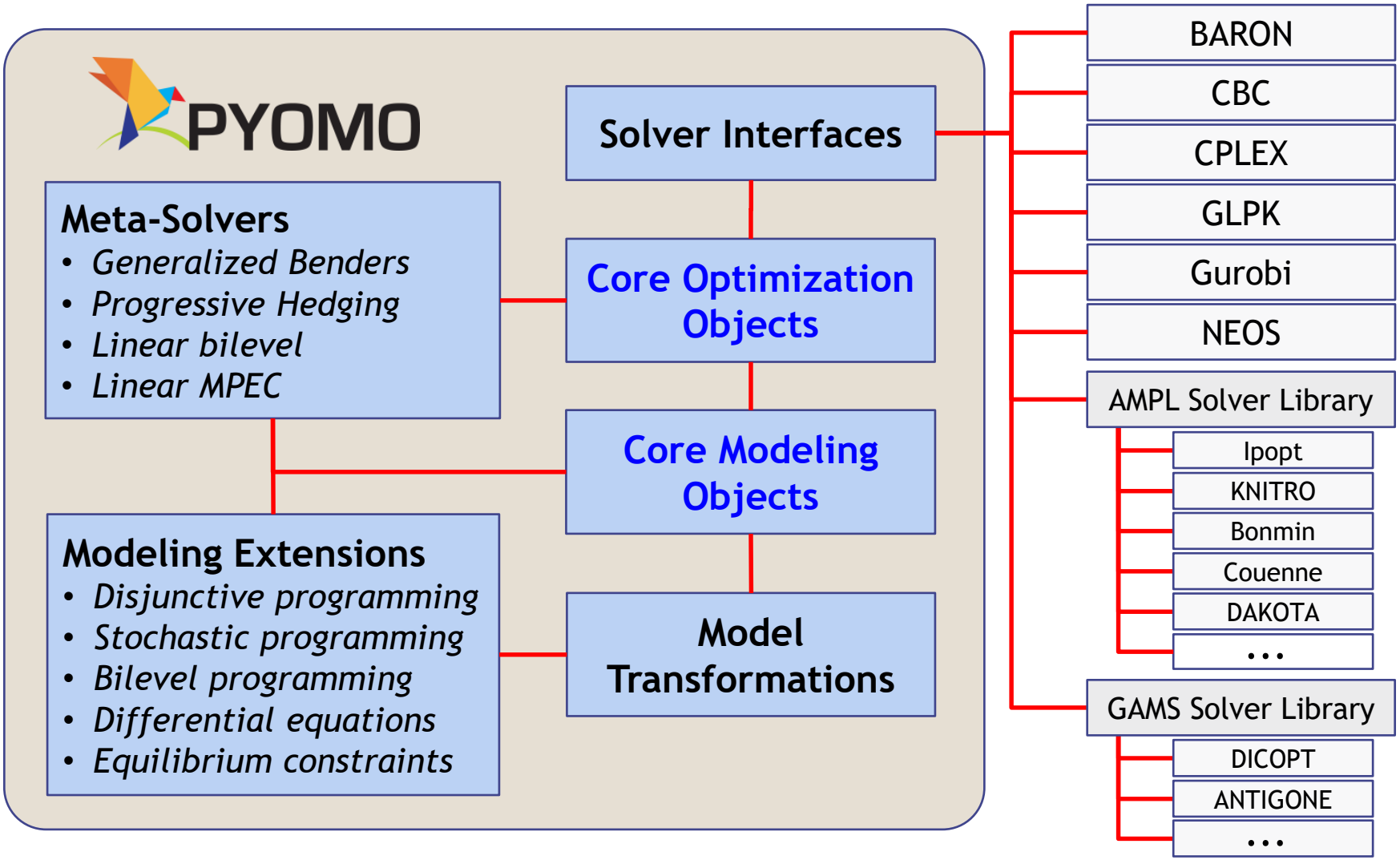
Why *we* needed Pyomo

Sandia National Laboratories

- 10 years ago…
  - Modeled primarily in AMPL (with some GAMS)
  - Developed solvers primarily in C++ (with MPI)
    - PICO, Coliny, DAKOTA, Opt++, APPSPACK
- This model worked, but…
  - Large code bases were unwieldly and not easily extensible
  - Commercial modeling environments lacked support for “higher level” modeling constructs, e.g. stochastic programming
  - No obvious path to implementing “meta algorithms”
    - Require frequent calls to optimization codes to solve subproblems
    - Difficult to implement in optimization modeling environments
    - Tedious to code directly against a solver’s API (and then you are tied to that solver)
  - Difficult to transfer our results to other organizations
  - Difficult to incorporate new ideas developed by the broader community
  - Dampened our ability to rapidly prototype ideas and explore new areas
    - We increasingly found ourselves providing *optimization support* to larger projects

CCR Center for Computing Research

Pyomo at a Glance



More than just mathematical modeling…

Sandia National Laboratories

Meta-solvers

- Integrate scripting and/or transformations into optimization solver
- Leverage Python's introspective nature to build "generic" capabilities
- e.g., progressive hedging, Benders decomposition

Model transformations (a.k.a. reformulations)

- Automate generation of one model from another
- Leverage Pyomo's object model to apply transformations sequentially
- e.g., DAE → NLP, GDP → Big M

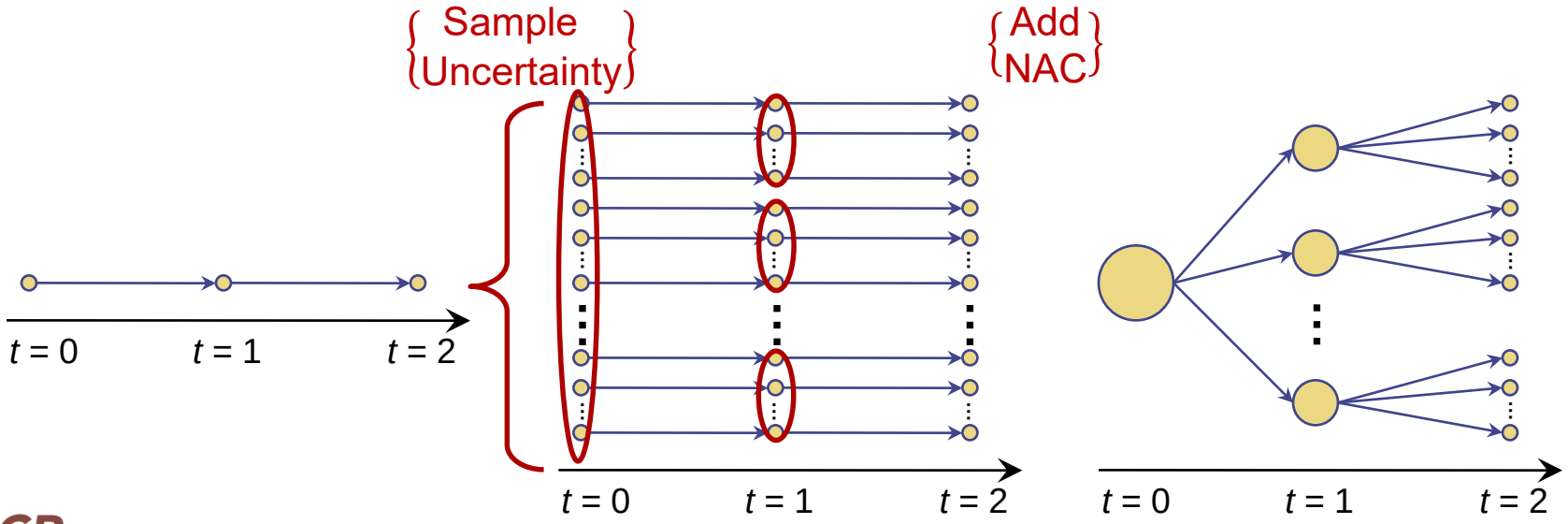
Scripting

- Construct models using native Python data
- Iterative analysis of models leveraging Python functionality
- Data analysis and visualization of optimization results

CCR Center for Computing Research

PySP: SP made Simple

- Step 1: write the deterministic model (in Pyomo)
  - You've (probably) already done this
- Step 2: generate scenario data
  - This is the hardest part (for UC, it took 2 years)
- Step 3: PySP expands the model & adds the NACs

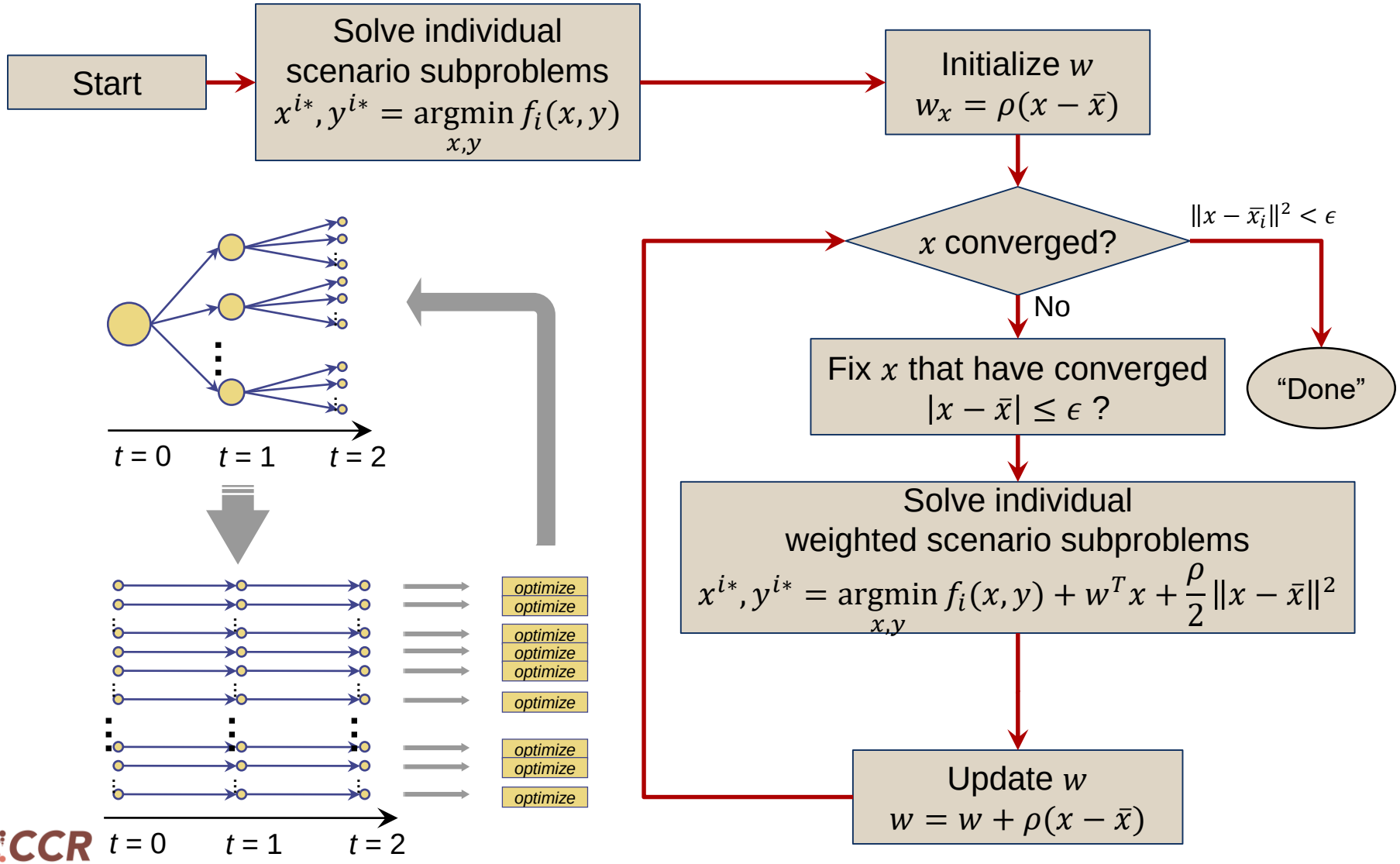


“So what?”

- The PySP process is conceptually no different than you would do in any other AML
  - Except in a traditional AML you would have to explicitly add and track the scenario index and add the NAC
  - If the Extensive Form is solvable, no significant difference between PySP and AMPL / GAMS / AIMMS…
    - But what if it’s not solvable?

- Structure: The real power of PySP
  - PySP explicitly understands the *structure* of the problem
  - We can *automate* decomposition strategies
    - Stage-wise decomposition (e.g., Benders decomposition)
    - Scenario-wise decomposition (e.g., Progressive hedging)

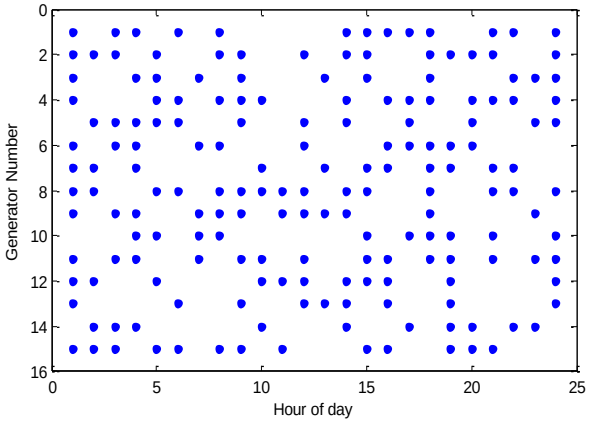
Progressive Hedging: the algorithm



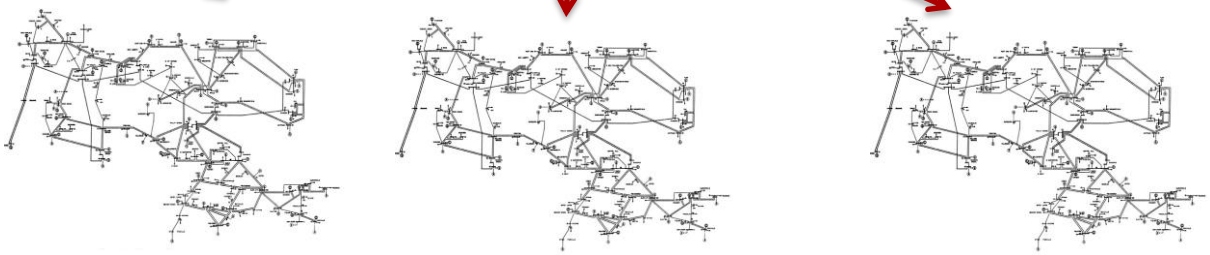
Stochastic Unit Commitment (at scale)

[J.-P. Watson, D. Woodruff]

Objective: Minimize expected cost



p_1 p_2 … p_N



Scenario 1 Scenario 2 … Scenario N

First stage variables:

- Unit On / Off

Nature resolves uncertainty

- Load
- Renewables output
- Forced outages

Second stage variables (*per time period*):

- Generation levels
- Power flows
- Voltage angles
- …

Progressive Hedging Results: WECC-240++

Sandia National Laboratories

Table 7 Solve time (in seconds) and solution quality statistics for PH executing on the *WECC-240-r1* instance, with $\alpha = 0.5$, $\mu = 6$, and $\gamma = 0.025$

# Scenarios	Convergence Metric	Obj. Value	PH L.B.	# Vars Fx.	Time	
		64-Core Workstation Results				Latest…
3	0.0 (20 iters)	64213.397	63235.381	4080	508	166
5	0.0 (in 18 iters)	62642.531	61767.253	4079	674	119
10	0.0 (in 35 iters)	61396.553	60476.604	4066	648	167
25	0.0 (in 22 iters)	60935.040	59992.622	4066	761	212
50	0.0 (in 15 iters)	60625.149	59631.839	4034	1076	280
100	0.0 (in 25 iters)	61155.387	60014.571	4080	1735	315
		Red Sky Results				
50	0.0 (in 16 iters)	60623.343	59779.813	4007	404	
100	0.0 (in 25 iters)	61120.943	60275.744	4080	549	

ISO-NE results are obtained on Red Sky on average in 10 minutes, 20 minutes in the worst case (with 100 scenarios)

CCR Center for Computing Research

Improved UC Formulations?

- Morales-Espana et al. (2013)
  - Extends prior tight formulation by Ostrowski et al.
- Shows off advantage of PH, in that improved deterministic models immediately impact stochastic solve times
- Results

Table 10 Solve time (in seconds) and solution quality statistics for PH executing on the *WECC-240-r1* instance, with $\alpha = 0.5$, $\mu = 3$, and the MTR deterministic UC model.

# Scenarios	Convergence Metric	Obj. Value	PH L.B.	# Vars Fx.	Time
		64-Core Workstation Results			
3	0.0 (in 36 iters)	64141.771	64109.021	4080	237
5	0.0 (in 23 iters)	62628.532	62499.212	4080	161
10	0.0 (in 26 iters)	61384.016	61327.734	4080	215
25	0.0 (in 41 iters)	60927.903	60850.717	4080	366
50	0.0 (in 11 iters)	60617.311	60470.956	4044	318

- ISO-NE results drop to 15 minutes maximum (10 average)

So what is the impact? ISO-NE analysis

- Cost-savings analysis for ISO-NE
  - 2004 Eastern Wind data
  - 50 wind scenarios per day
    - Generated using our tool chain based on epi-splines
    - (Simulated) actual taken from NREL database
  - 1 load scenario per day
    - Expected load computed using our epi-spline tool chain
    - Models fit using historical ISO-NE 2011 data
    - Actual taken from actual ISO-NE 2011 data
    - "Platinum" standard simulation, i.e., rolling horizon
  - Wind is *not* modeled as must-take
    - Per advice from NREL
    - In practice, there are days at these penetration levels in which it is impossible to use net load formulations w/o shedding

Cutting to the Chase: Cost Savings

Sandia National Laboratories

- Computed in terms of relative cost increase of deterministic (w/ 10% reserves) over stochastic (w/ 2% reserves)
  - Yes, this implies that stochastic does win (but)…
- Results in terms of percentages
  - Q1: 1.52%
  - Q2: 1.31%
  - Q3: 0.89%
  - Q4: 1.23%
- Not as significant as we would have anticipated, given the large wind penetration levels we simulated
  - For various reasons, we believe these results underestimate savings

CCR Center for Computing Research

Cutting to the Chase: Cost Savings

- Computed in terms of relative cost increase of deterministic (w/ 10% reserves) over stochastic (w/ 2% reserves)
  - Yes, this implies that stochastic does win (but)…

- Results in terms of percentages
  - Q1: 1.52% → ~$ 4M per month
  - Q2: 1.31% → ~$ 3M per month
  - Q3: 0.89% → ~$12M per month
  - Q4: 1.23% → ~$2.5M per month

  $64.5M "estimated savings" for 2011

- Not as significant as we would have anticipated, given the large wind penetration levels we simulated
  - For various reasons, we believe these results underestimate savings

Reliability Results

- We did not report load shedding and/or reserve shortfalls in the previous cost savings statistics
  - Placing arbitrary penalty values on these quantities is not useful
  - Distinct reporting allows more insight into system behaviors
- Stochastic UC
  - One load shedding event – peak day in July
  - Incurred due to particularly bad load forecast
- Deterministic UC
  - Five load shedding events – including the peak day in July
  - Additionally incurs reserve margin shortfalls on approximately 10% of all days in 2011
- Summary
  - Stochastic UC, despite lower reserve margins, is more reliable

Power grid contingency analysis

Power grid contingency analysis

Sandia National Laboratories

[with J.-P. Watson]

- U.S. ISO's must operate with "*N-1*" reliability
  - System must be able to "survive" loss of 1 generator / (non-radial) line
  - Not explicitly included in Unit Commitment model
  - Practice is to include "proxy constraints" and post-solve verification
- Some studies indicate that intentionally switching lines could improve contingency response
  - UC + N-1 + Transmission switching (e.g. [Hedman, et al. 2010])
    - *"Just allowing processing* [of the RTS-96 test case] *at the root node typically takes 20h on a desktop workstation…"*
    - *"While reducing [the] optimality gap to zero is an interesting academic issue…"*
- Case study: RTS-96 test case
  - 73 busses, 115 non-radial lines, 99 generators
  - 214 contingencies

CCR Center for Computing Research

What's the problem with Math Programming?

$$\text{Minimize}: \quad \sum_t \sum_g \left(c_g P_{g0t} + c_g^{SU} v_{gt} + c_g^{SD} w_{gt}\right)$$

$$\text{S.t.} \quad \theta^{\min} \le \theta_{nct} \le \theta^{\max}, \quad \forall\, n, c, t$$

$$\sum_{\forall\, k(n,.)} P_{kct} - \sum_{\forall\, k(.,n)} P_{kct} + \sum_{\forall\, g(n)} P_{g0t} = d_{nt},$$

$$\forall\, n, \quad c = 0, \text{ transmission contingency states } c, t$$

$$\sum_{\forall\, k(n,.)} P_{kct} - \sum_{\forall\, k(.,n)} P_{kct} + \sum_{\forall\, g(n)} P_{gct} = d_{nt},$$

$$\forall\, n, \text{ generator contingency states } c, t$$

$$P_{kc}^{\min} N1_{kc} z_{kt} \le P_{kct} \le P_{kc}^{\max} N1_{kc} z_{kt}, \quad \forall\, k, c, t$$

$$B_k(\theta_{nct} - \theta_{mct}) - P_{kct} + (2 - z_{kt} - N1_{kc}) M_k \ge 0, \quad \forall\, k, c, t$$

$$B_k(\theta_{nct} - \theta_{mct}) - P_{kct} - (2 - z_{kt} - N1_{kc}) M_k \le 0, \quad \forall\, k, c, t$$

$$P_g^{\min} N1_{gc} u_{gt} \le P_{gct} \le P_g^{\max} N1_{gc} u_{gt}, \quad \forall\, g, c, t$$

$$v_{g,t} - w_{g,t} = u_{g,t} - u_{g,t-1}, \quad \forall\, g, t$$

$$\sum_{q=t-UT_g+1}^{t} v_{g,q} \le u_{g,t}, \quad \forall\, g, t \in \{UT_g, \ldots, T\}$$

$$\sum_{q=t-DT_g+1}^{t} w_{g,q} \le 1 - u_{g,t}, \forall\, g, t \in \{DT_g, \ldots, T\}$$

$$P_{g0t} - P_{g0,t-1} \le R_g^{+} u_{g,t-1} + R_g^{SU} v_{g,t}, \quad \forall\, g, t$$

$$P_{g0,t-1} - P_{g0,t} \le R_g^{-} u_{g,t} + R_g^{SD} w_{g,t}, \quad \forall\, g, t$$

$$P_{gct} - P_{g0,t} \le R_g^{+}, \quad \forall\, g, c, t$$

$$P_{g0,t} N1_{gc} - P_{gct} \le R_g^{-}, \quad \forall\, g, c, t$$

$$0 \le v_{g,t} \le 1, \quad \forall\, g, t$$

$$0 \le w_{g,t} \le 1, \quad \forall\, g, t$$

$$u_{g,t} \in \{0, 1\}, \quad \forall\, g, t$$

- The "solution" for Unit Commitment + Transmission Switching + N-1 reliability

Hedman, et al., "Co-Optimization of Generation Unit Commitment and Transmission Switching With N-1 Reliability," *IEEE Trans Power Systems*, 25(2), pp.1052-1063, 2010

What's the problem with Math Programming?

Sandia National Laboratories

$$\text{Minimize}: \quad \sum_t \sum_g \left(c_g P_{g0t} + c_g^{SU} v_{gt} + c_g^{SD} w_{gt}\right)$$

$$\text{S.t.} \quad \theta^{\min} \le \theta_{nct} \le \theta^{\max}, \quad \forall\, n, c, t$$

$$\sum_{\forall\, k(n,.)} P_{kct} - \sum_{\forall\, k(.,n)} P_{kct} + \sum_{\forall\, g(n)} P_{g0t} = d_{nt},$$

$$\forall\, n, \quad c = 0, \text{ transmission contingency states } c, t$$

$$\sum_{\forall\, k(n,.)} P_{kct} - \sum_{\forall\, k(.,n)} P_{kct} + \sum_{\forall\, g(n)} P_{gct} = d_{nt},$$

$$\forall\, n, \text{ generator contingency states } c, t$$

$$P_{kc}^{\min} N1_{kc} z_{kt} \le P_{kct} \le P_{kc}^{\max} N1_{kc} z_{kt}, \quad \forall\, k, c, t$$

$$B_k(\theta_{nct} - \theta_{mct}) - P_{kct} + (2 - z_{kt} - N1_{kc}) M_k \ge 0, \quad \forall\, k, c, t$$

$$B_k(\theta_{nct} - \theta_{mct}) - P_{kct} - (2 - z_{kt} - N1_{kc}) M_k \le 0, \quad \forall\, k, c, t$$

$$P_g^{\min} N1_{gc} u_{gt} \le P_{gct} \le P_g^{\max} N1_{gc} u_{gt}, \quad \forall\, g, c, t$$

$$v_{g,t} - w_{g,t} = u_{g,t} - u_{g,t-1}, \quad \forall\, g, t$$

$$\sum_{q=t-UT_g+1}^{t} v_{g,q} \le u_{g,t}, \quad \forall\, g, t \in \{UT_g, \ldots, T\}$$

$$\sum_{q=t-DT_g+1}^{t} w_{g,q} \le 1 - u_{g,t}, \forall\, g, t \in \{DT_g, \ldots, T\}$$

$$P_{g0t} - P_{g0,t-1} \le R_g^{+} u_{g,t-1} + R_g^{SU} v_{g,t}, \quad \forall\, g, t$$

$$P_{g0,t-1} - P_{g0,t} \le R_g^{-} u_{g,t} + R_g^{SD} w_{g,t}, \quad \forall\, g, t$$

$$P_{gct} - P_{g0,t} \le R_g^{+}, \quad \forall\, g, c, t$$

$$P_{g0,t} N1_{gc} - P_{gct} \le R_g^{-}, \quad \forall\, g, c, t$$

$$0 \le v_{g,t} \le 1, \quad \forall\, g, t$$

$$0 \le w_{g,t} \le 1, \quad \forall\, g, t$$

$$u_{g,t} \in \{0, 1\}, \quad \forall\, g, t$$

- The "solution" for Unit Commitment + Transmission Switching + N-1 reliability
- The limited MP "toolbox"
  - $+,\ -,\ \times,\ \div$
  - sin, cos, tan, *etc.*
  - $y^x,\ e^x,\ \log_{10}(x)\,, \ln(x)$
  - *(functions in C^2)*

CCR Center for Computing Research

Hedman, et al., "Co-Optimization of Generation Unit Commitment and Transmission Switching With N-1 Reliability," *IEEE Trans Power Systems*, 25(2), pp.1052-1063, 2010

The challenge: MP is dense and subtle

Sandia National Laboratories

Minimize :

$$\sum_t \sum_g \left(c_g P_{g0t} + c_g^{SU} v_{gt} + c_g^{SD} w_{gt}\right)$$

S.t.

$$\theta^{\min} \le \theta_{nct} \le \theta^{\max}, \quad \forall\, n, c, t$$

$$\sum_{\forall\, k(n,.)} P_{kct} - \sum_{\forall\, k(.,n)} P_{kct} + \sum_{\forall\, g(n)} P_{g0t} = d_{nt},$$

$$\forall\, n, \quad c = 0, \text{ transmission contingency states } c, t$$

$$\sum_{\forall\, k(n,.)} P_{kct} - \sum_{\forall\, k(.,n)} P_{kct} + \sum_{\forall\, g(n)} P_{gct} = d_{nt},$$

$$\forall\, n, \text{ generator contingency states } c, t$$

$$P_{kc}^{\min} N1_{kc} z_{kt} \le P_{kct} \le P_{kc}^{\max} N1_{kc} z_{kt}, \quad \forall\, k, c, t$$

$$B_k(\theta_{nct} - \theta_{mct}) - P_{kct} + (2 - z_{kt} - N1_{kc}) M_k \ge 0, \quad \forall\, k, c, t$$

$$B_k(\theta_{nct} - \theta_{mct}) - P_{kct} - (2 - z_{kt} - N1_{kc}) M_k \le 0, \quad \forall\, k, c, t$$

$$P_g^{\min} N1_{gc} u_{gt} \le P_{gct} \le P_g^{\max} N1_{gc} u_{gt}, \quad \forall\, g, c, t$$

$$v_{g,t} - w_{g,t} = u_{g,t} - u_{g,t-1}, \quad \forall\, g, t$$

$$\sum_{q=t-UT_g+1}^{t} v_{g,q} \le u_{g,t}, \quad \forall\, g, t \in \{UT_g, \ldots, T\}$$

$$\sum_{q=t-DT_g+1}^{t} w_{g,q} \le 1 - u_{g,t}, \forall\, g, t \in \{DT_g, \ldots, T\}$$

$$P_{g0t} - P_{g0,t-1} \le R_g^{+} u_{g,t-1} + R_g^{SU} v_{g,t}, \quad \forall\, g, t$$

$$P_{g0,t-1} - P_{g0,t} \le R_g^{-} u_{g,t} + R_g^{SD} w_{g,t}, \quad \forall\, g, t$$

$$P_{gct} - P_{g0,t} \le R_g^{+}, \quad \forall\, g, c, t$$

$$P_{g0,t} N1_{gc} - P_{gct} \le R_g^{-}, \quad \forall\, g, c, t$$

$$0 \le v_{g,t} \le 1, \quad \forall\, g, t$$

$$0 \le w_{g,t} \le 1, \quad \forall\, g, t$$

$$u_{g,t} \in \{0, 1\}, \quad \forall\, g, t$$

To a first approximation:

- DCOPF
- Economic dispatch
- Unit commitment
- Transmission switching
- N-1 contingency

CCR Center for Computing Research

Hedman, et al., "Co-Optimization of Generation Unit Commitment and Transmission Switching With N-1 Reliability," *IEEE Trans Power Systems*, 25(2), pp.1052-1063, 2010

Sidebar: What do these have in common?

$$a = \sqrt{(x-3)^2+\epsilon}$$

$$\begin{aligned} a &= b + c \\ b &\leq M \cdot y \\ c &\leq M(1-y) \\ x-3 &= c-b \\ b &\geq 0 \\ c &\geq 0 \\ y &\in \{0,1\} \end{aligned}$$

$$\begin{aligned} a &\geq x-3 \\ a &\geq 3-x \end{aligned}$$

$$\begin{aligned} a &= b + c \\ x-3 &= c-b \\ b \geq 0 &\perp c \geq 0 \end{aligned}$$

$$a = \frac{2(x-3)}{1+e^{-\frac{x-3}{h}}} - x + 3$$

Sidebar: What do these have in common?

$$a = \sqrt{(x-3)^2+\epsilon}$$

$$\begin{aligned} a &= b + c \\ b &\le M \cdot y \\ c &\le M(1-y) \\ x-3 &= c-b \\ b &\ge 0 \\ c &\ge 0 \\ y &\in \{0,1\} \end{aligned}$$

$$a = abs(x-3)$$

$$\begin{aligned} a &\ge x-3 \\ a &\ge 3-x \end{aligned}$$

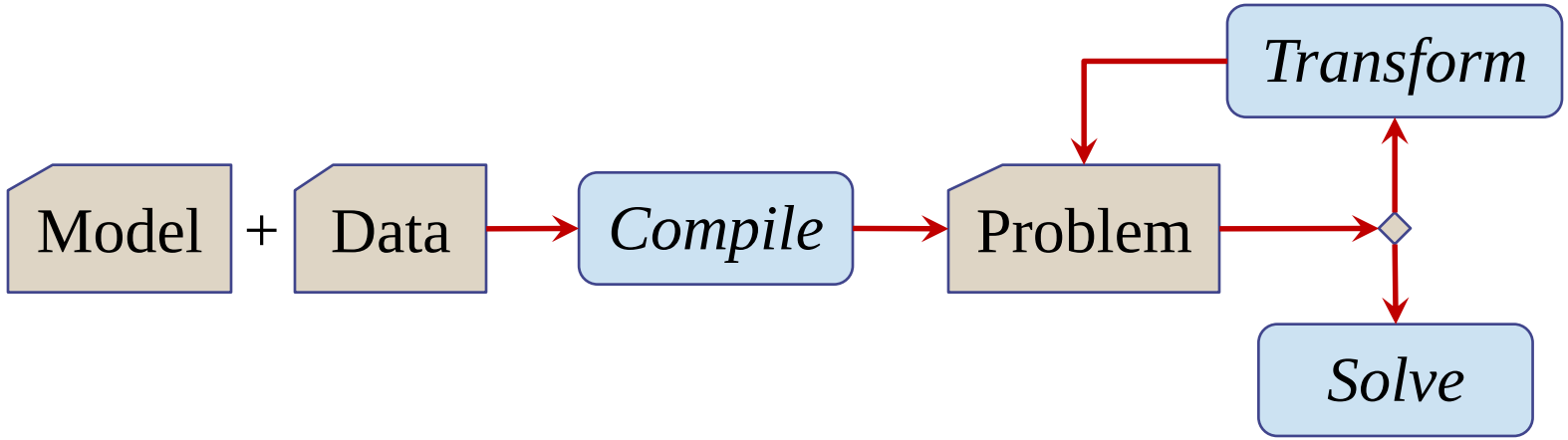
$$\begin{aligned} a &= b + c \\ x-3 &= c-b \\ b \ge 0 &\perp c \ge 0 \end{aligned}$$

$$a = \frac{2(x-3)}{1+e^{-\frac{x-3}{h}}} - x + 3$$

If we *mean* "$a = abs(x-3)$",
why don't we *write* that in our models???

A new solution workflow

- Model Transformations: Projecting problems to problems
  - Project from one problem space to another
  - Standardize common reformulations or approximations
  - Enables "Extended Math Programming"[1]
    - Develop new modeling constructs not supported by solvers
    - (Automatically) Convert these "unoptimizable" modeling constructs into equivalent optimizable forms



[1] - Ferris, et al. "An extended mathematical programming framework". *Computers & Chemical Engineering* 33(12) 2009.

Extended Math Programming (EMP)

Sandia National Laboratories

- Ferris, et al. (2009)
  - Modeling framework (domain-specific language) built on GAMS
  - Adds *support for "higher level"* constructs
    - Complementarity conditions, Variational inequalities, Bilevel problems, Disjunctive programming
  - Constructs are annotated through a separate input file
  - Interfaces to specialized solvers or provides *automated reformulations* for standard solvers

- Alternatively, EMP concepts could be implemented through an *object-oriented* framework

CCR Center for Computing Research

Why are we interested in transformations?

Sandia National Laboratories

- Separate model expression from how we intend to solve it
  - Defer decisions that improve tractability until solution time
  - Explore alternative reformulations or representations
  - Support *solver-specific* model customizations (e.g., `abs()`)
  - Support iterative methods that use different solvers requiring different representations (e.g., initializing NLP from MIP)
- Support "higher level" or non-algebraic modeling constructs
  - Express models that are "closer" to reality, e.g.:
    - Piecewise expressions
    - Disjunctive models (switching decisions & logic models)
    - Differential-algebraic models (dynamic models)
    - Bilevel models (game theory models)
- Reduce "mechanical" errors due to manual reformulation

Block-oriented modeling

- "Blocks"
  - Collections of model components
    - Variable, Parameter, Set, Constraint, etc.
  - Blocks may be arbitrarily nested
- Why blocks?
  - Support reusable modeling components
  - Express distinctly modeled concepts as distinct objects
  - Manipulate modeled components as distinct entities
  - Explicitly expose model structure (e.g., for decomposition)
  - Enables transformations and component namespaces
- Prior art
  - Ubiquitous in the simulation community
  - Rare in Math Programming environments
    - *Notable exceptions:* ASCEND, JModelica.org

Generalized disjunctive programming

- Disjunctions: selectively enforce sets of constraints
  - Sequencing decisions: x ends before y or y ends before x
  - Switching decisions: a process unit is built or not
  - Alternative selection: selecting from a set of pricing policies

- Implementation: leverage Pyomo "blocks"
  - `Disjunct`:
    - Block of Pyomo components
      - (Variable, Parameter, Constraint, etc.)
    - Boolean (binary) indicator variable determines if block is enforced
  - `Disjunction`:
    - Enforces logical OR/XOR across a set of Disjunct indicator variables
  - Logic constraints on indicator variables

$$\bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ h_{ik}(x) \le o \\ c_k = \gamma_{ik} \end{bmatrix}$$
$$\Omega(Y) = true$$

Simple Example: Task sequencing in Pyomo

```
model.TASKS = Set()
model.STAGES = Set()
model.L = Set(within= model.TASKS*model.TASKS*model.STAGES)
model.tau = Param(model.TASKS, model.STAGES)
model.t = Var(model.TASKS)

def _NoCollision(model, i, k, j):
    lhs = model.t[i] + sum(model.tau[i,m] for m in model.STAGES if m<j)
    rhs = model.t[k] + sum(model.tau[k,m] for m in model.STAGES if m<j)
    return [ lhs + model.tau[i,j] <= rhs,   rhs + model.tau[k,j] <= lhs ]
model.NoCollision = Disjunction( model.L, rule=_NoCollision )
```

$$\left[\begin{array}{c} Y_{ik} \\ t_i + \sum_{\substack{m \in J(i) \\ m<j}} \tau_{im} + \tau_{ij} \le t_k + \sum_{\substack{m \in J(k) \\ m<j}} \tau_{km} \end{array}\right] \vee \left[\begin{array}{c} Y_{ki} \\ t_k + \sum_{\substack{m \in J(k) \\ m<j}} \tau_{km} + \tau_{kj} \le t_i + \sum_{\substack{m \in J(i) \\ m<j}} \tau_{im} \end{array}\right]$$

$$\forall j \in C_{ik}, \forall i,k \in I, i<k$$

Simple Example: Task sequencing in Pyomo

```
model.TASKS = Set()
model.STAGES = Set()
model.L = Set(within= model.TASKS*model.TASKS*model.STAGES)
model.tau = Param(model.TASKS, model.STAGES)
model.t = Var(model.TASKS)

def _NoCollision(model, i, k, j):
    lhs = model.t[i] + sum(model.tau[i,m] for m in model.STAGES if m<j)
    rhs = model.t[k] + sum(model.tau[k,m] for m in model.STAGES if m<j)
    return [ lhs + model.tau[i,j] <= rhs,   rhs + model.tau[k,j] <= lhs ]
model.NoCollision = Disjunction( model.L, rule=_NoCollision )
```

$$\left[t_i + \sum_{\substack{m \in J(i) \\ m<j}} \tau_{im} + \overset{Y_{ik}}{\tau_{ij}} \leq t_k + \sum_{\substack{m \in J(k) \\ m<j}} \tau_{km} \right] \vee \left[t_k + \sum_{\substack{m \in J(k) \\ m<j}} \tau_{km} + \overset{Y_{ki}}{\tau_{kj}} \leq t_i + \sum_{\substack{m \in J(i) \\ m<j}} \tau_{im} \right]$$

$$\forall j \in C_{ik}, \forall i,k \in I, i<k$$

Solving disjunctive models

Sandia National Laboratories

- Few solvers "understand" disjunctive models
  - *Transform* model into standard math program
  - Big-M relaxation:
    - Relax all constraints in the disjuncts with "appropriate" M values
      - Automatically calculate M values for linear expressions

```
pyomo solve --solver=cbc --transform=gdp.bigm jobshop.py jobshop.dat
```

  - Convex hull relaxation (Balas, 1985; Lee and Grossmann, 2000)
    - Disaggregate variables in all disjuncts
    - Bound disaggregated variables with Big-M terms

```
pyomo solve --solver=cbc --transform=gdp.chull jobshop.py jobshop.dat
```

  - Algorithmic approaches
    - General cutting planes (e.g., Trespalacios & Grossmann)
    - Basic step-based tightening approaches (e.g., Trespalacios & Grossmann)
    - Logic-based outer approximation (e.g., Turkay & Grossmann, Chen & Grossmann)

CCR Center for Computing Research

A transformation-centric view of abs()

- Chaining transformations

$$f = abs(x) \Rightarrow \begin{array}{c} f = x^+ + x^- \\ x = x^+ - x^- \\ x^+ \geq 0 \perp x^- \geq 0 \end{array} \Rightarrow \begin{array}{c} f = x^+ + x^- \\ x = x^+ - x^- \\ \begin{bmatrix} Y \\ x^- = 0 \end{bmatrix} \vee \begin{bmatrix} \neg Y \\ x^+ = 0 \end{bmatrix} \\ x^+ \geq 0,\ x^- \geq 0 \end{array} \Rightarrow \begin{array}{c} f = x^+ + x^- \\ x = x^+ - x^- \\ x^- \leq My \\ x^- \leq M(1-y) \\ x^+ \geq 0,\ x^- \geq 0 \end{array}$$

```
model = ConcreteModel()
# […]
TransformFactory("abs.complements").apply_to(model)
TransformFactory("mpec.disjunctive").apply_to(model)
TransformFactory("gdp.bigm").apply_to(model)
```

Returning to RUC + Transmission Switching

Sandia National Laboratories

Minimize :
$$\sum_t \sum_g \left(c_g P_{g0t} + c_g^{SU} v_{gt} + c_g^{SD} w_{gt}\right)$$

S.t.
$$\theta^{\min} \le \theta_{nct} \le \theta^{\max}, \quad \forall\, n, c, t$$
$$\sum_{\forall\, k(n,.)} P_{kct} - \sum_{\forall\, k(.,n)} P_{kct} + \sum_{\forall\, g(n)} P_{g0t} = d_{nt},$$
$$\forall\, n, \quad c = 0, \text{ transmission contingency states } c, t$$
$$\sum_{\forall\, k(n,.)} P_{kct} - \sum_{\forall\, k(.,n)} P_{kct} + \sum_{\forall\, g(n)} P_{gct} = d_{nt},$$
$$\forall\, n, \text{ generator contingency states } c, t$$
$$P_{kc}^{\min} N1_{kc} z_{kt} \le P_{kct} \le P_{kc}^{\max} N1_{kc} z_{kt}, \quad \forall\, k, c, t$$
$$B_k(\theta_{nct} - \theta_{mct}) - P_{kct} + (2 - z_{kt} - N1_{kc}) M_k \ge 0, \quad \forall\, k, c, t$$
$$B_k(\theta_{nct} - \theta_{mct}) - P_{kct} - (2 - z_{kt} - N1_{kc}) M_k \le 0, \quad \forall\, k, c, t$$
$$P_g^{\min} N1_{gc} u_{gt} \le P_{gct} \le P_g^{\max} N1_{gc} u_{gt}, \quad \forall\, g, c, t$$
$$v_{g,t} - w_{g,t} = u_{g,t} - u_{g,t-1}, \quad \forall\, g, t$$
$$\sum_{q=t-UT_g+1}^{t} v_{g,q} \le u_{g,t}, \quad \forall\, g, t \in \{UT_g, \ldots, T\}$$
$$\sum_{q=t-DT_g+1}^{t} w_{g,q} \le 1 - u_{g,t}, \forall\, g, t \in \{DT_g, \ldots, T\}$$
$$P_{g0t} - P_{g0,t-1} \le R_g^{+} u_{g,t-1} + R_g^{SU} v_{g,t}, \quad \forall\, g, t$$
$$P_{g0,t-1} - P_{g0,t} \le R_g^{-} u_{g,t} + R_g^{SD} w_{g,t}, \quad \forall\, g, t$$
$$P_{gct} - P_{g0,t} \le R_g^{+}, \quad \forall\, g, c, t$$
$$P_{g0,t} N1_{gc} - P_{gct} \le R_g^{-}, \quad \forall\, g, c, t$$
$$0 \le v_{g,t} \le 1, \quad \forall\, g, t$$
$$0 \le w_{g,t} \le 1, \quad \forall\, g, t$$
$$u_{g,t} \in \{0, 1\}, \quad \forall\, g, t$$

To a first approximation:

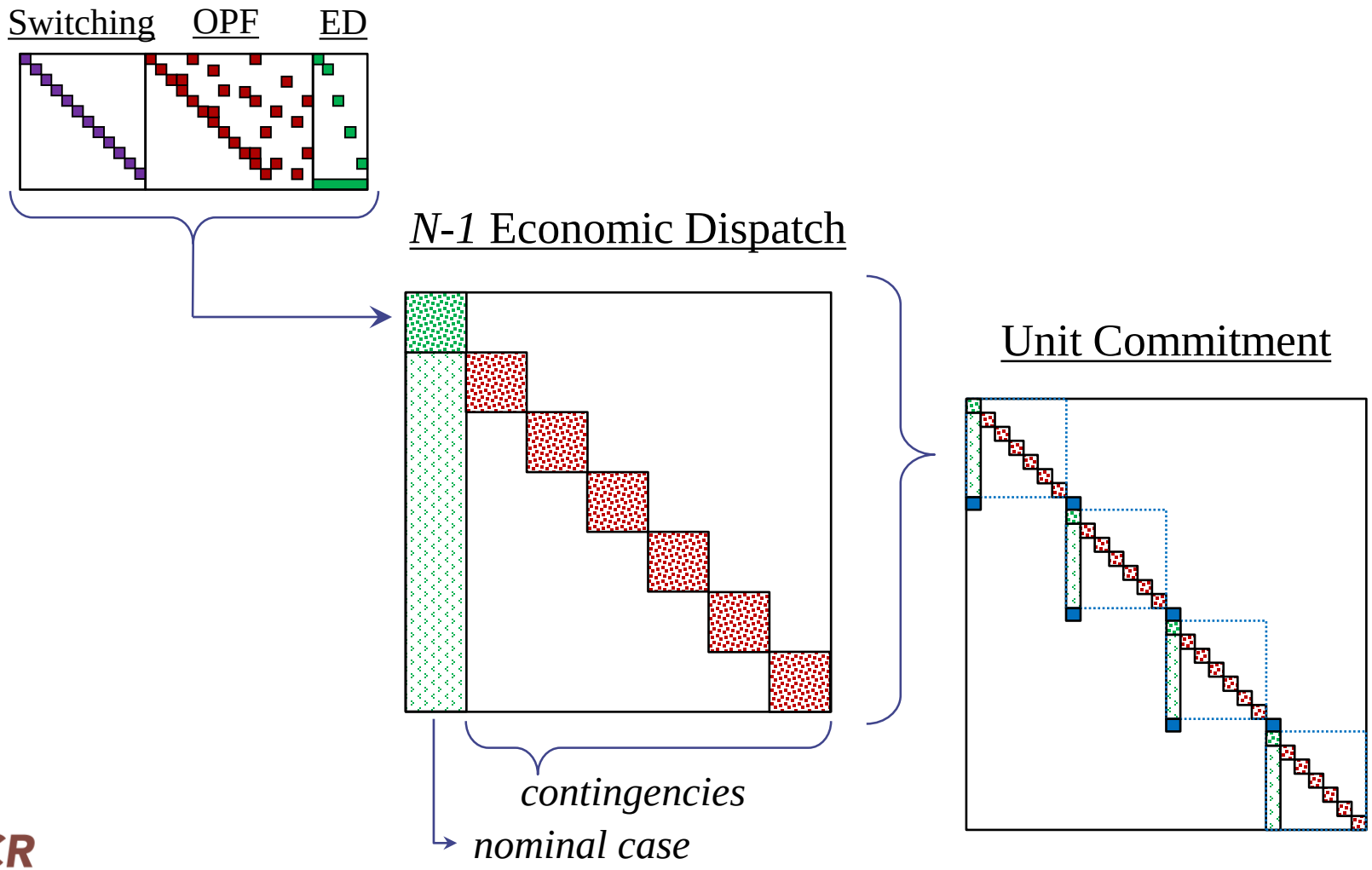
- DCOPF
- Economic dispatch
- Unit commitment
- Transmission switching
- N-1 contingency

CCR Center for Computing Research

Hedman, et al., "Co-Optimization of Generation Unit Commitment and Transmission Switching With N-1 Reliability," *IEEE Trans Power Systems*, 25(2), pp.1052-1063, 2010

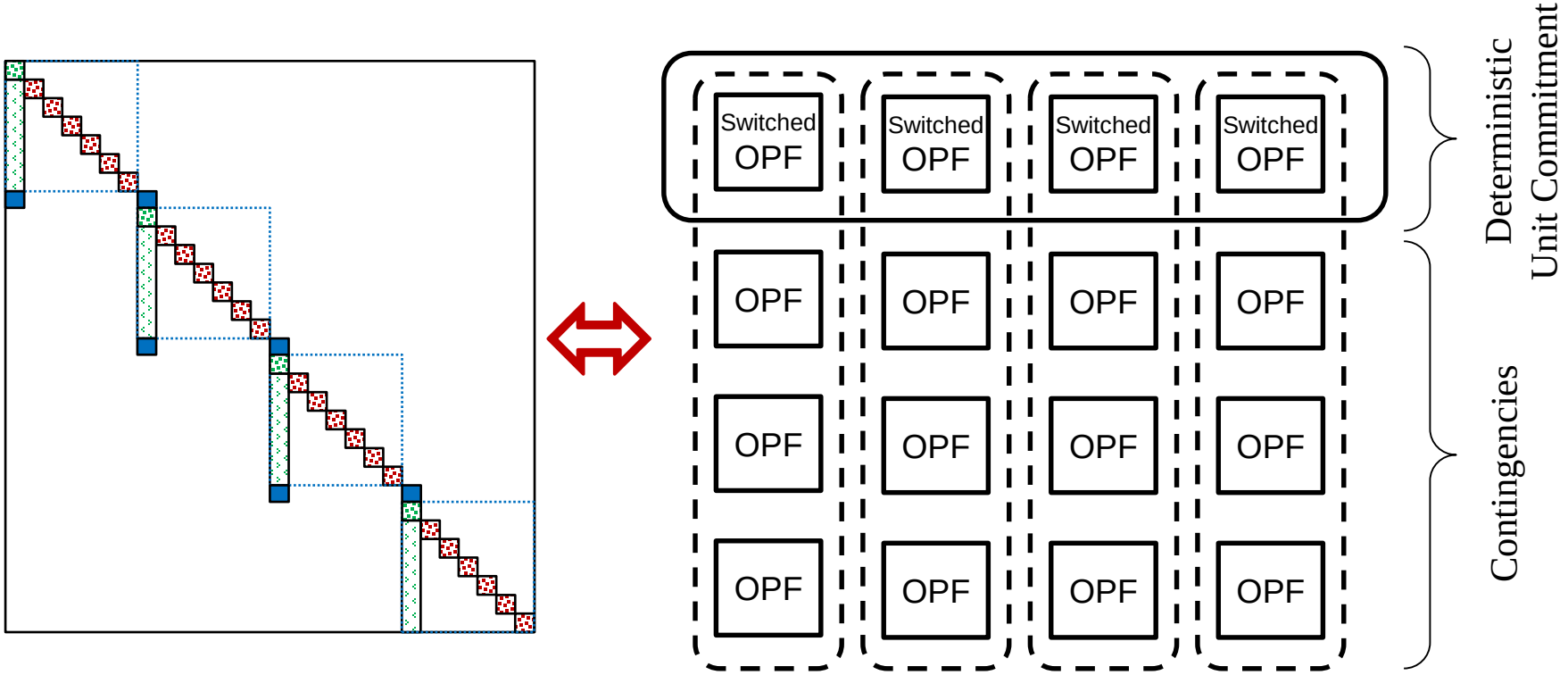
(Nonobvious) Inherent structure

(Sparsity pattern in model constraint matrix)



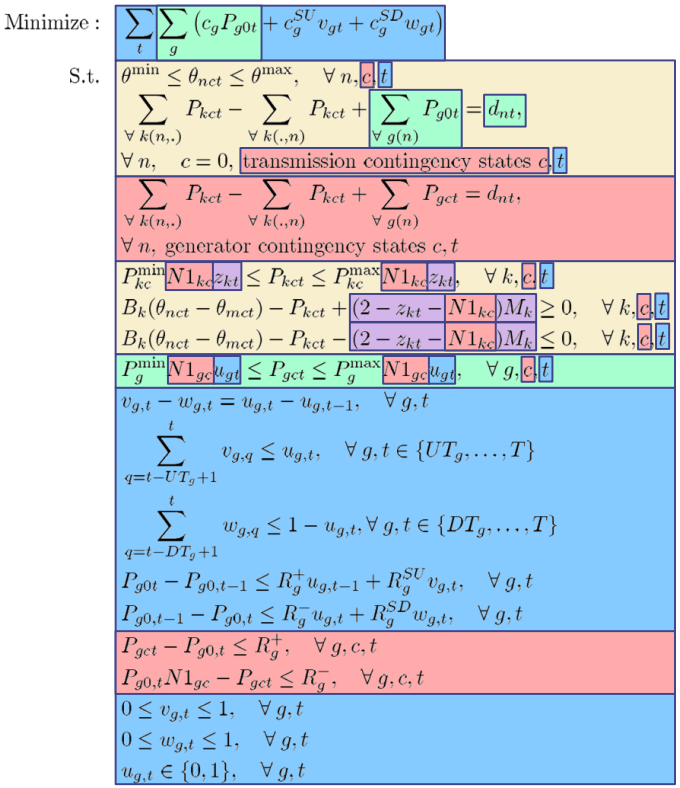
UC + N-1 + Switching block structure

- "2-D" grid of linked optimal power flow models



Expressing & preserving modeler intent

- If the *N-1* model has all this structure, why do we write it



- What I want:
  - Clear, structured syntax
  - Explicit switching decisions
  - Express block structure
    - Define meaningful components
    - “bus,” “line,” “generator”
  - Control over how this gets mapped to the solver

Explicitly expose disjunctive decisions

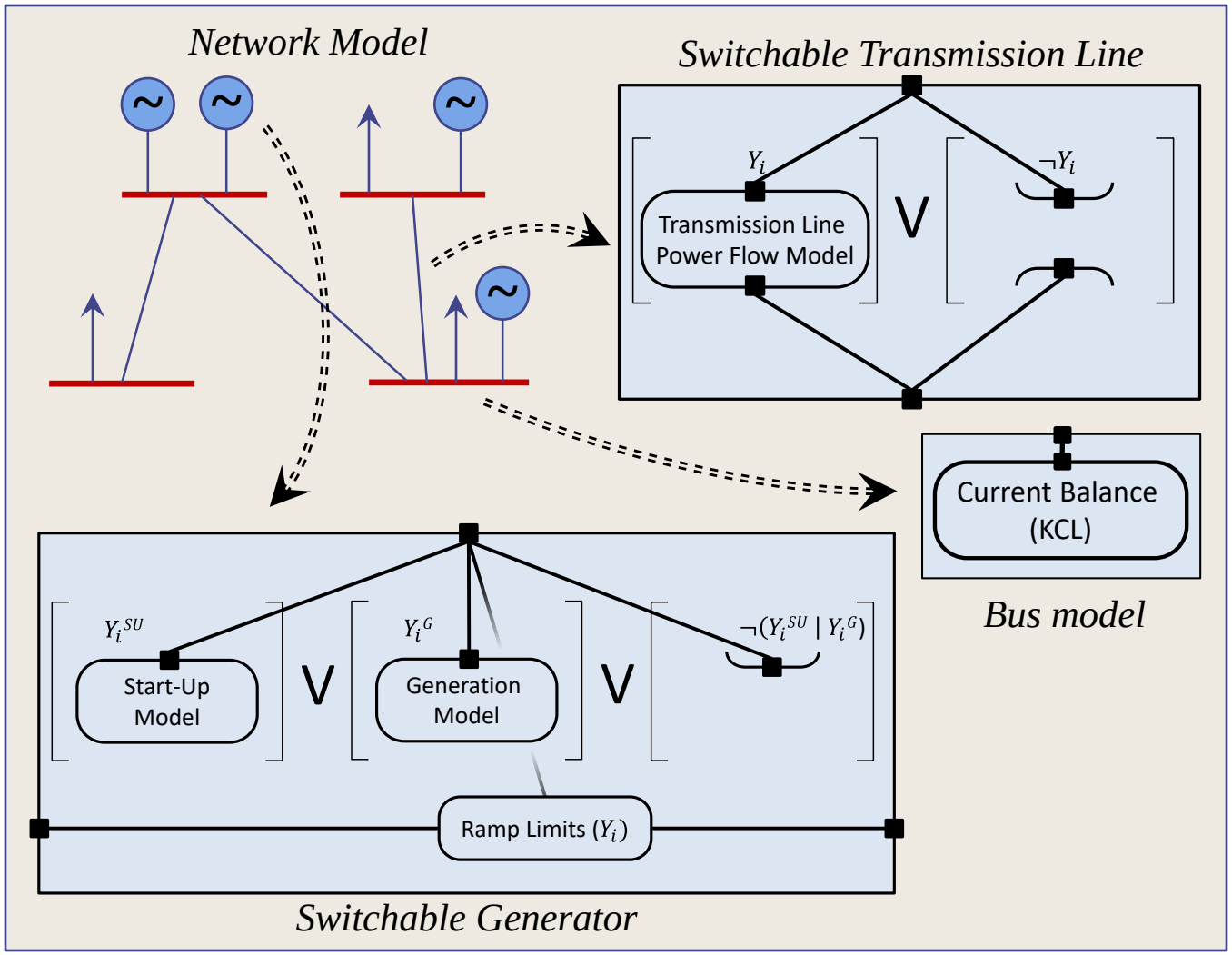
- Transmission switching:

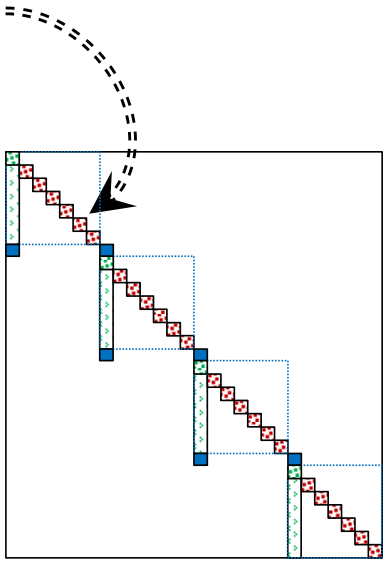
$$\begin{bmatrix} z_{kct} \\ P_{kct} = B_k(\theta_{k1} - \theta_{k2}) \end{bmatrix} \vee \begin{bmatrix} \neg z_{kct} \\ P_{kct} = 0 \end{bmatrix}$$

- Generation

$$\begin{bmatrix} u_{gt} \\ C_{gt} = P_{gt} c_g \\ R_g^+ \geq P_{gt} - P_{gt-1} \\ R_g^- \geq P_{gt-1} - P_{gt} \end{bmatrix} \vee \begin{bmatrix} v_{kt} \\ C_{gt} = P_{gt} c_g + c_g^{SU} \\ R_g^{SU} \geq P_{gt} - P_{gt-1} \end{bmatrix} \vee \begin{bmatrix} \neg(u_{kt} \mid v_{kt}) \\ C_{gt} = c_g^{SD} u_{kt-1} \\ R_g^{SD} \geq P_{gt-1} - P_{gt} \\ P_{gt} = 0 \end{bmatrix}$$

Embed within a structured model





Optimal Solution of RTS-96

- From Hedman, et al. 2010
  - N-1 UC solution: 3,245,997
  - N-1 UC w/ Switching: 3,125,185 (2 pass UC+switching heuristic)

	Rows	Columns	Binaries
Raw model	5,118,760	1,501,177	5,184
After presolve	2,634,851	1,062,290	4,476

- Restructured problem (complete N-1 UC w/ switching):

	Rows	Columns	Binaries
Raw model	21,232,224	13,129,692	3,796,830
After presolve	2,471,714	1,249,976	187,194

  - Solution (1e-4 gap): 2,990,004 (60,000 sec)
  - Automated Big-M relaxation (including automatic M calculation)
  - Default solver settings

Process modeling for power systems

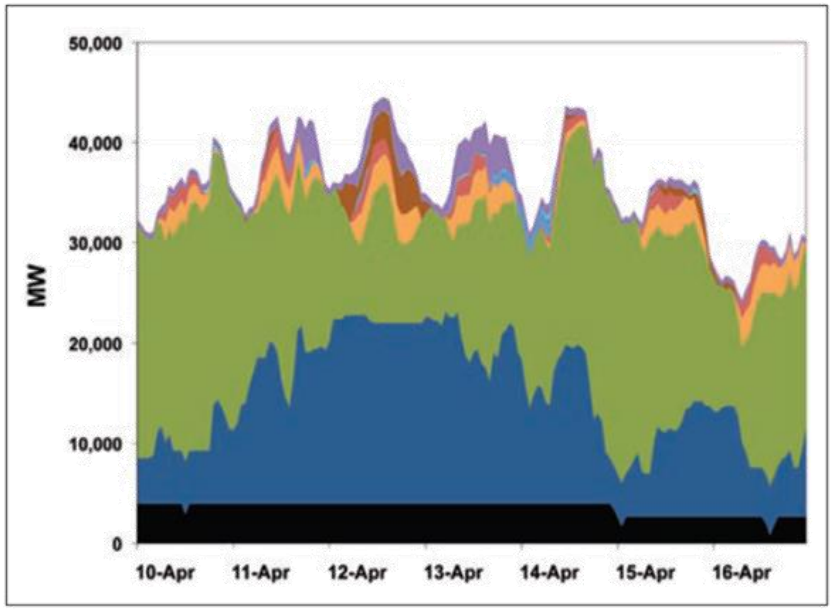
Sandia National Laboratories

CCR Center for Computing Research

Renewables impact *(chemical) processes*

- Renewables increase ramping of base-load generation
  - Market “favors” flexible generation
  - What is the impact (opportunity) for thermal plants?

- How do new energy processes (e.g. chemical looping) slot into the market?
  - Do what degree should they be designed for steady-state or dynamic operation?



The challenge of process modeling

- A tale of two paradigms
  - Sequential Modular Simulation and Equation Oriented Modeling

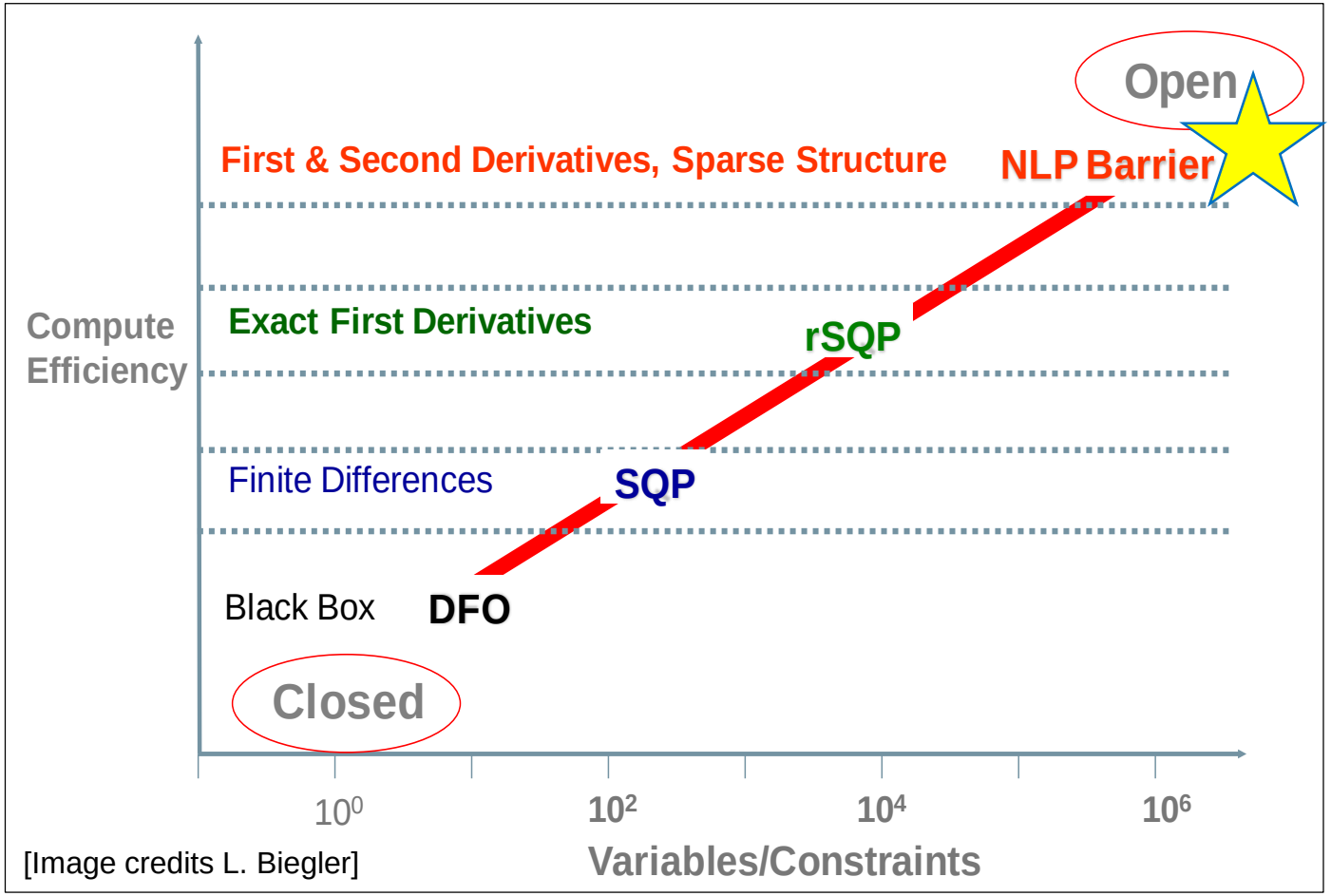
- Sequential Modular
  - ✓ Straightforward to set up
  - ✓ Straightforward simulation initialization
  - ✓ Modules may embed arbitrary sub-models
  - ✓ Robust solutions
  - Converging recycles can be challenging
  - "Arrow of time" makes inverse problems difficult:
    - Product specifications
    - Parameter estimation
    - Optimization

- Equation Oriented
  - ✓ Models are converged simultaneously, enabling
    - Product specifications
    - Path constraints
    - Parameter estimation
    - Optimization

    …even for complex processes
  - ✓ Efficient restart possible
  - ✓ Advanced, scalable algorithms available
  - Initialization challenging
  - Performance sensitive to formulation choice
  - Final models frequently complex and unintuitive.

The Goal: the "EO Utopia"

- Fully "open" models with complete structural and derivative information



[Image credits L. Biegler]

EMP for Process Development

- Key modeling needs
  - Modularity and composability
  - Continuous and discrete (logic-based) models
  - Continuous dynamics (physical systems)
  - Stochastic models / uncertainty quantification
- EMP capabilities
  - Hierarchical model definitions
  - Complementarity conditions
  - Generalized Disjunctive Programming (GDP)
  - (Discretized) systems of differential-algebraic equations (DAE)
  - Stochastic programming / design under uncertainty

Extensions to *dynamic systems*

Sandia National Laboratories

[with B. Nicholson]

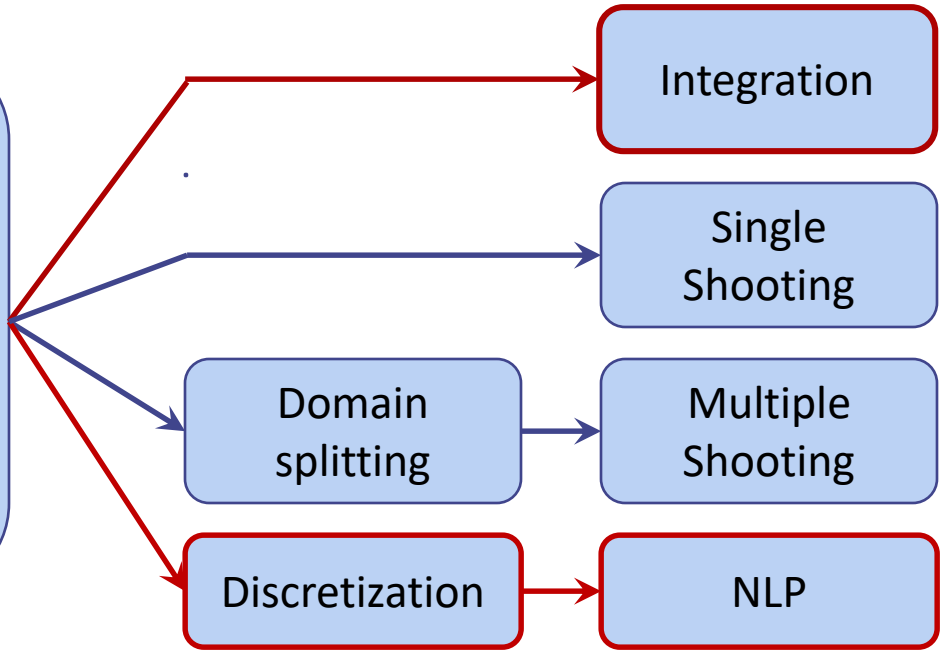
- Optimization of dynamic systems is *hard*.
  - In OR, think "multi-stage" problems
  - In "engineered systems", think differential equations
    - High fidelity *simulation* is difficult and expensive (e.g., HPC)
    - How to optimize?
      - Simulation-based optimization (*single shooting*)
      - Multiple shooting methods
      - Discretization (*collocation methods*)
    - Common theme: significant effort to rework formulation
      - Time: first ~6 months of a grad student's research
      - Error prone: many ways to make subtle mistakes
      - Inflexible: formulation specific to selected solution approach

Dynamic systems through EMP

- Model dynamical systems in a natural form
  - Systems of Differential Algebraic Equations (DAE)
  - Extend the Pyomo component model
    - `ContinuousSet`: A virtual set over which you can take a derivative
    - `DerivativeVar`: The derivative of a Var with respect to a ContinuousSet

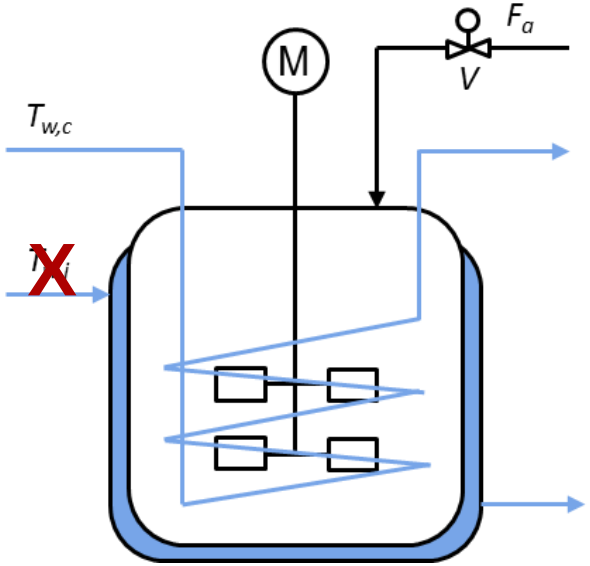
DAE model

$$\min \Psi(x(t), y(t), u(t))$$
$$\dot{x}(t) = f(x(t), y(t), u(t), t, p)$$
$$0 = g(x(t), y(t), u(t), t, p)$$
$$x(t) \in \Re^n$$
$$y(t) \in \Re^n$$
$$u(t) \in \Re^m$$



Optimal control

- Find the nominal control profiles such that the batch can be ‘saved’ given a partial cooling system failure at any point during the batch time[2]

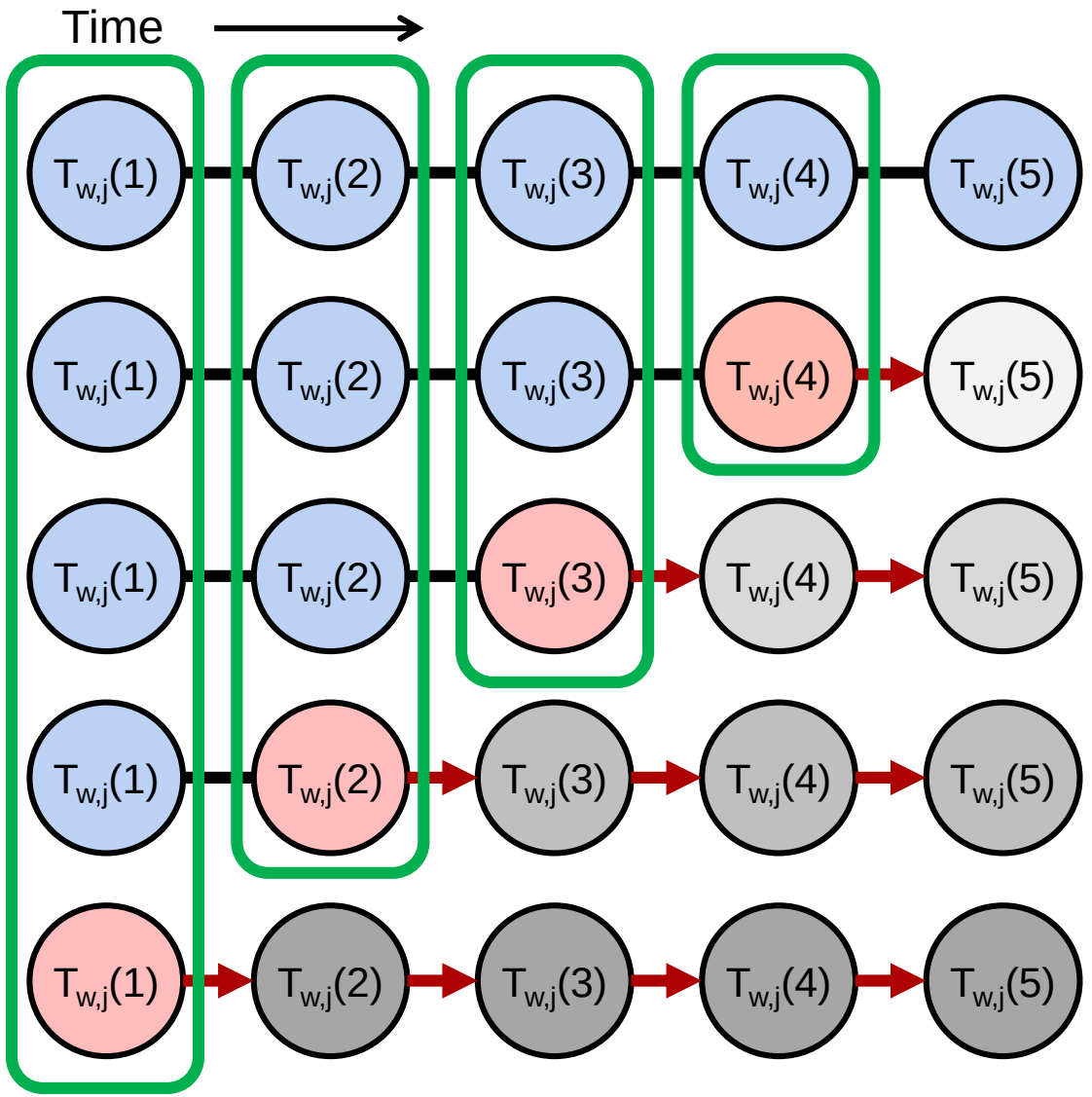


$$\left.\begin{aligned} T_{w,j}(t) &= T_{w,c}(t) && t \le t_{fail} \\ \left(\rho_w C_{p_w} V_j\right)\dot{T}_{w,j} &= \alpha_{w,js} A_j \frac{V_r}{V_{r,0}}\left(T_{w,j} - T_r\right) && t > t_{fail} \end{aligned}\right\}$$

[2] Abel and Marquardt, AIChE Journal (2000)

Optimal control

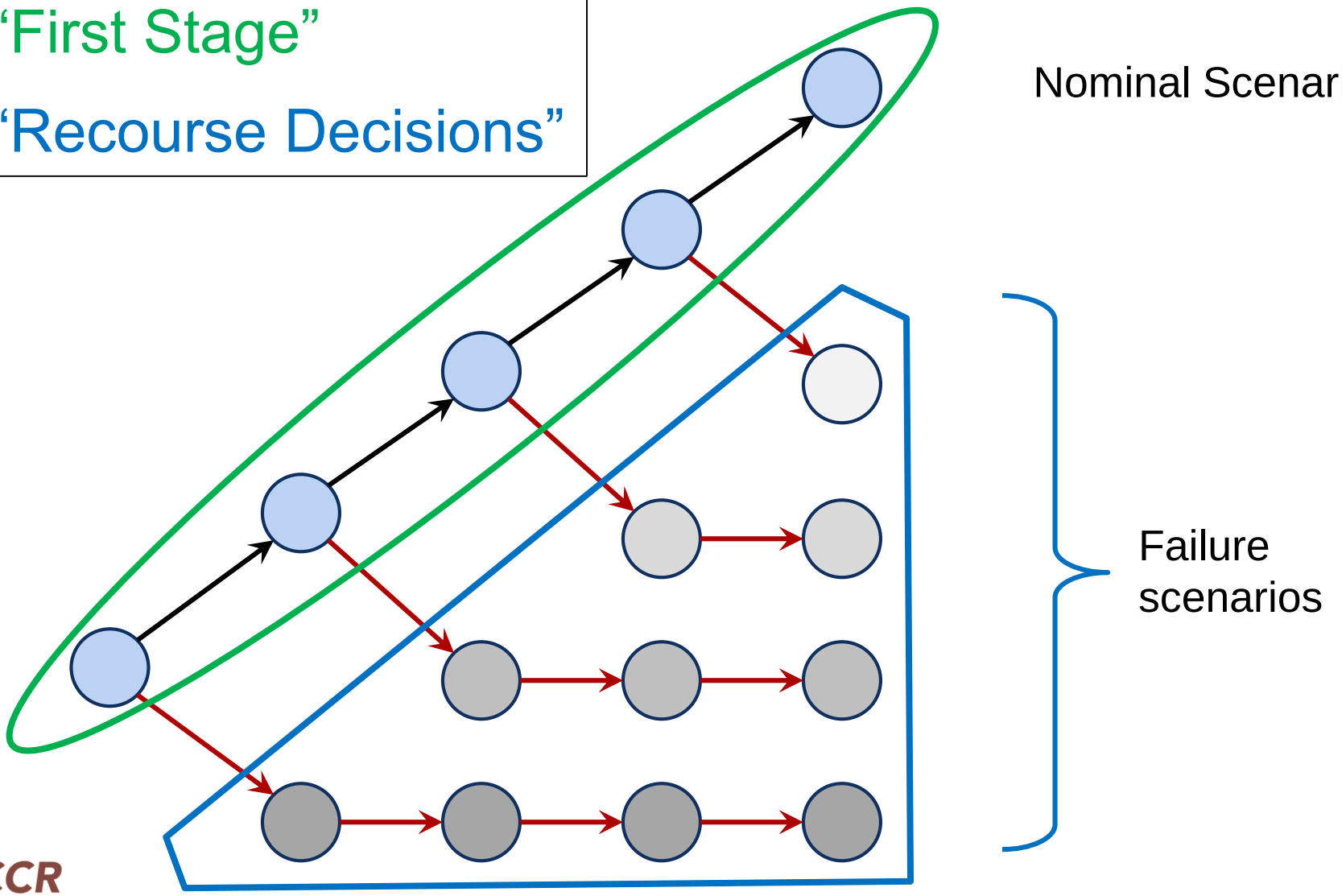
Nonanticipativity Constraints



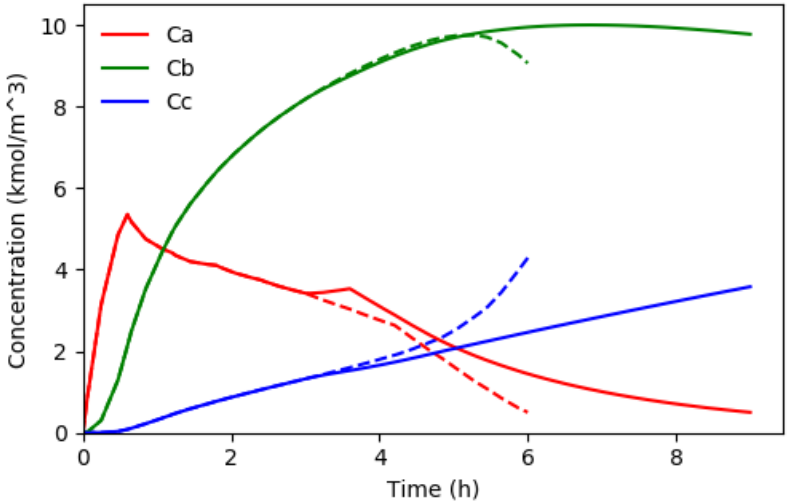
n-Stage SP as a 2-stage problem

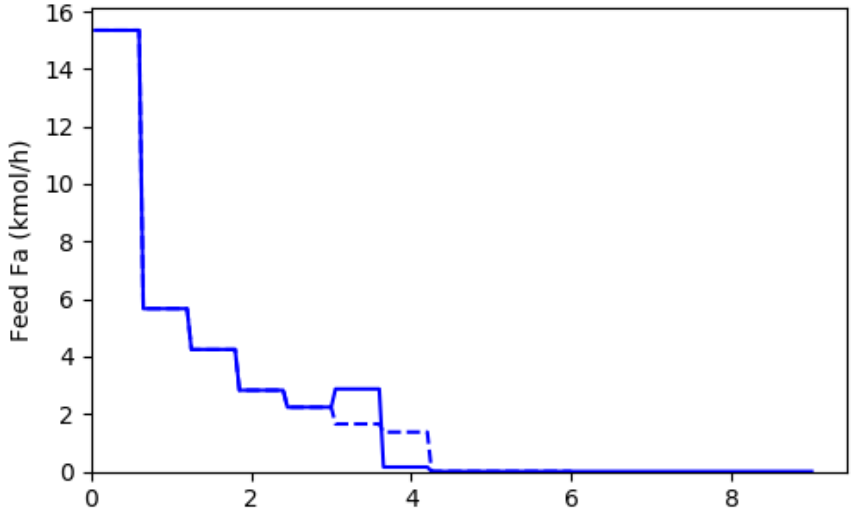
"First Stage"

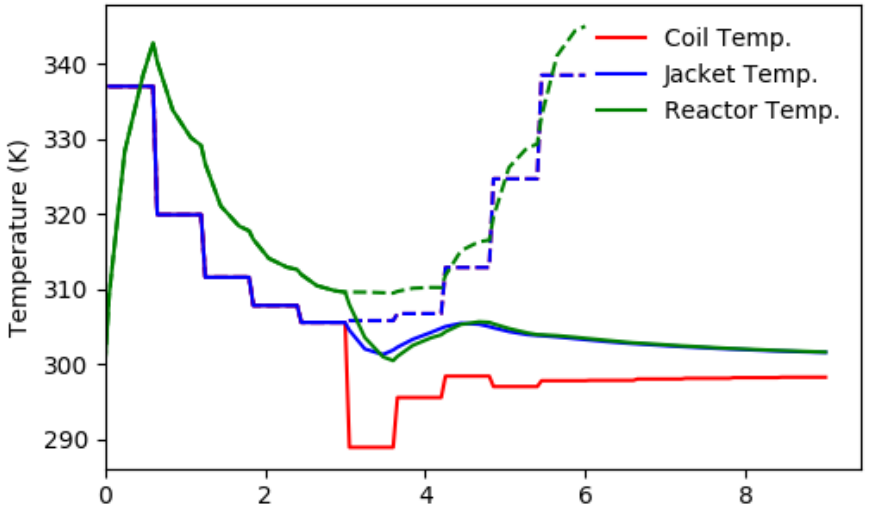
"Recourse Decisions"



Optimal control: t_{fail} = 3 h

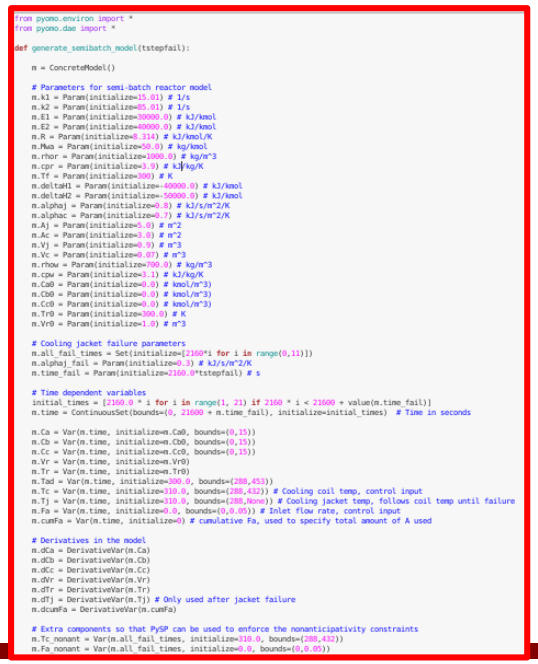




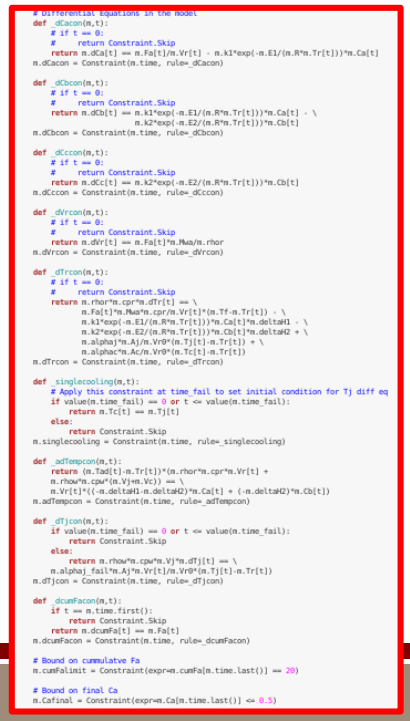


Optimal control implementation

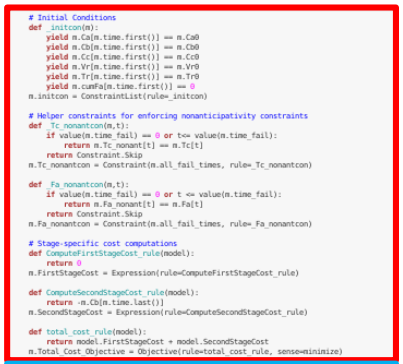
- 15 hours (including debugging)
- 300 lines of code
  - (60%)Deterministic dynamic model specification
  - (2%) Discretization
  - (18%) Stochastic problem formulation
  - (20%) Result plotting



```
from pyomo.environ import *
from pyomo.dae import *

def generate_semibatch_model(tstepfail):

    m = ConcreteModel()

    # Parameters for semi-batch reactor model
    m.k1 = Param(initialize=15.01) # 1/s
    m.k2 = Param(initialize=85.01) # 1/s
    m.E1 = Param(initialize=30000.0) # kJ/kmol
    m.E2 = Param(initialize=40000.0) # kJ/kmol
    m.R = Param(initialize=8.314) # kJ/kmol/K
    m.Mwa = Param(initialize=50.0) # kg/kmol
    m.rhor = Param(initialize=1000.0) # kg/m^3
    m.cpr = Param(initialize=4.19) # kJ/kg/K
    m.Tf = Param(initialize=300) # K
    m.deltaH1 = Param(initialize=-40000.0) # kJ/kmol
    m.deltaH2 = Param(initialize=-50000.0) # kJ/kmol
    m.alphaj = Param(initialize=0.8) # kJ/s/m^2/K
    m.alphac = Param(initialize=0.7) # kJ/s/m^2/K
    m.Aj = Param(initialize=5.0) # m^2
    m.Ac = Param(initialize=5.0) # m^2
    m.Vj = Param(initialize=0.9) # m^3
    m.Vc = Param(initialize=0.07) # m^3
    m.rhow = Param(initialize=700.0) # kg/m^3
    m.cpw = Param(initialize=3.1) # kJ/kg/K
    m.Ca0 = Param(initialize=0.0) # kmol/m^3)
    m.Cb0 = Param(initialize=0.0) # kmol/m^3)
    m.Cc0 = Param(initialize=0.0) # kmol/m^3)
    m.Tr0 = Param(initialize=300.0) # K
    m.Vr0 = Param(initialize=1.0) # m^3

    # Cooling jacket failure parameters
    m.all_fail_times = Set(initialize=[2160*i for i in range(0,11)])
    m.alphaj_fail = Param(initialize=0.3) # kJ/s/m^2/K
    m.time_fail = Param(initialize=2160.0*tstepfail) # s

    # Time dependent variables
    initial_times = [2160.0 * i for i in range(1, 21) if 2160 * i < 21600 + value(m.time_fail)]
    m.time = ContinuousSet(bounds=(0, 21600 + m.time_fail), initialize=initial_times)  # Time in seconds

    m.Ca = Var(m.time, initialize=m.Ca0, bounds=(0,15))
    m.Cb = Var(m.time, initialize=m.Cb0, bounds=(0,15))
    m.Cc = Var(m.time, initialize=m.Cc0, bounds=(0,15))
    m.Vr = Var(m.time, initialize=m.Vr0)
    m.Tr = Var(m.time, initialize=m.Tr0)
    m.Tad = Var(m.time, initialize=300.0, bounds=(288,453))
    m.Tc = Var(m.time, initialize=310.0, bounds=(288,432)) # Cooling coil temp, control input
    m.Tj = Var(m.time, initialize=310.0, bounds=(288,None)) # Cooling jacket temp, follows coil temp until failure
    m.Fa = Var(m.time, initialize=0.0, bounds=(0,0.05)) # Inlet flow rate, control input
    m.cumFa = Var(m.time, initialize=0) # cumulative Fa, used to specify total amount of A used

    # Derivatives in the model
    m.dCa = DerivativeVar(m.Ca)
    m.dCb = DerivativeVar(m.Cb)
    m.dCc = DerivativeVar(m.Cc)
    m.dVr = DerivativeVar(m.Vr)
    m.dTr = DerivativeVar(m.Tr)
    m.dTj = DerivativeVar(m.Tj) # Only used after jacket failure
    m.dcumFa = DerivativeVar(m.cumFa)

    # Extra components so that PySP can be used to enforce the nonanticipativity constraints
    m.Tc_nonant = Var(m.all_fail_times, initialize=310.0, bounds=(288,432))
    m.Fa_nonant = Var(m.all_fail_times, initialize=0.0, bounds=(0,0.05))
```



```
# Differential Equations in the model
def _dCacon(m,t):
    # if t == 0:
    #     return Constraint.Skip
    return m.dCa[t] == m.Fa[t]/m.Vr[t] - m.k1*exp(-m.E1/(m.R*m.Tr[t]))*m.Ca[t]
m.dCacon = Constraint(m.time, rule=_dCacon)

def _dCbcon(m,t):
    # if t == 0:
    #     return Constraint.Skip
    return m.dCb[t] == m.k1*exp(-m.E1/(m.R*m.Tr[t]))*m.Ca[t] - \
                       m.k2*exp(-m.E2/(m.R*m.Tr[t]))*m.Cb[t]
m.dCbcon = Constraint(m.time, rule=_dCbcon)

def _dCccon(m,t):
    # if t == 0:
    #     return Constraint.Skip
    return m.dCc[t] == m.k2*exp(-m.E2/(m.R*m.Tr[t]))*m.Cb[t]
m.dCccon = Constraint(m.time, rule=_dCccon)

def _dVrcon(m,t):
    # if t == 0:
    #     return Constraint.Skip
    return m.dVr[t] == m.Fa[t]*m.Mwa/m.rhor
m.dVrcon = Constraint(m.time, rule=_dVrcon)

def _dTrcon(m,t):
    if t == 0:
        return Constraint.Skip
    return m.rhor*m.cpr*m.dTr[t] == \
           (m.Fa[t]*m.Mwa*m.cpr/m.Vr[t]*(m.Tf-m.Tr[t]) - \
            m.k1*exp(-m.E1/(m.R*m.Tr[t]))*m.Ca[t]*m.deltaH1 - \
            m.k2*exp(-m.E2/(m.R*m.Tr[t]))*m.Cb[t]*m.deltaH2 + \
            m.alphaj*m.Aj/m.Vr0*(m.Tj[t]-m.Tr[t]) + \
            m.alphac*m.Ac/m.Vr0*(m.Tc[t]-m.Tr[t]))
m.dTrcon = Constraint(m.time, rule=_dTrcon)

def _singlecooling(m,t):
    # Apply this constraint at time_fail to set initial condition for Tj diff eq
    if value(m.time_fail) == 0 or t <= value(m.time_fail):
        return m.Tc[t] == m.Tj[t]
    else:
        return Constraint.Skip
m.singlecooling = Constraint(m.time, rule=_singlecooling)

def _adTempcon(m,t):
    return (m.Tad[t]-m.Tr[t])*(m.rhor*m.cpr*m.Vr[t] +
    m.rhow*m.cpw*(m.Vj+m.Vc)) == \
    m.Vr[t]*((-m.deltaH1-m.deltaH2)*m.Ca[t] + (-m.deltaH2)*m.Cb[t])
m.adTempcon = Constraint(m.time, rule=_adTempcon)

def _dTjcon(m,t):
    if value(m.time_fail) == 0 or t <= value(m.time_fail):
        return Constraint.Skip
    else:
        return m.rhow*m.cpw*m.Vj*m.dTj[t] == \
    m.alphaj_fail*m.Aj*m.Vr[t]/m.Vr0*(m.Tj[t]-m.Tr[t])
m.dTjcon = Constraint(m.time, rule=_dTjcon)

def _dcumFacon(m,t):
    if t == m.time.first():
        return Constraint.Skip
    return m.dcumFa[t] == m.Fa[t]
m.dcumFacon = Constraint(m.time, rule=_dcumFacon)

# Bound on cumulatve Fa
m.cumFalimit = Constraint(expr=m.cumFa[m.time.last()] <= 20)

# Bound on final Ca
m.Cafinal = Constraint(expr=m.Ca[m.time.last()] <= 0.5)
```



```
# Initial Conditions
def _initcon(m):
    yield m.Ca[m.time.first()] == m.Ca0
    yield m.Cb[m.time.first()] == m.Cb0
    yield m.Cc[m.time.first()] == m.Cc0
    yield m.Vr[m.time.first()] == m.Vr0
    yield m.Tr[m.time.first()] == m.Tr0
    yield m.cumFa[m.time.first()] == 0
m.initcon = ConstraintList(rule=_initcon)

# Helper constraints for enforcing nonanticipativity constraints
def _Tc_nonantcon(m,t):
    if value(m.time_fail) == 0 or t<= value(m.time_fail):
        return m.Tc_nonant[t] == m.Tc[t]
    return Constraint.Skip
m.Tc_nonantcon = Constraint(m.all_fail_times, rule=_Tc_nonantcon)

def _Fa_nonantcon(m,t):
    if value(m.time_fail) == 0 or t <= value(m.time_fail):
        return m.Fa_nonant[t] == m.Fa[t]
    return Constraint.Skip
m.Fa_nonantcon = Constraint(m.all_fail_times, rule=_Fa_nonantcon)

# Stage-specific cost computations
def ComputeFirstStageCost_rule(model):
    return 0
m.FirstStageCost = Expression(rule=ComputeFirstStageCost_rule)

def ComputeSecondStageCost_rule(model):
    return -m.Cb[m.time.last()]
m.SecondStageCost = Expression(rule=ComputeSecondStageCost_rule)

def total_cost_rule(model):
    return model.FirstStageCost + model.SecondStageCost
m.Total_Cost_Objective = Objective(rule=total_cost_rule, sense=minimize)
```

```
# Discretize model. Number of finite elements depends on the cooling failure time
numfe = m.time.last()/2160.0
disc = TransformationFactory('dae.collocation')
disc.apply_to(m, nfe=numfe, ncp=4)
disc.reduce_collocation_points(m, var=m.Fa, ncp=1, contset=m.time)
disc.reduce_collocation_points(m, var=m.Tc, ncp=1, contset=m.time)
return m
```

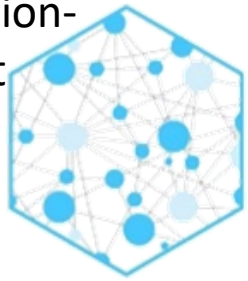
```
# Number of scenarios
scenarios = 10

def pysp_scenario_tree_model_callback():
    from pyomo.pysp.scenariotree.tree_structure_model \
        import CreateConcreteTwoStageScenarioTreeModel

    st_model = CreateConcreteTwoStageScenarioTreeModel(scenarios)

    first_stage = st_model.Stages.first()
    second_stage = st_model.Stages.last()

    # First Stage
    st_model.StageCost[first_stage] = 'FirstStageCost'
    st_model.StageVariables[first_stage].add('Fa_nonant[*]')
    st_model.StageVariables[first_stage].add('Tc_nonant[*]')

    # Second Stage
    st_model.StageCost[second_stage] = 'SecondStageCost'

    return st_model

def pysp_instance_creation_callback(scenario_name, node_names):
    failtimestep = int(scenario_name.replace('Scenario',''))
    instance = generate_semibatch_model(failtimestep-1)

    return instance
```

```
import os
from pyomo.environ import *
from pyomo.pysp.scenariotree.manager import \
    ScenarioTreeManagerClientSerial
from pyomo.pysp.ef import create_ef_instance

thisdir = os.path.dirname(os.path.abspath(__file__))

options = ScenarioTreeManagerClientSerial.register_options()

options.model_location = os.path.join(thisdir, 'semibatch.py')

manager = ScenarioTreeManagerClientSerial(options)
manager.initialize()

ef_instance = create_ef_instance(manager.scenario_tree, verbose_output=True)
print('Created EF instance')
solver = SolverFactory('ipopt')
```

```
solver.solve(ef_instance, tee=True)

nom = ef_instance.Scenario1
m = ef_instance.Scenario6

nomtime = [i/3600.0 for i in nom.time]
time = [i/3600.0 for i in m.time]

import matplotlib.pyplot as plt

grey1 = '0.7'
grey2 = '0.4'
grey3 = '0'

plt.subplot(313)
plt.plot(nomtime,[value(nom.Ca[t]) for t in nom.time],'--',color=grey1)
plt.plot(nomtime,[value(nom.Cb[t]) for t in nom.time],'--',color=grey2)
plt.plot(nomtime,[value(nom.Cc[t]) for t in nom.time],'--',color=grey3)
plt.plot(time,[value(m.Ca[t]) for t in m.time],label='Ca',color=grey1)
plt.plot(time,[value(m.Cb[t]) for t in m.time],label='Cb',color=grey2)
plt.plot(time,[value(m.Cc[t]) for t in m.time],label='Cc',color=grey3)
plt.legend(loc='best')
plt.xlabel('Time (h)')
plt.ylabel('Concentration (kmol/m^3)')
plt.xlim(xmin=0)
plt.ylim(ymin=0)

plt.subplot(311)
plt.plot(time[1:],[value(m.Fa[t])*3600 for t in m.time][1:],color=grey2)
plt.plot(nomtime[1:],[value(nom.Fa[t])*3600 for t in nom.time][1:],'--',color=grey2)
plt.legend(loc='best')
plt.ylabel('Feed Fa (kmol/h)')
plt.xlim(xmin=0)
plt.ylim(ymin=0)

plt.subplot(312)
plt.plot(nomtime[1:],[value(nom.Tc[t]) for t in nom.time][1:],'--',color=grey1)
plt.plot(nomtime[1:],[value(nom.Tj[t]) for t in nom.time][1:],'--',color=grey2)
plt.plot(nomtime,[value(nom.Tr[t]) for t in nom.time],'--',color=grey3)
plt.plot(time[1:],[value(m.Tc[t]) for t in m.time][1:],label='Coil Temp.',color=grey1)
plt.plot(time[1:],[value(m.Tj[t]) for t in m.time][1:],label='Jacket Temp.',color=grey2)
plt.plot(time,[value(m.Tr[t]) for t in m.time],label='Reactor Temp.',color=grey3)
plt.legend(loc='best')
plt.ylabel('Temperature (K)')
plt.xlim(xmin=0)
plt.show()
```

What are we doing now?

- Challenge: Develop and utilize multi-scale, simulation-based. computational tools and models to support the design, analysis, optimization, scale-up, operation and troubleshooting of innovative, advanced fossil energy systems



IDAES
Institute for the Design of Advanced Energy Systems

- Next generation modeling and optimization platform
  - Current tools insufficient to address demands of integrated systems.
  - Need a more flexible and open modeling environment
- Key capabilities
  - Process Synthesis, Integration, and Intensification
  - Process Design and Optimization
  - Process Control and Dynamics
  - Supports advanced solvers and computer architecture
  - Multi-scale modeling capabilities
  - Comprehensive, end-to-end uncertainty quantification
  - Complete provenance information
  - Couple with energy market models
  - Open source

NATIONAL ENERGY TECHNOLOGY LABORATORY

Sandia National Laboratories

BERKELEY LAB

Carnegie Mellon

WestVirginiaUniversity

CCR Center for Computing Research

IDAES Software Environment

Sandia National Laboratories

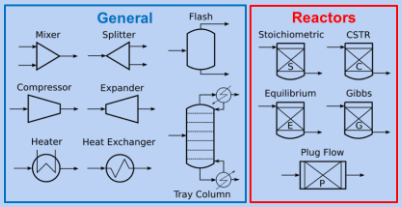
Machine Learning & Parameter Estimation for Physical Properties, Thermodynamics and Kinetics

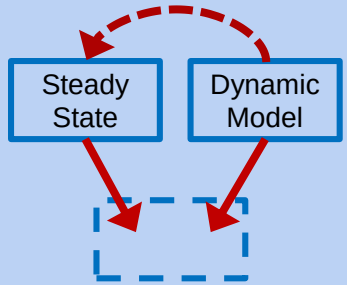
ALAMO a black-box modeling tool

RIPE

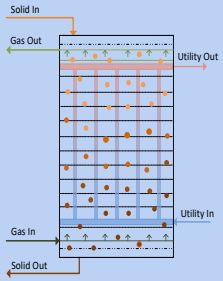
PySP

Hierarchical Process Model Library





Model Customization

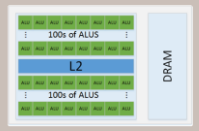


PYOMO

PYOMO DAE

Algebraic Modeling Language

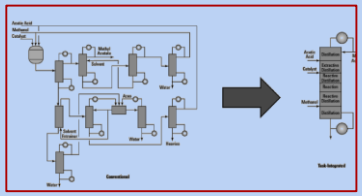
GPU



Solvers and Computational Platforms

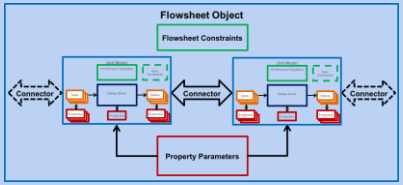
NERSC

Conceptual Design via Superstructure

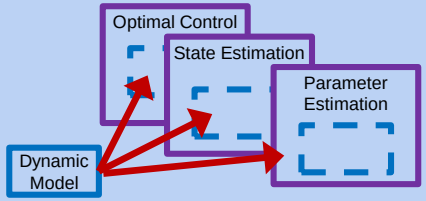


Pyomo.GDP

Process Design & Optimization Process Integration

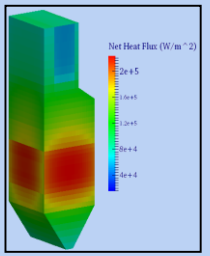


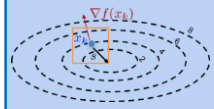
Dynamics & Control

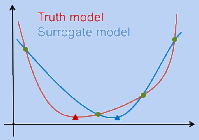


Multi-Scale Surrogate Modeling & Optimization

ALAMO a black-box modeling tool



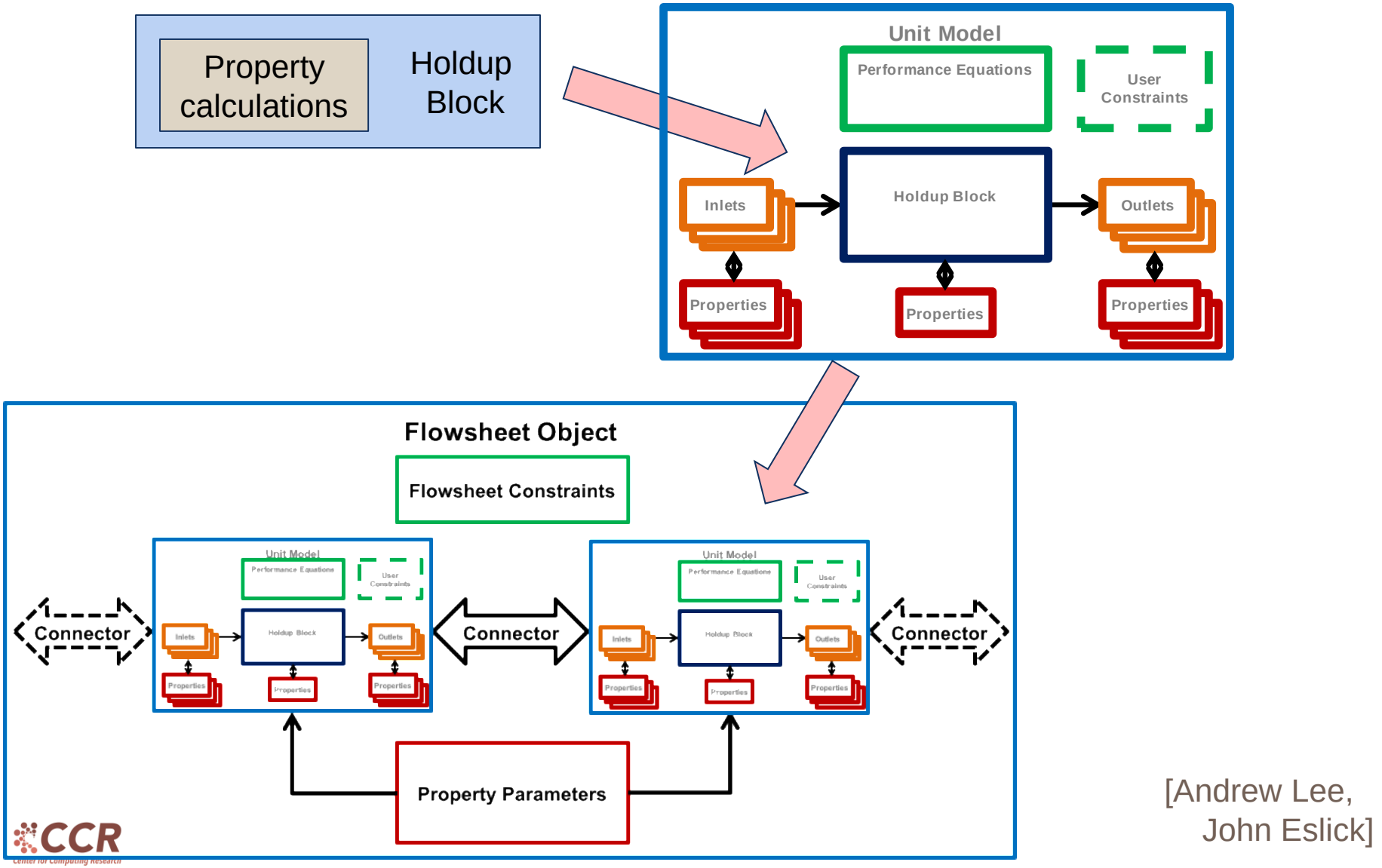




Incorporation and Assessment of Uncertainty Across Models/Scales PySP

CCR Center for Computing Research

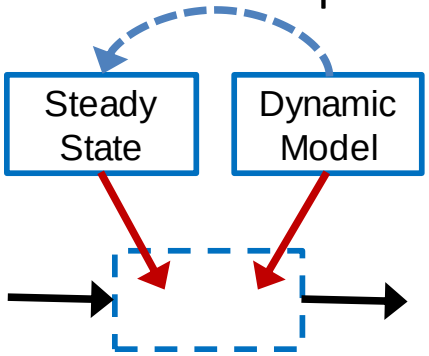
Block-oriented process modeling

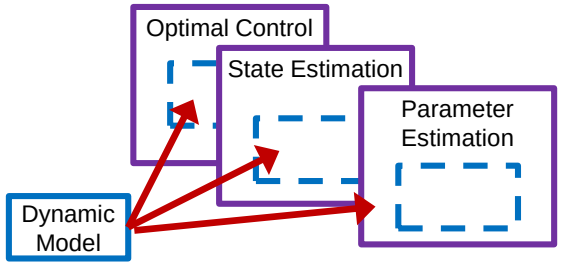


[Andrew Lee, John Eslick]

Extend to *Dynamics* and *Design*

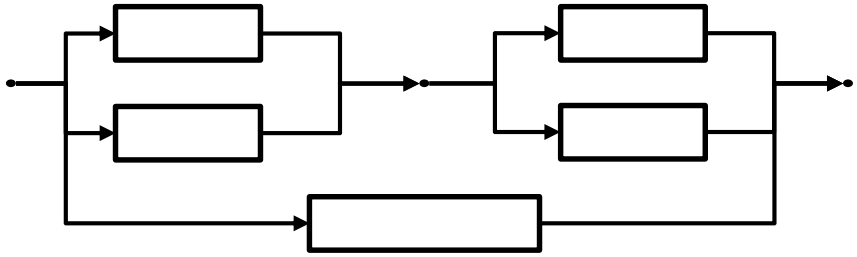
- Common *block-oriented* process library
  - Convertible to steady state or dynamic models through user options
  - Relies on the same core modeling engine
  - Applicable to multiple analysis domains





[B. Nicholson, T. Burgard, A. Lee, J. Eslick]

  - Wrapping process blocks in *disjunctions* enables the generation and evaluation of process superstructures



[Q. Chen, I. Grossmann]

Summary

- Power systems is a rich area for modeling, algorithmic, and application research (especially for Chemical Engineers!)
  - Stochastic unit commitment
  - Reliability and resiliency analysts
  - Expansion planning
  - Process development
  - Dynamic modeling and control
- Equation-oriented optimization is undergoing a Renaissance
  - New modeling languages and approaches
  - "Extended Math Programming" to bridge the gap between solvers and modelers/application domains
- Lower barrier to entry through Open Source Tools
  - Pyomo: www.pyomo.org / https://github.com/Pyomo/pyomo
  - IDAES: www.idaes.org / https://github.com/IDAES
    - Initial limited release planned for July, public release in December

Thank you!

- For more information…
- Project homepages
  - http://www.pyomo.org
  - http://software.sandia.gov/pyomo
- User mailing lists
  - pyomo-forum@googlegroups.com
- "The Book"

- Mathematical Programming Computation 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)