Progressive Hedging and Asynchronous Projective Hedging for Convex Stochastic Programming

February 22, 2021

RUTGERS

RUTGERS
Rutgers Business School
Newark and New Brunswick

Portions of this work joint with

Patrick Combettes, North Carolina State University
Jean-Paul Watson, Lawrence Livermore National Lab
David L. Woodruff, University of California, Davis

Typical ADMM and Operator Splitting Applications

- The most prominent applications of operator splitting and ADMM-class algorithms are in machine learning and image processing
- There has not been much operator splitting work on “OR-style” optimization problems
- With one exception:

Stochastic Programming

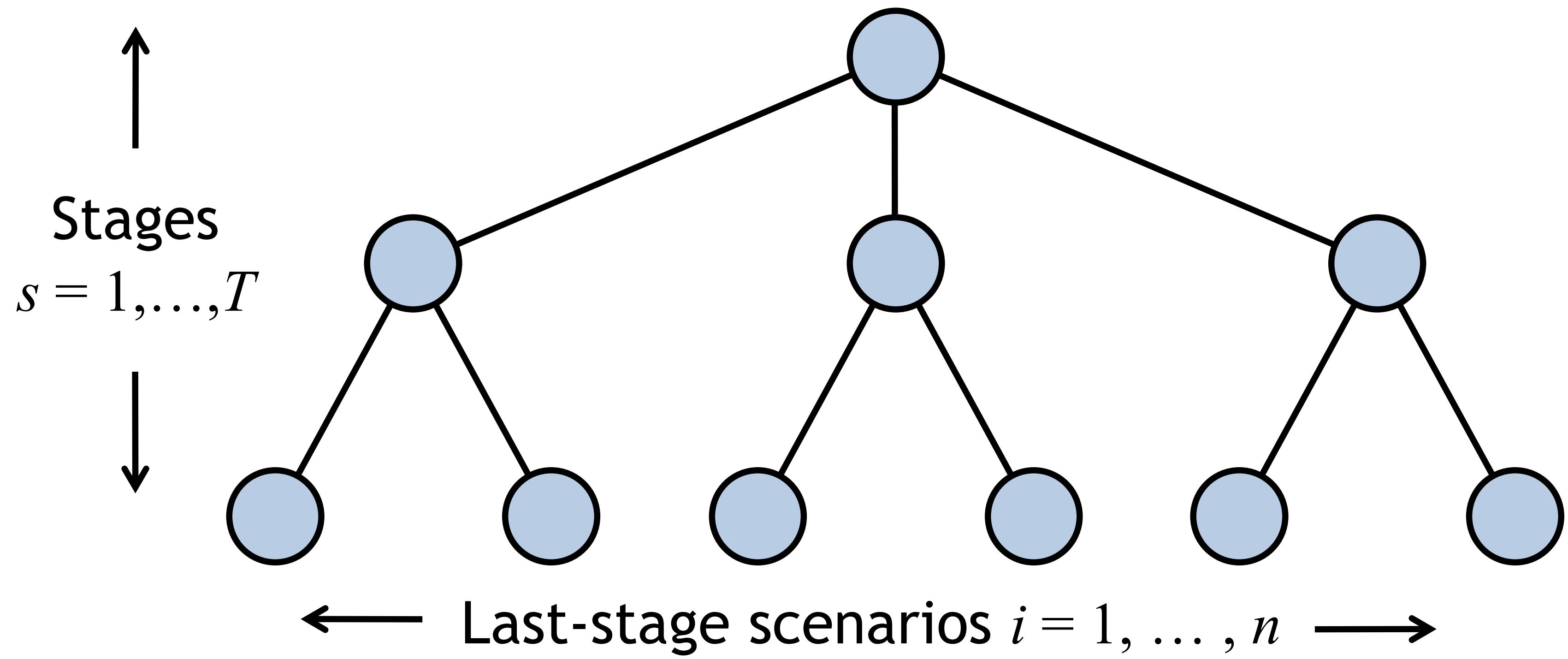
- Solving LP etc. models on an unfolding tree of random future scenarios
- Rockafellar and Wet’s *progressive hedging* algorithm (1991)

Progressive Hedging

- Working paper in late 1987, published in *Mathematics of Operations Research* in 1991
- Rockafellar and Wets knew that their method was a form of DR splitting / ADMM algorithm
- But proved its convergence from first principles
- After all, monotone operators and the ADMM were not much known in the OR community at the time
- I will present the method from an ADMM point of view, then switch to projective splitting

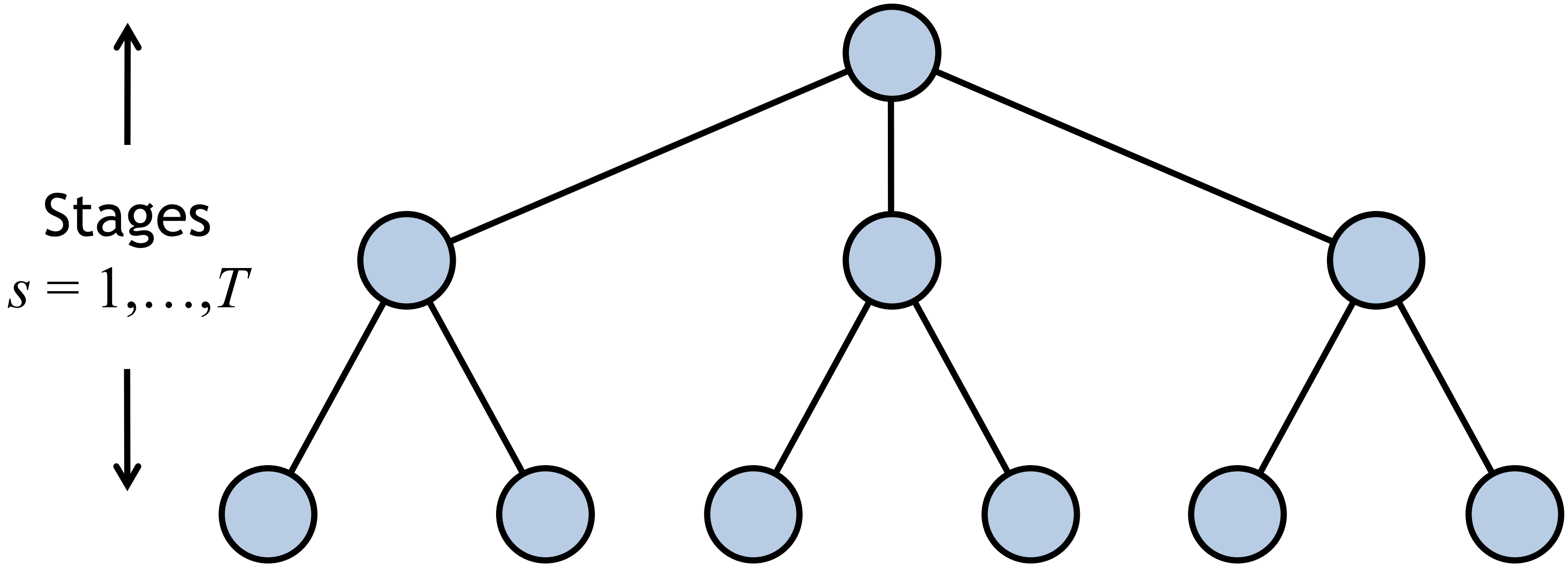
The Scenario Tree

- Consider a standard stochastic programming scenario tree:



- π_i is the probability of last-stage scenario $i = 1, \ldots, n$
- Will use “scenario” as a shorthand for “last-stage scenario”
- Typically a discrete-time and sampled approximation of some infinite or much larger model

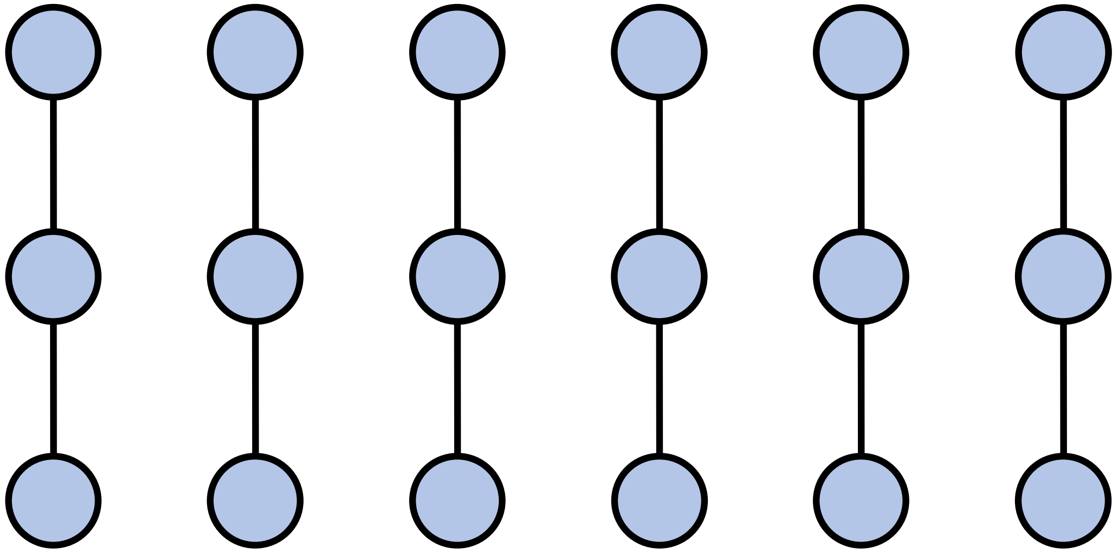
Stochastic Programming



- System walks randomly from the root to some leaf
- At each node there are decision variables, for example
  - How much of an investment to buy or sell
  - How much to run a power generator, etc...
- ... and constraints that depend on earlier decisions
- Model alternates decisions and uncertainty resolution

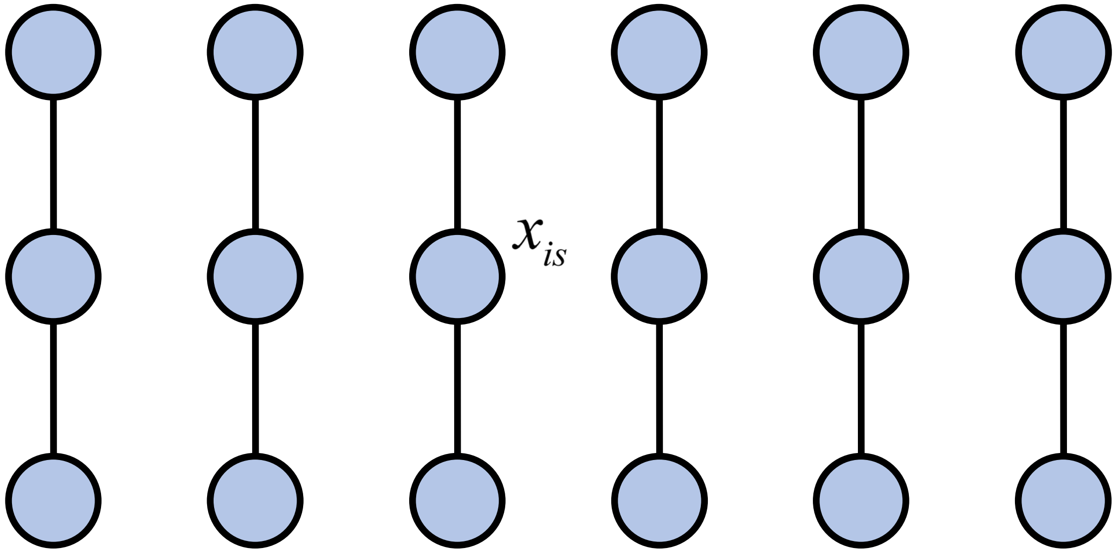
Problem Formulation and Notation

- Replicate decision variables: n copies at every stage



Problem Formulation and Notation

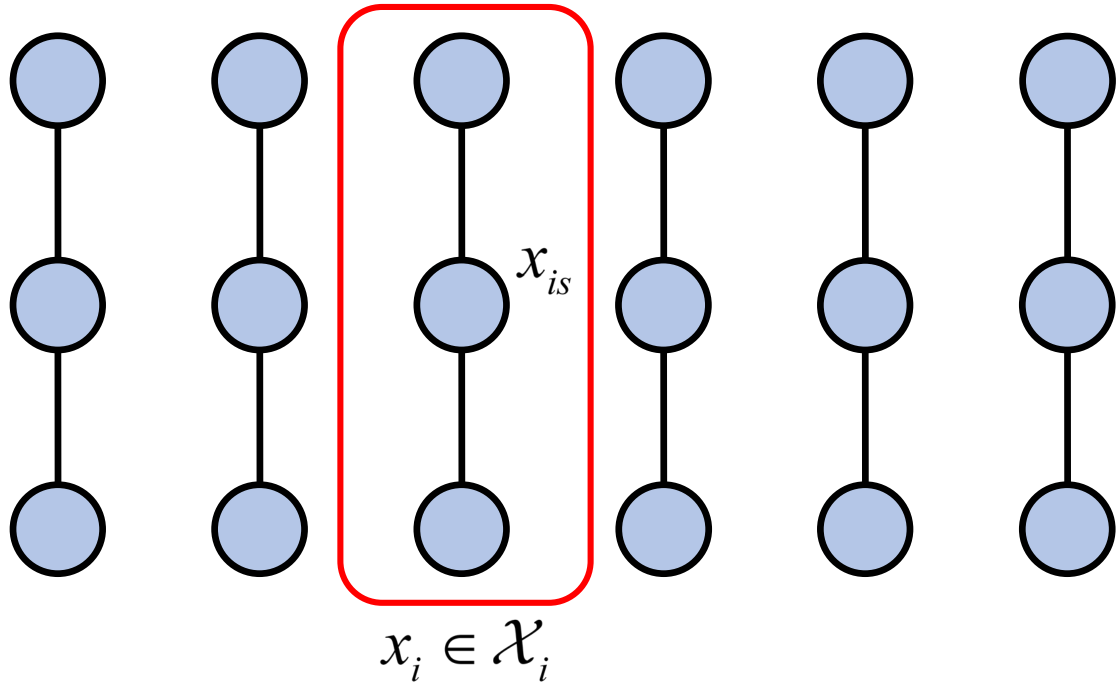
- Replicate decision variables: n copies at every stage



- x_{is} is the vector of decision variables for scenario i at stage s

Problem Formulation and Notation

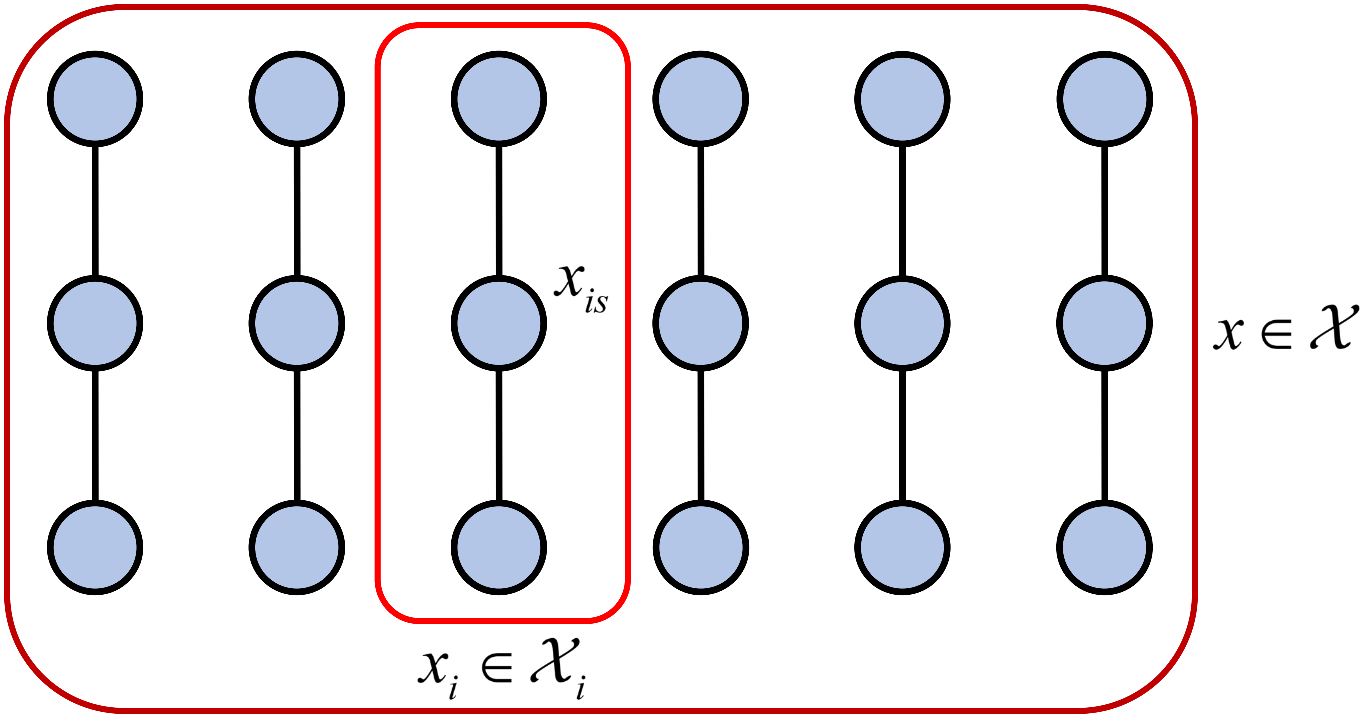
- Replicate decision variables: n copies at every stage



- x_{is} is the vector of decision variables for scenario i at stage s
- $\mathcal{X}_i$ is the space of all variables pertaining to scenario i ; elements are $x_i = (x_{i1}, \ldots, x_{iT})$

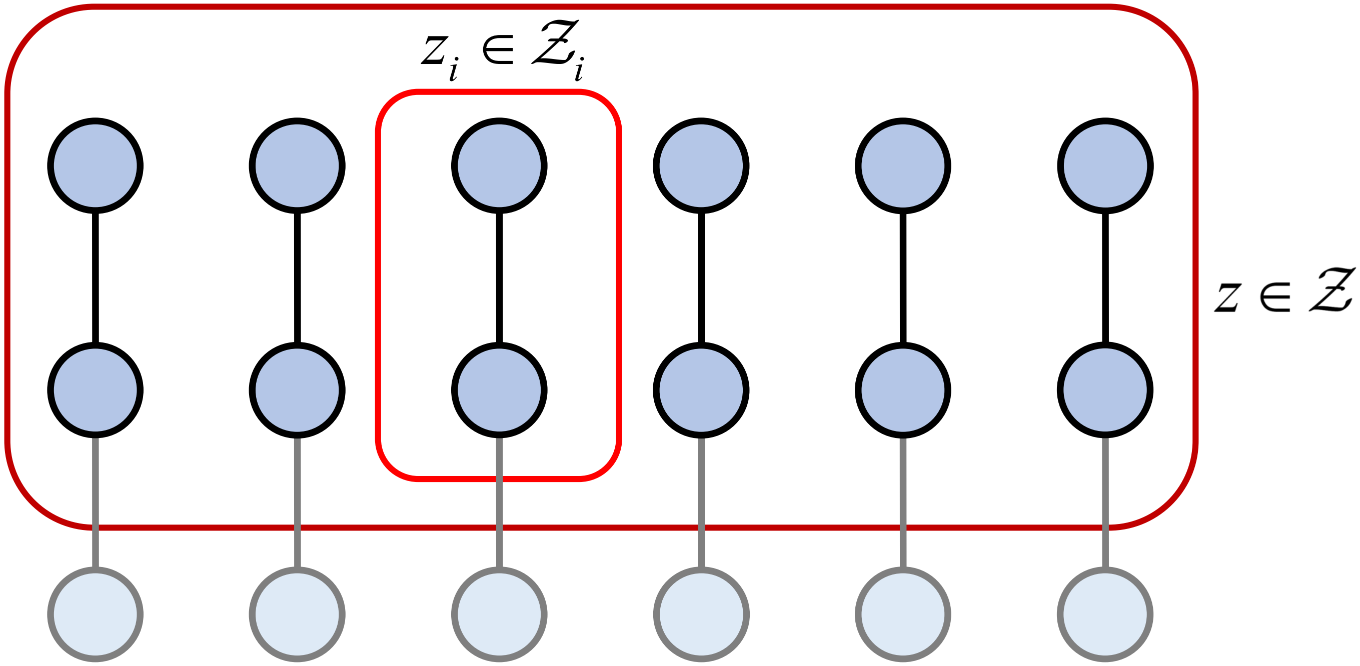
Problem Formulation and Notation

- Replicate decision variables: n copies at every stage



- x_{is} is the vector of decision variables for scenario i at stage s
- $\mathcal{X}_i$ is the space of all variables for scenario i; elements are
  $x_i = (x_{i1}, \ldots, x_{iT})$
- $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$ is space of all decision variables; elements are
  $x = (x_1, \ldots, x_n) = \big((x_{11}, \ldots, x_{1T}), \ldots, (x_{n1}, \ldots, x_{nT})\big)$

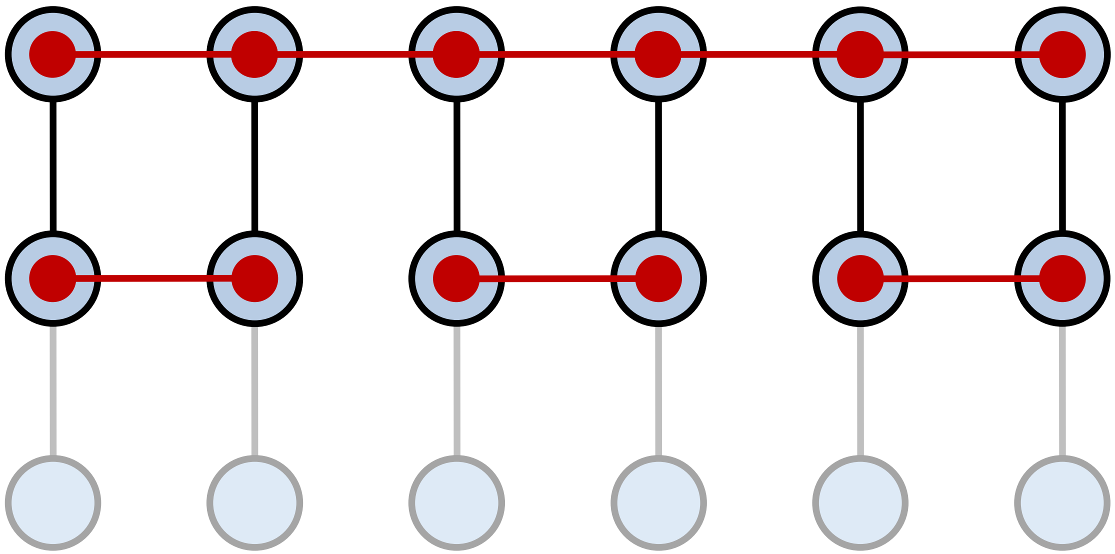
Problem Formulation and Notation

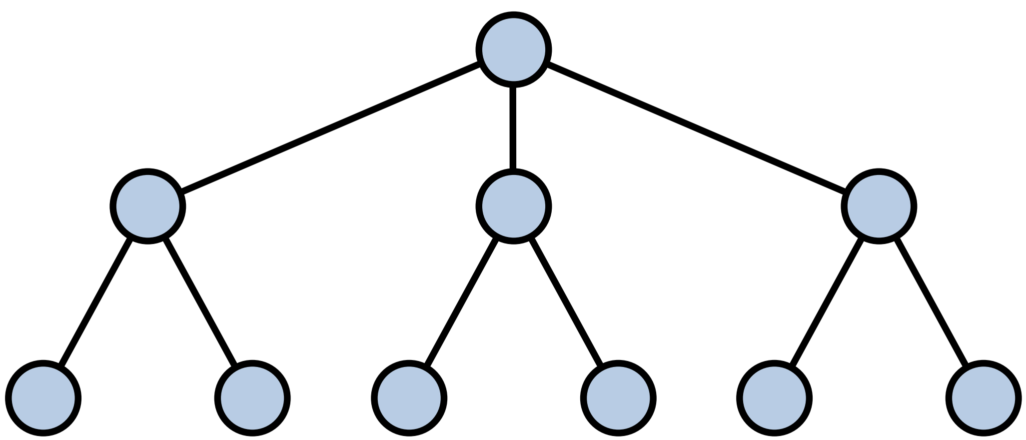


- $\mathcal{Z}_i$ is $\mathcal{X}_i$ without the last stage; elements $z_i = (z_{i1}, \ldots, z_{i,T-1})$
- $\mathcal{Z} = \mathcal{Z}_1 \times \cdots \times \mathcal{Z}_n$ is the space of all variables except the last stage: elements $z = (z_1, \ldots, z_n) = \big((z_{11}, \ldots, z_{1,T-1}), \ldots, (z_{n1}, \ldots, z_{n,T-1})\big)$

Nonanticipativity Subspace

- $\mathcal{N} \subset \mathcal{Z}$ is the subspace of $\mathcal{Z}$ meeting the *nonanticipativity constraints* that $z_{is} = z_{js}$ whenever scenarios i and j are indistinguishable at stage s





Projecting onto the Nonanticipativity Space

- Following Rockafeller and Wets (1991), we use the following probability-weighted inner product on $\mathcal{Z}$:

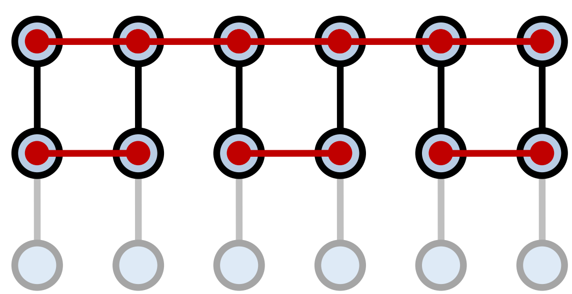
$$\left\langle (z_1,\ldots,z_n),(q_1,\ldots,q_n) \right\rangle = \sum_{i=1}^{n} \pi_i \left\langle z_i, q_i \right\rangle$$

- With this inner product, the projection map $\mathrm{proj}_{\mathcal{N}} : \mathcal{Z} \to \mathcal{N}$ is given by

$$\mathrm{proj}_{\mathcal{N}}(q) = z, \quad \text{where}$$

$$z_{is}^{k+1} = \frac{1}{\left(\sum_{j \in S(i,s)} \pi_j\right)} \sum_{j \in S(i,s)} \pi_j q_{js}^{k+1} \qquad i = 1,\ldots,n, \;\; s = 1,\ldots,T-1$$

and $S(i,s)$ is the set of scenarios indistinguishable from scenario i at time s.



Now Let’s Apply the ADMM

- So far, I have just shown the formulation of Rockafellar & Wets (1991) with some minor notation adjustments
- But now will derive PH using the ADMM instead of first principles

ADMM Notation

- Suppose $f : \mathcal{X} \to \mathbb{R} \cup \{+\infty\}$ is a convex function
- Suppose $g : \mathcal{Z} \to \mathbb{R} \cup \{+\infty\}$ is a convex function
- Suppose M is linear map $\mathcal{X} \to \mathcal{Z}$

$$\boxed{\min f(x) + g(Mx)}$$

- Equivalent formulation: $\boxed{\begin{array}{ll} \min & f(x) + g(z) \\ \text{ST} & Mx = z \end{array}}$

- The ADMM, for some fixed constant $\rho > 0$:

$$x^{k+1} \in \operatorname*{Arg\,min}_{x \in \mathbb{R}^n} \left\{ f(x) + \left\langle w^k, Mx \right\rangle + \tfrac{\rho}{2} \left\| Mx - z^k \right\|^2 \right\}$$

$$z^{k+1} \in \operatorname*{Arg\,min}_{z \in \mathbb{R}^m} \left\{ g(z) - \left\langle w^k, z \right\rangle + \tfrac{\rho}{2} \left\| Mx^{k+1} - z \right\|^2 \right\}$$

$$w^{k+1} = w^k + \rho(Mx^{k+1} - z^{k+1})$$

Setting Up the ADMM Formulation for Stochastic Programming

For each $i = 1, \ldots, n$, let

- $f_i : \mathcal{X}_i \to \mathbb{R} \cup \{+\infty\}$ be given by $f_i(x_i) = \pi_i h_i(x_i)$, where $h_i(\cdot)$ is the cost function for scenario i and $+\infty$ if any constraint within scenario i is violated
- $M_i : \mathcal{X}_i \to \mathcal{Z}_i$ be the map that just drops the last-stage variables from scenario i

h_i encapsulates all the costs and constraints across all stages in the (hypothetical) situation that you know the final outcome will be scenario i

Then our stochastic program is equivalent to

$$\boxed{\begin{array}{ll} \min & \displaystyle\sum_{i=1}^{p} f_i(x_i) \\ \text{ST} & (M_1 x_1, \ldots, M_n x_n) \in \mathcal{N} \end{array}}$$

Applying the ADMM

$$f(x)=\sum_{i=1}^{n} f_i(x_i) \qquad g(z)=\begin{cases}0, & z\in\mathcal{N}\\ +\infty, & z\notin\mathcal{N}\end{cases} \qquad M:(x_1,\ldots,x_n)\mapsto(M_1x_1,\ldots,M_nx_n)$$

Then the problem is equivalent to $\min f(x)+g(Mx)$

Applying the ADMM (and thus DR), we obtain

$$\boxed{\begin{aligned} x_i^{k+1} &= \underset{x_i\in\mathcal{X}_i}{\arg\min}\left\{f_i(x_i)+\left\langle M_ix_i,w_i^k\right\rangle+\tfrac{\rho}{2}\left\|M_ix_i-z_i^k\right\|^2\right\} && i=1,\ldots,n\\ z^{k+1} &= \mathrm{proj}_{\mathcal{N}}\left(Mx_1^{k+1}\right) && \\ w^{k+1} &= w^k+\rho(Mx^{k+1}-z^{k+1}) && i=1,\ldots,n \end{aligned}}$$

- Note that we always have $w^k=(w_1^k,\ldots,w_p^k)\in\mathcal{N}^\perp$. Why?
  - Projection means that $Mx-z\in\mathcal{N}^\perp$
  - Or, note that in ADMM/DR we always have $w^k\in\partial g(z^k)$
- Aside: applying DR to subspace indicator functions like g is equivalent to Spingarn's method of partial inverses

Progressive Hedging

- Writing the z and w operations out in detail, we obtain PH:

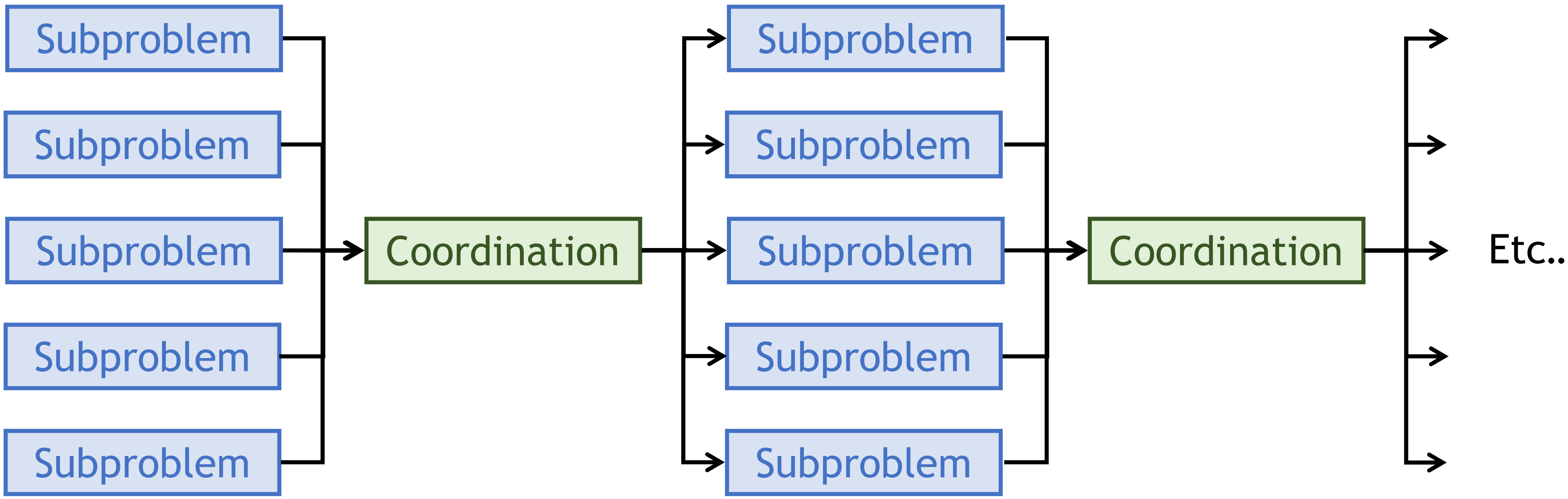
$$\boxed{\begin{array}{ll}
x_i^{k+1} = \underset{x_i \in \mathcal{X}_i}{\arg\min} \left\{ h_i(x_i) + \sum_{s=1}^{T-1} \left(\left\langle x_{is}, w_{is}^k \right\rangle + \frac{\rho}{2} \left\| x_{is} - z_{is}^k \right\|^2 \right) \right\} & i = 1,\ldots,n \\
z_{is}^{k+1} = \dfrac{1}{\left(\sum_{j \in S(i,s)} \pi_j \right)} \sum_{j \in S(i,s)} \pi_j x_{js}^{k+1} & \begin{array}{l} i = 1,\ldots,n \\ s = 1,\ldots,S-1 \end{array} \\
w_{is}^{k+1} = w_{is}^k + \rho(x_{is}^{k+1} - z_{is}^{k+1}) & \begin{array}{l} i = 1,\ldots,n \\ s = 1,\ldots,S-1 \end{array}
\end{array}}$$

- The $w^k \in \mathcal{N}^\perp$ condition can be written as $\sum_{j \in S(i,s)} \pi_j w_{js}^k = 0$ for all i and s

- By using canonical, non-probability-weighted inner products, we can also obtain an alternative version in which simple averages replace the weighted averages and the π_i appear in the x_i minimizations instead

Decomposition Methods

- PH is a form of *decomposition method*
- General form of decomposition methods:



- In any decomposition method, the subproblem computations can be operated in parallel
- But the coordination steps potentially pose a serial bottleneck

Noteworthy Properties of PH

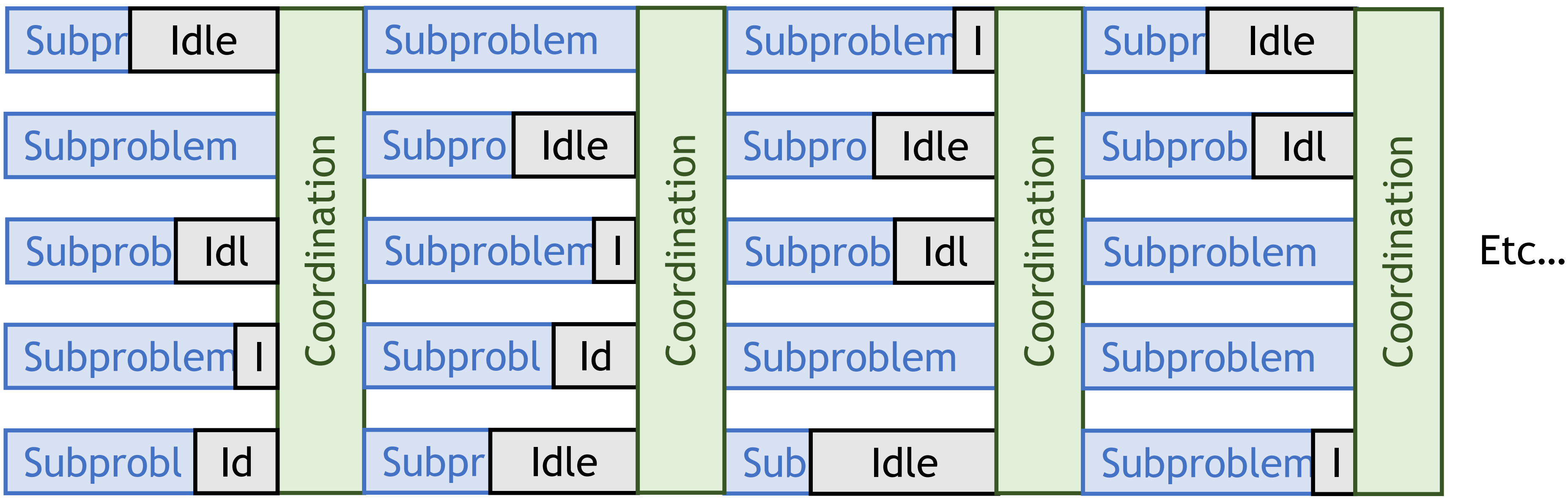
- The coordination computations in PH just consist of sums / averages and simple vector operations
  - These are faster than the "master" optimization problems other decomposition methods typically use…
  - … and can easily be implemented in a distributed manner (efficient parallel algorithms for sums etc.)
- PH handles multi-stage problems cleanly
  - Applying other decomposition methods to problems with 3 or more stages can require unwieldy "nested" versions
- The theory does not require linearity, only convexity
- Superficially, the algorithm is easily adapted to integer variables and nonconvex objectives or constraints
  - Although you lose the standard convergence theory and the method can become heuristic

Adoption of PH in Practice

- Progressive hedging did not “catch on” initially
- Convergence speed on practical problems was not spectacular
- However, its relative simplicity made it start to gain adherents with the advent of
  - Ever-larger problem isntances
  - Interest in problems with many stages
  - Wider availability of highly parallel computing
- So, 20+ years after initial publication, PH started getting used in practice
- The PySP system (Watson, Woodruff, Hart 2018) provides an accessible version of PH coupled with a flexible modeling environment (Pyomo – embedded in Python)
- There has been recent work on making its application with integer variables more rigorous

But Classic PH is Totally Synchronous

- In theory, you must solve every subproblem at every iteration
- The coordination step must wait for the slowest subproblem



Some possible remedies:

- Pack subproblems in processors and load balance (limits parallelism)
- Advanced bundle method variants
- Use projective splitting instead of ADMM / DR (this talk)

Projective Splitting: General Problem Setting

$$0 \in \sum_{i=1}^{n} G_i^* T_i(G_i x)$$

where

- $\mathcal{H}_0, \ldots, \mathcal{H}_n$ are real Hilbert spaces
- $T_i : \mathcal{H}_i \rightrightarrows \mathcal{H}_i$ are maximal monotone operators, $i = 1, \ldots, n$
- $G_i : \mathcal{H}_0 \to \mathcal{H}_i$ are bounded linear maps, $i = 1, \ldots, n$

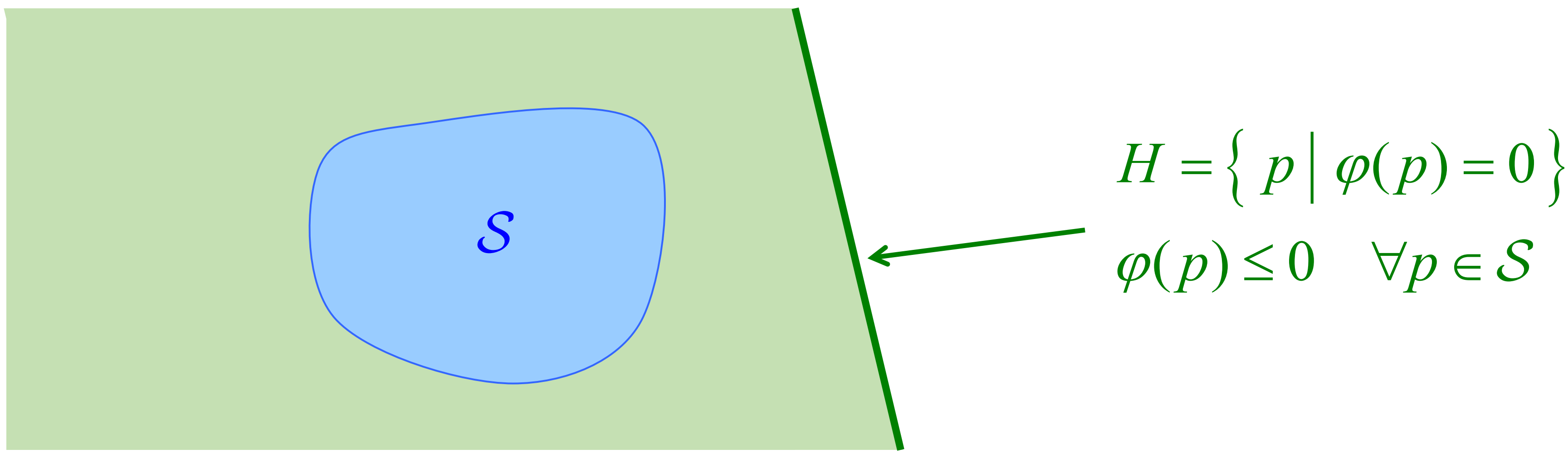
Kuhn-Tucker set / primal-dual solution set

$$\mathcal{S} = \left\{ (z, w_1, \ldots, w_n) \,\middle|\, (\forall i = 1, \ldots n)\; w_i \in T_i(G_i z),\; \sum_{i=1}^{n} G_i^* w_i = 0 \right\}$$

- This is a closed convex set (not immediate; various proofs)

Valid Inequalities for $\mathcal{S}$

- Take some $x_i, y_i \in \mathcal{H}_i$ such that $y_i \in T_i(x_i)$ for $i = 1, \ldots, n$, that is, $(x_i, y_i) \in \operatorname{graph} T_i$
- If $(z, \boldsymbol{w}) = (z, w_1, \ldots, w_n) \in \mathcal{S}$, then $w_i \in T_i(G_i z)$ for $i = 1, \ldots, n$
- So, $\left\langle x_i - G_i z, y_i - w_i \right\rangle \geq 0$ for $i = 1, \ldots, n$ by monotonicity of T_i
- Negate and add up: $\varphi(z, \boldsymbol{w}) = \sum_{i=1}^{n} \left\langle G_i z - x_i, y_i - w_i \right\rangle \leq 0 \quad \forall (z, \boldsymbol{w}) \in \mathcal{S}$



Making Sure these Inequalities are Affine

- Superficially, these inequalities are quadratic
- But with a little care we can make them affine
- One of several possible techniques:
  - Restrict the space to

$$\mathcal{V} = \mathcal{H}_0 \times \mathcal{W} \supset \mathcal{S}, \text{ where } \mathcal{W} = \left\{ \boldsymbol{w} = (w_1, \ldots, w_n) \in \mathcal{H}_1 \times \cdots \times \mathcal{H}_n \;\middle|\; \sum_{i=1}^{n} G_i^* w_i = 0 \right\}$$

  - Within this subspace, φ is affine since the quadratic terms are

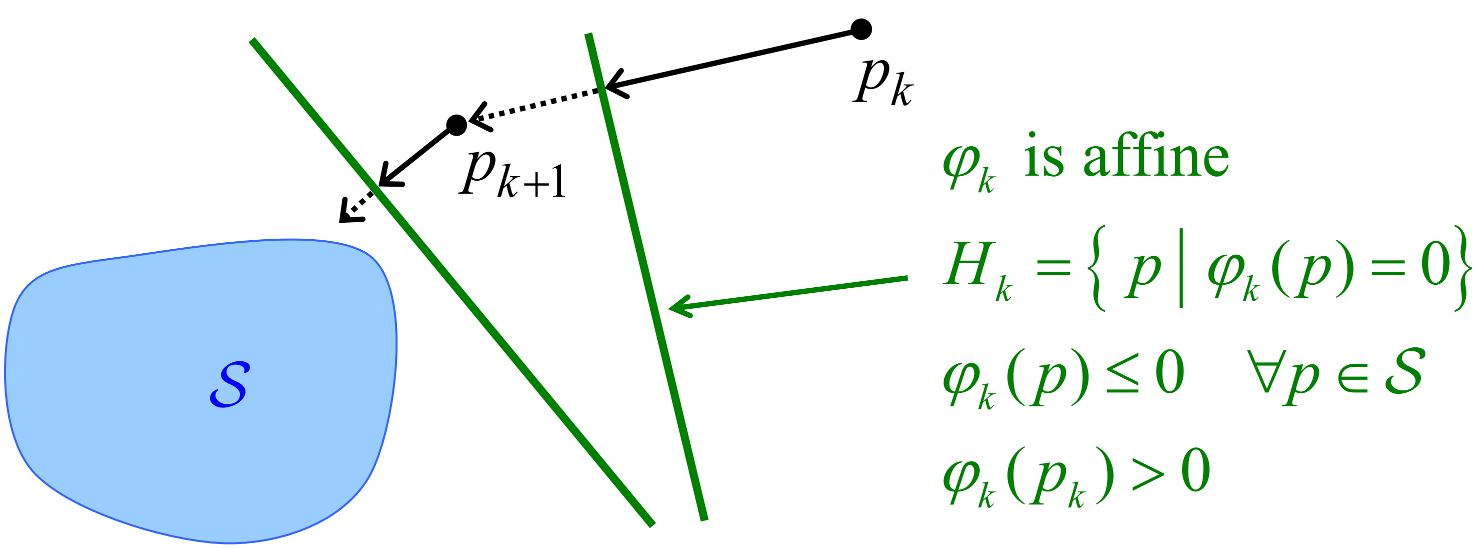
$$\sum_{i=1}^{n} \langle G_i z, -w_i \rangle = \sum_{i=1}^{n} \langle z, -G_i^* w_i \rangle = \left\langle z, -\sum_{i=1}^{n} G_i^* w_i \right\rangle = \langle z, -0 \rangle = 0$$

- Once we know φ is affine, projection onto the halfspace $H = \{ p \in V \mid \varphi(p) \le 0 \}$ is fairly straightforward

Generic Projection Method to Converge to a Point in a Closed Convex Set $\mathcal{S}$ in any Hilbert Space $\mathcal{V}$

Apply the following general template:

- Given $p^k \in \mathcal{V}$, choose some affine function φ_k with $\varphi_k(p) \le 0 \; \forall p \in \mathcal{S}$
- Project p^k onto $H_k = \left\{ p \,\middle|\, \varphi_k(p) \le 0 \right\}$, possibly with an overrelaxation factor $\nu_k \in [\varepsilon, 2-\varepsilon]$, yielding p_{k+1}, and repeat…



Projection Process in the Case of Projective Splitting

- Here, $p^k = (z^k, \boldsymbol{w}^k) = (z^k, w_1^k, \ldots, w_n^k)$ and we find φ_k by picking some $(x_i^k, y_i^k) \in \operatorname{graph} T_i \ (\forall i)$ and using the construction above

$$u^k = \operatorname{proj}_{\mathcal{W}}(x_1^k, \ldots, x_n^k) \qquad v^k = \sum_{i=1}^{n} G_i^* y_i^k$$

$$\tau_k = \left\|u^k\right\|^2 + \gamma \left\|v^k\right\|^2 \quad \text{(done if } \tau_k \approx 0\text{)}$$

$$\theta_k = \frac{\nu_k}{\tau_k} \max\left\{0, \sum_{i=1}^{n} \left\langle G_i z^k - x_i^k, y_i^k - w_i^k \right\rangle \right\}$$

$$z^{k+1} = z^k - \frac{\theta_k}{\gamma} v^k \qquad w^{k+1} = w^k - \theta_k u^k$$

- There are alternative approaches if $\operatorname{proj}_{\mathcal{W}}$ is difficult
- $\gamma > 0$ is an optional primal-dual scaling factor
- More complicated than ADMM/PH coordination step, but still just simple vector and sum operations (so could be distributed)

How to Pick the x_i^k, y_i^k — Basics

- If you pick $(x_i^k, y_i^k) \in \operatorname{graph} T_i$ completely arbitrarily, you may just orbit around $\mathcal{S}$ and not converge to it
- A workable choice: the "prox" operation for some scalar $c_{ik} > 0$

$$\boxed{(x_i^k, y_i^k) = \operatorname{Prox}_{T_i}^{c_{ik}}(G_i z^k + c_{ik} w_i^k)}$$

That is,

$$x_i^k = (I + c_{ik} T_i)^{-1}(G_i z^k + c_{ik} w_i^k) \qquad y_i^k = \frac{1}{c_{ik}}\left(G_i z^k + c_{ik} w_i^k - x_i\right)$$

- Then $c_{ik}(y_i^k - w_i^k) = G_i z^k - x_i^k$
- So $\left\langle G_i z^k - x_i^k, y_i^k - w_i^k \right\rangle = c_{ik}\left\| G_i z^k - x_i^k \right\|^2 = c_{ik}^{-1}\left\| y_i^k - w_i^k \right\|^2 \geq 0$
- Sum over i and get $\varphi_k(z^k, \boldsymbol{w}^k) > 0$ (cuts of current iterate)
- Can prove that this guarantees (weak) convergence to $\mathcal{S}$ if the c_{ik} are bounded away from zero and infinity

How to Pick the x_i^k, y_i^k — "Block Iterativity"

- Variation: do not have to activate every operator at every iteration (Combettes & E 2018)
  - For the rest, just recycle the previous x_i^k, y_i^k
- Let $M \geq 0$ be an integer
- Let $I_0, I_1, I_2, \ldots \subseteq \{1, \ldots, n\}$ be such that

$$(\forall i \geq 0) \quad \bigcup_{j=i}^{i+M} I_j = \{1, \ldots, n\}$$

- At iteration k, only activate the operators in I_k :

$$(x_i^k, y_i^k) = \mathrm{Prox}_{T_i}^{c_{ik}}(G_i z^k + c_{ik} w_i^k) \quad \forall i \in I_k$$
$$(x_i^k, y_i^k) = (x_i^{k-1}, y_i^{k-1}) \quad \forall i \in \{1, \ldots n\} \setminus I_k$$

- Convergence proof adapts ideas from successive projection methods for set intersection problems

How to Pick the x_i^k, y_i^k – "Lags"

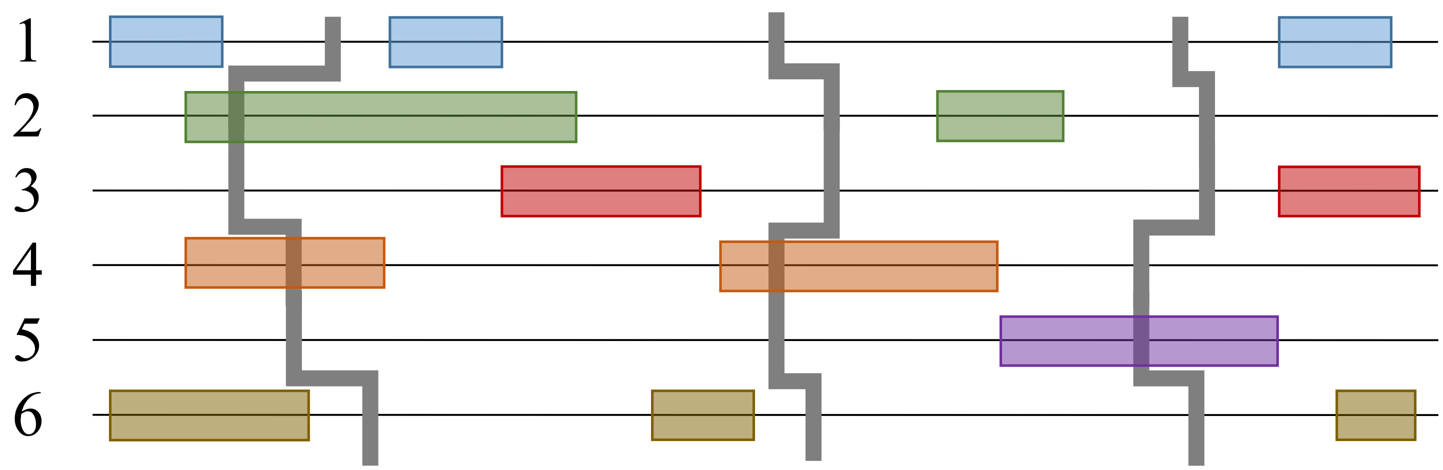
- Also from Combettes & E (2018)
- Let $K \geq 0$ be another integer
- Each prox operation may use data from up to K iterations ago
- Otherwise same as above

- So, for some $d(i,k)$ with $k \geq d(i,k) \geq k - K$,

$$
\begin{aligned}
(x_i^k, y_i^k) &= \mathrm{Prox}_{T_i}^{c_{ik}}\left(G_i z^{d(i,k)} + c_{ik} w_i^{d(i,k)}\right) & \forall i \in I_k \\
(x_i^k, y_i^k) &= (x_i^{k-1}, y_i^{k-1}) & \forall i \in \{1,\ldots n\} \setminus I_k
\end{aligned}
$$

Asynchrony (Sloppy Notation)

- Combining block iterativity and lags lets one drop strict coupling of coordination and subproblem processing
- For each i, suppose that a new $(x_i, y_i) \leftarrow \mathrm{Prox}_{T_i}^{c}(G_i z + c w_i)$ appears at least every t_1 time units, based on (z, w_i) that are at most t_2 time units old
- A projection step occurs at least every t_3 time units, based on $(x_i, y_i) \in \mathrm{graph}\, T_i,\ i = 1, \ldots, n$ that are most t_4 time units old
- Then $(z, \boldsymbol{w})$ converges (weakly) to a point in $\mathcal{S}$



Applying Projective Splitting to Stochastic Programming

Problem setup for stochastic programming

- $\mathcal{H}_0 = \mathcal{N}$ (run algorithm in nonanticipativity subspace)
- $\mathcal{H}_i = \mathcal{Z}_i$, but with its inner product multiplied by π_i
- $G_i : \mathcal{N} \to \mathcal{Z}_i$ selects the subvector relevant to scenario i
- $f_i(\tilde{x}_i) = \min\limits_{x_{iT}} \left\{ \pi_i h_i \left((\tilde{x}_i, x_{iT}) \right) \right\}$ minimizes scenario i's cost over the last-stage variables
    - Remember, scenario-infeasible points have $h_i(x_i) = +\infty$

Then our stochastic program is just

$$\min_{x \in \mathcal{H}_0} \sum_{i=1}^{n} f_i(G_i x)$$

So apply the method from earlier in the talk for $0 \in \sum_{i=1}^{n} G_i^* \partial f_i(G_i x)$

Some Technicalities

- This choice of f_i is convex for convex h_i
- But it is not generally guaranteed to be closed unless h_i has compact effective domain
- We need such an assumption to guarantee that f_i is closed and thus that $T_i = \partial f_i$ is maximal

- We also need some constraint-qualification-like conditions to be sure that the sufficient condition $0 \in \sum_{i=1}^{n} G_i^* \partial f_i(G_i x)$ is also necessary for optimality
  - Should not be a major concern in practice

Subproblem Processing

Subproblem: (many operating in parallel, asynchronously)

Let $0 < \rho_{\min} \le \rho_{\max} < \infty$ be fixed

Parameters for subproblem i:

- $z_i = (z_{i1}, \ldots, z_{i,T-1})$: scenario i "target" values, except last stage
- w_i : multipliers (same dimensions as z_i)

Get recent $z_i, w_i \in \mathcal{Z}_i$ from coordination process,

Select some $\rho \in [\rho_{\min}, \rho_{\max}]$

Let $x_i \in \operatorname*{Arg\,min}_{x_i} \left\{ h_i(x_i) + \langle M_i x_i, z_i \rangle + \frac{\rho}{2} \| M_i x_i - z_i \|^2 \right\}$

and $y_i = w_i + \rho(M_i x_i - z_i)$

Make i, $\tilde{x}_i \doteq M_i x_i$, y_i available to coordination process

Looks like PH subproblem + part of multiplier update

Coordination Process Variables

The coordination process maintains working variables:

- $z = (z_1, \ldots, z_n) \in \mathcal{N}$
- $w = (w_1, \ldots, w_n) \in \mathcal{N}^\perp$
- $\tilde{x} = (\tilde{x}_1, \ldots, \tilde{x}_n) \in \mathcal{Z}$ (the tildes mean no last-stage variables)
- $y = (y_1, \ldots, y_n) \in \mathcal{Z}$

At each iteration we also compute step direction vectors:

- $u = (u_1, \ldots, u_n) \in \mathcal{N}^\perp$
- $v = (v_1, \ldots, v_n) \in \mathcal{N}$

Scalar parameters:

- Primal-dual scaling factor $\gamma > 0$ (fixed?)
- Overrelaxation factor limits $0 < \nu_{\min} \le \nu_{\max} < 2$ (varying)

Coordination Process

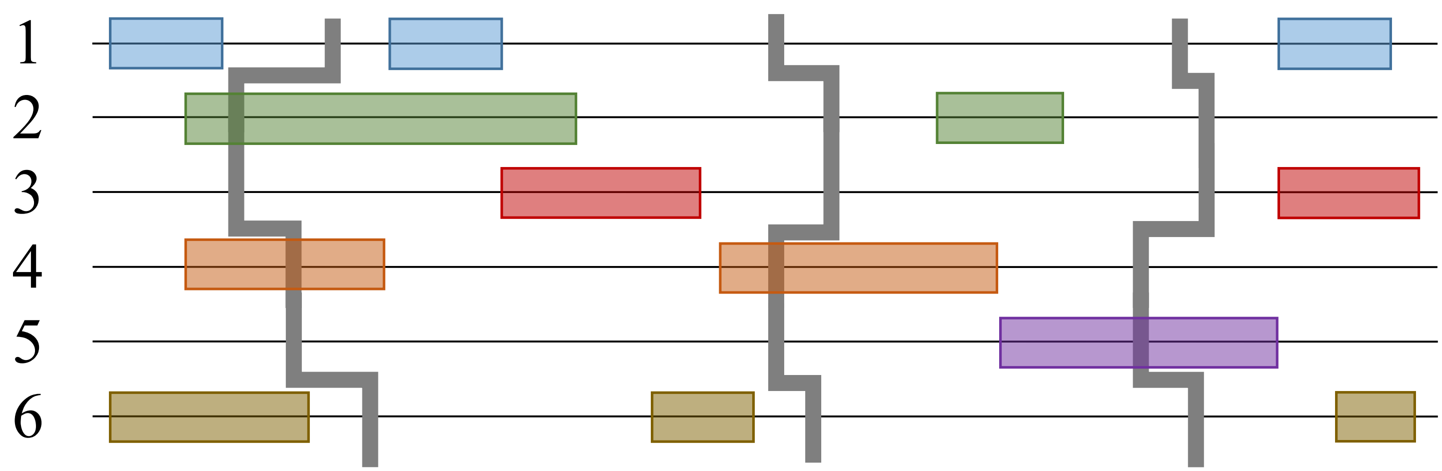
repeat
for $i = 1, \ldots, n$
let $\tilde{x}_i, y_i \in \mathcal{Z}_i$ be recent values from subproblem i
and let $\tilde{x} = (\tilde{x}_1, \ldots, \tilde{x}_n)$ and $y = (y_1, \ldots, y_n)$
$u \leftarrow \tilde{x} - \mathrm{proj}_{\mathcal{N}}(\tilde{x})$
$v \leftarrow \mathrm{proj}_{\mathcal{N}}(y)$
$\tau \leftarrow \|u\|^2 + \gamma \|v\|^2 = \sum_{i=1}^{n} \pi_i \|u_i\|^2 + \gamma \sum_{i=1}^{n} \pi_i \|v_i\|^2$
$\phi \leftarrow \langle z - \tilde{x}, w - y \rangle = \sum_{i=1}^{n} \pi_i \left(z_i - \tilde{x}_i\right)^{\mathsf{T}} \left(w_i - y_i\right)$
if $\phi > 0$ **then**
Choose some $\nu \in [\nu_{\min}, \nu_{\max}]$
$z \leftarrow z + (\nu\phi / \tau\gamma) v$
$w \leftarrow w + (\nu\phi / \tau) u$
until termination detected

- **Note:** this is not necessarily a central "master" process; it can be distributed

Asynchrony

Same conditions for convergence as in abstract asynchronous case above:

- Each subproblem is recomputed least every t_1 time units, based on (z, w_i) that are at most t_2 time units old
- A coordination step completes at least once every t_3 time units, based subproblem results that are at most t_4 time units old
- Then $(z, \mathbf{w})$ converges to a primal-dual solution, as in PH



Asynchronous Projective Hedging

- We call the resulting class of algorithms *asynchronous projective hedging* (APH)

Partial Resemblance to PH

- Subproblem has some recognizable pieces of the PH subproblem optimization step and multiplier update
  - Essentially the same minimization step for subproblems
  - $\mathrm{proj}_{\mathcal{N}}$ and simple vector operations
- The control process is somewhat more complicated than PH, but consists of the same fundamental operations
  - Nothing more complicated than $\mathrm{proj}_{\mathcal{N}}$
  - May still be implemented in a distributed way

But Now It's Asynchronous

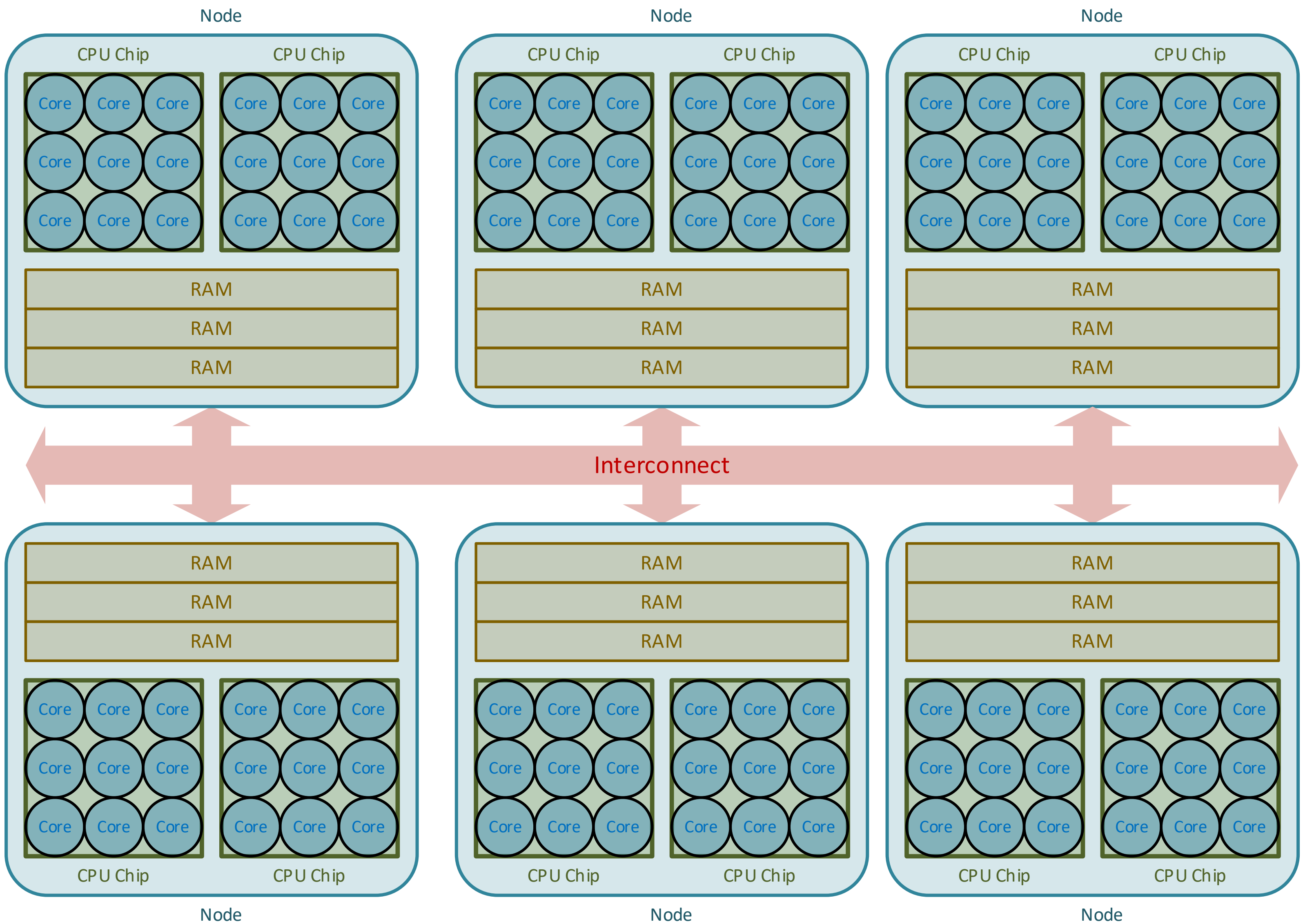
- Full synchronization has been replaced by loose timing bounds
- The subproblem and coordination processes can run at different speeds
  - No longer "locked in" to one coordination for every n subproblem solves
  - Now possible to have

$$\frac{\text{\# subproblem solves initiated by time } t}{\text{\# coordination steps by time } t} < n$$

Application

- **Problem:** SSN telecommunication design problem (Sen et al. 1994)
  - Standard test problem class in stochastic programming
  - For this exercise, we generated instances with up to 10^6 sample scenarios
  - Underlying number of scenarios finite but $\approx 10^{70}$
- **Hardware:** "Quartz" supercomputer at Lawrence Livermore
- **Software platform:** mpi-sppy (Kneuven et al. 2020)

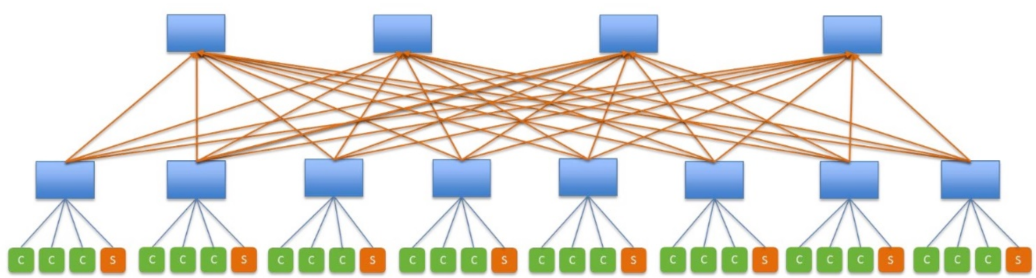
Standard Modern Supercomputer



- CPU cores share memory within each node
- Nodes communicate by messages through an interconnect

Quartz

- 32 CPU cores/node
- 128 GB RAM/node
- About 3,000 nodes
- Omni-Path interconnect (channel speed 100 G bits/second)



- Our biggest job so far used 250 nodes / 8,000 cores

mpi-sppy

- Sandia/Livermore package for stochastic programming
- Built on Pyomo optimization modeling environment that embeds in Python
- Coded in Python, but
  - Most CPU time spent solving subproblems (Gurobi etc.)
  - Or within numpy linear algebra kernels (calling BLAS)
- Has a “hub and spoke” architecture
  - But not in the classic “master-slave” sense
- “MPI” is how messages get sent between groups of processors (Gropp et al. 2014)

Lower Bounds

- Neither PH nor APH immediately provide pre-termination lower bounds on the optimal solutions value
- PH never fully minimizes the augmented Lagrangian, so it does not automatically provide a Lagrangian bound
- APH is similar
- But one can obtain one by doing an extra minimization of the ordinary Lagrangian (separable) – since $w^k \in \mathcal{N}^\perp$, compute

$$L(w^k) = \sum_{i=1}^{n} \min_{x_i} \left\{ f_i(x_i) + \left\langle M_i x_i, w_i^k \right\rangle \right\}$$

- And there may be other, application-specific lower bounds
- There is also a bound (E 2020) that one can derive directly from the PH process, but it requires estimation of a potentially large constant and may not be readily applicable

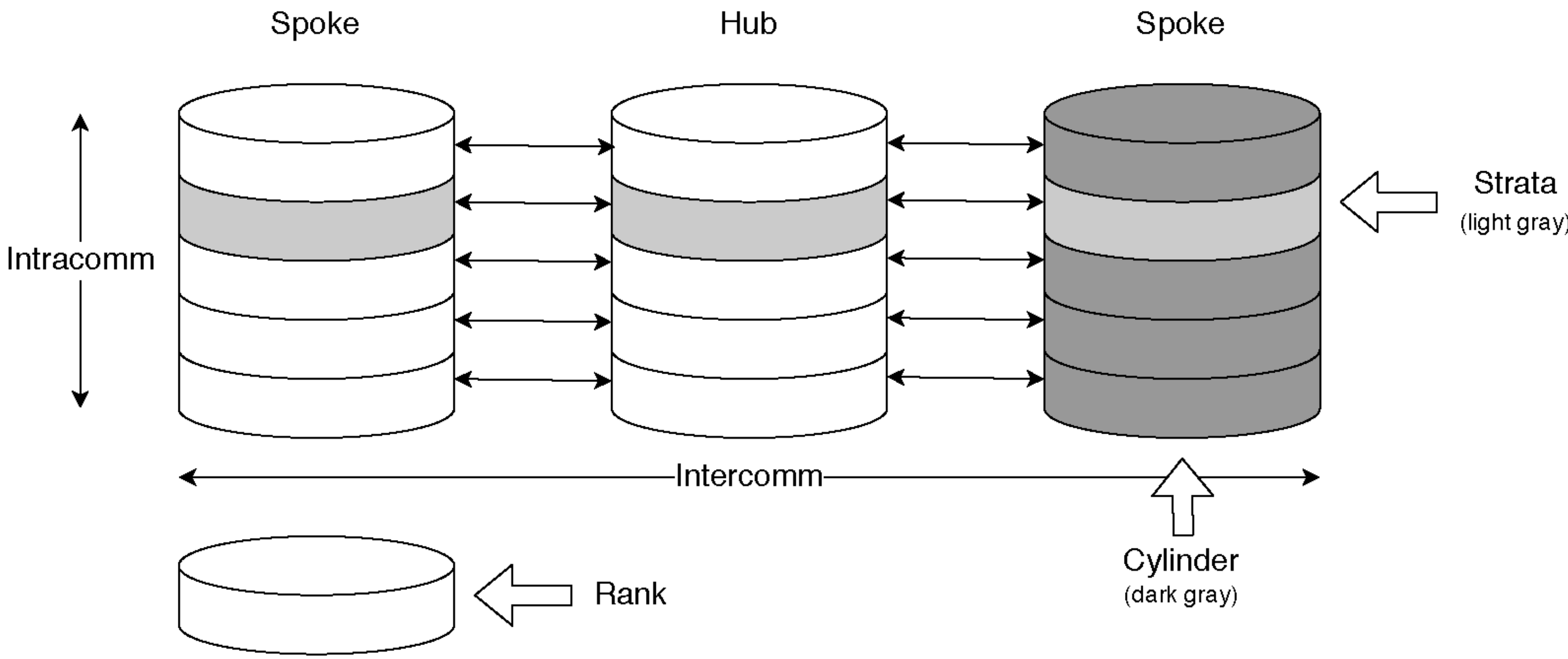
Upper Bounds

- Neither PH nor APH provide pre-termination feasible solutions
- Nonanticipativity is only satisfied in the limit
- Various strategies for deriving these feasible solutions from z

mpi-sppy Processor Organization

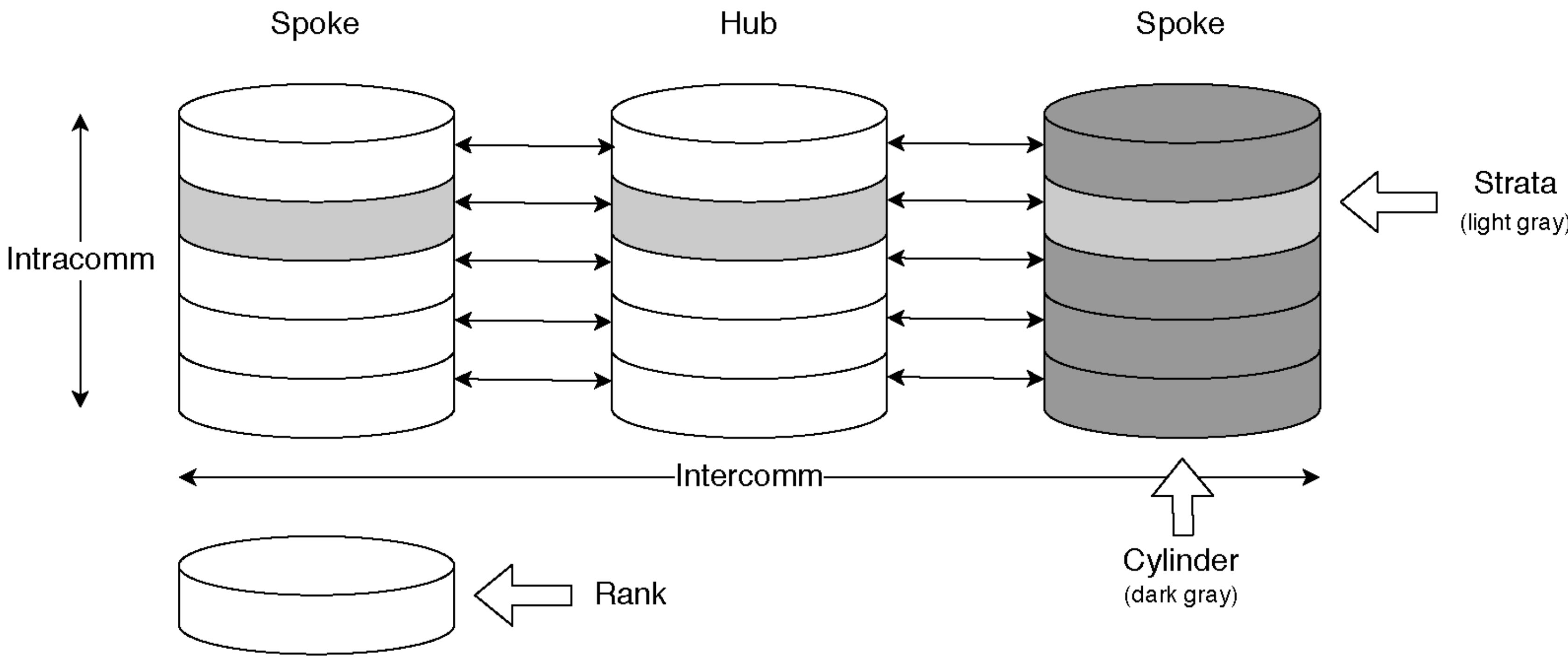
- A “hub” grouping of processors (possibly very large) runs the principal optimization algorithm (PH or APH)
- “Spoke” groupings of processors run auxiliary processes like upper and lower bounding (possibly several of each)
- The spoke operations are seeded from the principle $(z, \boldsymbol{w})$ iterates from the hub
- The spokes do not need to communicate directly with one another

The mpi-sppy Picture



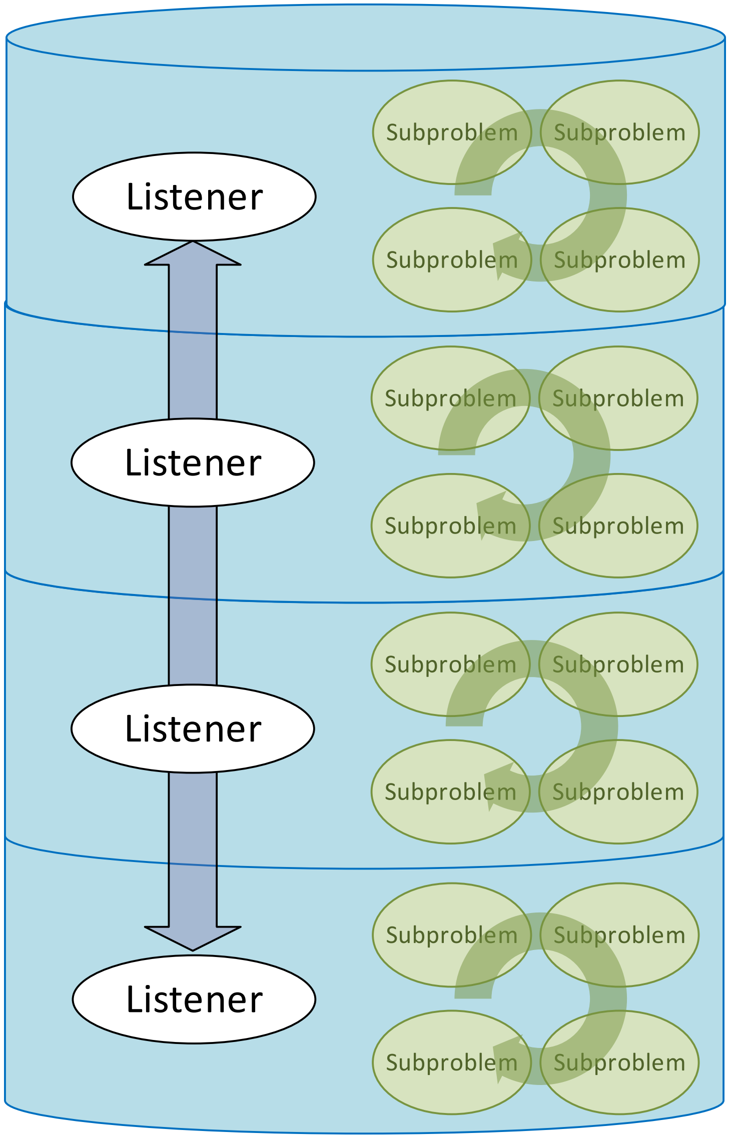
- A *rank* is a single shared memory space
  - Typically ~4 CPU cores (the most Gurobi efficiently can use for QPs)
  - Pack multiple ranks onto a node
  - Runs multiple threads and typically has multiple CPU cores
- Ranks are organized into *cylinders* and *strata*

The mpi-sppy Picture



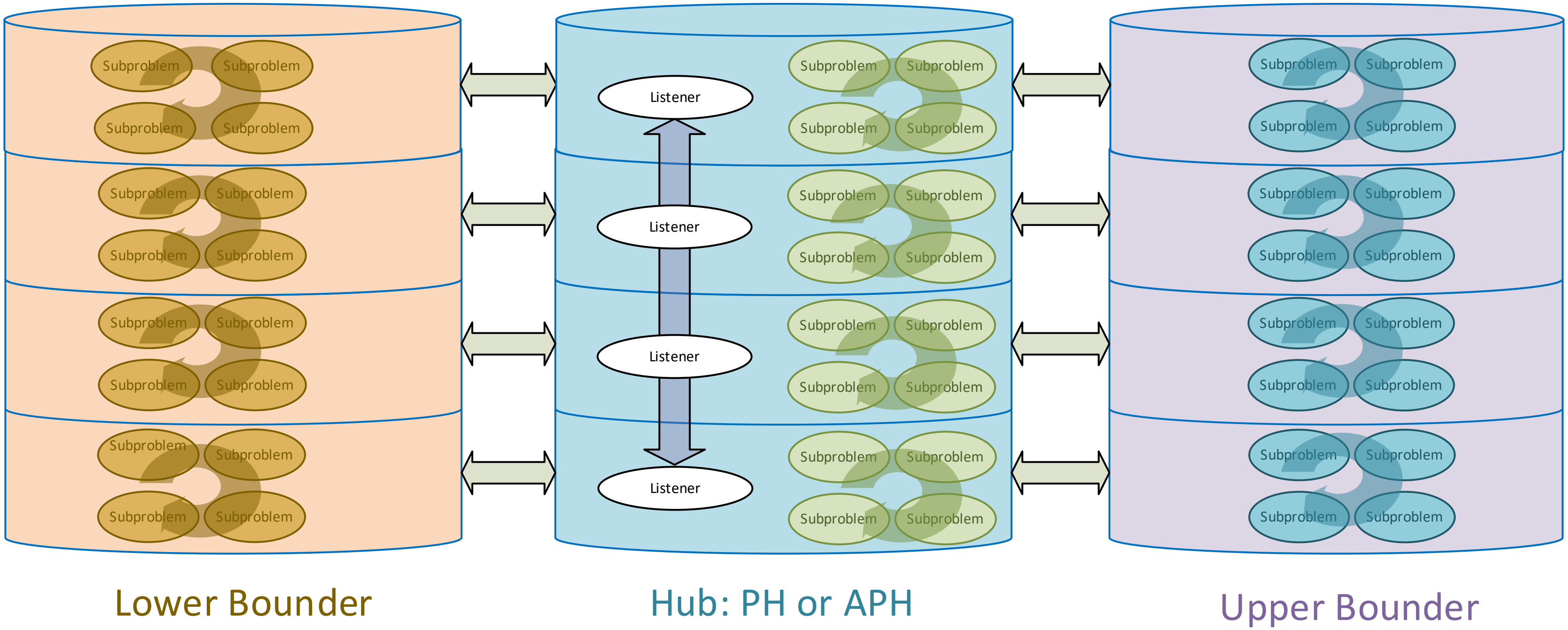
- Each rank stores the data for one or more scenarios
- Within a cylinder, each scenario is stored on only one rank
- Each cylinder has the same number of ranks
- The corresponding ranks in each cylinder (a stratum) each store data for the same scenarios (redundantly)

Within the Hub Cylinder



- Ranks may store more than one scenario
- Each rank only solves one scenario subproblem at a time
  - But Gurobi and numpy may employ multiple cores when doing so
- In PH, synchronous communication once every scenario has been solved
  - Subproblems stop during the communication
- In APH, an “listener” thread decides when “enough” scenarios have been solved (globally), then performs the communication needed for coordination
  - All in parallel with subproblem solves

Hub and Spoke



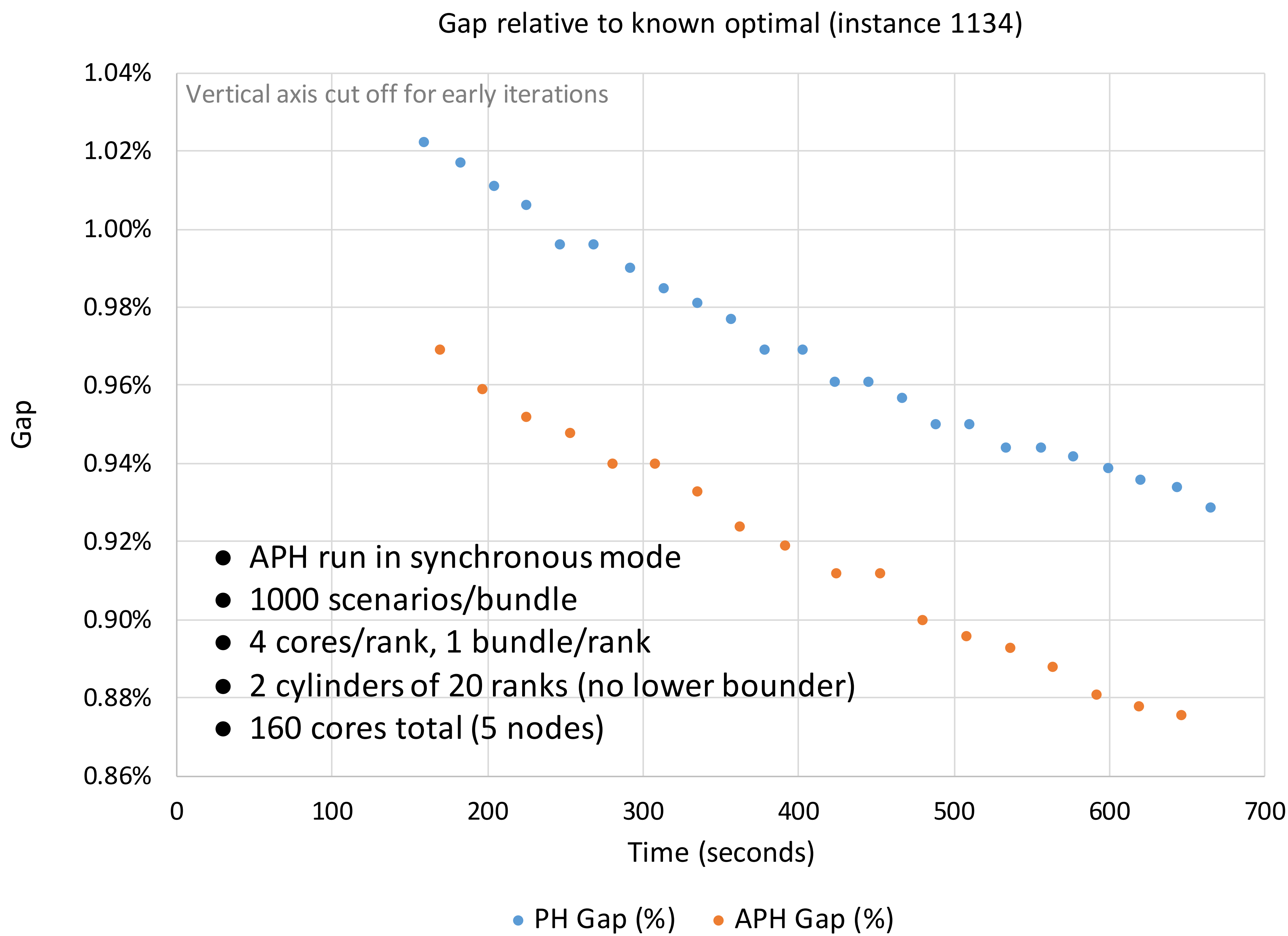
- The spokes are organized similarly to the hub
- Hub periodically sends z and/or w information to spokes
- Each rank sends or receives data only for the subproblems it owns
- Messages in different strata move in parallel

A Few More Points - Bundling and Subproblem Dispatch

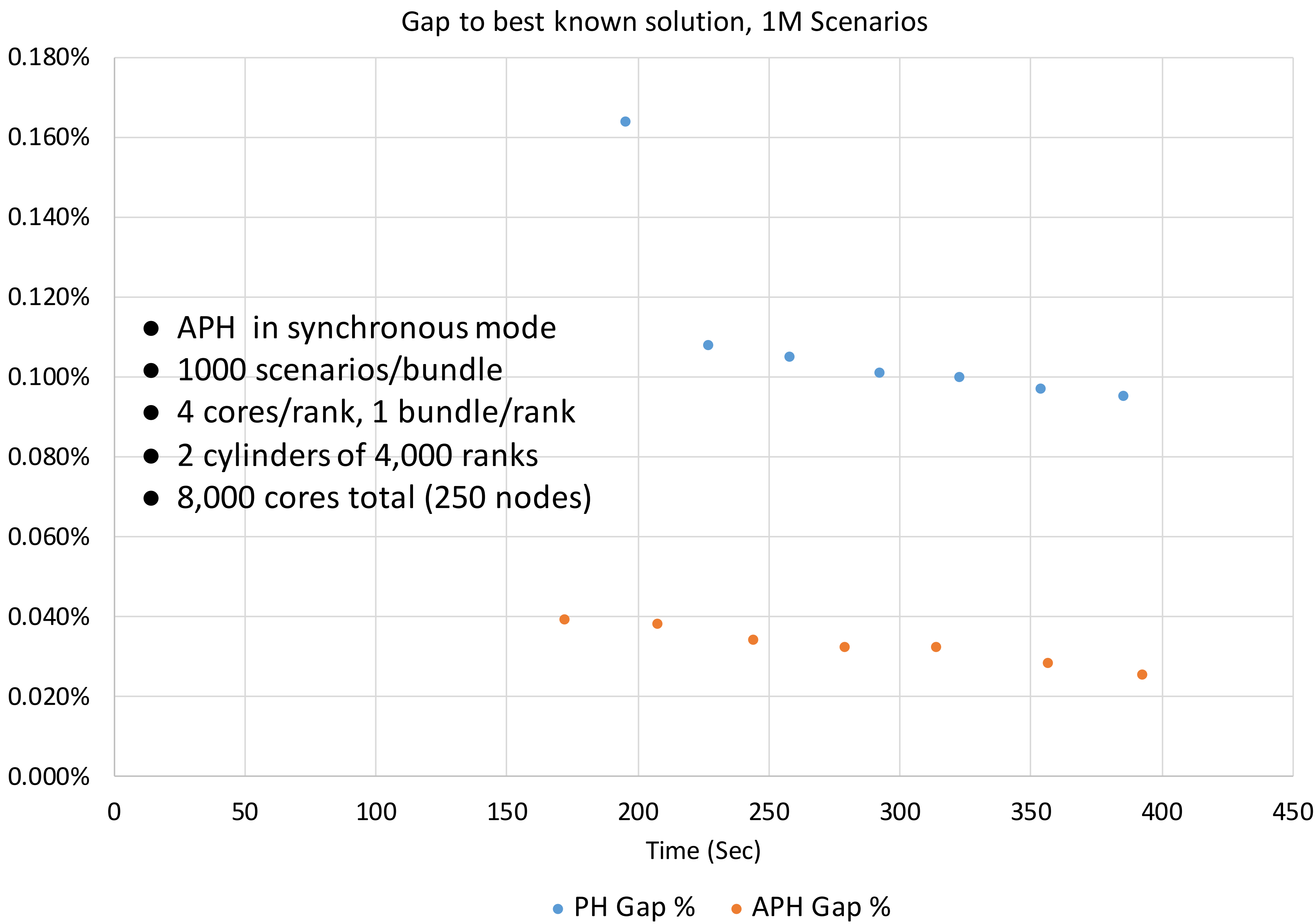
- For large-scale problems, it is common to use a “bundling” strategy
- Single-scenario subproblems are replaced by *bundles* of multiple scenarios
- Within a bundle/subproblem, scenarios are linked by explicit constraints (a “mini extensive form”)
- But the logic of nonanticipativity contraints (now between bundles) and PH / APH remain essentially the same

- In APH, we have a heuristic for selecting the most promising subproblem i to solve next within each rank
  - Based on value of $\pi_i \left(z_i - \tilde{x}_i \right)^{\mathsf{T}} \left(w_i - y_i \right)$

Preliminary Results: 20,000 Scenarios



Preliminary Results: 1,000,000 Scenarios



- APH in synchronous mode
- 1000 scenarios/bundle
- 4 cores/rank, 1 bundle/rank
- 2 cylinders of 4,000 ranks
- 8,000 cores total (250 nodes)

Size of the 1,000,000 Scenario Problem

The equivalent extensive-form LP would have about

- 795 million variables
- 265 million constraints (not counting simple variable bounds)
- 2.64 billion nonzero matrix entries

Much More to Do

- Dual instead of primal derivation of the APH method?
  - Would be more like classic derivation of ADMM from DR
  - Might eliminate annoying assumptions like a compact domain for the h_i
- Investigate making projective scaling methods in general more robust to problem scaling etc.
- The real problems the energy labs want to solve have nonconvexities and integer variables
  - Related to operating electric power grids
  - How can we help address such problems…
    - Rigorously?
    - Heuristically?
  - Asynchrony is an especially nice feature when the subproblems have integer variables