

School of Mathematics



Solving Very Large Financial Planning Problems on Massively Parallel Architecture

Andreas Grothey, University of Edinburgh

joint work with Jacek Gondzio, Marco Colombo

Overview

- Asset and Liability Management
 - Mean-Variance Formulation/Stochastic Programming
 - Structure of Problem
 - Model Extensions (NLP/scenario linking constraints)
- OOPS (Interior Point Solver)
 - Interior Point Methods
 - Structure Exploitation through Object Oriented Design
 - Parallel Implementation
- Results

Portfolio Optimization: Asset and Liability Management

- A set of assets $\mathcal{J} = \{1, \dots, J\}$ is given (e.g. bonds, stock, real estate).
- At every stage $t = 0, \dots, T-1$ we can buy or sell different assets.
- The return of asset j at stage t is *uncertain* (but distribution is known).

We have to make investment decisions: **what to buy or sell, at which time stage**

Objectives:

- maximize the final wealth
 - minimize the associated risk
- } \Rightarrow Mean Variance formulation:
 $\max \mathbf{E}(X) - \rho \text{Var}(X)$

Extensions (leading to nonlinear models)

- different risk measures (one-sided, expected shortfall/value at risk)
- (nonlinear) utility functions

Modelling: Multiperiod Mean-Variance Model**Variables:**

$x_{j,t}^h$ position in asset j at time t .

$x_{j,t}^s, x_{j,t}^b$ amount of asset j bought/sold at time t .

Parameters:

v_j value of asset j

$r_{j,t}$ return of asset j when held at time t

L_t, C_t liabilities/cash contributions at time t

Objective:

$$\max \mathbb{E}(X) - \rho \text{Var}(X), \quad X = (1 - c_t) \sum_j v_j x_{j,T}^h$$

Constraints:

$$x_{j,t}^h = (1 + r_{t-1,j})x_{j,t-1}^h - x_{j,t-1}^s + x_{j,t-1}^b \quad (\text{inventory})$$

$$C_t + (1 - c_t) \sum_j v_j x_{j,t}^s = L_t + (1 + c_t) \sum_j v_j x_{j,t}^b \quad (\text{cash balance})$$

Modelling: Multiperiod Mean-Variance Model**Variables:**

$x_{j,t}^h$ position in asset j at time t .

$x_{j,t}^s, x_{j,t}^b$ amount of asset j bought/sold at time t .

Parameters:

v_j value of asset j

$r_{j,t}$ return of asset j when held at time t

L_t, C_t liabilities/cash contributions at time t

Objective:

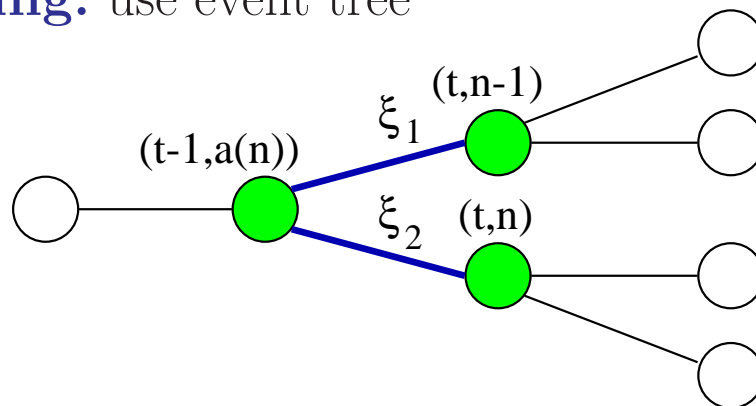
$$\max \mathbb{E}(X) - \rho \text{Var}(X), \quad X = (1 - c_t) \sum_j v_j x_{j,T}^h$$

Constraints:

$$x_{j,t}^h = (1 + r_{t-1,j})x_{j,t-1}^h - x_{j,t-1}^s + x_{j,t-1}^b \quad (\text{inventory})$$

$$C_t + (1 - c_t) \sum_j v_j x_{j,t}^s = L_t + (1 + c_t) \sum_j v_j x_{j,t}^b \quad (\text{cash balance})$$

Stochastic Programming: use event tree



- **Stages** represent points in time at which decisions are taken.
- **Nodes** represent possible futures (given by asset returns and probabilities):

$r_j^{(t,n)}$ return of asset j at node (t, n) .

$p_{(t,n)}$ probability of reaching node (t, n) .

$L_{(t,n)}, C_{(t,n)}$ liability payment/cash-contribution at node (t, n) .

Asset and Liability Management as Stochastic Program

With every node $i = (t, n)$ we associate **variables**:

$x_{j,i}^h$ the position in asset j at node i ;

$x_{j,i}^b, x_{j,i}^s$ the amount of asset j bought/sold at node i .

and **constraints**:

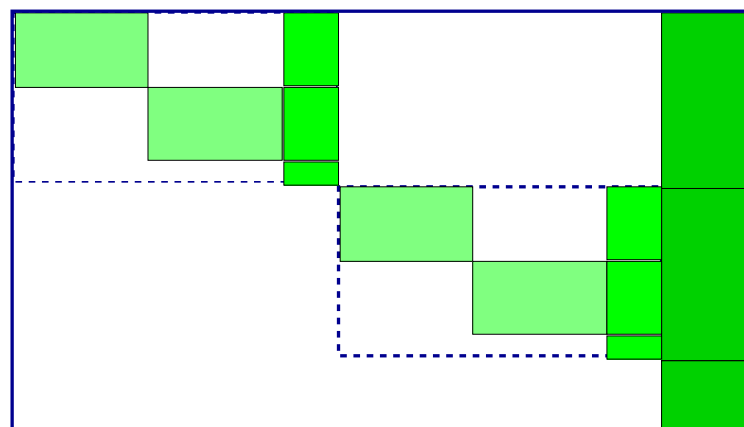
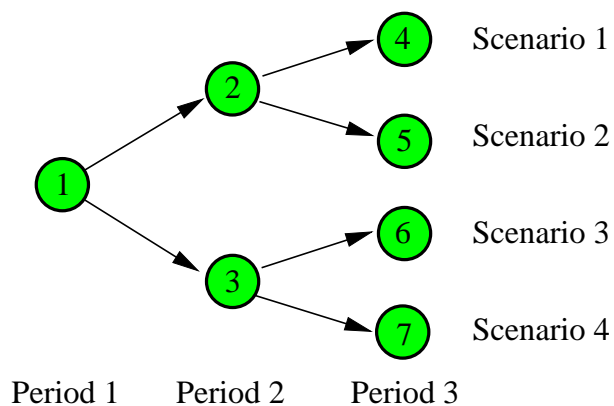
$$(1 + r_{i,j})x_{\pi(i),j}^h = x_{i,j}^h - x_{i,j}^b + x_{i,j}^s, \forall j \quad (\text{inventory})$$

$$\sum_j (1 + c_t)v_j x_{i,j}^b + L_i = \sum_j (1 - c_t)v_j x_{i,j}^s + C_i \quad (\text{cash balance})$$

Objective is sum over final stage scenarios

$$\begin{aligned} \min \mathbb{E}(X) - \rho \text{Var}(X) &= \mathbb{E}(X) - \rho \mathbb{E}[(X - E(X))^2] \\ &= (1 - c_t) \sum_{i \in L_T} p_i \sum_j v_j x_{i,j}^h - \rho \left[\sum_{i \in L_T} p_i [\dots] \right] \end{aligned}$$

Multistage Stochastic Programming

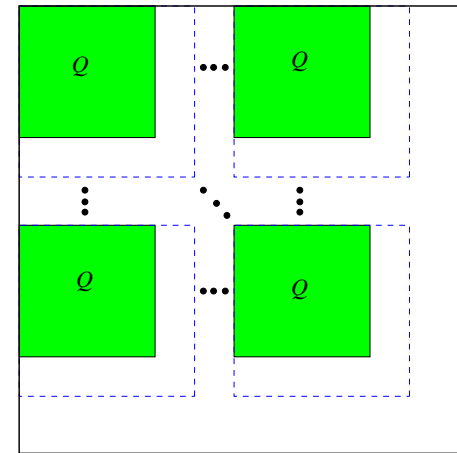


Symmetrical event tree with K realizations/node and T periods corresponds to

$$K^{T-1} \text{ scenarios} \quad \frac{K^T - 1}{K - 1} \text{ nodes (blocks)}$$

Structure of Objective

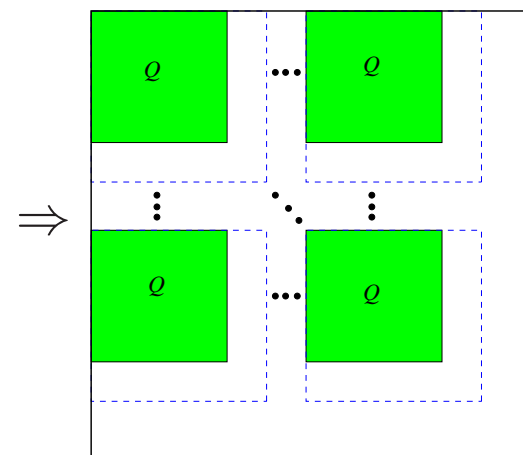
$$\begin{aligned} \mathbb{E}(X) - \rho \text{Var}(X) &= \mathbb{E}(X) - \rho[\mathbb{E}(X^2) - \mathbb{E}(X)^2] \\ &= \sum_{i \in L_T} p_i \sum_j v_j x_{ij}^h - \rho \left[\sum_{i \in L_T} p_i \sum_j (v_j x_{ij}^h)^2 - \left[\sum_{i \in L_T} p_i \sum_j v_j x_{ij}^h \right]^2 \right] \end{aligned} \Rightarrow$$



Dense, convex Hessian

Structure of Objective

$$\begin{aligned} \mathbb{E}(X) - \rho \text{Var}(X) &= \mathbb{E}(X) - \rho[\mathbb{E}(X^2) - \mathbb{E}(X)^2] \\ &= \sum_{i \in L_T} p_i \sum_j v_j x_{ij}^h - \rho \left[\sum_{i \in L_T} p_i \sum_j (v_j x_{ij}^h)^2 - \left[\sum_{i \in L_T} p_i \sum_j v_j x_{ij}^h \right]^2 \right] \end{aligned}$$

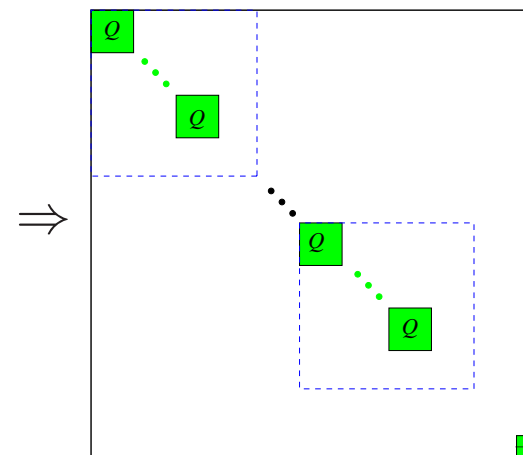


Dense, convex Hessian

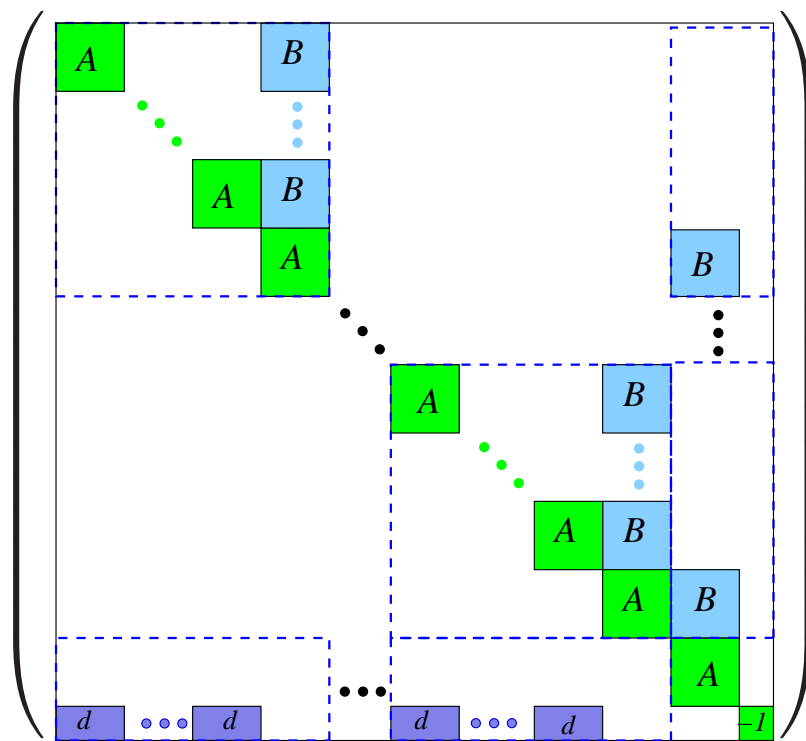
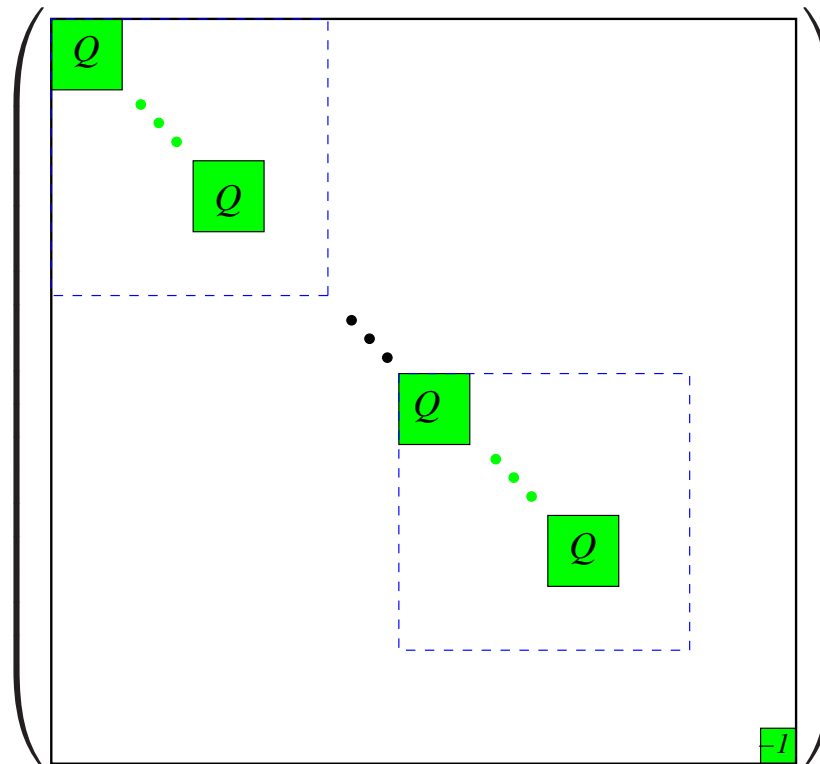
Alternatively:

$$\text{with } y = \sum_{i \in L_T} p_i \sum_j v_j x_{ij}^h$$

$$\mathbb{E}(X) - \rho \text{Var}(X) = y - \rho \left[\sum_{i \in L_T} p_i \sum_j (v_j x_{ij}^h)^2 - y^2 \right]$$



Sparse, nonconvex Hessian

ALM: Structure of matrices A and Q :Matrix A Matrix Q 

\Rightarrow Matrices are sparse, nested block structured

ALM: Extensions

Introduce two more (nonnegative) variables per final scenario $i \in L_t$ to model the positive and negative variation from the mean

$$(1 - c_t) \sum_{j=1}^J v_j x_{i,j}^h + s_i^+ - s_i^- = y.$$

Since $(s_i^+)^2, (s_i^-)^2$ cannot both be positive the variance is expressed as

$$\text{Var}(X) = \sum_{i \in L_t} p_i (s_i^+ - s_i^-)^2 = \sum_{i \in L_t} p_i ((s_i^+)^2 + (s_i^-)^2).$$

We model **downside risk** using a semi-variance $\mathbb{E}[(X - \mathbb{E}X)_-^2]$

$$\mathbb{E}[(X - \mathbb{E}X)_-^2] = \sum_{i \in L_t} p_i (s_i^+)^2.$$

Downside risk can be taken into account

- in the objective, or
- as a constraint.

ALM: Extensions

Standard Mean-Variance formulation:

$$\max y - \sigma \sum_{i \in L_t} p_i ((s_i^+)^2 + (s_i^-)^2) \quad \text{s.t.} \quad \begin{array}{l} (C1) \quad (\text{inventory constraints}) \\ (C2) \quad (\text{cash balance constraints}) \end{array} \quad (\text{QP})$$

Downside risk constrained:

$$\max y \quad \text{s.t.} \quad \begin{array}{l} \sum_{i \in L_t} p_i (s_i^+)^2 \leq \rho \\ (C1)/(C2) \end{array} \quad (\text{NLP})$$

Nonlinear utility function:

$$\max \sum_{i \in L_t} p_i \log(\sum v_j x_{i,j}^h) \quad \text{s.t.} \quad \begin{array}{l} \sum_{i \in L_t} p_i (s_i^+)^2 \leq \rho \\ (C1)/(C2) \end{array} \quad (\text{NLP})$$

Skewness term in objective:

$$\max y + \gamma \sum_{i \in L_t} p_i (s_i^+ - s_i^-)^3 \quad \text{s.t.} \quad \begin{array}{l} \sum_{i \in L_t} p_i ((s_i^+)^2 + (s_i^-)^2) \leq \rho \\ (C1) - (C3) \end{array} \quad (\text{NLP})$$

⇒ Solver allows flexibility in modelling

Stochastic Programming approach to Financial Planning

- Popular with academics, practitioners seem suspicious
 - Large problem sizes ✗
 - Places constraints on what can be modelled (?) ✗
- Traditionally solved by decomposition
 - Nested Benders Decomposition aka L-shaped method
 - Works well for LP models ✓
 - Unclear how to extend to quadratic/nonlinear models ✗
- Or solved by Interior Point Methods
 - Use taylored linear algebra
 - Only certain types of constraints are allowable (?) ✗

⇒ Structure Exploitation is key to succesful implementation

Solution Approaches: (nonexhaustive list, obviously)

For LPs:

- Benders decomposition and its extensions:
 - Benders, *Numerische Mathematik* 4 (1962).
 - Van Slyke and Wets, *SIAM J on Appl. Maths* 17 (1969).
 - Birge, *Operations Research* 33 (1985).
 - Ruszczynski, *Mathematical Programming* 33 (1985).
 - Gassmann, *Mathematical Programming* 47 (1990).
 - Mulvey and Ruszczyński, *Operations Research* 43 (1995).
 - Gondzio and Kouwenberg, *Operations Research* 49 (2001).
 - Linderoth and Wright, *Computational Opt. and Appl.* 24 (2003).
- Interior point methods:
 - Birge and Qi, *Management Science* 34 (1988).
 - Jessup, Yang and Zenios, *SIAM J on Opt.* 4 (1994).
 - Vladimirou and Zenios, *Annals of OR* 90 (1999).

For NLPs:

- Specialized interior point methods:
 - Steinbach, *Hierarchical Sparsity ...*, Uryasev and Pardalos (eds) 2000.
 - Blomvall and Lindberg, *A Riccati Solver ...*, *EJOR* 143, *OMS* 17 (2002).
 - Gondzio and Grothey, *OOPS: Exploiting structure in QPs and NLPs ...*

OOPS - Object Oriented Parallel (Interior Point) Solver

- Mantra: “Truly large scale problems are not only sparse but structured”
(due to e.g. dynamics, uncertainty, spatial distribution etc.)
- Exploiting structure is key to building efficient IPMs for large problems:
 - Faster linear algebra
 - Reduced memory use (by use of implicit factorization)
 - Possibility to exploit (massive) parallelism
 - **We assume that structure is known!** \Rightarrow no automatic detection.
- OOPS is a general purpose (parallel) Interior Point solver
 - Not tuned to any particular hardware
 - Not tuned to any particular problem (structure).
- OOPS can solve LP/QP/NLP problems.

Interior Point Methods (for LP/QP)

$$\min c^\top x + \frac{1}{2}x^\top Qx \quad \text{s.t.} \quad Ax = b \quad (\text{QP})$$

$$x \geq 0$$

Optimality conditions: $c + Qx - A^\top y - z = 0$

$$Ax = b$$

$$XZe = 0(\mu e)$$

$$x, z \geq 0$$

\Rightarrow Newton Step:

$$\begin{bmatrix} -Q & A^\top & I \\ A & 0 & 0 \\ Z & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} \xi_c \\ \xi_b \\ r_{xz} \end{bmatrix} = \begin{bmatrix} c + Qx - A^\top y - z \\ b - Ax \\ \mu e - XZe \end{bmatrix} \quad (\text{NS-QP})$$

\Rightarrow Reduced to

$$\begin{bmatrix} -Q - \Theta & A^\top \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \xi_c - X^{-1}r_{xz} \\ \xi_b \end{bmatrix}$$

where $\Theta = X^{-1}Z$, $X = \text{diag}(x)$, $Z = \text{diag}(z)$

Interior Point Methods (for NLP)

$$\min f(x) \quad \text{s.t.} \quad g(x) \leq 0$$

Add slacks to nonlinear inequalities:

$$\begin{aligned} \min f(x) \quad \text{s.t.} \quad & g(x) + z = 0 \\ & z \geq 0 \end{aligned} \quad (\text{NLP})$$

Optimality conditions:

$$\begin{aligned} \nabla f(x) + \nabla g(x)^\top y &= 0 \\ g(x) + z &= 0 \\ YZe &= 0 \quad (\mu e) \\ x, z &\geq 0 \end{aligned}$$

\Rightarrow Newton Step:

$$\begin{bmatrix} Q(x, y) & A(x)^\top \\ A(x) & -\Theta \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} -\nabla f(x) - A(x)^\top y \\ -g(x) - \mu Y^{-1}e \end{bmatrix} \quad (\text{NS-NLP})$$

where

$$\begin{aligned} Q(x, y) &= \nabla_{xx}^2 (f(x) + y^\top g(x)), \quad A(x) = \nabla g(x) \\ \Theta &= ZY^{-1}, \quad Y = \text{diag}(y), \quad Z = \text{diag}(z) \end{aligned}$$

Linear Algebra of IPMs

Main work: solve

$$\underbrace{\begin{bmatrix} -Q - \Theta & A^\top \\ A & 0 \end{bmatrix}}_{\Phi(QP)} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r \\ h \end{bmatrix} \quad \text{or} \quad \underbrace{\begin{bmatrix} -Q(x, y) & A(x)^\top \\ A(x) & \Theta \end{bmatrix}}_{\Phi(NLP)} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r \\ h \end{bmatrix}$$

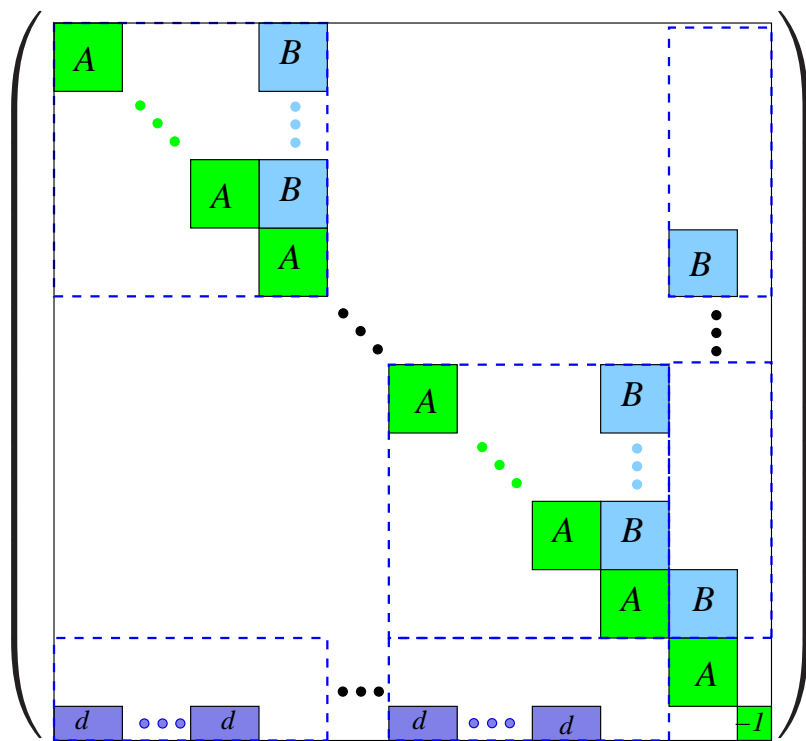
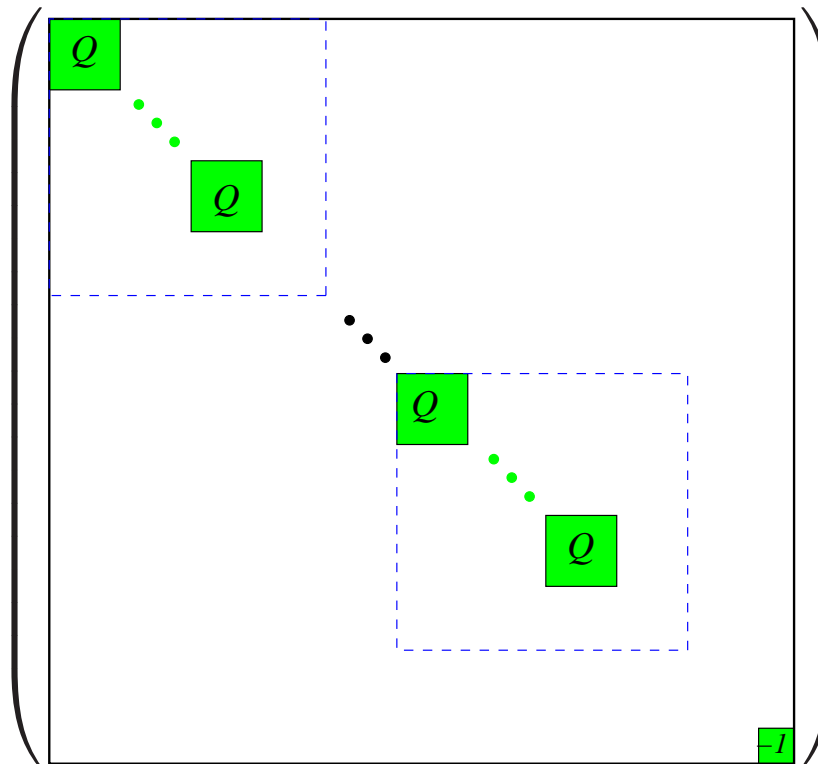
for several right-hand-sides at each iteration

⇒ Two stage solution procedure

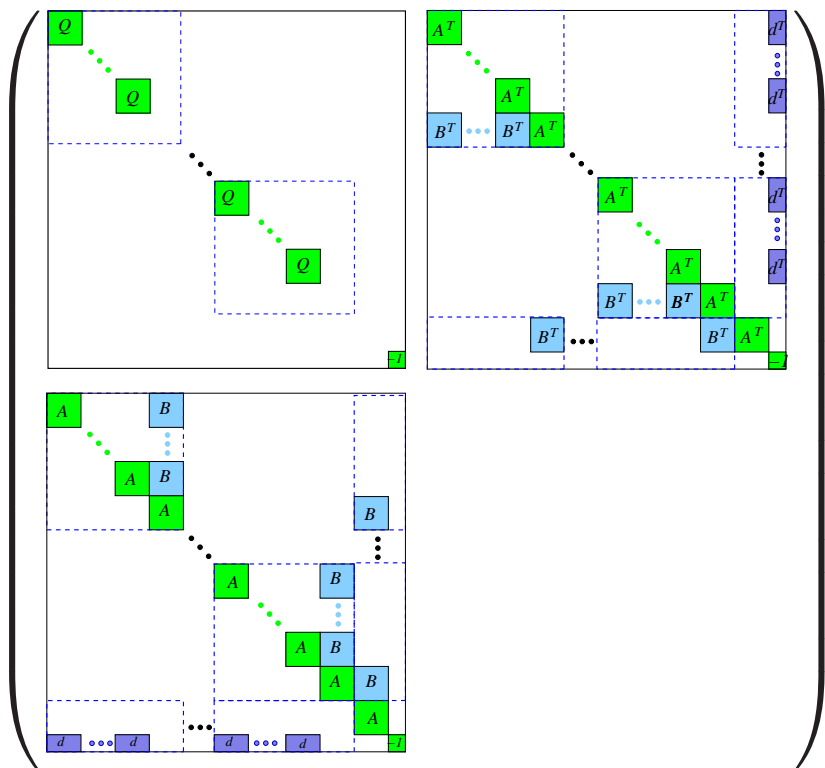
- factorize $\Phi = LDL^\top$
- backsolve(s) to compute direction $(\Delta x, \Delta y) + \text{corrections}$

⇒ Φ changes numerically but not structurally at each iteration

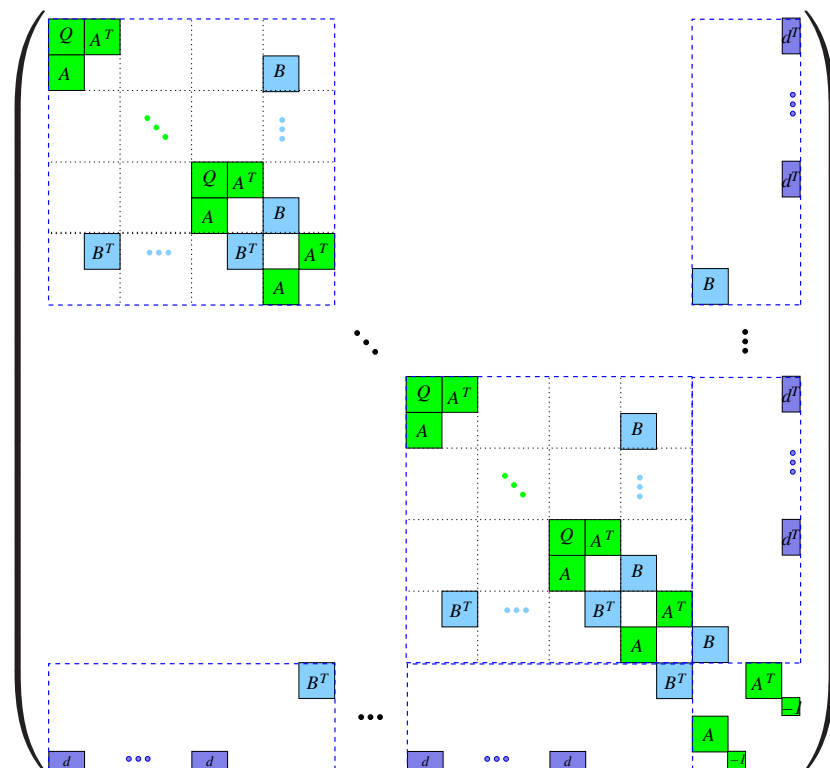
Key to efficient implementation is exploiting structure of Φ in these two steps

ALM: Structure of matrices A and Q :Matrix A Matrix Q 

Structures of A and Q imply structure of Φ :



$$\begin{pmatrix} Q & A^T \\ A & 0 \end{pmatrix}$$



$$P \begin{pmatrix} Q & A^T \\ A & 0 \end{pmatrix} P^{-1}$$

Structured Matrices

- Bordered Block-Diagonal Structure:

$$\Phi = \begin{pmatrix} A_1 & & & B_1^\top \\ & \cdots & & \vdots \\ & & A_n & B_n^\top \\ B_1 & \cdots & B_n & B_{n+1} \end{pmatrix}$$

- Block Tri-Diagonal Structure:

$$\Phi = \begin{pmatrix} A_1 & B_1^\top & & \\ B_1 & \cdots & \cdots & \\ & \cdots & \cdots & B_{n-1}^\top \\ & & B_{n-1} & A_n \end{pmatrix}$$

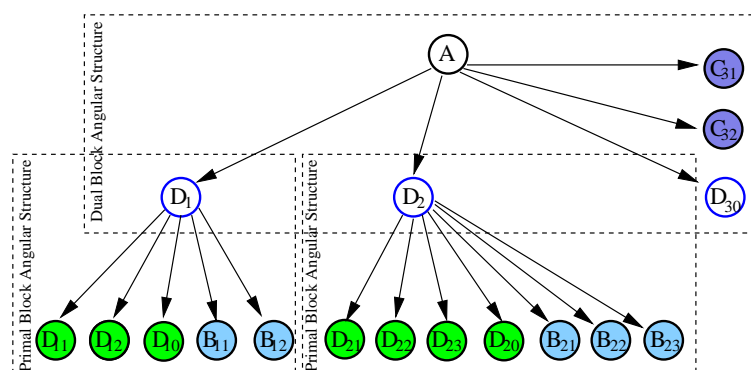
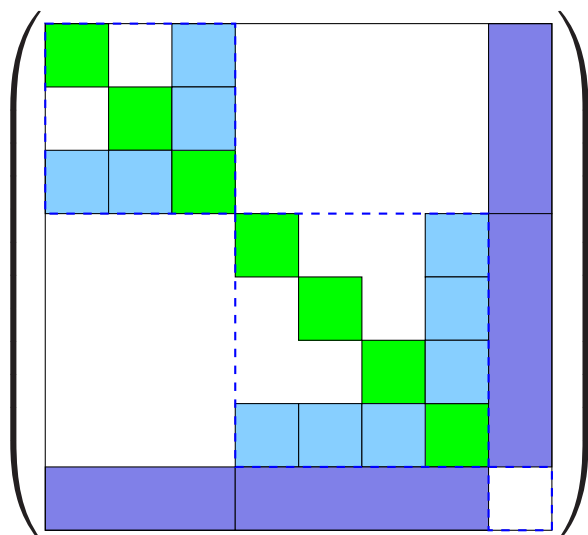
- Rank-corrector:

$$\Phi = D + RR^\top, \quad D \text{ easily invertible, } R \text{ small number of columns}$$

- Special Matrices (Network, Projection etc.)
- General Sparse Matrix

Structures may also be nested

Tree representation of matrices:



Structure Exploiting Linear Algebra

Every block of A , Q , Φ should have special structure exploiting linear algebra

- Blocks may be nested
- All blocks may be of different structure

Object-oriented approach

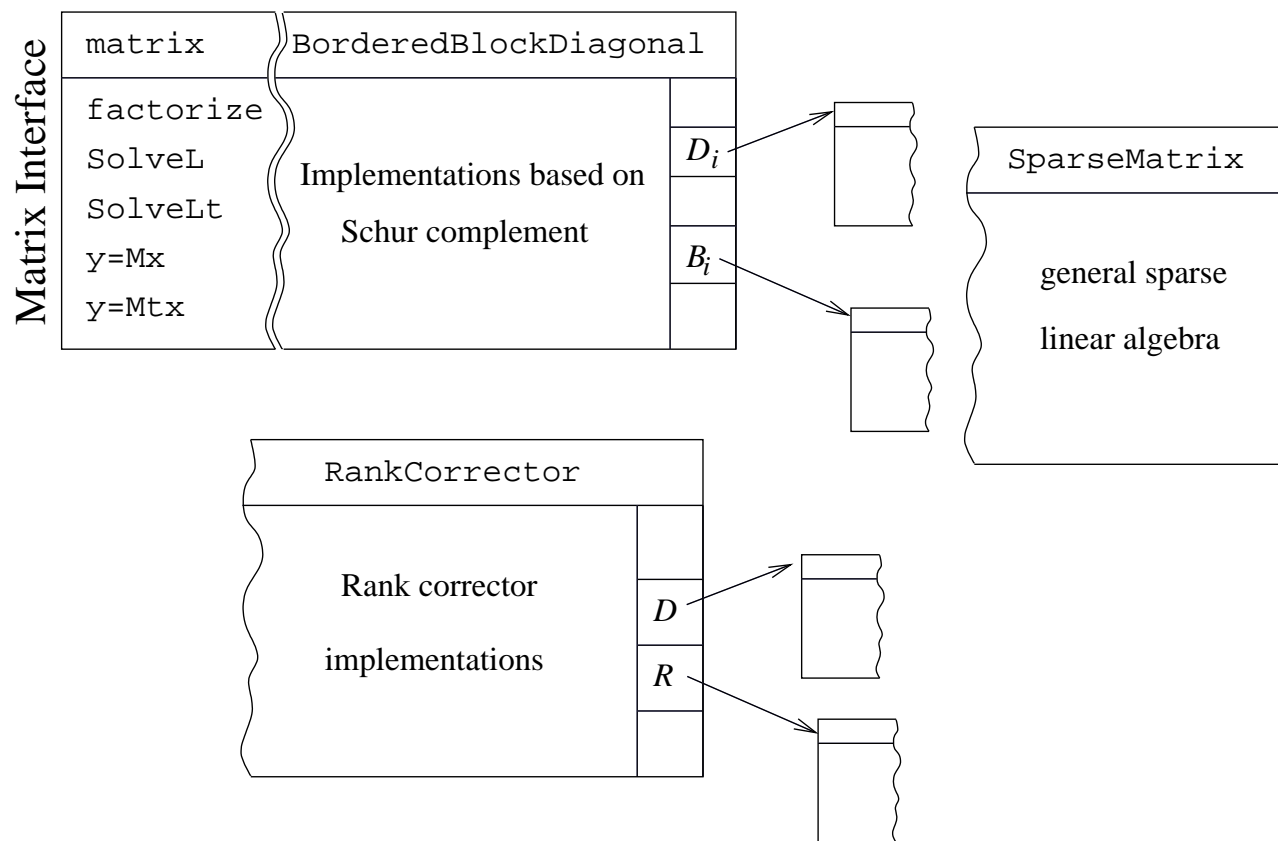
Every **node** needs its own special Linear Algebra implementation

⇒ prohibitive coding cost

Better: **Object Oriented** approach

- Every block of matrix (node) is implementations of abstract **matrix** interface
- **matrix** objects M can be accessed from outside by their **methods**, i.e.
 - Factorize M
 - Solve L , Solve D , Solve L^\top
 - Mx , $M^\top x$
- Each **matrix** object has its own implementation for these methods (most efficient for its structure, exploiting parallelism)
- allow self-referencing

Object-oriented approach:



Rebuild **tree** with matrix interface structures

matrix interface

Set of supported methods (\mathcal{M}):

- Factorize $M = LDL^\top$
- Solve L , Solve D , Solve L^\top
- Matrix-Vector products: Mx , $M^\top x$
- retrieve column/row from M
- set up $\Phi = P \begin{pmatrix} Q & A^\top \\ A & 0 \end{pmatrix} P^{-1}$

closure condition:

Any method in \mathcal{M} can be implemented for any supported structure by only using methods in \mathcal{M} on its sub-blocks

Example: Bordered Block-Diagonal Structure (Schur complement)

$$\underbrace{\begin{pmatrix} \Phi_1 & & & B_1^\top \\ & \dots & & \vdots \\ & & \Phi_n & B_n^\top \\ B_1 & \dots & B_n & \Phi_0 \end{pmatrix}}_{\Phi} = \underbrace{\begin{pmatrix} L_1 & & & \\ & \dots & & \\ & & L_n & \\ L_{1,0} & \dots & L_{n,0} & L_0 \end{pmatrix}}_L \underbrace{\begin{pmatrix} D_1 & & & \\ & \dots & & \\ & & D_n & \\ & & & D_0 \end{pmatrix}}_D \underbrace{\begin{pmatrix} L_1^\top & & & L_{1,0}^\top \\ & \dots & & \vdots \\ & & L_n^\top & L_{n,0}^\top \\ & & & L_0^\top \end{pmatrix}}_{L^\top}$$

- Cholesky-like factors can be obtained by Schur-complement:

$$\begin{aligned} \Phi_i &= L_i D_i L_i^\top & L_{i,0} &= B_i L_i^{-\top} D_i^{-1}, \quad i = 1, \dots, n \\ C &= \Phi_0 - \sum_{i=1}^n L_{i,0} D_i L_{i,0}^\top & C &= L_0 D_0 L_0^\top \end{aligned}$$

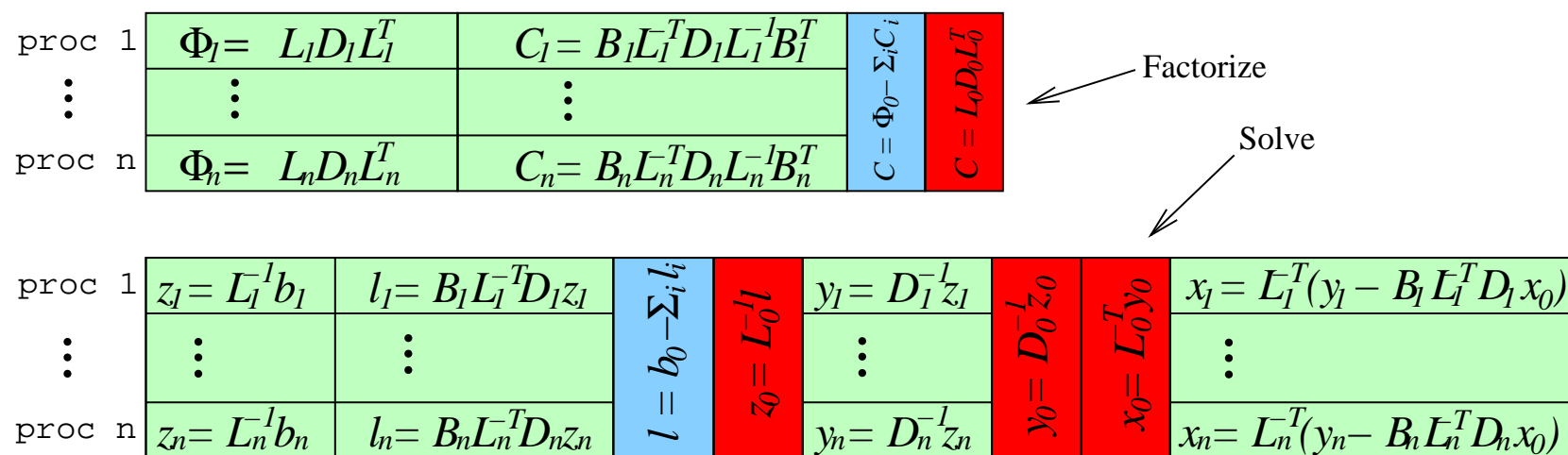
- And the system $\Phi x = b$ can be solved by

$$\begin{aligned} z_i &= L_i^{-1} b_i & x_0 &= L_0^{-\top} y_0 \\ z_0 &= L_0^{-1} (b_0 - \sum L_{i,0} z_i) & x_i &= L_i^{-\top} (y_i - L_{i,0}^\top x_0) \\ y_i &= D_i^{-1} z_i \end{aligned}$$

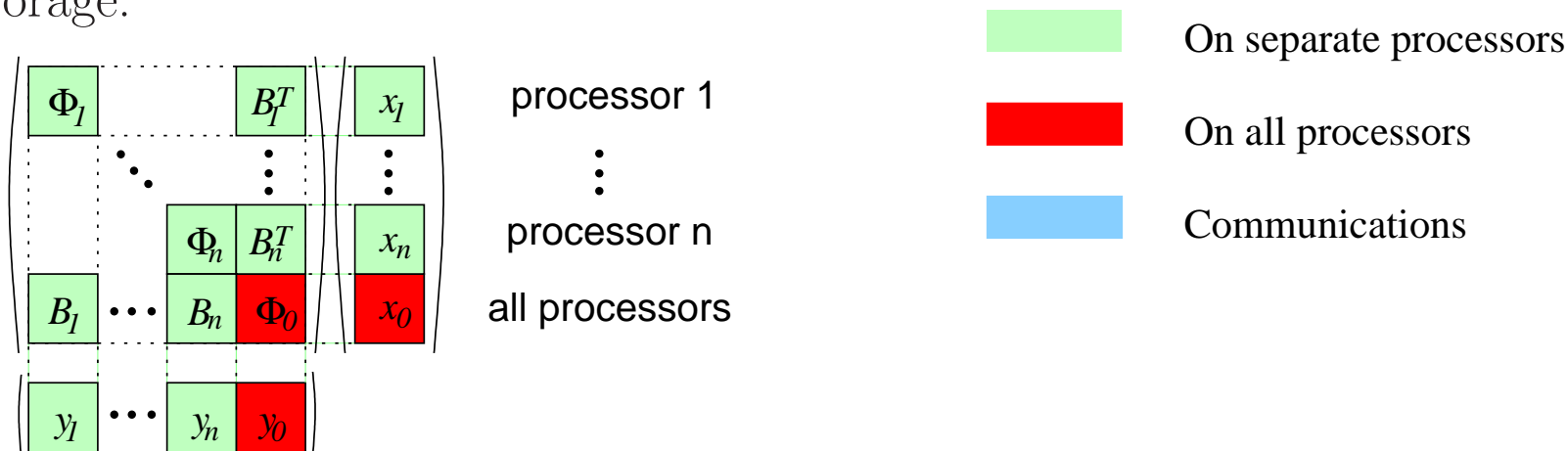
- Operations (Cholesky, Solve, Product) are only performed on sub-blocks
 \Rightarrow **Can also exploit structure in sub-blocks**

Exploiting Parallelism: Bordered Block-Diagonal Structure:

- Distribution of computations:



- Storage:



ALM: Size of largest problem

Optimization of 21 assets (stock market indices)

⇒ Scenario tree needs to capture:

- correlations of 21 assets
- correlations over time periods

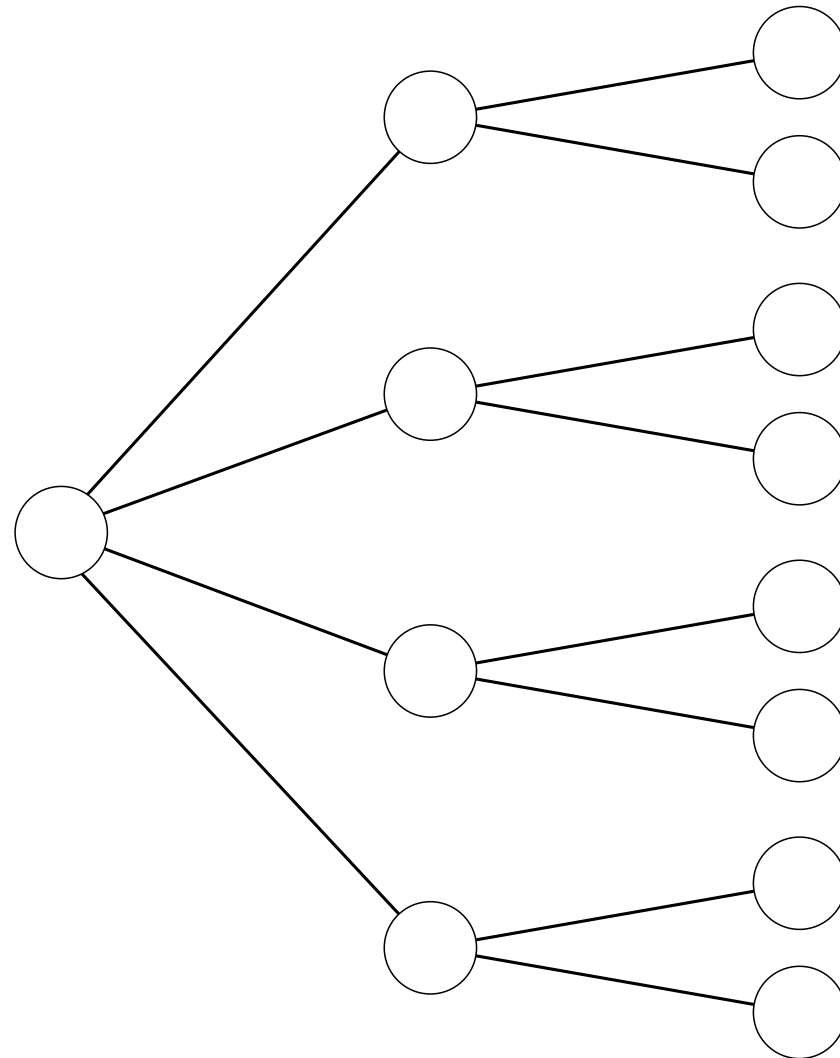
⇒ 100 branches over 6 stages = 10^{12} scenarios

Better:

- Scenario tree geometry: 128-30-16-10-5-4 ⇒ 16 million scenarios.
- Scenario tree generated using simulated geometric brownian motion with mean reversion.
- ⇒ 1.01 billion variables, 353 million constraints

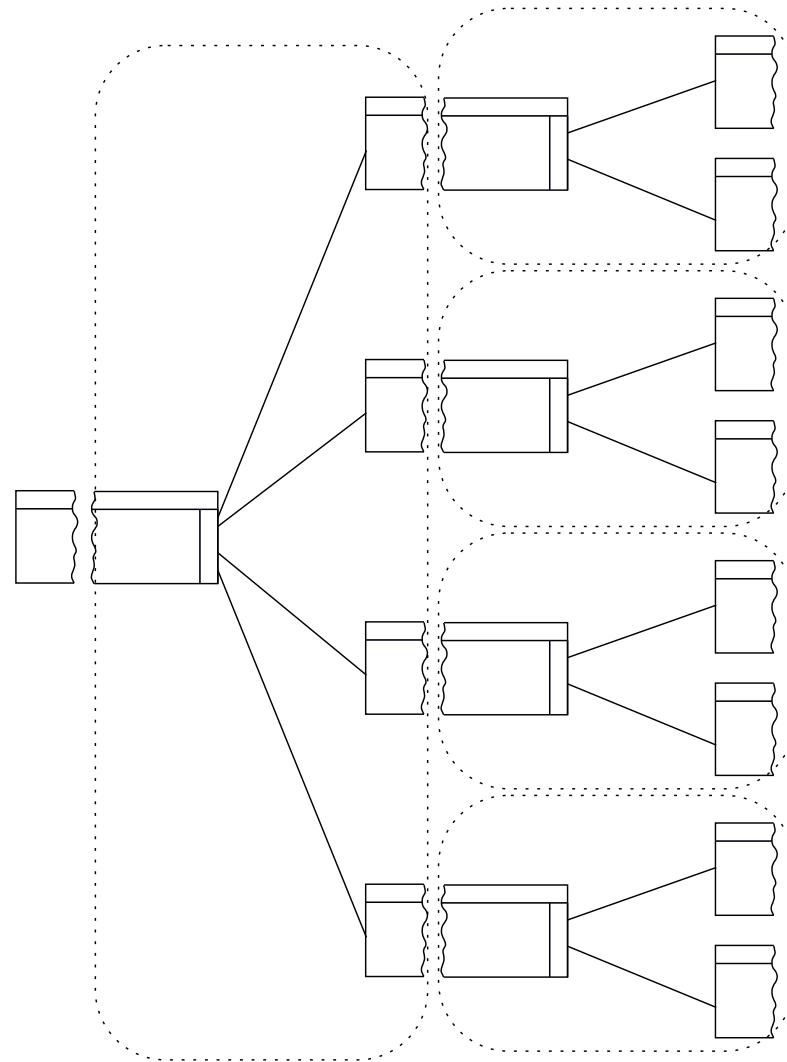
Bootstrapping

- Set up (shape of) Scenario Tree



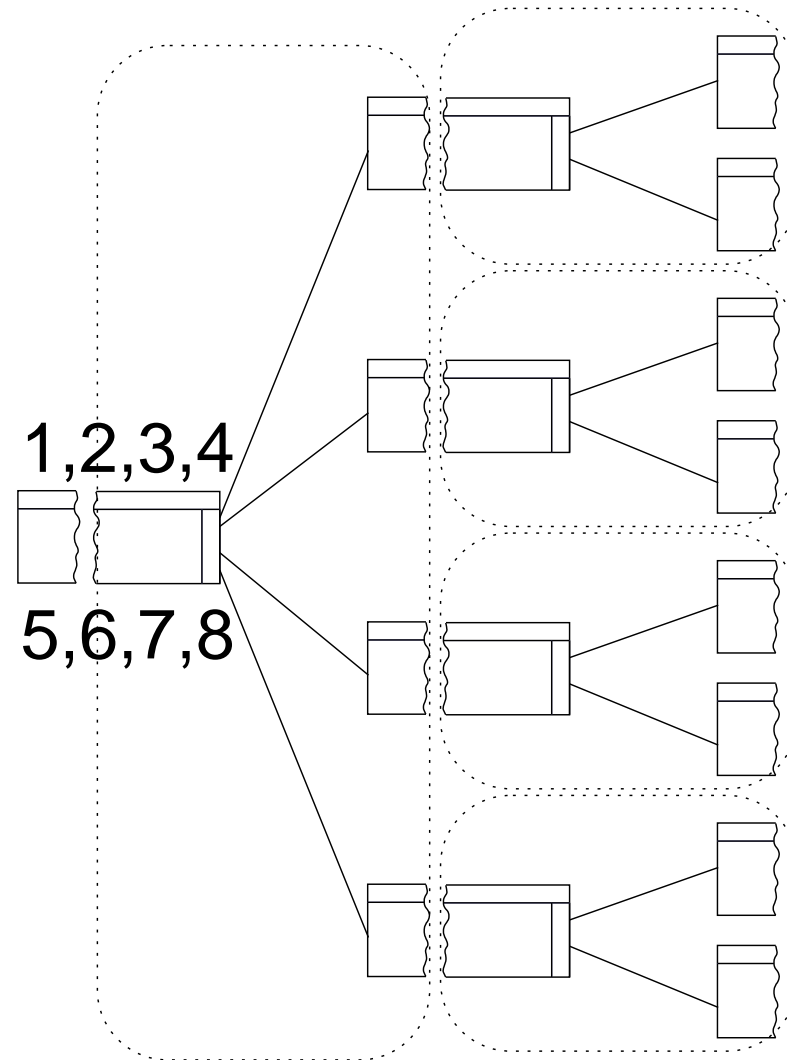
Bootstrapping

- Set up (shape of) Scenario Tree
- Convert to Algebra Tree (without data)



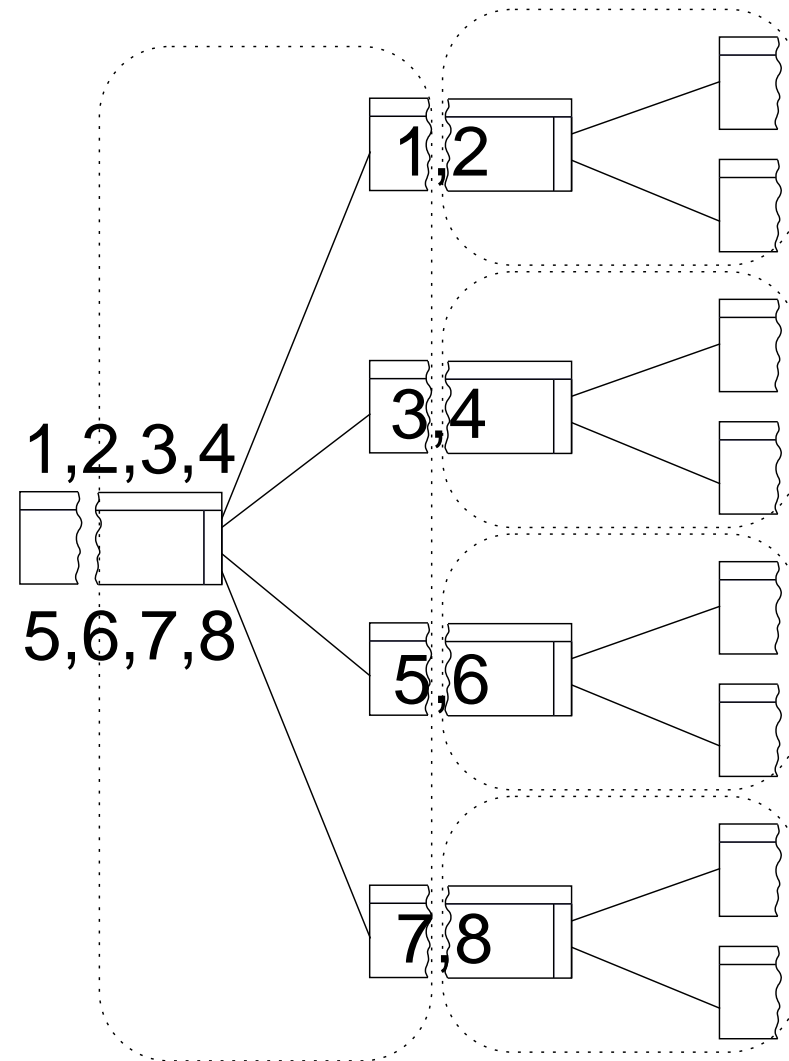
Bootstrapping

- Set up (shape of) Scenario Tree
- Convert to Algebra Tree (without data)
- Assign processors



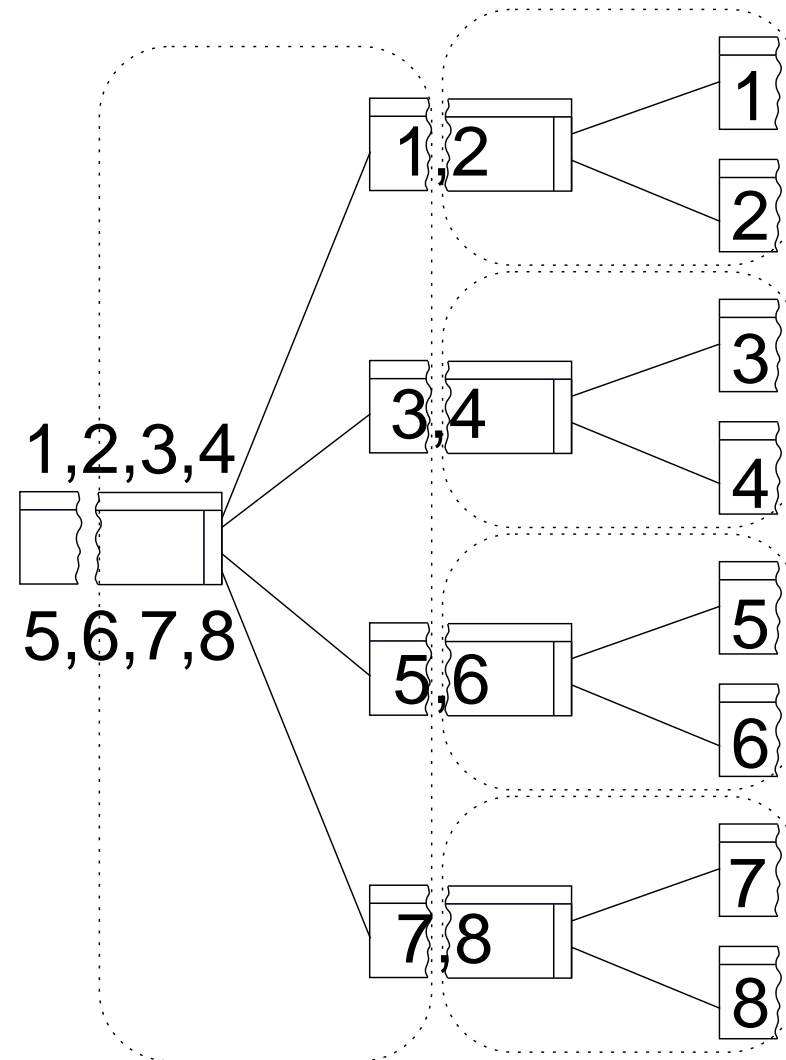
Bootstrapping

- Set up (shape of) Scenario Tree
- Convert to Algebra Tree (without data)
- Assign processors



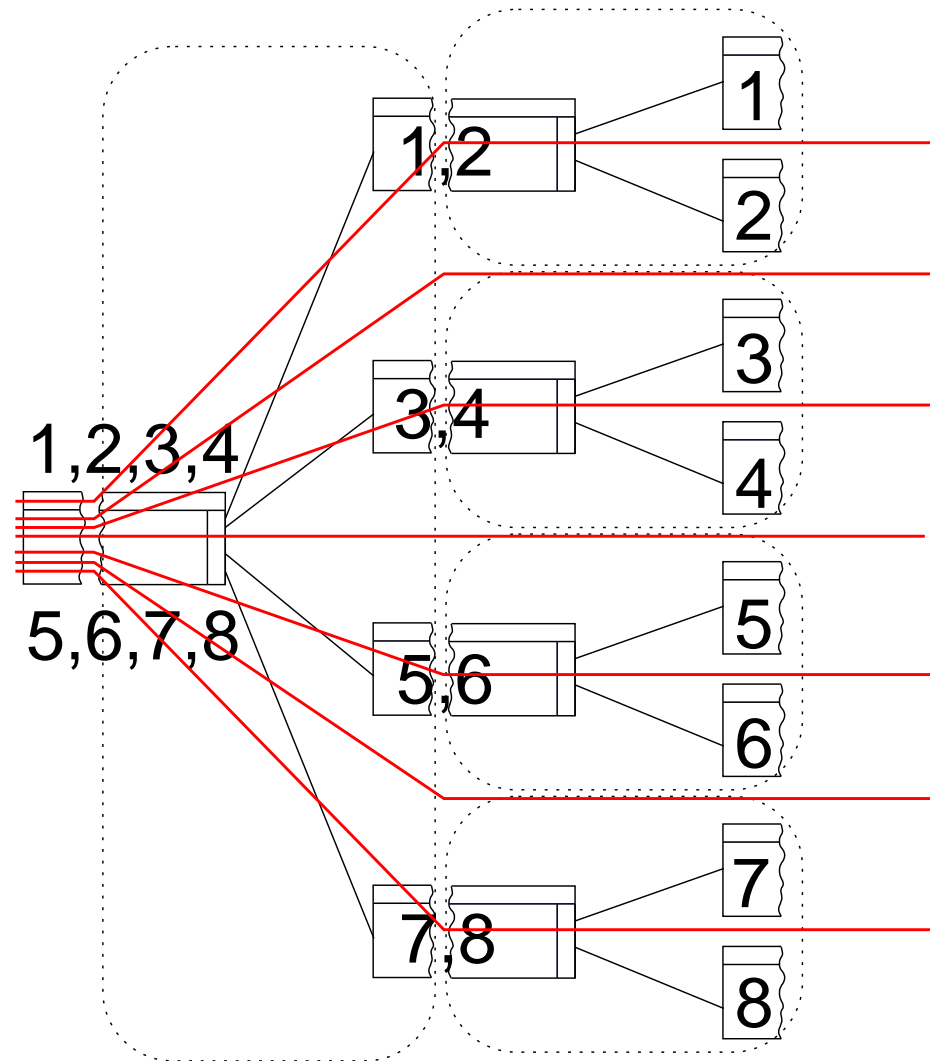
Bootstrapping

- Set up (shape of) Scenario Tree
- Convert to Algebra Tree (without data)
- Assign processors



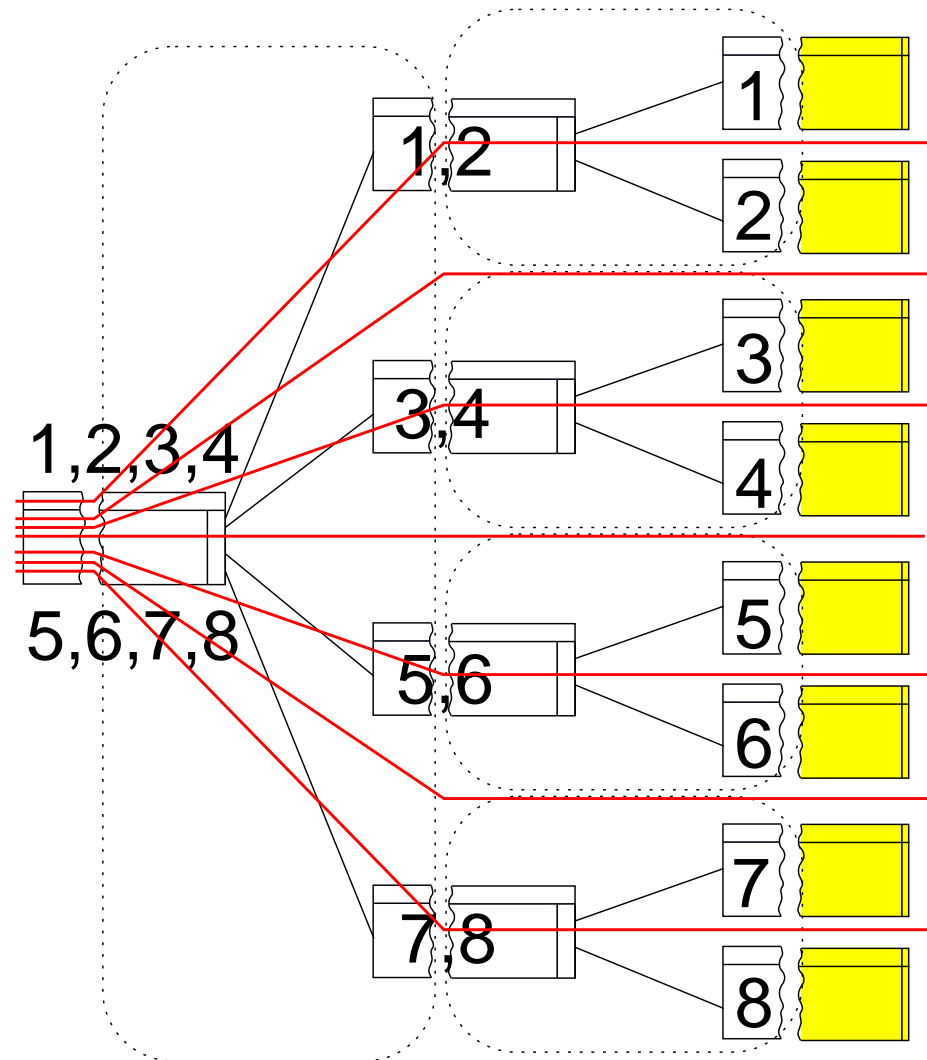
Bootstrapping

- Set up (shape of) Scenario Tree
- Convert to Algebra Tree (without data)
- Assign processors
- Divide between processors



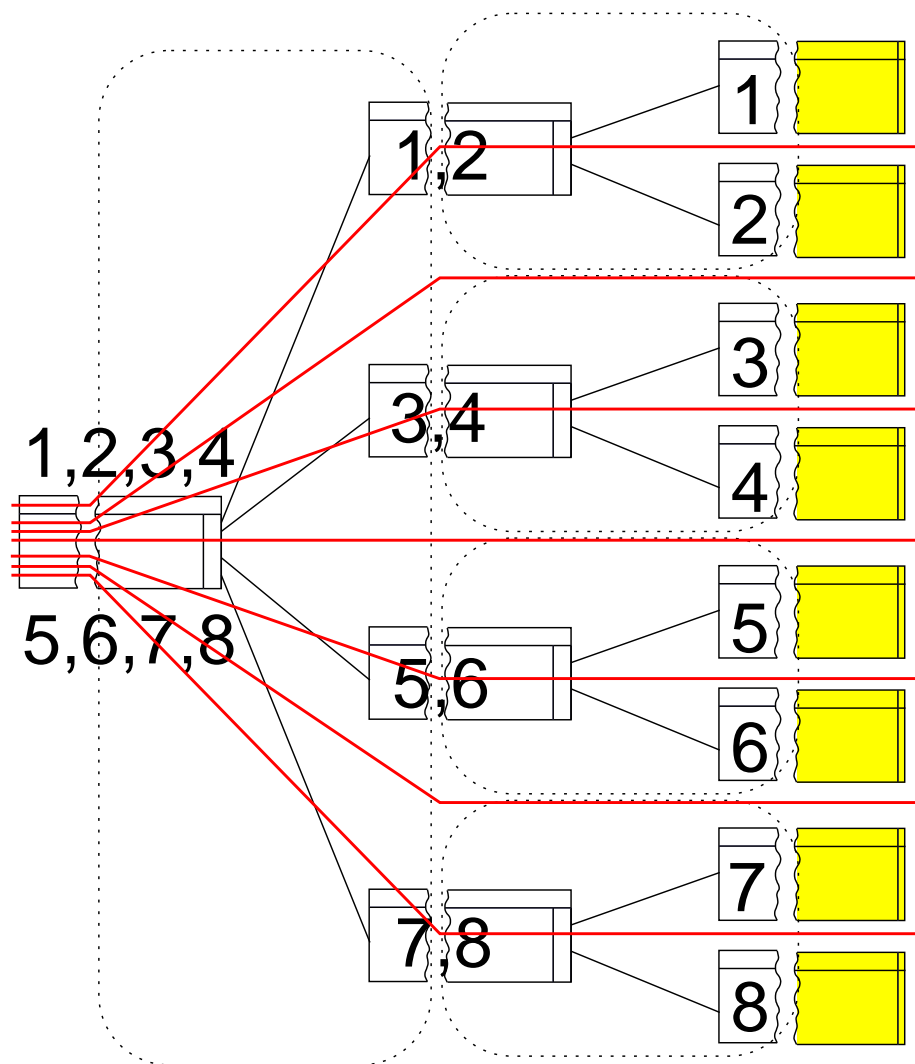
Bootstrapping

- Set up (shape of) Scenario Tree
- Convert to Algebra Tree (without data)
- Assign processors
- Divide between processors
- Load data on each processor (core + generate scenarios)



Bootstrapping

- Set up (shape of) Scenario Tree
- Convert to Algebra Tree (without data)
- Assign processors
- Divide between processors
- Load data on each processor (core + generate scenarios)
- Solve



Advantages of Object-Oriented approach:

- One implementation for each **type** of block: not for each block
- Separates logic of IPM from linear algebra
⇒ Always efficient linear algebra, independent of algorithm
- Automatic coarse grain parallelism
- Linear Algebra module can be used outside IPM

Application to Stochastic Programming

- Can deal with many linear/nonlinear formulations
- Can deal with many types of constraints (linking scenarios, etc).
- Complexity of Linear Algebra is \mathcal{O} (“number of nodes”)

Results (ALM: Mean-Variance QP formulation):

Problem	Stages	Blk	Assets	Scenarios	Constraints	Variables	iter	time	procs	machine
ALM1	5	10	5	11.111	66.667	166.666	14	86	1	SunFire 15K
ALM2	6	10	5	111.111	666.667	1.666.666	22	387	5	“
ALM3	6	10	10	111.111	1.222.222	3.333.331	29	1638	5	“
ALM4	5	24	5	346.201	2.077.207	5.193.016	33	856	8	“
UNS1	5	35	5	360.152	2.160.919	5.402.296	27	872	8	“
ALM5	4	64	12	266.305	3.461.966	9.586.981	18	1195	8	“
ALM6	4	120	5	1.742.521	10.455.127	26.137.816	18	1470	16	“
ALM7	4	120	10	1.742.521	19.167.732	52.275.631	19	8465	16	“

(SunFire 15K: 48 procs, 700MHz, 48GB shared memory)

Results (Nonlinear Variants: Problem Statistics)

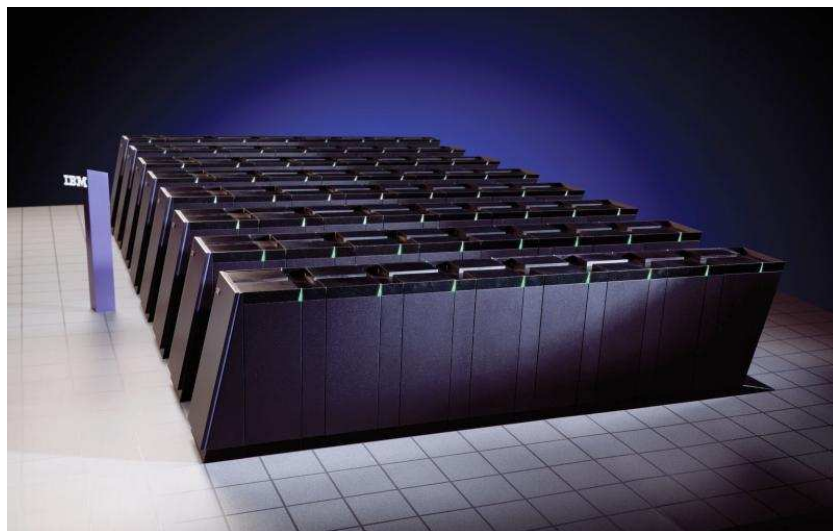
Problem	Stages	Blk	Assets	Total Nodes	Constraints	Variables
ALM1	3	70	40	4971	208,713	606,322
ALM2	4	24	25	14425	388,876	1,109,525
ALM3	4	55	20	169456	3,724,953	10,500,112

Results (Nonlinear Variants)

Problem	1 proc		2 procs		4 procs		8 procs	
	iter	time (s)	time (s)	pe	time (s)	pe	time (s)	pe
variant 1: semi-variance								
ALM1	35	568	258	1.10	141	1.01	92	0.76
ALM2	30	1073	516	1.04	254	1.05	148	0.91
ALM3	43	18799	9391	1.00	4778	0.98	2459	0.96
variant 2: logarithmic utility								
ALM1	25	448	214	1.05	110	1.02	72	0.78
ALM2	31	1287	618	1.04	306	1.05	179	0.90
ALM3	60	24414	12480	0.98	6275	0.97	3338	0.91
variant 3: skewness								
ALM1	50	820	390	1.05	208	1.02	130	0.79
ALM2	43	1466	715	1.03	396	0.93	207	0.89
ALM3	62	23664	11963	0.99	6131	0.97	3097	0.96

Comparison with CPLEX 9.1

Problem	Constraints	Variables	Blk	Stg	CPLEX 9.1		OOPS	
					time	memory	time	memory
C33	57,274	168,451	33	4	292	497MB	344	156MB
C50	130,153	382,801	50	4	1361	1.3GB	828	345MB
C70	253,522	745,651	70	4	(5254)	OoM	1627	664MB

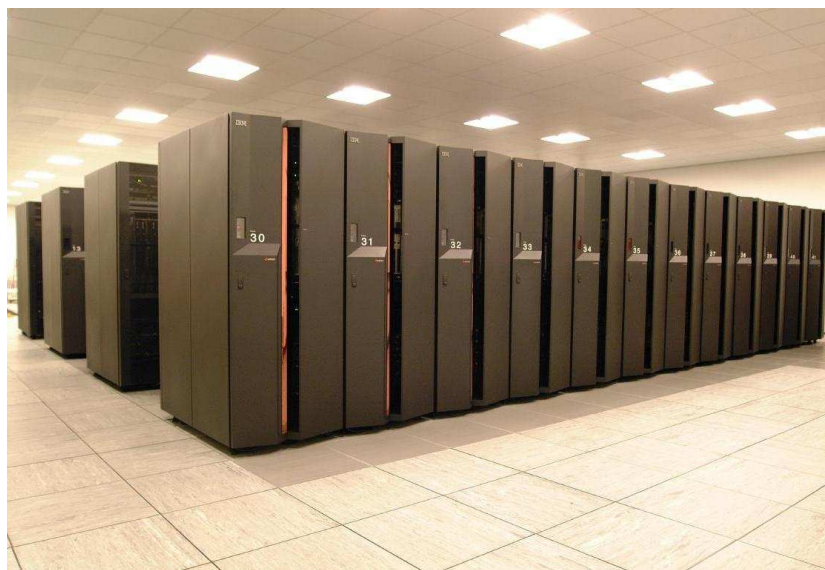


BlueGene/L (Edinburgh, Scotland)

- 2048 Processors
- 0.7GHz, 256Mb
- $R_{max} = 4.7$ TFlops
- #148 in top500.org list

HPCx (Daresbury, England)

- 1600 IBM Power-4 Processors
- 1.7GHz, 800Mb
- $R_{max} = 6.2$ TFlops
- \approx #100 in top500.org list



Results

Stages	Blk	Assets	Scenarios	Constraints	Variables	iter	time	procs	machine
7	128	6	12,831,873	64,159,366	153,982,477	42	3923	512	BG/L
7	64	14	6,415,937	96,239,056	269,469,355	39	4692	512	BG/L
7	128	13	12,831,873	179,646,223	500,443,048	45	6089	1024	BG/L
7	128	21	16,039,809	352,875,799	1,010,507,968	53	3020	1280	HPCx

Results

Stages	Blk	Assets	Scenarios	Constraints	Variables	iter	time	procs	machine
7	128	6	12,831,873	64,159,366	153,982,477	42	3923	512	BG/L
7	64	14	6,415,937	96,239,056	269,469,355	39	4692	512	BG/L
7	128	13	12,831,873	179,646,223	500,443,048	45	6089	1024	BG/L
7	128	21	16,039,809	352,875,799	1,010,507,968	53	3020	1280	HPCx

Parallel Efficiency (strong scaling)

nodes	peak Mem	time	Comm	Cholesky	Solves	MatVectProd
16	426MB	2587 (1.00)	24	1484 (1.00)	956 (1.00)	28.8 (1.00)
32	232MB	1303 (0.99)	13	743 (1.00)	485 (0.98)	18.0 (0.80)
64	132MB	688 (0.94)	6	377 (0.98)	270 (0.88)	13.0 (0.55)
128	84MB	348 (0.93)	3	187 (0.99)	139 (0.86)	9.0 (0.40)
256	56MB	179 (0.90)	3	93 (0.99)	73 (0.82)	5.8 (0.31)
512	46MB	94 (0.86)	2	47 (0.98)	39 (0.76)	3.9 (0.23)

Conclusions:

- OOPS is a general purpose IPM solver with object-oriented linear algebra
- Will scale up to massively parallel architecture
- Can solve problems with 10^9 variables using direct factorization methods
- Allows the solution of realistic financial planning problems

Current/Future Work on OOPS:

- Warmstarting by scenario aggregation
→ *Marco Colombo (TC2)*
- Incorporation of iterative solvers (structured pre-conditioners)
- Integrate into structured modelling language

Object-Oriented Parallel Solver (OOPS):

References:

- M. Colombo, J. Gondzio and A. Grothey, *A Warm-Start Approach for Large-Scale Stochastic Linear Programs*, Tech. Rep. MS-06-004, School of Maths, University of Edinburgh, 2006.
- J. Gondzio and A. Grothey, *Parallel interior point solver for structured quadratic programs: application to financial planning problems*, Annals of OR 152 (2007), Vol 1, pp 319–339.
- J. Gondzio and A. Grothey, *Solving nonlinear portfolio optimization problems with the primal-dual interior point method*, European Journal of Operational Research, 181 (2007), Vol 3, p.1019-1029.
- J. Gondzio and A. Grothey, *Exploiting Structure in Parallel Implementation of Interior Point Methods for Optimization*, Tech. Rep. MS-04-004, School of Maths, University of Edinburgh, Dec 2004.
- J. Gondzio and A. Grothey, *Direct Solution of Linear Systems of Size 10^9 Arising in Optimization with Interior Point Methods*, in: Parallel Processing and Applied Mathematics, Lecture Notes in Computer Sciences 3911.

Further Developments: Warmstarting

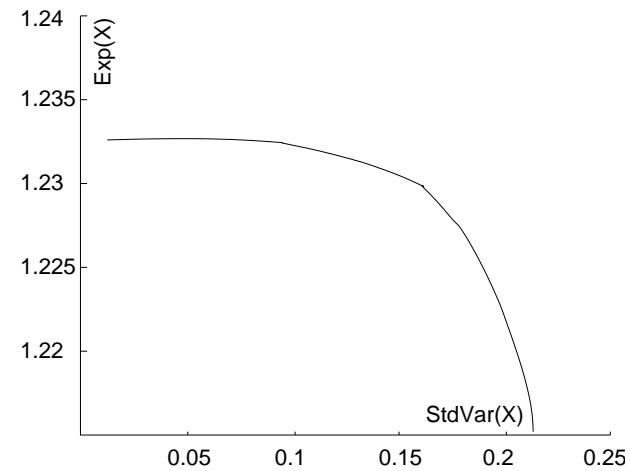
Possibilities for warmstarting are rarely used in Stochastic Programming

- Solution approaches for SP (Benders Decomposition/IPM) don't warmstart well
- “Interior Point Methods cannot be warmstarted”

Results (Efficient Frontier)

Mean Variance formulation:

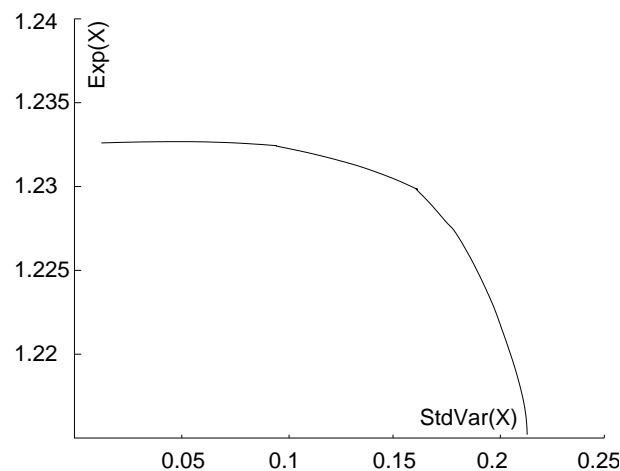
$$\max \mathbf{IE}(X) - \rho \text{Var}(X)$$



Results (Efficient Frontier)

Mean Variance formulation:

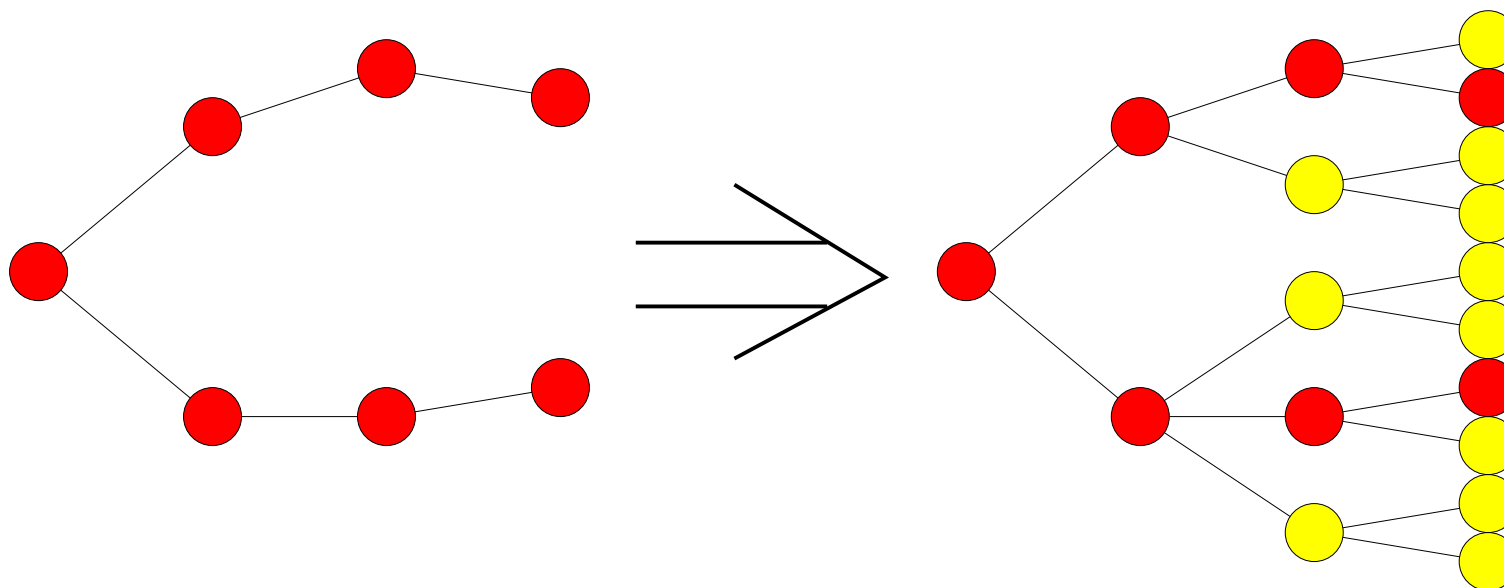
$$\max \mathbf{IE}(X) - \rho \text{Var}(X)$$



constraints	variables	procs	$\rho = 0.001$	0.005	0.01	0.05	0.1	0.5	1	5	10
223,321	76,881	1	14	14	14	14	14	13	17	16	17
			14	5	5	5	4	5	5	8	8
533,725	198,525	1	14	14	14	14	14	15	18	18	17
			14	5	5	5	6	5	5	9	10
5,982,604	16,316,191	32	24	23	24	23	25	22	24	23	24
			24	8	11	13	11	13	12	12	14
70,575,308	192,478,111	512	52	53	45	43	44	42	44	46	46
			52	13	13	15	15	16	16	23	25

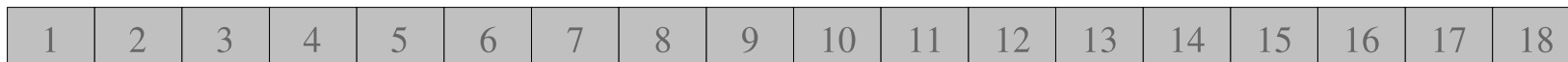
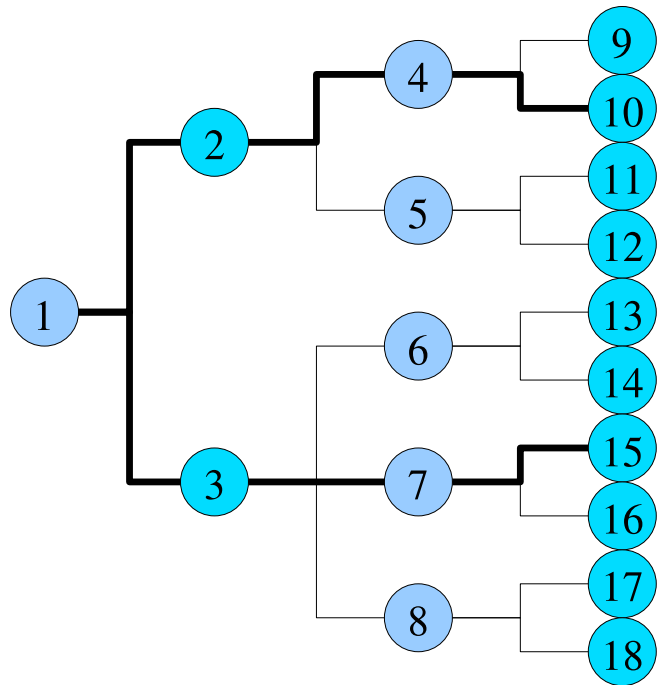
⇒ 50%-60% of iterations saved on average

Idea: Use Warmstart to speed up solution of Stochastic Program

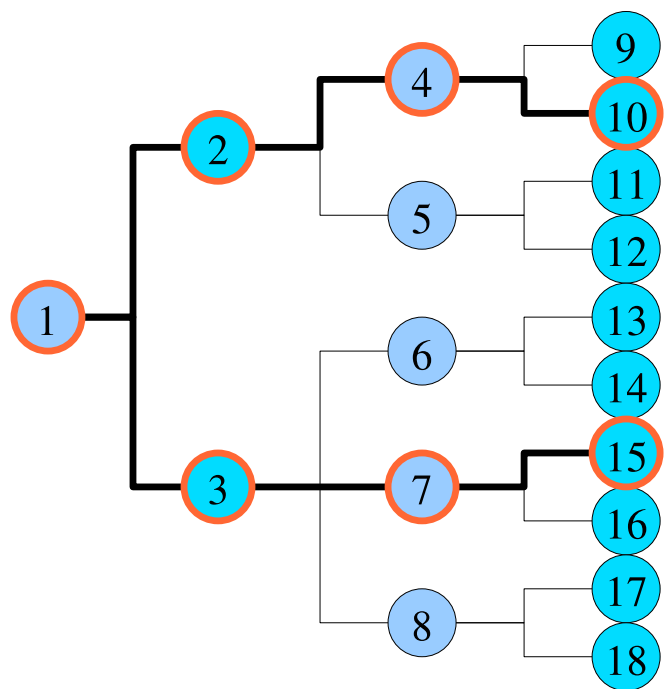


- Solve problem on a **reduced** scenario tree first
 - Copy solution vectors from **nearby** scenarios to construct starting point for **full** problem.
 - Solve full problem (using this starting point)
- ⇒ Saves up to 60% of iterations on problem up to 4096 scenarios.

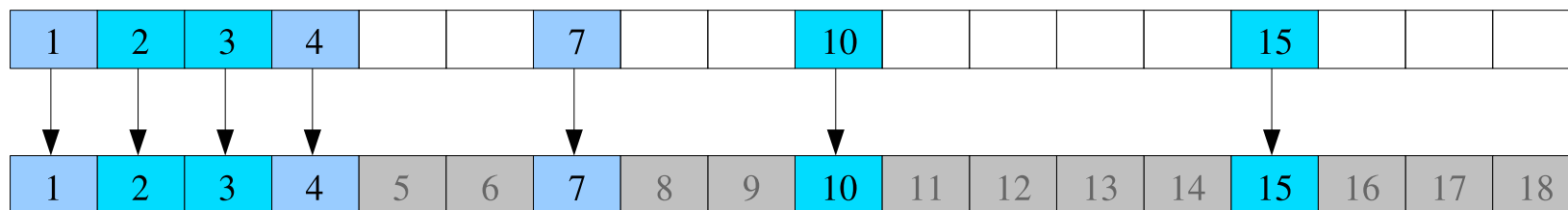
Construction of the warm-start iterate



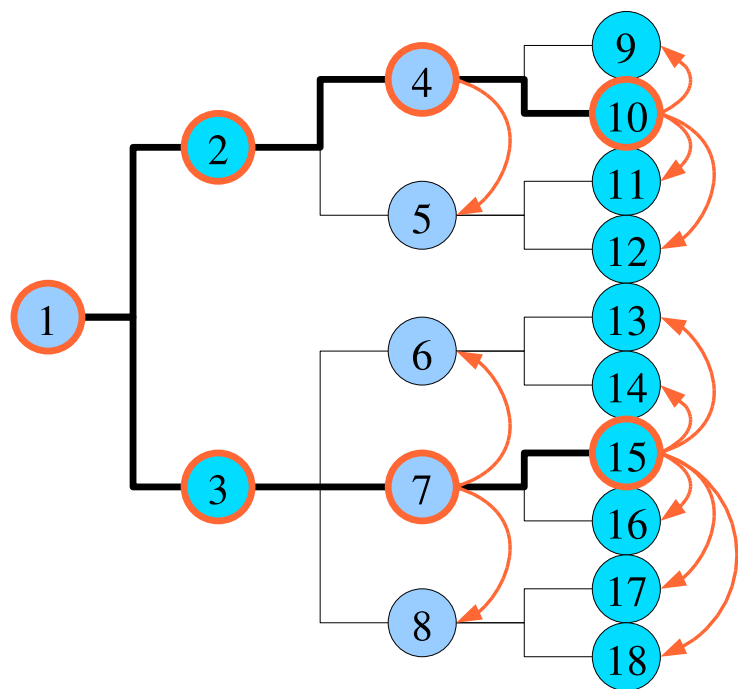
Construction of the warm-start iterate



Nodes in the reduced tree:
the solution is already available



Construction of the warm-start iterate



Nodes in the reduced tree:
the solution is already available

Remaining nodes:
copy the solution from the corresponding
reduced-tree node

