SMS++: a system for structured optimization, with applications

Antonio Frangioni[1] Rafael Durbano Lobato[2] Wim van Ackooij[3]

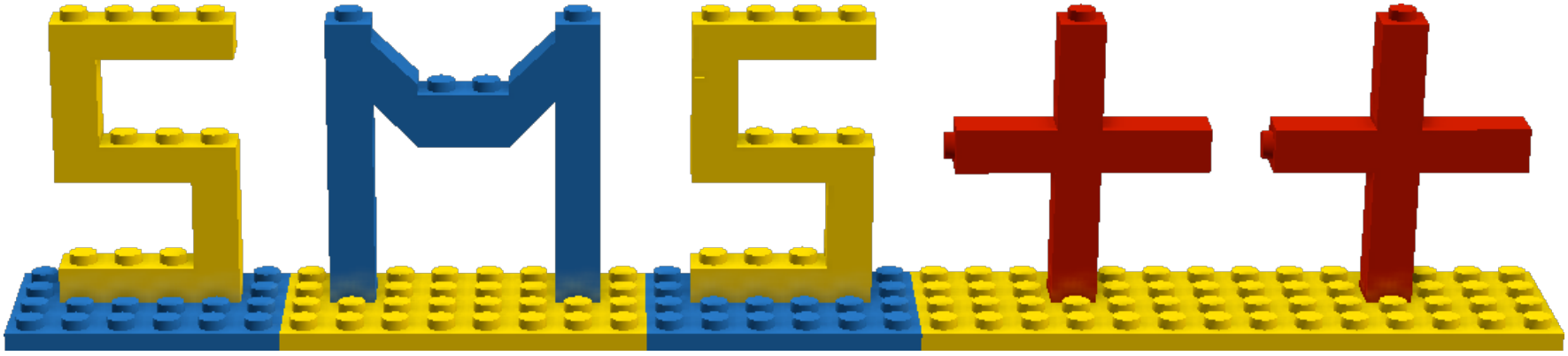
[1]Dipartimento di Informatica, Università di Pisa

[2]Chamber of Electric Energy Commercialization, Brazil

[3]EDF R&D OSIRIS

May the 17th 2023, Montréal

Outline



https://gitlab.com/smspp/smspp-project

"For algorithm developers, from algorithm developers"

- Open source (LGPL3)
- 1 "core" repo, 1 "umbrella" repo, 10+ problem and/or algorithmic-specific repos (public, more in development)
- Extensive Doxygen documentation https://smspp.gitlab.io
- But no real user manual as yet (except for myself)

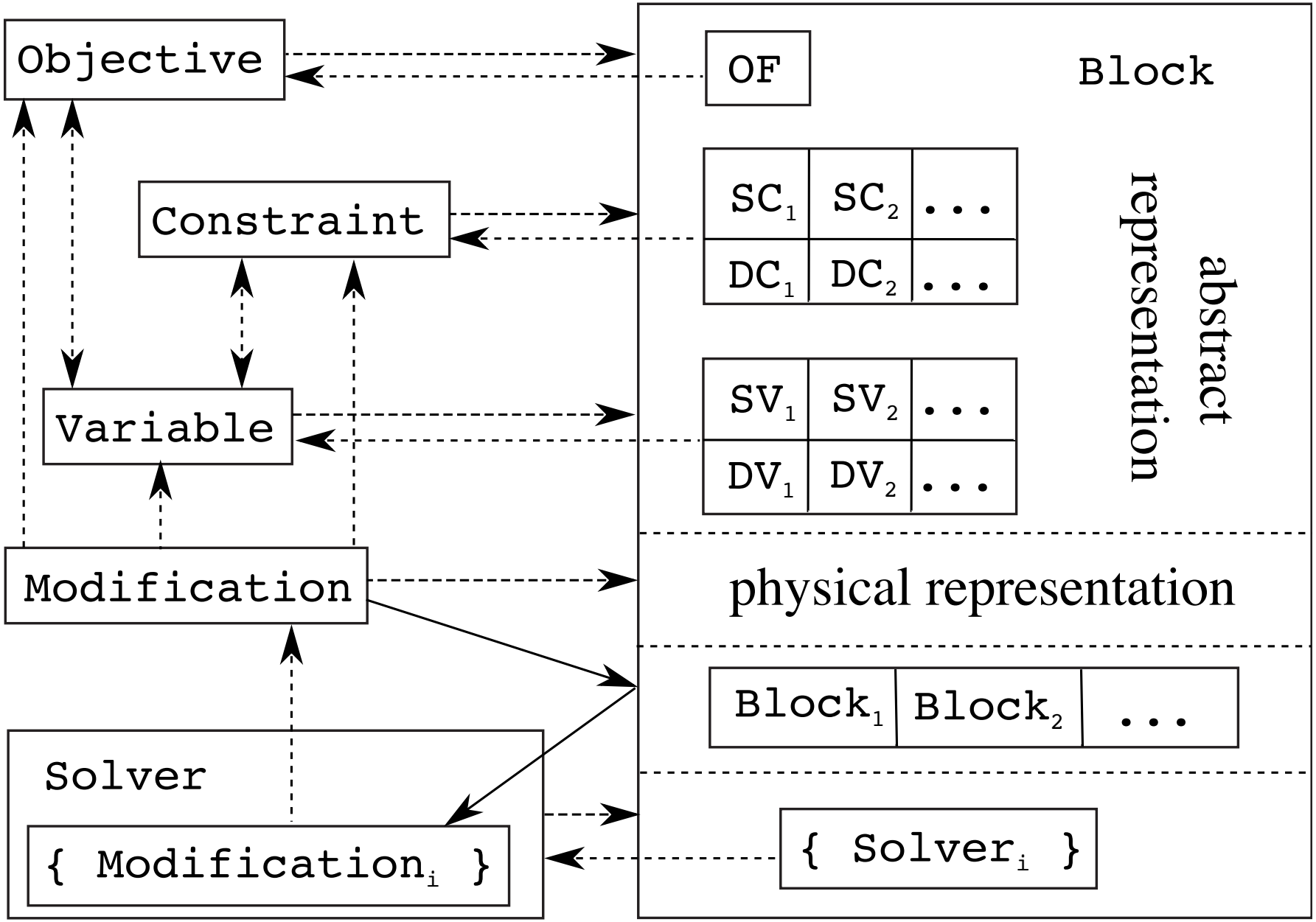
What SMS++ is

- A core set of C++-17 classes implementing a modelling system that:
  - explicitly supports the notion of `Block` ≡ nested structure
  - separately provides "semantic" information from "syntactic" details (list of constraints / variables ≡ one specific formulation among many)
  - allows exploiting specialised `Solver` on `Block` with specific structure
  - manages any dynamic change in the `Block` beyond "just" generation of constraints/variables
  - supports reformulation / restriction / relaxation of `Block`
  - has built-in parallel processing capabilities
  - should be able to deal with almost anything (bilevel, PDE, . . .)
- A hopefully growing set of specialized `Block` and `Solver`
- In perspective an ecosystem fostering collaboration and code sharing: a community-building effort as much as a (suite of) software product(s)

What SMS++ is not

- An algebraic modelling language: `Block` are C++ code
  (although it provides some modelling-language-like functionalities)
- For the faint of heart: primarily written for algorithmic experts
  (although users may benefit from having many pre-defined `Block`)
- Stable: only version 0.5.2 (as of tonight), lots of further development ahead, significant changes in (parts) of interfaces actually expected
  (although current `Block` / `Solver` very thoroughly tested)
- Interfaced with many existing solvers: `Cplex`, `SCIP`, `MCFClass`, `StOpt`
  (although the list is growing, `Gurobi` and `HiGHS` in the works)
- Ripe with native structure-exploiting solvers: `LagrangianDualSolver`, `SDDPSolver` (although the list should hopefully grow)

A Crude Schematic



Block

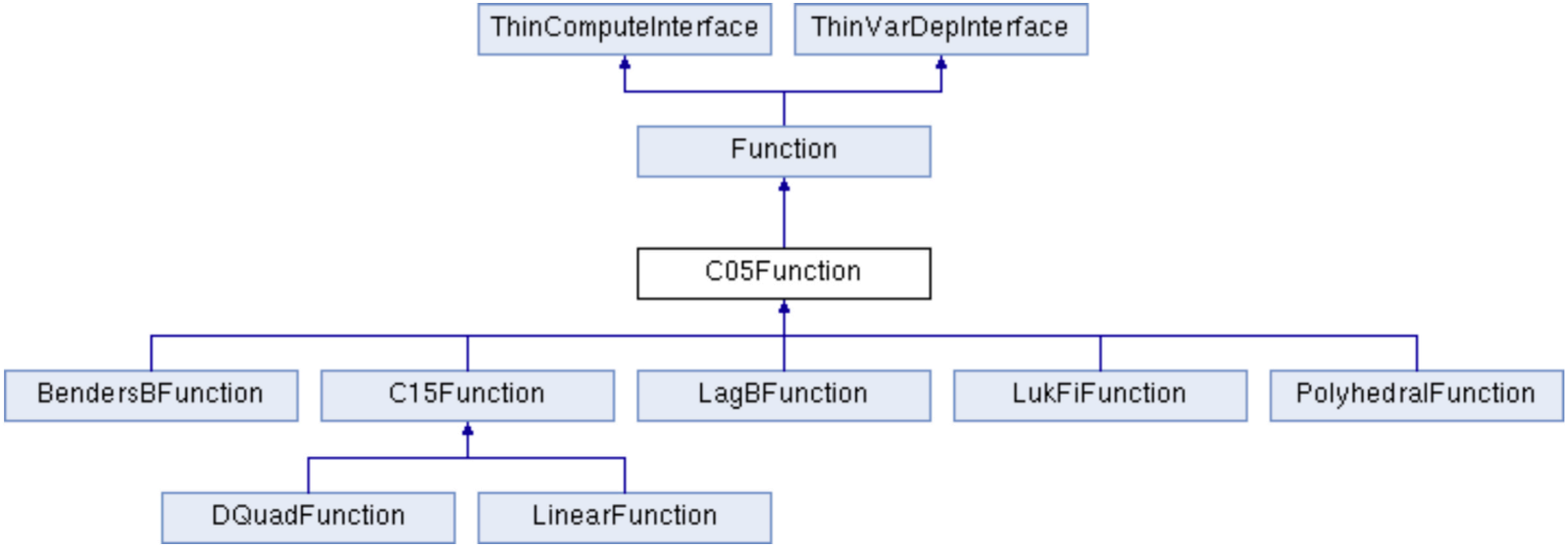
- `Block` = abstract class representing the general concept of "a (part of a) mathematical model with a well-understood identity"
- Each `:Block` a model with specific structure
  (e.g., `MCFBlock:Block` = a Min-Cost Flow problem)
- Physical representation of a `Block`: whatever data structure is required to describe the instance (e.g., G, b, c, u)
- Possibly alternative abstract representation(s) of a `Block`:
  - one `Objective` (but possibly vector-valued)
  - any # of groups of static/dynamic `Variable` / `Constraint`
  - a group = a single, `std::vector` or `boost::multi_array` (of `std::list` for dynamic)

  (e.g., arc-flow formulation vs. path-flow formulation)
- Any # of sub-`Blocks` (recursively), possibly of specific type
  (e.g., `MMCFBlock:Block:` has k `MCFBlock:Block` inside)
- `generate_abstract_[variables/constraints]()` for column/row generation (with `Configuration` for options)

Variable, Constraint, Objective

- Abstract concepts, thought to be extended (a matrix, a function, . . . , a matrix constraint, a PDE constraint, a bilevel program, . . .)
- `Variable` does not even have a value, can be fixed and unfixed
- `Constraint`, `Objective` depends from a set of `Variable` (`:ThinVarDepInterface`), must be `compute()`-d (possibly costly, `:ThinComputeInterface`)
- `Constraint` can be relaxed and enforced
- `RealObjective:Objective` implements "value is an extended real"
- `Objective` of sub-`Blocks` summed to that of father `Block`
- Anyone knows which `Block` it belongs to
- Fundamental design decision: "name" of anything = its memory address
  $\Longrightarrow$ copying something makes a different something
  $\Longrightarrow$ dynamic somethings always live in `std::lists`
- `:Modification` for changes (fix / unfix, relax / enforce, max / min, . . .)

Function



- Real-valued `Function` of a set of `Variable` (`:ThinVarDepInterface`)
- Must be `compute()`-d w.r.t. the current value of the `Variable`, possibly a costly operation (`:ThinComputeInterface`) $\Longrightarrow$ approximate computation supported in a quite general way
- `FunctionModification[Variables]` for "easy" changes $\Longrightarrow$ reoptimization (shift, adding/removing "quasi separable" `Variable`)
- `C05Function`/`C15Function` deal with $1^{st}/2^{nd}$ order information (not necessarily continuous)

Closer to the ground

- `ColVariable:Variable`: "value = one single real" (possibly $\in \mathbb{Z}$)
- `RowConstraint:Constraint`: "$l \leq$ a real $\leq u$" $\Longrightarrow$
  has dual variable (single real) attached to it
- `OneVarConstraint:RowConstraint`: "a real" $=$
  a single `ColVariable` $\equiv$ bound constraints
- `FRowConstraint:RowConstraint`: "a real" given by a `Function`
- `FRealObjective:RealObjective`: "value" given by a `Function`
- `LinearFunction:Function`: a linear form in `ColVariable`
- `DQuadFunction:Function`: a separable quadratic form
- Many things missing (`AlgebraicFunction`, `DenseLinearFunction`, `Matrix/VectorVariable`, ...)

Solver

- Any # of `Solver` attached to a `Block` (and its sub-`Block`) to solve it
- Specialised `:Solver` for specific `:Block` use the physical representation
  $\Longrightarrow$ abstract representation of `Block` only constructed on demand
- General-purpose `:Solver` rather uses the abstract representation
- `Variable` always present to for `Solver` to write solution in
  (this may change with physical solution concept, under development)
- Tries to cater for all the important needs:
  - optimal and sub-optimal solutions, provably unbounded/unfeasible
  - time/resource limits for solutions, but restarts (reoptimization)
  - any # of multiple solutions produced on demand
  - lazily reacts to changes in the data of the `Block` via `Modification`
- Slanted towards `RealObjective` ($\approx$optimality = up/low-er bounds)
- `CDASolver:Solver`: bounds are associated to (multiple) dual solutions
- Provides general events mechanism (`ThinComputeInterface` does)

Modification

- Any change in Block is communicated to each interested Solver (attached to the Block or any of its ancestor) via a Modification
- Two different kinds of Modification (what changes):
  - physical Modification, only specialized Solver concerned
  - abstract Modification, only Solver using it concerned
- Abstract Modification used to keep both representations in sync
- A Block declares which abstract changes it supports
- Heavy stuff can be attached to a Modification (e.g., added / deleted dynamic Variable / Constraint) deleted when last Solver finishes (smart pointers)
- Solver supposedly reoptimize to improve efficiency, which is easier if you can see all list of changes at once (lazy update)
- GroupModification to (recursively) pack many Modification together $\Longrightarrow$ different "channels" in Block

Support to (coarse-grained) Parallel Computation

- `Block` can be (r/w) `lock()`-ed and `read_lock()`-ed
- `[...]lock()`-ing a `Block` recursively `[...]lock()`s all inner `Block`
- `lock()` (but not `read_lock()`) sets an owner and records its `std::thread::id`; other `lock()` from the same thread fail
- Similar mechanism for `read_lock()`, any # of concurrent reads
- Write starvation not handled yet
- A `Solver` can be "lent an ID" (solving an inner `Block`)
- `Solver`'s set of `Modification` under "active guard" (`std::atomic`)
- General `State` of `Solver` for checkpointing (and reoptimization)
- New `Change` concept: `Modification` + data, can be de/serialize-d, undo-`Change` can be produced when `apply()`-ed to `Block`
- Distributed computation under development, can exploit general de/serialize `Block` / `Change` capabilities

R^{3}Block

- Often reformulation crucial, but also relaxation or restriction: `get_R3_Block()` produces one, possibly using sub-`Blocks`'
- Obvious special case: copy (clone) should always work
- Available R^{3}Blocks `:Block`-specific, a `:Configuration` needed
- R^{3}Block completely independent (new `Variable` / `Constraint`), useful for algorithmic purposes (branch, fix, solve, ...)
- Solution of R^{3}Block useful to `Solver` for original `Block`: `map_back_solution()` (best effort in case of dynamic `Variable`)
- Sometimes keeping R^{3}Block in sync with original necessary: `map_forward_Modification()`, task of original `Block`
- `map_forward_solution()` and `map_back_Modification()` useful, e.g., dynamic generation of `Variable` / `Constraint` in the R^{3}Block
- `:Block` is in charge of all this, thus decides what it supports

A lot of other support stuff

- `Block` produces `Solution` object, possibly using its sub-`Block`s', that can be stored and (linearly) combined
- `[C/O/R]BlockConfiguration` and `[R]BlockSolverConfiguration`: tree-structured objects (as `Block`) to configure the `Block`, register and configure all its `Solver`, cleanup everything in one blow
- Almost everything (`Block`, `Configuration`, `Solver`, `Change`, ...) has a factory and/or methods to de/serialize themselves to `netCDF` files
- A methods factory for changing the physical representation without knowing of which `:Block` it exactly is (standardised interface)
- `AbstractBlock` for constructing a model a-la algebraic language, can be derived for "general `Block` + specific part"
- `PolyhedralFunction[Block]`, very useful for decomposition
- `AbstractPath` for indexing any `Constraint` / `Variable` in a `Block`
- `FakeSolver:Solver` stashes away all `Modification`, `UpdateSolver:Solver` immediately forwards/R^{3}Bs them

Main Existing :Block

- MCFBlock / MMCFBlock: single/multicommodity flow
- BinaryKnapsackBlock (actually mixed-integer)
- CapacitatedFacilityLocationBlock (didactic)
- UCBlock for UC, several UnitBlock and NetworkBlock for components
- LagBFunction:{C05Function,Block} transforms any Block (with appropriate Objective) into its dual function
- BendersBFunction:{C05Function,Block} transforms any Block (with appropriate Constraint) into its value function
- StochasticBlock implements realizations of scenarios into any Block (using methods factory)
- SDDPBlock represents multi-stage stochastic programs suitable for Stochastic Dual Dynamic Programming
- Others under active development (but not released yet)

Main "Basic" :Solver

- MCFSolver: templated wrapper to MCFClass[1] for MCFBlock
- DPBinaryKnapsackSolver, nontrivial support for reoptimizations
- ThermalUnitDPSolver for ThermalUnitBlock (state-of-the-art)
- MILPSolver: constructs matrix representation of any "MILP" Block + CPXMILPSolver:MILPSolver and SCIPMILPSolver:MILPSolver wrappers for Cplex and SCIP (to be improved)
- CDASolver:[Parallel]BundleSolver: SMS++-native version of[2] (still shares some code, dependency to be removed), optimizes any (sum of) C05Function, several (but not all) state-of-the-art tricks
- SDDPSolver: wrapper for SDDP solver StOpt[3] using StochasticBlock, BendersBFunction and PolyhedralFunction
- SDDPGreedySolver: greedy forward simulator for SDDPBlock

---

[1] https://github.com/frangio68/Min-Cost-Flow-Class

[2] https://gitlab.com/frangio68/ndosolver_fioracle_project

[3] https://gitlab.com/stochastic-control/StOpt

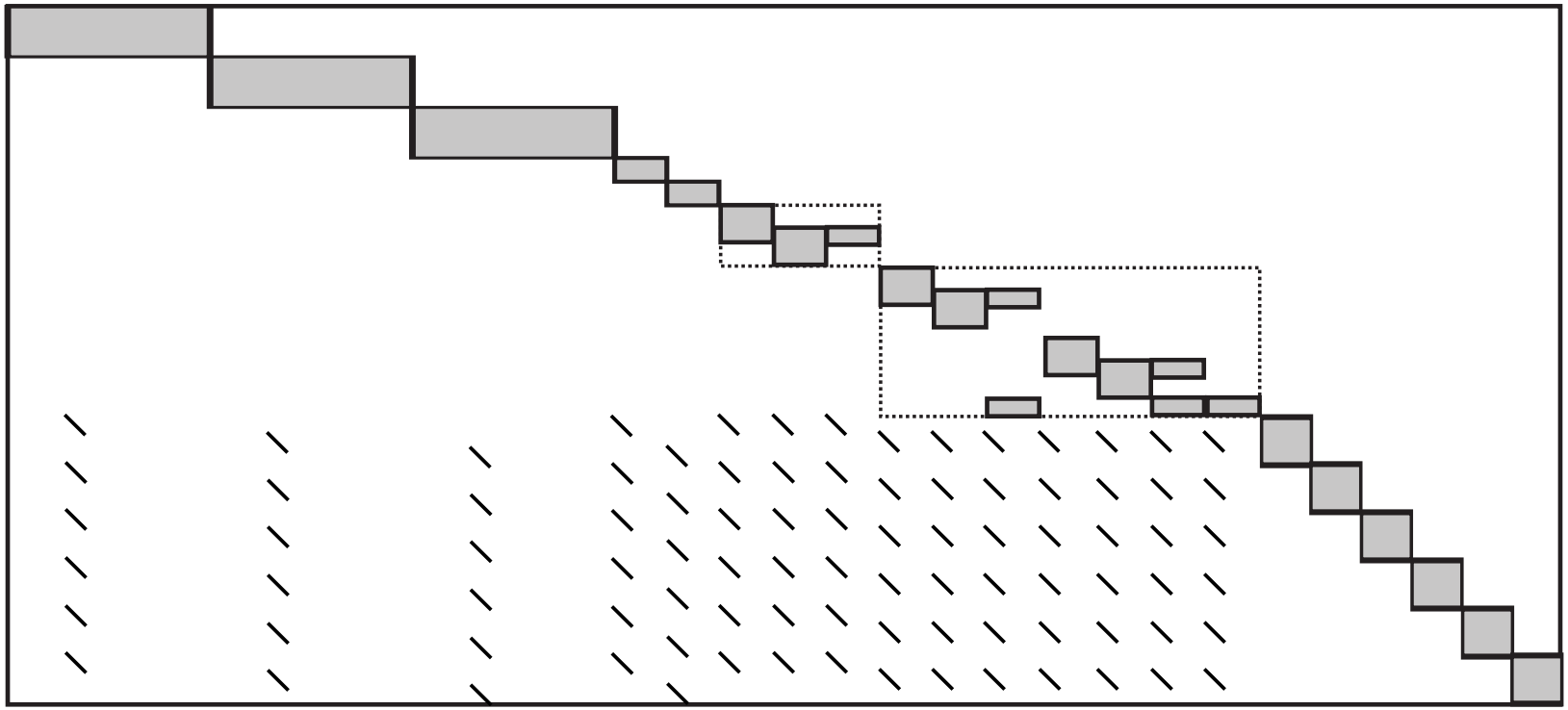
Our Masterpiece: `LagrangianDualSolver`

- Works for any `Block` with natural block-diagonal structure: no `Objective` or `Variable`, all `Constraint` linking the inner `Block`
- Using `LagBFunction` stealthily constructs the Lagrangian Dual w.r.t. linking `Constraint`, R^3B-ing or "stealing" the inner `Block`
- Solves the Lagrangian Dual with appropriate `CDASolver` (e.g., `BundleSolver`), stores dual and "convexified" solution in original `Block`
- Can attach `LagrangianDualSolver` and (say) `:MILPSolver` to same `Block`, solve in parallel!
- Weeks of work in days/hours (if `Block` of the right form already)
- Hopefully soon `BendersDecompositionSolver` (crucial component `BendersBFunction` existing and tested)
- Multilevel nested parallel heterogeneous decomposition by design (but I'll believe it when I'll see it running)

Outline

Unit Commitment

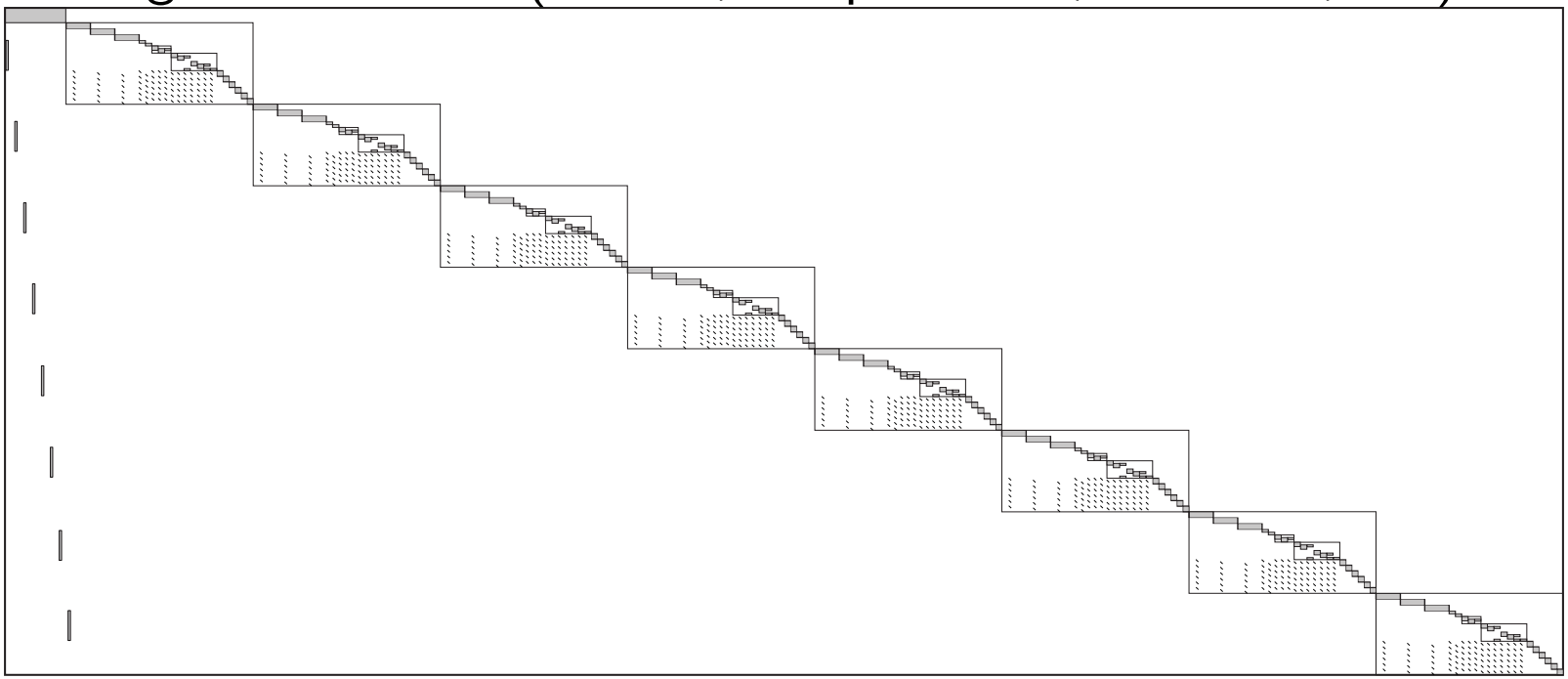
- Schedule a set of generating units to satisfy the demand at each node of the transmission network at each time instant of the horizon



- Several types of almost independent blocks + linking constraints
- Perfect structure for Lagrangian relaxation
- `UCBlock` + `ThermalUnitBlock`, `HydroUnitBlock`, ... + `DCNetworkBlock`, `EnergyCommunityNetworkBlock` ...

Seasonal Storage Valuation

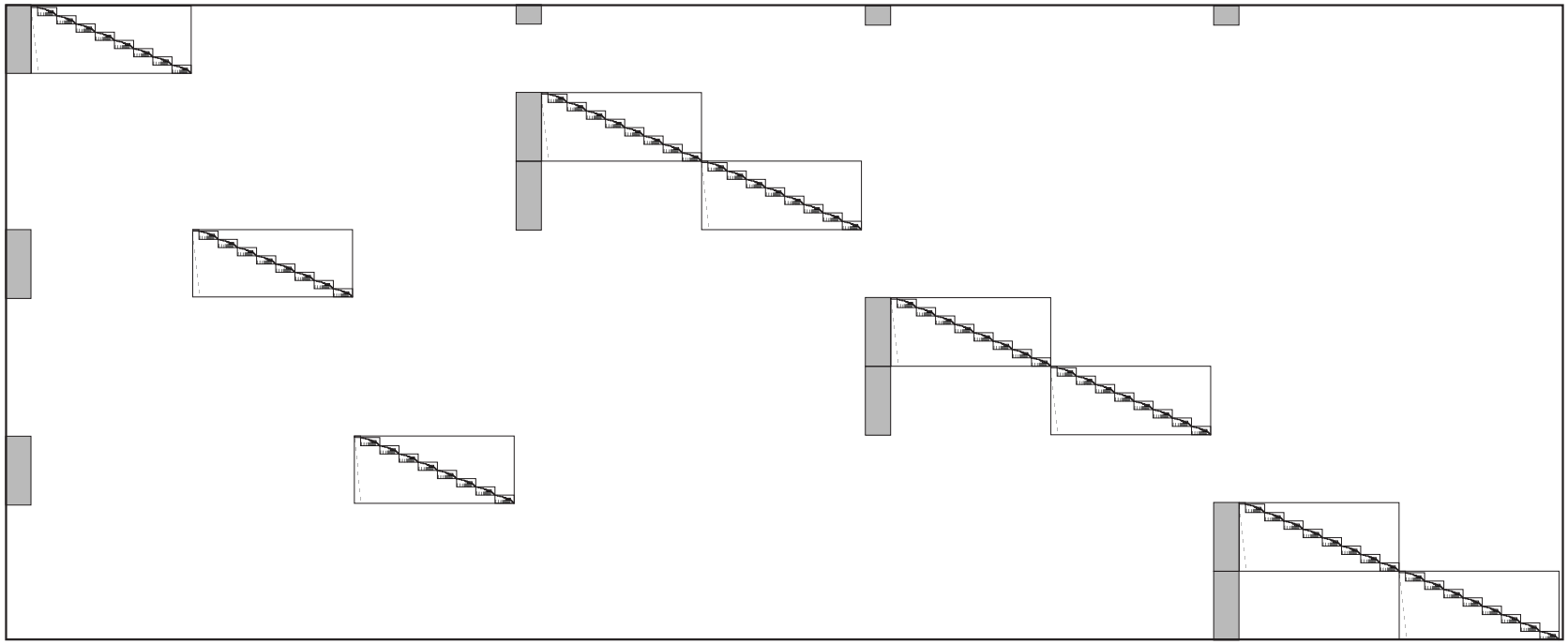
- Mid-term (1y) cost-optimal management of water levels in reservoirs considering uncertainties (inflows, temperatures, demands, . . .)



- Very large size, nested structure
- Perfect structure for Stochastic Dual Dynamic Programming
- `SDDPBlock` with as many sub-`Block` as periods, a `StochasticBlock` inside each `LagBFunction`, dynamic `PolyhedralFunction` to represent the (approximate) value-of-water function, one `UCBlock` inside each one

Investment Layer

- Long-term (30y) optimal (cost, pollution, CO_2 emissions, . . .) planning of production/transmission investments considering multi-level uncertainties scenarios (technology, economy, politics, . . .)



- Many scenarios, huge size, multiple nested structure $\Longrightarrow$ multiple nested Benders' or Lagrangian decomposition and/or SDDP
- Ad-hoc concept for "multiple copies of a `Block`", could be generalised
- Rather nasty with SMS++, at all doable without?

Outline

Unit Commitment – some results

- `ThermalUnitBlock` provides 6 different formulations, one "exact"
- `ThermalUnitDPSolver` provides efficient solution of 1-UC problems
- `LagrangianDualSolver` (using `BundleSolver`) + `ThermalUnitDPSolver` provides best trade-off between bound tightness and computational cost as size of the instance grows

	3bin		DP		pt		LR	
units	time	gap	time	gap	time	gap	time	gap
10	0.21	1.03	78.56	0.67	1.00	0.53	0.46	0.67
20	0.90	0.93	480.02	0.51	2.58	0.27	0.83	0.51
50	4.18	0.81	3836.78	0.08	9.92	0.09	1.19	0.08

- Perspective Cuts always included (bounds too much worse if not)
- Obvious trade-off between root bound and LP cost, DP impractical
- Cplex cuts effective for small n / weaker formulation, less otherwise

Sample computational results: overall B&C

	3-bin				DP				p_t			
n	time	opt	nodes	gap	time	opt	nodes	gap	time	opt	nodes	gap
10	28	5	275	0.01	832	5	599	0.01	5	5	41	0.01
20	7036	2	3561	0.08	7902	2	1961	0.05	1066	5	1234	0.01
50	10000	0	1619	0.12	10000	0	695	0.14	8095	1	2303	0.03
10	21	5	163	0.09	500	5	444	0.10	2	5	1	0.08
20	6002	2	1980	0.11	5490	4	1237	0.11	37	5	74	0.10
50	6052	2	1042	0.14	6927	3	504	0.11	160	5	148	0.08

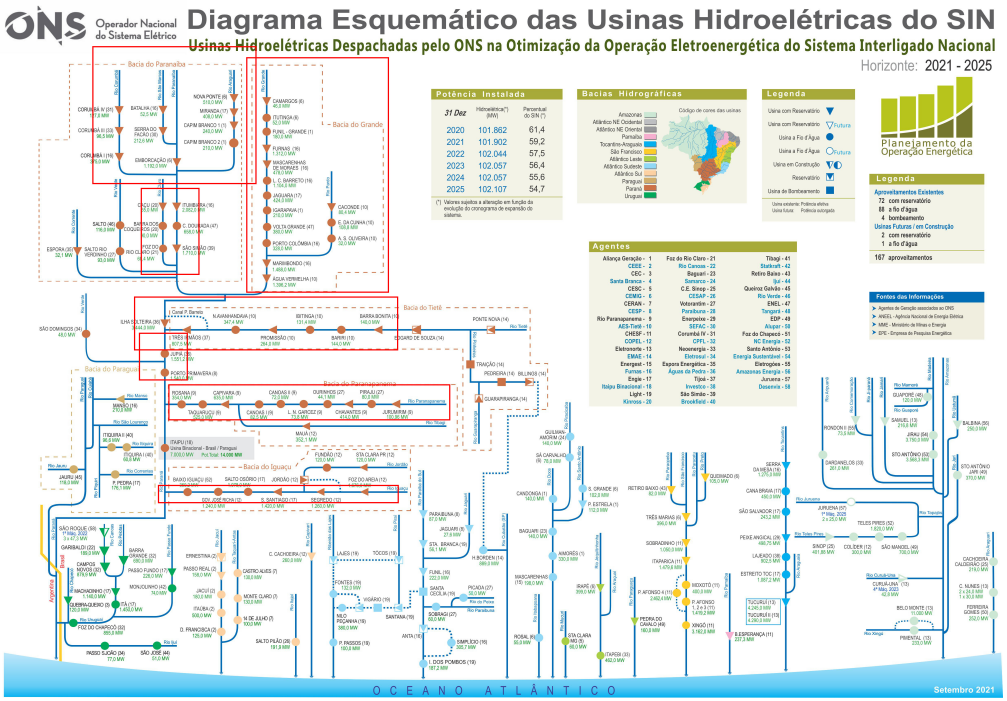
- Above stop gap 1e-4, below stop gap 1e-3 (even less in practice)
- p_t formulation promising: maybe smaller exact formulation?
- $|T| = 24$, again p_t suffers more than 3bin for larger T
- Stabilised Structured DW may make DP / p_t^h (more) competitive (but 10 years in the making and still a lot of work to do)

Seasonal Storage Valuation – some results I

- `SDDPSolver` requires convex problem: any of the above

- Brazilian hydro-heavy system:
  53 hydro (3 cascade), 98 thermal (coal, gas, nuclear), stochastic inflows (20 scenarios)

- Out-of-sample simulation:
  1000 scenarios



	Cont. relax.	Lag. relax.
Cost: Avg. / Std.	4.6023e+9 / 1.3608e+9	4.5860e+9 / 1.3556e+9

- Only 0.4% better, but just changing a few lines in the `Configuration`
  (Lagrangian about 4 times slower, but can be improved)

Seasonal Storage Valuation – some results II

- Single node (Switzerland)
- 60 stages (1+ year), 37 scenarios, 168 time instants (weekly UC)
- Units: 3 intermittent, 5 thermals, 1 hydro
- Out-of-sample simulation: all 37 scenarios to integer optimality

	Cont. relax.	Lag. relax.
Cost: Avg. / Std.	1.3165e+11 / 2.194e+10	1.2644e+11 / 2.167e+10
Time:	25m	7h30m

- Much longer, but:
  - simulation cost $\approx$ 30m per scenario, largely dominant
  - save 4% just changing a few lines in the configuration
  - LR time can be improved (`ParallelBundleSolver` not used)

Seasonal Storage Valuation – some results III

- A different single node (France)
- 60 stages (1+ year), 37 scenarios, 168 time instants (weekly UC)
- 83 thermals, 3 intermittent, 2 batteries, 1 hydro
- Out-of-sample simulation: all 37 scenarios to integer optimality

	Cont. relax.	Lag. relax.
Cost: Avg. / Std.	3.951e+11 / 1.608e+11	3.459e+11 / 8.903e+10
Time:	5h43m	7h54m

- Time not so bad (and 3h20m on average simulation per scenario) using `ParallelBundleSolver` with 5 threads per scenario
- That's 14% just changing a few lines in the configuration
- Starts happening regularly enough (and lower variance) to be believable

Investment Layer – some results I

- Simplified version: solve SDDP only once, run optimization with fixed value-of-water function + simulation (`SDDPGreedySolver`)
- EdF EU scenario: 11 nodes (France, Germany, Italy, Switzerland, Eastern Europe, Benelux, Iberia, Britain, Balkans, Baltics, Scandinavia), 20 lines
- Units: 1183 battery, 7 hydro, 518 thermal, 40 intermittent
- 78 weeks hourly (168h), 37 scenarios (demand, inflow, RES generation)
- Investments: 3 thermal units + 2 transmission lines.
- Average cost: original (operational) 6.510e+12
  optimized (investment + operational) 5.643e+12
- This is $\approx$ 1 Trillion Euro, 15%
- Running time: ??? hours for value-of-water functions (EdF provided) + 10 hours (4 scenarios in parallel + `ParallelBundleSolver` with 6 threads) for the investment problem

Investment Layer – some results II

- Simplified version (fixed value-of-water with continuous relaxation)
- Same 11 nodes, 19 lines
- Less units: 7 hydros, 44 thermals, 24 batteries, and 42 intermittent
- More investments: 82 units + 19 transmission lines.
- 78 weeks hourly (168h), 37 scenarios (demand, inflow, RES generation)
- Average cost: original (operational) 3.312e+12
  optimized (investment + operational) 1.397e+12
- This is $\approx$ 2 Trillion Euro, 137%
- Running time: 48 hours for value-of-water functions (2 nodes = 96 cores) + 5h 20m to solve the investment problem (1 nodes = 48 core)

Investment Layer – some results III

- Same simplified version as above
- EdF EU scenario: 14 nodes (France, Germany, Italy, Switzerland, Eastern Europe, Benelux, Iberia, Britain, Balkans, Baltics, Denmark, Finland, Sweden, Norway), 28 lines
- Units: 62 thermals, 54 intermittent, 8 hydros, 39 batteries
- 78 weeks hourly (168h), 37 scenarios (demand, inflow, RES generation)
- Investments: 99 units of all kinds + all transmission lines
- Average cost: original (operational) 3.465e+12
  optimized (investment + operational) 4.708e+11
  one order of magnitude saving (suspect most value of lost load)
  636% better investing on just 4 lines and 10 hydrogen power plants
  (and run stopped early on for a numerical error in `Cplex`)
- Running time: 7 hours on 48 cores, 375GB of RAM

Investment Layer – the (Little-)Big Kahuna results

- The true version: value-of-water recomputed anew for each investment
- But still simplified: only one scenario (long way to go)
- As usual, SDDP with Continuous or Lagrangian
- One node (48 core, 375Gb) not enough, must either MPI-distribute over many or run on larger nodes (48 core, 800Gb of RAM suffice)
- After ≈648h (several time-outs&resumes, maintenance breaks, . . .)
  simulation-based: investment + operational 4.708e+11
  SDDP-based: investment + operational 4.537e+11 (17 billion€ saving)
- Perhaps better idea: warm-start SDDP-based from simulation-based, got an even better 4.325e+11 to start with in 24h (avoid `Cplex`)
- warm-started SDDP-based currently running, no more results to show
- Two small-ish (≈10000h) CINECA grants to debug&test, a much bigger one needed to run the real Big Kahuna (one more decomposition level)
- But we are getting there, thanks to `SMS++`

Outline

The many things that we do not have (yet)

- A relaxation-agnostic Branch-and-X `Solver`
- Many other forms of (among many other things):
  - `Variable` (`Vector/MatrixVariable`, `FunctionVariable`, ...)
  - `Constraint` (`SOCConstraint`, `SDPConstraint`, `PDEConstraint`, `BilevelConstraint`, `EquilibriumConstraint`, ...)
  - `Objective` (`RealVectorObjective`, ...)
  - `Function` (`AlgebraicFunction`, ...)
- Better handling of many things (groups of stuff, `Modification`, ...)
- Interfaces with many other general-purpose solvers (`GuRoBi`, `OSISolverInterface`, `Couenne`, OR-tools `CP-SAT Solver`, ...)
- Many many many more `:Block` and their specialised `:Solver`
- Translation layers from "real" modelling languages (AMPL, JuMP, ...)
- In a word: users/mindshare – chicken-and-egg problem

Outline

Conclusions and (a lot of) future work

- SMS++ is there, actively developed
- Allows exploiting multiple nested heterogeneous structure, $\approx$ the only system designed for huge-scale (in particular, stochastic) problems
- Could become really useful after having attracted mindshare, self-reinforcing loop (very hard to start)

Conclusions and (a lot of) future work

- SMS++ is there, actively developed
- Allows exploiting multiple nested heterogeneous structure, $\approx$ the only system designed for huge-scale (in particular, stochastic) problems
- Could become really useful after having attracted mindshare, self-reinforcing loop (very hard to start)
- Hefty, very likely rather unrealistic, sough-after impacts:
  - improve collaboration and code reuse, reduce huge code waste
  - significantly increase the addressable market of decomposition
  - a much-needed step towards higher uptake of parallel methods
  - the missing marketplace for specialised solution methods
  - a step towards a reformulation-aware modelling system[4]

Conclusions and (a lot of) future work

- SMS++ is there, actively developed
- Allows exploiting multiple nested heterogeneous structure, $\approx$ the only system designed for huge-scale (in particular, stochastic) problems
- Could become really useful after having attracted mindshare, self-reinforcing loop (very hard to start)
- Hefty, very likely rather unrealistic, sough-after impacts:
  - improve collaboration and code reuse, reduce huge code waste
  - significantly increase the addressable market of decomposition
  - a much-needed step towards higher uptake of parallel methods
  - the missing marketplace for specialised solution methods
  - a step towards a reformulation-aware modelling system[4]
- As much a community-building effort as an actual software project
- Lots of fun to be had, all contributions welcome

---

[4] F., Perez Sanchez "Transforming Mathematical Models Using Declarative Reformulation Rules" *LNCS*, 2011