**The 7th International Conference "Distributed Computing and Grid-technologies in Science and Education" (GRID'2016)**

# High-level software for finite-dimensional and dynamic optimization in distributed computing infrastructure

**Alexander P. Afanasiev, Vladimir V. Voloshinov**

***Center of Distributed Computing,***
***Institute for Information Transmission Problems RAS, Moscow***

*Supported by the Russian Foundation for Basic Research (grant # 16-11-10352)*

*LIT JINR, Dubna, 2016*

# Content of the report

**Optimization modeling (OM) on the base of Dynamical Optimization & Mathematical Programming**

**Software for OM considered:**

- **solvers (LP/MILP, NLP/MINLP ...);**
- **algebraic modeling languages translators (AMPL, GAMS, Mosel-Xpress …).**

**Review of existing technologies of OM in distributed computing environment**

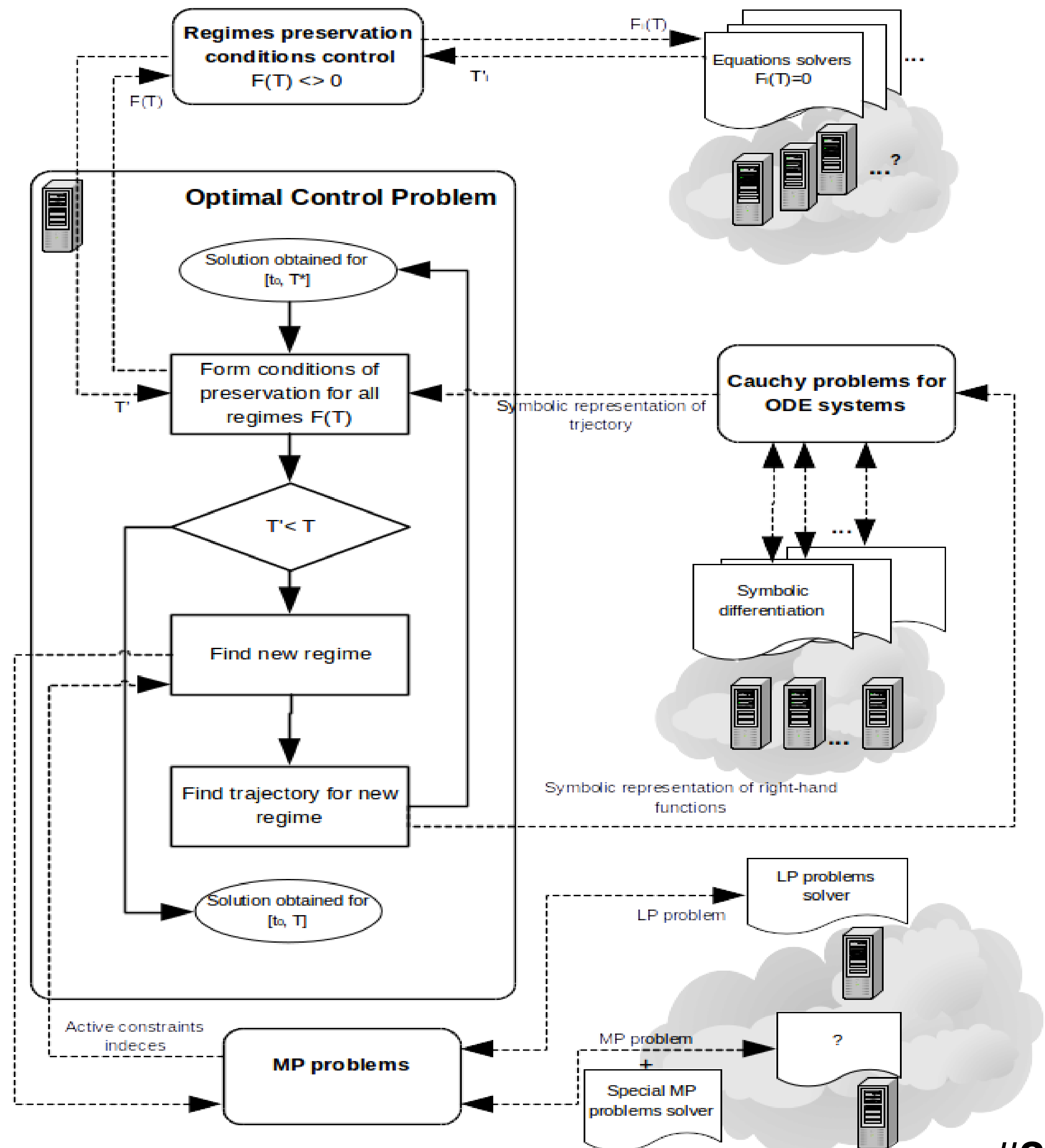
**Principles of our approach (cloud platform Everest & AMPLX)**

**Examples of AMPLX demos & applications, including**

- **branch-and-bound algorithm of nanomaterial structure identification with a joint X-Ray and neutron diffraction;**
- **Coarse-grained algorithms for MILP (coarse grained B&B, local elimination algorithms for MILP with quiasi-block constraints structure**

# We began with optimal control problem (OCP)

**Historically, our research on the subject has been inspired by an optimal trajectories continuation method (Alexandr Afanasiev) which suites for a distributed computing environment**



# OCP with linear constraints

$$\int_0^T \langle g(x(t)), u(t)\rangle dt \to \min$$

$$\dot{x}(t) = u(t), \quad x(t_0) = x_0,$$

$$K(x(t)) \cdot u(t) = L(x(t)),$$

$$M(x(t)) \cdot u(t) \geq N(x(t)).$$

$$x(t) = (x_1(t), \ldots, x_n(t)) \qquad u = (u_1(t), \ldots, u_n(t))$$

$K(x(t))$ $-\ k \times n$ *matrix* $\qquad$ $M(x(t))$ $-\ m \times n$ *matrix*

$L(x(t))$ $-$ *matrix* $k \times 1$ $\qquad$ $N(x(t))$ $-$ *matrix* $m \times 1$

# OCP with linear constraints => Linear Programming

**Find locally optimal control, i.e. for the beginning of the trajectory**

$$\langle g(x_0), u \rangle \to \min$$

$$K(x_0) \cdot u = L(x_0),$$

$$M(x_0) \cdot u \geq N(x_0).$$

$$x_0 = (x_{01}, \ldots, x_{0n})$$

$$K(x_0) \ - k \times n \ matrix$$

$$L(x_0) \ - matrix \ k \times 1$$

$$u = (u_1, \ldots, u_n)$$

$$M(x_0) \ - m \times n \ matrix$$

$$N(x_0) \ - matrix \ m \times 1$$

# Local OCP with linear constraints (regime)

$$\left\langle g(x_0), u^* \right\rangle = \min$$

$$\int_0^T \langle g(x(t)), u(t)\rangle dt = \min$$

$$K(x_0)\cdot u^* = L(x_0),$$
$$M_A(x_0)\cdot u^* = N_A(x_0).$$
$$M_P(x_0)\cdot u^* > N_P(x_0).$$

$$\dot{x}(t) = u(t), \quad x(t_0) = x_0,$$
$$K(x(t))\cdot u(t) = L(x(t)),$$
$$M_A(x(t))\cdot u(t) = N_A(x(t)).$$
$$M_P(x(t))\cdot u(t) > N_P(x(t)).$$

***Nonlinear inequalities***

There exist [0,T], that if $t \in [0,T]$

$$u(t) = \dot{x}(t) = \begin{pmatrix} K(x(t)) \\ M_A(x(t)) \end{pmatrix}^{-1} \begin{pmatrix} L(x(t)) \\ N_A x(t)) \end{pmatrix}, \quad x(0) = x_0, \quad \textit{Caucshy problem}$$

# Continuation of the optimal trajectories in linear OCP

Let $x^*(t)$ is the optimal trajectory of the problem

$$\int_0^T \langle g(x(t)), u(t) \rangle dt \to \min$$

$$\dot{x}(t) = u(t), \quad x(t_0) = x_0,$$

$$K(x(t)) \cdot u(t) = L(x(t)),$$

$$M(x(t)) \cdot u(t) \geq N(x(t)).$$

Continuation of the optimal trajectories to $[0, T+\Delta]$ is connected with the problem

$$\int_t^{t+\Delta} \langle g(x(\tau)) + \mu(\tau), u(\tau) \rangle d\tau \to \min, \quad t \in [0, T],$$

$$\dot{x}(\tau) = u(\tau), \quad x(t) = x^*(t),$$

$$K(x(\tau)) \cdot u(\tau) = L(x(\tau)),$$

$$M(x(\tau)) \cdot u(\tau) \geq N(x(\tau)).$$

# Decomposition is invertible

**Decomposition of Computational Problems into subproblems which may be solved by EXISTENT s/w tool**

**Typical for Mathematics, Physics, Chemistry, Biology...**

**Optimization (LP, NLP, ...)**

**Differential Equations**

Mathematical Statistics

Data Analysis

etc.

# Mathematical Programming Problems (MP)

$$f_o(p, x) \to \min_x,$$ ***object (goal) function***

$$f_i(p, x) \leqslant 0 \;\; (i \in I)$$ ***inequalities constraints***

$$g_j(p, x) = 0 \;\; (j \in J)$$ ***equalities constr., (I, J ) - indices sets (multi-index-, symbolic ...)***

$$x \in M(p), \; x = (x^C, x^Z)$$

$$x^C \in R^{N_C}, \; x^Z \in Z^{N_Z}$$

$$p \in P$$

***M* – additional *“simple constraints”* (≤0≥, *interval, integer/boolean*)**

**- parameters' setting & «consistence checking»**

**State-of-the-art s/w support MP of various types (constraint functions' types; presence of binary/integer variables):**

- **LP/MILP – linear programming (mixed-integer LP);**
- **QP/MIQP – quadratic goal, linear constraints (MI*);**
- **QCQP/MI* – QP + quadratic constraints;**
- **NLP/MINLP – general non-linear (differentiable) functions;**
- **convex NLP/MILP – convex on all variables (including integer ones);**

**...**

# MP solvers must support ...

$$f_o(p,x) \to \min_x,$$ *object (goal) function*

$$\alpha_i | f_i(p,x) \leqslant 0 \quad (i \in I)$$ *inequalities constraints*

$$\beta_j | g_j(p,x) = 0 \quad (j \in J)$$ *equalities constr., (I, J ) - indices sets (multi-index-, symbolic ...)*

$$x \in M(p), \; x = \left(x^C, x^Z\right)$$

$$x^C \in R^{N_C}, \; x^Z \in Z^{N_Z}$$

*M* – additional *“simple constraints”* (≤0≥, *interval, integer/boolean*)

$$p \in P$$ - parameters' setting & «consistence checking»

**In theory & numerical methods often use:**

**- Lagrange approach (functions & multipliers/dual variables)**

$$L(\alpha,\beta,x) = f_o(x) + \sum_{i \in I} \alpha_i \cdot f_i(x) + \sum_{j \in J} \beta_j \cdot g_j(x)$$

**- 1st, 2nd derivatives for NLP:**

$$\nabla_x f_o(p,x), \nabla_x f_i(p,x) \; (i \in I), \nabla_x g_j(p,x) \; (j \in J)$$

$$\nabla_{xx} f_o(p,x), \nabla_{xx} f_i(p,x) \; (i \in I), \nabla_{xx} g_j(p,x) \; (j \in J)$$

# Solvers for optimization problems

**Non-exhaustive list of solvers we tried/use in our researches**

**COmputational INfrastructure for Operations Research, www.coin-or.org ("IBM's aegis"), more than 40 solvers&libs: since ~2005**

**CBC/CLP (LP, MILP),**

**Ipopt (NLP),**

**Bonmin(CBC/Ipopt) (MINLP, convex on cont. & int. vars)**

**Zuse Institute Berlin, Germany,**

**SCIP (LP, MILP, MIQP, ), ver. 1.0 2007, ver. 3.2.1 the last**

**GLPK (LP, MILP), A. Makhorin, Mosc. Avia. Institute, ~2002**

**LP_SOLVE, (LP, MILP) Eindhoven University of Technology, Netherlands, since ~2000**

**Commercial: KNITRO, SNOPT, Gurobi, CPLEX**

**Commersial *XPRESS** Fico Optimization (deserves special attention)**

# The concept of Algebraic Modeling Languages

**AML - Algebraic Model Languages (AMPL, GAMS, Zimpl, etc).**

**Common features:**

- **Convenient (symbolic "TeX-like") description of object & constraints functions**
- **Separation of "symbolic/abstract" models and numerical data for multivariate computation (parameter sweeping)**
- **Automatic differentiation (Jacobian & Hessian)**
- **Support of "Lagrange formalism" - access to optimal variables and duals found by solver**
- **Unified open-source (even for "commercial" AMLs) API for solvers' and applications' developers**

**Usage of AMLs is crucial at preliminary phases of R&D**

# There are a number of AMLs

**Non-exhaustive list:**

**AMPL - A Modeling Language for Mathematical Programming,**
**AT&T Bell Laboratories, D.M. Gay, Brian W. Kernighan,**
**since 1980-x (1985), http://www.ampl.com**

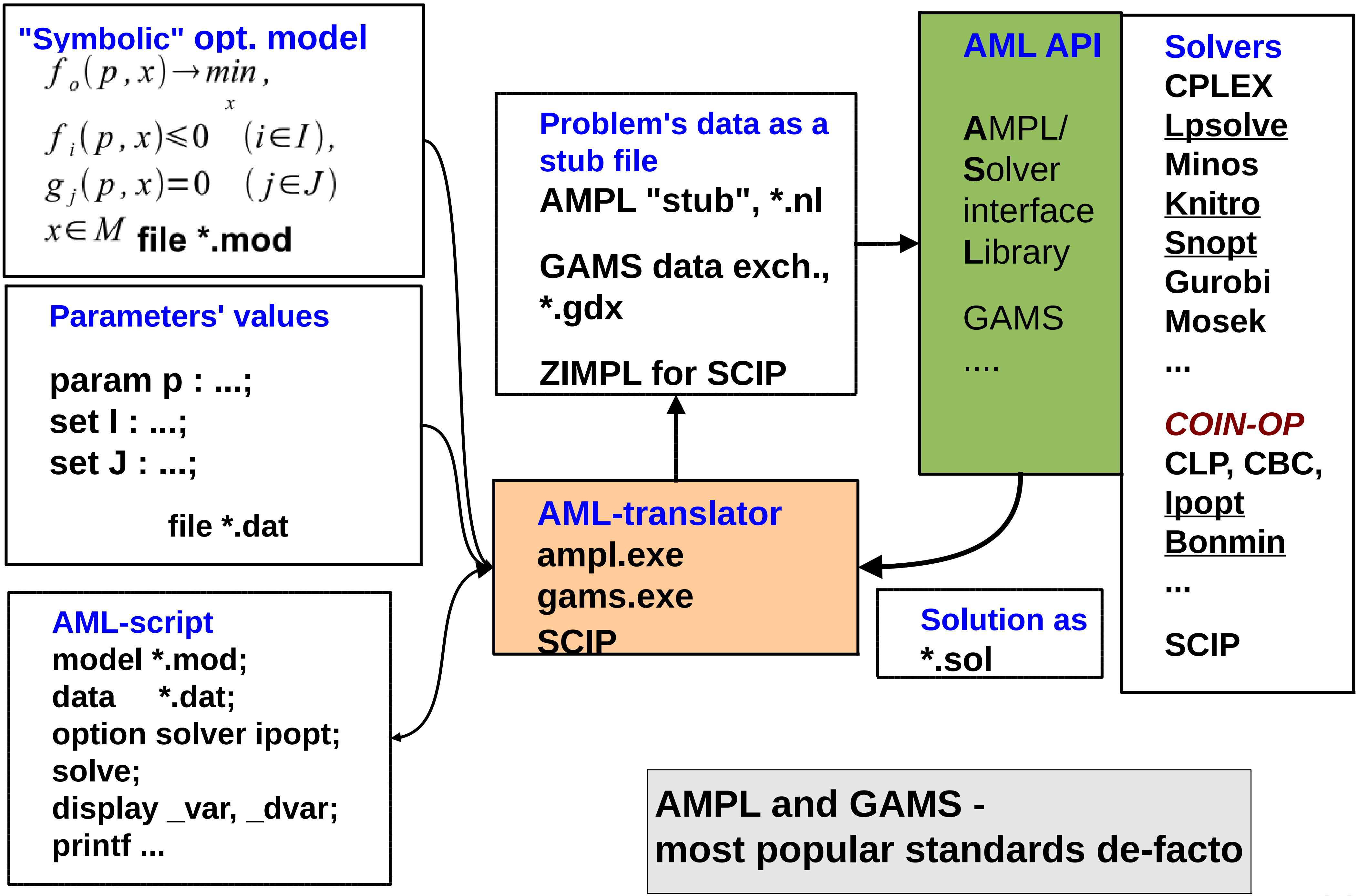
**GAMS - General Algebraic Modeling System,**
**International Bank for Reconstruction and Development,**
**since 1976, http://www.gams.com**

**XPRESS-MOSEL – c 2001, c 2010 FICO Xpress Optimization Suite,**
**http://fico.com**
**Zimpl - since 2004, http://zimpl.zib.de/ (LP, MILP, NLP ?)**
**Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB)**

OPL - Optimization Programing Lang., IBM,
ILOG CPLEX (LP, QP, ...), CP Optimizer, http://www-01.ibm.com/

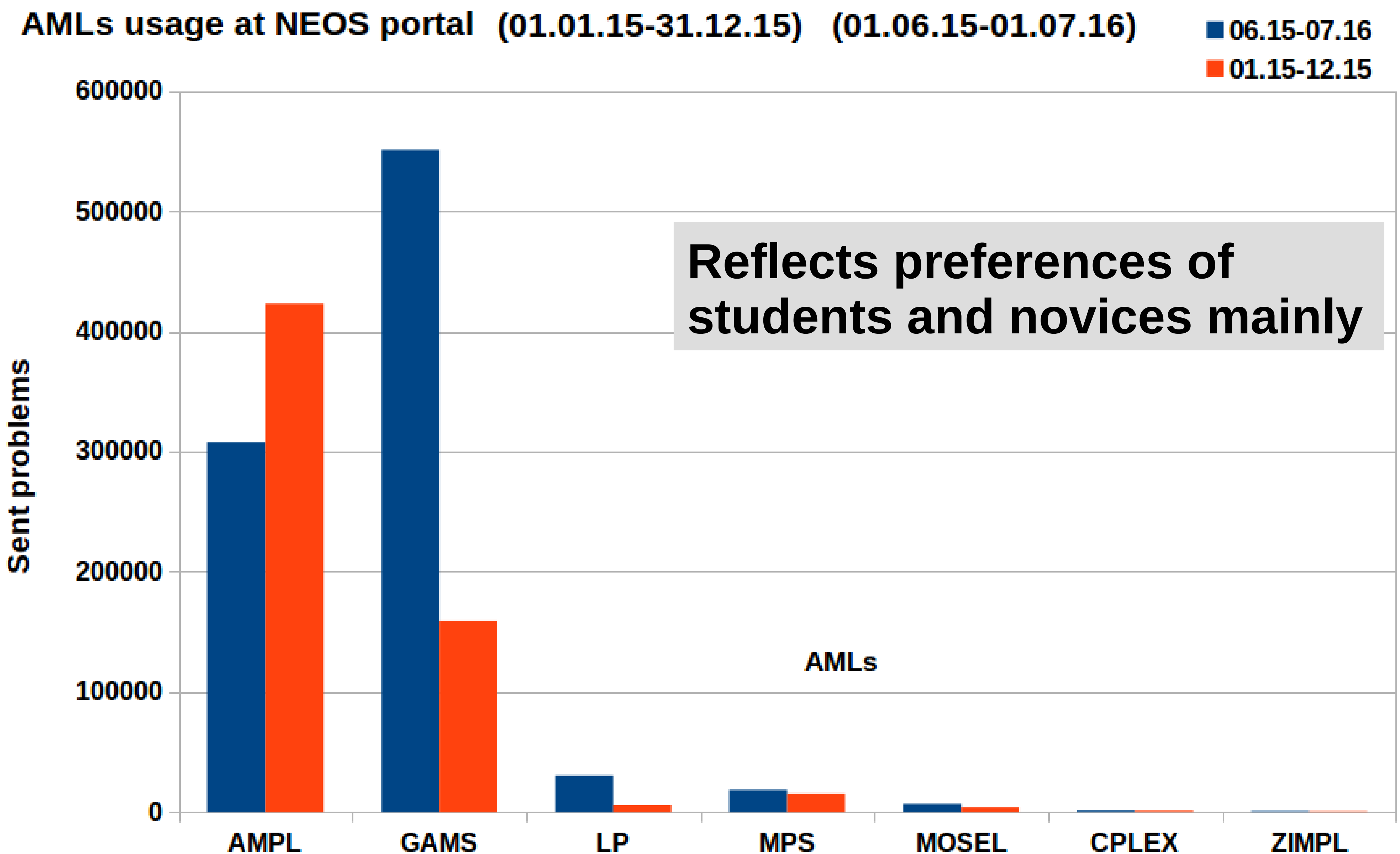
GNU MathProg - "subset" of AMPL for GLPK, GNU LP Kit,
Andrey Makhorin, MAI, since 2000, http://www.gnu.org/software/glpk/

# General scheme of AMLs usage



# Rating of AMLs at NEOS portal (GAMS vs AMPL)

**https://neos-server.org/neos/report.html**



# Optimization & distributed computing

**Typical problems:**

- efficient usage of state-of-the-art and/or emerging solvers on available, heterogeneous, computing infrastructure
- keep “traditional” R&D practice, especially, at the beginning phases of researches

glpk, lpsolve,
clp, cbc, bonmin,
ipopt, scip, etc.

Solvers

???

???


Researchers
and their R&D
specific requirements

Computing resources

AMPL,
GAMS,
Zimpl, etc

???

???

MPI,
Object oriented m/w
Web-services,
REST-services, etc

Modeling lang. & interpreters

Middleware

# Technologies & practice (NEOS & Kestrel client)

**NEOS-Server: «acquaintance» portal with state-of-the-art solvers & AMLs, http://www.neos-server.org/neos/**
**Dozens of solvers (~40), compatible with AMPL, GAMS, ZIMPL, XPRESS-Mosel …**

**Simple Web-forms to submit computing jobs**
**Client aplications for remote access to NEOS:**
**- Submission Tool (Python + Java GUI): instead of Web-forms**
**- Kestrel NEOS-client for AMPL- & GAMS-translators (XML-RPC):**

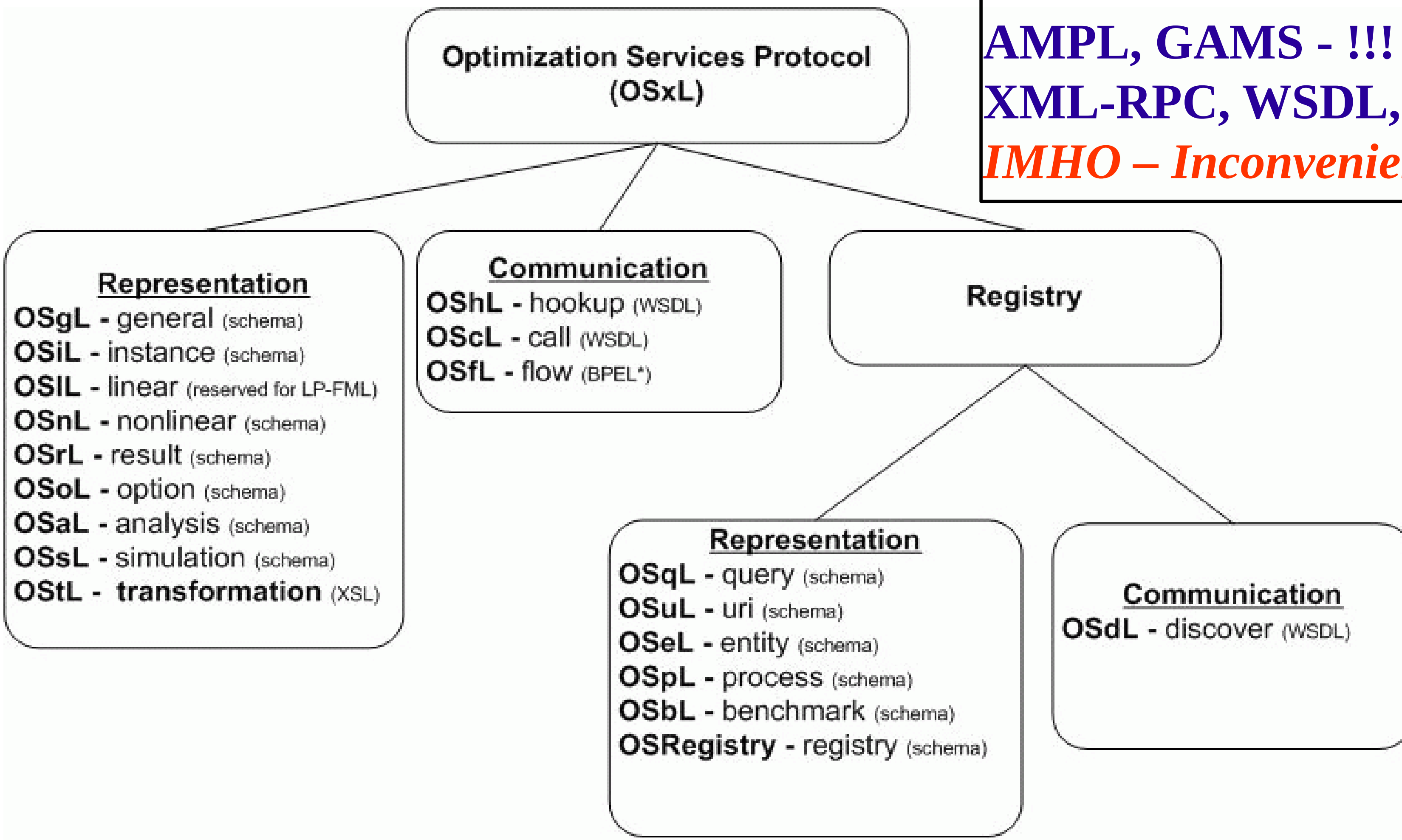
```
option solver kestrel;
option kestrel_options 'solver=<solverName>';
option neos_server 'www.neos-server.org:3332';
...
solve # Synchronous/blocking remote call of NEOS-solvers
```

**Suites for demonstrations, but problematic for reserch & «industry»**

# Technologies & practice (COIN-OS, Web-services)

**Since 2004, project Optimization Services, www.optimizationservices.org, under the aegis of COIN-OR (IBM) www.COIN-OR.org/projects/OS.xml**

COIN solvers !!!
AMPL, GAMS - !!!
XML-RPC, WSDL, BPEL - ???
*IMHO – Inconvenient & cumbersome!*



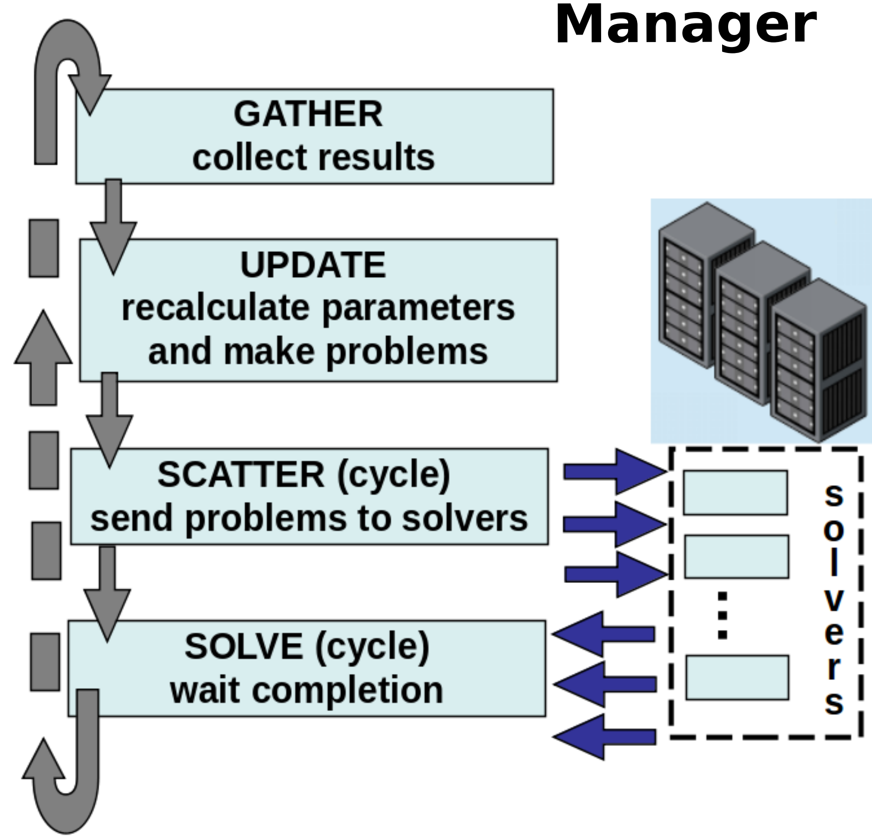

*OSmL: a modeling language and NOT an Optimization Services Protocol
*BPEL: Business Process Execution Language for flow orchestration.

# Technologies & practice (GAMS Grid extension & GUSS)

**In 2006 GAMS team proposes GAMS Grid Extension for.**
**In 2012 they introduces notion of GUSS: Gather-Update-Solve/Scatter for <u>typical template of computing scenarios with optimization models suiting for parallelization</u>**

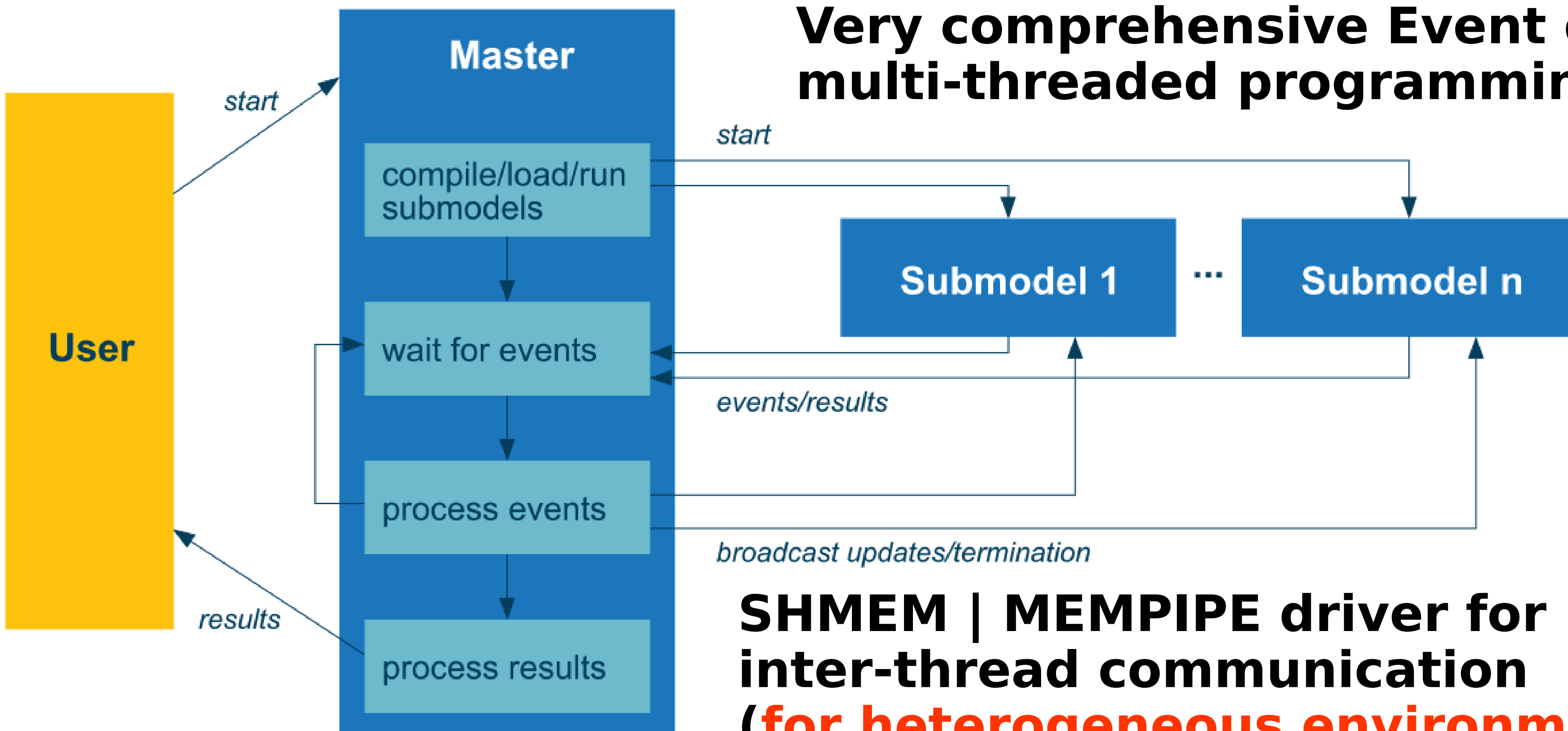
**- Dantzig-Wolfe, Benders ...**
**decomposition for block-constraints**
**- Parameter Sweeping**
**- low-dimension (1-3d) – global optimization**
**- MILP with quasi-block constraints**
**...**

**SunGrid Engine**
**Condor Res. Manager**




**Data exchange «units» :**
**=>problems as stub-files;**
**<=solutions in files**

**Mosel AML & programming language (commercial since 2001).**
**Supported by Fico, http://fico.org**
**Mosel programs are compiled into binary code for Mosel Virtual Machine including very fast XPRESS solver!**
**Since 2010 г. – Fico Optimization Modeler Suite supports distributed computing in Fico Cloud**




**Very comprehensive Event driven multi-threaded programming**

**SHMEM | MEMPIPE driver for inter-thread communication (for heterogeneous environment)**

# Our approach. Requirements

**Everest: Cloud Platform to deploy/develop REST-services, http://everest.distcom.org, REST - as an architectural style**

**HTTP, JSON (JavaScript Object Notation): transport protocol & message format (plain text),**

**Web-User-Interface (WUI) by HTML+JavaScript**

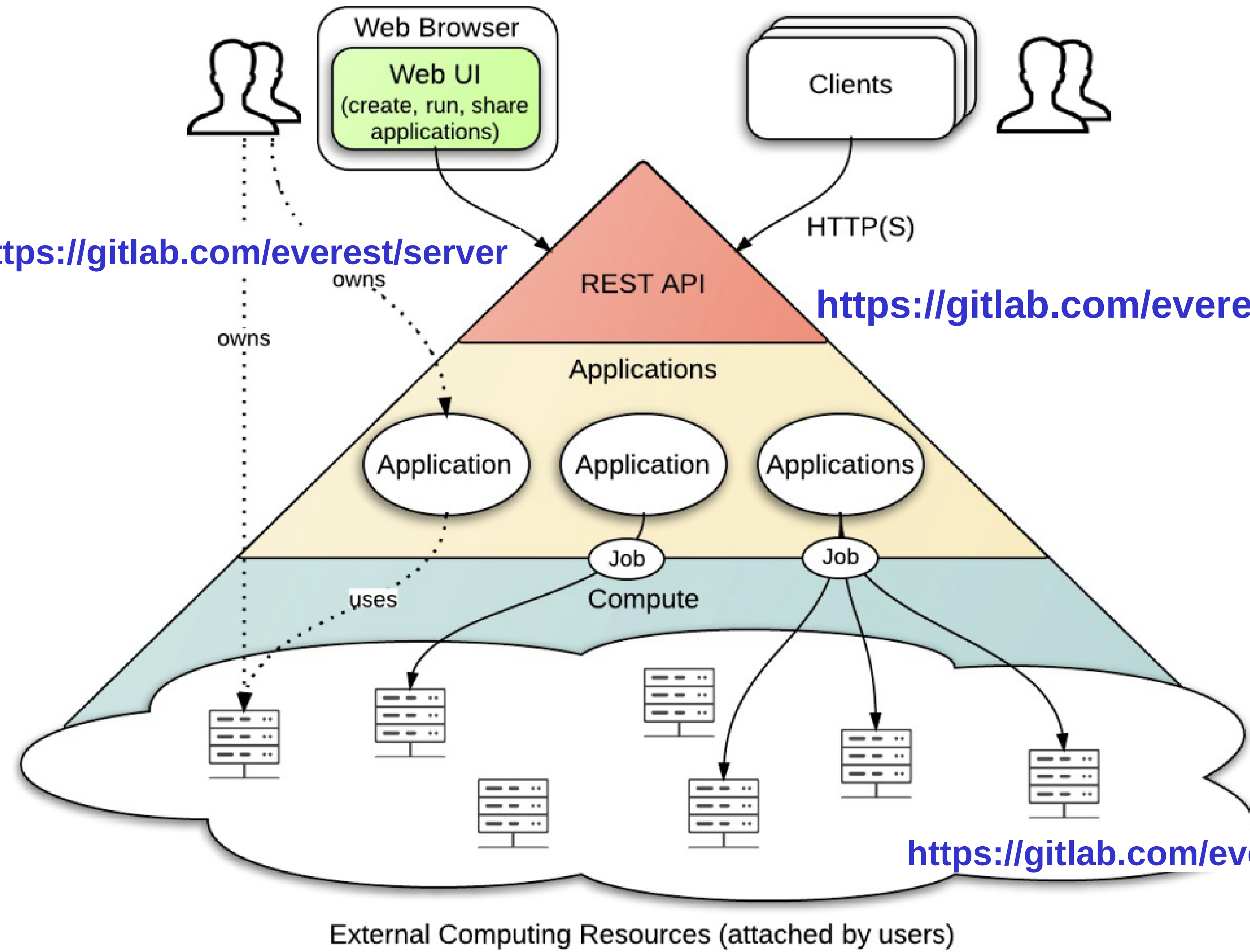
**AMPL - description of optimization modeling & computing scenarios including "coarse-grained" decomposition algorithms (high-level)**

**AMPL-compatible solvers CLP, CBC, Ipopt, Bonmin, SCIP (LP/MILP, NLP, MINLP), BnB (MILP, global opt)**

**Everest Python API & Everest Task Protocol – for low-level data exchange (solver⇔solver, ampl⇔solver)**

# Everest platform architecture outlines

**Describe/Develop/Deploy REST-services representing existing applications**




https://gitlab.com/everest/server

https://gitlab.com/everest/python-api

https://gitlab.com/everest/agent

# AMPLX = AMPL + REST-services of optimization

**We propose technology to run any AMPL-script by standard AMPL-translator in such a way that:**

- **all MP problems (as well as dynamically composed during running of the script) will be solved by remote solvers;**
- **sets of independent sub-problems will be solved in parallel by a pool of the computing resources, whose computing power might be changed transparently for users.**

**Simple methodology/recommendation (verbal) to modify any AMPL-program for AMPLX: replace AMPL-operators *solve, repeat {...}*, *for {...}* with AMPLX “templates”.**

**Implementation: Python + Everest Python API + AMPL-«macroses» [https://gitlab.com/ssmir/amplx](https://gitlab.com/ssmir/amplx)**

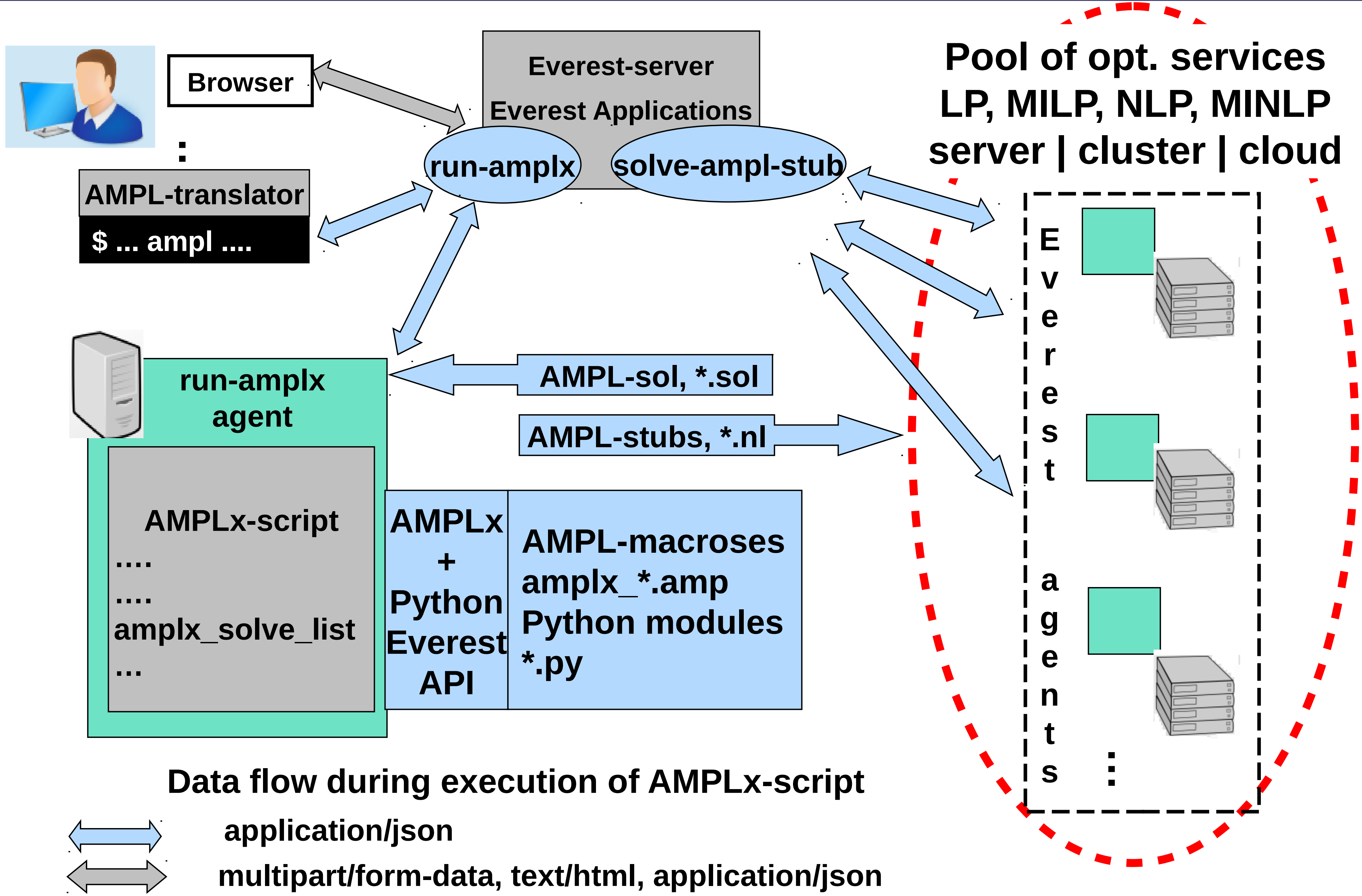
# AMPLX templates for modification of an AMPL-script

```
option solver ipopt # select AMPL-solver
option ipopt_options “acceptable_tol=10e-8 …” # solver options
... # AMPL operators
for {p in SetP} {
... # AMPL operators, preparing SubProb[p]
  solve SubProb[p];
... # operators, using solution of SubProb[p]
}
... # more scripts
```

**Replace AMPL *for* {…} or loop {...} with AMPLx “template”**

```
commands amplx_globals.amp;                    # Initialize AMPLx once (!)
let __amplx_solver := “ipopt";                 # set AMPL-solver, «connected» toAMPLx
let __amplx_solver_opts_file := "ipopt.opt";   # solver options file
... # AMPL operators, intact (!)
let __amplx_probSet := {};                     # free list of sub-problems AMPLx
for {p in SetP} { # UPDATE problems sycle
 ... # AMPL operators, preparing  SubProb[p]
 problem SubProb[p];                           # switch in the context of SubProb[p]
 let __amplx_probName := ("SubProb_" & p);     # set problem’s name for AMPLx,
 commands amplx_write_problem_stub.amp;        # write stub & update list of sub-probs
}
commands amplx_solve_problems_list.amp;        # parallel solving of all problems in the list
for {p in SetP} { # GATHER results cycle
... # AMPL  operators, preparing SubProb[p] , intact (!)
 problem SubProb[p];                           # switch to SubProb[p] context
 solution ("SubProb_" & p & ".sol");           # load results from *.sol file DELIVERED by AMPLx
... # ooperators, using solution of SubProb[p]
}
... #
```

# AMPLX architecture



# Transport problems (classical block structure)

**Set of commodities should be supplied from a number of storages to to the consumers over transport network with limited bandwidth.**

***Sets:* *O* – warehouses, *D* — deliver point, *P* — commodities**

**S*upply*$_{o,p}$ - volume of *p* in storage *o*, D*emand*$_{d,p}$ - consumption of *p* in *d***

**$c_{o,d,p}$ - *transport cost* of p unit over arc (o->d)**

**$l_{o,d}$ - bandwidth of arc (o->d)**

$$\sum_{o \in O, d \in D, \in P} c_{o,d,p} \cdot x_{o,d,p} \xrightarrow[x_{o,d,p}]{} \min,$$

$$\sum_{d \in D} x_{o,d,p} \leqslant \mathsf{S}_{o,p} \, (o \in O, p \in P), \quad \sum_{o \in O} x_{o,d,p} \geqslant \mathsf{D}_{d,p} \, (d \in D, p \in P),$$

$$\sum_{p \in P} x_{o,d,p} \leqslant l_{o,d} \, (o \in O, d \in D), \quad x_{o,d,p} \geqslant 0 \;\; (o \in O, d \in D, \in P)$$

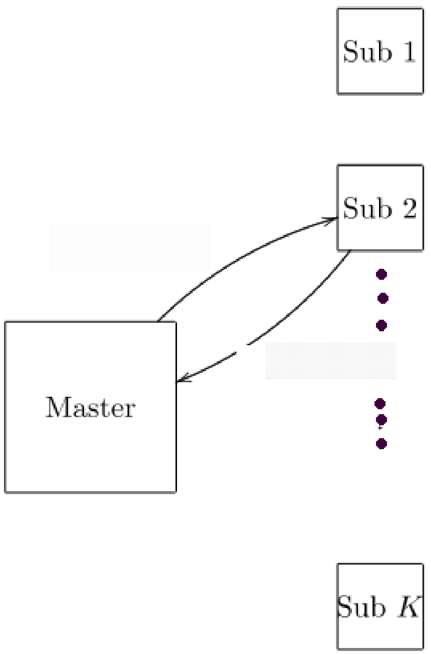
**Wellknown class LP with block structure for decompose algorithms (Dantzig-Wolfe, Benders) and their demo-implementations in GAMS, MOSEL, ... AMPL**

**www.ampl.com/NEW/LOOP2/multi2.mod, multi2.run, multi.dat**

# Demo AMPL Dantzig-Wolfe (multi2.run) is not parallel

$$\min c^T x \quad Ax = b \quad x \geq 0$$

$$Ax = \begin{pmatrix} B_0 & B_1 & B_2 & \dots & B_K \\ & A_1 & & & \\ & & A_2 & & \\ & & & \ddots & \\ & & & & A_K \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_K \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_K \end{pmatrix}$$




**«Original» AMPL demo script**
**http://www.ampl.com/NEW/LOOP2/ multi2.run**
**uses cycle [*for*] to solve sub-problems in turn**

```
...
for {p in PROD} { printf "\nPRODUCT %s\n\n", p;
   solve SubII[p];
   ...
   if Reduced_Cost[p] < - 0.00001 then  {
   /* change subproblems parameters */;
   ...
   };
...
```

# AMPLX-script (multi2_amplx_[cbc|scip|ipopt].amp)

**Replace *for {...}* for three groups (GUSS in terms of AMPLx)**

**Note that dual vars at optimal solution are required**

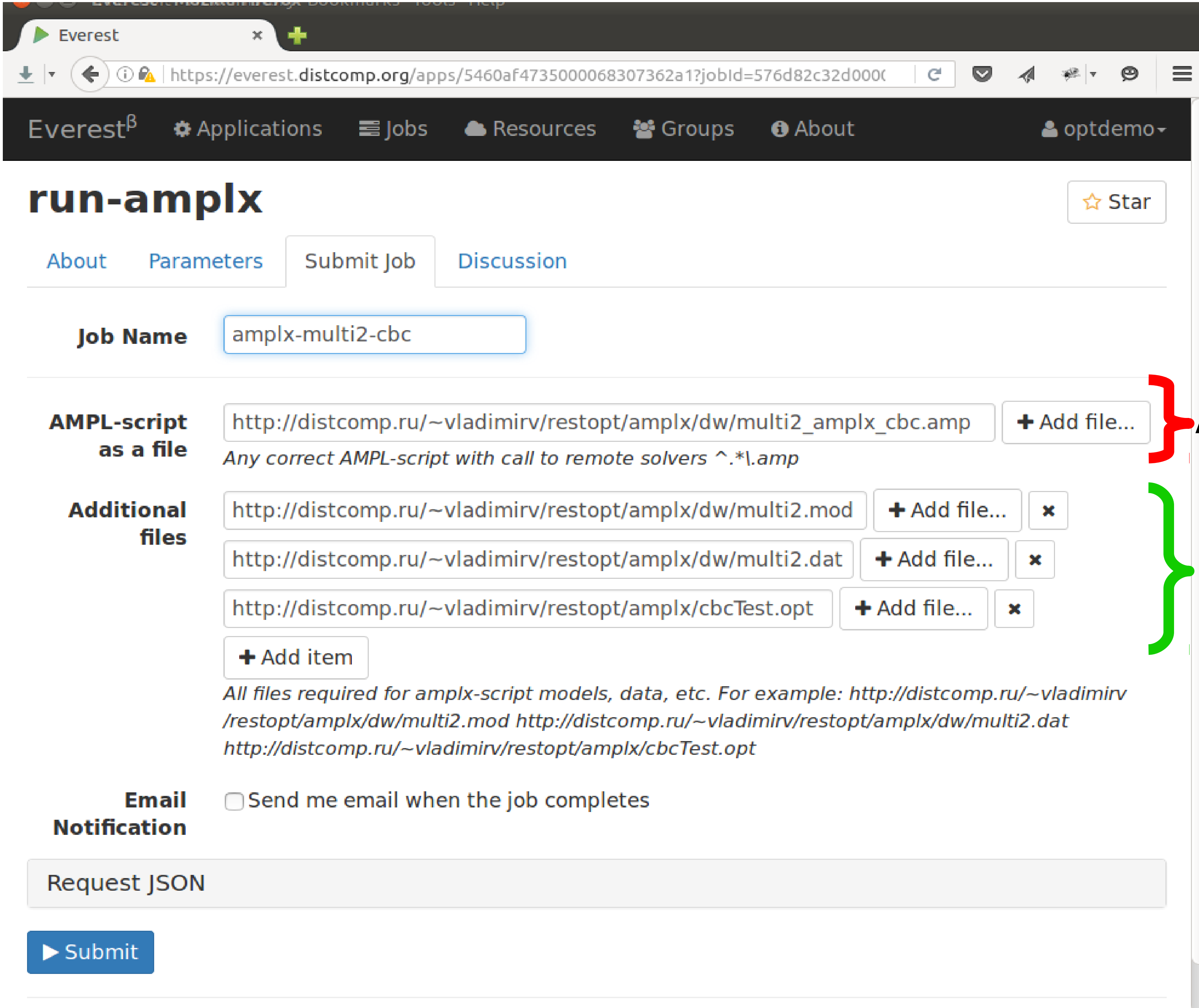
```
for {p in PROD} { printf "\nPRODUCT %s\n\n", p;
   solve SubII[p];
   ...
   if Reduced_Cost[p] < - 0.00001 then  {
   /* change subproblems parameters */;
   ...
   };
```

```
for {p in PROD} { printf "\nPRODUCT %s ==> stub \n\n", p;
   problem SubII[p];
   let __amplx_probName := ("SubII_" & p);
   commands amplx_write_problem_stub.amp; # Generates sub-problems AMPL-stubs
}

commands amplx_solve_problems_list.amp; # Parallel solving of SubII_*

for {p in PROD} { printf "\nPRODUCT %s <== solution\n\n", p;
      # solve SubII[p]
      problem SubII[p];
      solution ("SubII_" & p & ".sol");
if Reduced_Cost[p] < - 0.00001 then  {
   /* change subproblems parameters */;
   ...
   };
```

# multi2_amplx_[cbc|scip|ipopt].amp start Web-form



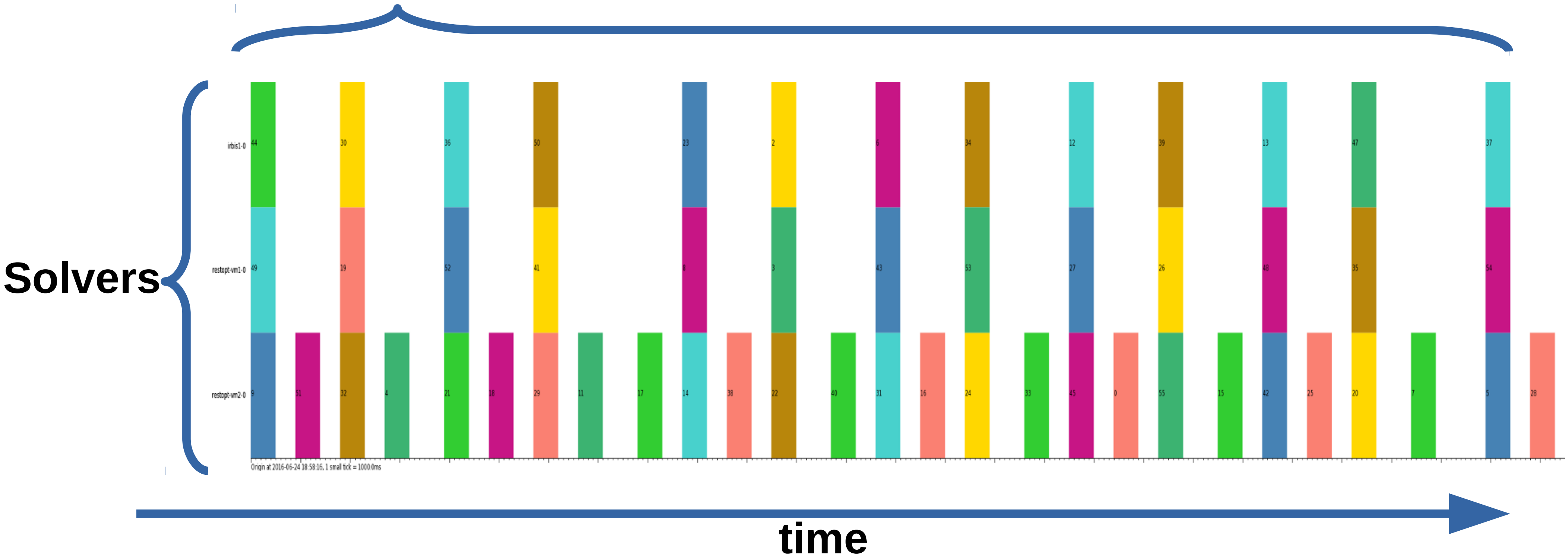

AMPLx script

Kept intact

# multi2_amplx_*.amp execution timespan/solvers log

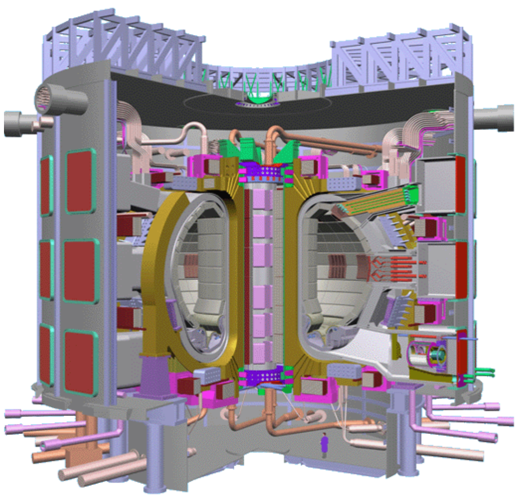
**We developed some profiling tools to analyze logs of distributed execution of AMPLX algorithms in solvers pool presented in our Everest infrastructure**

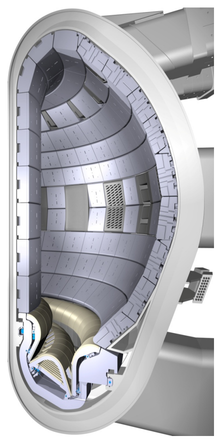
**56 subproblems (both Master* and Sub-problems) ~ 300 sec**

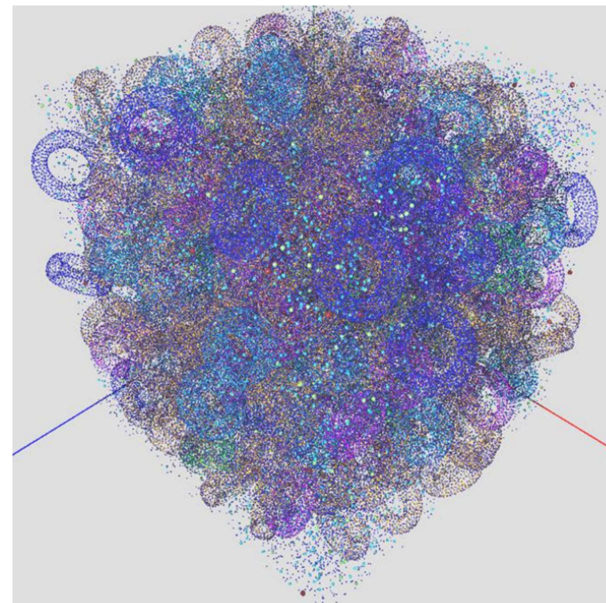


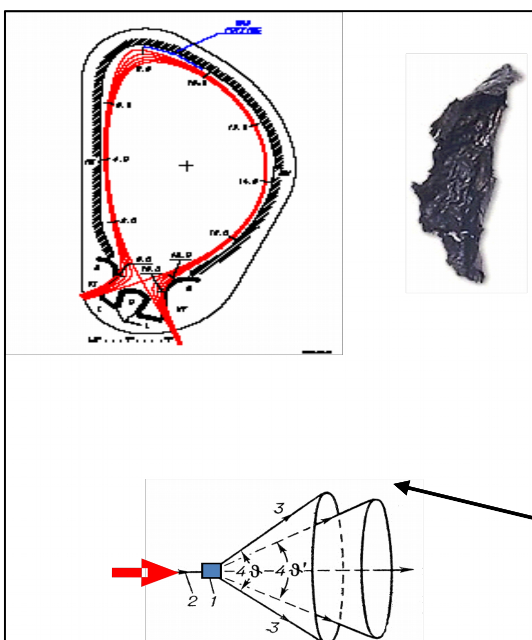
# Carbon nano-structure by X-Ray & neutron diffraction

**Structure identification of amorphous carbonaceous nanomaterials deposited in vacuum chamber of thermonuc. reactor Tokamak T-10 with a joint x-ray and neutron diffraction data analysis**

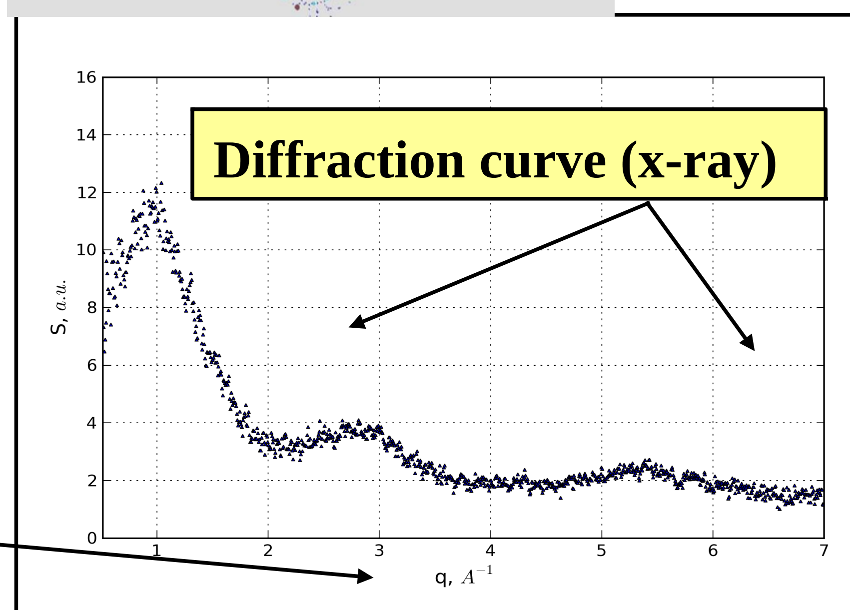




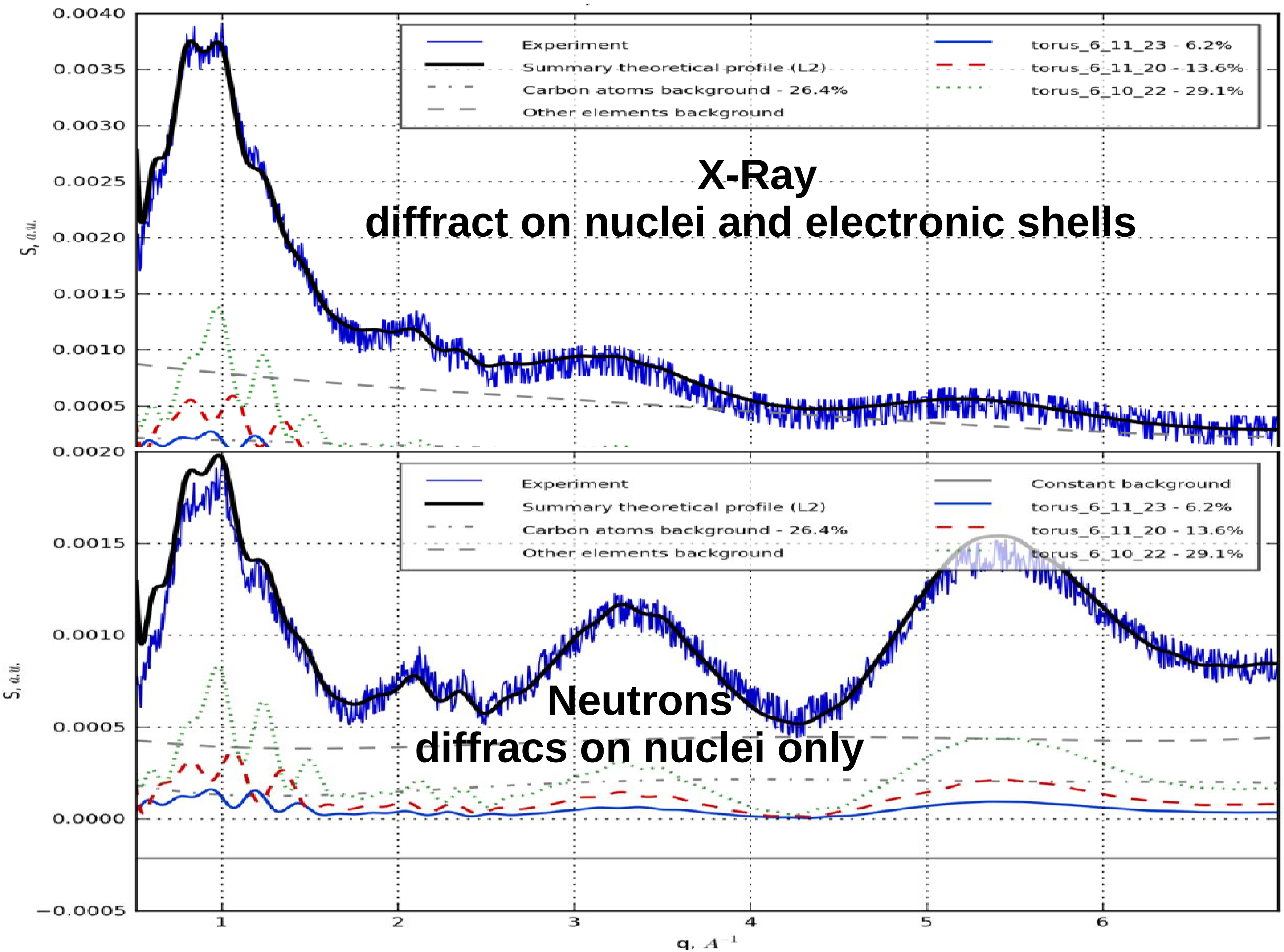




$$q = \frac{4\pi}{\lambda}\sin\left(\frac{\theta}{2}\right)$$

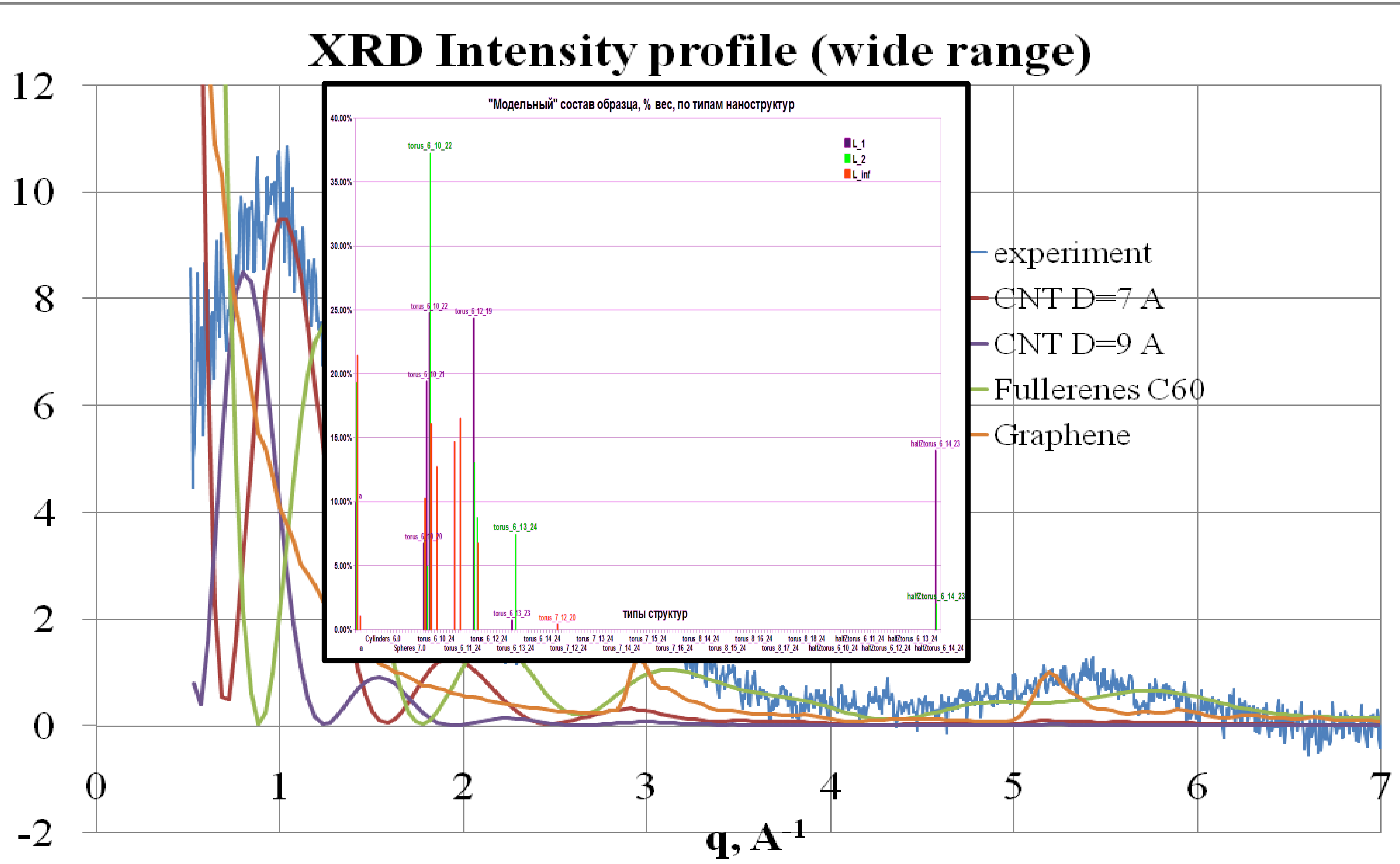


# Diffraction curves for X-Ray & neutron diffraction



# Main approach – sampling curves & opt. fitting

**Modeling diffraction on homogeneous amorphous fractions of nanoparticles, then optimization identification of these fractions' portions in the sample**



# Formalization as Nonlinear Math. Programming Problems

**We use three independent criteria of model-experiment error to get an additional estimate of the accuracy of the final results**

$$K\sum_{j=1}^{m}\left|z_j^{\mathsf{Xr}}\right| + (1-K)\sum_{k=1}^{n}\left|z_k^{\mathsf{Ne}}\right| \xrightarrow[(\mathbf{z}^{\mathsf{Xr}},\mathbf{z}^{\mathsf{Ne}},\mathbf{x},\mathbf{y},a,A,c,B)]{} \min$$

**err. criterion L$_1$**

$$K\sum_{j=1}^{m}\left(z_j^{\mathsf{Xr}}\right)^2 + (1-K)\sum_{k=1}^{n}\left(z_k^{\mathsf{Ne}}\right)^2 \xrightarrow[(\mathbf{z}^{\mathsf{Xr}},\mathbf{z}^{\mathsf{Ne}},\mathbf{x},\mathbf{y},a,A,c,B)]{} \min$$

**err. criterion L$_2$**

$$K\max_{j=1:m}\left|z_j^{\mathsf{Xr}}\right| + (1-K)\max_{k=1:n}\left|z_k^{\mathsf{Ne}}\right| \xrightarrow[(\mathbf{z}^{\mathsf{Xr}},\mathbf{z}^{\mathsf{Ne}},\mathbf{x},\mathbf{y},a,A,c,B)]{} \min$$

**err. criterion L$_{\text{inf}}$**

$$\left\{\begin{array}{l}
z_j^{\mathsf{Xr}} = S_{\mathsf{exp}}^{\mathsf{Xr}}(q_i) - \sum_{i=1}^{N} x_i\cdot S_i^{\mathsf{Xr}}(q_j) - a\cdot S_C^{\mathsf{Xr}}(q_j) - A\cdot S_{\mathsf{impur}}^{\mathsf{Xr}}(q_j)\ \ (j=1:m) \\
z_k^{\mathsf{Ne}} = S_{\mathsf{exp}}^{\mathsf{Ne}}(q_k) - \sum_{i=1}^{N} y_i\cdot S_k^{\mathsf{Ne}}(q_k) - c\cdot S_C^{\mathsf{Ne}}(q_k) - B\cdot S_{\mathsf{impur}}^{\mathsf{Ne}}(q_k)\ \ (k=1:n) \\
\sum_{i=1}^{N} x_i + a = A,\ \ \sum_{i=1}^{N} y_i + c = B,\ \ (x_i, y_i, a, A, c, B) \geqq 0 \\
\boxed{\dfrac{x_i}{A} = \dfrac{y_i}{B}\ (i=1:N),\ \ \dfrac{a}{A} = \dfrac{c}{B}}
\end{array}\right.$$

**Bilinear (!!!) constraints to "reduce to a common denominator" data of two experiments**

# Formulation of B&B algorithm to solve nonlinear MP

$$y_i = t\cdot x_i \ (i{=}1{:}N),\ c{=}t\cdot a,\ B{=}t\cdot A,\quad t_{\text{lo}} \leqslant t \leqslant t_{\text{up}} \quad \equiv \quad \boxed{\frac{x_i}{A} = \frac{y_i}{B}\ (i{=}1:N),\ \frac{a}{A} = \frac{c}{B}}$$

$$\begin{cases} z_j^{\text{Xr}} = S_{\text{exp}}^{\text{Xr}}(q_i) - \sum_{i=1}^{N} x_i\cdot S_i^{\text{Xr}}(q_j) - a\cdot S_C^{\text{Xr}}(q_j) - A\cdot S_{\text{impur}}^{\text{Xr}}(q_j)\ (j = 1:m) \\ z_k^{\text{Ne}} = S_{\text{exp}}^{\text{Ne}}(q_k) - \sum_{i=1}^{N} t\cdot x_i\cdot S_k^{\text{Ne}}(q_k) - t\cdot a\cdot S_C^{\text{Ne}}(q_k) - t\cdot A\cdot S_{\text{impur}}^{\text{Ne}}(q_k)\ (k = 1:n) \\ \sum_{i=1}^{n} x_i + a = A,\ \ (x_i, a, A) \geqq 0,\ \ t_{\text{lo}} \leqslant t \leqslant t_{\text{up}} \end{cases}$$

**Branching over interval of possible values of SCALAR parameter *t* (after fixing – we get convex MP problems)**

$$t_{\text{lo}} \qquad t_{\nu_1} \qquad t_{\nu_2} \qquad t_{\text{up}}$$

**On small sub-intervals bilinear inequalities are relaxed by linear ones – and we get convex (even linear for L1, Linf) MP relaxation problems.**

$$t \in [t_{\nu_1}, t_{\nu_2}]$$

$$t_{\nu_1}\cdot x_i \leqslant y_i \leqslant t_{\nu_2}\cdot x_i\ (i{=}1{:}N),\ t_{\nu_1}\cdot a \leqslant c \leqslant t_{\nu_2}\cdot a,\ t_{\nu_1}\cdot A \leqslant B \leqslant t_{\nu_2}\cdot A$$

# Everest-application implementing alg. By AMPLX

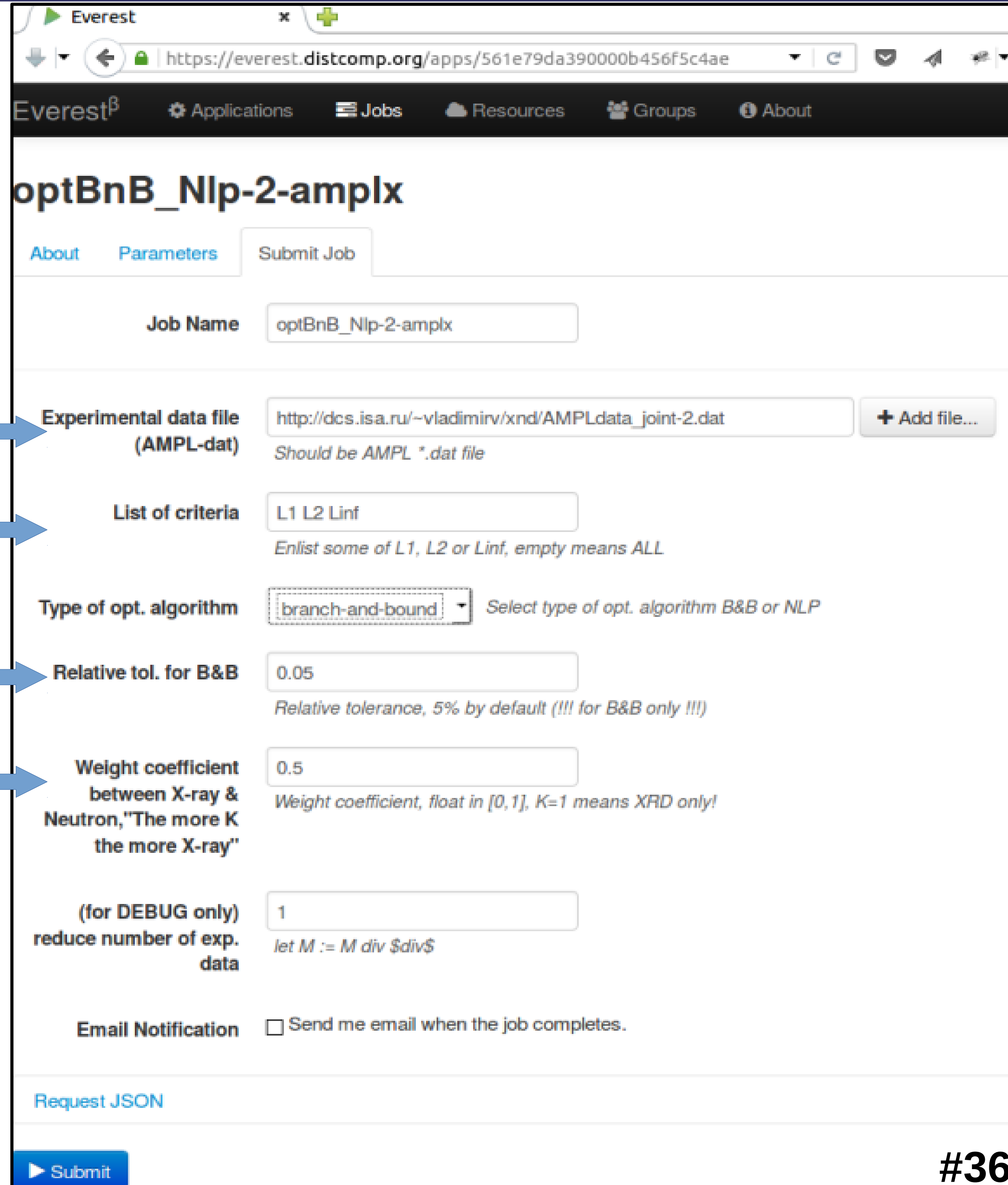
**Job submission Web-form:**

**data-file;**

**criteria to be used;**

**tolerance to stop B&B;**

**X-Ray<>Neutron weight coefficient**



# “Behind the scene” Everest manages AMPLX session jobs

Everest[β] Applications Jobs Resources Groups About optdemo

## Jobs

Auto Update | Update | Filters

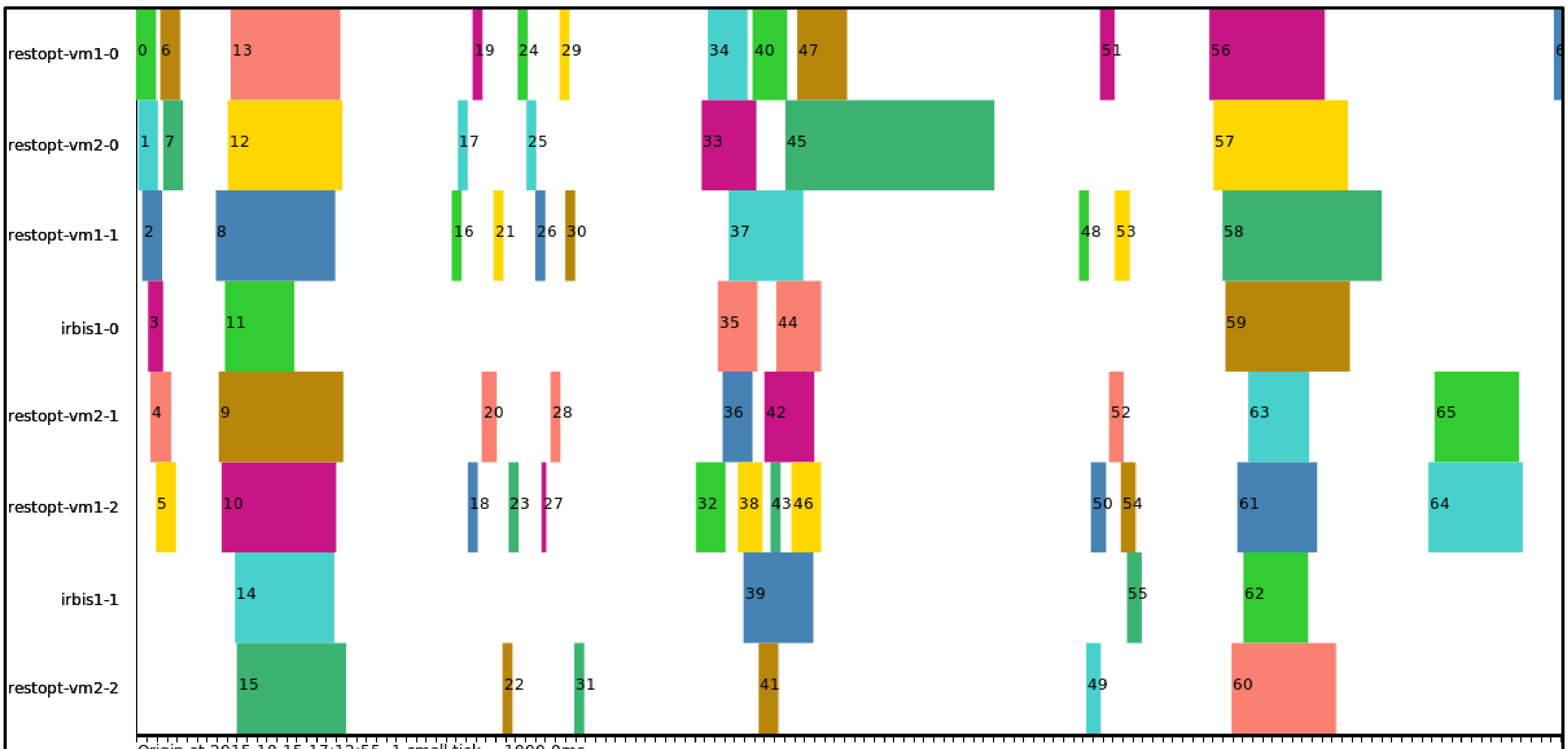
| Name | Application | State | Submitted | Finished | Actions |
|---|---|---|---|---|---|
| amplx4944 - Job 7 | solve-ampl-stub | DONE | 27 Nov 2015 15:40:07 | 27 Nov 2015 15:41:53 | |
| amplx4944 - Job 6 | solve-ampl-stub | RUNNING | 27 Nov 2015 15:40:01 | | |
| amplx4944 - Job 5 | solve-ampl-stub | DONE | 27 Nov 2015 15:39:54 | 27 Nov 2015 15:41:31 | |
| amplx4944 - Job 4 | solve-ampl-stub | DONE | 27 Nov 2015 15:39:49 | 27 Nov 2015 15:41:55 | |
| amplx4944 - Job 3 | solve-ampl-stub | RUNNING | 27 Nov 2015 15:39:43 | | |
| amplx4944 - Job 2 | solve-ampl-stub | RUNNING | 27 Nov 2015 15:39:38 | | |
| amplx4944 - Job 1 | solve-ampl-stub | RUNNING | 27 Nov 2015 15:39:33 | | |
| amplx4944 - Job 0 | solve-ampl-stub | DONE | 27 Nov 2015 15:39:27 | 27 Nov 2015 15:41:09 | |
| amplx4944 - Job 7 | solve-ampl-stub | DONE | 27 Nov 2015 15:37:58 | 27 Nov 2015 15:38:25 | |
| amplx4944 - Job 6 | solve-ampl-stub | DONE | 27 Nov 2015 15:37:54 | 27 Nov 2015 15:38:17 | |
| amplx4944 - Job 5 | solve-ampl-stub | DONE | 27 Nov 2015 15:37:50 | 27 Nov 2015 15:38:14 | |
| amplx4944 - Job 4 | solve-ampl-stub | DONE | 27 Nov 2015 15:37:46 | 27 Nov 2015 15:38:09 | |
| amplx4944 - Job 3 | solve-ampl-stub | DONE | 27 Nov 2015 15:37:42 | 27 Nov 2015 15:38:10 | |
| amplx4944 - Job 2 | solve-ampl-stub | DONE | 27 Nov 2015 15:37:38 | 27 Nov 2015 15:38:02 | |
| amplx4944 - Job 1 | solve-ampl-stub | DONE | 27 Nov 2015 15:37:34 | 27 Nov 2015 15:38:02 | |
| amplx4944 - Job 0 | solve-ampl-stub | DONE | 27 Nov 2015 15:37:30 | 27 Nov 2015 15:37:48 | |
| optBnB_Nlp-2-amplx | optBnB_Nlp-2-amplx | RUNNING | 27 Nov 2015 15:36:42 | | |

# Problem of effective resource load (1)

**Everest logs analysis via Everest Python API helps to increase effectiveness of resource usage**

**Tracing of session with successive solving of three problems (L1, L2, Linf) in one OptBnb* job.**

**More than 60 sub-tasks took ~25 minutes. Not more than 8 solvers (from 16 available) worked in parallel**

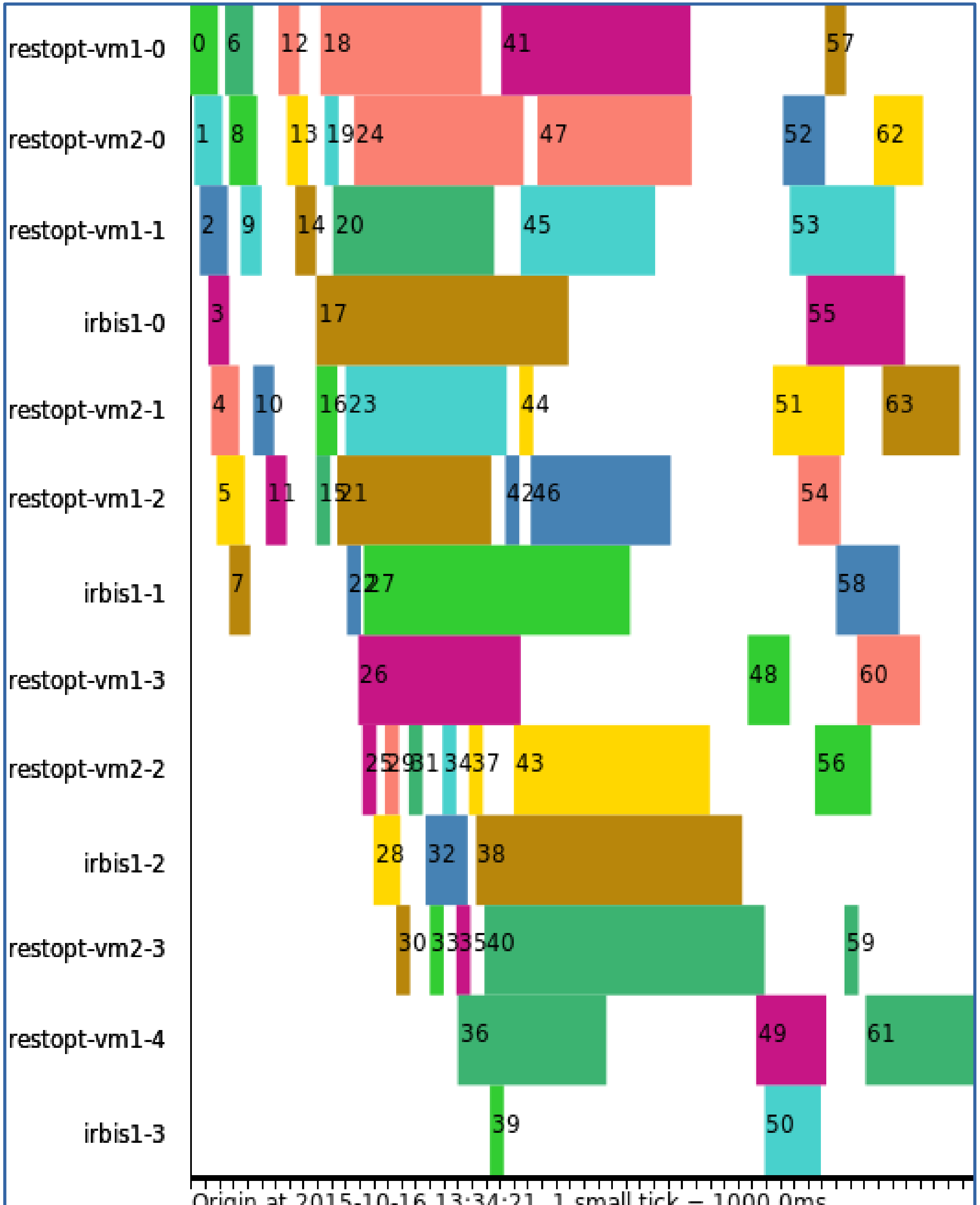


# Problem of effective resource load (2)

**Parallel submission of three problems (L1, L2, Linf) in three OptBnb* jobs.**

**Three jobs took ~10 min.**

**13 solvers (from 16 available) worked in parallel**



# Coarse-grained & fixed decomposition for MILP

**Relatively "non-standard" approaches for discrete (MILP) problems**

**Preliminary analysis of constraints, then "fixed" decomposition into sub-problems might be solved in parallel**

**Two examples (both based on AMPLx):**

**1. Local Elimination Algorithms (LEA) for MILP with quasi-block constraints' structure investigated by Dr. Oleg A. Shcherbina (Crimean Federal University, Institut für Mathematik Universität Wien, Austria) (here we collaborate with prof. Vladimir I.Tsurkov, Computing Center RAN)**

**2. Coarse-grained B&B for MILP with preliminary heuristic fixed decomposition into subproblems by fixing some of binary variables**

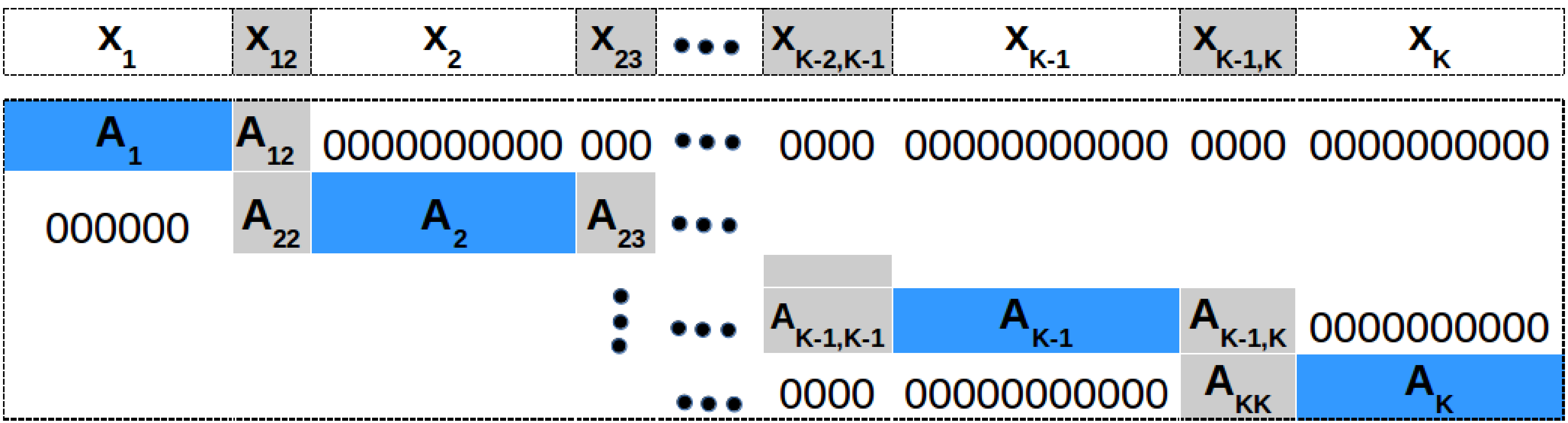
# Local Elimination Algorithms for quasi-block MILP

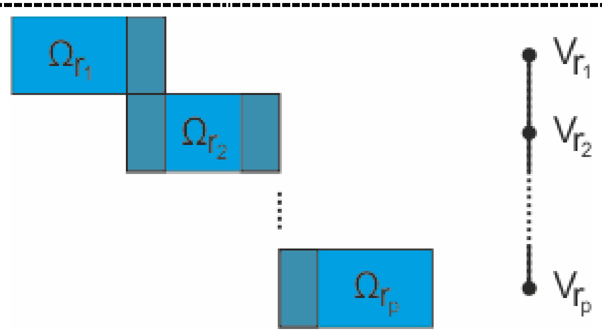
$$c^{\mathbf{T}}x \to \max$$
$$\mathbf{Ax} \leq \mathbf{b}$$
$$\mathbf{x} \in \{0,1\}^n$$

$$\begin{cases} c_1x_1 + c_2x_2 + \ldots + c_nx_n \to \max, \\ a_{11}x_1 + \ldots + a_{1n}x_n \leq b_1, \\ \ldots \\ a_{m1}x_1 + \ldots + a_{mn}x_n \leq b_m, \\ \{x_1;\ldots;x_n\} \in \{0;1\}. \end{cases}$$

**"free" vars**

**"binding" vars**




**MILP with quasi-block structure, of "stairs" type.**

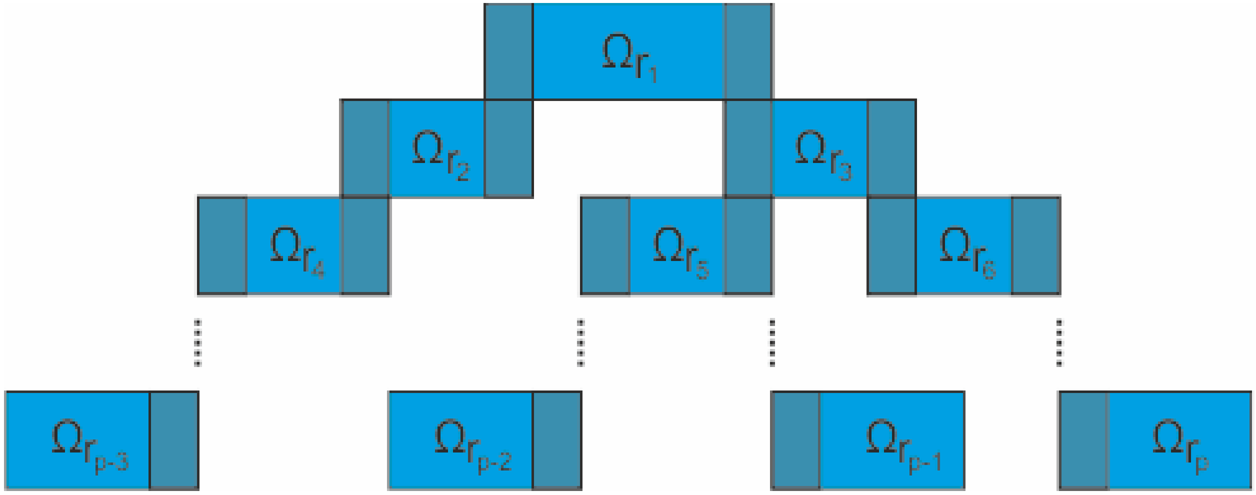
# LEA for MILP with tree-type quasi-block constraints

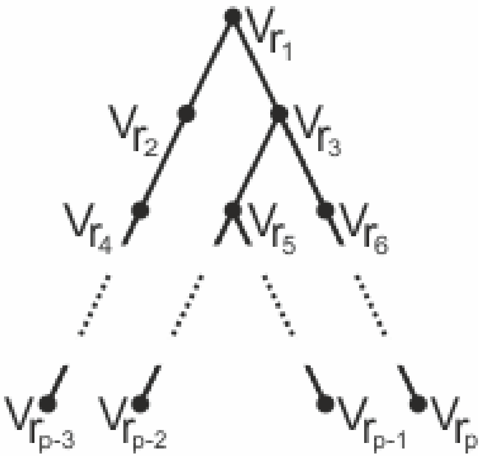
$$c^{\mathbf{T}}x \to \max$$
$$\mathbf{Ax} \leq \mathbf{b}$$
$$\mathbf{x} \in \{0,1\}^n$$

$$\begin{cases} c_1x_1 + c_2x_2 + \ldots + c_nx_n \to \max, \\ a_{11}x_1 + \ldots + a_{1n}x_n \leq b_1, \\ \ldots \\ a_{m1}x_1 + \ldots + a_{mn}x_n \leq b_m, \\ \{x_1; \ldots; x_n\} \in \{0; 1\}. \end{cases}$$

**“free” vars/block**

**“binding” vars/blocks**



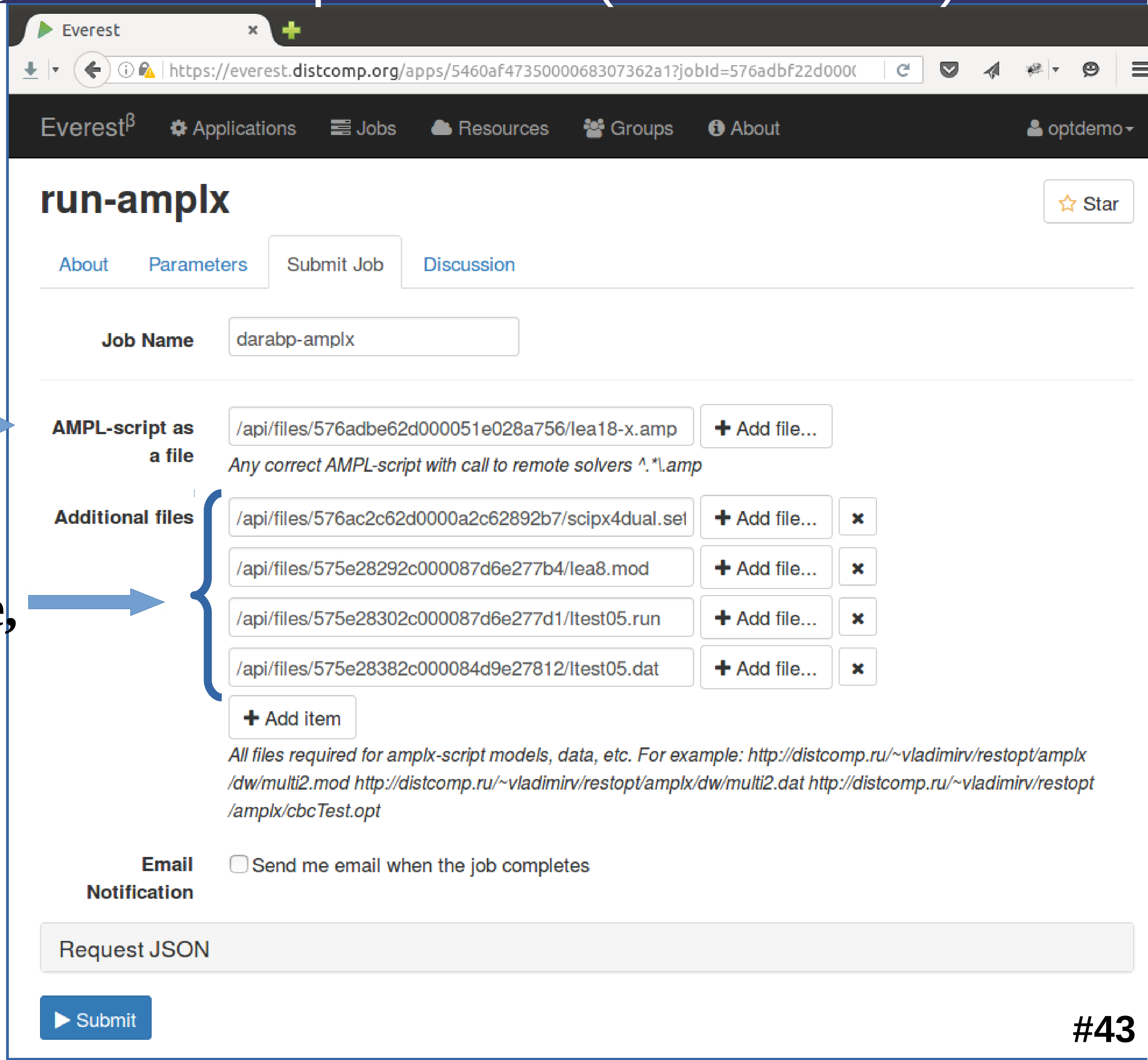




**MILP with quasi-block structure, of “tree” type.**

# LEA @ AMPLx experiments (Submit form)

**LEA as AMPLX**

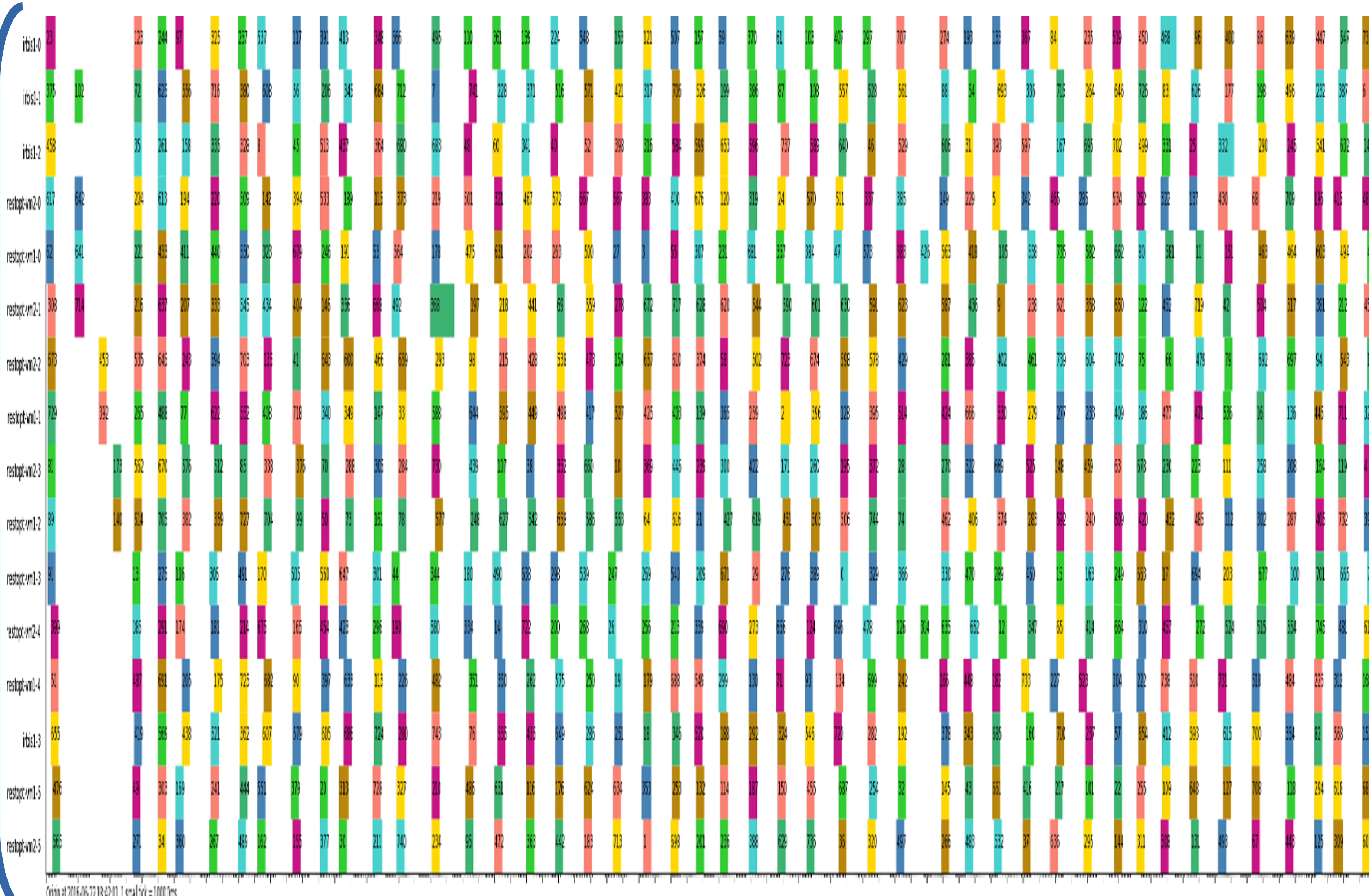
**Data, quasi-block structure, etc**



# LEA @ AMPL experiments (Time-profile-plotting)

**100x50000, quasi-block, tree-type, ~750 subtasks**

**Everest processors**

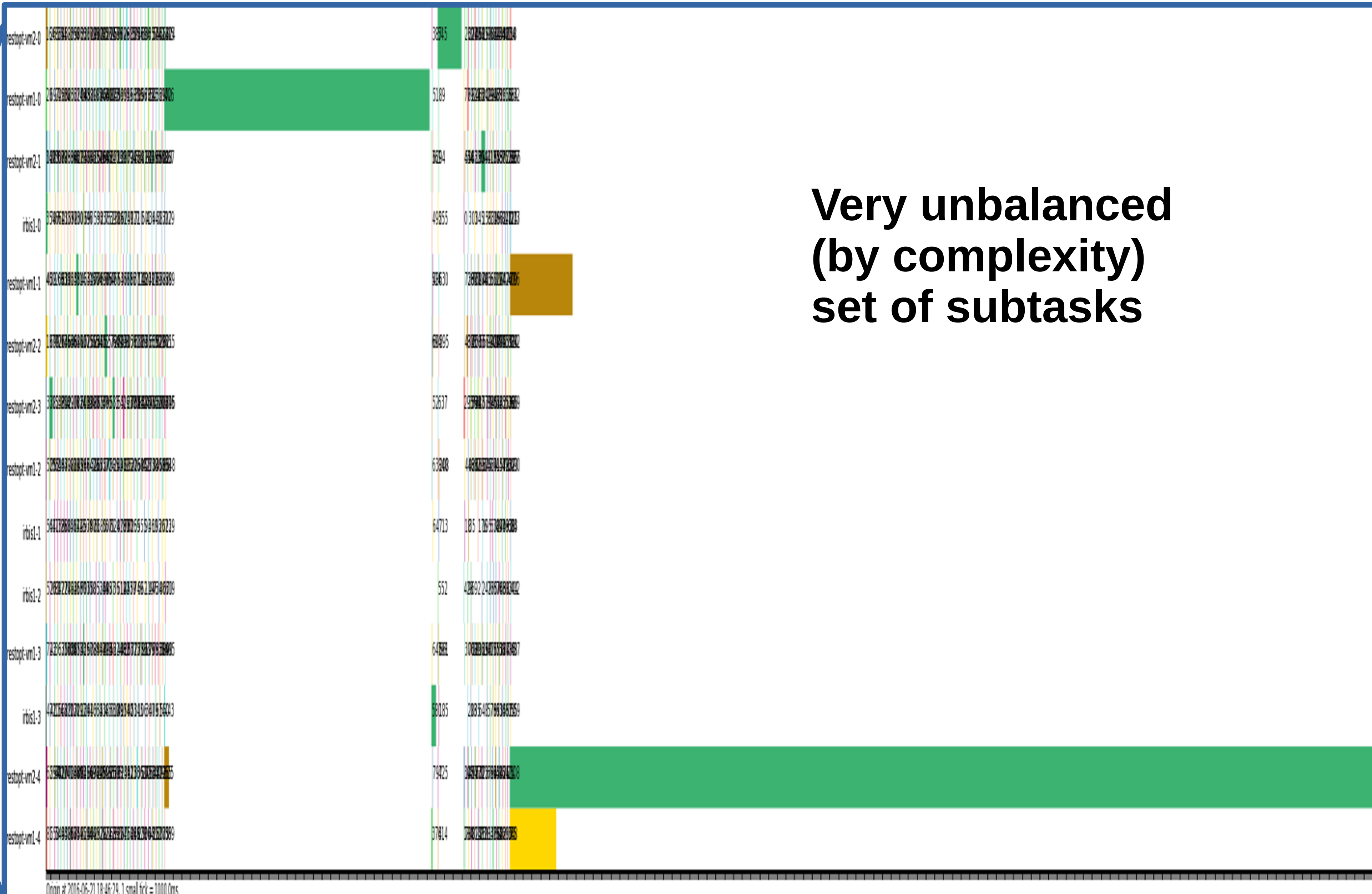


**Time, 0 <=======> ~15 min (standalone > 3 hours)**

# LEA @ AMPL experiments (Time-profile-plotting)

**100x100000, quasi-block, tree-type, ~850 subtasks**

**Everest processors**

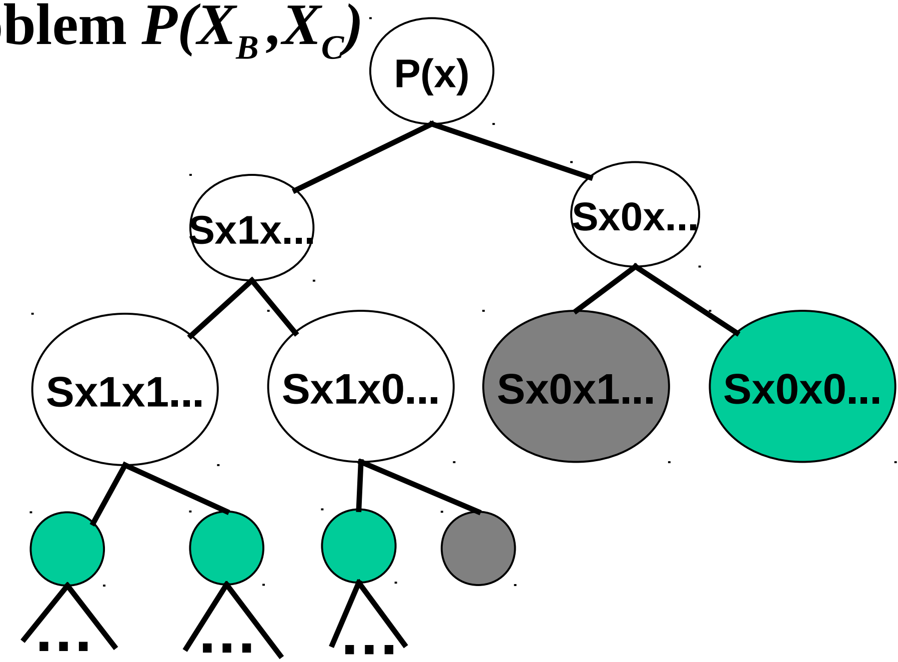



**Time, 0 <=========> ~5.5 hours (standalone ????)**

# Branch-and-bound for MI... problem (e.g. boolean)

**General scheme of <u>search tree</u> traversal for problem $P(X_B, X_C)$**

$$f_0(X_B, X_C) \to \min_{X_B, X_C} \quad (X_C, X_B) \in Q$$




**Current state of B&B (changed dynamically):**

**- list of nodes to be processed (green);**

**- known Upper-Bound (aka <u>incumbent</u> | record)**

$$\mathbf{UB} = f_0(X'_B, X'_C) : \quad (X'_C, X'_B) \in Q$$

**Node (subproblem) operation:**

**1) calc. Lower-Bound of S, LB(S), by relaxation of boolean constraints to, e.g. LP;**

**2) if, accidentally, feasible set of variables found** $(X''_C, X''_B) \in Q \Rightarrow$ **update UB:**

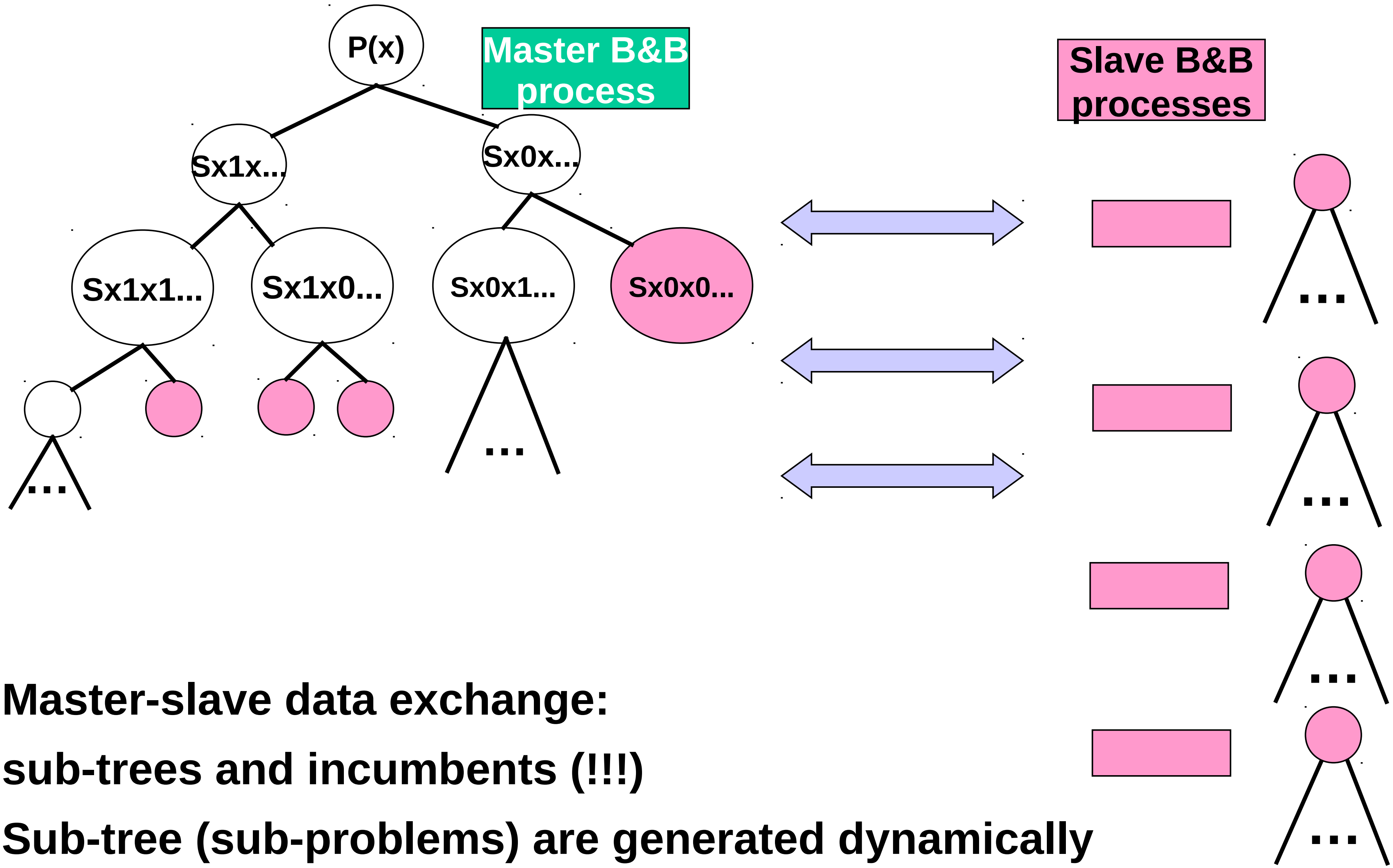
$$\mathbf{UB} := \min \{\mathbf{UB}, f_0(X''_B, X'')\}$$

**3) if LB(S) >= UB – discard node from the list (grey);**

**4) select boolean variable to split node and add new ones to the tree**

**B&B is one of the best algorithms which is suited for parallel implementation**

# Fine-grained decomposition of B&B (traditional approach)



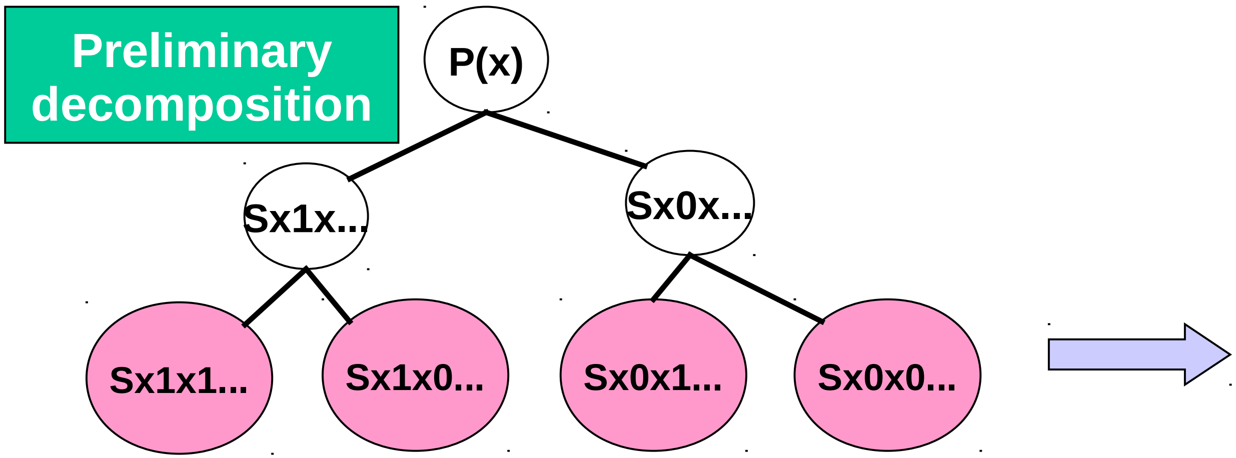

**Master-slave data exchange:**

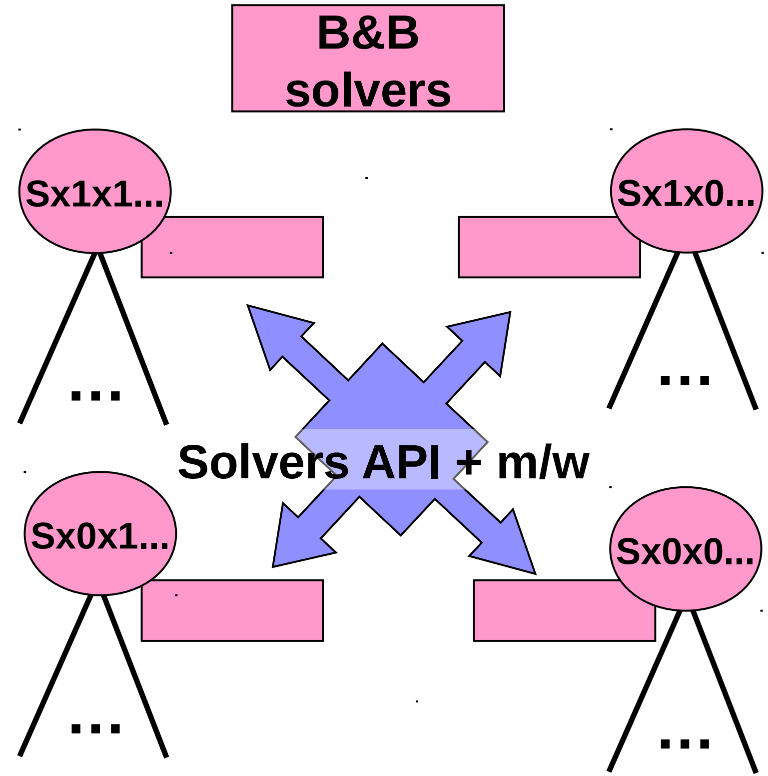
**sub-trees and incumbents (!!!)**

**Sub-tree (sub-problems) are generated dynamically**

**Usually, the approach is based on MPI and run at high-performance cluster**

# Coarse-grained (“static”) decomposition of B&B




**B&B solvers exchanges with incumbents only (!!!)**

**The approach is not so popular as fine-grained one, but is much more easy for implementation via solvers’ API and some “light-weight” middleware, e.g. Erlang, Zeroc Ice, ZeroMQ etc.**

**Preliminary decomposition is crucial for speed-up and requires analysis of the problem’s data ! E.g. by AMPL (!)**

# Traveling Salesmen Problem by Coarse-grained B&B (1)

$$\sum_{i>j} d_{ij} x_{ij} \to \min_{x_{ij}, f_{ij}} \text{ wrt:}$$

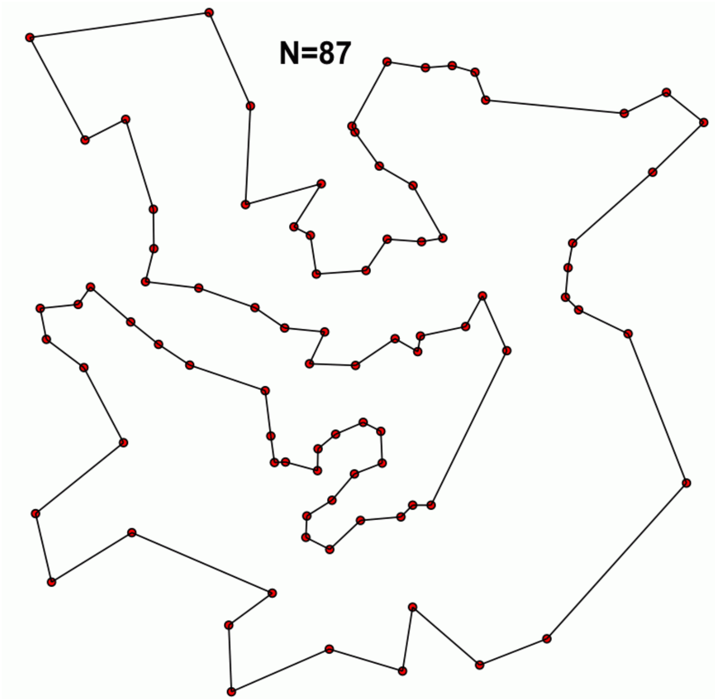
$$\sum_{j \in V, i>j} x_{ij} + \sum_{j \in V, i<j} x_{ij} = 2 \quad (i \in V = \{1:n\});$$

$$f_{ij} \le \left( \begin{cases} n, \text{ if } i = 1 \\ n-1, \text{ if } i > 1 \end{cases} \right) * \left( \begin{cases} x_{ij}, \text{ if } i < j \\ x_{ji}, \text{ if } i > j \end{cases} \right) \quad ((i,j) \in V \times V);$$

$$\sum_{j:(i,j) \in V \times V} f_{ij} - \sum_{j:(i,j) \in V \times V} f_{ji} \le \begin{cases} n-1, \text{ if } i = 1 \\ -1, \text{ if } i > 1 \end{cases} \quad (i \in V);$$

$$\sum_{j:(i,j) \in V \times V} f_{ij} \ge 1 \quad (i \in V);$$

$$x_{ij} = \{0, 1\}.$$




**"Random" selection of $x_{ij}$ to decompose doesn't give speed-up**

**Heuristic rule: sort $\{d_{ij}\}$ in ascending order and decompose by $x_{ij}$:=0|1 corresponding to the smallest $d_{ij}$ (to get "balanced" by incumbents subproblems ??)**

**Subproblems has been generated as AMPL-stubs by special AMPL "preprocessing" script**

# Traveling salesmen problem coarse-grained experiment (2)

## dCBC prototype (CBC, CBC API + Erlang)

| N | T(CBC), min | T(SCIP), min |
|---|---|---|
| 80 | 5.3 | 1.6 |
| 90 | 20.3 | 6 |
| 100 | 623 | 10 |
| 110 | >10000 | 75 |

| N | n of Xij fixed | n of subprobs | T(CBCx1), min | T(dCBC), min | Speed Up |
|---|---|---|---|---|---|
| 80 | 4 | 16 | 5.3 | 2 | 200% |
| 90 | 5 | 32 | 20.3 | 11 | 300% |
| 100 | 6 | 64 | 623 | 229 | ∞ !!! |
| 110 | 7 | 128 | >10000 | 1212 | |

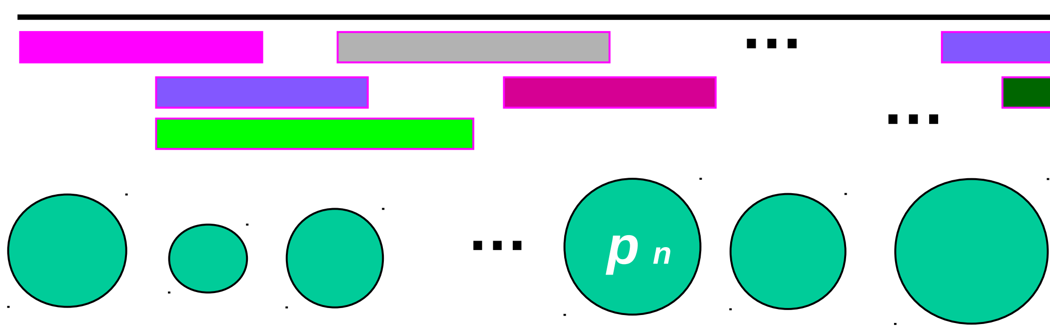
**Computing resources (12 CBC instances) :**

**8 CBC instances at 2 x Intel Xeon E5620 @ 2.40GHz**

**4 CBC instances at Intel Core i7-2600K @ 3.40GHz**

# Task-to-worker scheduling problem (fully deterministic)

**Queue (with arrival times) of tasks of known complexities (processing times)**




**Pull of "workers" of known comp. power, $p_n$**

$T_k$, $\tau_k$ — $T_k$ – arrival time, $\tau_k$ – *processing time for "unit" of comp. power*

**Need to determine a "schedule", i.e. set of variables $\{x_{kn}, t_k\}$:**

**- boolean $x_{kn} = 1$ if task "*k*" is submitted to worker "*n*" (*0* – if not);**

**- continues $t_k \geq T_k$ – task submission time.**

**Constraints:**

**- $t_k \geq T_k$ (submission after arrival)**

**- each worker can process only one task at a time.**

**Objective: minimize time of queue completion**

$[t_k \;,\; t_k + (\tau_k / p_n)]$ **- actual task's span, if $x_{kn} = 1$**

# Task-worker scheduling problem coarse-grained experiment (1)

$$z \to \min_{t_k, x_{kn}, z} \text{ w.r.t:}$$

$$t_k \ge T_k \quad (k = 1:K);$$

$$\sum_{n=1:N} x_{kn} = 1 \ (k = 1:K) \quad \left( \text{or} \sum_{n=1:N} x_{kn} \ge 1 \ (k = 1:K) \right);$$


$$t_k + \frac{\tau_k}{p_n} x_{kn} \le t_{k'} + C \cdot (2 - x_{kn} - x_{k'n}) \ (k = 1:K, k < k' \le K, n = 1:N);$$

$$t_k + \frac{\tau_k}{p_n} x_{kn} \le z \ (k = 1:K, n = 1:N);$$

$$z, t_k \in \mathrm{R}^1, \ x_{kn} = \{0 \mid 1\}, \ k = 1:K, n = 1:N$$

**“Random” selection of $x_{kn}$ to decompose doesn’t give speed-up**

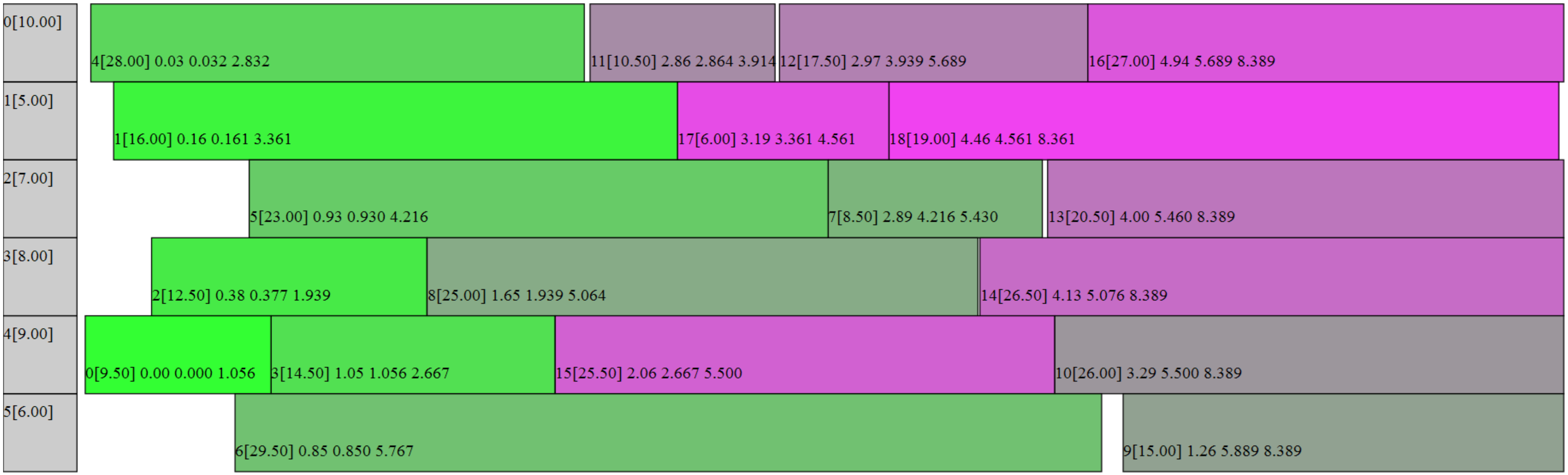
**Heuristic rule: sort $\{\tau_k/p_n\}$ in ascending order and decompose by $x_{kn}$:=0|1 corresponding to the smallest $\tau_k/p_n$**

**(to get “balanced” by incumbents subproblems ??)**

**Subproblems has been generated as AMPL-stubs by special AMPL “preprocessing” script**

# Task-worker scheduling problem coarse-grained experiment (2)

## dCBC prototype (SCIP, SCIP API + Erlang)




**6 workers, 19 tasks, exact solution**

**Computing resources (40 SCIP) :**

| host | n of SCIP instances | T(SCIPx1), sec |
|---|---|---|
| server-1 | 8 X<br>Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz | 929.7 |
| server-2 | 16 X<br>Intel(R) Xeon(R) CPU E5620 @ 2.40GHz | 1368.59 |
| xen-vm-2 | 8 X<br>Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz | 2172.27 |
| xen-vm-1 | 8 X<br>Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz | 2270.38 |

**6 boolean $x_{kn}$ has been fixed**

**(64 subproblems) took 720 sec.**

**The result is rather poor, speedup is less than 25% (720 vs 930 seconds).**

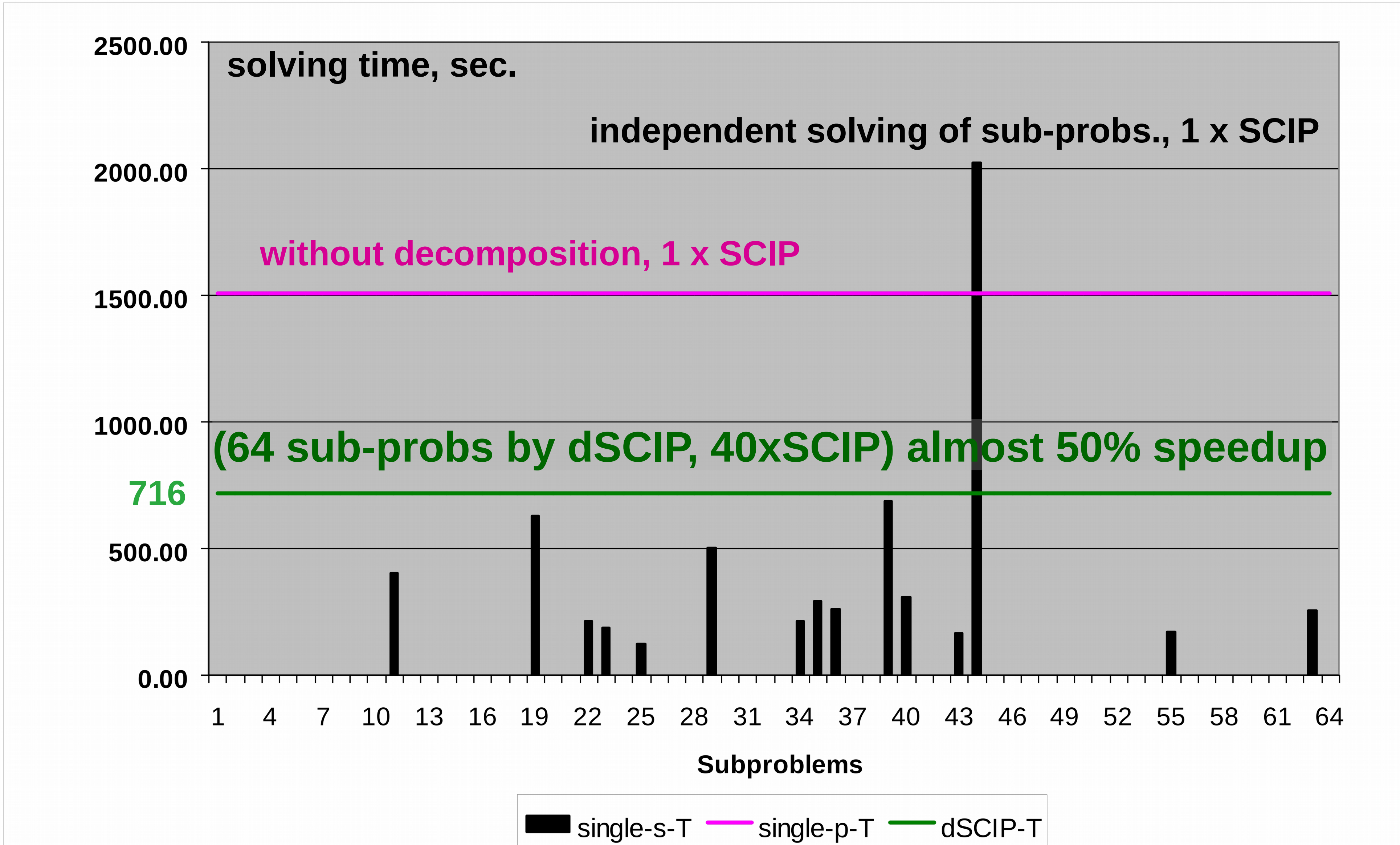
**Very different performance, no load balance.**

# Task-worker scheduling problem coarse-grained experiment (3)

## 2 x (20-cores VM at www.DigitalOcean.com)

**40 SCIP at QEMU Virtual CPU version 1.0 @ 2.4Ghz**

**The same decomposition by 6 boolean vars. into 64 sub-problems**



# TO-DO Plans

**Increase computing power of the computing resources (dedicated for optimization) connected to Everest:**

**stand alone servers and server with**
**Intel Xeon Phi co-processor (+ ~50 cores);**
**small cluster deploying now in our Center (+~20 cores).**

**Use Everest Task Protocol and special “multi-task” Everest jobs to exchange message within special AMPLx session**

**To allow to use in AMPLx “pure” Python computing scenarios on the base on Pyomo, http://www.pyomo.org, an open source package supporting AMPL-stub/solution formats and compatible with AMPL-solvers**

# Instead of conclusion

**Our contacts:** **http://distcomp.ru,**

**Everest platform web-site:** **http://everest.distcomp.org,**

**AMPLX sources:** **https://gitlab.com/ssmir/amplx**

**Examples of AMPLx-scripts:**

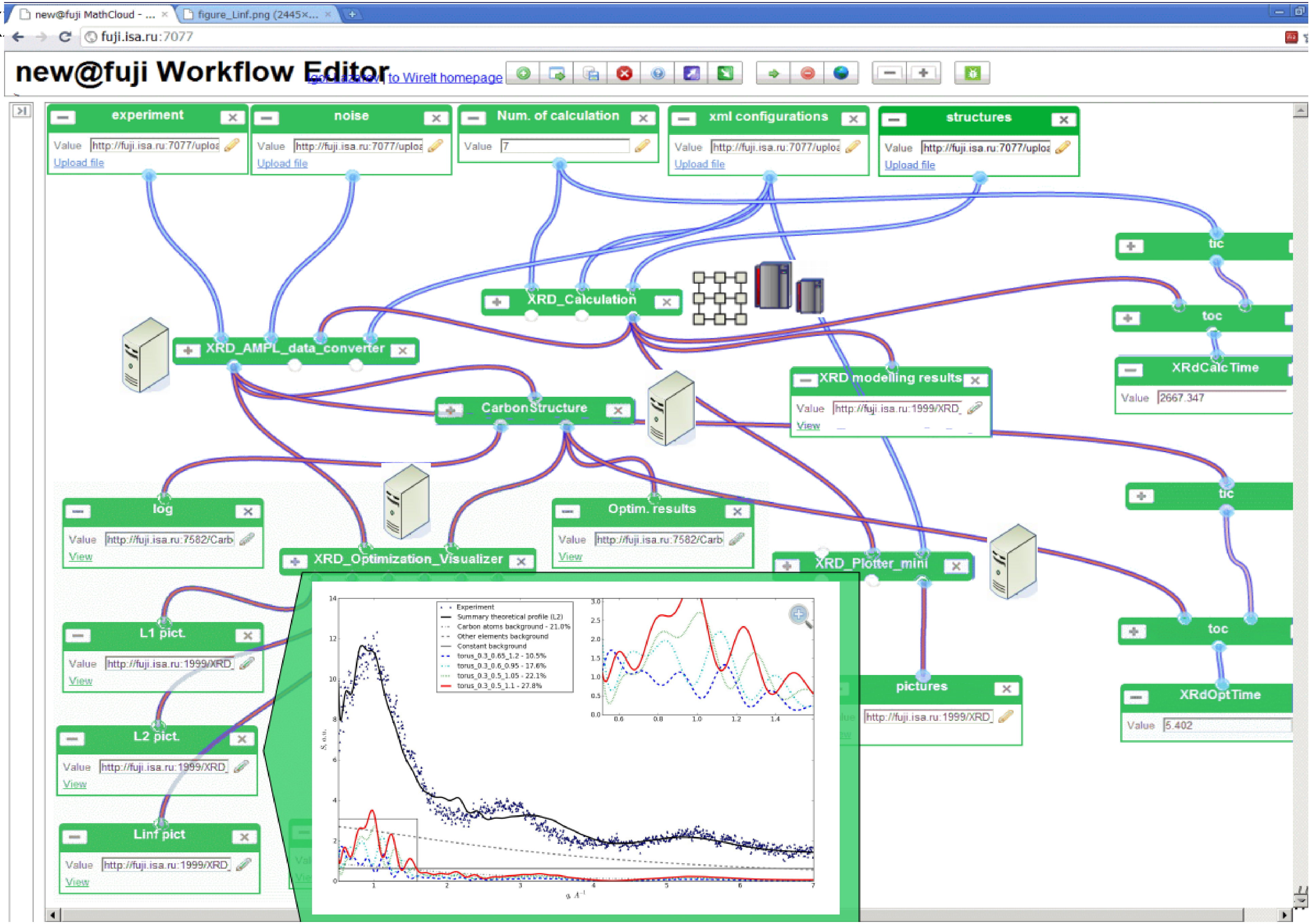
**http://distcomp.ru/~vladimirv/restopt/amplx**

**Thank you for your attention.**

**Questions?**

# Visual “spaghetti-wire” programming vs. scrpting

x



# Visual “spaghetti-wire” programming vs. scrpting

## Too complex even for simple calculation