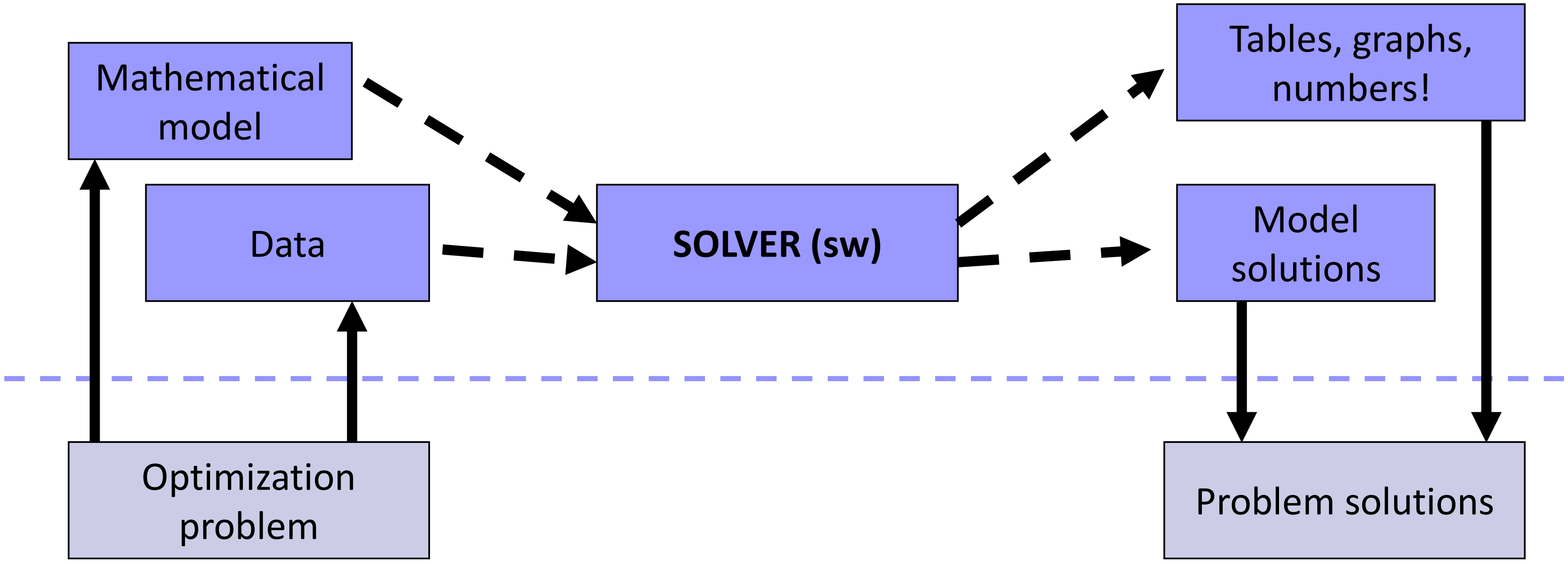
Solvers for Mathematical Programming

Solvers (optimizing engines)

A **solver** is a software application that takes the description of an optimization problem as input and provides the solution of the model (and related information) as output.

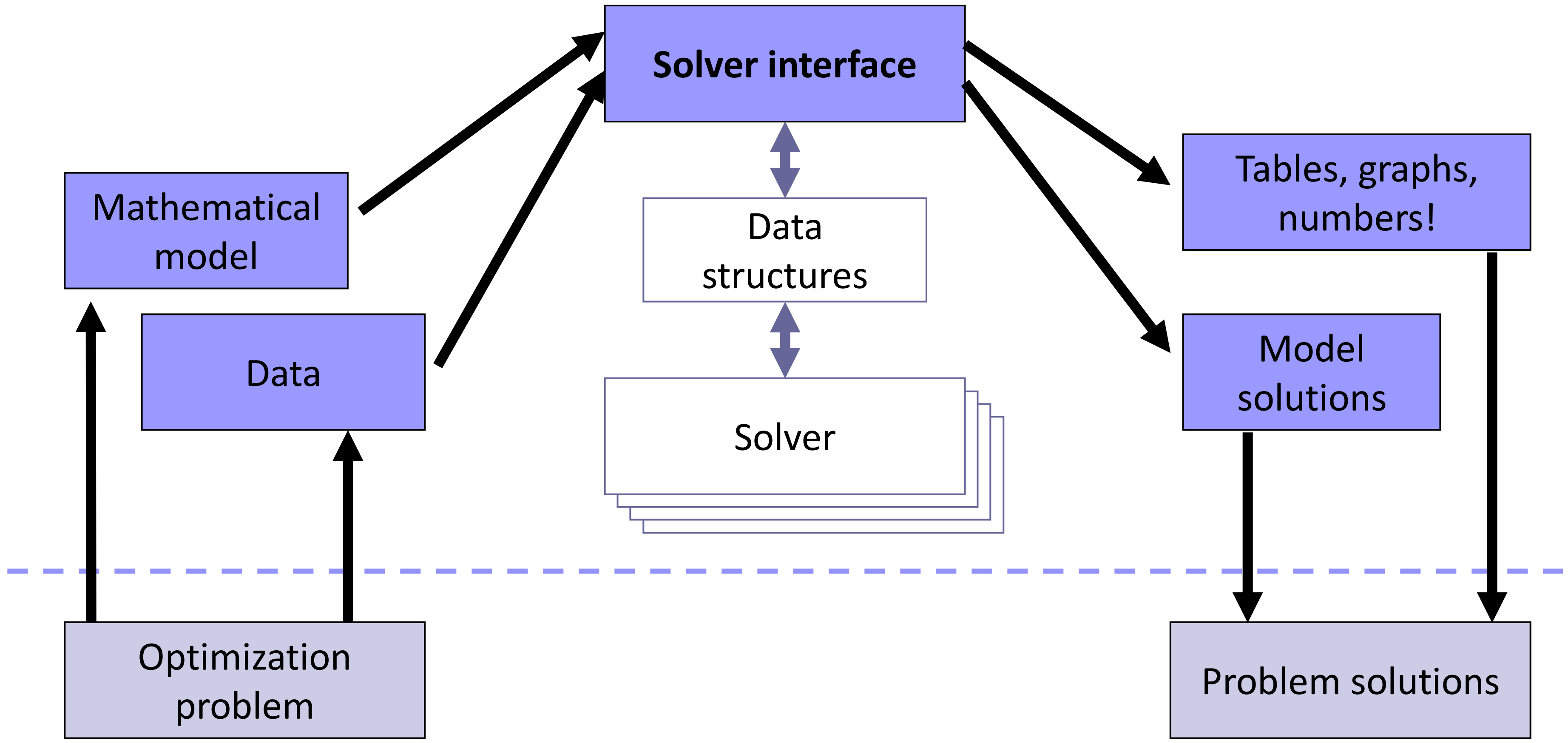


MILP solvers

- **M**ixed **I**nteger **Linear P**rogramming solvers most used in practice:
  - very efficient
  - numerical stability
  - easy to use or embed
- 1 000 000 000 speed-up in the last 15 years
  - hardware speed-up: x 1000
  - simplex improvements: x 1000
  - branch-and-cut improvement: x 1000
- Cplex, Gurobi, Xpress, Scip, Lindo, GLPK, Google OR Tools **etc.**

Solver interfaces

A solver can be accessed via **modelling languages** or **general-purpose-language libraries**



IBM Ilog Cplex

- One of the first MILP solvers
- Includes **state-of-the-art** technology
- One of the best solvers available (Gurobi, Xpress)
- Possible interfaces
  - Interactive optimizer
  - **OPL** / AMPL / ZIMPL … algebraic modelling language
  - **C – API libraries (Callable libraries)**
  - C++ libraries (Concert technologies)
  - **Python (with *docplex*)** / Java / .Net wrapper libraries
  - Matlab / Excel plugins

Accessing / Getting IBM Ilog Cplex

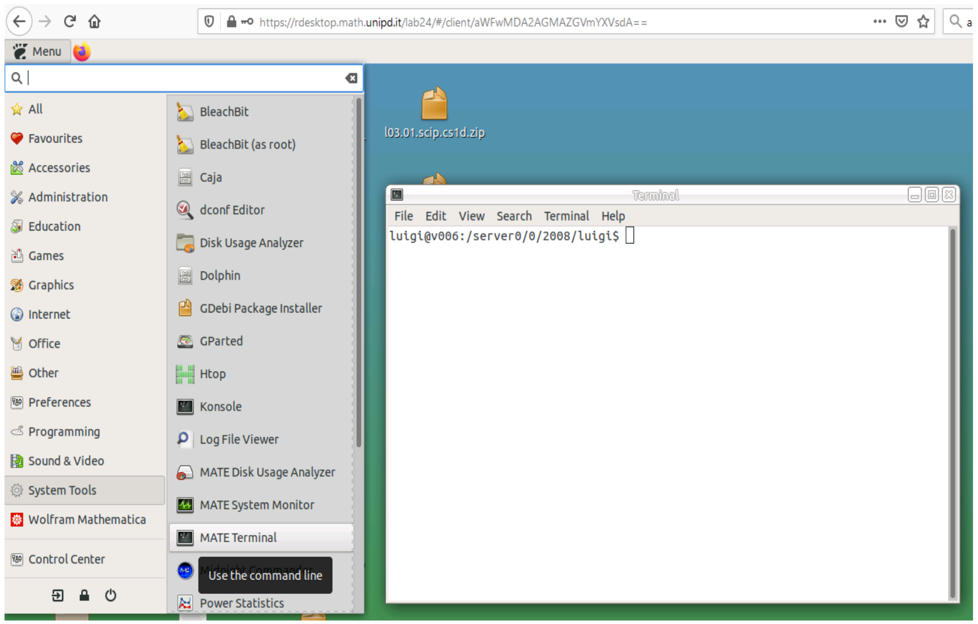
- Installed at LabTA/LabP140 and virtual *Lab24hr*
- From home
  - Getting your own free academic license (!)
  - Virtual *Lab24hr*
  - Accessing OPL via ssh / X-windows (or similar)
  - Accessing Cplex via ssh
- See Getting access to Lab resources: instructions for details!

Optimization Programming Language - OPL

- Close to algebraic modelling language
  - direct mapping of sets, parameters, decision variables, constraints
  - use algebraic primitives (`forall`, `sum` etc.)
- Integrated Development Environment (IDE) available
- Included in the Cplex Studio package
- Learning OPL by examples

Basic commands (in Lab)

In a **`terminal`** window
(e.g. MATE Terminal)



```
luigi@v006:/server0/0/2008/luigi$
```

- To enable Cplex Studio

  `.  cplex_env` (notice “dot blank”)

- To run the OPL IDE

  `[/opt/ibm/ILOG/CPLEX_Studio128/opl/]`**`oplide`**

IDE commands

- Basic OPL projects
  - **model files** (.mod): models in OPL language
  - **data files** (.dat): parameters data
  - **Run Configurations**: collect models and data to configure a specific problem instance
- Basic IDE commands
  - `File->New->OPL Project`
    (create a new project in a specific directory)
  - `File->Import->Existing OPL Project`
    (open an existing project)
  - `Help->Help Contents->IDE and OPL-> Optimization Programming Language (OPL)`

A first simple model `[1.mix_perfumes]` 1/2

- decision variables:

```
dvar <dvar_type> decision_variable_name;
```

`<dvar_type>`	=	`float`	(real variables)
		`float+`	(real variables ≥ 0)
		`int`	(integer variables)
		`int+`	(integer variables ≥ 0)
		`boolean`	(binary variables)

- Objective function:

  `maximise` (or `minimise`) `<expression>;`

A first simple model `[1.mix_perfumes]` 2/2

- Constraints:

```
subject to {
 constraint1_name: <expression> = <expr.>;
 constraint2_name: <expression> <=<expr.>;
 ...
}
```

`<expression>` = e.g.

```
4*var1 + 5*var2 + log(10)*var3 >=
                       var4 - 0.7*var5
```

try with diet_food...

Generalizing the model [3.mix_general_model] 1/2

- Sets

```
setof(<data_type>) set_name = { <element_list> };
<data_type> = string, int, float, etc. etc.
```

- Parameters

```
<data_type> parameter_name = parameter_value;
<data_type> 1dim_vector_name[set_name] =
                           [element1,element2,...];
<data_type> 2dim_vector_name[set1][set2] = [
     [element_1_1,element_1_2, element_1_3, ...],
     [element_2_1,element_2_2, element_2_3, ...],
     ...
];
<data_type> Ndim_vec[set1][set2]…[setN] = …
```

(**N** nesting levels of [ ])

Generalizing the model `[3.mix_general_model]` 2/2

- Constraints

```
forall ( k in set ) {
     constraint_name: <expression using index k>
}
```

- Decision variables

```
dvar <dvar_type> decision_variable_name;
dvar <dvar_type> 1dim_dec_var_vector[set_name];
dvar <dvar_type> 2dim_dec_var_vector[set1][set2];
dvar <dvar_type> Ndim_dec_var[set1][set2]…[setN];
```

`<expression using index k>` = e.g.

```
  sum(i in I,j in J)(x[i,k] + L[i]*y[i,j]) >= D[k]
```

Separating model and data

[4.mix_general_dataout] 1/4

- .mod file (cont.)

```
//sets
setof(<data_type>) set_name = ...;


//parameters
<data_type> parameter_name = ...;
<data_type> 1dim_vector_name[set_name] = ...;
<data_type> 2dim_vector_name[set1][set2] = ...;
<data_type> Ndim_vec [set1][set2] ]…[setN] = ...;
```

Separating model and data

[4.mix_general_dataout] **2/4**

- (cont.) **.mod** file

```
//decision variables
dvar <dvar_type> decision_variable_name;
dvar <dvar_type> 1dim_dec_var_vector[set_name];
dvar <dvar_type> 2dim_dec_var_vector[set1][set2];
dvar <dvar_type> Ndim_dec_var[set1][set2]…[setN];
```

Separating model and data

`[4.mix_general_dataout]` **3/4**

- `.dat` file

```
set_name = { element1, element2, ...}

parameter_name = <value>;
1dim_vector_name = [element1,element2,...];
2dim_vector_name = [
     [element_1_1,element_1_2, element_1_3, ...],
     [element_2_1,element_2_2, element_2_3, ...],
     ...
];
```

Separating model and data

`[wines.dat]` **4/4**

- Change data to solve the following problem:

  we produce two types of wines (wine1 and wine2) using four types of grapes (grape1, grape2, grape3 and grape4). The unit profit for wines is respectively 21 and 10 euros. The availability of grapes is respectively 100, 200, 50 and 150 units. A unit of wine1 requires 1.5, 0.8, 1.0 and 0.3 units of grapes 1, 2, 3 and 4 respectively. A unit of wine2 requires 1.0, 2.0, 0.5 and 1.1 units of grapes 1, 2, 3 and 4 respectively. Determine the production mix to maximize the profit

try with cover models…

Exercises

- Min cost covering `[cover.mod, cover.food.dat]`
- Basic transportation model `[`*`transport`*` OPL project]`
  - Additional constraint 1: if the cost of link from i to j is at most *LowCost*, then the flow on this link should be at least *LowCostMinOnLink*
  - Additional constraint 2: destination *SpecialDestination* should receive at least *MinToSpecialDest* units from each origin, but for origin *SpecialOrigin*
- Facility location with fixed costs `[`*`LocationWithFixedCosts`*` OPL project]`
  - **Expressions** to define data-dependent big-M constants
  - Additional constraint: at most/least max/min number of open locations
  - **`New – settings`** : ".ops" files (optimization parameters, e.g. global time limit)
- OPL project, model and data for *(do it yourself!)*
  - the "Moving scaffolds between yards" problem
  - The "Four Italian friends" problem

OPL «advanced» topics: modeling tools

- **dexpr**: factorizes a recurrent expression involving `dvars`

```
dexpr float total_cost = sum(i in I) C[i] *x[i];
minimize total_cost;
```

- **tuple**: custom "type" that collects data

```
setof(int) NODES = asSet(1..10);
tuple directed_arc {
  int node_from;
  int node_to;
}
setof(direct_arc) ARCS = {<1,2>,<5,4>,<2,8>};
Float Cost[ARCS] = [ 5, 7, 10];
dvar xflow[ARCS];
... e.g., among constraints ...
forall(arc in ARCS : cost[arc]>6) xflow[arc]<=2.0;
```

see `transport_tuple.mod`

OPL «advanced» topics: scripting

- OPL provides a full "programming" language
- Instructions are embedded in a statement
  `execute { <instructions> }`
- May be used for pre-processing (e.g., to prepare data) or
- post-processing (e.g., to display solutions) or
- … much more! (e.g., a complex algorithm that iteratively solves the defined MILP models with modified data)

```
execute{
  writeln( "MAIN FLOWS:");
  for ( arc in ARCS ) {
    if ( xflow[arc] > 0 ) {
      writeln("from ",arc.node_from," to ",arc.node_to,
                                       " flows ",xflow[arc]);  }
  }
}
```

Direct access to model's elements

see `transport_tuple.mod`

DOcplex – a Python interface to Cplex

- Built upon the Cplex Python API
- Exploits Python syntax to provide "easy" and flexible encoding of the mathematical model notation, e.g.:
  - Dictionaries for sets of variables
  - **`for`**…**`in`**…**`if`**… to encode "forall" quantifiers or sum indices
- If interested, find instructions through
  - DOcplex landing page: http://ibmdecisionoptimization.github.io/docplex-doc/
    - *Getting started with DOcplex*
    - *Math. Progr. Modeling for Python using docplex.mp*
  - Modeling examples on **moodle** (Lab Section) DOcplex sample models

Lab organization: OPL* or Cplex API?

Are you a student from the Master Degree in Computer Science **and** can you code in C or C++?

- **YES**: you will learn how to build models using the Cplex-API libraries[(1)] (to be used for the "lab exercise-part I"), **STOP**.
- **NO**: do you know C or C++ programming language?
  - **NO**: you will continue implementing models with OPL*[(2)] (to be used for the "lab exercise-part I"), **STOP**.
  - **YES**: you can choose if learning the Cplex-API[(1)] or implementing models with OPL*[(2)] (you can choose if to use the Cplex-API or OPL* for the "lab exercise-part I"). **STOP**.

(1) You are a *Cplex guy* **(2)** You are an *OPL guy*

* or any **agreed** alternative, e.g., DOcplex or Matlab connector etc.

Cplex Callable Libraries

- C API towards *LP/QP/MIP/MIQP* algorithms
- Basic objects: **`Environment`** and **`Problem`**
- **`Environment`**: license, optimization parameters …
- **`Problem`**: contains problem information: variables, constraints …)
- (at least one) environment and problem must be created

```
CPXENVptr CPXopenCPLEX / CPXcloseCPLEX

CPXLPptr  CPXcreateprob / CPXfreeprob
```

Cplex API functions

- The two objects can be accessed (e.g. to add variables or constraints, or to solve a problem) via the functions provided by the API
- (Almost) all the API functions can be called as

```
int CPXfuncName (environment[,problem],...);
```

Error code (0=ok)
CPXgeterrorstring returns a description of the error

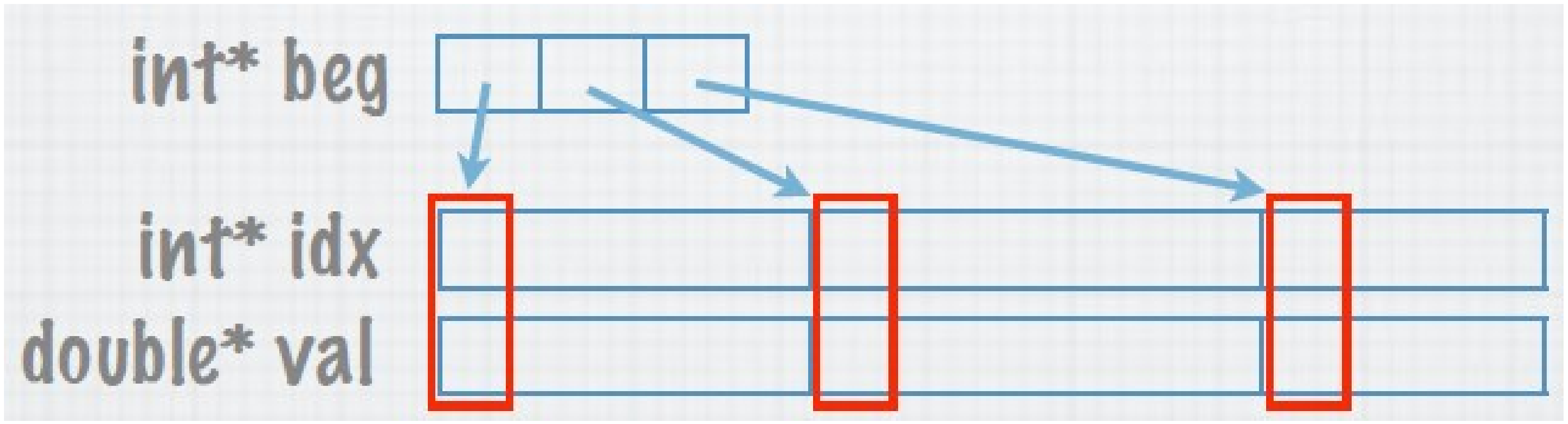
Basic objects

Parameters

cpxmacro.h

Sparse matrix representation

- Sparse matrix: many zero entries
- Compact representation:
  - Explicit representation of “nonzeroes”
  - Linearization into indexes (**idx**) and values (**val**) vectors
  - A third vector to indicate where rows begins (**beg**)



addrow.xls