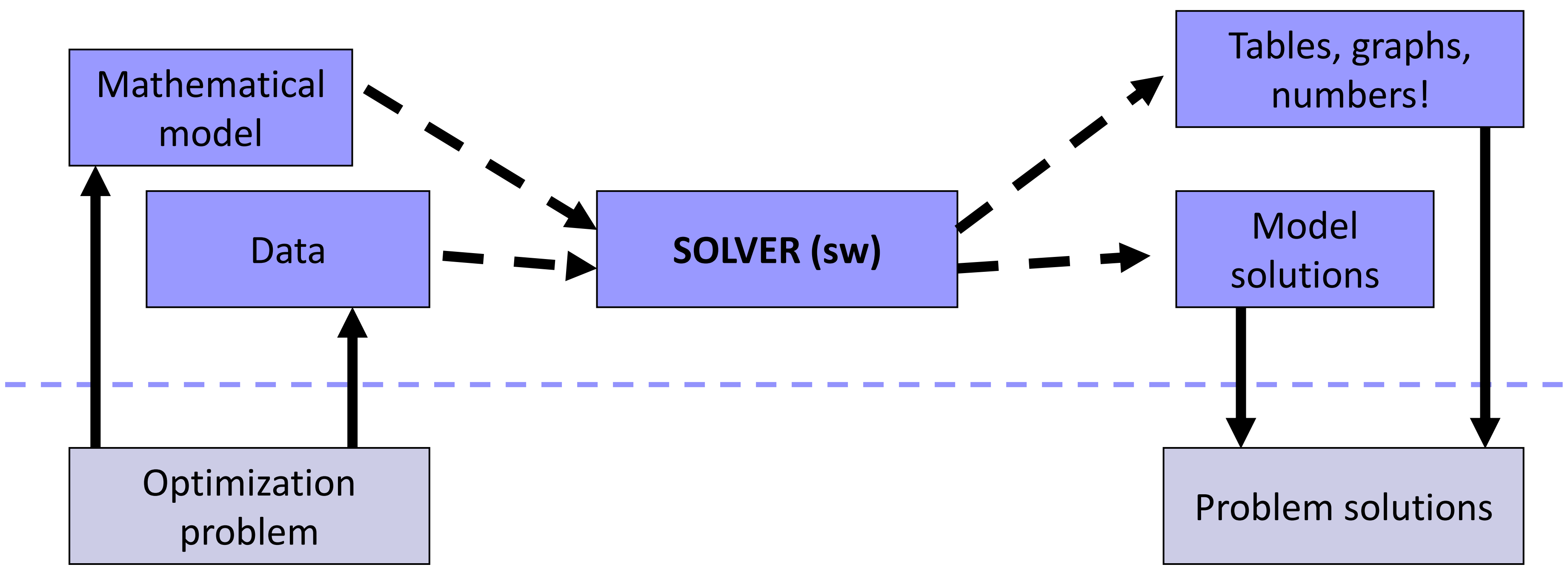
Solvers for Mathematical Programming

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.

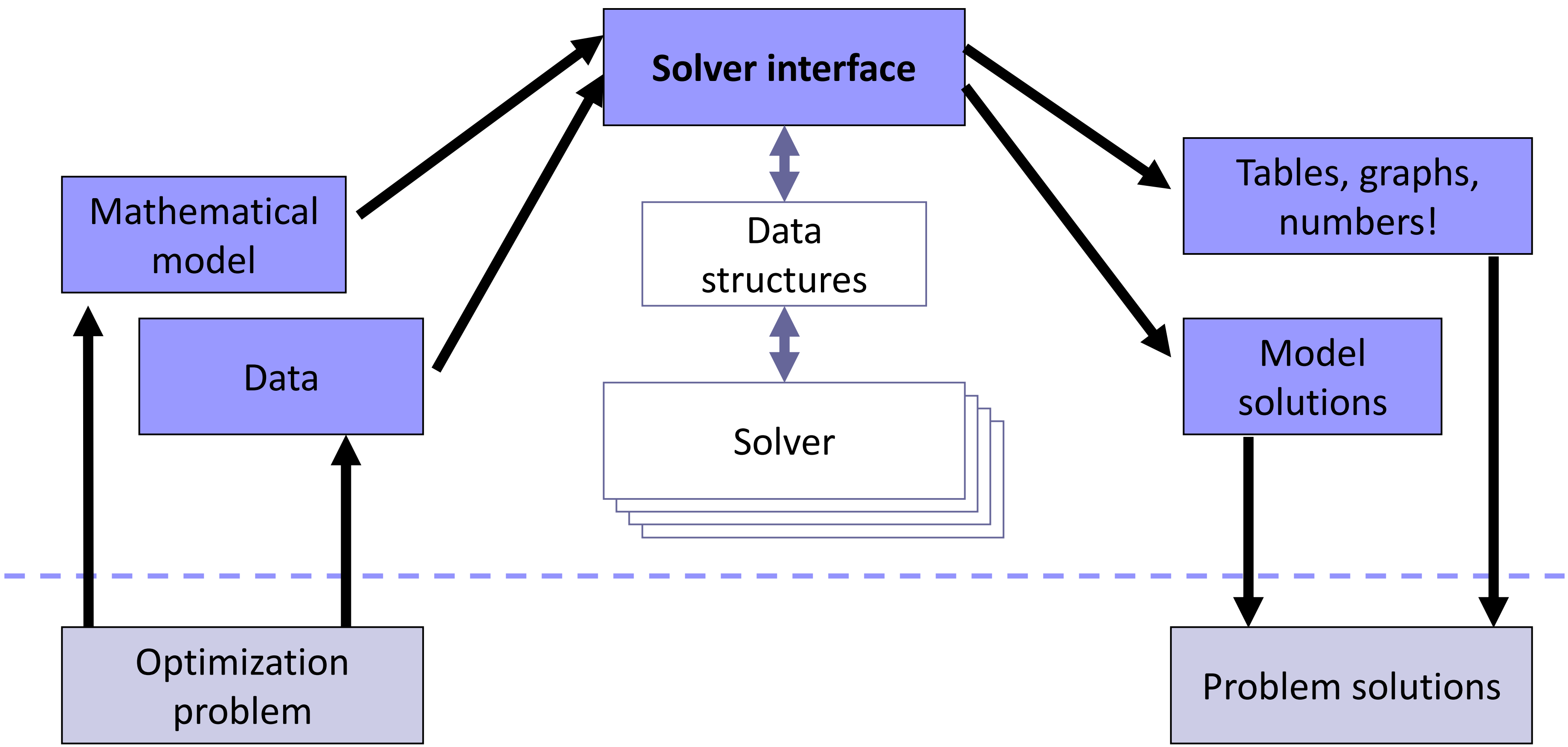


MILP solvers

- **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
- e.g. Cplex, Gurobi, Xpress, Scip, Lindo, GLPK 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** / 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.)

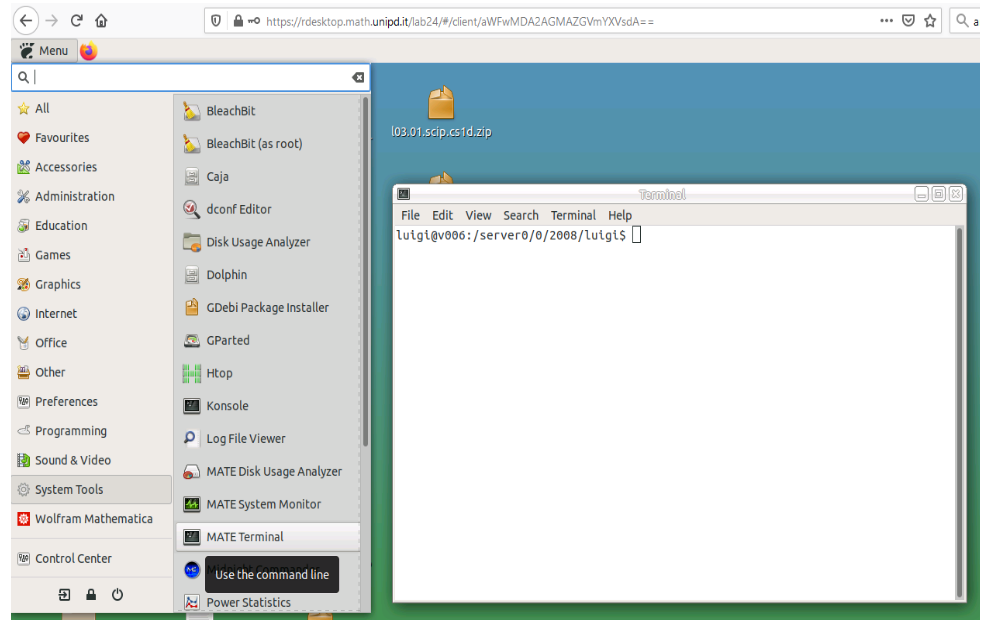
- Integrated Development Environment (IDE) available

- Included in the Cplex Studio package

- Learning OPL by examples

Basic commands (in Lab)

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



- 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>;
        constraint2_name: <expression>;
        ...
}
```

`<expression>` = e.g.

```
sum( i in setI, j in setJ )
        <expression using indexes i and j>
```

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];
```

Separating model and data

[4.mix_general_dataout] **1/3**

- **.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] **2/3**

- (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]` **3/3**

- `.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, ...],
      ...
];
```

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]`
  - 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

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*

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

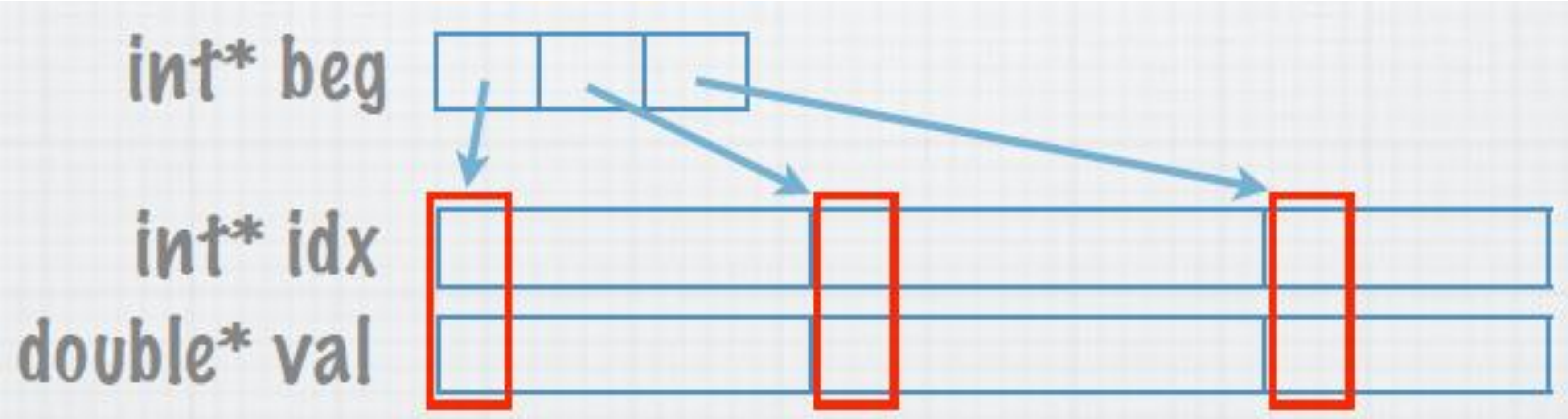
Basic objects

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