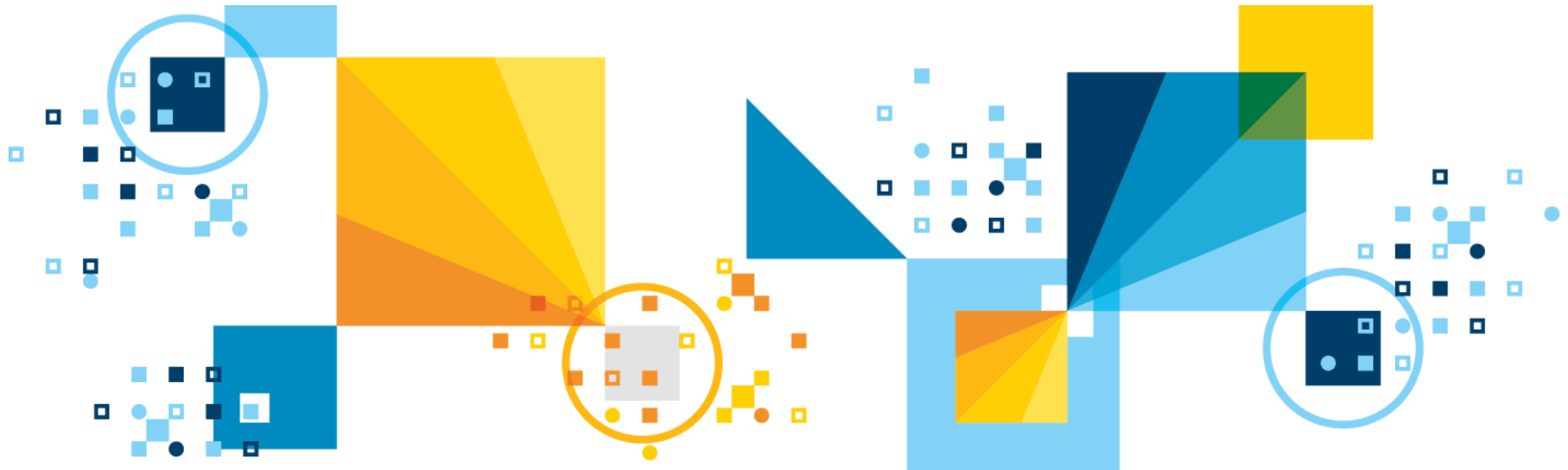
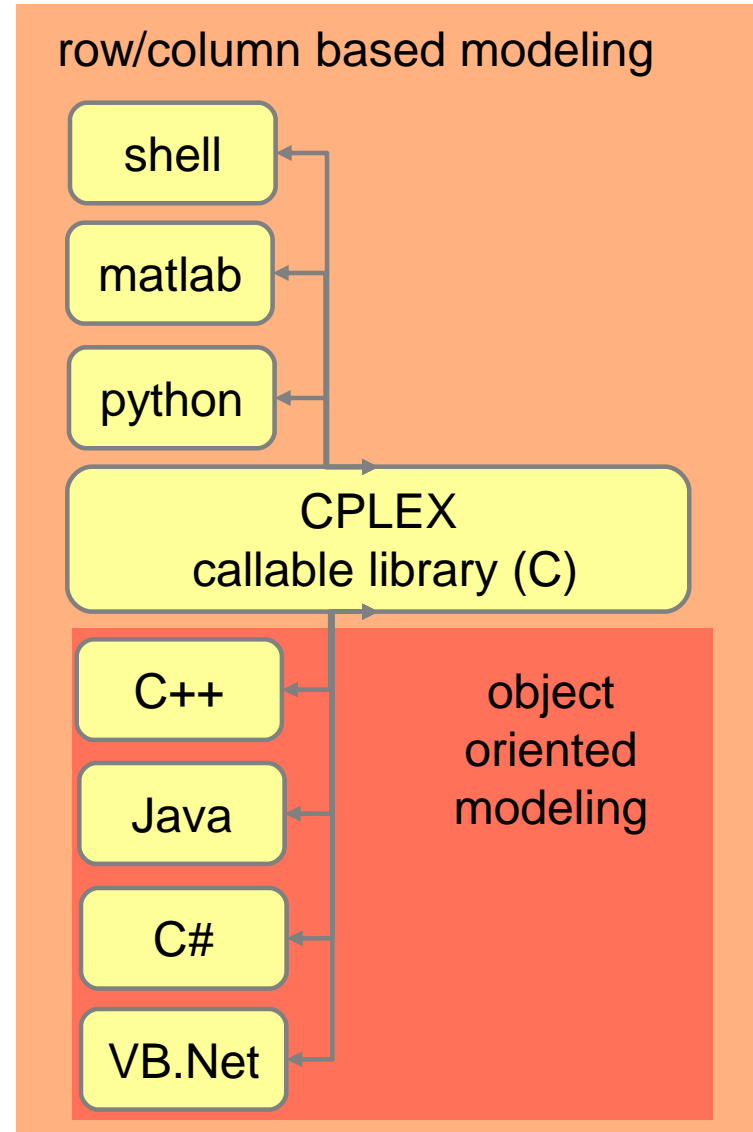
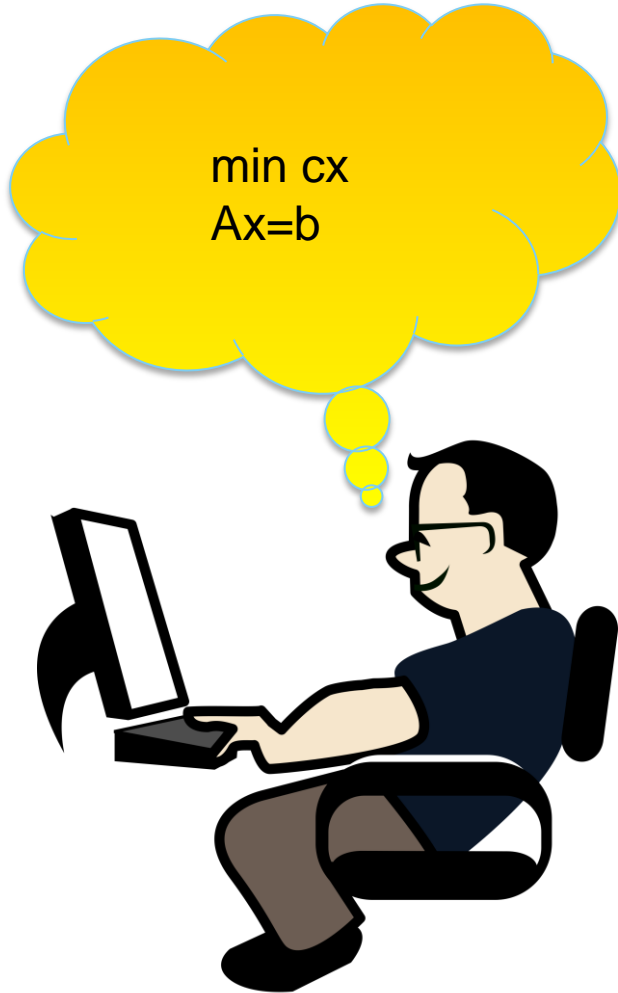


CPLEX APIs

@CPLEX School 2017 Montreal



Interfacing with CPLEX: APIs



Interfacing with CPLEX: modeling languages



Optimization Programming Language (OPL)

- write models in a more descriptive form
- write models in a more compact form
- faster prototyping, easier maintenance
- easier access to data (Excel, database, ...)

```
forall (i in I)
  sum (j in J)
    sum (k in K)
      x[i][j][k] <= 1;
```

- scriptable
- model editor
- IDE support (eclipse based)

OPL Projects | Debug | ECP (Equitable Coach Problem for CO@Work 2015) | Run Configurations | Run (default) | ECP.mod : CPLEX

Problem browser | Variables | Breakpoints

Solution with objective 17.333

Name	Value
Data (11)	
avg	11.667
D	1
Formation	[1 2 3 1]
G	0
game_length	20
M	2
nplayers	7
Periods	1..20
Players	{<"Josh" {3 2}> <"Simon" {3}> <"J
Positions	0..3
S	3
Decision variables (3)	
eminus	[02.6667 0.66667 0.66667 0.666
eplus	[0.33333 0 0 0 0 0 0 0 0 8.3333
x	[[[0 0 1 0] [0 0 0 1] [0 0 0 0] [0 1

```

1 range Periods = 1..20; // Periods that make up the game.
2 int nplayers = 7; // Number of players on field
3
4 range Positions = 0..3; // The different positions
5 int G = 0; int D = 1;
6 int M = 2; int S = 3;
7
8 int Formation[Positions] = [ 1, 2, 3, 1 ]; // The formation for
9 // which we optimize
10
11 tuple Player { // Players we need to schedule
12     string name;
13     {int} play;
14 };
15 {Player} Players = {
16     <"Josh", {S, M}>, <"Simon", {S}>, <"Jordy", {S}>,
17     <"Chris", {M, D}>, <"Andy", {M}>, <"Richie", {M, D}>,
18     <"Tritto", {D}>, <"Guy", {D}>, <"Neil", {M}>,
19     <"Justin", {M}>, <"Steve", {D}>, <"Cory", {G}>
20 };
21 float avg = card(Periods) * nplayers / card(Players);
22
23 // Decision variables
24 dvar float+ eplus[Players];
25 dvar float+ eminus[Players];
26 dvar boolean x[Periods][player in Players][p in Positions] in 0..((p in player.play) ? 1 : 0);
27
28 minimize sum(p in Players) (eplus[p] + eminus[p]);
29
30 subject to {
31     // Each player can play at most one position
32     forall (player in Players)
33     forall (period in Periods)
34         sum(p in Positions) x[period][player][p] <= 1;
35
36     // Correct number of players for each position
37     forall (p in Positions) forall (period in Periods)
38         sum (player in Players : p in player.play)
39             x[period][player][p] == Formation[p];
40
41     // Calculates minutes above/below average
42     forall (player in Players) {
43         sum (p in player.play) sum (period in Periods) x[period][player][p] - avg == eplus[player] - eminus[player];
44     }
45 }
    
```

Problems | Scripting log | Solutions | Conflicts | Relaxations | Engine log | Statistics | Profiler | CPLEX Servers

0 items				
Description	Resource	Path	Location	Type

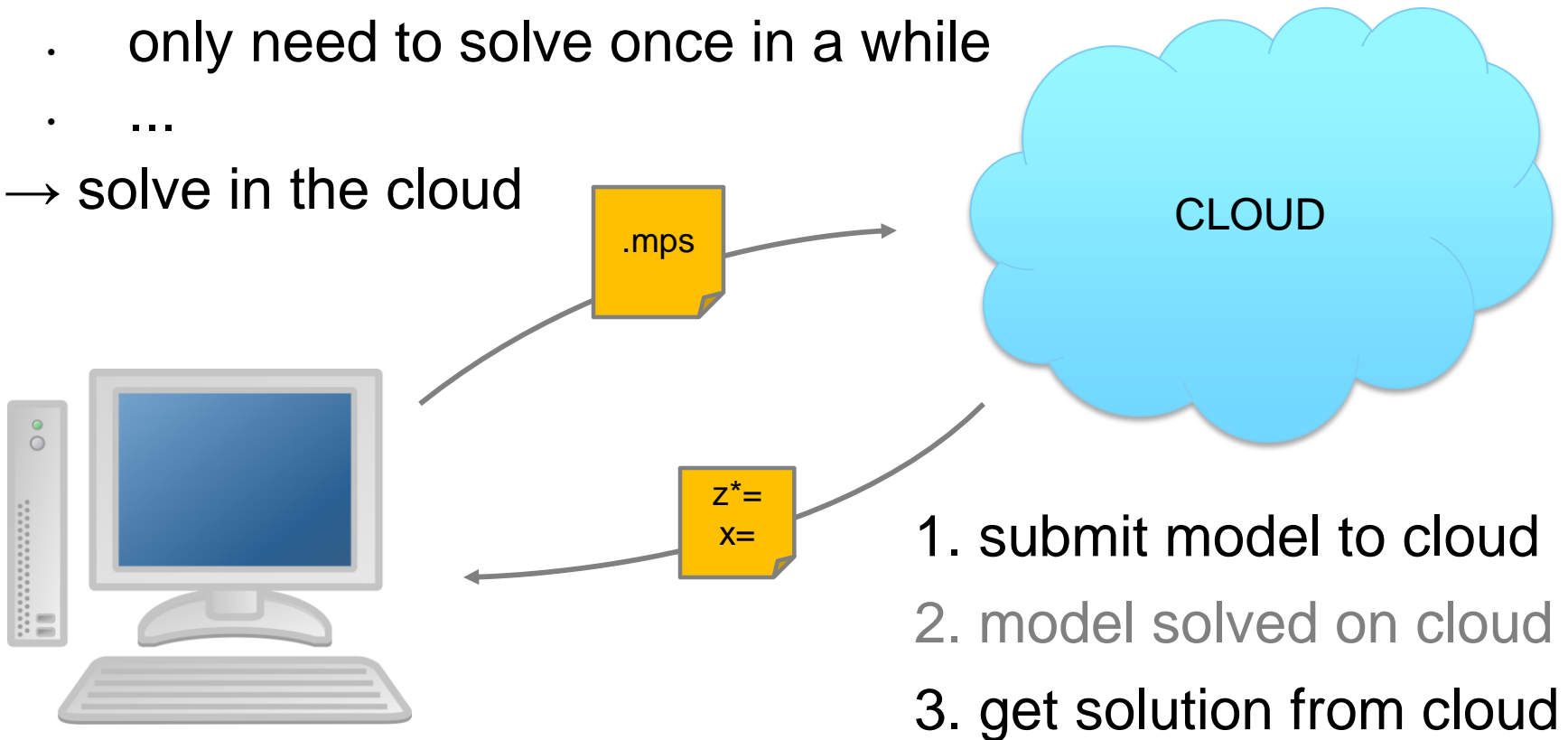
Writable | Insert | 39 : 45 - 1346 | 00:00:00:11

CPLEX in the CLOUD

What if no local resources to solve model?

- model too hard
- only need to solve once in a while
- ...

→ solve in the cloud



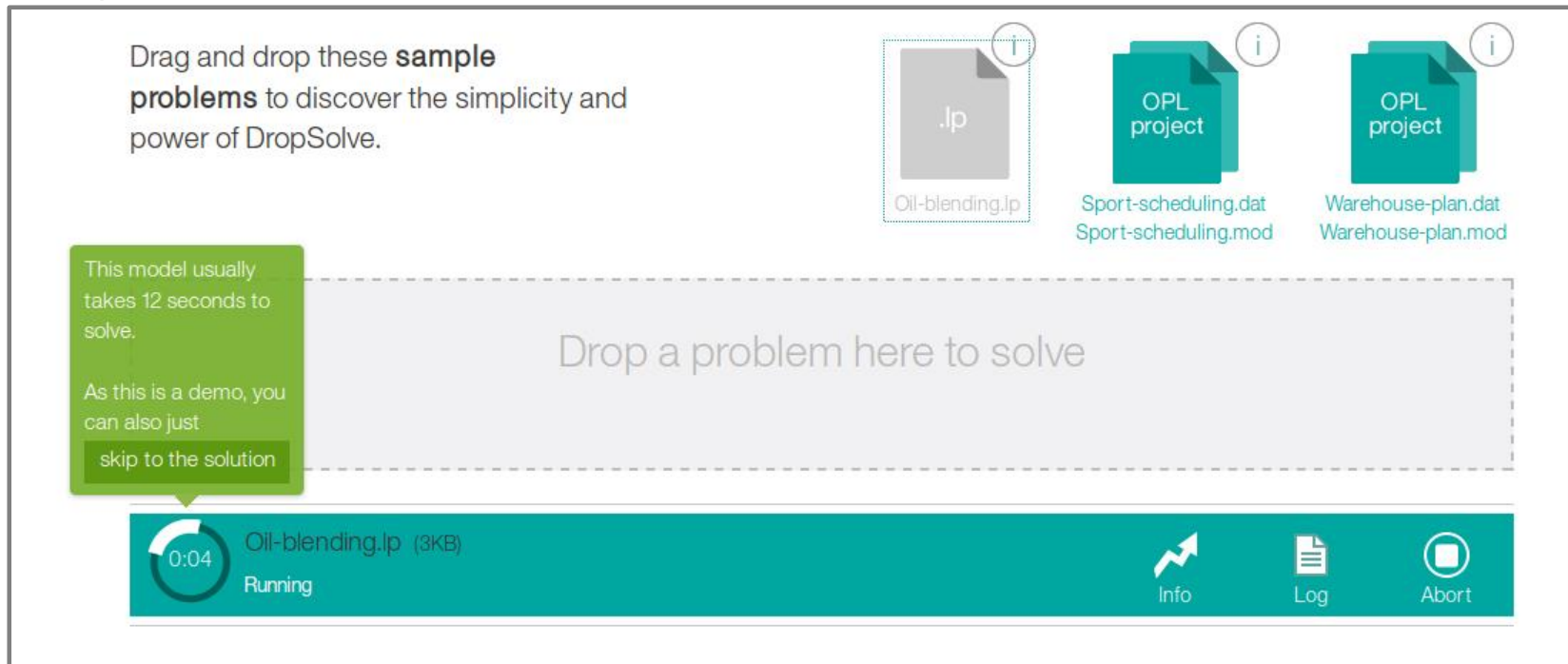
CPLEX in the CLOUD

Two ways to access CPLEX in the cloud

1. DropSolve

<https://dropsolve-oaas.doccloud.ibmcloud.com/software/analytics/docloud>

Drag and drop your model file from disk to web browser



Drag and drop these **sample problems** to discover the simplicity and power of DropSolve.

This model usually takes 12 seconds to solve.
As this is a demo, you can also just skip to the solution

Drop a problem here to solve

Oil-blending.lp (3KB)
Running

Info Log Abort

The screenshot shows a web interface for DropSolve. At the top, there is a text prompt: "Drag and drop these sample problems to discover the simplicity and power of DropSolve." Below this, there are three sample problem icons: a .lp file named "Oil-blending.lp", and two OPL project files named "Sport-scheduling.dat" and "Warehouse-plan.dat". A large dashed box in the center contains the text "Drop a problem here to solve". A green callout box on the left provides information about the "Oil-blending.lp" model, stating it usually takes 12 seconds to solve and that users can skip to the solution in a demo. At the bottom, a teal task bar shows the "Oil-blending.lp (3KB)" file is "Running" with a progress indicator of 0:04. To the right of the task bar are icons for "Info", "Log", and "Abort".

CPLEX in the CLOUD

Two ways to access CPLEX in the cloud

2. REST API

<https://developer.ibm.com/doccloud/>

<https://developer.ibm.com/doccloud/docs/welcome/>

Access the solve service via its REST API

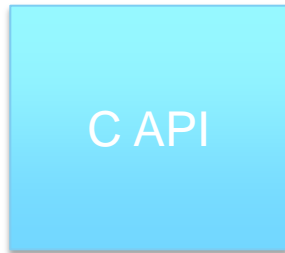
- ready-to-use clients provided for Java and Python, e.g.

```
JobExecutor executor = JobExecutorFactory.createDefault();
JobClient jobclient = JobClientFactory.createDefault(BASE_URL,
                                                    APIKEY_CLIENTID);
jobclient.newRequest().input(new File("model.mps"))
            .output(new File("x.sol"))
            .execute(executor).get();
```

- With any HTTP client (cURL, ...)

So what should you use?

Ease of use



Flexibility

- Hard to tell in general
- Depends on use case
- But...

docplex <https://pypi.python.org/pypi/docplex>

- Pure Python modeling API
- Opensource (pypi, github)
- Support local and cloud solves
- Why Python?
 - You don't need to learn yet another DSL
 - Python provides enough *syntactic sugar* for modeling
 - Get direct access to the whole Python software ecosystem
(quite strong in the scientific community: numpy, scipy, iPython/Jupyter notebooks, etc...)

Callable library: CPLEX C API

- Used for tight/efficient integration into applications
- C libraries are broadly callable also from higher level languages (no need to stick to C for using the C API!)
- Most comprehensive CPLEX API:
if it is not in the C API, it doesn't exist.
- Also the harder to use (C itself is rather low-level)



Callable library 101

Based on two opaque objects:

- **environment**: parameters/callbacks/license checking
- **problem**: problem data (variables, constraints, objective...)

CPX(C)ENVptr

CPXopenCPLEX/CPXcloseCPLEX

CPX(C)LPptr

CPXcreateprob/CPXfreeprob

Callable library 101

- The two objects can be inspected/modified by using appropriate functions
- Almost all functions follow the same pattern:

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



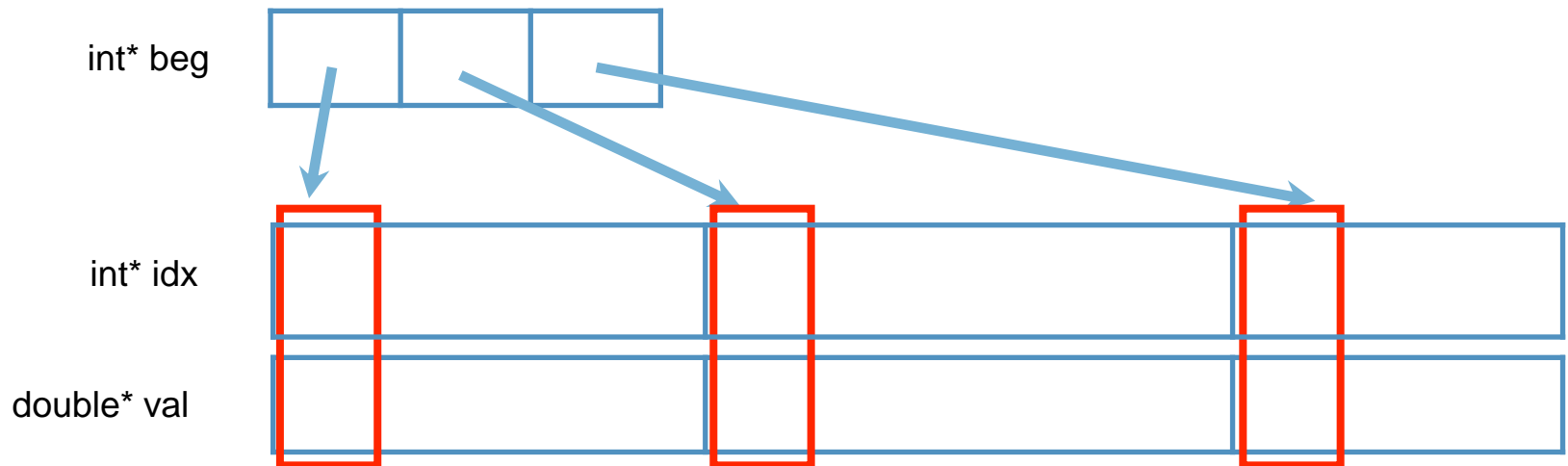
error code (0=ok)

opaque objects

parameters

Callable library 101

- As with many C libraries, only basic data types are supported:
 - Scalars (by value and/or pointer): double/int
 - Arrays: double*/int*
 - Strings: char*
- For passing around matrices, a sparse data structure is used:



Help/Documentation

IBM Knowledge Center: online documentation

<https://www.ibm.com/support/knowledgecenter/SSSA5P>

IBM developerWorks: community forum

<https://www.ibm.com/developerworks>

Legal Disclaimer

- © IBM Corporation 2017. All Rights Reserved.
- The information contained in this publication is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this publication, it is provided AS IS without warranty of any kind, express or implied. In addition, this information is based on IBM's current product plans and strategy, which are subject to change by IBM without notice. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this publication or any other materials. Nothing contained in this publication is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.
- References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in this presentation may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.
- Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.
- Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.
- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Other company, product, or service names may be trademarks or service marks of others.

IBM®