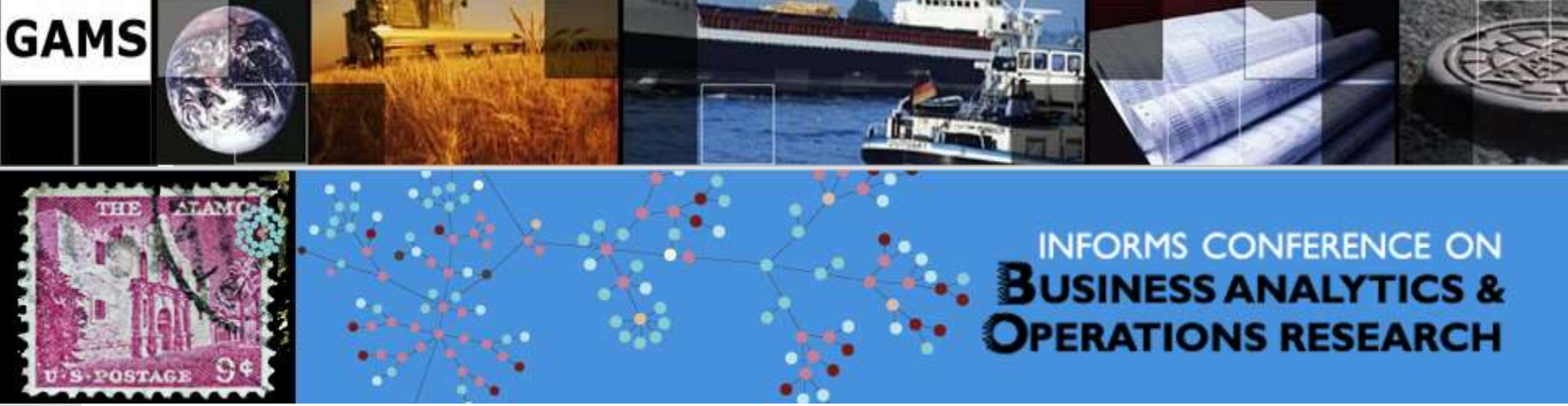


Deploying Your Application Built Around GAMS

Franz Nelissen FNelissen@gams.com

Lutz Westermann LWestermann@gams.com

GAMS Development Corp.

GAMS Software GmbH

www.gams.com

San Antonio, TX, April 2013

GAMS

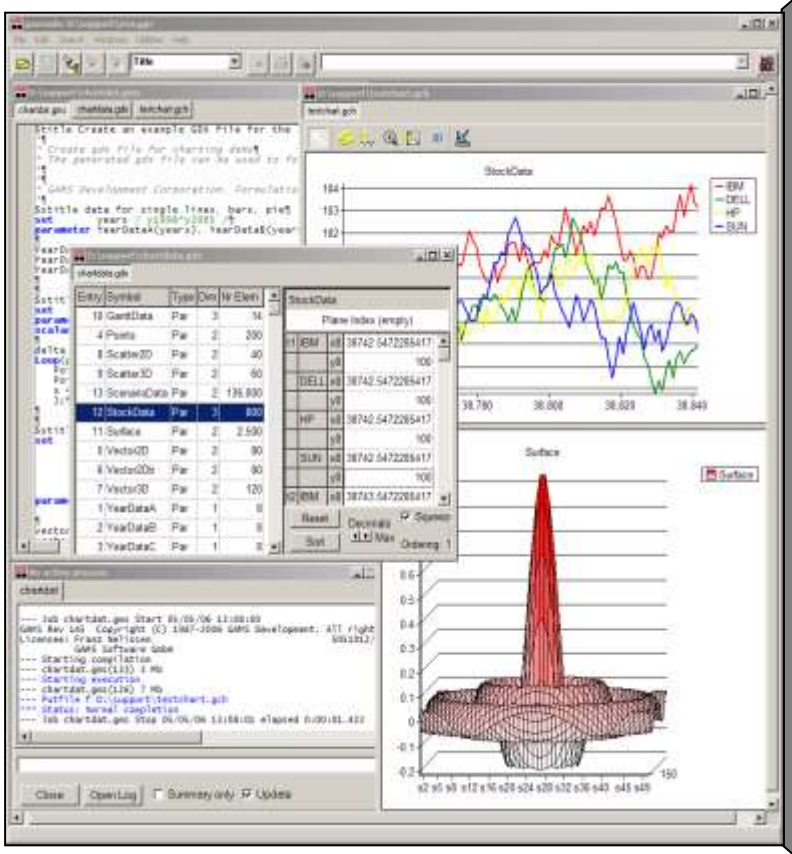
Company Background

- Roots: World Bank, 1976
- Went commercial in 1987
- GAMS Development Corporation (Washington, Houston)
- GAMS Software GmbH (Frechen, Braunschweig)
- Tool Provider: **G**eneral **A**lgebraic **M**odeling **S**ystem

Agenda

- What is GAMS?
- Building an Application around GAMS
- What is New?

GAMS at a Glance



- Algebraic Modeling Language with balanced mix of declarative and procedural elements
- Platform independent: 10+ Platforms
- Hassle-free switch of solution methods and solvers:
  - 10+ MP classes
  - 30+ Solvers
- Open architecture and interfaces to other systems

GAMS Design Principles

Independence of

- Model and operating system
- Model and solution methods (solver)
- Model and modeling system
- Model and data
- Model and user interface

Models benefit from

- Advancing hardware
- Enhanced / new solver technology
- Improved / upcoming interfaces to other systems

Agenda

- **Building an Application around GAMS**
- The Cutting Stock Problem
- Object Oriented GAMS API
- Some Examples (c#)

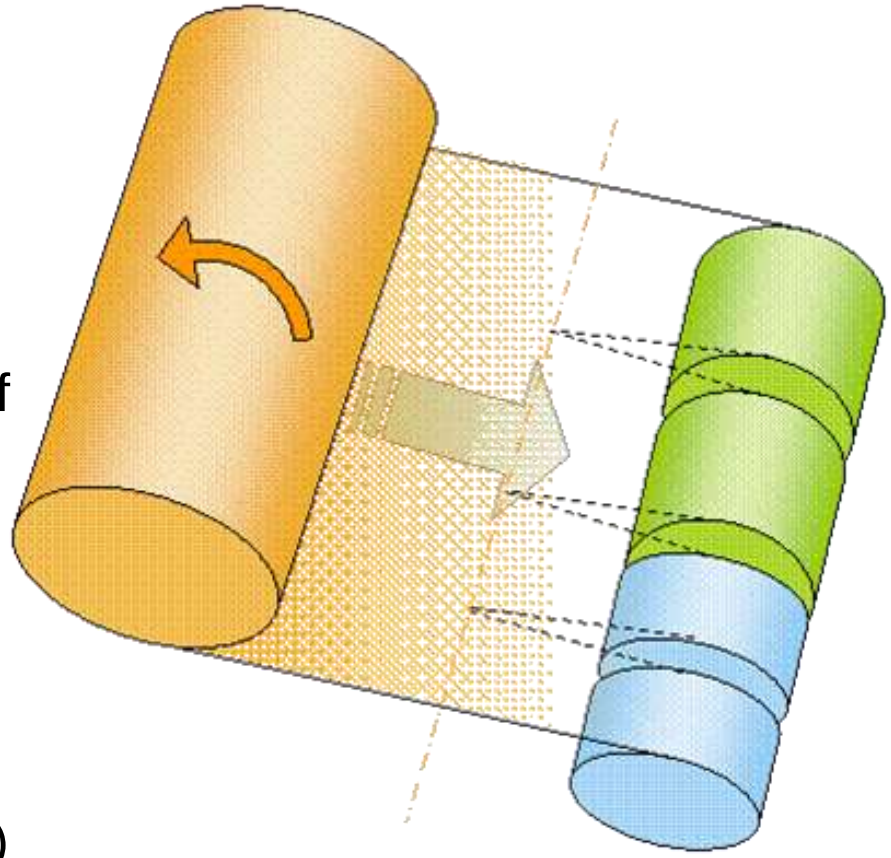
Cutting Stock Problem

- Problem: Paper rolls of fixed width (“raw”) must be cut into smaller portions (“finals”).

- Input:
  - Width of the raw
  - Demand: Widths and number of finals

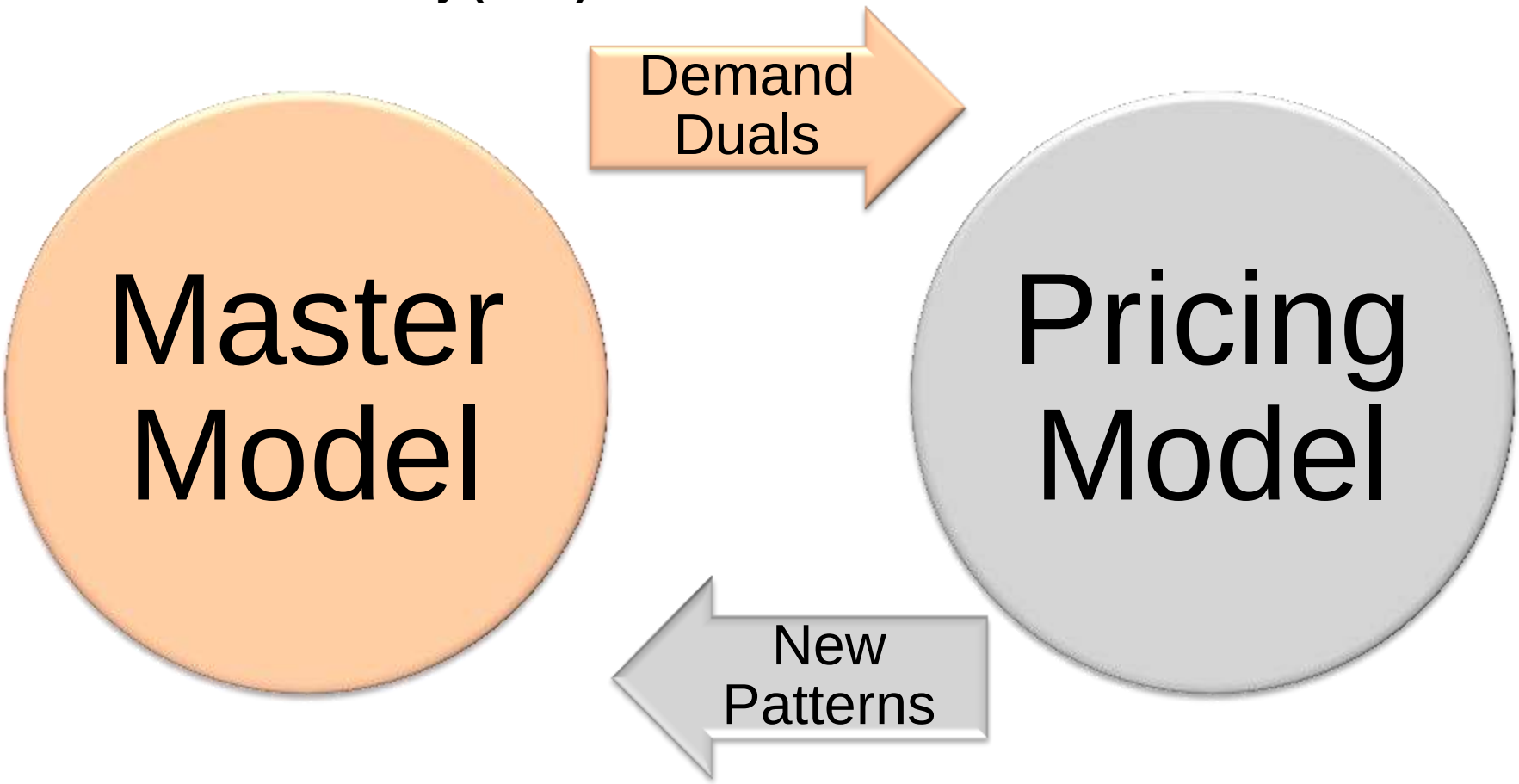
- Objective: Minimize the required number of raws

- Output:
  - Combination of cuts (“patterns”)
  - Produced number of each patterns
  - Number of required raws



Cutting Stock Problem: Column Generation

Gilmore & Gomory (1961)



Cutting Stock Problem: Modeler

cutstock.gms | cutstock.lst

```
Set  i    widths    /w1*w4/
Parameter
     r    raw width /100/
     w(i) width     /w1  45
                     w2  36
                     w3  31
                     w4  14/
     d(i) demand    /w1  97
                     w2 610
                     w3 395
                     w4 211/

* Gilmore-Gomory column generation algorithm

Set  p         possible patterns  /p1*p1000/
     pp(p)     dynamic subset of p
Parameter
     aip(i,p) number of width i in pattern growing in p;


* Master model
Variable xp(p)      patterns used
         z          objective variable
Integer variable xp; xp.up(p) = sum(i, d(i));

Equation numpat     number of patterns used
         demand(i) meet demand;

numpat..      z =e= sum(pp, xp(pp));
demand(i)..   sum(pp, aip(i,pp)*xp(pp)) =g= d(i);

model master /numpat, demand/;

* Pricing problem - Knapsack model
Variable  y(i) new pattern;
Integer variable y; y.up(i) = ceil(r/w(i));

Equation defobj
         knapsack knapsack constraint;

defobj..      z =e= 1 - sum(i, demand.m(i)*y(i));
knapsack..    sum(i, w(i)*y(i)) =l= r;

model pricing /defobj, knapsack/;
```

cutstock.gms | cutstock.lst

```
* Initialization - the initial patterns have a single width
pp(p) = ord(p)<=card(i);
aip(i,pp(p))$(ord(i)=ord(p)) = floor(r/w(i));
*display aip;

Scalar done  loop indicator /0/
Set    pi(p) set of the last pattern; pi(p) = ord(p)=card(pp)+1;

option optcr=0,limrow=0,limcol=0,solprint=off;

While(not done and card(pp)<card(p),
   solve master using rmip minimizing z;
   solve pricing using mip minimizing z;

* pattern that might improve the master model fou
   if(z.l < -0.001,
      aip(i,pi) = round(y.l(i));
      pp(pi) = yes; pi(p) = pi(p-1);
   else
      done = 1;
   );
);
display 'lower bound for number of rolls', master.objval;

option solprint=on;
solve master using mip minimizing z;

Parameter patrep Solution pattern report
          demrep Solution demand supply report;

patrep('# produced',p) = round(xp.l(p));
patrep(i,p)$patrep('# produced',p) = aip(i,p);
patrep(i,'total') = sum(p, patrep(i,p));
patrep('# produced','total') = sum(p, patrep('# produced',p));

demrep(i,'produced') = sum(p,patrep(i,p)*patrep('# produced',p));
demrep(i,'demand') = d(i);
demrep(i,'over') = demrep(i,'produced') - demrep(i,'demand');

display patrep, demrep;
```

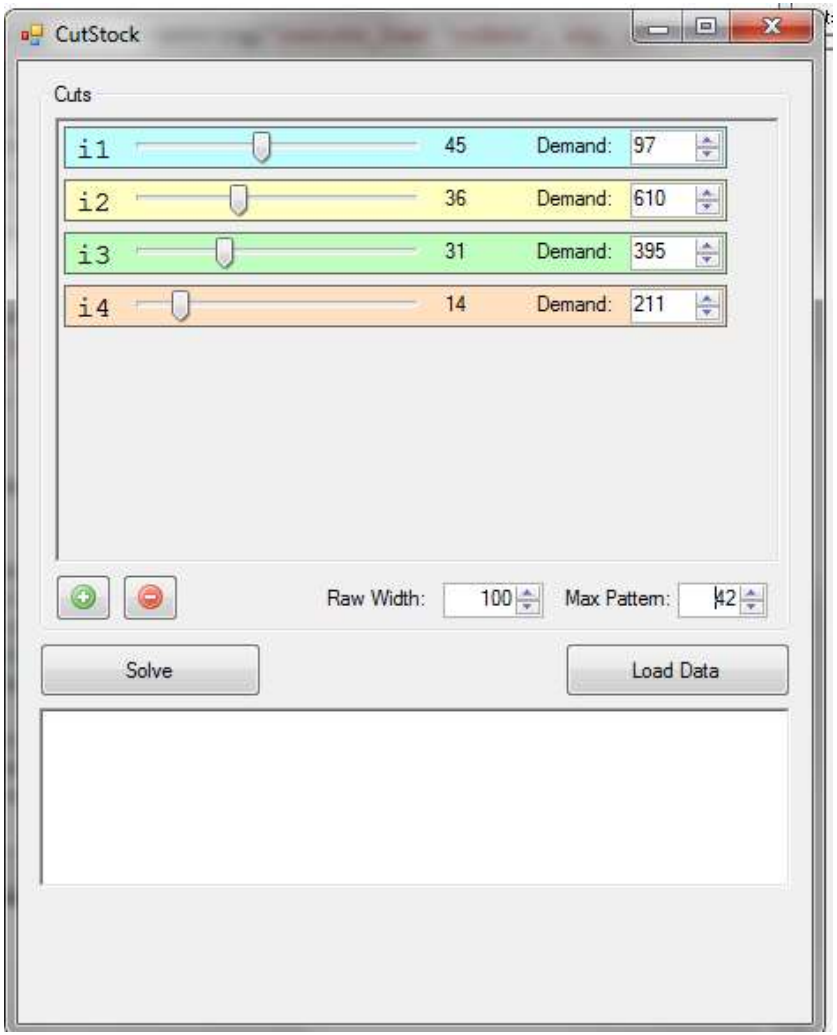
patrep(*, *): Solution pattern report

	p1	p2	p5	p6	total
w1	2				2
w2		2	2	1	5
w3				2	2
w4			2		2
# produced	49	100	106	198	453

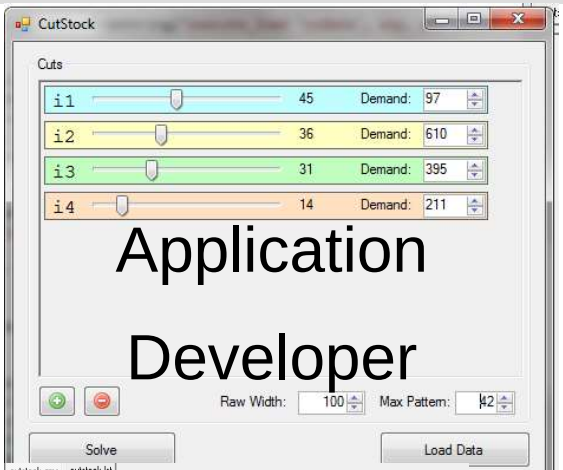
demrep(*, *): Solution demand supply report

	produced	demand	over
w1	98	97	1
w2	610	610	
w3	396	395	1
w4	212	211	1

Cutting Stock: Application Developer



Bridging The Gap: GAMS OO API



Application Developer

Modeler

```
* Initialization - the initial patterns have a single width
pp(p) = ord(p)<=card(i);
aip(i,pp(p))$(ord(i)=ord(p)) = floor(r/w(i));
*display aip;

Scalar done  loop indicator /0/
Set    pi(p) set of the last pattern; pi(p) = ord(p)=card(pp)+1;

option optcr=0,limrow=0,limcol=0,solprint=off;

While(not done and card(pp)<card(p),
   solve master using rmip minimizing z;
   solve pricing using mip minimizing z;

* pattern that might improve the master model found?
   if(z.l < -0.001,
      aip(i,pi) = round(w.l(i));
      pp(pi) = yes; pi(p) = yes;
   else
      done = 1;
   );
);
display 'lower bound for number of rolls', master.objval;

option solprint=on;
solve master using mip minimizing z;

Parameter patrep Solution pattern report
          demrep Solution demand supply report;

patrep('# produced',p) = round(xp.l(p));
patrep(i,p)$patrep('# produced',p) = aip(i,p);
patrep(i,'total') = sum(p, patrep(i,p));
patrep('# produced','total') = sum(p, patrep('# produced',p));

demrep(i,'produced') = sum(p,patrep(i,p)*patrep('# produced',p));
demrep(i,'demand') = d(i);
```

patrep(*, *): Solution pattern report

	p1	p2	p5	p6	total
w1	2				2
w2		2	2	1	5
w3				2	2
w4			2		2
# produced	49	100	106	198	453

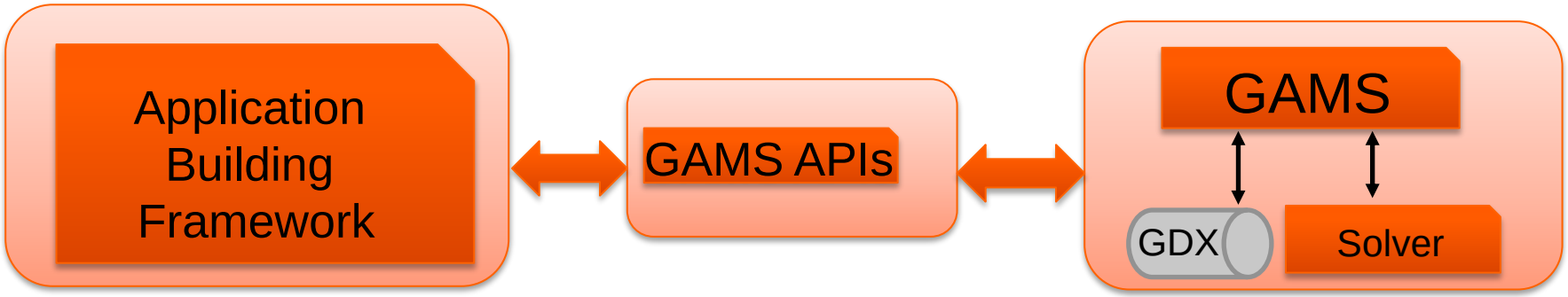
demrep(*, *): Solution demand supply report

	produced	demand	over
w1	98	97	1
w2	610	610	
w3	396	395	1
w4	212	211	1

GAMS OO API

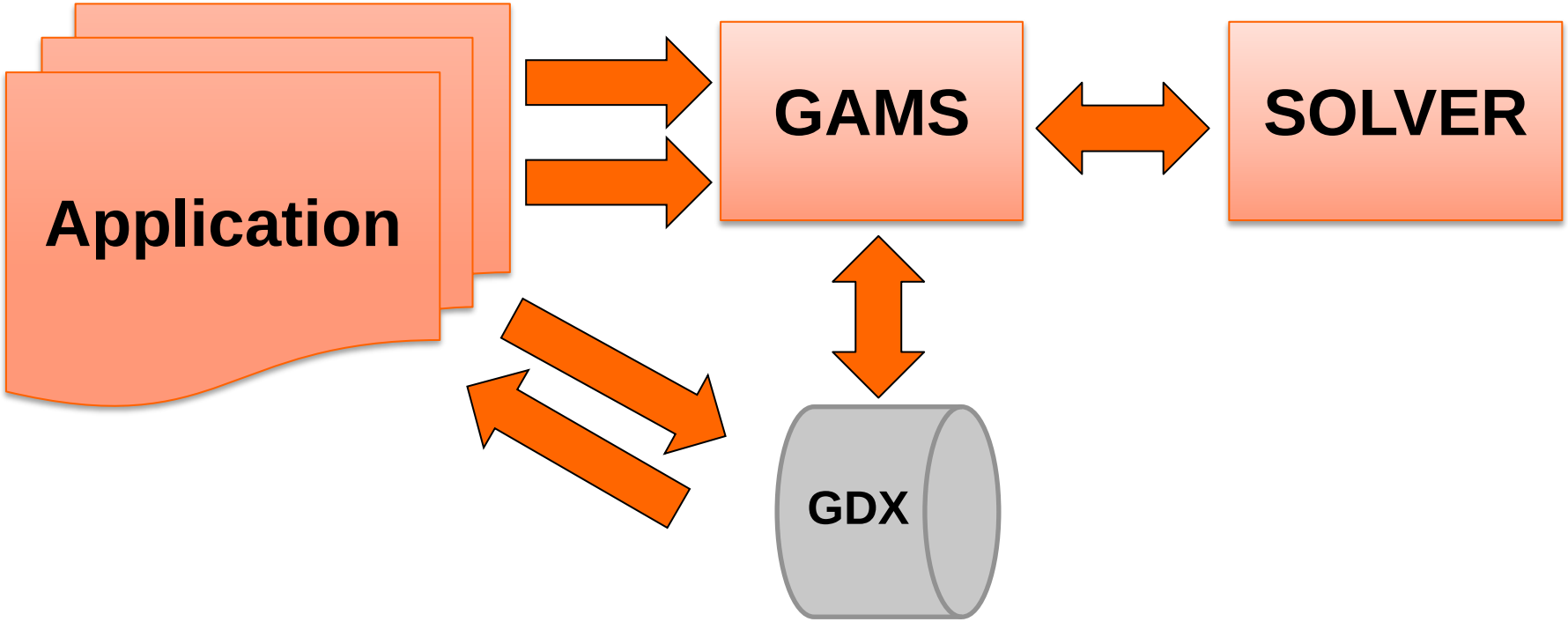
Concept: Separation of Tasks

- Use GAMS for modeling and optimization tasks
- Programming languages like C# (.NET), Java and Python are well-suited for developing applications (GUI, Web, …)
- Frameworks available for a wide range of specific tasks, e.g GUI and Web development
- Communication between GAMS and application through GAMS APIs



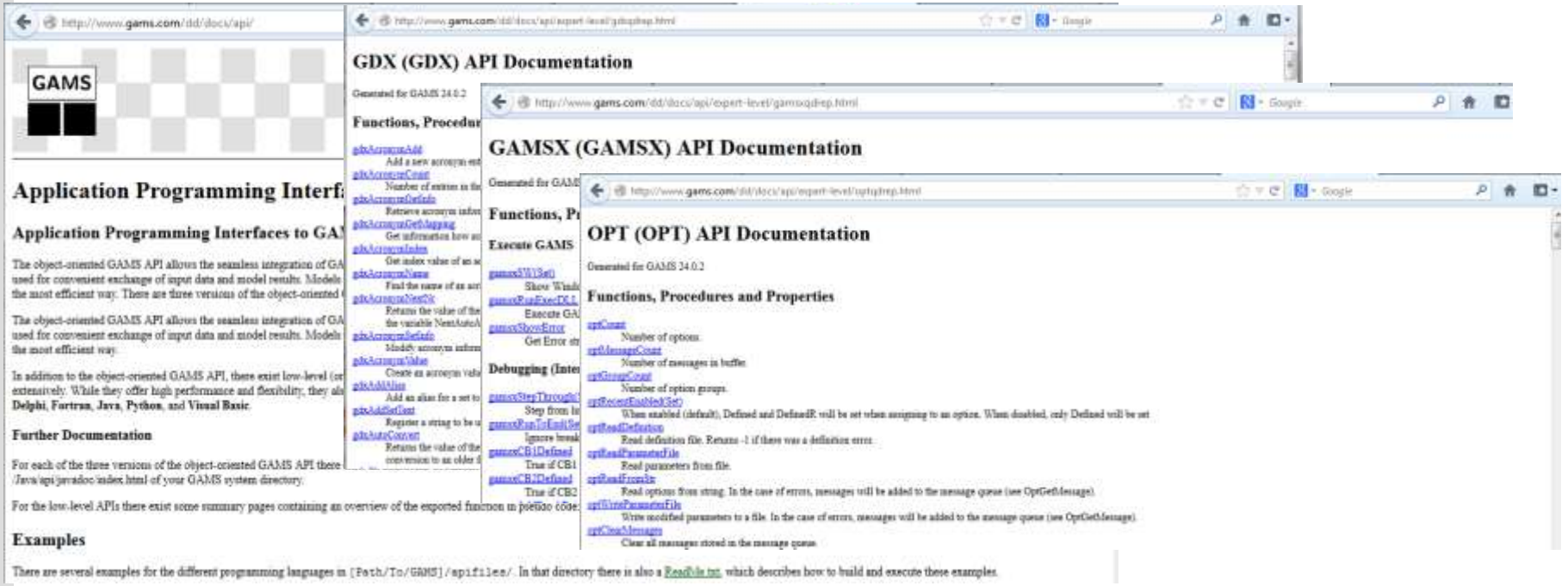
GAMS API: Basic Functionality

1. GDX API: Create Input for GAMS Model
2. Callout to GAMS
   - Option API: GAMS option settings
   - GAMSX API: Start and controls GAMS Job
3. GDX API: Get Solution from GAMS Model



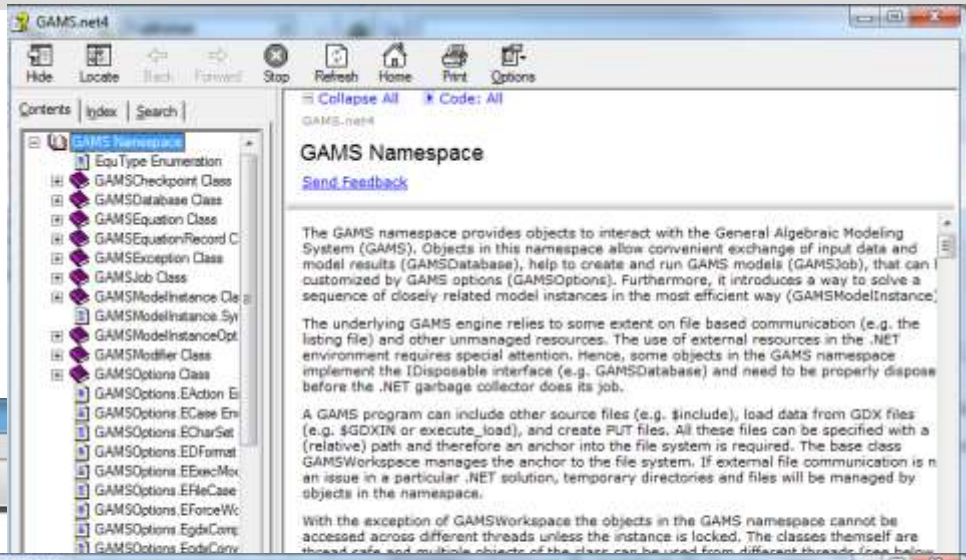
Low level GAMS API

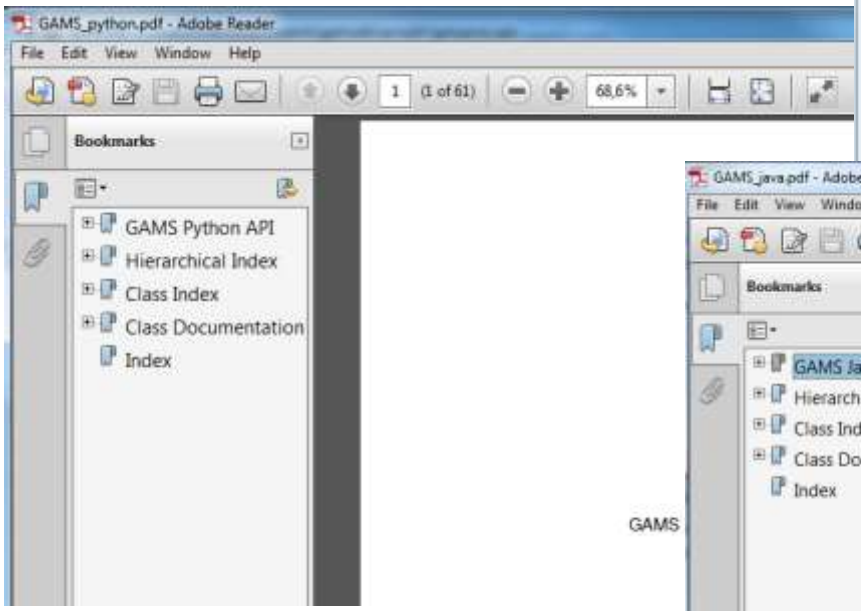
- High performance and flexibility
- Automatically generated APIs for several languages (C, Delphi, Java, Python, C#, …)
- Data Exchange (GDX), job control (GAMSX), options (OPT)
- Part of any GAMS distribution, no license required

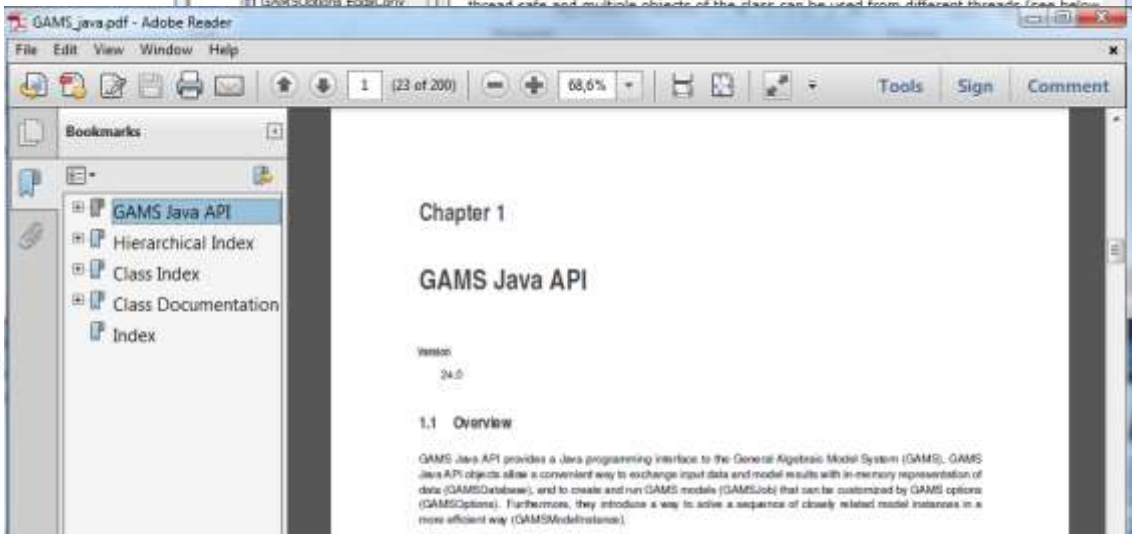


Object-oriented GAMS API

- Additional layer on top of the low level APIs
- Object-oriented: .NET, Java, and Python
- Part of any GAMS distribution, no license required





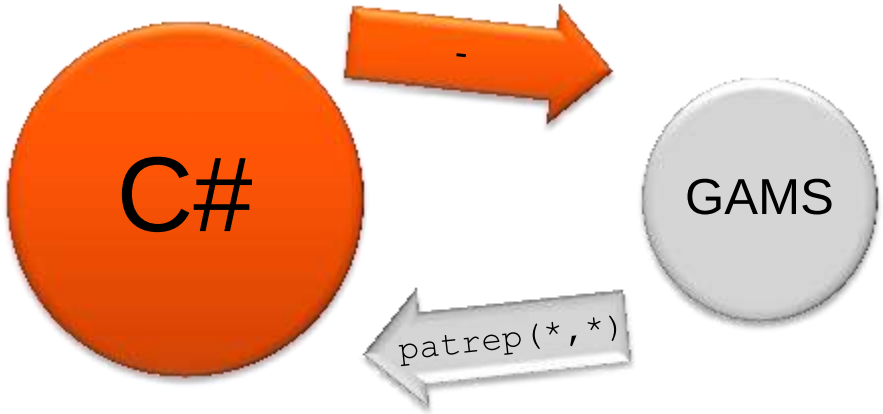


Features of the Object Oriented API

- No modeling capability, model is still written in GAMS
- Prepare input data and retrieve results in a convenient way → *GAMSDatabase class*
- Set GAMS and Solver Options → *GAMSOptions class*
- Control GAMS execution → *GAMSJob class*
- Scenario Solving: Feature to solve multiple very similar models in a dynamic and efficient way →*GAMSModelInstance class*

Application Step 1

- What are we going to do?
  1. Run the model
  2. Print out the results
  3. Change the solver used
- Interface:

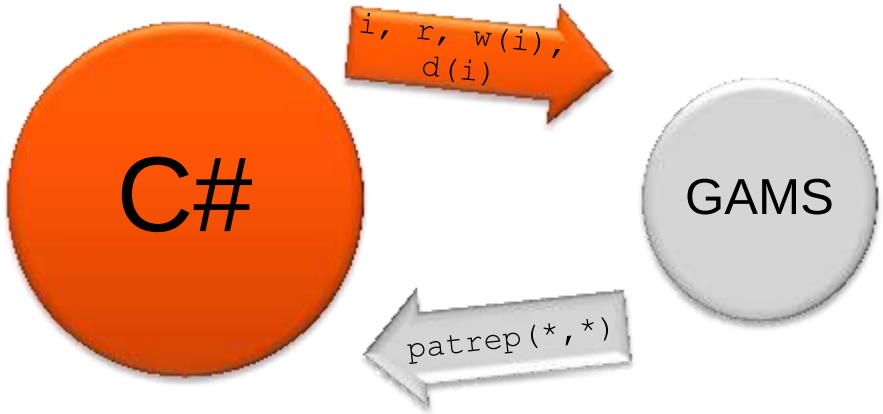


- What do we need?
  1. GAMSWorkspace & GAMSJob
  2. GAMSOptions (Solvers)
  3. GAMSParameter

➔ Step1.cs

Application Step 2

- What are we going to do?
  - Define the input data within the application
  - Store the data and pass it to the model
  - Check for Errors
- Interface:



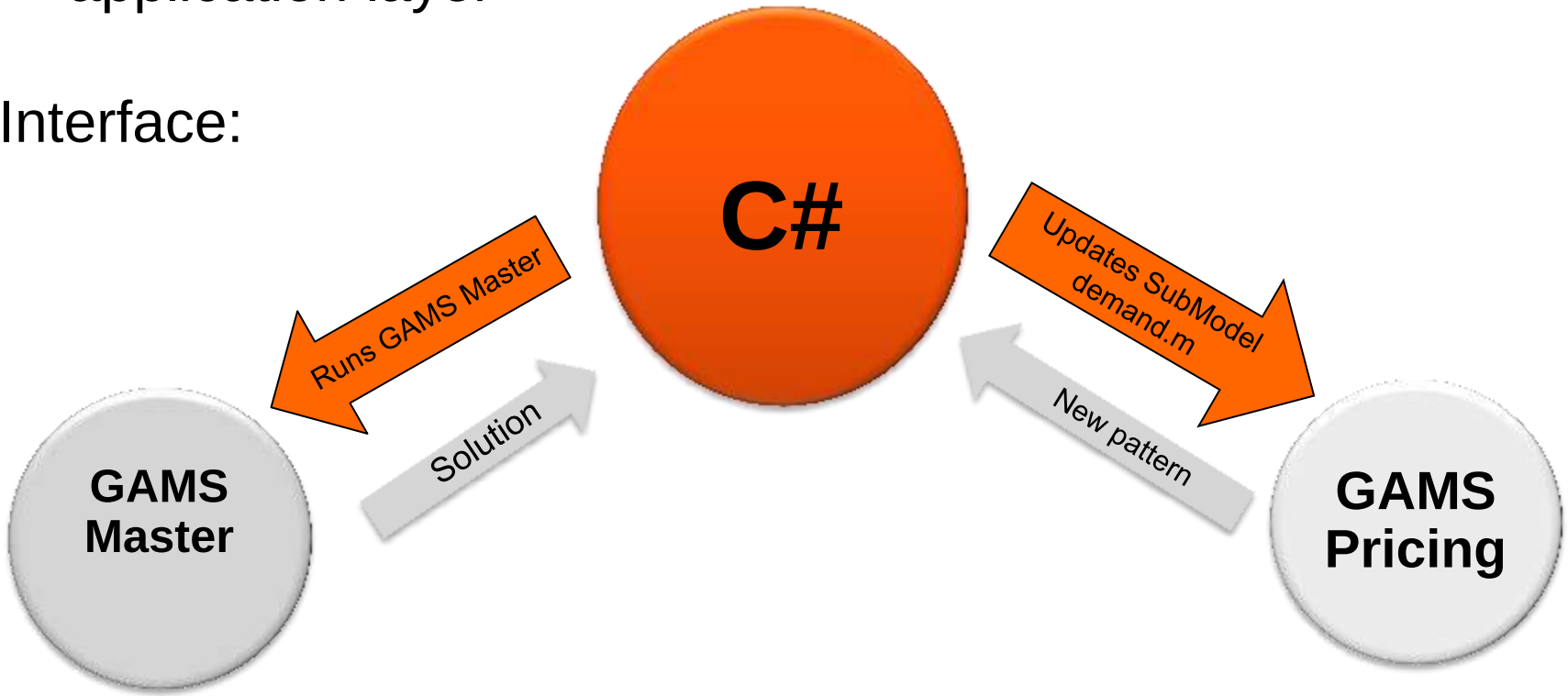
- What do we need?
  - Cutstock Class
  - GAMSOptions
  - GAMSException

➔ Step2.cs

Application Step 3

- What are we going to do?
  - Move the logic of the algorithm from GAMS into the application layer

- Interface:



GAMSJob.run - Method

```
masterJob.Run(opt, masterCP, cutstockData);
```

- Each solve needs to regenerate the model
- Model rim can change
- Data exchange between solves possible
- User updates GAMS Symbols instead of matrix coefficients

GAMSModelinstance.solve - Method

```
subMI.Solve();
```

- Works with particular model instance
- Saves model generation and solver setup time
- Hot start (keeps the model hot inside the solver and uses solver's best update and restart mechanism)
- Model rim unchanged between different solves
- Data exchange between solves possible

Excursus: GAMSModelInstance etc.

GAMSJob

- Manages the execution of a GAMS program given by GAMS model source

creates

GAMSCheckpoint

- Captures the state of a `GAMSJob`

initializes

GAMSModelInstance

- A single mathematical model generated by a GAMS solve statement

modifies

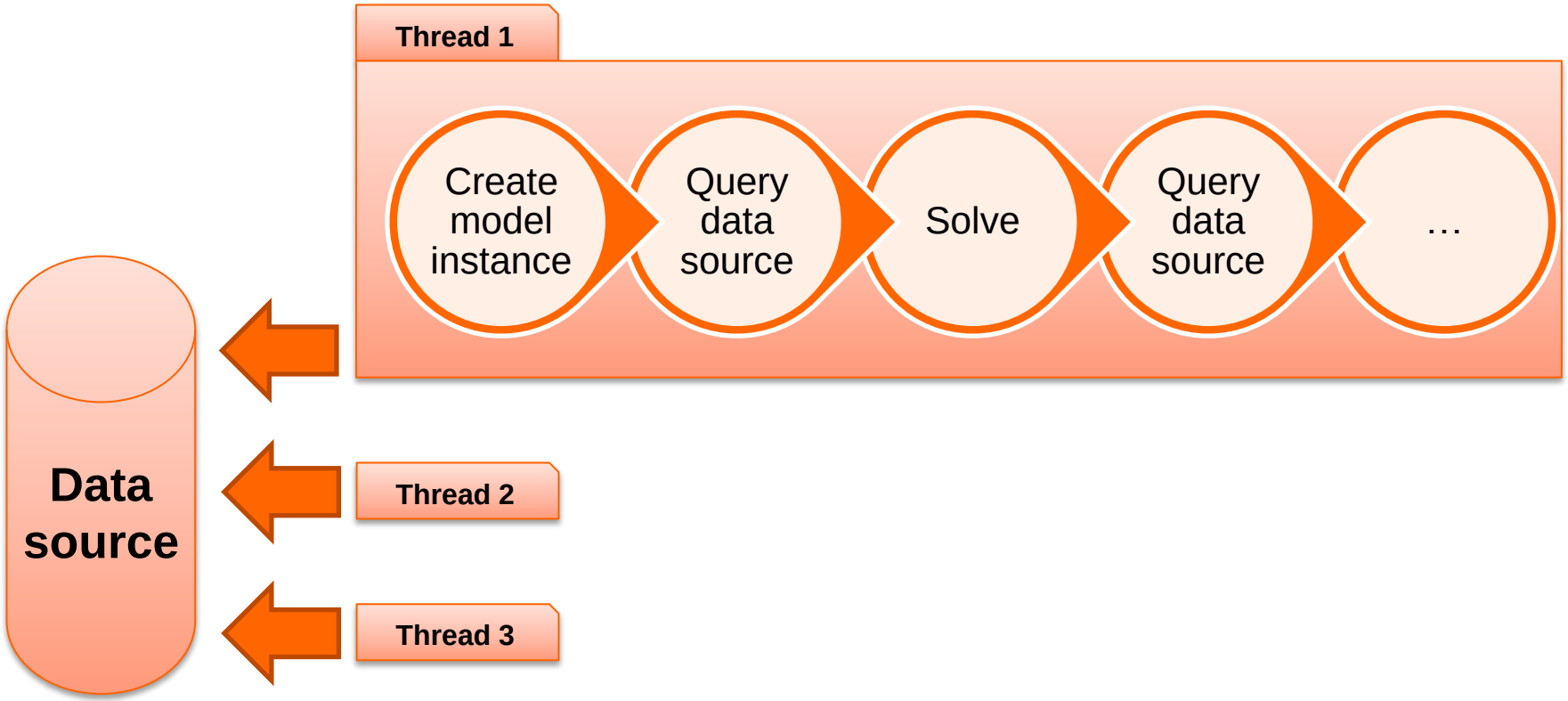
GAMSModifier

- Marks elements of a `GAMSModelInstance` to be modifiable

➔ Step3.cs

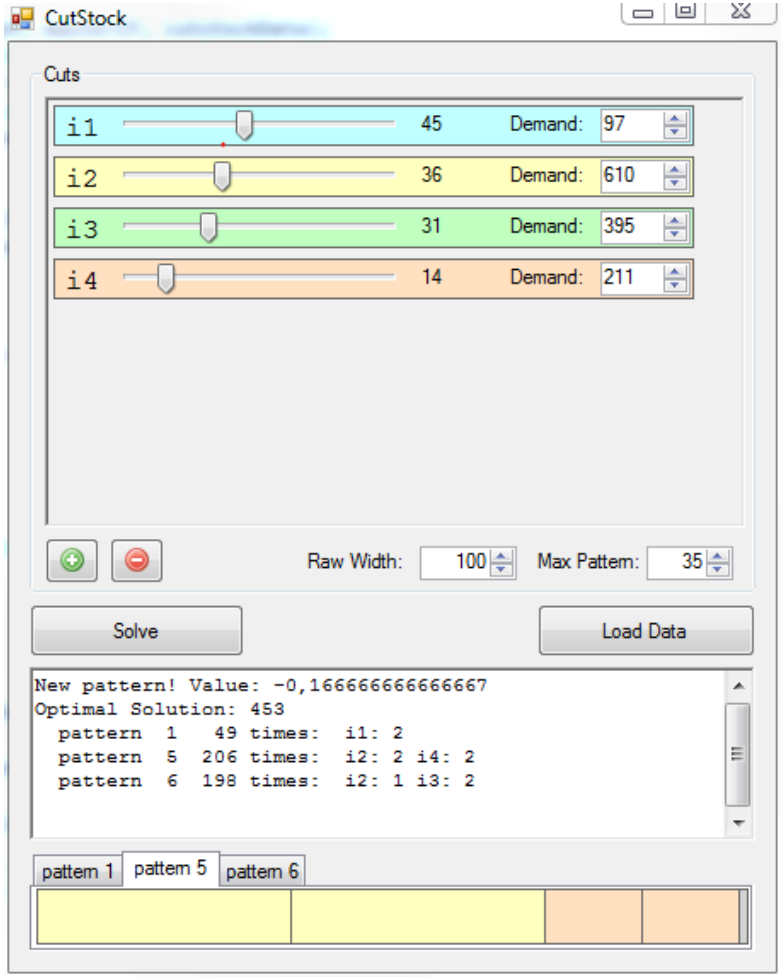
GAMSModelInstances in Parallel

- Multiple GAMSModelInstances running in parallel with one common data source:



Application Step 4

- What are we going to do?
  - Combine model and graphical user interface



➔ Step4.cs

Forthcoming Distribution 24.1

- Improved data handling for large datasets
- New or improved functions
- …

- Add new functions to GAMSModelinstance:
  - CopyModelInstance: Copies a ModelInstance to a new ModelInstance which gets constructed at this call
  - Interrupt: Sends interrupt signal to running GAMSModelInstance
- Added new functions to GAMSSymbol:
  - CopyToArray: Copies values of a dense symbol into a dense array
  - CopyToSqzdArray: Copies values of a sparse symbol into a squeezed array
  - CopySparseToDenseArray: Copies values of a sparse symbol into a dense array
  - CopyFromDenseArray: Copies values from dense array into a symbol
  - CopySliceFromDenseArray: Copies values from slice of dense array into a symbol
- Added new functions to GAMSSymbol and all its derived classes: GetRecord, which finds a record if it exists and adds it if not
- Added new functions to GAMSWorkspace: AddJobFromGamsLib, AddJobFromTestLib, AddJobFromEmpLib, AddJobFromDataLib and AddJobFromFinLib
- Added new properties to GAMSWorkspace: Version, MajorRelNumber, MinorRelNumber and GOLDRelNumber
- Change the GAMSWorkspace.Debug parameter from a Boolean flag to an enum type called DebugLevel
- Add new sub class of GAMSException: GAMSExceptionExecution. This provides additional info about the reason of the failed execution.
- Allow to define SymbolUpdateType for each GAMSModifier separately
- Significant performance improvement for function GAMSWorkspace.AddDatabase(GAMSDatabase)

Agenda

- What is new?
  - GAMS System
  - Platforms
  - Solvers
  - Interfaces

What is new: GAMS System

- Support for user-defined:
  - Macros
  - Function libraries
- Asynchronous execution
- Scenario Solver (GUSS)
- Extended Mathematical Programming (EMP)

What is new: Platforms

- Support for MAC OS X
- Cross-platform licenses
- Wine (Linux, Mac**)**

What is new: Solvers

- **GLoMIQO**: Branch-and-bound global optimization for mixed-integer quadratic models

- **Lindo**: Global and stochastic optimization

- **SULUM:** Linear and mixed integer optimization

New GAMS Distribution 24.0.2

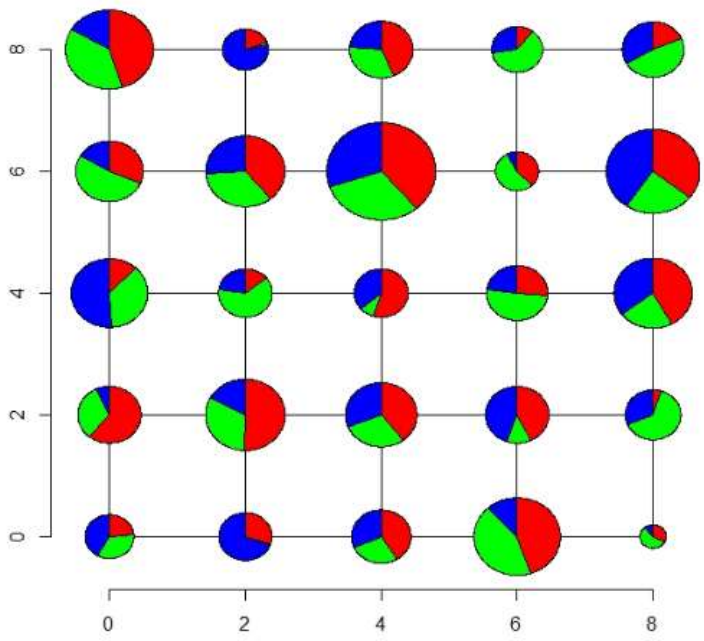
Released February 24, 2013 www.gams.com/download

- **Solver updates**
  - BARON 11.9.1
  - CONOPT 3.15I
  - CPLEX 12.5
  - GLOMIQO 2.0
  - GUROBI 5.1
  - KNITRO 8.0
  - LINDO 7.0.1.487
  - MOSEK 6 rev 148
  - XPRESS 23.01.06
  - …

What is new: Interfaces

- OO API for .NET, Java and Python

- GDXRRW
  - Connects R-System with GAMS
  - Presents data in a natural form for R users



Source: http://blog.modelworks.ch

Thank You !

USA

GAMS Development Corp.
1217 Potomac Street, NW
Washington, DC 20007
USA

Phone: +1 202 342 0180
Fax: +1 202 342 0181

sales@gams.com

Europe

GAMS Software GmbH
P.O. Box 40 59
50216 Frechen
Germany

Phone: +49 221 949 9170
Fax: +49 221 949 9171

info@gams.de

http://www.gams.com