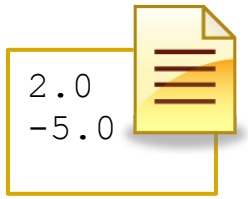
Service Oriented Computing EnviRonment (SORCER)

Evaluators, Vars, Models

R. Kolonay
S. Burton
M. Sobolewski

Outline

- Matlab Example to illustrate SORCER basics
  - Provider
  - Context
  - Task
  - Job
  - Requestor
- Evaluators, Filters, Vars, Multi-Fidelity, Models
- ModelClient example
  - Command line
  - Matlab
- Optimization
  - From SORCER
  - From Matlab



c:\myStuff\input1.txt

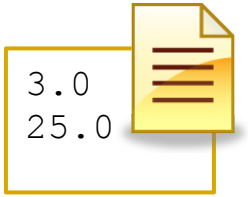
ctx_in = {'c:\myStuff\input1.txt'}

Example Matlab function...takes a cell array as input, returns a cell array as output.

doService1.m

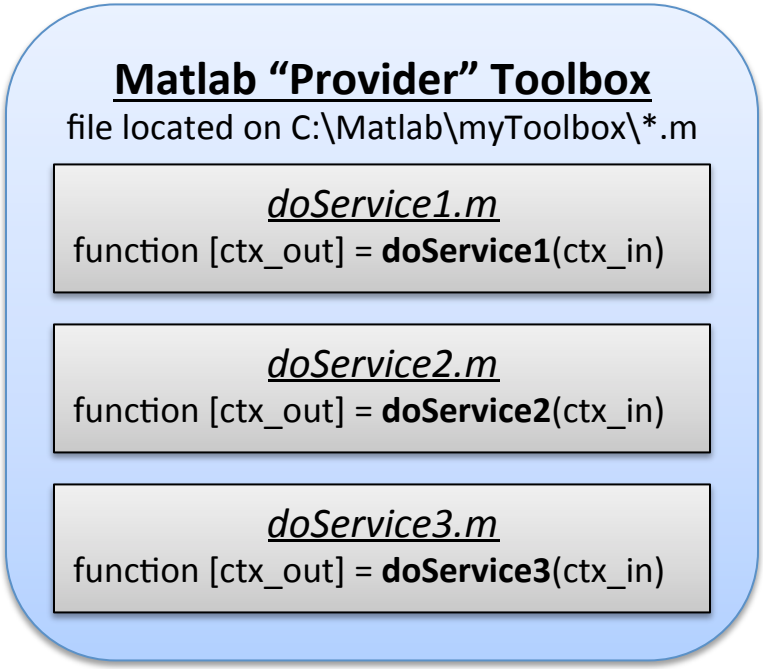
```
function [ctx_out] = doService1(ctx_in)

inputFile = ctx_in{1};

x = load(inputFile);
y1 = x(1) +1;
y2 = x(2)^2;
y = [y1; y2];

outputFile = 'c:\scratchDir\output1.txt'

save(outputFile, 'y', '-ascii');

ctx_out = {outputFIle};
```

ctx_out = {'c:\scratchDir\output1.txt'}



c:\scratchDir\output1.txt

A Matlab Toolbox is a collection of functions (*.m files) in a directory...



Matlab "Requestor" script calls Toolbox functions and passes data between function calls...

Matlab "Requestor" Script

file located on C:\myStuff\requestor.m

```
addpath('C:\Matlab\myToolbox')

ctx_in1 = {'c:\myStuff\input1.txt'}

ctx_out1 = doService1(ctx_in1);


ctx_in2 = {     ctx_out1{1}
                'c:\myStuff\input2.txt'     };
ctx_out2 = doService2(ctx_in2)

display(ctx_out2);
```

c:\myStuff\input1.txt

c:\myStuff\input2.txt

Matlab "Requestor" Script

file located on C:\myStuff\requestor.m

```
addpath('C:\Matlab\myToolbox')


ctx_in1 = {'c:\myStuff\input1.txt'}
ctx_out1 = doService1(ctx_in1);


ctx_in2 = {  ctx_out1{1}              'c:
\myStuff\input2.txt' };
ctx_out2 = doService2(ctx_in2)

display(ctx_out2);
```

Matlab "Provider" Toolbox

file located on C:\Matlab\myToolbox*.m

doService1.m

function [ctx_out] = **doService1**(ctx_in)

doService2.m

function [ctx_out] = **doService2**(ctx_in)

doService3.m

function [ctx_out] = **doService3**(ctx_in)

c:\myStuff\input1.txt

c:\myStuff\input2.txt

Matlab "Requestor" Script

file located on C:\myStuff\requestor.m

```
addpath('C:\Matlab\myToolbox')


ctx_in1 = {'c:\myStuff\input1.txt'}
ctx_out1 = doService1(ctx_in1);


ctx_in2 = {  ctx_out1{1}              'c:
\myStuff\input2.txt' };
ctx_out2 = doService2(ctx_in2)

display(ctx_out2);
```

ctx_in1

Matlab "Provider" Toolbox

file located on C:\Matlab\myToolbox*.m

doService1.m

function [ctx_out] = **doService1**(ctx_in)

doService2.m

function [ctx_out] = **doService2**(ctx_in)

doService3.m

function [ctx_out] = **doService3**(ctx_in)

c:\myStuff\input1.txt

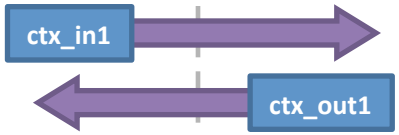
c:\myStuff\input2.txt

c:\scratchDir\output1.txt

Matlab "Requestor" Script

file located on C:\myStuff\requestor.m

```
addpath('C:\Matlab\myToolbox')


ctx_in1 = {'c:\myStuff\input1.txt'}
ctx_out1 = doService1(ctx_in1);


ctx_in2 = {  ctx_out1{1}              'c:
\myStuff\input2.txt' };
ctx_out2 = doService2(ctx_in2)

display(ctx_out2);
```



Matlab "Provider" Toolbox

file located on C:\Matlab\myToolbox*.m

doService1.m
function [ctx_out] = **doService1**(ctx_in)

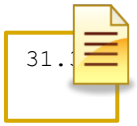
doService2.m
function [ctx_out] = **doService2**(ctx_in)

doService3.m
function [ctx_out] = **doService3**(ctx_in)

c:\myStuff\input1.txt

c:\myStuff\input2.txt

c:\scratchDir\output1.txt

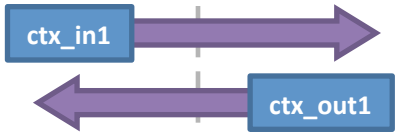


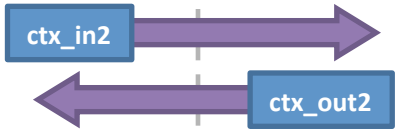
c:\scratchDir\output2.txt

Matlab "Requestor" Script

file located on C:\myStuff\requestor.m

```
addpath('C:\Matlab\myToolbox')


ctx_in1 = {'c:\myStuff\input1.txt'}
ctx_out1 = doService1(ctx_in1);


ctx_in2 = {  ctx_out1{1}              'c:
\myStuff\input2.txt' };
ctx_out2 = doService2(ctx_in2)

display(ctx_out2);
```





Matlab "Provider" Toolbox

file located on C:\Matlab\myToolbox*.m

doService1.m

function [ctx_out] = **doService1**(ctx_in)

doService2.m

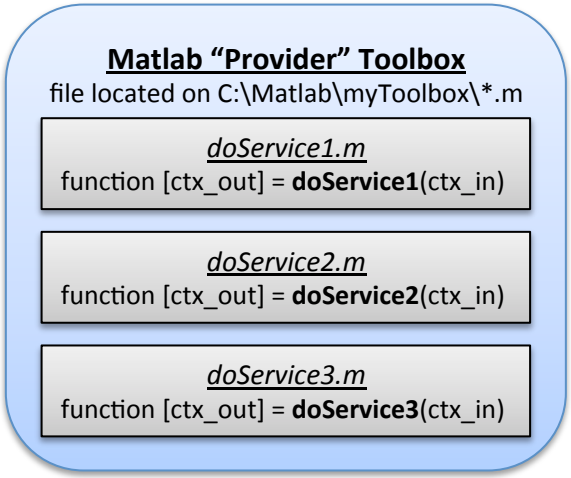
function [ctx_out] = **doService2**(ctx_in)

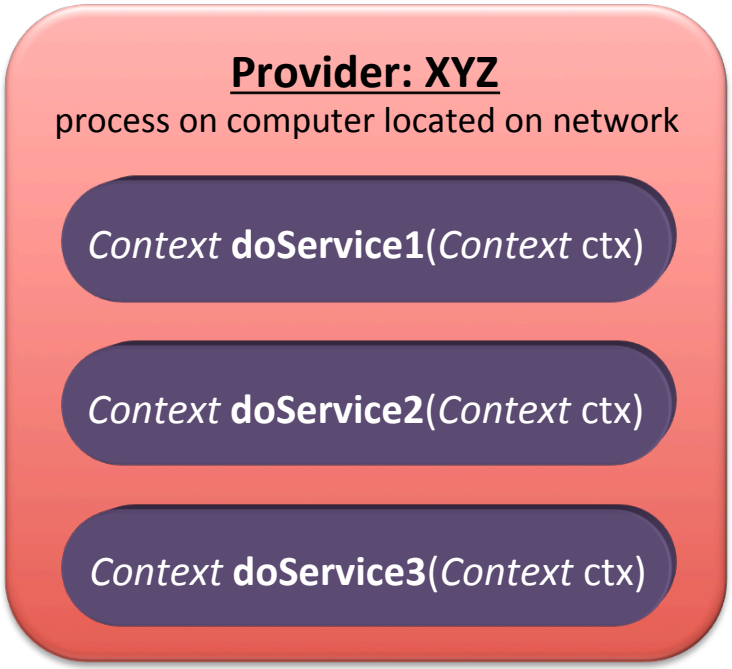
doService3.m

function [ctx_out] = **doService3**(ctx_in)

SORCER Provider is like a Matlab Toolbox...

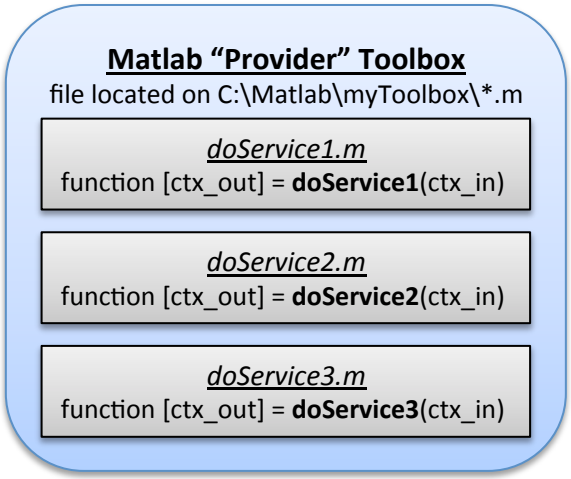
...but instead of files in a directory, it's a Java object on a remote computer

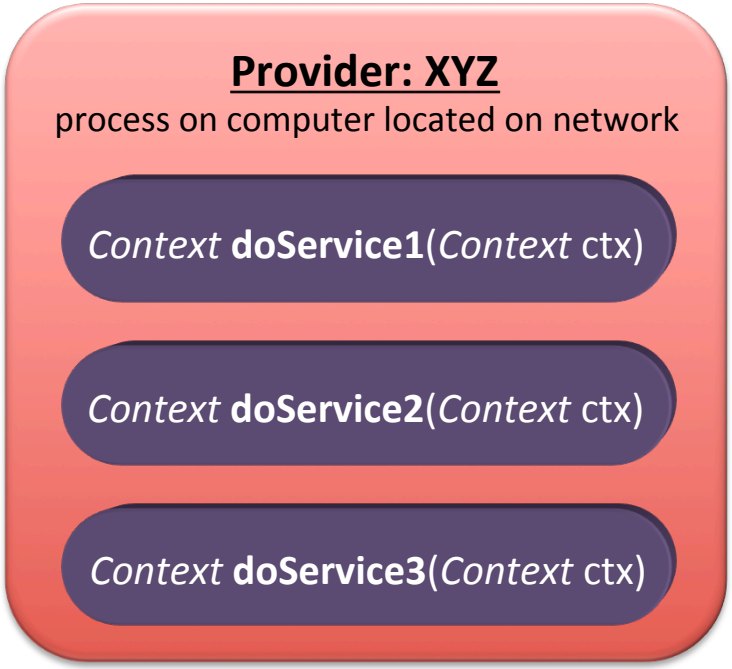




A SORCER Provider has services that take a Context as input and return a Context as output.

A Context object is like a Matlab cell object or structure... key-value pairs.





Values in Cell objects in Matlab are accessed by index...

...values in SORCER Context objects are accessed via a key (a.k.a. path) or by datatype

Matlab Cell: ***index ⇔ value***

ctx_in1{1} = 'c:\myStuff\input1.txt'

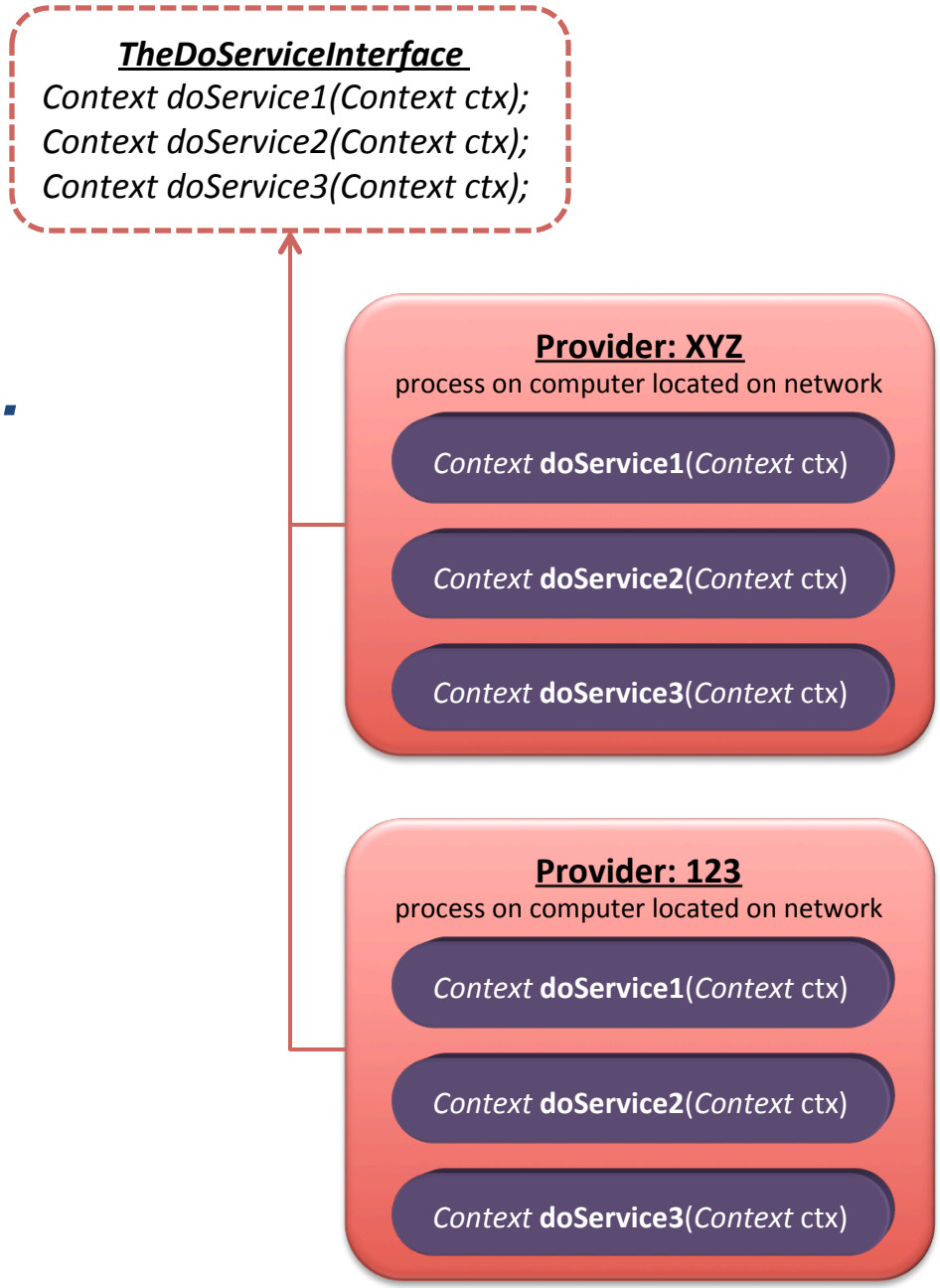
SORCER Context: ***key ⇔ value***
datatype ⇔ value

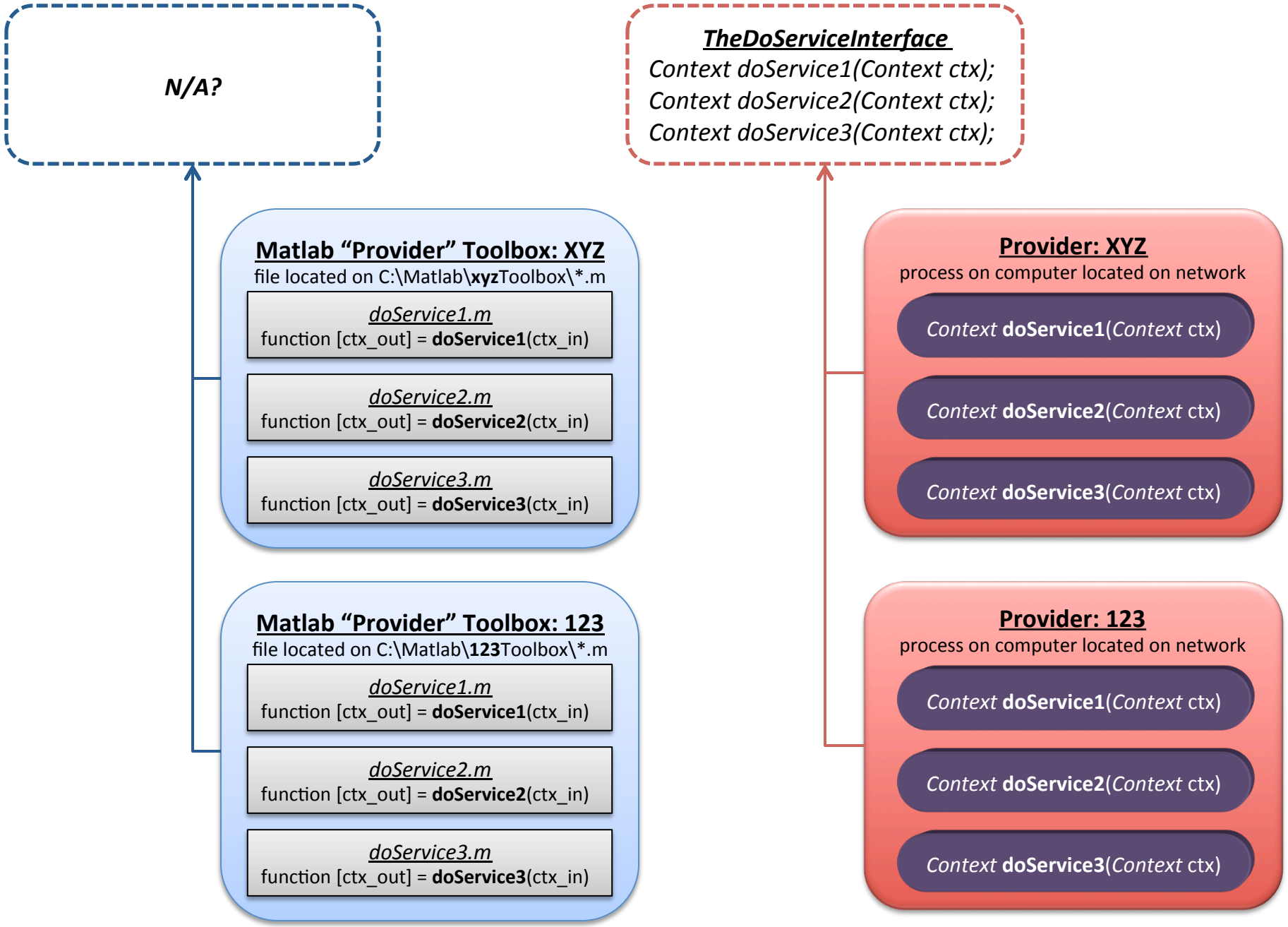
"INPUT/FILE1" ⇔ new URL("file:\\c:\myStuff\input1.txt")

Providers generally implement an interface…

…so that multiple implementations of services may exist

…and users of services only need to specify/ write code for the interface.




N/A?
TheDoServiceInterface
Context doService1(Context ctx);
Context doService2(Context ctx);
Context doService3(Context ctx);
Matlab "Provider" Toolbox: XYZ
file located on C:\Matlab\xyzToolbox*.m
doService1.m
function [ctx_out] = doService1(ctx_in)
doService2.m
function [ctx_out] = doService2(ctx_in)
doService3.m
function [ctx_out] = doService3(ctx_in)
Matlab "Provider" Toolbox: 123
file located on C:\Matlab\123Toolbox*.m
doService1.m
function [ctx_out] = doService1(ctx_in)
doService2.m
function [ctx_out] = doService2(ctx_in)
doService3.m
function [ctx_out] = doService3(ctx_in)
Provider: XYZ
process on computer located on network
Context doService1(Context ctx)
Context doService2(Context ctx)
Context doService3(Context ctx)
Provider: 123
process on computer located on network
Context doService1(Context ctx)
Context doService2(Context ctx)
Context doService3(Context ctx)

To access Provider services, you need to know the:

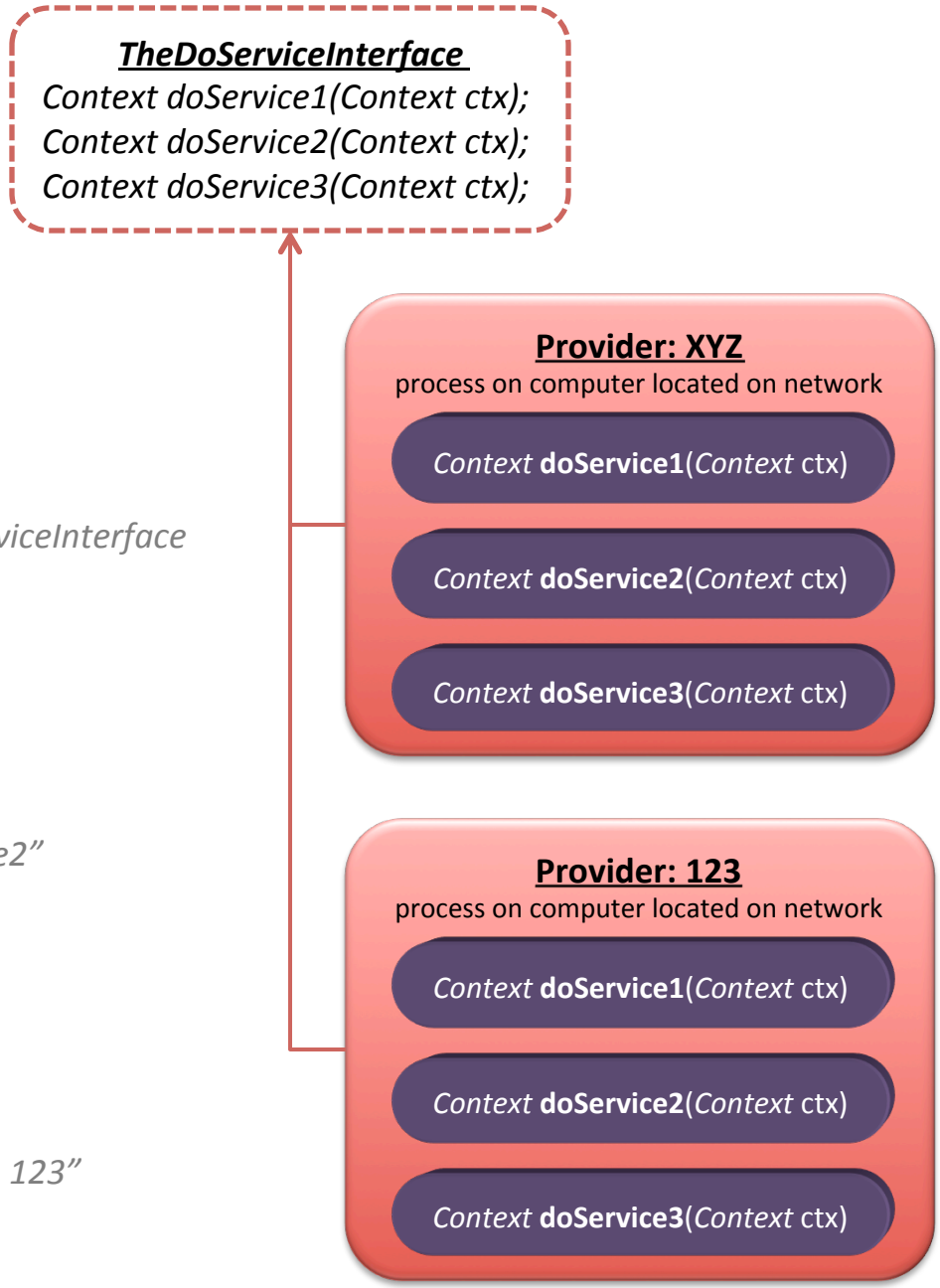
Interface/Class *e.g., TheDoServiceInterface*

and

Service Name *e.g., "doService2"*

…optionally

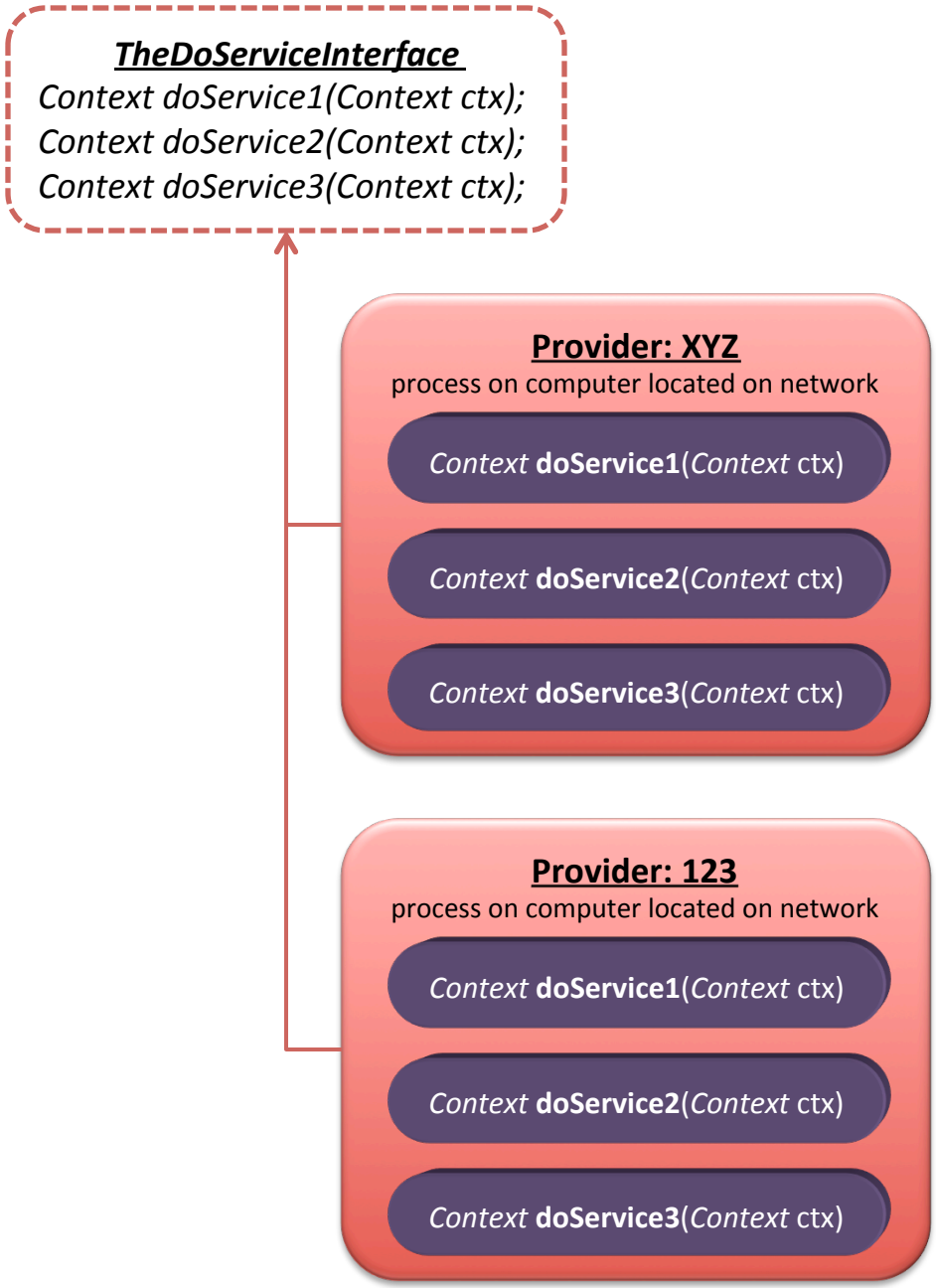
Provider Name *e.g., "Provider: 123"*



SORCER Providers may have the same services…

…if you need a specific Provider of a service, can access by Provider name

…if not, just ask for the Class/Interface and service name.



A SORCER Requestor is like a Matlab script that calls Toolbox functions...

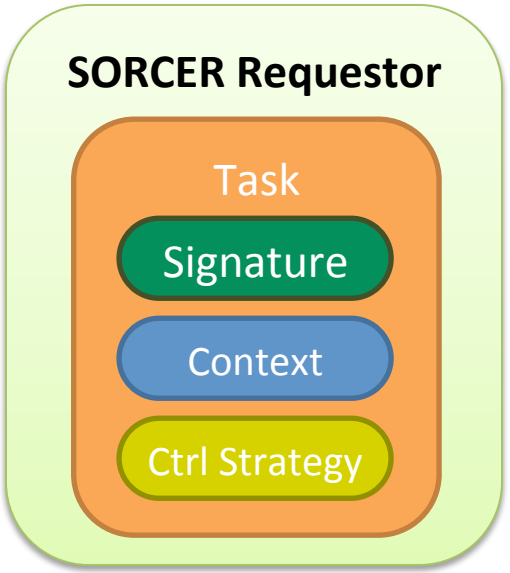
...it's a Java program that creates several objects to access and pass data to-and-from SORCER services

To use a service in SORCER, one needs to construct a Task…

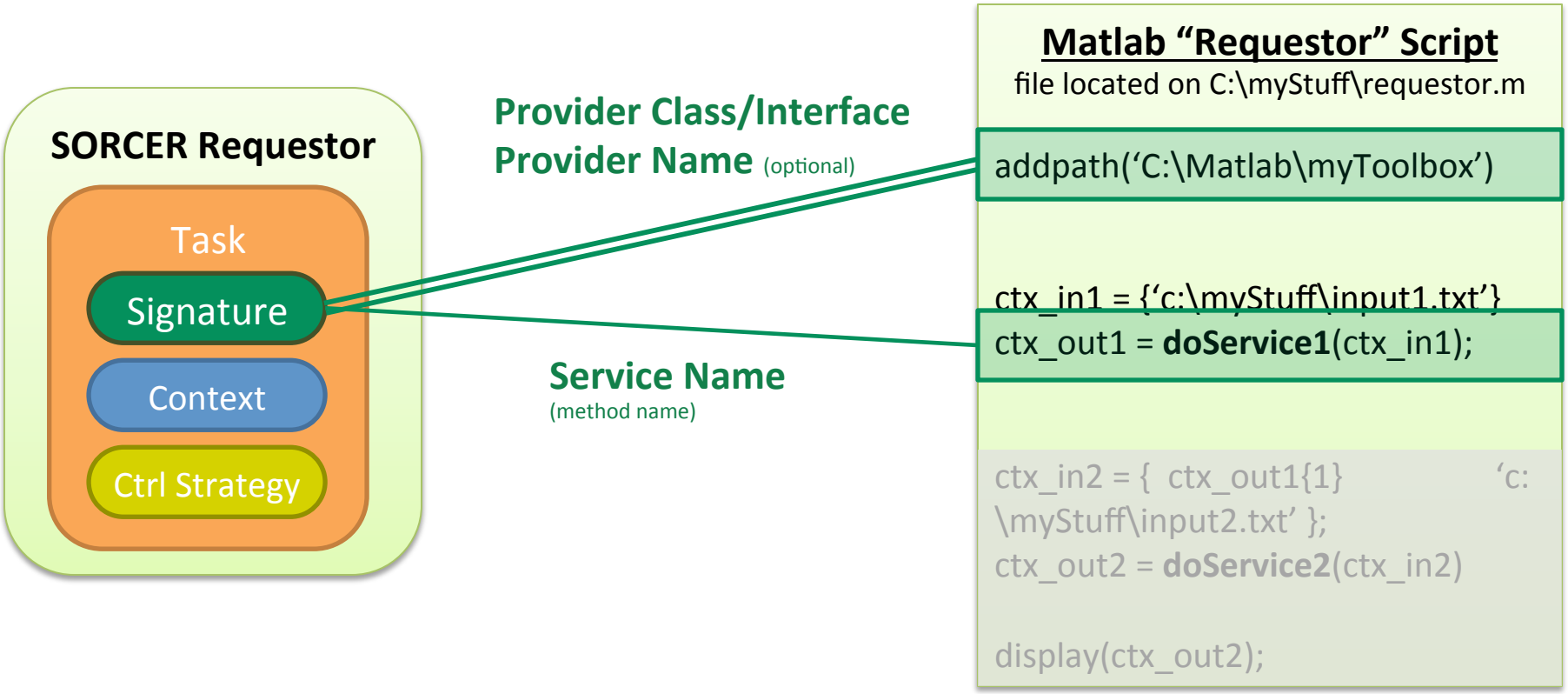
Matlab "Requestor" Script

file located on C:\myStuff\requestor.m

```
addpath('C:\Matlab\myToolbox')


ctx_in1 = {'c:\myStuff\input1.txt'}
ctx_out1 = doService1(ctx_in1);


ctx_in2 = {  ctx_out1{1}                'c:
\myStuff\input2.txt' };
ctx_out2 = doService2(ctx_in2)

display(ctx_out2);
```

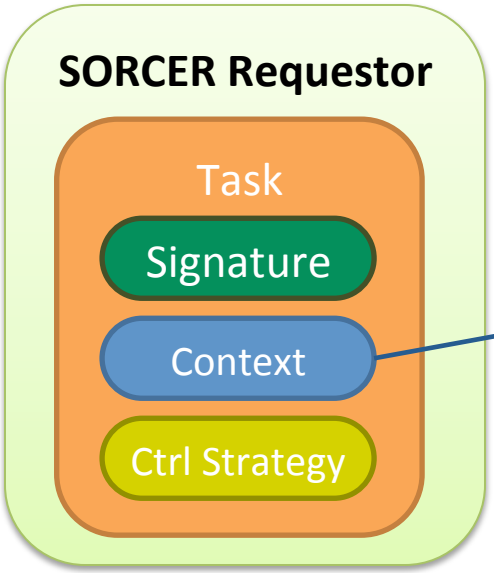


A SORCER Task has two primary entities: Signature and Context…



…the Signature specifies potential service Providers and the service name.

A SORCER Context is both the argument for a Service and the return value from a Service…



Key ⇔ Value Pairs

{1} ⇔'c:\myStuff\input1.txt'

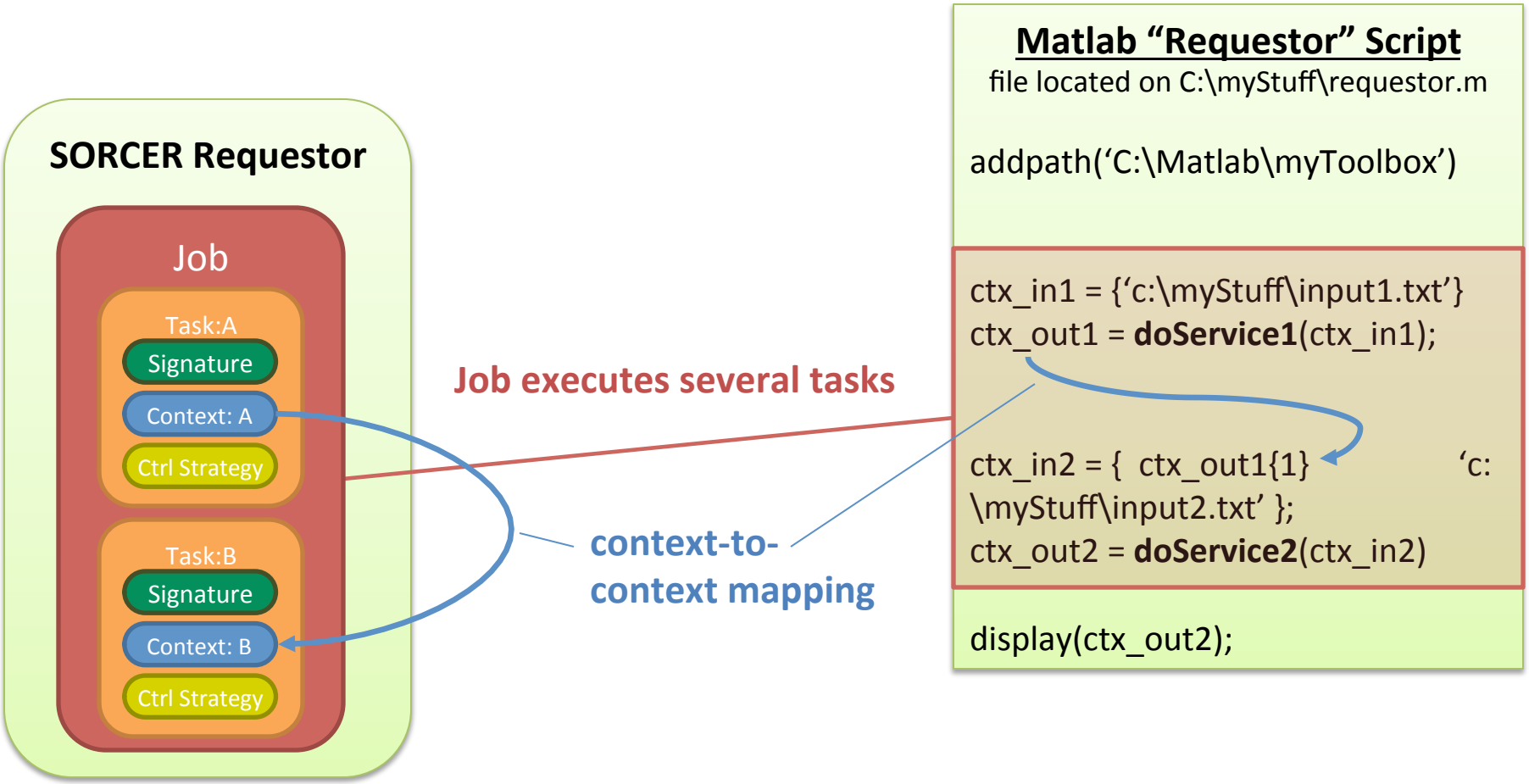
Matlab "Requestor" Script

file located on C:\myStuff\requestor.m

```
addpath('C:\Matlab\myToolbox')

ctx_in1 = {'c:\myStuff\input1.txt'}
ctx_out1 = doService1(ctx_in1);

ctx_in2 = {  ctx_out1{1}                'c:
\myStuff\input2.txt' };
ctx_out2 = doService2(ctx_in2)

display(ctx_out2);
```

…A Context is a set of key⇔ value pairs; keys look like file paths, values are any Java Object (e.g., URL objects)

A SORCER Job is a collection of Tasks…



Summary…

Provider: *Java object on the network that implements an interface (a.k.a. Service Type) with services required by the interface*

Service: *Provider method that takes a Context as input and returns a Context as output*

Context*:* *Argument for Provider Services; key ⇔ value pairs*

Summary…

Requestor: *Java program that creates several objects to access and pass data to-and-from SORCER services*

Task: *Java object that encapsulates data necessary to identify a service and the Context argument for that service*

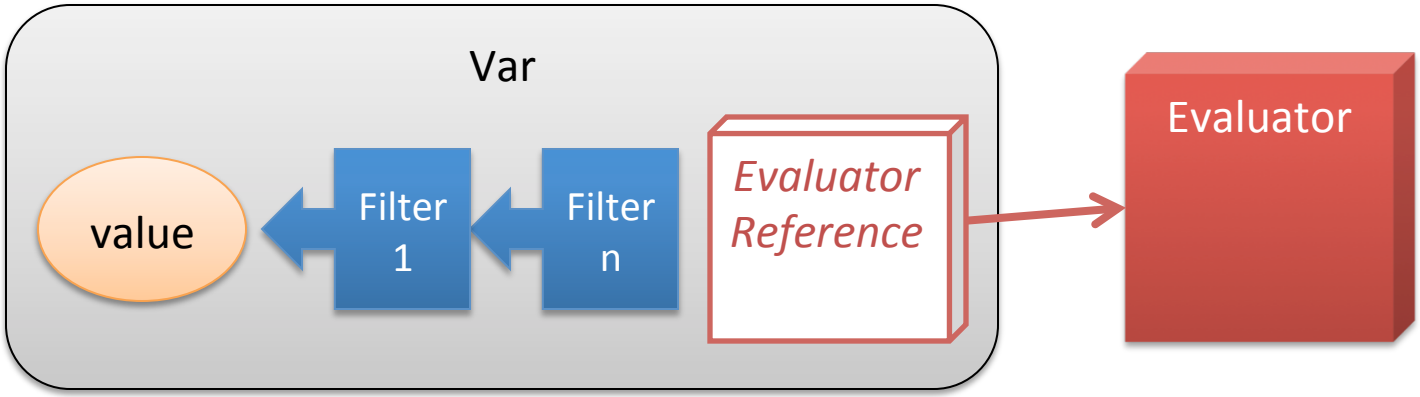
Job*:* *Collection of Tasks*

What about Math?

…Tasks and Jobs only deal with Contexts in and out, and typically contexts contain references to aggregate data (i.e., URLs)

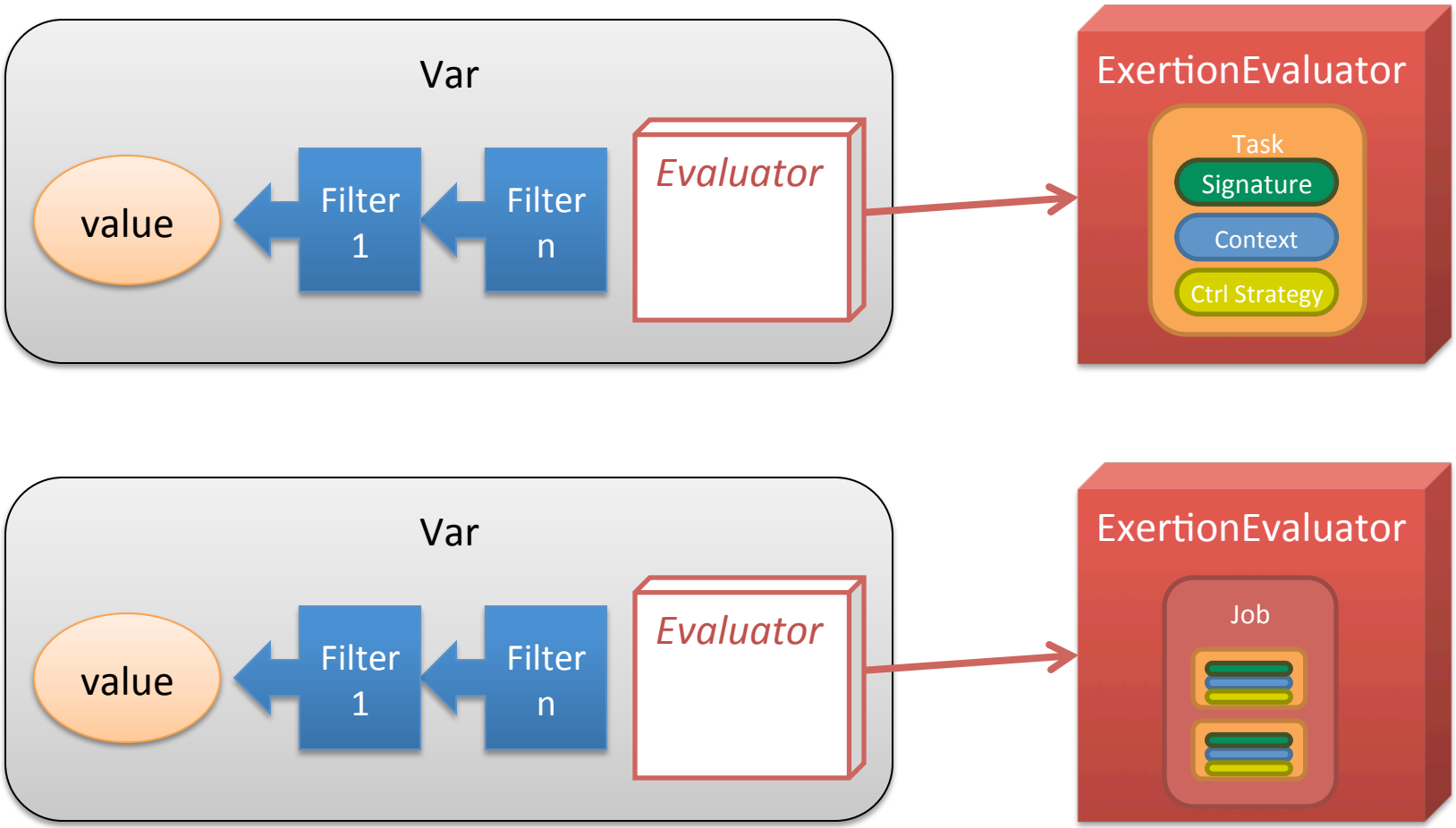
…Var objects in SORCER are the way to math

A Var object is a combination of an Evaluator and Filter…

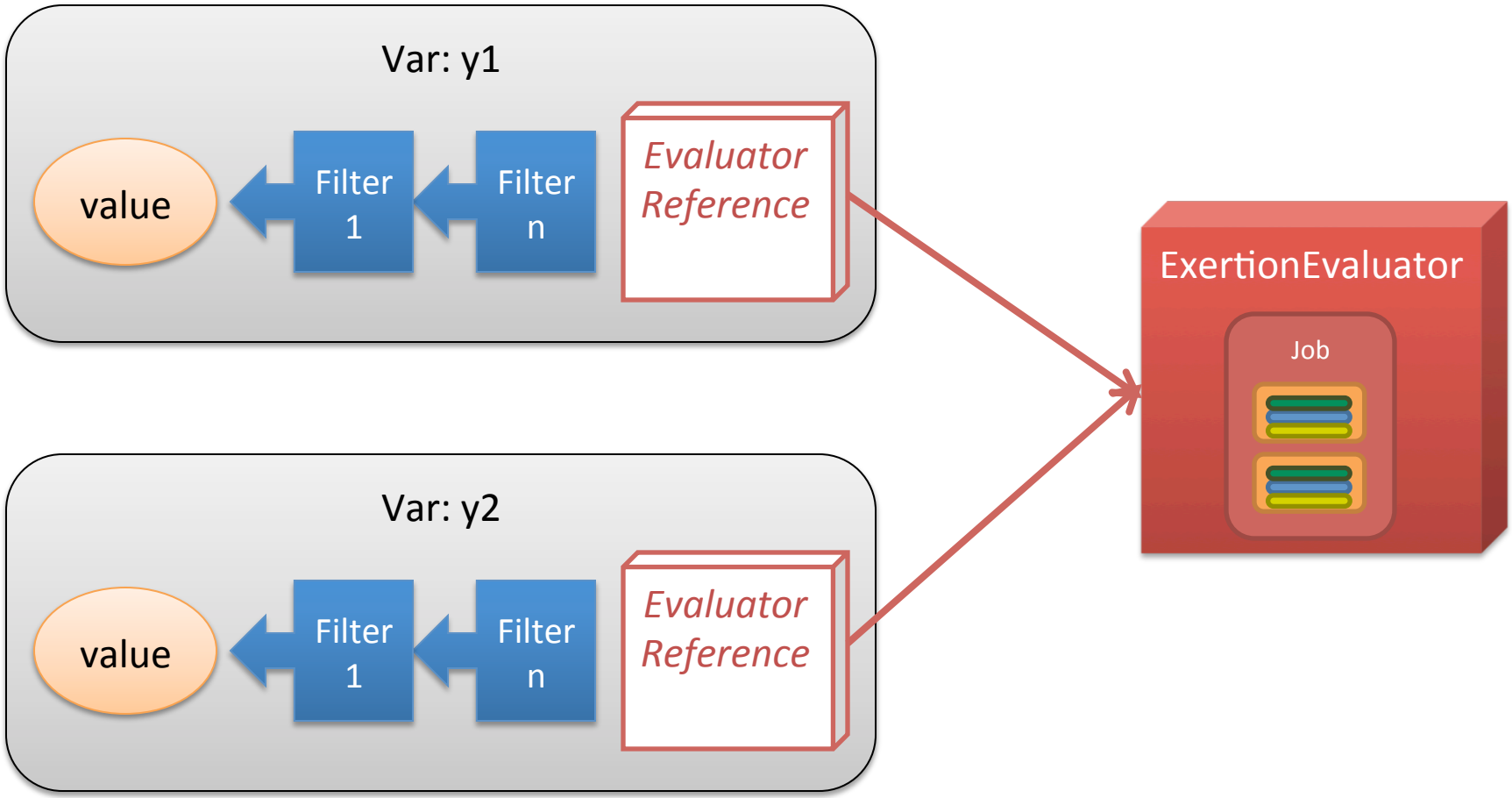


…Evaluators produce aggregate data, Filter reduce to scalar (e.g.)

Jobs and Tasks have an evaluator, called ExertionEvaluator…

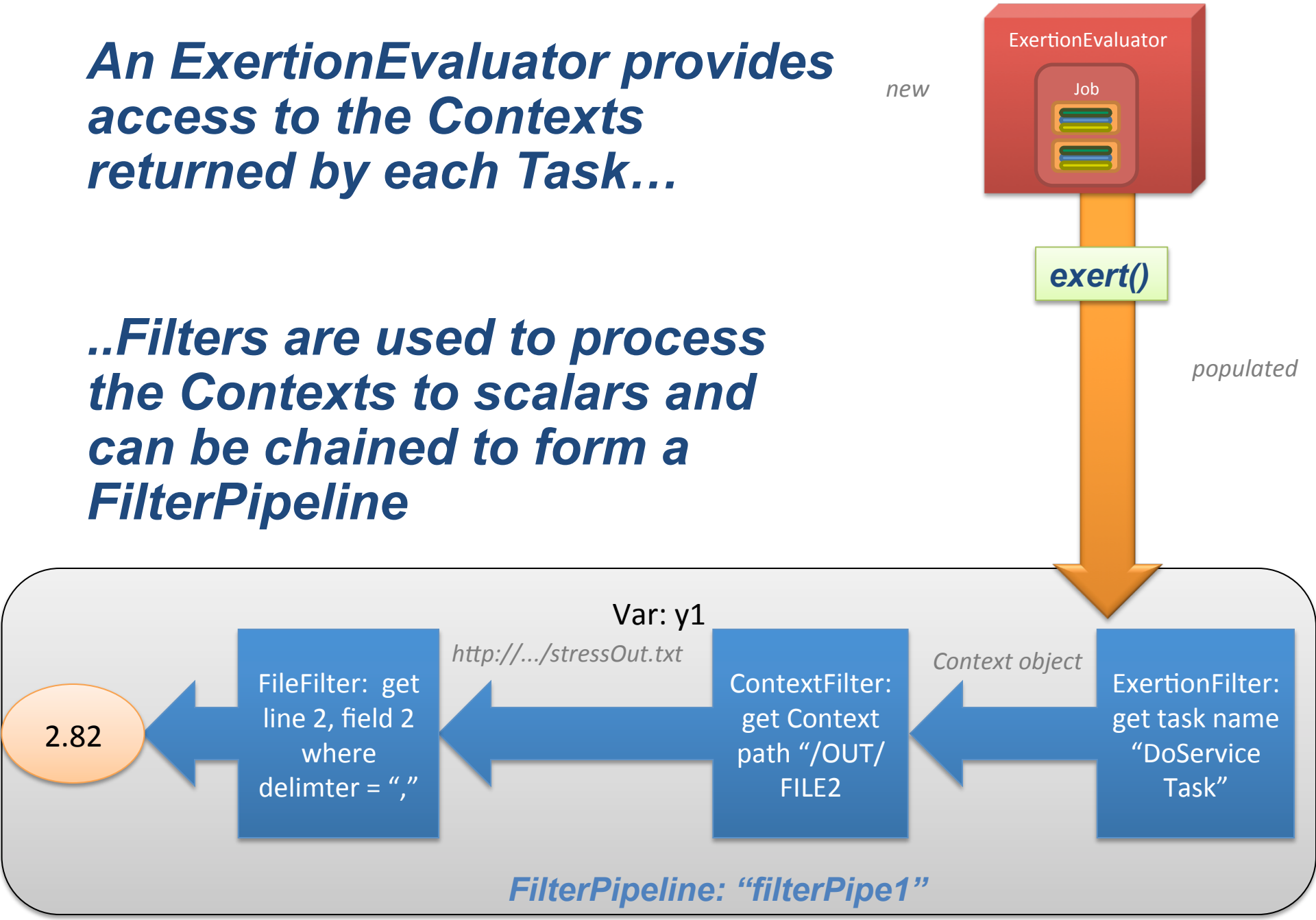


One Evaluator can produce data for many Vars…



An ExertionEvaluator provides access to the Contexts returned by each Task…

..Filters are used to process the Contexts to scalars and can be chained to form a FilterPipeline



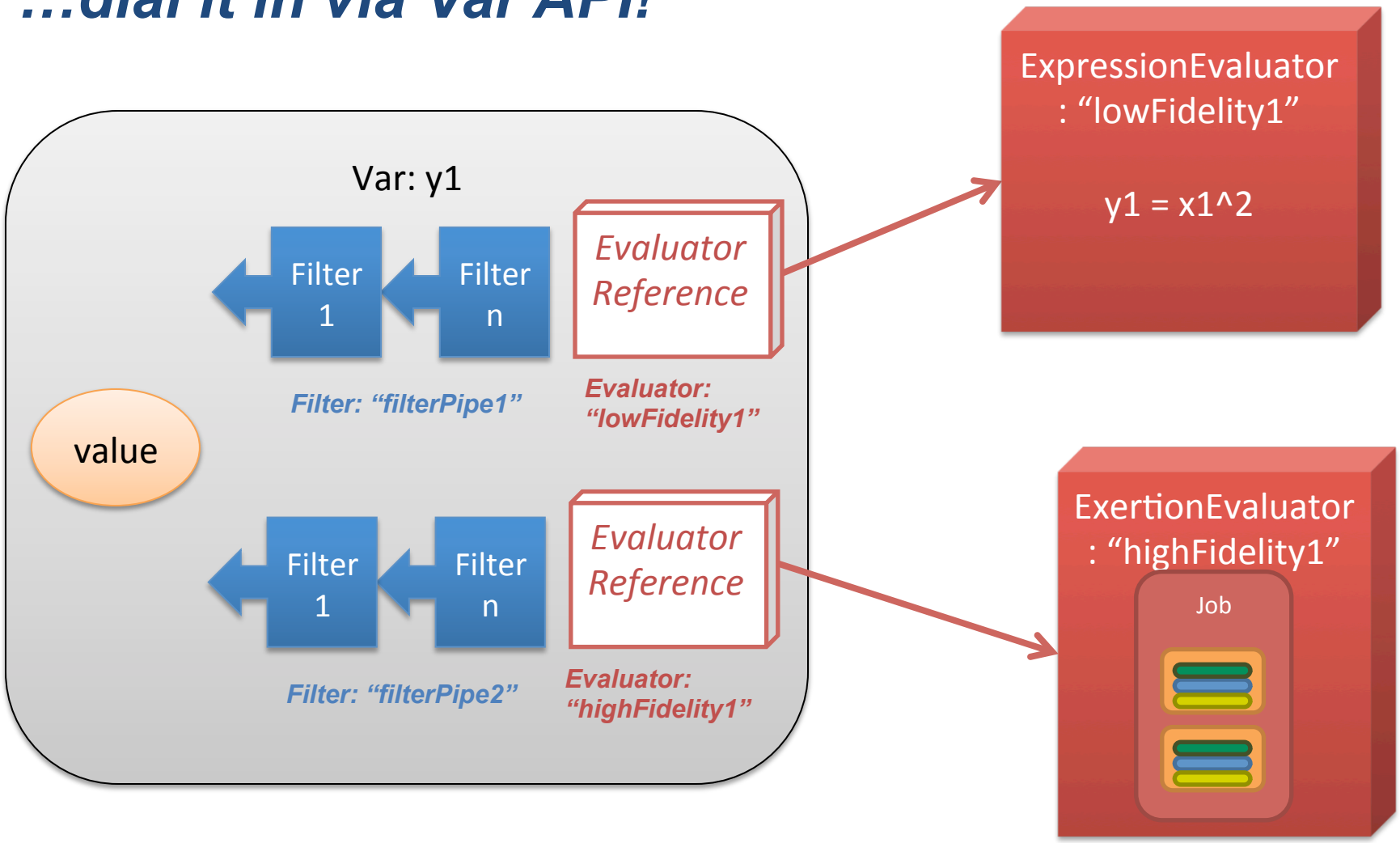
Lots of Evaluators…

- *ExertionEvaluator*
- *ExpressionEvaluator*
- *MethodEvaluator*

Lots of Filters…

- *FileFilter (for writing too)*
- *ObjectFilter*
- *MethodFilter*

What about Multi-Fidelity? …dial it in via Var API!



How do I share a bunch of Var objects?

Model Classes

…Models are collections of Vars that may be published as a Provider

…Published models may be used by, e.g., an Optimization Provider

How do I use Models?

…if you have a reference to a Model in a Requestor, you may query the Model with a ModelContext

…if the Model is published, you may query it via a Task

…also, the ModelClient class may be used in the Requestor, command line and Matlab (or any Java program).

Live Demo