

Thoughts on Passing Sets of Parameters Through WDL Workflows

Draft v3
2017_03_27
M. Noble

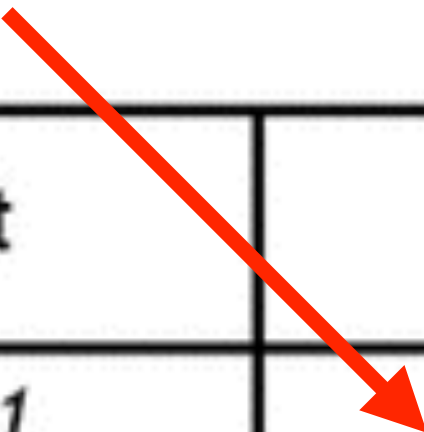
Why belabor this to such an extent?

Important to get it right, because:
our design should foster end->end conceptualization,
making it easier to reason about
(and explain or diagnose)
behavior of the entire system

Your time & minds are our most valuable asset
we need to spend them wisely

*These ideas sprang from GDAC use case, but
can in principle be more widely utilized.*

GDAN not completely out of the gate yet; sharpen the saw



	Urgent	Not Urgent
Important	Quadrant 1 <i>Examples:</i> <ul style="list-style-type: none">• Unscheduled rework• Last-minute changes• Dealing with late inputs from stakeholders, team• Forcing decisions & closure	Quadrant 2 <i>Examples:</i> <ul style="list-style-type: none">• Thoughtful, creative work• High-quality outputs• Productive collaborations• Training & development• Recreation & family time
Not Important	Quadrant 3 <i>Examples:</i> <ul style="list-style-type: none">• Low-value, but required, reports & presentations• Non-project emergencies• Miscellaneous interruptions• Administrivia	Quadrant 4 <i>Examples:</i> <ul style="list-style-type: none">• Over-analysis ("analysis-paralysis")• Pointless web-surfing• Gossip, idle speculation• Self-indulgent perfectionism

Definitions

An [engine](#) orchestrates the execution of workflows

A [workflow](#) is a directed, acyclic graph of one or more steps

A synonym for workflow is [pipeline](#); this term is commonly used by scientists

Each step in a workflow is a [task](#) (*singleton, atomic unit of execution*)

A [module](#) is the underlying code executed by a task (*e.g. Python, R*)

A [configuration](#) is the set of parameters uniquely specifying an instance of task execution; this includes inputs, outputs, and module version

Parameter values are assigned [dynamically](#) (*during runtime*) or [statically](#) (*beforehand*)

A static value is a constant, immutable [literal](#) in the configuration

A dynamic value is obtained by reading an [attribute](#), i.e. some named metadatum stored in runtime context outside the task

Two examples of [runtime context](#) are the Unix environment (*variables*) or [FireCloud workspace](#) (*data model attributes*)

The engine uses a [coordination language](#) to bind modules, tasks & workflows together and orchestrate their execution (*e.g. chaining outputs of one to inputs of another*)

Two such coordination languages are [WDL](#) & [CWL](#)

Most of these are well understood

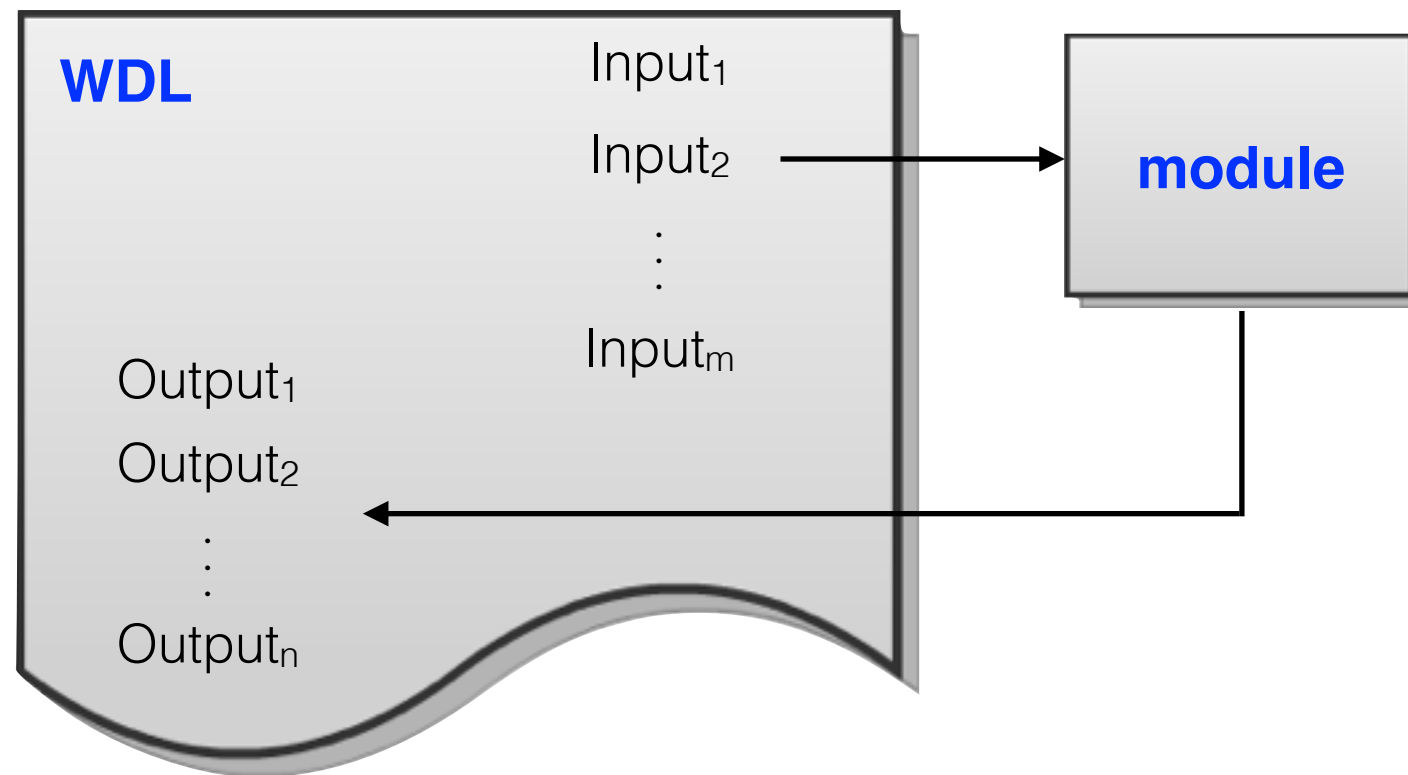
But enumerated here for completeness

As well as to underscore that for clarity & simplicity the eventual production version of FISSFC (currently in alpha) will likely hew to classical nomenclature:

task / flow / config

To minimize confusion of overloaded *workflow/task/config/method/repository* terminology between WDL & FireCloud

Typical singleton Task

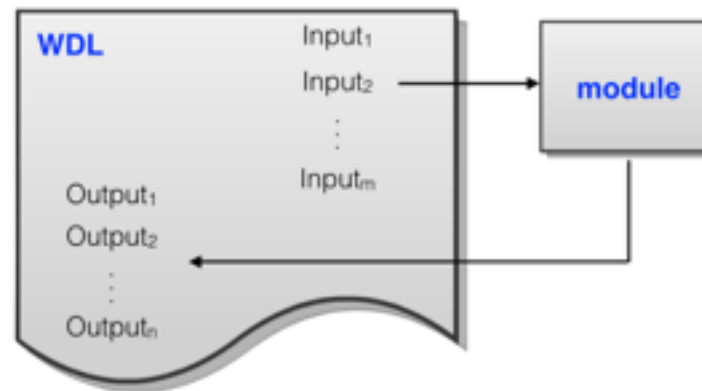


WDL specifies M input parameters
Those are passed to module
Which generates N artifacts
(figures, tables, metadata, etc)

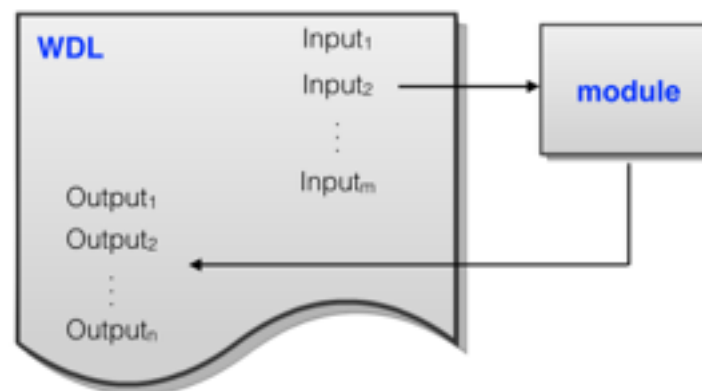
Each artifact we want to use elsewhere
must be listed in Output section

Typical Workflow

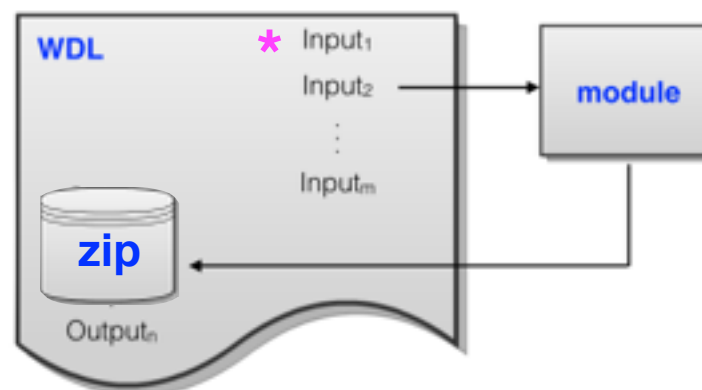
Analysis



Summary
Report
(html)



Package
for distribution



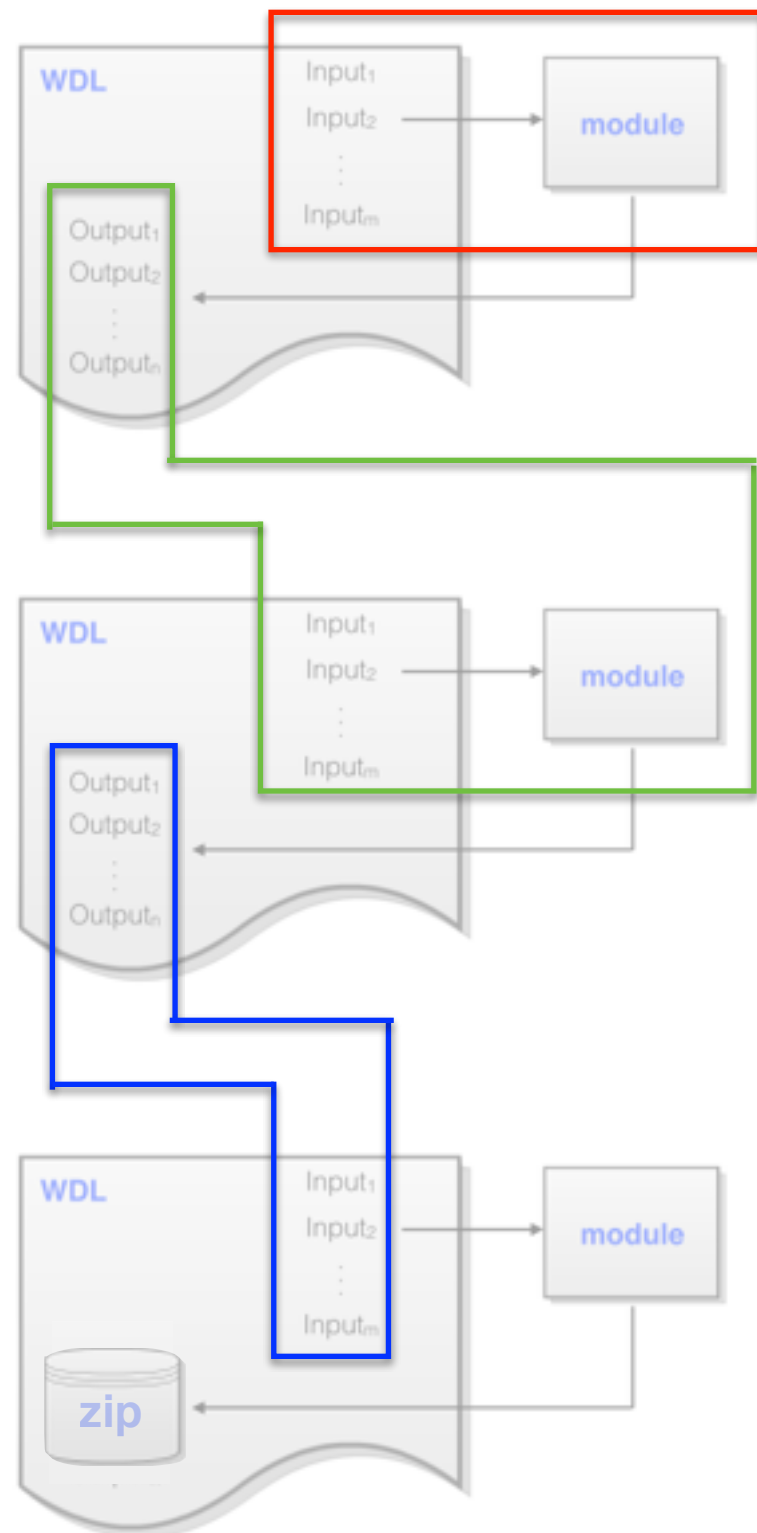
Composite of at
least 3 tasks,
typically 6 parts:

3 WDL
3 modules

Inputs & Outputs
explicitly
enumerated

Necessary for
persistence
after cloud VM
execution

Typical meme for moving a bolus of parameters thru a pipeline



Inputs listed twice:

First in WDL, then in
Then CLI args of Module1

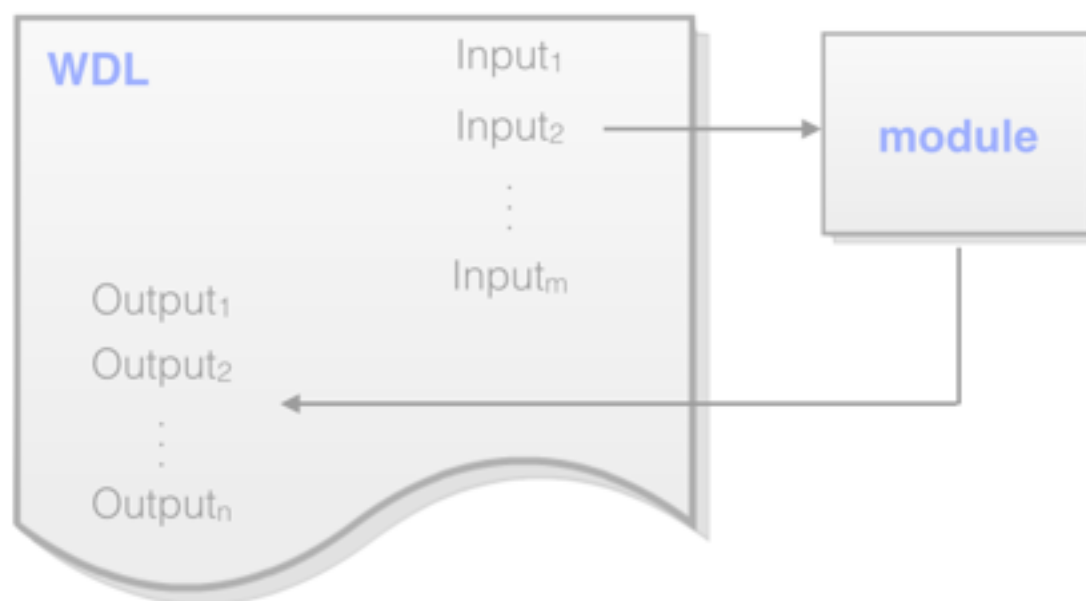
Then outputs listed thrice:

First in WDL1
Then WDL2 inputs
Then CLI args of Module2

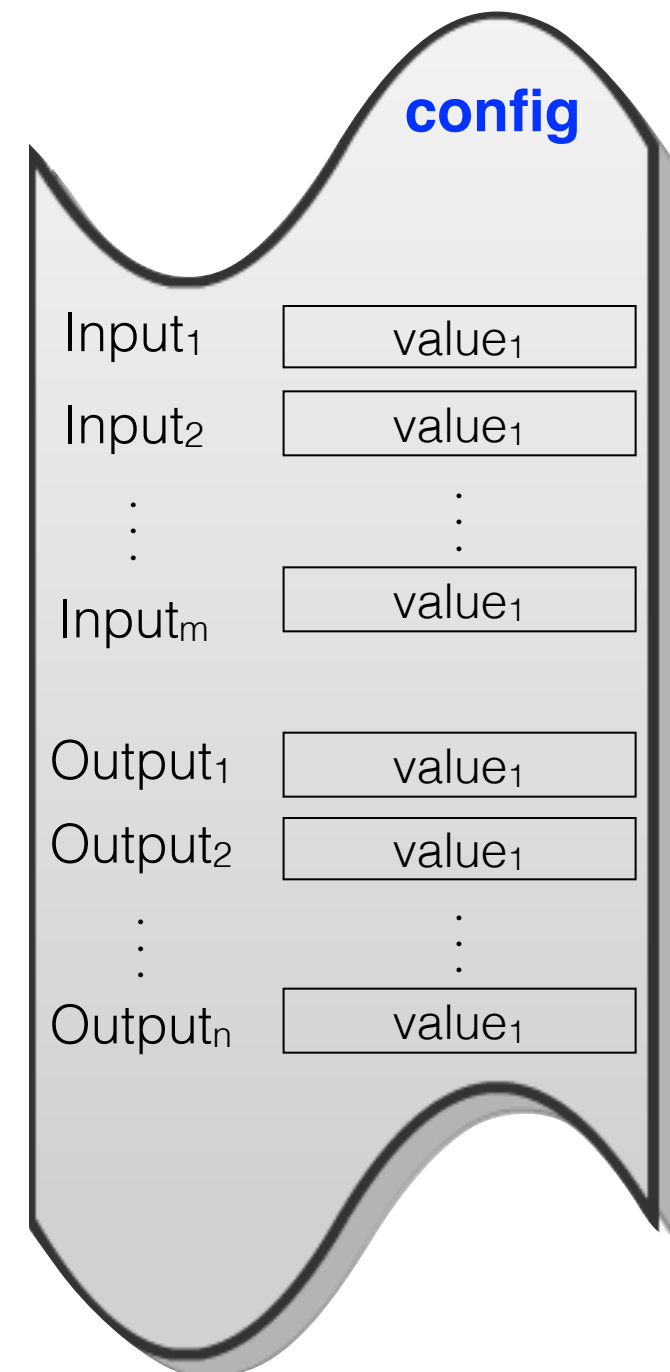
WDL1 outputs listed 4th,5th time
Passthru to WDL3 for packaging

Then CLI of packager
WDL2 outputs listed twice
First in WDL2
Then WDL3 inputs

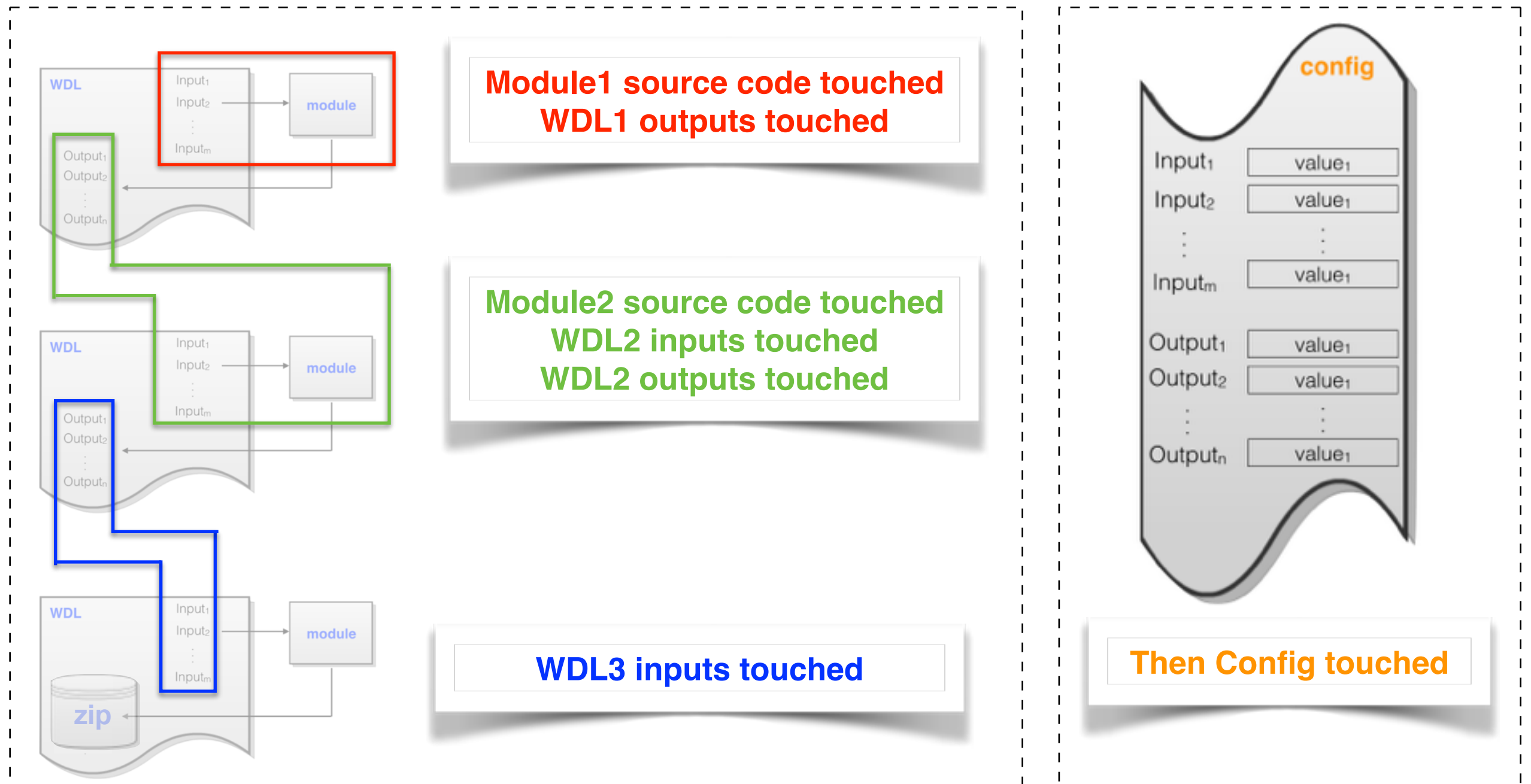
Quite verbose / duplicative
But it gets even worse



2X worse for each Task:
because each input/output
ALSO replicated in engine
configuration (e.g. FireCloud)



And Worse Still: suppose Task₁ outputs change, then ...



as many as 7 changes needed, across:

2 module src files
3 WDL files
1 task configuration

This is painfully tedious ...

Error prone ...

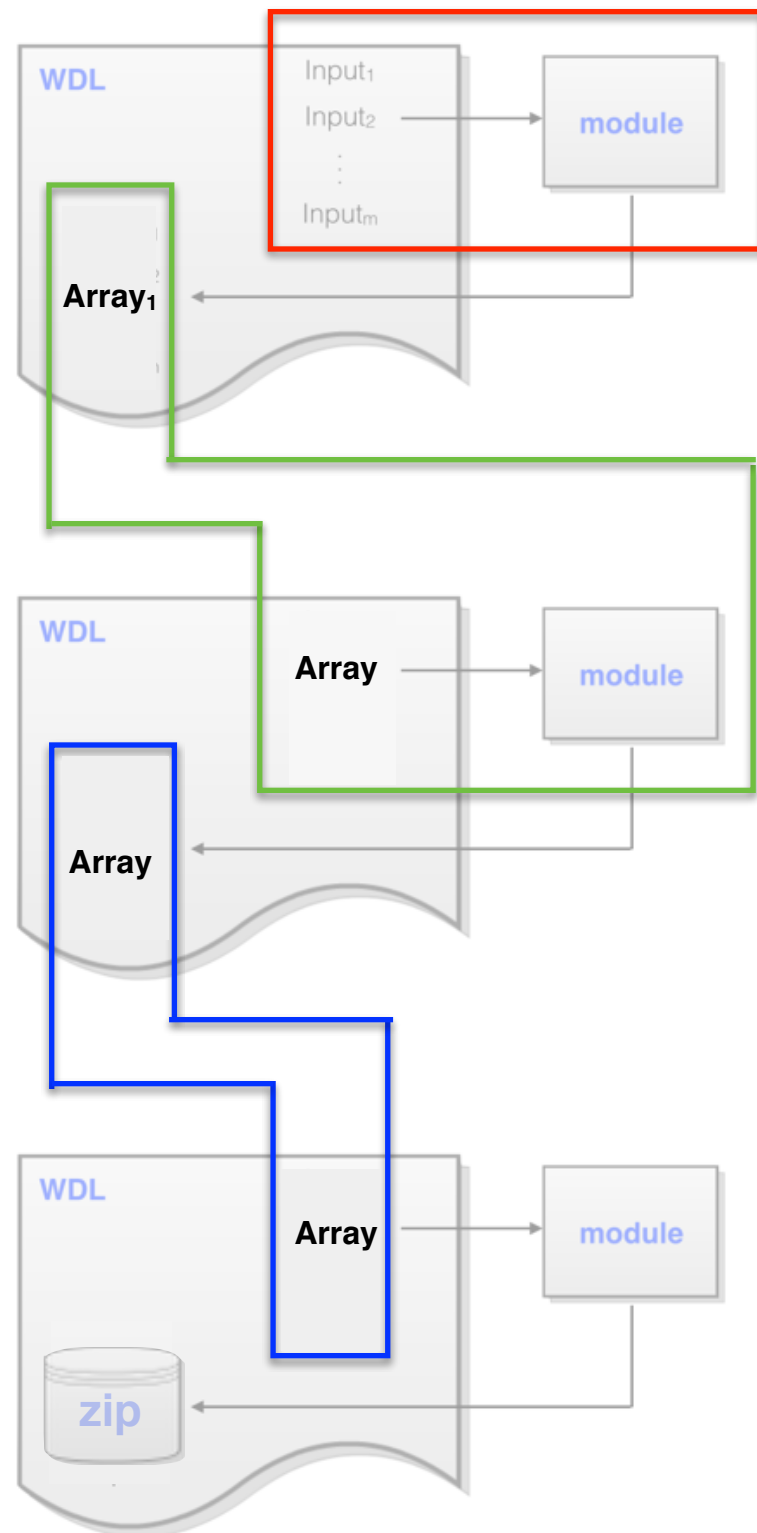
Boring ...

Expensive to write & maintain ...

With a different approach, much of it is not needed

Arrays for moving bolus of parameters thru pipeline

Outputs
globbed
into array



same as before

1 output argument in WDL1
1 input argument in WDL2
1 CLI arg of Module2
Module2 unpacks array in
given order

1 input arg to WDL3

Impact

Far less
verbose &
duplicative

Easier to
comprehend,
write & debug
end-to-end
workflow

Easier to change:
Often no WDL
or config change
when Task1
outputs change.
Usually only
Modules 1 & 2
src need change

What is this unconventional blasphemy?
Aren't we supposed to name parameters?

Why? There is a VERY long history of positional arg passing

We are all familiar with it, e.g. Python file open:

```
open(name[, mode[, buffering]])
```

What's being suggested here is essentially same, but:

- without parentheses
- and across scopes (module -> WDL -> module)
- instead of within a module scope

Moreover: the parameters ARE effectively named

Output analysis modules DO name them (e.g. [all_lesions.txt](#))

Input report can look for those names in array

Semantically this is no less of a name coupling than

Task1 WDL: Output lesions_file = “all_lesions.txt”

Report 2 WDL: Input lesions_file;

Report.R: -I \${lesions_file}

...

It's no more “hard coded” ... AND .. it achieves the same
“Parameter Migration” effect WITH MUCH LESS boilerplate

There seems no need to duplicate name as the params
pass through scopes: WDL-> module -> WDL -> module ...

Here, the greater goods seem to be:

- greater programmer efficiency
- less programmer boredom
- less workflow boilerplate
- greater codebase maintainability

Explicitly naming parameters while they pass through scopes seems to be a needless & costly safety blanket
(at least for the common GDAC use case)

Related Points

- Helps if we largely adopt “package everything” philosophy
Example: `zip *.*`
(which seems to be right approach, anyway for reproducibility)
- It's OK to have SOME duplication: e.g.
 - ✓ identifying params that can be used in OTHER workflows
 - ✓ (i.e. not just by report in the originating workflow)
 - ✓ by writing to persistent attributes in the data model
- But that, too, is very contained:
 - ✓ only output section of originating task WDL need change
 - ✓ not everything downstream

How can these arrays be instantiated? *

From WDL variable definitions:

```
Array[String] outputs = ["$plot", "$table", "$document"]
```

Single-file glob:

```
Array[String] outputs = glob("*.png")
```

Multi-file glob (untested):

```
Array[String] outputs = [ glob("*.png"), glob("*.txt"), glob("*.tsv")]
```

Better yet: see these 2 RFEs for WDL

<https://github.com/broadinstitute/wdl/issues/89>

<https://github.com/broadinstitute/wdl/issues/90>

Another Way: use tar/zip files
+ manifest to pass params

Reference implementation exploring this approach
is given in gdac-firecloud smoketest workflow

<https://github.com/broadinstitute/gdac-firecloud/tree/master/workflows/smoketest>

Which encapsulates these 2 tasks

https://github.com/broadinstitute/gdac-firecloud/tree/master/tasks/smoketest_task
https://github.com/broadinstitute/gdac-firecloud/tree/master/tasks/smoketest_report

In principle *zip* *.* can be automatically added to each
composite workflow WDL during “make sync”

Other (unfinished) Thoughts

Task1:

```
Array[String] outputs = [...]
```

Task2:

```
Array[String] outputs = [...]
```

Task3 (packaging):

```
String archive_name = ...
```

```
command {
```

```
    unix%  zip ${archive_name} task1.outputs[] + task2.outputs[]
```

```
    ...
```

```
}
```

This looks appealing simple, but has leads to other concerns:

- packaging implemented distinctly for each workflow
- instead of enabling `% make SYNC` to include it automatically within each