Models for model integration

SECOND ANNUAL CROPS IN SILICO SYMPOSIUM AND WORKSHOP

Oxford, UK, June 27, 2017

Daniel S. Katz

Assistant Director for Scientific Software & Applications, NCSA

Research Associate Professor, CS

Research Associate Professor, ECE

Research Associate Professor, iSchool

dskatz@illinois.edu, d.katz@ieee.org, @danielskatz

(slides available at doi: 10.6084/m9.figshare.5146921)

NCSA

National Center for Supercomputing Applications
University of Illinois at Urbana–Champaign

Multiscale

- The world has multiple scales
- In modeling, a common challenge is determining the correct scale to capture a phenomenon of interest
  - In computer science, a parallel problem is describing a problem with the right level of abstraction
    - Capture the details you care about and ignore those you don't
- But multiple phenomena interact, often at different scales
- We often know how to solve a part of the problem with sufficient accuracy, but when we combine multiple parts of the problem at various scales, we need to couple the solution methods too

Questions about each model

- What are the key coordinates?
  - Spatial, temporal, other
- What's a characteristic scale?
  - O(cm), O(minute), O(nucleotide base), O(temperature)
- How are the scales related?
  - Overlapping, separated, contiguous
- What are the inputs and outputs?
- Does the model have internal state? Or side effects?
- Dynamic or steady-state?

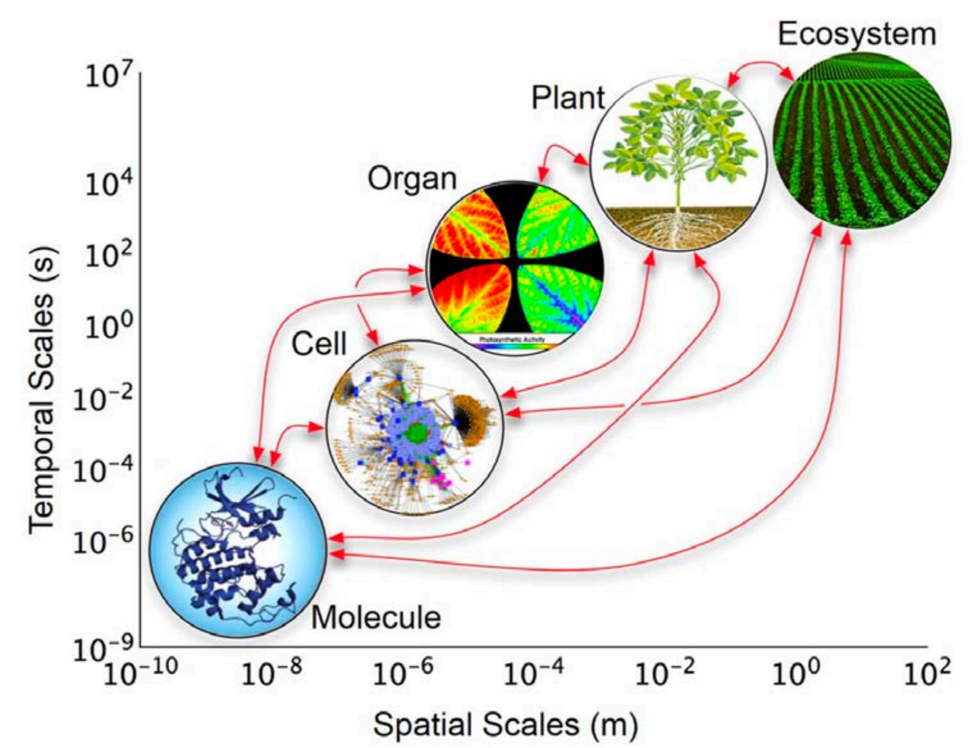


FIGURE 1 | Layers of organization of biological models across temporal and spatial scales. The *y*-axis represents real-time in which changes occur at each biological level; the *x*-axis represents the relative size or space which the biological level encompasses. The arrows indicate possible direct interactions among scales. Organ level image is from Kim et al. (2001).

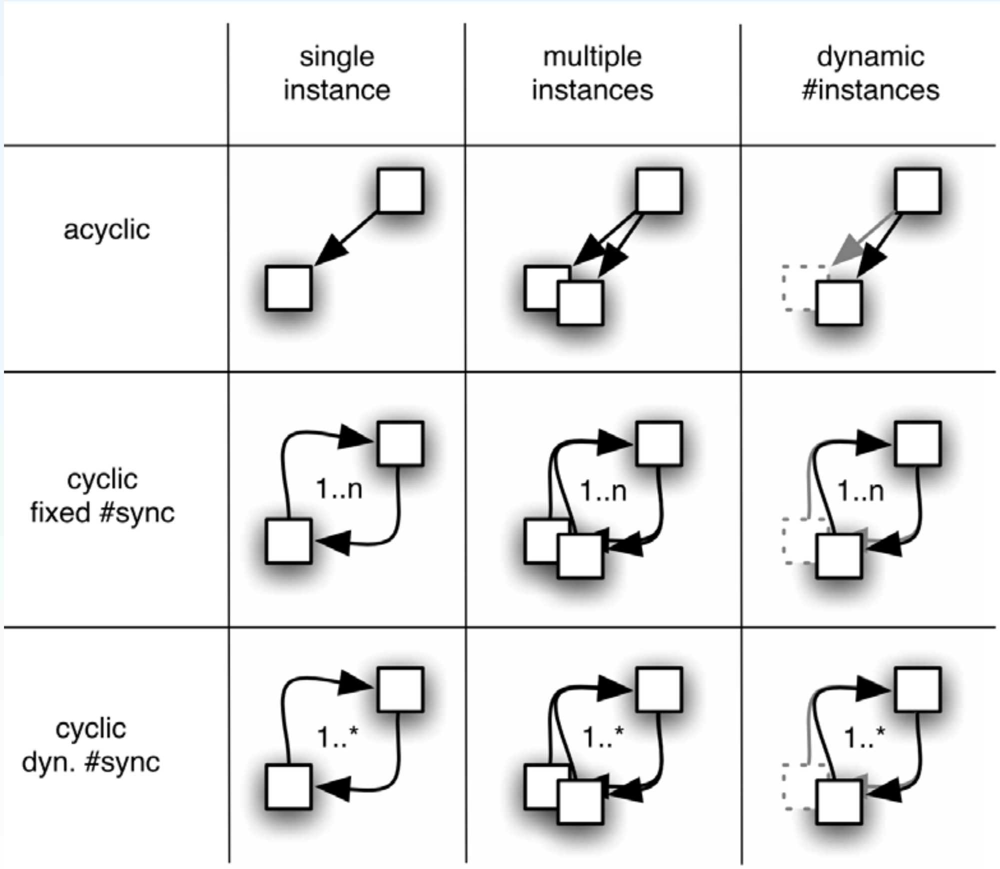
A. Marshall-Colon, S. P. Long, D. K. Allen, et al. "Crops In Silico: Generating Virtual Crops Using an Integrative and Multi-scale Modeling Platform," *Frontiers in Plant Science*, v.8, page 786, 2017. doi: 10.3389/fpls.2017.00786

Coupling methods

- Determine the models to run & how they iterate/interact
- Coupling options (ordering, automation, timescale)
  - “Manual” coupling (sequential, manual, days)
    - Inputs to a code at one scale are influenced by study of the outputs of a previously run code at another scale
  - “Loose” coupling (sequential, automated, minutes)
    - Typically performed using workflow tools
  - “Tight” coupling (concurrent, automated, seconds)
    - Typically performed using framework, maybe in single memory space
- Boundary between options can be fuzzy
- Choice often depends on how frequently the interactions are required, and how much work the codes do independently

A model for model coupling

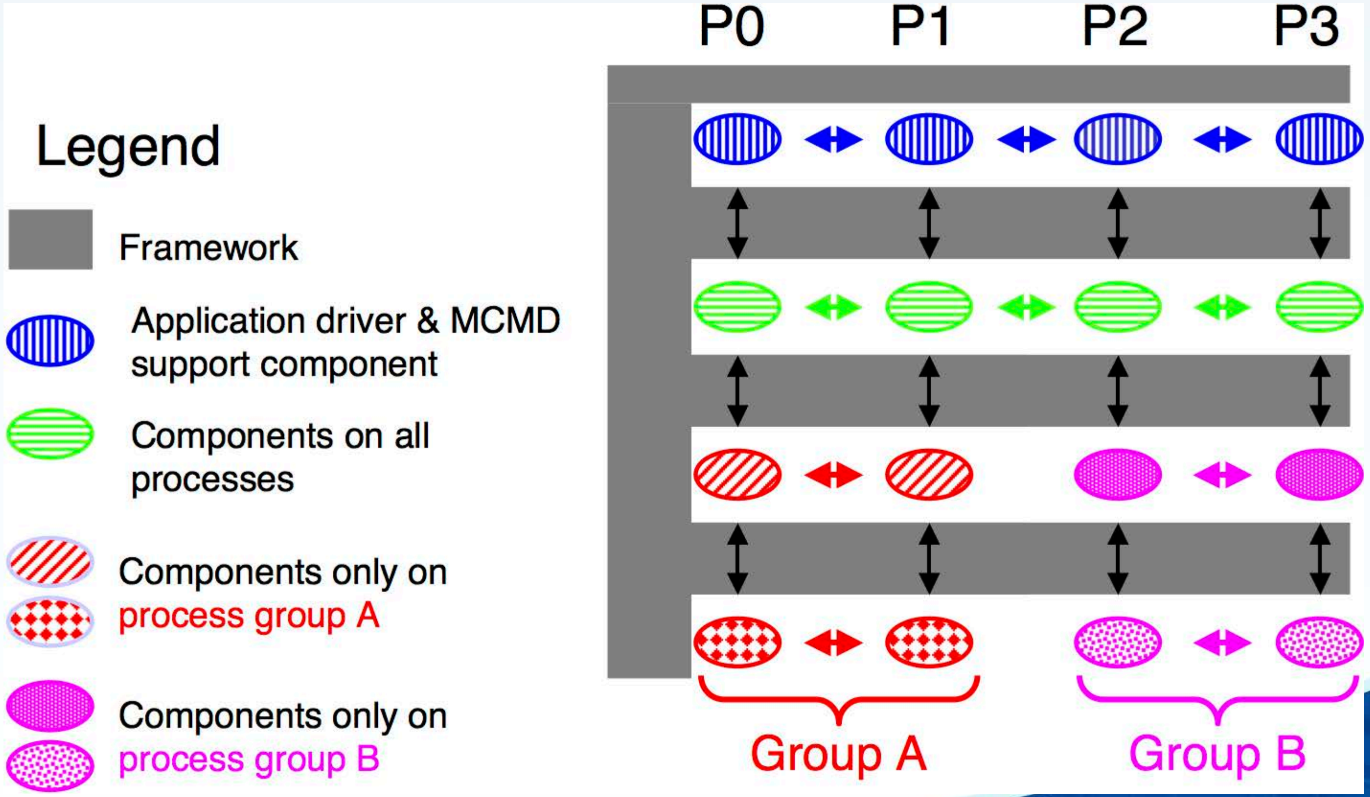
- Is the coupling topology cyclic or acyclic, or does only parts contain cycles?
- Are there multiple instances of certain models, and if so, can the number be statically determined?
- Can the number of synchronization points be statically determined?



J. Borgdorff, C. Bona-Casas, M. Mamonski, et al., A Distributed Multiscale Computation of a Tightly Coupled Model Using the Multiscale Modeling Language, *Procedia Computer Science*, v. 9, 2012. doi: 10.1016/j.procs.2012.04.064

Interaction with infrastructure

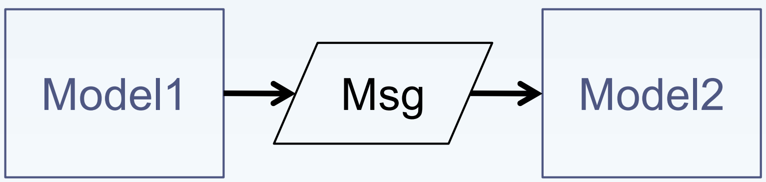
- Single system: laptop, cluster, cloud (single remote cluster)
- Distributed system: clouds
- Which (how many) memory space(s)
- Coordination: framework, script/glue
- Communication: internal (eg MPI), files, messages
- Control: in/run/out, in/run/…/run/out, in/run/out/in/run/out/in/run/out/…



D. E. Bernholdt, B. A. Allan, R. Armstrong, et al., "A Component Architecture for High-Performance Scientific Computing," *International Journal of High Performance Computing Applications*, v. 20(2), Summer 2006. doi: 10.1177/1094342006064488

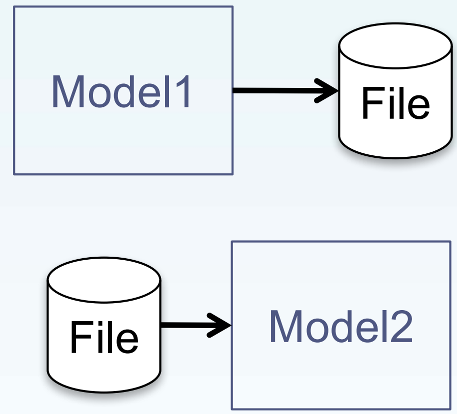
More on coupling

Tight



Implement via MPI, framework, etc.
(e.g., PDEs in a single memory space)

Loose

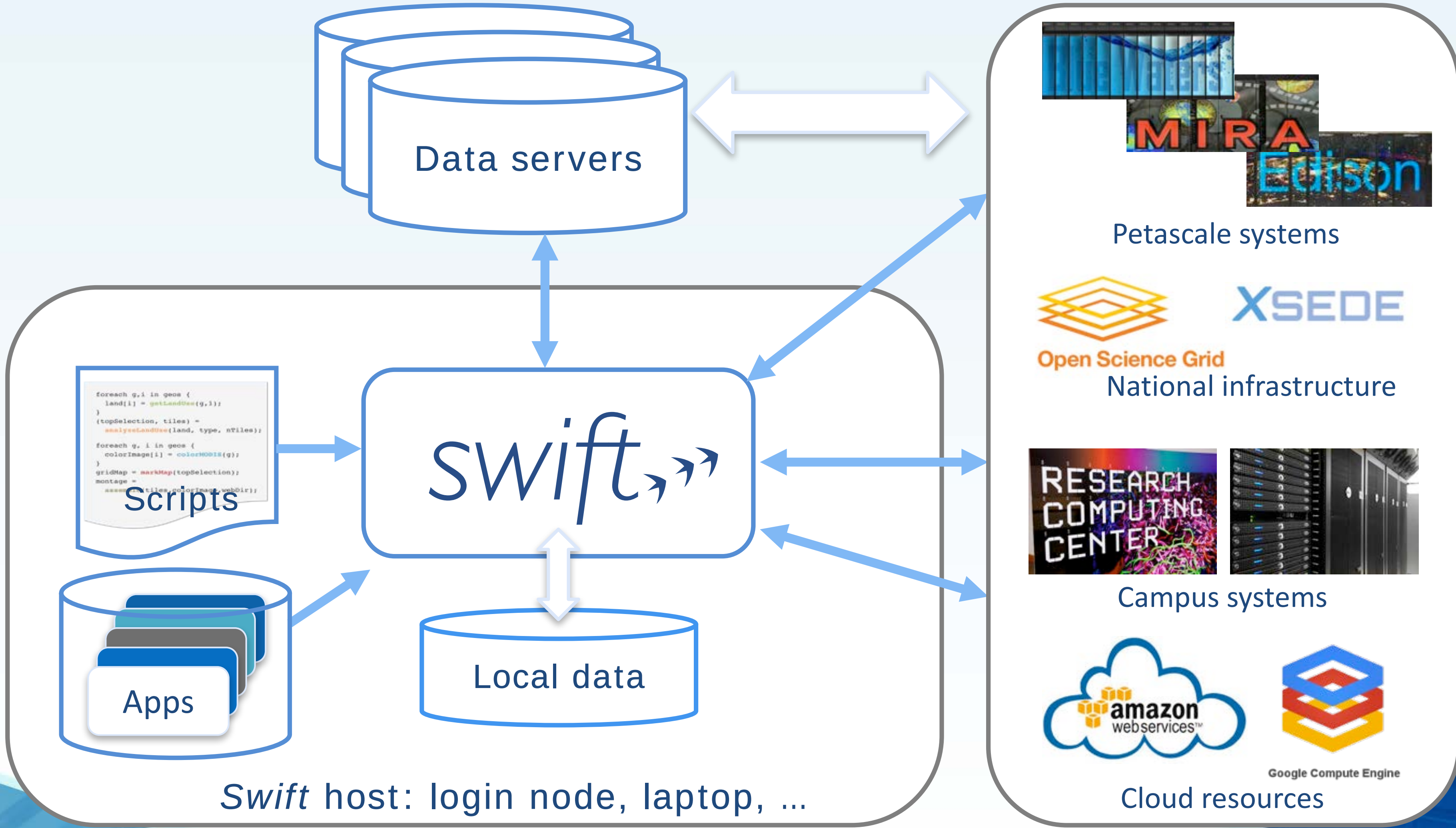


Implement via workflow system

Swift

- A C-like workflow language for programming the interaction of models (in/run/out)
  - External processes that communicate via files
  - Functions that communicate via variables
  - Sequential or parallel
- A runtime that supports portable workflows – deployable on many resources (clusters, HPC, clouds)
- Provides natural concurrency at runtime through automatic data flow analysis and task scheduling
- Data structures and script operations to support scientific computing
- Provenance gathered automatically
- http://swift-lang.org/

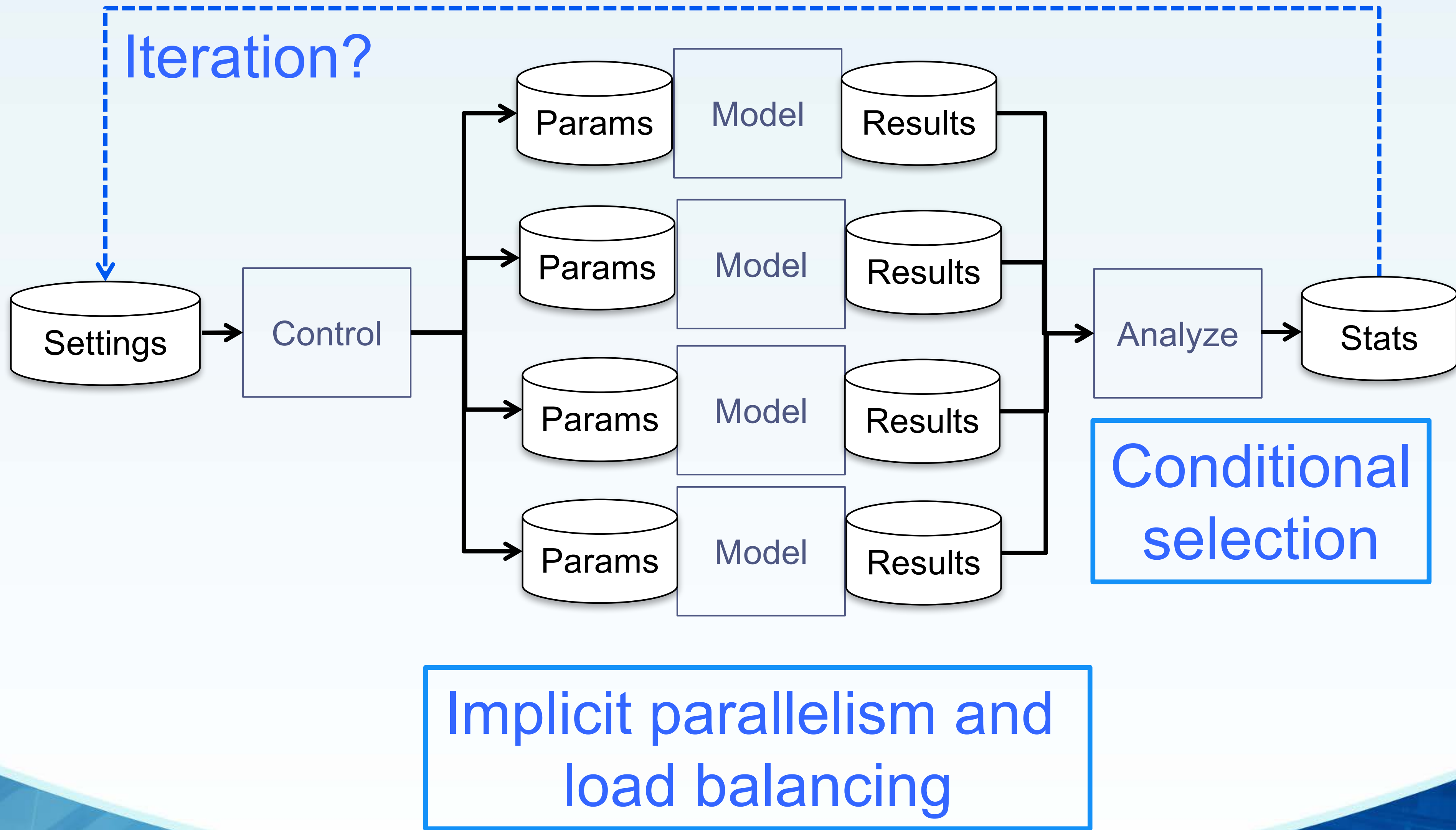
Swift enables execution of simulation campaigns across multiple HPC and cloud resources



Swift model

- Variables are single assignment futures
  - Variables that be “used” before they are filled/closed
  - Unassigned variables are empty/open
- Variables can represent files
  - When a file doesn’t exist, the variable is open
  - When a file exists, the variable is closed
- All initial tasks found at runtime
  - Additional tasks can be created during run
- Tasks with satisfied dependencies (closed variables) are run on whatever resources are available
- These create files/variables that allow more tasks to run

Swift concurrency and complexity



Parsl

- Python-based implementation of the Swift concept
  - A fully parallel scripting library
- Tasks can be models (in/run/out) or (python) functions that communicate via files or data objects
- Easy to run: on clusters, clouds and grids
- Sends work to disparate resource providers
- Fast: launches thousands of tasks per second
- Under active development
- http://parsl-project.org

Simple Parsl example

```
# Import Parsl
import parsl
from parsl import *

# Create a pool of threads to execute functions
workers = ThreadPoolExecutor(max_workers=4)
# Pass workers to the DataFlowKernel, which will execute Apps over them
dfk = DataFlowKernel(workers)

@App('python', dfk)
def pi(total):
    # function that creates total random points in 1x1 box and returns the
number that fall in a circle inside that box
    return (number)

@App('python', dfk)
def avg_three(a,b,c):
    return (a+b+c)/3
```

Simple Parsl example, cont.

```
a, b, c = pi(10**6), pi(10**6), pi(10**6)
# returns immediately, with a, b, c futures

avg_pi  = avg_three(a, b, c)
# returns immediately, with avg_pi future
# once a, b, c are calculated, this will start running

# Print the results
print("A: {0:.5f} B: {1:.5f} C: {2:.5f}".format(a.result(), b.result()
, c.result()))
# blocks until a, b, c are calculated

print("Average: {0:.5f}".format(avg_pi.result()))
# blocks until avg_pi is calculated
```

Some reading

- D. E. Bernholdt, B. A. Allan, R. Armstrong, et al., "A Component Architecture for High-Performance Scientific Computing," *International Journal of High Performance Computing Applications*, v. 20(2), Summer 2006. doi: 10.1177/1094342006064488
- D. Groen, S. J. Zasada and P. V. Coveney, "Taxonomy of Multiscale Computing Communities," *2011 IEEE Seventh International Conference on e-Science Workshops*, 2011. doi: 10.1109/eScienceW.2011.11
- J. Borgdorff, C. Bona-Casas, M. Mamonski, et al., A Distributed Multiscale Computation of a Tightly Coupled Model Using the Multiscale Modeling Language, *Procedia Computer Science*, v. 9, 2012. doi: 10.1016/j.procs.2012.04.064
- A. Marshall-Colon, S. P. Long, D. K. Allen, et al. "Crops In Silico: Generating Virtual Crops Using an Integrative and Multi-scale Modeling Platform," *Frontiers in Plant Science*, v.8, page 786, 2017. doi: 10.3389/fpls.2017.00786
- M. Wilde, N. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, I. Foster, "Swift: A language for distributed parallel scripting," Parallel Computing, v.37(9), pp. 633-652, 2011. doi: 10.1016/j.parco.2011.05.005