# Introducing PRECIP – Pegasus Repeatable Experiments for the Cloud in Python

Sepideh Azaroosh, Mats Rynge, Gideon Juve, Ewa Deelman
University of Southern California's Information Sciences Institute

## Overview

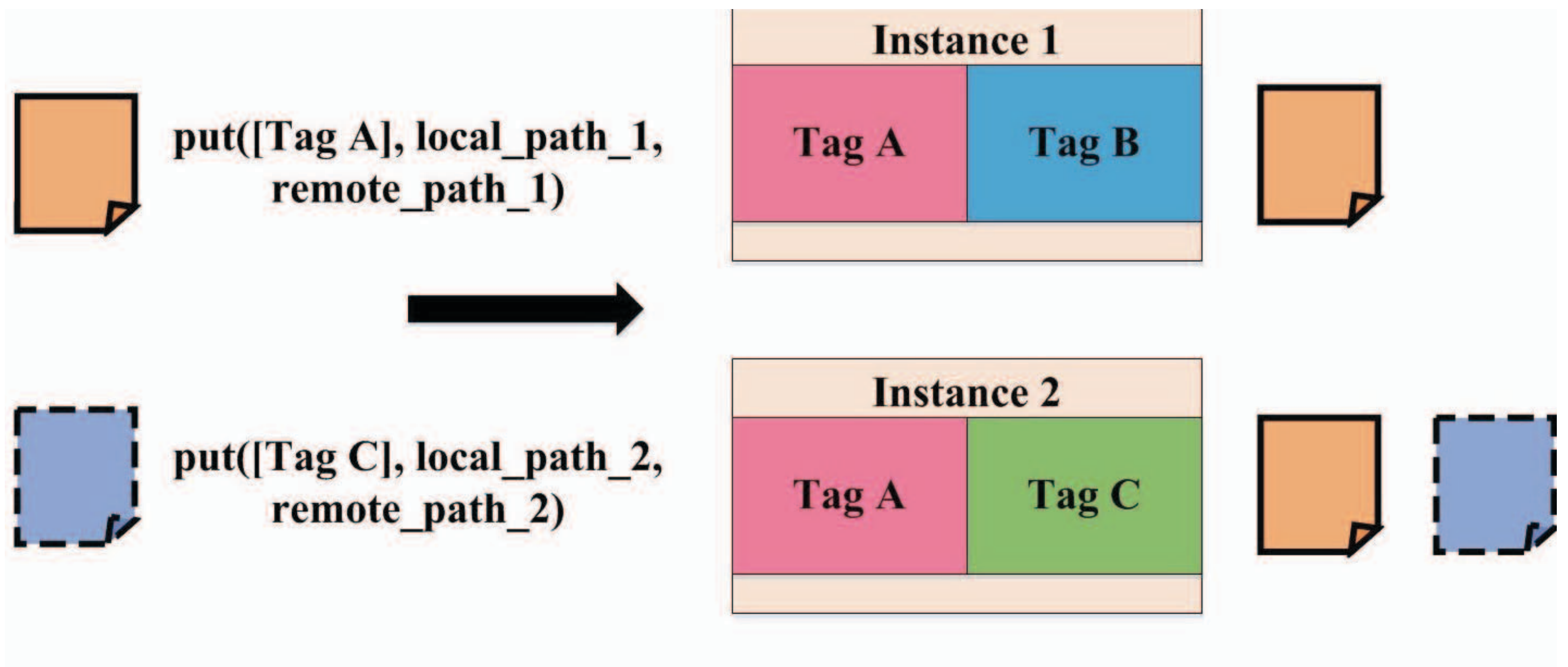
PRECIP, Pegasus Repeatable Experiments for the Cloud in Python, is a new API which manages scientific experiments on the FutureGrid test bed. PRECIP was developed to be used on academic cloud infrastructures such as OpenStack, as well as commercial ones like Amazon Web Services. PRECIP provides a solid and easy to use API for end user computer science but can also be used as an exploration playground for evaluating cloud systems. The goal of the API is to be flexible and simple to use in Python to control your experiments, and provide the base for repeatable, shareable and peer reviewable experiments.

## Benefits of PRECIP

- Cloud interoperability. The scientist only needs to learn the API to run or move experiments across multiple clouds.
- Repeatability. The scientists can run the same experiment for evaluation and peer reviewing purposes over and over again with reproducible results.
- Automatic logging. PRECIP steps are time stamped and fully recorded.
- Instance tagging. Tags are used to handle experiment resources.
- Basic managing of provisioning, SSH keys, and security groups in a fault tolerance manner.
- PRECIP will clean all leftovers after itself at the end of each experiment.

## Instance Tagging

- A flexible and powerful handle for addressing a large number of instances.
- Arbitrary tags can be attached to an instance during its provisioning phase
- Each instance can have an arbitrary number of tags.
- Tags are global and available during the whole experiment.
- Tags are used to identify, manipulate and interact with instances.


put([Tag A], local_path_1, remote_path_1)
Instance 1
Tag A
Tag B
put([Tag C], local_path_2, remote_path_2)
Instance 2
Tag A
Tag C

Uploading files to subset of instances identified by tags

## Example

This example illustrates how to use PRECIP to test the performance of an application. In this case, PRECIP sets up a HTCondor pool and executes a workflow.

```
#!/usr/bin/python

import os
import time

from precip import *

try:

    exp = OpenStackExperiment(
            os.environ['OPENSTACK_URL'],
            os.environ['OPENSTACK_ACCESS_KEY'],
            os.environ['OPENSTACK_SECRET_KEY'])

    exp.provision("ami-0000004c", tags=["master"], instance_type="m1.large")
    exp.provision("ami-0000004c", tags=["compute"], instance_type="m1.large", count=2)

    exp.wait()

    master_priv_addr = exp.get_private_hostnames(["master"])[0]

    exp.copy_and_run(["master"], "./bootstrap.sh")
    exp.copy_and_run(["compute"], "./bootstrap.sh", args=[master_priv_addr])

    time.sleep(60)

    exp.run(["master"], "condor_status")

    exp.run(["master"], "cd ~/montage && ./run-montage", user="wf")

finally:
    exp.deprovision([])
```

Keys and endpoints setup

Provisioning of master node and a set of workers

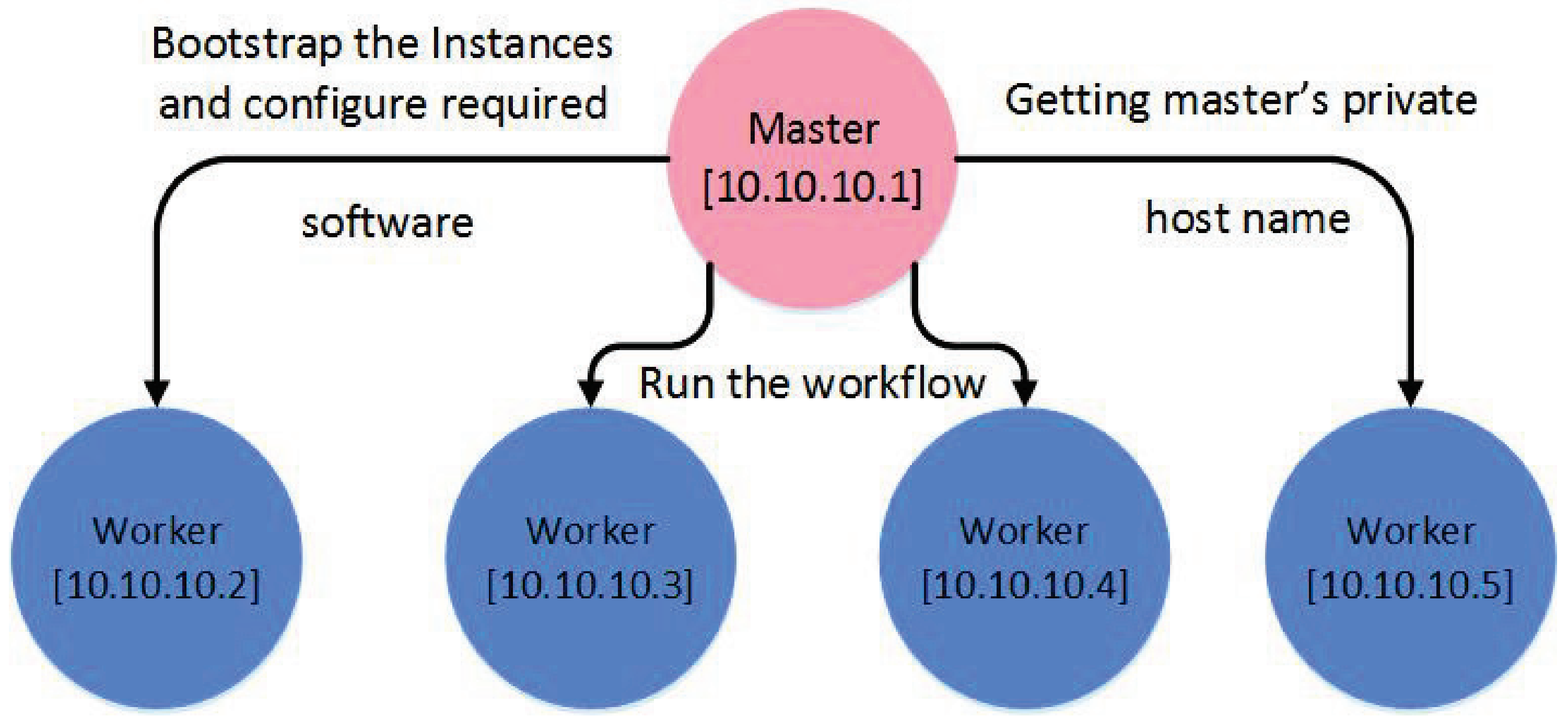
Wait for all instances to boot and become accessible – this acts like a barrier for the experiment to start

The workers get the private host name of the master

Installing HTCondor and Pegasus

Run the Workflow

Cleaning all the leftovers at the end


Bootstrap the Instances and configure required software

Getting master's private host name

Master [10.10.10.1]

Run the workflow

Worker [10.10.10.2]

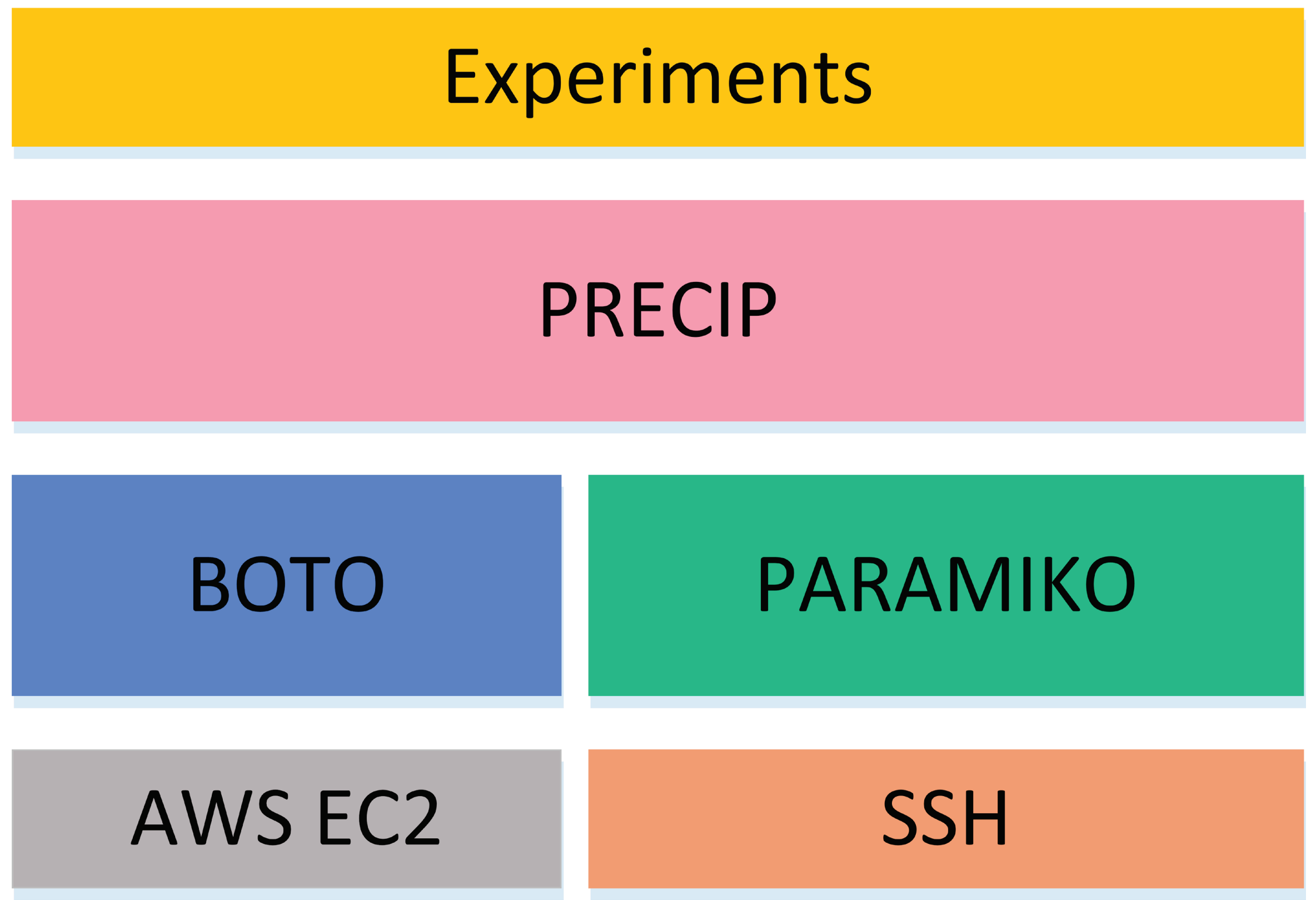
Worker [10.10.10.3]

Worker [10.10.10.4]

Worker [10.10.10.5]

The resulting instances are tagged and used to run the experiment

## Architecture


Experiments

PRECIP

BOTO

PARAMIKO

AWS EC2

SSH

Layered Architecture of PRECIP. This figure depicts that PRECIP is using Boto and Paramiko libraries for EC2 access and SSH functionality.

## Virtual Machine Images

- Users can use their own virtual machine images (VMI) that may have special software stacks or custom kernel.
- VMIs do not require to have any PRECIP-specific software installed.
- Experiments can use basic Unix-like images or use the API to run bootstrap scripts on the VMIs to install or configure the required software for more complex experiments.

## Future Work

- Help FutureGrid projects with their experiments
- Provide a set of modules for common instance setups, for example: Hadoop, shared filesystem, HTCondor pool, …
- Integration with logging frameworks such as NetLogger

## Acknowledgements

This material is based upon work supported in part by the National Science Foundation under Grant No. 0910812 to Indiana University for "FutureGrid: An Experimental, High-Performance Grid Test-bed." Partners in the FutureGrid project include U. Chicago, U. Florida, San Diego Supercomputer Center - UC San Diego, U. Southern California, U. Texas at Austin, U. Tennessee at Knoxville, U. of Virginia, Purdue U., and T-U. Dresden.

**http://pegasus.isi.edu/precip**