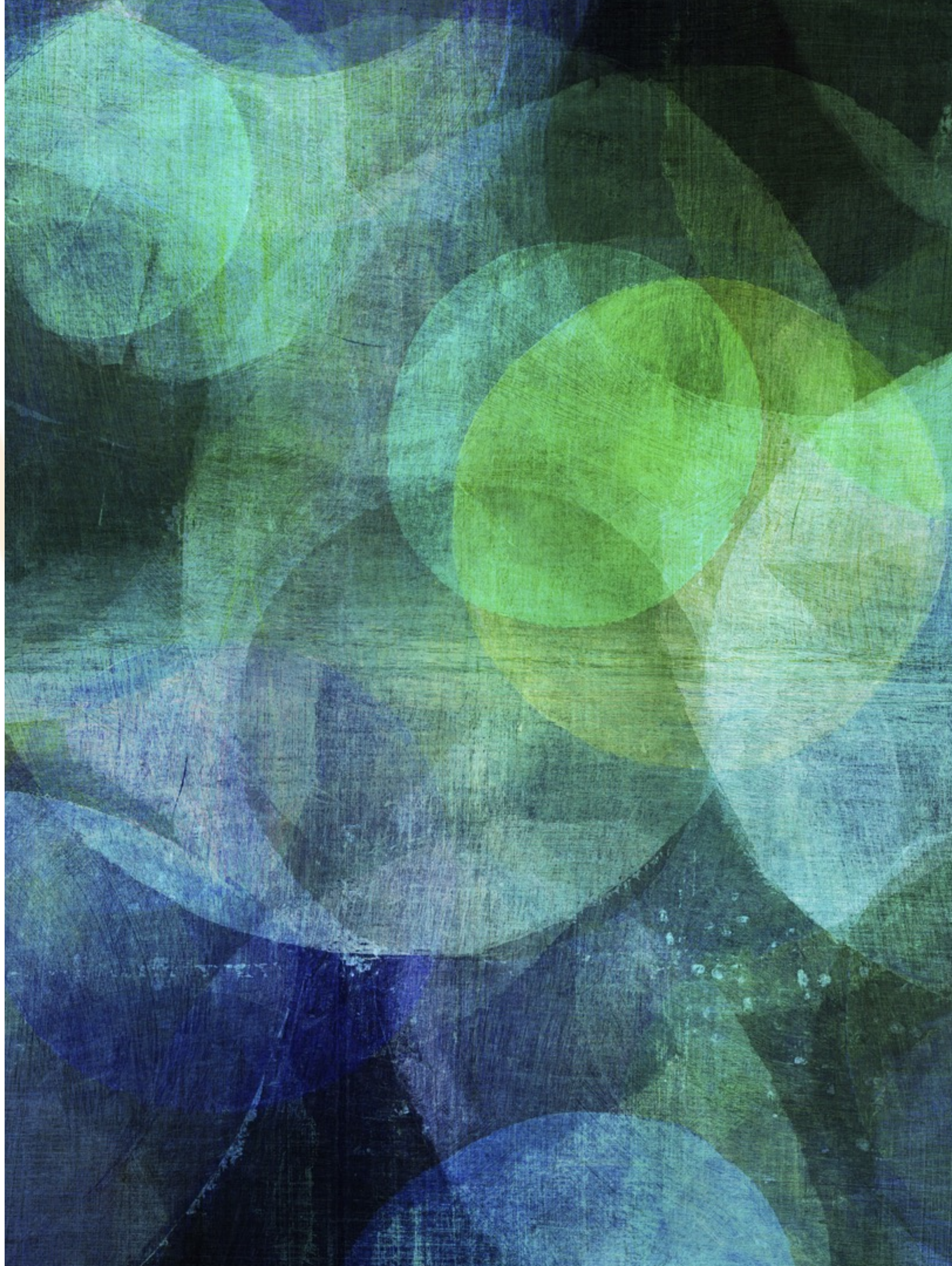
AWS DATA PROCESSING INFRASTRUCTURE 3C

Nan Dun
nan.dun@acm.org



DISCLAIMER

I. *"All data, information, and opinions expressed in this presentation is for informational purposes only.   I do not guarantee the accuracy or reliability of the information provided herein. This is a personal presentation. The opinions expressed here represent my own and not those of my employer."*

II. *"The copyright of photos, icons, charts, trademarks presented here belong to their authors."*

III. *"I could be wrong."*

OUTLINE

- Terraform Debugging
- Ansible Debugging
- Spot Instances self-tagging
- Python Multiprocessing

TERRAFORM DEBUGGING

- Enable Debug info
  - TF_LOG = TRACE | DEBUG | INFO | WARN | ERROR
- Look at crash.log when Terraform crashes

ANSIBLE DEBUGGING

- Add “strategy: debug”
- Debug command
  - p task/vars/host/result
  - task.args[key] = value
  - vars[key] = value
  - r(edo)
  - c(ontinue)
  - q(uit)

```
- hosts: test
   strategy: debug
  tasks:
  ...
```

More information: https://docs.ansible.com/ansible/playbooks_debugger.html

SPOT INSTANCE SELF-TAGGING

- Instance Metadata (execute following command on a running instance!)
  - $ curl http://169.254.169.254/
  - $ curl http://169.254.169.254/latest/meta-data/
  - $ curl http://169.254.169.254/latest/meta-data/ami-id
  - $ curl http://169.254.169.254/latest/user-data/instance-id
- Call "aws ec2 create-tags …" to tag instance itself
  - However, not scalable…
    - If hundreds of instances query simultaneously, API request limit will be exceeded
- Put them all in userdata

More information: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html

PYTHON MULTIPROCESSING: PASSING ARGUMENTS

```
import multiprocessing

def worker(num):
    """thread worker function"""
    print 'Worker:', num
    return

if __name__ == '__main__':
    jobs = []
    for i in range(5):
        p = multiprocessing.Process(target=worker, args=(i,))
        jobs.append(p)
        p.start()
```

More information: https://pymotw.com/2/multiprocessing/index.html

PYTHON MULTIPROCESSING: MAP

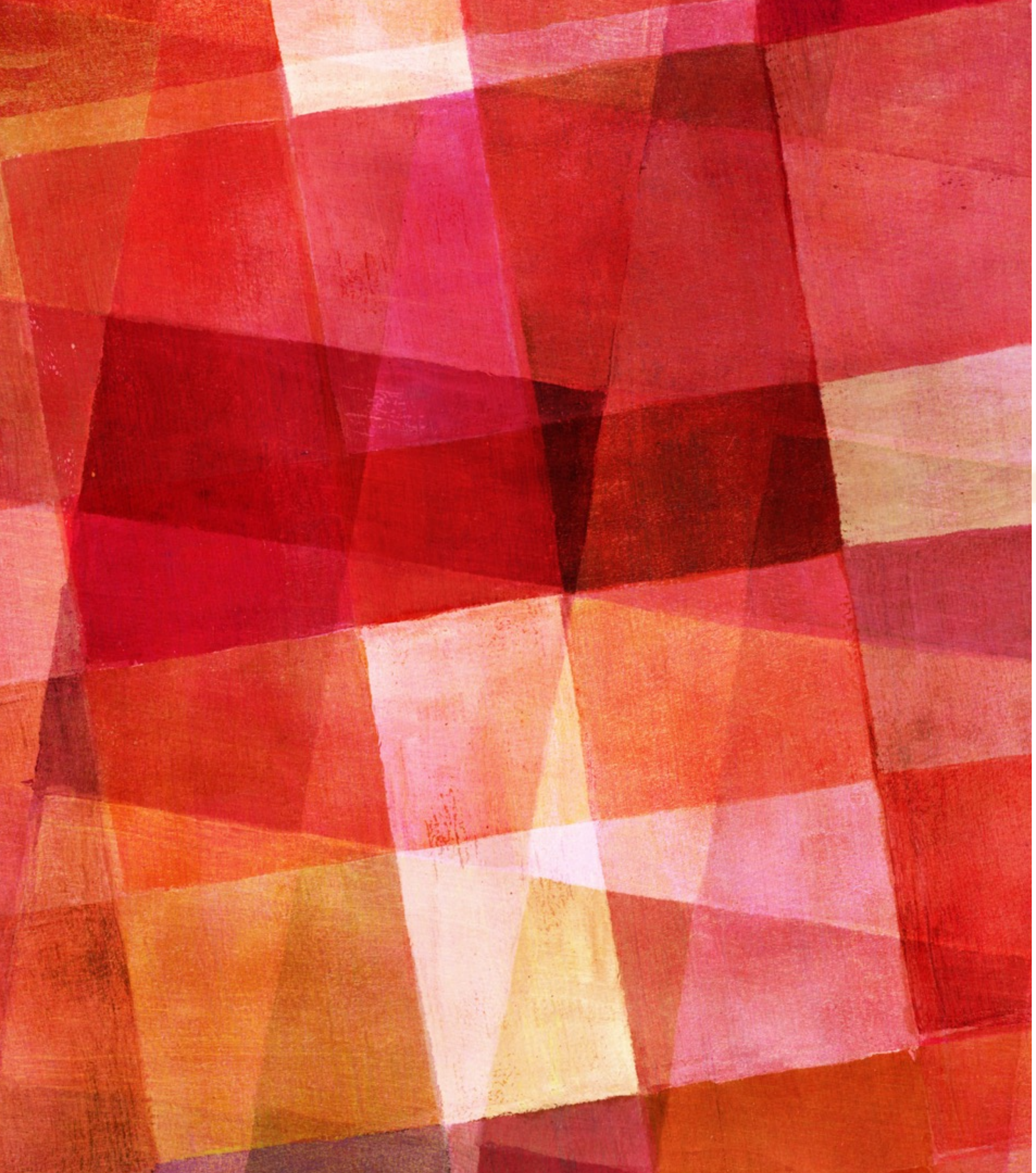
```
import multiprocessing

def do_calculation(data):
    return data * 2

def start_process():
    print 'Starting', multiprocessing.current_process().name

if __name__ == '__main__':
    inputs = list(range(10))
    print 'Input   :', inputs

    builtin_outputs = map(do_calculation, inputs)
    print 'Built-in:', builtin_outputs

    pool_size = multiprocessing.cpu_count() * 2
    pool = multiprocessing.Pool(processes=pool_size,
                                initializer=start_process)
    pool_outputs = pool.map(do_calculation, inputs)
    pool.close() # no more tasks
    pool.join()  # wrap up current tasks

    print 'Pool    :', pool_outputs
```

RANGE READ

- Data is already partitioned by color and date
- Data is not partitioned within one single object, and we don't have to do so, because
  - Our record length is 80
  - Range read will give us all we need

```
data = obj.get(Range='bytes=7920-8000')['Body']
```

Read the 100th record



QUESTIONS

- bittiger-aws@googlegroups.com