



Orchestrator: A post-mortem on an automated MMO testing framework

David Press

davidp@ccpgames.com



Who is CCP?

CARBON

- 600 person company.
- Working on 3 AAA games.
- Eve Online – 370k subscribers, 65k PCU
- Dust 514 – Upcoming FPS integrated with Eve.
- World of Darkness – Upcoming MMO.



What is Carbon?

CARBON

- Shared technology platform.
- Used in all 3 games.
- Developers of all 3 games work in the same branch.
 - 121 programmers
- Updated Carbon code is immediately used in all 3 games.



How do we manage this chaos?

CARBON

- Too much work to test all 3 projects in all configurations whenever Carbon code is changed.
- Automated testing
- Immediately tells us what broke.
 - How it broke.
 - Who broke it.
- View test history and logs from each test
 - Catch low probability bugs.
- Programmers can shelve CLs and get all automated tests to run on them before checking them in.



Types of Automated Testing

CARBON

- Unit Testing
- Component Testing
- System Testing



- Unit Testing
- Component Testing
- **System Testing**



- What makes testing MMOs unique?
- 2 demos of our framework, Orchestrator, in action.
- Architecture of Orchestrator.
- Lessons Learned



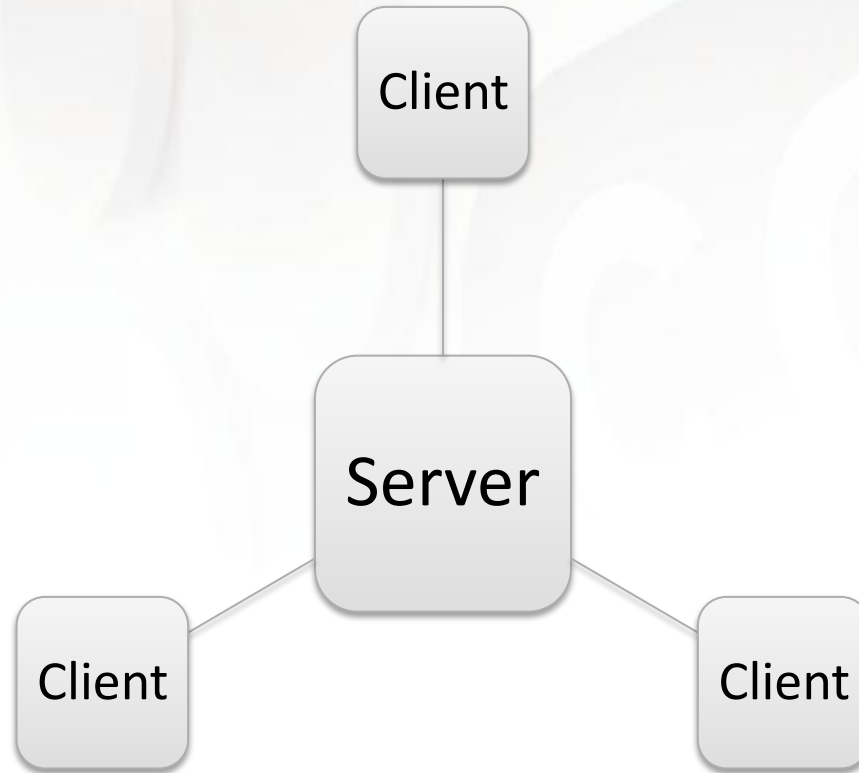
- What makes testing MMOs unique?
- 2 demos of our framework, Orchestrator, in action.
- Architecture of Orchestrator.
- Lessons Learned



- How do you automate a client-server, distributed, persistent, sharded, asynchronous, realtime, scalable system?
 - Very Carefully

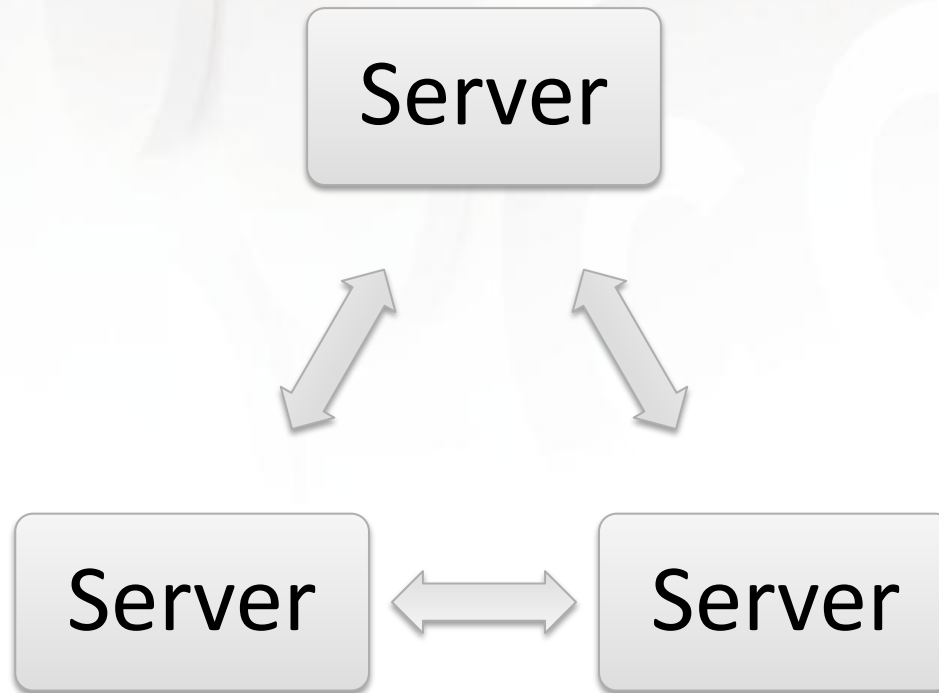


- Client/server



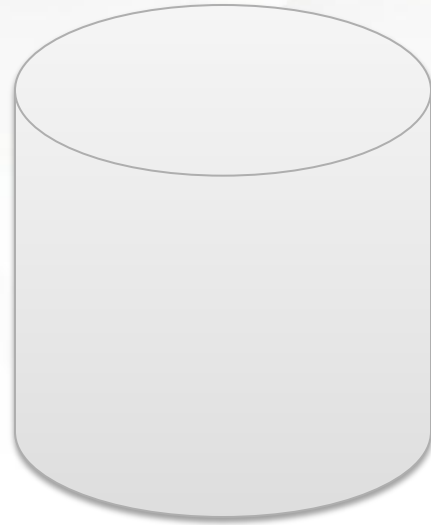


- Distributed system

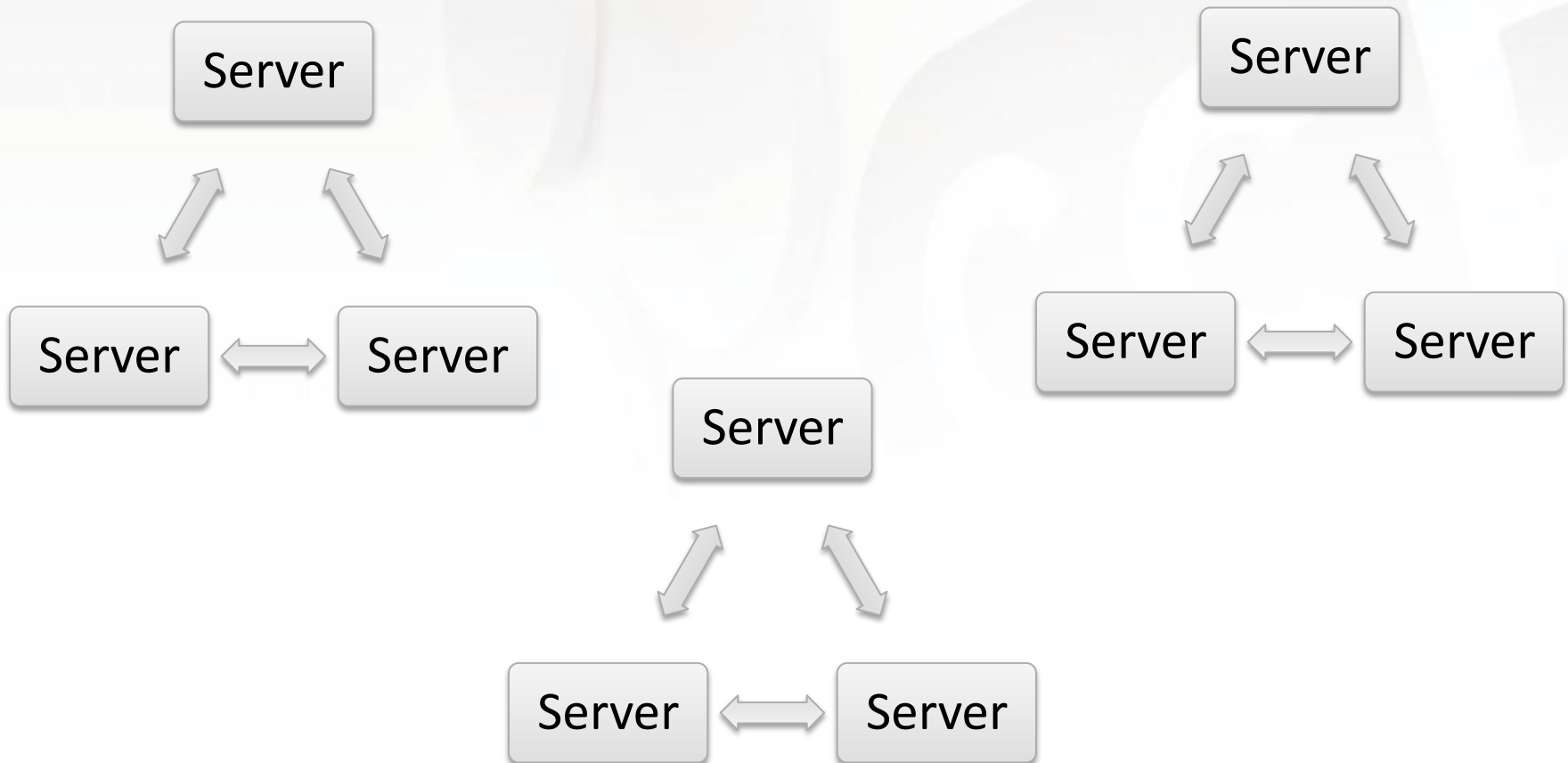




- Persistent Storage

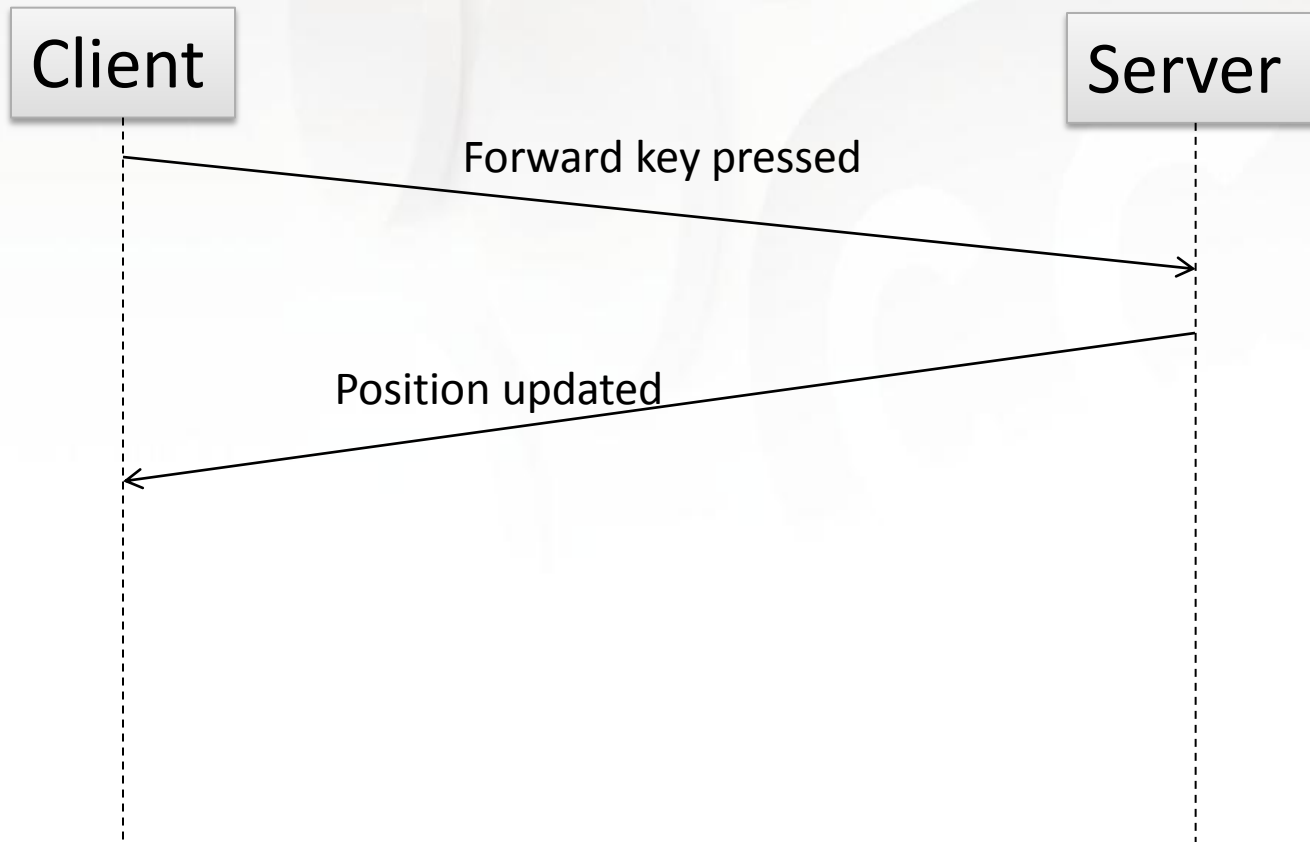


- Shards





- Asynchronous – Even harder than multithreaded.





- Realtime Simulation

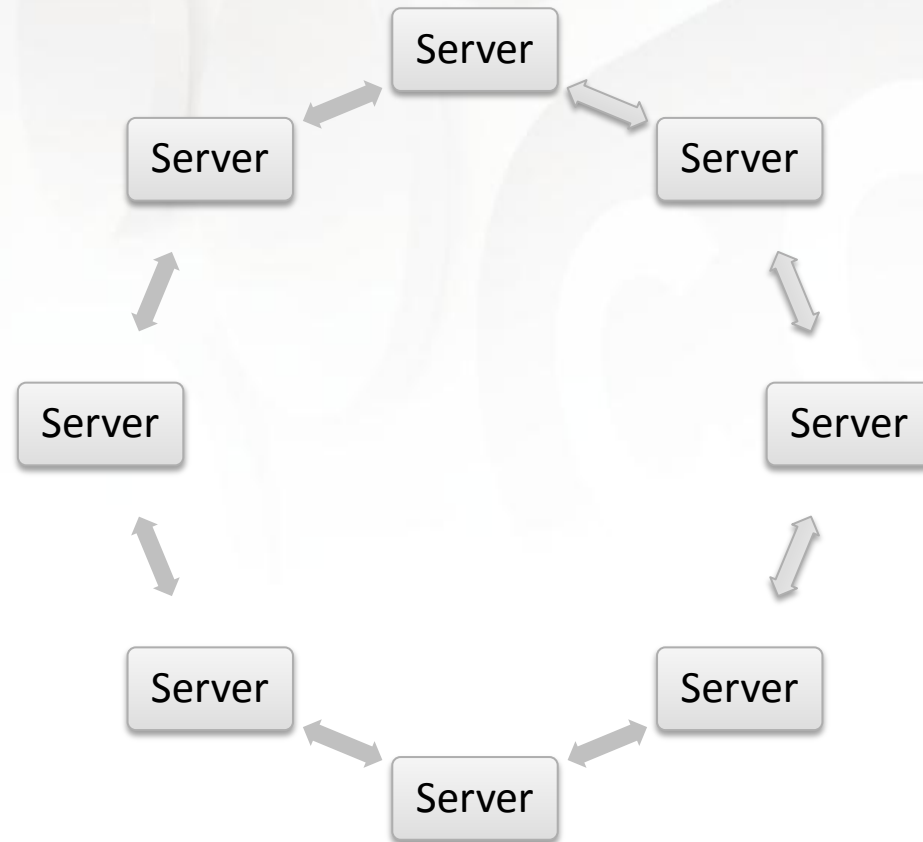




MMO Architecture Overview

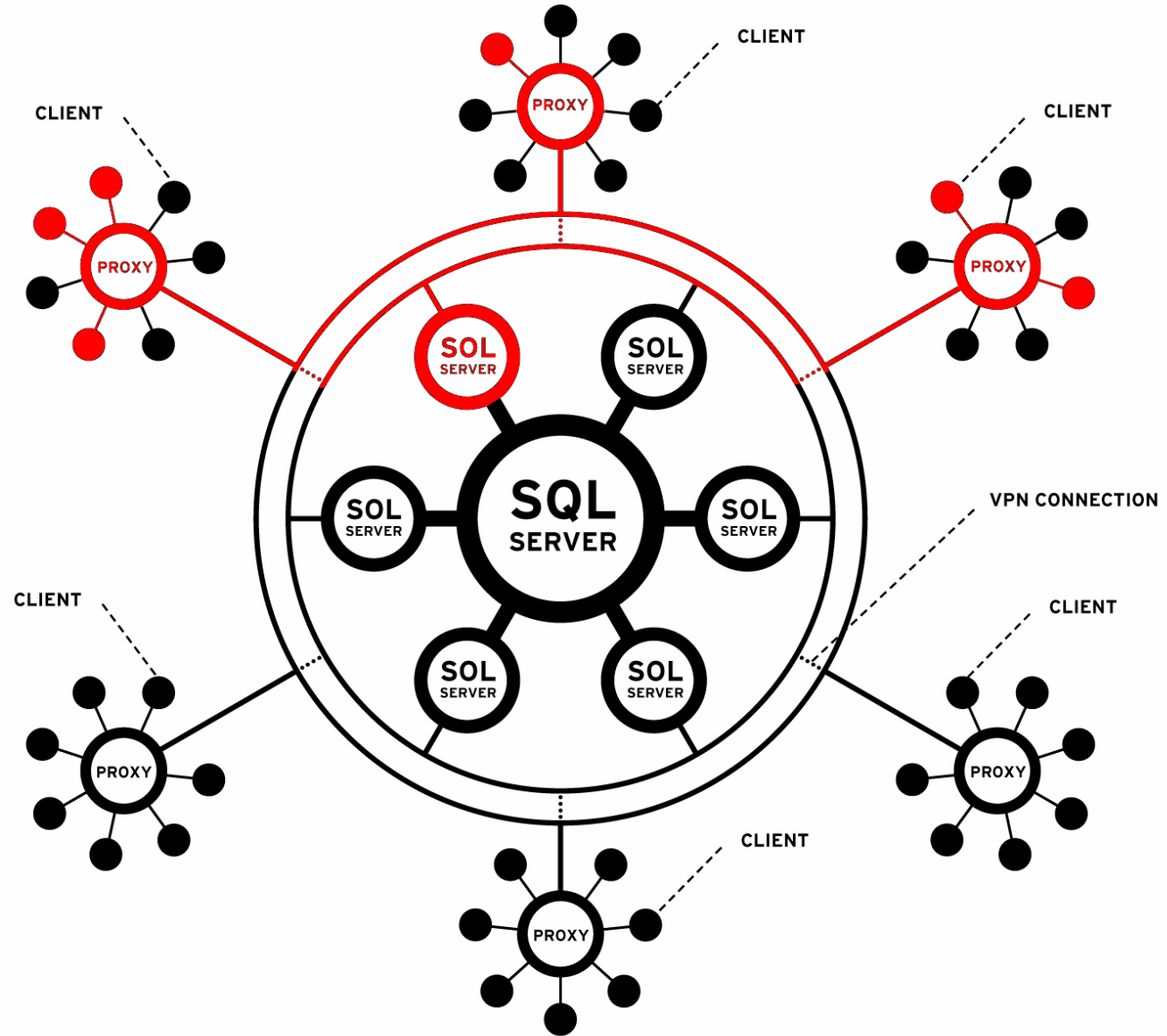
CARBON

- Scalable





CCP MMO Architecture





- What makes testing MMOs unique?
- **2 demos of our framework, Orchestrator, in action.**
- Architecture of Orchestrator.
- Lessons Learned



Demo 1

CARBON

- Networked movement
- 2 clients, 1 server, 1 proxy.
- Log both clients into the same worldspace.
- Move client 2's player a few meters.
- On client 1, check if client 2's player is at the same position as it is on client 2.



Demo 1

CARBON

[Demo](#)



Demo 1

- Two ways to write this test
- Write a script for each client, communicate between them to order their operations correctly.
 - Yuck.
- Write a single master script that communicates the relevant operations to the clients in sequence.
 - More familiar programming model.
 - Easier to read the code.



Code for Demo 1

```
class NetworkedMovementTests(systemTest.TestCase):
    __clients__ = ["client1", "client2"]


def setUp(self):
    systemTest.TestCase.setUp(self,
        waitForGraphics=True,
        worldSpaceID=TEST_WORLD_SPACE_ID)
```



Code for Demo 1

```
class NetworkedMovementTests(systemTest.TestCase):  
    __clients__ = ["client1", "client2"]  
  
def setUp(self):  
    systemTest.TestCase.setUp(self,  
        waitForGraphics=True,  
        worldSpaceID=TEST_WORLD_SPACE_ID)
```

Standard
jUnit
interface

A blue arrow points from the text 'Standard jUnit interface' to the 'systemTest.TestCase' part of the class definition in the code above.



Code for Demo 1

```
class NetworkedMovementTests(systemTest.TestCase):  
    __clients__ = ["client1", "client2"]  
  
def setUp(self):  
    systemTest.TestCase.setUp(self,  
        waitForGraphics=True,  
        worldSpaceID=TEST_WORLD_SPACE_ID)
```

Start two clients



Code for Demo 1

```
class NetworkedMovementTests(systemTest.TestCase):  
    __clients__ = ["client1", "client2"]  
  
def setUp(self):  
    ←————— Run for each test in this suite  
    systemTest.TestCase.setUp(self,  
        waitForGraphics=True,  
        worldSpaceID=TEST_WORLD_SPACE_ID)
```



```
class NetworkedMovementTests(systemTest.TestCase):  
    __clients__ = ["client1", "client2"]  
  
    def setUp(self):  
        systemTest.TestCase.setUp(self,  
            waitForGraphics=True,  
            worldSpaceID=TEST_WORLD_SPACE_ID)
```

Utility function to make server and clients log in to given worldspace and wait until all graphics are loaded



Code for Demo 1

CARBON

```
SystemTestUtils.TeleportPlayerTo(self.client1,  
                                  (0,0,0))
```

```
SystemTestUtils.TeleportPlayerTo(self.client2,  
                                  (2,0,0))
```



Code for Demo 1

CARBON

```
SystemTestUtils.TeleportPlayerTo(self.client1,  
                                  (0,0,0))  
SystemTestUtils.TeleportPlayerTo(self.client2,  
                                  (2,0,0))
```

Teleport
players next
to each other

A blue text annotation 'Teleport players next to each other' is positioned to the right of the code. Two blue arrows originate from this text: one points to the coordinate '(0,0,0)' in the first line of code, and the other points to the coordinate '(2,0,0)' in the second line of code.



Code for Demo 1

CARBON

```
def testClient1CanSeeClient2Move(self):  
    SysTestUtils.PlayerMove(self.client2, 5.0,  
                             timeToWait=30000)
```



Code for Demo 1

CARBON

```
def testClient1CanSeeClient2Move(self): ← A particular  
    SysTestUtils.PlayerMove(self.client2, 5.0, test  
                               timeToWait=30000)
```



Code for Demo 1

```
def testClient1CanSeeClient2Move(self):  
    SysTestUtils.PlayerMove(self.client2, 5.0,  
                             timeToWait=30000)
```

Move the player for client2 5.0 meters and wait up to 30 seconds for her to get there



Code for Demo 1

CARBON

```
SysTestUtils.TestEntitySync(self.client2.charid,  
                             self.server,  
                             self.client2,  
                             maxDist=0.1,  
                             timeToWait=30000)
```





Code for Demo 1

CARBON

```
SysTestUtils.TestEntitySync(self.client2.charid,  
self.server,  
self.client2,  
maxDist=0.1,  
timeToWait=30000)
```

Check if the position of player2 on client2 is within 0.1m of the position of player2 on the server, waiting up to 30s

A blue arrow originates from the first parameter 'self.client2.charid' in the code block and points towards the explanatory text below.



Code for Demo 1

CARBON

```
SysTestUtils.TestEntitySync(self.client2.charid,  
                             self.server,  
                             self.client1,  
                             maxDist=0.1,  
                             timeToWait=30000)
```




Code for Demo 1

CARBON

```
SysTestUtils.TestEntitySync(self.client2.charid,  
self.server,  
self.client1,  
maxDist=0.1,  
timeToWait=30000)
```

Check if the position of player2 on client1 is within 0.1m of the position of player2 on the server, waiting up to 30s





Code for Demo 1

CARBON

```
def setUp(self):
    systemTest.TestCase.setUp(self,
        waitForGraphics=True,
        worldSpaceID=TEST_WORLD_SPACE_ID)
    SystemTestUtils.TeleportPlayerTo(self.client1, (0,0,0))
    SystemTestUtils.TeleportPlayerTo(self.client2, (2,0,0))

def testClient1CanSeeClient2Move(self):
    SysTestUtils.PlayerMove(self.client2, 5.0, timeToWait=30000)
    SysTestUtils.TestEntitySync(self.client2.charid, self.server, self.client2,
        maxDist=0.1, timeToWait=30000)
    SysTestUtils.TestEntitySync(self.client2.charid, self.server, self.client1,
        maxDist=0.1, timeToWait=30000)
```



Code for Demo 1

```
def TestEntitySync(entID, app1, app2, maxDist=0.5,
                  timeToWait=30000):
    def Synced():
        app1Pos = GetEntityPosition(app1, entID)
        app2Pos = GetEntityPosition(app2, entID)
        dist = geo2.Vec3Distance(app1Pos, app2Pos)
        return dist <= maxDist

    synced = WaitForCondition(Synced, timeToWait,
                             pollTime = 100)
    assertTrue(synced, "Entity positions are desynced")
```



Code for Demo 1

CARBON

```
def TestEntitySync(entID, app1, app2, maxDist=0.5,  
                  timeToWait=30000):
```

```
    def Synced():  
        app1Pos = GetEntityPosition(app1, entID)  
        app2Pos = GetEntityPosition(app2, entID)  
        dist = geo2.Vec3Distance(app1Pos, app2Pos)  
        return dist <= maxDist
```

Local function to
test if the positions
match

```
    synced = WaitForCondition(Synced, timeToWait,  
                             pollTime = 100)  
    assertTrue(synced, "Entity positions are desynced")
```



Code for Demo 1

CARBON

```
def TestEntitySync(entID, app1, app2, maxDist=0.5,  
                  timeToWait=30000):
```

```
    def Synced():
```

```
        app1Pos = GetEntityPosition(app1, entID)
```

```
        app2Pos = GetEntityPosition(app2, entID)
```

```
        dist = geo2.Vec3Distance(app1Pos, app2Pos)
```

```
        return dist <= maxDist
```

Get position of this
entity on client and
server

```
    synced = WaitForCondition(Synced, timeToWait,  
                             pollTime = 100)
```

```
    assertTrue(synced, "Entity positions are desynced")
```



Code for Demo 1

CARBON

```
def TestEntitySync(entID, app1, app2, maxDist=0.5,  
                  timeToWait=30000):
```

```
    def Synced():
```

```
        app1Pos = GetEntityPosition(app1, entID)
```

```
        app2Pos = GetEntityPosition(app2, entID)
```

```
        dist = geo2.Vec3Distance(app1Pos, app2Pos)
```

```
        return dist <= maxDist
```

Wait until Synced
returns True

```
synced = WaitForCondition(Synced, timeToWait,  
                          pollTime = 100)
```

```
assertTrue(synced, "Entity positions are desynced")
```




Code for Demo 1

```
def TestEntitySync(entID, app1, app2, maxDist=0.5,
                  timeToWait=30000):
    def Synced():
        app1Pos = GetEntityPosition(app1, entID)
        app2Pos = GetEntityPosition(app2, entID)
        dist = geo2.Vec3Distance(app1Pos, app2Pos)
        return dist <= maxDist

    synced = WaitForCondition(Synced, timeToWait,
                             pollTime = 100)
    assertTrue(synced, "Entity positions are desynced")
```

← Assert if positions don't match after timeToWait ms



Code for Demo 1

```
def GetEntityPosition(app, entID):  
    ent = app.entityService.FindEntityByID(entID)  
    return ent.GetComponent("position").position
```



Demo 2

CARBON

- Transferring between servers.
- 1 client, 2 servers, 1 proxy.
- Set up server 1 to be responsible for worldspace 1, and server2 for worldspace 2.
- Log client into worldspace 1.
- Walk through portal to worldspace 2.
- Check that client's player is in worldspace 2 on client and in worldspace 2 on server 2 and not in worldspace 1 on server 1.



Demo 2

CARBON

[Demo](#)



- What makes testing MMOs unique?
- 2 demos of our framework, Orchestrator, in action.
- **Architecture of Orchestrator.**
- Lessons Learned



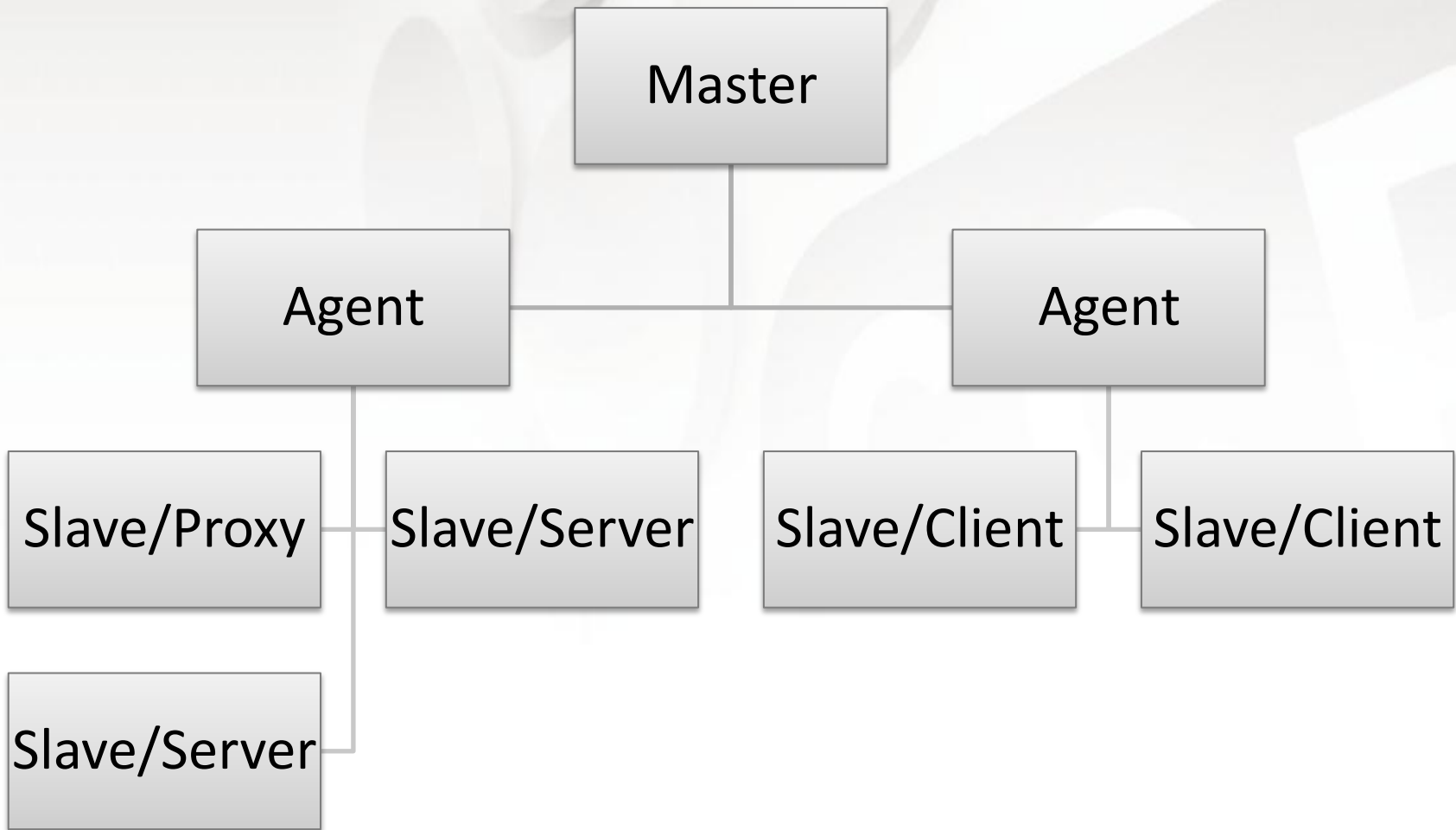
Single Script – Multiple Programs

CARBON

- Need architecture for having a single script control multiple programs.



Orchestrator Architecture





Slave

- Runs in the process of the proxy/server/client.
- Hooks to access any part of the app.



Agent

- Runs on each machine that a slave runs on.
- Starts/stops slave apps.
- Relays messages to/from slave apps.
- Passes exceptions back to master.



- Executes the test script, sending commands to agents.
- GUI for selecting which test(s) to run and reporting errors and failures.



- How do you make the test script look like normal single-process code?
- Python!
 - `self.client1.fooService.FooMethod()`
- How do you deal with a test that is twiddling a “complex” object?



- ObjectWrapper class
 - Stores objectID, nodeID
 - Implements `__getattr__`, `__setattr__`, `__call__`, `__eq__`, `__neq__`
 - `__getattr__` and `__call__` return the appropriate object inside of another ObjectWrapper



- In our teleport function, we used to have the following code to wait on the master until the player was teleported to a new scene:

```
while player.scene.sceneID != targetSceneID:  
    sleep(1.0)
```



How Python makes this easy

- `player.scene`

Master

```
__getattr__("scene")  
->  
ObjWrap(sceneObjectID,  
        nodeID)
```

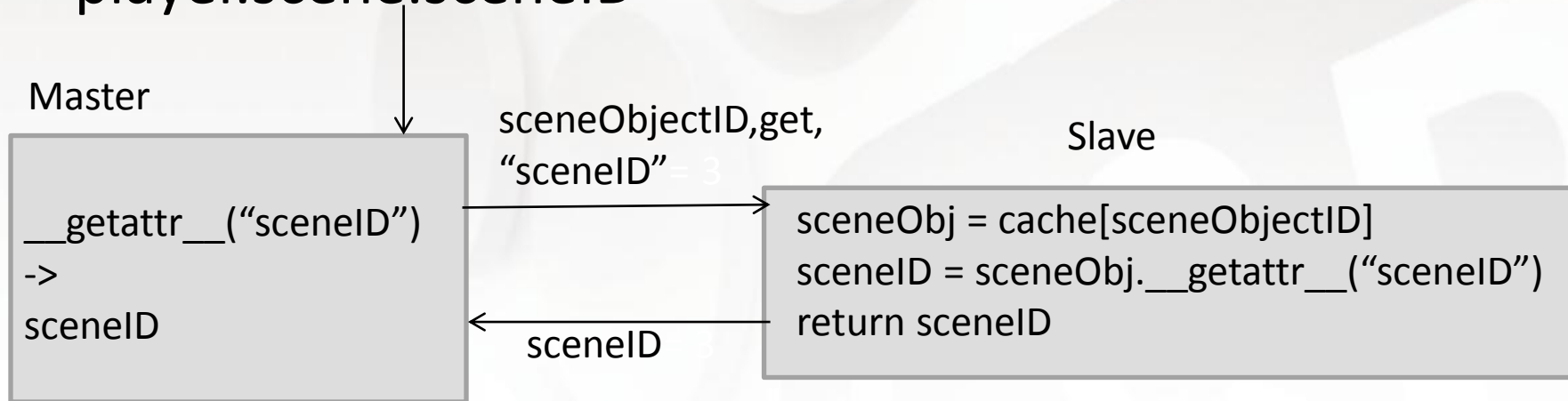
playerObjectID ,
get, "scene" = 2

Slave

```
playerObj = cache[playerObjectID]  
sceneObj = playerObj.__getattr__("scene")  
sceneObjectID = hash(sceneObj)  
cache[sceneObjectID] = sceneObj  
return sceneObjectID
```

sceneObjectID

- `player.scene.sceneID`





This makes asynchronicity problems worse

CARBON

- Every “.” is a round-trip from master to slave
- `player.scene.sceneID`
 - Any amount of time could pass between getting the “scene” and then trying to grab the `sceneID` off of it.
 - Scene unloaded
 - `clientPlayer` removed from scene



Make it deterministic

CARBON

- Could write a function on the slave that just does the same loop and call that from master.
- Listen to events that the client is already sending out for internal use (with a timeout):

```
client.RegisterEventCallback("OnEntityTeleport", self.OnEntityTeleport)
```



- What makes testing MMOs unique?
- 2 demos of our framework, Orchestrator, in action.
- Architecture of Orchestrator.
- **Lessons Learned**



Don't Test Everything

- Only test basic functionality of each major system or maintenance burden becomes too high.
 - Can I move?
 - Can I punch?
 - Can I chat?
 - Can I join a group?
- World of Darkness project used to have around 120 system tests. Now it has about 40.



Avoid implementation details

- Do not directly inspect implementation details of systems your are testing.
- In an asynchronous system, not only will your test be broken by changes to the implementation, but also by changes in the timing.



- Build up a library of high-level, well-tested functions that can be used in lots of tests
 - CreateNPC
 - PlayerMove
 - SelectEntity
 - PerformAction



- Programmer who wrote the system should write the test for the system – not a separate QA Engineer.
- Writing tests for MMOs is hard and requires domain knowledge of the system being tested.
- QA Engineers couldn't keep up with changes to the system and Programmers weren't nice enough to keep them informed.



Sleep is the devil

- Putting in a sleep for an arbitrary amount of time to fix a bug is the sign of a race condition that is just being avoided, not fixed.
- Make sure events are created for what you're waiting for and listen for them (with appropriate timeout).



Questions?

CARBON

We're Hiring!

<http://ccpgames.com/jobs>