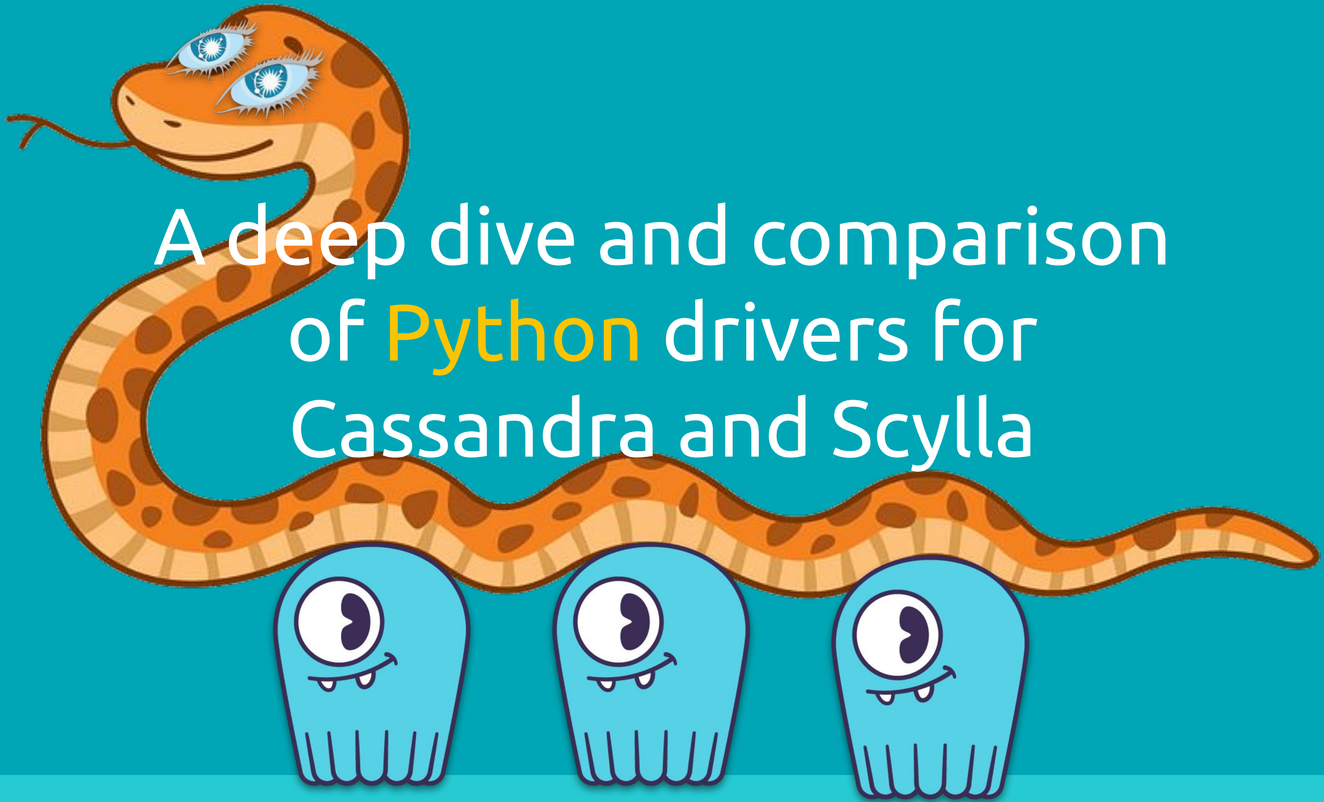


A deep dive and comparison of Python drivers for Cassandra and Scylla

Why and how we wrote a Python driver for Scylla

EuroPython 2020

Bonjour !

CTO at numberly

Alexys Jacob

Gentoo Linux developer

- dev-db / **mongodb / redis / scylla**
- sys-cluster / **keepalived / ipvsadm / consul**
- dev-python / **pymongo**
- cluster + containers **team member**

Open Source contributor

- MongoDB
- Scylla
- Apache Airflow
- Python Software Foundation **contributing member**

EuroPython uses Discord… Discord uses Scylla!

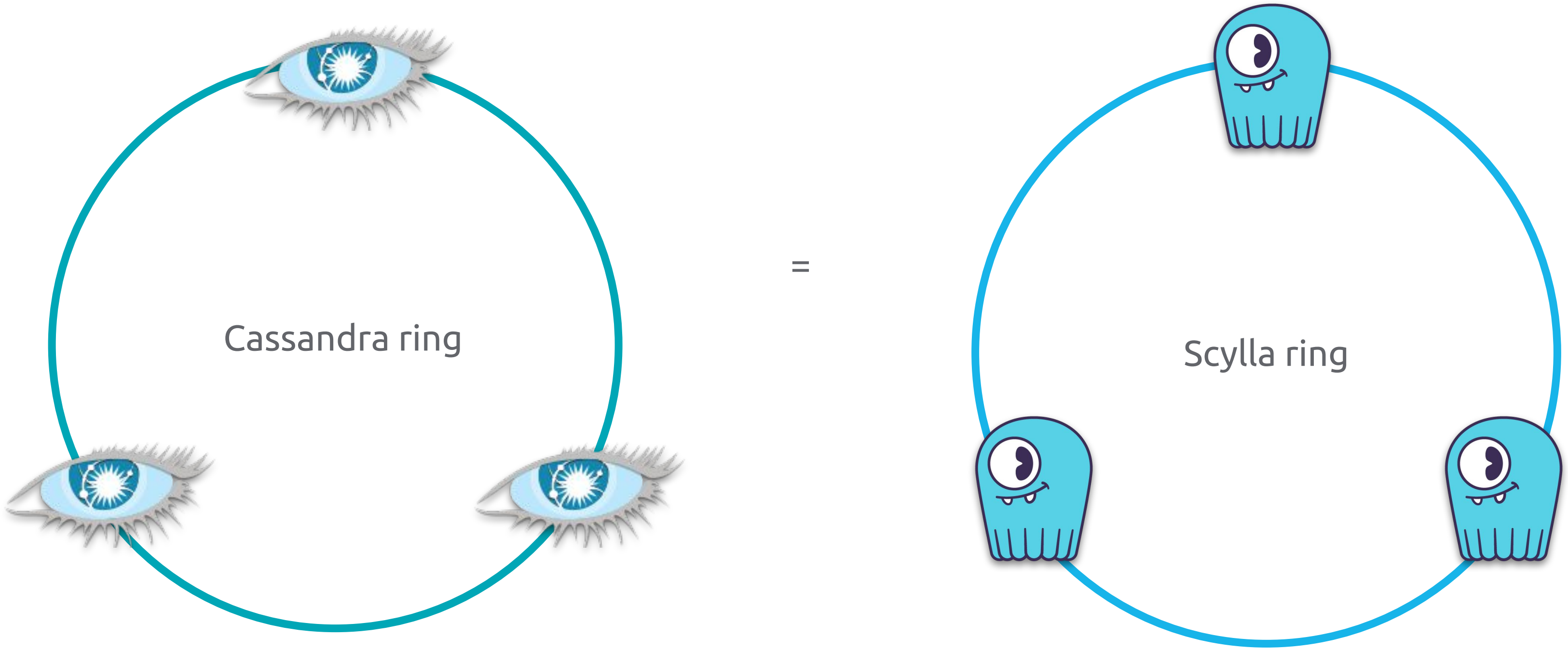
Check out the talk of Mark Smith, Director of Engineering at Discord

Leveraging Consistent Hashing in Python applications

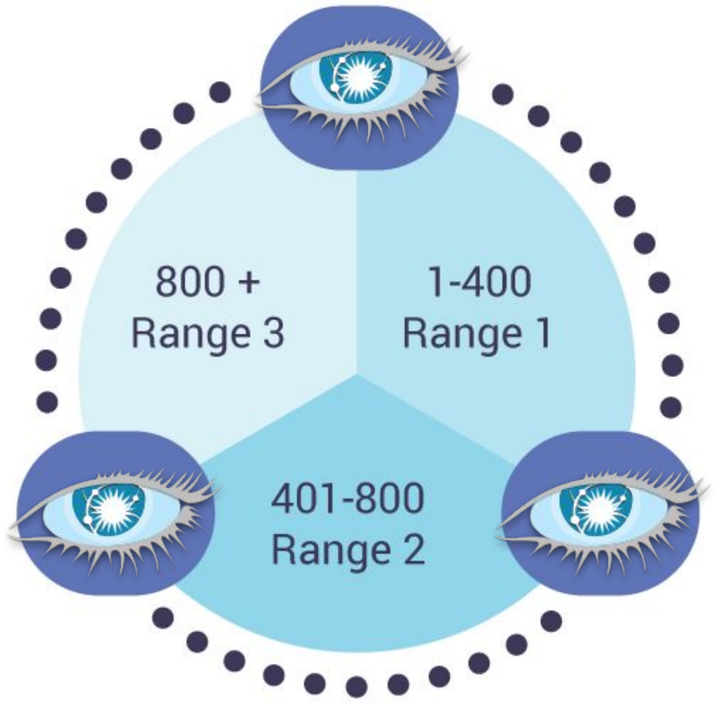
Check out my talk from EuroPython 2017 to get deeper into consistent hashing

Deep dive
Cassandra & Scylla token ring architectures

A cluster is a collection of nodes

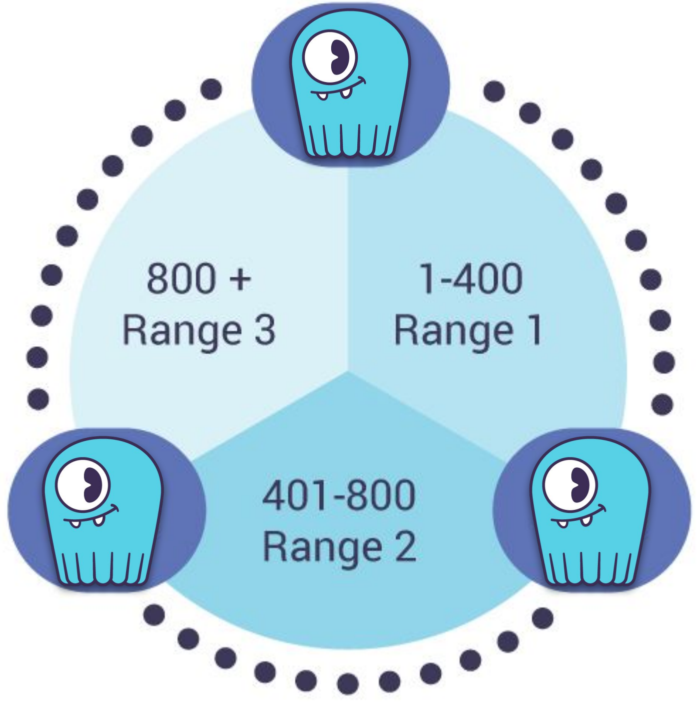


Each node is responsible for a partition on the token ring



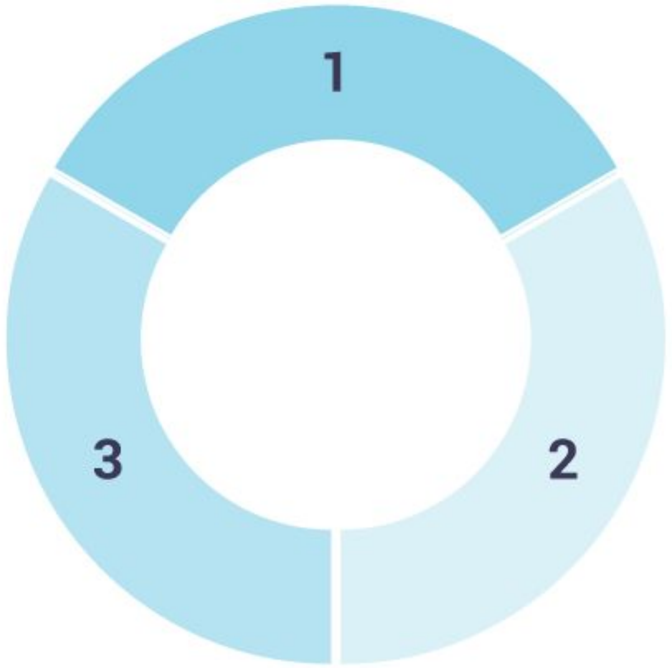
Cassandra ring

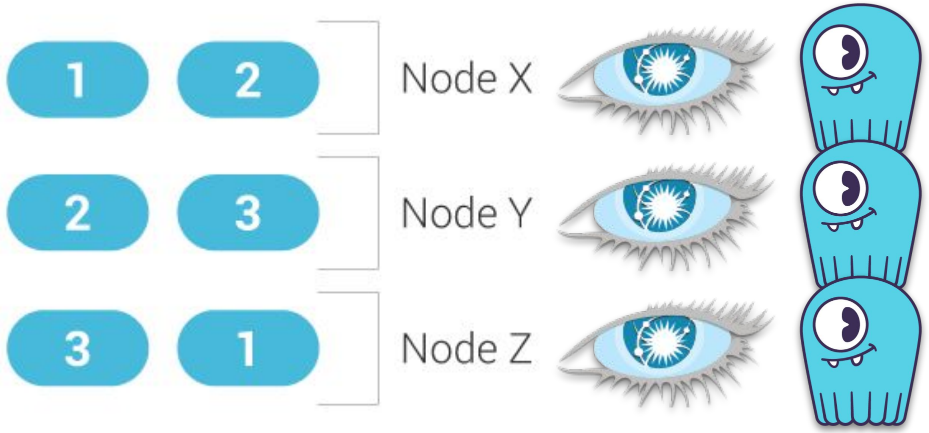
=



Scylla ring

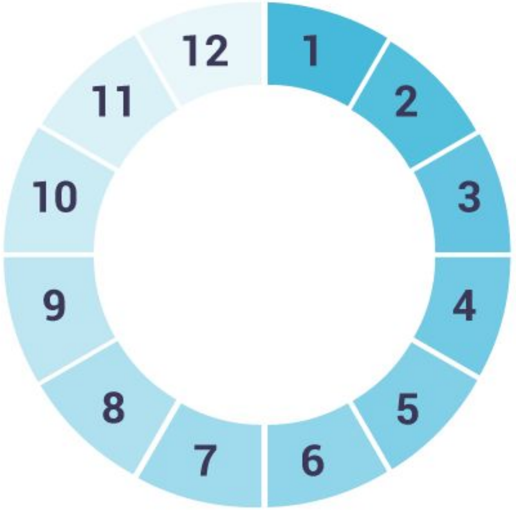
Replication Factor provides higher data availability

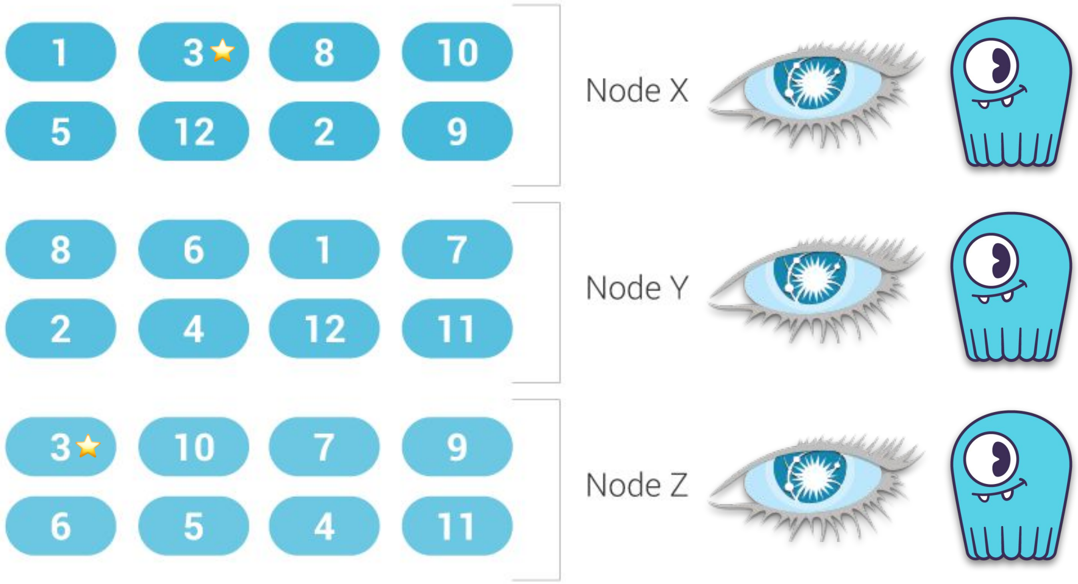




Replication Factor = 2

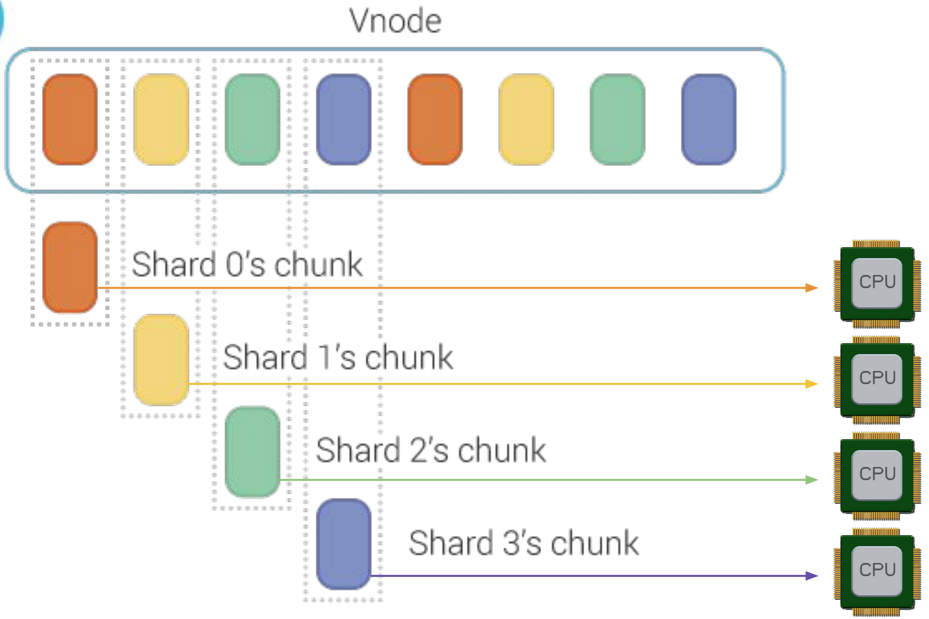
Virtual Nodes = better partition distribution between nodes



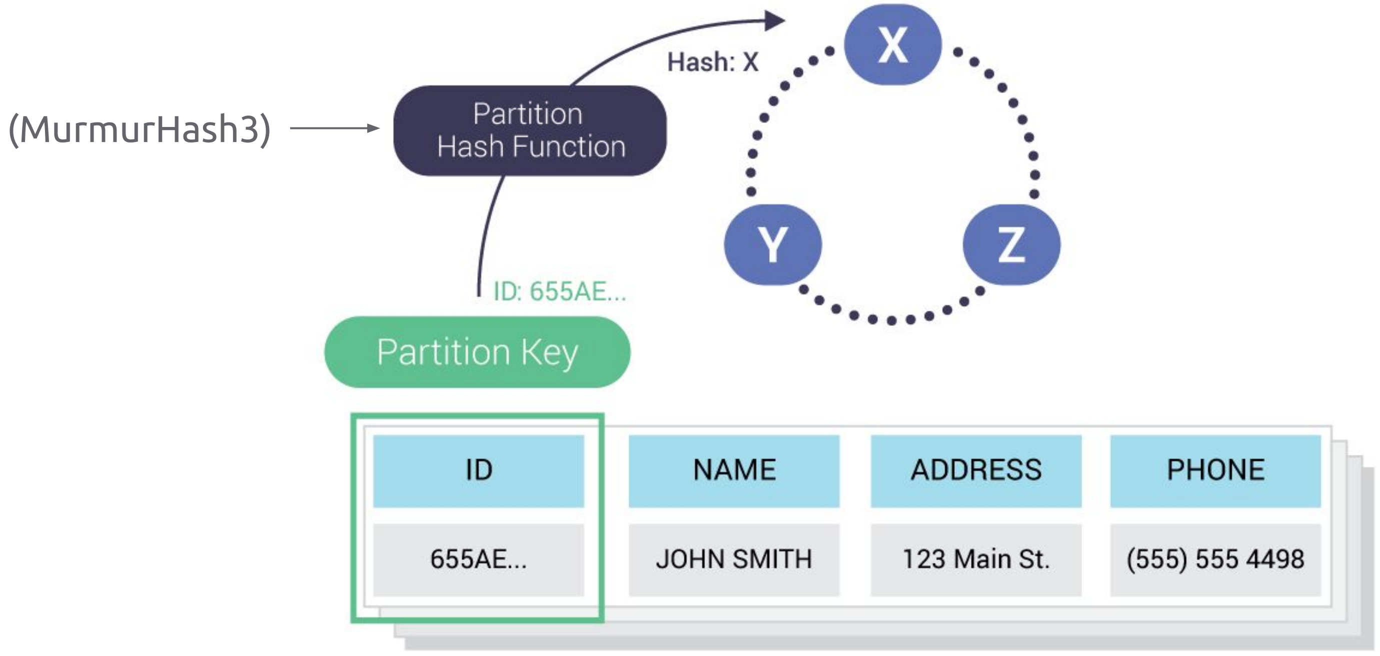


Replication Factor = 2

Scylla's Virtual Nodes are split into shards bound to cores!



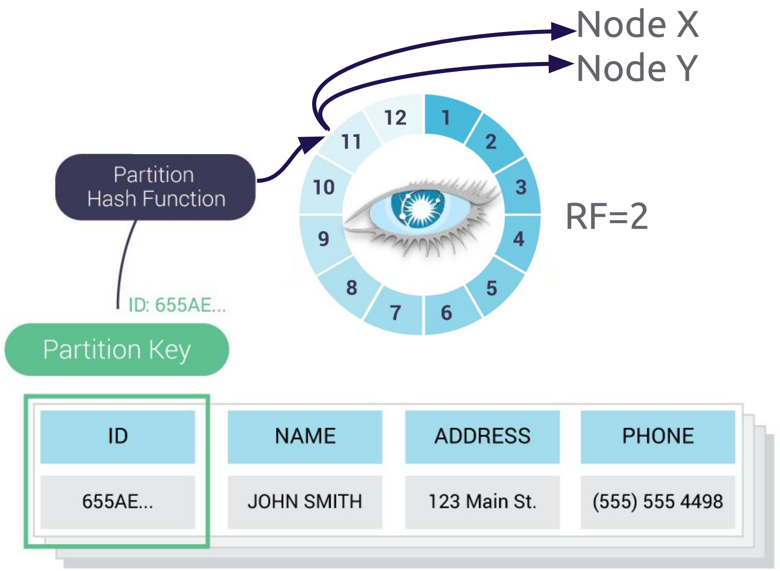
Rows are located on nodes by hashing their partition key



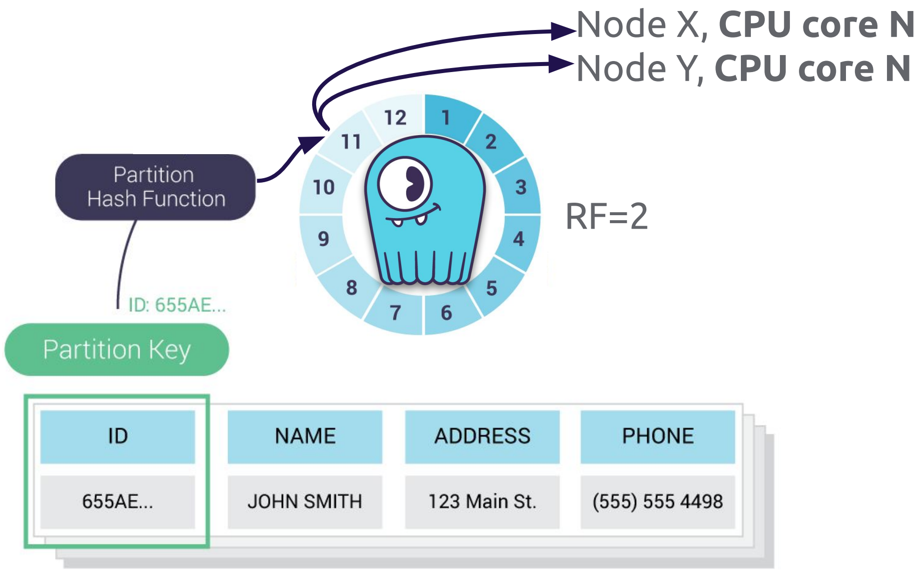
Take away: shard-per-node vs shard-per-core architecture

Cassandra
hash(Partition Key) token leads to RF*nodes

Scylla
hash(Partition Key) token leads to RF*nodes **cores**



ID	NAME	ADDRESS	PHONE
655AE...	JOHN SMITH	123 Main St.	(555) 555 4498

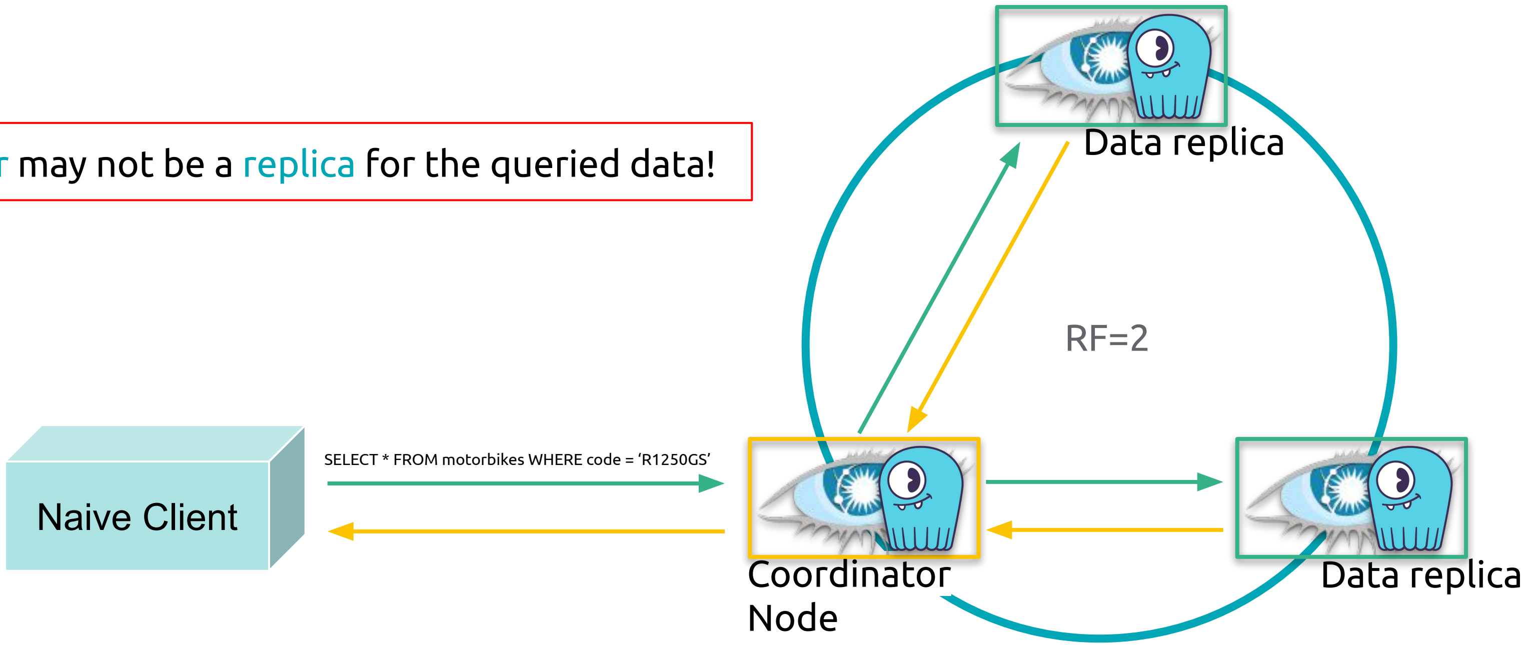


ID	NAME	ADDRESS	PHONE
655AE...	JOHN SMITH	123 Main St.	(555) 555 4498

Client drivers should leverage the token ring architecture!

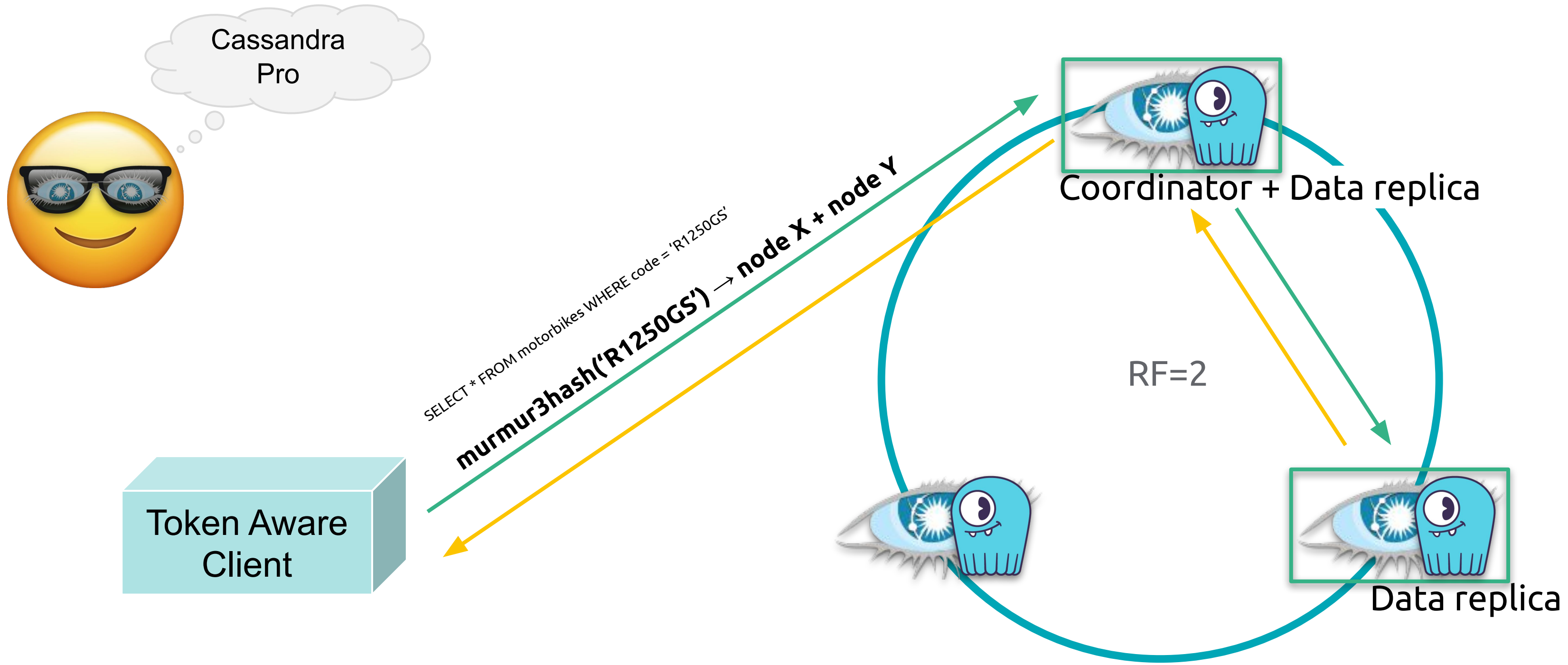
Naive clients route queries to any node (coordinator)

The coordinator may not be a replica for the queried data!

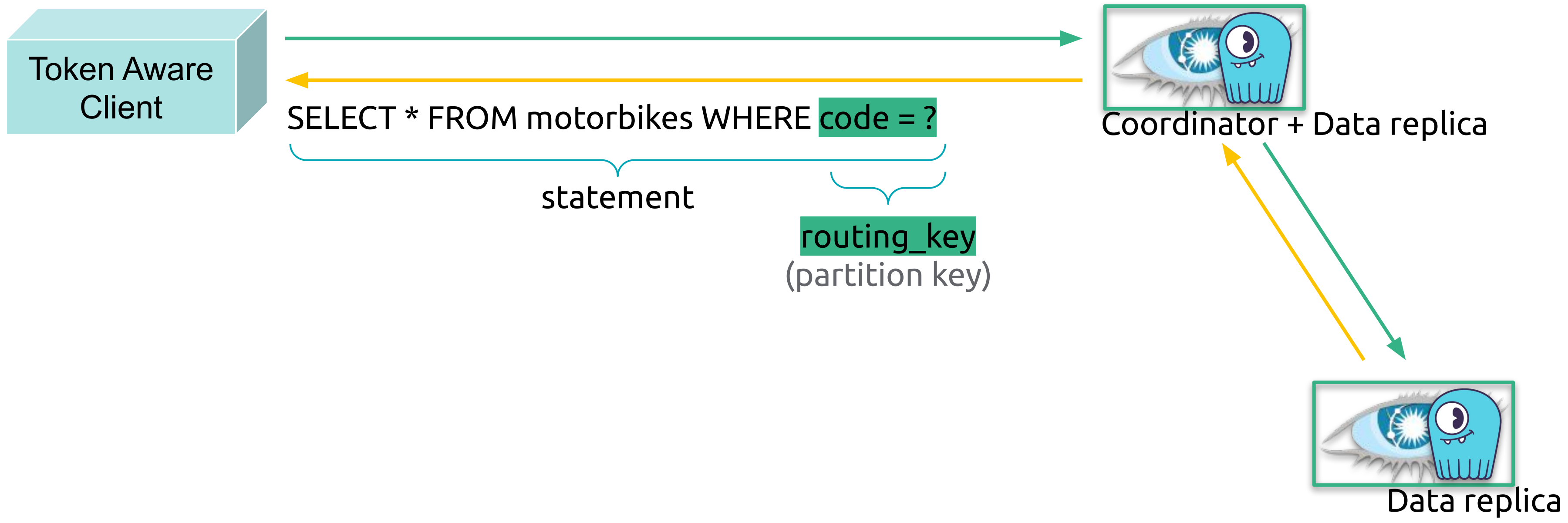


Deep dive
Python cassandra-driver TokenAwarePolicy

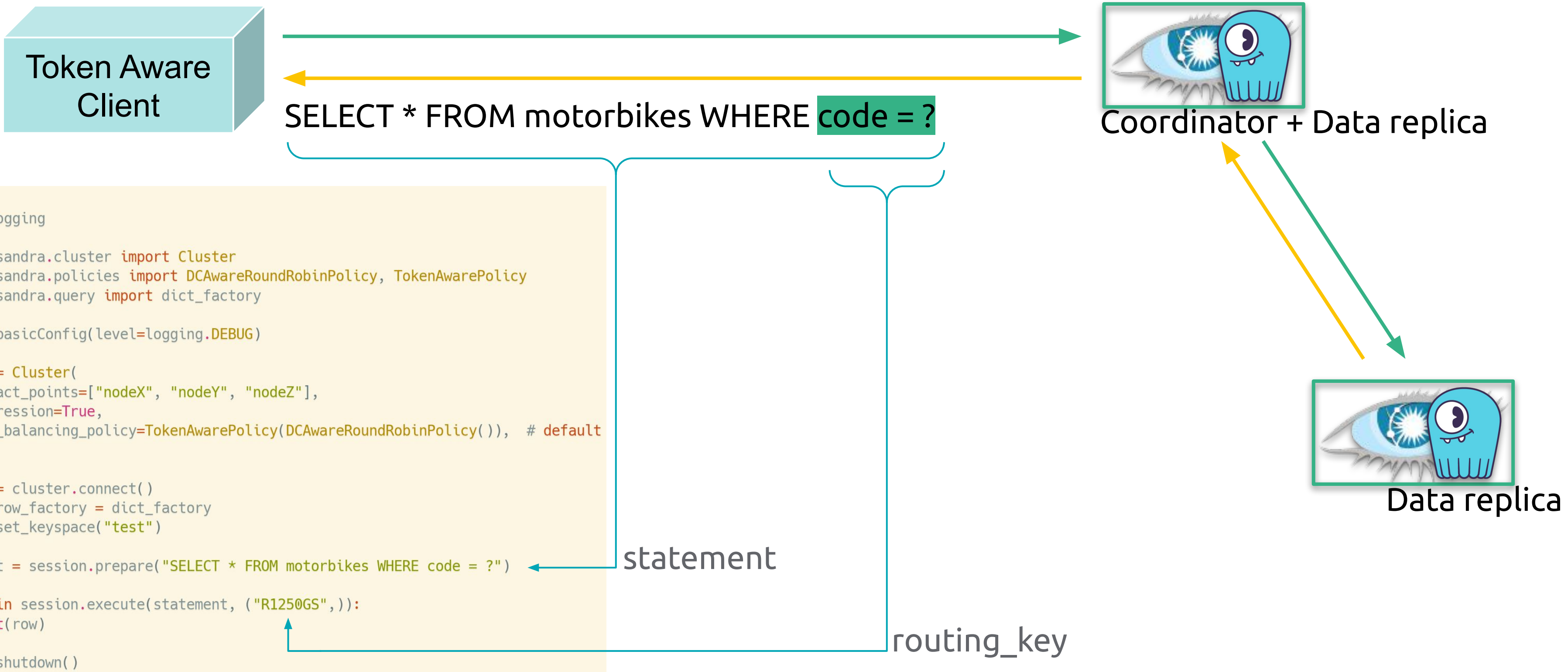
Token Aware clients route queries to the right node(s)!


Cassandra Pro
Token Aware Client
SELECT * FROM motorbikes WHERE code = 'R1250GS'
murmur3hash('R1250GS') → node X + node Y
Coordinator + Data replica
RF=2
Data replica

TokenAwarePolicy: Statement + routing key = node(s)

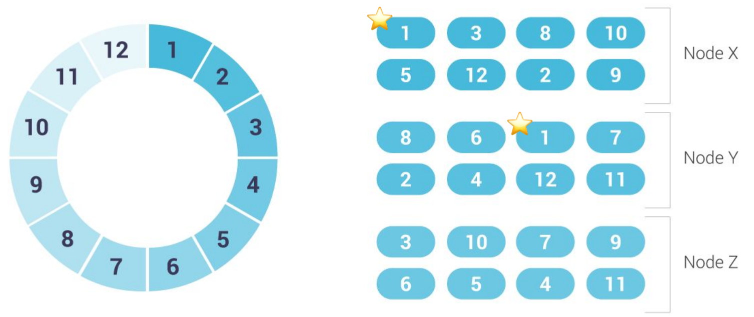

Token Aware Client
SELECT * FROM motorbikes WHERE code = ?
statement
routing_key
(partition key)
Coordinator + Data replica
Data replica

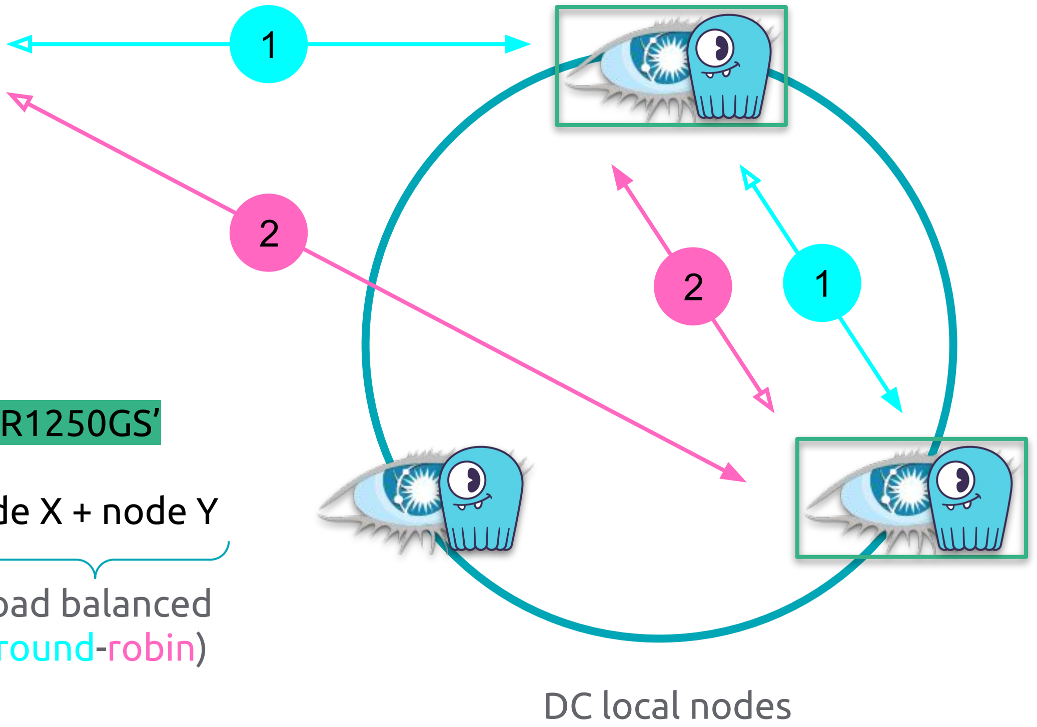
TokenAwarePolicy: Statement + routing key = node(s)


Token Aware Client
SELECT * FROM motorbikes WHERE code = ?
Coordinator + Data replica
Data replica
statement
routing_key

```python
import logging

from cassandra.cluster import Cluster
from cassandra.policies import DCAwareRoundRobinPolicy, TokenAwarePolicy
from cassandra.query import dict_factory

logging.basicConfig(level=logging.DEBUG)

cluster = Cluster(
    contact_points=["nodeX", "nodeY", "nodeZ"],
    compression=True,
    load_balancing_policy=TokenAwarePolicy(DCAwareRoundRobinPolicy()),  # default
)

session = cluster.connect()
session.row_factory = dict_factory
session.set_keyspace("test")

statement = session.prepare("SELECT * FROM motorbikes WHERE code = ?")

for row in session.execute(statement, ("R1250GS",)):
    print(row)

cluster.shutdown()
```

Default TokenAwarePolicy(DCAwareRoundRobinPolicy)



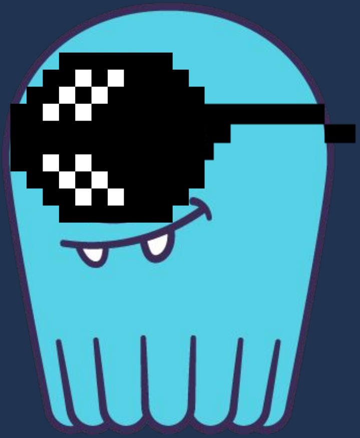


SELECT * FROM motorbikes WHERE code = ‘R1250GS’

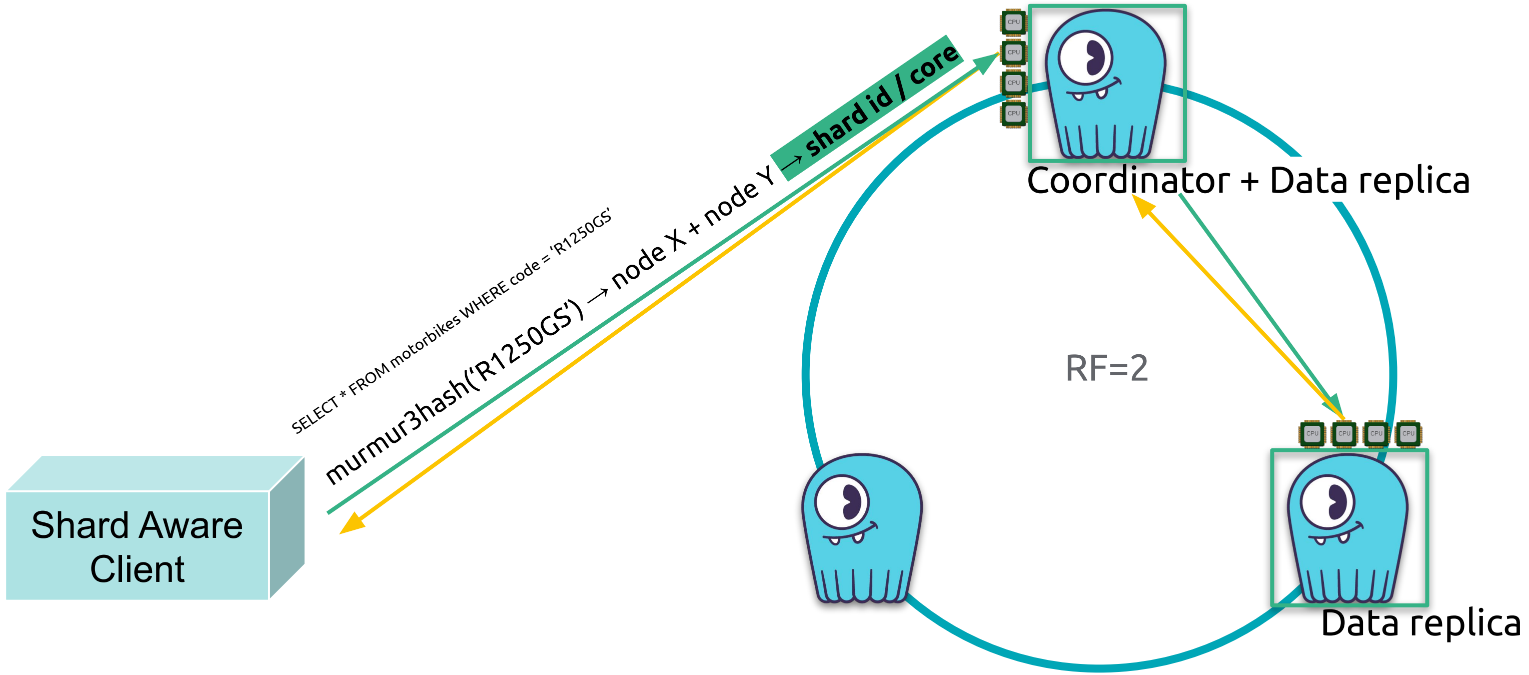
murmur3hash(‘R1250GS’) = partition 1 = $\underbrace{\text{node X + node Y}}_{\text{load balanced (round-robin)}}$

Can’t beat my Cassandra’s TokenAwarePolicy(DCAwareRoundRobinPolicy)!

Yes you can.
Use Scylla and a shard-per-core aware driver!



Shard Aware clients route queries to the right node(s) + core!

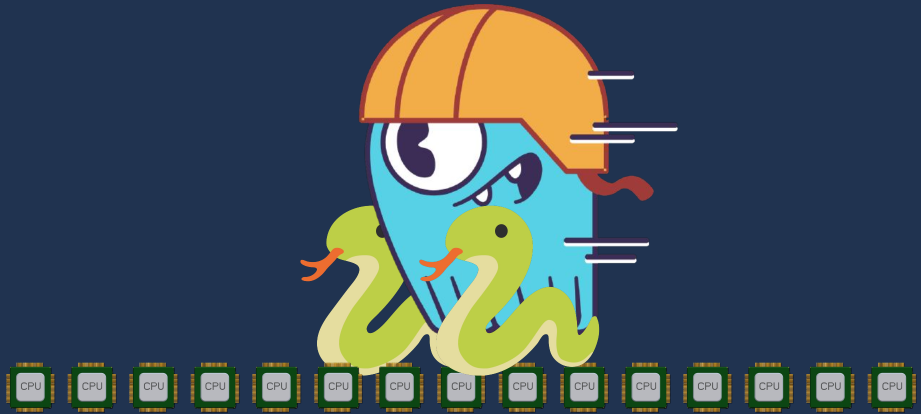

SELECT * FROM motorbikes WHERE code = 'R1250GS'
murmur3hash('R1250GS') → node X + node Y → shard id / core
Coordinator + Data replica
RF=2
Data replica
Shard Aware Client
CPU

Scylla shard aware drivers: Python was missing!

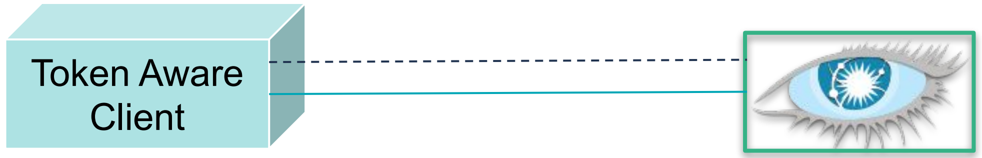
Forks of DataStax drivers to retain maximal compatibility and foster fast iteration

- **Java**
  - First one officially released in 2019
- **Go (gocql, gocqlx)**
  - Used in scylla-manager and other Go based tooling
- **C++**
  - WIP

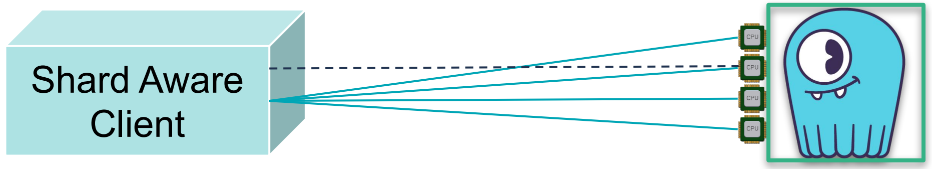
Let’s make a Python shard-aware driver!



cassandra-driver / scylla-driver structural differences



- 1 control connection (cluster metadata, topology)
- 1 connection per node
- Token calculation selects the right connection to node to route queries



- 1 control connection (cluster metadata, topology)
- 1 connection per core per node
- Token calculation selects the right node
- Shard id calculation selects the connection to the right core to route queries

TODO: from cassandra-driver to scylla-driver

- **1 control connection (cluster metadata, topology)**
  - Use as-is

- **1 connection per core per node**
  - Connection needs to detect Scylla shard aware clusters (while retaining compatibility with Cassandra clusters)
  - HostConnection pool should open a Connection to every core of its host/node

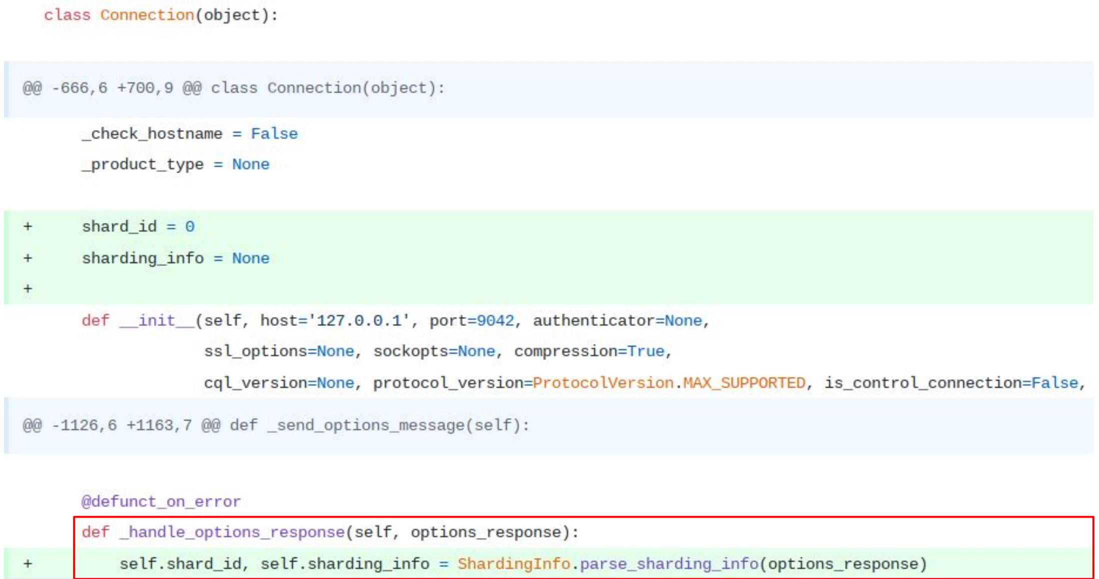
- **Token calculation selects the right node**
  - Use TokenAwarePolicy as-is

- **Shard id calculation selects the right connection to core to route queries**
  - Cluster should pass down the query routing_key to the pool to allow connection selection
  - Implement shard id calculation based on the query routing_key token
  - HostConnection pool should select the connection to the right core to route the query

Implementing shard-awareness for scylla-driver

😍Inspired by Java driver’s shard aware implementation, Israel Fruchter paved the path and made the first PR for Python shard-awareness!

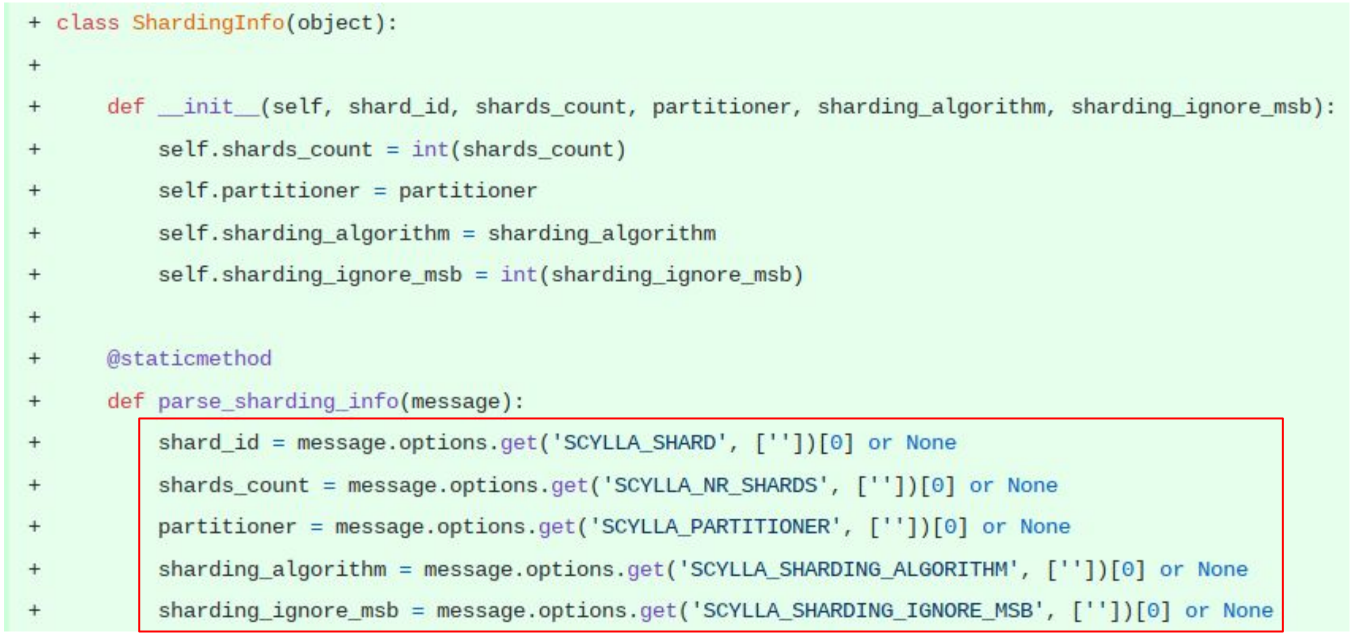
- Connection needs to detect Scylla shard aware clusters (while retaining compatibility with Cassandra clusters)



```
class Connection(object):

@@ -666,6 +700,9 @@ class Connection(object):
        _check_hostname = False
        _product_type = None

+       shard_id = 0
+       sharding_info = None
+
        def __init__(self, host='127.0.0.1', port=9042, authenticator=None,
                     ssl_options=None, sockopts=None, compression=True,
                     cql_version=None, protocol_version=ProtocolVersion.MAX_SUPPORTED, is_control_connection=False,
@@ -1126,6 +1163,7 @@ def _send_options_message(self):

        @defunct_on_error
        def _handle_options_response(self, options_response):
+           self.shard_id, self.sharding_info = ShardingInfo.parse_sharding_info(options_response)
```

scylla-driver shard-awareness detection

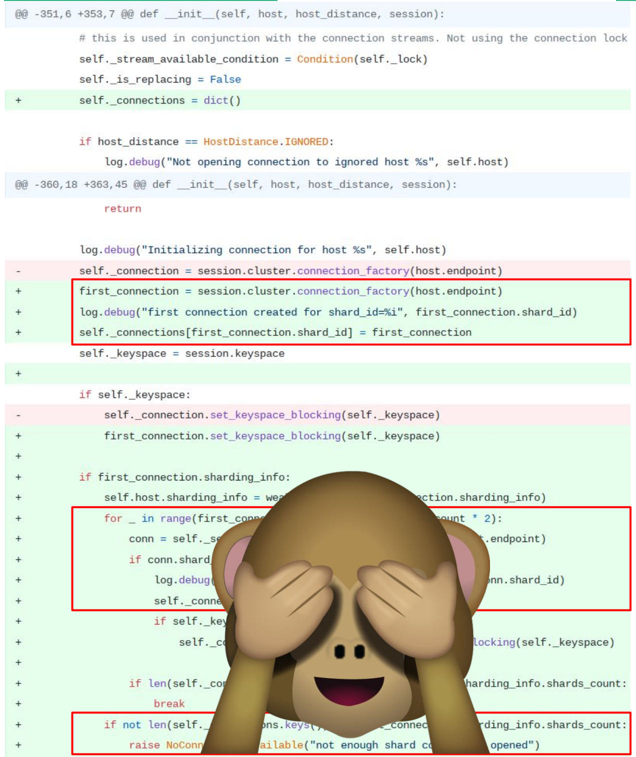
- Connection detects Scylla shard aware clusters thanks to response message options:



```
+ class ShardingInfo(object):
+
+     def __init__(self, shard_id, shards_count, partitioner, sharding_algorithm, sharding_ignore_msb):
+         self.shards_count = int(shards_count)
+         self.partitioner = partitioner
+         self.sharding_algorithm = sharding_algorithm
+         self.sharding_ignore_msb = int(sharding_ignore_msb)
+
+     @staticmethod
+     def parse_sharding_info(message):
+         shard_id = message.options.get('SCYLLA_SHARD', [''])[0] or None
+         shards_count = message.options.get('SCYLLA_NR_SHARDS', [''])[0] or None
+         partitioner = message.options.get('SCYLLA_PARTITIONER', [''])[0] or None
+         sharding_algorithm = message.options.get('SCYLLA_SHARDING_ALGORITHM', [''])[0] or None
+         sharding_ignore_msb = message.options.get('SCYLLA_SHARDING_IGNORE_MSB', [''])[0] or None
```

scylla-driver connections to shards/cores

- HostConnection pool should open a Connection to every core of its host/node



```
@@ -351,6 +353,7 @@ def __init__(self, host, host_distance, session):
          # this is used in conjunction with the connection streams. Not using the connection lock
          self._stream_available_condition = Condition(self._lock)
          self._is_replacing = False
+         self._connections = dict()

          if host_distance == HostDistance.IGNORED:
              log.debug("Not opening connection to ignored host %s", self.host)
@@ -360,18 +363,45 @@ def __init__(self, host, host_distance, session):
              return

          log.debug("Initializing connection for host %s", self.host)
-         self._connection = session.cluster.connection_factory(host.endpoint)
+         first_connection = session.cluster.connection_factory(host.endpoint)
+         log.debug("first connection created for shard_id=%i", first_connection.shard_id)
+         self._connections[first_connection.shard_id] = first_connection
          self._keyspace = session.keyspace
+
          if self._keyspace:
-             self._connection.set_keyspace_blocking(self._keyspace)
+             first_connection.set_keyspace_blocking(self._keyspace)
+
+         if first_connection.sharding_info:
+             self.host.sharding_info = wea                    ection.sharding_info)
+             for _ in range(first_conn                       ount * 2):
+                 conn = self._se                               .endpoint)
+                 if conn.shard
+                     log.debug(                                nn.shard_id)
+                     self._conne
+                     if self._key
+                         self._c                               locking(self._keyspace)
+
+                 if len(self._co                               harding_info.shards_count:
+                     break
+             if not len(self._        ons.keys(       _connec      rding_info.shards_count:
+                 raise NoConn        ailable("not enough shard co          opened")
```

self._connections keys = shard id, values = connection obj

first connection detects shard support on the node

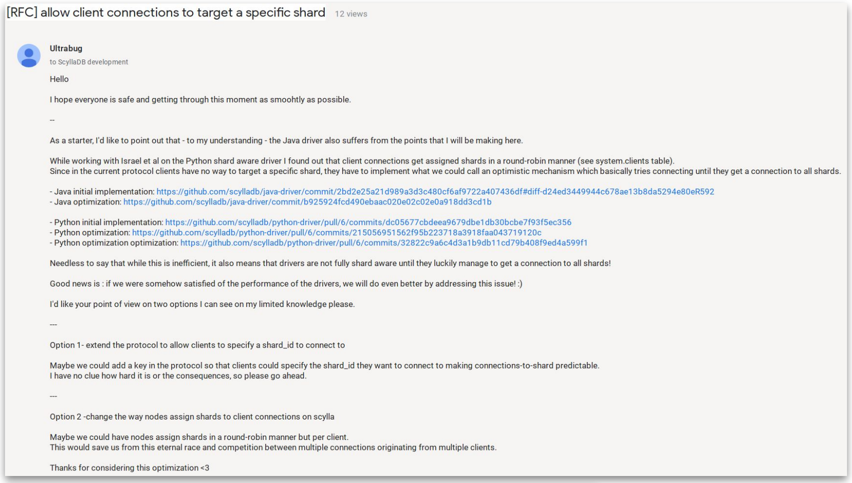
synchronous and optimistic way to get a connection to all cores... we try at max 2*number of cores on the node...

...and fail if not fully connected!

The Connection to every core problem

- **There is no way for a client to specify which shard/core it wants to connect to!**
  - Would require Scylla protocol to diverge from Cassandra's
  - This means that all other Scylla drivers are affected!
  - Sent an RFC on the mailing-list to raise the problem
  - Current status looking good
    - Client source port based shard attribution logic
    - Currently being implemented!

- TODO: connection to cores optimization
  - Fix startup time with asynchronous connection logic
  - On startup try to connect to every shard only once
  - A connection to all shard should not be mandatory



[RFC] allow client connections to target a specific shard 12 views

Ultrabug
to ScyllaDB development

Hello

I hope everyone is safe and getting through this moment as smoohtly as possible.

--

As a starter, I'd like to point out that - to my understanding - the Java driver also suffers from the points that I will be making here.

While working with Israel et al on the Python shard aware driver I found out that client connections get assigned shards in a round-robin manner (see system.clients table).
Since in the current protocol clients have no way to target a specific shard, they have to implement what we could call an optimistic mechanism which basically tries connecting until they get a connection to all shards.

- Java initial implementation: https://github.com/scylladb/java-driver/commit/2bd2e25a21d989a3d3c480cf6af9722a407436df#diff-d24ed3449944c678ae13b8da5294e80eR592
- Java optimization: https://github.com/scylladb/java-driver/commit/b925924fcd490ebaac020e02c02e0a918dd3cd1b

- Python initial implementation: https://github.com/scylladb/python-driver/pull/6/commits/dc05677cbdeea9679dbe1db30bcbe7f93f5ec356
- Python optimization: https://github.com/scylladb/python-driver/pull/6/commits/215056951562f95b223718a3918faa043719120c
- Python optimization optimization: https://github.com/scylladb/python-driver/pull/6/commits/32822c9a6c4d3a1b9db11cd79b408f9ed4a599f1

Needless to say that while this is inefficient, it also means that drivers are not fully shard aware until they luckily manage to get a connection to all shards!

Good news is : if we were somehow satisfied of the performance of the drivers, we will do even better by addressing this issue! :)

I'd like your point of view on two options I can see on my limited knowledge please.

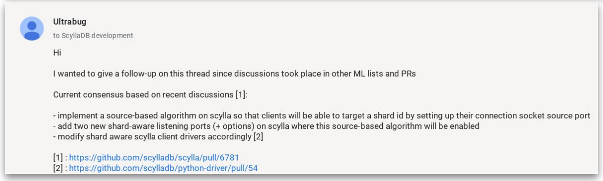
Option 1- extend the protocol to allow clients to specify a shard_id to connect to

Maybe we could add a key in the protocol so that clients could specify the shard_id they want to connect to making connections-to-shard predictable.
I have no clue how hard it is or the consequences, so please go ahead.

Option 2 -change the way nodes assign shards to client connections on scylla

Maybe we could have nodes assign shards in a round-robin manner but per client.
This would save us from this eternal race and competition between multiple connections originating from multiple clients.

Thanks for considering this optimization <3



Ultrabug
to ScyllaDB development

Hi

I wanted to give a follow-up on this thread since discussions took place in other ML lists and PRs

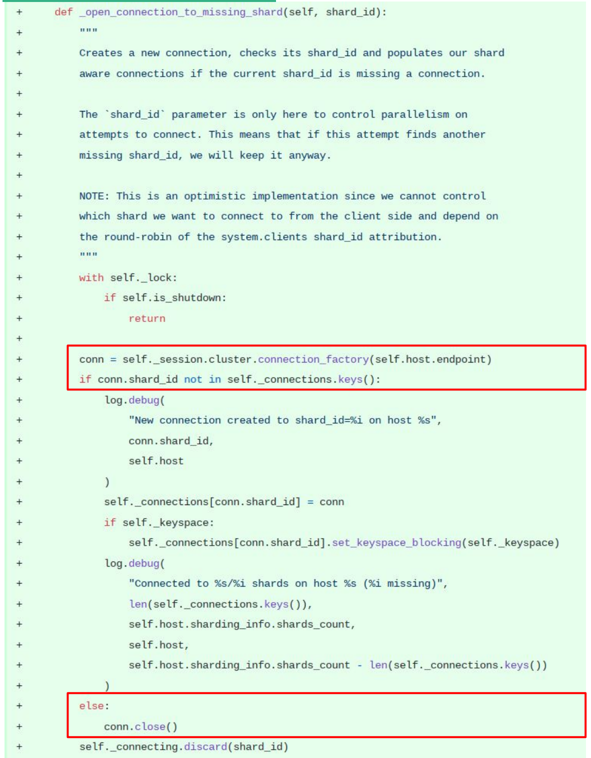
Current consensus based on recent discussions [1]:

- implement a source-based algorithm on scylla so that clients will be able to target a shard id by setting up their connection socket source port
- add two new shard-aware listening ports (+ options) on scylla where this source-based algorithm will be enabled
- modify shard aware scylla client drivers accordingly [2]

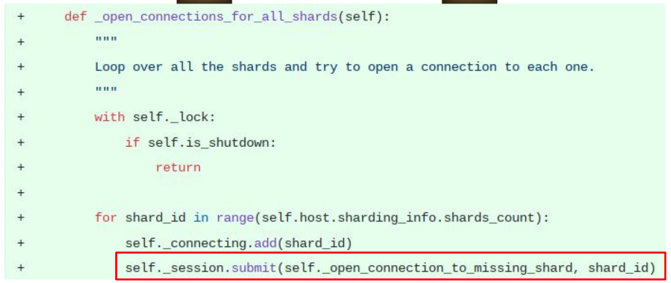
[1] : https://github.com/scylladb/scylla/pull/6781
[2] : https://github.com/scylladb/python-driver/pull/54

scylla-driver enhanced connections to shards/cores

- HostConnection pool should open a Connection to every core of its host/node



```
+   def _open_connection_to_missing_shard(self, shard_id):
+       """
+       Creates a new connection, checks its shard_id and populates our shard
+       aware connections if the current shard_id is missing a connection.
+
+       The `shard_id` parameter is only here to control parallelism on
+       attempts to connect. This means that if this attempt finds another
+       missing shard_id, we will keep it anyway.
+
+       NOTE: This is an optimistic implementation since we cannot control
+       which shard we want to connect to from the client side and depend on
+       the round-robin of the system.clients shard_id attribution.
+       """
+       with self._lock:
+           if self.is_shutdown:
+               return
+
+       conn = self._session.cluster.connection_factory(self.host.endpoint)
+       if conn.shard_id not in self._connections.keys():
+           log.debug(
+               "New connection created to shard_id=%i on host %s",
+               conn.shard_id,
+               self.host
+           )
+           self._connections[conn.shard_id] = conn
+           if self._keyspace:
+               self._connections[conn.shard_id].set_keyspace_blocking(self._keyspace)
+           log.debug(
+               "Connected to %s/%i shards on host %s (%i missing)",
+               len(self._connections.keys()),
+               self.host.sharding_info.shards_count,
+               self.host,
+               self.host.sharding_info.shards_count - len(self._connections.keys())
+           )
+       else:
+           conn.close()
+       self._connecting.discard(shard_id)
```

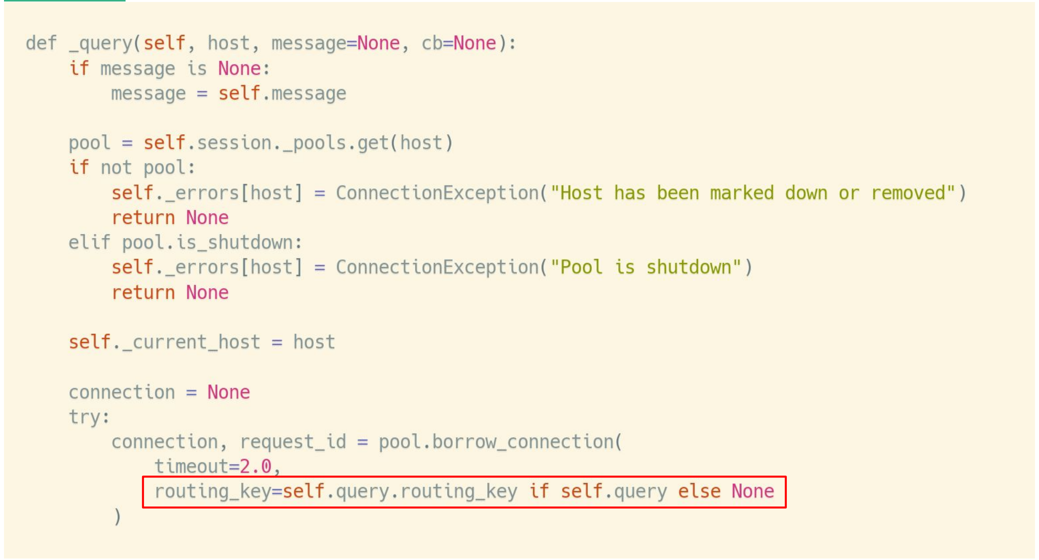


```
+   def _open_connections_for_all_shards(self):
+       """
+       Loop over all the shards and try to open a connection to each one.
+       """
+       with self._lock:
+           if self.is_shutdown:
+               return
+
+       for shard_id in range(self.host.sharding_info.shards_count):
+           self._connecting.add(shard_id)
+           self._session.submit(self._open_connection_to_missing_shard, shard_id)
```

asynchronous!

scylla-driver routing key token to core calculation

- Cluster should pass down the query routing_key to the pool to allow connection selection



```
def _query(self, host, message=None, cb=None):
    if message is None:
        message = self.message

    pool = self.session._pools.get(host)
    if not pool:
        self._errors[host] = ConnectionException("Host has been marked down or removed")
        return None
    elif pool.is_shutdown:
        self._errors[host] = ConnectionException("Pool is shutdown")
        return None

    self._current_host = host

    connection = None
    try:
        connection, request_id = pool.borrow_connection(
            timeout=2.0,
            routing_key=self.query.routing_key if self.query else None
        )
```

- Implement shard id calculation based on the query routing_key token
  - Pure Python calculation function was badly impacting driver performance and latency...!

Performance concern: move shard id calculation to Cython

- cassandra.shard_info: Cython shard id calculation used by HostConnection to route queries

```
cdef class ShardingInfo():

    @staticmethod
    def parse_sharding_info(message):
        """
        Detect Scylla shard awareness support from response options message
        """
        shard_id = message.options.get('SCYLLA_SHARD', [''])[0] or None
        shards_count = message.options.get('SCYLLA_NR_SHARDS', [''])[0] or None
        partitioner = message.options.get('SCYLLA_PARTITIONER', [''])[0] or None
        sharding_algorithm = message.options.get('SCYLLA_SHARDING_ALGORITHM', [''])[0] or None
        sharding_ignore_msb = message.options.get('SCYLLA_SHARDING_IGNORE_MSB', [''])[0] or None

        if not (shard_id or shards_count or partitioner == "org.apache.cassandra.dht.Murmur3Partitioner" or
            sharding_algorithm == "biased-token-round-robin" or sharding_ignore_msb):
            return 0, None

        return int(shard_id), ShardingInfo(shard_id, shards_count, partitioner, sharding_algorithm, sharding_ignore_msb)


    def shard_id_from_token(self, int64_t token_input):
        """
        Find the right shard id (core) from the given routing_key's token
        This is how we route queries to the right core!
        """
        cdef uint64_t biased_token = token_input + (<uint64_t>1 << 63);
        biased_token <<= self.sharding_ignore_msb;
        cdef int shardId = (<__uint128_t>biased_token * self.shards_count) >> 64;
        return shardId
```

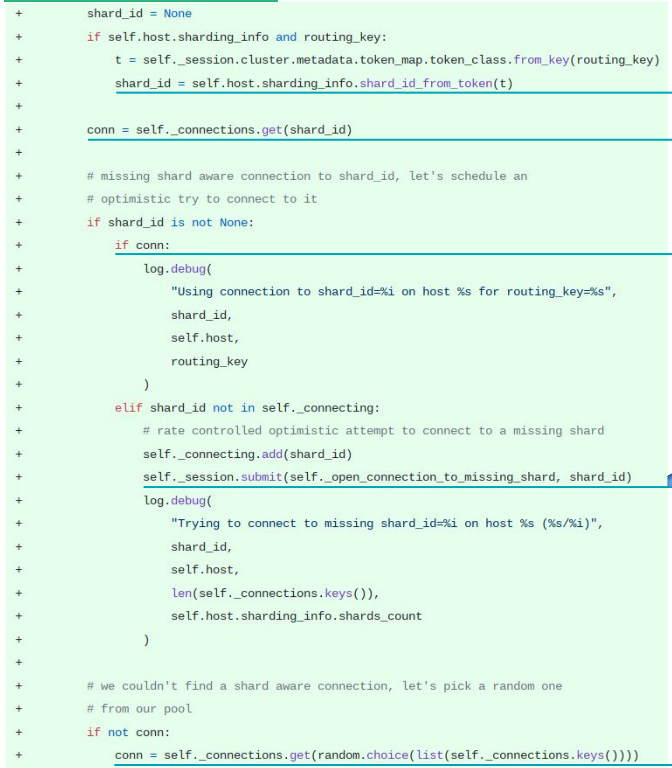
Pure Python
429.0309897623956 nsec per call

Cython
63.073349883779876 nsec per call

Almost 7x faster!

At the heart of scylla-driver's shard-awareness logic

- HostConnection pool selects the connection to the right core to route the query



```
+        shard_id = None
+        if self.host.sharding_info and routing_key:
+            t = self._session.cluster.metadata.token_map.token_class.from_key(routing_key)
+            shard_id = self.host.sharding_info.shard_id_from_token(t)
+
+        conn = self._connections.get(shard_id)
+
+        # missing shard aware connection to shard_id, let's schedule an
+        # optimistic try to connect to it
+        if shard_id is not None:
+            if conn:
+                log.debug(
+                    "Using connection to shard_id=%i on host %s for routing_key=%s",
+                    shard_id,
+                    self.host,
+                    routing_key
+                )
+            elif shard_id not in self._connecting:
+                # rate controlled optimistic attempt to connect to a missing shard
+                self._connecting.add(shard_id)
+                self._session.submit(self._open_connection_to_missing_shard, shard_id)
+                log.debug(
+                    "Trying to connect to missing shard_id=%i on host %s (%s/%i)",
+                    shard_id,
+                    self.host,
+                    len(self._connections.keys()),
+                    self.host.sharding_info.shards_count
+                )
+
+        # we couldn't find a shard aware connection, let's pick a random one
+        # from our pool
+        if not conn:
+            conn = self._connections.get(random.choice(list(self._connections.keys())))
```

Calculate shard id from query routing_key token

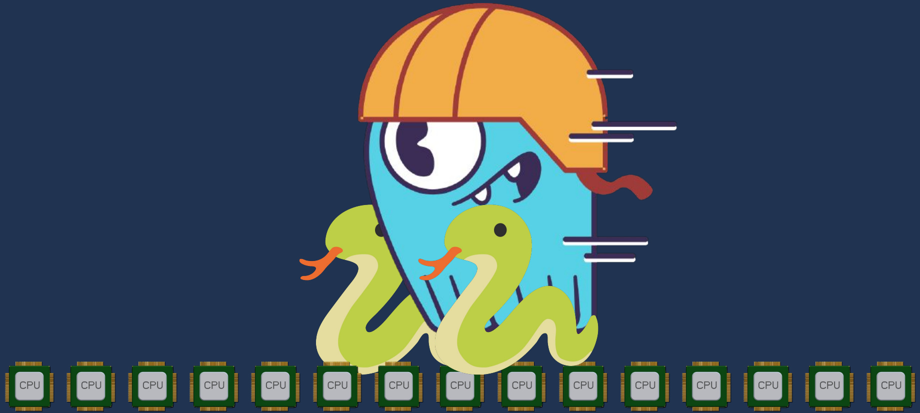
Try to find a connection to the right shard id/core

Use our **direct connection to the right core** to route the query!

No connection to the right core yet, asynchronously try to get one

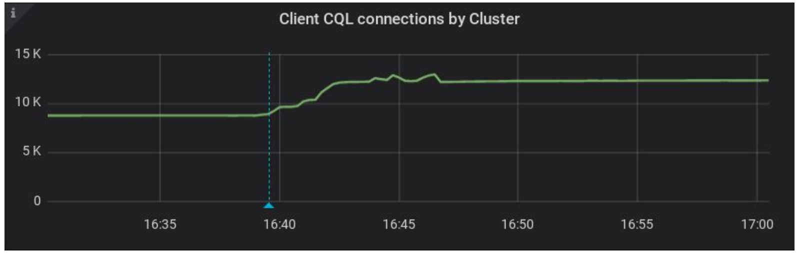
There was no connection to the right core, pick a random one #legacy

Python shard-aware driver
expectations & production results

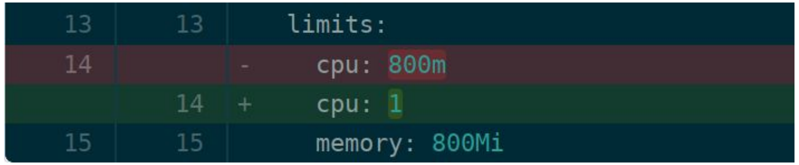


scylla-driver expectations checks

- **1 connection per core per node**
  - Number of cores on node times more connections open to each cluster node ✅
    - Production real-time processing rolling update effect:

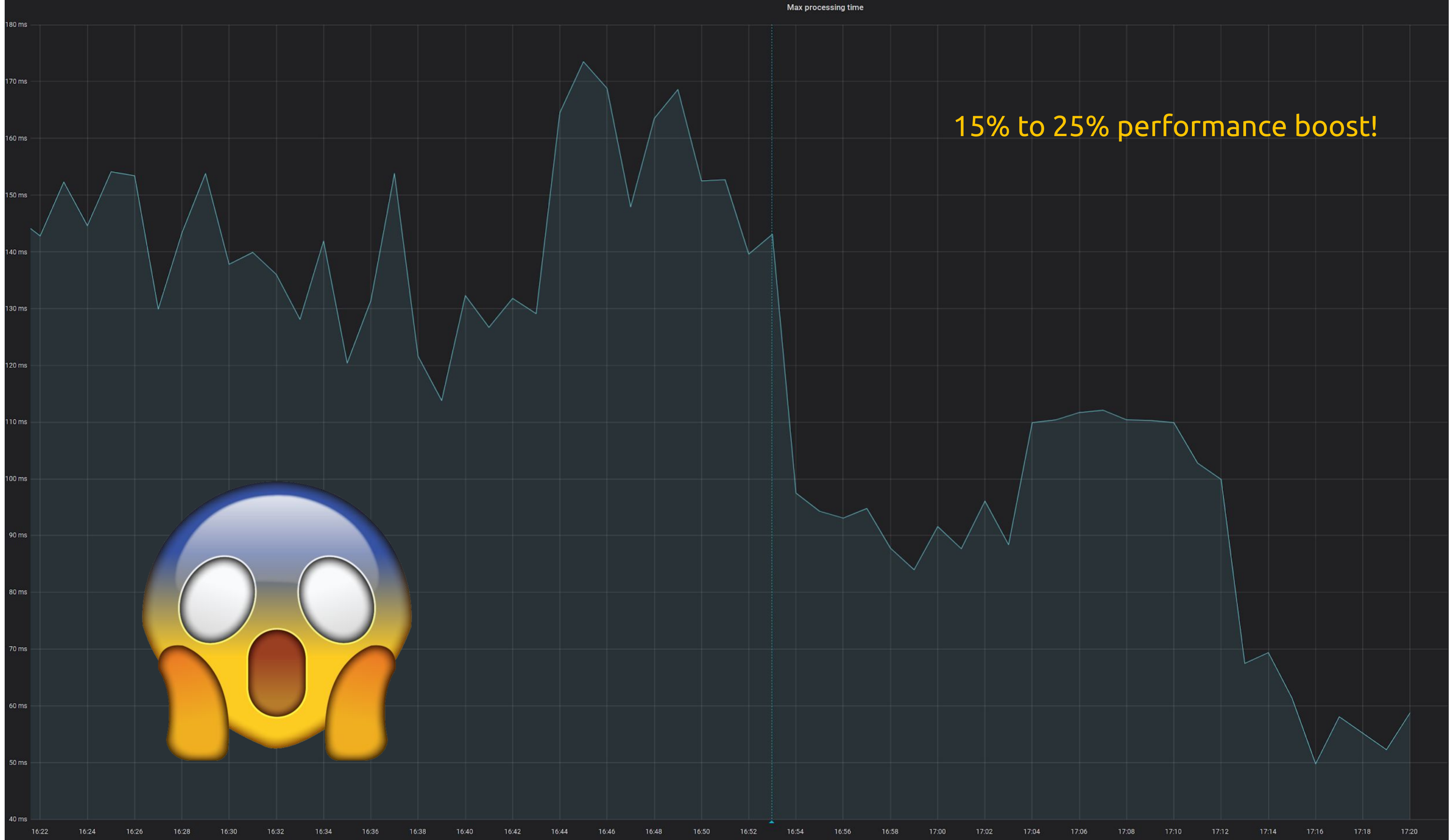


  - More CPU requirements to handle/keepalive more connections ✅
    - Production Kubernetes resources adjustment to avoid pod CPU saturation / throttling

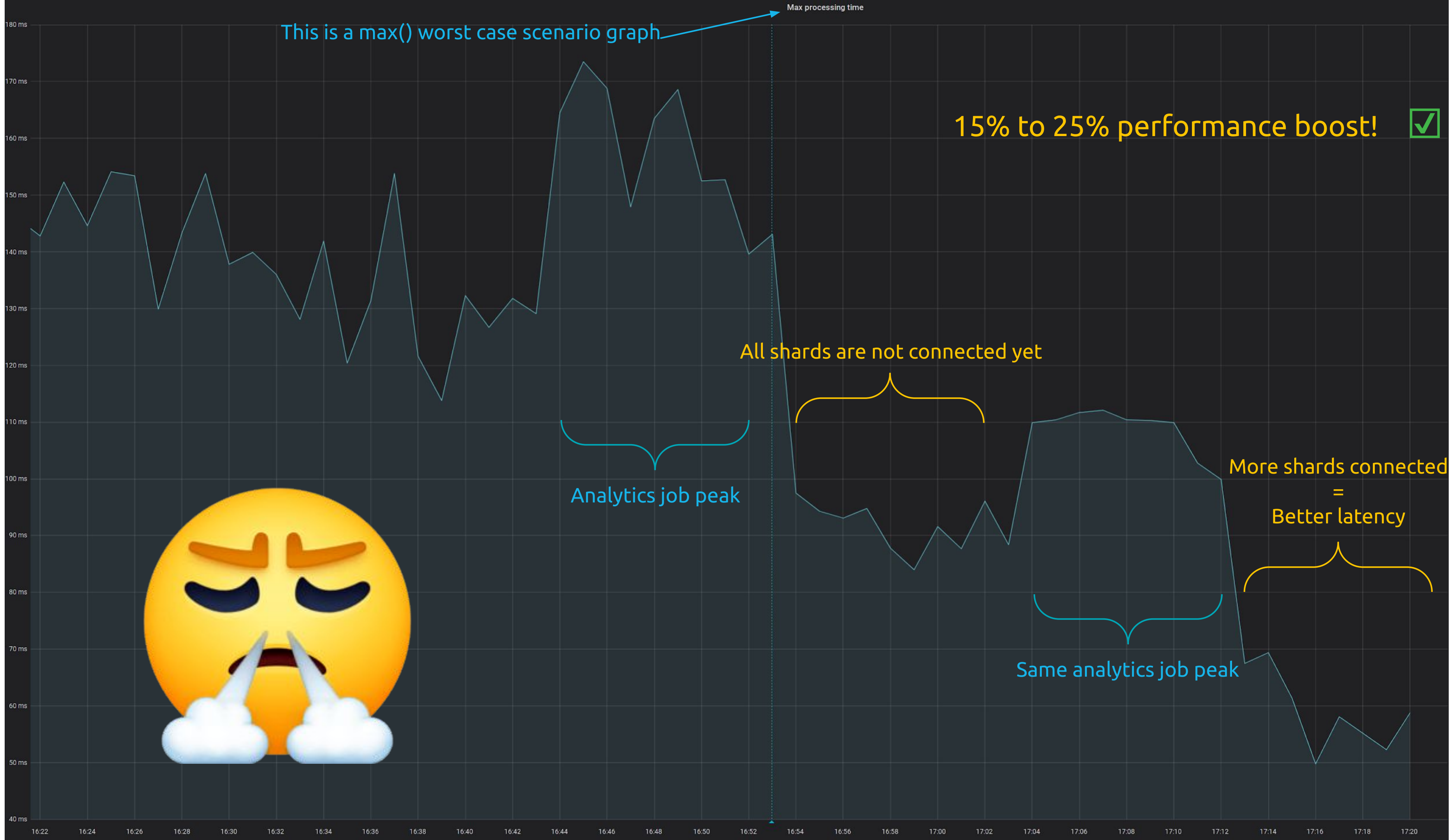


```
        limits:
-         cpu: 800m
+         cpu: 1
          memory: 800Mi
```

- **Routing queries to the right core of the right node**
  - **Reduced query latency...**

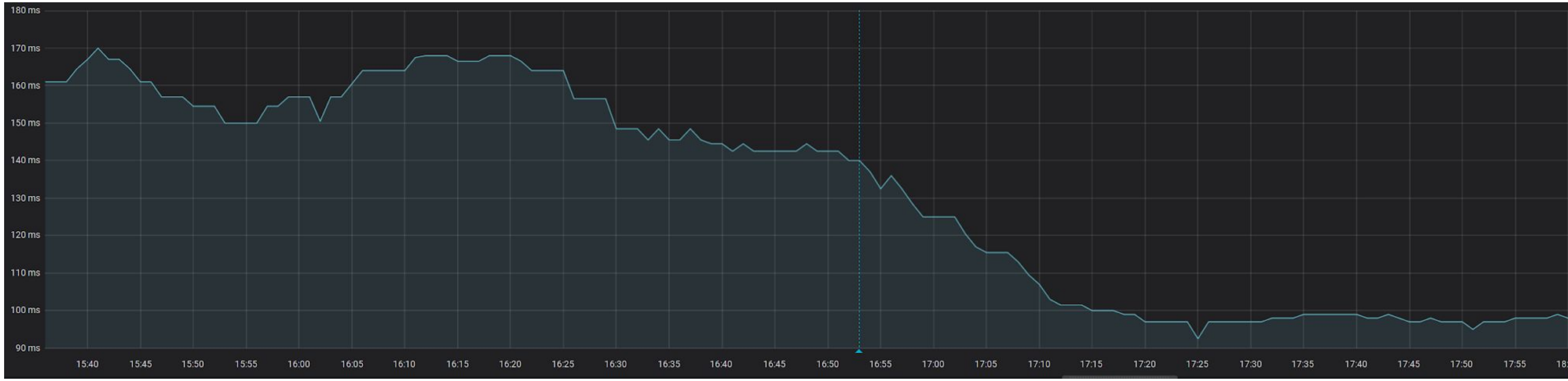


15% to 25% performance boost!

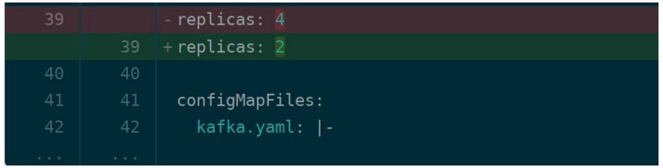

Max processing time
This is a max() worst case scenario graph
15% to 25% performance boost! ✅
All shards are not connected yet
Analytics job peak
Same analytics job peak
More shards connected
=
Better latency
180 ms
170 ms
160 ms
150 ms
140 ms
130 ms
120 ms
110 ms
100 ms
90 ms
80 ms
70 ms
60 ms
50 ms
40 ms
16:22
16:24
16:26
16:28
16:30
16:32
16:34
16:36
16:38
16:40
16:42
16:44
16:46
16:48
16:50
16:52
16:54
16:56
16:58
17:00
17:02
17:04
17:06
17:08
17:10
17:12
17:14
17:16
17:18
17:20

scylla-driver shard-awareness is awesome!

- **movingMedian(max(processing_time), “15min”)**



- **Unexpected (and cool) side effect**
  - Reduced Scylla cluster load + reduced client latency = reduced resources on Kubernetes for the same workload!



```
-replicas: 4
+replicas: 2

 configMapFiles:
   kafka.yaml: |-
...
```

scylla-driver recent & upcoming enhancements

Recent additions: shard-aware capability and connection statistics helpers

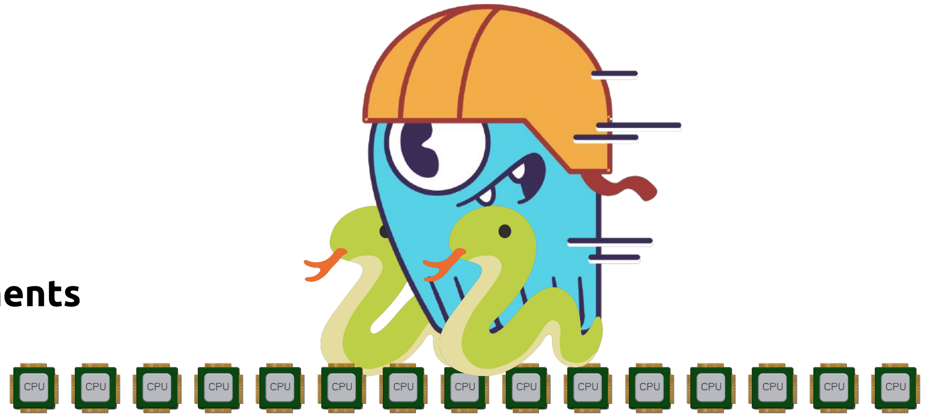
```
from cassandra.cluster import Cluster

cluster = Cluster()
session = cluster.connect()

if cluster.is_shard_aware():
    print("connected to a scylla cluster")

stats = cluster.shard_aware_stats()
if all([v["shards_count"] == v["connected"] for v in stats.values()]):
    print("successfully connected to all shards of all scylla nodes")
```

Use shard capable ports on Scylla when available

- scylla/pull/6781
- scylladb/python-driver/pull/54

Improve Scylla specific documentation

Merge & rebase latest cassandra-driver improvements



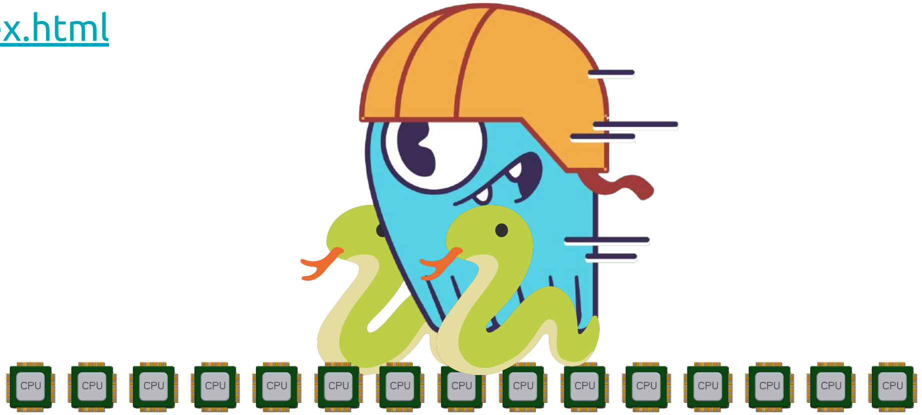
$ pip install scylla-driver

Repository
https://github.com/scylladb/python-driver

PyPi
https://pypi.org/project/scylla-driver/

Documentation
https://scylladb.github.io/python-driver/master/index.html

Chat with us on ScyllaDB users Slack #pythonistas
https://slack.scylladb.com/



Thanks for attending and making this EuroPython a success!

Catch me online: @ultrabug

Discord talk channel
Late questions, deep-dive remarks? Let's keep in touch :)

BRIAN BREAKOUTS
#talk-cassandra-scylla-drivers

Discord Numberly channel
Sponsor talk session tomorrow, **Friday July 24th at 12:00 CEST**

- Real-world experience sharing
- Open Source creations & contributions overview
- Conference talks experience, updates and feedbacks

SPONSOR EXHIBIT
#numberly