Anatomy of

DEVIEW
2015

presto

유동민

TREASURE DATA

Presto Anatomy

DEVIEW 2015

1. Who
2. What
3. Library
4. Code Generation
5. Plugin
6. Future

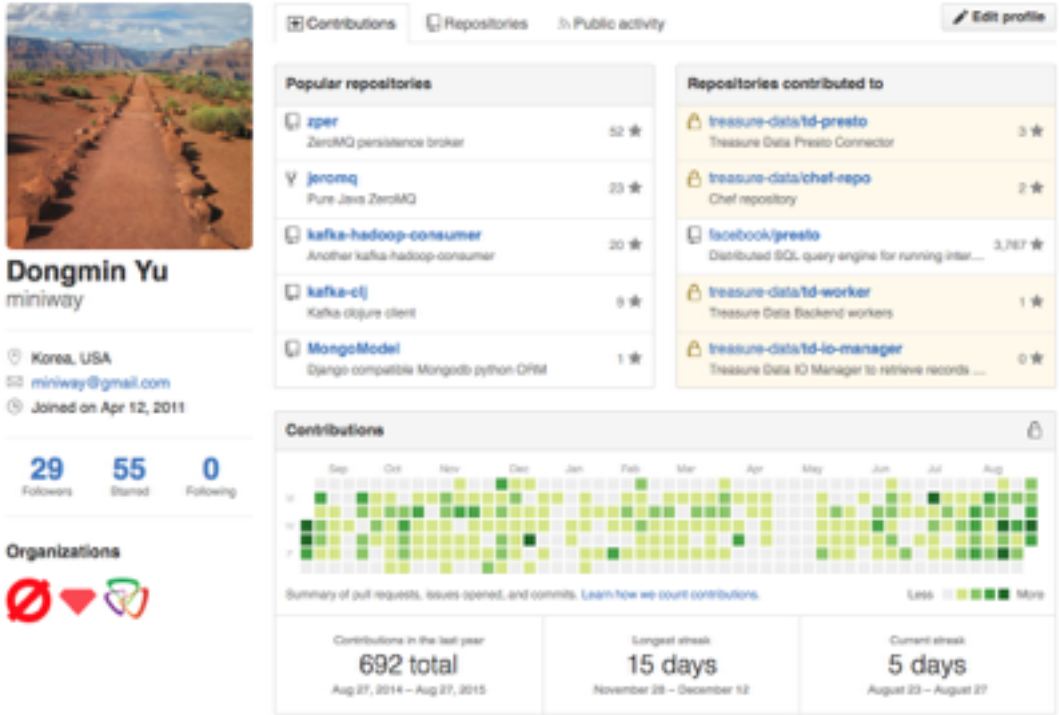
Who am I

Who uses Presto

Who am I

DEVIEW 2015

ZeroMQ Committer, Presto Contributor



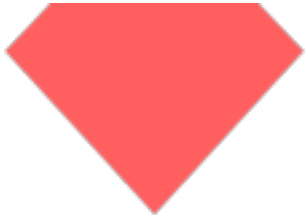
Who am I

DEVIEW 2015

49 Contributors
7 Facebook Developer

Who uses Presto

DEVIEW 2015



TREASURE DATA

Who uses Presto

DEVIEW 2015

@ Facebook / daily

Scan PBs (ORC?)

Trillions Rows

30K Queries

1000s Users

@ Netflix / daily

Scan 15+ PBs (Parquet)

2.5K Queries

300 Users

1 coordinator / r3.4xlarge

220 workers / r3.4xlarge

@ TD / daily

Scan PBs (A Columnar)

Billions Rows

18K Queries

2 coordinators / r3.4xlarge

26 Workers / c3.8xlarge

http://www.slideshare.net/dain1/presto-meetup-2015
http://www.slideshare.net/NezihYigitbasi/prestoatnetflixhadoopsummit15

Who uses Presto – Airbnb

Airpal – a web-based, query execution tool

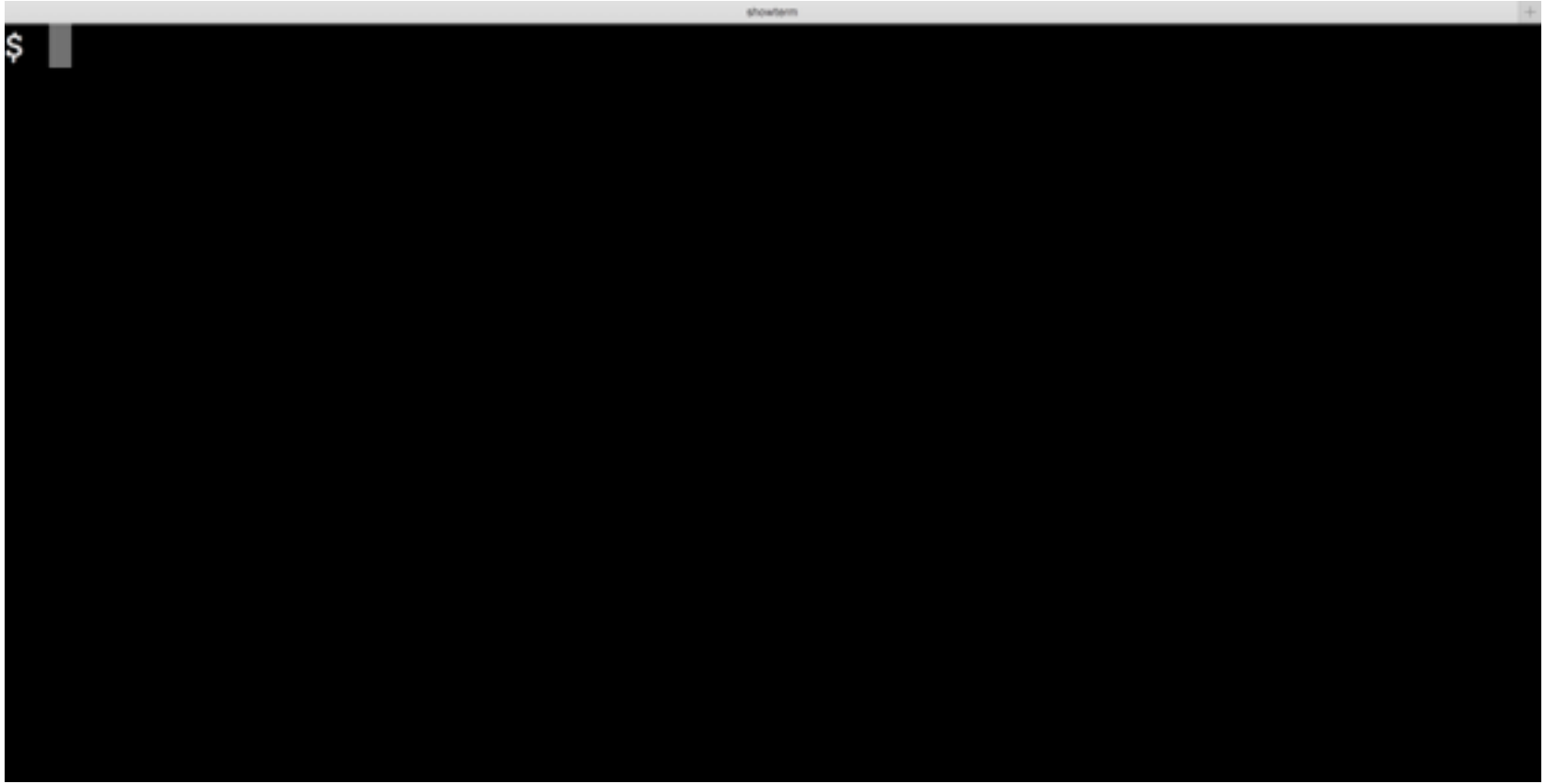
Presto is amazing. It's an order of magnitude faster than Hive in most our use cases. It reads directly from HDFS, so unlike Redshift, there isn't a lot of ETL before you can use it. It just works.

– Christopher Gutierrez, Manager of Online Analytics, Airbnb

What is Presto

Presto – Command Line

DEVIEW 2015



Presto is

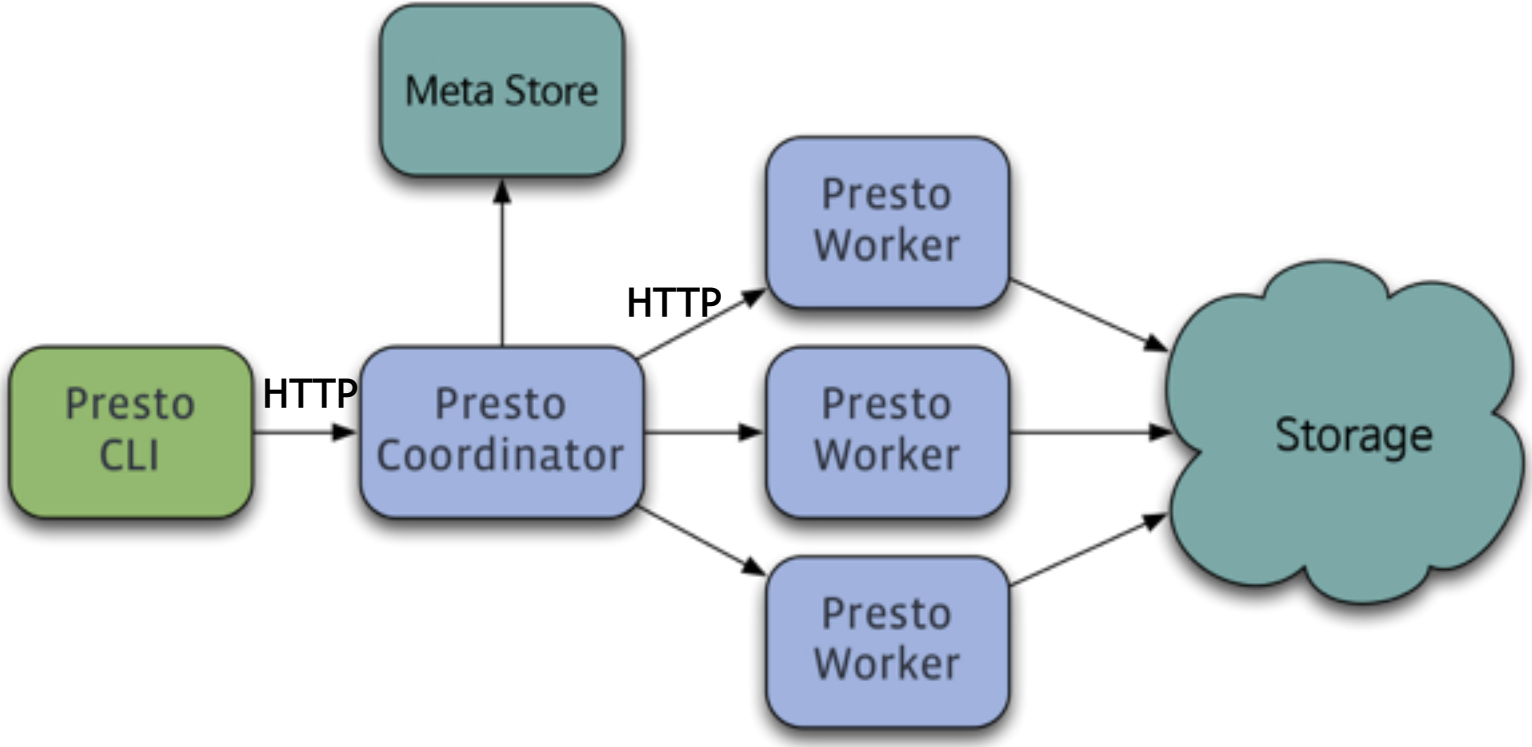
An open source distributed SQL query engine
for running interactive analytic queries
against data sources of all sizes
ranging from gigabytes to petabytes.

http://prestodb.io

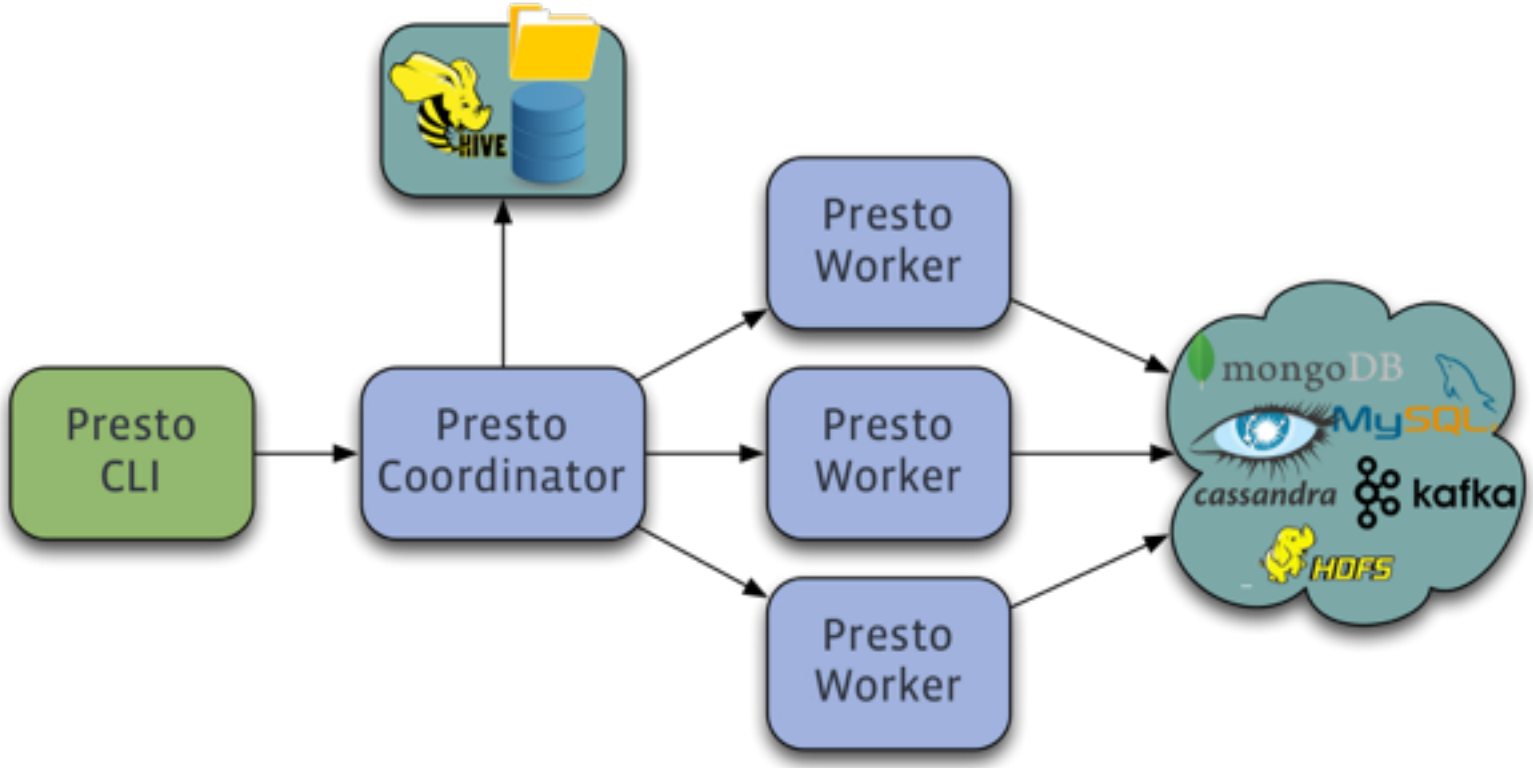
Presto is

- Fast !!! (10x faster than Hive)
- Even faster with new Presto ORC reader
- Written in Java with a pluggable backend
- Not SQL-like, but ANSI-SQL
- Code generation like LLVM
- Not only source is open, but open sourced (No private branch)

Presto is


Meta Store
Presto Worker
HTTP
Presto CLI
HTTP
Presto Coordinator
Presto Worker
Storage
Presto Worker

Presto is


HIVE
Presto CLI
Presto Coordinator
Presto Worker
Presto Worker
Presto Worker
mongoDB
MySQL
cassandra
kafka
HDFS

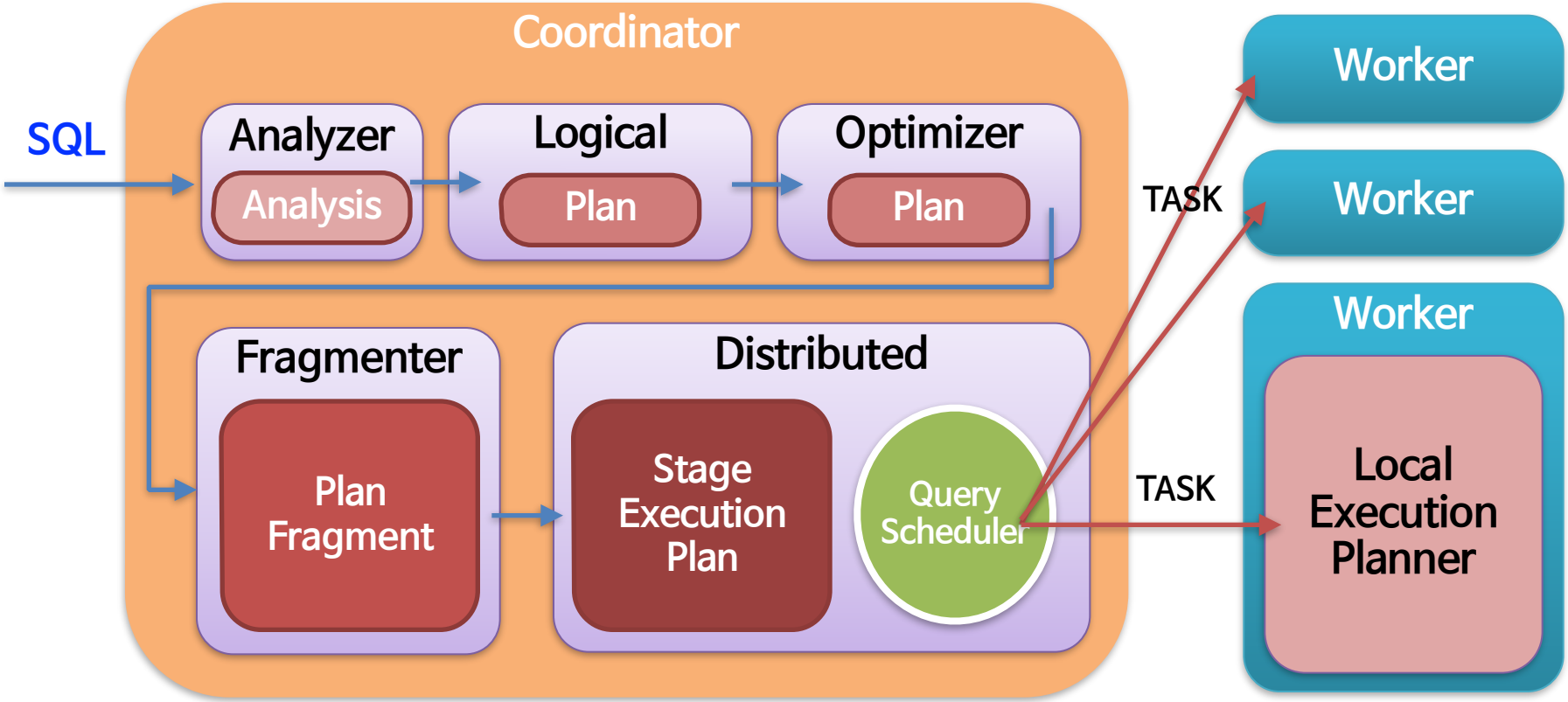
Presto is

DEVIEW 2015

```
CREATE TABLE mysql.hello.order_item AS
SELECT o.*, i.*
FROM hive.world.orders o — TABLESAMPLE SYSTEM (10)
JOIN mongo.deview.lineitem i — TABLESAMPLE BERNOULLI (40)
  ON o.orderkey = i.orderkey
WHERE conditions..
```

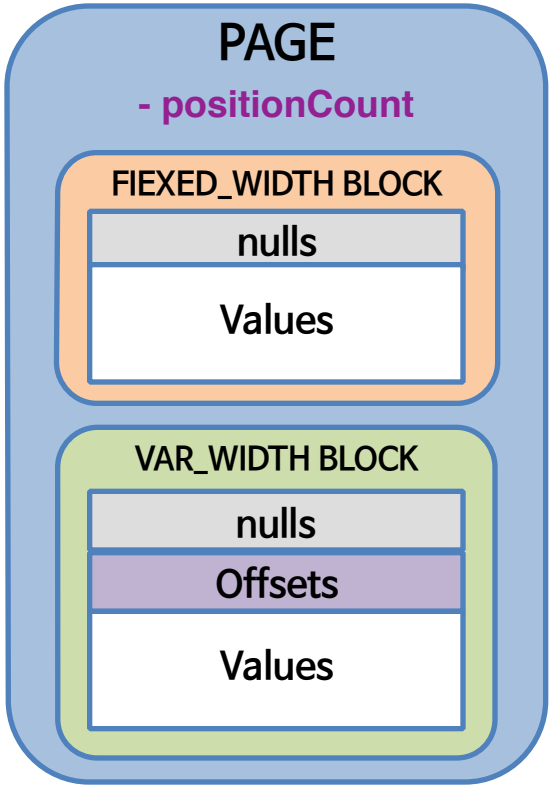
Presto – Planner

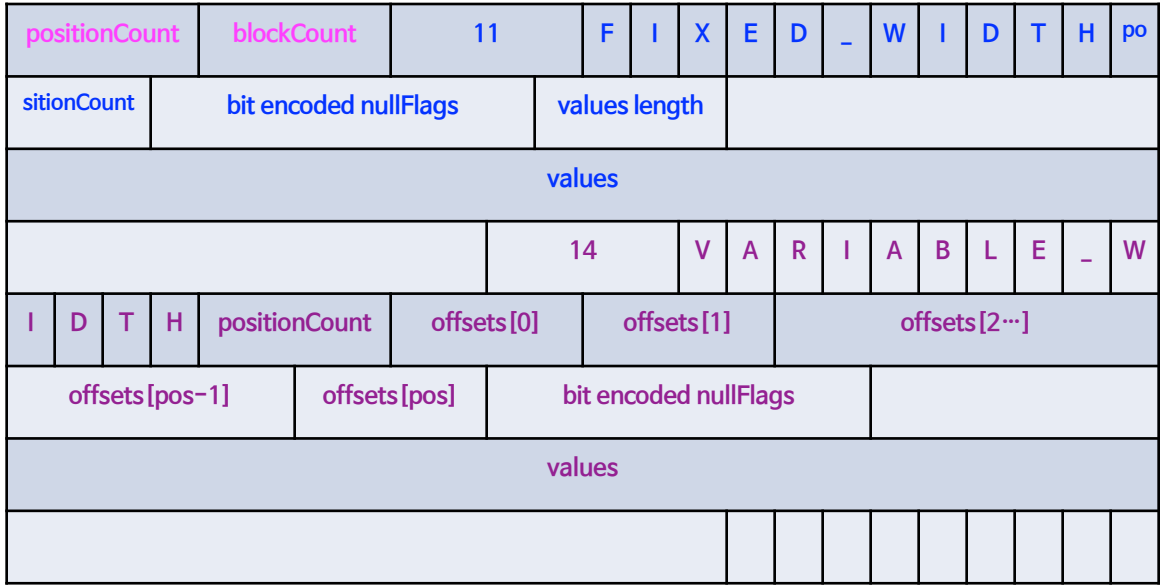
DEVIEW 2015



Presto – Page

DEVIEW 2015

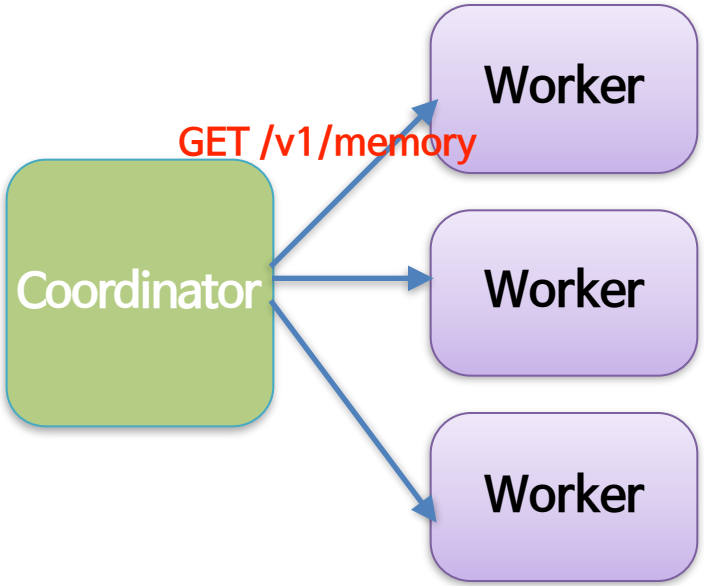


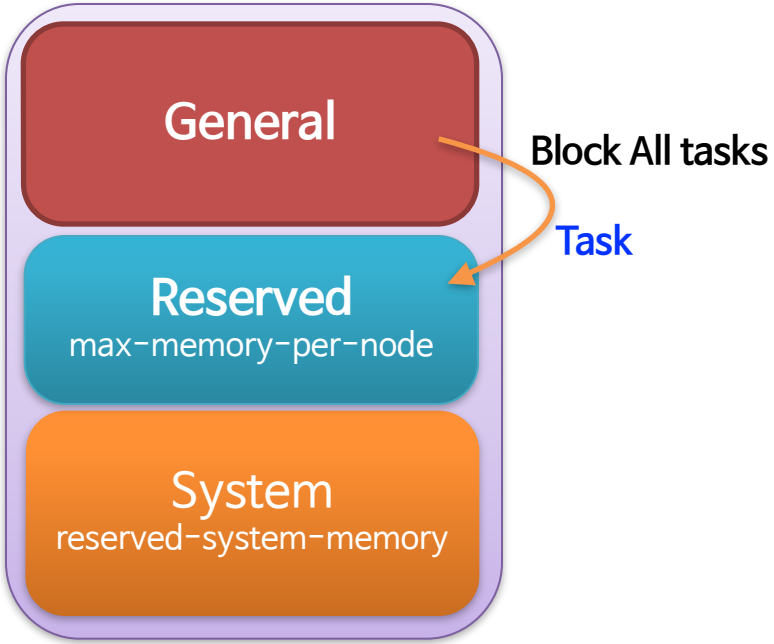


Page / Block serialization

Presto – Cluster Memory Manager

DEVIEW 2015





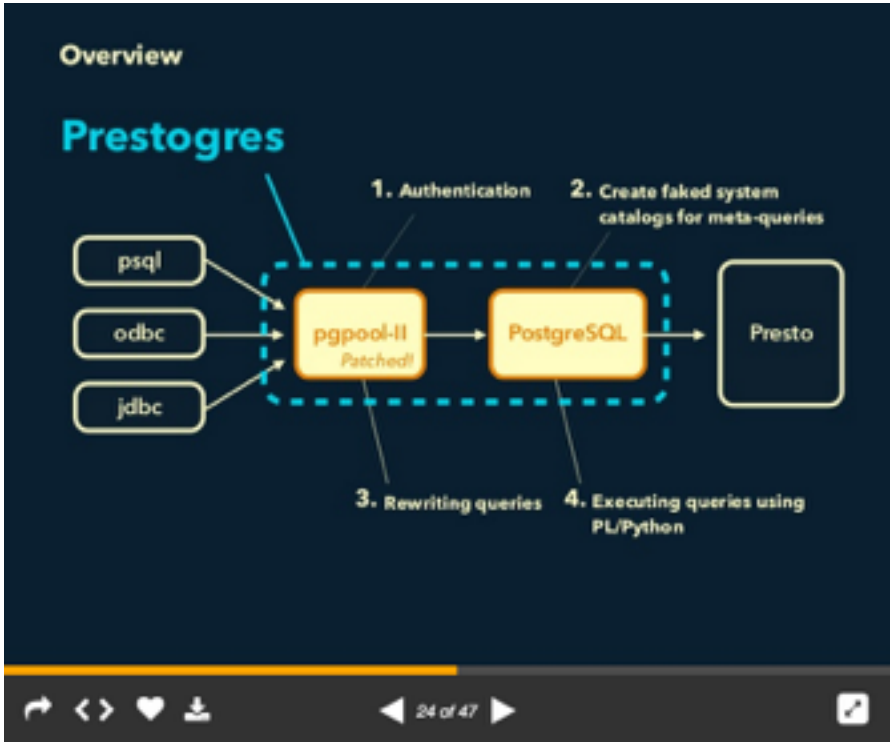
@Config("query.max-memory") = 20G
@Config("query.max-memory-per-node") = 1G
@Config("resources.reserved-system-memory") = 40% of -Xmx

Presto - IndexManager

- LRU worker memory Cache
- @Config(task.max-index-memory) = 64M
- Table (List<IndexColumn> columns)
- Good for Key / Constant tables

Presto – Prestogres

DEVIEW 2015



- Clients to use PostgreSQL protocol to run queries on Presto
- Modified pgpoll-ii
- No ODBC driver yet
- Supports Tableau, ChartIO and etc

https://github.com/treasure-data/prestogres

Presto – Security

DEVIEW 2015

- Authentication
  - Single Sign On – Kerberos, SSL client cert
- Authorization
  - Simple allow / deny

Presto – Misc.

- Presto Verifier
- Presto Benchmark Driver
- Query Queue
  - ${USER} and ${SOURCE} based maxConcurrent, maxQueued
- JDBC Driver
- Python / Ruby Presto Client (Treasure Data)
- Presto Metrics (Treasure Data)
  - GET /v1/jmx/mbean/{mbean}, /v1/query/{query}, /v1/node/
- Presto Python/Java Event Collector (Treasure Data)
  - QueryStart, SpitCompletion, QueryCompletion


1.27 function_not_found
1.00 invalid_function_argument
1.00 invalid_session_property
1.00 user_canceled
Presto: Recent Drivers
No Data
Presto: Failure Rate (5min.)
Query stalled time (sec.)
Presto: Started Queries Rate (5min.)
Presto: Planning Queries
Presto: Queued Query
Presto: Running Queries
Presto: Finished queries
Presto: Rebounded Workers
Presto: Failed Queries
Presto: Canceled Queries
Presto: Average Query Response Time
Presto: Max Response Time
Presto: Queued Driver
Presto: Running Drivers
Presto: Completed Drivers
Presto: Pending Splits
Presto: Blocked Splits
Presto: Swap Usage (%)
Presto: Off-Heap Memory Usage
Coordinator CPU Usage (system)
Coordinator CPU Usage (user)
Presto: Coordinator JVM Memory Usage
Presto: Coordinator: net.bytes_*
Worker Network Sent
Worker CPU usage(system)
Worker CPU usage (user)
Worker Memory Usage (JVM)
Worker Memory Usage (system)
Presto: StorageManager - directAlloc
presto.task_executor.numTasks

Library

Slice – Efficient memory accessor

DEVIEW 2015

- https://github.com/airlift/slice
- ByteBuffer is slow
- Slices.allocate(size), Slices.allocateDirect(size)
- Slices.wrappedBuffer(byteBuffer)
- sun.misc.Unsafe
  - Address
    - ((DirectBuffer) byteBuffer).getAddress()
    - unsafe.ARRAY_BYTE_OFFSET + byteBuffer.arrayOffset()
  - unsafe.getLong(address), unsafe.getInt(address),
  - unsafe.copyMemory(src, address, dst, address)

Airlift – Distributed service framework

DEVIEW 2015

- https://github.com/airlift/airlift
- Core of Presto communication
  - HTTP
  - Bootstrap
  - Node discovery
  - RESTful API
  - Dependency Injection
  - Configuration
  - Utilities

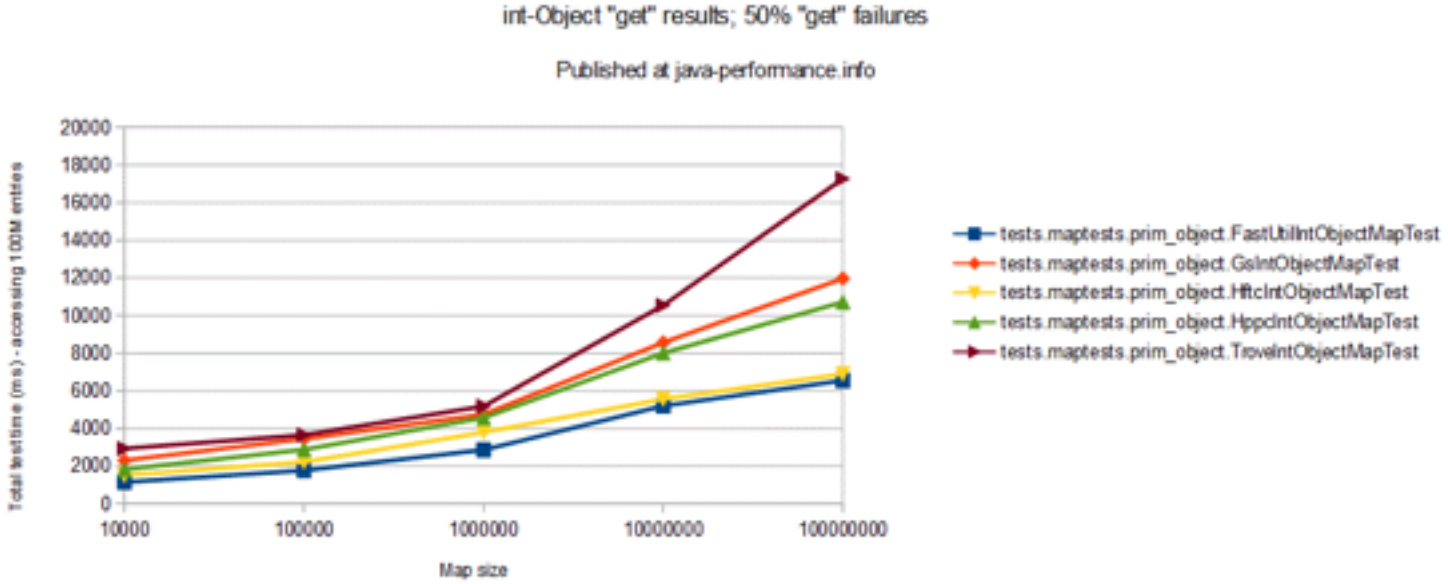
ex) https://github.com/miniway/presto-event-collector

```
@Path("/v2/event")
public class EventResource {
  @POST
  public Response createQuery(EventRequests events) {
    ...
  }
}
public class CollectorMainModule implements ConfigurationAwareModule {
   @Override
   public synchronized void configure(Binder binder) {
      discoveryBinder(binder).bindHttpAnnouncement("collector");
      jsonCodecBinder(binder).bindJsonCodec(EventRequest.class);
      jaxrsBinder(binder).bind(EventResource.class);
   }
   public static void main(String[] args) {
     Bootstrap app = new Bootstrap(ImmutableList.of(
          new NodeModule(),
          new DiscoveryModule(),
          new HttpServerModule(),
          new JsonModule(), new JaxrsModule(true),
          new EventModule(),
          new CollectorMainModule()
     ));
     Injector injector = app.strictConfig().initialize();
     injector.getInstance(Announcer.class).start();
   }
}
```

Fastutil – Fast Java collection

DEVIEW 2015

- FastUtil 6.6.0 turned out to be consistently fast.
- Koloboke is getting second in many tests.
- GS implementation is good enough, but is slower than FastUtil and Koloboke.



http://java-performance.info/hashmap-overview-jdk-fastutil-goldman-sachs-hppc-koloboke-trove-january-2015/

ASM – Bytecode manipulation

DEVIEW 2015

```
package pkg;

public interface SumInterface {
   long sum(long value);
}

public class MyClass implements SumInterface {
   private long result = 0L;

   public MyClass(long value) {
      result = value;
   }

   @Override
   public long sum(long value) {
      result += value;
      return result;
   }
}
```

```
ClassWriter cw = new ClassWriter(0);
cw.visit(V1_7, ACC_PUBLIC,
         "pkg/MyClass", null,
         "java/lang/Object",
         new String[] { "pkg/SumInterface" });

cw.visitField(ACC_PRIVATE,
              "result", "J", null, new Long(0));

// constructor
MethodVisitor m = cw.visitMethod(ACC_PUBLIC,
                 "<init>", "(J)V", null, null);
m.visitCode();

// call super()
m.visitVarInsn(ALOAD, 0);  // this
m.visitMethodInsn(INVOKESPECIAL,
                 "java/lang/Object", "<init>", "()V",
                  false);
```

ASM – Bytecode manipulation (Cont.)

DEVIEW 2015

```
// this.result = value
m.visitVarInsn(ALOAD, 0); // this
m.visitVarInsn(LLOAD, 1 ); // value
m.visitFieldInsn(PUTFIELD,
                 "pkg/MyClass", "result", "J");
m.visitInsn(RETURN);
m.visitMaxs(-1, -1).visitEnd();

// public long sum(long value)
m = cw.visitMethod(ACC_PUBLIC , "sum", "(J)J",
                   null, null);
m.visitCode();

m.visitVarInsn(ALOAD, 0); // this
m.visitVarInsn(ALOAD, 0); // this
m.visitFieldInsn(GETFIELD,
                 "pkg/MyClass", "result", "J");
m.visitVarInsn(LLOAD, 1); // value
```

```
// this.result + value
m.visitInsn(LADD);
m.visitFieldInsn(PUTFIELD,
                 "pkg/MyClass", "result", "J");

m.visitVarInsn(ALOAD, 0); // this
m.visitFieldInsn(GETFIELD,
                 "pkg/MyClass", "result", "J");
m.visitInsn(LRETURN);
m.visitMaxs(-1, -1).visitEnd();

cw.visitEnd();
byte[] bytes = cw.toByteArray();

ClassLoader.defindClass(bytes)
```

Library – Misc.

- JDK 8u40 +
- Guice – Lightweight dependency injection
- Guava – Replacing Java8 Stream, Optional and Lambda
- ANTLR4 – Parser generator, SQL parser
- Jetty – HTTP Server and Client
- Jackson – JSON
- Jersey – RESTful API

Byte Code Generation

Function Variable Binding

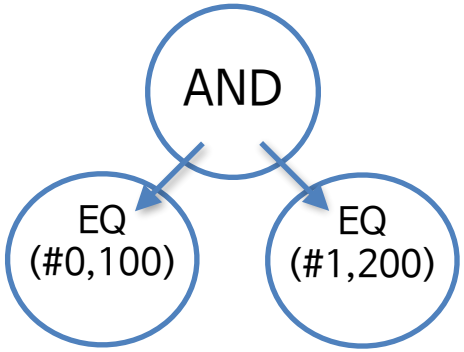
Code Generation

- ASM
- Runtime Java classes and methods generation base on SQL
- 30% of performance gain
- Where
  - Filter and Projection
  - Join Lookup source
  - Join Probe
  - Order By
  - Aggregation

Code Generation – Filter

DEVIEW 2015

```
SELECT * FROM lineitem
    WHERE orderkey = 100 AND quantity = 200
```



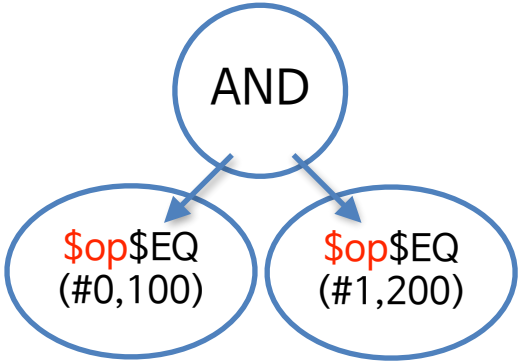
Logical Planner

```
class AndOperator extends Operator {
   private Operator left = new EqualOperator(#1, 100);
   private Operator right = new EqualOperator(#2, 200);
   @Override
   public boolean evaluate(Cursor cur)
   {
      if (!left.evaluate(cur)) {
         return false;
      }
      return right.evaluate(cur);
   }
}
class EqualOperator  extends Operator {
   @Override
   public boolean evaluate(Cursor c)
   {
      return cur.getValue(position).equals(value);
   }
}
```

Code Generation – Filter

DEVIEW 2015

```
@ScalarOperator(EQUAL)
@SqlType(BOOLEAN)
public static boolean equal(@SqlType(BIGINT) long left,
                            @SqlType(BIGINT) long right) {
   return left == right;
}
=> MethodHandle("$operator$EQUAL(long, long): boolean")
```



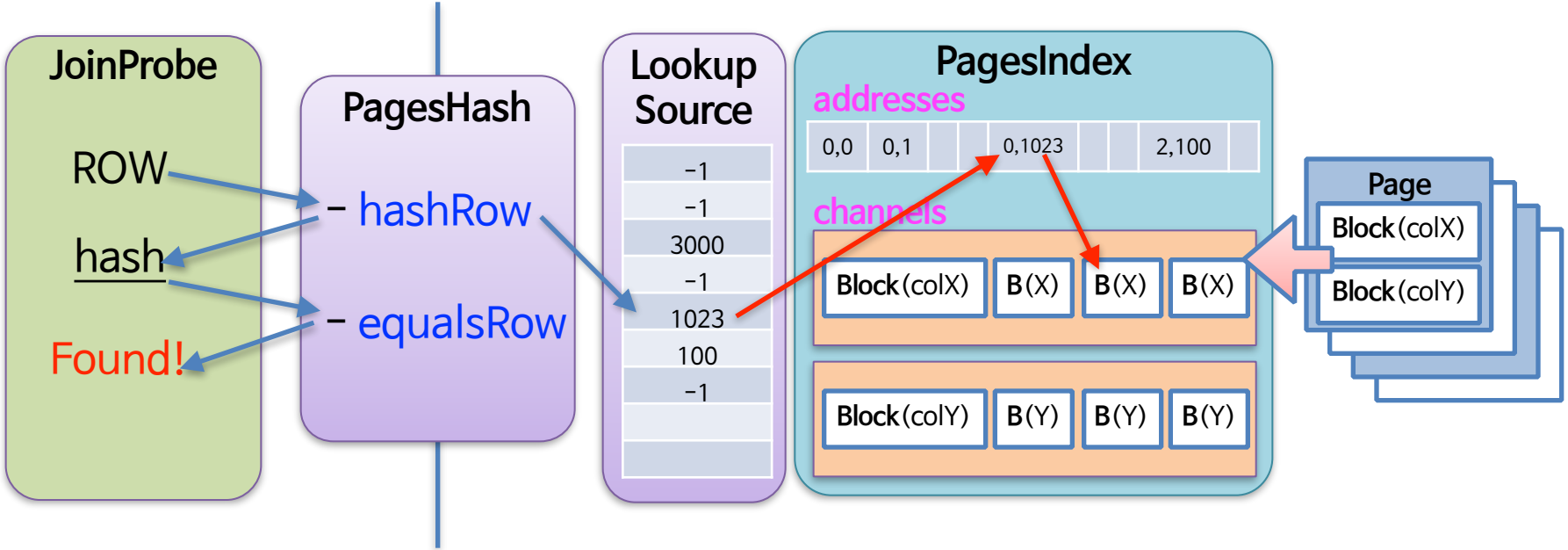
Local Execution Planner

```
// invoke MethodHandle( $operator$EQUAL(#0, 100) )
push cursor.getValue(#0)
push 100
$statck = invokeDynamic boostrap(0) $operator$EQUAL
if (!$stack) { goto end; }

push cursor.getValue(#1)
push 200
$stack = invokeDynamic boostrap(0) $operator$EQUAL

end:
return $stack
```

Code Generation – Join Lookup Source

DEVIEW 2015

```
SELECT col1, col2, col3 FROM tabA JOIN tabB
    ON tabA.col1 = tabB.colX /* BIGINT */ AND tabA.col2 = tabB.colY /* VARCHAR */
```



Code Generation – PageHash

DEVIEW 2015

```
class PagesHash {
   List<Type> types = [BIGINT, VARCHAR]
   List<Integer> hashChannels = [0,1]
   List<List<Block>> channels = [
            [BLOCK(colX), BLOCK(colX), ...] ,
            [BLOCK(colY), BLOCK(colY), ...]
   ]
}
```

```
class (Compiled)PageHash {
    Type type_colX = BIGINT
    Type type_colY = VARCHAR
    int hashChannel_colX = 0
    int hashChannel_colY = 1
    List<Block> channel_colX =
            [ BLOCK(colX), BLOCK(colX), ··· ]
    List<Block> channel_colY =
            [ BLOCK(colY), BLOCK(colY), ··· ]
}
```

Code Generation – PageHash (Cont.)

DEVIEW 2015

```
long hashRow (int position,
                    Block[] blocks) {
  int result = 0;
  for (int i = 0; i < hashChannels.size(); i++) {
    int hashChannel = hashChannels.get(i);
    Type type = types.get(hashChannel);

    result = result * 31 +
              type.hash(blocks[i], position);
  }
  return result;
}
```

```
long (Compiled)hashRow (int position,
                              Block[] blocks) {
    int result = 0;
    result = result * 31 +
          type_colX.hash(block[0], position);
    result = result * 31 +
          type_colY.hash(block[1], position);
    return result;
}
```

Code Generation – PageHash (Cont.)

DEVIEW 2015

```
boolean equalsRow (
        int leftBlockIndex, int leftPosition,
        int rightPosition, Block[] rightBlocks) {
    for (int i = 0; i < hashChannels.size(); i++) {
        int hashChannel = hashChannels.get(i);
        Type type = types.get(hashChannel);
        Block leftBlock =
                    channels.get(hashChannel)
                                .get(leftBlockIndex);
        if (!type.equalTo(leftBlock, leftPosition,
                    rightBlocks[i], rightPosition)) {
            return false;
        }
    }
    return true;
}
```

```
boolean (Compiled)equalsRow (
            int leftBlockIndex, int leftPosition,
            int rightPosition, Block[] rightBlocks) {
    Block leftBlock =
            channels_colX.get(leftBlockIndex);
    if (!type.equalTo(leftBlock, leftPosition,
                rightBlocks[0], rightPosition)) {
            return false;
    }
    leftBlock =
            channels_colY.get(leftBlockIndex);
    if (!type.equalTo(leftBlock, leftPosition,
                rightBlocks[1], rightPosition)) {
            return false;
    }
    return true;
}
```

Method Variable Binding

DEVIEW 2015

1. regexp_like(string, pattern) → boolean
2. regexp_like(string, cast(pattern as RegexType)) // OperatorType.CAST
3. regexp_like(string, new Regex(pattern))
4. MethodHandle handle = MethodHandles.insertArgument(1, new Regex(pattern))
5. handle.invoke (string)

```
@ScalarOperator(OperatorType.CAST)
@SqlType("RegExp")
public static Regex castToRegexp(@SqlType(VARCHAR) Slice pattern){
  return new Regex(pattern.getBytes(), 0, pattern.length());
}
@ScalarFunction
@SqlType(BOOLEAN)
public static boolean regexpLike(@SqlType(VARCHAR) Slice source,
                                 @SqlType("RegExp") Regex pattern){
   Matcher m = pattern.matcher(source.getBytes());
   int offset = m.search(0, source.length());
   return offset != -1;
}
```

Plugin – Connector

Plugin

- Hive
  - Hadoop 1, Hadoop2, CDH4, CDH 5
- MySQL
- PostgreSQL
- Cassandra
- MongoDB
- Kafka
- Raptor
- Machine Learning
- BlackHole
- JMX
- TPC-H
- Example

Plugin – Raptor

- Storage data in flash on the Presto machines in ORC format
- Metadata is stored in MySQL (Extendable)
- Near real-time loads (5 – 10mins)
- 3TB / day, 80B rows/day , 5 secs query
- CREATE VIEW myview AS SELECT …
- DELETE FROM tab WHERE conditions…
- UPDATE (Future)
- Coarse grained Index : min / max value of all columns
- Compaction
- Backup Store (Extendable)

No more  ?!

Plugin - How to write

DEVIEW 2015

- https://prestodb.io/docs/current/develop/spi-overview.html
- ConnectorFactory
  - ConnectorMetadata
  - ConnectorSplitManager
  - ConnectorHandleResolver
  - ConnectorRecordSetProvider (PageSourceProvider)
  - ConnectorRecordSinkProvider (PageSinkProvider)
- Add new Type
- Add new Function (A.K.A UDF)

Plugin – MongoDB

DEVIEW 2015

- https://github.com/facebook/presto/pull/3337
- 5 Non-business days
- Predicate Pushdown
- Add a Type (ObjectId)
- Add UDFs (objectid(), objectid(string))

```
public class MongoPlugin implements Plugin {
    @Override
    public <T> List<T> getServices(Class<T> type) {
        if (type == ConnectorFactory.class) {
            return ImmutableList.of(
                    new MongoConnectorFactory(…));
        } else if (type == Type.class) {
            return ImmutableList.of(OBJECT_ID);
        } else if (type == FunctionFactory.class) {
            return ImmutableList.of(
                    new MongoFunctionFactory(typeManager));
        }
        return ImmutableList.of();
    }
}
```

Plugin – MongoDB

DEVIEW 2015

```
class MongoFactory implements ConnectorFactory {
  @Override
  public Connector create(String connectorId) {
    Bootstrap app = new Bootstrap(new
MongoClientModule());
    return app.initialize()
                .getInstance(MongoConnector.class);
  }
}

class MongoClientModule implements Module {
  @Override
  public void configure(Binder binder) {
    binder.bind(MongoConnector.class)
          .in(SINGLETON);
    ...
    configBinder(binder)
          .bindConfig(MongoClientConfig.class);
  }
}
```

```
class MongoConnector implements Connector {
  @Inject
  public MongoConnector(
      MongoSession mongoSession,
      MongoMetadata metadata,
      MongoSplitManager splitManager,
      MongoPageSourceProvider
                pageSourceProvider,
      MongoPageSinkProvider
                pageSinkProvider,
      MongoHandleResolver
                handleResolver) {
    ...
  }
}
```

Plugin – MongoDB UDF

DEVIEW 2015

```
public class MongoFunctionFactory
      implements FunctionFactory {
   @Override
   public List<ParametricFunction> listFunctions()
   {
      return new FunctionListBuilder(typeManager)
            .scalar(ObjectIdFunctions.class)
            .getFunctions();
   }
}
public class ObjectIdType
      extends AbstractVariableWidthType {
    ObjectIdType OBJECT_ID = new ObjectIdType();

   @JsonCreator
   public ObjectIdType() {
      super(parseTypeSignature("ObjectId"),
               Slice.class);
   }
}
```

```
public class ObjectIdFunctions {
   @ScalarFunction("objectid")
   @SqlType("ObjectId")
   public static Slice ObjectId() {
      return Slices.wrappedBuffer(
                  new ObjectId().toByteArray());
   }
   @ScalarFunction("objectid")
   @SqlType("ObjectId")
   p.s Slice ObjectId(@SqlType(VARCHAR) Slice value) {
      return Slices.wrappedBuffer(
         new ObjectId(value.toStringUtf8()).toByteArray()
   }
   @ScalarOperator(EQUAL)
   @SqlType(BOOLEAN)
   p.s boolean equal(@SqlType("ObjectId") Slice left,
                     @SqlType("ObjectId") Slice right) {
      return left.equals(right);
   }
}
```

Future

Presto – Future

- Cost based optimization
- Join Hint (Right table must be smaller)
- Huge Join / Aggregation
- Coordinator High Availability
- Task Recovery
- Work Stealing
- Full Pushdown
- Vectorization
- ODBC Driver (Early 2016, Teradata)
- More security (LDAP, Grant SQL)

SELECT question FROM you

Thank You