Drizzle Replication Roadmap

David Shrewsbury
shrewsbury.dave@gmail.com
http://dshrewsbury.blogspot.com
IRC: Shrews

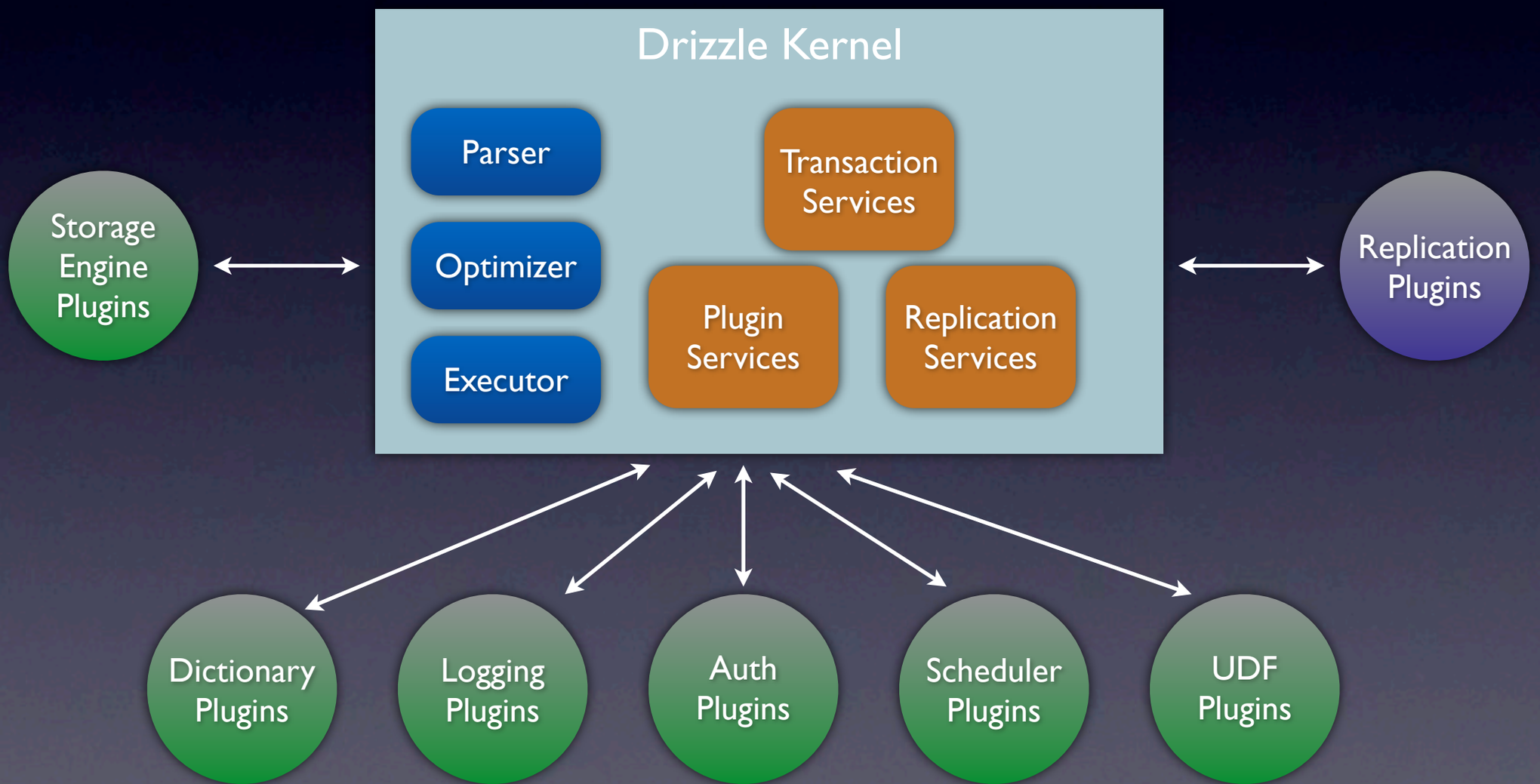
Today's Topics

- Drizzle Replication Architecture
- Current Replication Features
- Future Replication Features
- Replication Demo

Drizzle Replication Architecture

- Nothing like MySQL
- Row-based changes, not statement based
- Google Protobuffers to represent changes in db
- File based or InnoDB based transaction logs
- Replication solutions implemented as plugins

Kernel and Plugins



Kernel Role in Replication

- Creates replication (protobuffer) messages
- Distributes replication messages when complete
- APIs for registered replication plugins

Kernel Role in Replication

- Creates replication (protobuffer) messages
- Distributes replication messages when complete
- APIs for registered replication plugins

What are Protobuf Messages?

- Open source library from Google (http://code.google.com/p/protobuf/)
- Flexible and efficient mechanism for serializing structured data

```
message Person
{
    required int32 id = 1;
    required string name = 2;
    optional string email = 3;
}
```

Message Definition

```
message Person
{
    required int32 id = 1;
    required string name = 2;
    optional string email = 3;
}
```

- Each message contains field names and types.
- Types can be numerical, boolean, string, or raw data, or another message (sub-message).
- Fields can be optional, required, or repeated.

How Are They Used?

- Drizzle's basic unit of work representing server changes
- Messages for transactions, DDL, server events
- Also used to represent various internal objects and metadata: tables, schemas, etc.

Drizzle Message Example

```
message Transaction
{
    required TransactionContext transaction_context = 1;
    repeated Statement statement = 2;
    optional Event event = 3;
    optional uint32 segment_id = 4;
    optional bool end_segment = 5;
}

message Statement
{
    enum Type { ROLLBACK=0; INSERT=1; DELETE=2; ... }
    required Type type = 1;
    required uint64 start_timestamp = 2;
    required uint64 end_timestamp = 3;
    option string sql = 4;   /* May contain the original SQL */
    ....
}
```

Generating Target Source

- Drizzle messages defined in .proto files, then generate source code to manipulate messages
- GPB compiler takes .proto file as input and produces source code for the target language.
- Support for C++, Java, Python, Perl, Ruby, etc.

```
protoc -cpp_out=$PWD transaction.proto
```

Kernel Role in Replication

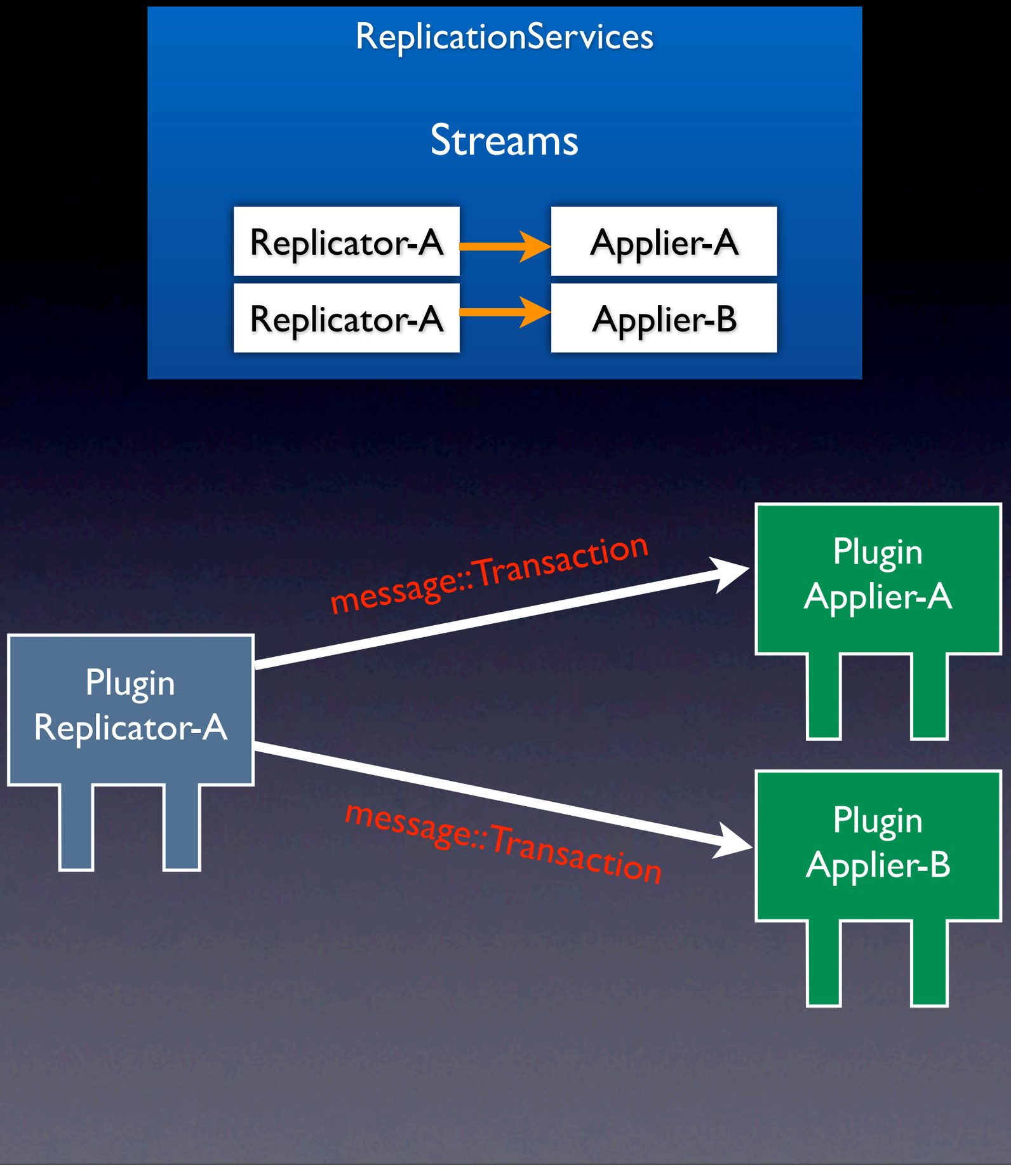
- Creates replication (protobuffer) messages
- Distributes replication messages when complete
- APIs for registered replication plugins

Replication Plugin Types

- *replicator* - Plugin that receives all replication messages from the kernel and transforms them in some useful way.
- *applier* - Plugin that receives messages from a replicator plugin and does some kind of work (log/analyze/transmit) with them.

Replication Streams

- ReplicationServices kernel module maintains a list of replication streams.
- A stream is a *replicator* and *applier* pair
- Kernel → *replicators* → *appliers*


ReplicationServices
Streams
Replicator-A
Applier-A
Replicator-A
Applier-B
message::Transaction
Plugin
Applier-A
Plugin
Replicator-A
message::Transaction
Plugin
Applier-B

Example Replicator

- filtered_replicator plugin
- Filters replication protobuf messages by schema or table name
- To enable the plugin and filter out changes for the *foo* schema:

```
drizzled --plugin-add=filtered_replicator
         --filtered-replicator.filteredschemas=foo
```

Example Appliers

- InnoDB and file-based transaction logs
- By default, each pairs with the `default_replicator` to receive ALL protobuf messages
- Each applies the messages differently, but never modifies the message.

Viewing Replication Streams

- Active replication streams are stored in a DATA_DICTIONARY table

```
drizzle> SELECT * FROM data_dictionary.replication_streams;
+----------------------+--------------------------+
| REPLICATOR           | APPLIER                  |
+----------------------+--------------------------+
| filtered_replicator | transaction_log_applier |
+----------------------+--------------------------+
1 row in set (0.000452 sec)
```

Kernel Role in Replication

- Creates replication (protobuffer) messages
- Distributes replication messages when complete
- APIs for registered replication plugins

replicator API

- Implement `plugin::TransactionReplicator` interface
- Protobuffer message can be modified before passing it along to the applier.

- Only a single method needs to be implemented:

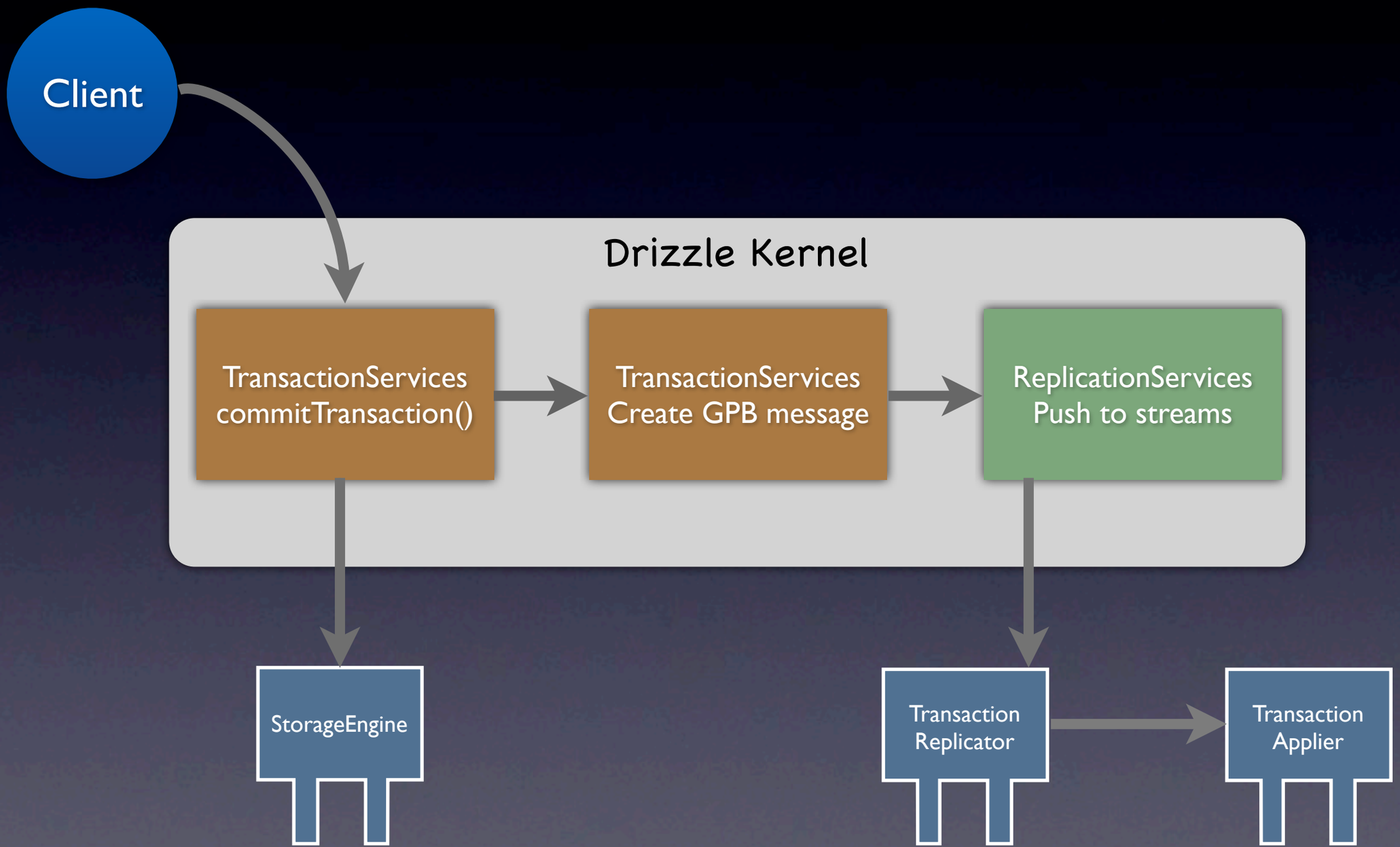
```
/* replicate() should call in_applier->apply() */
virtual ReplicationReturnCode replicate(TransactionApplier *in_applier,
                                        Session &session,
                                        message::Transaction &to_replicate)= 0;
```

applier API

- Implement `plugin::TransactionApplier` interface
- Transaction message cannot be modified.
- Only a single method needs to be implemented:

```
virtual ReplicationReturnCode apply(Session &in_session,
                                    const message::Transaction &to_apply)= 0;
```

Putting it all together



Available Replication Plugins

- Custom replicators (e.g. filtered_replicator)
- Transaction logs
- Replication to other systems

File Based Transaction Log

- Author: Jay Pipes
- Provides a file-based log of compressed, serialized Transaction messages
- Supports checksumming of individual messages
- Flexible file sync behavior
- Can use any available replicator plugin
- Very well tested for correctness

Log Metadata

```
$ ./sbin/drizzle --transaction-log.enable &


drizzle> show tables from data_dictionary like 'trans%';
+-----------------------------------+
| Tables_in_data_dictionary (trans%) |
+-----------------------------------+
| TRANSACTION_LOG                   |
| TRANSACTION_LOG_ENTRIES           |
| TRANSACTION_LOG_TRANSACTIONS      |
+-----------------------------------+
```

InnoDB Based Transaction Log

- Authors: Brian Aker, Joe Daly, Stewart Smith, Andrew Hutchings
- Internal InnoDB table is the transaction log
- Exposed in DATA_DICTIONARY
- Transaction recorded in log table within the same transaction it is recording (always in sync)
- Avoids extra fsync's of the file-based log; no group commit required.
- Very well tested for correctness

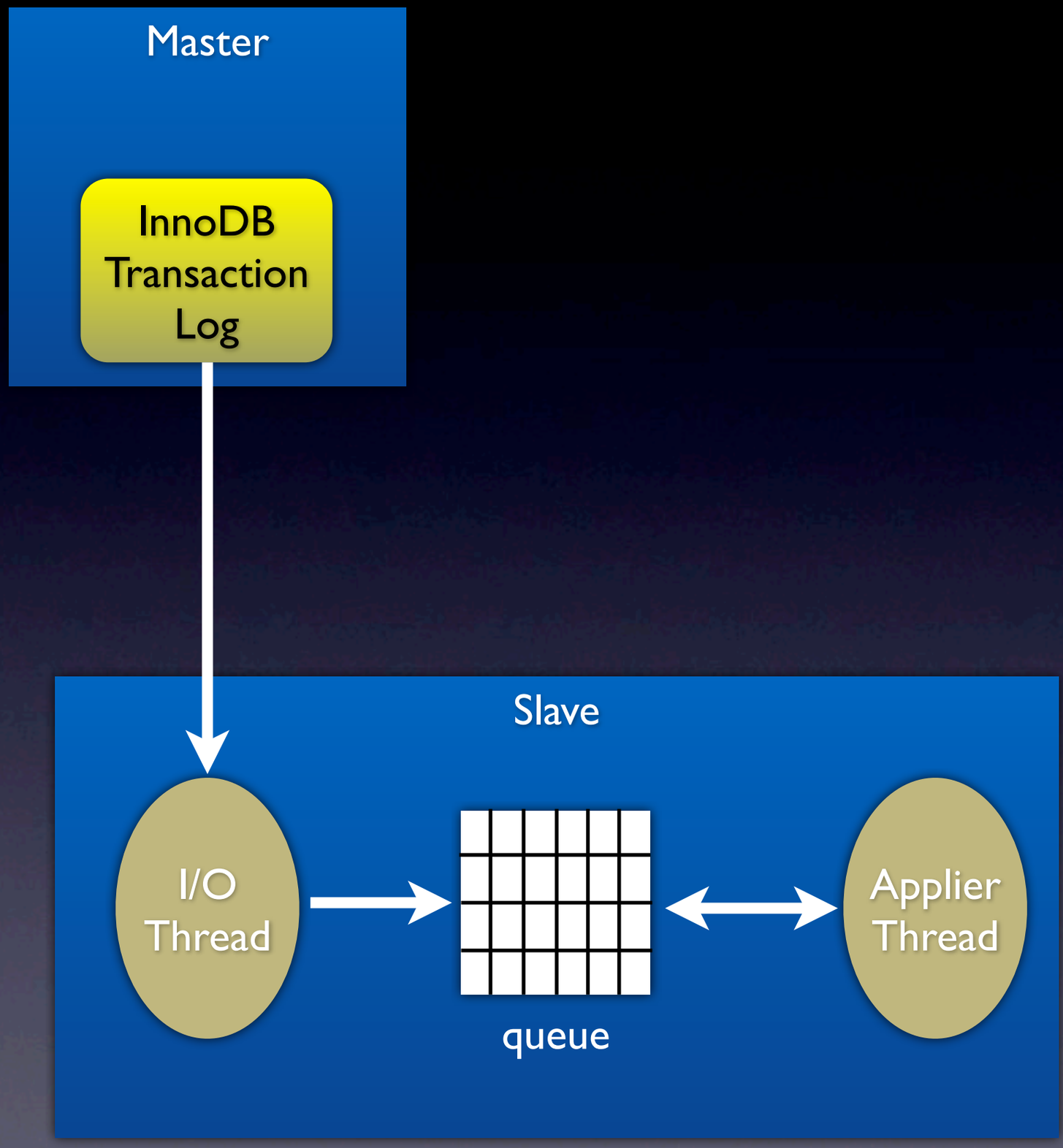
```
$ ./sbin/drizzle --innodb.replication-log &


drizzle> show create table data_dictionary.innodb_replication_log\G
*************************** 1. row ***************************
       Table: INNODB_REPLICATION_LOG
Create Table: CREATE TABLE `INNODB_REPLICATION_LOG` (
  `TRANSACTION_ID` BIGINT NOT NULL,
  `TRANSACTION_SEGMENT_ID` BIGINT NOT NULL,
  `COMMIT_ID` BIGINT NOT NULL,
  `END_TIMESTAMP` BIGINT NOT NULL,
  `TRANSACTION_MESSAGE_STRING` TEXT COLLATE utf8_general_ci NOT NULL,
  `TRANSACTION_LENGTH` BIGINT NOT NULL
) ENGINE=FunctionEngine COLLATE = utf8_general_ci REPLICATE = FALSE


drizzle> show create table data_dictionary.sys_replication_log\G
*************************** 1. row ***************************
       Table: SYS_REPLICATION_LOG
Create Table: CREATE TABLE `SYS_REPLICATION_LOG` (
  `ID` BIGINT,
  `SEGID` INT,
  `COMMIT_ID` BIGINT,
  `END_TIMESTAMP` BIGINT,
  `MESSAGE_LEN` INT,
  `MESSAGE` BLOB,
  PRIMARY KEY (`ID`,`SEGID`) USING BTREE,
  KEY `COMMIT_IDX` (`COMMIT_ID`,`ID`) USING BTREE
) ENGINE=InnoDB COLLATE = binary REPLICATE = FALSE
```

RabbitMQ Replication

- Author: Marcus Eriksson (http://developian.blogspot.com)
- Sends replication messages to an external RabbitMQ message server
- Available as Drizzle plugin, or...
- External Java app that uses the transaction log (http://www.rabbitreplication.org/)

Native Replication

- Direct replication between Drizzle servers
- Uses master's InnoDB transaction log
- Similar in design to MySQL replication (I/O thread and applier thread)
- No relay log files; InnoDB table used for queue
- Status information stored in tables
- Initial implementation: ~2000 LOC


Master
InnoDB
Transaction
Log
Slave
I/O
Thread
queue
Applier
Thread

Using the Slave Plugin

Master

```
drizzled --innodb.replication-log
```

Slave

```
drizzled --plugin-add=slave \
         --slave.config-file=/path/to/slave.cfg
```

Example Elliott slave.cfg

```
master-host = hostA
master-port = 3306
master-user = slave_user
master-pass = secret

# try reconnecting 10 times before giving up
max-reconnects = 10
seconds-between-reconnects = 30

# sleep interval before looking for more master data
io-thread-sleep = 5

# sleep interval between local queue checks
applier-thread-sleep = 5
```

Replication Tools

- *drizzledump* - backup program; outputs current InnoDB transaction log position
- *drizzletrx* - dump transaction log as SQL or Google Protobuf messages (file-based or InnoDB)

drizzledump

```
$ drizzledump --single-transaction junk

-- drizzledump 2011.03.13.2241 libdrizzle 7, for unknown-linux-gnu (x86_64)
--
-- Host: localhost    Database: junk
-- ------------------------------------------------------
-- Server version 2011.03.13.2241 (Drizzle server)

-- SYS_REPLICATION_LOG: COMMIT_ID = 123, ID = 5000

<snip>

-- Dump completed on 2011-Mar-31 12:00:21
```

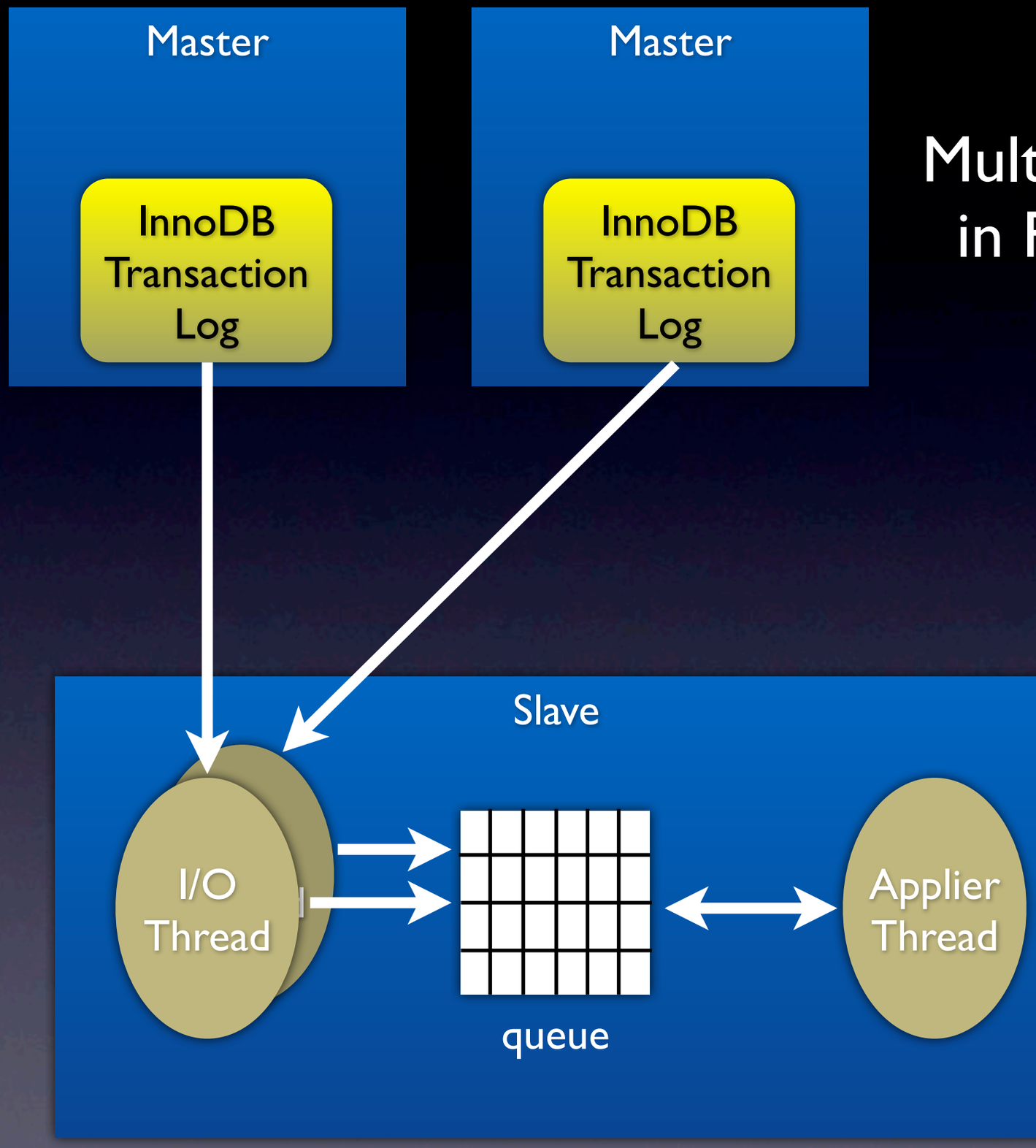
drizzletrx

```
$ drizzletrx --use-innodb-replication-log -u root -p 3306

-- EVENT: Server startup
SET AUTOCOMMIT=0;
CREATE SCHEMA `junk` COLLATE utf8_general_ci;
COMMIT;
```

```
$ drizzletrx --use-innodb-replication-log -u root -p 3306 --raw

transaction_context {
  server_id: 1
  transaction_id: 772
  start_timestamp: 1301586845363917
  end_timestamp: 1301586845363920
}
event {
  type: STARTUP
}
segment_id: 1
end_segment: true
```

Future and Potential Replication Features

- Chained replication support
- Per-catalog replication
- End-to-end Checksumming
- Replicate from MySQL to Drizzle
- Multiple masters for single slave



Multi-master support in Fremont release

Getting More Info

- FreeNode IRC channel: #drizzle
- drizzle-discuss@lists.launchpad.net
- http://docs.drizzle.org

Drizzle Developer Day

- Santa Clara Hilton Hotel - Coastal Ballroom
- Friday, April 15th - 9:30am to 4pm
- Learn how to use Drizzle, contribute code, replication, catalogs, storage engines, testing and benchmarking

Replication Demo