AQUAQ Analytics

AquaQuarantine:

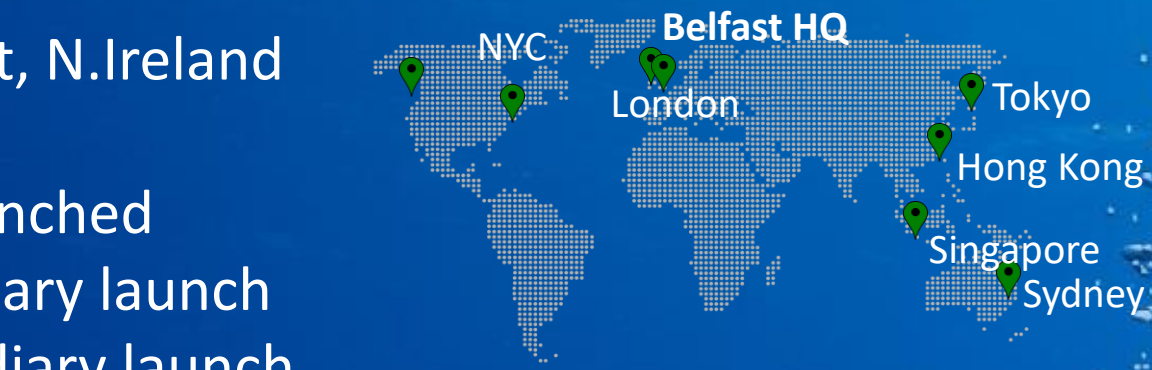
TorQ Tickerplant

Experts in fast data solutions

for demanding environments

AQUAQ Analytics

- Established in 2011
- Headquarters in Belfast, N.Ireland
- Headcount of 160 staff
- 2016 US Subsidiary launched
- 2018 Singapore subsidiary launch
- 2020 Hong Kong subsidiary launch



What do we do?

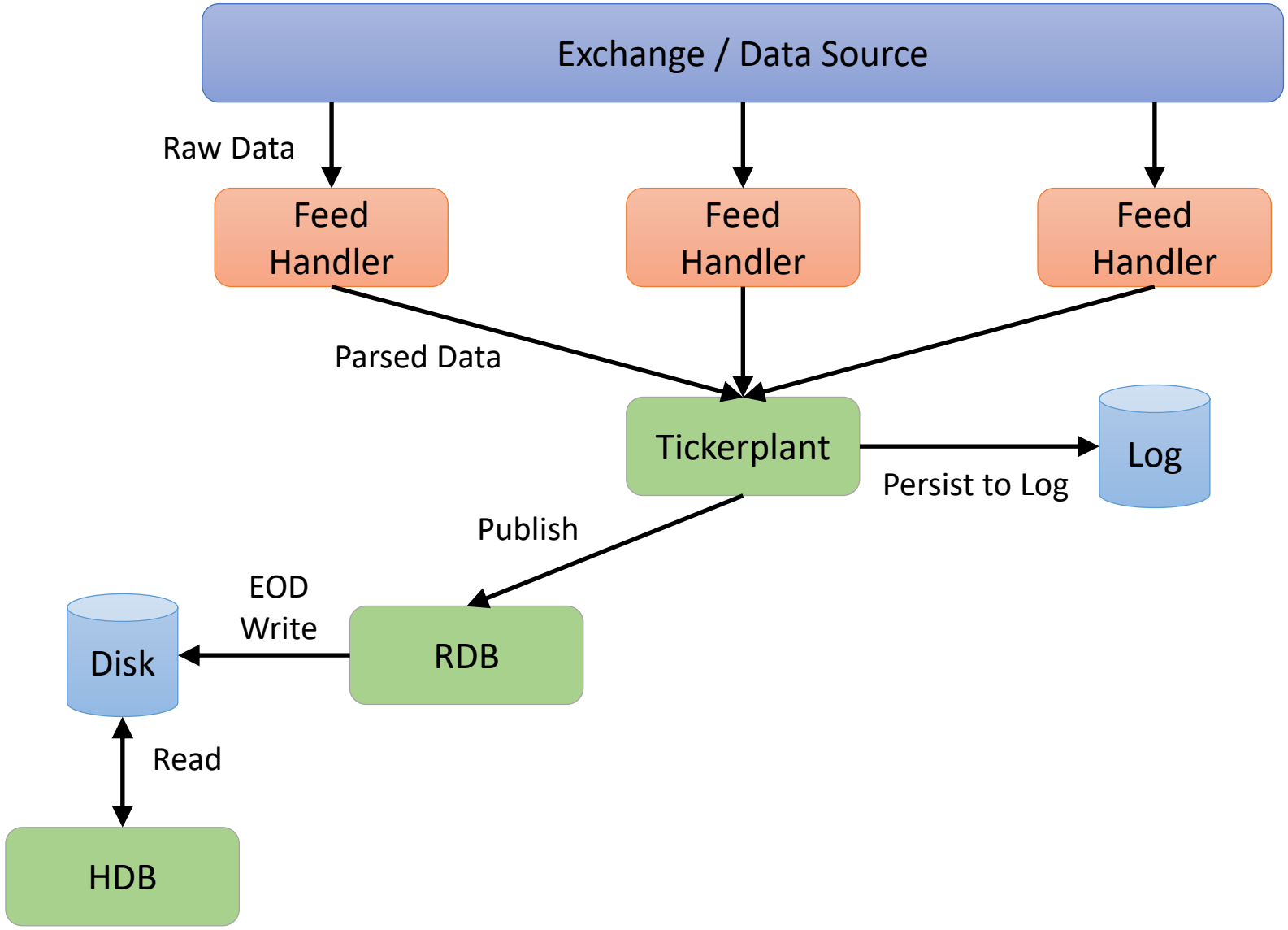
Technology Consultancy Services

Altair Panopticon Professional Services

Remote (24/7) Support Centre of Excellence

AQUAQ Analytics

Why?



AQUAQ Analytics

Why?

- TorQ Tickerplant is the kdb+tick TP with some minor modifications
- In various installations we've tweaked some things, or thought "wouldn't it be nice if…"
- Main drivers are:
  - Greater flexibility
  - Easier customisation
  - Easier to support
  - Building for future enhancements

AQUAQ Analytics

Backward Compatibility

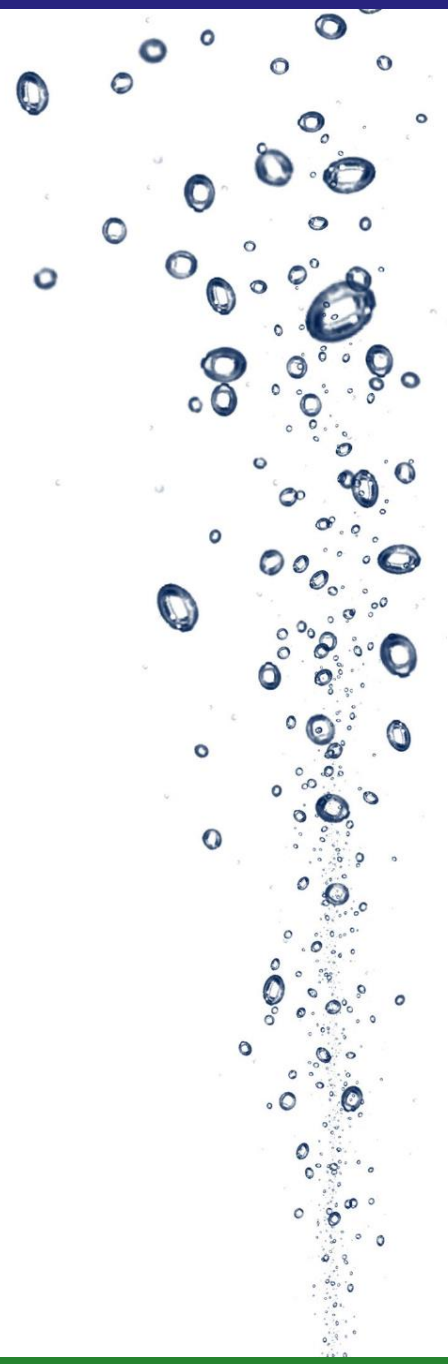
- Need to ensure backward compatibility is maintained
- Maintain interface:
  - .u.upd/.u.sub/.u.add/log file format
- TorQ TP allows configuration of:
  - Immediate or Batch publish
  - Data Timezone
  - Roll Timezone + Roll Time Offset e.g. FX: timestamp in UTC, roll at 5pm NYC local

AQUAQ Analytics

1. More granular log files

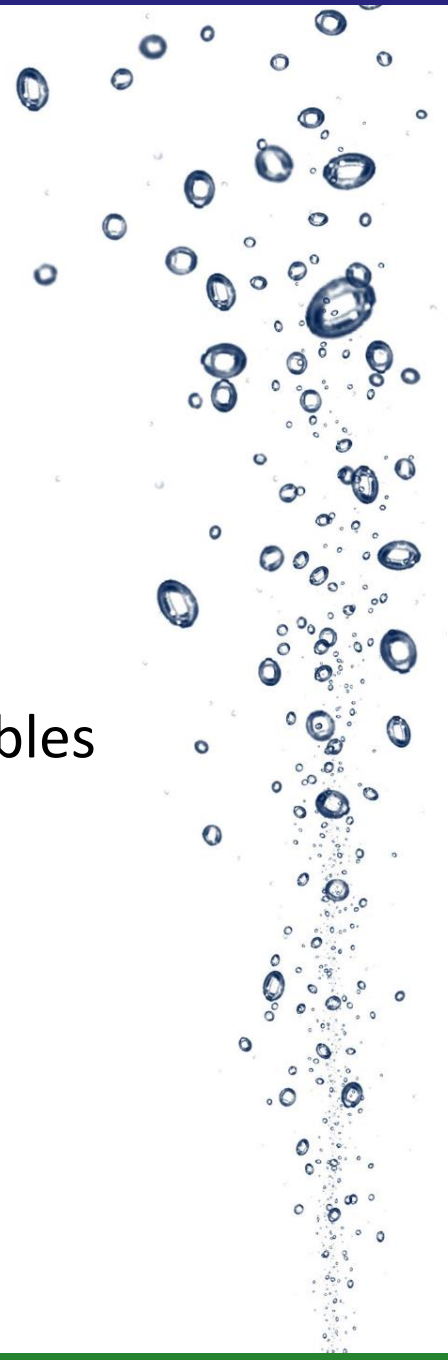
AQUAQ Analytics

- Current TP has one log file per date
- Pros:
  - Simple
  - Message sequence is maintained
- Cons:
  - Can become unwieldy
  - Whole file must be replayed for subset of data
  - On restart log must be counted



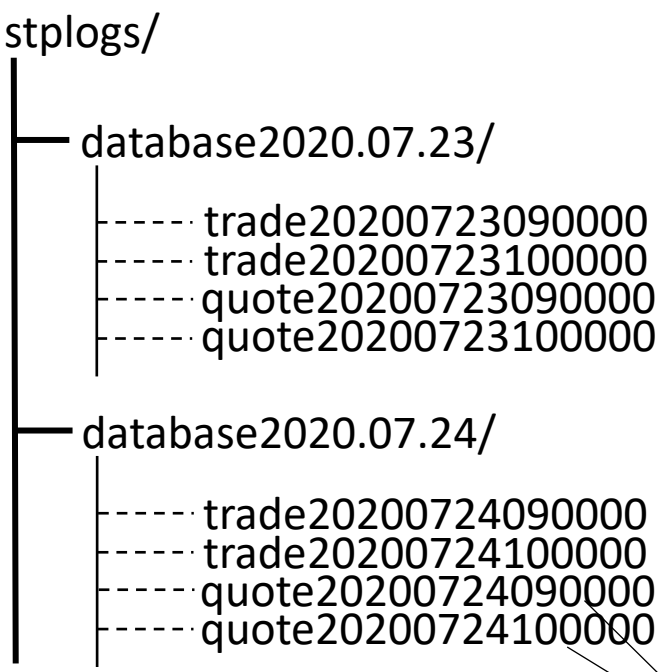
AQUAQ Analytics

- Proposed solution is to provide multiple logging formats
  - `periodic– single log for all tables, rolled on a configurable period
  - `tabular- single log for per able, rolls daily
  - `tabperiod- single log per table per period
  - `none- single log, rolls daily
  - `custom- different logging method per table, or no logging for some tables
  - Easy to extend e.g. per instrument log files (might be useful in FX)
- In all cases, we have a directory of log files

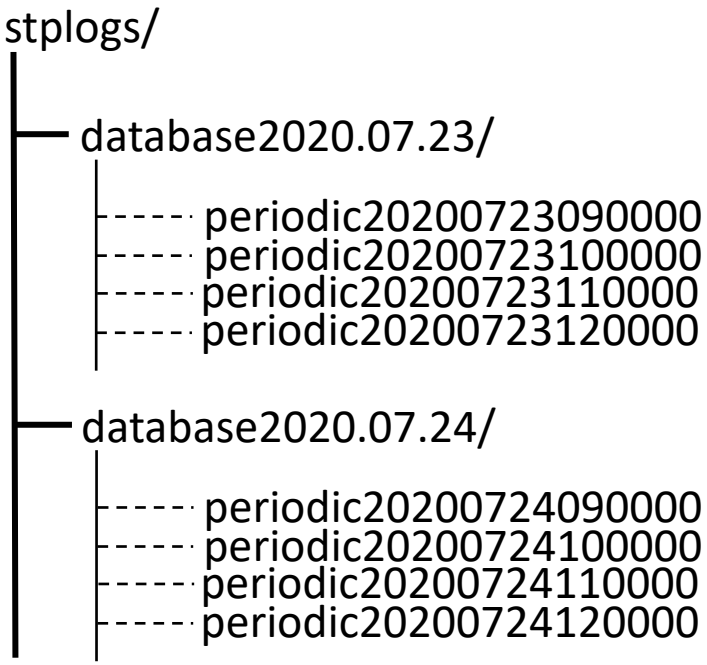


AQUAQ Analytics

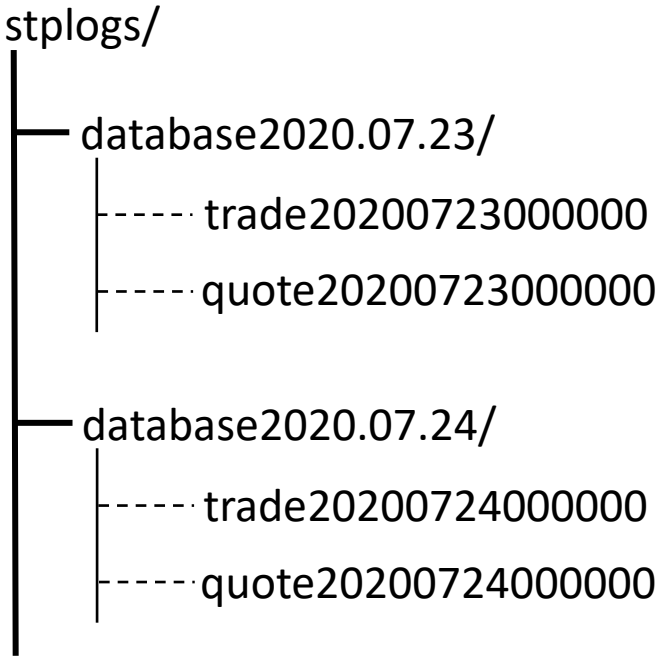
Default (tabperiod)



Periodic



Tabular



Lognames timestamped with start of logging period

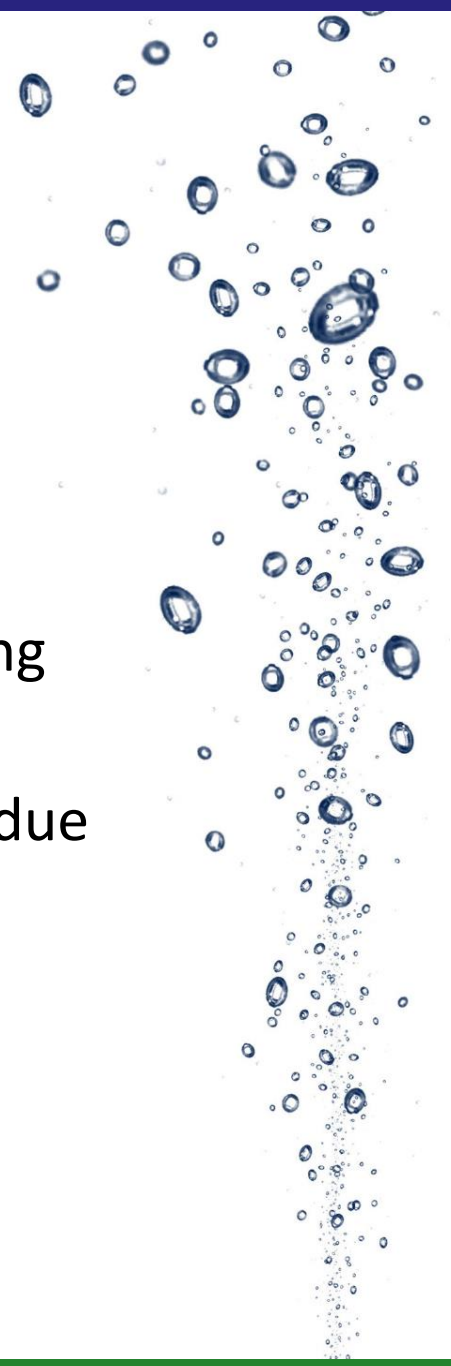
AQUAQ Analytics

- We also maintain a meta file at the top level of directory containing meta info on each file. Log file directory is self describing
- Meta info is:
  - Time period the log file covers
  - Tables contained in log file
  - Message Count (on best efforts basis)
  - Schema of tables contained

```
q)last .stpm.metatable
seq        | 18i
logname    | `:/home/tptest/stplogs/database2020.08.10/trade_iex20200810140000
start      | 2020.08.10D14:00:00.690988000
end        | 0Np
tbls       | ,`trade_iex
msgcount   | 0i
schema     | (,`trade_iex)!,+`time`sym`price`size`stop`cond`ex`srctime!(`timestamp$();`g#`symbol$();`float$();`int$();`boolean$();"";"";`timestamp$())
additional| ,()!()
```

AQUAQ Analytics

- Clients must replay all the log files they need data from
- Schema info is good for:
  - Historic compatibility
  - Intraday schema changes
- If inter-table message sequencing is required, use `periodic or `none logging modes
- When the TP restarts it creates a new log file with 0 count (instant restart due to no replay)
- Subscribers have to handle both `.u.end and `.u.endp (end-of-period)



AQUAQ Analytics

2. New Batch Publish Mode

AQUAQ Analytics

- Support same options as current TP:
  - Immediate: log and publish immediately
  - Batch: log, publish on a timer
- Added `memorybatch option
  - Log to memory
  - On timer, flush to log and publish
- Pros:
  - Better performance (less disk writes)
- Cons:
  - Additional failure case to manage (once it’s received by TP it hasn’t been logged)

AQUAQ Analytics

- Easy to add own logging modes
- Need to add upd function and timer function

```
[q).stplg.upd
              | ()
memorybatch   | {[t;x;now]
  t insert updtab[t] . (x;now);
 }
defaultbatch| {[t;x;now]
  t insert x:.stplg.updtab[t] . (x;now);
  `..loghandles[t] enlist(`upd;t;x);
  // track tmp counts, and add these after publish
  @[`.stplg.tmpmsgcount;t;+;1];
  @[`.stplg.tmprowcount;t;+;count first x];
 }
immediate    | {[t;x;now]
  x:updtab[t] . (x;now);
  `..loghandles[t] enlist(`upd;t;x);
  @[`.stplg.msgcount;t;+;1];
  @[`.stplg.rowcount;t;+;count first x];
  .stpps.pub[t;x]
 }
```

```
[q).stplg.zts
              | ()
memorybatch   | {
  {[t]
    if[count value t;
      `..loghandles[t] enlist (`upd;t;value flip value t);
      @[`.stplg.msgcount;t;+;1];
      @[`.stplg.rowcount;t;+;count value t];
      .stpps.pubclear[t]];
  }each .stpps.t;
 }
defaultbatch| {
  // publish and clear all tables, increment counts
  .stpps.pubclear[.stpps.t];
  // after data has been published, updated the counts
  .stplg.msgcount+:.stplg.tmpmsgcount;
  .stplg.rowcount+:.stplg.tmprowcount;
  // reset temp counts
  .stplg.tmpmsgcount:.stplg.tmprowcount:()!();
 }
immediate    | {}
```

AQUAQ Analytics

3. Flexible Update Handling

AQUAQ Analytics

- Supply a dictionary of update functions per table
- Default definition is to add current timestamp (this is useful in most use cases)

```
q).stplg.updtab
logmsg    | {(enlist(count first x)#y),x}
quote     | {(enlist(count first x)#y),x}
quote_iex | {(enlist(count first x)#y),x}
trade     | {(enlist(count first x)#y),x}
trade_iex | {(enlist(count first x)#y),x}
          | {(enlist(count first x)#y),x}
```

- Maintain sequence number which can be appended to incoming messages
- Allow publishers to send multiple updates in one message (which will get the same timestamp and sequence number)

```
q).stp.upd
{[t;x]
    // snap the current time and check for period end
    .stplg.checkends now:.z.p+.eodtime.dailyadj;
    // Type check allows update messages to contain multiple tables/data
    $[0h<type t;
      .stplg.updmsg'[t;x;now];
      .stplg.updmsg[t;x;now]
    ];
    .stplg.seqnum+:1;
 }
```

AQUAQ Analytics

- Add error trap mode to capture failed messages to a separate error log
- Useful for debugging

```
q).stplg.updtab
logmsg    | {(enlist(count first x)#y),x}
quote     | {(enlist(count first x)#y),x}
quote_iex| {(enlist(count first x)#y),x}
trade     | {(enlist(count first x)#y),x}
trade_iex| {(enlist(count first x)#y),x}
          | {(enlist(count first x)#y),x}
```

- Maintain both message count and row counts per table

AQUAQ Analytics

4. Flexible Publishing

AQUAQ Analytics

- Old TP pub/sub code was based on u.q from kdb+ tick, extended to incorporate broadcast publish (-25!)
- Clients could subscribe based on table and optionally sym
- New subscription functionality:
  - Supports same interface (.u.sub[tables;syms])
  - Utilises broadcast publish for subscriptions for all data from a table
  - Removes requirement for a table to have column called `sym
  - Allows any “where” filter to be applied
  - Allows columns to be specified
  - Result is less subscriber side filtering => lower overhead
- Complex where filtering should be reserved for non-primary capture tickerplants (e.g. chained tickerplants)

AQUAQ Analytics

5. Compatibility

AQUAQ Analytics

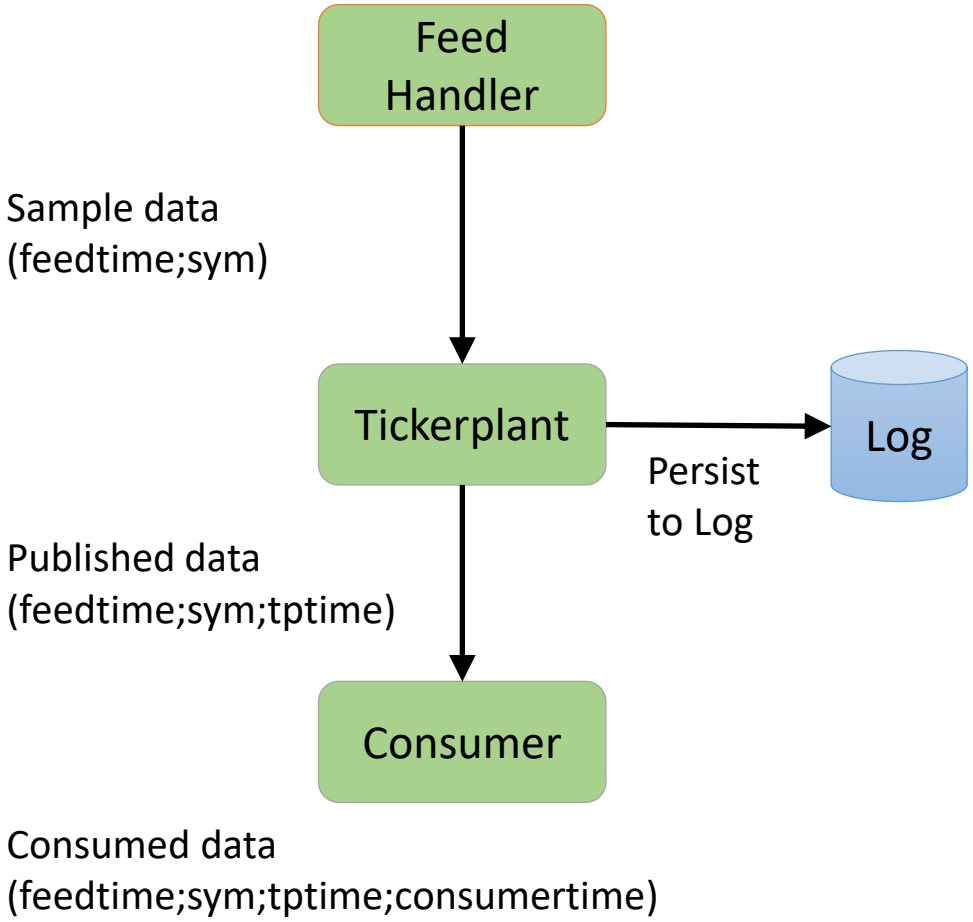
- All TorQ components will work with both old and new tickerplants
  - RDB
  - WDB
  - Chained TP
  - Subscribers
  - Tickerlog Replay
- Will automatically replay multiple log files
- Will utilise new subscription mechanism
- We would like to phase out use of the old tickerplant, but will leave in place

AQUAQ Analytics

6. Performance

AQUAQ Analytics

Test Harness



- Tests executed for:
  - Single inserts (1 record)
  - Bulk inserts (1000 records)
- Measurements taken for:
  - Messages per second
  - Latencies between each point
  - Drift
- Lots of variables in real life!
  - Disk performance
  - Message size
  - Number of consumers
  - Etc…

AQUAQ Analytics

- Results below are for single inserts
- Bulk insert results were broadly similar (dependent on I/O)
- Error mode adds a small over head to every insert

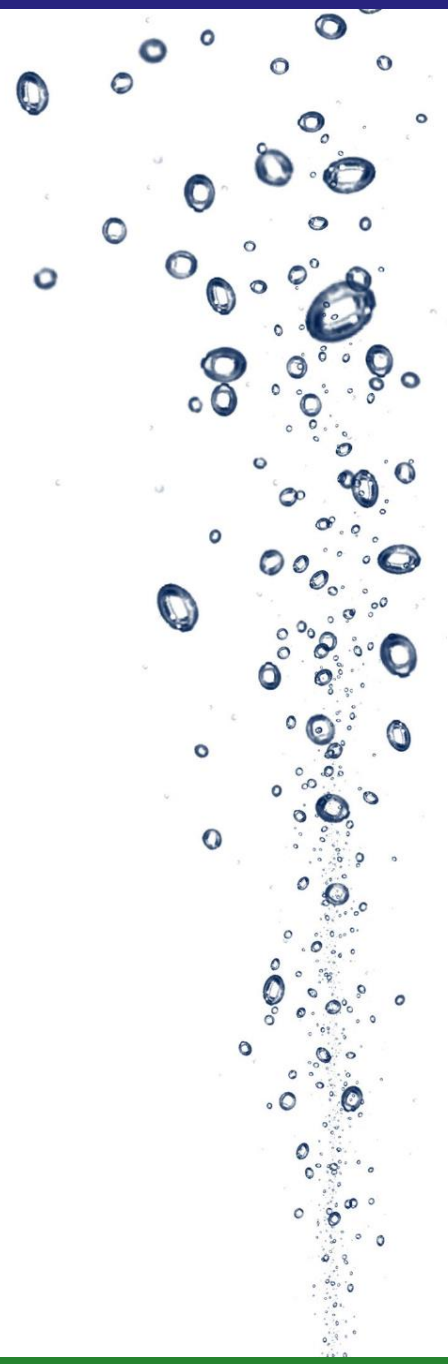
Mode	Old TP (messages p/s)	Segmented TP
Immediate	60,000	66,000
Immediate+Error	N/A	65,000
Batch	127,000	100,000
Memory Batch	N/A	164,000

AQUAQ Analytics

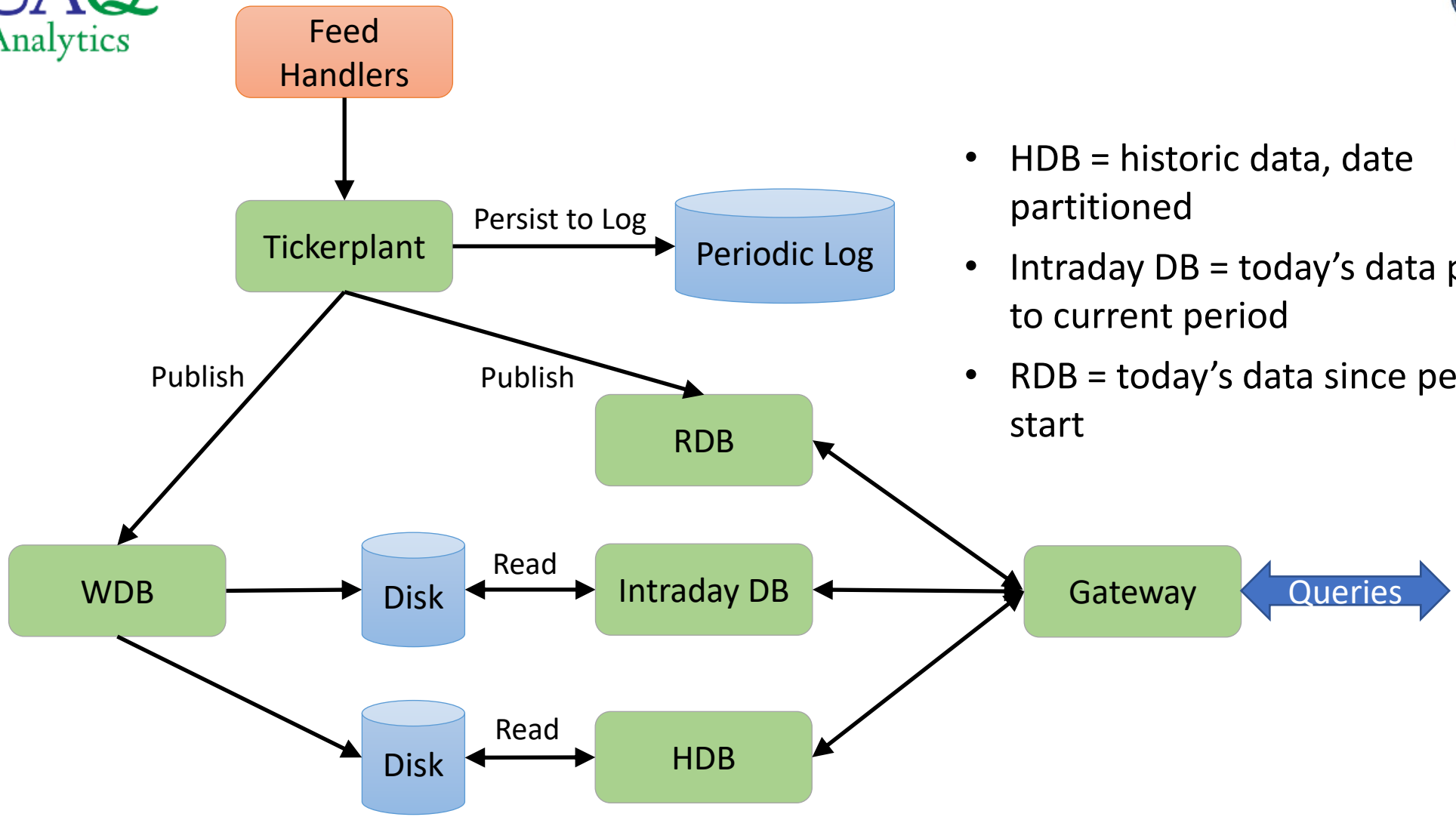
7. Future Planning

AQUAQ Analytics

- Main driver was to allow more granular log files
  - Easier to manage
  - Targeted replays (table prioritisation)
- More frequent log files leads to additional design considerations
  - Old tickerplant had periodic log files where period = one day
  - One day aligns with date partitioning scheme of on disk database
  - More frequent log files lends itself to more granular writes
  - Also snapshots services become easier



AQUAQ Analytics



- HDB = historic data, date partitioned
- Intraday DB = today's data prior to current period
- RDB = today's data since period start

AQUAQ Analytics

- Intraday DB format:
  - Must be easy to append new data intraday
  - Should be well structured for efficiency of majority of queries
- Int partitioned to represent period YYYYMMDDHHMM e.g. 202008071400
- Int partitions mapped via a table to timeperiod+other set of identifiers

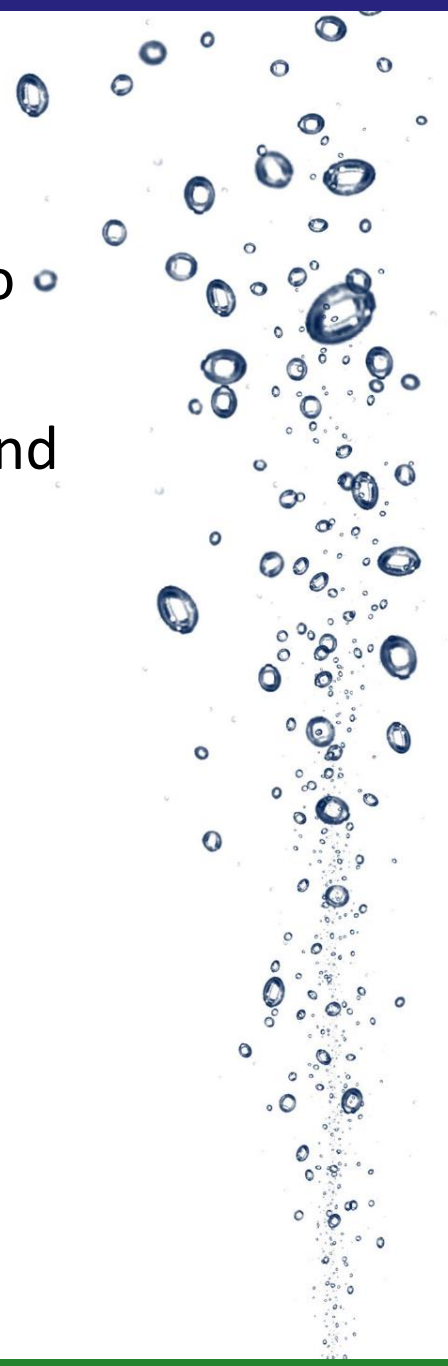
```
q)intmap
periodstart                   sym| int
------------------------------| ---
2020.08.07D00:00:00.000000000 a  | 0
2020.08.07D00:00:00.000000000 b  | 1
2020.08.07D00:00:00.000000000 c  | 2
2020.08.07D01:00:00.000000000 a  | 3
2020.08.07D01:00:00.000000000 b  | 4
2020.08.07D01:00:00.000000000 c  | 5
2020.08.07D02:00:00.000000000 a  | 6
2020.08.07D02:00:00.000000000 b  | 7
2020.08.07D02:00:00.000000000 c  | 8
```

- API functions for data access across RDB/Intraday/HDB would be best

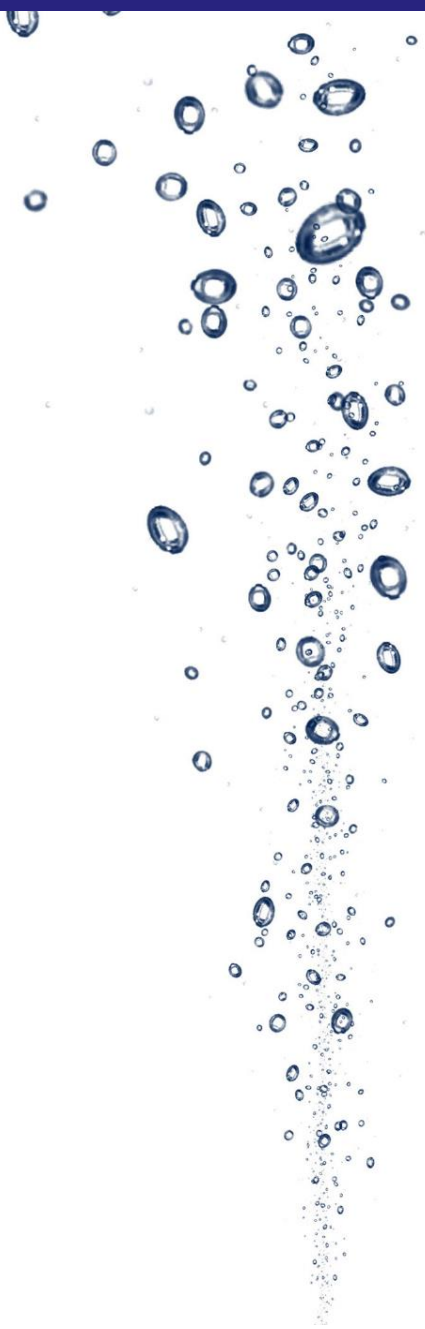
AQUAQ Analytics

- If retention of HDB date partitioning format required then probably best to build that separately
- Could remove date partitioned HDB in some cases and just move to RDB and period partitioned HDB (continuous rolling HDB)

- Pros:
  - Lower overall memory usage
  - Faster recovery of RDB or other subscribers following failure
- Cons:
  - Greater complexity



AQUAQ Analytics



Thanks!

Q+A