DE LA RECHERCHE À L'INDUSTRIE

DISTRIBUTED LUSTRE ACTIVITY TRACKING

Henri DOREAU | CEA/DAM
henri.doreau@cea.fr

CEA, DAM, DIF, F-91297 Arpajon France

www.cea.fr

LustreEco 2015

BACKGROUND

MDT changelog as a notification mechanism

- The metadata servers can provide us with a stream of changelog records
- Used as an asynchronous notification facility
- Interested parties must subscribe (register/deregister) and poll for records

Unbalanced situations may occur…

- One MDT/Numerous subscribers
- One reader/Numerous MDT

  typically: robinhood facing DNE

As well as clearly suboptimal ones

- Ephemeral readers constantly registering/deregistering
- Ephemeral readers going away for a long time before re-appearing
- Readers filtering out most records

  …but getting the whole stream anyway

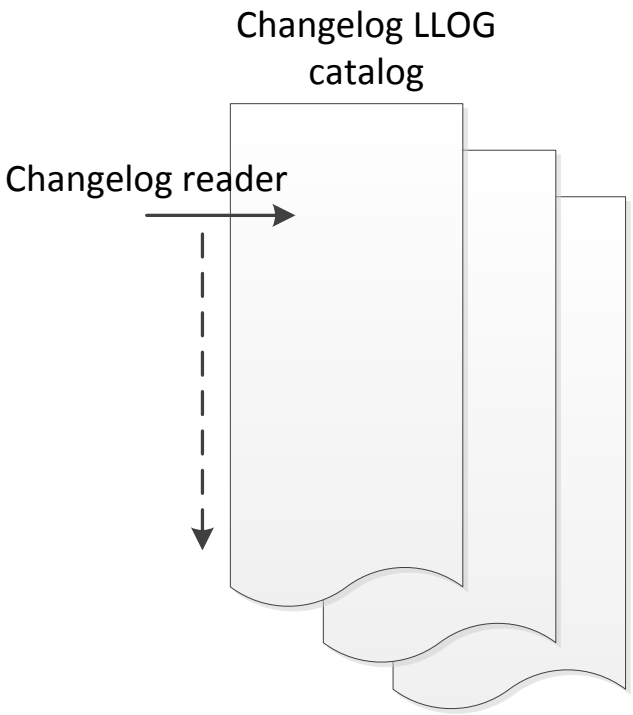
LUSTRE CHANGELOG MECHANISM

MDT changelog as a notification mechanism

- Any client can read it
- Specify start record
- Loop over available records
- Acknowledge for a given reader, up to a certain point (batch ack)

Specifications/Performance

- Relies on lustre KUC
- Up to 100k record/s/MDT (if you play fair)



USING LUSTRE CHANGELOG

Server-side steps

- Registration

```
# lctl --device lustre-MDT0000 changelog_register
lustre-MDT0000: Registered changelog userid 'cl1'
```

- Deregistration

```
# lctl --device lustre-MDT0000 changelog_deregister
lustre-MDT0000: Deregistered changelog user 'cl1'
```

Client-side steps

- Repeat
  - Start (MDT, start record #)
  - Recv / consume / free
  - Acknowledge consumed records
  - Stop
- Until EOF

TYPICAL USE

Robinhood policy engine

- Major (single?) real-life changelog user
- Follows incremental filesystem changes
  - Requires initial scan to populate the DB
  - No need to re-scan the whole namespace periodically

Lustre-rsync

- Replicates all changes into a second namespace

Custom monitoring

- Use `lfs changelog` to display records as text
- Insert lines into a Logstash/Elasticsearch/Kibana system

RECORD FORMAT EVOLUTIONS

Initial - Lustre 2.0

- Single record format (`struct changelog_rec`)
- `RENAME` operations required two records
  - Had to follow each other
  - Though they did not always
  - Difficult to process properly

Extended – Lustre 2.3 (backported to 2.1)

- Introduced `struct changelog_ext_rec`
- Added new flag, extra fields for `RENAME`
- All records remapped as extended ones before getting delivered

Flexible – Lustre 2.7

- Common header (`struct changelog_rec`)
- Optional extensions, with corresponding flags and accessors (introduced jobid field)
- Client expresses capabilities, server-side remapping if needed
- Compatibility between old and new applications/clients/servers

GOALS

Based on the existing changelog API

- Broadcast the stream (publish/subscribe) to numerous unregistered clients
- Distribute stream processing
- Re-order the records to optimize final processing
  - Can drop records that cancel out each other (create/unlink patterns)
  - Can group records by target FID or parent FID
  - Offload this work from reader applications (e.g.: *Robinhood Policy Engine*)

More generally

- Stream pre-processing
- Versatile distribution scheme
- Relaxed constraints on the MDS

LCAP PROXY

LCAP LUSTRE CHANGELOG PROXY

Stands for *Lustre Changelog Aggregate & Publish*

- Client/Server architecture
  - liblcapclient
  - lcapd
  - processing modules
- Essentially a Lustre changelog proxy
  - Seen as a single changelog reader by Lustre
  - Lives in userland
  - Re-ordering and distribution schemes implemented as loadable modules
- Official CEA project
  - Freely distributed (**https://github.com/cea-hpc/lcap.git**)

LIBLCAPCLIENT

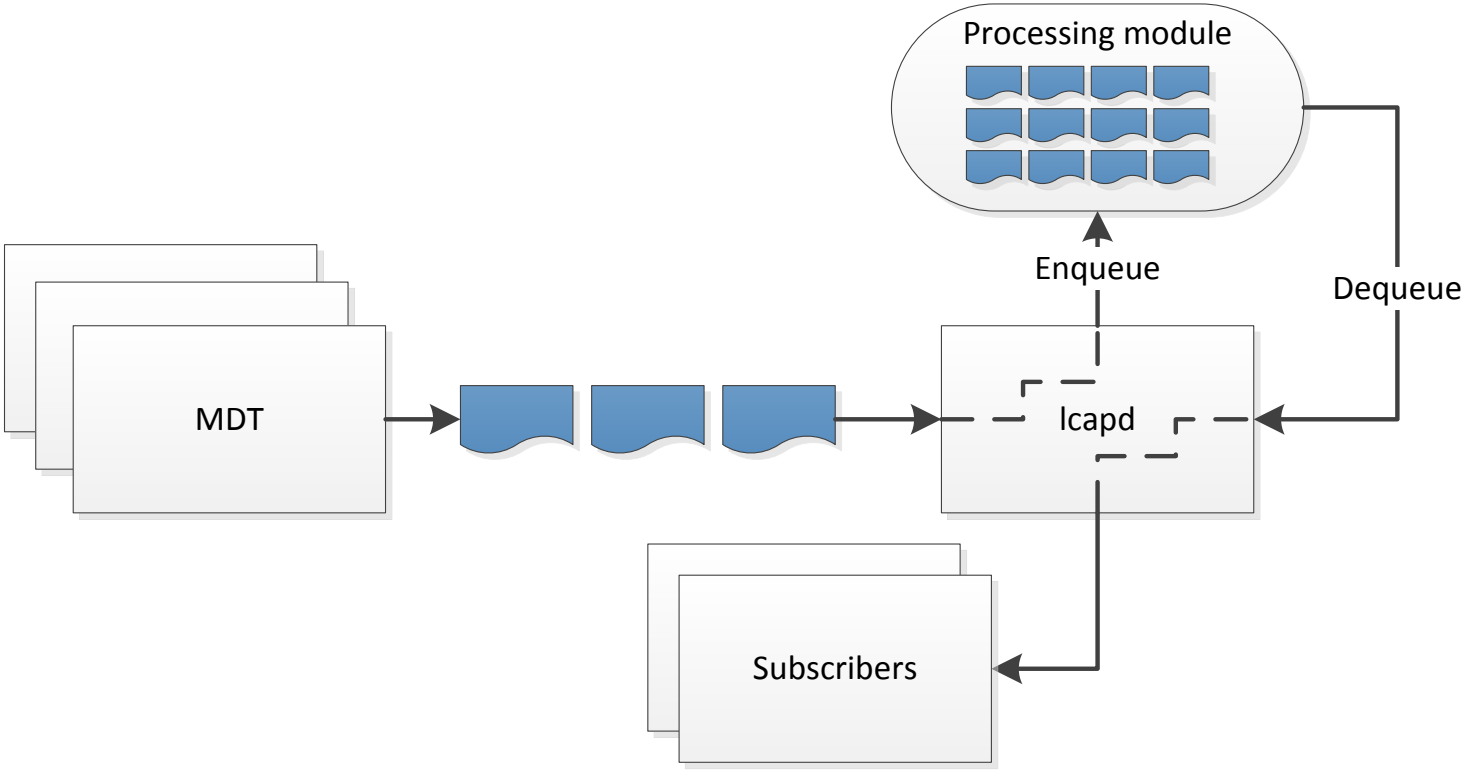
As close as possible from liblustreapi

- Proxified channel (default)
  - `lcap_changelog_start()`
  - `lcap_changelog_receive()`
  - `lcap_changelog_clear()`
  - `lcap_changelog_fini()`
  - `lcap_changelog_setopt()`
- NULL-channel
  - lcap_CL_DIRECT flag to `lcap_changelog_start()`
  - Other flags mapped to their lustreapi equivalent
  - Functions then directly call their lustreapi siblings
- Client only needs the server URI (taken from env)

LCAPD

Implements all the logic

- All communications based on the (excellent) Zeromq message passing library
- Purpose-specific policies



TRANSACTION CHAIN

Transactionnal aspect remains preserved (or not, you choose)

- Reader applications acknowledge records up to a given index
  - « cheap » local ACK
  - Periodically sent back to the server
- Server gets informed
- lcapd acknowledges records in lustre according to the loaded policy
- Examples:
  - Can use min(acknowledgements)
  - Can decide to acknowledge unread records if there are no readers (broadcast)

UNDER THE HOOD: MESSAGE PASSING

ØMQ

- Lightweight message passing library
- Adaptive patterns (REQ/REP, PUB/SUB, PUSH/PULL…)
- Asynchronous I/O
- Familiar API (close to BSD sockets)
- Excellent documentation
- Used for internodes and interthread communications
  - The *lockless monster* isn't a monster anymore!
- Free and actively developed software (see http://zeromq.org)

UNDER THE HOOD: POLICIES

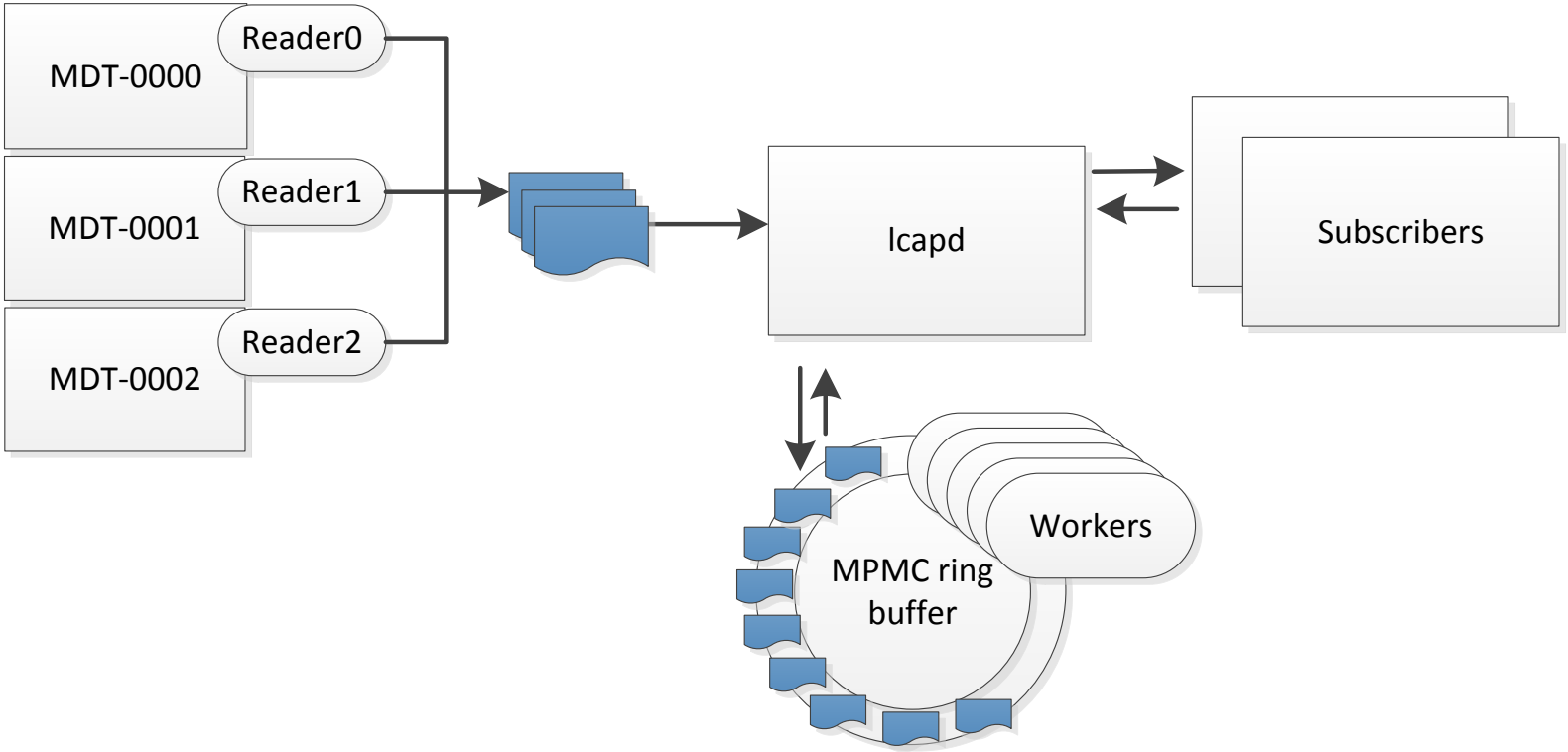
Aggregation and distribution modules

- Policies implemented as modules
  - executed server-side
- Distributed as shared libraries
- Expose a pre-defined API
  - Enqueue records (allow re-ordering)
  - Dequeue records (allow distribution strategies)
  - Indicate up to which record # to clear server-side

UNDER THE HOOD: POLICIES (2)

N collaborative threads

- One changelogs reader thread per MDT
- Requests push/pulled to policy worker threads
- Can share nothing or operate a common data structure



UNDER THE HOOD: BATCHING

Aggregation

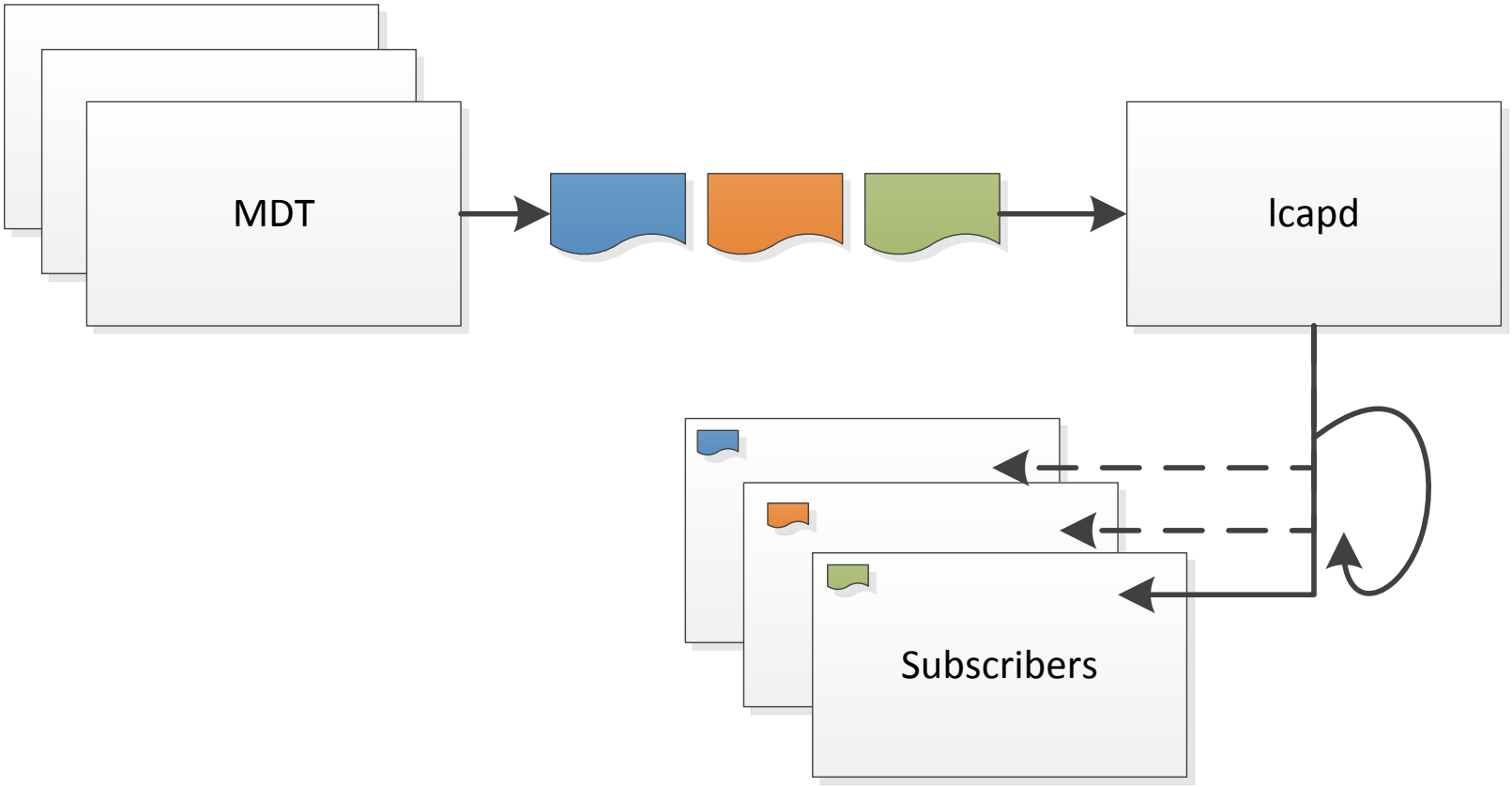
- Policies can internally re-order records as they want
- Records are batch sent to the client
- Policies can decide how to deliver stream to a given client
  - Can group by target FID
  - Can group by source MDS
  - Can rely on simple time windowing

DISTRIBUTION STRATEGIES

LOAD BALANCING

Distribute stream processing between two instances of robinhood

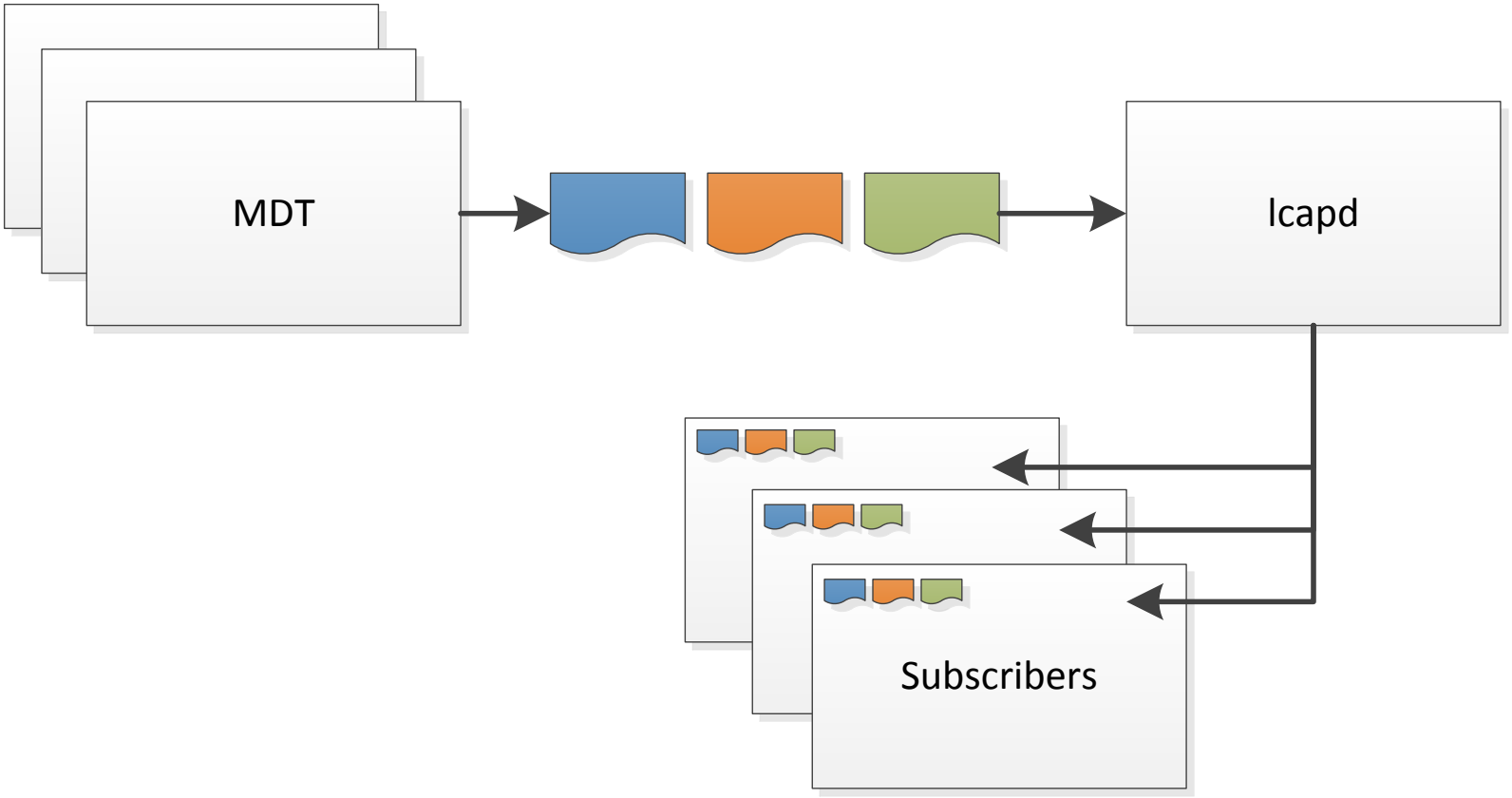
- Round-robin between end readers



BROADCAST

Replicate stream to many ephemeral readers

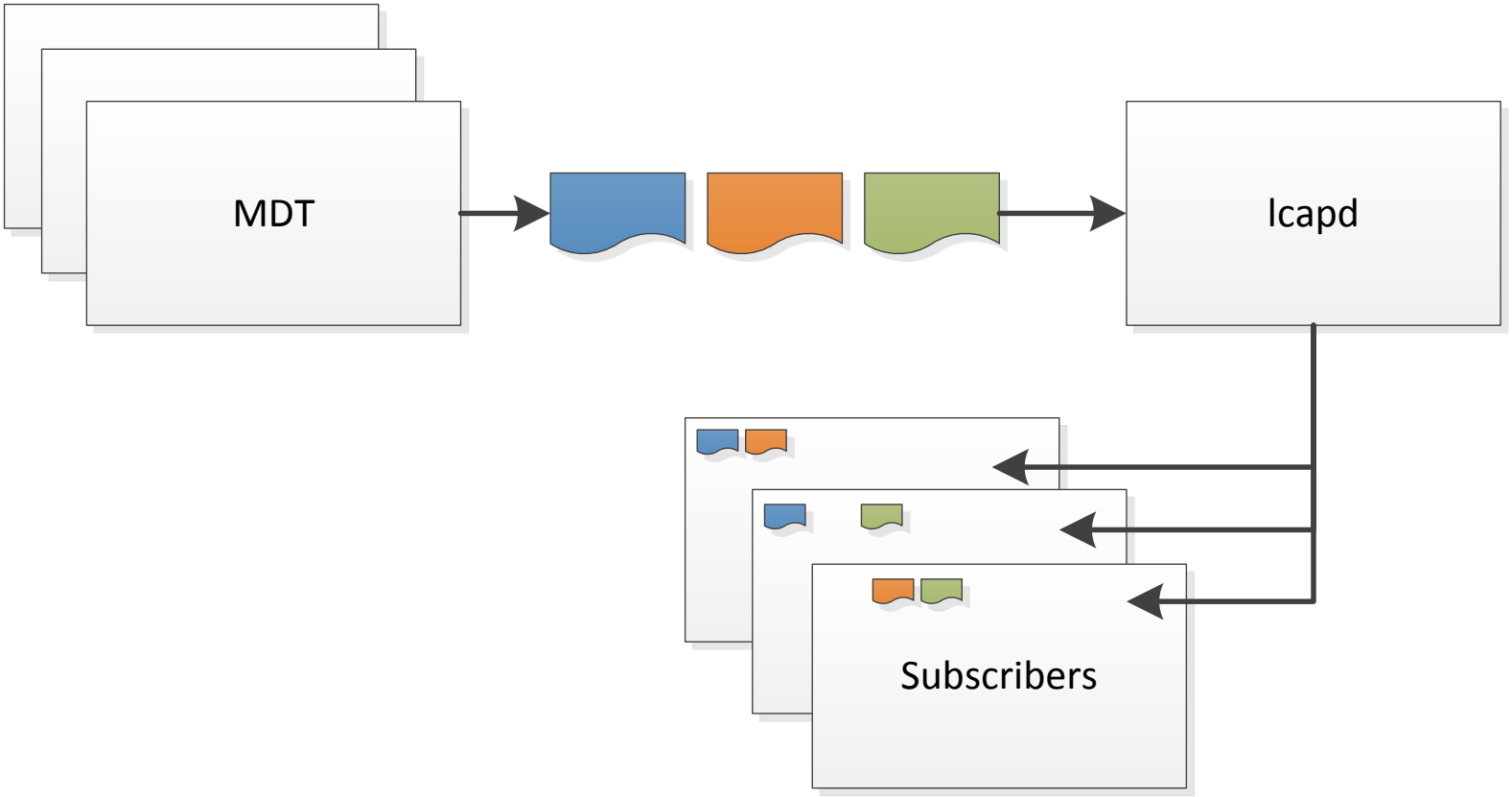
- Publish/Subscribe mechanism



FILTERED BROADCAST

Replicate partial stream (filter out records)

- Publish/Subscribe mechanism, records not matching client filters are not delivered



CONCLUSION

CONCLUSION

Interesting prospectives

- Already proven easy to extend/experiment with
- Ongoing
  - Recovery procedures (clients must survive a server restart)
  - Implement consumer groups
  - Implement adaptive batching

Stabilize and mature the project

- Not used in production yet
- Explore corner cases
- Profile and optimize using at scale deployments

WANT TO TRY IT?

Disclaimer: lcap is still under heavy work ☺

- Implemented in C (kernel style, minus tabs)
- Limited dependencies (lustreapi/pthread/zmq)
- LGPLv3

https://github.com/cea-hpc/lcap.git

THANK YOU!

ANY QUESTION?

Commissariat à l'énergie atomique et aux énergies alternatives
CEA/DAM Ile de France | 91297 Arpajon Cedex
T. +33 (0)1 69 26 40 00

DAM Ile de France

Etablissement public à caractère industriel et commercial | RCS Paris B 775 685 019