DE LA RECHERCHE À L'INDUSTRIE

SCALABLE CHANGELOGS DISTRIBUTION WITH CLAP

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

www.cea.fr

BACKGROUND

MDS changelogs as a notification mechanism

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

Unbalanced situations may occur…

- One MDS/Numerous subscribers
- One reader/Numerous MDS

  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

GOALS

Based on the existing changelogs 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

CLAP PROXY

CLAP LUSTRE CHANGELOGS PROXY

Stands for *changelogs Aggregate & Publish*

- Client/Server architecture
  - libclapclient
  - clapd
  - processing modules
- Essentially a Lustre changelogs proxy
  - Seen as a single changelogs 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/clap.git**)

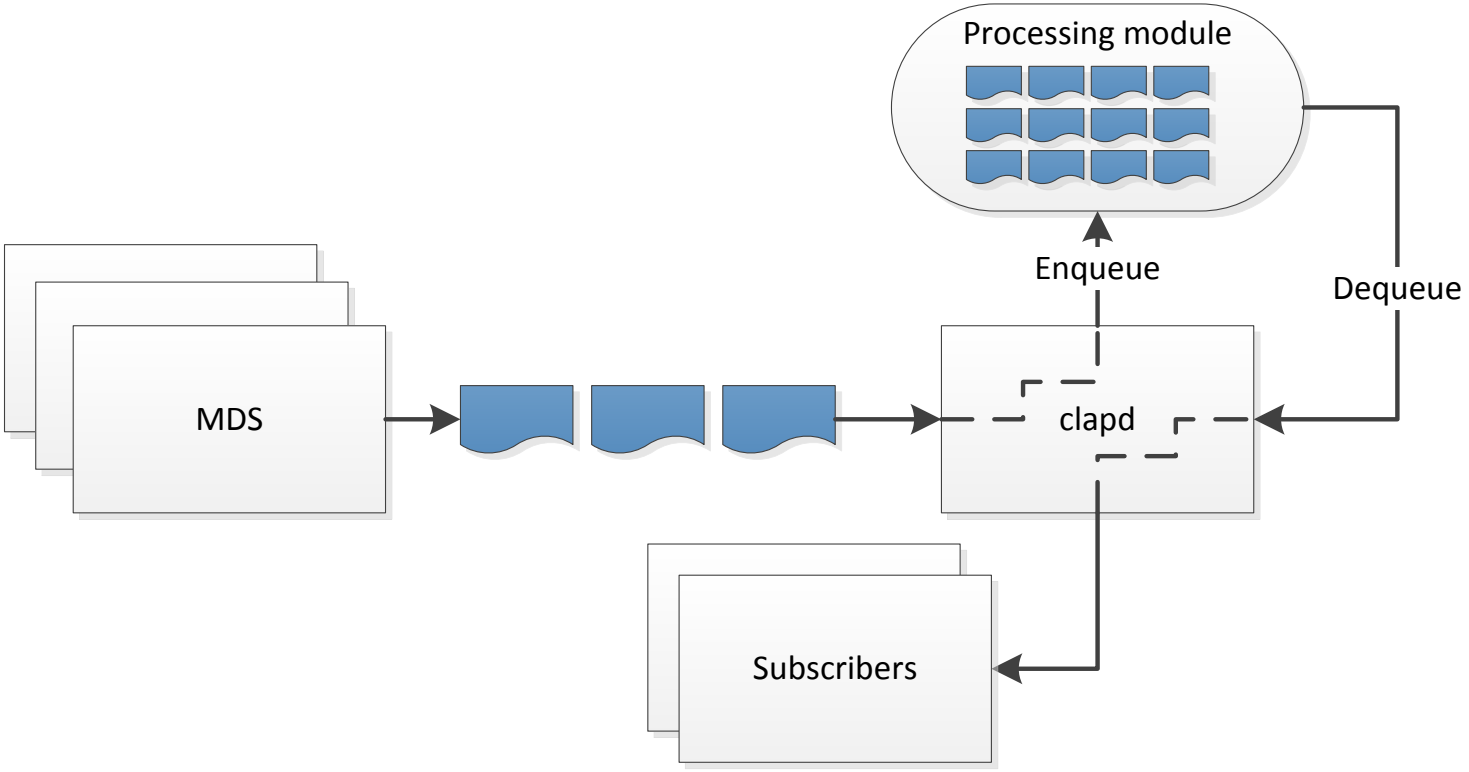
LIBCLAPCLIENT

As close as possible from liblustreapi

- Proxified channel (default)
  - `clap_changelog_start()`
  - `clap_changelog_receive()`
  - `clap_changelog_clear()`
  - `clap_changelog_fini()`
  - `clap_changelog_setopt()`
- NULL-channel
  - CLAP_CL_DIRECT flag to `clap_changelog_start()`
  - Other flags mapped to their lustreapi equivalents
  - Functions then directly call their lustreapi siblings
- Client only needs the server URI (taken from env)

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
- Policy gets informed
- Policy instructs clapd what/when to acknowledge to the MDS
- 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

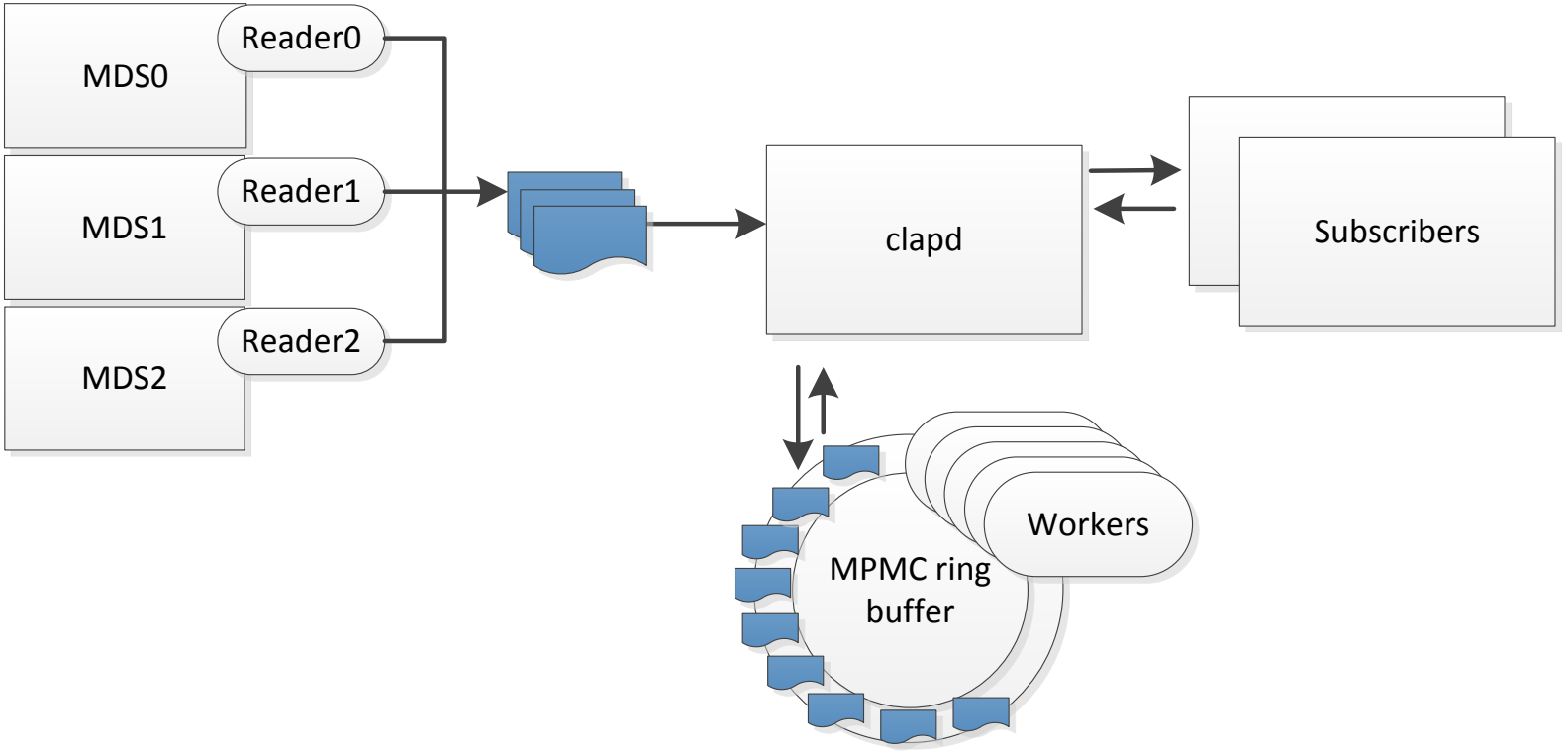
Aggregation and distribution modules

- 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 MDS
- Requests push/pulled to policy worker threads
- Can share nothing or operate a common data structure



UNDER THE HOOD: BATCHING

Aggregation

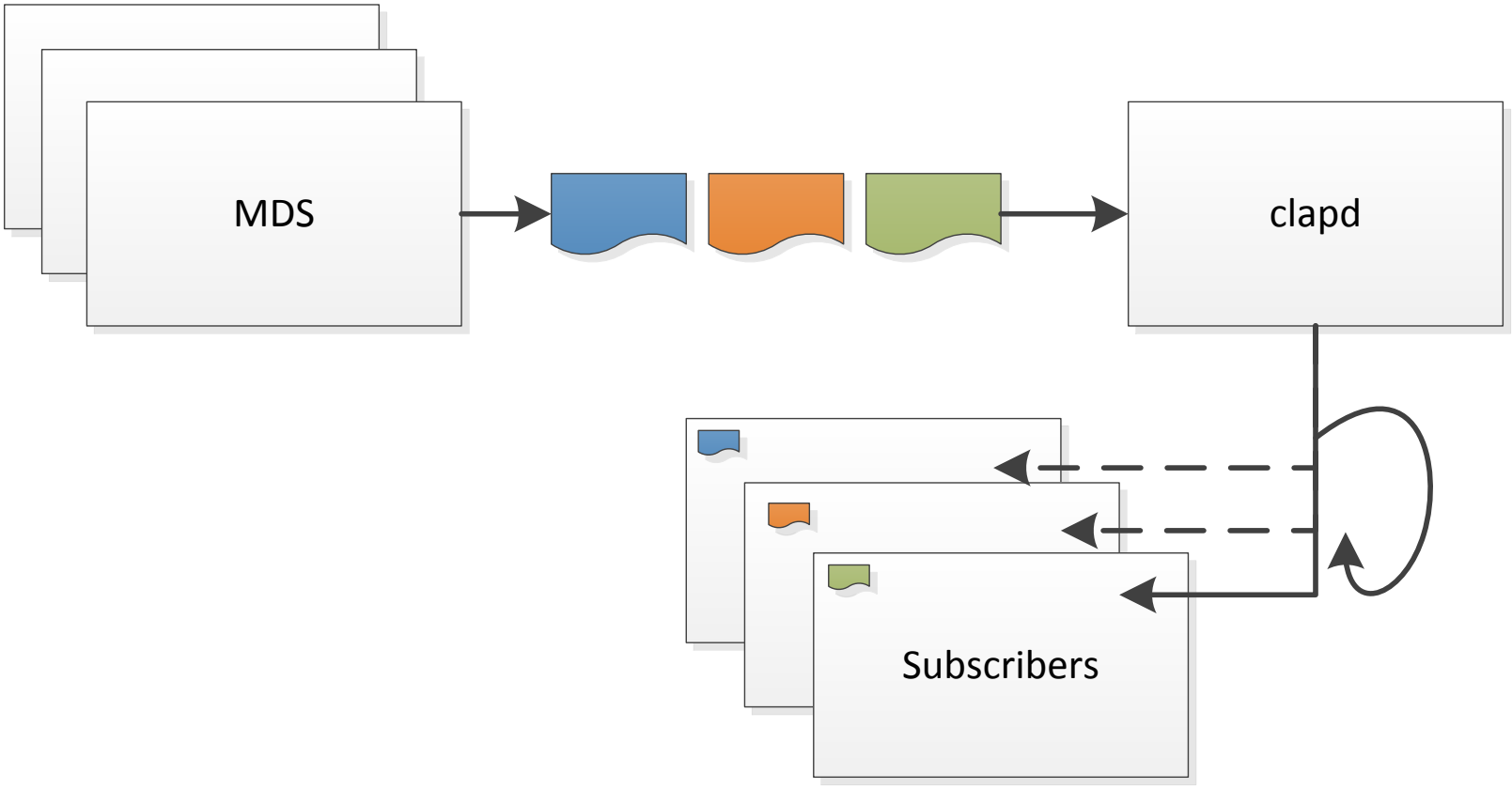
- Policies can internally re-order records as they want
- Records are batch sent to the client
- Policies can decide how to deliver stream to a given client
  - Can group by target FID
  - Can group by source MDS
  - Can rely on simple time windowing

DISTRIBUTION STRATEGIES

LOAD BALANCING

Distribute stream processing between two instances of robinhood

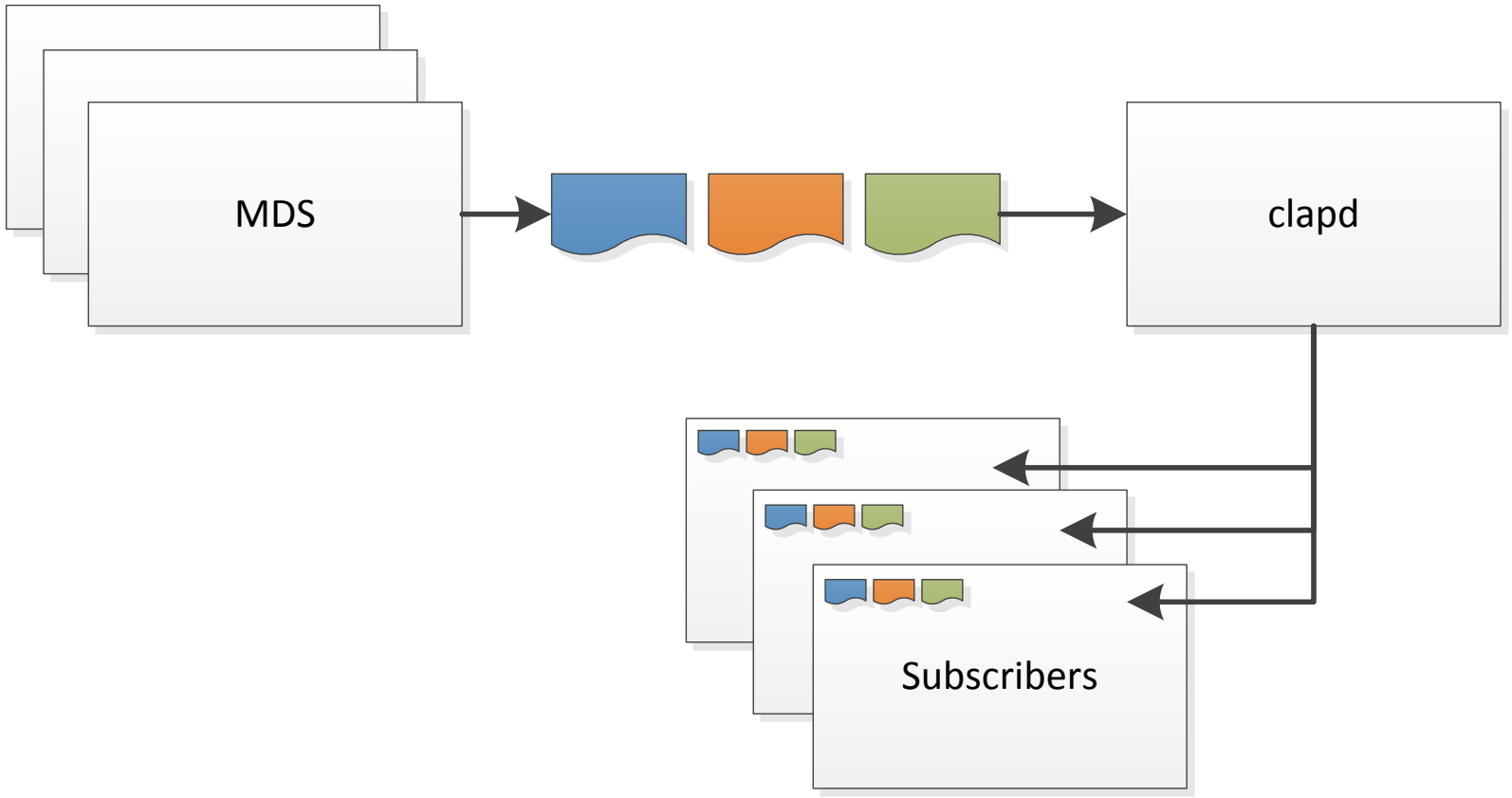
- Round-robin between end readers



BROADCAST

Replicate stream to many ephemeral readers

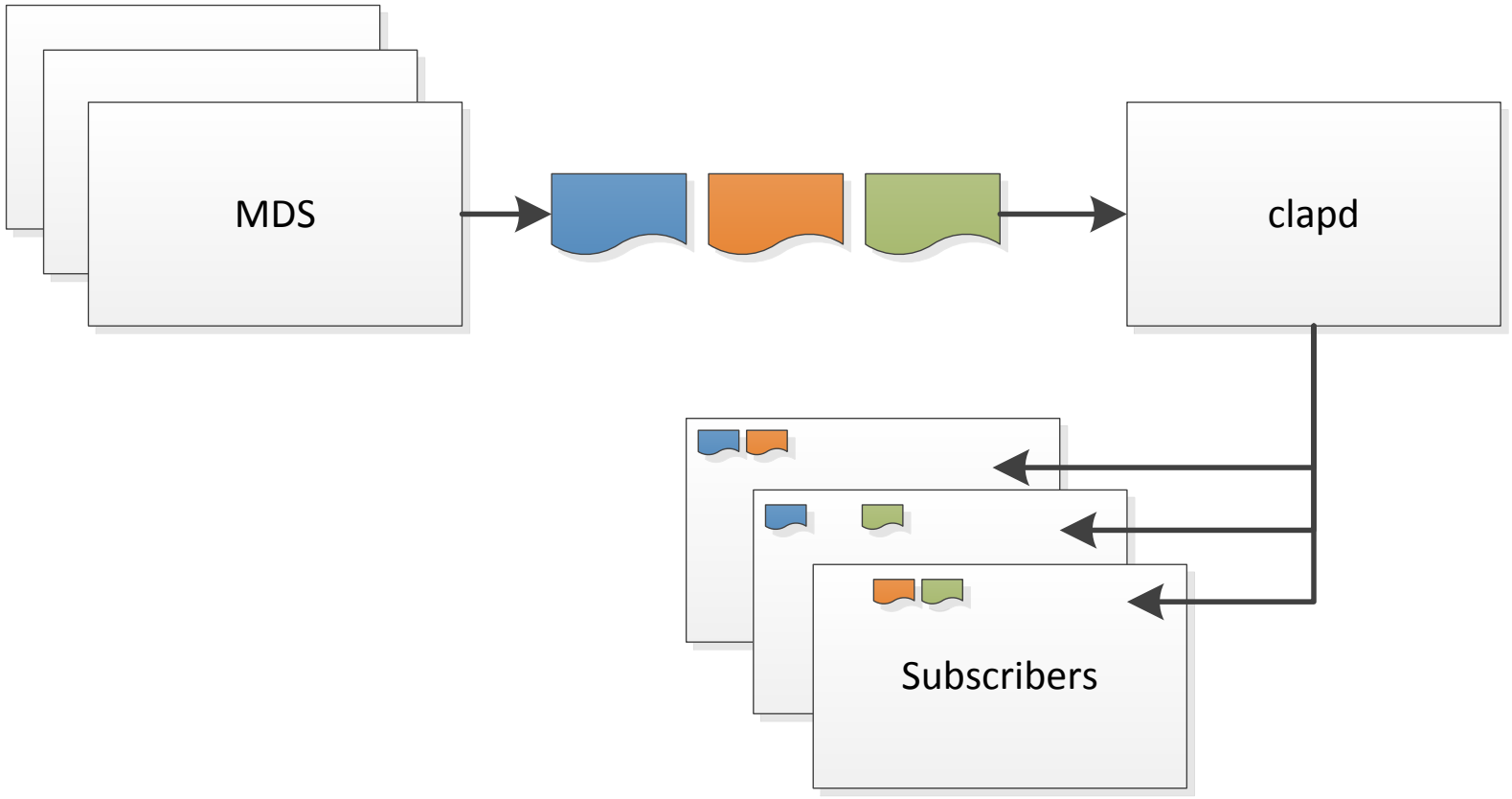
- Publish/Subscribe mechanism



FILTERED BROADCAST

Replicate partial stream (filter out records)

- Publish/Subscribe mechanism, records not matching client filters aren't delivered



CONCLUSION

CONCLUSION

Interesting prospectives

- Already proven easy to extend/experiment with
- Ongoing
  - Write more elaborated policies
  - Make clap able to stack them
  - Implement adaptive batching

Stabilize and mature the project

- Not yet used in production
- Improve resiliency
  - Clients currently can’t recover from a server (clapd) crash
- Profile and optimize using at scale deployments

WANT TO TRY IT?

Disclaimer: clap is still under heavy work ☺

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

https://github.com/cea-hpc/clap.git (soon)

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