A SECURE PUBLIC CACHE FOR YARN APPLICATION RESOURCES

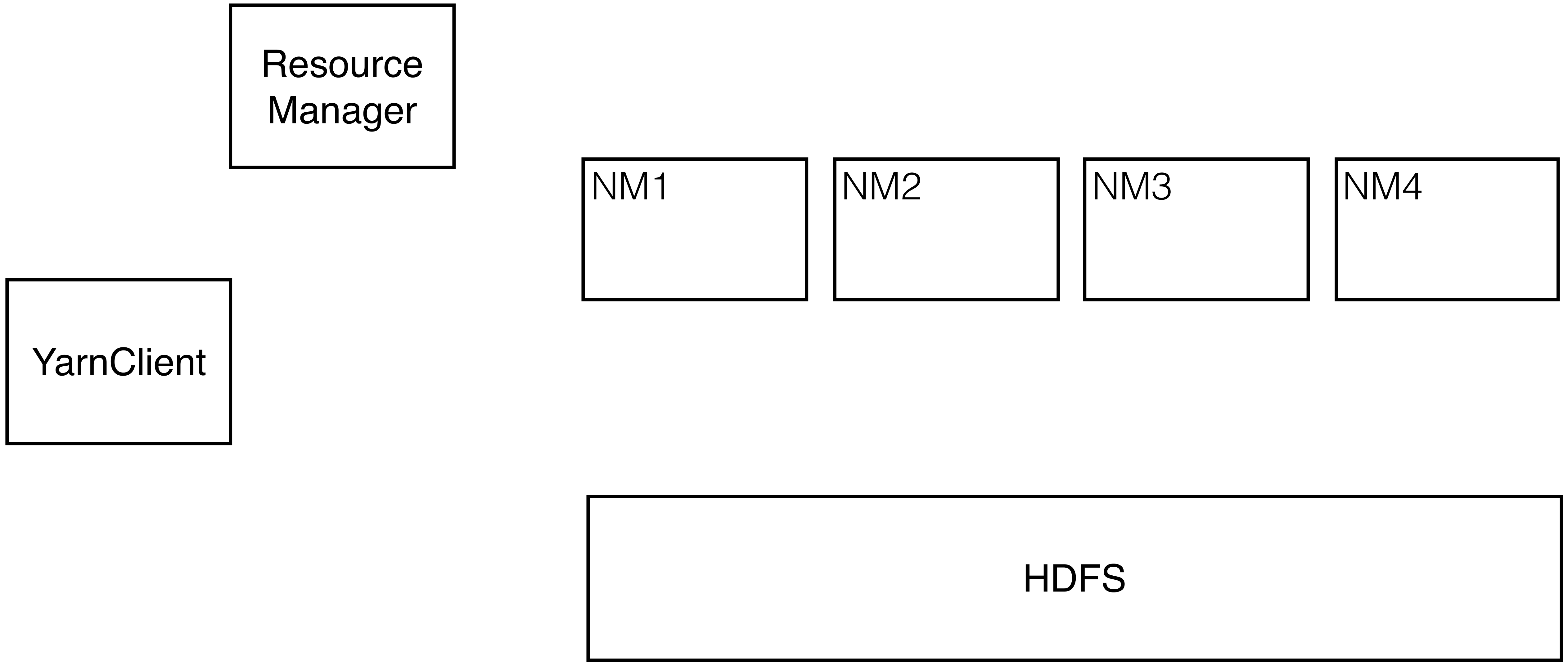
CHRIS TREZZO
@CTREZZO

AGENDA

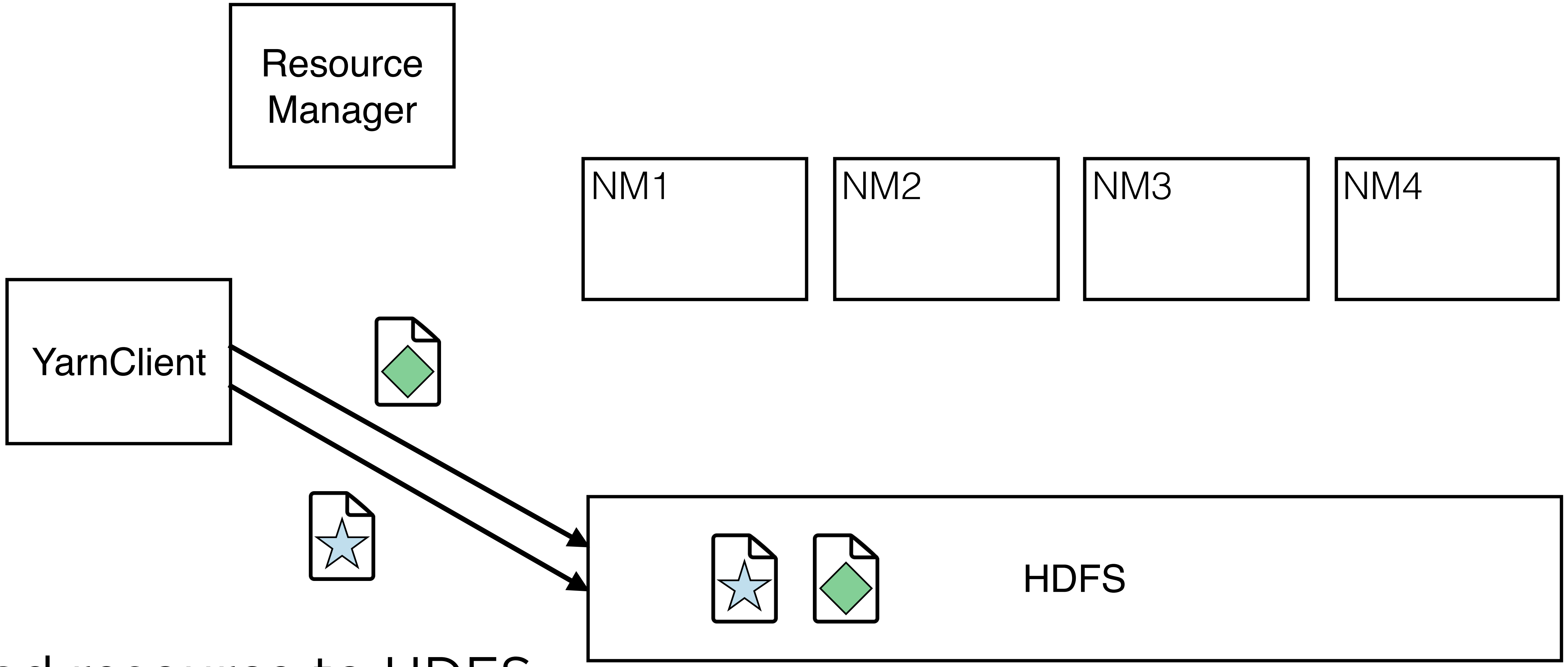
- YARN Localization
- Shared cache design overview
- Adding resources to the cache

- Does it work?
- Anti-patterns that cause churn
- Admin Tips
- Dev Tips

LOCALIZATION OVERVIEW

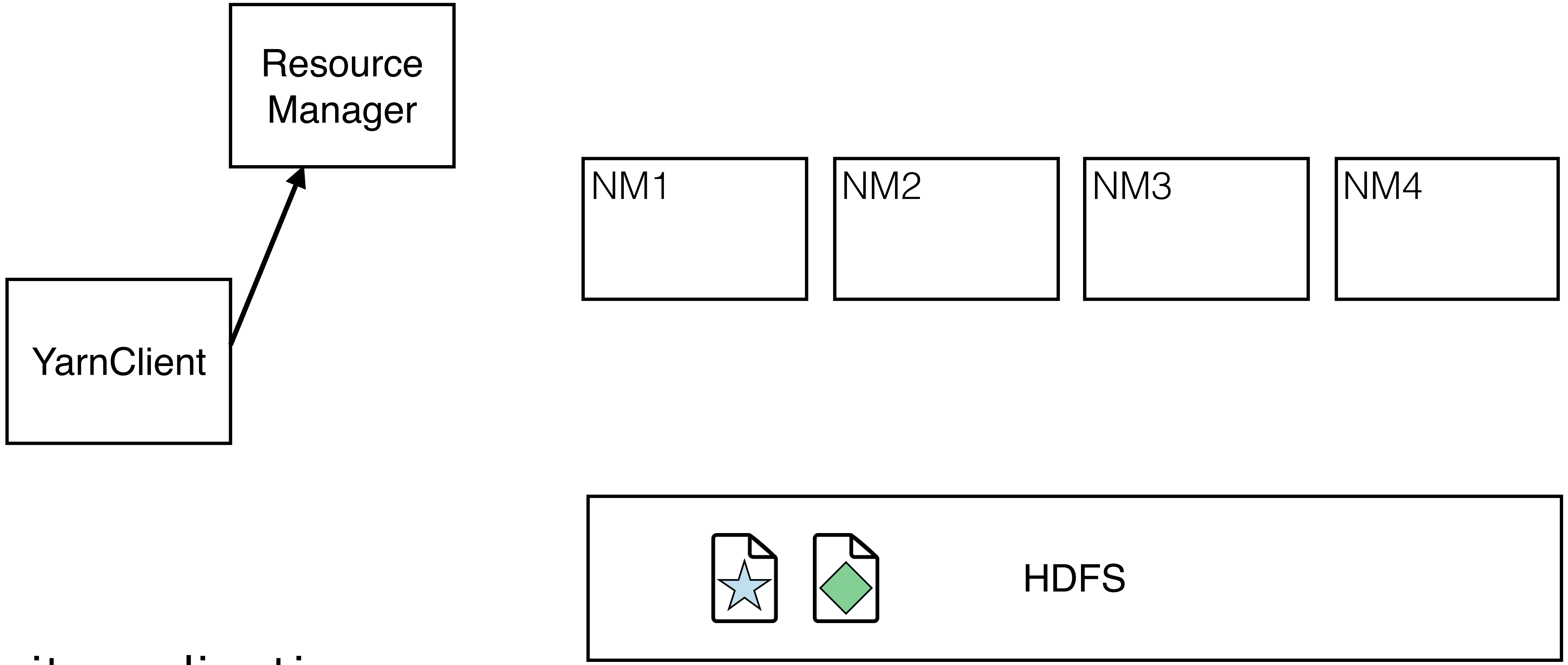


LOCALIZATION OVERVIEW



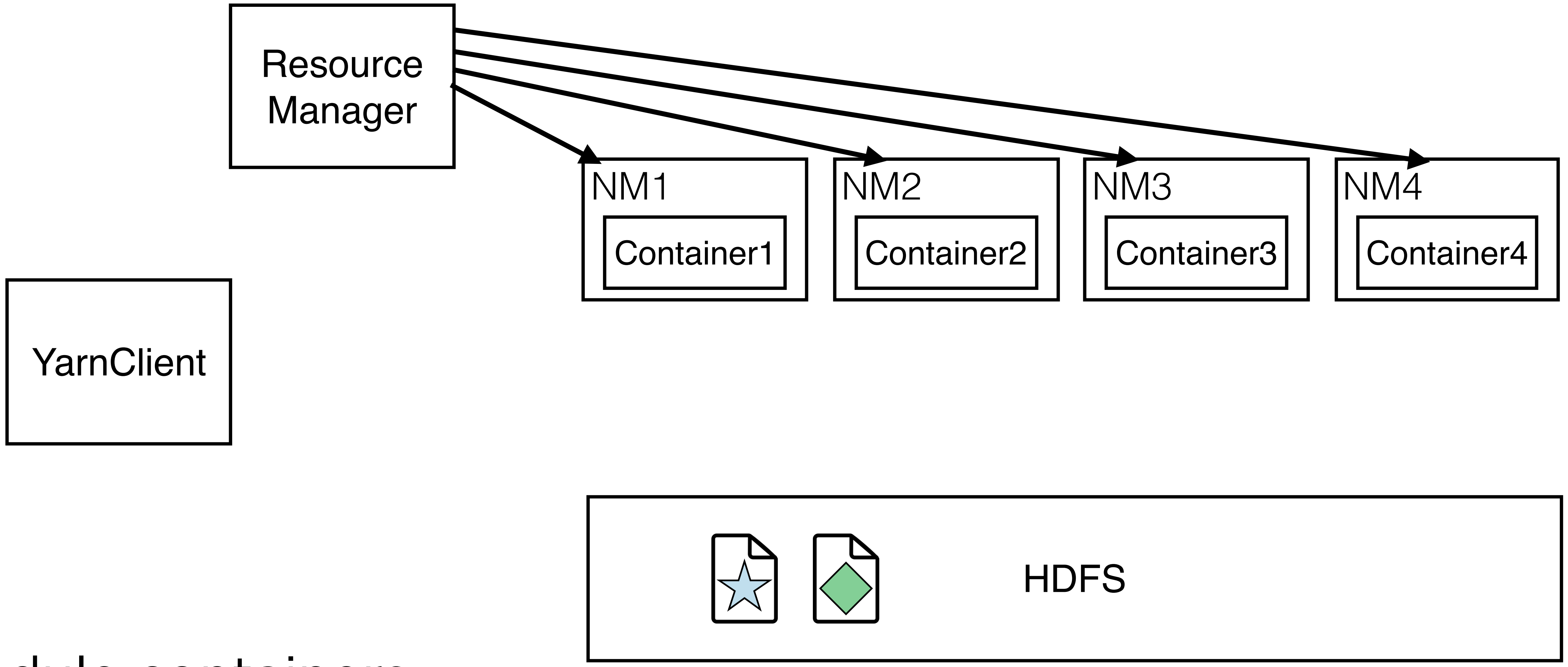
1. Upload resource to HDFS

LOCALIZATION OVERVIEW



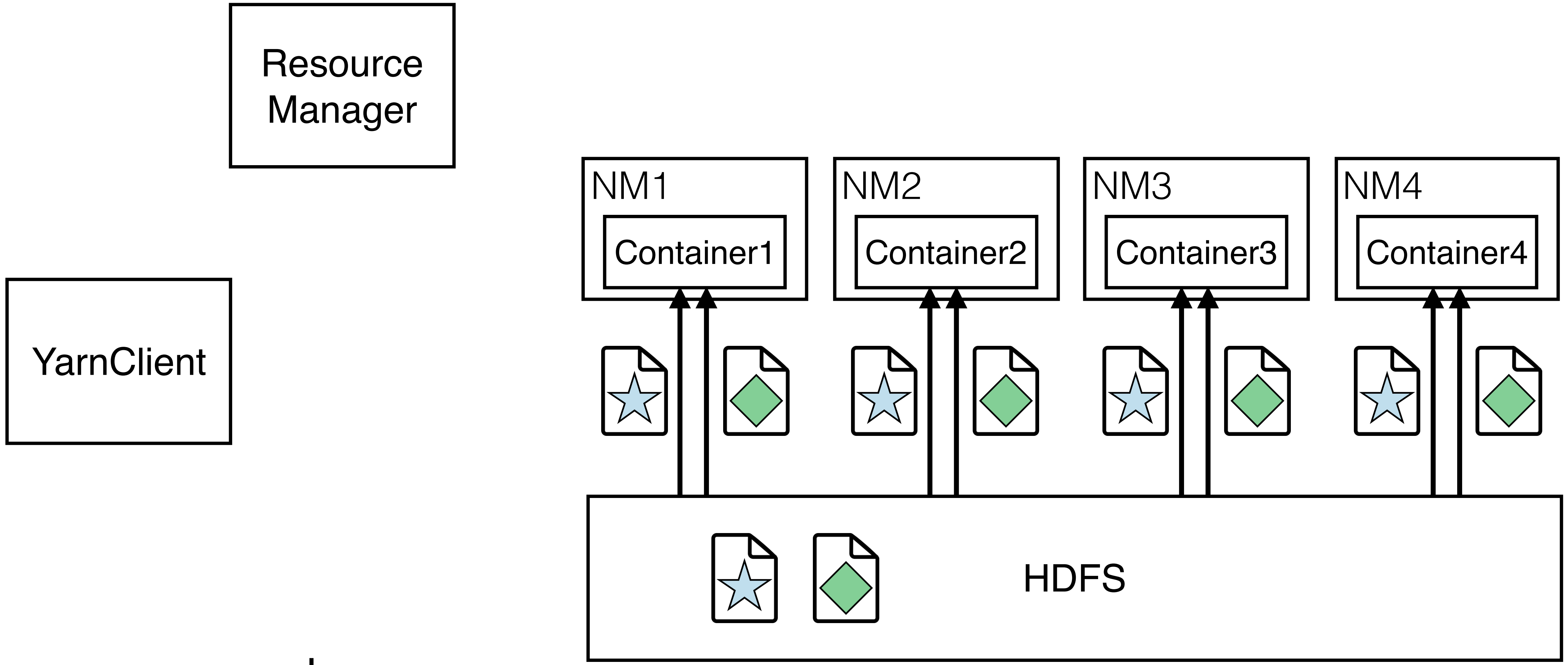
2. Submit application

LOCALIZATION OVERVIEW



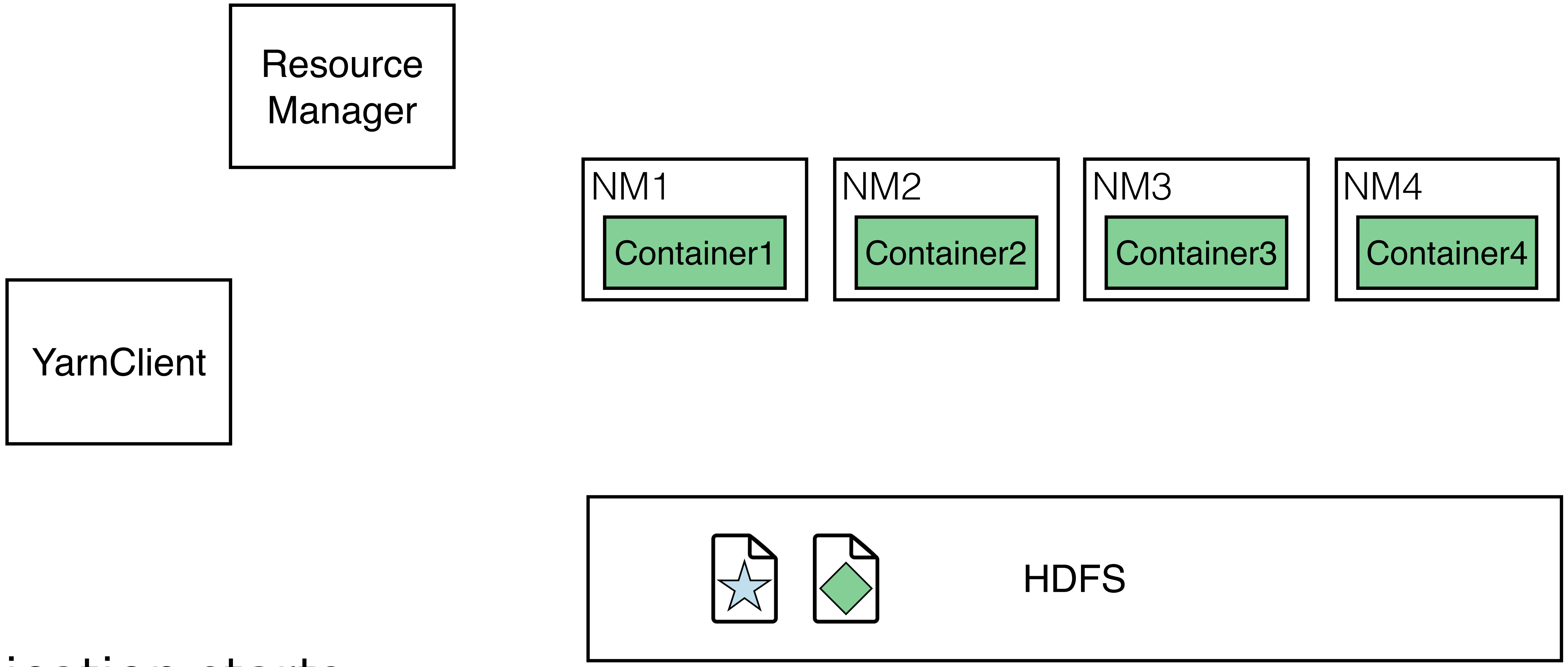
3. Schedule containers

LOCALIZATION OVERVIEW



4. Copy resources to local disk

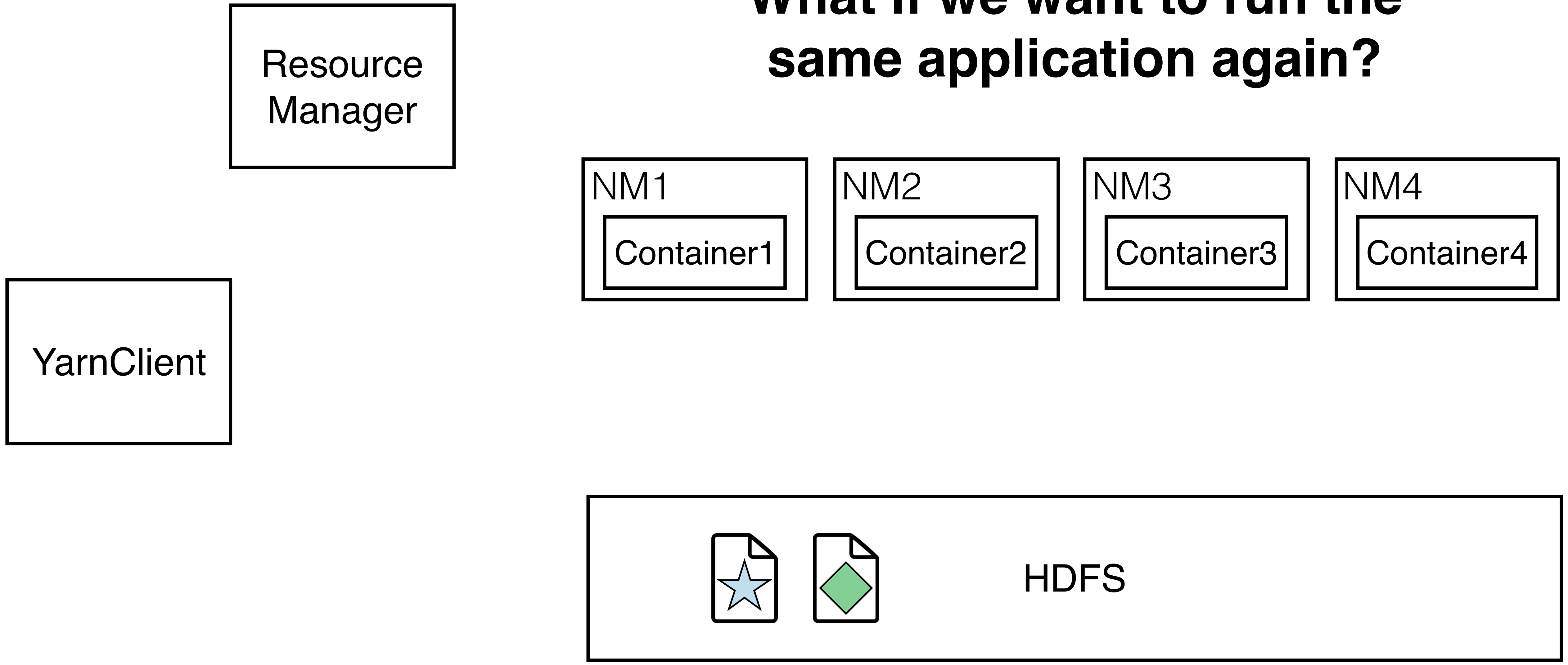
LOCALIZATION OVERVIEW



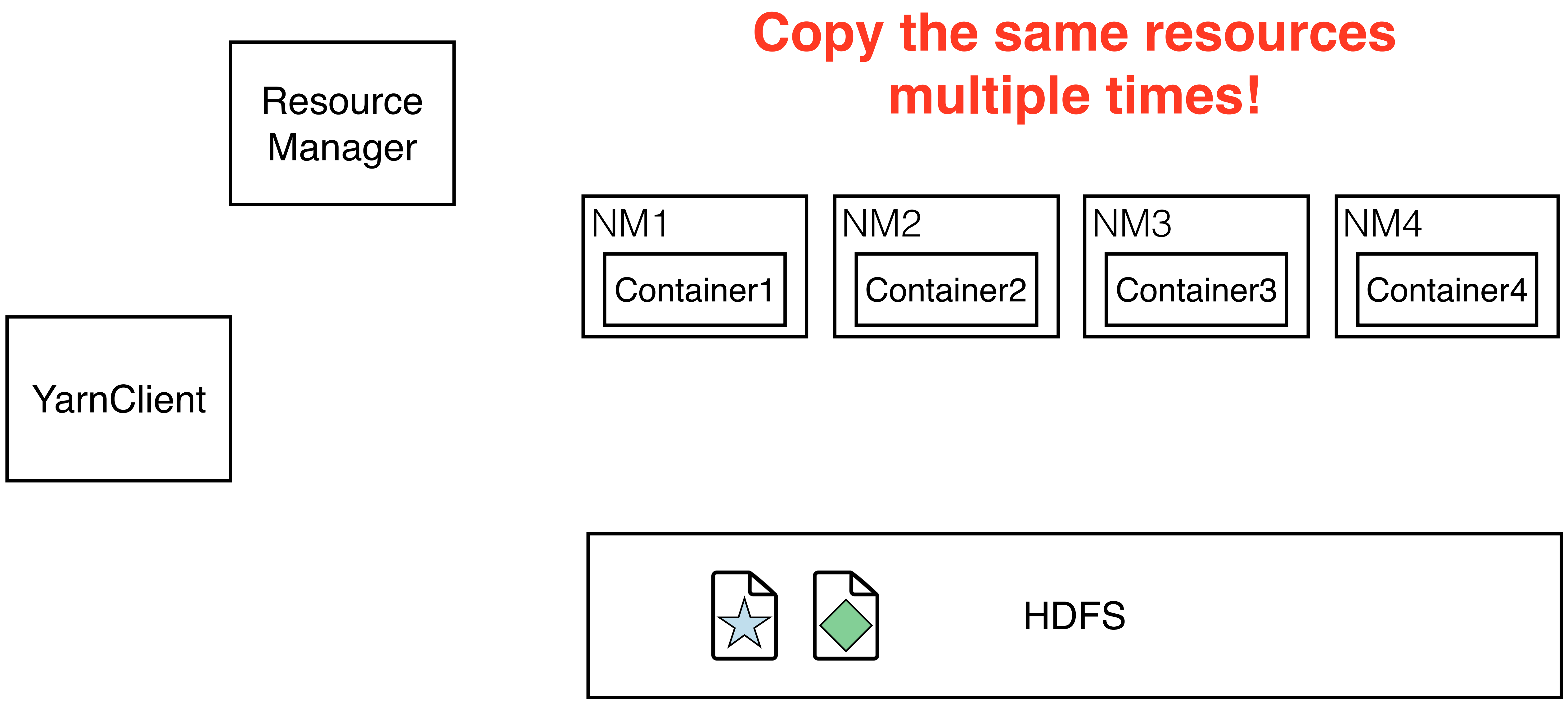
5. Application starts running

LOCALIZATION OVERVIEW

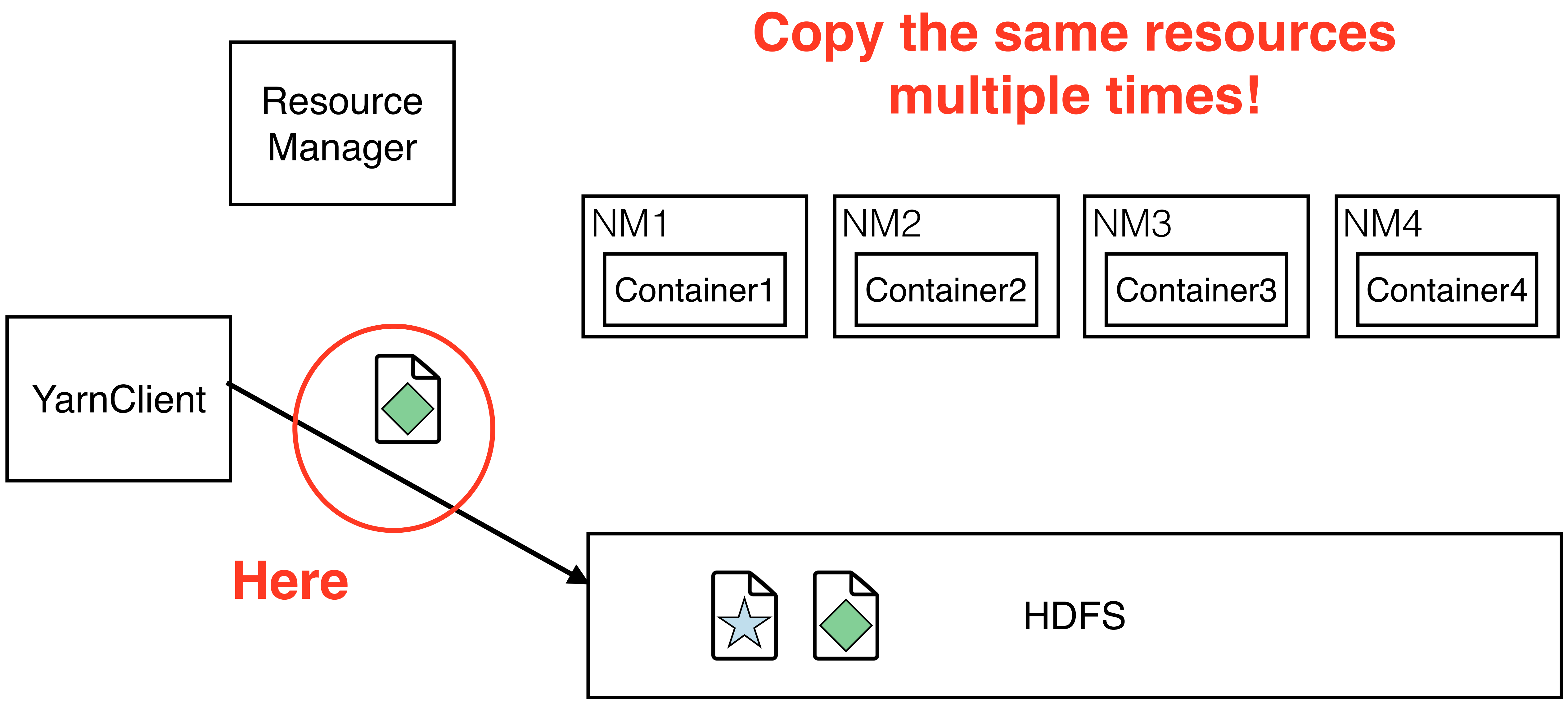
What if we want to run the same application again?



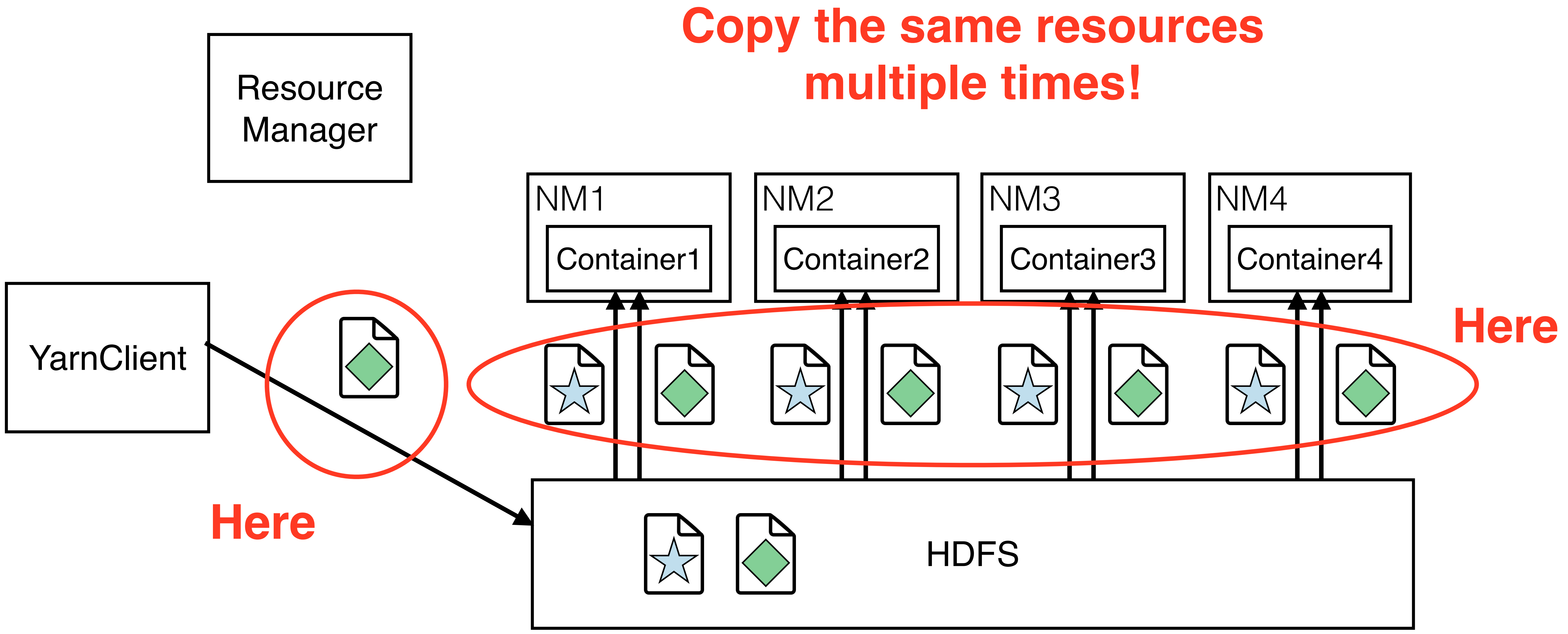
LOCALIZATION OVERVIEW



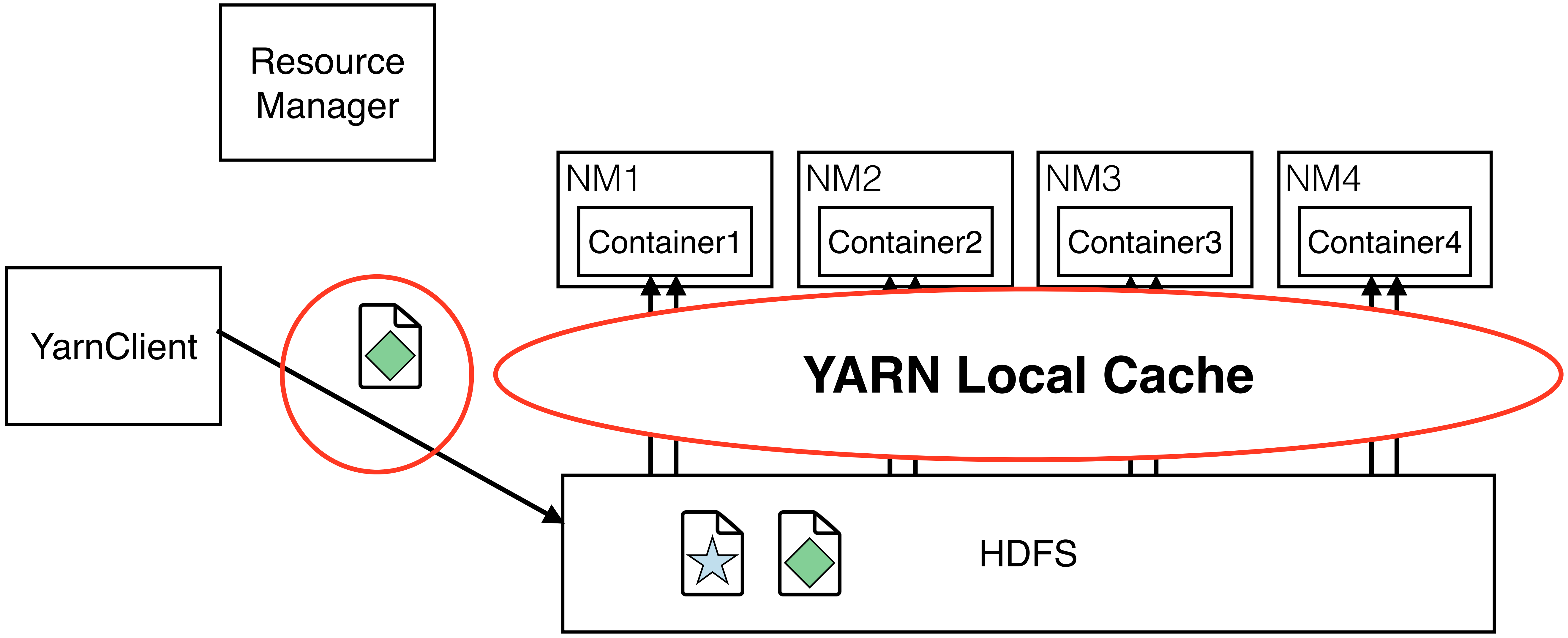
LOCALIZATION OVERVIEW



LOCALIZATION OVERVIEW



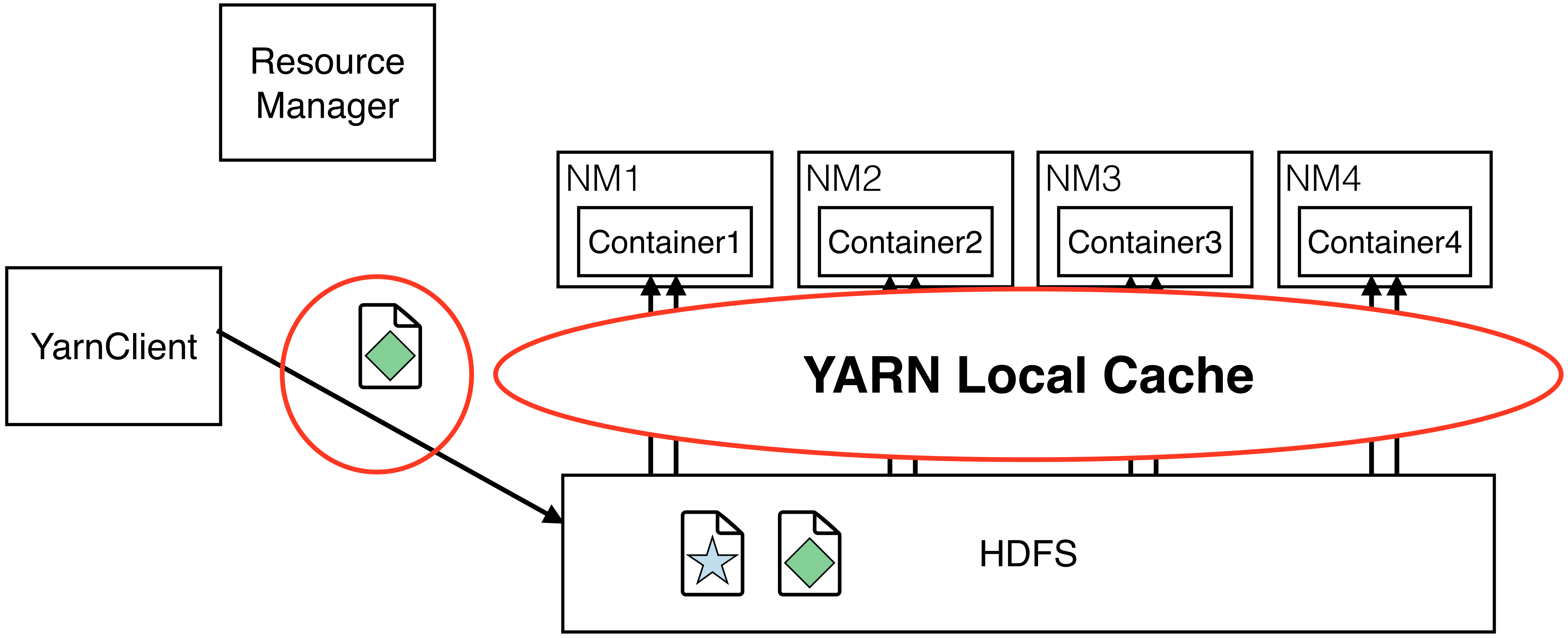
LOCALIZATION OVERVIEW



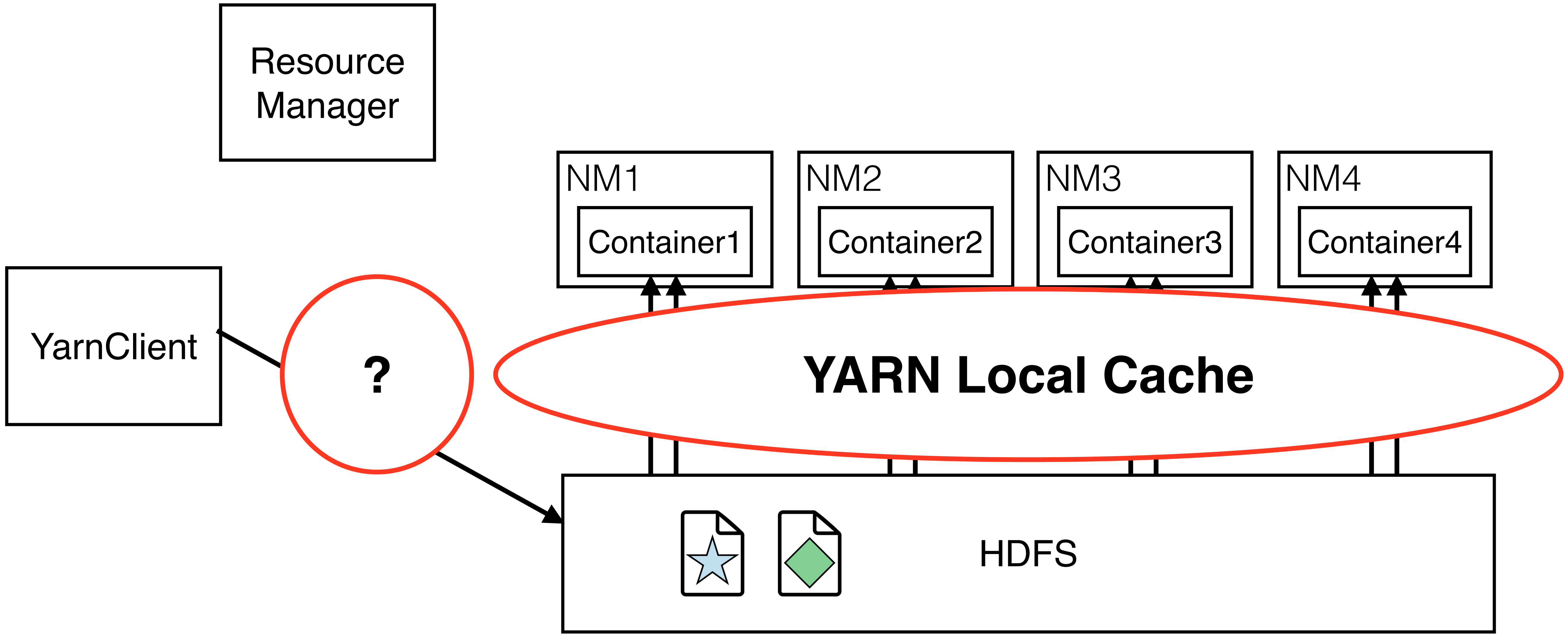
YARN LOCAL CACHE

- Caches application resources on Node Manager local disk
- Maintains a target size via an LRU eviction policy
- Three resource visibilities
  - Public - Shared by everyone
  - User - The same user
  - Application - The same application
- Resources identified based on HDFS path

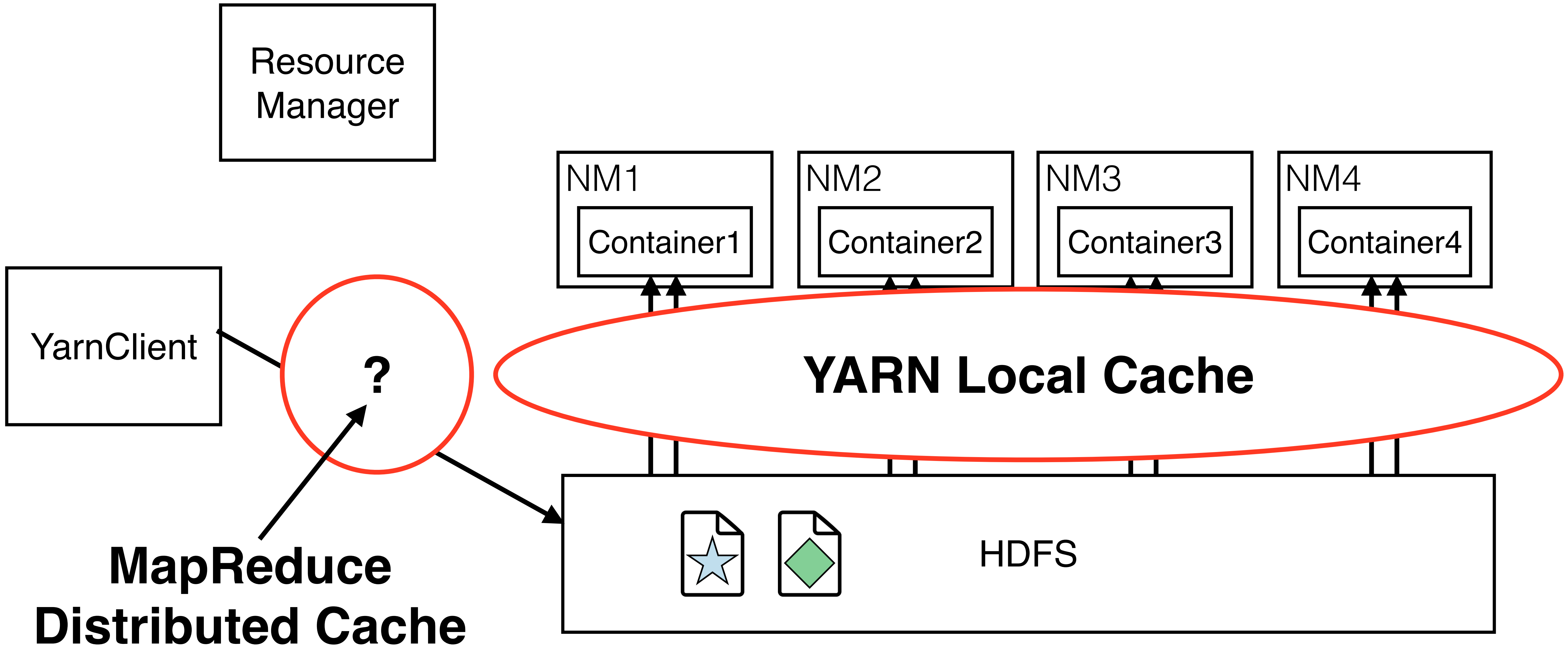
LOCALIZATION OVERVIEW



LOCALIZATION OVERVIEW



LOCALIZATION OVERVIEW



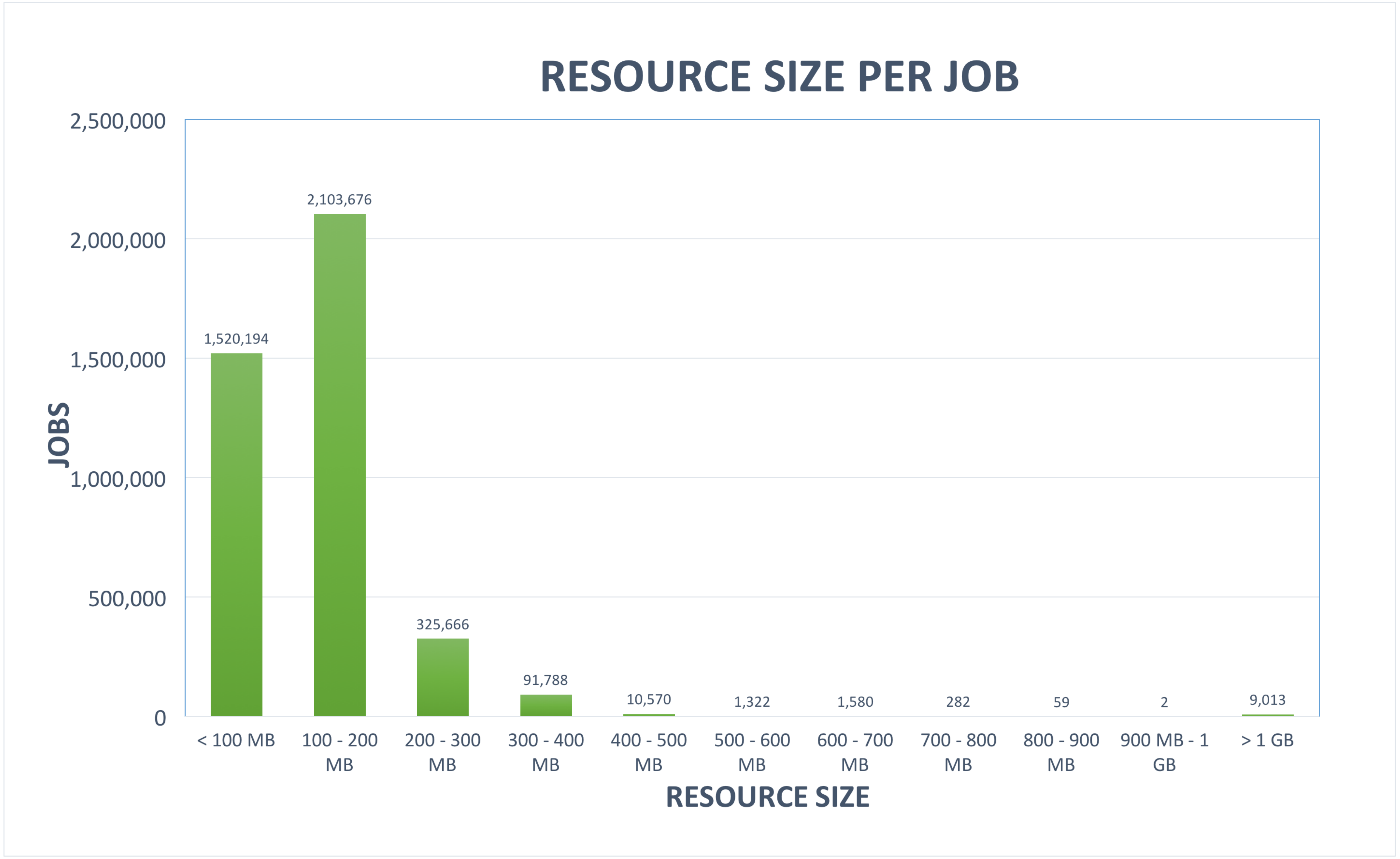
MAPREDUCE DISTRIBUTED CACHE

- Caches MapReduce job resources in HDFS
- API for retrieving known paths in HDFS
- Sets YARN local cache visibility based on HDFS file permissions
- When sharing resources it does not manage:
  - Resource location
  - Resource cleanup
  - Resource integrity

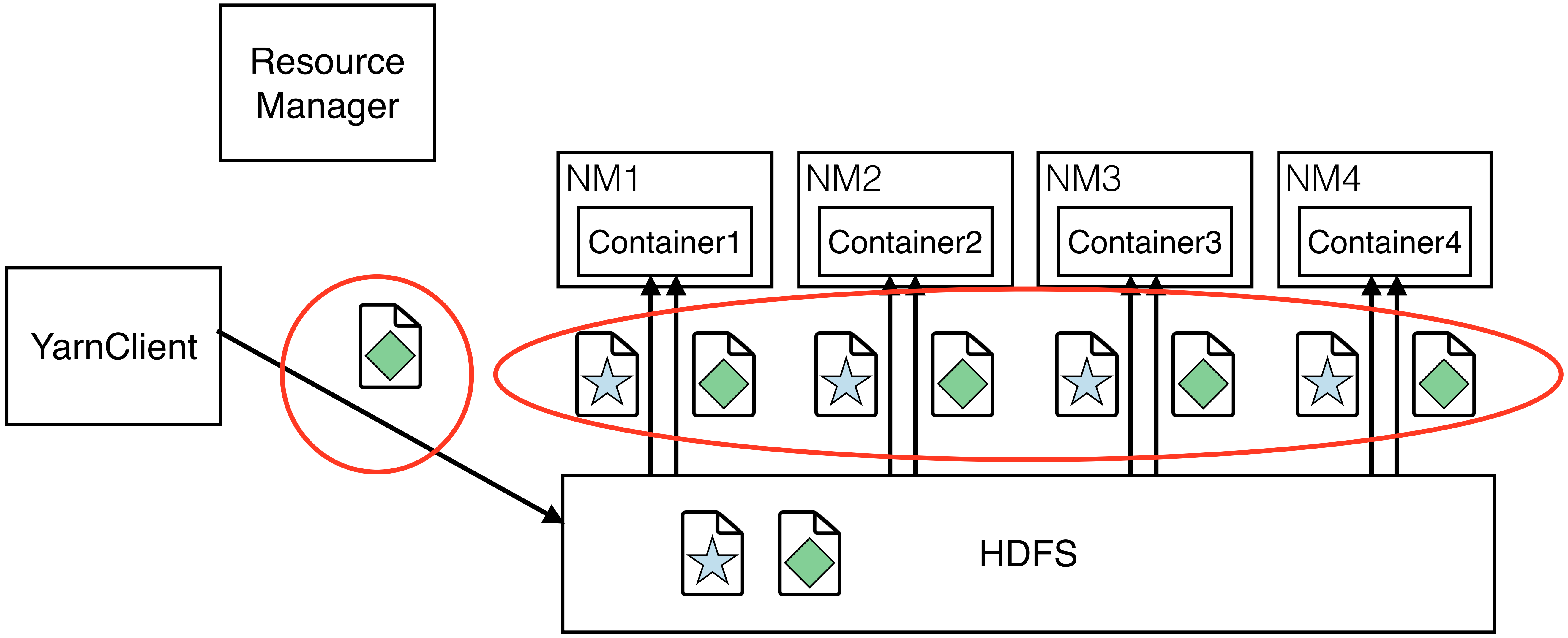
RESOURCE SHARING IN PRACTICE

- Little to no resource sharing between applications
  - Lack of coordination at the HDFS level
  - Majority of applications are MapReduce jobs that upload resources into staging directories (i.e. application level visibility)

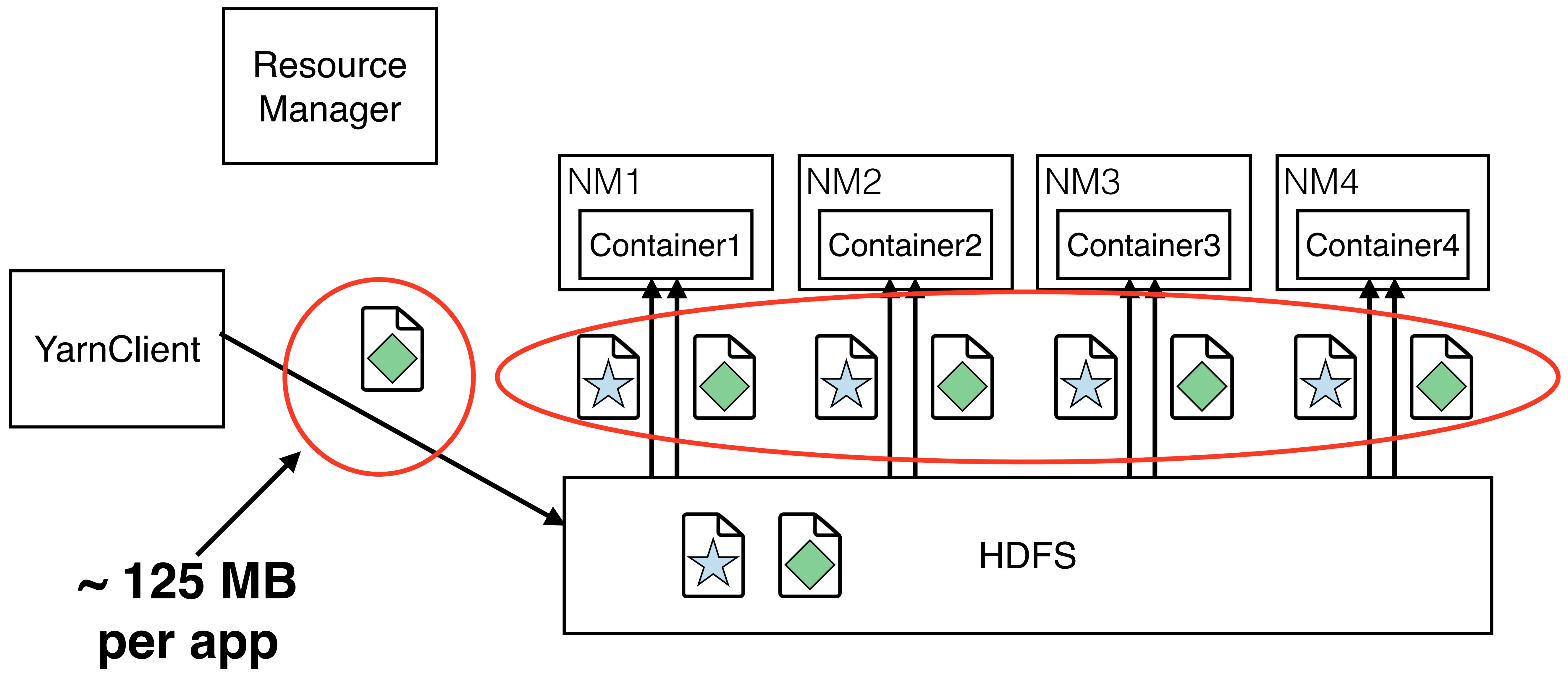
HOW BIG IS THIS PROBLEM?



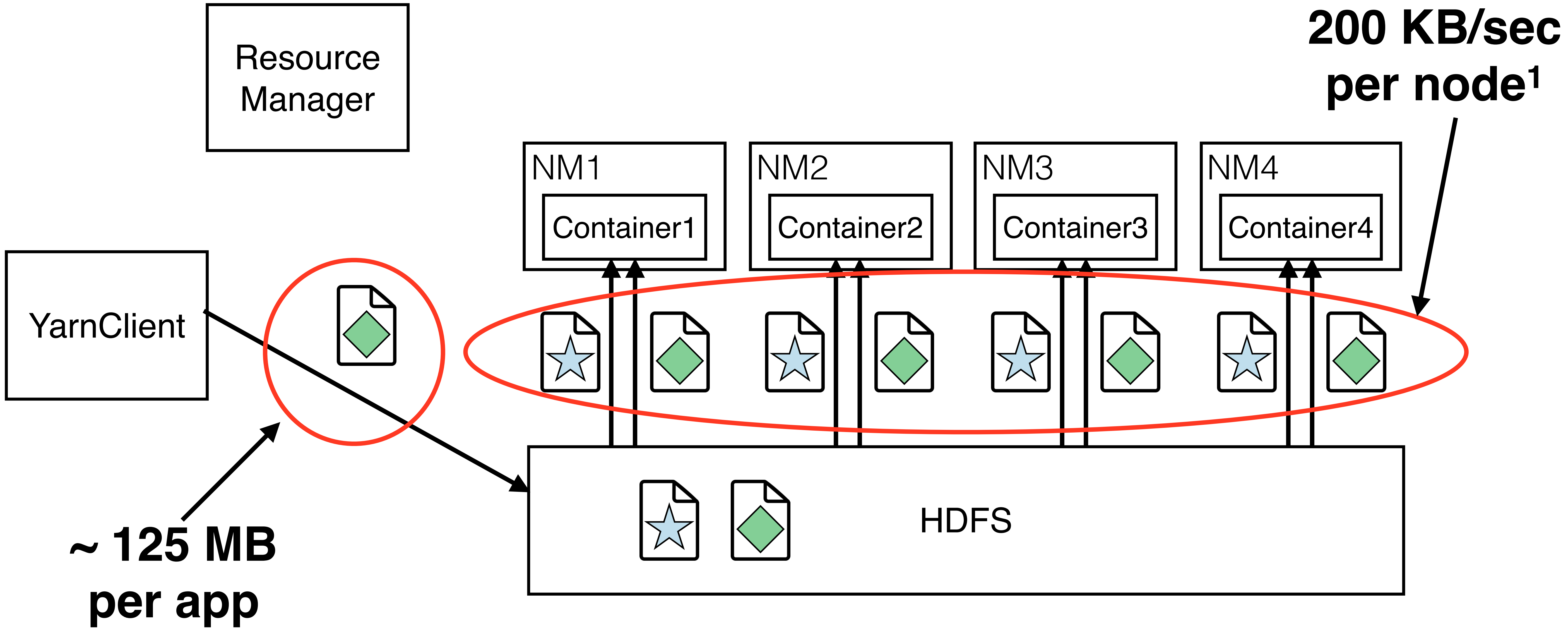
HOW BIG IS THIS PROBLEM?



HOW BIG IS THIS PROBLEM?



HOW BIG IS THIS PROBLEM?



1. Assuming 1.7 containers launched per node per minute

YARN SHARED CACHE

- YARN-1492
- Currently in production at Twitter for ~ 2 years
- 100's of thousands of applications use it a day
  - MapReduce jobs (i.e. Scalding/Cascading)
- Working towards open source release
  - The YARN feature is in 2.7 (Beta)
  - Full MapReduce support coming (MAPREDUCE-5951)

DESIGN GOALS

- Scalable
  - Accommodate large number of cache entries
    - Thousands of cached resources
  - Have minimal load on Namenode and Resource Manager
  - Handle spikes in cache size gracefully

DESIGN GOALS

- Secure
  - Identify resources in the cache based on their contents, not storage location
  - Trust that if the cache says it has a resource, it actually has the resource

DESIGN GOALS

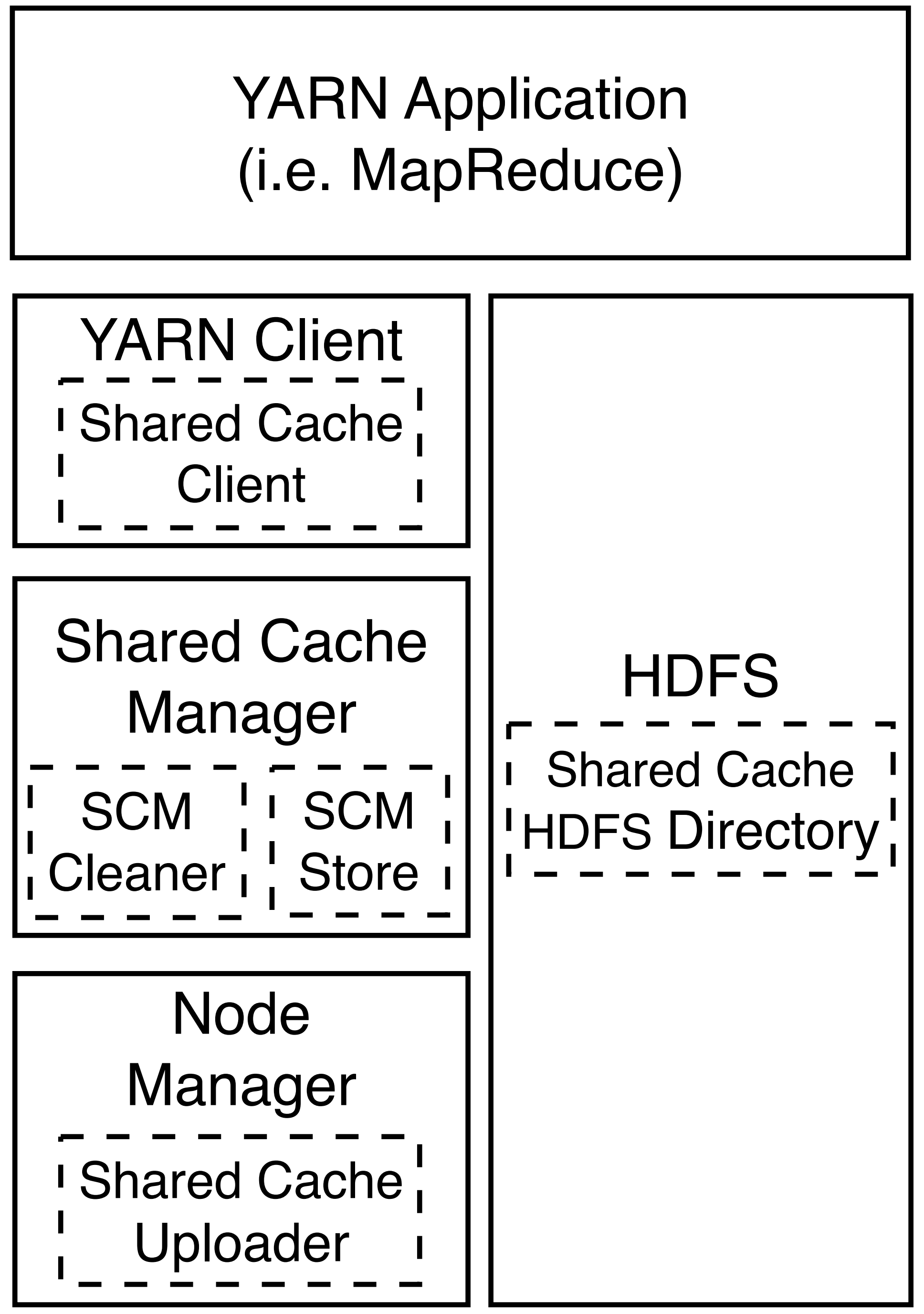
- Fault tolerant
    - YARN applications continue running without shared cache
    - Shared cache should tolerate restarts
        - Either persist or recreate cache resource meta data

DESIGN GOALS

- Transparent
  - YARN application developer perspective
    - Management of the cache (adding/deleting resources)
  - MapReduce job developer perspective
    - Jobs can use the cache with no change

DESIGN OVERVIEW

- Shared Cache Client
- Shared Cache HDFS Directory
- Shared Cache Manager
- Shared Cache Uploader

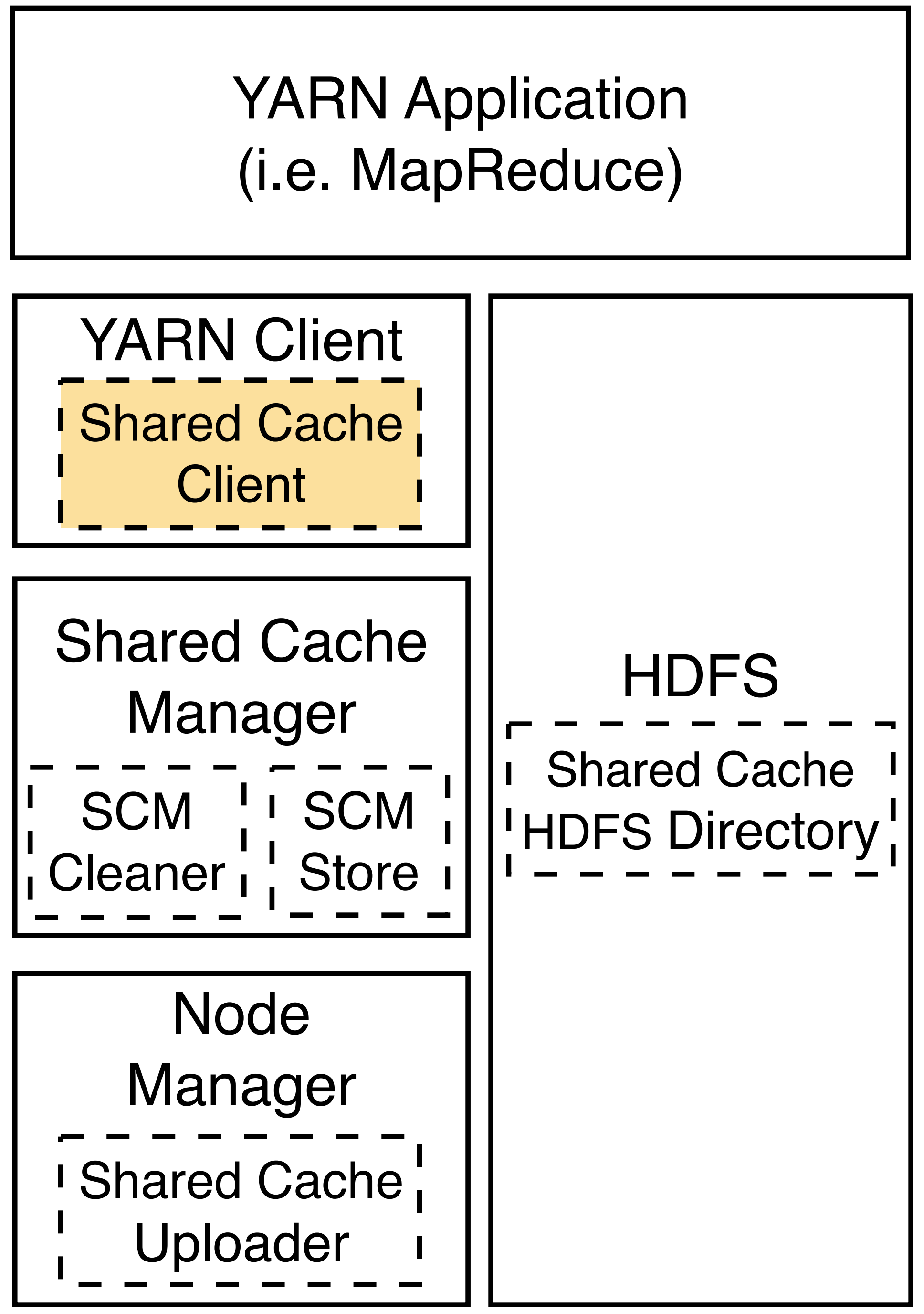


SHARED CACHE CLIENT

- Interacts with shared cache manager
- Responsible for
  - Computing checksum of resources
  - Claiming application resources
- Public API:

```
Path use(String checksum, String appId);
```

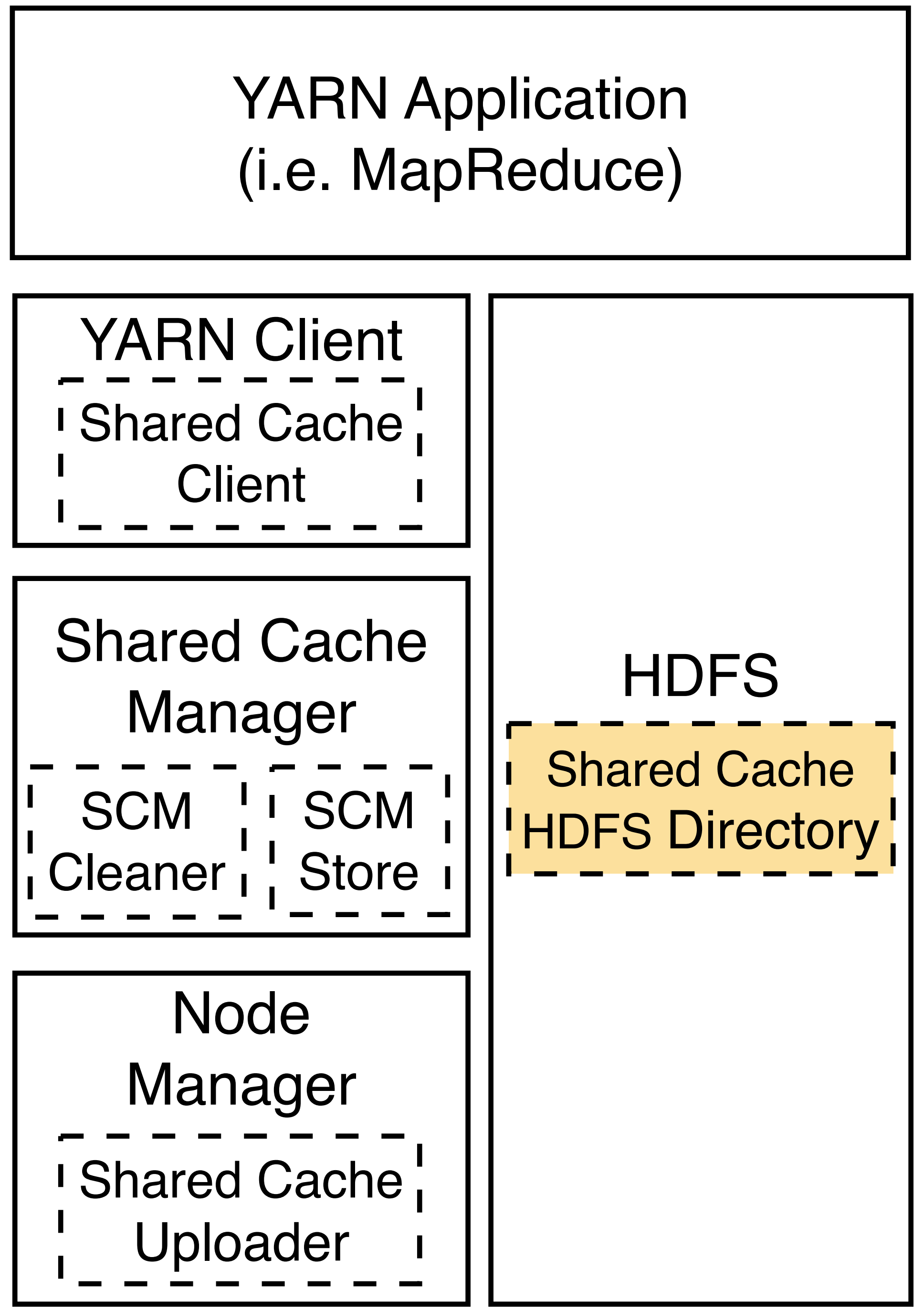
- Return Value
  - HDFS path to resource in cache
  - Null if not in the shared cache



HDFS DIRECTORY

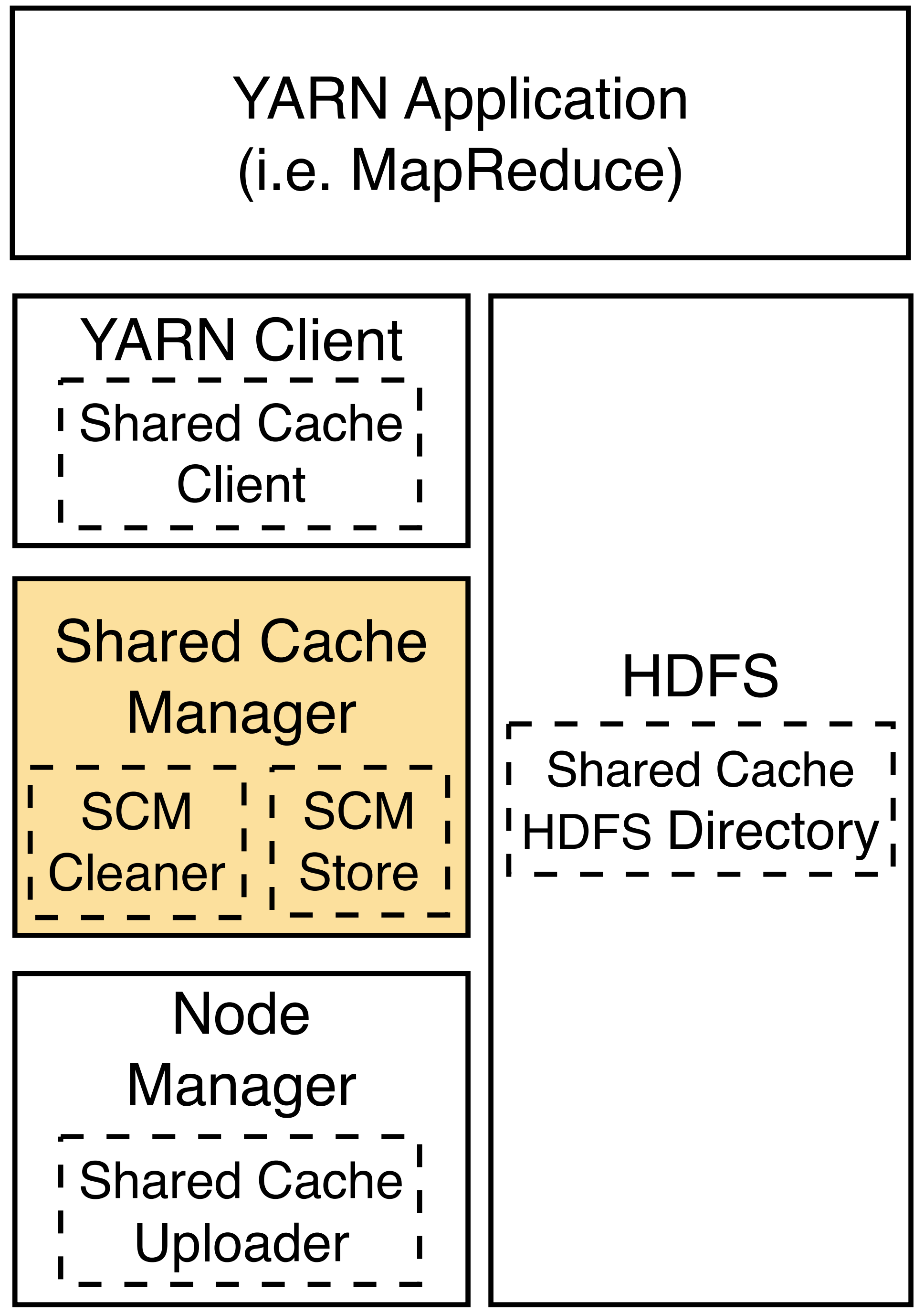
- Stores shared cache resources
- Protected by HDFS permissions
  - Globally readable
  - Writing restricted to trusted user
- Only modified by uploader and cleaner
- Directory structure:

```
/sharedcache/a/8/9/a896857d078/foo.jar
/sharedcache/5/0/f/50f11b09f87/bar.jar
/sharedcache/a/6/7/a678cb1aa8f/job.jar
```



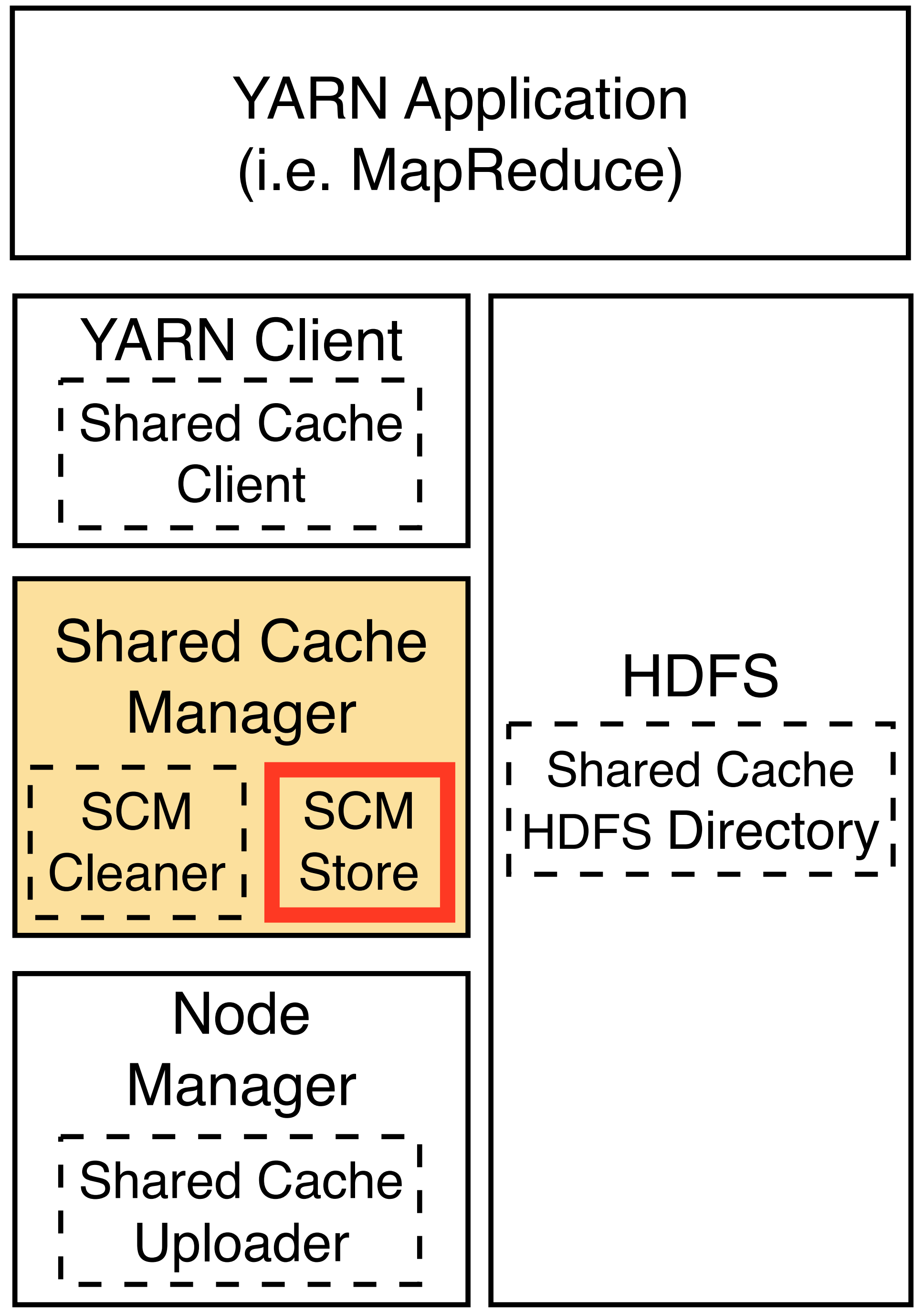
SHARED CACHE MANAGER

- Serves requests from clients
- Manages resources in the shared cache
  - Metadata
  - Persisted resources in HDFS



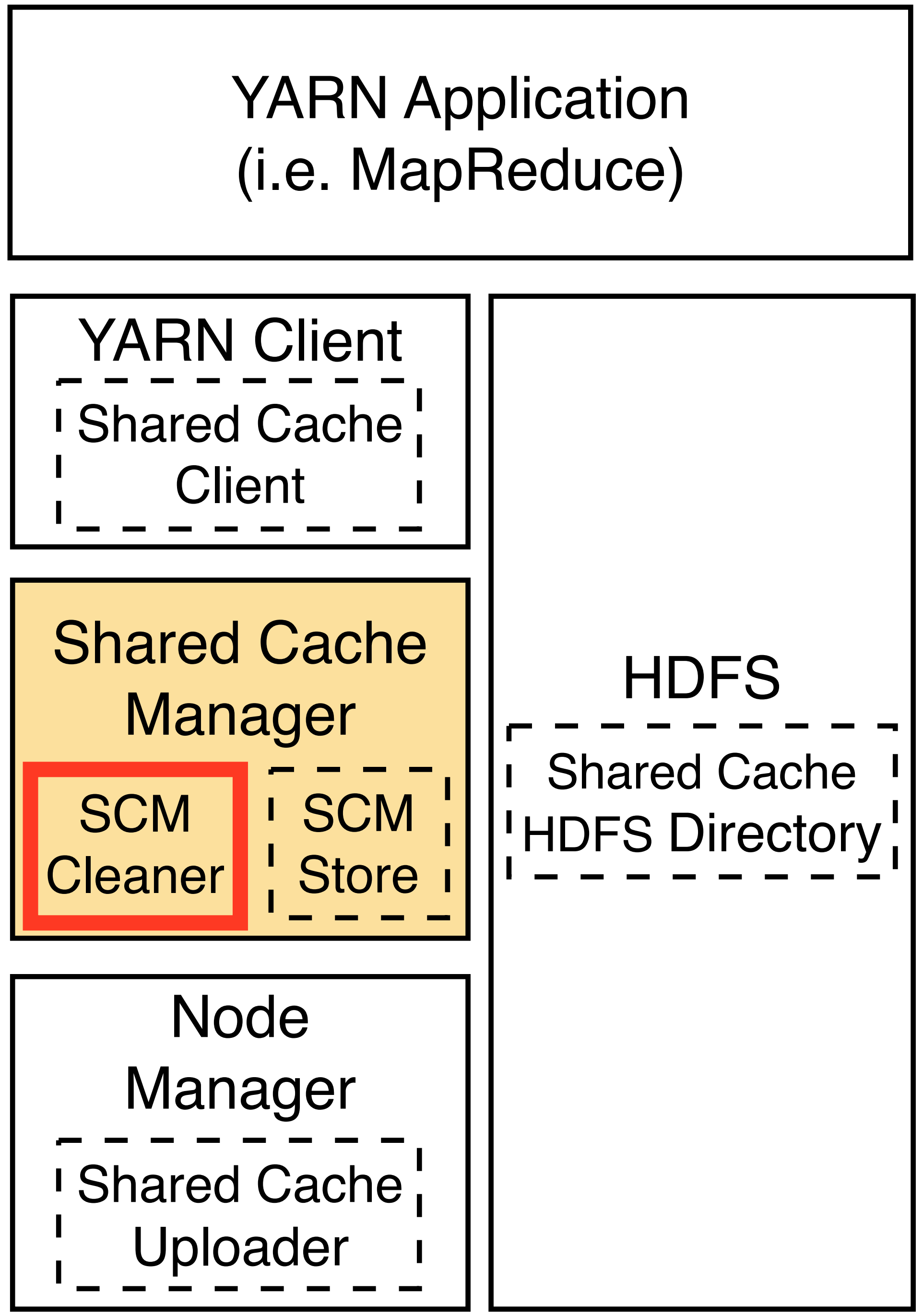
SHARED CACHE MANAGER

- Store
  - Maintains cache metadata
    - Resources in the shared cache
    - Applications using resources
  - Implementation is pluggable
    - Currently uses in-memory store
    - Recreates state after restart



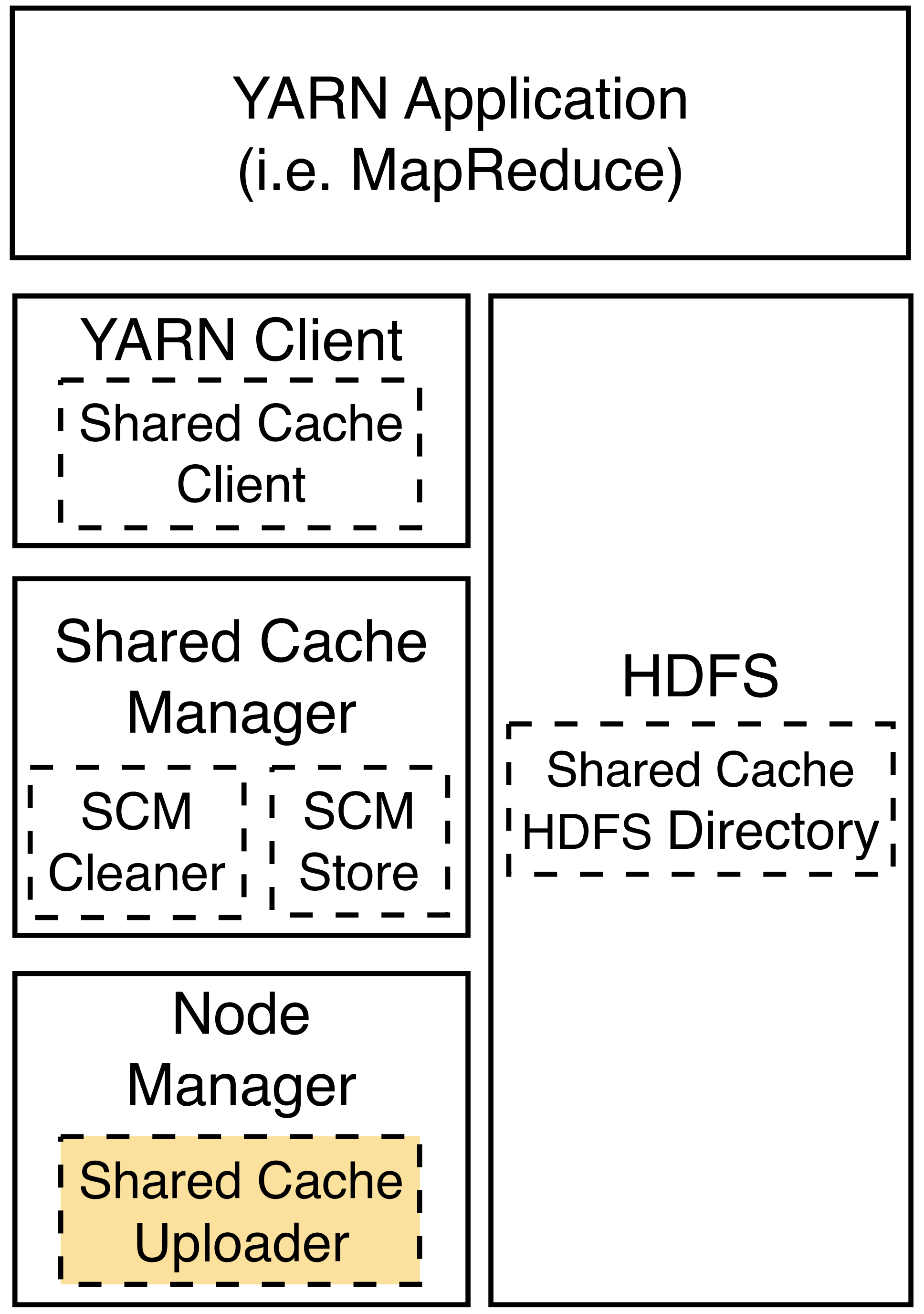
SHARED CACHE MANAGER

- Cleaner
  - Maintains persisted resources in HDFS
  - Runs periodically (default: once a day)
- Evicts resource if both of the following hold:
  - Resource has exceeded stale period (default: 1 week)
  - There are no live applications using the resource

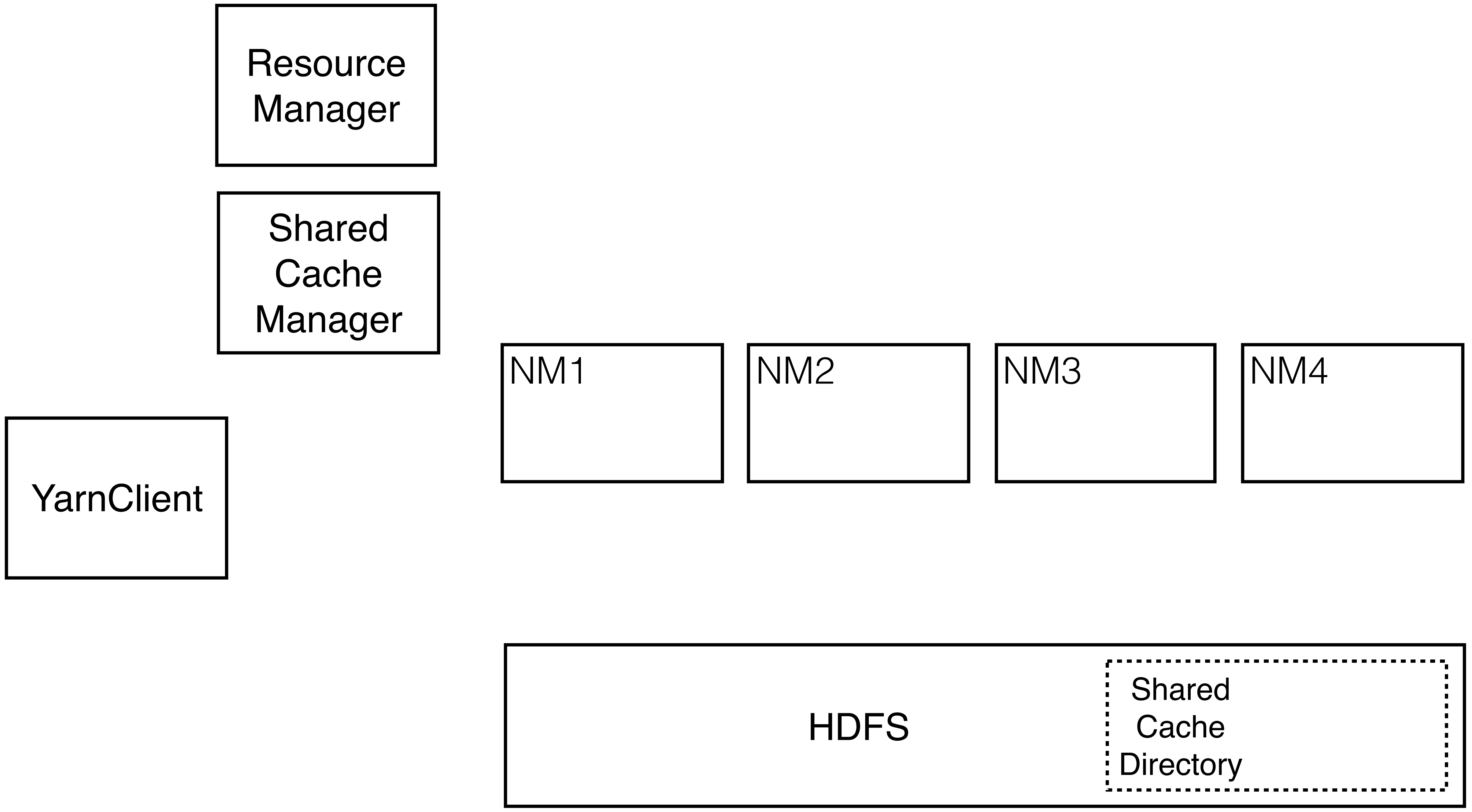


SHARED CACHE UPLOADER

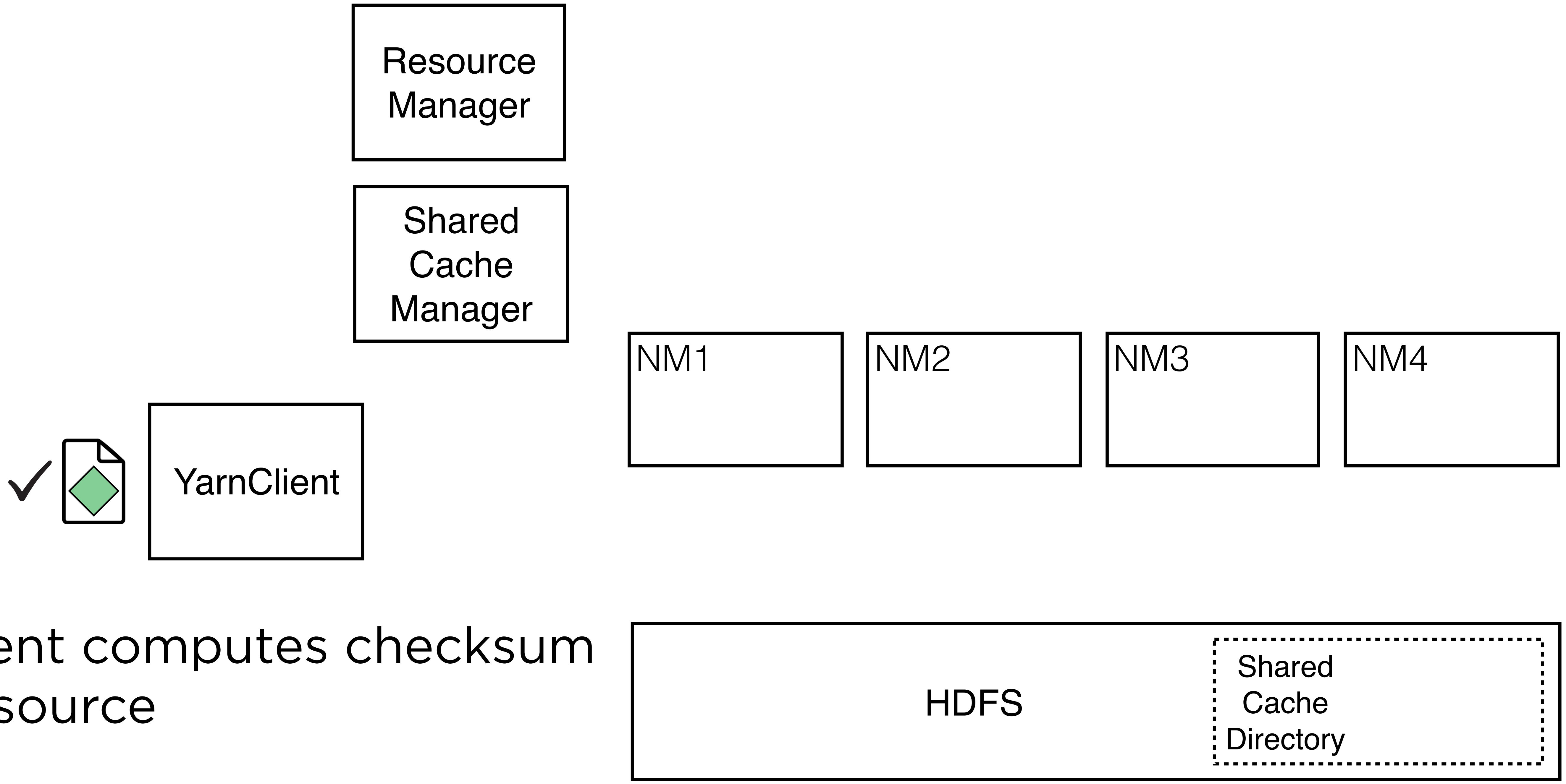
- Runs on Node Manager
- Adds resources to the shared cache
  - Verifies resource checksum
  - Uploads resource to HDFS
  - Notifies the shared cache manager
- Asynchronous from container launch
- Best-effort: failure does not impact running applications



ADDING TO THE CACHE

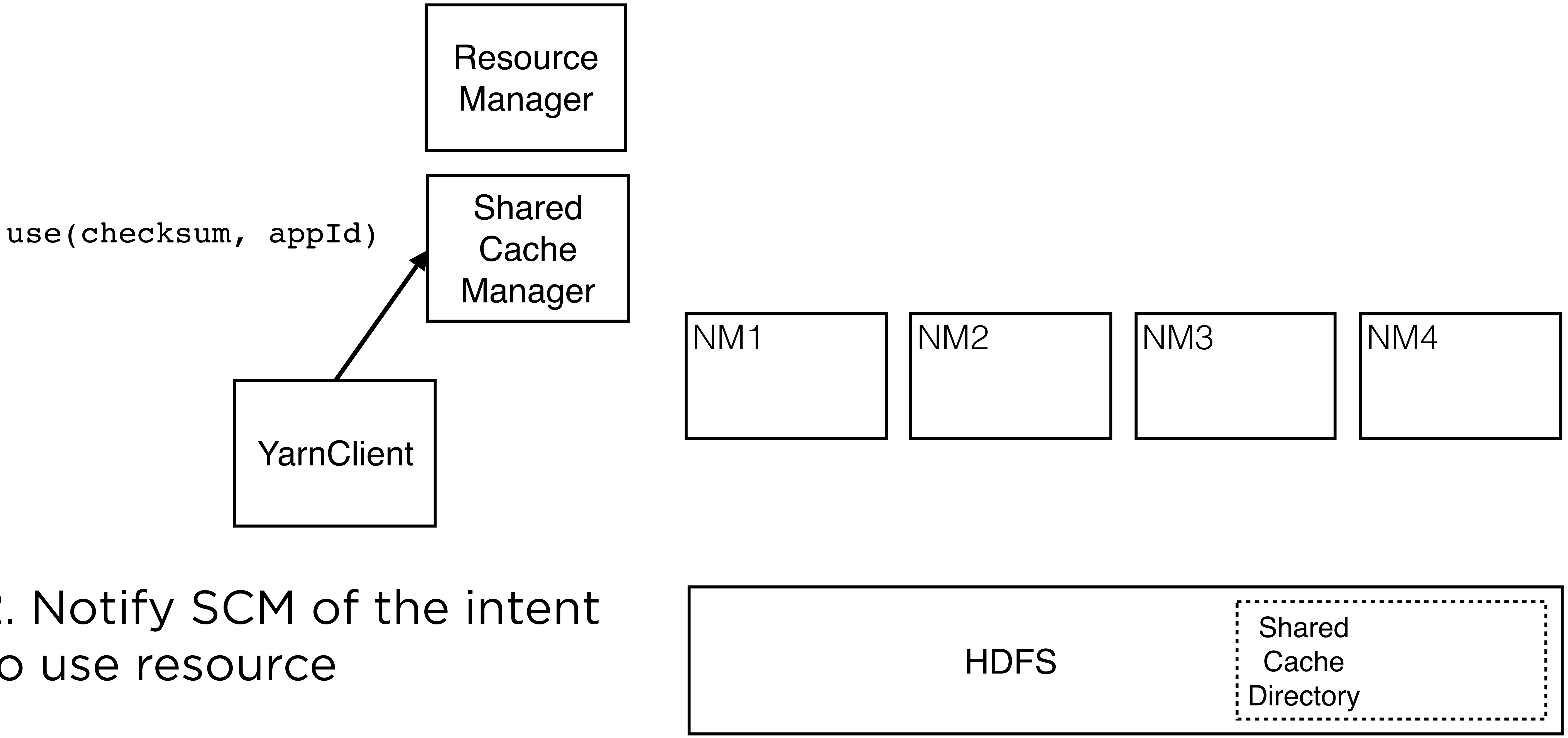


ADDING TO THE CACHE



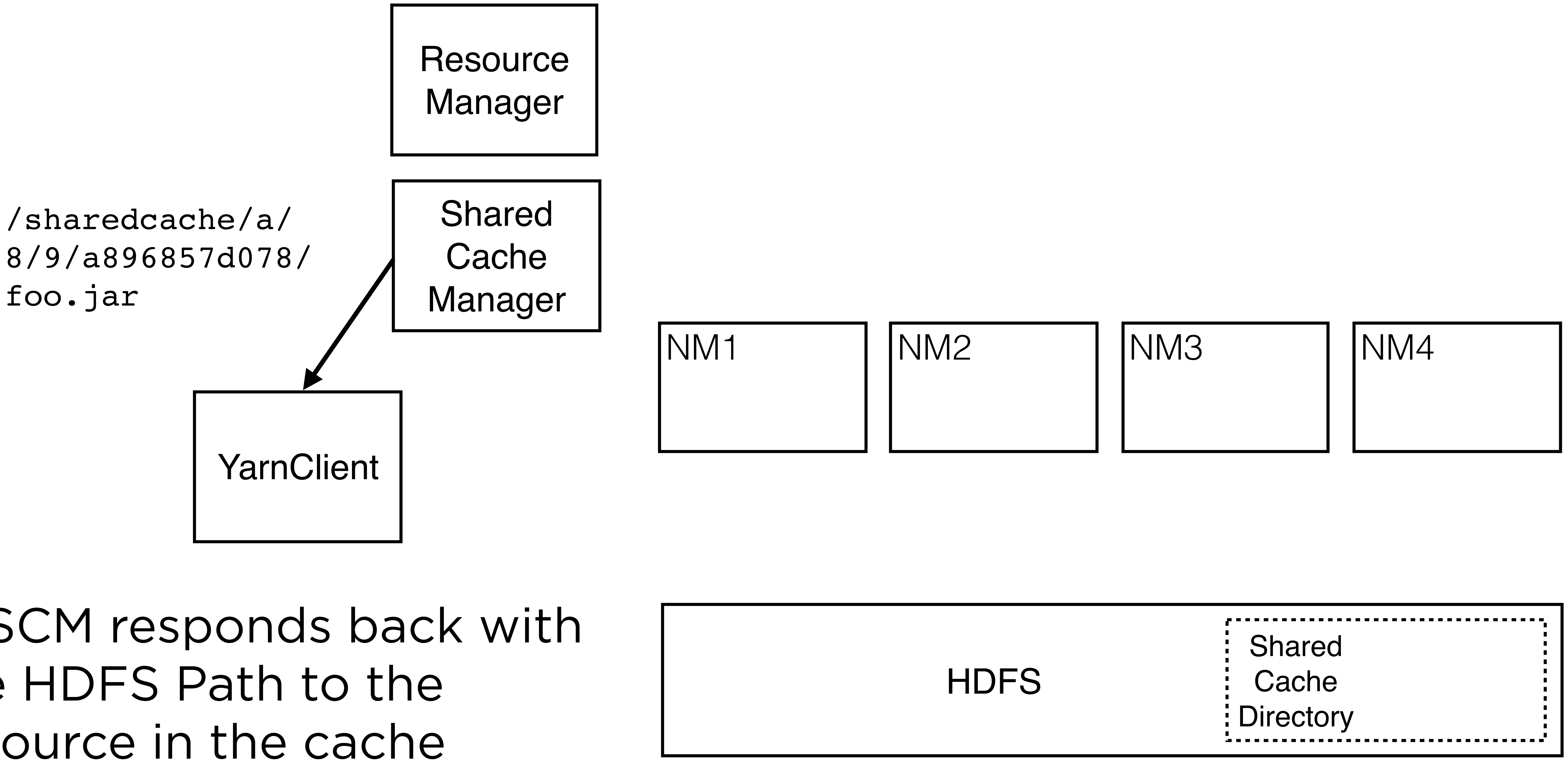
1. Client computes checksum of resource

ADDING TO THE CACHE



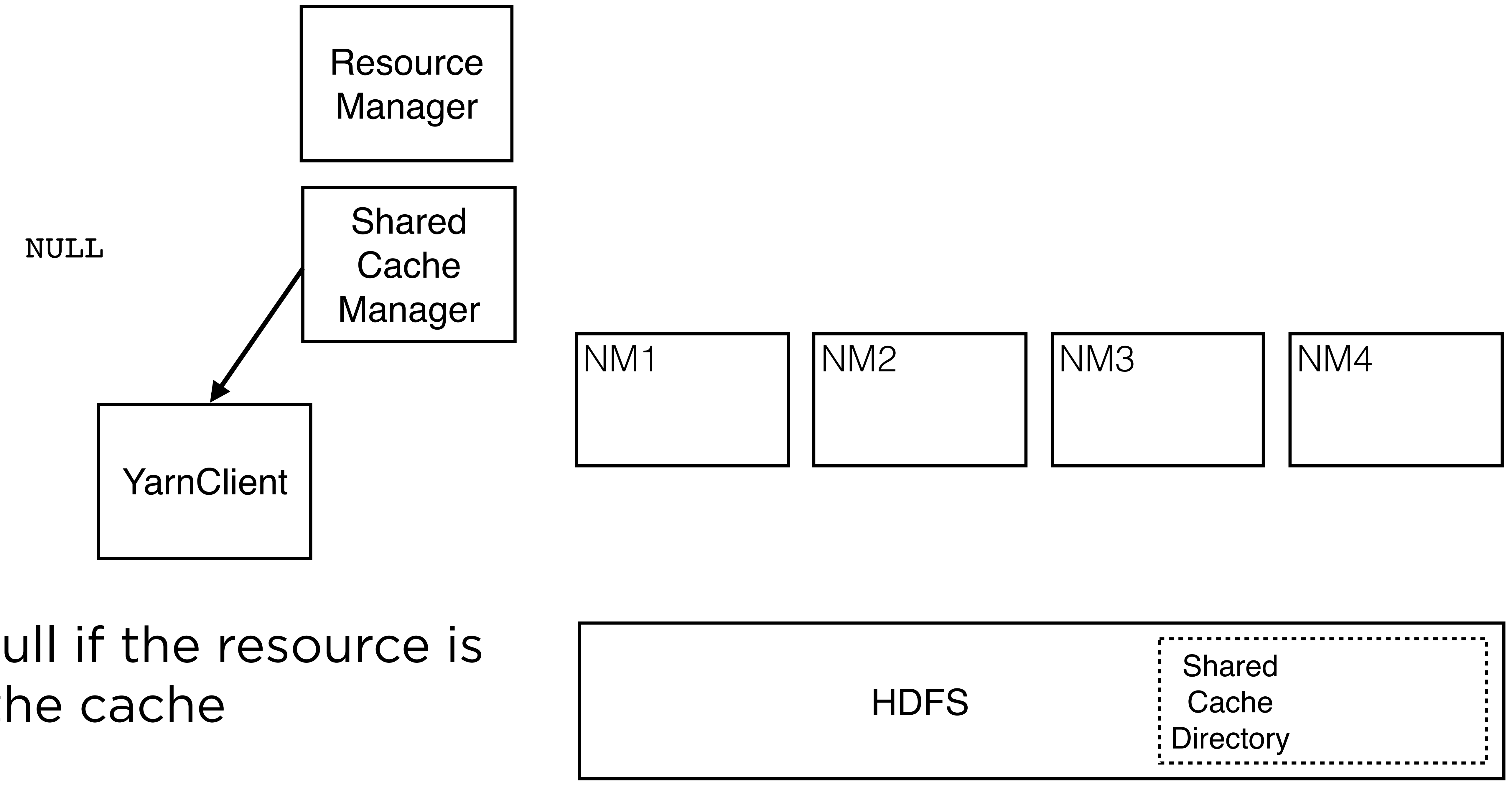
2. Notify SCM of the intent to use resource

ADDING TO THE CACHE



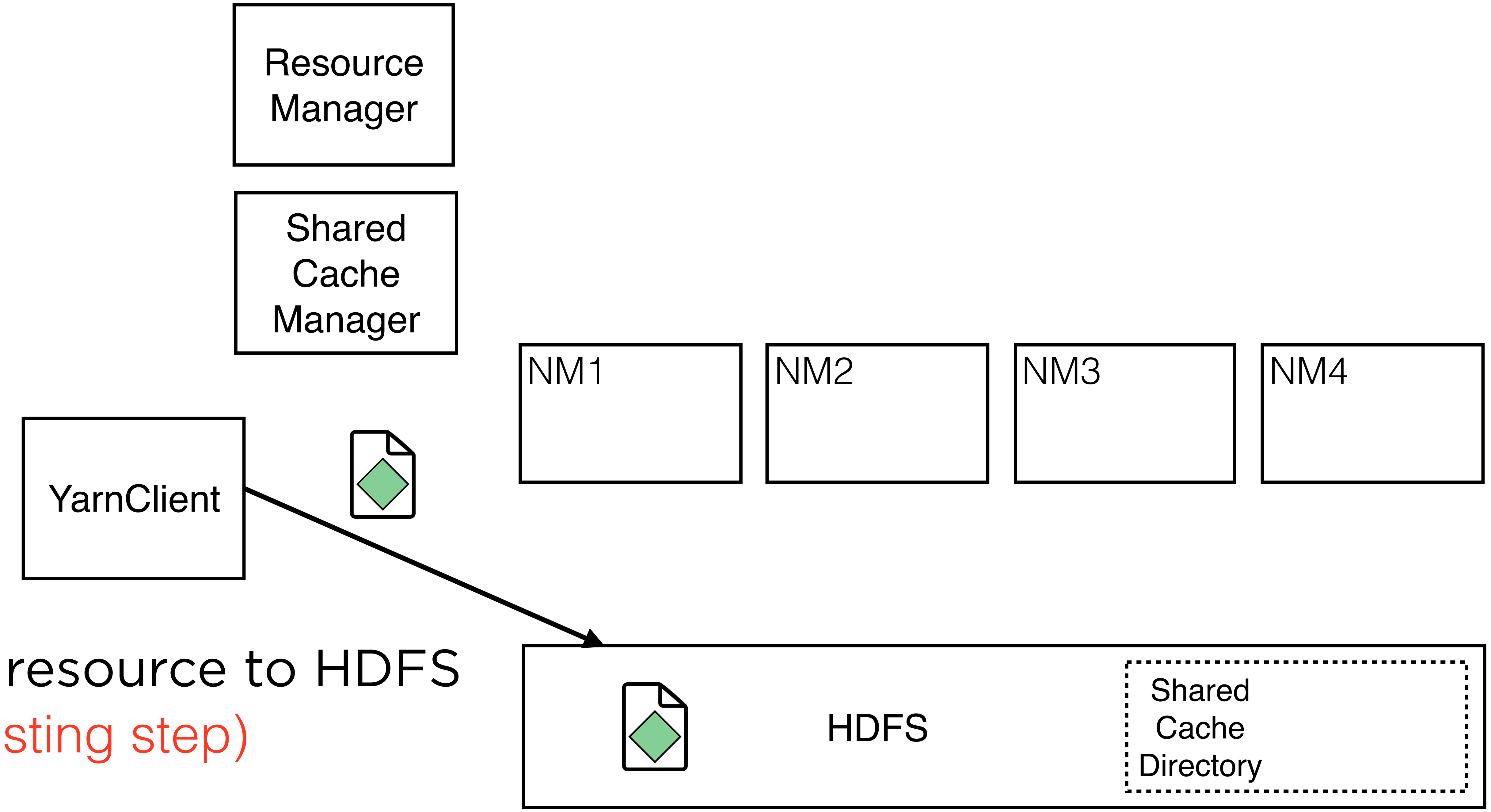
3. SCM responds back with the HDFS Path to the resource in the cache

ADDING TO THE CACHE



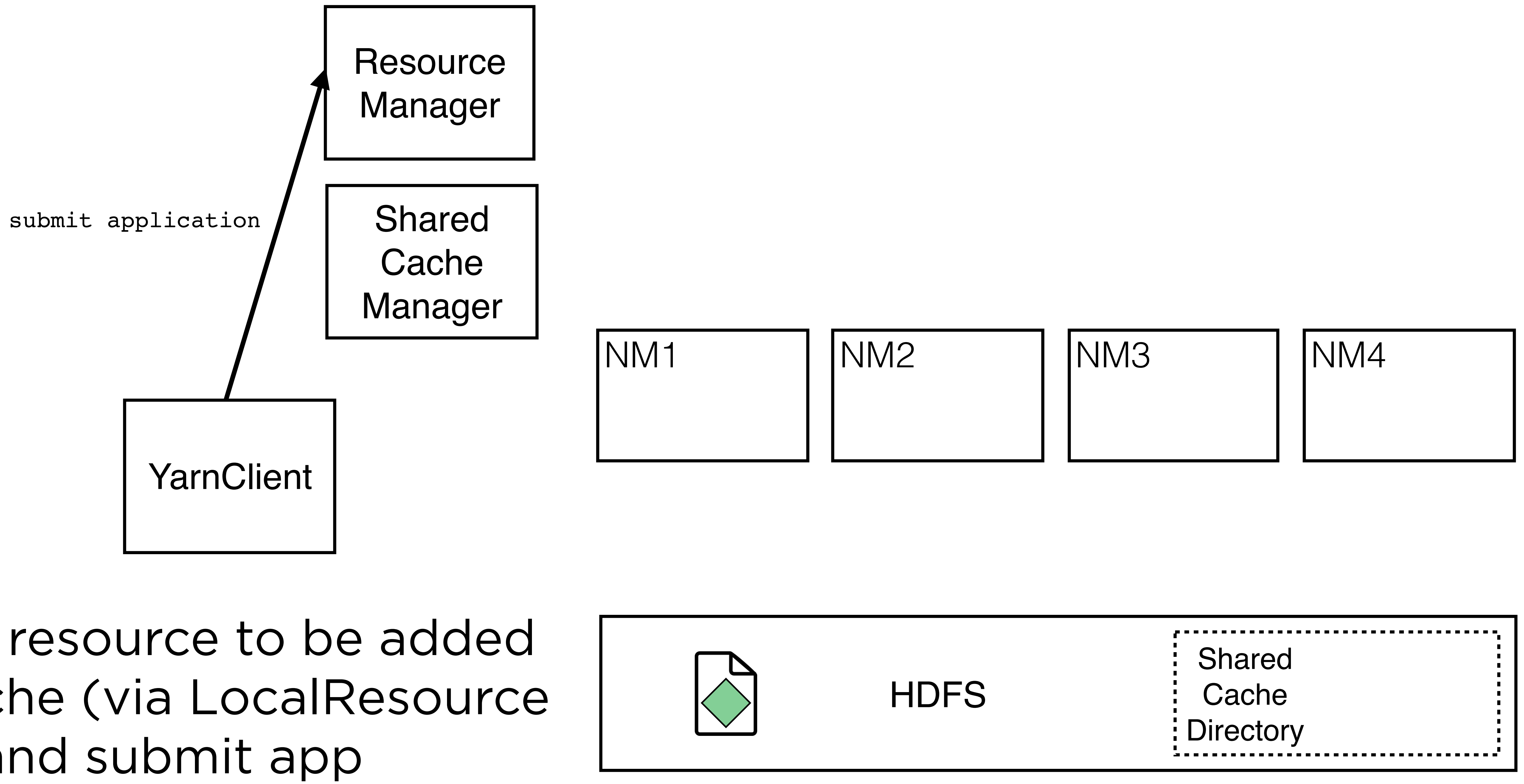
3. Or Null if the resource is not in the cache

ADDING TO THE CACHE



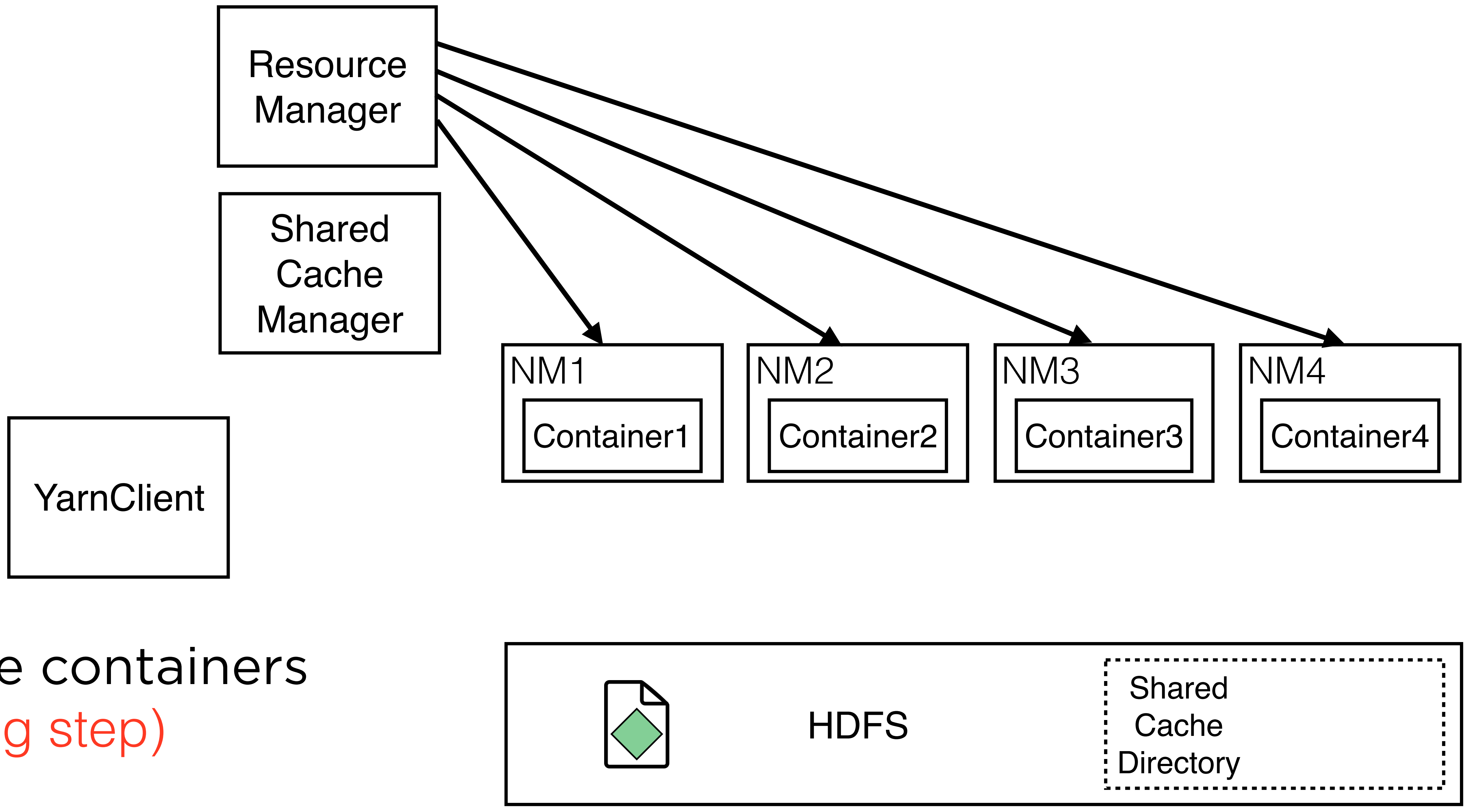
4. Upload resource to HDFS
(Existing step)

ADDING TO THE CACHE



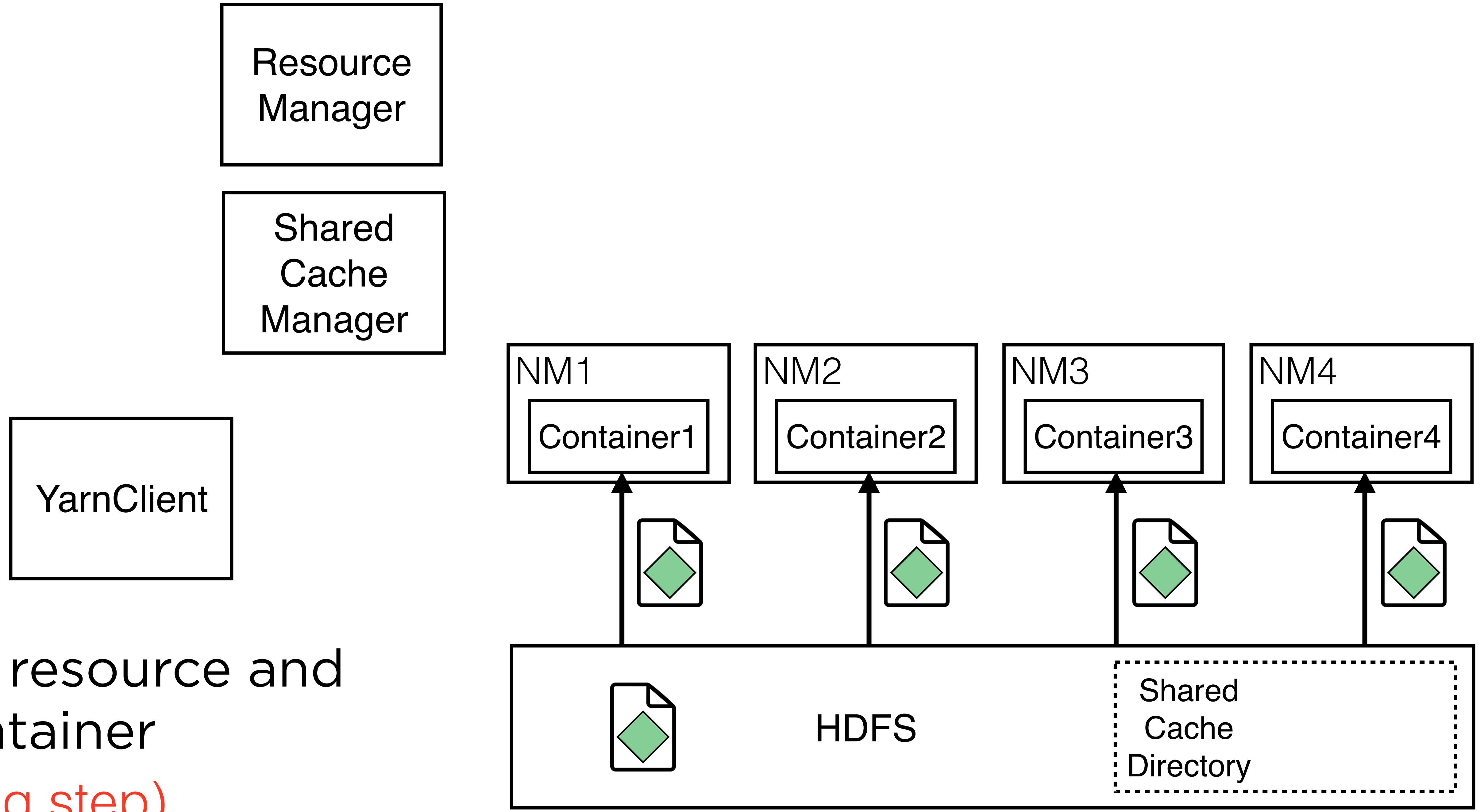
5. Set resource to be added to cache (via LocalResource API) and submit app

ADDING TO THE CACHE



6. Schedule containers
(Existing step)

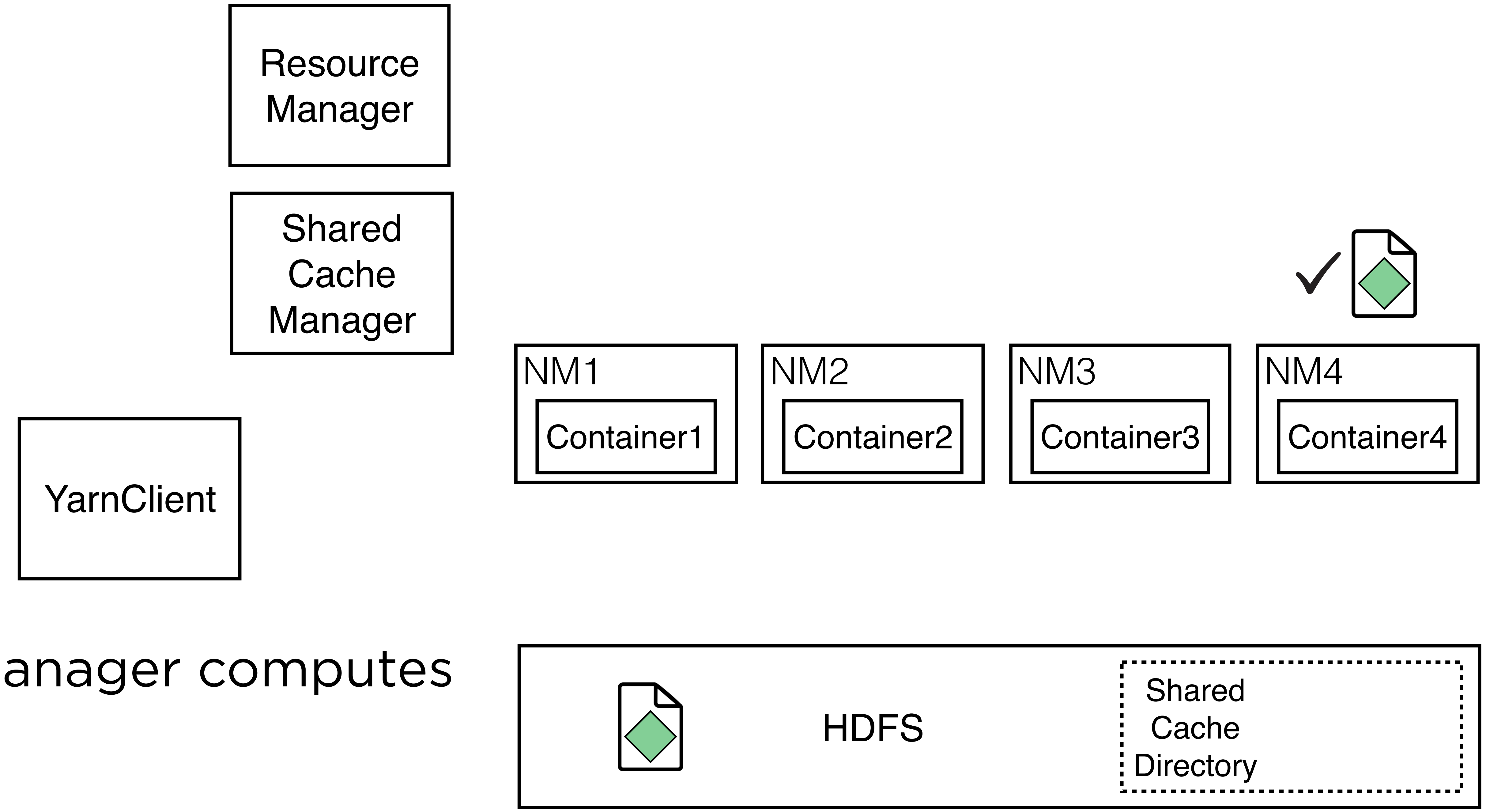
ADDING TO THE CACHE



7. Localize resource and launch container

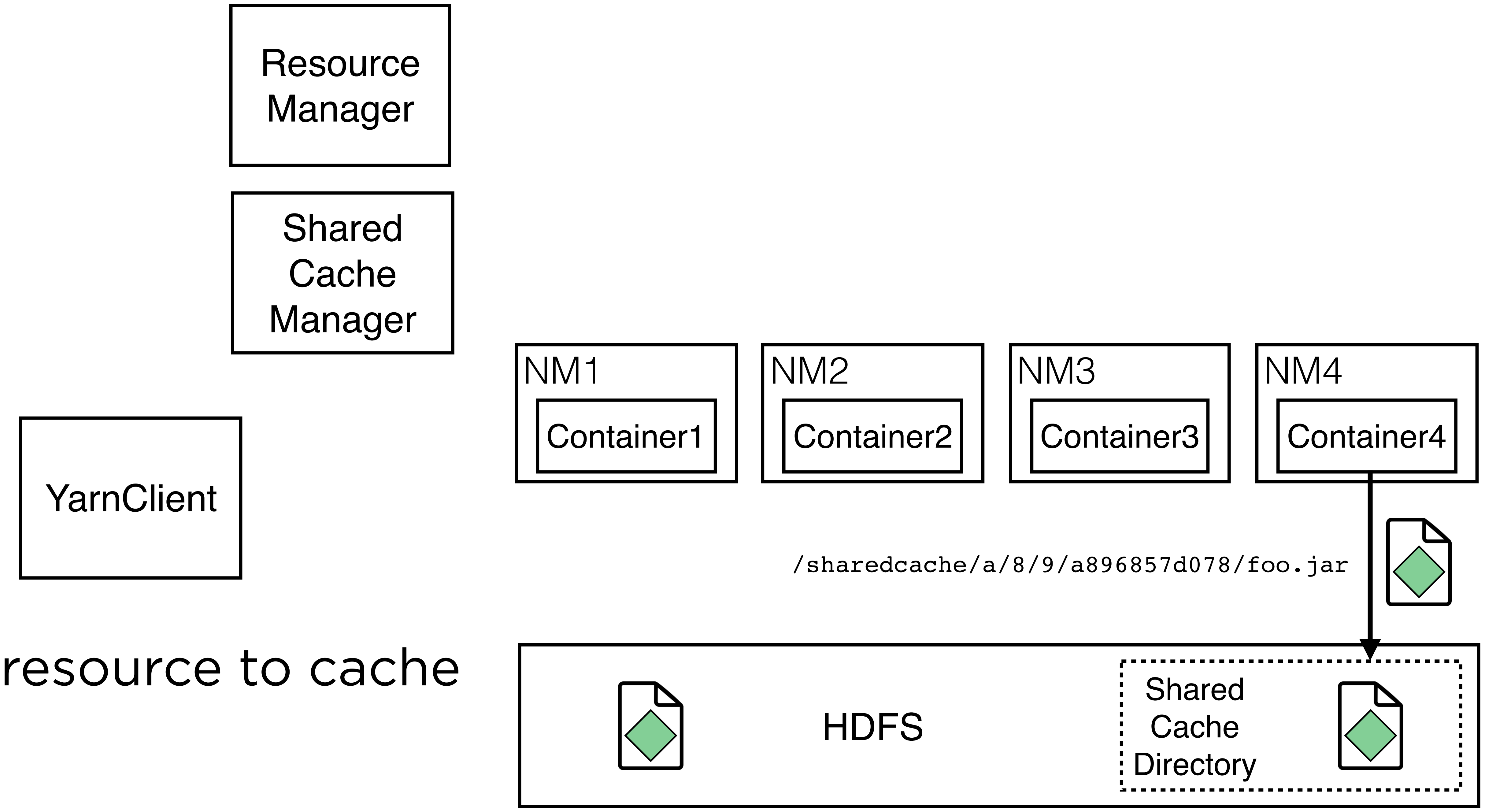
(Existing step)

ADDING TO THE CACHE



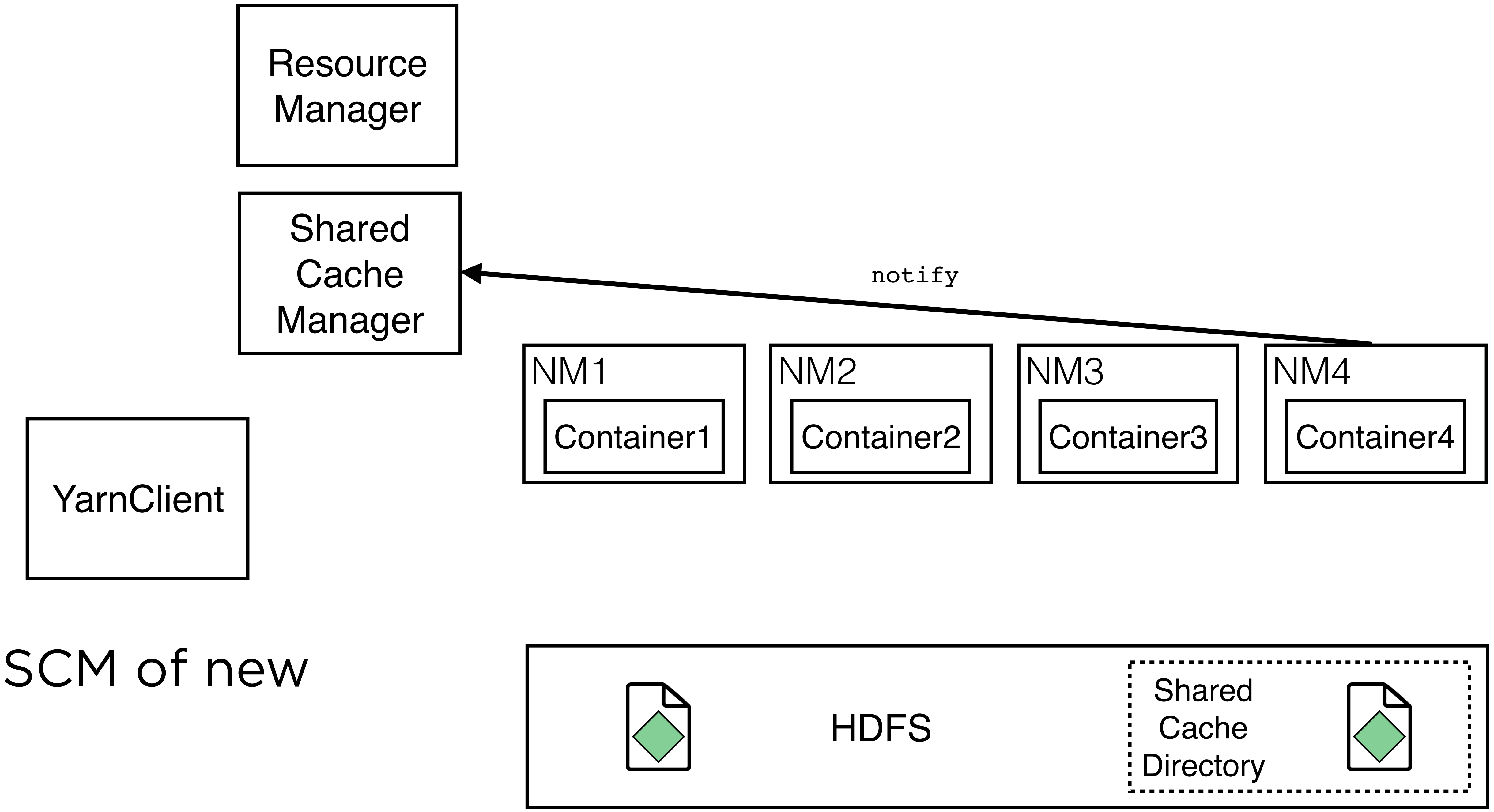
8. Node manager computes checksum

ADDING TO THE CACHE



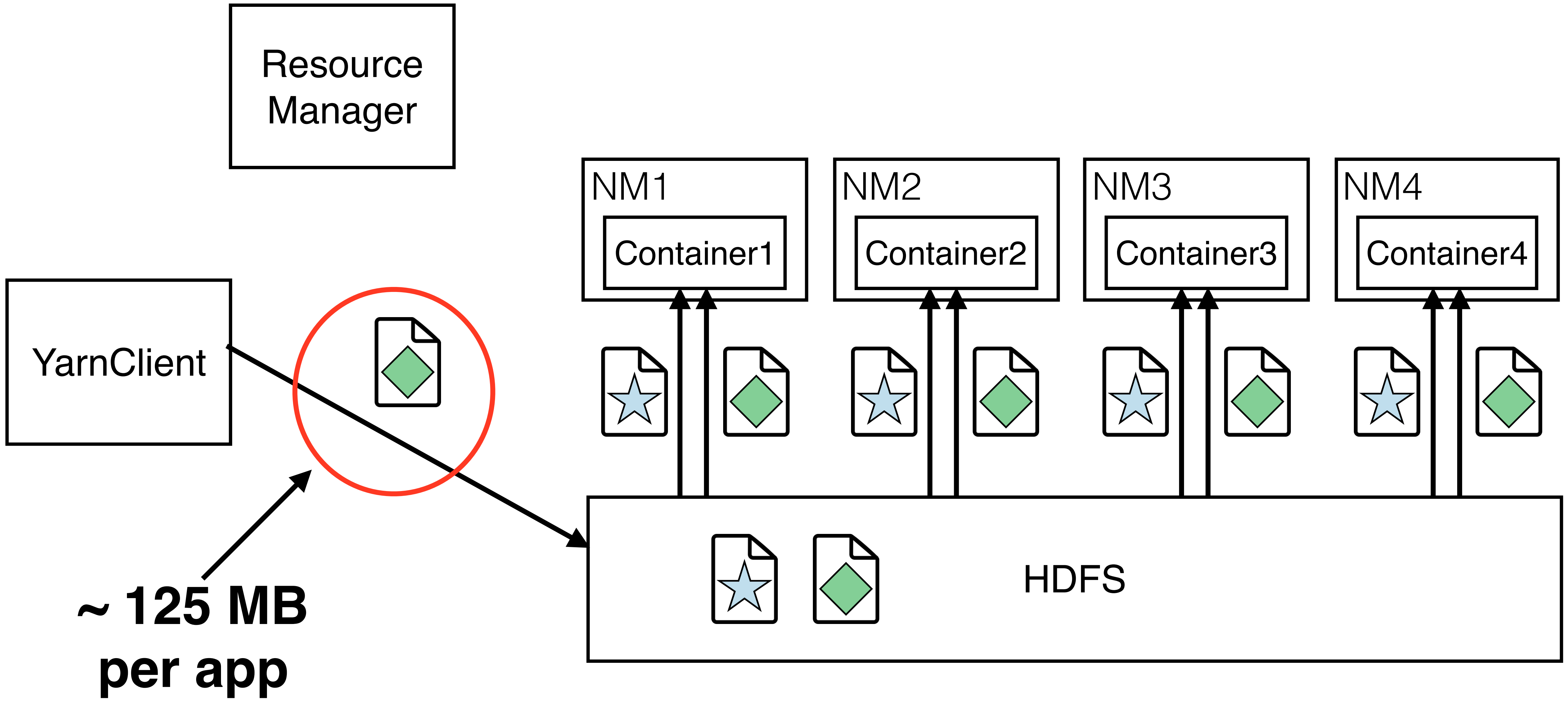
9. Upload resource to cache

ADDING TO THE CACHE

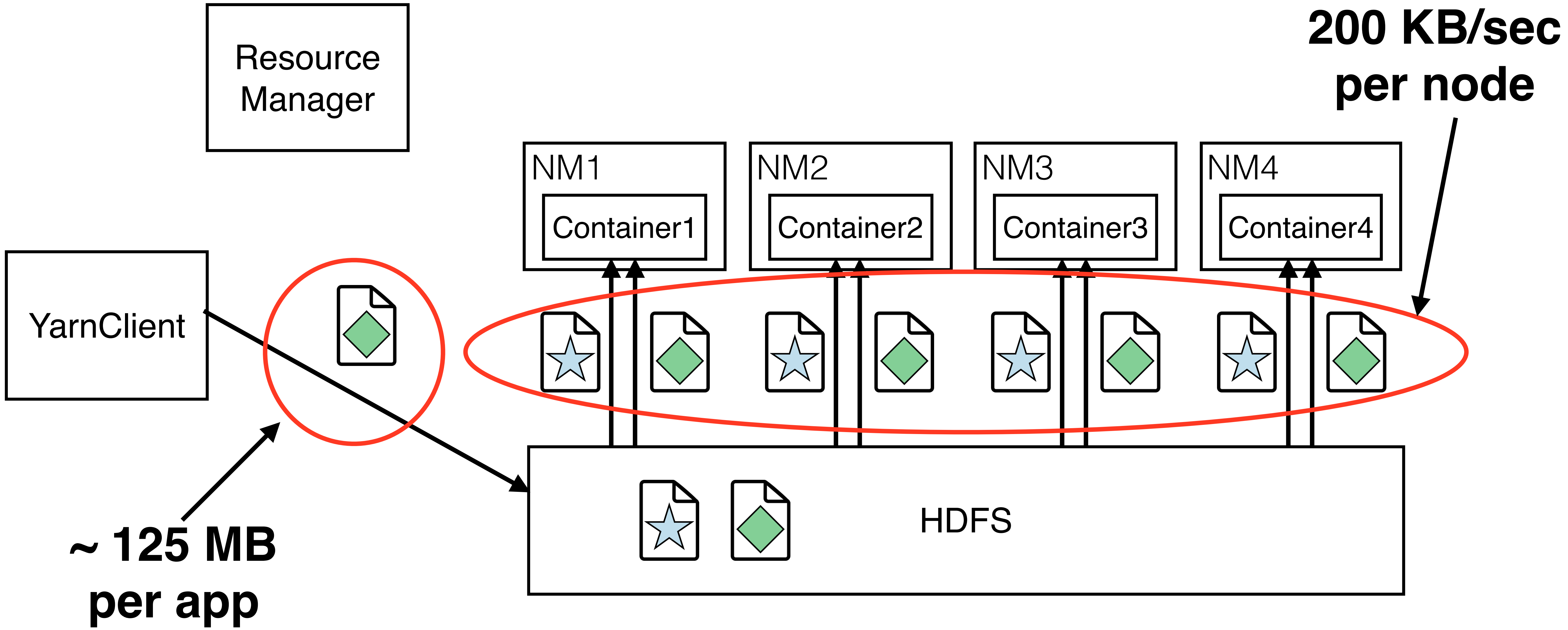


10. Notify SCM of new resource

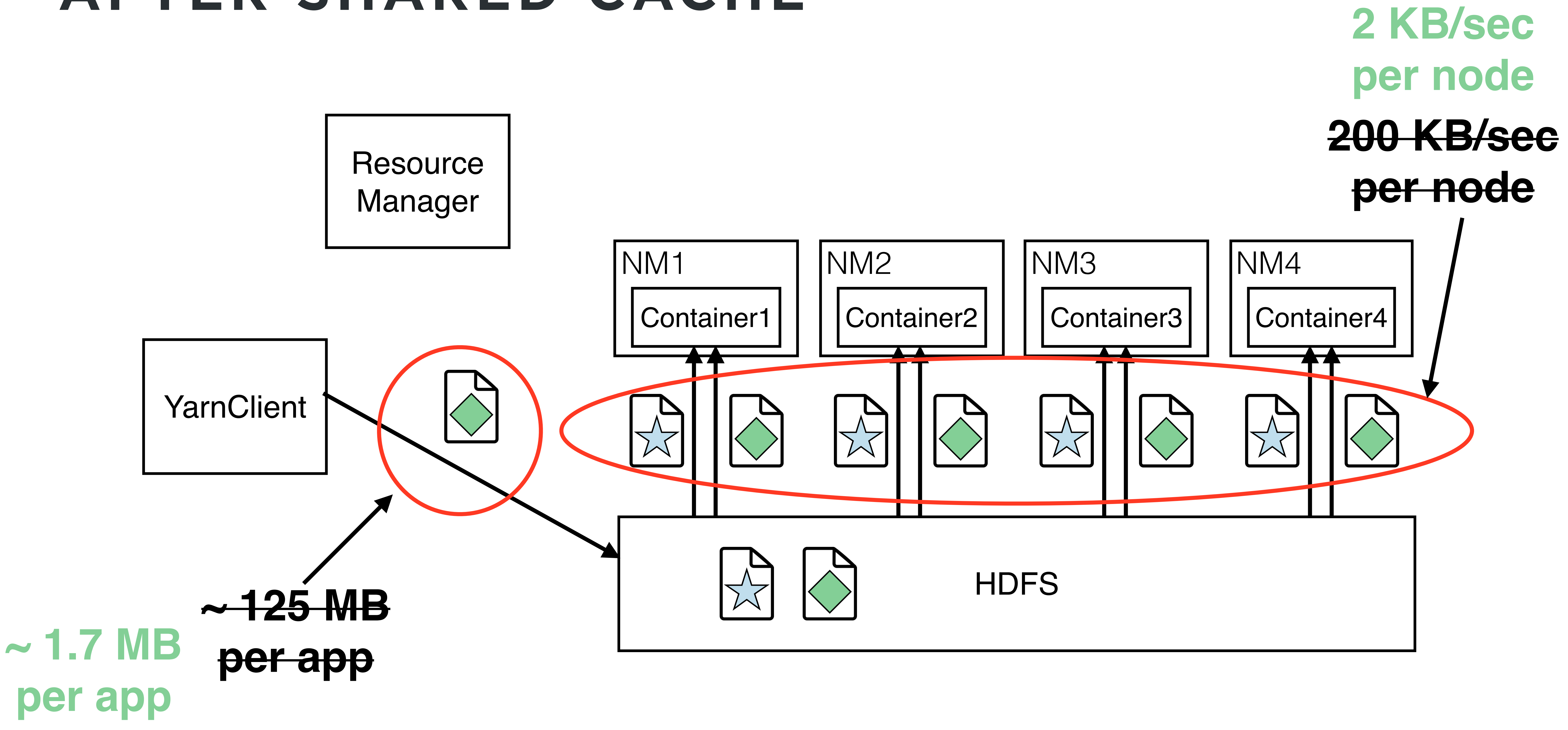
DOES IT WORK?


Resource Manager
YarnClient
NM1
Container1
NM2
Container2
NM3
Container3
NM4
Container4
HDFS
~ 125 MB per app

DOES IT WORK?


Resource Manager
YarnClient
NM1
Container1
NM2
Container2
NM3
Container3
NM4
Container4
200 KB/sec per node
HDFS
~ 125 MB per app

AFTER SHARED CACHE



BEFORE SHARED CACHE

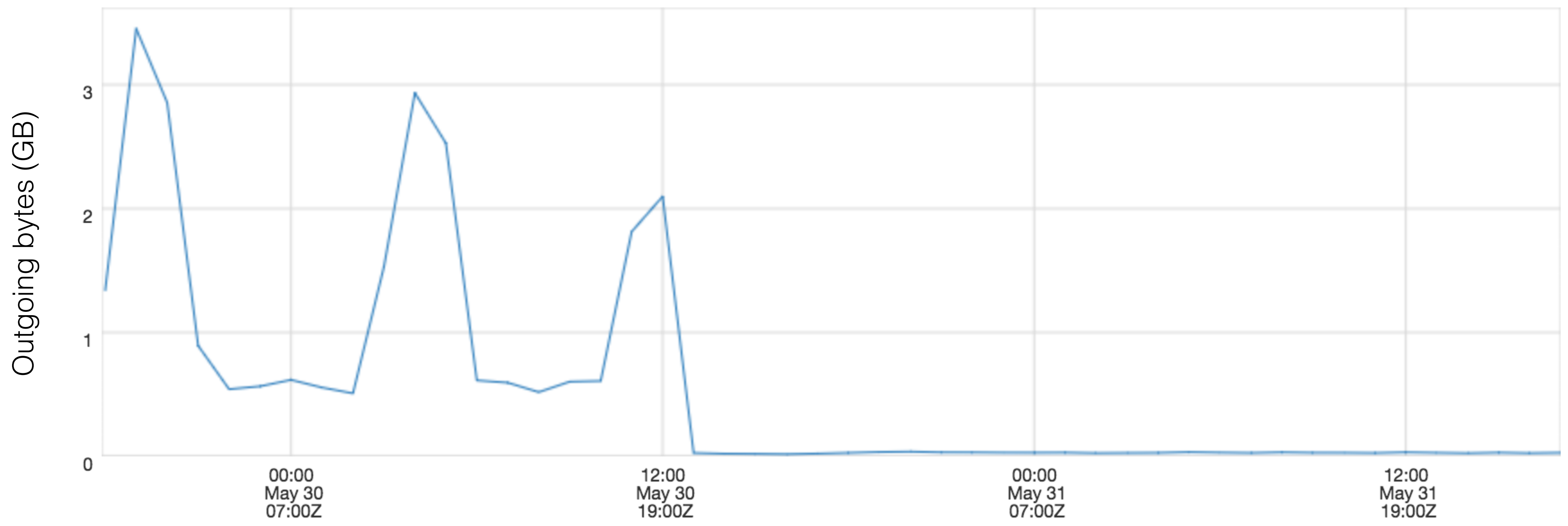
- Each application uploaded and localized (on average)
  - ~ 12 resources
  - ~ 125 MB total size
- Each container localized (on average)
  - ~ 6 resources
  - ~ 63 MB total size

AFTER SHARED CACHE

- Each application uploaded and localized (on average)
    - ~ 0.16 resources
    - ~ 1.7 MB total size
- Each container localized (on average)
    - ~ 0.08 resources
    - ~ 840 KB total size
- Saving localization bandwidth by 99%!

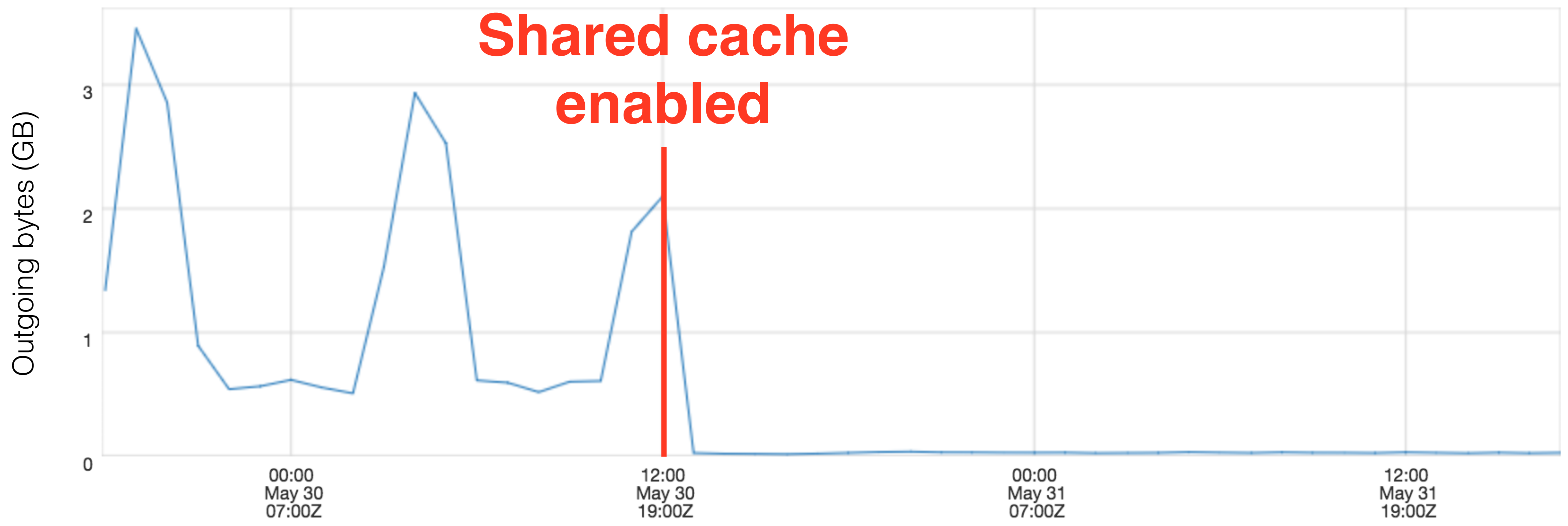
DOES IT WORK?

- Eliminates network bandwidth usage completely for a client that submits jobs constantly



DOES IT WORK?

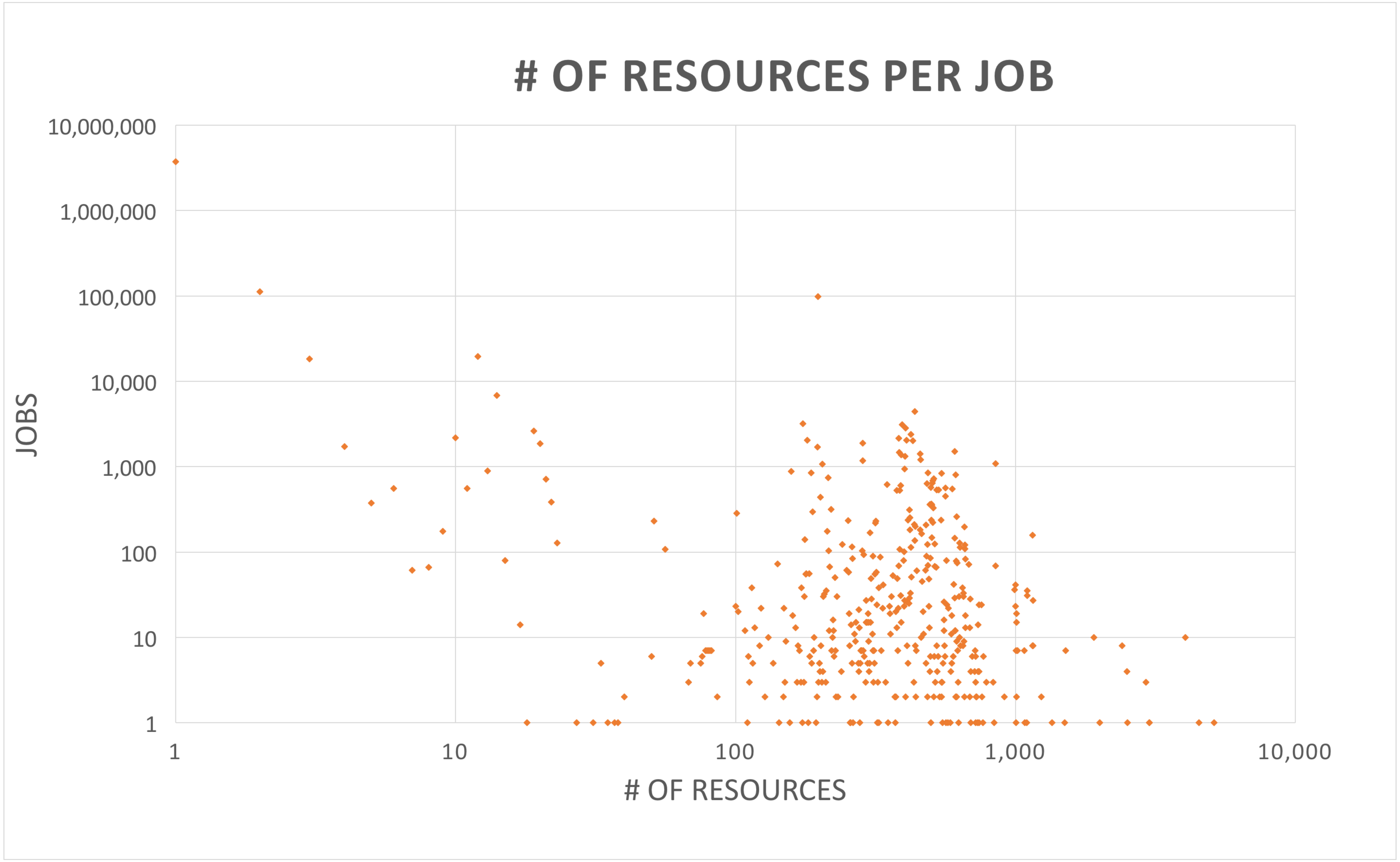
- Eliminates network bandwidth usage completely for a client that submits jobs constantly



ANTI-PATTERNS THAT CAUSE CHURN

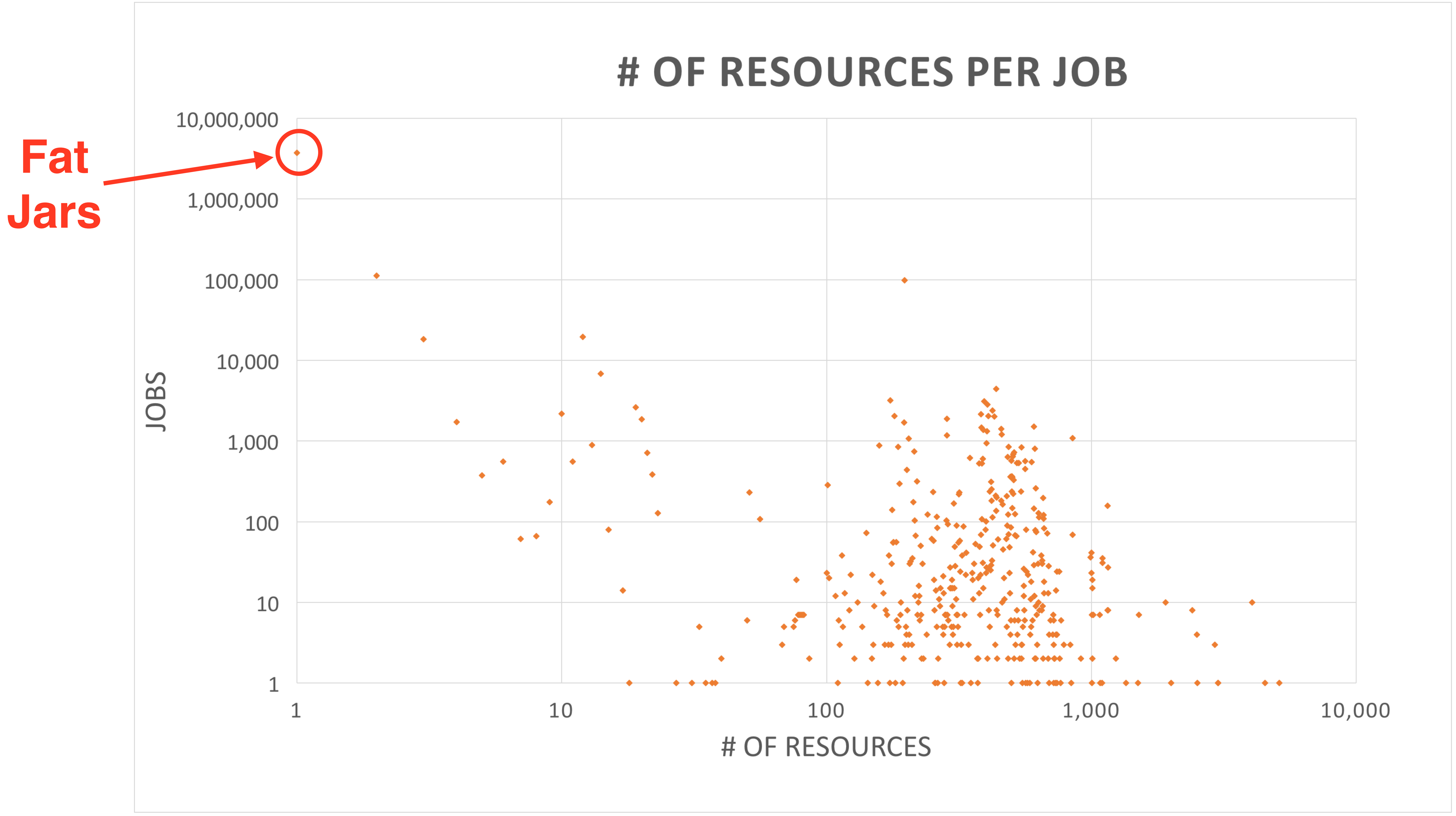
- Resource churn
  - Build process may introduce artificial change
  - Fat jars

FAT JARS



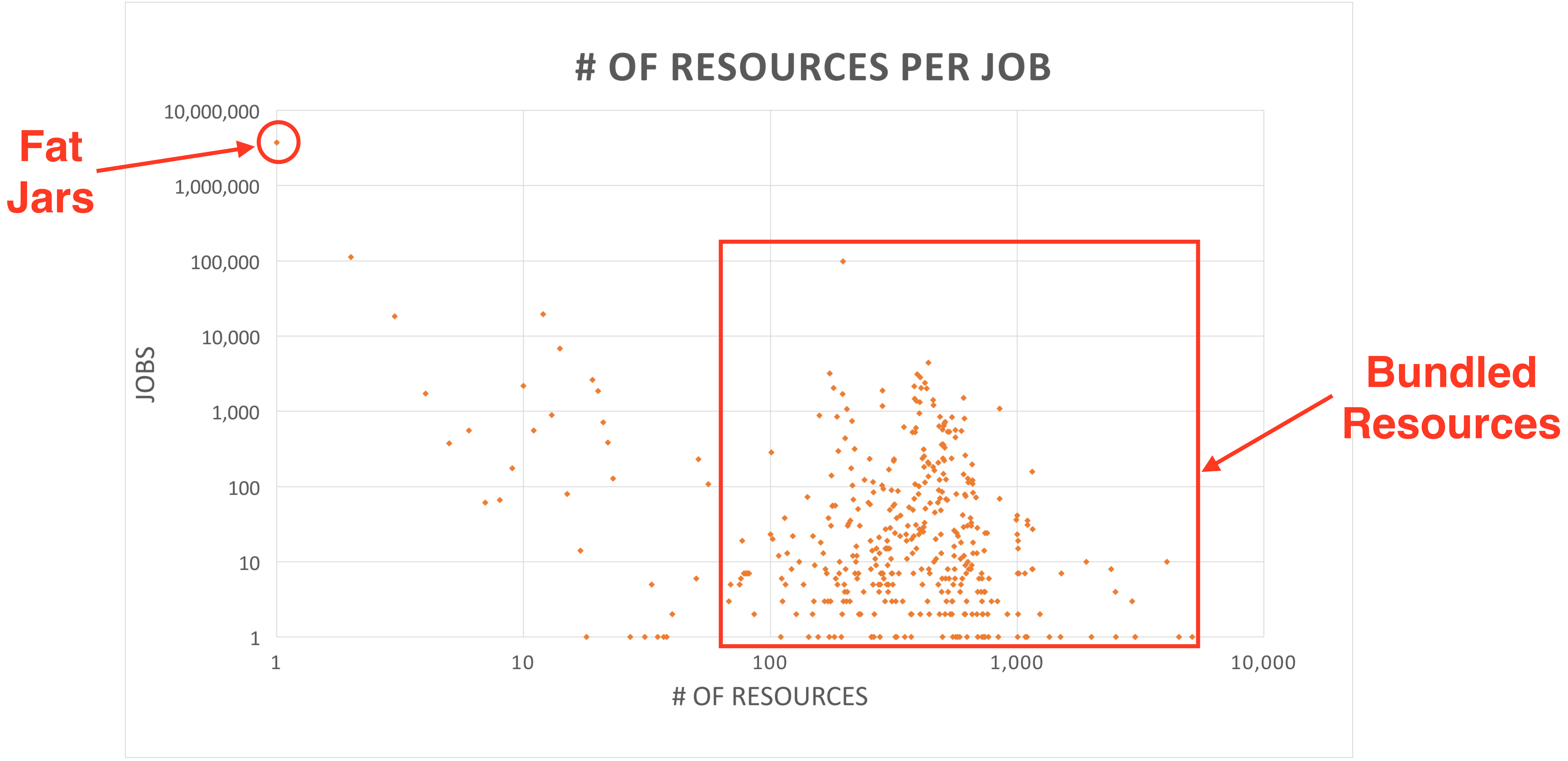
# OF RESOURCES PER JOB
10,000,000
1,000,000
100,000
10,000
1,000
100
10
1
JOBS
1
10
100
1,000
10,000
# OF RESOURCES

FAT JARS



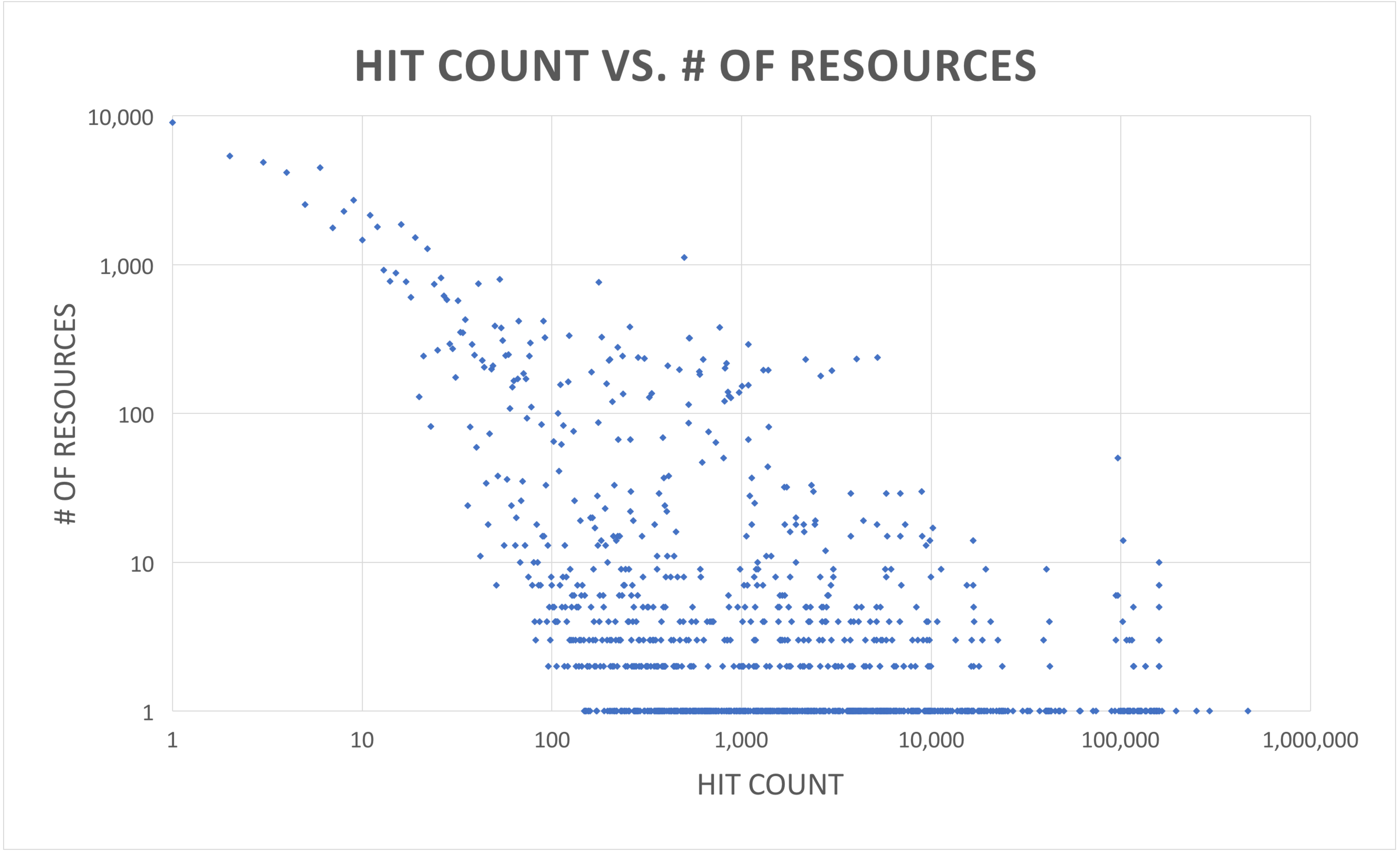
Fat Jars
# OF RESOURCES PER JOB
JOBS
10,000,000
1,000,000
100,000
10,000
1,000
100
10
1
1
10
100
1,000
10,000
# OF RESOURCES

FAT JARS

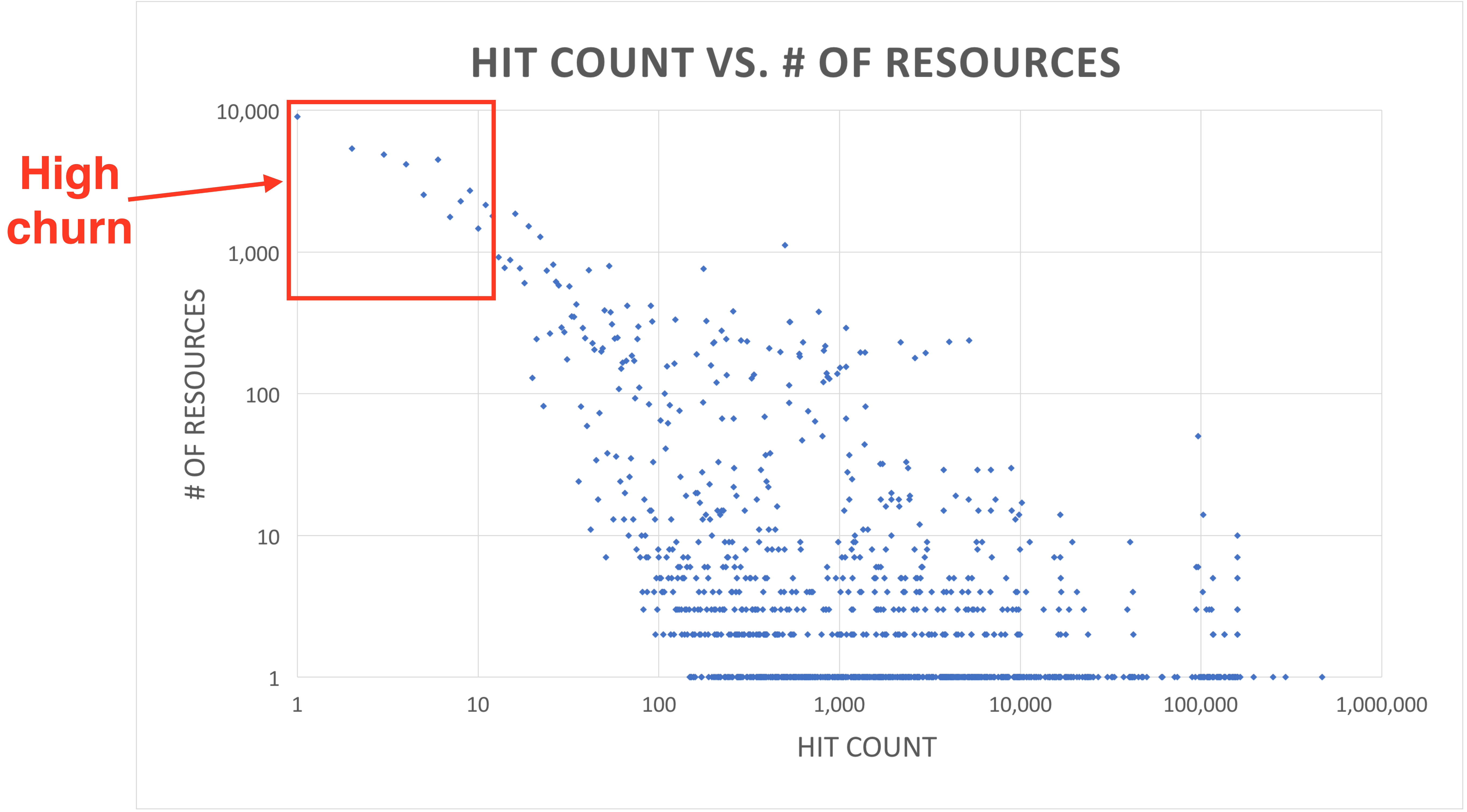


# OF RESOURCES PER JOB
10,000,000
1,000,000
100,000
10,000
1,000
100
10
1
JOBS
1
10
100
1,000
10,000
# OF RESOURCES
Fat Jars
Bundled Resources

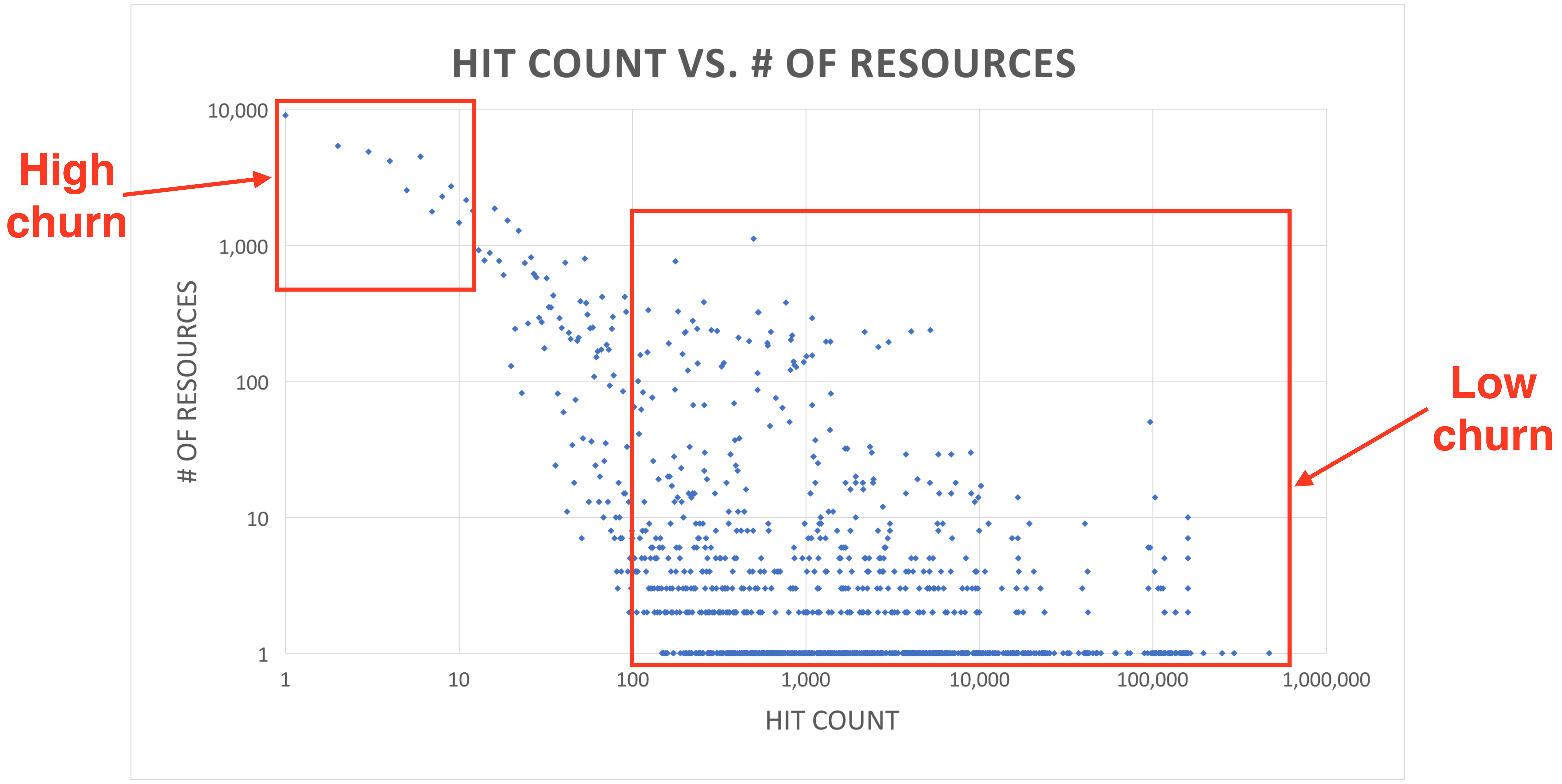
RESOURCE CHURN



RESOURCE CHURN



RESOURCE CHURN



ANTI-PATTERNS THAT CAUSE CHURN

- Local cache churn
  - Local cache size too small for working set
  - Competing use of public cache (e.g. pig jar cache, cascading intermediate files)

ADMIN TIPS

- Avoid “fat jars”
- Make your jars repeatable
- Set the local cache size appropriately
  - `yarn.nodemanager.localizer.cache.target-size-mb=65536`
- Increase public localizer thread pool size
  - `yarn.nodemanager.localizer.fetch.thread-count=12`
- Adjust the cleaner frequency to your usage pattern
  - `yarn.sharedcache.cleaner.period.minutes (default: 1 day)`

DEV TIPS

- YARN Developer
  - Invoke `use` API to claim resources
  - Use the LocalResource API to add resources
- MapReduce Developer
  - Set `mapreduce.job.sharedcache.mode`
    - `jobjar,libjar,files,archives`

ACKNOWLEDGEMENTS

- Code: Chris Trezzo, Sangjin Lee, Ming Ma
- Shepherd: Karthik Kambatla
- Design Review
  - Karthik Kambatla
  - Vinod Kumar Vavilapalli
  - Jason Lowe
  - Joep Rottinghuis

Q&A - THANKS!

- Chris Trezzo - @ctrezzo