Dynamically Resizable Logical Record Cache

2023 TPF Users Group Conference
April 24-26, Dallas, TX
Application Development

Claire Durant

Background: Logical record caches

- Reside in system heap; provide fast access to frequently-used data
- Stored in 64-bit or recoverable system heap
- Created by an API call or from a cache definition
- Contain a fixed number of entries
- Optionally configure a castout time, or expiration time
  - Also allow unique castout times for individual entries

Background: Cache definitions

- “Blueprint” for a cache
- Use ZCACH commands to set attributes & create the cache
- Change cache attributes without making application changes on newCache API calls

Resizing logical record caches

As-is: Resizing caches

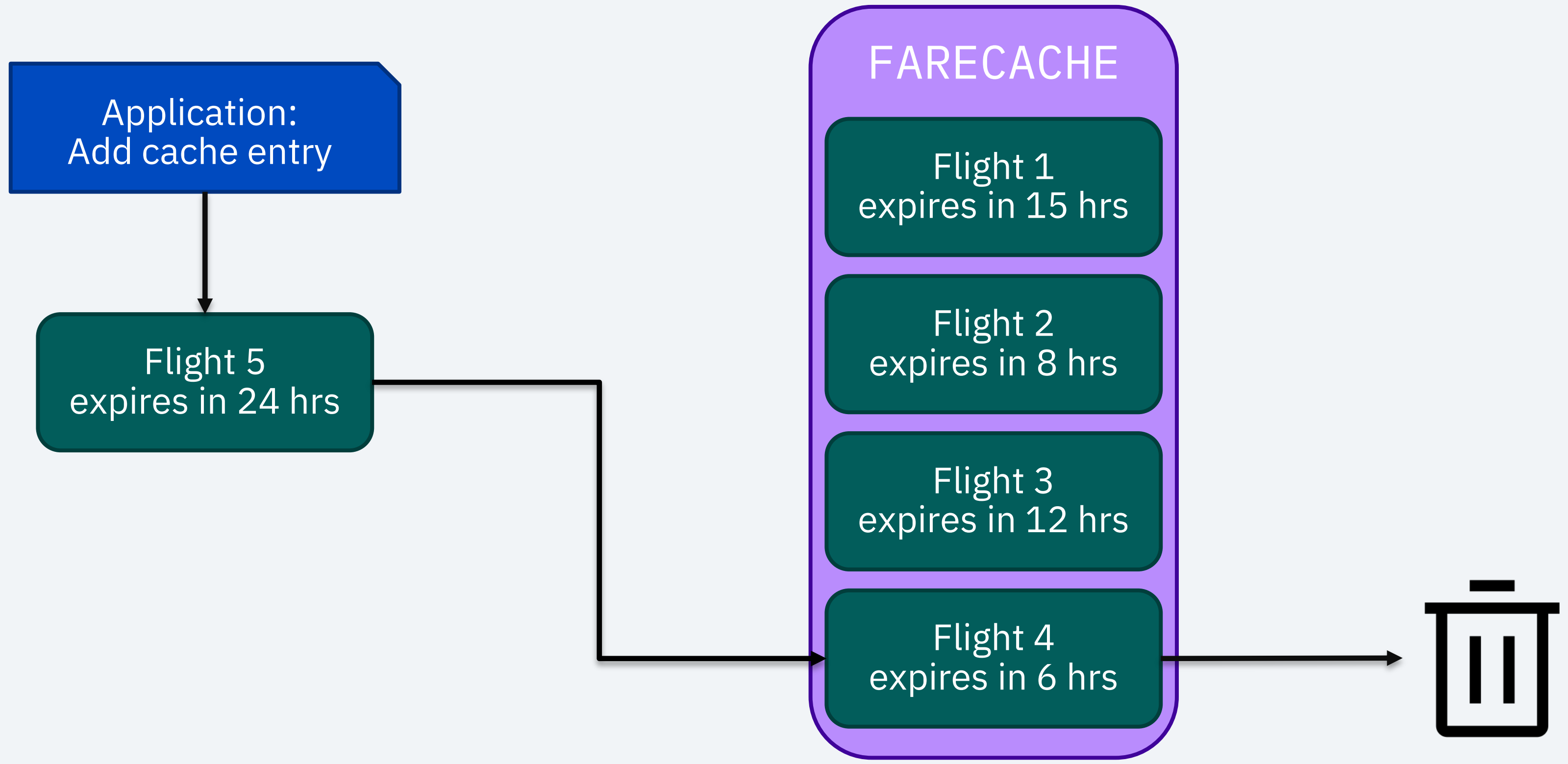
If a cache isn’t large enough to hold the entries your workload requires, then applications will spend valuable system resources to re-retrieve or re-create data that would normally be in cache.

The size of the cache needs to be increased to match the needs of the workload.

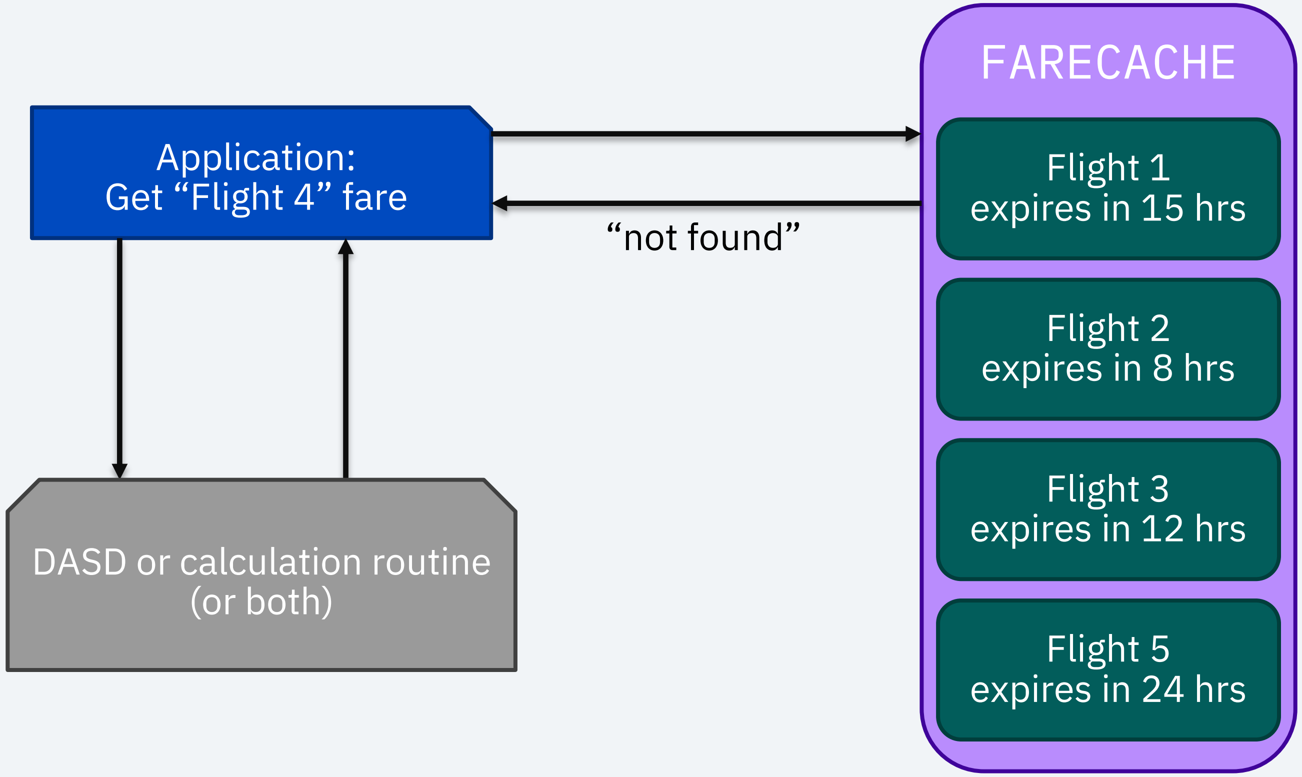
The only way to increase the cache's size is to delete & re-create the cache.

Repopulating large, empty caches can have significant impacts on resource utilization for an extended period of time. In addition, transactional traffic that expects data to be in cache can be impacted.

Oversaturating a cache



The consequences…



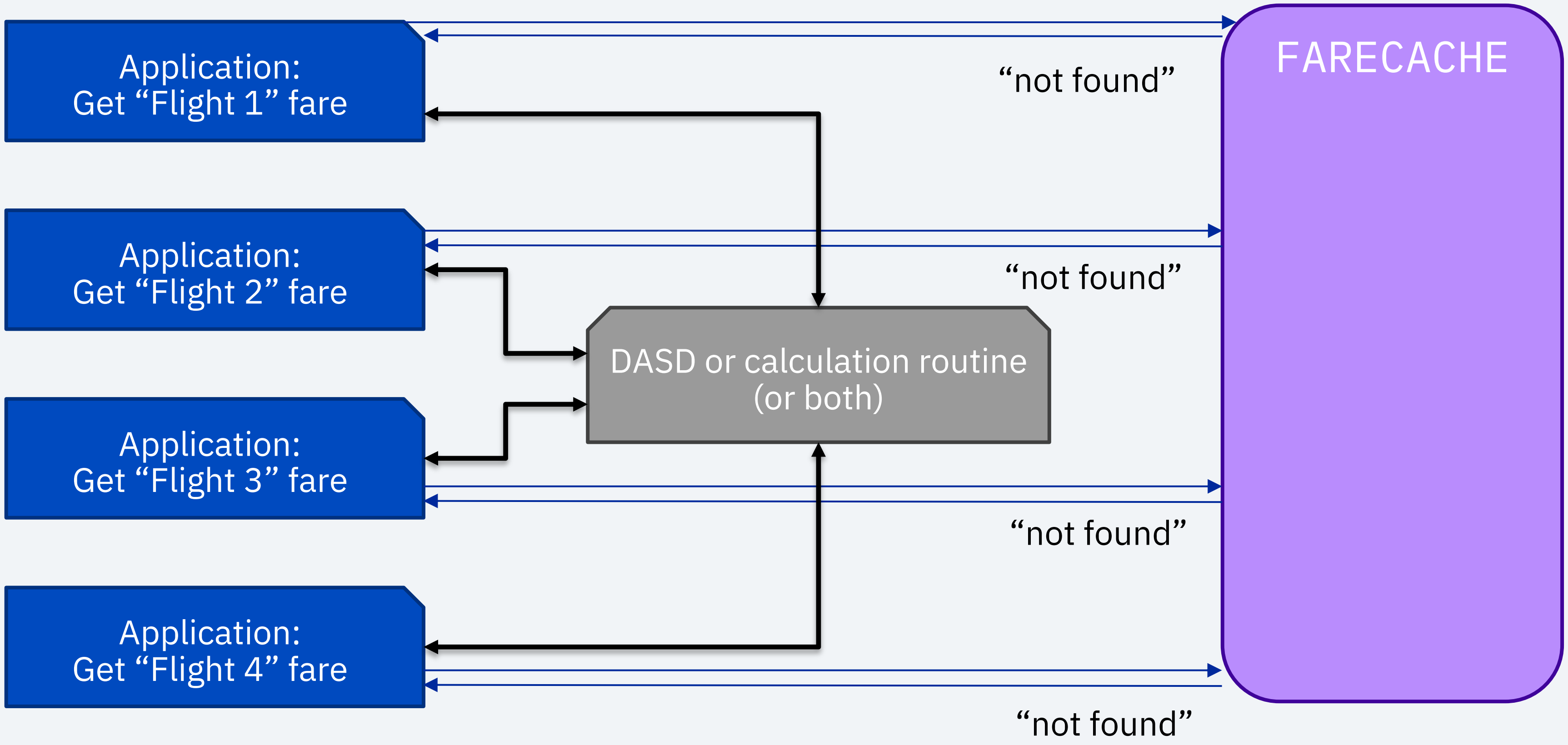
The cycle of an oversaturated cache

- Find entry **X** in cache: not found
- Get entry **X** data from DASD or calculations (or both)
- Add entry **X** to cache (casting out existing entry **Y**)
- Find entry **Y** in cache: not found
- Get entry **Y** data from DASD or calculations (or both)
- Add entry **Y** to cache (casting out entry **Z**)...

Increasing cache size, the old way

- Increase the number of entries in cache definition
- Delete the cache
- Re-create the cache
  - Sorry, all your cached data is gone

Result of losing cached data



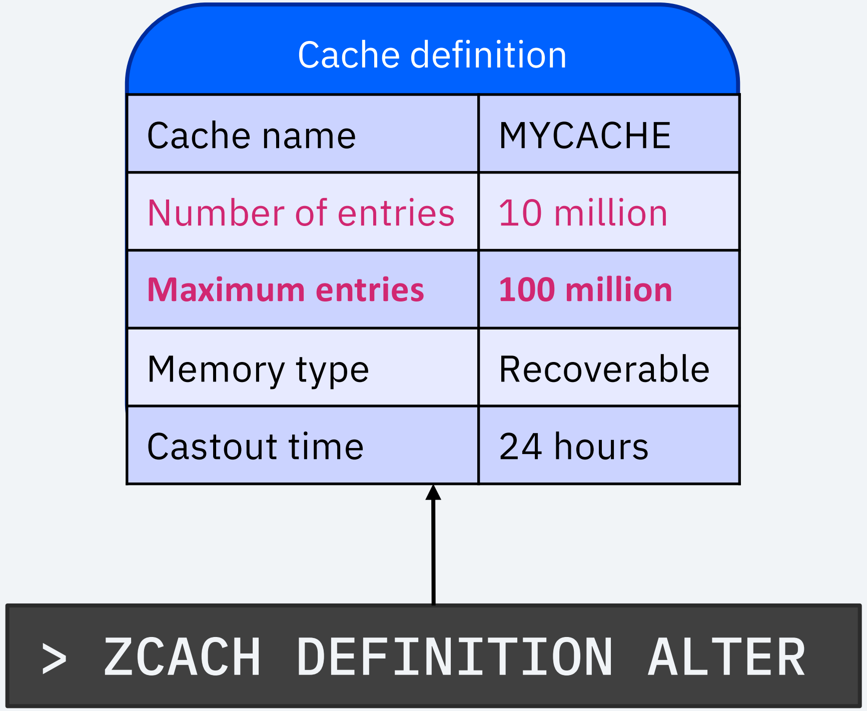
To-be: Resizing caches

If a cache isn’t large enough to hold the entries your workload requires, then applications will spend valuable system resources to re-retrieve or re-create data that would normally be in cache.

The size of the cache needs to be increased to match the needs of the workload.

If you can resize a cache while it’s being used, you can avoid the impact of repopulating a freshly created cache to both resource utilization and transactional traffic.

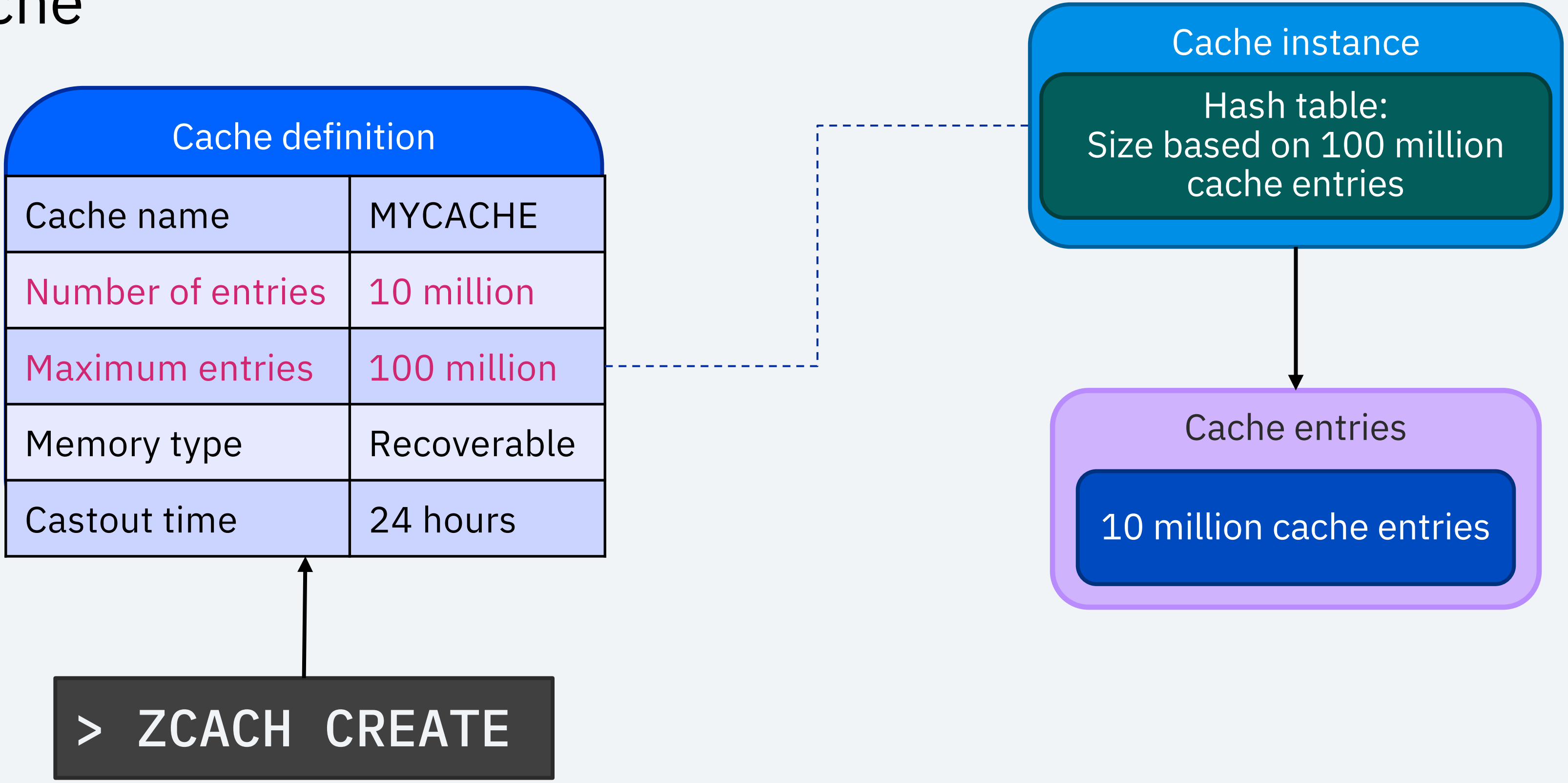
Resizing a cache: Set maximum number of entries



Cache definition	
Cache name	MYCACHE
Number of entries	10 million
Maximum entries	**100 million**
Memory type	Recoverable
Castout time	24 hours

```
> ZCACH DEFINITION ALTER
```

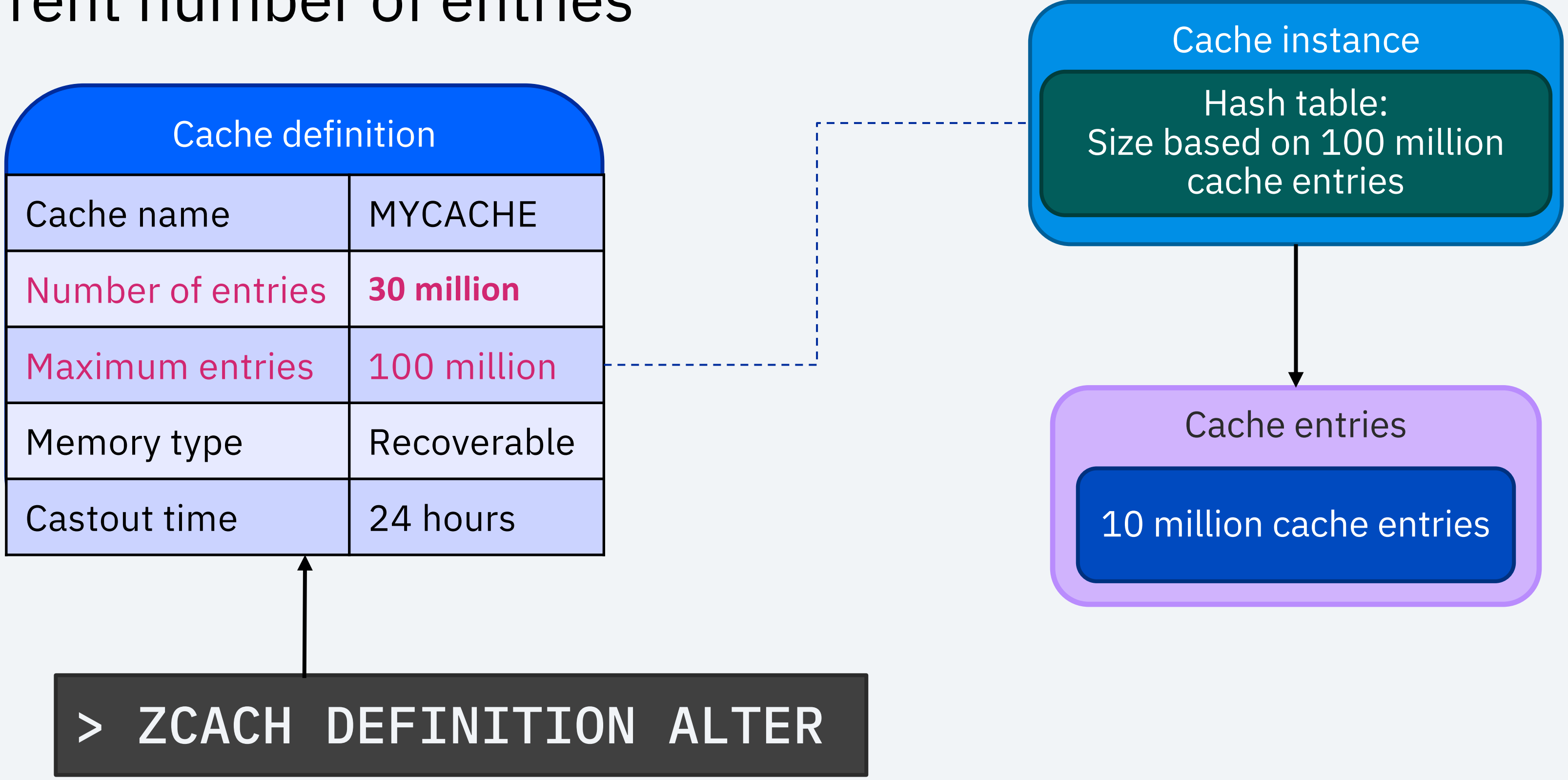
Resizing a cache: Create the cache



Cache name	MYCACHE
Number of entries	10 million
Maximum entries	100 million
Memory type	Recoverable
Castout time	24 hours

```
> ZCACH CREATE
```

Resizing a cache: Increase current number of entries

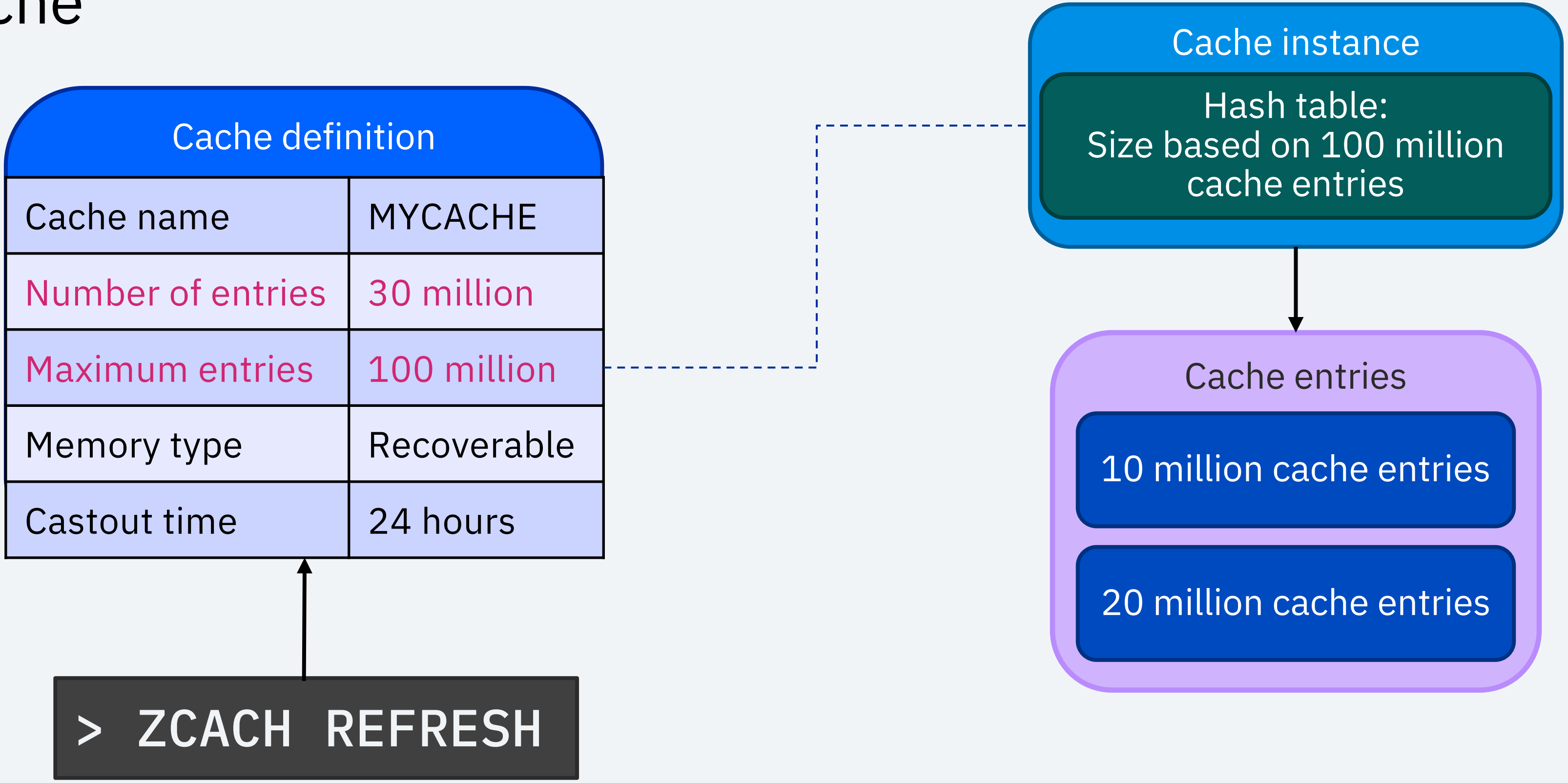


Cache definition

Cache name	MYCACHE
Number of entries	**30 million**
Maximum entries	100 million
Memory type	Recoverable
Castout time	24 hours

```
> ZCACH DEFINITION ALTER
```

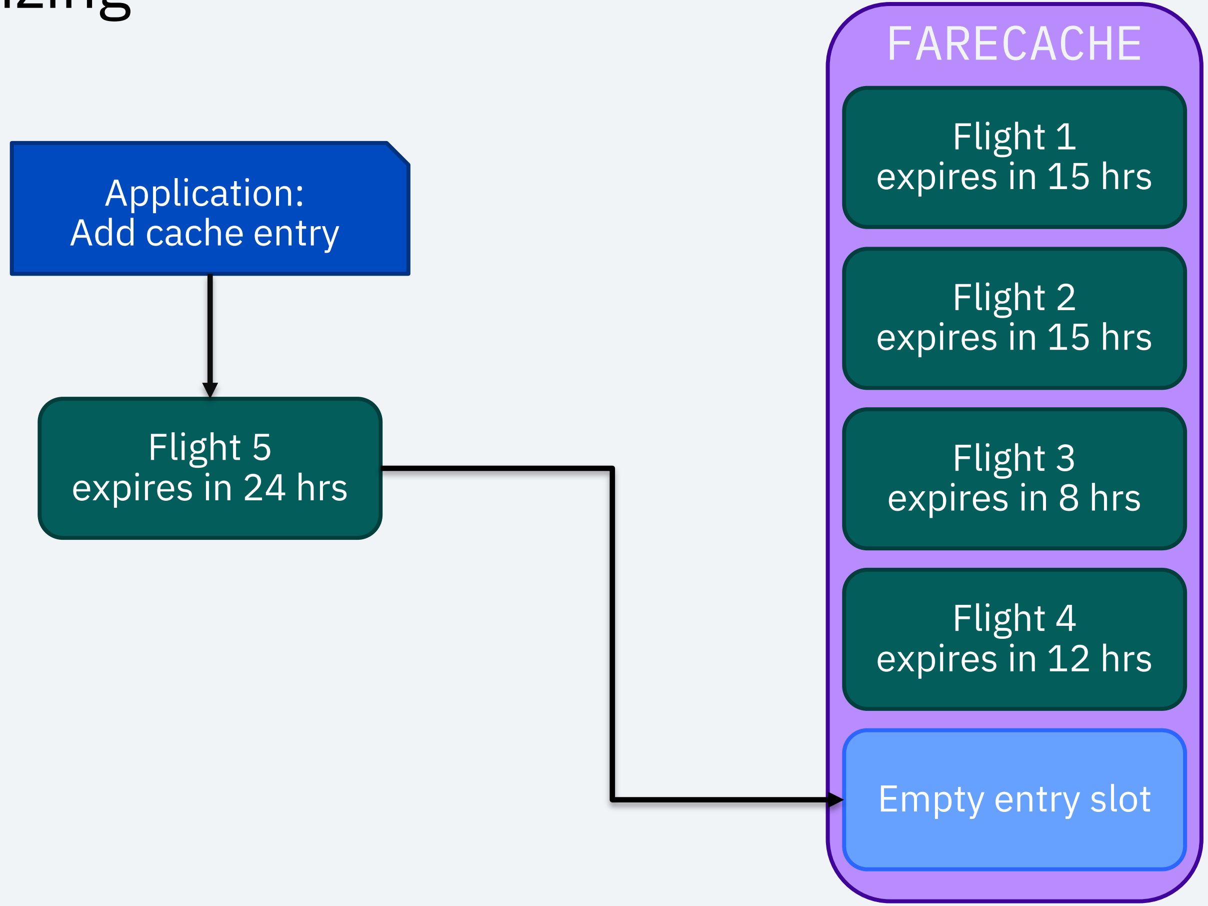
Resizing a cache: Refresh the cache



Cache definition	
Cache name	MYCACHE
Number of entries	30 million
Maximum entries	100 million
Memory type	Recoverable
Castout time	24 hours

```
> ZCACH REFRESH
```

After resizing



Details: resizable caches

- To resize a cache, it must be managed by a cache definition
- Cache must be processor unique with an entry size less than or equal to 4096 bytes
- Number of entries can’t be dynamically decreased
- Maximum number of entries can only be set when the cache is created
  - Set it to a value that supports future growth!

Scalability improvements

As-is

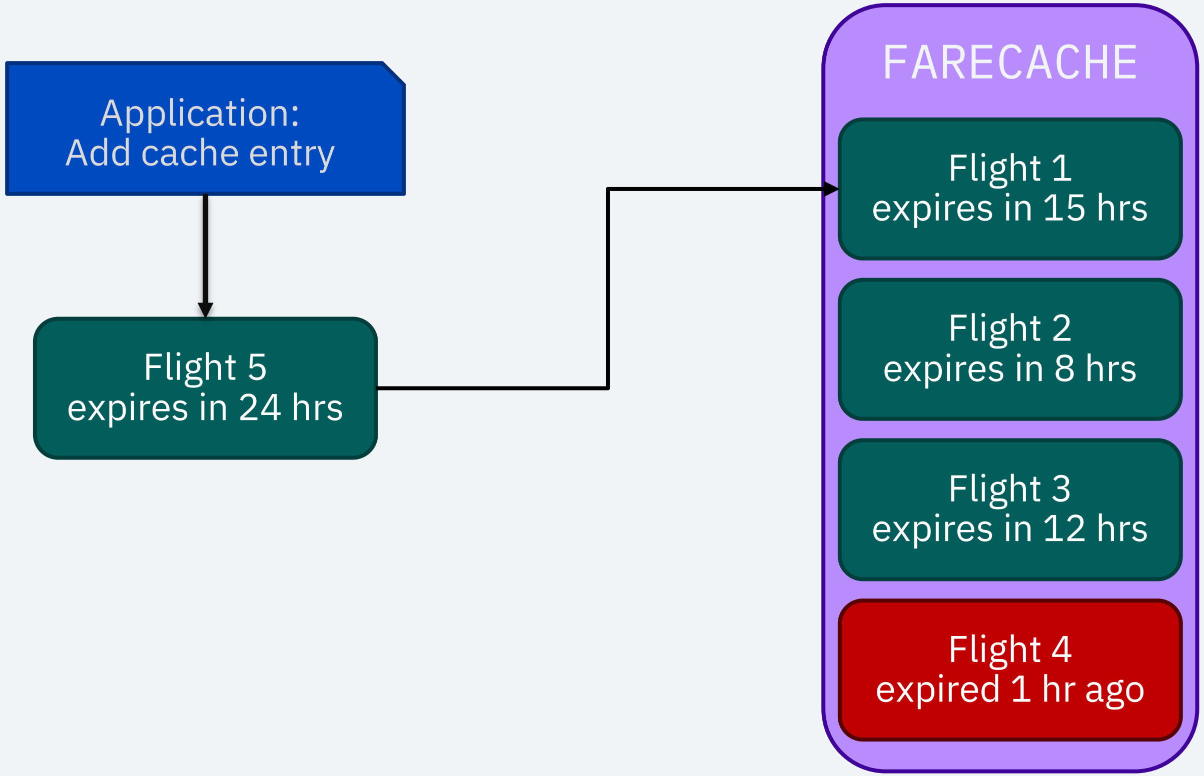
When an application adds an entry to a full cache, logical record cache support enters “castout processing,” searching for an expired entry to cast out and thereby make room for the new entry.

As a cache gets larger, the searches associated with castout processing can get more expensive. In addition, searches for expired entries are performed using application ECB time, potentially adding to an application’s cost per transaction.

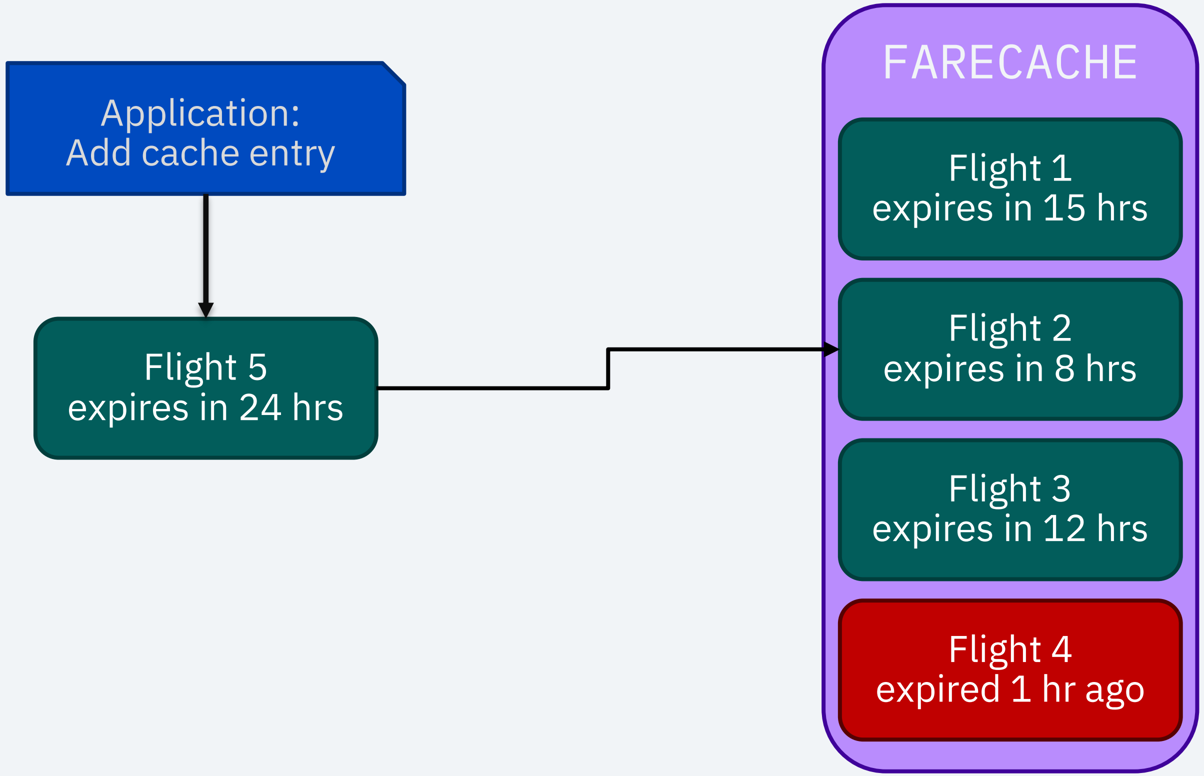
To-be

Logical record cache support keeps track of entries' expiration time and efficiently finds an expired entry without performing costly searches.

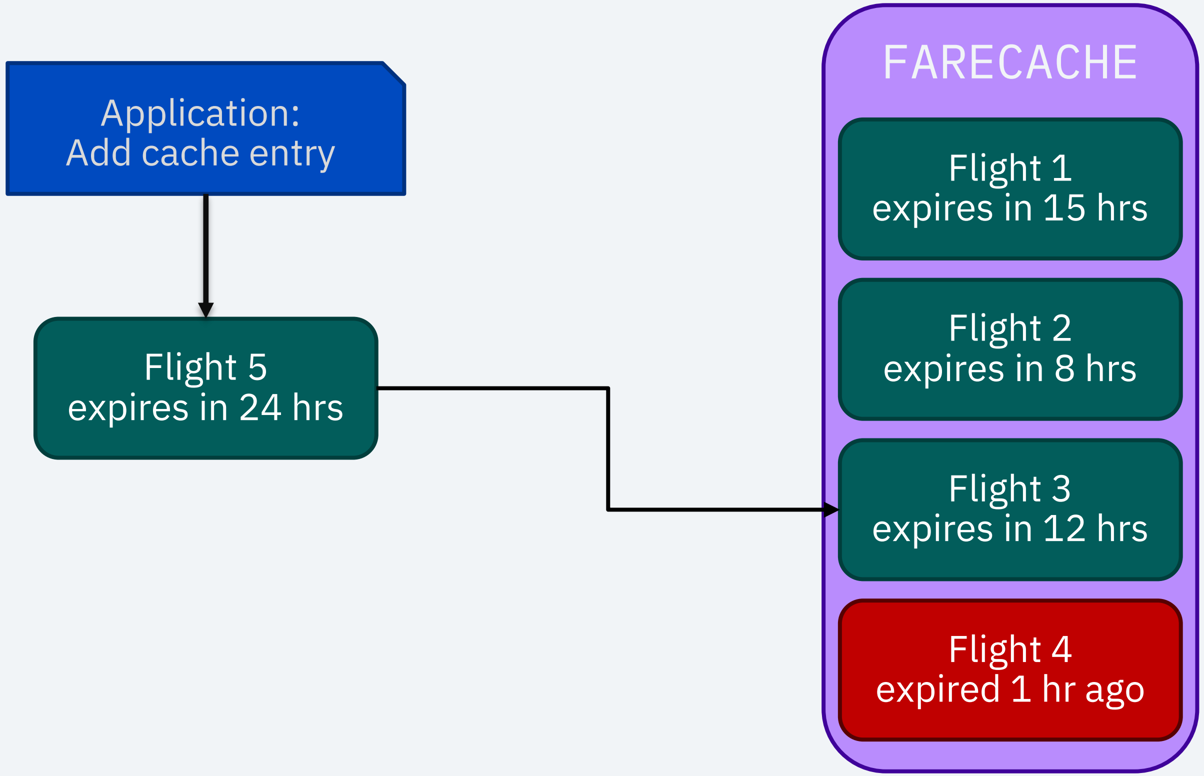
Castout processing (before APAR)



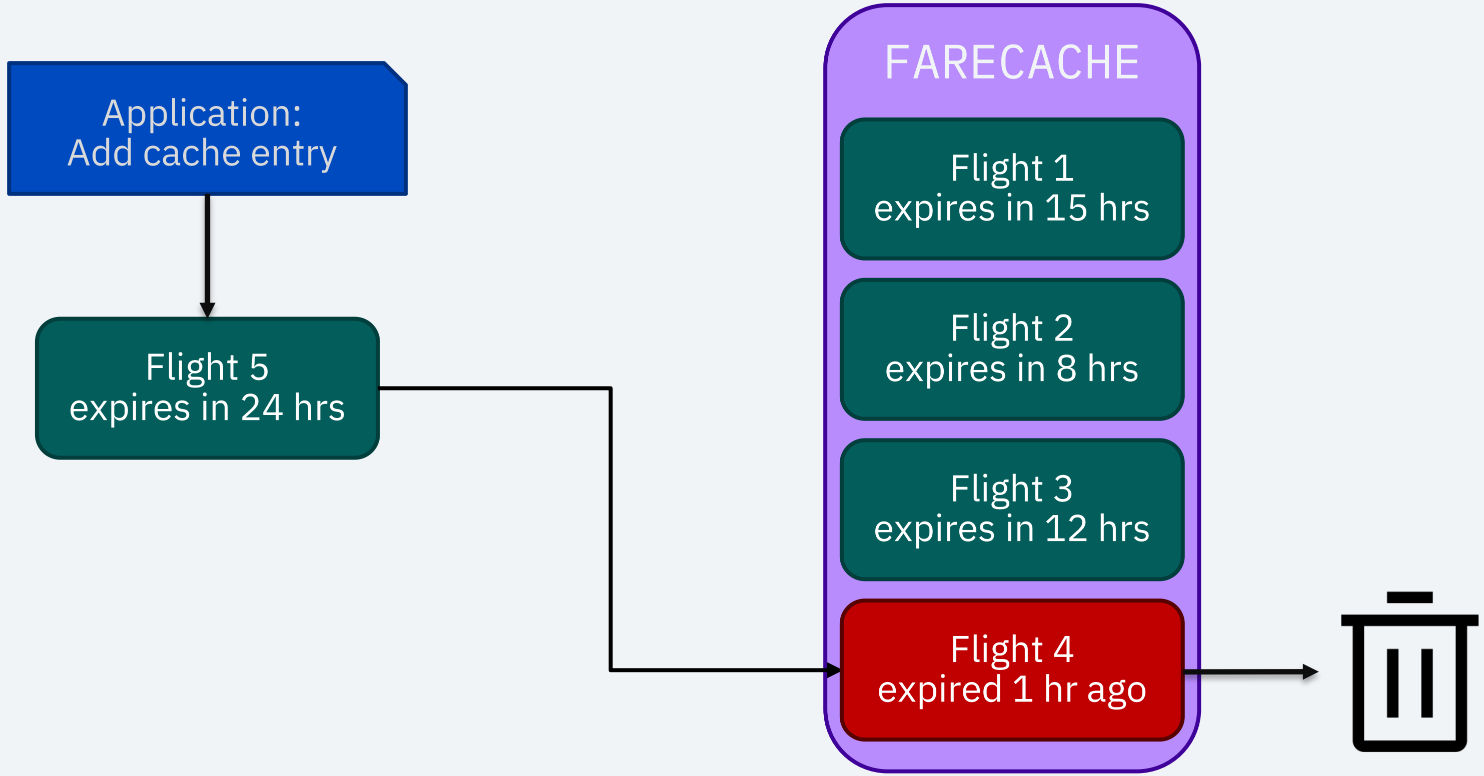
Castout processing (before APAR)



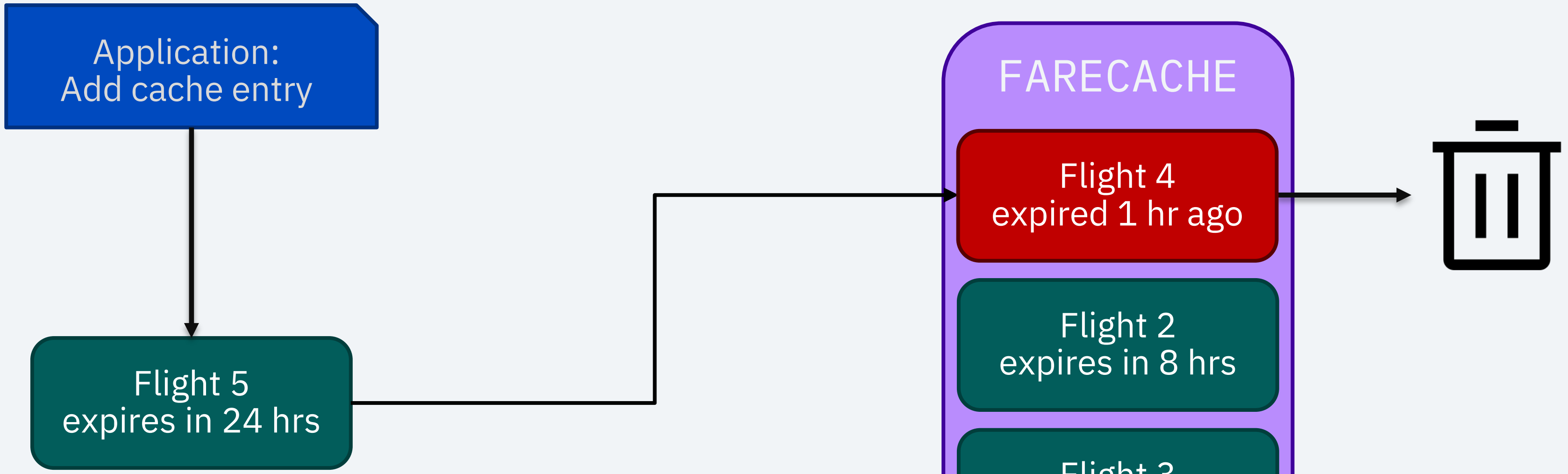
Castout processing (before APAR)



Castout processing (before APAR)



Castout processing (with APAR applied)



- This is a logical view only; we don't actually move cache entries in memory
- Only applies to caches with global castout time
- If no entries are expired, we cast out least recently used

Usability improvements

As-is

Because cache entries aren't removed as they expire, expired entries are not detected by online displays of in-use cache entries. As a result, there might appear to be more in-use entries than there really are.

```
> ZCACH DISPLAY COUNTS MYCACHE

          CACHE ENTRIES DEFINED     100000
          CACHE ENTRIES IN USE      100000
          .
          .
          .
```

To-be

Online displays of in-use cache entries can more accurately account for expired entries.

```
> ZCACH DISPLAY COUNTS MYCACHE

          CACHE ENTRIES DEFINED     100000
          CACHE ENTRIES IN USE      47813
          .
          .
          .
```

Logical record cache sweeper

- Before enhancement, cache support kept expired entries in cache until an app added a new entry to a full cache
- IBM-controlled cache sweeper removes entries as they expire
- Provides more accurate metrics of in-use cache entries
  - `ZCACH DISPLAY COUNTS`
  - Data collection & reduction

Conclusion

- Use cache definitions to increase the number of entries in a cache without losing existing cached data
- Castout processing is improved to eliminate costly searches through a cache using application ECB time
- Cache sweeper removes expired entries from the cache to provide more accurate cache usage metrics
- Delivered in APARs PJ46253 & PH30577 (Nov. 2022)

Thank you