Logical Record Cache – Or why I don't get out of bed for less than 5GB

David Morley - Marriott
z/TPF Technical Architect

TPF User Group – Denver 2019
SOA Services Subcommittee

Formatted Output Builder

MARSHA supports semi-static data for descriptive content about rooms and rates. This content was originally green screen text but a service was created in TPF41 that allowed it to be rendered dynamically according to a template in 6 different languages including English, German, Spanish, Simplified Chinese, French and Portuguese.

Content is processed natively in UTF-8.

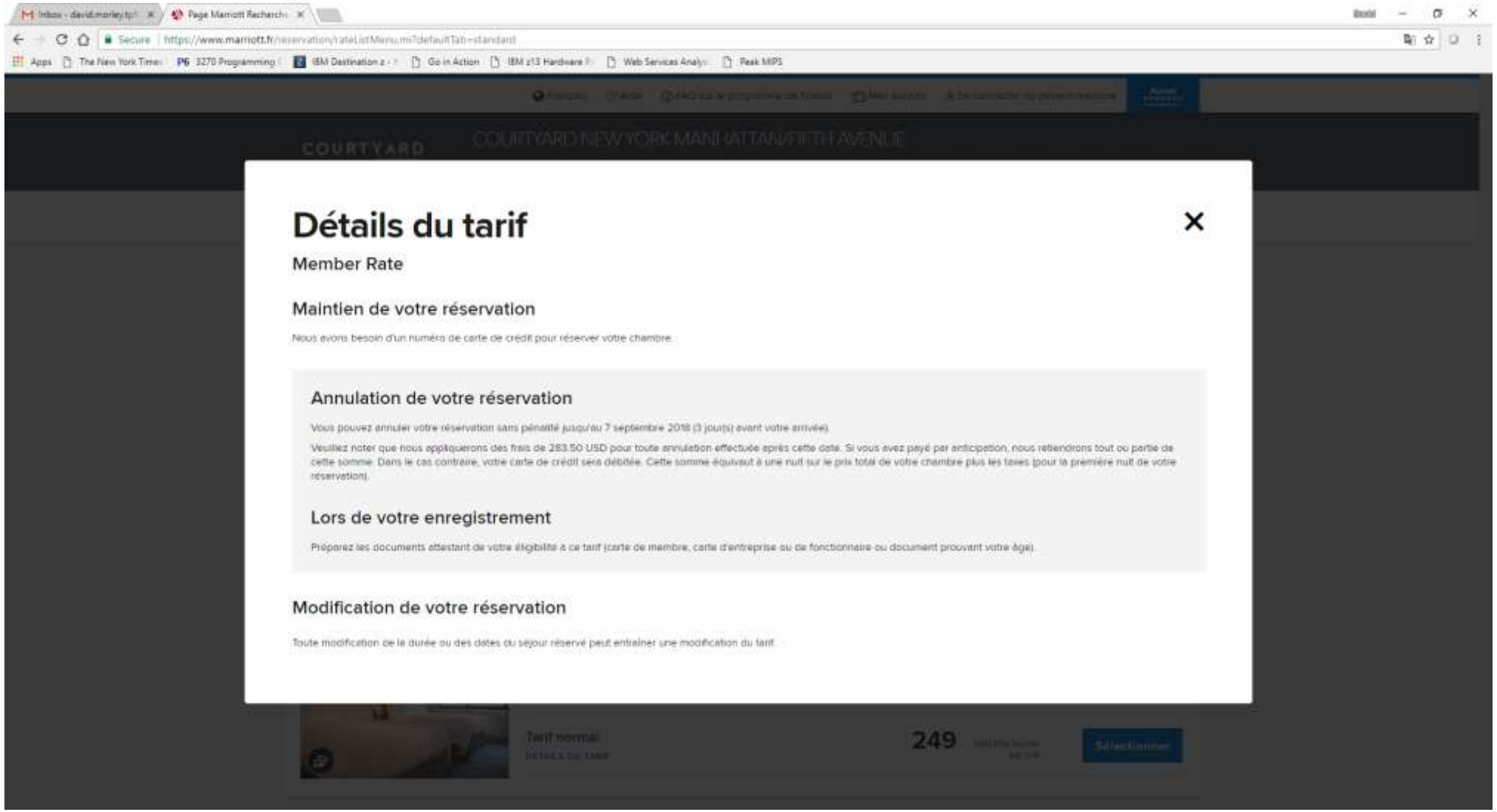
It was incredibly popular! Within 6 months it was supporting about 4000 calls per second.

Cost of real-time rendering was quite high, at 4000 calls per second the service consumed 10% of all CPU resources.

Post z/TPF migration a Format 2 Global Table was added for English language room related content.

Subsequently refactored to use the Logical Record Cache that extended coverage to all languages and rate related content.

12 languages in real-time



Current State

Content is now rendered in 12 languages:

- Original 6: English, German, Spanish, Simplified Chinese, French and Portuguese.
- Plus: Russian, Japanese, Italian, Korean, Traditional Chinese and Arabic

Different descriptions by Channel

- Website vs Mobile vs GDS vs OTA

Different descriptions by Date of Arrival

- Seasonal Promotions

It now supports 80,000 calls per second, or 20x original implementation.

Adding new languages is a configuration exercise. Starwood supported several other languages:

- Dutch, Indonesian, Polish, Thai and Turkish.

Cache automatically adjusts for most popular content

- Degrades gracefully – Least Recently Used (LRU) algorithm.
- Monitor for growth of properties and languages and channels.

Cache Design

Access to language content is through an API.

- Do NOT expose cache API directly to application layer.

API was ‘injected’ to replace an ENTRC to KOD0.

- KOD0 is a DF Application that pulls the content and renders the description according to a pattern for the description.

Cache is designed to balance accuracy with hit rate.

- 100% accuracy causes ‘outages’ due to corporate-wide changes (cache would be wiped-out).
- Acceptable lifetime (convergence) 6-9 hours.

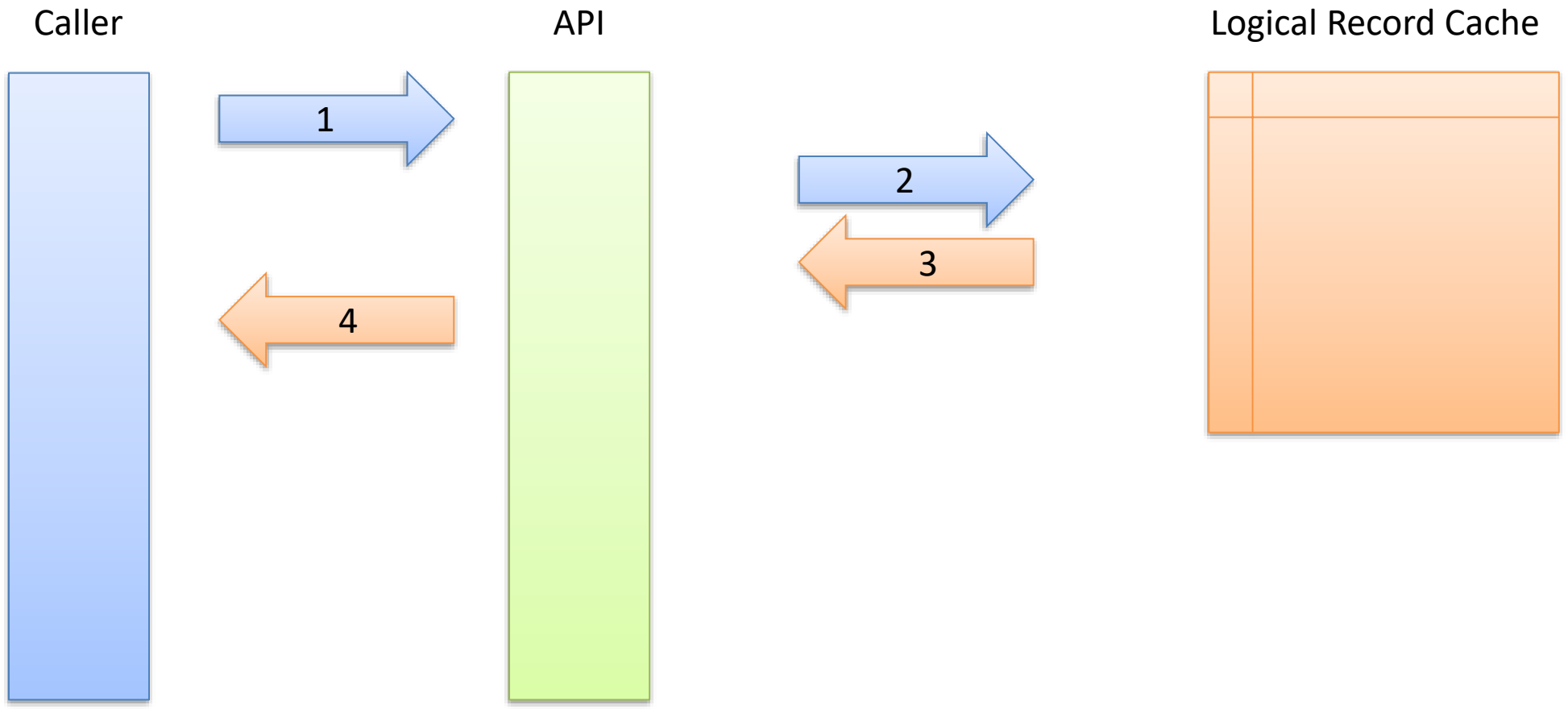
API was then redirected into the cache.

- Cache hit returns content.
- Cache misses go regular path (enters KOD0).
  - Result from KOD0 is written to cache before returning to caller.

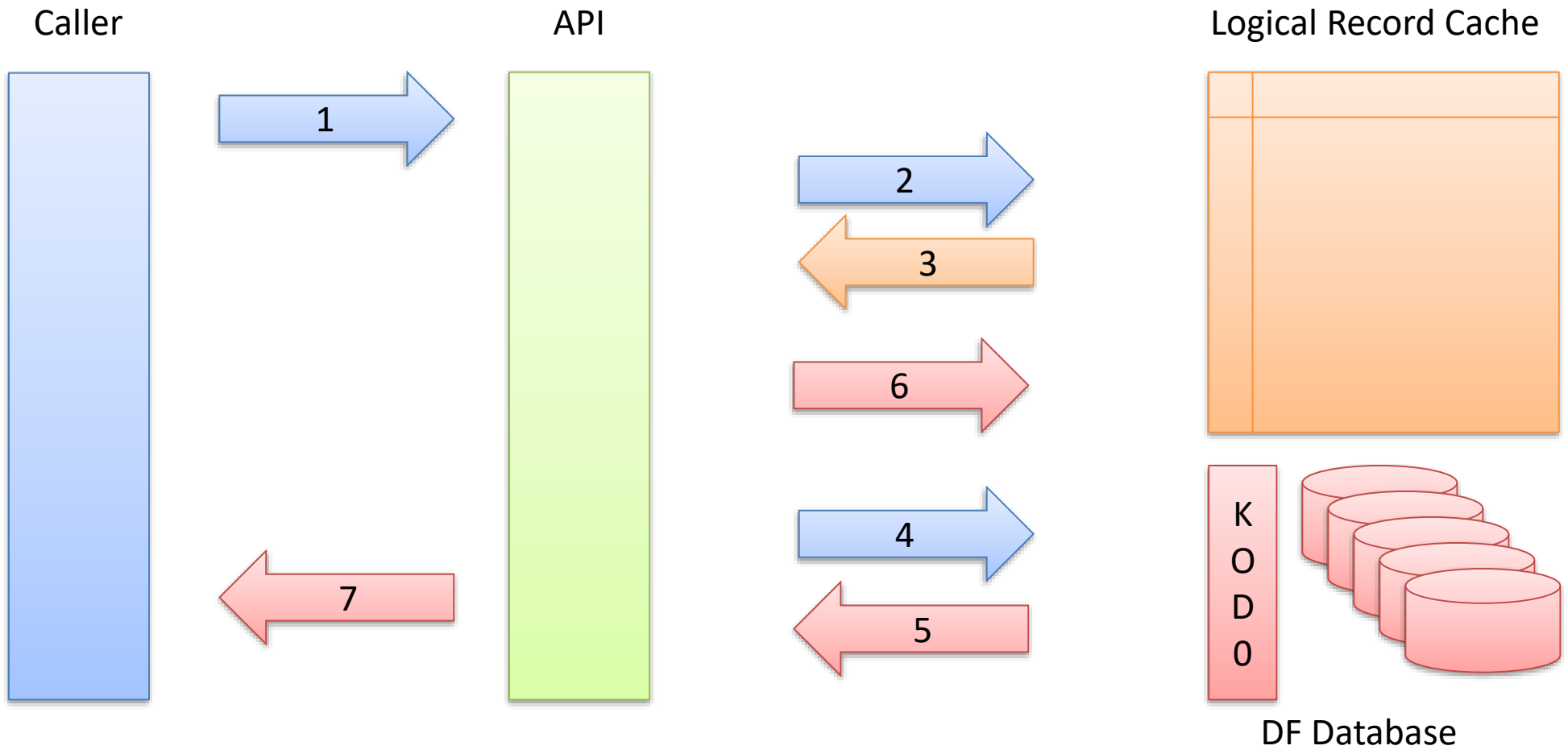
Initial implementation focused on C++ users.

- API was C++ only.
- Assembler paths were ‘going away’.

Logical Record Cache – Look Aside Design



Logical Record Cache – Look Aside Design



Current Logical Record Cache

Current Configuration

4 caches:

- 200K Items (Room Pool Name) : 100MB
- 200K Items (Room Pool Description) : 100MB
- 128M Items (Rate Description Index) : 20GB
  - 640K Items (Rate Description) : 400MB

Cache size ~21GB.

Waiting on the APARs to extend it further.

Caches now support 80,000 calls per second and reduces CPU by 50%.

Deployment Experiences

Rapid expansion quickly found the limits of z/TPF handling of system heap.

- CTL-571 when >1GB of system heap is being allocated by GSYSC.
- Use pre-allocated storage (PUT14).

Tuning the cache is ‘tricky’ due to limited insight into the hash tables.

- Cannot see how many items are ‘in-use’ – PUT15 (?) enhancement.

Cannot disable cache during peak load.

- System would slow to a crawl while cache repopulates.
- 20% of the data provides 80% of performance benefit.

Hide Cache from developers.

- Create an API to access data ‘object’.
- Intercept API to use cache and write back.
- Delete cache entries on updates (if possible) to extended time to live of data.

Avoid Castouts – Cache is too small for working set.

Summary

LRC is superior to VFA if it avoids a lot of record processing.

- Impedance Mismatch.

Once caches offset a certain level of CPU they move from nice to have to indispensable.

- e.g. 10-15%.

Could no more run without cache as we could run without VFA.

- LRC 80K calls per second @ 93% hit rate = saves 50% of CPU, >1.2M I/Os per sec.
- VFA >5M calls per second @ 99.8% hit rate = saves 30% of CPU*, >5M I/Os per sec.

Questions?

This page intentionally left blank.