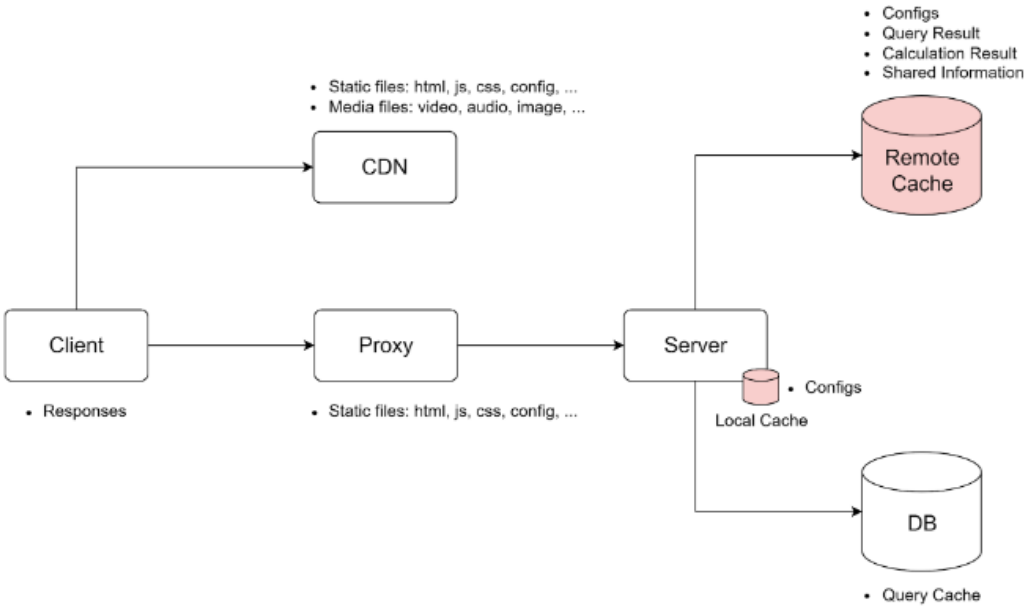
Cache

Cache lưu trữ dữ liệu tạm thời phục vụ cho các request yêu cầu xử lý trong tương lai 1 cách nhanh tróng hơn. và là 1 thành phần của phần cứng hoặc phần mềm được lưu trữ tạm thời.
Dữ liệu trong cahe thường rơi vào 2 trường hợp :

- 1 bản coppy data của data source .
- Là kết quả tính toán từ trước và được lưu vào .
  Cache là 1 tấm chắn bảo vệ database
  Cache lưu dữ liệu trên memory , memory là phần cứng có sự hạn chế.

Trade off
vì memory có hạn nễn sẽ đánh đổi giữa

- performance vs cost(space).
- performance vs consistency .



client sẽ gửi request lên proxy từ proxy sẽ forward tới sever backend .
ở sever backend này sẽ query data từ databse hoặc remote cache.
ngoài ra thì sever cũng có memory của bản thân nó hay được gọi là local cache.
CDN sử dụng lưu trữ

1. **Database caching**: Databases often have built-in caching for frequently accessed data. This can significantly speed up repeated queries.

2. **Application caching**: In the application layer, developers can implement caching to store and reuse frequently used data instead of re-fetching or recalculating it. This can be done using in-memory data structures or external caching systems like Redis or Memcached.

3. **HTTP caching**: Web servers and browsers use caching to store HTML pages, images, and other resources. This reduces the load on the server and makes pages load faster for users.

4. **Content Delivery Network (CDN) caching**: CDNs cache static resources closer to users to reduce latency.

5. **Browser caching**: Browsers cache resources locally to speed up page loading times.

Read strategies

- read through
- read -aside

---

read aside hoạt động như nào .
đầu tiên khi đọc dữ liệu sẽ đọc vào trong cache . Nếu dữ liệu có trong cache (**Cache hit** có dữ liệu **Cache miss** không có dữ liệu)
sẽ trả dữ liệu về Ứng dụng (Application).

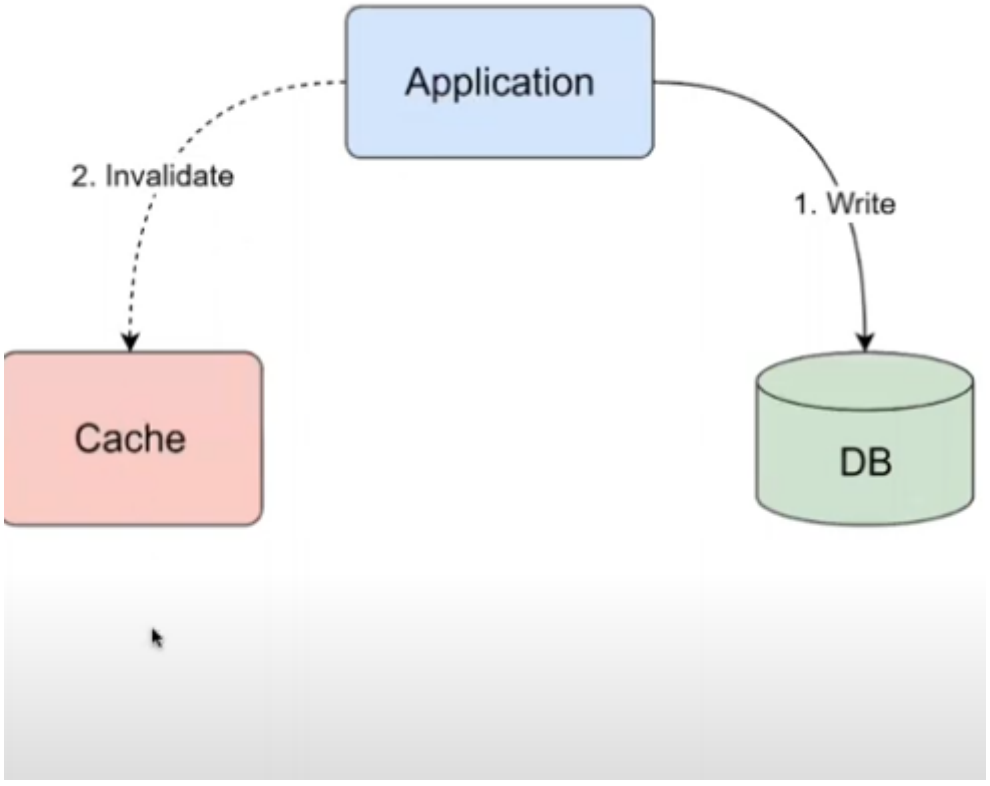
Trong trường hợp dữ liệu không có trong cache thì Application sẽ đọc vào database sau đó truyền về application và ghi lại cache để phục vụ các cache đằng sau.

Read aside

- ưu:
  - chống chịu lỗi của Cache. (cache vị go down chết thì ứng dụng của mình vẫn hoạt động bth . Luồng xử lý của ứng dụng vẫn hoạt đọng dc)
  - khả năng linh hoạt về lưu trữ dữ liệu ở trong cache.
- nhược:
  - Làm phức tạp ứng dụng.
  - Dữ liệu không nhất quán.

Write Strategies

- write-Through
- Write back
- write-around
  đối với 2 write trên thì cache đứng trước database . Application của mình chỉ thao tác với cache thôi.



  đối với write around
- Invalid cache asynchronously : xử lý bất đồng bộ với cache . Khi mà cập nhập hoặc xoá các mục trong bộ nhớ cache mà không cần chờ đợi quá trình đồng bộ hoàn tất .
  ưu điểm :
- Decoupling cache and storage systems: nghĩa là tách biệt cache và hệ thống lưu trữ dữ liệu ra riêng. để giảm sự phụ thuộc của 2 hệ thống .
  nhược điểm :
- Inefficient for Frequently Accessed Data : không hiệu quả cho dữ liệu cập nhập thường xuyên
- Data inconsistency: không nhất quán.
  ví dụ như data trong cache và db khác nhau .
  thường là 1 trong những vấn đề phổ biến hay xảy ra.
  cách giải quyết đó là vô hiệu hoá cache
  Time-based : cache sẽ dc xoá hoặc vô hiệu hoá sau 1 khoảng thời gian
  Command-based: vô hiệu hoá hoặc xoá bằng lệnh được gửi từ sever
  Event-based: được vô hiệu hoá hoặc xoá khi xảy ra 1 sự kiện nhất định
  Group-based.

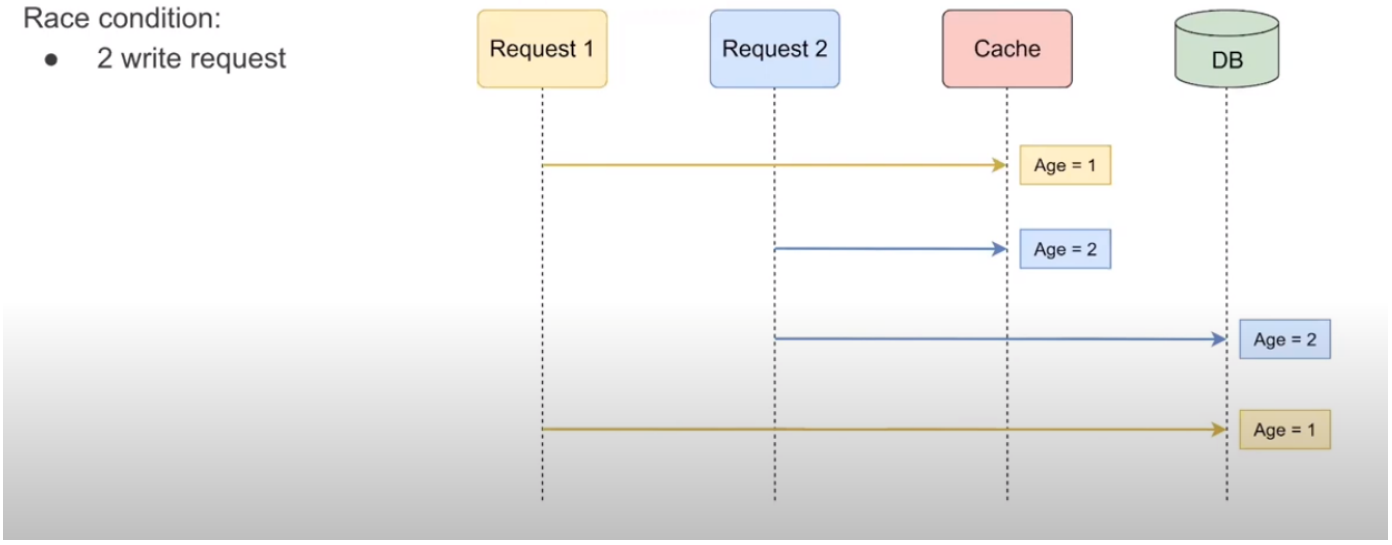
Tại sao Cache validation lại khó
Là độ phức tạp của nhiều yếu tố

- Timing (thời gian) : làm sao để xác định thời gian time to live (TTL) cho phù hợp.
- Concurrency(tính đồng thời). gây ra vấn đề về race condition.
- Data relationships. Nếu cache được lưu có quan hệ với 1 số cache khác khi invalid 1 thằng thì nó sẽ invalid những thằng xung quanh nó.
- unlike database cache can be everywhere and anywhere.
- thing can go wrong in milion different ways

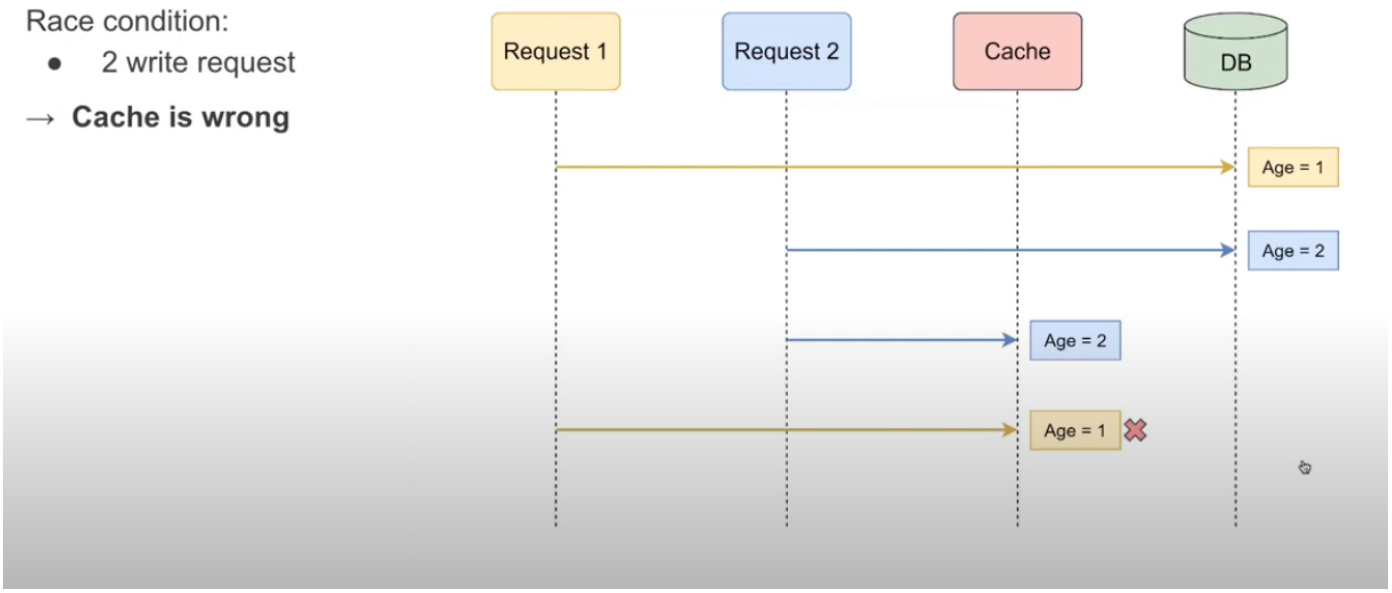
Cache và database khác nhau điểm nào?

Cache sẽ sử dụng ở mọi nơi- client- proxy- cdn- remote cache- Database là nằm tập chung ở 1 nơi.

Race condition :ví dụ 2 thao tác đến cùng 1 lúc nếu như có 1 cơ chế xử lý đúng thì dữ liệu của 2 thao tác đó ko còn valid nữa
thử giải quyết vấn đề data inconsistency
nếu upadte trước và update cahe sau. --- write around ---
... Update cache trước ...



xảy ra race condition -> db wrong
... Update cache sau ...



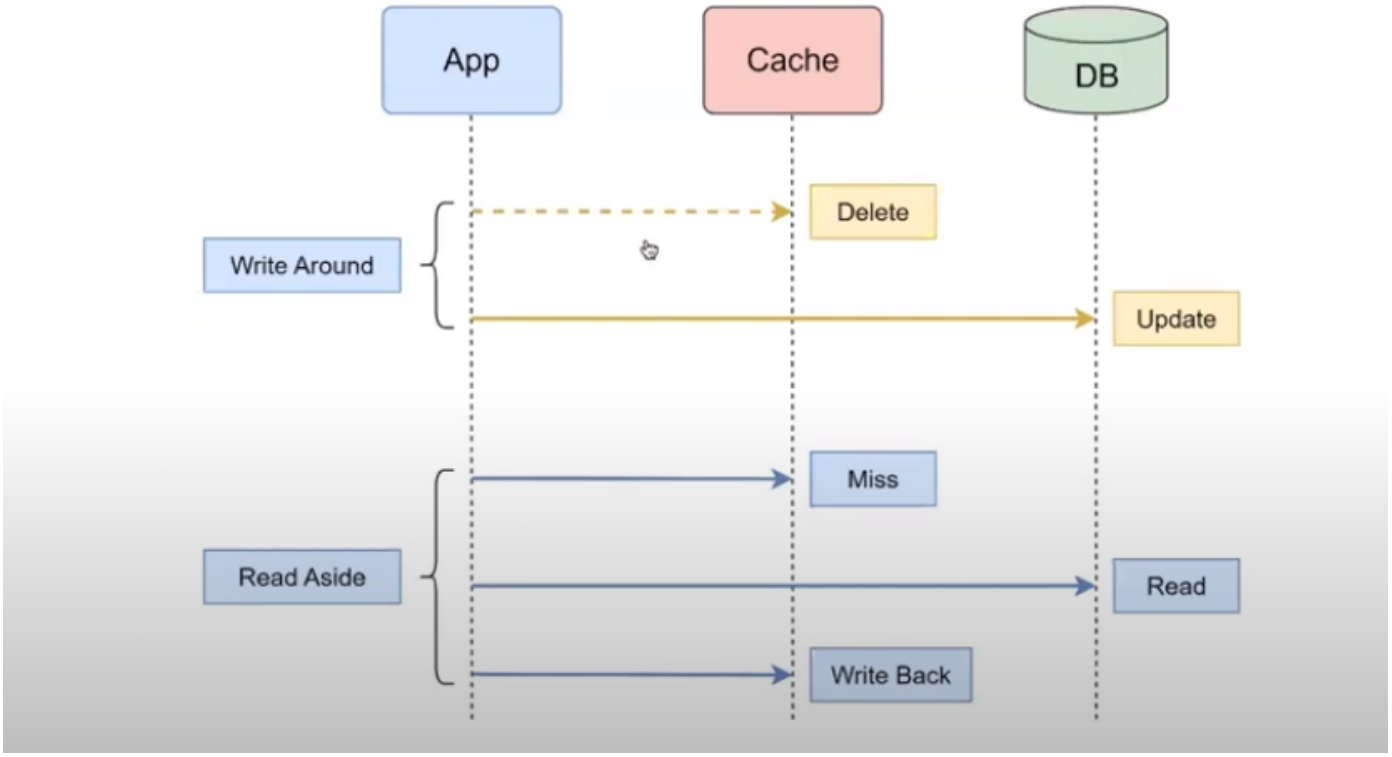
dù cho sử dụng first or later cũng không giải quyết được vấn đề data inconsistency at all

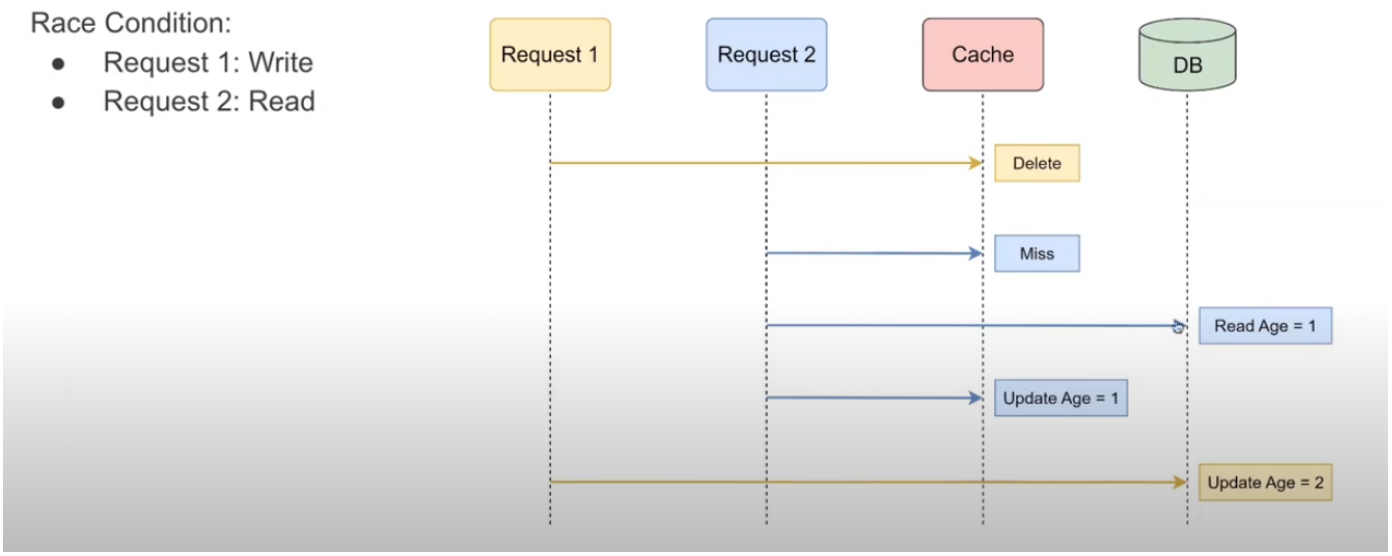
write around

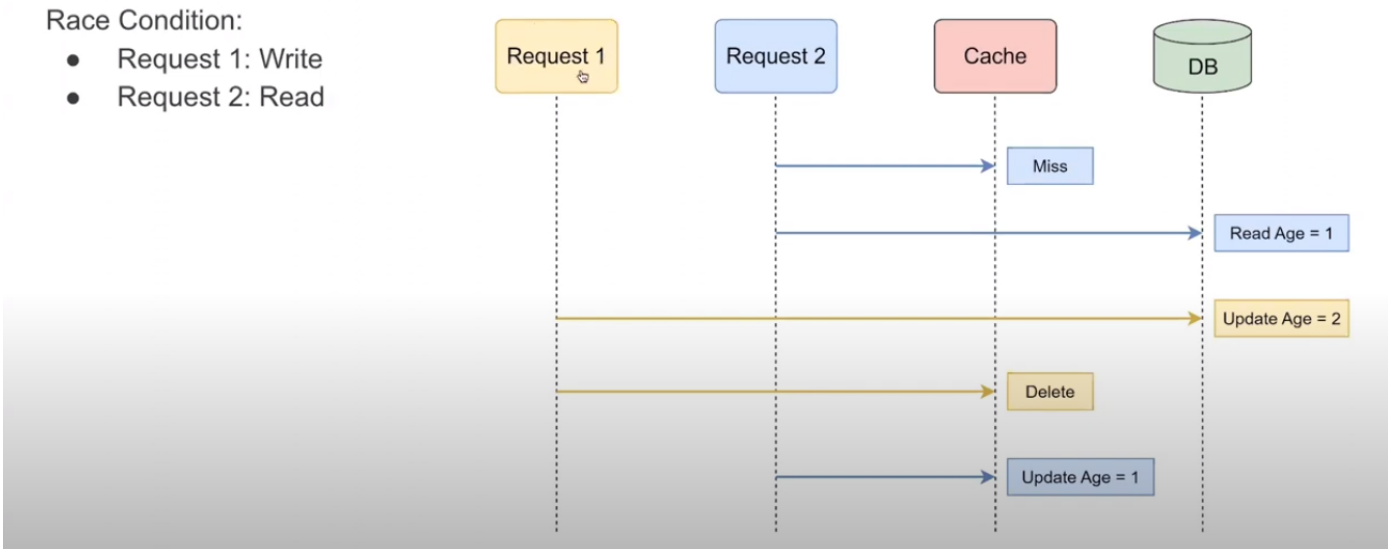
còn đối với delete cache trước và sau thì db sẽ đúng nhưng no data in cache after 2 write

.

Nếu kết hợp Write around và Read Aside thì ?



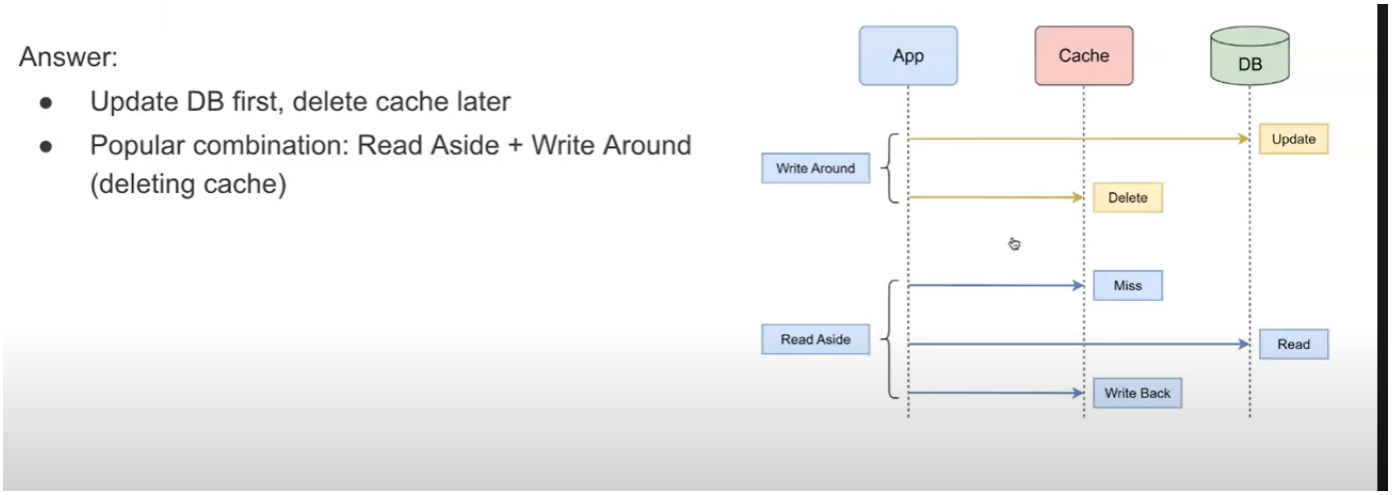




vẫn còn tồn tại data consistency

-> DB đúng cache sai.

Thì trong những lựa chọn trên chúng ta sẽ chon Update DB trước delete sau



vì ghi dữ liệu vào trong cache sẽ nhanh hơn là ghi vào db nên đoạn khoảng ở Read aside khi read db xong nó sẽ write cache ngay nên sẽ hiếm khi bi request 1 chèn chân vào .

Thế thì tiếp đến câu hỏi làm sao để hạn chế việc data consistency ?
mặc dù là tỉ lệ xảy ra thấp nhưng vẫn có thể xảy ra
-> add short TTL to cache data