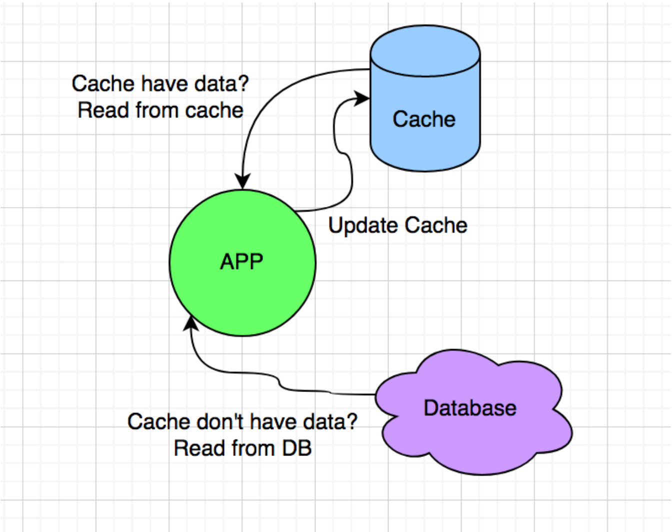
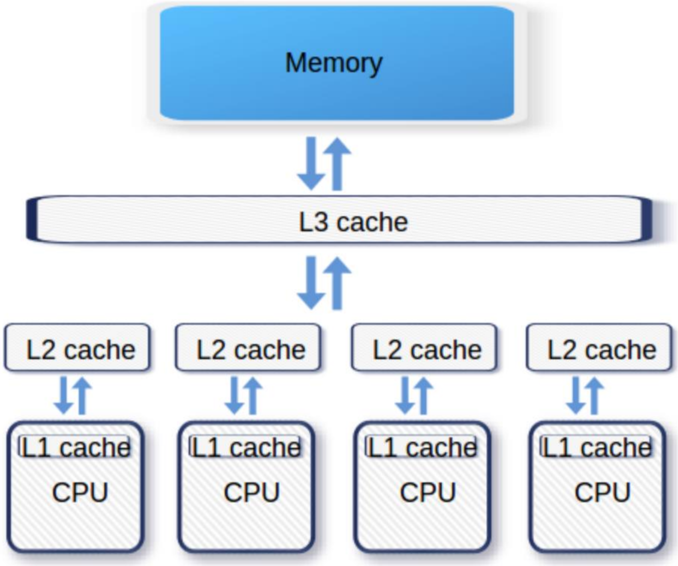
আজকে আমরা জানবো ক্যাশিং কি এবং এল-আর-ইউ ক্যাশ কিভাবে কাজ করে। তুমি যদি ক্যাশিং সম্পর্কে আগে থেকেই জানো তাহলে প্রথম অংশটা দ্রুত পড়ে পরের অংশে চলে যেতে পারো।

ধরা যাক তুমি একটি অ্যান্ড্রয়েড অ্যাপ তৈরি করছো। অ্যাপের ইউজারদের তুমি ৩টা রকমের লেভেল অ্যাসাইন করেছো, প্লাটিনাম, গোল্ড আর সিলভার। যখনই কোনো ইউজার অ্যাপ চালু করে তখনই তোমার ডেটাবেসে ইউজারের লেভেল চেক করতে হয়। তোমার অ্যাপ এ ইউজার আছে এক লাখ এবং প্রতি ঘন্টায় কয়েক হাজার ইউজার অ্যাপটি ব্যবহার করে। একসময় তুমি দেখলে ডাটাবেজের রিকুয়েস্ট খুব বেশি বেড়ে যাচ্ছে এবং পেজ লোড স্লো হয়ে যাচ্ছে। এই সমস্যার সমাধান হলো কিছু ইউজারের লেভেল মেমরিতে সেভ করে রাখা। যখনই ইউজার অ্যাপ চালু করবে তখন আগে তুমি দেখবে সেই ইউজারের লেভেল মেমরিতে সেভ করা আছে নাকি, যদি থাকে তাহলে তোমার আর ডাটাবেসে রিকুয়েস্ট পাঠানো লাগবে না। এই প্রক্রিয়াটাকেই বলা হয় ক্যাশিং।



ক্যাশিং যে সবসময় ডাটাবেস থেকে পড়ার জন্য করা হবে সেটা না। কম্পিউটারে হার্ডডিস্ক থেকে ডাটা নিয়ে র‍্যামে রাখা হয় দ্রুত এক্সেস করার জন্য, এটাও এক ধরনের ক্যাশিং। আবার প্রতিটা সিপিউ কোর কিছু ডাটা নিজের কাছে ক্যাশ করে রাখে যাতে র‍্যাম থেকেও বেশি না পড়তে হয়। বুঝতেই পারছো ক্যাশিং অনেক লেয়ারে হতে পারে।

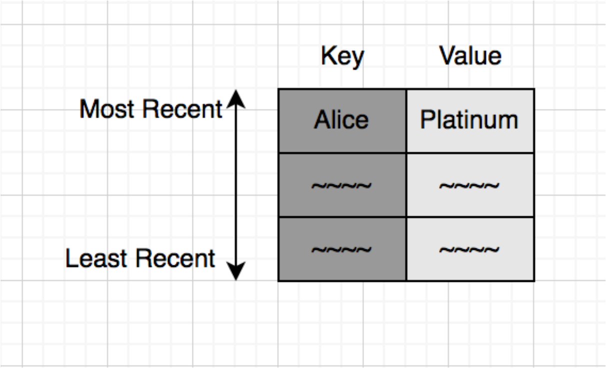


এখন ক্যাশিং করার সময় একটা সমস্যার মুখোমুখি সবসময় হতে হয়, সেটা হলো ক্যাশ মেমরির আকার ডাটাবেস, হার্ডডিস্ক থেকে অনেক ছোট হয়। তোমার হার্ডডিস্কে হয়তো ৫ টেরাবাইট জায়গা আছে, কিন্তু তোমার র‍্যামের সাইজ খুব বেশি হলে ৩২ গিগাবাইট। সমস্যা যেটা হয়, সব ডাটা একসাথে ক্যাশ মেমরিতে লোড করা যায় না। নতুন ডাটা লোড করার সময় যদি ক্যাশ ফুল হয়ে যায় তাহলে পুরানো কিছু ডাটা ফেলে দিতে হয়। এই প্রসেসটাকে বলা হয় Cache Eviction বা ক্যাশ এভিকশন পলিসি। ক্যাশ এভিকশনের জন্য অনেক ধরনের অ্যালগরিদম আছে, যেমন: FIFO (First in first out), FILO (First in last out), LRU (least recently used) ইত্যাদি।

LRU ক্যাশ

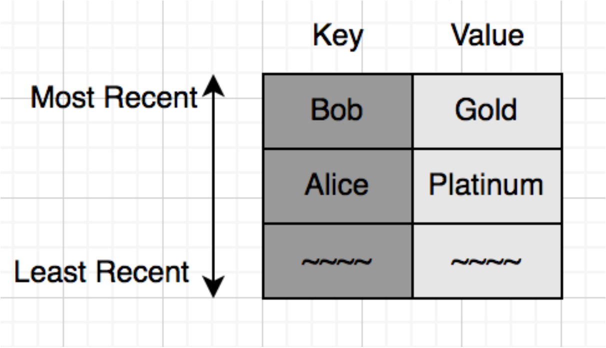
আমরা আজকে দেখবো LRU ক্যাশ কিভাবে কাজ করে এবং কিভাবে ইমপ্লিমেন্ট করতে হয়। LRU ক্যাশ প্রতিবার ডাটা এক্সেস করার সময় মনে রাখে শেষ কখন সেই ডাটা এক্সেস করা হয়েছিলো। নতুন ডাটা ক্যাশে সেভ করার সময় যদি ক্যাশ ফুল হয়ে যায় তাহলে সে যে ডাটা পুরোনো হয়ে গেছে সেটাকে ফেলে দেয়। এক্ষেত্রে যে ডাটা যত আগে এক্সেস করা হয়েছিলো সেটা তত পুরানো (least recent)।

একটা ছোট সিমুলেশন করে ফেলি। মনে করো তোমার ক্যাশ মেমরি সাইজ মাত্র ৩, অর্থাৎ মাত্র ৩জন ইউজারের ডাটা আমরা ক্যাশে রাখতে পারবো। শুরুতেই মনে করো Alice লগইন করলো এবং তুমি ক্যাশে কুয়েরি করে দেখলে যে Alice সেখানে নেই। তুমি ডাটাবেসে কুয়েরি করে দেখলে Alice একজন প্লাটিনাম ইউজার, তুমি সেই ডাটা ক্যাশে রেখে দিলে। নিচের ছবি দেখো:



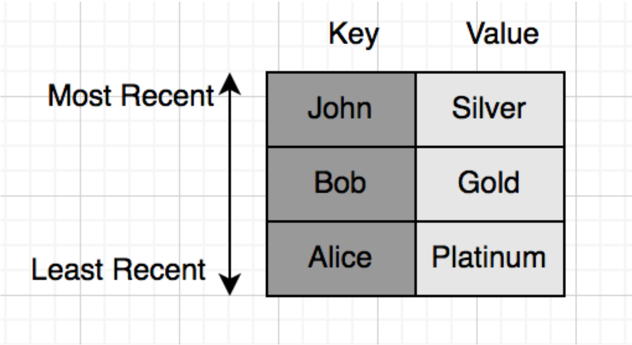
Key	Value
Alice	Platinum
~~	~~~~
~~	~~~~

এখন আরেকজন ইউজার Bob আসলো যে একজন গোল্ড ইউজার, তাকেও ক্যাশে যোগ করি:



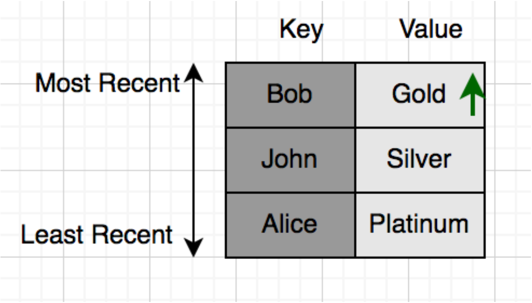
Key	Value
Bob	Gold
Alice	Platinum
~~	~~~~

এরপরে আসলো সিলভার ইউজার John:



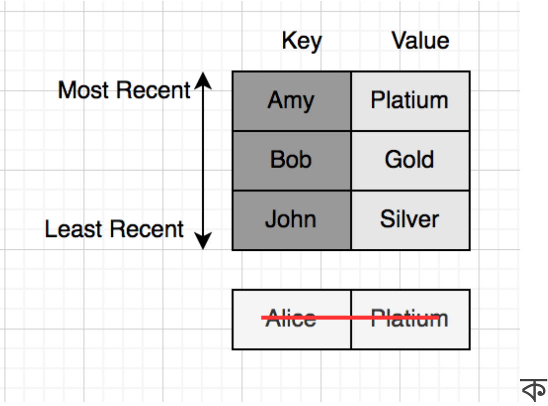
Key	Value
John	Silver
Bob	Gold
Alice	Platinum

এদিকে Bob করলো কি, অ্যাপে অন্য একটা পেজ ওপেন করলো যেই পেজে আবার ক্যাশে কুয়েরি করলো। এখন Bob এর স্ট্যাটাসে ক্যাশেই আছে, ডাটাবেস কুয়েরি করা দরকার হবে না। এদিকে Bob এর স্ট্যাটাস যেহেতু সবশেষে এক্সেস করা হয়েছে, সে হয়ে যাবে ‘most recent’, বাকিদের অর্ডার একই থাকবে:



Key	Value
Bob	Gold
John	Silver
Alice	Platinum

এখন যদি চতুর্থ কোনো ইউজার Amy আসে, তাহলে কি হবে? যে Key তে সবার পুরোনো সে বাদ যাবে, এক্ষেত্রে সে হলো Alice।



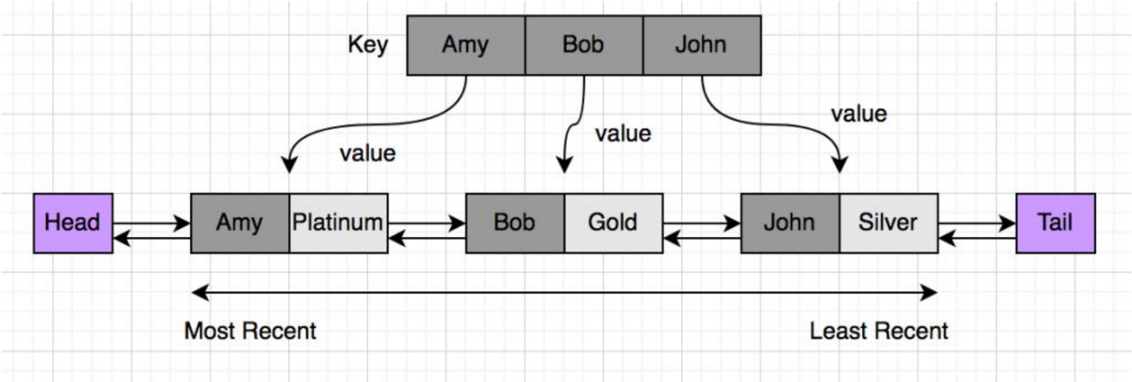
Key	Value
Amy	Platium
Bob	Gold
John	Silver
~~Alice~~	~~Platium~~

ক্যাশ ইমপ্লিমেন্ট করার সময় আমাদের মূলত ২টা বেসিক ফাংশন ইমপ্লিমেন্ট করতে হয়। একটা হলো get(key) যেটা ইনপুট হিসাবে একটা key নেয় এবং ক্যাশ থেকে সেই key টা খুজে বের করে, আরেকটি হলো put(key, value) যেটা একটা key-value পেয়ারকে ক্যাশ এ সেভ করে। তো আমরা এখন দেখবো কিভাবে LRU ক্যাশের জন্য আমি এই দুটি ফাংশন ইমপ্লিমেন্ট করতে পারি।

- **অ্যালগরিদম ১:** আমরা একটা লিস্ট/ভেক্টর ব্যবহার করতে পারি ক্যাশ হিসাবে। লিস্টের প্রতিটা পজিশনে (key, value) এর সাথে সাথে শেষ কখন এক্সেস করা হয়েছিলো সেটা রেখে দিতে হবে। put অপারেশনের জন্য একটা লুপ চালিয়ে সবার পুরানো ইনডেক্সটাকে রিপ্লেস করে দিবো। get অপারেশনের জন্য লুপ চালিয়ে key টাকে খুজে বের করবো। তাহলে দুটি অপারেশনেরই টাইম কমপ্লেক্সিটি O(n), আমাদের আরেকটু ভালো কিছু দরকার।
- **অ্যালগরিদম ২:** লিস্টের বদলে আমরা একটা হ্যাশম্যাপ ব্যবহার করতে পারি। আগের মতোই put অপারেশনের জন্য key গুলোর উপর লুপ চালিয়ে পুরানোটা রিপ্লেস করে দিবো O(n) কমপ্লেক্সিটিতে। তবে get অপারেশন এখন খুব দ্রুত কাজ করবে, হ্যাশম্যাপে এভারেজ কেস এ O(1) এ key খুজে বের করা যায়। কিছুটা উন্নতি হয়েছে।
- **অ্যালগরিদম ৩:** আগের অ্যালগরিদমের বটলনেক ছিলো সবথেকে পুরানো key টা খুজে বের করা। কোন ডাটা স্ট্রাকচার ব্যবহার করলে key গুলো সর্টেড থাকবে? আমরা একটা ব্যালেন্সড বাইনারি সার্চ ট্রি ব্যবহার করতে পারে। সেক্ষেত্রে put এবং get দুটোই O(logn) এ কাজ করবে।
- **অ্যালগরিদম ৪:** ডাবলি (Doubly) লিংকড-লিস্ট ব্যবহার করে দুটি অপারেশনই O(1) এ করা সম্ভব। আজকের লেখায় এই অ্যালগরিদমটাই বিস্তারিত বর্ণনা করবো।

ডাবলি (Doubly) লিংকড-লিস্ট ব্যবহার করে LRU ক্যাশ

উপরে অ্যালগরিদম ২ তে হ্যাশম্যাপ ব্যবহার করে একটা সলিউশনের কথা বলেছিলাম। হ্যাশম্যাপে key-value পেয়ারগুলো রেখে দিলে আমরা খুব দ্রুত key খুজে পাচ্ছি। কিন্তু সমস্যা হচ্ছিলো সবথেকে পুরানো key টা খুজে বের করতে। সেটার সমাধান করতে আমরা ডাবলি (Doubly) লিংকড-লিস্ট ব্যবহার করবো। ডাবলি লিংকড লিস্টে প্রতিটা নোড তার সামনের এবং পিছের নোডকে পয়েন্ট করে। নিচের ছবিটা দেখো, তাহলে ব্যাখ্যা করতে সুবিধা হবে:



আমি এবার আমার ডাটাগুলোকে একটা লিংকড লিস্টে রেখে দিয়েছি। লিস্টের প্রতিটা নোড একটা করে key-value পেয়ার। লিস্টের বামে হেড এর কাছে থাকবে সবথেকে নতুন ডাটা, ডানে থাকবে সবথেকে পুরানো ডাটা। সেই সাথে আমার আগের মতোই একটা হ্যাশম্যাপ থাকবে তবে এবার হ্যাশম্যাপে শুধুমাত্র ভ্যালুর বদলে থাকবে লিংকড লিস্টের নোডের অ্যাড্রেস।

এবার যখন একটা ডাটা আসবে তখন খুজে দেখতে হবে যে ডাটাটা আগেই হ্যাশম্যাপে আছে নাকি। যদি থেকে থাকে তাহলে লিংকড লিস্ট থেকে সেই নোডটা আগে ডিলিট করে দিতে হবে। এরপর নতুন একটা নোড লিস্টের বামে বসিয়ে দিতে হবে। কোনো ডাটা যখন কুয়েরি করা হবে তখনও একই কাজ করতে হবে কারণ যে ডাটা কুয়েরি করা হচ্ছে সেটাই হয়ে যাচ্ছে সবথেকে নতুন ডাটা।

ডাবলি লিংকড লিস্টে নোড ডিলিট করা খুব সহজ, পিছের নোডটাকে সামনের নোডের সাথে পয়েন্ট করিয়ে দিলেই হয়ে যায়। আর হ্যাশম্যাপে নোডের অ্যাড্রেস রেখে দেয়ার কারণে O(1) এভারেজেই নোডটা খুজে পাওয়া যাবে।

তাহলে এখন আর Put অপারেশনের জন্য সবথেকে পুরানো ডাটা লুপ চালিয়ে খুজে বের করা লাগবে না। যদি ক্যাশের ক্যাপাসিটি পূর্ণ হয়ে যায় তাহলে সবথেকে বামের নোডটাকে ডিলিট করে দিলেই হয়ে যাচ্ছে যেটা O(1)O(1) এই করা সম্ভব।
তাহলে আমরা এমন একটা অ্যালগরিদম পেয়ে গেলাম যেটা O(1)O(1) এ LRU ক্যাশ থেকে ডাটা রিড-রাইট করতে পারে।
আমরা সি++ এ কোডটা ইমপ্লিমেন্ট করতে পারি। শুরুতেই আমি লিংকড লিস্টের নোড ক্লাস ডিফাইন করবো:

```
struct Node {
    string key;
    string value;
    Node* prev;
    Node *next;

    Node() {
```

```
        prev = NULL;
        next = NULL;
    }

    Node(string _key, string _value) {
        key = _key;
        value = _value;
        Node();
    }
};
```

লিংকড লিস্টটা ইমপ্লিমেন্ট করাই এই কোডের কঠিন কাজ। মেইন লজিক সহজ হয়ে যাবে যদি আমরা লিংকড লিস্টের একটা ক্লাস তৈরি করে ফেলি। আমাদের লিংকড লিস্টের বামে নোড ঢুকানো আর লিস্ট থেকে নোড ডিলিট করার মেথড লাগবে।

```
struct DoublyList {
    Node* head;
    Node* tail;

    DoublyList() {
        head = new Node();
        tail = new Node();

        head->next = tail;
        tail->prev = head;
    }

    void erase(Node *curr) {
        curr->prev->next = curr->next;
        curr->next->prev = curr->prev;
    }

    void insertHead(Node* curr) {
        Node* temp = head->next;
        head->next = curr;
        curr->next = temp;
        temp->prev = curr;
        curr->prev = head;
    }

    string popTail() {
        Node* last = tail->prev;
        string key = last->key;
        last->prev->next = tail;
        tail->prev = last->prev;
        return key;
    }
};
```

আমাদের বেসিক স্ট্রাকচার রেডি, এবার ক্যাশটি ইমপ্লিমেন্ট করি:

```
class LRUCache {

private:
    int capacity;
    map<string, Node*> cache;
    DoublyList *list;

public:

    LRUCache(int _capacity) {
        capacity = _capacity;
        list = new DoublyList();
    }

    string get(string key) {
        if (cache.find(key) == cache.end()) {
            return "NOT FOUND";
        }

        Node *curr = cache[key];
        list->erase(curr);
        list->insertHead(curr);

        return curr->value;
    }

    void put(string key, string value) {
        Node* newNode = new Node(key, value);

        if (cache.find(key) != cache.end()) { //Found in hashmap
            Node *curr = cache[key];
            curr->value = value;

            list->erase(curr);
            list->insertHead(curr);
        } else {
            cache[key] = newNode;
            list->insertHead(newNode);

            if (cache.size() > capacity) {
                Node* lastNode = list->popLast();
                cache.erase(lastNode->key);
            }
        }
    }
};
```

কোডটা স্টেপ বাই স্টেপ ফলো করলেই বুঝে যাবে কিভাবে এটা কাজ করে। তুমি যদি প্র্যাকটিস করতে চাও তাহলে লিটকোডে নিজের কোড সাবমিট করতে পারো। LRU ক্যাশ খুবই কমন একটা ইন্টারভিউ কোশ্চেন।

বাস্তবে হয়তো তোমার কখনোই এটা ইমপ্লিমেন্ট করতে হবে না, বেশিভাগ ক্যাশিং মেকানিজমই LRU মেথড সাপোর্ট করে, যেমন রেডিস। ক্যাশের ডাটার আরো কিছু প্রোপার্টি থাকে, যেমন TTL

বা Time to live, এটা দিয়ে ডিফাইন করা হয় কতক্ষণ পর একটা ডাটা একাই এক্সপায়ার হয়ে যাবে।

তুমি চিন্তা করে বের করার চেষ্টা করতে পারো সেটা কিভাবে ইমপ্লিমেন্ট করতে হয়!

হ্যাপি কোডিং!