造輪子前請多想三秒
淺談 URL MASKING服務
的幕後 STACK

CLYANG@CSIE.IO

#WHOAMI?

- 96級碩班 (HSNG lab)
  - Lab 401SA
- csie.io co-funder
  - http://techccu.csie.io/2015/slides/lightning_terces.pdf
  - 還沒註冊記得趕快註冊啊！
- 一碰到前端相關事務就哭哭～
  - 本服務如果沒有Terces的大力協助以及Vexed, icross.cc, Brucehsu提供許多幫忙是不可能成真。
  - 最後依舊要感謝Vexed以及Terces提供快速有效的**前端技術諮詢**，才能讓網頁有最佳的呈現。

Before we start

只是希望你
多想三秒，
不是叫你都
用現成的!

請不要再**DS/ALGORTIHM**課的時候說：

老師，學長說用現成的又好又快

那...這三秒該想些什麼？

- 服務穩定度 (Service Stability)
  - 使用者想用時都可以獲得服務
- 服務可擴充度 (Service Scalability)
  - 突然被雷打到，服務瞬間爆紅，可否在短時間內擴充容量
- 服務維護性 (Service Maintainability)
  - 各套件是否容易監測，發生問題時是否容易回復

URL MASKING 是什麼?

把

HTTPS://
WWW.CS.CCU.EDU.T
W/~ABC106U/
PROJECT1/

變成

HTTPS://ABC.CSIE.IO

OKAY, 那URL MASKING該怎麼做呢？

- wildcard dns record 指到系上直接在做 virtual hosting的設定
  - *.csie.io , 將所有無法正解的dns record皆指向特定IP
- Pros
  - 簡單 + 相對好設定
  - 沒有多餘的overheads
- Cons
  - 已經畢業了，沒有權限 T_T

OKAY, 那URL MASKING該怎麼做呢？

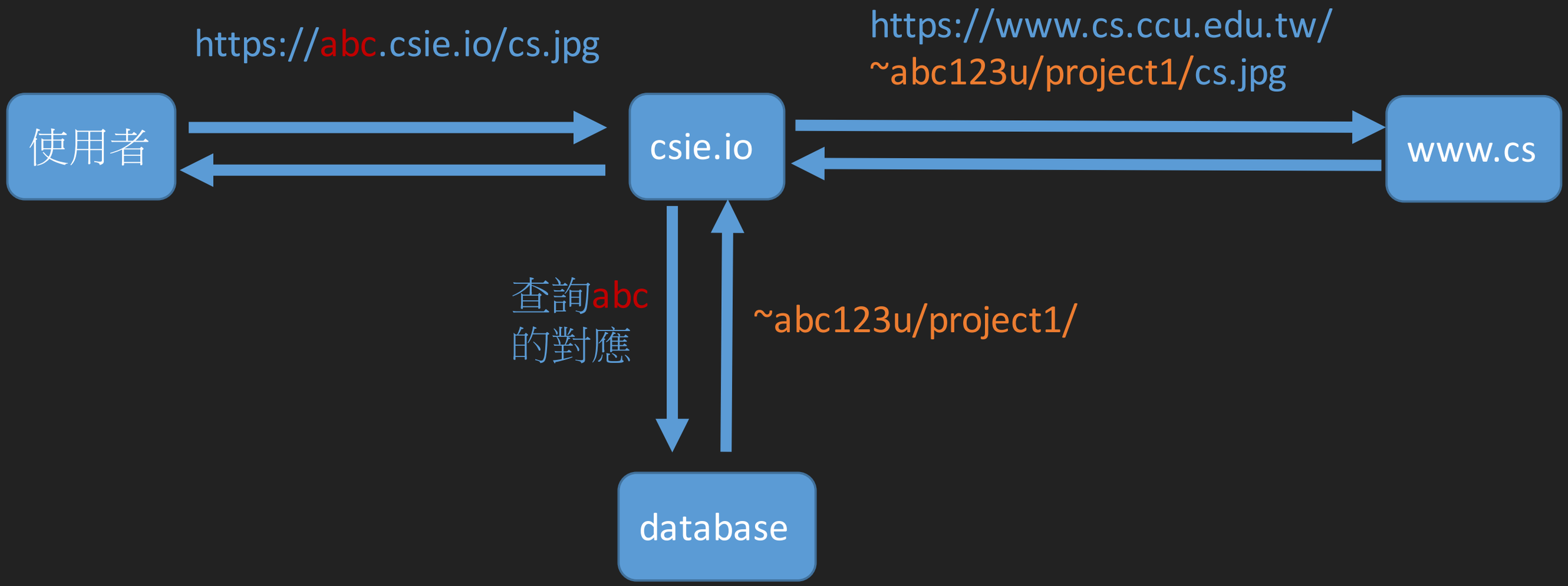
- wildcard dns record 指到csie.io，動態用iframe罩在整個網頁外面
- Pros
  - 簡單好實作
- Cons
  - 當頁面中有複雜的javascript時，會遭遇到跨domain而無法執行的問題
  - 使用者要手動改code，負擔太大

OKAY, 那URL MASKING該怎麼做呢？

- wildcard dns record 指到csie.io，使用reverse proxy + 動態url rewrite rules來做對應 (mapping)
- Pros
  - 使用者一秒設定好，開心打電動去
- Cons
  - 不管哪套web server，撰寫rewrite rules都痛苦萬分
  - 需處理多種例外，eg. request loops
  - overheads增加

OVERHEADS增加? WHY?

https://abc.csie.io/cs.jpg
https://www.cs.ccu.edu.tw/~abc123u/project1/cs.jpg

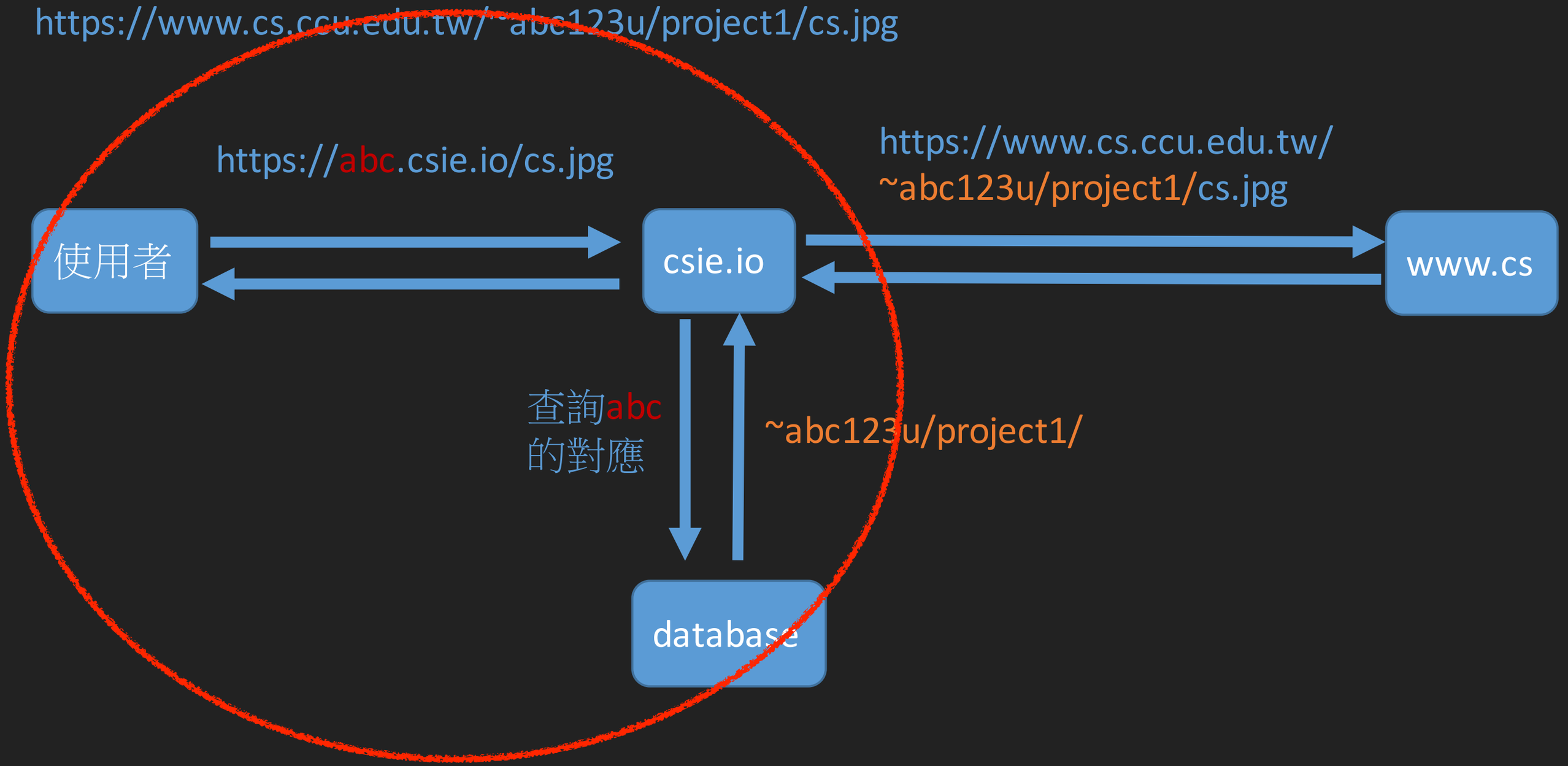


選擇使用OPENRESTY

- 由agentzh (章亦春) 開發，整合Nginx + Lua + Luajit
- Pros
  - Nginx+Lua是full asynchronous (non-blocking)
  - Luajit速度效率極佳，頗接近原生C/C++ code
  - 許多大型網站使用，每天處理數億PV的流量
  - 豐富的第三方套件: dns / websocket / redis / mysql
  - 方便的套件管理系統 opm (即將上線)

先來看看如前半段怎麼做

https://abc.csie.io/cs.jpg
https://www.cs.ccu.edu.tw/~abc123u/project1/cs.jpg



取出HOSTNAME及其對應

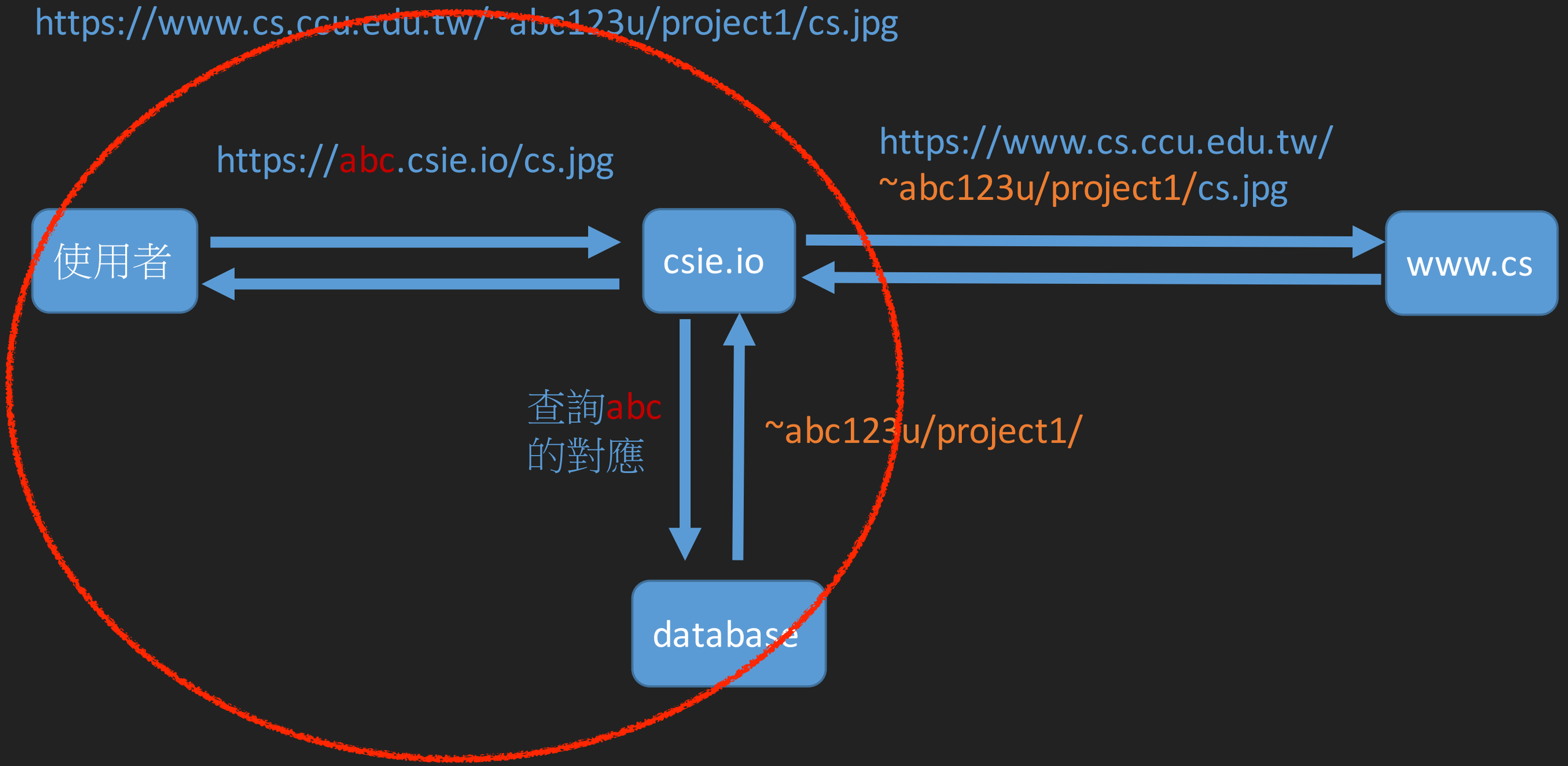
- abc -> ~abc123u/project1/

```
server {
    listen 443 ssl http2;
    server_name *.csie.io;
    more_set_headers 'Server: clyang-kerker/1.1';

    location ~ ^/ {
        set $prefix '';
        set $dsturi '';
        access_by_lua '
            local host = ngx.var.host:match("([^,]+).csie.io")
            -- 用取出的hostname查詢後端DB
        ';
    }
}
```

有沒有覺得有哪裡怪怪的？

https://abc.csie.io/cs.jpg
https://www.cs.ccu.edu.tw/~abc123u/project1/cs.jpg

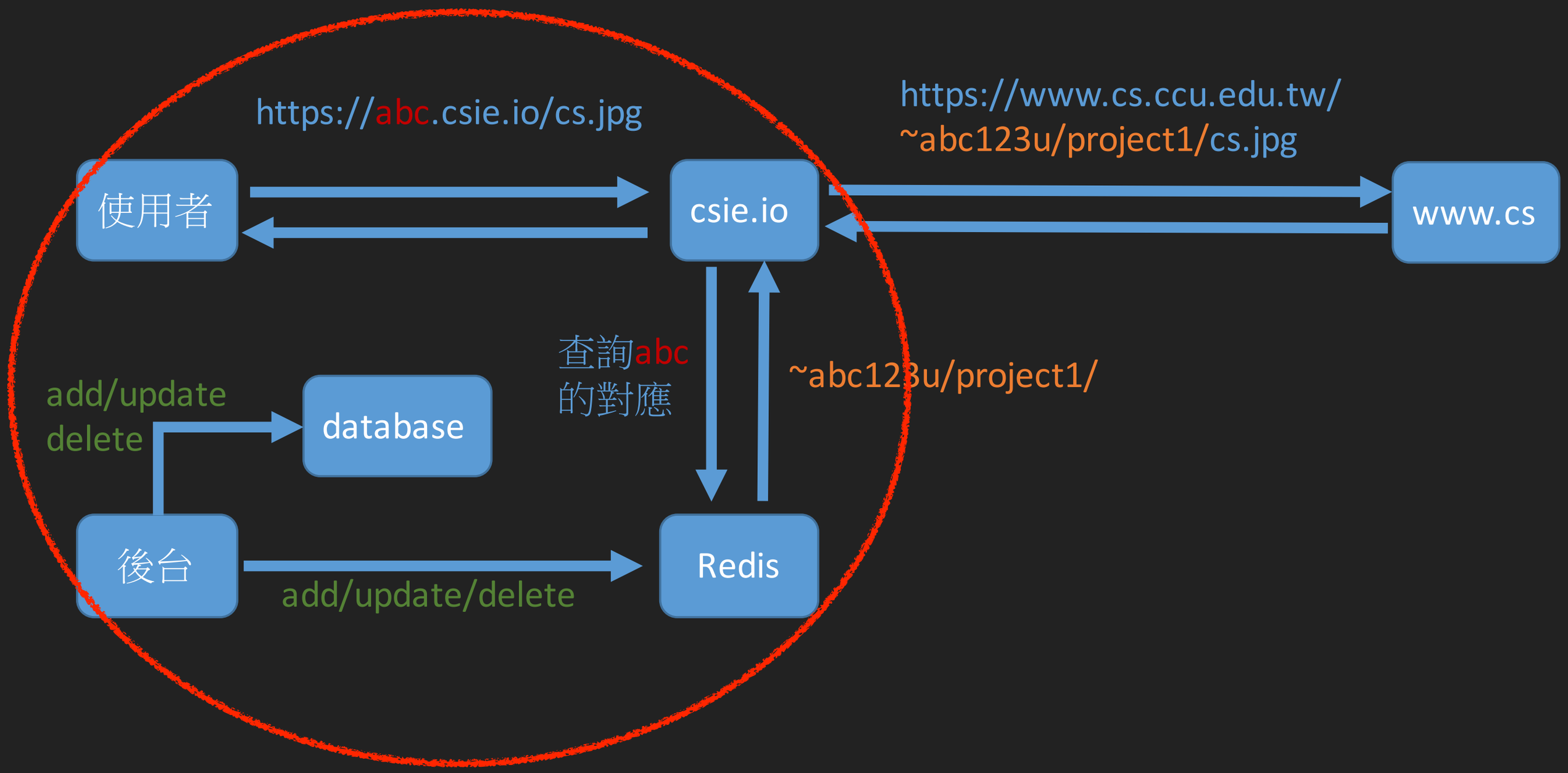


沒錯!! 每個REQUEST都去撈DB實在不明智

- 我們需要一個簡單、易用、速度夠快的key-value查詢機制
  - a1 => abc
  - a2 => def
- 所以我們引進了redis，why?
  - in-memory data store
  - 豐富的data structure
  - 可將memory中的data回存硬碟
  - 可作replication

所以架構變成這樣

https://abc.csie.io/cs.jpg
https://www.cs.ccu.edu.tw/~abc123u/project1/cs.jpg



讓我們來完成前半段的CODE

- redis使用unix socket連接
- 切勿使用lua socket，會讓整體效能嚴重下滑
- 請愛用openresty內建的cosocket

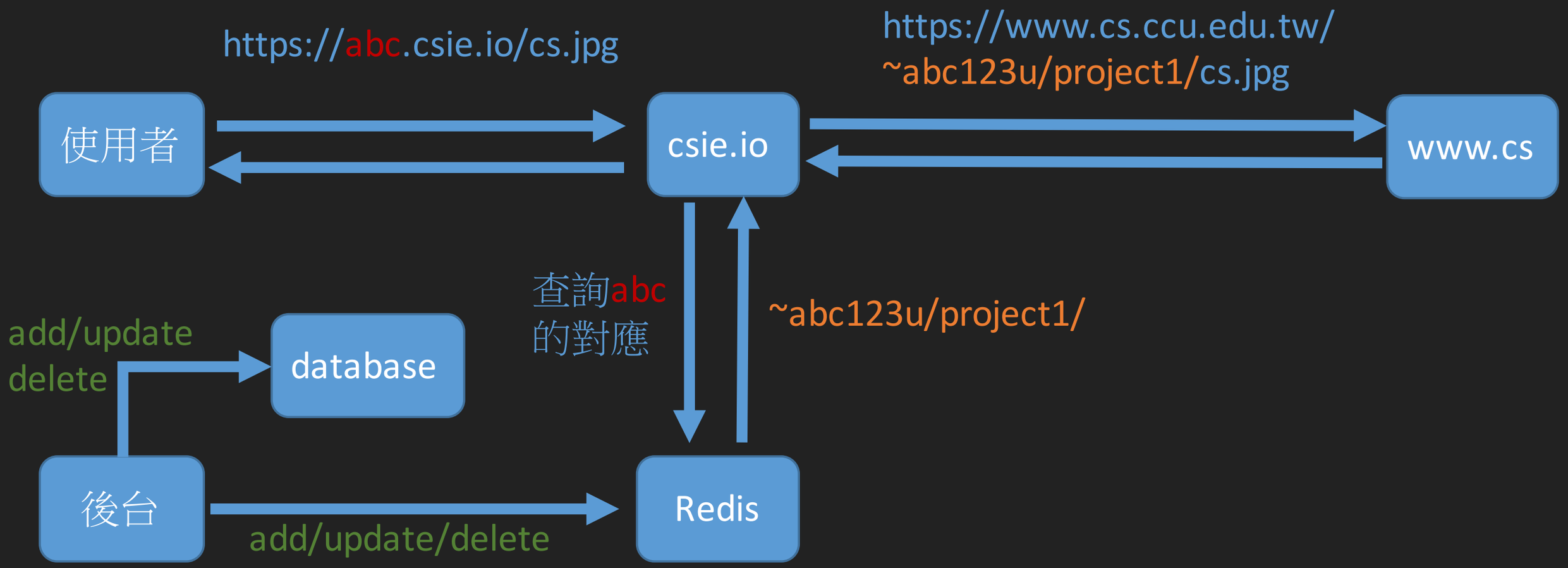
```
server {
    listen 443 ssl http2;
    server_name *.csie.io;
    more_set_headers 'Server: clyang-kerker/1.1';

    location ~ ^/ {
        set $prefix '';
        set $dsturi '';
        access_by_lua '
            local host = ngx.var.host:match("([^,]+).csie.io")

            local redis = require "resty.redis"
            local red = redis:new()
            local ok, err = red:connect("unix:/var/run/redis/redis.sock")
            red:select(1) -- 需處理連接錯誤
            local prefix, err = red:get(host)
            if prefix == ngx.null then
                ngx.log(ngx.ERR, "Unable  Redis prefix key: ", host)
                return ngx.exit(500)
            else
                i, j = string.find(ngx.var.request_uri, prefix)
                if  i ~= nil then
                    -- https://abc.csie.io/~abc123u/project1/xyz.html
                    dsturi = string.gsub(ngx.var.request_uri, prefix, "", 1)
                else
                    dsturi = ngx.var.request_uri
                end
                ngx.var.prefix = prefix
                ngx.var.dsturi = dsturi
            end
        ';
        proxy_pass https://www.cs.ccu.edu.tw$prefix$dsturi;
    }
}
```

前一頁短短一行就把後半段也搞定了！

https://abc.csie.io/cs.jpg
https://www.cs.ccu.edu.tw/~abc123u/project1/cs.jpg



BUT!! 人生最厲害就是這個BUT!!

使用者: ㄟ~用了後網頁開起來好慢啊

所以是哪邊慢了？一起來找找~

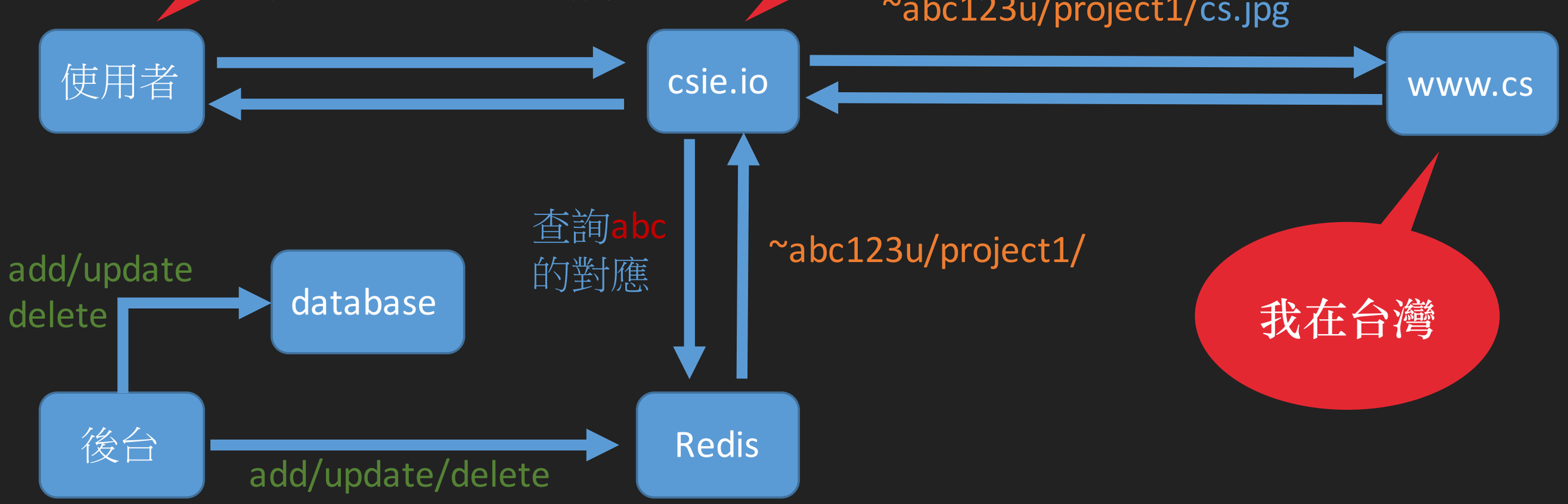
https://abc.csie.io/cs.jpg
https://w tw/~abc123u/project1

我在台灣

我在新加坡

.csie.io/cs.jpg

www.cs.ccu.edu.tw/~abc123u/project1/cs.jpg



我們需要快取，將不容易變動的檔案都預存一份

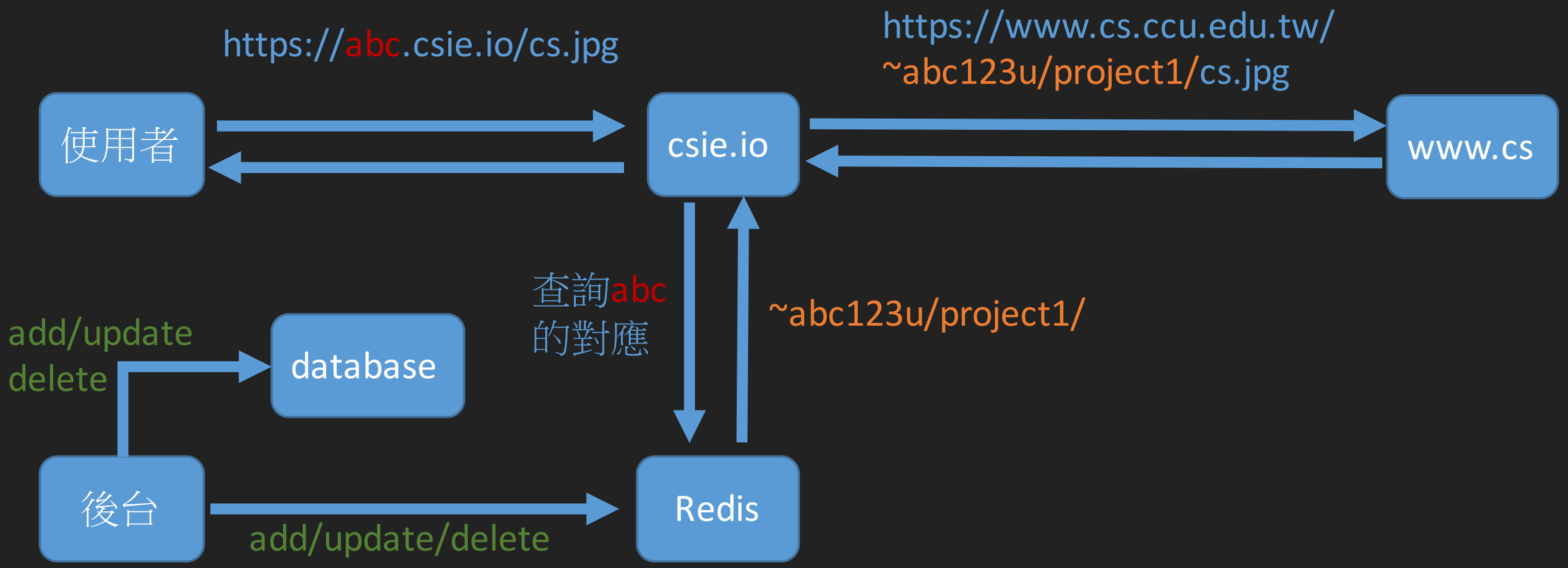
- 選擇使用Apache Traffic Server (ATS)
  - 是Yahoo!使用的YTS的公眾版，用戶夠多 eg. Akamai
  - 三大設定檔, {remap, cache, records}.config
- remap.config
  - regex_map http://(.*)/ http://www.cs.ccu.edu.tw/
- records.config
  - CONFIG proxy.config.cache.ram_cache.size INT 536870912
- storage.config
  - /var/cache/trafficserver 768M

我們需要快取，將不容易變動的檔案都預存一份

- cache.config中只需兩行
  - url_regex=www.cs.ccu.edu.tw/.+\.(js|css)(\?.+)?$ ttl-in-cache=1h
  - url_regex=www.cs.ccu.edu.tw/.+\.(jpg|png|gif)(\?.+)?$ ttl-in-cache=1h
- 然後記得更新一下Nginx設定檔
  - proxy_pass http://127.0.0.1:8080$prefix$dsturi;

所以網頁只要有點擊過，之後都不用走後半段了！

https://abc.csie.io/cs.jpg
https://www.cs.ccu.edu.tw/~abc123u/project1/cs.jpg



孩子，你以為使用者會放過你嗎？

使用者:怎麼動態產生的頁面都不會更新？

原來ATS預設是AGGRESSIVE CACHE

- 只好再更新一下cache.config
  - url_regex=www.cs.ccu.edu.tw/.+/nocache/.+ ttl-in-cache=1s
  - dest_domain=cs.ccu.edu.tw suffix=php action=never-cache
  - url_regex=www.cs.ccu.edu.tw/.+\.(js|css)(\?.+)?$ ttl-in-cache=1h
  - url_regex=www.cs.ccu.edu.tw/.+\.(jpg|png|gif)(\?.+)?$ ttl-in-cache=1h

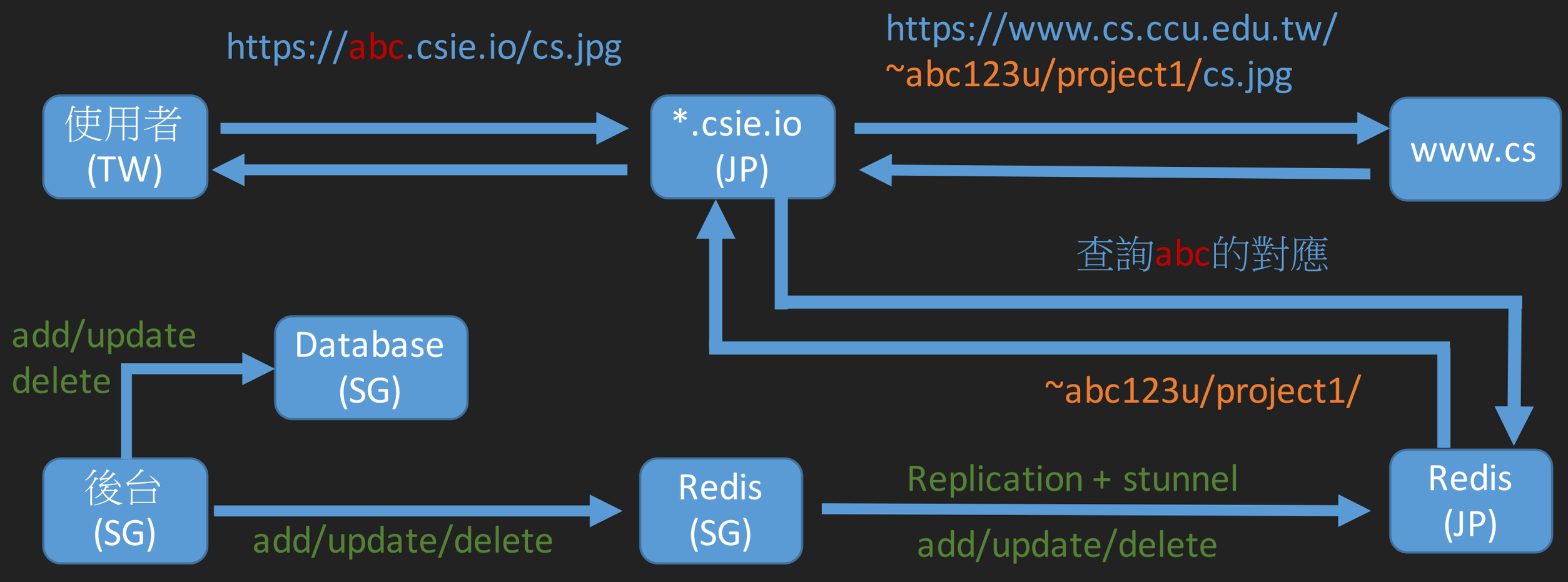
RULE #1: 永遠別認為使用者會就此滿意

使用者:連新加坡很慢捏～不能搬近一點嗎？

還好平常有做好事，剛好可以得到一台在日本的LINODE

- 所以我們的架構又進化了

https://abc.csie.io/cs.jpg
https://www.cs.ccu.edu.tw/~abc123u/project1/cs.jpg



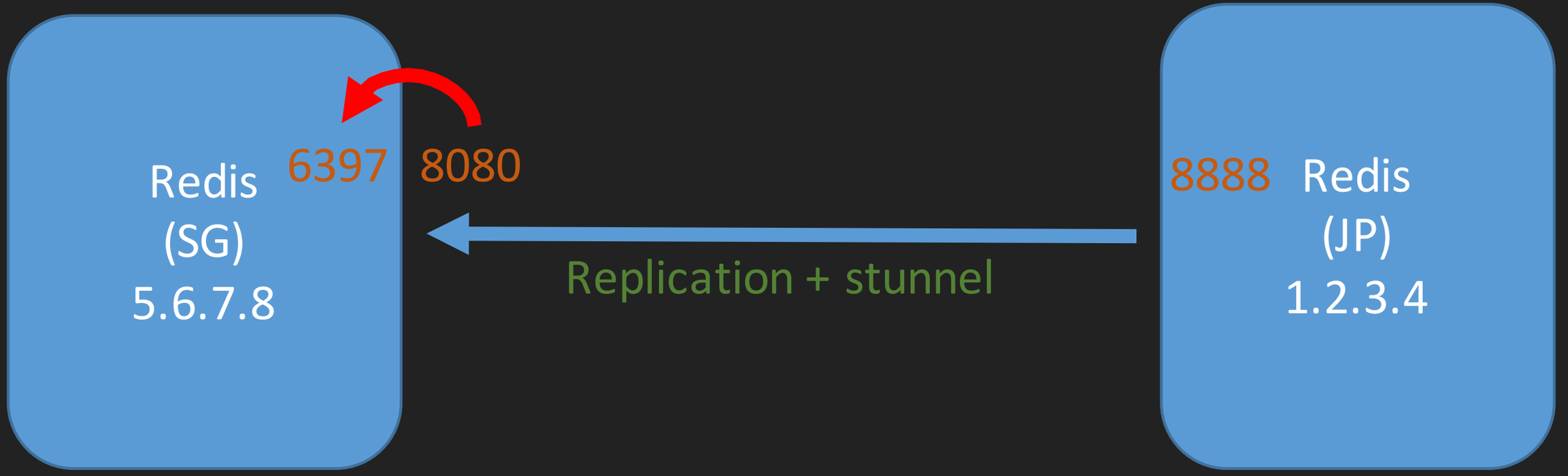
REDIS REPLICATION + STUNNEL

- redis做replication相當容易，只需在slave的redis.conf指定master的IP以及連接port即可
  - slaveof 127.0.0.1 8888
  - BUT!!! master與client之間是沒有加密的！
- 所以我們使用stunnel來建立master與slave之間的安全通道

REDIS REPLICATION + STUNNEL

client = no
[redis]
accept = 5.6.7.8:8080
connect = 127.0.0.1:6379

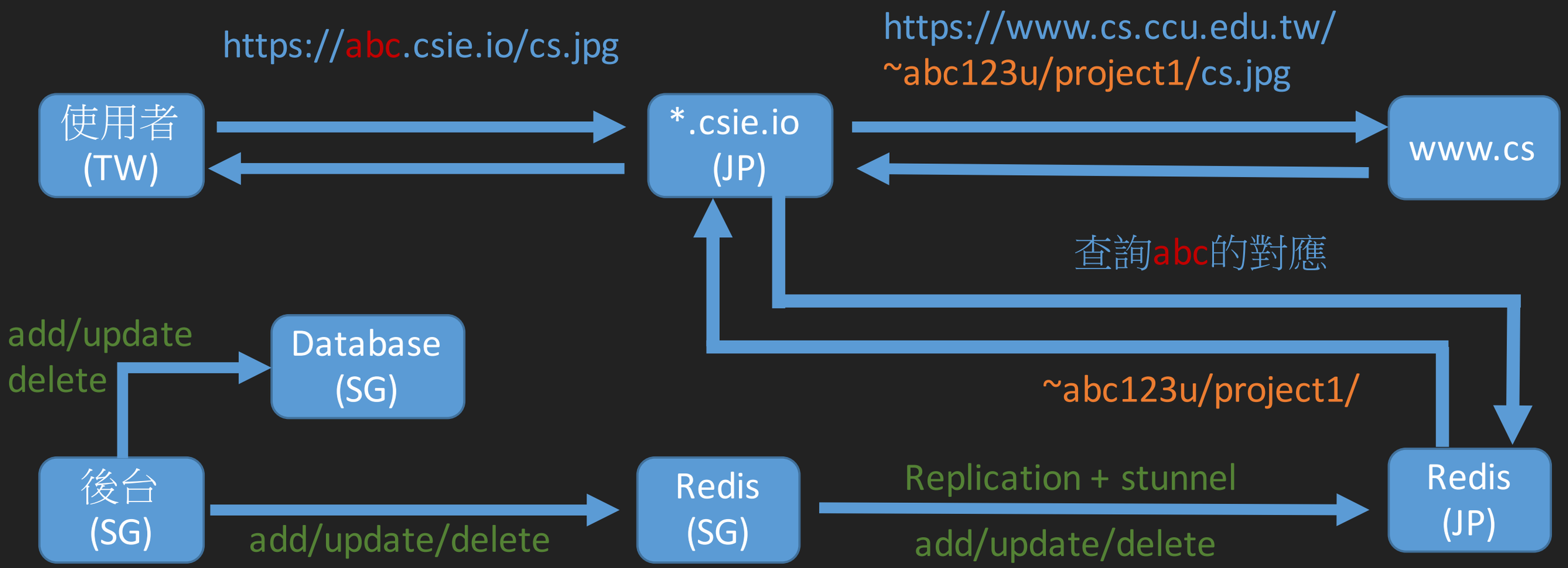
client = yes
[redis]
connect = 5.6.7.8:8080
accept = 127.0.0.1:8888



你覺得還有可以改進的地方嗎？

讓我們再看一次架構

https://abc.csie.io/cs.jpg
https://www.cs.ccu.edu.tw/~abc123u/project1/cs.jpg



REDIS快雖快，但是頻繁的被存取看起來還是不太順眼啊

- 一旦有個request過來，我們有信心認為，之後的幾秒一定會有數個同個hostname的request過來
  - https://abc.csie.io/jquery.js
  - https://abc.csie.io/main.css
- lua_shared_dict讓你可以直接將data以key-value的型態存在Nginx的shared memory zone中，讓各個worker直接存取
- 因此我們從redis讀到資料後，便短暫的存了一份在shared dict中，在這幾秒內的requests都會從shared dict中取得
- 根據觀察，在某些網頁結構下，減少約20%的CPU loading

```
lua_shared_dict uridict 1m;
server {
    location ~ ^/ {
        set $prefix '';
        set $dsturi '';
        access_by_lua '
            local host = ngx.var.host:match("([^,]+).csie.io")

            local redis = require "resty.redis"
            local red = redis:new()
            local ok, err = red:connect("unix:/var/run/redis/redis.sock")
            red:select(1) -- 需處理連接錯誤

            local uris = ngx.shared.uridict
            local prefix, flags = uris:get(host)
            if prefix == nil then -- 殘念shared dict找不到，來去問redis
                local redis = require "resty.redis"
                local red = redis:new()
                local ok, err = red:connect("unix:/var/run/redis/redis.sock")
                red:select(1)
                prefix, err = red:get(host)
                uris:set(host, prefix, 5) -- 把結果存進shared dict, 存個五秒即可
            end
            -- 後面處理的code跟前面一樣
        ';
        proxy_pass http://127.0.0.1:8080$prefix$dsturi;
    }
}
```

加油，最後一張投影片了！前面睡著沒關係，至少這頁要看

- 造輪子前不妨多survey現有解法，真的找不到再自己來也不遲
  - 尤其是直接面對使用者的production環境
- 雲端聽起來很fancy，但卻不是完全worry free，糟糕的架構下，也很難快速的 (或是完全無法) scale up
  - 基本功是重要的，很多架構在業界都是標準做法，差別只在於你是否有能力將這些做法套在你的專案上
- 多看些國內外架構類的文章，看多了就會慢慢有fu
  - 我知道這超級老套，但這真的是事實
- 最後，openresty跟redis值得花點時間去研究
  - 如果你真的很忙，最起碼把redis玩一玩吧！

REFERENCE

- OpenResty最佳实践 (使用openresty必看)
  - https://moonbingbing.gitbooks.io/openresty-best-practices/content/lua/main.html
- 本文件使用到的CC授權圖片網址
  - https://www.flickr.com/photos/paolofefe/6222744957/
  - https://www.flickr.com/photos/sebastiandooris/12293954916/

THANK YOU