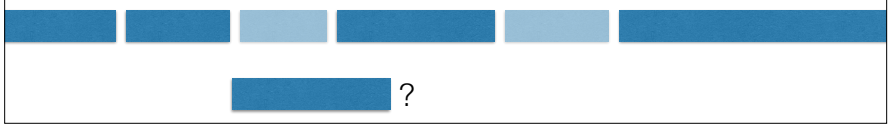
Operating Systems

Memory: caching

Last time…

Last time…

- How do you minimize memory fragmentation?



?

When does free memory need to be managed?

- OS level
  - (early on) in segmented memory systems
  - to manage kernel memory
- program level
  - in languages such as C and C++: malloc/free, new/delete to manage the heap
  - Java has **new** but no **delete**, why?
  - when you know the usage pattern of your application and handling it yourself is faster: eg memcached

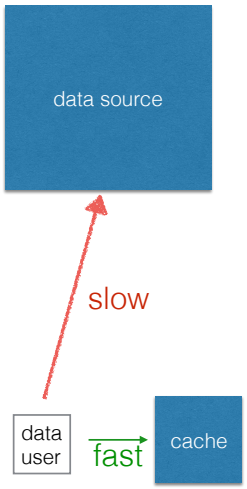
Recap: memory management

- **efficient and flexible memory use**
  - paging and segmenting system
  - multi-level page tables -> sparse addressing
  - shared code
- **security and isolation**
  - branch and bound
  - read/write access
- **speeding up data retrieval**
  - caching: TLB, virtual and physical caches
- **managing free memory (depends on the actual use)**
  - not a problem if chunks are uniform size
  - techniques: coalescing, hole selection, buddy system, slab allocation
  - language support for managing heap: malloc/free, garbage collection

Caching

- why? (main idea, examples)
- how? (challenges)

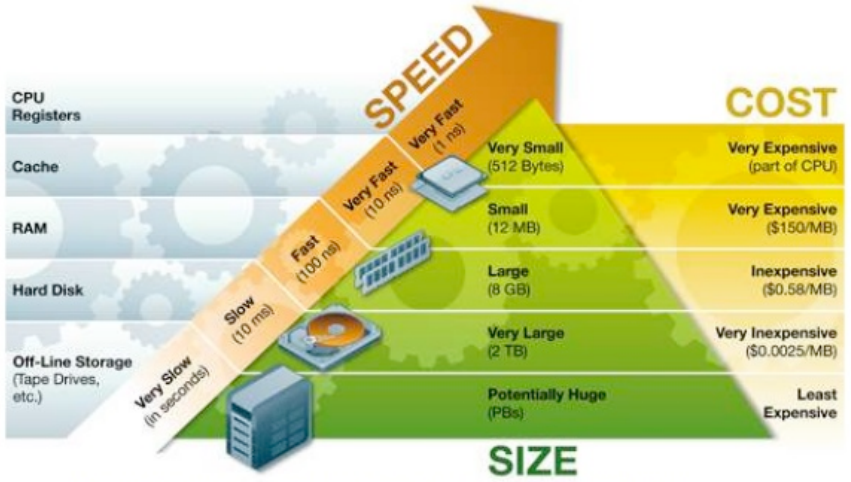
Why? caching speeds up data lookups



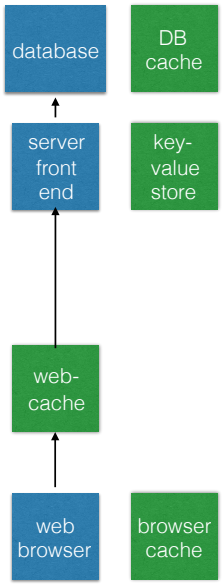

```
get (data):
    if data in cache:    cache hit!
        return data
    else:                cache miss
        get it from data source
        save data to cache
```

average fetch time =
Pr(hit) x (latency from cache) +
Pr(miss) x (latency from source)

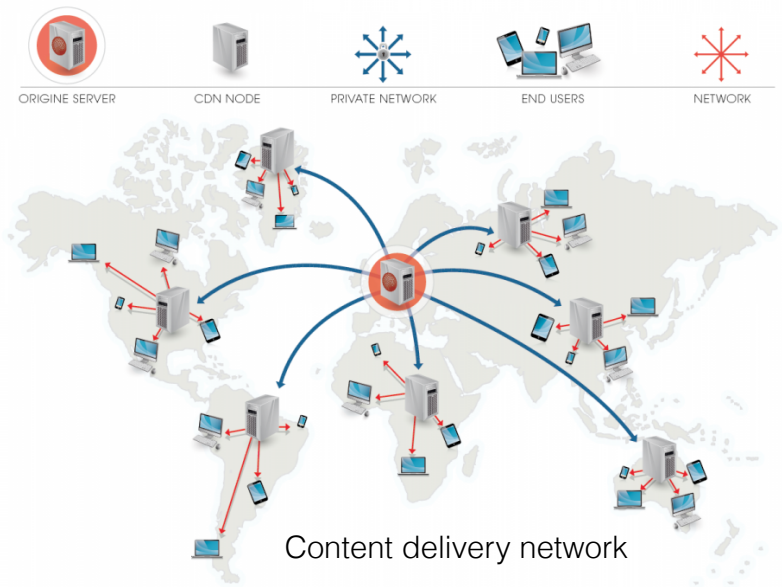
Extended Memory Hierarchy



Source: http://www.ts.avnet.com/uk/products_and_solutions/storage/hierarchy.html



caching in a network

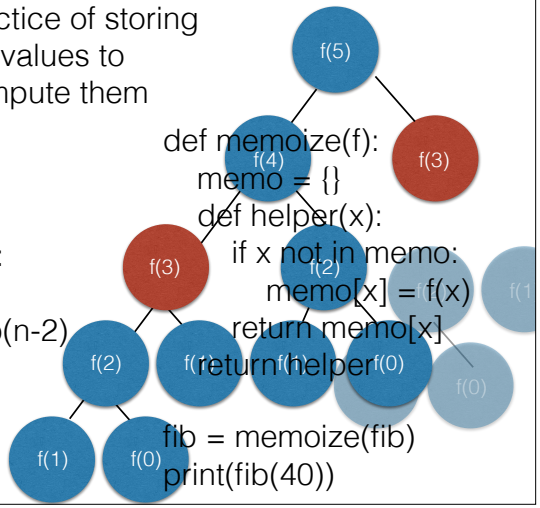


Source: https://4cornermedia.com/wp-content/uploads/2016/03/CDN-1.png

Memoization

in algorithms, the practice of storing previously computed values to avoid having to recompute them

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    return fib(n-1) + fib(n-2)
```



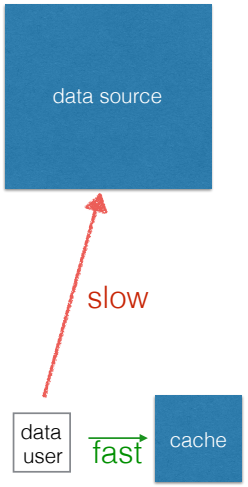

```
def memoize(f):
    memo = {}
    def helper(x):
        if x not in memo:
            memo[x] = f(x)
        return memo[x]
    return helper

fib = memoize(fib)
print(fib(40))
```

Examples of caching

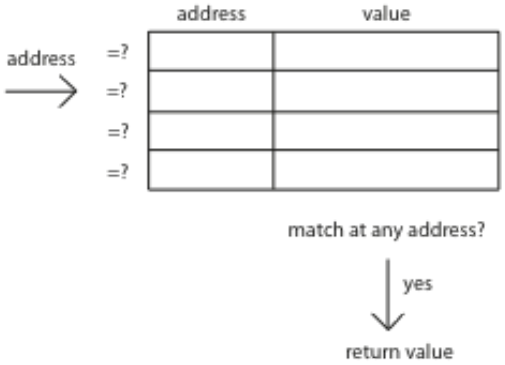
- hardware level:
  - TLB: cache for addresses
  - L1-L3: data caches
- OS level: demand paging
- software level: memoization
- network level
  - browser, webcache, server cache, database cache
  - content delivery networks

Caching: challenges

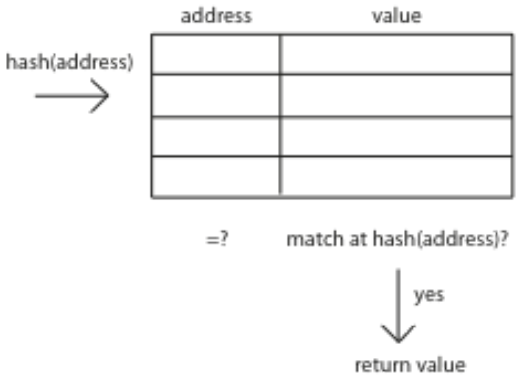
- efficient lookups (today)
- cache coherence (a little today, mostly later)
- replacement policy (next lecture)



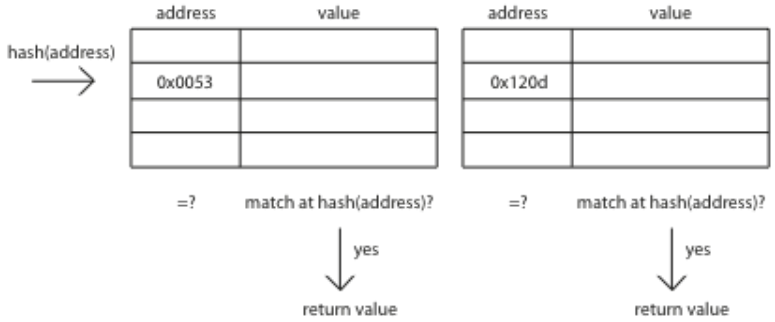
Lookups at the hardware level: fully associative cache



Lookups at the hardware level: direct mapped



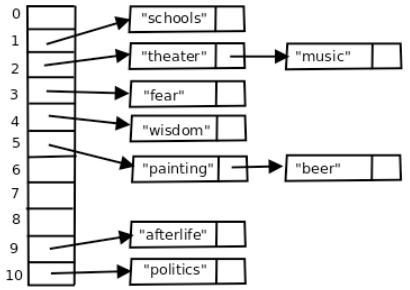
Lookups at the hardware level: set associative



Hash tables

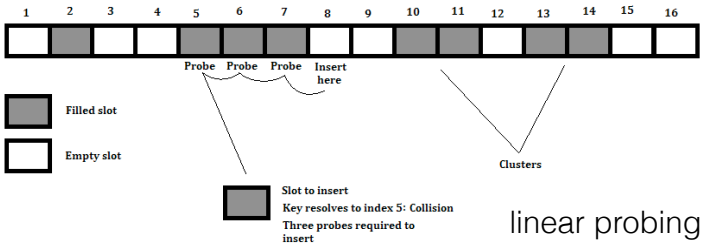
- O(1) lookups on average

How do you resolve collisions?



hash chaining

How do you resolve collisions?

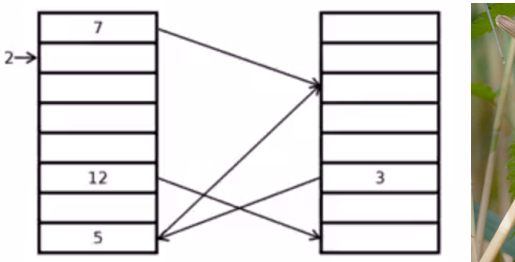


linear probing

Open addressing: store values directly in the array

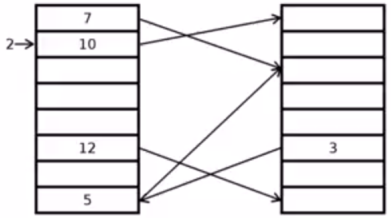
Cuckoo hashing

Easy Insertion



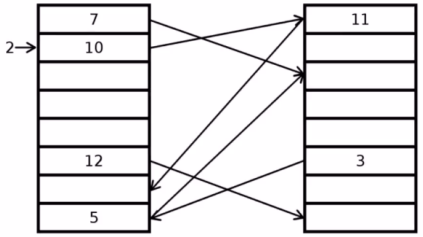
Cuckoo hashing

Inserting With 1 Conflict



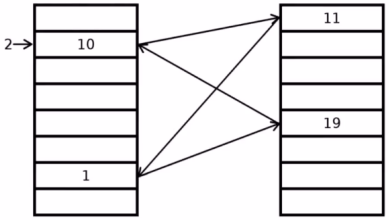
Cuckoo hashing

Inserting With 2 Conflicts



Cuckoo hashing

Infinite Loop



Cuckoo hashing

- Pagh and Rodler, 2001
- expected amortized O(1)
- O(1) lookups in worst case, rather than expected case

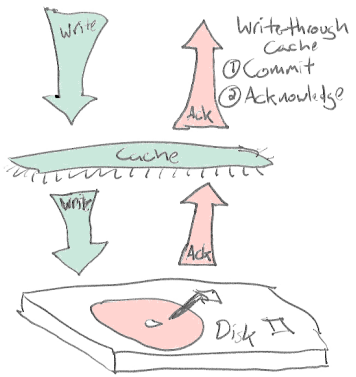
Efficient lookups

- CPU cache
  - fully associative cache
  - direct mapped cache
  - set associative cache
- software: hash table
  - hash chaining
  - open addressing: linear probing, **cuckoo hashing**

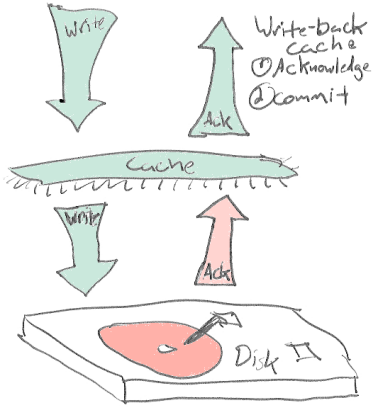
Cache coherence

- how to detect and repair stale data?

2 writing policies



write-through



write-back or write-behind

Source: http://blog.fosketts.net/2010/10/07/4-horsemen-cache/

The problem with multiple caches

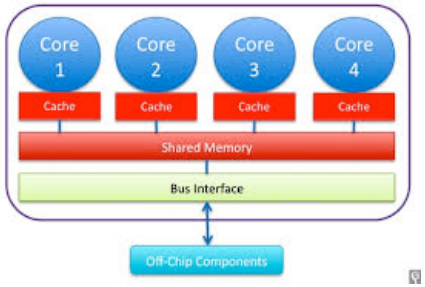
Alice

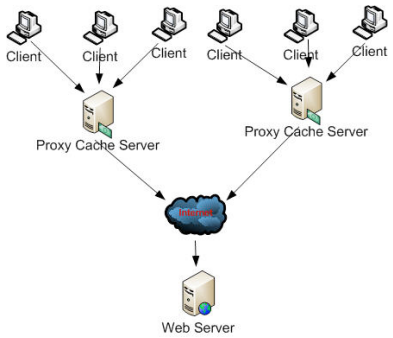
Barbara

Charlie

What time should we go to the movies?

Caching of shared memory

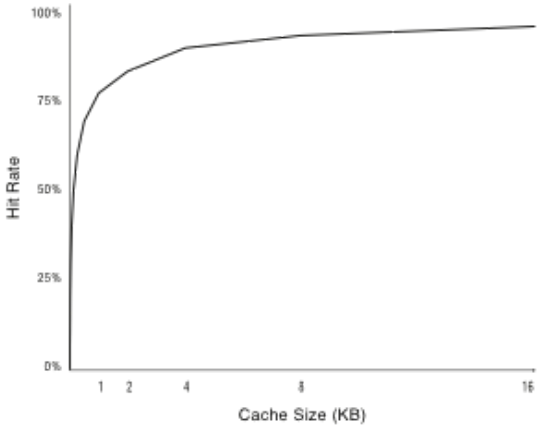




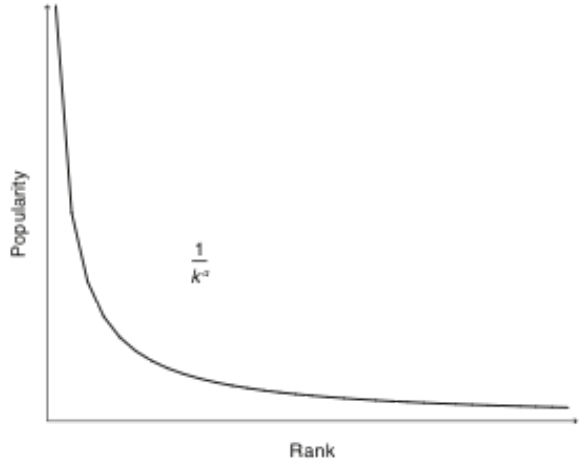
Sources: http://www.codeproject.com/KB/web-cache/ExploringCaching/cache_array.jpg
http://embedded-computing.com/articles/conquering-concurrency-bugs-multicore-systems/

Working set

the amount of memory a process requires at any given time

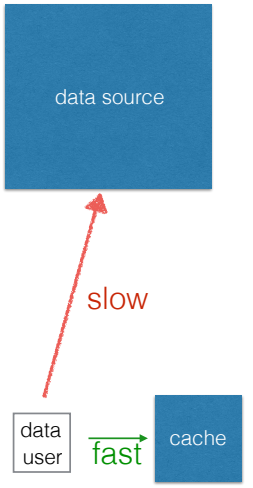


Zipf distribution



Recap

- caching speeds up data lookups if the data is likely to be re-requested again
- data structures for O(1)-lookup
  - set-associative (hardware)
  - hash tables (software)
- cache coherence
  - with 1 cache: buffer writes back to memory
  - with multiple caches: things get complicated
- next time: replacement policy



Acknowledgements

- http://ospp.cs.washington.edu/slides.html
- cuckoo hashing explanation: Bridget Howell, Joe Whitney: https://www.youtube.com/watch?v=OBuGqu2d4v4