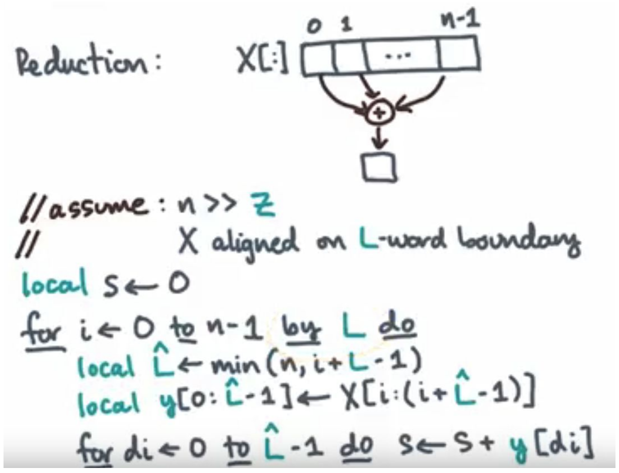
Lesson 3-4 Cache-Oblivious Algorithms

In a cache aware algorithm, the value of L is determined by the cache size.

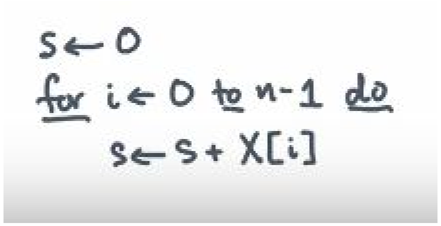
$Q_{aware}(n;z;L) = \theta(n/L)$



Reduction: X[:]

```
//assume : n >> Z
//           X aligned on L-word boundary
local s ← 0
for i ← 0 to n-1 by L do
    local $\hat{L}$ ← min(n, i+L-1)
    local y[0:$\hat{L}$-1] ← X[i:(i+$\hat{L}$-1)]
    for di ← 0 to $\hat{L}$-1 do s ← s + y[di]
```

If the algorithm makes no reference to fast memory or its parameters (z, L) … then the algorithm is oblivious (to fast memory).



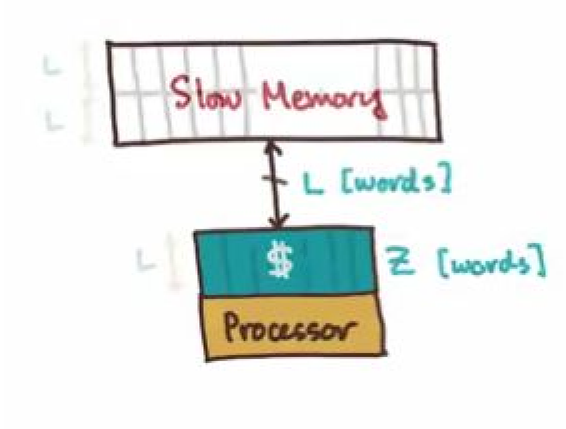
```
s ← 0
for i ← 0 to n-1 do
    s ← s + X[i]
```

$Q_{oblivious}(n;z;L) = O(n/L)$

There are two questions that need to be asked ….

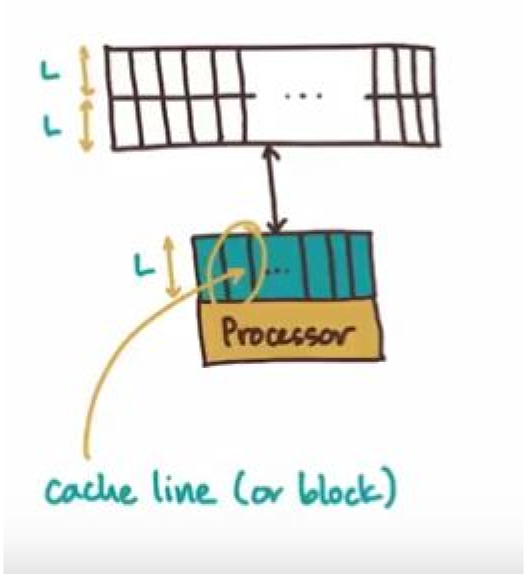
1. How do we model automatic fast memory?
2. Can an oblivious algorithm match I/O performance of an ‘aware’ algorithm?

The Ideal-Cache Model



This is the ideal cache model that will be used in the lesson.

The L is the same as the transfer size.



In a program loads and stores are issued for the slow memory address.
For example:
The program wants to load a value from memory a.
First it checks to see if the value of a is in fast memory.
If it is, it just uses it this is a cache hit.

A cache miss occurs when the value is not in the cache. Then it must be retrieved from slow memory and a copy is stored in fast memory.

The hardware must store an entire cache line of data, so other memory values around the desired memory location will also be retrieved and stored from memory.

For a store instruction:
If the value is in the cache, it is a cache hit. The value will be updated in the cache and stored in slow memory.

If the store value is not in the cache, this is a cache miss.
For a cache miss the entire memory line must be transferred from slow memory to the cache.

<u>The ideal cache is fully associative.</u> This means the transfer from slow memory is allowed to go to any line in the cache.
At some point, the cache will be full. So to load a new line, an old line will have to be evicted from the cache.
If the line that is to be evicted has not been written to memory, it is ‘dirty’. A dirty line must be written before it can be evicted, this is called a store-eviction.

Optimal replacement → evict line or block accessed furthest in the future.

Summary of Ideal Cache Model

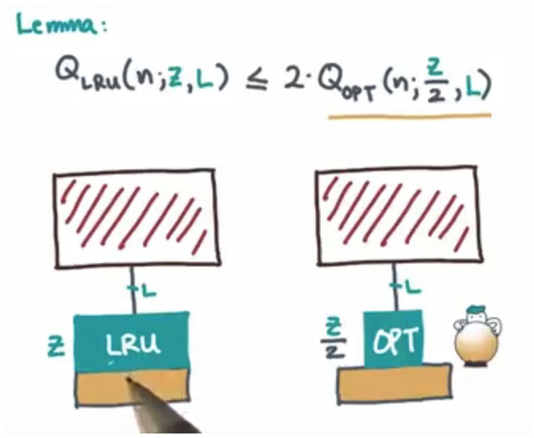
1. Program issues load and store ops
2. Hardware manages Z/L cache lines
3. Slow memory-cache transfers in lines or blocks of size L
4. Accessing a value in cache = cache hit,otherwise it is a cache miss
5. Cache is fully associative
6. Assume optimal replacement policy

Q(n;Z,L) = # misses + store evictions

LRU - least recently used
This replacement policy looks for the least recently used address to evict from the cache.
LRU will cause more evictions that optimal replacement.

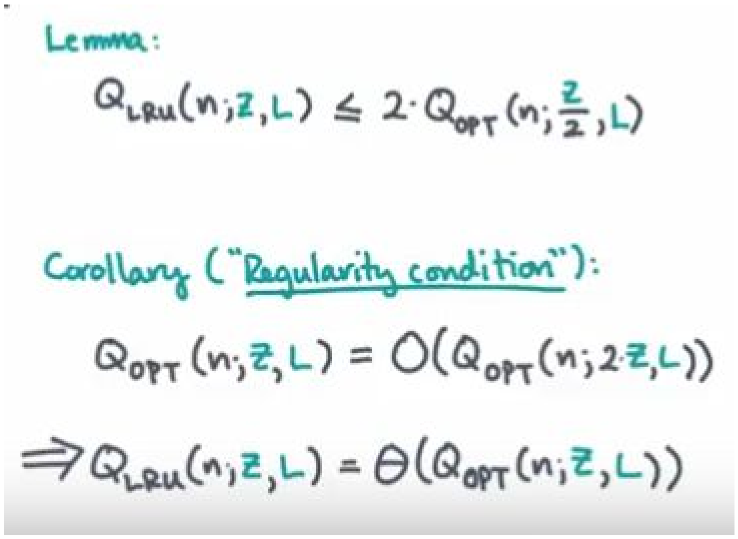
How ideal is the ideal cache?
Key assumptions:

1. optimal replacement
2. two-levels of memory
3. Automatic replacement
4. fully associative



Take a cache model that uses a LRU policy. Compare it to a cache that uses an optimal replacement policy that has ½ the cache size.

This lemma can be used in the following way:
Design an algorithm on an ideal model with optimal replacement. It should be asymptotically close to a model with an LRU policy and twice the cache.


Lemma:
$Q_{LRU}(n;Z,L) \leq 2 \cdot Q_{OPT}(n;\frac{Z}{2},L)$
Corollary ("Regularity condition"):
$Q_{OPT}(n;Z,L) = O(Q_{OPT}(n;2Z,L))$
$\Rightarrow Q_{LRU}(n;Z,L) = \Theta(Q_{OPT}(n;Z,L))$

If you design an algorithm for an ideal model, if the misses are regular, it will perform in a similar fashion to the ideal model.

The Tall-Cache Assumption
The cache should be taller (with regards to the number of lines - z) than it is wide (the size of the lines, L).
This means an efficient algorithm might be linked to the choice of data structure.