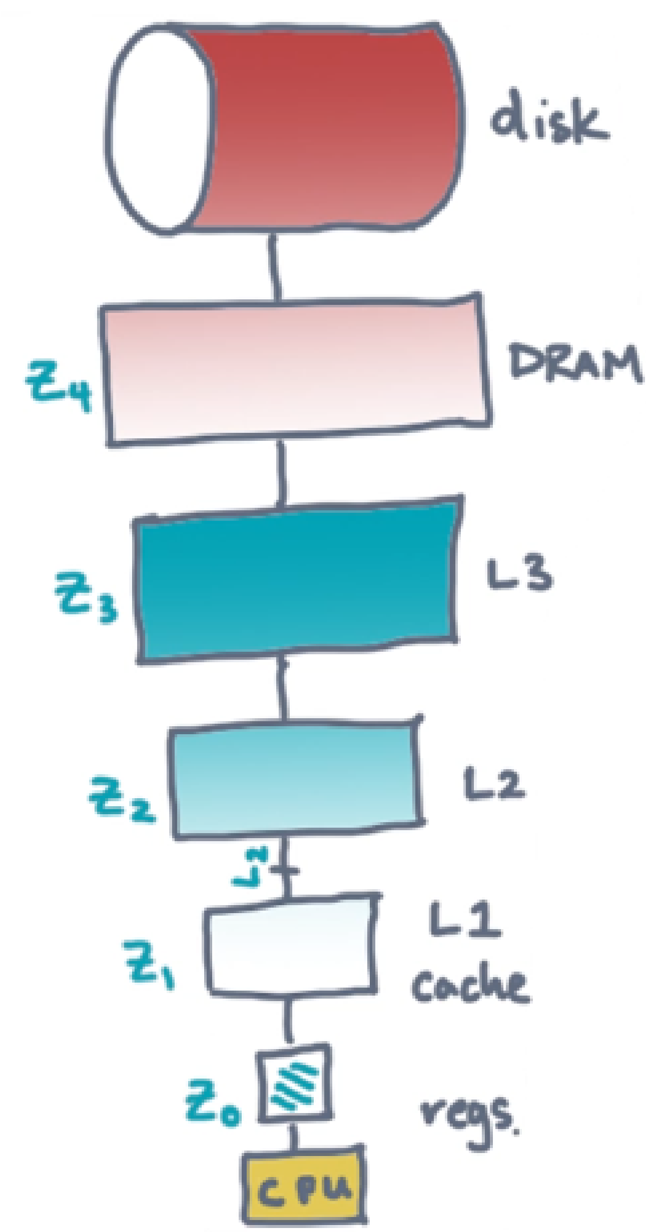
# Basic Model of Locality

## Introduction

1. Real machines have memory hierarchies
   - As we get closer to the processor, memory gets faster but also smaller
2. Usual way of analyzing algorithms doesn't consider memory hierarchy
   - Need to consider memory for best performance




Memory Hierarchy

## A First, Basic Model

1. von Neumann architecture has a processor connected to main memory (slow)
   - In between processor and main memory, there's a faster memory
     - The size of this memory is Z words
2. Rules:
   - Local data rule: Processor may only compute on data in fast memory
   - Block transfer rule: Slow-fast transfers in blocks of size L (words)
     - Can't load a single address, need to consider how data is aligned
3. Costs:
   - Work: W(n) = # of computation operations
   - Transfers: Q(n;Z,L) = # of L-sized slow-fast transfers (loads/stores)
     - Consider this the I/O complexity
4. Example:
   - Reduction (sum all elements in an array)
     - W(n) is greater than or equal to n-1 additions (O(n))
     - Q(n;Z,L) is greater than or equal to ceil(n/L) transfers (O(n/L))
   - I/O complexity doesn't depend on Z
     - Doesn't reuse data, which is bad

## Two Level Memories

1. Which of the following pairs are examples of two-level (slow+fast) memories?
   - Hard disk + main memory (true)
   - L1 cache + CPU registers (true)
   - Tape store + hard disk (true)
   - Remote server RAM + local server RAM (true)
   - The Internet + your brain (true)

## Alignment

1. How many transfers are necessary in the worst case, assuming nothing about alignment?
   - Q(n;Z,L) is less than or equal to ceil(N/L) + 1
   - Suppose n=4, L=2
     - If n is aligned on a word boundary, we need two transfers (ceil(n/L))
     - If n is not aligned on a word boundary, we need one extra transfer
   - Typically ignore this, especially when n » L

## Minimum Transfers to Sort

1. Give a simple (trivial) lower bound on the asymptotic number of transfers when sorting an array of elements with a comparison-based algorithm
   - W(n) = O(n * log(n))
   - Q(n;Z,L) = O(ceil(n/L))
     - Must read each element at least once, so the algorithm is bounded by ceil(n/L)
   - Actual answer is (n/L)log(n/L)/log(Z/L)

## Minimum Transfers to Multiply Matrices

1. C = MATMUL(A, B) where A, B, C are n x n
2. Give a simple (trivial) lower bound on the asymptotic number of transfers
   - W(n) = O(n^3) (non-Strassen)
   - Q(n;Z,L) = O(ceil(n^2/L))
     - Must touch all n^2 elements divided by the block size
   - Tighter lower bound is O(n^3/L/sqrt(Z))

## I/O Example Reduction

1. Consider the example of summing all elements in an array (reduction)

```
int s = 0; // local
for(int i = 0; i < n-1; i++)
    s = s + x[i];
```

2. Written more explicitly in terms of transfers:

```
int s = 0; // local
for(int i = 0; i < n-1; i++)
{
    int Lhat = min(n, i+L-1); // local, handle special case
    int y[0:Lhat-1] = X[i:(i+Lhat-1)]; // Lhat <= L
    for(int j = 0; j < Lhat - 1; j++)
        s = s + y[j]
}
```

2. Observations:
   - Painful, but clear
   - Yes, caches exist, but aren't sufficient to guarantee high performance

## Matrix Vector Multiply

1. Consider a matrix-vector multiply algorithm y = A * x
   - A is arranged in column-major order (A[i,j] = A[i + j * n])
   - i is number of rows, j is number of columns
2. Which of the following does fewer transfers?
   - Algorithm A:

   ```
   for(int i = 0; i < nrows; i++)
    for(int j = 0; j < ncols; j++)
        y[i] += A[i,j] * x[j]
   ```

   - Algorithm B:

   ```
   for(int j = 0; j < ncols; j++)
    for(int i = 0; i < nrows; i++)
        y[i] += A[i,j] * x[j]
   ```

3. Assumptions:
   - Z = 2n + O(L)
   - L divides n (L | n)
   - x, y, and A are aligned on word boundaries (L)
     - Don't need to worry about alignment issues
4. Algorithm B does fewer transfers because it iterates over rows in the inner-most loop
   - Algorithm A: Q(n;Z,L) = 3 * n / L + n^2
   - Algorithm B: Q(n;Z,L) = 3 * n / L + n^2 / L
5. In the sequential model, these algorithms look identical
   - Consider a fully-associative cache; will this solve the issue with algorithm A

## Algorithmic Design Goals

1. Work optimality: Two-level algorithm should do the same asymptotic work as the RAM algorithm
   - W(n) = O(W'(n))
2. High computational intensity: Ratio of work to words transferred
   - Maximize I(n;Z,L) = W(n) / ( L * Q(n;Z,L) )
   - Intensity has units of operations per word
   - Measures data reuse (more operations per words in fast memory)
     - Can't sacrifice work optimality in favor of intensity

## Which is Better?

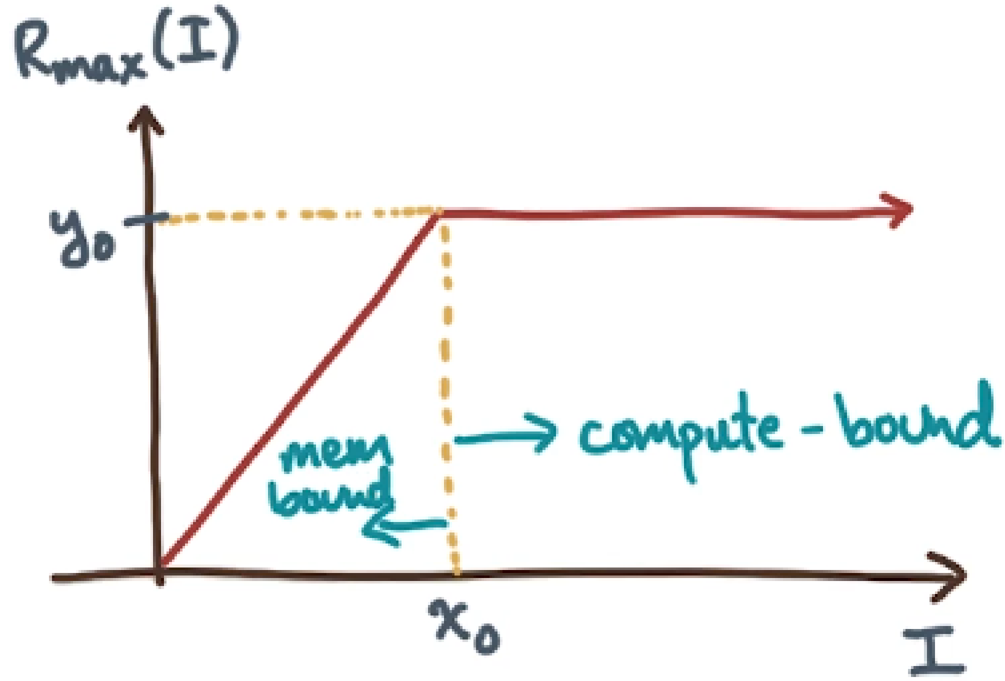
1. Consider the following algorithms:
    - Algorithm 1:
        - W1(n) = O(n)
        - Q1(n;Z,L) = O(n/L)
    - Algorithm 2:
        - W2(n) = O(n * log(n))
        - Q2(n;Z,L) = O( n / (L * logZ) )
2. Which algorithm is better?
    - Insufficient information: We want low work and high intensity
        - I1 = O(1)
        - I2 = O(log(n) * logZ)
    - Algorithm 1 does lower work while algorithm 2 has higher intensity

## Intensity - Balance and Time

1. Time to complete (Tcomp) = tau * W
    - tau = time to complete an operation = time / op
2. Time to execute Q transfers (Tmem) = alpha * L * Q
    - alpha = time to move word between slow/fast memory = time / word
3. Minimum time to execute T >= max(Tcomp, Tmem)
    - Assumes perfect overlap
4. Refactor 3 such that T >= tau * W * max(1, (alpha/tau)/(W/(LQ)))
    - tau * W is ideal computation time (assumes communication is free)
    - Second term is communication (transfer) penalty
    - Numerator (alpha/tau) is referred to as "machine balance"
        - How many operations can be executed in the time it takes to move a word of data?
        - Use B to refer to machine balance
    - Denominator is just intensity, has units operations/word
5. Minimum time to execute T >= tau * W * max(1, B/I)
6. Maximum time to execute T <= tau * W * (1 + B/I)
7. Normalized performance R = tau * W' / T
    - W' is work of best sequential algorithm
    - R <= W' / W * min(1, I/B)
    - Measure of performance is inversely proportional to time
        - Higher values are better

## Roofline Plots

1. Rmax = W' / W * min(1, I/B)




Roofline Plot

2. What are the values of x0 and y0?
    - x0 = B
    - y0 = W_star / W
3. If an algorithm is to the right of x0, we say it's compute-bound
4. If an algorithm is to the left of x0, we say it's memory-bound

## Intensity of Conventional Matrix Multiply Part 1

1. Consider a non-Strassen matrix multiply algorithm

```
for(int i = 0; i < n-1; i++)
{
    // read A[i,:]
    for(int j = 0; j < n-1; j++)
    {
        // read C[i,j] and B[:,j]
        for(int k = 0; k < n-1; k++)
        {
            C[i,j] += A[i,k] * B[k,j]
        }
        // store C[i,j]
    }
}
```

2. Assumptions:
    - L = 1 word
    - Z = 2n + O(1)
3. What is the intensity of this algorithm?
    - W(n) = n ^ 3
    - Q(n;Z) = n ^ 2 + 2 * n ^ 2 + n ^ 3

    - n ^ 2 reads for A
    - n ^ 3 reads for B
    - 2 * n ^ 2 reads/writes for C
  - I(n;Z) = W(n) / Q(n;Z) = 1
4. Only n ^ 2 data, which suggests there's a factor of n available for reuse

## Intensity of Conventional Matrix Multiply Part 2

1. Consider a matrix multiply algorithm where we load blocks of data instead of single elements

```
for i <- 0 to n-1 by b do
    for j <- 0 to n-1 by b do
        let Chat = b x b block at C[i,j]
        for k <- 0 to n-1 by b do
            let Ahat = b x b block at A[i,k]
            let Bhat = b x b block at B[k,j]
            Chat = Chat + Ahat * Bhat
        C[i,j] block <- Chat
```

2. Assumptions:
   - L = 1
   - b | n
   - n | Z
   - Z = 3 * b ^ 2 + O(1)
3. What is the intensity of this algorithm?
   - W(n) = n ^ 3
   - Q(n,Z) = n ^ 3 / b
   - Intensity = W(n) / Q(n,Z) = b = sqrt(Z)

## Informing the Architecture

1. Suppose you have an efficient machine for a matrix multiply at a particular problem size.
   - If the machine balance doubles, by how much should the size of fast memory increase?
2. Fast memory size must increase by a factor of 4 because the intensity of a matrix multiply is sqrt(Z)
   - Rmax = W_star / W * min(1, sqrt(Z)/B)
   - If B doubles, Z must increase by a factor of 4

## Conclusion

1. The two-level model captures the most important effects of real memories, capacity and transfer size
   - Lots of research on locality-sensitive algorithms based on this model
2. To exploit a memory hierarchy algorithmically, organize data accesses to maximize reuse
   - For an algorithm to scale well to future memory hierarchies, you want intensity to at least match, but preferably exceed, the machine balance