



Lecture Participation Poll #16

Log onto pollev.com/cse374

Or

Text CSE374 to 22333

Lecture 16: Trie Cont

CSE 374: Intermediate
Programming Concepts and
Tools

Administrivia

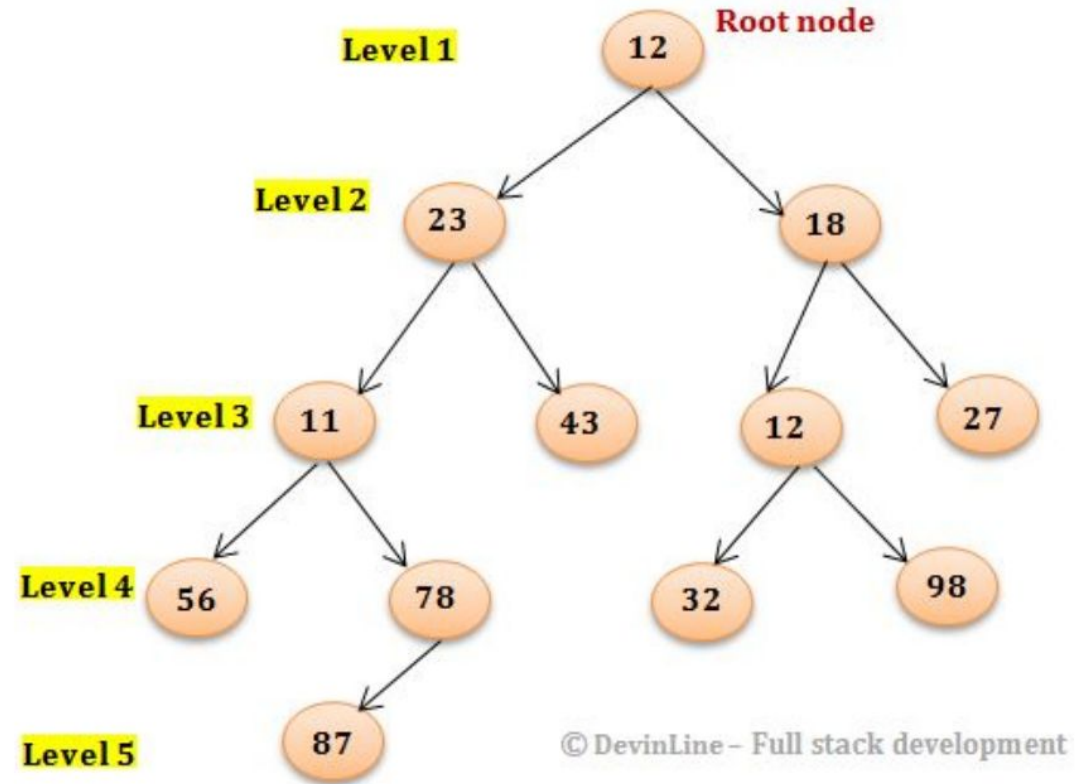
Assignments

- HW4 turn in coming later today

Binary Trees

```
struct BinaryTreeNode
{
    int data;
    struct BinaryTreeNode* left;
    struct BinaryTreeNode* right;
}
struct BinaryTree
{
    struct BinaryTreeNode* root;
}
```

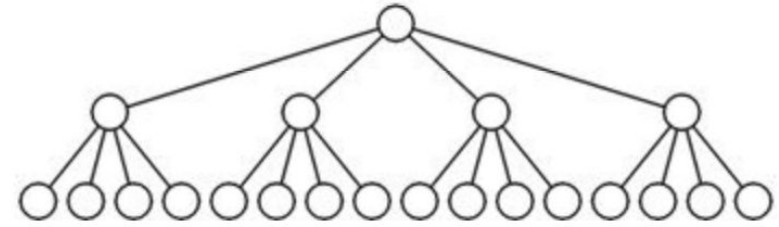
Binary tree



N-Ary Tree

```
struct TrinaryTreeNode
{
    char* data;
    struct TrinaryTreeNode* left;
    struct TrinaryTreeNode* middle;
    struct TrinaryTreeNode* right;
}

struct QuadTreeNode
{
    char* data;
    struct QuadTreeNode* children[4];
}
```



- Binary trees just one formal can have any “branching number”
- Trinary trees have branching number of three
- For arbitrarily large branching numbers, arrays can make more sense than lists of named pointers.

Prefix Tree (Trie)

Tries are a character-by-character set-of-Strings implementation

Nodes store *parts of keys* instead of *keys*

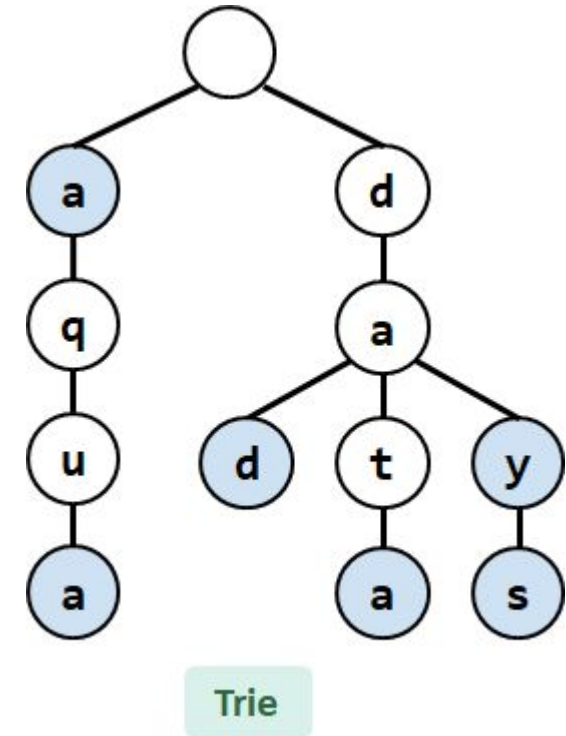
Compact data storage

Key of each node defined entirely by position

efficient worst case searching

strings often use 26-ary tree

- predictive text
- spell check



HW5 - T9 Trie

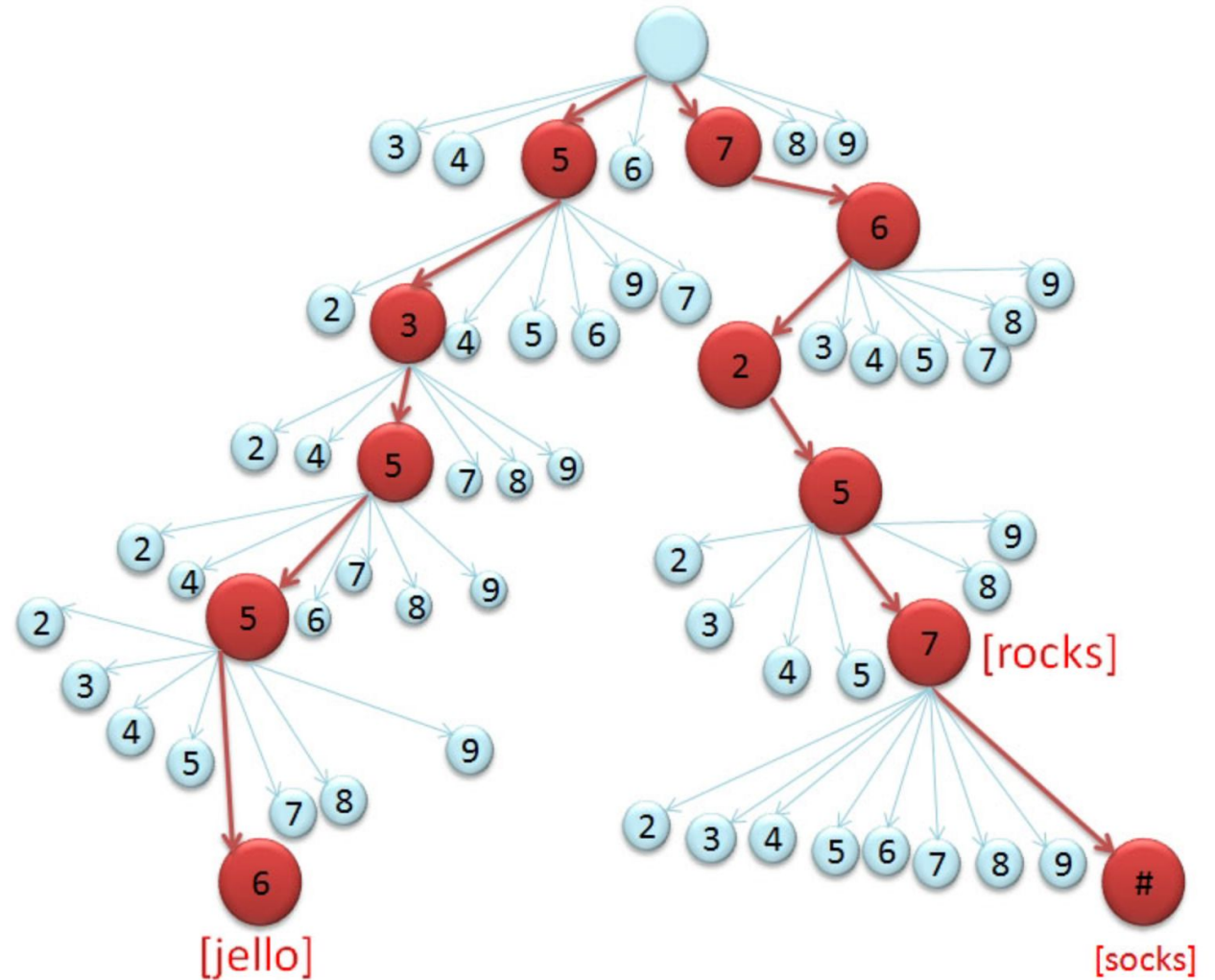
Synopsis

In this assignment, you will build programs to implement T9 predictive text, a text input mode developed originally for cell phones and still used for numeric keypads. Each number from 2-9 on the keypad represents three or four letters, the number 0 represents a space, and 1 represents a set of symbols such as { , . ! ? } etc. The numbers from 2-9 represent letters as follows:

- 2 => ABC
- 3 => DEF
- 4 => GHI
- 5 => JKL
- 6 => MNO
- 7 => PQRS
- 8 => TUV
- 9 => WXYZ



Classic trie data structures have edges labeled with letters to store prefixes of strings. But for this application, we use a compressed trie that has only 10 possible branches at each node instead of 26, since the digits 0-9 represent the 26 letters, space and symbols. Because of this, an extra layer of complexity is needed to figure out the string represented by a path.



If a word with the same numeric sequence already exists in the trie, add the new word to the trie as a link to a new node with an edge labeled '#' instead of one of the digits 2-9. (The words linked from a node by the '#' edges essentially form a "linked list" of words that have the same numeric code, but we use additional tree nodes to link the words together instead of defining a separate kind of linked-list node just for this situation.)

Trie Struct

```
typedef struct TrieNode {
    char *word;
    struct TrieNode *children[NUM_CHILDREN];
} TrieNode;
```

```
typedef struct Trie {
    TrieNode *root;
} Trie;
```

```
TrieNode* makeNode() {
    TrieNode *t = (TrieNode*) malloc(sizeof(TrieNode));
    if (t == NULL) {
        return NULL;
    }
    for (int i = 0; i < NUM_NODES; i++) {
        t->next[i] = NULL;
    }
    t->word = NULL;
    return t;
}
```

Inserting word into T9 Trie

```
/*
Recursively follows or inserts nodes starting from previous node until the location for word is found, at which
point it is inserted. Current_letter is the index of word where the recursive algorithm is currently at.
*/

int node_insert(TrieNode *previous_node, char word[], int current_letter) {
    if (word[current_letter] == '\\0') { // word is empty
        // word is empty
    }
    int digit = letter_to_digit(word[current_letter]);

    if (previous_node->children[digit] == NULL) {
        // node doesn't exist, create it
    } else { // node already exists
        current_node = //next unexamined child of previous node
    }
    if (word[current_letter + 1] == '\\0') { // at the end of the word
        if (current_node->word == NULL) { // current node doesn't have a word yet
            // save word here
        } else {
            // current node already has a word, add it as an additional completion
        }
    } else { // not at the end of the string, so continue to the next letter
        return node_insert(current_node, word, current_letter + 1);
    }
}
```




Memory Architecture

Thought experiment

```
public int sum1(int n, int m, int[][] table) {  
    int output = 0;  
    for (int i = 0; i < n; i++) {  
        for (int j = 0; j < m; j++) {  
            output += table[i][j];  
        }  
    }  
    return output;  
}
```

```
public int sum2(int n, int m, int[][] table) {  
    int output = 0;  
    for (int i = 0; i < n; i++) {  
        for (int j = 0; j < m; j++) {  
            output += table[j][i];  
        }  
    }  
    return output;  
}
```

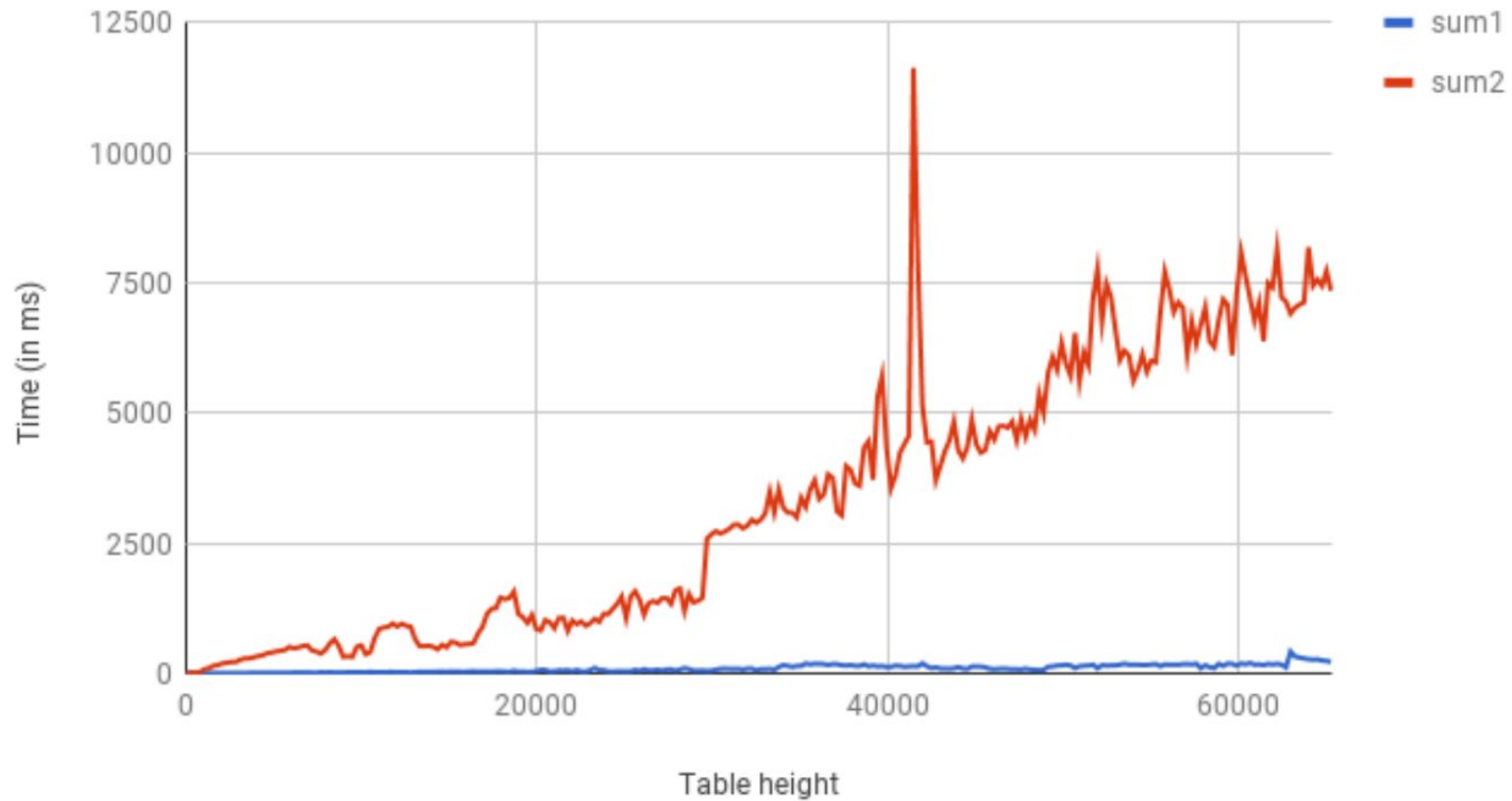
What do these two methods do?

What is the big- Θ


$\Theta(n*m)$

Thought Experiment Graphed

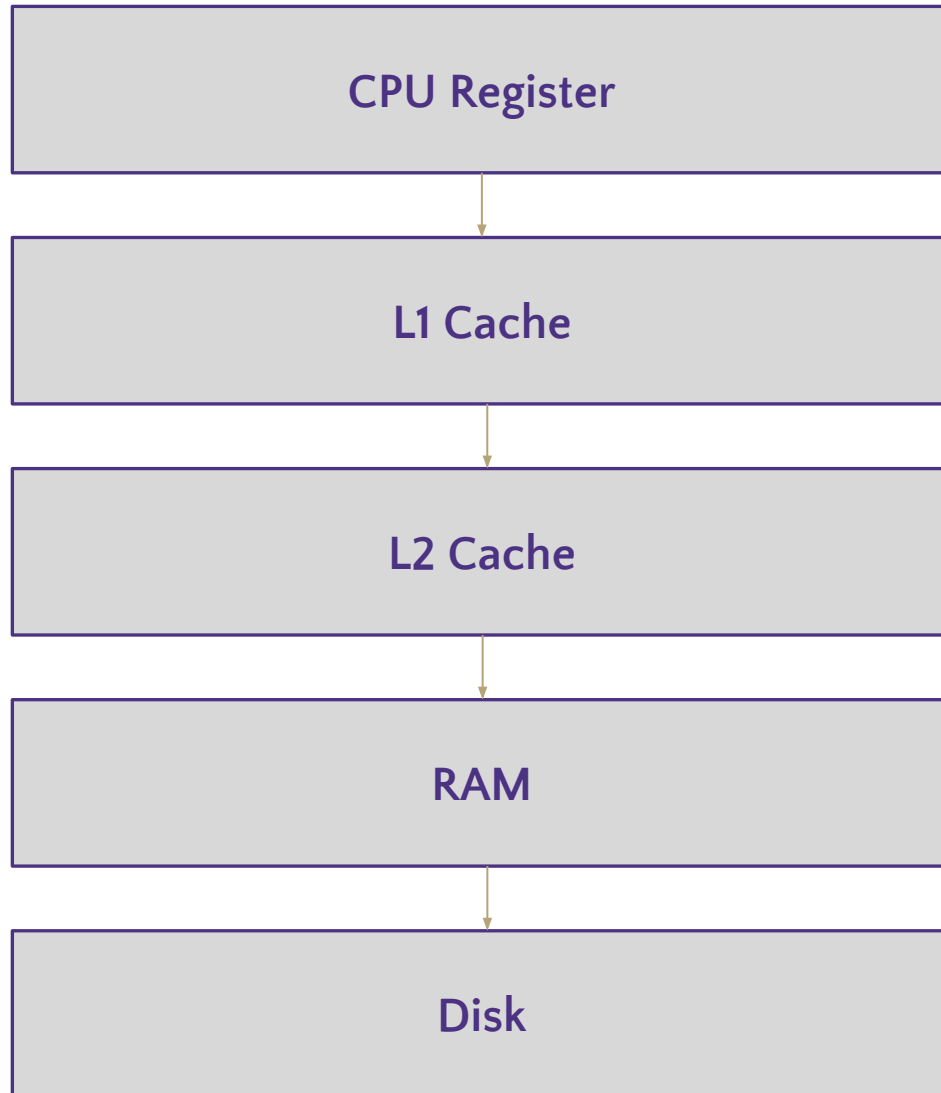
Running sum1 vs sum2 on tables of size $n \times 4096$



Incorrect Assumptions

- Accessing memory is a quick and constant-time operation 
- Sometimes accessing memory is cheaper and easier than at other times
- Sometimes accessing memory is very slow

Memory Architecture



What is it?	Typical Size	Time
The brain of the computer!	32 bits	≈free
Extra memory to make accessing it faster	128KB	0.5 ns
Extra memory to make accessing it faster	2MB	7 ns
Working memory, what your programs need	8GB	100 ns
Large, longtime storage	1 TB	8,000,000 ns

RAM (Random-Access Memory)

- RAM is where data gets stored for the programs you run. Think of it as the main memory storage location for your programs.

- RAM goes by a ton of different names: memory, main memory, RAM are all names for this same thing.



Process Name	Memory	Compressed M...	Threads
kernel_task	1.19 GB	0 bytes	144
IntelliJ IDEA	1,018.0 MB	194.7 MB	56
Microsoft PowerPoint	545.1 MB	238.9 MB	18
WindowServer	330.7 MB	170.9 MB	8
nsurlsessiond	320.8 MB	239.4 MB	3
Mattermost Helper	315.4 MB	32.0 MB	19
Google Chrome	291.7 MB	17.5 MB	31
Google Chrome Helper (Rend...	243.4 MB	91.5 MB	14
zoom.us	239.7 MB	61.8 MB	20
Google Chrome Helper (Rend...	236.6 MB	26.7 MB	14
Google Chrome Helper (GPU)	235.2 MB	19.7 MB	10
Google Chrome Helper (Rend...	203.4 MB	27.9 MB	16
Sublime Text	186.5 MB	170.9 MB	12
spindump	158.4 MB	80.0 MB	3
SystemUIServer	148.5 MB	24.9 MB	4
Finder	139.9 MB	56.3 MB	4
java	128.2 MB	61.3 MB	24
java	126.3 MB	110.3 MB	23
java	124.4 MB	27.8 MB	28
mds_stores	115.5 MB	36.2 MB	4
Mattermost	112.3 MB	37.5 MB	44
Cold Turkey Blocker	109.1 MB	49.2 MB	9
Google Chrome Helper (Rend...	102.8 MB	33.0 MB	16
Mail	91.4 MB	25.6 MB	7
Google Chrome Helper (Rend...	90.1 MB	62.4 MB	13
Google Chrome Helper (Rend...	88.1 MB	54.8 MB	13
Mattermost Helper	82.5 MB	44.8 MB	5
Google Chrome Helper (Rend...	77.4 MB	32.5 MB	13
Google Chrome Helper (Rend...	72.7 MB	51.4 MB	13

MEMORY PRESSURE	
Physical Memory:	16.00 GB
Memory Used:	9.81 GB
Cached Files:	1.94 GB
Swap Used:	628.0 MB

RAM can be represented as a huge array

RAM:

- addresses, storing stuff at specific locations
- random access



=



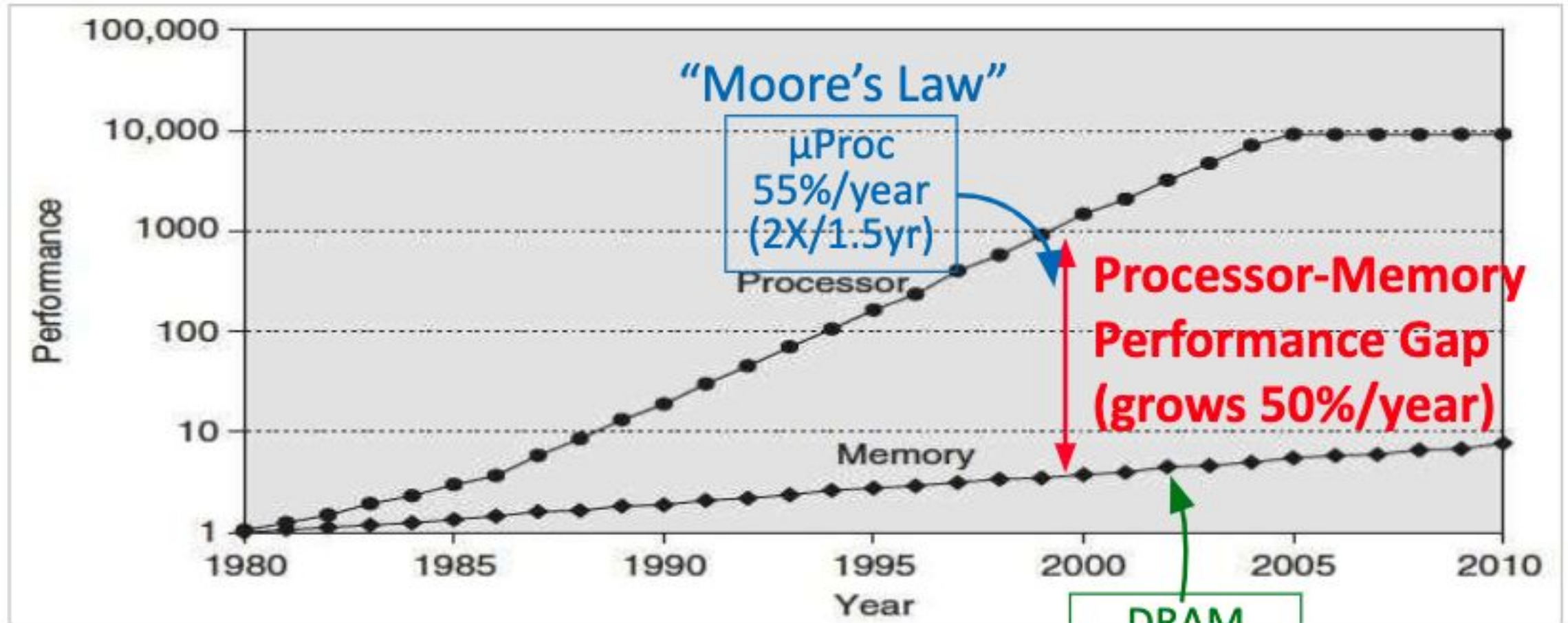
Arrays

- indices, storing stuff at specific locations
- random access

This is a main
takeaway

If you're interested in deeper than this : <https://www.youtube.com/watch?v=fpnE6UAfbtU> or take some EE classes?

Processor – Memory Gap



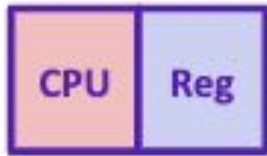
1989 first Intel CPU with cache on chip

1998 Pentium III has two cache levels on chip

DRAM
7%/year
(2X/10yrs)

Problem: Processor-Memory Bottleneck

Processor performance
doubled about
every 18 months



Bus latency / bandwidth
evolved much slower

Main
Memory

Core 2 Duo:
Can process at least
256 Bytes/cycle



Core 2 Duo:
Bandwidth
2 Bytes/cycle
Latency
100-200 cycles (30-60ns)

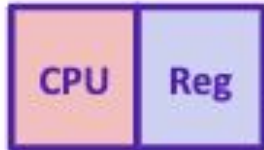


Problem: lots of waiting on memory

cycle: single machine step (fixed-time)

Problem: Processor-Memory Bottleneck

Processor performance
doubled about
every 18 months



Bus latency / bandwidth
evolved much slower



Core 2 Duo:
Can process at least
256 Bytes/cycle



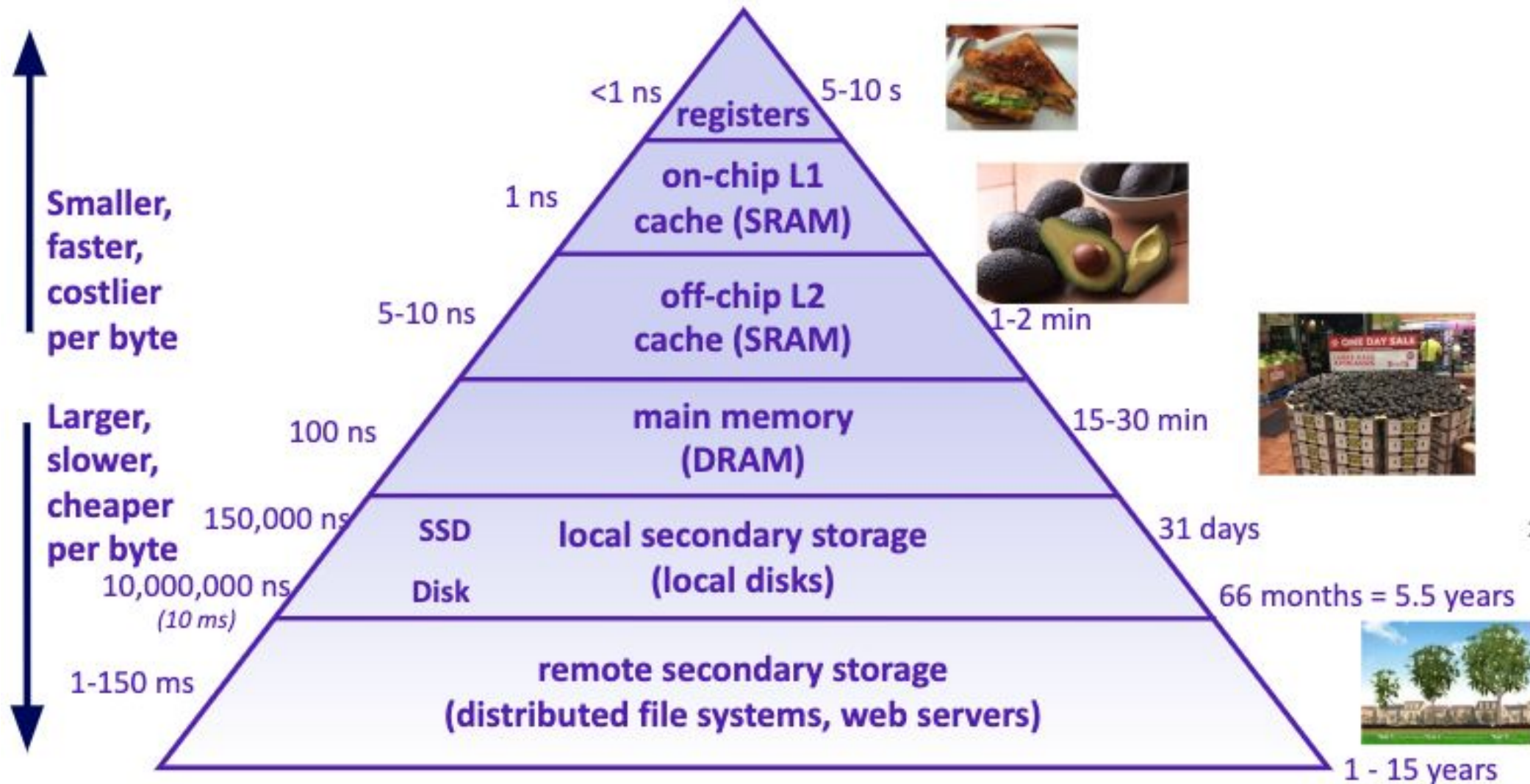
Core 2 Duo:
Bandwidth
2 Bytes/cycle
Latency
100-200 cycles (30-60ns)



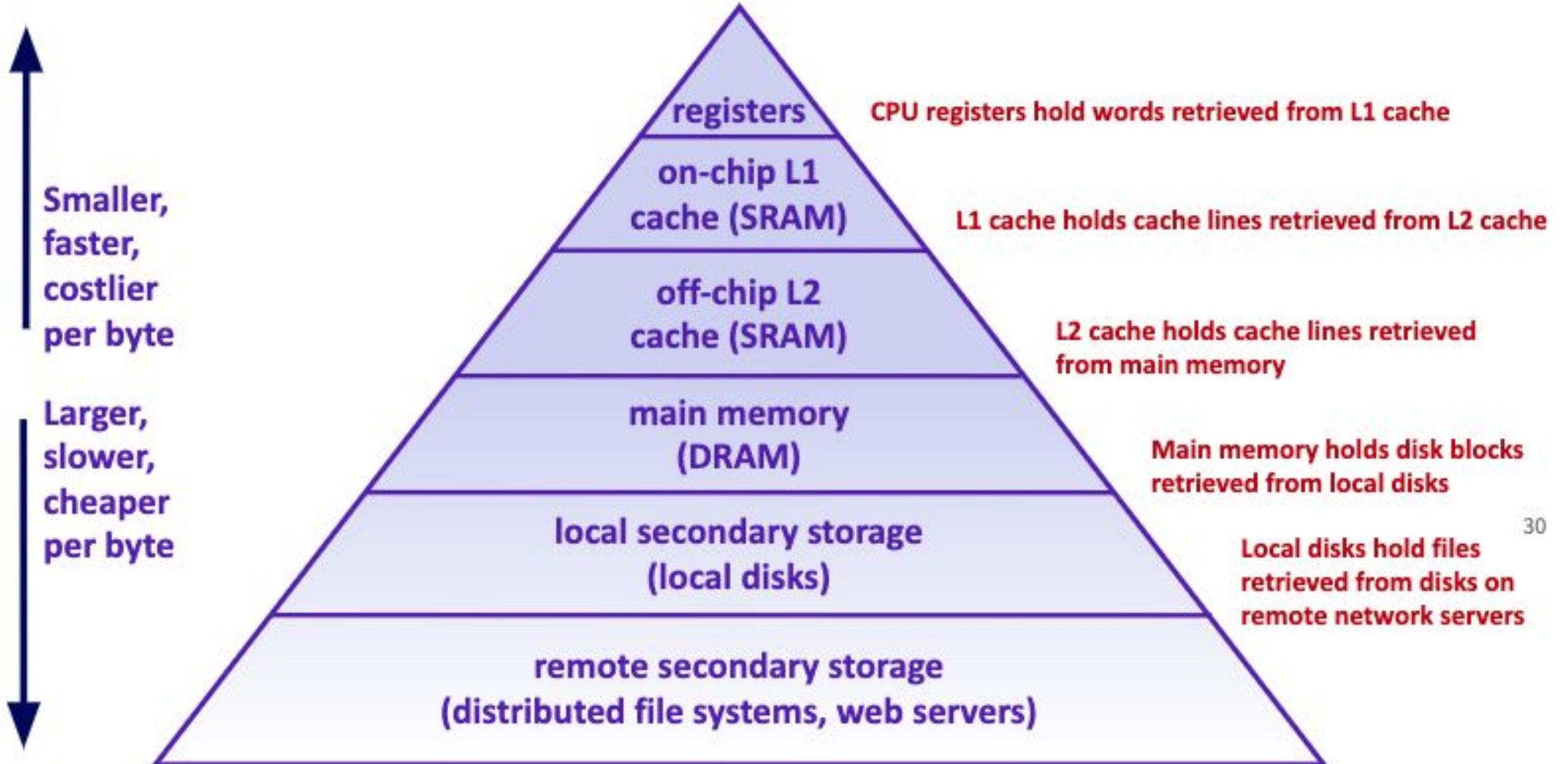
Solution: caches

cycle: single machine step (fixed-time)

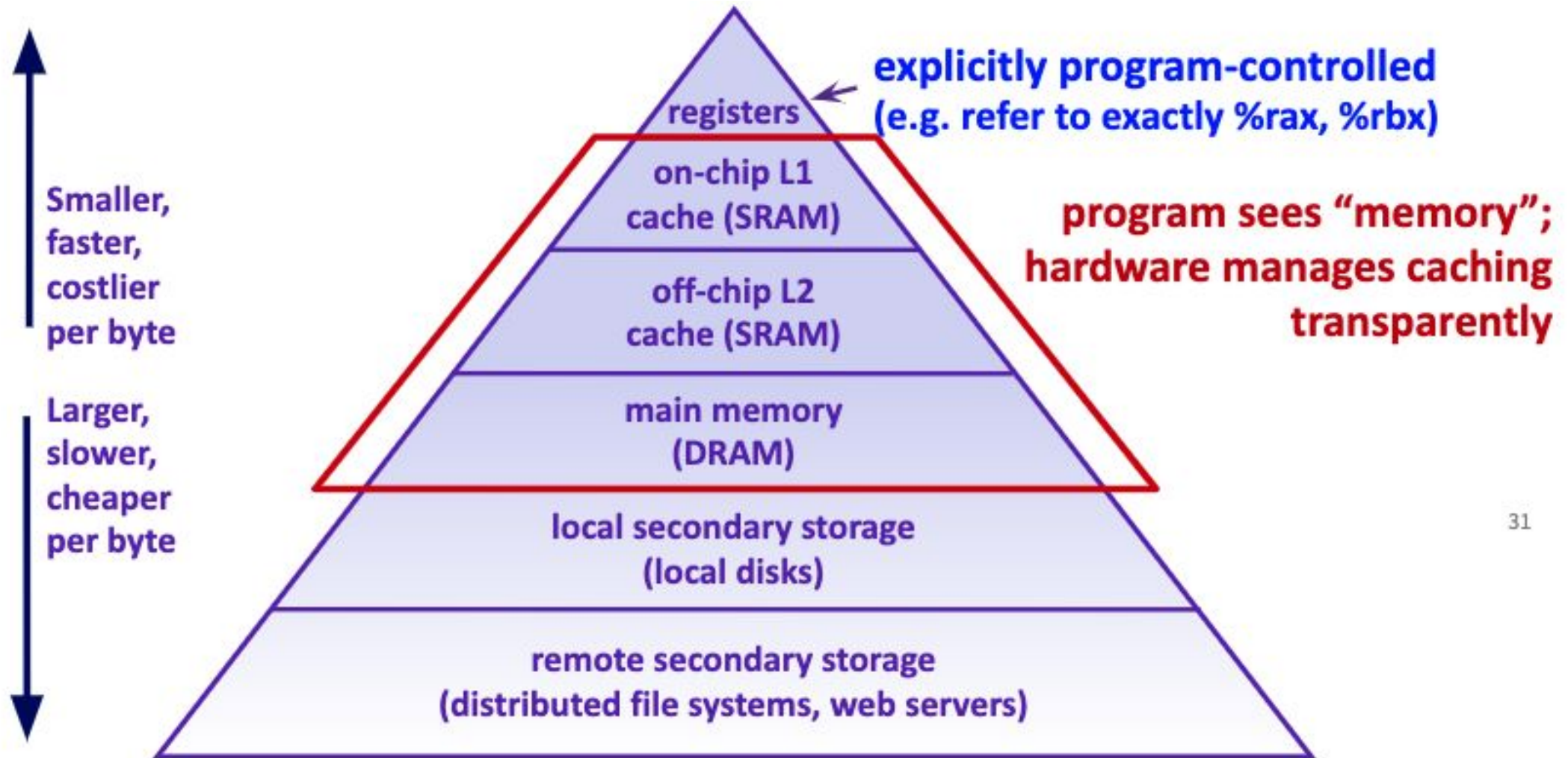
Example Memory Hierarchy



Example Memory Hierarchy



Example Memory Hierarchy



31

Review: Binary, Bits and Bytes

binary

- A base-2 system of representing numbers using only 1s and 0s
- - vs decimal, base 10, which has 9 symbols

bit

- The smallest unit of computer memory represented as a single binary value either 0 or 1

byte

The most commonly referred to unit of memory, a grouping of 8 bits

Can represent 265 different numbers (28)

1 Kilobyte = 1 thousand bytes (kb)

1 Megabyte = 1 million bytes (mb)

1 Gigabyte = 1 billion bytes (gb)

Decimal	Decimal Break Down	Binary	Binary Break Down
0		0	
1		1	
10		1010	
12		1100	
127		01111111	

Memory Architecture

Takeaways:

- the more memory a layer can store, the slower it is (generally)
- accessing the disk is **very** slow

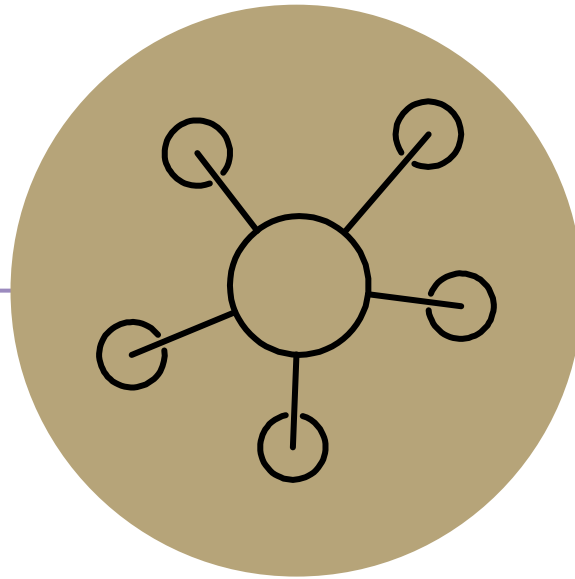
Computer Design Decisions

-Physics

- Speed of light
- Physical closeness to CPU

-Cost

- “good enough” to achieve speed
- Balance between speed and space



Appendix