

EECS 498 System Design of a Search Engine

Winter 2026

Lecture 7: Memory-mapped hash tables

Nicole Hamilton

[https://web.eecs.umich.edu/~nham/
nham@umich.edu](https://web.eecs.umich.edu/~nham/nham@umich.edu)

agenda

1. Course details.
2. Mascot vote results.
3. Team photos.
4. Relocatable data structures.
5. Memory mapping a hash table.

agenda

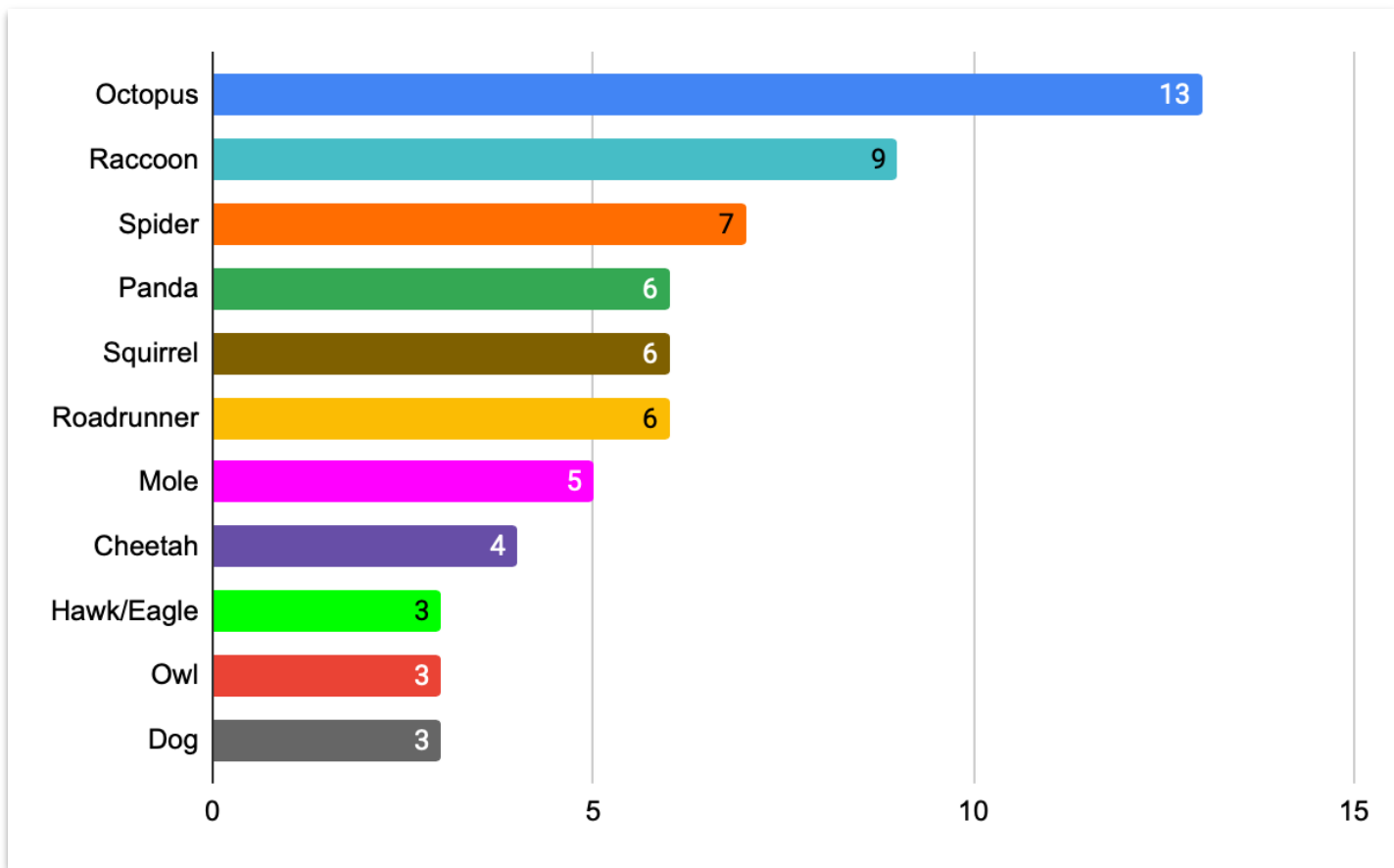
1. Course details.
2. Mascot vote results.
3. Team photos.
4. Relocatable data structures.
5. Memory mapping a hash table.

details

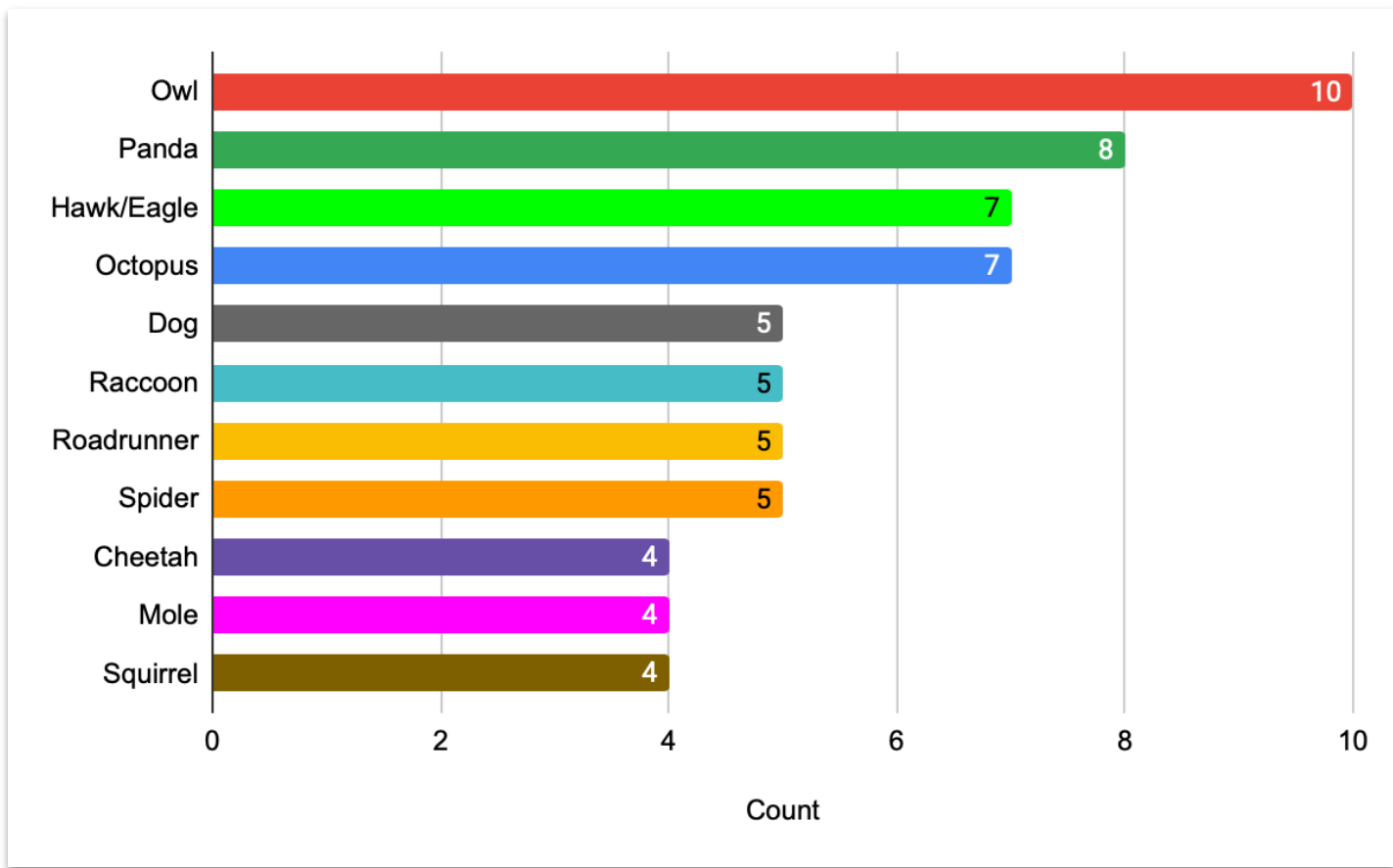
1. Project plans details posted. Keep it short and simple.
2. Minor updates to a some of the AG assignments on Canvas.
3. Questions about homework, headers, C++ version, whatever.

agenda

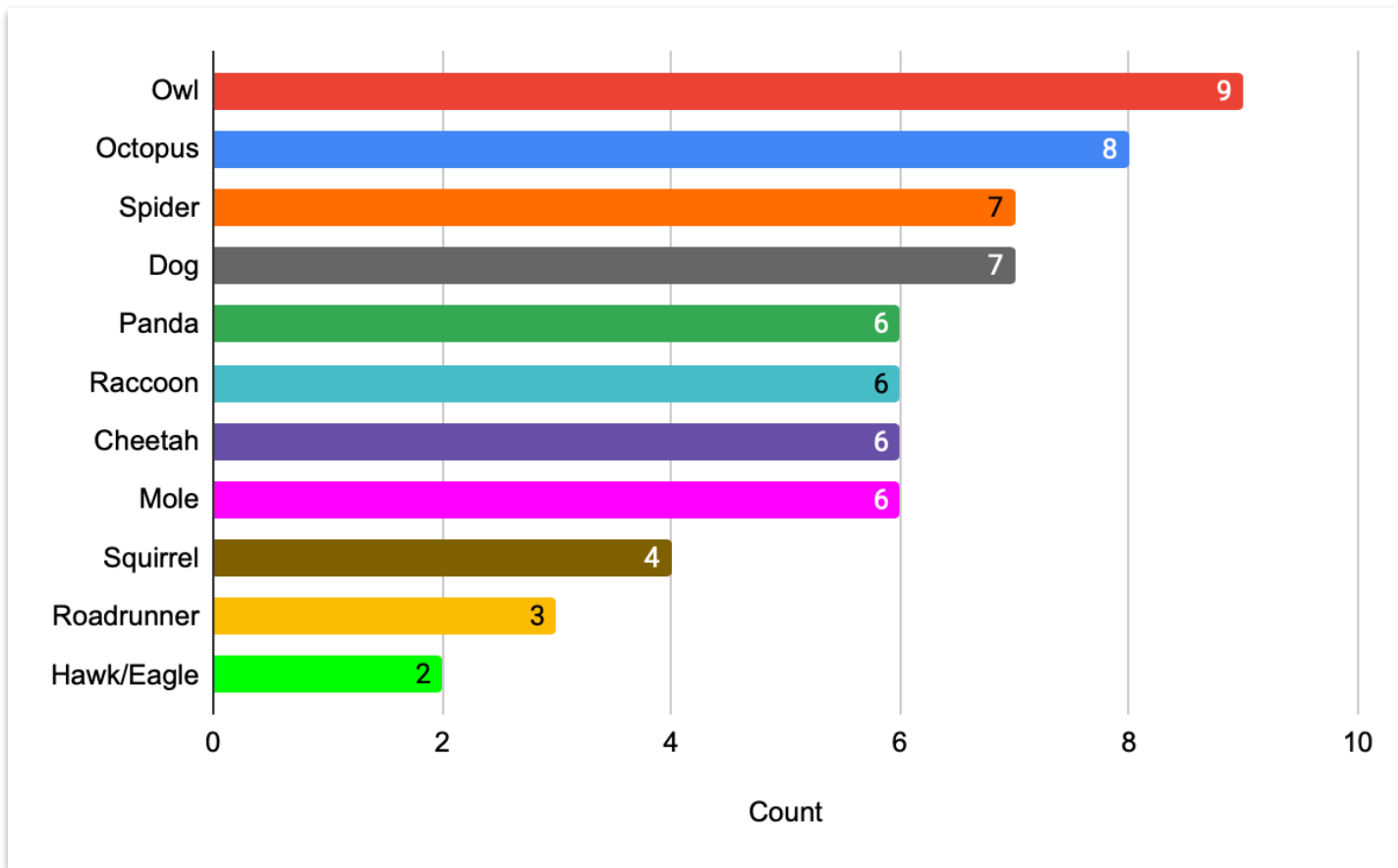
1. Course details.
2. Mascot vote results.
3. Team photos.
4. Relocatable data structures.
5. Memory mapping a hash table.



First Choice



Second Choice



Third Choice

ranked choice voting

If any candidate has $>50\%$, then they win.

Otherwise, remove the lowest tied candidates, and add their voters' next choice to the respective totals if it's not already been eliminated.

Continue until a winner is found.

round 1

65 votes, 33 is a majority.

Leader: Octopus 13

Next: Raccoon 9, Spider 7, Panda6, Roadrunner6, Squirrel 6,
Mole 5, Cheetah 4

Eliminated: Dog 3, Hawk/eagle 3, Owl 3

round 2

65 votes still active, 33 is a majority.

Leader: Octopus 14

Next: Raccoon 11, Roadrunner 8, Spider 8, Panda 7, Squirrel 7

Eliminated: Cheetah 5. Mole 5

round 3

60 votes still active, 31 is a majority.

Leader: Octopus 15

Next: Raccoon 11, Roadrunner 9, Spider 9

Eliminated: Panda 8, Squirrel 8

round 4

53 votes still active, 27 is a majority.

Leader: Octopus 19

Next: Raccoon 15, Spider 10

Eliminated: Roadrunner 9

round 5

48 votes still active, 25 is a majority.

Leader: Octopus 20

Next: Raccoon 17

Eliminated: Spider 11

round 6

42 votes still active, 22 is a majority.

Leader: Octopus 25 (majority)

Eliminated: Raccoon 17

agenda

1. Course details.
2. Mascot vote results.
3. Team photos.
4. Relocatable data structures.
5. Memory mapping a hash table.









Anthony

Andrew

Nathaniel

Satvik

William

Vasu

Ashmit

Cache Me

If You Can

The true search engine of the real fake.





ClosedAI Platform

Developer quickstart

Make your first API request in minutes.
Learn the basics of the ClosedAI platform.

Get started

javascript

```
1 import ClosedAI from "closedai";
2 const client = new ClosedAI();
3
4 const response = await client.responses.create({
5   model: "gpt-5.2",
6   input: "Generate the best search engine with a <1 second response",
7 });
8
9 console.log(response.output_text);
```



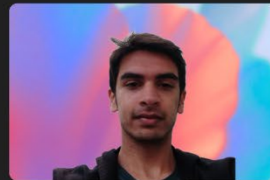
Models

[View all](#)



Jun Kit Lim New

Woke up one day just to go back to sleep



Abhishek Dahad

Used to jump out of trees recreationally



Ethan Schnathorst

If tomatoes are a fruit, ketchup is a smoothie



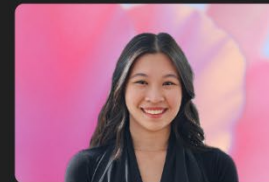
Annie Jiang

Eats cereal like it's fine dining



Kai Jun Han

Believes parallel parking is a social construct



Tessa Lee

They say money talks, but mine whispers

**Hey, we just met you, and this is crazy
but our results will leave you asking...**

Carlos

Achintya

Yuchen

Sharif

Xinyang

Thomas

Crawl Me Maybe!



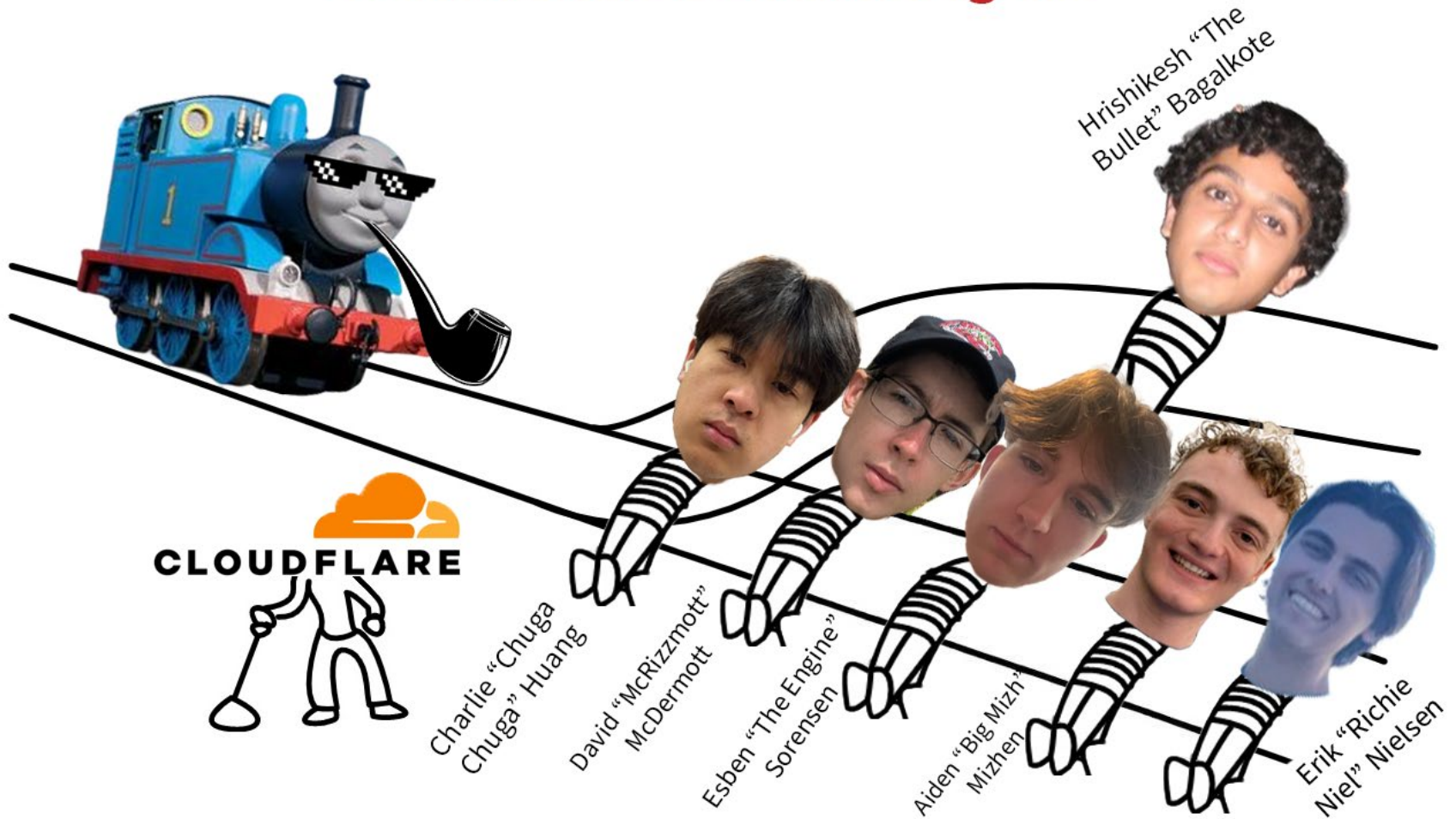
STEVE BHARGAV ZAYN CHIRAG AKASH ZACHARY JACKSON
TCHANA DEVABHAKTUNI BAIG BHAT SARADA DUNN SEIFFERLY

THE SEARCH IS ENDLESS
OCEANS 8⁷



WARNER BROS. PICTURES PRESENTS
A VILLAGE ROADSHOW PICTURES & RAINBOW ROAD PRODUCTION
A SANDRA BULLOCK "OCEAN'S EIGHT" CAST: BLANCHETT ANNE HATHAWAY ANTHONY MANN SARAH PAULSON
AUNYASINA WITH RHIANNA AND HELENA BONHAM CARTER MUSIC BY DANIEL PELLISSERTON COSTUME DESIGNER SARAH EDWARDS EDITOR JULIETTE WELFELING PRODUCTION DESIGNER ALAN OSTERLAND EXECUTIVE PRODUCERS
PRODUCED BY GEORGE CLAYTON JOHNSON & JASON GOLDBERGER WRITTEN BY MICHAEL TALEGROSS DIANA ALVAREZ JESSE GERMAN AND BRUCE BERMAN PRODUCED BY GARY ROSS DIRECTED BY GARY ROSS & SANDRA BULLOCK
PG-13
ONLY IN THEATERS
JUNE 8
#OCEANS8

Seamus the Search Engine

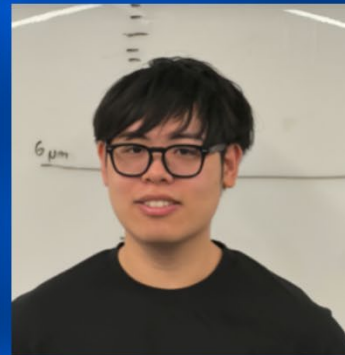




Free Information” extremist developers of ShinkanSearch escape authorities after a brief encounter with the Tokyo



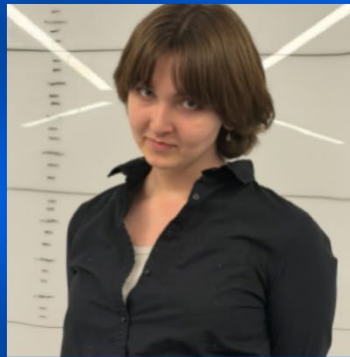
Abby Troost



Jaegun Song



Yutong Lyu



Jess Morton



Claudia Reyes



Yaire Lopez-Quiroz

LIVE

BREAKING: “THE MOST MOST WANTED” SPOTTED

“THESE CRIMINAL BADDIES MUST BE STOPPED” - THE FBI



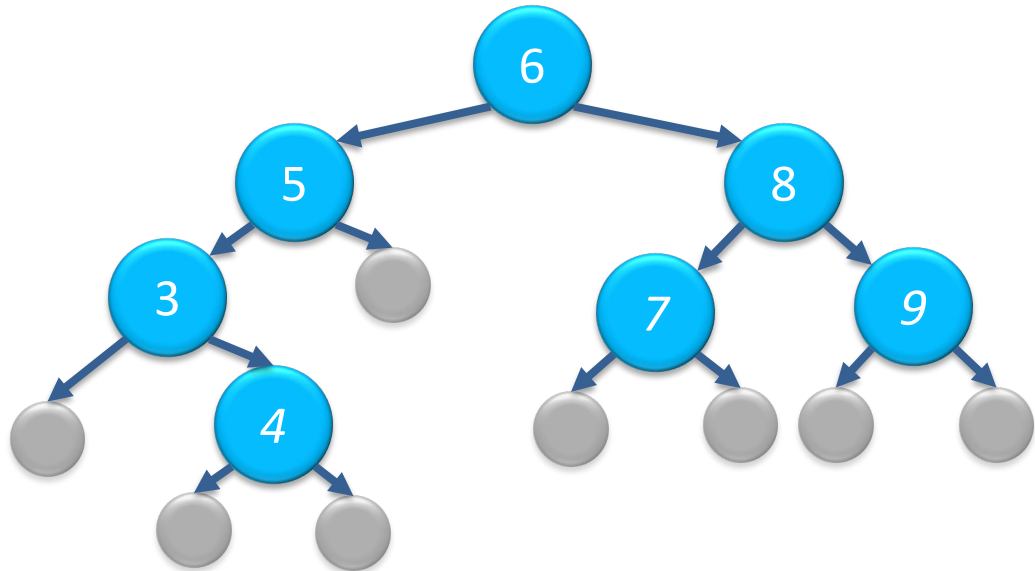
agenda

1. Course details.
2. Mascot vote results.
3. Team photos.
4. Relocatable data structures.
5. Memory mapping a hash table.

the problem with pointers

If there are pointers, you can't just memcopy the whole structure to a new location because it will break the pointers.

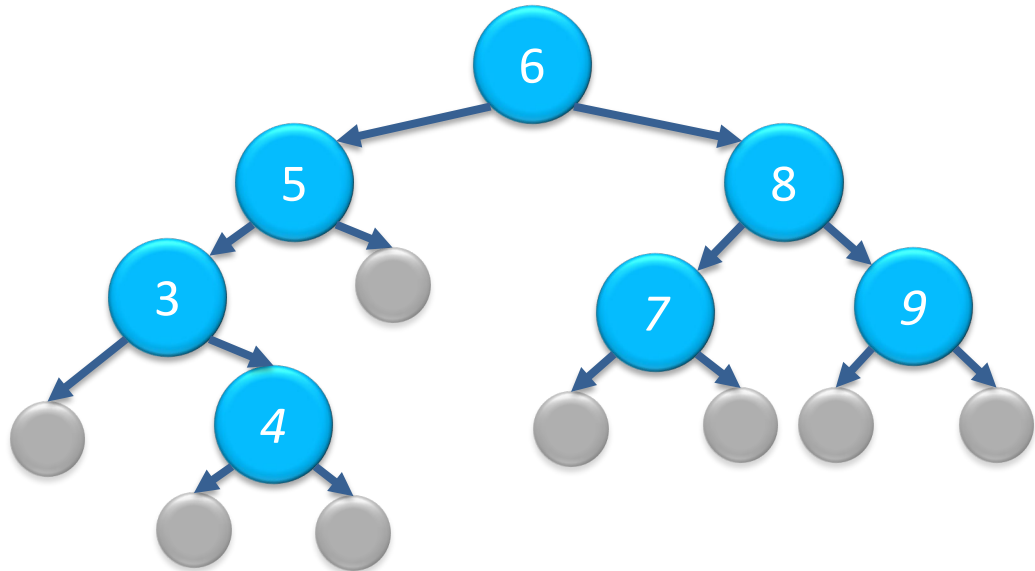
```
// Binary tree
struct node
{
    int  datum;
    node *left,
        *right;
};
```



offsets

The solution is to allocate all the nodes within a single block of memory and replace pointers with offsets from the start of the block.

```
// Binary tree  
struct node  
{  
    int datum;  
    int left,  
    right;  
};
```



agenda

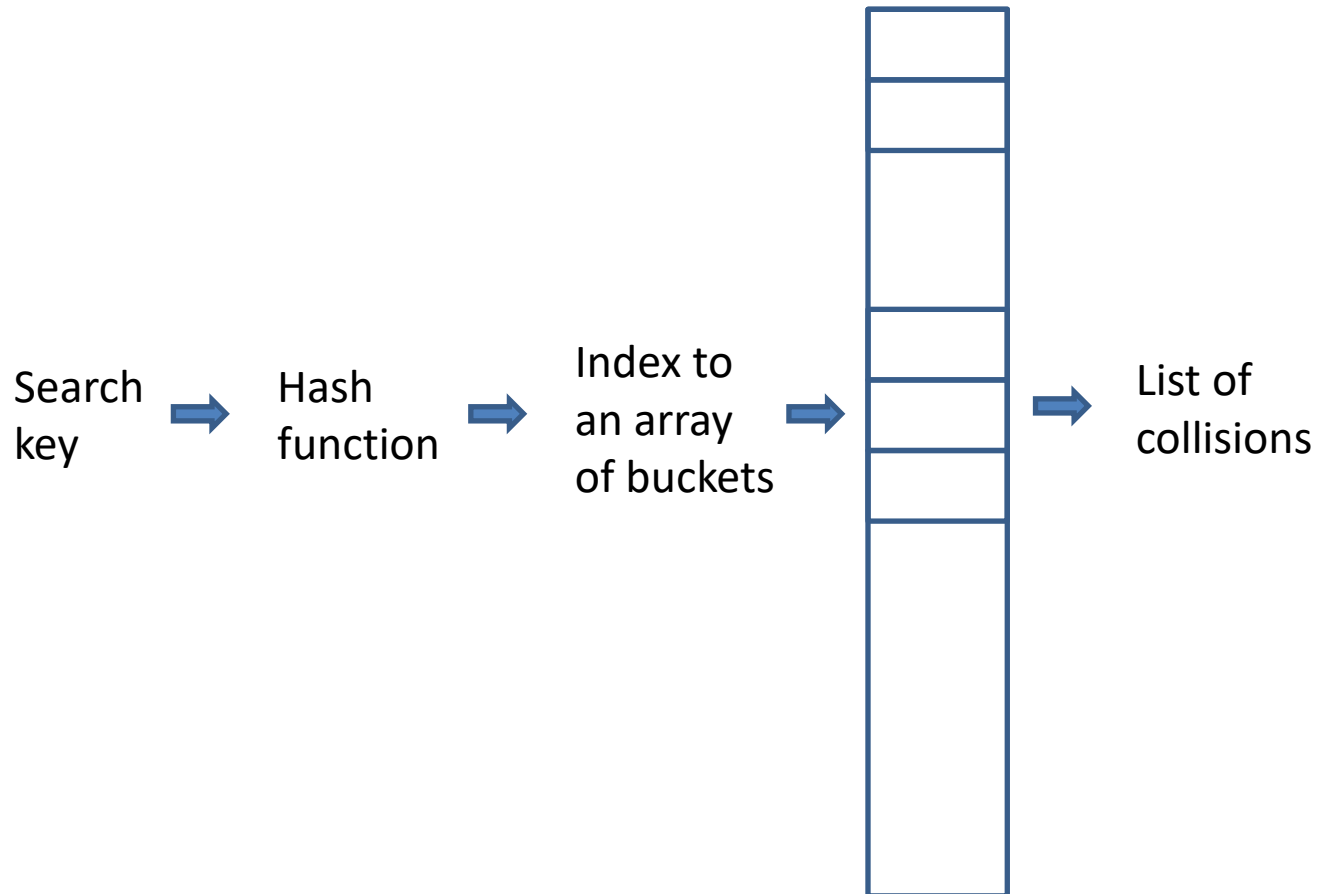
1. Course details.
2. Serializing data structures.
3. Relocatable data structures.
4. **Memory mapping a hash table.**

hashing

Create an array of buckets to hold the items you wish to index based on a key field.

Create a simple function that can map the key to a bucket index.

If multiple different keys map to the same bucket, that's called a collision that we collect on a list.



HashTable homework

In this assignment, you will build a simple hash table with collisions collected on a linked list.

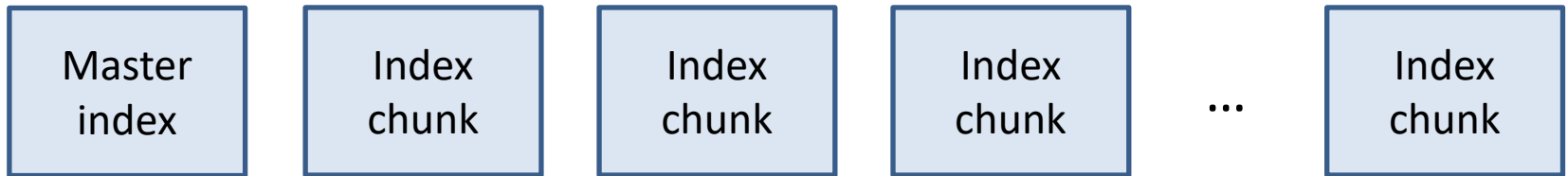
In the follow-on HashBlob and HashFile assignments, you will:

1. Serialize your hashtable, including the collisions lists, into a contiguous, minimal-size block of memory with pointers replaced with offsets.
2. Write it to a file.
3. In a separate program, memory map the file and use as a live hash table for searching.

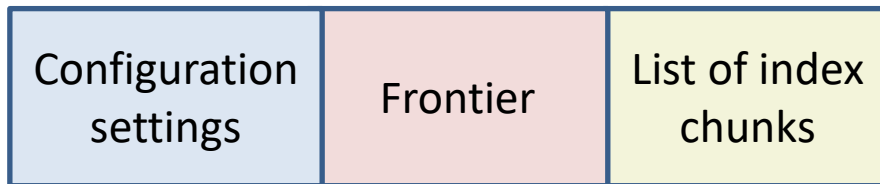
HashTable homework

This is leading up to how we will build the index on disk but memory-map it for running queries.

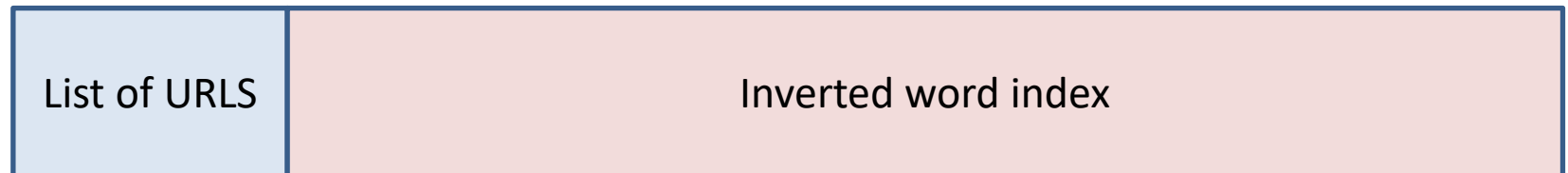
A search engine index is typically a set of files



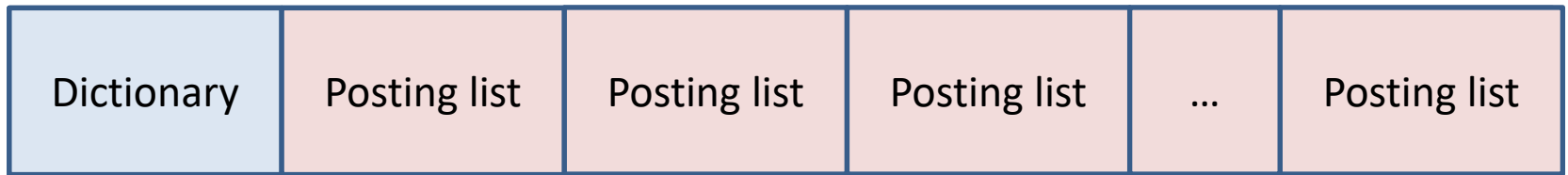
A master index



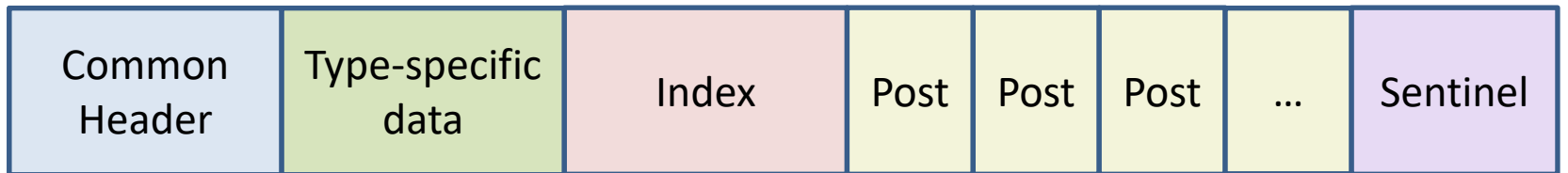
Each index chunk



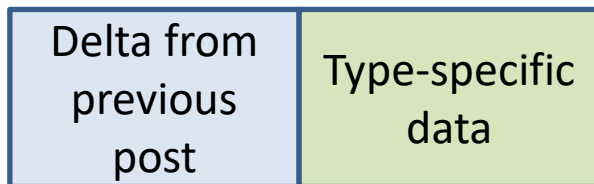
The inverted word index



A posting list



An individual post



```
tcsh-1% g++ HashTable.cpp Common.cpp -o HashTable
```

```
tcsh-2% ./HashTable
```

```
Usage: HashTable [ -vL ] wordsin.txt
```

Reads words from wordsin.txt, creates a HashTable in memory, optimizes it, then searches it for words read from cin.

-v means verbose output

-L means read whole lines as words

```
tcsh-3% ./HashTable Testcases/Hamlet.txt
```

```
now is the time
```

```
36    now
```

```
299   is
```

```
929   the
```

```
32    time
```

```
tcsh-4%
```

```
// Simple hash table template.
```

```
template< typename Key, typename Value > class Tuple
{
public:
    Key key;
    Value value;

    Tuple( const Key &k, const Value v ) : key( k ), value( v )
    {
    }
};
```

```
template< typename Key, typename Value > class Bucket
{
public:
    Bucket *next;
    uint32_t hashValue;
    Tuple< Key, Value > tuple;

    Bucket( const Key &k, uint32_t h, const Value v ) :
        tuple( k, v ), next( nullptr ), hashValue( h )
    {
    }
};
```

```

template< typename Key, typename Value > class HashTable
{
private:

    // Your code here.

    Bucket< Key, Value > **buckets;
    size_t numberOfBuckets;

    friend class Iterator;
    friend class HashBlob;

public:

    Tuple< Key, Value > *Find( const Key k, const Value initialValue )
    {
        // Search for the key k and return a pointer to the
        // ( key, value ) entry. If the key is not already
        // in the hash, add it with the initial value.

        // Your code here.

        return nullptr;
    }

```

```
Tuple< Key, Value > *Find( const Key k ) const
{
    // Search for the key k and return a pointer to the
    // ( key, value ) entry. If the key is not already
    // in the hash, return nullptr.

    // Your code here.

    return nullptr;
}
```

```

void Optimize( double loading = 1.5 )
{
    // Modify or rebuild the hash table as you see fit
    // to improve its performance now that you know
    // nothing more is to be added.

    // Your code here.
}

// Your constructor may take as many default arguments
// as you like.

HashTable( )
{
    // Your code here.
}

~HashTable( )
{
    // Your code here.
}

```

```

class Iterator
{
private:

    friend class HashTable;

    // Your code here.

    Iterator( HashTable *table, size_t bucket, Bucket<Key, Value> *b )
    {
        // Your code here.
    }

public:

    Iterator( ) : Iterator( nullptr, 0, nullptr )
    {
    }

    ~Iterator( )
    {
    }

:
};

```

```
Iterator begin( )  
    {  
        // Your code here.  
    }  
  
Iterator end( )  
    {  
        // Your code here.  
    }  
};
```

```

// Common code used by the various hashing sample applications.

using Hash = HashTable< const char *, size_t >;
using Pair = Tuple< const char *, size_t >;

// Build a HashTable of strings and numbers of occurrences, given a vector
// of strings representing the words. You may assume the vector and the
// strings will not be changed during the lifetime of the Hash.

// You may add additional parameters but they must have defaults in
// Common.h.

// Caller is responsible for deleting the Hash.

Hash *BuildHashTable( const vector< string > &words )
{
    // Your code here.

    return nullptr.
}

```

```

void CollectWordsIn( int argc, char **argv, vector< string > &words )
{
:
// Open wordsin.txt

ifstream wordsin( *argv );
argc--;

assert( wordsin.is_open( ) );

string word;
:
    while ( wordsin >> word )
        {
            words.push_back( word );
            sumLengths += word.size( );
        }
:
}

```

```
// Sample HashTable application
```

```
void Usage( )  
{  
    cout <<  
  
        "Usage:  HashTable [ -vL ] wordsin.txt\n"  
        "\n"  
        "Reads words from wordsin.txt, creates a HashTable\n"  
        "in memory, optimizes it, then searches it for words\n"  
        "read from cin.\n"  
        "\n"  
        "-v means verbose output\n"  
        "-L means read whole lines as words\n";  
  
    exit( 0 );  
}
```

```

using Hash = HashTable< const char *, size_t >;
using Pair = Tuple< const char *, size_t >;

void Search( const Hash *hashtable )
{
    // Search the HashTable for words read from cin.

    string word;
    while ( cin >> word )
    {
        const Pair *p = hashtable->Find( word.c_str( ) );
        cout << ( p ? p->value : 0 ) << "    " << word << endl;
    }
}

```

```

int main( int argc, char **argv )
{
    if ( argc < 2 )
        Usage( );

    vector< string > words;

    CollectWordsIn( argc, argv, words );

    if ( optVerbose )
    {
        :
    }
    else
    {
        Hash *hashtable = BuildHashTable( words );
        hashtable->Optimize( );
        Search( hashtable );
        delete hashtable;
    }
}

```

HashBlob

In the next homework, you'll serialize your hash table into a relocatable blob that can be written to a file.

```
tcsh-1% g++ --std=c++17 HashBlob.cpp Common.cpp -o HashBlob
tcsh-2% ./HashBlob
Usage: HashBlob [ -L ] wordsin.txt [ blobfile ]
```

Reads words from wordsin.txt, creates a hashblob,
then searches it for words read from cin.

If blobfile is specified, it writes the hashblob
into the blobfile.

-v means verbose output

-L means read whole lines as words

```
tcsh-3% ./HashBlob Testcases/Hamlet.txt
```

```
now is the time
```

```
36    now
```

```
299   is
```

```
929   the
```

```
32    time
```

```
tcsh-4%
```

```

class HashBlob
{
    // This will be a hash specifically designed to hold an
    // entire hash table as a single contiguous blob of bytes.
    // Pointers are disallowed so that the blob can be
    // relocated to anywhere in memory

    // The basic structure should consist of some header
    // information including the number of buckets and other
    // details followed by a concatenated list of all the
    // individual lists of tuples within each bucket.

public:

    static const size_t HashBlobMagic = 0x4e69636f6c65,
        CurrentVersion = 0;

    size_t MagicNumber,
        Version,
        BlobSize,
        NumberOfBuckets,
        Buckets[ Unknown ];

    // The SerialTuples will follow immediately after.
    :
};

```

```
tcsh-11% ./HashBlob Testcases/Hamlet.txt MyBlob.bin
tcsh-12% g++ --std=c++17 HashFile.cpp Common.cpp -o HashFile
tcsh-13% ./HashFile
Usage: HashFile [ -v ] blobfile
```

Map the blobfile into memory and then search it for words read from cin.

```
-v means verbose output
tcsh-14% ./HashFile MyBlob.bin
now is the time
36    now
299   is
929   the
32    time
tcsh-15%
```