Midterm Review

Intermediate Computer Systems Programming, Fall 2025

Instructor: Travis McGaha

TAs: Theodor Bulacovschi Ash Fujiyama

Administrivia

- Midterm Details Posted: In-class on Wed Oct 22nd
  - You can bring a 1 sheet (double sided) 8.5 x 11 sheet of paper of notes.
    You can type it or handwrite it, but it must be your own (no two students should have the same sheet)
  - Clobber Policy
  - IF YOU CAN'T MAKE THE EXAM LET ME KNOW AS SOON AS YOU ARE AWARE
- HW07: Posted!
  - We don't expect you to work on it till after midterm, but you can start whenever.
  - Shouldn't be too long (hopefully). You are doing an "Embarrassingly Parallel" problem
- Mid-sem Survey & Check-in posted after the exam, due on Mon after exam
  - (checkin will just be reopens and survey)

Administrivia: TEST

- TEST THE UBUNTU COLOR CONTRAST

- Do these themes work better going forward for you to read the code projected?

Midterm Philosophy / Advice (pt. 1)

- I do not like midterms that ask you to memorize things
  - You will still have to memorize some critical things.
  - I will hint at some things, provide documentation or a summary of some things. (for example: I will list some of the functions that may be useful and a brief summary of what the function does)

- I am more interested in questions that ask you to:
  - Apply concepts to solve new problems
  - Analyze situations to see how concepts from lecture apply

- Will there be multiple choice?
  - If there is, you will still have to justify your choices

Midterm Philosophy / Advice (pt. 1.5)

- I do not like midterms that ask you to memorize things
  - You will still have to memorize some critical things.
  - I will hint at some things, provide documentation or a summary of some things. (for example: I will list some of the functions that may be useful and a brief summary of what the function does)

- Example of some documentation I may provide:

map<K, V> / unordered_map<K, V>

Function signature	Quick description
`bool contains(const K& key);`	Iff the map contains the specified key
`V& operator[](const K& key);`	Returns associated value. If key not present then default inserts it.
`V& at(const K& key);`	^^ but throws std::out_of_range if the key is not present
`iterator find(const K& key);`	Returns an itertor to the key-value pair of specified key, or end() if not found.
`iterator begin();`	Iterator to the first key-value pair in the container
`iterator end();`	Iterator to the end of the container. Not valid to be de-referenced.
`size_t size();`	Number of elements in the container

pthread_join

SYNOPSIS

```
int pthread_join(pthread_t thread, void **retval);
```

DESCRIPTION

The pthread_join() function waits for the thread specified by thread to terminate. If that thread has already terminated, then pthread_join() returns immediately.

If retval is not NULL, then pthread_join() copies the return value of the target thread into the location pointed to by retval.

Midterm Philosophy / Advice (pt. 2)

- I am still trying to keep the exam fair to you, you must remember some things
  - High level concepts or fundamentals. I do not expect you to remember every minute detail.
    - E.g. how a multi level page table works should be know, but not the exact details of what is in each page table entry
    - (I know this boundary is blurry, but hopefully this statement helps)

- I am NOT trying to “trick” you (like I sometimes do in poll everywhere questions)

Midterm Philosophy / Advice (pt. 3)

- I am trying to make sure you have adequate time to stop and think about the questions.
  - You should still be wary of how much time you have
  - But also, remember that sometimes you can stop and take a deep breath.
- Remember that you can move on to another problem.
- Remember that you can still move on to the next part even if you haven't finished the current part

Midterm Philosophy / Advice (pt. 4)

- On the midterm you will have to explain things
- Your explanations should be more than just stating a topic name.
- Don't just say something like (for example) "because of threads" or just state some facts like "threads are parallel and lightweight processes".
- State how the topic(s) relate to the exam problem and answer the question being asked.

Disclaimer

- **THIS REVIEW IS NOT EXHAUSTIVE**
- **Topics not in this review are still testable**
  - **We recommend going through the course material. Lecture polls, recitation worksheets, and the previous homeworks.**

Review Topics

- C++ Programming
- C++ Memory
- git
- Caches & Locality
- Processes
- Threads

C++ Programming (pt 1)

- Implement the function filter() which takes in a vector of integers and a set of integers. The function returns a new vector that contains all of the integers of the input vector, except for any elements that were in the set.

- For example, the following code should print
  - 4
  - 5

```
vector<int> v {3, 4, 5};
set<int> s {3, 6};

auto res = filter(v, s);

for (auto& num : res) {
  cout << num << endl;
}
```

C++ Programming (pt 1)

```
vector<int> filter(const vector<int>& numbers
                   const set<int>& omit) {

  vector<int> result{};

  for (const auto& num : numbers) {
    if (!omit.contains(num)) {
      result.push_back(num);
    }
  }
  return result;
}
```

C++ Programming (pt 2)

- Implement the function invert() which takes in a map that maps strings to other strings. The function returns a map of strings to vectors of strings that represents the "reverse mapping" of the input map. In other words, the keys in the result map should be all the values in the input map. The values in the output map should be all keys that mapped to that value in the input map.

- For example, consider:

```
map<string, string> map;
map["radar"] = "tacoma";
map["rain"] = "tacoma";
map["transit"] = "philly";

map<string, vector<string>> res = invert(map);
// res should be:
// {
//   "tacoma" -> ["radar", "rain"],
//   "philly" -> ["transit"],
// }
```

C++ Programming (pt 2)

```
map<string, vector<string>> invert(const map<string, string>& map) {
  map<string, vector<string>> res;

  for (const auto& kv : map) {
    res[kv.second].push_back(kv.first);
  }

  return res;
}
```

C++ Programming (pt 3)

- Implement the function lookup() which takes in a map that maps “words” to another map. The inner map contains document names and how many times the word shows up in the specified document. Your function also takes in a vector of words called "query". Your code returns a vector of all documents that contains every word in the query. Each document returned also has the total count of words
- For example, consider:

```
map<string, map<string, int>> index;
index["bye"]["the_wall.txt"] = 2; // bye shows up twice in the_wall.txt
index["bye"]["lyrics.txt"] = 1;
index["hi"]["lyrics.txt"] = 3;
index["hi"]["blank.txt"] = 50;

vector<pair<string, int>> res = lookup(index, {"bye", "hi"});
// res should be:
// { ("lyrics.txt", 4) }
```

- You can make helper functions or structs if you want.

C++ Programming (pt 3)

```
vector<pair<string, int>> lookup(const map<string, map<string, int>>& index,
                                 const vector<string>& query) {









}
```

C++ Programming (pt 3)

```
vector<pair<string, int>> lookup(const map<string, map<string, int>>& index,
                                 const vector<string>& query) {
  vector<pair<string, int>> result;
  if (query.size() == 0) {
    return result;
  }

  for (const auto& p : index.at(query.at(0))) {
    result.push_back(p);
  }

  for (size_t i = 1; i < query.size(); ++i) {
    for (size_t j = 0; j < result.size(); ++j) {
      const auto& inner_map = index.at(query.at(i));
      const auto& curr_result = result.at(j);
      if (inner_map.contains(curr_result.first)) {
        curr_result.second += inner_map.at(curr_result.first);
      } else {
        result.erase(result.begin() + j); // erase takes an iterator
        --j;
      }
    }
  }
  return result;
}
```

Many possible solutions.
This one we tried to use as few functions of the std library as possible.

You can use other things in the std library if you like.

Note that operator[] on the map would not work here cause it is const.

C++ Memory Diagram & Allocatic

- Consider the following code that uses std::list (linked list)
- How many memory allocations occur in this code?
- What is the state of memory when we reach HERE?

```
int main() {
  list<coord> l;
  coord rn = {1, 1};
  l.push_back(rn);
  rn = {2, 2};
  l.push_back(rn);
  rn = {3, 3};
  l.push_back(rn);
  list<coord> result = std::move(scale(l));
  // HERE
}
```

```
struct coord {
  int x;
  int y;
}

list<coord> scale(list<coord> to_norm) {
  int total_x = 0;
  int total_y = 0;
  for (coord r : to_norm) {
    total_x += r.x;
    total_y += r.y;
  }

  for (coord& r : to_norm) {
    r.x *= total_x;
    r.y *= total_y;
  }

  return to_norm;  // result is moved
}
```

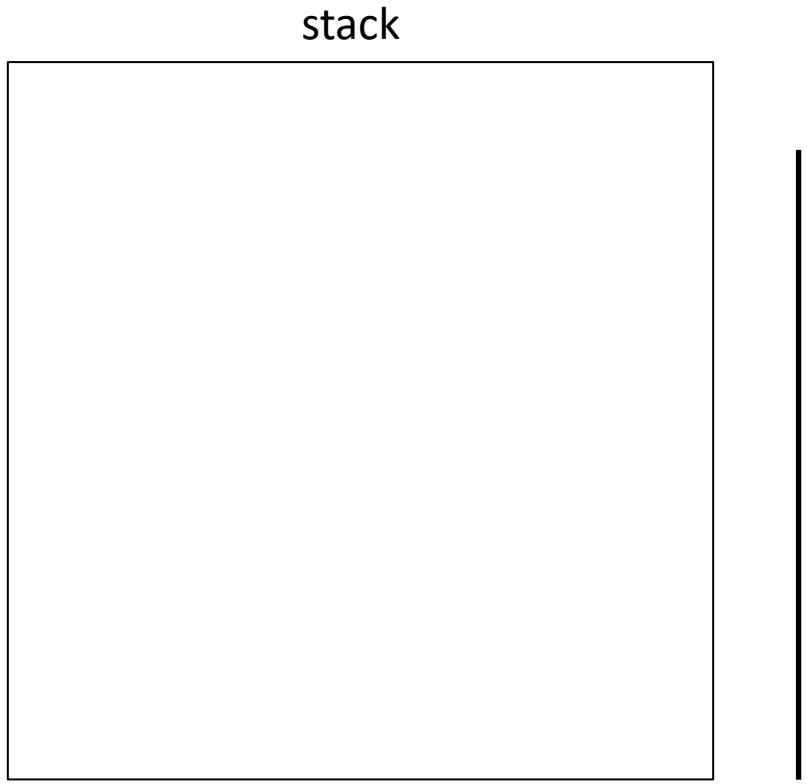
C++ Memory Diagram & Allocations

- Consider the following code that uses std::list (linked list)
- How many memory allocations occur in this code?
  - **0 for initial construction of the list in main**
  - **3 for push_back in main (1 for each node that must be allocated for the list)**
  - **3 for copy constructing the list as a parameter to scale()**
  - **0 for iterating ove the list in scale(). Yes we do make a copy of the coord structs, but those are just ints, no memory allocation needed**
  - **0 for moving the returned list out to the list result in main**

- **6 in total**

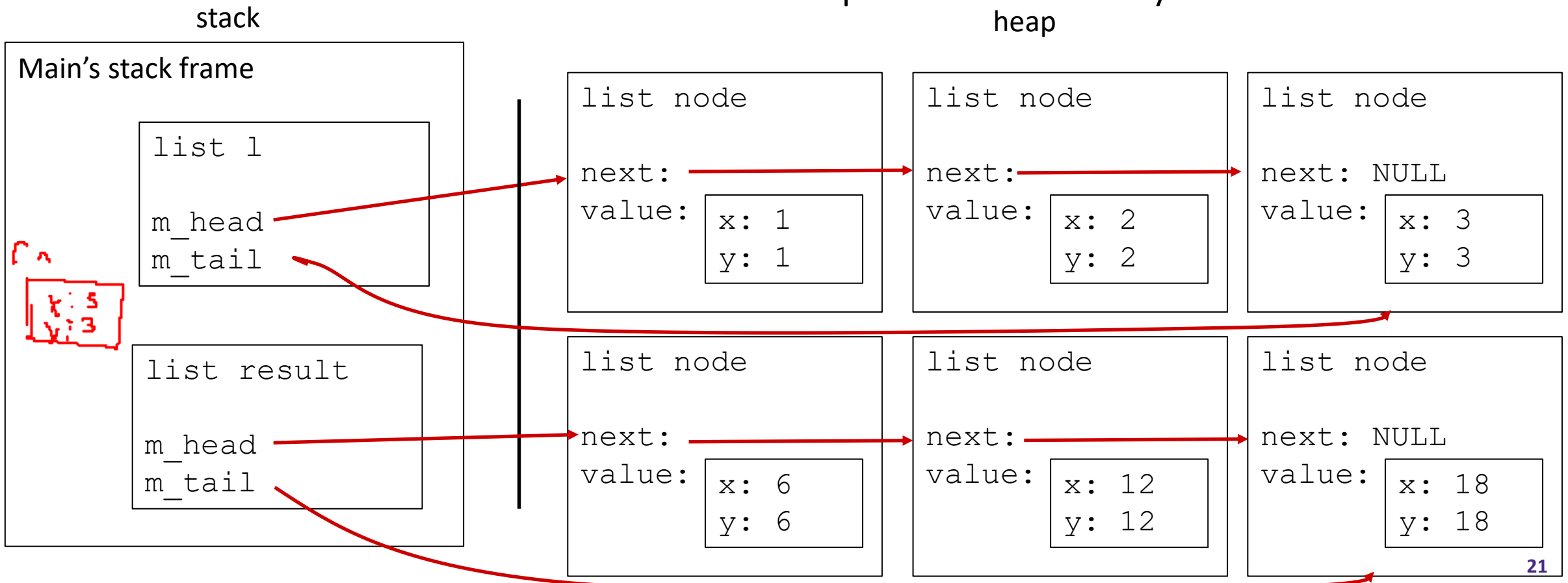
C++ Memory Diagram & Allocations

- Memory Diagram:
  - Since we didn't go over the exact internals of the linkedlist, it would have been fine to have a slightly different linked list structure (e.g. no tail_ pointer) as long as it was clear it was a linked list and the nodes were on the heap similar to how they are here:



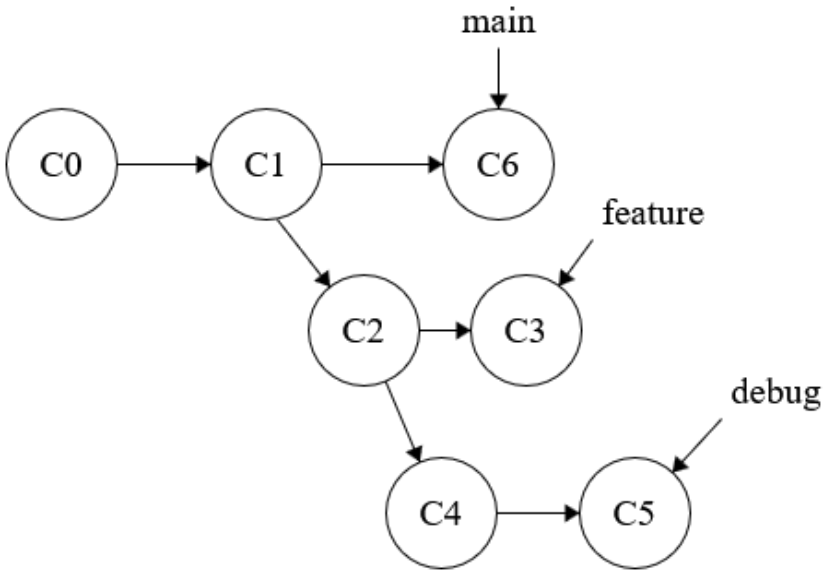
C++ Memory Diagram & Allocations

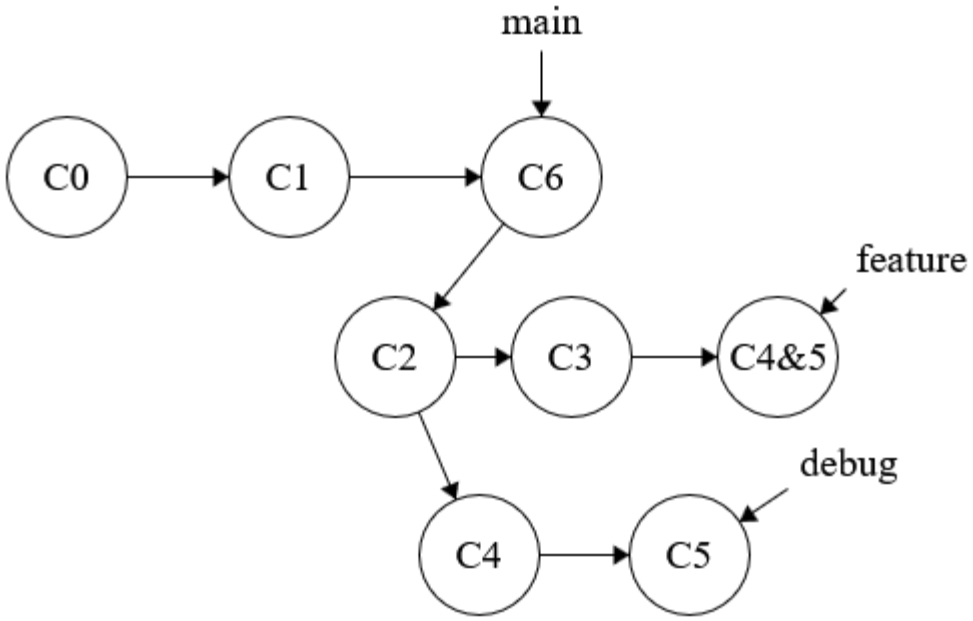
- Memory Diagram:
  - Since we didn't go over the exact internals of the linkedlist, it would have been fine to have a slightly different linked list structure (e.g. no tail_ pointer) as long as it was clear it was a linked list and the nodes were on the heap similar to how they are here:



git (pt. 1)

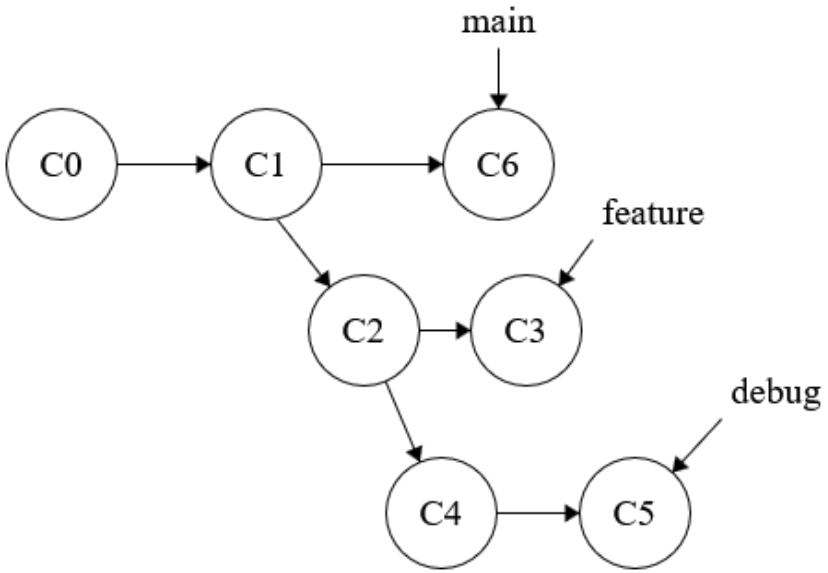
- What steps are needed to go from the git tree on the left to the tree on the right? You don't need to list the exact command, but you should be able to explain the command "close enough".
- Assume HEAD starts on main

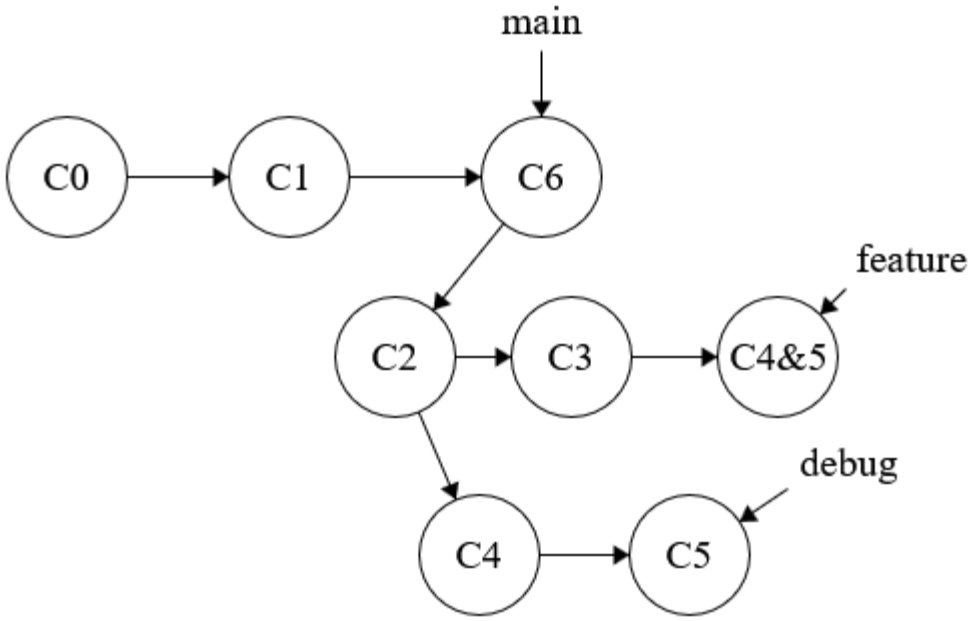




git (pt. 1)

- Exact commands:
  - `git checkout feature`
  - `git merge --squash debug`
  - `git commit` (need to commit after merging. if you forgot this we wouldn't care much)
  - `git rebase main`



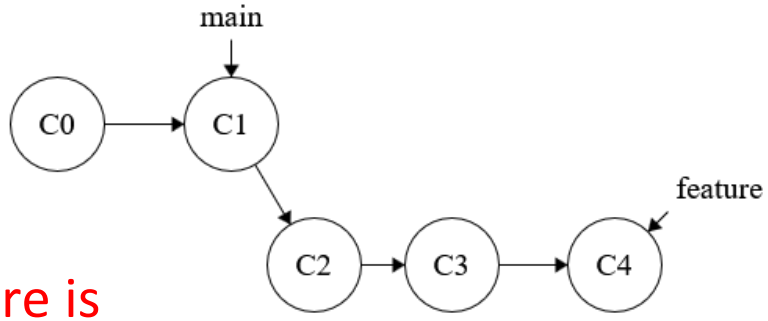


git (pt. 2)

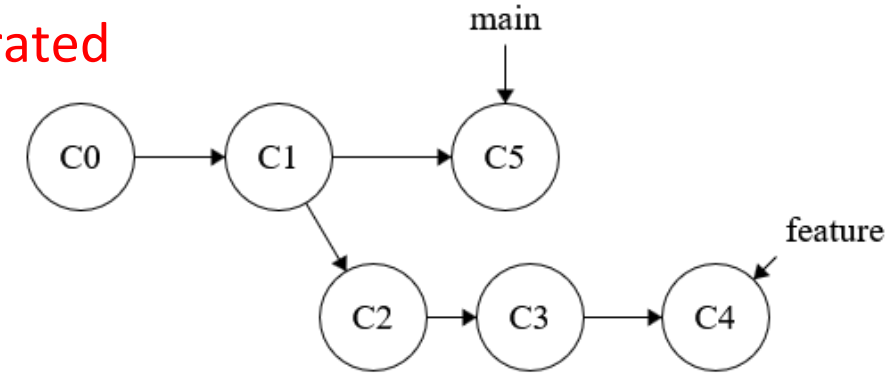
- Describe a scenario where someone may want to rebase a branch

git (pt. 2)

- Describe a scenario where someone may want to rebase a branch
  - Diagrams aren't expected in your answer but are welcome. This is here to help explain the concept to you
  - Consider the case we start working on a new branch that branched off from main (doesn't have to be main, could be another branch).



  - But then a bug fix is made in main and now there is a new commit in main! You want to keep working on your branch, and you wish your code had incorporated that new commit from the start. You want C5 to be the new "base" from which your branch came from. You can use git rebase to do this.



Locality

- For each scenario choose whether reading a file using posix read or via a std::ifstream would be faster. Briefly explain your answer.

- You need to read the first 32 bytes of the file which contains some metadata about it. Once you have finished reading the metadata, you are done with the file.

- You have a binary file containing machine code (binary encoding of assembly instructions). You read the file 64-bytes at a time so that you can read one instruction at a time.

Locality

- For each scenario choose whether reading a file using posix read or via a std::ifstream would be faster. Briefly explain your answer.

- You need to read the first 32 bytes of the file which contains some metadata about it. Once you have finished reading the metadata, you are done with the file.
  - POSIX read. It avoid the overhead of allocating a buffer that we won't benefit from since we know exactly how many bytes we want to read from the file. Though the performance gain is not big.

Locality

- For each scenario choose whether reading a file using posix read or via a std::ifstream would be faster. Briefly explain your answer.

- You have a binary file containing machine code (binary encoding of assembly instructions). You read the file 64-bytes at a time so that you can read one instruction at a time.
  - std::ifstream since it would buffer the contents of the file, so each time we read another 64 bytes, it will read it from a buffer sometimes instead of going to the filesystem. Reading from memory is a lot faster than going to the filesystem.

Caches Q1

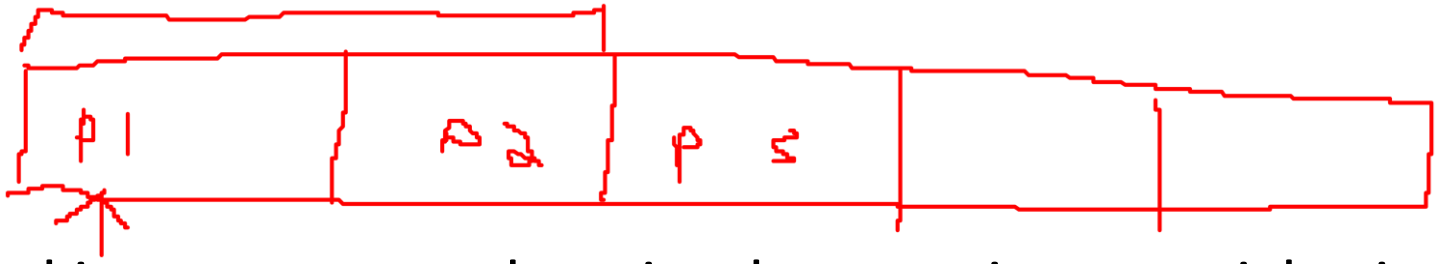
- Let's say we are making a program that simulates various particles interacting with each other. To do this we have the following structs to represent a color and a point

```
struct color {
  int red, green, blue;
};
```

```
struct point {
  double x, y;
  struct color c;
};
```

- If we were to store 100 point structs in an array, and iterate over all of them, accessing them in order, roughly how many cache hits and cache misses would we have?
  - Assume:
    - a cache line is 64 bytes
    - the cache starts empty
    - `sizeof(point)` is 32 bytes, `sizeof(color)` is 16 bytes

Caches Q1



- Let's say we are making a program that simulates various particles interacting with each other. To do this we have the following structs to represent a color and a point

```
struct color {
  int red, green, blue;
};
```

```
struct point {
  double x, y;
  struct color c;
};
```

- If we were to store 100 point structs in an array, and iterate over all of them, accessing them in order, roughly how many cache hits and cache misses would we have?
  - Assume:
    - a cache line is 64 bytes
    - the cache starts empty
    - `sizeof(point)` is 32 bytes, `sizeof(color)` is 16 bytes

Roughly every other time we access a point struct, it will already be in the cache. The other 50% of the time, it needs to be fetched from memory

Caches Q2

- Consider the previous problem with point and color structs.
- In our simulator, it turns out a VERY common operation is to iterate over all points and do calculations with their X and Y values.
- How else can we store/represent the point objects to make this operation faster while still maintaining the same data? Roughly how many cache hits would we get from this updated code?

Caches Q2

- Consider the previous problem with point and color structs.
- In our simulator, it turns out a VERY common operation is to iterate over all points and do calculations with their X and Y values.
- How else can we store/represent the point objects to make this operation faster while still maintaining the same data? Roughly how many cache hits would we get from this updated code?

Change point to just be:

```
struct point {
  double x, y;
}
```

Then Store two arrays:

```
array<point, 100> arr1;
array<color, 100> arr2;
// point at index I
// has color arr2[i]
```

Each time we access a point, we can now load 4 points into the cache. We now get ~25 cache misses and 75 hits

Cache Q3

- Typically, a **`bool`** variable is 1 byte. How much space does a **`bool`** strictly ***need*** though?
  - 1 bit

- C++ goes against the standard implementation of a vector for the bool type, and instead has each bool stored as a bit instead of the type a stand-a-lone Boolean variable would be stored as.
  - Travis thinks this was a horrible design decision, but there is a reason why they did this. What are those reasons?

Cache Q3

- Typically, a **`bool`** variable is 1 byte. How much space does a **`bool`** strictly ***need*** though?
  - 1 bit

- C++ goes against the standard implementation of a vector for the bool type, and instead has each bool stored as a bit instead of the type a stand-a-lone Boolean variable would be stored as.
  - Travis thinks this was a horrible design decision, but there is a reason why they did this. What are those reasons?
  - **A lot less space is taken up, and as a side effect of that, you probably don't have to call malloc as often and will have better cache performance**

Cache Q3

- If we stored a vector of 120 **`bool`**s, and wanted to iterate over all of them, roughly how many cache hits & misses would we have if we:
  - You can assume a cache line is 64 bytes.
  - If we used a **`vector<bool>`** that allocates the bools normally (1 byte per bool)
  - If we use a **`vector<bool>`** that represents each bool with a single bit

Cache Q3

- If we stored a vector of 120 **`bool`**s, and wanted to iterate over all of them, roughly how many cache hits & misses would we have if we:
  - You can assume a cache line is 64 bytes.
  - If we used a **`vector<bool>`** that allocates the bools normally (1 byte per bool)
    - 2 cache misses, 118 cache hits
  - If we use a **`vector<bool>`** that represents each bool with a single bit
    - 1 cache miss, 119 cache hits

IPC

- The following code intends to use a global variable so that a child process reads a string and the parent prints it.
- Briefly describe two reasons why this program won't work. You can assume it compiles.

```
string message;

void child();
void parent();

int main() {
  pid_t pid = fork();
  if (pid == 0) {
    child();
  } else {
    parent();
  }
}
void child() {
  cin >> message;
}
void parent() {
  cout << message;
}
```

IPC

- The following code intends to use a global variable so that a child process reads a string and the parent prints it.

- Briefly describe two reasons why this program won't work. You can assume it compiles.
  - After fork is called, global variables are no longer shared. Each process has its own "message"
  - There is no synchronization to know if the parent prints after the child reads.

```
string message;

void child();
void parent();

int main() {
  pid_t pid = fork();
  if (pid == 0) {
    child();
  } else {
    parent();
  }
}
void child() {
  cin >> message;
}
void parent() {
  cout << message;
}
```

IPC

- Describe how we would have to rewrite the code if we wanted it to work. Keeping the multiple processes and calls to fork(). Be specific about where you would add the new lines of code.

```
string message;

void child();
void parent();

int main() {
  pid_t pid = fork();
  if (pid == 0) {
    child();
  } else {
    parent();
  }
}
void child() {
  cin >> message;
}
void parent() {
  cout << message;
}
```

IPC

- Describe how we would have to rewrite the code if we wanted it to work. Keeping the multiple processes and calls to fork(). Be specific about where you would add the new lines of code.

- ONE POSSIBLE ANSWER:

```
string message;
int fds[2];

void child();
void parent();

int main() {
  pipe(fds);
  pid_t pid = fork();
  if (pid == 0) {
    close(fds[0]);
    child();
  } else {
    close(fds[1]);
    parent();
  }
}
void child() {
  cin >> message;
  wrapped_write(fds[1], message);
}
void parent() {
  wrapped_read(fds[0], message);
  cout << message;
}
```

Process Synchronization

- Which of the following outputs are possible? How?
  - 1213
  - 3112
  - 2312
  - 1123

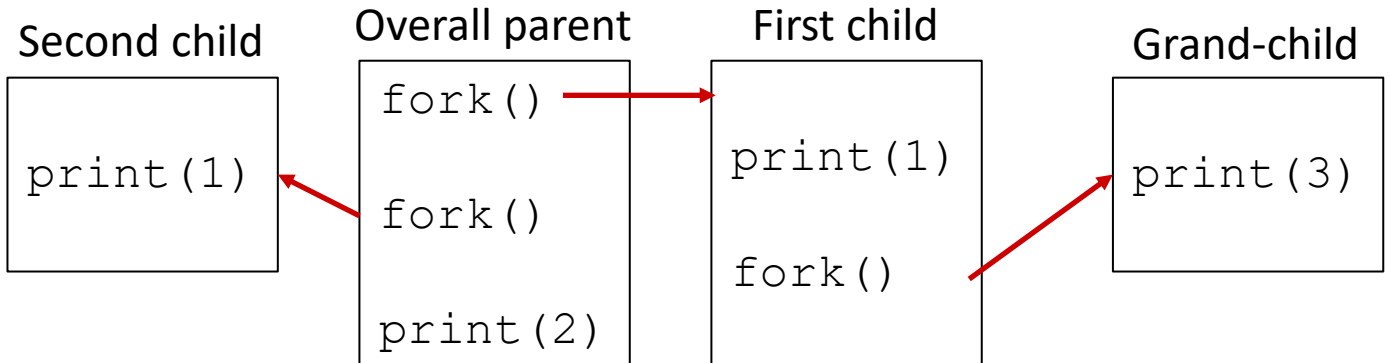
- If we wanted to change the code to guarantee that 1312 is printed. How could we do that?
  - There must still be 4 processes forked in a similar way (The initial process can't fork 3 direct children)
  - Each process must print out the same number as before.

```
int main() {
  pid_t pid = fork();
  bool flag = false
  if (pid == 0) {
    flag = true;
    cout << "1" << endl;
  }

  pid = fork();

  if (pid == 0) {
    if (flag) {
      cout << "3" << endl;
    } else {
      cout << "1" << endl;
    }
  } else if (!flag) {
    cout << "2" << endl;
  }
}
```

Process Synchronization

- Which of the following outputs are possible? How?
  - 1213 Possible
  - 3112 Not Possible
  - 2312 Not Possible
  - 1123 Possible

- If we draw the processes and ordering within a process we get:
  - Within each process the events must happen in that order. So the first child must print(1) before it forks the child that prints 3, so there must be a 1 printed before 3 is printed.



Process Synchronization

- If we wanted to change the code to guarantee that 1312 is printed. How could we do that?
  - There must still be 4 processes forked in a similar way (The initial process can't fork 3 direct children)
  - Each process must print out the same number as before.

  - One possible answer:
    - Main thing: using waitpid to enforce ordering
    - Make first child fork its own child before "earlier" so that when parent waits for it, it implicitly guarantee that both child and grandchild have finished.
    - Make children processes exit to make sure it doesn't continue running code it shouldn't run.

```
int main() {
  pid_t pid = fork();
  bool flag = false
  if (pid == 0) {
    flag = true;
    cout << "1" << endl;
    pid = fork();
    if (pid == 0) {
      cout << "3" << endl;
      exit(EXIT_SUCCESS);
    }
    waitpid(pid, NULL, 0);
    exit(EXIT_SUCCESS);
  }
  waitpid(pid, NULL, 0);
  pid = fork();

  if (pid == 0) {
    cout << "1" << endl;
    exit(EXIT_SUCCESS);
  }
  waitpid(pid, NULL, 0);
  cout << "2" << endl;
}
```

Threads & Data Races

- Consider the following pseudocode that uses threads. Assume that file.txt is large file containing the contents of a book. Assume that there is a **main()** that creates one thread running **first_thread()** and one thread for **second_thread()**

```
string data = "";  // global

void* first_thread(void* arg) {
  f = open("file.txt", O_RDONLY);
  while (!f.eof()) {
     string data_read = f.read(10 chars);
     data = data_read;
  }
}
void* second_thread(void* arg) {
  while (true) {
    if (data.size() != 0) {
      print(data);
    }
    data = "";
  }
}
```

- There is a data race.
  How do we fix it using just a mutex?
  (where do we add calls to lock and unlock?)

Threads & Data Races

- There is a data race. How do we fix it using just a mutex? (where do we add calls to lock and unlock?)

```
string data = "";  // global

void* first_thread(void* arg) {
  f = open("file.txt", O_RDONLY);
  while (!f.eof()) {
     string data_read = f.read(10 chars);
     data = data_read;
  }
}
void* second_thread(void* arg) {
  while (true) {
    if (data.size() != 0) {
      print(data);
    }
    data = "";
  }
}
```

Threads & Data Races

- There is a data race. How do we fix it using just a mutex? (where do we add calls to lock and unlock?)

```
string data = "";  // global
pthread_mutex_t mutex;

void* first_thread(void* arg) {
  f = open("file.txt", O_RDONLY);
  while (!f.eof()) {
     string data_read = f.read(10 chars);
     pthread_mutex_lock(&mutex);
     data = data_read;
     pthread_mutex_unlock(&mutex);
  }
}
```

Threads & Data Races

- There is a data race. How do we fix it using just a mutex? (where do we add calls to lock and unlock?)

```
string data = "";  // global
pthread_mutex_t mutex;

void* second_thread(void* arg) {
  while (true) {
    pthread_mutex_lock(&mutex);
    if (data.size() != 0) {
      print(data);
    }
    data = "";
    pthread_mutex_unlock(&mutex);
  }
}
```

Threads & Data Races

- After we remove the data race on the global string, do we have deterministic output? (Assuming the contents of the file stays the same).

```
string data = "";  // global

void* first_thread(void* arg) {
  f = open("file.txt", O_RDONLY);
  while (!f.eof()) {
     string data_read = f.read(10 chars);
     data = data_read;
  }
}

void* second_thread(void* arg) {
  while (true) {
    if (data.size() != 0) {
      print(data);
    }
    data = "";
  }
}
```

Threads & Data Races

- After we remove the data race on the global string, do we have deterministic output? (Assuming the contents of the file stays the same).
  - No, we could still have a difference in output depending on when threads are run. It is possible a the first thread overwrites the global before second thread reads it

    This is the distinction between a data race and a race condition (more on race condition vs data race after exam)

```
string data = "";  // global

void* first_thread(void* arg) {
  f = open("file.txt", O_RDONLY);
  while (!f.eof()) {
     string data_read = f.read(10 chars);
     data = data_read;
  }
}

void* second_thread(void* arg) {
  while (true) {
    if (data.size() != 0) {
      print(data);
    }
    data = "";
  }
}
```

Threads & Data Races

- There is an issue of inefficient CPU utilization going on in this code. What is it?

```
string data = "";  // global

void* first_thread(void* arg) {
  f = open("file.txt", O_RDONLY);
  while (!f.eof()) {
     string data_read = f.read(10 chars);
     data = data_read;
  }
}

void* second_thread(void* arg) {
  while (true) {
    if (data.size() != 0) {
      print(data);
    }
    data = "";
  }
}
```

Threads & Data Races

- There is an issue of inefficient CPU utilization going on in this code. What is it?

  - Busy waiting possible in second_thread. It will keep looping and executing the code over and over again even if the data is not ready.

```
string data = "";  // global

void* first_thread(void* arg) {
  f = open("file.txt", O_RDONLY);
  while (!f.eof()) {
     string data_read = f.read(10 chars);
     data = data_read;
  }
}

void* second_thread(void* arg) {
  while (true) {
    if (data.size() != 0) {
      print(data);
    }
    data = "";
  }
}
```