CS 211 : Thurs 03/07 (lecture 19)

Prof. Hummel
(he/him)

- **Topics: last day! Modern languages looking to replace C and C++ --- Go and Rust**

MARCH 2024

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

www.a-printable-calendar.com

Notes:

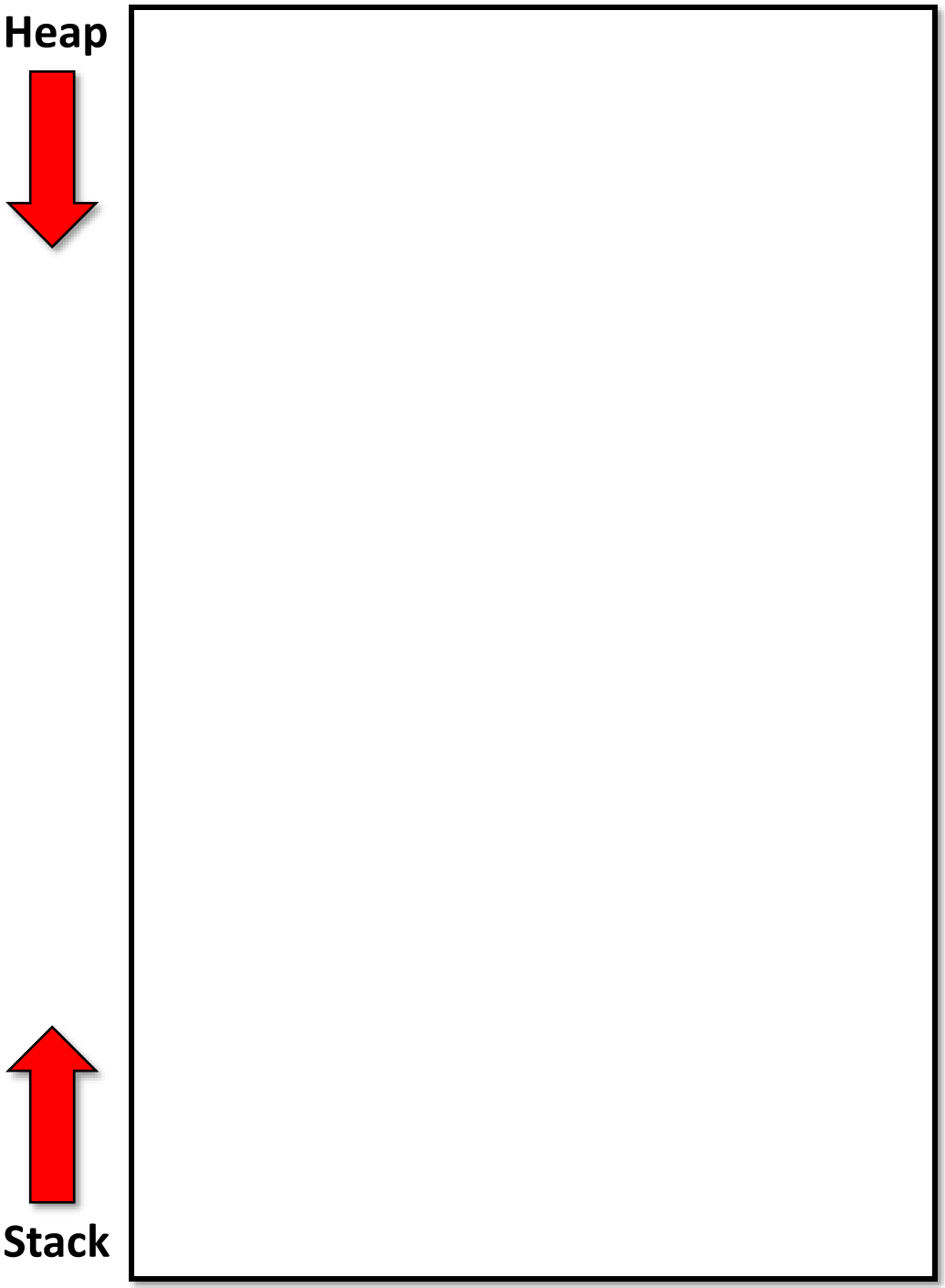
- *Lecture slides available on Canvas*
- ***Project 08** due Friday night (can submit as late as Sunday with late days / penalty).*
- ***Final exam**: Tues 3/12 (noon) or Fri 3/15 (noon)*
  - *You can take exam on either day*
  - *Diagram memory; write a C++ program (with C?)*
  - *Hand-written, no computer, no internet, no notes*

Northwestern University

Diagram memory

```
int main() {
  int     x;
  char*   p;
  string  s;
  map<int,string> M;

  x = 12;

  p = (char*) malloc(5);
  strcpy(p, "A3");

  s = "apple";

  M[456] = "cat";
  M[123] = "dog";
  M[789] = "bird";
```

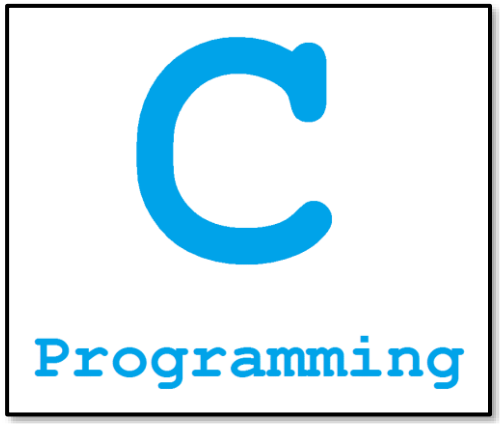
```
  int     x2;
  char*   p2;
  string  s2;
  map<int,string> M2;

  x2 = x;
  p2 = p;
  s2 = s;
  M2 = M;
```



Observations

- **When you copy a C variable, the bits are copied**
  - *Known as a "shallow" copy*
- **When you copy a C++ object, the bits are copied + what those bits point to**
  - *Known as a "deep" copy*
  - *Classes need* ***copy constructor****, also overload* ***operator=***


C++


C
Programming

C++ and C are dangerous

- **Just ask the White House ☺**

https://www.infoworld.com/article/3713203/white-house-urges-developers-to-dump-c-and-c.html

White House urges developers to dump C and C++

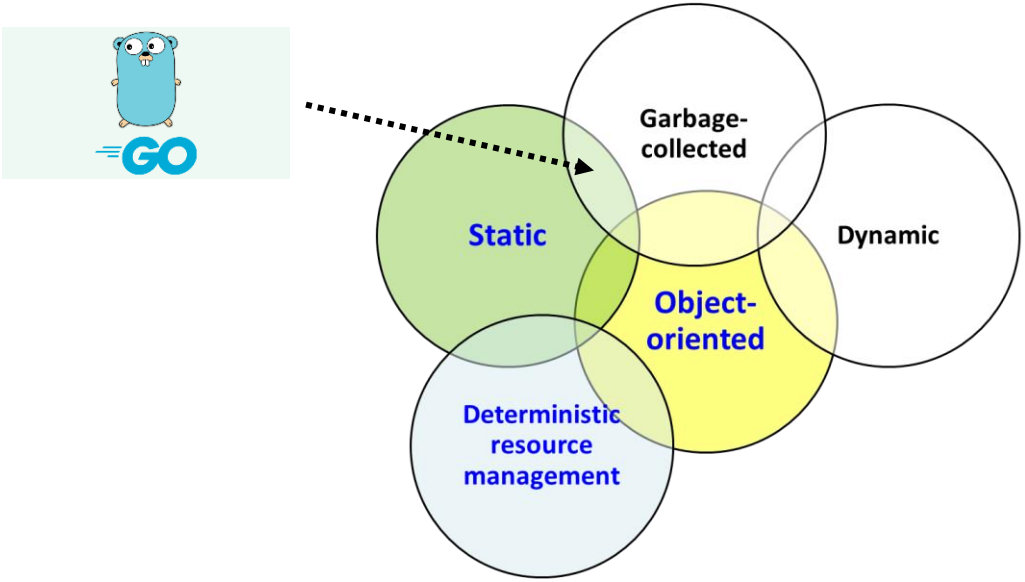
Biden administration calls for developers to embrace memory-safe programing languages and move away from those that cause buffer overflows and other memory access vulnerabilities.

Companies using Go

Organizations in every industry use Go to power their software and services View all stories

The Go programming language

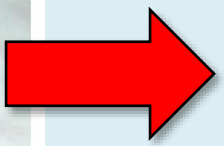
- **Go was created by Google as a replacement for C**
  - *Fast and low-level like C*
  - *Adds garbage collection (no more calls to free!)*
  - *Support for concurrent programming & networking*



Demo

- **Converting images to grayscale…**

File: cake.ppm



File: cake-grayscale.ppm

Sequential version

- **For the most part, Go == C**

```
func main() {
  fmt.Printf("PPM image file> ");

  var filename string;
  fmt.Scanln(&filename);

  fmt.Println(fmt.Sprintf("Reading '%s'...", filename));

  w, h, d, pixels := ReadImageFile(filename);

  fmt.Println("Converting to grayscale...");

  grayscale(w, h, d, pixels);

  filename = strings.Replace(filename, ".ppm", "-grayscale.ppm", 1);

  fmt.Println(fmt.Sprintf("Writing '%s'...", filename));

  WriteImageFile(filename, w, h, d, pixels);

  fmt.Println("done");
}
```

Parallel version

- Use "execution threads" to process rows simultaneously…

File: cake.ppm

Built-in concurrency / parallelism

- **The most important feature of Go is concurrency**
- **Go is one of the few languages where concurrency was designed in from the start --- not added later**

```
func grayscale(width int, height int, depth int, pixels [][]int) {

  for r := range pixels {
     grayOneRow(width, pixels[r]);
  }

}
```

```
func grayscale(width int, height int, depth int, pixels [][]int) {

  for r := range pixels {
     go grayOneRow(width, pixels[r]);
  }

}
```

run the function call concurrently with main thread

Question

- **It compiles and runs, but is it correct?**

```
func grayscale(width int, height int, depth int, pixels [][]int) {

  for r := range pixels {
     go grayOneRow(width, pixels[r]);
  }

}
```

A) yes, that's the point of Go --- parallelism should be easy

B) yes, it works but for large images the above creates too many worker threads and slows down

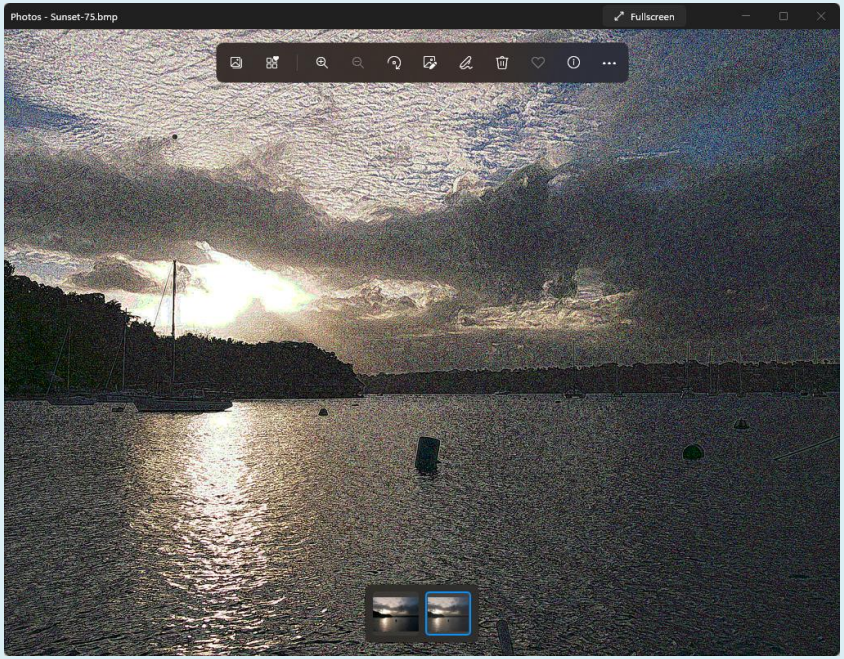
C) no, the above fails --- parallelism is never this easy

You have to wait for threads to finish…

```
var wg sync.WaitGroup  // create a waitgroup object

for ... {

   wg.Add(1);  // add one more to group that is running:

   go func() { // start worker thread
     defer wg.Done();

     << do something work >>

   }();

}

wg.Wait();  // wait for ALL threads to finish:
```

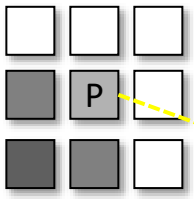
Demo #2

- Contrast stretching…
- Increase image contrast to “pull” details from the dark



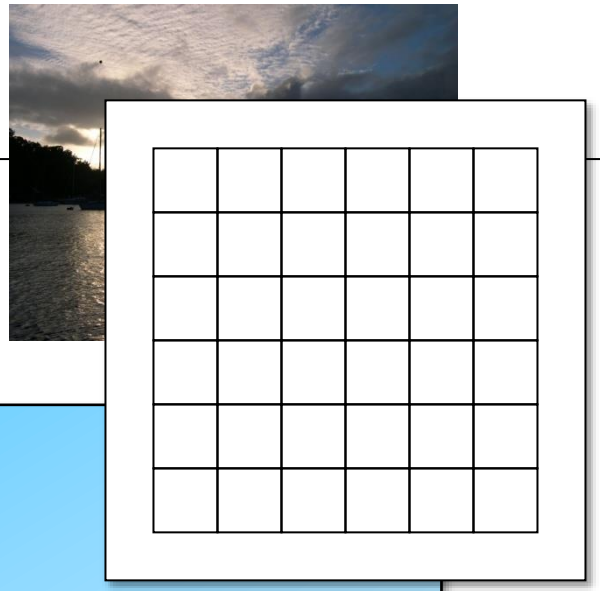
Contrast stretching

- *Contrast stretching:*
  - *increase image contrast by making lighter areas lighter / darker areas darker*
  - *for all pixels P, update P based on its 8 neighbors*
  - *repeat…*



Sunset.bmp

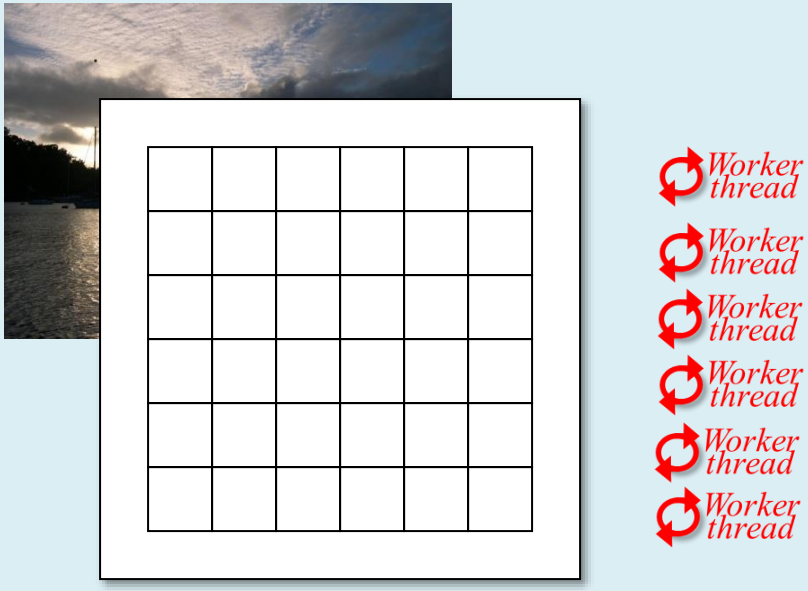
Algorithm



```
while (step < NumSteps && !converged)
{
  step++; diffs = 0;

  foreach (non-boundary row r of M)   /* for each pixel, stretch… */
    foreach (non-boundary column c of M)
    {
       M'[r][c] += stretch(M[r-1][c-1], M[r-1][c], M[r-1][c+1],
                           M[r][c-1],   M[r][c],   M[r][c+1],
                           M[r+1][c-1], M[r+1][c], M[r+1][c+1]);
       if (M'[r][c] != M[r][c])
         diffs++;
    }

  converged = (diffs == 0);

  foreach (non-boundary row r of M)   /* update original matrix with new values */
    foreach (non-boundary column c of M)
      M[r][c] = M'[r][c];
}
```

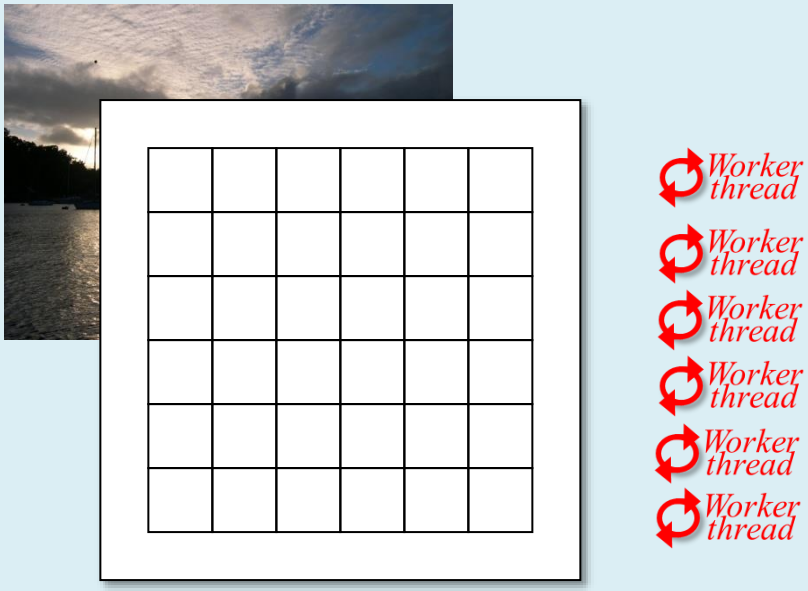
Parallel version

- **Process the rows in parallel…**



Question

- **Given the parallel version, which of the following is the best answer?**



A) the **more threads** we create, the faster it goes

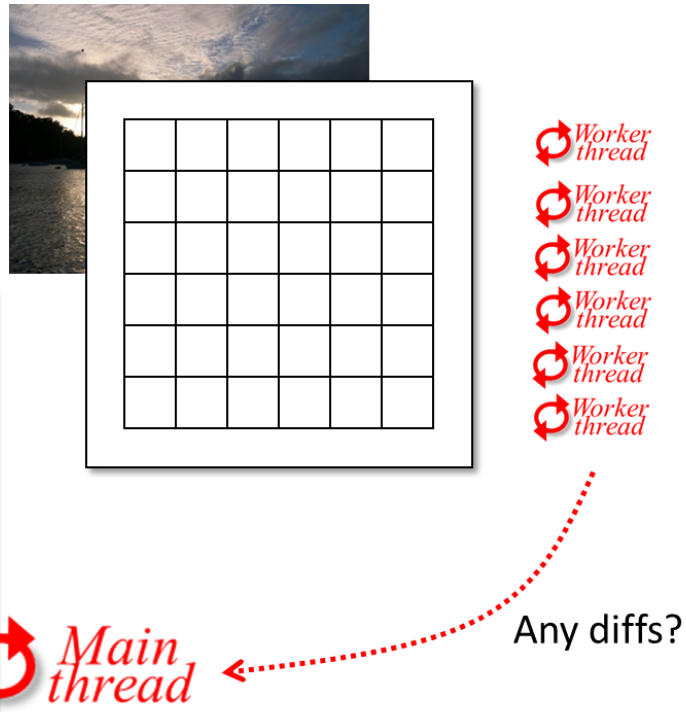
B) the **more cores** we have, the faster it goes

C) the **more memory** we have, the faster it goes

Avoiding memory errors

- **We have to avoid two threads reading/writing the same memory at the same time**
  1. *Threads read from M and write to a new matrix M'*
  2. *Threads send "diffs" to main thread via msg-passing*

```
while (step < NumSteps && !converged)
{
  step++; diffs = 0;

  foreach (non-boundary row r of M)    /* for each pixel, stretch... */
    foreach (non-boundary column c of M)
    {
       M'[r][c] += stretch(M[r-1][c-1], M[r-1][c], M[r-1][c+1],
                           M[r][c-1],   M[r][c],   M[r][c+1],
                           M[r+1][c-1], M[r+1][c], M[r+1][c+1]);
       if (M'[r][c] != M[r][c])
         diffs++;
    }

  converged = (diffs == 0);

  foreach (non-boundary row r of M)    /* update original matrix with new values */
    foreach (non-boundary column c of M)
      M[r][c] = M'[r][c];
}
```



Top Apps Built With Rust

Rust is used to build apps because it is a popular programming language. It helps prevent memory issues and application performance issues. Below are some of the apps that were built using rust.

Dropbox

Dropbox's file-syncing engine is partially built with Rust code. Writing, testing, and debugging the engine is difficult due to its high concurrency. When dealing with concurrent code and complex programs, Rust's static types and comprehensive compile-time checks outperform dynamically typed languages such as Python.

Figma

Figma is a web-based design tool for vector graphics and interface prototypes. Figma updated their multiplayer synchronization engine from Typescript to Rust to improve performance because their server could not keep up with user growth.

Cloudflare

Rust is a tool used by Cloudflare in its core edge logic. Rust functions as a substitute for C, which is memory-unsafe. Cloudflare agrees that Rust is a strong option for adequate performance.

Facebook

The source control backend for Facebook, now known as Meta, was originally written in Python, but was rewritten in Rust. Meta needed a compiled language to rewrite their source control backend, and Rust appealed to them because of its security features. The source control team has adopted Rust since then.

Discord

The client and server sides of Discord's codebase are both written in Rust. For instance, the team could scale to 11 million concurrent users by utilizing Elixir Native Implemented Functions (NIFs). Additionally, Discord has rewritten their Read States service, which was originally written in Go, in the Rust programming language.

The Rust programming language

- **Rust was created by Mozilla as a safer C++**
  - *Fast, efficient language like C++*
  - *No memory errors*
  - *No memory leaks*

C++ memory errors

- **C++ is better than C, but memory errors still possible**

```
int main()
{
  cout << "starting" << endl;

  set<int> S;
  S.insert(22);
  S.insert(35);
  S.insert(15);

  auto iter = S.find(15);

  cout << *iter << endl;









  cout << "done" << endl;

  return 0;
}
```

Rust

- **Rust introduces the concept of ownership**
  - *Only one "owner" of an object / memory*

```
int main()
{
  cout << "starting" << endl;

  set<int> S;
  S.insert(22);
  S.insert(35);
  S.insert(15);

  set<int> S2 = S;
```

C++ makes a copy, there are now two trees

In Rust, S2 now owns the tree and S is invalid --- S cannot be used

Rust

- **Others can "borrow" a reference to an object / memory**
  - *But memory cannot be freed if there are references*

```
int main()
{
  cout << "starting" << endl;

  set<int> S;
  S.insert(22);
  S.insert(35);
  S.insert(15);

  auto iter = S.find(15);

  cout << *iter << endl;

  S.erase(15);
  iter++;
  cout << *iter << endl;

  cout << "done" << endl;

  return 0;
}
```

C++ allows the deletion, causing a program crash

Rust does not allow the deletion, throwing an exception

Examples

- **Here's what actual Rust code looks like…**

https://swallez.github.io/introduction-to-rust/slide-15.html

https://swallez.github.io/introduction-to-rust/slide-32.html

- **Thank you everyone**
- **You worked hard, well done!**

What's due?

Project 08 *is due Friday night*

Extra credit C++ project? Plan to release something next week…