Objectives - Tue 4/9/2019

- HOWTO: X forwarding with PuTTY + Xming
- Unit testing
- Parallel programming (brief introduction)

⚠ There will be a gratuitous image of animal remains in today's lecture.

These slides were significantly improved after the 12pm lecture (section 2). Content is the same, except for the section on types of test coverage and addition of example code (and the gory image).

HOWTO:
X forwarding with PuTTY + Xming

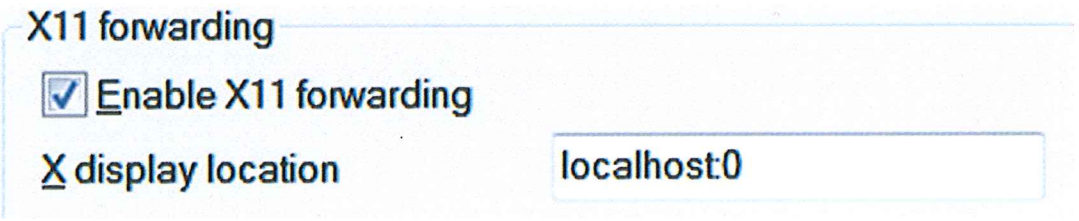
HOWTO: X forwarding on Windows

- Install and run Xming
  - 

- In bash:

```
$ display 6x6_24bit.bmp
```

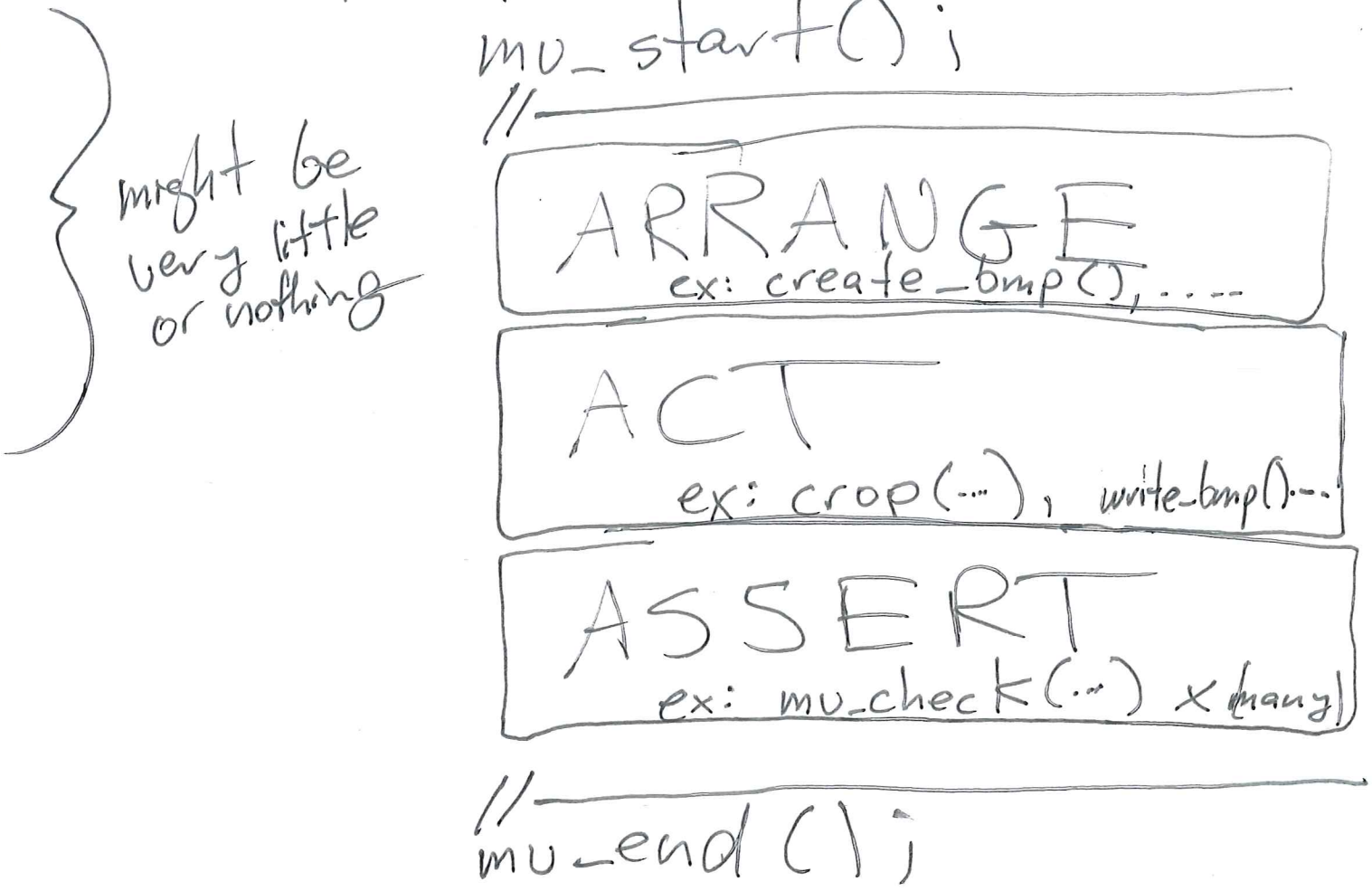
- If this doesn't work, double-check your setup that you did in Week 1. In particular…



Unit testing

3 A's

- Arrange
- Act
- Assert



Also known as the "AAA (Arrange-Act-Assert)" pattern

"The only way to make the deadline — the only way to go fast — is to keep the code as clean as possible at all times."

Credit: Robert C. Martin, from the book *Clean Code*

Order

- Tests should be able to run in any order
  - Ex: test_read(...) should not depend on test_write(...)
  - It shouldn't matter if you run...

```
mu_run( test_write );
mu_run( test_read );
```

... or ...

```
mu_run( test_read );
mu_run( test_write );
```

- You should be able to comment out some tests without affecting others
  - Normally, you should be running all tests together
  - Need enough support code so each test is indepdendent.
- Every test should start with a clean slate

No manual inspection required

- The tests should be able to run on their own
  - Running all tests should require no human effort.

- This is the foundation of regression testing
  - Regression testing means running all tests whenever something changes and/or periodically (e.g., nightly).

Bugs vs. run-time error handling

- "Bugs" are flaws in your code.
  - Ex: You forgot to check for something.

- "Run-time error handling" means ensuring that the program behaves in a way that is helpful to the user, even when it receives unexpected or malformed inputs
  - Ex: malformed BMP header

"Support functions" vs. "Helper functions"

For purposes of HW12 in ECE 264 (Spring 2019):

- "Support function" is used much like a helper function, but may be tested by external code (i.e., for the homework)
  - set_pixel(...) and create_bmp(...)
  - Note: "Support function" is not standard terminology.
- "Helper function"
  - _[illegible](...)
  - Not expected to be accessed by any external code.
  - This is standard terminology.

```
////// IMPLEMENTATION CODE //////
void report_weather(bool is_sunny, bool is_raining) {
    if(is_sunny) {
        printf("The sun is shining.\n");
    }
    else {
        printf("The sun is not shining.\n");
    }

    if(is_raining) {
        printf("It is raining.\n");
    }
}

////////// TEST CODE //////////
void test_report_weather_1() { // LINE coverage
    report_weather(true,  true);  // The sun is shining.  It is raining.
    report_weather(false, true);  // The sun is not shining.  It is raining.
}

void test_report_weather_2() { // BRANCH coverage
    report_weather(true,  true);  // The sun is shining.  It is raining.
    report_weather(false, false); // The sun is not shining.
}

void test_report_weather_3() { // PATH coverage
    report_weather(true,  true);  // The sun is shining.  It is raining.
    report_weather(true,  false); // The sun is shining.  It is raining.
    report_weather(false, true);  // The sun is not shining.  It is raining.
    report_weather(false, false); // The sun is not shining.
}
```

Types of test code coverage

- "Line coverage" means every line of the code being tested was executed at least once.
- "Branch coverage" means for every conditional jump (If/While/For/Switch), we took the jump (condition true) and did not take the jump (condition false) at least once.
- "Path coverage" means we tested every possible path through the code (unique combination of branches). *This can be hard.*

line coverage ⊆ branch coverage ⊆ path coverage

not checked in ECE 264

Thinking of test cases

- Easy cases
  - Answer is obvious (to you). If the test fails, you should have no doubt in your mind about whether the test itself is correct or not.
  - Ex: print_integer(5, 10)
- "Edge cases" (boundaries)
  - Extreme values for inputs (e.g., parameters, input files, etc.).
  - Ex: print_integer(INT_MIN, 10)
- "Corner cases" (turning points)
  - Look for `if(▒){…}`, `while(▒){…}`, `for(▒){…}`, and `▒?▒:▒` in your code
  - Will be captured whenever you have 100% branch coverage (hard)
  - Ex: print_integer(0, 10); print_integer(10, 16); print_integer(9, 16);
- Special cases (look for "except" in spec)
  - Look for words like "… except when…" or "Note: If …" in the specification.
  - Ex: mintf("%")

Note: This is not standard terminology. These are the instructor's invented terms.

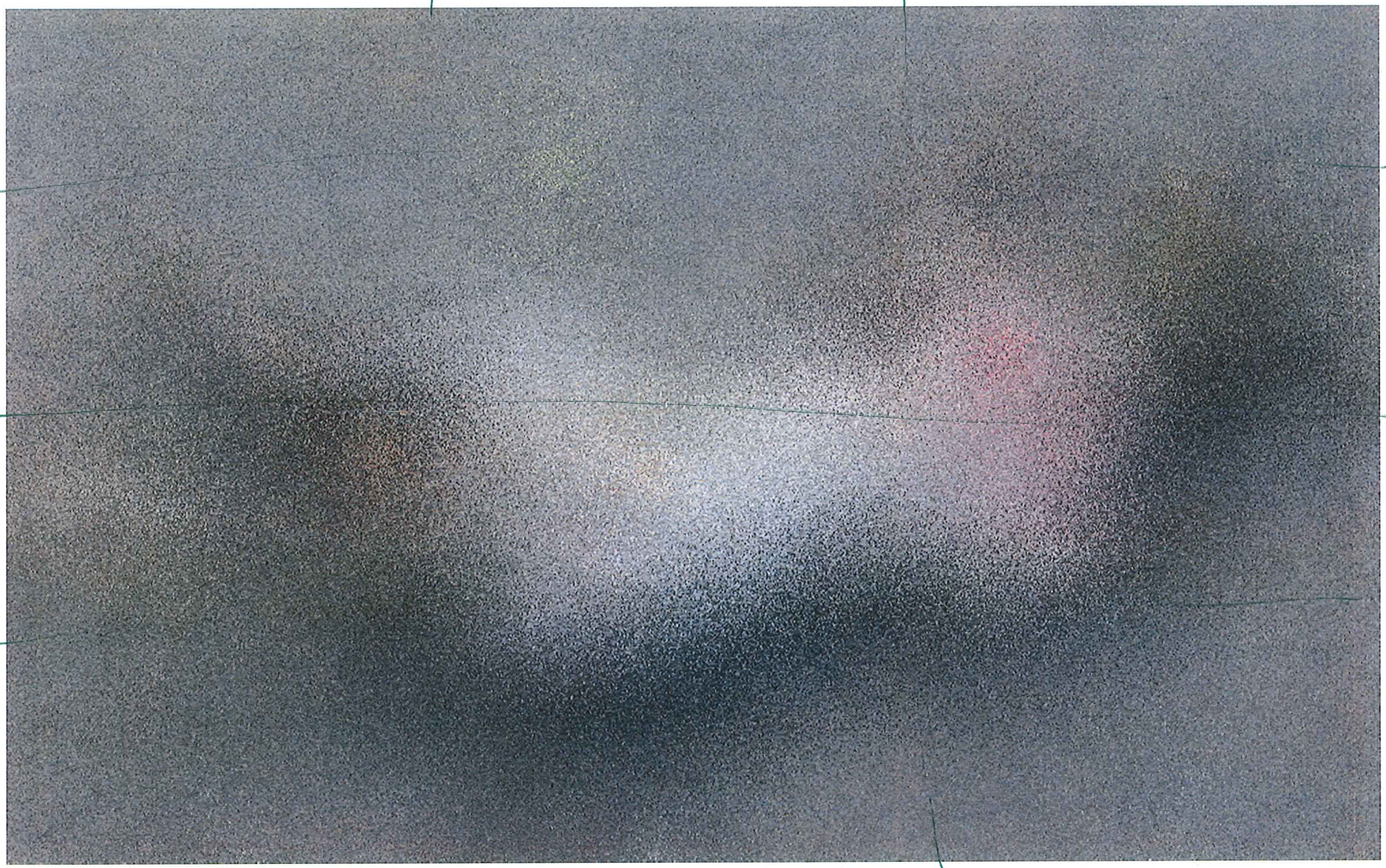
Parallel programming

The next slide is disgusting

Need for parallel processing

credit: 'Achird' @ Wikipedia CC-SA-4.0

Need for parallel processing



Parallel programming

processes	threads
Often an application that you started explicitly Ex: ./test_bmp Ex: vim test_bmp.c Ex: bash	Created by a process to perform part of the work that process is responsible for Ex: Background spell-check Ex: Process part of an image
Memory is separate from all other processes	Shares memory with other threads of the same process

Application of threads

- Suppose you need to run an image filter on a 4000x3000-pixel image.
- Each pixel takes 1 microsecond (10^{-6} secs).
- If we process sequentially (one thread):
  - ≥12 seconds
- If we process in parallel with 12 threads:
  - ≥1 second per 1000x1000-pixel segment
  - ≥1 second to process entire image
- Note: We assume each pixel is independent.