Race conditions

Definition and Example: Race condition

- Different tasks in the program have operations that can interfere with each other depending on the order in which they are performed

- First example: Commercial kitchen with many chefs
  - All need to use a single cutting board, which they clean after use
  - Works fine if each chef finishes before the next starts, but contaminates food if two try to use it together
  - Solution: Buy more cutting boards

Cutting board represents shared variable that shouldn't be shared

Second example: Chefs who need to interact

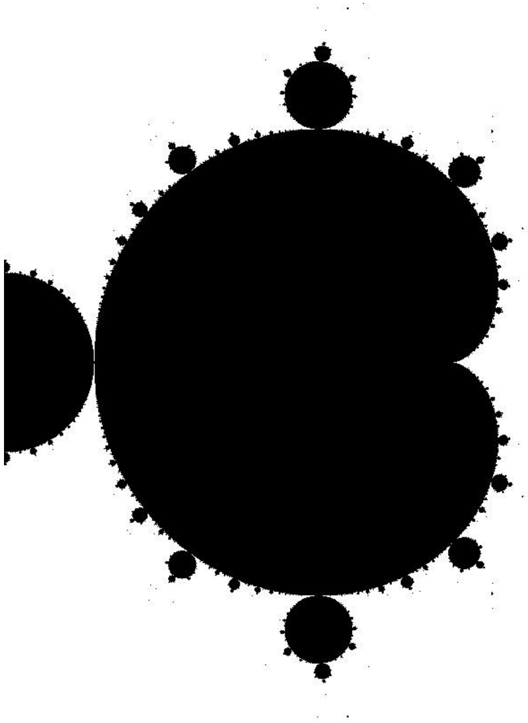
- Chefs making many-layered opera cake
    - Layers can be made separately, but can't all try to deposit on serving plate at once
    - Called a reduction: many contributions combined into one final product

Image: Arnold Gatilao, via Wikipedia

Race conditions in code: Mandelbrot set

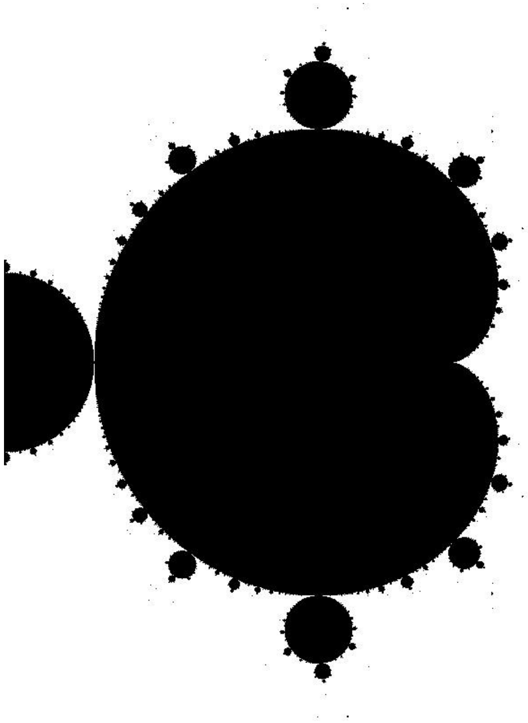
- Fractal image composed by black and white pixels

- For our purposes, just assume we have a function mandelbrot that takes real-valued coordinates and returns whether that pixel is in the set (i.e. is black)



Race conditions in code: Mandelbrot set

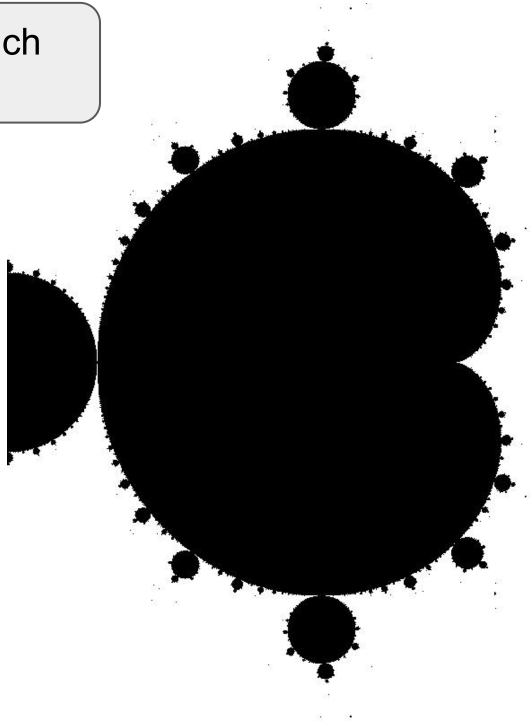
```
for (int j = 0; j < numRows; j++) {
    for (int i = 0; i < numCols; i++) {
        x = ((double)i / numCols -0.5) * 2;
        y = ((double)j / numRows -0.5) * 2;

        pixels[i][j] = mandelbrot(x,y);
    }
}
```



Race conditions in code: Mandelbrot set

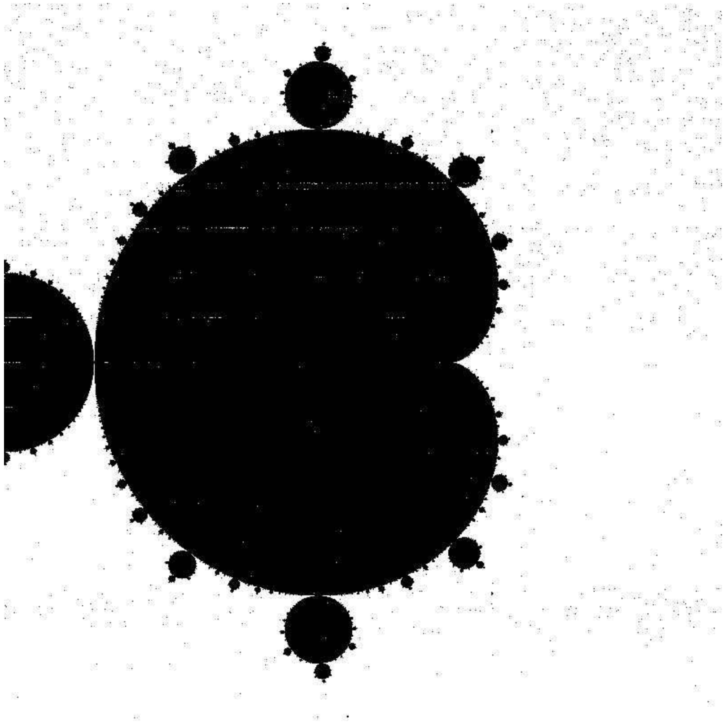
Creates multiple threads, which split iterations of the j loop

```
#pragma omp parallel for
for (int j = 0; j < numRows; j++) {
    for (int i = 0; i < numCols; i++) {
        x = ((double)i / numCols -0.5) * 2;
        y = ((double)j / numRows -0.5) * 2;

        pixels[i][j] = mandelbrot(x,y);
    }
}
```



Parallel output

```
#pragma omp parallel for
for (int j = 0; j < numRows; j++) {
     for (int i = 0; i < numCols; i++) {
          x = ((double)i / numCols -0.5) * 2;
          y = ((double)j / numRows -0.5) * 2;

          pixels[i][j] = mandelbrot(x,y);
     }
}
```



Solving the race

- Need to give each thread its own copies of `x` and y
- Two approaches:
  - Add qualifier to pragma

    ```
    #pragma omp parallel for private(x,y)
    ```

  - Declare variables inside the loop

    ```
    for (int i = 0; i < numCols; i++) {
        double x = ((double)i / numCols -0.5) * 2;
        double y = ((double)j / numRows -0.5) * 2;
        ...
    ```

Race involving a reduction

```
#pragma omp parallel for private(x,y)
for (int j = 0; j < numRows; j++) {
    for (int i = 0; i < numCols; i++) {
        x = ((double)i / numCols -0.5) * 2;
        y = ((double)j / numRows -0.5) * 2;

        int pixel = mandelbrot(x,y);
        pixels[i][j] = pixel;
        if(pixel == 0)
            numBlack++;
    }
}
```

How the race can happen

C code	Actual operations
numBlack++	Load into register
	Increment register
	Store from register

How the race can happen

C code	**First task**	**Second task**
numBlack++	Load into register	
	Increment register	Load into register
	Store from register	Increment register
		Store from register

Fixing the race

Variable into which results are combined

```
#pragma omp parallel for private(x,y) reduction(+:numBlack)
for (int j = 0; j < numRows; j++) {
    for (int i = 0; i < numCols; i++) {
        x = ((double)i / numCols -0.5) * 2;
        y = ((double)j / numRows -0.5) * 2;

        int pixel = mandelbrot(x,y);
        pixels[i][j] = pixel;
        if(pixel == 0)
            numBlack++;
    }
}
```

Operation used to combine them