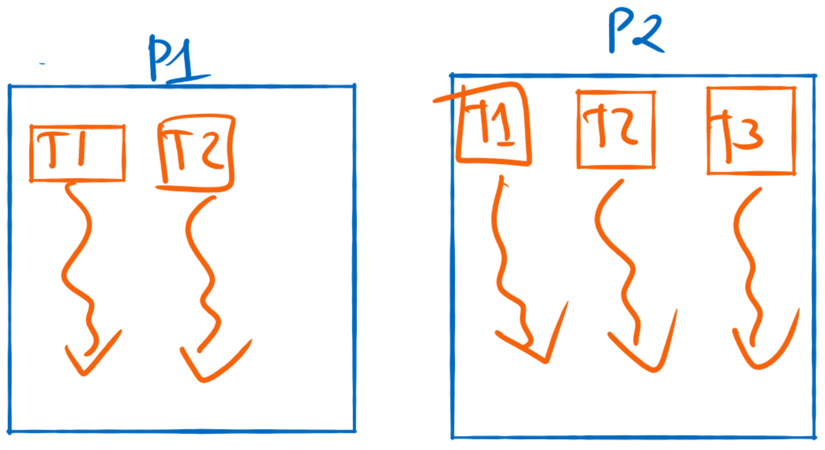
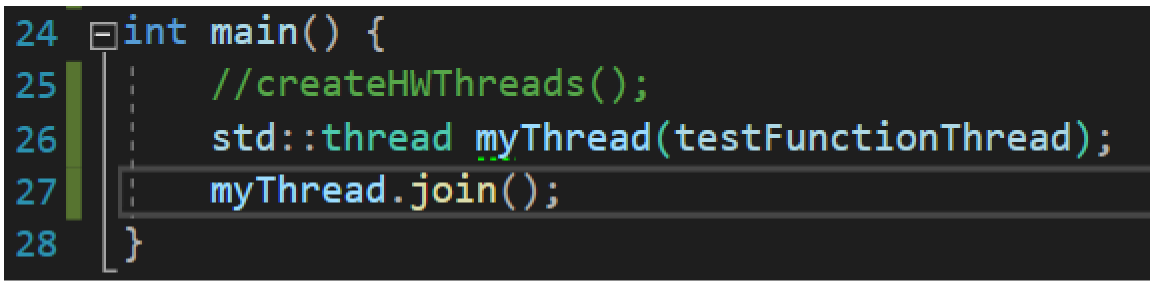
Recall that:



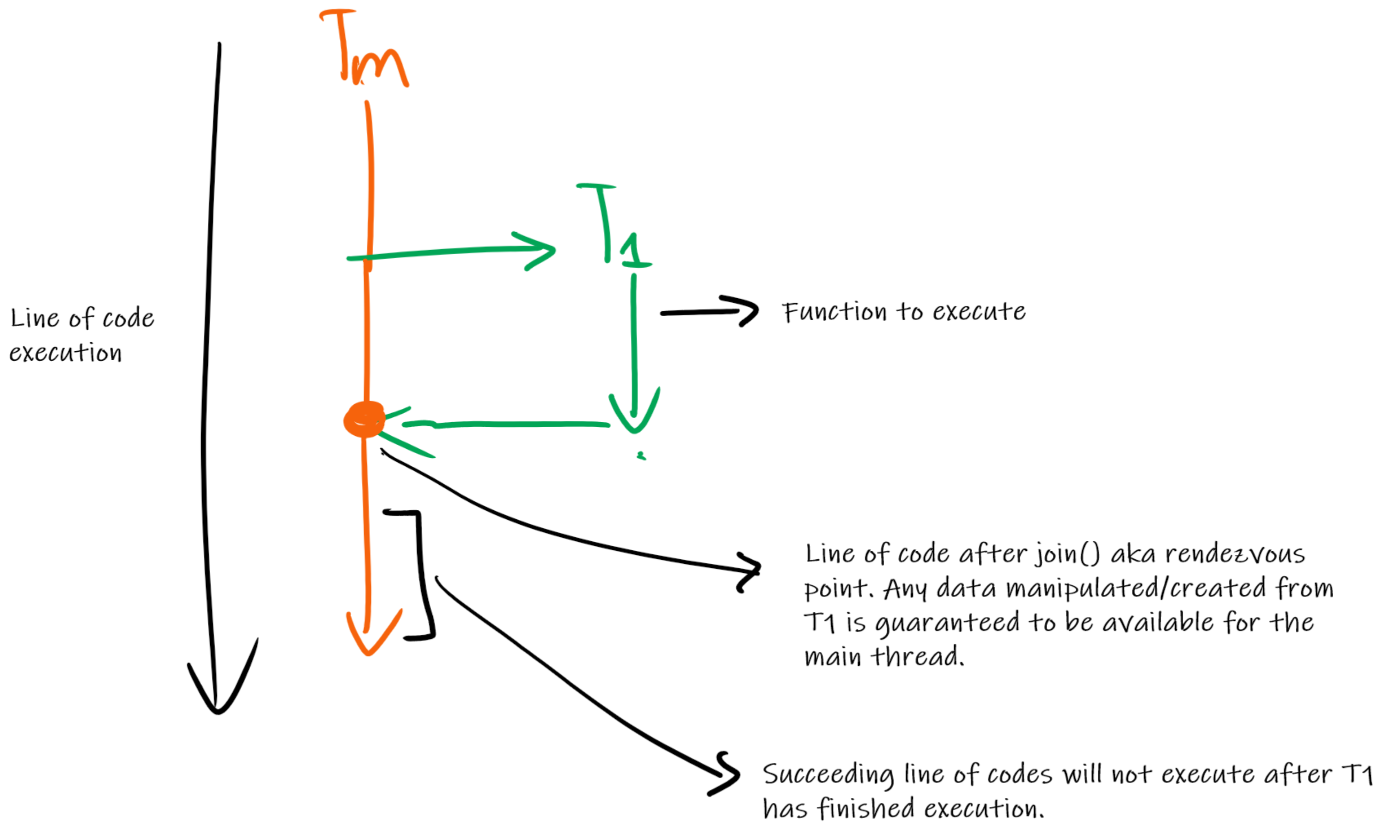
- A thread is a lightweight process
- A sequence of instructions to execute in parallel with another thread.
- Since it is within a process, memory is shared.
- Multiple threads can access the same data structure, functions, etc.

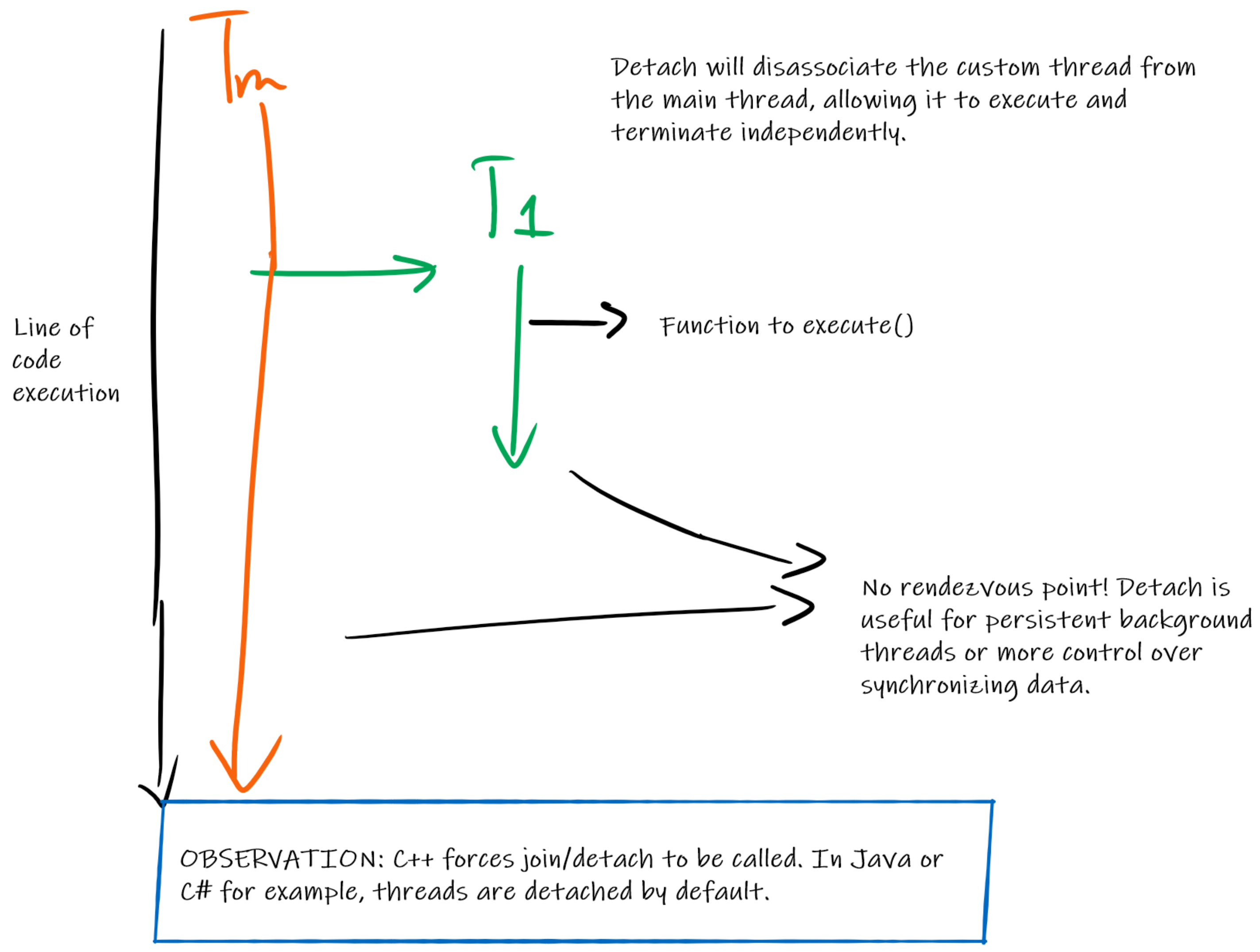
Discussing previous hello world thread. C++ has a guarding mechanism that forces the developer to call either join() or detach(). Why?



```cpp
int main() {
    //createHWThreads();
    std::thread myThread(testFunctionThread);
    myThread.join();
}
```

Based on C++'s compiler design, all threads must call either join() or detach().


Tm
T1
Line of code execution
Function to execute
Line of code after join() aka rendezvous point. Any data manipulated/created from T1 is guaranteed to be available for the main thread.
Succeeding line of codes will not execute after T1 has finished execution.



Detach will disassociate the custom thread from the main thread, allowing it to execute and terminate independently.

No rendezvous point! Detach is useful for persistent background threads or more control over synchronizing data.

OBSERVATION: C++ forces join/detach to be called. In Java or C# for example, threads are detached by default.

What to discuss next?
Discuss the class implementation of IET Thread for OOP thread handling.

What to discuss next?
Discuss hands-on activity on using threads with SFML. GDPARCM_HO2 and GDPARCM_HO3.

Prerequisites: The students should have SFML setup and C++ working.

GDPARCM_HO2 = This is an application that is single-threaded. The assets are loaded into the screen with a fixed time delay. Since there is no background thread, this causes the frame rate to drop.

SOLUTION: Use a background thread.

GDPARCM_HO3 = While using background threads to speed up loading. This application modifies the threading part, such that a batch of assets will be loaded all at once, in the hopes of making it faster. 1 asset = 1 thread. Notice that spawning a huge number of threads all at once will cause the frame rate to drop again.

SOLUTION: Limit number of threads by enforcing thread pooling.

Activity
Assumption: All threads are detached by default.

- Create N independent threads that periodically print different print statements. All threads are persistent until the main thread has stopped.

```
void main() {
        int numThreads = 100;
        MyThread[] threadList = new MyThread[numThreads];

        for(int i = 0; i < numThreads; i++) {
                MyThread thread = new MyThread(i);
                threadList[i] = thread
                threadList[i].start();
        }

        sleep(10000);P
        for(int i = 0; i < numThreads; i++) {
                threadList[i].dismiss();
        }
}

class MyThread {
        int id;
        bool running = false;

        public MyThread(int id) {
                this.id = id
        }P

        public void dismiss() {
                this.running = false;
        }

        void run() {
                this.running = true;

                while(this.running) {
                        print("Hello I am thread #: ", this.id);

                        sleep(1000);
                }
        }
}
```

- Create an application that sorts N arrays independently. Once finished, print each array separately.

```
void main() {
        int batchSize = 100;
        int[][] batchArrays = <assume each array is instantiated with random size>
        MyThread[] threadList = new MyThread[batchSize];

        for(int i = 0; i < batchSize; i++) {
                MyThread thread = new MyThread(i);
                threadList[i] = thread
                threadList[i].start();
        }

        <do not use sleep for waiting!>

        bool standby = true;

        while(standby) {
                for(int i = 0; i < batchSize; i++) {
                        standby = false;
                        if(threadList[i].isRunning()) {
                                standby = true;
                                break; //if > 1 thread is running, set standby = true
                        }
                }

        }
}

class MyThread {
        int id;
        int[] array;
        bool running = false;
        public MyThread(int id, int[] toSort) {
                this.id = id
                this.array = toSort;
        }

        void run() {
                this.running = true;
                sort(this.array);
                this.running = false;
        }

        public bool isRunning() {
                return this.running;
        }
}
```

- Create an application that sorts a single array. Partition the array as evenly as possible, given **M** threads. Once finished, print the array.

```
void main() {
        int arrayToSort = <some array to sort>
        int numThreads = <some number of threads>

        int division = floor(arrayToSort.length / numThreads);

        int lowerB = 0;
        int upperB = division;
        for(int i = 0; i < numThreads; i++) {
                MyThread thread = new MyThread(i, lowerB, upperB);
                threadList[i] = thread
                threadList[i].start();

                lowerB = upperB + 1;
                upperB = upperB + division
        }

        <do not use sleep for waiting!>

        bool standby = true;

        while(standby) {
                for(int i = 0; i < numThreads; i++) {
                        standby = false;
                        if(threadList[i].isRunning()) {
                                standby = true;
                                break; //if > 1 thread is running, set standby = true
                        }
                }

        }
}

class MyThread {
        int id;
        int[] array;
        bool running = false;

        int lowerB, upperB;

        public MyThread(int id, int[] toSort, int lowerB, int upperB) {
                this.id = id
                this.array = toSort;
                this.lowerB, upperB = lowerB, upperB;
        }

        void run() {
                this.running = true;
                sort(this.arraym, lowerB, upperB);
                this.running = false;
        }

        public bool isRunning() {
                return this.running;
        }}
```

- Create an application that checks if a given range of numbers is prime. The application should use N-independent threads to check different portions of the range.

```
class PrimeCheckerThread extends Thread {
   private int start;
   private int end;
   private boolean isPrime[];
```

```
    public PrimeCheckerThread(int start, int end, boolean[] isPrime) {
        this.start = start;
        this.end = end;
        this.isPrime = isPrime;
    }

    public void run() {
        for (int i = start; i <= end; i++) {
            if (isPrime[i]) { // check only if not already marked as non-prime
                for (int j = i * i; j <= end; j += i) {
                    isPrime[j] = false; // mark multiples as non-prime
                }
            }
        }
    }
}

public class PrimeNumberChecker {
    public static void main(String[] args) {
        int n = 100000; // range of numbers to check
        int numThreads = 4; // number of threads

        boolean[] isPrime = new boolean[n + 1];
        Arrays.fill(isPrime, true); // initially assume all numbers are prime

        PrimeCheckerThread[] threads = new PrimeCheckerThread[numThreads];
        int rangePerThread = n / numThreads;

        for (int i = 0; i < numThreads; i++) {
            int start = i * rangePerThread + 1;
            int end = (i + 1) * rangePerThread;
            if (i == numThreads - 1) {
                end = n;
            }
            threads[i] = new PrimeCheckerThread(start, end, isPrime);
            threads[i].start();
        }

        for (Thread thread : threads) {
            try {
                thread.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        // Print prime numbers
        for (int i = 2; i <= n; i++) {
            if (isPrime[i]) {
                System.out.print(i + " ");
            }
        }
    }
}
```

- Create an application that searches for a specific character subset in a very long string. The application should use N independent threads to search different areas of the string.

```
  class SubsetSearchThread extends Thread {
      private String string;
      private String subset;
      private int startIndex;
      private int endIndex;
      private boolean found;

      public SubsetSearchThread(String string, String subset, int startIndex, int endIndex) {
          this.string = string;
          this.subset = subset;
          this.startIndex = startIndex;
          this.endIndex = endIndex;
          this.found = false;
```

```
    }

    public void run() {
        for (int i = startIndex; i <= endIndex && !found; i++) {
            if (string.substring(i, Math.min(i + subset.length(),
string.length())).equals(subset)) {
                found = true;
            }
        }
    }

    public boolean isFound() {
        return found;
    }
}

public class SubsetSearcher {
    public static void main(String[] args) {
        String longString = "This is a very long string with a hidden subset to find.";
        String subset = "hidden";
        int numThreads = 4;

        int stringLength = longString.length();
        int rangePerThread = stringLength / numThreads;

        SubsetSearchThread[] threads = new SubsetSearchThread[numThreads];

        for (int i = 0; i < numThreads; i++) {
            int start = i * rangePerThread;
            int end = Math.min((i + 1) * rangePerThread, stringLength - 1);
            threads[i] = new SubsetSearchThread(longString, subset, start, end);
            threads[i].start();
        }

        for (Thread thread : threads) {
            try {
                thread.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        boolean found = false;
        for (SubsetSearchThread thread : threads) {
            if (thread.isFound()) {
                found = true;
                break;
            }
        }

        if (found) {
            System.out.println("Character subset found in the string.");
        } else {
            System.out.println("Character subset not found in the string.");
        }
    }
}
```