

**wait() and notify()**

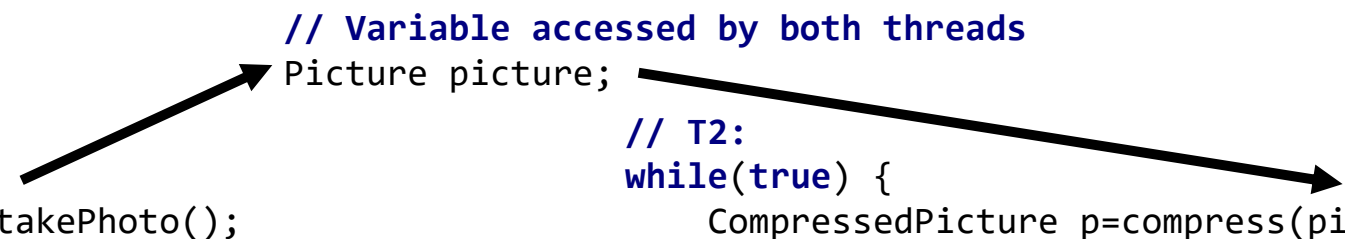
# Two dependent threads

- Sometimes we have the situation that a thread has to wait until another thread has done a specific action. Example:
  - Thread 1 (T1) is a camera that continuously takes pictures
  - Thread 2 (T2) runs an algorithm to compress pictures
- Wrong way to implement that:

```
// Variable accessed by both threads
Picture picture;

// T1:
while(true) {
    picture = takePhoto();
}

// T2:
while(true) {
    CompressedPicture p=compress(picture);
    writePictureToFile(p);
}
```



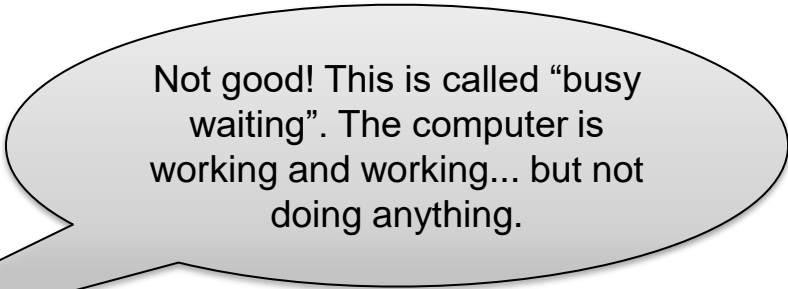
- This doesn't work if T1 and T2 work at different speeds. We want:
  - If there is no new picture, T2 should wait.
  - T1 should not make pictures faster than T2 can compress them

# Solution with while-loops

- Our idea: When T2 starts compressing a picture it sets the picture variable to null to indicate to T1 that the next photo can be taken. If there is no picture, T2 waits.
- Here is a bad implementation:

```
// T1:  
while(true) {  
    Picture currentPicture =takePhoto();  
    while(picture!=null) { } // wait for T1  
    picture = currentPicture; // give the picture to T2  
}
```

```
// T2:  
while(true) {  
    while(picture==null) { } // wait for T2  
    Picture currentPicture = picture; // get picture from T2  
    picture = null; // tell T1 to continue  
  
    CompressedPicture p=compress(currentPicture);  
    p.writeToFile();  
}
```



Not good! This is called “busy waiting”. The computer is working and working... but not doing anything.

# Correct code

```
// Code for T1
while(true) {
    Picture currentPicture = takePhoto();
    synchronized(someObject) {
        while(picture!=null) {
            try {
                someObject.wait();
            }
            catch(InterruptedException e) { throw new RuntimeException("...", e); }
        }
        picture = currentPicture;
        someObject.notify();
    }
}
```

All access to the picture variable is inside the synchronized statement

If picture!=null, task T1 sleeps until it gets a notification from T2

Here, we notify T2 that a new picture is ready

```
// Code for T2:
while(true) {
    Picture currentPicture;
    synchronized(someObject) {
        while(picture==null) {
            try {
                someObject.wait();
            }
            catch(InterruptedException e) { throw new RuntimeException("...", e); }
        }
        currentPicture=picture;
        picture=null;
        someObject.notify();
    }
}
```

If picture==null, task T2 waits until it gets a notification from T1

Here, we notify T1 that it can give T2 a new picture

```
CompressedPicture p=compress(currentPicture);
p.writeToFile();
}
```

# wait() and notify()

- The wait() method puts the calling thread to sleep until another thread calls notify() on the same object (or if an InterruptedException happens)
- When waking up from a wait(), you should always check whether the condition you were waiting for is fulfilled. It could be that the thread has been waken up by accident. In our example, we have put wait() inside a while-loop to be sure that picture!=null in T1:

```
while (picture != null) {  
    try {  
        someObject.wait();  
    }  
    catch (InterruptedException) { throw new RuntimeException("...", e); }  
}
```

# Multiple threads waiting

- It's possible that multiple threads wait at the same time for the same object
  - In that case, `notify()` will randomly choose one waiting thread
  - Alternatively, you can wake up all waiting threads with `notifyAll()`
- Example:
  - T1 has two cameras and produces two pictures at the same time
  - Two threads T2 and T3 wait to compress pictures
  - T1 can use `notifyAll()` to wake up both threads T2 and T3 at the same time

# Synchronization and wait() and notify()

- A thread can only call wait() or notify() on an object if it owns the monitor of the object, i.e. wait() and notify() must be always inside a synchronized block

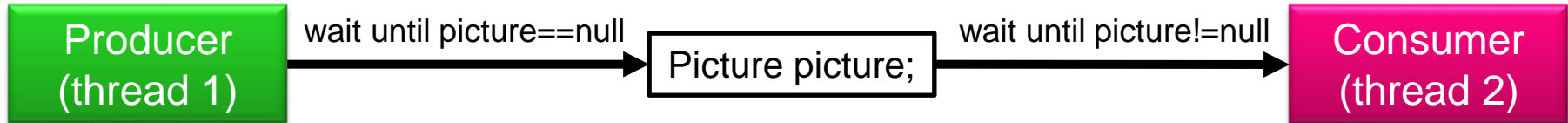
```
// T1:  
synchronized(someObject) {  
    someObject.wait();  
    i++;  
}
```

```
// T2:  
synchronized(someObject) {  
    someObject.notify();  
}
```

- When a thread calls wait(), the thread releases the monitor
- When a waiting thread is waken up, it waits until the thread that called notify() releases the monitor
- Example: Let's assume T1 first gets the monitor of someObject
  1. T1 calls wait() and releases the monitor of someObject
  2. T2 can enter the monitor and calls notify()
  3. T1 has to wait until T2 leaves the monitor
  4. When T2 has left, T1 can execute i++

# Producer-Consumer

- Our picture example is called a Producer-Consumer problem



- Sometimes, you will have multiple producers and consumers. In that case, having a variable for only one picture does not make sense. Instead, a buffer (a list, an array, a queue,...) is used. Often, you want to limit the size of the buffer, so it doesn't become too large.

