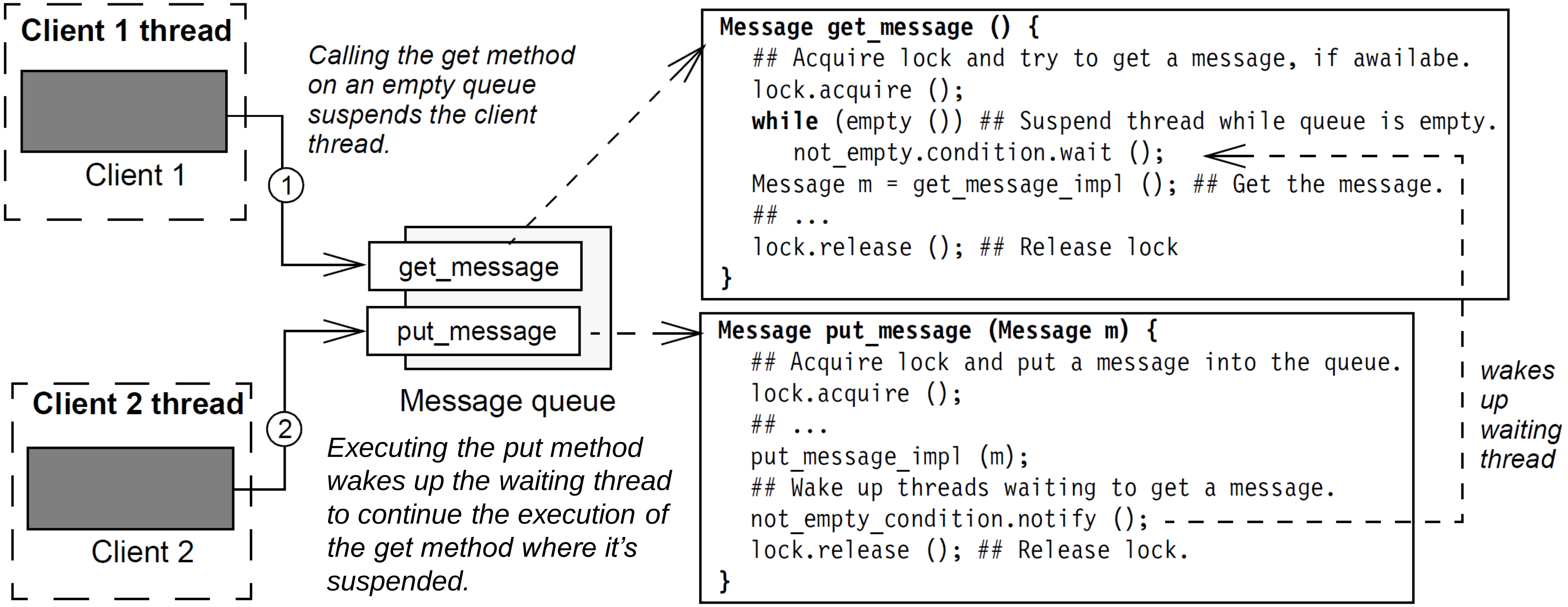
The Guarded Suspension Pattern

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand what condition variables are
- Note a human known use of condition variables
- Know what pattern condition variables implement



Calling the get method on an empty queue suspends the client thread.

Executing the put method wakes up the waiting thread to continue the execution of the get method where it's suspended.

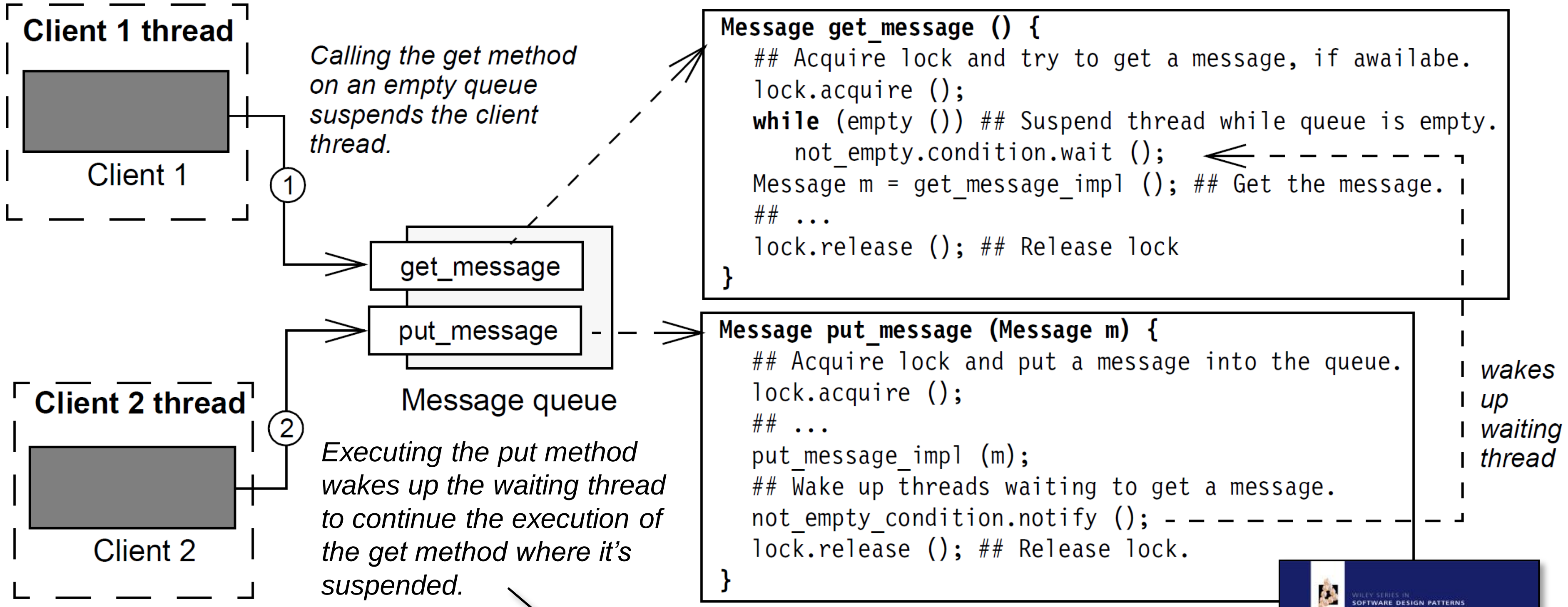
```
Message get_message () {
  ## Acquire lock and try to get a message, if awailabe.
  lock.acquire ();
  while (empty ()) ## Suspend thread while queue is empty.
     not_empty.condition.wait ();
  Message m = get_message_impl (); ## Get the message.
  ## ...
  lock.release (); ## Release lock
}
```

```
Message put_message (Message m) {
  ## Acquire lock and put a message into the queue.
  lock.acquire ();
  ## ...
  put_message_impl (m);
  ## Wake up threads waiting to get a message.
  not_empty_condition.notify ();
  lock.release (); ## Release lock.
}
```

Implementing Guarded Suspension with CVs

Implementing Guarded Suspension with CVs

- CVs are most often used to implement the *Guarded Suspension* pattern

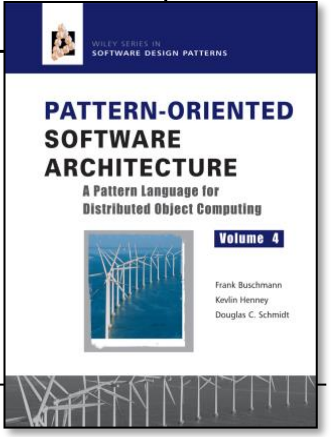


Client 1 thread
Client 1
1
Calling the get method on an empty queue suspends the client thread.
get_message
put_message
Message queue
Client 2 thread
Client 2
2
Executing the put method wakes up the waiting thread to continue the execution of the get method where it's suspended.
wakes up waiting thread

```
Message get_message () {
  ## Acquire lock and try to get a message, if awailabe.
  lock.acquire ();
  while (empty ()) ## Suspend thread while queue is empty.
     not_empty.condition.wait ();
  Message m = get_message_impl (); ## Get the message.
  ## ...
  lock.release (); ## Release lock
}
```

```
Message put_message (Message m) {
  ## Acquire lock and put a message into the queue.
  lock.acquire ();
  ## ...
  put_message_impl (m);
  ## Wake up threads waiting to get a message.
  not_empty_condition.notify ();
  lock.release (); ## Release lock.
}
```

Require both a lock to be acquired & a precondition to be satisfied before an operation can be executed


WILEY SERIES IN SOFTWARE DESIGN PATTERNS

PATTERN-ORIENTED SOFTWARE ARCHITECTURE

A Pattern Language for Distributed Object Computing

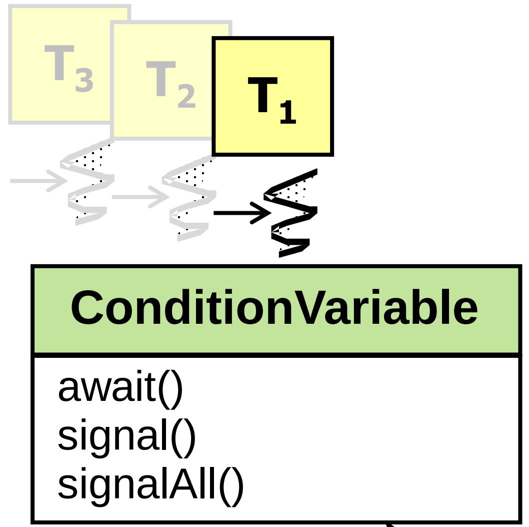
Volume 4

Frank Buschmann
Kevin Henney
Douglas C. Schmidt

See en.wikipedia.org/wiki/Guarded_suspension

Implementing Guarded Suspension with CVs

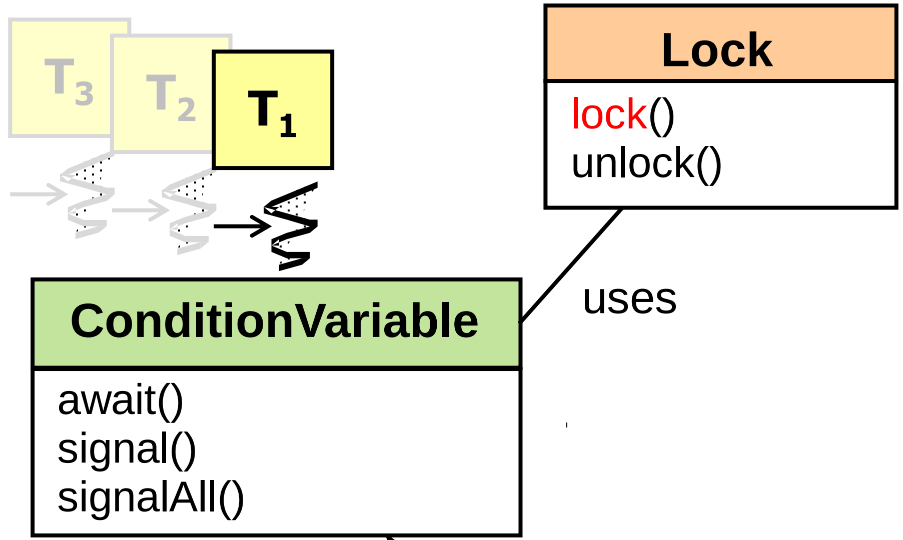
- This pattern is applied to operations that can run only when a condition is satisfied



```
Lock l = new Lock()
Condition cond =
  l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
  cond.await()
doOperationProcessing()
```

Implementing Guarded Suspension with CVs

- This pattern is applied to operations that can run only when a condition is satisfied, e.g.,
  - a lock is acquired

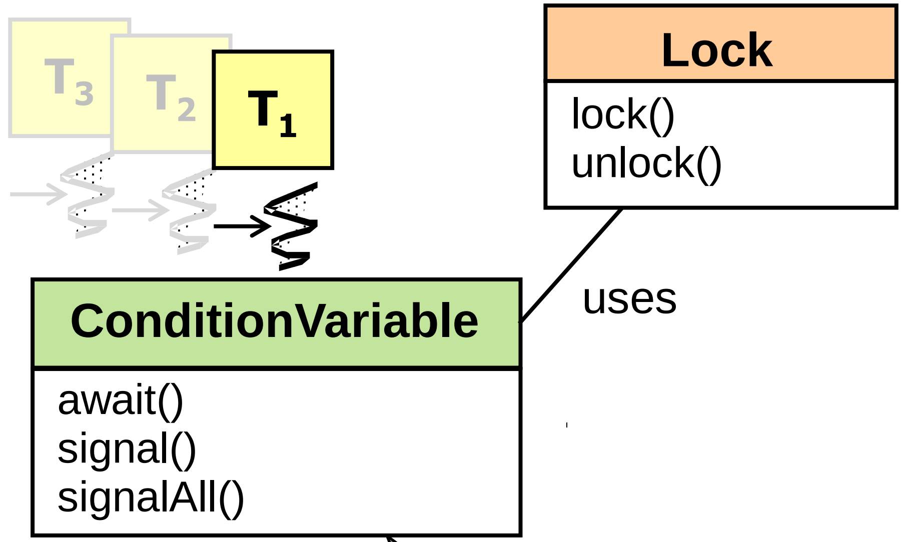


```
Lock l = new Lock()
Condition cond =
  l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
  cond.await()
doOperationProcessing()
```

A condition variable is *always* associated with a lock

Implementing Guarded Suspension with CVs

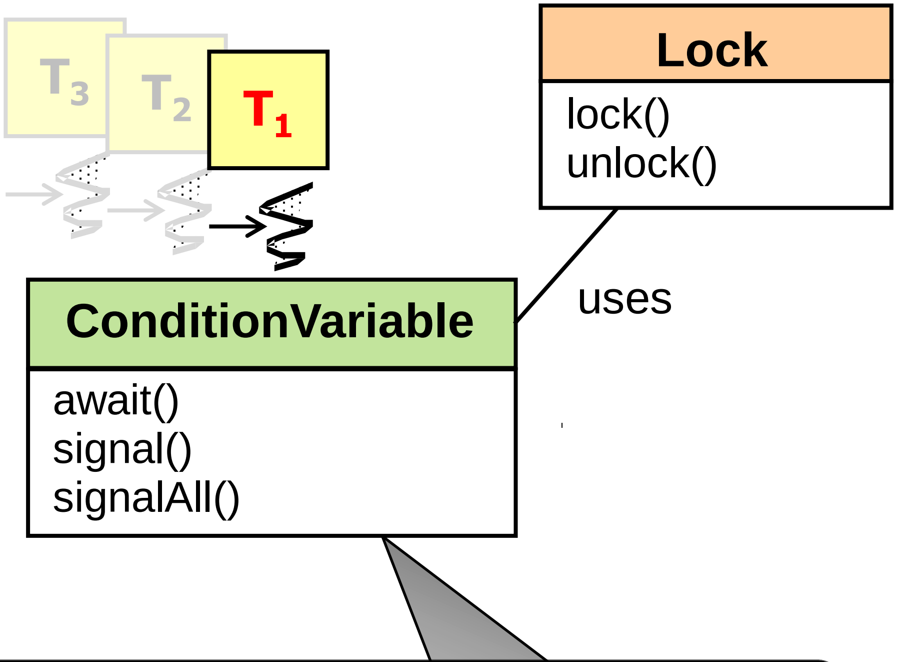
- This pattern is applied to operations that can run only when a condition is satisfied, e.g.,
  - a lock is acquired
  - a precondition holds



```
Lock l = new Lock()
Condition cond =
  l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
  cond.await()
doOperationProcessing()
```

Implementing Guarded Suspension with CVs

- In this example thread T_1 uses a CV to suspend its execution until thread T_n notifies it that shared state it's waiting on *may* now be satisfied

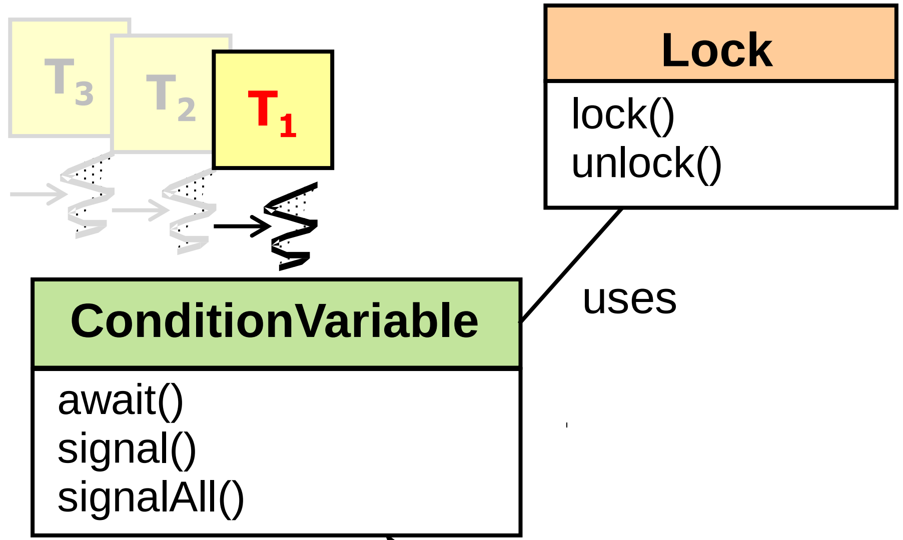


```
Lock l = new Lock()
Condition cond =
  l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
  cond.await()
doOperationProcessing()
```

See www.youtube.com/watch?v=mJZZNHekEQw

Implementing Guarded Suspension with CVs

- In this example thread T_1 uses a CV to suspend its execution until thread T_n notifies it that shared state it's waiting on *may* now be satisfied

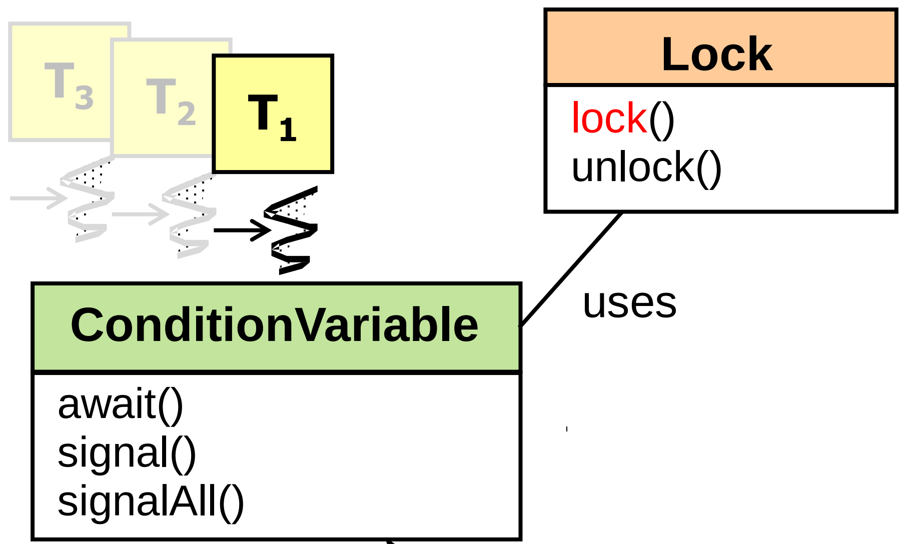


```
Lock l = new Lock()
Condition cond =
  l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
  cond.await()
doOperationProcessing()
```

Note the tentative nature of "may"..

Implementing Guarded Suspension with CVs

- In this example thread T_1 uses a CV to suspend its execution until thread T_n notifies it that shared state it's waiting on *may* now be satisfied

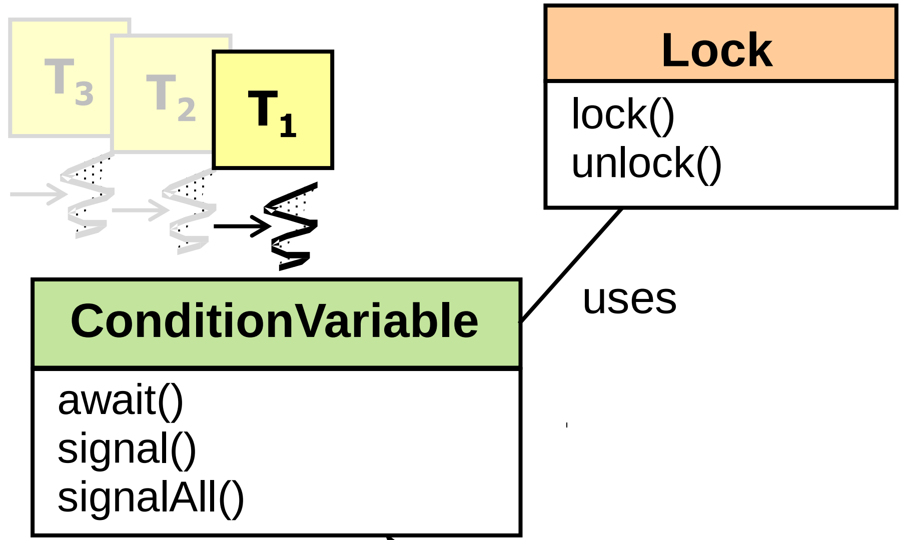


First, a lock must be acquired..

```
Lock l = new Lock()
Condition cond =
  l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
  cond.await()
doOperationProcessing()
```

Implementing Guarded Suspension with CVs

- In this example thread T_1 uses a CV to suspend its execution until thread T_n notifies it that shared state it's waiting on *may* now be satisfied

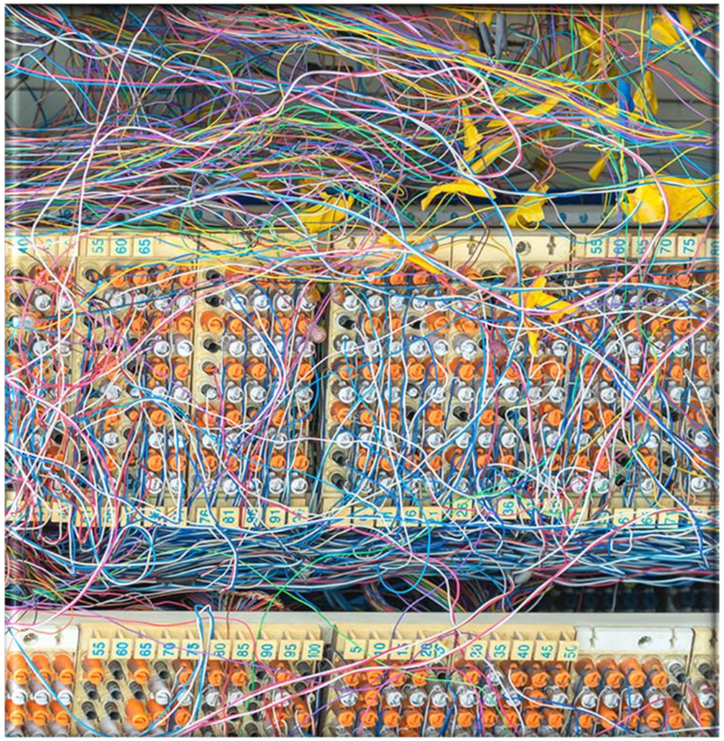


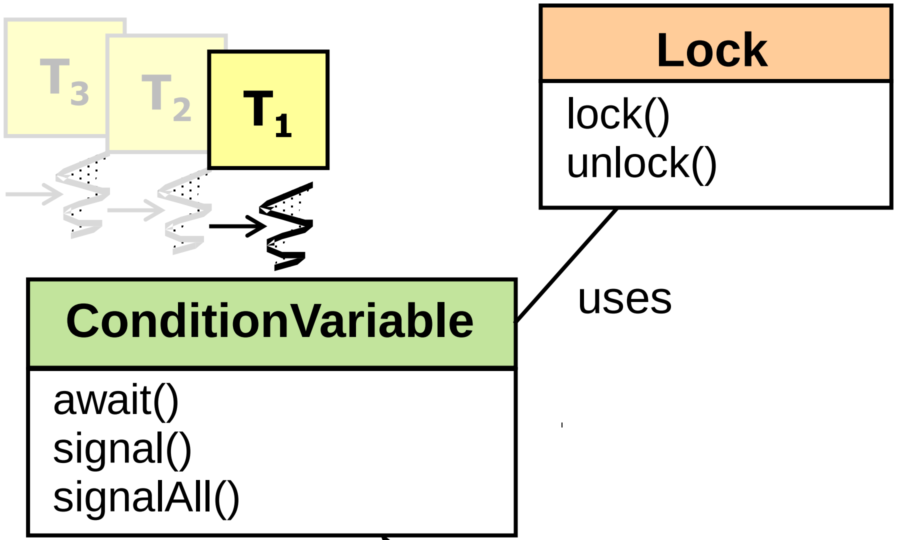
Second, a condition is checked (in a loop) with the lock held..

```
Lock l = new Lock()
Condition cond =
  l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
  cond.await()
doOperationProcessing()
```

Implementing Guarded Suspension with CVs

- In this example thread T_1 uses a CV to suspend its execution until thread T_n notifies it that shared state it's waiting on *may* now be satisfied
  - A condition can be arbitrarily complex

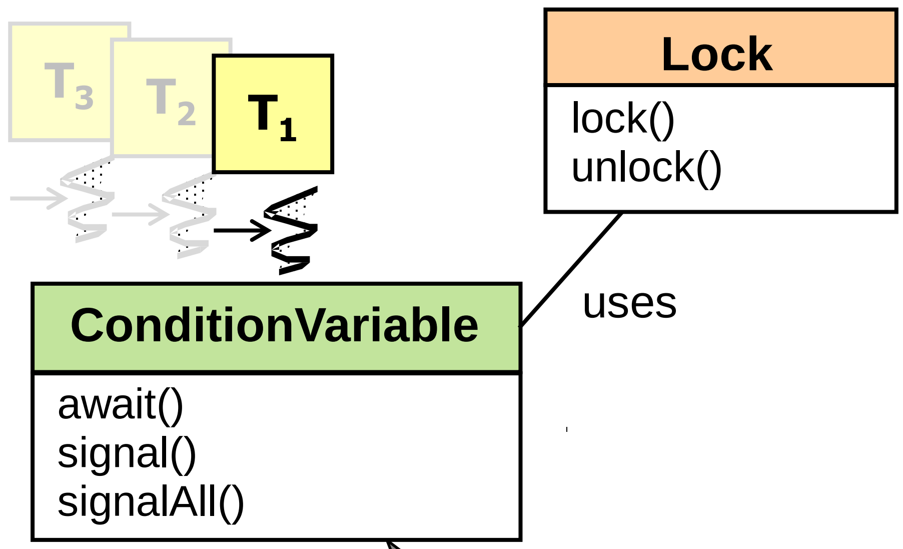




```
Lock l = new Lock()
Condition cond =
  l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
  cond.await()
doOperationProcessing()
```

Implementing Guarded Suspension with CVs

- In this example thread T_1 uses a CV to suspend its execution until thread T_n notifies it that shared state it's waiting on *may* now be satisfied
  - A condition can be arbitrarily complex



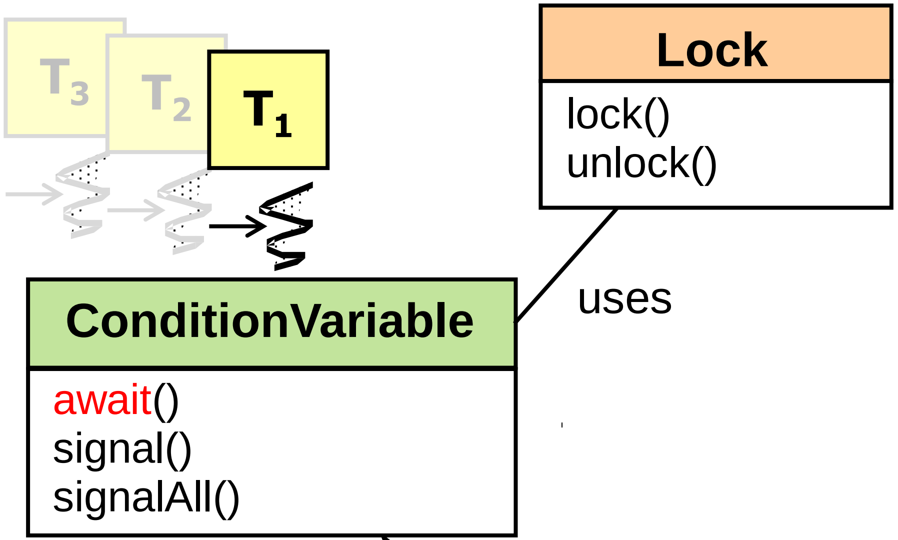
e.g., a method call, an expression that involves shared state, etc.

```
Lock l = new Lock()
Condition cond =
  l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
  cond.await()
doOperationProcessing()
```

Any state shared between threads must be protected by a lock associated with the CV

Implementing Guarded Suspension with CVs

- In this example thread T_1 uses a CV to suspend its execution until thread T_n notifies it that shared state it's waiting on *may* now be satisfied
  - A condition can be arbitrarily complex

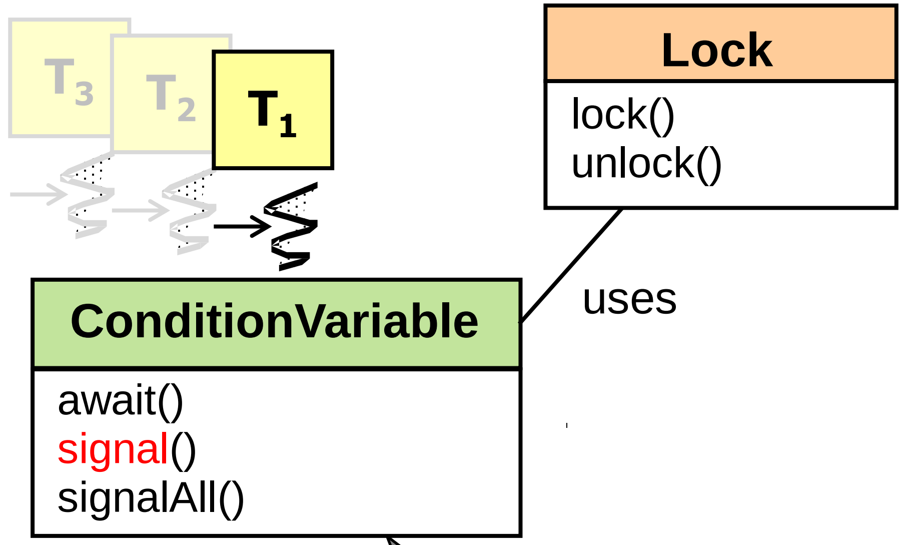


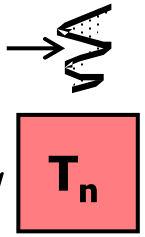
The calling thread will block (possibly repeatedly) while the condition is not satisfied (await() atomically releases the lock)

```
Lock l = new Lock()
Condition cond =
  l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
  cond.await()
doOperationProcessing()
```

Implementing Guarded Suspension with CVs

- In this example thread T_1 uses a CV to suspend its execution until thread T_n notifies it that shared state it's waiting on *may* now be satisfied
  - A condition can be arbitrarily complex





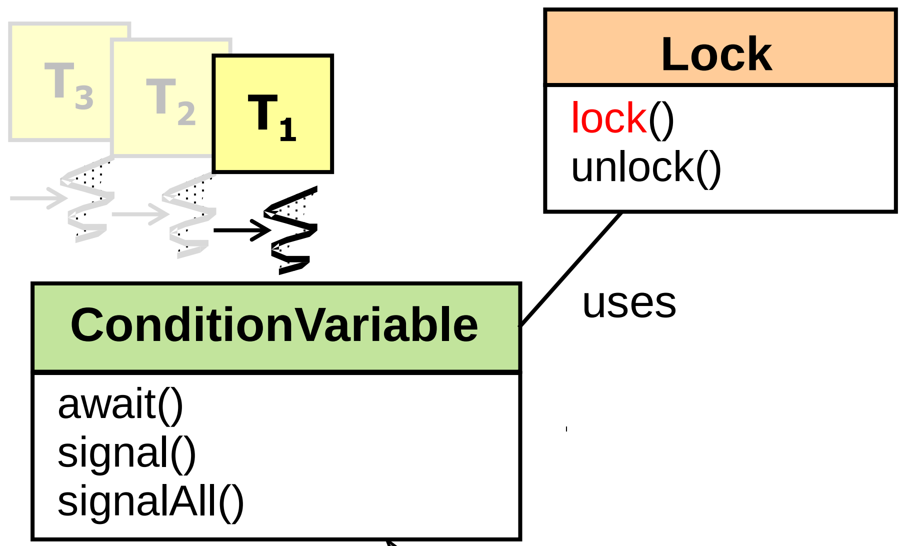
```
cond.signal()
```

Another thread can signal condition when shared state may now be true

```
Lock l = new Lock()
Condition cond =
  l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
  cond.await()
doOperationProcessing()
```

Implementing Guarded Suspension with CVs

- In this example thread T_1 uses a CV to suspend its execution until thread T_n notifies it that shared state it's waiting on *may* now be satisfied
  - A condition can be arbitrarily complex

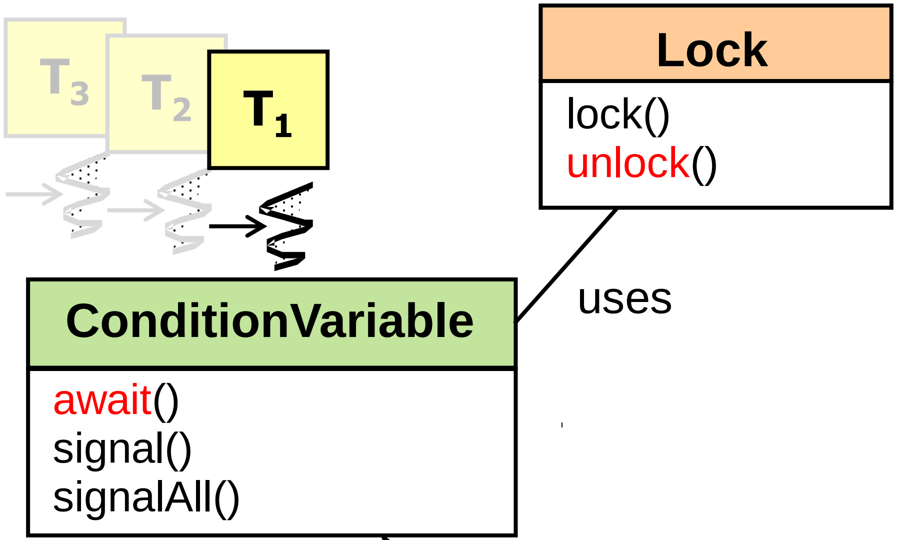


await() reacquires the lock & condition is rechecked in loop

```
Lock l = new Lock()
Condition cond =
  l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
  cond.await()
doOperationProcessing()
```

Implementing Guarded Suspension with CVs

- In this example thread T_1 uses a CV to suspend its execution until thread T_n notifies it that shared state it's waiting on *may* now be satisfied
  - A condition can be arbitrarily complex
  - Waiting on a CV releases the lock & suspends the thread *atomically*

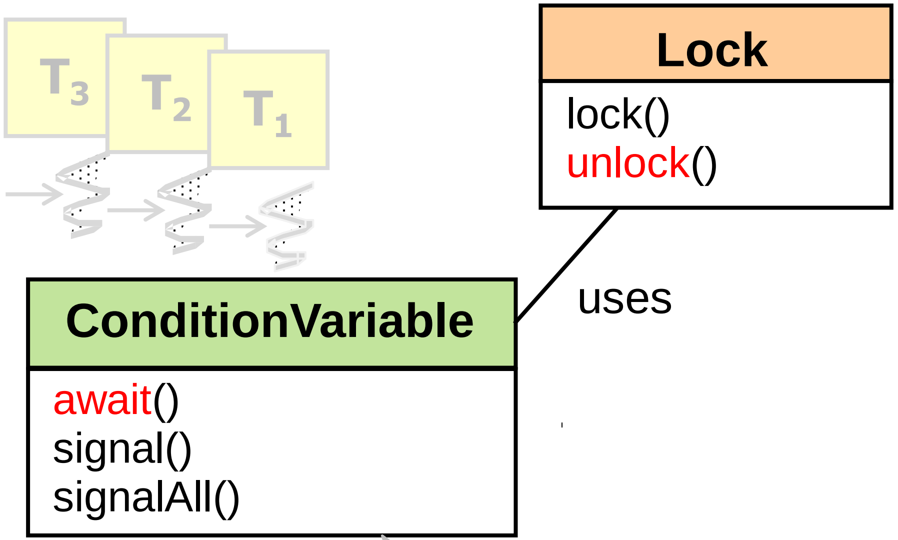


```
Lock l = new Lock()
Condition cond =
  l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
  cond.await()
doOperationProcessing()
```

The lock is released when the thread is suspended on the CV

Implementing Guarded Suspension with CVs

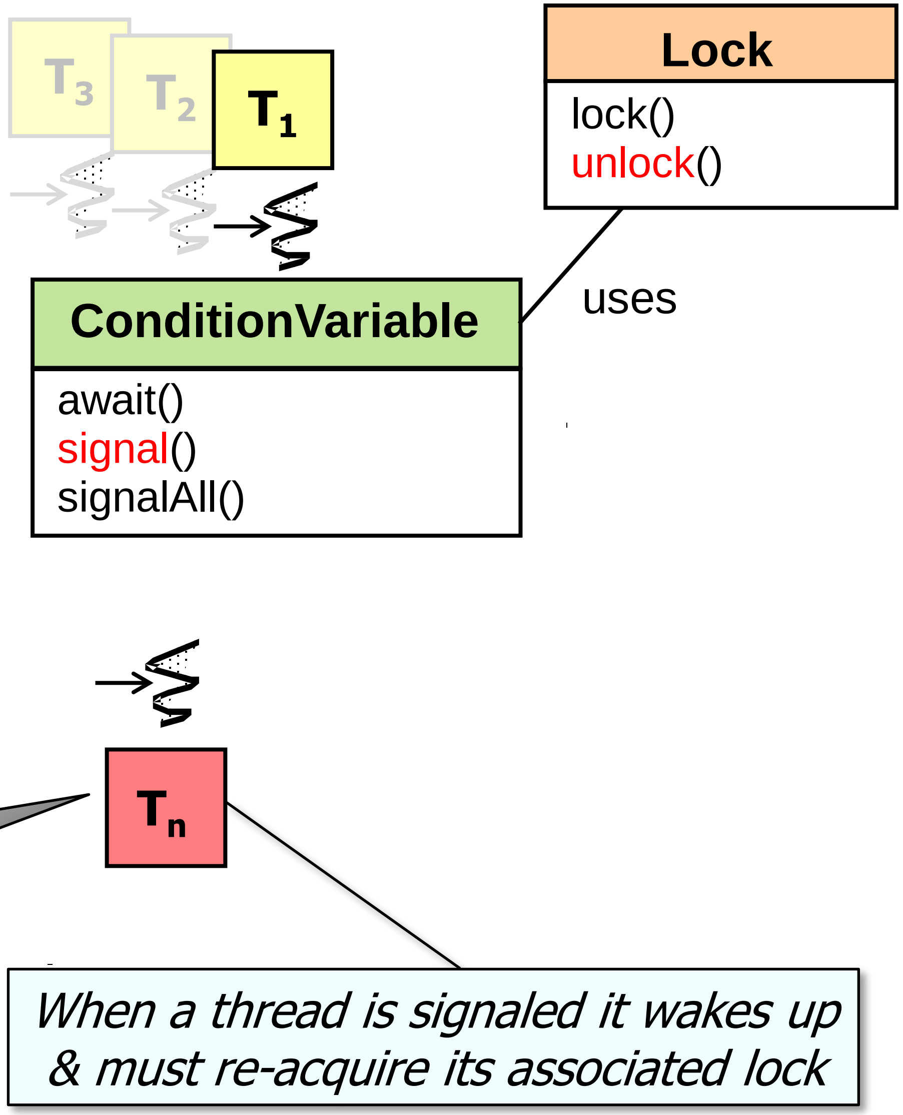
- In this example thread T_1 uses a CV to suspend its execution until thread T_n notifies it that shared state it's waiting on *may* now be satisfied
  - A condition can be arbitrarily complex
  - Waiting on a CV releases the lock & suspends the thread *atomically*
    - Thread T_1 is suspended until thread T_n signals the CV



```
Lock l = new Lock()
Condition cond =
  l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
  cond.await()
doOperationProcessing()
```

Implementing Guarded Suspension with CVs

- In this example thread T_1 uses a CV to suspend its execution until thread T_n notifies it that shared state it's waiting on *may* now be satisfied
  - A condition can be arbitrarily complex
  - Waiting on a CV releases the lock & suspends the thread *atomically*
    - Thread T_1 is suspended until thread T_n signals the CV

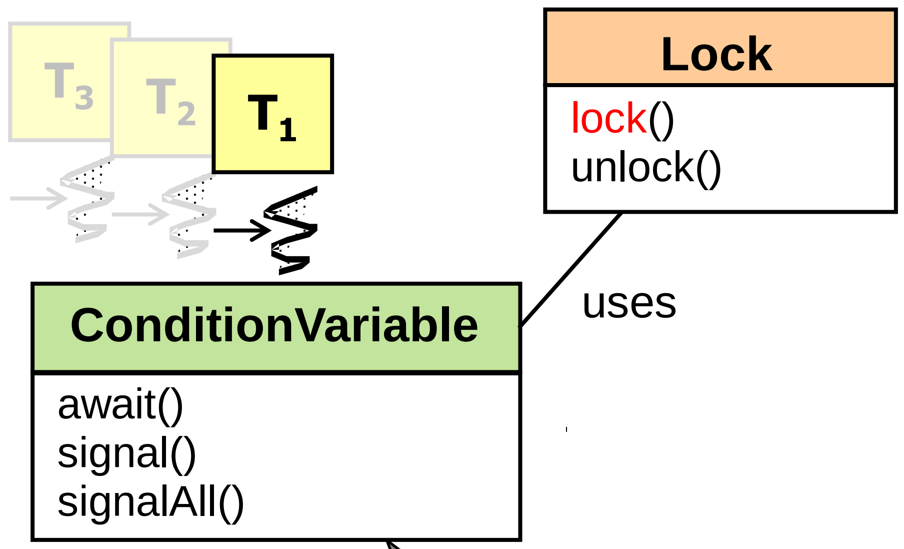


```
cond.signal()
```

When a thread is signaled it wakes up & must re-acquire its associated lock

Implementing Guarded Suspension with CVs

- In this example thread T_1 uses a CV to suspend its execution until thread T_n notifies it that shared state it's waiting on *may* now be satisfied
  - A condition can be arbitrarily complex
  - Waiting on a CV releases the lock & suspends the thread *atomically*
    - Thread T_1 is suspended until thread T_n signals the CV

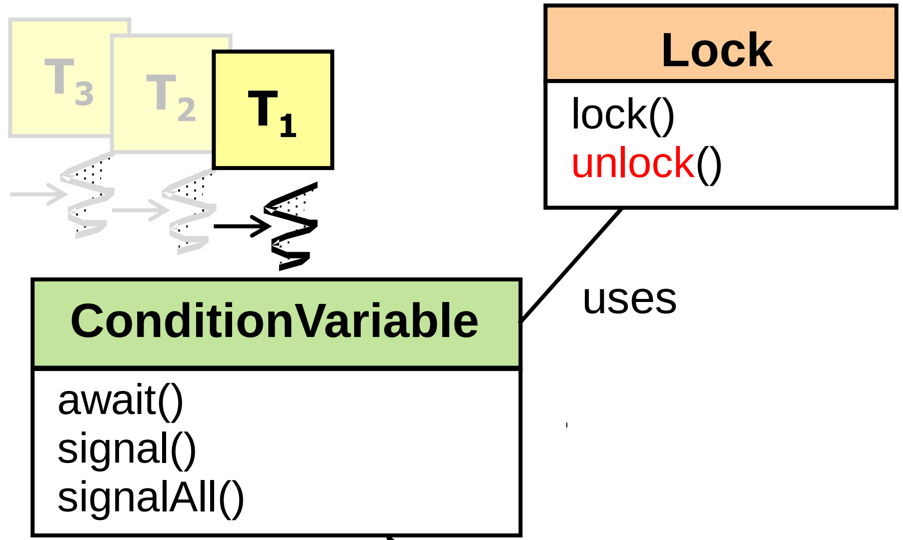


```
Lock l = new Lock()
Condition cond =
  l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
  cond.await()
doOperationProcessing()
```

After lock is re-acquired the thread can reevaluate its condition to see if it's satisfied

Implementing Guarded Suspension with CVs

- In this example thread T_1 uses a CV to suspend its execution until thread T_n notifies it that shared state it's waiting on *may* now be satisfied
  - A condition can be arbitrarily complex
  - Waiting on a CV releases the lock & suspends the thread *atomically*
    - Thread T_1 is suspended until thread T_n signals the CV

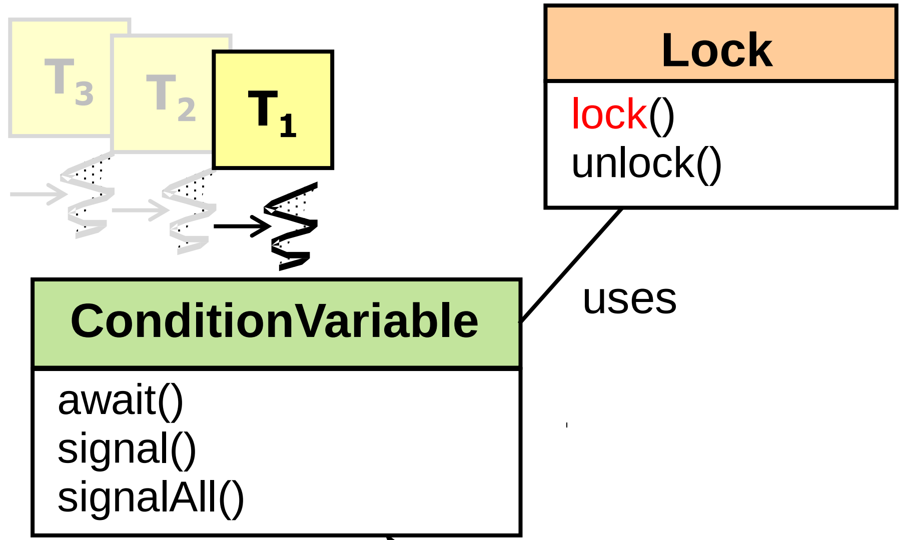


```
Lock l = new Lock()
Condition cond =
  l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
  cond.await()
doOperationProcessing()
```

If condition is not satisfied the thread must wait (which releases the lock atomically)

Implementing Guarded Suspension with CVs

- In this example thread T_1 uses a CV to suspend its execution until thread T_n notifies it that shared state it's waiting on *may* now be satisfied
  - A condition can be arbitrarily complex
  - Waiting on a CV releases the lock & suspends the thread *atomically*
    - Thread T_1 is suspended until thread T_n signals the CV



```
Lock l = new Lock()
Condition cond =
  l.newCondition()
...
l.lock()
while (conditionNotSatisfied())
  cond.await()
doOperationProcessing()
```

After the lock is re-acquired & the condition is satisfied the operation can proceed (with lock held)

End of the Guarded Suspension Pattern