

# Android Concurrency: The Monitor Object Pattern (Part 2)



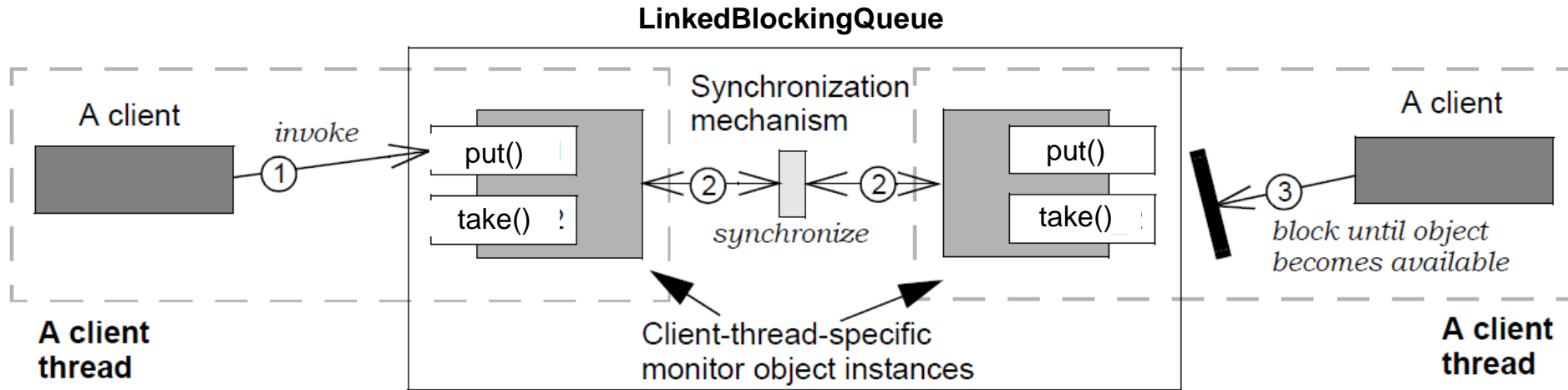
Douglas C. Schmidt  
[d.schmidt@vanderbilt.edu](mailto:d.schmidt@vanderbilt.edu)  
[www.dre.vanderbilt.edu/~schmidt](http://www.dre.vanderbilt.edu/~schmidt)

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



# Learning Objectives in this Part of the Module

- Understand how *Monitor Object* is implemented & applied in Android



# Monitor Object

# POSA2 Concurrency

## Implementation

LinkedBlockingQueue is a bounded BlockingQueue based on linked nodes that queues elements in FIFO order

Added in API level 1

### LinkedBlockingQueue

extends `AbstractQueue<E>`

implements `Serializable BlockingQueue<E>`

`java.lang.Object`

↳ `java.util.AbstractCollection<E>`

↳ `java.util.AbstractQueue<E>`

↳ `java.util.concurrent.LinkedBlockingQueue<E>`

### Class Overview

An optionally-bounded `blocking queue` based on linked nodes. This queue orders elements FIFO (first-in-first-out). The *head* of the queue is that element that has been on the queue the longest time. The *tail* of the queue is that element that has been on the queue the shortest time. New elements are inserted at the tail of the queue, and the queue retrieval operations obtain elements at the head of the queue. Linked queues typically have higher throughput than array-based queues but less predictable performance in most concurrent applications.

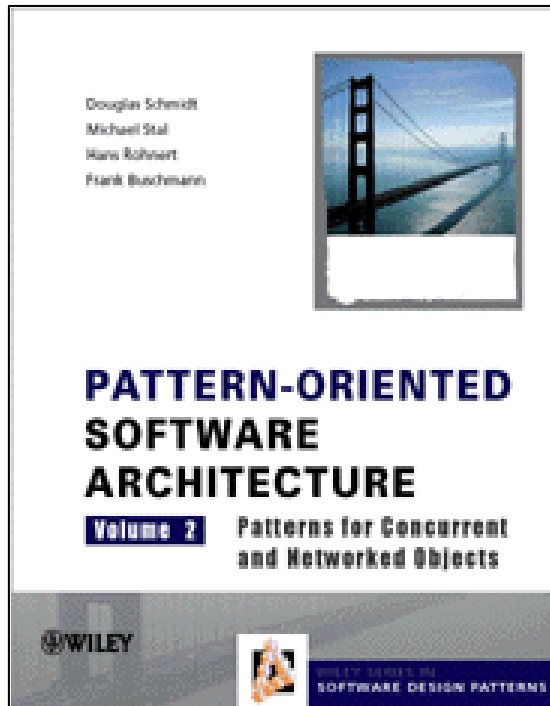
The optional capacity bound constructor argument serves as a way to prevent excessive queue expansion. The capacity, if unspecified, is equal to `MAX_VALUE`. Linked nodes are dynamically created upon each insertion unless this would bring the queue above capacity.

This class and its iterator implement all of the *optional* methods of the `Collection` and `Iterator` interfaces.

# Monitor Object

# POSA2 Concurrency

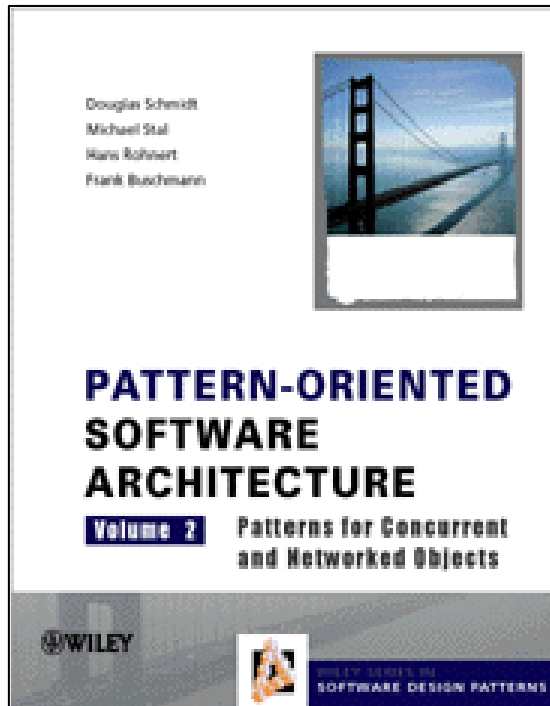
## Implementation Steps



# Monitor Object

# POSA2 Concurrency

## Implementation Steps



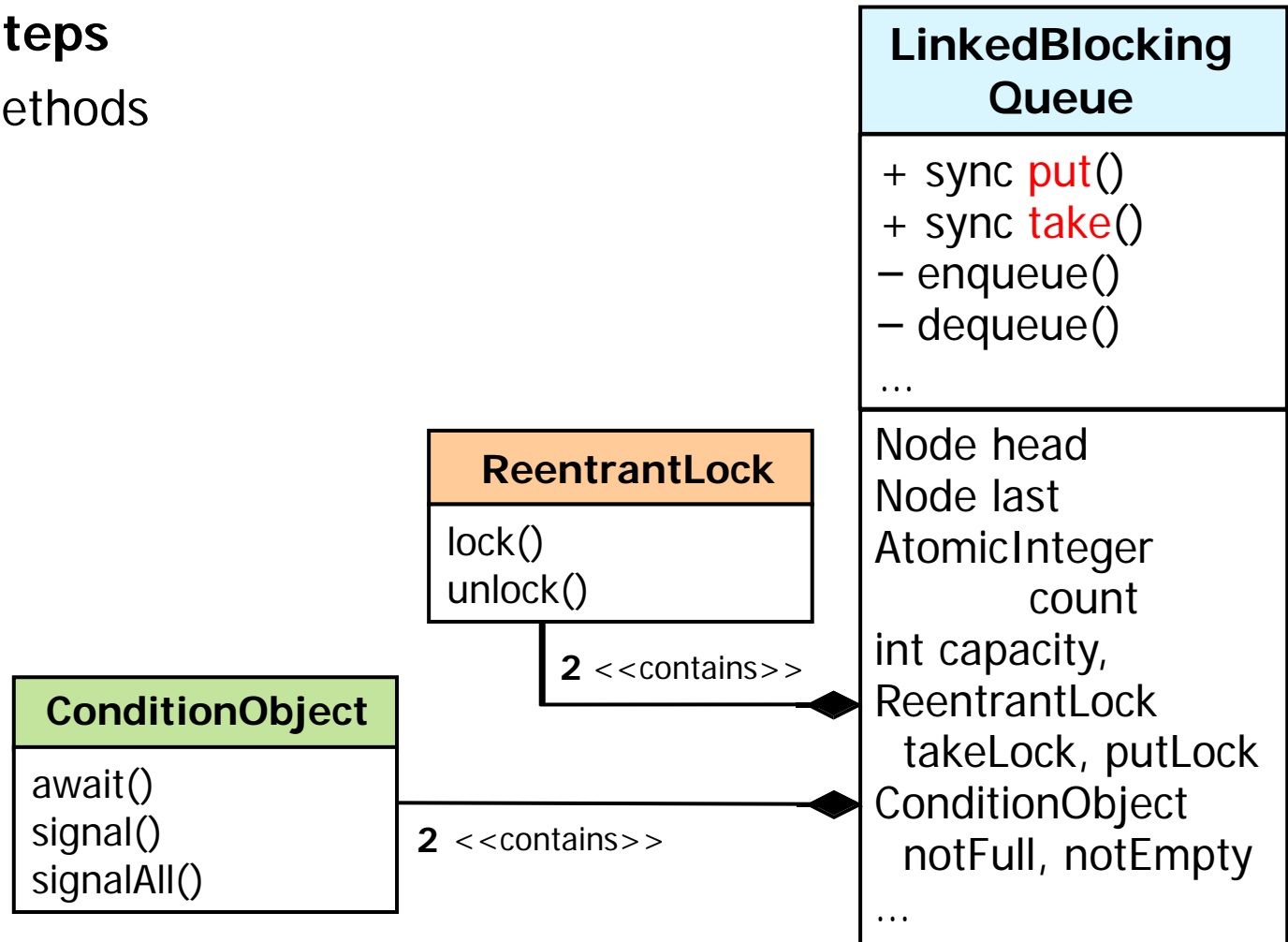
See [www.dre.vanderbilt.edu/~schmidt/PDF/monitor.pdf](http://www.dre.vanderbilt.edu/~schmidt/PDF/monitor.pdf) for *Monitor Object*

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods



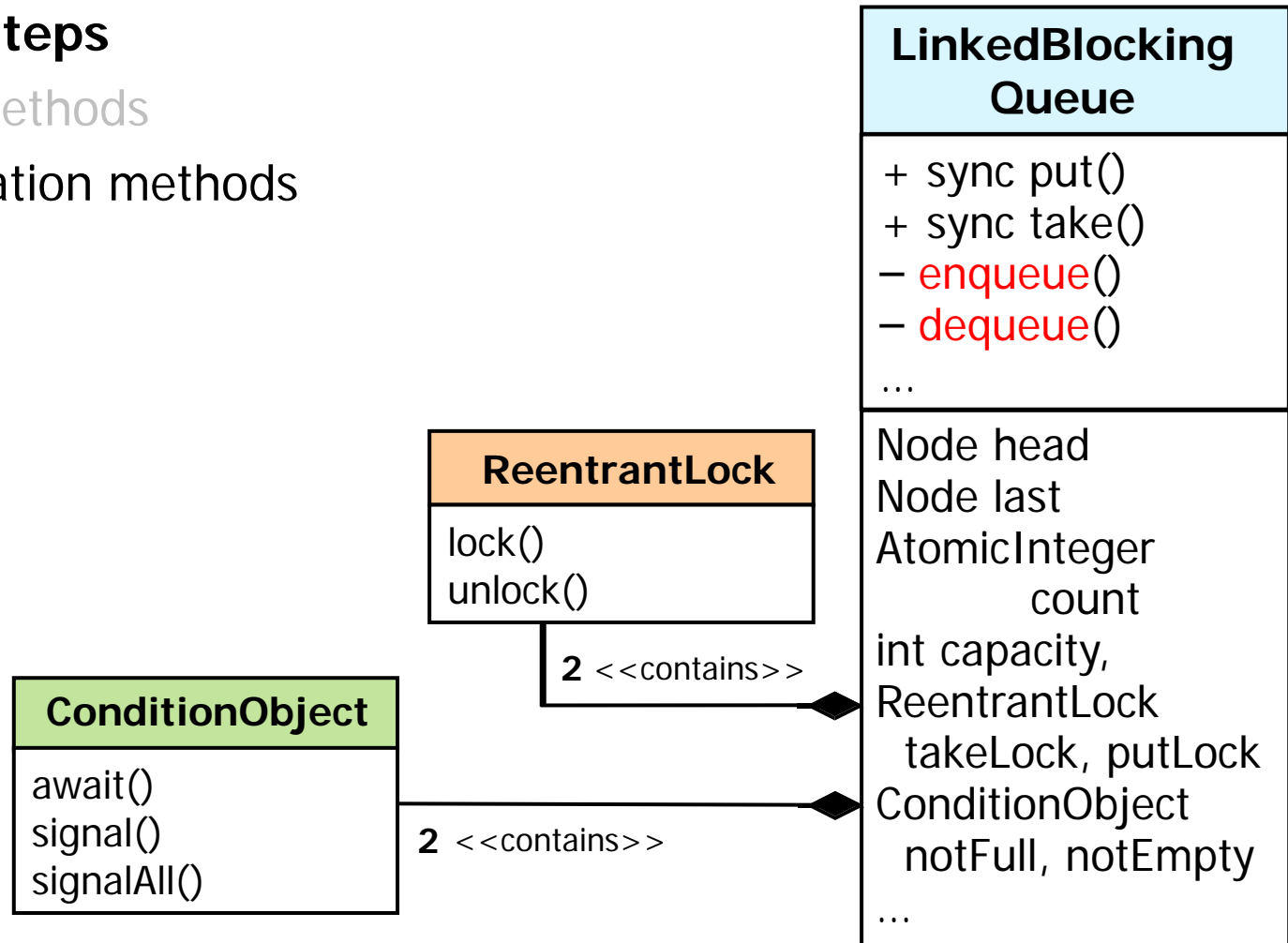


# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods

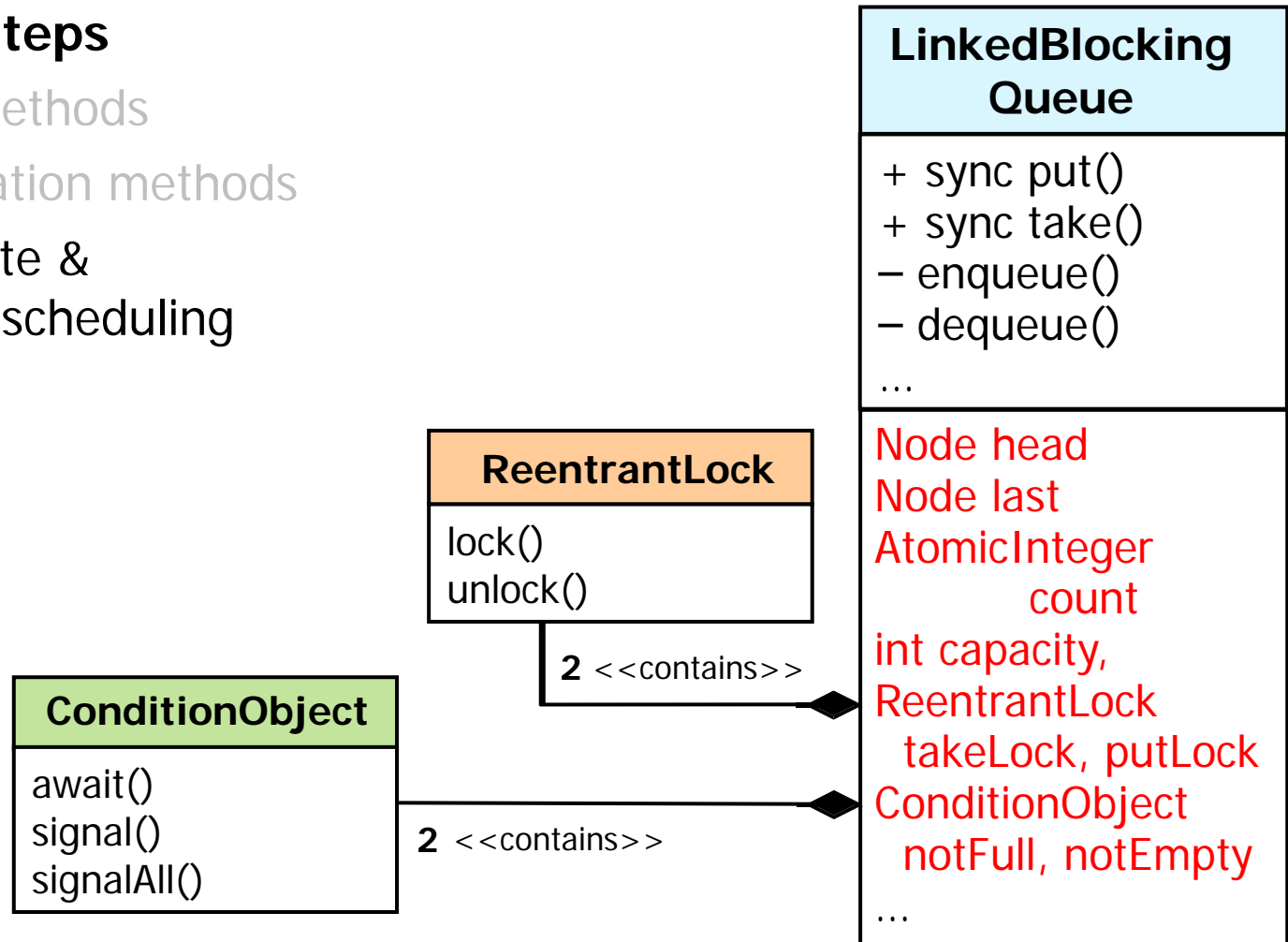


# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms



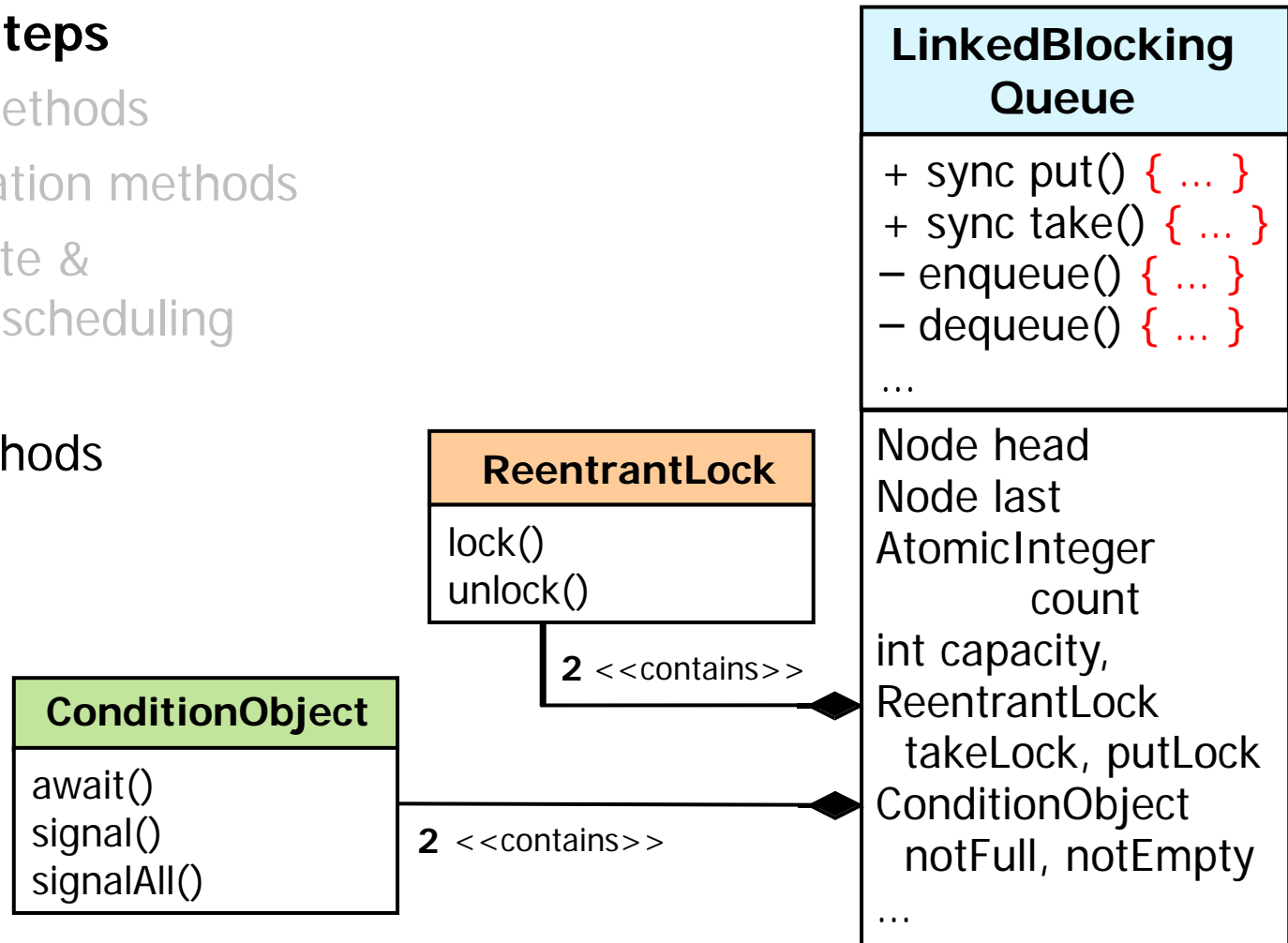


# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members



# Defining the Interface & Implementation Methods

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods

### Monitor Object

+ sync method<sub>1</sub>()

...

+ sync method<sub>n</sub>()



# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods

### Monitor Object

+ **sync** method<sub>1</sub>()

...

+ **sync** method<sub>n</sub>()



# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
  - e.g., methods inherited from the BlockingQueue interface

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {

    public void put(E e) ...

    public E take() ...

    ...
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
  - e.g., methods inherited from the BlockingQueue interface

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {

    public void put(E e) ...

    public E take() ...

    ...
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
  - e.g., methods inherited from the BlockingQueue interface

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {

    public void put(E e) ...

    public E take() ...

    ...
}
```

*Insert the specified element into a queue, waiting if necessary for space to become available*



# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
  - e.g., methods inherited from the BlockingQueue interface

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {

    public void put(E e) ...

    public E take() ...

    ...
}
```

*Retrieve & remove the head of the queue, waiting if necessary until an elements becomes available*

# Monitor Object

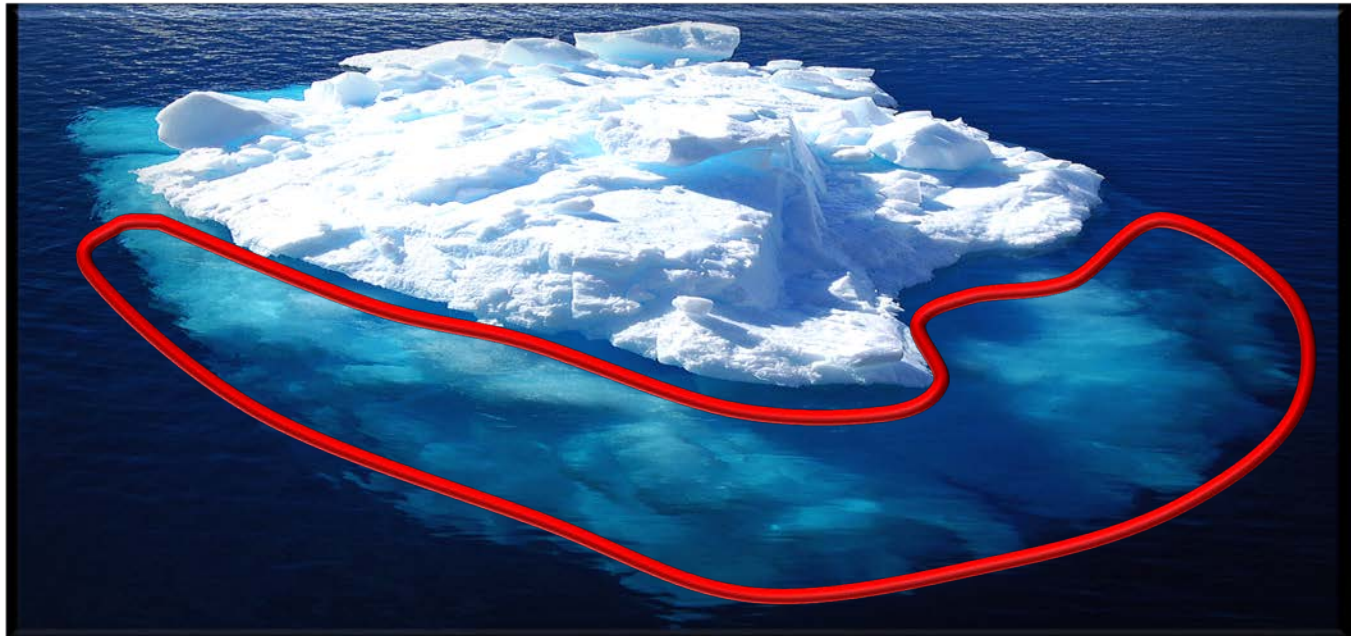
# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods

### Monitor Object

+ sync method<sub>1</sub>()  
...  
+ sync method<sub>n</sub>()  
– method<sub>1</sub>()  
...  
– method<sub>n</sub>()



# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- See the POSA2 *Thread-Safe Interface* pattern for design rationale

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {

    public void put(E e) ...

    public E take() ...

    ...

    private void enqueue(Node<E> x) ...

    private E dequeue() ...

    ...
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- See the POSA2 *Thread-Safe Interface* pattern for design rationale

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {

    public void put(E e) ...

    public E take() ...

    ...

    private void enqueue(Node<E> x) ...

    private E dequeue() ...

    ...
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- See the POSA2 *Thread-Safe Interface* pattern for design rationale

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {

    public void put(E e) ...

    public E take() ...

    ...

    private void enqueue(Node<E> x) ...

    private E dequeue() ...

    ...
}
```

*Links a node at the end of the queue*

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- See the POSA2 *Thread-Safe Interface* pattern for design rationale

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {

    public void put(E e) ...

    public E take() ...

    ...

    private void enqueue(Node<E> x) ...

    private E dequeue() ...

    ...
}
```

*Removes a node from  
the head of the queue*

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- See the POSA2 *Thread-Safe Interface* pattern for design rationale

*These interface methods  
acquire & release the  
monitor locks*

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {

    public void put(E e) ...

    public E take() ...

    ...

    private void enqueue(Node<E> x) ...

    private E dequeue() ...

    ...
}
```



# Defining Internal State & Synchronization & Scheduling Mechanisms

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms

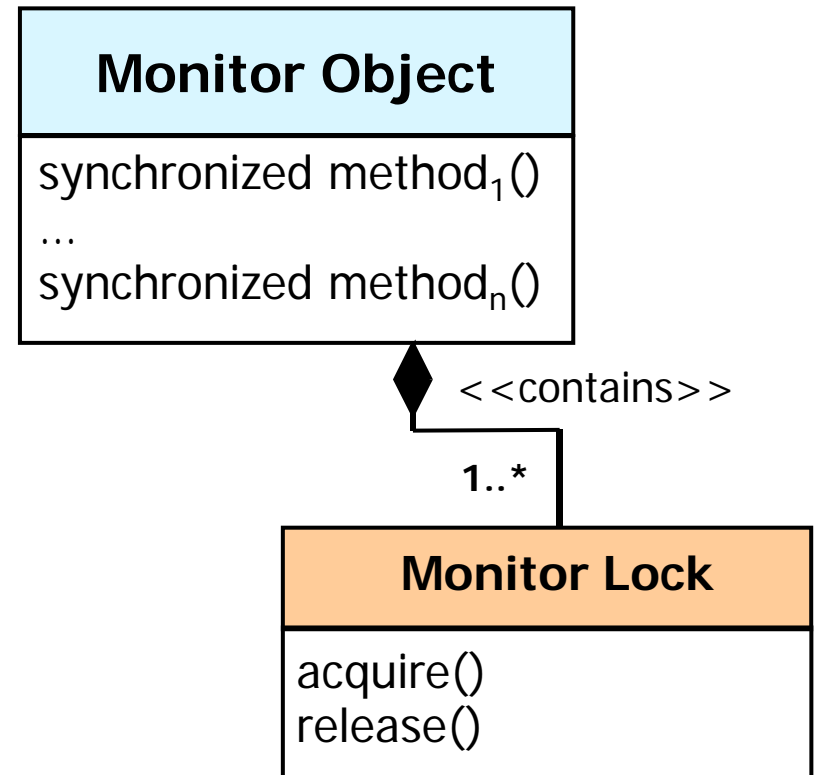


# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms

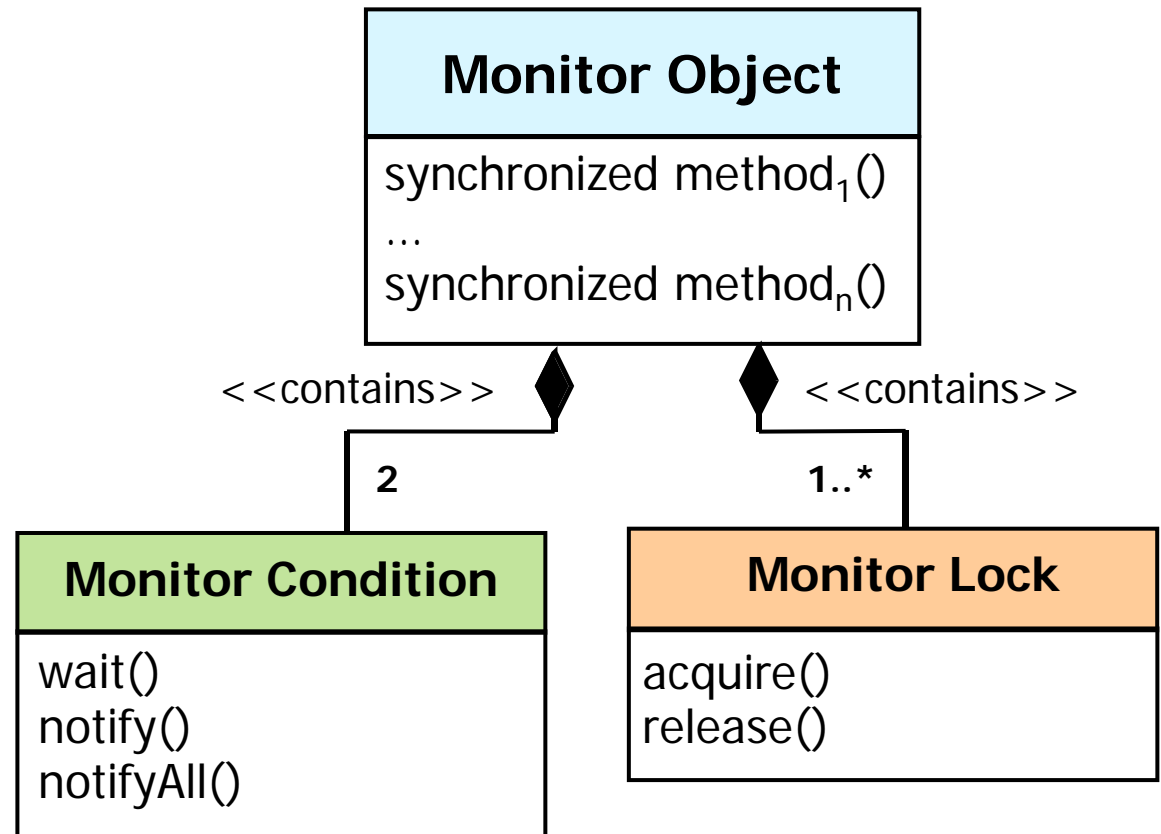


# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms

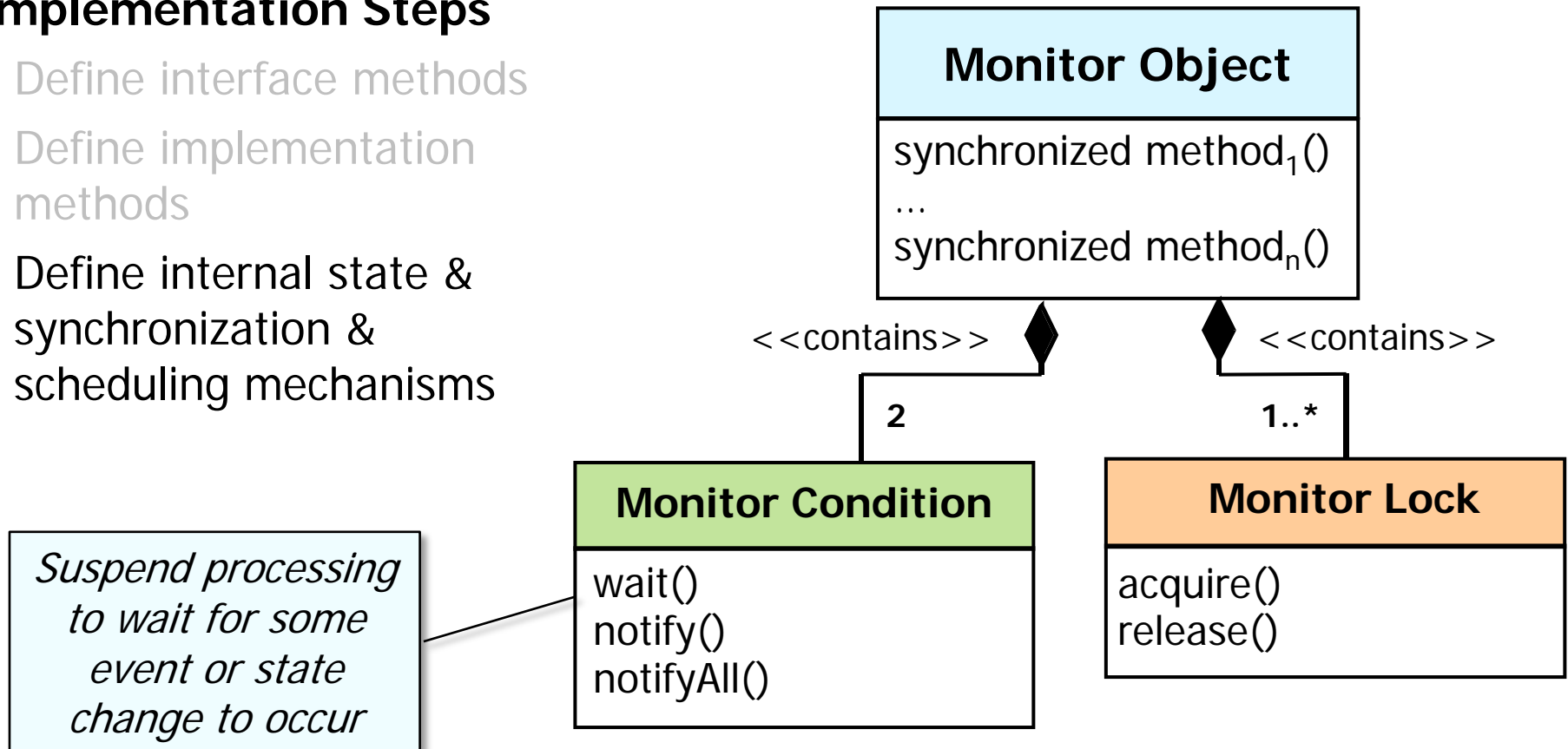


# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms

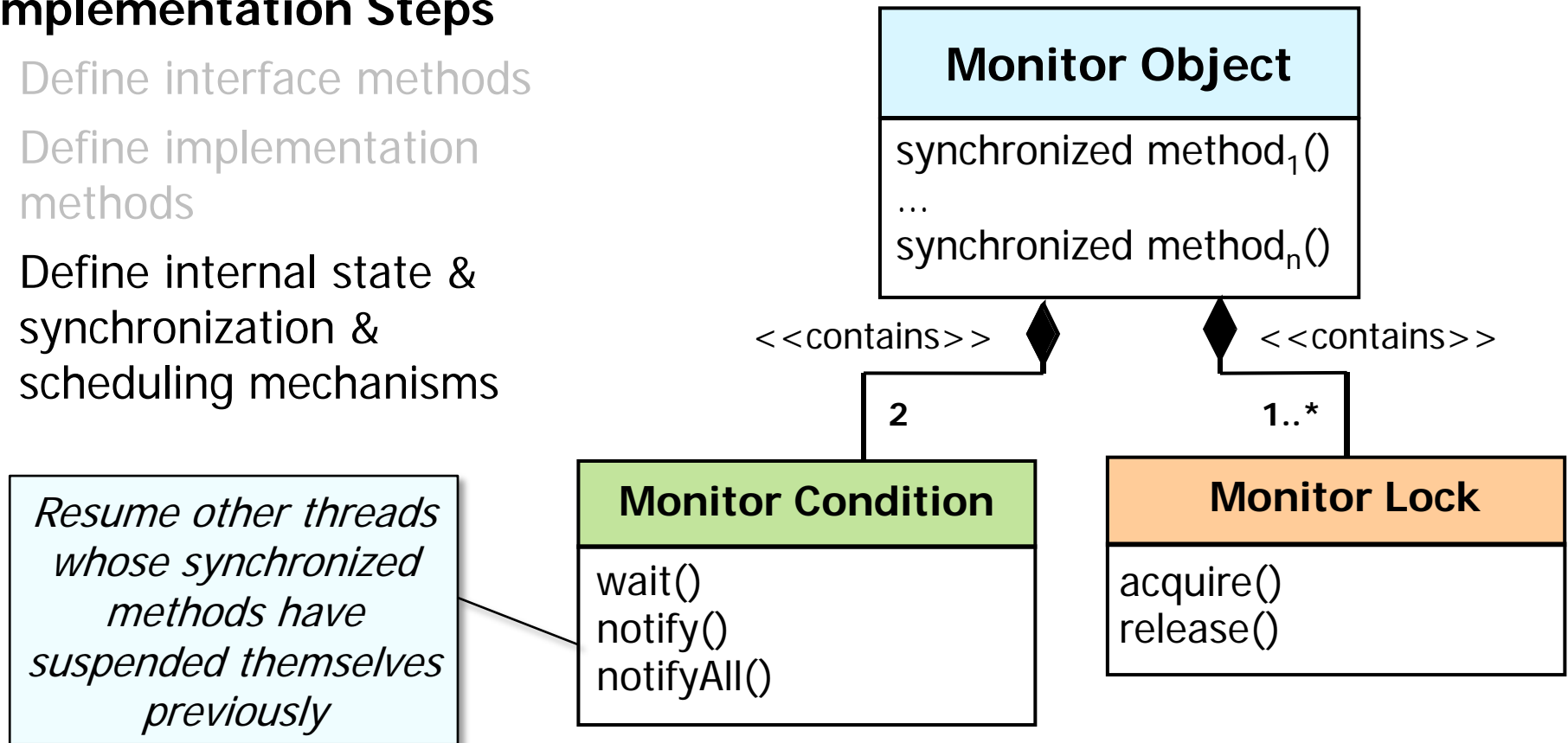


# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms

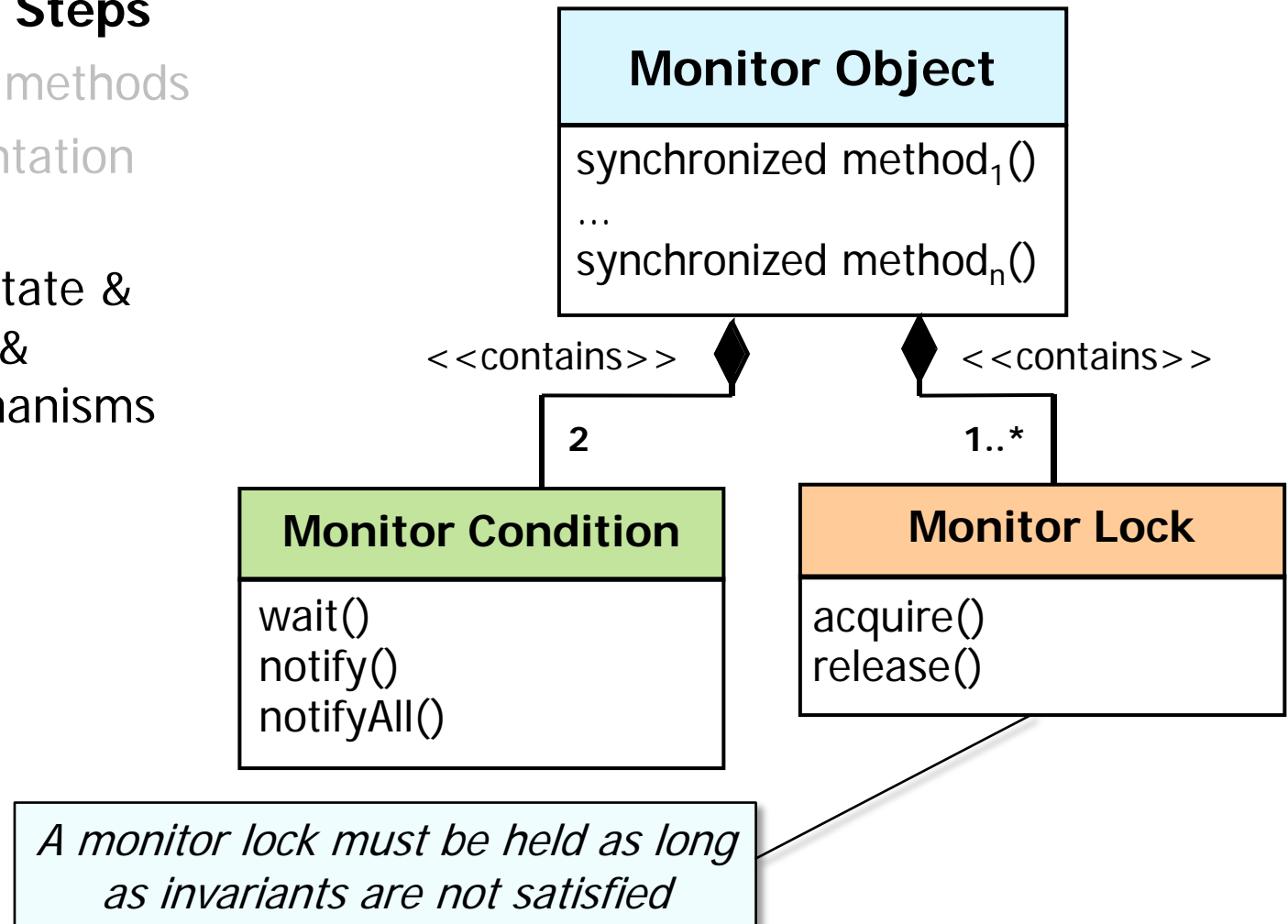


# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms



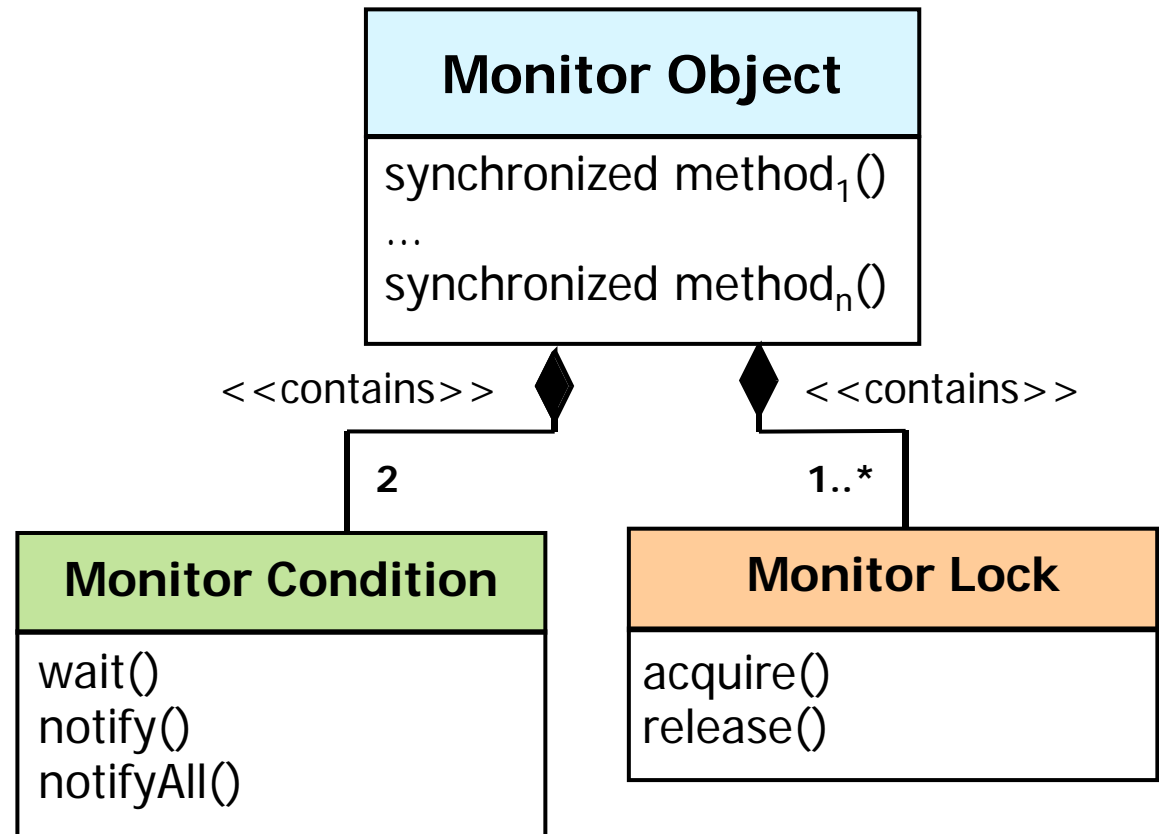


# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
  - Invariants must hold when a synchronized method waits on a monitor condition

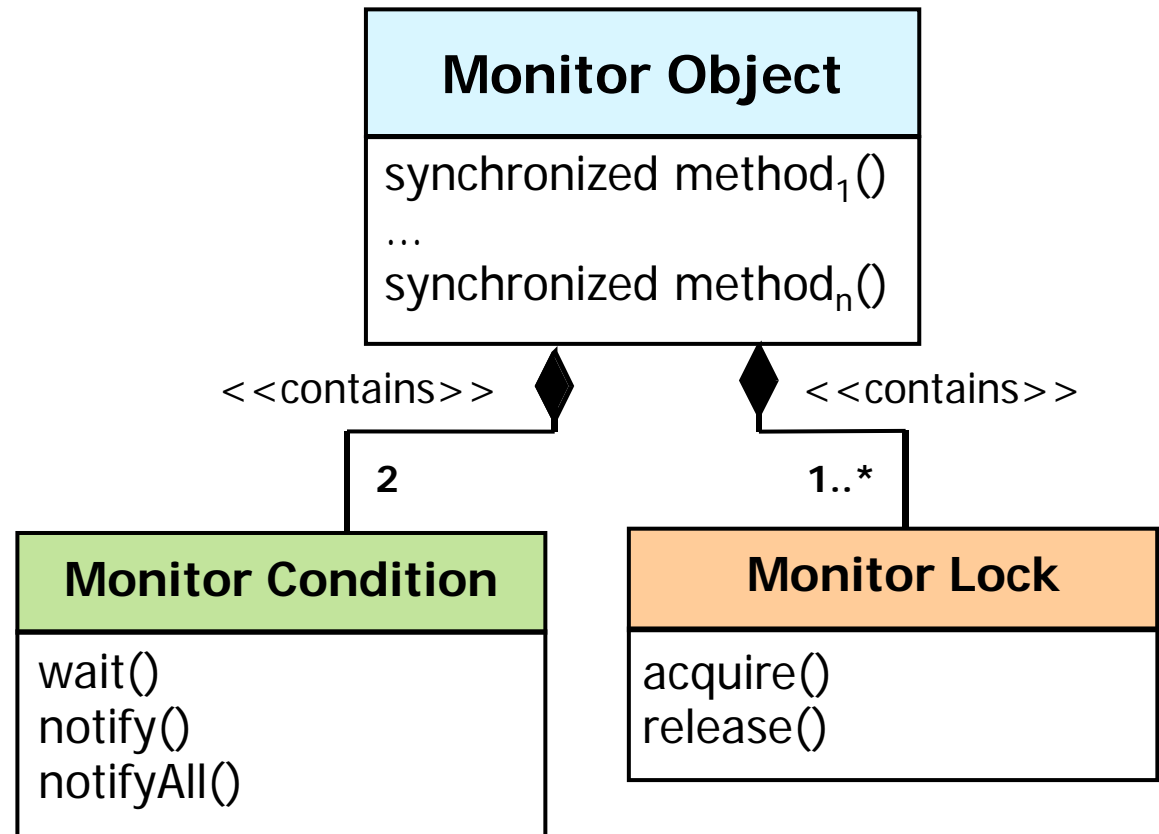


# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
  - Invariants must hold when a synchronized method waits on a monitor condition
  - They must also hold before processing resumes after a monitor condition is notified



# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    static class Node<E> {
        E item;
        Node<E> next;
        Node(E x) { item = x; }
    }

    private transient Node<T> head;
    private transient Node<T> last;
    private final int capacity;
    private final AtomicInteger count
        = new AtomicInteger(0);
    ...
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    static class Node<E> {
        E item;
        Node<E> next;
        Node(E x) { item = x; }
    }

    private transient Node<T> head;
    private transient Node<T> last;
    private final int capacity;
    private final AtomicInteger count
        = new AtomicInteger(0);
    ...
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    static class Node<E> {
        E item;
        Node<E> next;
        Node(E x) { item = x; }
    }

    private transient Node<T> head;
    private transient Node<T> last;
    private final int capacity;
    private final AtomicInteger count
        = new AtomicInteger(0);
    ...
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    static class Node<E> {
        E item;
        Node<E> next;
        Node(E x) { item = x; }
    }

    private transient Node<T> head;
    private transient Node<T> last;
    private final int capacity;
    private final AtomicInteger count
        = new AtomicInteger(0);
    ...
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    static class Node<E> {
        E item;
        Node<E> next;
        Node(E x) { item = x; }
    }

    private transient Node<T> head;
    private transient Node<T> last;
    private final int capacity;
    private final AtomicInteger count
        = new AtomicInteger(0);
    ...
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    static class Node<E> {
        E item;
        Node<E> next;
        Node(E x) { item = x; }
    }

    private transient Node<T> head;
    private transient Node<T> last;
    private final int capacity;
    private final AtomicInteger count
        = new AtomicInteger(0);
    ...
}
```



# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
  - `LinkedBlockingQueue` uses classes defined in the `java.util.concurrent` package

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    private final ReentrantLock takeLock
        ...;
    private final Condition notEmpty
        ...;
    private final ReentrantLock putLock
        ...;
    private final Condition notFull
        ...;
    ...
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
  - `LinkedBlockingQueue` uses classes defined in the `java.util.concurrent` package

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    private final ReentrantLock takeLock
        ...;
    private final Condition notEmpty
        ...;
    private final ReentrantLock putLock
        ...;
    private final Condition notFull
        ...;
    ...
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
  - `LinkedBlockingQueue` uses classes defined in the `java.util.concurrent` package

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    private final ReentrantLock takeLock
        ...;
    private final Condition notEmpty
        ...;
    private final ReentrantLock putLock
        ...;
    private final Condition notFull
        ...;
    ...
}
```

# Implementing All Methods & Data Members

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members

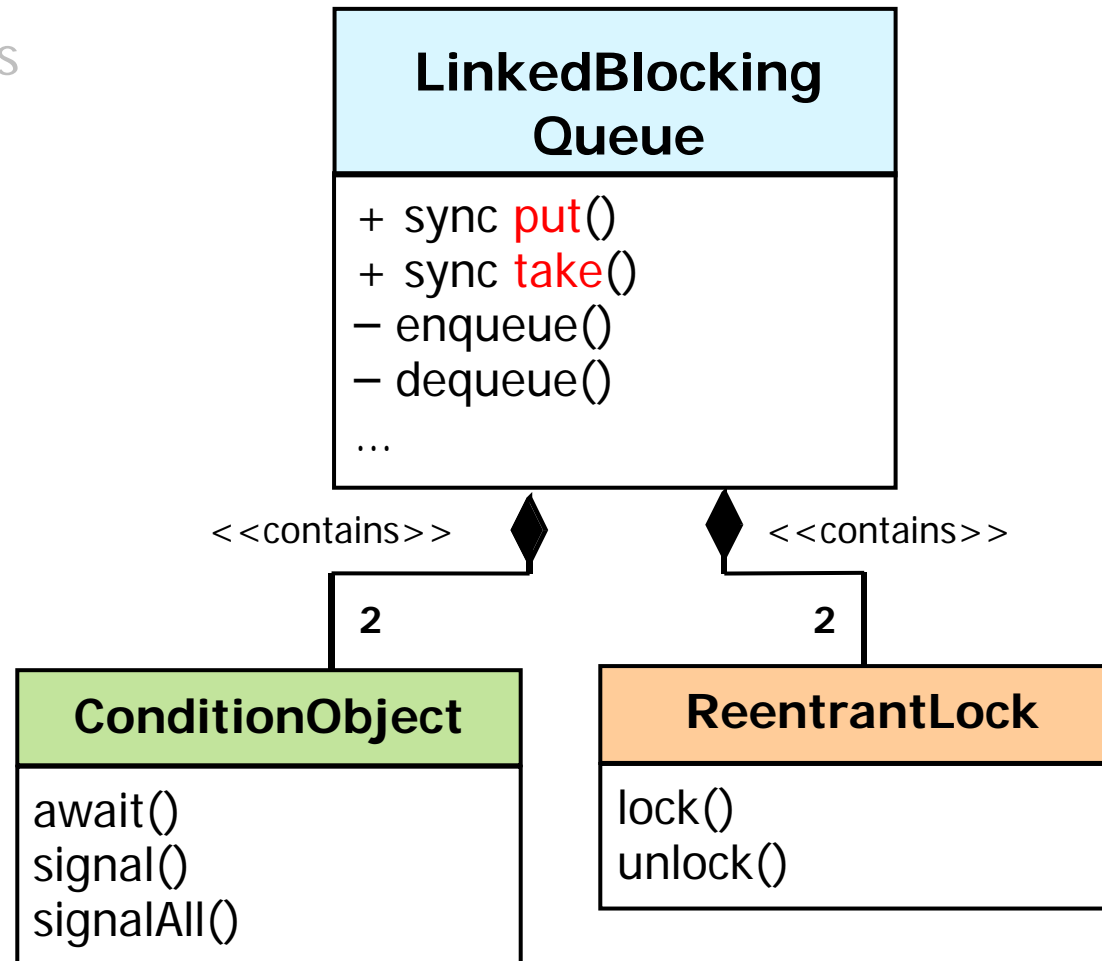


# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members



# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members

```
public class LinkedBlockingQueue<E>  
    extends AbstractQueue<E>  
    implements BlockingQueue<E>, ... {  
    ...  
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    private final ReentrantLock takeLock
        = new ReentrantLock;
    private final Condition notEmpty
        = takeLock.newCondition();
    private final ReentrantLock putLock
        = new ReentrantLock;
    private final Condition notFull
        = putLock.newCondition();
    ...
    private final AtomicInteger count
        = new AtomicInteger(0);
    ...
}
```



# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    private final ReentrantLock takeLock
        = new ReentrantLock();
    private final Condition notEmpty
        = takeLock.newCondition();
    private final ReentrantLock putLock
        = new ReentrantLock();
    private final Condition notFull
        = putLock.newCondition();
    ...
    private final AtomicInteger count
        = new AtomicInteger(0);
    ...
}
```

## Monitor Object

## POSA2 Concurrency

### Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public LinkedBlockingQueue
        (int capacity) {
        if (capacity <= 0)
            throw new
                IllegalArgumentException();
        this.capacity = capacity;
        last = head = new Node<E>(null);
    }
    ...
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public LinkedBlockingQueue
        (int capacity) {
        if (capacity <= 0)
            throw new
                IllegalArgumentException();
        this.capacity = capacity;
        last = head = new Node<E>(null);
    }
    ...
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public void put(E e) ...

    public E take() ...

    ...

    private void enqueue(Node<E> x) ...

    private E dequeue() ...
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public void put(E e) ...

    public E take() ...

    ...

    private void enqueue(Node<E> x) ...

    private E dequeue() ...
```

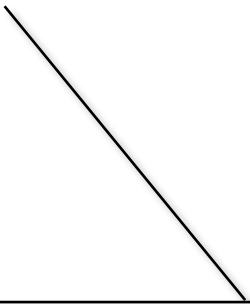
# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public void put(E e) ... {
        ...
```



*Insert the specified element into a queue, waiting if necessary for space to become available*

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public void put(E e) ... {
        ...
        int c = -1;
        Node<E> node = new Node(e);
        final ReentrantLock putlock =
            this.putLock;
        final AtomicInteger count =
            this.count;
        putlock.lockInterruptibly();
        ...
    }
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public void put(E e) ... {
        ...
        int c = -1;
        Node<E> node = new Node(e);
        final ReentrantLock putlock =
            this.putLock;
        final AtomicInteger count =
            this.count;
        putLock.lockInterruptibly();
        ...
    }
}
```



# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public void put(E e) ... {
        ...
        try {
            while (count.get == capacity)
                notFull.await();

            enqueue(node);
            ...
        }
    }
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public void put(E e) ... {
        ...
        try {
            while (count.get == capacity)
                notFull.await();

            enqueue(node);
            ...
        }
    }
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public void put(E e) ... {
        ...
        try {
            while (count.get == capacity)
                notFull.await();

            enqueue(node);
            ...
        }
    }
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public void put(E e) ... {
        ...
        try {
            while (count.get == capacity)
                notFull.await();

            enqueue(node);
            ...
        }
    }
}
```

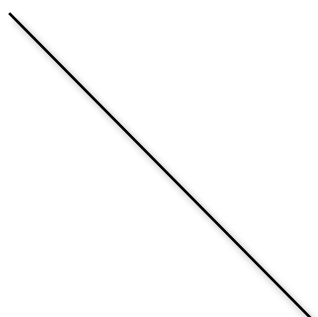
# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    private void enqueue(Node<E> node) {
        last = last.next = node;
    }
    ...
}
```



*Called only when  
the lock is held*

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public void put(E e) ... {
        ...
        try {
            ...
            c = count.getAndIncrement();
            if (c + 1 < capacity)
                notFull.signal();
        } finally {
            putLock.unlock();
        }
        ...
    }
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public void put(E e) ... {
        ...
        try {
            ...
            c = count.getAndIncrement();
            if (c + 1 < capacity)
                notFull.signal();
        } finally {
            putLock.unlock();
        }
        ...
    }
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public void put(E e) ... {
        ...
        try {
            ...
            c = count.getAndIncrement();
            if (c + 1 < capacity)
                notFull.signal();
        } finally {
            putLock.unlock();
        }
        ...
    }
```



# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public void put(E e) ... {
        ...
        if (c == 0)
            signalNotEmpty();
    }
    ...
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public void put(E e) ... {
        ...
        if (c == 0)
            signalNotEmpty();
        }
    ...
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public void put(E e) ... {
        ...
        if (c == 0)
            signalNotEmpty();
    }
    ...

    private void signalNotEmpty() {
        final ReentrantLock takeLock =
            this.takeLock;
        takeLock.lock();
        try { notEmpty.signal(); }
        finally { takeLock.unlock(); }
    }
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public void put(E e) ... {
        ...
        if (c == 0)
            signalNotEmpty();
    }
    ...

    private void signalNotEmpty() {
        final ReentrantLock takeLock =
            this.takeLock;
        takeLock.lock();
        try { notEmpty.signal(); }
        finally { takeLock.unlock(); }
    }
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public void put(E e) ... {
        ...
        if (c == 0)
            signalNotEmpty();
    }
    ...

    private void signalNotEmpty() {
        final ReentrantLock takeLock =
            this.takeLock;
        takeLock.lock();
        try { notEmpty.signal(); }
        finally { takeLock.unlock(); }
    }
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public E take() ... {
        ...
```

*Retrieve & remove the head  
of the queue, waiting if  
necessary until an elements  
becomes available*

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public E take() ... {
        E x; ...
        final AtomicInteger count =
            this.count;
        final ReentrantLock takeLock =
            this.takeLock;
        takeLock.lockInterruptibly();
        try {
            while (count.get() == 0)
                notEmpty.await();
            x = dequeue(); ...
        } finally { takeLock.unlock(); }
        ...
        return x; ...
    }
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public E take() ... {
        E x; ...
        final AtomicInteger count =
            this.count;
        final ReentrantLock takeLock =
            this.takeLock;
        takeLock.lockInterruptibly();
        try {
            while (count.get() == 0)
                notEmpty.await();
            x = dequeue(); ...
        } finally { takeLock.unlock(); }
        ...
        return x; ...
    }
}
```



# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public E take() ... {
        E x; ...
        final AtomicInteger count =
            this.count;
        final ReentrantLock takeLock =
            this.takeLock;
        takeLock.lockInterruptibly();
        try {
            while (count.get() == 0)
                notEmpty.await();
            x = dequeue(); ...
        } finally { takeLock.unlock(); }
        ...
        return x; ...
    }
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public E take() ... {
        E x; ...
        final AtomicInteger count =
            this.count;
        final ReentrantLock takeLock =
            this.takeLock;
        takeLock.lockInterruptibly();
        try {
            while (count.get() == 0)
                notEmpty.await();
            x = dequeue(); ...
        } finally { takeLock.unlock(); }
        ...
        return x; ...
    }
}
```

# Monitor Object

# POSA2 Concurrency

## Implementation Steps

- Define interface methods
- Define implementation methods
- Define internal state & synchronization & scheduling mechanisms
- Implement all methods & data members
  - Initialize data members
  - Apply *Thread-Safe Interface* pattern
  - Apply *Guarded Suspension* pattern

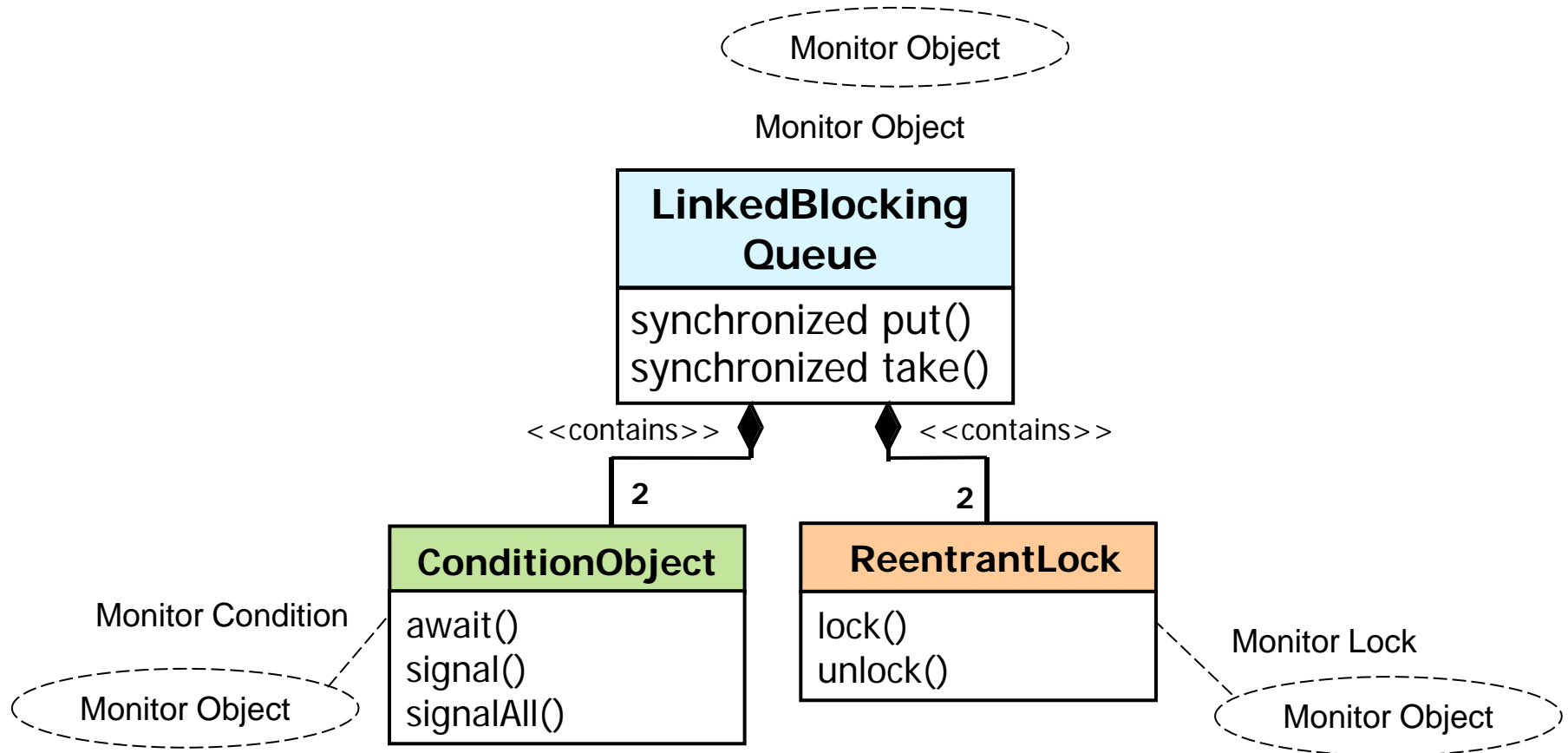
```
public class LinkedBlockingQueue<E>
    extends AbstractQueue<E>
    implements BlockingQueue<E>, ... {
    ...
    public E take() ... {
        E x; ...
        final AtomicInteger count =
            this.count;
        final ReentrantLock takeLock =
            this.takeLock;
        takeLock.lockInterruptibly();
        try {
            while (count.get() == 0)
                notEmpty.await();
            x = dequeue(); ...
        } finally { takeLock.unlock(); }
        ...
        return x; ...
    }
}
```

# Applying Monitor Object in Android

# Monitor Object

# POSA2 Concurrency

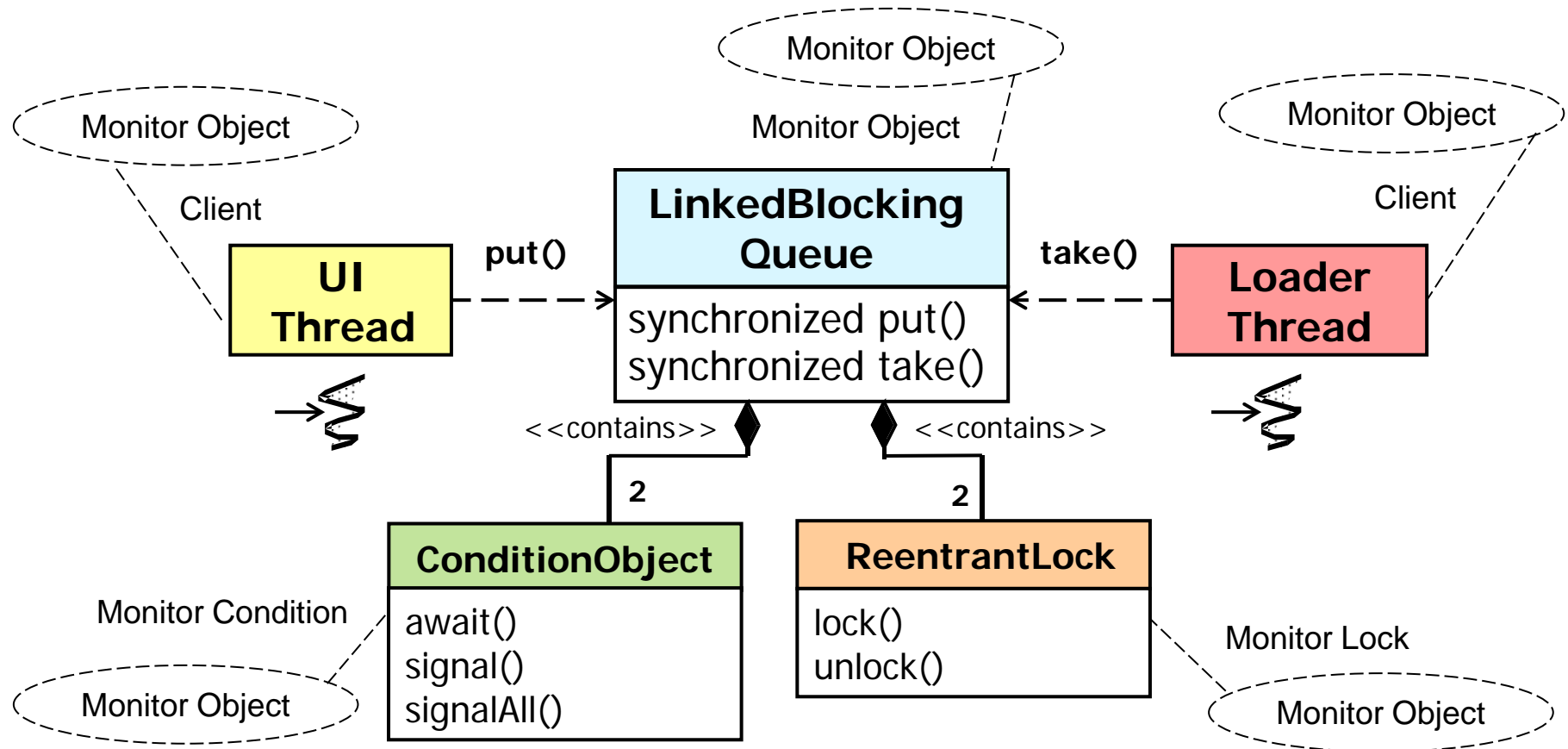
## Applying Monitor Object in Android



# Monitor Object

# POSA2 Concurrency

## Applying Monitor Object in Android

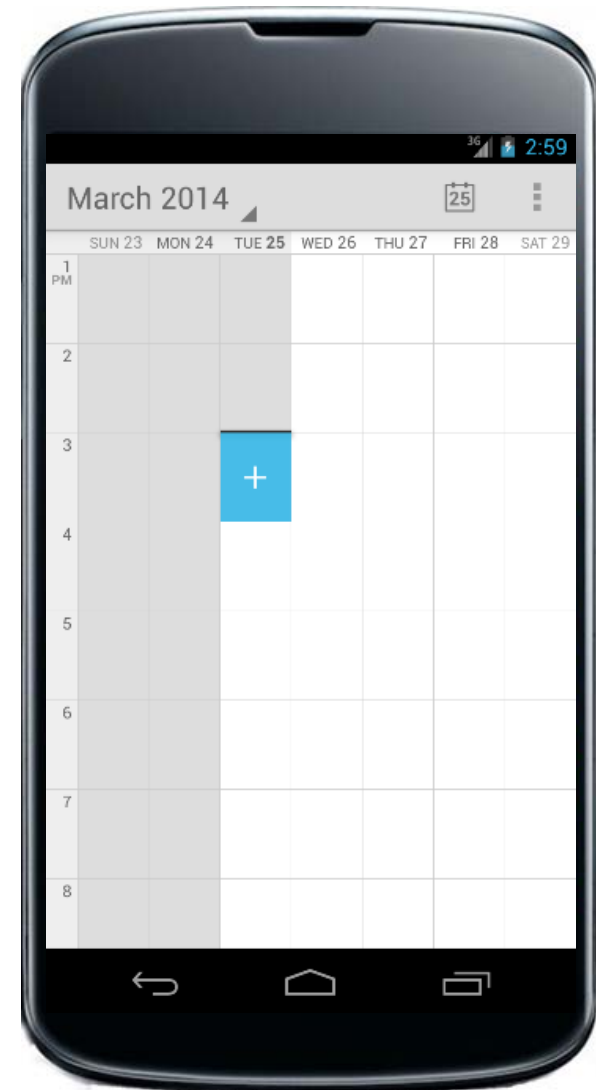


## Monitor Object

## POSA2 Concurrency

### Applying Monitor Object in Android

- Android's Calendar application manages events from each account synchronized with a device



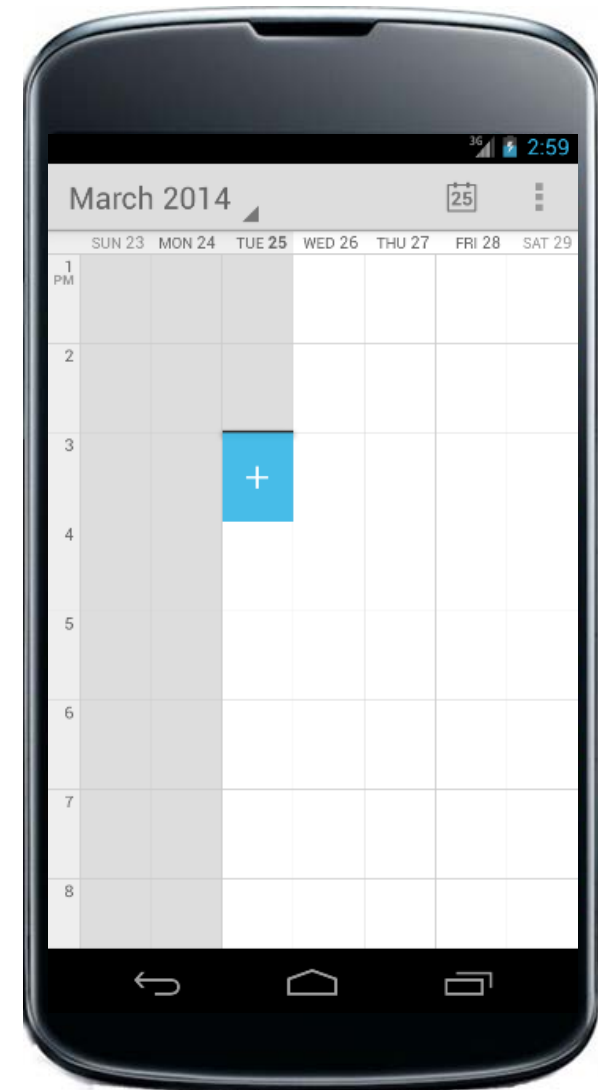
See [packages/apps/Calendar](https://source.android.com/packages/apps/Calendar) for the Calendar application source code

## Monitor Object

## POSA2 Concurrency

### Applying Monitor Object in Android

- Android's Calendar application manages events from each account synchronized with a device
- e.g., it displays, creates, edits, & deletes calendar events



See [packages/apps/Calendar](https://source.android.com/packages/apps/Calendar) for the Calendar application source code



## Monitor Object

## POSA2 Concurrency

### Applying Monitor Object in Android

- Android's Calendar application manages events from each account synchronized with a device
- EventLoader loads calendar events in a LoaderThread based on requests passed to it from UI thread via a LinkedBlockingQueue

```
public class EventLoader {  
    ...  
    private LinkedBlockingQueue  
        <LoadRequest> mLoaderQueue;  
  
    public EventLoader(...) {  
        ...  
        mLoaderQueue = new  
            LinkedBlockingQueue  
                <LoadRequest>();  
        ...  
    }  
}
```

## Monitor Object

## POSA2 Concurrency

### Applying Monitor Object in Android

- Android's Calendar application manages events from each account synchronized with a device
- EventLoader loads calendar events in a LoaderThread based on requests passed to it from UI thread via a `LinkedBlockingQueue`

```
public class EventLoader {  
    ...  
    private LinkedBlockingQueue  
        <LoadRequest> mLoaderQueue;  
  
    public EventLoader(...) {  
        ...  
        mLoaderQueue = new  
            LinkedBlockingQueue  
                <LoadRequest>();  
        ...  
    }
```

## Monitor Object

## POSA2 Concurrency

### Applying Monitor Object in Android

- Android's Calendar application manages events from each account synchronized with a device
- EventLoader loads calendar events in a LoaderThread based on requests passed to it from UI thread via a LinkedBlockingQueue

```
public class EventLoader {  
    ...  
    private LinkedBlockingQueue  
        <LoadRequest> mLoaderQueue;  
  
    public EventLoader(...) {  
        ...  
        mLoaderQueue = new  
            LinkedBlockingQueue  
                <LoadRequest>();  
        ...  
    }  
}
```

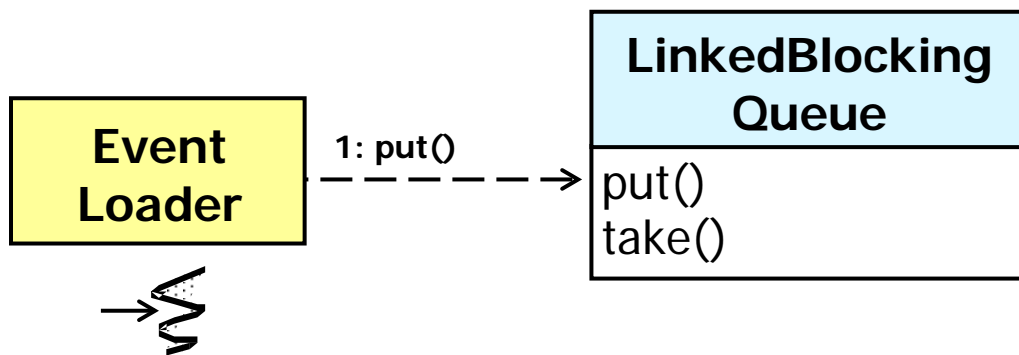
# Monitor Object

# POSA2 Concurrency

## Applying Monitor Object in Android

- Android's Calendar application manages events from each account synchronized with a device
- EventLoader loads calendar events in a LoaderThread based on requests passed to it from UI thread via a LinkedBlockingQueue

```
public class EventLoader {  
    ...  
    private LinkedBlockingQueue  
        <LoadRequest> mLoaderQueue;  
}
```



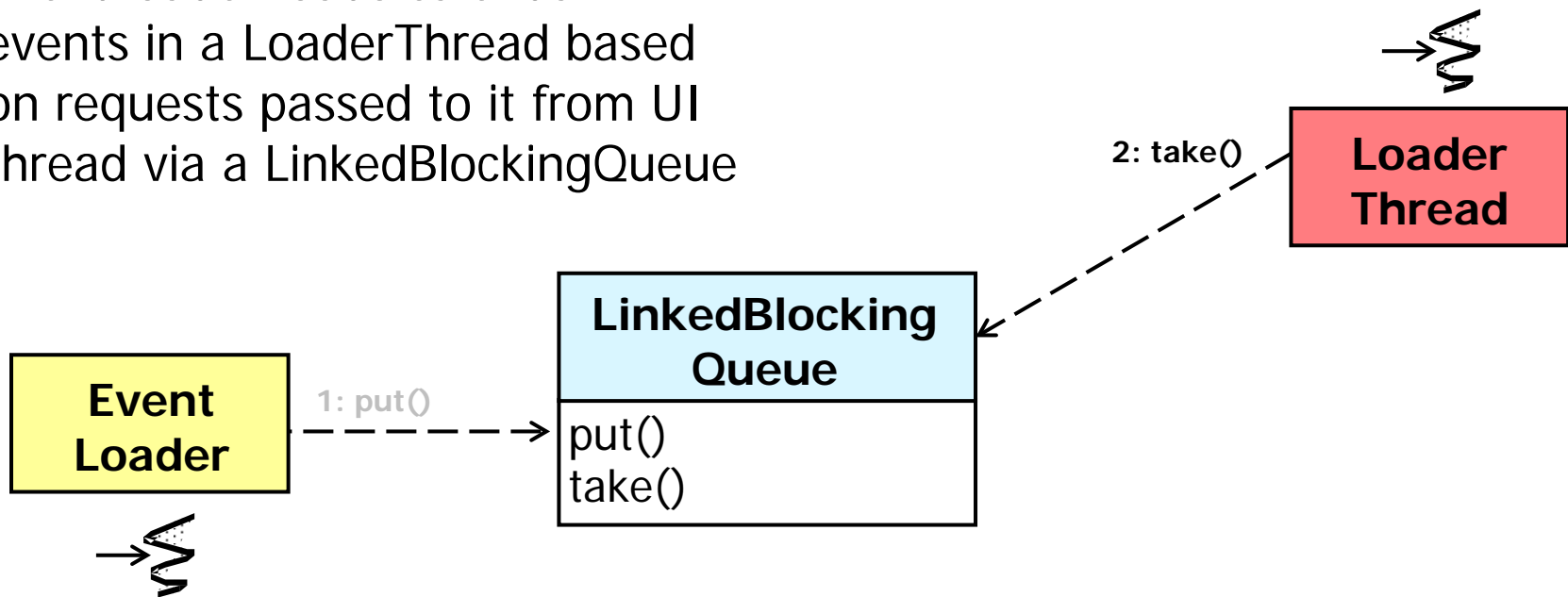
# Monitor Object

# POSA2 Concurrency

## Applying Monitor Object in Android

- Android's Calendar application manages events from each account synchronized with a device
- EventLoader loads calendar events in a LoaderThread based on requests passed to it from UI thread via a LinkedBlockingQueue

```
public class EventLoader {  
    ...  
    private LinkedBlockingQueue  
        <LoadRequest> mLoaderQueue;  
}
```



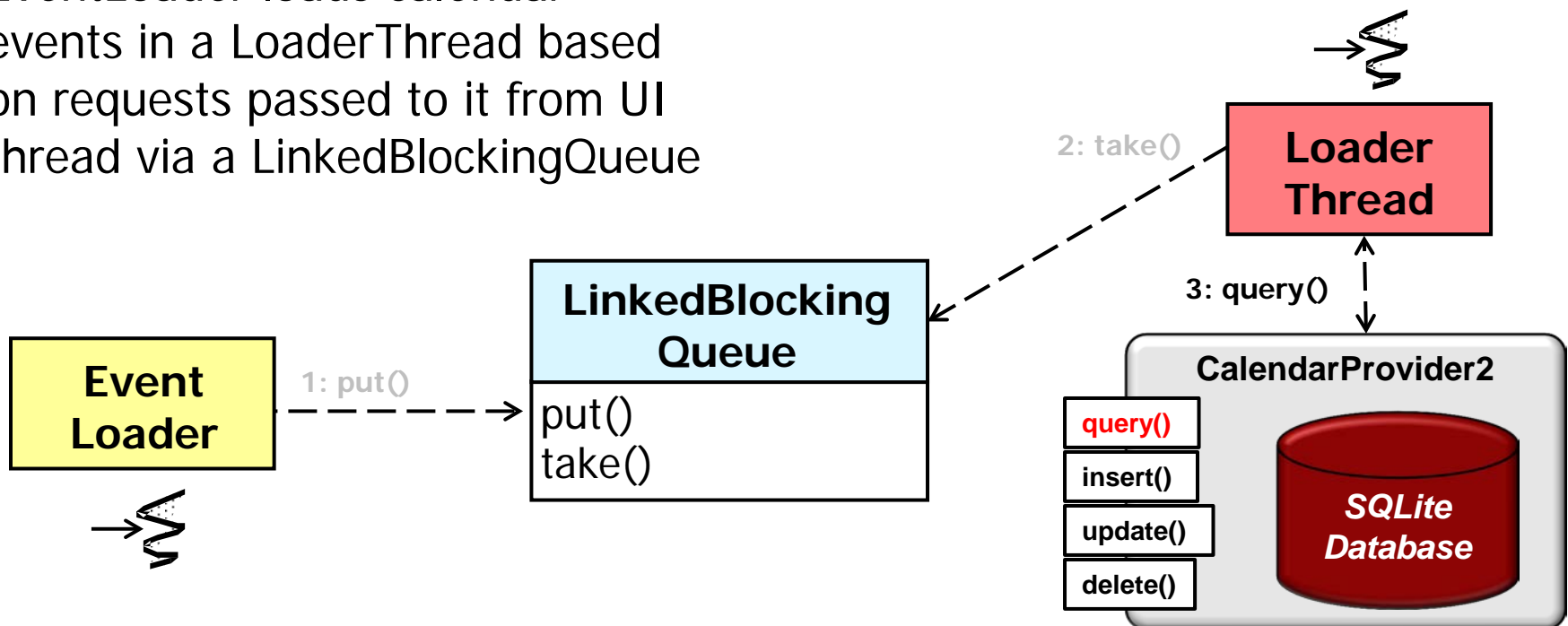
# Monitor Object

# POSA2 Concurrency

## Applying Monitor Object in Android

- Android's Calendar application manages events from each account synchronized with a device
- EventLoader loads calendar events in a LoaderThread based on requests passed to it from UI thread via a LinkedBlockingQueue

```
public class EventLoader {  
    ...  
    private LinkedBlockingQueue  
        <LoadRequest> mLoaderQueue;  
}
```



## Monitor Object

## POSA2 Concurrency

### Applying Monitor Object in Android

- Android's Calendar application manages events from each account synchronized with a device
- EventLoader loads calendar events in a LoaderThread based on requests passed to it from UI thread via a LinkedBlockingQueue

```
public class EventLoader {  
    ...  
    public void  
        startBackgroundThread() {  
        mLoaderThread =  
            new LoaderThread  
                (mLoaderQueue, this);  
        mLoaderThread.start();  
        ...  
    }
```

## Monitor Object

## POSA2 Concurrency

### Applying Monitor Object in Android

- Android's Calendar application manages events from each account synchronized with a device
- EventLoader loads calendar events in a LoaderThread based on requests passed to it from UI thread via a LinkedBlockingQueue
- The `loadEventsInBackground()` method sends a load request to the LoaderThread via `queue.put()`

```
public class EventLoader {  
    ...  
    public void  
        loadEventsInBackground(...) {  
        ...  
        LoadEventsRequest request =  
            new LoadEventsRequest(...);  
        ...  
        mLoaderQueue.put(request);  
        ...  
    }
```



## Monitor Object

## POSA2 Concurrency

### Applying Monitor Object in Android

- Android's Calendar application manages events from each account synchronized with a device
- EventLoader loads calendar events in a LoaderThread based on requests passed to it from UI thread via a LinkedBlockingQueue
- The loadEventsInBackground() method sends a load request to the LoaderThread via queue put()

```
public class EventLoader {  
    ...  
    public void  
        loadEventsInBackground(...) {  
        ...  
        LoadEventsRequest request =  
            new LoadEventsRequest(...);  
        ...  
        mLoaderQueue.put(request);  
        ...  
    }  
}
```

## Monitor Object

## POSA2 Concurrency

### Applying Monitor Object in Android

- Android's Calendar application manages events from each account synchronized with a device
- EventLoader loads calendar events in a LoaderThread based on requests passed to it from UI thread via a LinkedBlockingQueue
- The loadEventsInBackground() method sends a load request to the LoaderThread via queue put()
- LoaderThread run() blocks on queue take() waiting for new requests

```
public class EventLoader {  
    ...  
    private static class  
        LoaderThread extends Thread {  
        LinkedBlockingQueue  
            <LoadRequest> mQueue;  
        ...  
        public void run() {  
            while (true) {  
                ...  
                LoadRequest request =  
                    mQueue.take();  
                ...  
                request.processRequest  
                    (...);  
            }  
        }  
    }  
}
```

## Monitor Object

## POSA2 Concurrency

### Applying Monitor Object in Android

- Android's Calendar application manages events from each account synchronized with a device
- EventLoader loads calendar events in a LoaderThread based on requests passed to it from UI thread via a LinkedBlockingQueue
- The loadEventsInBackground() method sends a load request to the LoaderThread via queue put()
- LoaderThread run() blocks on queue take() waiting for new requests

```
public class EventLoader {  
    ...  
    private static class  
        LoaderThread extends Thread {  
        LinkedBlockingQueue  
            <LoadRequest> mQueue;  
        ...  
        public void run() {  
            while (true) {  
                ...  
                LoadRequest request =  
                    mQueue.take();  
                ...  
                request.processRequest  
                    (...);  
            }  
        }  
    }  
}
```

## Monitor Object

## POSA2 Concurrency

### Applying Monitor Object in Android

- Android's Calendar application manages events from each account synchronized with a device
- EventLoader loads calendar events in a LoaderThread based on requests passed to it from UI thread via a LinkedBlockingQueue
- The loadEventsInBackground() method sends a load request to the LoaderThread via queue put()
- LoaderThread run() blocks on queue take() waiting for new requests
  - It processes these requests by querying the Calendar Provider

```
public class EventLoader {  
    ...  
    private static class  
        LoaderThread extends Thread {  
        LinkedBlockingQueue  
            <LoadRequest> mQueue;  
        ...  
        public void run() {  
            while (true) {  
                ...  
                LoadRequest request =  
                    mQueue.take();  
                ...  
                request.processRequest  
                    (...);  
            }  
        }  
    }  
}
```

## Monitor Object

## POSA2 Concurrency

### Applying Monitor Object in Android

- Android's Calendar application manages events from each account synchronized with a device
- EventLoader loads calendar events in a LoaderThread based on requests passed to it from UI thread via a LinkedBlockingQueue
- The loadEventsInBackground() method sends a load request to the LoaderThread via queue put()
- LoaderThread run() blocks on queue take() waiting for new requests
  - It processes these requests by querying the Calendar Provider

```
public class EventLoader {  
    ...  
    private static class  
        LoadEventsRequest ... {  
        ...  
        public void processRequest  
            (EventLoader eventLoader) {  
            Event.loadEvents(...);  
            ...  
        }  
    }  
}
```

# Monitor Object

# POSA2 Concurrency

## Applying Monitor Object in Android

- Android's Calendar application manages events from each account synchronized with a device
- EventLoader loads calendar events in a LoaderThread based on requests passed to it from UI thread via a LinkedBlockingQueue
- The loadEventsInBackground() method sends a load request to the LoaderThread via queue put()
- LoaderThread run() blocks on queue take() waiting for new requests
  - It processes these requests by querying the Calendar Provider

```
public class EventLoader {  
    ...  
    private static class  
        LoadEventsRequest ... {  
        ...  
        public void processRequest  
            (EventLoader eventLoader) {  
            Event.loadEvents(...);  
            ...  
        }  
    }  
}
```

*These long-running  
queries run in the  
background thread*

## Monitor Object

## POSA2 Concurrency

### Applying Monitor Object in Android

- Android's Calendar application manages events from each account synchronized with a device
- EventLoader loads calendar events in a LoaderThread based on requests passed to it from UI thread via a LinkedBlockingQueue
- The loadEventsInBackground() method sends a load request to the LoaderThread via queue put()
- LoaderThread run() blocks on queue take() waiting for new requests
- processRequest() posts a callback to the UI thread

```
public class EventLoader {  
    ...  
    private static class  
        LoadEventsRequest ... {  
        ...  
        public void processRequest  
            (EventLoader eventLoader) {  
            Event.loadEvents(...);  
            ...  
            eventLoader.mHandler.  
                post(successCallback);  
            ...  
        }  
    }  
}
```

# Monitor Object

# POSA2 Concurrency

## Applying Monitor Object in Android

- Android's Calendar application manages events from each account synchronized with a device
- EventLoader loads calendar events in a LoaderThread based on requests passed to it from UI thread via a LinkedBlockingQueue
- The loadEventsInBackground() method sends a load request to the LoaderThread via queue put()
- LoaderThread run() blocks on queue take() waiting for new requests
- processRequest() posts a callback to the UI thread

```
public class EventLoader {  
    ...  
    private static class  
        LoadEventsRequest ... {  
        ...  
        public void processRequest  
            (EventLoader eventLoader) {  
            Event.loadEvents(...);  
            ...  
            eventLoader.mHandler.  
                post(successCallback);  
            ...  
        }  
    }  
}
```

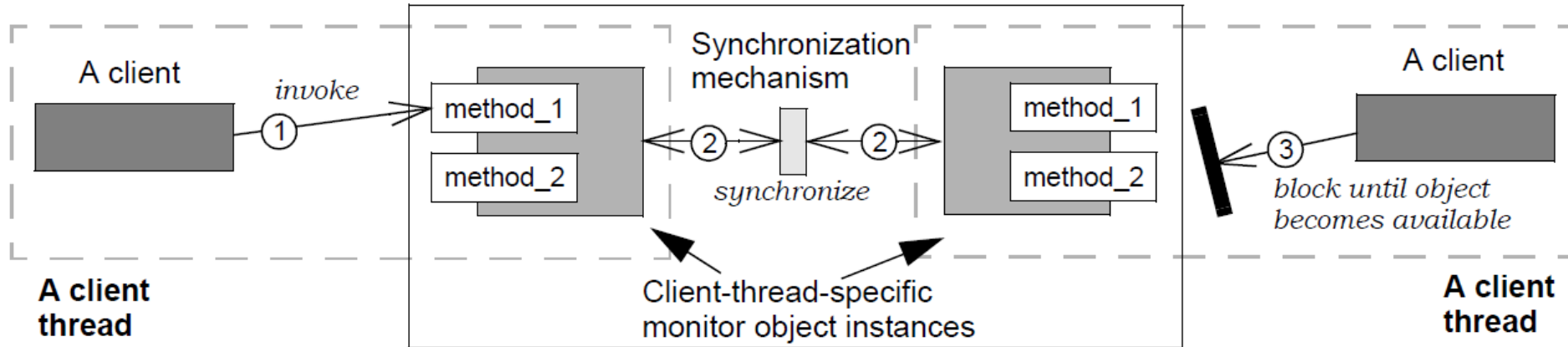


# Summary



# Summary

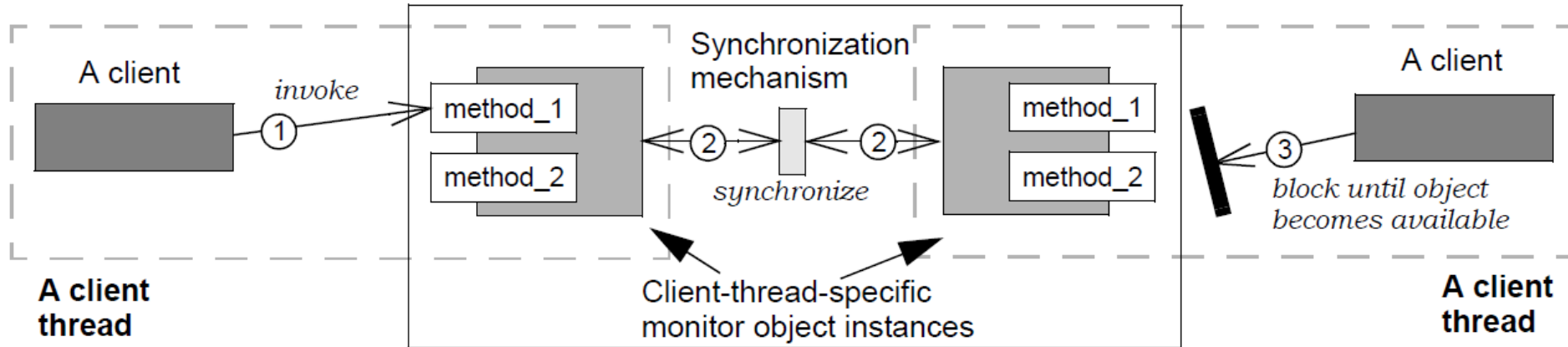
### Monitor object



- Android applies *Monitor Object* to many class libraries & frameworks accessed concurrently by multiple threads

# Summary

### Monitor object



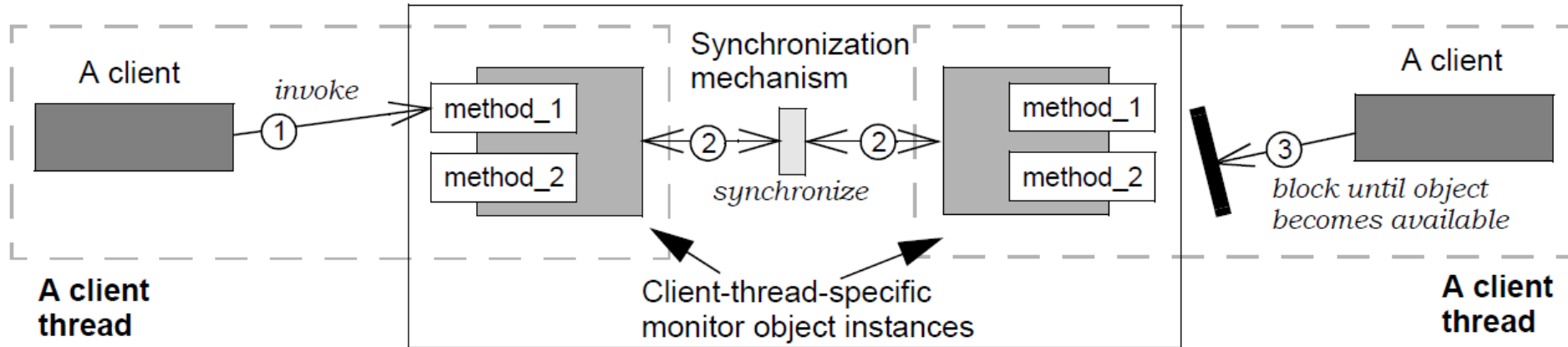
- Android applies *Monitor Object* to many class libraries & frameworks accessed concurrently by multiple threads
- e.g., AsyncTask,
  - \*BlockingQueue,
  - LinkedBlockingDeque,
  - \*ThreadPoolExecutor,
  - etc.

### java.util.concurrent

Utility classes commonly useful in concurrent programming. This package includes a few small standardized extensible frameworks, as well as some classes that provide useful functionality and are otherwise tedious or difficult to implement. Here are brief descriptions of the main components. See also the `java.util.concurrent.locks` and `java.util.concurrent.atomic` packages.

# Summary

### Monitor object

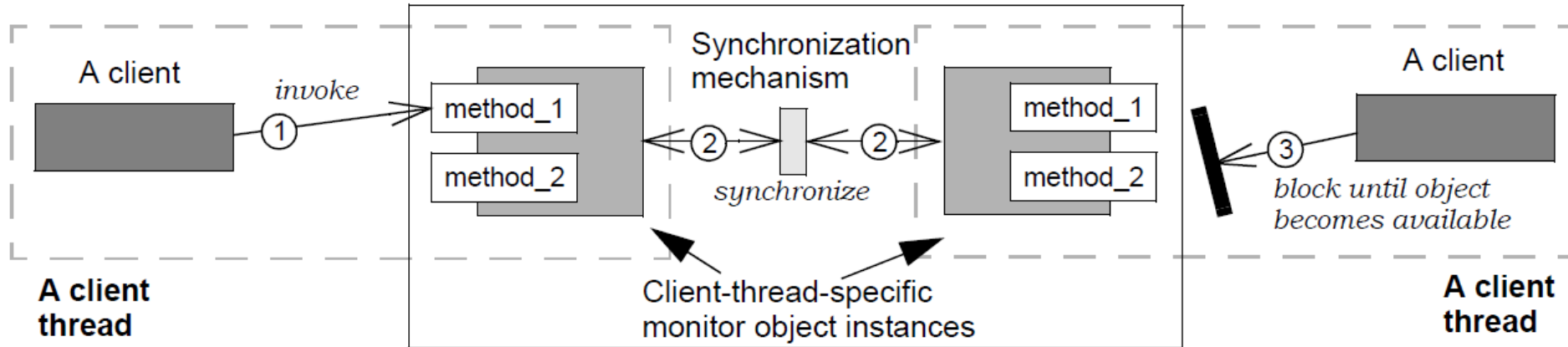


- Android applies *Monitor Object* to many class libraries & frameworks accessed concurrently by multiple threads
- Android's monitor objects use `java.util.concurrent` classes more than Java's built-in monitor objects



# Summary

### Monitor object

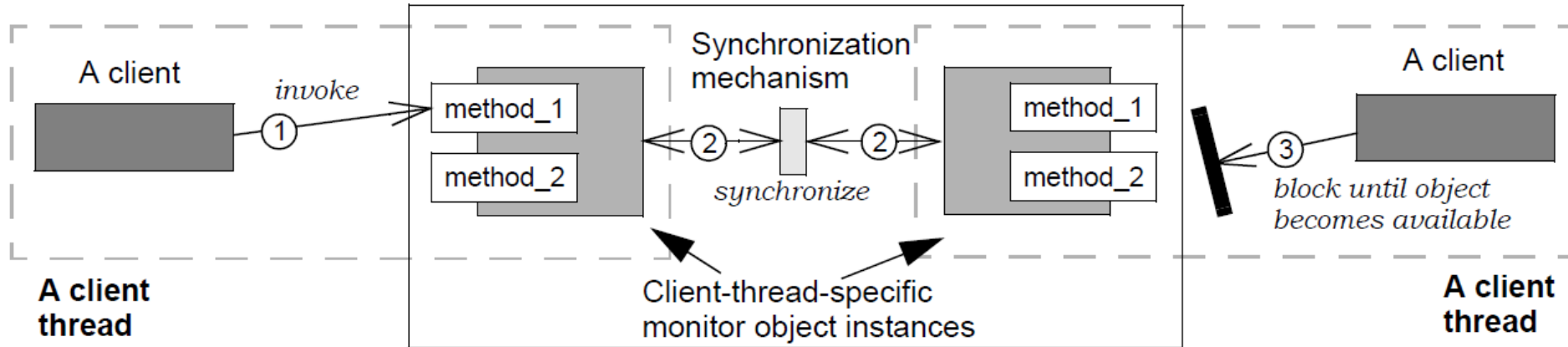


- Android applies *Monitor Object* to many class libraries & frameworks accessed concurrently by multiple threads
- Android's monitor objects use `java.util.concurrent` classes more than Java's built-in monitor objects
- `java.util.concurrent` provides greater flexibility & capabilities



# Summary

### Monitor object



- Android applies *Monitor Object* to many class libraries & frameworks accessed concurrently by multiple threads
- Android's monitor objects use `java.util.concurrent` classes more than Java's built-in monitor objects
- The *Monitor Object* pattern guides these Android components

