LINEARIZABILITY

Concurrent Objects

- Can be called concurrently by many threads
- Examples
  - Work Stealing Queue

Concurrent Objects

- Can be called concurrently by many entities
- Examples
  - Work Stealing Queue
  - C Runtime library
  - Operating System
  - Data bases

Correctness Criteria

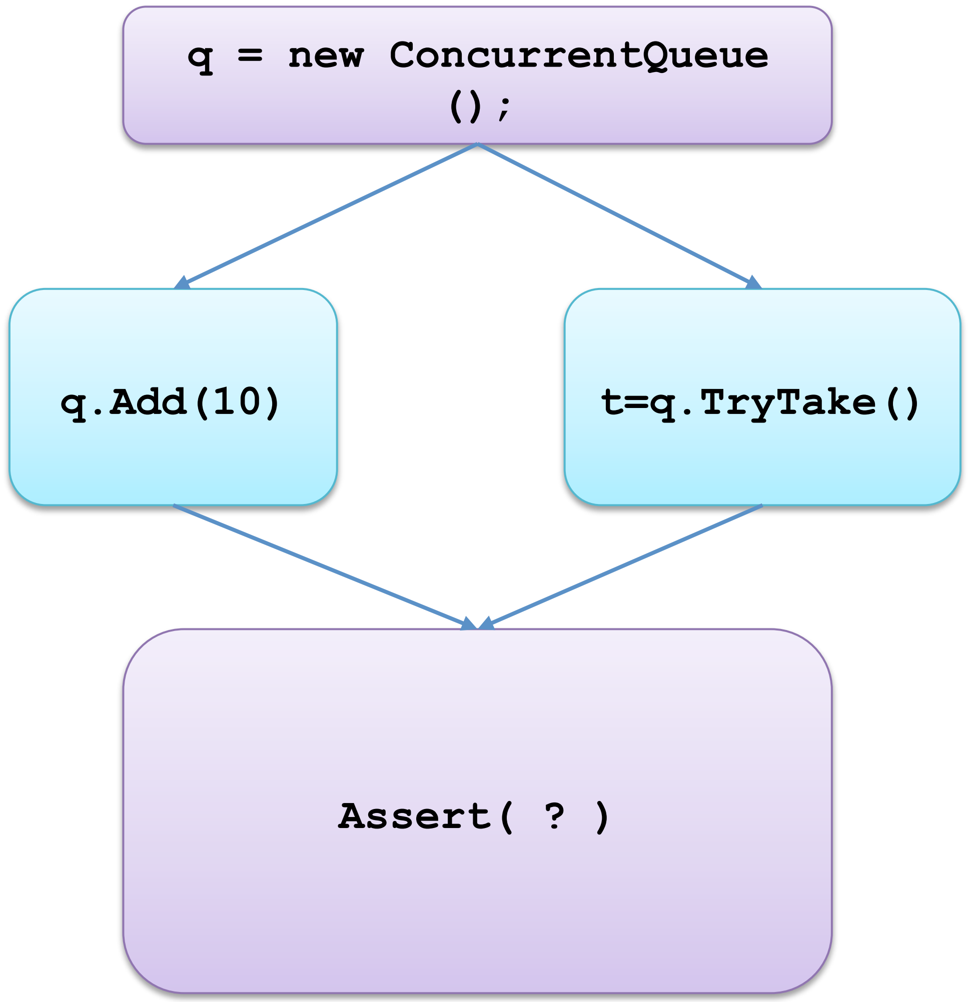
- Informally called “thread safety”
- What does “thread safety” mean to you?

A Simple Concurrent Object

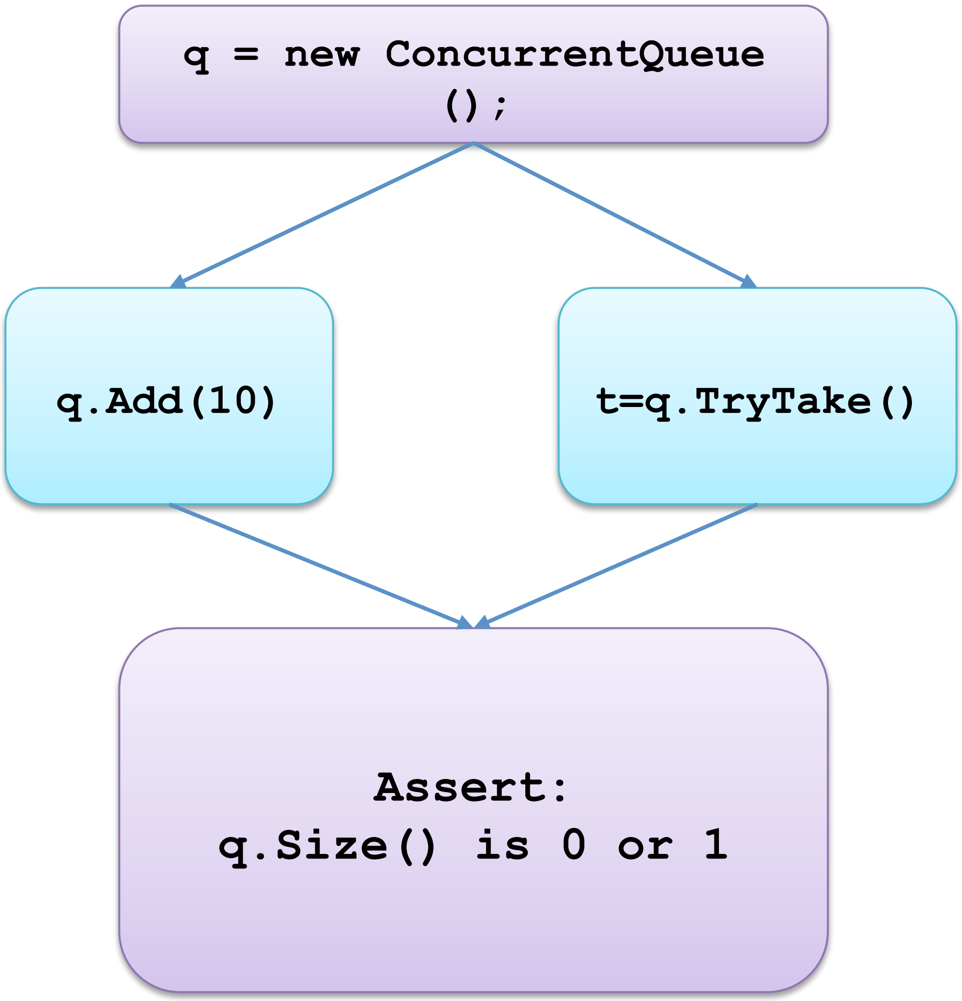
- Sequential Queue
  - Add(item)
  - TryTake() returns an item or “empty”
  - Size() returns # of items in queue

- Consider ConcurrentQueue and its relationship to Queue

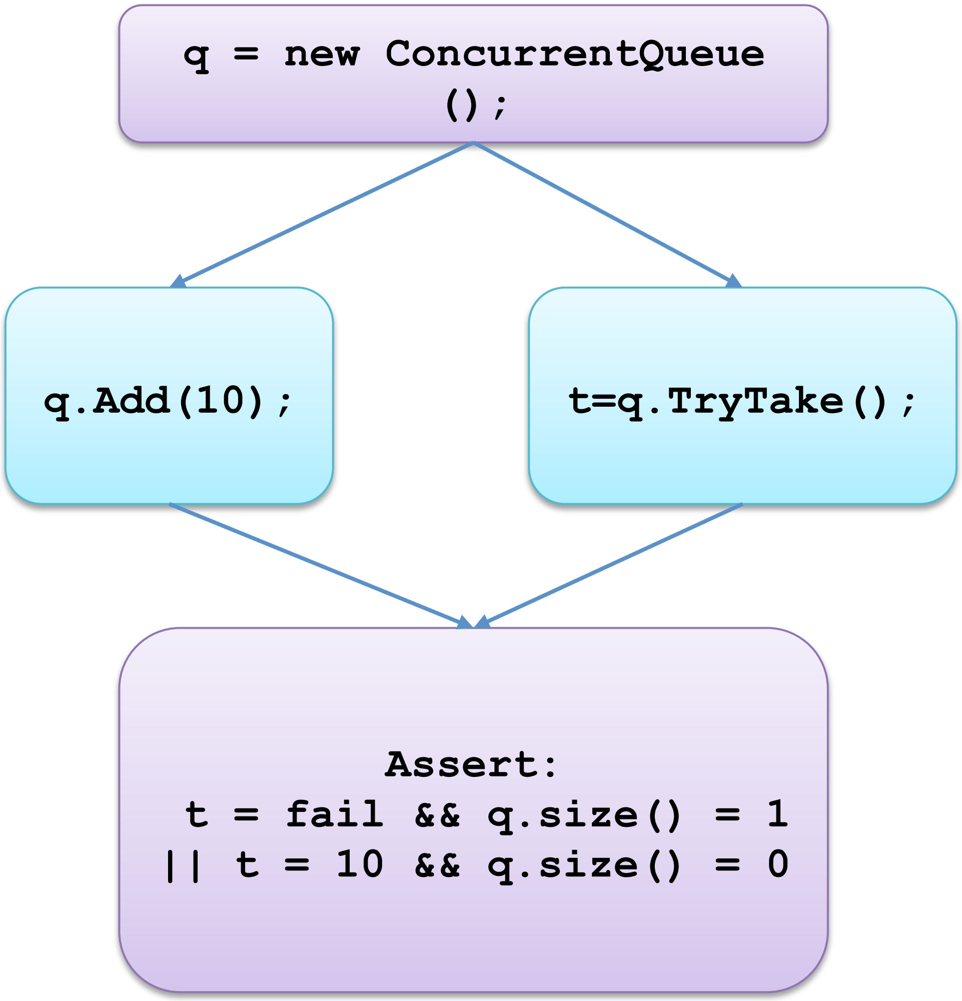
Let’s Write a Test



Let’s Write a Test



Let’s Write a Test



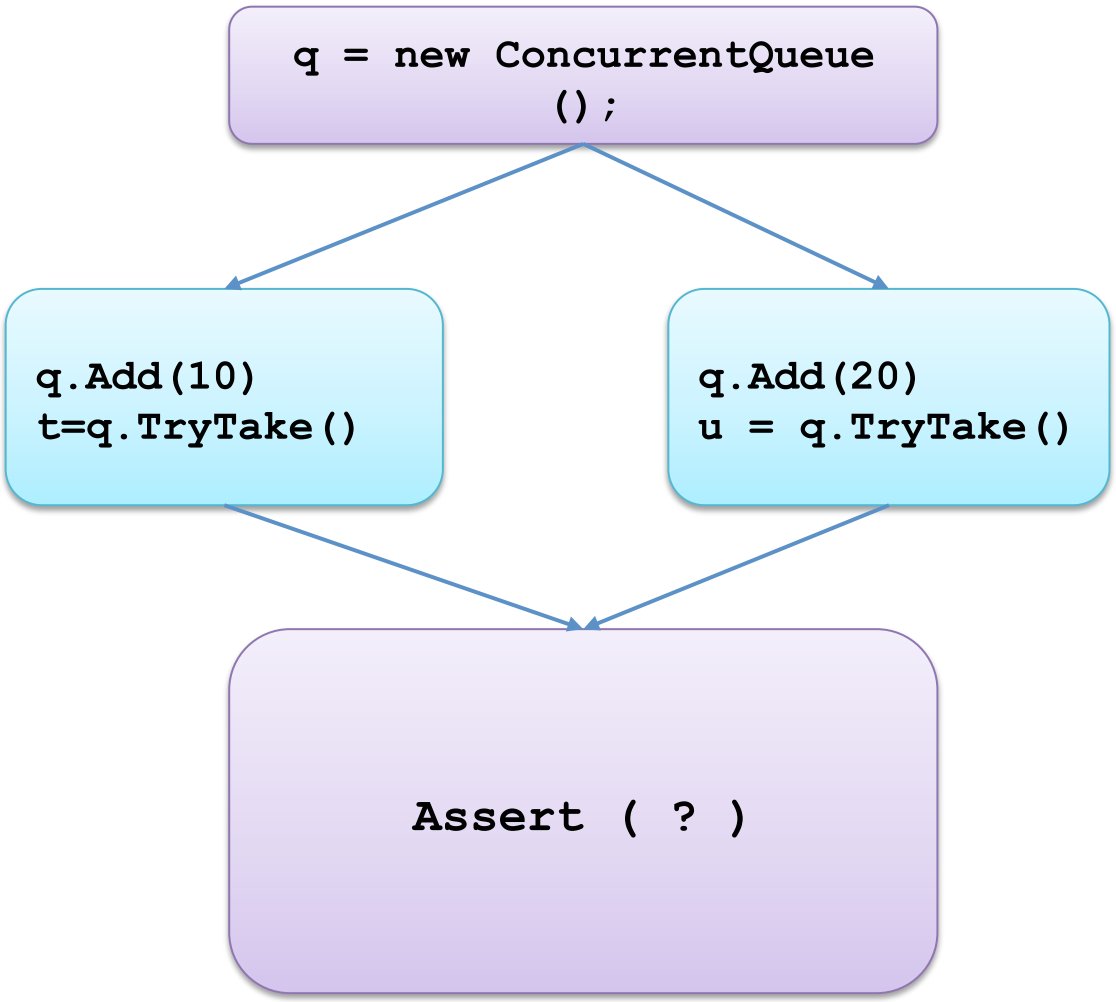
```
q = new ConcurrentQueue
();
```

```
q.Add(10);
```

```
t=q.TryTake();
```

```
Assert:
 t = fail && q.size() = 1
|| t = 10 && q.size() = 0
```

Let’s Write a Test



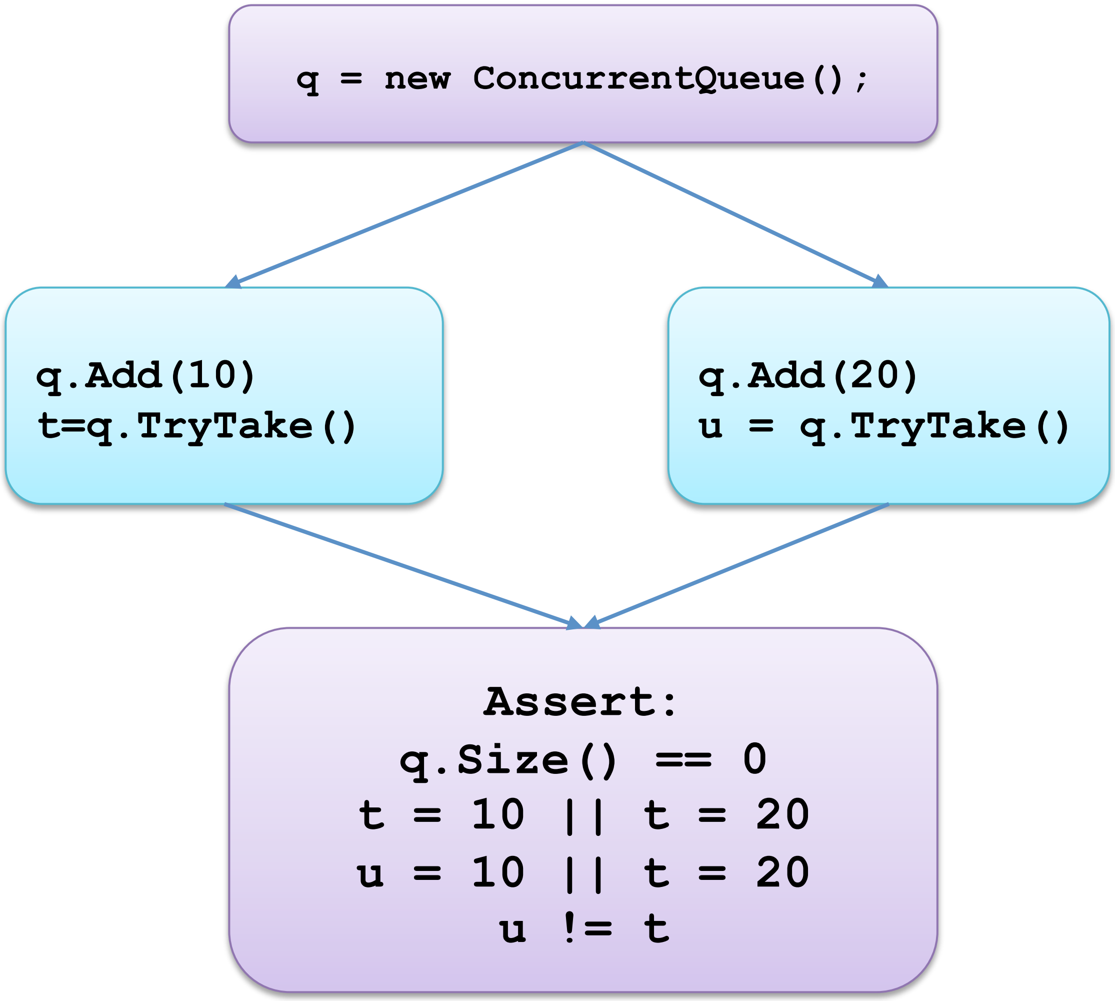
```
q = new ConcurrentQueue
();
```

```
q.Add(10)
t=q.TryTake()
```

```
q.Add(20)
u = q.TryTake()
```

```
Assert ( ? )
```

Let’s Write a Test



```
q = new ConcurrentQueue();
```

```
q.Add(10)
t=q.TryTake()
```

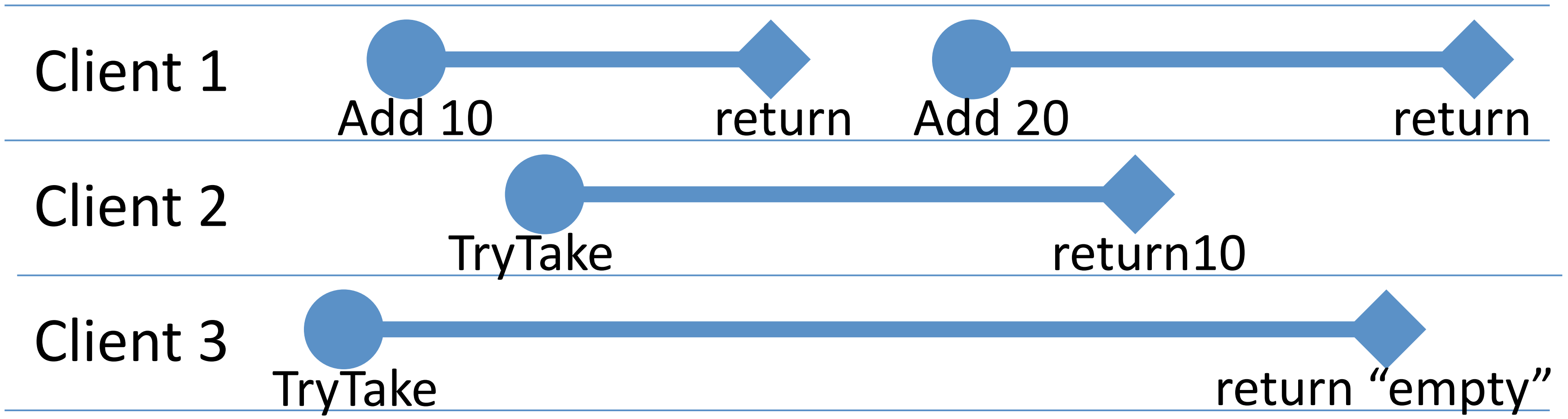
```
q.Add(20)
u = q.TryTake()
```

```
Assert:
q.Size() == 0
t = 10 || t = 20
u = 10 || t = 20
u != t
```

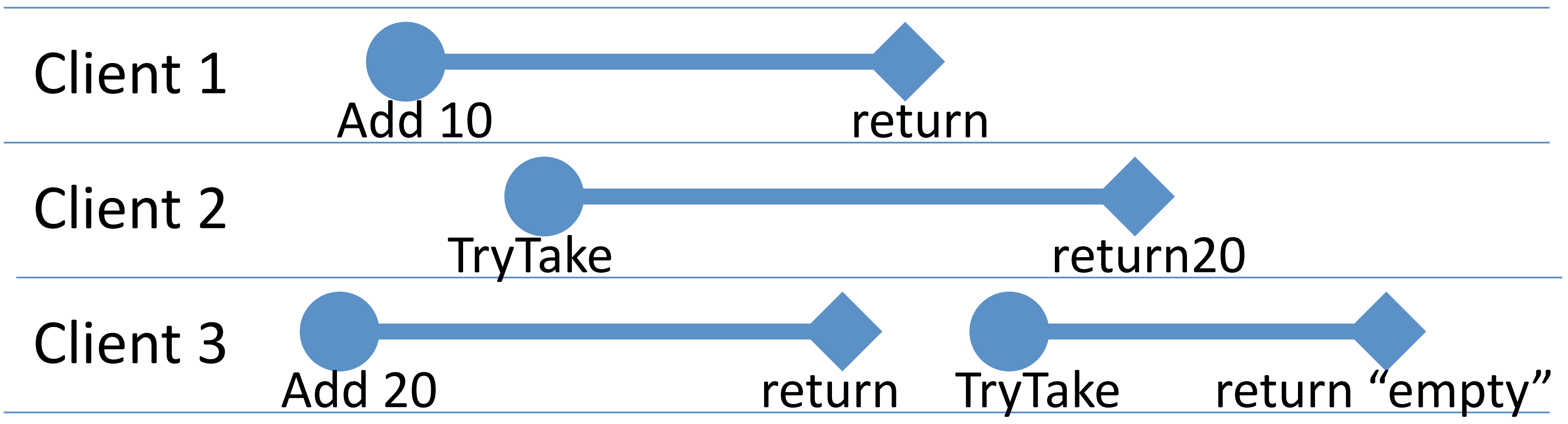
Linearizability

- The correctness notion closest to “thread safety”
- A concurrent component behaves ***as if*** only one thread can enter the component at a time

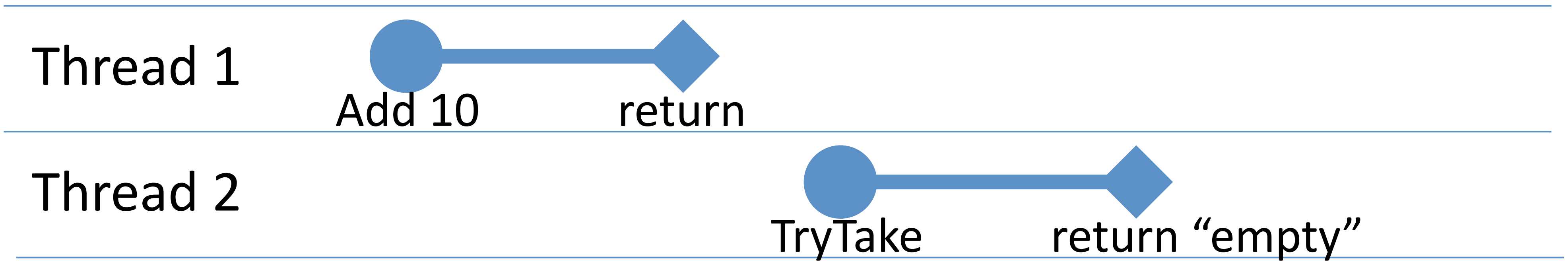
“Expected” Behavior?



"Expected" Behavior?



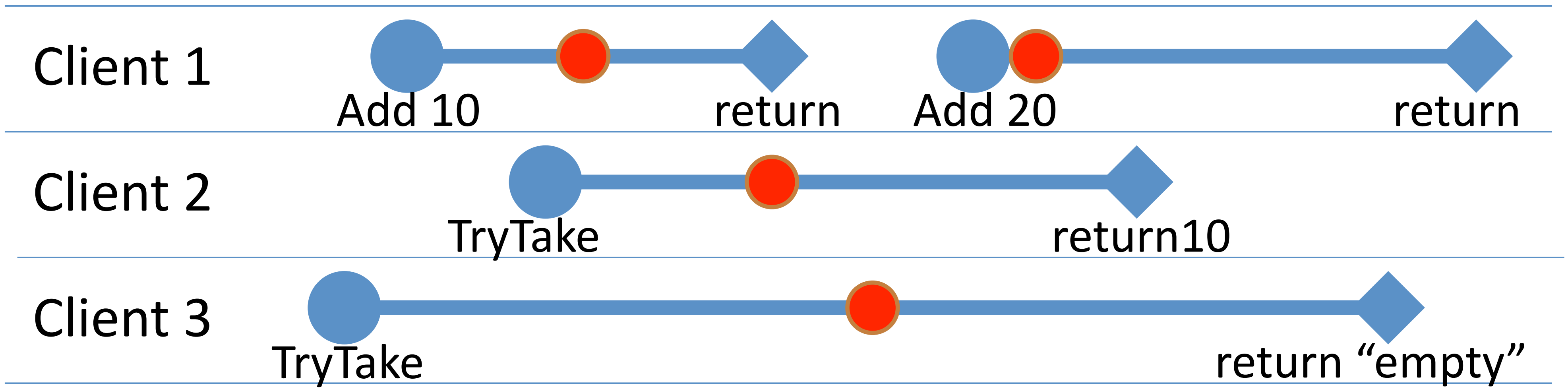
Expected Behavior?



Linearizability

- Component is *linearizable* if all operations
  - Appear to take effect atomically at a single temporal point
  - And that point is between the call and the return

- "As if the requests went to the queue one at a time"



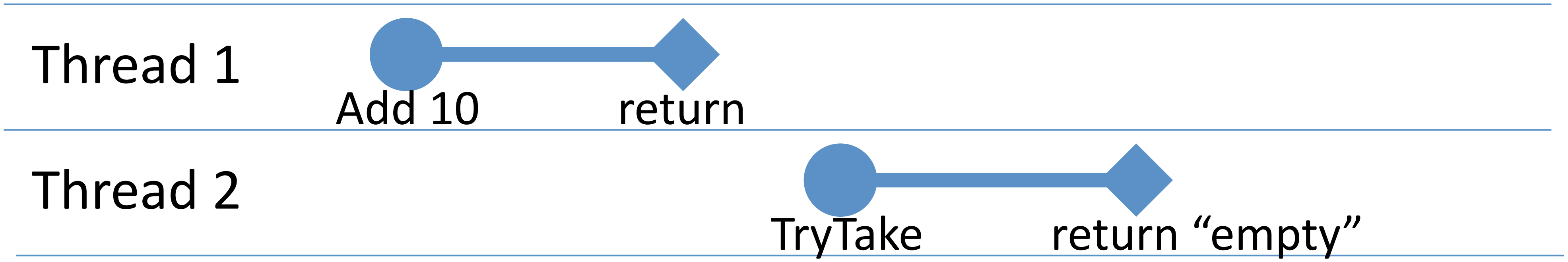
Linearizability vs Seriazliability?

- Serializability
  - All operations (transactions) appear to take effect atomically at a single temporal point

Linearizability vs Seriazliability?

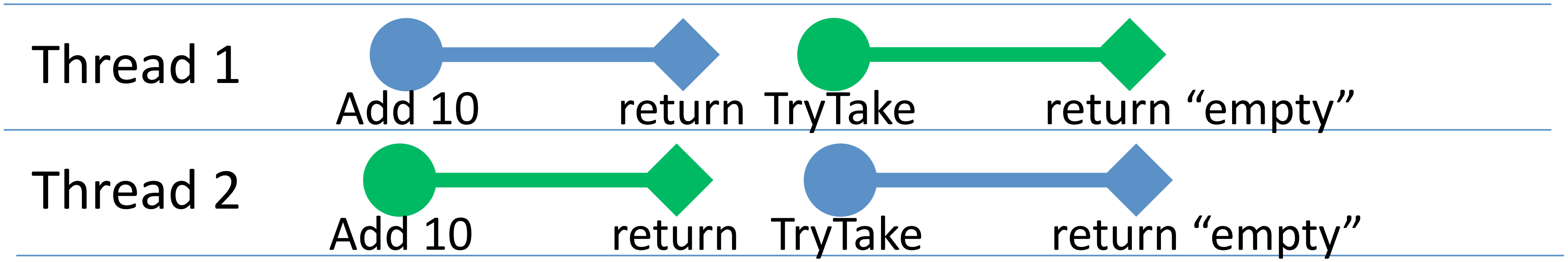
- Serializability
  - All operations (transactions) appear to take effect atomically at a single temporal point
- Linearizability
  - All operations to take effect atomically at a single temporal point
  - That point is between the call and return

Serializable behavior that is not Linearizable



- Linearizability assumes that there is a global observer that can observe that Thread 1 finished before Thread 2 started

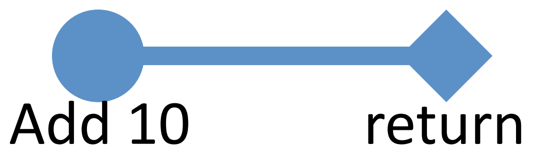
Serializability does not compose



- The behavior of the blue queue and green queue are individually serializable
- But, together, the behavior is not serializable

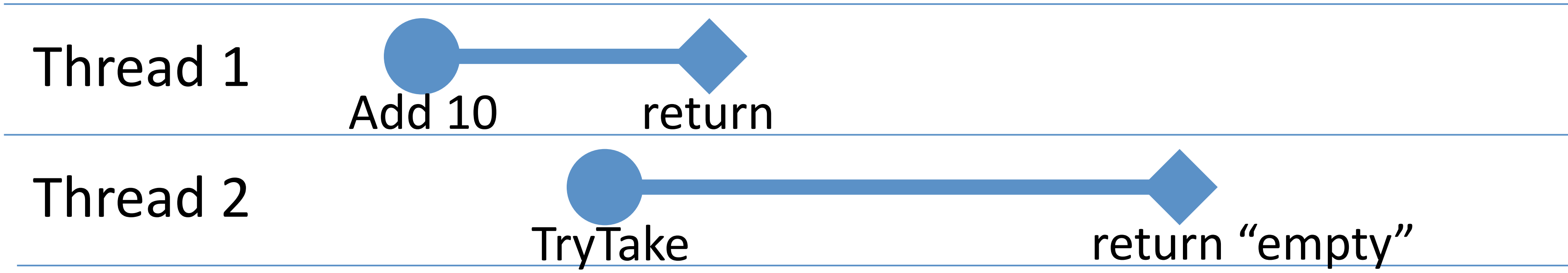
Formalizing Linearizability

- Define the set of observables for each operation
  - Call operation: value of all the arugments
  - Return operation:
- An event:
  - Thread Id, Object Id, Call/Return, Operation, Observables



A Concurrent History

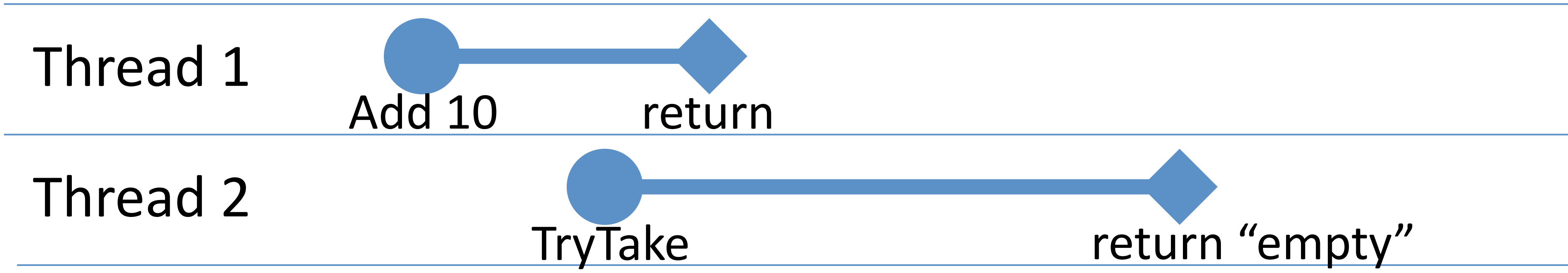
- Sequence of Events
  - <T1, q, Call, Add, 10>
  - <T2, q, Call, TryTake, void>
  - <T1, q, Ret, Add, void>
  - <T2, q, Ret, TryTake, “empty”>



A Concurrent History

- Sequence of Events
  - <T1, q, Call, Add, 10>
  - <T2, q, Call, TryTake, void>
  - <T1, q, Ret, Add, void>
  - <T2, q, Ret, TryTake, “empty”>

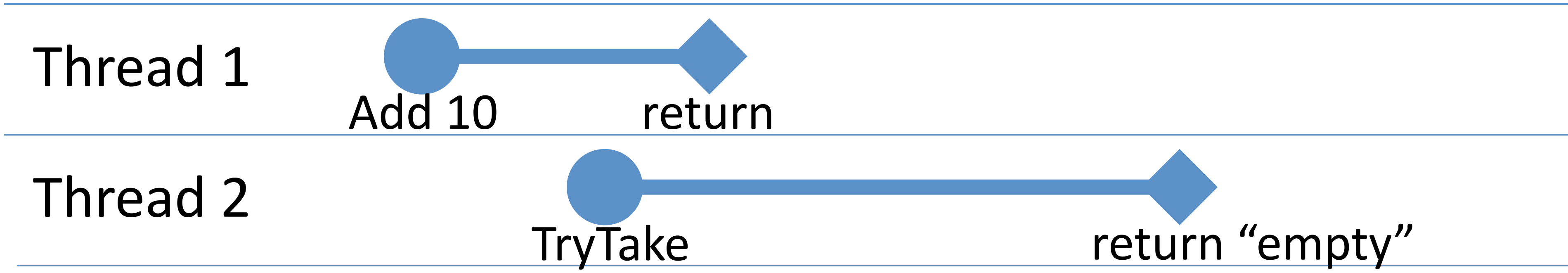
We will only focus on single object histories



A Concurrent History

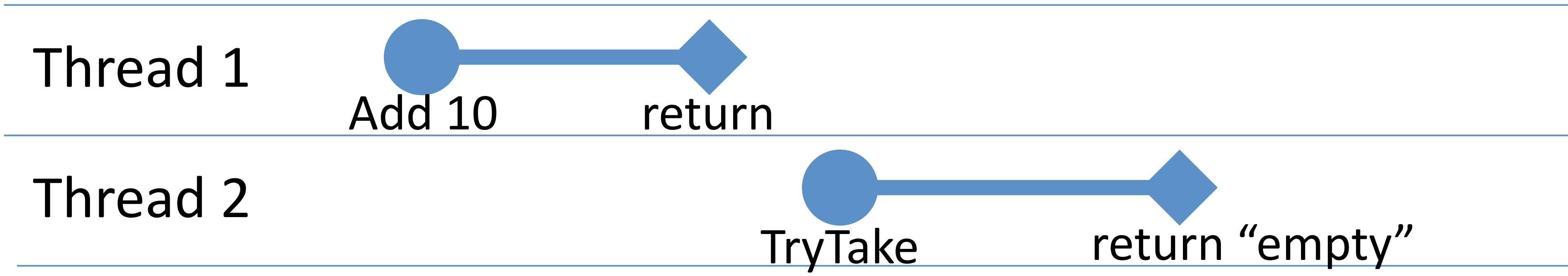
- Sequence of Events
  - <T1, q, Call, Add, 10>
  - <T2, q, Call, TryTake, void>
  - <T1, q, Ret, Add, void>
  - <T2, q, Ret, TryTake, “empty”>

Also, we will only focus on complete histories – every call has a return



A Serial History

- A concurrent history where every call is followed by its matching return
  - <T1, q, Call, Add, 10>
  - <T1, q, Ret, Add, void>
  - <T2, q, Call, TryTake, void>
  - <T2, q, Ret, TryTake, “empty”>

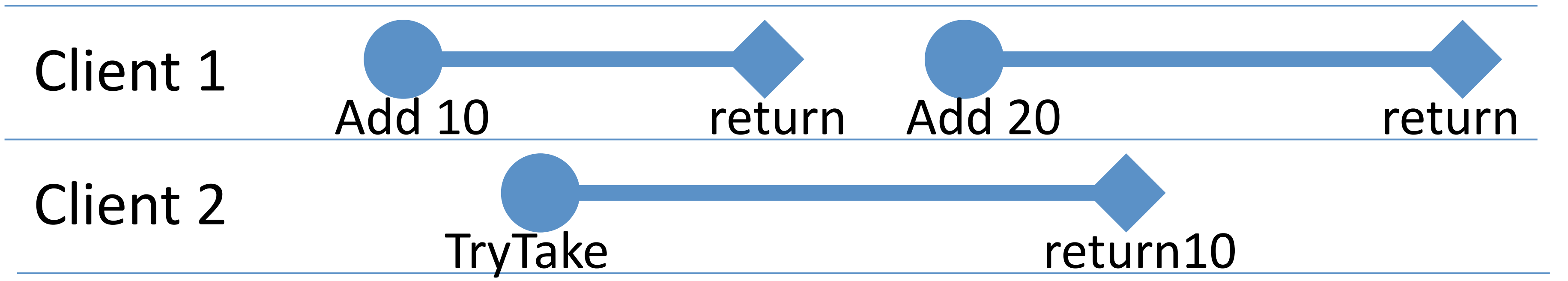


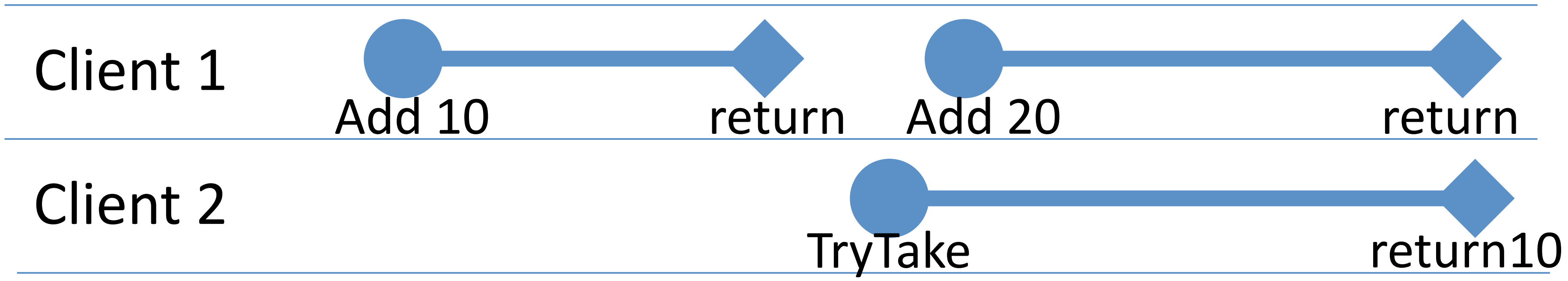
Sequential Specification of an Object

- The set of all serial histories define the sequential behavior of an object
- Assume we have a mechanism to enumerate this set and store the set in a database

Equivalent Histories

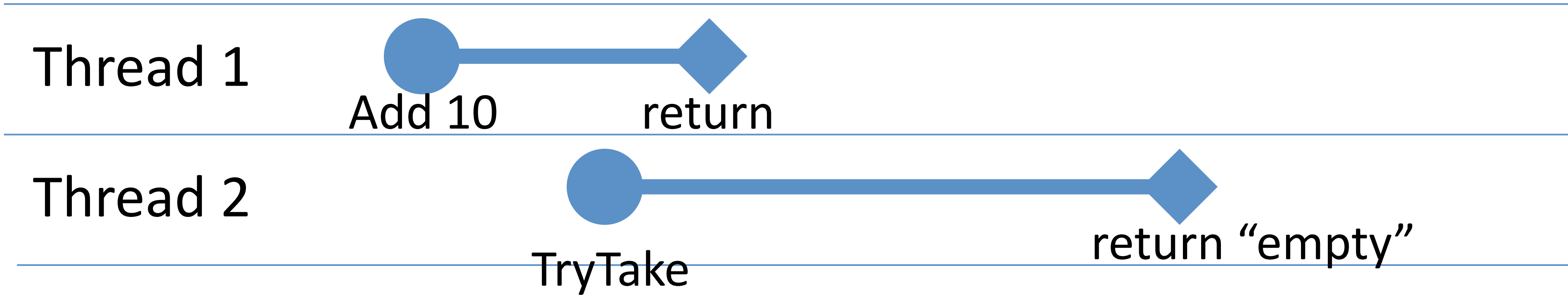
- Two concurrent histories are equivalent if
  - Each thread performs operations in the same order
  - And sees the same observations





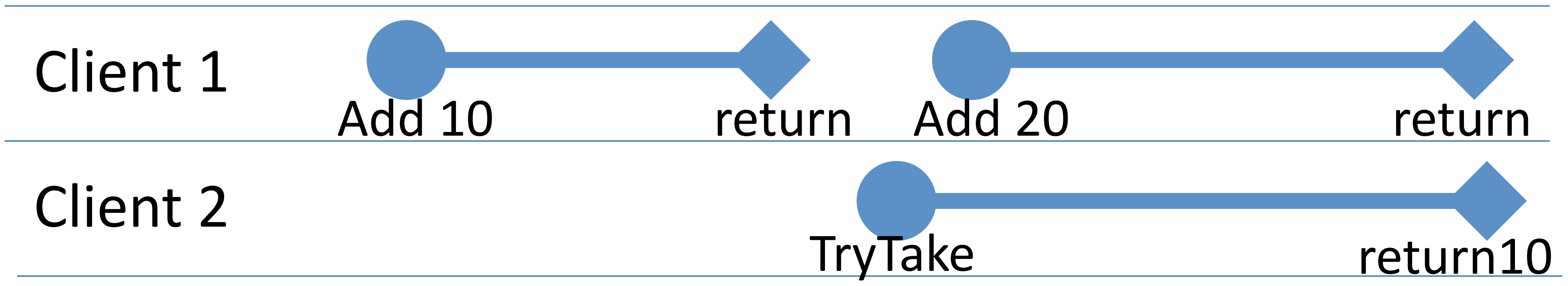
Concurrent Operations in a History

- Two operations p and q are concurrent in a history if their duration overlap
  - ! (p.ret < q.call | | q.ret < p.call)



Concurrent Operations in a History

- Two operations p and q are concurrent in a history if their duration overlap
  - ! (p.ret < q.call || q.ret < p.call)
- Non-Concurrent operations define a “performed-before” order



Linearizability

- A concurrent history is linearizable if it is equivalent to a (serial) history in the sequential specification,

- Such that all operations that are "performed before" in the concurrent history are also "performed before" in the serial history