The Producer-Consumer Collections: Queue, Stack and Bag

Simon Robinson

@TechieSimon | TechieSimon.com

Module 4 Overview

- How `ConcurrentQueue<T>` differs from the generic `Queue<T>`
- How `ConcurrentStack<T>` and `ConcurrentBag<T>` compare to `ConcurrentQueue<T>`
- Producer-consumer scenarios
- `IProducerConsumerCollection<T>`

Course Overview

Concurrent dictionary	Producer-consumer	Best practices
3. Concurrent Dictionary Demo	5. Producer-Consumer and BlockingCollection Demo	
2. Introducing Concurrent Dictionary	4. Queues, Stacks and Bags	6. Some best practices
1. Introducing the Concurrent Collections		

Concurrent Collections – The Full List…

General-purpose

`ConcurrentDictionary<TKey, TValue>`

Partitioners

`Partitioner<T>`

`OrderablePartitioner<T>`

`Partitioner`

`EnumerablePartitionerOptions`

Producer-consumer

`ConcurrentQueue<T>`

`ConcurrentStack<T>`

`ConcurrentBag<T>`

`BlockingCollection<T>`

`IProducerConsumerCollection<T>`

Concurrent Collections – The Full List…

General-purpose

ConcurrentDictionary<TKey, TValue>

Let's compare ConcurrentQueue<T> with Queue<T>!

Partitioner<T>

OrderablePartitioner<T>

Partitioner

EnumerablePartitionerOptions

Producer-consumer

ConcurrentQueue<T>

ConcurrentStack<T>

ConcurrentBag<T>

BlockingCollection<T>

IProducerConsumerCollection<T>

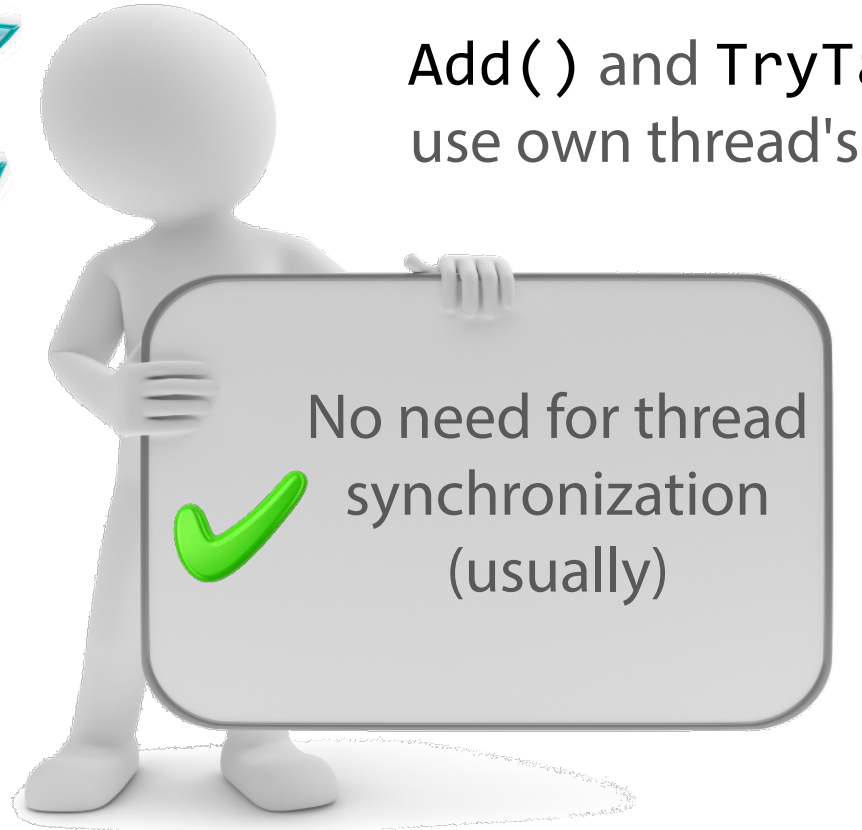
CODE DEMO

This slide must not appear in the recorded course

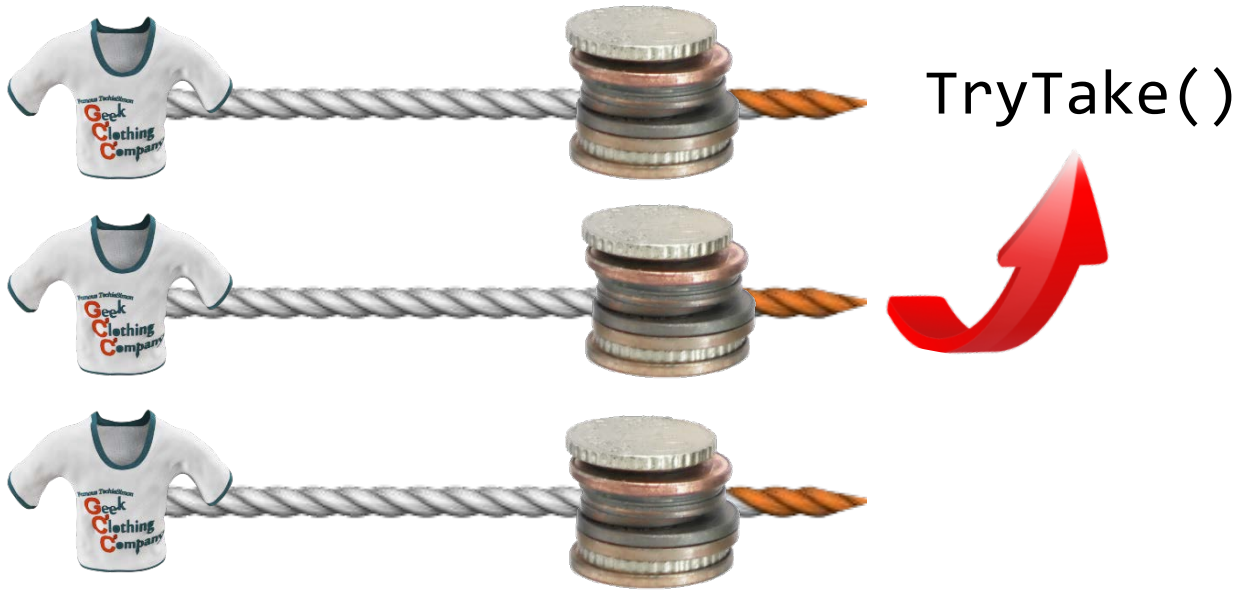
How ConcurrentBag Works

Separate collection of items
for each thread

`Add()` and `TryTake()`
use own thread's queue



How ConcurrentBag Works



If thread has no items, it'll try to **steal** an item from another thread

ConcurrentBag Performance

Can be very fast…

…but only if same threads add and take items to minimize stealing

Producer-Consumer Collections

Why are these producer-consumer collections?

Producer-consumer

- `ConcurrentQueue<T>`

- `ConcurrentStack<T>`

- `ConcurrentBag<T>`

- `BlockingCollection<T>`
- `IProducerConsumerCollection<T>`

Queues etc. vs. Dictionaries

General-purpose

`ConcurrentDictionary<TKey, TValue>`

Easy direct access to any element

No direct access to elements

- Hence not suitable as general-purpose collections

Producer-consumer

`ConcurrentQueue<T>`

`ConcurrentStack<T>`

`ConcurrentBag<T>`

`BlockingCollection<T>`

`IProducerConsumerCollection<T>`

What Are Queues etc. Good For?

Tasks being produced

Tasks being consumed

Producer-Consumer

Requests for pages

Threads processing requests

Producer-consumer scenario:

- Something produces items
- Something else consumes them

`ConcurrentQueue<T>` would be great here!

Producer-Consumer

Requests for pages

Threads processing requests

Producer-consumer scenario:

Most common reason for using


`ConcurrentQueue<T>`


`ConcurrentStack<T>`


`ConcurrentBag<T>`

IProducerConsumerCollection<T>

ConcurrentQueue<T>

ConcurrentStack<T>

ConcurrentBag<T>

Almost identical APIs, hence…

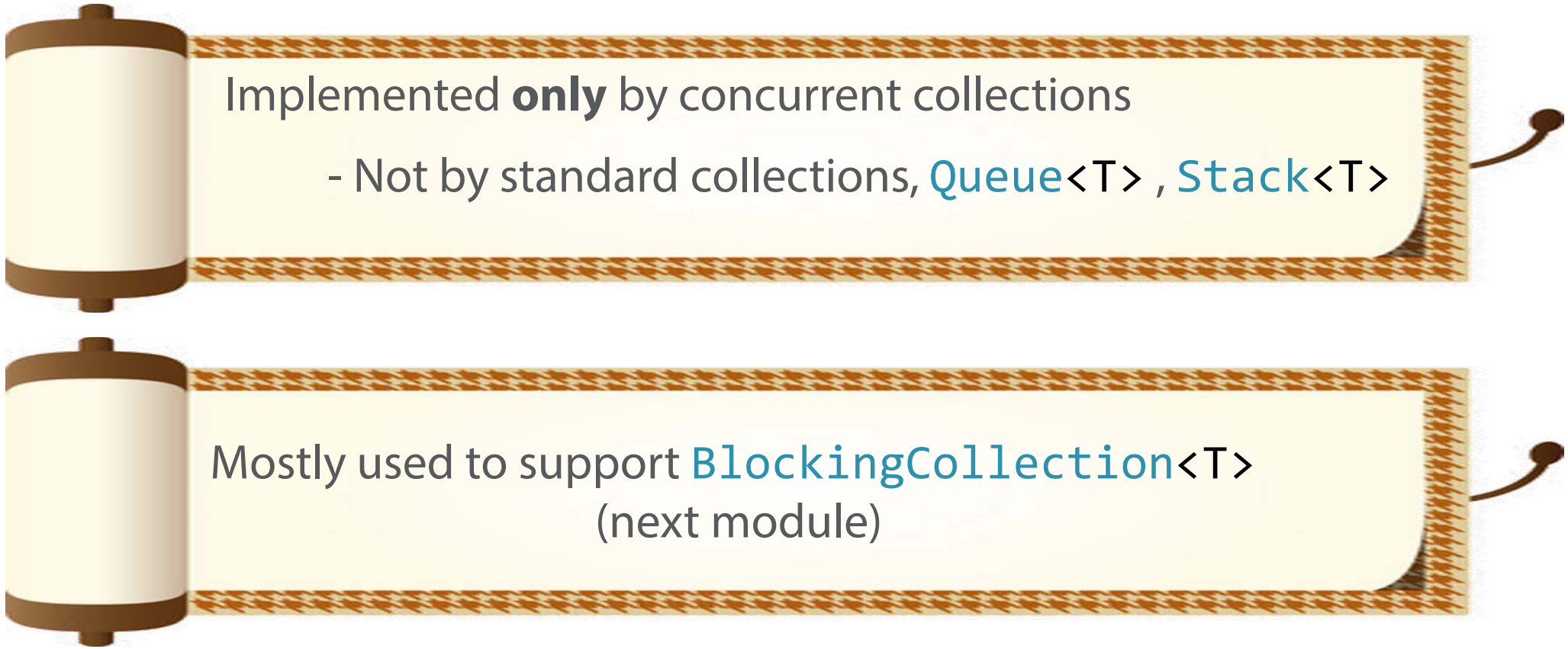
```
IProducerConsumerCollection<T> : IEnumerable<T>, ICollection

        bool TryAdd( T item )
        bool TryTake( out T item )
        int Count { get; }
```

More flexible than Add()

CODE DEMO

This slide must not appear in the recorded course

IProducerConsumerCollection<T>



Module 4 Summary

- `ConcurrentQueue<T>` similar to `Queue<T>` except for dequeuing
- Queue stack and bag APIs same except for terminology
  - `IProducerConsumerCollection<T>` captures this
- Producer-consumer scenarios, eg. tasks being generated