Some Concurrent Collections Best Practices

Simon Robinson

@TechieSimon | TechieSimon.com

Module 6 Overview

- Access concurrent collections sparingly
- Some methods and properties are slow
- Enumerators
- When should you use concurrent collections?

Performance

Concurrent collections
are slower than the corresponding
standard generic collections

CODE DEMO

This slide must not appear in the recorded course

Under the Hood

General-purpose

```
ConcurrentDictionary<TKey, TValue>
```

Producer-consumer

```
ConcurrentQueue<T>
ConcurrentStack<T>
ConcurrentBag<T>
BlockingCollection<T>
IProducerConsumerCollection<T>
```

Under the Hood

General-purpose

```
ConcurrentDictionary<TKey, TValue>
```

Producer-consumer

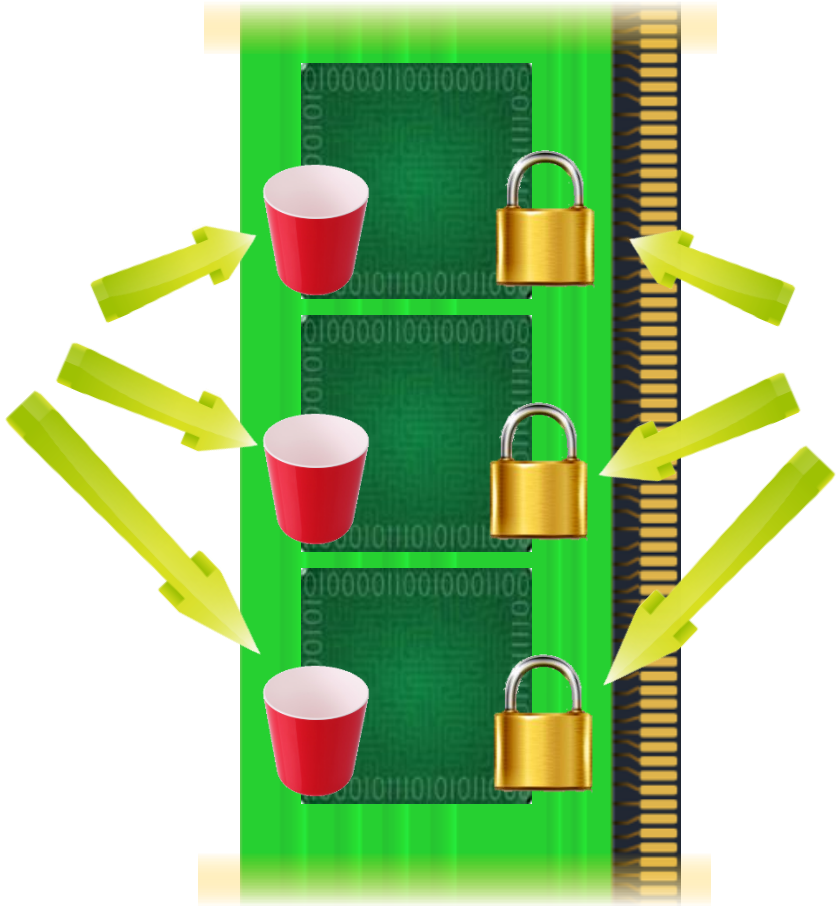
```
ConcurrentQueue<T>
ConcurrentStack<T>
ConcurrentBag<T>
BlockingCollection<T>
IProducerConsumerCollection<T>
```

Under the Hood - Concurrent Dictionary

Hash table

Buckets

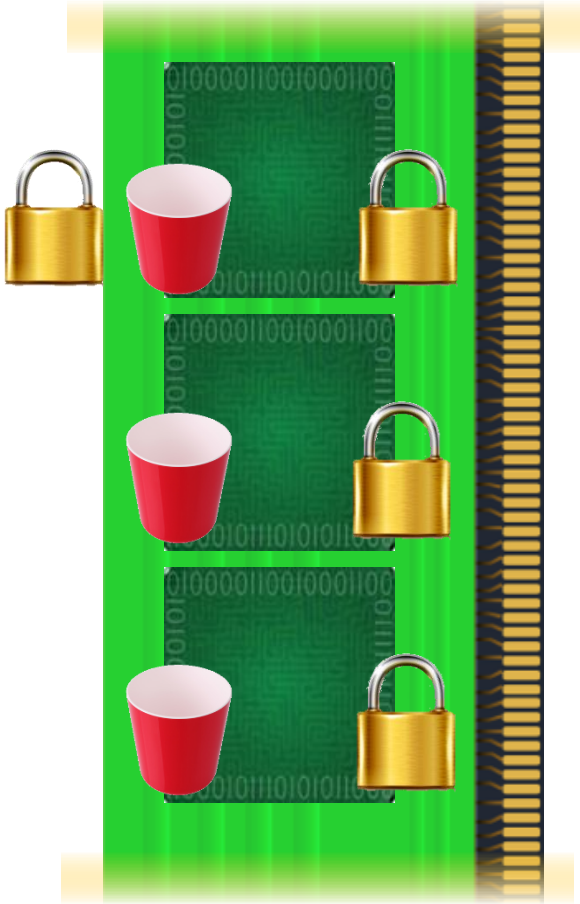
(Portion of dictionary data in each bucket)



Separate lock for each data portion

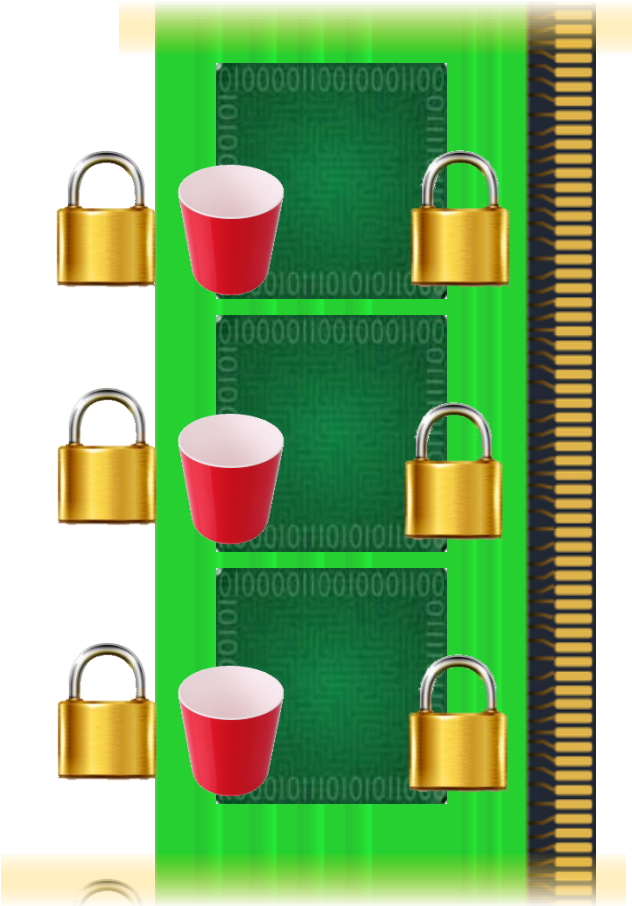
Under the Hood - Concurrent Dictionary

Hash table

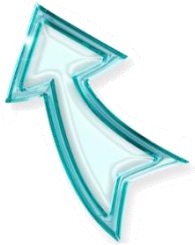


Under the Hood - Concurrent Dictionary

Hash table



`IsEmpty`



This is important
if you query the
aggregate state
of the dictionary

This locks everything!

Methods that Lock Extensively

ConcurrentDictionary
methods

- IsEmpty
- Count
- Clear()
- ToArray()
- CopyTo()
- Values
- Keys

ConcurrentBag
methods

- IsEmpty
- Count
- ToArray()
- CopyTo()
- GetEnumerator()

All these methods query the state of the collection

Except
Clear()
- This sets the state of the collection

Good Practice...

Avoid querying the state of concurrent collections too often

Why?

Answer can be out of date as soon as you have it

For all collections

Hurts performance

Dictionary and bag

Enumerators

Enumerating concurrent collections

- do it the same way as for ordinary collections

```
foreach (var item in stock)
{
    // etc.
```

```
var items = from item in stock
            where // etc.
```

CODE DEMO

This slide must not appear in the recorded course

Enumerating Collections that Change

General-purpose

```
ConcurrentDictionary<TKey, TValue>
```

Enumeration results not guaranteed

Producer-consumer

```
ConcurrentQueue<T>
ConcurrentStack<T>
ConcurrentBag<T>
```

Snapshot of collection taken
Subsequent changes ignored

Concurrent Dictionary Snapshot

These will all give you a snapshot:

```
// ConcurrentDictionary<string, int > stock;
foreach (var item in stock.ToArray())
{
```

```
// ConcurrentDictionary<string, int > stock;
foreach (var item in stock.Keys)
{
```

```
// ConcurrentDictionary<string, int > stock;
foreach (var item in stock.Values)
{
```

When Do You Need Concurrent Collections?

Multiple threads?

When Do You Need Concurrent Collections?

Multiple readers

Multiple writers

When Do You Need Concurrent Collections?

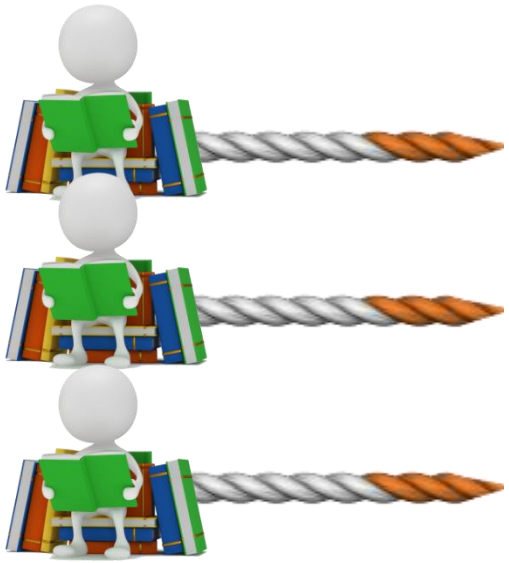
Multiple readers

What if
reading causes fields to be modified
under the hood?

Eg. lazy-initialized properties

When Do You Need Concurrent Collections?

Multiple readers



Standard Collections

Often work but risky

Concurrent Collections

Thread-safe

Immutable Collections

Thread-safe

Immutable Collections

```
using System.Collections.Immutable;
```

BuyAndSell.sln - Manage NuGet Packages

Installed packages

Online

All

nuget.org

Microsoft and .NET

Search Results

Updates

Stable Only | Sort by: Relevance | immutable collections

Rackspace.Collections.Immutable
Compatibility library for using the Microsoft Immutable Collections library in projects targeting .NET 3.5 and newer.

System.Collections.Immutable
Provides immutable collections that allow CPU and memory efficient mutation via new references.
Install

System.Collections.Immutable for .Net 4.0
Retargeted for .Net 4.0. Unofficial, unsupported build

Microsoft Immutable Collections
Provides immutable collections that allow CPU and memory efficient mutation via new references.

Immutable.Fody
Converts types to Immutable.

Iesi Collections

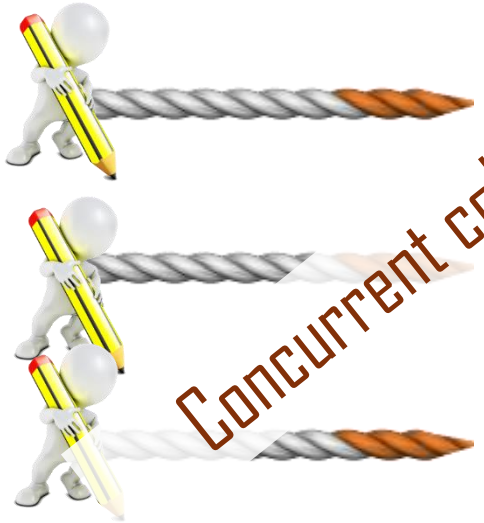
Created by: Microsoft
Id: System.Collections.Immutable
Version: 1.1.36
Last Published: 11/05/2015
Downloads: 118312
License
View License
Project Information
Report Abuse
Description:
This package provides collections that are thread safe and guaranteed to never change their contents, also known as immutable collections. Like strings, any methods that perform modifications will not change the existing instance but instead return a new instance. For efficiency reasons, the implementation uses a sharing mechanism to ensure that newly created instances share as much

When Do You Need Concurrent Collections?

Multiple readers

Concurrent or immutable collections (standard might work)

Multiple writers



Concurrent collections

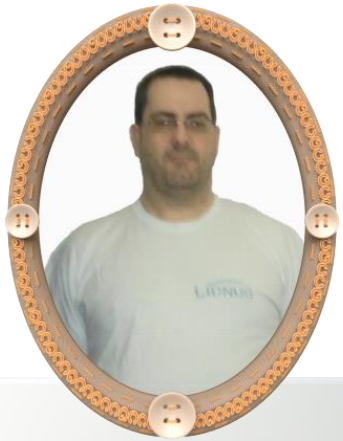
Mixed

Concurrent collections

Module 6 Summary

- Use concurrent collections sparingly from multiple threads to make your apps fast
- Avoid asking concurrent collections for aggregate state
  - Info can immediately be out of date, and it can hurt performance
- Can enumerate concurrent collections as they are changing
- Multiple threads require concurrent collections - unless all threads are readers

Thanks to the Models...



Peter Shaw

Ramdevi Maharjan

Juliette Reinders Folmer

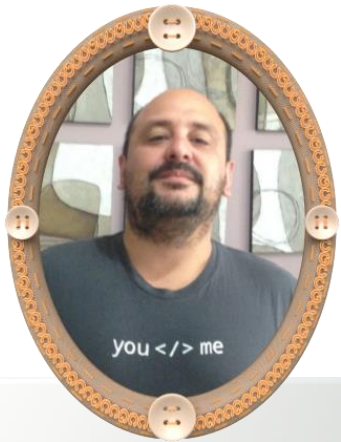
Mark Seemann

Sahil Malik

Koffi Sessi

Lisa Larson-Kelley

Nick Marx



Xavier Morera

Course Summary

- ConcurrentDictionary is the main general-purpose thread-safe collection
- Use BlockingCollection with a concurrent queue, stack or bag for producer-consumer scenarios
- Use one single method call for each operation on a concurrent collection
- Avoid code that needs to know the aggregate state of concurrent collections

C# Concurrent Collections

Simon Robinson

@TechieSimon | TechieSimon.com