BASTA!®
.NET, VISUAL STUDIO & MORE

Oliver Sturm, DevExpress

Der Vorteil der Statik – Immutable Data

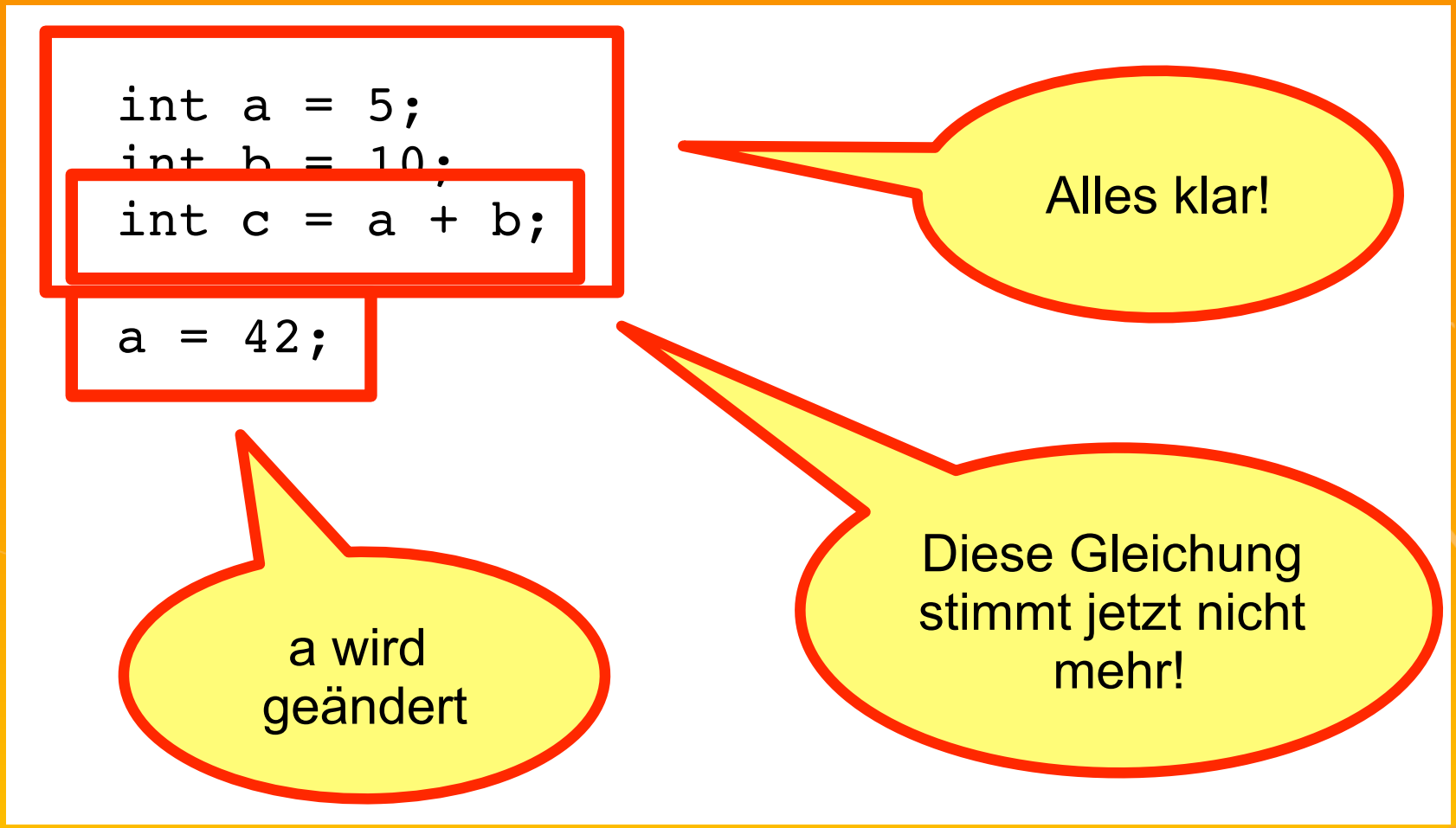
Wer ist eigentlich dieser Oliver Sturm?

- Ich bin
  - Interessiert an Programmiersprachen, Datenbanken und vielen anderen Sachen
  - Microsoft MVP für C#
  - Autor der C# Corner im dot.net magazin
- Ich
  - Arbeite für Developer Express als Technical Evangelist und Lead Program Manager
  - Blogge unter http://www.sturmnet.org/blog
- Sie sollten
  - Mir Emails schicken an oliver@sturmnet.org

Agenda

- Grundlagen für und wider Veränderbarkeit
- Einfache und komplexe Daten unveränderbar gestalten
- Code innerhalb von Funktionen

(Falsche?) Annahmen



Veränderung ist gut…

- Das ist ein Paradigma der imperativen Programmierung
- Statusänderung wird hier als zentrale Idee der Programmierung gesehen
- Für manche Aufgaben ist ein Status -gesteuerter Algorithmus die beste Lösung, besonders in imperativen Sprachen

Veränderung ist schlecht…

- Veränderung schafft Probleme
- Stichwort Debugging:
  - “Warum ist a auf einmal 5?”
- Veränderung ist ein Feind der Skalierbarkeit
  - Probleme beim gemeinsamen Zugriff
  - Locking & Co. werden oft als Workaround eingesetzt

Idee: Nichts mehr ändern

- Gute Idee. Aber:
- Skalierung macht Veränderung notwendig
- Also: Der Programmierer muss die Wahl haben, sich für oder gegen Veränderbarkeit zu entscheiden, mit möglichst klaren Grenzen

Einfache Schritte

- Variablen als unveränderbar betrachten
- Felder als readonly deklarieren
- Anstelle von Änderungen an Klasseninstanzen werden neue Instanzen erzeugt

Änderbare Klasse

```
class Person {
  private string name;
  public string Name {
    get { return name; }
    set { name = value; }
  }
}
```

Nicht änderbare Klasse

```
class ImmutablePerson {
  private readonly string name;
  public string Name { get { return name; } }

  public ImmutablePerson(string name) {
    this.name = name;
  }

  public ImmutablePerson ChangeName(
    string newName) {
    return new ImmutablePerson(newName);
  }
}
```

Structs?

- Structs sollten unveränderbar implementiert werden
- Problem: Structs sind Wertetypen
  - Besondere Probleme, wenn Structs veränderbar sind
  - Oft nicht effizient, wenn Clones erzeugt werden

Demo

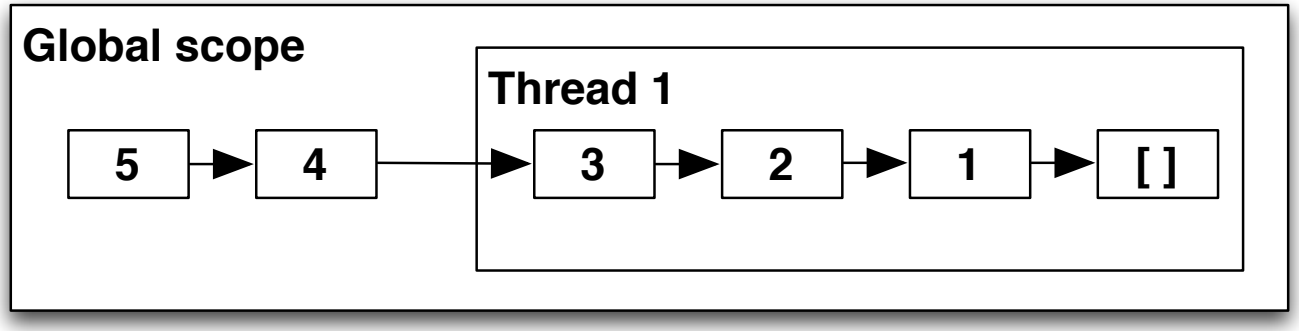
Einfache Typen

Isolierung/Sichtbarkeit

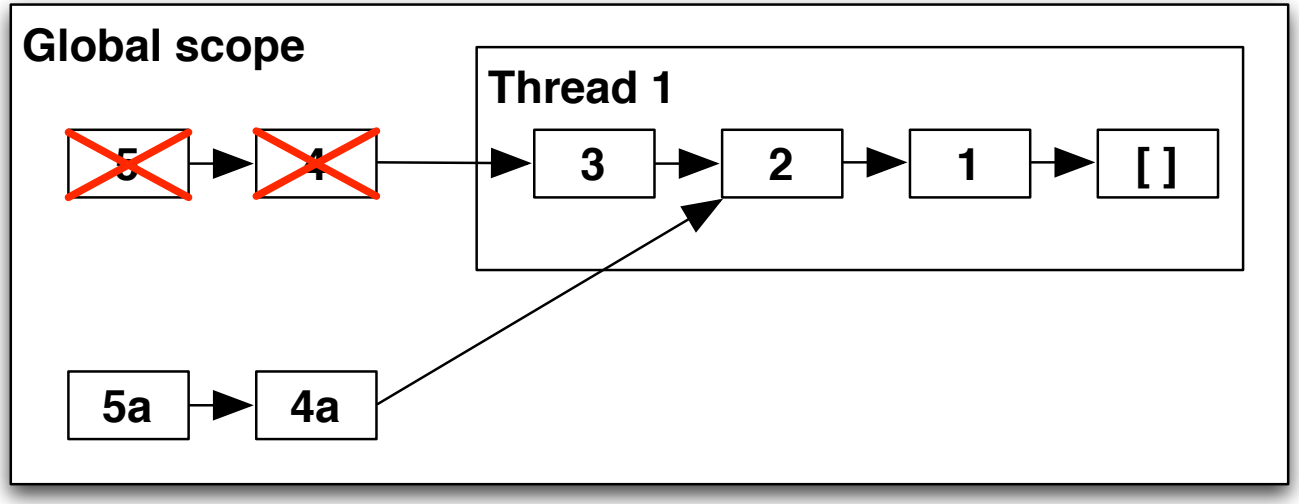
- Szenarien bei Skalierung
  1. Threads arbeiten für sich
  2. Jeder Thread arbeitet an einem Teil des Ganzen
  3. Threads müssen kommunizieren
- 2 und 3 sind oft algorithmisch austauschbar
- Unveränderbare Datentypen garantieren Stabilität von Daten für einen beliebigen Teilbereich

Listentypen

Adding elements



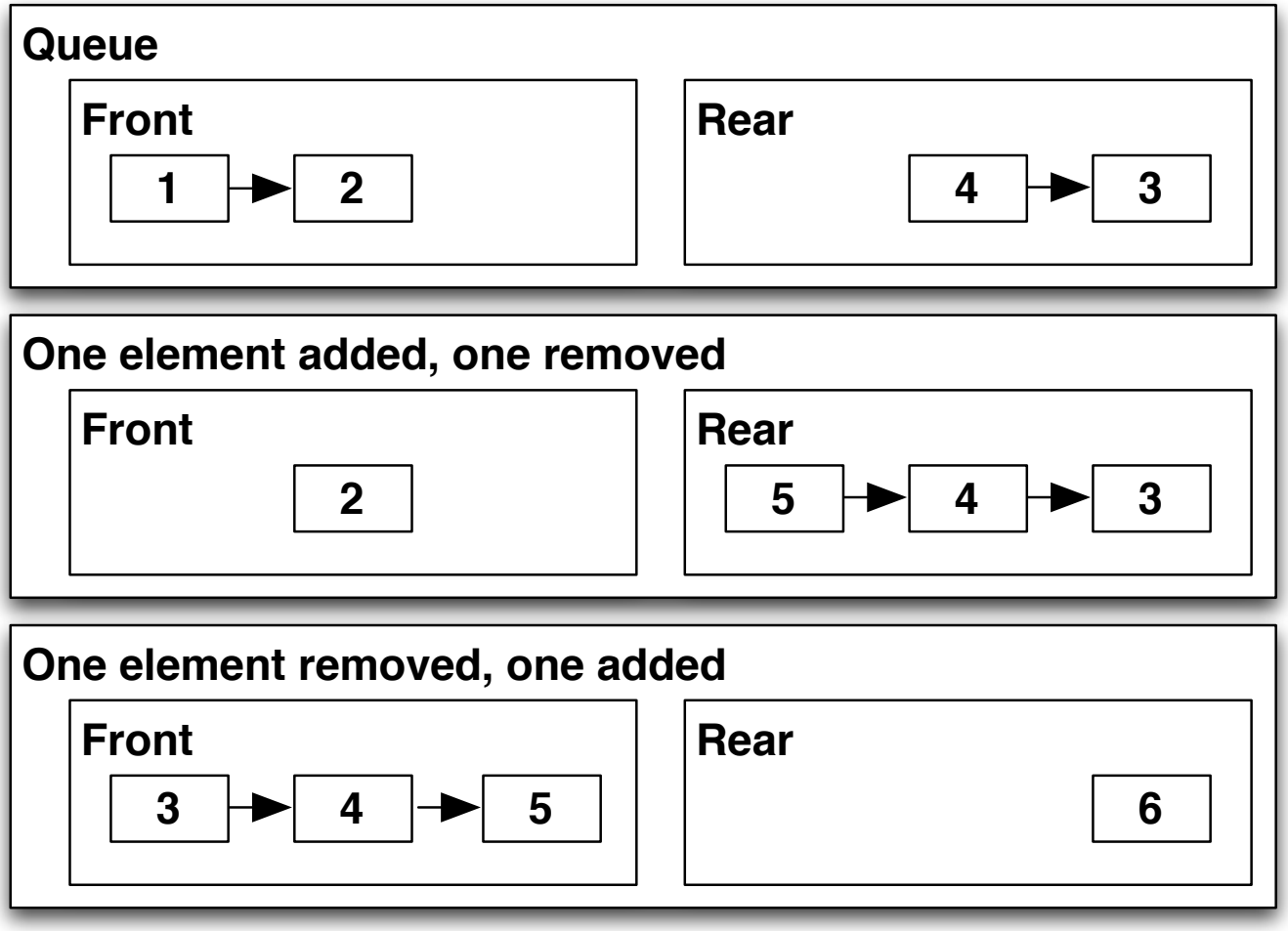
Removing element 3



Demo

Listentypen

(FIFO) Queue



Code in Funktionen

- Variablen wiederverwenden – keine gute Idee
- Iterationen in C# benötigen meistens Änderbarkeit
- Alternativen:
  - Rekursion – nur begrenzt verwendbar
  - Standard higher order functions verschieben das Problem weg aus unserem Code
  - Saubere Struktur

Demo

Code in Funktionen

Was wir nicht gesehen haben

- Unveränderbare Daten sind ein wichtiger Schritt zur Vermeidung von Nebeneffekten
- Allgemeine Softwarestabilität steigt zusammen mit der Einfachheit des Testens und Debuggens
- Versuchen Sie es mal selbst, es wird Ihnen gefallen!

Zusammenfassung

- Für die Skalierbarkeit ist die Koordination von veränderbaren und unveränderbaren Daten sehr wichtig
- Auch komplexe Datentypen können zu Isolationszwecken unveränderbar gestaltet werden
- Alles ist möglich in C#, Compilerunterstützung wäre aber nett...

Vielen Dank

Bitte kontaktieren Sie mich jederzeit zum Inhalt dieser Session

oliver@sturmnet.org

Mein Buch “Functional Programming in C#” wird irgendwann in 2009 bei Wrox erscheinen, und enthält viele weitere Informationen zum Thema

Algorithmen für List und Queue stammen aus Chris Okasaki’s Buch “Purely functional data structures”