Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások

---

11. előadás

Reaktív programozás

Dr. Cserép Máté
mcserep@inf.elte.hu
https://mcserep.web.elte.hu

Reaktív programozás

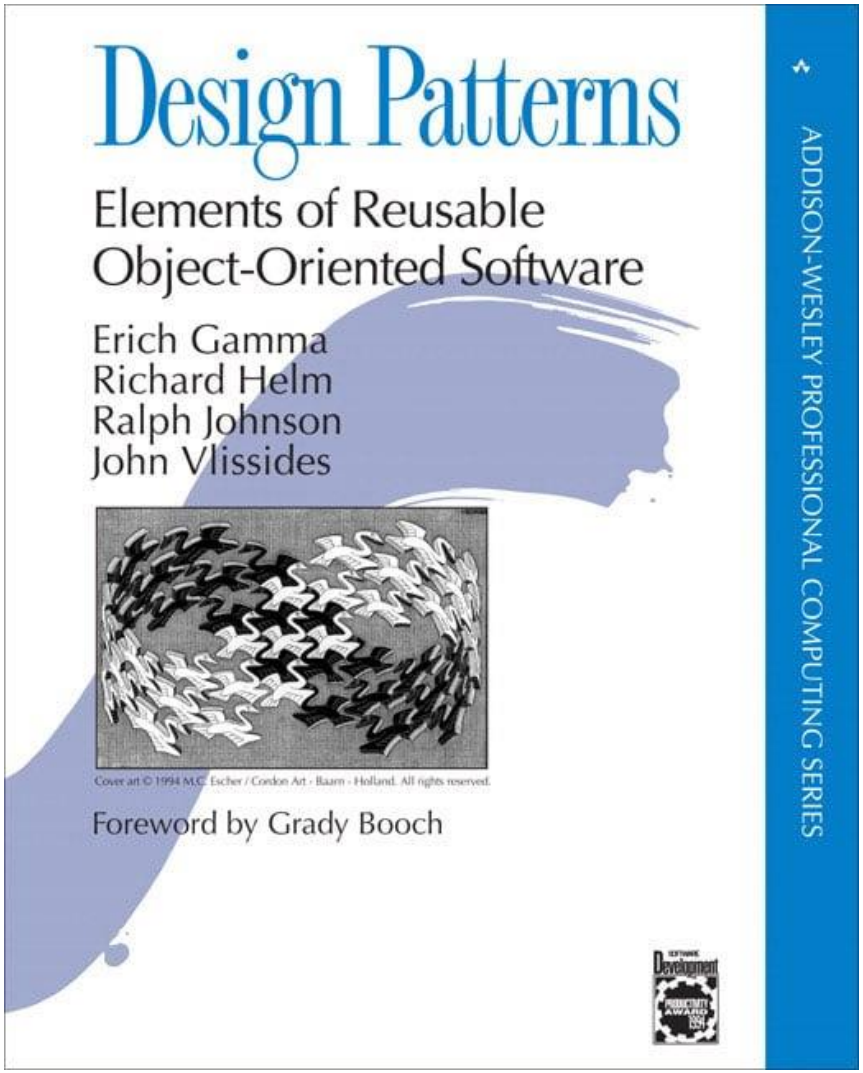
Reaktív programozás

- Az *Avalonia UI* támogatja a *reaktív programozás*t a *ReactiveUI* kerentrendszer használatával (https://www.reactiveui.net/)
  - Az pedig az *Rx.NET*-re épül (*Reactive Extensions for .NET*) https://reactivex.io/
- Mi az a reaktív programozás?
  - *"Reactive programming is a declarative programming paradigm that is based on the idea of asynchronous event processing and data streams."*
- Példa:

```
int a = 5; int b = 10;
int c = a + b;
Console.WriteLine($"c = {c}");
a = 20;
Console.WriteLine($"c = {c}");
```

Reaktív programozás

Tervezési minták



Kezdetek:

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, a.k.a. "Gang-of-Four" (GoF): *Design Patterns: Elements of Reusable Object-Oriented Software*, 1994

Martin Fowler: *Analysis Patterns – Reusable Object Models*, 1996

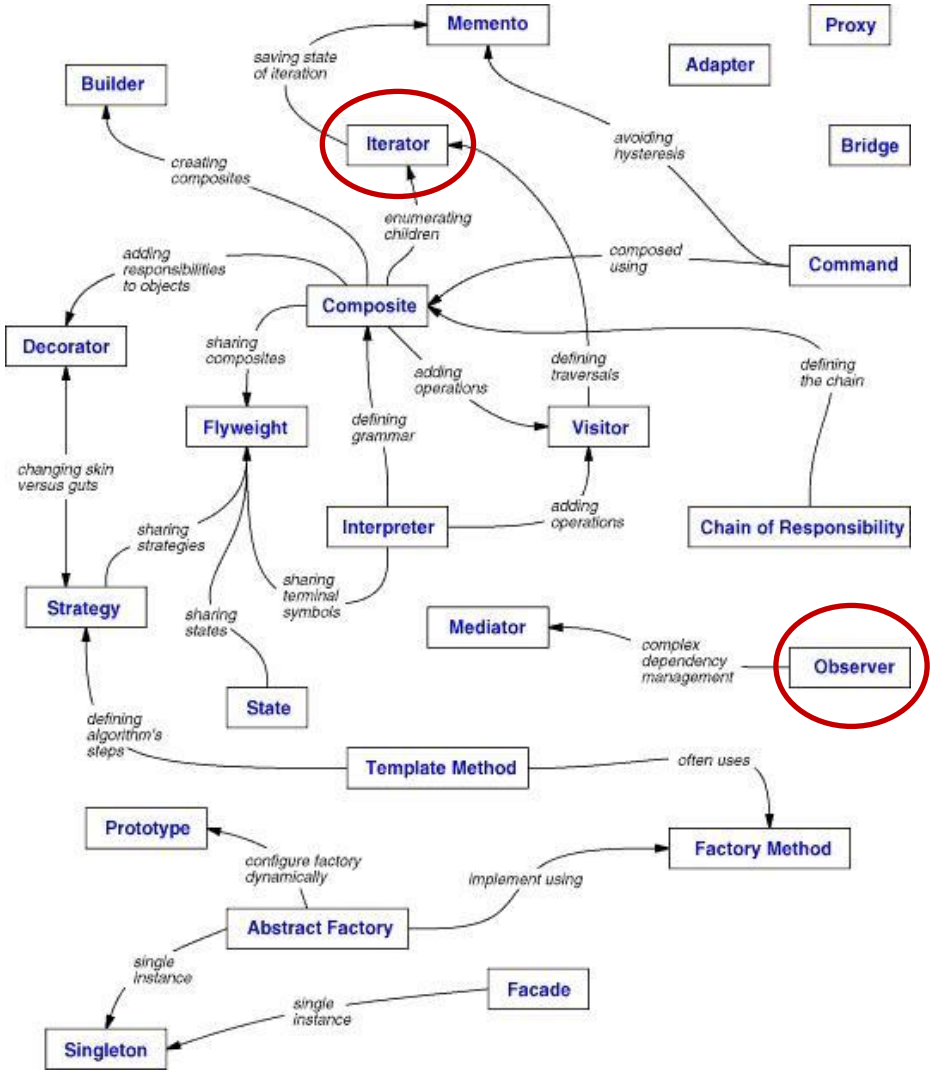
Reaktív programozás

Tervezési minták

- Mi az a tervezési minta?
  - GoF: „Descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context.”
  - MF: „A pattern is an idea that has been useful in one practical context and will probably be useful in others.”
  - MF: „Patterns are a starting point, not a destination.”

Reaktív programozás

Tervezési minták

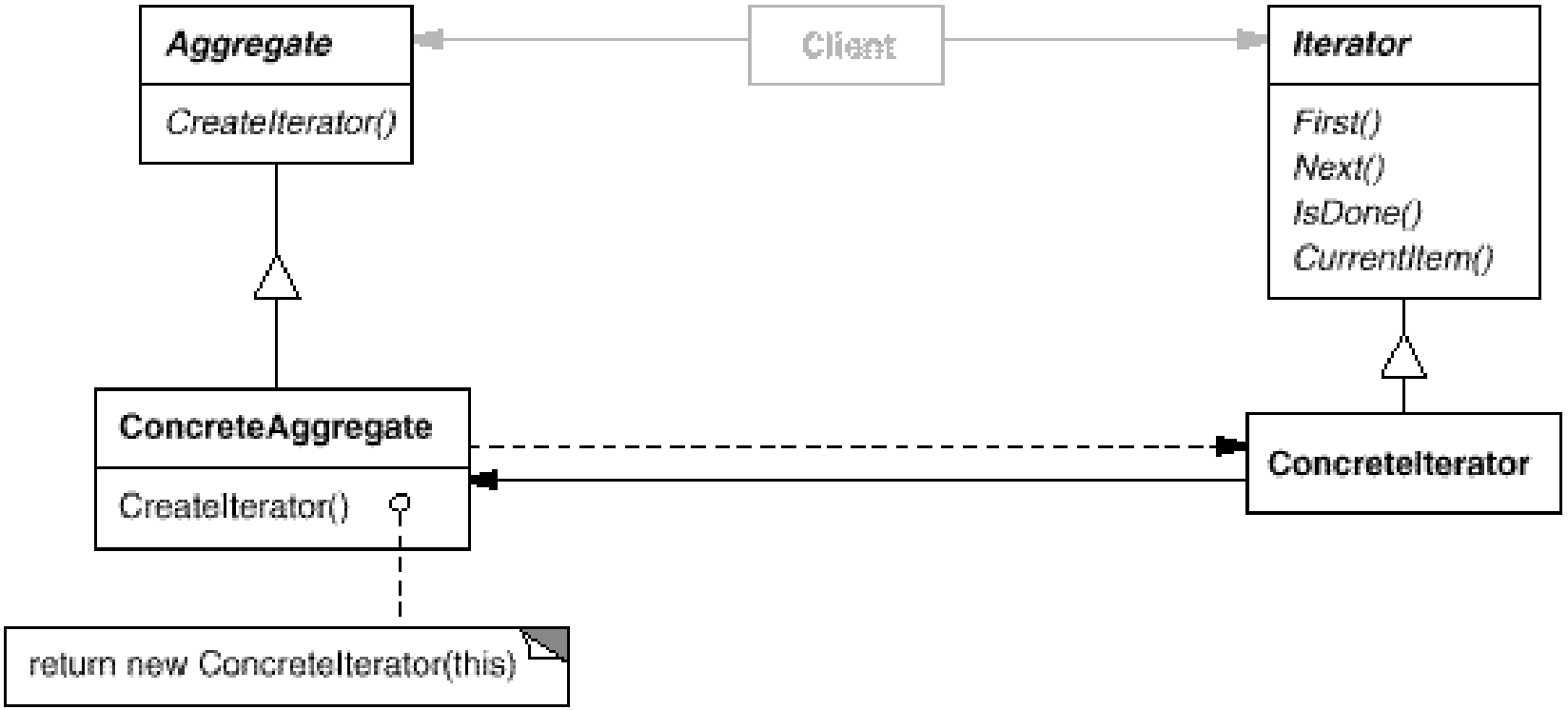


GoF tervminták és kapcsolataik

- Az *Iterator* és az *Observer* tevmintát alaposabban is megismerjük hamarosan.
- Vegyük észre, hogy az eredeti szerzők nem feltétlen gondoltak a két tervminta együttes használatára.

Reaktív programozás

Iterator tervezési minta



Reaktív programozás

Iterator tervezési minta

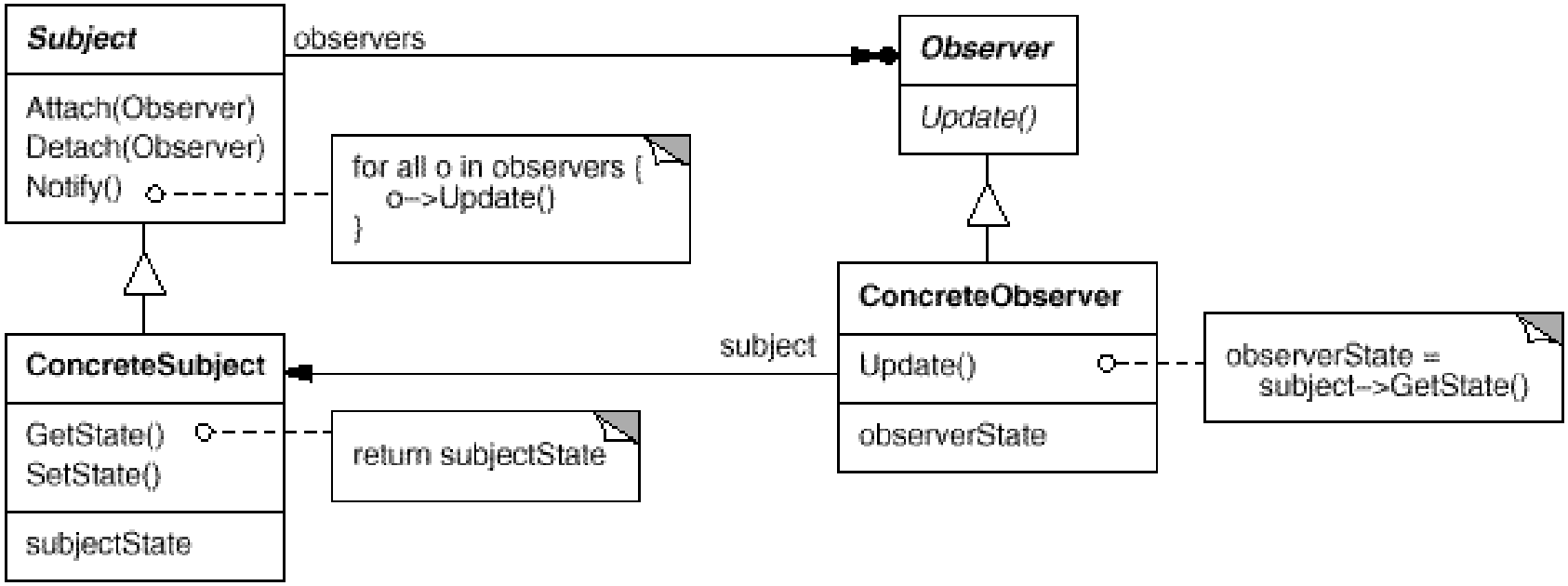
- Az *Iterator* tervminta C#-ban a .NET standard szintjén megjelenik az **`IEnumerable`** és **`IEnumerator`** típusok révén.
  - Ezek megvalósítása (egyszerűsítésekkel):

```
public interface IEnumerable<out T>
{
    IEnumerator<T> GetEnumerator();
}

public interface IEnumerator<out T>
{
    T Current { get; }
    bool MoveNext();
    void Reset();
}
```

Reaktív programozás

Observer tervezési minta



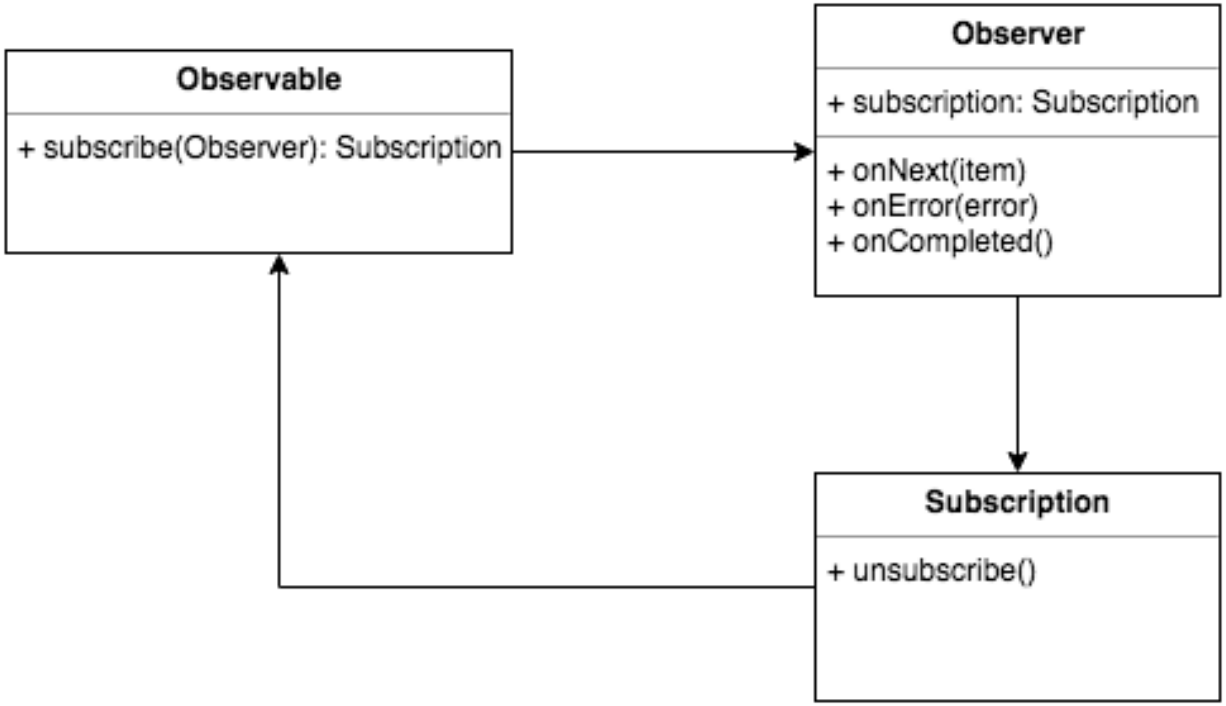
Felismerjük-e az *Observer* tervezési mintát eddigi C# ismereteinkben?

- A C# eseményei az eseménykezelői az *Observer* tervmintát valósítják meg.

Reaktív programozás

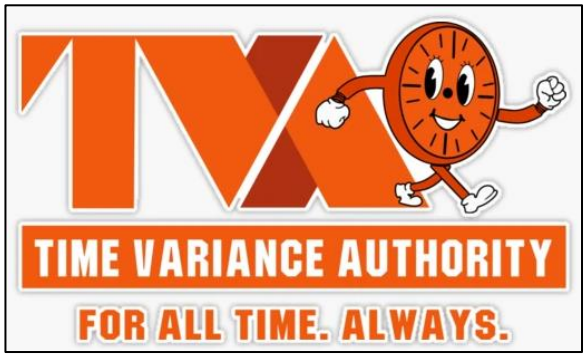
Megfigyelhető felsorolók

- Kombináljuk az *iterator* és az *observer* tervezési mintákat!
  - Mit kapunk eredményül?



Reaktív programozás

Reaktív programozás



- Reaktív programozás legfontosabb elemei:
  - Időben változó, megfigyelhető változók (*time variant variables*), amelyeket adatfolyamokként (*data streams*) is felfoghatunk
  - Aszinkron végrehajtás és ütemezők támogatása (*observerek* és *observable*-ök ütemzése: mit melyik szálon kell végrehajtani)
  - Operátorok
    - Szűrők: *filter, skip, take*
    - Kombinációs: *concat, merge, zip*
    - Transzformációs: *map, groupby*
    - stb.

Reaktív programozás

ReactiveX

- A reaktív programozáshoz használt egyik népszerű szoftverkönyvtár az eredetileg a Microsoft által kidolgozott, mára nyílt forráskódú *ReactiveX* (röviden: *Rx*) könyvtár.
  - számos programozási nyelvhez elérhető, C#/.NET-hez az *Rx.NET* (*Reactive Extensions for .NET*) könytárban
  - a **`System.Reactive`** NuGet csomaggal adhatjuk a projekteinkhez
- *Rx.NET*-ben a megfigyelhető felsorolók közös ős interfésze az **`IObservable<T>`** típus
  - ilyen megfigyelhető felsorolókat számos különböző módon létrehozhatunk az **`Observable`** osztály gyártó műveleteihez.

Reaktív programozás

ReactiveX

- a legelemibb módja az `Observable.Create<T>()` használata
- példa egy szöveges állomány soronkénti megfigyelhető felsorolására:

```
var myObservable = Observable.Create<string>(observer =>
{
  try {
    using var reader = new StreamReader("file.txt");
    string line;
    while ((line = reader.ReadLine()) != null)  {
      observer.OnNext(line);
      // Minden sor esetén új elemet jelzünk
    }
    observer.OnCompleted();
    // Jelezzük, hogy véget ért a felsorolás
  }
  catch (Exception ex) {
    observer.OnError(ex);
    // Jelezzük, hogy hibával ért véget a felsorolás
  }
  return () => { };
  // a felsoroló felszabadítását végző tevékenység
});
```

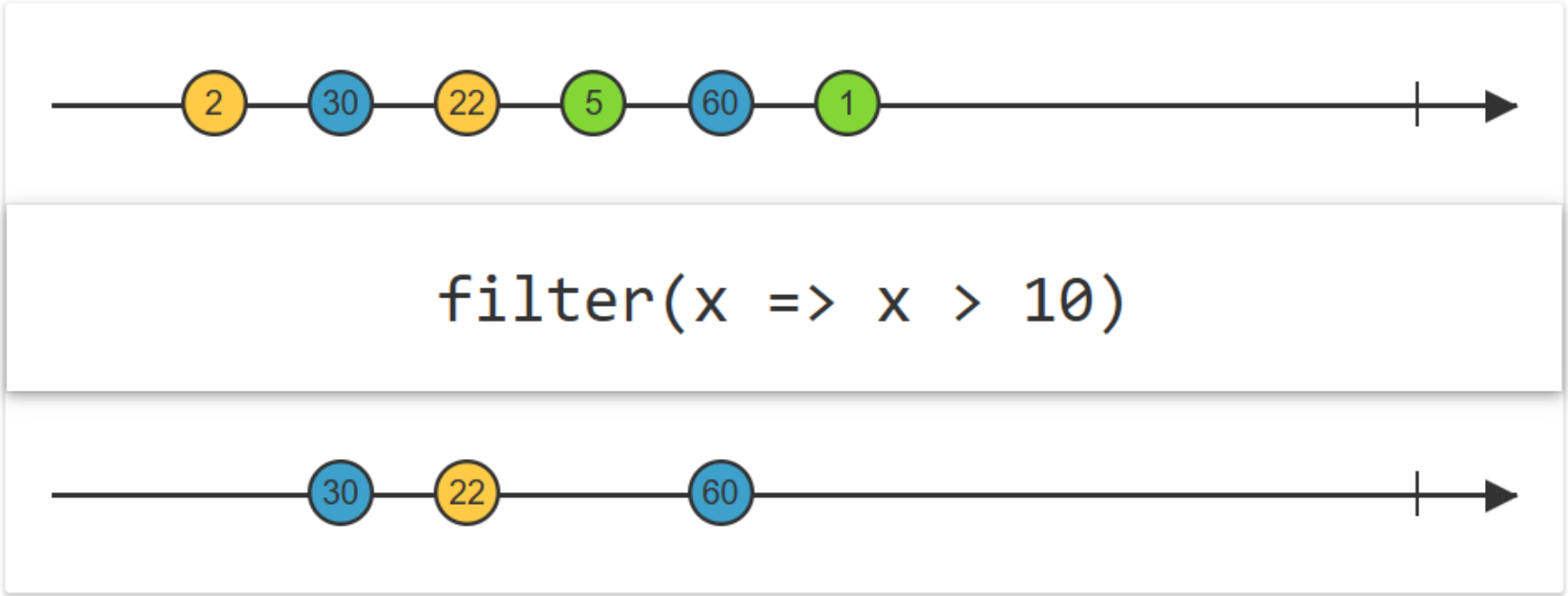
Reaktív programozás

ReactiveX

- számos segéd gyártó eljárás segíti munkánkat, hogy a leggyakoribb feladatokra könnyebben is definiálhassunk megfigyelhető felsorolókat, például:
  - **`Observable.Return():`** megadott érték egyszeri megfigyelhető felsorolása
  - **`Observable.Generate():`** megadott kiindulási állapotból, megállási feltétellel és iterációs szabállyal elemek felsorolása
  - **`Observable.Interval():`** elemek időzített felsorolása megadott időközönként
  - **`Observable.Start():`** számításigényes eljárás aszinkron végrehajtása és az eredmény megfigyelhető felsorolása
- a felsorolt elemeket operátorokkal szűrhetjük, transzformálhatjuk, csoportosíthatjuk

Reaktív programozás

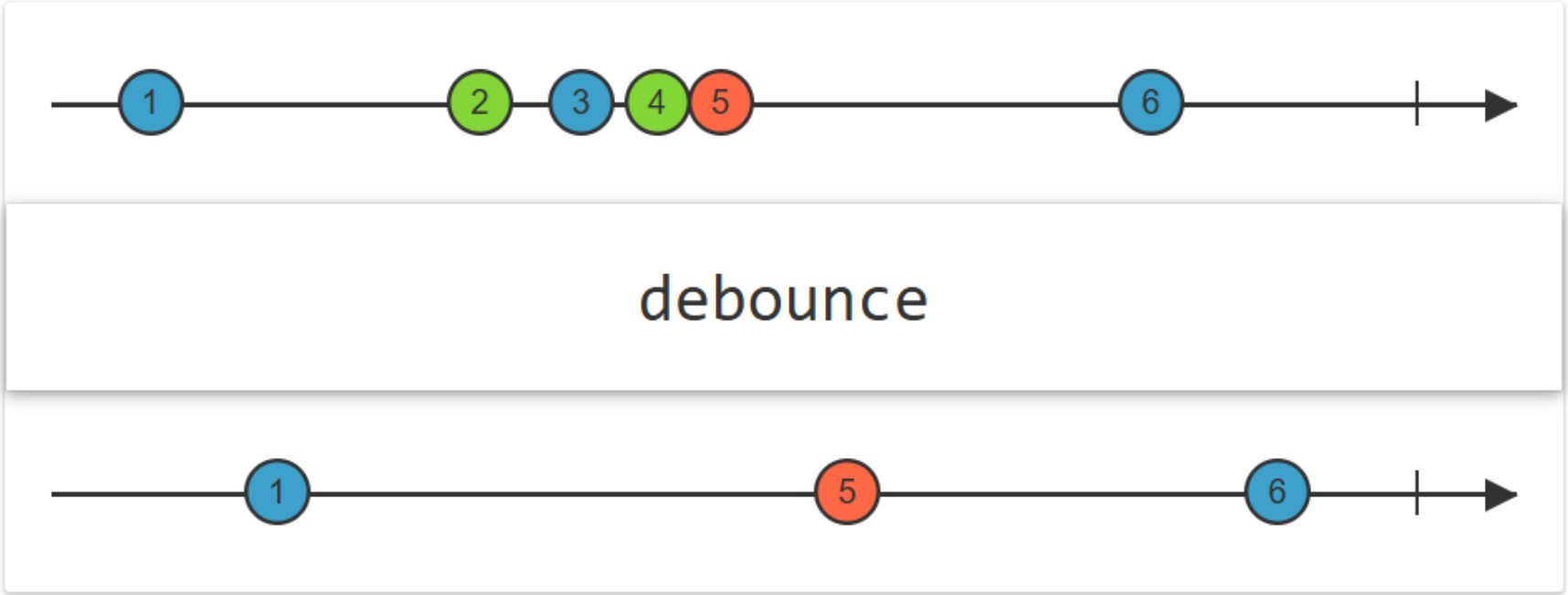
ReactiveX – Operátorok



C# (Rx.NET): **`Where()`**

Reaktív programozás

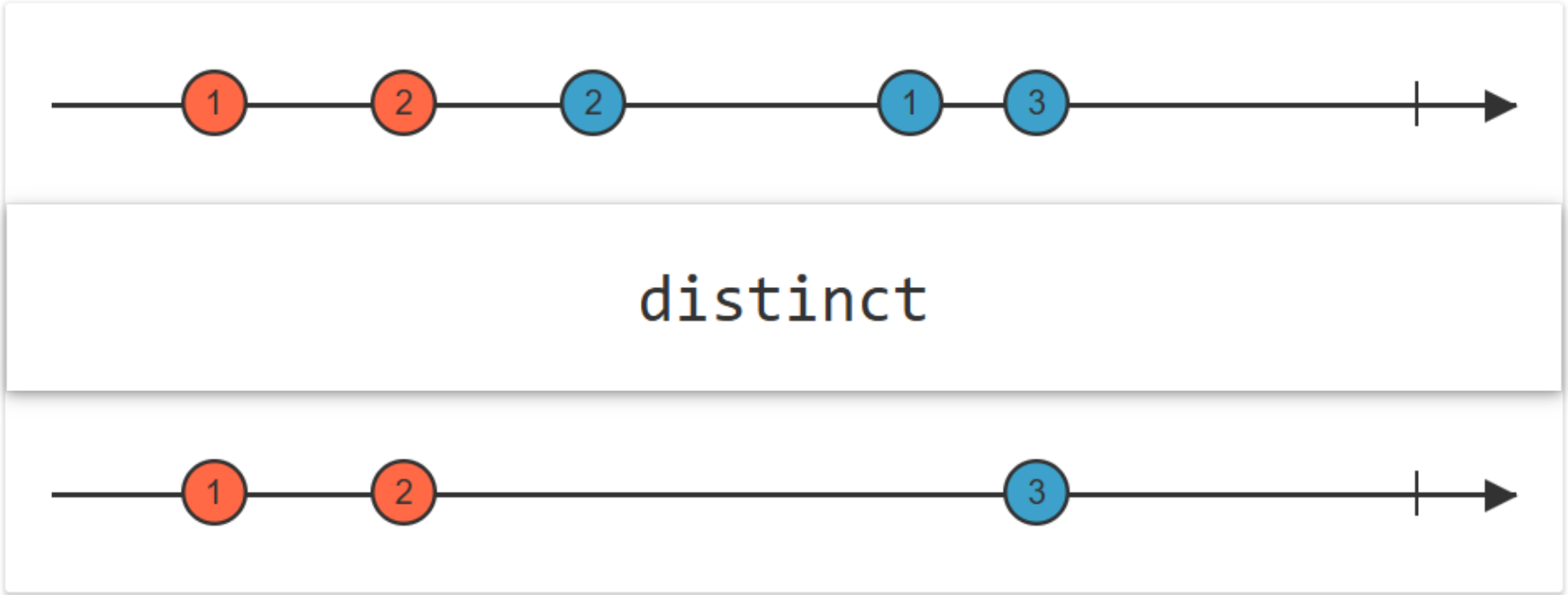
ReactiveX – Operátorok



C# (Rx.NET): **`Throttle()`**

Reaktív programozás

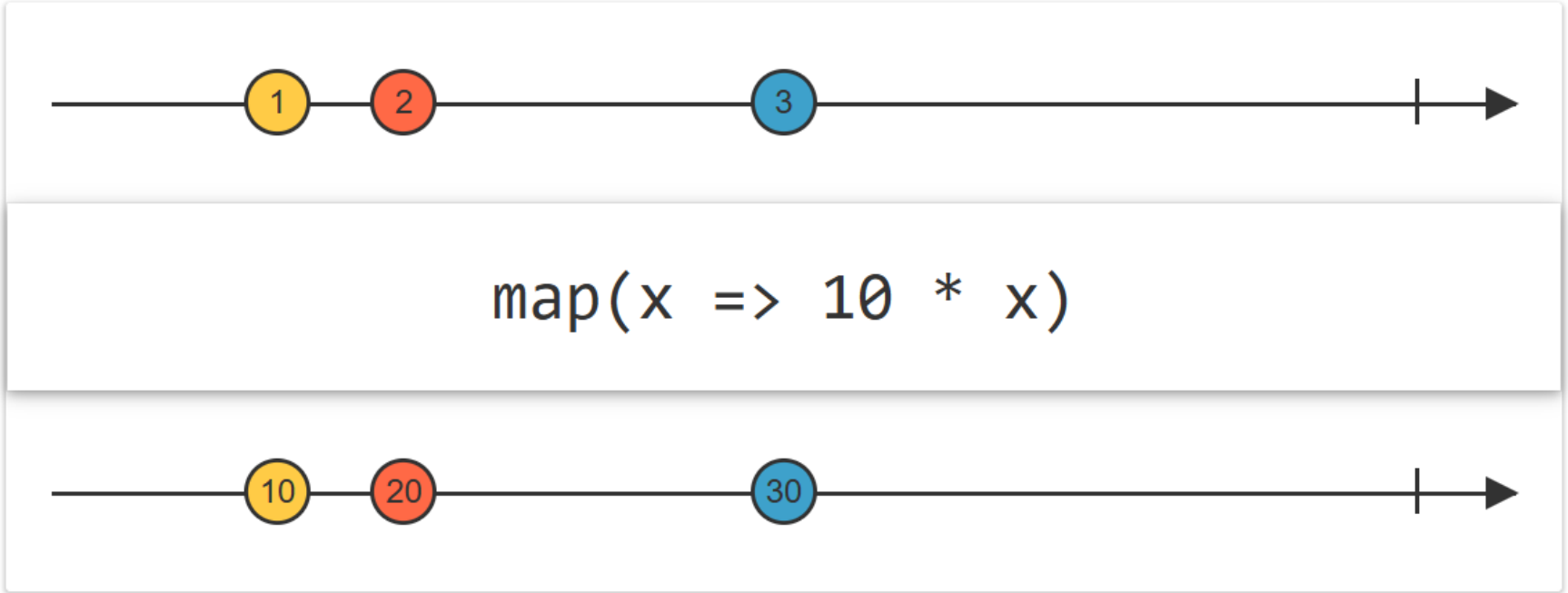
ReactiveX – Operátorok



C# (Rx.NET): **`Distinct()`, `DistinctUntilChanged()`**

Reaktív programozás

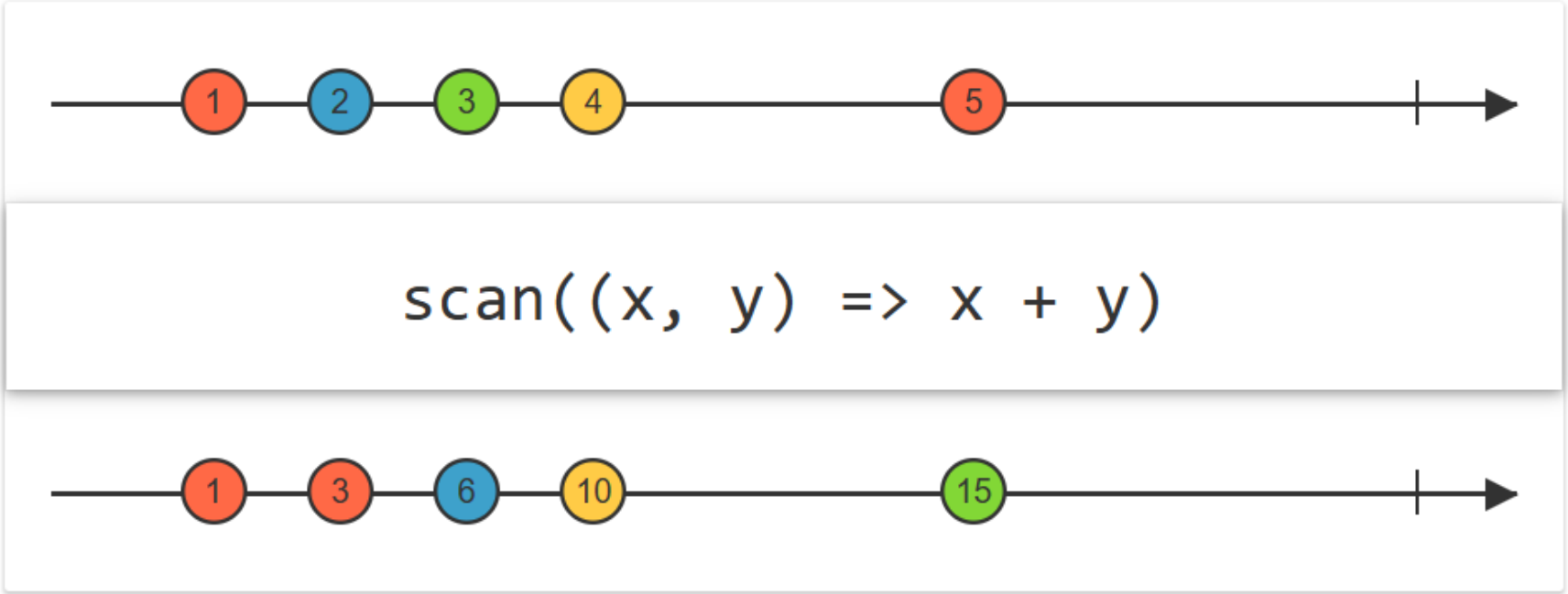
ReactiveX – Operátorok



C# (Rx.NET): **`Select()`**

Reaktív programozás

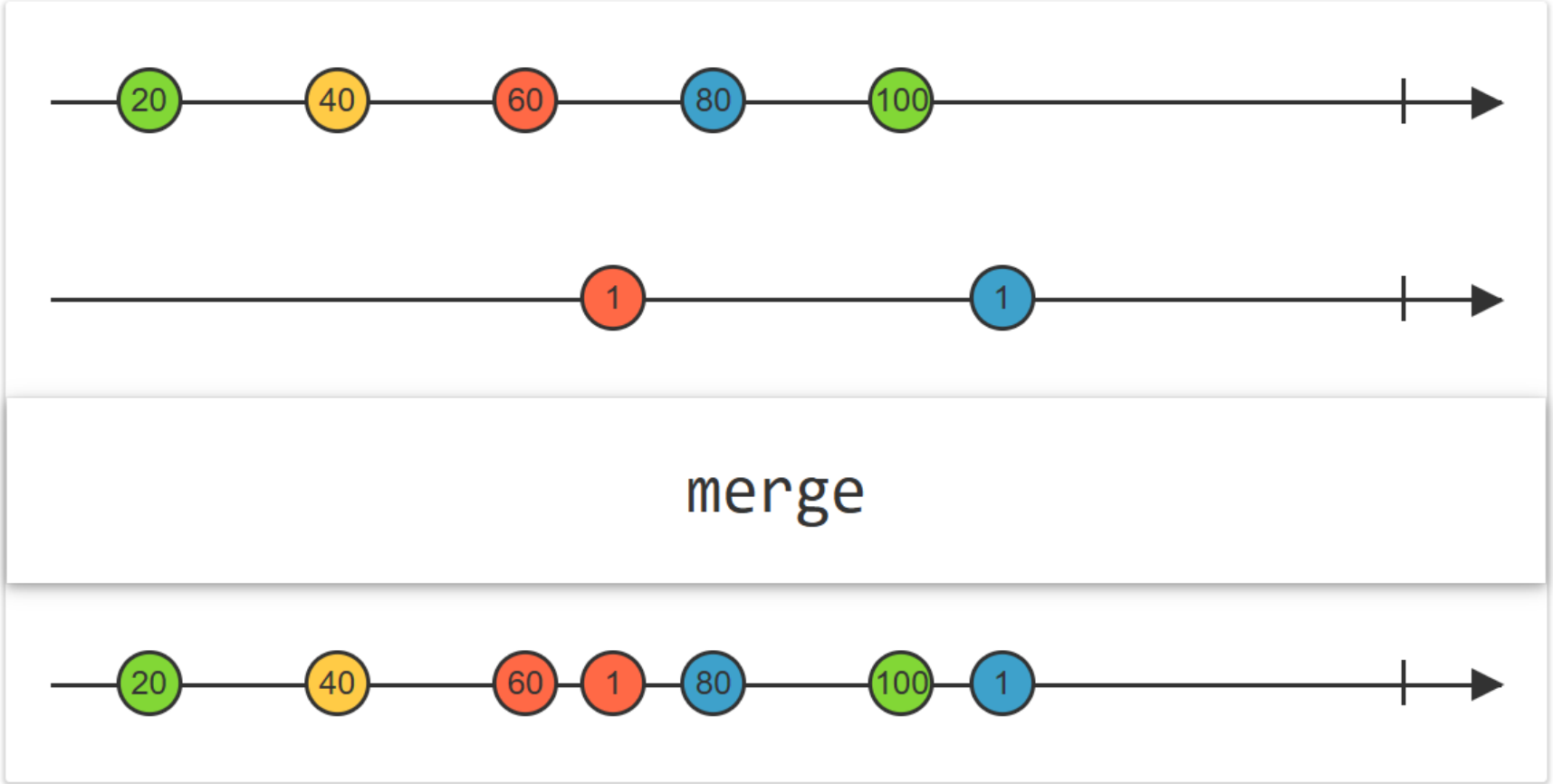
ReactiveX – Operátorok



C# (Rx.NET): **`Scan()`**

Reaktív programozás

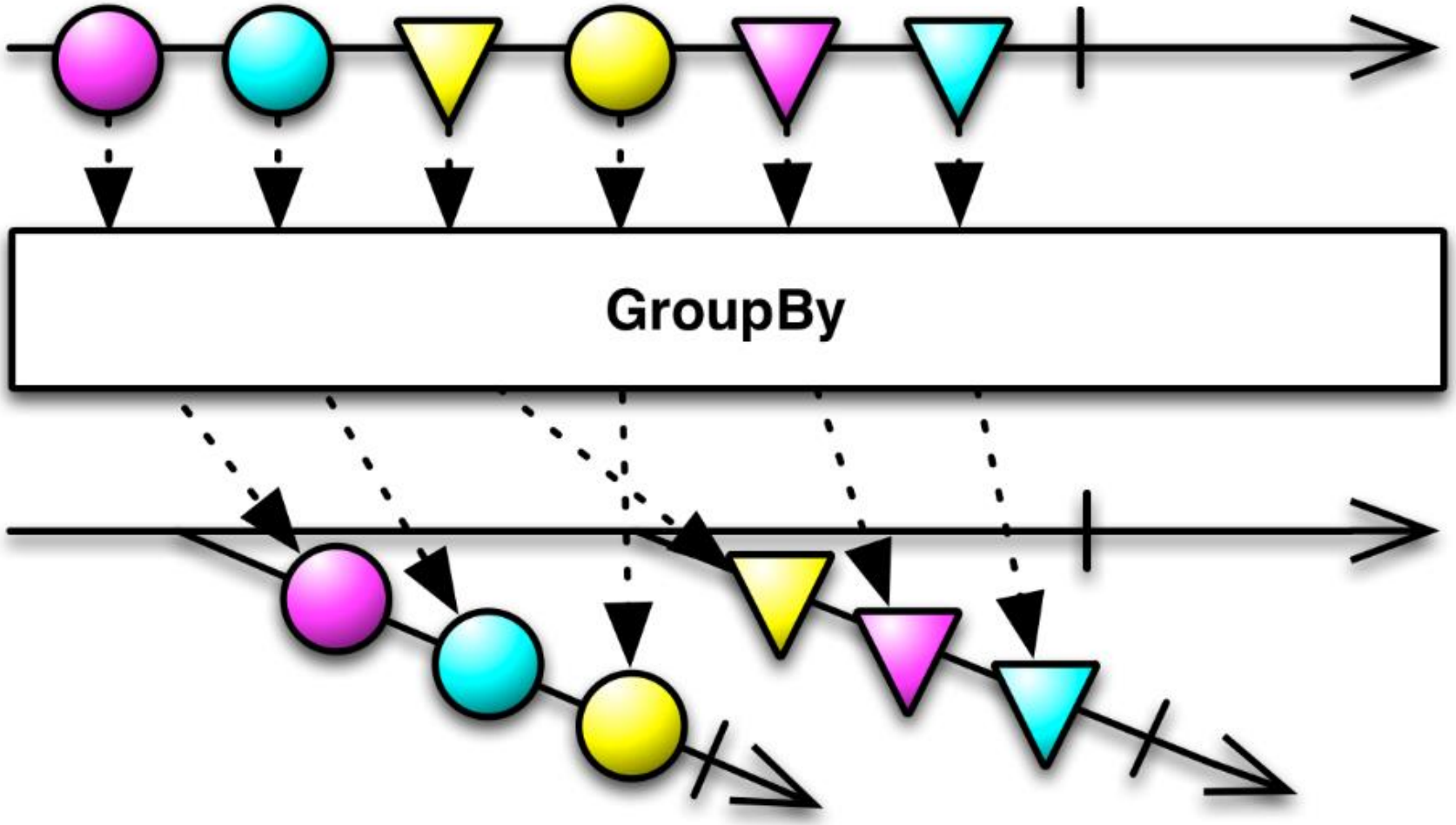
ReactiveX – Operátorok



C# (Rx.NET): **`Merge()`**

Reaktív programozás

ReactiveX – Operátorok



C# (Rx.NET): `GroupBy()`

Reaktív programozás

ReactiveX - Feliratkozás

- megfigyelhető felsorolóra feliratkozni a **`Subscribe()`** eljárásával tudunk:

```
subscription = myObservable.Subscribe(
  param => /* … */, // onNext
  ex => /* … */,    // onError
  () => /* … */     // onSuccess
);
```

- a hibakezelés és a felsorolás végének kezelése opcionális:

```
subscription = myObservable.Subscribe(
  param => Console.WriteLine(param);
);
```

Reaktív programozás

Példa

- Példa (*observable* létrehozása szimulált szenzor adatokra):

```
var sensorStream = Observable
        .Interval(TimeSpan.FromMilliseconds(250))
        .Select(t => new SensorReading
        {
            SensorId = t % 3, // 3 szenzor
            Celsius = Random.Shared.Next(-10, 60),
            Timestamp = DateTime.UtcNow
        });
```

- Ahol:

```
public class SensorReading
{
    public long SensorId { get; set; }
    public int Celsius { get; set; }
    public DateTime Timestamp { get; set; }
}
```

Reaktív programozás

Példa

- Példa (*observable* feldolgozása operátorokkal):

```
var processedStream = sensorStream
        .Where(r => r.Celsius >= -5 && r.Celsius <= 50)
        .Select(r => new ProcessedReading
        {
            SensorId = r.SensorId,
            Celsius = r.Celsius,
            Fahrenheit = r.Celsius * 9.0 / 5.0 + 32,
            Timestamp = r.Timestamp,
            Status = r.Celsius > 35 ? "WARNING" : "OK"
        })
        .GroupBy(r => r.SensorId);
```

- Ahol:

```
public class ProcessedReading
{
    public long SensorId { get; set; }
    public int Celsius { get; set; }
    public double Fahrenheit { get; set; }
    public DateTime Timestamp { get; set; }
    public string Status { get; set; }
}
```

Reaktív programozás

Példa

- Példa (feliratkozás *observable* objektumra):

```
processedStream.Subscribe(group => {
    Console.WriteLine(
        $"New group: Sensor {group.Key}");

    group.Subscribe(reading =>
    {
        Console.WriteLine(
            $"Sensor {reading.SensorId} | " +
            $"{reading.Celsius}°C / " +
            $"{reading.Fahrenheit:F1}°F | " +
            $"{reading.Status} | " +
            $"{reading.Timestamp:HH:mm:ss}"
        );
    });
});
```

Reaktív programozás

Multicasting

- alapértelmezetten egy új feliratkozáskor a megfigyelhető felsoroló inicializálási logikája végrehajtódik
- *multicast* megoldáshoz megfigyelhető felsorolónk elindítását a `Publish()` eljárással manuálissá alakítjuk

```
var sensorStream = Observable
      .Interval(TimeSpan.FromMilliseconds(250))
      //...
      .Publish(); // multicast

var sub1 = sensorStream.Subscribe(/* ... */);
var sub2 = sensorStream.Subscribe(/* ... */);

sensorStream.Connect(); // felsorolás indítása
```

Reaktív programozás

Multicasting

- az elindítást és a felszabadítást automatizálhatjuk a **`RefCount()`** használatával
  - ez egy hivatkozás (referencia) számlálást fog a háttérben végezni, és az első feliratkozáskor indítja a felsorolást, az utolsó feliratkozás felszabadításakor pedig a felsorolót is felszabadítja

```
var sensorStream = Observable
      .Interval(TimeSpan.FromMilliseconds(250))
      //...
      .Publish().RefCount(); // multicast

var sub1 = sensorStream.Subscribe(/* ... */);
// az első feliratkozással indul a felsorolás
var sub2 = sensorStream.Subscribe(/* ... */);
```

Reaktív programozás

Multicasting

- *ReactiveX*-ben a megfigyelhető felsorolókat (*observable*) két kategóriára oszthatjuk:
  - *Cold observable*: a feliratkozás (`Subscribe()`) hozza létre a gyártót, amely az elemeket felsorolja.
    - Többszörös feliratkozás több gyártót fog létrehozni.
  - *Hot observable*: a feliratkozás nem hoz létre gyártót, hanem egy azon kívül létező objektumot figyel meg és sorol fel.
    - Többszörös feliratkozás nem duplikálja a gyártót.
- Ebben a terminológiában a *multicast* használata (`Publish()`) a *cold observable*-ből egy *hot observable*-t készít.

Reaktív programozás

ReactiveUI

- Az asztali grafikus alkalmazások felületi eseményei és állapotváltozásai is kezelhetőek reaktív módon
  - .NET keretrendszerben ezt a ReactiveUI könyvtár teszi lehetővé
  - könnyen integrálható amely Windows Forms, Windows Presentation Foundation és Avalonia UI felületű alkalmazásokkal is 1-1 NuGet csomagnak a projekthez adásával
  - a *ReactiveUI* keretrendszer az *Rx.NET* osztálykönyvtárra épül
- MVVM architektúra esetén a nézetmodell megfigyelhető tulajdonságainak (**`INotifyPropertyChanged`** interfész) változása egy megfigyelhető felsorolóként is értelmezhető.
- Hasonló szemlélettel a parancsok (**`ICommand`** interfész) végrehajthatósága (**`CanExecute`**) is tekinthető egy logikai értékeket felsoroló objektumnak, amely változásairól eseményeket küld (**`CanExecuteChanged`**), azaz megfigyelhető.