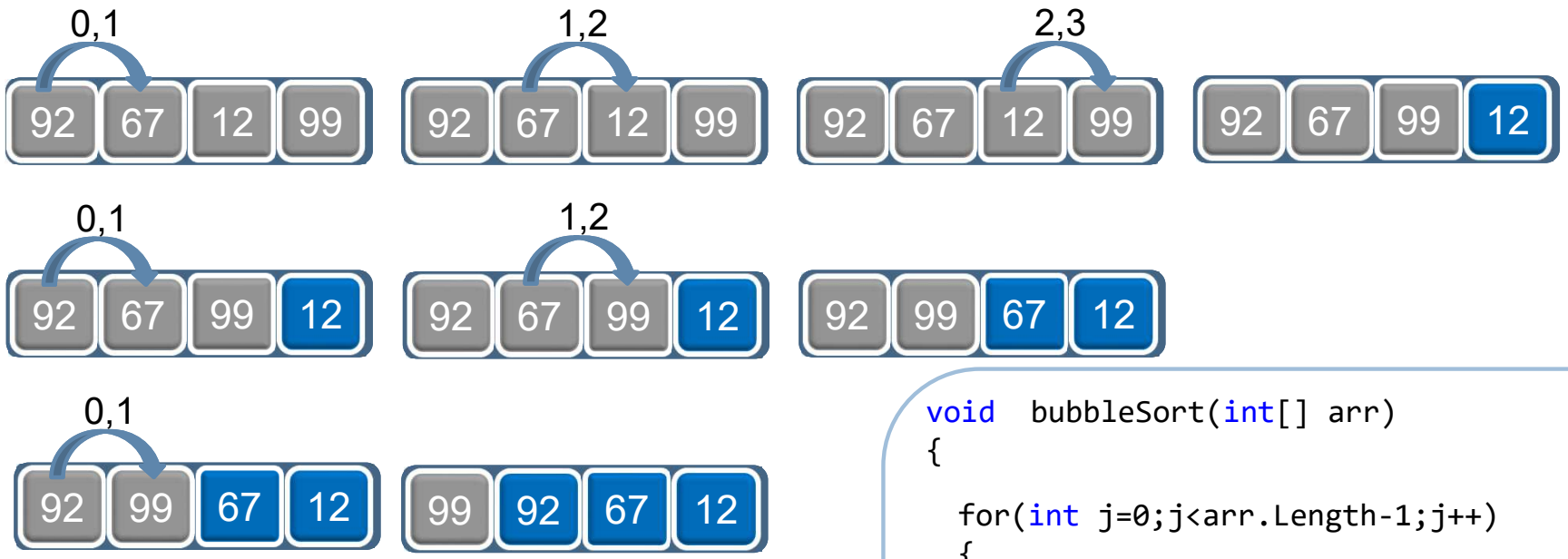
Delegate and Event

Bubble Sort Example (descending)

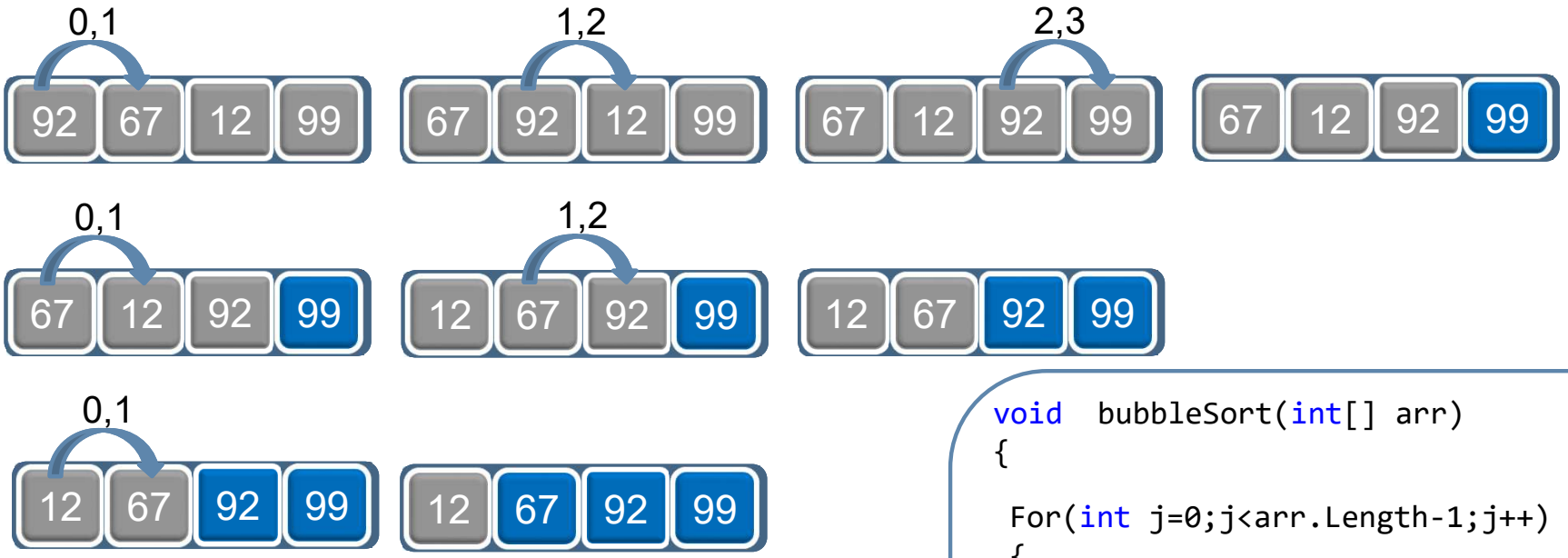



```
void  bubbleSort(int[] arr)
{

  for(int j=0;j<arr.Length-1;j++)
  {
    for(int i=0;i<arr.Length-1 -j ;i++)
    {
      if(arr[i]<arr[i + 1])
      {
       swap(ref arr[i],ref arr[i+1]);
      }
    }

  }
}
```

Bubble Sort Example (Ascending)




```
void  bubbleSort(int[] arr)
{

 For(int j=0;j<arr.Length-1;j++)
 {
  For(int i=0;i<arr.Length-1  -j  ;i++)
  {
    if(arr[i]>arr[i + 1])
    {
      swap(ref arr[i],ref arr[i+1]);
    }
  }
 }
}
```

Assignment

- Implement and Trace Bubble Sort

Delegate

- Why Delegate

```
static bool sortAscending(int L, int M)
{
        return (L > M);
}
```

```
static bool sortDescending(int L, int M)
{
        return (L < M);
}
```

```
void  bubbleSort(int[] arr, [redacted] )
{
  for(int j=0;j<arr.Length;j++)
  {
    for(int i=0;i< arr.Length-1-j;i++)
     {
       if ( [redacted] )
       {
          Swap(ref arr[i],ref arr[i+1]);

       }
     }
  }
}
```

Delegate

- Delegate is a special data type used as a **reference to method** enables passing method around like any data
- Declare Delegate Data type

```
public delegate bool Mydelegate (int L, int M);
```

delegate keyword
To identify this data type
As a Delegate

Return type of a method
That could be referenced
By a variable of this delegate

Name of delegate
Data type

Parameter list of a method
That could be referenced
By a variable of this delegate

Variable of Mydelegate could refer to any method that return bool and takes 2 integers

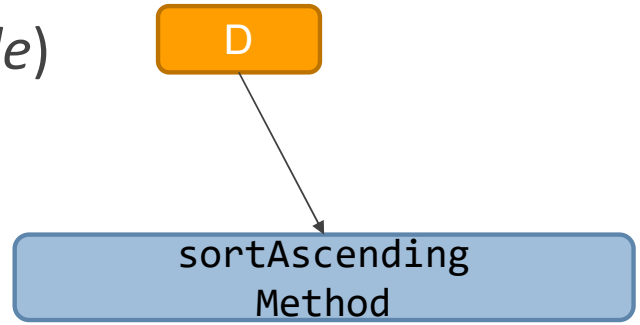
Delegate

- Declare delegate variable (*reference variable*)

```
Mydelegate D;
```

Data Type

Variable Name



- Instantiate an object of delegate (initializing delegate variable with value)

```
D = sortAscending;
…
D = sortDescending;
```

```
D = new Mydelegate(sortAscending);
…
D = new Mydelegate(sortDescending);
```

Delegate

- Passing delegate variable to method

```
bubbleSort(arr, D);
            //or
bubbleSort(arr, sortAscending);
```

Delegate variable

Value
(method name)

```
void bubbleSort(int[] arr, Mydelegate del)
{
…
  if ( del ( arr[i] , arr[i + 1] ) )
  {
   Swap(ref arr[i],ref arr[i+1]);
  }
…
}
```

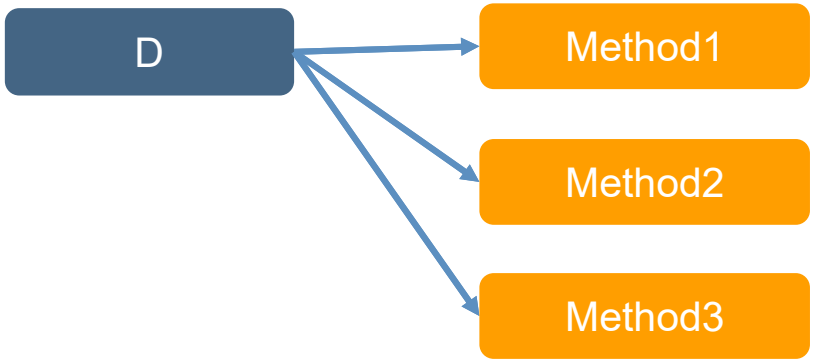
- Invoking delegate

```
D(x,y); //As if it is a method
```

Delegate

- Multicast Delegate

```
D = Method1;
D += Method2;
D += Method3;
```



Generic Delegate

```
public static void
swap(ref int l,ref int m)
{
  int temp;
  temp = l;
  l = m;
  m = temp;
}
```

- Normal Delegate

```
public delegate void sDelegate(ref int x, ref int y);
```

- Generic Delegate
  - Return void

```
public delegate void swapDelegate<T>(ref T x, ref T y);
```

```
public delegate void swapDelegate2<T1,T2>(ref T1 x, ref T2 y);
```

Generic Delegate

```
public static int sum(int l, int m)
{
  int result;
  result =l+m;
  return result;
}
```

- Generic Delegate
  - Return data type

```
public delegate TResult SumDelegate<T1,T2,TRseult>(T1 x, T2 y);
```

```
SumDelegate<int,int , int> sumd = sum;
int result = sumd(10, 15);
```

General Purpose Delegates (C# 3.0)

specifies that the type parameter is ***Contravariant***
(Same type or super class type)

- System.Action (void as a return)
  - public delegate void Action ();
  - public delegate void Action< in T>(T arg)
  - public delegate void Action<in T1, in T2>(in T1 arg1, in T2 arg2) T16

specifies that a type parameter is ***Covariant***.
(Same type or derived type)

- System.Func
  - public delegate TResult Func< out TResult>();
  - public delegate TResult Func<in T, out TResult>(T arg)
  - public delegate TResult Func<in T1, in T2, out TResult>(in T1 arg1, in T2 arg2)

```
public delegate bool Mydelegate(int L, int M);
….
Mydelegate d;
```

□Gụŋç□îŋŧʃ□□îŋŧʃ□□čộộľ□□đ□

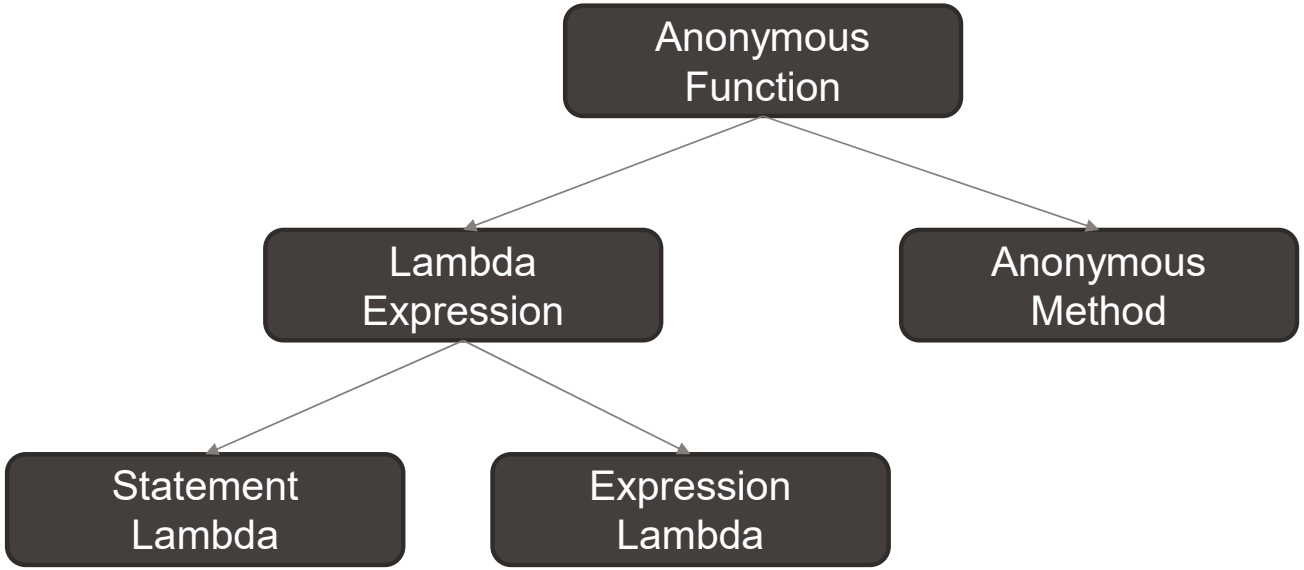
```
void bubbleSort(int[] arr,Func<int, int, bool> d)
{
}
```

Assignment

- Modify Menu Program
  - Using **List<T>.Sort(Comparison<T>)**
    - Write anonymous method that implement Comparison< T> delegate for sort employee (by salary ,by Name) ascending or descending

Anonymous Function

- An anonymous function is an "inline" statement or expression that can be used wherever a *delegate type is expected*



Anonymous Method

```
public delegate bool Mydelegate(int L, int M); //Declare Delegate

void bubbleSort(int[] arr, Mydelegate d) //Declare Method
{ ...}
```

```
bubbleSort( arr,
                    delegate ( int A, int B )
                              {
                              return A > B ;
                              }

);
```

Anonymous Method

Anonymous Method

- anonymous method could access outer method variables

```
void method1()
        {
            string str ="Hello";
            method2(delegate (string s)
              {
                Console.writline(str +" " +s);
              }
            );
        }
void mehod2(Action<string> action)
        {
            action("world");
        }
```

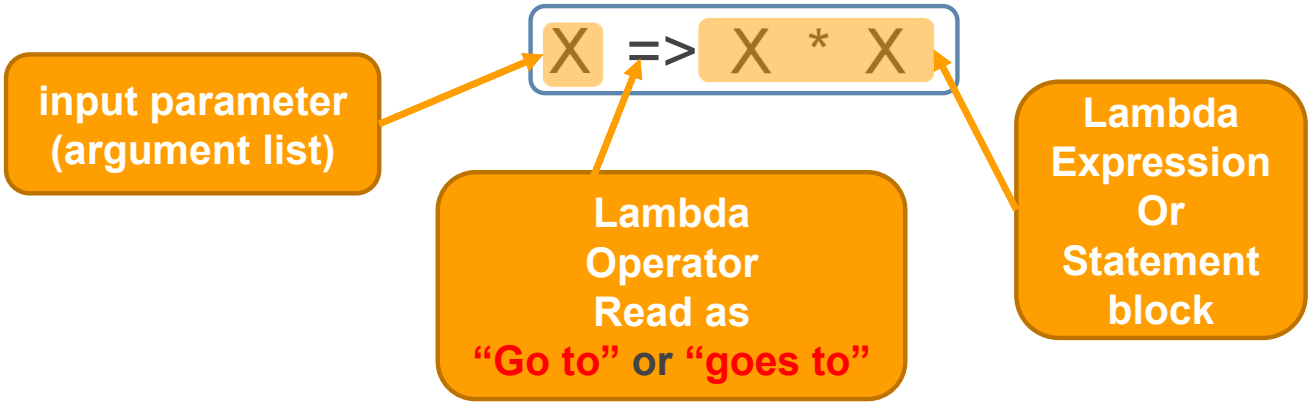
Anonymous Method

- It cannot contain jump statement like *goto* , *break* or *continue*.
- It cannot access *ref* or *out* parameter of an outer method.
- It cannot have or access unsafe code.
- It cannot be used on the left side of the is operator.

Lambda Expressions

- Code could be represented as Expression that compiled to delegate
- Delegate Types
  - Action delegate →no return
  - Func delegate → return
- Lambda expression anatomy

```
Func<int, int> square = x => x * x;
Console.WriteLine(square(5)); // 25
```



Lambda Expression

- Statement Lambda

```
bubbleSort(arr,
        (A,B) =>
        {
          return A > B;
        }
);
```

  - One parameter

```
A=>
```

  - Parameterles

```
()=>
```

Lambda Expression

- Expression Lambda

```
bubbleSort( arr, (a, b) => a < b );
```

Expression Lambda

Expression-bodied members C# 6 -7

- Could be used whenever the logic of the member Consist of single-line statements
- This member could be
  - Method
  - Property

```
member => expression;
```

Expression-bodied members C# 6 -7

- Method

```
class Person
{
    private string fname;
    private string lname;
    public Person(string firstName, string lastName)
    {
        fname = firstName;
        lname = lastName;
    }
    public override string ToString() => $"First Name:{fname} \nLastName:{lname}";
    public void Display() => Console.WriteLine(ToString());
}
```

Expression-bodied members C# 6 -7

- Property
  - Read only Property (get only)

```
PropertyType PropertyName => expression;
```

```
□řụčľîç□çľăşş□Ľộçắţîộŋ
□□□□
□□□□□□řsîŵắţê□ştsîŋĝ□ľộçắţîộŋŃắṇê□
□□□ □ □
 □ □□□řụčľîç□ştsîŋĝ□Ńắṇê□□□□ľộçắţîộŋŃắṇê□
□□□□□
```

Expression-bodied members C# 6 -7

- Property
  - set, get

```
řụčľîç□çľǎşş□Ľộçǎţîộŋ
□
□□□řsîŵǎţê□şţsîŋĝ□ľộçǎţîộŋŃǎṇê□
□□ □□
 □ řụčľîç□şţsîŋĝ□Ńǎṇê
□□□□□□□□
□□□□□□□□□□□□ĝêţ□□□□ľộçǎţîộŋŃǎṇê□
□□□□□□□□□□□□ṣêţ□□□□ľộçǎţîộŋŃǎṇê□□□ŵǎľụê□
□□□□□□□□
□
```

Implicit Typed Local Variable var

- *var* keyword
  - Used for declare a variable its data type would be detected at the run-time by the compiler
    - Variable initialization must be on the declaration

```
employee em = new employee { ID = 10, Name = "Ahmed", Salary = 1000f };
em.ID = 2;
```

```
var v = new employee { ID = 10, Name = "Ahmed", Salary = 1000f };
v.ID = 2;
```

Anonymous Type

- Anonymous type is a simple class that is created by the compiler on the fly to store a set of values
- Declare an object of anonymous data type done by using *new* keyword followed by object initializer
- Var keyword must be used to reference anonymous Type since it doesn't have a type name

```
var v2 = new { Price = 200f, Name = "juice" };
v2.name = "milk"; //error readonly
```

Object Initializer

properties

Anonymous Type

- Anonymous type overrides methods
  - ToString()
  - Equals()
- Mainly used in LINQ

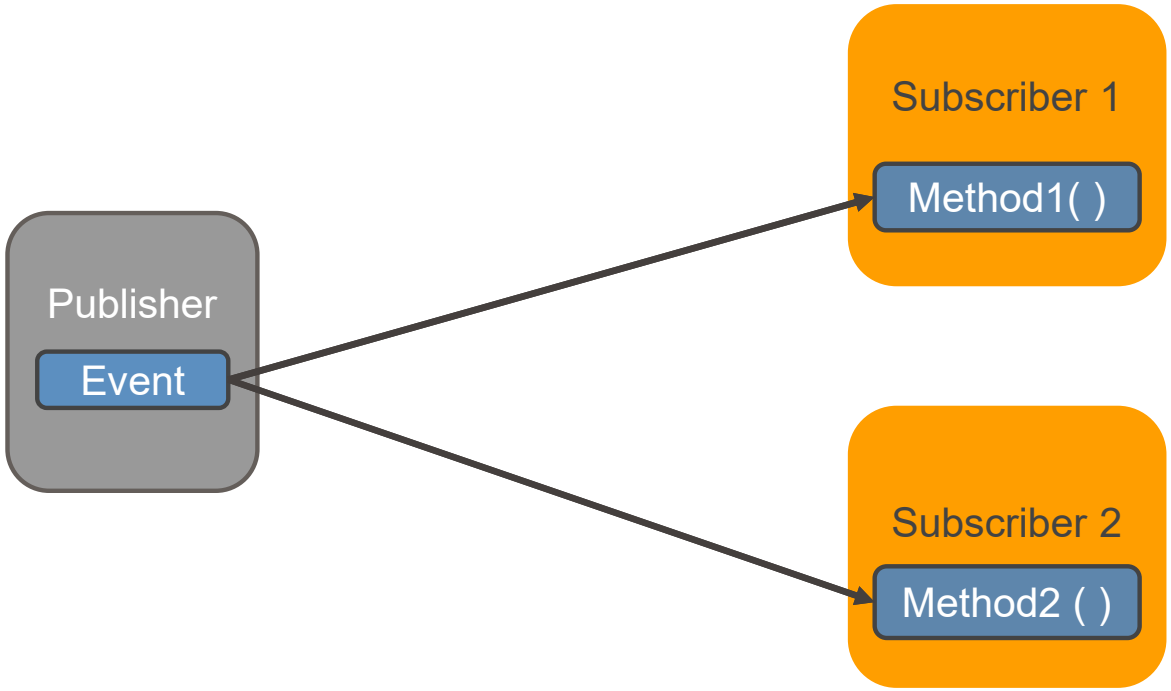
Events

- Event is a ***special*** multicast Delegate variable
- Event Declaration

```
public delegate void mydelegate(int x); // declare type (delegate)
…
public event mydelegate d; // declare event
```

Events

- Publisher Subscriber Design Pattern
  - Event
  - Event Handler

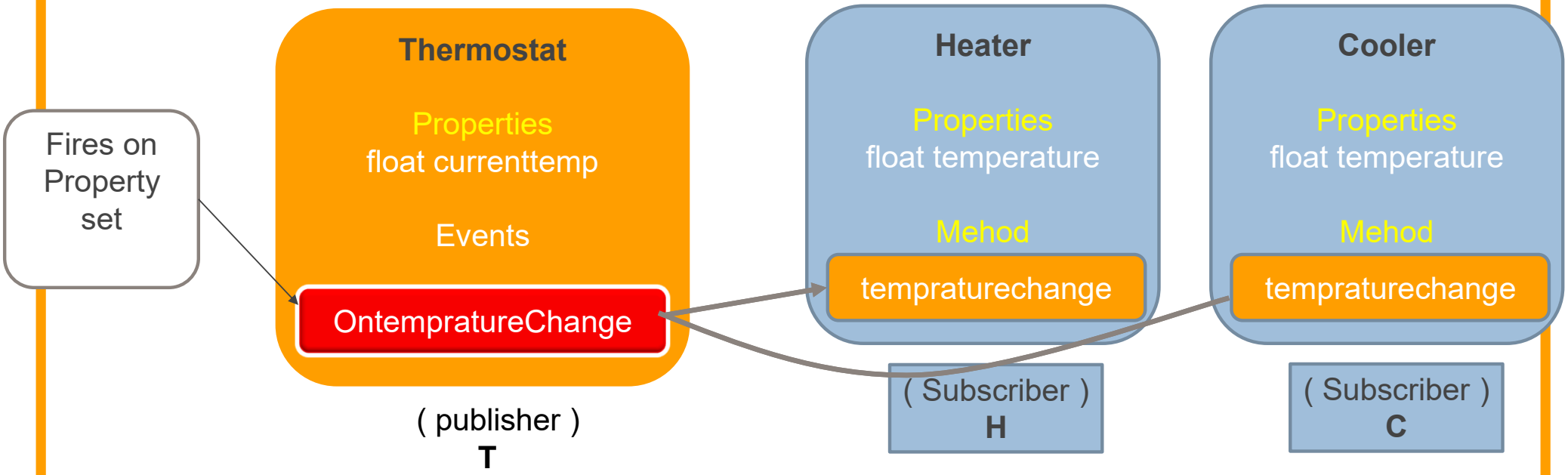


Why event?

- Prevent errors or bugs results from
  - Cancelling other subscriber by (encapsulation subscriber)
    - Using Assignment += instead =
    - Only subscriber can cancel its subscription
  - Only Publisher can invoke (call) event

Assignment

- Write a program for heater , cooler and thermostat




```
T.OntempratureChange += C.tempraturechange;
T.OntempratureChange += H.tempraturechange;
```

Thank you

You can find me

wael.radwan@gmail.com