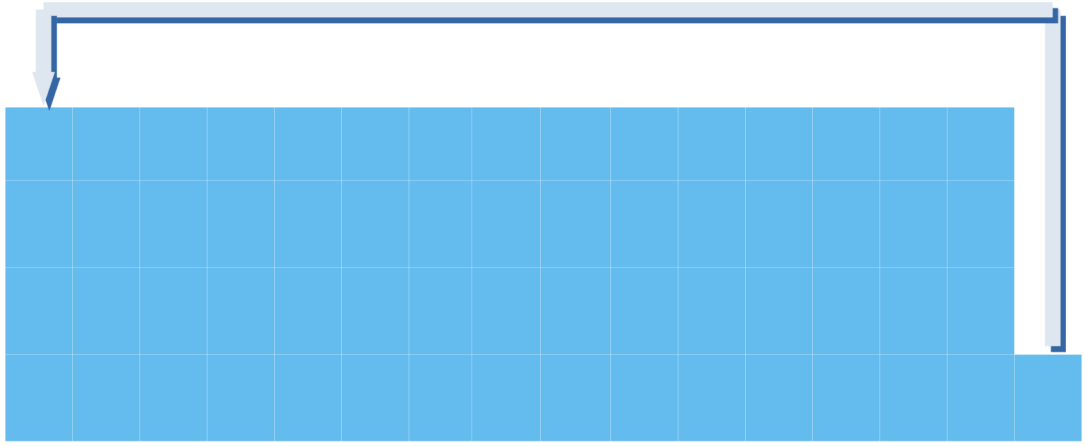
Circular Buffer



Fixed capacity: `CircularBuffer<Data>(N)` stores no more than `N` elements

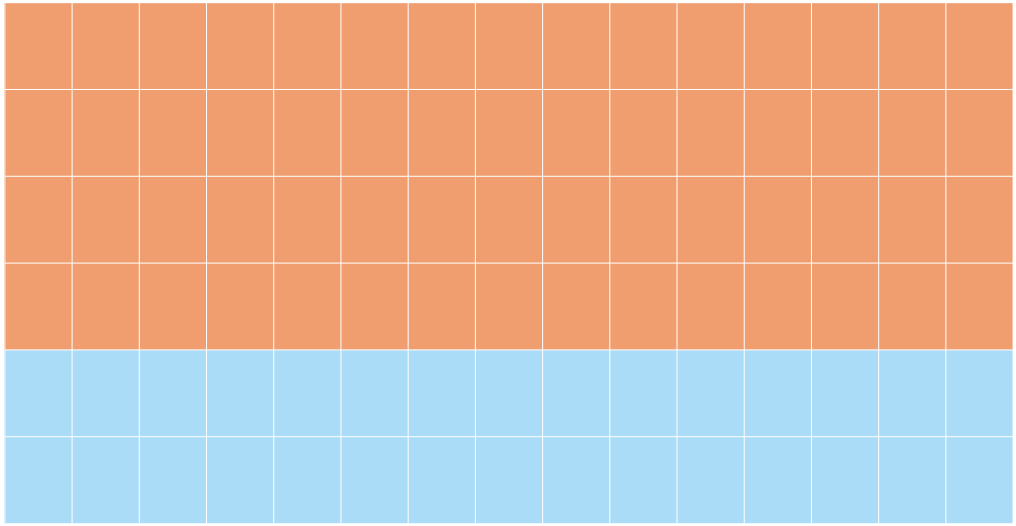
Computing Averages


```
(from row in rows select row.Value).Average()
```

Static Extension Methods
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods

Language-Integrated Query (LINQ)
https://docs.microsoft.com/en-us/dotnet/standard/linq/
http://www.java2s.com/Tutorial/CSharp/0450__LINQ/Catalog0450__LINQ.htm

Computing Averages

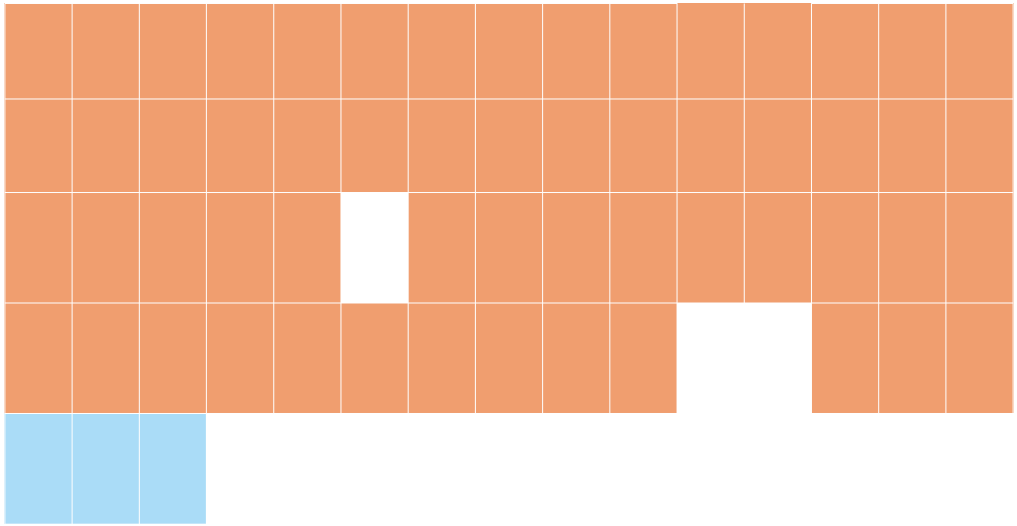


```
class Data {
 DateTime timeStamp;
 Int32 value;
}
```

```
 (from row in rows
where (now – row.TimeStamp).TotalMinutes <= interval
select row.Value).Average()
```

Extend the buffer to more than 60 elements - compute 1 minute average

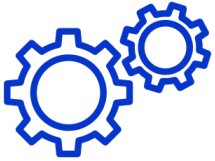
Computing Averages



```
(from row in rows
where (now – row.TimeStamp).TotalMinutes <= interval
select row.Value).Average()
```

Missed info and buffer spans longer than 1 minute - compute 1 minute average

Design





https://en.wikipedia.org/wiki/Windows_service

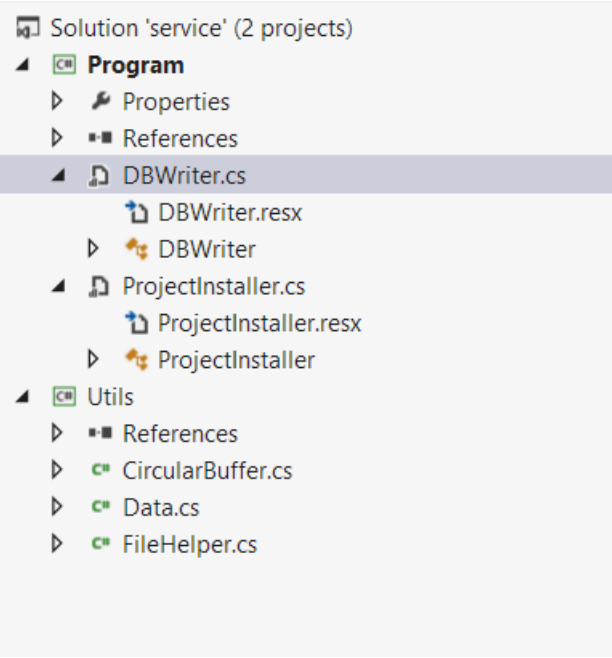
https://en.wikipedia.org/wiki/Performance_Monitor

Windows Service with two timers:

- collect System Performance Metric every second
- calculate Averages on demand
- optionally calculate Averages once per minute

Services Controller manages install, start, end of a service
does not need user to be logged on
Services respond to custom commands

Service Component



Every *Windows Service* is a subclass of `System.ServiceProcess.ServiceBase`

The code is largely IDE-generated: `Program`,`Service`, `ProjectInstaller` classes

One is required to provide the constructor, override `OnStart`, `OnStop`,`OnCustomCommand` and `Dispose`

One can add method named `Main` with no special meaning: *Windows Service* cannot be run directly

One can instantiate a `System.Timers.Timer` and create method conventionally named `OnElapsedTimer` to handle `Elapsed` event

https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicebase
https://docs.microsoft.com/en-us/dotnet/api/system.timers.timer

Data Collector

control.ps1*

```
1  param (
2      [String]$name = 'DBWriter',
3      [String]$filepath = 'C:\temp\loadaverage.txt',
4      [int]$command = 200
5  )
6  [System.Reflection.Assembly]::LoadWithPartialName('System.ServiceProcess')
7  $controller = new-object System.ServiceProcess.ServiceController($name)
8  $controller.ExecuteCommand($command)
9  get-content -path $filepath
10
```

Powershell script loads `System.ServiceProcess.dll`, finds Windows Service by `$name`, invokes its `customCommand` Service computes load averages and saves information to the file

```
protected override void OnCustomCommand(int command) {
    base.OnCustomCommand(command);
    switch (command) {

        case(200):
            AverageData();
            Commit();
            break;
```