

EKON 27



Parallel Programming with OmniThreadLibrary

Primož Gabrijelčič

About me

Primož Gabrijelčič

<http://primoz.gabrijelcic.org>

- programmer, MVP, writer, blogger, consultant, speaker
- Blog <http://thedelphigeek.com>
- Twitter [@thedelphigeek](https://twitter.com/thedelphigeek)
- Skype [gabr42](https://skype.com/gabr42)
- LinkedIn [gabr42](https://www.linkedin.com/in/gabr42)
- GitHub [gabr42](https://github.com/gabr42)
- SO [gabr](https://stackoverflow.com/users/117077/gabr)

The Delphi Geek

random ramblings on Delphi, programming, Delphi programming, and all the rest

Friday, July 21, 2023

CQLBr for Delph/Lazarus

It is so nice when you see how a small idea grows into a nice, rounded project!

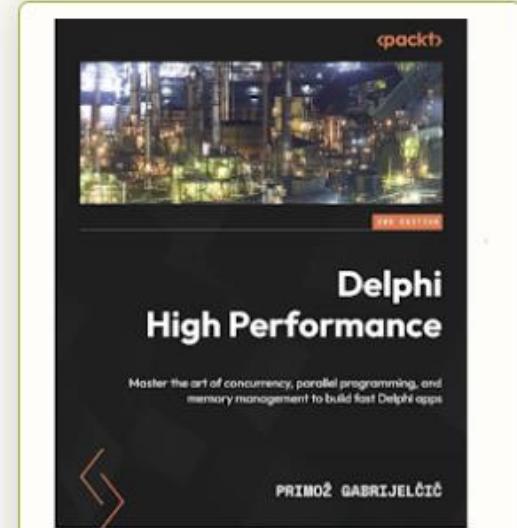
Years ago I wrote a unit that allowed you to write SQL statements as Pascal code ([GpSQLBuilder](#)). This has allowed me to write a code like this:

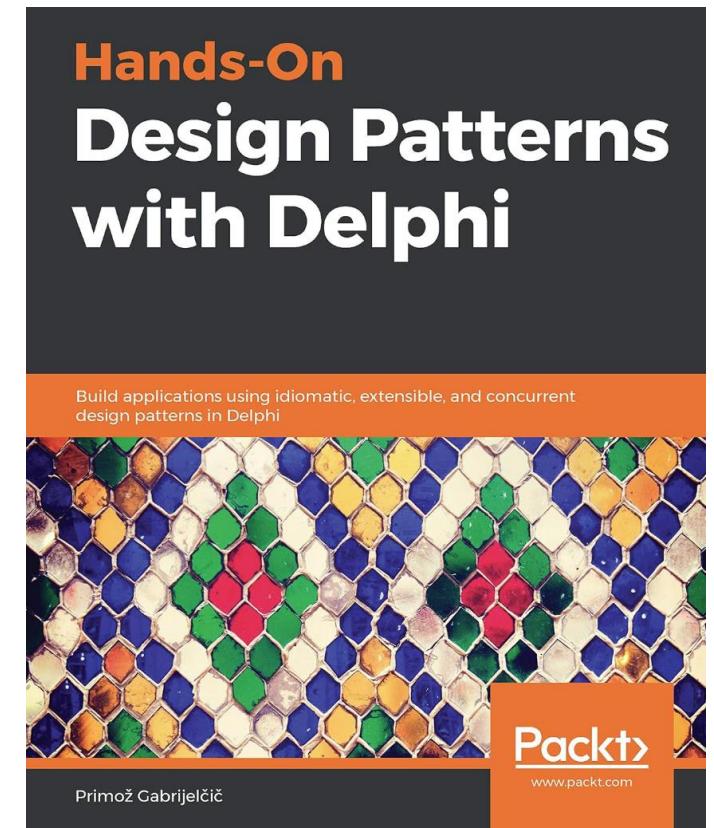
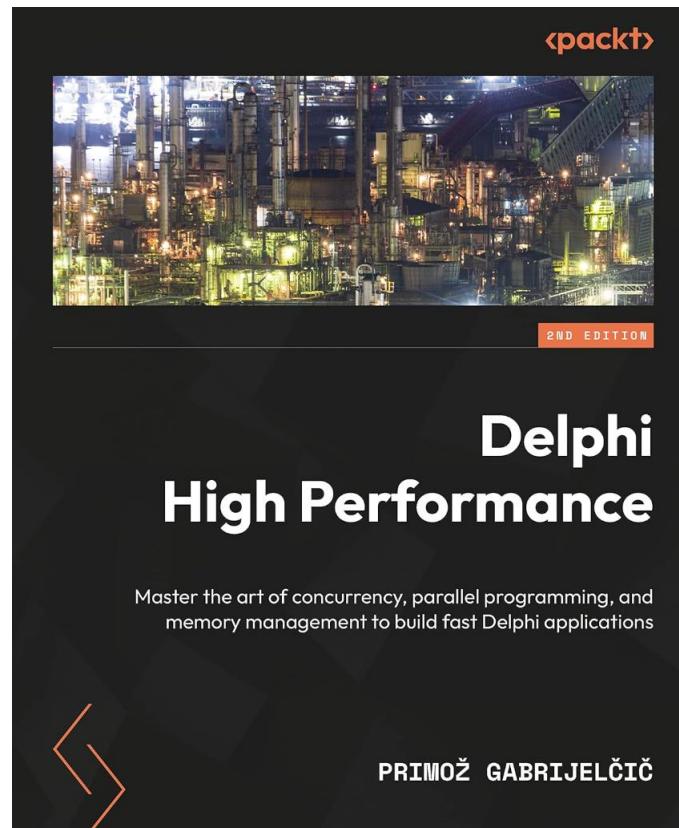
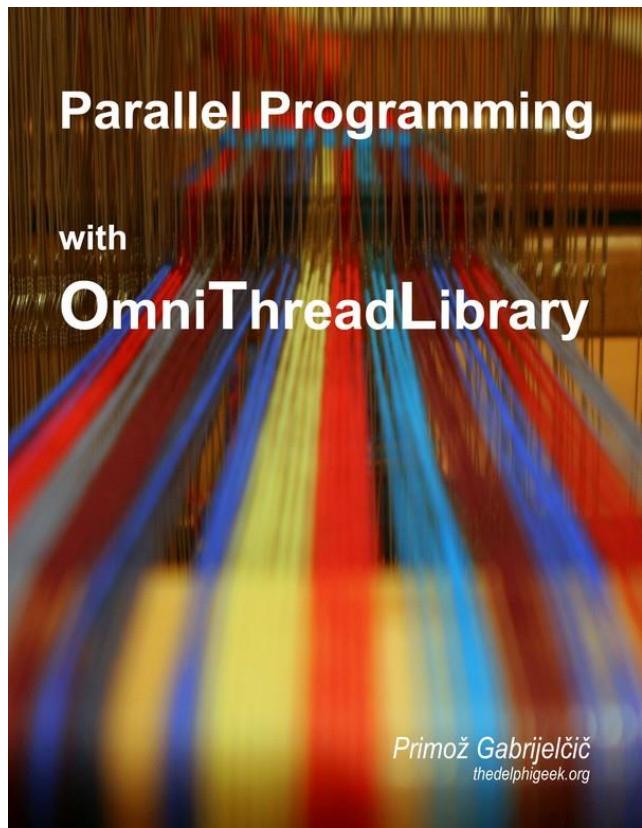
```
query := CreateGpSQLBuilder;
query
  .Select.All
  .From(DB_TEST)
  .OrderBy(
    query.&Case
      .When([COL_2, '< 0']).&Then(COL_3)
      .&Else(COL_4)
    .&End);
```

It was a small project with minimum support -- as long as it generated SQL code that I've needed, I was fine with it. Much of the SQL language support was missing, there was no support for different SQL dialects and so on ...

Luckily, Isaque Pinheiro liked the idea and converted it into a full-fledged library with support for multiple SQL dialects, much more complete SQL language support, units tests, installer, a ton of samples and more.

[Read more »](#)





<https://delphi-books.com/>

OmniThreadLibrary

OmniThreadLibrary

- VCL for multithreading / Parallel programming patterns
- Released under an OpenBSD license
- Supports Delphi 2007 to Delphi 12
 - Yes, 18 releases!
 - Windows 32- or 64-bit; VCL / service / console
 - Delphi 2007: Only low-level parallel programming
 - Delphi 2009, 2010: Slight limitations
- Installation
 - GetIt
 - ‘download and unpack’

OmniThreadLibrary on the web

- **Home** <https://www.omnithreadlibrary.com>
- **Code** <https://github/gabr42/omnithreadlibrary>
- **Book** <https://leanpub.com/>, <https://lulu.com/shop>
- **Support** <https://en.delphipraxis.net/forum/32-omnithreadlibrary/>
- **Demos** OmniThreadLibrary *tests* folder

OmniThreadLibrary

- Two distinct “flavours”
 - Low-level multithreading
 - High-level multithreading (not in Delphi 2007)
- Data structures
 - TOmniValue
 - Synchronization
 - Communication
 - All can be used with Delphi’s TThread or ITask!

Low-level OmniThreadLibrary

Low-level OmniThreadLibrary

- Design principles
 - Tasks
 - Bidirectional messaging
 - Lock-free data structures
 - Thread pools
 - Fluent programming
 - NO big Execute function

Task management

- Task control: IOmniTaskControl
 - Controls the lifecycle of the task
 - Handles communication on the owner side
 - Typically lives in main thread (not required)
- Task: IOmniTask
 - Controls the task
 - Handles communication on the task side

Simplest possible task

```
procedure TfrmLowlevel.btnSimpleTaskClick(Sender: TObject);
begin
  CreateTask(procedure (const task: IOmniTask)
    begin
      MessageBeep(MB_ICONEXCLAMATION);
    end,
    'Beep')
  .Unobserved
  .Run;
end;
```

Task worker

- Method
- Anonymous method
- Class (preferred)
 - “event-handler” style programming

Parameters

- `IOmniTaskControl.SetParameter('name', value)`
- `ITask.Param['name']`

Timers

- SetTimer
- ClearTimer

Communication

- ITaskControl/ITask
 - Comm.Send, Comm.Receive
- ITaskControl
 - OnMessage
 - Invoke
- ITask
 - Automatic dispatch
 - Invoke, InvokeOnSelf

Communication

- Communication replaces locking!
- Event handlers are always executed in the correct thread

Exit status

- ITask
 - SetExitStatus
- ITaskControl
 - ExitCode
 - ExitMessage
 - FatalException
 - DetachException

Task organization

- Parallel
 - Task groups
 - Join
 - Leave
- Sequential
 - ChainTo

```
task2 := CreateTask(SecondTask);
```

```
task1 := CreateTask(FirstTask).ChainTo(task2).Run;
```

Cancellation

- Cancellation token
 - IOmniCancellationToken
- IOmniTaskControl
 - CancelWith(token)

Windows

- MsgWait
- Alertable
- RegisterWaitObject(THandle, method)
- UnregisterWaitObject(THandle)

High-Level OmniThreadLibrary

Patterns

• Tasks

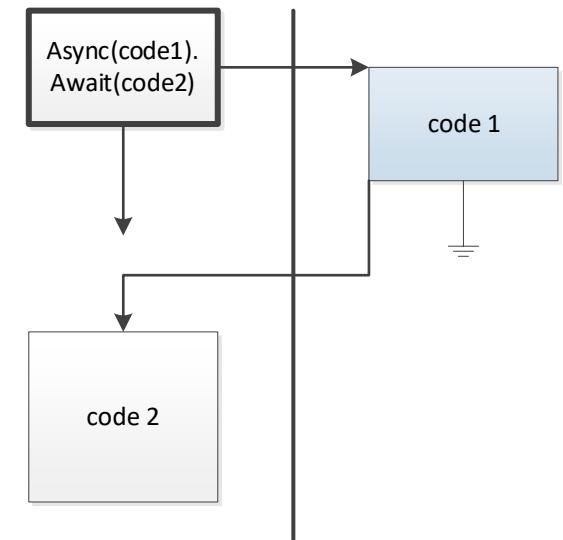
- Patterns
- No need to manage thread/task “plumbing” anymore
- OtlParallel

OmniThreadLibrary patterns

- Async/Await
- Future
- Join
- Parallel For
- Parallel ForEach
- Parallel Task
- Background Worker
- Fork/Join
- Pipeline
- Map/Reduce
- Timed Task

Async/Await

```
Async(code1).Await(code2);
```

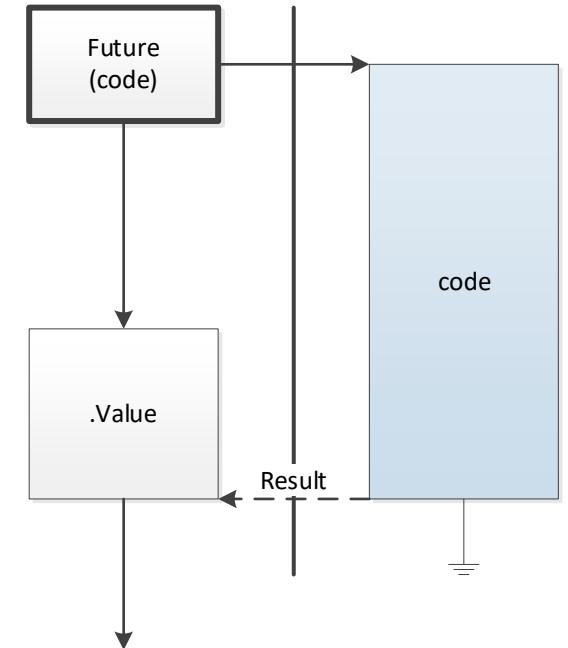


Future

`Parallel.Future<T>(func)`

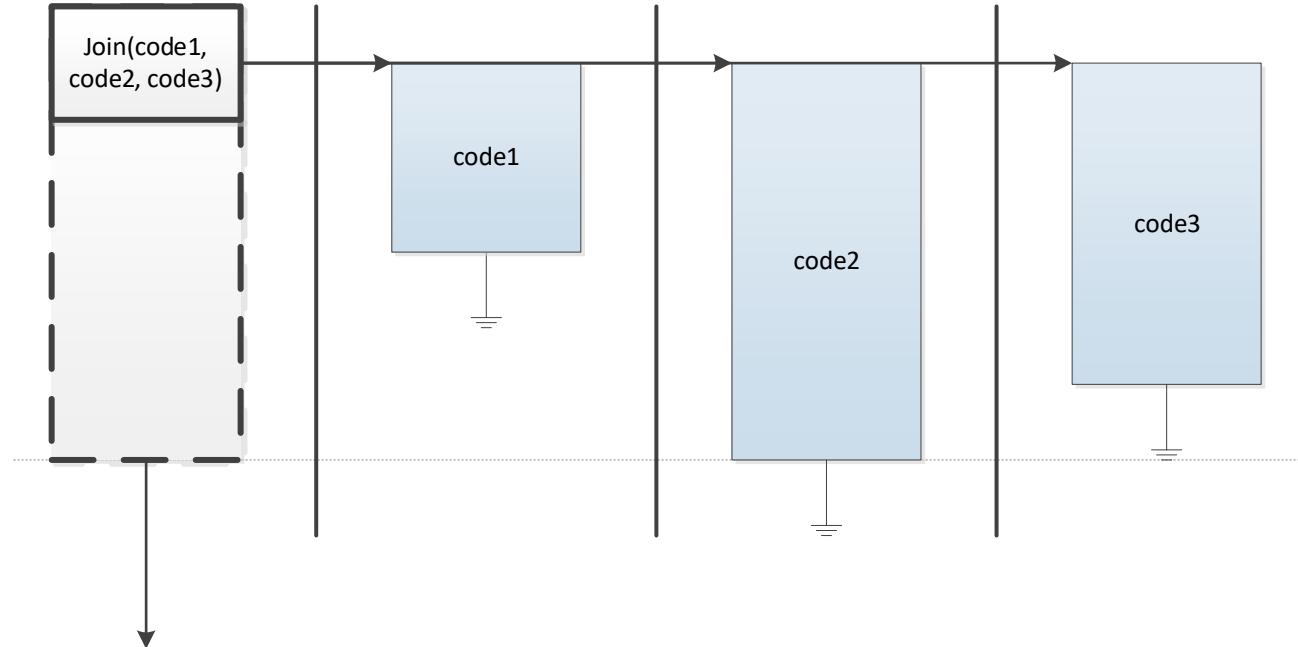
- PPL

`TTask.Future<T>(func)`



Join

```
Parallel.Join([code1, code2, ...]);
```

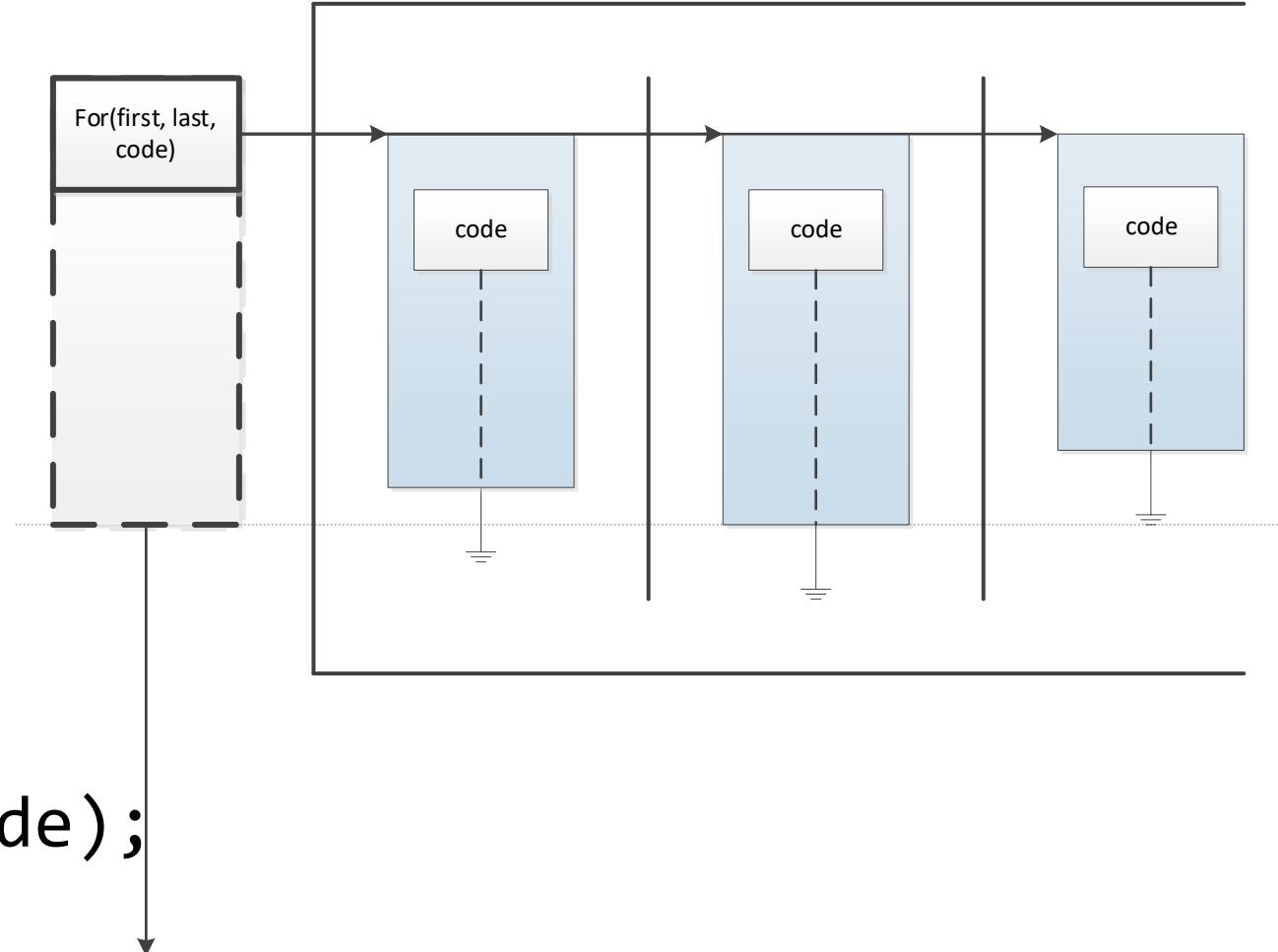


- PPL

```
TParallel.Join([Task1, Task2...]);
```

Parallel For

```
Parallel.For(from, to)  
.Execute(code);
```

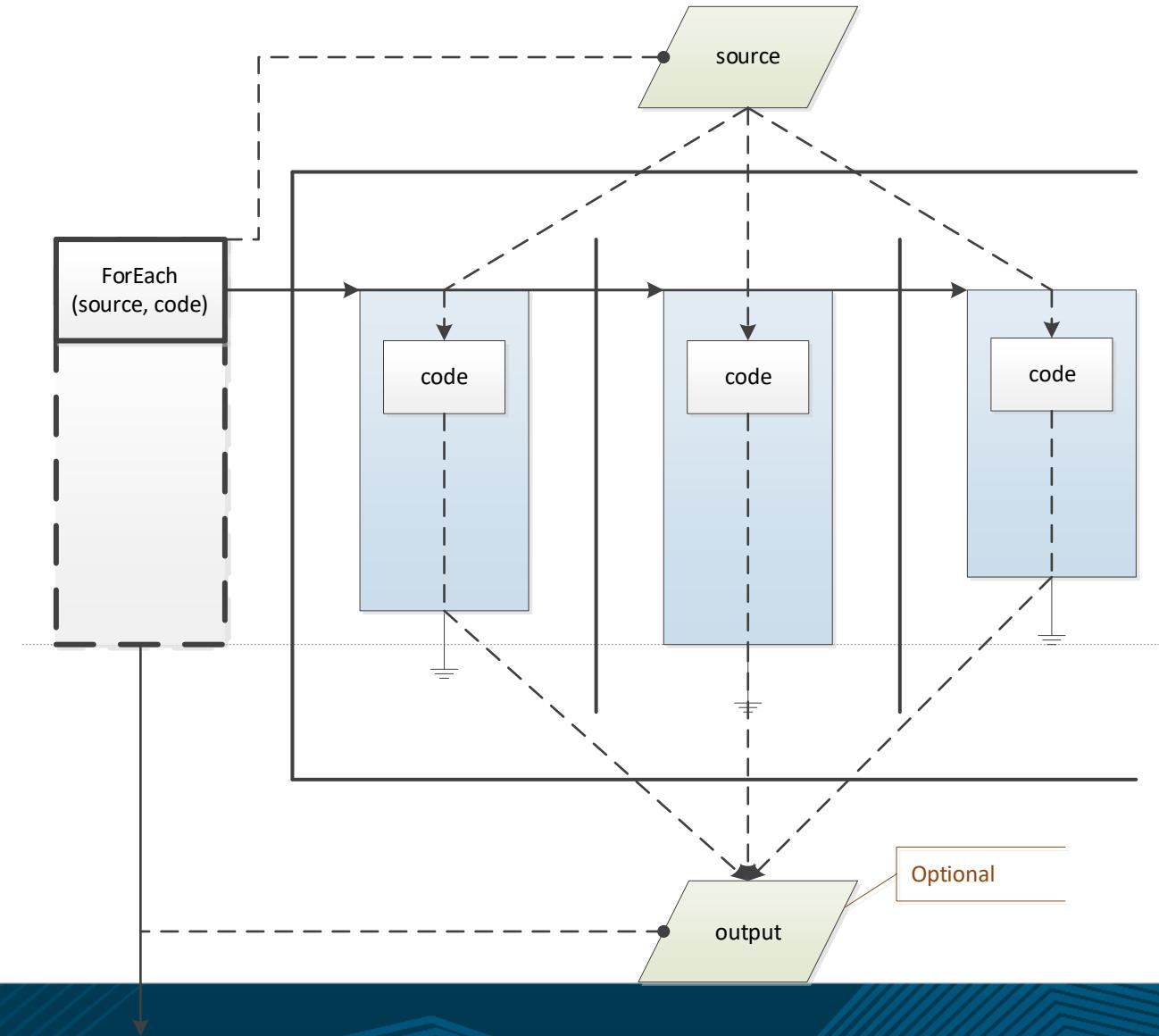


- PPL

```
TParallel.For(from, to, code);
```

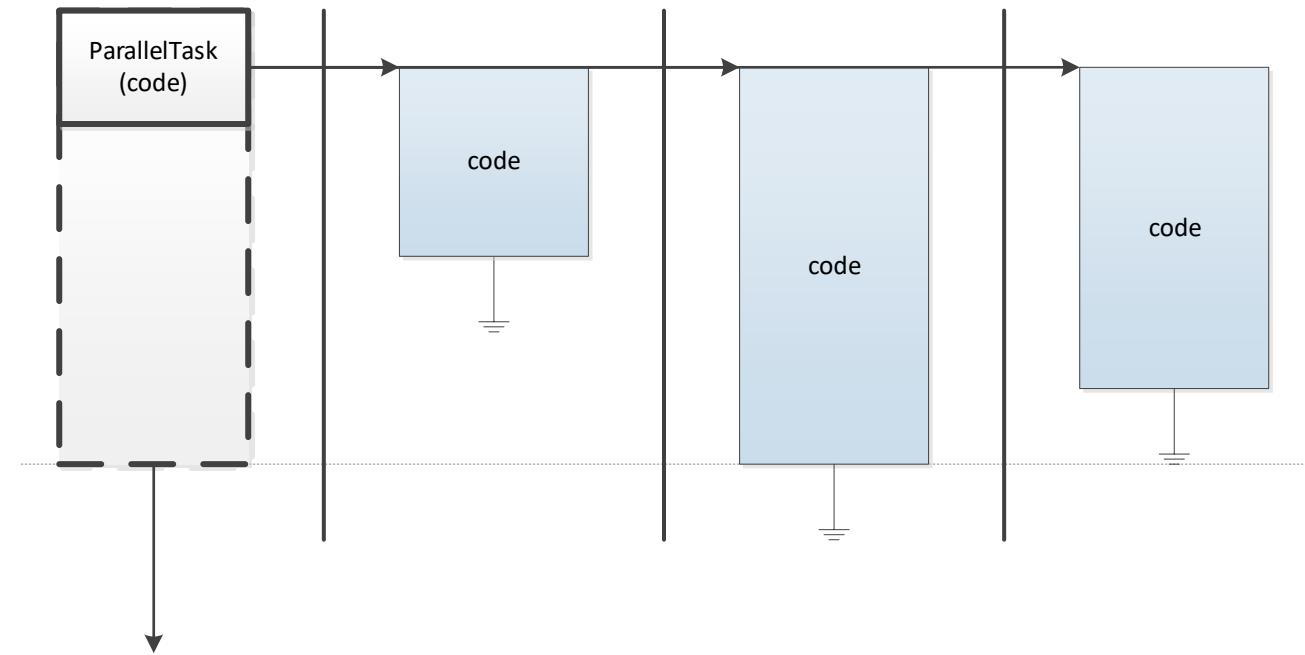
Parallel ForEach

```
Parallel.ForEach(  
    dataSource, code);
```



Parallel Task

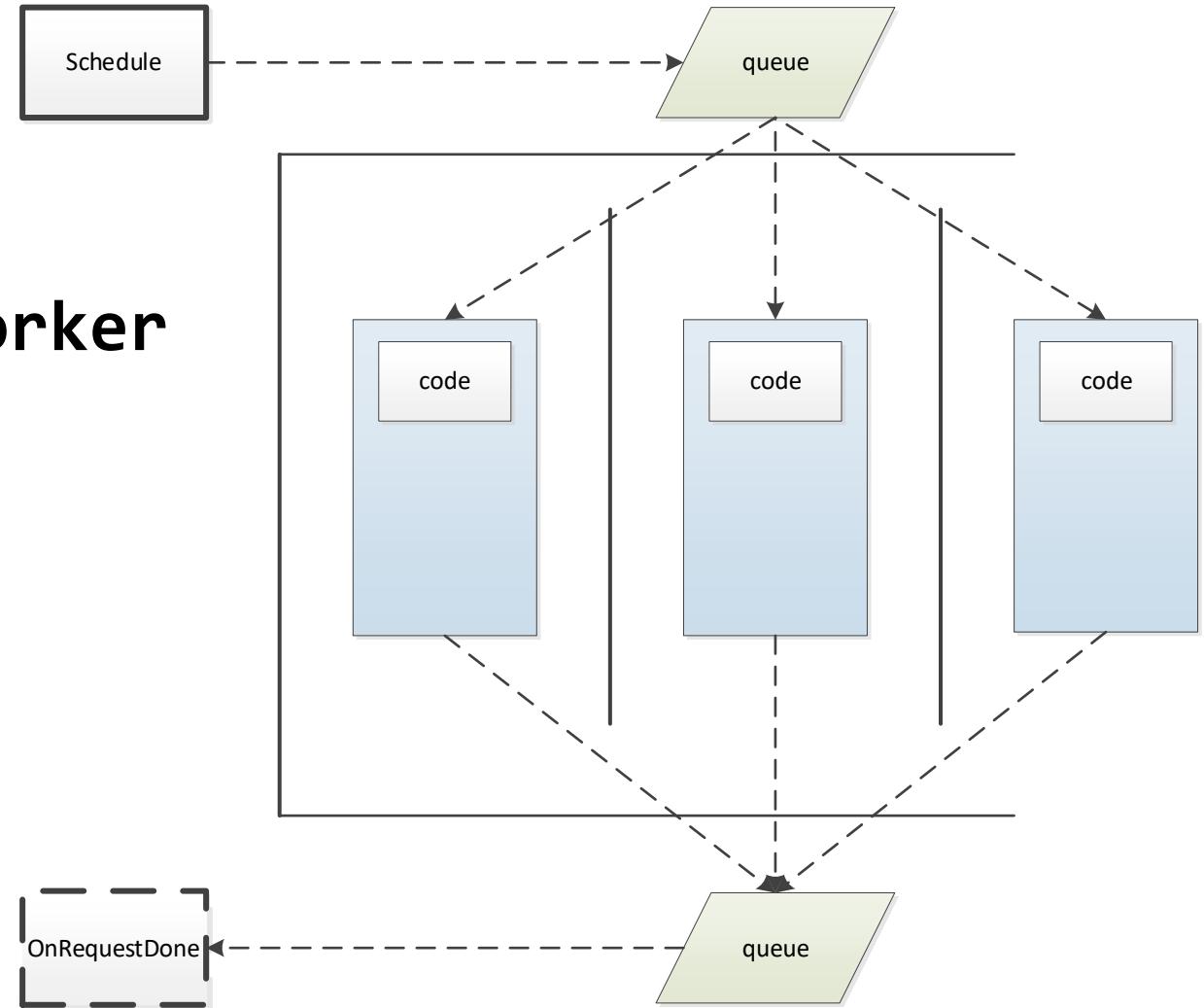
```
Parallel.ParallelTask  
.Execute(code);
```



Background Worker

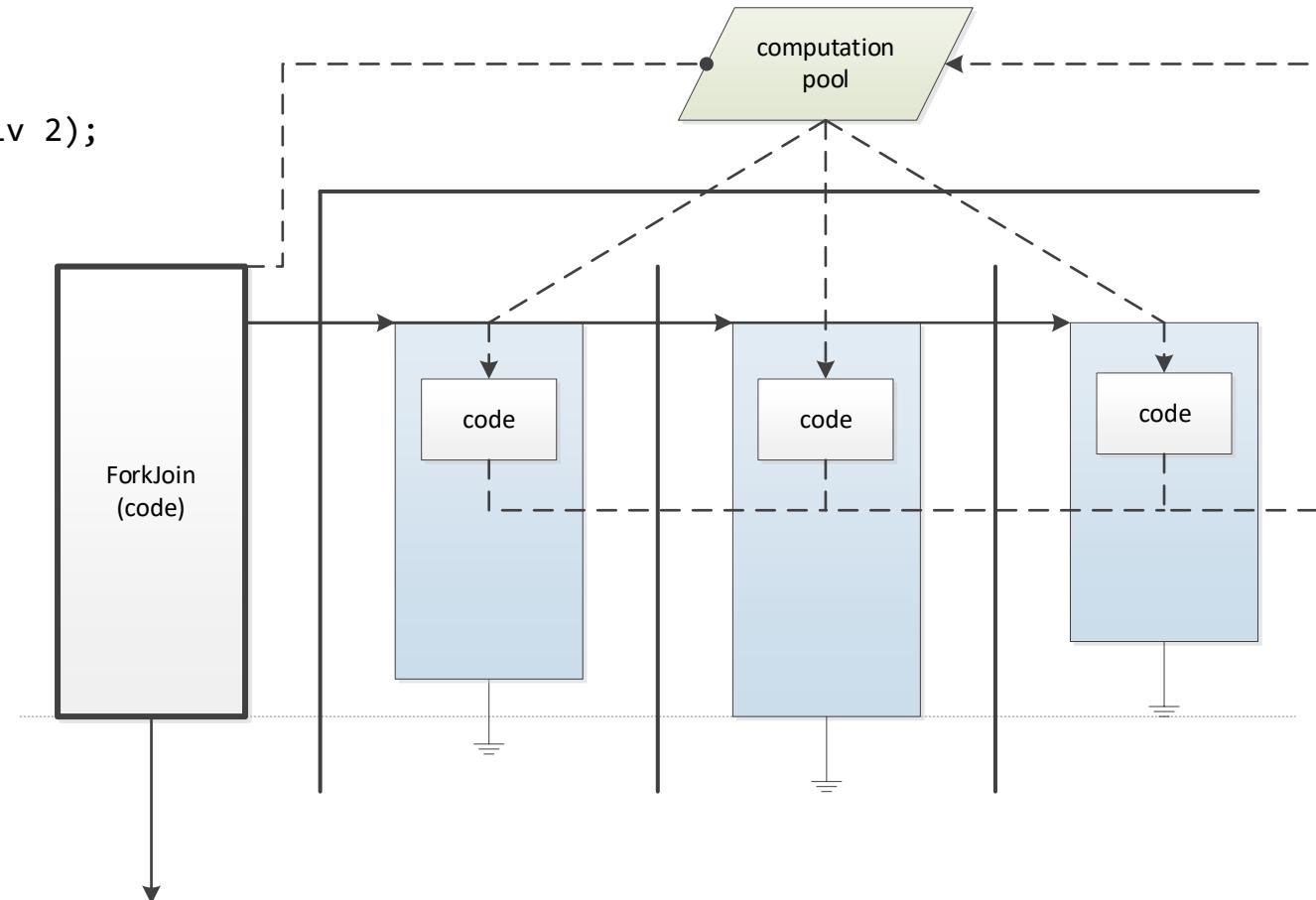
```
bw := Parallel.BackgroundWorker  
  .Execute(code1)  
  .OnRequestDone(code2);
```

```
bw.Schedule(workItem);
```



Fork/Join

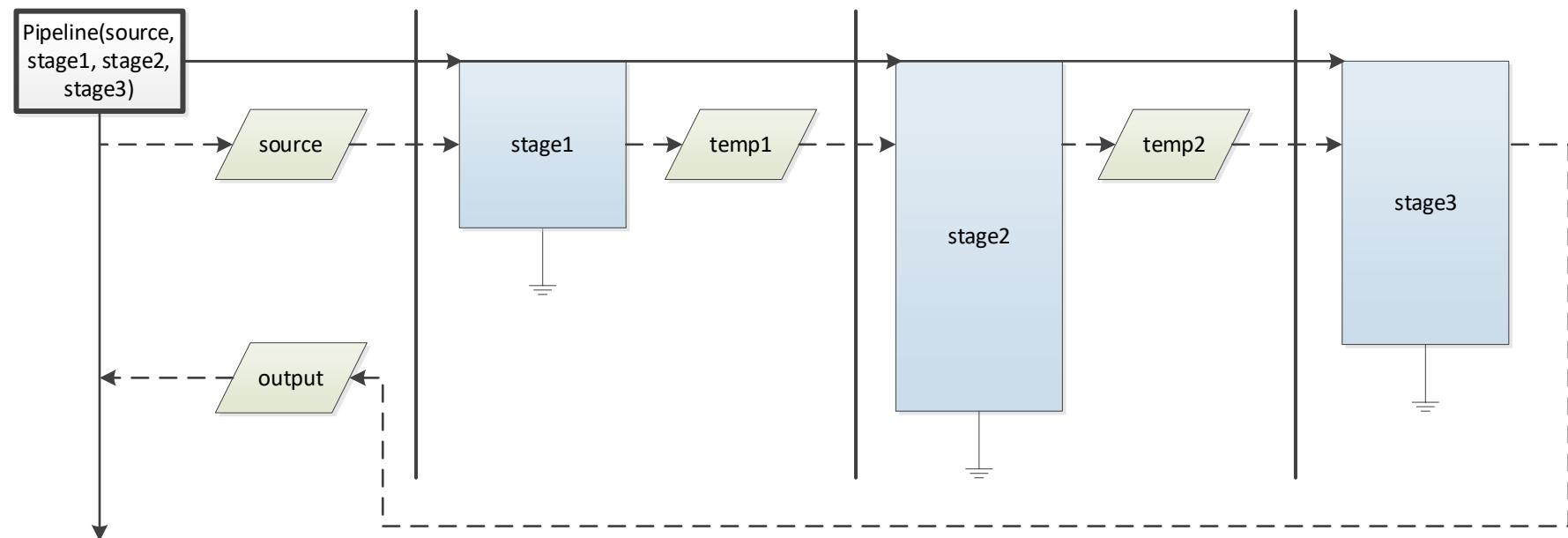
```
procedure TParallelSorter.QuickSort(left, right: integer);
begin
  ...
  pivotIndex := Partition(left, right, (left + right) div 2);
  sortLeft := FForkJoin.Compute(
    procedure begin
      QuickSort(left, pivotIndex - 1);
    end);
  sortRight := FForkJoin.Compute(
    procedure begin
      QuickSort(pivotIndex + 1, right);
    end);
  sortLeft.Await;
  sortRight.Await;
  ...
end;
```



Pipeline

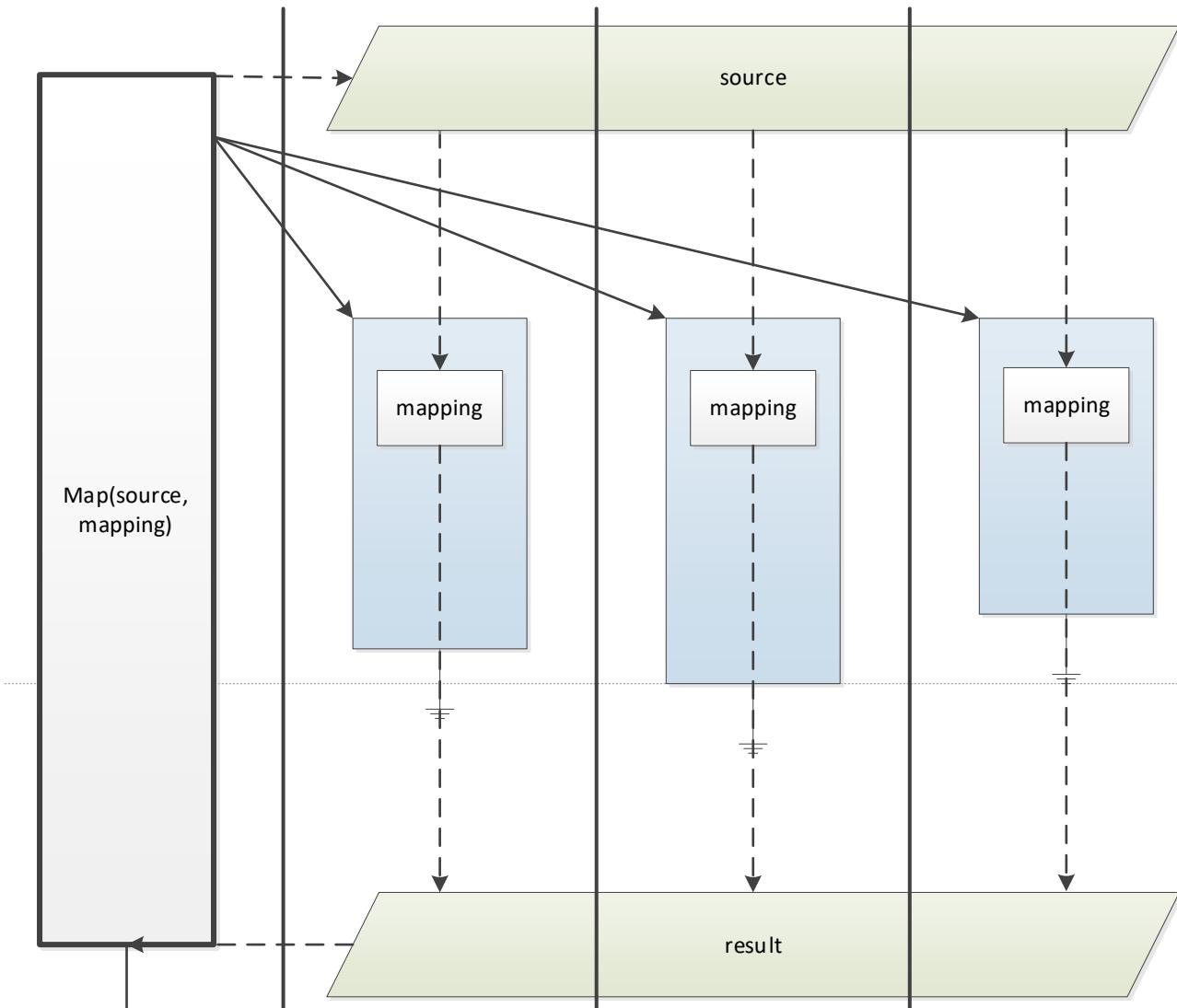
`Parallel.Pipeline`

`.Stage(code1).Stage(code2).Stage(code3).Run;`



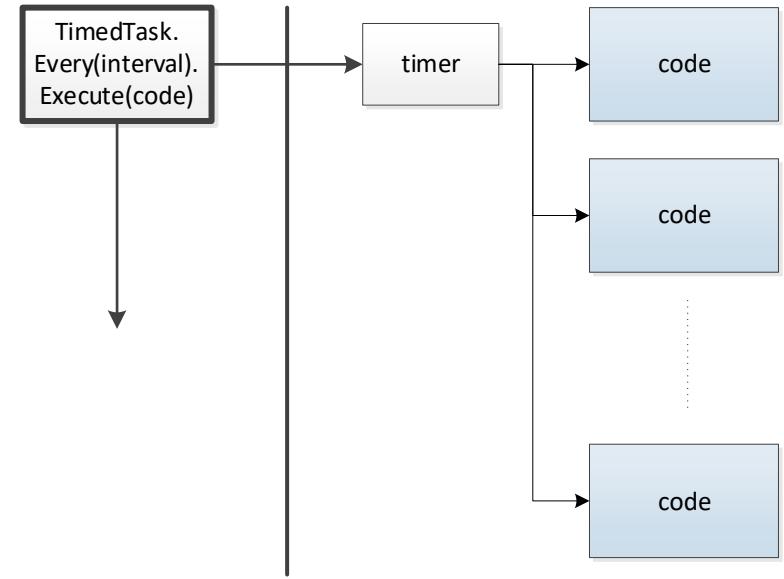
Map/Reduce

```
Parallel.Map<T1, T2>  
(TArray<T1>, mapper)  
: TArray<T2>
```



Timed Task

```
Parallel.TimedTask  
.Every(interval).Execute(code);
```



Data Structures

Also useful in TThread / ITask

Records/Objects

- *OtlCommon*
- TOmniValue
- TOmniWaitableValue
- TOmniCounter

Fixed-size collections

- *OtlContainers*
- TOmniBoundedStack
- TOmniBoundedQueue

Unbound collection

- *OtlCollections*
- TOmniBlockingCollection

Synchronization

- *OtISync*
- IOmniCancellationToken
- Atomic<T>
- Locked<T>



Q&A