ITDevCon

Delphi European Conference

Multithreading Made Simple with OmniThreadLibrary

Primož Gabrijelčič

October 27/28 2011 VERONA

bit Time software

Introduction

OmniThreadLibrary is …

- … VCL for multithreading
  - Simplifies programming tasks
  - Componentizes solutions
  - Allows access to the bare metal
- … trying to make multithreading possible for mere mortals
- … providing *well-tested* components packed in *reusable* classes with *high-level* parallel programming support
- … parallel programming today!

Project status

- OpenBSD license
- Actively developed
  - 1008 commits
- Actively used
  - 2.0: 2710 downloads [in 7 months]
  - 2.1: 1187 downloads [in 3 months]
  - 2.2: current release, XE2 support
- Delphi 2007 and above; currently Win32 only

Installation

- Download last installation from the Google Code or checkout the SVN repository
  - code.google.com/p/omnithreadlibrary/
- Add installation folder and its *src* subfolder to the project search path or Win32 library path
- Add the *OtlParallel* unit to the *uses* list
- That's all folks!

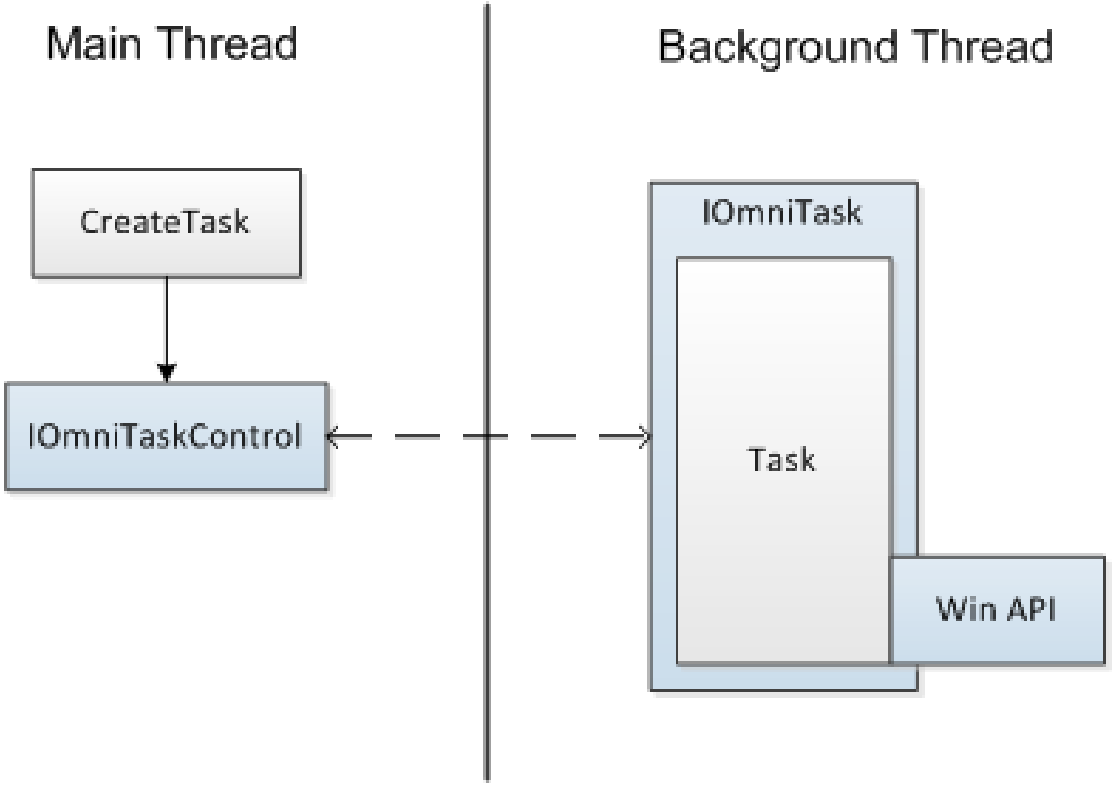
Future

- Win64
- FMX
- OS/X
- iOS?

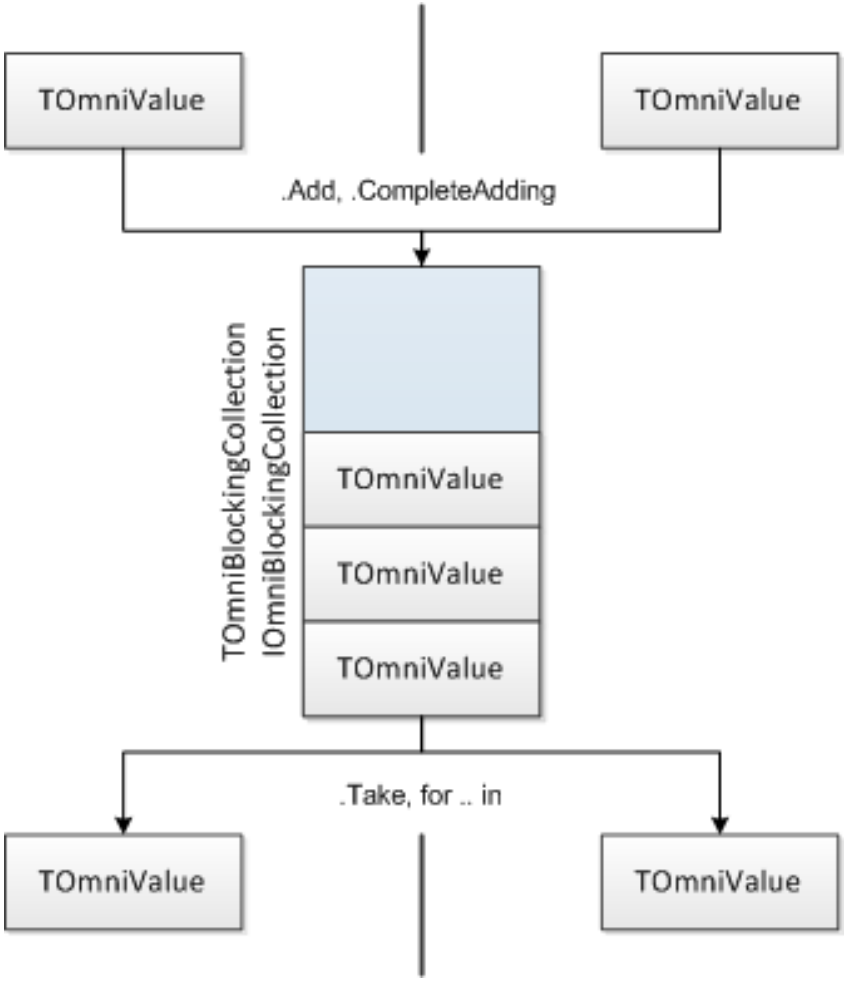
OmniThreadLibrary basics

ITDevCon

Task vs. thread



Communication



High-level multithreading

ITDevCon

“New programmers are drawn to multithreading like moths to flame, with similar results.”

-Danny Thorpe

Why high-level approach?

- Designing parallel solutions is hard
- Writing multithreaded code is hard
- Testing multicore applications is hard
- Debugging multithreading code is pure insanity
- Debugging multithreading code is hard

High-level multithreading

- Async
  - start background task and continue
- Future
  - start background calculation and retrieve the result
- Join
  - start multiple background tasks and wait
- ParallelTask
  - start multiple copies of one task and wait

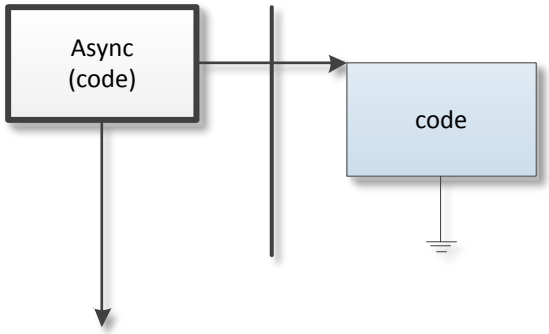
High-level multithreading

- ForEach
  - parallel iteration over many different containers
- Pipeline
  - run a multistage process
- Fork/Join
  - divide and conquer, in parallel

- Delphi 2009 required

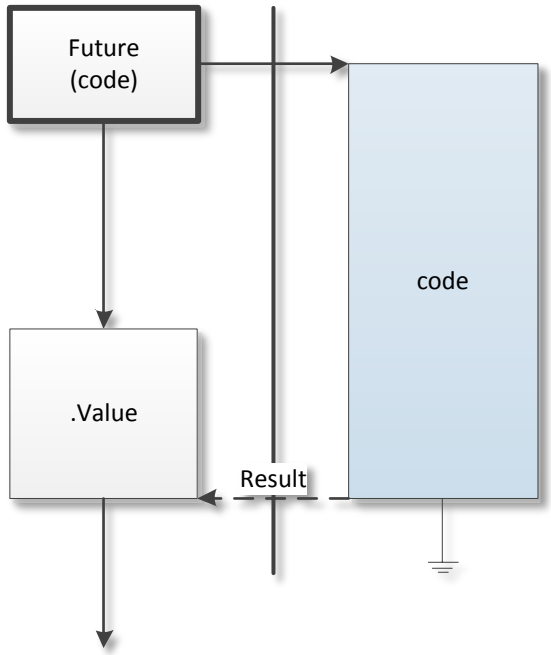
Async

- `Parallel.Async(code)`



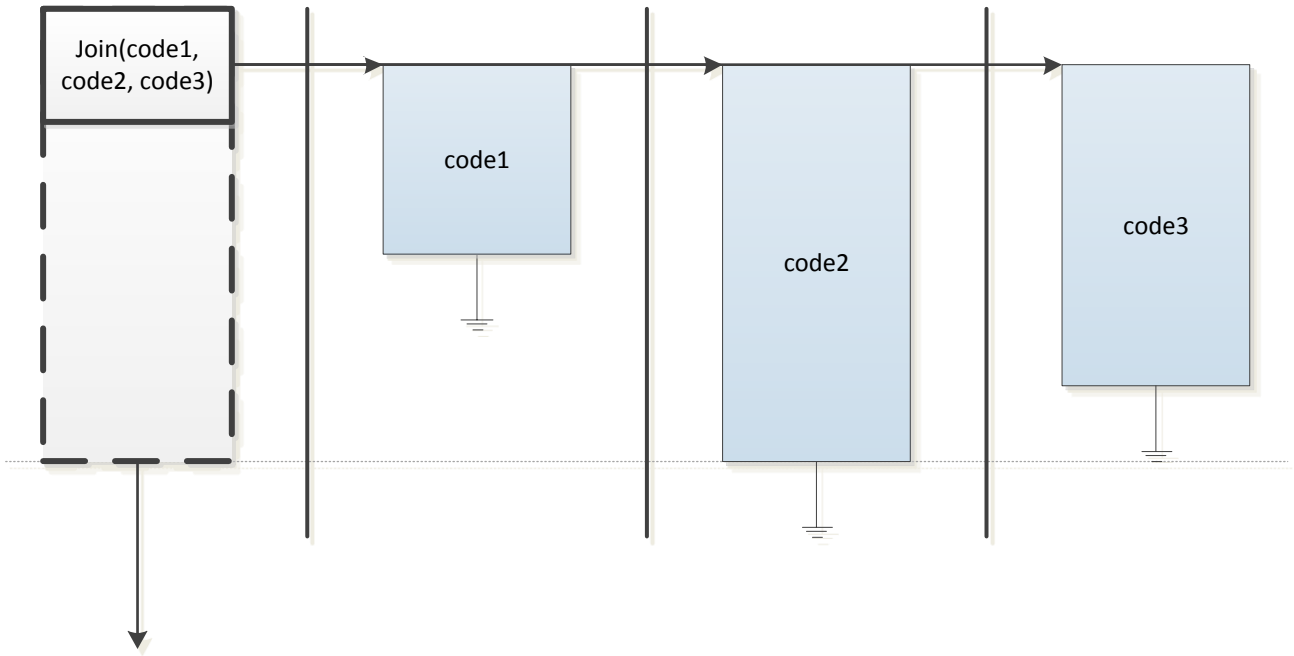
Future

- `Future:=Parallel.Future<type>.(calculation);`
- `Query Future.Value;`

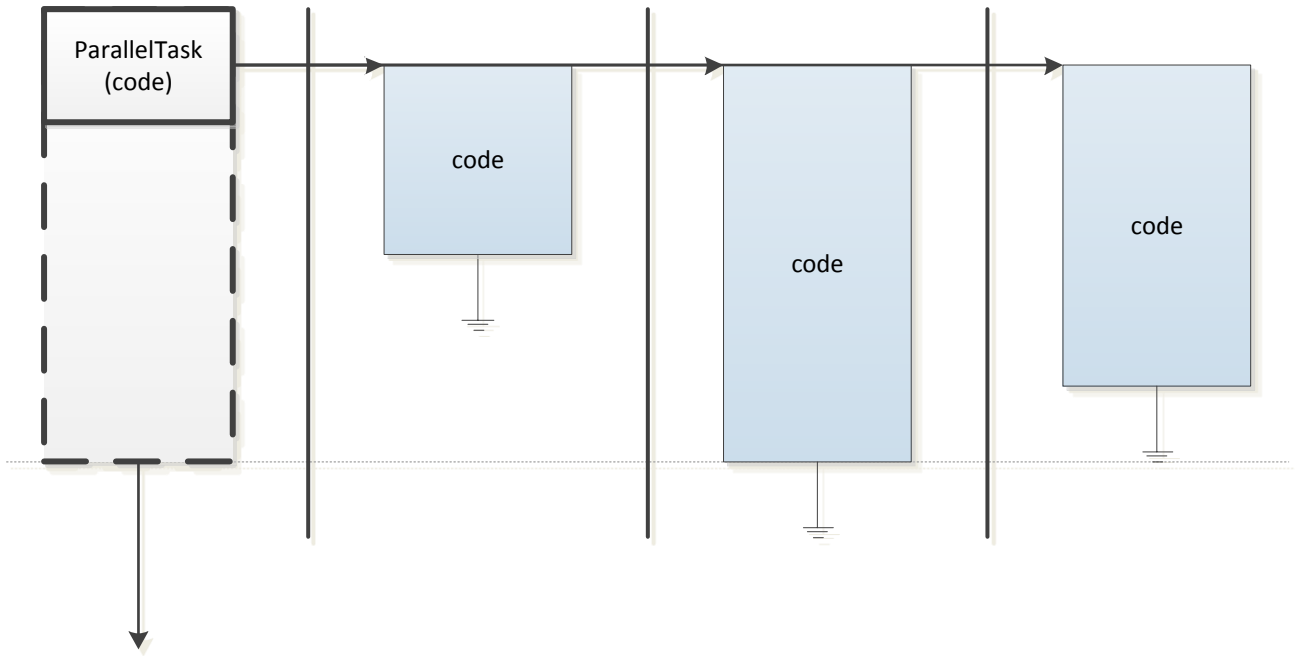


Join

- `Parallel.Join([`*`task1, task2, task3, … taskN`*`]).Execute`

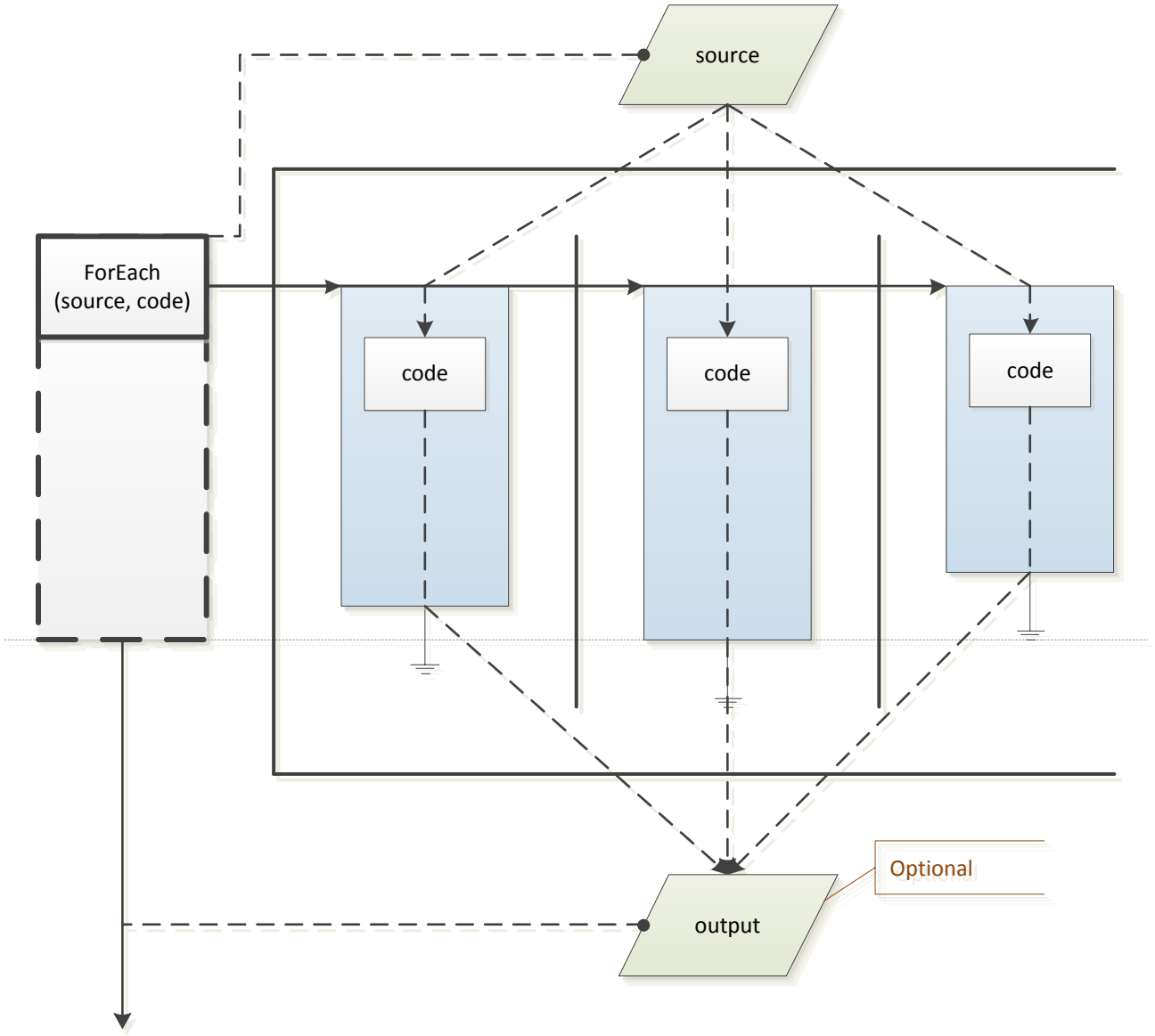


- `Parallel.ParallelTask.Execute (code)`

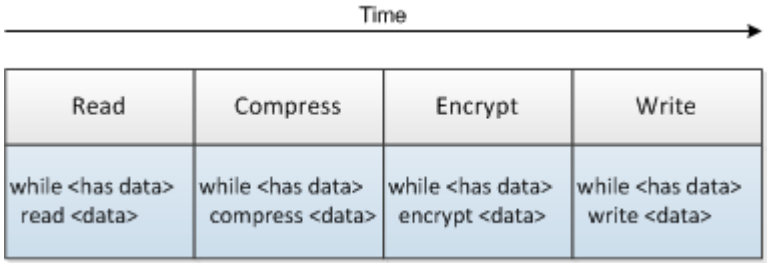


- ```
  Parallel.ForEach(from, to).Execute(
    procedure (const value: integer);
    begin
      //…
    end)
  ```
- ```
  Parallel.ForEach(source).Execute(
    procedure (const value: TOmniValue);
    begin
      //…
    end)
  ```

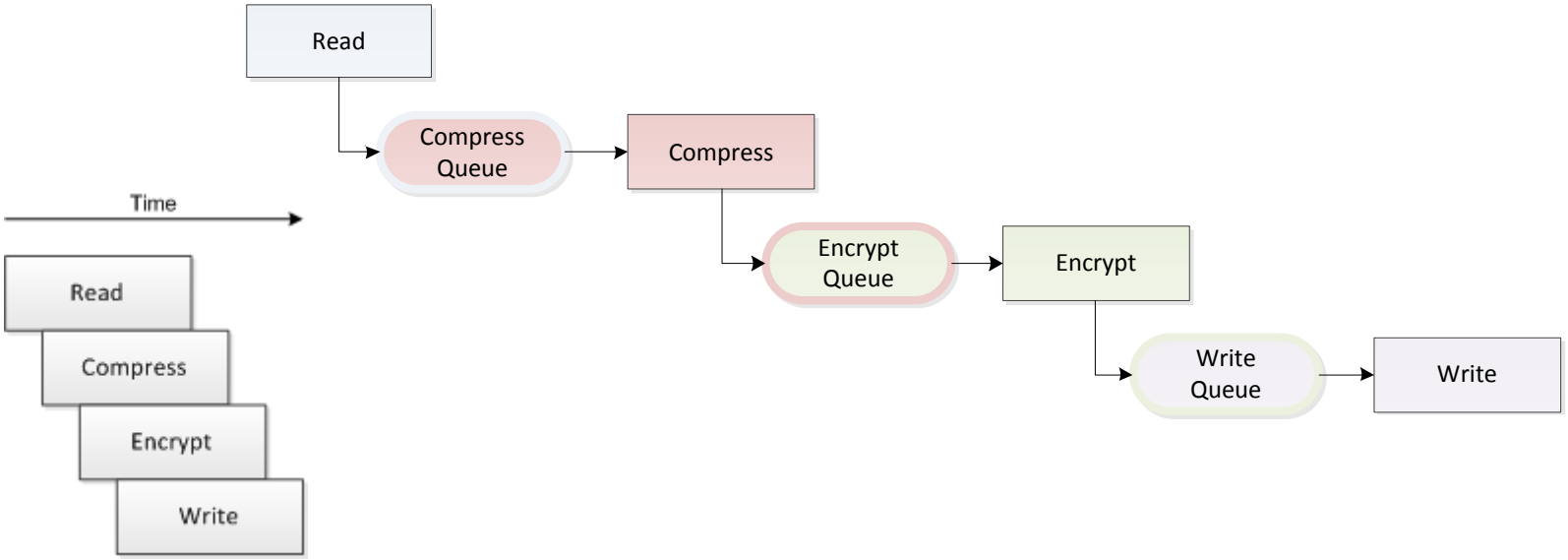
ParallelFor



Pipeline example

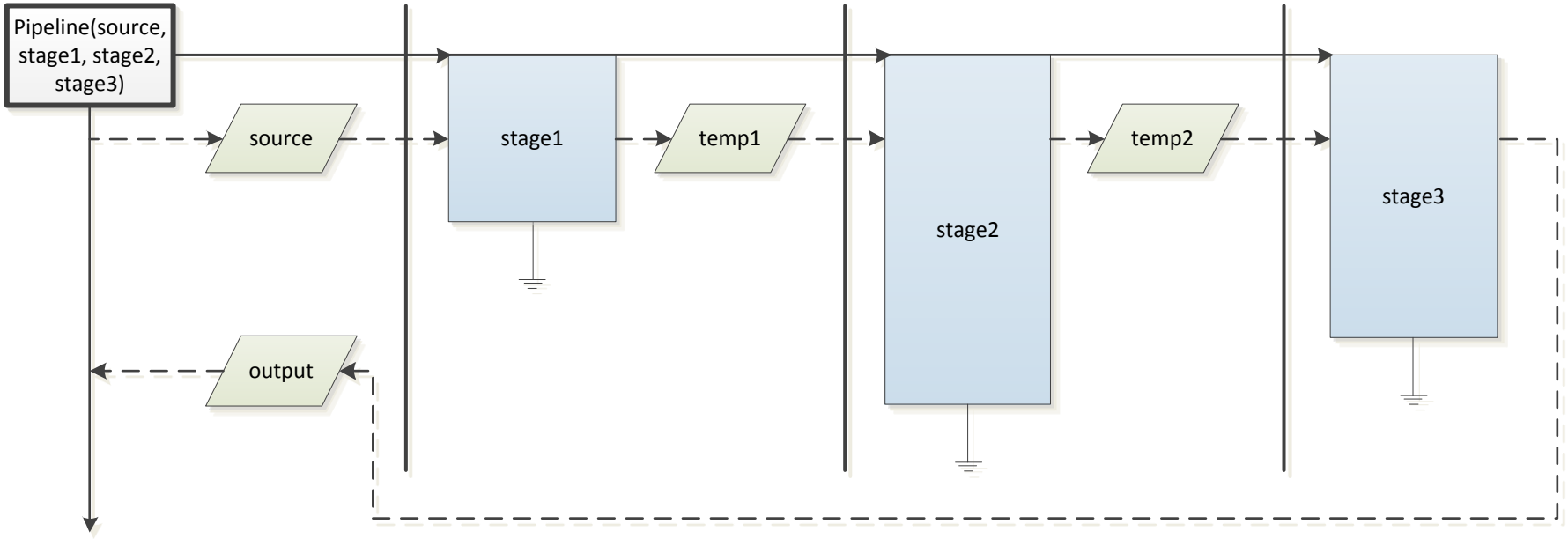



```
while <has data>
   read <data>
   insert <data> into <compress queue>

while read <compress queue>
   compress <data>
   insert <data> into <encrypt queue>

while read <encrypt queue>
   encrypt <data>
   insert <data> into <write queue>

while read <write queue>
   write <data>
```

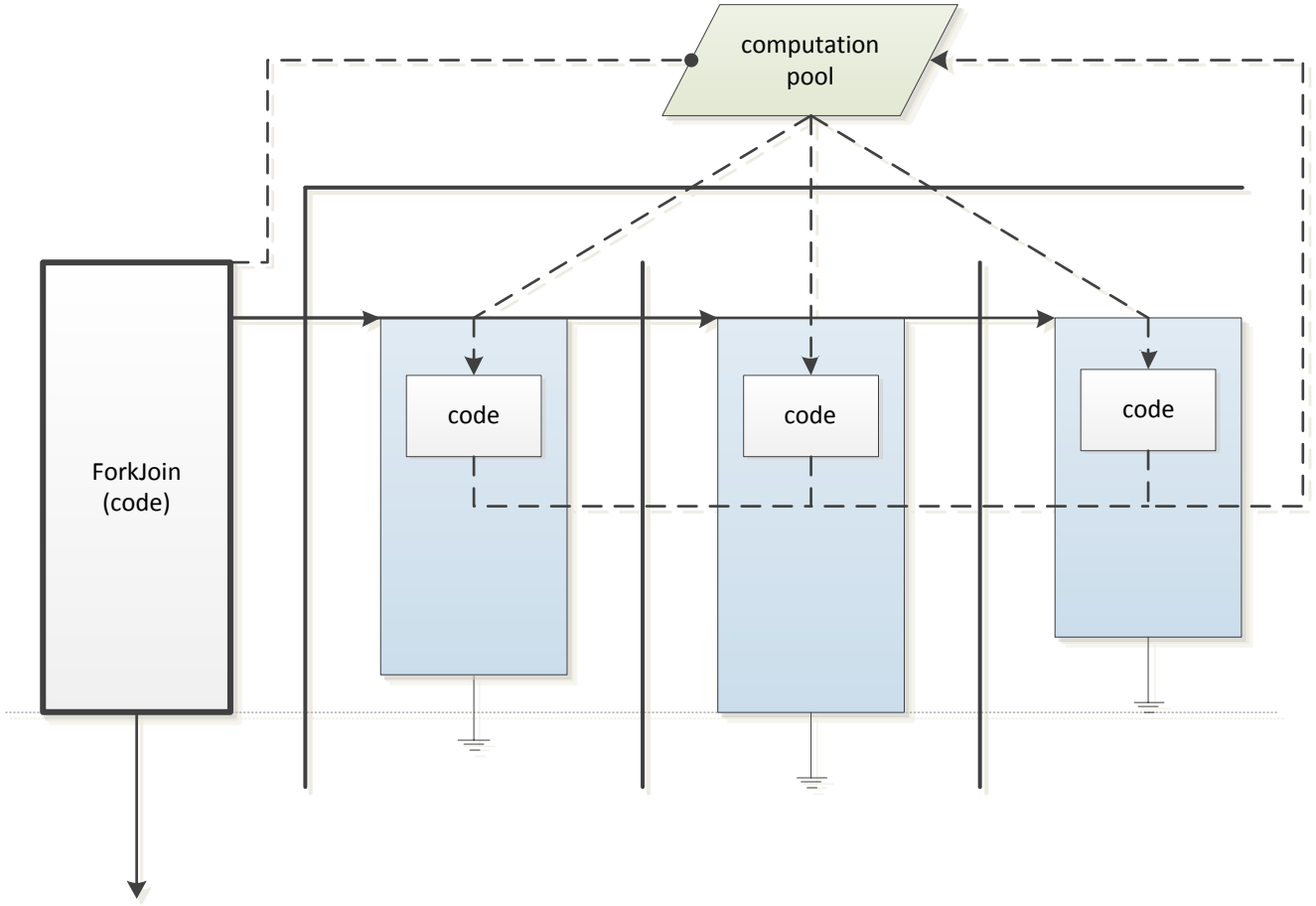


Pipeline

- Parallel.Pipeline([stage1, stage2, stage3]).Run



- Divide and conquer



Multithreading is hard?

Why high-level approach?

- Designing parallel solutions is hard
- Writing multithreaded code is hard
- Testing multicore applications is hard
- Debugging multithreading code is pure insanity
- Debugging multithreading code is hard

Questions?

ITDevCon