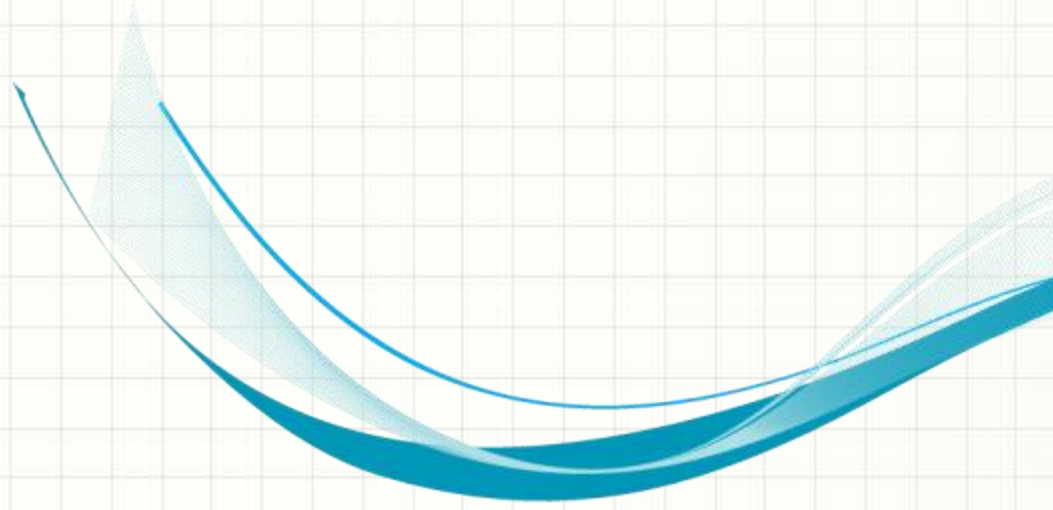




GETTING FULL SPEED WITH DELPHI

[WHY SINGLE-THREADING IS NOT ENOUGH?]

Primož Gabrijelčič
primoz@gabrijelcic.org
www.thedelphigeek.com
otl.17slon.com



Multithreading

A decorative blue wavy line with a lighter blue shadow, curving from the top left towards the bottom left of the slide.

What?

- The art of doing multiple things at the same time



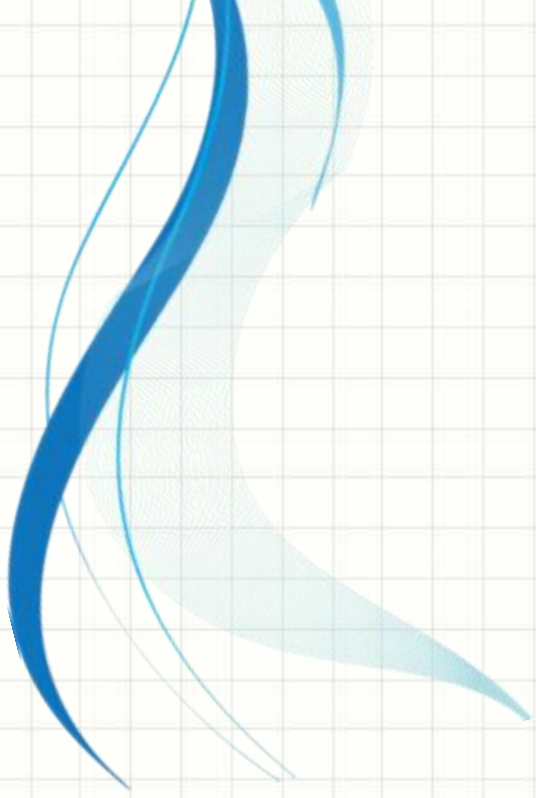
Why?

- The end of free lunch



How?

- OmniThreadLibrary



When?

- Rarely



WHAT?

Threading


- A thread is a line of execution through a program
 - There is always one thread
- Multitasking (and multithreading)
 - Cooperative
 - Preemptive
 - Time slicing
 - Parallel execution

Processes vs. Threads

- Pros
 - Processes are isolated – data protection is simple
- Cons
 - Processes are isolated – data sharing is simple
 - Processes are *heavy*, threads are *light*

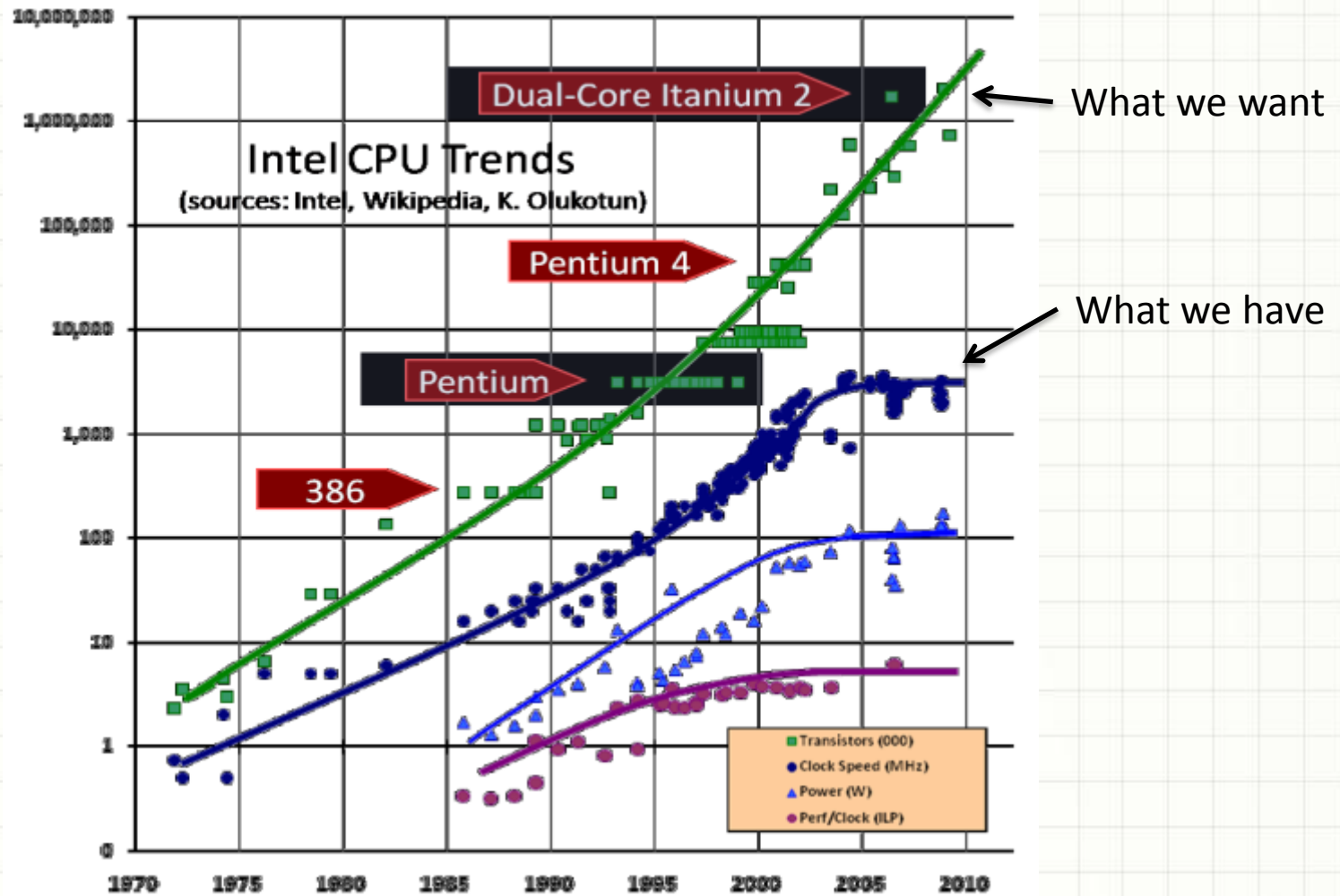
Problems

- Data sharing
 - Messaging
 - Synchronization
- Synchronization causes
 - Race conditions
 - Deadlocking
 - Livelocking
- Slowdown



WHY?

The End of Free Lunch





How?

Four paths to multithreading

- The Delphi Way
 - TMyThread = **class**(TThread)
- The Windows Way
 - FHandle := BeginThread(nil, 0, @ThreadProc, Pointer(Self), 0, FThreadID);
- The Lightweight Way (AsyncCalls)
 - TAsyncCalls.Invoke(**procedure begin**
DoTheCalculation;
end);
- OmniThreadLibrary

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

Project Status

- OpenBSD license
- Actively developed
 - 886 commits [code.google.com/p/omnithreadlibrary/]
- Actively used
 - 2.0: 1206 downloads
 - 1.05: 2028 downloads

[March 1st, 2011]
- Almost no documentation





How?

High level multithreading

- Join
 - Futures
 - Pipelines
 - Fork/Join
 - Parallel *for*
-
- Delphi 2009 required

Join

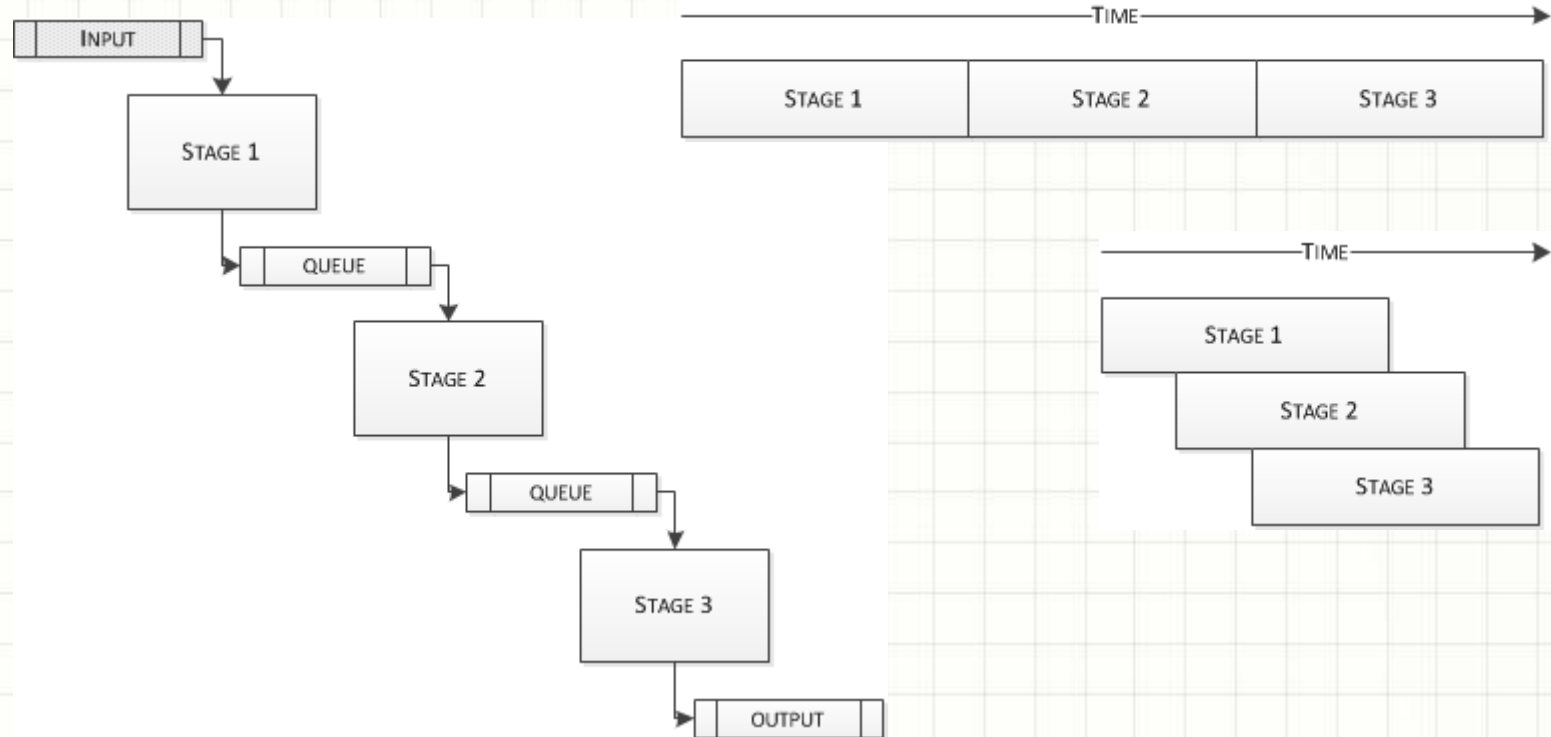
- Divide and Wait
 - Start multiple background calculations
 - Wait for all to complete
 - No result is returned (directly)
- Two basic forms
 - `Join(task1, task2);`
 - `Join([task1, task2, task3, ...
taskN]);`

Future

- Wikipedia
 - *“They (futures) describe an object that acts as a proxy for a result that is initially not known, usually because the computation of its value has not yet completed.”*
 - Start background calculation, wait on result.
- How to use?
 - `Future:=TOmniFuture<type>.Create(
 calculation);`
 - `Query Future.Value;`

Pipeline

- Multistage process



Pipelines

var

pipeOut: IOmniBlockingCollection;

```
pipeOut := Parallel.Pipeline  
    .Stage(StageGenerate)  
    .Stage(StageMult2)  
    .Stage(StageSum)  
    .Run;
```

Fork/Join

- Divide and conquer
 - Execute multiple tasks
 - Wait for them to terminate
 - Collect results
 - Proceed
- Subtasks may spawn new subtasks

Fork/Join

```
max1 := forkJoin.Compute(  
  function: integer begin  
    Result := ...  
  end);
```

```
max2 := forkJoin.Compute(  
  function: integer begin  
    Result := ...  
  end);
```

```
Result := Max(max1.Value, max2.Value);
```

Parallel For

Parallel

```
.ForEach(1, CMaxSGPrimeTest)
.Execute(
    procedure (const value: integer)
    begin
        if IsPrime(value) then
            numPrimes.Increment;
    end);
```

Messaging

- TOmniMessageQueue
- TOmniQueue
 - Dynamically allocated, $O(1)$ enqueue and dequeue, threadsafe, microlocking queue
- TOmniBlockingCollection
- TOmniValue

Tasks vs. Threads

- *Task* is part of code that has to be executed
- *Thread* is the execution environment
- You take care of the task,
OTL takes care of the thread

Task Execution

- `CreateTask(task_procedure)`
- `CreateTask(task_method)`
- `CreateTask(TOmniWorker_object)`
- `CreateTask(anonymous_procedure)`

Thread Pool

- Starting up a thread takes time
- Thread pool keeps threads alive and waits for tasks
- Automatic thread startup/shutdown
- User code executed at thread creation
 - Connection pool
- `.Run` \Rightarrow `.Schedule`



WHEN?



Danger!

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

-Danny Thorpe

When To Use

- Slow background process
- Background communication
- Executing synchronous API
- Multicore data processing
- Multiple clients

Keep in mind

- Don't parallelize everything
- Don't create thousands of threads
- Rethink the algorithm
- Prove the improvements
- Test, test and test

Be Afraid

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





QUESTIONS?