Configuring, Building, and Iterating

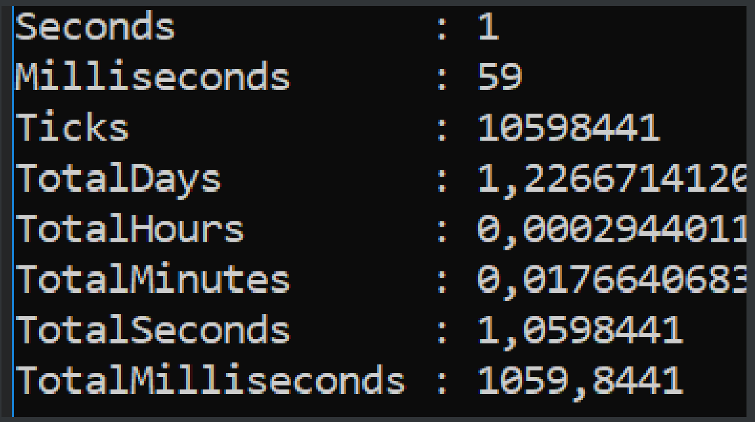
Dub x Redub x Reggae

Features	Dub	Redub	Reggae
Parallel Compilation	✗	✓	✓
Baked Commands	✗	✗	✓
Global Cache	✓	✓	✗
Configuration Freedom	✗	✗	✓
Focus on User Experience	✓	✓	✗
Backed by D Code	✓	✓	✗
No External Dependencies	✓	✓	✗

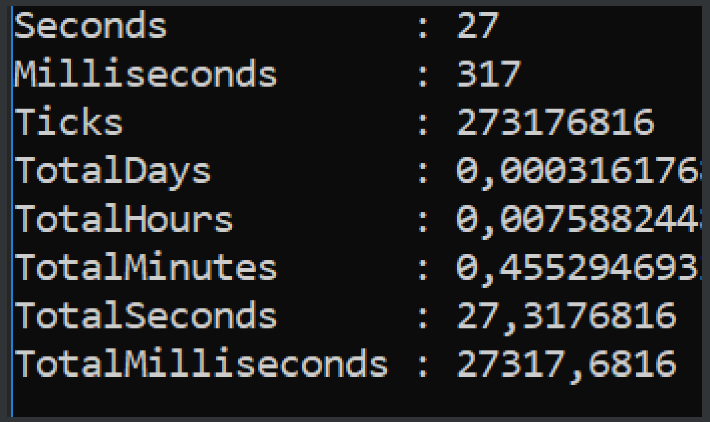
Supported Projects

Dub’s first run

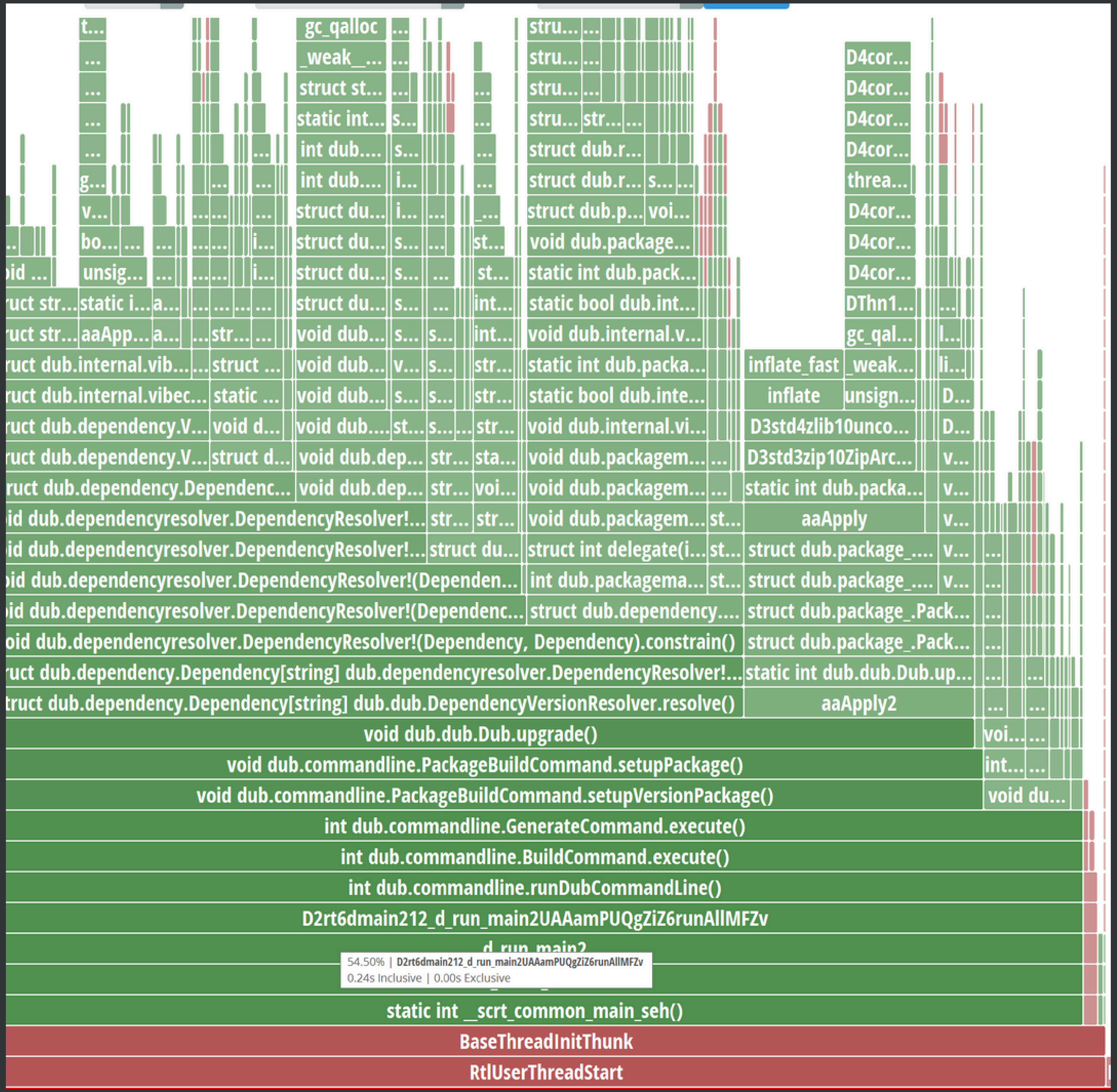
- Dub’s trial of creating dub.selections.json is painfully slow
- On first run, a full rebuild takes 27 seconds, against 1 second from redub
- Really hard to understand what’s going on



```
Seconds           : 1
Milliseconds      : 59
Ticks             : 10598441
TotalDays         : 1,2266714126
TotalHours        : 0,0002944011
TotalMinutes      : 0,0176640683
TotalSeconds      : 1,0598441
TotalMilliseconds : 1059,8441
```



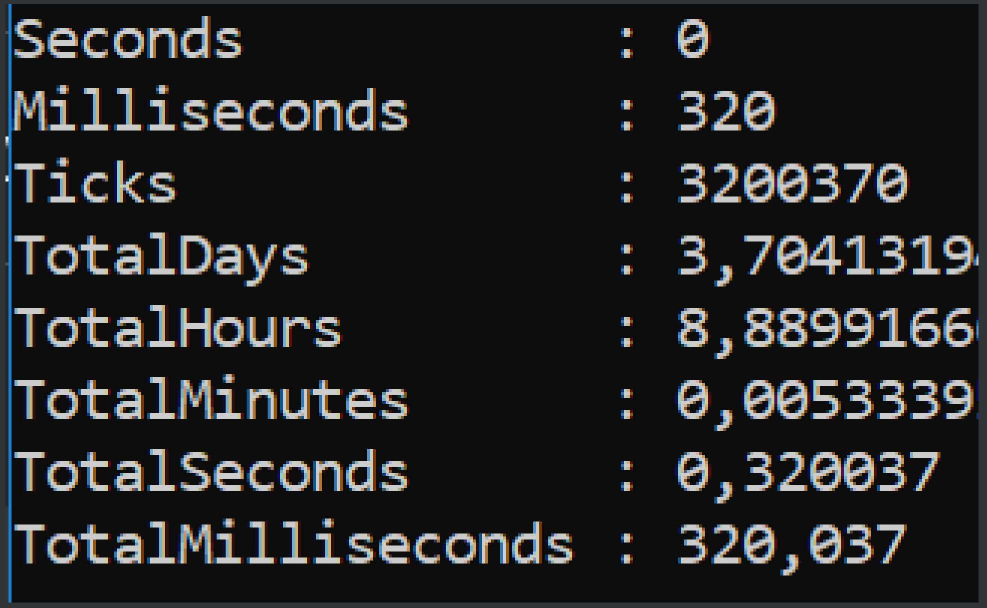
```
Seconds           : 27
Milliseconds      : 317
Ticks             : 273176816
TotalDays         : 0,000316176
TotalHours        : 0,007588244
TotalMinutes      : 0,455294693
TotalSeconds      : 27,3176816
TotalMilliseconds : 27317,6816
```



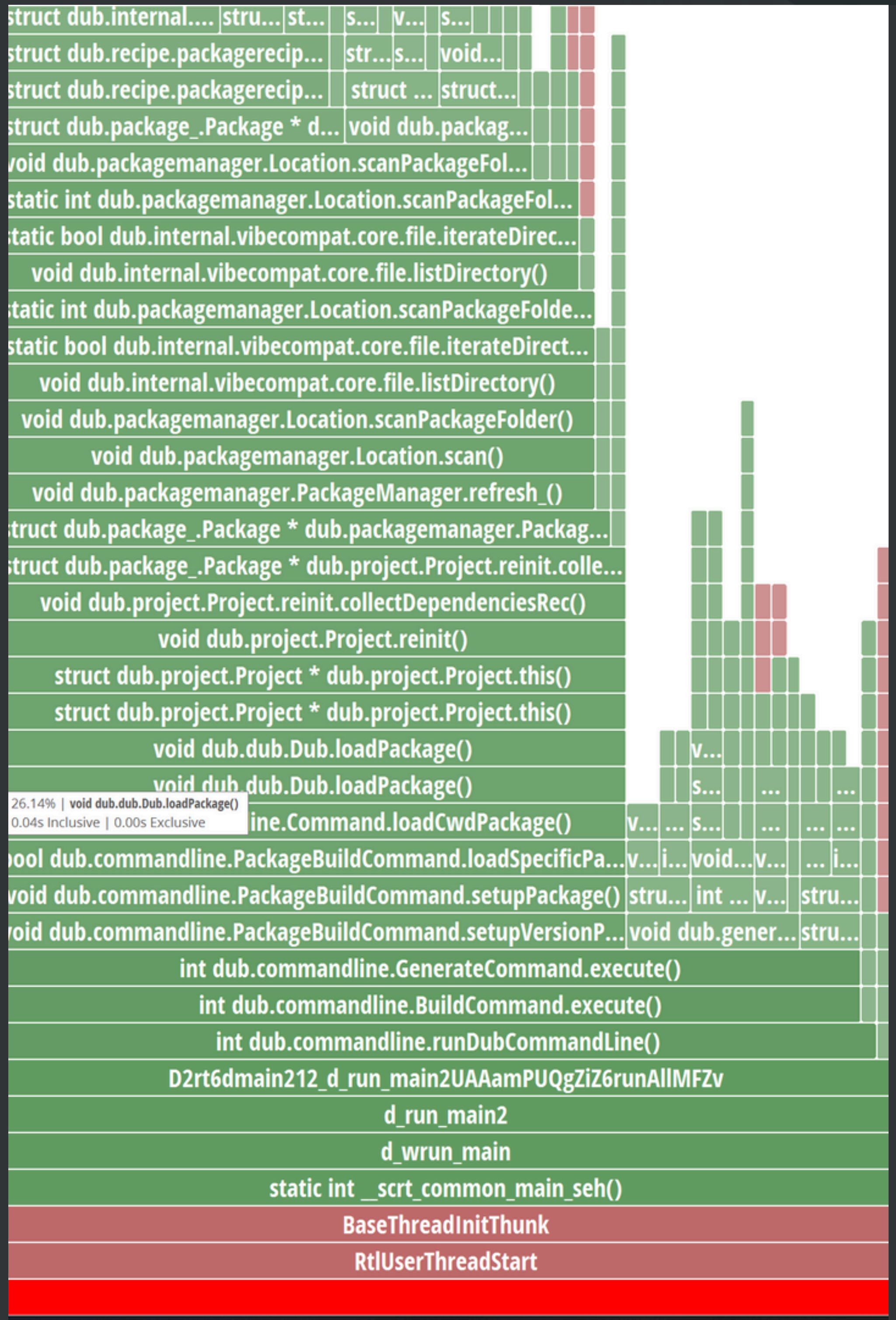
Debugging Dub

- Very slow iteration, making it way harder to test
- On Amd uProf, dub.loadPackage() costs 26% out of 37% of the processing time [70% of the time is spent on package loading]
- dyaml the main culprit as it creates a very deep call stack.

Time spent on up-to-date build



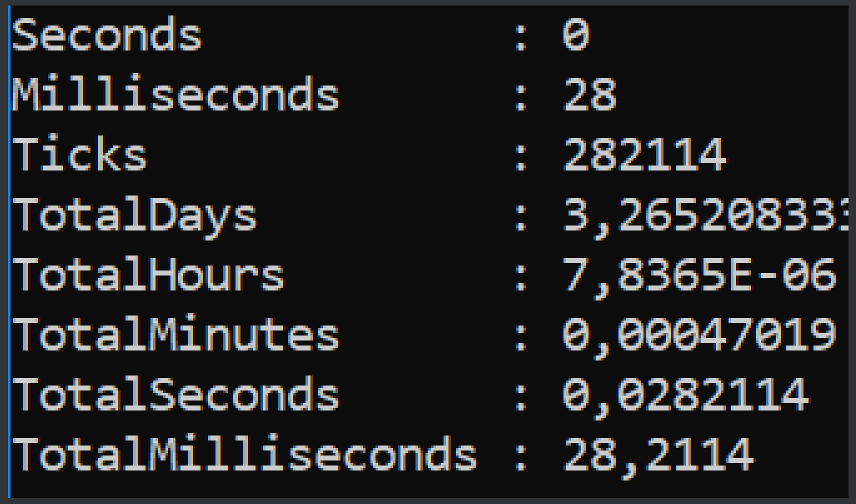
```
Seconds           : 0
Milliseconds      : 320
Ticks             : 3200370
TotalDays         : 3,7041319
TotalHours        : 8,8899166
TotalMinutes      : 0,0053339
TotalSeconds      : 0,320037
TotalMilliseconds : 320,037
```



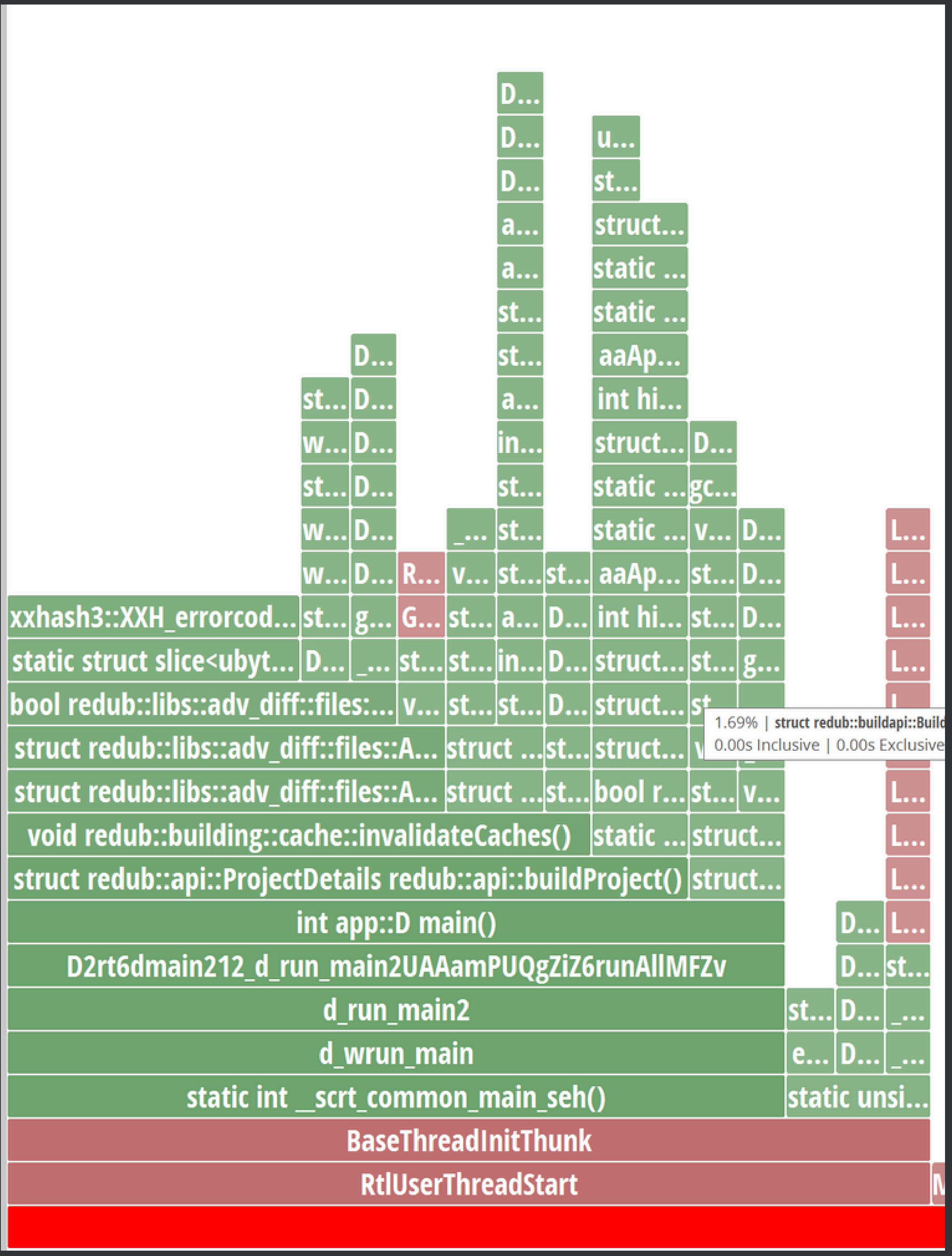
Comparing to Redub

- Very shallow call stack
- On Amd uProf, dub.loadPackage() costs 1.69% of the processing time is spent on package loading
- It has enough time to even content hash check
- The complete operation takes 8% of the time dub takes

Time spent on up-to-date-build



```
Seconds           : 0
Milliseconds      : 28
Ticks             : 282114
TotalDays         : 3,26520833
TotalHours        : 7,8365E-06
TotalMinutes      : 0,00047019
TotalSeconds      : 0,0282114
TotalMilliseconds : 28,2114
```

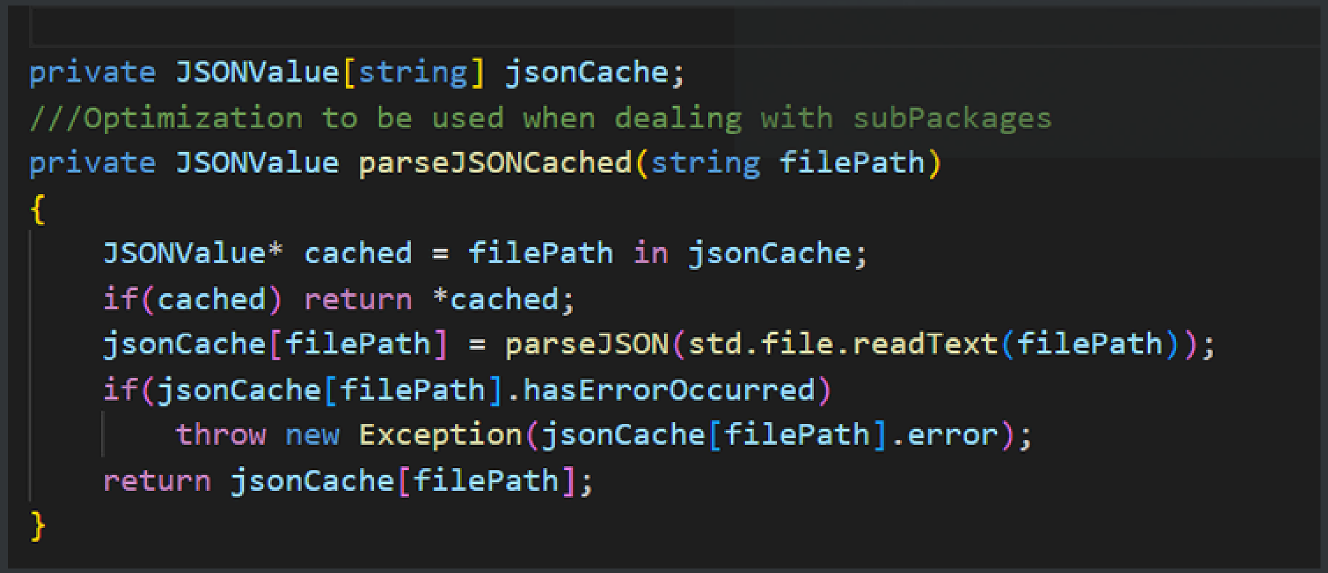


Comparing to Redub

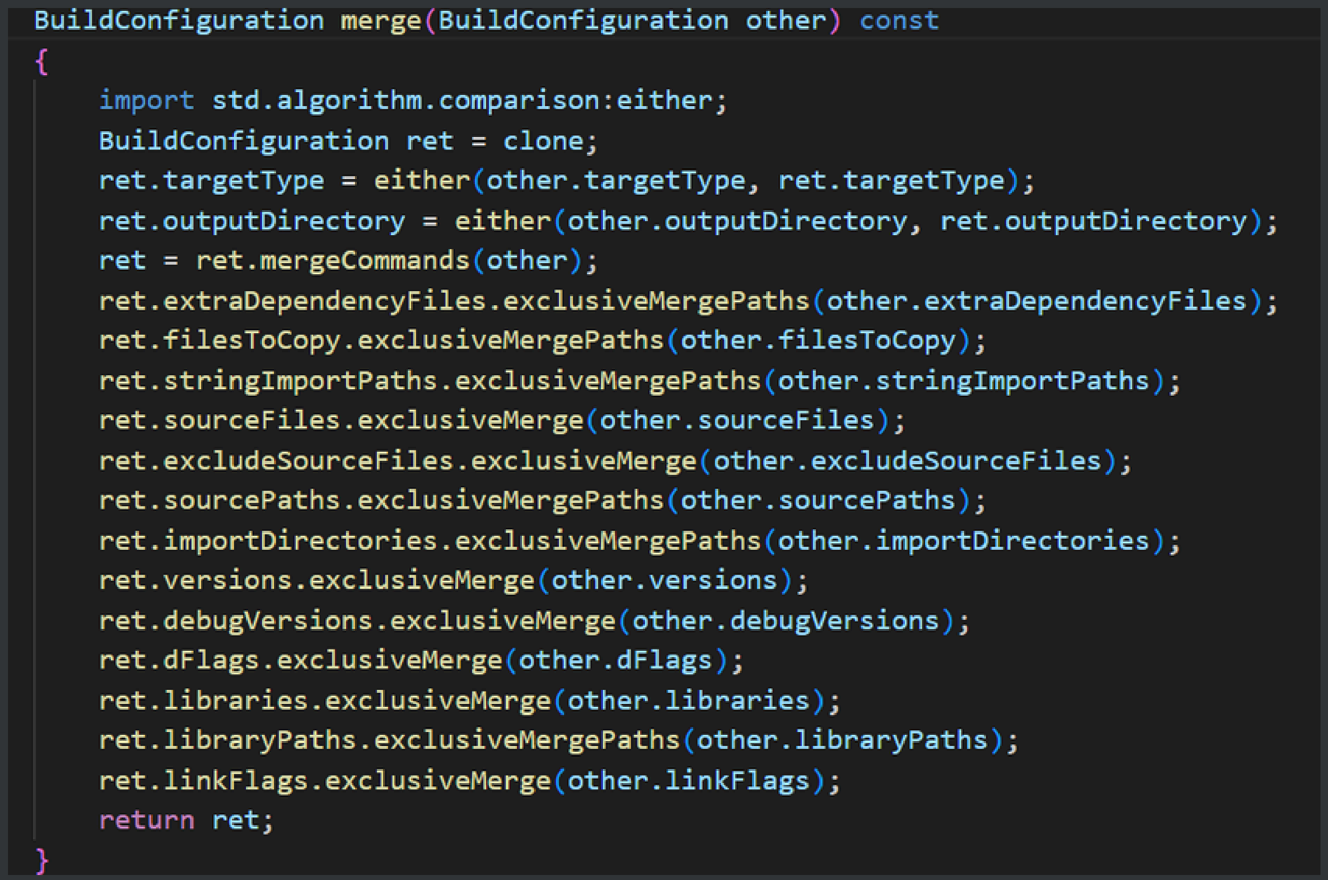
- Redub does not use a dub.selections file
- Plenty of projects does not run without a version lock file -- SemVer is not being used correctly in projects, though adapting to create a lock files is easy to do, I don’t think it is the best way to solve the problem.
- Redub uses version specification for matching the current files in the developer’s environment. If none match, it delegates a dub fetch

How Redub improved

- **Avoid state mutation**, unless used for caching operations
- **Cache JSON** files
- **Better JSON Algorithm**
- **No Intermediate Representation:** single file type for consistency and lower workload
- **Lazy package parsing**
- **Functional Programming:** So, one does not need to read too much code to understand it

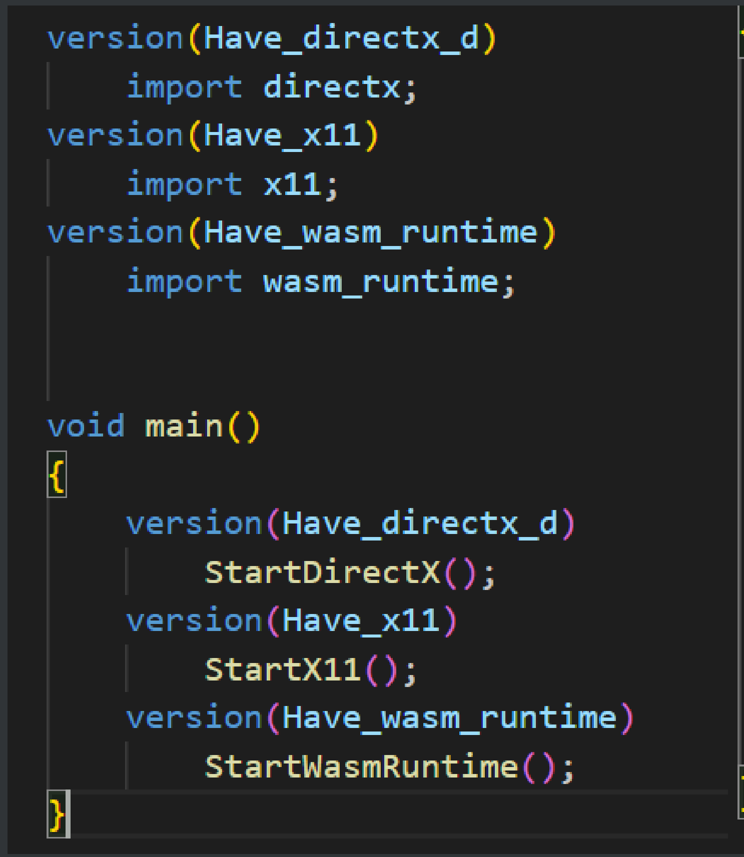


```d
private JSONValue[string] jsonCache;
///Optimization to be used when dealing with subPackages
private JSONValue parseJSONCached(string filePath)
{
    JSONValue* cached = filePath in jsonCache;
    if(cached) return *cached;
    jsonCache[filePath] = parseJSON(std.file.readText(filePath));
    if(jsonCache[filePath].hasErrorOccurred)
        throw new Exception(jsonCache[filePath].error);
    return jsonCache[filePath];
}
```

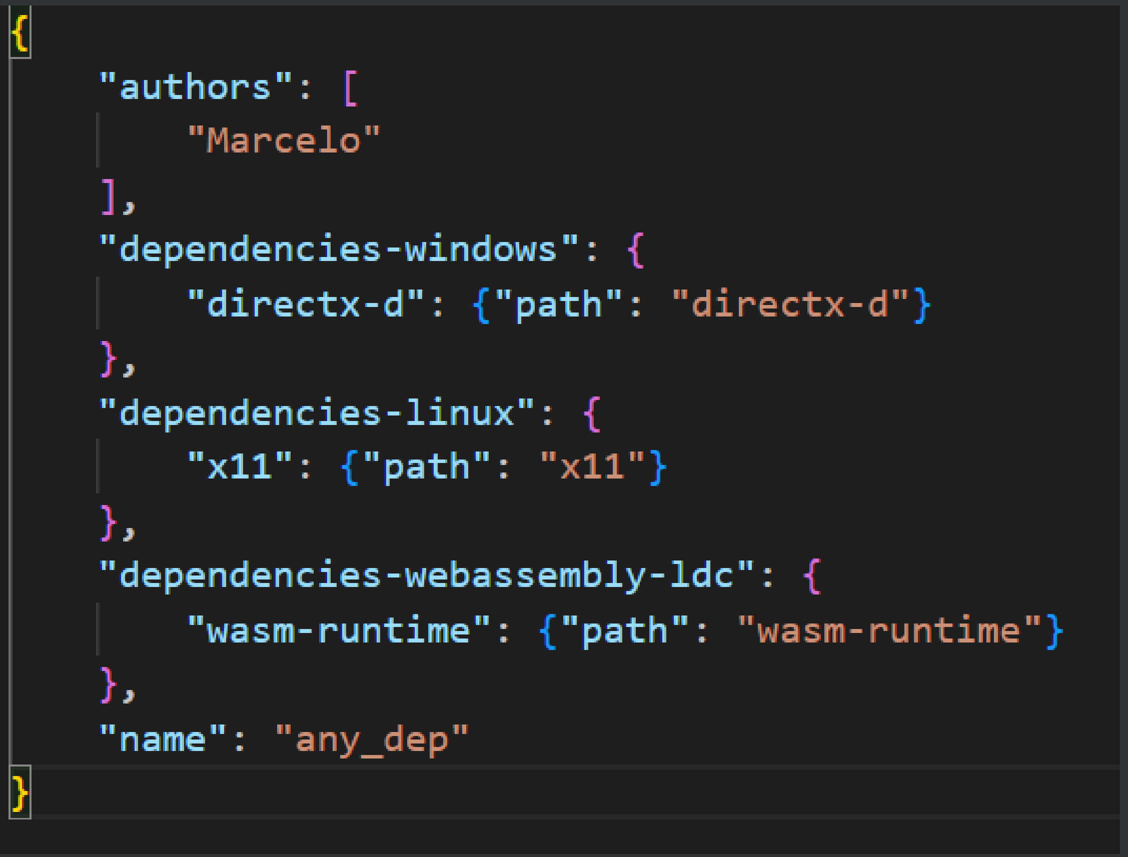


```d
BuildConfiguration merge(BuildConfiguration other) const
{
    import std.algorithm.comparison:either;
    BuildConfiguration ret = clone;
    ret.targetType = either(other.targetType, ret.targetType);
    ret.outputDirectory = either(other.outputDirectory, ret.outputDirectory);
    ret = ret.mergeCommands(other);
    ret.extraDependencyFiles.exclusiveMergePaths(other.extraDependencyFiles);
    ret.filesToCopy.exclusiveMergePaths(other.filesToCopy);
    ret.stringImportPaths.exclusiveMergePaths(other.stringImportPaths);
    ret.sourceFiles.exclusiveMerge(other.sourceFiles);
    ret.excludeSourceFiles.exclusiveMerge(other.excludeSourceFiles);
    ret.sourcePaths.exclusiveMergePaths(other.sourcePaths);
    ret.importDirectories.exclusiveMergePaths(other.importDirectories);
    ret.versions.exclusiveMerge(other.versions);
    ret.debugVersions.exclusiveMerge(other.debugVersions);
    ret.dFlags.exclusiveMerge(other.dFlags);
    ret.libraries.exclusiveMerge(other.libraries);
    ret.libraryPaths.exclusiveMergePaths(other.libraryPaths);
    ret.linkFlags.exclusiveMerge(other.linkFlags);
    return ret;
}
```

Beyond the performance improvement



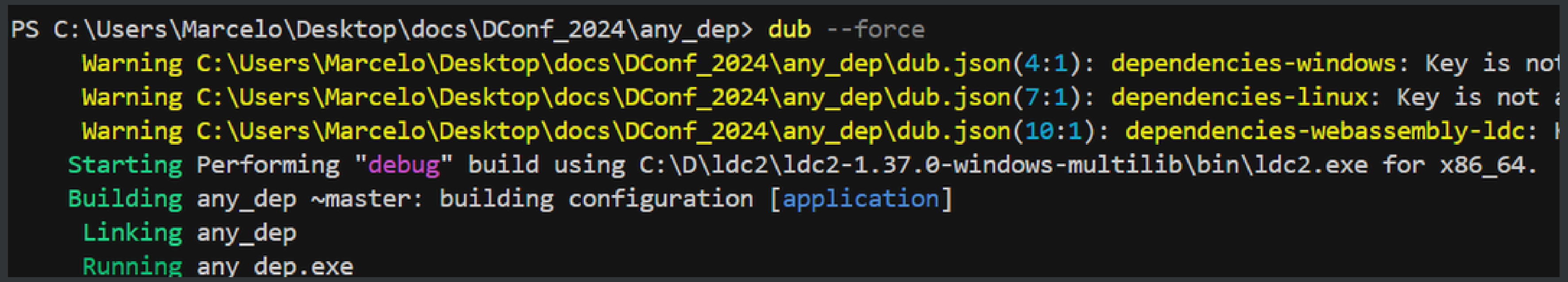
```d
version(Have_directx_d)
    import directx;
version(Have_x11)
    import x11;
version(Have_wasm_runtime)
    import wasm_runtime;


void main()
{
    version(Have_directx_d)
        StartDirectX();
    version(Have_x11)
        StartX11();
    version(Have_wasm_runtime)
        StartWasmRuntime();
}
```



```json
{
    "authors": [
        "Marcelo"
    ],
    "dependencies-windows": {
        "directx-d": {"path": "directx-d"}
    },
    "dependencies-linux": {
        "x11": {"path": "x11"}
    },
    "dependencies-webassembly-ldc": {
        "wasm-runtime": {"path": "wasm-runtime"}
    },
    "name": "any_dep"
}
```

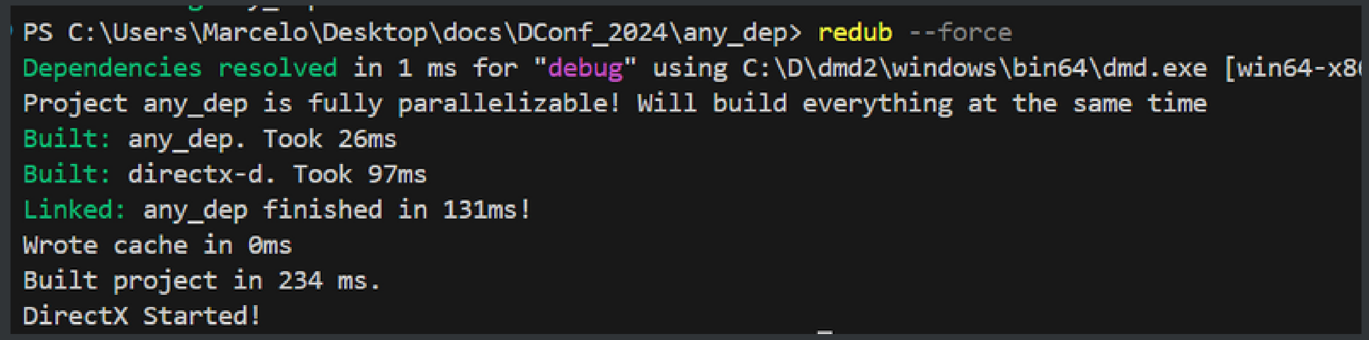
- Dependency per OS
- Dependency per compiler
- Does not require the dependency to be in folder if it is not being used
- Have_version gets more meaning

DUB



```
PS C:\Users\Marcelo\Desktop\docs\DConf_2024\any_dep> dub --force
     Warning C:\Users\Marcelo\Desktop\docs\DConf_2024\any_dep\dub.json(4:1): dependencies-windows: Key is not
     Warning C:\Users\Marcelo\Desktop\docs\DConf_2024\any_dep\dub.json(7:1): dependencies-linux: Key is not
     Warning C:\Users\Marcelo\Desktop\docs\DConf_2024\any_dep\dub.json(10:1): dependencies-webassembly-ldc:
    Starting Performing "debug" build using C:\D\ldc2\ldc2-1.37.0-windows-multilib\bin\ldc2.exe for x86_64.
    Building any_dep ~master: building configuration [application]
     Linking any_dep
     Running any_dep.exe
```

REDUB



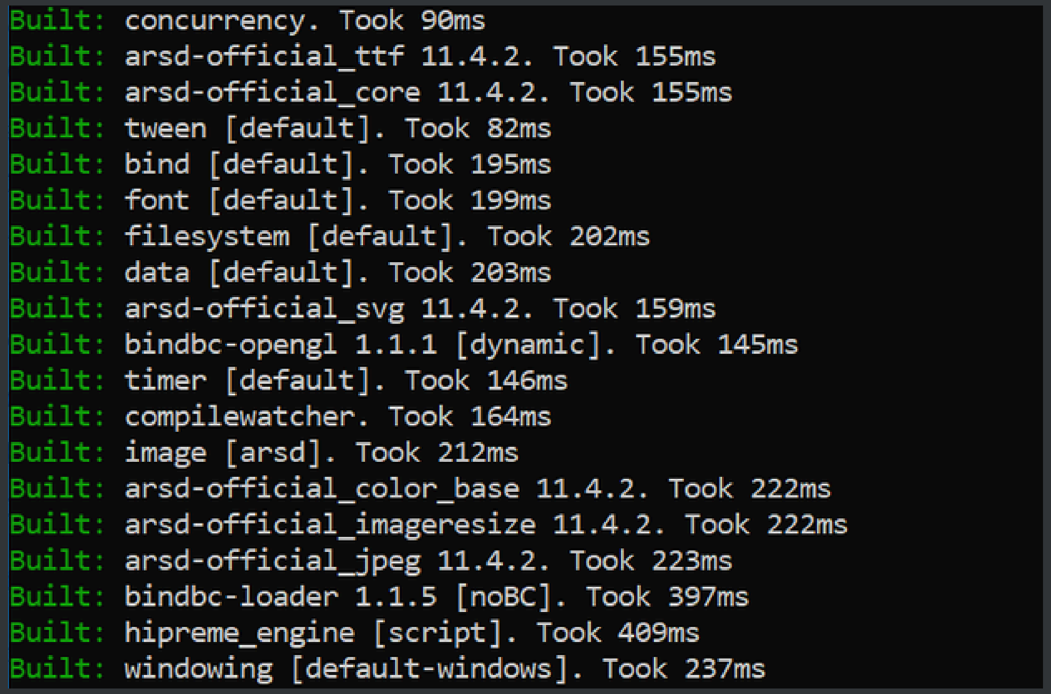
```
PS C:\Users\Marcelo\Desktop\docs\DConf_2024\any_dep> redub --force
Dependencies resolved in 1 ms for "debug" using C:\D\dmd2\windows\bin64\dmd.exe [win64-x8
Project any_dep is fully parallelizable! Will build everything at the same time
Built: any_dep. Took 26ms
Built: directx-d. Took 97ms
Linked: any_dep finished in 131ms!
Wrote cache in 0ms
Built project in 234 ms.
DirectX Started!
```

REDUB - CONFIGURATION FLEXIBILITY

Beyond the performance improvement

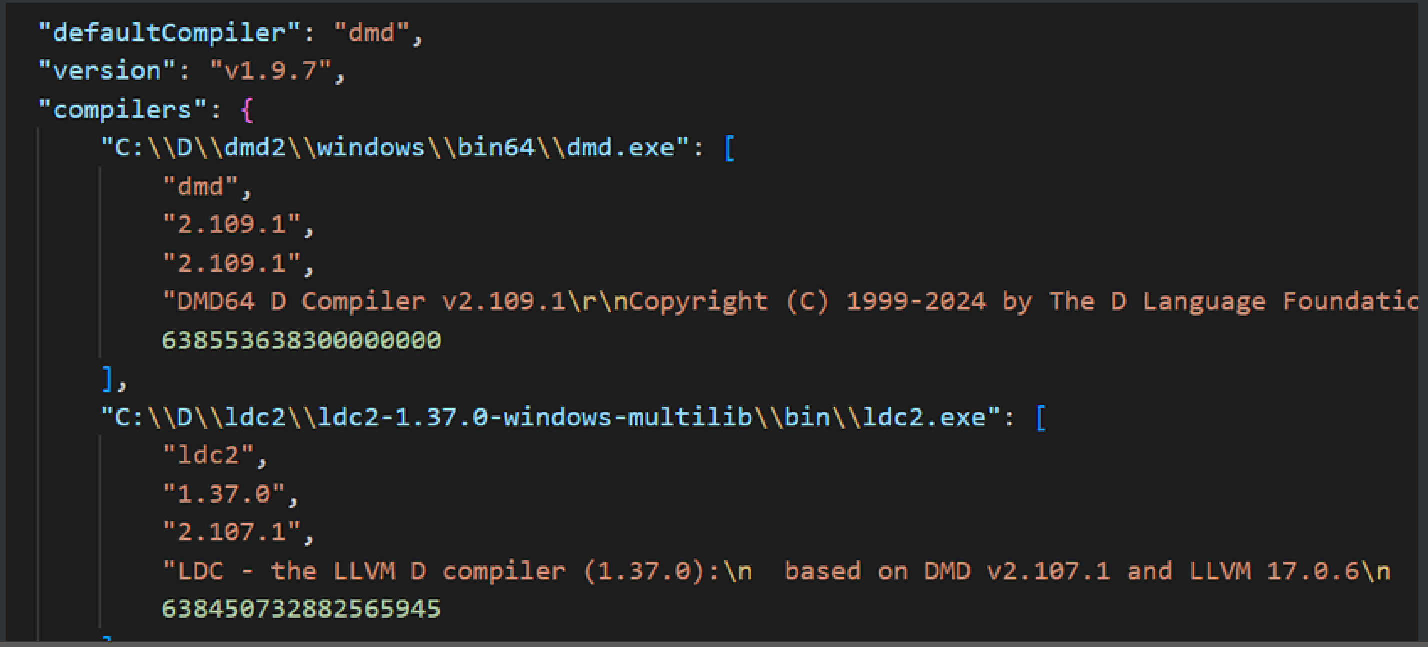
- Suggests newer versions when having a compilation error
- Caches compiler version and type
- Configurable default compiler
- Constant compilation time awareness
- Uses content hashing, so the file is not rebuilt on a simple resave

Compilation timings



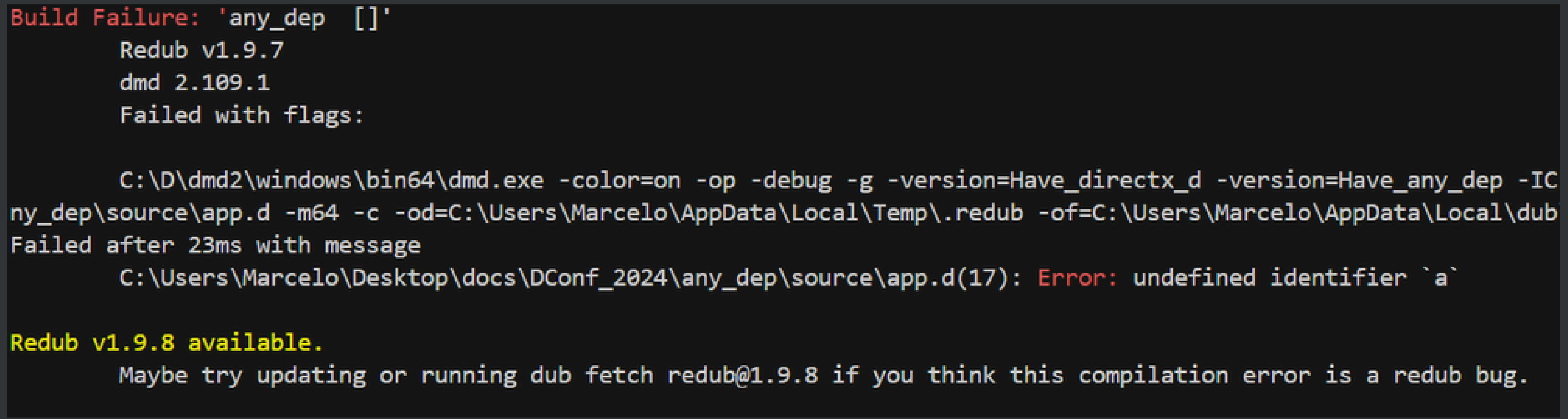
```
Built: concurrency. Took 90ms
Built: arsd-official_ttf 11.4.2. Took 155ms
Built: arsd-official_core 11.4.2. Took 155ms
Built: tween [default]. Took 82ms
Built: bind [default]. Took 195ms
Built: font [default]. Took 199ms
Built: filesystem [default]. Took 202ms
Built: data [default]. Took 203ms
Built: arsd-official_svg 11.4.2. Took 159ms
Built: bindbc-opengl 1.1.1 [dynamic]. Took 145ms
Built: timer [default]. Took 146ms
Built: compilewatcher. Took 164ms
Built: image [arsd]. Took 212ms
Built: arsd-official_color_base 11.4.2. Took 222ms
Built: arsd-official_imageresize 11.4.2. Took 222ms
Built: arsd-official_jpeg 11.4.2. Took 223ms
Built: bindbc-loader 1.1.5 [noBC]. Took 397ms
Built: hipreme_engine [script]. Took 409ms
Built: windowing [default-windows]. Took 237ms
```

Redub meta information



```
"defaultCompiler": "dmd",
"version": "v1.9.7",
"compilers": {
    "C:\\D\\dmd2\\windows\\bin64\\dmd.exe": [
        "dmd",
        "2.109.1",
        "2.109.1",
        "DMD64 D Compiler v2.109.1\r\nCopyright (C) 1999-2024 by The D Language Foundatic
        638553638300000000
    ],
    "C:\\D\\ldc2\\ldc2-1.37.0-windows-multilib\\bin\\ldc2.exe": [
        "ldc2",
        "1.37.0",
        "2.107.1",
        "LDC - the LLVM D compiler (1.37.0):\n  based on DMD v2.107.1 and LLVM 17.0.6\n
        638450732882565945
```

Newer version suggestion



```
Build Failure: 'any_dep  []'
        Redub v1.9.7
        dmd 2.109.1
        Failed with flags:

        C:\D\dmd2\windows\bin64\dmd.exe -color=on -op -debug -g -version=Have_directx_d -version=Have_any_dep -IC
ny_dep\source\app.d -m64 -c -od=C:\Users\Marcelo\AppData\Local\Temp\.redub -of=C:\Users\Marcelo\AppData\Local\dub
Failed after 23ms with message
        C:\Users\Marcelo\Desktop\docs\DConf_2024\any_dep\source\app.d(17): Error: undefined identifier `a`

Redub v1.9.8 available.
        Maybe try updating or running dub fetch redub@1.9.8 if you think this compilation error is a redub bug.
```

Development Plan

Starting Small

- **All at Once:** Create a struct that can contain any content inside dub.json
- **Path Based:** Don’t care about version, only where to parse
- **Parallelize:** Try to be stateless whenever possible so it is easier to deal with parallel
- **Dub’s Output**: Use verbose and try to replicate
- **Support Hipreme Engine:** Redub must also support its dependencies
- **Find where dub is slow:** Understand and tackle those places
- **Build and Run:** Do not care about package management or other responsibilities dub have on the beginning

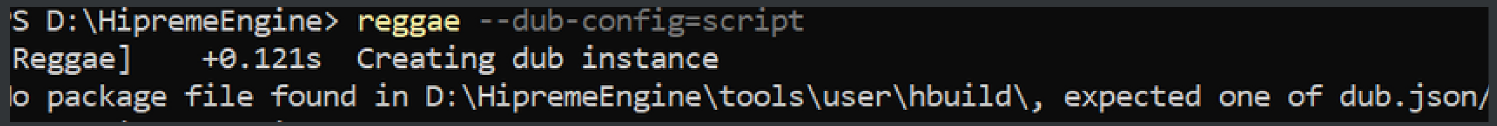
Going Further with content hashing

- **Avoid Building:** Dub always rebuilt your project if you hit save on your code editor, redub needs to address that
- **Hashing Check:** Fast algorithm needed, and a quick research would show xxhash as an option. It’s 3 times faster than using md5 in practice
- **Idle Searching:** When running in parallel, the main thread was idle, but it was possible to start hash calculating after the first file finished, providing a big increase in speed
- **Separate Responsibility:** Redub created a new package called **adv_diff,** all it does is storing a cache formula which could be compared to others and output their differences, so, the code were easier to maintain

Compilation Time Comparison

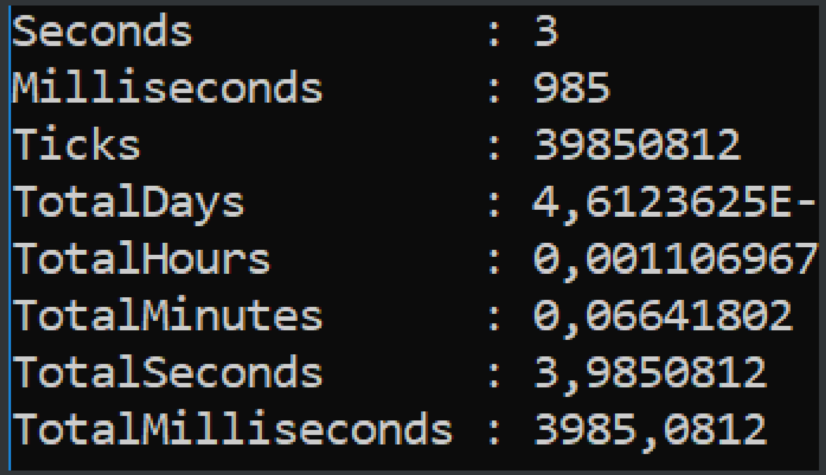
Hipreme Engine - Using DMD

Reggae's just too hard to configure, in that case, it is trying to find a file which doesn't even exist



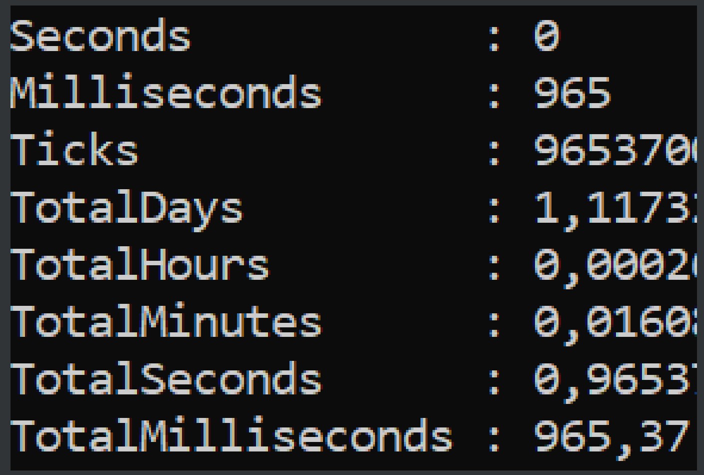
```
S D:\HipremeEngine> reggae --dub-config=script
Reggae]     +0.121s  Creating dub instance
o package file found in D:\HipremeEngine\tools\user\hbuild\, expected one of dub.json/
```

Full Rebuild on **DUB**



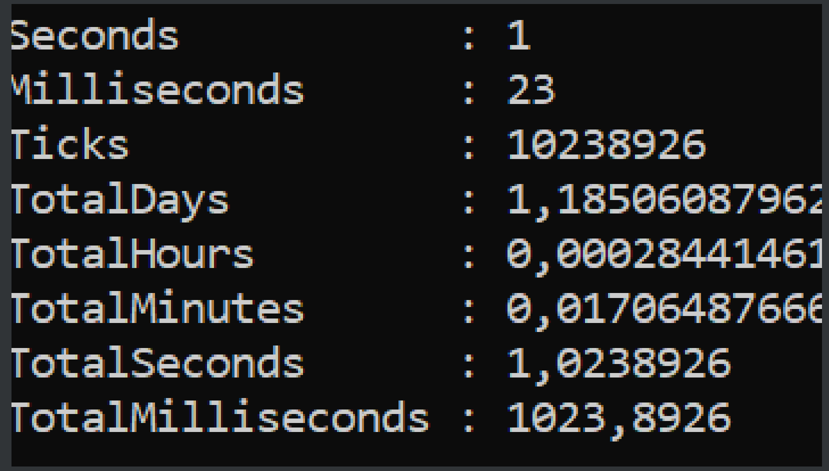
```
Seconds           : 3
Milliseconds      : 985
Ticks             : 39850812
TotalDays         : 4,6123625E-
TotalHours        : 0,001106967
TotalMinutes      : 0,06641802
TotalSeconds      : 3,9850812
TotalMilliseconds : 3985,0812
```

Full Rebuild on **REDUB**

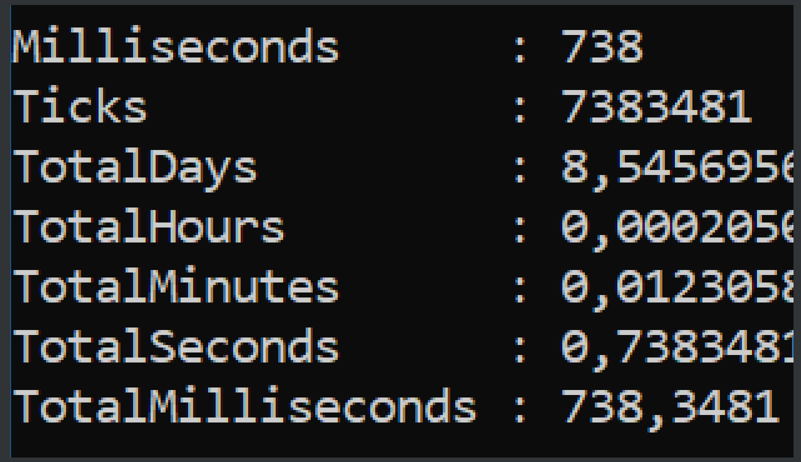


```
Seconds           : 0
Milliseconds      : 965
Ticks             : 965370
TotalDays         : 1,1173
TotalHours        : 0,0002
TotalMinutes      : 0,0160
TotalSeconds      : 0,9653
TotalMilliseconds : 965,37
```

Root Only Build



```
Seconds           : 1
Milliseconds      : 23
Ticks             : 10238926
TotalDays         : 1,18506087962
TotalHours        : 0,00028441461
TotalMinutes      : 0,01706487666
TotalSeconds      : 1,0238926
TotalMilliseconds : 1023,8926
```

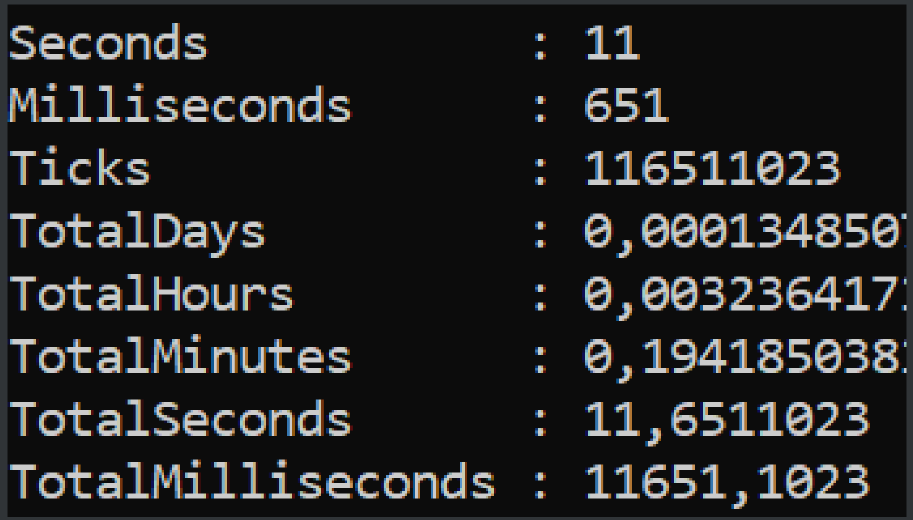


```
Milliseconds      : 738
Ticks             : 7383481
TotalDays         : 8,5456956
TotalHours        : 0,0002050
TotalMinutes      : 0,0123058
TotalSeconds      : 0,738348
TotalMilliseconds : 738,3481
```

Compilation Time Comparison

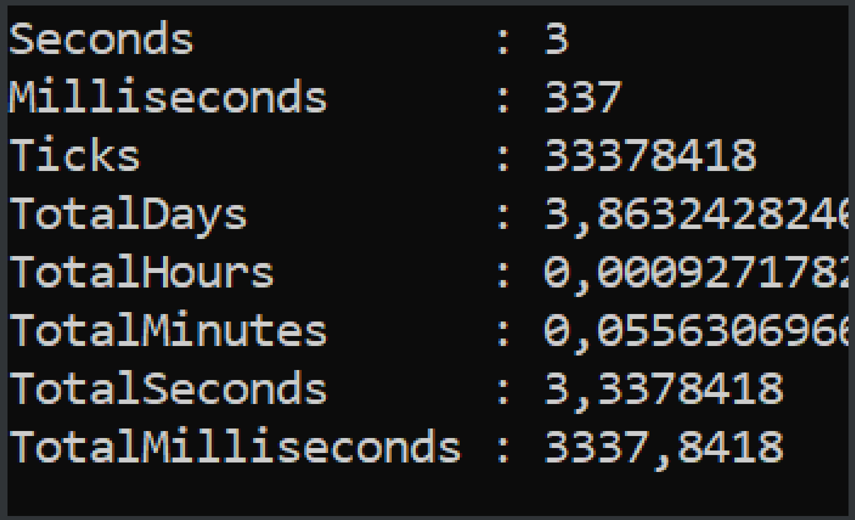
VibeD - Hello World - Full rebuild using DMD

DUB



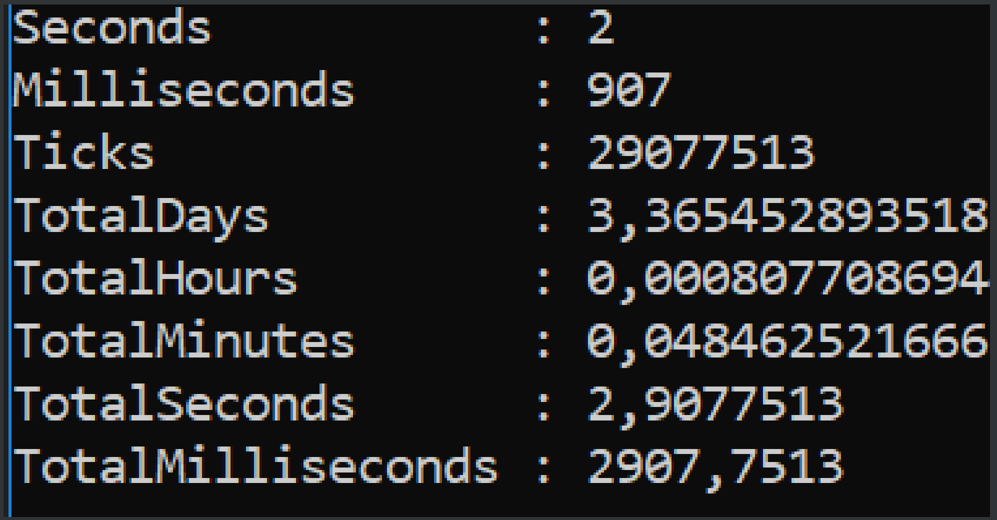
```
Seconds           : 11
Milliseconds      : 651
Ticks             : 116511023
TotalDays         : 0,000134850
TotalHours        : 0,003236417
TotalMinutes      : 0,194185038
TotalSeconds      : 11,6511023
TotalMilliseconds : 11651,1023
```

REDUB

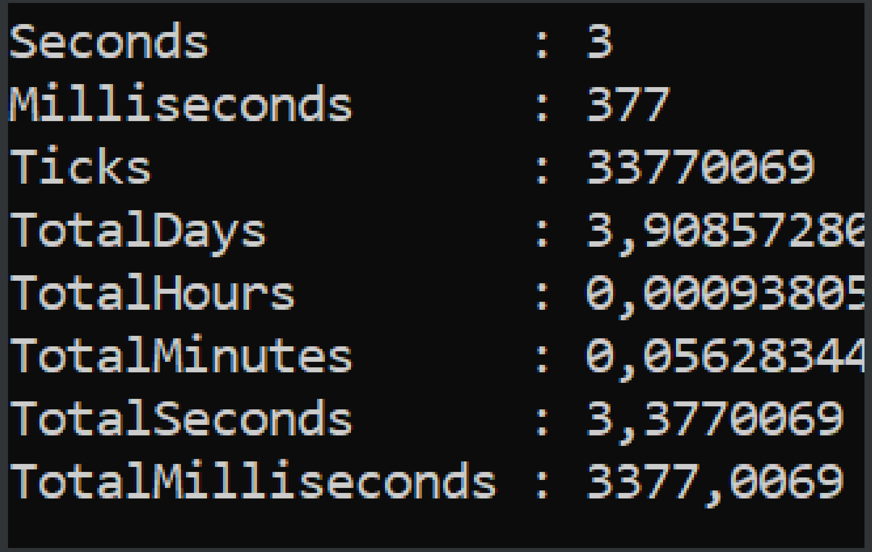


```
Seconds           : 3
Milliseconds      : 337
Ticks             : 33378418
TotalDays         : 3,863242824
TotalHours        : 0,000927178
TotalMinutes      : 0,055630696
TotalSeconds      : 3,3378418
TotalMilliseconds : 3337,8418
```

REGGAE + NINJA



```
Seconds           : 2
Milliseconds      : 907
Ticks             : 29077513
TotalDays         : 3,365452893518
TotalHours        : 0,000807708694
TotalMinutes      : 0,048462521666
TotalSeconds      : 2,9077513
TotalMilliseconds : 2907,7513
```



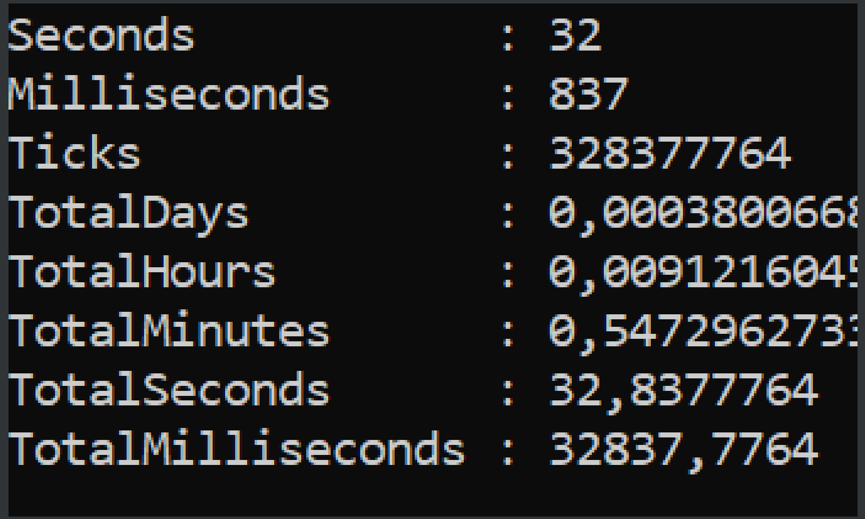
```
Seconds           : 3
Milliseconds      : 377
Ticks             : 33770069
TotalDays         : 3,9085728
TotalHours        : 0,00093805
TotalMinutes      : 0,05628344
TotalSeconds      : 3,3770069
TotalMilliseconds : 3377,0069
```

Some investigative work should be done on ninja. On the first run, it actually went faster than redub, but after running ‘dub’, it started going at the same speed as redub

Compilation Time Comparison

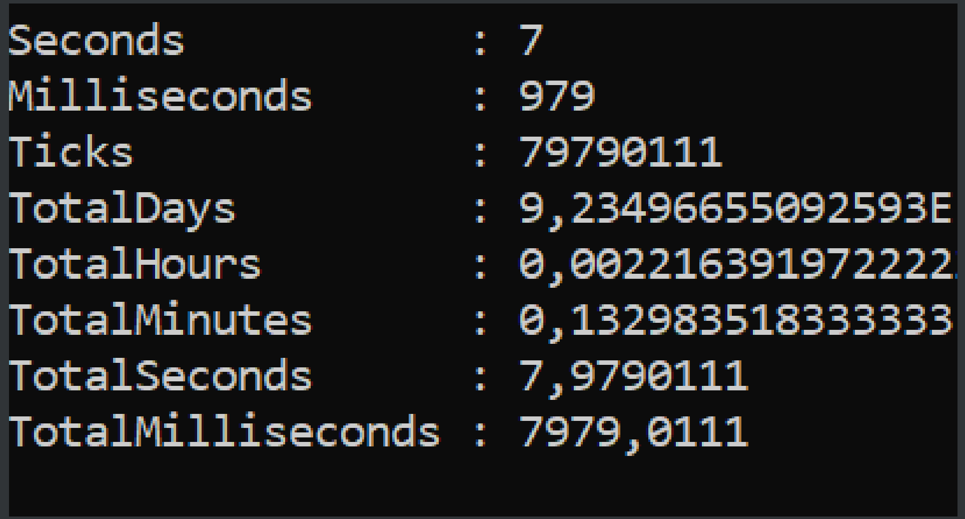
Dub Registry - Full Rebuild Only

DUB



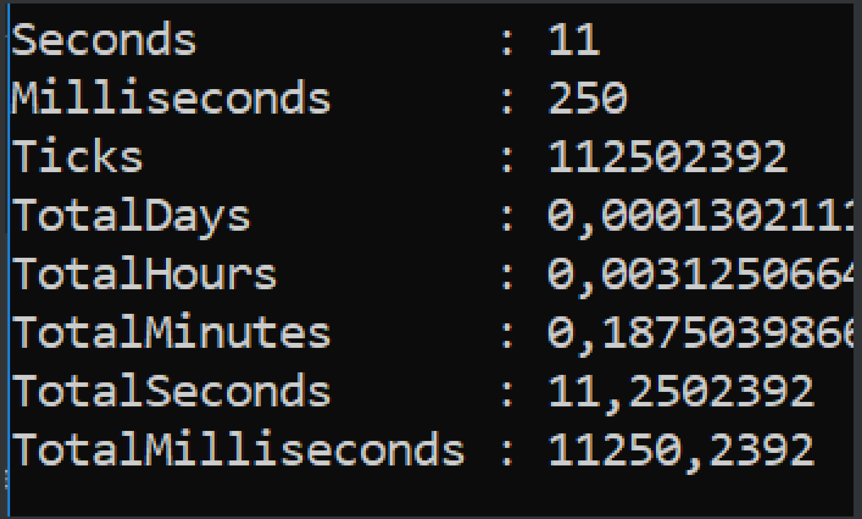
```
Seconds           : 32
Milliseconds      : 837
Ticks             : 328377764
TotalDays         : 0,0003800668
TotalHours        : 0,0091216045
TotalMinutes      : 0,5472962733
TotalSeconds      : 32,8377764
TotalMilliseconds : 32837,7764
```

REDUB



```
Seconds           : 7
Milliseconds      : 979
Ticks             : 79790111
TotalDays         : 9,23496655092593E
TotalHours        : 0,002216391972222
TotalMinutes      : 0,132983518333333
TotalSeconds      : 7,9790111
TotalMilliseconds : 7979,0111
```

REGGAE + NINJA



```
Seconds           : 11
Milliseconds      : 250
Ticks             : 112502392
TotalDays         : 0,0001302111
TotalHours        : 0,0031250664
TotalMinutes      : 0,1875039866
TotalSeconds      : 11,2502392
TotalMilliseconds : 11250,2392
```

But How Redub went faster than Ninja

- On Windows, depending on how many libraries are you linking (or maybe, even other unknown parameter), the incremental linker, which is the default for MSVCLinker actually is able to slowdown your compilation!
- Beyond simply “building”, redub also aims to achieve a better default configuration, it have a configuration based on how many dependencies there are. Currently, incremental linker is deactivated whenever there are more than 3 dependencies

DUB
Community Usage Division

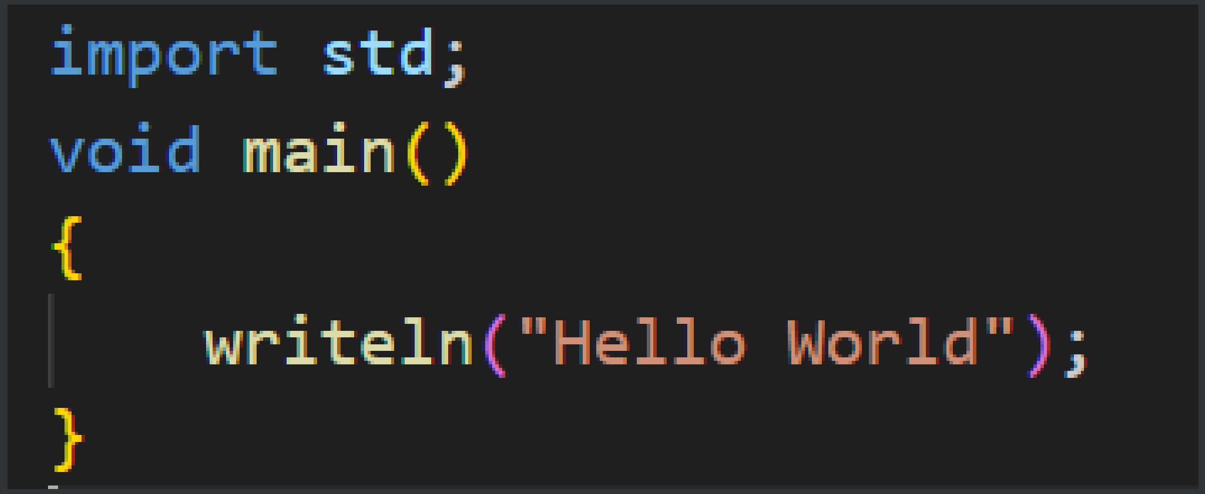
- **Lack of Understanding:** How the D compiler works, how to organize their package
- **Dub's Scope:** Every project can have a quite different scope, this causes a dissociation on everyone's expectation on what dub should and should not do
- **Separate Compilation:** When it causes more harm than good
- **Dub as a Library:** Usage is heavily considered when taking decisions, but it's use-case is too rare. SemVer should be used instead of preventing progress, dub is not a compiler.

Semantic 1

- Text Parsing
- Import Evaluation
- Mixin Template evaluation
- Strategic usage can avoid 1 and 2 trigger
- Easiest to Optimize
- Easy compiler performance killer

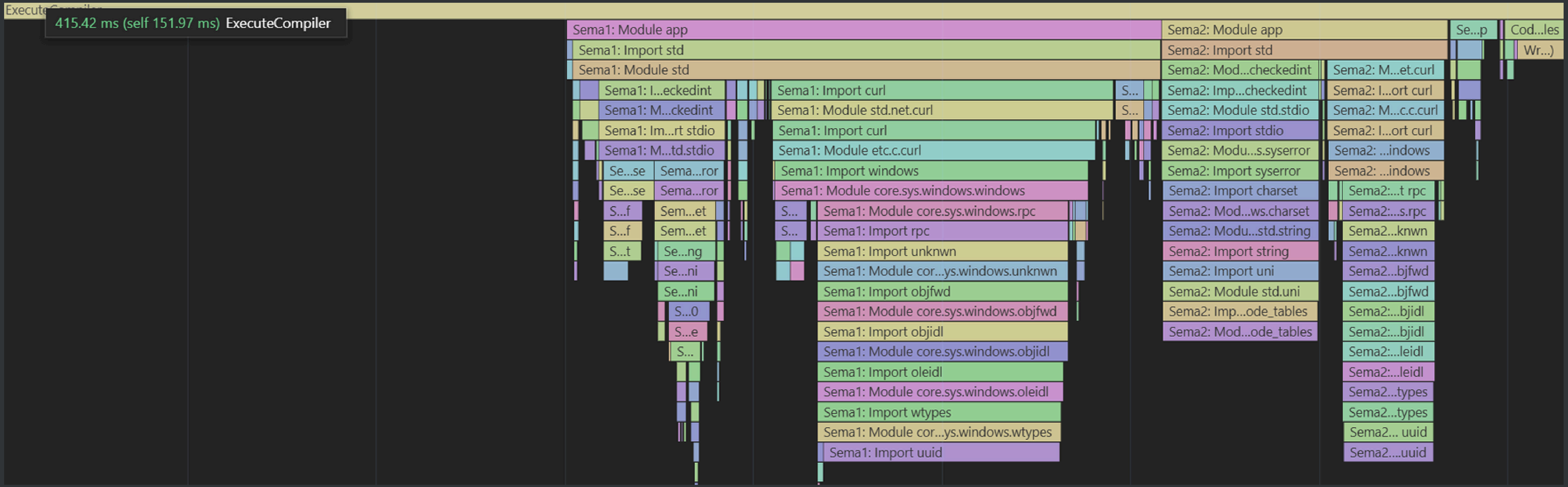
Semantic 1 - Irresponsible usage of imports

- Don’t do this inside a community package!



```d
import std;
void main()
{
    writeln("Hello World");
}
```

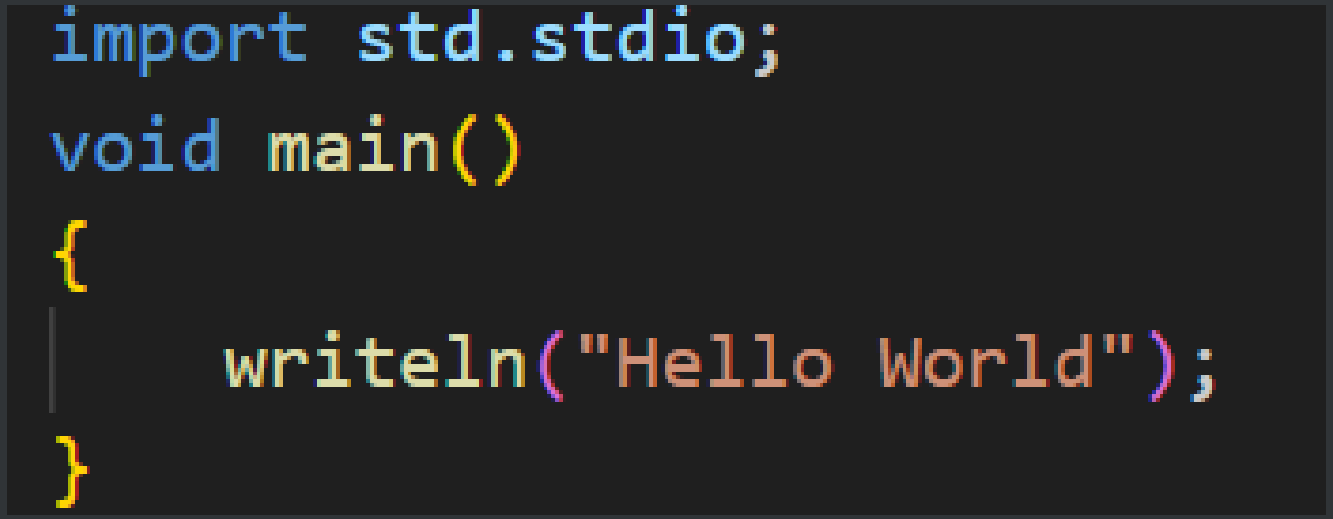
- 400ms for a simple Hello World
- package.d (import std) should be avoided for any non simple case.



415.42 ms (self 151.97 ms) ExecuteCompiler

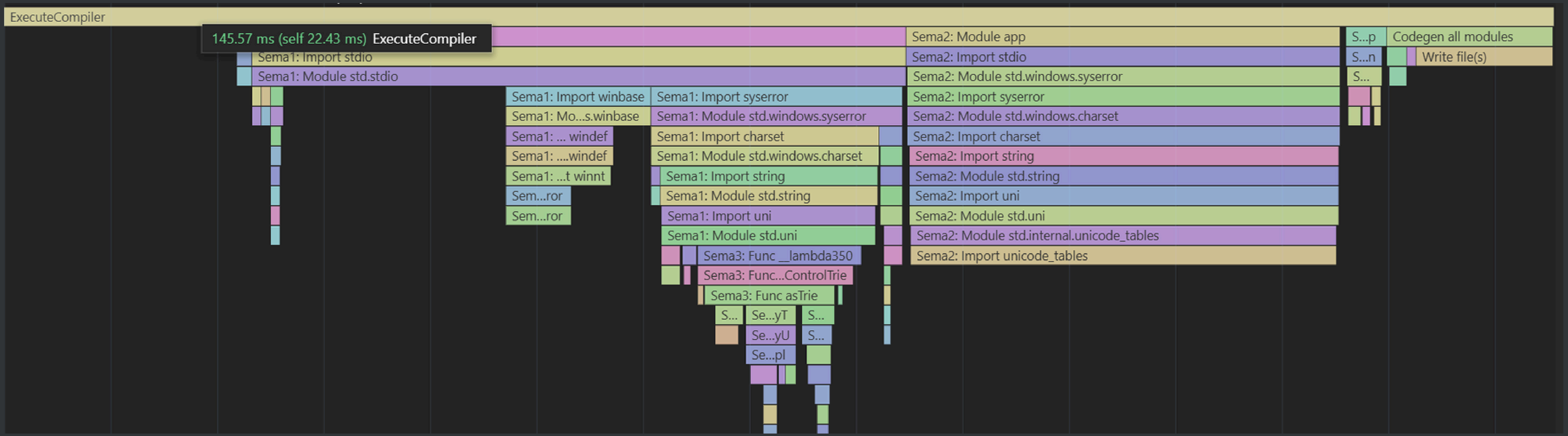
Semantic 1 - Solving imports

- A simple change by adding **.stdio** after **std**



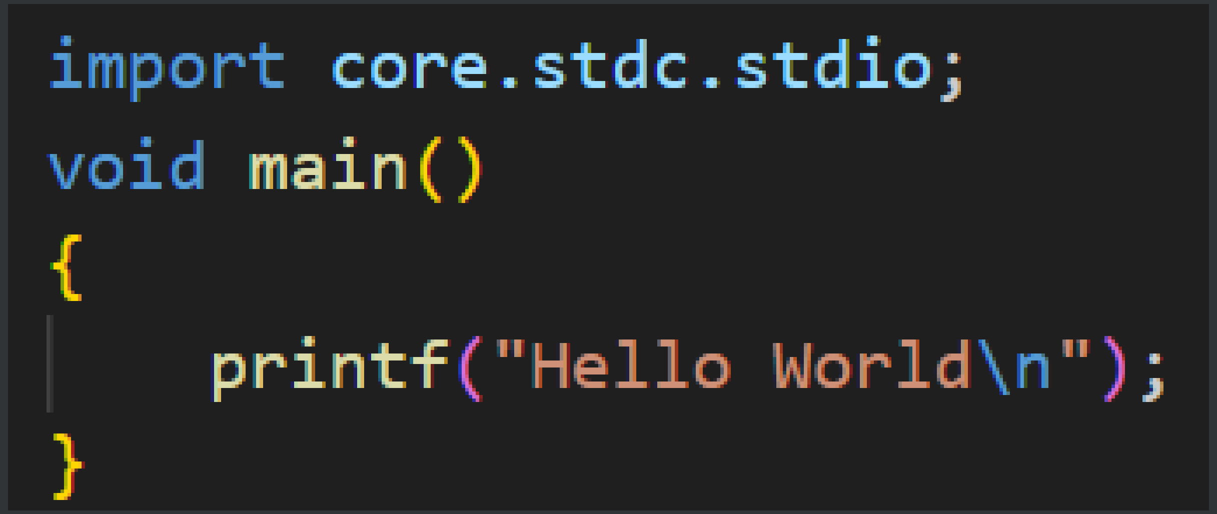
```d
import std.stdio;
void main()
{
    writeln("Hello World");
}
```

- Drops from 400ms to 140ms
- A compilation aware library could go even further



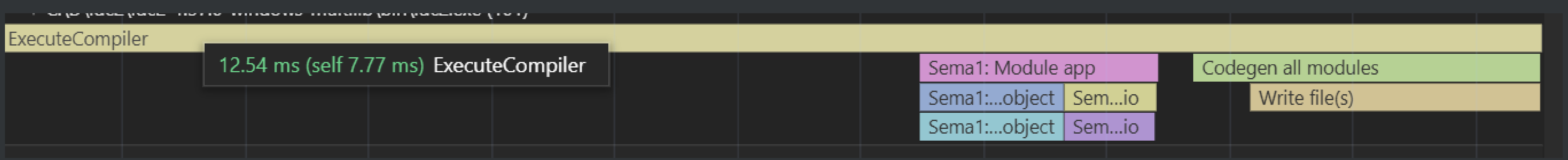
Semantic 1 - Going further with imports

- Using C’s stdio printf directly



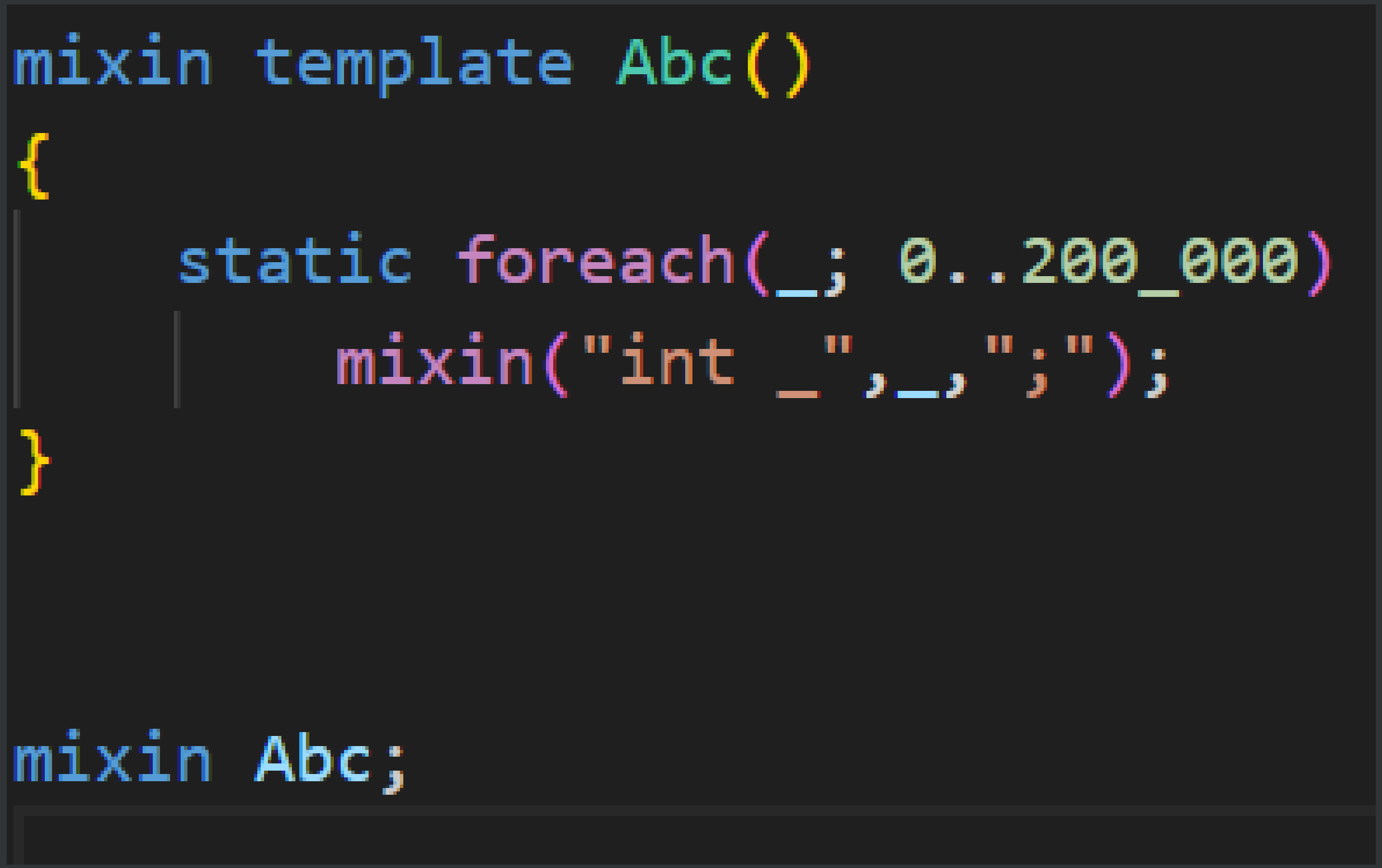
```d
import core.stdc.stdio;
void main()
{
    printf("Hello World\n");
}
```

- Drops from 140ms to 12ms
- Implementation is very different, but achieves the same result for the same program



```
ExecuteCompiler
12.54 ms (self 7.77 ms) ExecuteCompiler
Sema1: Module app          Codegen all modules
Sema1:...object  Sem...io      Write file(s)
Sema1:...object  Sem...io
```

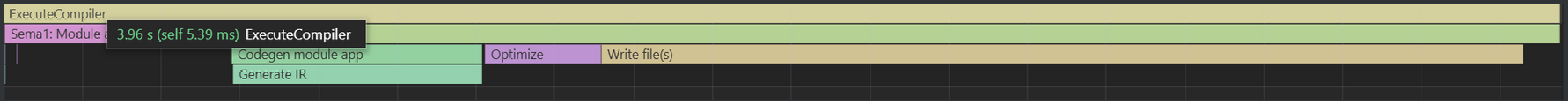
Semantic 1 - Mixin Templates



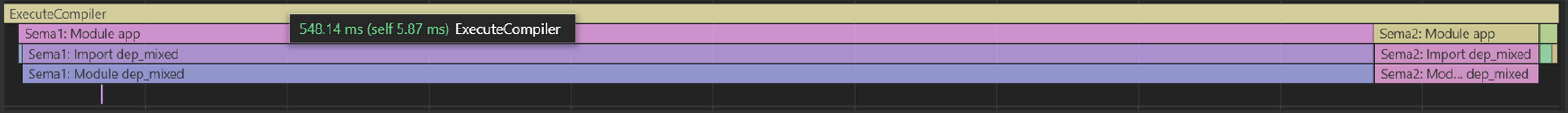
```d
mixin template Abc()
{
    static foreach(_; 0..200_000)
        mixin("int _",_,";");
}


mixin Abc;
```

- 470ms is spent on Sema 1
- It increased codegen time since it is an absurd example
- This is an example on a situation where Sema 1 can lead to a really bigger compilation time
- Importing that module will waste time

As a main module



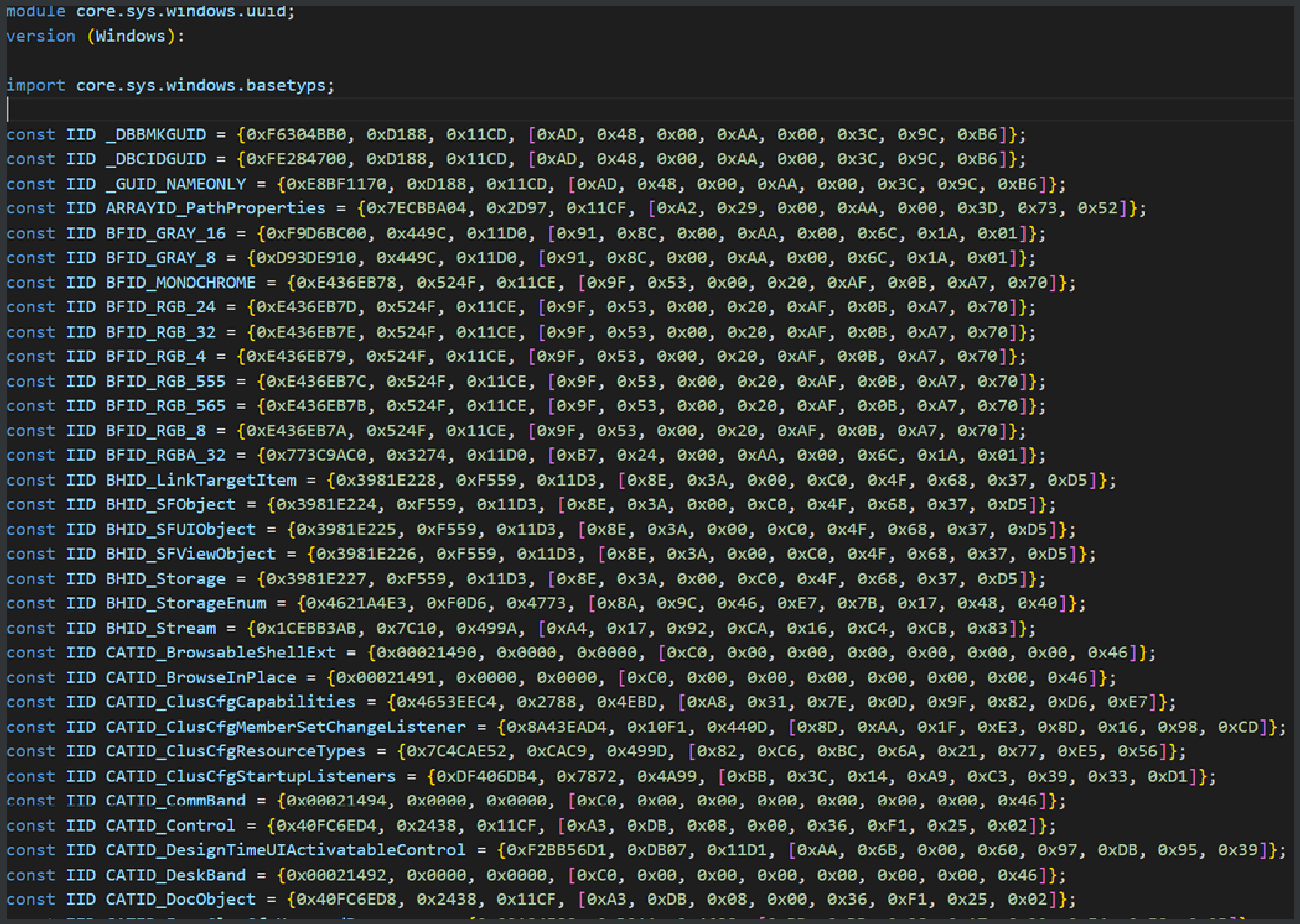
As a dependency (notice also Sema 2 here)



Semantic 2

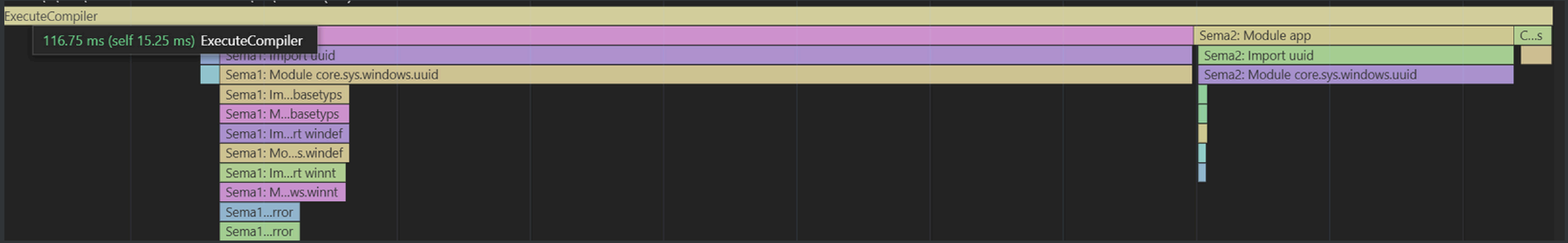
- CTFE (Compile Time Function Evaluation)
- Default Initializers
- Classic Example: std.internal.unicode_tables
- Obvious Bottlenecks

Semantic 2 - Default Initialization

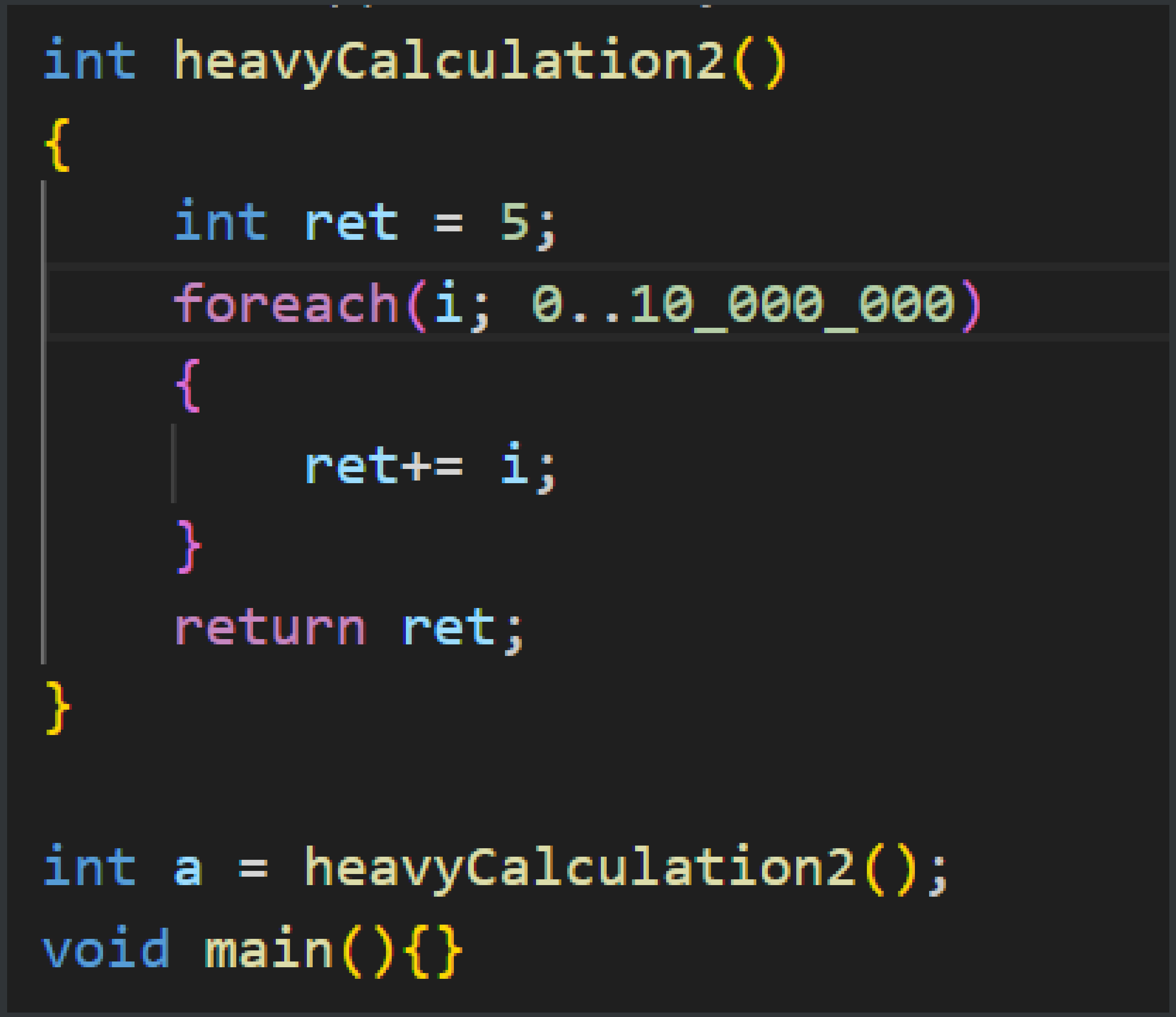


```
module core.sys.windows.uuid;
version (Windows):

import core.sys.windows.basetyps;

const IID _DBBMKGUID = {0xF6304BB0, 0xD188, 0x11CD, [0xAD, 0x48, 0x00, 0xAA, 0x00, 0x3C, 0x9C, 0xB6]};
const IID _DBCIDGUID = {0xFE284700, 0xD188, 0x11CD, [0xAD, 0x48, 0x00, 0xAA, 0x00, 0x3C, 0x9C, 0xB6]};
const IID _GUID_NAMEONLY = {0xE8BF1170, 0xD188, 0x11CD, [0xAD, 0x48, 0x00, 0xAA, 0x00, 0x3C, 0x9C, 0xB6]};
const IID ARRAYID_PathProperties = {0x7ECBBA04, 0x2D97, 0x11CF, [0xA2, 0x29, 0x00, 0xAA, 0x00, 0x3D, 0x73, 0x52]};
const IID BFID_GRAY_16 = {0xF9D6BC00, 0x449C, 0x11D0, [0x91, 0x8C, 0x00, 0xAA, 0x00, 0x6C, 0x1A, 0x01]};
const IID BFID_GRAY_8 = {0xD93DE910, 0x449C, 0x11D0, [0x91, 0x8C, 0x00, 0xAA, 0x00, 0x6C, 0x1A, 0x01]};
const IID BFID_MONOCHROME = {0xE436EB78, 0x524F, 0x11CE, [0x9F, 0x53, 0x00, 0x20, 0xAF, 0x0B, 0xA7, 0x70]};
const IID BFID_RGB_24 = {0xE436EB7D, 0x524F, 0x11CE, [0x9F, 0x53, 0x00, 0x20, 0xAF, 0x0B, 0xA7, 0x70]};
const IID BFID_RGB_32 = {0xE436EB7E, 0x524F, 0x11CE, [0x9F, 0x53, 0x00, 0x20, 0xAF, 0x0B, 0xA7, 0x70]};
const IID BFID_RGB_4 = {0xE436EB79, 0x524F, 0x11CE, [0x9F, 0x53, 0x00, 0x20, 0xAF, 0x0B, 0xA7, 0x70]};
const IID BFID_RGB_555 = {0xE436EB7C, 0x524F, 0x11CE, [0x9F, 0x53, 0x00, 0x20, 0xAF, 0x0B, 0xA7, 0x70]};
const IID BFID_RGB_565 = {0xE436EB7B, 0x524F, 0x11CE, [0x9F, 0x53, 0x00, 0x20, 0xAF, 0x0B, 0xA7, 0x70]};
const IID BFID_RGB_8 = {0xE436EB7A, 0x524F, 0x11CE, [0x9F, 0x53, 0x00, 0x20, 0xAF, 0x0B, 0xA7, 0x70]};
const IID BFID_RGBA_32 = {0x773C9AC0, 0x3274, 0x11D0, [0xB7, 0x24, 0x00, 0xAA, 0x00, 0x6C, 0x1A, 0x01]};
const IID BHID_LinkTargetItem = {0x3981E228, 0xF559, 0x11D3, [0x8E, 0x3A, 0x00, 0xC0, 0x4F, 0x68, 0x37, 0xD5]};
const IID BHID_SFObject = {0x3981E224, 0xF559, 0x11D3, [0x8E, 0x3A, 0x00, 0xC0, 0x4F, 0x68, 0x37, 0xD5]};
const IID BHID_SFUIObject = {0x3981E225, 0xF559, 0x11D3, [0x8E, 0x3A, 0x00, 0xC0, 0x4F, 0x68, 0x37, 0xD5]};
const IID BHID_SFViewObject = {0x3981E226, 0xF559, 0x11D3, [0x8E, 0x3A, 0x00, 0xC0, 0x4F, 0x68, 0x37, 0xD5]};
const IID BHID_Storage = {0x3981E227, 0xF559, 0x11D3, [0x8E, 0x3A, 0x00, 0xC0, 0x4F, 0x68, 0x37, 0xD5]};
const IID BHID_StorageEnum = {0x4621A4E3, 0xF0D6, 0x4773, [0x8A, 0x9C, 0x46, 0xE7, 0x7B, 0x17, 0x48, 0x40]};
const IID BHID_Stream = {0x1CEBB3AB, 0x7C10, 0x499A, [0xA4, 0x17, 0x92, 0xCA, 0x16, 0xC4, 0xCB, 0x83]};
const IID CATID_BrowsableShellExt = {0x00021490, 0x0000, 0x0000, [0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x46]};
const IID CATID_BrowseInPlace = {0x00021491, 0x0000, 0x0000, [0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x46]};
const IID CATID_ClusCfgCapabilities = {0x4653EEC4, 0x2788, 0x4EBD, [0xA8, 0x31, 0x7E, 0x0D, 0x9F, 0x82, 0xD6, 0xE7]};
const IID CATID_ClusCfgMemberSetChangeListener = {0x8A43EAD4, 0x10F1, 0x440D, [0x8D, 0xAA, 0x1F, 0xE3, 0x8D, 0x16, 0x98, 0xCD]};
const IID CATID_ClusCfgResourceTypes = {0x7C4CAE52, 0xCAC9, 0x499D, [0x82, 0xC6, 0xBC, 0x6A, 0x21, 0x77, 0xE5, 0x56]};
const IID CATID_ClusCfgStartupListeners = {0xDF406DB4, 0x7872, 0x4A99, [0xBB, 0x3C, 0x14, 0xA9, 0xC3, 0x39, 0x33, 0xD1]};
const IID CATID_CommBand = {0x00021494, 0x0000, 0x0000, [0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x46]};
const IID CATID_Control = {0x40FC6ED4, 0x2438, 0x11CF, [0xA3, 0xDB, 0x08, 0x00, 0x36, 0xF1, 0x25, 0x02]};
const IID CATID_DesignTimeUIActivatableControl = {0xF2BB56D1, 0xDB07, 0x11D1, [0xAA, 0x6B, 0x00, 0x60, 0x97, 0xDB, 0x95, 0x39]};
const IID CATID_DeskBand = {0x00021492, 0x0000, 0x0000, [0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x46]};
const IID CATID_DocObject = {0x40FC6ED8, 0x2438, 0x11CF, [0xA3, 0xDB, 0x08, 0x00, 0x36, 0xF1, 0x25, 0x02]};
```

- core.sys.windows.uuid
- That is also a big bottleneck on parallel builds when imports happen
- Public packages usually imports core.sys.windows causing huge increase on compilation time

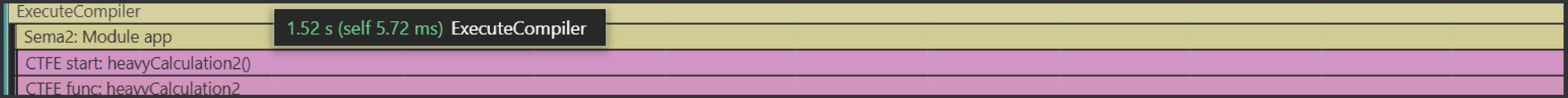


Semantic 2 - CTFE



```d
int heavyCalculation2()
{
    int ret = 5;
    foreach(i; 0..10_000_000)
    {
        ret+= i;
    }
    return ret;
}

int a = heavyCalculation2();
void main(){}
```

- int a is initialized on compile time
- A calculation that takes a lot of time happens there
- 1.52 seconds took as a result of that
- In this case, the bottleneck is very obvious

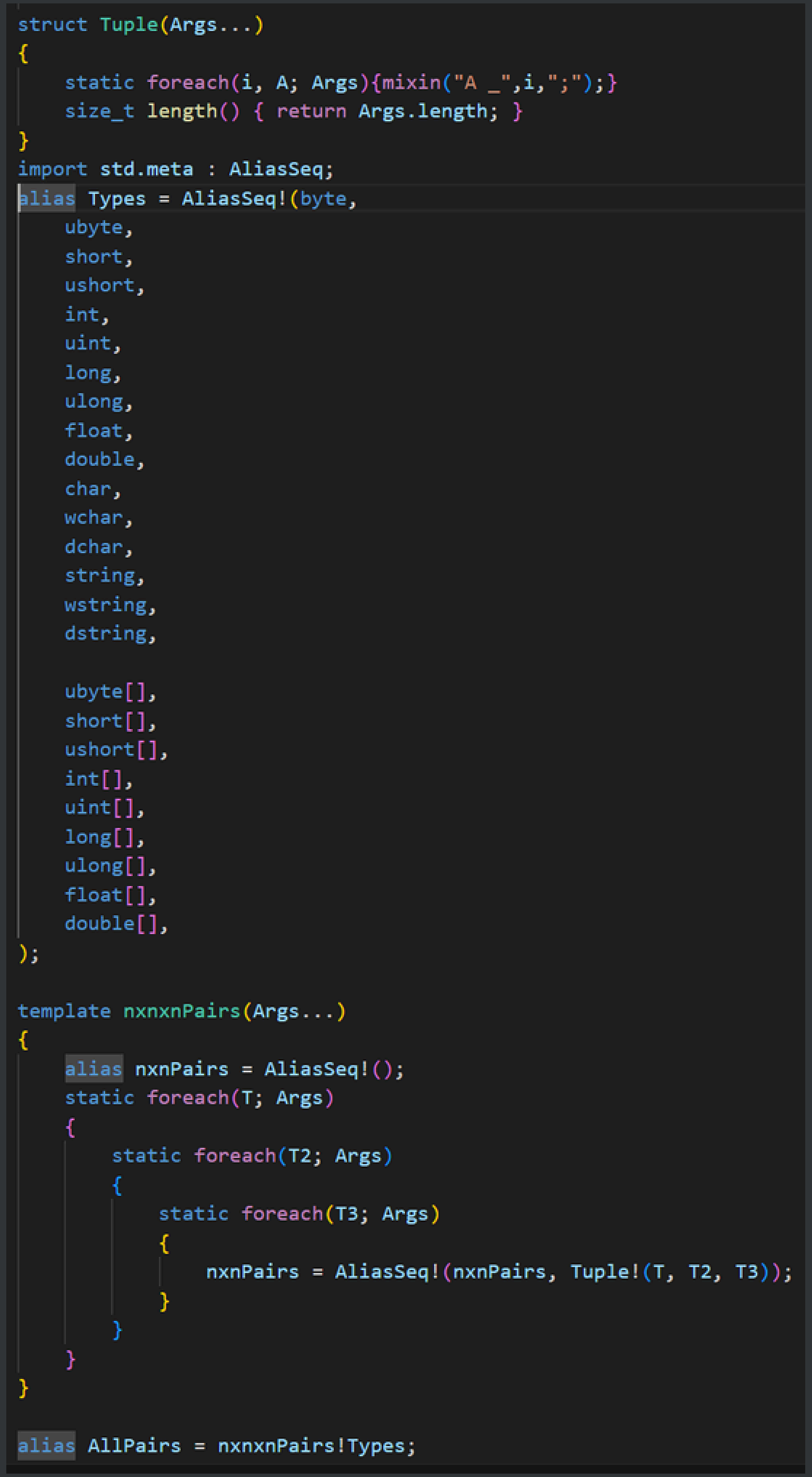


```
ExecuteCompiler
Sema2: Module app        1.52 s (self 5.72 ms) ExecuteCompiler
CTFE start: heavyCalculation2()
CTFE func: heavyCalculation2
```

Semantic 3

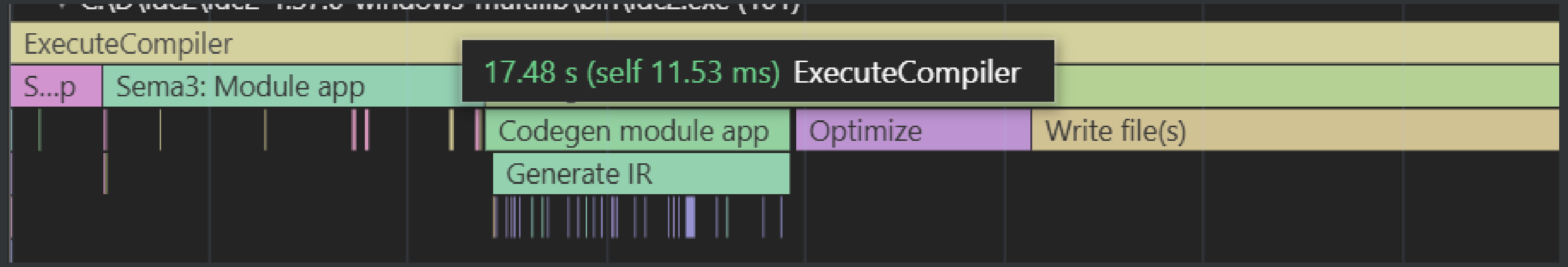
- Template Expansion
- Compile Time Reflection
- Metaprogramming
- Triggers a new Sema1 and Sema2 pass
- Function Body Evaluation
- Can kill your compilation time

Semantic 3 - Effect on compilation time

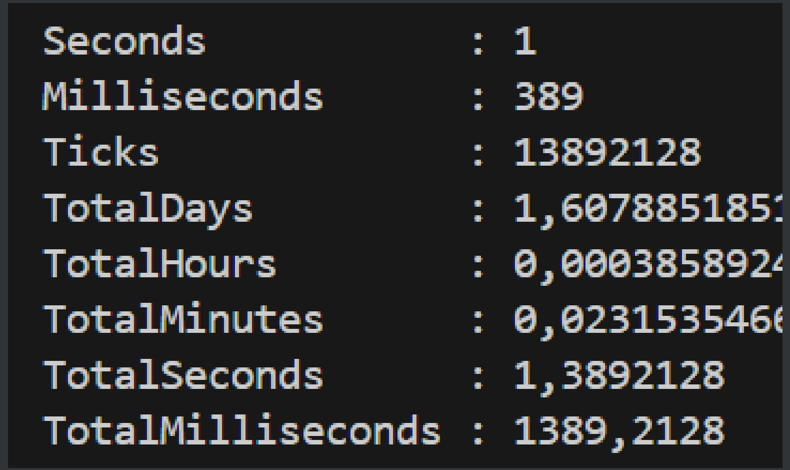


```d
struct Tuple(Args...)
{
    static foreach(i, A; Args){mixin("A _",i,";");}
    size_t length() { return Args.length; }
}
import std.meta : AliasSeq;
alias Types = AliasSeq!(byte,
    ubyte,
    short,
    ushort,
    int,
    uint,
    long,
    ulong,
    float,
    double,
    char,
    wchar,
    dchar,
    string,
    wstring,
    dstring,

    ubyte[],
    short[],
    ushort[],
    int[],
    uint[],
    long[],
    ulong[],
    float[],
    double[],
);

template nxnxnPairs(Args...)
{
    alias nxnPairs = AliasSeq!();
    static foreach(T; Args)
    {
        static foreach(T2; Args)
        {
            static foreach(T3; Args)
            {
                nxnPairs = AliasSeq!(nxnPairs, Tuple!(T, T2, T3));
            }
        }
    }
}

alias AllPairs = nxnxnPairs!Types;
```

- Template that generates n^3 tuples
- 17.48 seconds on full compilation time
- 460ms on Sema1 (as reparse has to occur)
- 1.92 seconds on Sema3
- Even if it doesn't look much on Sema3, the codegen time heavily increases based on it

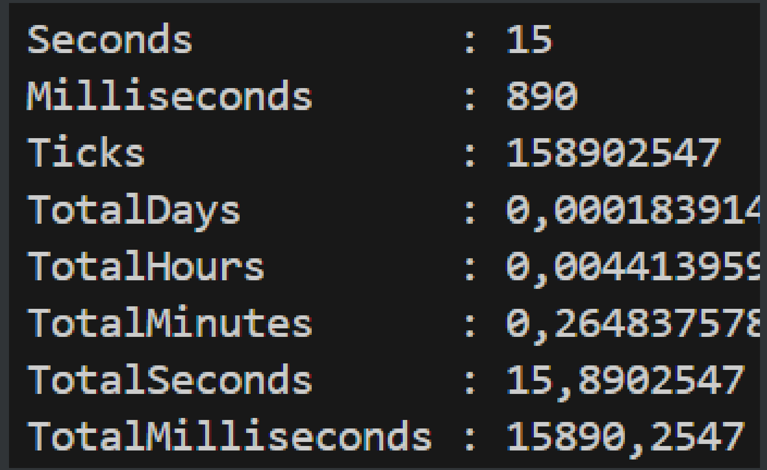


DMD with **-c -o-**



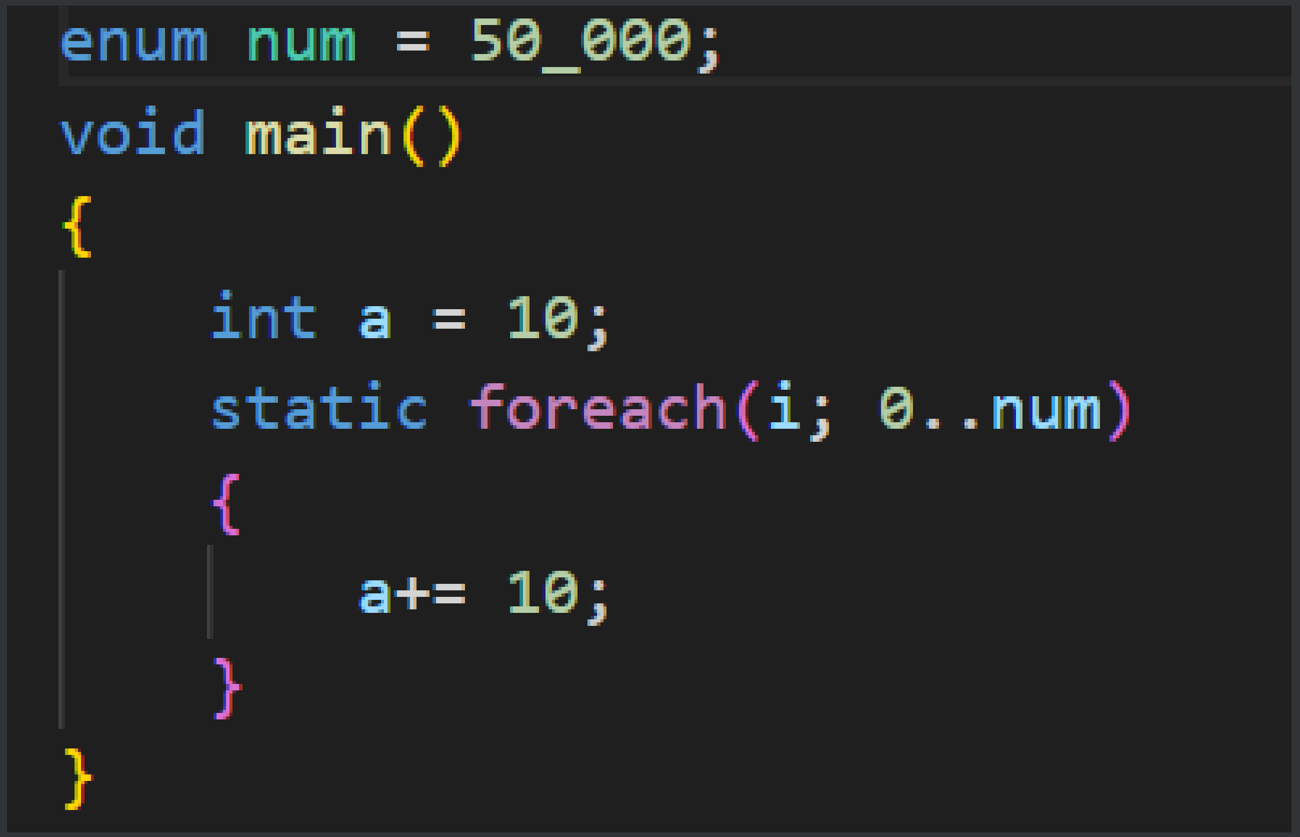
```
Seconds           : 1
Milliseconds      : 389
Ticks             : 13892128
TotalDays         : 1,6078851851
TotalHours        : 0,000385892
TotalMinutes      : 0,0231535466
TotalSeconds      : 1,3892128
TotalMilliseconds : 1389,2128
```

DMD with codegen



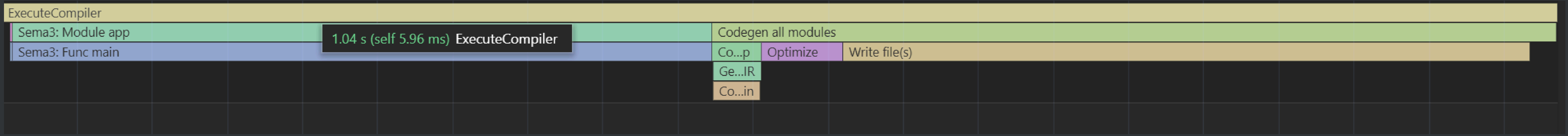
```
Seconds           : 15
Milliseconds      : 890
Ticks             : 158902547
TotalDays         : 0,000183914
TotalHours        : 0,004413959
TotalMinutes      : 0,264837578
TotalSeconds      : 15,8902547
TotalMilliseconds : 15890,2547
```

Semantic 3 - Effect on compilation time



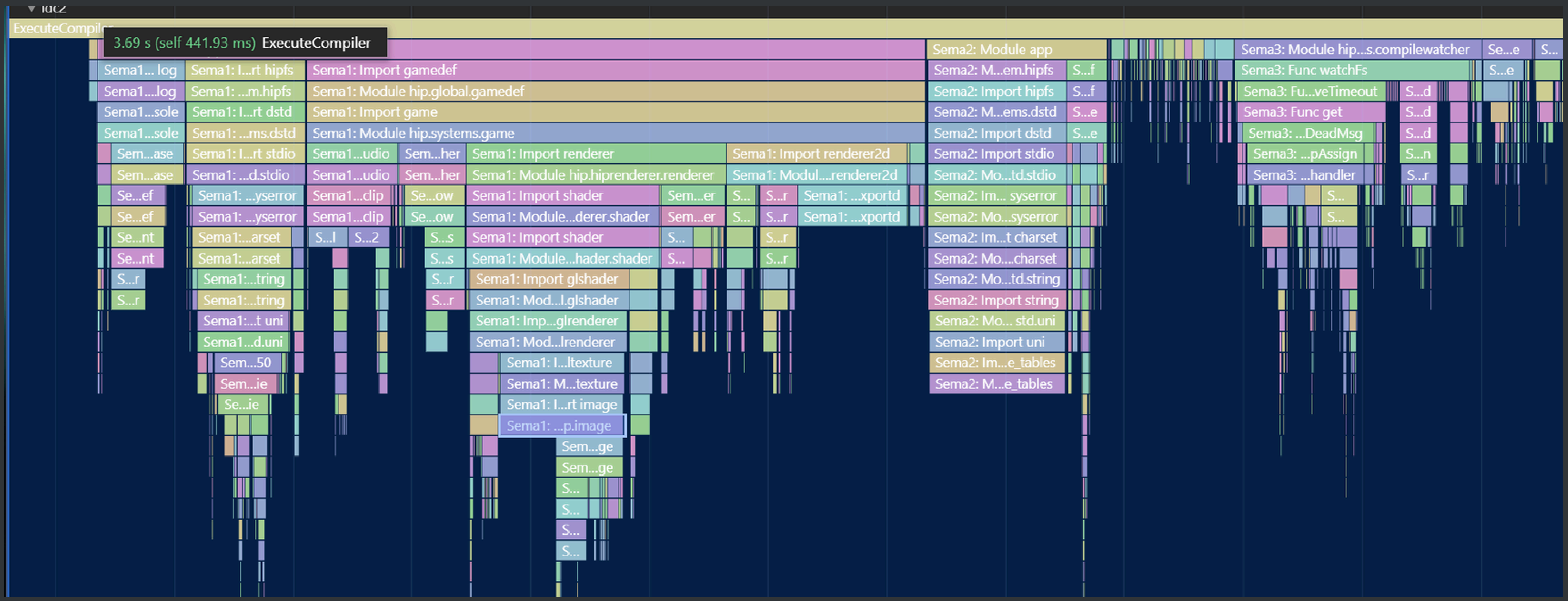
```
enum num = 50_000;
void main()
{
    int a = 10;
    static foreach(i; 0..num)
    {
        a+= 10;
    }
}
```

- Static foreach with simple addition, no variable creation
- 1.05 seconds on full compilation time
- 500ms on Sema3 (as reparse has to occur)
- 500ms on codegen
- As you can see, again, codegen time increased linearly with Sema3



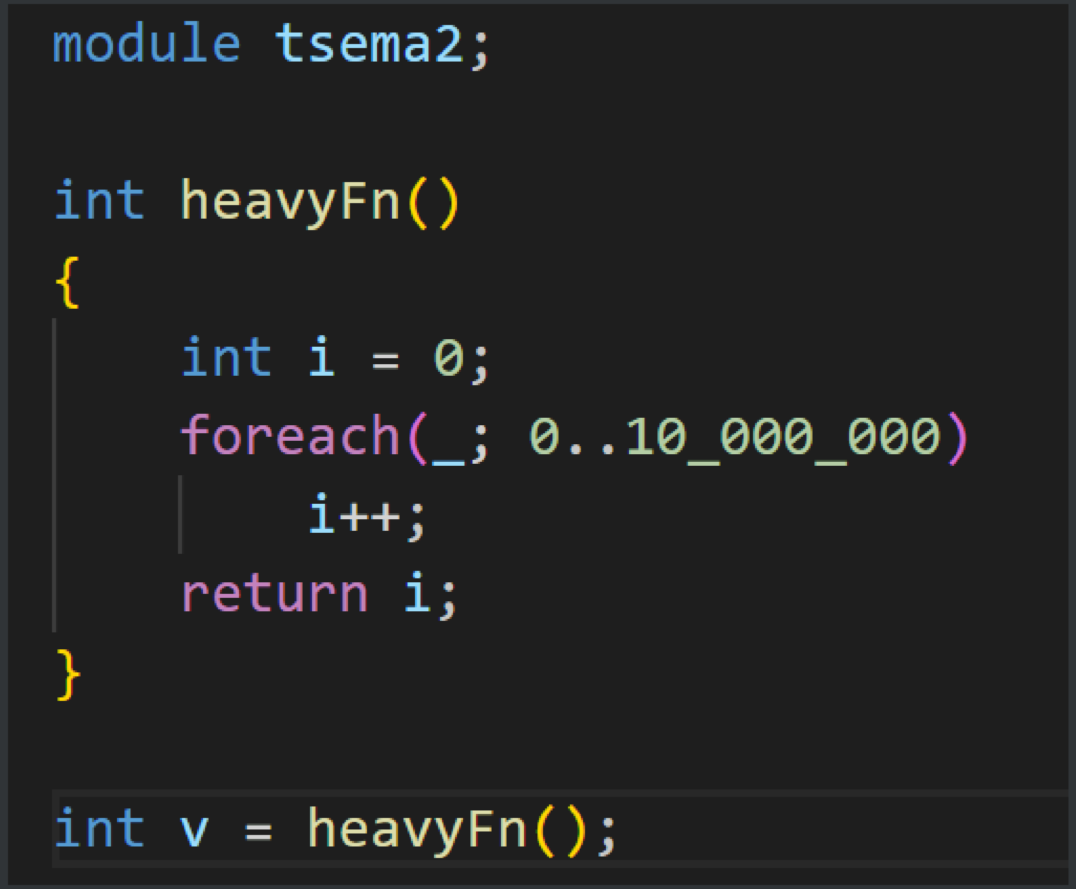
Optimization Case

- Hipreme Engine will be used as an example
- Build takes 3.7 seconds for only the main module
- This is how it was before any optimization was done

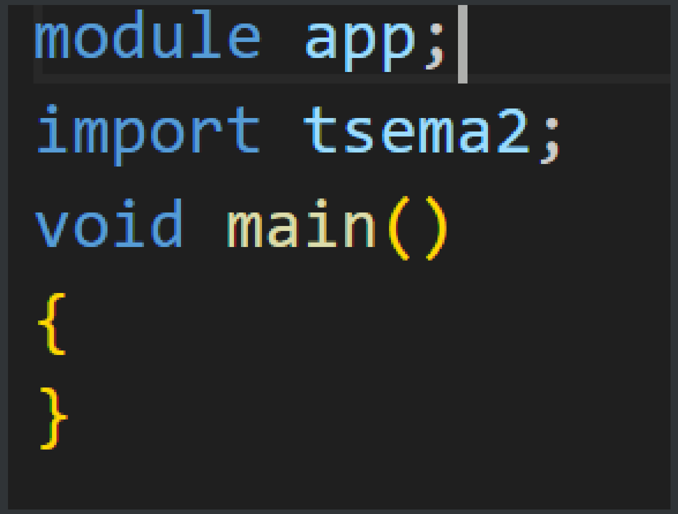


Understanding Semantic 1 problem

Separate Compilation: Rettriggers int v = heavyFn()

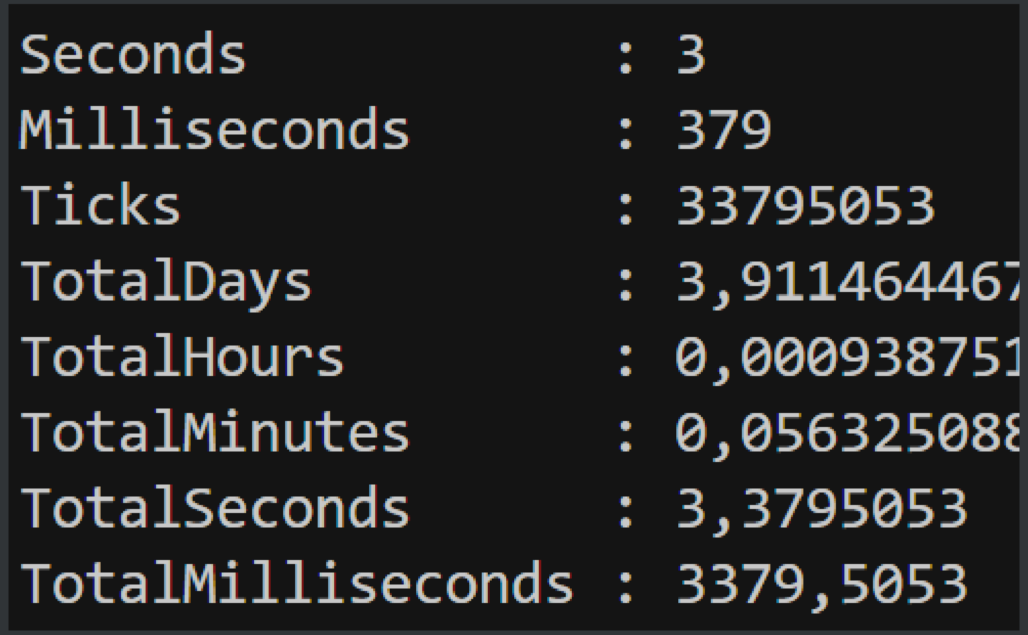


```d
module tsema2;

int heavyFn()
{
    int i = 0;
    foreach(_; 0..10_000_000)
        i++;
    return i;
}

int v = heavyFn();
```



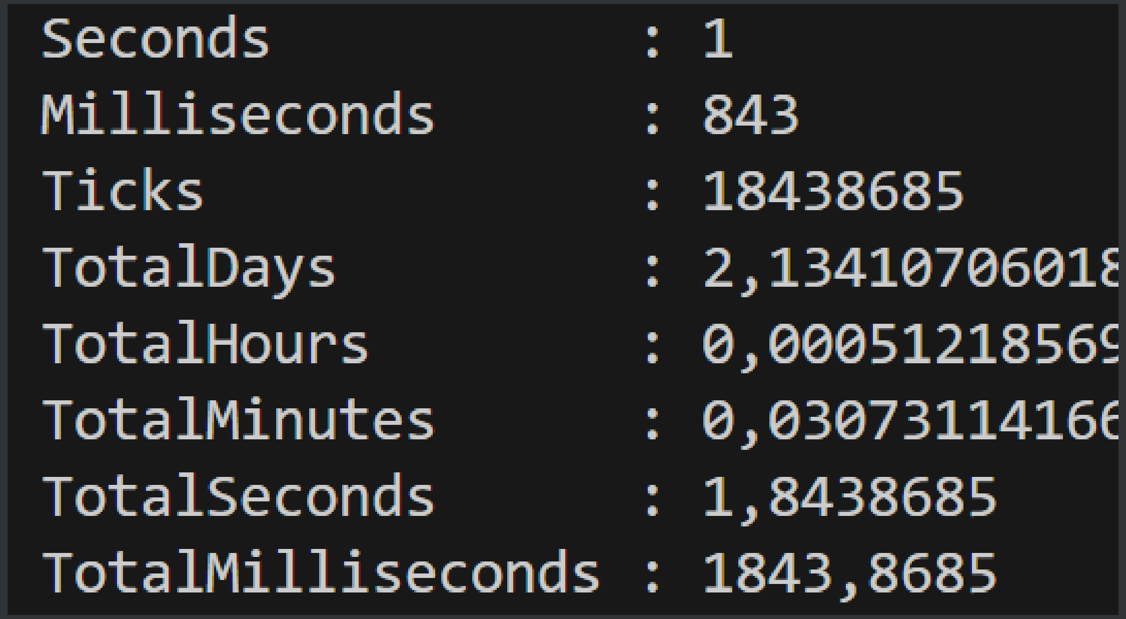
```d
module app;
import tsema2;
void main()
{
}
```

Separate



```
Seconds           : 3
Milliseconds      : 379
Ticks             : 33795053
TotalDays         : 3,911464467
TotalHours        : 0,000938751
TotalMinutes      : 0,05632508
TotalSeconds      : 3,3795053
TotalMilliseconds : 3379,5053
```

All at once



```
Seconds           : 1
Milliseconds      : 843
Ticks             : 18438685
TotalDays         : 2,13410706018
TotalHours        : 0,00051218569
TotalMinutes      : 0,03073114166
TotalSeconds      : 1,8438685
TotalMilliseconds : 1843,8685
```

Each dependency importing a module containing CTFE , will have to reevaluate, making dub unsuitable for those cases.

Semantic 1 - Solving Semantic 2 trigger

- Use static this() for runtime initialization
- Use a runtime getter

```d
private int _v;
ref int v()
{
    if(!_v) _v = heavyFn();
    return _v;
}
```

```d
int heavyFn()
{
    int i = 0;
    foreach(_; 0..10_000_000)
        i++;
    return i;
}

int v;
static this()
{
    v = heavyFn();
}
```

- Reduced separate compilation time by 3 seconds.
- All at once is now only 20ms faster on a full rebuild.

Separate

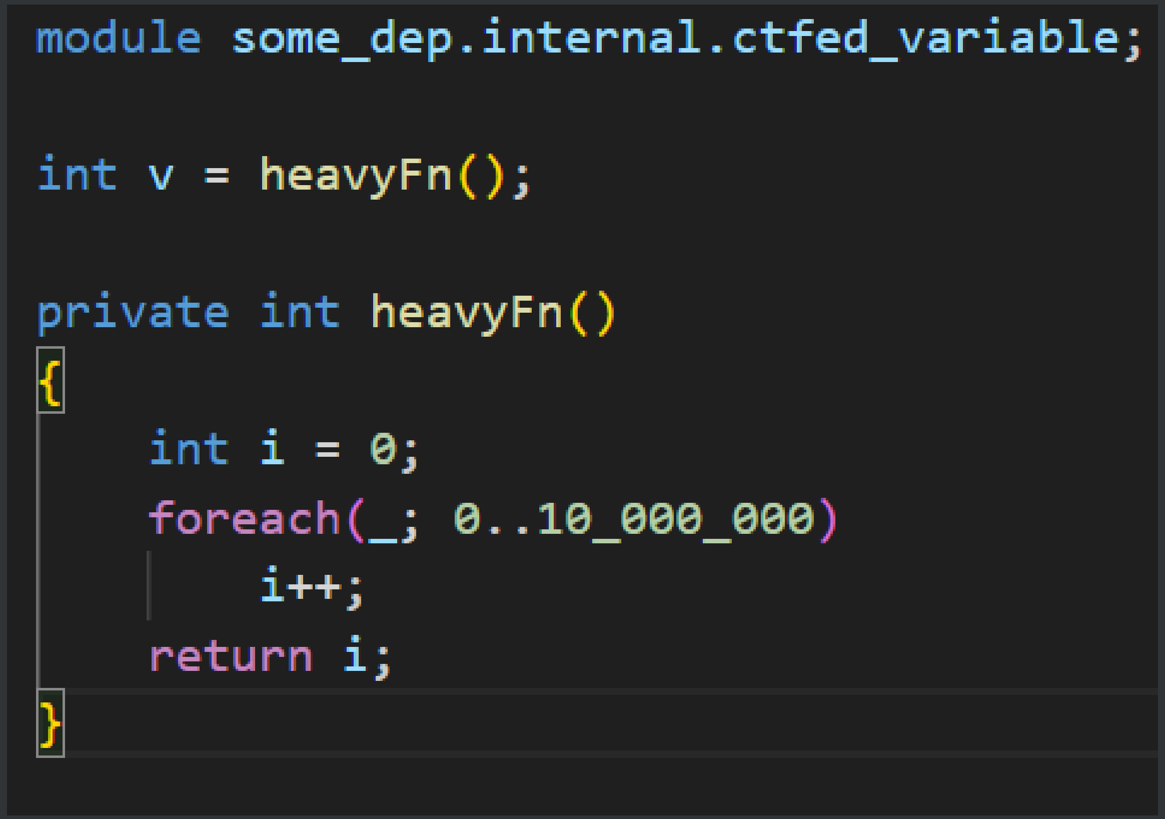
```
Seconds           : 0
Milliseconds      : 339
Ticks             : 3391768
TotalDays         : 3,9256574
TotalHours        : 9,4215777
TotalMinutes      : 0,0056529
TotalSeconds      : 0,3391768
TotalMilliseconds : 339,1768
```

All at once

```
Seconds           : 0
Milliseconds      : 310
Ticks             : 3104359
TotalDays         : 3,5930081
TotalHours        : 8,62321944
TotalMinutes      : 0,00517393
TotalSeconds      : 0,3104359
TotalMilliseconds : 310,4359
```

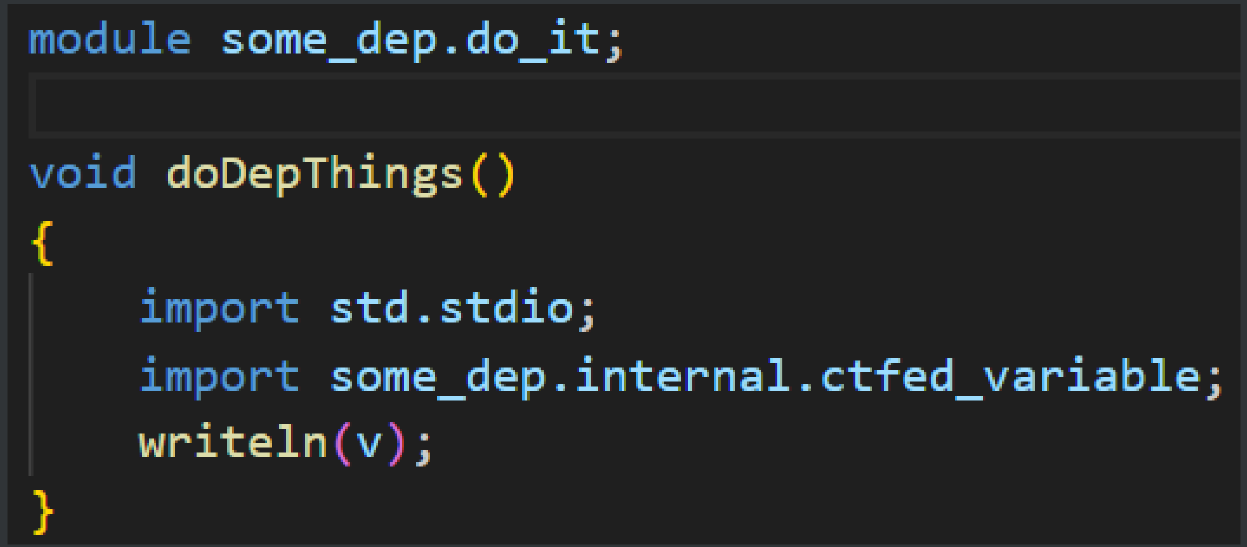
Semantic 1 - Private Dependencies

If you don’t want to cheat and still keep doing the CTFE, private dependencies is the way to go

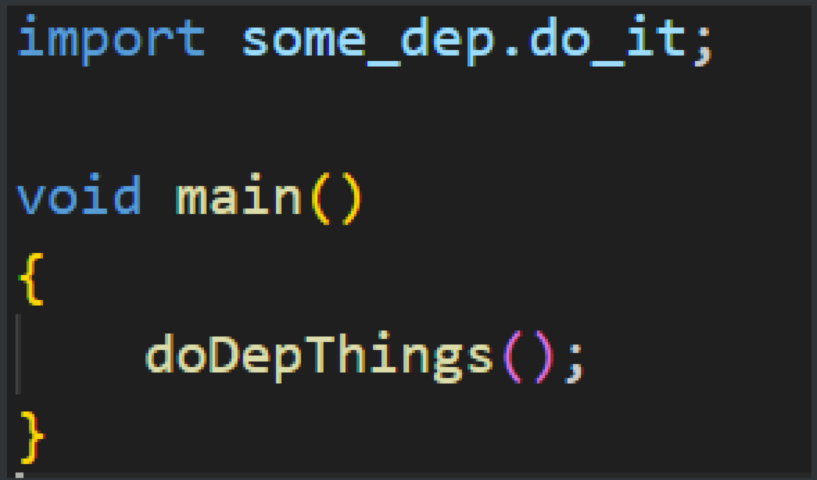


```d
module some_dep.internal.ctfed_variable;

int v = heavyFn();

private int heavyFn()
{
    int i = 0;
    foreach(_; 0..10_000_000)
        i++;
    return i;
}
```

- Create a module which is not imported by user
- Import it inside the function implementation
- Now, CTFE cost is paid only once

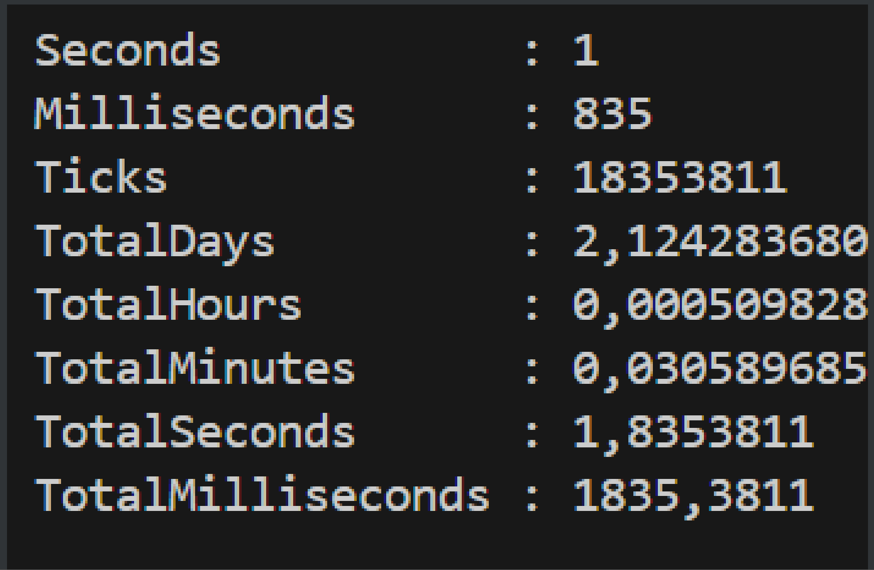


```d
module some_dep.do_it;

void doDepThings()
{
    import std.stdio;
    import some_dep.internal.ctfed_variable;
    writeln(v);
}
```



```d
import some_dep.do_it;

void main()
{
    doDepThings();
}
```

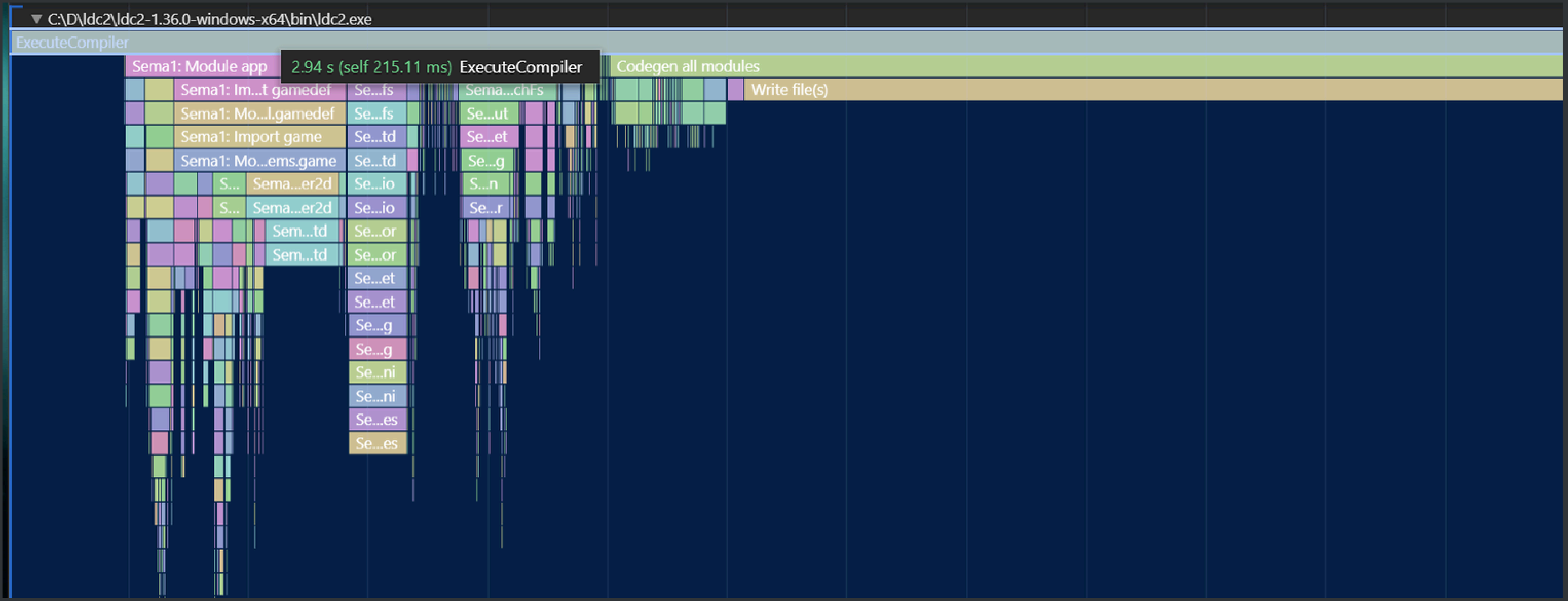
Separate



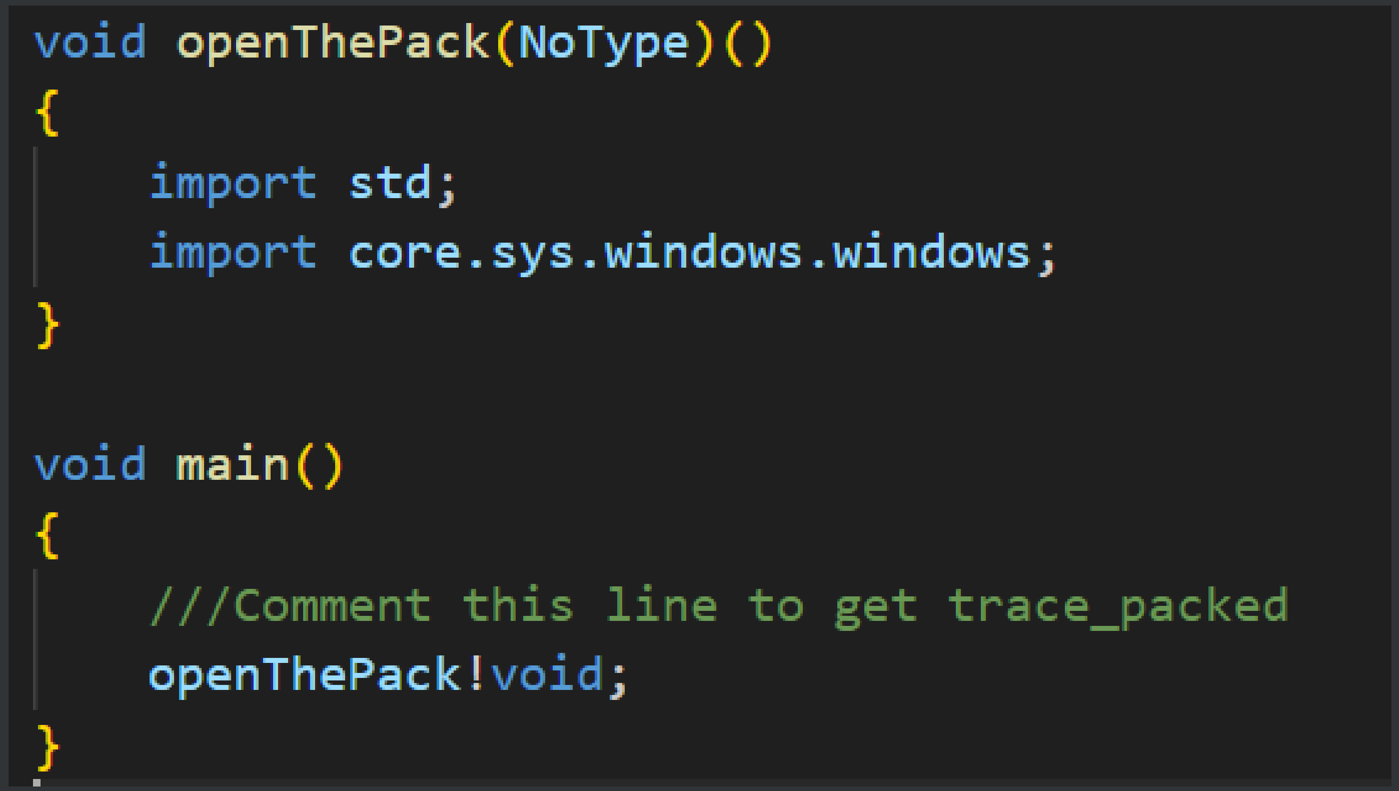
```
Seconds           : 1
Milliseconds      : 835
Ticks             : 18353811
TotalDays         : 2,124283680
TotalHours        : 0,000509828
TotalMinutes      : 0,030589685
TotalSeconds      : 1,8353811
TotalMilliseconds : 1835,3811
```

Result of applying Semantic 1 optimization

- Reduced build time from 3.7 seconds to 2.95 seconds
- Look how cleaner it is to understand what is causing bottlenecks now
- Keep in mind that Hipreme Engine already had a fair amount of build time optimization

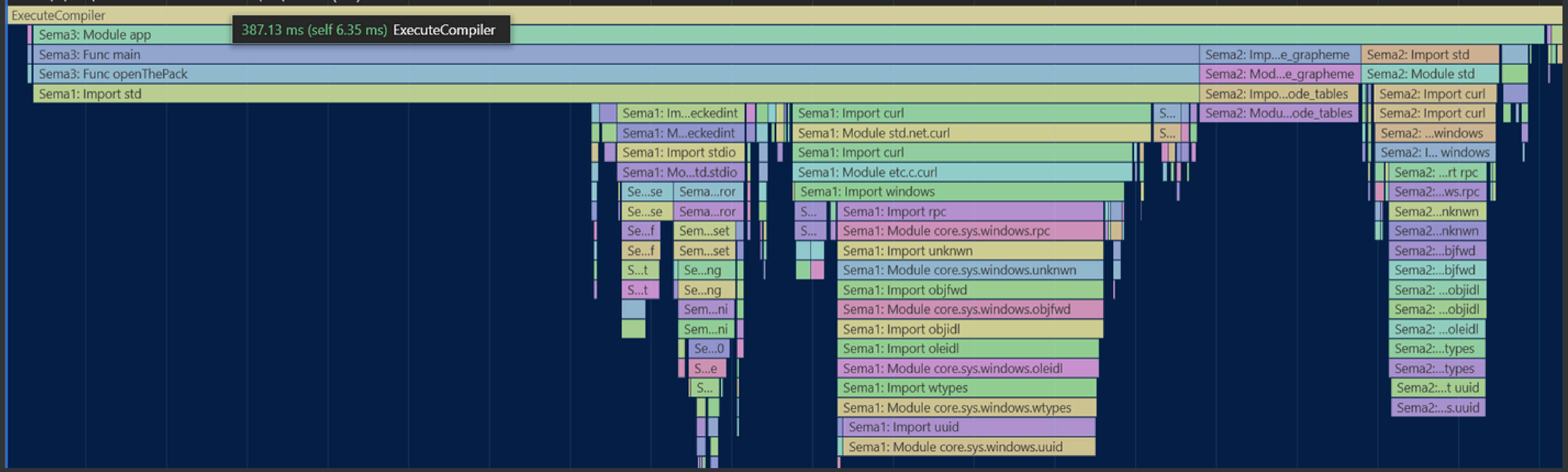


Semantic 3 - Technique for improvement



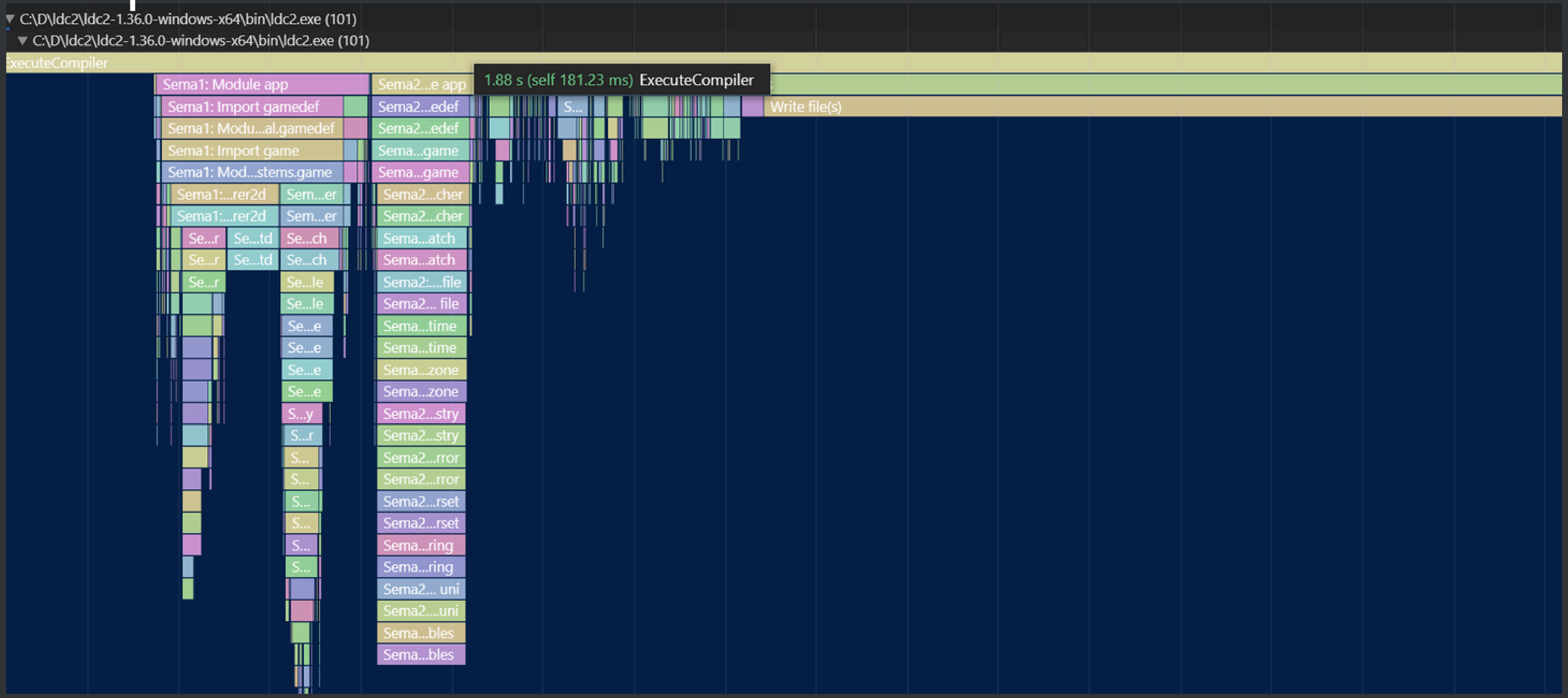
```
void openThePack(NoType)()
{
    import std;
    import core.sys.windows.windows;
}

void main()
{
    ///Comment this line to get trace_packed
    openThePack!void;
}
```

- A case of import being evaluated on template expansion
- Since Sema1 is very small, this module is perfect to be imported
- That is a case of triggering a full Sema 1, 2 and 3



Result of applying Semantic 3 optimization

- Compilation went from 2.95 seconds to 1.9 seconds
- Overall result was 3.69 to 1.88 - Cutting almost 50% of the time required
- Graph is even clearer now, making it even easier to improve

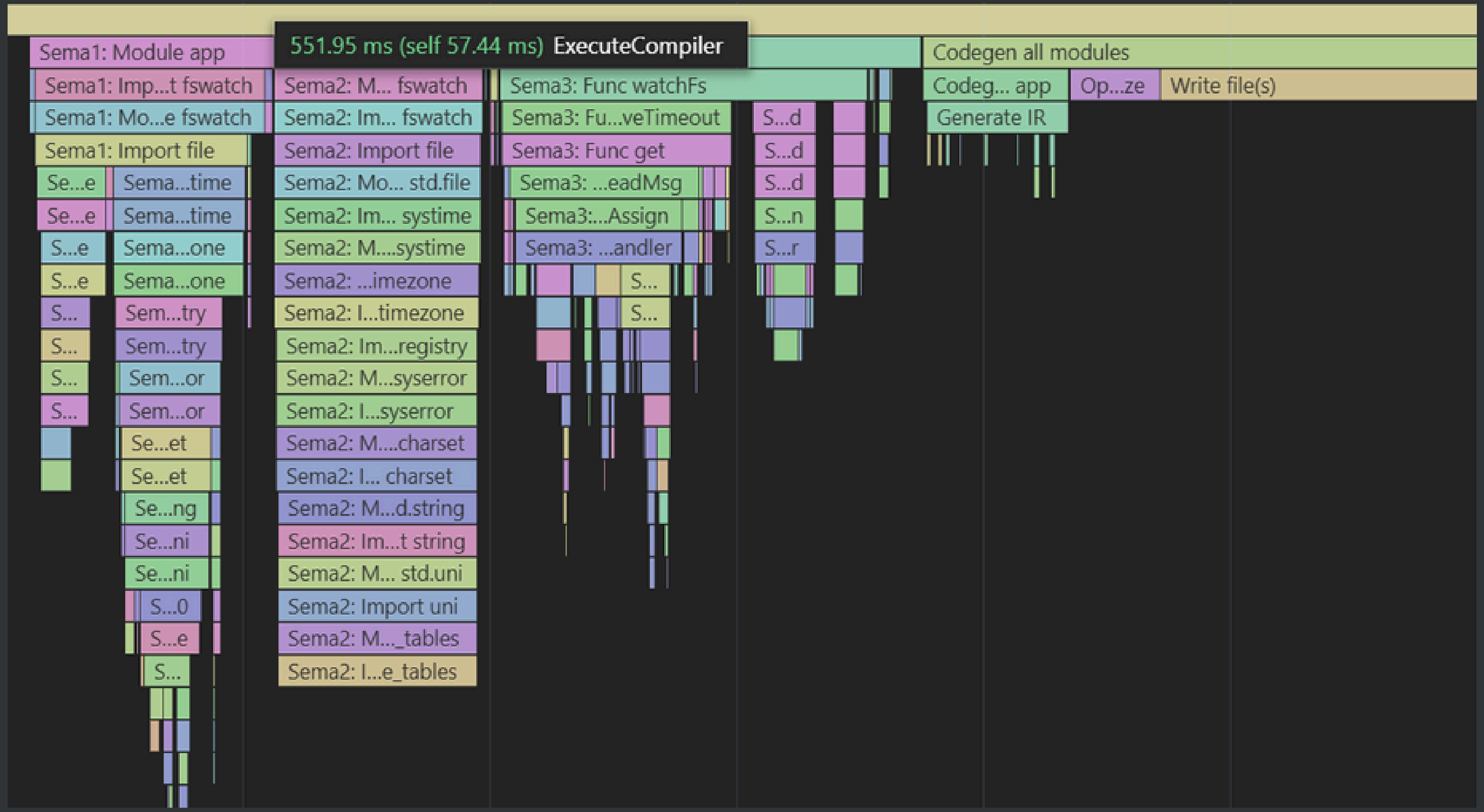


REDUB - HIPREME ENGINE POST SEMA 3 OPTIMIZATION

Separate Compilation Optimization

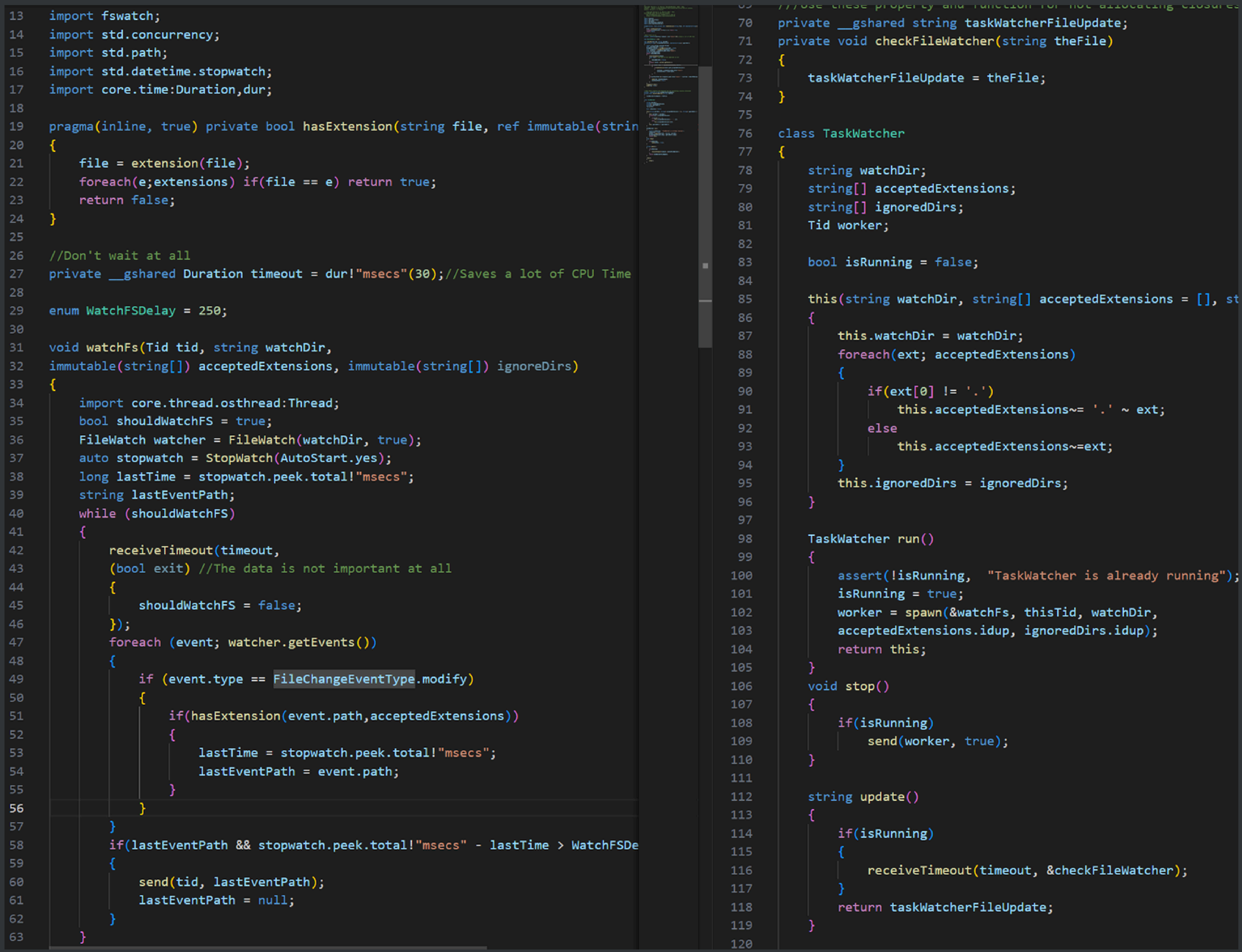
Case: You've just added a new library on your root project. This library is taking a lot of time to build [Remember that it's 550ms starting from 0ms]

Solution: Create a library which wraps that new library and makes it available a minimal API to be imported



TaskWatcher: A module which uses FSWatch for watching the filesystem and use Threads for creating no blocking operation + queue system

Separate Compilation Optimization

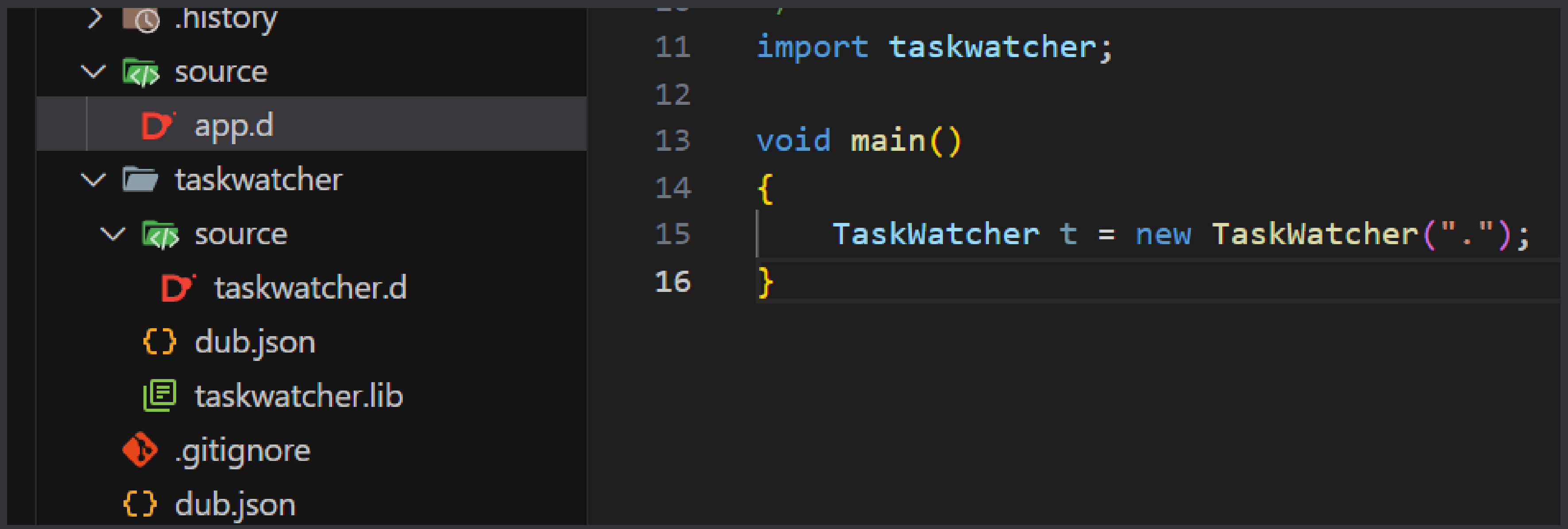


```d
import fswatch;
import std.concurrency;
import std.path;
import std.datetime.stopwatch;
import core.time:Duration,dur;

pragma(inline, true) private bool hasExtension(string file, ref immutable(strin
{
    file = extension(file);
    foreach(e;extensions) if(file == e) return true;
    return false;
}

//Don't wait at all
private __gshared Duration timeout = dur!"msecs"(30);//Saves a lot of CPU Time

enum WatchFSDelay = 250;

void watchFs(Tid tid, string watchDir,
immutable(string[]) acceptedExtensions, immutable(string[]) ignoreDirs)
{
    import core.thread.osthread:Thread;
    bool shouldWatchFS = true;
    FileWatch watcher = FileWatch(watchDir, true);
    auto stopwatch = StopWatch(AutoStart.yes);
    long lastTime = stopwatch.peek.total!"msecs";
    string lastEventPath;
    while (shouldWatchFS)
    {
        receiveTimeout(timeout,
        (bool exit) //The data is not important at all
        {
            shouldWatchFS = false;
        });
        foreach (event; watcher.getEvents())
        {
            if (event.type == FileChangeEventType.modify)
            {
                if(hasExtension(event.path,acceptedExtensions))
                {
                    lastTime = stopwatch.peek.total!"msecs";
                    lastEventPath = event.path;
                }
            }
        }
        if(lastEventPath && stopwatch.peek.total!"msecs" - lastTime > WatchFSDe
        {
            send(tid, lastEventPath);
            lastEventPath = null;
        }
    }
```

```d
///Use these property and function for not allocating closures
private __gshared string taskWatcherFileUpdate;
private void checkFileWatcher(string theFile)
{
    taskWatcherFileUpdate = theFile;
}

class TaskWatcher
{
    string watchDir;
    string[] acceptedExtensions;
    string[] ignoredDirs;
    Tid worker;

    bool isRunning = false;

    this(string watchDir, string[] acceptedExtensions = [], st
    {
        this.watchDir = watchDir;
        foreach(ext; acceptedExtensions)
        {
            if(ext[0] != '.')
                this.acceptedExtensions~= '.' ~ ext;
            else
                this.acceptedExtensions~=ext;
        }
        this.ignoredDirs = ignoredDirs;
    }

    TaskWatcher run()
    {
        assert(!isRunning,  "TaskWatcher is already running");
        isRunning = true;
        worker = spawn(&watchFs, thisTid, watchDir,
        acceptedExtensions.idup, ignoredDirs.idup);
        return this;
    }
    void stop()
    {
        if(isRunning)
            send(worker, true);
    }

    string update()
    {
        if(isRunning)
        {
            receiveTimeout(timeout, &checkFileWatcher);
        }
        return taskWatcherFileUpdate;
    }

```

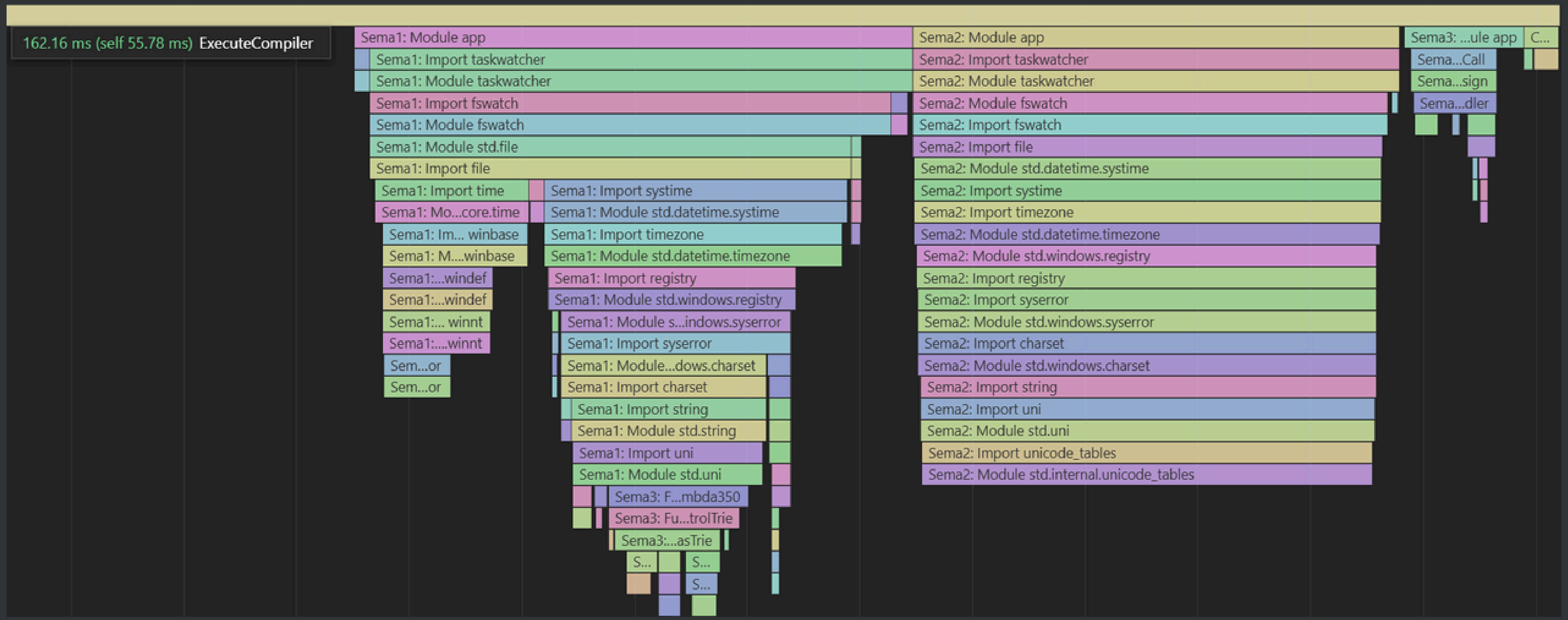
Separate Compilation Optimization

Result of putting your module that imports fswatch on a library



```
> .history
v source
    app.d
v taskwatcher
  v source
      taskwatcher.d
    dub.json
    taskwatcher.lib
  .gitignore
  dub.json
```

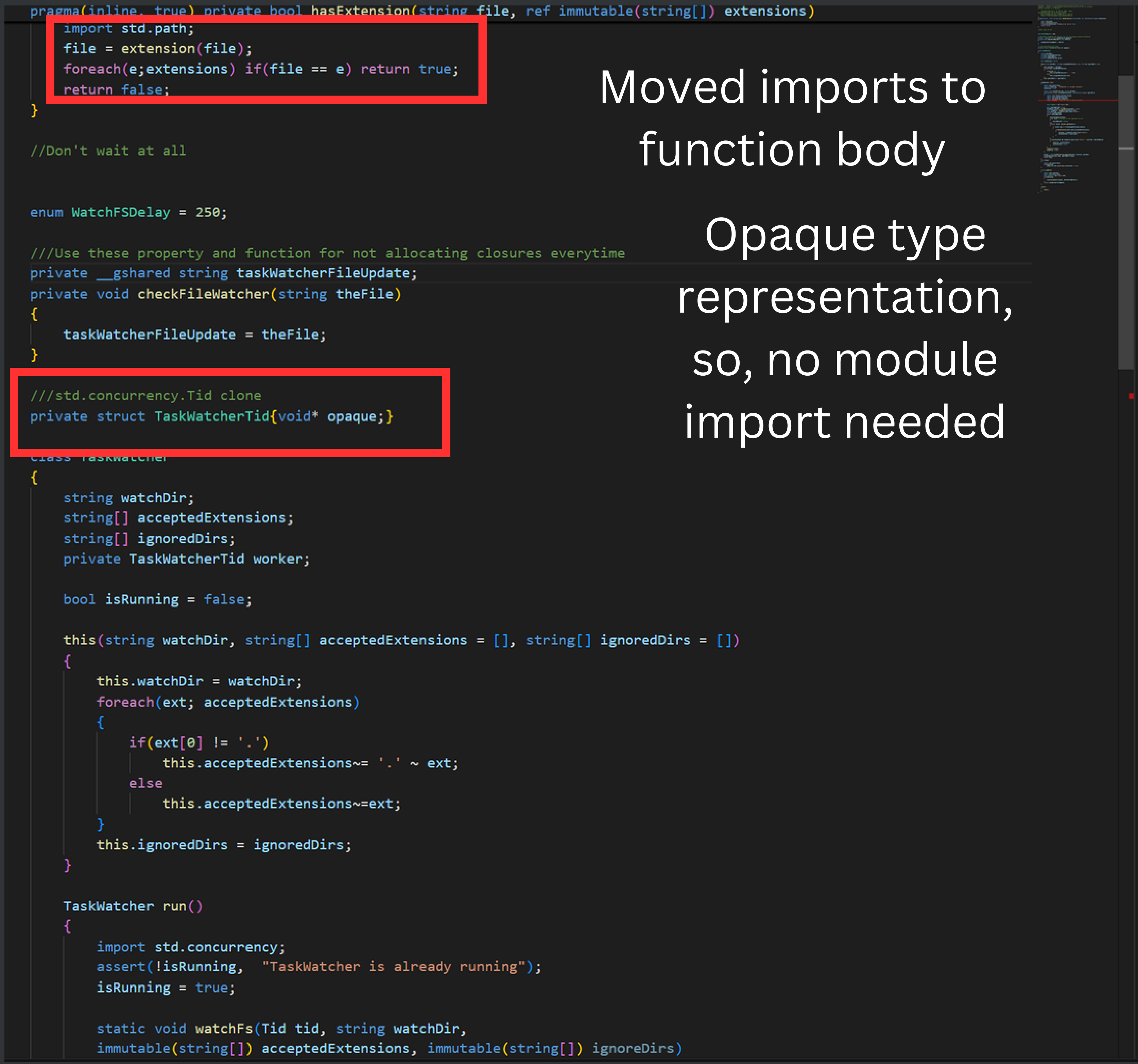
```d
import taskwatcher;

void main()
{
    TaskWatcher t = new TaskWatcher(".");
}
```



- **Speed Gain:** By simply making it inside a separate dependency, root goes from 500ms to 162ms.
- **Paralell Compilation:** You'll also get parallel compilation
- **Every Unit Matters:** When parallel, it is important to keep every compilation unit small to keep the overall compilation fast

Applying Optimization Concepts

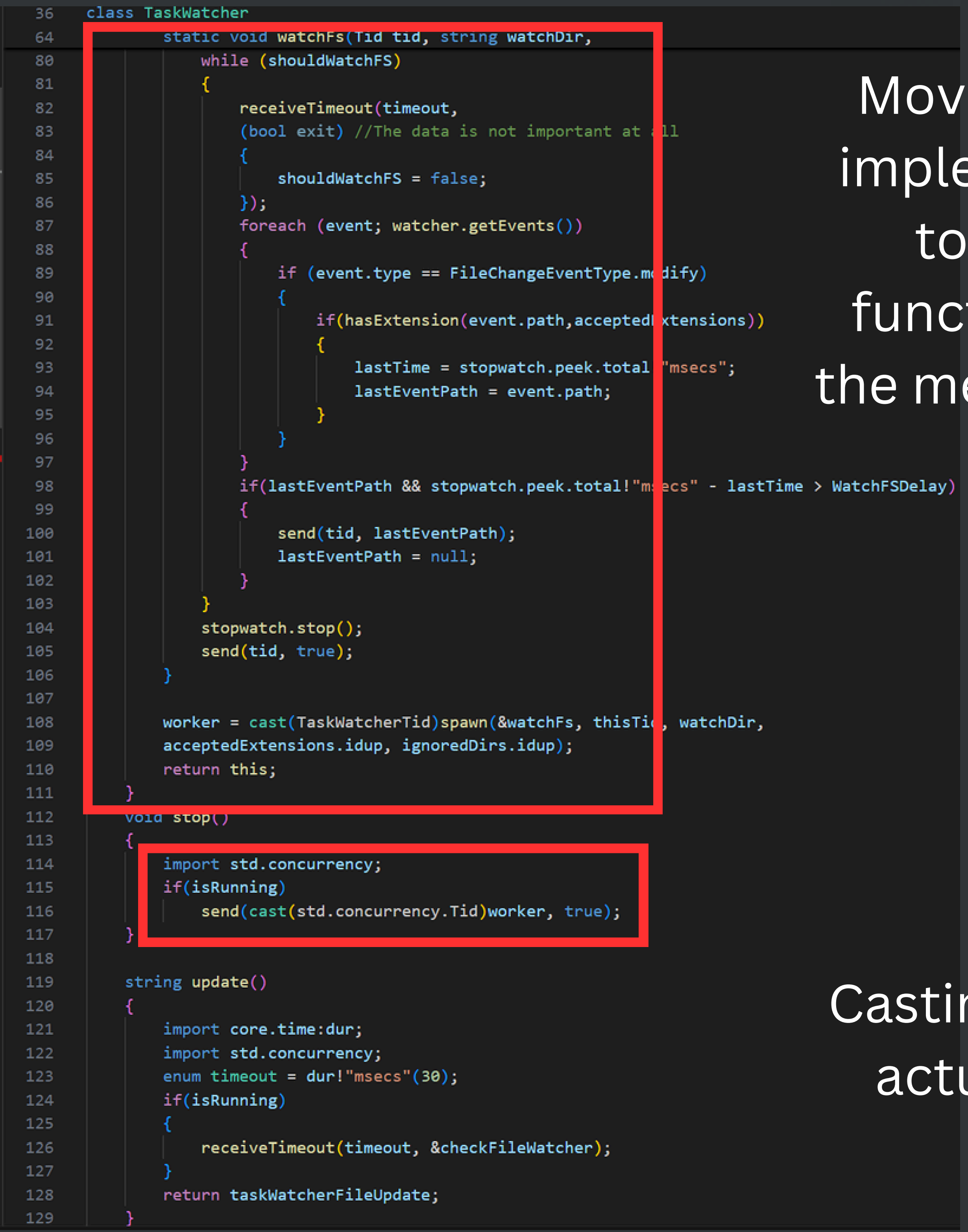
That result was pretty good, but let us improve it



```d
pragma(inline, true) private bool hasExtension(string file, ref immutable(string[]) extensions)
    import std.path;
    file = extension(file);
    foreach(e;extensions) if(file == e) return true;
    return false;
}

//Don't wait at all


enum WatchFSDelay = 250;

///Use these property and function for not allocating closures everytime
private __gshared string taskWatcherFileUpdate;
private void checkFileWatcher(string theFile)
{
    taskWatcherFileUpdate = theFile;
}

///std.concurrency.Tid clone
private struct TaskWatcherTid{void* opaque;}
class TaskWatcher
{
    string watchDir;
    string[] acceptedExtensions;
    string[] ignoredDirs;
    private TaskWatcherTid worker;

    bool isRunning = false;

    this(string watchDir, string[] acceptedExtensions = [], string[] ignoredDirs = [])
    {
        this.watchDir = watchDir;
        foreach(ext; acceptedExtensions)
        {
            if(ext[0] != '.')
                this.acceptedExtensions~= '.' ~ ext;
            else
                this.acceptedExtensions~=ext;
        }
        this.ignoredDirs = ignoredDirs;
    }

    TaskWatcher run()
    {
        import std.concurrency;
        assert(!isRunning,  "TaskWatcher is already running");
        isRunning = true;

        static void watchFs(Tid tid, string watchDir,
        immutable(string[]) acceptedExtensions, immutable(string[]) ignoreDirs)
```

Moved imports to function body

Opaque type representation, so, no module import needed

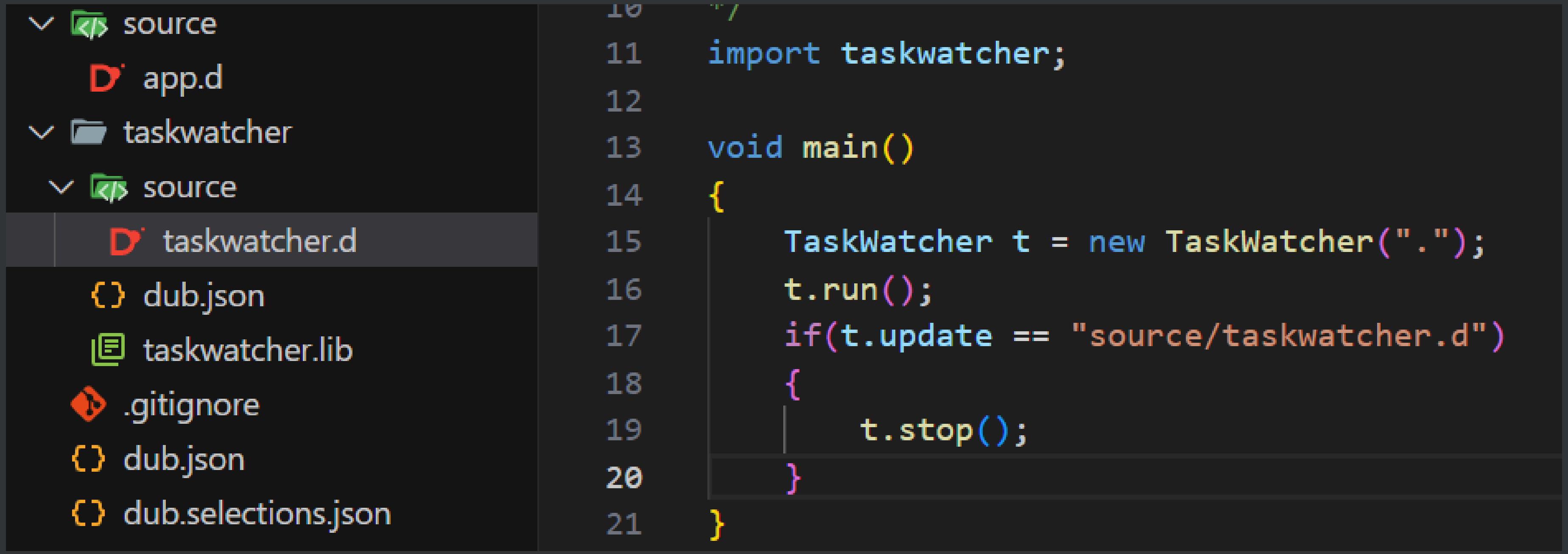


```d
class TaskWatcher
        static void watchFs(Tid tid, string watchDir,
            while (shouldWatchFS)
            {
                receiveTimeout(timeout,
                (bool exit) //The data is not important at all
                {
                    shouldWatchFS = false;
                });
                foreach (event; watcher.getEvents())
                {
                    if (event.type == FileChangeEventType.modify)
                    {
                        if(hasExtension(event.path,acceptedExtensions))
                        {
                            lastTime = stopwatch.peek.total!"msecs";
                            lastEventPath = event.path;
                        }
                    }
                }
                if(lastEventPath && stopwatch.peek.total!"msecs" - lastTime > WatchFSDelay)
                {
                    send(tid, lastEventPath);
                    lastEventPath = null;
                }
            }
            stopwatch.stop();
            send(tid, true);
        }

        worker = cast(TaskWatcherTid)spawn(&watchFs, thisTid, watchDir,
        acceptedExtensions.idup, ignoredDirs.idup);
        return this;
    }
    void stop()
    {
        import std.concurrency;
        if(isRunning)
            send(cast(std.concurrency.Tid)worker, true);
    }

    string update()
    {
        import core.time:dur;
        import std.concurrency;
        enum timeout = dur!"msecs"(30);
        if(isRunning)
        {
            receiveTimeout(timeout, &checkFileWatcher);
        }
        return taskWatcherFileUpdate;
    }
```

Moved private implementation to a static function inside the method body

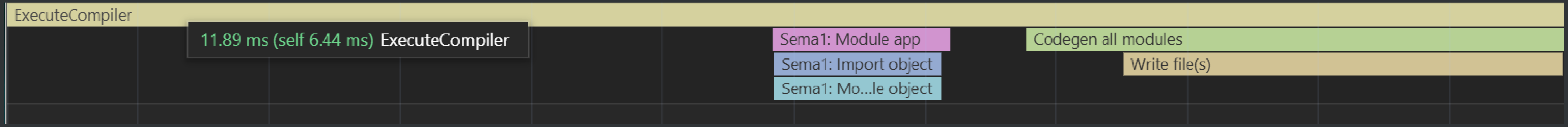
Castings to the actual type

Results of Applying Optimization Concepts



```
source
    app.d
taskwatcher
    source
        taskwatcher.d
    dub.json
    taskwatcher.lib
.gitignore
dub.json
dub.selections.json
```

```d
*/
import taskwatcher;

void main()
{
    TaskWatcher t = new TaskWatcher(".");
    t.run();
    if(t.update == "source/taskwatcher.d")
    {
        t.stop();
    }
}
```

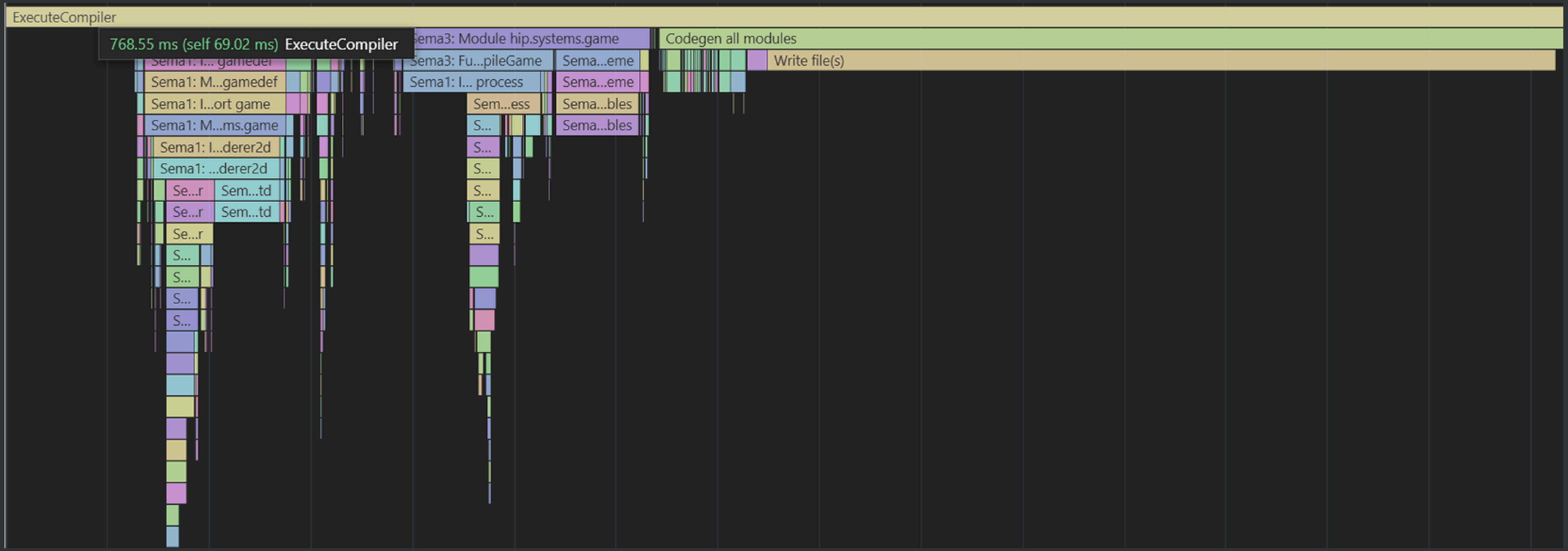


```
ExecuteCompiler
11.89 ms (self 6.44 ms) ExecuteCompiler
Sema1: Module app
Sema1: Import object
Sema1: Mo...le object
Codegen all modules
Write file(s)
```

The end result is that the main module gone from 550ms to 12ms. Now, that module is completely free of any kind of impact from the imported dependency and user implementation. The old module still costs 550ms, but the user one can be used with completely freedom.

Fast Forward of constant applying principles

- The results of constant application of that is the reduction from 3.69 to 0.76 - Cutting almost 80% of the time required
- Now, it is small enough for not needing to focus on that.



On the Future of D compiler

- ~~Parallel Compilation~~: Although it could be great, it would not greatly improve UX on iteration, since I doubt it's speed would increase much.
- **Compiler as a library**: Minimal amount needed, for example, given a path, return all the modules that are imported (and their respective path), this would be enough to supply only the required files to build and then tool could reuse **.obj**
- **Compiler Daemon:** When having compiler as a library being used, an initial compiler daemon could be achieved by library users. The tool could supply which file had changed while the library supply its imports, so, the tool could know which files to build.
- **Parallel Generation**: The semantic passes may be single threaded, since it is how it stores information on what to build, but maybe the generation itself could be parallelized

The other way around: LDC has a way to store it's IR on PC. This greatly increases the compilation speed on the next run. DMD could do something similar, like, the user supplying object files directory and which files had changes, so, DMD could reuse those existing object files and ignore code generation of the ones which weren't supplied as changed and aren't found on the dependency chain

On the Future of REDUB

- **redub.json:** When having that file present, it would use instead of dub.json, providing better flexibility and a migration path for new features, no new format
- **Feature System:** HBuild from Hipreme Engine but specified in the declaration
- **dub.selections.json generation?**: I don't really like that idea, this could heavily depend on how messy it gets with users
- **Replacing Dub:** I don't really care or want that it replaces dub. This would greatly reduce it's autonomy
- **Redub Daemon:** Redub may use a filewatcher to build on project changes, but it's viability and integration with compiler library is still being considered

QUESTIONS ?