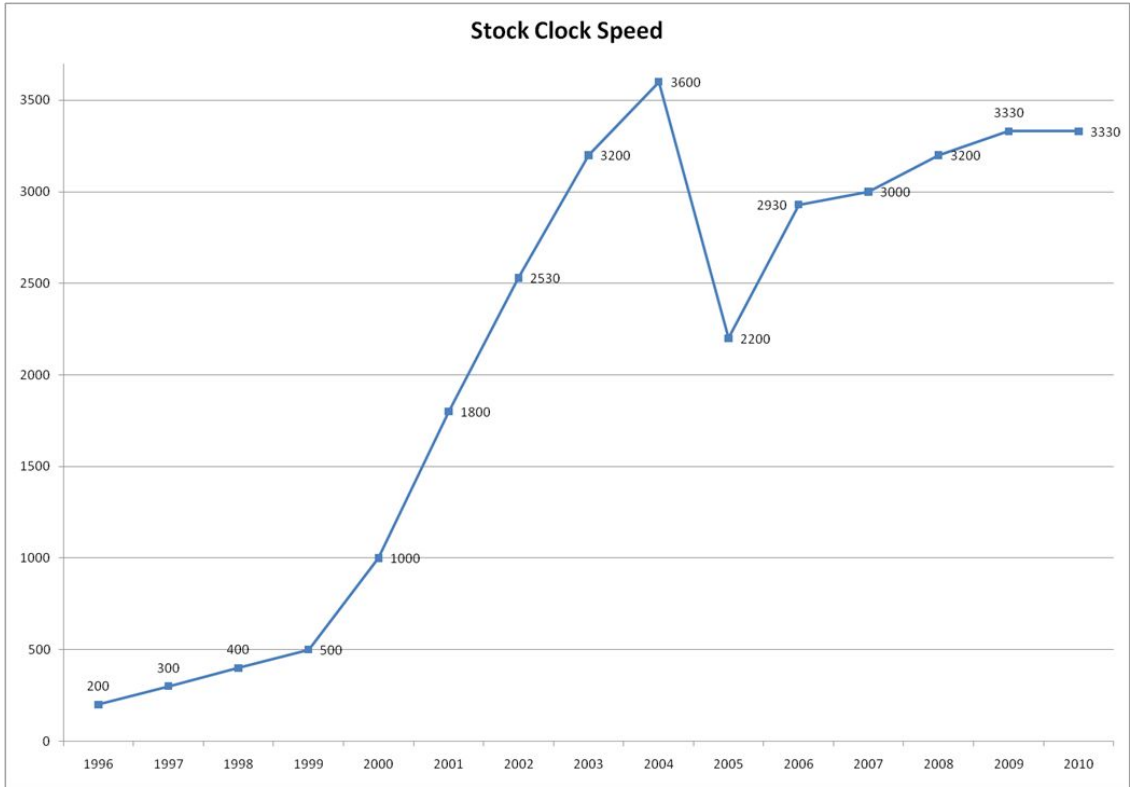
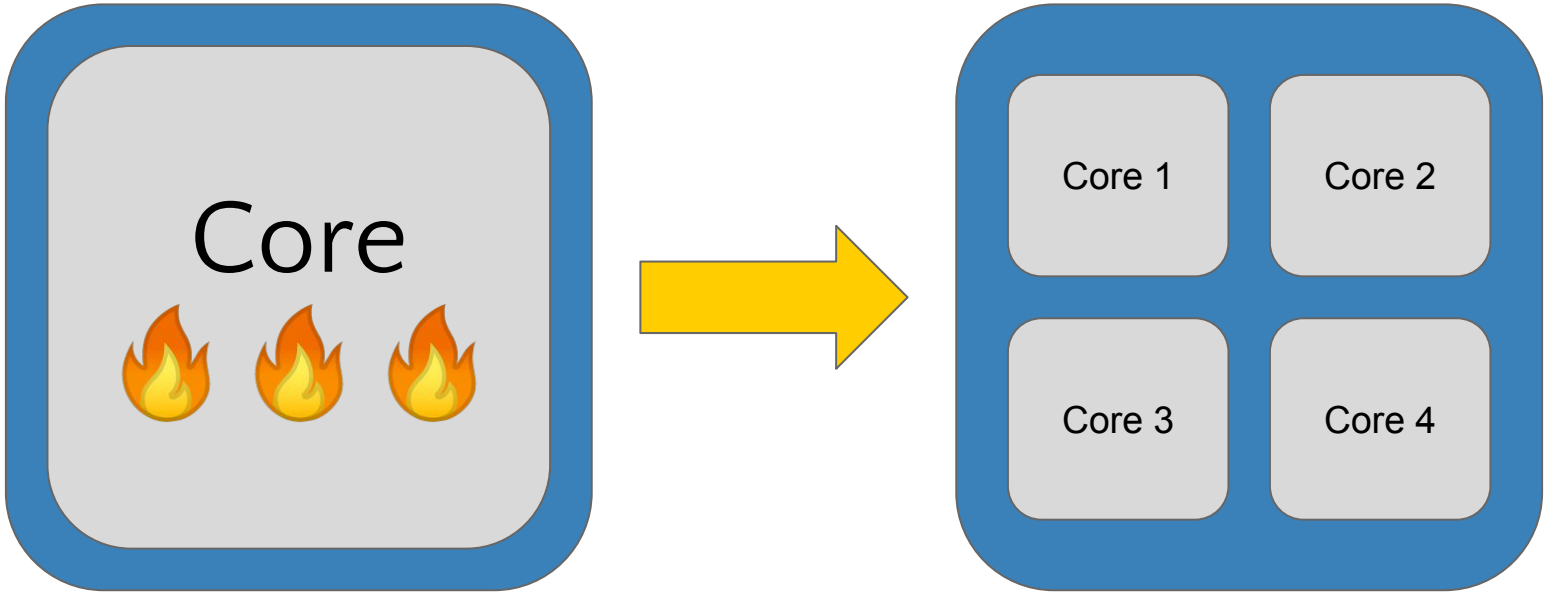
Making Games Start Fast

A Story About Concurrency

Once upon a time, in the early 2000s...



We had a small clock speed cap issue...



But the solution was easy!

The End

And then I woke up

Hello!

I am Mathieu Ropert

I'm a Tech Lead at Paradox Development Studio where I make Hearts of Iron IV, Stellaris and more.

You can reach me at:

 mro@puchiko.net

 @MatRopert

 https://mropert.github.io

About this talk

- Threads
- Locks…
- … and how to avoid them
- Investigating threading efficiency

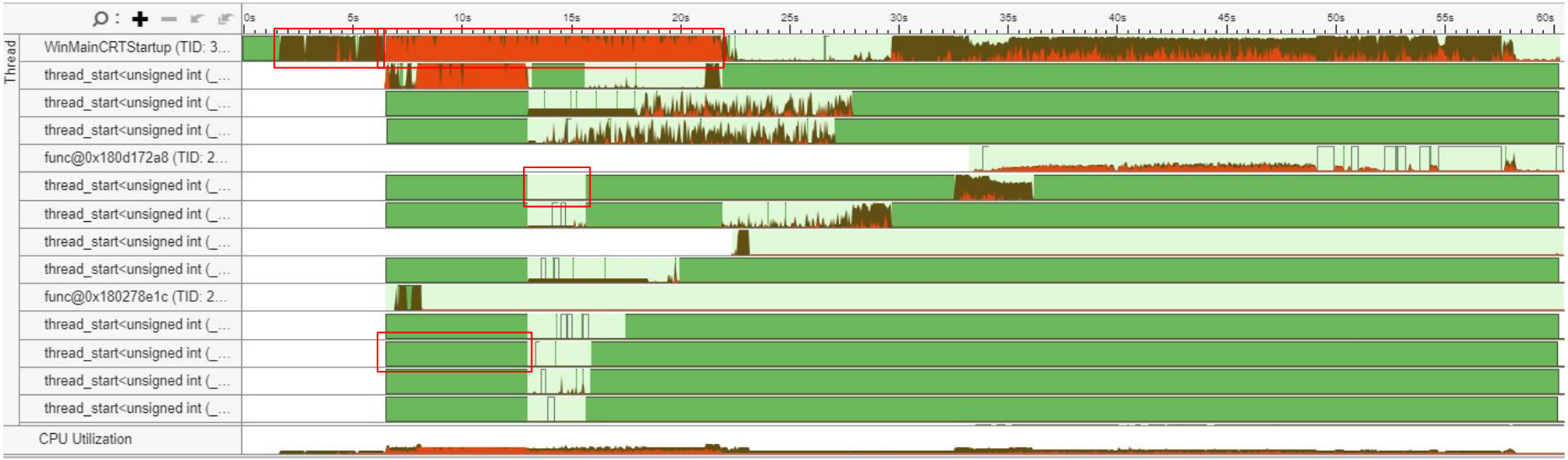
1 Showcase

Time to start a game!

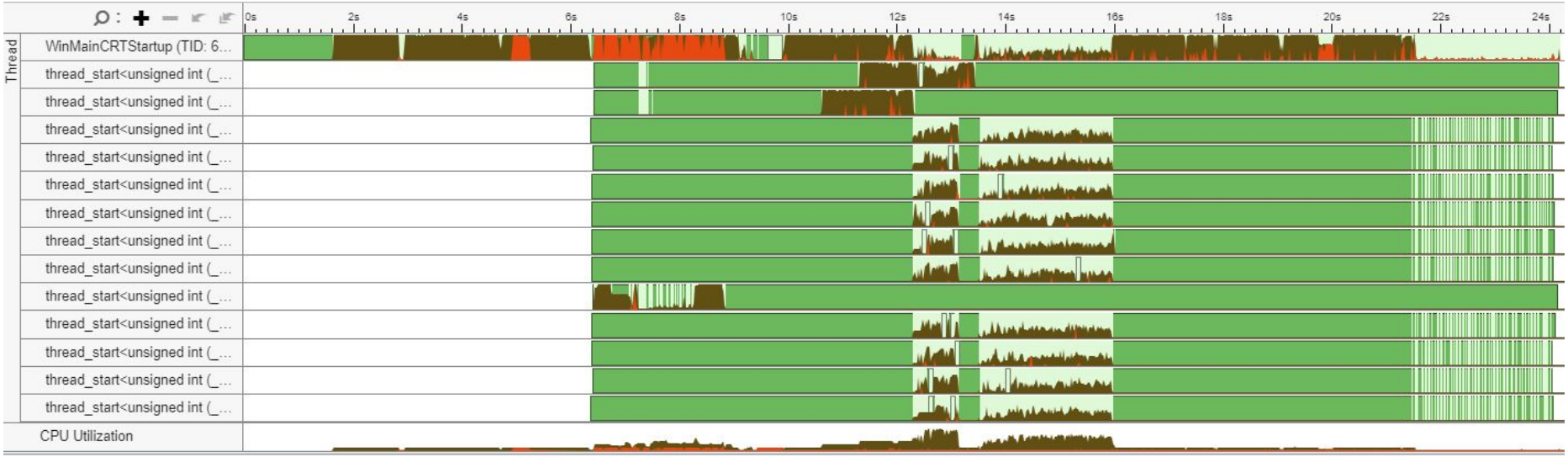
Demo Time!

Some Metrics

- Stellaris 2.7 starts in 54s
- Stellaris 2.8 beta starts in 21s
- Same amount of work
- Both rely on multithreading



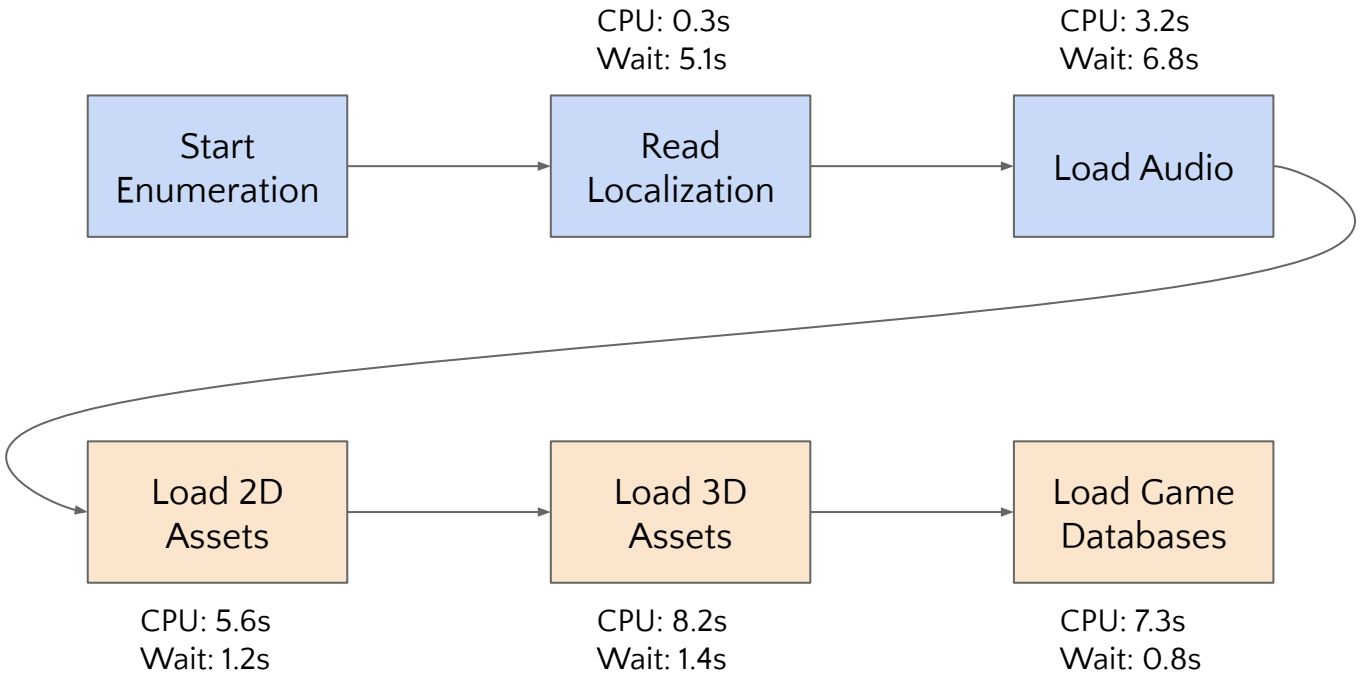
2.7 (Old) Startup CPU Usage



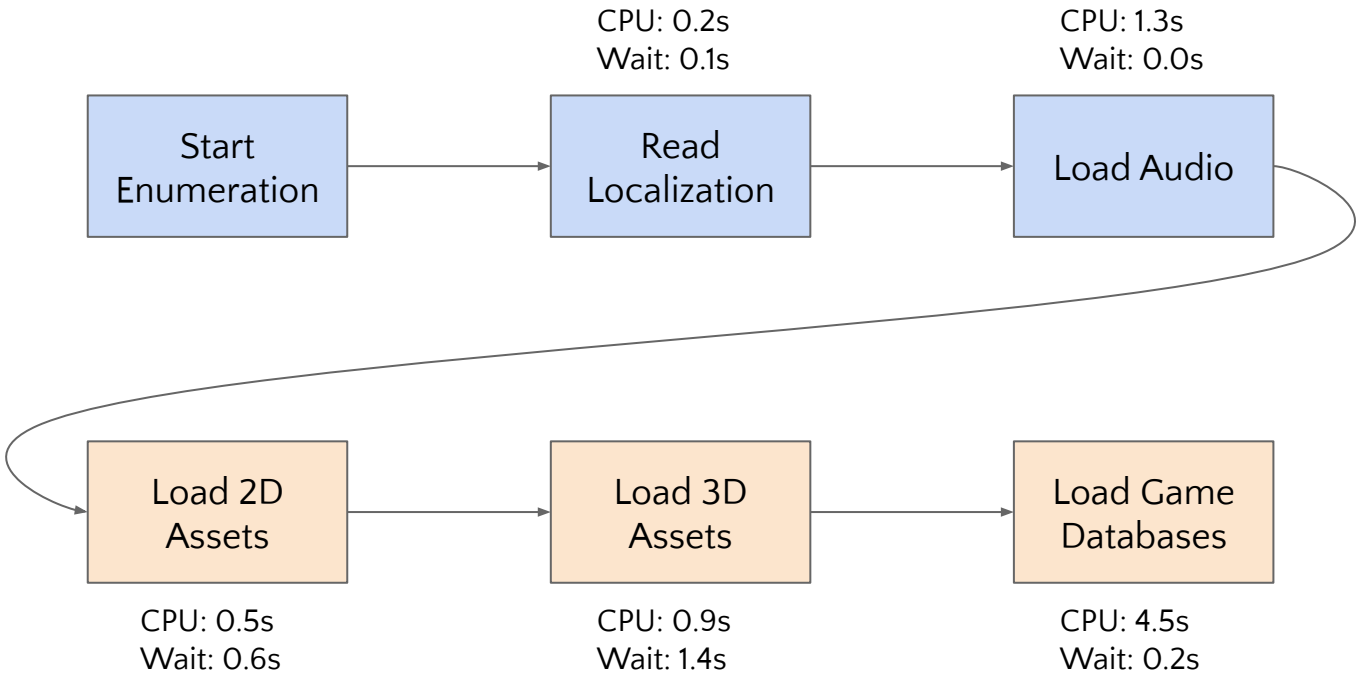
2.8 (New) Startup CPU Usage

Startup Breakdown

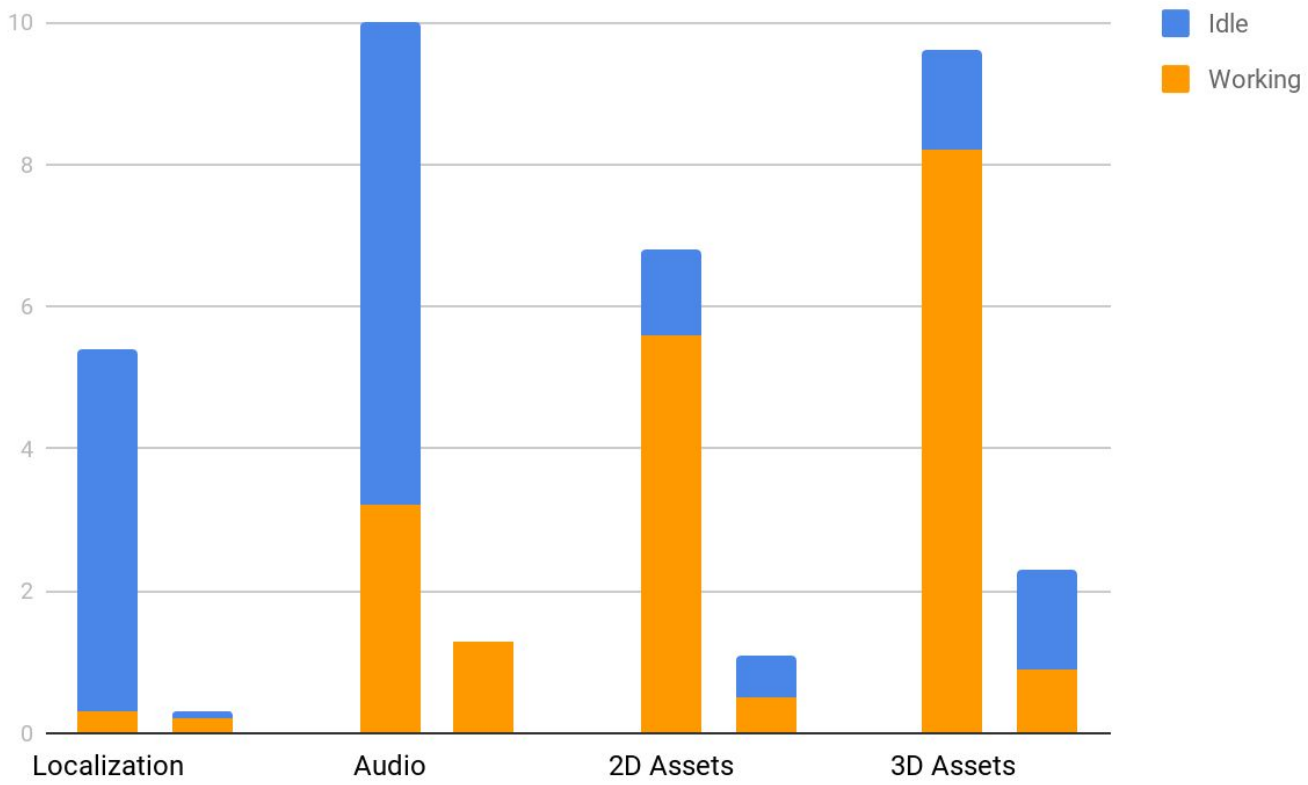
- Enumerate asset files
- Read localization
- Load textures, models and audio
- Load game rules & databases



2.7 (Old) Startup Profile



2.8 (New) Startup Profile



Side by Side Comparison

High CPU Time

- Single threaded code
- Inefficient algorithms
- Branch misprediction, cache misses
- Spin locks

High CPU Time

- Single threaded code
- Inefficient algorithms
- Branch misprediction, cache misses
- Spin locks

High Wait Time

- Disk I/O
- Network calls
- Locks
- Synchronization

High Wait Time

- Disk I/O
- Network calls
- Locks
- Synchronization

2 Locks

Or why they're evil

[...] instead of picking up Djikstra's cute acronym we should have called the basic synchronization object "the bottleneck".

-- David Butenhof

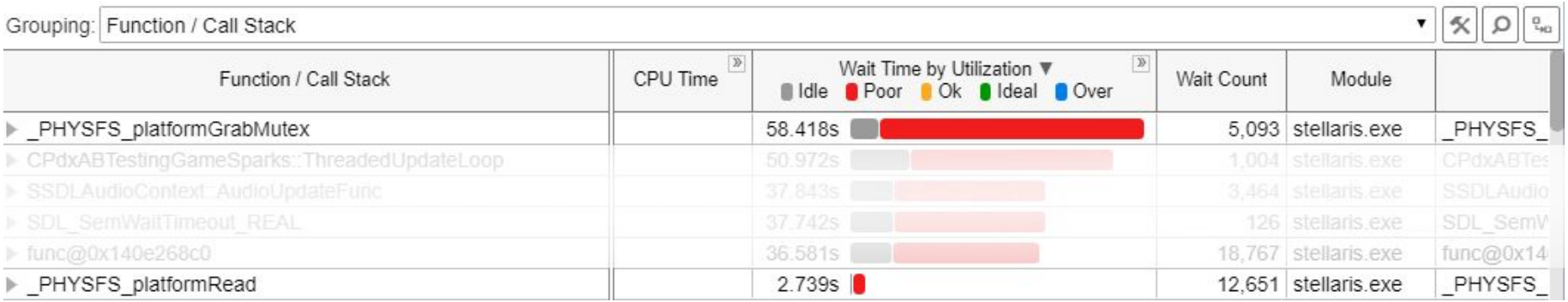
Bottlenecks are useful at times, sometimes indispensible (sic) -- but they're never GOOD. At best they're a necessary evil.

-- David Butenhof

Grouping: Function / Call Stack

Function / Call Stack	CPU Time	Wait Time by Utilization ▼ (Idle, Poor, Ok, Ideal, Over)	Wait Count	Module	
_PHYSFS_platformGrabMutex		58.418s	5,093	stellaris.exe	_PHYSFS_
CPdxABTestingGameSparks::ThreadedUpdateLoop		50.972s	1,004	stellaris.exe	CPdxABTes
SSDLAudioContext::AudioUpdateFunc		37.843s	3,464	stellaris.exe	SSDLAudio
SDL_SemWaitTimeout_REAL		37.742s	126	stellaris.exe	SDL_SemW
func@0x140e268c0		36.581s	18,767	stellaris.exe	func@0x14
_PHYSFS_platformRead		2.739s	12,651	stellaris.exe	_PHYSFS_

Stellaris 2.7 (Ref) Profile by Wait Time



Stellaris 2.7 (Ref) Profile by Wait Time

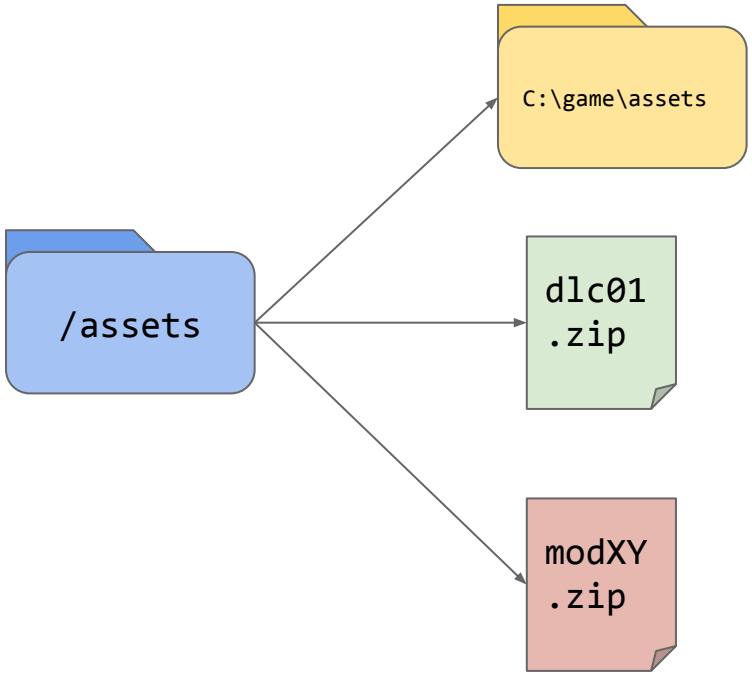
Wait Times

- FS access mutex costs 20 times the actual disk I/O time
- Would potentially be faster single threaded
- Why is it there in the first place?

What's a PhysFS?

- Open source multiplatform VFS library
- Mount folders, drives and archives
- Intended for use in video games
- Written in C 😢

What's a PhysFS?




```
PHYSFS_mount("/assets",
             "C:\game\assets", 1);
```

```
PHYSFS_mount("/assets",
     "C:\game\dlcs\dlc001.zip", 1);
```

```
PHYSFS_mount("/assets",
     "C:\user\mod\modXY.zip", 1);
```

PhysFS Threading Model

- Designed in the early 2000s
- Mostly concerned about thread safety
- One big mutex to protect all state
- Scales really badly if multiple threads do I/O

PhysFS State

- Mount points / library settings
- Open files list
- Per-thread last error code
- Per-archive state

Improving PhysFS Locking

- Split main mutex into several
- Remove the error code mutex entirely
- Introduce toggle to disable configuration mutex at user request

Improvement Results, Round 1

- Wait time on PhysFS mutex entirely disappears
- Wait time goes down, game starts a bit faster
- CPU load increases

Improvement Results, Round 1

- Wait time on PhysFS mutex entirely disappears
- Wait time goes down, game starts a bit faster
- CPU load increases


STOP
UN TRAIN PEUT
EN CACHER
UN AUTRE

A lock may hide another

Improvements, Round 2

- Our PhysFS wrapper had a spin lock on opens
- Was supposed to address a race condition on Linux
- Could simply be removed now

Side by Side Comparison

Thoughts about locks

- Locks may make code thread safe, but they also make it thread inefficient
- Keeping a computation lock free may require refactoring to use another approach
- Adding a lock might look fine in profiler because another bottleneck exists upstream

3

Threading Computations

Keeping those cores busy

Side by Side Comparison

Loading graphic assets

- Loading of both 2D and 3D assets was single threaded
- Huge speedup potential if we could spread it on all cores
- But DirectX9 would not play ball

Direct3D 9 does not default to thread safe. However, when you call **CreateDevice** or **CreateDeviceEx** to create a device, you can specify the D3DCREATE_MULTITHREADED flag to make the Direct3D 9 API thread safe. This causes significant synchronization overhead. Therefore, making the Direct3D 9 API thread safe is not recommended because performance can be degraded.

https://docs.microsoft.com/en-us/windows/win32/direct3d11/overviews-direct3d-11-render-multi-thread-differences

Consoles being ahead in tech? Madness!

Redesigning for multithread

- Switching to DX11 allows for multithreaded texture and model loaded
- Loading algorithms needed to be rewritten
- 2D and 3D asset loading needed a different approach

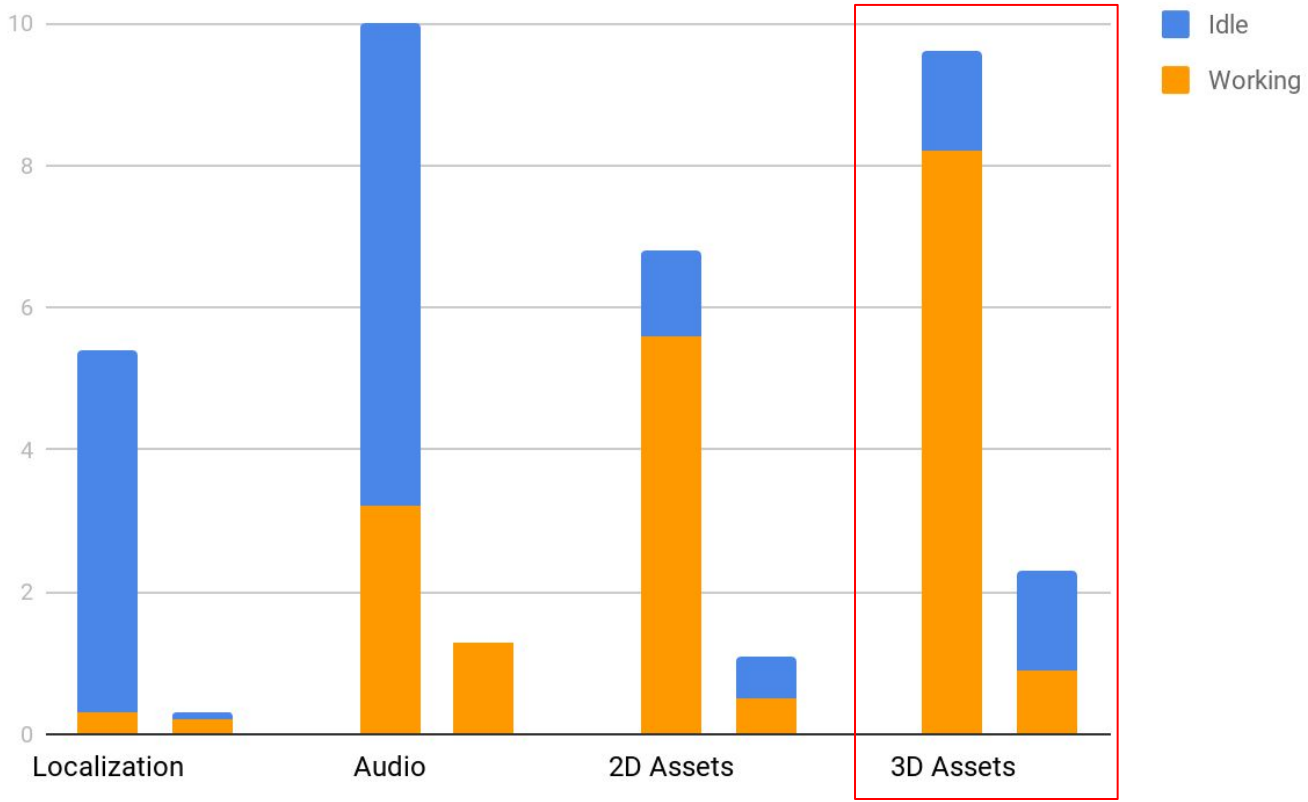
3D Assets Load

```
for ( const auto& Entry : _Pdx3DTypes )
{
   Entry._Value->InitForDevice( *this );
}
```

3D Assets Parallel Load

```
auto LoadFn = [&]( auto pType )
{
   pType->InitForDevice( *this );
};

PdxParallelFor( _Pdx3DTypes, LoadFn );
```



Side by Side Comparison

Easy parallel computing

- Moving from serial to parallel doesn't have to be hard
- Some loops already fulfills all the requirements to be replaced by a parallel_for

Easy parallel computing

- Iterations do not write to any shared state
- Order of iteration isn’t important
- No locks are being used

What if I need to update shared state?

Refactoring shared state

- Make copies
- Split problematic iteration in 2 loops
  - Parallel apply using a private working set
  - Serial loop to combine results
- Lock “smart”

2D Assets Load

```
for ( const auto& Entry : _SpriteTypes )
{
   Entry._Value->InitForDevice( *this );
}
```

2D Assets Parallel Load?

```
auto LoadFn = [&]( auto pSpriteType )
{
   pSpriteType->InitForDevice( *this );
};

PdxParallelFor( _SpriteTypes, LoadFn );
```

2D Assets Bottleneck

```
void CSpriteType::InitForDevice()
{
    _Texture = _Graphics.GetTextureHandler()
                        ->AddTexture( _TextureFile );
}
```

2D Assets Bottleneck

```
int CTextureHandler::AddTexture(
                        const string& Filename )
{
   scoped_lock Lock( _Mutex );
   int Idx = _Textures.Find( Filename );
   return Idx != -1 ? Idx :
      _Textures.Add( Load( Filename ) );
}
```

Removing Bottleneck

- Split sprite initialization in three phases
  - Each sprite declares which textures it needs
  - Load all requested textures
  - Bind loaded textures to sprites

- Reduce locking scope

Bottleneck mitigation

```
{
    scoped_lock Lock( _Mutex );
    int Idx = _Textures.Find( Filename );
    if ( Idx != -1 ) return Idx;
}
auto Texture = Load( Filename ) );
...
```

Bottleneck mitigation

```
...
{
   scoped_lock Lock( _Mutex );
   int Idx = _Textures.Find( Filename );
   return Idx != -1 ? Idx :
      _Textures.Add( move( Texture ) );
}
```

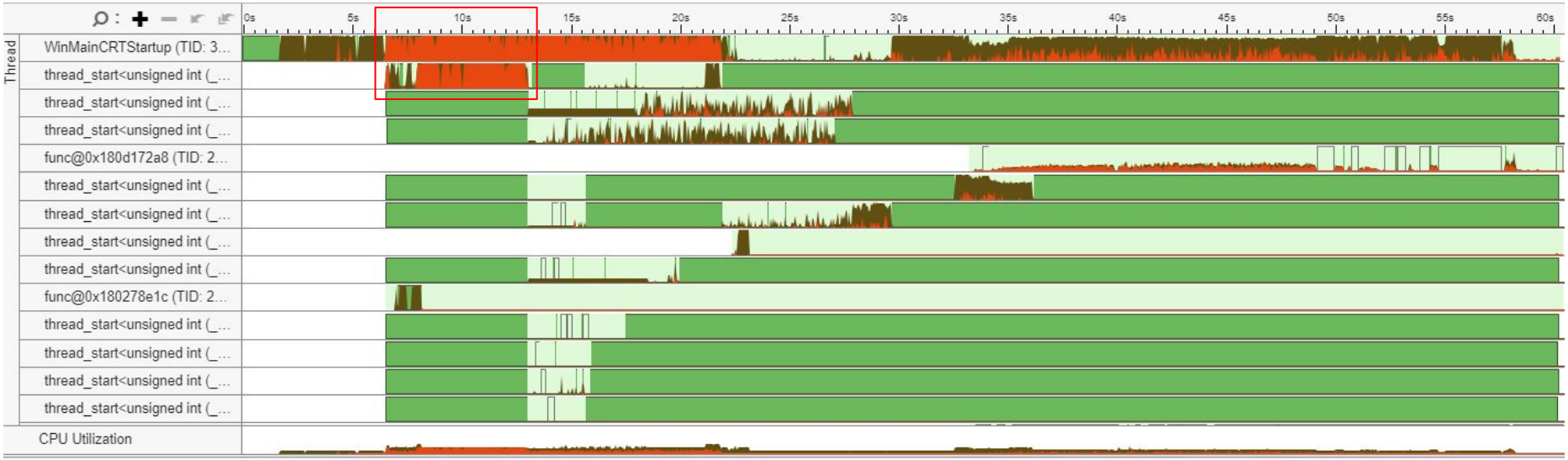
Side by Side Comparison

Improving 2D Assets Load

- Tactical changes to the locking scheme allowed for parallelization
- Even better gains could be achieved by refactoring to lock free
- But we already reduced from 6.8s to 1.1s

4 Going Asynchronous

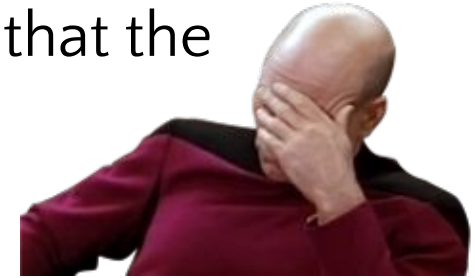
Making it a problem for the future



2.7 (Old) Startup CPU Usage

Surprise gains

- 5s were spent waiting on a network call
- Initially put inside a future…
- … until refactoring demonstrated that the results were never used



Working Smart

- Data has to be loaded, but does it have to loaded *now*?
- Our goal is to display the main menu as fast as possible
- We can continue loading in background

Side by Side Comparison

Loading audio

- Most of the audio load CPU time was spent reading music tracks from zips
- Can we optimize unzip() further?
- PhysFS still has a lock per archive, so multithreading might be tricky

Loading audio

- How many music tracks do we need to display loading screen and main menu?
- One. Stellaris main theme.
- Others will not be needed until the player starts a new game or loads a save

Loading audio

- Load only main theme immediately
- Start a background thread to load the rest
- Wait on it when we are about to drop in game
- Could potentially be applied to other assets

5 Wrapping up

Want to have more tooling yet?

In conclusion

- Locks solve thread safety at the cost of threading efficiency
- If your algorithm requires locking to parallelize, consider another approach
- Do not underestimate the potential gain of revisiting older code with threading in mind

Furthermore

Furthermore, I think your build should be destroyed

Thanks!

Any questions ?

You can reach me at

- mro@puchiko.net
- @MatRopert
- @mropert
- https://mropert.github.io

References

- *PhysFS performance, a story of threading and locking* - Mathieu Ropert, July 26th, 2020


Interesting Countries
12:00, 1 January, 1007
The End of Harmony
Change Scenario
Stalliongrad
Select Country
Equestria
Crystal Empire
Stalliongrad
Changeling Lands
Kingdom of O ...
Griffonian Empire
River Republic
Random Country
NEW IN 1.8
Information
Leader
Altidiya
Ideology
Communist
Government
Socialist Regime
Elections
May 1006
Ruling Party
SCP
STALLIONGRAD
Select Country
Brief History
The Soviet Republic of Stalliongrad, comprising of the Equestrian north-east provinces, had only recently split off from Equestria and the Crystal Empire in the hopes of forming a new society in which all ponies are equal. Unfortunately the Stalliongrad of today is far from the what had been envisioned in its founding. Rapid industrialisation has failed, and food has become scarce. With Equestrian sanctions, the threat of famine is very real. The ruling Communist Party, crippled by a leadership crisis, is incapable. Only after the following election, once the party has a clear leader, will Stalliongrad be able to face the oncoming storm.
Altidiya
Communist
Socialist Regime
May 1006
SCP
Custom Game Rules
Regular
Veteran
Elite
Historical AI Focuses