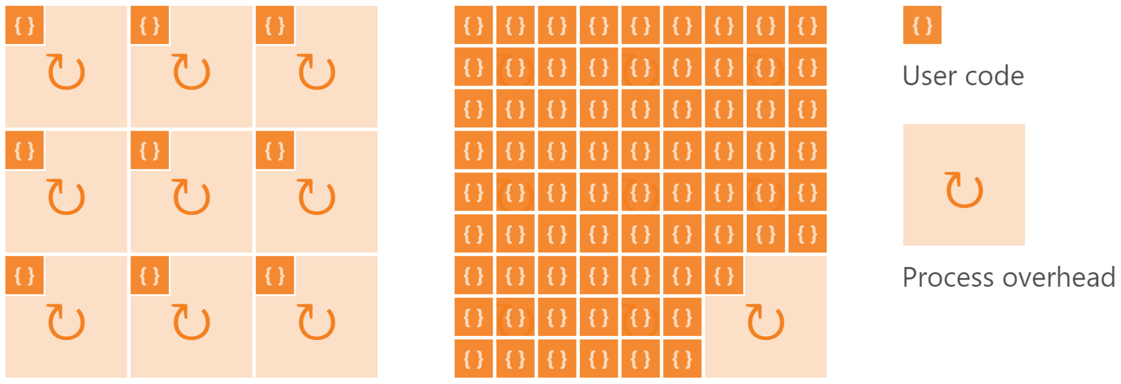
Linnea:

- Isolates = completely separate instances of the V8 engine
    - https://v8docs.nodesource.com/node-0.8/d5/dda/classv8_1_1_isolate.html
- V8 is the main container class
- V8 = "Google's open source javascript engine"
- Useful for running javascript simultaneously and isolated in multiple threads
    - ^ from isolated-vm readme
- isolated-vm helps with passing data between isolates easily → data must be "transferable"
    - Async and sync versions of functions
- https://developers.cloudflare.com/workers/learning/how-workers-works



Traditional architecture　　Workers V8 isolates

    -
    - Isolates are created within an existing environment, so you don't have to start up a whole VM/container situation -- this makes them faster

Claire (First run through):

https://v8docs.nodesource.com/node-0.8/d4/da0/v8_8h_source.html#l02777

- V8 isolates (look at def of these, different VMs or different processes/components within the V8?) have separate states, use different objects
- Isolates can be used by one thread at a time, and you can use them in parallel (different threads running at the same time are in different isolates)
- So they are more like protection domains??
- Locker/Unlocker API for synchronization
- Isolate class public methods are...
    - New: create new isolate
    - GetCurrent: returns current isolate
    - Enter: sets this isolate as the one the thread has entered and saves the isolate that the thread was previously in (if one), thread must hold a lock to enter

    - Exit: restoring the previously entered isolate in the current thread (thus exiting the current one)
    - Dispose: gets rid of isolate (somehow), no thread can be in it for it to be disposed
    - SetData: set some data associated with the isolate
    - GetData: get the data (if it was previously set)
- Scope is a class defined within the isolate class, Scope is isolate->enter and ~Scope is isolate->exit

Genny notes:

- run time: also known as execution environment, is part of a language implementation which executes code and is present at run-time
- Isolate allows you to create JS environments which are isolated from each other, useful for running untrusted code in a secure way or running JS simultaneously in multiple threads
- need to make the data transferable
- synchronous and asynchronous version of methods
    - asynchronous functions return a promise, some versions are “ignored” and don’t return
- asynchronous runs in order they were queued, FIFO
- Class: Isolate, each handle to an isolate is transferable and the isolate will remain valid as long as someone holds a handle to it or anything inside of it, garbage collector will clean up its memory
- V8 engine: Google’s open source Javascript engine → their compiling engine
- isolate is like a “sandbox” to run in (normally these are used for testing)
- memory is completed isolated
- this feels like an alternative to containers and virtual machines

Group meeting 4/2/21:

- Node.js is a backend javascript runtime environment
- Javascript is client side, Node.js is server side
- Javascript can run in any engine, Node.js must run in V8
- Server side javascript is the javascript code on the server that deals with client requests (from the user, for data and such...)
- V8 engine is a core component of the Node.js system
- Node is a runtime environment that “runs on” v8, and v8 implements Isolates
- V8 compiles and Node runs it?
    - https://www.geeksforgeeks.org/difference-between-node-js-and-javascript/#:~:text=Javascript%20can%20only%20be%20run,be%20run%20outside%20the%20browser.&text=Javascript%20can%20run%20in%20any,V8%20engine%20of%20google%20chrome.
- Isolated-vm runs V8 with separate instances of the environment in separate isolates
- Isolates start up way faster than processes on a container
- Cloudflare Isolate talk: https://www.infoq.com/presentations/cloudflare-v8/

- Cloudflare blog notes: https://blog.cloudflare.com/cloud-computing-without-containers/

Questions:

- Isolates share user level Java runtime libraries but are still isolated from each other... how?
- What does it mean for a function to belong to an isolate?
- Container vs isolate?
- How do we seamlessly switch between isolates?
- If you can give isolates handles to each other, how is that isolated…?

Group meeting 4/11/21:

Focus on…
- Security (Linnea)
- Communication (Claire)
- Performance (Genny)

Group meeting 4/12/21:

Why is there a separate isolate allocator?

Looking at isolate_allocator.cc to see if we can get greater insight into how the memory for isolates is allocated and isolated from one another.

make_unique(isolate_allocator) at the beginning of Isolate::New method, make _unique is a c++ method, returns a pointer, only one pointer can point to that sections of memory at a time, you can call other_pointer = std::move(pointer)

Isolate::New creates a unique pointer to an isolate allocator, and then sets an isolate pointer equal to the isolate allocator pointer cast as an isolate pointer.

isolate->root()? Determines what the isolate can access?

IsolateAllocator() creates new and ~IsolateAllocator() is a deconstructor/deletes it

Isolate_root = heap reservation address + k isolate bias page size
Bounded page allocator, allocated pages within the pre-reserved region of virtual space

A reservation is passed into CommitPagesForIsolate(), we get the reservation from InitReservation()

InitReservation() uses GetRandomMapAddr() method, basically finds memory for an isolate to be allocated
kIsolateRootBiasPageSize()???

There is a heap inside the isolate, Heap heap_, and a read_only_heap_,

Linnea more notes 4/13
Summarize the project, what it is, what its goals are, and why it exists.
First of all it's helpful to explain what Node and V8 are. V8 is Google's Javascript "engine," according to its own definition. An engine in this context is a system that runs other code. As a Javascript engine, V8 takes in JS code and compiles and executes it. This involves handling the order of execution, managing memory, and providing the data types and objects necessary. (https://hackernoon.com/javascript-v8-engine-explained-3f940148d4ef). The execution order is controlled by the call stack and memory is allocated on the heap, which is managed by a garbage collector. Importantly, it provides the Javascript flow of a single threaded application, though this doesn't mean the C++ implementation of V8 is single threaded.

Next, Node is self-defined as "a Javascript runtime which runs on the V8 engine." The runtime is the environment your code runs on while code is running. Javascript is a language designed to be run on the browser -- on the client-side. Node.js allows you to write Javascript code and run it on the server side instead. Node also provides some tools and libraries to make this easier. (can't cite these but https://www.quora.com/What-does-it-mean-to-say-that-Node-js-is-a-runtime-environment-for-executing-JavaScript-Is-it-a-programming-language-per-se-Is-it-a-framework-or-a-library and https://stackoverflow.com/questions/3900549/what-is-runtime). Essentially, to run server-side JS code, you'd first need to run Node, which uses V8 to execute your code.

The most common users are web servers. Millions of sites want to run their server-side JS code to service user requests, and these need to also have low latency. In the advent of serverless computing, there has been a need to service many tenants on the same hardware. Since these tenants don't trust each other, they need to be isolated. This led to innovations like Virtual Machines and containers. However, running an entire VM, which involves a virtual kernel and userspace specific to each tenant can be slow and consume a lot of space. While start up times for VMs have gotten faster, it's still somewhat slow.

This is where isolates come in...

What is the target domain of the system? Where is it valuable, and where is it not a good fit? These are all implemented in an engineering domain, thus are the product of trade-offs. No system solves all problems (despite the claims of marketing material).

Node isolates target the domain of running server-side Javascript code, which is generally web servers. Companies that provide serverless infrastructure would find it valuable, because they need to service many independent tenants with as little latency as possible. Because Isolates don't need to run in their own process, the start-up time is faster even than just creating a new process, which is essential for tenants that don't see as much traffic and would have cold starts with another system.

- The target domain is for serverless infrastructure
- It's valuable because it allows developers to write JS code that they want to run, rather than a whole VM, and they can also have isolation
- It's also valuable for efficiency. Because isolates are within a single process, you don't have the "cold start" of starting the new process to service a request; you already have an existing process and it takes much less time to create an Isolate from that
- It wouldn't be a good fit if you were going to run anything other than JS code for a website to run on a web server, because you don't have isolation outside of the Isolate inside of the process. If you needed isolation of all the libraries or within the memory of the entire system or for doing system calls, you'd probably prefer a VM.

Linnea looking for tests 4/14
https://github.com/v8/v8/blob/dc712da548c7fb433caed56af9a021d964952728/test/cctest/test-managed.cc

This is just creating isolates and making sure they get disposed and that kind of thing

Claire 4/15-15 on isolates sharing memory, communicating, interacting with one another

Each isolate has its own unique chunk of memory that only it has access to, but, multiple isolates can have access to the same context… Therefore, this is how isolates can share javascript libraries and execute library functions. It seems like this would also make the system faster because instead of compiling javascript libraries within each individual javascript runtime environment (v8 engine) within each isolate, isolates can share a single library that only has to be compiled once. This does not jeopardize the security of the isolate in the sense of memory isolation because each isolate can have its own pointer to the context (think of an acyclic directed graph) but no isolate has access to another isolate's memory space. You can think of these shared contexts in a sense as shared memory between the isolates.

Where is the return value stored in this instance?

Linnea 4/15 and 4/16
SetThreadLocals seems to set global data that's "local" to that current thread. So in Isolate::Enter when we call `Isolate::SetIsolateThreadLocals(Isolate ptr, PerIsolateThreadData ptr)` that calls `Thread::SetThreadLocal` for both variables, which just takes in the key-value pair. This function is implemented differently depending on the

platform, but if we look at the POSIX implementation, we see that it gets the “pthreadkey” associated with the key and calls the pthread_setspecific function with the key and value.

- https://github.com/v8/v8/blob/dc712da548c7fb433caed56af9a021d964952728/src/base/platform/platform-posix.cc#L992 → SetThreadLocal
- https://github.com/v8/v8/blob/master/src/execution/isolate.cc#L3164 → SetIsolateThreadLocals
- https://github.com/v8/v8/blob/master/src/execution/isolate.cc#L3797 → Where it’s called in Isolate::Enter with the isolate (this) and the PerIsolateThreadData as arguments

Group meeting 4/16

`Isolate::InitializeOncePerProcess` This function creates the thread local isolate key and isolate data key

- https://github.com/v8/v8/blob/dc712da548c7fb433caed56af9a021d964952728/src/execution/isolate.cc#L467
- This function has to be called for each process

https://github.com/v8/v8/blob/dc712da548c7fb433caed56af9a021d964952728/tools/debug_helper/debug-helper.h#L161

This references the isolate.heap_ having some kind of list structure and front pointer

heap::SetUpSpaces:
https://github.com/v8/v8/blob/dc712da548c7fb433caed56af9a021d964952728/src/heap/heap.cc#L5124 setting up sections within the heap? Does something with the space_ variable that gets initialized to nullptrs in heap::SetUp (https://github.com/v8/v8/blob/dc712da548c7fb433caed56af9a021d964952728/src/heap/heap.cc#L5039)
These are called in Isolate::Init (https://github.com/v8/v8/blob/master/src/execution/isolate.cc#L3574)

Readonlyheap:
https://github.com/v8/v8/blob/dc712da548c7fb433caed56af9a021d964952728/src/heap/read-only-heap.cc#L63

- If statement for if it’s shared or not
- Mutex on it
- Taking a lock on the shared ptr artifacts variable
    - If artifacts null, create them (as ReadOnlySharedArtifacts)
- Deserialize into isolate maps it in?

Read only space and read only heap?

In ReadOnlyHeap::Setup:

- We want to look at this method to see what is in an Isolates read only heap (such as maybe pointers to libraries, what objects?, pointers to the rest of their memory space?)

- If the read only heap is shared, then how do they keep their memory isolated?
- If pointer compression is enabled, then each isolate has its own read only heap, but if pointer compression is not enabled, then each isolate shares one read only heap
- If the space is shared than we take the mutex and lock, and then if the artifacts aren't set up yet then we do that. If the artifacts already exist, then we set up the read only heap of the Isolate using these artifacts
- There is a call to InitializeIsolateRoots:
    - Memcpy's the read_only_roots associated with the current SoleReadOnlyHeap type into the isolates **root_table**
    -

- If read only **spac**e is not shared, it creates a new read_only_heap, passing in a new read only space, which is created by passing in the isolate->heap()
- Look at new read space method?
- https://github.com/v8/v8/blob/dc712da548c7fb433caed56af9a021d964952728/src/heap/read-only-heap.cc#L63

Linnea more notes 4/16
- Roots? https://github.com/v8/v8/blob/dc712da548c7fb433caed56af9a021d964952728/src/roots/roots.h

Looking for how memory allocation works
- Memory allocator class: https://github.com/v8/v8/blob/dc712da548c7fb433caed56af9a021d964952728/src/heap/memory-allocator.h#L58
- Comments say: "A space acquires chunks of memory from the OS. the memory allocator allocates and deallocates pages for the paged heap spaces and large pages for large object spaces"
- Inner Unmapper class
- Allocation mode is pooled or regular
- Allocate page, allocate large page, allocate ReadOnlyPage functions
- Has an associated isolate_ var
- Allocate page template function: https://github.com/v8/v8/blob/dc712da548c7fb433caed56af9a021d964952728/src/heap/memory-allocator.cc#L632

Linnea more notes 4/18
- What is the "reference monitor"
    - Memory allocator? Spaces?
- https://github.com/v8/v8/blob/dc712da548c7fb433caed56af9a021d964952728/src/heap/memory-allocator.cc#L480
- AllocateChunk has a BaseSpace pointer *owner

- baseSpace allocate takes a mutex and somehow actually adds a page
- https://github.com/v8/v8/blob/dc712da548c7fb433caed56af9a021d964952728/src/heap/cppgc/heap-space.cc#L23
- MemoryChunkInitialize takes in the isolate.heap_ (https://github.com/v8/v8/blob/dc712da548c7fb433caed56af9a021d964952728/src/heap/memory-allocator.cc#L490) and the newly allocated memory
    - Here's the implementation: https://github.com/v8/v8/blob/dc712da548c7fb433caed56af9a021d964952728/src/heap/memory-chunk.cc#L106
    - Doesn't make any sense to me :(
- Allocate functions "reserve but don't commit"?

Linnea more notes 4/19
- Heap structure: new space, old space, code space: https://youtu.be/3bVcTFOKRyo

Claire and Linnea 4/20
- An Isolate is an instance of V8
- A context is the execution environment for running code
- So you have to have a context within the Isolate in order to actually run any code

Genny notes 4/20:
- what is the kisolateindex within PropertyCallbackInfo
- promise defined on line 4820 in v8.h
- what is class V8_EXPORT
- compiled_wasm module for compiled web assembly operations → compiler architecture notes, etc.
- in most of the v8.h defined modules, isolates are passed in
- resource_contraints specify the amount of resources runtime JS can use
    - initial_heap_size_in_bytes
    - maximum_heap_size_in_bytes
- each isolate is its on v8 runtime and gets max constraints
- promise_hook with different types called
- microtasks to hold what the isolates need to do
    - they are of type Local<Function>
- A JIT code event is issued each time code is added, moved or removed.
- isolate defined on 8217
    - scope → Stack-allocated class which sets the isolate for all operations executed within a local scope
        - constructor function: isolate->Enter();
        - constructor function: isolate->Exit();
    - Set the callback to invoke to check if wasm code generation should be allowed.
- v8 locker and unlocker to ensure that thread constraints are not violated line 10662 in v8.h
- https://v8.dev/blog/10-years

- Actually, we can run things in parallel, but we don't create threads, and we don't sync them. The virtual machine and the operating system run the I/O in parallel for us, and when it's time to send data back to our JavaScript code, the JavaScript part is the one that runs in a single thread.
- In other words, everything runs in parallel except for our JavaScript code. Synchronous blocks of JavaScript code are always run one at a time:
- Screeps → https://blog.screeps.com/2018/03/changelog-2018-03-05/

Claire 4/18/21

isolate/scheduler.cc in isolated-vm
https://github.com/laverdet/isolated-vm/blob/27cb89d3c3c4ef433af22ac8efa38a26f1caabda/src/isolate/scheduler.cc

Scheduler::InterputIsolate/InterruptSyncIsolate
- Change status of scheduler to running
- Request an interrupt (sync or aysnc)
    - https://github.com/laverdet/isolated-vm/blob/19b3624b962c13526f78949fd8a6391895a9463d/src/isolate/environment.cc
    - This executes IsolateEnvironment::InterruptEntryImplementation, which executes the top interrupt and pops it off the list, and continues this until the array of interrupts is empty

An Isolate Environment in isolated-vm…

IsolateEnvironment::AsyncEntry():
- Takes executor lock
- Has an infinite loop that has 3 established queues (tasks, handle_tasks, interrupts), goes through and takes a lock to get the current state of each queue, then loops through each and runs each task until they are empty

Scheduler::WakeIsolate:
- Changes status (of scheduler?) to running
- Move isolate pointer into env_ref attribute of this scheduler
- SendWake()

https://github.com/laverdet/isolated-vm/blob/19b3624b962c13526f78949fd8a6391895a9463d/src/isolate/holder.cc
Holder.cc

IsolateTaksRunner::PostTask/PostDelayedTask:
- If you successfully take the weak_env lock, then you get the scheduler of this environment and take the lock and then push/move the task passed into the method onto the tasks queue

IsolateHolder::ScheduleTask

- If the task can be run inline, then do so
- If not, then lock the ref->scheduler lock, then move the task either onto the handle_tasks or tasks queue associated with the lock (of the scheduler), and then if wake_isolate boolean is true, call lock->WakeIsolate

V8 TaskRunner
https://github.com/v8/v8/blob/dc712da548c7fb433caed56af9a021d964952728/test/inspector/task-runner.cc

Callbacks and Promises

Event triggers another function (pass function into callback)

Linnea's video:

- Single instance of v8 is an isolate
- Have separate garbage collected heap, not possible to pass an object from one to another
- Each isolate has multiple contexts, one for each <iframe>, and every extension gets its own context per <iframe>

https://www.npmjs.com/package/isolated-vm#class-context-transferable

- Transferable values may pass between isolates
- Isolated vm we can create an isolate, create a context, compile a script within that context, and then run the script…
- Maybe when talking about efficiency reference the example at the end of this document
- 1 script per context
- Can be many contexts per isolate

Frame attributes (in the struct) are in isolate.cc line 855

Isolate::MayAccess method passes in a handle to the context you want to access and the handle to the javascript object you are passing?

Isolate::RequestInterrupt line 1476, takes in a callback and void* data, pushes an interrupt entry onto the api_interrputs_queue_ which is unique to this isolate?, and then calls requestApiInterrupt();

RequestApiInterrupt:

Line 2606 GetIncumbentContext

Each isolate has an unorderd map of contexts, maps a Persisent context to a copyable persistent trait context, and isolate also keeps track of the last_recorder_context, and the top entry of the v8 context backup incumbent stack

Isolate struct from isolate.h:
Line 1964 - thread data table and lock for it

Contexts struct from context.h:

- Contexts are FixedArray-like objects having a fixed header with a set of common fields
- Contexts must always be allocated by Heap::AllocateContext() or Factory::NewContext