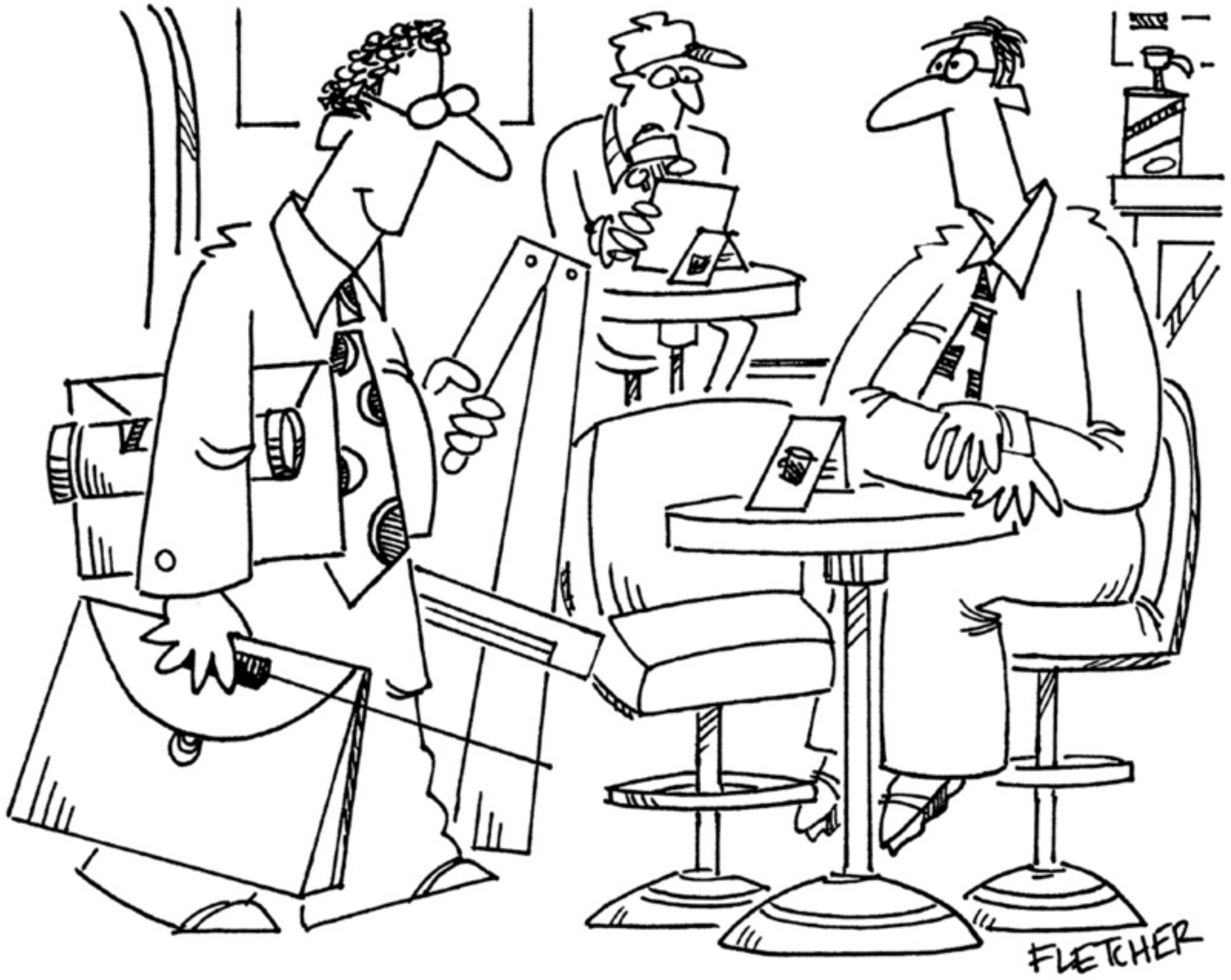
Using D for Development of Large Scale Primary Storage

Liran Zvibel
Weka.IO, CTO
liran@weka.io
@liranzvibel

Agenda

- What are we doing
- Our Infrastructure in D
- Challenges
- Working Together
- Q&A



Even before they ordered their lattes, Larry sensed that Pastor Jim might have an agenda.

About Weka.IO

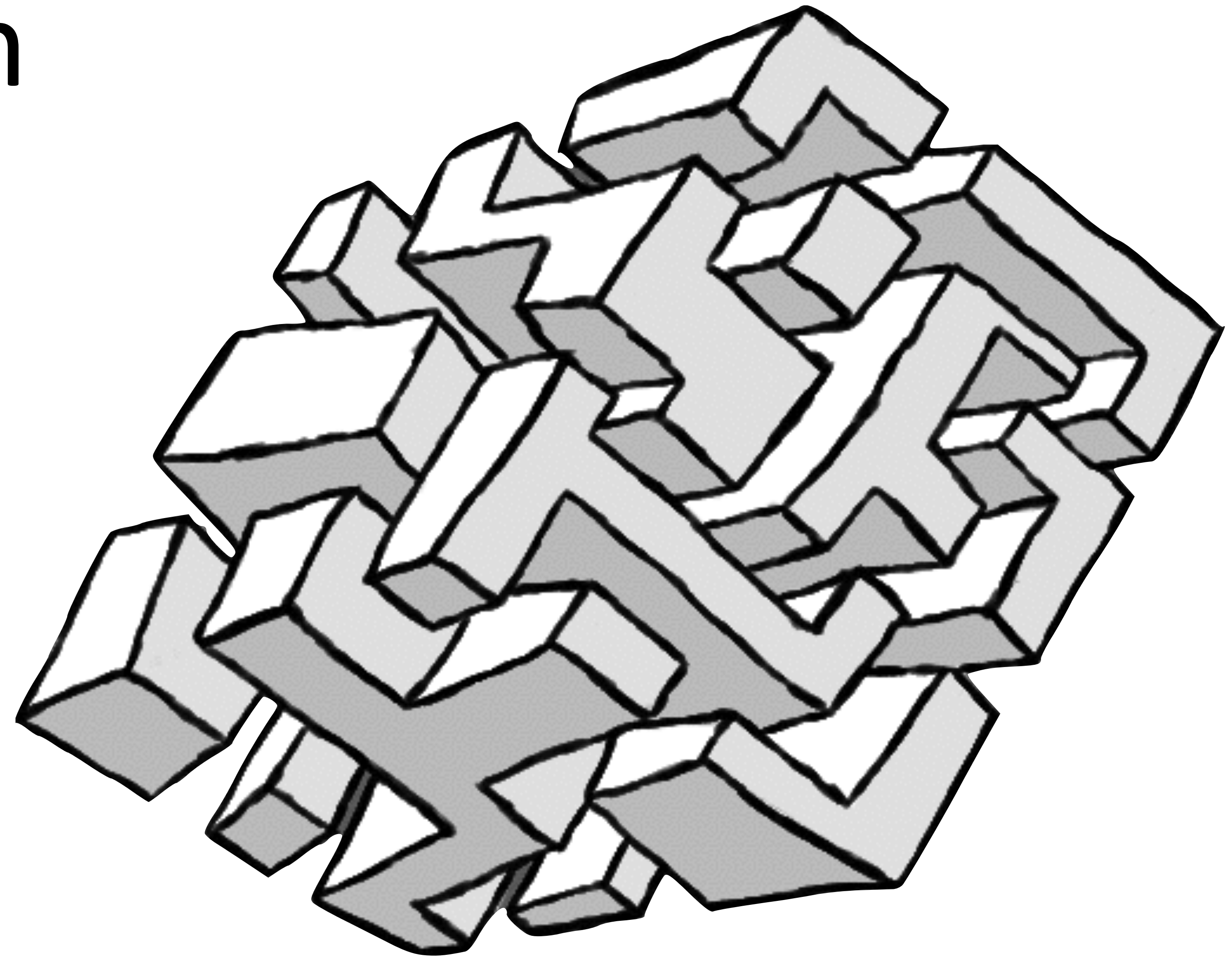
- Israeli based
- Defining the future of software defined, scale out storage for the cloud-based datacenter
- VC backed company (NVP, Gemini), largest round-A of 2014
- Currently 20 engineers, many XIV veterans.
- Started developing in D early 2014
- D project size: 120k loc, internal code: 113k loc [400/380 modules], 13 packages with package.d files.

Storage system requirements

- Extremely reliable, “always on”.
- High performance data path, measured in μsecs
- Complicated “control path”/“management code”
- Distributed nature due to HA requirements
- Low level interaction with HW devices
- Some kernel-level code, some assembly
- Language has to be efficient to program, and fit for large projects

What did we do previously?

- C codebase
- A lot of auto-generated code from XML for RPC, clustering code and external APIs (CLI, GUI)
  - Requires a complicated build process
  - Difficult to understand where “magic” code comes from
- Our own implementation of Classes/polymorphism and templates mainly for containers
- Python based CLI and administration



The Weka.IO framework

- Userspace processes
- 100% CPU, polling based on networking and storage
- Asynchronous programming model, using Fibers and a Reactor
- Memory efficient, zero-copy everything, very low latency
- GC free, lock-free efficient data structures
- RPC framework (with no IDL)

Infrastructure

Traces

- Problem:
  - Resiliency is very high, reproducing errors is too expensive, all bugs must be fixed!
  - you cannot ‘gdb’ a single fiber in a distributed system, as you’re going to change the interactions (stop other fibers, change timings)
  - You cannot print text (formatted or not)
    - Too slow
    - Output will fill local drives very quickly
    - Very inefficient to filter/search

Traces: Requirements

- Seamless logging of function entry and exit, incl. arguments, out params
  - `@notrace` a function if you DON'T want it to be traced
- Efficient logging
  - `INFO!"autorecovery is %s"(localState.autoRecovery);`
- Synchronizes several threads to single log
- Very efficient binary representation
- Very efficient runtime "blitting" of data
- Very efficient filtering/searching based on data, text is only generated screenful at a time

```
61 00023         --> weka.bucket.service.takeOver(bucketId = BucketId<17>, failedNodeId = NodeId<65535>)
61 00023            LOG INFO Attempting taking over BucketId<17>
61 00023            --> weka.journal.journal.Journal.takeOver(this = 0x00007FDEE69EF800, startSynchronization = false)
61 00023               --> weka.journal.journal.Journal.isLeaseValid(this = 0x00007FDEE69EF800)
61 00023               <-- weka.journal.journal.Journal.isLeaseValid(return = false)
61 00023               LOG INFO JOURNAL: NodeId<61> taking over BucketId<17> from NodeId<65535>
61 00023            <-- weka.journal.journal.Journal.takeOver(return = true, info = TakeOverInfo(firstOpId = BEOpId<18446744073709551615>, lastOpId = BEOpId<18446744073709551615>,
61 00023            LOG DEBUG local take over info BucketId<17>: TakeOverInfo(firstOpId = BEOpId<18446744073709551615>, lastOpId = BEOpId<18446744073709551615>, previousJournalOwne
61 00023            LOG DEBUG #RPC client invokes weka.journal.service_interface.IJournalService.takeOver, cookie=RPCCookie<144182258285150629> dest=NodeId<81>
61 00023            LOG DEBUG #RPC client invokes weka.journal.service_interface.IJournalService.takeOver, cookie=RPCCookie<144182258285150630> dest=NodeId<21>
61 01485      LOG DEBUG #RPC server invokes isReady cookie=RPCCookie<216261842555634055> from NodeId<81>#6679
61 01485      --> weka.bucket.service.BucketService.isReady(this = 0x000000003B29640)
61 01485      <-- weka.bucket.service.BucketService.isReady(return = false)
61 00023            LOG DEBUG took over journal of BucketId<17> on NodeId<81>. info: TakeOverInfo(firstOpId = BEOpId<18446744073709551615>, lastOpId = BEOpId<18446744073709551615>,
61 00029            LOG DEBUG Node NodeId<41> isn't ready yet
61 01486      LOG DEBUG #RPC server invokes isReady cookie=RPCCookie<216195871857967536> from NodeId<21>#48032
61 01486      --> weka.bucket.service.BucketService.isReady(this = 0x000000003B29640)
61 01486      <-- weka.bucket.service.BucketService.isReady(return = false)
61 00029            LOG DEBUG Node NodeId<21> isn't ready yet
61 00029            LOG DEBUG Node NodeId<1> isn't ready yet
61 00029            LOG DEBUG Nodes not alive yet. waited for 5 nodes
61 00029            LOG DEBUG numAlive=0, status=NON_AVAILABLE
61 00029            LOG DEBUG #RPC client invokes weka.bucket.service_interface.IBucketService.isReady, cookie=RPCCookie<216239852323078567> dest=NodeId<81>
61 00029            LOG DEBUG #RPC client invokes weka.bucket.service_interface.IBucketService.isReady, cookie=RPCCookie<216239852323078568> dest=NodeId<61>
61 00029            LOG DEBUG #RPC client invokes weka.bucket.service_interface.IBucketService.isReady, cookie=RPCCookie<216239852323078569> dest=NodeId<41>
61 00029            LOG DEBUG #RPC client invokes weka.bucket.service_interface.IBucketService.isReady, cookie=RPCCookie<216239852323078570> dest=NodeId<1>
61 00029            LOG DEBUG #RPC client invokes weka.bucket.service_interface.IBucketService.isReady, cookie=RPCCookie<216239852323078571> dest=NodeId<21>
61 01488      LOG DEBUG #RPC server invokes isReady cookie=RPCCookie<216239852323078568> from NodeId<61>#6734
61 01488      --> weka.bucket.service.BucketService.isReady(this = 0x000000003B29640)
61 01488      <-- weka.bucket.service.BucketService.isReady(return = false)
61 00029            LOG DEBUG Node NodeId<61> isn't ready yet
61 00029            LOG DEBUG Node NodeId<81> isn't ready yet
61 01489      LOG DEBUG #RPC server invokes isReady cookie=RPCCookie<216217862090523689> from NodeId<41>#43658
61 01489      --> weka.bucket.service.BucketService.isReady(this = 0x000000003B29640)
61 01489      <-- weka.bucket.service.BucketService.isReady(return = false)
61 00023            LOG DEBUG took over journal of BucketId<17> on NodeId<21>. info: TakeOverInfo(firstOpId = BEOpId<18446744073709551615>, lastOpId = BEOpId<18446744073709551615>,
61 00023            LOG INFO Take over of BucketId<17> completed successfully - TakeOverInfo(firstOpId = BEOpId<18446744073709551615>, lastOpId = BEOpId<18446744073709551615>, prev
61 00023            --> weka.events.impl.logEvent!(BucketTakeOver).logEvent(event = BucketTakeOver(bucketId = BucketId<17>, newNodeId = NodeId<61>, prevNodeId = NodeId<65535>))
61 00023               --> weka.events.shm.EventsShmStruct.writeEvent!(BucketTakeOver).writeEvent(this = 0x00007FDEE81F8000, event = BucketTakeOver(bucketId = BucketId<17>, newNode
61 00023                  --> weka.events.shm.EventsShmStruct.modIndex(this = 0x00007FDEE81F8000, idx = 10)
```

Steps in getting it to work

- Instrumenting the code to make sure we can tweak functions and classes/structs/enums/etc…
- CTFE/static code generates “blitting” code
- An updater process gives each function/log unique id
- Lockless runtime code efficiently dumps data to shared memory
- Runtime daemon dumps that memory to files
- Interactive reader lets engineers navigate runtime history (or present)

RPC — No IDL

- No IDL :)
- Only define `interface` for that RPC domain
- Then implement server in a `struct`, and get automatically generated sync/async callers
- Allows changing signature semantics (`out -> ref`, etc)
- Very easy to use
- Can asynchronously RPC many remote nodes “MultiCall”

```
    switch (funcId) {
        foreach(i, name; METHODS) {
            enum FUNCID = FIRST_USER_RPC_FUNCID + METHOD_IDS[i];
            static assert(__traits(getOverloads, INTERFACE, name).length == 1,
                    "Overloads not supported in RPC interfaces: " ~ name);
        alias Decl = FunctionTypeOf!(__traits(getMember, INTERFACE, name));
        static assert (__traits(hasMember, T, name),
            T.stringof ~ " is missing " ~ name ~ " of type " ~ Decl.stringof);
        static assert(__traits(getOverloads, T, name).length == 1,
                    "Overloads not supported in RPC implementations: " ~ name);
        alias Impl = FunctionTypeOf!(__traits(getMember, T, name));
        static assert (is(ReturnType!Impl == ReturnType!Decl) &&
                is(ParameterTypeTuple!Impl == ParameterTypeTuple!Decl),
            T.stringof ~ "." ~ name ~ " does not implement " ~
            INTERFACE.stringof ~ "." ~ name ~ ". Expected `" ~
            Decl.stringof ~ "`, found `" ~ Impl.stringof ~ "`");

        enum Storages1 = ParameterStorageClassTuple!Impl;
        enum Storages2 = ParameterStorageClassTuple!Decl;

        foreach(j, s; Storages1) {
            static assert (s == Storages2[j], T.stringof ~ "." ~ name ~ " parameter " ~ text(j) ~
                " is " ~ (cast(ParameterStorageClass)s).stringof ~ ", expected " ~ (cast(ParameterStorageClass)Storages2[j]).to!string);
        }
            case FUNCID:
                return invokeServerFunc!(FUNCID, name)(impl, preamble, request, reader, response, replay);
        }
        default:
            ERROR!("#RPC server " ~ T.stringof ~ " got invalid function id: %d")(funcId);
            dumpError(preamble, response, RPCFuncRet.PROTOCOL_ERROR, "Invalid function " ~ text(funcId));
            return true;
    }
}
```

Fiber related

- Fiber local storage defined anywhere in the code:
  - `alias currentEosId = FiberLocal!(EosId, “currentEosId")`
- Throw in fiber
- Extract backtrace from fibers

No-GC efficient data structures

- One-to-many lockless queue
- Lists, linked lists, queues
- Static bit arrays
- Cyclic buffer, cyclic queue
- Set, different Hash (dict) implementations
- Fixed arrays
- Resource pools
- TypedIdentifier (Can be moved to Typedef)
- format — NoGC formatting, compile time parsing of fmt str

Other goodies

- JSONRPC
- Http Server + Client without curl (also without SSL)
- readline implementation for out cli
- assers - assertEq, assertOp
- gc_hacks - accessing GC data (Why isn’t exported?!?)
- TimePoint, TimeOut could extend std.datetime
- reflection — overcome private/public restriction for generic reflection in standard code
- accessors that automatically and transitively wrap members and notify of changes

Challenges

Garbage Collection

- Always running, low latency applications cannot rely on GC
  - If you cannot stop for more than 1msec, the amount of memory you can scan is limited
- The standard library assumes GC is used, so it cannot be used
- Associative arrays, dynamic arrays, `map` and `filter` cannot be used since delegates forces GC
- The runtime state of the GC is private, makes it very difficult to debug and optimize

Compilation issues

- Compiling the project “at once” does not scale:
  - Takes a long time
  - Takes a lot of memory (DMD almost 30GB, GDC even more)
  - Does not leverage modern multi-core CPUs
- This is still a smallish project. What happens in few years when we have a large team?
- This way we cannot leverage (cache) previous compilations to make sure new compilations are quicker

Compile by object issues

- Expected signature differs from generated

@property @nogc @trusted weka.reactor.reactor.TimedCallback* weka.lib.pools.newpool.Pool!(weka.reactor.reactor.TimedCallback, uint, 1u, false).Pool.Ptr.value()

```
@property T* value() @trusted {
            if (_index == INVALID) {
                return null;
            }
            if (_index == uint.max -3) {
                // This is just to prove that this function cannot be nothrow and also @nogc
                throw new Exception(format("This is impossible %s", "bla"));
            }
          version(poolGuards) {
       assert(_elements[_index].magic1 == MAGIC1, format("%s: Magic1 is corrupt", &_elements[_index]));
       assert(_elements[_index].magic2 == MAGIC2, format("%s: Magic2 is corrupt", &_elements[_index]));
    }
    version(generationTracker) assert (_generation == _elements[_index].generation,
       format("%s: stale generation (%s), should be %s", &_elements[_index],
          _generation, _elements[_index].generation));
    return &_elements[_index].value;
        }
```

Compile by object — cont

- Current import system is not compatible
  - Transitive closure of all imports is usually a very large group
  - Means that almost any change forces way too many compilations
  - Compilation process is way too long since too much is compiled
- Possible solution:
  - identify imports that are relevant for the public part of that module.
    - `<external>?? import some_module;`
  - When compiling a single object, treat imported modules as they were only the header with the external imported modules used only

Compile by object — cont

- Executable size when monolithic compiled: 124MB
- Executable size when compiled by object: 1.4GB
- Some modules end up taking 10s of MBs, summed to over 6GB
- Even with `-allinst` some templates are not generated automatically

Optimizing compilers

- GDC still has issues running our code (fibers related and other stuff)
- Could not get LDC to compile our code (keeps segfaulting in the compilation process)

- Inlining C library functions does not work

- We ended up with no optimizing compiler

Ordering issues

- `static this` won't load the process because of cycles, many times are not relevant
  - When the project grows large enough it's difficult to make sure there are no import loops
  - Basically renders the feature not usable

Primitive integer types clunkiness

- `cast(ushort)(80 + someUShort)`
- `cast(ushort)(someUShort % (2^^ 16 -1))`
- `cast(ushort)(someUShort / 10)`
- `foreach(x; 0 .. 10000) cast(ushort)x;`
- `someUShort << 3; 10k -> 80k, also >>3`
- `cast(ushort)(someUShort | someUShort)`
- `cast(ushort)(someUShort & someUShort)`
- `~someUShort – what is the type?`

Recoup

- Developing in D for over a year — single language for control and data path!
- Getting a huge productivity boost
- Extensive usage of generic programming, CTFE and other features
- Invested a lot in infrastructure, starting to reap the fruits

- Large scale real time projects could be better supported

Helping each other

- We have a lot of library/utility functions we can donate
- We have a lot of code “hacks” we do to get things to work

- Looking for strong D contractors to be a bridge to the D community

liran@weka.io

Peta

Exa

Zetta

Yotta

Xenna

Weka (10^{30})

Questions?

WEKA.IO