Less Magic
“The most important principle in designing a programming language is defining a small kernel that all other constructs use.”
—Simon Peyton-Jones
Magic Hurts

- Increases the surface of the language
- Must implement, document, explain, maintain
- “Quod licet Iovi, non licet bovi”
- User-available facilities awkward, different semantics
- Difficulties for tooling
Examples

• Built-in hashtables
Examples

- Built-in hashtables
  - Iteration is special
Examples

- Built-in hash tables
  - Iteration is special
- Built-in slices
Examples

- Built-in hashtables
  - Iteration is special
- Built-in slices
  - Iteration is special
  - Handling of qualifiers is special
Examples

- Built-in hashtables
  - Iteration is special
- Built-in slices
  - Iteration is special
  - Handling of qualifiers is special
- Even Object
  - ?
Can the compiler optimize this?

- Hat tip to Johan Engelen

```c++
ubyte foo(immutable ubyte[] arr) {
    auto temp = arr[2];
    fun();
    return temp + arr[2];
}
```
Nope

- OK to cast immutable data to immutable ubytes representing it
- Reasonable to assume immutable data won’t change
- However:
  - Object has magic: the monitor field
  - Class objects cannot be seen as immutable bytes!
  - All other data types can!
“Here you strike and there it cracks.”
—Romanian proverb
We need reference counting

- Must work with `@safe` code
  - Memory deallocation woes
- Must work with `pure` code
  - Memory (de)allocation woes
  - Need to improve the spec
- Must work with `@nogc` code
  - Memory (de)allocation woes
- Must work with `immutable` data
  - The reference counter ruins the day
- The same exact issues as the magic monitor!
Add the __mutable storage class
  - Much cheaper than a qualifier
Cancels transitive application of qualifiers on field access
Applies to private members only
Can only be manipulated by @system code
  - mutable in mutable objects
  - shared in shared objects
  - const in const objects
Interested? Let’s talk during the Hackathon!
ProtoObject
Fixing classes

- Object: design predates pure, @nogc, @safe, and immutable
- Four primitives: toString, toHash, opCmp, and opEquals
- Each violates some of the attributes/qualifiers
- The static factory doesn’t help either!
Fixing **classes**: Proposed

- ProtoObject becomes the base of all classes
- MonitoredProtoObject inherits ProtoObject
  - Has one `__mutable` field!
- Object inherits MonitoredProtoObject
- Object remains the default base class
  - And the one introducing factory
  - 100% backward compatible
  - 100% forward looking
factory

- Currently: forces all classes in a lib to be linked in
- Better: use an opt-in interface
- Interface registers factory function with registry

```java
interface Sweatshop<T> {
  ...
}

class Product : Sweatshop!Product {
  ...
}
```
Old code continues to work
New code should inherit ProtoObject
Implement primitives with better techniques
  - Interfaces
  - Templates
  - Visitation
Clunks like monitor and factory are opt-in
Copying Objects
**this(this)**

- Intent: avoid multiple maintenance points
- Design predates introspection
- Today trivially solved

```c
struct S {

    ... 

    this(ref S rhs) {
        foreach (i, e; rhs.tupleof) 
            this.tupleof[i] = e;
    }

}
```
this(this)

- Design predates immutable, pure
- Currently not typechecked properly
- Defining typechecking would be a major effort
- A Pyrrhic victory even if done perfectly
  - Very complex
  - Very unlike the rest of D
Plan

- Design and use copy constructors
- Leverage typechecking in constructors
- Virtually no learning curve
- No new work invested in fixing \texttt{this(this)}
  - Continue accepting it as is
  - Marginalize
  - Deprecate
Copying vs. Moving

- Fundamentally different operations
- When moving, source and target always have same type
- Moving does not duplicate resources
- Intercepting moves subject of a different DIP
Systematic Introspection
Various introspection mechanisms:

- `is(typeof(e))`, `is(typeof(e) == T)`
- `__traits(isThat, T)`
- `std.traits`
- Atomic option: `__traits(compiles, e)`
Issues

- No underlying framework
- Inconsistent “API”
- Awkward to use
- Fun with ParameterStorageClassTuple, anyone?
- Tenuous handing of function overloads
Vision

- Introspection framework
- Structure follows declaration structure:
  - Open some module with `Module!"name"
  - Inside: data/types/function
der(chedule|mination)
  - Each has specific information attached
  - Hierarchical access follows declaration scopes
Example: data

- Get all global definitions:

```cpp
struct Data {
    string name;
    string type;
    string[] attributes;
}
...
enum Data[] d = Module!"mymod".data;
```
**Example: functions**

```c++
struct Function {
    string name;
    string type;
    string resultType;
    string resultModifier; // "" or "ref"
    Parameter[] params;
    string[] attributes;
}
...
enum Function[] d = Module!"mymod".functions;
```
Example: functions

```cpp
struct Parameter {
    string type;
    string modifier; // ", "out", or "ref"
    string[] attributes;
}
```
Approach

- Simple, self-explanatory data structures
- No insistence on hierarchies
- Prefer CTFE to templates
  - Strings that can be mixed in
- Wherever possible allow CT and RT use
Compile-time: what do we want?

- Detailed module information
  - Data
  - Types
  - Aliases
  - Enums
  - Functions
  - Module cdtors
  - Unittests
  - ...

- Use easily done with `mixin` + simple wrappers
Run-time: what do we want?

- Essential/interface module information
  - Types
  - Functions
- Create objects dynamically
- Invoke functions dynamically
  - Use Variant for params, results
- No need to support the entire language!
  - No ref, out, ...
  - Client decides on @safe etc. at bind time
To Conclude
One Theme to Unify Them All
One Theme to Unify Them All

- **mutable**: enable refcounting w. `immutable` `@nogc pure @safe`
One Theme to Unify Them All

- __mutable: enable refcounting w. immutable @nogc pure @safe
- ProtoObject: classes that work w. immutable @nogc pure @safe
• `mutable`: enable refcounting w. `immutable @nogc pure @safe`
• `ProtoObject`: classes that work w. `immutable @nogc pure @safe`
• `this(this)`: encapsulated types that work w. `immutable @nogc pure @safe`
● `__mutable`: enable refcounting w. `immutable @nogc pure @safe`
● `ProtoObject`: classes that work w. `immutable @nogc pure @safe`
● `this(this)`: encapsulated types that work w. `immutable @nogc pure @safe`
● `Introspection`: whaaaaa?
A Good Programming Language
Enforces its own abstractions
Celebrates its own abstractions
immutable @nogc pure @safe
~this()