Array Slicing *fo' Shizzle!*

Some Background

- 2001: Web servers (Tomcat, Resin, WebLogic, etc)
- 2002: Bull in a china shop

A Java Enigma

- Overt GC reliance has potential to make any language or system appear sluggish
- Java has few easy-to-apply alternatives
- Libraries & systems can appear GC-bound
- Heap can be a synchronization bottleneck
- Yet, Java flourished on the server

GC Pressure Reduction

- Low-level access; malloc/free etc
- Precision new & delete overrides
- Temporal struct or class on the stack
- Free-lists
- *Array slicing*

Array Slicing fo' Shizzle!

- Easy-to-use array aliasing
- Native to the language

auto ***slice*** = array [**start .. end**];

Three Use-Cases

- Text Processing
- Sliced HTTP
- Clustering with Tina

1: Integer as Text

```
char[] uitoa (uint val, char[] output=null)
{
    if (output.length < uint.dig)
        output.length = uint.dig;
    auto len = output.length;
    do {
        output [--len] = val % 10;
        } while (val /= 10);
    return output [len .. $];
}
```

1: UTF Conversion

```
dchar[] toUtf32 (char[] input, dchar[] output=null)
{
    if (output.length < input.length)
        output.length = input.length;

    // convert input utf8 into output utf32
    // …

    return output [0 .. len];
}
```

1: Parsing Lines of Text

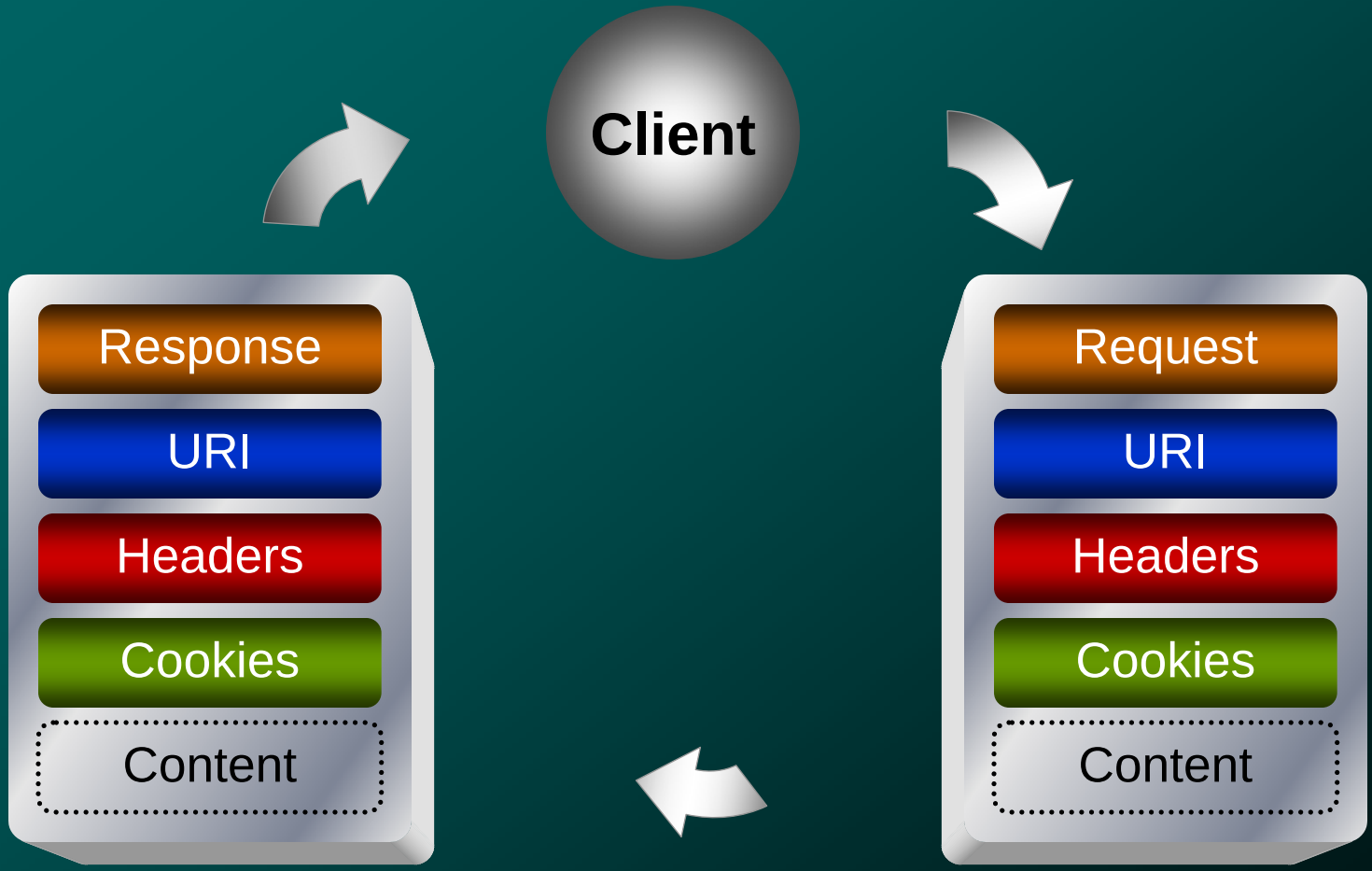
```
// option 1 via an array and .dup
foreach (line; splitLines (myText))
            …

// option 2 via opApply & slicing
foreach (line; lines (myText))
            …
```

The quick
brown fox
jumps over
the lazy
dog

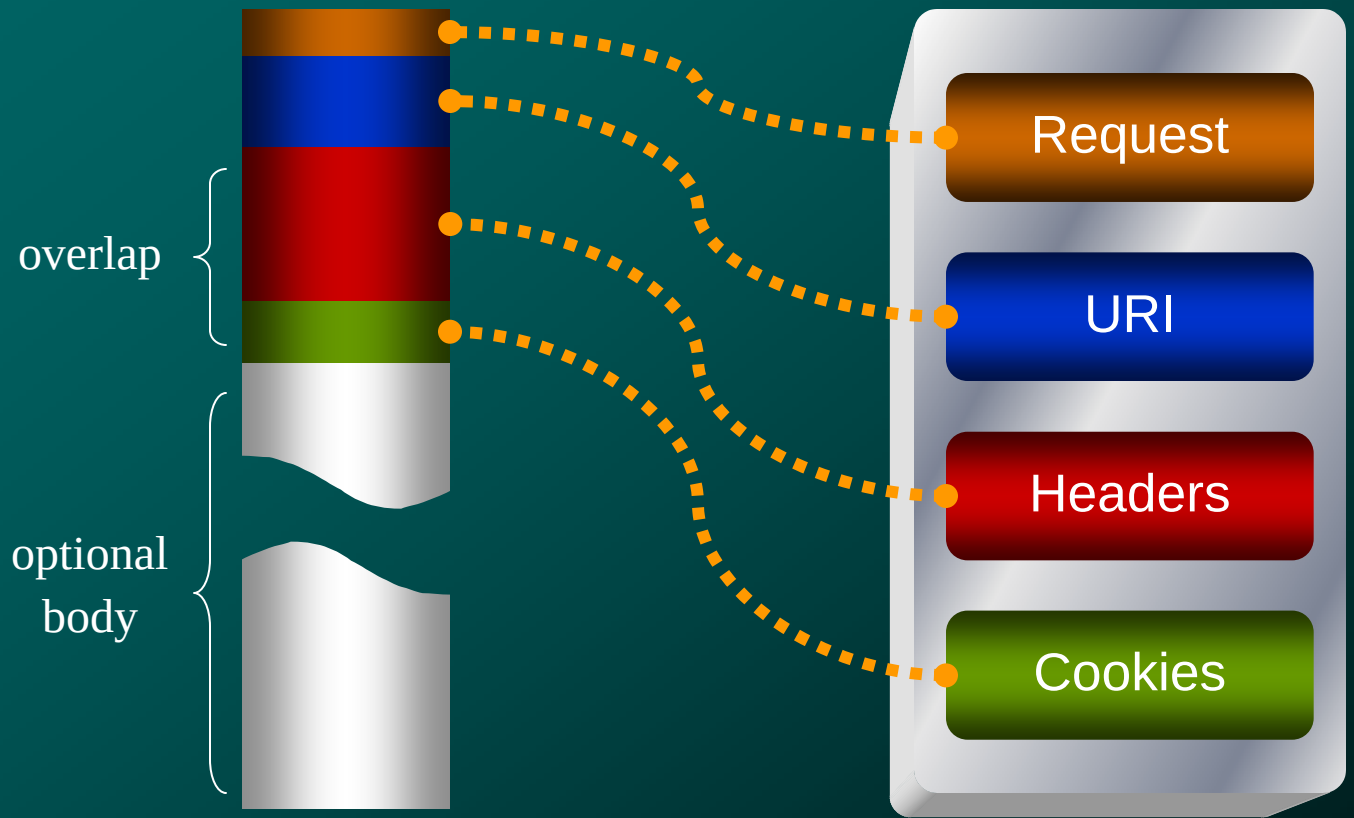
How now
brown cow

The rain in
Spain falls
mainly on

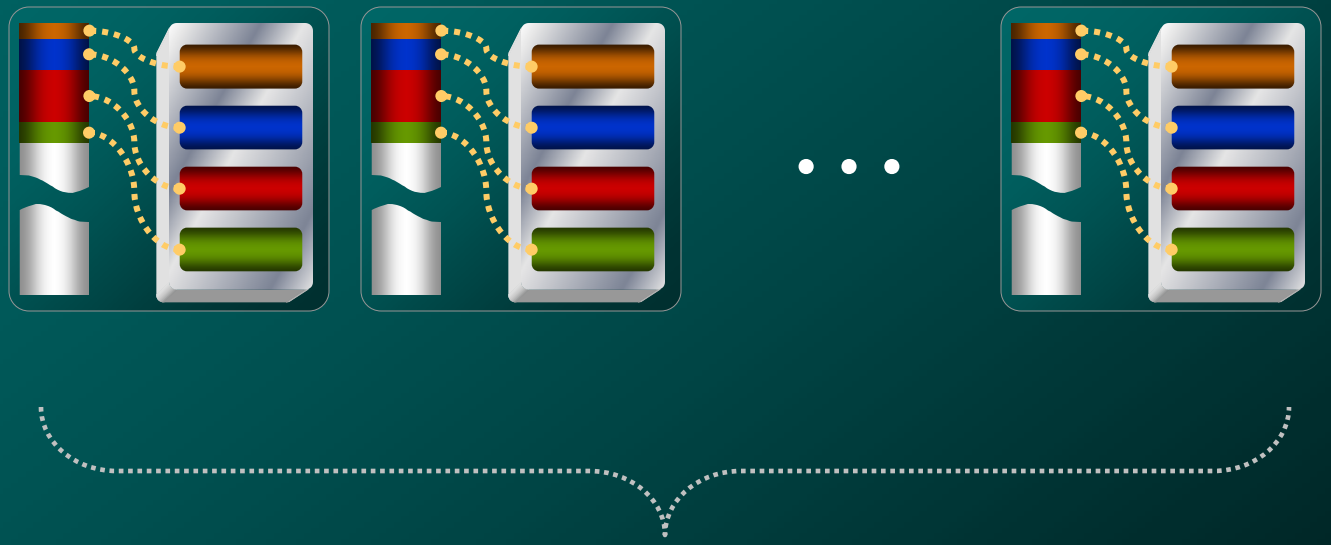
2: Sliced HTTP



2: Sliced HTTP Server

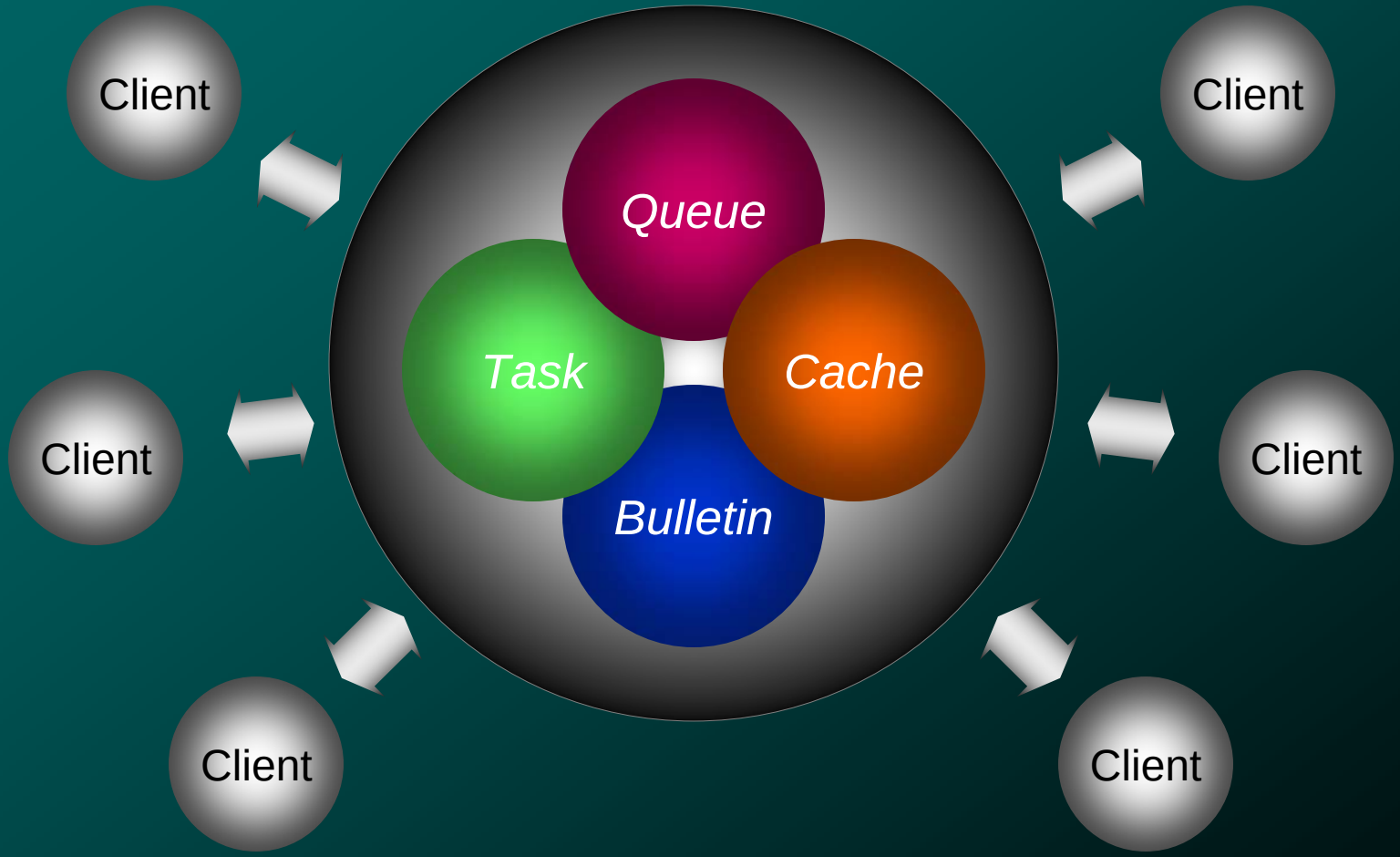


2: Thread Local Storage

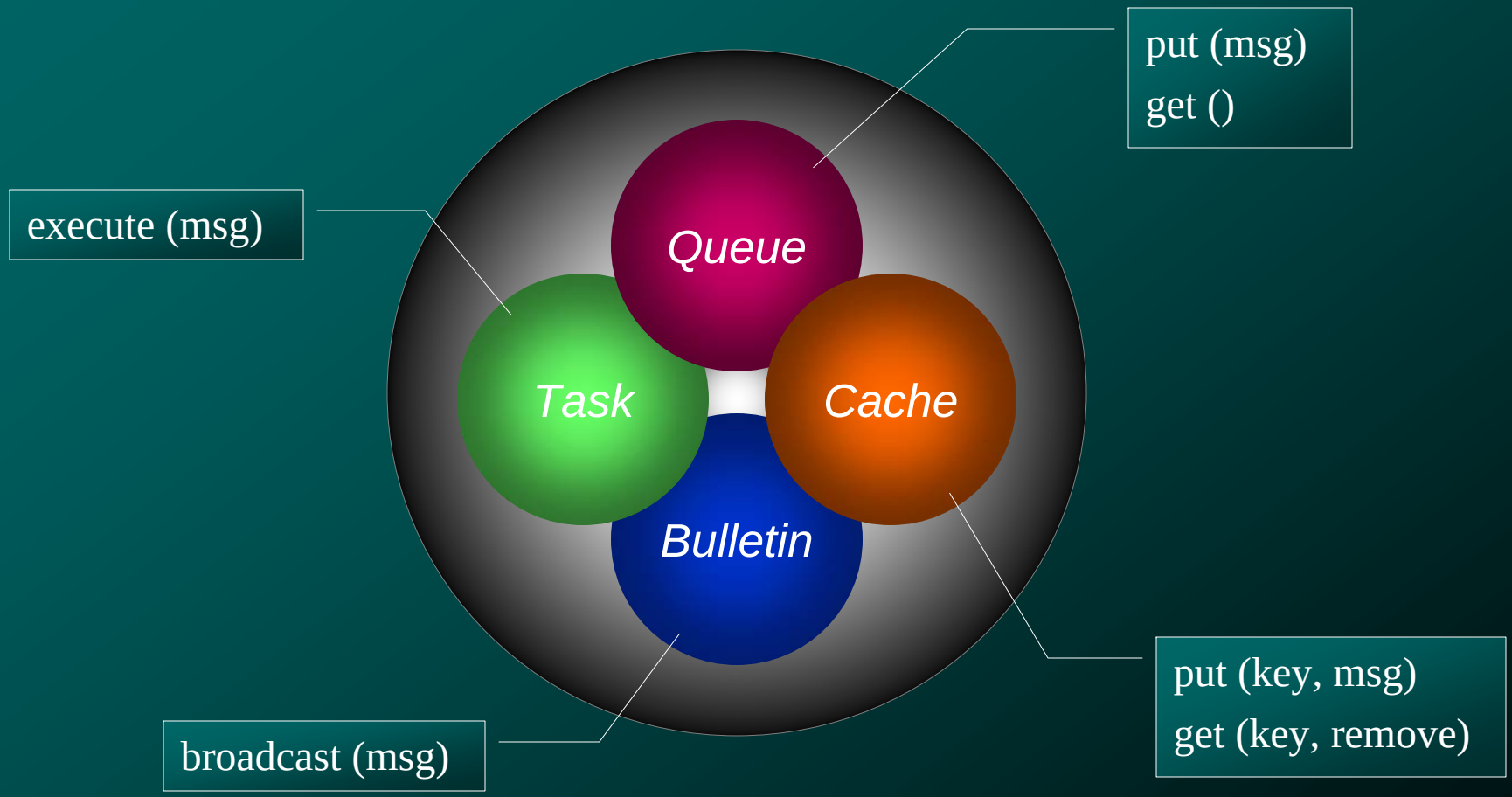


service pool

3: Clustering Model



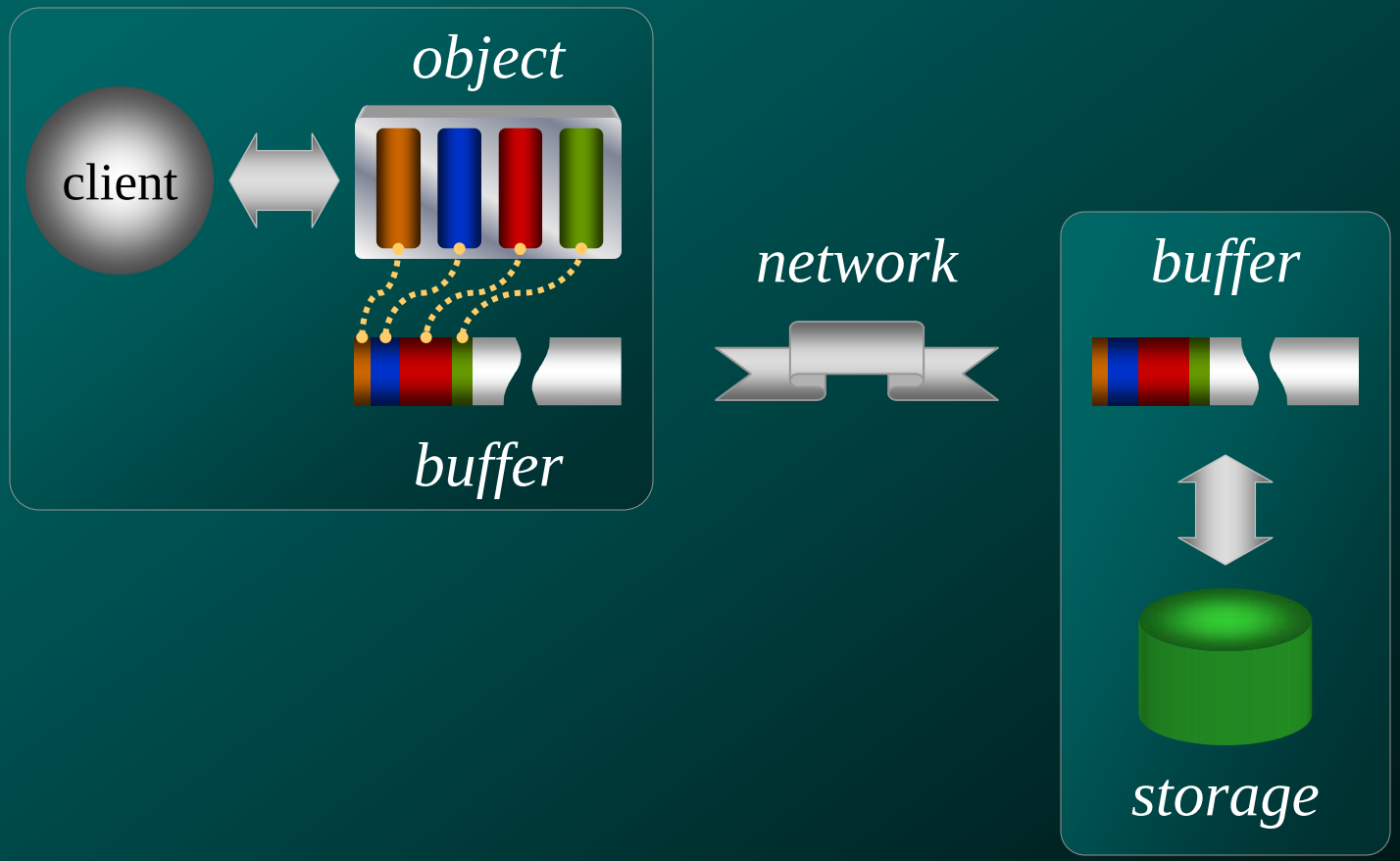
3: Model Services



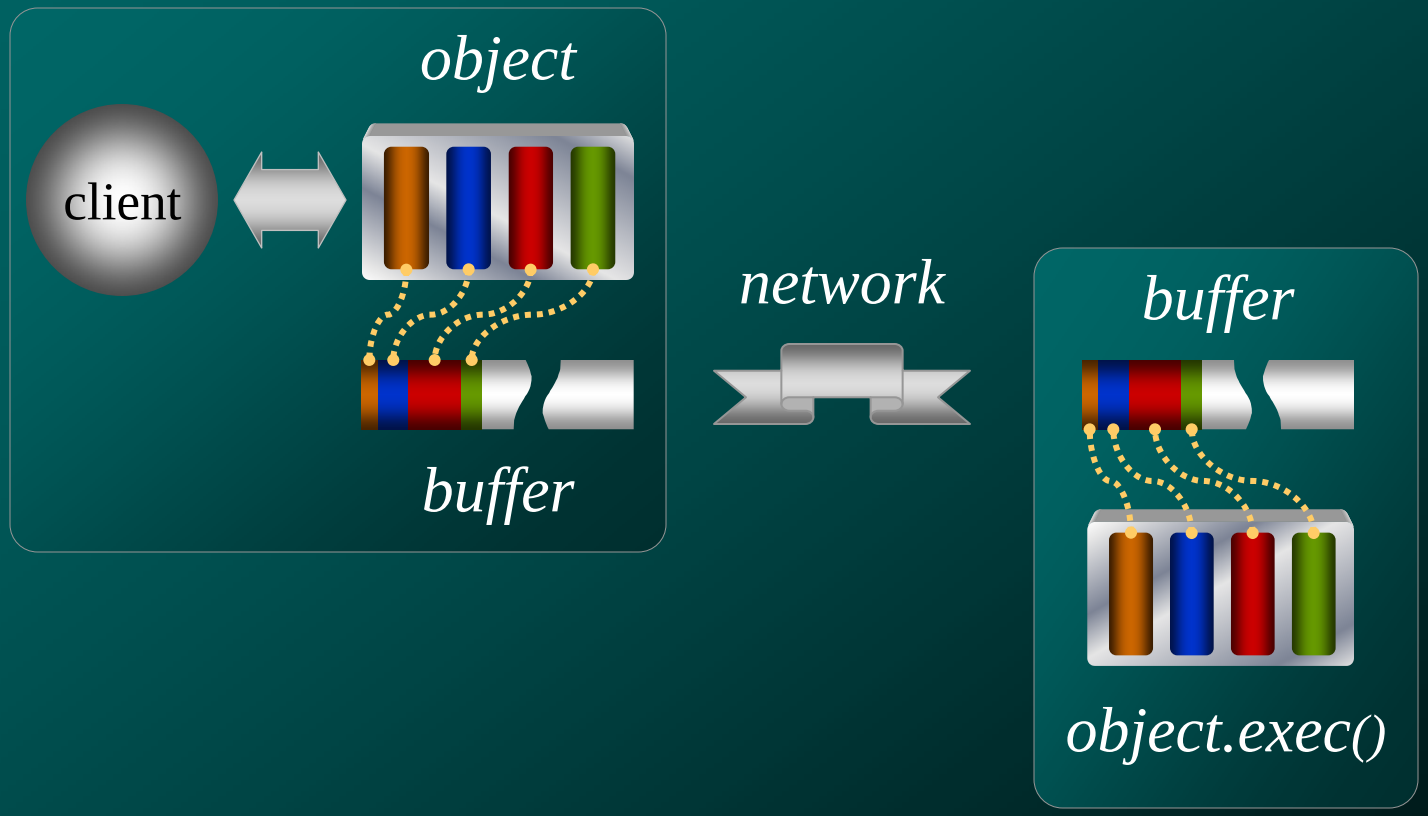
3: A Cluster called Tina

- High performance
- Peer to peer design (no SPOF)
- Serialized D objects
- Near zero GC activity

3: Storage Request



3: Task Request



Tango Connection

- Text processing supports slicing
  - Conversion
  - Tokenizing
  - Streaming
  - Utilities
- IO subsystem supports slicing
  - Protocol readers (allocators)
  - Buffering

Wrap Up

- Overt GC reliance has potential to make any language or system appear sluggish
- Array slicing can be a powerful alternative
- D libraries & systems can easily be GC-lite
- Less multi-thread contention
- Show some GC love

Thanks!