Conquering Callback Mountain

Grokking non-blocking code with Mojo::IOLoop::Delay

Follow along!

github.com/kanatohodets/delays-talk

Where's the (block) party?

- Ask Obama
  - Beers @ White House 1m30s
- Ask Putin
  - Shirtless fishing in Baikal 45s
- Ask Trump
  - Pancakes @ Trump Tower 2m15s
- Time to party (TTP): **4m30s**

Where's the party?

- Ask Obama
- Ask Trump
- Ask Putin

- Putin: Baikal 45s
- Obama: White House 1m30s
- Trump: Trump Tower 2m15s

TTP?

2m 15s!

Amazing Non-blockage!

- Faster response times
- More efficient resource usage
- Hip / trendy / used by rockstar ninjas

WEBSCALE

Show me the code!

```
use Mojolicious::Lite;

get '/' => sub ($c) {
    $c->render(text => "HELLO");
};

app->start;
```

Show me the code!

```
use Mojolicious::Lite;

get '/' => sub ($c) {
    $c->render(text => "HELLO");
};

app->start;
```

Show me the code!

```
use Mojolicious::Lite;

get '/' => sub ($c) {
    $c->render(text => "HELLO");
};

app->start;
```

Show me the code!

```
use Mojolicious::Lite;

get '/' => sub ($c) {
    $c->render(text => "HELLO");
};

app->start;
```

Show me the code!

```
use Mojolicious::Lite;

get '/' => sub ($c) {
    $c->render(text => "HELLO");
};

app->start;
```

Show me the code!

```
use Mojolicious::Lite;

get '/' => sub ($c) {
    $c->render(text => "HELLO");
};

app->start;
```

Show me the code!

```
use Mojolicious::Lite;

get '/' => sub ($c) {
    $c->render(text => "HELLO");
};

app->start;
```

Show me the code!

```
use Mojolicious::Lite;

get '/' => sub ($c) {
    $c->render(text => "HELLO");
};

app->start;
```

Magic

Async secrets revealed:

TCP chat server

(because parsing HTTP is kinda complicated)

Event loop!

- poll, select, epoll, kqueue, etc.
- Let's get close to the metal.
- Client: telnet

Event loop!

- poll, select, epoll, kqueue, etc.
- Let's get close to the metal.
- Client: telnet

Event loop!

- poll, select, epoll, kqueue, etc.
- Let's get close to the metal.
- Client: telnet

Event loop!

oll, select, epoll,
queue, etc.

et's get close to the

Event loop!

l, select, epoll,

ueue, etc.

Event loop!

ect,

queue

Prelude: IO::Select

```
#!/usr/bin/env perl
use 5.22.0; use strict;
use IO::Select;
use IO::Socket;
use experimental qw(signatures postderef);

my $listen = IO::Socket::INET->new(Listen =>
1, LocalPort => 8080, ReuseAddr => 1)
    or die "can't bind listen socket: $!";

my $select = IO::Select->new( $listen );
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Wait a second!

```
on_read => sub ($data) {
    <broadcast stuff>
}
```

Where have we seen that before?

Bingo!

```
use Mojolicious::Lite;

get '/' => sub ($c) {
    $c->render(text => "HELLO");
};

app->start;
```

Bingo!

```
get '/' => sub ($c) {
    $c->render(text => "HELLO");
};
```

and

```
on_read => sub ($data) {
    <broadcast stuff>
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

Event Loop: TCP chat

```
my (@broadcast, %handlers);
while (1) {
    for my $sock ($select->can_read(1)) {
        if ($sock == $listen) {
            my $new = $listen->accept;
            $select->add($new);
            $handlers{$new->fileno} = {
                on_read => sub ($data) { push @broadcast, $new->fileno . ": $data" }
            };
            push @broadcast, sprintf("* %s connected\n", $new->fileno);
        } else {
            if ($sock->sysread(my $buffer, 4096, 0)) {
                $handlers{$sock->fileno}->{on_read}->($buffer);
            } else {
                push @broadcast, sprintf("* %s disconnected\n", $sock->fileno);
                $select->remove($sock) and $sock->close;
            }
        }
    }
    if (my @writable = $select->can_write(1)) {
        if (my $message = shift @broadcast) {
            $_->print($message) for @writable;
        }
    }
}
```

BOOM, WhatsApp killer!

How about some HTTP?

Price Gouging Bar API

- The more you buy, the more you pay.
- First beer = $3.
- Second beer = $5, etc.
- We need a scalable backend!

Price Gouging Bar API

Prelude: Mojo

```
#!/usr/bin/env perl
use v5.22.0; use warnings;
use Mojolicious::Lite;
use Mojo::Pg;
use Mojo::IOLoop;
use experimental qw(signatures postderef);

helper pg =>
    sub { state $pg = Mojo::Pg->new('postgresql://
btyler@localhost/my_cool_db') };
#… stuff
app->start;
```

Price Gouging Bar API

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```

Price Gouging Bar API

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```

Price Gouging Bar API

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```

Price Gouging Bar API

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```

Price Gouging Bar API

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```

Price Gouging Bar API

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```

Price Gouging Bar API

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```

Price Gouging Bar API

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```

Price Gouging Bar API

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```

Price Gouging Bar API

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```

Price Gouging Bar API

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```

Price Gouging Bar API

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```

Price Gouging Bar API

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```

Price Gouging Bar API

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```

Price Gouging Bar API

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```

Argh!

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```

Argh!

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```

Argh!

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```


EVERYTHING IS
TERRIBLE

A more structured approach

```
# … stuff
my $delay = Mojo::IOLoop->delay(
    sub ($d) {  }, # step 1
    sub ($d, @args) {  }, # step 2
    sub ($d, @args) {  }, # step 3
    sub ($d, @args) {  } # finish
)->wait;
```

A more structured approach

```
# … stuff
my $delay = Mojo::IOLoop->delay(
    sub ($d) {  }, # step 1
    sub ($d, @args) {  }, # step 2
    sub ($d, @args) {  }, # step 3
    sub ($d, @args) {  } # finish
)->wait;
```

A more structured approach

```
# … stuff
my $delay = Mojo::IOLoop->delay(
    sub ($d) {  }, # step 1
    sub ($d, @args) {  }, # step 2
    sub ($d, @args) {  }, # step 3
    sub ($d, @args) {  } # finish
)->wait;
```

A more structured approach

```
# … stuff
my $delay = Mojo::IOLoop->delay(
    sub ($d) {  }, # step 1
    sub ($d, @args) {  }, # step 2
    sub ($d, @args) {  }, # step 3
    sub ($d, @args) {  } # finish
)->wait;
```

A more structured approach

```
# … stuff
my $delay = Mojo::IOLoop->delay(
    sub ($d) {  }, # step 1
    sub ($d, @args) {  }, # step 2
    sub ($d, @args) {  }, # step 3
    sub ($d, @args) {  } # finish
)->wait;
```

A more structured approach

```
# … stuff
my $delay = Mojo::IOLoop->delay(
    sub ($d) {  }, # step 1
    sub ($d, @args) {  }, # step 2
    sub ($d, @args) {  }, # step 3
    sub ($d, @args) {  } # finish
)->wait;
```

Structure (plus errors)

```
# … stuff
my $delay = Mojo::IOLoop->delay(
    sub ($d) {  }, # step 1
    sub ($d, @args) {  }, # step 2
    sub ($d, @args) {  }, # step 3
    sub ($d, @args) {  } # finish
)->catch(sub ($d, $err) {
    #blorg, an error!
})->wait;
```

Structure (plus errors)

```
# … stuff
my $delay = Mojo::IOLoop->delay(
    sub ($d) {  }, # step 1
    sub ($d, @args) {  }, # step 2
    sub ($d, @args) {  }, # step 3
    sub ($d, @args) {  } # finish
)->catch(sub ($d, $err) {
    #blorg, an error!
})->wait;
```

Delays!

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $delay = Mojo::IOLoop->delay(
        sub ($d) {
            $d->data(name => $name);
            $c->pg->db->query('SELECT id FROM customers WHERE name = ?', $name, $d->begin);
        },
        sub ($d, $err, $res) {
            die $err if $err;
            my $customer_id = $res->array->[0];

            $d->data(customer_id => $customer_id);
            $c->pg->db->query('SELECT COUNT(1) FROM beer_log WHERE customer_id = ?', $customer_id, $d->begin);
        },
        sub ($d, $err, $res) {
            die $err if $err;
            my $count = $res->array->[0];

            $d->data(count => $count);
            $c->pg->db->query('SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start', $count, $d->begin);
        },
        sub ($d, $err, $res) {
            die $err if $err;
            my $price = $res->array->[0];

            my ($name, $customer_id, $count) = $d->data->@{qw(name customer_id count)};
            $c->render(json => { name => $name, id => $customer_id, beers => $count, price => $price });
        })->catch(sub ($d, $err) {
            $c->render(text => "blorg error! $err", code => 500);
    })->wait;
};
```

Delays!

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $delay = Mojo::IOLoop->delay(
        sub ($d) {
            $d->data(name => $name);
            $c->pg->db->query('SELECT id FROM customers WHERE name = ?', $name, $d->begin);
        },
        sub ($d, $err, $res) {
            die $err if $err;
            my $customer_id = $res->array->[0];

            $d->data(customer_id => $customer_id);
            $c->pg->db->query('SELECT COUNT(1) FROM beer_log WHERE customer_id = ?', $customer_id, $d->begin);
        },
        sub ($d, $err, $res) {
            die $err if $err;
            my $count = $res->array->[0];

            $d->data(count => $count);
            $c->pg->db->query('SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start', $count, $d->begin);
        },
        sub ($d, $err, $res) {
            die $err if $err;
            my $price = $res->array->[0];

            my ($name, $customer_id, $count) = $d->data->@{qw(name customer_id count)};
            $c->render(json => { name => $name, id => $customer_id, beers => $count, price => $price });
        })->catch(sub ($d, $err) {
            $c->render(text => "blorg error! $err", code => 500);
    })->wait;
};
```

Delays!

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    # … stuff
};
```

Delays!

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $delay = Mojo::IOLoop->delay(
        sub ($d) {
            $d->data(name => $name);
            $c->pg->db->query('SELECT id FROM customers WHERE name = ?', $name, $d->begin);
        },
        sub ($d, $err, $res) {
            die $err if $err;
            my $customer_id = $res->array->[0];

            $d->data(customer_id => $customer_id);
            $c->pg->db->query('SELECT COUNT(1) FROM beer_log WHERE customer_id = ?', $customer_id, $d->begin);
        },
        sub ($d, $err, $res) {
            die $err if $err;
            my $count = $res->array->[0];

            $d->data(count => $count);
            $c->pg->db->query('SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start', $count, $d->begin);
        },
        sub ($d, $err, $res) {
            die $err if $err;
            my $price = $res->array->[0];

            my ($name, $customer_id, $count) = $d->data->@{qw(name customer_id count)};
            $c->render(json => { name => $name, id => $customer_id, beers => $count, price => $price });
        })->catch(sub ($d, $err) {
            $c->render(text => "blorg error! $err", code => 500);
    })->wait;
};
```

(the structure from before)

```
# … stuff
my $delay = Mojo::IOLoop->delay(
    sub ($d) {  }, # step 1
    sub ($d, @args) {  }, # step 2
    sub ($d, @args) {  }, # step 3
    sub ($d, @args) {  } # finish
)->wait;
```

Delays!

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $delay = Mojo::IOLoop->delay(
        sub ($d) {
            $d->data(name => $name);
            $c->pg->db->query('SELECT id FROM customers WHERE name = ?', $name, $d->begin);
        },
        sub ($d, $err, $res) {
            die $err if $err;
            my $customer_id = $res->array->[0];

            $d->data(customer_id => $customer_id);
            $c->pg->db->query('SELECT COUNT(1) FROM beer_log WHERE customer_id = ?', $customer_id, $d->begin);
        },
        sub ($d, $err, $res) {
            die $err if $err;
            my $count = $res->array->[0];

            $d->data(count => $count);
            $c->pg->db->query('SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start', $count => $d->begin);
        },
        sub ($d, $err, $res) {
            die $err if $err;
            my $price = $res->array->[0];

            my ($name, $customer_id, $count) = $d->data->@{qw(name customer_id count)};
            $c->render(json => { name => $name, id => $customer_id, beers => $count, price => $price });
        })->catch(sub ($d, $err) {
            $c->render(text => "blorg error! $err", code => 500);
    })->wait;
};
```

First Steps

```
sub ($d) {
    $d->data(name => $name);
    $c->pg->db->query('SELECT id FROM customers
WHERE name = ?', $name, $d->begin);
},
sub ($d, $err, $res) {
    die $err if $err;
    my $customer_id = $res->array->[0];

    $d->data(customer_id => $customer_id);
    $c->pg->db->query('SELECT COUNT(1) FROM beer_log
WHERE customer_id = ?', $customer_id, $d->begin);
},
# … more steps in the delay
```

First Steps

```
sub ($d) {
    $d->data(name => $name);
    $c->pg->db->query('SELECT id FROM customers
WHERE name = ?', $name, $d->begin);
},
sub ($d, $err, $res) {
    die $err if $err;
    my $customer_id = $res->array->[0];

    $d->data(customer_id => $customer_id);
    $c->pg->db->query('SELECT COUNT(1) FROM beer_log
WHERE customer_id = ?', $customer_id, $d->begin);
},
# … more steps in the delay
```

First Steps

```
sub ($d) {
    $d->data(name => $name);
    $c->pg->db->query('SELECT id FROM customers
WHERE name = ?', $name, $d->begin);
},
sub ($d, $err, $res) {
    die $err if $err;
    my $customer_id = $res->array->[0];

    $d->data(customer_id => $customer_id);
    $c->pg->db->query('SELECT COUNT(1) FROM beer_log
WHERE customer_id = ?', $customer_id, $d->begin);
},
# … more steps in the delay
```

First Steps

```
sub ($d) {
    $d->data(name => $name);
    $c->pg->db->query('SELECT id FROM customers
WHERE name = ?', $name, $d->begin);
},
sub ($d, $err, $res) {
    die $err if $err;
    my $customer_id = $res->array->[0];

    $d->data(customer_id => $customer_id);
    $c->pg->db->query('SELECT COUNT(1) FROM beer_log
WHERE customer_id = ?', $customer_id, $d->begin);
},
# … more steps in the delay
```

First Steps

```
sub ($d) {
    $d->data(name => $name);
    $c->pg->db->query('SELECT id FROM customers
WHERE name = ?', $name, $d->begin);
},
sub ($d, $err, $res) {
    die $err if $err;
    my $customer_id = $res->array->[0];

    $d->data(customer_id => $customer_id);
    $c->pg->db->query('SELECT COUNT(1) FROM beer_log
WHERE customer_id = ?', $customer_id, $d->begin);
},
# … more steps in the delay
```

First Steps

```
sub ($d) {
    $d->data(name => $name);
    $c->pg->db->query('SELECT id FROM customers
WHERE name = ?', $name, $d->begin);
},
sub ($d, $err, $res) {
    die $err if $err;
    my $customer_id = $res->array->[0];

    $d->data(customer_id => $customer_id);
    $c->pg->db->query('SELECT COUNT(1) FROM beer_log
WHERE customer_id = ?', $customer_id, $d->begin);
},
# … more steps in the delay
```

First Steps

```
sub ($d) {
    $d->data(name => $name);
    $c->pg->db->query('SELECT id FROM customers
WHERE name = ?', $name, $d->begin);
},
sub ($d, $err, $res) {
    die $err if $err;
    my $customer_id = $res->array->[0];

    $d->data(customer_id => $customer_id);
    $c->pg->db->query('SELECT COUNT(1) FROM beer_log
WHERE customer_id = ?', $customer_id, $d->begin);
},
# … more steps in the delay
```

First Steps

```
sub ($d) {
    $d->data(name => $name);
    $c->pg->db->query('SELECT id FROM customers
WHERE name = ?', $name, $d->begin);
},
sub ($d, $err, $res) {
    die $err if $err;
    my $customer_id = $res->array->[0];

    $d->data(customer_id => $customer_id);
    $c->pg->db->query('SELECT COUNT(1) FROM beer_log
WHERE customer_id = ?', $customer_id, $d->begin);
},
# … more steps in the delay
```

First Steps

```
sub ($d) {
    $d->data(name => $name);
    $c->pg->db->query('SELECT id FROM customers
WHERE name = ?', $name, $d->begin);
},
sub ($d, $err, $res) {
    die $err if $err;
    my $customer_id = $res->array->[0];

    $d->data(customer_id => $customer_id);
    $c->pg->db->query('SELECT COUNT(1) FROM beer_log
WHERE customer_id = ?', $customer_id, $d->begin);
},
# … more steps in the delay
```

Delays!

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $delay = Mojo::IOLoop->delay(
        sub ($d) {
            $d->data(name => $name);
            $c->pg->db->query('SELECT id FROM customers WHERE name = ?', $name => $d->begin);
        },
        sub ($d, $err, $res) {
            die $err if $err;
            my $customer_id = $res->array->[0];

            $d->data(customer_id => $customer_id);
            $c->pg->db->query('SELECT COUNT(1) FROM beer_log WHERE customer_id = ?', $customer_id, $d->begin);
        },
        sub ($d, $err, $res) {
            die $err if $err;
            my $count = $res->array->[0];

            $d->data(count => $count);
            $c->pg->db->query('SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start', $count, $d->begin);
        },
        sub ($d, $err, $res) {
            die $err if $err;
            my $price = $res->array->[0];

            my ($name, $customer_id, $count) = $d->data->@{qw(name customer_id count)};
            $c->render(json => { name => $name, id => $customer_id, beers => $count, price => $price });
        })->catch(sub ($d, $err) {
            $c->render(text => "blorg error! $err", code => 500);
    })->wait;
};
```

Final Steps

```
# … previous delay steps
sub ($d, $err, $res) {
    die $err if $err;
    my $count = $res->array->[0];

    $d->data(count => $count);
    $c->pg->db->query('SELECT MAX(price) FROM beer_price_scale WHERE ?
>= range_start', $count, $d->begin);
},
sub ($d, $err, $res) {
    die $err if $err;
    my $price = $res->array->[0];

    my ($name, $customer_id, $count) =
        $d->data->@{qw(name customer_id count)};
    $c->render(json => { name => $name, id => $customer_id, beers =>
$count, price => $price });
}
# … catch and ->wait
```

Final Steps

```
# … previous delay steps
sub ($d, $err, $res) {
    die $err if $err;
    my $count = $res->array->[0];

    $d->data(count => $count);
    $c->pg->db->query('SELECT MAX(price) FROM beer_price_scale WHERE ?
>= range_start', $count, $d->begin);
},
sub ($d, $err, $res) {
    die $err if $err;
    my $price = $res->array->[0];

    my ($name, $customer_id, $count) =
        $d->data->@{qw(name customer_id count)};
    $c->render(json => { name => $name, id => $customer_id, beers =>
$count, price => $price });
}
# … catch and ->wait
```

Final Steps

```
# … previous delay steps
sub ($d, $err, $res) {
    die $err if $err;
    my $count = $res->array->[0];

    $d->data(count => $count);
    $c->pg->db->query('SELECT MAX(price) FROM beer_price_scale WHERE ?
>= range_start', $count, $d->begin);
},
sub ($d, $err, $res) {
    die $err if $err;
    my $price = $res->array->[0];

    my ($name, $customer_id, $count) =
        $d->data->@{qw(name customer_id count)};
    $c->render(json => { name => $name, id => $customer_id, beers =>
$count, price => $price });
}
# … catch and ->wait
```

Final Steps

```
# … previous delay steps
sub ($d, $err, $res) {
    die $err if $err;
    my $count = $res->array->[0];

    $d->data(count => $count);
    $c->pg->db->query('SELECT MAX(price) FROM beer_price_scale WHERE ?
>= range_start', $count, $d->begin);
},
sub ($d, $err, $res) {
    die $err if $err;
    my $price = $res->array->[0];

    my ($name, $customer_id, $count) =
        $d->data->@{qw(name customer_id count)};
    $c->render(json => { name => $name, id => $customer_id, beers =>
$count, price => $price });
}
# … catch and ->wait
```

Final Steps

```
# … previous delay steps
sub ($d, $err, $res) {
    die $err if $err;
    my $count = $res->array->[0];

    $d->data(count => $count);
    $c->pg->db->query('SELECT MAX(price) FROM beer_price_scale WHERE ?
>= range_start', $count, $d->begin);
},
sub ($d, $err, $res) {
    die $err if $err;
    my $price = $res->array->[0];

    my ($name, $customer_id, $count) =
        $d->data->@{qw(name customer_id count)};
    $c->render(json => { name => $name, id => $customer_id, beers =>
$count, price => $price });
}
# … catch and ->wait
```

Final Steps

```
# … previous delay steps
sub ($d, $err, $res) {
    die $err if $err;
    my $count = $res->array->[0];

    $d->data(count => $count);
    $c->pg->db->query('SELECT MAX(price) FROM beer_price_scale WHERE ?
>= range_start', $count, $d->begin);
},
sub ($d, $err, $res) {
    die $err if $err;
    my $price = $res->array->[0];

    my ($name, $customer_id, $count) =
        $d->data->@{qw(name customer_id count)};
    $c->render(json => { name => $name, id => $customer_id, beers =>
$count, price => $price });
}
# … catch and ->wait
```

Final Steps

```
# … previous delay steps
sub ($d, $err, $res) {
    die $err if $err;
    my $count = $res->array->[0];

    $d->data(count => $count);
    $c->pg->db->query('SELECT MAX(price) FROM beer_price_scale WHERE ?
>= range_start', $count, $d->begin);
},
sub ($d, $err, $res) {
    die $err if $err;
    my $price = $res->array->[0];

    my ($name, $customer_id, $count) =
        $d->data->@{qw(name customer_id count)};
    $c->render(json => { name => $name, id => $customer_id, beers =>
$count, price => $price });
}
# … catch and ->wait
```

Final Steps

```
# … previous delay steps
sub ($d, $err, $res) {
    die $err if $err;
    my $count = $res->array->[0];

    $d->data(count => $count);
    $c->pg->db->query('SELECT MAX(price) FROM beer_price_scale WHERE ?
>= range_start', $count, $d->begin);
},
sub ($d, $err, $res) {
    die $err if $err;
    my $price = $res->array->[0];

    my ($name, $customer_id, $count) =
        $d->data->@{qw(name customer_id count)};
    $c->render(json => { name => $name, id => $customer_id, beers =>
$count, price => $price });
}
# … catch and ->wait
```

Ta-da!

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    my $delay = Mojo::IOLoop->delay(
        sub ($d) {
            $d->data(name => $name);
            $c->pg->db->query('SELECT id FROM customers WHERE name = ?', $name, $d->begin);
        },
        sub ($d, $err, $res) {
            die $err if $err;
            my $customer_id = $res->array->[0];

            $d->data(customer_id => $customer_id);
            $c->pg->db->query('SELECT COUNT(1) FROM beer_log WHERE customer_id = ?', $customer_id, $d->begin);
        },
        sub ($d, $err, $res) {
            die $err if $err;
            my $count = $res->array->[0];

            $d->data(count => $count);
            $c->pg->db->query('SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start', $count, $d->begin);
        },
        sub ($d, $err, $res) {
            die $err if $err;
            my $price = $res->array->[0];

            my ($name, $customer_id, $count) = $d->data->@{qw(name customer_id count)};
            $c->render(json => { name => $name, id => $customer_id, beers => $count, price => $price });
        })->catch(sub ($d, $err) {
            $c->render(text => "blorg error! $err", code => 500);
    })->wait;
};
```

Callbacks

```
get '/booze_check' => sub ($c) {
    my $name = $c->param('name');
    $c->render_later;

    my $customer_sql = 'SELECT id FROM customers WHERE name = ?';
    $c->pg->db->query($customer_sql, $name => sub ($db, $err, $res) {
        my $customer_id = $res->expand->hashes->[0]->{id};

        my $count_sql = 'SELECT COUNT(1) FROM beer_log WHERE customer_id = ?';
        $c->pg->db->query($count_sql, $customer_id => sub ($db, $err, $res) {
            my $count = $res->hashes->[0]->{count};

            my $price_sql = 'SELECT MAX(price) FROM beer_price_scale WHERE ? >= range_start';
            $c->pg->db->query($price_sql, $count => sub ($db, $err, $res) {
                my $price = $res->array->[0];

                $c->render(json => { name => $name, id => $customer_id,
                                     beers => $count, price => $price });
            });
        });
    });
};
```

Bonus!

```
get '/multi_query' => sub ($c) {
    $c->render_later;

    my $delay = Mojo::IOLoop->delay(
        sub ($d) {
            $c->pg->db->query('select ?::text, pg_sleep(4)',
                'Obama', $d->begin);
            $c->pg->db->query('select ?::text, pg_sleep(2)',
                'Putin', $d->begin);
        },
        sub ($d, $Obama_err, $obama_res, $putin_err, $putin_res) {
            $c->render(json => {
                obama => $obama_res->hashes->[0],
                putin => $putin_res->hashes->[0]
            });
        }
    )->wait;
};
```

Bonus!

```
get '/multi_query' => sub ($c) {
    $c->render_later;

    my $delay = Mojo::IOLoop->delay(
        sub ($d) {
            $c->pg->db->query('select ?::text, pg_sleep(4)',
                'Obama', $d->begin);
            $c->pg->db->query('select ?::text, pg_sleep(2)',
                'Putin', $d->begin);
        },
        sub ($d, $obama_err, $obama_res, $putin_err, $putin_res) {
            $c->render(json => {
                obama => $obama_res->hashes->[0],
                putin => $putin_res->hashes->[0]
            });
        }
    )->wait;
};
```

Bonus!

```
get '/multi_query' => sub ($c) {
    $c->render_later;

    my $delay = Mojo::IOLoop->delay(
        sub ($d) {
            $c->pg->db->query('select ?::text, pg_sleep(4)',
                'Obama', $d->begin);
            $c->pg->db->query('select ?::text, pg_sleep(2)',
                'Putin', $d->begin);
        },
        sub ($d, $obama_err, $obama_res, $putin_err, $putin_res) {
            $c->render(json => {
                obama => $obama_res->hashes->[0],
                putin => $putin_res->hashes->[0]
            });
        }
    )->wait;
};
```

Questions?