[Introduction to] Writing non-blocking code ... in Node.js and Perl

Jay Janssen
Percona, Inc.

Problems solved

- Standard programs assume sequential execution of statements
- Long running statements block the program while waiting for a response
- Non-blocking programming allows your process to handle other tasks while waiting for long operations to complete

Good use cases

- Servers that need to handle lots of concurrency, but that often block on network or IO operations
  - example: node.js
- A process that coordinates a lot of tasks, in particular timed tasks
  - example: execute an arbitrary set of tasks, each with their own schedule
- Any process that interacts with an external system
  - example: a process that interacts heavily with Disk, Database, webservices, etc.

Overview

- High level Languages: Perl AnyEvent and Node.js, not C/C++ libevent level programming
- Agenda:
  - Intro to non-blocking language concepts
  - Example code in Node.js and Perl
  - Useful libraries
  - Tradeoffs

Intro to non-blocking language concepts

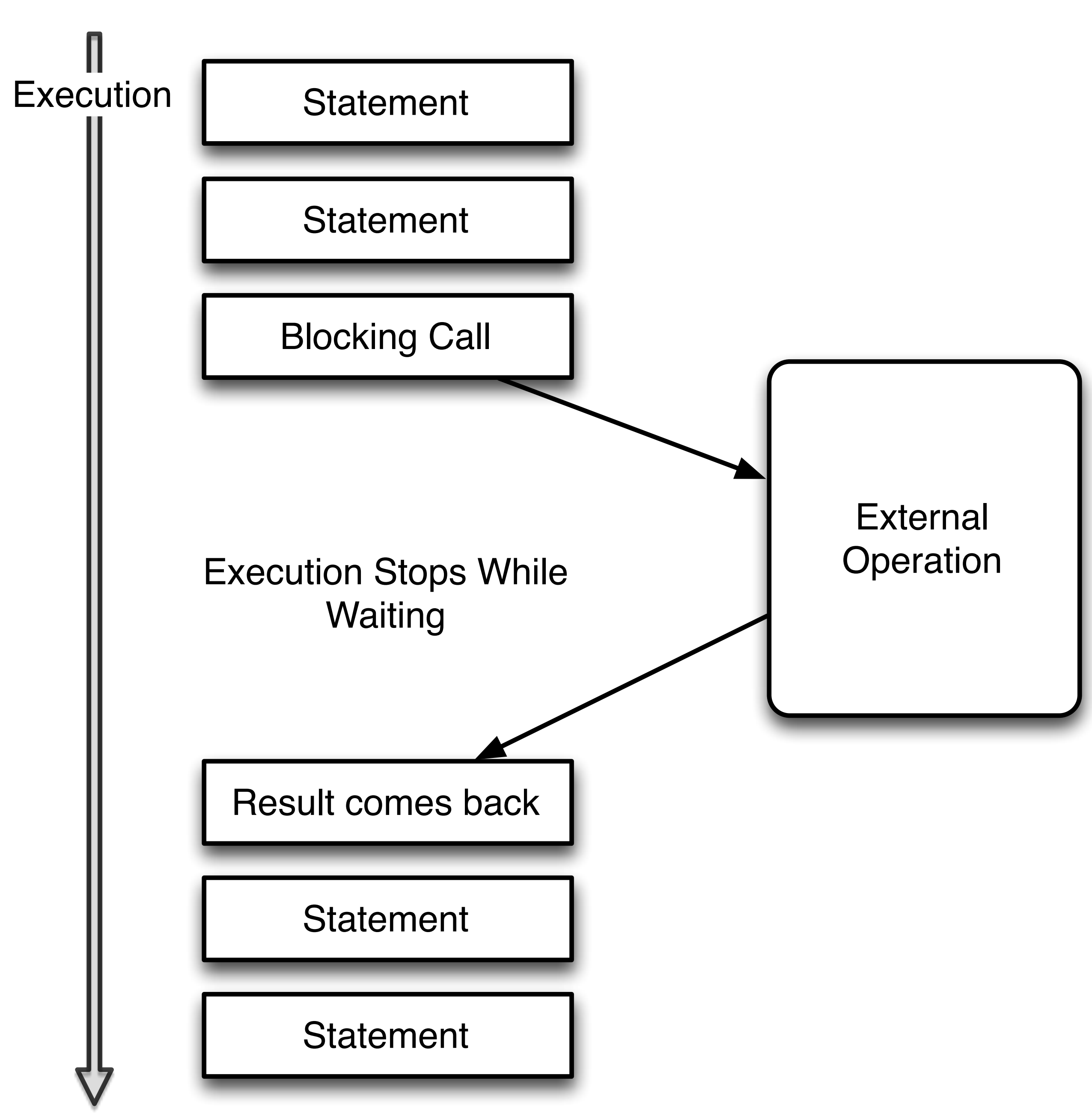
Definitions

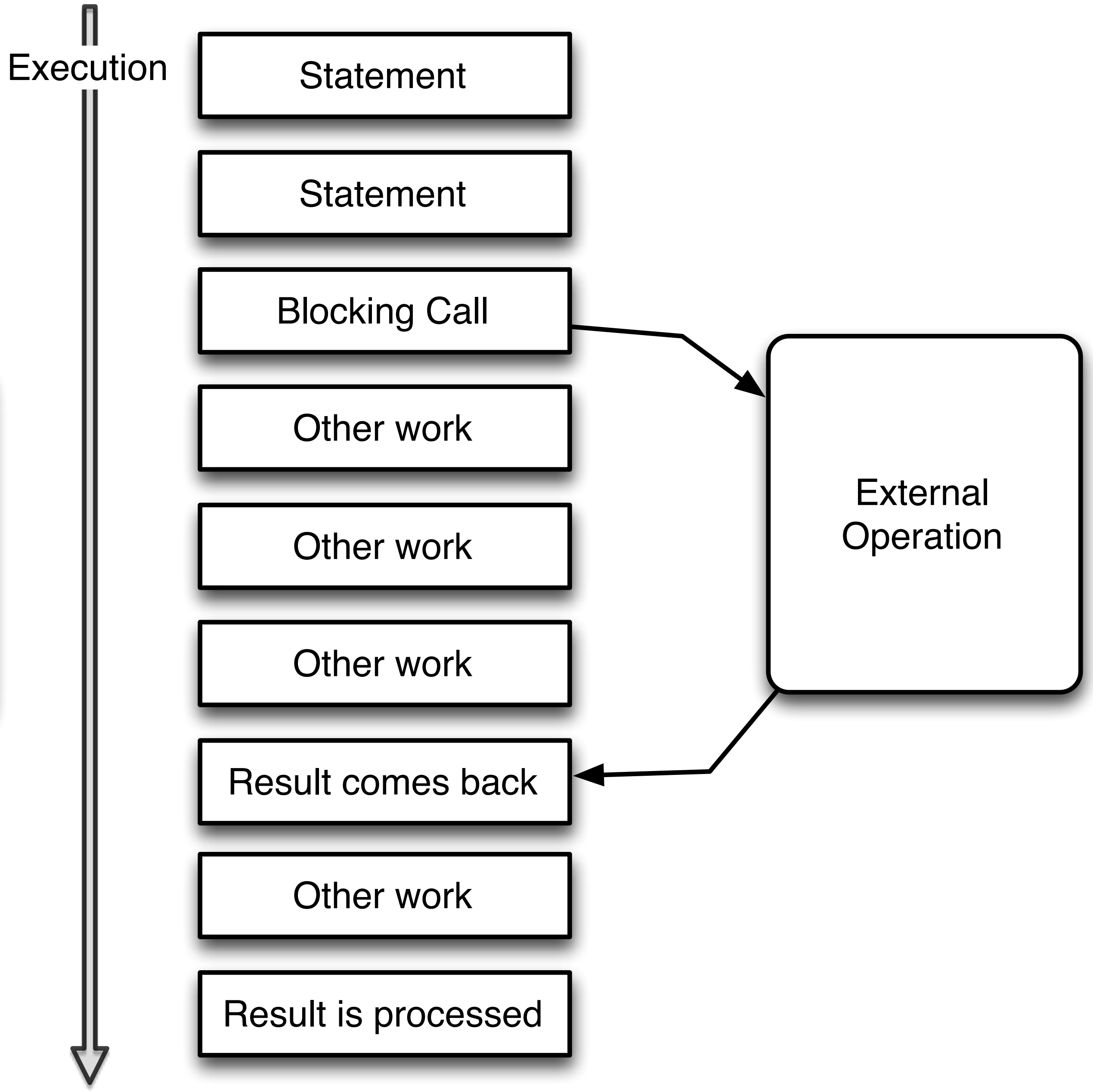
- **procedural single-process**: a process that runs a single execution path
- **procedural threaded:** process running multiple execution paths simultaneously across CPU cores (ithreads)
- **non-blocking process:** process runs multiple execution paths that are executed serially (but interwoven) on a single CPU core

More Definitions

- **blocking call**: a function that waits for an answer before continuing execution.
- **non-blocking call**: a function that starts some operation (e.g., send a message to a server) and then returns immediately continuing execution. The response from the call is handled via a callback passed to the function.

Blocking vs Non-blocking flow





What does it look like?

```perl
#!/usr/bin/perl -w

use strict;

use AnyEvent;
use AnyEvent::HTTP;

my $cv = AnyEvent->condvar;

http_get "http://forecast.weather.gov/MapClick.php?lat=42.12100&lon=-77.94750&FcstType=dwml", sub {
    my( $data, $headers ) = @_;

    print "Got HTTP response: " . $headers->{Status} . "\n";

    $cv->send;
};

print "Request sent\n";

$cv->wait;

print "Data received\n";
```

How does it really execute?

- True execution is serial
- Non-blocking calls allow execution to continue while the call completes
- All pending tasks are tracked and resumed when the response callback is ready to be processed
- Code paths cannot be interrupted until they enter the event loop , so precise scheduling is not possible

PERCONA LIVE

Advantages and Disadvantages

- Eliminates busy loops with ‘sleep’, write event handlers instead
- Brings scheduling inside of user-space
- Less/No locking needed compared with a threaded process
- Can fully utilize a CPU core and no more
- Less available libraries (i.e., Perl)

Node.js and Perl Examples

Timers -- AnyEvent

```
#!/usr/bin/perl -w

use strict;

use AnyEvent;

my $cv = AnyEvent->condvar;

my $t = AnyEvent->timer( after => 5, cb => sub {
    print "5 seconds passed\n";
    $cv->send;
});

my $t2 = AnyEvent->timer( after => 1, interval => 1, cb => sub {
    print "1 second passed\n";
});

$cv->wait;
```

Timers - Node.js

```
var intervalID = setInterval( function() {
    console.log( "1 second passed");
}, 1000 );

setTimeout( function() {
    console.log( "5 seconds passed" );
    clearInterval( intervalID );
}, 5000 );
```

HTTP curl calls -- AnyEvent::HTTP

```
#!/usr/bin/perl -w

use strict;

use AnyEvent;
use AnyEvent::HTTP;

my $cv = AnyEvent->condvar;

http_get "http://forecast.weather.gov/MapClick.php?lat=42.12100&lon=-77.94750&FcstType=dwml", sub {
    my( $data, $headers ) = @_;

    print "Got HTTP response: " . $headers->{Status} . "\n";
    print "Data is " . length( $data ) . " bytes long\n";

    $cv->send;
};

print "Request sent\n";

$cv->wait;

print "Data received\n";
```

HTTP calls -- Node

```
var http = require('http');

http.get(
    {
        host:'forecast.weather.gov',
        path: '/MapClick.php?lat=42.12100&lon=-77.94750&FcstType=dwml'
    },
    function( res ) {
        var data;

        console.log( "Got HTTP response " + res.statusCode );

        res.on( 'data', function( chunk ) {
            data += chunk;
        });

        res.on( 'end', function() {
            console.log( "Data is " + data.length + " bytes long" );
            console.log('Data received');
        })
    }
);

console.log('Request sent');
```

AnyEvent::HTTPD
vs
Node.js

AIO -- Perl

Serve large files - Node.js

```
var http = require('http');
var fs = require('fs');

var i = 1;
http.createServer(function (request, response) {
    console.log('starting #' + i++);
    var stream = fs.createReadStream('data.bin', { bufferSize: 64 * 1024 });
    stream.pipe(response);
}).listen(8000);

console.log('Server running at http://127.0.0.1:8000/');
```

Database -- AnyEvent::DBI

```perl
#!/usr/bin/perl -w

use strict;

use AnyEvent;
use AnyEvent::DBI;

my $cv = AnyEvent->condvar;

my $dbh = new AnyEvent::DBI "DBI:mysql:dbname=test", 'root', '';

$dbh->exec ("select * from test where id=?", 10, sub {
    my ($dbh, $rows, $rv) = @_;

    $#_ or die "failure: $@";

    my $arr = $rows->[0];
    print "(";
    print join( ', ', @$arr );
    print ")\n";

    $cv->send;
});

$cv->wait;
```

Database - Node

```
var mysql = require( 'mysql' );

var client = mysql.createClient( {
   user: 'root',
   database: 'test'
});

client.query(
  'select * from test where id = ?', [ 10 ],
  function ( err, results, fields ) {
      if( err ) {
          throw err;
      }
      console.log( results );
      client.end();
  }
);
```

Interesting AnyEvent modules

- Memcached
- Curl::Multi
- AIO
- Redis
- Handle
- Filesys::Notify
- Gearman
- XMLRPC
- DNS
- MongoDB
- Pg
- IRC
- BDB
- Run
- JSONRPC
- Twitter
- Worker
- CouchDB
- Riak
- RabbitMQ
- Kanye

PERCONA LIVE

The tradeoff

Issues with Non-blocking Libraries

- Less common and unsupported by central development teams in general
- Concurrency tends to be limited -- limitations of the client protocol
- Transactional support tends to not "just work".

Debugging Non-blocking Code

- Hard-to-reproduce race condition crashes
- Traditional "stack trace" debugging much harder to do
  - Some non-blocking frameworks offer debugging tools (e.g., Perl's "Coro")
- Context variables may be modified/undefined/etc. between the initial call and the callback.
  - This is what tripped me up the most.

Example Crash

```
my $temporary_object = Obj->new();

$object->do_query_nb( "SELECT * FROM my_table", callback => sub {
  my( $result ) = @_;
  $temporary_object->print( $result );
  # ^^ We get a crash on an undefined
  # object here, why?
} );

undef $temporary_object;
```

Code Organization

- Code gets nested deeply very quickly
- A set of non-blocking tasks that each depend on the previous to finish can be tedious to code.
- Some helpers exist to work-around this:
  - Node.js ‘async’ library
  - Perl - Coro and rouse_cb/rouse_wait
- Otherwise, think carefully about your Object structure and build helper functions to help

Coro rouse_cb example

```
#!/usr/bin/perl -w

use strict;

use Coro;

use AnyEvent;
use AnyEvent::DBI;

my $cv = AnyEvent->condvar;

my $dbh = new AnyEvent::DBI "DBI:mysql:dbname=test", 'root', '';

# non-blocking call to the DB, defer handling response until later.
$dbh->exec( "select * from test where id=?", 10, Coro::rouse_cb );

# do something else here

# now block this execution path until the query results come back.
my ($dbh2, $rows, $rv) = Coro::rouse_wait;

$#_ or die "failure: $@";

my $arr = $rows->[0];
print "(";
print join( ', ', @$arr );
print ")\n";
```

Node.js ‘async’ library example

```
var async = require( 'async' );
//...

// Start a "transaction"
async.waterfall([
    // statement 1
    function( callback ) {
        client.query( 'select * from test where id = ?', [ 10 ], callback );
    },
    // // statement 2, gets all arguments passed to last to last callback
    function( last_results, last_fields, callback ) {
        console.log( last_results );
        client.query( 'select * from test where id = ?', [ 5 ], callback );
    }
],

// This function gets called if any statement produces an error, or all statements complete successfully
function( err, result ) {
    if( err ) {
        throw err;
    } else {
        console.log( "Done successfully" );
        console.log( result );
        client.end();
    }
});
```

Conclusion

- Non-blocking programming made coding fun for me again
- See all my code examples in a working VM:
  - https://github.com/jayjanssen/non-blocking-code-examples