Erlang Bootcamp 2

Erlang Bootcamp 2:

Agenda

- (Re)Introducing gen_server
- Supervisors
- OTP Applications
- rebar

Agenda

- TCP Sockets
- Distributed Erlang
- Webmachine

Introducing
`gen_server`

Introduction

- Abstracts away all low-level server details
- Integrates with the rest of OTP
- Makes writing servers a snap!

What is a behavior?

- Defines an API
- Contract between modules
- Provide support functions (optional)

gen_server

- Implemented as a behavior
- Defines an API for server logic module
- Provides server logic support functions

Startup
&
Shutdown

init/ 1

- Called during startup
- Responsible for generating initial server state
- Returns {ok, State} or {stop, Reason}

```
init([]) ->
    {ok, #state{}}.
```

terminate/2

- Called during teardown
- Responsible for cleaning up resources
- Return value is ignored

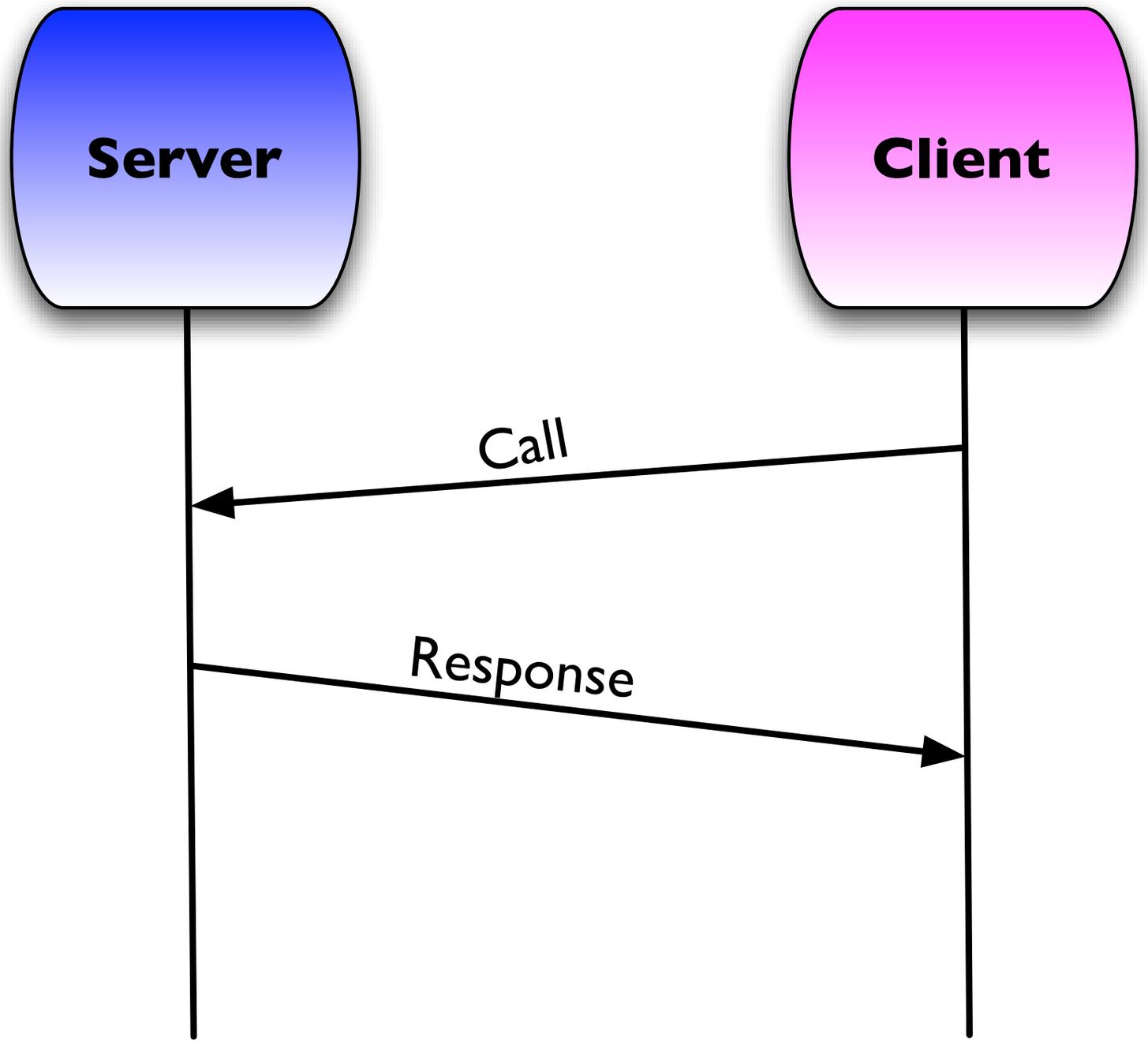
```
terminate(_Reason, _State) ->
    ok.
```

Message Types

‘Call’ Messages

- Client is blocked until server replies
- Default timeout interval is 5 seconds
- Clients use gen_server:call/3,4

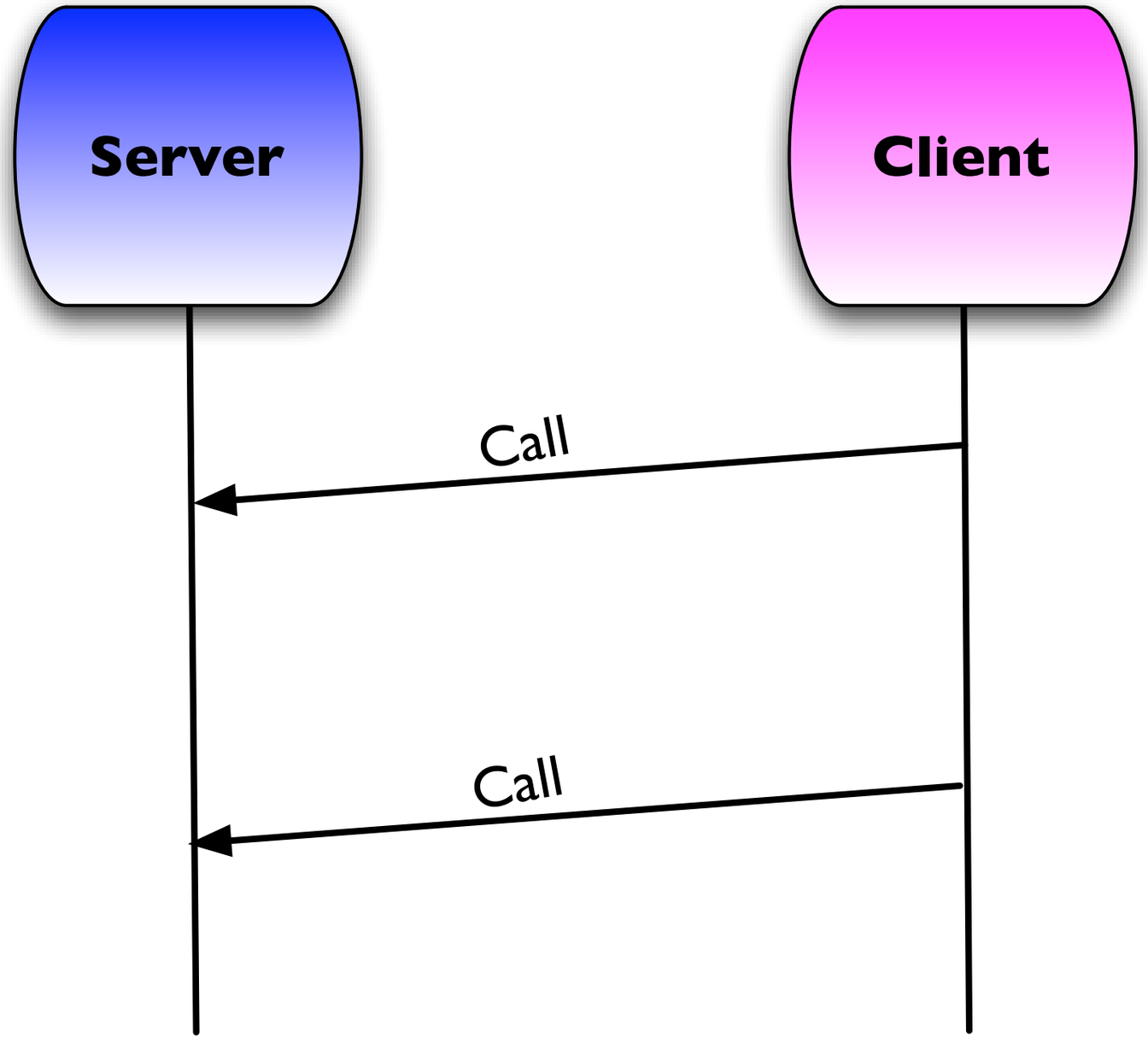
‘Call’ Messages



‘Cast’ Messages

- Asynchronous from the client’s POV
- Server isn’t required to reply
- Clients use gen_server:cast/2

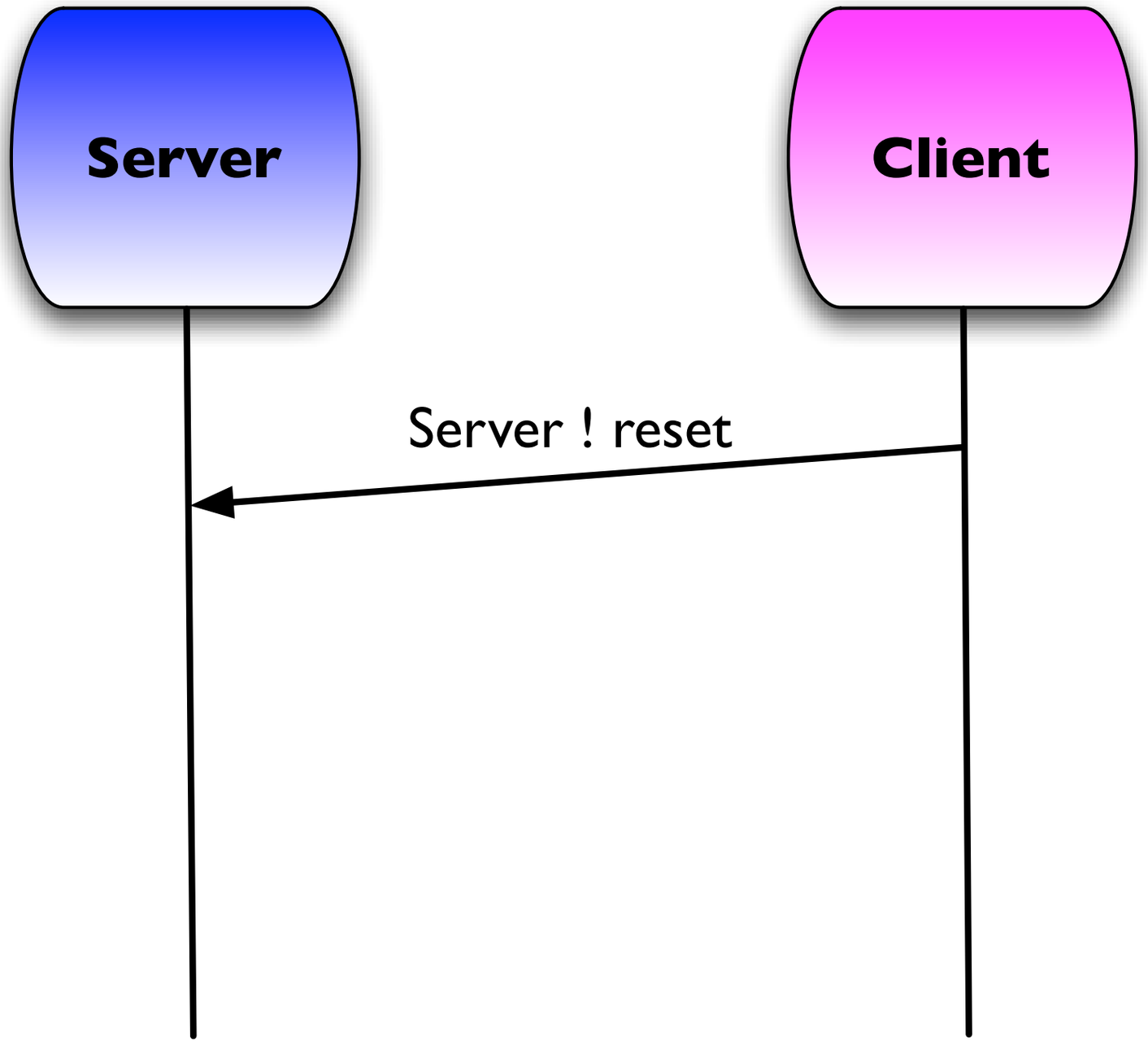
‘Cast’ Messages



‘Info’ Messages

- Asynchronous
- Server isn’t required to reply
- Delivered via “raw” message send

‘Info’ Messages



Handling Messages

‘Call’ Messages

```
handle_call(current_time, _From, State) ->
    {reply,
     calendar:now_to_local_time(erlang:now()),
     State};
```

‘Cast’ Messages

```
handle_cast(foo, State) ->
    do_something(foo),
    {noreply, State};
```

‘Info’ Messages

```
handle_info(reset, State) ->
  {noreply,reset(State)};
```

Lab Time!

- Implement an in-memory cache using gen_server. Cached values should be stored in an Erlang dictionary.
- Cached values should be stored in a tuple containing the value and a timestamp. Values older than X seconds should be deleted.
- The gen_server should expose 4 client functions: get/1, put/2, del/1, count/0.

Supervisors

What are supervisors?

- Processes which monitor other processes
- Supervisors can supervise other supervisors
- Very configurable (good and bad)

How Do They Work?

- Each child process is linked to its supervisor
- Supervisors restart crashed children

Configuring Children

{RestartPolicy, Children}

Restart Policy

- Controls how a crashed child is handled
- one_for_one, one_for_all, rest_for_one
- Applied to all children

Restart Interval

- Describes the "fatal crash scenario"
- Prevents supervisor from restarting children forever

```
{one_for_one, 10, 60}
```

Child Specification

- How to start the child
- How to manage the child
- Which code belongs to the child

{Id, StartFun, Restart, Shutdown, Type, Modules}

Start Function

- Child entry point
- MFA-formatted tuple: {Module, Fun, Args}
- Starts and links the child to the supervisor

```
{time_server,
  {time_server, start_link, []}
```

Restart

- Determines when the restart policy is used
  - Permanent: always restarted
  - Temporary: never restarted
  - Transient: restarted when it crashes

```
{time_server,
   {time_server, start_link, []},
   permanent
```

Shutdown

- How to stop the child
  - brutal_kill: shoot it in the head (kill -9)
  - <integer>: wait for X milliseconds before brutal_kill-ing
  - infinity: wait forever

```
{time_server,
   {time_server, start_link, []},
   permanent, 2000
```

Type

- worker: Normal child process
- supervisor: Nested supervisor. Should use ‘infinity’ shutdown

```
{time_server,
   {time_server, start_link, []},
   permanent, 2000, worker
```

Modules

- Modules which make up the child process
- Normally a single element list

```
{time_server,
  {time_server, start_link, []},
  permanent, 2000, worker,
[time_server]}
```

Lab Time!

- Write a supervisor for the cache server. Make sure the cache server starts successfully after you call the supervisor's start_link().

Working with OTP Applications

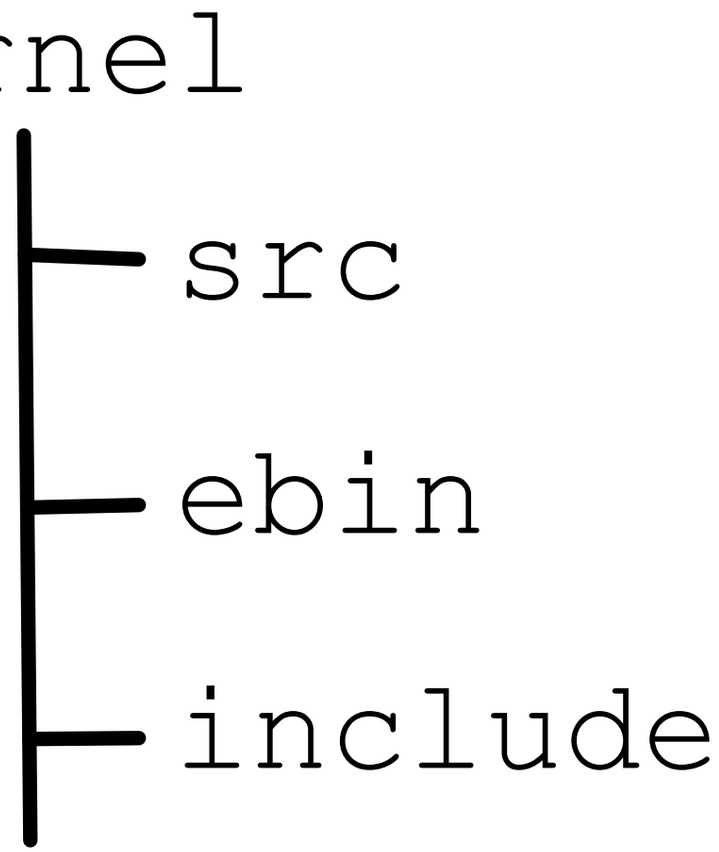
What is an OTP Application?

- Recommended format for organizing projects
- Compatible with the OTP libraries and the Erlang runtime
- Compatible with tools like rebar, proper, and dialyzer

OTP App Structure

- Top-level directory named after the app
- Erlang source kept in ‘src’
- Compiled `.beam` files kept in ‘ebin’
- Header files kept in ‘include’
- App descriptor (`.app`) stored in ‘ebin’

OTP App Structure



Application Descriptor

- Plain text file named after the application
- Describes app metadata:
  - Entry point
  - Version
  - Included modules
  - Registered names
- Ends in '`.app`'

.app Example

```
{application, erlz,
 [
  {description, "Wrapper for the fastlz compression library"},
  {vsn, "0.1"},
  {registered, []},
  {applications, [kernel,
                  stdlib]},
  {env, []}
 ]}.
```

Minimum .app Requirements

- Application
- Version
- Applications list
- Modules list

.app Example

```
{application, webmachine,
 [{description, "webmachine"},
  {vsn, "1.8.0"},
  {modules, [webmachine,
    webmachine_app,
    webmachine_decision_core,
    [....]
]},
{registered, []},
{mod, {webmachine_app, []}},
{env, [
       {dispatch_list, []}
      ]},
{applications, [kernel, stdlib, crypto, ssl]}]}.
```

.app Example

```
{application, webmachine,
 [{description, "webmachine"},
  {vsn, "1.8.0"},
  {modules, [webmachine,
    webmachine_app,
    webmachine_decision_core,
    [....]
]},
{registered, []},
{mod, {webmachine_app, []}},
{env, [
       {dispatch_list, []}
      ]},
{applications, [kernel, stdlib, crypto, ssl]}]}.
```

.app Example

```
{application, webmachine,
 [{description, "webmachine"},
  {vsn, "1.8.0"},
  {modules, [webmachine,
    webmachine_app,
    webmachine_decision_core,
    [....]
]},
{registered, []},
{mod, {webmachine_app, []}},
 {env, [
         {dispatch_list, []}
       ]},
{applications, [kernel, stdlib, crypto, ssl]}]}.
```

`Application` Behavior

- Abstracts the 'entry point' concept
- Responsible for starting the supervision tree
- Two functions: `start/2` **and** `stop/1`

```
-module(foo_app).

-behaviour(application).

%% Application callbacks
-export([start/2, stop/1]).
```

```
start(_StartType, _StartArgs) ->
    foo_sup:start_link().

stop(_State) ->
    ok.
```

Pains of Building an OTP App

- Maintaining the modules list
- Building source
  - Include paths
  - Dependencies
- Automating builds

rebar

What is rebar?

- Project generator
- Build tool
- Dependency manager

rebar as

Project Generator

Project Generator

`rebar create-app appid=`*`app_name`*

- Generates OTP dirs (ebin, src)
- Creates top-level supervisor and app modules
- Creates `app_name.app`

rebar as Build Tool

Supported File Types

- Erlang source (`.erl`)
- Protocol Buffers (`.proto`)
- Linked-in drivers (`.c,.cpp,Makefiles`)
- NIFs

Supported File Types

- Leex/Yecc grammars (`.xrl`, `.yrl`)
- Eunit (tests & fixtures)
- Common Test
- QuickCheck

Controlling The Build Process

- `rebar.config` **controls all**
- Kept in top of application directory

Example
rebar.config

```
{erl_opts, [debug_info, warnings_as_failures]}.

{deps, [{erlzmq2, ".*",
         {git, "git://github.com/zeromq/erlzmq2.git",
          "HEAD"}}]}.
```

Example
rebar.config

```
{erl_opts, [debug_info,
            warnings_as_failures]}.

{deps, [{erlzmq2, ".*",
         {git, "git://github.com/zeromq/erlzmq2.git",
          "HEAD"}}]}.
```

Compiling Erlang

- Sanity checks `.app` file on each compile
- Use `erl_opts` to configure compiler
- Automagically populate modules list (!!!)

rebar: `rebar compile`

rebar as Dependency Manager

rebar Dependencies

- **Not** a package manager
- Dependencies on 3rd party *Erlang source* projects
- Supports `svn`, `git`, and `hg`

Declaring Dependencies

App Name

App Version

```
{deps, [{erlzmq2, ".*",
         {git, "git://github.com/zeromq/erlzmq2.git",
          "HEAD"}}]}.
```

VCS Type

VCS Version/Branch/Tag

VCS URL

Using Dependencies

- Stored in deps/
- Fetched when needed
- Automatically added to the build-time codepath

rebar: `rebar [check|get|delete]-deps`

Lab Time!

- Package the cache server as an OTP application:
  - .app file
  - application behavior
- Build the project using rebar
- Verify the cache server starts after calling 'application:start(cache_server).'

Network Programming with TCP Sockets

TCP Sockets

- Two categories: clients & servers
- Clients connect to servers
- Servers accept incoming connections
- Both use `gen_tcp` and `inet` modules to interact with sockets

gen_tcp

- Interface for working with TCP sockets
- Used by clients and servers
- Works with IPv4 and IPv6

`gen_tcp` Clients

- Use `connect/3,4` to open connection
- Use `send/2` and `recv/2,3` to communicate

```
{ok, Sock} = gen_tcp:connect(“localhost”,
                             5432,
                             [binary,
                              {active, false}]).

gen_tcp:send(Sock, <<“Hello”>>).
```

```
{ok, Sock} = gen_tcp:connect(“localhost”,
                             5432,
                             [binary,
                              {active, false}]).

gen_tcp:send(Sock, <<“Hello”>>).
```

Socket Options

- Initial values set during socket creation
- Additional values set via `inet:setopts/2`

Socket Options

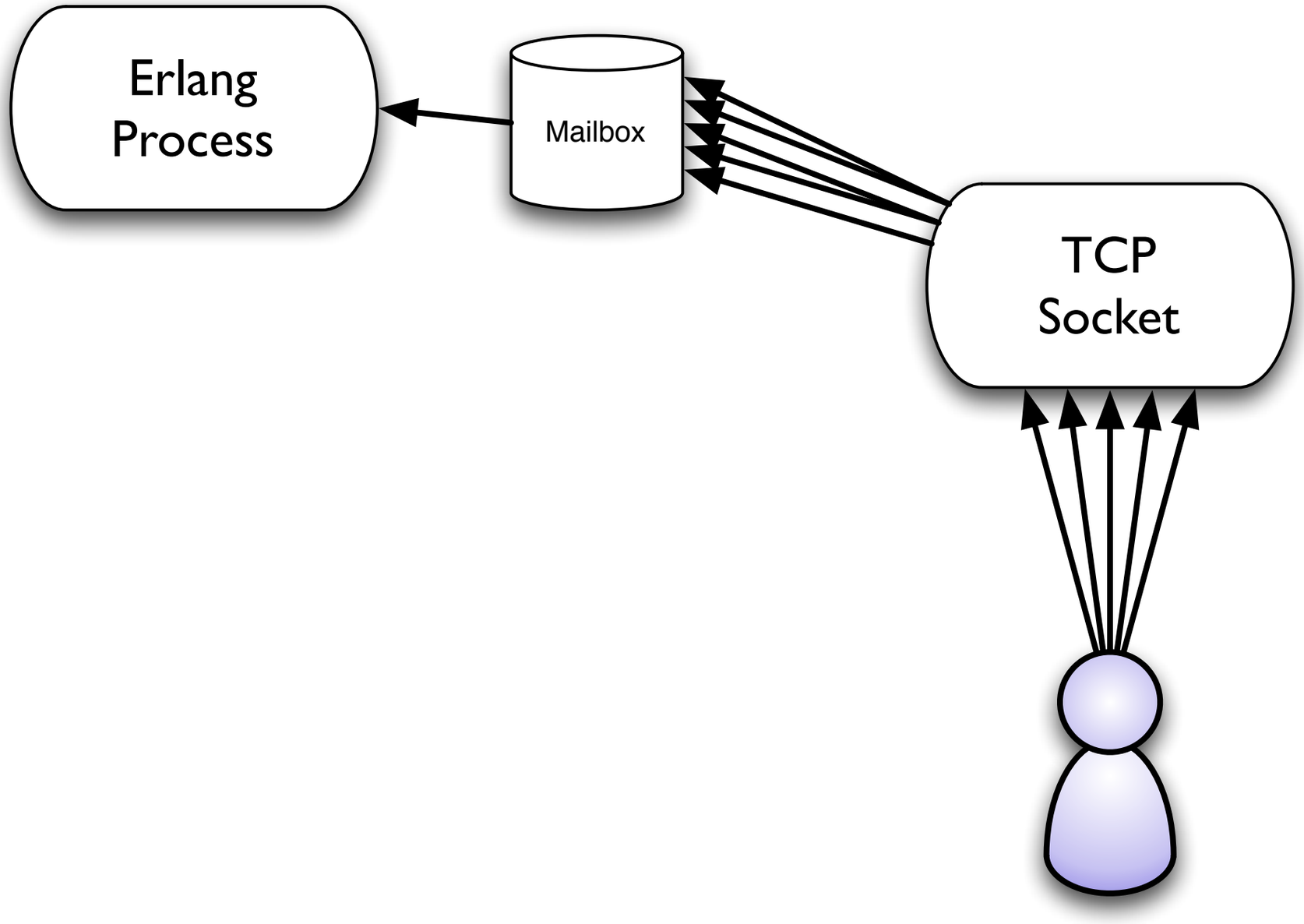
Option	Description
`{active, [true|false|once]}`	Flow control
`{packet, [raw|*N*|line|binary|http|...]}`	Enables data parsing
`{recbuf, *N*}` `{sndbuf, *N*}`	Sets the size of the receive and send buffers

Flow Control

- Active - Erlang sends data as it's **received**

(Fast but prone to crashes)

{active, true}



Flow Control

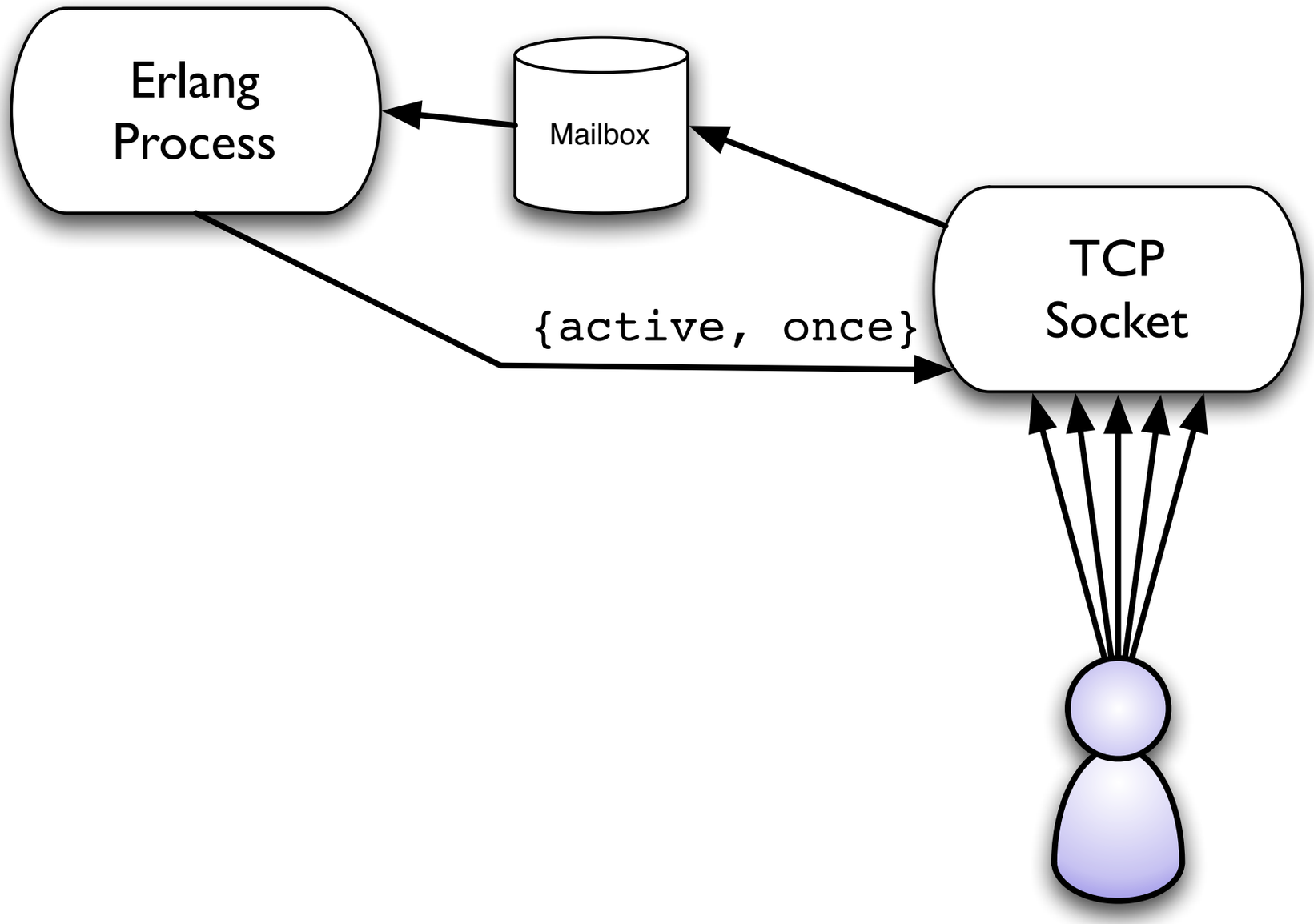
- Active - Erlang sends data as it's **received**

(Fast but prone to crashes)

- Active Once - Erlang sends data as it's **acked**

(Slower but more stable)

{active, once}



Server Sockets

- Use `gen_tcp:listen/2` to open server socket
- Use `gen_tcp:accept/1,2` in a tail recursive process
- Spawn a new process for each connection

Why gen_nb_server?

- Tired of writing accept loops
- Other libs were too heavy
- Push accept down into `prim_inet`

gen_nb_server

- gen_server + sockets = fun
- Listen on multiple IPs and ports
- Add/remove IPs and ports at runtime

Using gen_nb_server

- Implement all the gen_server callbacks
- `init/2` **instead of** `init/1`
- `sock_opts/0`
- `new_connection/4`

Socket Handling Tips

- Use active mode sockets whenever possible
- Use {active, once} to protect against flooding
- Use appropriate packet type to pre-parse incoming data

Supervisors As ‘Process Factories’

Aren't Factories For Java?

NO.

Supervision Is A Good Idea

- Single point of creation
- Easier to trace execution path
- Supervision tree is easier to understand
- Makes debugging easier

simple_one_for_one

- Supervisor for a single process type (module)
- All children are started dynamically
- Crashed children are **not** restarted

```
-behaviour(supervisor).

%% API
-export([start_link/0,
         new_handler/1]).

%% Supervisor callbacks
-export([init/1]).

-define(SERVER, ?MODULE).

start_link() ->
    supervisor:start_link({local, ?SERVER}, ?MODULE, []).

new_handler(Conn) ->
    supervisor:start_child(?SERVER, [Conn]).

init([]) ->
    {ok, {{simple_one_for_one, 0, 1},
          [{conn_handler, {conn_handler, start_link, []}},
           temporary, brutal_kill, worker, [conn_handler]]}}.
```

```
-behaviour(supervisor).

%% API
-export([start_link/0,
         new_handler/1]).

%% Supervisor callbacks
-export([init/1]).

-define(SERVER, ?MODULE).

start_link() ->
    supervisor:start_link({local, ?SERVER}, ?MODULE, []).

new_handler(Conn) ->
    supervisor:start_child(?SERVER, [Conn]).

init([]) ->
    {ok, {{simple_one_for_one, 0, 1},
          [{conn_handler, {conn_handler, start_link, []}},
           temporary, brutal_kill, worker,
           [conn_handler]]}}.
```

```
-behaviour(supervisor).

%% API
-export([start_link/0,
         new_handler/1]).

%% Supervisor callbacks
-export([init/1]).

-define(SERVER, ?MODULE).

start_link() ->
    supervisor:start_link({local, ?SERVER}, ?MODULE, []).

new_handler(Conn) ->
    supervisor:start_child(?SERVER, [Conn]).

init([]) ->
    {ok, {{simple_one_for_one, 0, 1},
          [{conn_handler, {conn_handler, start_link, []}},
           temporary, brutal_kill, worker, [conn_handler]]}}.
```

```
-behaviour(supervisor).

%% API
-export([start_link/0,
         new_handler/1]).

%% Supervisor callbacks
-export([init/1]).

-define(SERVER, ?MODULE).

start_link() ->
    supervisor:start_link({local, ?SERVER}, ?MODULE, []).

new_handler(Conn) ->
    supervisor:start_child(?SERVER, [Conn]).

init([]) ->
    {ok, {{simple_one_for_one, 0, 1},
          [{conn_handler, {conn_handler, start_link, []}},
           temporary, brutal_kill, worker, [conn_handler]]}}.
```

Lab Time!

- Write a socket front-end to the cache server using gen_nb_server.
- Each connection will be handled by a separate Erlang process. This means you will need to create a new process when `new_connection/4` is called.
- Clients will use a simple line-based protocol to send commands to the server:

  ```
  get <key> \r\n
  del <key> \r\n
  put <key> <value> \r\n
  ```

- Servers responses can be one of three types:

  `+ok\r\n` (no payload)

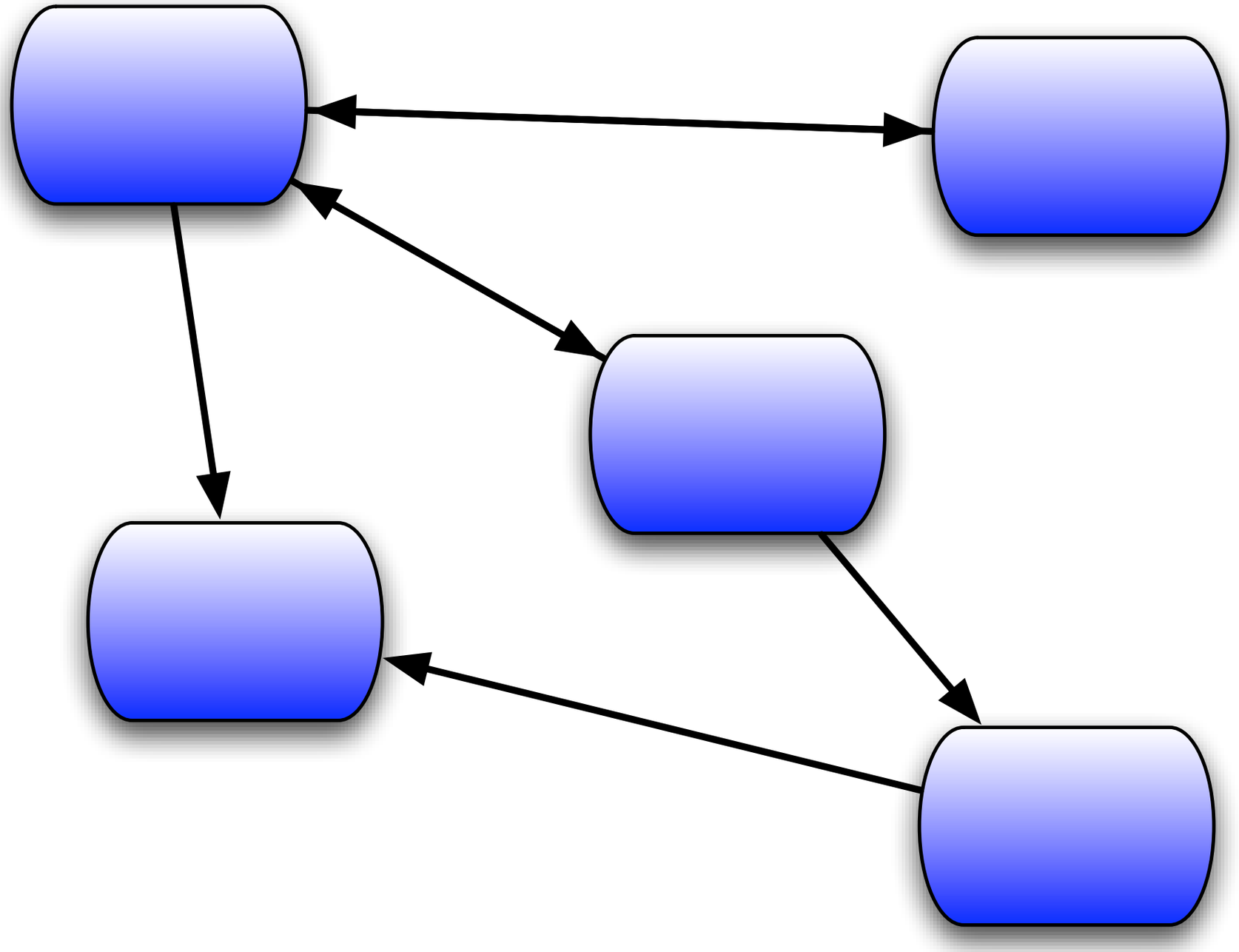
  `+ok <value>\r\n` (with payload)

  `-err\r\n` (error or not found)

  Hint: *Socket options* `[list, {packet, line}, {active, once}]` *and the* `string` *module will be very helpful.*

Distributed Erlang

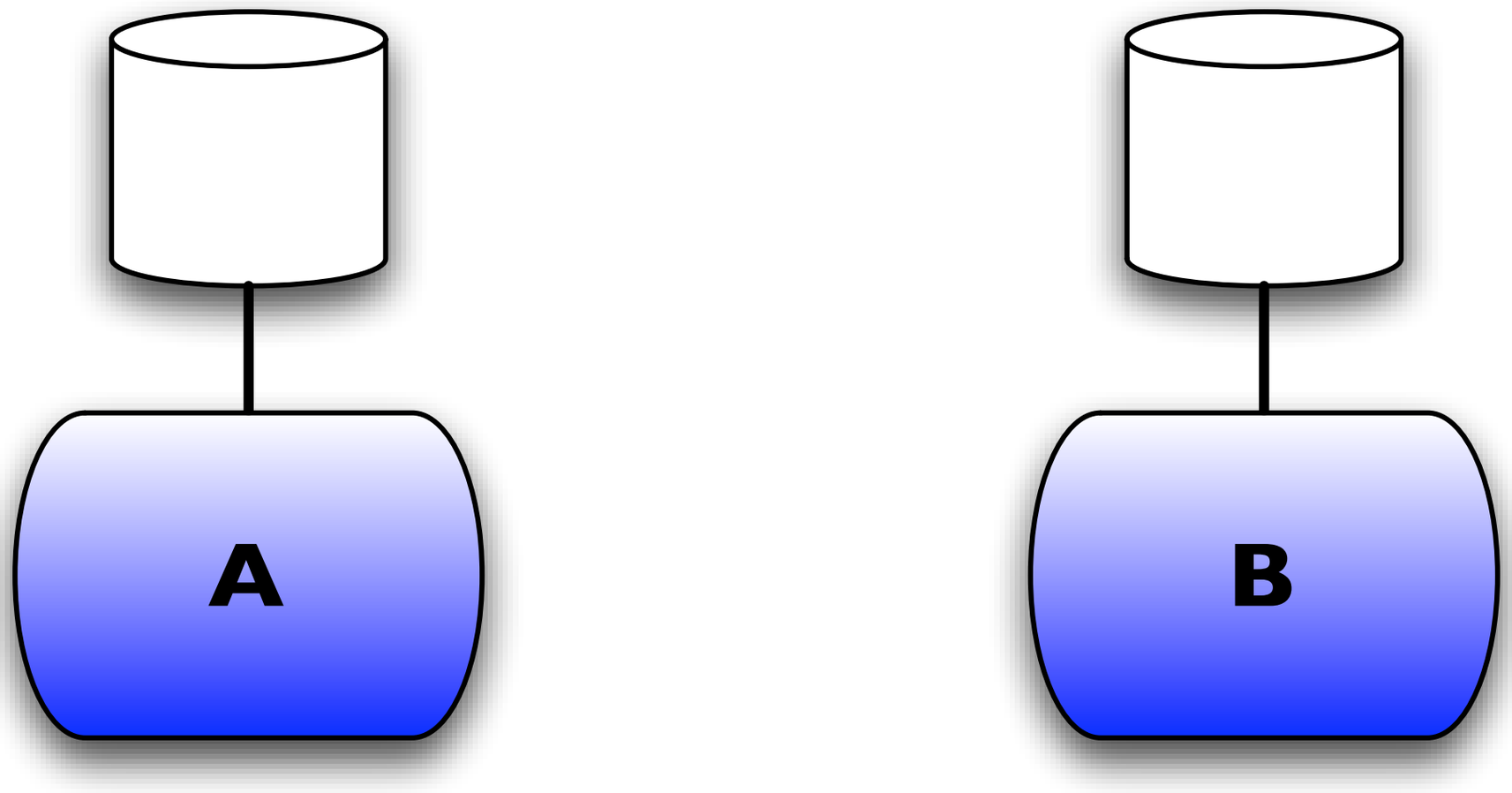
Distributed Erlang



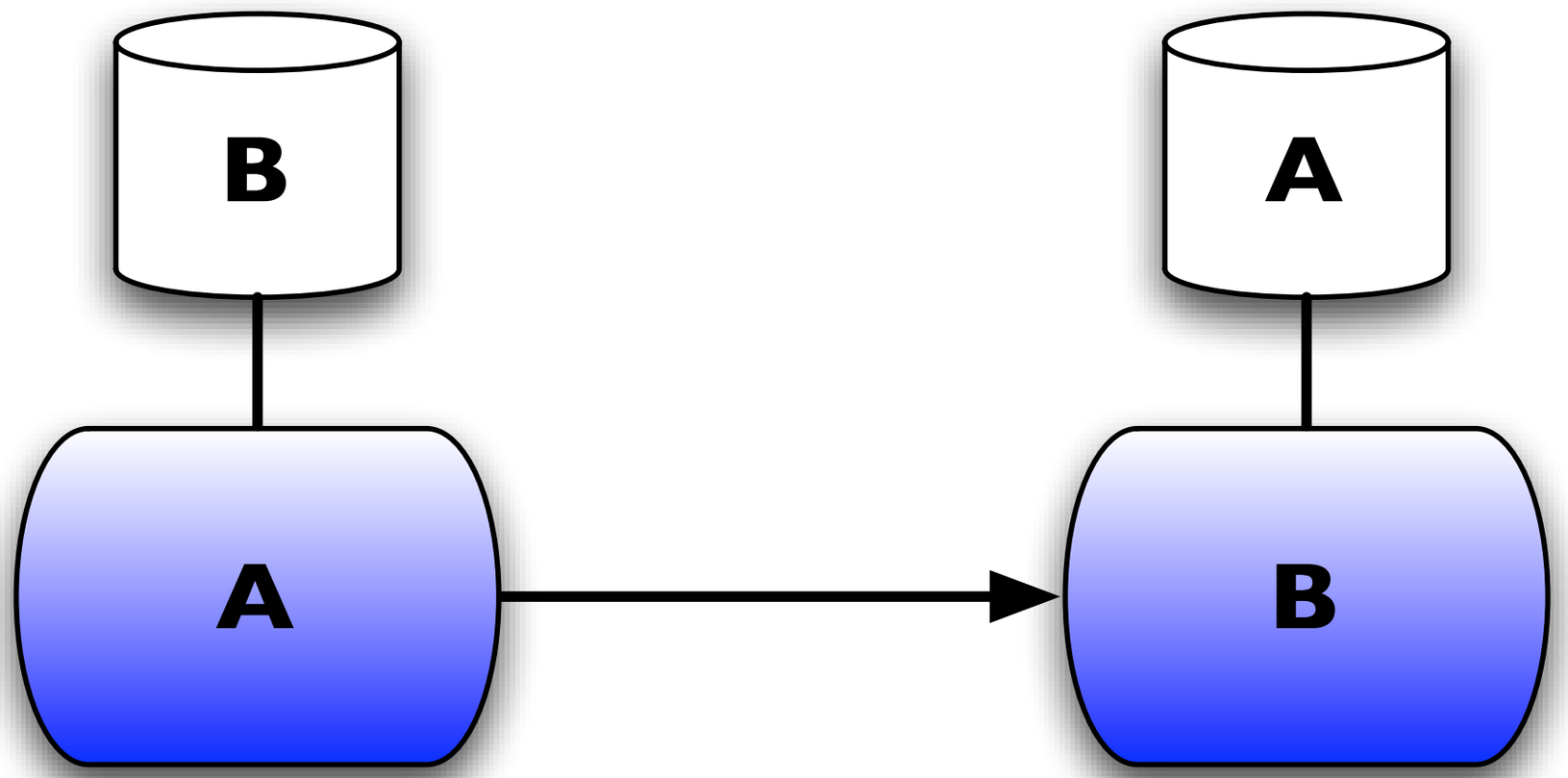
Distributed Erlang

- Each node maintains its own node list
- Nodes discover each other via `net_adm:ping/1` **or** `net_kernel:connect/1`
- New nodes "inherit" the node list of the node they ping

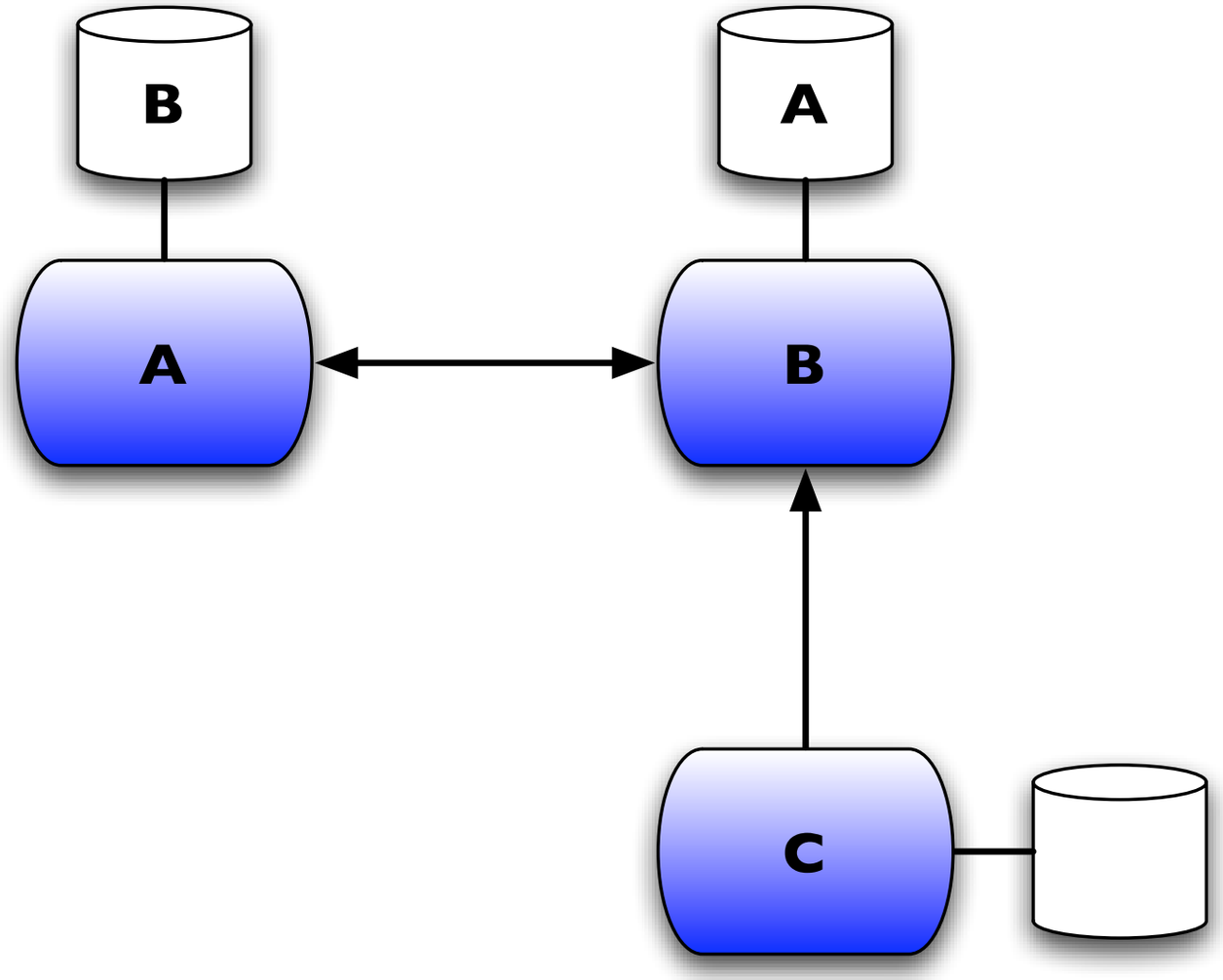
Node Discovery



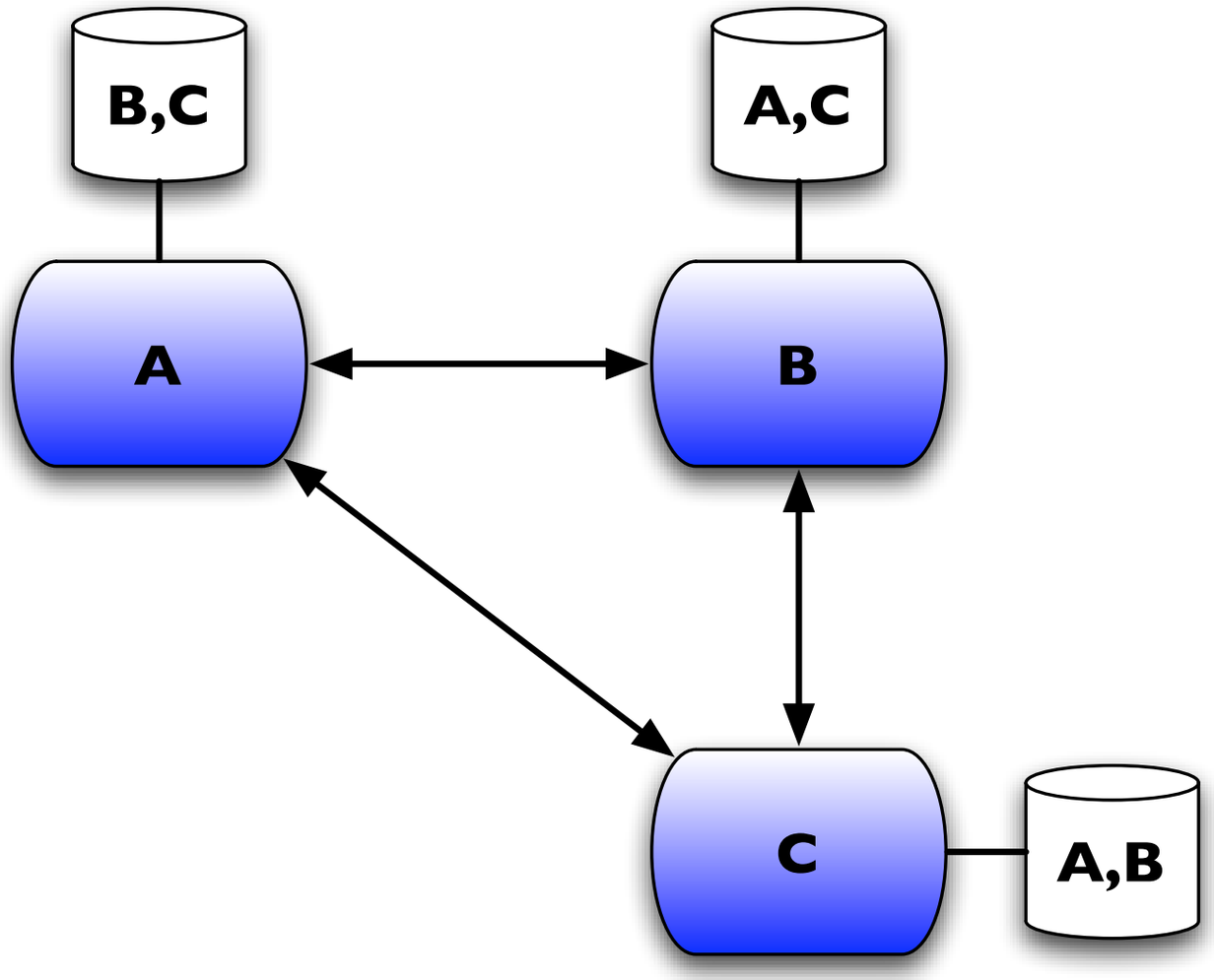
Node Discovery



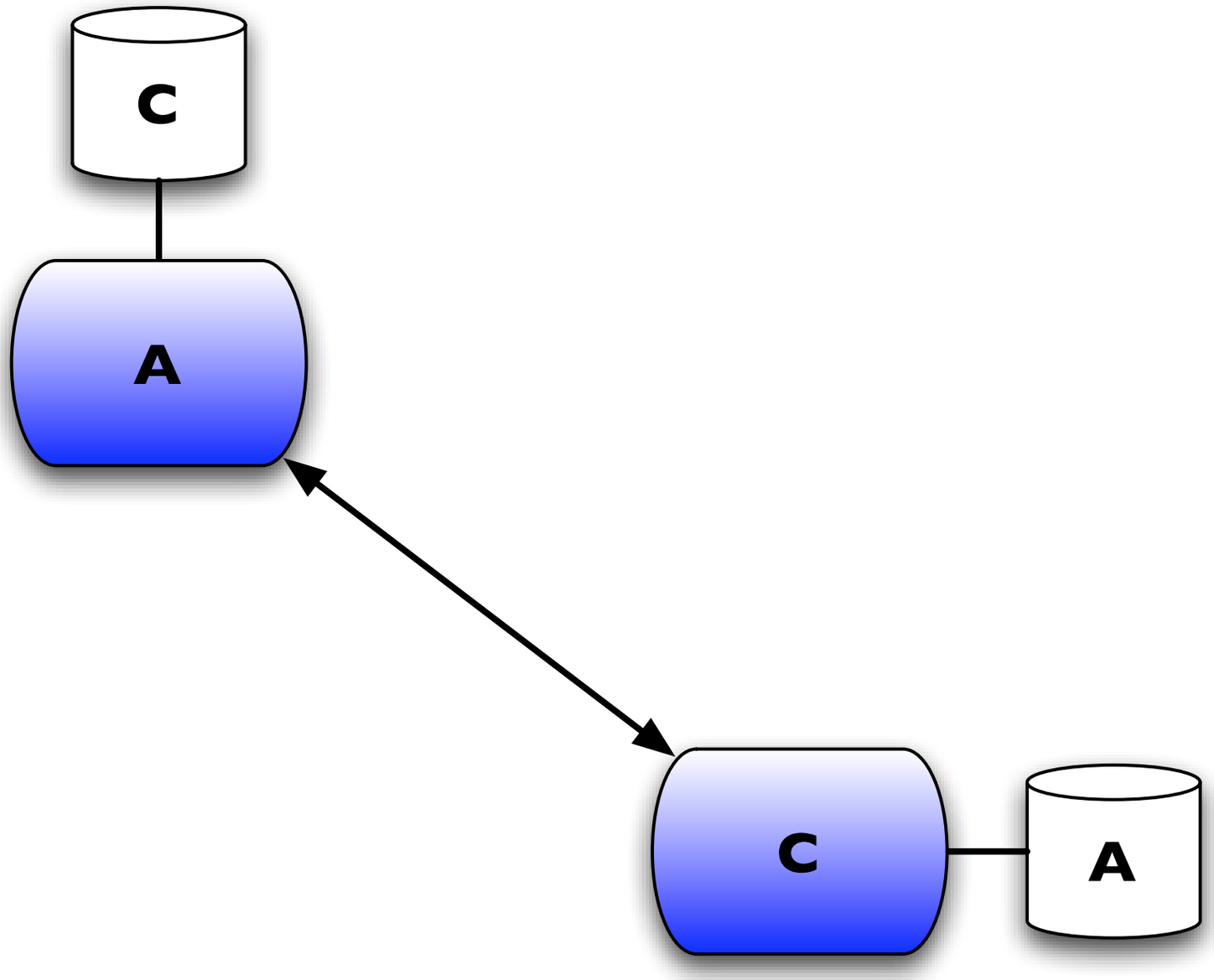
Node Discovery



Node Discovery



Dropping Nodes



Fully Meshed Networks Suck

Nodes	Connections
4	6
16	120
32	496 (!!!)

Registered Processes

- Associates a human-friendly name to a pid
- Default scope is local to the hosting node
- Use global name registry to register names across nodes

Note: Global names can be a bottleneck

Process Groups

- Named groups of processes across nodes
- Implement distributed process pools
- Use pg2 instead of pg

Note: process groups can cause problems in clusters experiencing membership churn.

Useful pg2 Functions

- `pg2:create/1`
- `pg2:join/2`
- `pg2:leave/1`
- `pg2:get_members/1`,
  `pg2:get_closest_member/1`

Lab Time!

Goal: Create a new server which will round-robin between cache servers running on different nodes. All adds and deletes must be replicated to all cache servers.

The load balancing server will:

- Use gen_server
- Provide the same functional interface but delegate all work to the backend cache servers
- Register itself globally
- Provide a function backend servers can use to register themselves
- Use process monitors to remove dead backend servers

The backend cache servers will:

- Register themselves with the load balancer when they start. It is OK to hardcode the name.

webmachine

What is webmachine?

~~Framework~~

Toolkit

A toolkit for building RESTful HTTP resources

```
-module(hello_world_resource).
-export([init/1, to_html/2]).

-include_lib("webmachine/include/webmachine.hrl").

init([]) ->
  {ok, "Hello, world"}.

to_html(Req, State) ->
  Output = io_lib:format("<html><body>~s</body></html>",
                         [State]),
  {Output, Req, State}.
```

```
-module(hello_world_resource).
-export([init/1, to_html/2]).

-include_lib("webmachine/include/webmachine.hrl").

init([]) ->
  {ok, "Hello, world"}.

to_html(Req, State) ->
  Output = io_lib:format("<html><body>~s</body></html>",
                         [State]),
  {Output, Req, State}.
```

```
-module(hello_world_resource).
-export([init/1, to_html/2]).

-include_lib("webmachine/include/webmachine.hrl").

init([]) ->
  {ok, "Hello, world"}.

to_html(Req, State) ->
  Output = io_lib:format("<html><body>~s</body></html>",
                         [State]),
  {Output, Req, State}.
```

```
-module(hello_world_resource).
-export([init/1, to_html/2]).

-include_lib("webmachine/include/webmachine.hrl").

init([]) ->
  {ok, "Hello, world"}.

to_html(Req, State) ->
  Output = io_lib:format("<html><body>~s</body></html>",
                         [State]),
  {Output, Req, State}.
```

```
-module(hello_world_resource).
-export([init/1, to_html/2]).

-include_lib("webmachine/include/webmachine.hrl").

init([]) ->
  {ok, "Hello, world"}.

to_html(Req, State) ->
  Output = io_lib:format("<html><body>~s</body></html>",
                         [State]),
  {Output, Req, State}.
```

```
GET /test HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0
Accept: text/html;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: UTF-8,*
Keep-Alive: 300
Connection: keep-alive

HTTP/1.x 200 OK
Server: MochiWeb/1.1 WebMachine/1.3
Date: Mon, 22 Jun 2009 02:27:46 GMT
Content-Type: text/html
Content-Length: 78
```

```
-module(hello_world_resource).
-export([init/1, to_html/2]).
-export([generate_etag/2]).

-include_lib("webmachine/include/webmachine.hrl").

init([]) ->
  {ok, "Hello, world"}.

to_html(Req, State) ->
  Output = io_lib:format("<html><body>~s</body></html>",
                         [State]),
  {Output, Req, State}.

generate_etag(Req, State) ->
  {mochihex:to_hex(crypto:md5(State)), Req, State}.
```

```
GET /test HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0
Accept: text/html;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: UTF-8,*
Keep-Alive: 300
Connection: keep-alive

HTTP/1.x 200 OK
Server: MochiWeb/1.1 WebMachine/1.3
Etag: bc6e6f16b8a077ef5fbc8d59d0b931b9
Date: Mon, 22 Jun 2009 02:29:46 GMT
Content-Type: text/html
Content-Length: 78
```

```
GET /test HTTP/1.1
Host: localhost:8000
.
.
.
If-None-Match: bc6e6f16b8a077ef5fbc8d59d0b931b9
```

```
HTTP/1.x 304 Not Modified
Server: MochiWeb/1.1 WebMachine/1.3
Etag: bc6e6f16b8a077ef5fbc8d59d0b931b9
Date: Mon, 22 Jun 2009 02:30:00 GMT
```

```
-module(hello_world_resource).
-export([init/1, to_html/2]).
-export([generate_etag/2, encodings_provided/2]).

-include_lib("webmachine/include/webmachine.hrl").

init([]) ->
  {ok, "Hello, world"}.

to_html(Req, State) ->
  Output = io_lib:format("<html><body>~s</body></html>",
                         [State]),
  {Out, Req, State}.

generate_etag(Req, State) ->
  {mochihex:to_hex(crypto:md5(State)), Req, State}.

encodings_provided(Req, State) ->
  {[{"gzip", fun(X) -> zlib:gzip(X) end}], Req, State}.
```

```
GET /test HTTP/1.1
Host: localhost:8000
.
.
.
Accept-Encoding: gzip,deflate

HTTP/1.x 200 OK
Server: MochiWeb/1.1 WebMachine/1.3
Etag: bc6e6f16b8a077ef5fbc8d59d0b931b9
Date: Mon, 22 Jun 2009 02:46:57 GMT
Content-Type: text/html
Content-Length: 71
Content-Encoding: gzip
```

Request Routing

http://foo.com/items

```
{["items"], grocery_item_resource, []}.
```

URL path

Resource module

Init params

http://foo.com/items/chocolate

```
{["items", item], grocery_item_resource, []}.
```

GET

```
-module(hello_world_resource).
-export([init/1,to_html/2]).

-include_lib("webmachine/include/webmachine.hrl").

init([]) ->
  {ok, "Hello, world"}.

to_html(Req, State) ->
  Output = io_lib:format(“<html><body>~s</body></html>”,
                         [State]),
  {Output, Req, State}.
```

Must be idempotent

PUT

```
-module(grocery_list_resource).
-export([init/1,allowed_methods/2]).
-export([content_types_accepted/2,from_json/2]).

-include_lib("webmachine/include/webmachine.hrl").

init([]) ->
  {ok, []}.
```

```
allowed_methods(Req, State) ->
  {['PUT'], Req, State}.

content_types_accepted(Req, State) ->
  {[{"application/json", from_json}], Req, State}.

from_json(Req, State) ->
  %% Create/update logic here
```

DELETE

```
-module(grocery_list_resource).
-export([init/1,allowed_methods/2]).
-export([delete_resource/2]).

-include_lib("webmachine/include/webmachine.hrl").

init([]) ->
  {ok, []}.

allowed_methods(Req, State) ->
  {['DELETE'], Req, State}.

delete_resource(Req, State) ->
  %% Deletion logic here
```

POST

Hmmmm

Problems with POST

- Overloaded semantics
- Create or update?

Creation via POST

- `allowed_methods/2`
- `post_is_create/2`
- `create_path/2`
- `content_types_accepted/2`
- handler function

Update via POST

- `allowed_methods/2`
- `content_types_accepted/2`
- handler function

Lab Time!

- Write a webmachine application to provide a RESTful interface to the cache server.
- URLs should be of the form /cache/<key>
- GETs should fetch cache entries, DELETEs should delete cache entries, and PUTs should store cache entries.