EPIC

AGENDA

- What is EPIC?
- History
- Node Layout
- Process Layout

Wait!

You said Code!

What is...?

EPIC

Engine for Combat

Turn Based

2D Hex Space

Hand Crafted AI

Big Conflicts

2000+ UNITS

MANY CONFLICTS (SCALABLE)

Enough

Enough

ADVERTISING

I WAS LYING...

What is...?

What is...?

EPIC

Open Source

Some Nice Erlang Examples

Alpha Ware

Just Epic For Wordplay

Aimed To Conquer World

EPIC

history

Third

Implementation

Fault Tolerant

Great Scalability

Surprisingly Slow

In Erlang

EPIC

Node Layout

Node Layout

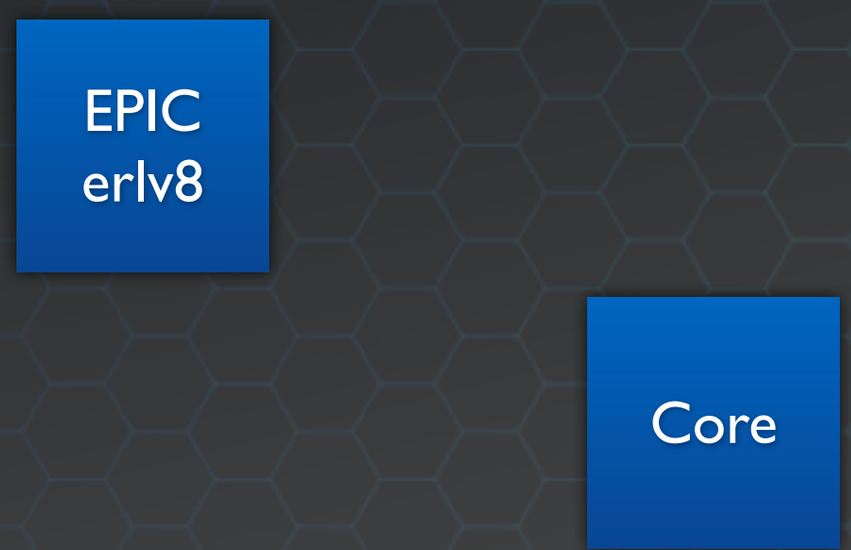
Node Layout

Core

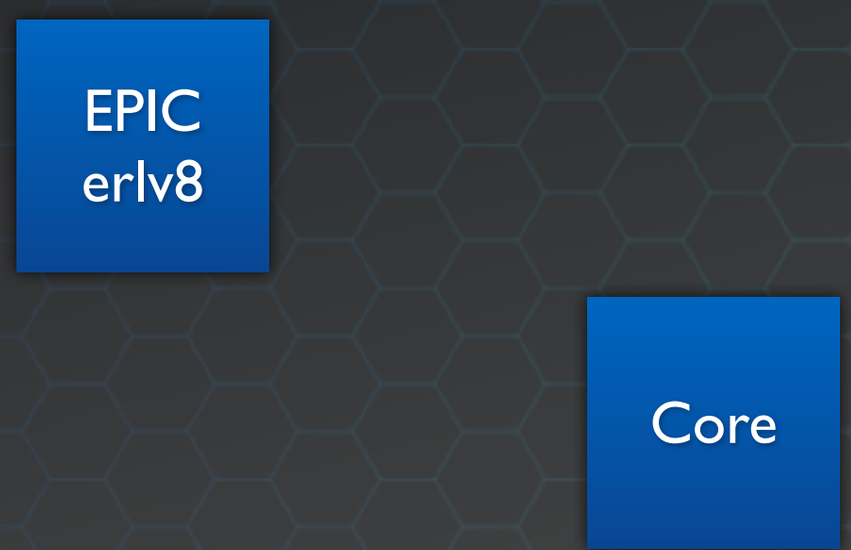
Node Layout

Core

Node Layout



Node Layout



Node Layout

Node Layout

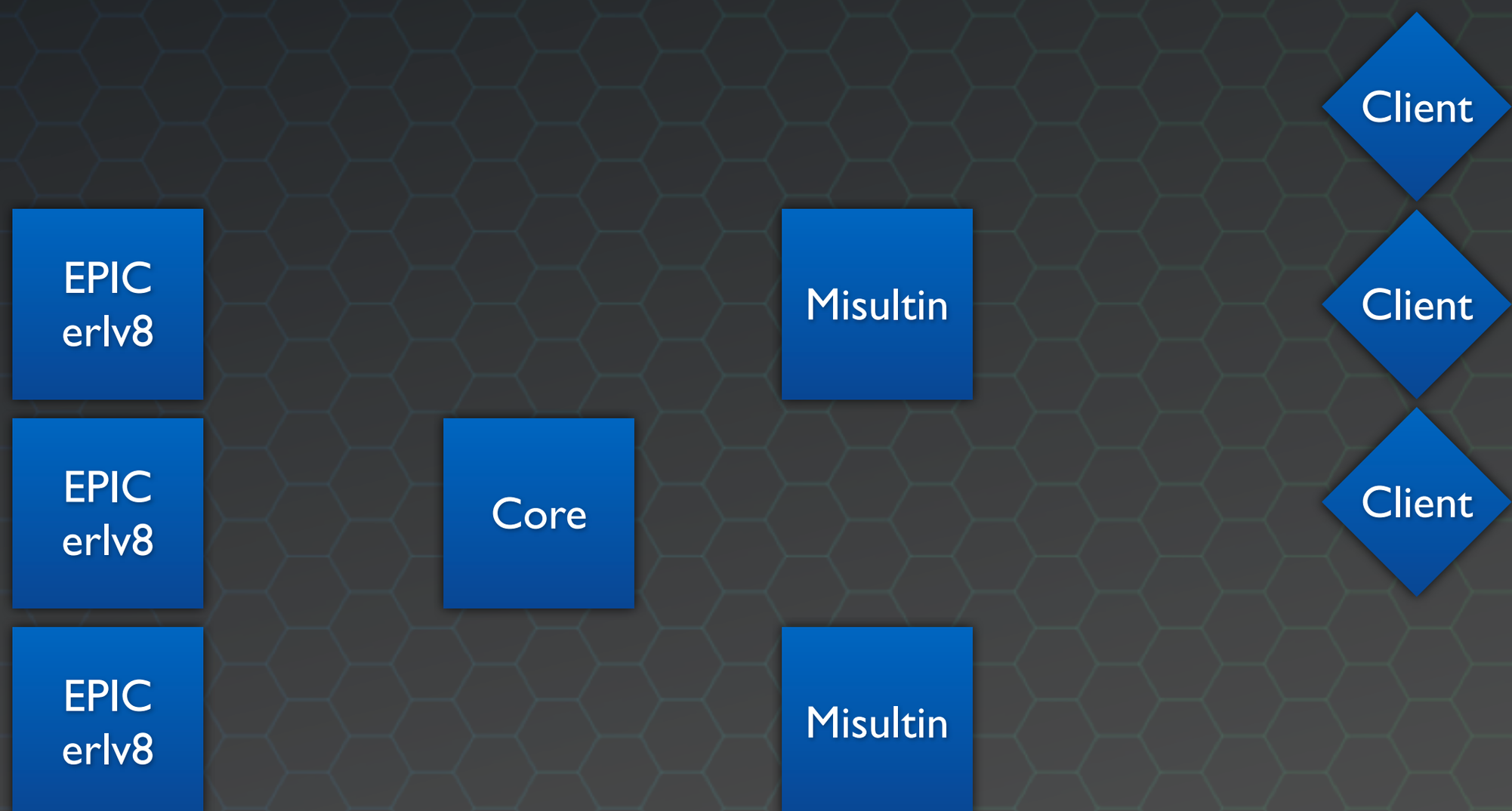
Node Layout

Node Layout

Node Layout

Node Layout

Node Layout

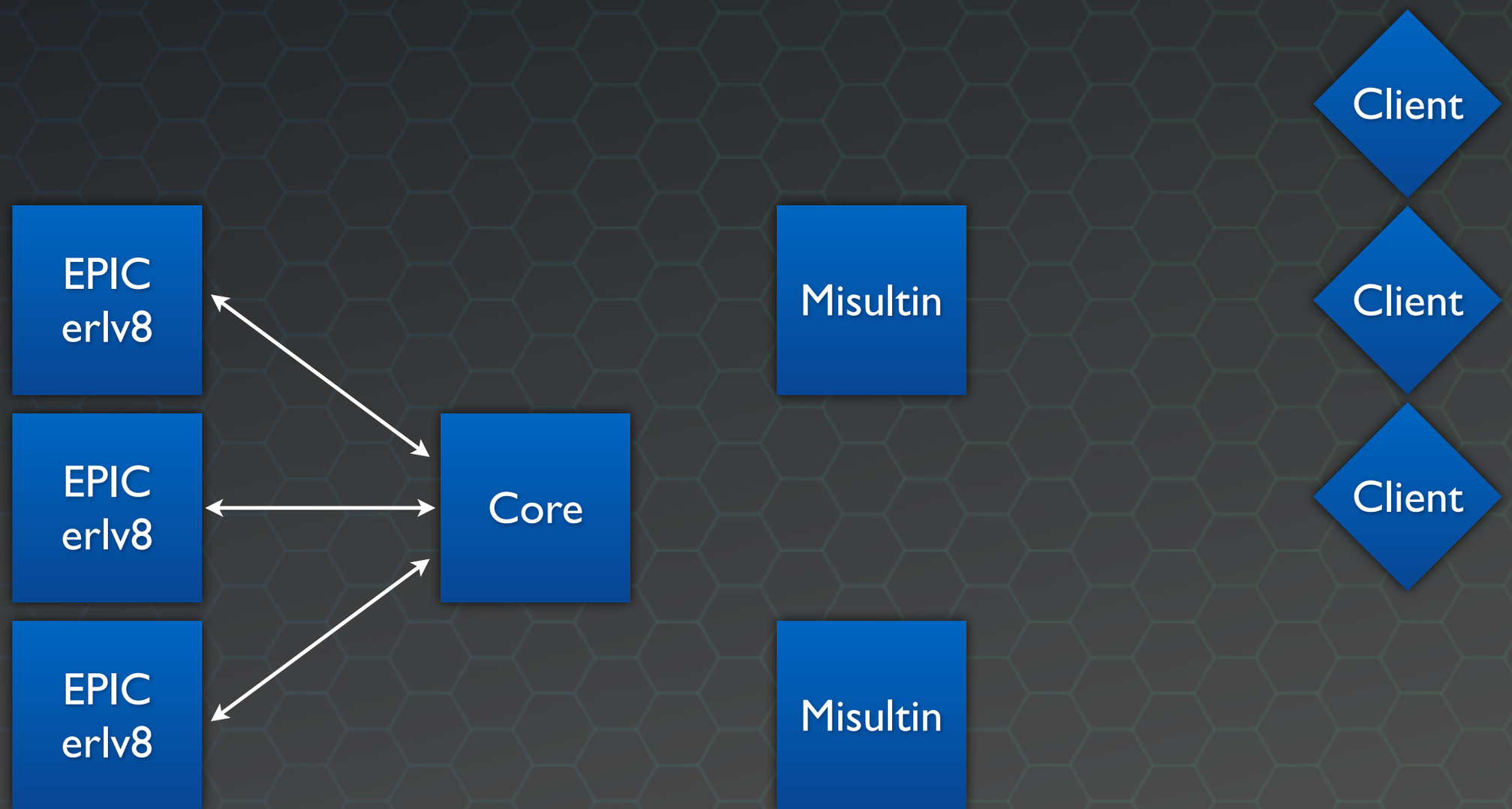

EPIC
erlv8
EPIC
erlv8
EPIC
erlv8
Core
Misultin
Misultin
Client
Client
Client

Node Layout

Place a Fight

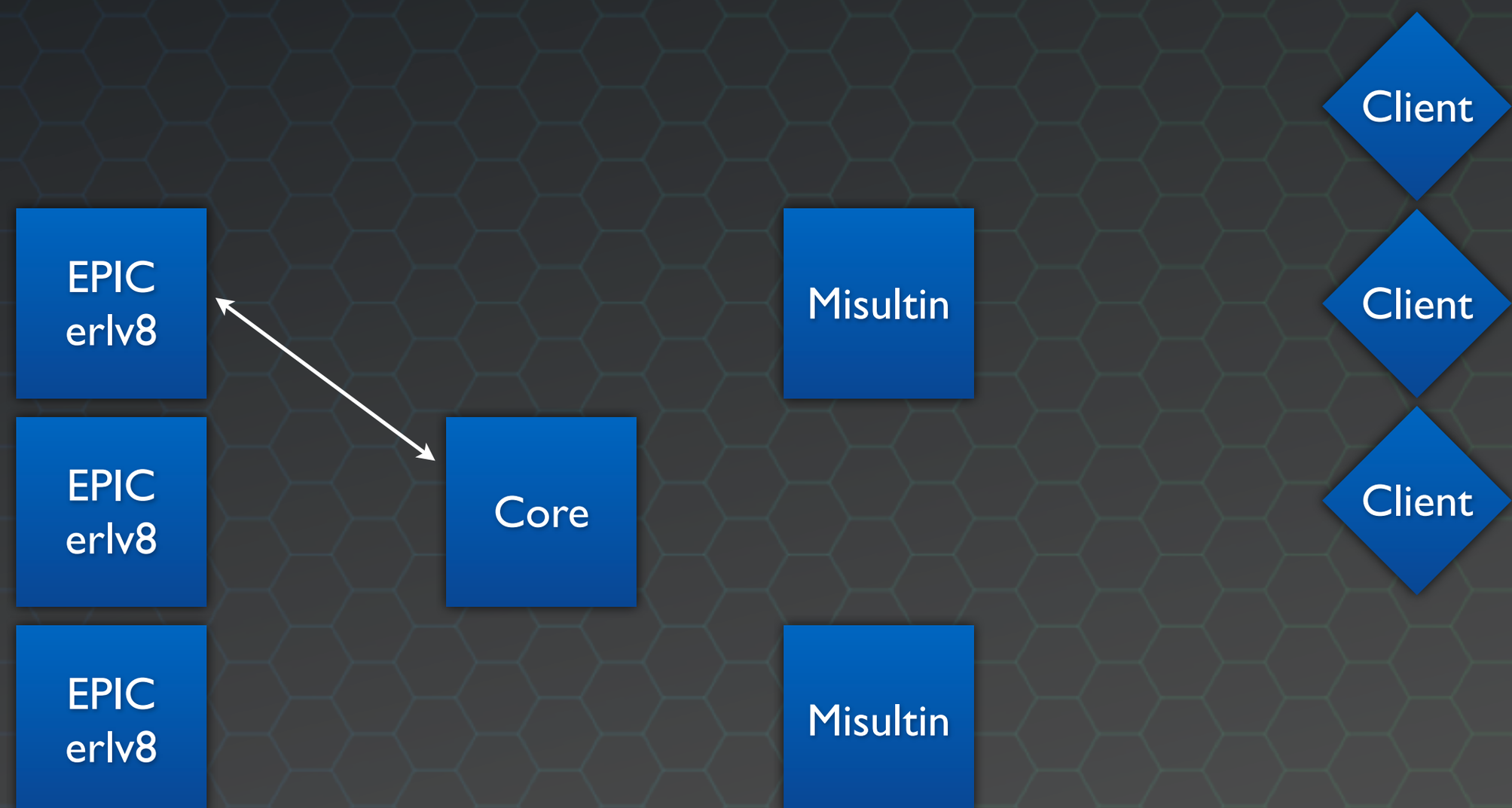
Node Layout

Place a Fight



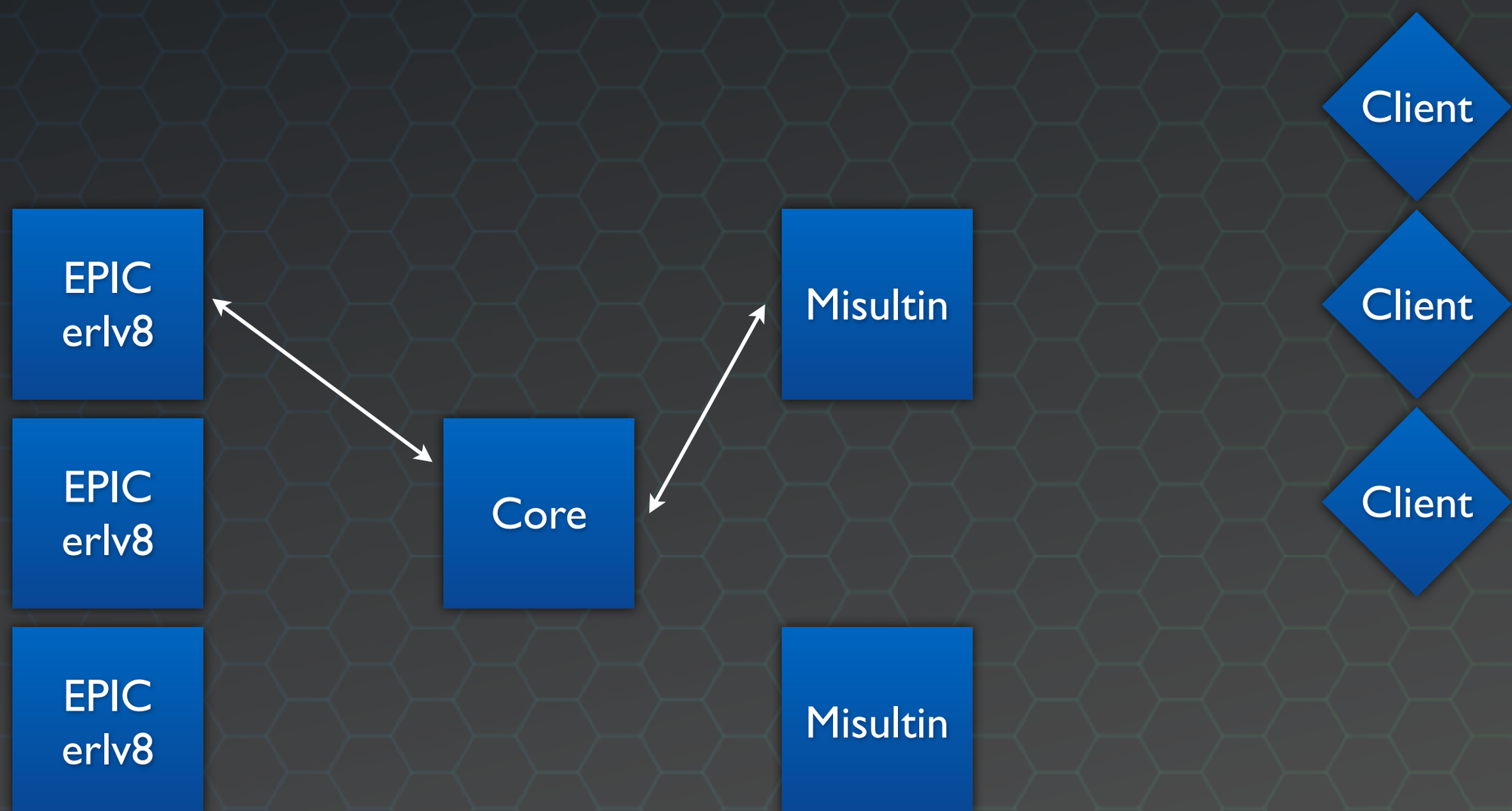
Node Layout

Subscribe



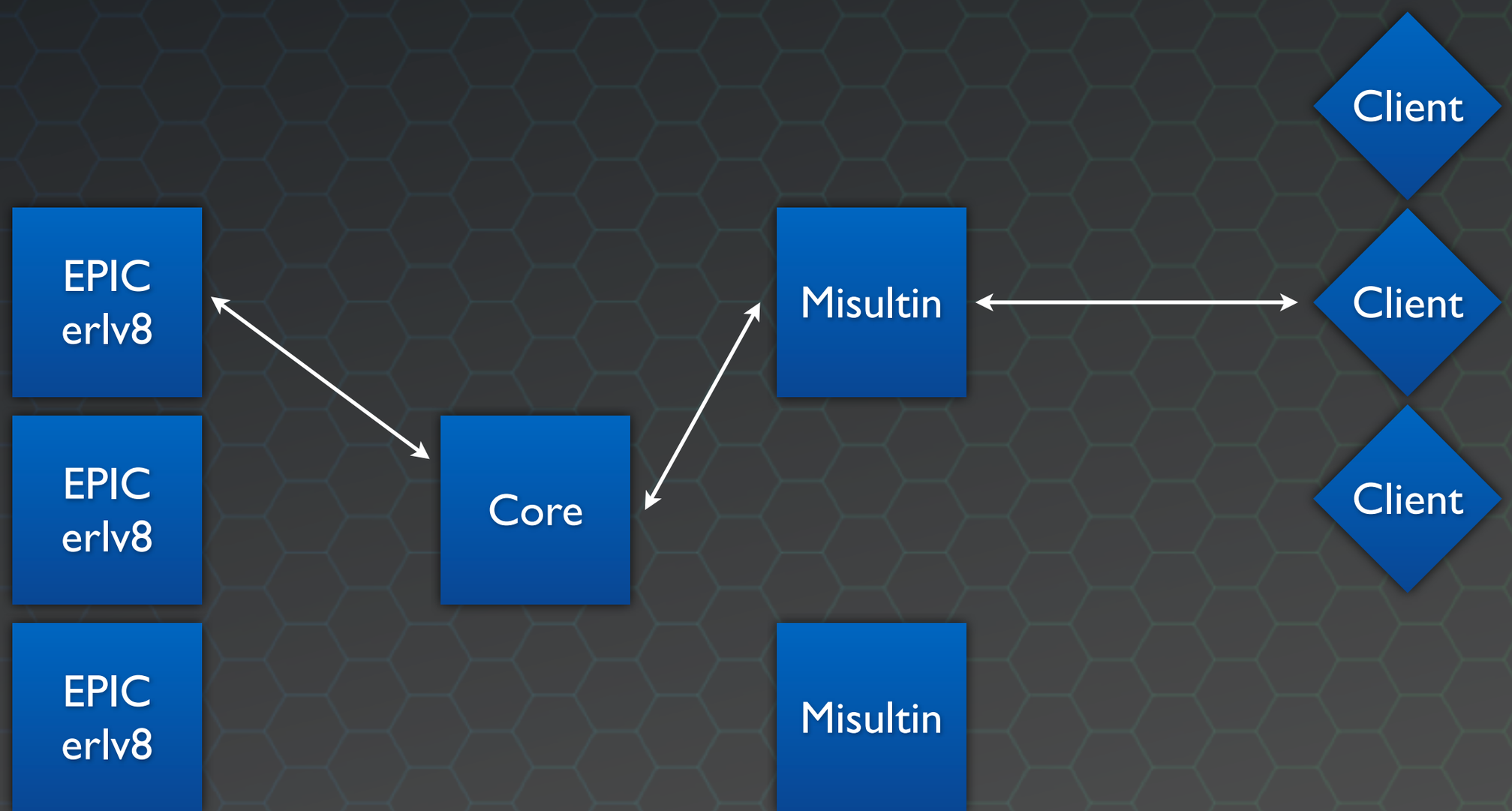
Node Layout

Subscribe

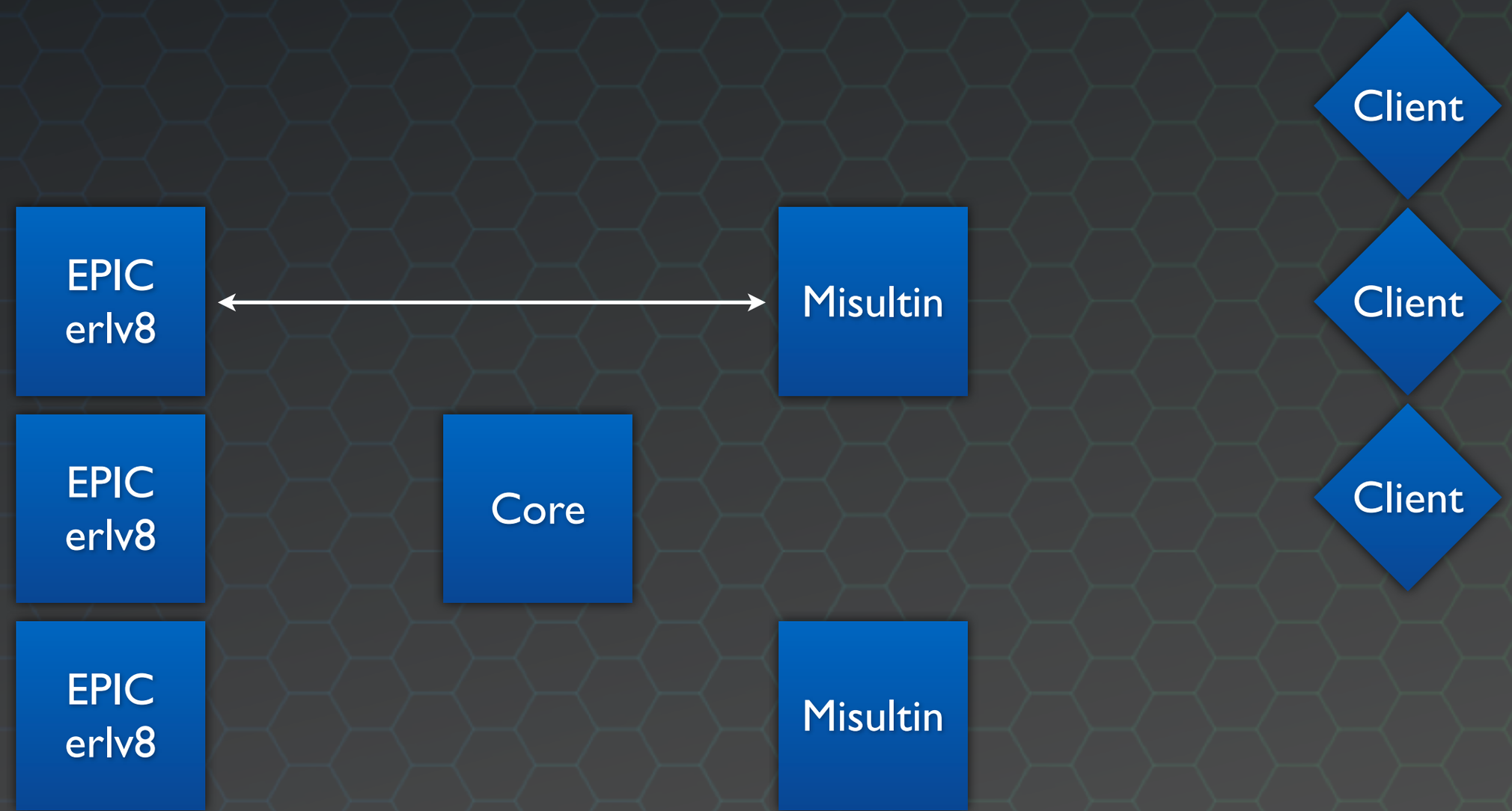


Node Layout

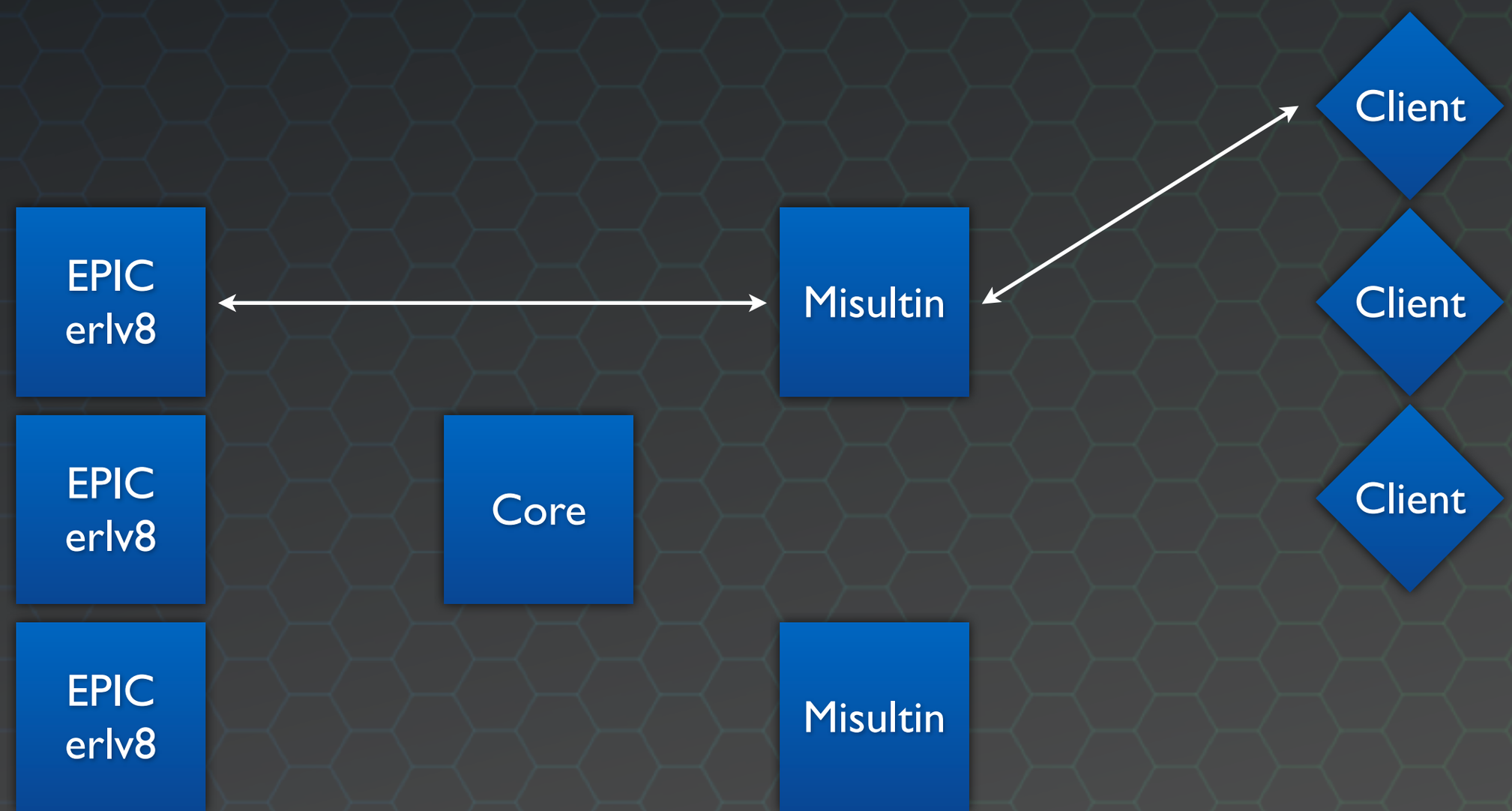
Subscribe



Node Layout
Report Events



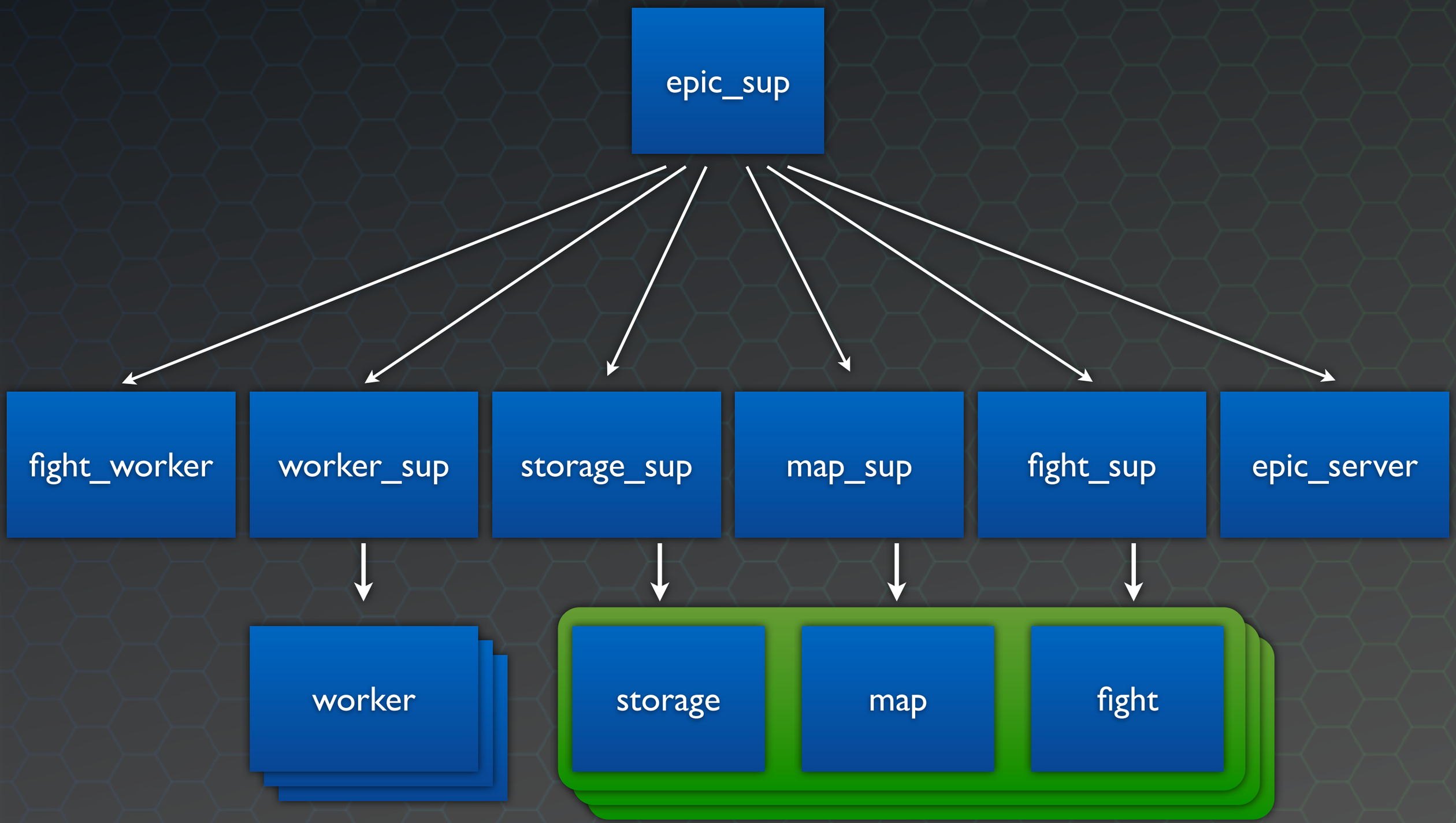
Node Layout
Report Events



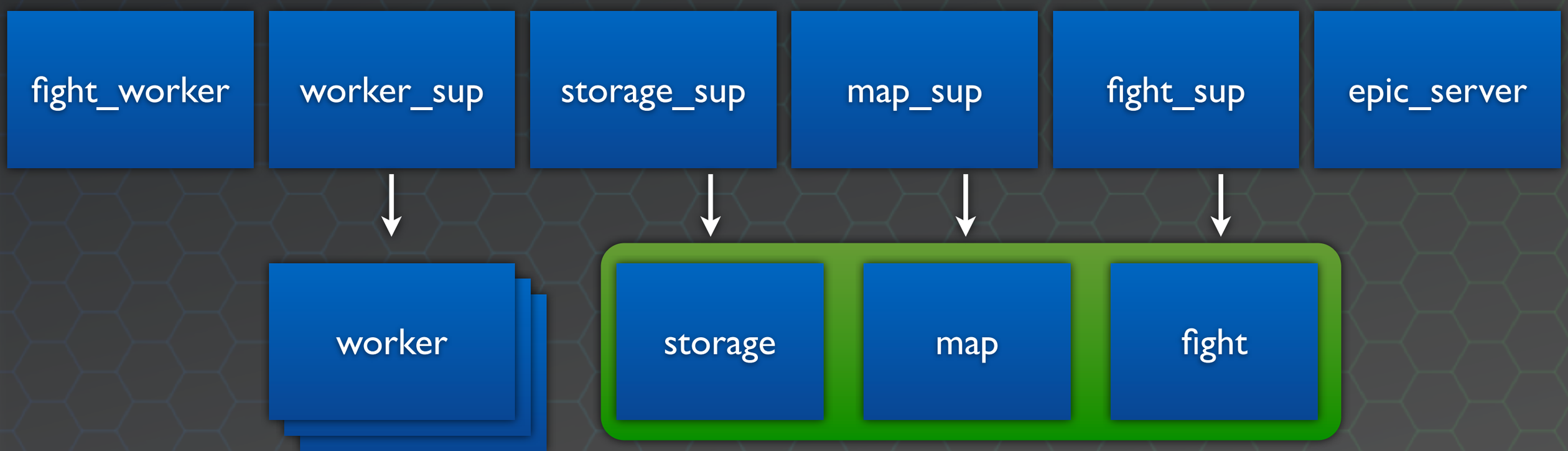
EPIC

Process Layout

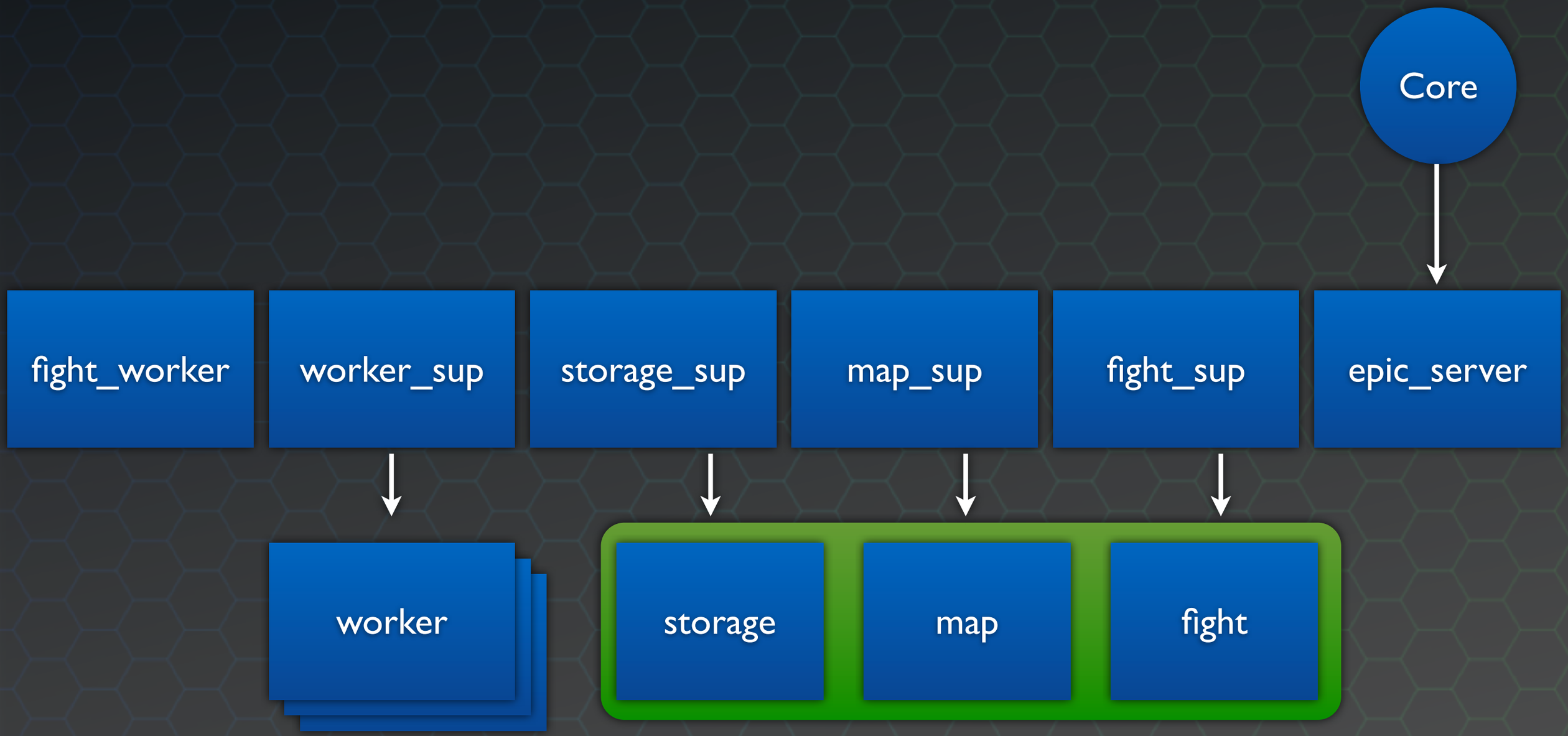
EPIC


epic_sup
fight_worker
worker_sup
storage_sup
map_sup
fight_sup
epic_server
worker
storage
map
fight

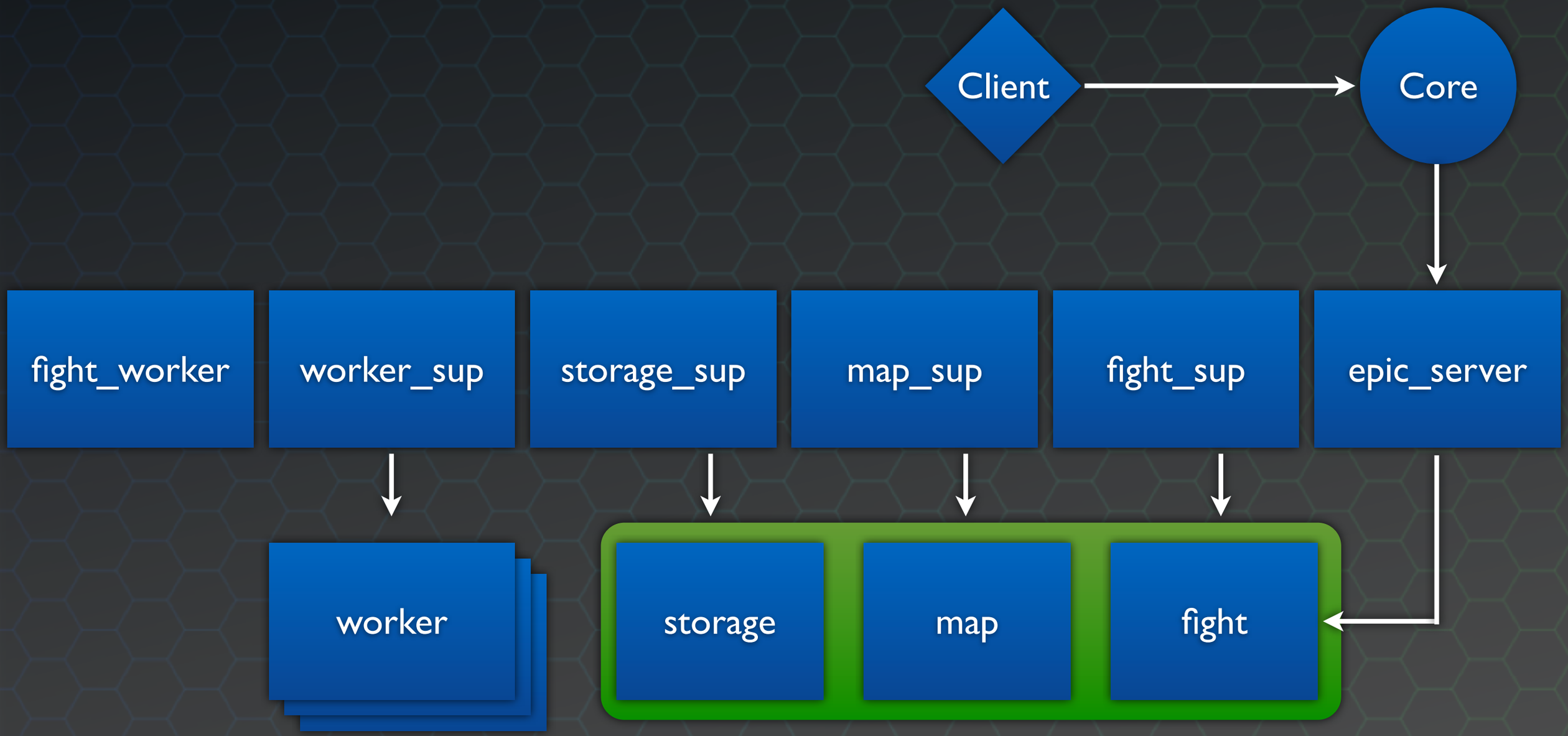
Add Fight


fight_worker
worker_sup
storage_sup
map_sup
fight_sup
epic_server
worker
storage
map
fight

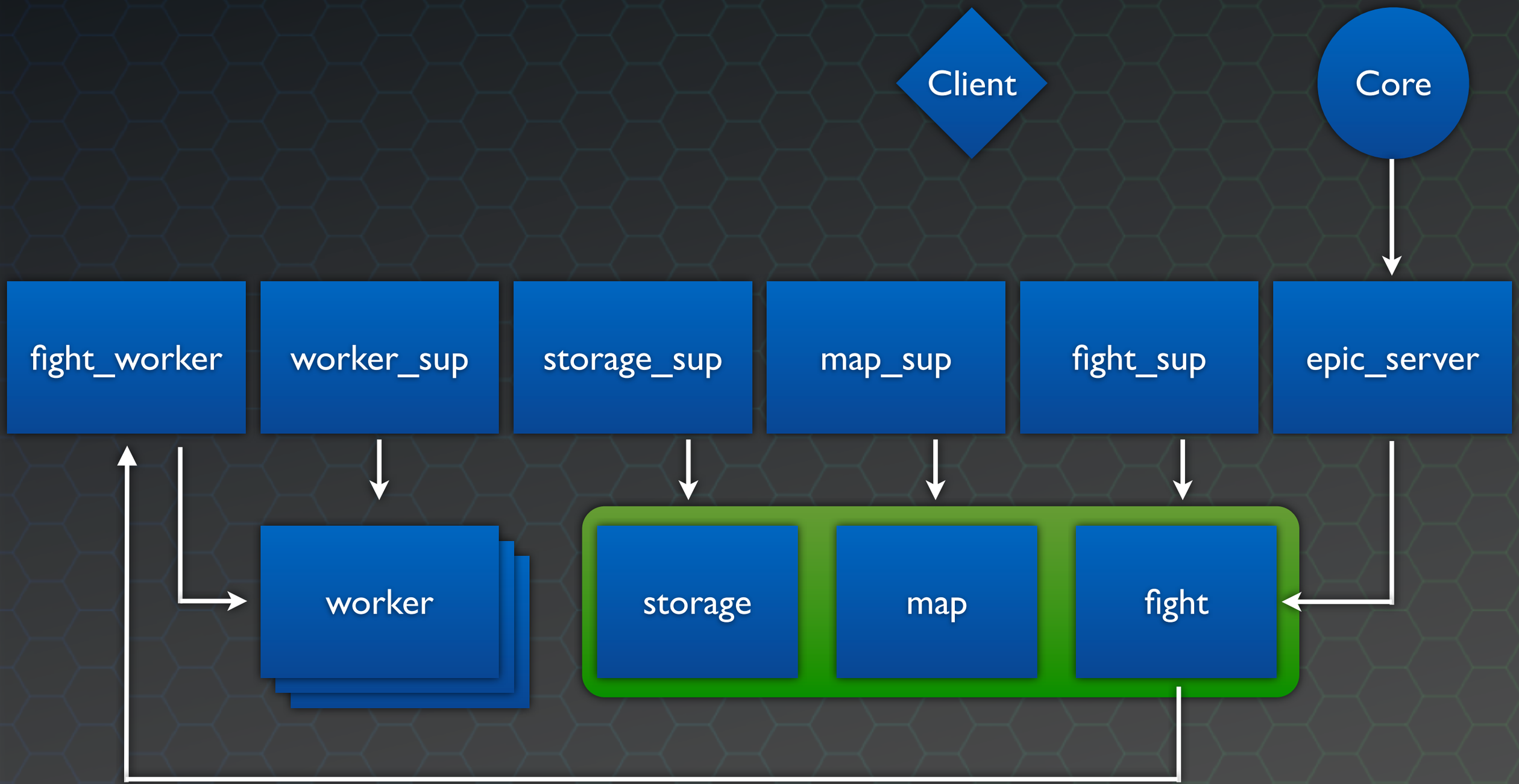
Add Fight


Core
fight_worker
worker_sup
storage_sup
map_sup
fight_sup
epic_server
worker
storage
map
fight

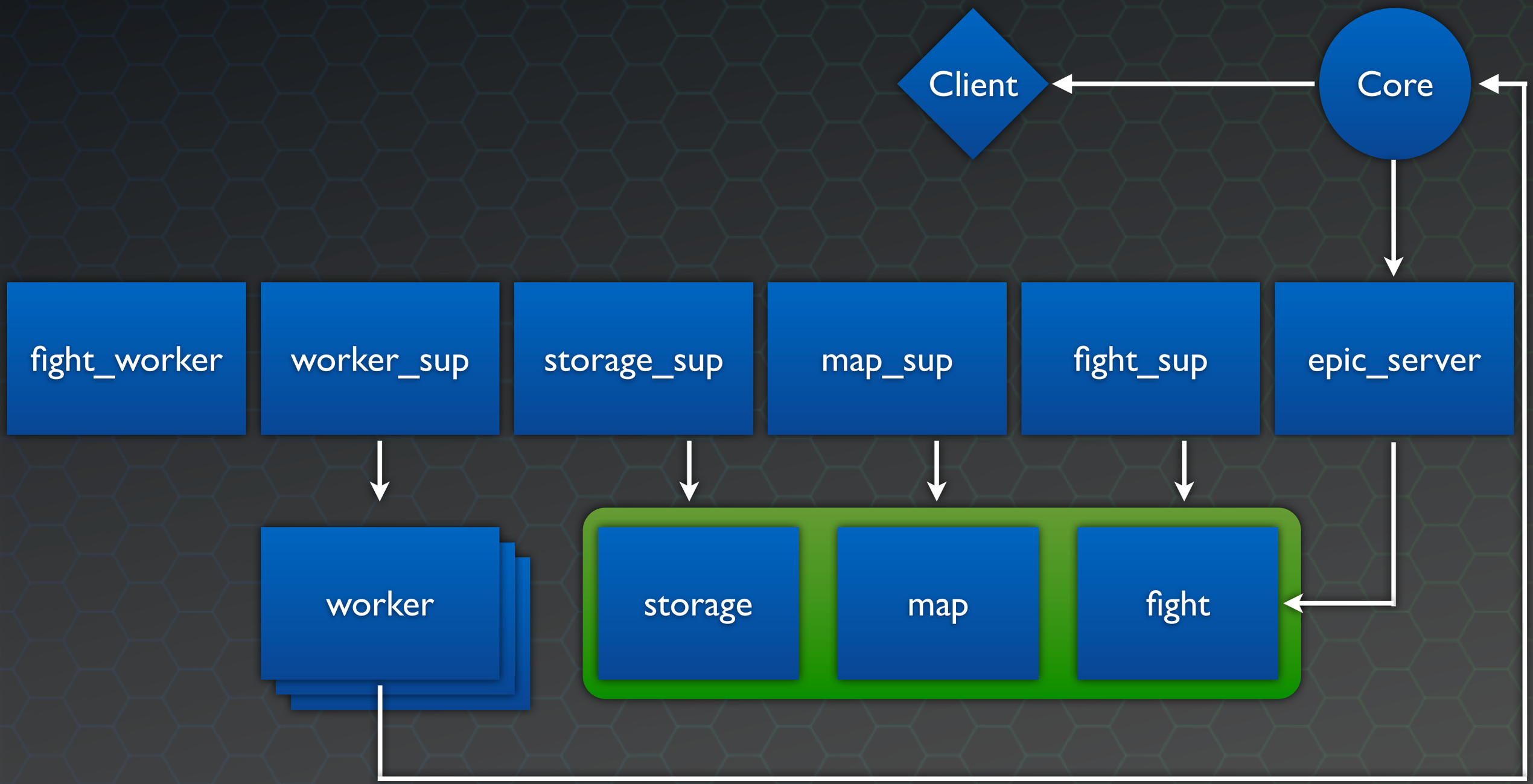
SUBSCRIBE


Client
Core
fight_worker
worker_sup
storage_sup
map_sup
fight_sup
epic_server
worker
storage
map
fight

Turn


Client
Core
fight_worker
worker_sup
storage_sup
map_sup
fight_sup
epic_server
worker
storage
map
fight

Report


Client
Core
fight_worker
worker_sup
storage_sup
map_sup
fight_sup
epic_server
worker
storage
map
fight

EPIC

Surprise!

Big Tasty Code

WEBSOCKETS

```
init([]) ->
        % trap_exit -> this gen_server needs to be supervised
    process_flag(trap_exit, true),
        % start misultin & set monitor
    misultin:start_link([{port, ?PORT},
                         {loop, fun(Req) -> handle_http(Req) end},
                         {ws_loop, fun (Ws) ->
                                        "/fight/" ++ Fight = Ws:get(path),
                                        {ok, FPid} = center_server:get_fight(Fight),
                                        {ok, Server} = ws_sup:start_child(Ws, FPid),
                                        handle_websocket(Ws, Server)
                                    end}, {ws_autoexit, false}]),
    erlang:monitor(process, misultin),
    {ok, #state{port = ?PORT}}.
```

WEBSOCKETS

```
init([]) ->
        % trap_exit -> this gen_server needs to be supervised
    process_flag(trap_exit, true),
        % start misultin & set monitor
    misultin:start_link([{port, ?PORT},
                         {loop, fun(Req) -> handle_http(Req) end},
                         {ws_loop, fun (Ws) ->
                                        "/fight/" ++ Fight = Ws:get(path),
                                        {ok, FPid} = center_server:get_fight(Fight),
                                        {ok, Server} = ws_sup:start_child(Ws, FPid),
                                        handle_websocket(Ws, Server)
                                    end}, {ws_autoexit, false}]),
    erlang:monitor(process, misultin),
    {ok, #state{port = ?PORT}}.
```

```
handle_websocket(Ws, Server) ->
    receive
        {browser, Data} ->
            ws:incoming(Server, Data),
            handle_websocket(Ws, Server);
        closed ->
            ws:stop(Server),
            closed;
        _Ignore ->
            handle_websocket(Ws, Server)
    end.
```

Connecting Nodes

```
-define(CORE, 'dscore@schroedinger').

%% ====================================
%% Application callbacks
%% ====================================

start(_StartType, _StartArgs) ->
    net_adm:ping(?CORE),
    epic_sup:start_link().
```

Connecting Nodes

```
-define(CORE, 'dscore@schroedinger').

%% ====================================
%% Application callbacks
%% ====================================

start(_StartType, _StartArgs) ->
    net_adm:ping(?CORE),
    epic_sup:start_link().
```

```
init([]) ->
    storage:init(),
    {ok, #state{fights = dict:new()}, 1000}.
```

Connecting Nodes

```
-define(CORE, 'dscore@schroedinger').

%% ====================================
%% Application callbacks
%% ====================================

start(_StartType, _StartArgs) ->
    net_adm:ping(?CORE),
    epic_sup:start_link().
```

```
init([]) ->
    storage:init(),
    {ok, #state{fights = dict:new()}, 1000}.
```

```
handle_info(timeout, State) ->
    {ok, Pid} = center:register(self()),
    io:format("Pid: ~p.~n", [Pid]),
    Ref = erlang:monitor(process, Pid),
    io:format("Ref: ~p.~n", [Ref]),
    {noreply, State#state{ref = Ref}};
```

EPIC

ending!