Erlang and KAZOO

Presented by: James Aimonetti

Lead Architect @ 2600Hz

https://github.com/2600hz/kazoo

2600hz CLOUD TELECOM

What is Kazoo?

* Distributed Telephony Engine

* Layered approach to processing calls/events

* Event-driven design

2600hz
CLOUD TELECOM

Distributed Telephony Engine

* Requirements

- Redundancy/ Fault tolerance
    - a) Supervision of calls
    - b) Server and data center

- High level call handling
- Scale horizontally—easily

2600hz CLOUD TELECOM

Layered Approach to Processing Calls/events

* Under the hood:
  - OpenSIPS/ Kamailio
  - FreeSWITCH
  - RabbitMQ
  - BigCouch
  - Kazoo

Layered Approach to Processing Calls/events

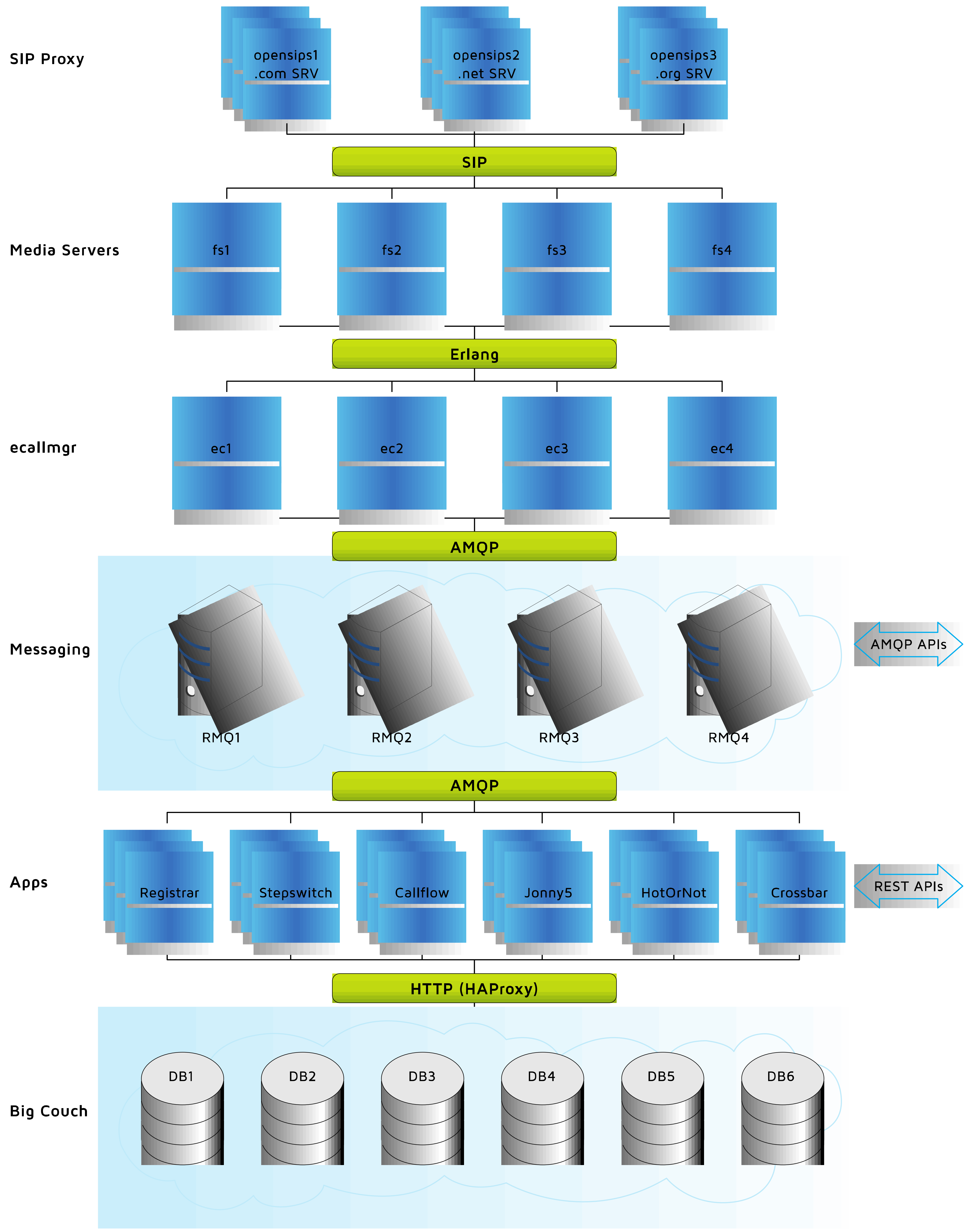
* SIP
  - Carriers/Clients <=> OpenSIPS/ Kamailio/ FreeSWITCH

* Distributed Erlang
  - FreeSwitch <=> ecallmgr (low level FreeSWITCH abstraction)

* AMQP
  - ecallmgr/whapps <=> RabbitMQ

*HTTP
  - whapps <=> BigCouch
  - Crossbar (whapp) <=> REST Clients (KazooUI)
  - Pivot (whapp) <=> Your Web Server

Kazoo Server Layout



2600hz CLOUD TELECOM

Event Driven Design

* Incoming calls

* Registrations

* HTTP REST APIs

* Timer-based cleanup

* Direct DB manipulation (naughty)

2600hz CLOUD TELECOM

Building Kazoo

* Utilities (wh_json, wh_json_validator, wh_util)

* AMQP-based behaviour (gen_listener)

* Caching of data (wh_cache)

* Callflow processing (cf_exe + friends)

* Coupling FTW (gen_listener + gen_fsm)

Utilities (wh_json, wh_json_validator, wh_util)

wh_json (lib/whistle-1.0.0/src/wh_json.erl)

* Beginning
- Used mochijson2 for encoding/decoding
- Interact with the data structure as opaque object (like dict)

* Now
- types defined for use in specs
    a) wh_json:object() and wh_json:objects() most common

- aliasing type conversion (get_binary_value, get_integer_value)
- ability to change encoder/decoder (using ejson atm)

Utilities (wh_json, wh_json_validator, wh_util)

wh_json_validator (lib/whistle-1.0.0/src/wh_json_validator.erl)
  ** based on http://tools.ietf.org/html/draft-zyp-json-schema-03

* Beginning
  - Edouard (intern) wrote first module

* Now
  - Both Karl and James have written their versions
  - is_valid(JObj :: wh_json:object(), Schema :: wh_json:object())
  - now returns {'pass', FixedJObj :: wh_json:object()} |
    {'fail', [{FailedKeyPath, FailureMessage},...]}

  ** pending rewrite or using 3rd party library

Utilities (wh_json, wh_json_validator, wh_util)

wh_util (lib/whistle-1.0.0/src/wh_util.erl)

* Beginning
    - Dumping ground
        a) Type conversion, timer offsets, encoding account IDs, and more

* Now
    - Slowly breaking out into meaningfully-named modules

2600hz
CLOUD TELECOM

AMQP-based behaviour (gen_listener)

gen_listener lib/whistle-1.0.0/src/gen_listener.erl
- built on top of gen_server
- async processing of received AMQP messages

* Beginning
- Each consumer used low level primitives (new_queue, consume)
- Lots of channels (expensive), lots of queues
- Each process responsible for handling broker/connection errors
- Herding cats

* Now
- Use gen_listener
- Spawns handlers for matching messages
- Fewer consumers (faster startup)
- All AMQP-specific code is hidden from application code
    a) Toying with XMPP extensions to gen_listener

Caching of data (wh_cache)

wh_cache (lib/whistle-1.0.0/src/wh_cache.erl)

* Beginning
    - dict wrapped with a gen_server, with per-entry TTL
    - registered name, for all to use
    - serialized access
    - mostly for caching DB objects
    - Simple API: store, fetch, peek, erase, flush

Caching of data (wh_cache)

wh_cache (lib/whistle-1.0.0/src/wh_cache.erl)

* Evolving

- Migrated to ETS-based storage
- using a record instead of 2-tuples with more metadata
- Application-localized cache processes
- complex work stored
- callbacks on entry expiration/expulsion
- soft real-time decision making easier
- invalidating entries remained problematic

2600hz CLOUD TELECOM

Caching of data (wh_cache)

wh_cache (lib/whistle-1.0.0/src/wh_cache.erl)

* Current
    - using gen_listener (when appropriate) in place of gen_server
    - feed document change events to invalidate cache entries
    - near real-time updates now
    - original API intact, more bells and whistles (bells and kazoos?)
    - auto-flush tied to other system events (server topology changes)

Callflow processing (cf_exe + friends)

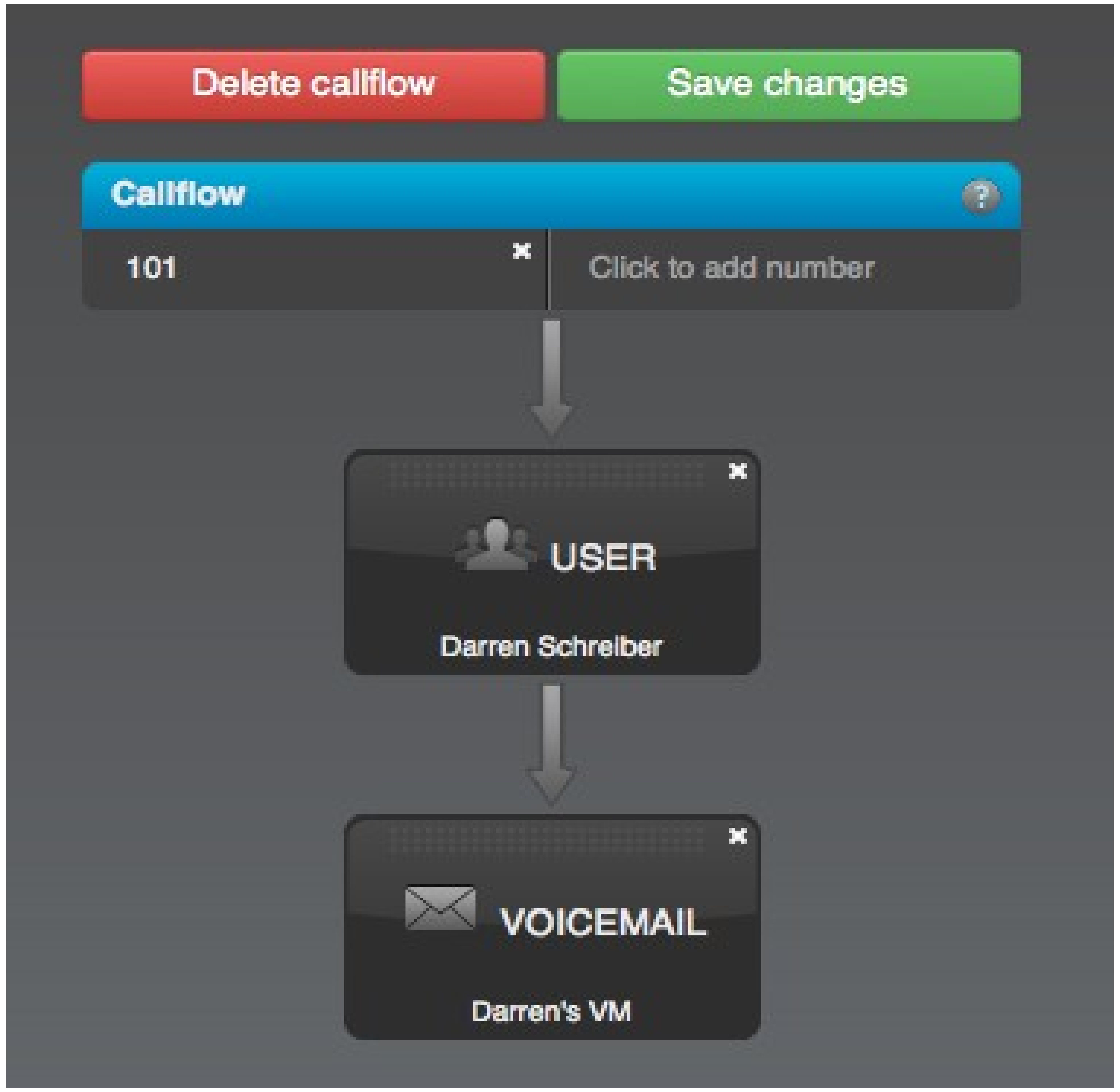
whistle_apps/apps/callflow/src/cf_exe.erl
whistle_apps/apps/callflow/src/modules/

* Beginning
    - Processed callflow directly, moving into the cf_* modules
    - Lots of defensive code (Pokemon exception handling)

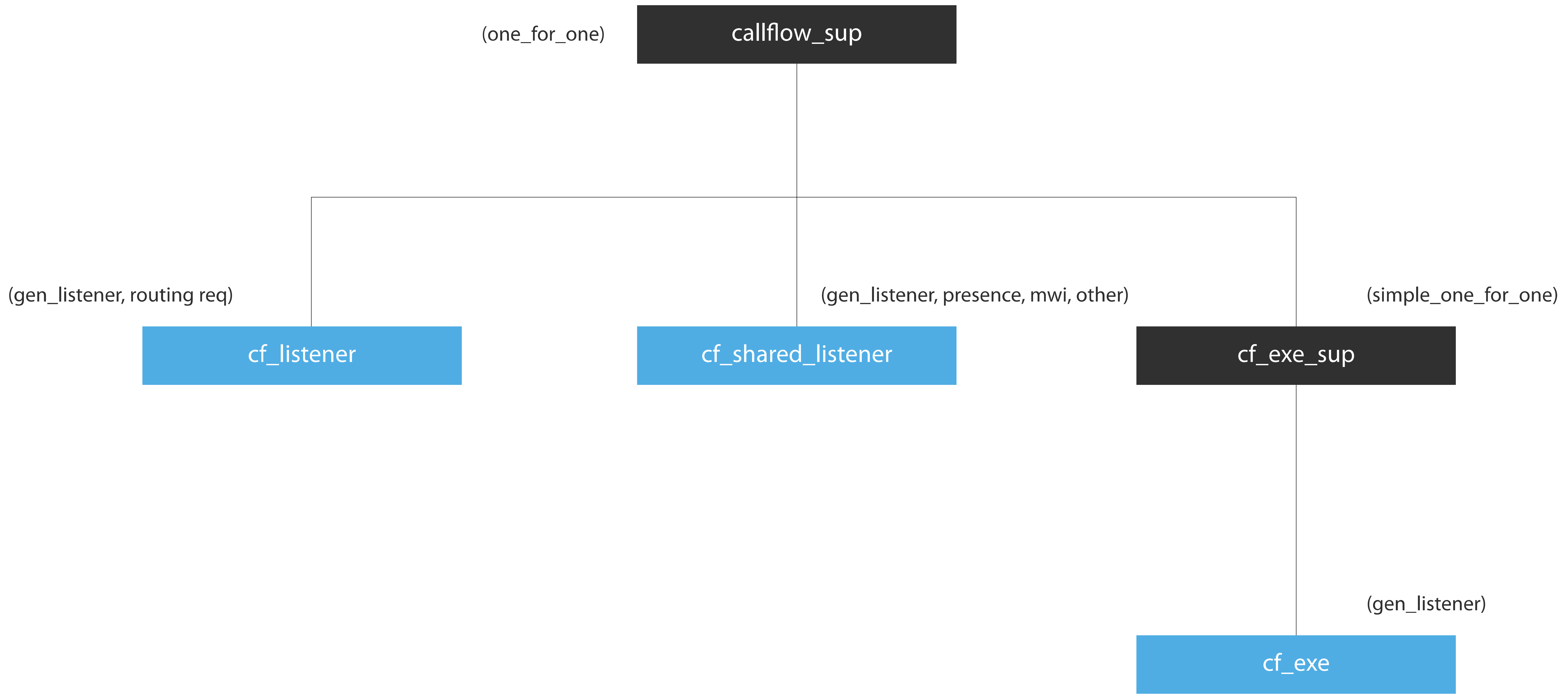
* Now
    - cf_exe is a gen_listener, receives and proxies call events
    - navigates the callflow JSON tree
    - spawns/monitors cf_* modules to process the nodes in the tree

Callflow processing (cf_exe + friends)

```
"flow": {
      "data": {
         "id": "128d81866e595be608a51e51e03be",
         "timeout": "20",
         "can_call_self": false
      },
      "module": "user",
      "children": {
         "_": {
            "data": {
               "id": "9afa4973c3b4440f522955fc023a9"
            },
            "module": "voicemail",
            "children": {}
         }
      }
   }
```



Callflow application process layout



2600hz CLOUD TELECOM

Coupling FTW (gen_listener + gen_fsm)

* ACDc: Automatic Call Distribution commander (call queues)

* Both agents and call queues are represented by these couplings
    - Better than mixing FSM-style state transitions into gen_server

2600hz CLOUD TELECOM

Coupling FTW (gen_listener + gen_fsm)

* Beginning

```
handle_cast({dtmf, DTMF}, #state{status='connecting'}=State) →
  State1 = process_dtmf(DTMF, State), % might update status
  {noreply, State};
handle_cast({dtmf, _}, State) →
  {noreply, State};
```

** Fine for very simple cases, but quickly degenerates

Coupling FTW (gen_listener + gen_fsm)

ACDc layout of agent processes:

* Supervisory Tree

    - acdc_agents_sup:
        - simple_one_for_one of acdc_agent_sup's

    - acdc_agent_sup:
        - one_for_all supervisor, per agent

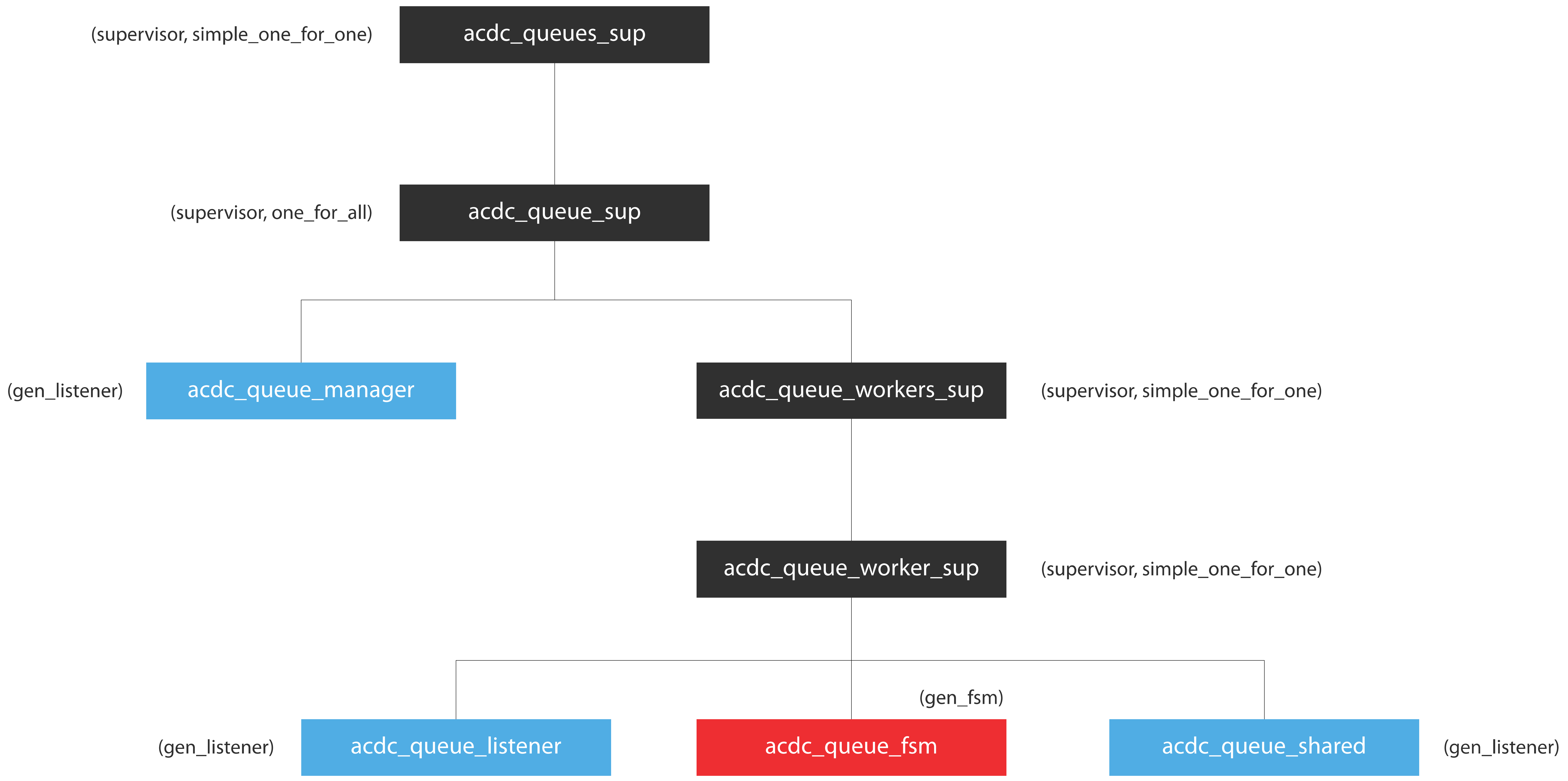
Coupling FTW (gen_listener + gen_fsm)

Processes under acdc_agent_sup

- acdc_agent_fsm:
    - gen_fsm
    - main states – ready, connecting, answered, waiting, paused, outbound

- acdc_agent_listener:
    - gen_listener
    - handles receiving call events, acdc events, etc
    - handles sending call commands, acdc commands, etc
    - feeds received events into FSM, recv commands from FSM

Call Queue Process layout



KAZOO

Project: https://github.com/2600hz/kazoo

Website: http://2600hz.org

Me: james@2600hz.org

Interested? We're unofficially hiring for Erlang programmers!

2600hz
CLOUD TELECOM