
EXPLORE
With Erlang Solutions

Erlang SOLUTIONS

HELLO

Szymon Mentel

szymon.mentel@erlang-solutions.com

www.erlang-solutions.com

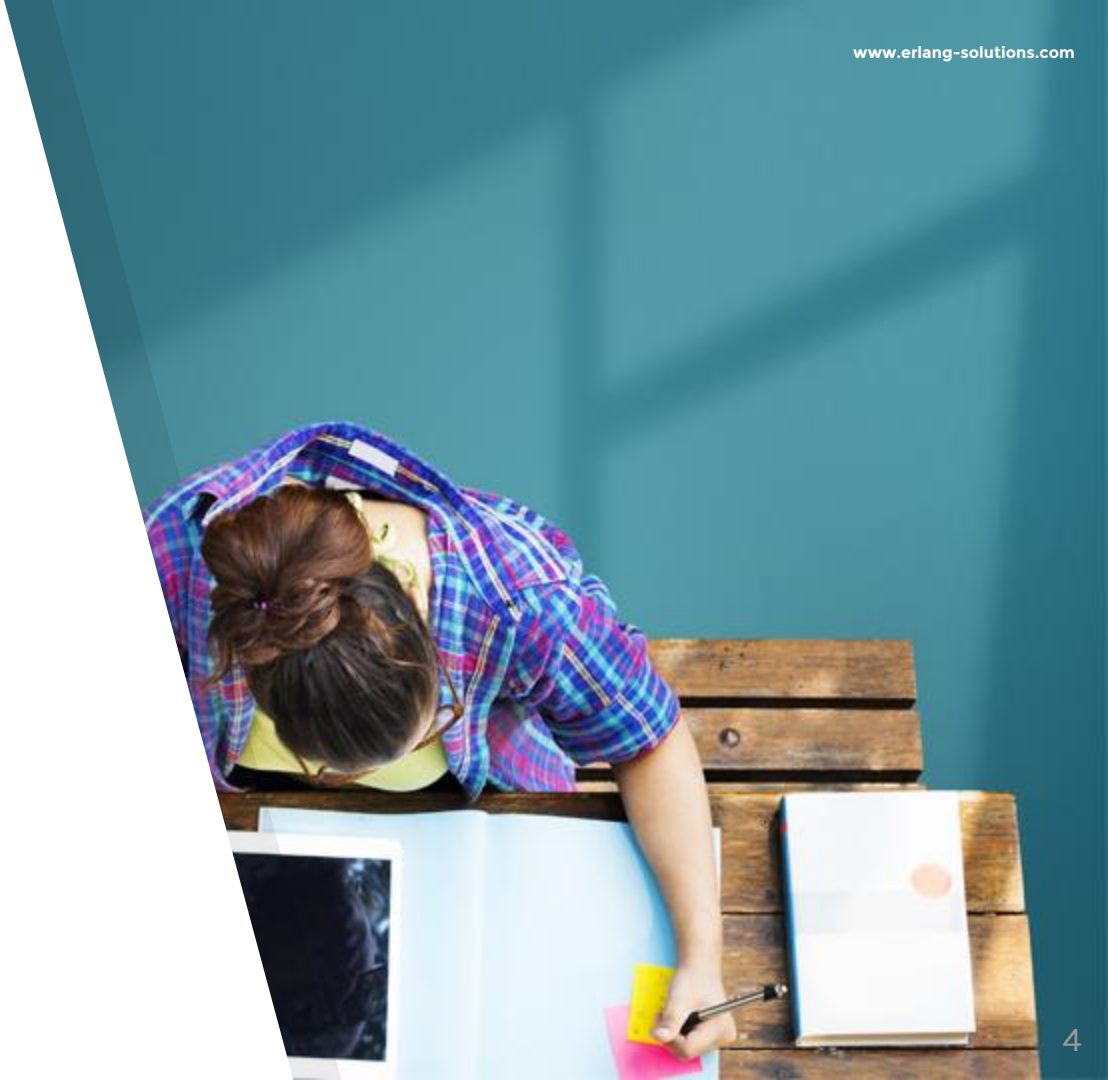
Erlang SOLUTIONS

CRYSTALLIZE YOUR MIXTURE

Elixir Tracing Techniques

1.

What is Tracing in the Erlang VM?



TRACING

- Provides means to monitor concurrency, code execution and memory usage
- No need to trace-compile the code
- Can be used in live system
- **:erlang.trace/3** enables/disables the low-level tracing mechanism in the Erlang Runtime System (which run Elixir!)

TRACE EVENTS

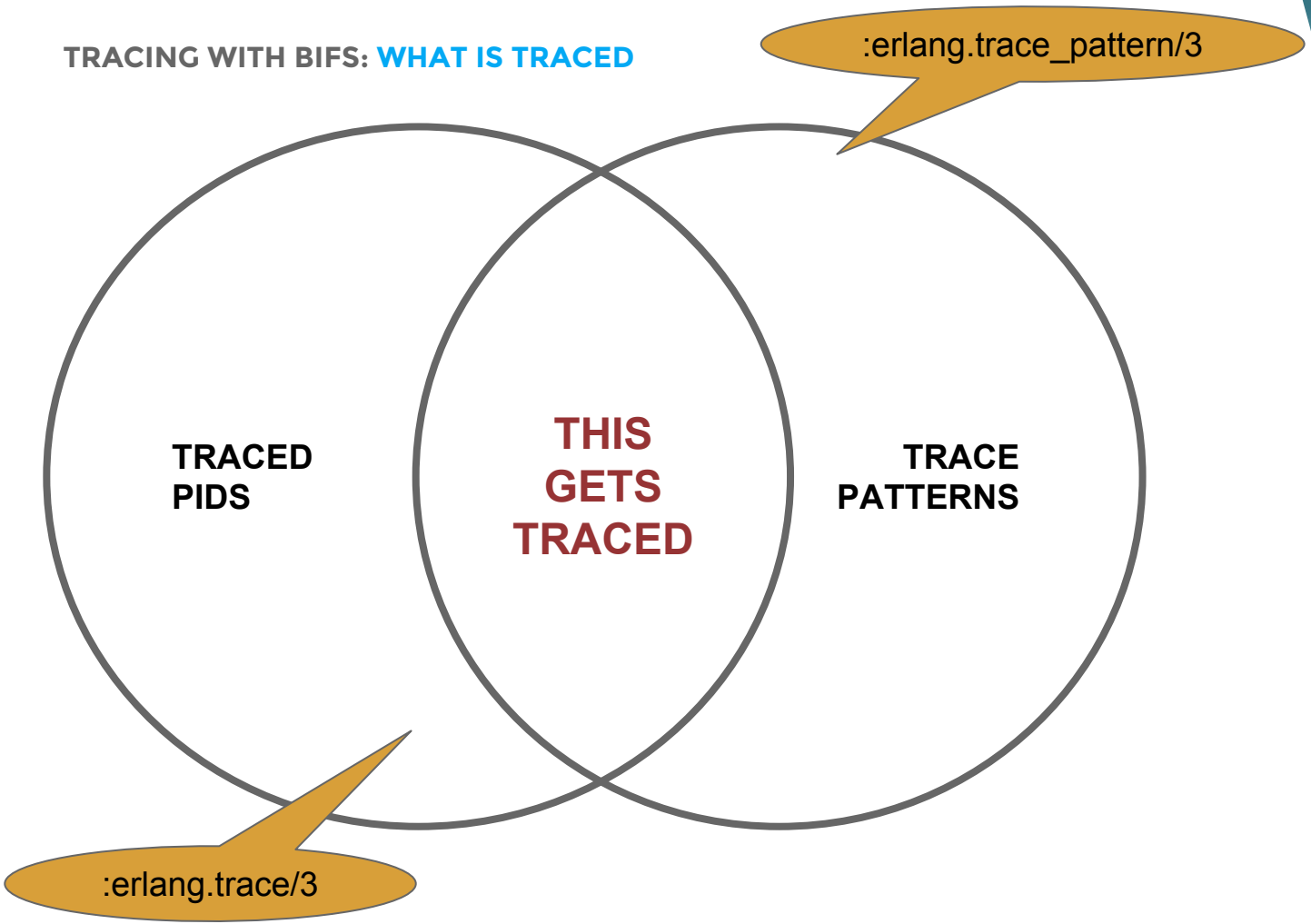
```
{:trace, pid, tag, data1 [,data2]}
```

- Events are sent as messages
- The events are about
  - Message passing
  - GC and memory usage
  - Process activity
- At any one given time, only one process may receive trace events from another process

2.

Low Level Tracing with BIFs

TRACING WITH BIFS: WHAT IS TRACED



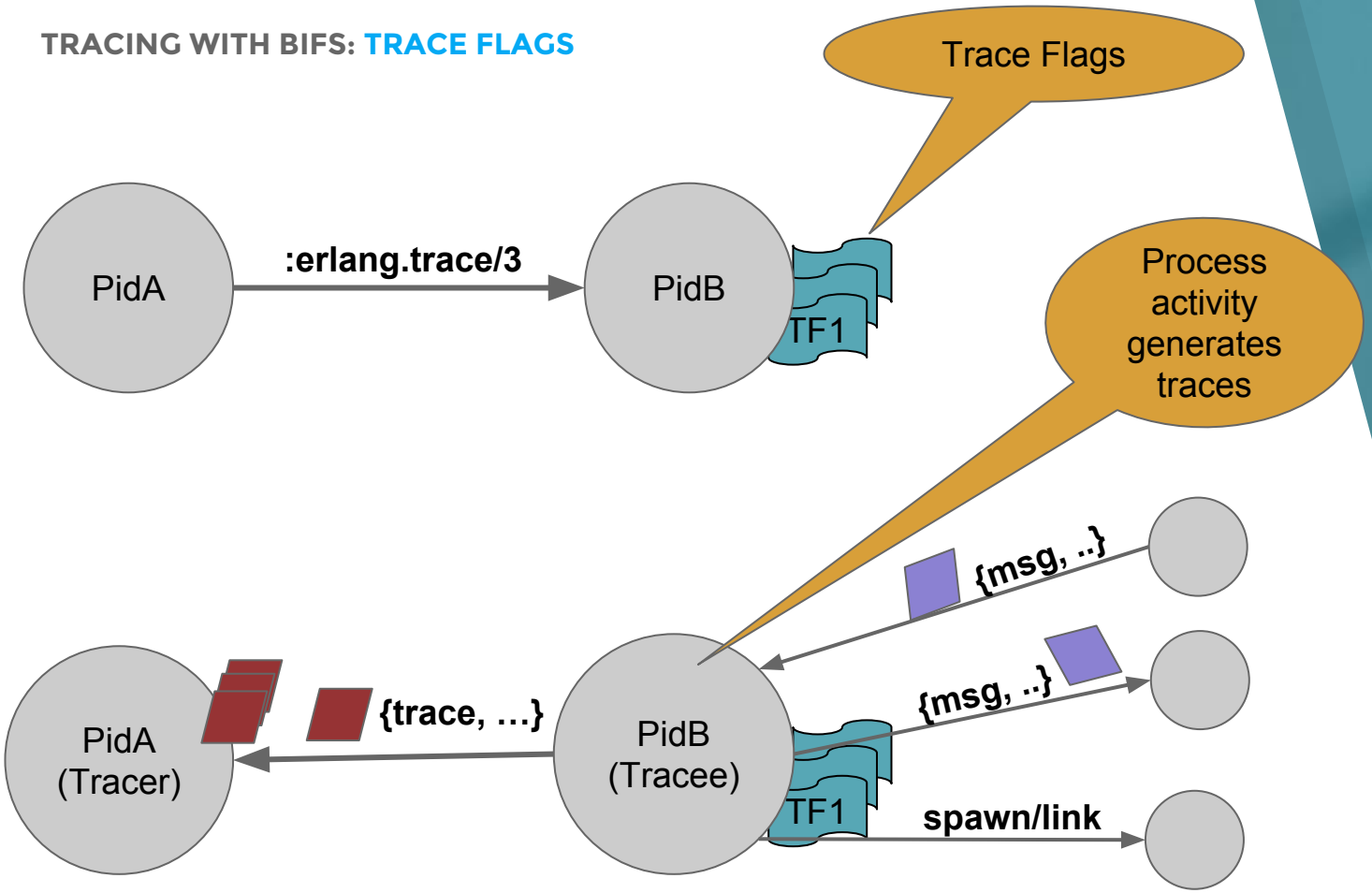
TRACING WITH BIFS: BASICS

```
@doc """
Enables tracing of the commands sent to the KVServer
"""
def trace_commands() do
  procs = :erlang.trace(:all, true, [:call])
  ptrns = :erlang.trace_pattern({KVServer.Command, :run, 1},
                                true, [])
  Logger.info "Matched #{procs} procs and #{ptrns} patterns"
end
```

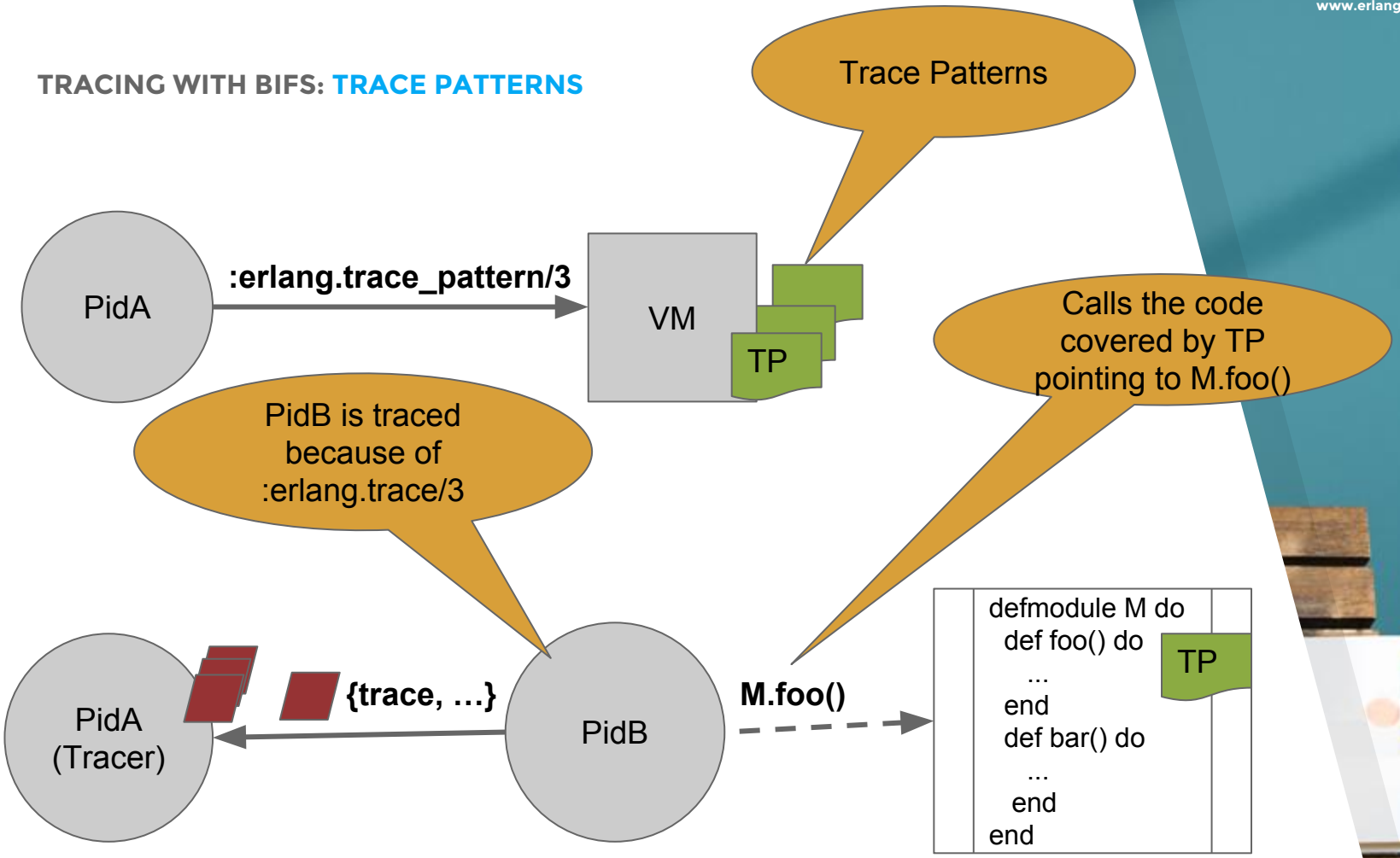
Trace Flags - Who?

{mod, fun, arity} - What?

TRACING WITH BIFS: TRACE FLAGS

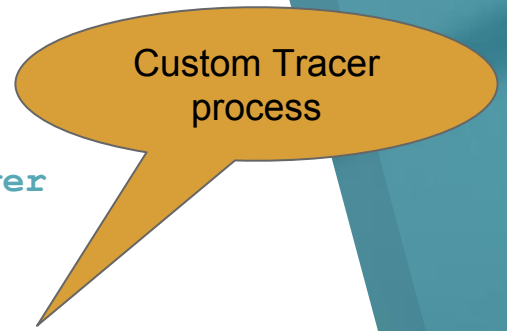


TRACING WITH BIFS: TRACE PATTERNS

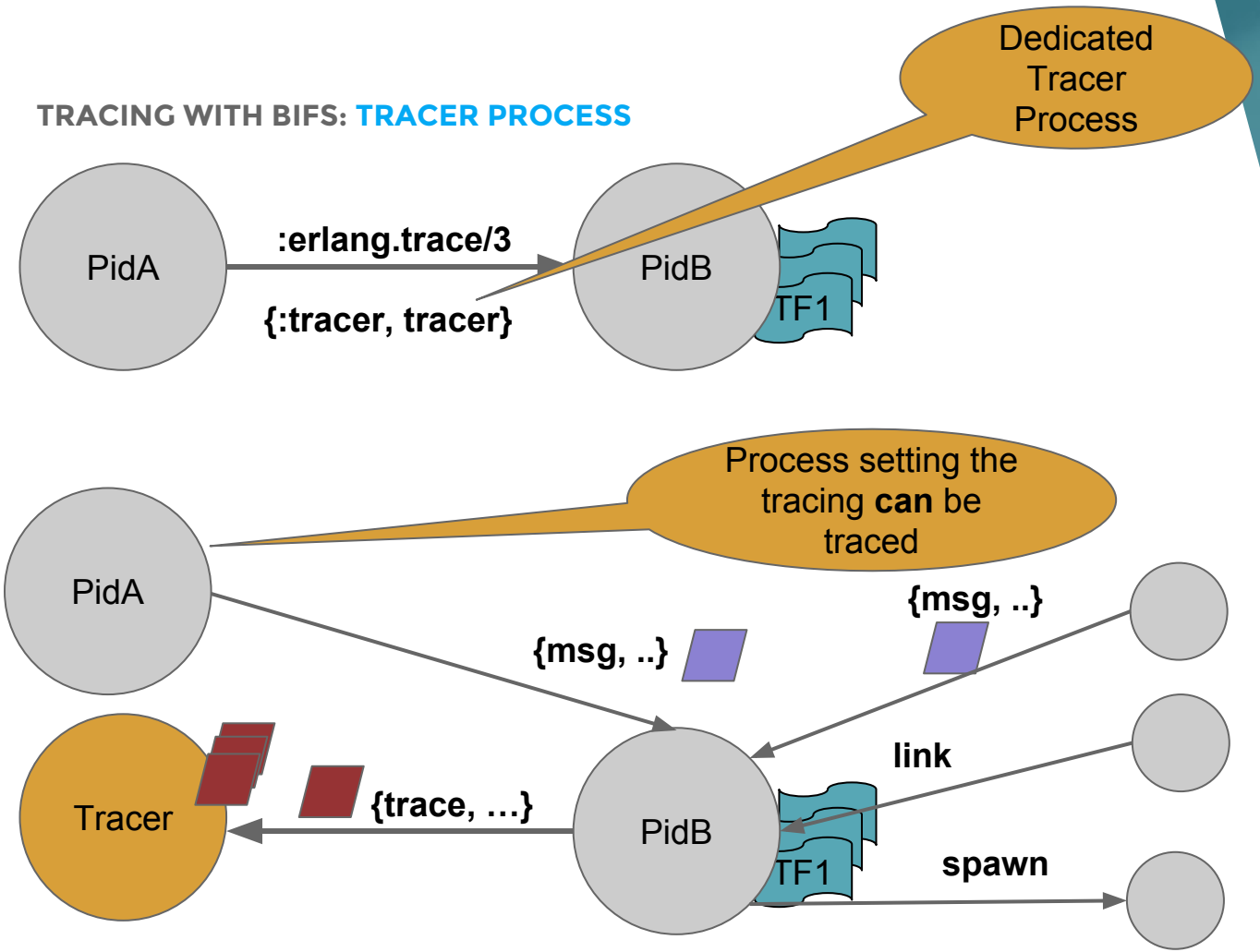


TRACING WITH BIFS: TRACER PROCESS

```
@doc """
Enables tracing of the commands sent to the KVServer
"""
def trace_commands(tracer \\ self) do
  procs = :erlang.trace(:all, true, [:call, {:tracer, tracer}])
  ptrns = :erlang.trace_pattern({Command, :run, 1}, true, [])
  Logger.info "Matched #{procs} procs and #{ptrns} patterns"
end
```



TRACING WITH BIFS: TRACER PROCESS



TRACING WITH BIFS: MATCH SPECIFICATION

```
@doc """
Enables tracing of the commands sent to the KVServer and the
return values
"""
def trace_commands_and_returns(tracer \\ self) do
    procs = :erlang.trace(:all, true, [:call, {:tracer, tracer}])
    ptrns = :erlang.trace_pattern(
      _MFA = {KVServer.Command, :run, 1},
      _match_fun = [{_Head = :_,
                     _Conditions = [],
                     _Body = [{:return_trace}]}],
      _Flags = [])
    Logger.info "Matched #{procs} procs and #{ptrns} patterns"
  end
```

The Match Specification

TRACING WITH BIFS: MATCH SPECIFICATION

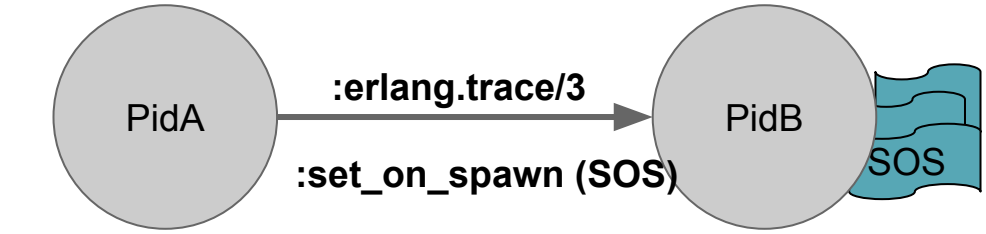
```
[{
  _bead = [{:"$1", :_, :_, :_}],
  _conditions = [{:==, :"$1", :put}],
  _body = [{:return_trace}]
}],
```

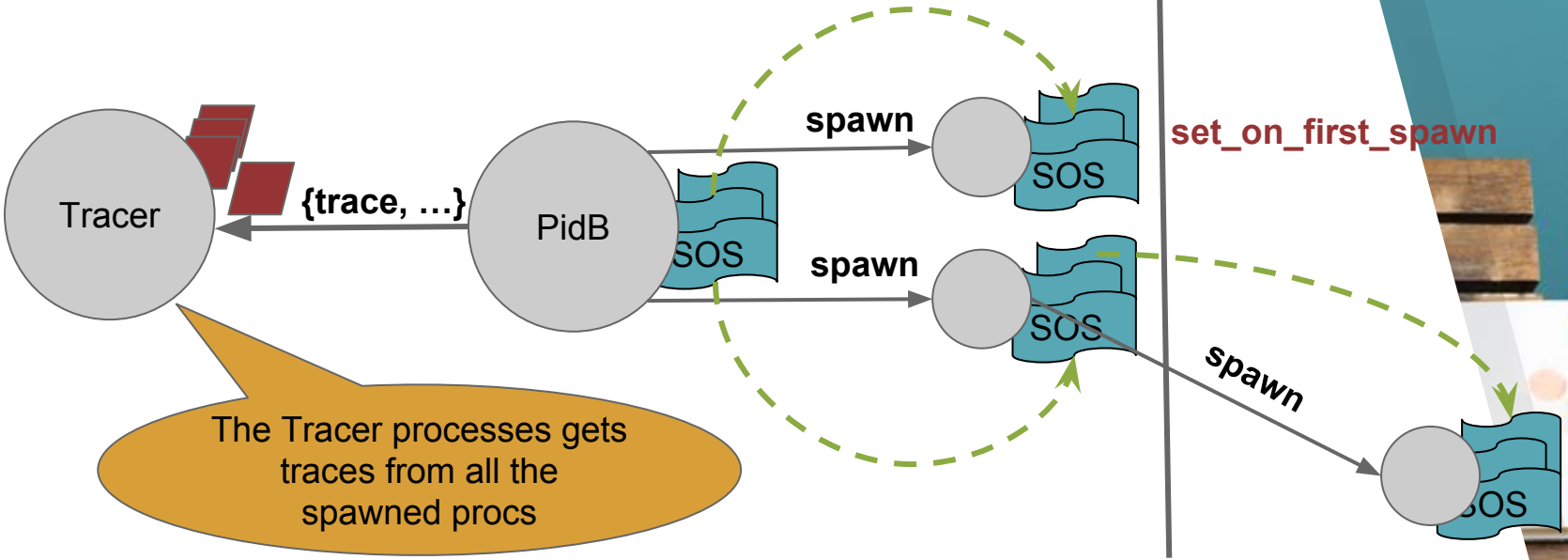
- 4-argument calls
- The first argument is the atom **:put**
- On match, return the result of the call (**:return_trace**)

TRACING WITH BIFS: TRACE FLAGS INHERITANCE

```
@doc """
Enables tracing of only those commands that are supervised
"""
def trace_supervised_commands(tracer \\ self) do
  procs = :erlang.trace(ExTrace.kvserver_task_supervisor_pid(),
    true, [:call, :set_on_spawn, {:tracer, tracer}])
  ptrns = :erlang.trace_pattern(
    _MFA = {KVServer.Command, :run, 1},
    _match_fun = [{_Head = :_,
                   _Conditions = [],
                   _Body = [{:return_trace}]}],
    _Flags = [])
  Logger.info "Matched #{procs} procs and #{ptrns} patterns"
end
```

TRACING WITH BIFS: TRACE FLAGS INHERITANCE





3.

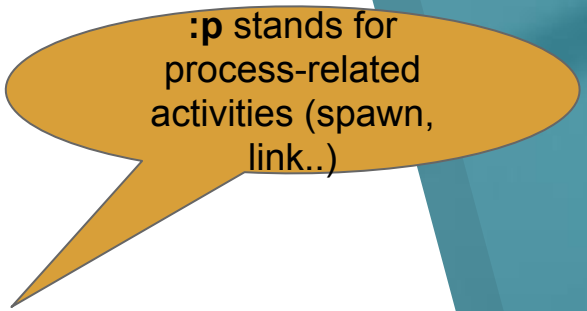
Mid Level Tracing with dbg

THE DBG

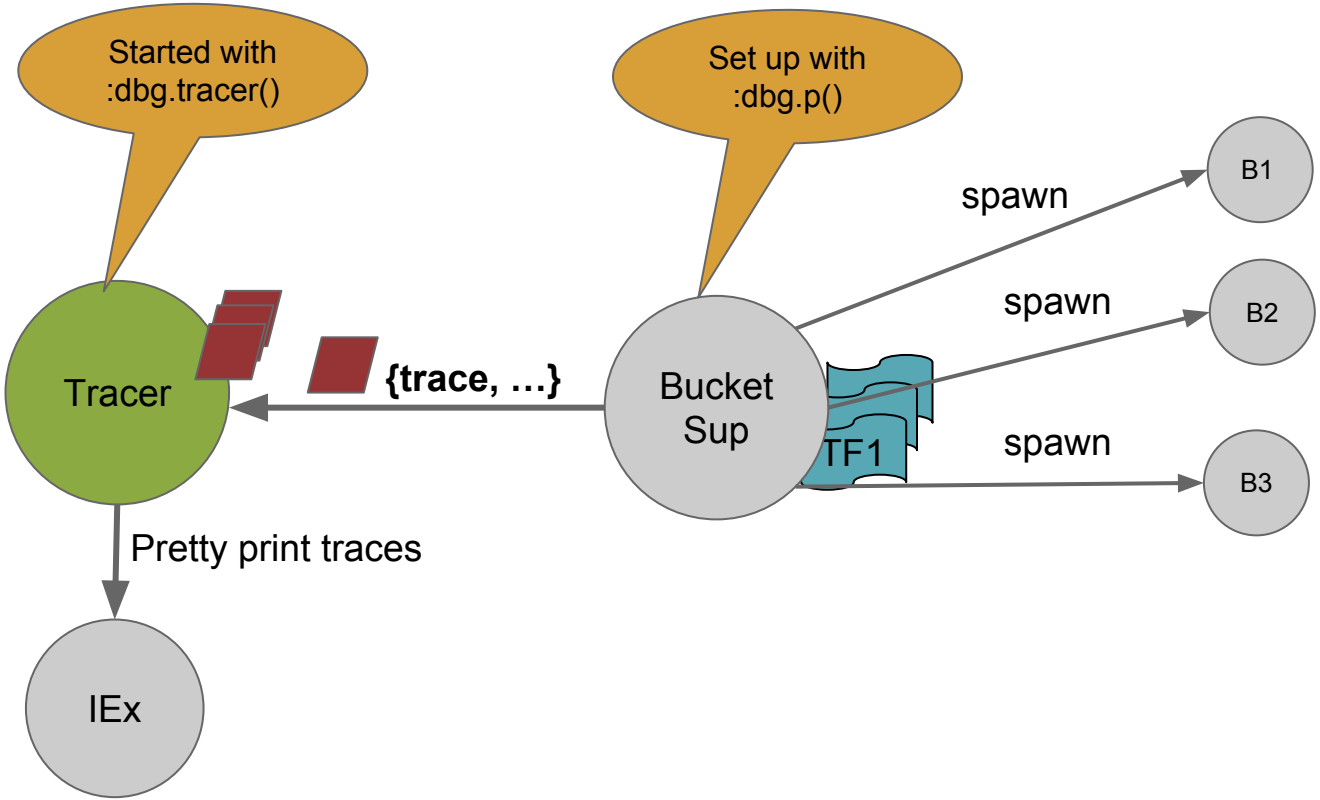
- The Text Based Trace Facility
- Interface to the :trace/3 and the :trace_pattern/2-3 BIFs
- Suitable on large systems

TRACING WITH DBG: SIMPLE EXAMPLES

```
@doc """
Enables tracing of the bucket processes
"""
def trace_buckets() do
    :dbg.tracer()
    :dbg.p(ExTrace.kv_bucket_supervisor_pid(), :p)
end
@doc """
Disables tracing of the bucket processes
"""
def stop_clear() do
    :dbg.p(:all, :clear)
    :dbg.stop_clear()
end
```



TRACING WITH DBG: SIMPLE EXAMPLES



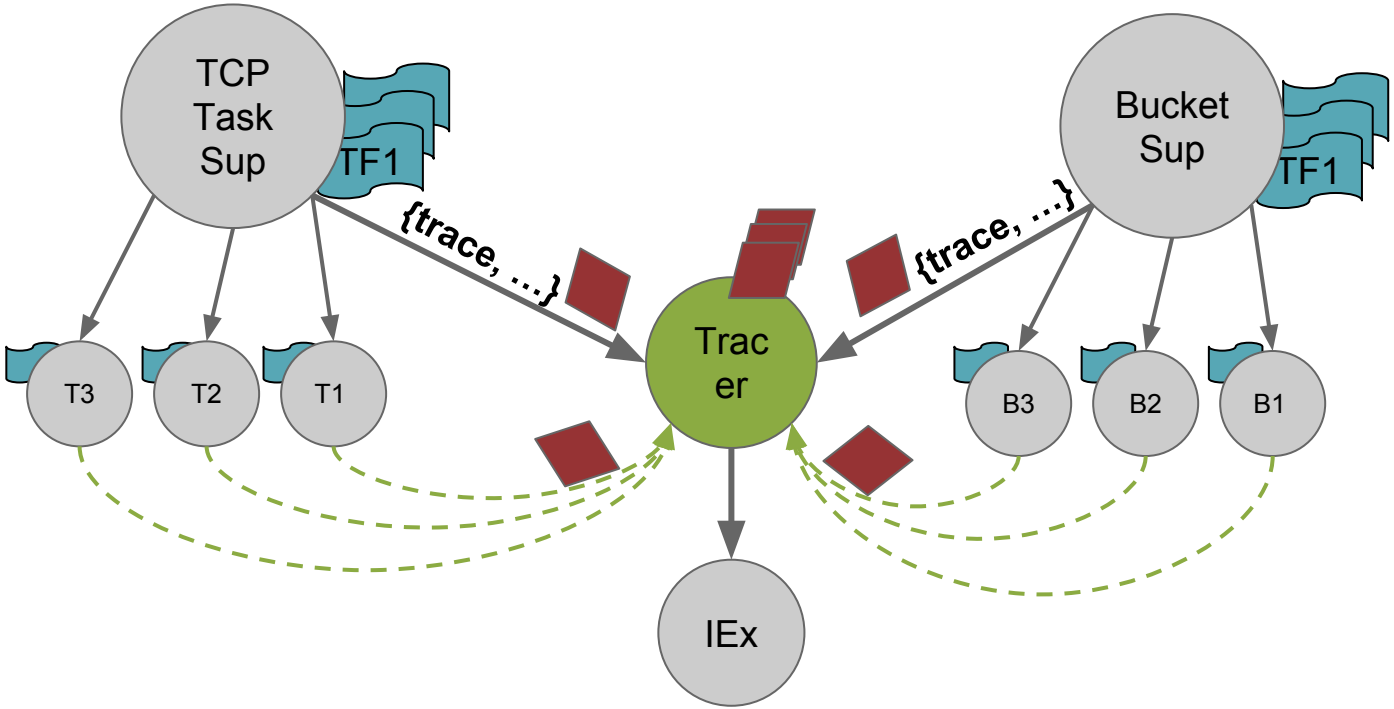
TRACING WITH DBG: SIMPLE EXAMPLES

```
@doc """
Enables tracing processes involved in handling the request from a
TPC Command up to the Bucket
"""
def trace_chain() do
    :dbg.tracer()
    {:dbg.p(ExTrace.kvserver_task_supervisor_pid,[:p, :sos]),
     :dbg.p(ExTrace.kv_bucket_supervisor_pid, [:p, :sos])}
end
```

:sos stands for **set_on_spawn**

KV.TaskSupervisor is for processes serving TCP requests

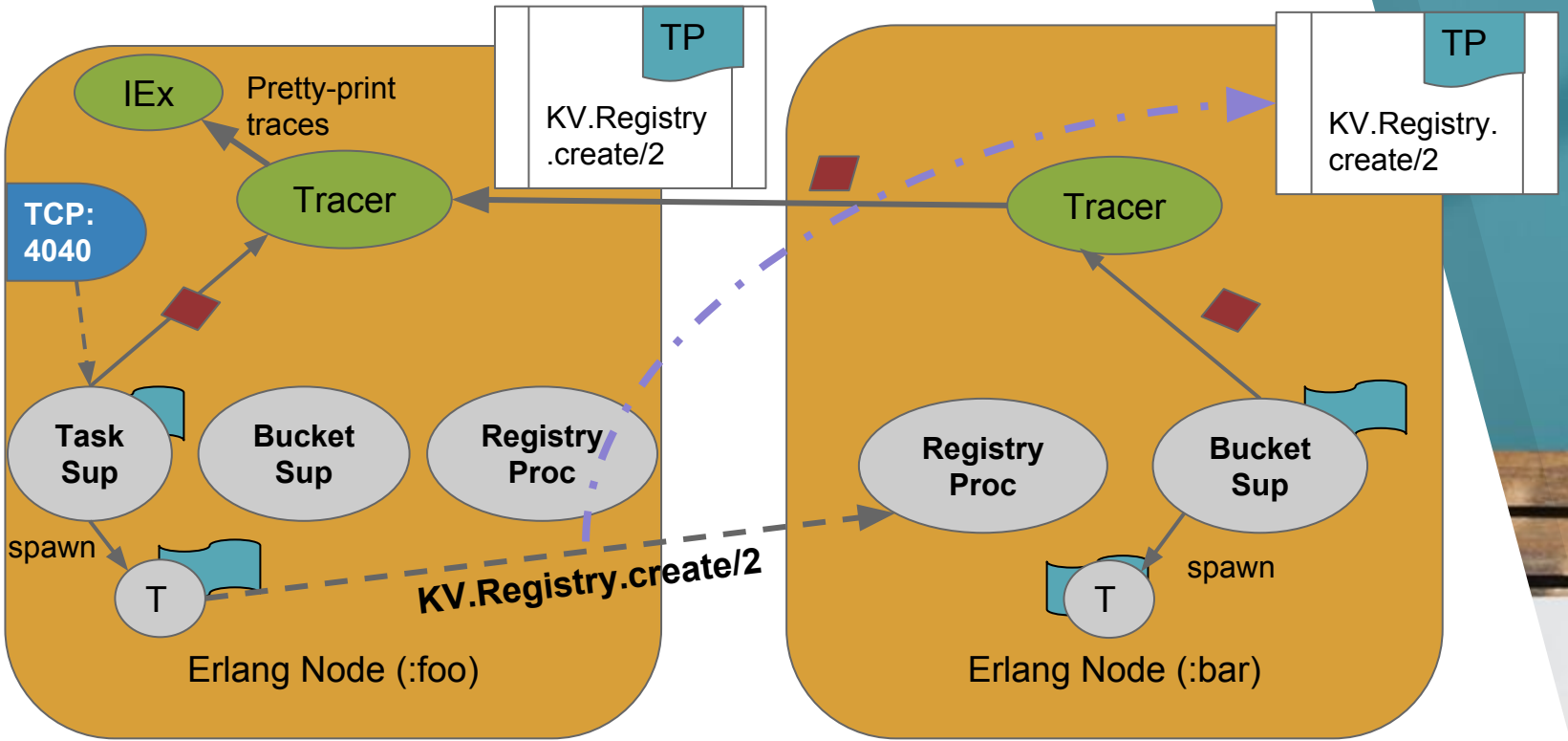
TRACING WITH DBG: SIMPLE EXAMPLES



TRACING WITH DBG: DISTRIBUTED EXAMPLE

```
@doc """
Enables tracing processes involved in handling the request from
a TPC Command up to the Bucket - regardless of the nodes they run on
"""
def trace_chain_distributed() do
    :dbg.tracer()
    true = for n <- nodes() do :dbg.n(n) end |> Enum.all?(&(elem(&1,1) == :ok))
    :dbg.p(:processes, :c)
    ms = [{
           [:_, :_],
           [],
           [{:enable_trace, KV.Bucket.Supervisor, :procs},
            {:enable_trace, KV.Bucket.Supervisor, :set_on_spawn}]
           }]
    {:dbg.p(ExTrace.kvserver_task_supervisor_pid,[:p, :sos]),
     :dbg.tpl(KV.Registry, :create, ms)}
end
```

TRACING WITH DBG: DISTRIBUTED EXAMPLES



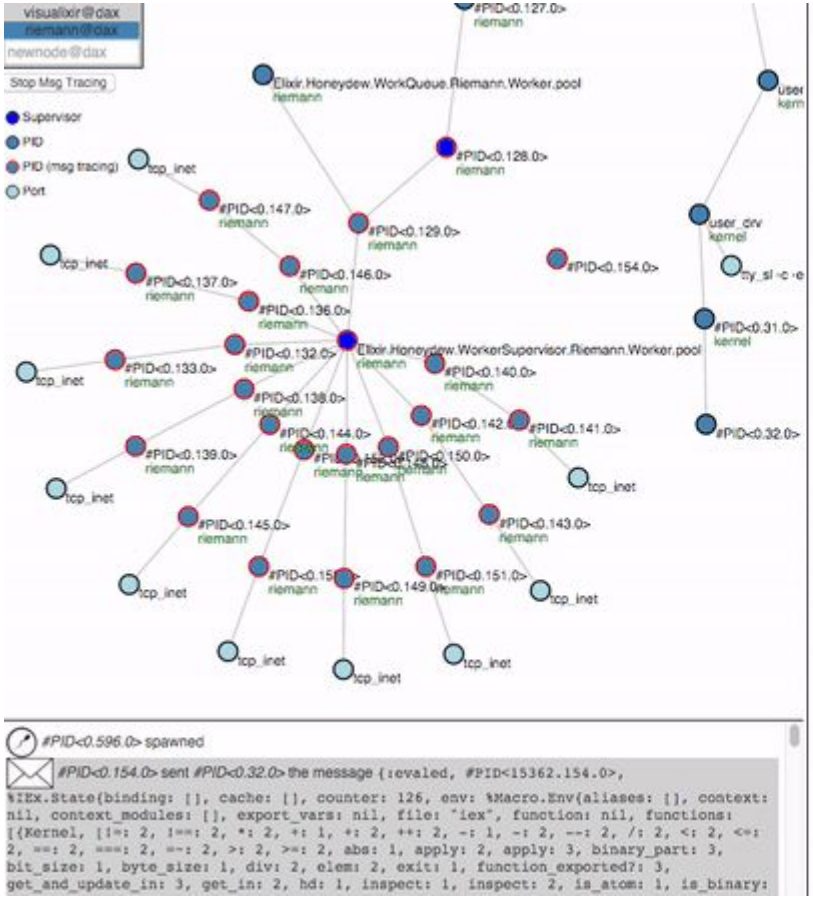
TRACING WITH DBG: TRACING TO A PORT

TODO

4.

High Level tracing with Visualixir

VISUALIXIR


visualixir@dax
riemann@dax
newnode@dax
Stop Msg Tracing
Supervisor
PID
PID (msg tracing)
Port
Elixir.Honeydew.WorkQueue.Riemann.Worker.pool
Elixir.Honeydew.WorkerSupervisor.Riemann.Worker.pool
#PID<0.127.0>
#PID<0.128.0>
#PID<0.129.0>
#PID<0.132.0>
#PID<0.133.0>
#PID<0.136.0>
#PID<0.137.0>
#PID<0.138.0>
#PID<0.139.0>
#PID<0.140.0>
#PID<0.141.0>
#PID<0.142.0>
#PID<0.143.0>
#PID<0.144.0>
#PID<0.145.0>
#PID<0.146.0>
#PID<0.147.0>
#PID<0.149.0>
#PID<0.150.0>
#PID<0.151.0>
#PID<0.154.0>
#PID<0.31.0>
#PID<0.32.0>
tcp_inet
#PID<0.596.0> spawned
#PID<0.154.0> sent #PID<0.32.0> the message {:evaled, #PID<15362.154.0>,
%IEx.State{binding: [], cache: [], counter: 126, env: %Macro.Env{aliases: [], context:
nil, context_modules: [], export_vars: nil, file: "iex", function: nil, functions:
[{Kernel, [!=: 2, !==: 2, *: 2, +: 1, +: 2, ++: 2, -: 1, -: 2, --: 2, /: 2, <: 2, <=:
2, ==: 2, ===: 2, =~: 2, >: 2, >=: 2, abs: 1, apply: 2, apply: 3, binary_part: 3,
bit_size: 1, byte_size: 1, div: 2, elem: 2, exit: 1, function_exported?: 3,
get_and_update_in: 3, get_in: 2, hd: 1, inspect: 1, inspect: 2, is_atom: 1, is_binary:

“

Leave your IO.puts “HEREEE!” behind, and start using **Erlang VM tracing facility.**

Erlang SOLUTIONS

THANK YOU

Szymon Mentel
szymon.mentel@erlang-solutions.com
www.erlang-solutions.com