MODELLING AND VERIFYING DISTRIBUTED APPLICATIONS WITH CONCUERROR

Stavros Aronis (@Vahnatai)

CODE BEAM SF

Erlang SOLUTIONS

THIS TALK

- Concurrency errors in Erlang
- Concuerror Basics
- Concuerror vs Distributed Applications
- **vnet**: a new modelling library
  - Highlights
  - Design
  - Implementation
  - Experiences

#CodeBEAMSF

CONCURRENCY ERRORS

CONCURRENCY ERRORS

- Scheduling dependent
- = not triggered in every execution
- Examples:
    - Bad synchronisation (e.g., “use before initialisation”)
    - Atomicity violations (e.g., x = x + 1)
    - Deadlocks

CONCURRENCY ERRORS IN ERLANG

- “Shared nothing” helps a lot
- However, sharing (must) exist:
  - Message passing (i.e., mailboxes)
    - Unexpected orderings
    - Unexpected timeouts
  - Global data (e.g., registry)
  - ETS tables
  - …

EXAMPLE

```
Child =
  spawn(
    fun() ->
      receive
        ok -> ok
      after
        100 -> timeout
      end
    end),
register(child, Child),
catch child ! ok.
```

EXAMPLE

```
Child =
  spawn(
    fun() ->
      receive
        ok -> ok
      after
        100 -> timeout
      end
    end),

timer:sleep(200),

register(child, Child),
catch child ! ok.
```

First attempt at async programming. - @jonathansampson (14 Dec 2015)

https://twitter.com/jonathansampson/status/676487374495342592

DEALING WITH CONCURRENCY ERRORS

- Let it crash?
- Debug “Heisenbugs”?
- Think hard?
- Try mathematical verification?
- Try “stress testing”?
- Try randomised testing?

- … how to ensure no errors remain?

SYSTEMATIC CONCURRENCY TESTING

- Explore **all** possible schedulings
- Systematically
- No errors found = **None existing**

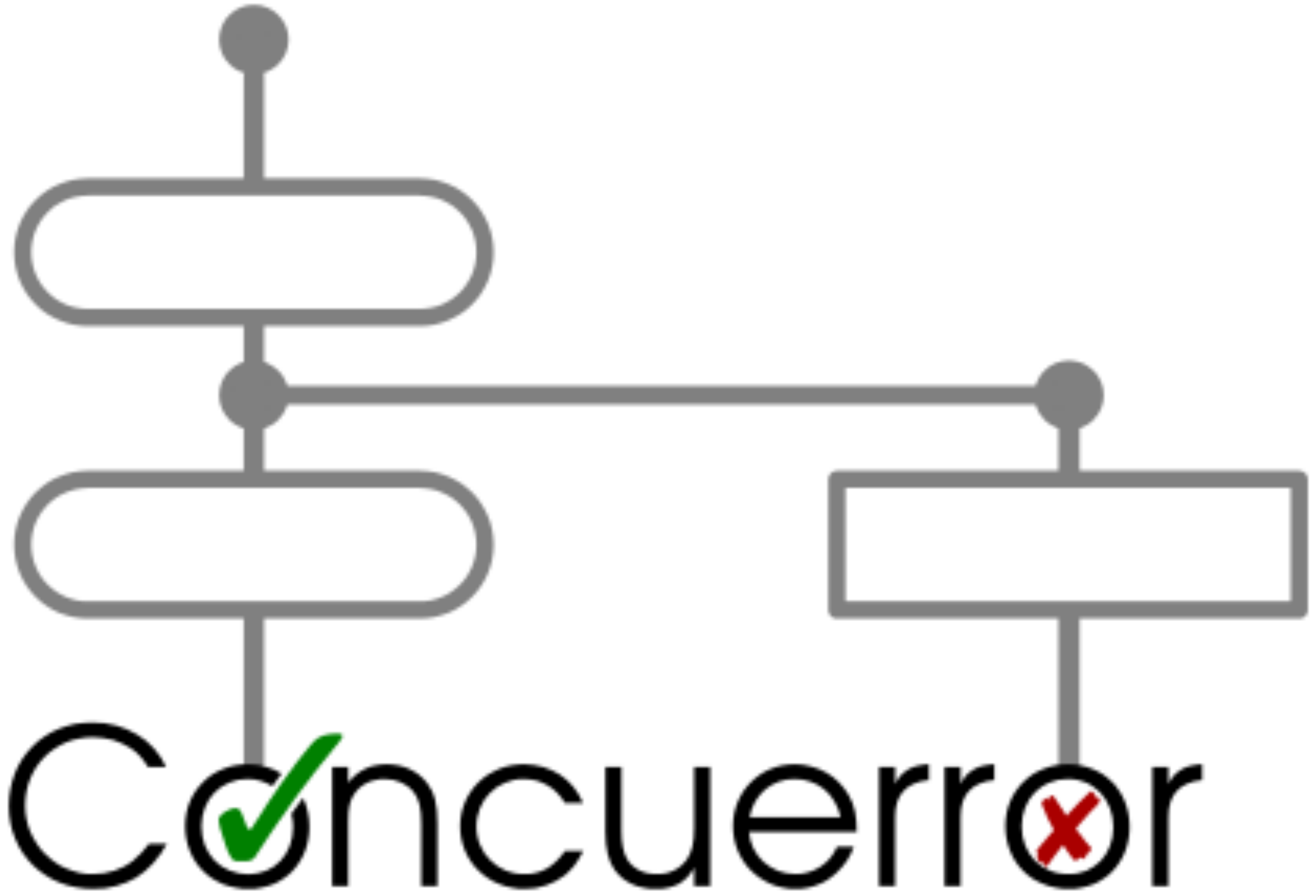
SYSTEMATIC CONCURRENCY TESTING

- Using a single ‘scheduler’
  - Execute an arbitrary (finite) scheduling
  - Check for errors
- Backtrack to latest “scheduling choice”
  - Pick a different scheduling
- Repeat until:
  - an error is found OR
  - **all choices have been explored**

#CodeBEAMSF

HTTPS://CONCUERROR.COM

CONCUERROR



HTTPS://CONCUERROR.COM

- is a tool for systematic concurrency testing
- is open source
- runs **tests** under all possible schedulings
  - … “intelligently”
- detects ‘abnormal’ process exits and deadlocks
  - … provides a corresponding trace

EXAMPLE / DEMO

```
Child =
  spawn(
    fun() ->
      receive
        ok -> ok
      after
        100 -> timeout
      end
    end),
register(child, Child),
catch child ! ok.
```

#CodeBEAMSF

CONCUERROR VS OTP

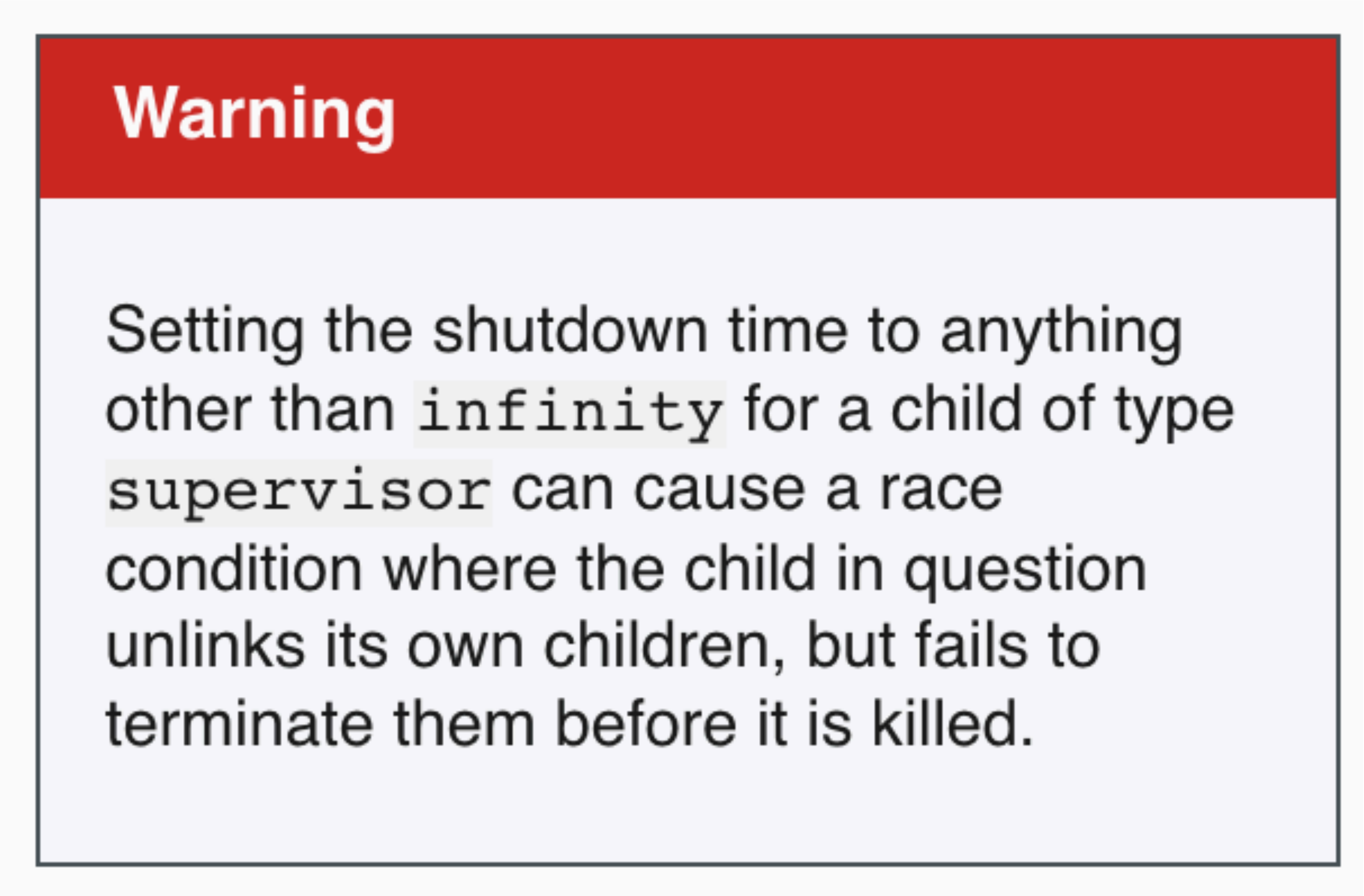
```
...
handle_call(stop, State) ->
  {stop, normal, ok, State).
...
```

```
gen_server:call(server,
stop),

gen_server:start({local,
server}, ...)
```

*… **gen_server:stop/1** added in OTP 18*

CONCUERROR VS OTP (ROUND 2)



Warning

Setting the shutdown time to anything other than `infinity` for a child of type `supervisor` can cause a race condition where the child in question unlinks its own children, but fails to terminate them before it is killed.

… ***warning added*** *in OTP 21.2 (Dec 12th, 2018)*
http://erlang.org/doc/man/supervisor.html

#CodeBEAMSF

THE CASE OF KRED/KDB

- OTP will (anyway) get you (really) far!
- However, sometimes you have more complex problems to solve

Klarna.

Smoooth payments.

DISTRIBUTED APPLICATIONS

THE CASE OF KRED/KDB

- Distributed system
- Built in-house
- Handling transactions
- Leader/follower-based replication

Klarna.

Smoooth payments.

#CodeBEAMSF

THE CASE OF KRED/KDB

- Concurrency errors related to distribution
- Review / redesign
  - Work by **Viktória Fördős** and **Dániel Szoboszlay**
- Prototype new ideas
- Engineers could prove correctness…
- … or have some fun instead!

Klarna.

Smoooth payments.

#CodeBEAMSF

DISTRIBUTED APPLICATIONS

- Erlang’s built-in ops are “transparent”
  - Message passing behaves similarly
  - Processes behave similarly
  - Registry not straightforward due to name clashes
- Additional sources of errors:
  - Node crashing
  - Node disconnects

LET'S CONCUERROR?

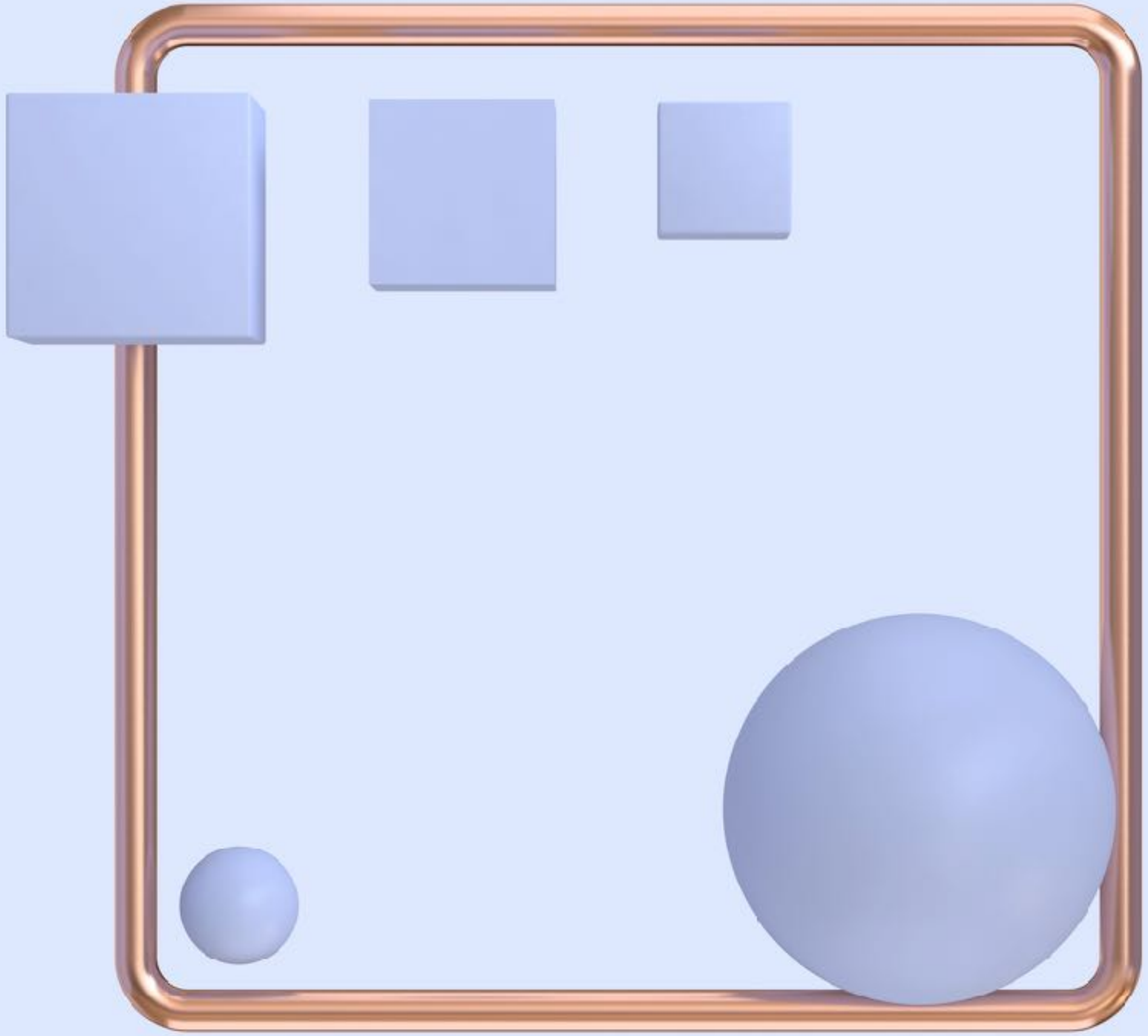
Unfortunately not…

CONCUERROR VS DISTRIBUTED APPLICATIONS

- (Currently) supports ONLY single-node
- Extending Concuerror is difficult
- Tricky to use on “production” code

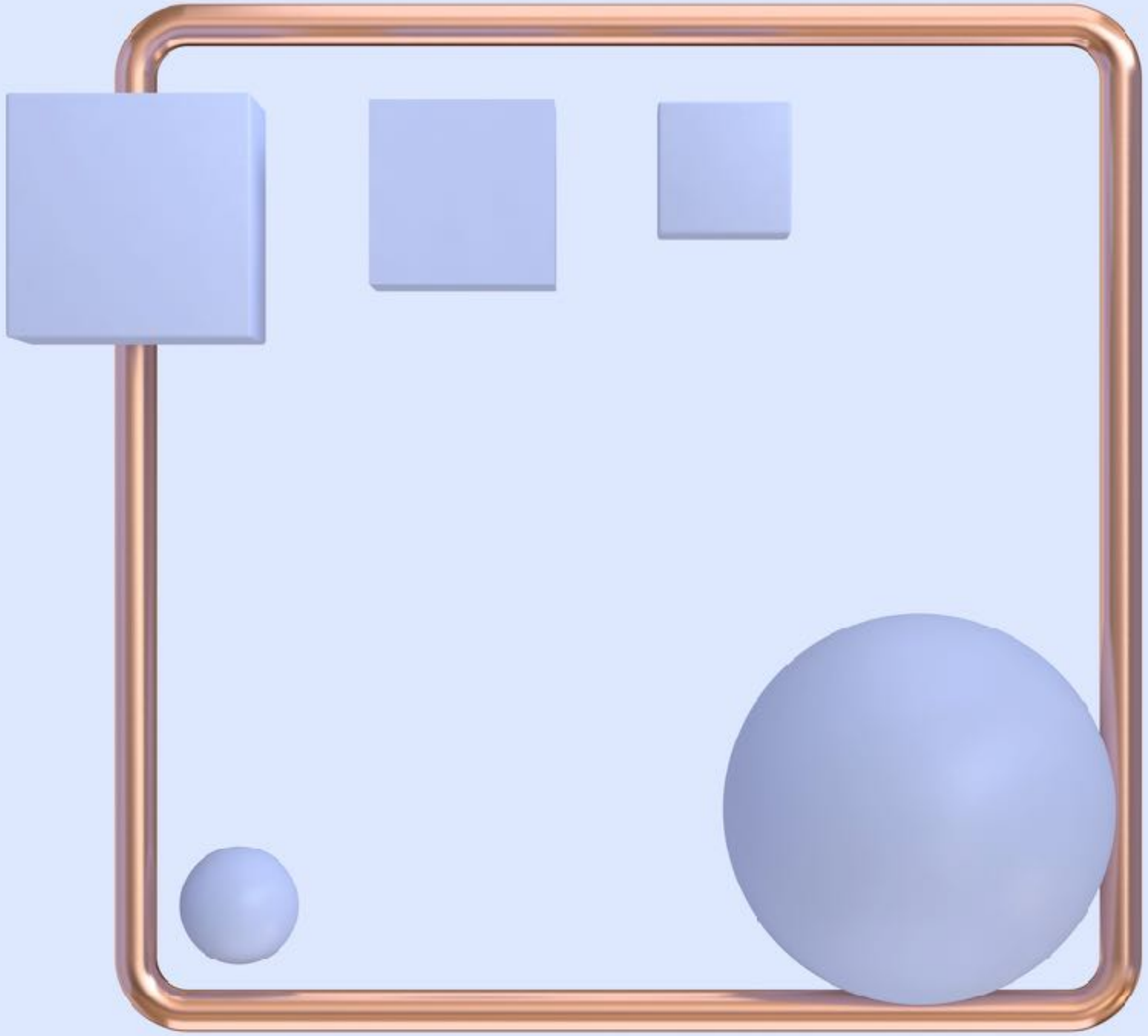
- Lets try something different…

HTTPS://GITHUB.COM/KLARNA/ VNET



VNET: HIGHLIGHTS

- Is a modelling library
- Is open source
  - https://github.com/klarna/vnet
- Was presented in Erlang Workshop 2018
  - https://concuerror.com/publications



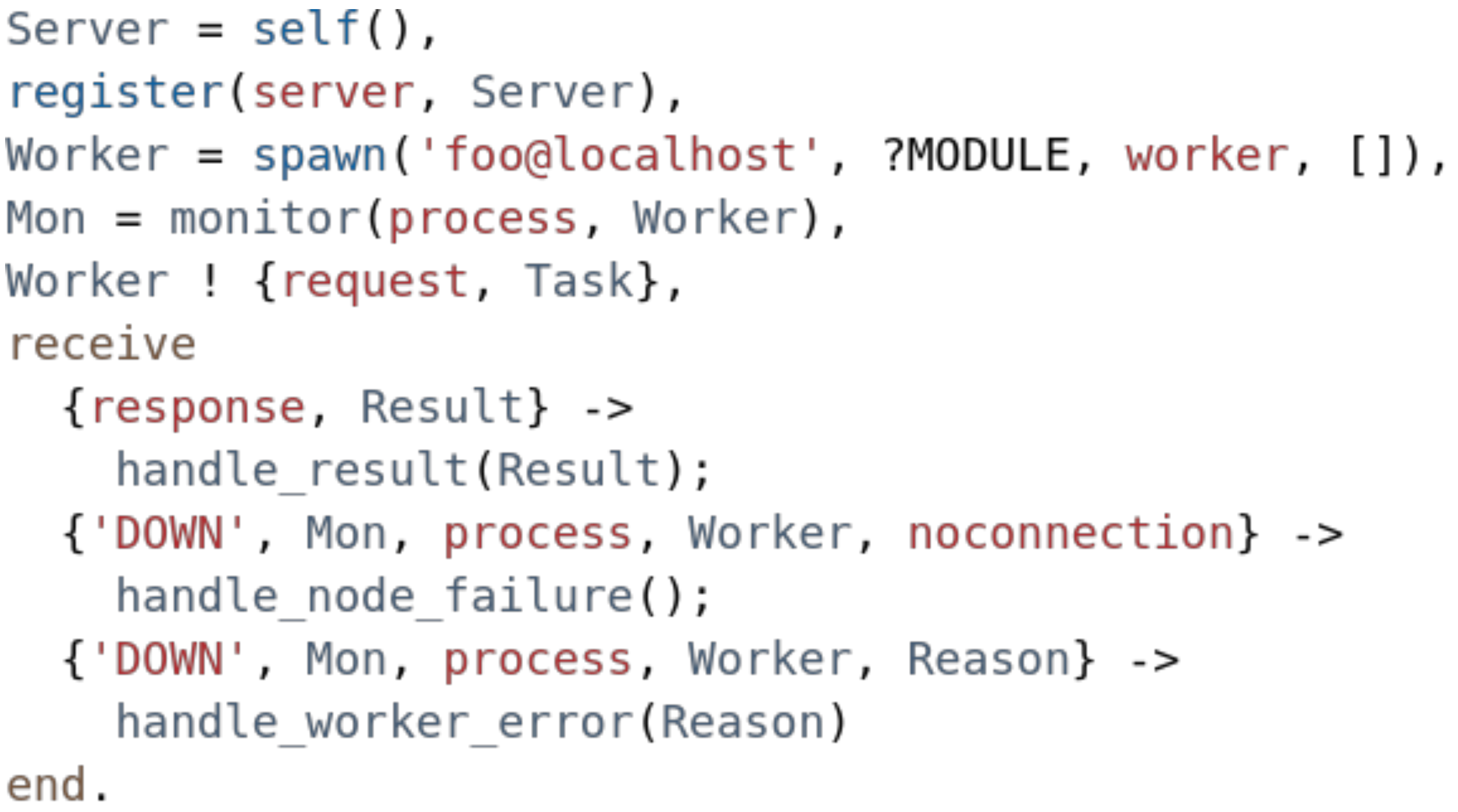
VNET: HIGHLIGHTS (MORE)

- Enables testing/verification of distributed applications with single-node tools
- Can simulate node crashes and disconnections
- Is compatible with OTP behaviours
  - Most Erlang built-in ops work “as is”
  - Registry via… `{via, vnet, Name}`

CODE BEAM SF Erlang SOLUTIONS

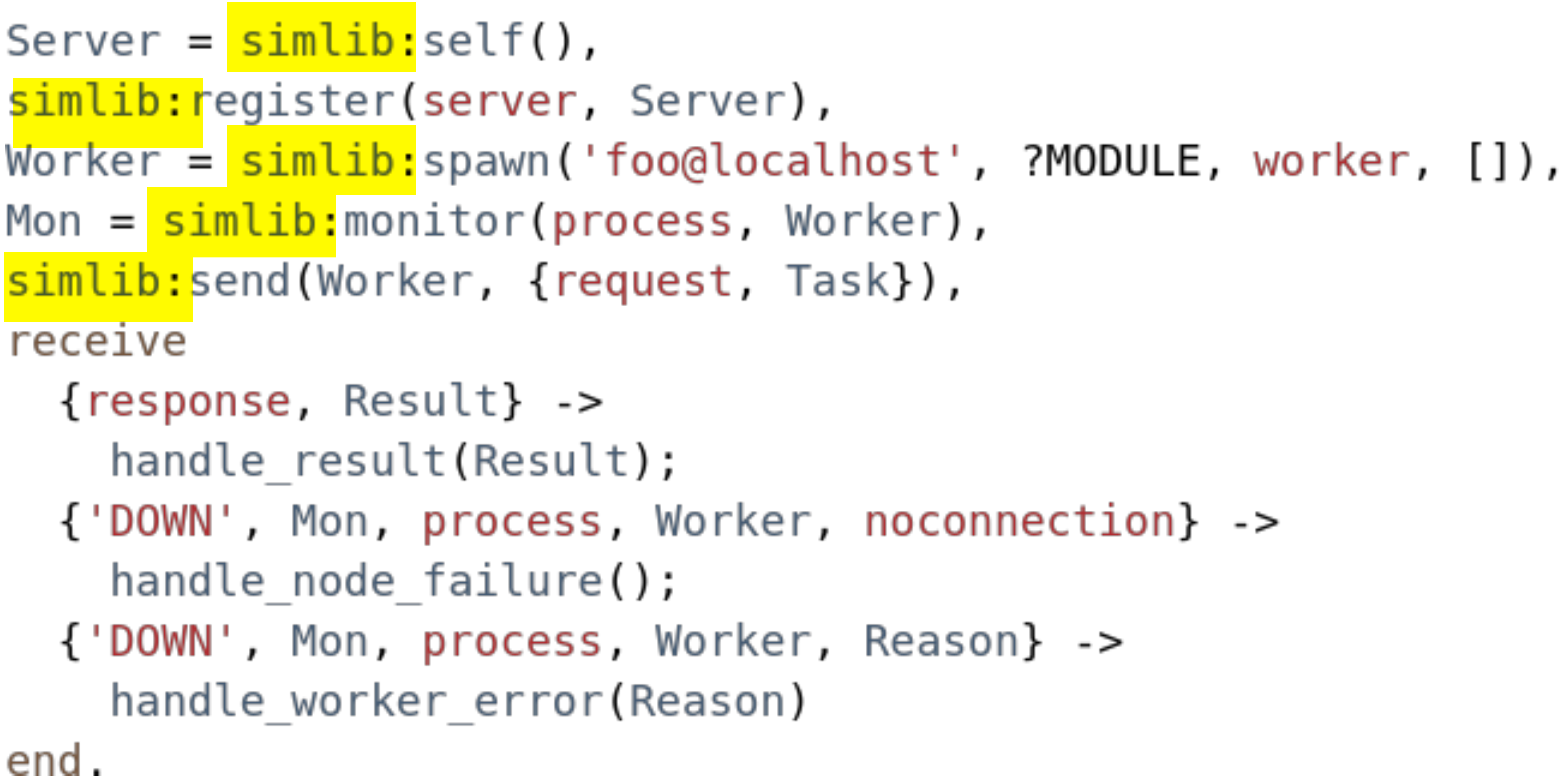
#CodeBEAMSF

VNET: DESIGN



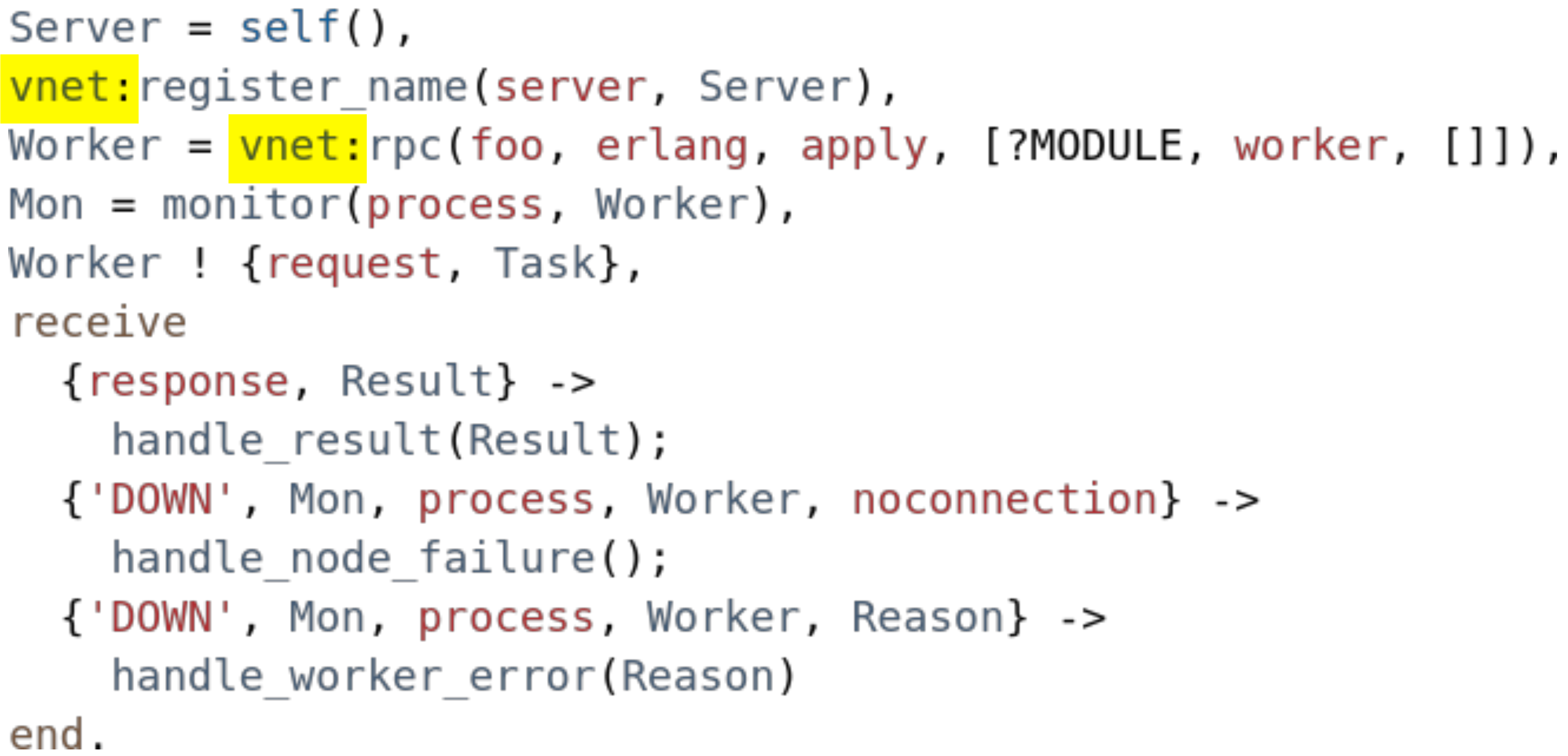
```
Server = self(),
register(server, Server),
Worker = spawn('foo@localhost', ?MODULE, worker, []),
Mon = monitor(process, Worker),
Worker ! {request, Task},
receive
  {response, Result} ->
    handle_result(Result);
  {'DOWN', Mon, process, Worker, noconnection} ->
    handle_node_failure();
  {'DOWN', Mon, process, Worker, Reason} ->
    handle_worker_error(Reason)
end.
```

VNET: DESIGN

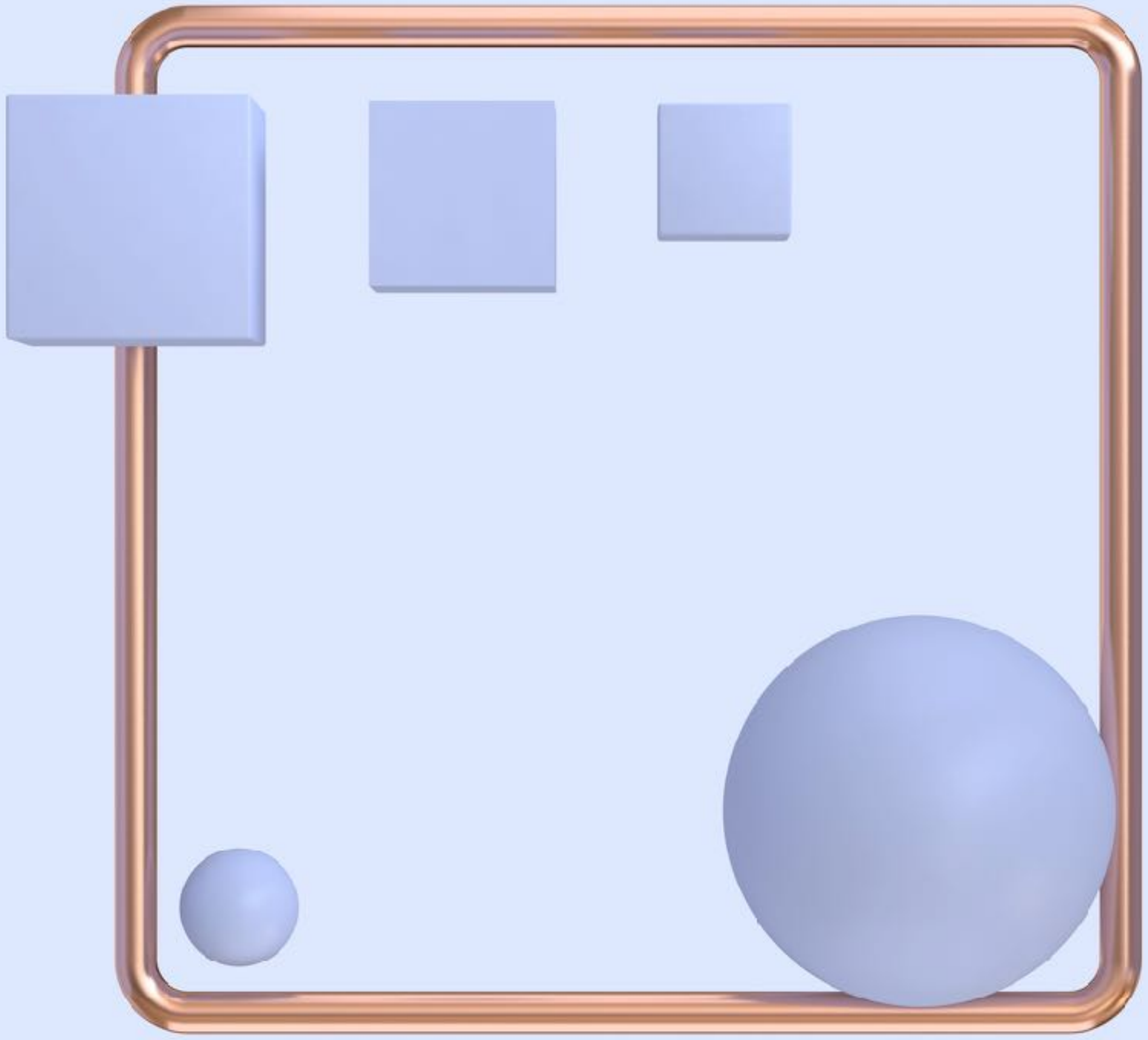


```
Server = simlib:self(),
simlib:register(server, Server),
Worker = simlib:spawn('foo@localhost', ?MODULE, worker, []),
Mon = simlib:monitor(process, Worker),
simlib:send(Worker, {request, Task}),
receive
  {response, Result} ->
    handle_result(Result);
  {'DOWN', Mon, process, Worker, noconnection} ->
    handle_node_failure();
  {'DOWN', Mon, process, Worker, Reason} ->
    handle_worker_error(Reason)
end.
```

VNET: DESIGN

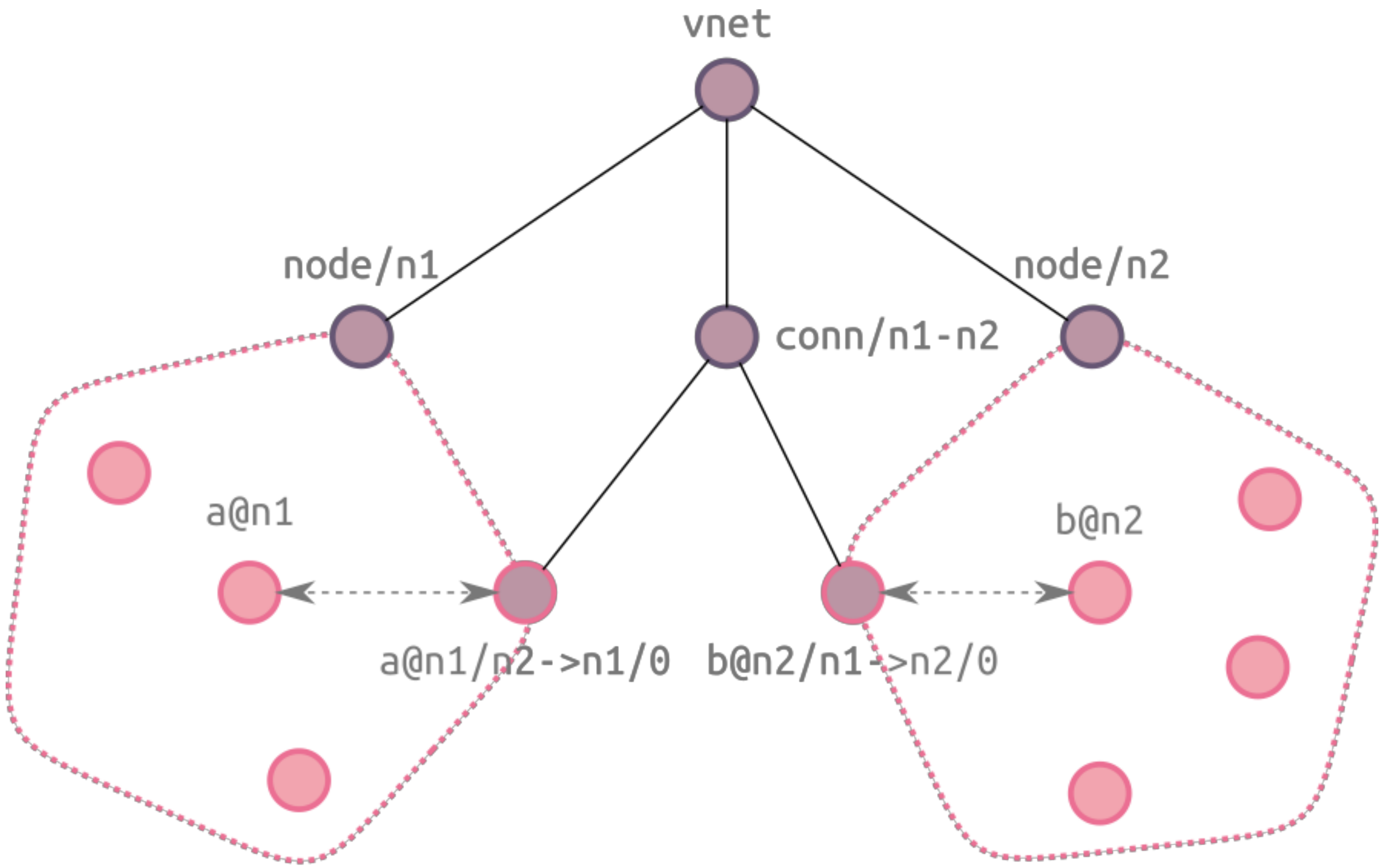


```
Server = self(),
vnet:register_name(server, Server),
Worker = vnet:rpc(foo, erlang, apply, [?MODULE, worker, []]),
Mon = monitor(process, Worker),
Worker ! {request, Task},
receive
  {response, Result} ->
    handle_result(Result);
  {'DOWN', Mon, process, Worker, noconnection} ->
    handle_node_failure();
  {'DOWN', Mon, process, Worker, Reason} ->
    handle_worker_error(Reason)
end.
```



VNET: DESIGN

- Allows use of OTP behaviours
- Allows controlling connections
- Handles registry name clashes

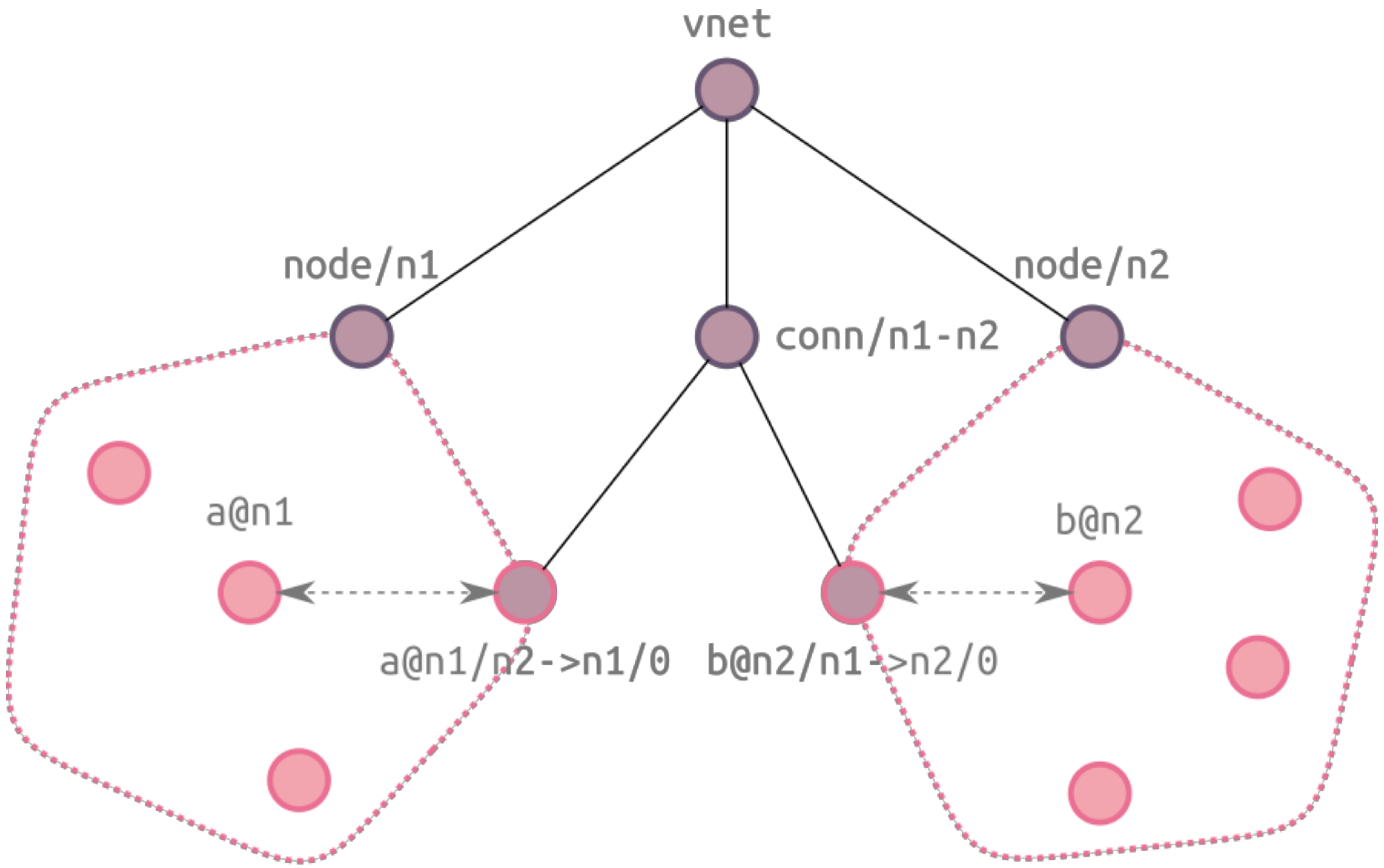


VNET: IMPLEMENTATION

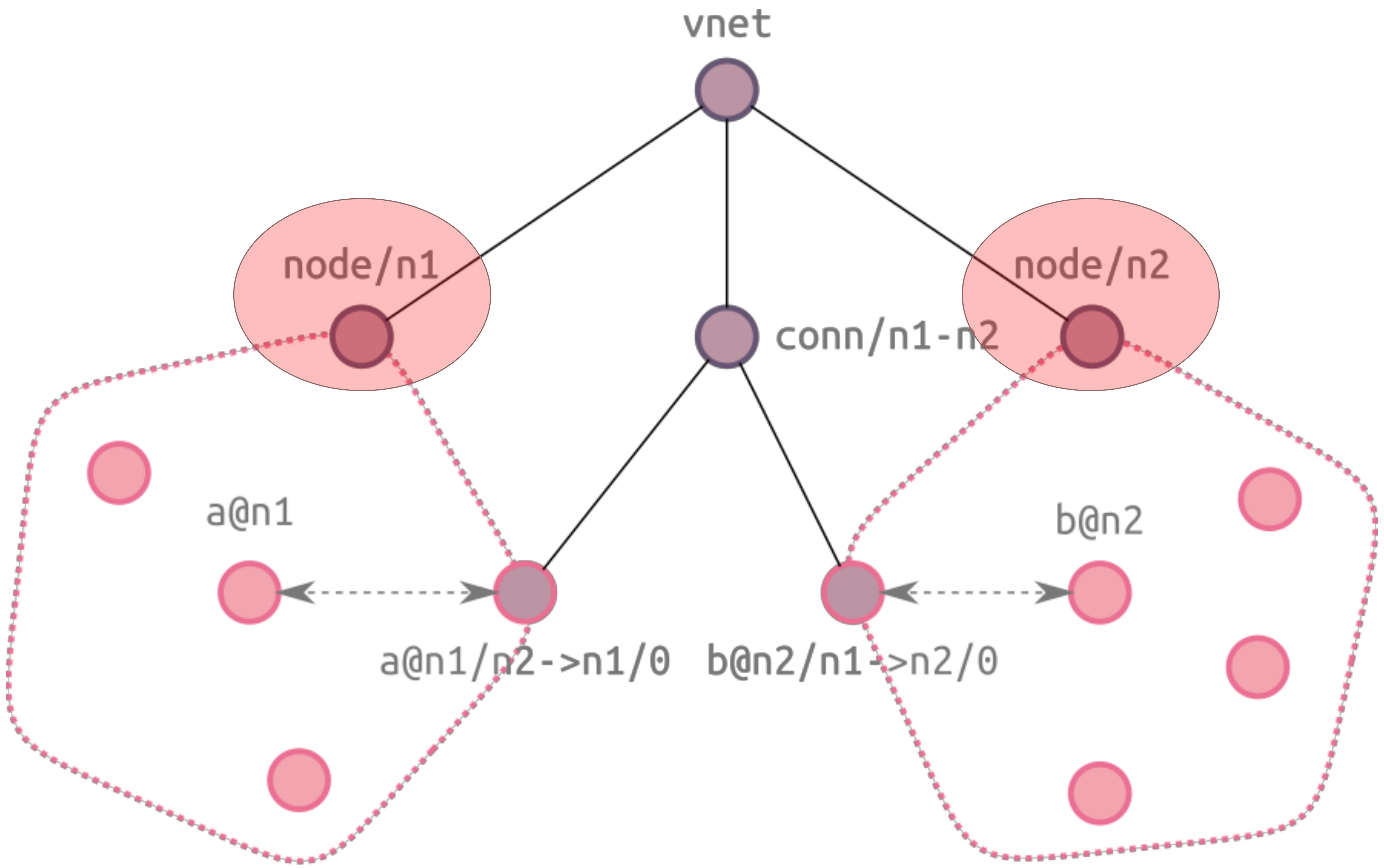
1. Custom name registry
2. vnode processes
3. connection processes
4. proxy processes

VNET: CUSTOM NAME REGISTRY

- Supporting the “via” mechanism
- `<name> becomes <name>@<vnode>`
- `vnet:tab/2` for ETS table names



#CodeBEAMSF

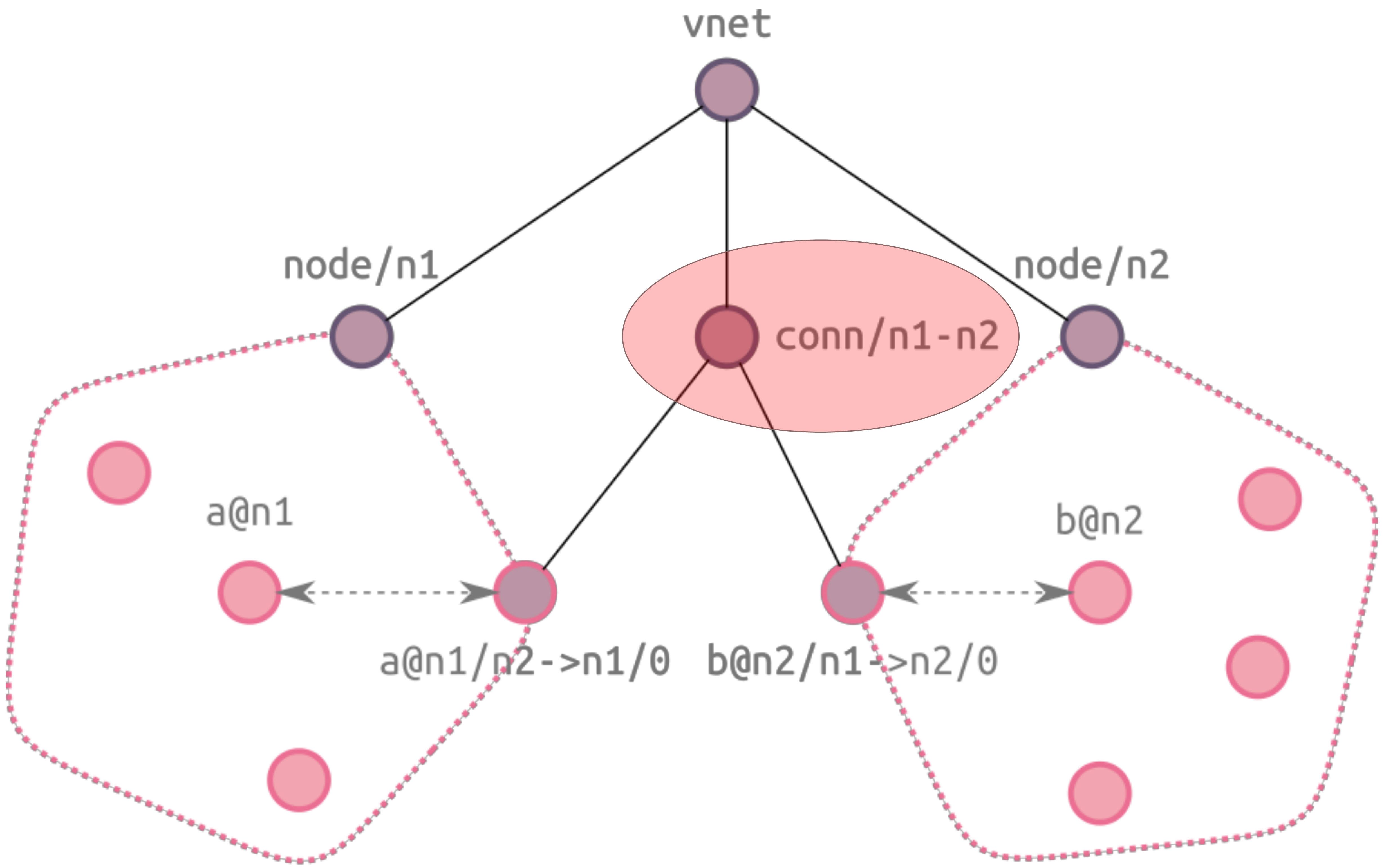


VNET: VNODE PROCESSES

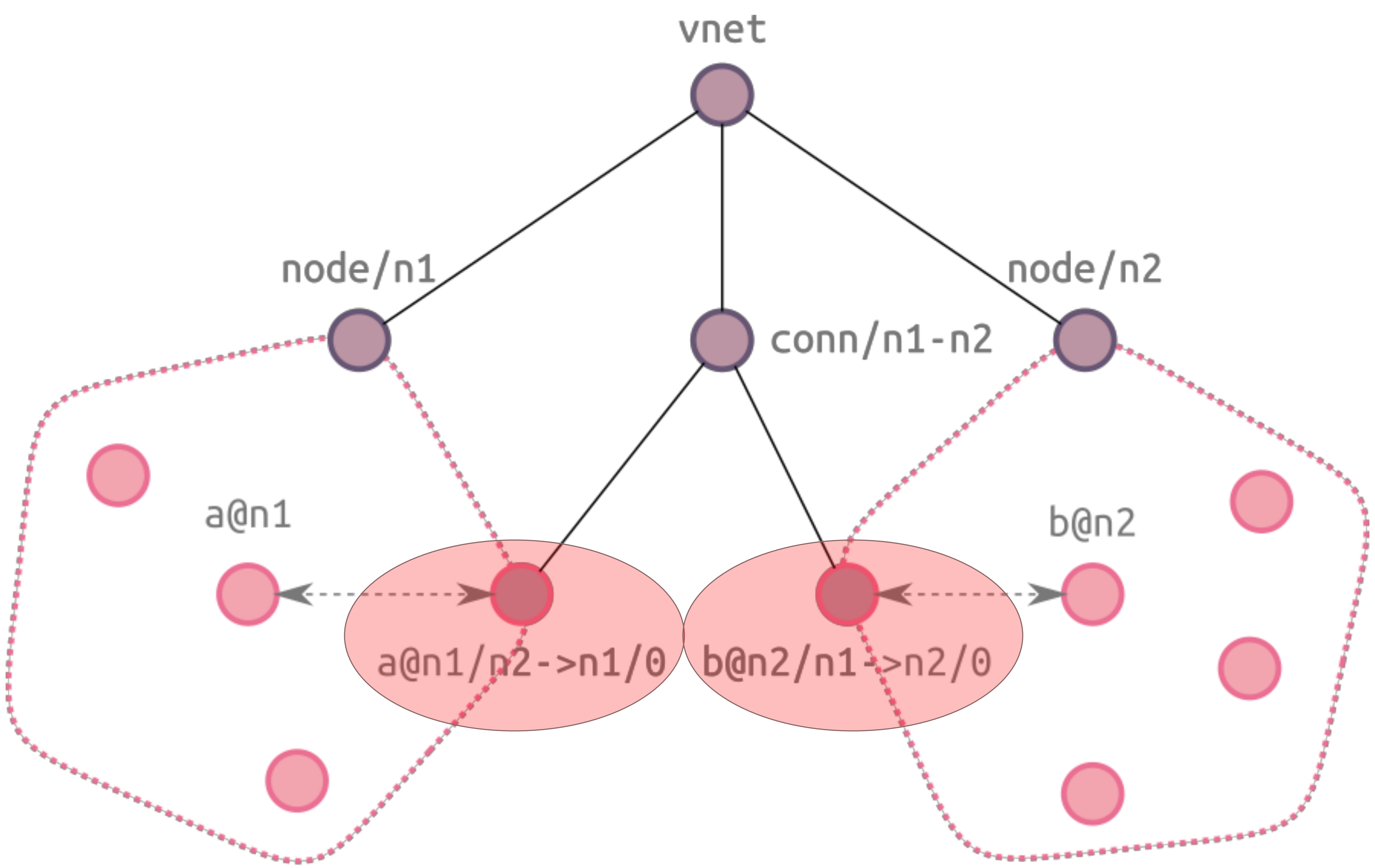
- Group leader of processes in a node
  - Inherited on spawn
  - Marks processes belonging to node
- Kill “node’s” processes if node goes down

VNET: CONNECTION PROCESSES

- One per connected node pair
- Control connect/disconnect scenarios
- Responsible for proxy processes



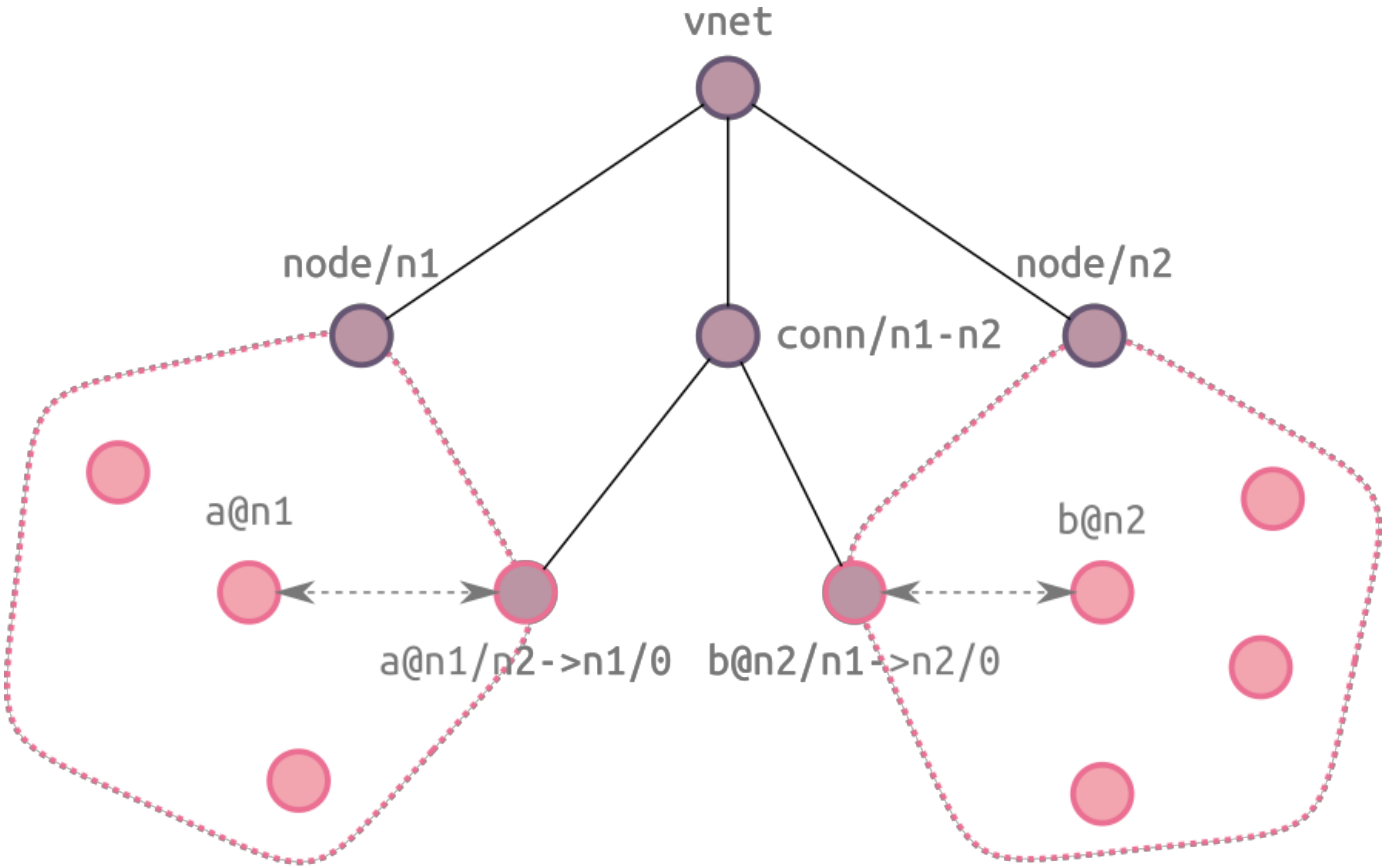
#CodeBEAMSF



VNET: PROXY PROCESSES

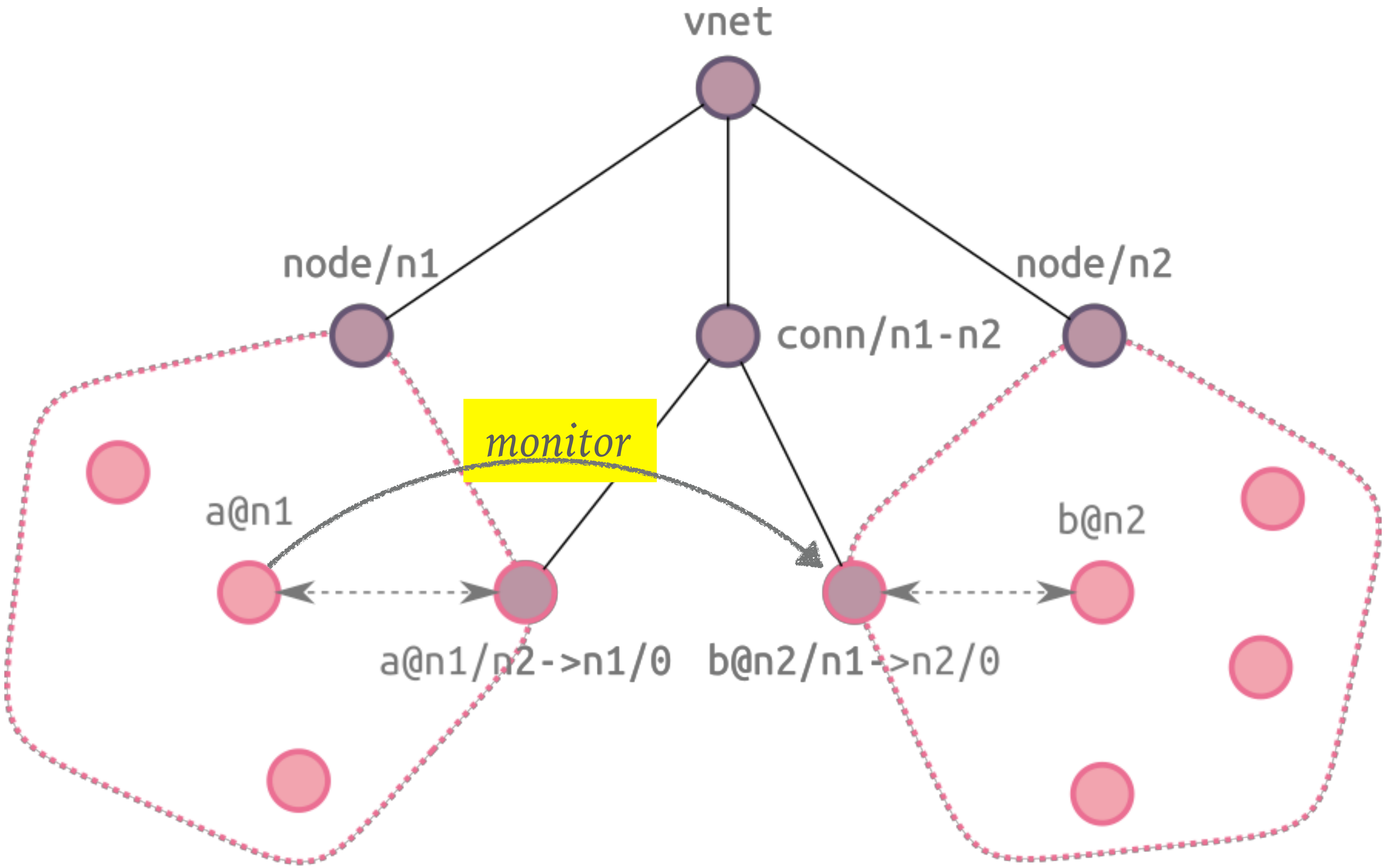
- … where all the magic happens!
- One per process, connection & direction
  - On demand!
- Each proxy process:
  - … proxies a process with regard to a connected node
  - Acts as target of remote links, monitors & messages
  - Inspects/rewrites messages perhaps replacing PIDs with suitable proxies

VNET: EXAMPLE

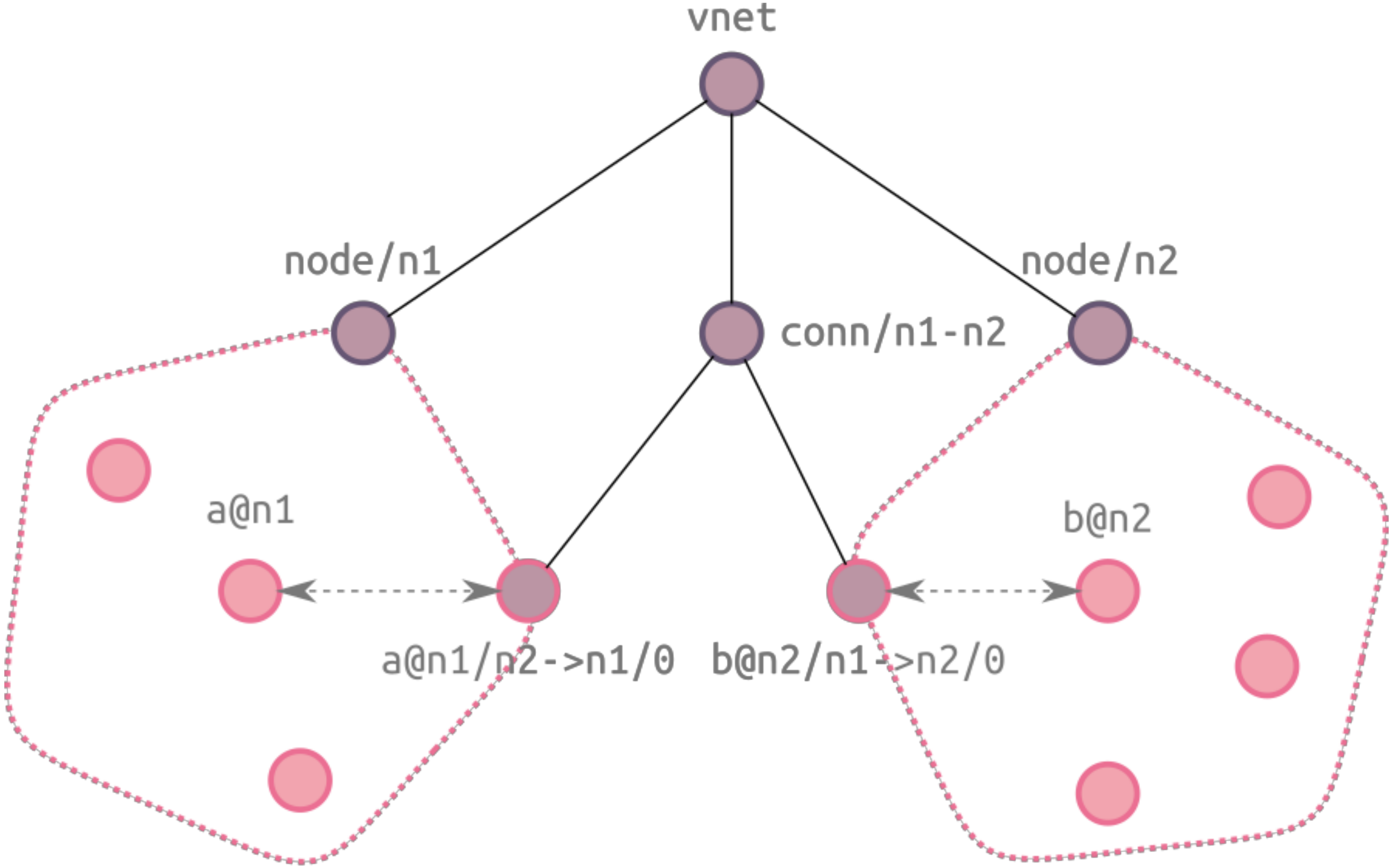


#CodeBEAMSF

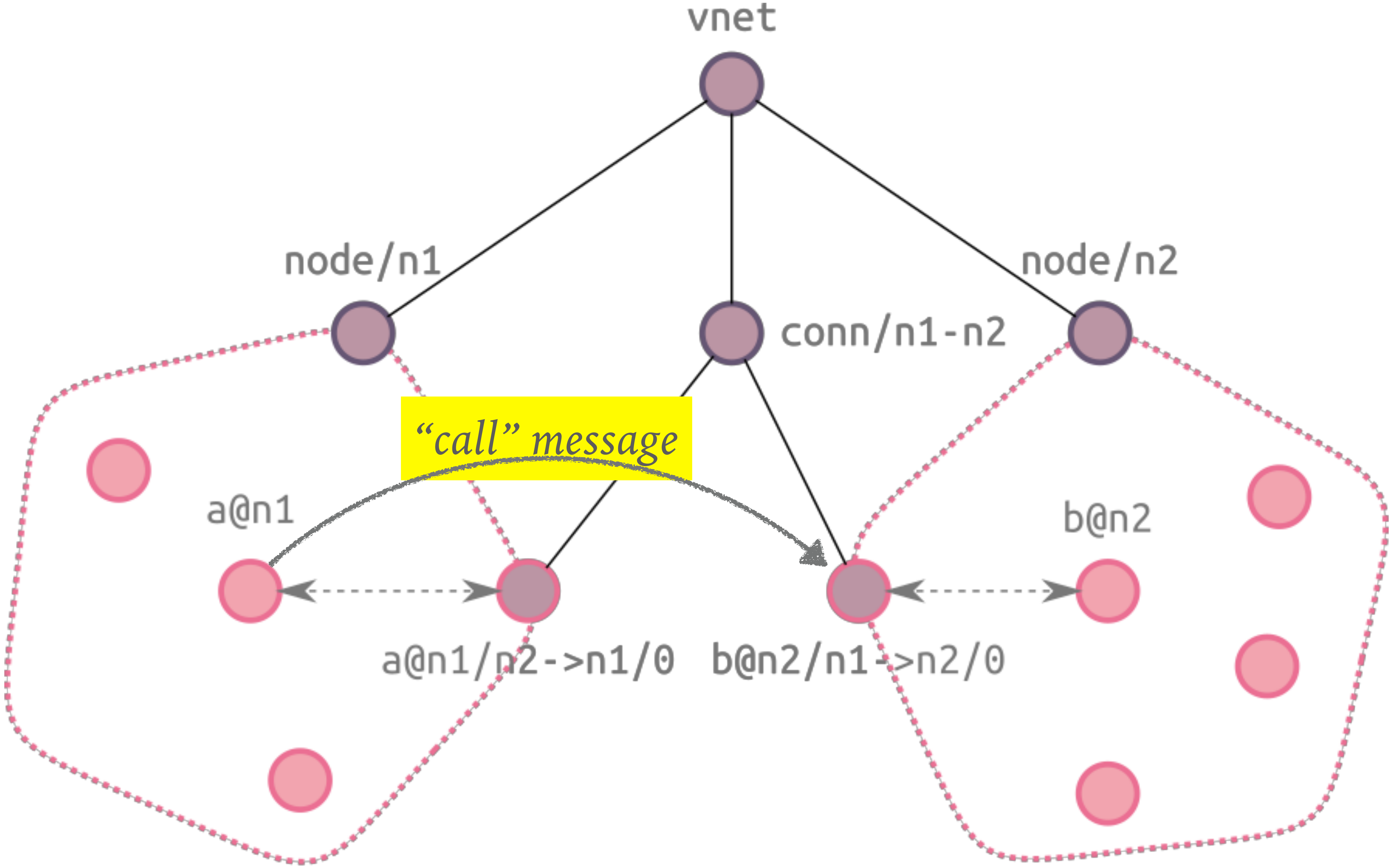
VNET: EXAMPLE



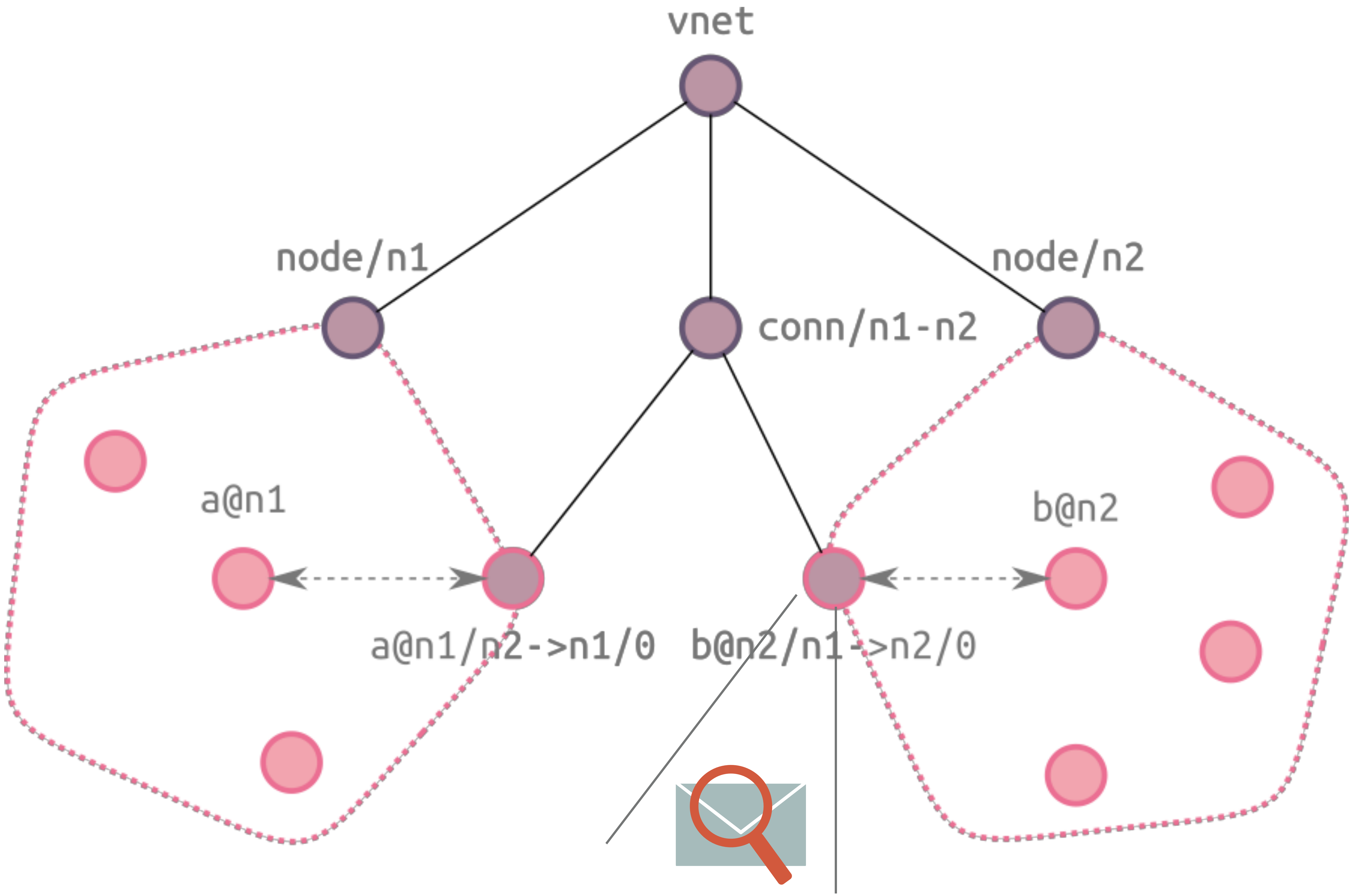
VNET: EXAMPLE



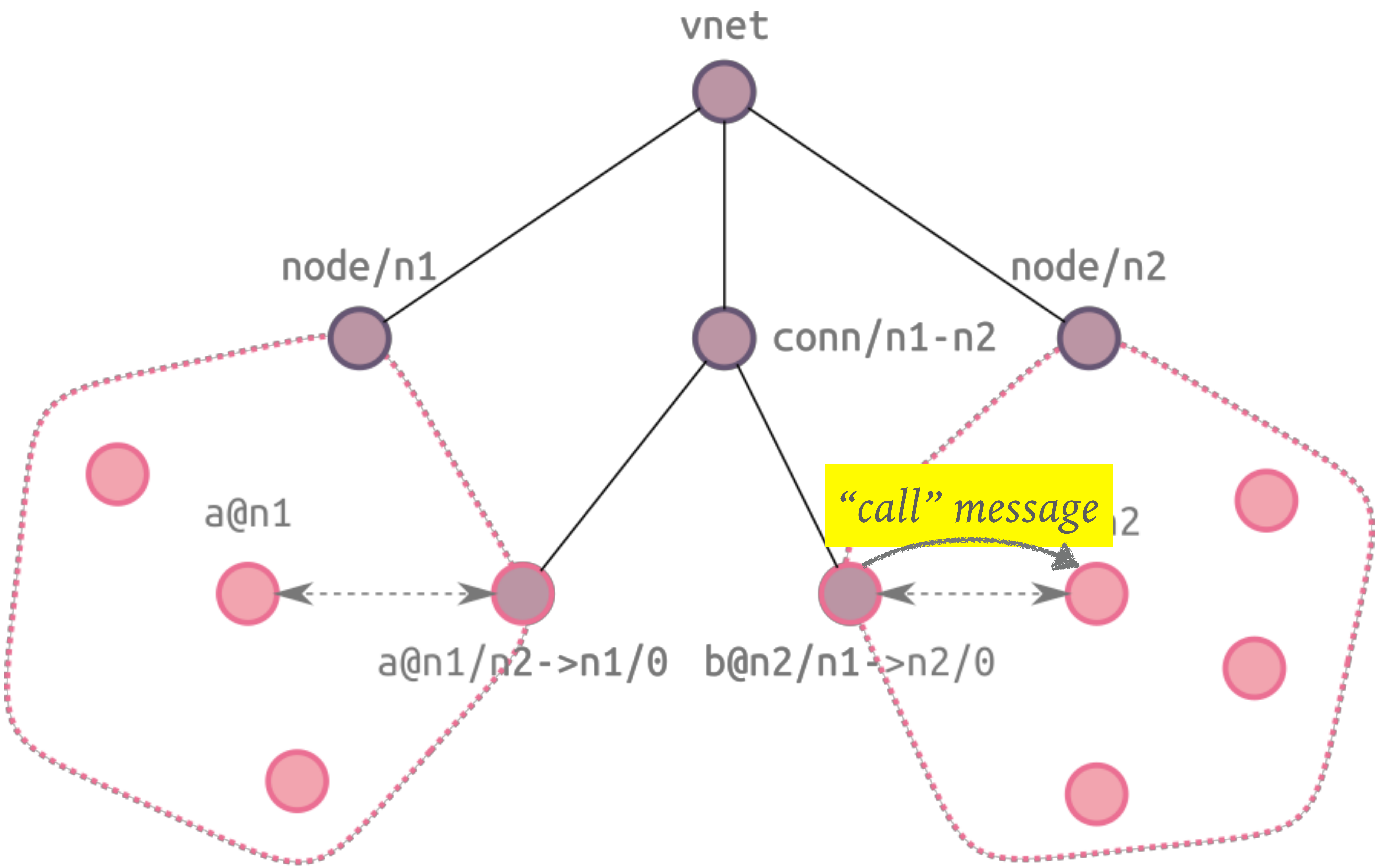
VNET: EXAMPLE



VNET: EXAMPLE

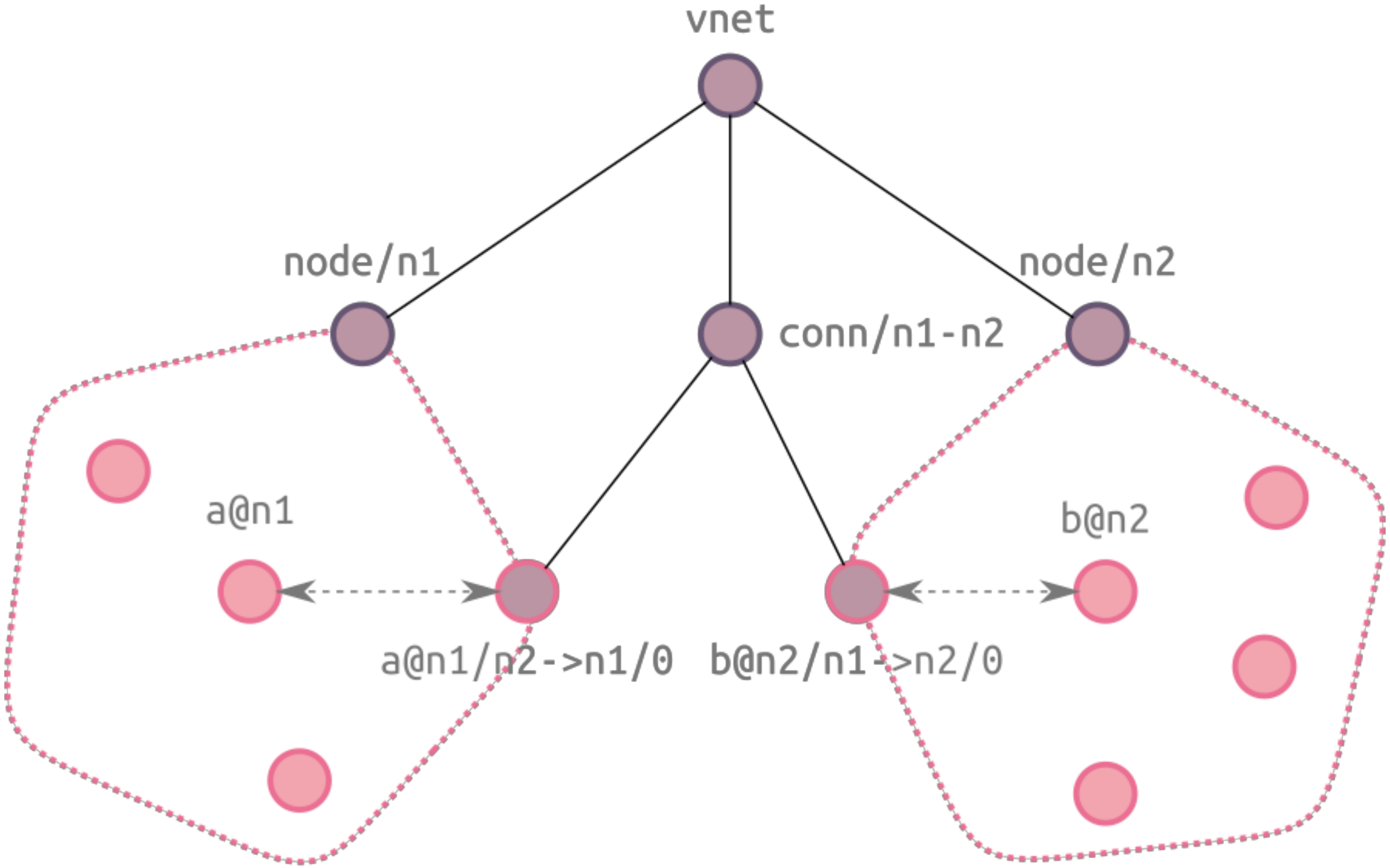


VNET: EXAMPLE

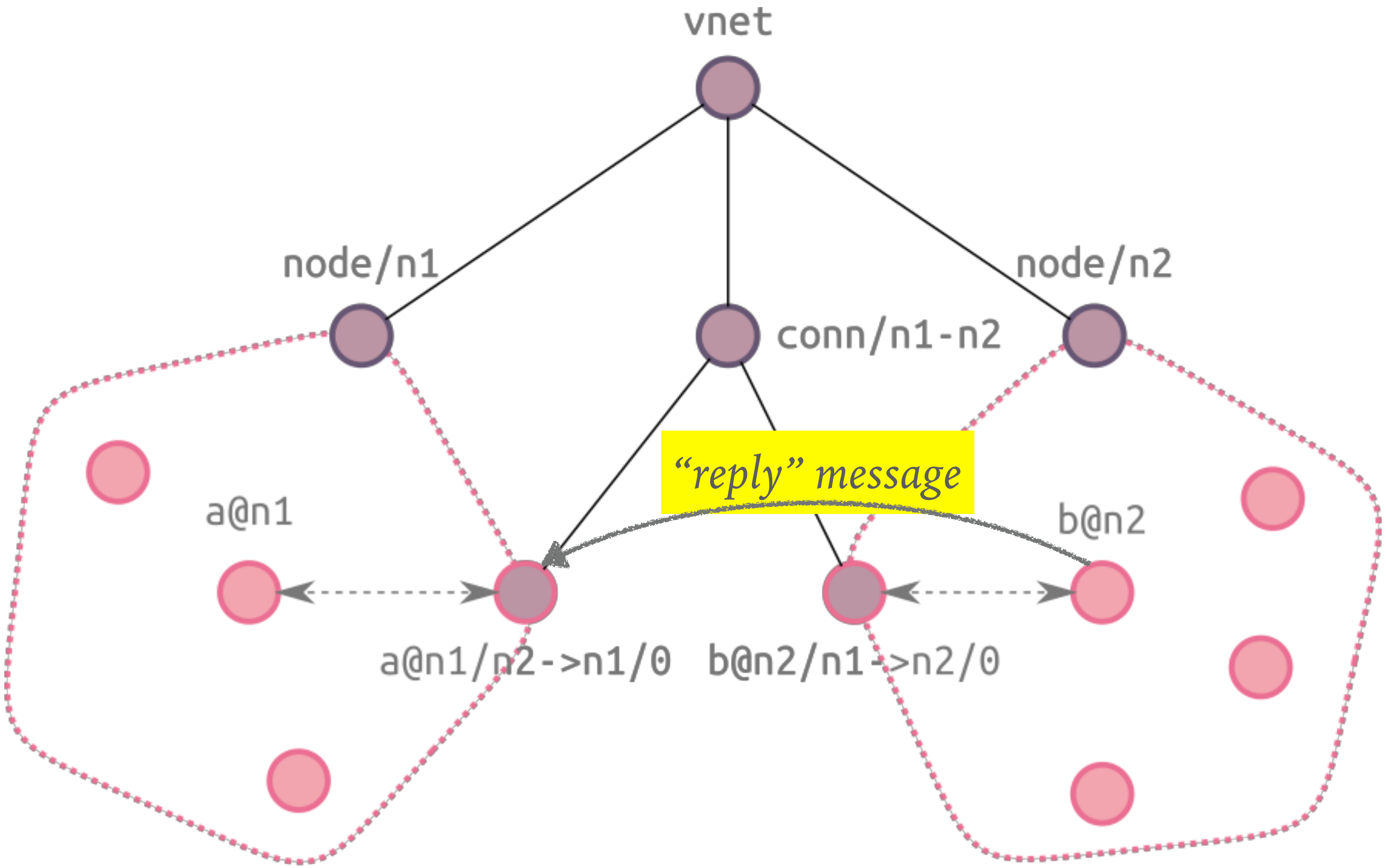


"call" message

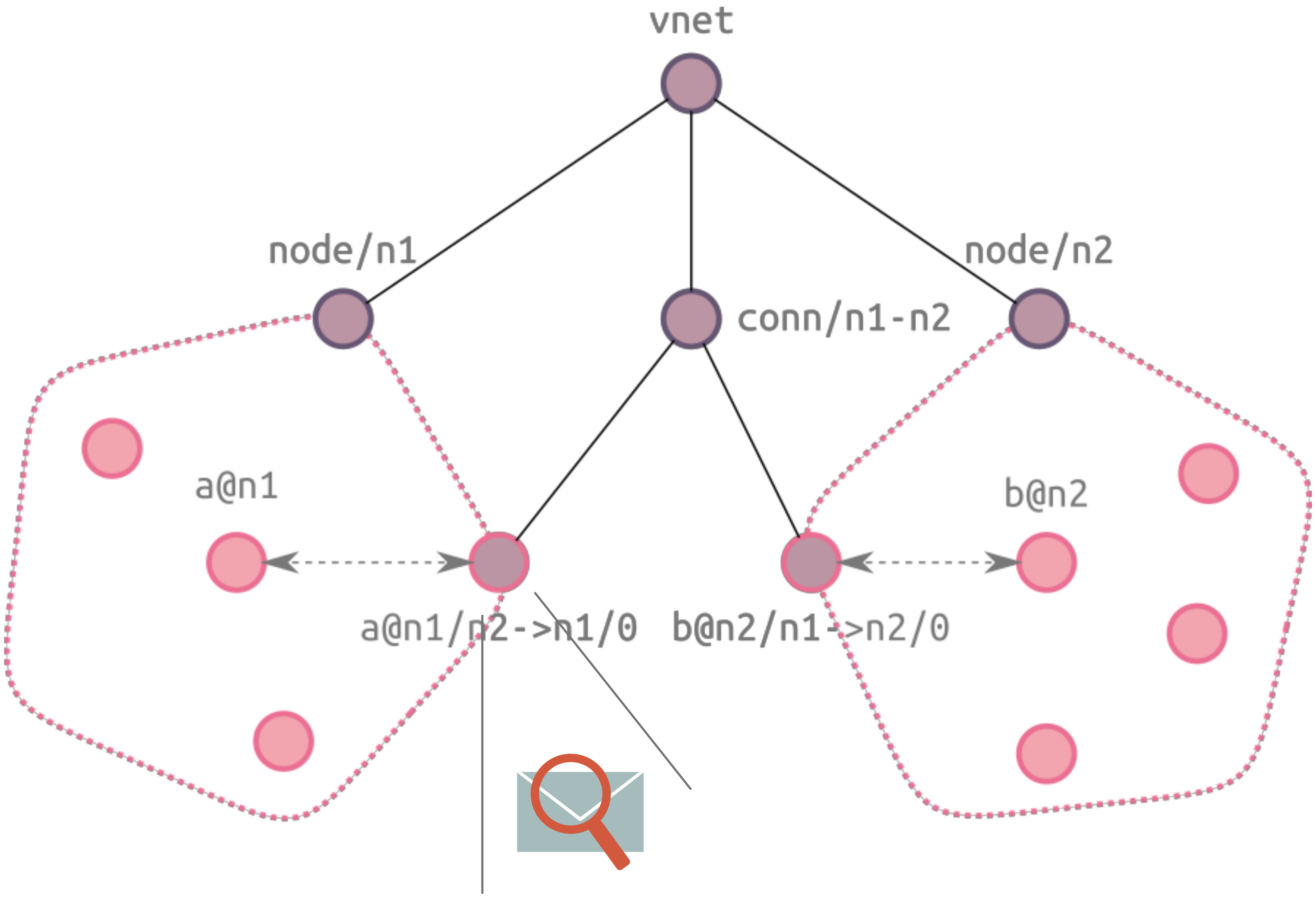
VNET: EXAMPLE



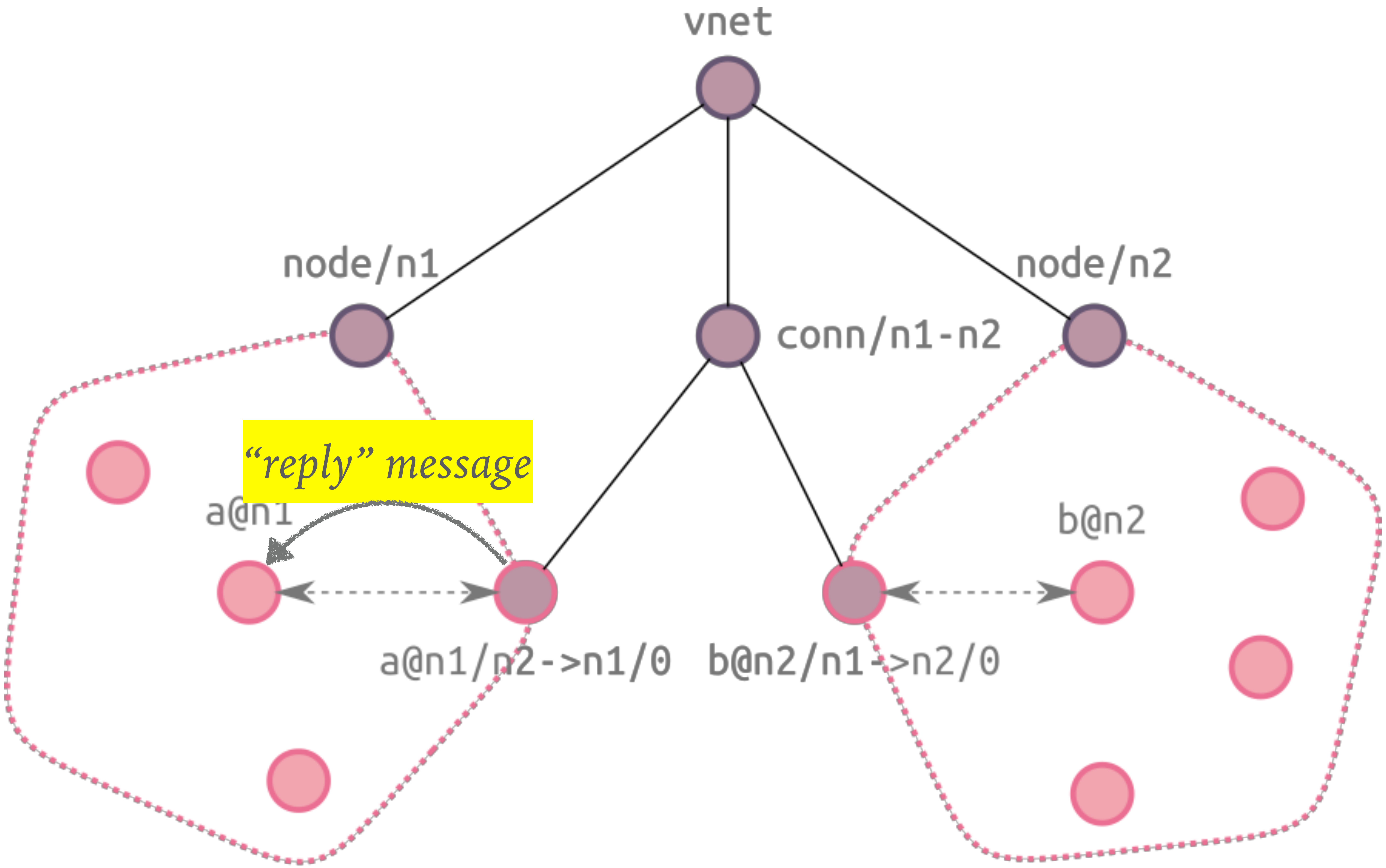
VNET: EXAMPLE



VNET: EXAMPLE

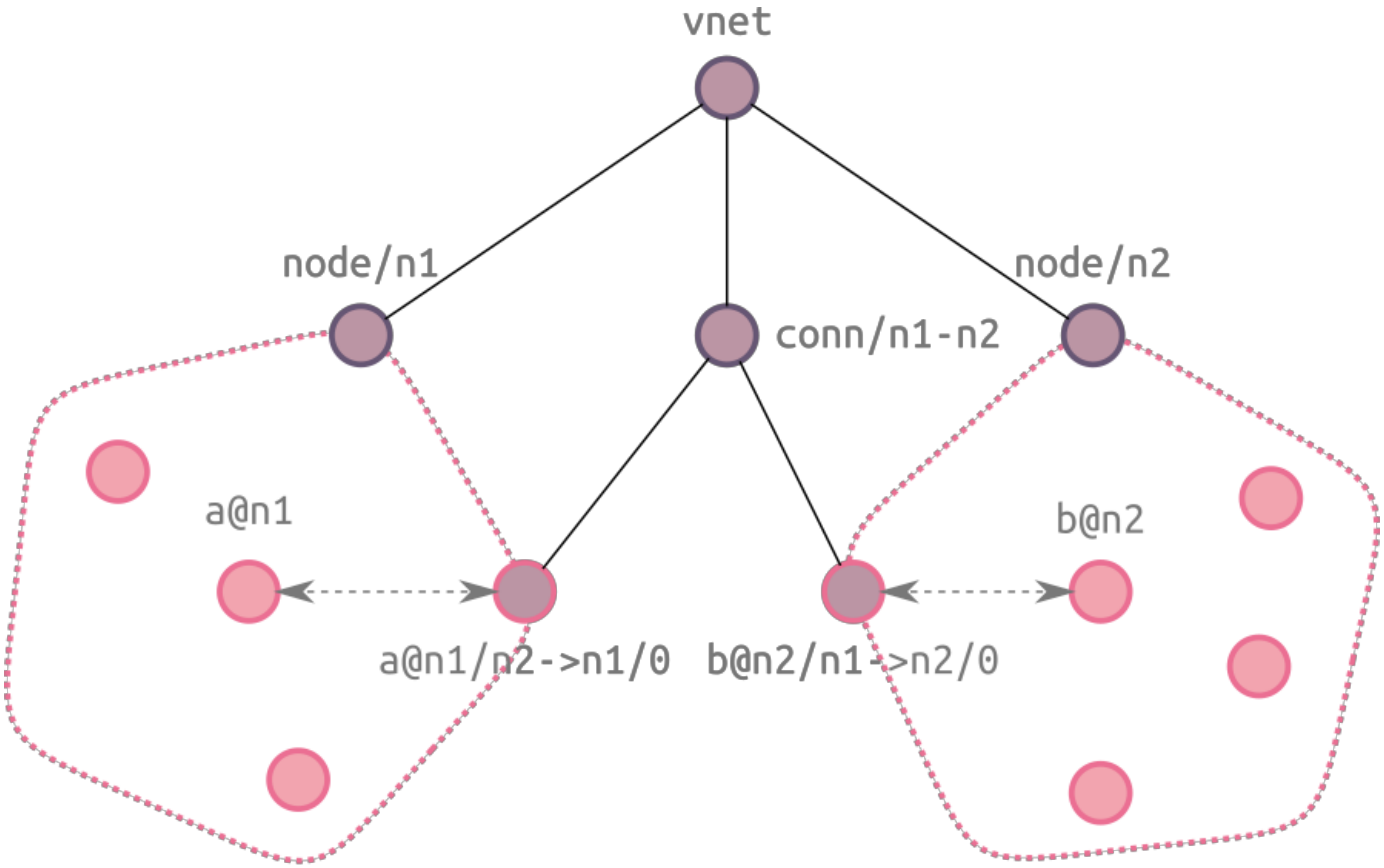


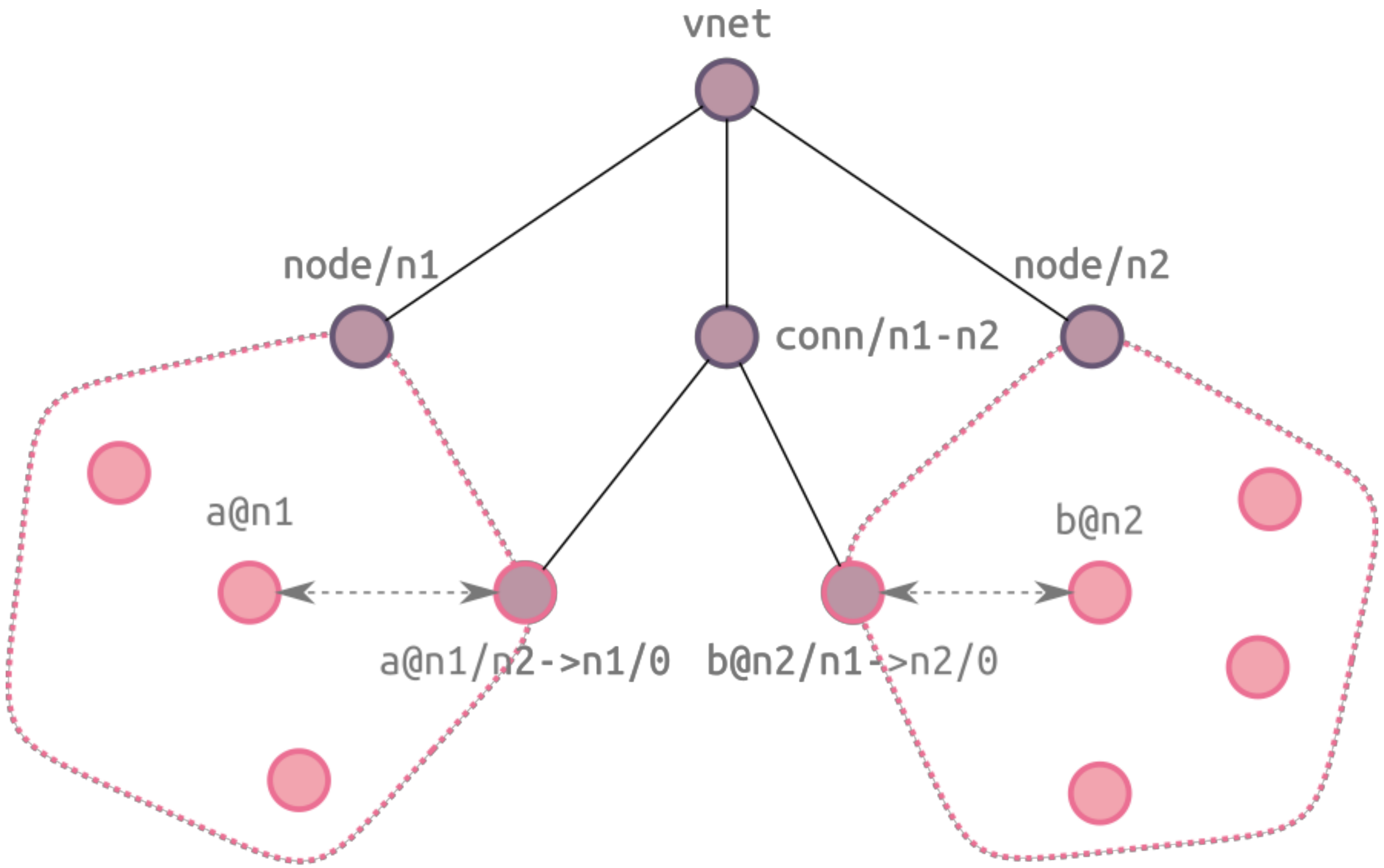
VNET: EXAMPLE



#CodeBEAMSF

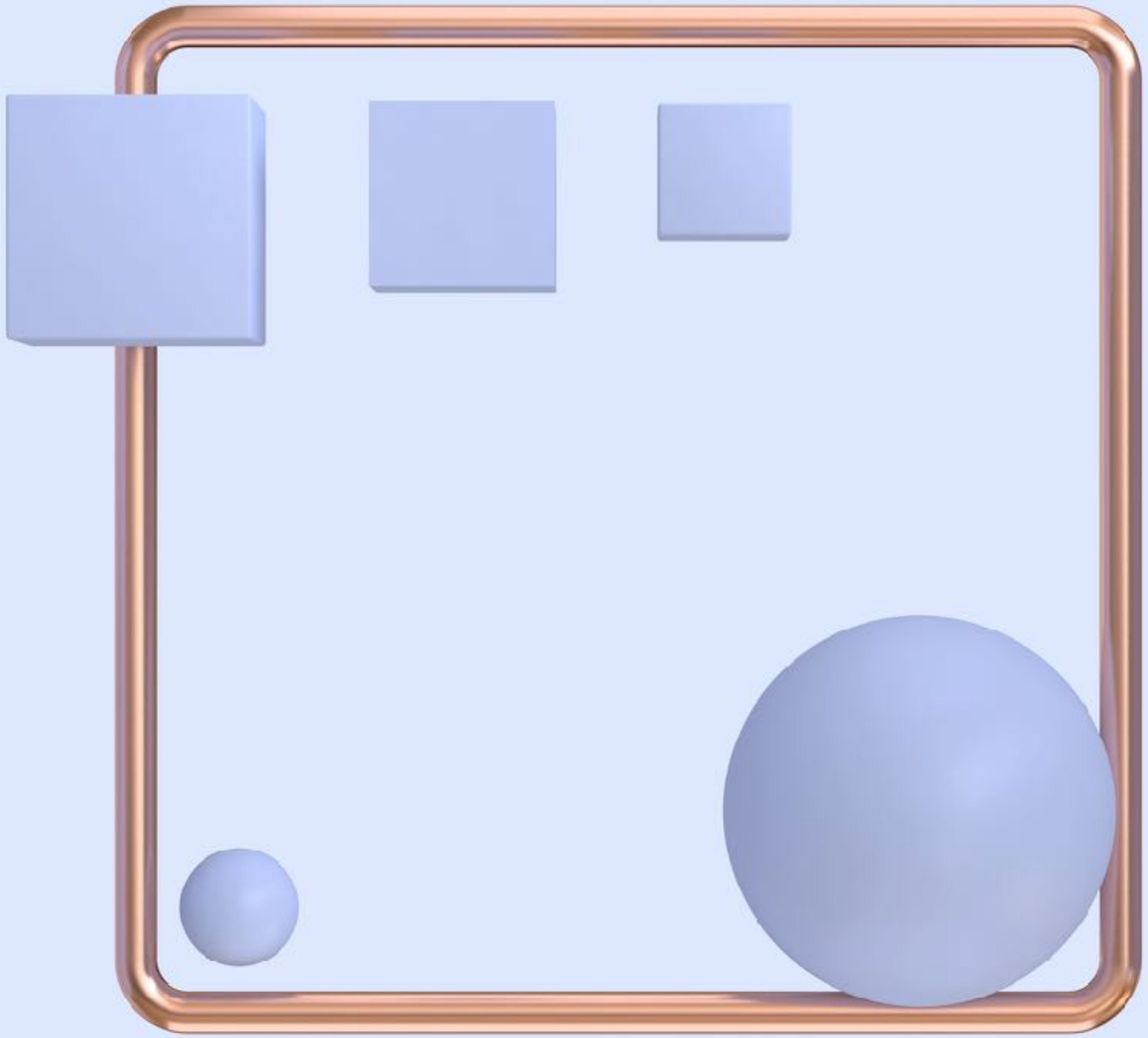
VNET: EXAMPLE





VNET: LIMITATIONS

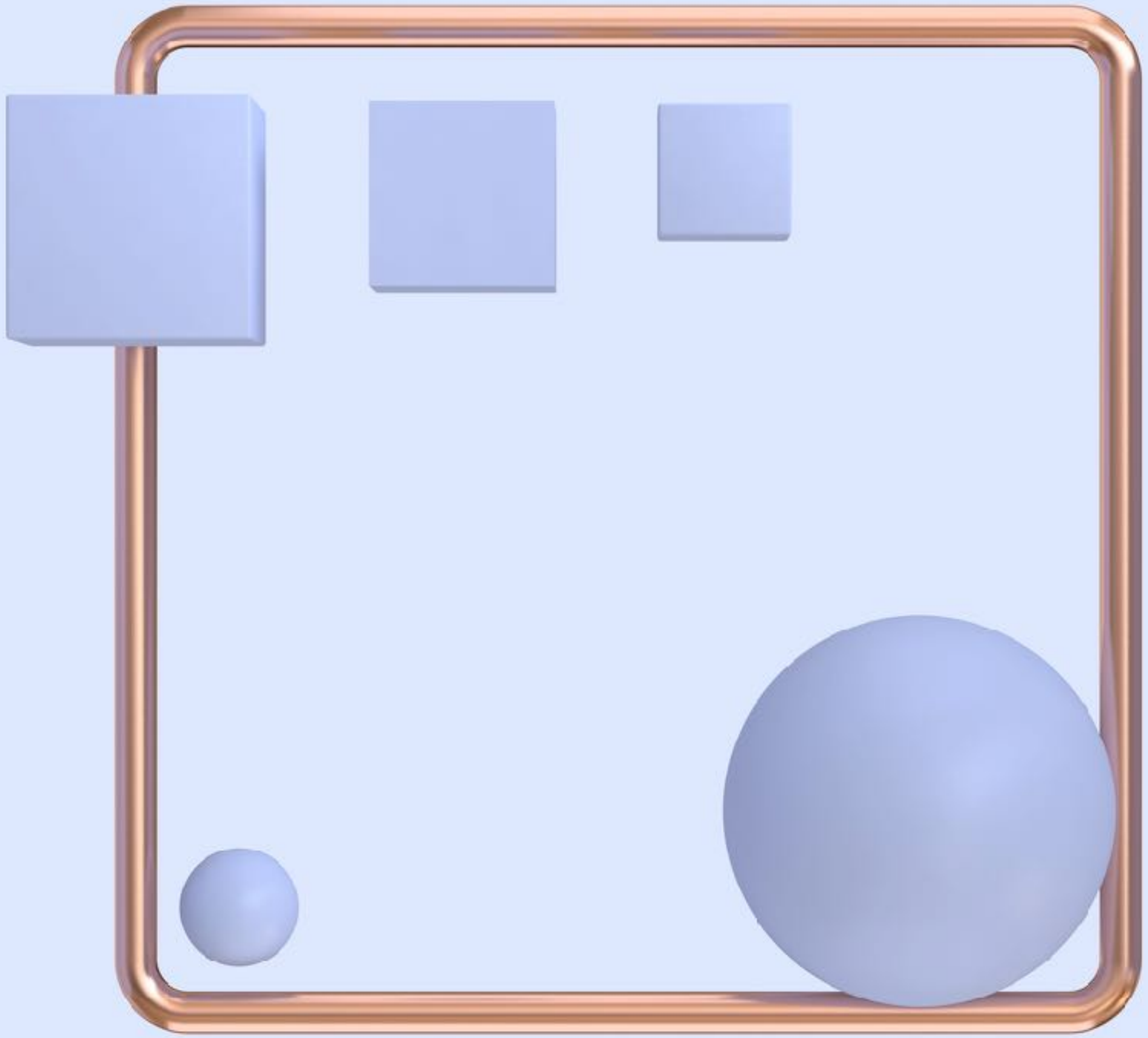
- Models/simulations are usually not ideal
- E.g. “Responses” can arrive out-of-order with monitor signals
- Explained in detail in the paper



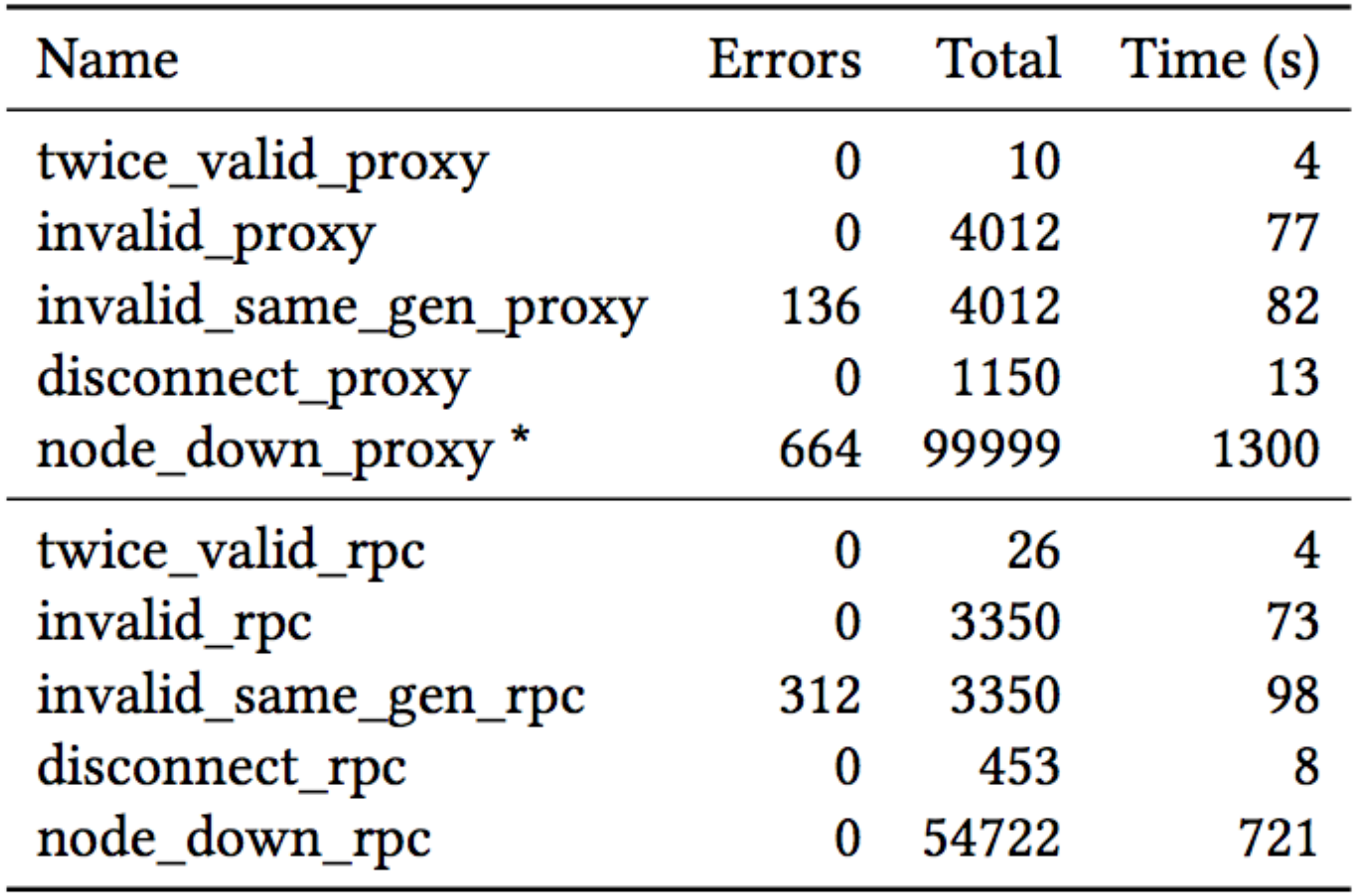
VNET: EXPERIENCES

- KRED/KDB model (not public)
- Simple distributed system
  - Counter server (node A)
    - Supervised gen_server
    - Counter's value survives restarts
  - 'Good client' (node B)
  - 'Bad client' (node C)

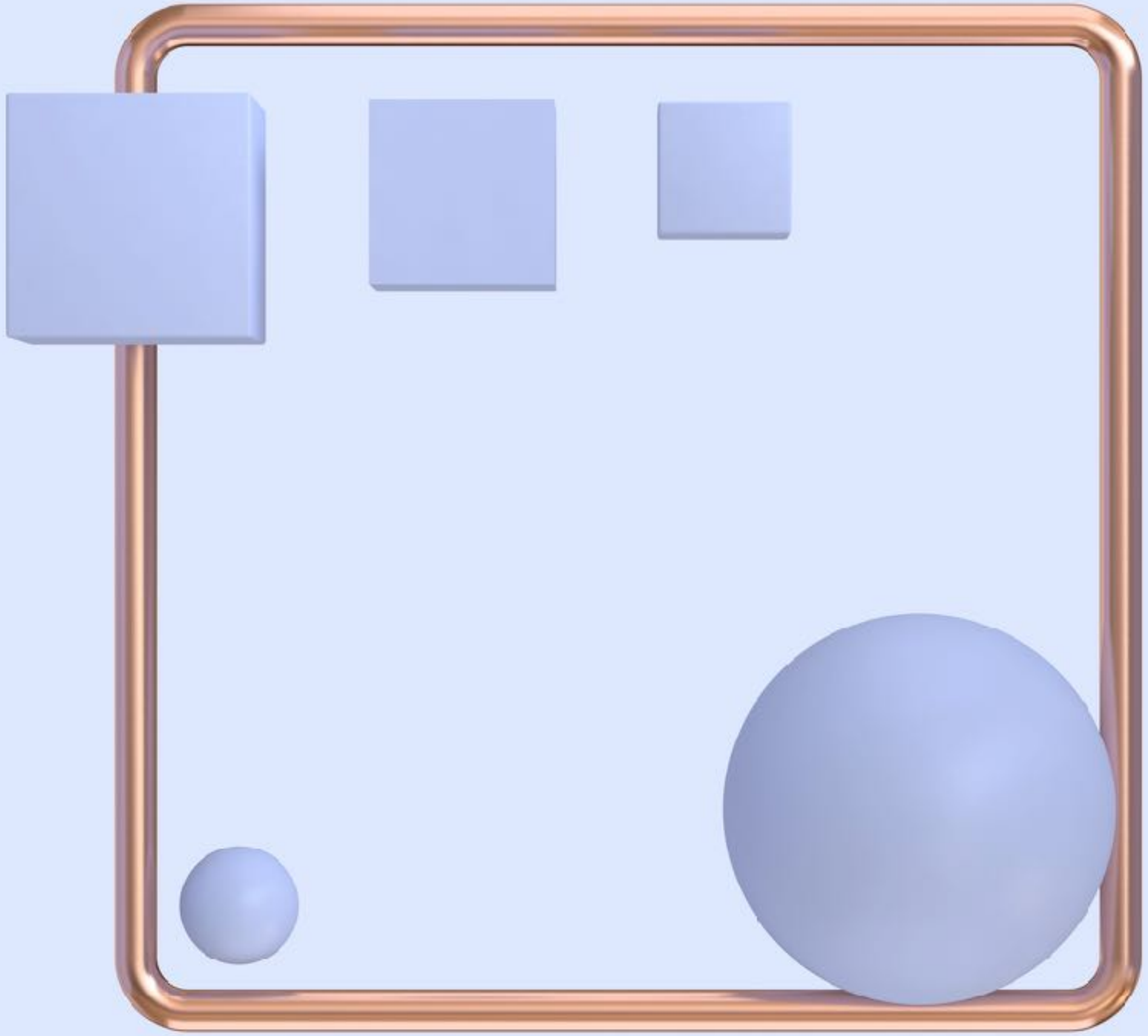
#CodeBEAMSF



VNET: SIMPLE DISTRIBUTED SYSTEM



Name	Errors	Total	Time (s)
twice_valid_proxy	0	10	4
invalid_proxy	0	4012	77
invalid_same_gen_proxy	136	4012	82
disconnect_proxy	0	1150	13
node_down_proxy *	664	99999	1300
twice_valid_rpc	0	26	4
invalid_rpc	0	3350	73
invalid_same_gen_rpc	312	3350	98
disconnect_rpc	0	453	8
node_down_rpc	0	54722	721



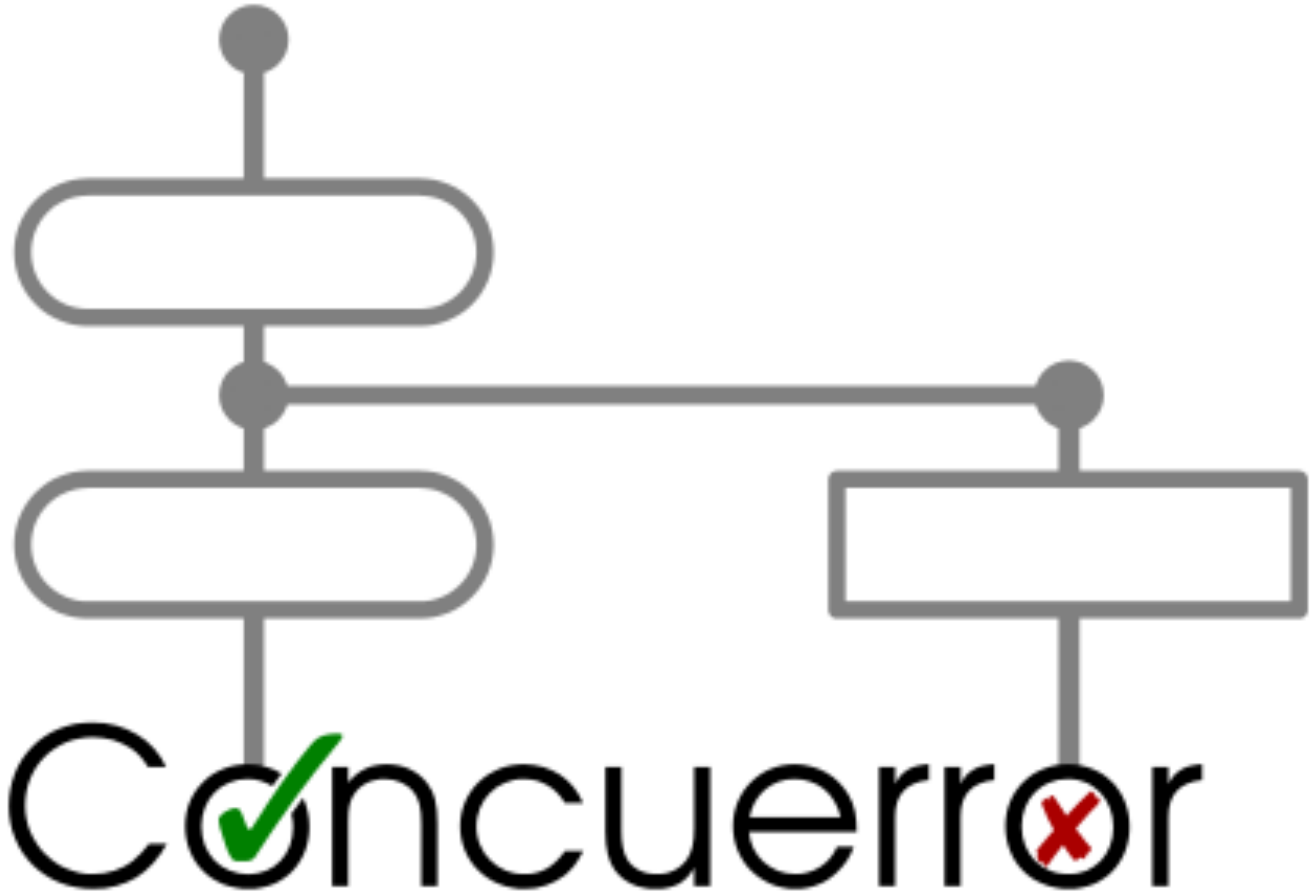
VNET: EXPERIENCES

- Concuerror has a steep learning curve
- Start simple!
- Inspect detected races
- Ask me for help!

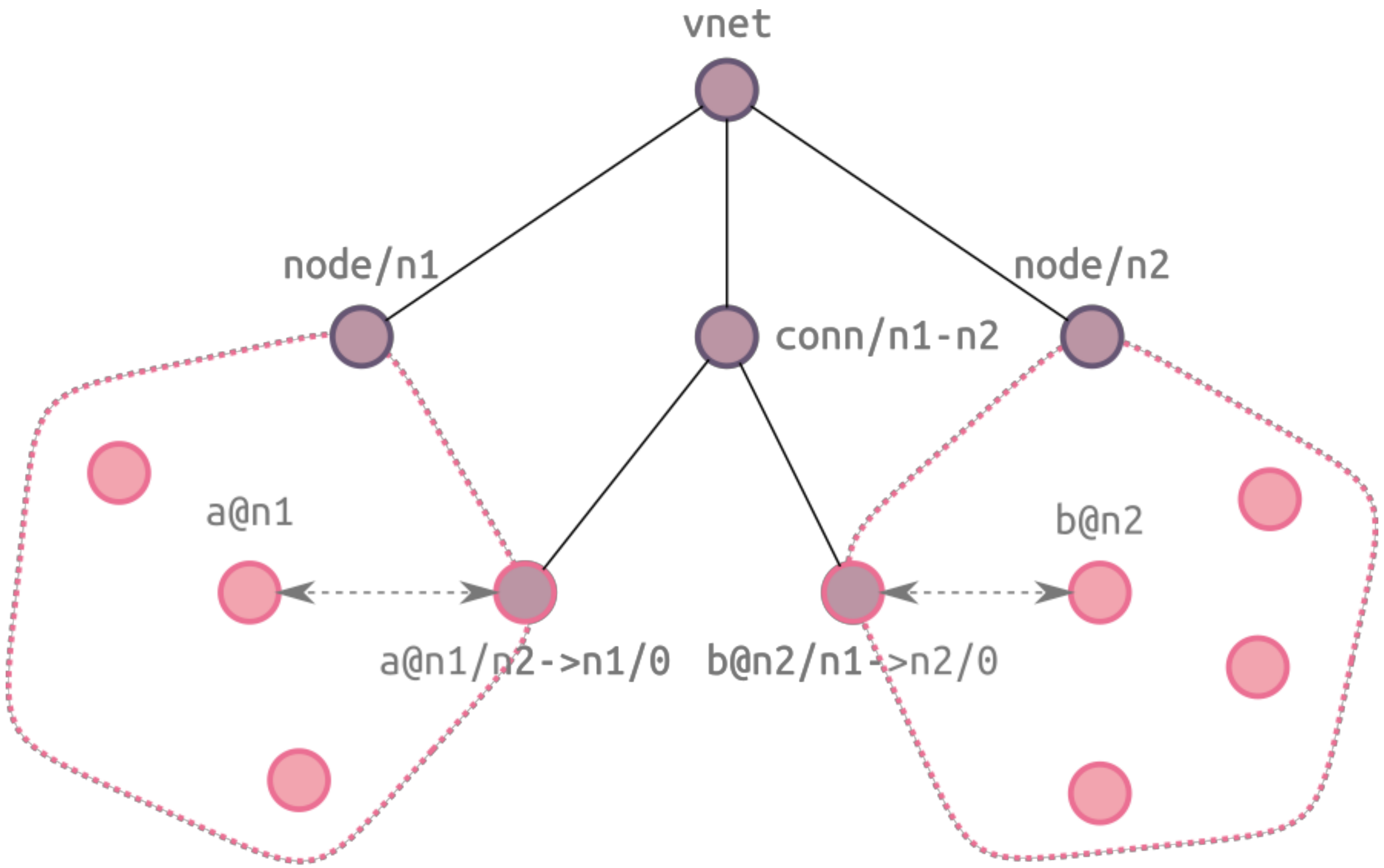
WRAPPING UP!

CONCUERROR

- Makes you understand concurrency
- Is very effective on models & prototypes
- Can **verify** safety of test scenarios
- Catches design flaws early



HTTPS://CONCUERROR.COM



VNET

- Enables modelling distributed systems on a single Erlang node
- Works out-of-the-box with OTP
- Try it out!
  - https://github.com/klarna/vnet
  - See the test/counter_server_example
- Read the paper!
  - https://concuerror.com/publications

PLAY WITH CONCUERROR & VNET!

- Race conditions are **tricky**!
- Modelling is **fun**!
- Prototypes are **useful**!
- Concurrency testing is **easy**!
- Verification is **possible**!

Thank you!

#CodeBEAMSF