HARU CONFERENCE

ELIXIR를 이용한 분산시스템

OVERVIEW

- Erlang/OTP
- Elixir
- Distributed systems
- Wrapping up

ERALNG/OTP

1986 Ericsson 개발

1998 오픈소스화

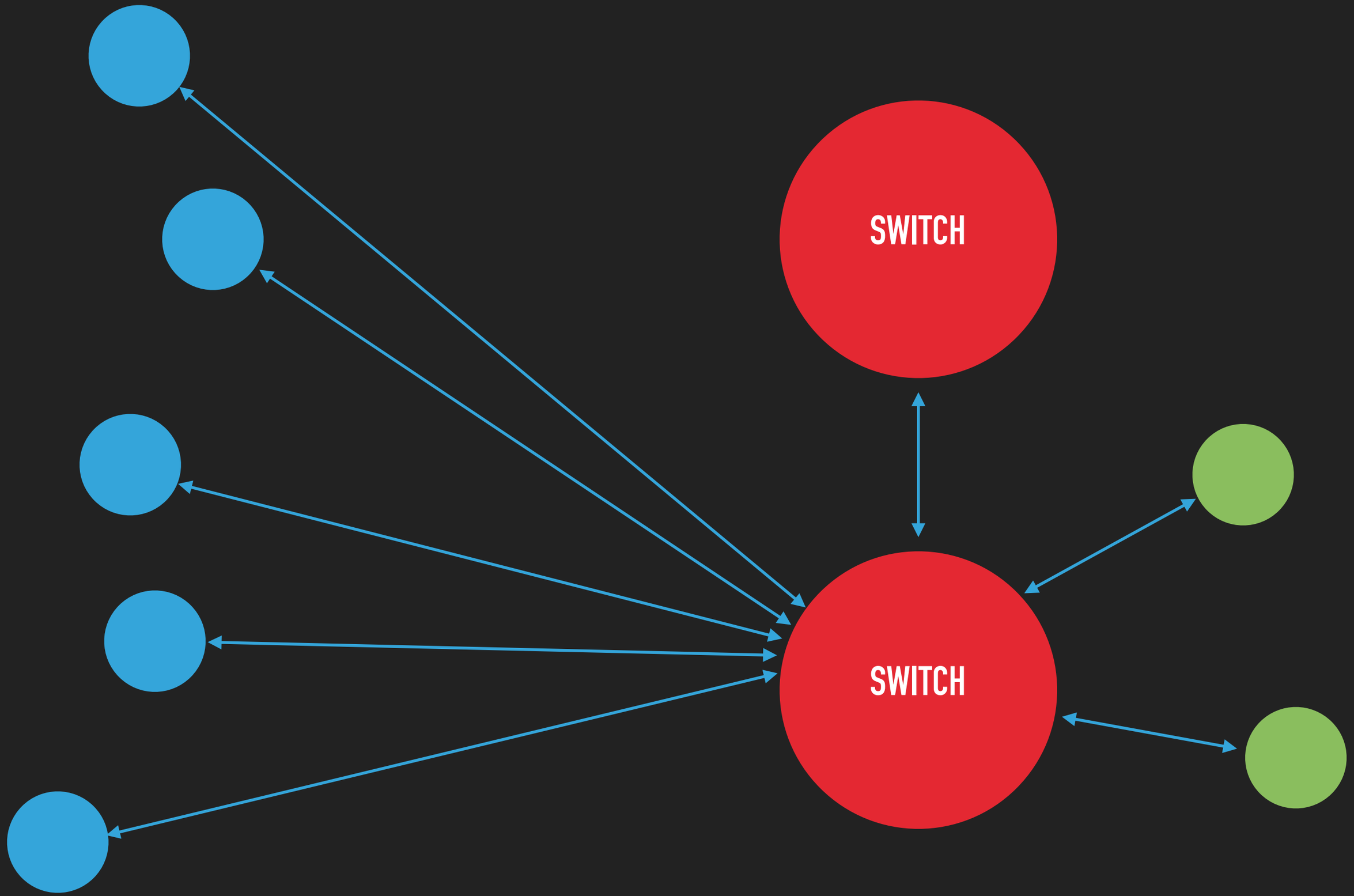
2016 Erlang OTP 19.x

1986 Ericsson 개발

1995 Java

1998 오픈소스화

2016 Erlang OTP 19.x

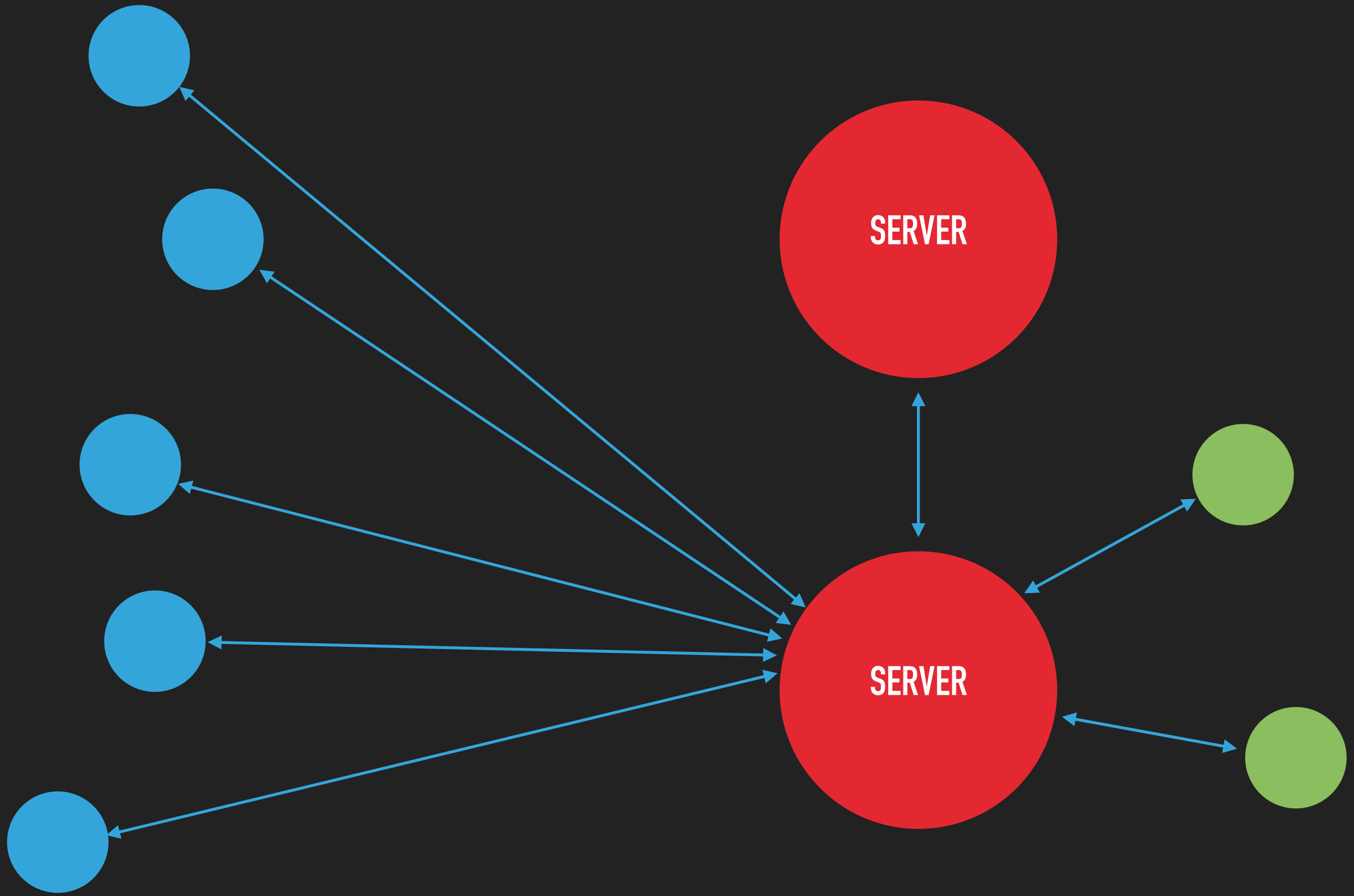


http://stackoverflow.com/questions/1636455/where-is-erlang-used-and-why/1637137#1637137

CONCURRENCY.

FAULT TOLERANCE.

DISTRIBUTION.

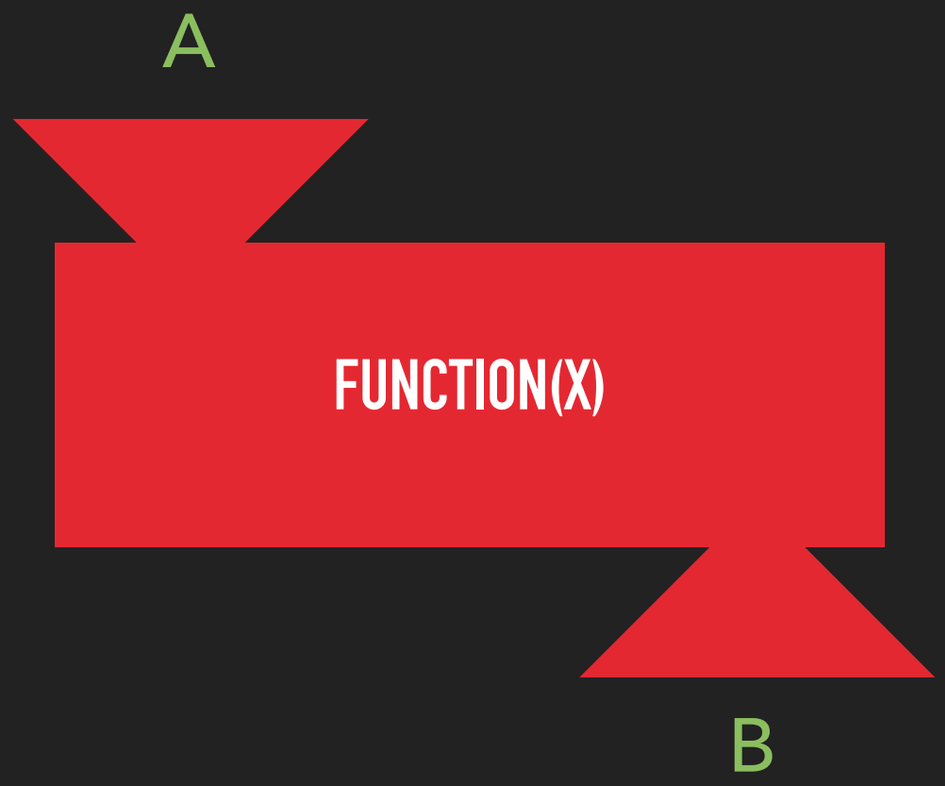


http://stackoverflow.com/questions/1636455/where-is-erlang-used-and-why/1637137#1637137

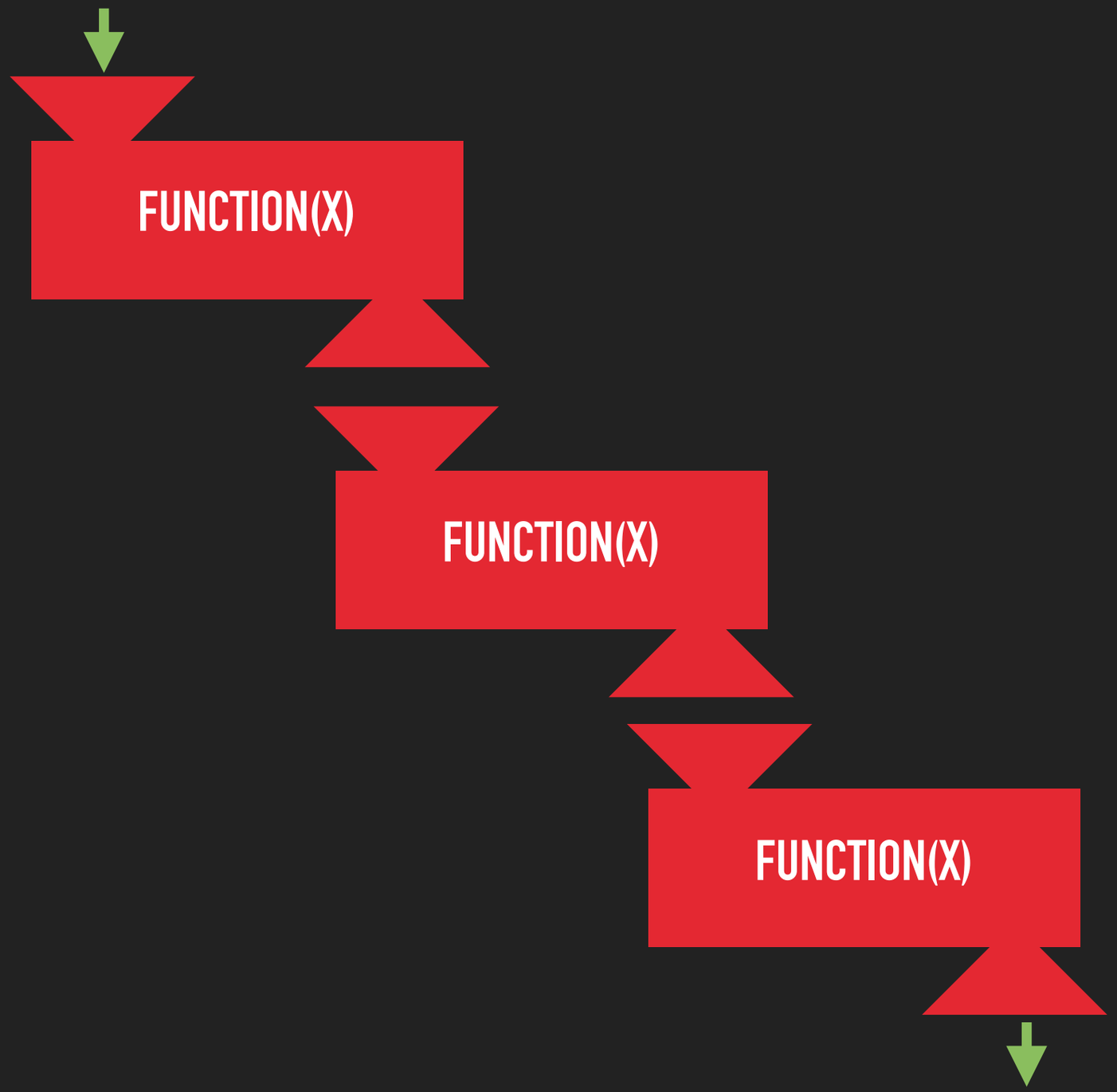
ERLANG/OTP

- Functional language
- Immutable variables
- Strong, dynamic typing
- Actor model
- OTP
  - Behavior

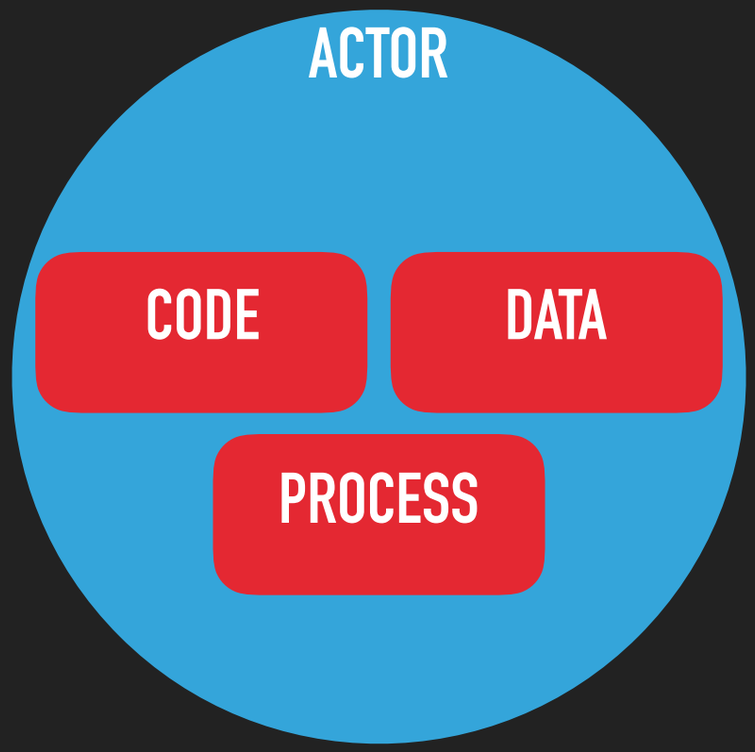
FUNCTIONAL LANGUAGE



FUNCTIONAL LANGUAGE

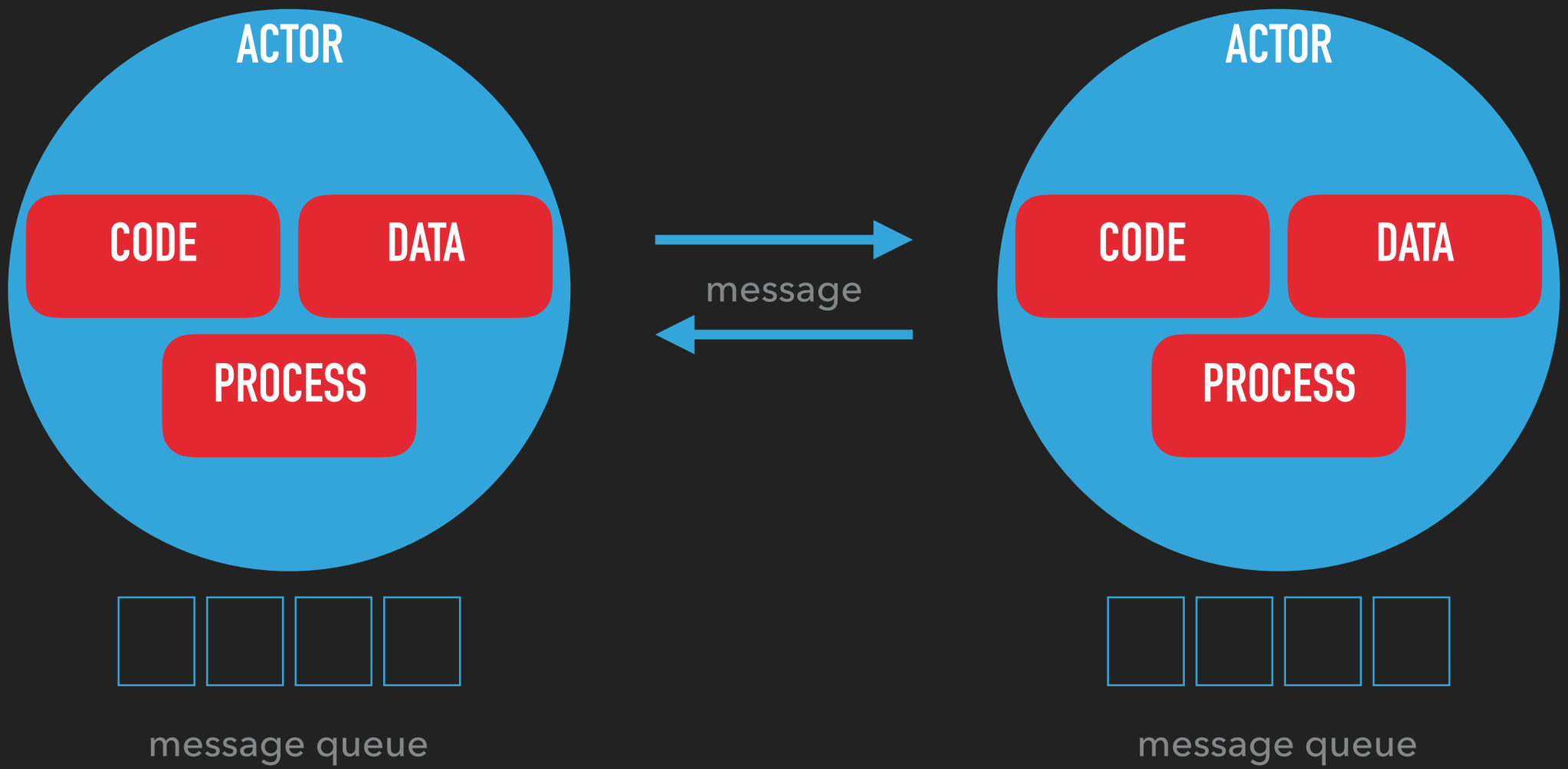


ACTOR MODEL

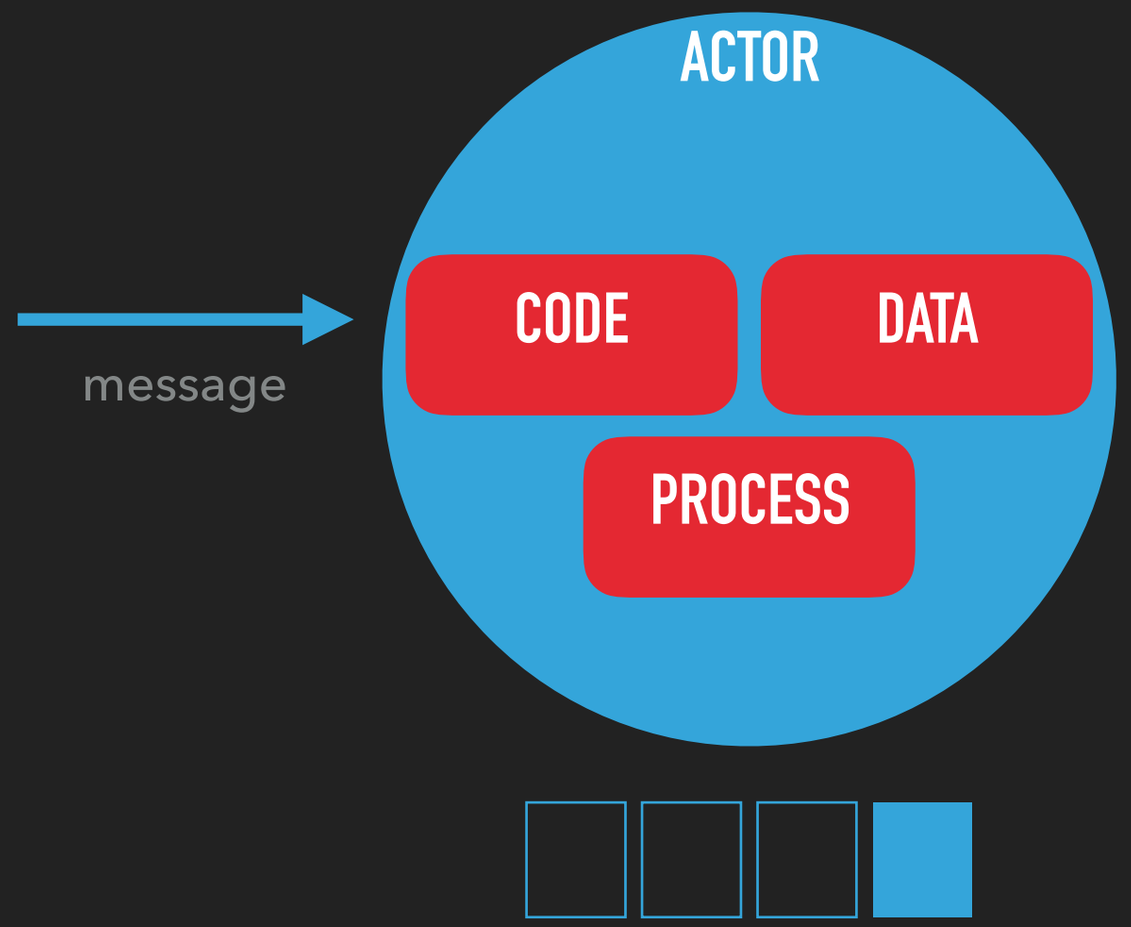


Actor = Lightweight Process + Data + Code

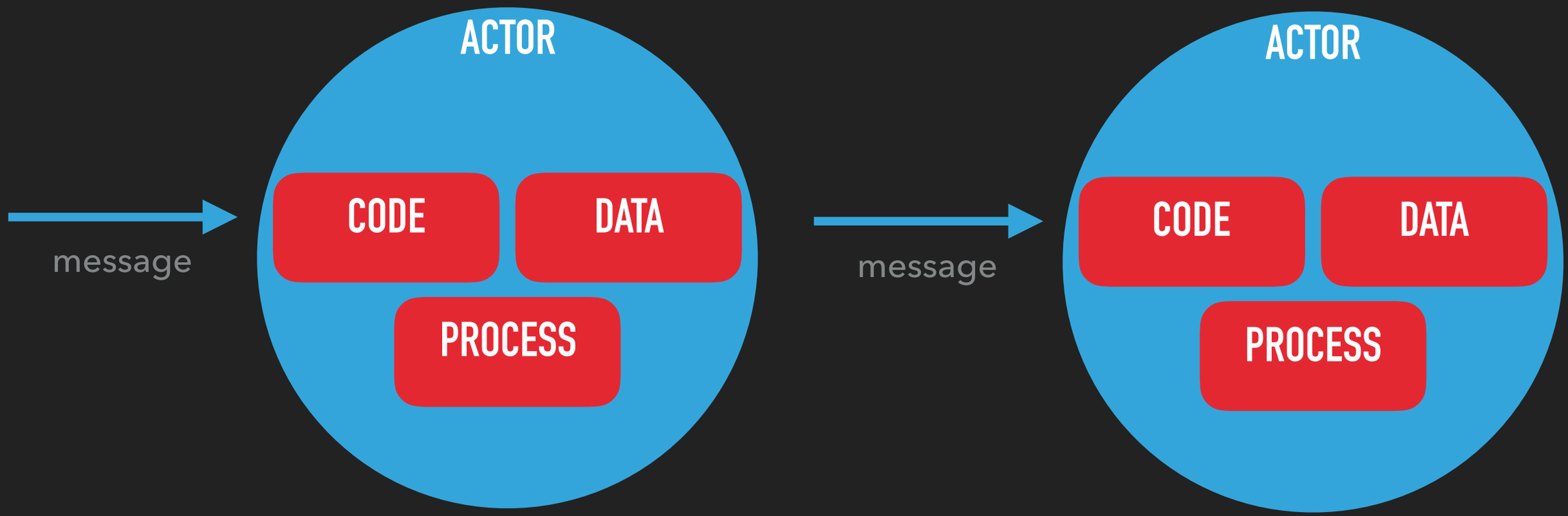
ACTOR MODEL


ACTOR
CODE
DATA
PROCESS
message
ACTOR
CODE
DATA
PROCESS
message queue
message queue

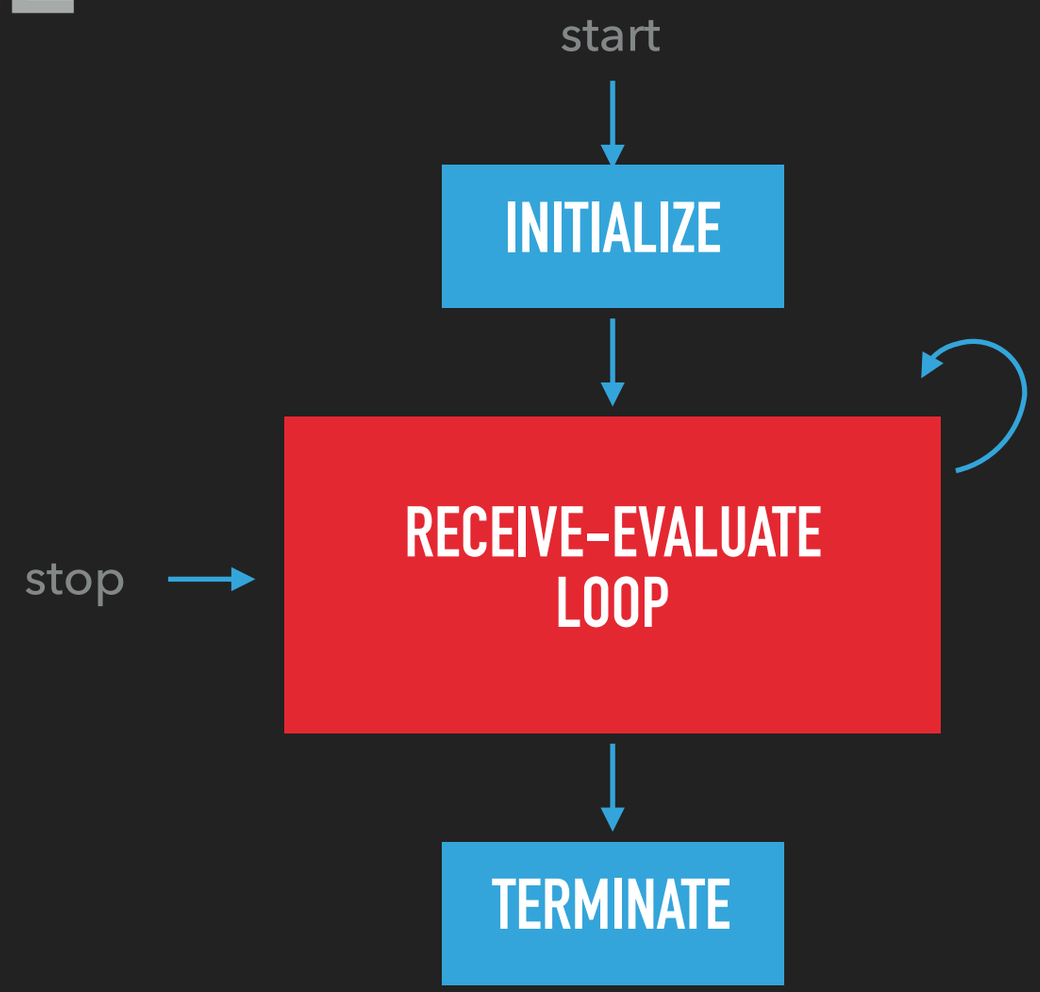
ACTOR MODEL



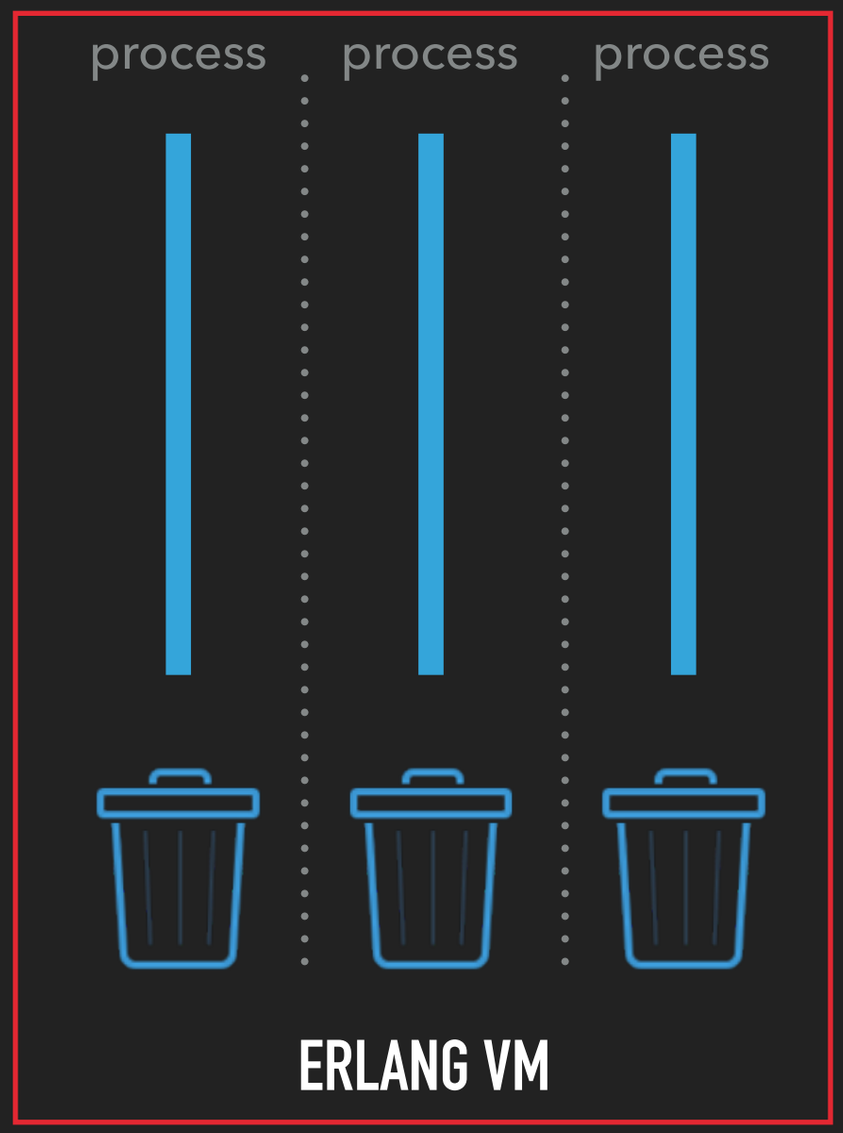
ACTOR MODEL


ACTOR
CODE
DATA
PROCESS
message
ACTOR
CODE
DATA
PROCESS
message

ACTOR MODEL
LIFE CYCLE



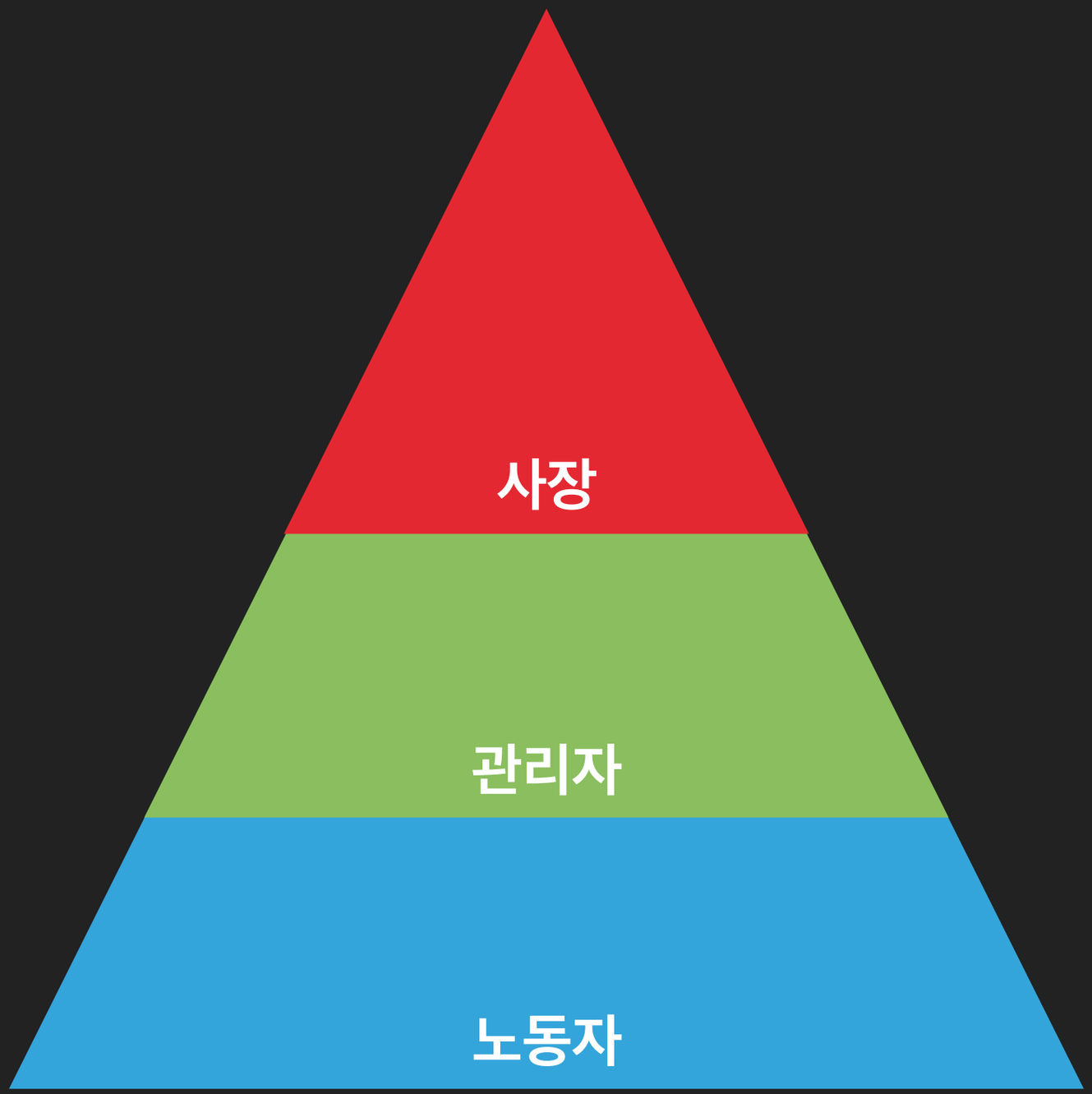
ACTOR MODEL
GARBAGE COLLECTOR



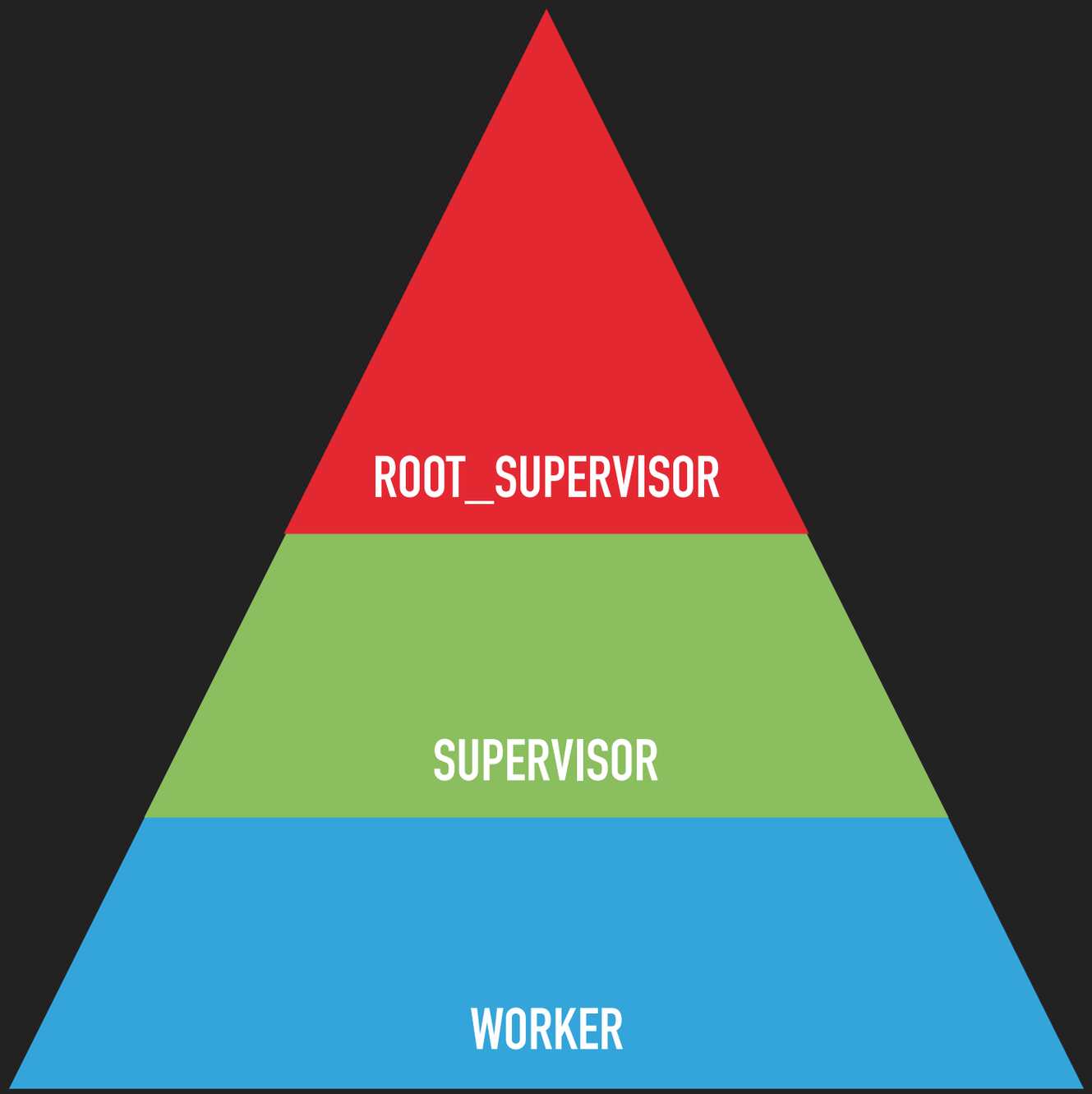
https://viethip.com/category/erlang/

ACTOR MODEL

ACTOR MODEL



ACTOR MODEL



OTP-BEHAVIOR

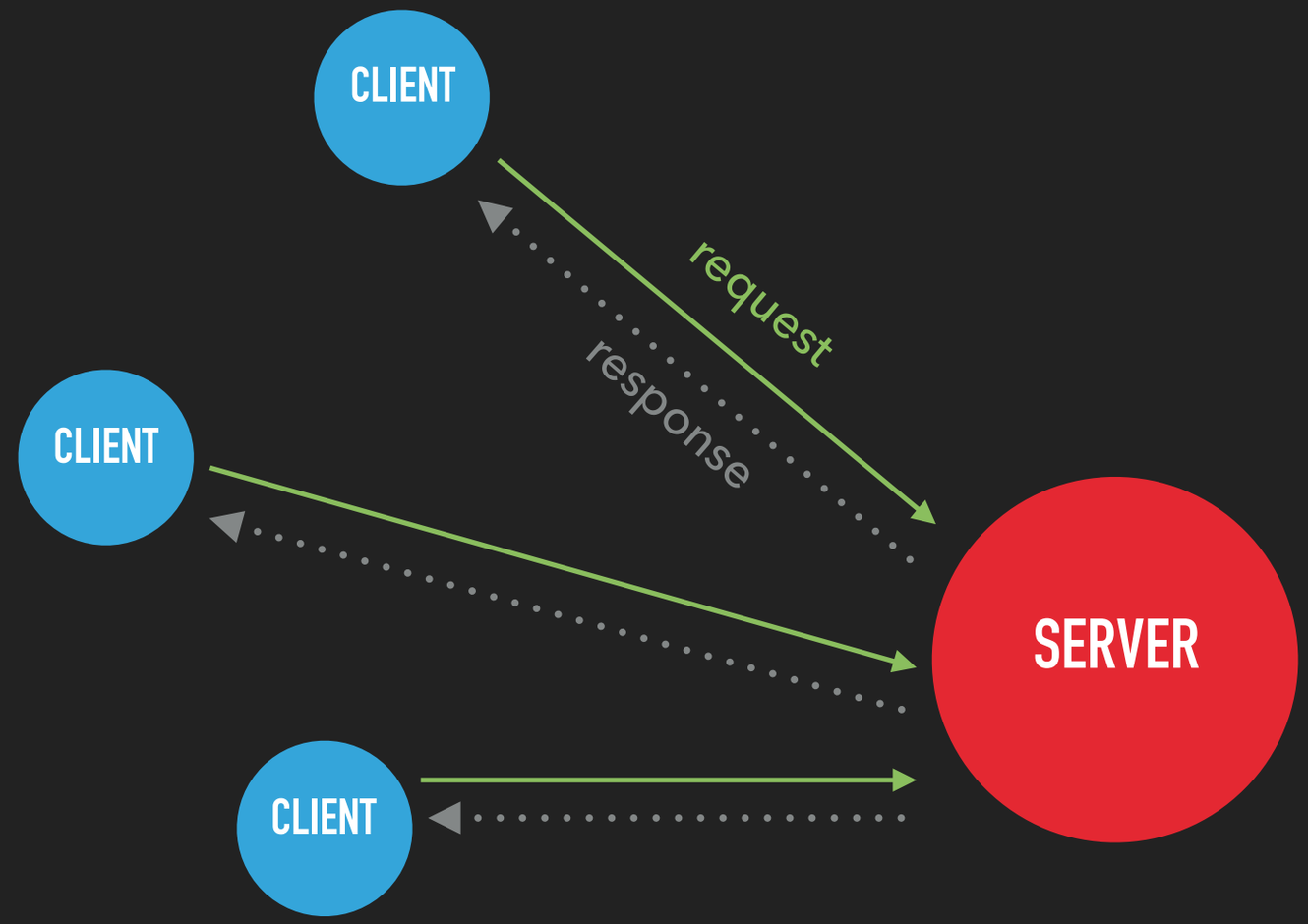
- GenServer
- Supervisor
- GenEvent
- GenFsm
- GenStatem

OTP-BEHAVIOR

- GenServer
- Supervisor
- GenEvent
- GenStatem

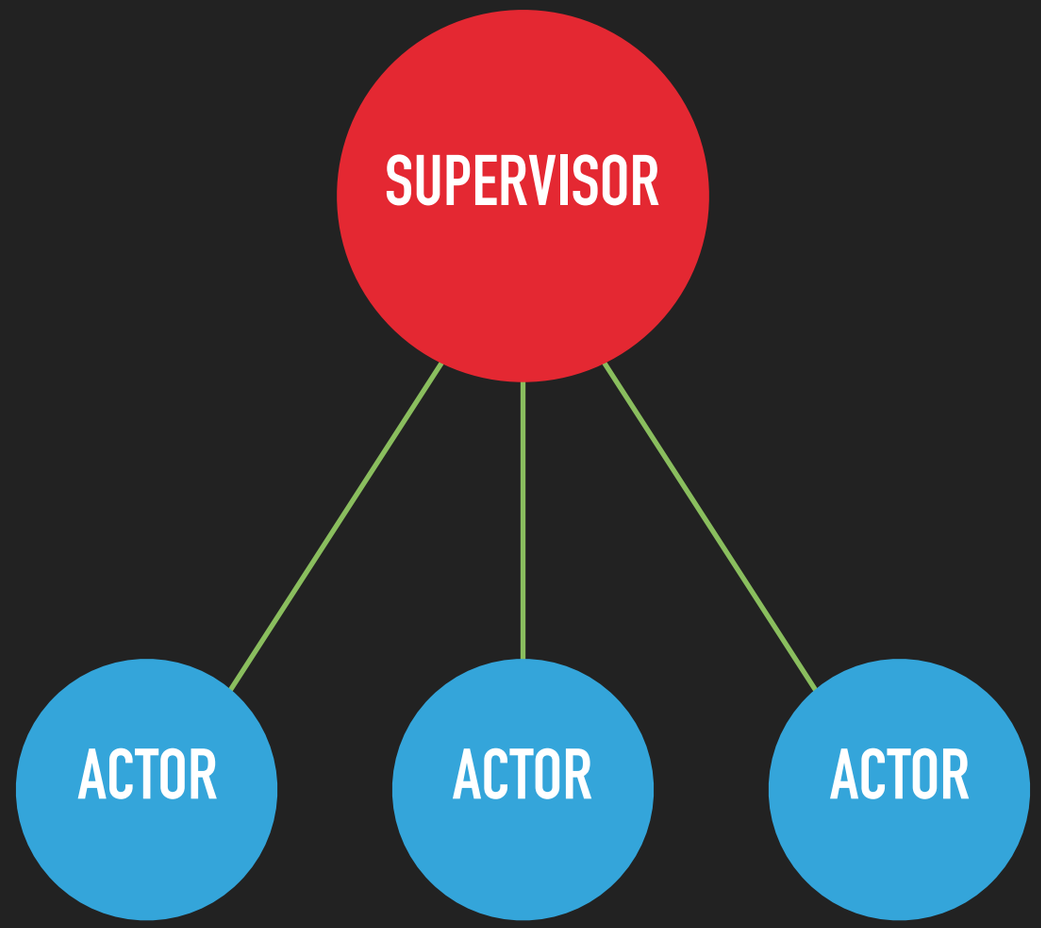
GENSERVER

GENERIC SERVER



http://erlang.org/doc/design_principles/gen_server_concepts.html

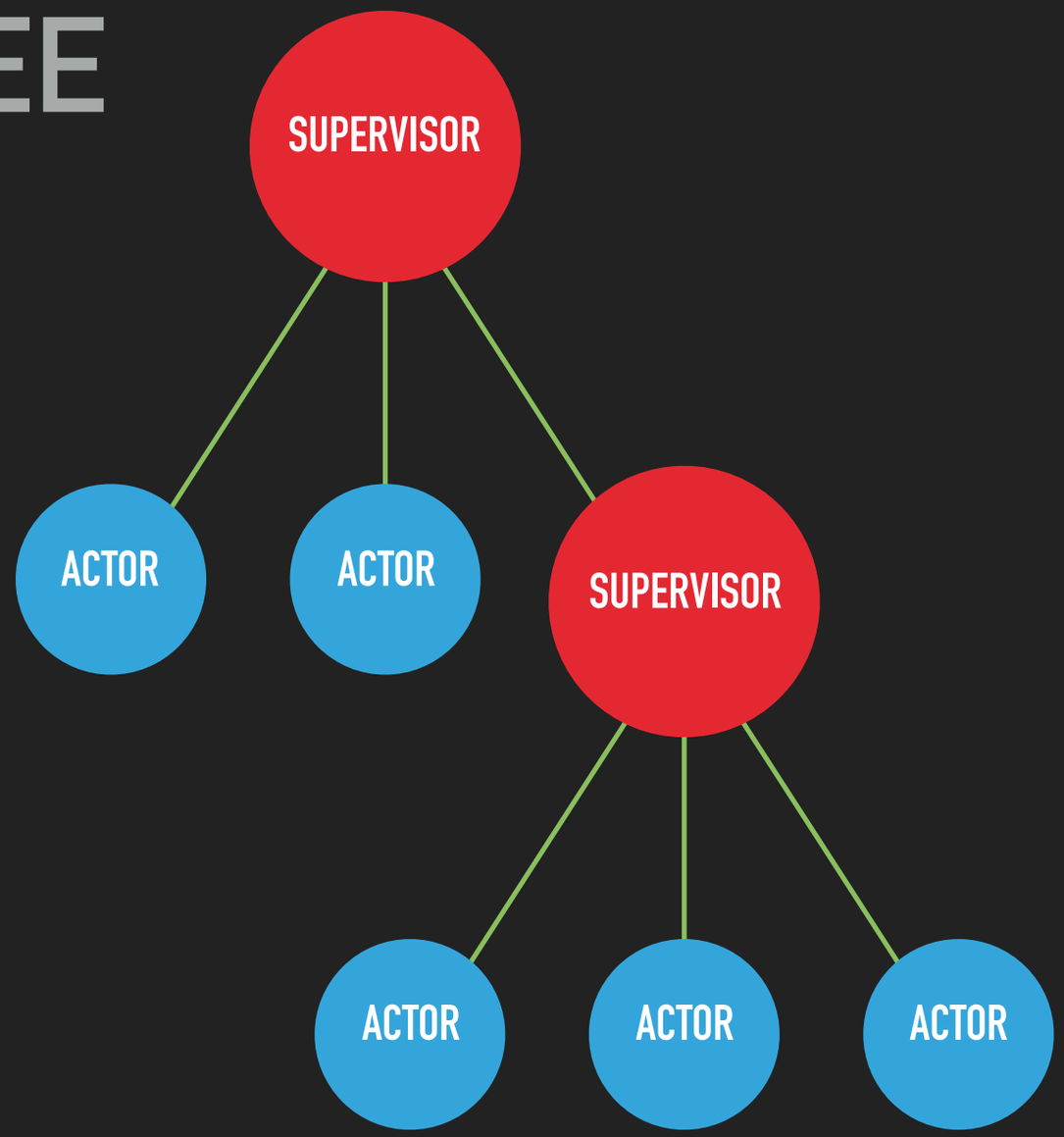
SUPERVISOR



http://erlang.org/doc/design_principles/gen_server_concepts.html

SUPERVISOR

SUPERVISION TREE



http://erlang.org/doc/design_principles/gen_server_concepts.html

“Erlang was designed for writing concurrent programs that run forever.”

A History of Erlang – Joe Amstrong

ELIXIR

2012 José Valim 개발

2016 Elixir 1.3.x

ELIXIR

- Functional language
- Immutable variables
- Strong, dynamic typing
- Actor model

ELIXIR

ELIXIR

- Support tools
  - ExUnit - Unit test
  - Mix - Build tool
  - Standard library
  - Metaprogramming
- Ecosystem
  - Hex - Package management
  - Phoenix - Web framework

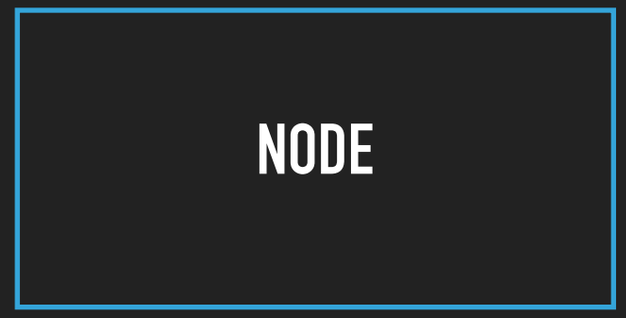
DISTRIBUTED SYSTEMS

“A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable”

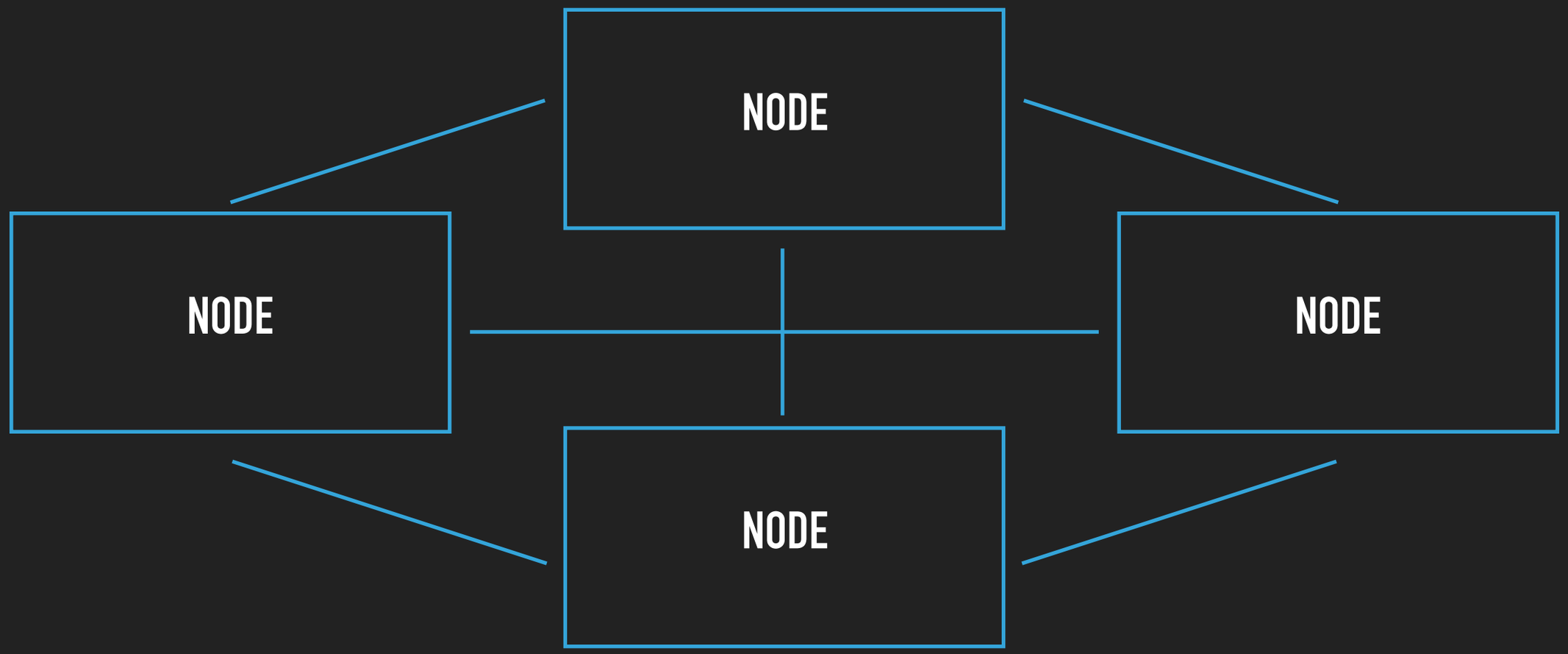
Leslie Lamport

ERLANG CLUSTER

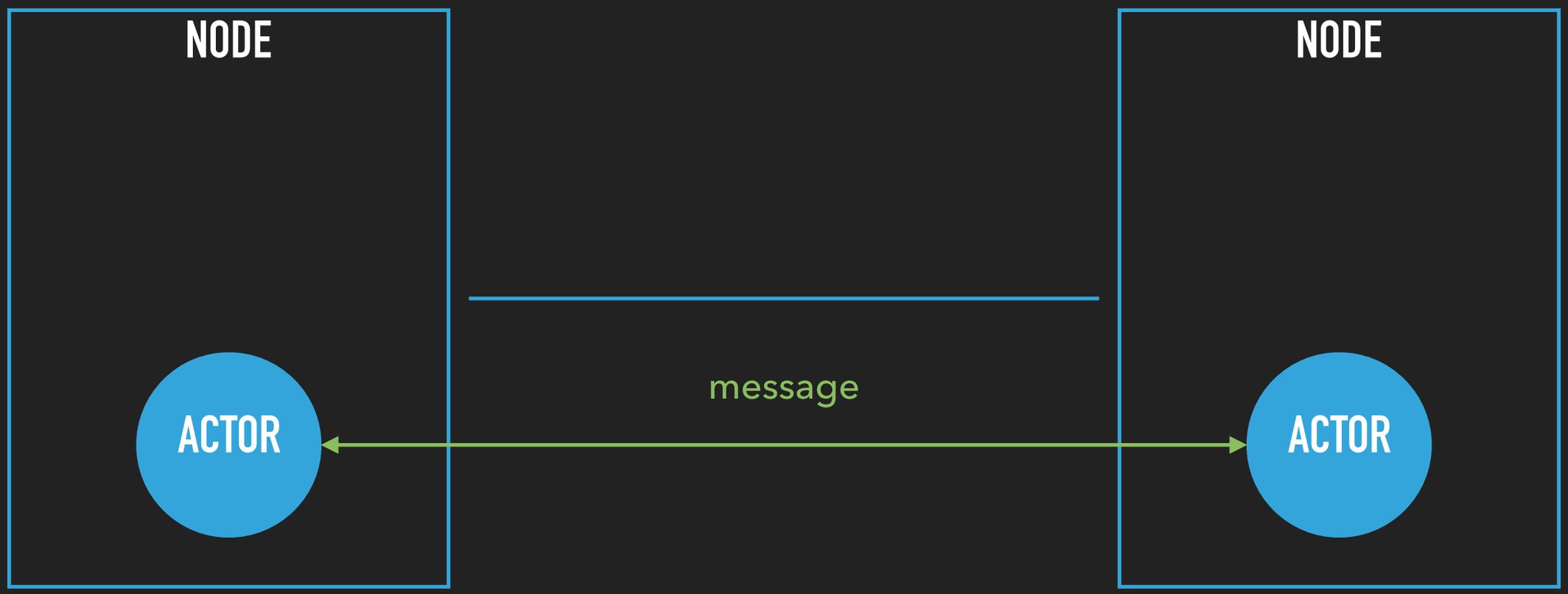
A distributed Erlang system consists of a number of Erlang runtime systems communicating with each other. Each such runtime system is called a node.



ERLANG CLUSTER



ERLANG CLUSTER



WHY DISTRIBUTED SYSTEMS?

WHY DISTRIBUTED SYSTEMS?

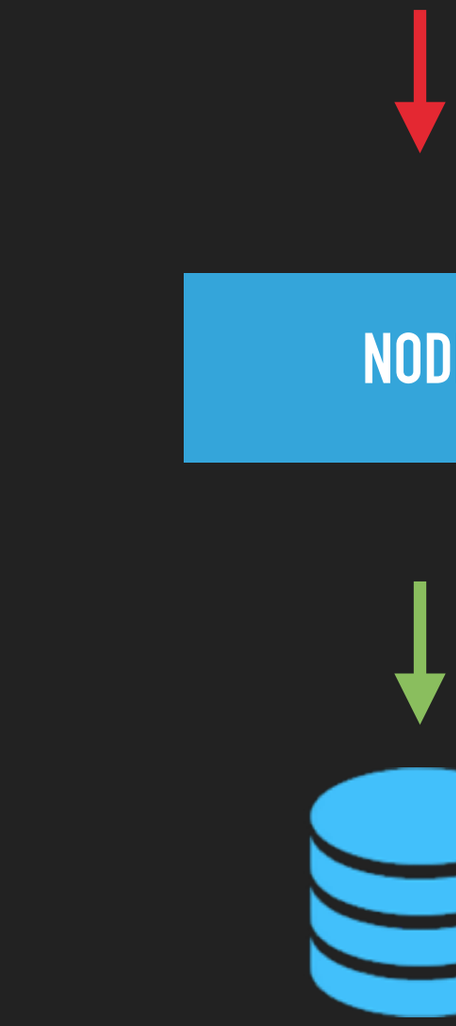
- Enhanced Performance
- Higher Availability

PERFORMANCE

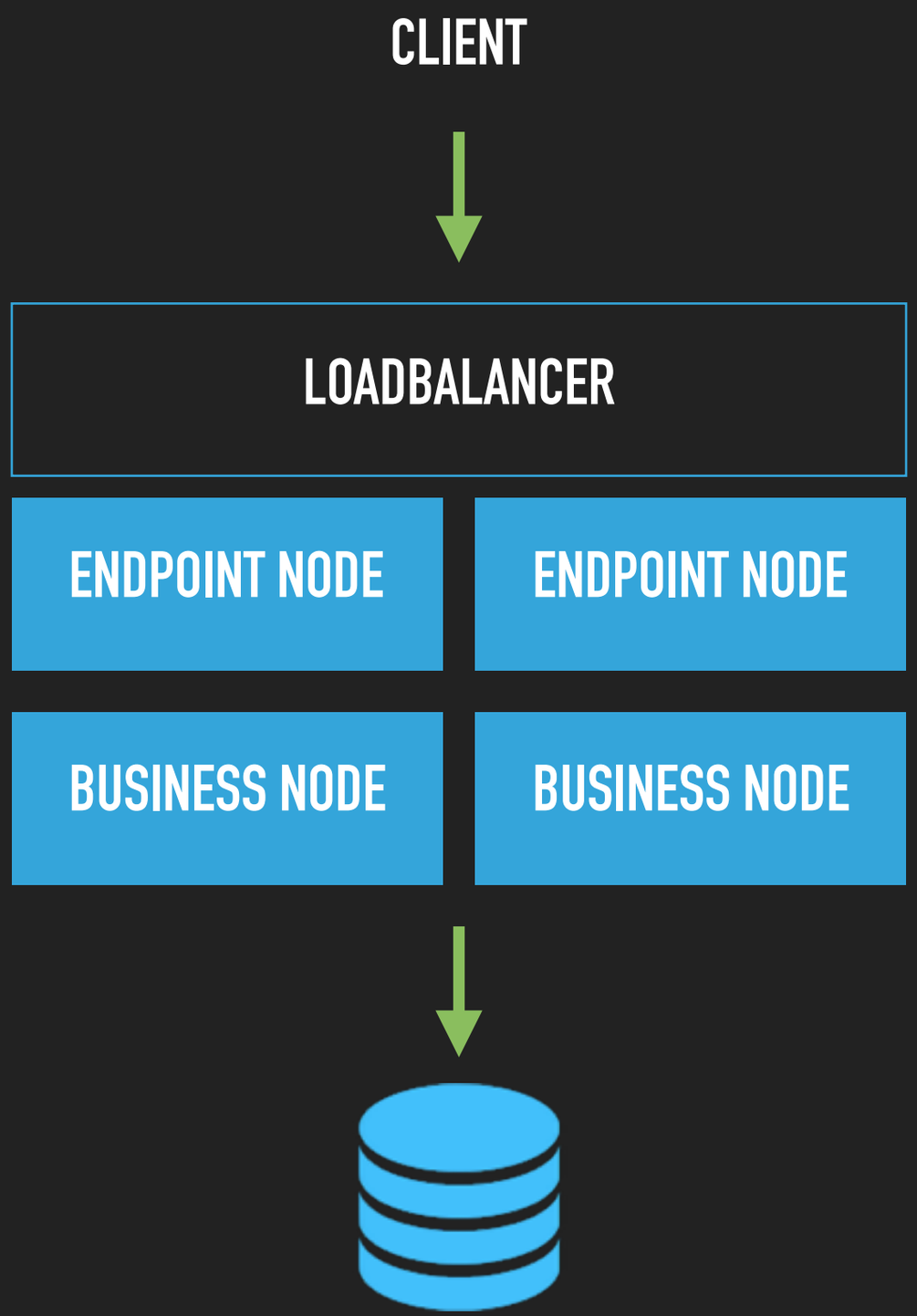
- Latency
- Throughput
- Computing power

PERFORMANCE

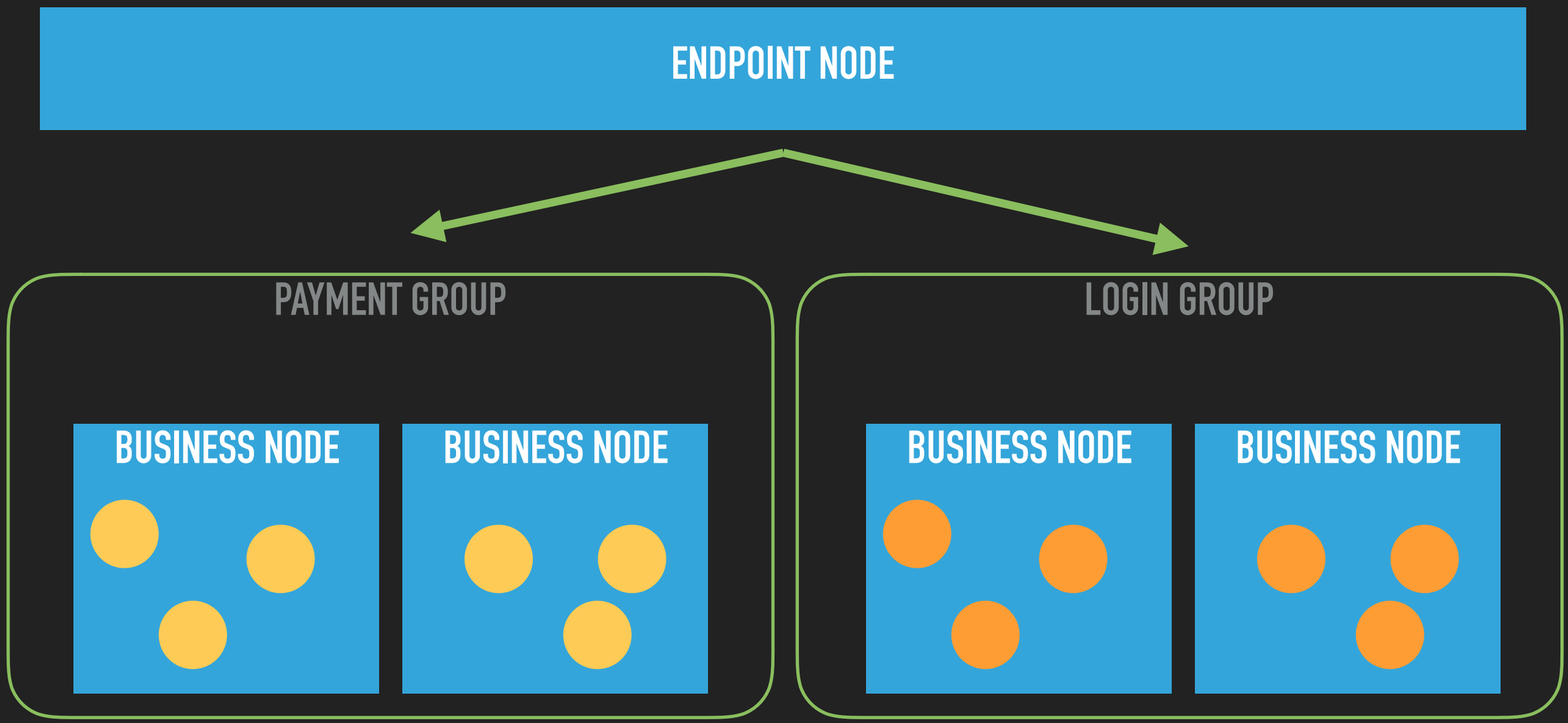
CLIENT



PERFORMANCE



PERFORMANCE



PERFORMANCE

ENDPOINT NODE

Stateless

PAYMENT GROUP

BUSINESS NODE

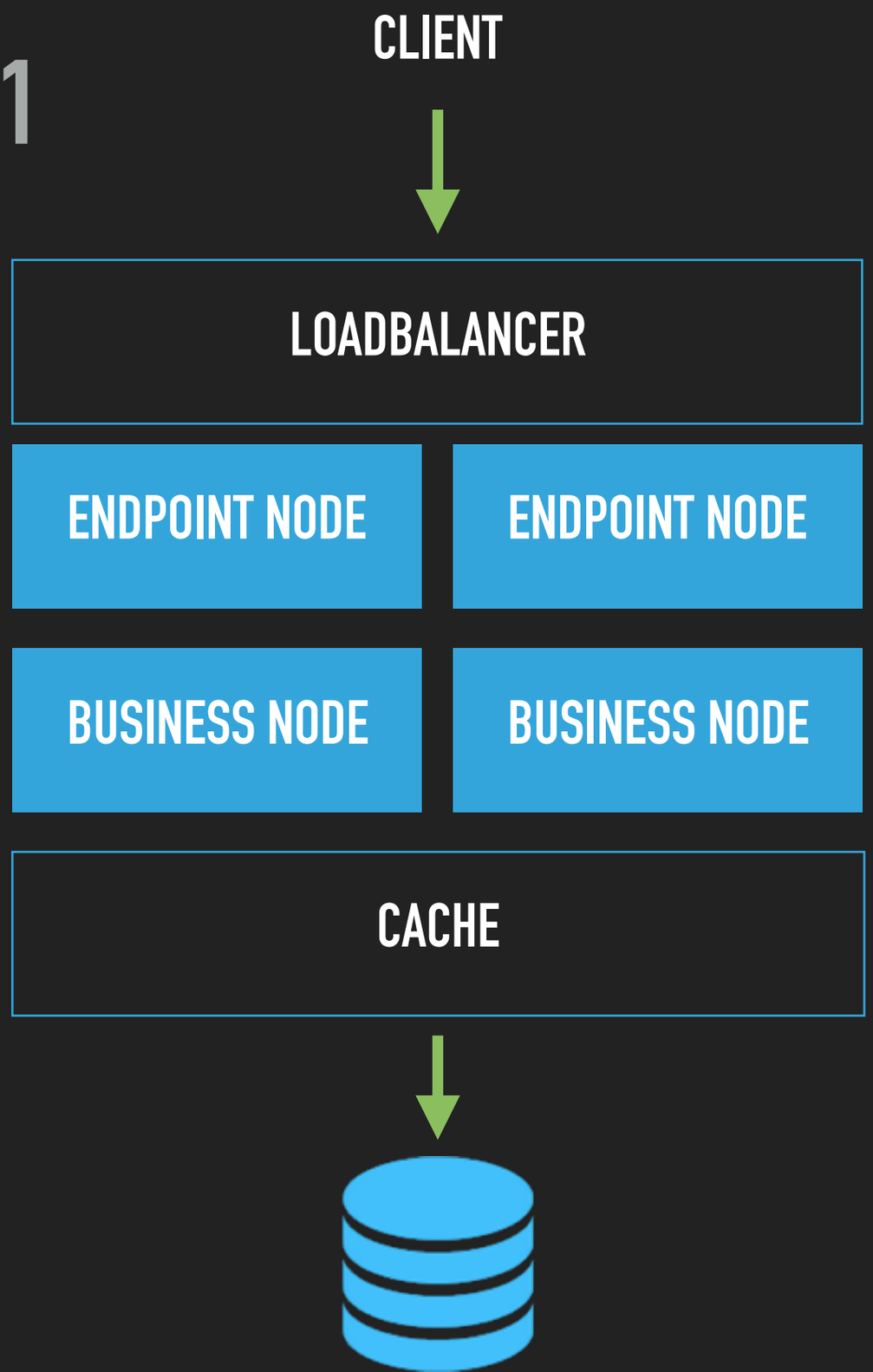
BUSINESS NODE

OGIN GROUP

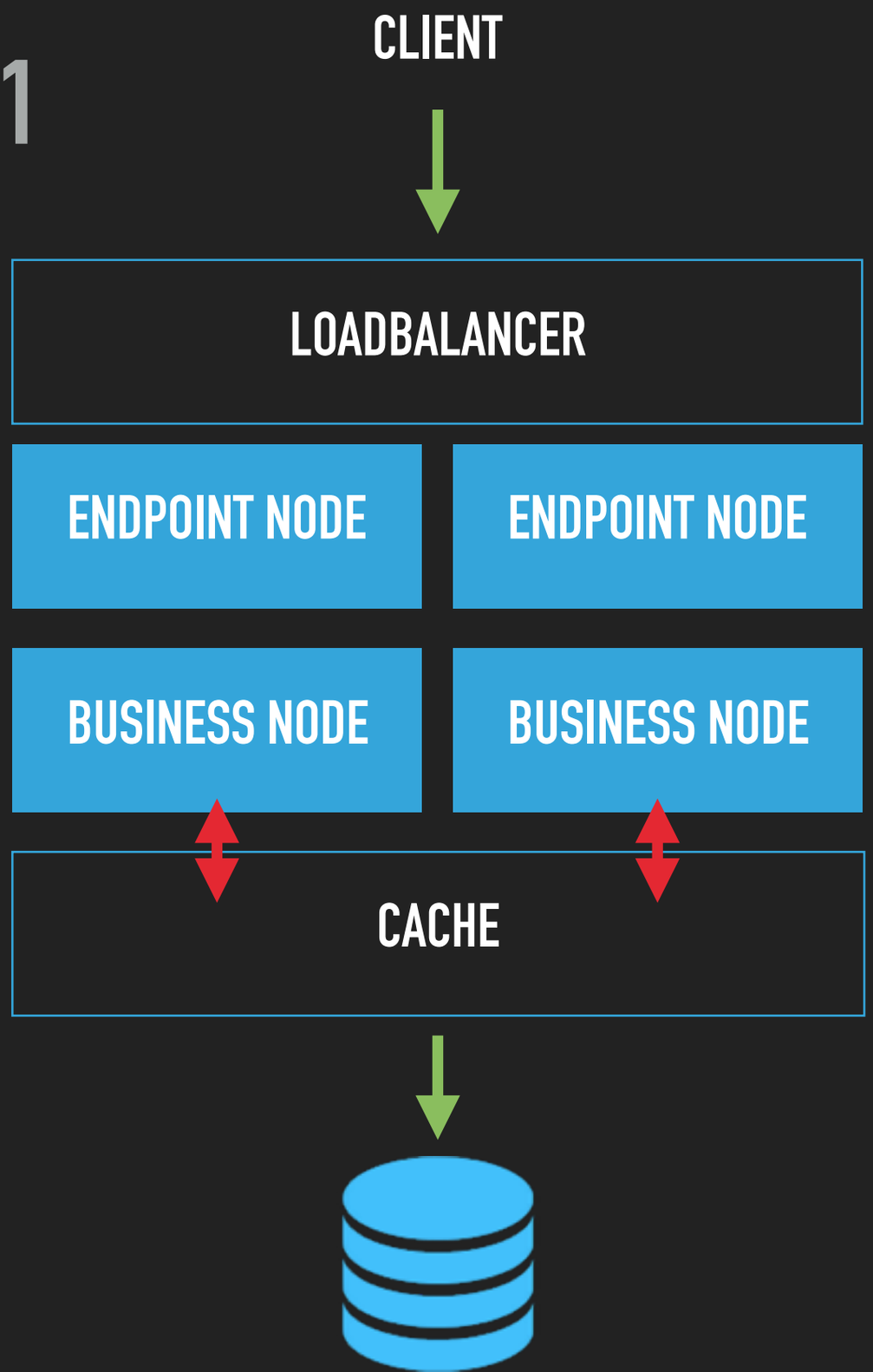
BUSINESS NODE

BUSINESS NODE

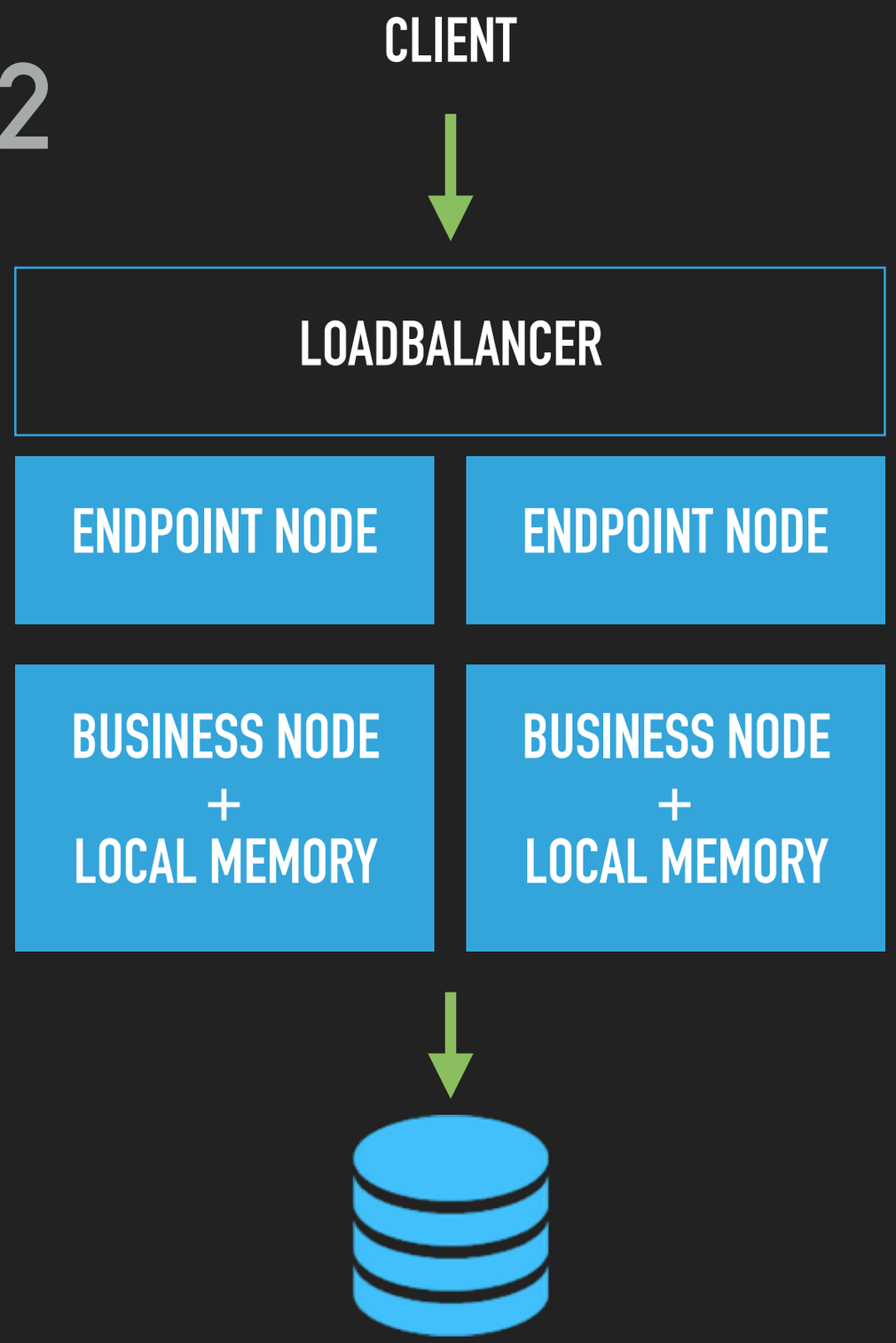
PERFORMANCE
STATEFUL #1



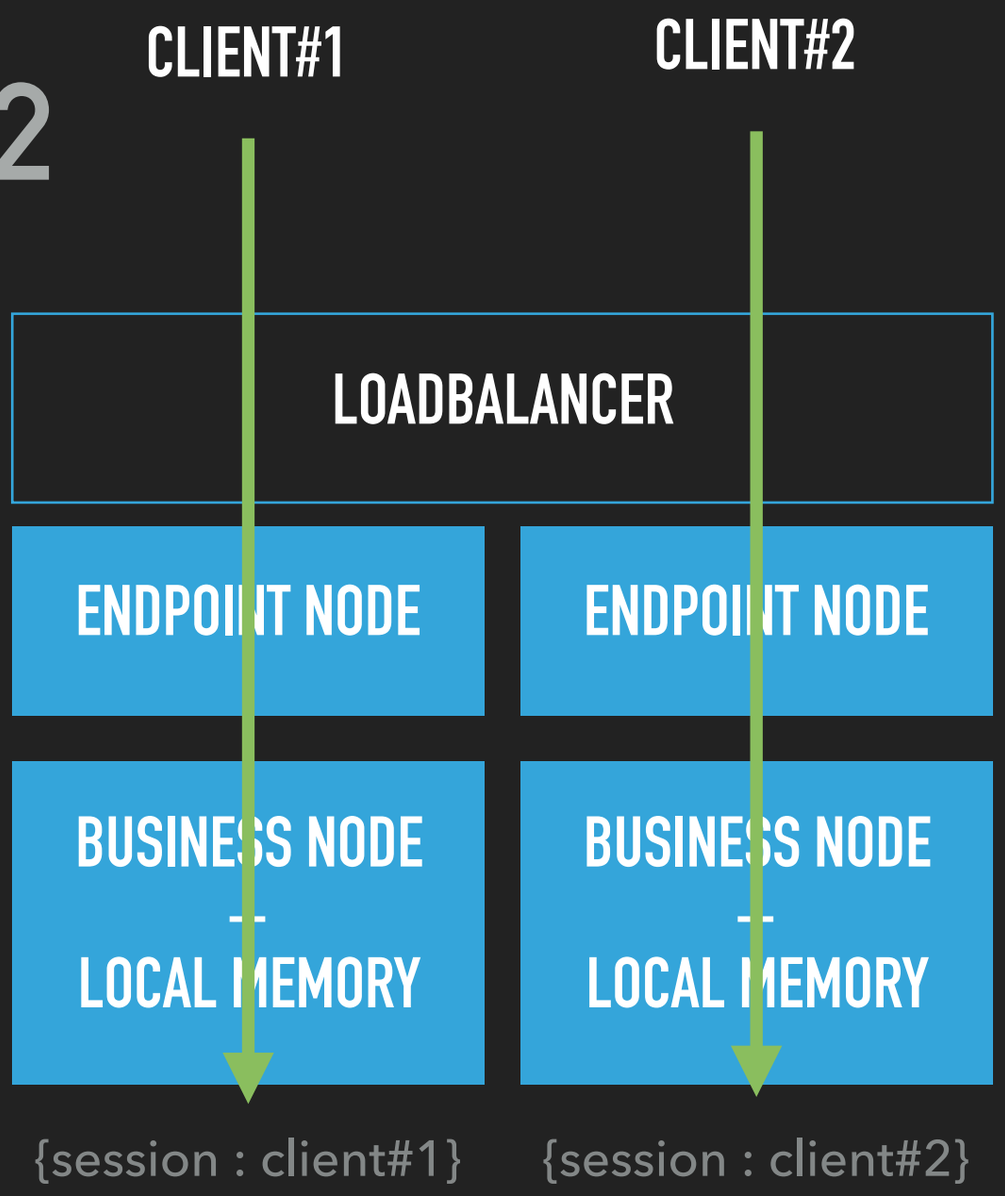
PERFORMANCE
STATEFUL #1



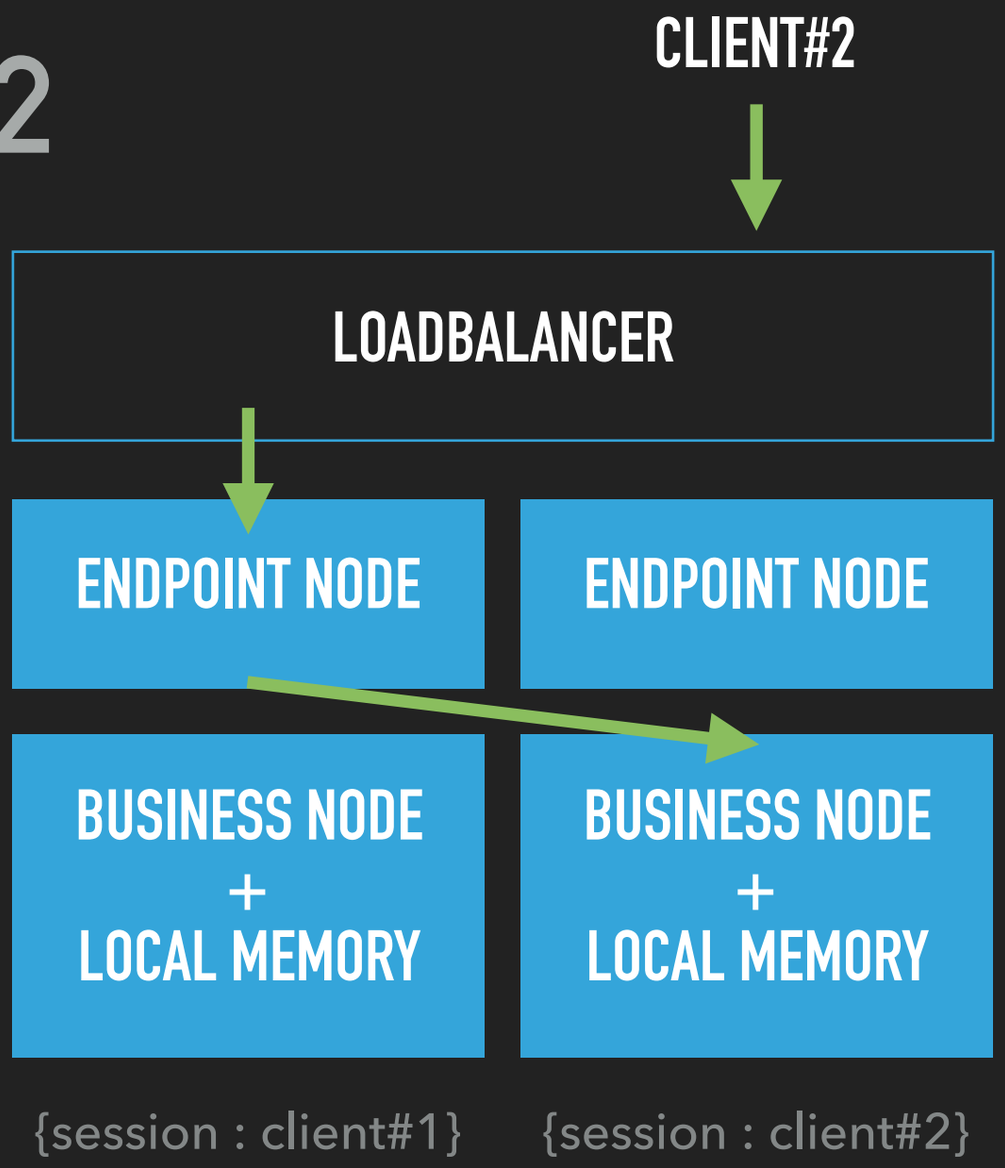
PERFORMANCE
STATEFUL #2



PERFORMANCE
STATEFUL #2



PERFORMANCE
STATEFUL #2

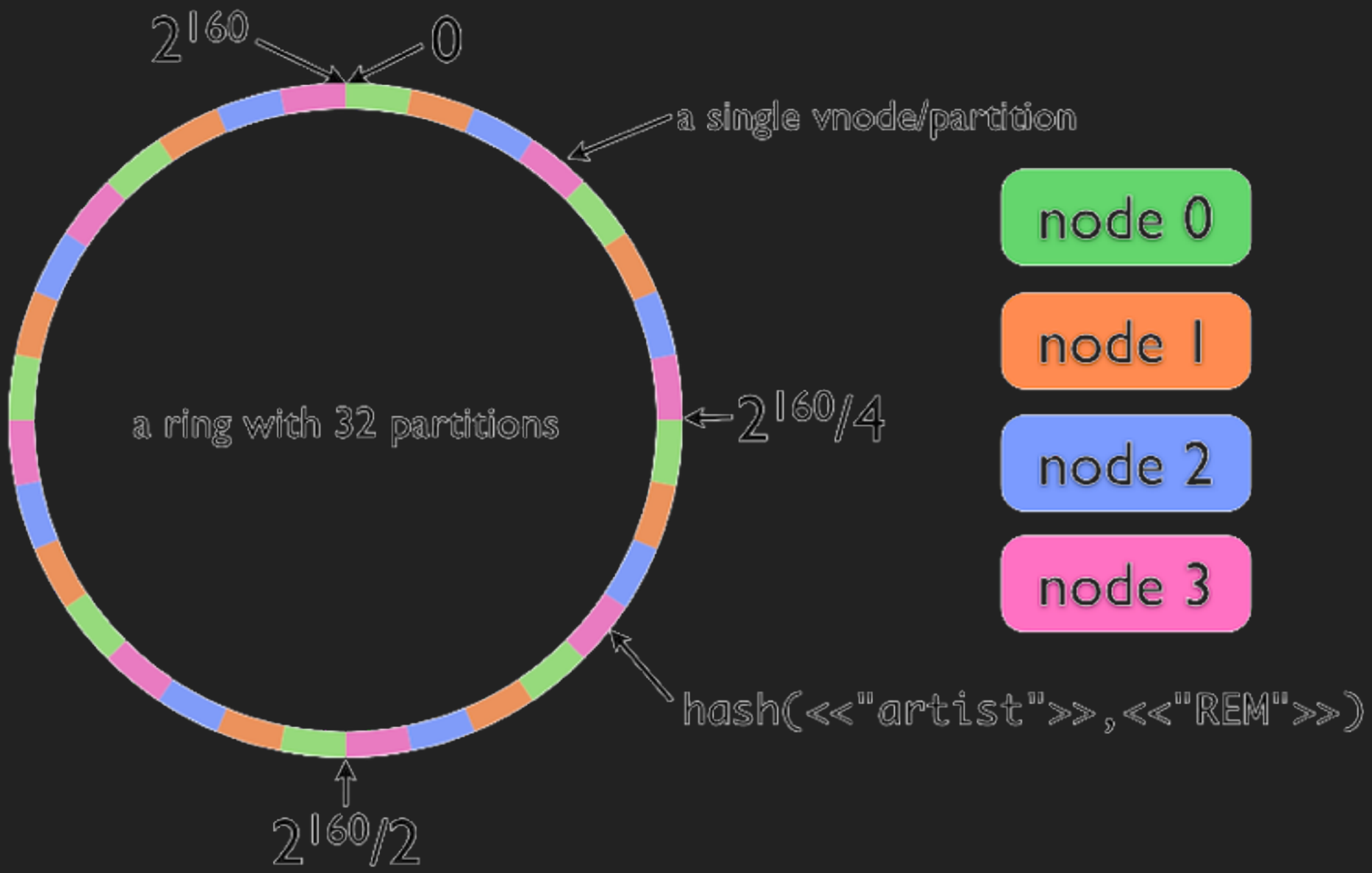


PERFORMANCE

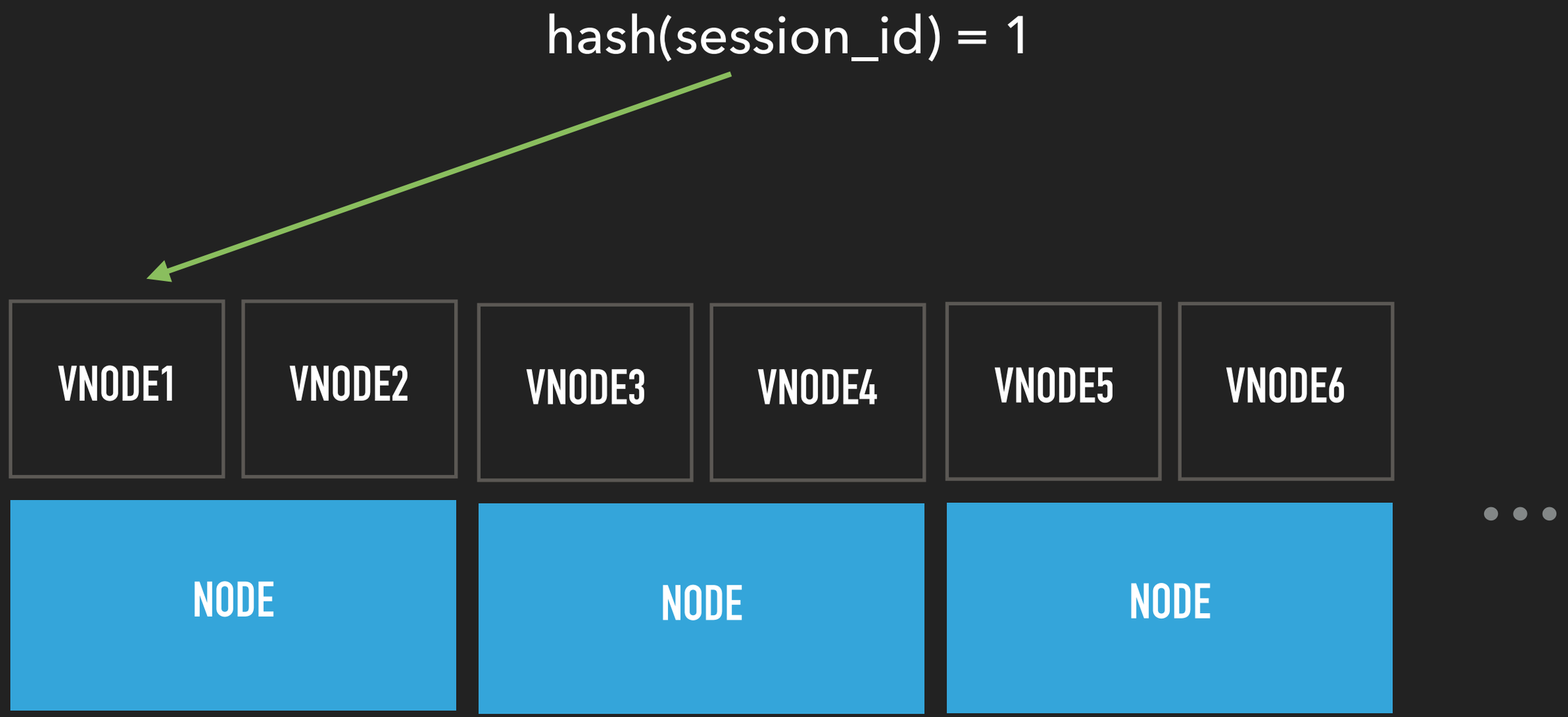
STATEFUL #2

RIAK_CORE

PERFORMANCE
STATEFUL #2



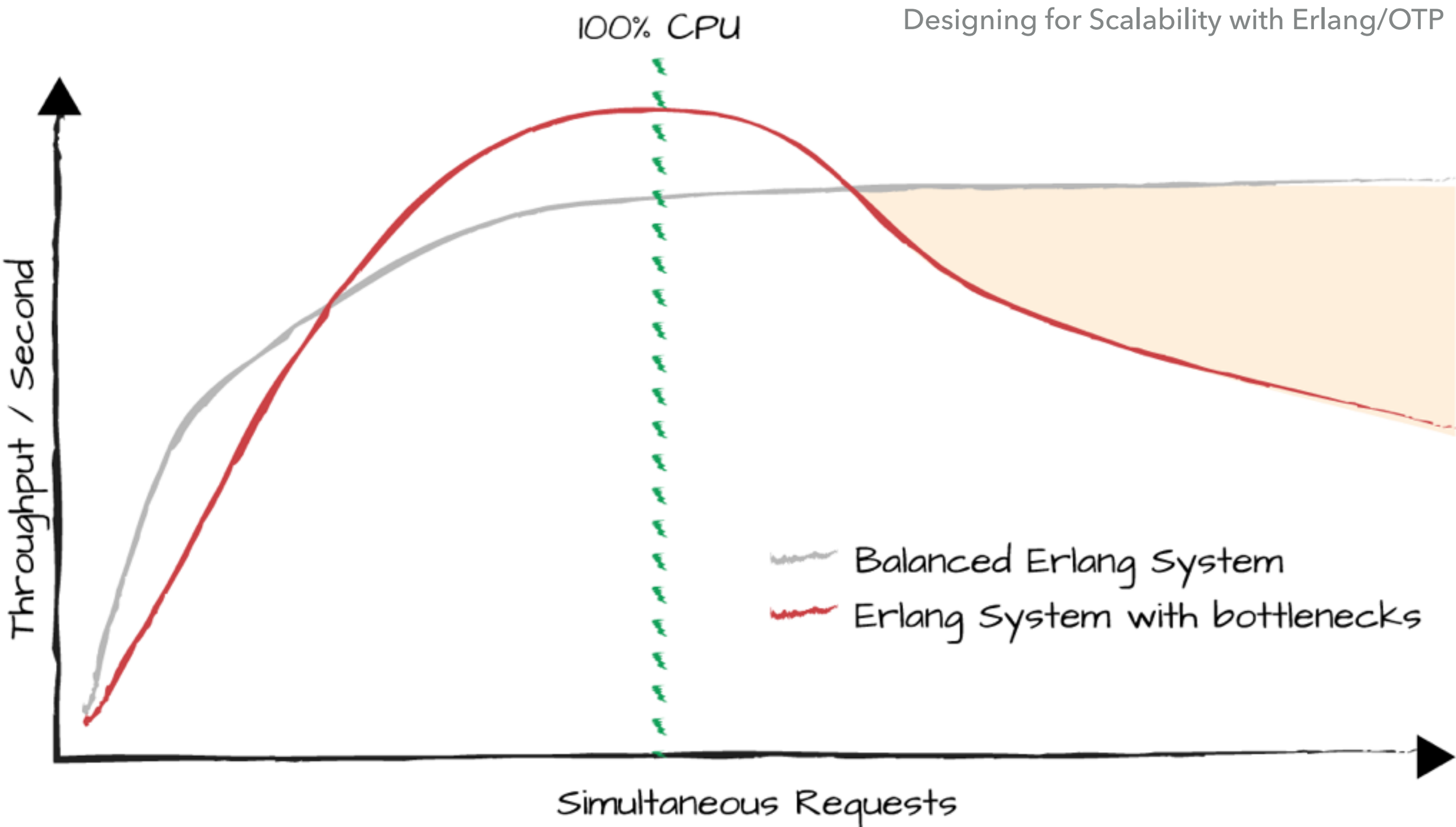
PERFORMANCE
STATEFUL #2



https://www.youtube.com/watch?v=sYYOLaJ-VDQ

PERFORMANCE

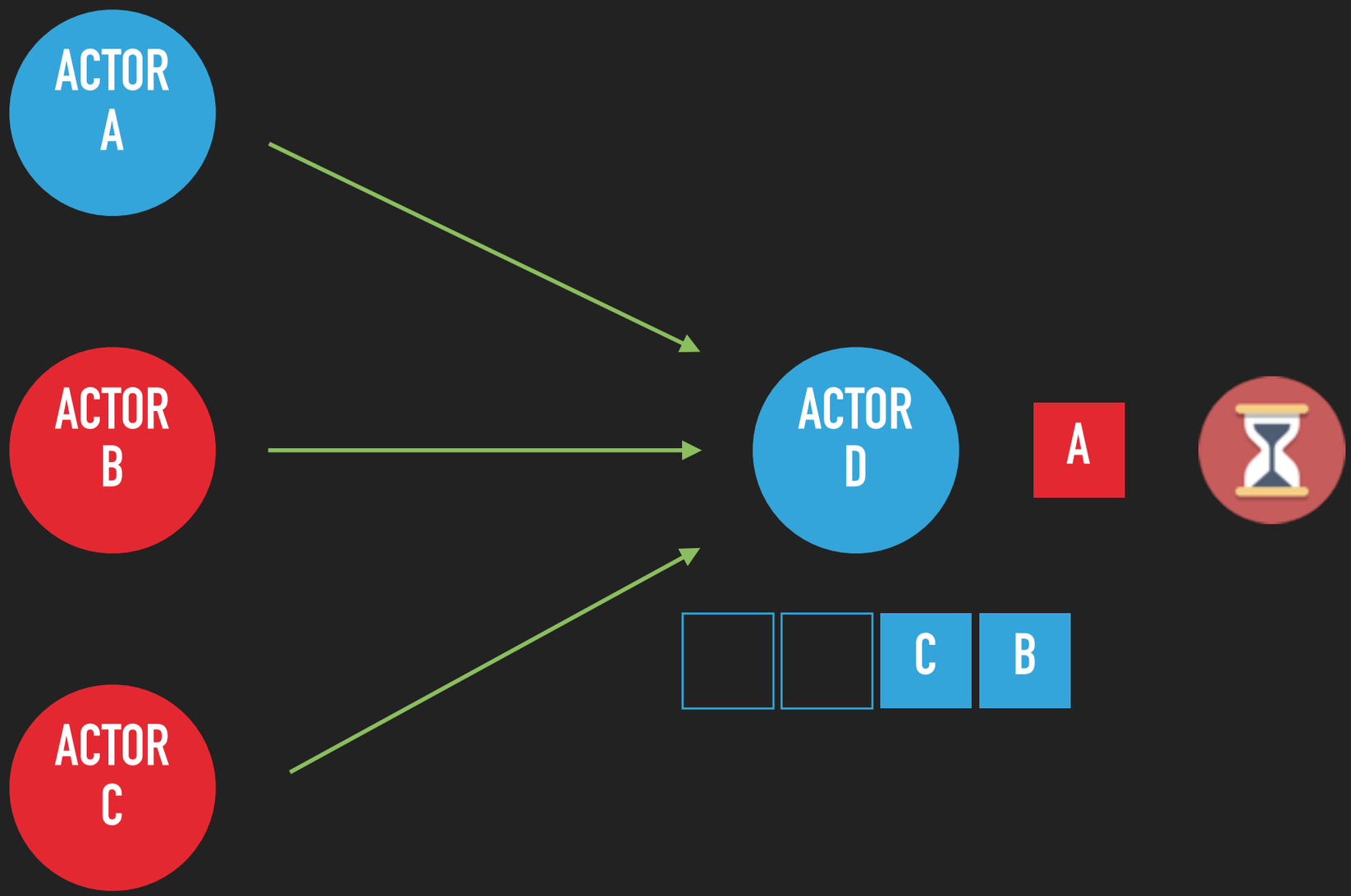
Designing for Scalability with Erlang/OTP



PERFORMANCE

SYNCHRONOUS CALLS

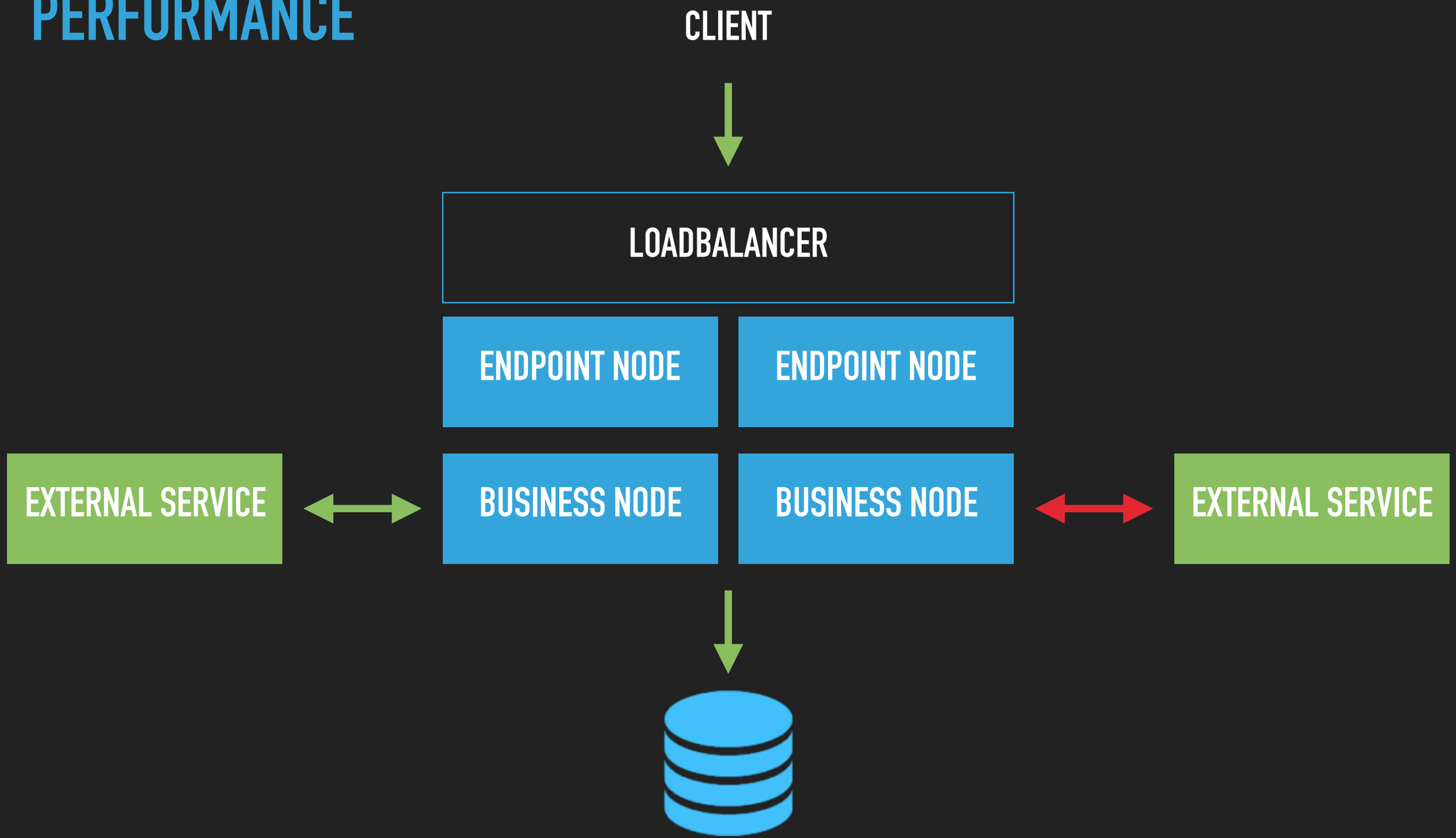
PERFORMANCE
SYNCHRONOUS CALLS



PERFORMANCE

EXTERNAL SERVICE

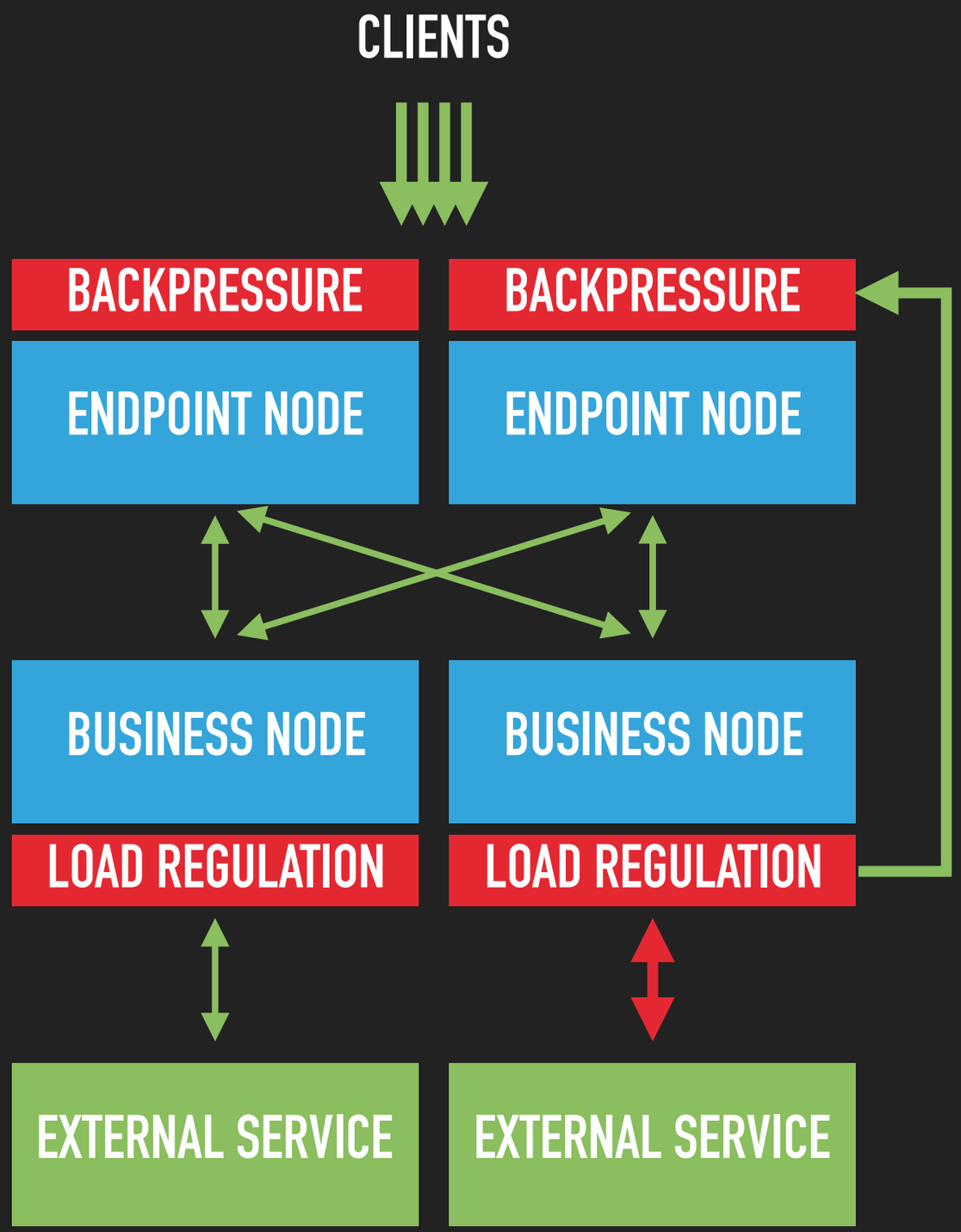
PERFORMANCE



PERFORMANCE

LOAD REGULATION AND BACKPRESSURE

PERFORMANCE

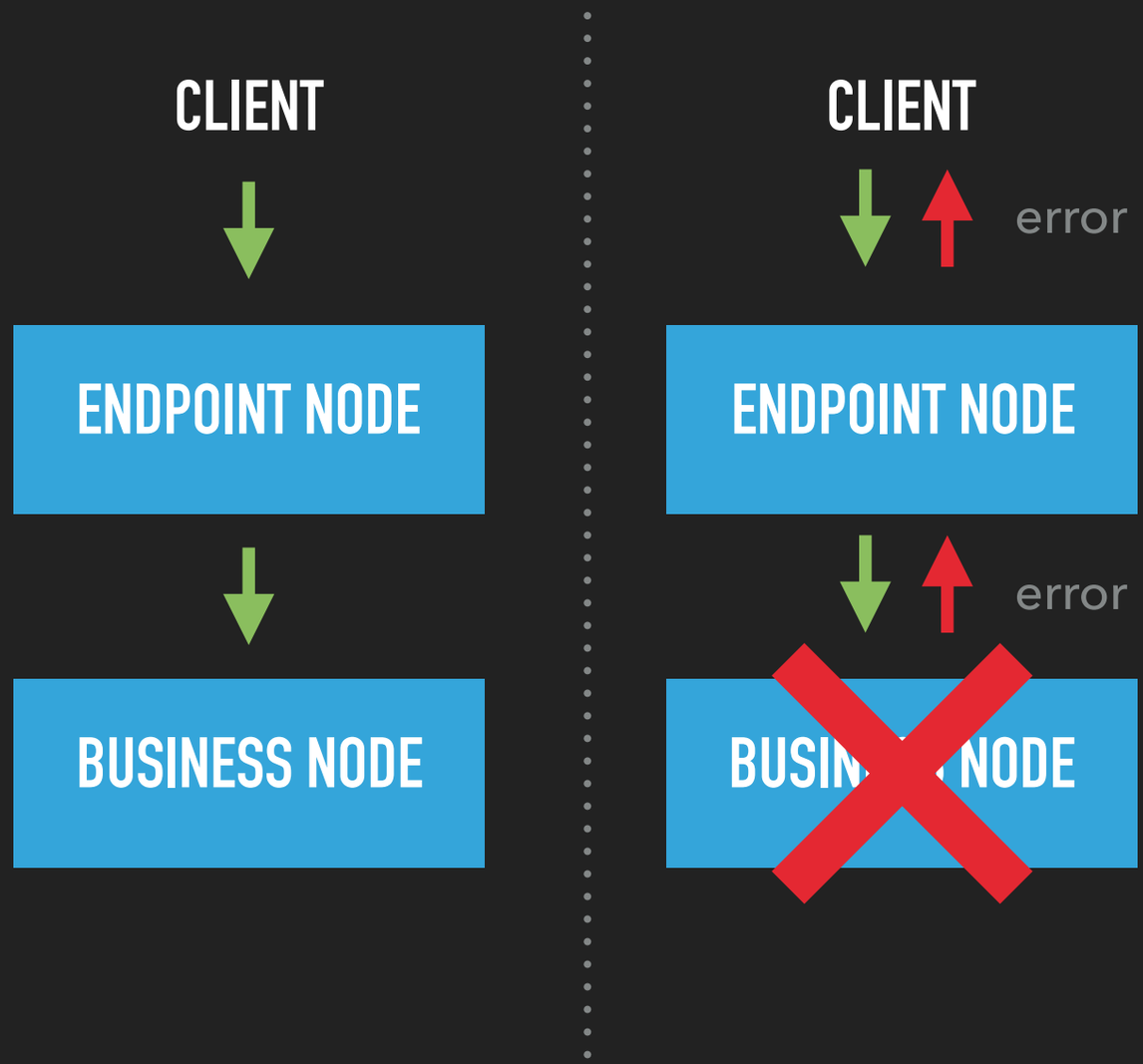


AVAILABILITY

- Fault Tolerance
- Resilience
- Reliability

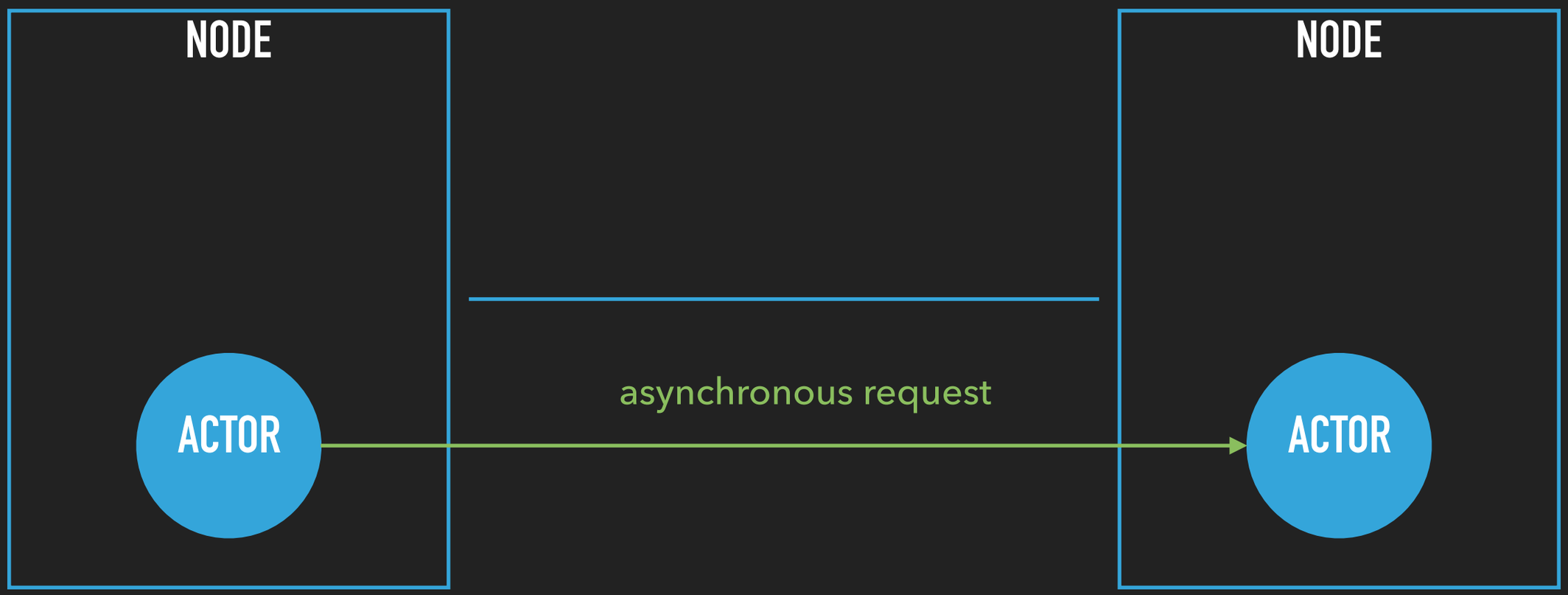
AVAILABILITY

FAULT TOLERANCE



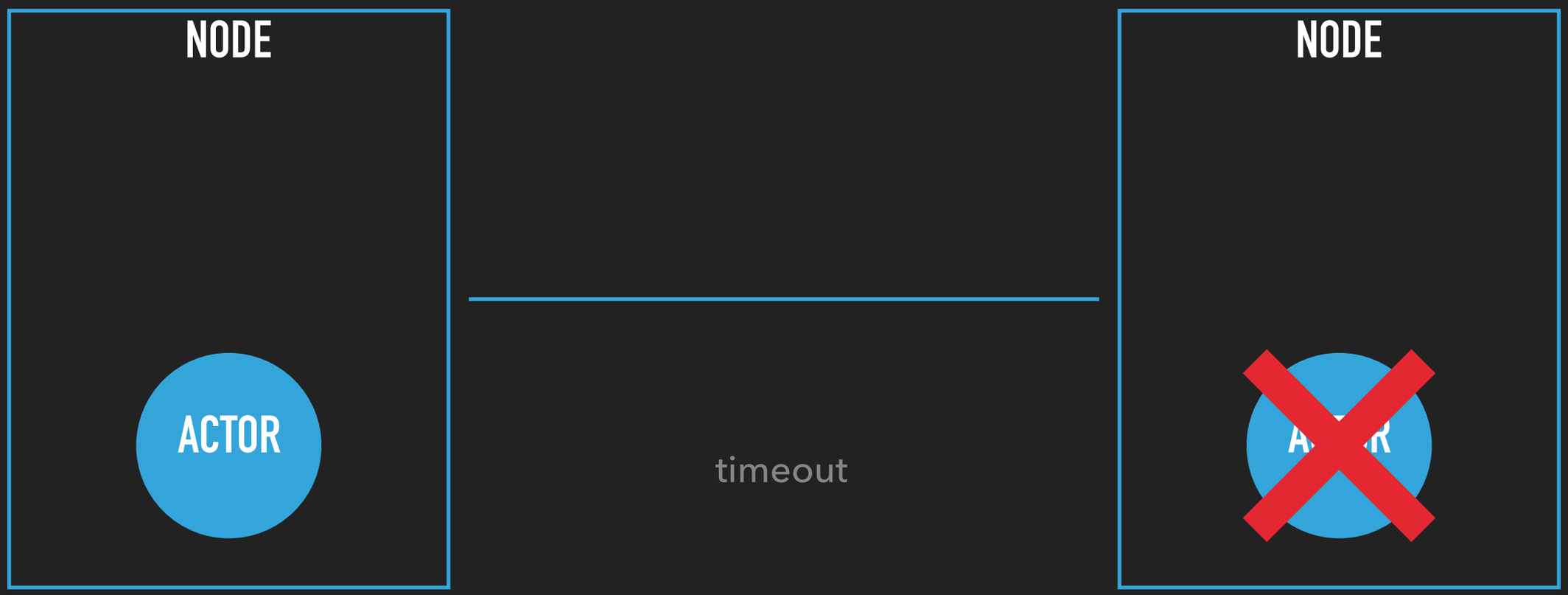
AVAILABILITY

FAULT TOLERANCE



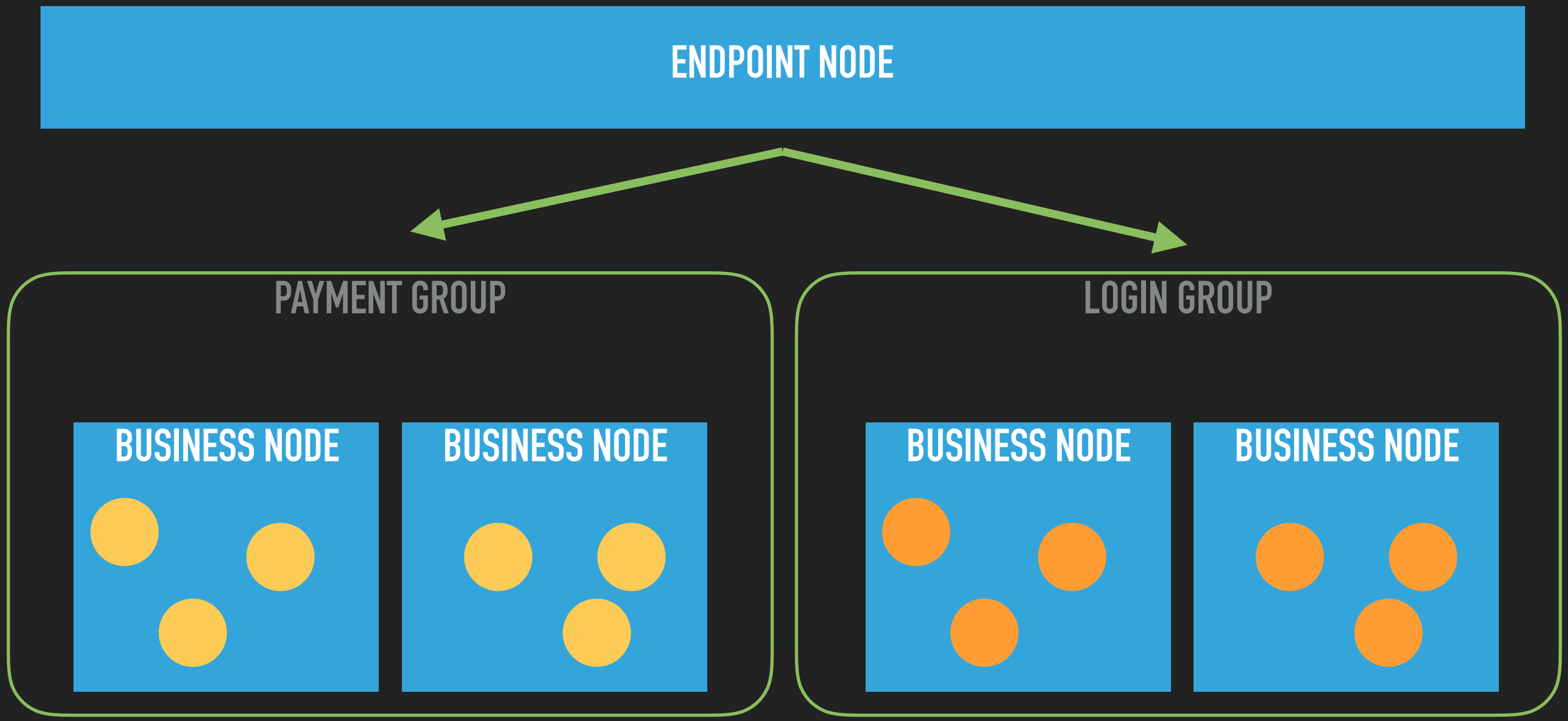
AVAILABILITY

FAULT TOLERANCE



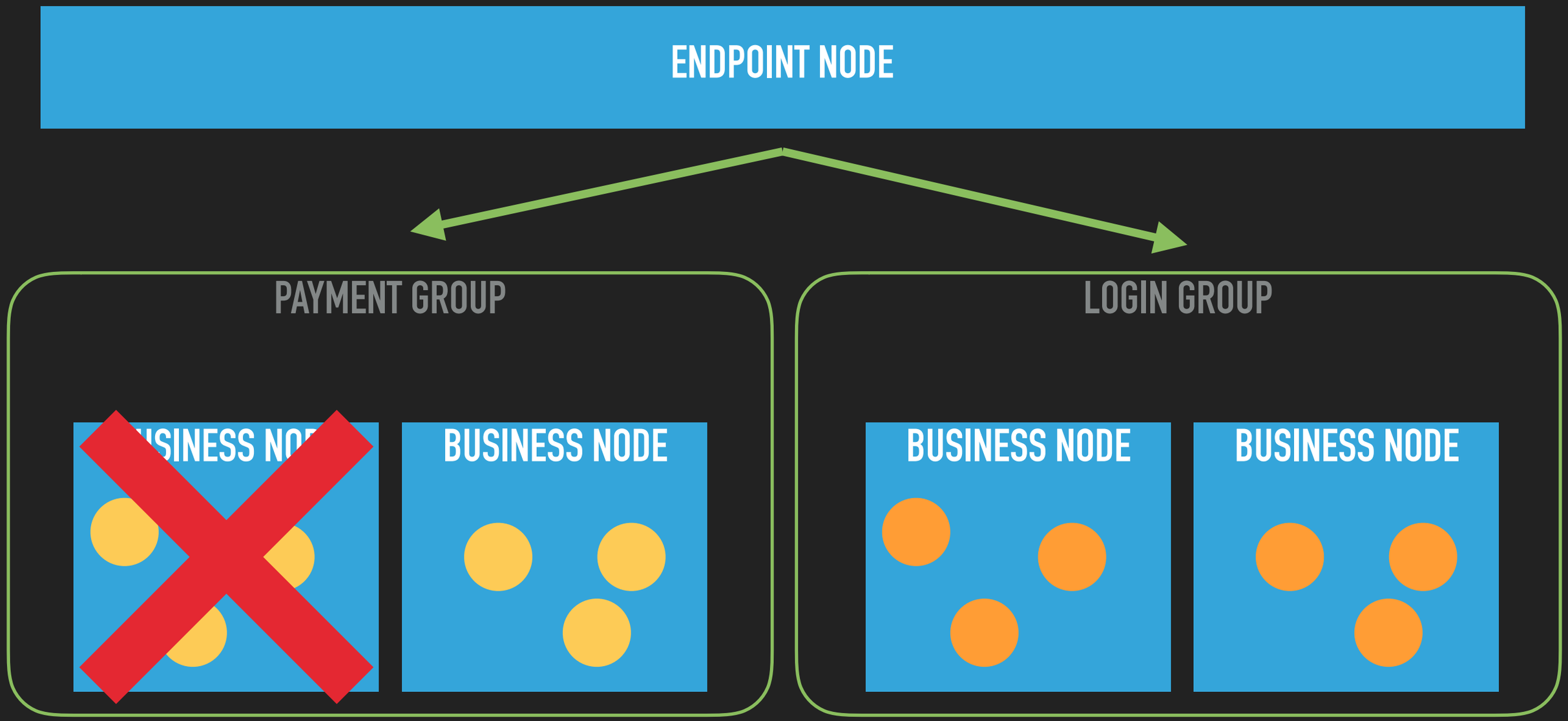
AVAILABILITY

RESILIENCE - NODE



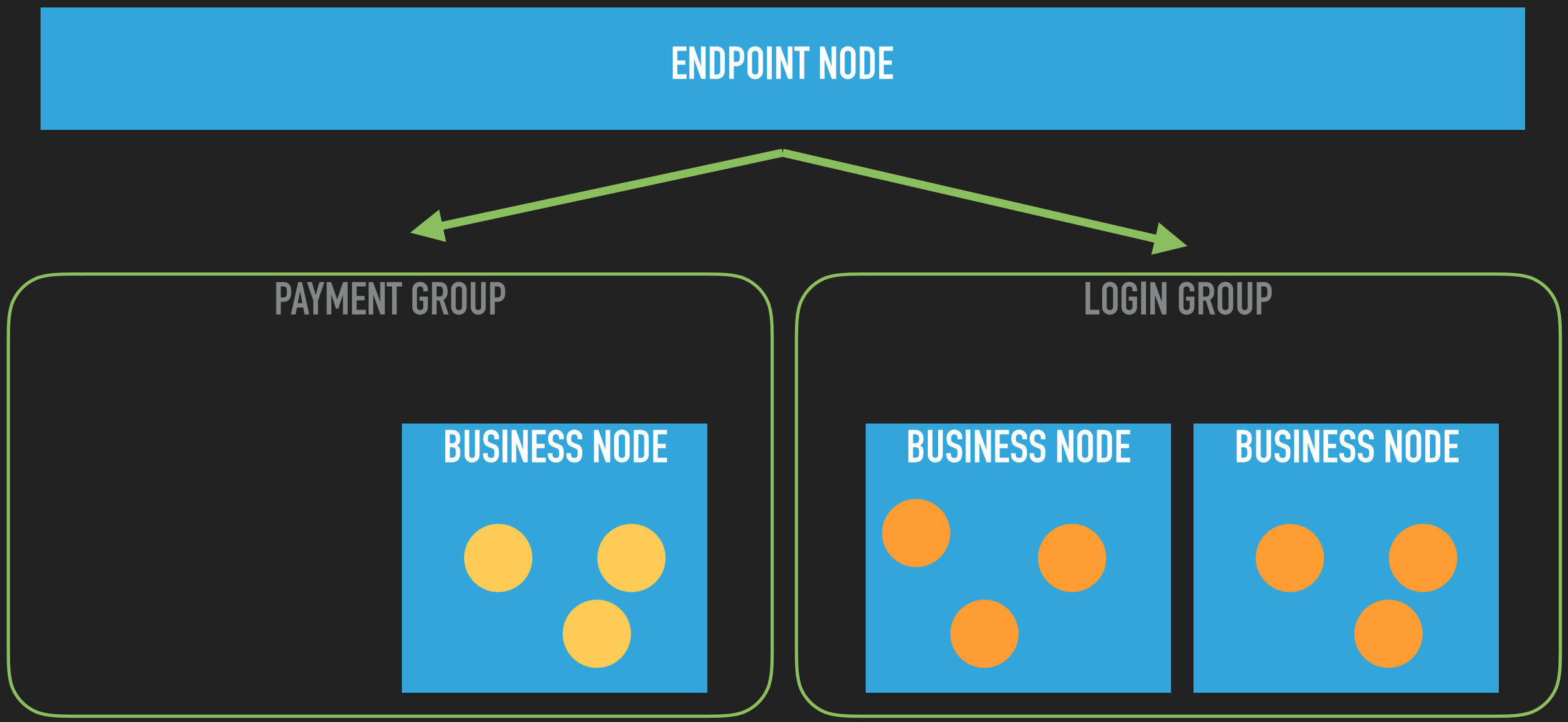
AVAILABILITY

RESILIENCE - NODE



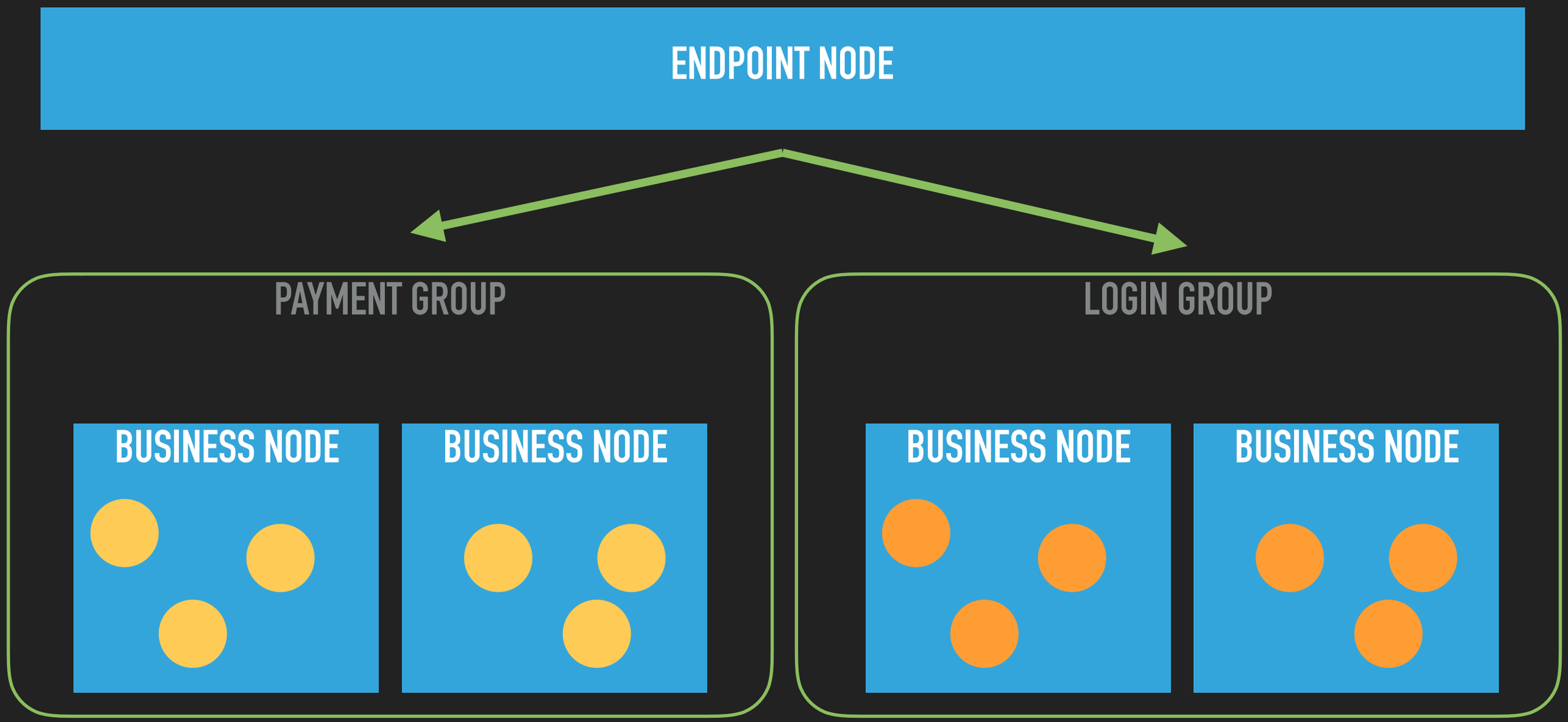
AVAILABILITY

RESILIENCE - NODE



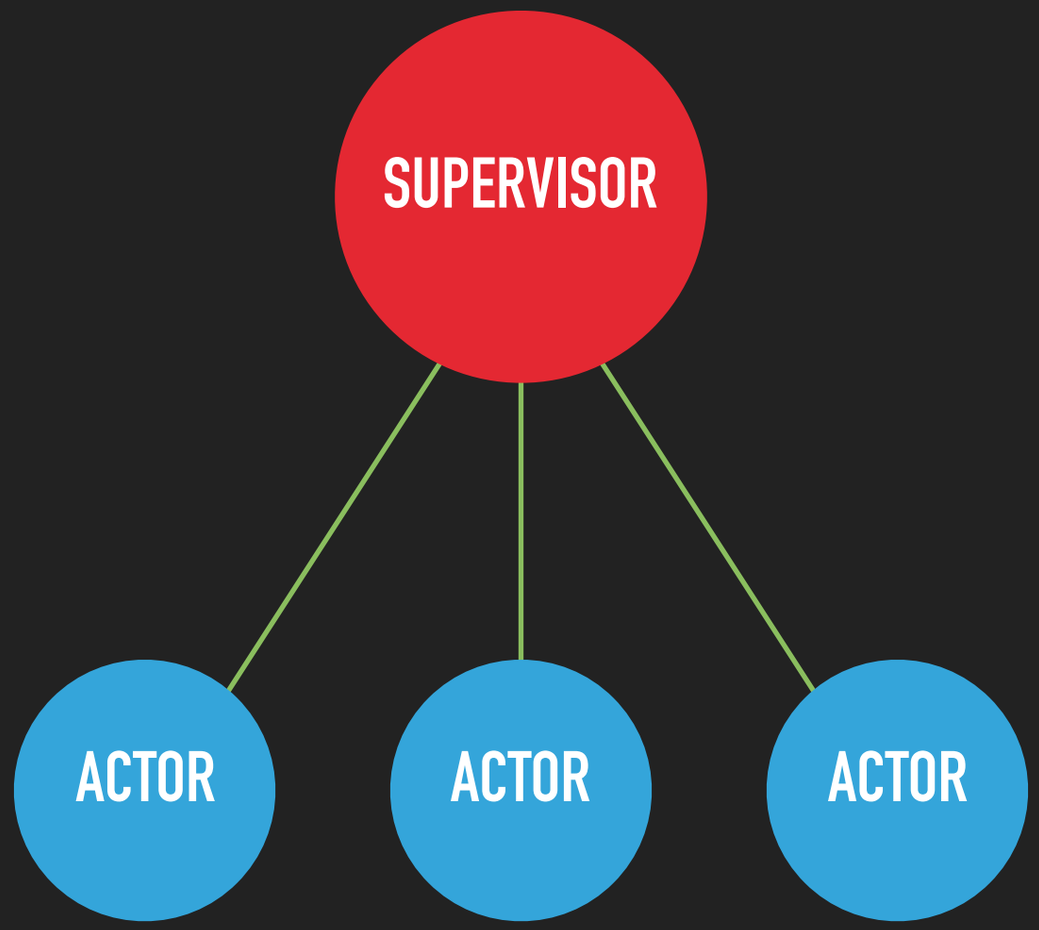
AVAILABILITY

RESILIENCE - NODE



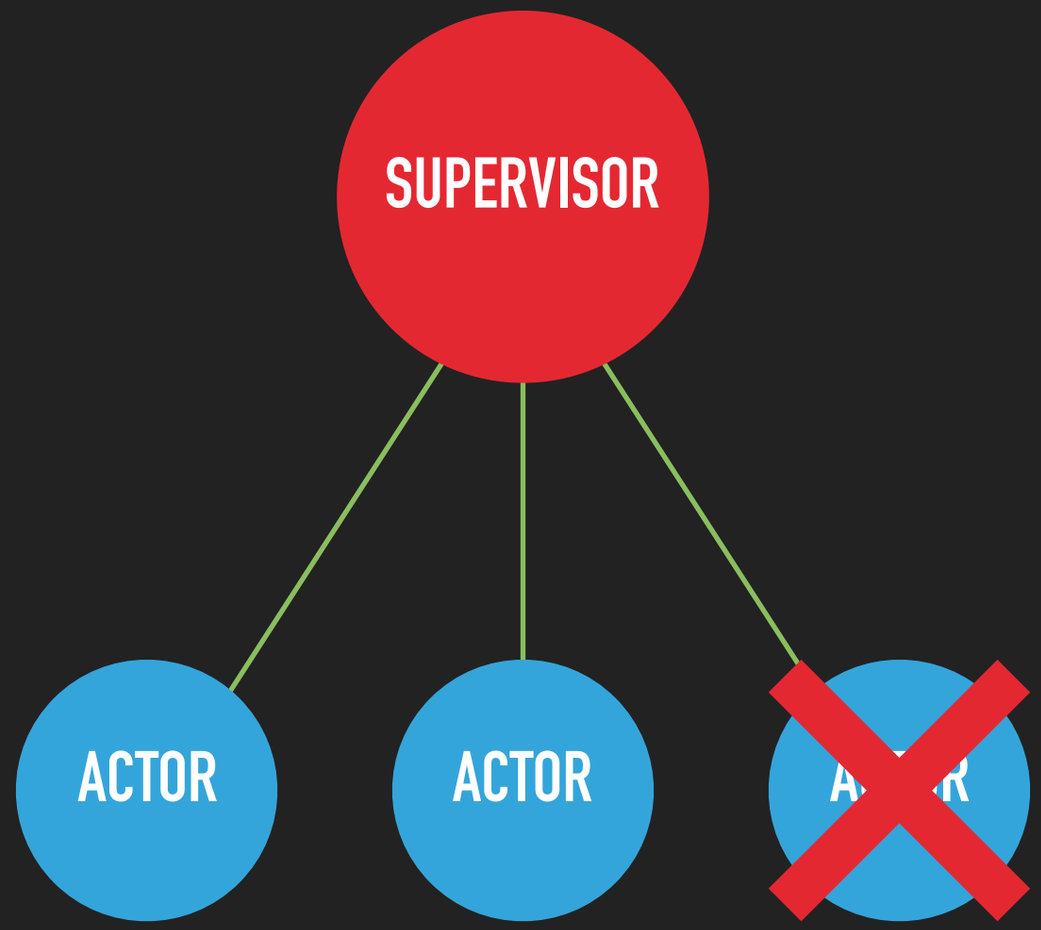
AVAILABILITY

RESILIENCE - ACTOR



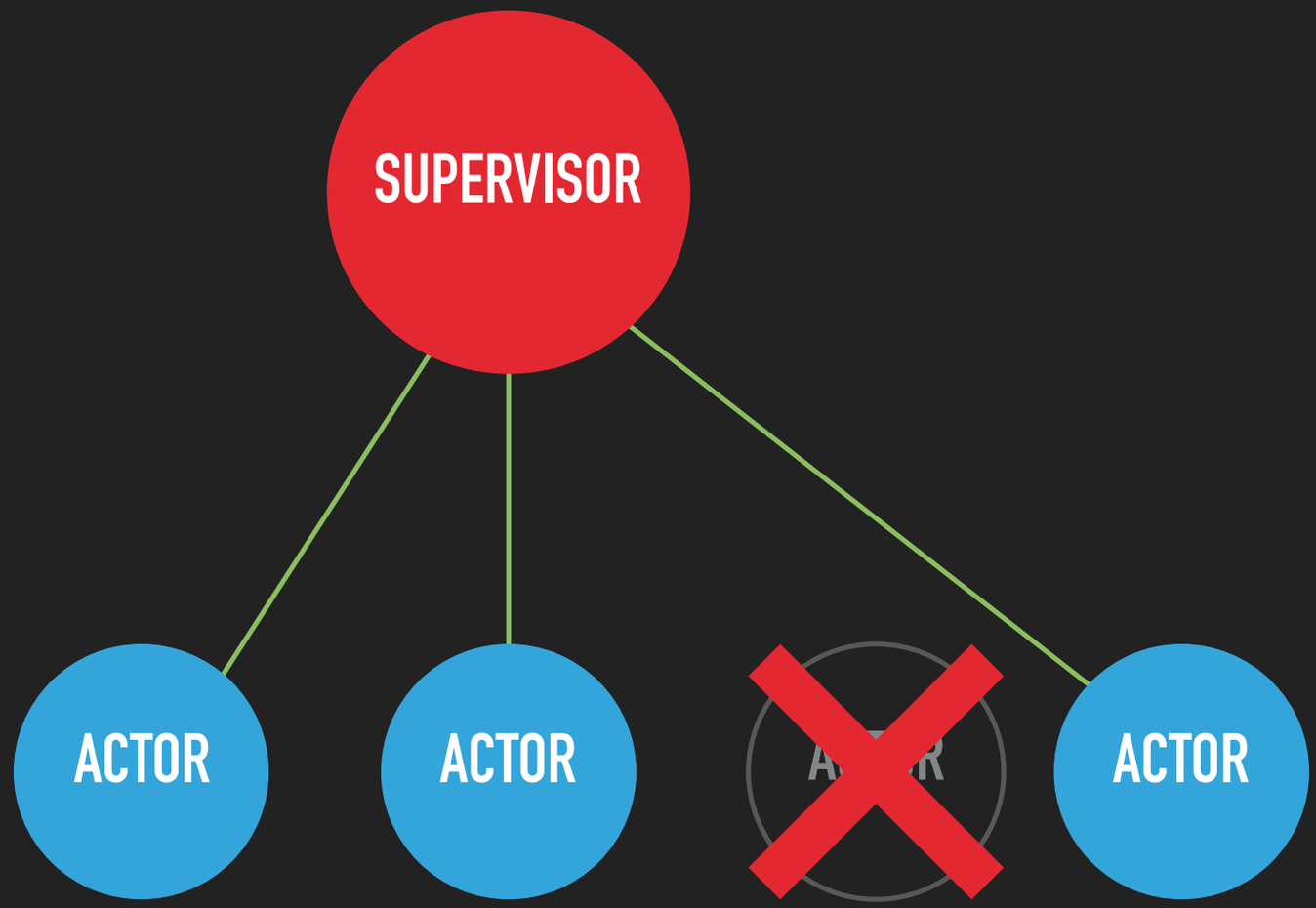
AVAILABILITY

RESILIENCE - ACTOR



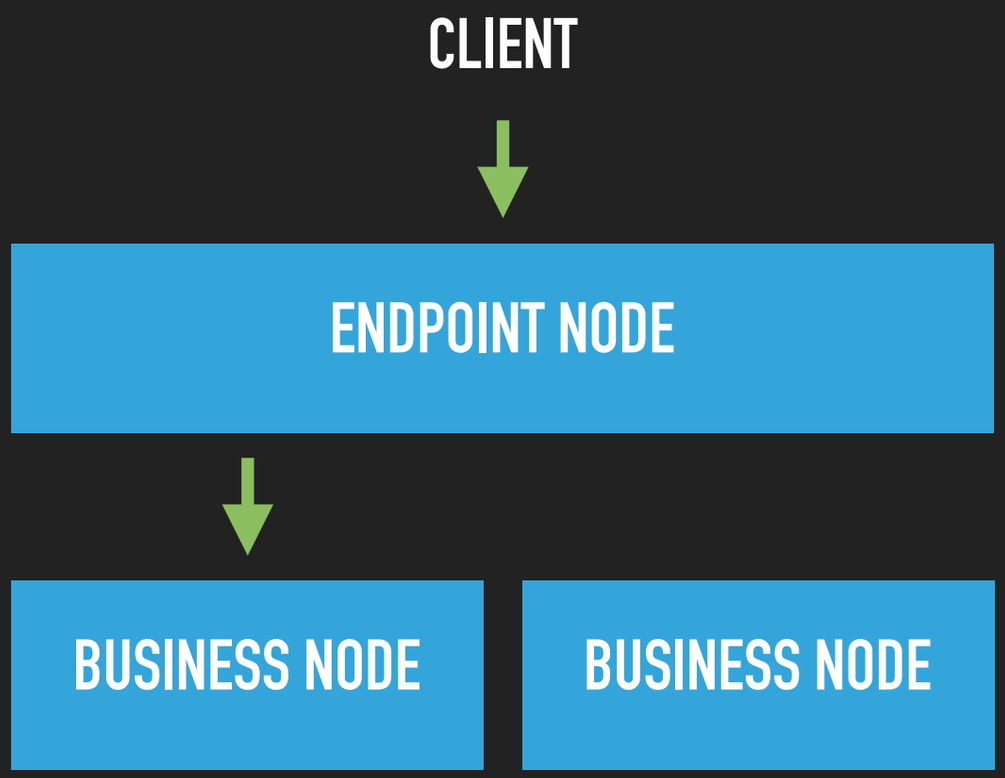
AVAILABILITY

RESILIENCE - ACTOR



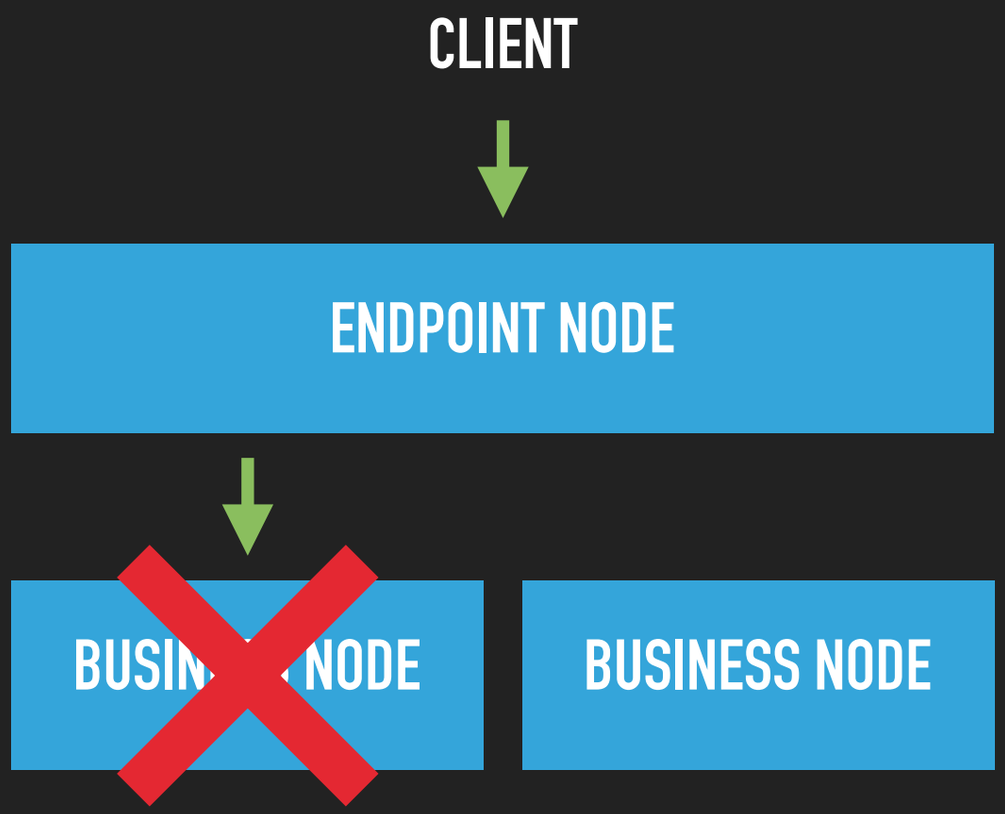
AVAILABILITY

RELIABILITY



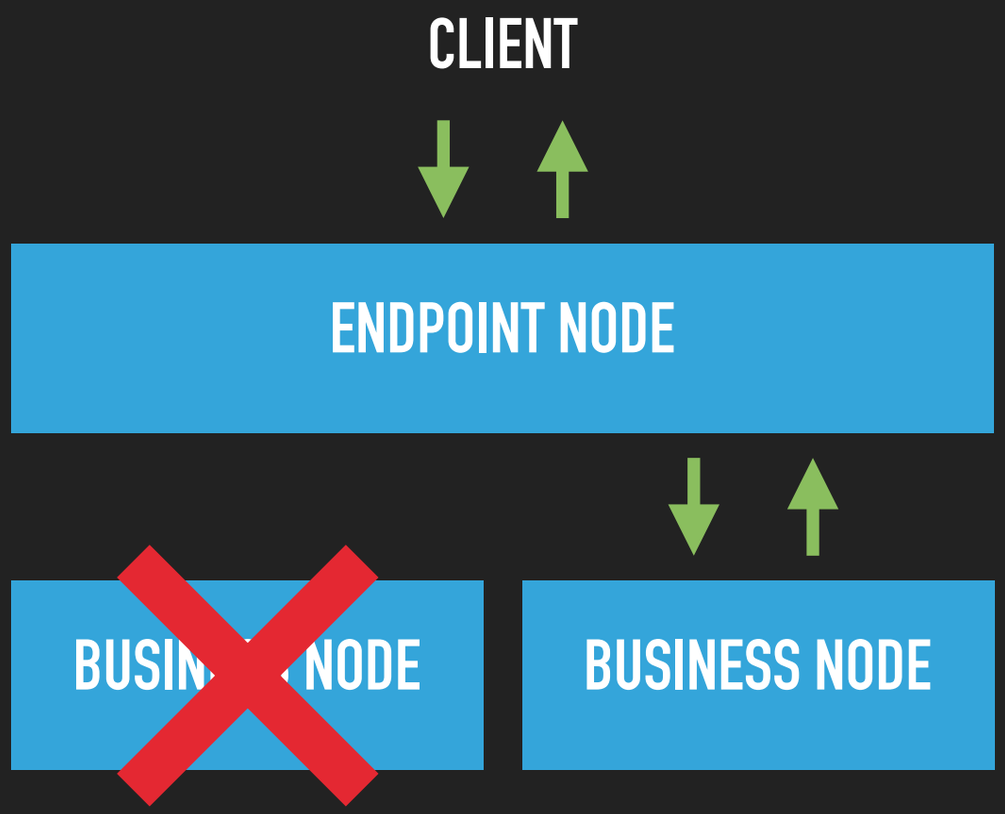
AVAILABILITY

RELIABILITY



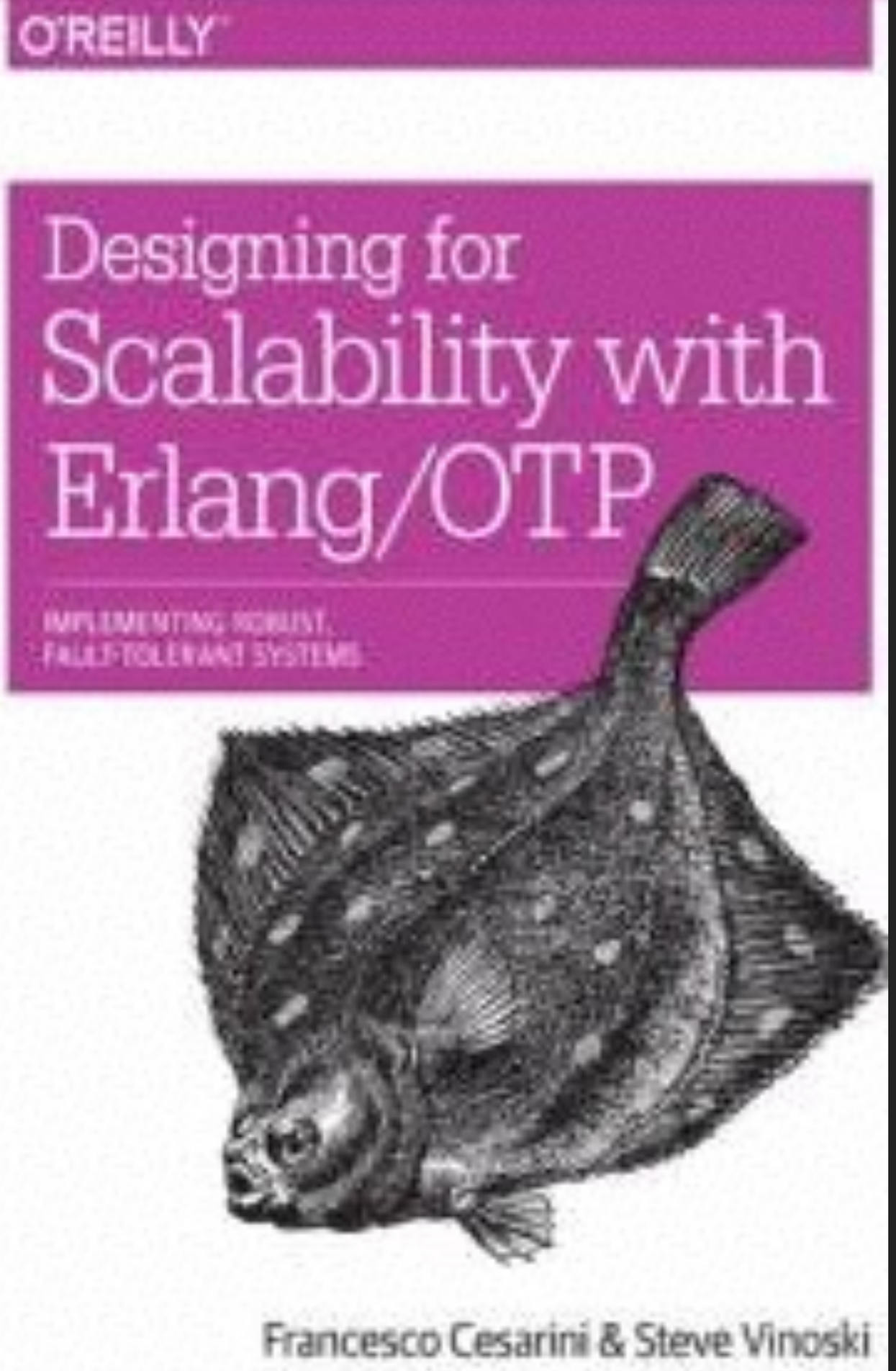
AVAILABILITY

RELIABILITY



WRAPPING UP

NO SILVER BULLET



Designing for Scalability with Erlang/OTP

The Pragmatic Programmers

Programming Elixir 1.3

Functional
|> Concurrent
|> Pragmatic
|> Fun

Dave Thomas

Foreword by
José Valim,
Creator of Elixir

Programming Elixir 1.3

Erlang/Elixir Korea User Group

https://www.facebook.com/groups/elixir.korea/

REFERENCES

- Books
  - Designing for Scalability with Erlang/OTP
  - Programming Elixir 1.3 Reliability
  - The Little Elixir & OTP Guidebook
- Videos
  - Experimenting with Superpowered Web Services: Phoenix on Riak_Core https://www.youtube.com/watch?v=sYYOLaJ-VDQ
  - Erlang Factory SF 2016 - Concurrency + Distribution = Scalability + Availability, a Journey Architecting Erlang Systems https://www.youtube.com/watch?v=_IZMQMuphfo

THANK YOU...