ERLANG DISTRIBUTION

GOING BEYOND THE FULLY CONNECTED MESH

DIPL. PHYS. PEER STRITZINGER GMBH

LIGHTKONE

Lightweight computation for networks at the edge

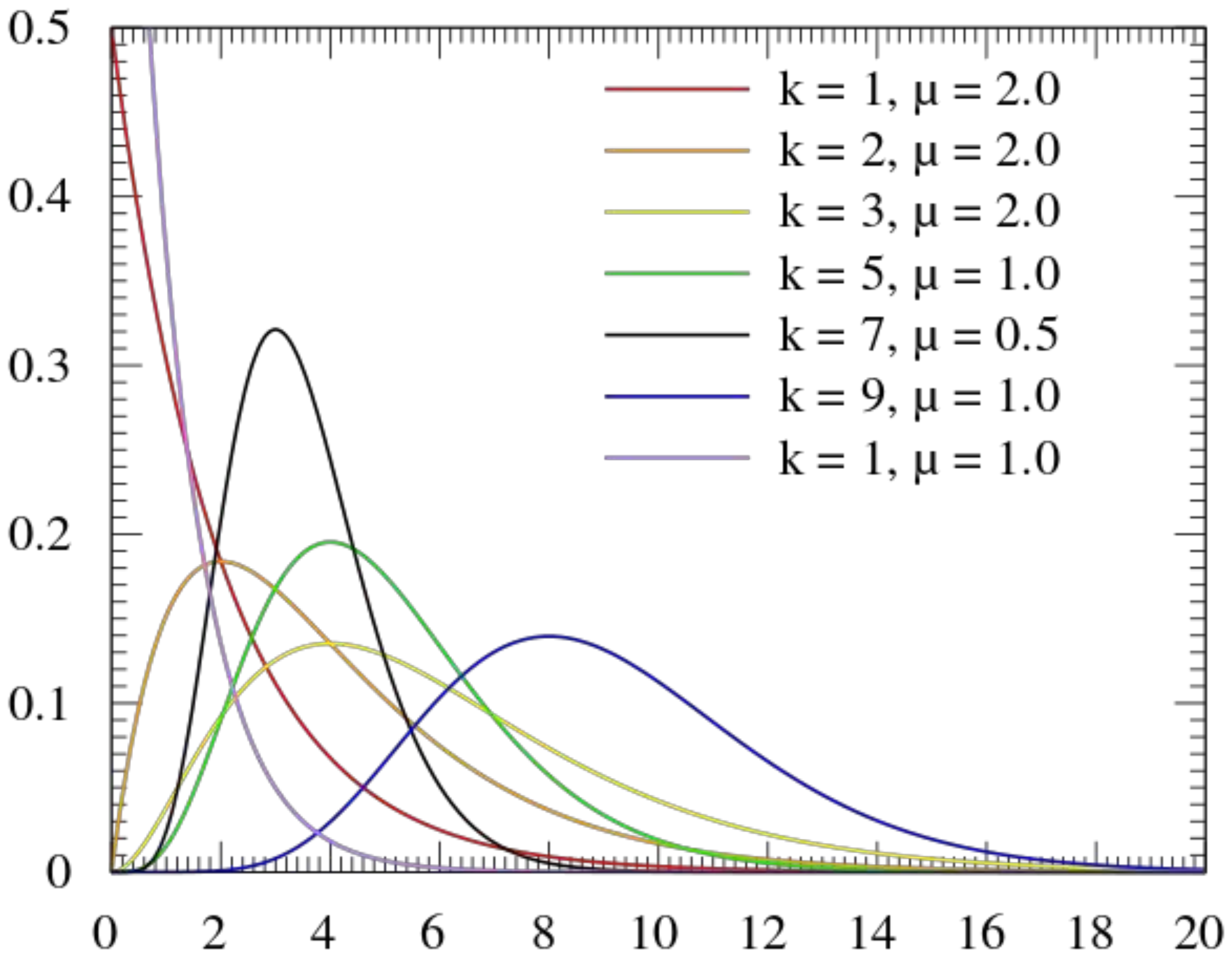
Funded by the Horizon 2020 Framework Programme of the European Union

- Computation at the Edge
- CRDTs
- Gossip protocols

DIPL. PHYS. PEER STRITZINGER GMBH

ERLANG DISTRIBUTION

$$f(x;k,\mu) = \frac{x^{k-1} e^{-\frac{x}{\mu}}}{\mu^k (k-1)!} \quad \text{for } x, \mu \geq 0.$$



ERLANG DISTRIBUTION

- Transparent distribution protocol
- Send Messages
- Link Processes
- Monitor Processes

PROBLEMS

- Default is fully connected mesh
- Fully connected mesh doesn't scale well
  - Hidden nodes can help, but is not a solution
- Global process registry requires fully connected mesh

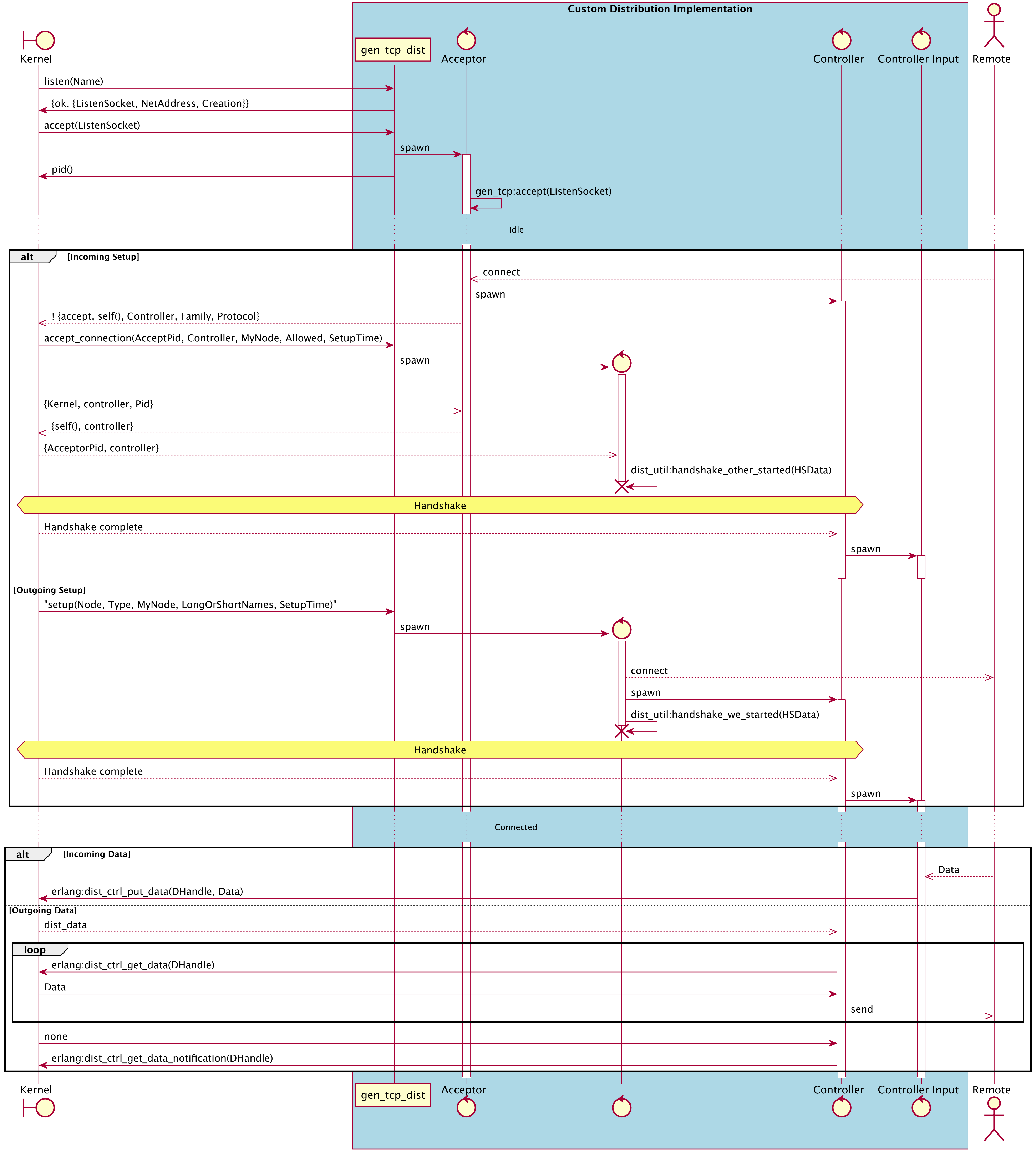
UDP DISTRIBUTION PROTOTYPE

DIPL. PHYS.
PEER STRITZINGER GMBH

IMPLEMENTATION

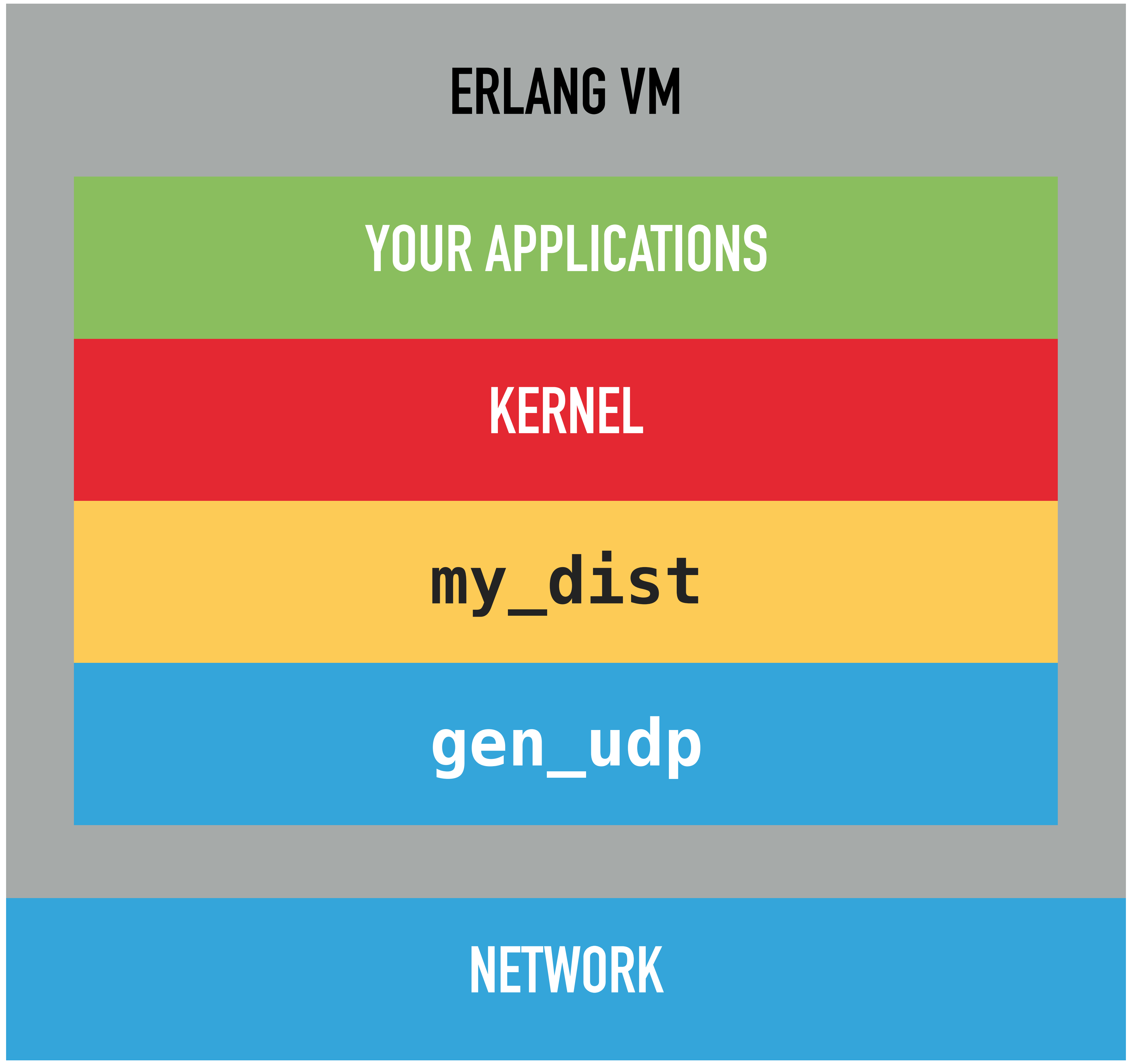
- Started out based on our work on working around head-of-line blocking
    - Alternative ways to avoid the problem
- Stepping stone towards experience with building multiple distribution implementations
- Learn how to support less connection oriented ways of implementing distribution

Uh...



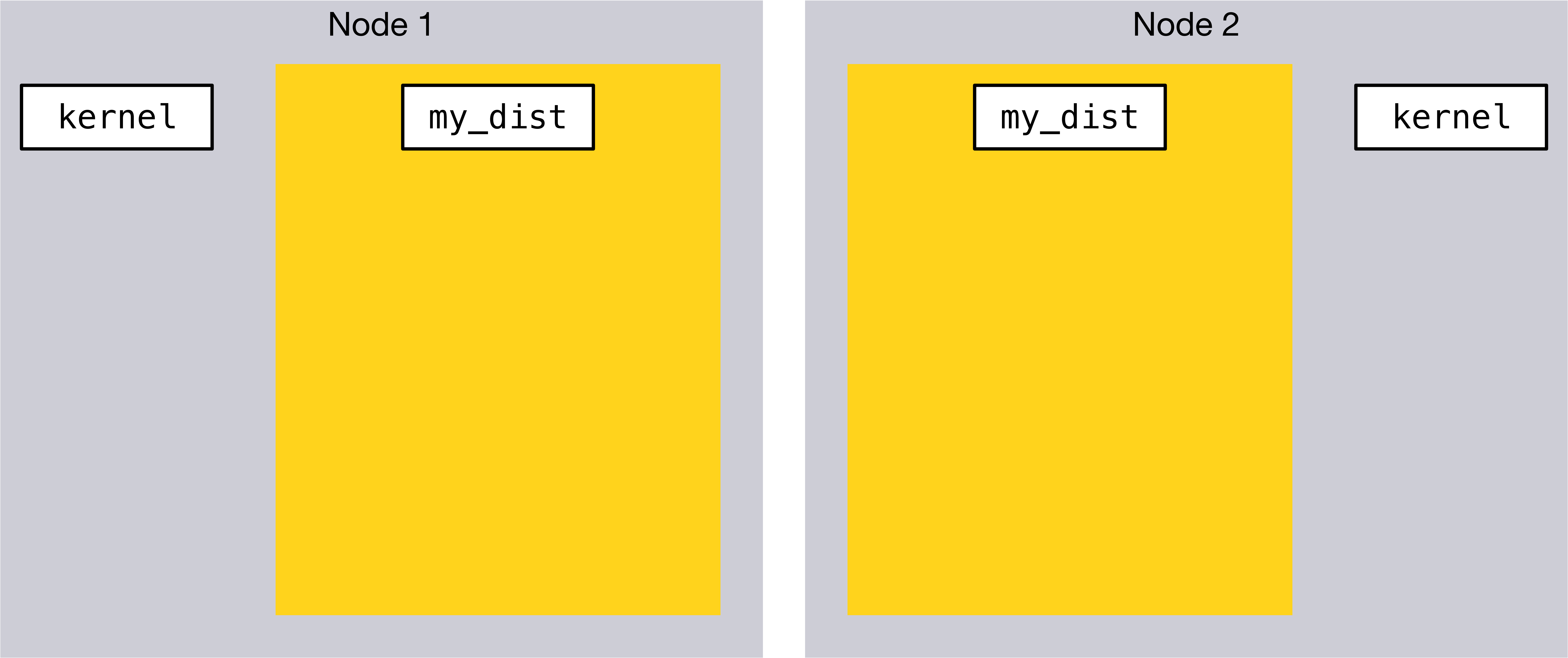
DIPL. PHYS.
PEER STRITZINGER GMBH

THE DISTRIBUTION “STACK”

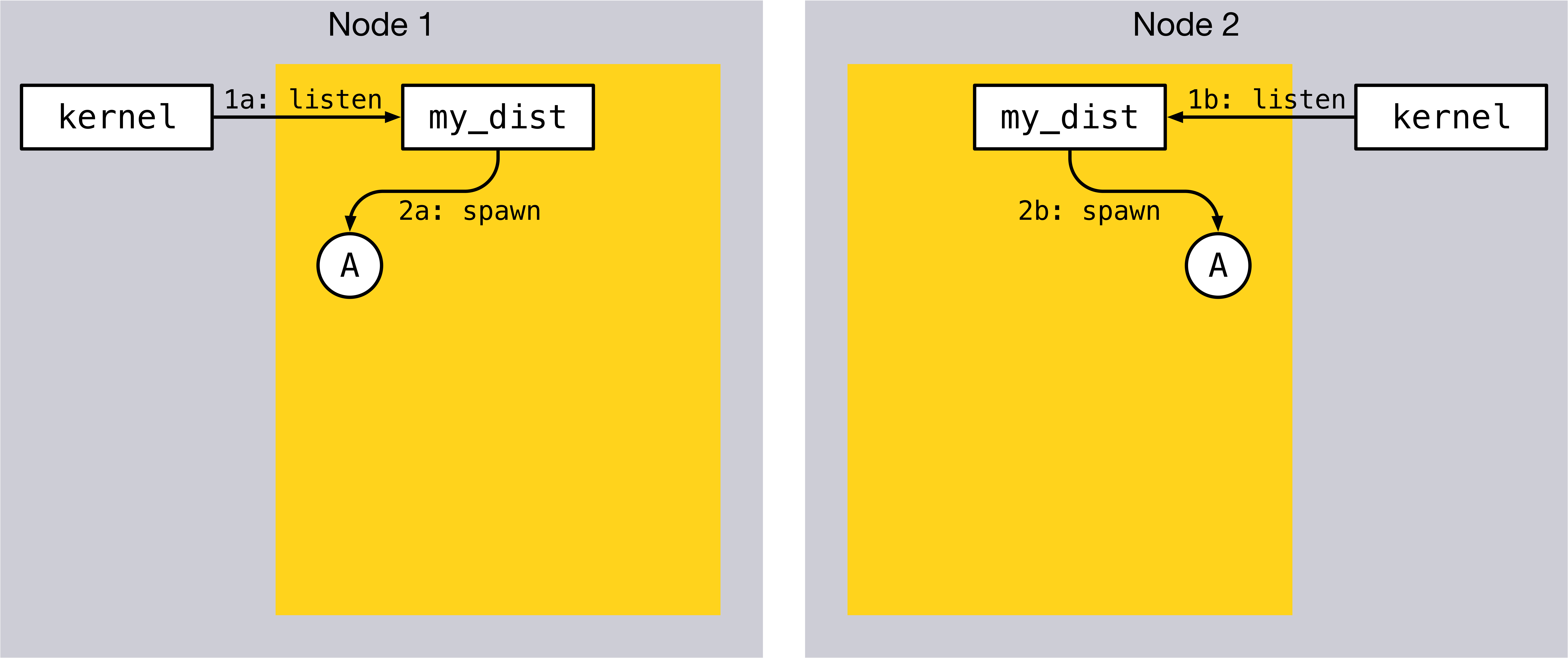


DIPL. PHYS. PEER STRITZINGER GMBH

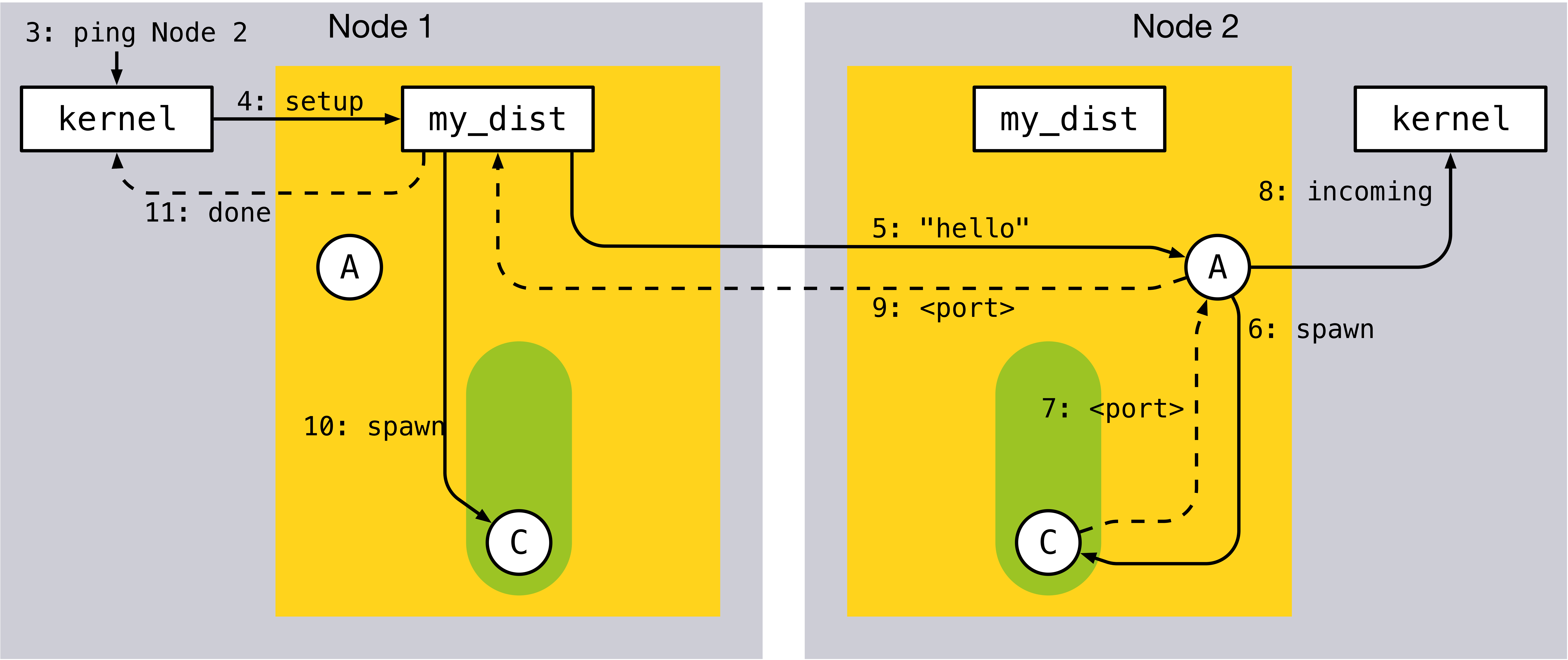
UDP DISTRIBUTION PROTOTYPE



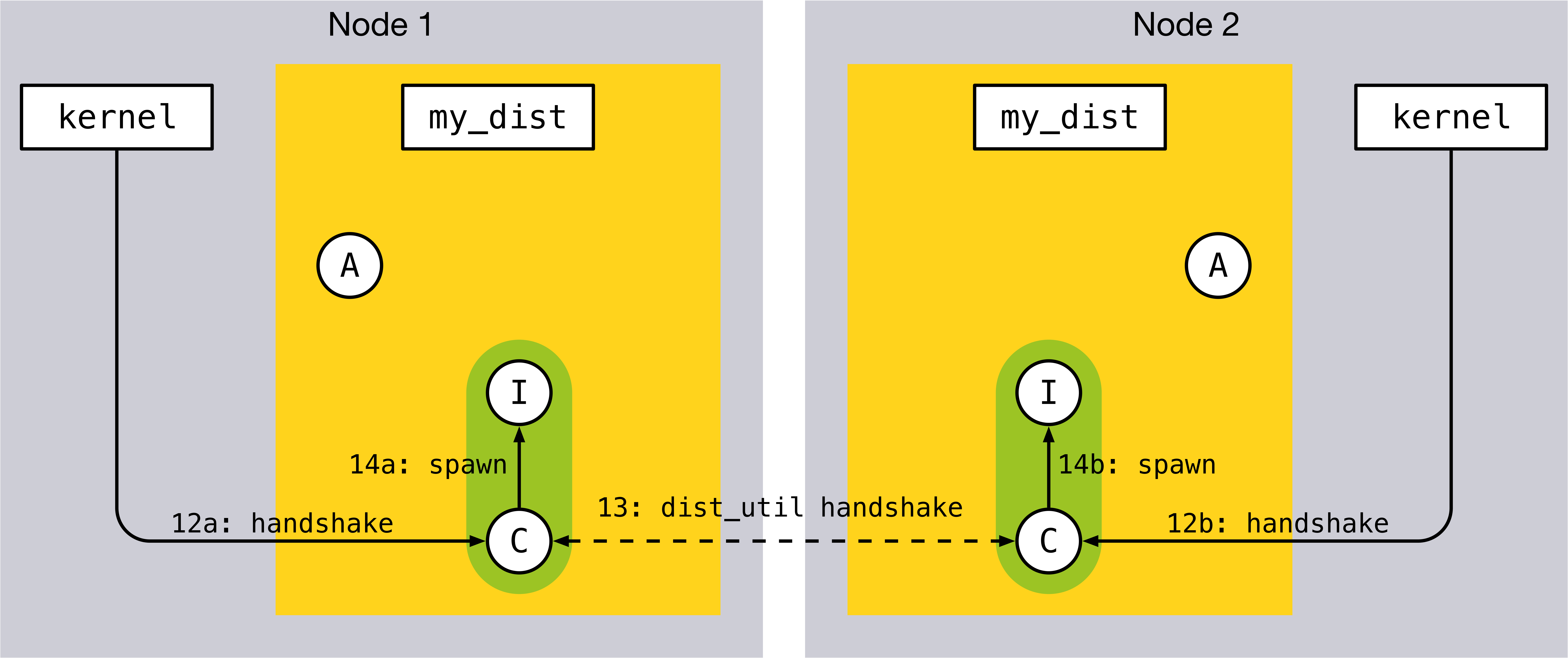
UDP DISTRIBUTION PROTOTYPE



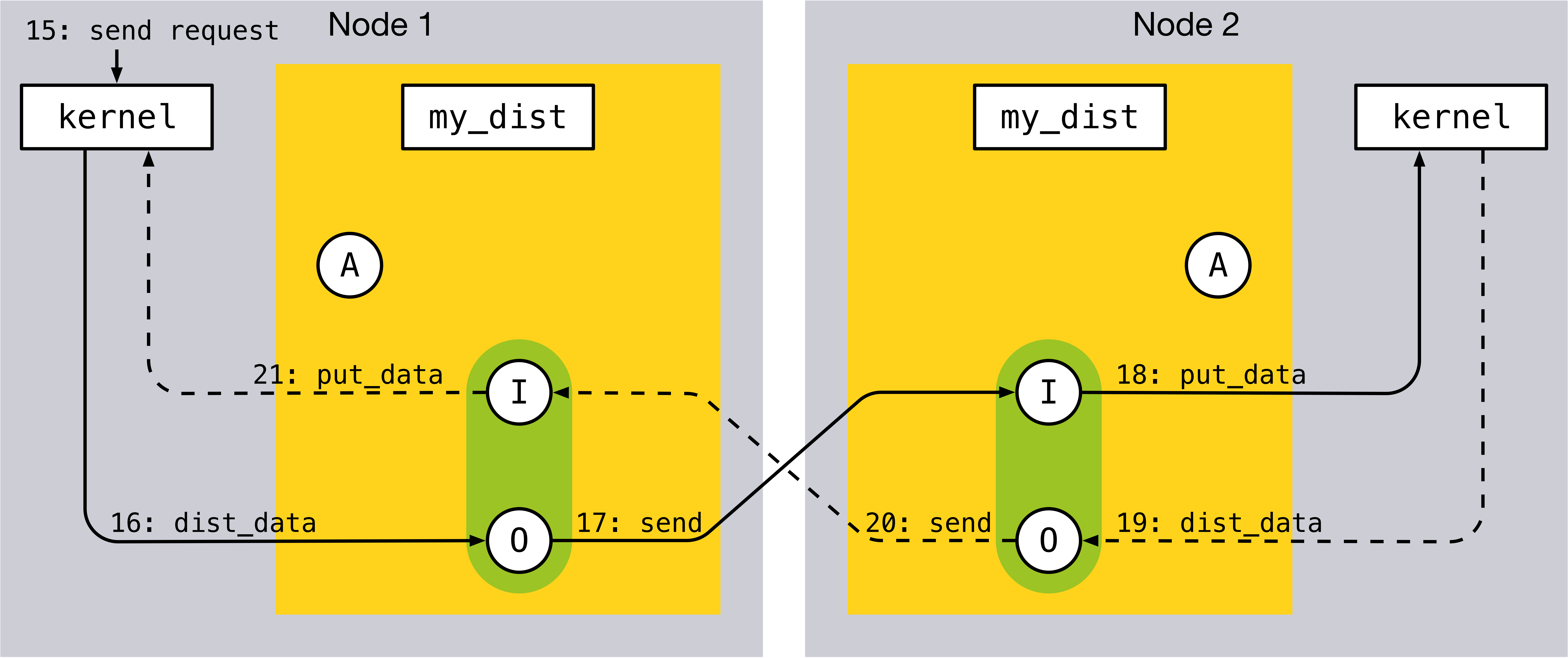
UDP DISTRIBUTION PROTOTYPE



DIPL. PHYS. PEER STRITZINGER GMBH


Node 1
Node 2
kernel
my_dist
A
I
C
14a: spawn
12a: handshake
13: dist_util handshake
my_dist
kernel
A
I
C
14b: spawn
12b: handshake

UDP DISTRIBUTION PROTOTYPE



DIPL. PHYS.
PEER STRITZINGER GMBH

UDP DISTRIBUTION PROTOTYPE SUMMARY

- One acceptor process per node
  - Opens a separate UDP listening port for connection attempts
- Two processes, one input and one output, per node connection
  - Could have been one process, but better throughput this way
- (not shown) Erlang Port Mapping Daemon (epmd) used to get the initial acceptor port to connect to

DEMO

DIPL. PHYS.
PEER STRITZINGER GmbH

NEXT STEPS

GENERIC DISTRIBUTION BEHAVIOR

A DISTRIBUTION BEHAVIOR

- The UDP prototype is quite similar to the TCP example from OTP
- What if we could make a behavior that encapsulates all the complexity of the current API and process model?
  - What would such a behavior look like?
  - What are the valid process models?
  - Can we combine this with a pluggable transport layer as well?

CURRENT STATE

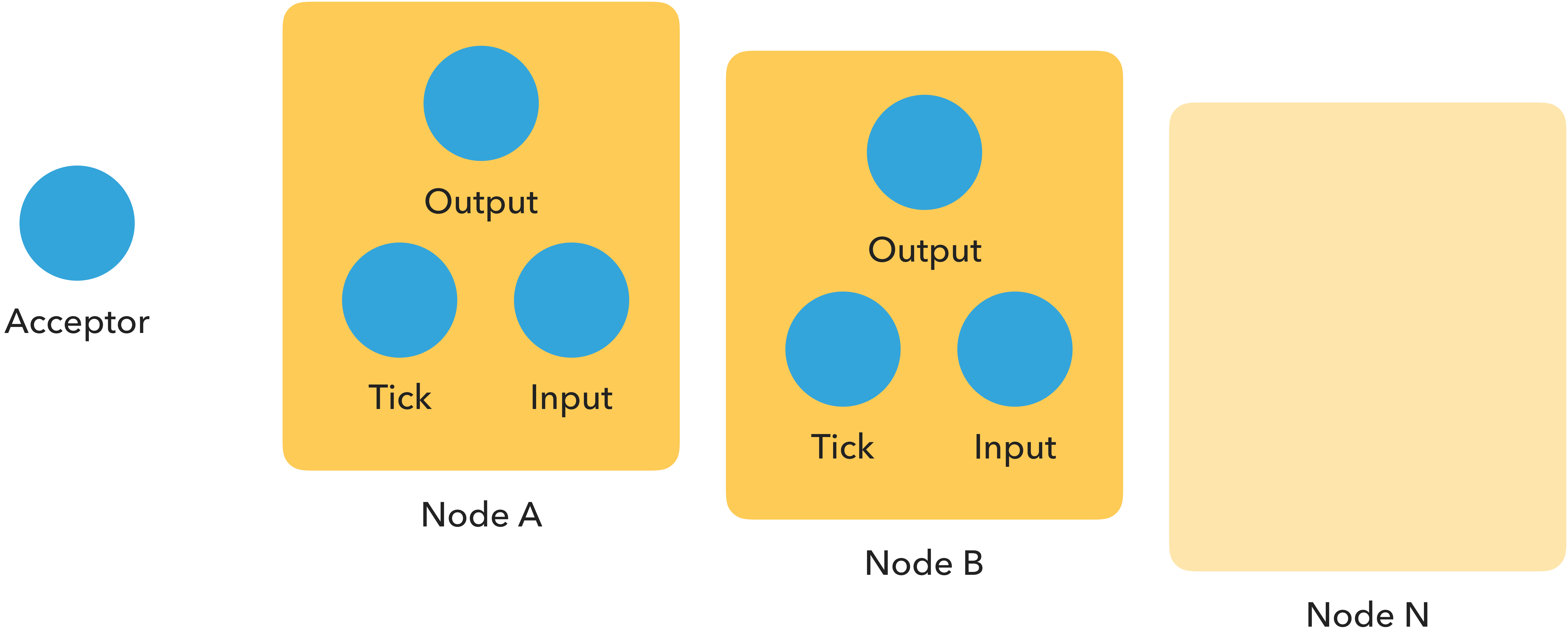
- Custom distribution API exposed in OTP 20
- API is based on the internal implementation for TCP

CHALLENGES USING THE LOW-LEVEL OTP API

- Distribution starts early, even before the IO server
  - No supervision, errors sometimes hidden or hard to expose
- The API surface is fragmented
  - callbacks
  - calls
  - message passing

PROCESS MODEL

- TCP example uses 1 process for listening and 3 processes per node connection



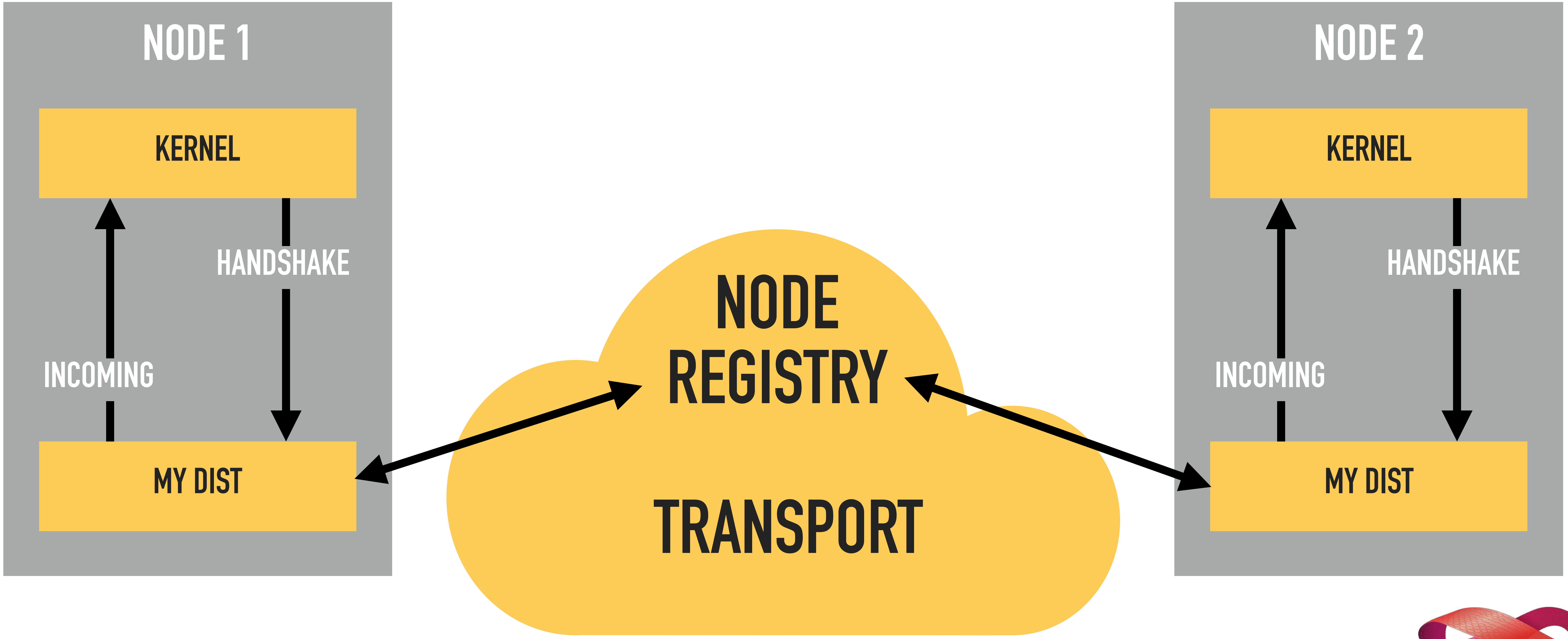
PROCESS MODEL

- Custom transport may require 0-N processes for listening and 0-N processes for connections
- Unsolved problem: generic callback behavior for running a dynamic number of processes with unique roles
- **Plan of attack:** Stick with the 1-3 model for now, experiment with other models later

API PROPOSAL

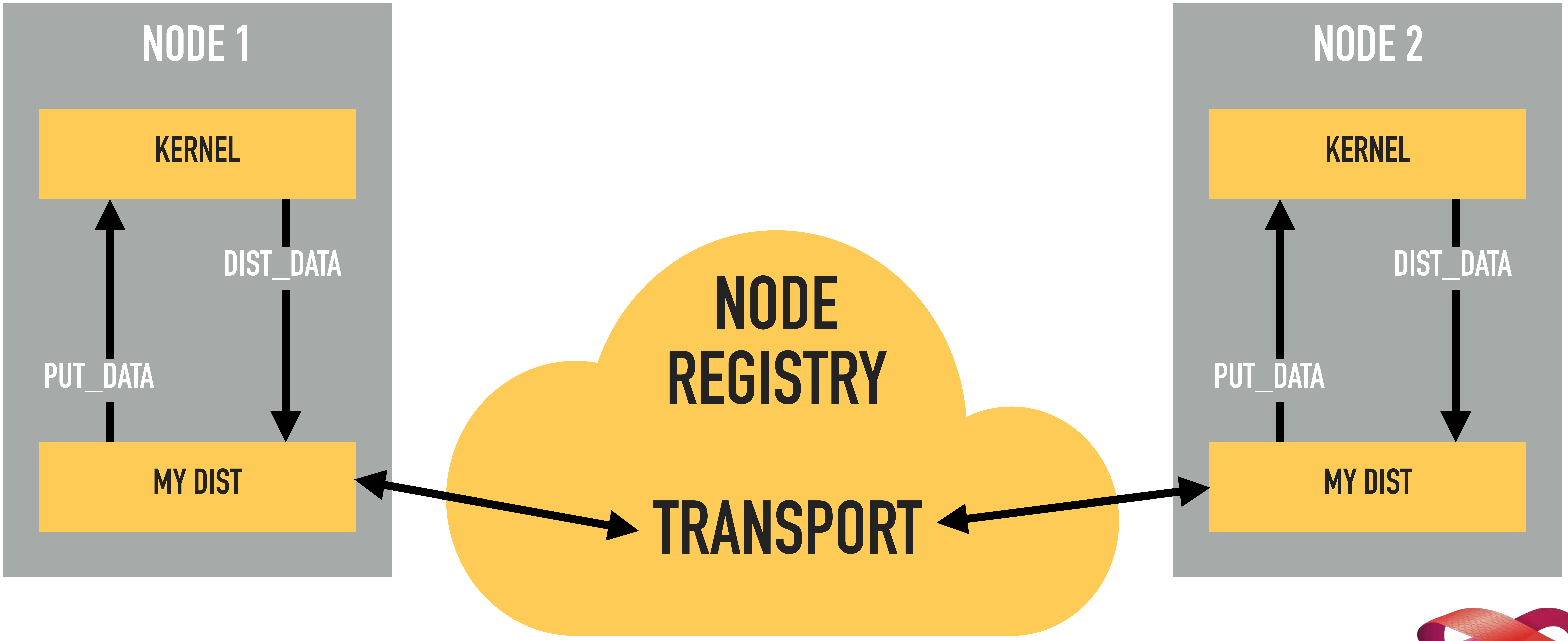
- Acceptor
  - acceptor_init/0
  - acceptor_info/2
  - acceptor_controller_spawned/3
  - acceptor_terminate/1
- Output
  - output_init/1
  - output_send/2
- Input
  - input_init/1
  - input_info/2
- Tick
  - tick_init/1
  - tick_trigger/2

HIGH LEVEL APPROACH

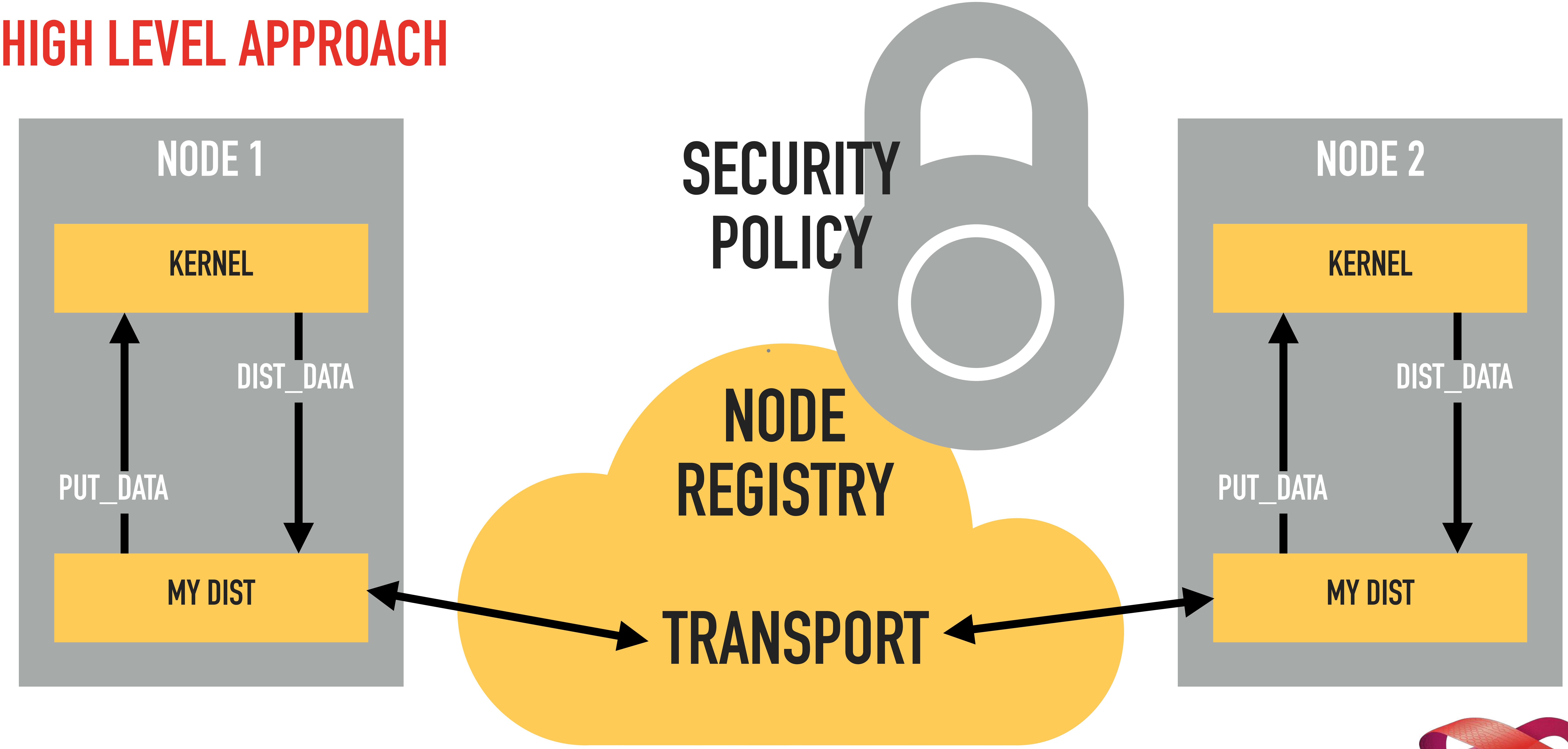


DIPL. PHYS.
PEER STRITZINGER GMBH

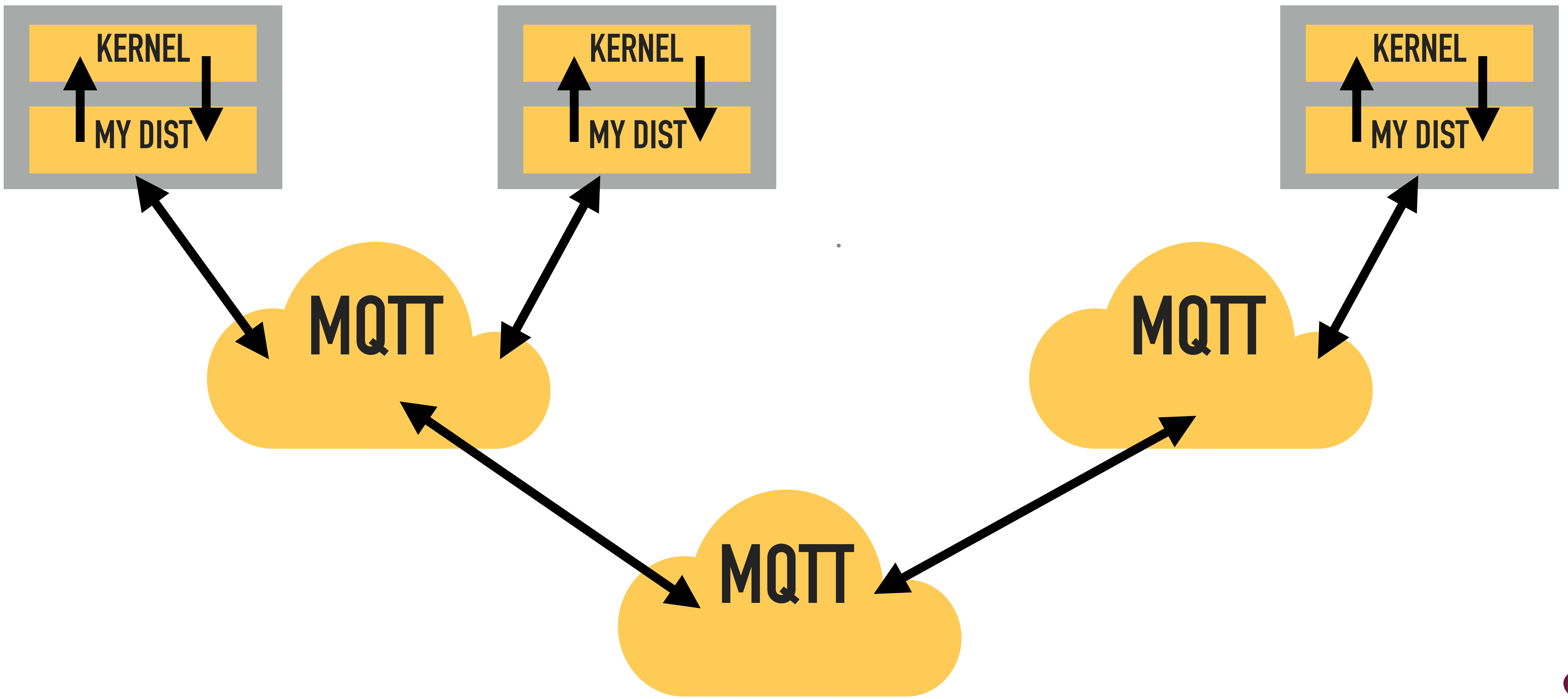
HIGH LEVEL APPROACH



HIGH LEVEL APPROACH



DISTRIBUTION OVER MQTT PUB SUB



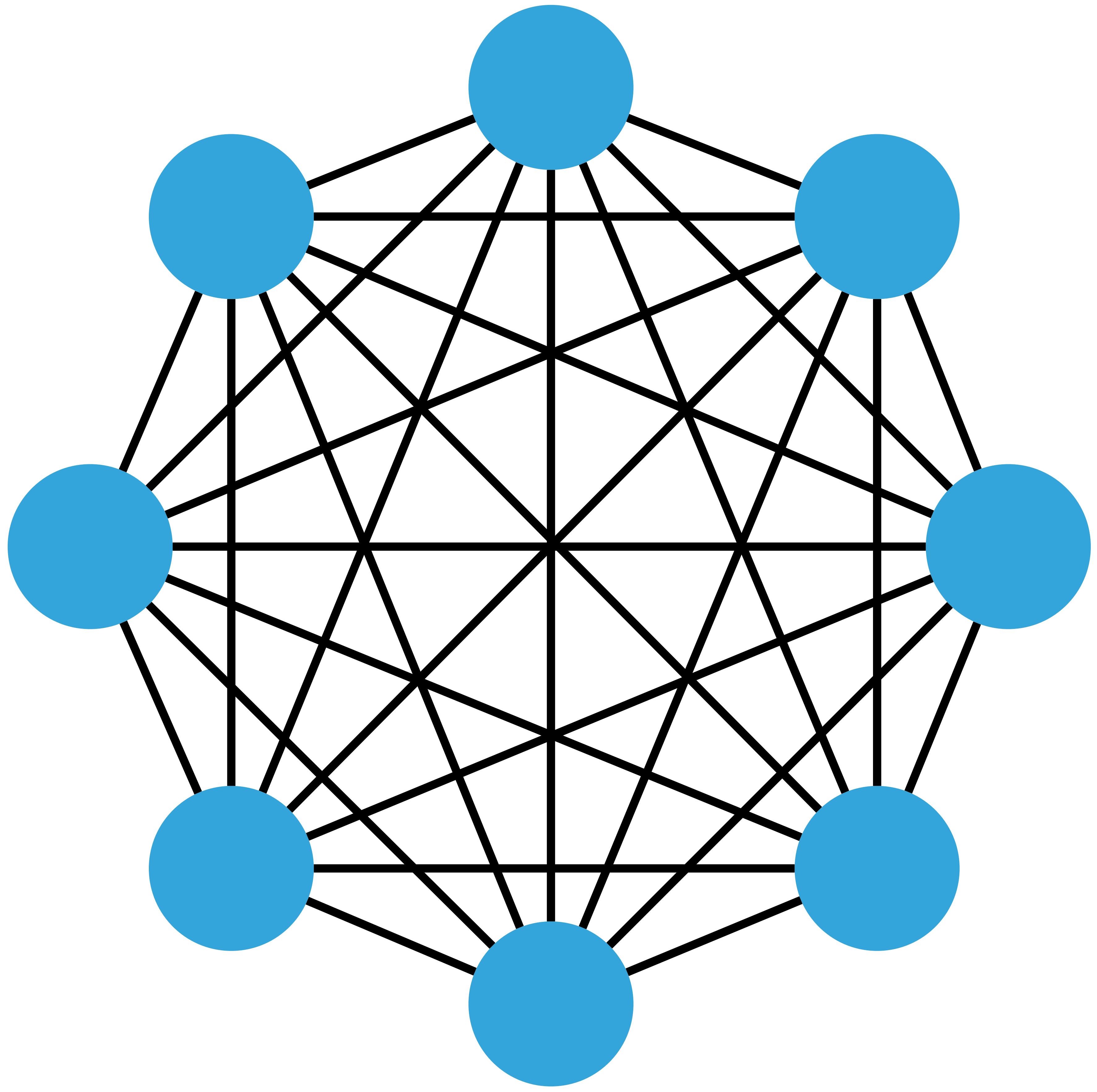
HIGH-LEVEL DISTRIBUTION

HETEROGENOUS NETWORKS

DIPL. PHYS.
PEER STRITZINGER GMBH

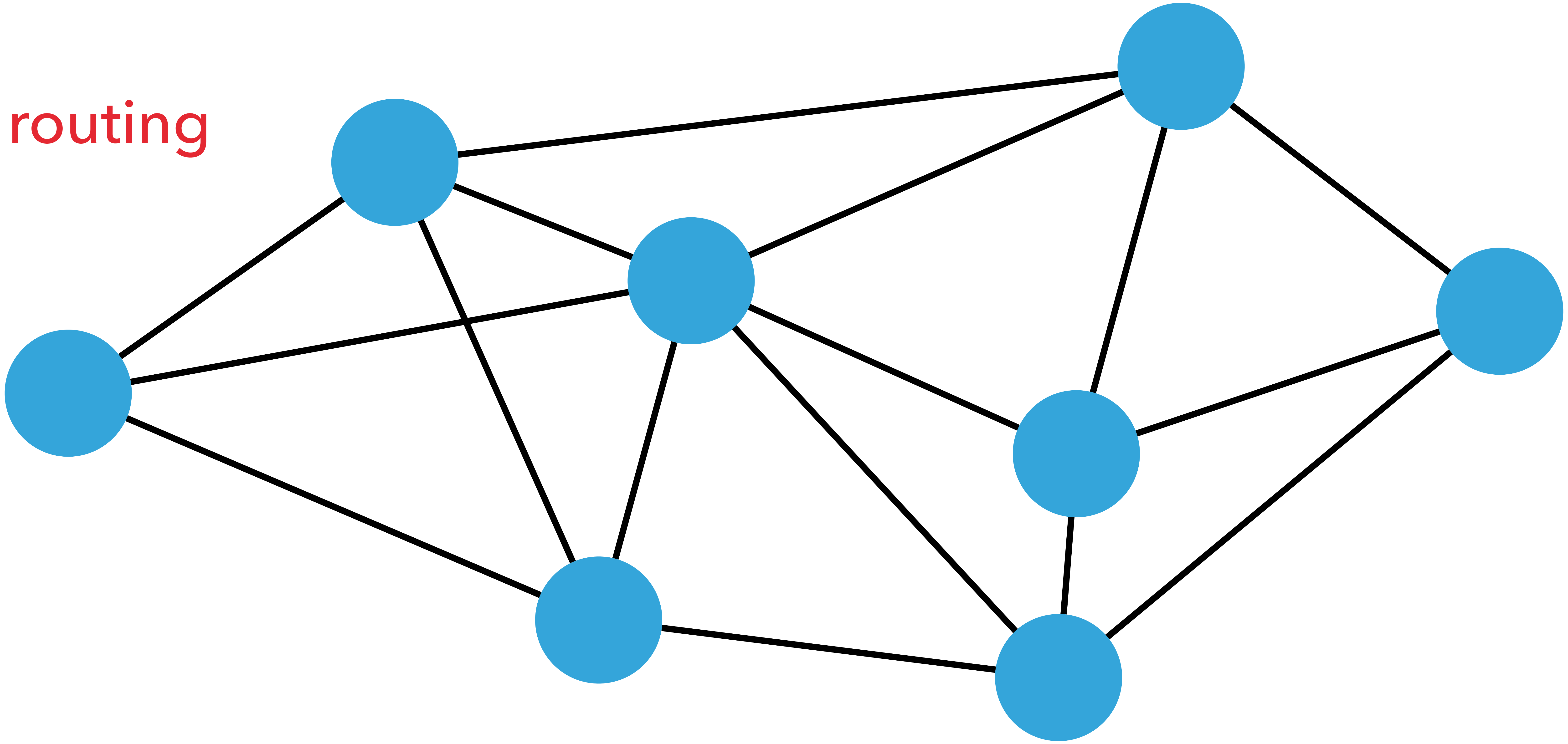
FULLY CONNECTED MESH

- Doesn't scale!



NORMAL MESH

- Scales!
- But needs routing



CUSTOM DISTRIBUTION TO THE RESCUE

- Topologies can be made transparent to the application layer
- Transports can also be made transparent
- Requirements
  - A way to represent “virtual” node connections
  - Can be made transparent to the application

TIME SENSITIVE NETWORKING

ETHERNET TSN

DIPL. PHYS.
PEER STRITZINGER GMBH

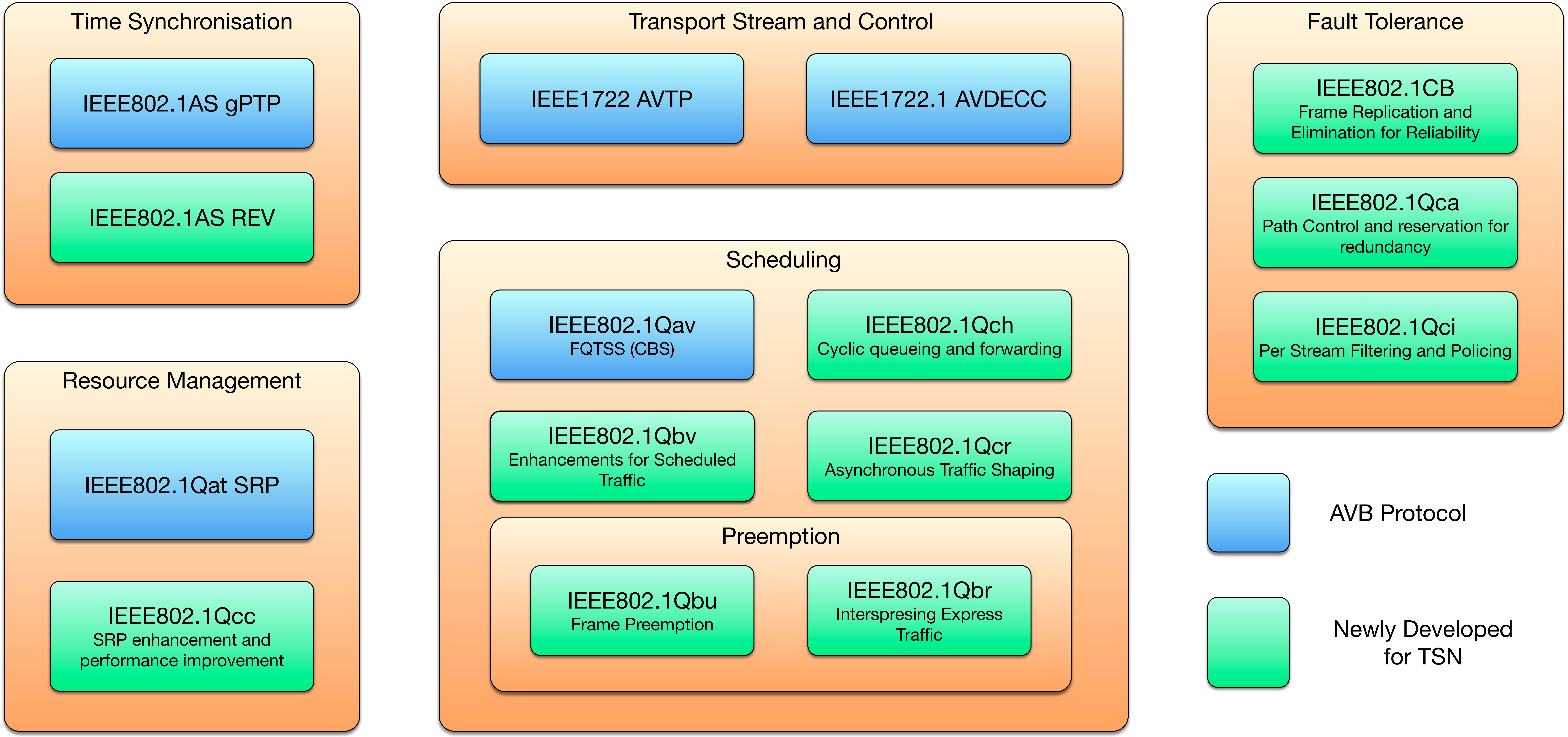
WHAT'S IT ABOUT

- Bounded transmission latency
- Low transmission latency
- Reliable delivery of Ethernet packets

+ HARD REALTIME ERLANG PROCESSES

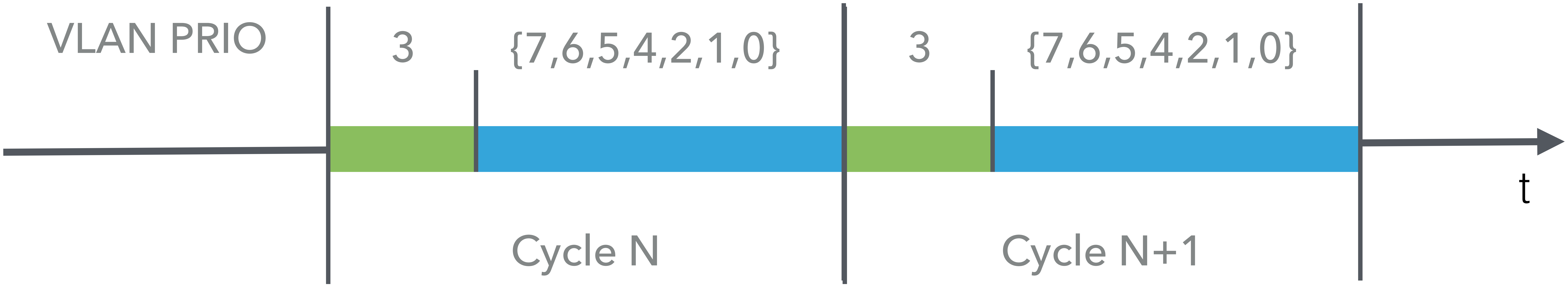
=

DISTRIBUTED HARD REALTIME APPLICATIONS



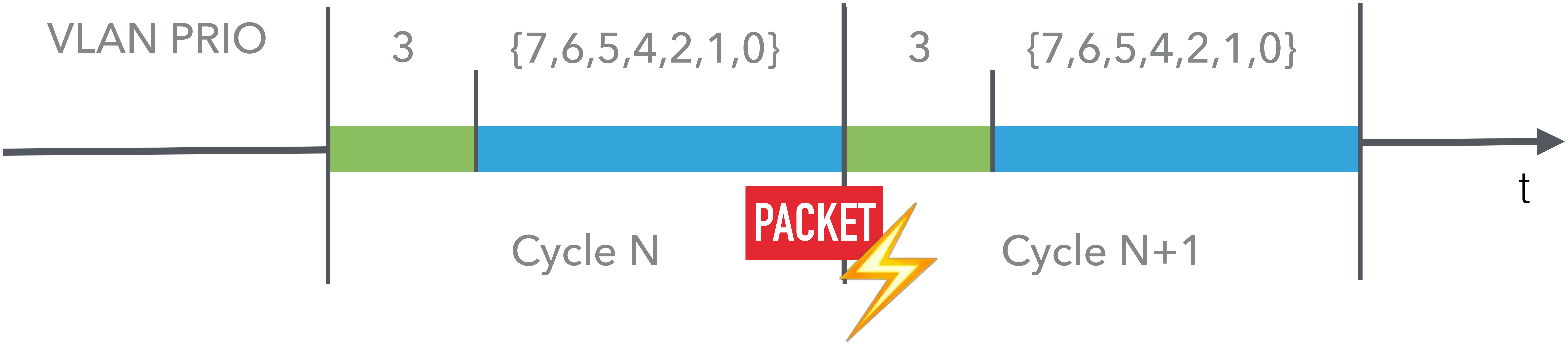
DIPL. PHYS.
PEER STRITZINGER GmbH

IEEE 802.1Qbv

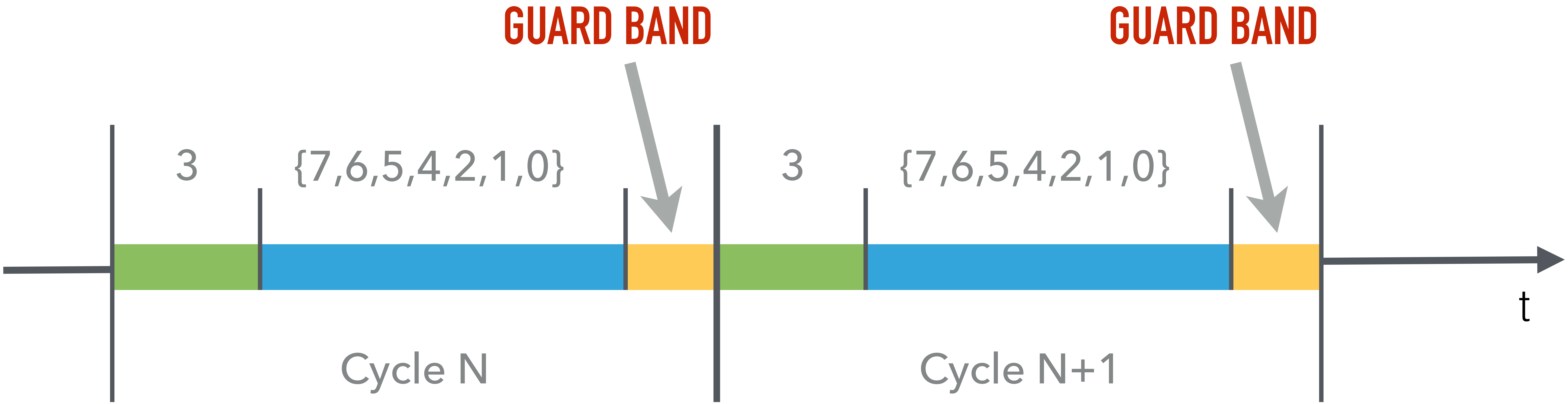


DIPL. PHYS. PEER STRITZINGER GMBH

IEEE 802.1Qbv



IEEE 802.1Qbv



RESULTS & FUTURE WORK

SUMMARY

DIPL. PHYS.
PEER STRITZINGER GMBH

RESULTS & CURRENT STATE

- Generic distribution behavior
  - WIP
  - Multiple API alternatives
- TSN
  - Implementing a TSN Switch with Shortest-Path-Bridging
  - Using Erlang for the control plane

FUTURE WORK

- Erlang distribution
  - Connection oriented UDP
    - QUIC
  - MQTT prototype
  - „Virtual node connections“
- Industrial networking
  - Prototype UDP over TSN
  - Implement real-time control prototypes

Kickstarer funded!

GRiSP 2

pre-order at

www.grisp.org

- Bare-metal Erlang
- Elixir & Nerves
- SoM module
- 696 Mhz - Faster CPU
- 128 Mb RAM - Twice the memory
- Wi-Fi *and* Ethernet

THANK YOU!

QUESTIONS?

www.stritzinger.com

@peerstr www.grisp.org @grisporg