EMQX Webinar

MQTT Broker Cluster Scalability How is it done in EMQX?

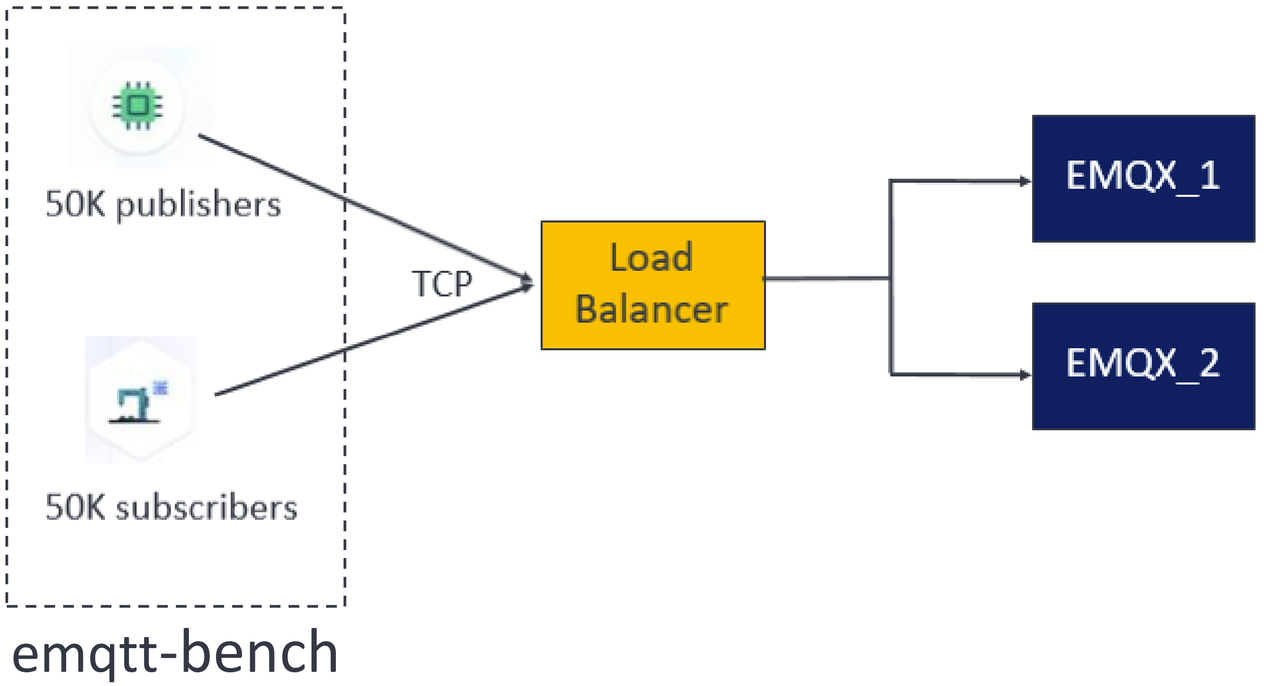
April 13th
11:00am EST / 6:00pm CET / 4:00pm UTC

Speaker:
Kary Ware, Sales Engineer @EMQ

Agenda

- The foundation of EMQX
- EMQX clustering currently in EMQX v4.x
- Improved clustering in EMQX v5.0
- Demo: Load testing using emqtt-bench
  - 100K connections, 100K messages/sec
  - Connect the clients during the presentation
- Summary and Q & A

Demo Preparation

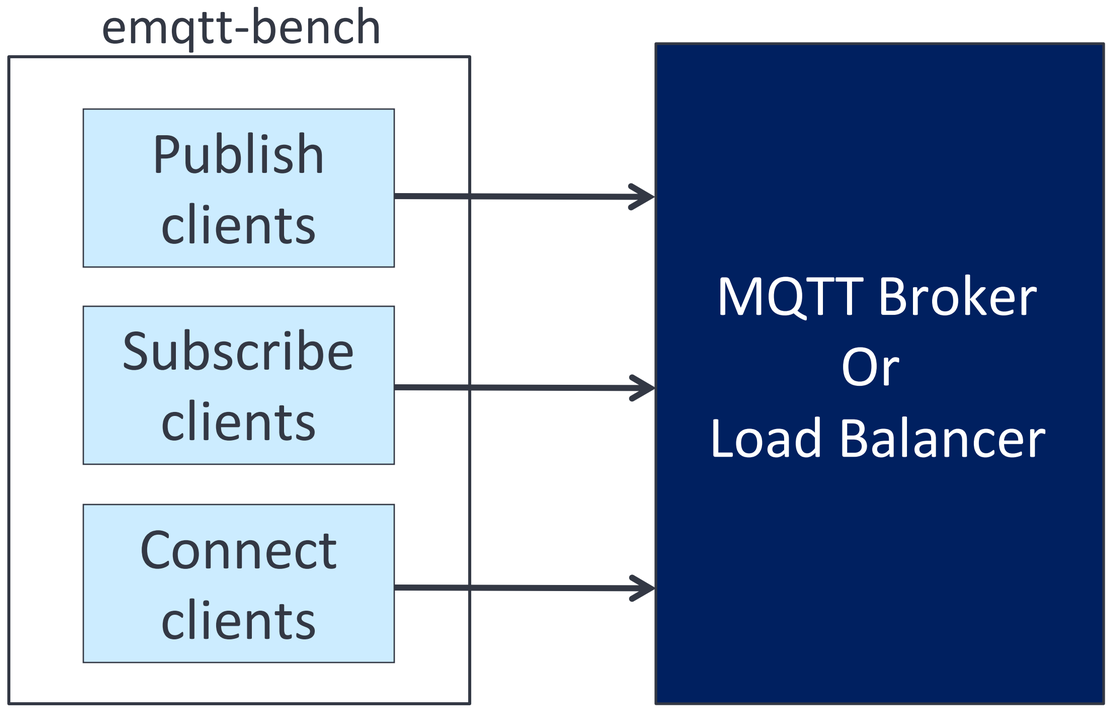


Start the 50 K subscribers

In emqtt-bench

emqtt-bench

`emqtt_bench` is an MQTT v5.0 benchmark tool written in Erlang.
For load testing: Can generate millions of clients
Publish, Subscribe, or just connect



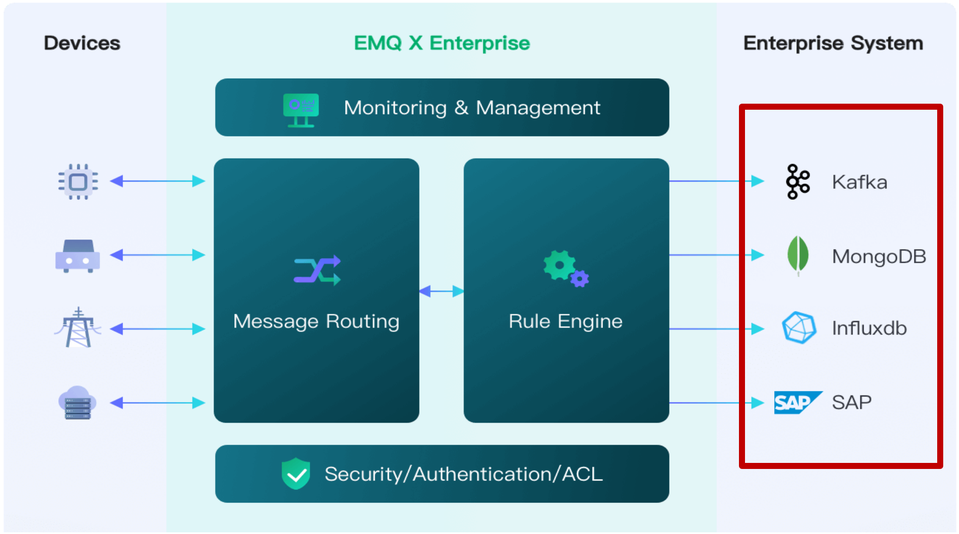
https://github.com/emqx/emqtt-bench

Subscribe: ./emqtt_bench **sub -c 50000** -h hostadd **-t test/%i** --qos 1 --ifaddr 192.168.0.10, 192.168.0.20

Publish: ./emqtt_bench **pub -c 50000** -h hostaddr **-t test/%i** --qos 1 --interval_of_msg 1000 --ifaddr 192.168.1.10, 192.168.0.20

The foundation of EMQX

EMQ X CLOUD



EMQX, Erlang, OTP, BEAM VM

EMQX is built using Erlang, with OTP libraries, that runs on the BEAM VM

Erlang was created by Ericsson in 1986 to develop their telecom products more efficiently

Not many languages available, so needed to create their own

Ericsson's software requirements:

- Work in large real time systems - Develop efficiently
- Concurrency – Be able to handle 100K's of calls
- Distributed – Over several computers
- Fault tolerant – Failure of a single switch would not bring down the entire system
- High reliability and high availability – Continuous operation over many years
- Hot upgrade – Upgrade software without stopping the system

Summary: A concurrent, fault-tolerant, scalable system

Erlang: The Movie

1992 demo by the creators of Erlang

Make a point-to-point phone call

Leave the call connected

Attempt a 3-way conference call

BUT, it doesn't work.

Because of a bug in the program....

Erlang: The Movie (cont)

With the point-to-point call still connected.

They correct the bug.

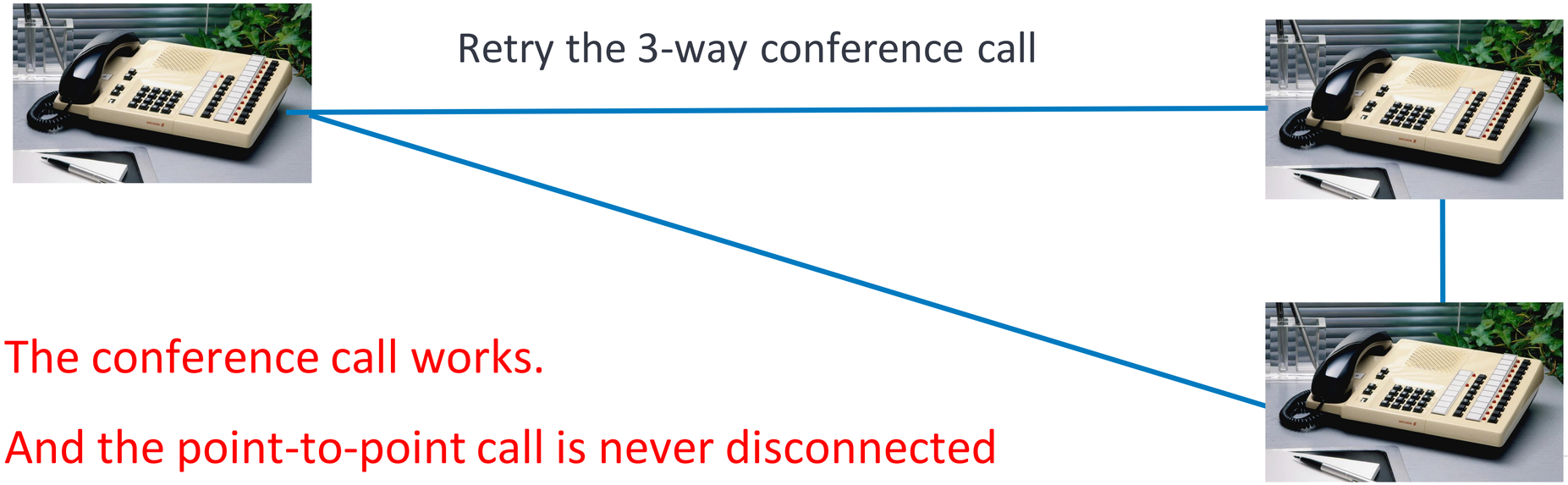
And restart the system.

```
case multi_no {
    Other_no =>
        Parallel_CH
        call:conversa
```

```
> ["lots/feature"].
[consult lots/feature]
-- redefining module fea
{GC}[lots/feature consul

> run.
```

Erlang: The Movie (cont)

With the point-to-point call still connected.



The conference call works.

And the point-to-point call is never disconnected

Erlang: The Movie (Conclusion)

point-to-point phone call

This could have been 1 million calls and none would have been disconnected.
Because of the Erlang feature of process isolation.

The bug was a reproducible error in a common feature.

So, this would have been caught when testing.

```
case ulti_no {
    Other_no =>
        Parallel_CH
        call:conversa
```

Erlang's strength is that it can **detect** and **recover** from **transient,** hard to reproduce errors.

Characteristics of Erlang

- Many lightweight concurrent processes: Can be millions
- Isolated processes: No shared memory or pointers
- Asynchronous communication among the processes
- Fault tolerant: A part of the system may crash, but not the entire system
- Non-blocking: One process cannot lock up the entire system
- "Soft real-time": A missed hardware deadline is not a failure
- Hot code updates: Update the code without stopping the system

Some components of Erlang

BEAM VM

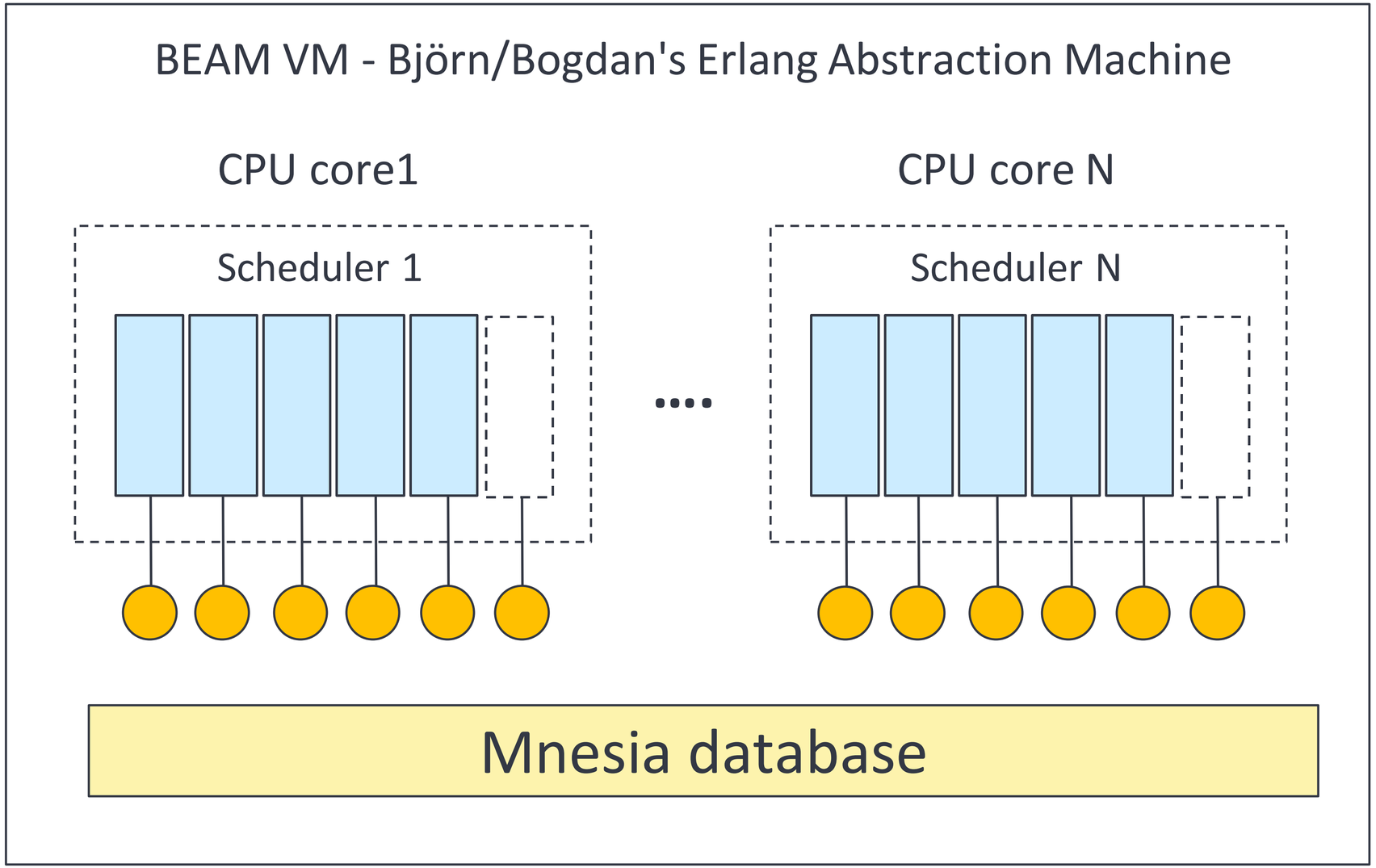
Multiple schedulers

(one per CPU core)

Many concurrent

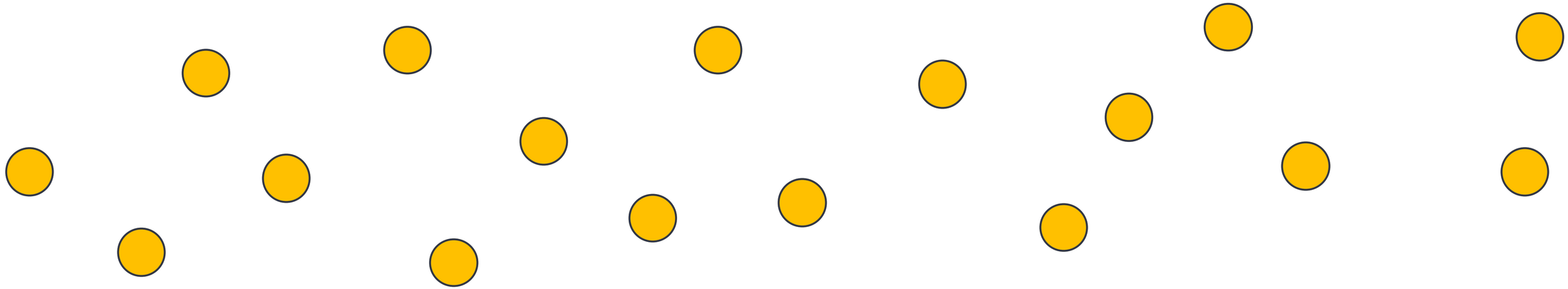
processes

Mnesia database



Lightweight concurrent processes

Can be millions of concurrent processes



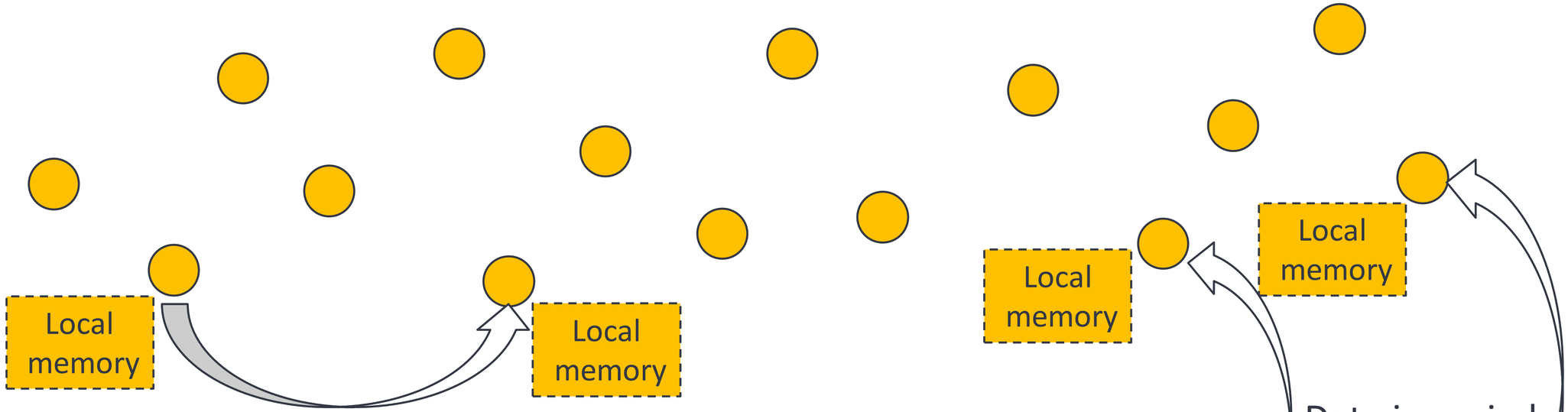
Each process

- Lightweight – minimum size is 326 words
- Isolated from each other
- Own local memory space – not shared

EMQX

- Approximately one process per client
- 100 M clients = 100 M processes

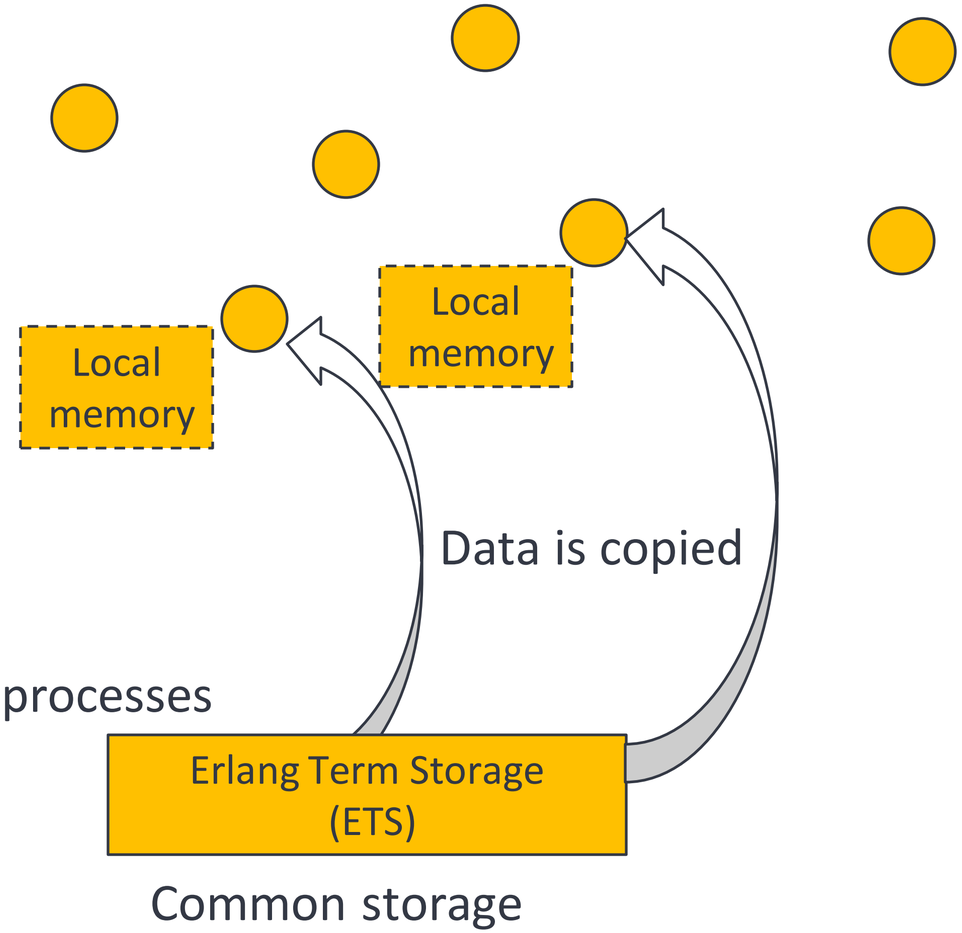
Asynchronous message passing



Message passing is non-blocking

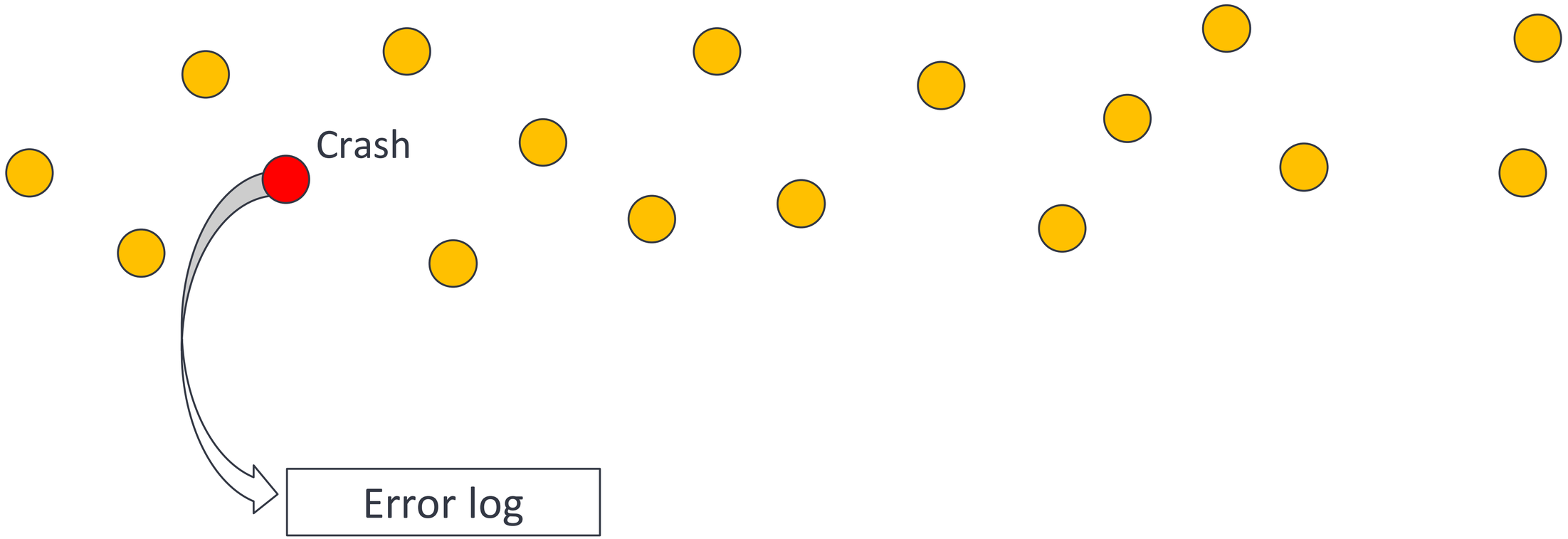
Message data is copied – no shared memory between processes

Each process has own local memory



Important for isolation...

Isolation

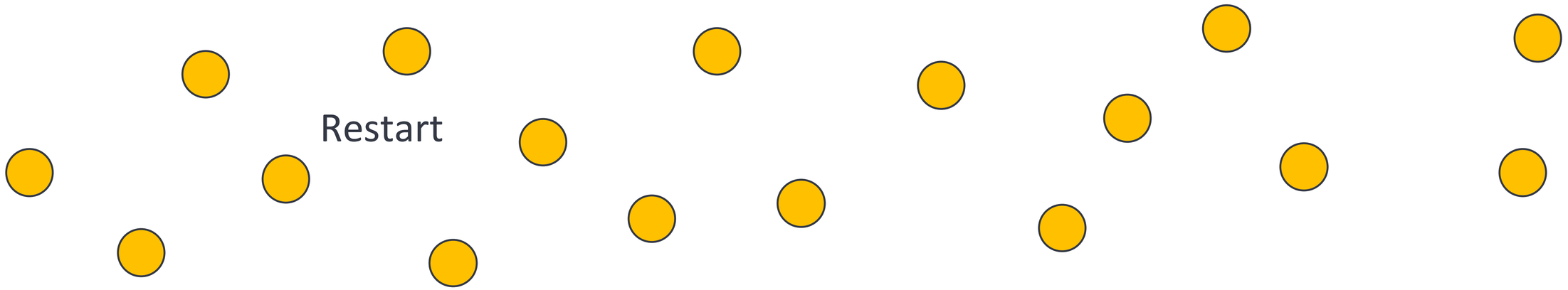


A single crashed process does not affect the other processes

The rest of the system is still running

Crash is recorded in an error log for later analysis

Fault tolerance



System still running even though one processed crashed

Reason for crash

Transient error: A restart may cause the process to continue working

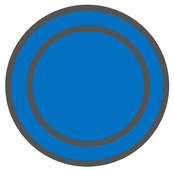
Reproducible error: Error log and testing will help to debug the issue

Supervisor tree

Processes organized in a tree – Supervisor tree
Processes can be supervisors or workers

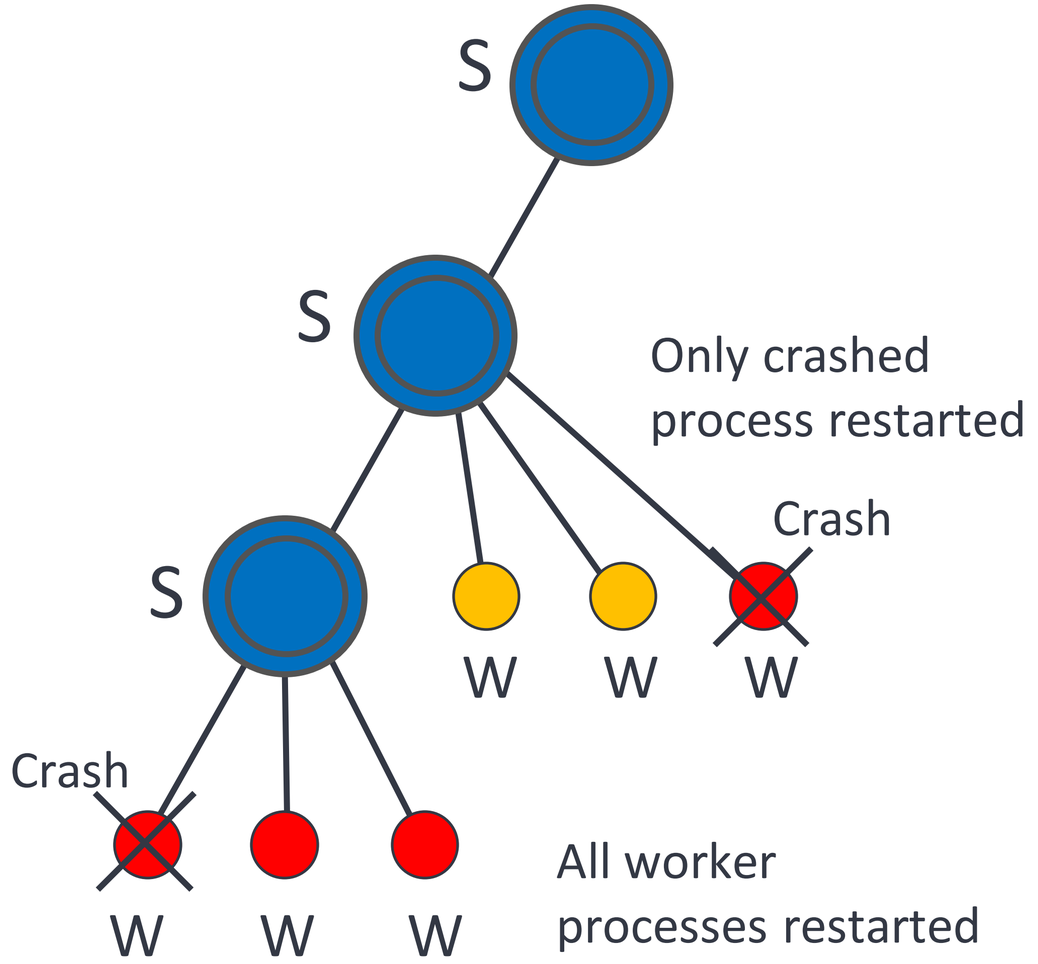
Workers:
Do the work

Supervisors:
Monitor a set of workers
Are notified when one of its workers crashes
Can restart only the crashed worker
Or restart all of its workers



Supervisor

Worker



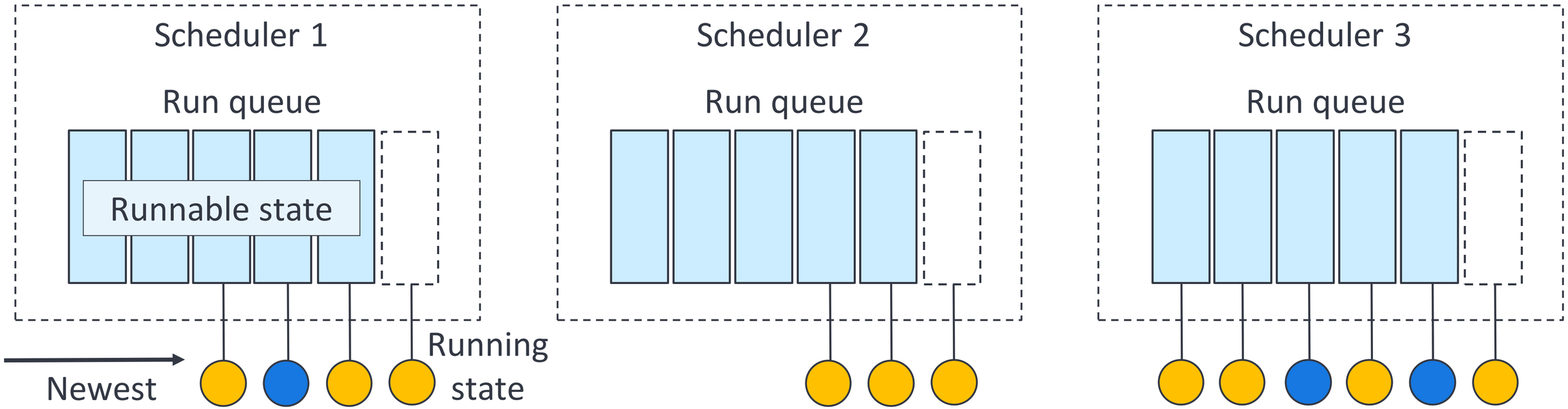
Schedulers

Processes run in a scheduler – both supervisor and worker processes

Each scheduler has its own FIFO run queue

Processes can have a priority – max, high, normal, low

One scheduler per CPU core



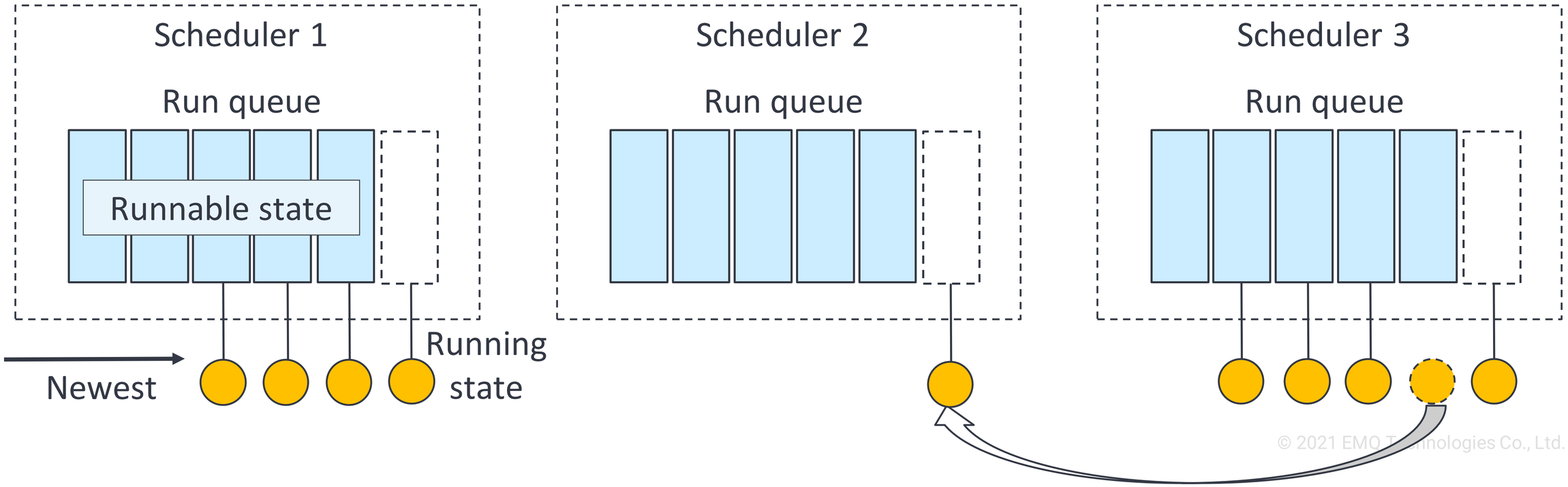
Load balancing

If a scheduler has no processes to run...

then it will try to steal a process from another scheduler.

Goal is to not have idle schedulers.

The system will try to minimize the number of schedulers and put the empty ones to sleep.



Anti-blocking mechanism

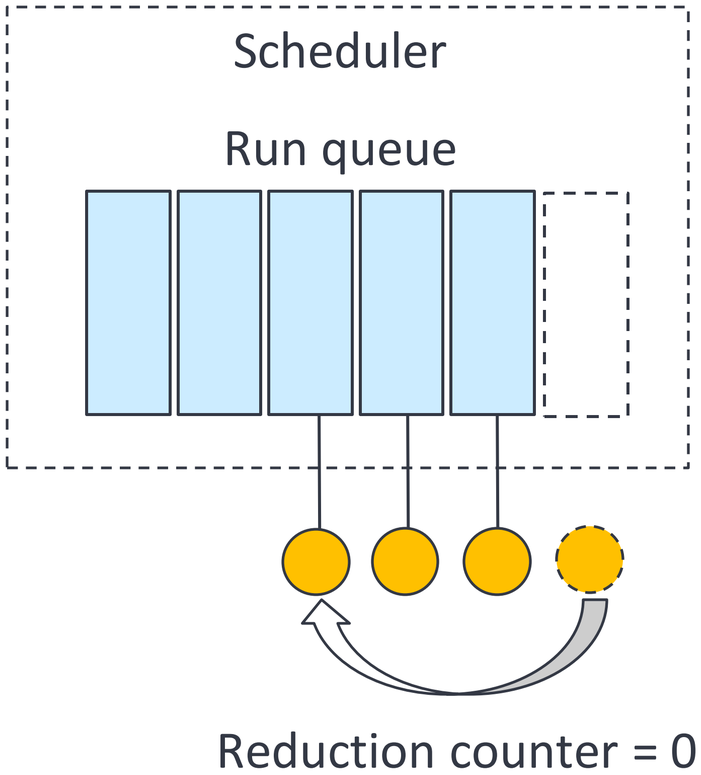
Preemptive scheduling

Prevents processes from using 100% of the run time.

A process is moved to the end of the queue after performing a certain amount of work.

Called Reductions:

- Reduction counter initialized to 2000
- Decremented for work done: Function calls, garbage collection, etc
- Process rescheduled when reduction counter = 0



Hibernation

Process is moved to the **Waiting** state...

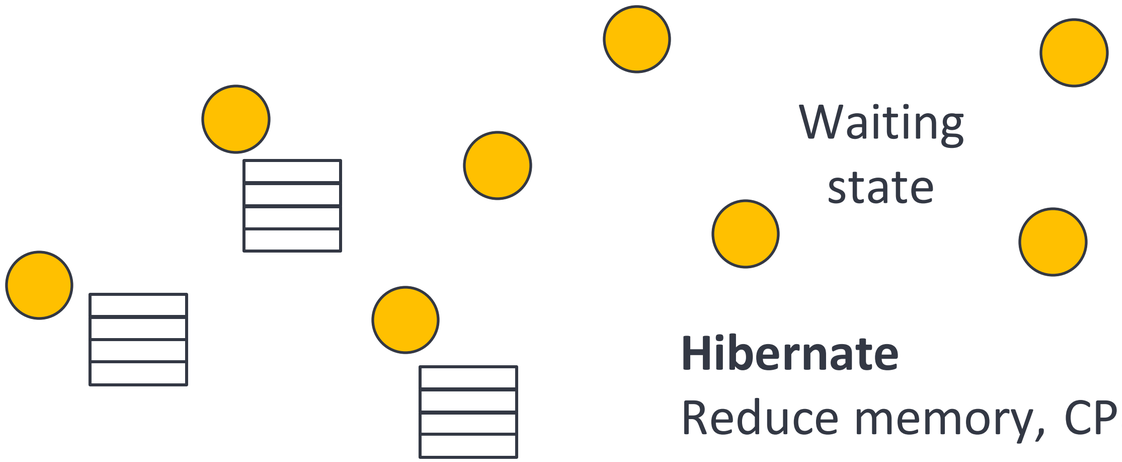
if waiting for messages that are not in its message queue.

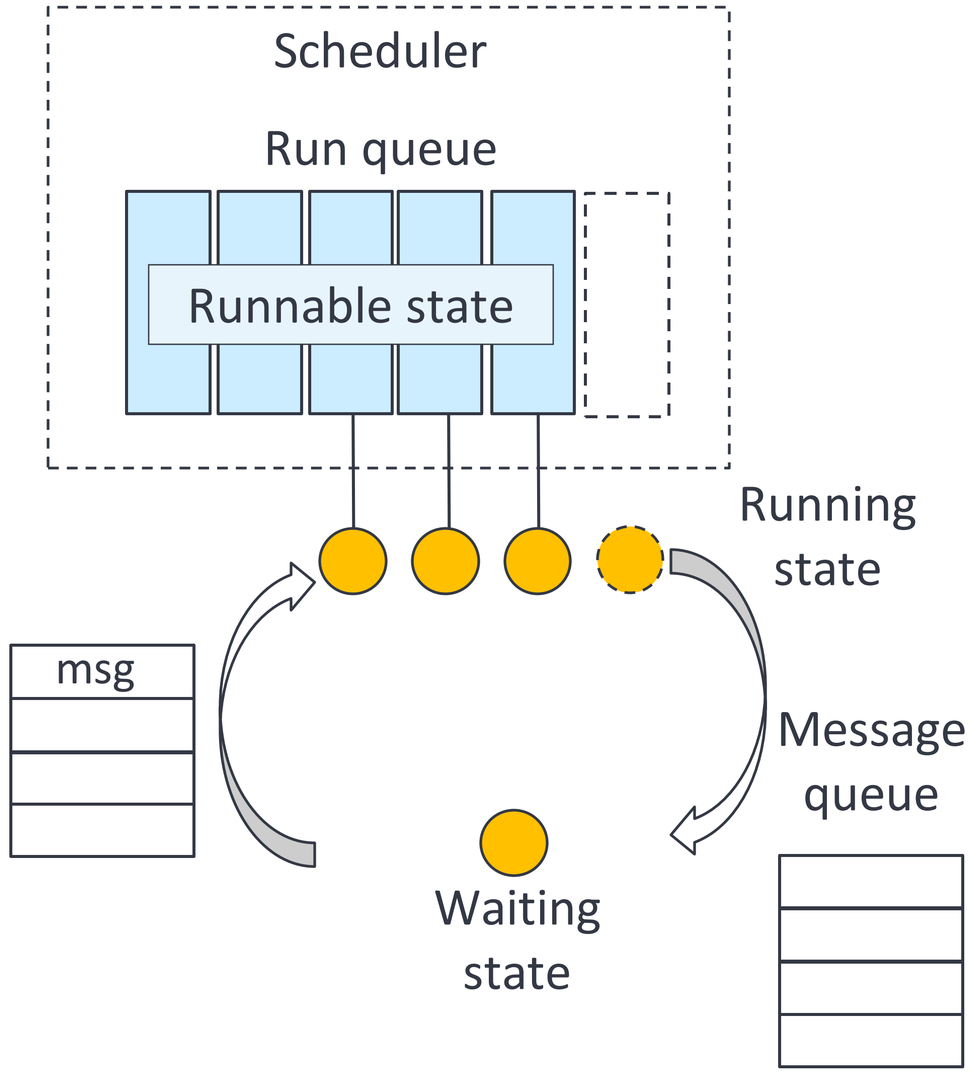
(each process has own message queue)

Process moved back on queue to runnable state...

when message arrives.

Can be millions of processes, but not all active





Hibernate

Reduce memory, CPU load as much as possible

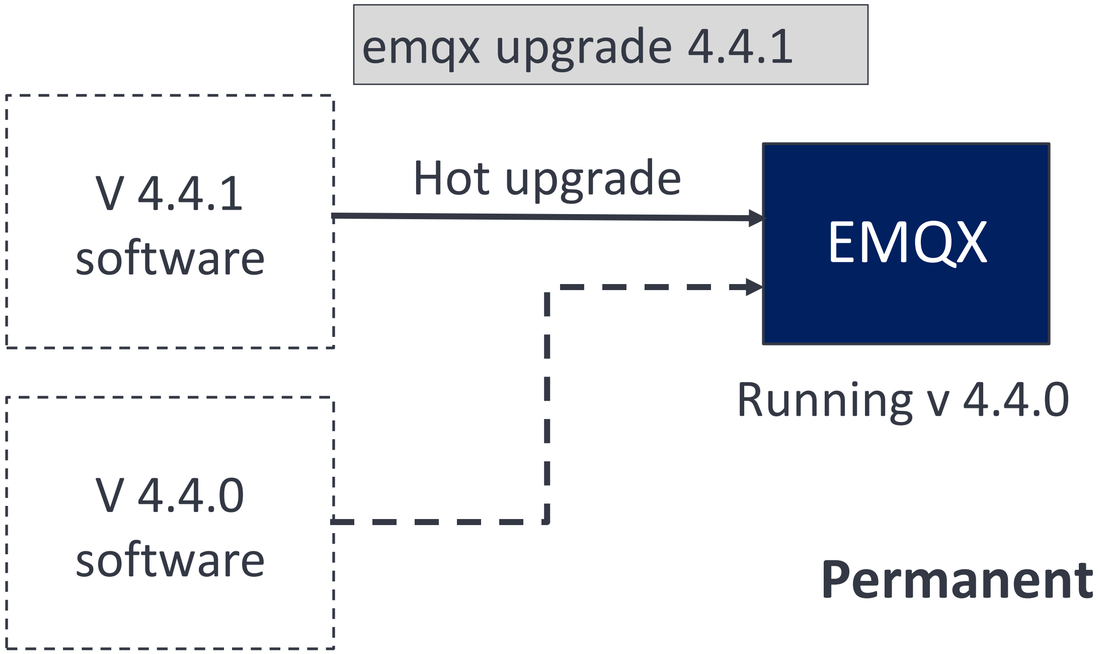
EMQX hot upgrade

Update the EMQX release patch version without shutting down the system

Patch version is third digit of version number: 4.4.**X**

Hot upgrades from 4.4.0 -> 4.4.1, 4.4.0 -> 4.4.2, ..., etc.

But 4.2.x cannot be hot upgraded to 4.3.0 or 5.0.



After hot upgrade:

```
emqx versions
```

Result:

```
Installed versions:
  4.4.1    permanent
  4.4.0    old
```

Permanent: This version will be loaded if EMQX restarted

Can choose to have the old version as permanent

Hot upgrade steps

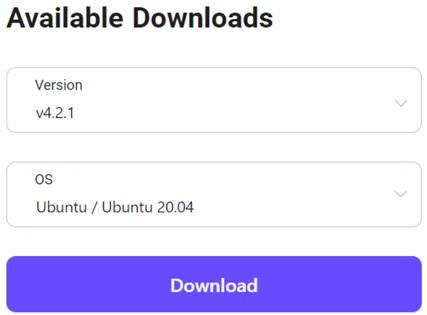
Example: Upgrade from v 4.2.0 to 4.2.1

Verify the current running version

```
$ emqx versions

Installed versions:
* 4.2.0 permanent
```

Download the new patch release and copy to releases directory

```
$ cp emqx-4.2.1.zip ${EMQX_ROOT_DIR}/releases/
```



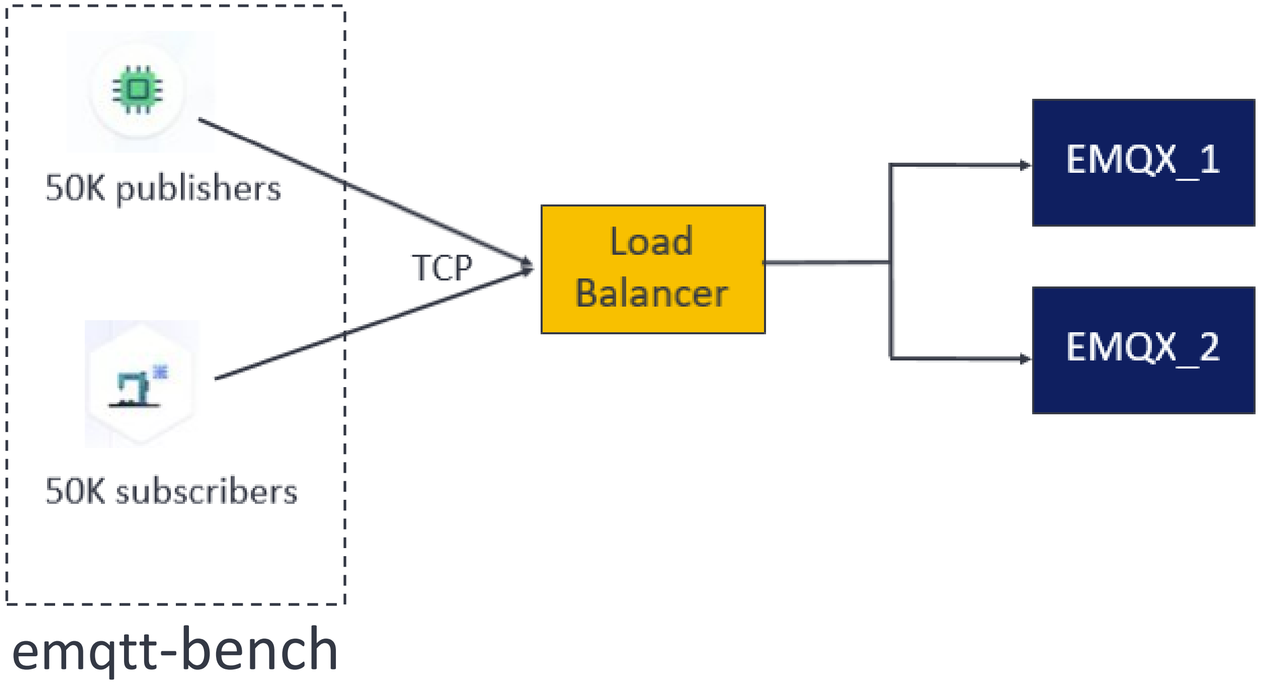
Upgrade to the new patch release

```
$ emqx upgrade 4.2.1

Release 4.2.1 not found, attempting to unpack releases/emqx-4.2.1.tar.gz
Unpacked successfully: "4.2.1"
Installed Release: 4.2.1
Made release permanent: "4.2.1"
```

Verify that the new release is running

```
$ emqx versions

Installed versions:
* 4.2.1 permanent
* 4.2.0 old
```

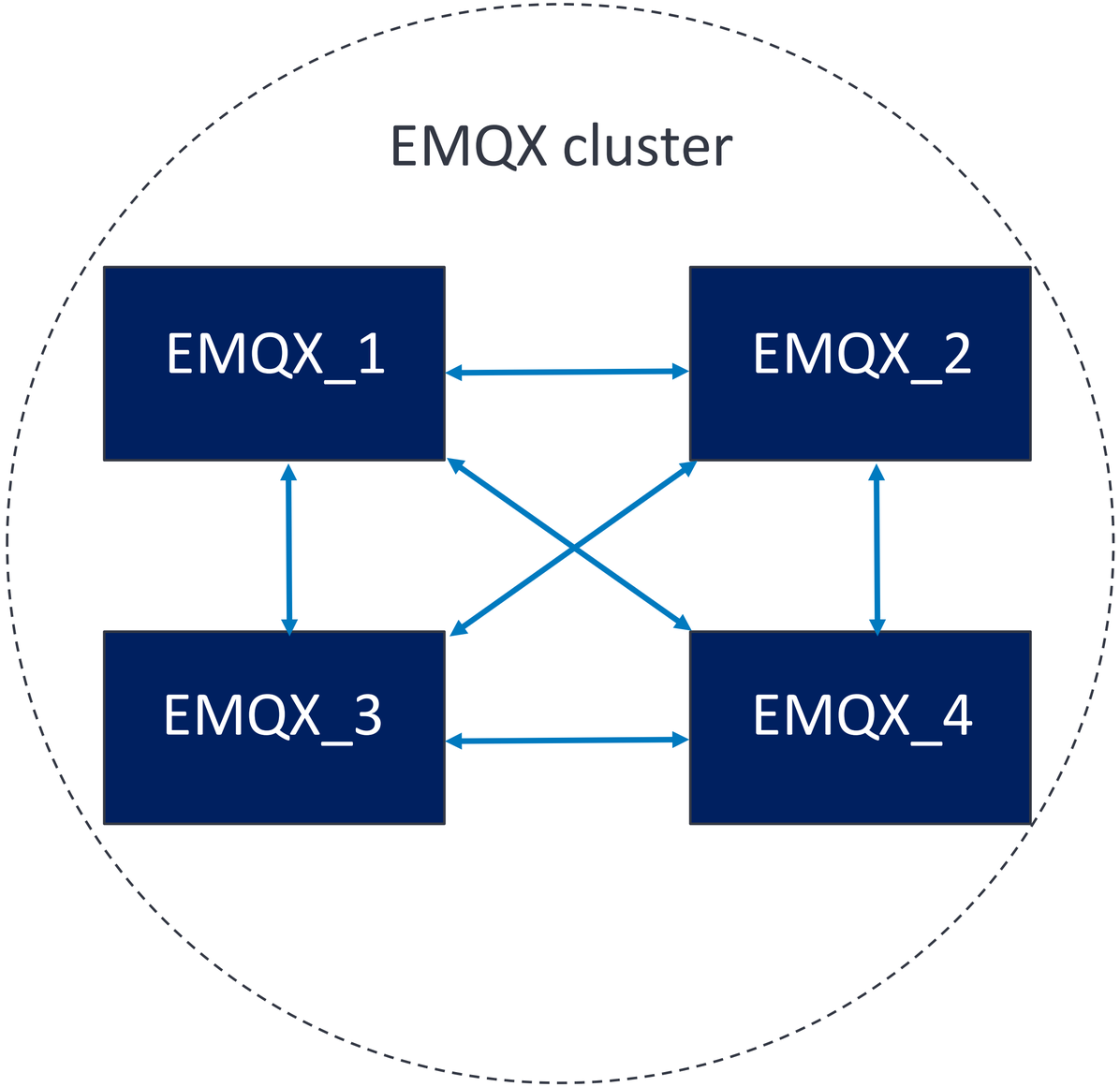
Demo Preparation



Start the 50 K publishers

In emqtt-bench

EMQX clustering currently in EMQX v4.x



Erlang node

Runnable system

BEAM VM

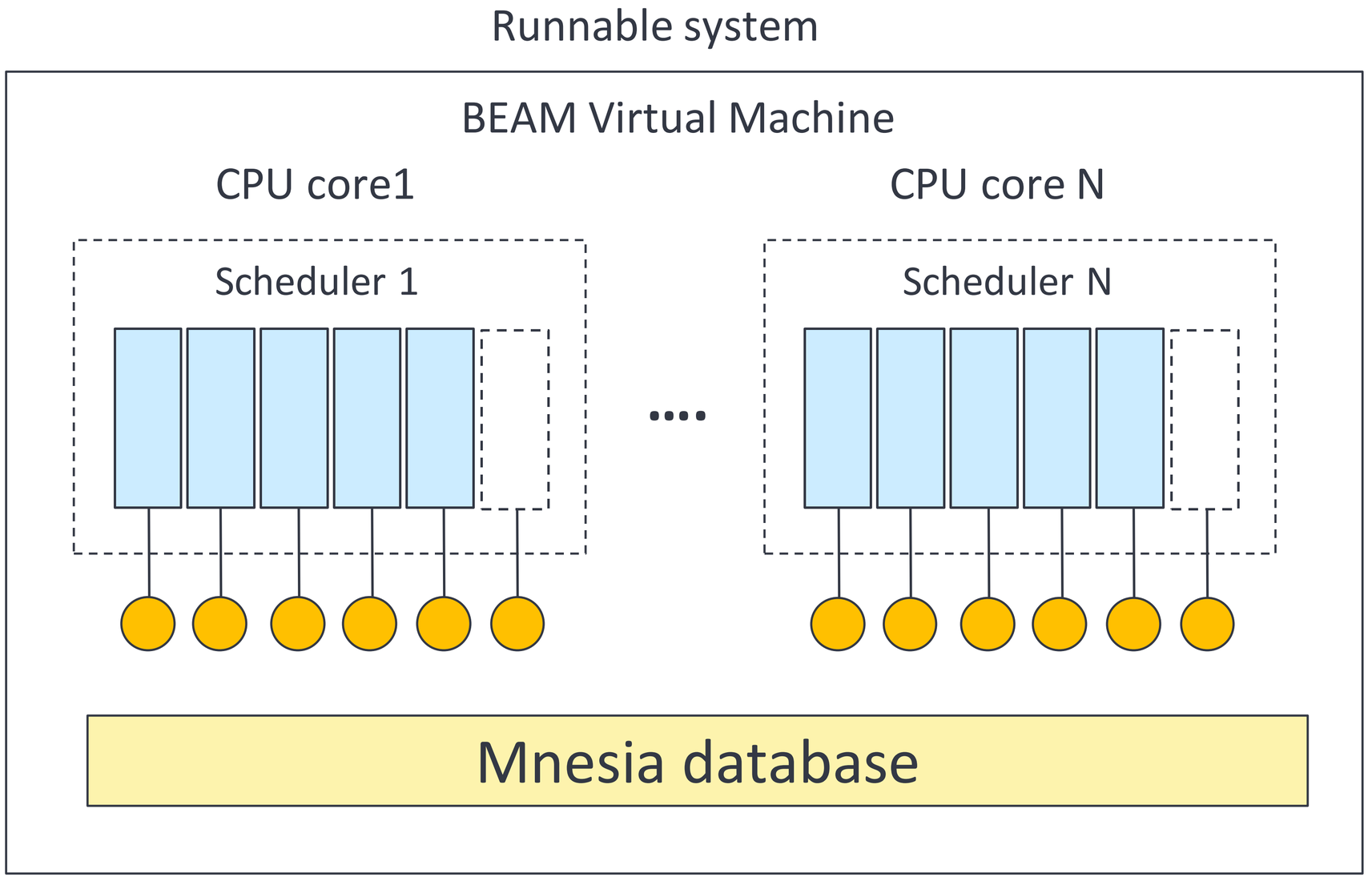
Multiple schedulers

(one per CPU core)

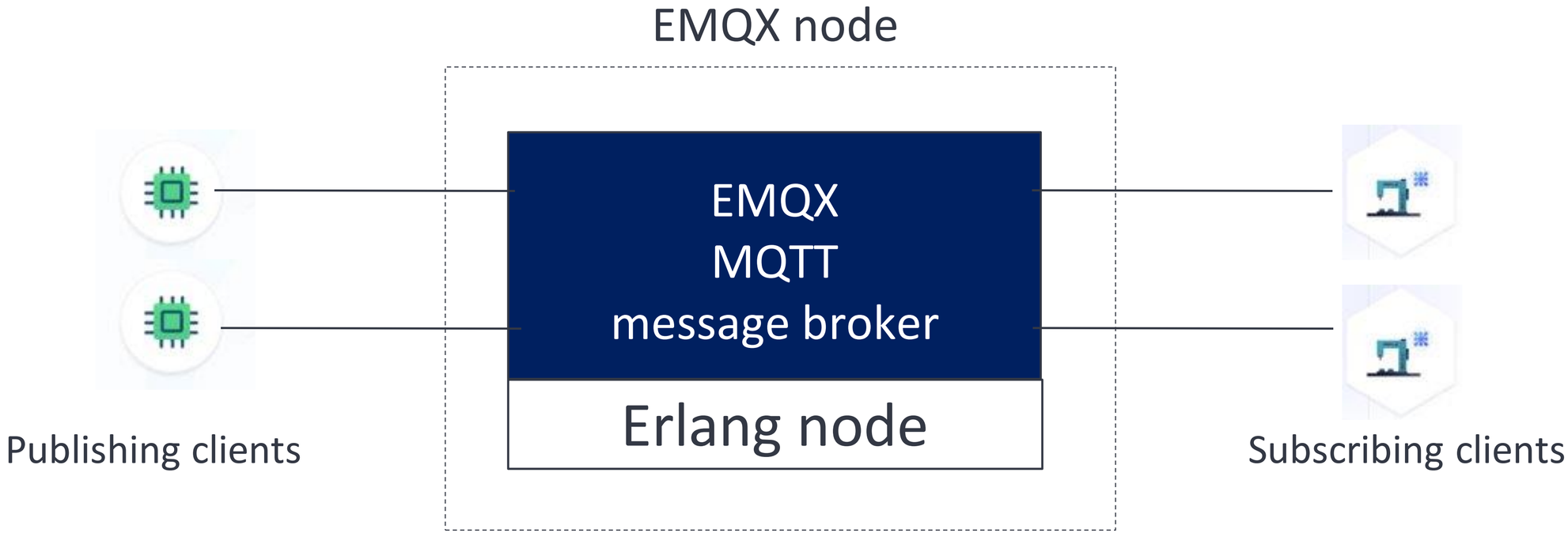
Many concurrent

processes

Mnesia database



EMQX node

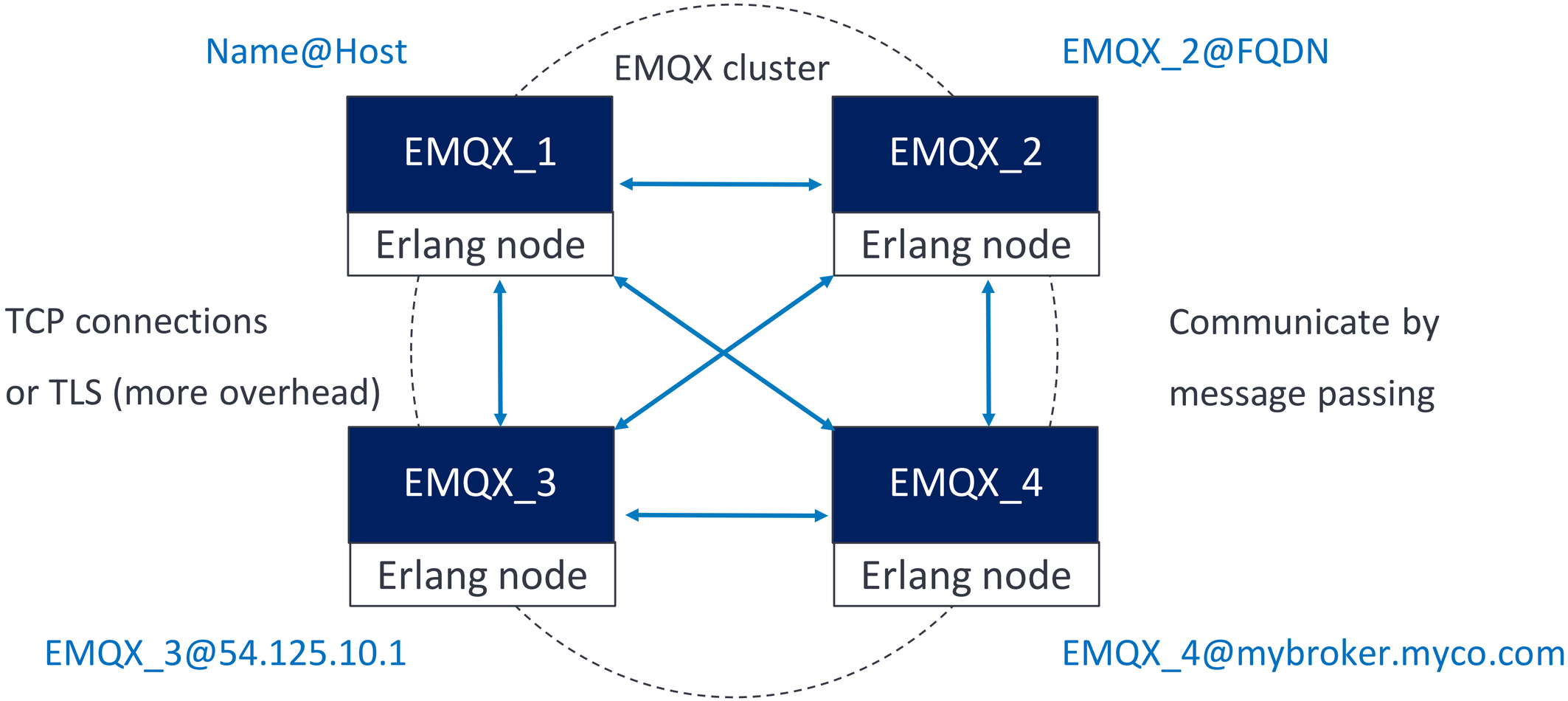


Each EMQX node supports approximately:

- 1 million connected clients
- 1 million messages per second

Not a hard limit. Based on performance.

Distributed EMQX



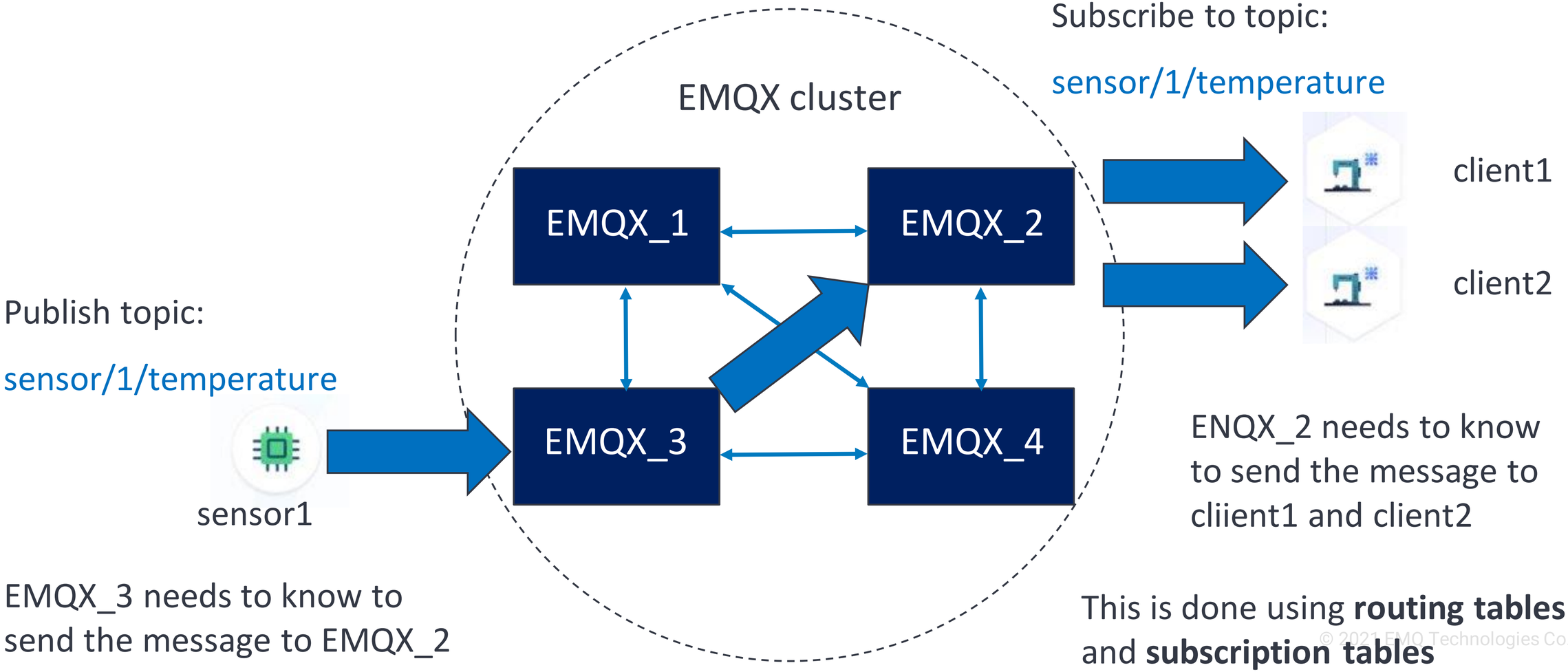
Mesh topology: All nodes have connections to all the other nodes

Nodes join one node then are automatically connected to the cluster

Topic message routing

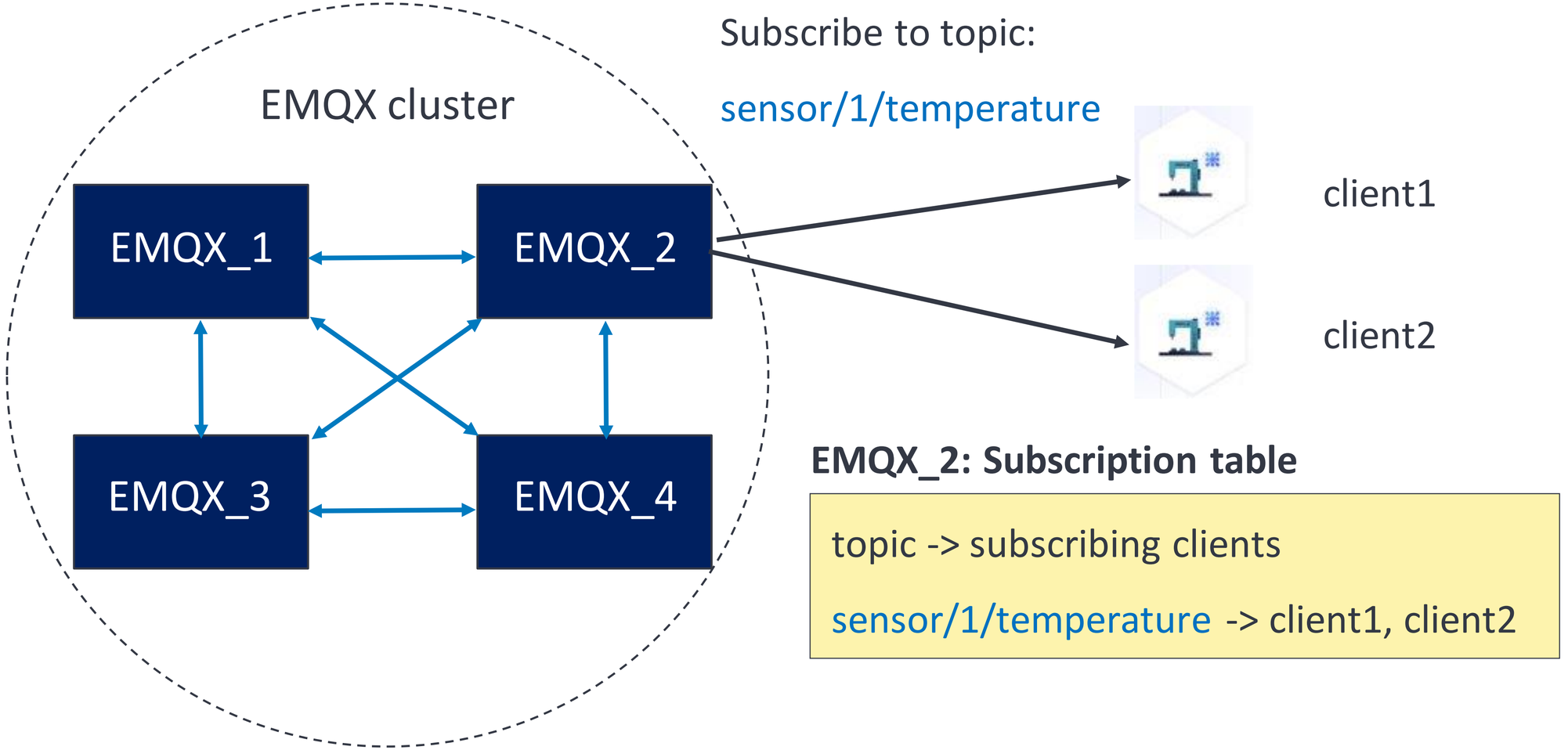
Clients connect to one node in the cluster

Incoming topic messages need to be routed to appropriate nodes and delivered to subscribing clients.



Subscription table

Each node has its own subscription table – which of its clients have subscribed to a topic

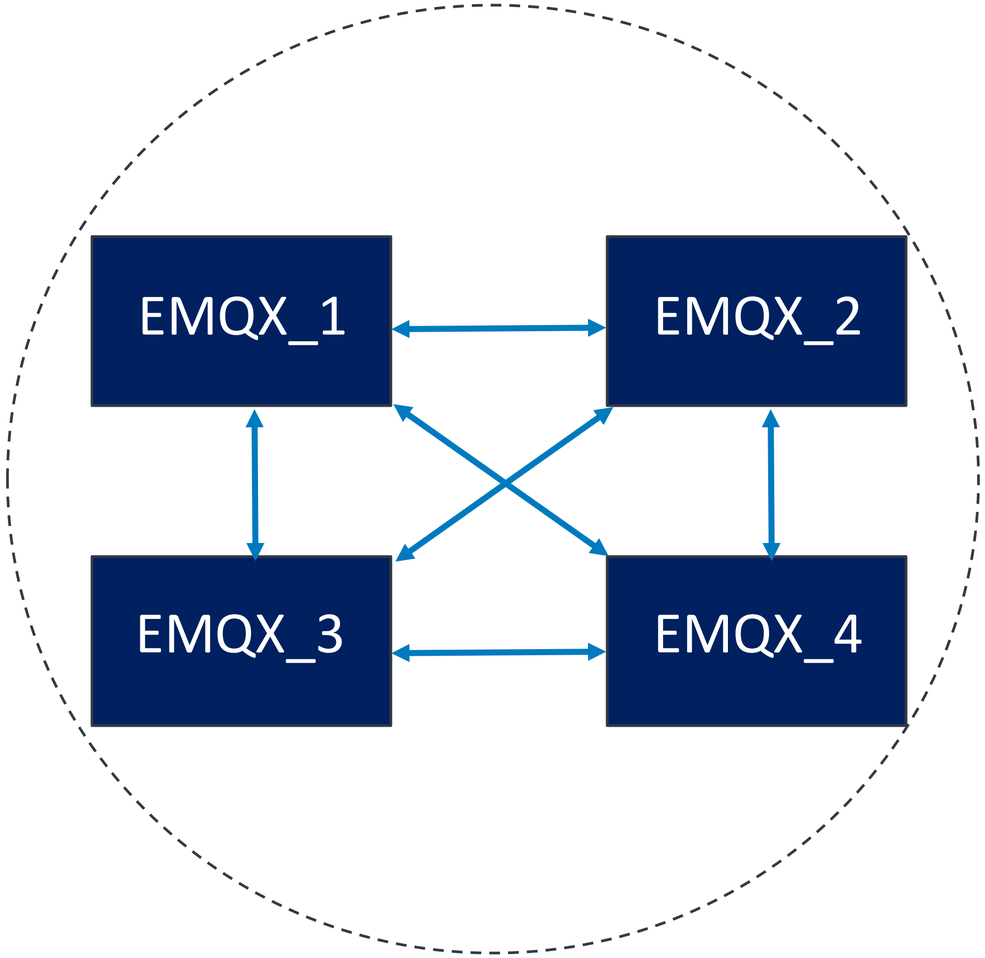


EMQX_2: Subscription table

topic	->	subscribing clients
sensor/1/temperature	->	client1, client2

Nodes then inform other nodes about its topic subscriptions

Routing table



EMQX_2: Subscription table

topic -> subscribing clients

sensor/1/temperature -> client1, client2

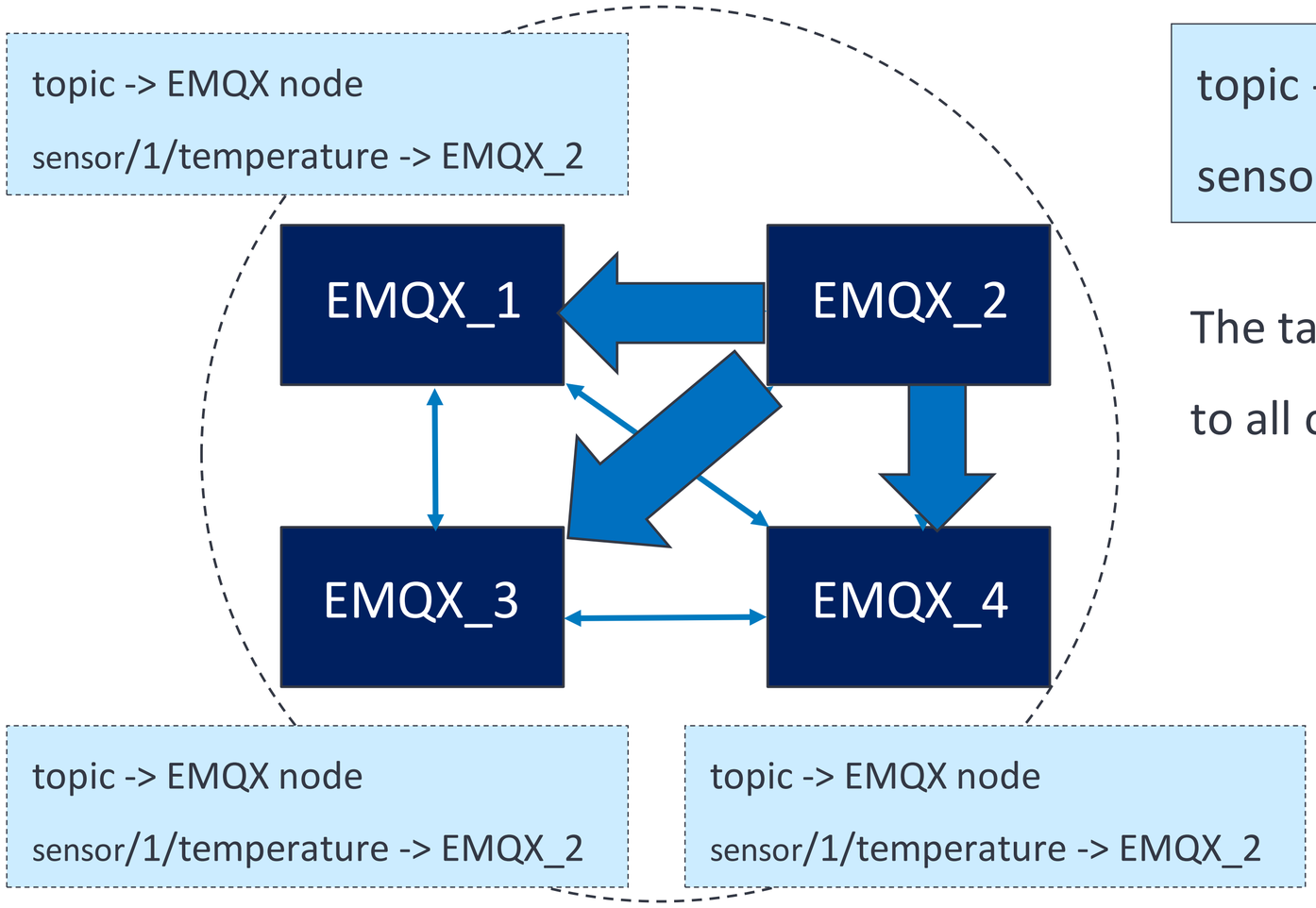
Cluster routing table

topic -> EMQX node

sensor/1/temperature -> EMQX_2

Each node enters its topic subscriptions into its routing table

Routing table replicated to all other nodes



Cluster routing table

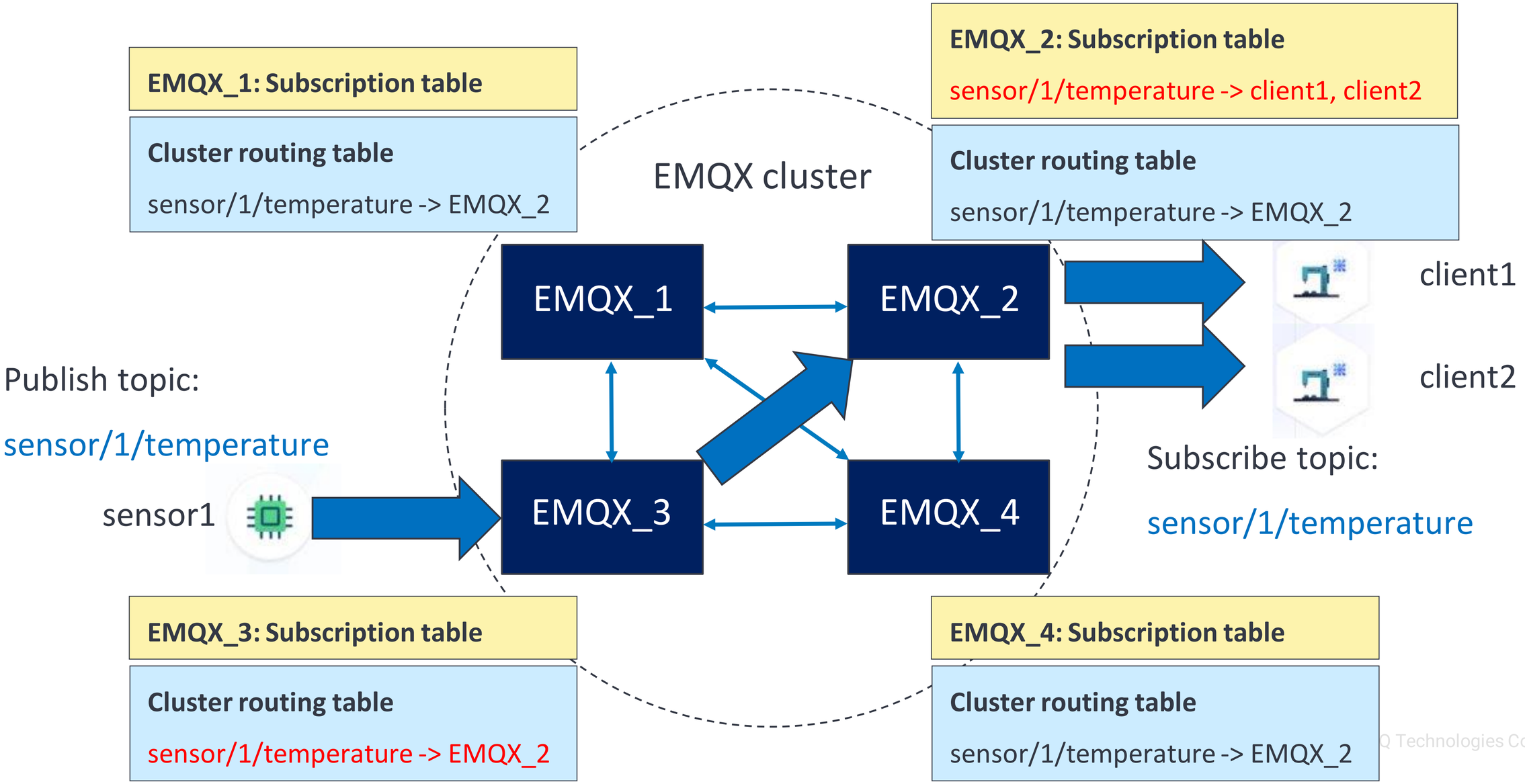
topic -> EMQX node
sensor/1/temperature -> EMQX_2

The table is automatically replicated to all other nodes by the **Mnesia** DB

Each node has own copy of the routing table for the cluster

Message routing using subscription and routing tables

Published message is routed from EMQX_3 to EMQX_2 and then delivered to client1 and client2



Mnesia database

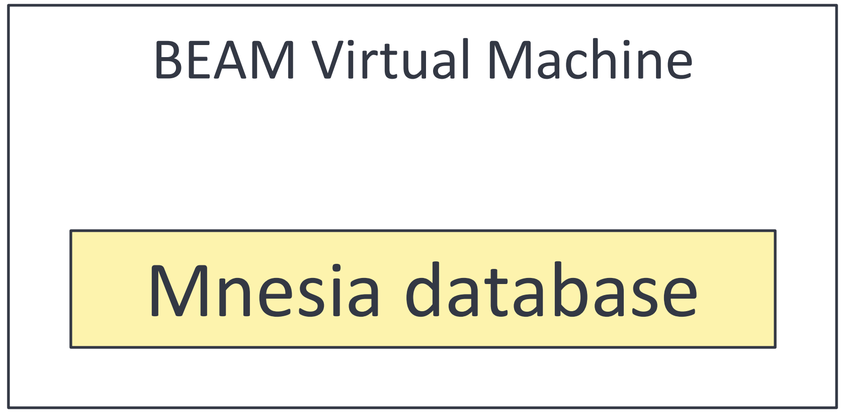
Mnesia is an embedded, distributed, transactional, NoSQL database.

Embedded

Data access is about as fast as with local variables

Distributed

Replicates the table data across all EMQX nodes
All nodes have own local copy of the data
Fault tolerant against nodes going down



Transactional

Multiple read and update operations can be grouped together into one transaction.
Atomic transactions: Entire transaction either completes or fails. No inconsistent states

NoSQL database

Operates with Erlang directly. No SQL query language

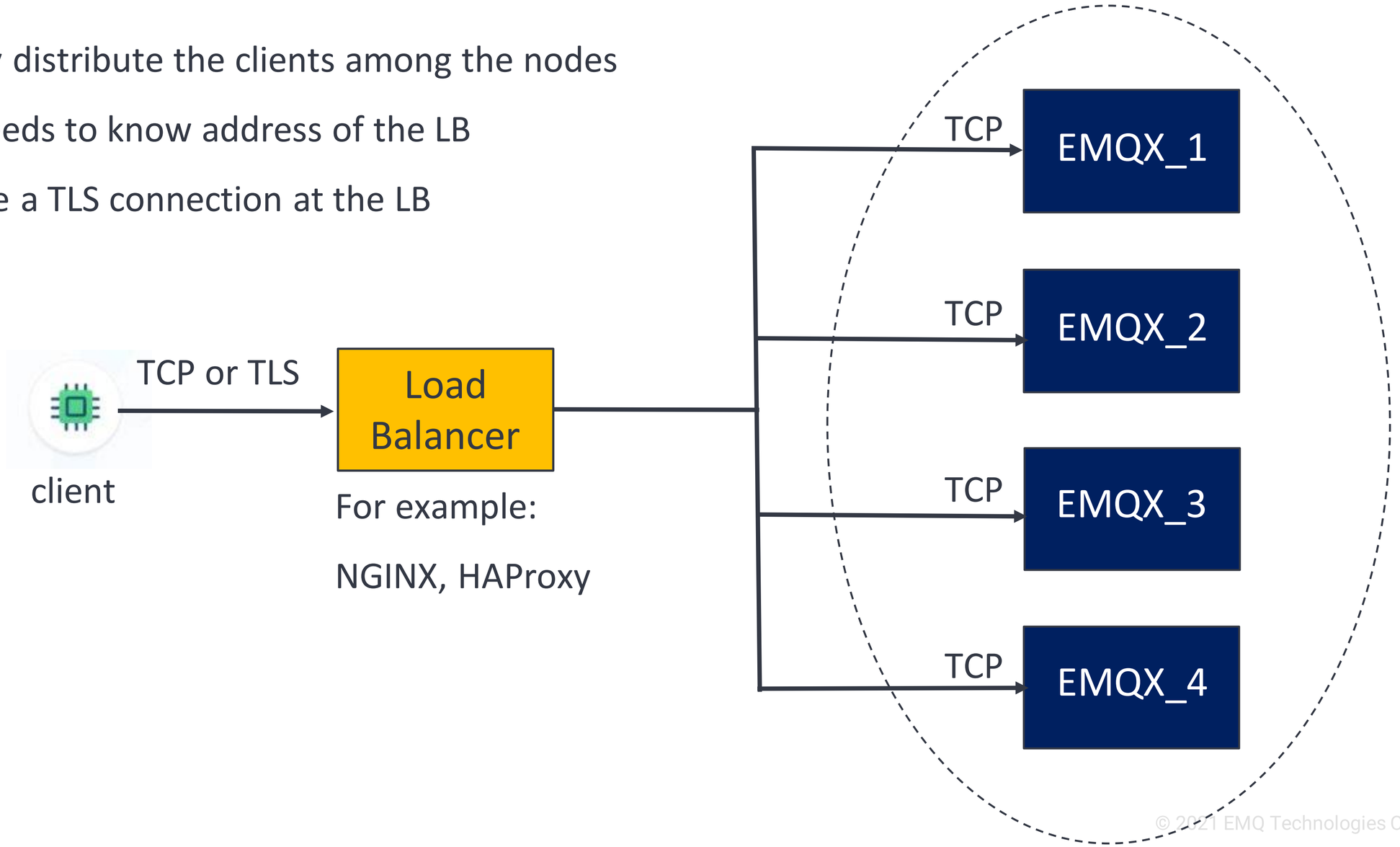
www.emqx.com/en/blog/mqtt-broker-clustering-part-3-challenges-and-solutions-of-emqx-horizontal-scalability

Load balancing

Automatically distribute the clients among the nodes

Client only needs to know address of the LB

Can terminate a TLS connection at the LB



Maximum EMQX nodes and connections per cluster (Pre v5.0)

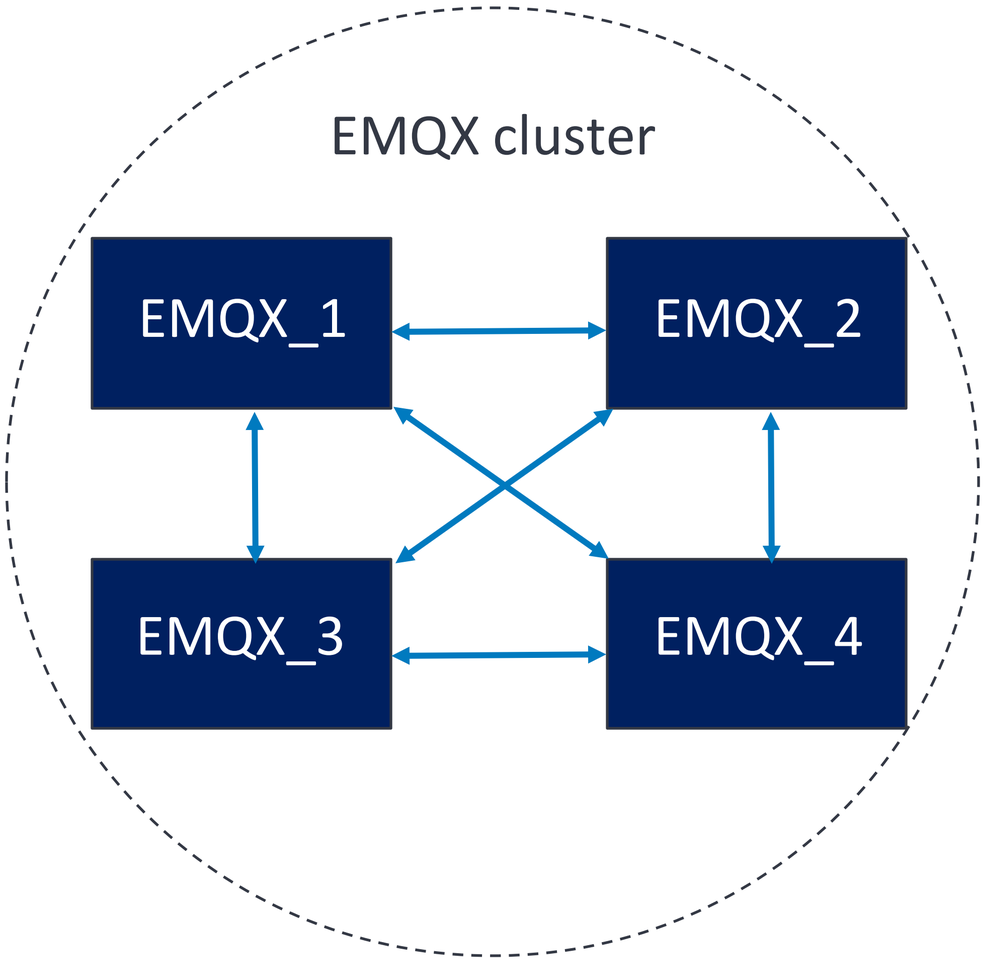
Generally:

Maximum EMQX nodes per cluster: 10

Maximum client connections per node: 1M

Total number of client connections: 10 x 1M = 10M

Not a hard limit, but based upon performance



For N nodes:

Number connections between nodes: N x (N-1) / 2

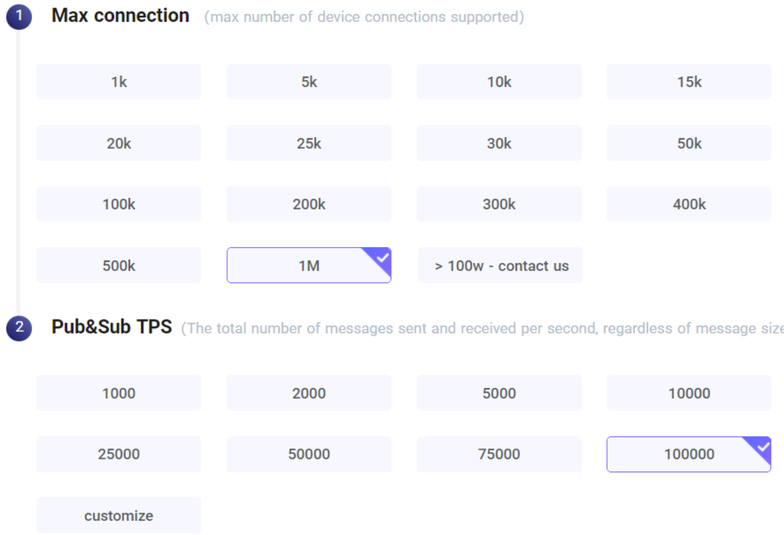
As N get larger:

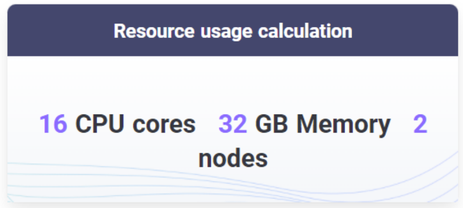
Connections, and therefore routing table replications, increase by a factor of N^2

The N^2 factor limits scalability, but we have solved this in EMQX v5.0.

Server hardware requirements

Server Estimate www.emqx.com/en/server-estimate





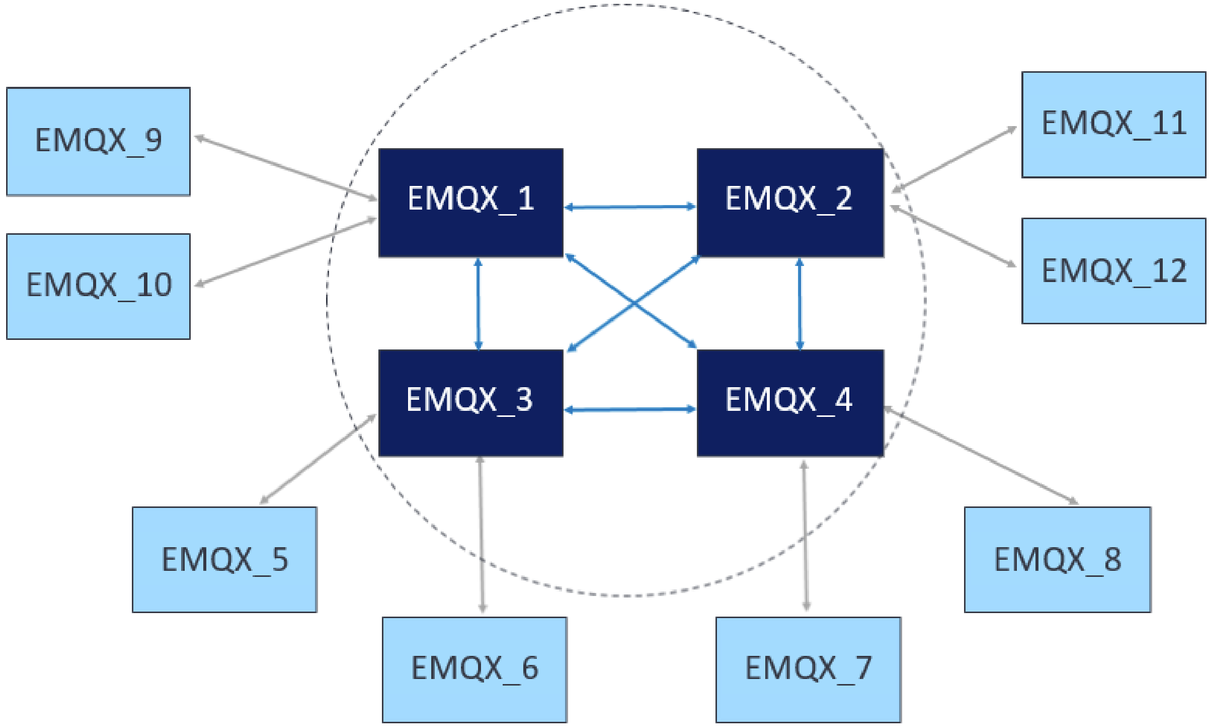
Pub & Sub TPS does not depend on message size

TPS = Transactions Per Second

= Total number messages **published and received** per second

Max connections	Pub & Sub TPS	Memory GB	CPU Cores	Nodes
1 K	100	4	2	2
1 K	20 K	8	4	2
1 M	1 K	32	16	2
1 M	100 K	32	16	2

Improved clustering in EMQX v5.0



100 million subscribers

Blog / EMQX

EMQX Newsletter 2022-01 | 100 million subscribers milestone reached

100-million Milestone Reached!

By the end of January, EMQX team managed to reach 100 million unique wildcard subscribers in a 22-nodes EMQX 5.0 cluster. The team is continue optimising the performance. Then we'll try to run some tests with real traffic. When the release is stable, we'll publish the design and test setup in detail as blog posts.

www.emqx.com/en/blog/emqx-newsletter-202201

EMQX v5.0: 100 M subscribers with 22 nodes

How did we do it?

Answer: We solved the N^2 connection scalability issue as new nodes are added.

N^2 connection scaling: message routing vs table replication

Topic message routing

Performance of topic message routing as new nodes are added is minimally affected by the N^2 connection scaling...

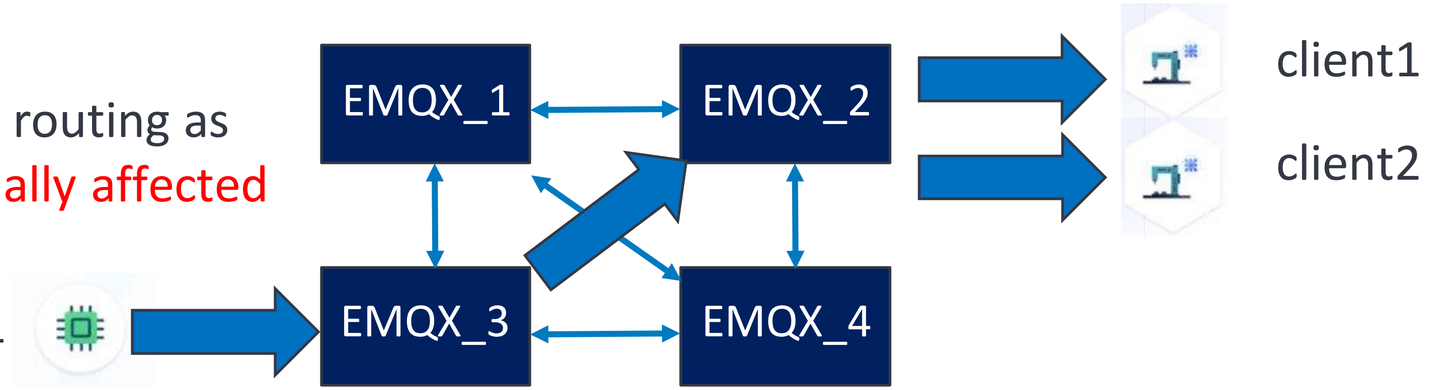
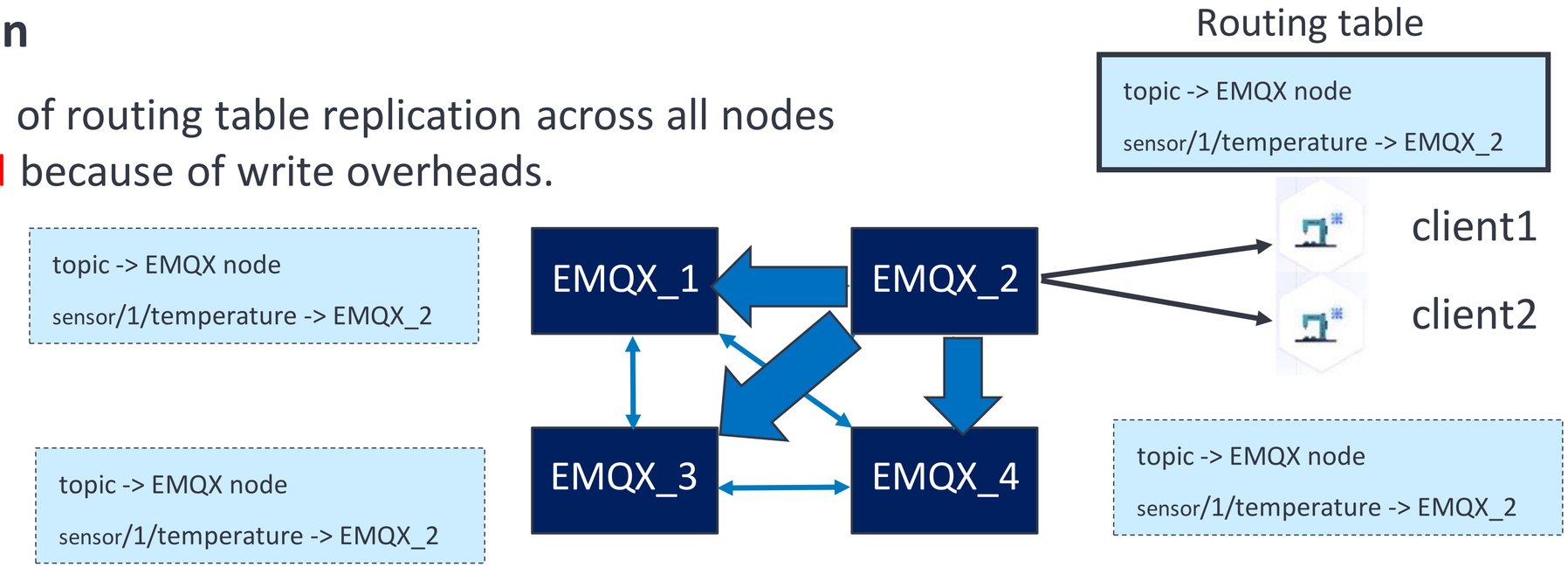


Table replication

But, performance of routing table replication across all nodes is greatly affected because of write overheads.



Routing table needs to be replicated across all nodes every time a client subscribes / unsubscribes to a topic.

Connection scaling solution for EMQX v5.0

Topic message routing

Keep the full-mesh topology
for topic message routing

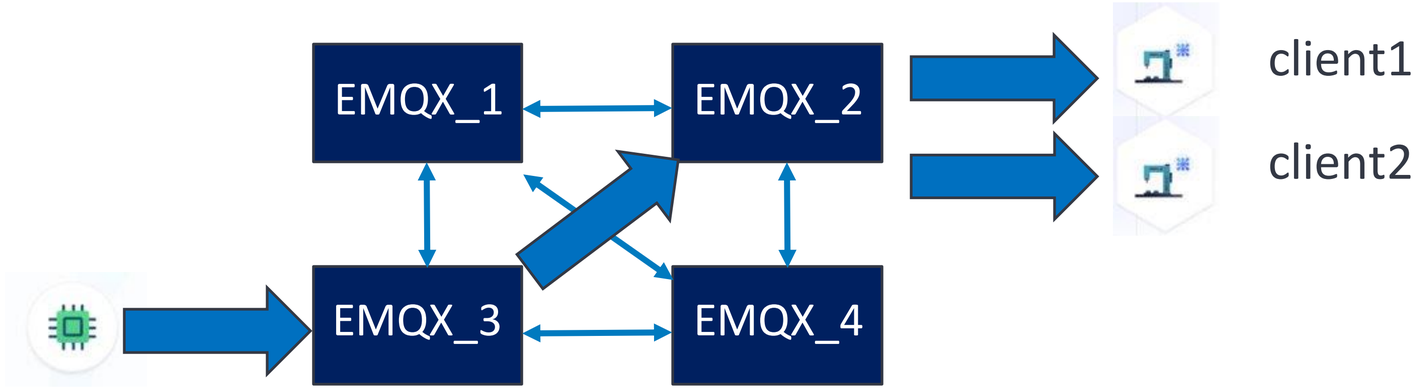


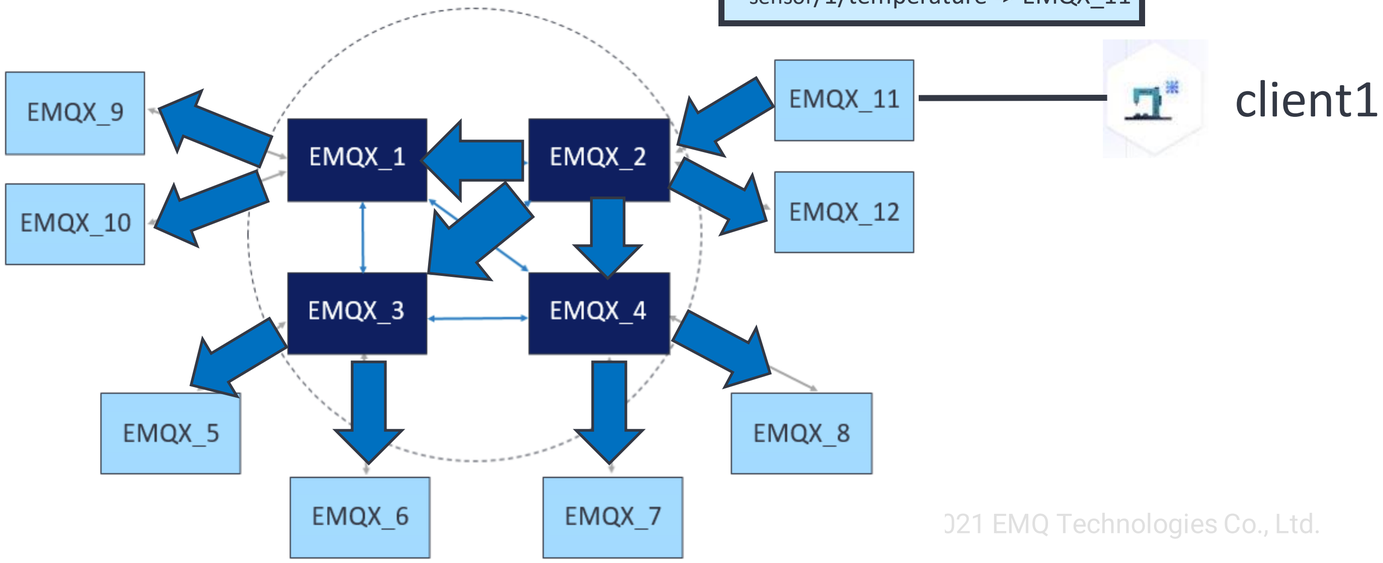
Table replication

Break the full-mesh topology for replication of the cluster routing table

Routing table

topic -> EMQX node

sensor/1/temperature -> EMQX_11



Adding new nodes now scales connections
by a factor of N instead of N^2.

New node types

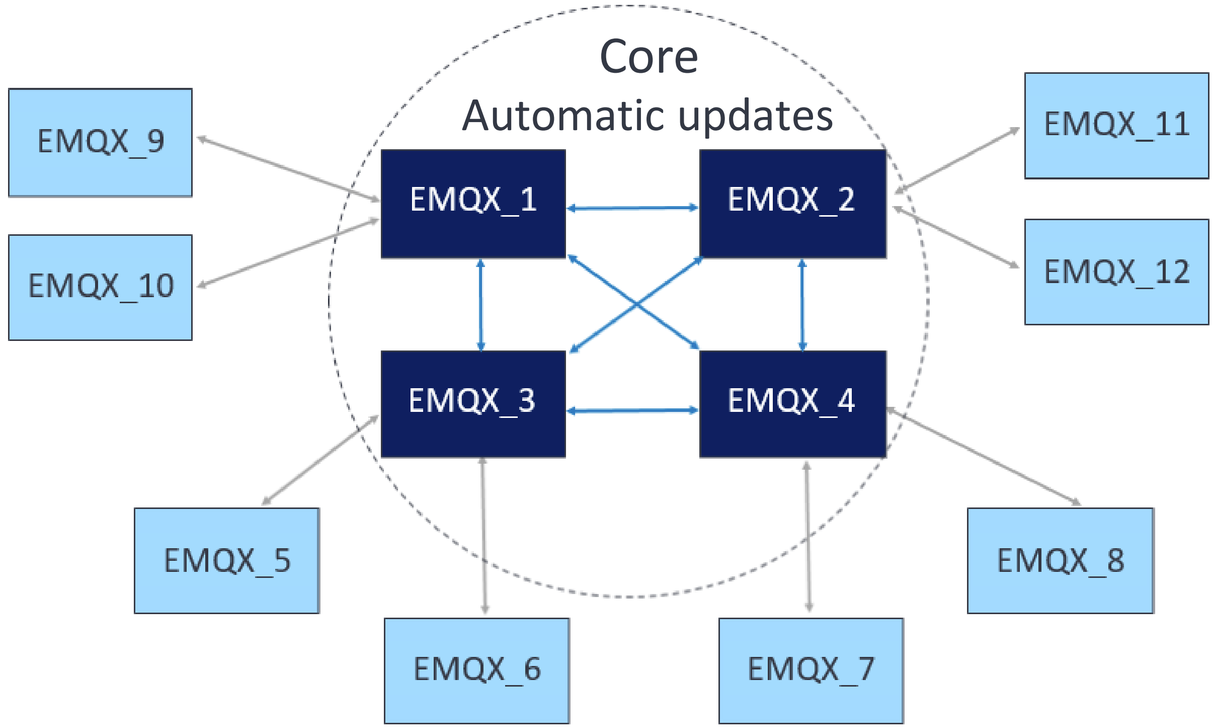
Core node (existing type)

Same as the existing node type

Full mesh topology

Table data is updated synchronously across nodes

Same scaling N^2 problem, but N will be smaller.



Replicant node (new type)

Connects to the core nodes (but **not** to themselves)

Acts like a client to the core node

Requests updates to data which are performed only by the core nodes

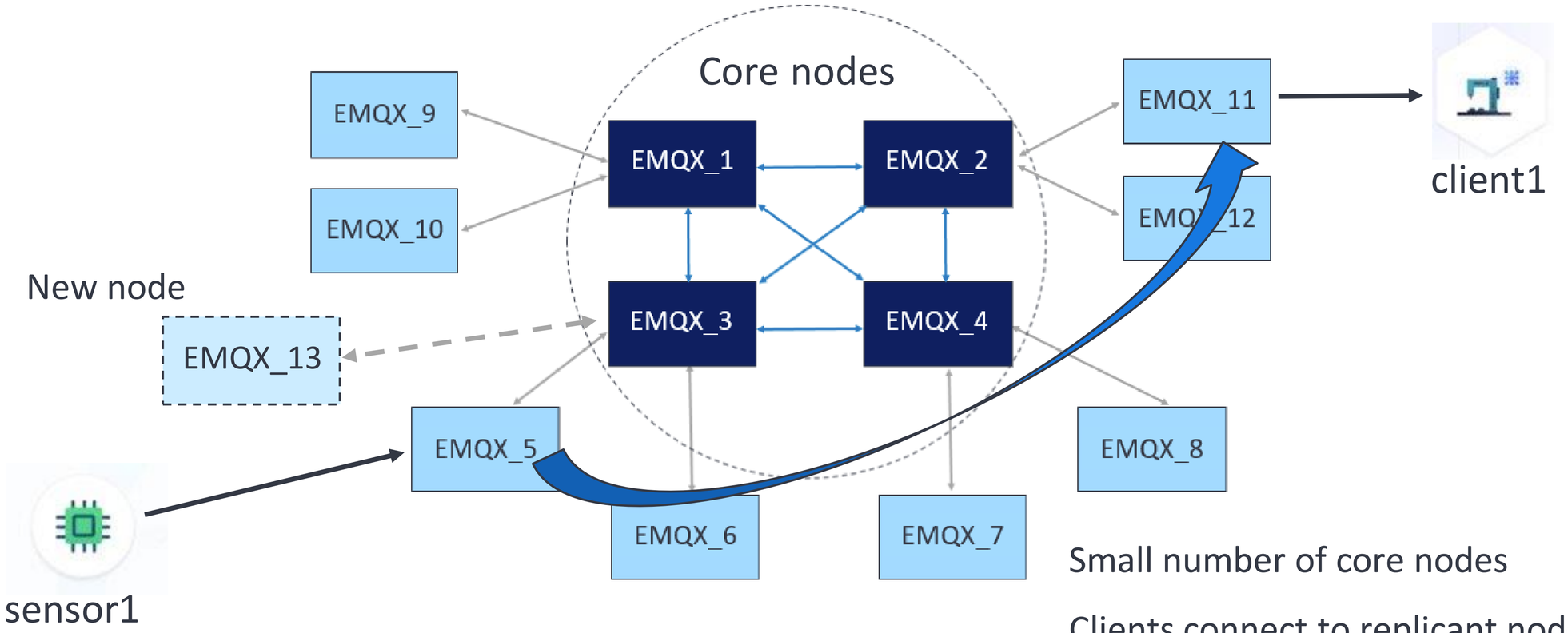
Data is updated asynchronously

Have a local copy of the table data, so fast access times

Number of replicant nodes does not affect the write throughput

Enables autoscaling: adding/removing replicant nodes does not change data redundancy

Core and replicant node topology



Small number of core nodes

Clients connect to replicant nodes

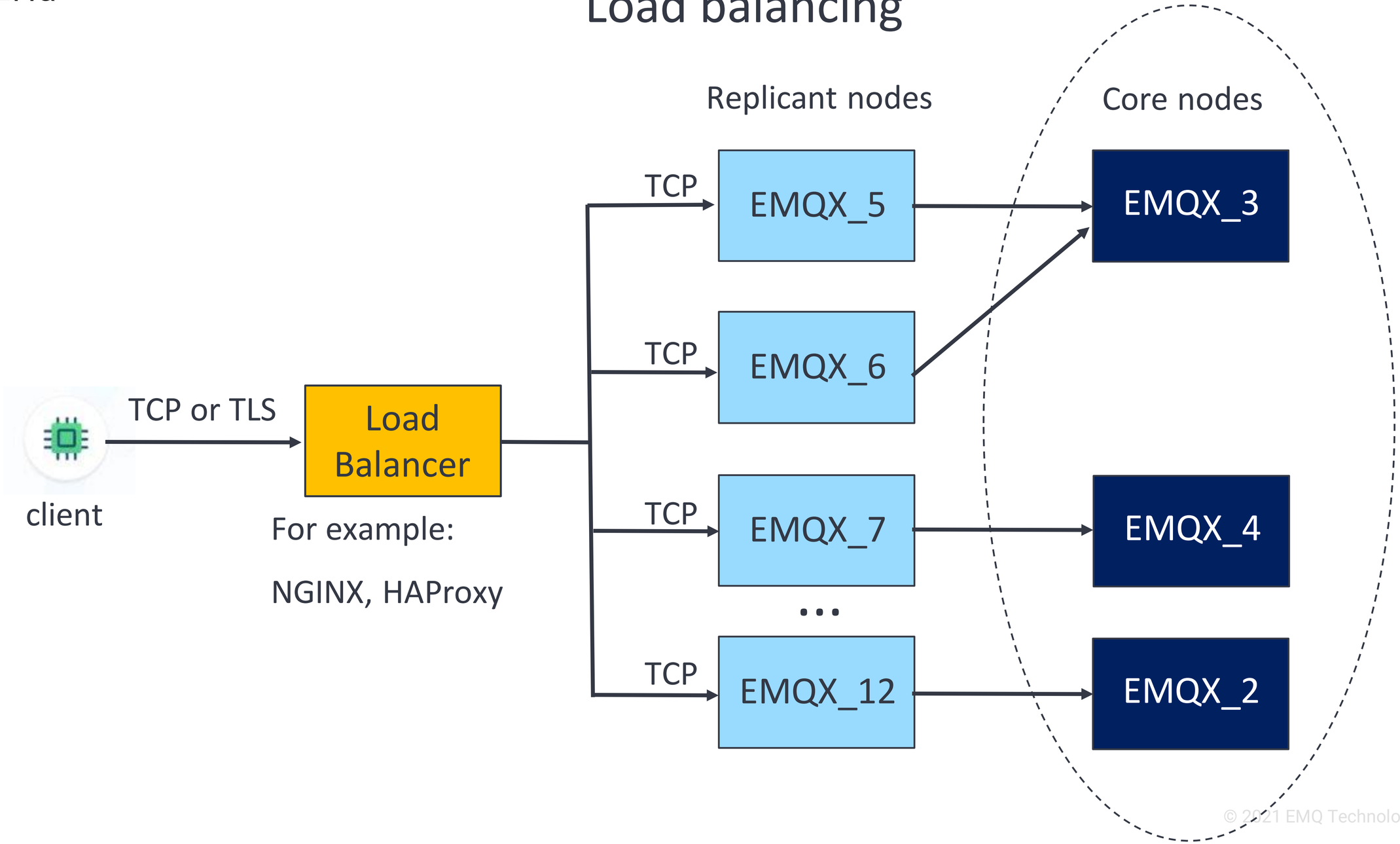
Scaling done by adding / deleting replicant nodes

Adding replicant nodes scales by N Instead of N^2

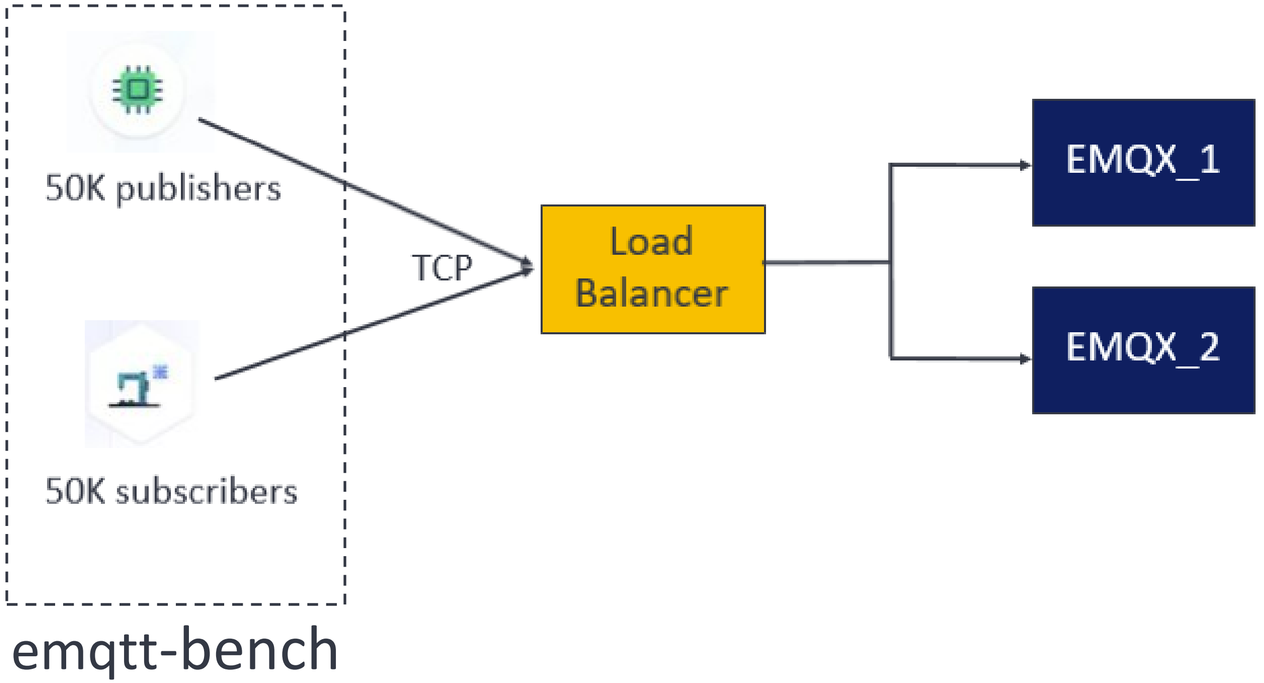
New Mria extension of Mnesia: Based on Mnesia

https://github.com/emqx/mria

Load balancing



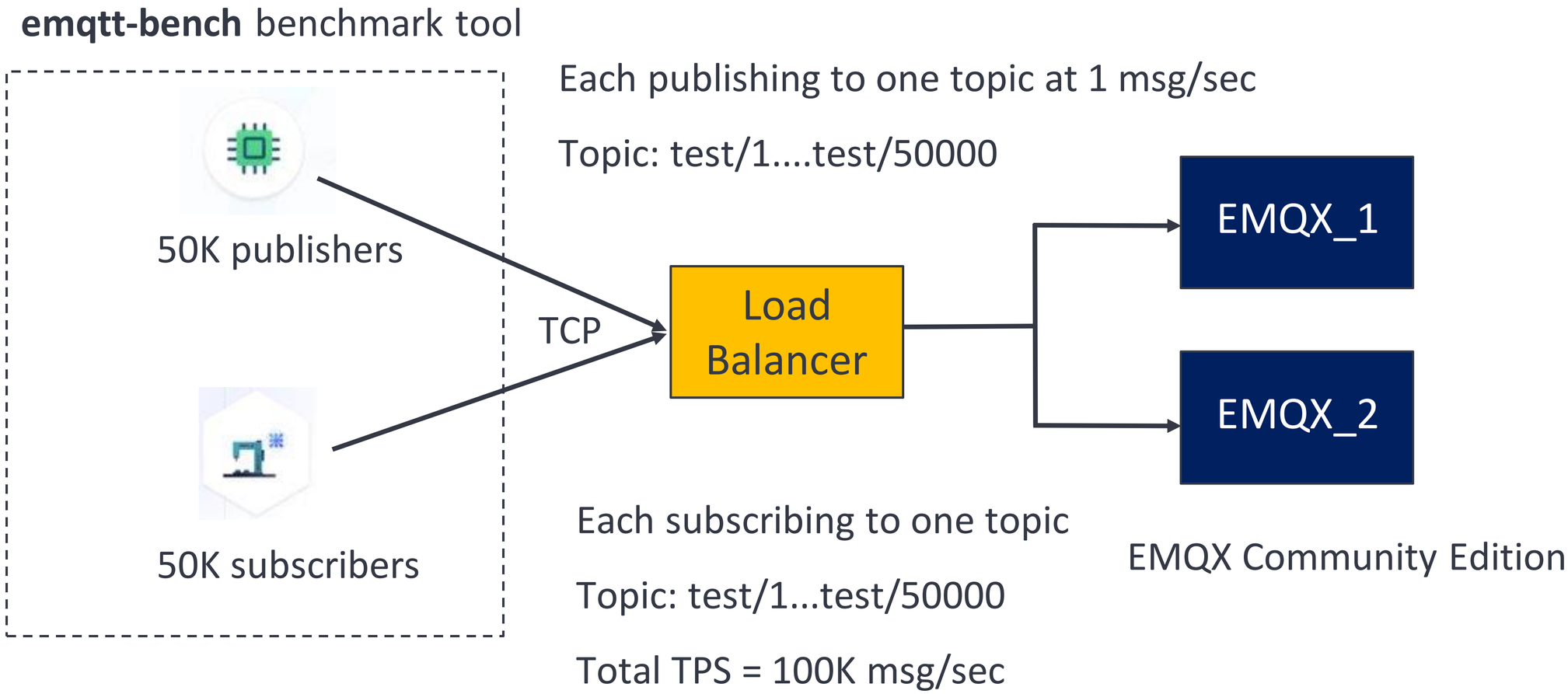
Demo Setup



100K Connections

100K messages / second

Demo setup



Virtualized on AWS using Cloud Development Kit (CDK)

Steps to try it yourself in local installation

Download EMQX Community Edition	www.emqx.io/downloads
Download emqtt-bench benchmark tool	https://github.com/emqx/emqtt-bench
Verify computer hardware requirements	www.emqx.com/en/server-estimate
Verify tuning requirements	www.emqx.io/docs/en/v4.4/tutorial/tune.html

emqtt-bench commands

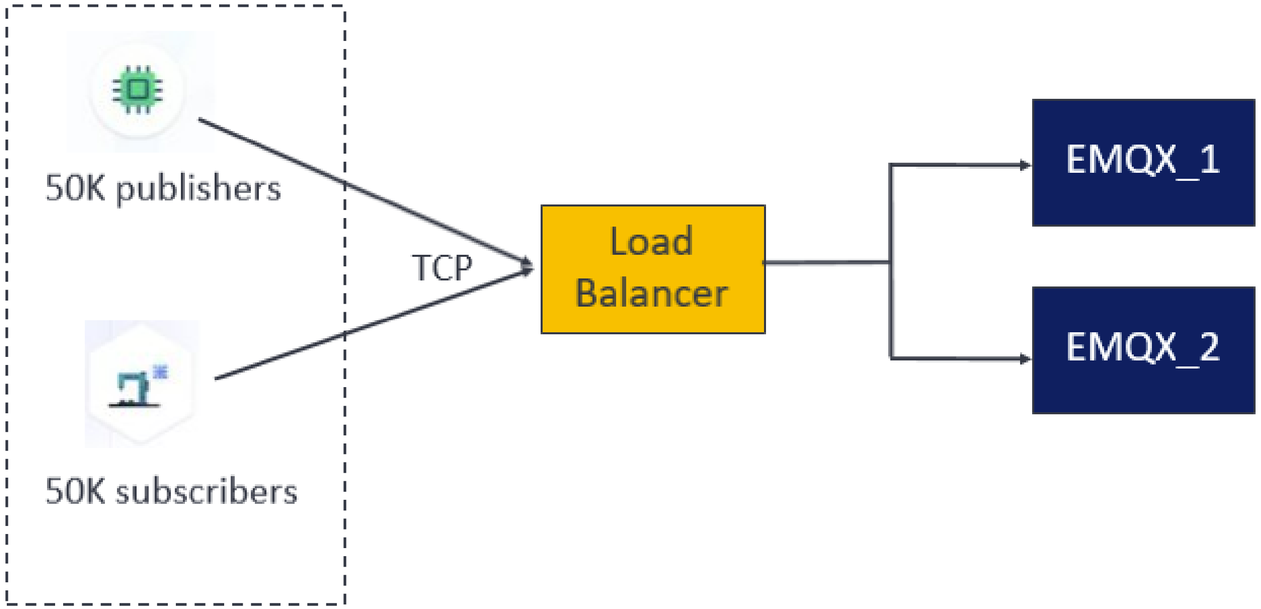
Subscribe

```
./emqtt_bench sub -c 50000 -i 5 -t test/%i --qos 1  --ifaddr 192.168.0.10, 192.168.0.20
```

Publish

```
./emqtt_bench pub -c 50000 --qos 1 --inflight 1 --interval_of_msg 1000  -t test/%i  --ifaddr 192.168.1.10, 192.168.0.20
```

Demo



100K Connections

100K messages / second

Welcome to join EMQX Community

Discord

https://discord.gg/C2zpUvPnRC

slack

https://slack-invite.emqx.io/

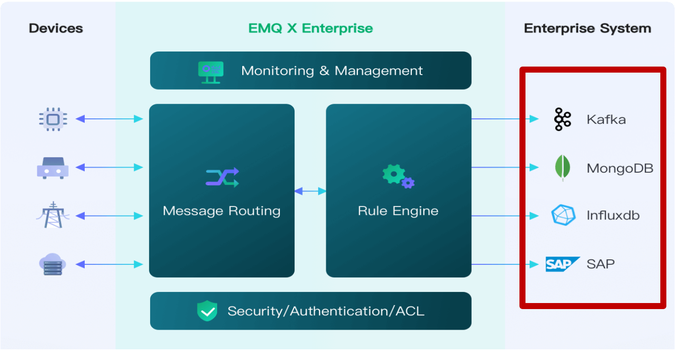
Forum

https://github.com/emqx/emqx/discussions

Happy to discuss your favorite topics with YOU

Questions and Answers

Q & A



Sign up for your free trial today! www.emqx.com

Try Free →

And try out the demo yourself