SCALING DISTRIBUTED ERLANG

Zandra – Erlang/OTP - Ericsson

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

OVERVIEW

Distributed Erlang Today

Future Plans/Considerations

Scaling Distributed Erlang

DISTRIBUTED ERLANG

Used for Elixir distribution too

Multiple Interconnected Nodes

Peer to Peer

Fully Connected

WHAT IS A NODE?

Erlang VM/beam
- -name/-sname
- OS process

Many concurrent processes
- Scheduling
- Load Balancing
- Message Passing
- ...

MESSAGE PASSING

Transparent (local/remote processes):

```
send pid, message
```

Not transparent (only local processes):

```
send name, message
```

```
send {name, node}, message
```

LOCAL REGISTRATION

```
Process.register(pid, name)

Process.unregister(name)

Process.whereis(name)

Process.send(name, message)
```

:global

```
register_name(name, pid)

re_register_name(name, pid)

unregister_name(name)

whereis_name(name)

send(name, message)
```

Node joins...

WILL THIS SCALE?

WILL THIS SCALE?

No....

Works up to 32-50 nodes

We are working on it...

SCALABILITY PLANS

Issue: Connections stay up
Plan: Automatic disconnects

SCALABILITY PLANS

Issue: Connections stay up
Plan: Automatic disconnects

Issue: Too many connections
Plan: Avoid fully connected network

SCALABILITY PLANS

Issue: Connections stay up
Plan: Automatic disconnects

Issue: Too many connections
Plan: Avoid fully connected network

Issue: Expensive when new nodes join
Plan: Make node joins cheaper

SCALABILITY PLANS

Issue: Connections stay up
Plan: Automatic disconnects

Issue: Too many connections
Plan: Avoid fully connected network

Issue: Expensive when new nodes join
Plan: Make node joins cheaper

Issue: `:global` chats a lot
Plan: Make `:global` less chatty

:GLOBAL ISSUES

Fully Connected

`:global` sets it up...

file descriptors not unlimited

Avoid it now:

`-connect_all false`

Plan:

Change `:global`

:global

```
register_name(name, pid)

re_register_name(name, pid)

unregister_name(name)

whereis_name(name)

send(name, message)
```

Node joins...

DISTRIBUTED HASH TABLE

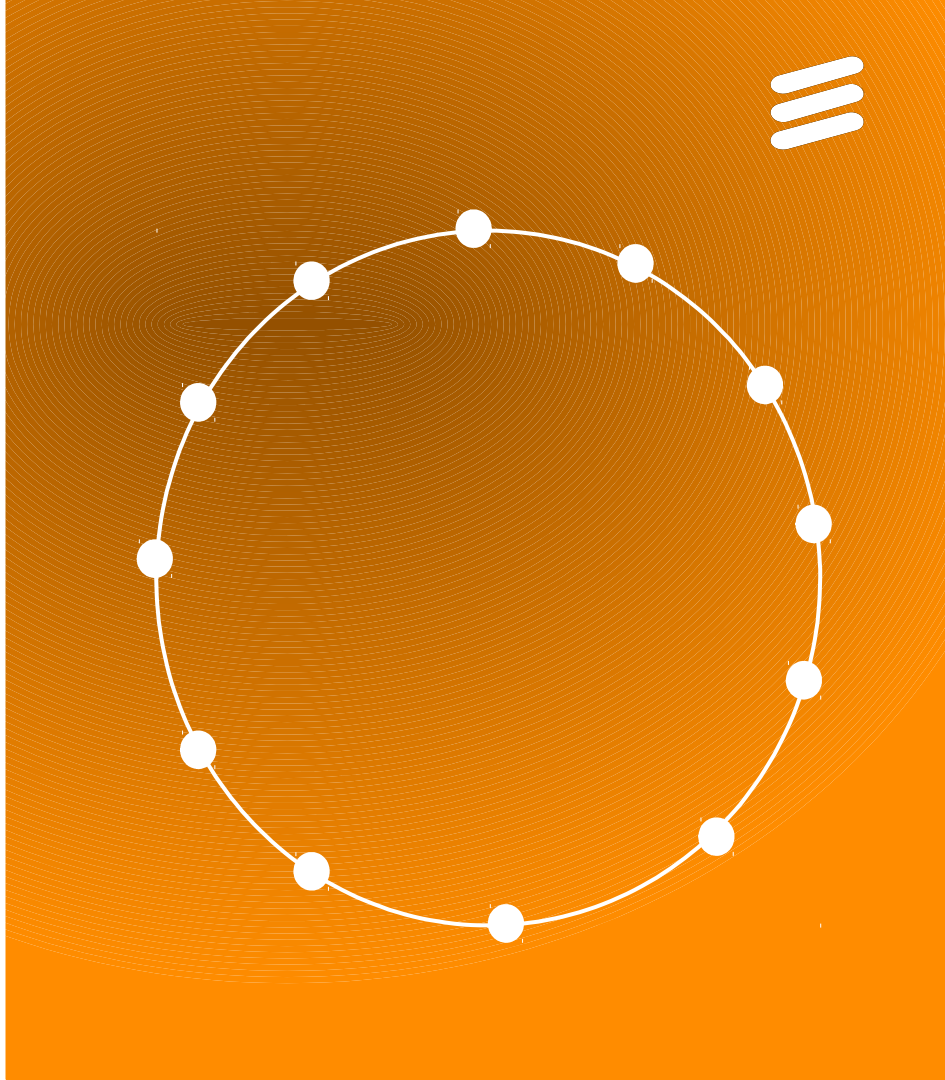
Distributed Key Value store

Consistent Hashing

Decentralized

Scalable

Fault Tolerant

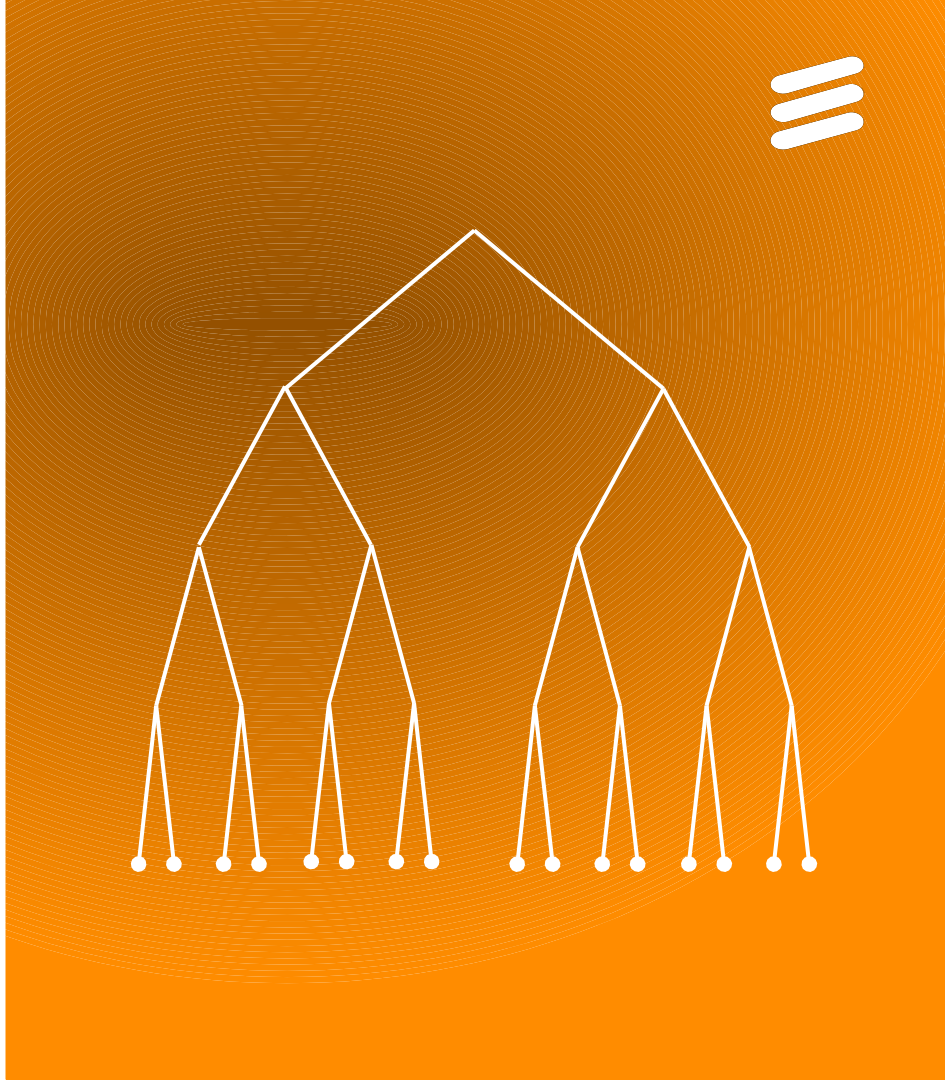


OUR CHOICE: KADEMLIA

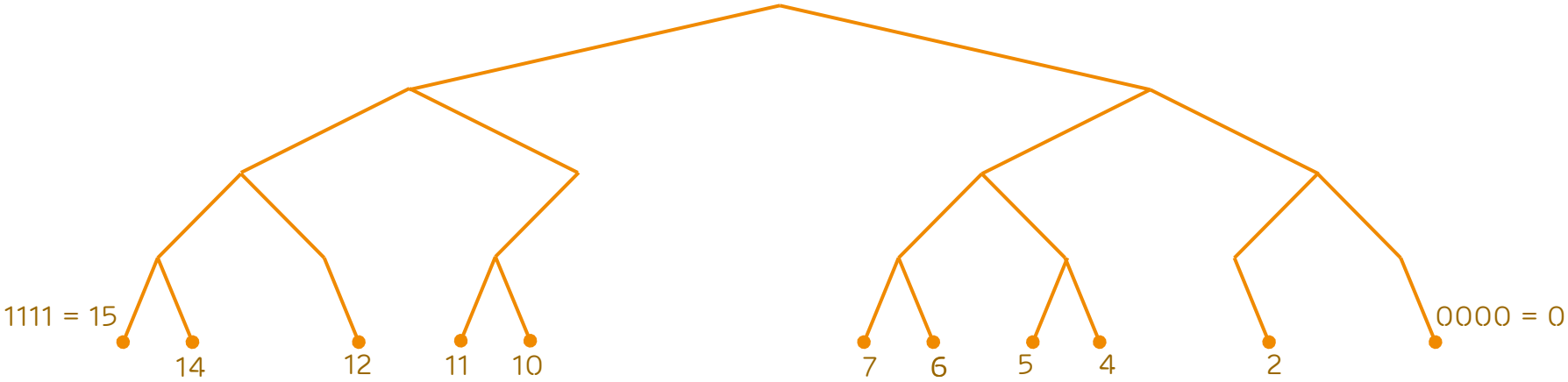
O(log(n)) lookup & store time

Easy to maintain

Config info spread during lookups



KADEMLIA: TREE



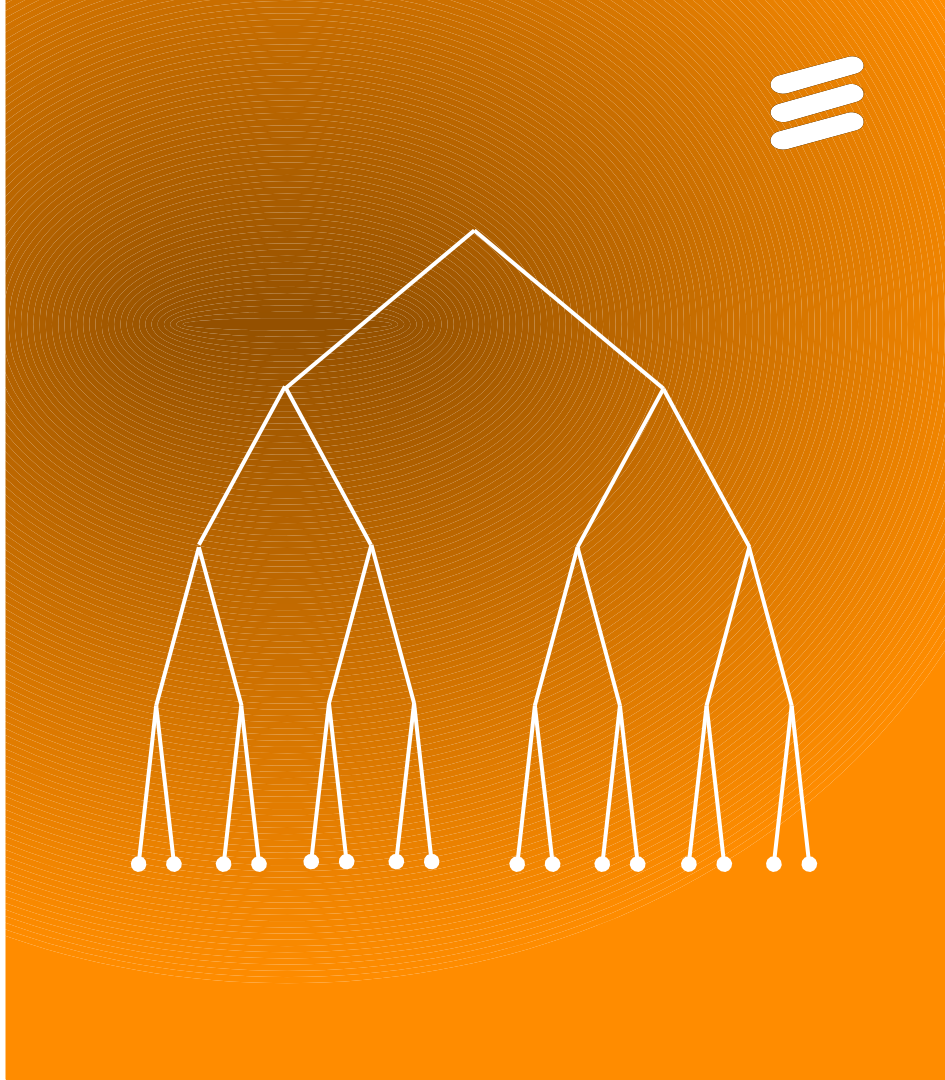
KADEMLIA: TREE

Nodes:

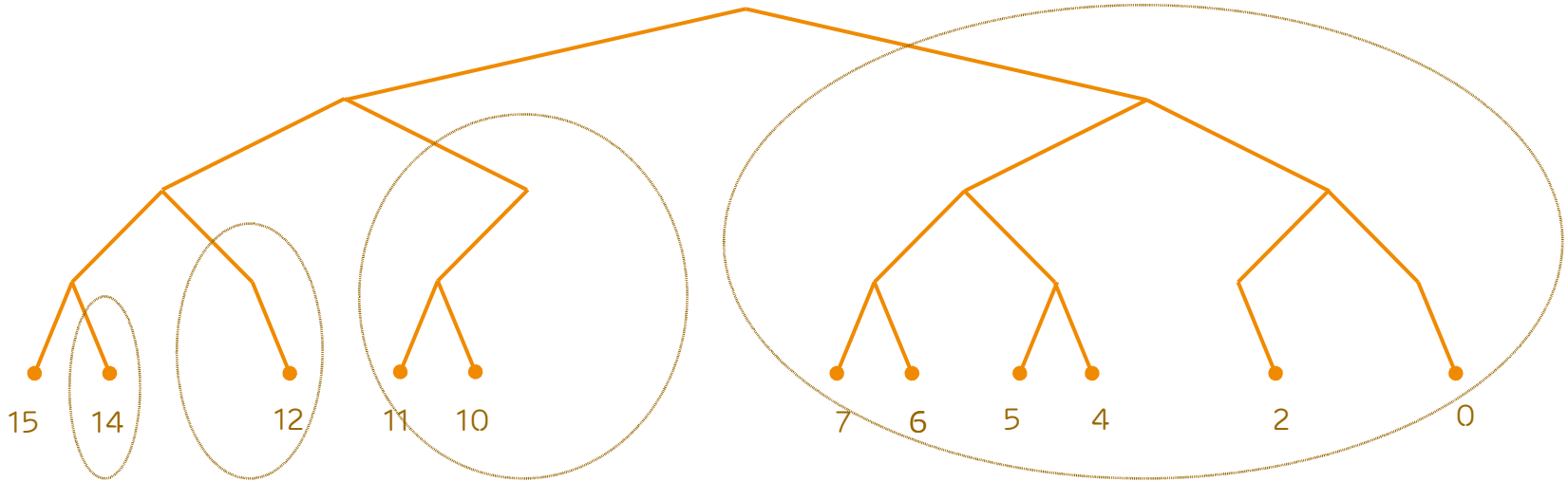
- Leaves in a binary search tree

Node ID = hash(node_name)

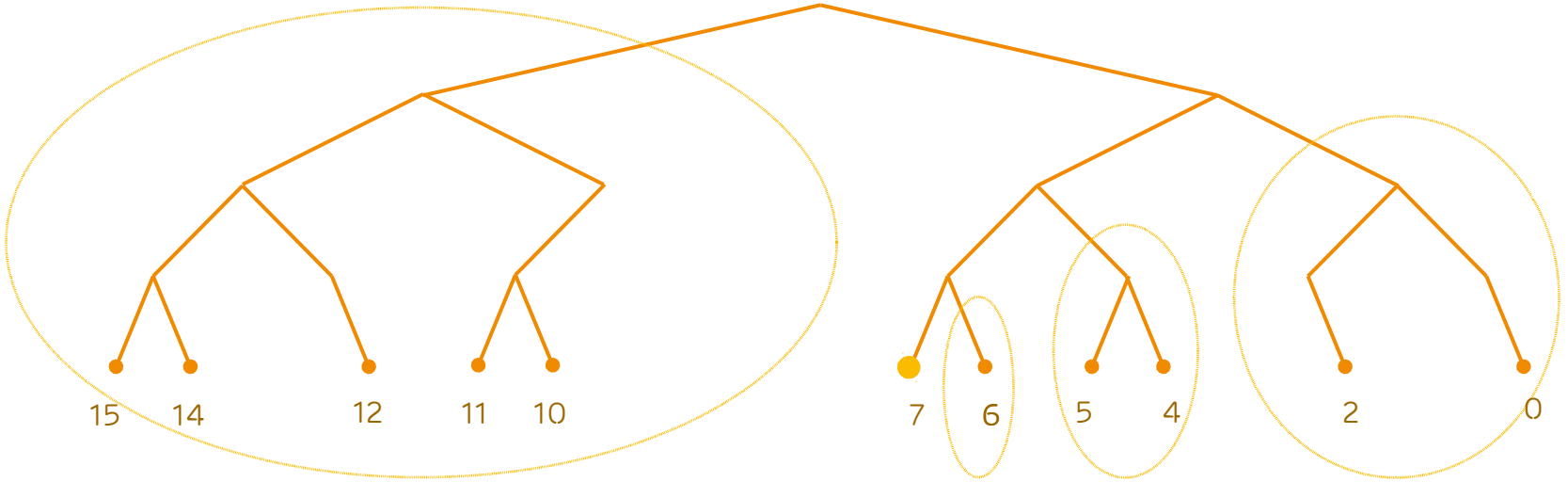
- Good distribution
- Used to locate values



KADEMLIA: SUBTREES



KADEMLIA: SUBTREES

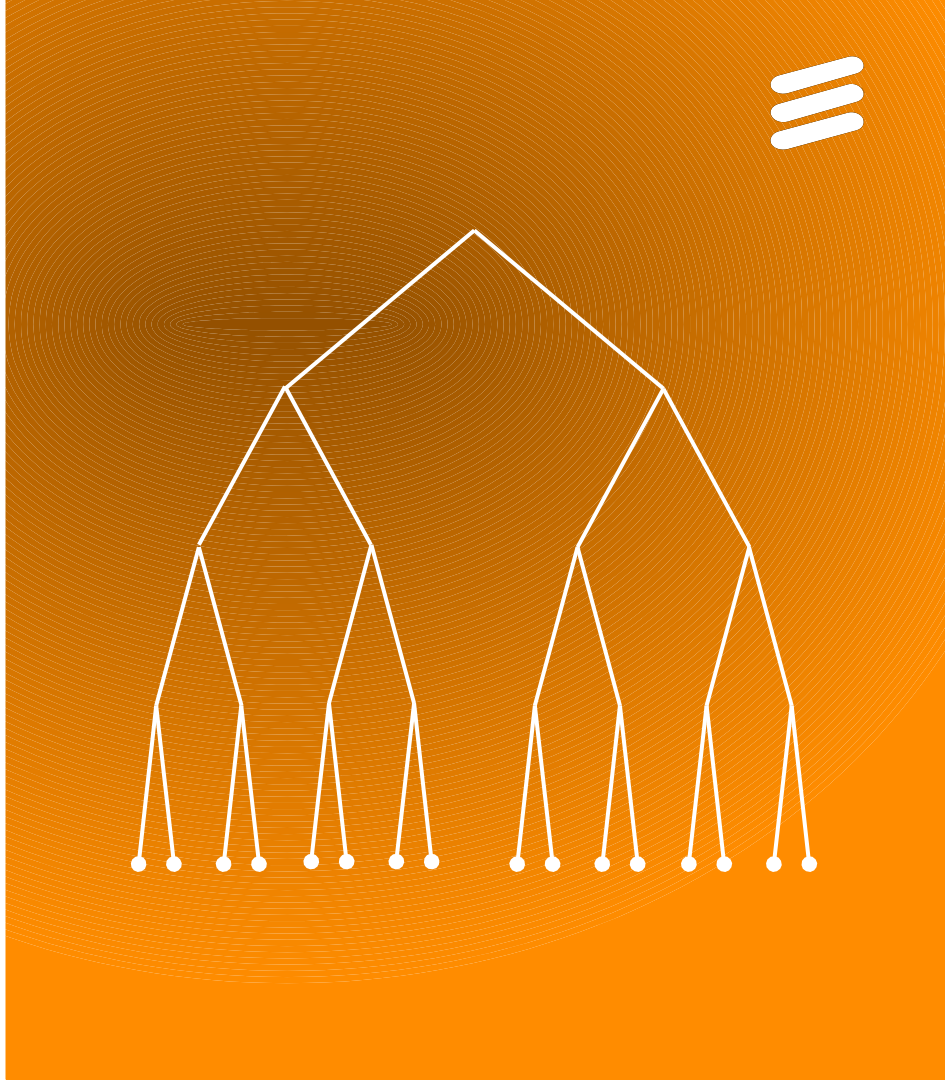


SUBTREES

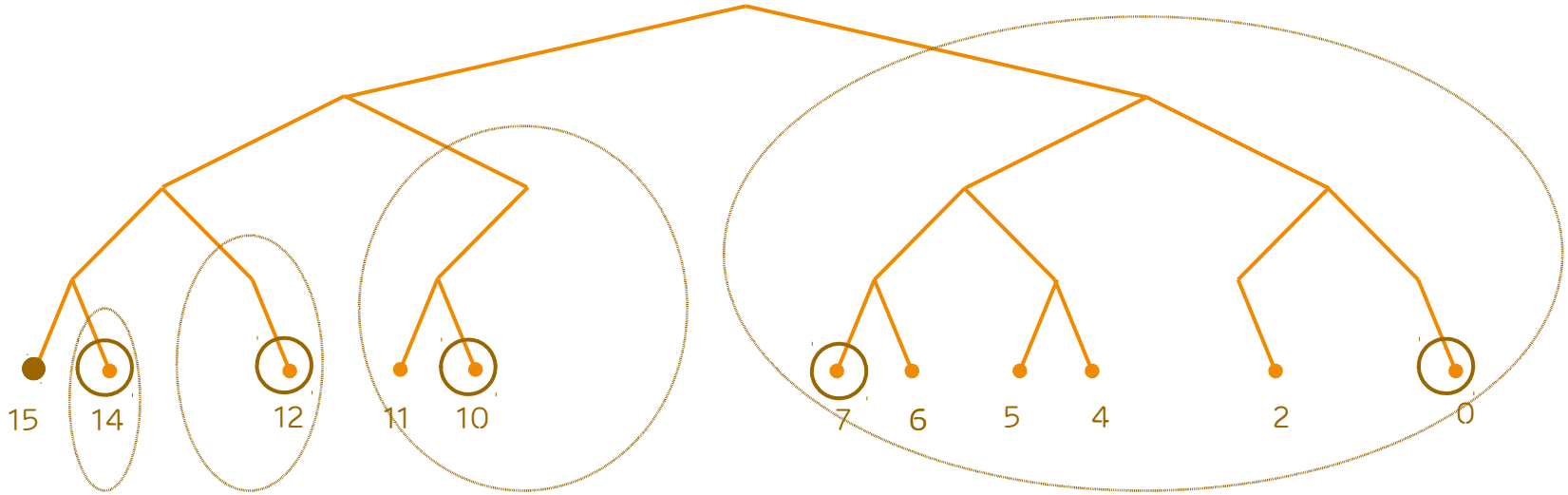
Subtrees
- Common Prefixes
- Distance

Guarantee:
- know at least 1 node per subtree

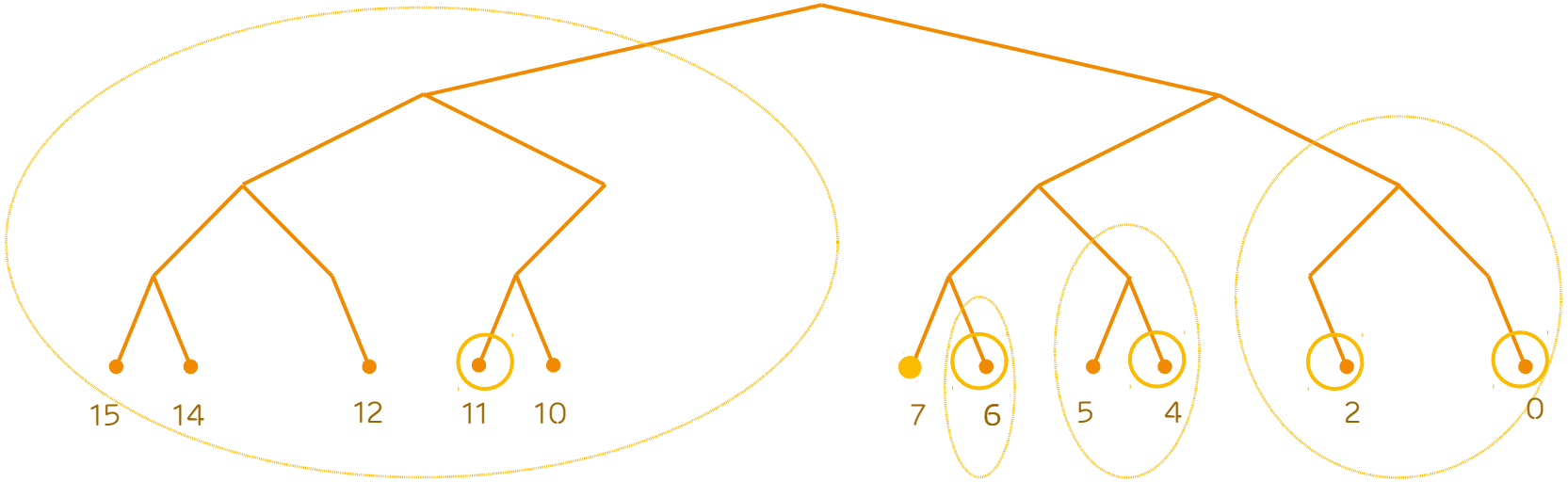
Can know up to K per subtree



KADEMLIA: ROUTING TABLES



KADEMLIA: ROUTING TABLES



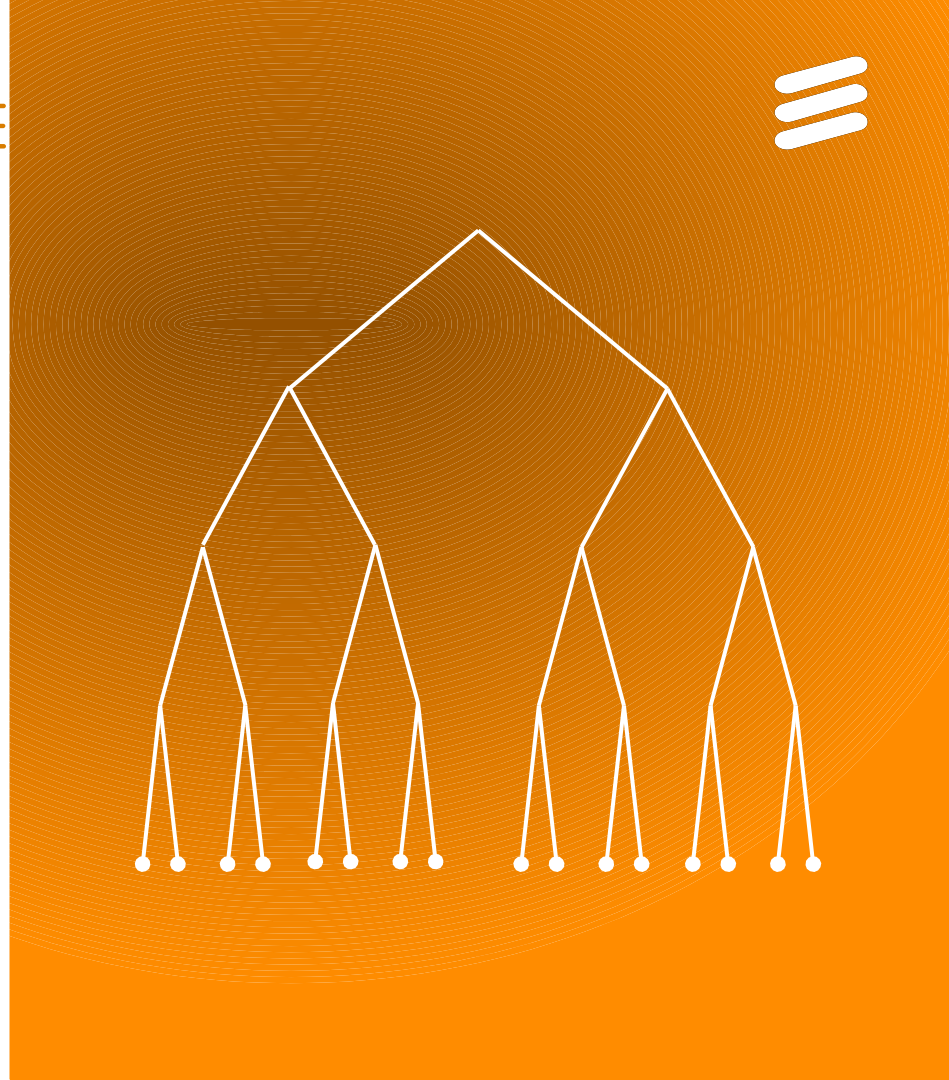
KADEMLIA: ROUTING TABLE

XOR based distance

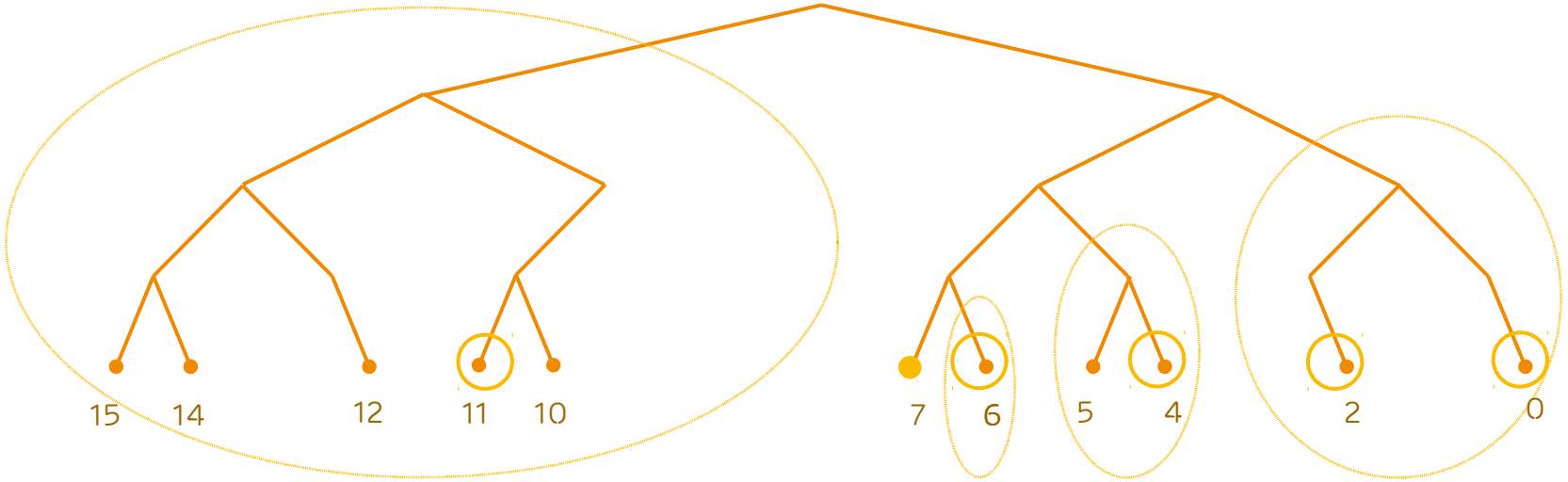
```
A xor B
```

Node A: 0111 (= 7)
Node B: 1011 (= 11)

Distance: 1100
Subtree number: 1000



KADEMLIA: ROUTING TABLES

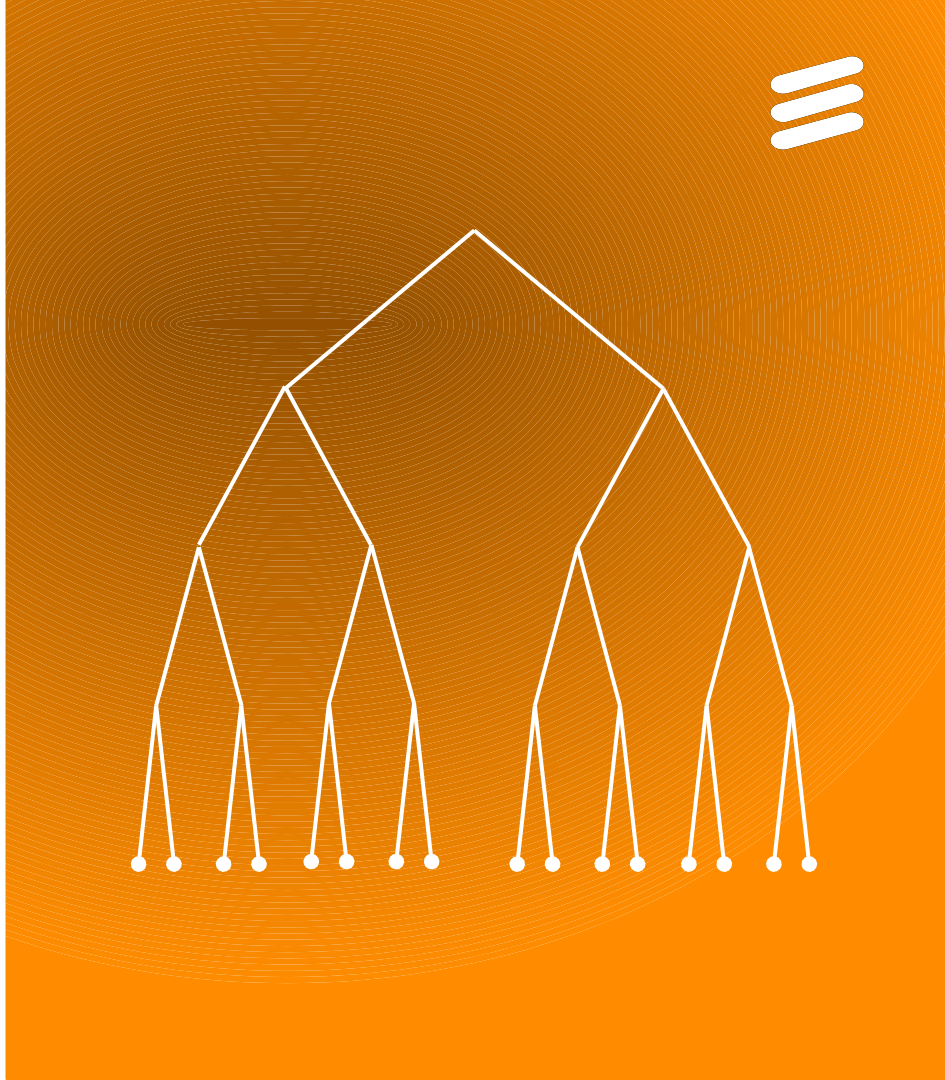


KADEMLIA: NODE LOOKUP

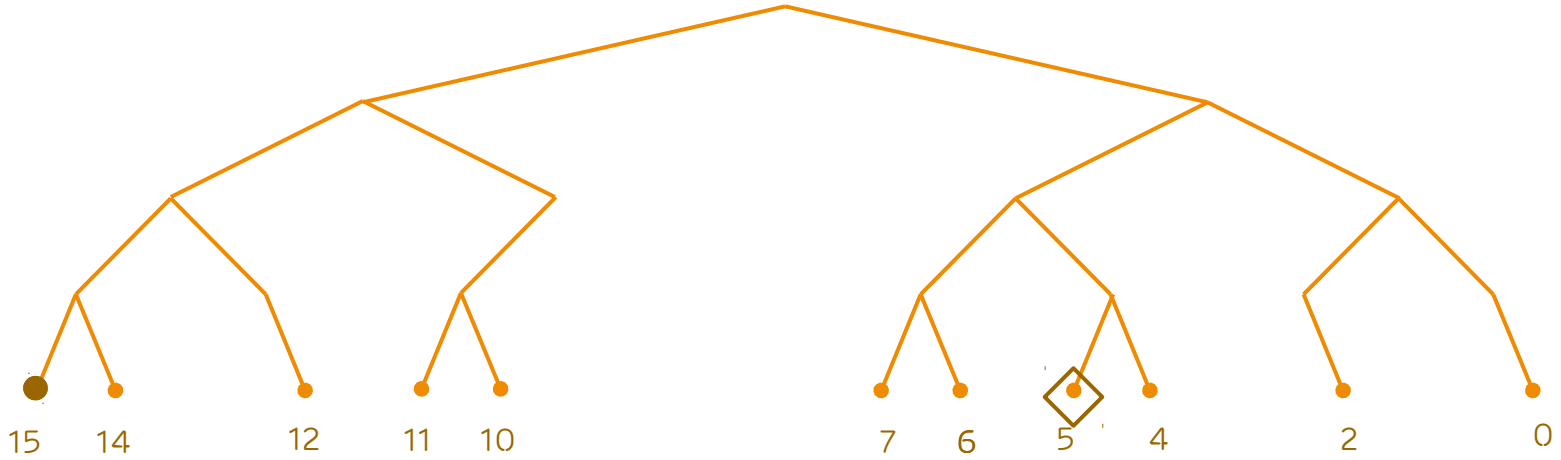
K nodes closest to a key

Storing, lookups and joins

```
:global.register(name, pid)
:global.whereis(name)
:global.send(name, message)
```

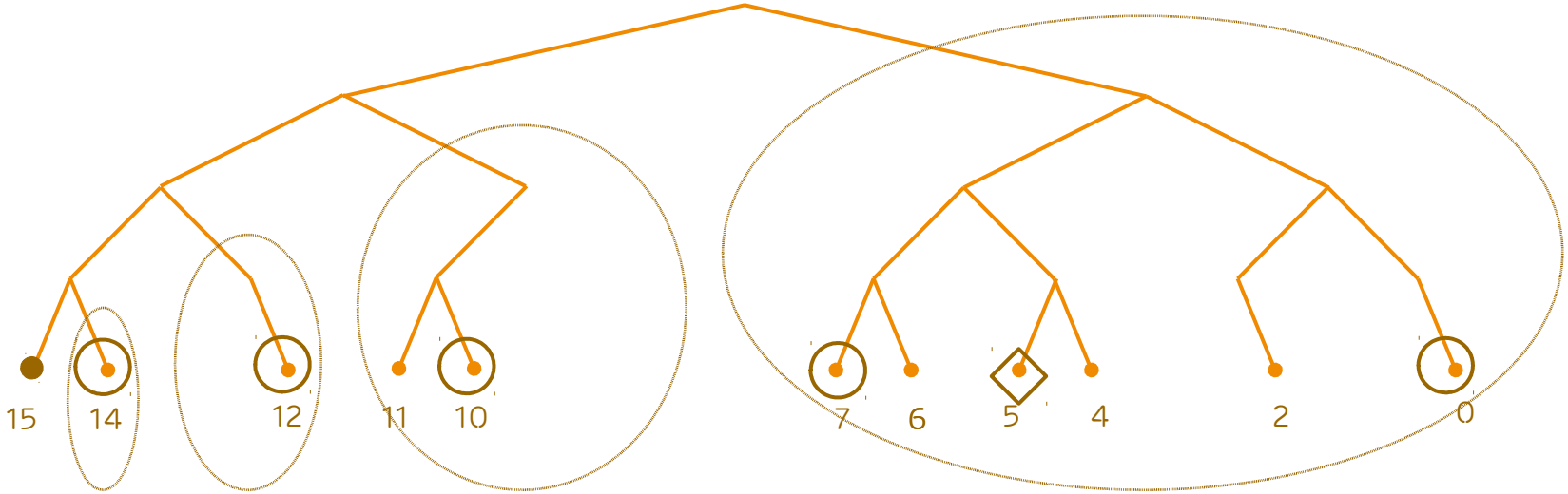


WHEREIS_NAME (WORST CASE, NO REPLICATION)

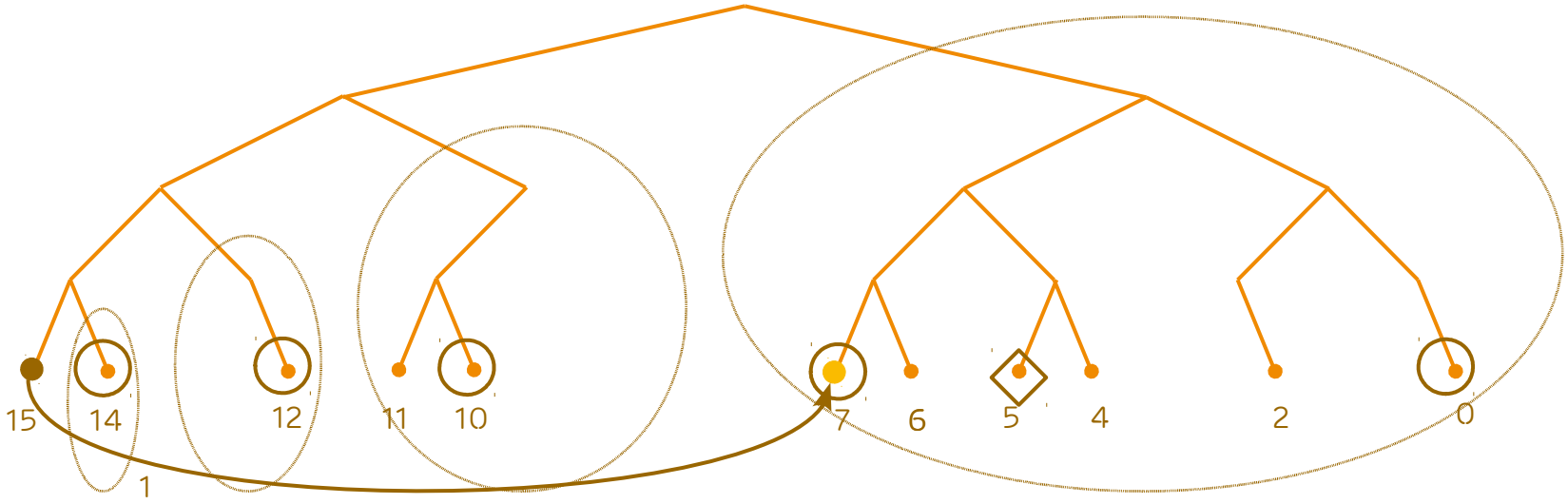



```
:global.whereis_name(:this_name)

hash(:this_name) = 5
```

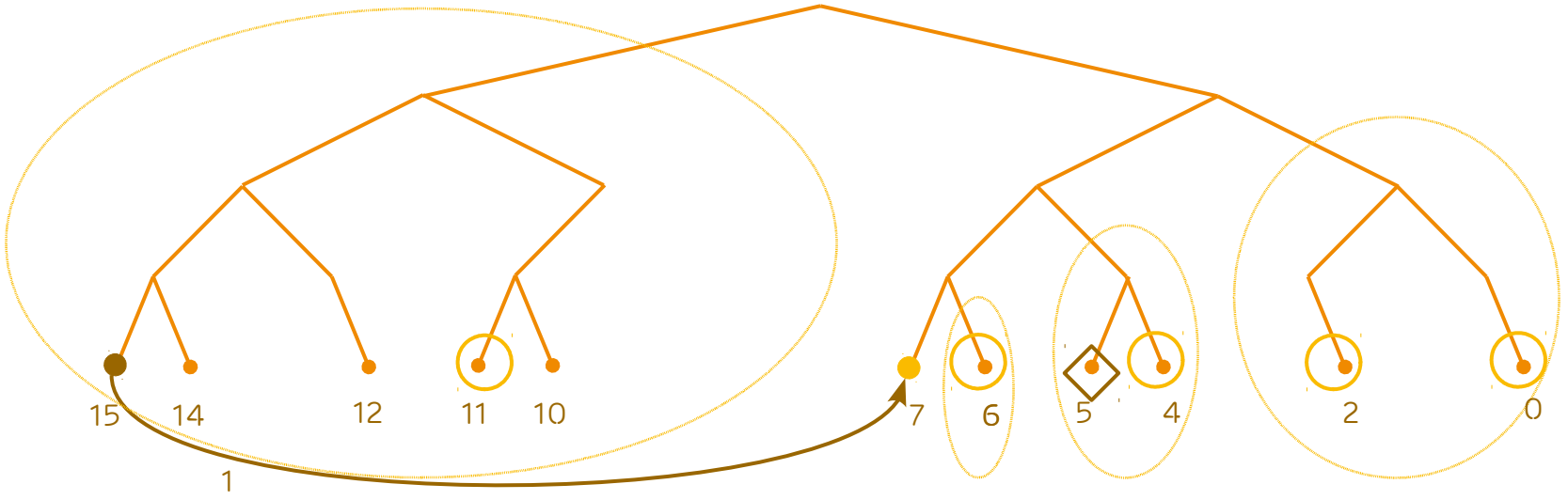
WHEREIS_NAME



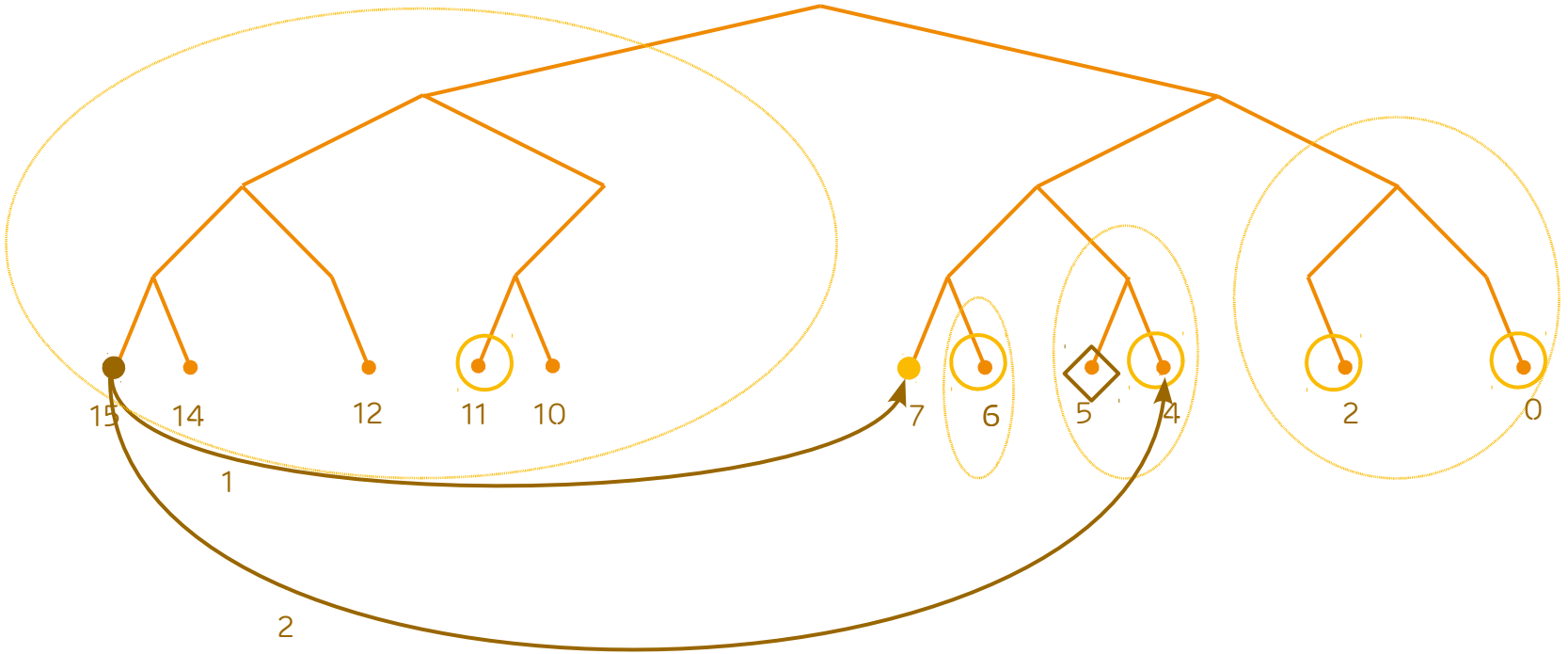
WHEREIS_NAME



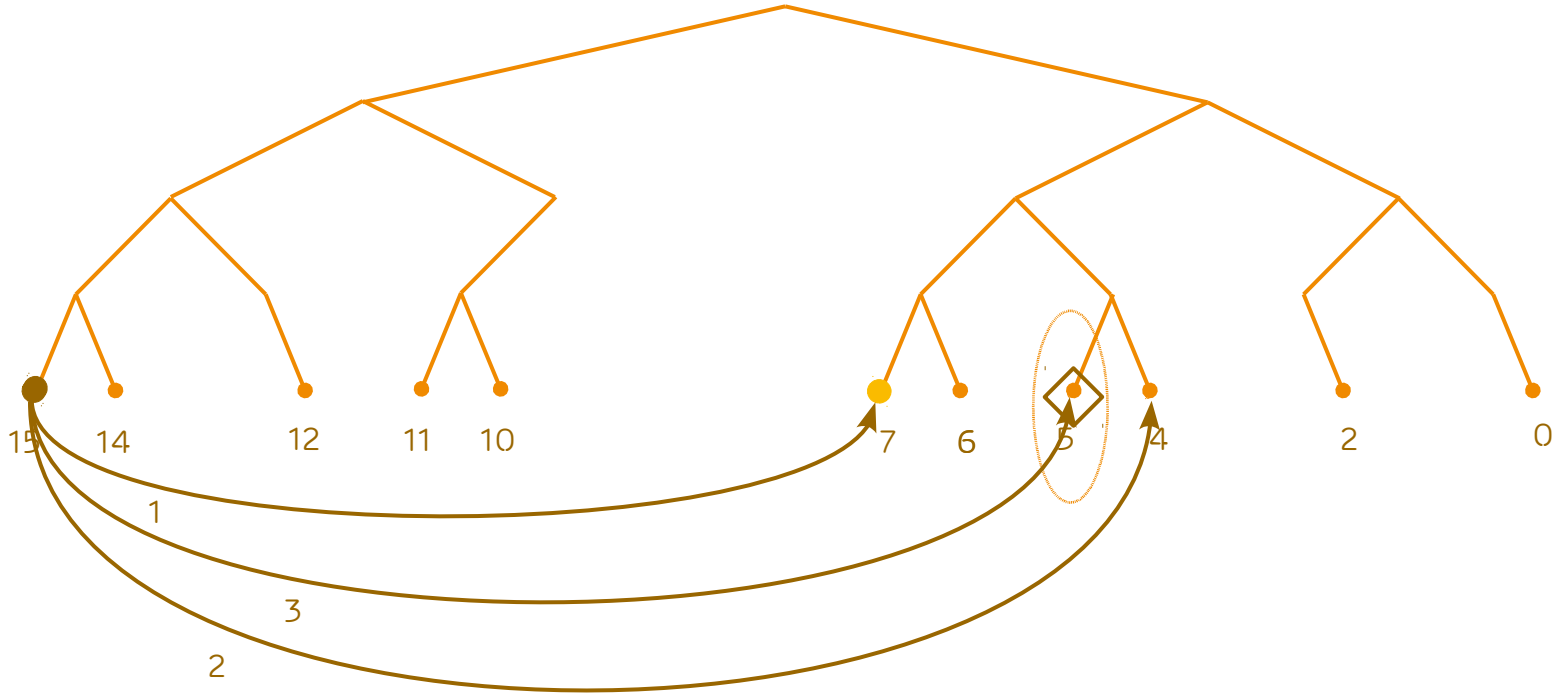
WHEREIS_NAME



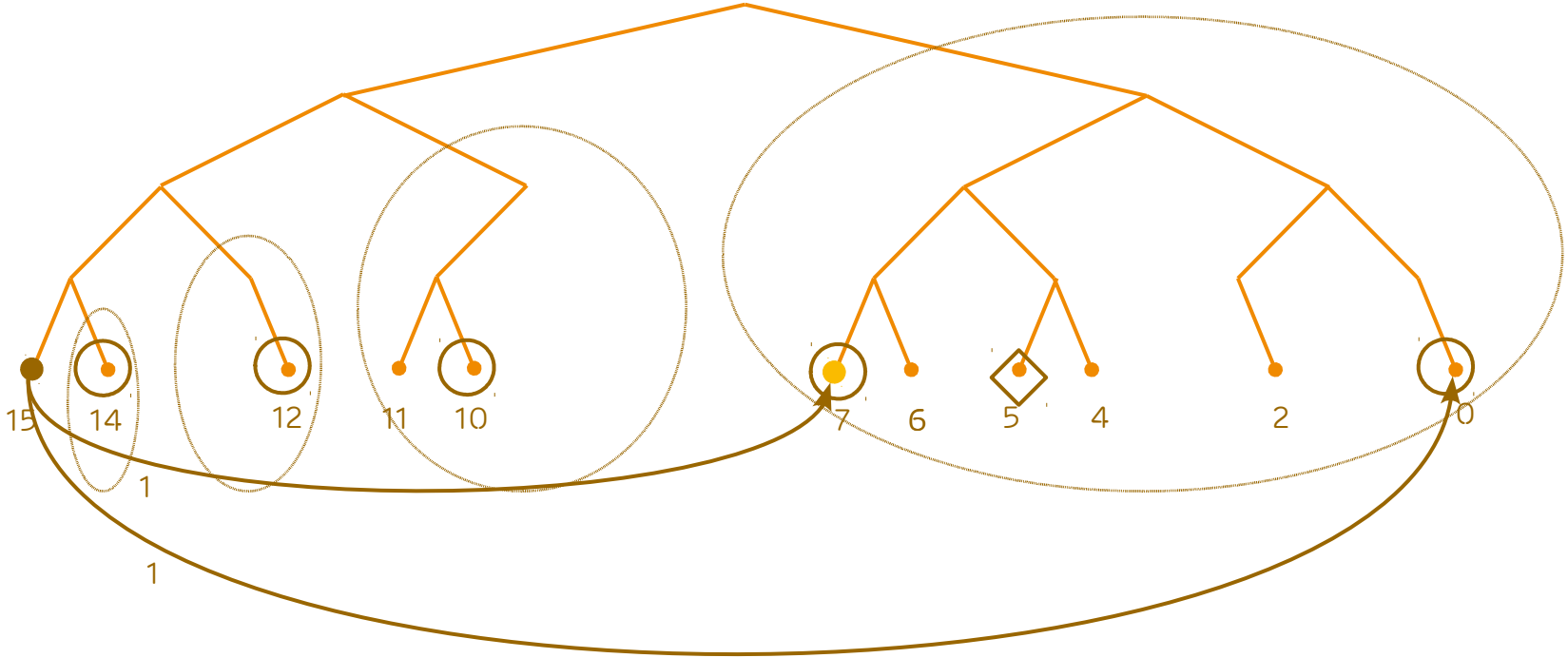
WHEREIS_NAME



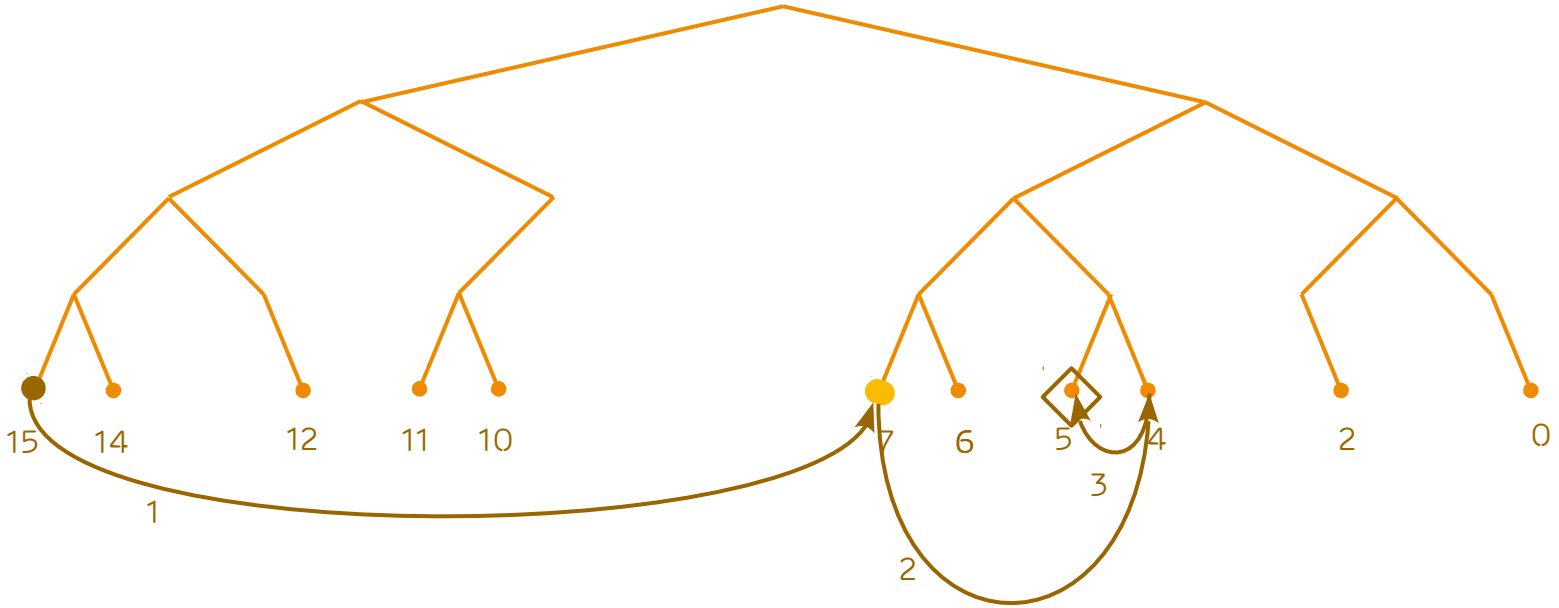
WHEREIS_NAME



PARALLEL LOOKUP



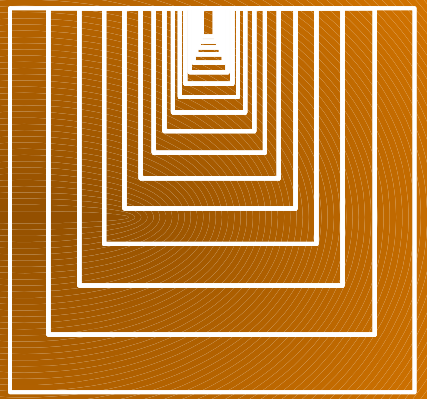
RECURSIVE LOOKUP

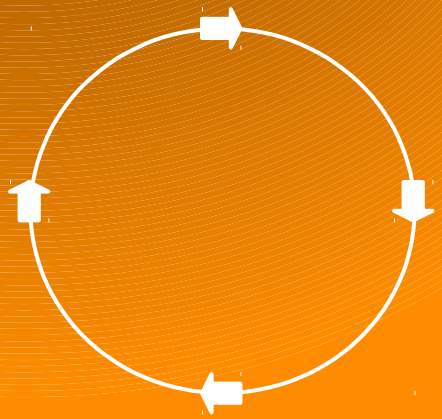


RECURSIVE VS ITERATIVE

recursive
› most connections already up
› faster

iterative
› updating routing tables
› parallel

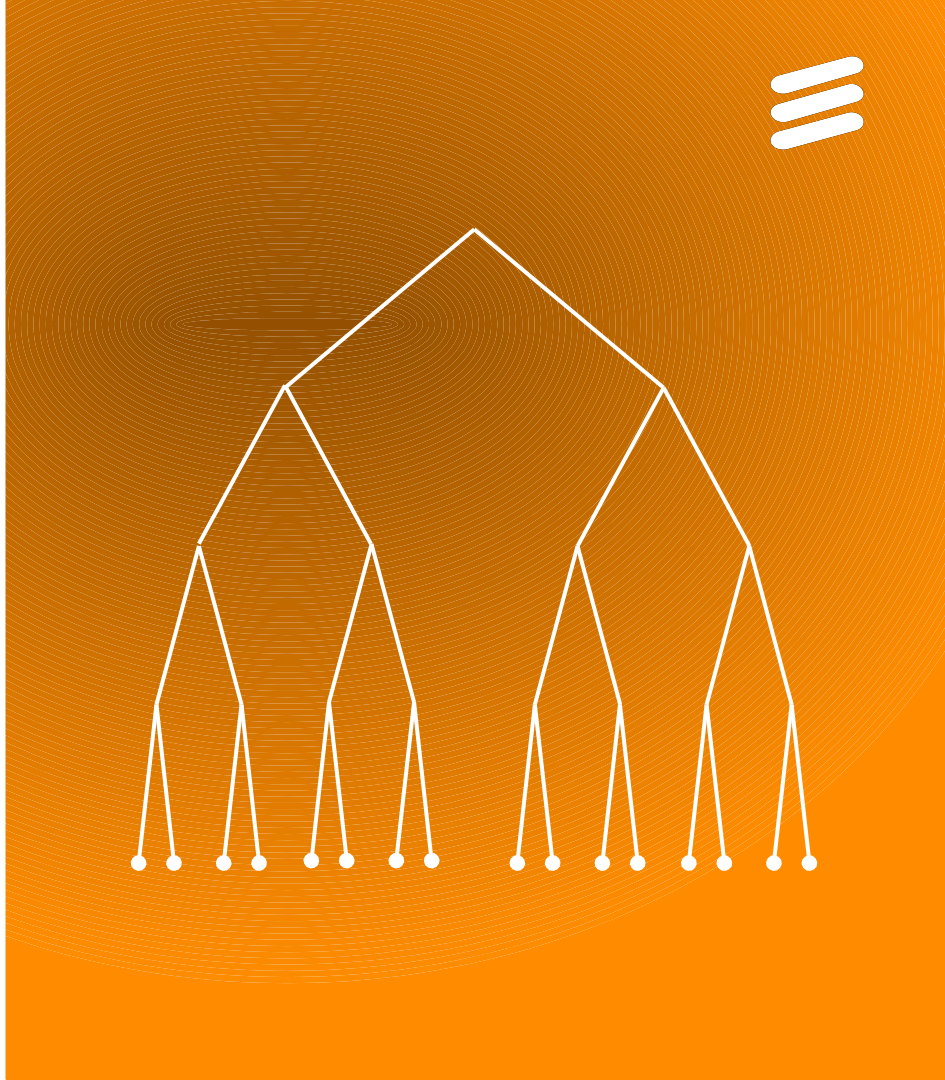




:global

```
register_name(name, pid)

re_register_name(name, pid)

unregister_name(name)

whereis_name(name)

send(name, message)
```

Node joins...

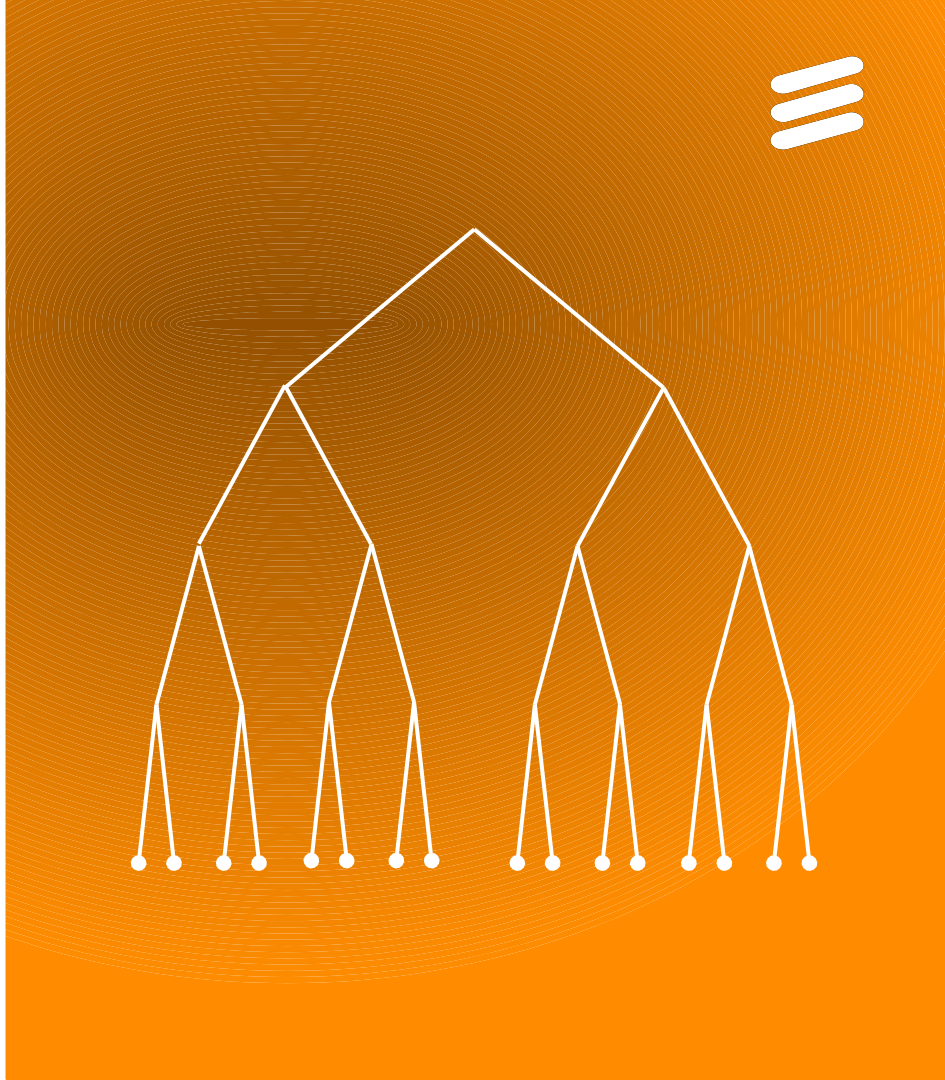


"OWNER" NODE

`unregister_name`

`re_register_name`

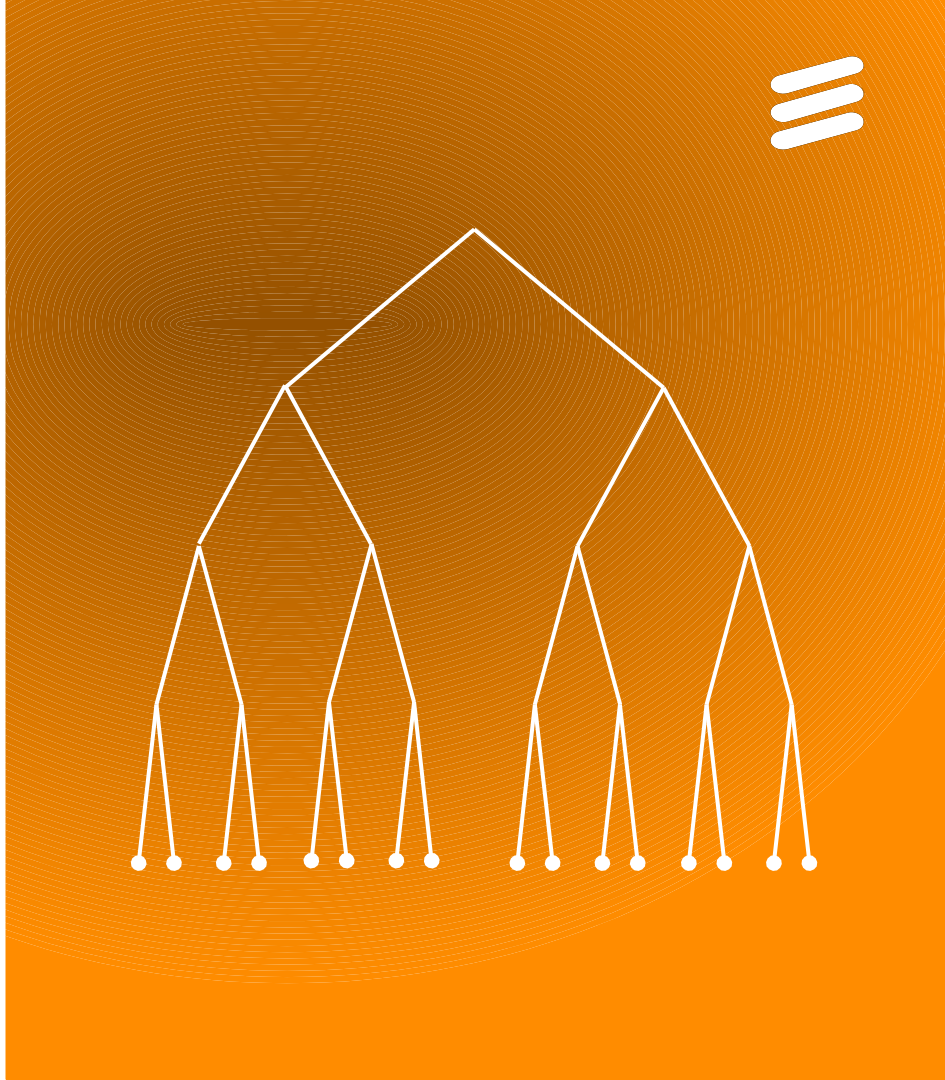
Caching



:global

```
register_name(name, pid)

re_register_name(name, pid)

unregister_name(name)

whereis_name(name)

send(name, message)
```

Node joins...



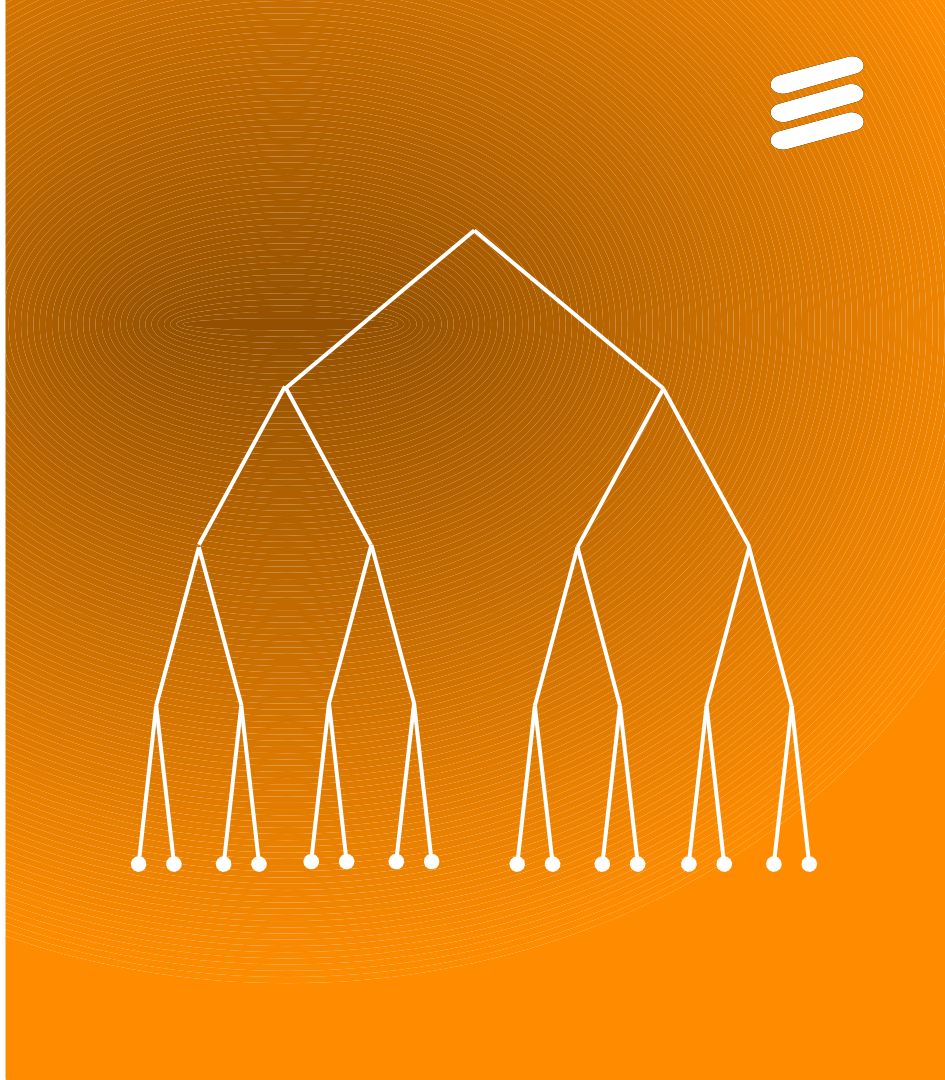
NEW NODE JOINS

1) Has to know one other node

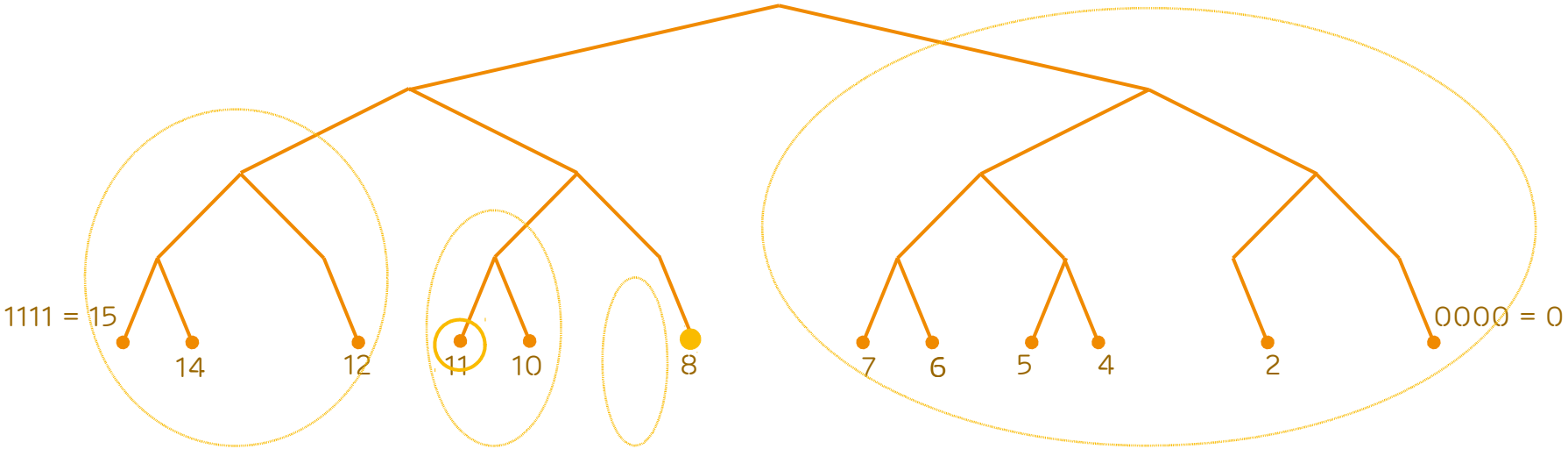
2) Look up the own node ID

3) One node lookup per subtree

Populates its own routing table
Spreads information about itself

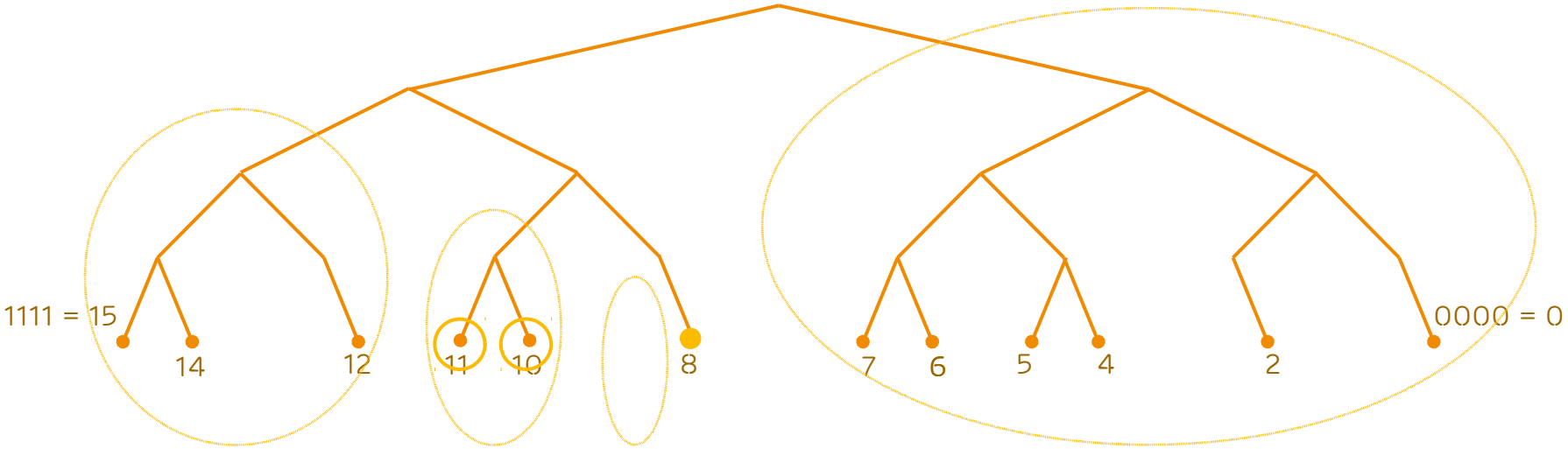


NODE 8 JOINS



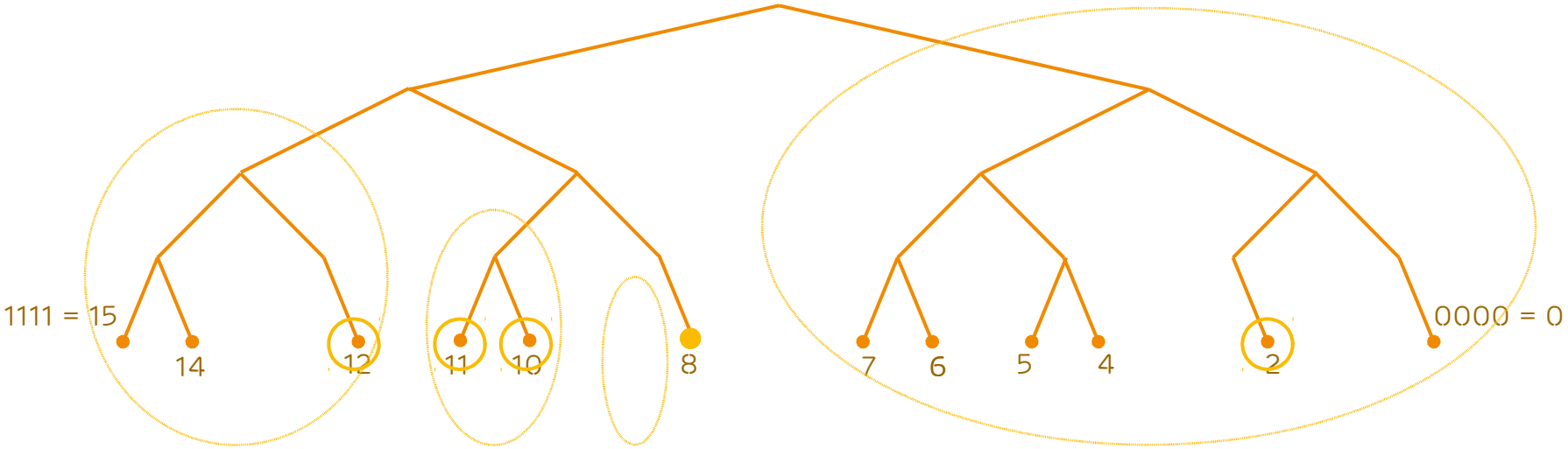
1) KNOW ONE NODE: 11

NODE 8 JOINS



2) LOOKUP 8 (ITSELF)

NEW NODE JOINS



3) ONE LOOKUP PER SUBTREE
HERE: 10, 13 AND 3

AUTOMATIC DISCONNECTS

Avoid too many connections

Inactive connections brought down

Can, should and will be fixed

WILL THIS SCALE THEN?

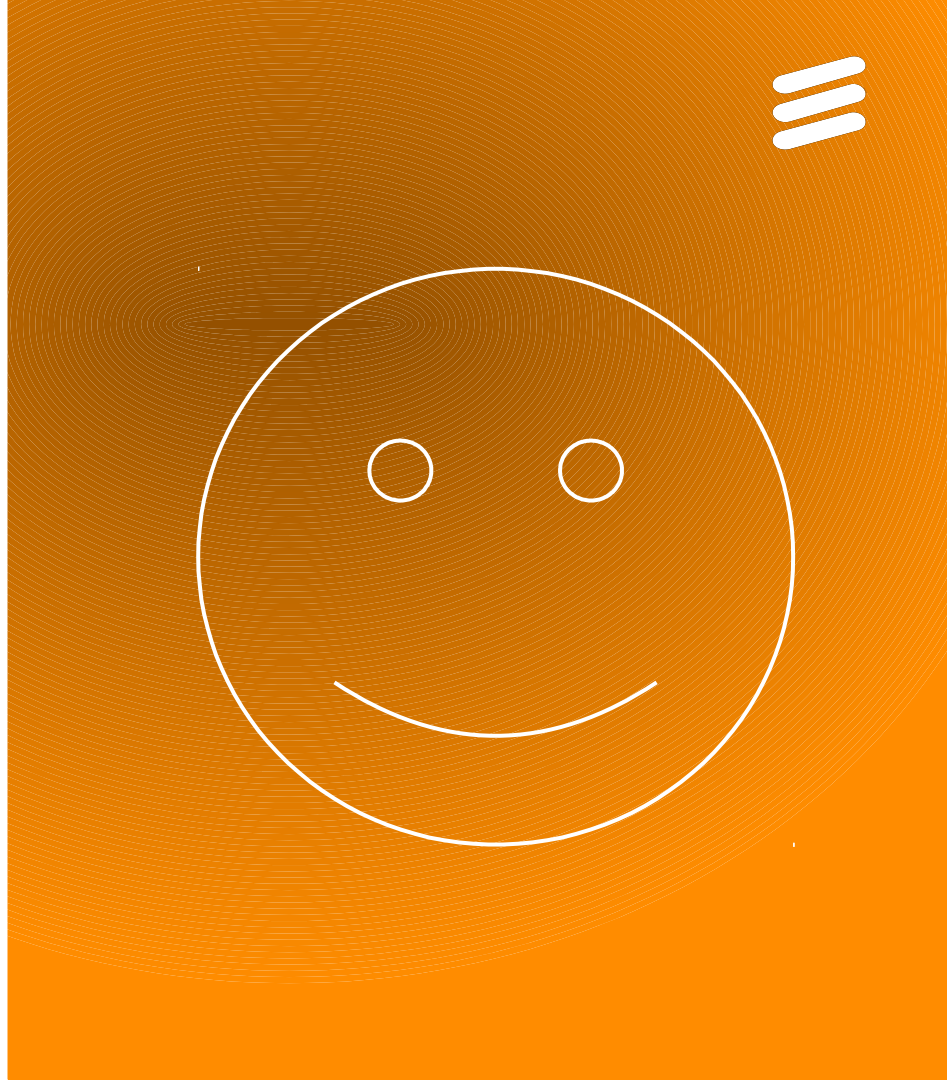
WILL THIS SCALE THEN?

Research: yes

Initial measures: looks promising

Left:

- More measurements
- Optimizations
- A lot more fun stuff!



OTHER CONSIDERATIONS

RPC improvements

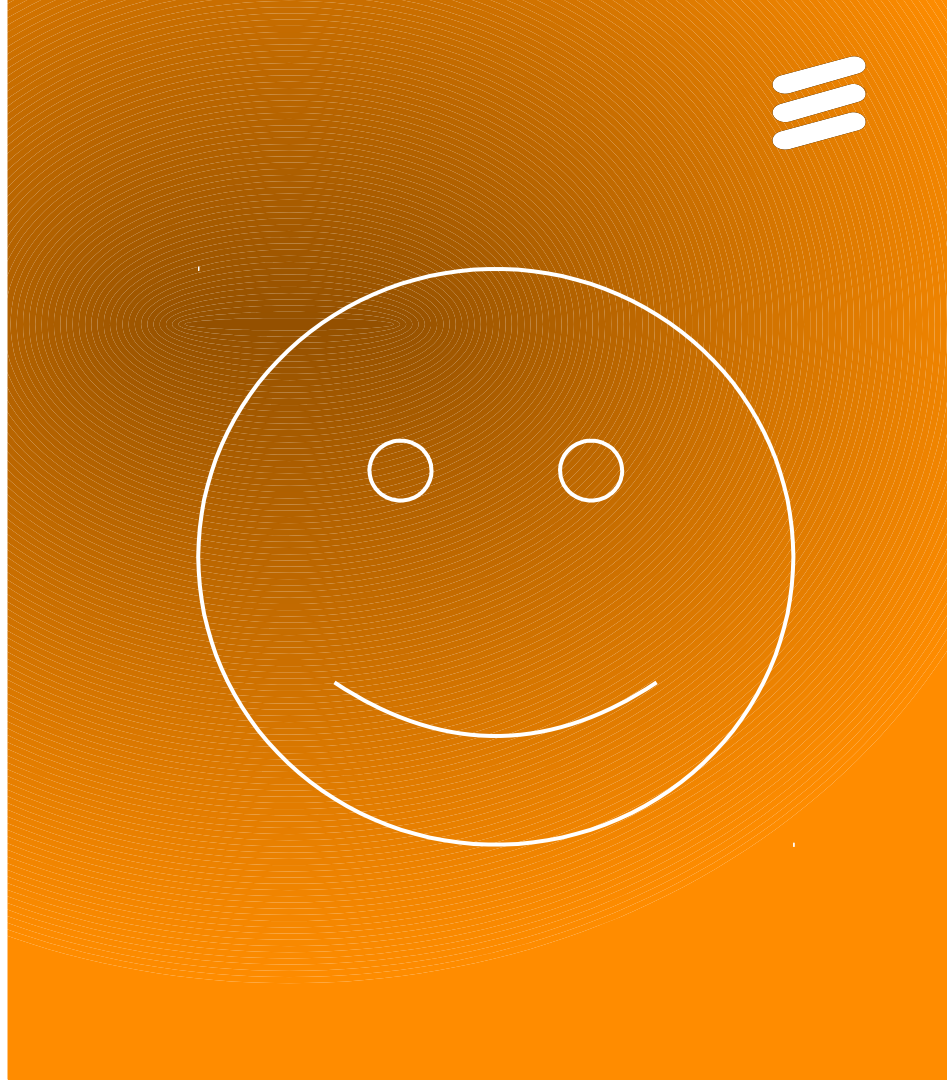
Remote spawn_monitor

EPMD in Erlang

Protocol improvements (TCP/SSL)

Fragment large messages

Preserve Sub-term Sharing



FEEDBACK WELCOME!

Bugs/feature Requests

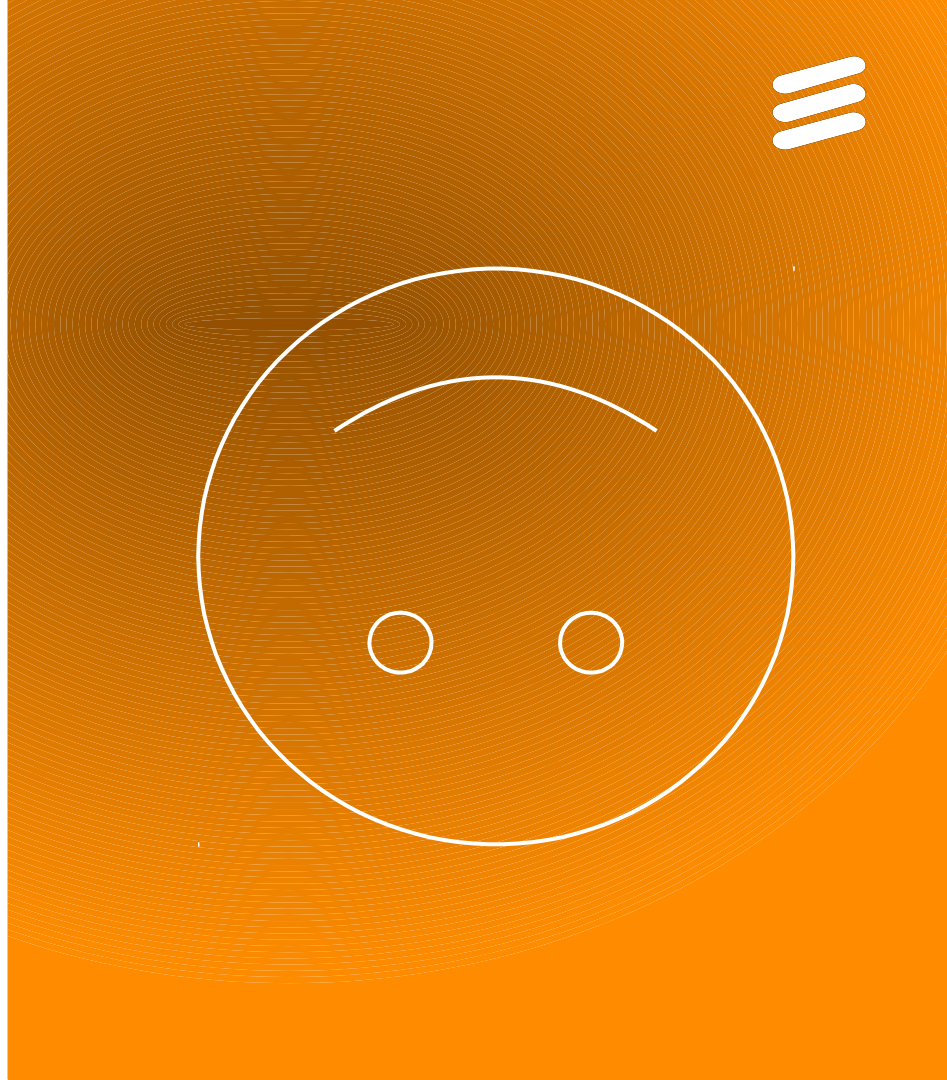
bugs.erlang.org

erlang-questions mailing list

erlang.org/community

Me:

zandra@erlang.org



THANK YOU!