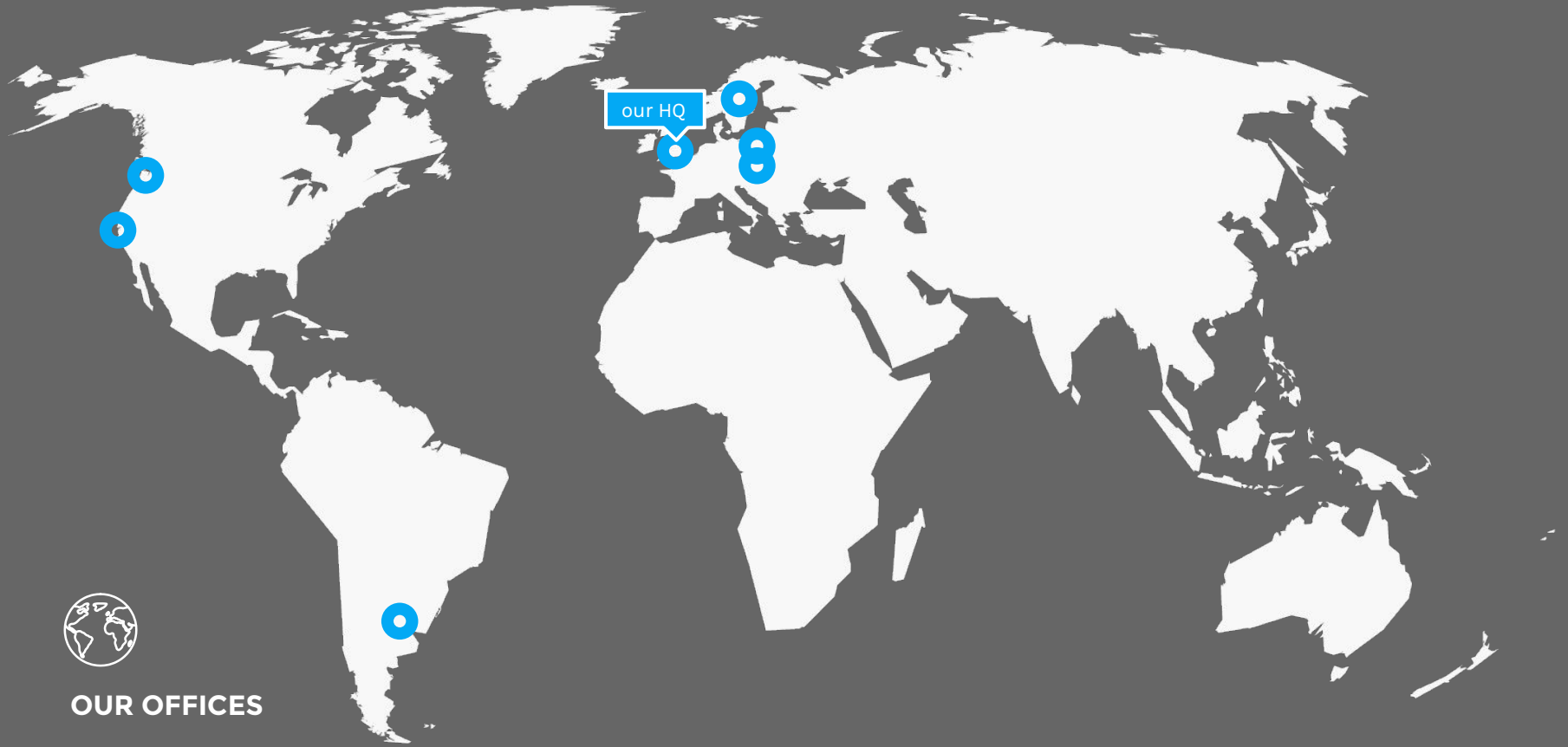
Writing Scalable and Fault-Tolerant Systems in Erlang

Why does one of the most popular NoSQL databases use Erlang?

Csaba Hoch

csaba.hoch@erlang-solutions.com
www.erlang-solutions.com

My background



OUR OFFICES

Contents

1.

Erlang

Agner Krarup Erlang (1878 – 1929)

History

- Makes it easier to build scalable, fault-tolerant systems
- Prolog roots
- 1998: open sourced

Erlang's basic features

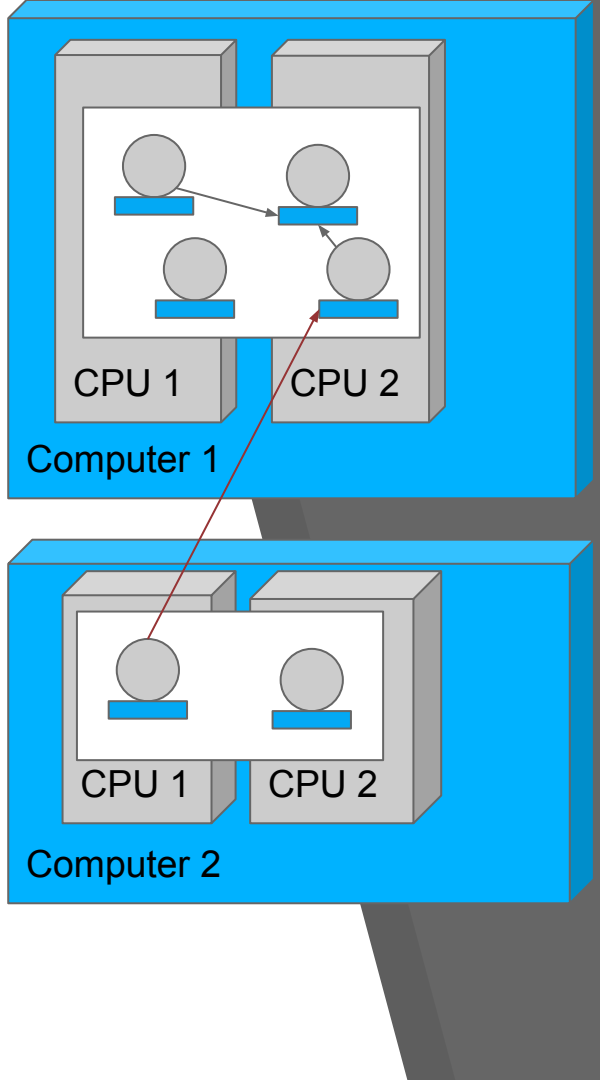
- Functional
- Immutable data structures (terms, e.g. `[a, {b, 1}]`)
- Dynamic typing
- Garbage collection

```
% hello.erl
-module(hello).
-export([hello_world/0]).

hello_world() ->
    io:fwrite("hello, world\n").
```

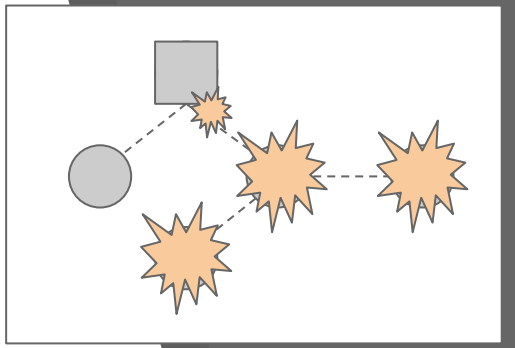
Erlang's key features

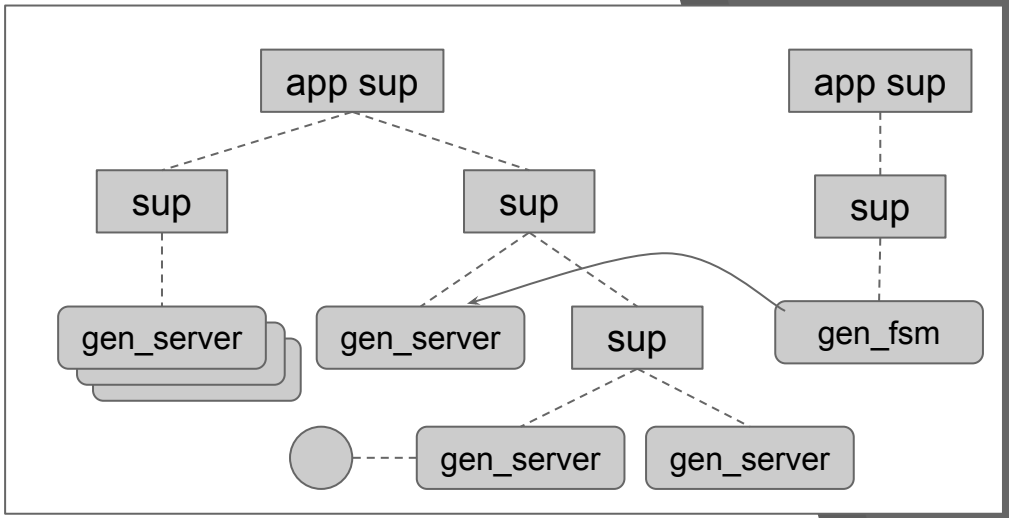
- Concurrency via **processes** and the **actor model**
- Parallelism via **multi-core** (processes)
- Parallelism via **distribution** (nodes)
- **Hot code loading**
- **Tracing**



Fault tolerance

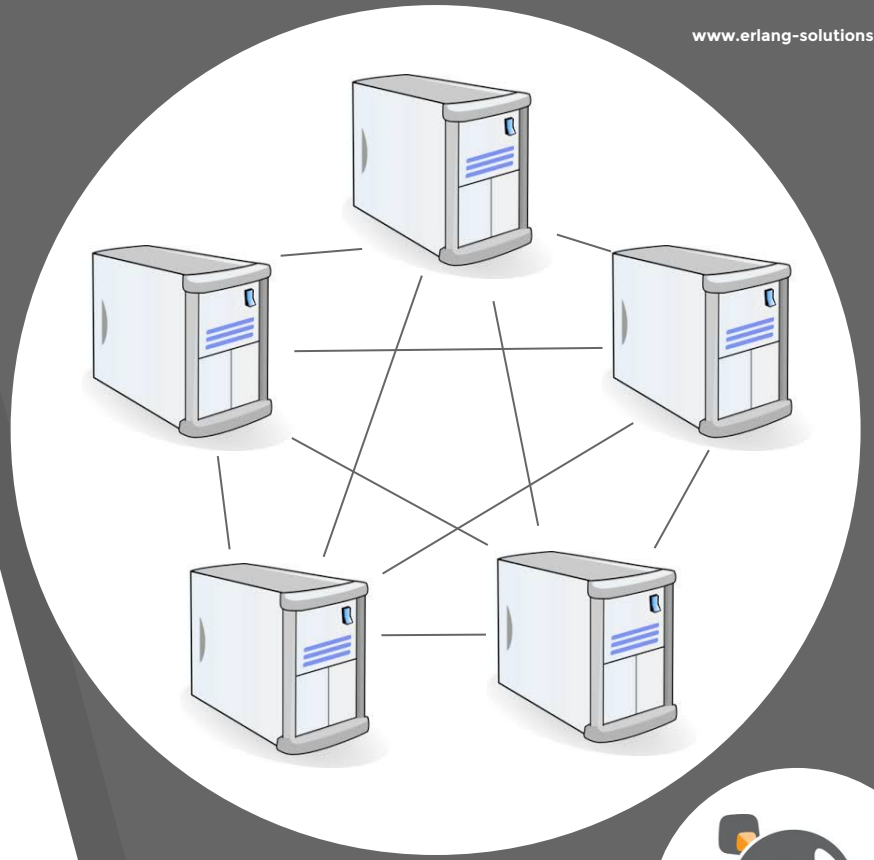
- **Links**
- **Templates** (OTP behaviors): supervisor, gen_server, etc.
- **Distribution**





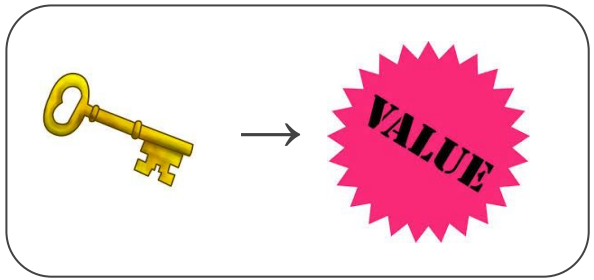
2.

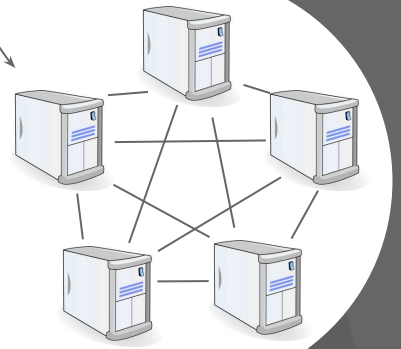
Riak



basho

Dynamo/Riak = a distributed KV-store



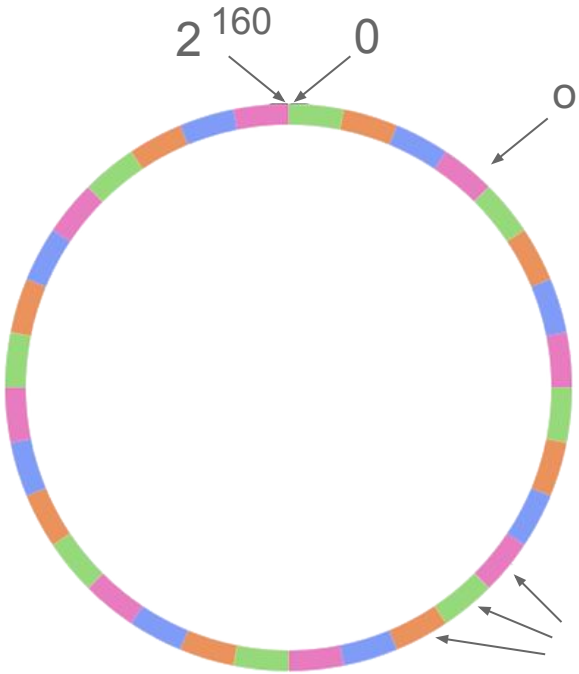


"Shopping cart/hcs42" → ["book_12345", "dvd_765"]

Dynamo/Riak properties

- Highly available
  - Distributed
  - Fault-tolerant
  - Masterless
- Scales easily

The ring



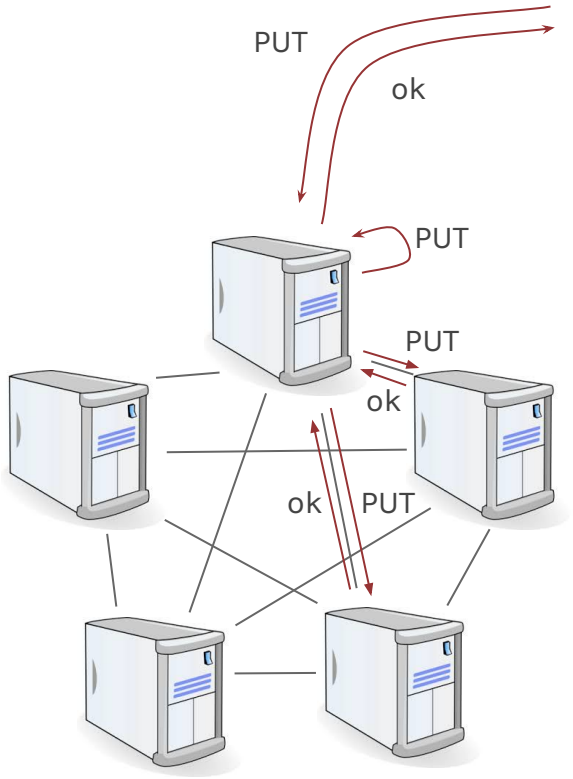
Node 1

Node 2

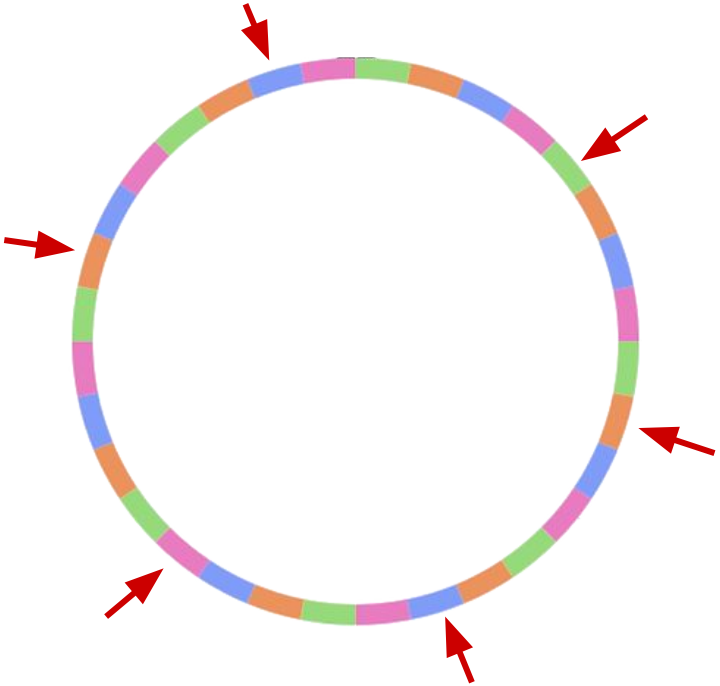
Node 3

Node 4

"Shopping cart/hcs42" → ["book_12345", "dvd_765"]



Scaling easily



Node 1

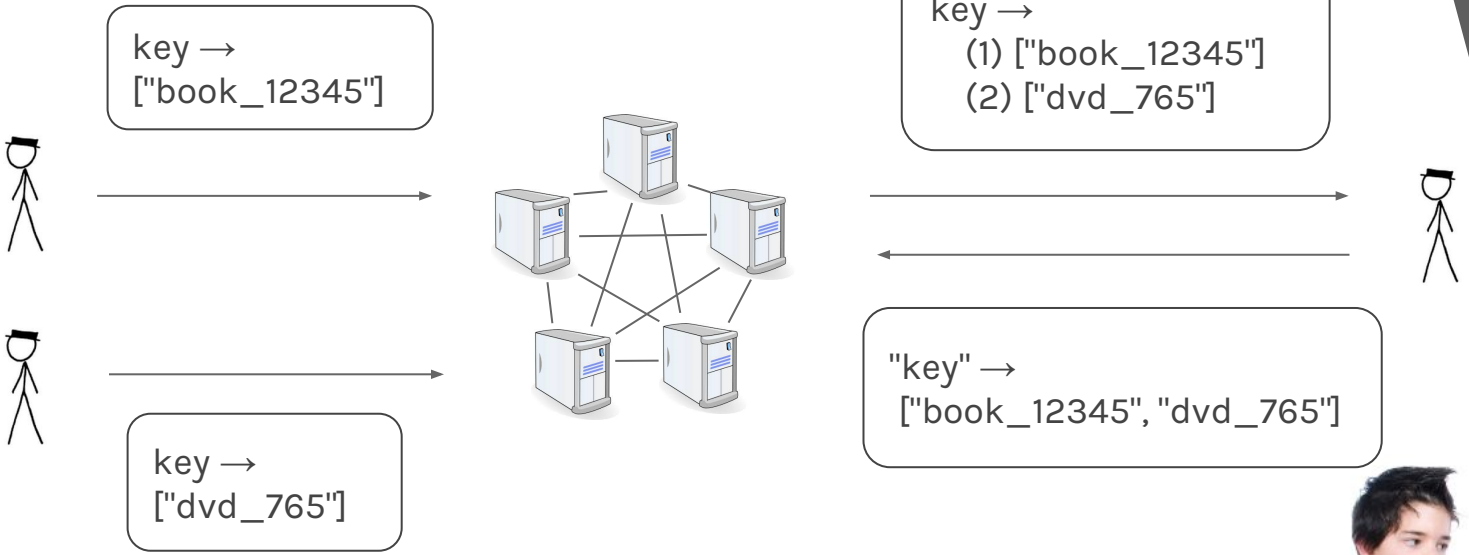
Node 2

Node 3

Node 4

Node 5

Siblings and conflicts



3.

Erlang & Riak

Implementation: help from Erlang

- Concurrency/Parallelism within a node → Erlang **processes**
- Multiple nodes, communication between processes in them → Erlang **distribution**
- Fault tolerance → **supervisor trees**
- Overload protection: What to do if you cannot handle more input? → Process **mailboxes**

Operations & support: help from Erlang

- Investigating errors → **tracing**
- Fixing errors → **hot code loading**

Conclusion

Fault tolerance: Algorithms + Tools

THANK YOU!

Any questions?

csaba.hoch@erlang-solutions.com
www.erlang-solutions.com

Why is FP good for Erlang?

- Possibility of **not copying data** between processes
- Easier to **implement** the VM (GC)
- Easier to **upgrade code**

Where is Erlang used?

- Messaging, Databases, Telecom, Payments, Gaming, ...
- Common theme:
  - many small messages → concurrent processes
  - large systems → parallel nodes