Winning as a Start-up by Failing Fast

Torben Hoffmann
Chief Architect @ Basho
thoffmann@basho.com
@LeHoff

90% of all start-ups fail

source: http://www.forbes.com/sites/neilpatel/2015/01/16/90-of-startups-will-fail-heres-what-you-need-to-know-about-the-10/#2f77631555e1

The other 10%

Characteristics:

The product is perfect for the market

The entrepreneur does not ignore anything

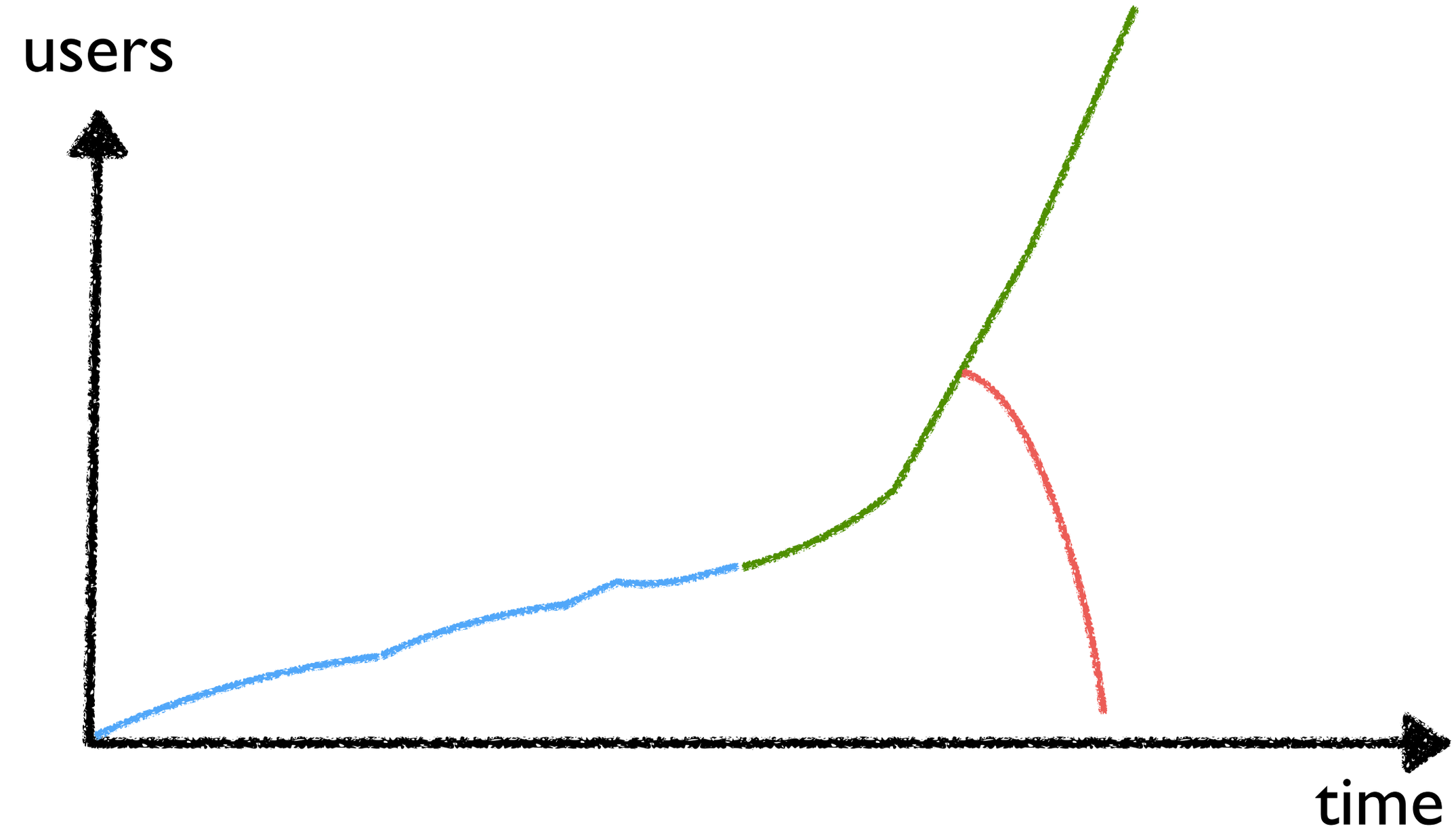
The company grows fast

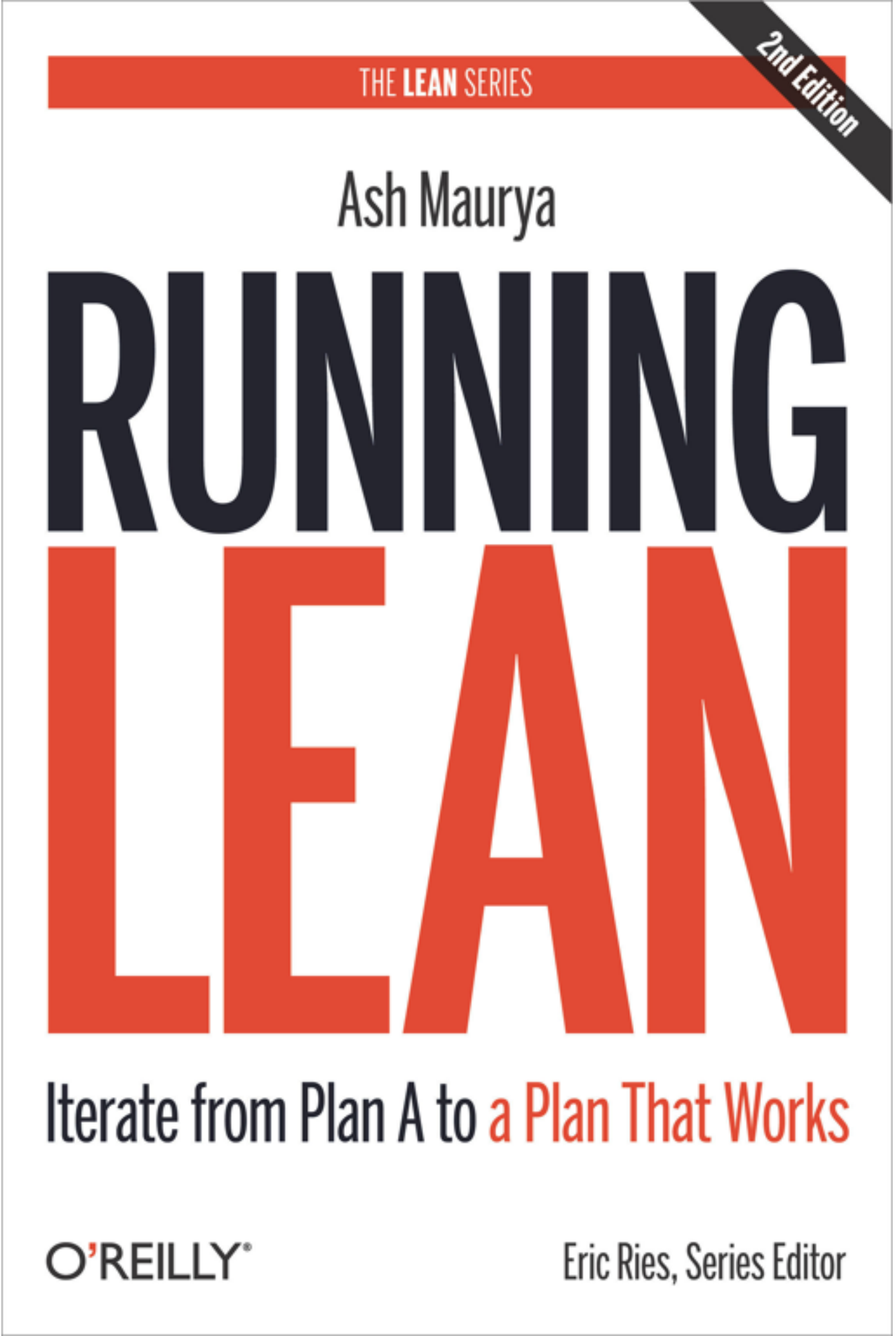
The team knows how to recover

The Message

Evolving your start-up through **failures!!!**

Building your system with


users
time



Experiments

Hypothesis

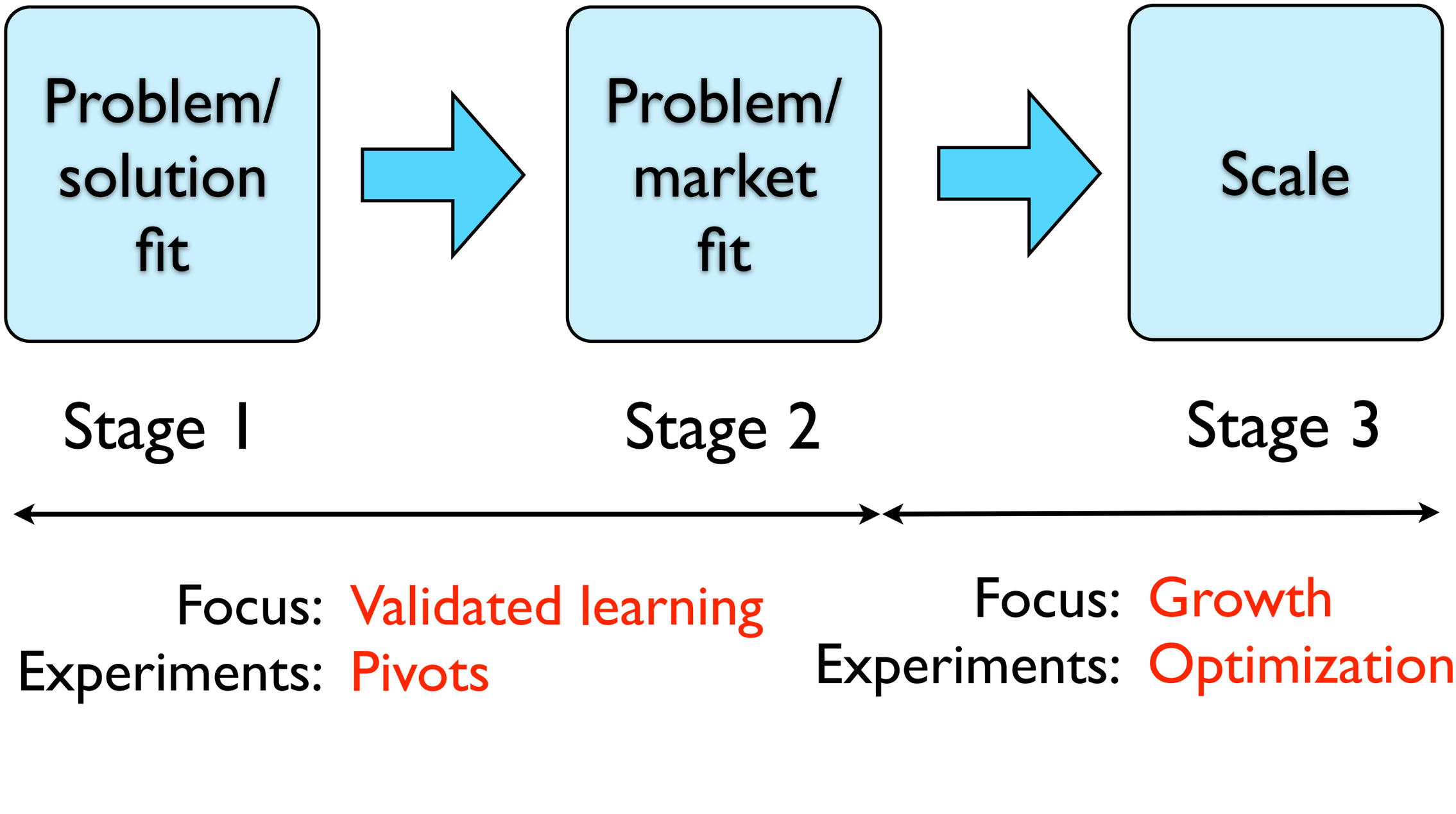
Pivot

What are they asking you to do?

Fail Fast!

Embrace failure!!

From zero to $$$


Problem/ solution fit
Problem/ market fit
Scale
Stage 1
Stage 2
Stage 3
Focus: Validated learning
Experiments: Pivots
Focus: Growth
Experiments: Optimizations

Development Speed

Many stacks have this

Not our primary concern

Experiments

Try new things

Keep the system running

Supervisors!

Pivot

Must be easy to tweak your code

and re-use parts of it

The Golden Trinity of Erlang

I have not failed. I've just found 10,000 ways that won't work.”

Thomas A. Edison

All failures are created equal, but some are more equal than others...

Good Failures

Leads to insights about your business

A step towards a business model that works

Bad Failures

Downtime due to bad quality software

Tech choices that limits exploitation of business failures

Silver Bullet

Fred Brooks differentiates between:

Essential complexity

Accidental complexity

Changeability

Software spends 80% of its life in maintenance

…so you'd better write it so it is easy to change!

1st data model is wrong

2nd data model is wrong

by induction…

all data models

are

wrong

NoSQL

Flexibility ⇒ velocity drops slower

Don’t use sche

your middlena

and your lastname is von Upgrade

Spend your time on features, not schemas!

RiakKV on one node?

You can do it, but not optimal :-(

Benefits on availability when you scale

Either way - the DAL gives you freedom

NoSQL with Mnesia

Simple record:

```
-record(entry,
        { key,
          value }).
```

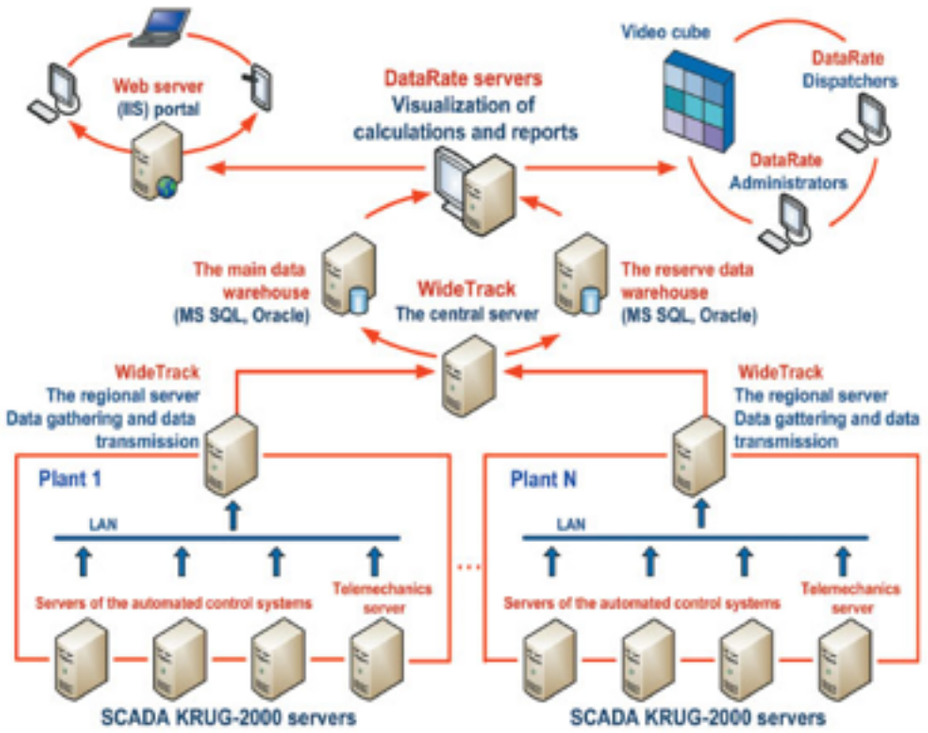
`value` is of whatever type you feel like

Share Nothing

Only one thing breaks.

Period!

Failures



Programming errors
Disk failures
Network failures

Anything that can go wrong, will go wrong.

Murphy

Whatever can happen will happen.

De Morgan

source: http://www.krug-soft.com/297.html

Fault In-Tolerance

Most programming paradigmes are
fault in-tolerant
 ⇒ must deal with all errors or die

source: http://www.thelmagazine.com/BrooklynAbridged/archives/2013/05/14/should-we-be-worried-about-this-brooklyn-measles-outbreak

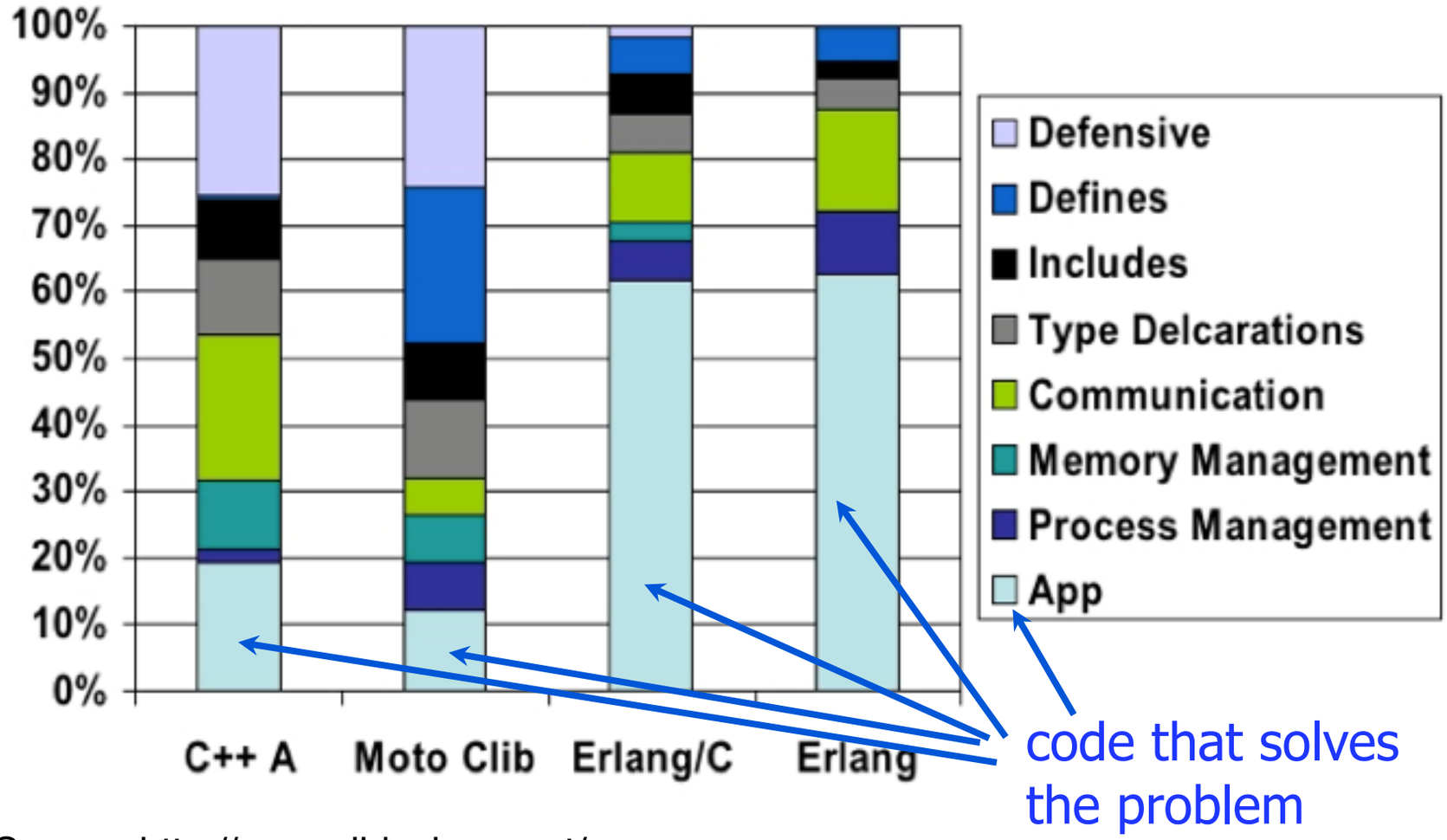
Fault Tolerance

Erlang/Elixir is fault tolerant by design

⇒ failures are embraced and managed

Benefits of fail-fast & supervision

Data Mobility component breakdown



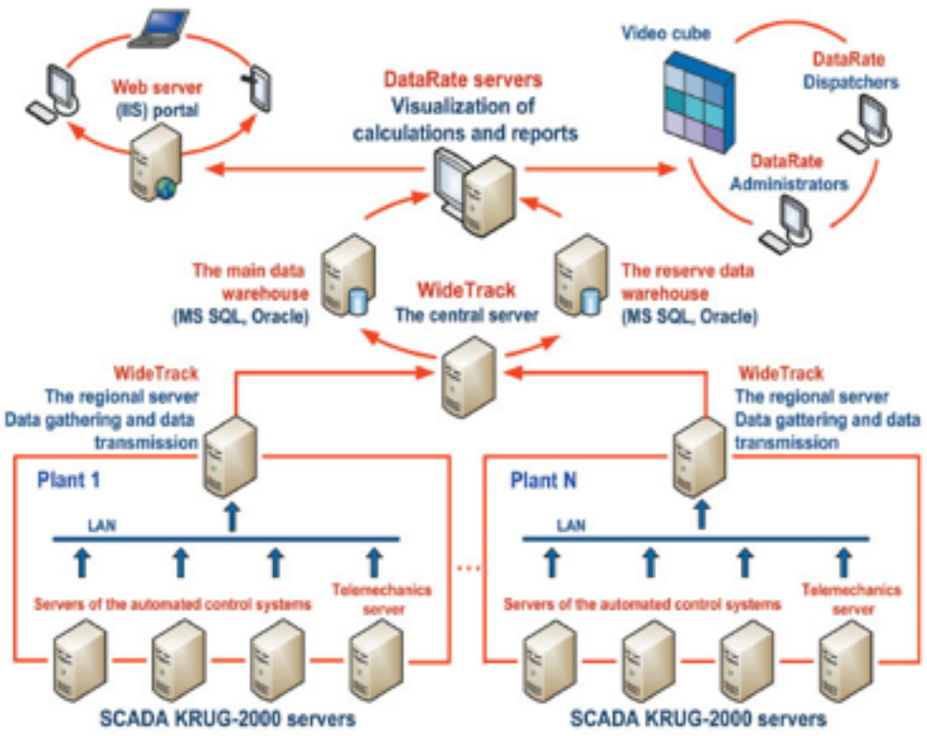
Source: http://www.slideshare.net/JanHenryNystrom/productivity-gains-in-erlang

Erlang @ 3x

Embrace failure!!

If you can wrap failure you might have a product

Riak's Sweetspot



Disk failures

Network failures

Riak's Solution

Replicate data to counter failing nodes

Detect failing nodes and provide substitutes

Handoff - return data to rightful owner

AAE - repair after failures

Realities of software development

Source: http://www.thejournal.ie/readme/lunch-atop-skyscraper-photo-men-irish-shanaglish-518110-Jul2012/

Business benefits of supervisors

Only one process dies

isolation gives continuous service

Corner cases can be fixed at leisure

Product owner in charge!

Not the software!

Software architecture that supports iterative development

This is not a warning…

this is a threat!

Clean APIs

One module to front an app

Send messages through a module API

I have the pid, can't I just bang it a message?

No, please don't!!!

I know it is a GenServer, can't I just GenServer.call it?

How do you want to die?

It's your own fault. What did @lehoff tell you about sending messages without using a module API?

Source: https://www.google.co.uk/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwj14bL92rDMAhWIoD4KHd9ADVQQjRwIBw&url=https%3A%2F%2Fwww.cartoonstock.com%2Fdirectory%2Fm%2Fmedieval_family.asp&bvm=bv.120853415,d.dmo&psig=AFQjCNGBYIYUGnOti-7ULAPsG_rj8hrc7g&ust=1461911491500319

Protocols

Protocol

=

How to solve a problem together

Ostrich Development

You can document your protocols or not, they are still there!!

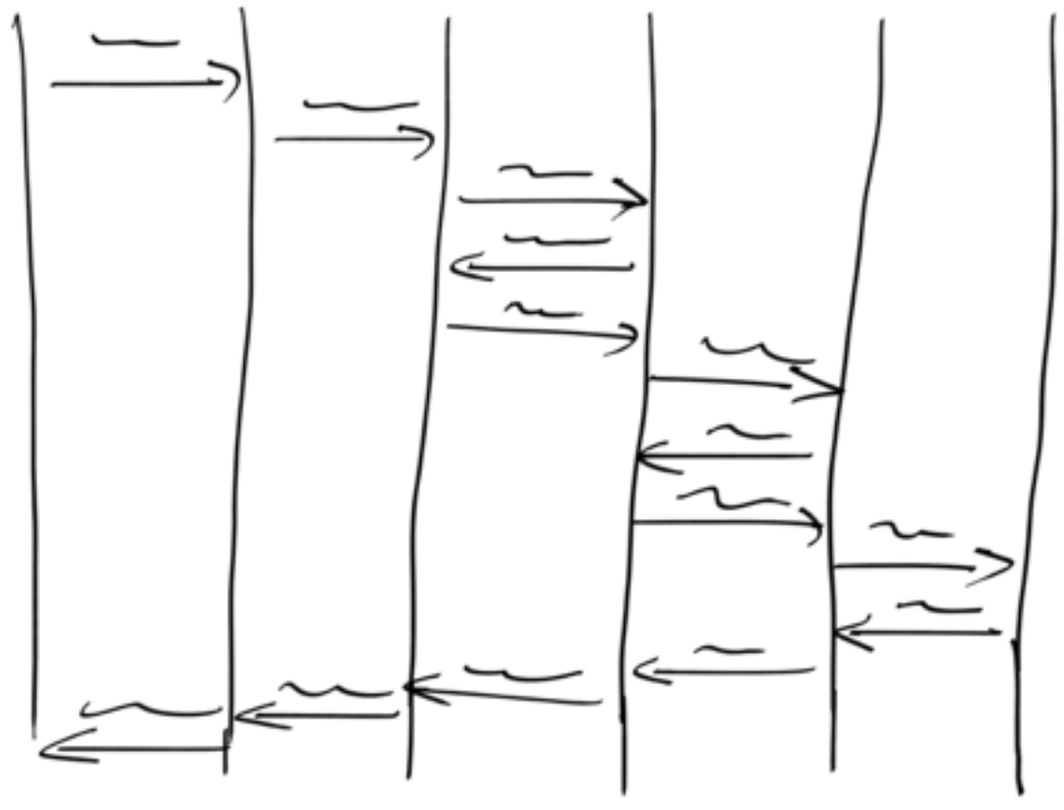
Embrace them in design and stay in control

Cheaper to change a diagram than code…

source: http://www.joy-of-cartoon-pictures.com/images/ostrich-cartoon.jpg

Interaction Diagram

Message Sequence Chart



Limit the Logic

Let each process deal with its own stuff

Leave complicated coordination to others

ACQUIRE

THE GAME OF CORPORATE ACQUISITIONS

ACQUIRE

HYDRA

FUSION

AMERICA

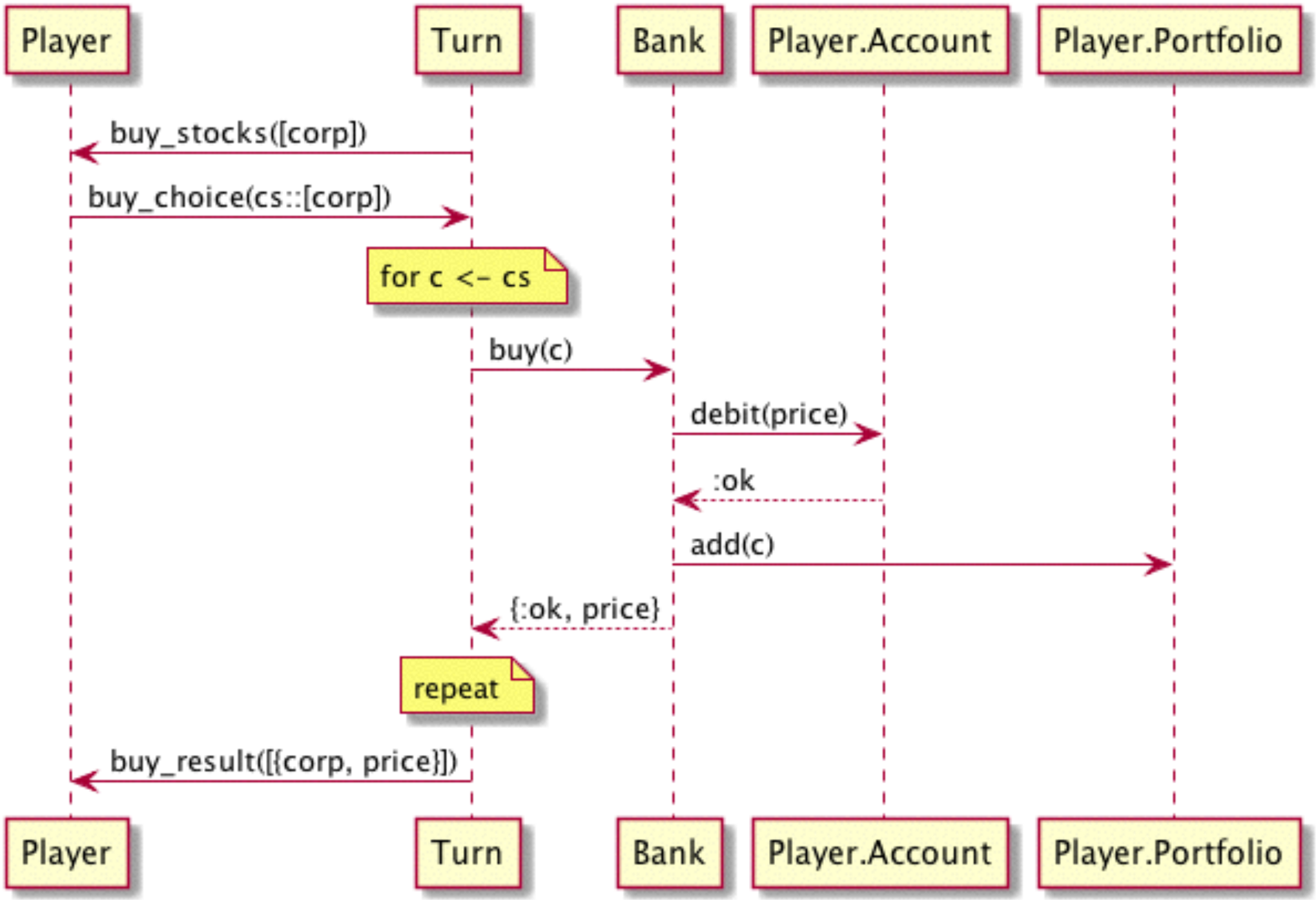
PHOENIX

Quantum

Buy Stocks


Player
Turn
Bank
Player.Account
Player.Portfolio
buy_stocks([corp])
buy_choice(cs::[corp])
for c <- cs
reserve c
{:ok, price, ref}
debit(price)
ok
buy(ref)
:ok
add(c)
repeat
buy_result([{corp, price}])
Complicated!!!
Player
Turn
Bank
Player.Account
Player.Portfolio

Buy Stocks Simplified



Clean Code

Had I read it back then…

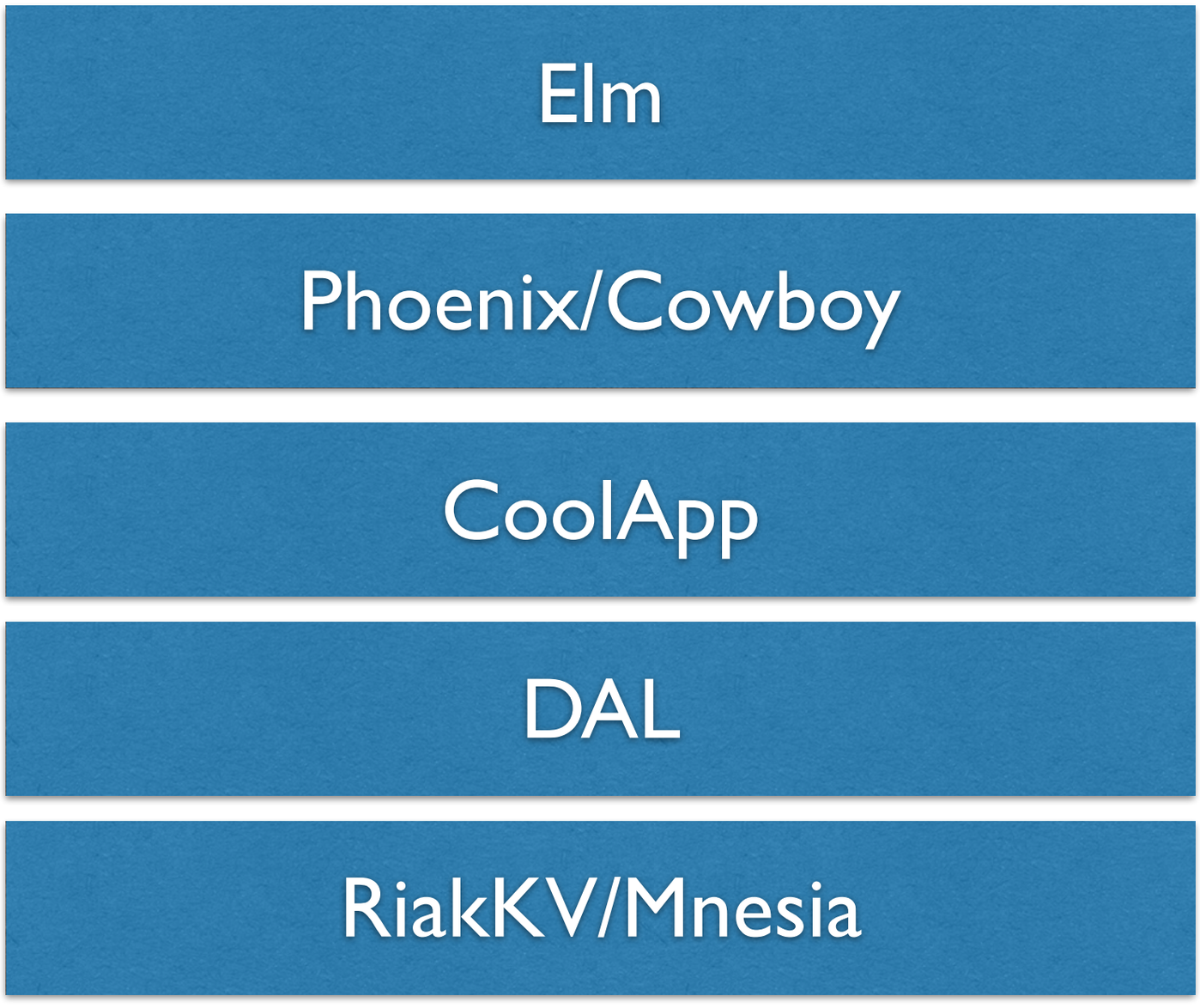
the temporal nature of the 1st design would have been avoided!

wishful thinking…

Putting it together

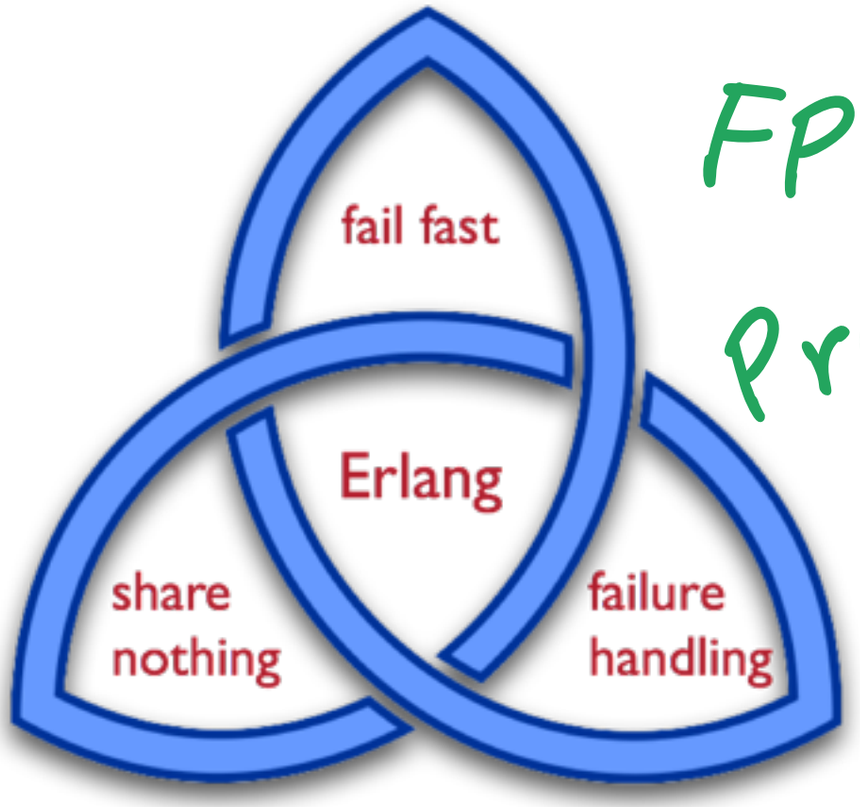
Stateful is the new black!

Architecture



DAL: Data Abstraction Layer

Failures



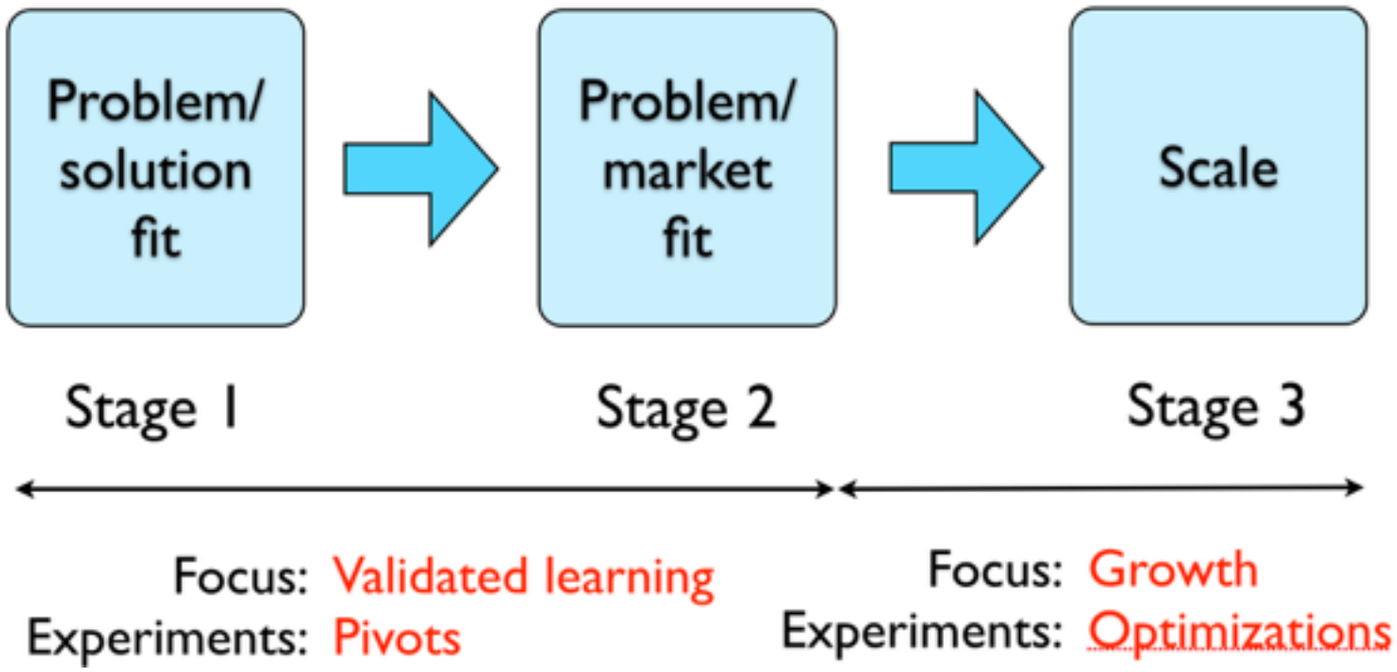
FP

Protocols

NoSQL

Embrace them

Learn from them



Try to make the good failures!

THANK YOU!

WE’RE HIRING!

- UK Client Service Engineer
- Developer Advocate EMEA

bashojobs.theresumator.com

VISIT OUR STAND

Experience our Riak TS demo and be entered to win a Scalextric set!

Get your invitation to our IoT Riak TS Roadshow

basho