# microservices

## *choices* and *challenges*

James Lewis - February, 2015
jalewis@thoughtworks.com
@boicy

ThoughtWorks®

ThoughtWorks®

# A 100-YEAR COMPANY

***Our mission is to better humanity through software and help drive the creation of a socially and economically just world.***

To run a sustainable business.

To champion software excellence and revolutionize the IT industry.

To advocate passionately for social and economic justice.

ThoughtWorks®

# GLOBAL BUSINESS: GLOBAL COMMUNITY

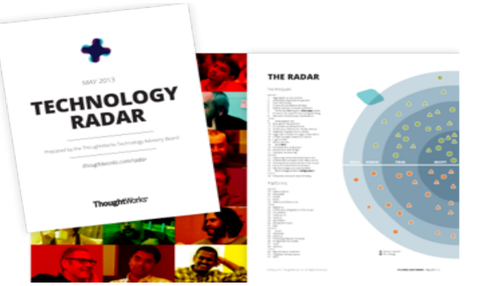
ThoughtWorks®

# OVER 20 YEARS OF THOUGHT LEADERSHIP


OpenMRS
MEDICAL RECORD SYSTEM

CALATRAVA

RAPIDFTR


...to name a few

*a question*

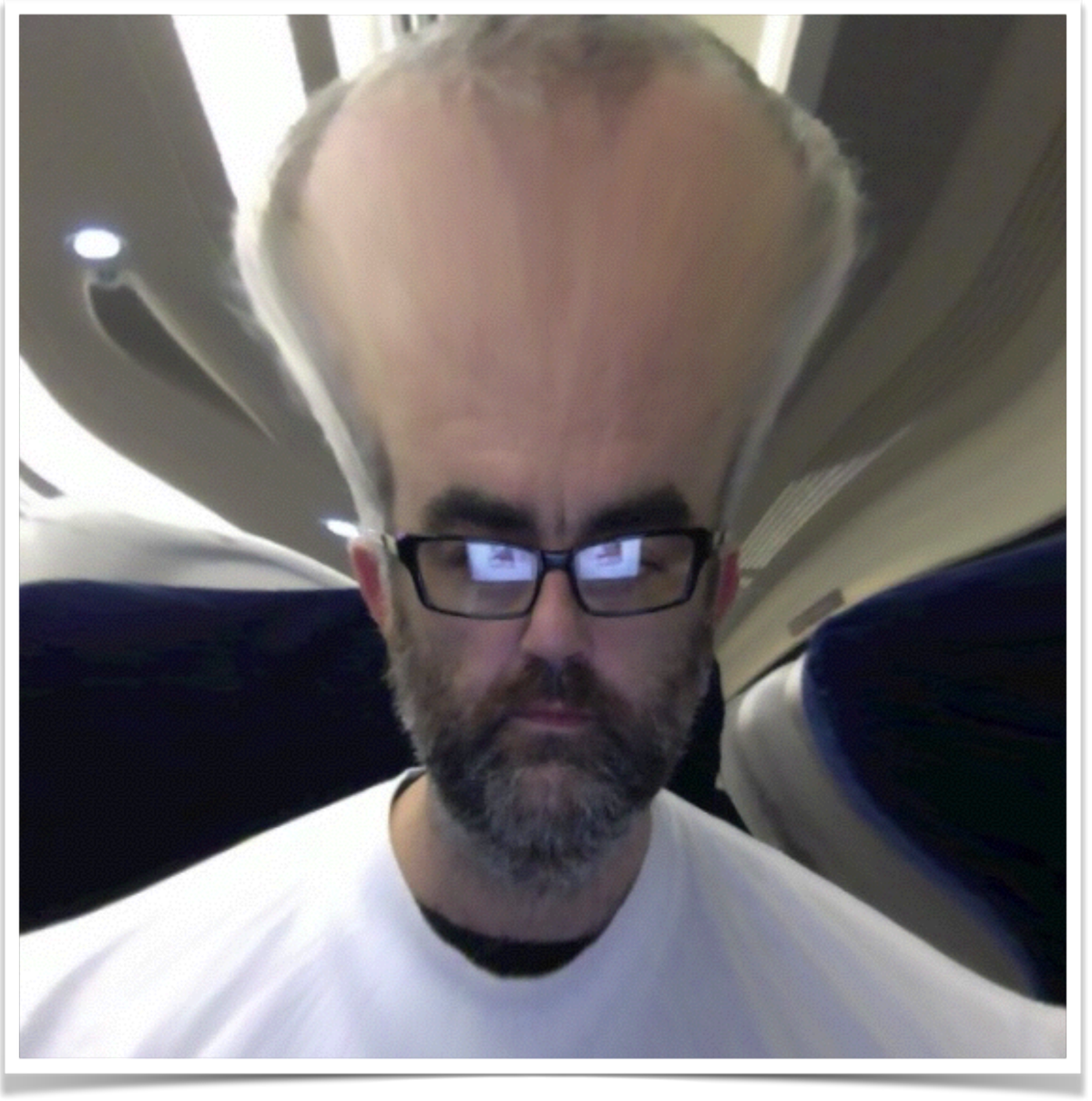
*how do you build systems that are*

***cheap to replace***

***quick to scale***

***withstand failure***

***and which allow you to go as "fast as possible"?***



# characteristics of microservices

*componentisation via services*

*organised around business capabilities*

*decentralised data management*

*products not projects*

*decentralised governance*

*smart endpoints and dumb pipes*

*evolutionary design*

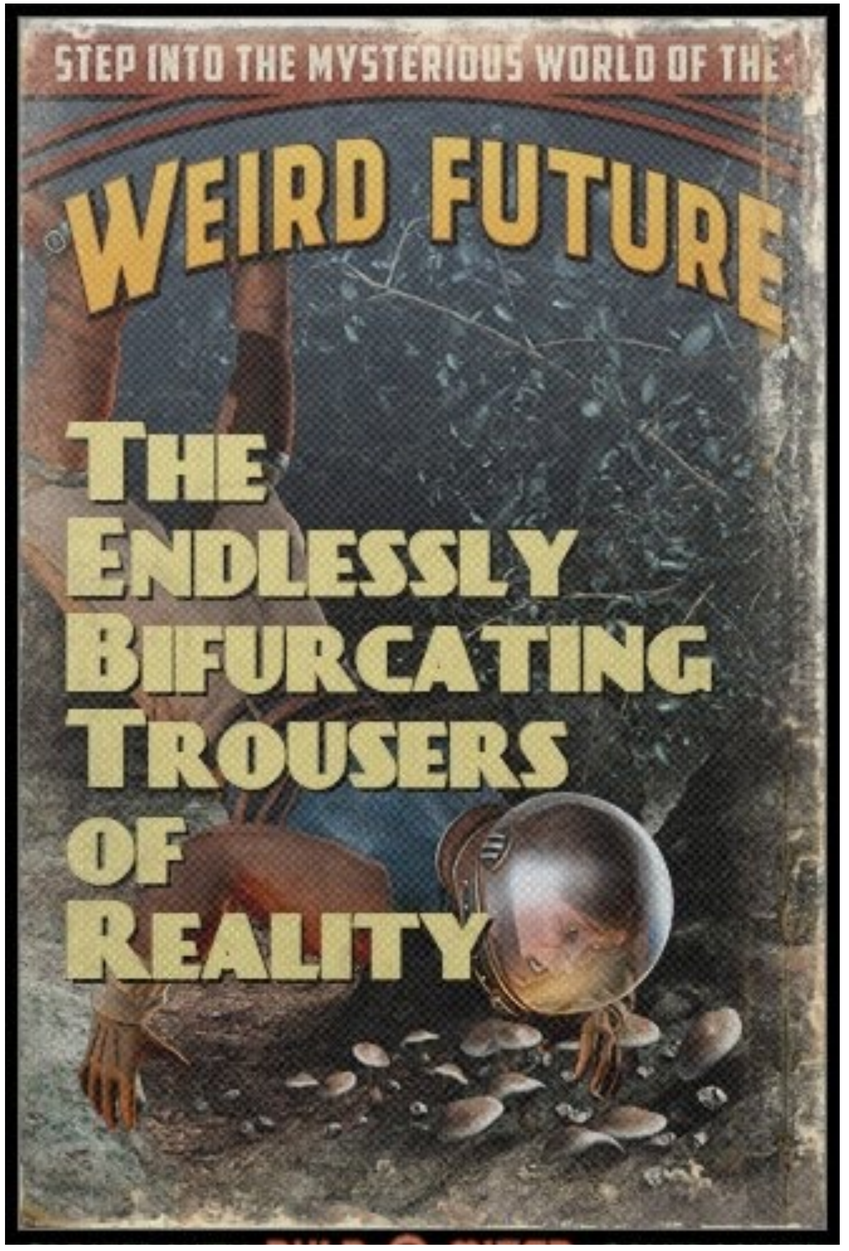
*infrastructure automation*

*designed for failure*

http://martinfowler.com/articles/microservices.html

***“It is perfectly true, as philosophers say, that life must be understood backwards. But they forget the other proposition, that it must be lived forwards.”***

*Søren Kierkegaard*


STEP INTO THE MYSTERIOUS WORLD OF THE
WEIRD FUTURE
THE ENDLESSLY BIFURCATING TROUSERS OF REALITY

# History

The lawful good product owners of the publishing house had long lived in awe and fear of their publishing systems.

In awe, for they had made a tremendous amount of Gold, but in fear of the time taken to change them, their slowness and their fragility.

A messenger was sent to fetch help from a distant land famed for it’s mighty wizards. You have taken up the challenge...

# 1.

You must save the product owners by rebuilding their website. You start off the project. In the course of discussions you discover that your goals are three fold:

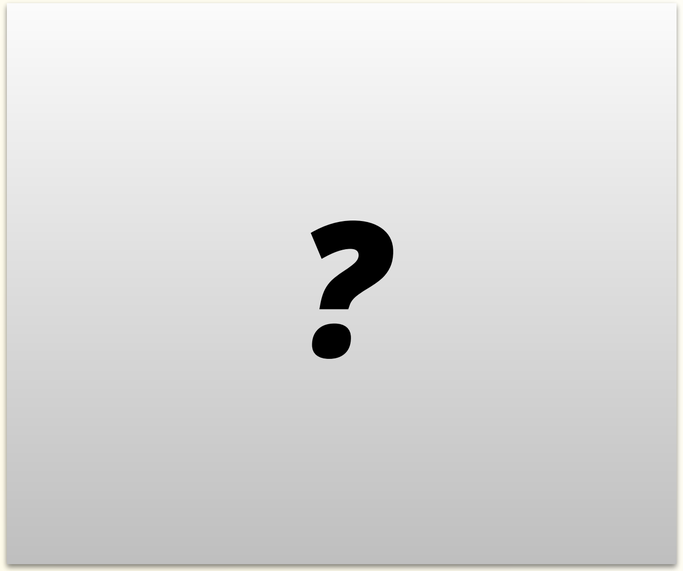
1. improve availability
2. improve performance
3. reduce the cost of delay

An Enterprise Architect approaches and addresses you.

You may use:

| | |
|---|---|
| Summon Walking Skeleton | turn to 4 |
| Analysis Paralysis | turn to 3 |

If you have none of these you will have to draw your sword and fight (turn to **178**)



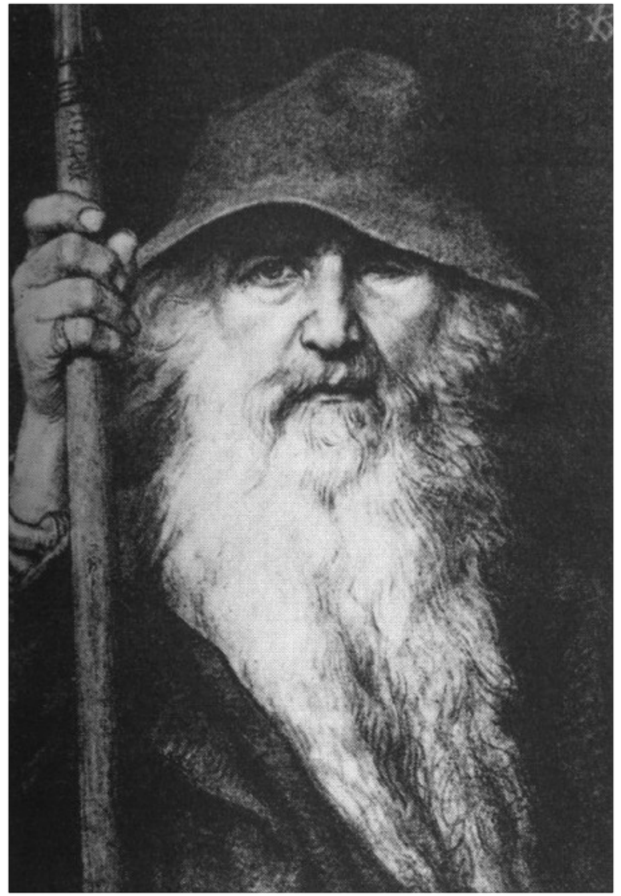
# 3.

You cast Analysis Paralysis at the Enterprise Architect.

“Foolish young adventurer” says the architect, “we follow the evolutionary school of architecture and we shall have none of the lawful-evil ways of waterfall”.

The last thing you see before everything goes dark is the architect incanting in a strange voice.

You have died. Turn to page 1.



# 1.

You must save the product owners by rebuilding their website. You start off the project. In the course of discussions you discover that your goals are three fold:

1. improve availability
2. improve performance
3. reduce the cost of delay

An Enterprise Architect approaches and addresses you.

You may use:

| | |
|---|---|
| Summon Walking Skeleton | turn to 4 |
| Analysis Paralysis | turn to 3 |

If you have none of these you will have to draw your sword and fight (turn to **178**)

# 4.

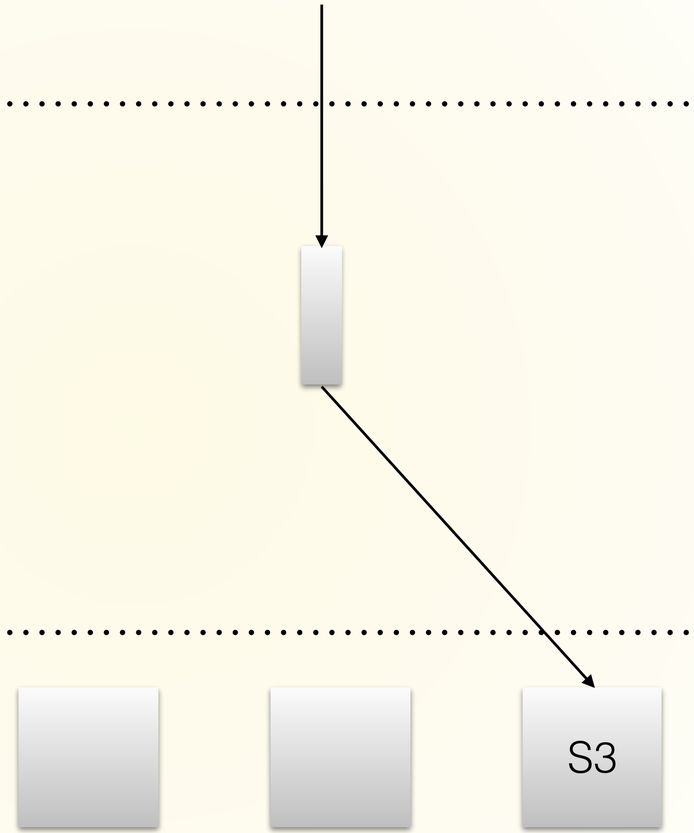
Your walking skeleton coalesces in a cloud of noxious gasses and solidifies as a java dropwizard application.

You reach into your backpack and deploy the content store. Your walking skeleton reaches out it's skeletal arms and grabs armfuls of raw xml.

Would you like to:

Transform the xml inside the skeleton — turn to 6

Use a magic box — turn to 5



# 5.

You throw the magic box in between the walking skeleton and the content store.

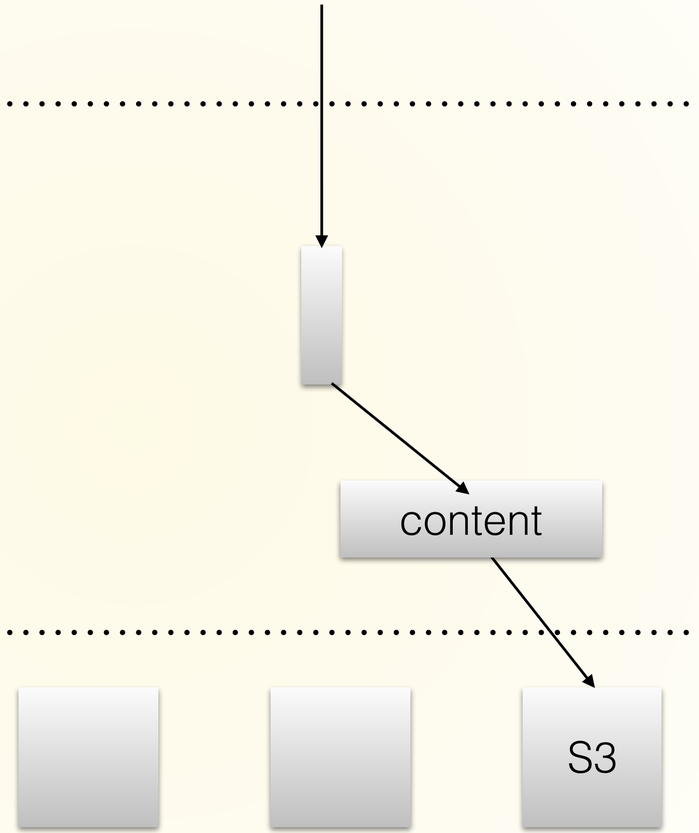
A villager approaches and exclaims: “this beautiful content I see in front of me seems to take an awful long time to get here”

You must somehow make the content arrive faster.

If you have a http cache in your inventory, you may use it now.

Cache in between S3 and content turn to 10

Cache in between skeleton and content turn to 33



# 6.

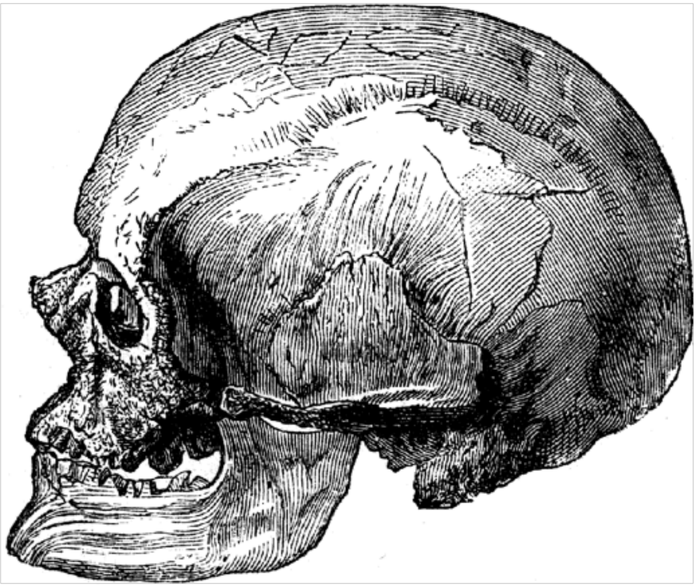
The skeleton gurgles, grunts and then doubles in size.

A villager approaches and exclaims: “this beautiful content I see in front of me seems to take an awful long time to get here”

You try to add a cache into the skeleton’s bony skull. First you cast sticky sessions. With a splash it rebounds, soaking you in the stench of the unscalable.

Desperately, you try terracotta and then the oracle of coherence. Nothing seems to work. The murky substances overwhelm you.

You have died. turn to page 1.



# 10.

The cache causes the content load times to drop from 300ms to 150ms.

The villager says “this wonderful content is now arriving more swiftly than even the knight-messengers of the Empress”.

The villagers are happy but all too soon, all is not well for the content has a long tail. You must work out how to refresh the content when it changes.

You can either:

Refresh the content when it appears from the ether turn to 150

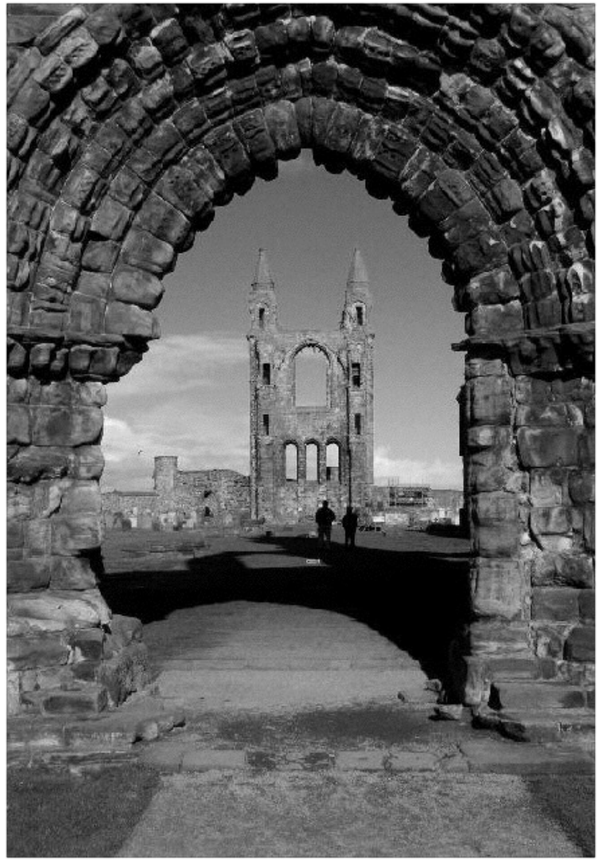
Trust that it will be fast enough on first view turn to 22

# 22.

The tail is just too long. When villagers or merchants try to use the content it is just too slow to arrive.

The amount of Gold diminishes and over the years the village fades into a forgotten hamlet, then to a legend and a myth.

You have died, turn to page 1.



# 150.

Content trickles into the store. You keep up by listening for the new content and casting “wget” on the cache to keep it refreshed.

New types of content appears - content the villagers have never seen before. Content the walking skeleton is unable to combat.

Fortunately, through Continuous Delivery you are able to keep up with the changed content but the cache doesn’t. The cache becomes stale.

How will you keep your delivery continuous?

cast cache shards — turn to **255**

If you are unable to shard the cache turn to page **48**

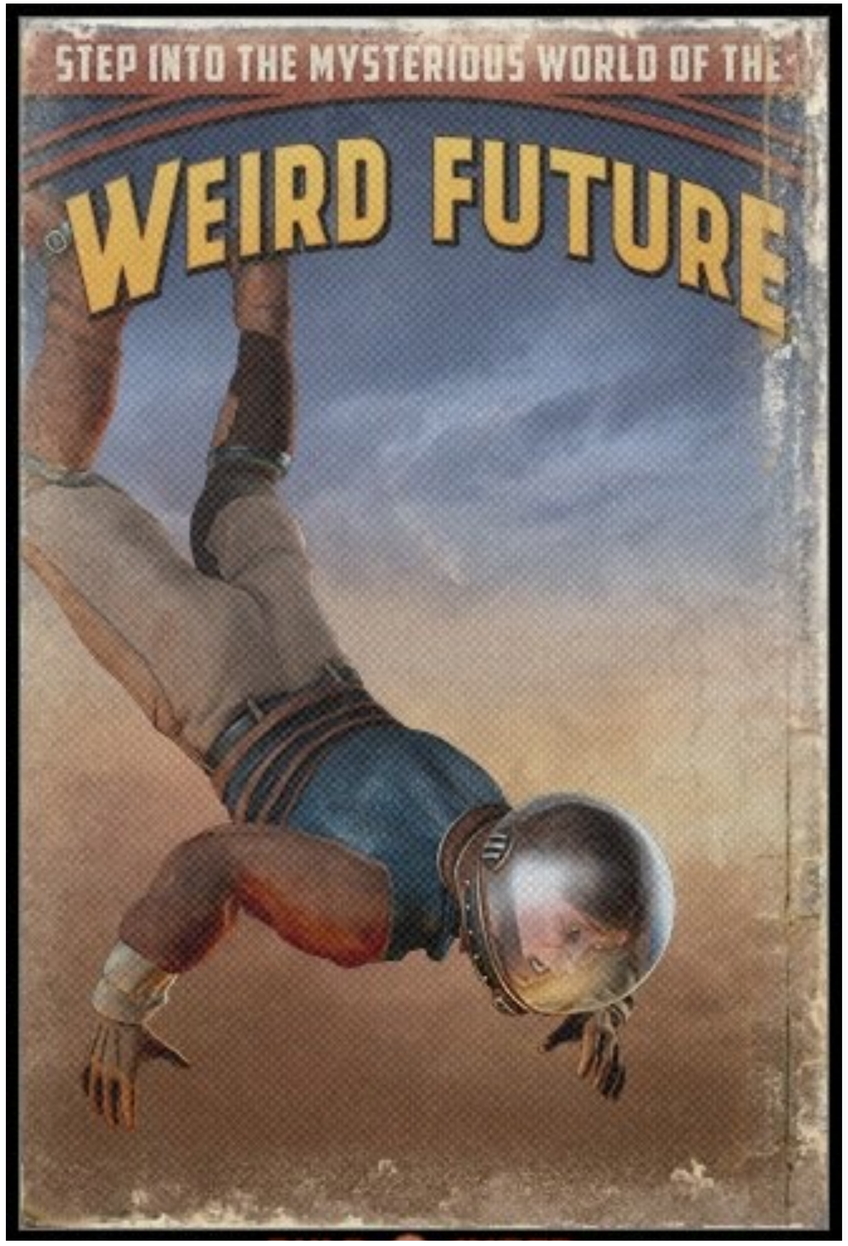
# 33.

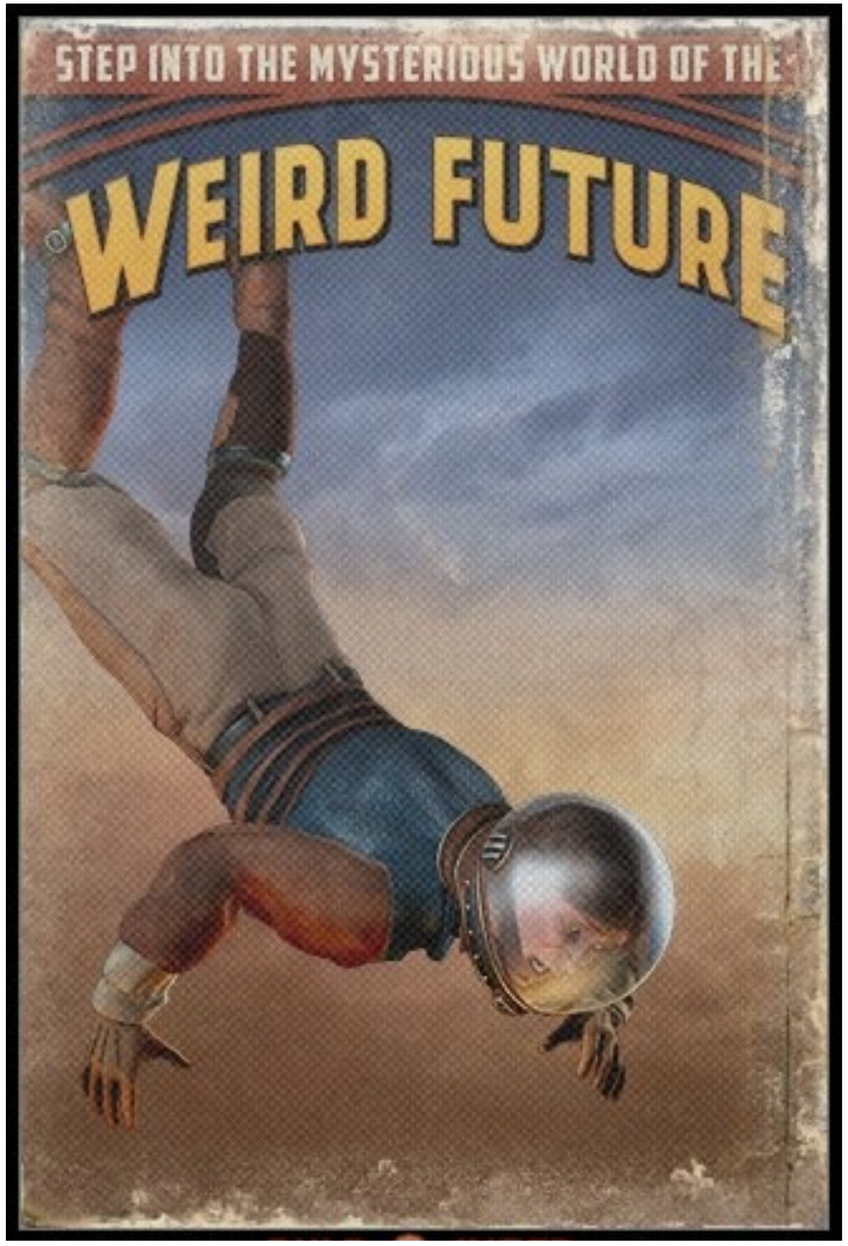
The HTTP cache has an instant effect. Latency drops from 300ms to 10ms.

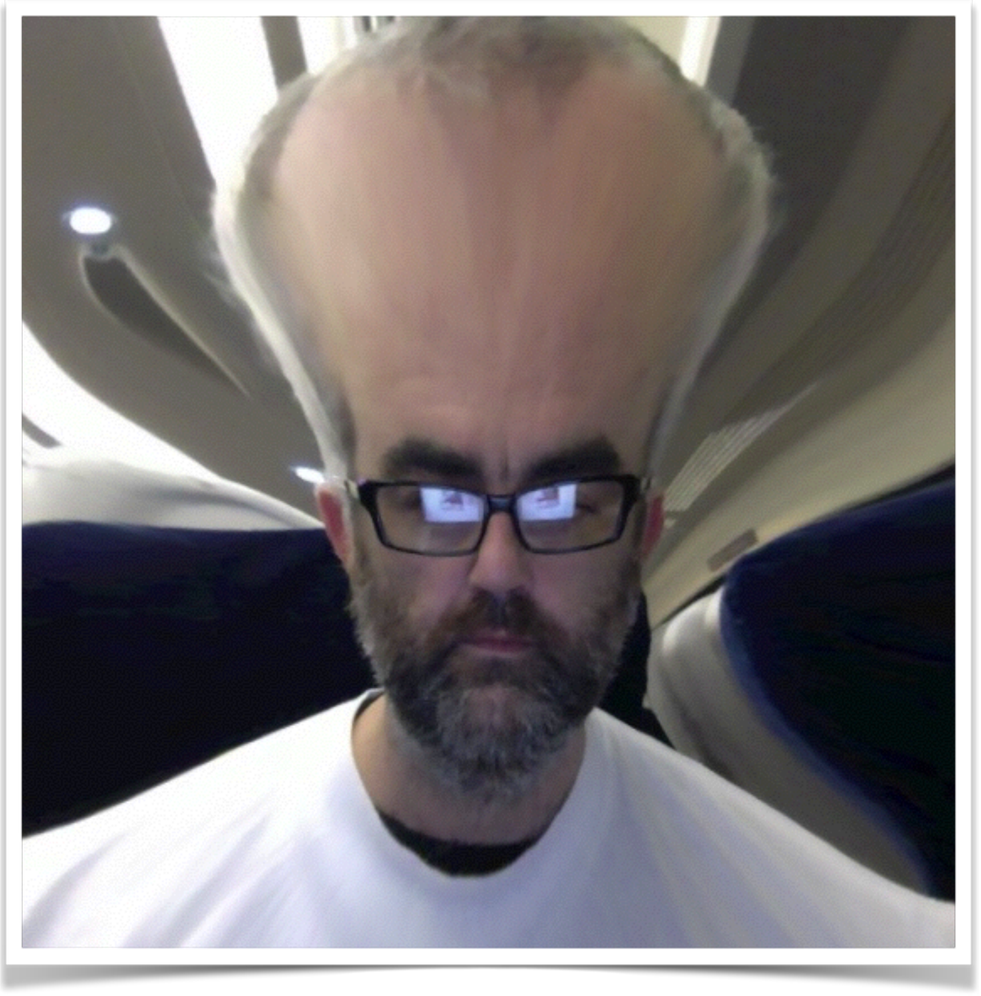
Changes to the content mount up. Every time one of the lawful-good researches publishes something, the cache must be refreshed. Every time the skeleton changes it's appearance, the cache must be refreshed.

The villagers need you to do something. Will you:

| | |
|---|---|
| Suffer the long tail | turn to 22 |
| Refresh the cache on API and content changes | turn to 150 |


STEP INTO THE MYSTERIOUS WORLD OF THE
WEIRD FUTURE


STEP INTO THE MYSTERIOUS WORLD OF THE
WEIRD FUTURE



***there is no such thing as the “right” way***

do we split now or later?

is 1000 LoC too big?

are 600 services too many?

***let future you worry about that when you know more***


when to use microservices


IT DEPENDS!
HELL YEAH!

Are you a startup? Do you have to get to market really soon?

Are you exploring your domain?

Do you want to exercise options at a later date?

*more options for replacing*

*more options for scaling*

*more options for deploying*

# characteristics of microservices

*componentisation via services*

*organised around business capabilities*

*decentralised data management*

*products not projects*

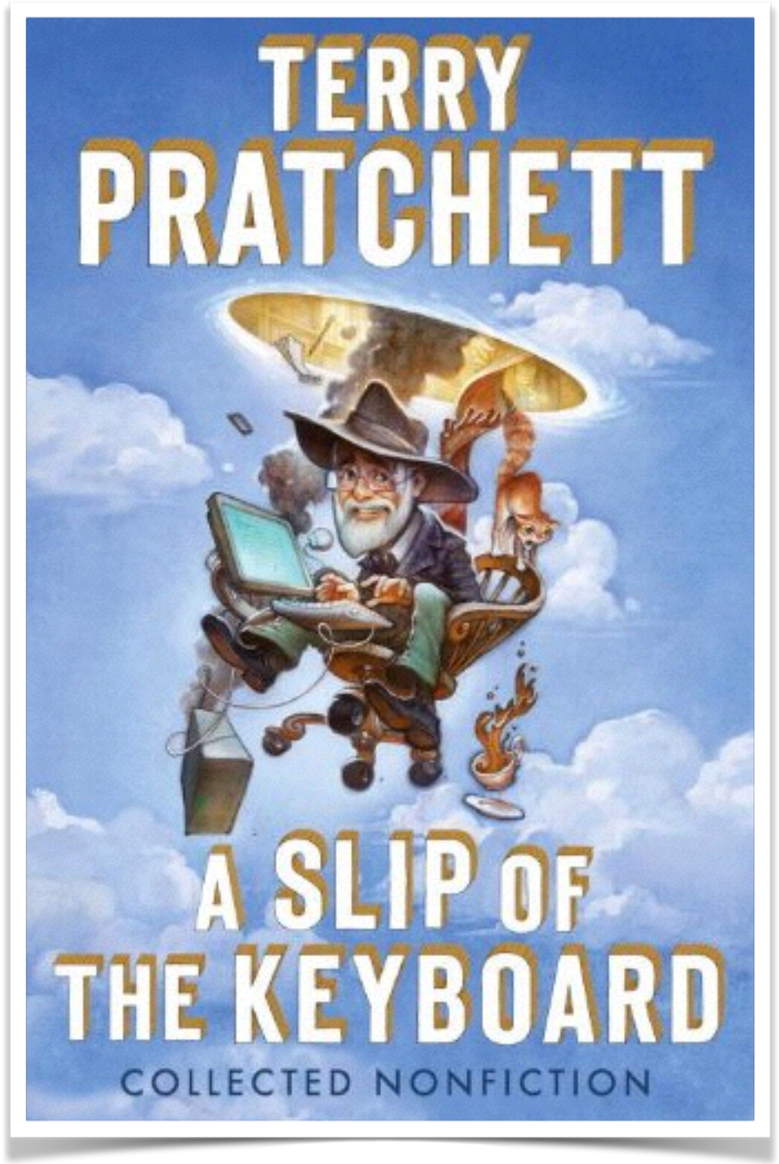
*decentralised governance*

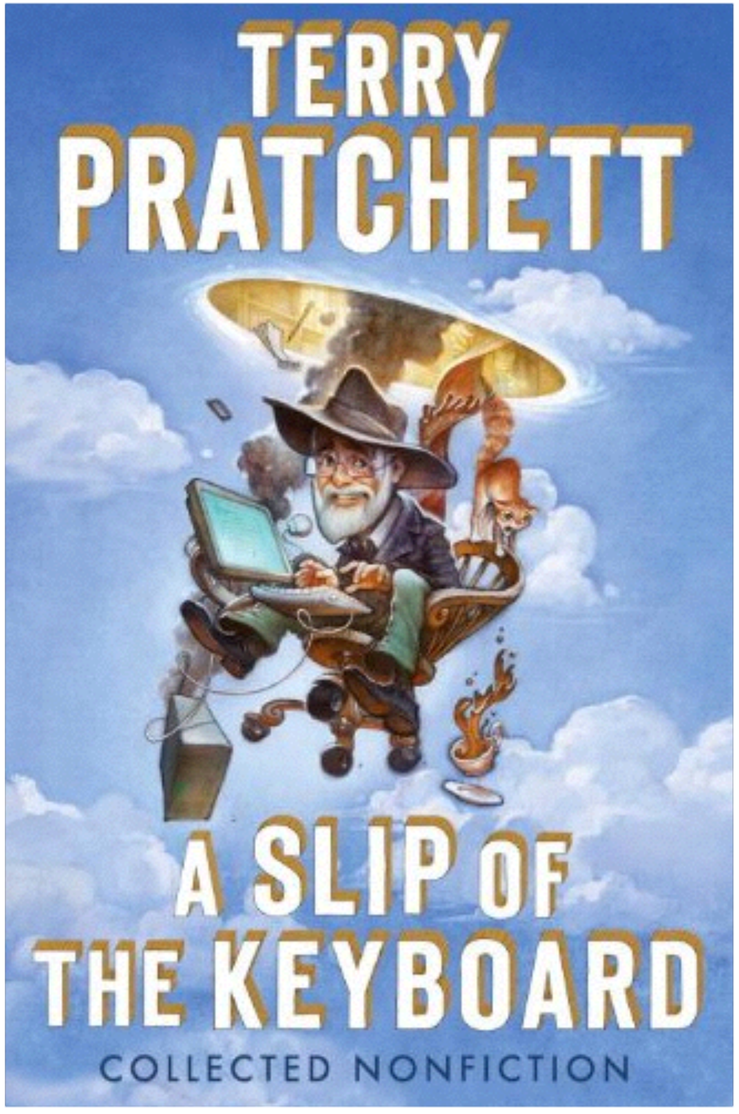
*smart endpoints and dumb pipes*

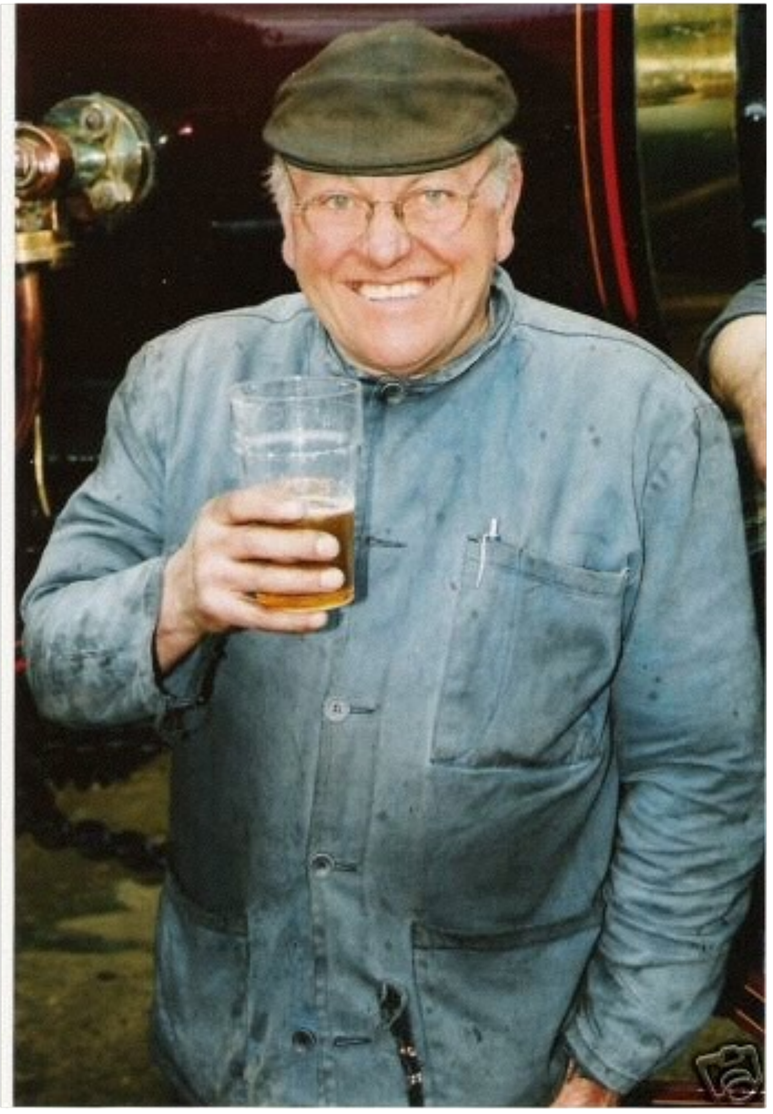
*evolutionary design*

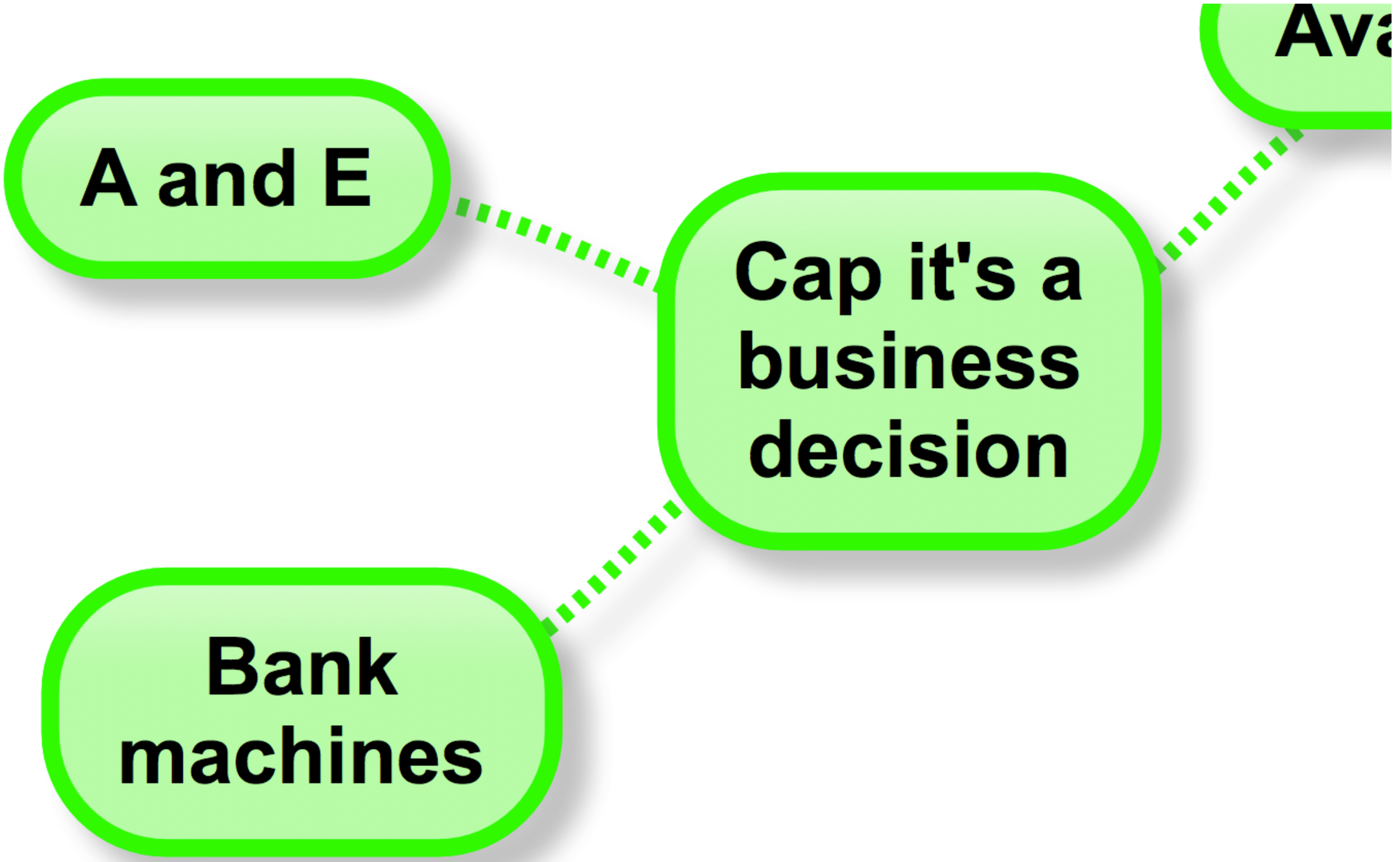
*infrastructure automation*

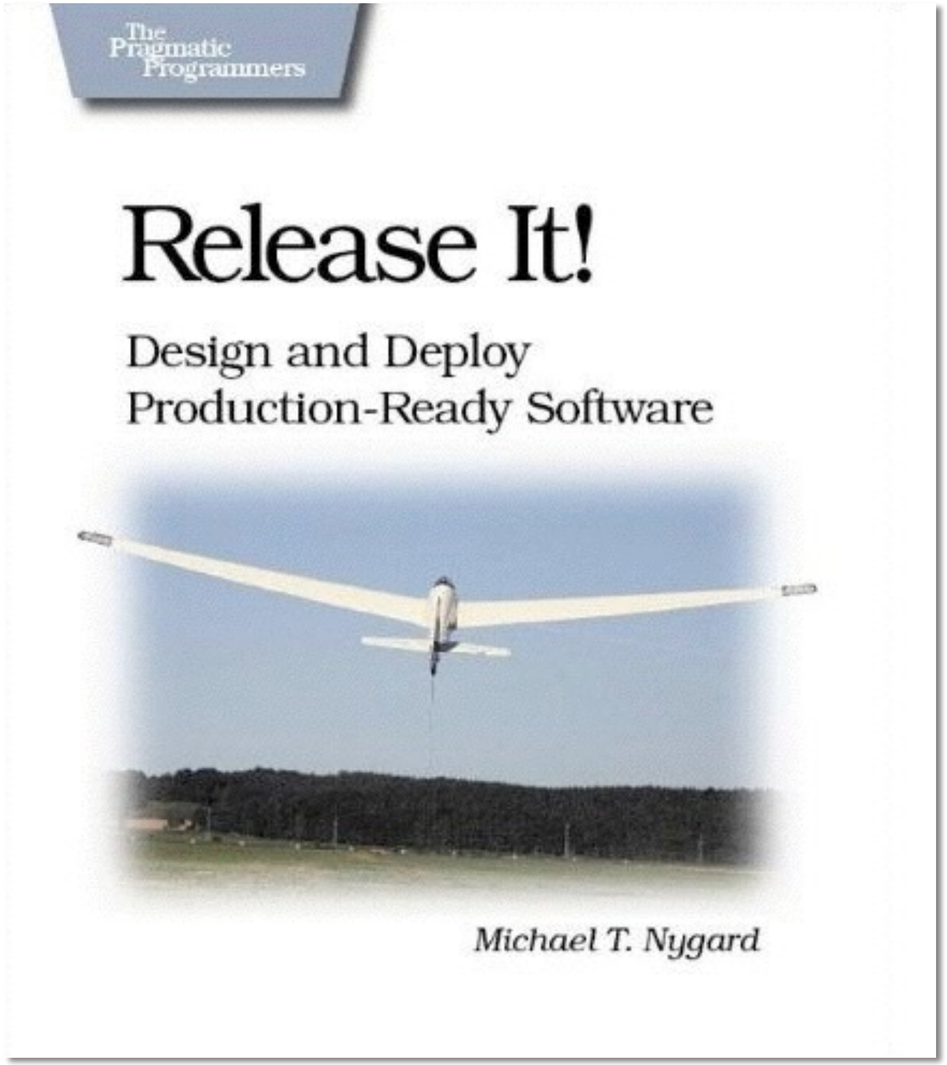
***designed for failure***


TERRY
PRATCHETT
A SLIP OF
THE KEYBOARD
COLLECTED NONFICTION


TERRY
PRATCHETT
A SLIP OF
THE KEYBOARD
COLLECTED NONFICTION

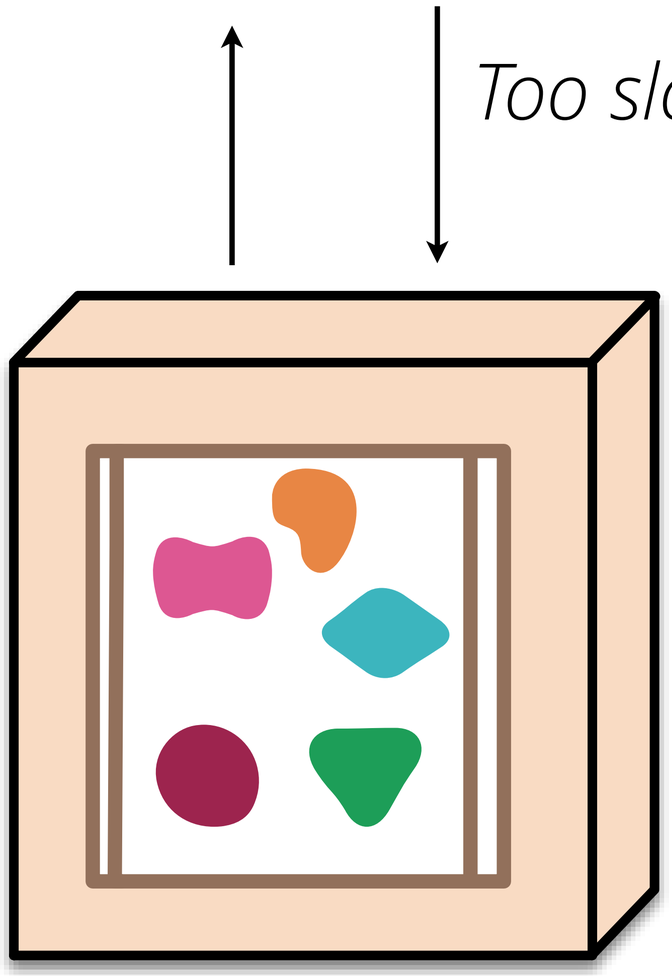



A and E
Cap it's a business decision
Bank machines




***"Every socket, process, pipe, or remote procedure call can and will hang. Even database calls [...]"***

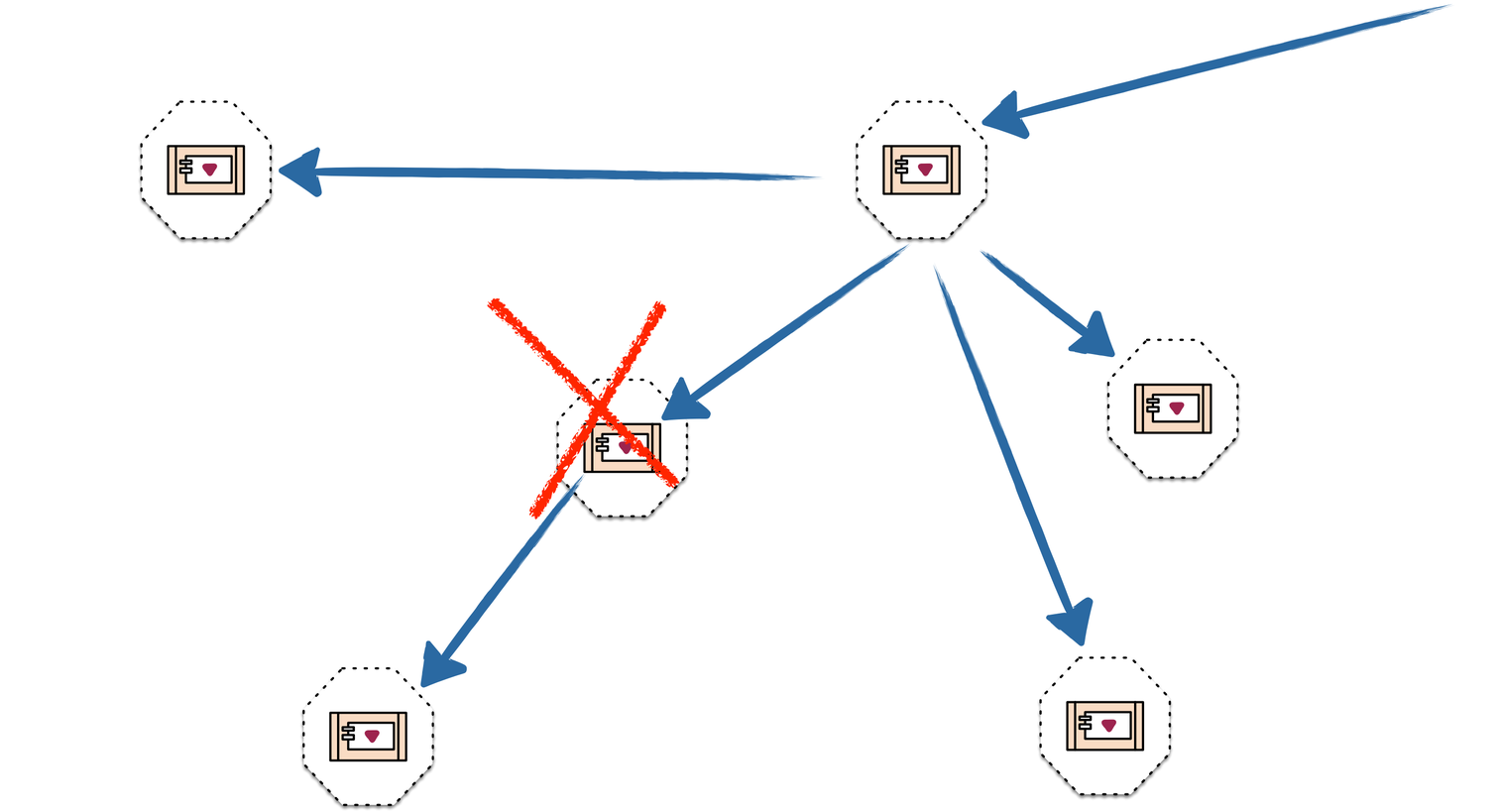
*M. Nygard, "Release It"*




**UP**

**DOWN**

# *Too slow? Where is the problem?*



**UP?** **DOWN?**

# characteristics of microservices

*componentisation via services*

*organised around business capabilities*

*decentralised data management*

*products not projects*

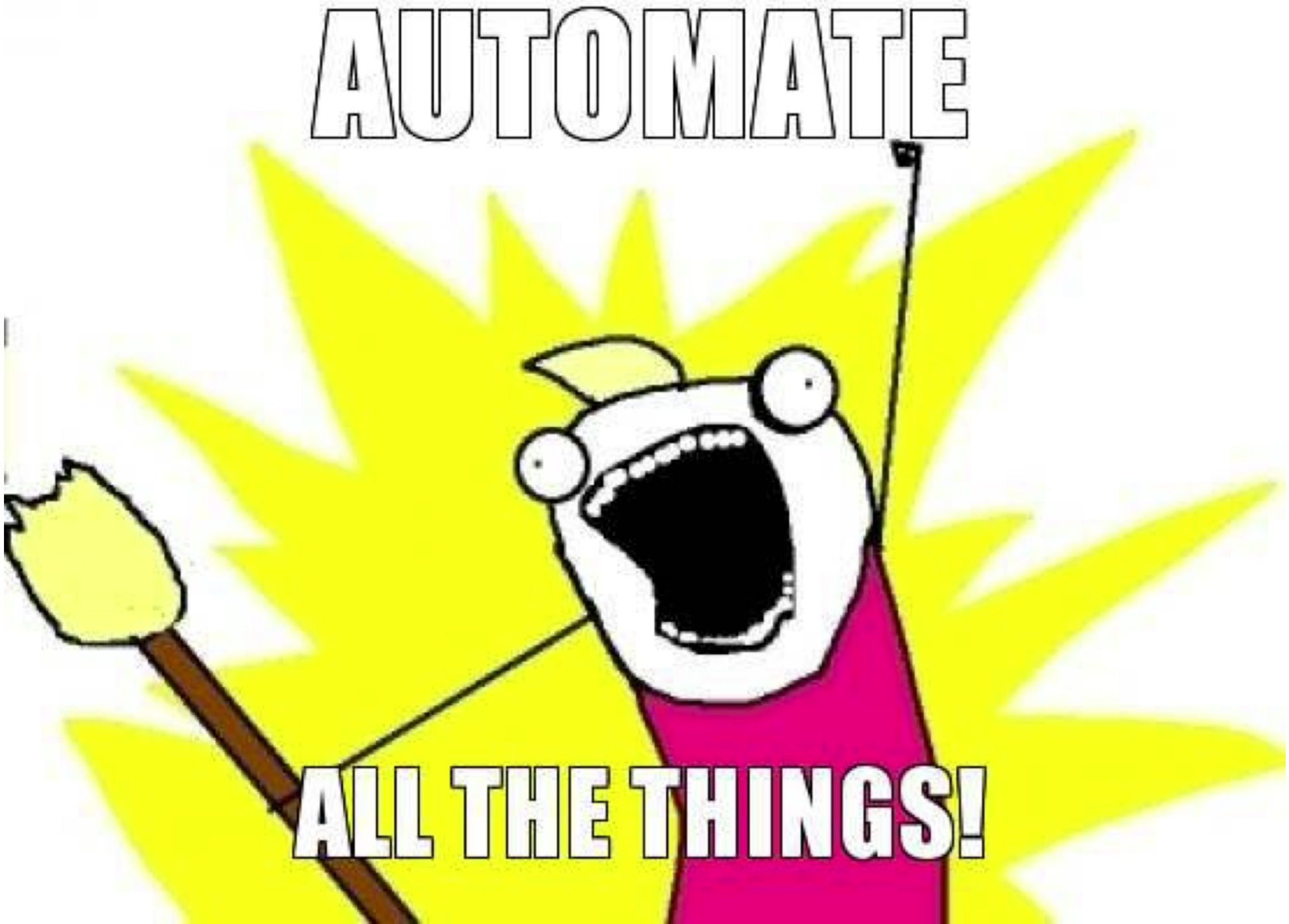
*decentralised governance*

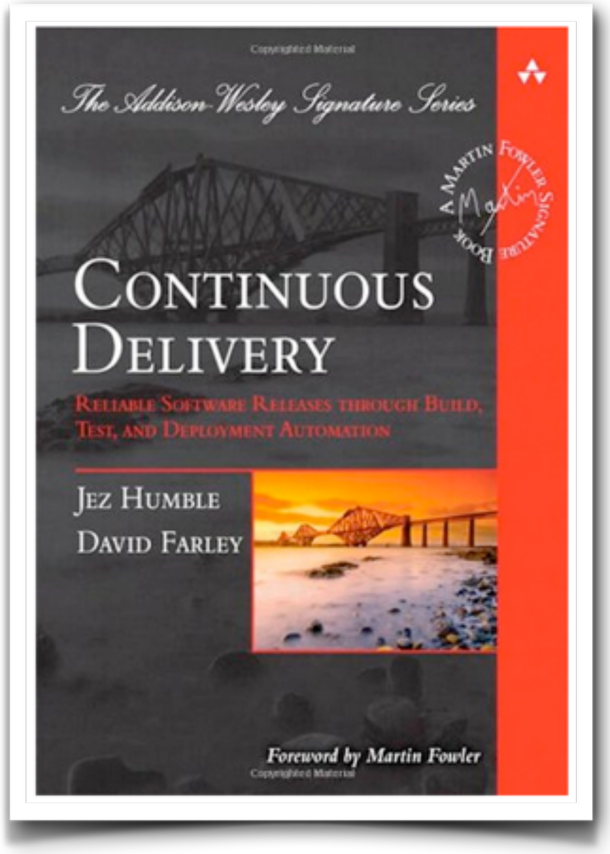
*smart endpoints and dumb pipes*

*evolutionary design*

***infrastructure automation***

*designed for failure*

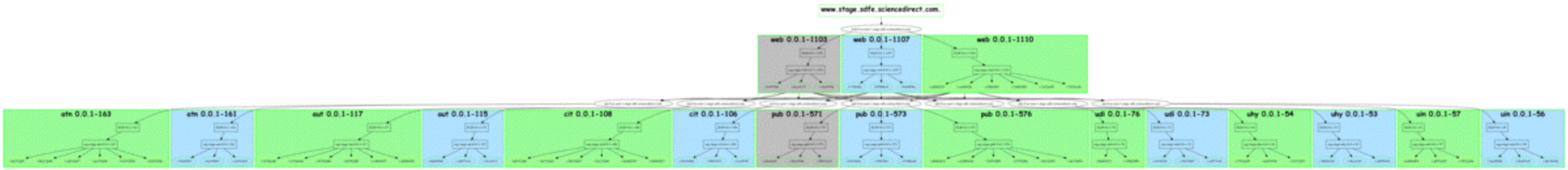

AUTOMATE
ALL THE THINGS!




***blue / green deploys***

***canary releases***

***infrastructure as code***


www.stage.sdfe.sciencedirect.com.
web 0.0.1-1103
web 0.0.1-1107
web 0.0.1-1110
otn 0.0.1-163
otn 0.0.1-161
out 0.0.1-117
out 0.0.1-115
cit 0.0.1-108
cit 0.0.1-106
pub 0.0.1-571
pub 0.0.1-573
pub 0.0.1-576
udi 0.0.1-76
udi 0.0.1-73
uhy 0.0.1-54
uhy 0.0.1-53
uin 0.0.1-57
uin 0.0.1-56

***“I see it all perfectly; there are two possible situations - one can either do this or that. My honest opinion and my friendly advice is this: do it or do not do it - you will regret both.”***

*Søren Kierkegaard*

# FINAL THOUGHTS

# the 17 rules of UNIX programming

1. Rule of **Modularity**: Write simple parts connected by clean interfaces.
2. Rule of **Clarity**: Clarity is better than cleverness.
3. Rule of **Composition**: Design programs to be connected to other programs.
4. Rule of **Separation**: Separate policy from mechanism; separate interfaces from engines.
5. Rule of **Simplicity**: Design for simplicity; add complexity only where you must.
6. Rule of **Parsimony**: Write a big program only when it is clear by demonstration that nothing else will do.
7. Rule of **Transparency**: Design for visibility to make inspection and debugging easier.
8. Rule of **Robustness**: Robustness is the child of transparency and simplicity.
9. Rule of **Representation**: Fold knowledge into data so program logic can be stupid and robust.
10. Rule of **Least Surprise**: In interface design, always do the least surprising thing.
11. Rule of **Silence**: When a program has nothing surprising to say, it should say nothing.
12. Rule of **Repair**: When you must fail, fail noisily and as soon as possible.
13. Rule of **Economy**: Programmer time is expensive; conserve it in preference to machine time.
14. Rule of **Generation**: Avoid hand-hacking; write programs to write programs when you can.
15. Rule of **Optimization**: Prototype before polishing. Get it working before you optimize it.
16. Rule of **Diversity**: Distrust all claims for “one true way”.
17. Rule of **Extensibility**: Design for the future, because it will be here sooner than you think.


The Pragmatic Programmers
Release It!
Design and Deploy
Production-Ready Software
The Addison-Wesley Signature Series
Enterprise Integration Patterns
Designing, Building, and Deploying Messaging Solutions
Gregor Hohpe
Bobby Woolf
The Addison-Wesley Signature Series
Continuous Delivery
Reliable Software Releases through Build, Test, and Deployment Automation
Jez Humble
David Farley
Foreword by Martin Fowler
The Art of UNIX Programming
Eric S. Raymond
Domain-Driven DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler
REST in Practice

# THE ONE TRUE WAY?

# BETTERIDGE'S LAW OF HEADLINES

*“Any headline which ends in a question mark can be answered by the word **no**."*

## the 16th rule of unix programming

# The Rule of Diversity

***DISTRUST ALL CLAIMS FOR “ONE TRUE WAY”***

***I see it all perfectly; there are two possible situations - one can either do this or that. My honest opinion and my friendly advice is this: do it or do not do it - you will regret both.***

*Søren Kierkegaard*

***think*** *about failure*

***think*** *about failure*

***automate*** *your infrastructure*

***think*** *about failure*

***automate*** *your infrastructure*

***evolve*** *your system*

***think*** ***automate*** ***evolve***

ThoughtWorks®

***think*** ***automate*** ***evolve***

---

**THANKS**

---


***James Lewis***

***jalewis@thoughtworks.com***

***@boicy***