

Cloud Native is About Culture, Not Containers

(how to not fail at cloud native)

Holly Cummins
IBM
`@holly_cummins`





what **is** cloud native?



Daniel Bryant

@danielbryantuk

Following



I've gotta hand it to [@bibryam](#), he's got a great way of framing things... :-)



Bilgin Ibryam @bibryam

ESB -> Microservices -> CloudNative

8:42 AM - 7 Aug 2018

2 Retweets 7 Likes





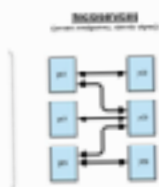
Daniel Bryant

@danielbryantuk

Following



I've gotta hand it to [@bibryam](#), he's got a great way of framing things... :-)



Bilgin Ibryam @bibryam

ESB -> Microservices -> CloudNative

8:42 AM - 7 Aug 2018

2 Retweets 7 Likes



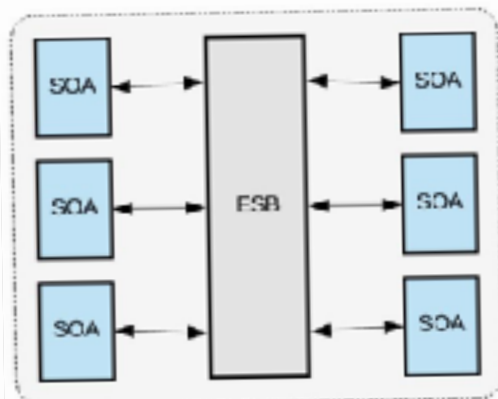
**Daniel Bryant**

@danielbryantuk

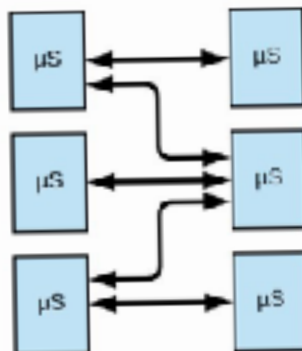
Following

**SOA/ESB**

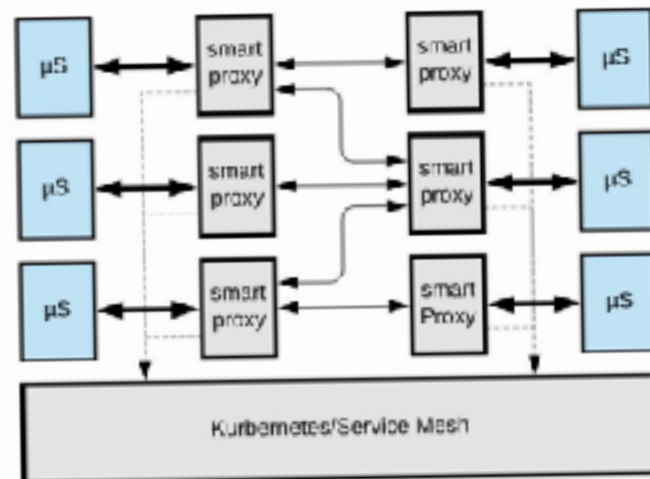
(smart pipes, dumb endpoints)

**Microservices**

(smart endpoints, dumb pipes)

**Cloud Native**

(smart platform, dumb services)



(a great
article, btw)

The screenshot shows a web browser displaying the InfoQ website. The URL in the address bar is www.infoq.com/articles/microservices-post-kubern.... The InfoQ logo is in the top left, with navigation links for Development, Architecture & Design, AI, ML & Data Engineering, Culture & Methods, DevOps, and Videos with Transcripts. A banner for the Software Development Conference in San Francisco is visible. Below the navigation bar, there's a section for 'You are here: InfoQ Homepage > Articles > Microservices in a Post-Kubernetes Era'. The article title 'Microservices in a Post-Kubernetes Era' is prominently displayed. Below the title, it says 'Posted by Bilgin Ibryam, reviewed by Daniel Bryant on Sep 01, 2018. Estimated reading time: 8 minutes'. There are buttons for 'Like', 'Discuss', and 'Share'. A 'Reading List' button is also present. The article content includes a 'Key Takeaways' section with a bulleted list. On the right, there's a 'RELATED CONTENT' section with links to other articles and their authors' profile pictures.

InfoQ
En | 中文 | 日本 | Fr | De
1,201,580 Aug unique visitors

Development Architecture & Design AI, ML & Data Engineering Culture & Methods DevOps Videos with Transcripts

Software Development Conference
San Francisco Nov 6-8, 2018
London Mar 4 - 5, 2019

Streaming Machine Learning Reactive Microservices Containers .NET All topics

The Architects' Newsletter

You are here: [InfoQ Homepage](#) > [Articles](#) > [Microservices in a Post-Kubernetes Era](#)

Microservices in a Post-Kubernetes Era

Like / Posted by [Bilgin Ibryam](#), reviewed by [Daniel Bryant](#) on Sep 01, 2018. Estimated reading time: 8 minutes 1 Discuss

Share

Reading List Read later

Key Takeaways

- The microservice architecture is still the most popular architectural style for distributed systems. But Kubernetes and the cloud native movement has redefined certain aspects of application design and development at scale.
- On a cloud native platform, observability of services is not enough. A more fundamental prerequisite is to make microservices automatable, by implementing health checks, reacting to signals, declaring resource consumption, etc.
- In the post-Kubernetes era, using libraries to implement operational networking programs features that include...

RELATED CONTENT

- [eBay Replatforming to Kubernetes, Envoy and Kafka: Intending to Open Source Hardware and Software](#) Sep 18, 2018
- [Networking Your Microservices Applications](#) Sep 21, 2018
- [Designing Events-First Microservices](#) Sep 13, 2018

(a great
article, btw)

The screenshot shows a web browser displaying an article on the InfoQ website. The browser's address bar shows the URL `www.infoq.com/articles/microservices-post-kubern...`. The InfoQ logo is in the top left, with navigation links for Development, Architecture & Design, AI, ML & Data Engineering, Culture & Methods, DevOps, and Videos with Transcripts. A banner for the 'Software Development Conference San Francisco Nov 6-8, 2018' is visible in the top right. Below the navigation bar, a horizontal menu lists various topics: Streaming, Machine Learning, Reactive, Microservices, Containers, .NET, and All topics. The article title 'Microservices' is prominently displayed in the center, with a sub-header '8 minutes' and a 'Discuss' button. Below the title, there are social media sharing icons and buttons for 'Reading List' and 'Read later'. The article content is divided into two main sections: 'Key Takeaways' on the left and 'RELATED CONTENT' on the right. The 'Key Takeaways' section lists three bullet points about microservice architecture, Kubernetes, and operational networking. The 'RELATED CONTENT' section lists three related articles with their respective authors' profile pictures.

InfoQ
En | 中文 | 日本 | Fr | De
1,201,580 Aug unique visitors

Development Architecture & Design AI, ML & Data Engineering Culture & Methods DevOps Videos with Transcripts

Software Development Conference San Francisco Nov 6-8, 2018 London Mar 4 - 5, 2019

Streaming Machine Learning Reactive Microservices Containers .NET All topics

The Architects' Newsletter

Microservices

8 minutes 1 Discuss

Share + i t y o f e m

Reading List Read later

Key Takeaways

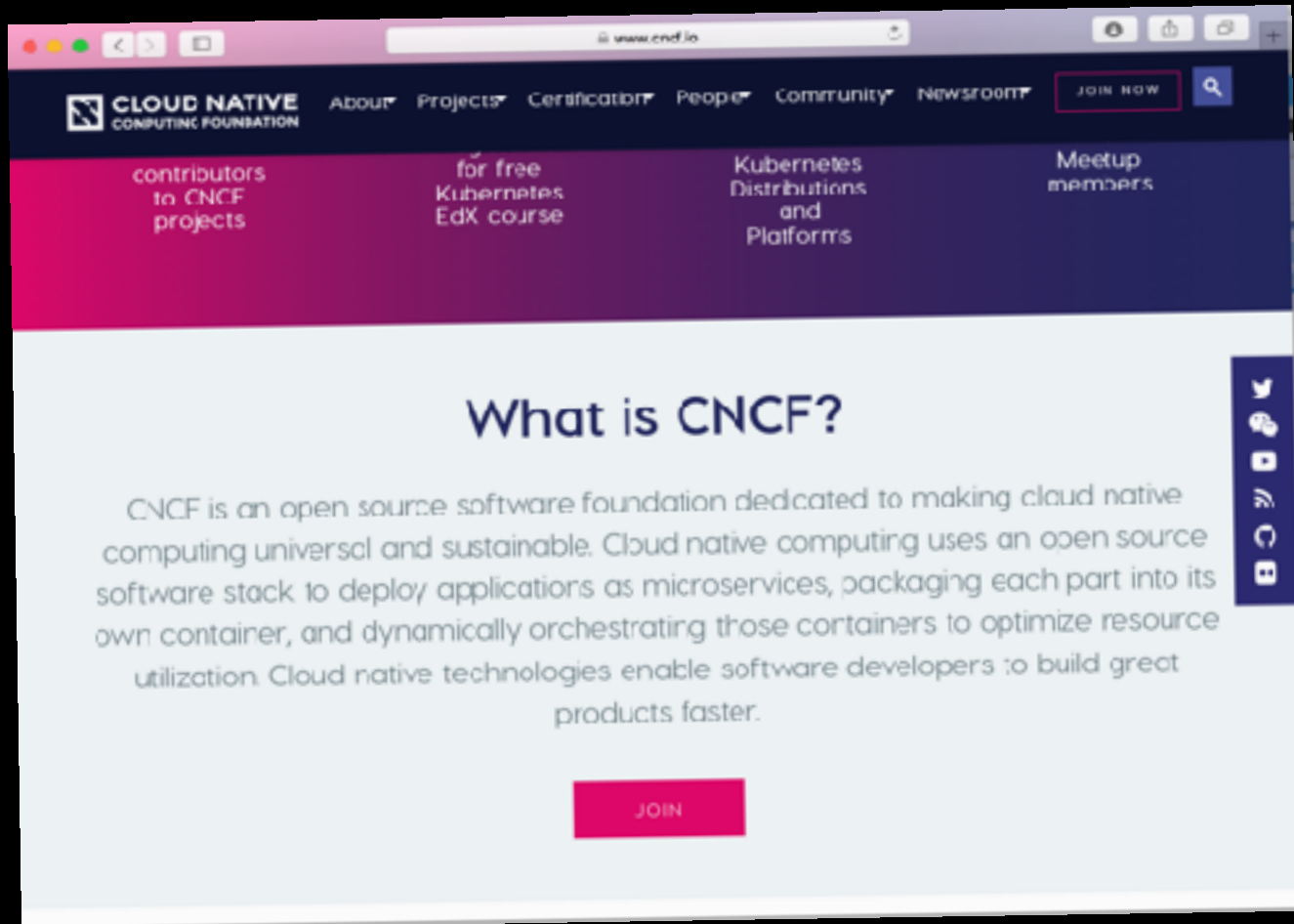
- The microservice architecture is still the most popular architectural style for distributed systems. But Kubernetes and the cloud native movement has redefined certain aspects of application design and development at scale.
- On a cloud native platform, observability of services is not enough. A more fundamental prerequisite is to make microservices automatable, by implementing health checks, reacting to signals, declaring resource consumption, etc.
- In the post-Kubernetes era, using libraries to implement operational networking programs features that include...

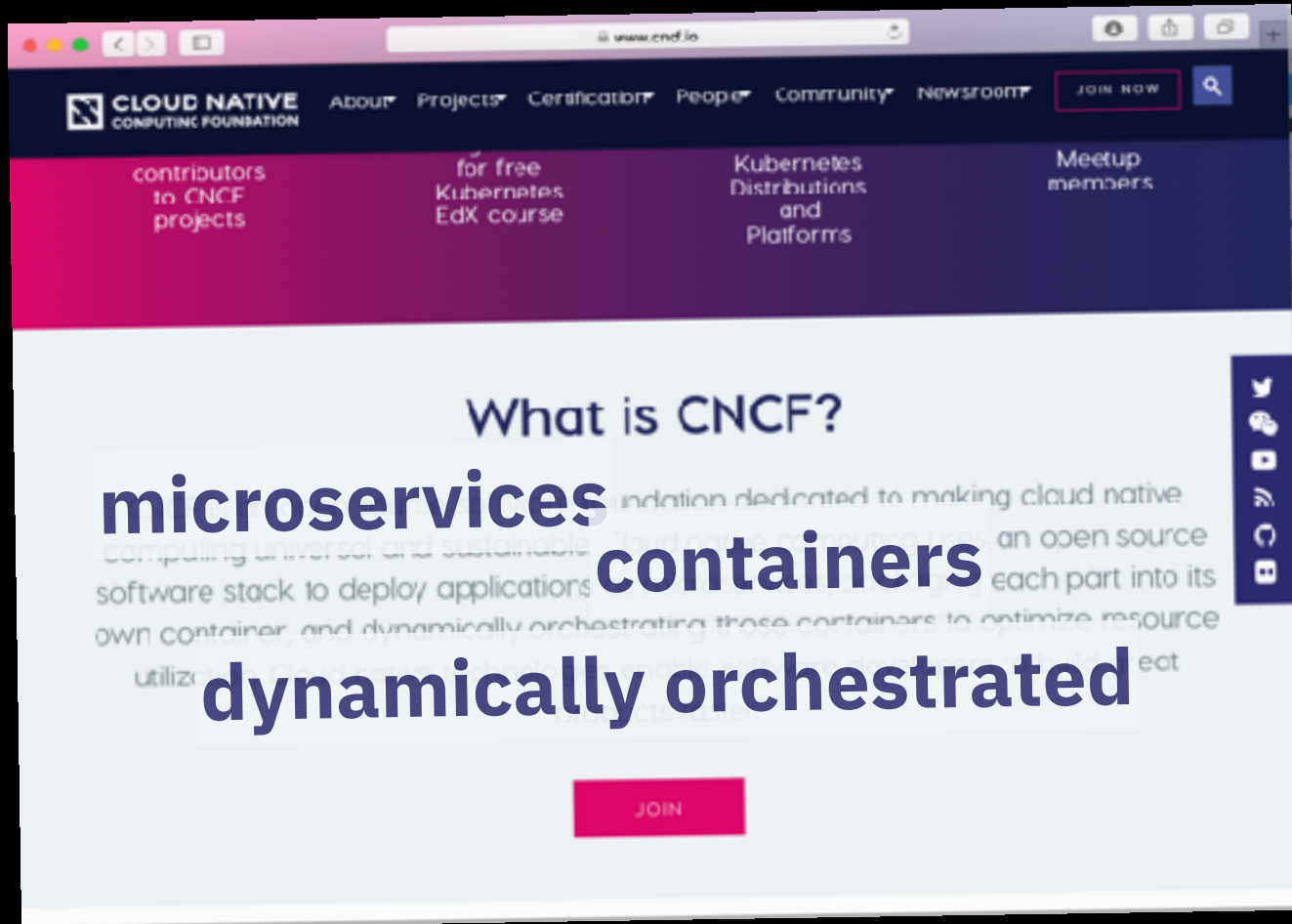
RELATED CONTENT

eBay Replatforming to Kubernetes, Envoy and Kafka: Intending to Open Source Hardware and Software Sep 18, 2018

Networking Your Microservices Applications Sep 21, 2018

Designing Events-First Microservices Sep 13, 2018





The logo for QCon, featuring a green 'Q' and blue 'Con' on a white rectangular background with a blue border.

QCon

"the cloud native
computing foundation
is **wrong** ...
about cloud native."



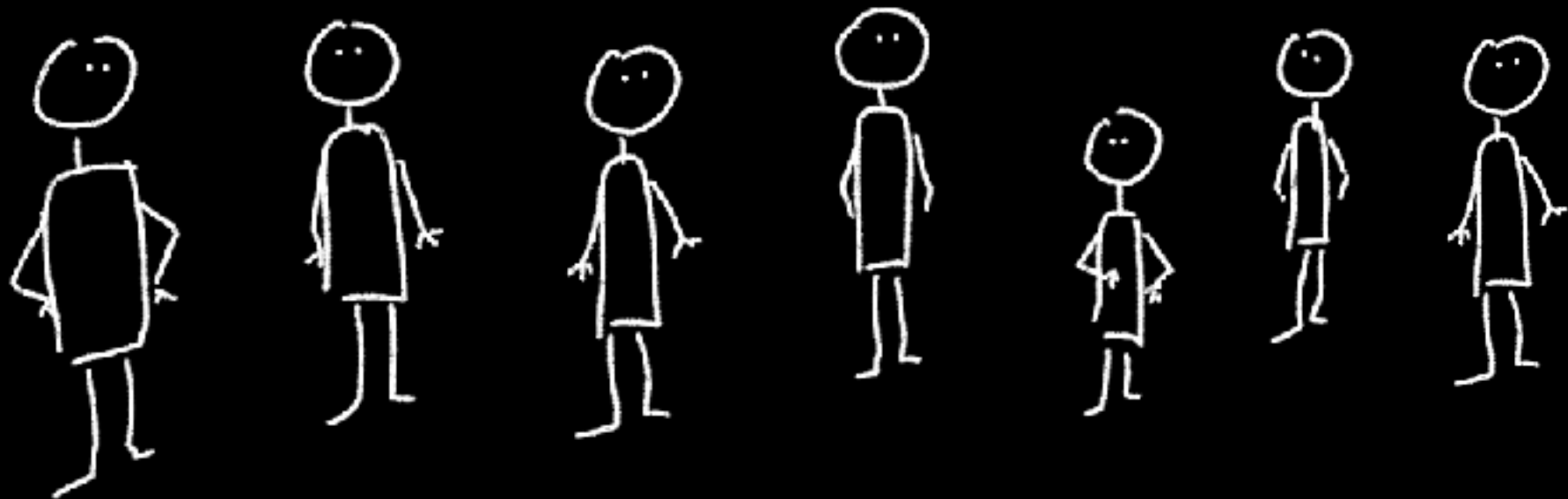
The logo for QCon, featuring a green 'Q' and blue 'Con' in a sans-serif font, set against a white rectangular background with a blue border.

QCon

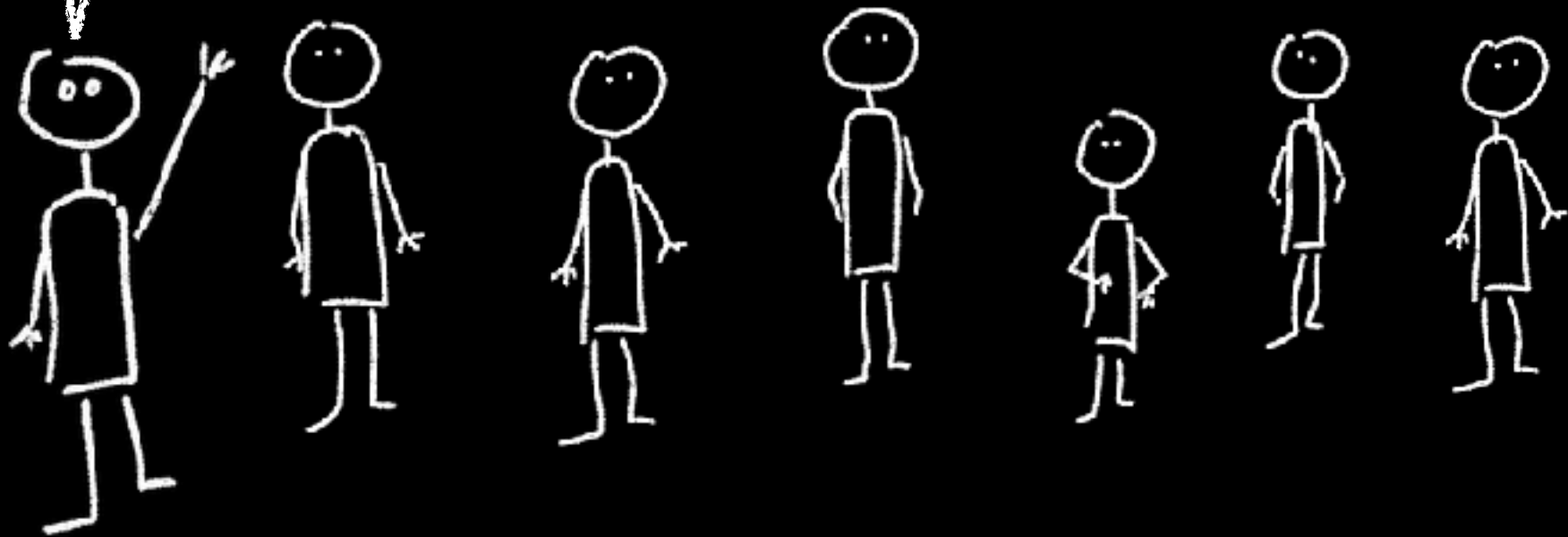
"the cloud native
computing foundation
is **wrong** ...
about cloud native."



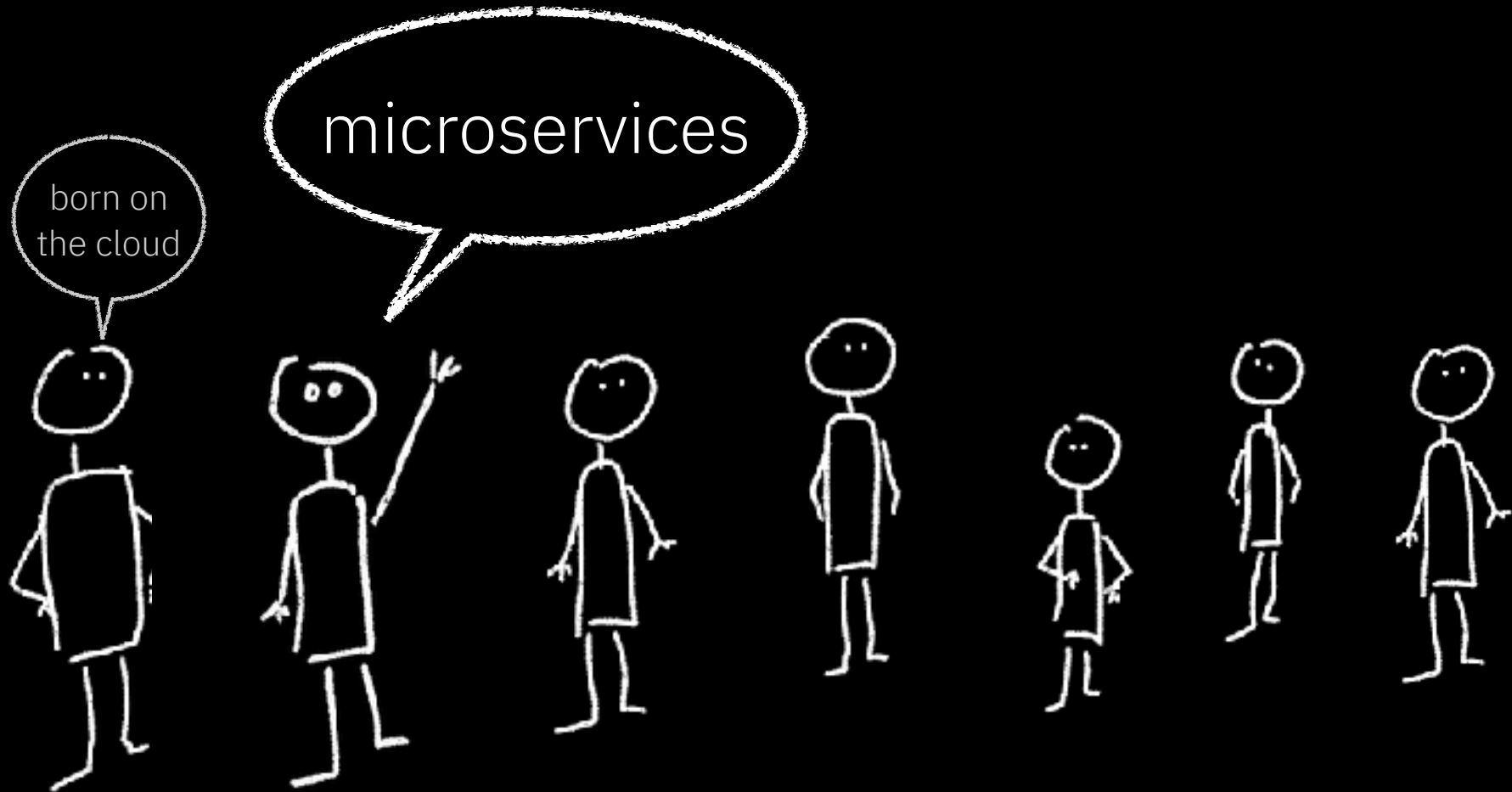
Dr Holly



born on
the cloud







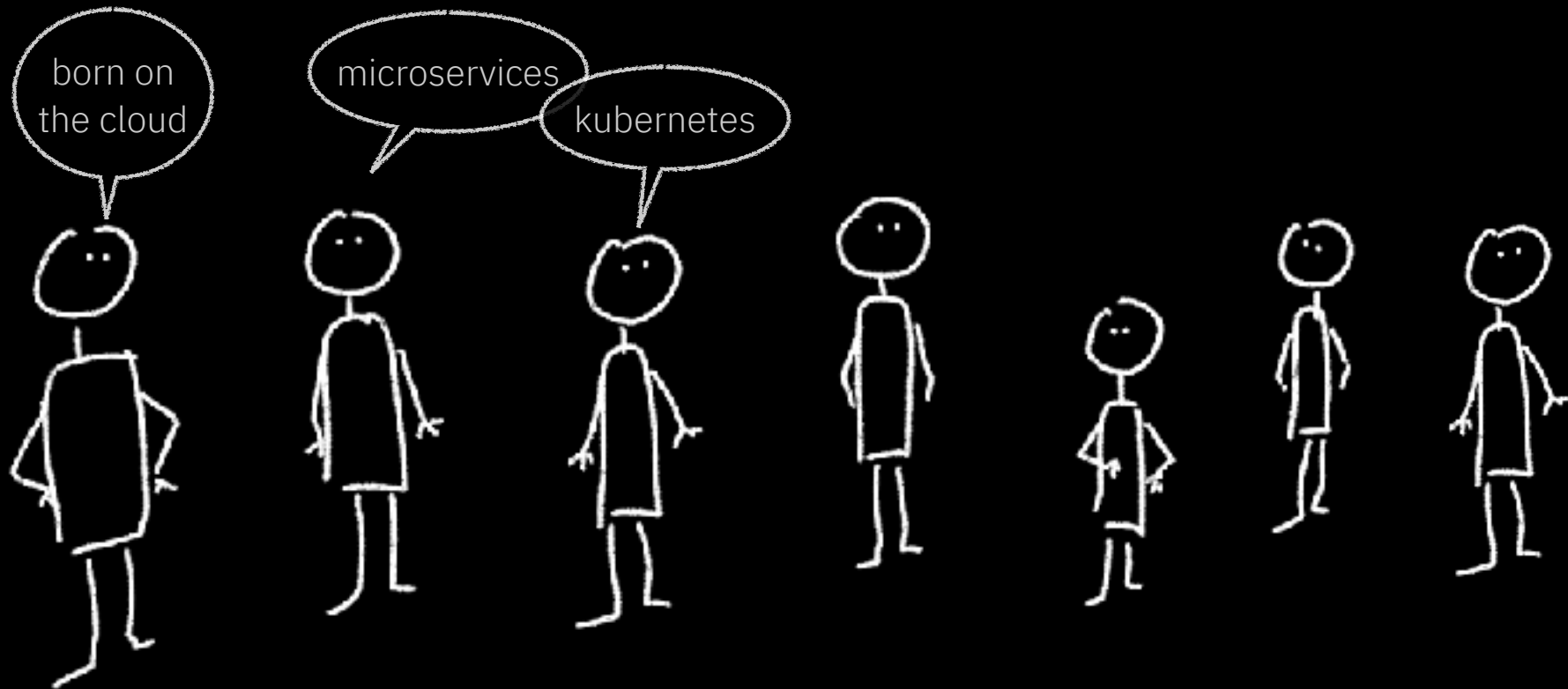


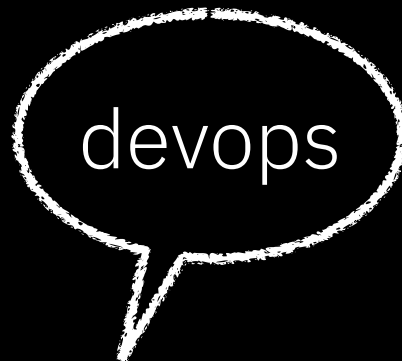


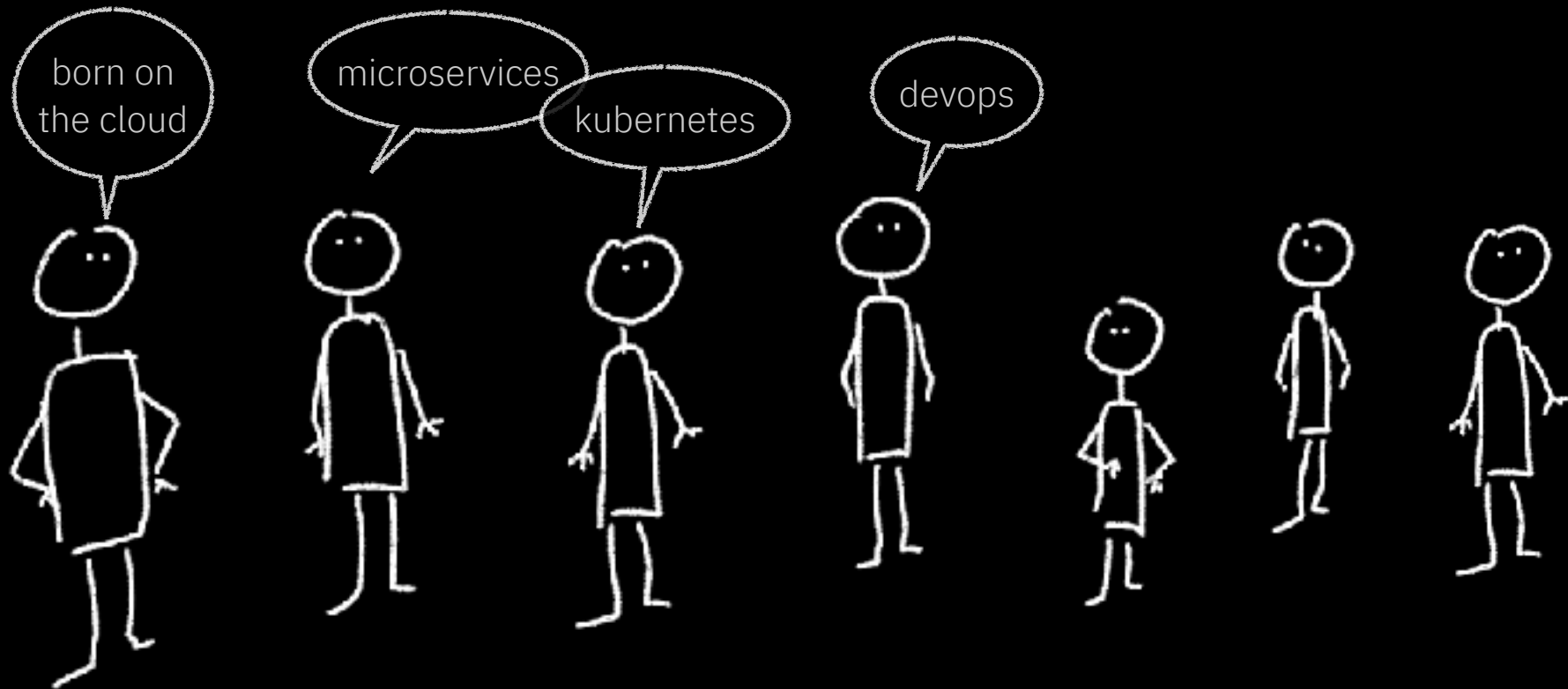
kubernetes

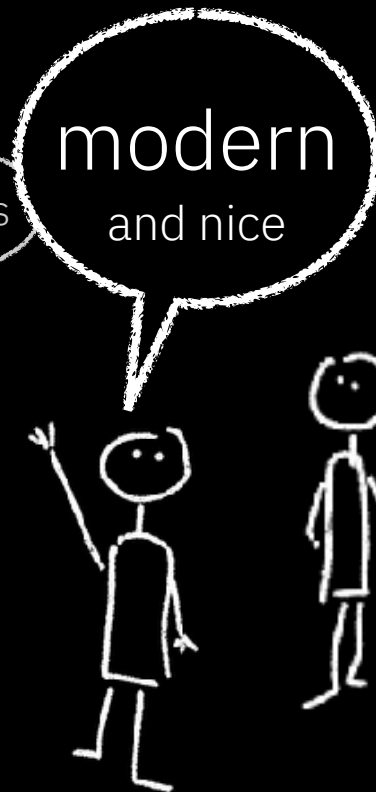
born on
the cloud

microservices

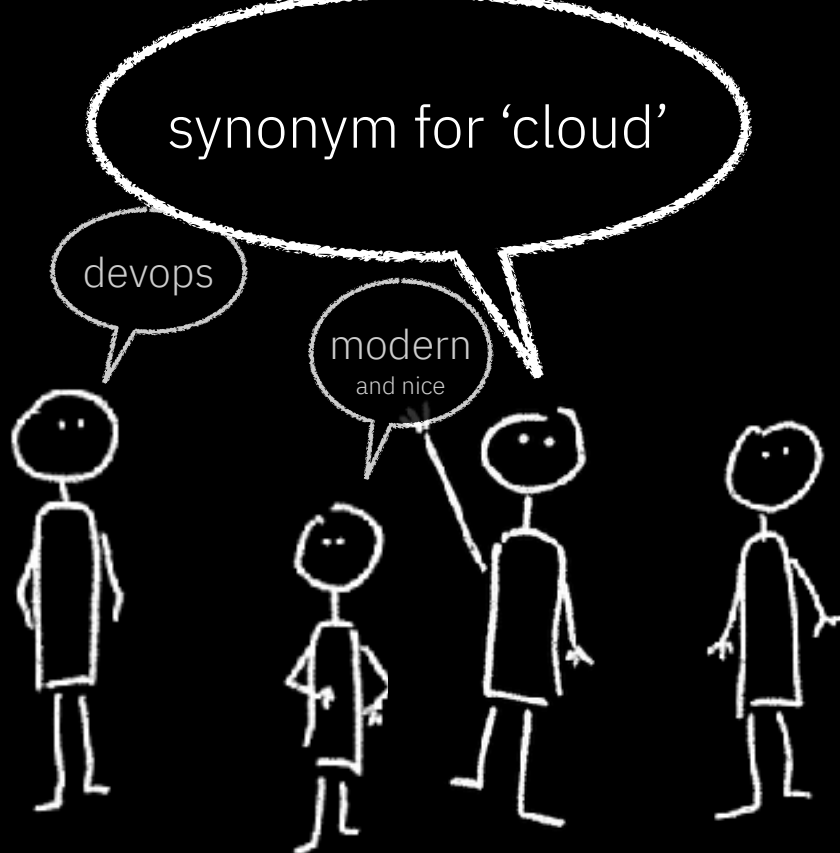


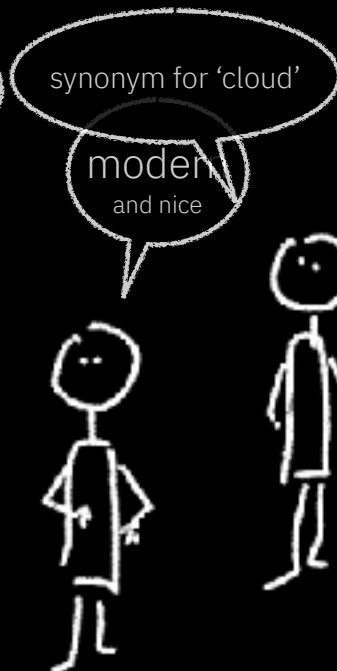


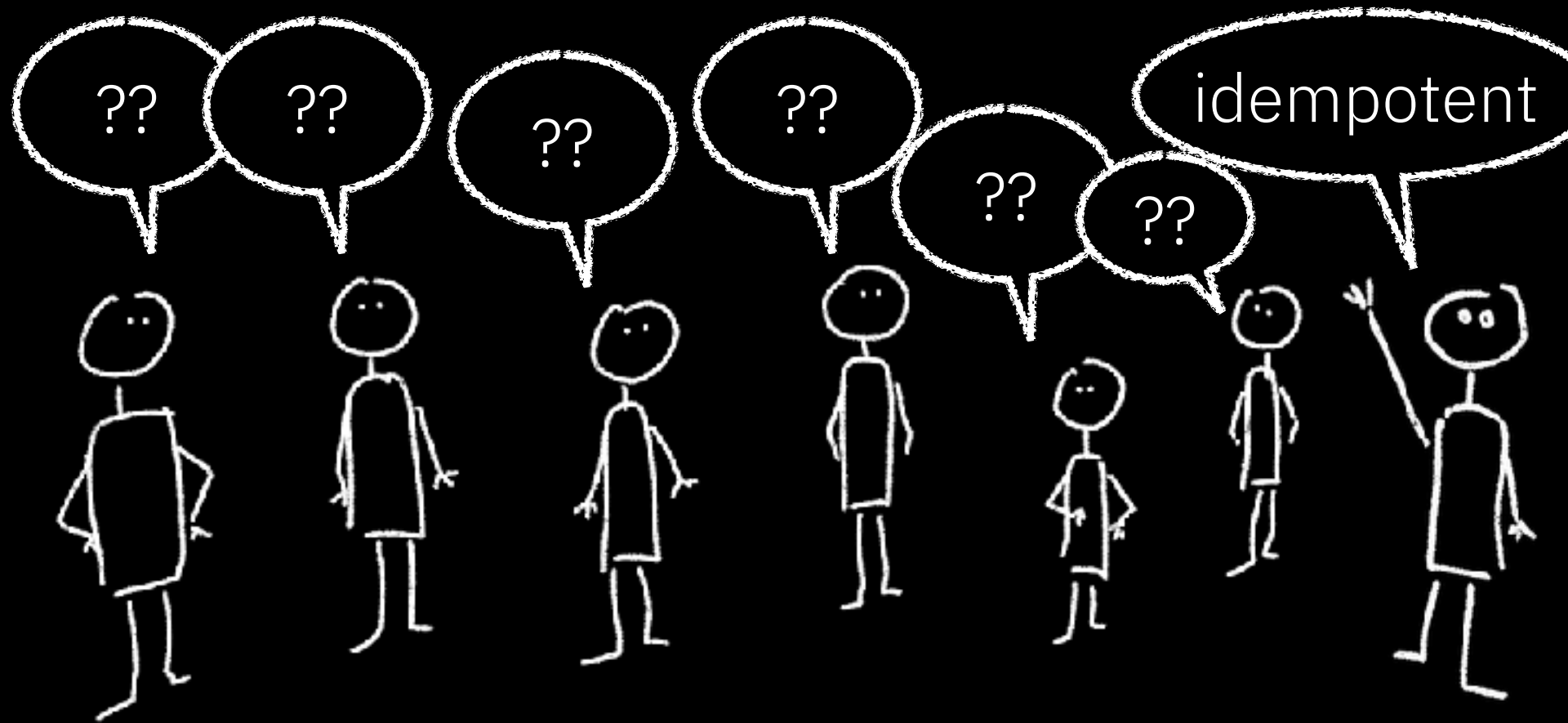


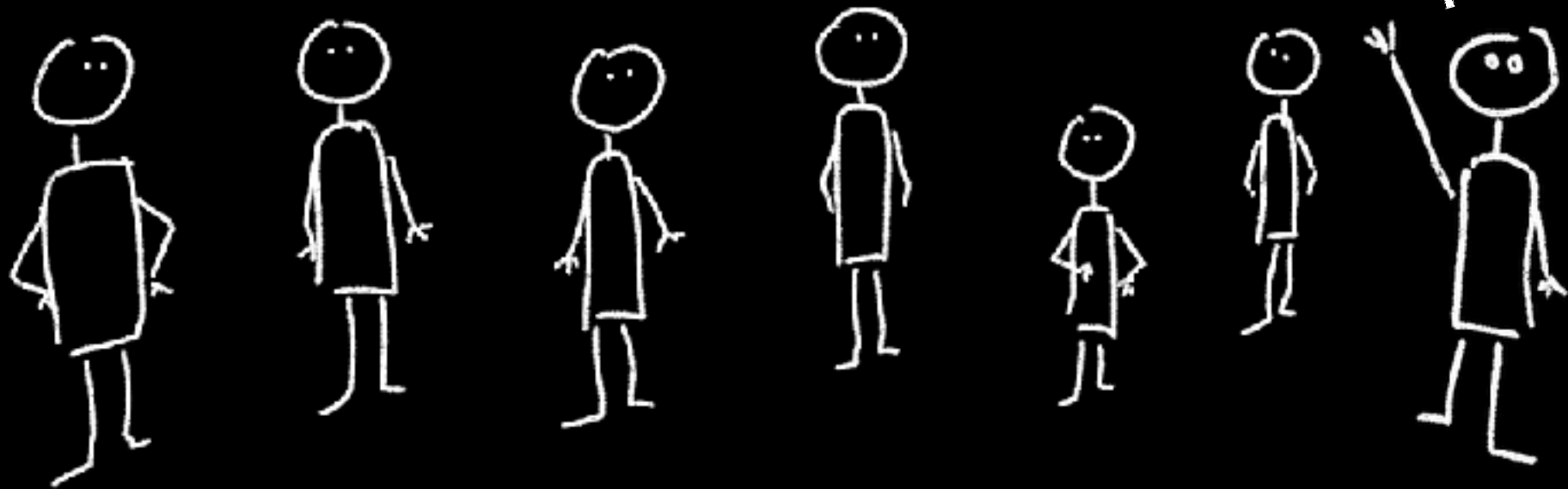


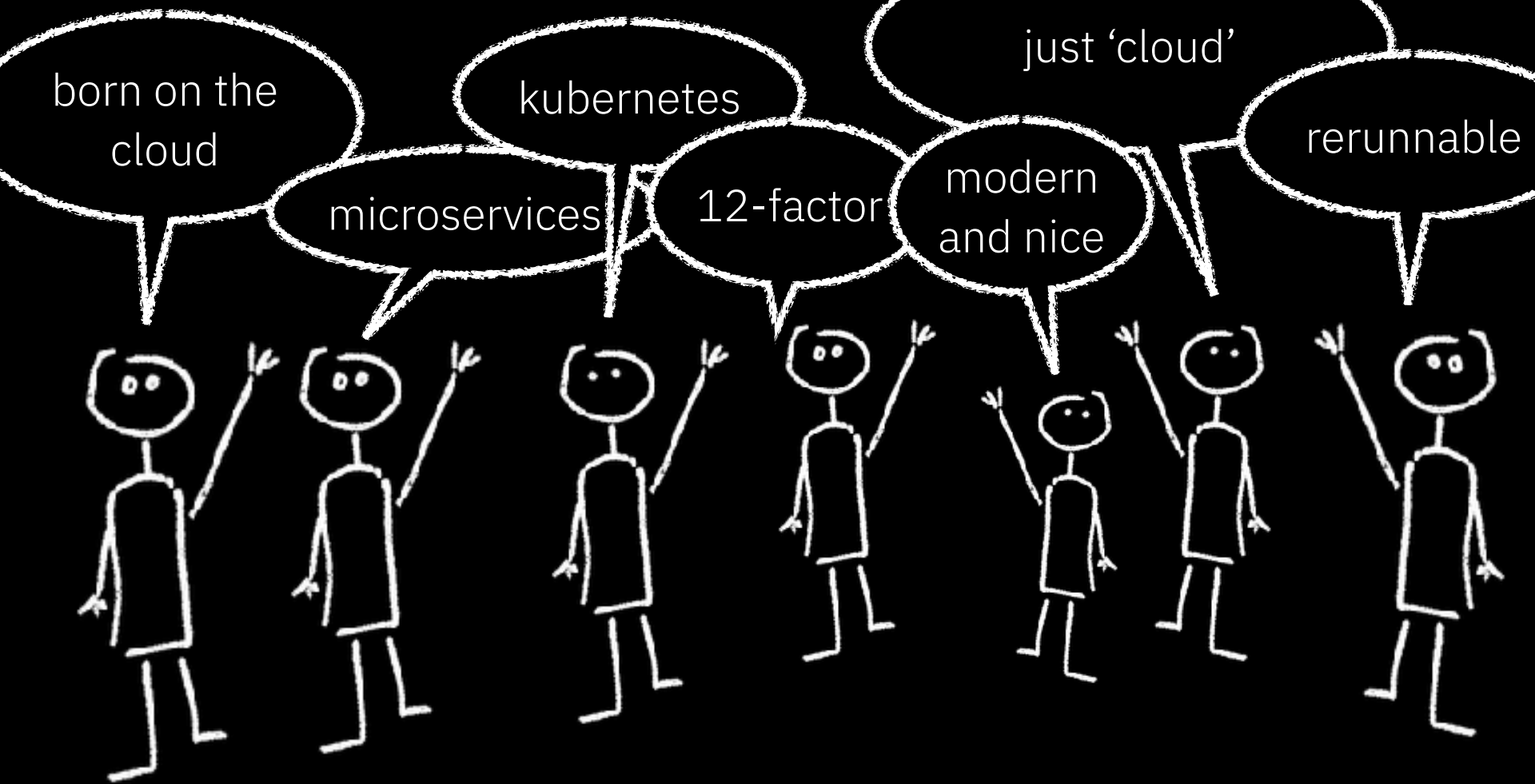




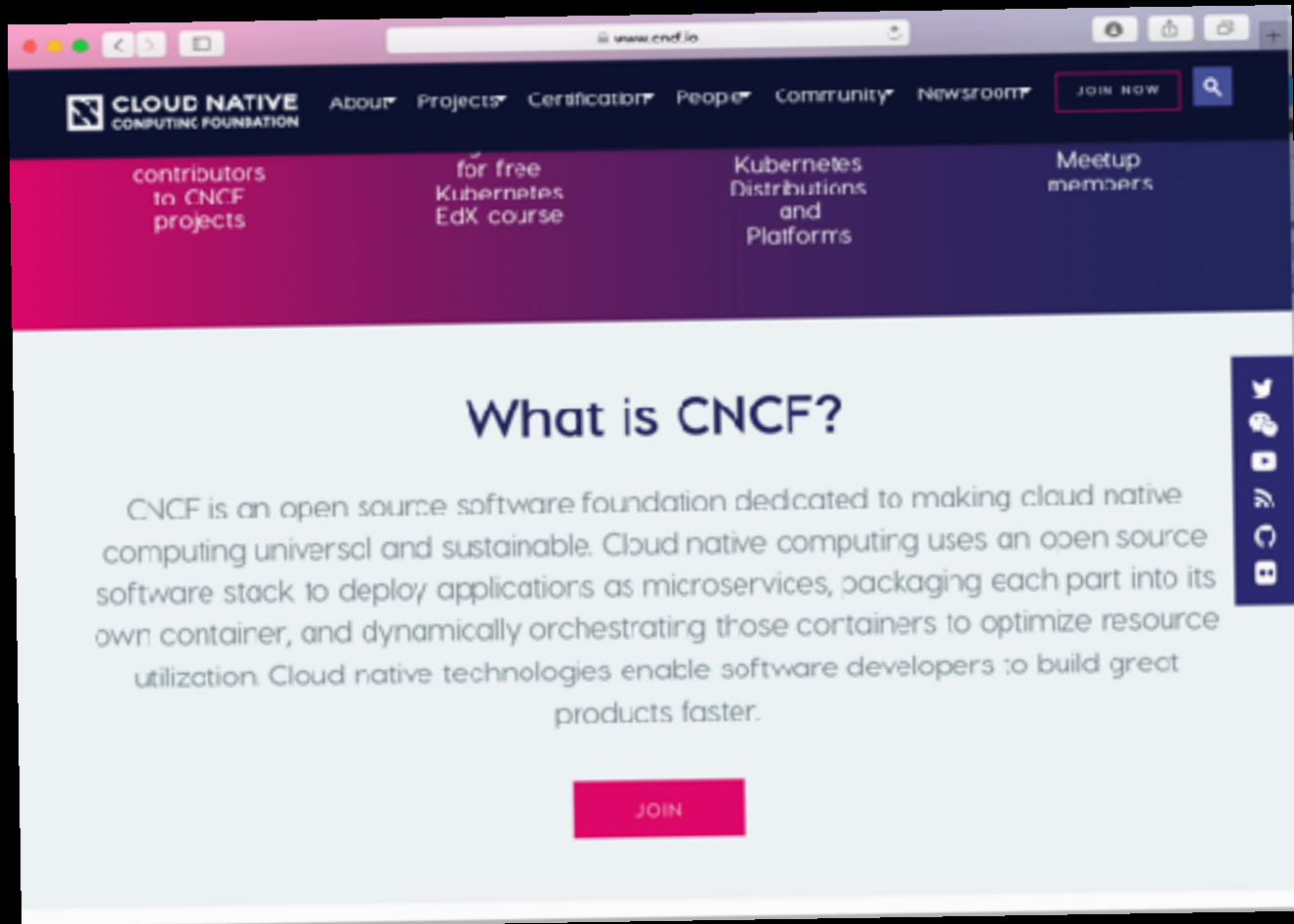


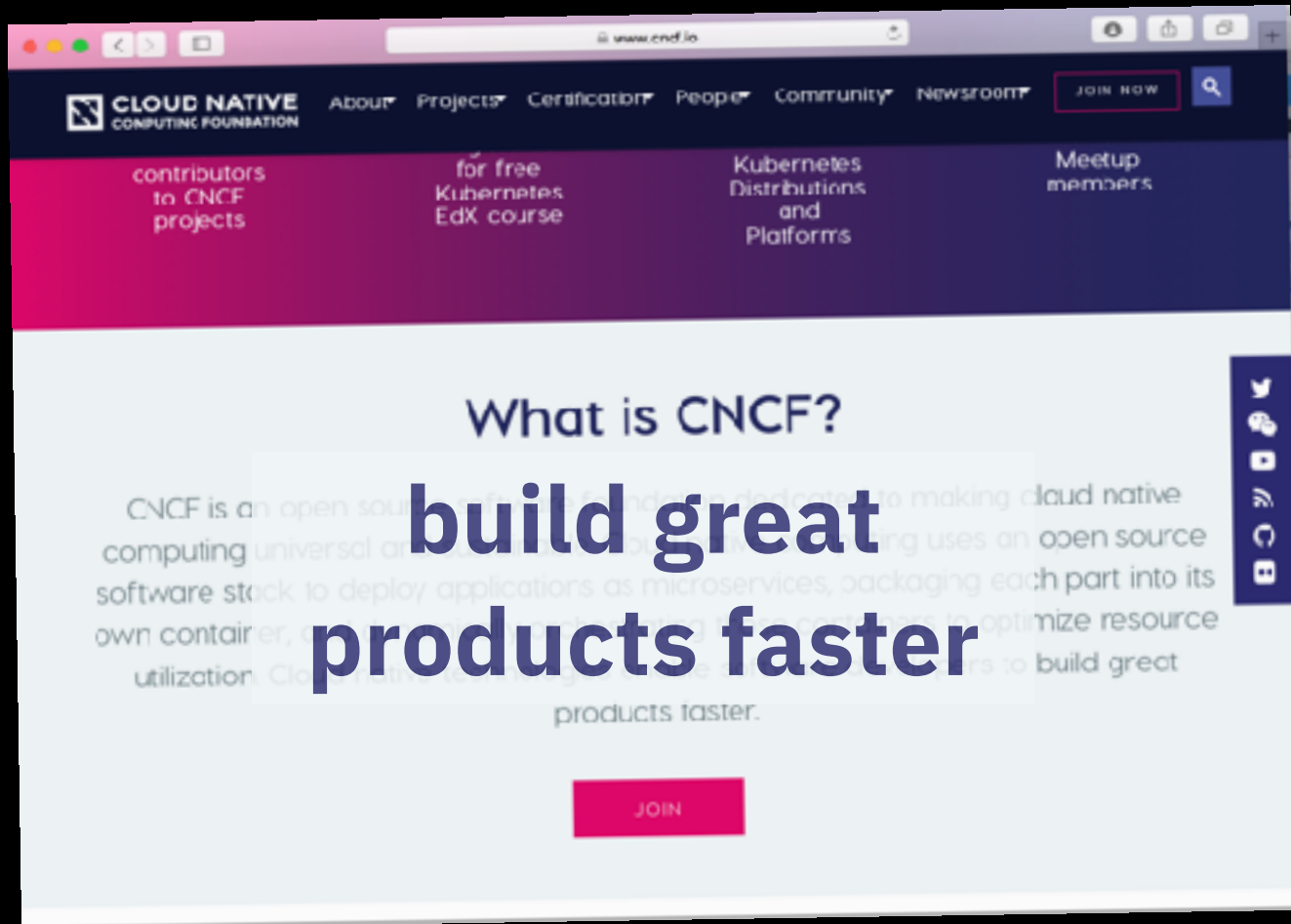






why?





what **problem** are
we trying to solve?

“everyone else
is doing it?”

why cloud?

cost





e l a s t i c i t y



exotic capabilities



why cloud native?



12
factors

12 factors

how to write a
cloud application
so you don't get
electrocuted

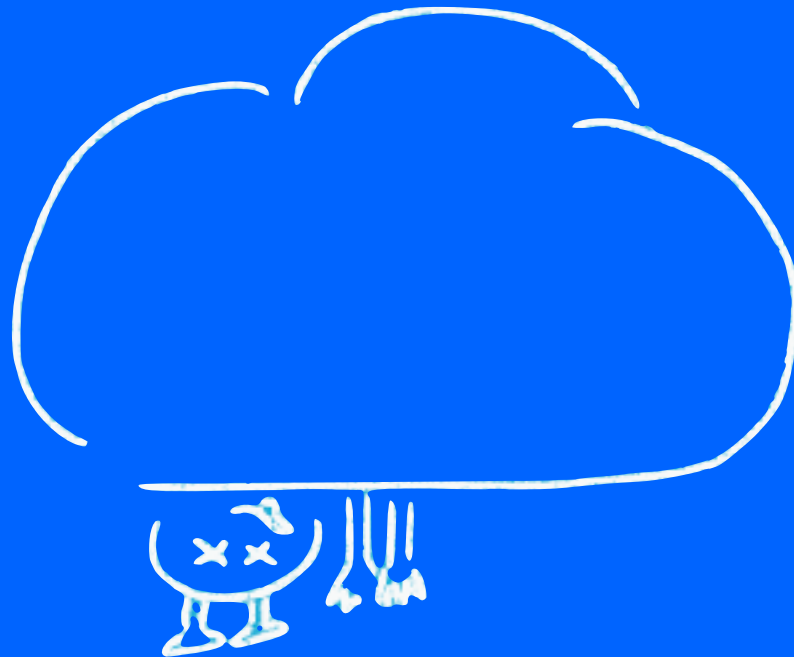
cloud native is not a
synonym for
‘microservices’

if 'cloud native' has to be a
synonym for anything, it would be
'idempotent'

if 'cloud native' has to be a
synonym for anything, it would be
'idempotent'

which definitely needs a synonym

how to fail at cloud native



I'm a consultant with the IBM Garage.

These are my scary stories



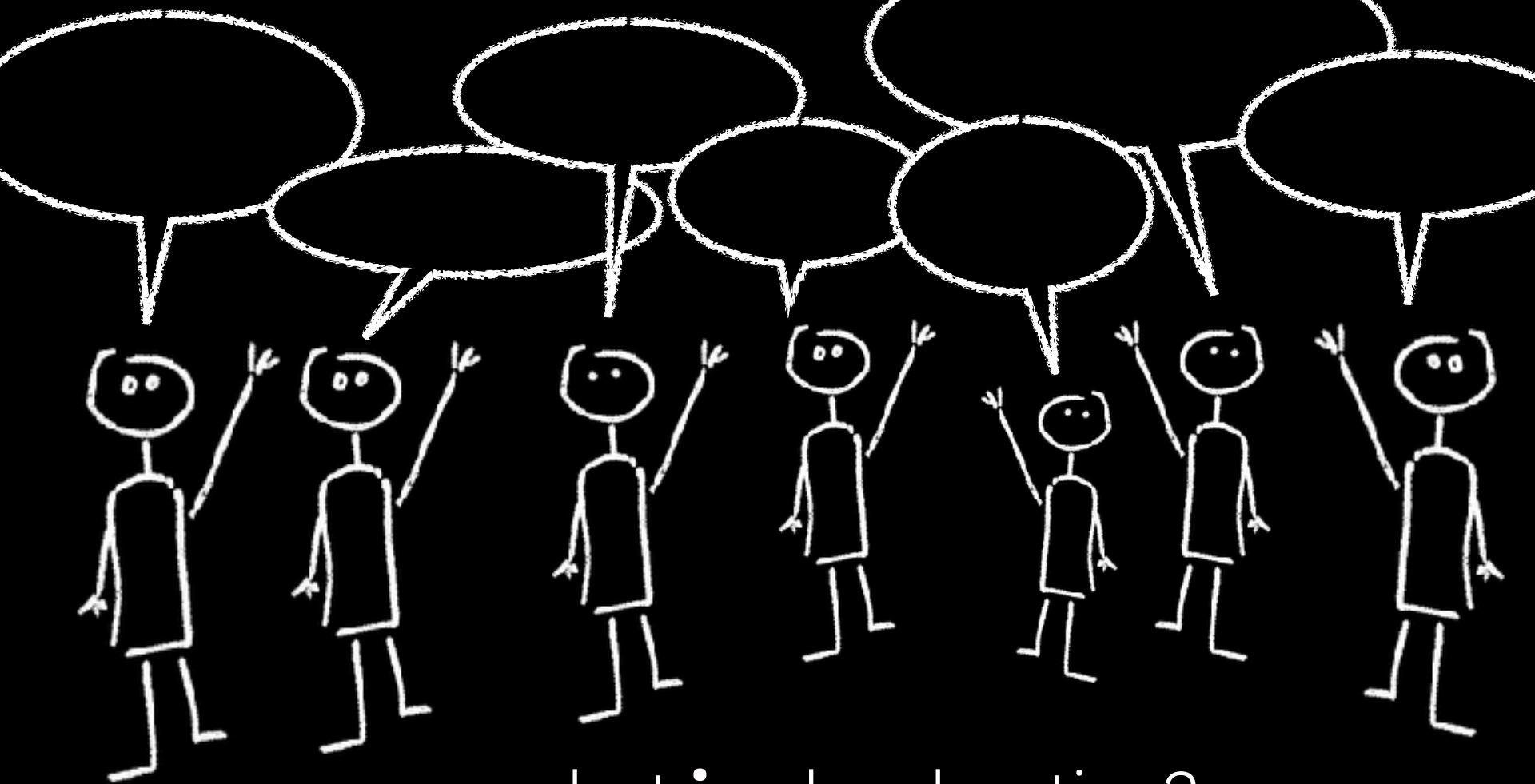
fail

the magic
morphing
meaning





so, what **is** cloud native?



so, what **is** cloud native?

why are
there no
microservices in
this cloud native
app Alice?

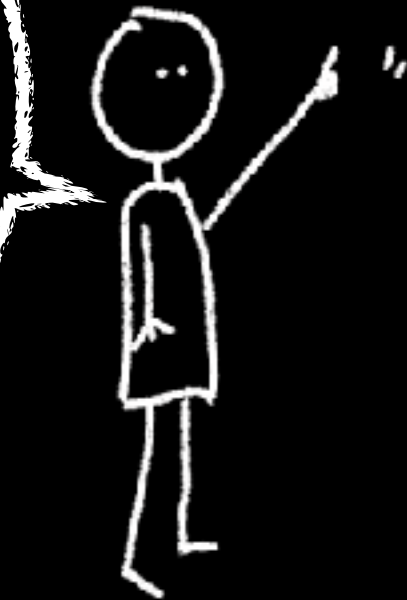


fail

the muddy
goal



why is the
cloud **only**
saving us
money, Alice?



fail



microservices
envy



microservices
are not the **goal**

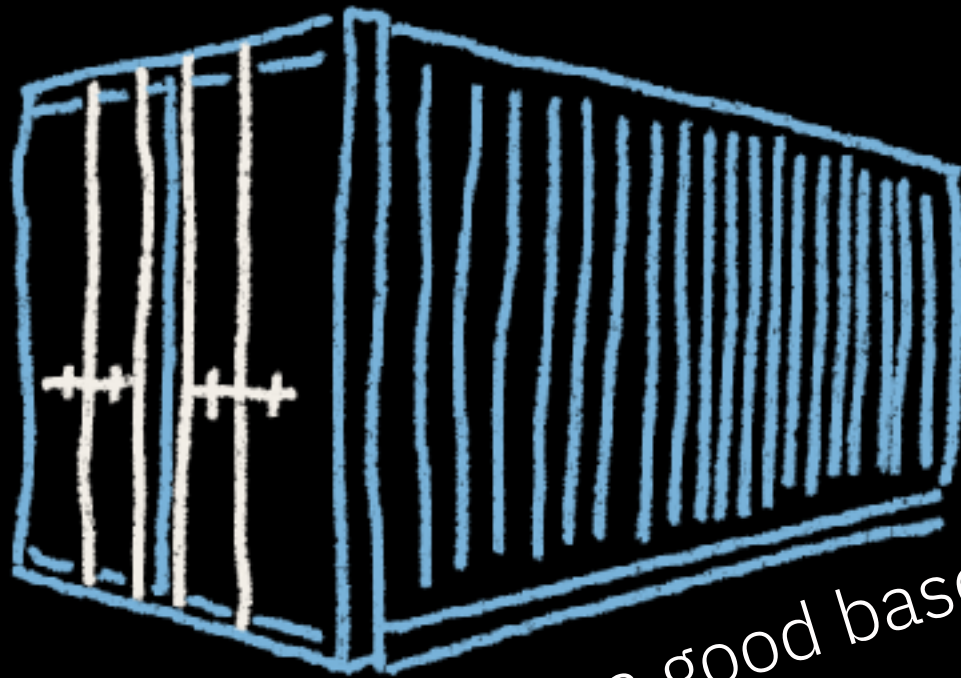
microservices
are not the **goal**

they are the **means**

“we’re going too slowly.
we need to get rid of COBOL
and make microservices!”

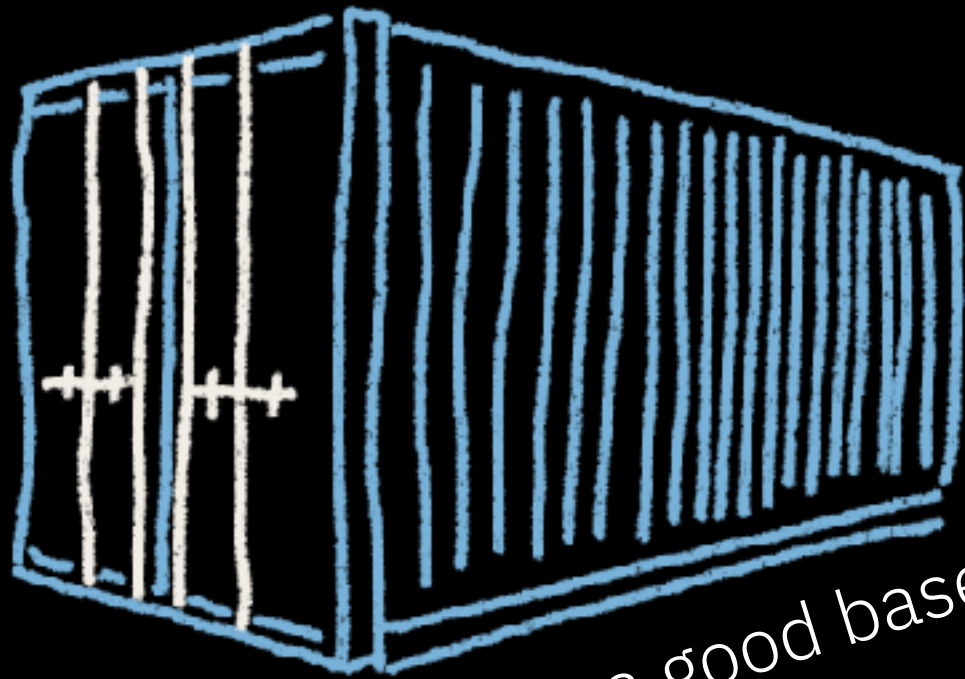
“we’re going too slowly.
we need to get rid of COBOL
and make microservices!”

“... but our release board
only meets twice a year.”



containers are a good base

it's not a
competition
to see how
many you
can have



containers are a good base

distributed monolith

distributed monolith

but without compile-time checking
... or guaranteed function execution

reasons **not** to do microservices

small team

not planning to release independently

don't want complexity of a service mesh - or
worse yet, rolling your own

domain model doesn't split nicely

fail

cloud-native
spaghetti



“every time we change one
microservice, another breaks”

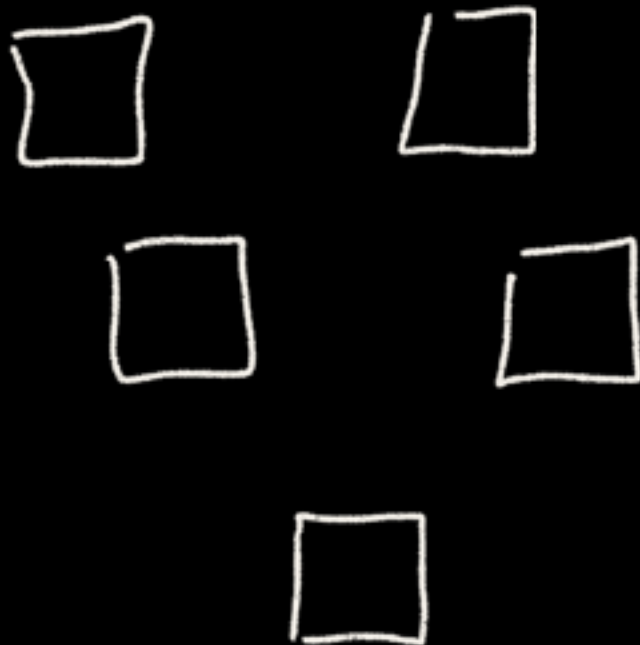
distributed \neq decoupled



cloud-native spaghetti is still spaghetti

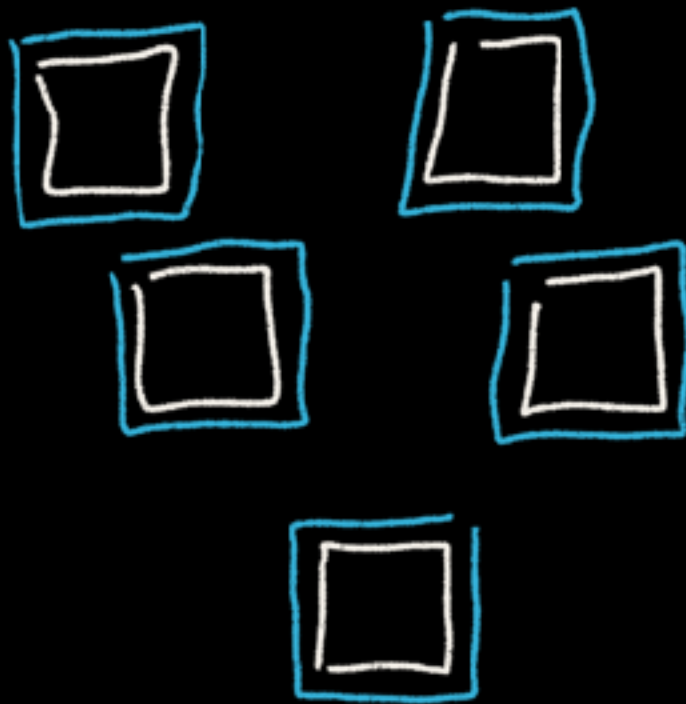
(Image: Cloudy with a Chance of Meatballs.)

“each of our microservices has
duplicated the same object model ...
with twenty classes and seventy fields”



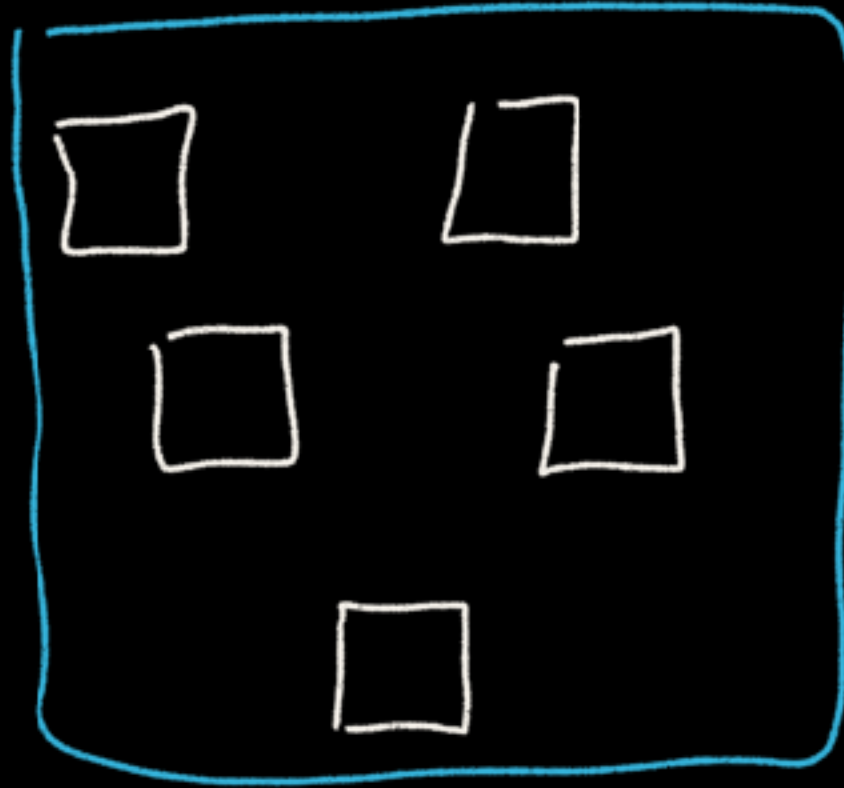
Microservice

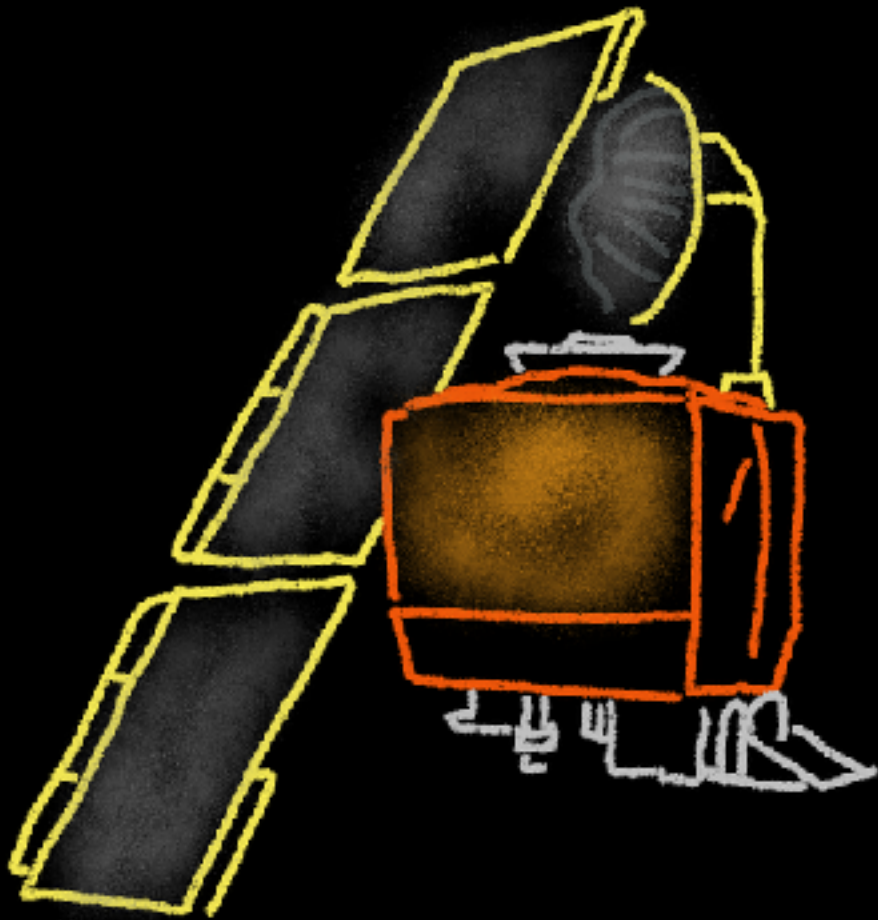
Domain

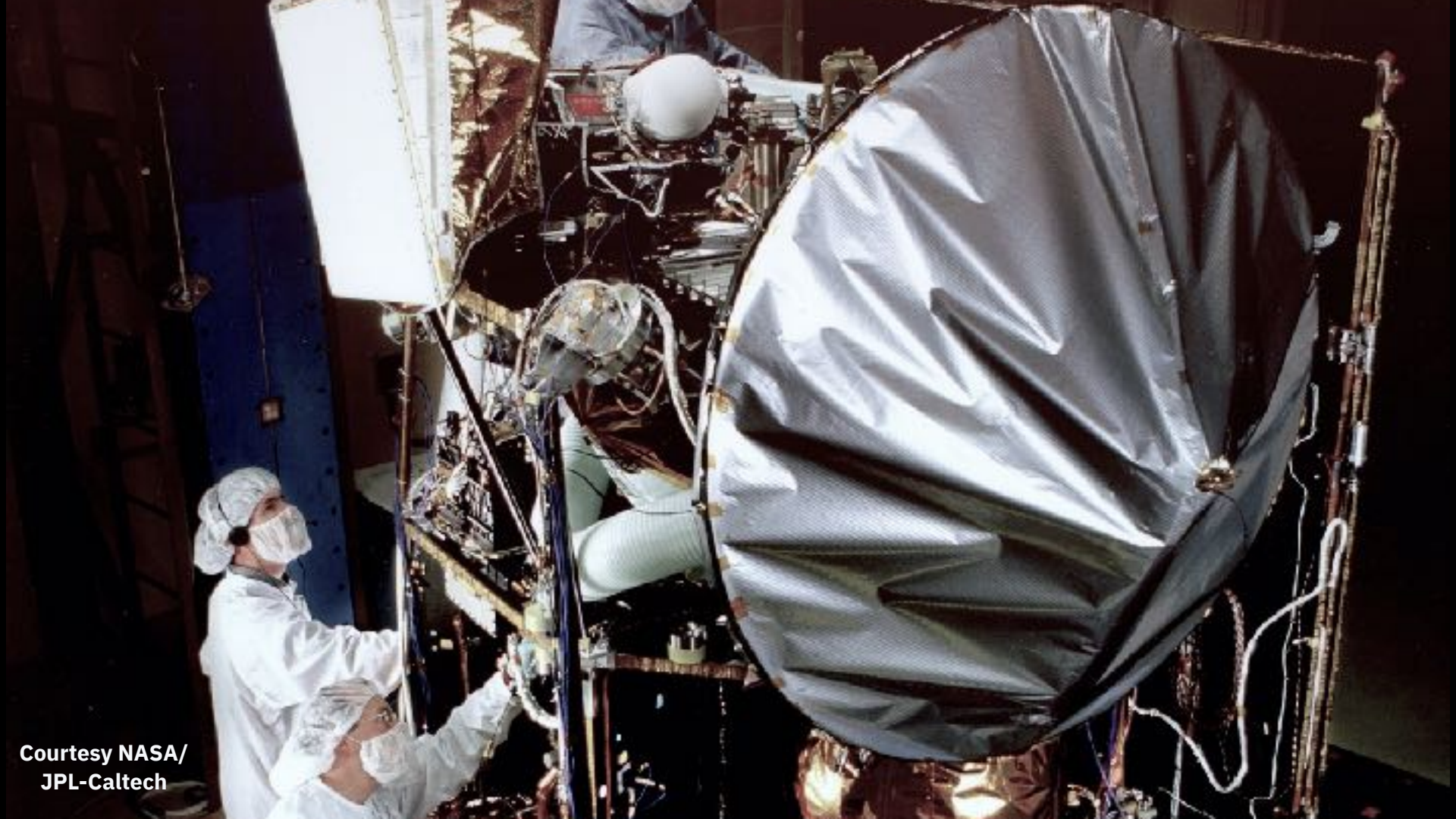


Microservice

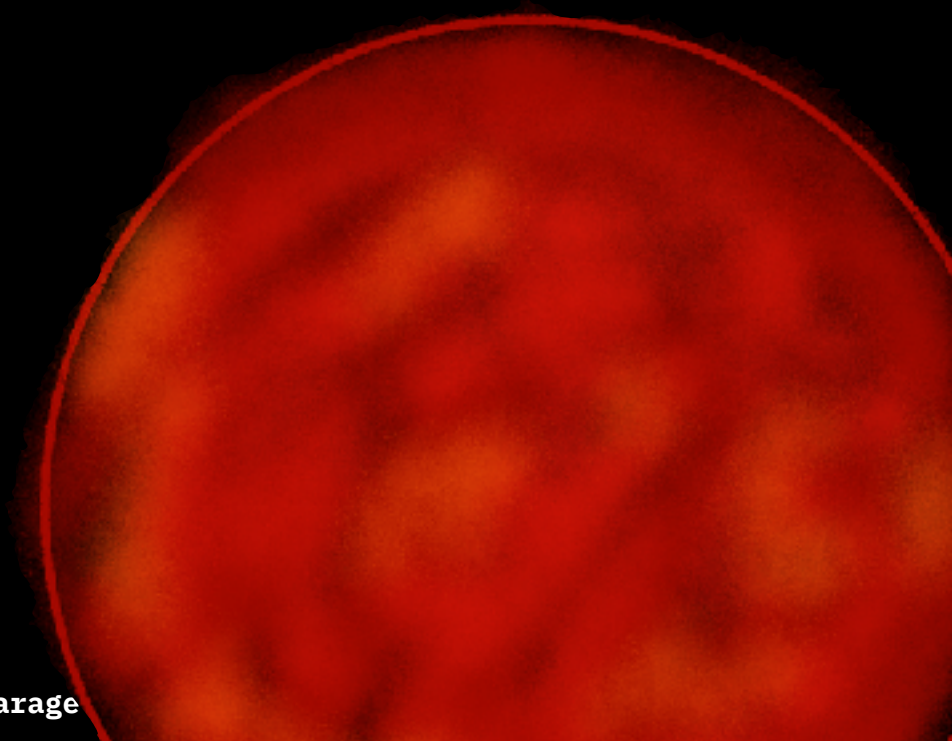
Domain

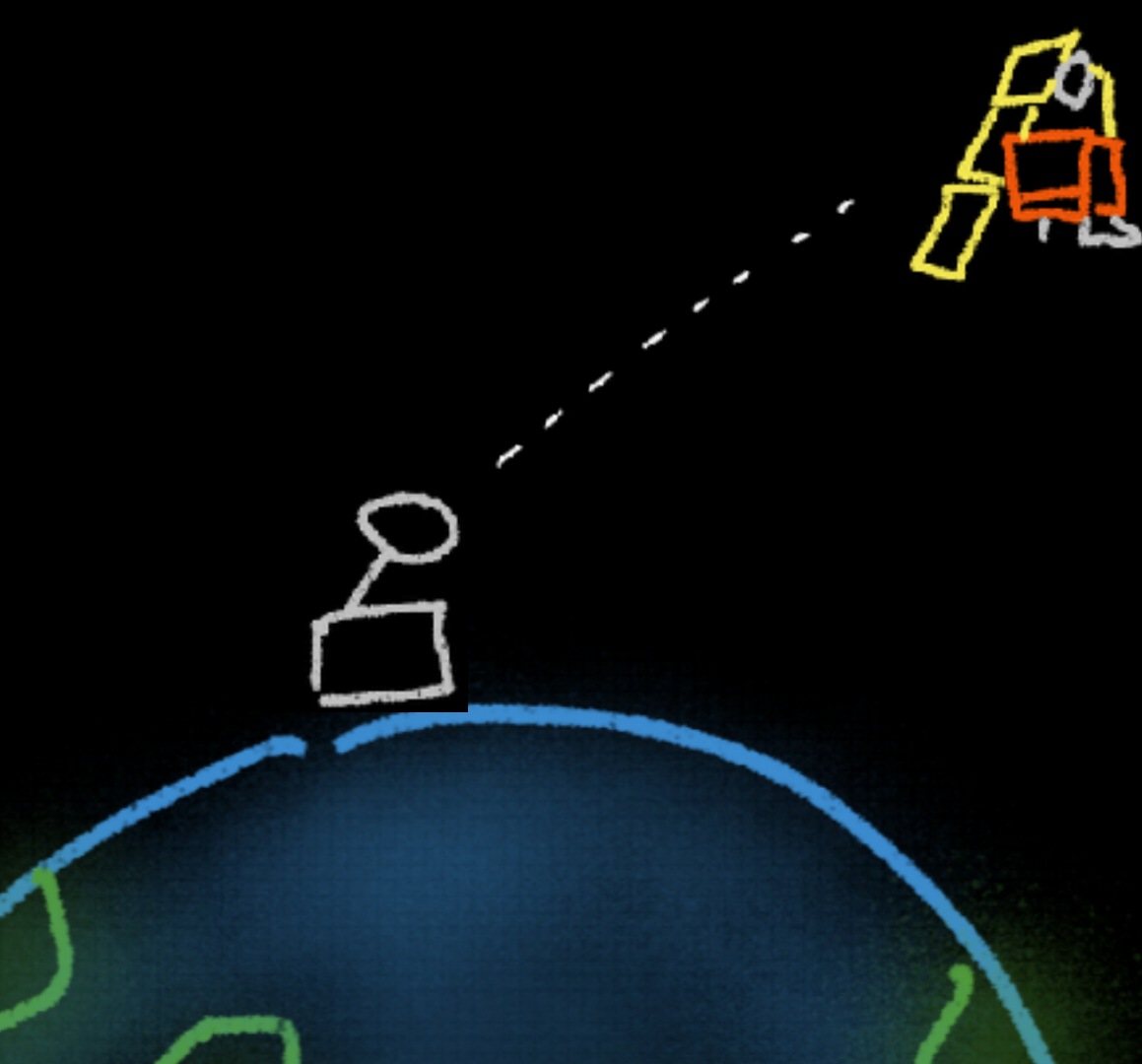


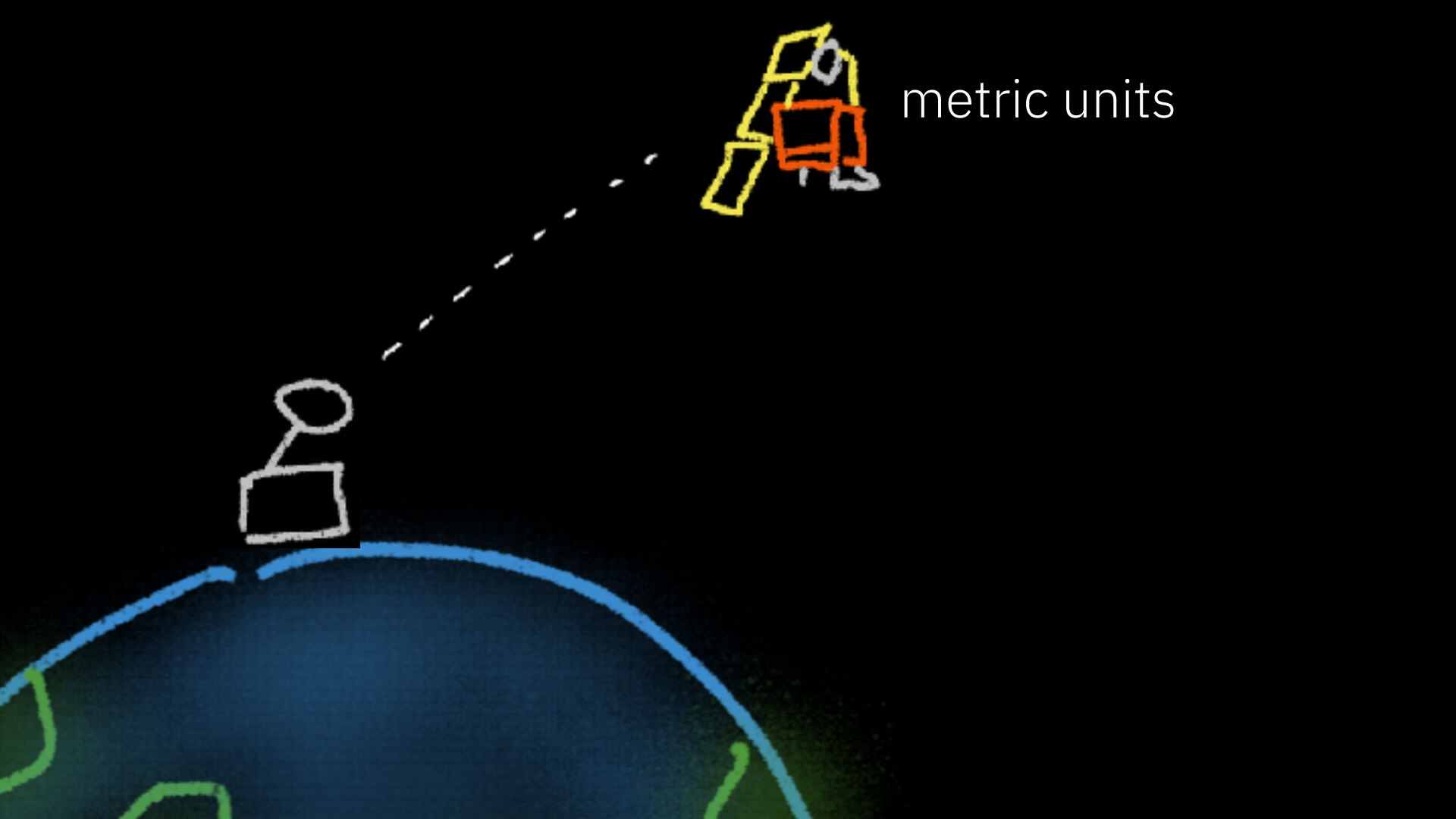




Courtesy NASA/
JPL-Caltech







metric units

imperial
units



metric units

imperial
units



metric units

distributing
did not help

microservices **need**
consumer-driven contract tests

fail

the not-actually-continuous
continuous integration and
continuous deployment



“we have a CI/CD”

CI/CD is something you **do**
not a tool you buy

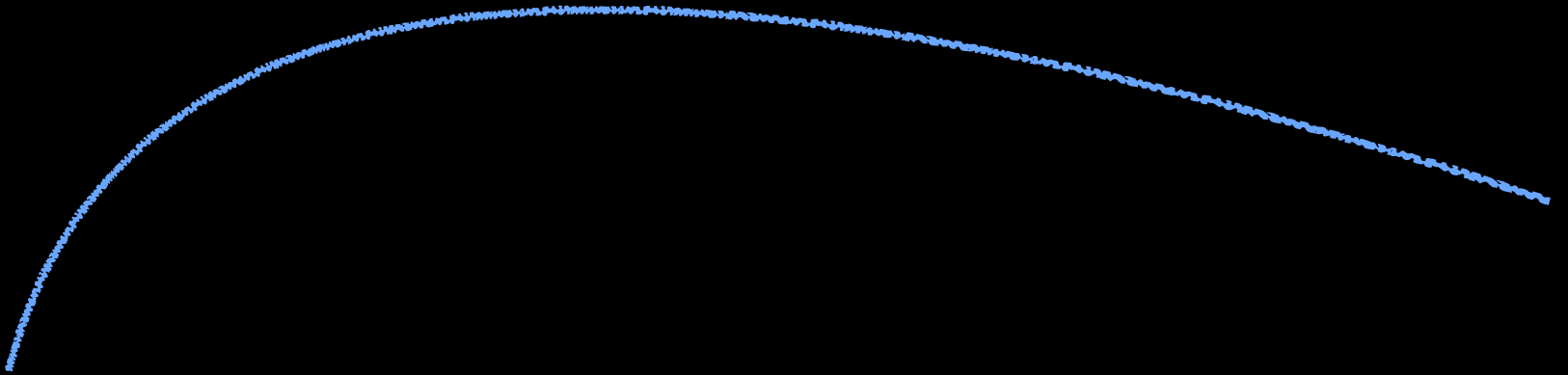
“i’ll merge my branch
into our CI next week”

“CI/CD ... CI/CD ... CI/CD ...
we release every six months ...
CI/CD”

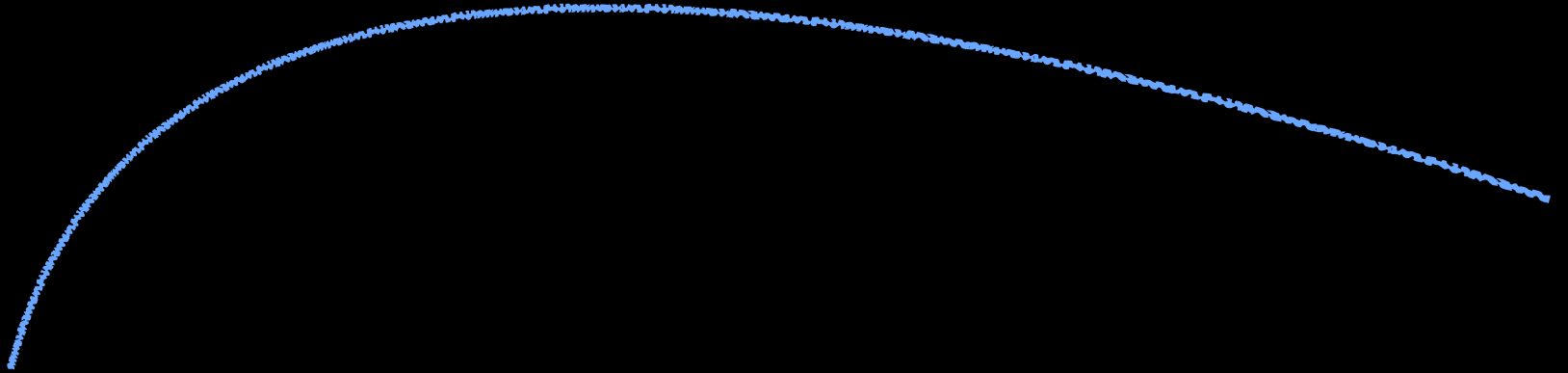
continuous.

I don't think that word means
what you think it means.

how often should you push to master?

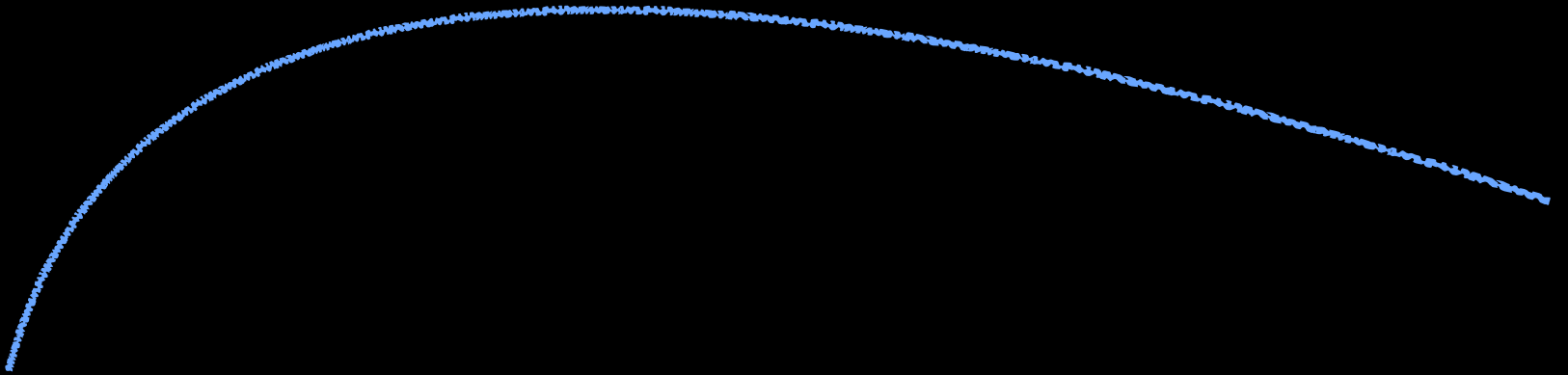


how often should you integrate?



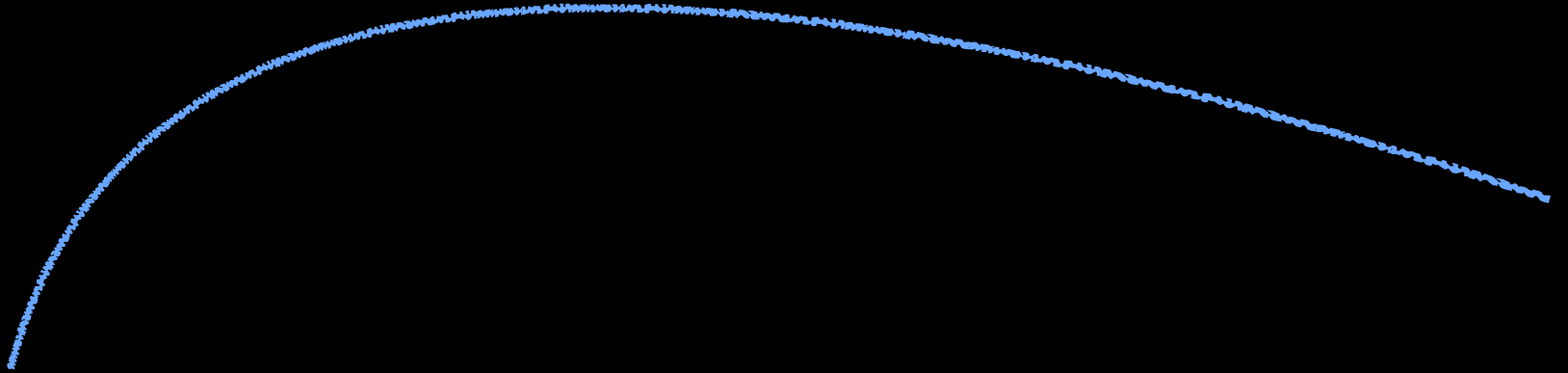
how often should you integrate?

every character



how often should you integrate?

every character



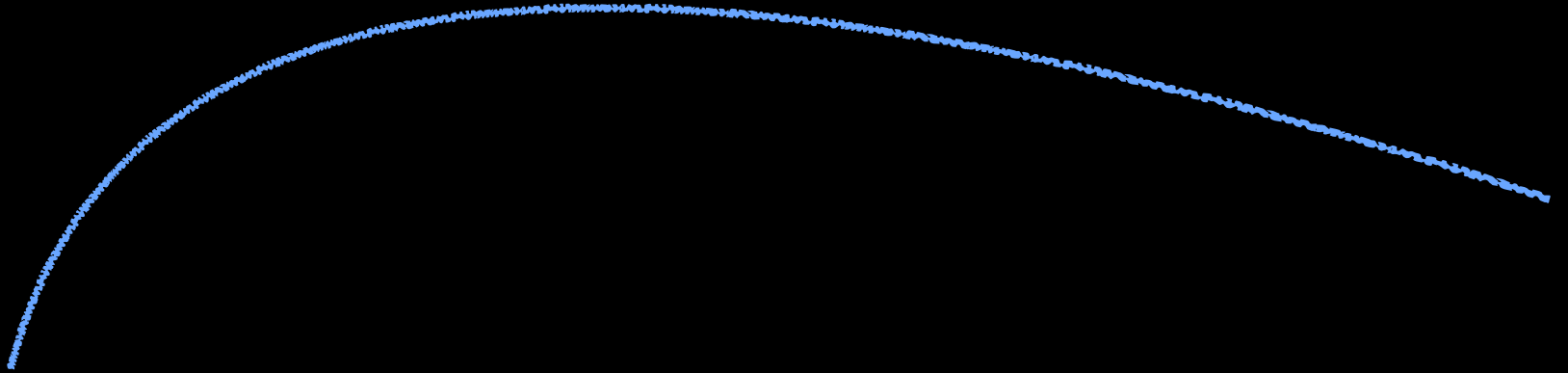
actually continuous
... but stupid

how often should you integrate?

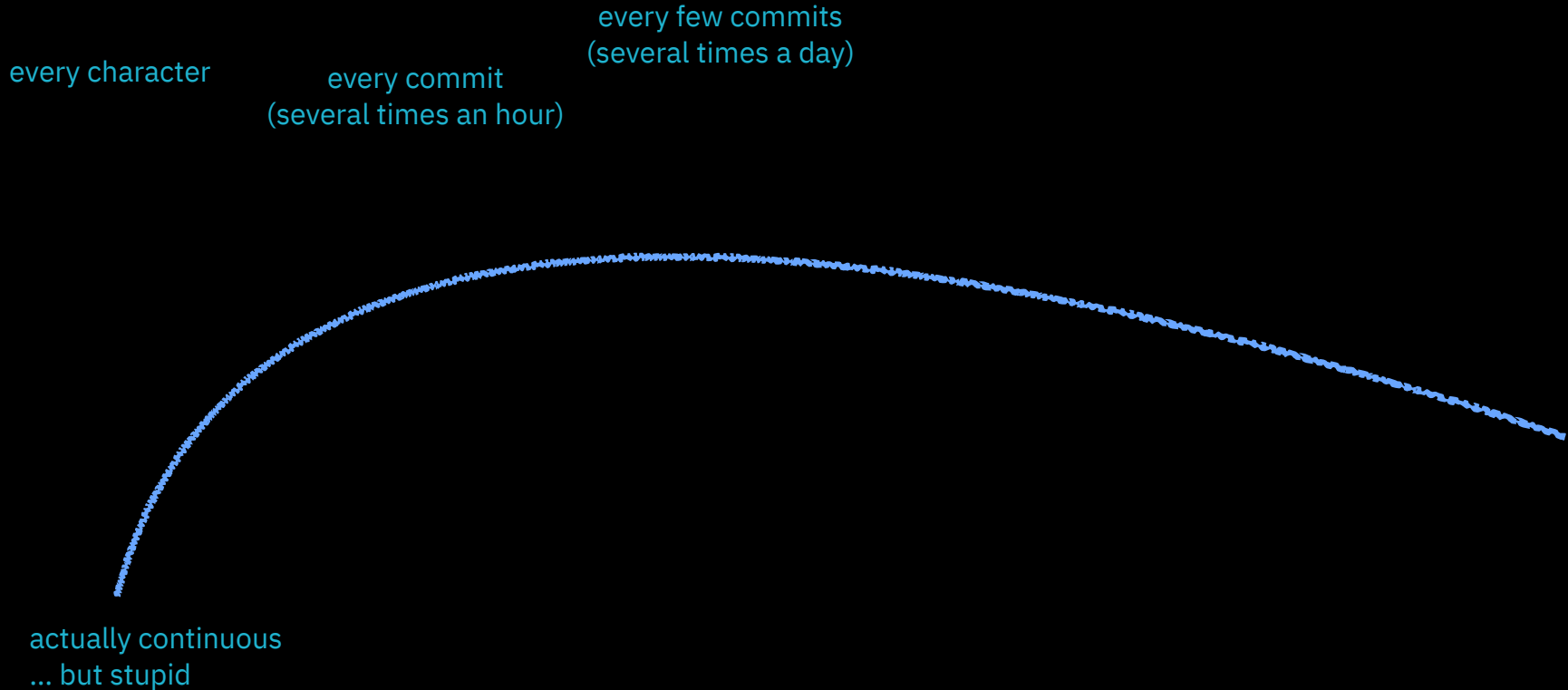
every character

every commit
(several times an hour)

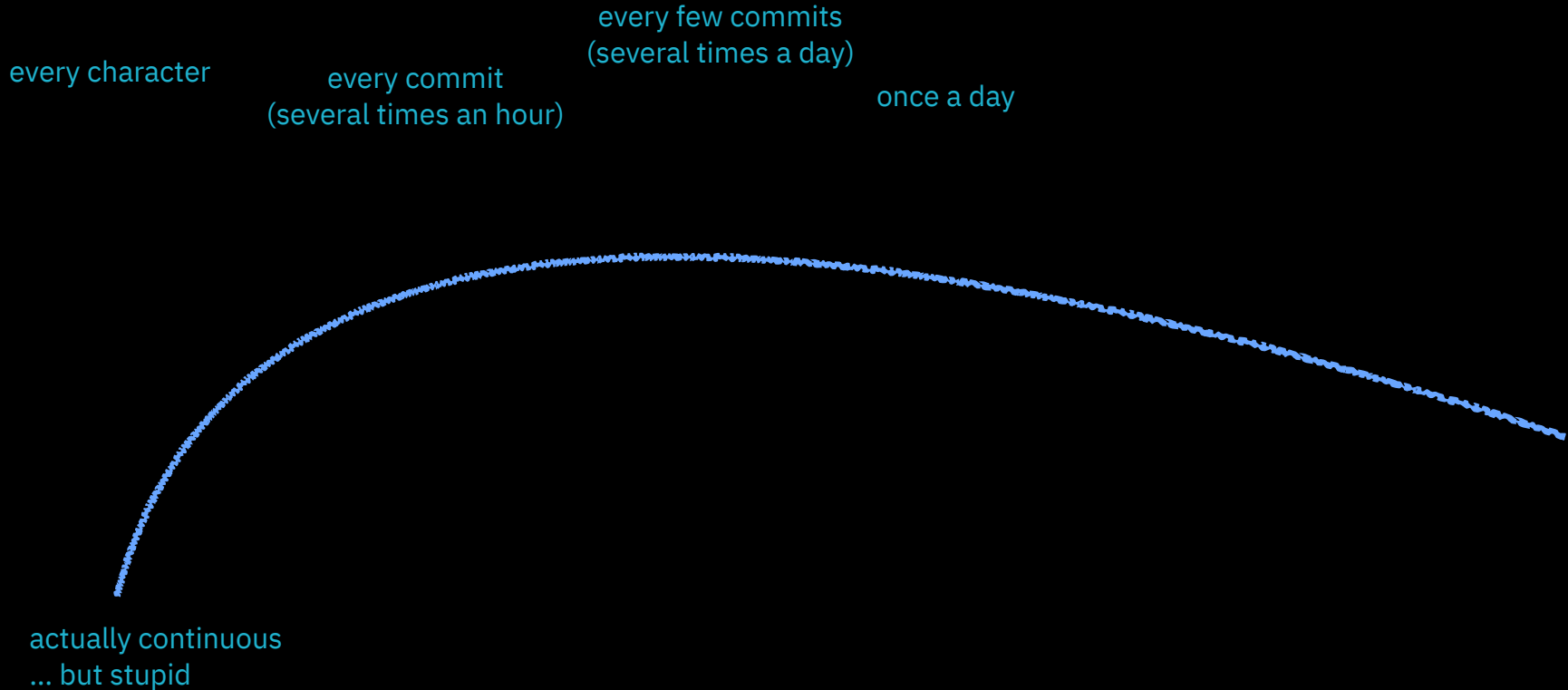
actually continuous
... but stupid



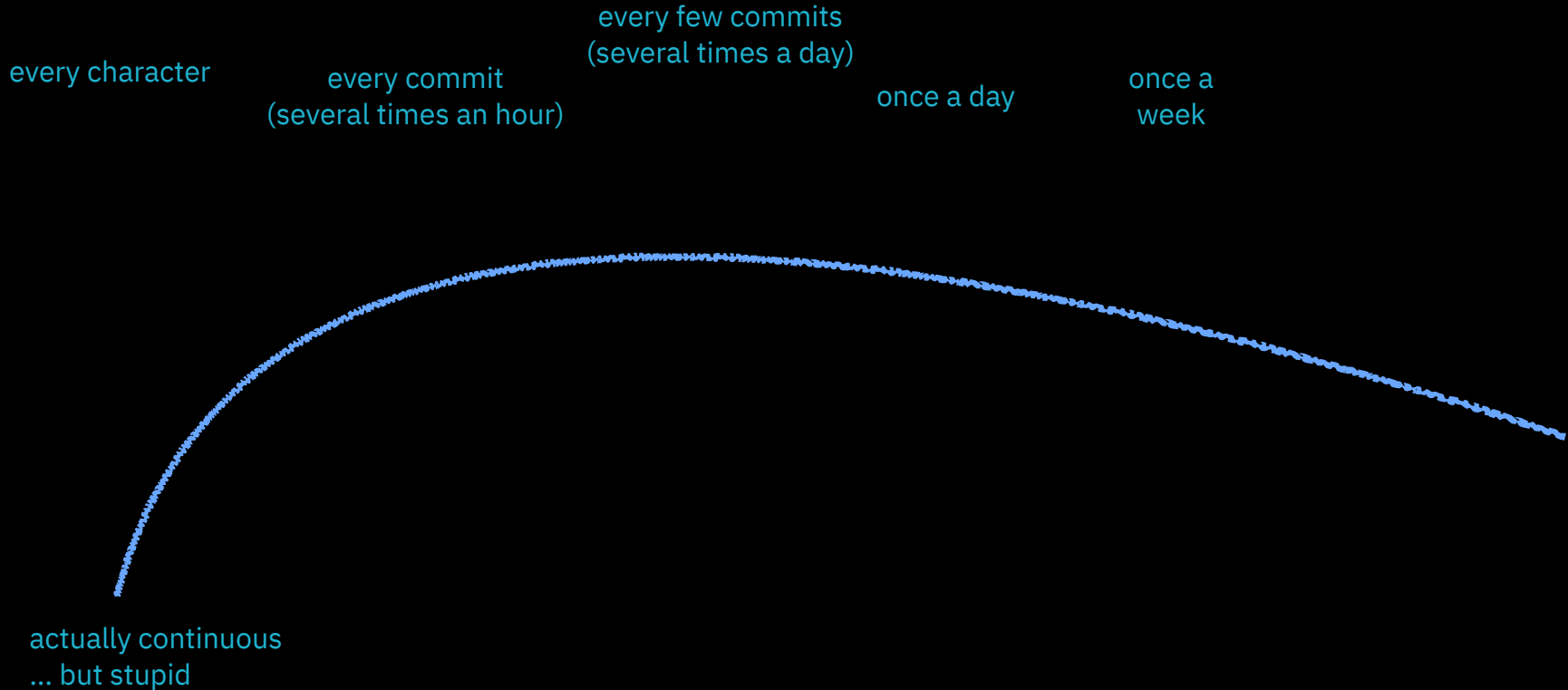
how often should you integrate?



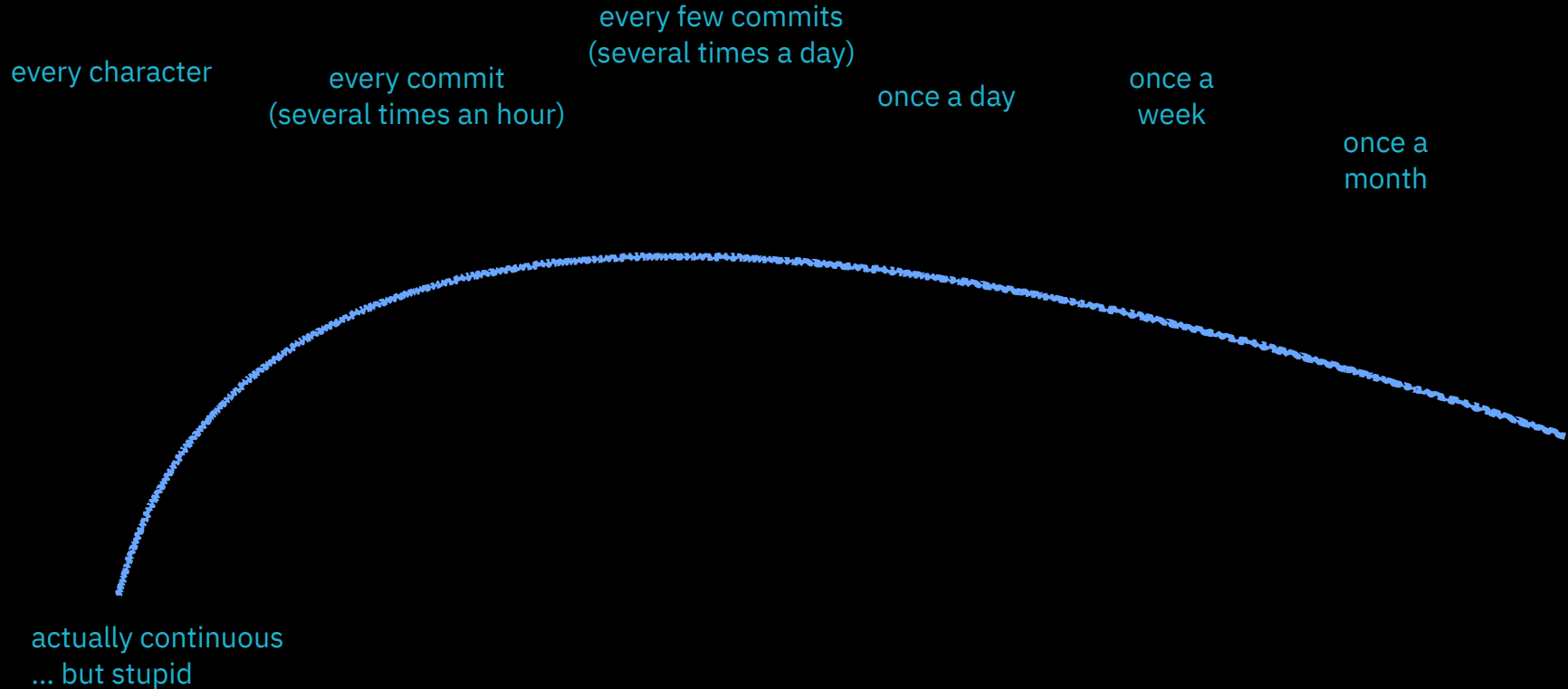
how often should you integrate?



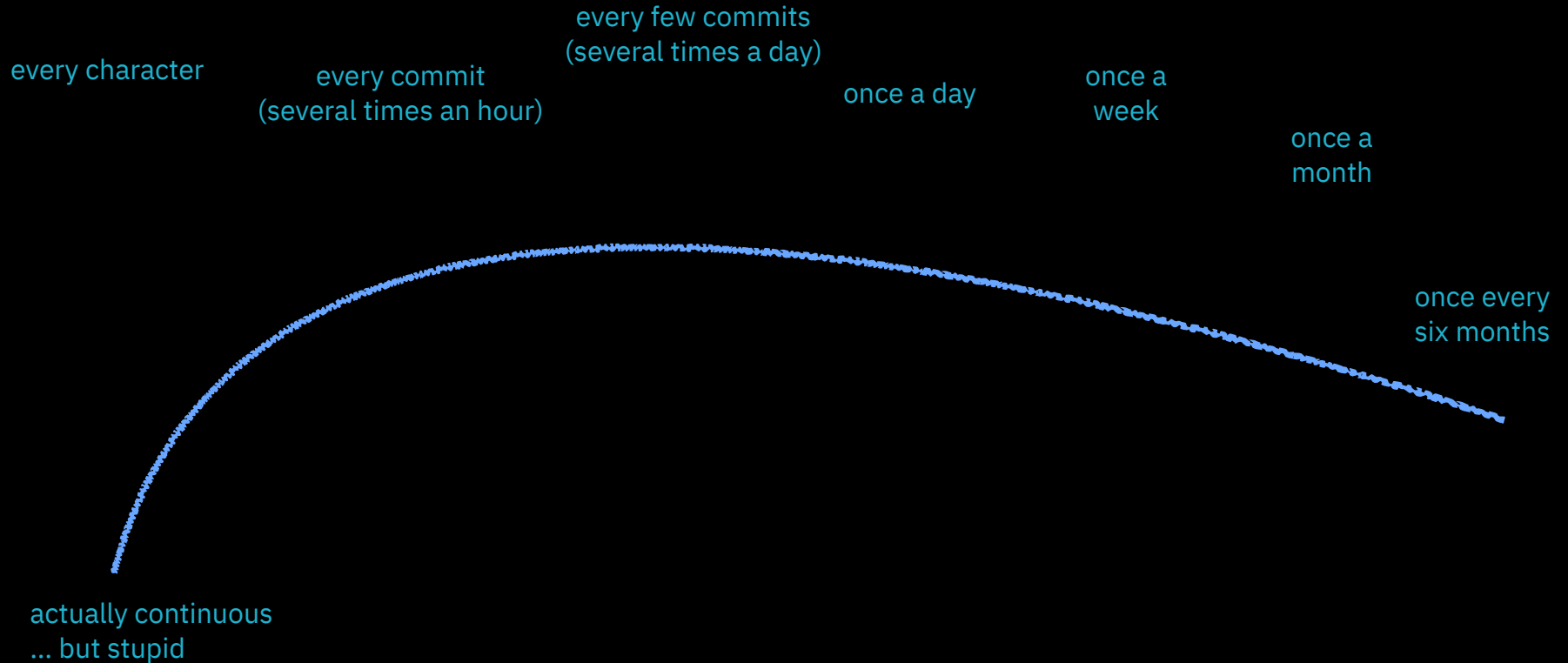
how often should you integrate?



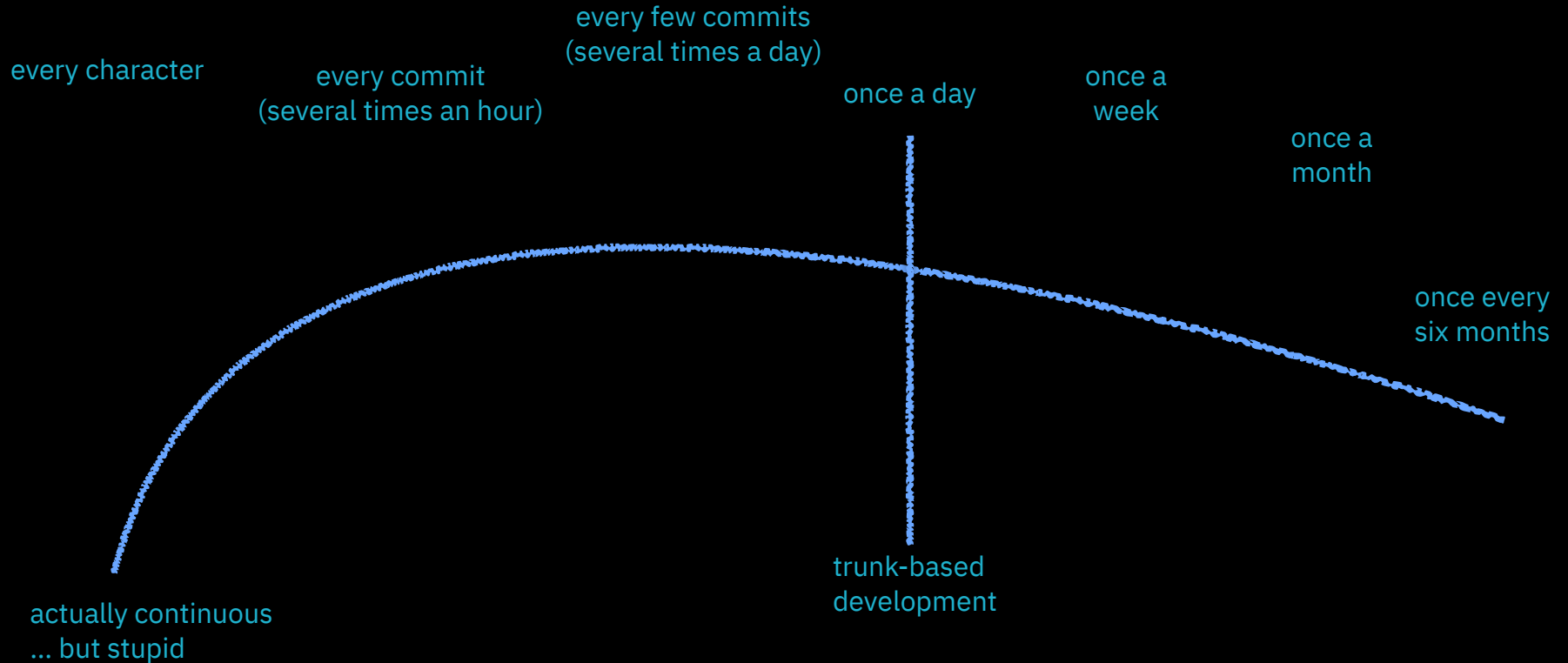
how often should you integrate?



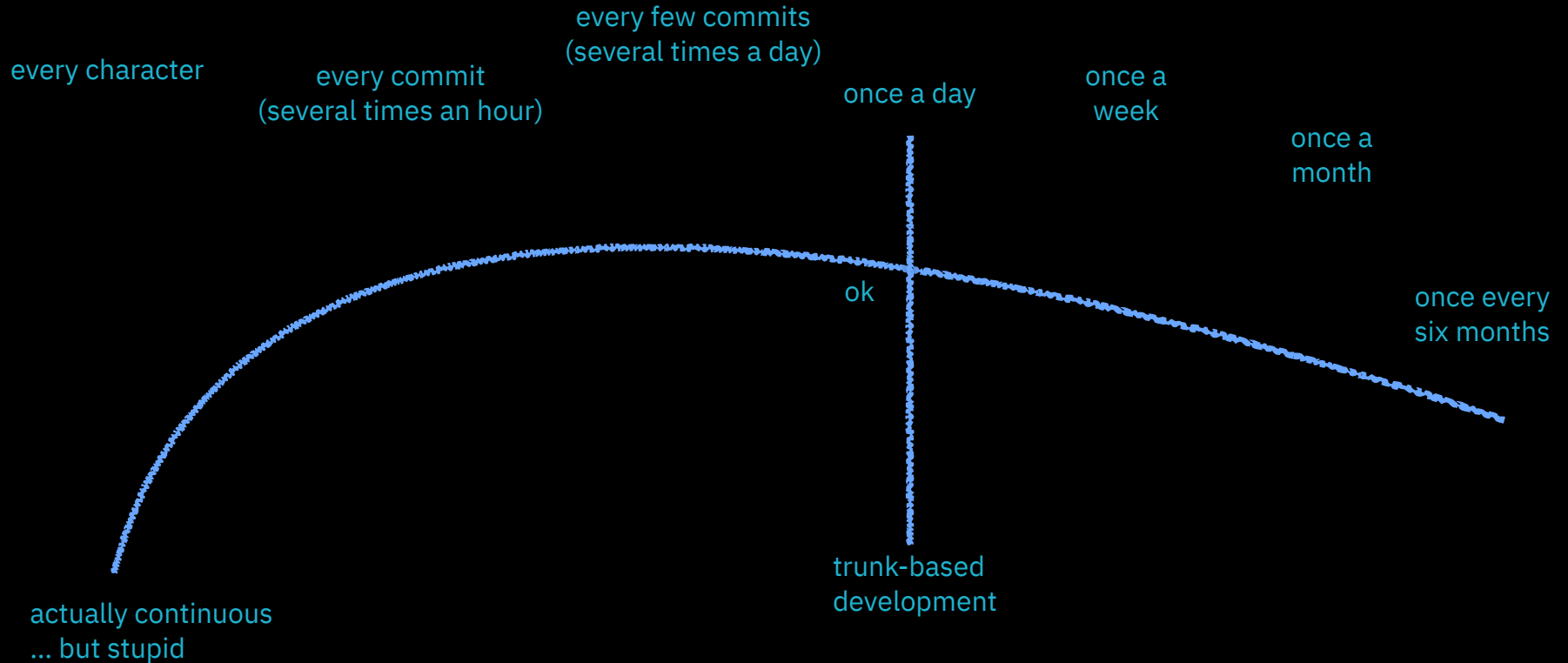
how often should you integrate?



how often should you integrate?



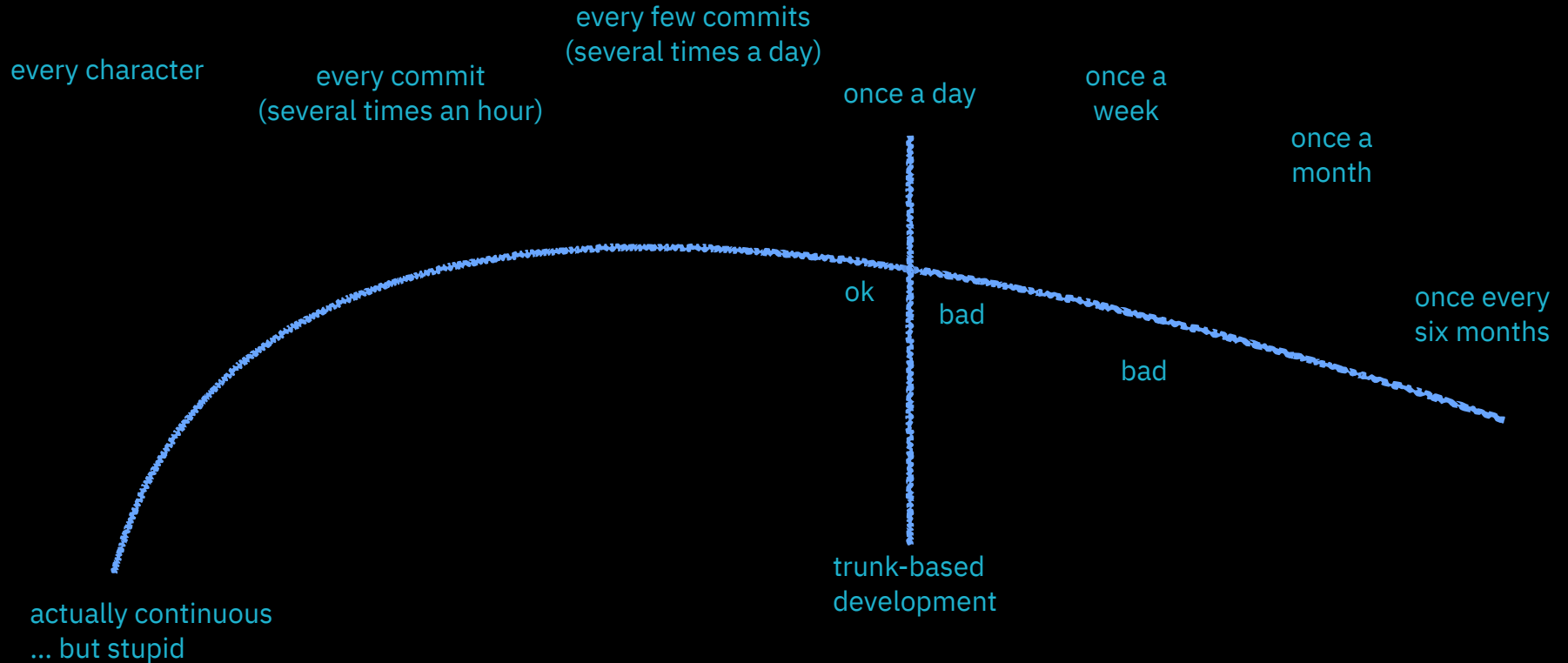
how often should you integrate?



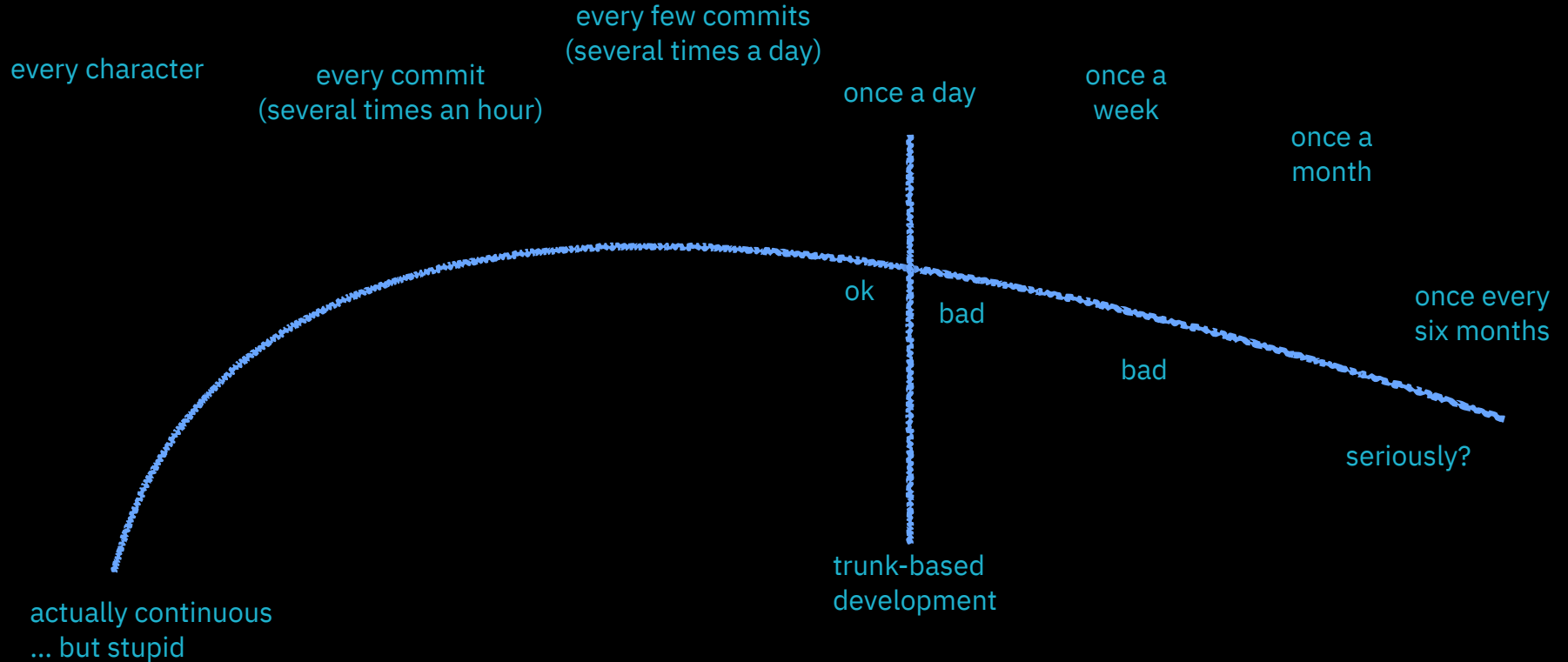
how often should you integrate?



how often should you integrate?



how often should you integrate?



actually continuous
... but stupid

trunk-based
development

seriously?

how often should you integrate?



how often should you release?

every push
(many times a day)

every user story

every epic

once a sprint

once a
quarter

once
every two
years



how often should you deploy?

every push
(many times a day)

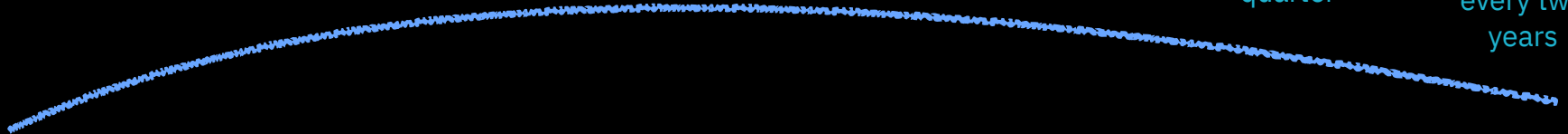
every user story

every epic

once a sprint

once a
quarter

once
every two
years



how often should you deploy?

every push
(many times a day)

every user story

every epic

once a sprint

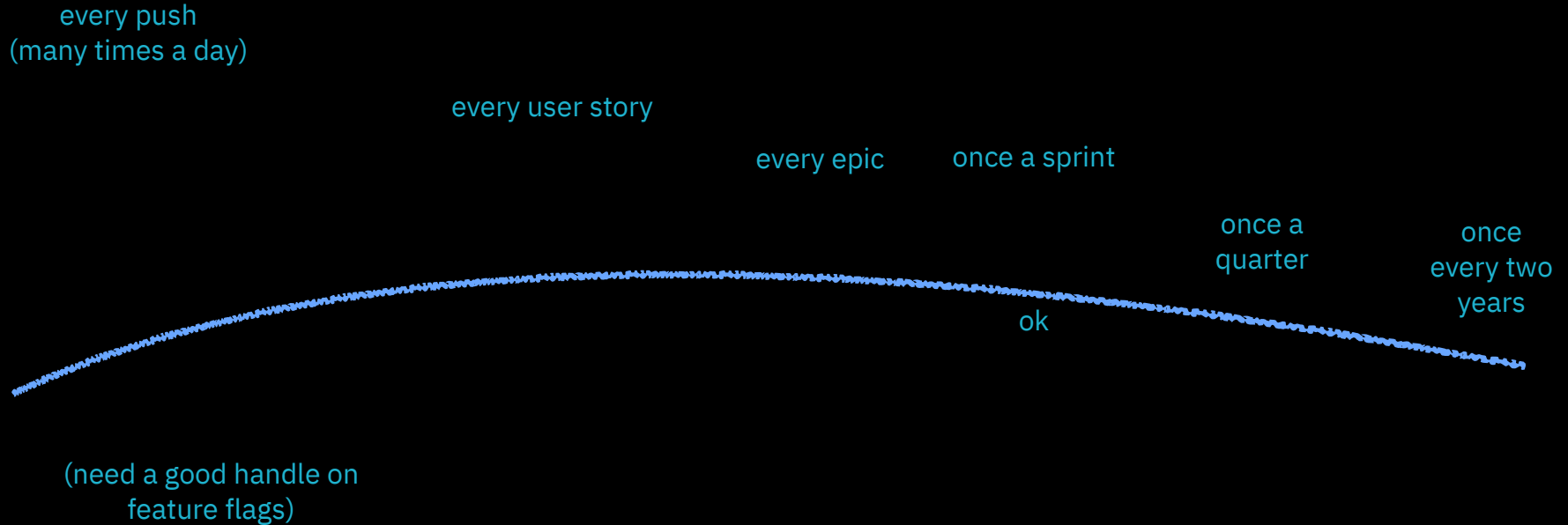
once a
quarter

once
every two
years

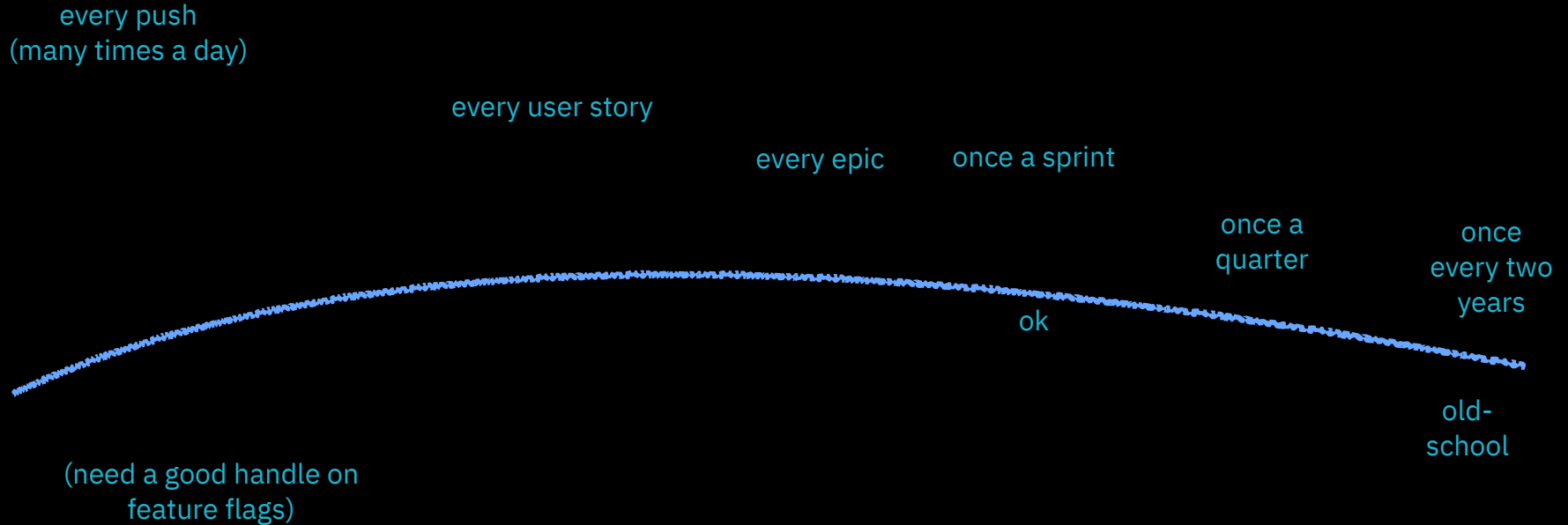
(need a good handle on
feature flags)



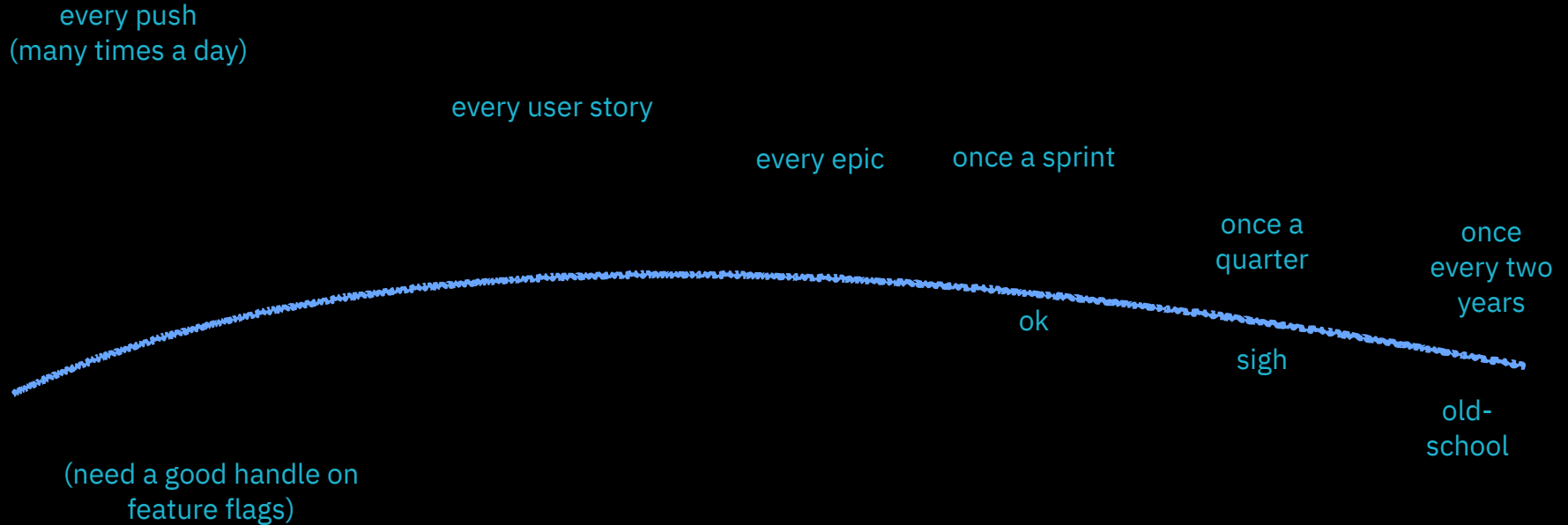
how often should you deploy?



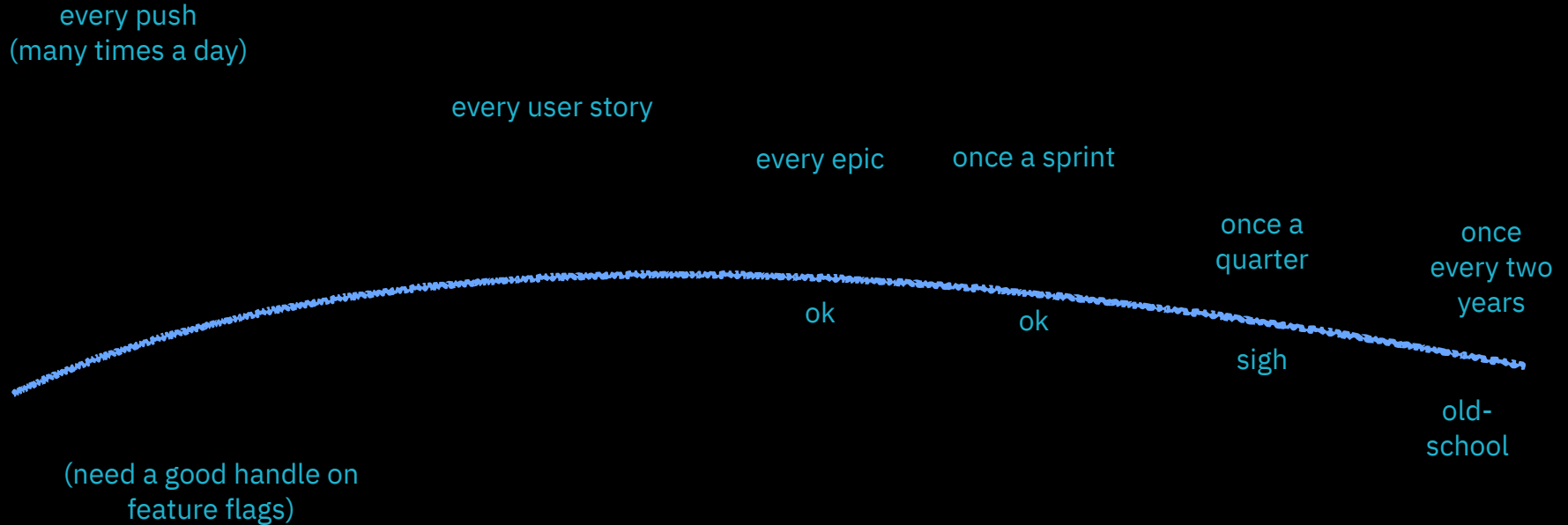
how often should you deploy?



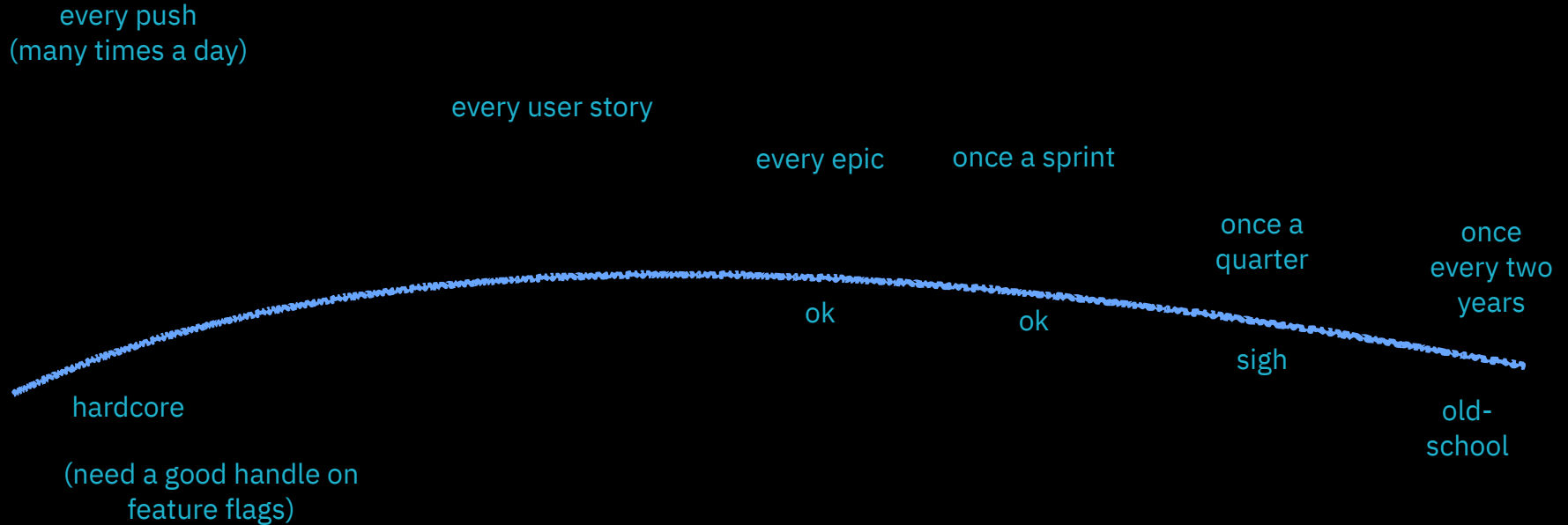
how often should you deploy?



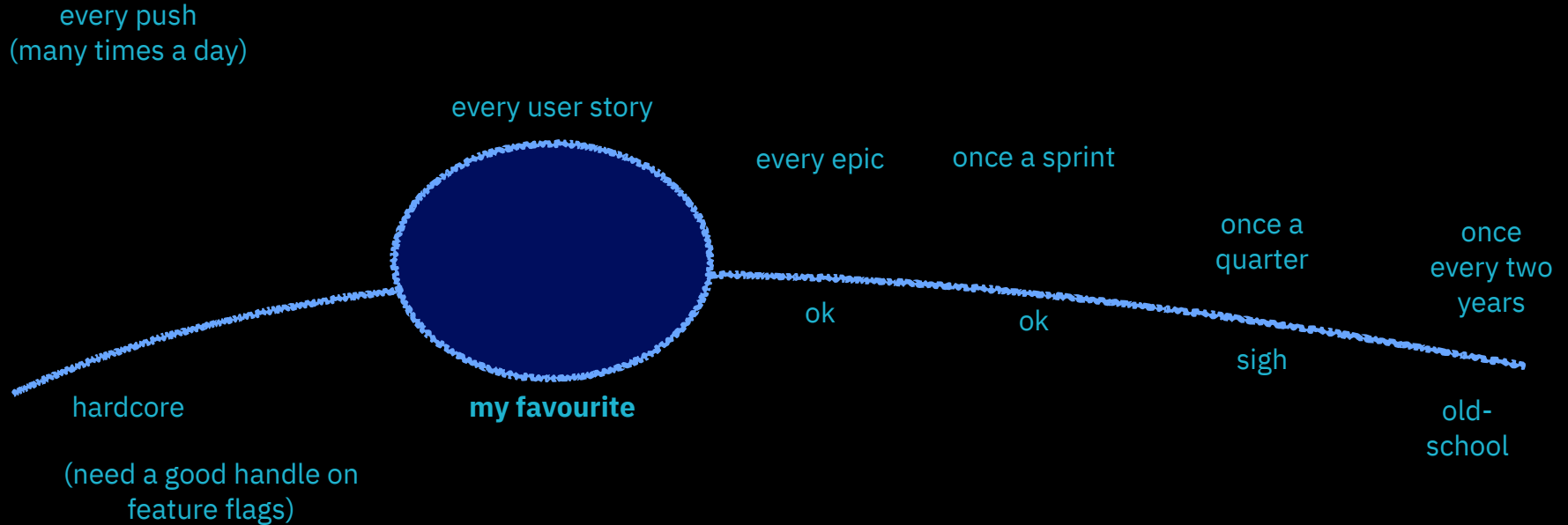
how often should you deploy?



how often should you deploy?



how often should you deploy?

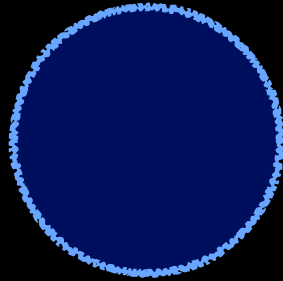


how often should you test in staging?

how often should you deliver?

how often should you deliver?

every push



my favourite

“we can’t actually **release** this.”



what's stopping more
frequent deploys?

“we can’t release this microservice...
we deploy all our microservices at
the same time.”

“we can’t ship until every
feature is complete”

if you're not embarrassed by
your first release it was too late

- Reid Hoffman

speed

speed

what's the point of architecture that
can go faster, if you don't go faster?

what's the point of
architecture that can go
faster, if you don't go faster?



drive a car

feedback is good
engineering

feedback is good business

deferred wiring

feature flags

A/B testing canary deploys

fail

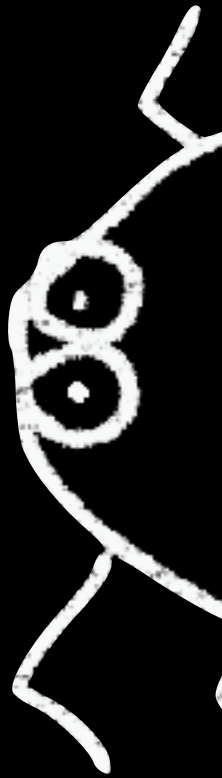
the
'someday'
automation



“our tests aren’t
automated”

“we don’t know if
our code works”

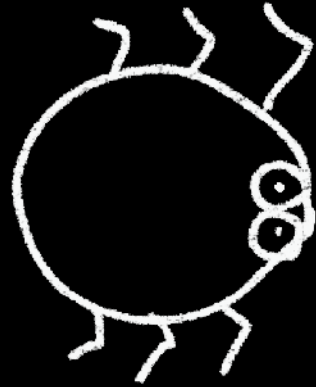
“we don’t know if
our code works”



systems **will** behave in
unexpected ways

dependency updates can
change behaviour

“we can’t ship
until we have
more confidence
in the quality”



microservices **need**
automated integration tests

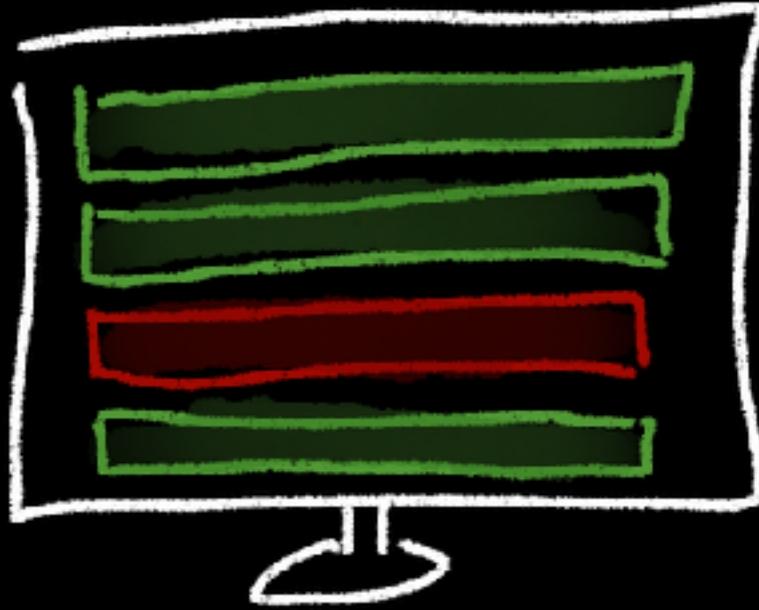


not a good CI/CD indicator

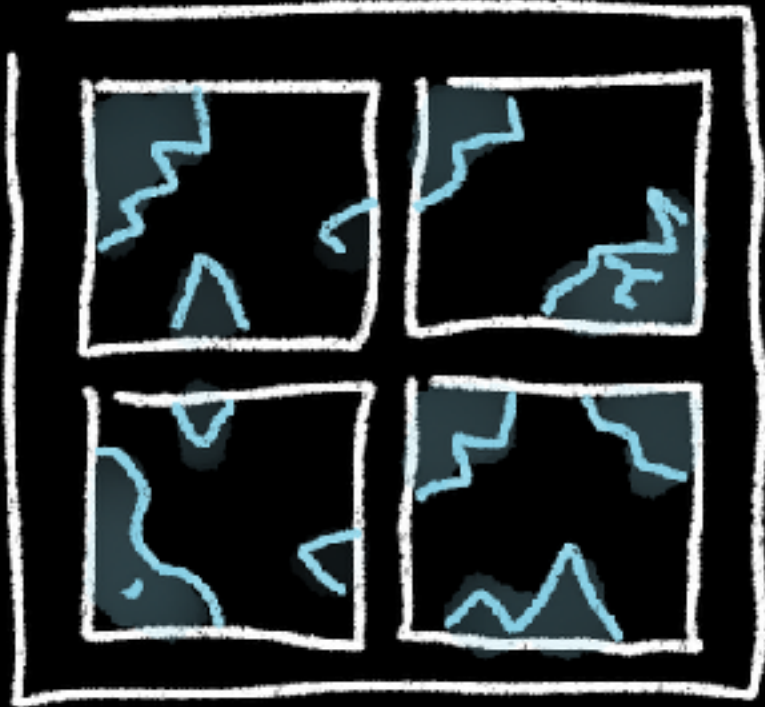


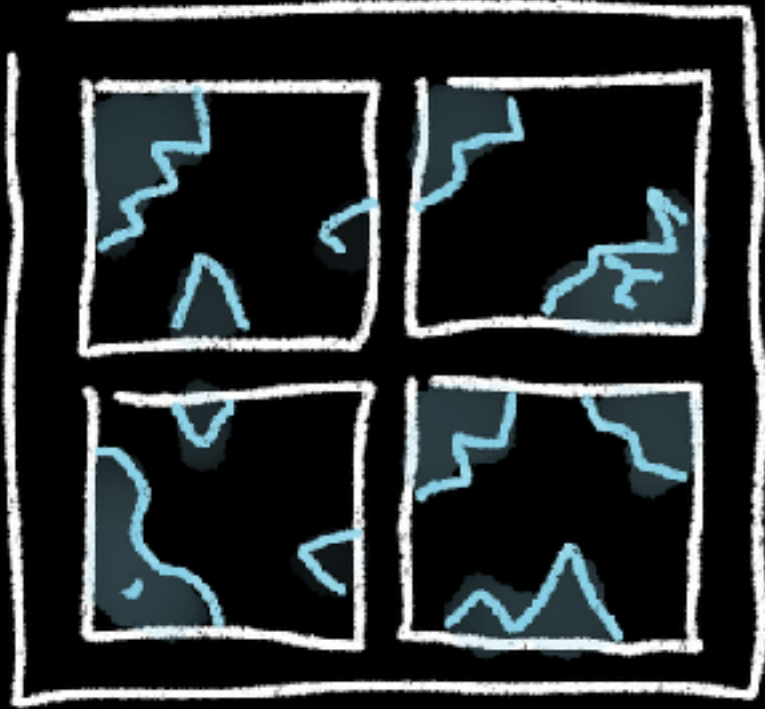
a good CI/CD indicator

“we don’t know when the
build is broken”



a good build radiator



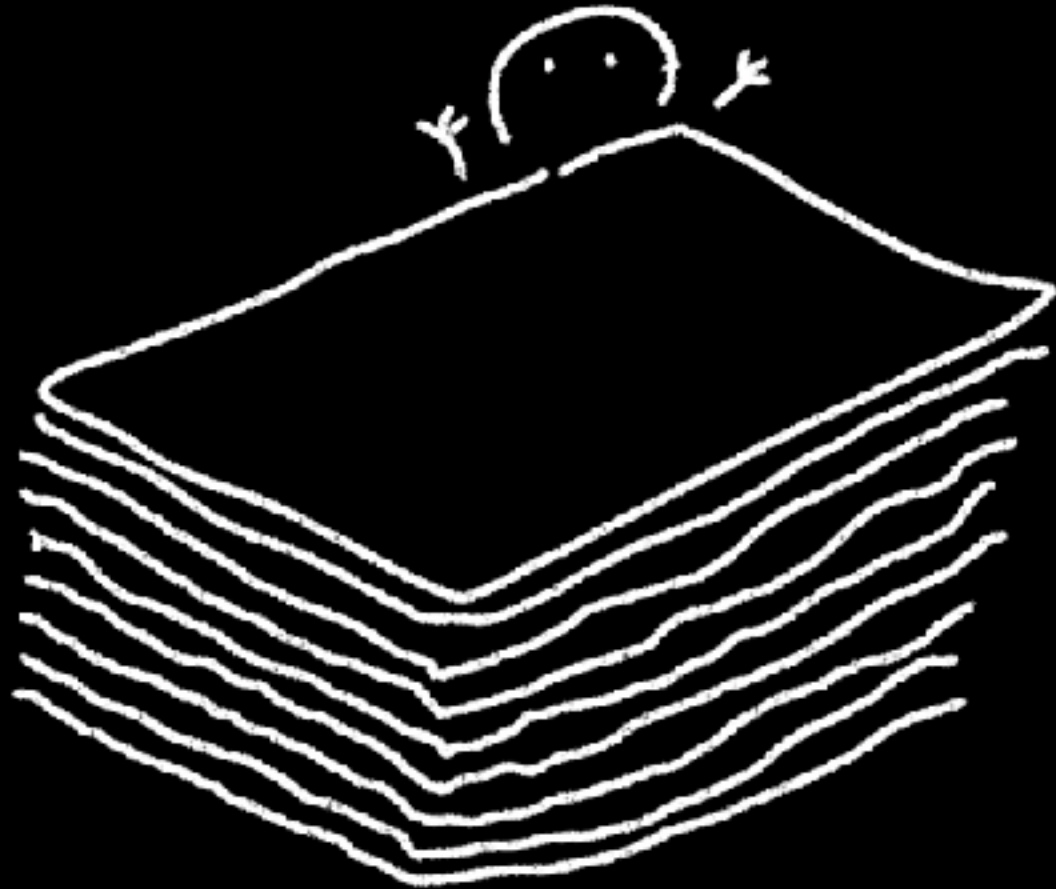


“oh yes, that
build has been
broken for a
few weeks...”

fail

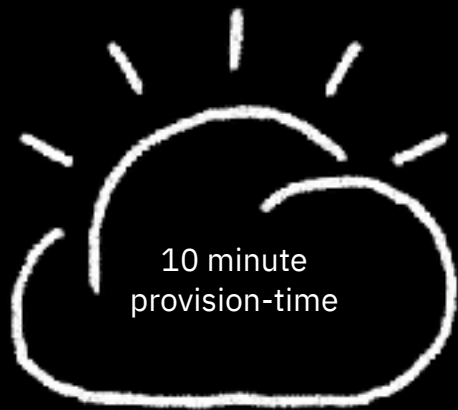
the locked-
down totally
rigid inflexible
un-cloudy cloud





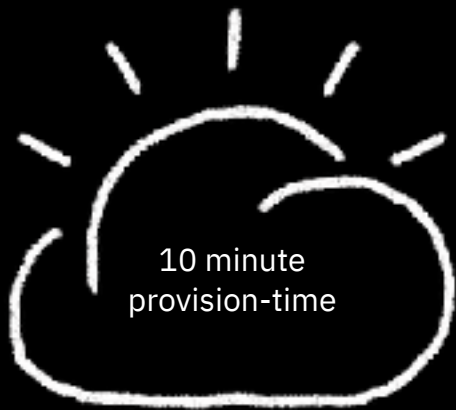
“we’ve scheduled the
architecture board
review for a month
after the project is
ready to ship”

“this provisioning
software is broken”

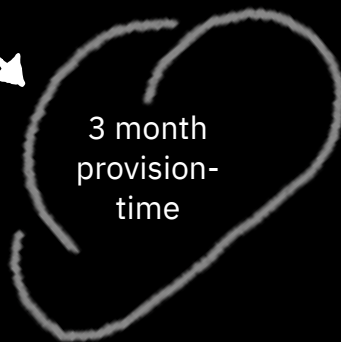


what we sold

“this provisioning
software is broken”

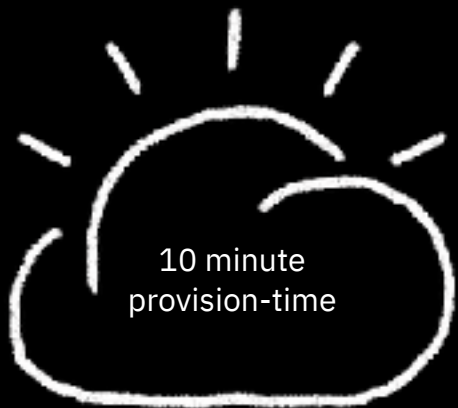


what the
client
thought
they'd got

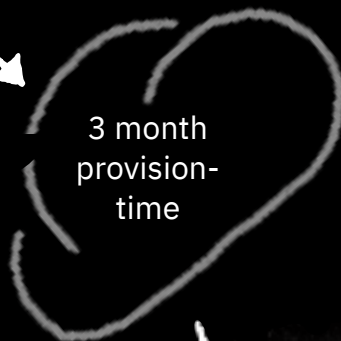


what we sold

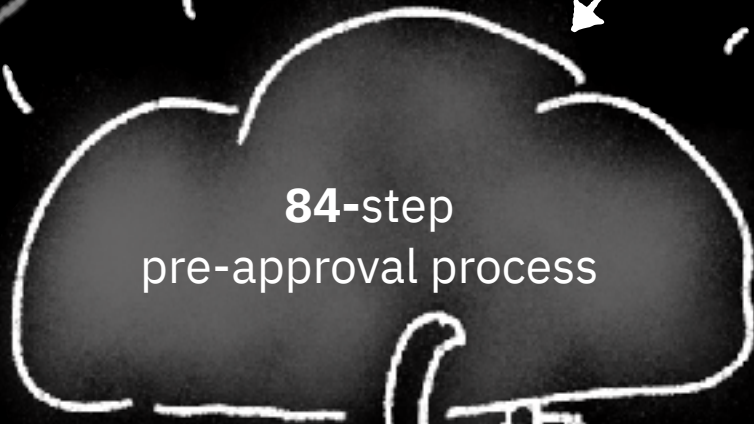
“this provisioning
software is broken”



what the
client
thought
they'd got



the reason



what we sold

“this provisioning
software is broken”







old-style governance isn't going to work



Provider A

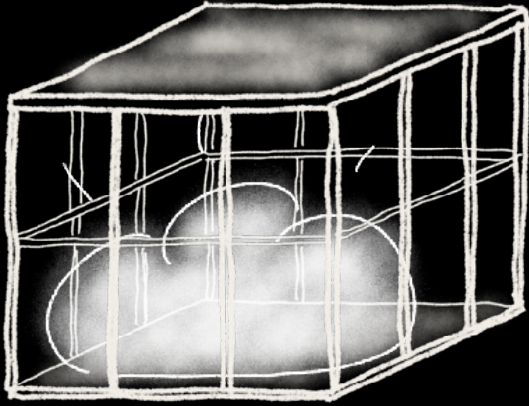


Provider A



Provider B

“we’re going to change cloud provider
to fix our procurement process!”



Provider A



Provider B

“we’re going to change cloud provider
to fix our procurement process!”

if the developers are the
only ones changing, cloud
native is not going to work

fail

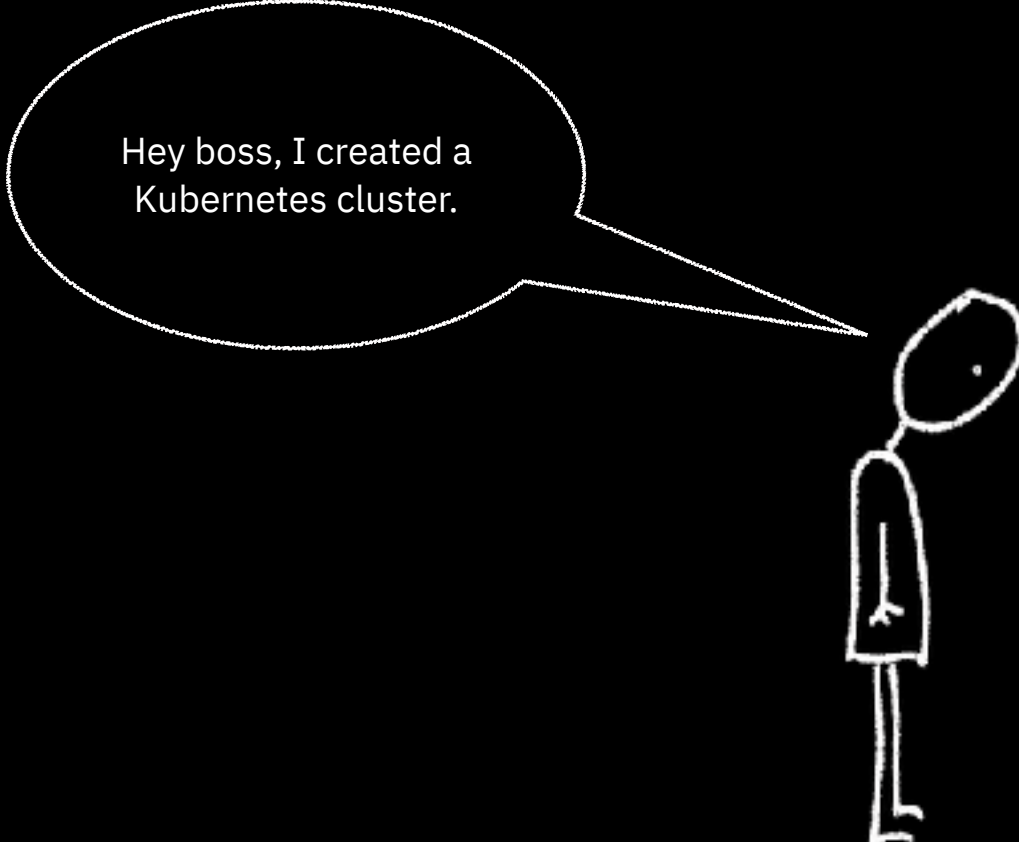
the
mystery
money pit



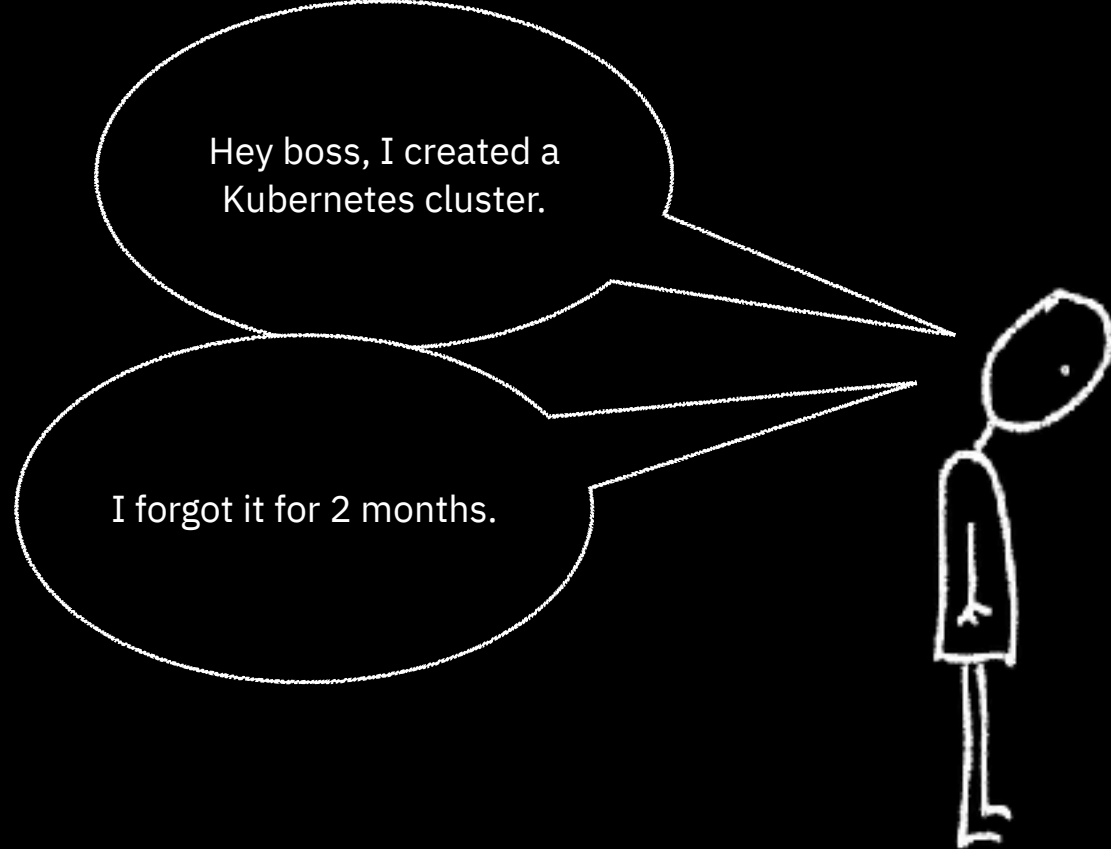
the cloud makes it so **easy**
to provision hardware.

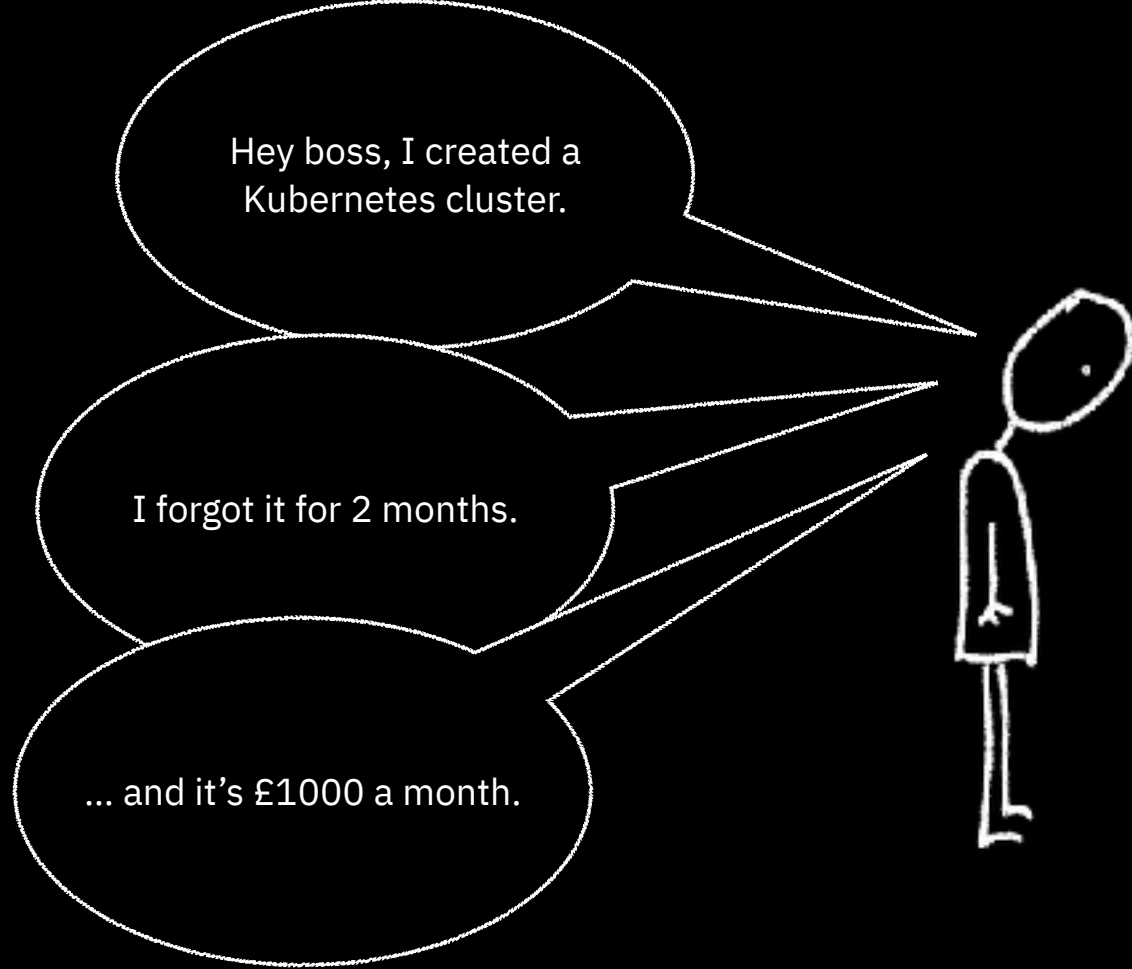
that doesn't mean the
hardware is free.

or useful.

A simple white line drawing of a stick figure on a black background. The figure is standing and facing left. A large speech bubble originates from the figure's head, pointing towards the upper left. Inside the speech bubble is the text "Hey boss, I created a Kubernetes cluster.".

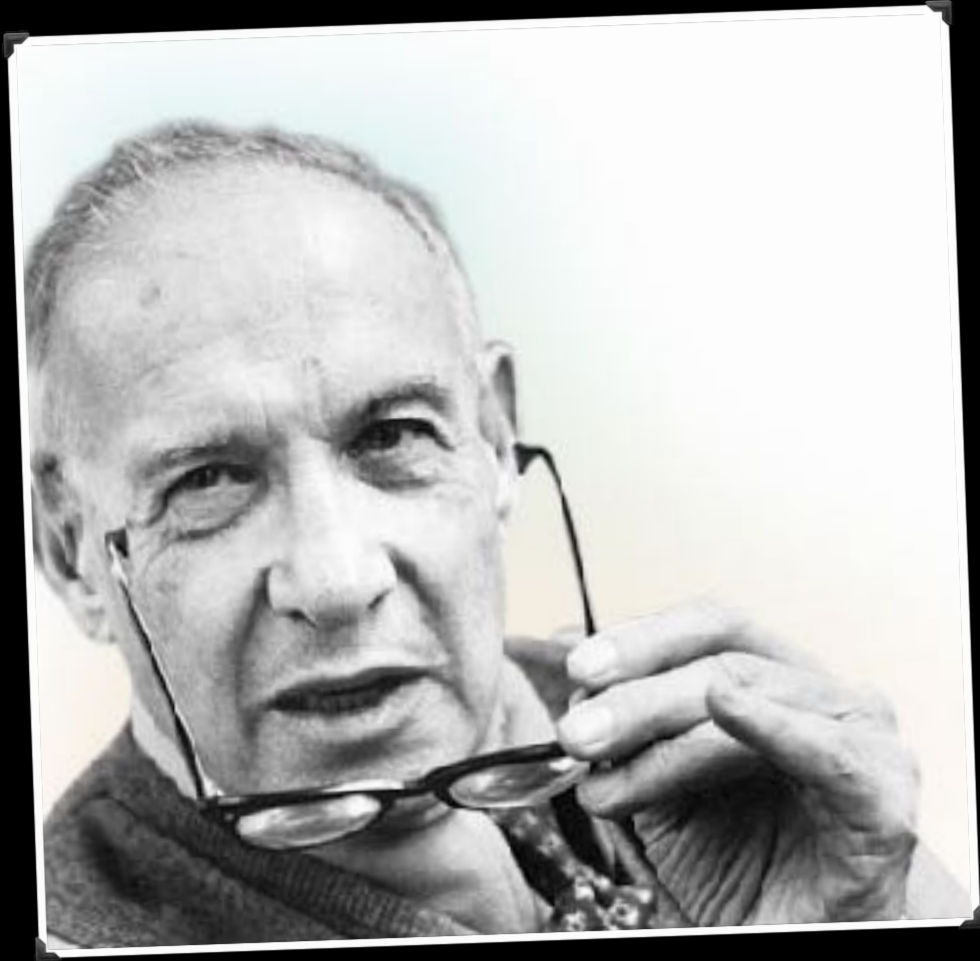
Hey boss, I created a
Kubernetes cluster.





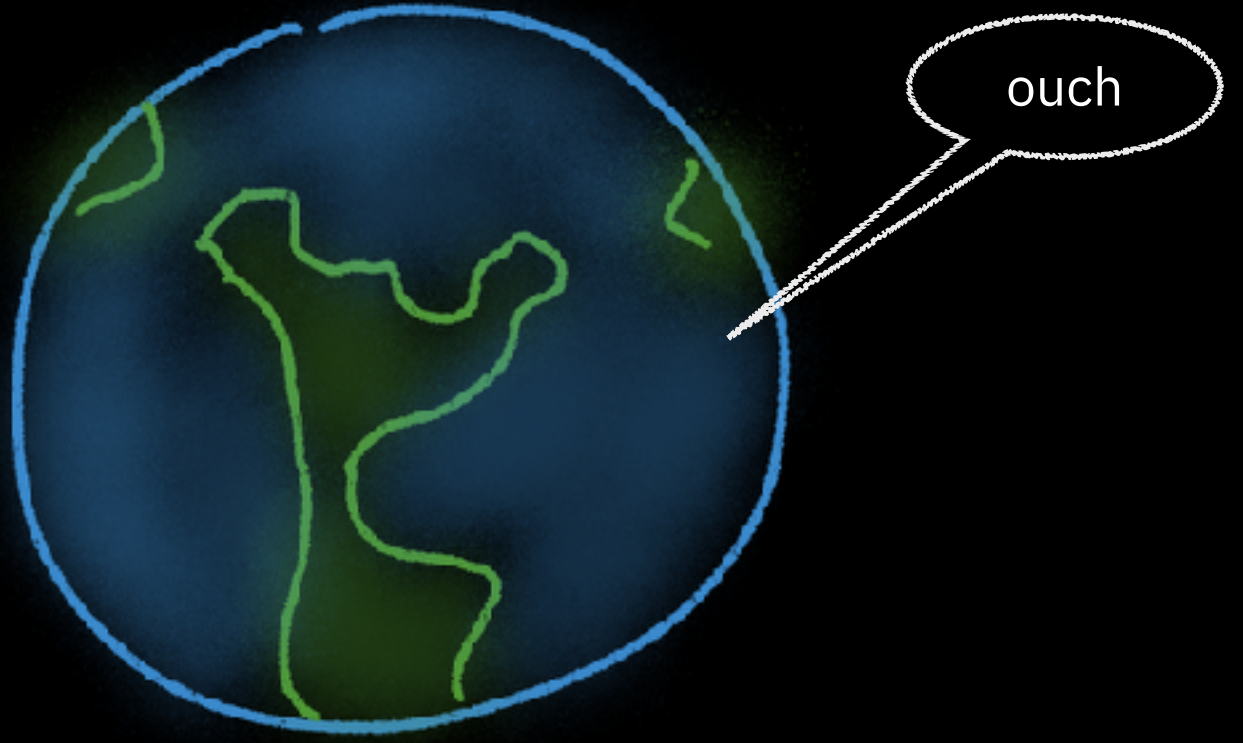






There is surely nothing
quite so useless as
doing with great
efficiency what should
not be done at all.

— Peter Drucker



“we have **no idea** how much
we’re spending on cloud.”

cloud to manage your clouds



Clusters

Name	Namespace	Labels	Endpoint	Status	Nodes	Kubernetes Version	Kubernetes Version	Storage	Memory	CPU
aks	eksnamespace	cloud=Azure datacenter=us-east-1 environment=prod name=eks owner=ccp region=US vendor=AKS	-	Ready	3	3.12-dns	v1.12.6	-	3%	9%
eks	eksnamespace	cloud=AWS datacenter=us-east-1 environment=public name=eks owner=ccp region=US vendor=KS	-	Ready	3	3.12-dns	v1.12.6-eks-7c34c0	-	5%	12%
gke	gknamespace	cloud=Google datacenter=us-east-1-b environment=prod name=gke owner=ccp region=US vendor=GKE	-	Ready	3	3.12-dns	v1.12.7-gke-12	-	8%	26%
lcp	lcpnamespace	cloud=IBM datacenter=hankfurt environment=Dev name=lcp owner=ccp region=EU vendor=ICP	Launch	Ready	5	3.12-dns	v1.12.4-lcp-ee	100%	35%	17%
iks	iknamespace	cloud=IBM datacenter=berlin environment=secure location=public name=iks owner=ccp region=EU vendor=IKS	-	Ready	2	3.12-dns	v1.12.9-IKS	-	59%	44%
		cloud=IBM datacenter=hankfurt								

fail



microservices
ops mayhem



SRE

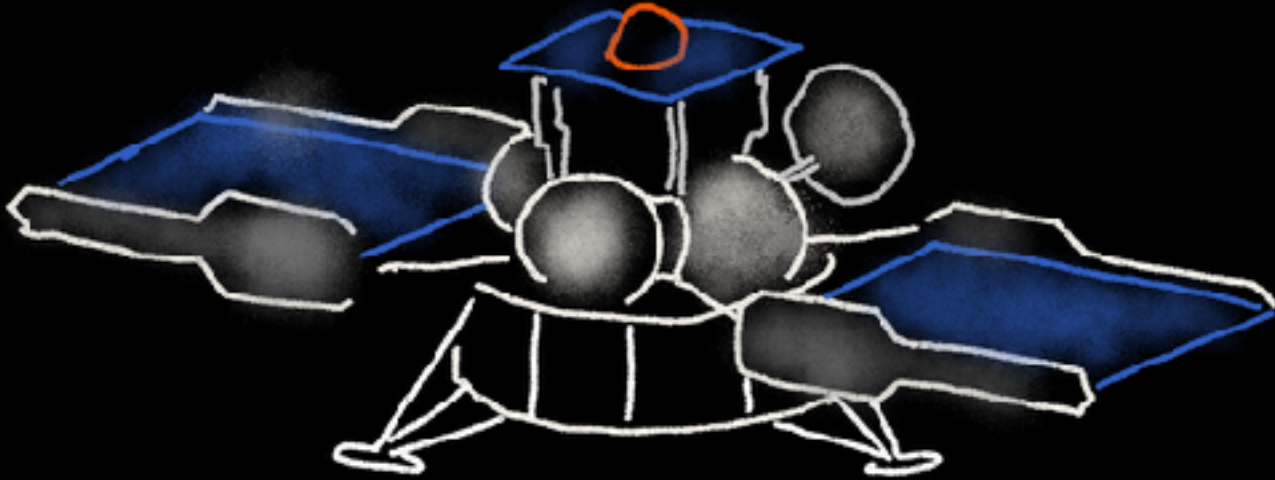
site reliability engineering



make
releases
deeply **boring**

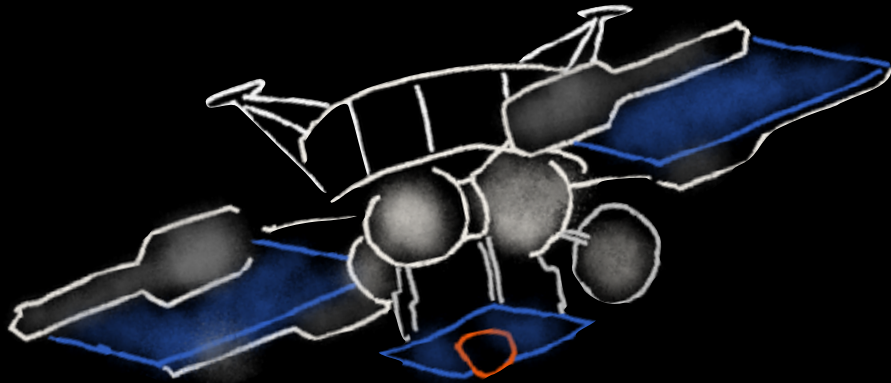


how to brick a spaceprobe

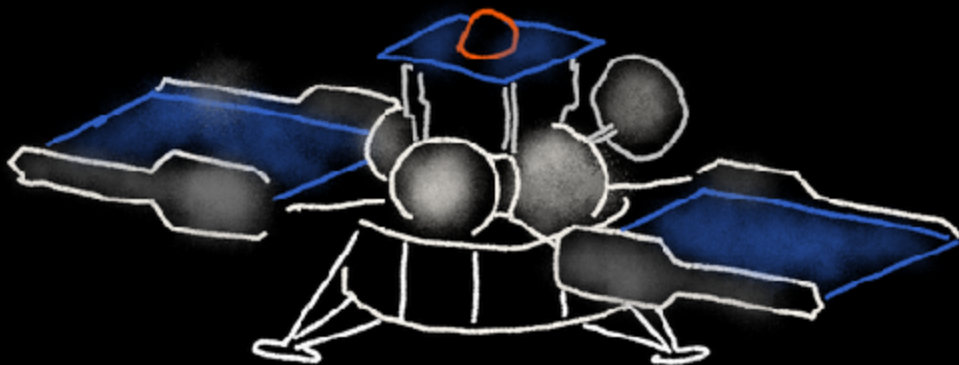


Phobos 1

“we couldn’t get the automated checks to work, so we bypassed them”



recoverability



unrecoverable

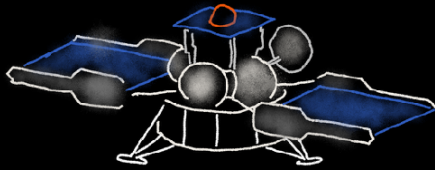
manual
intervention

bricked

back in ms
no data loss

fast, but
data lost

handoffs



handoffs bad
automation good

ways to
succeed at
cloud native



be clear on what you're
trying to achieve

optimise for feedback



@holly_cummins