why are DISTRIBUTED SYSTEMS so hard?

@deniseyu21

deniseyu.io

Land acknowledgment:

Portland is built on the unceded, traditional Tribal lands of the Multnomah, Kathlamet, Clackamas, other Chinook bands, Oregon City Tumwater, Tualatin Kalapuya, Molalla, and many other Tribes who make their homes along the Columbia and Willammette rivers.

In using this land, it is also important to acknowledge the policies of genocide, relocation, and assimilation that still impact many Indigenous and Native American families today.

Adapted from pcc.edu/about/diversity/cascade

Software Eng @ Pivotal

TORONTO-BASED

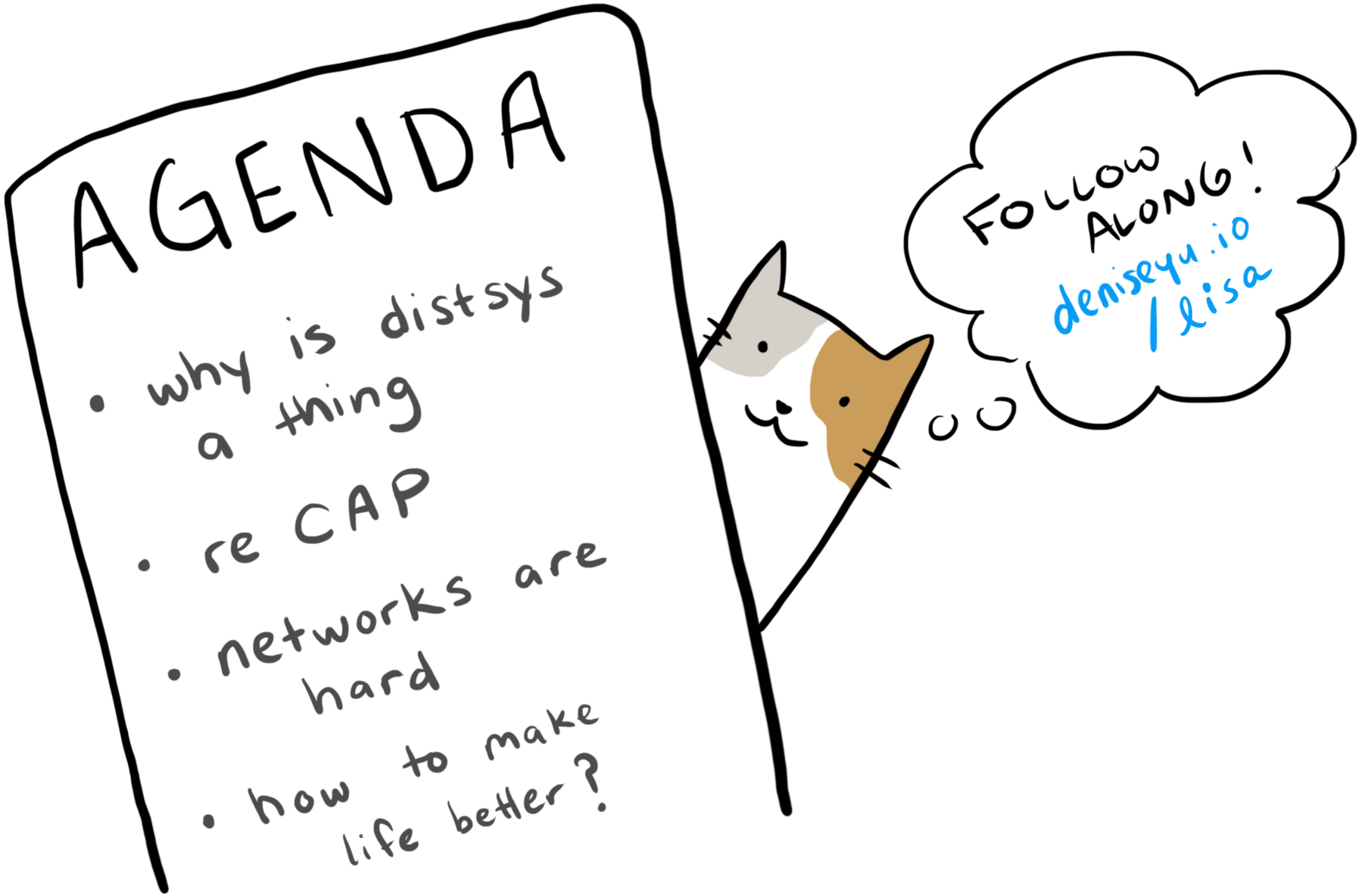
Cloud Foundry

Concourse ci.org

HELLO, #LISA19!

@deniseyu21

Tech-doodling enthusiast → deniseyu.io/art

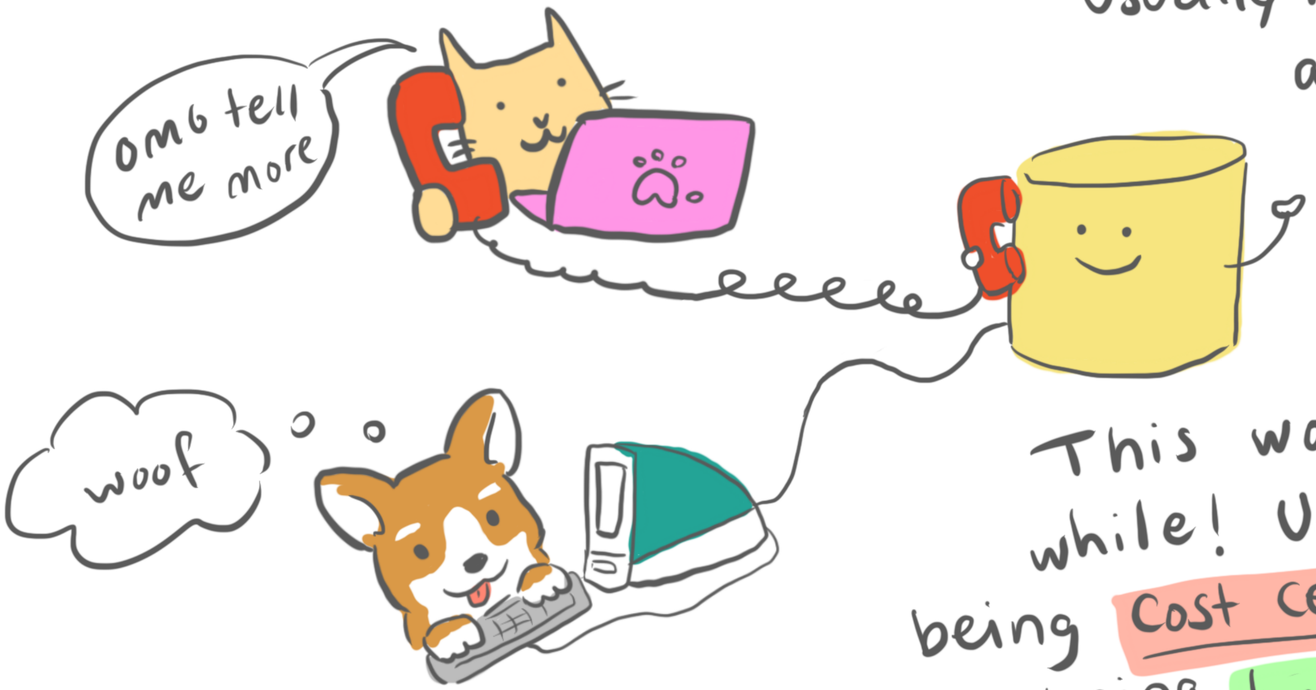

AGENDA
• why is distsys a thing
• re CAP
• networks are hard
• how to make life better?
FOLLOW ALONG!
deniseyu.io/lisa

@deniseyu21

A long time ago,

in a datacenter not too far away...

All business applications talked to one database, usually hosted on a company's own hardware.



This worked for a while! Until IT stopped being cost centers & started being business enablers.

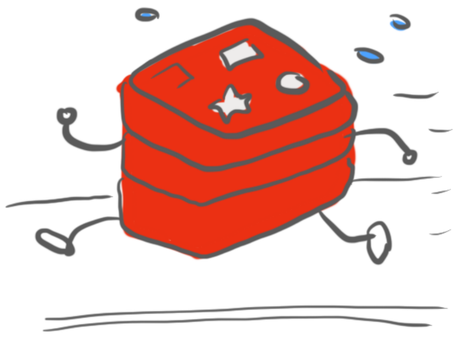
@deniseyu21

Data storage & retrieval needs evolved as software became business differentiators.

Business analysis & data warehouses

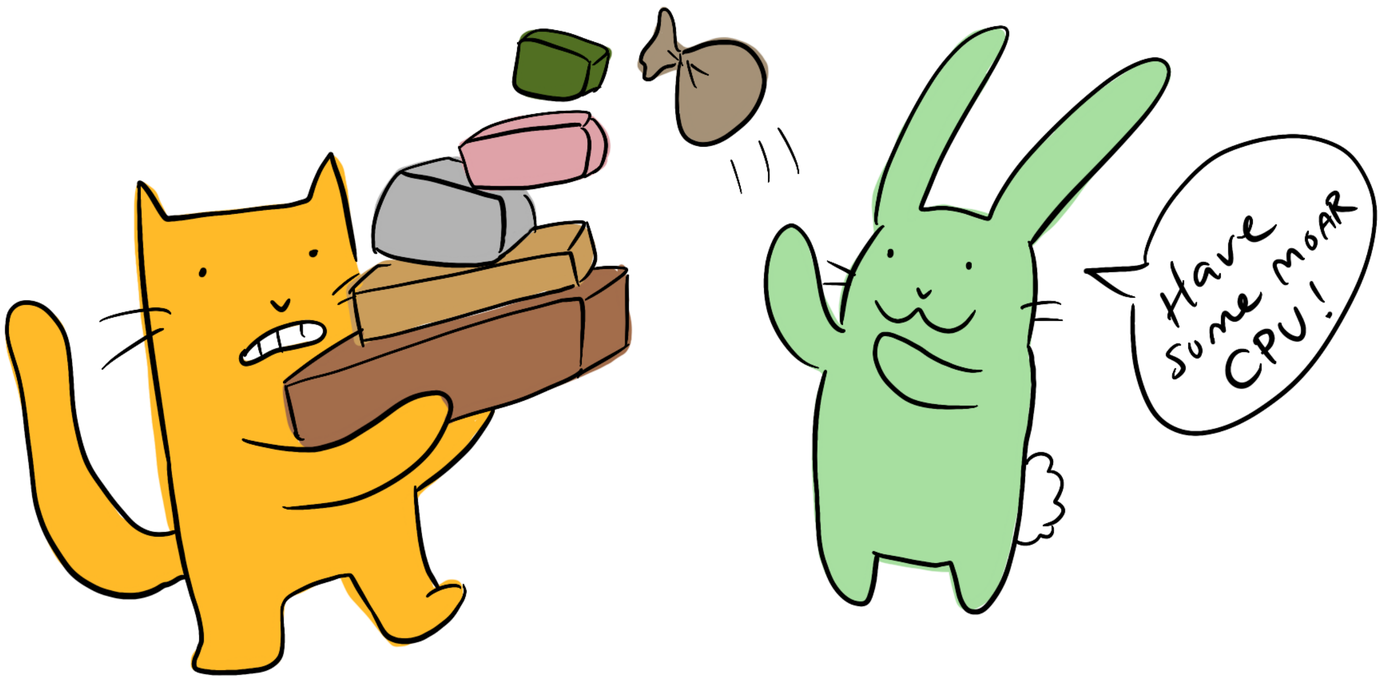
ML & Natural language processing

Faster, bigger queries!



@deniseyu21

So we scaled vertically...

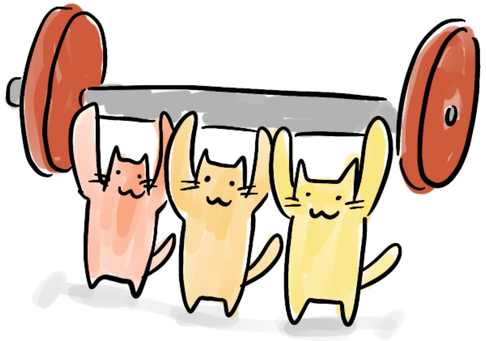


until unit economics (or physics) caught up.

@deniseyu21

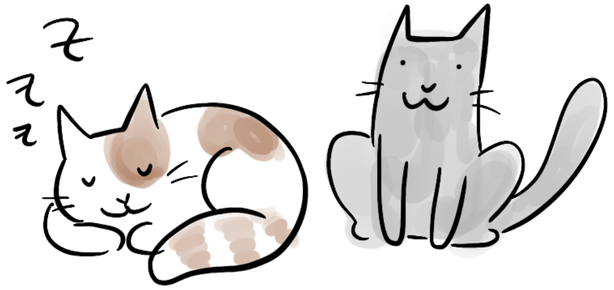
@deniseyu21

REASONS TO HORIZONTALLY DISTRIBUTE:



Scalability:

one machine cannot handle request or data size



Availability:

if one machine goes down, others keep working

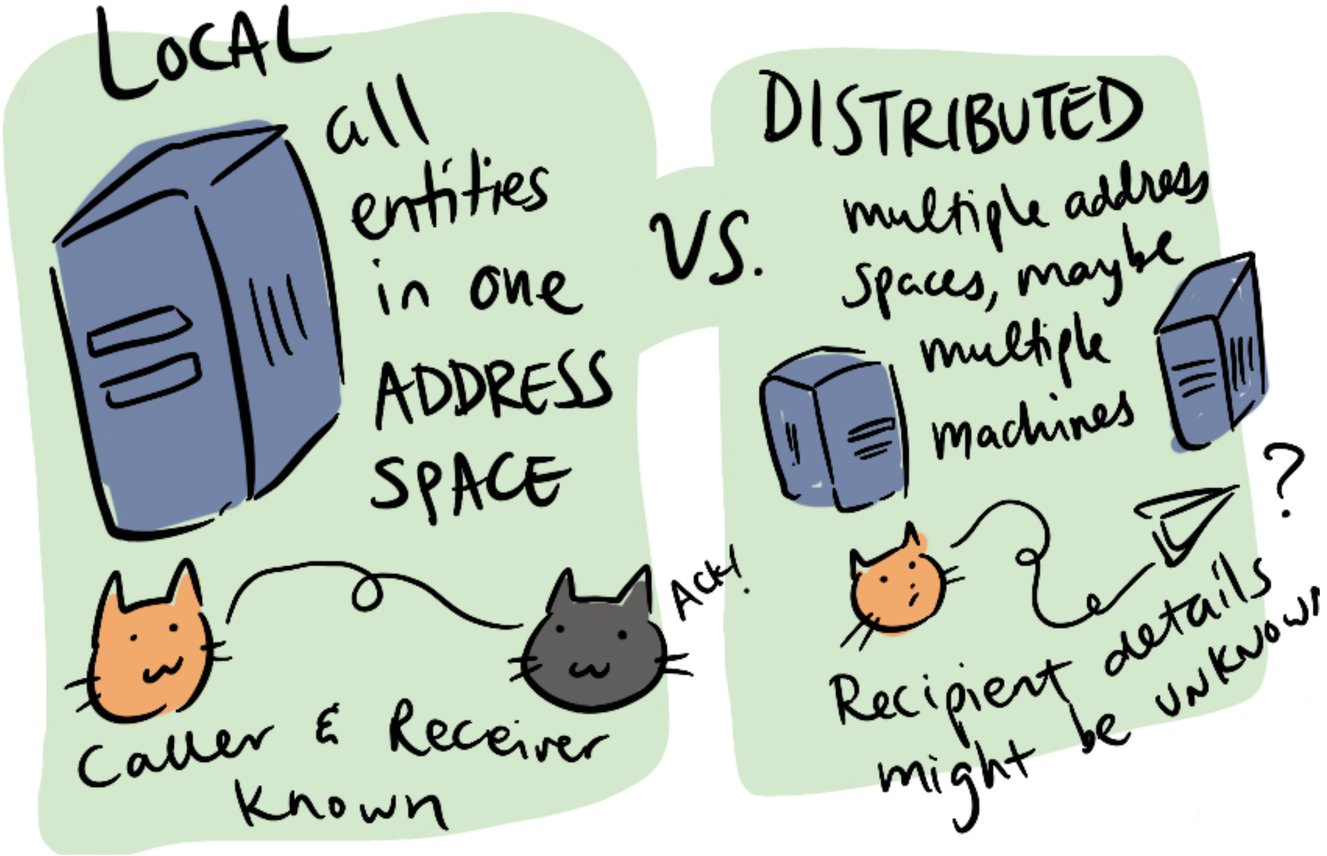
Latency:

go faster when data is stored geographically closer to users

@deniseyu21

what does it actually mean to run a distributed system?

@deniseyu21


LOCAL
all entities in one ADDRESS SPACE
Ack!
Caller & Receiver Known
VS.
DISTRIBUTED
multiple address spaces, maybe multiple machines
?
Recipient details might be unknown

@deniseyu21

a note on distributed computing

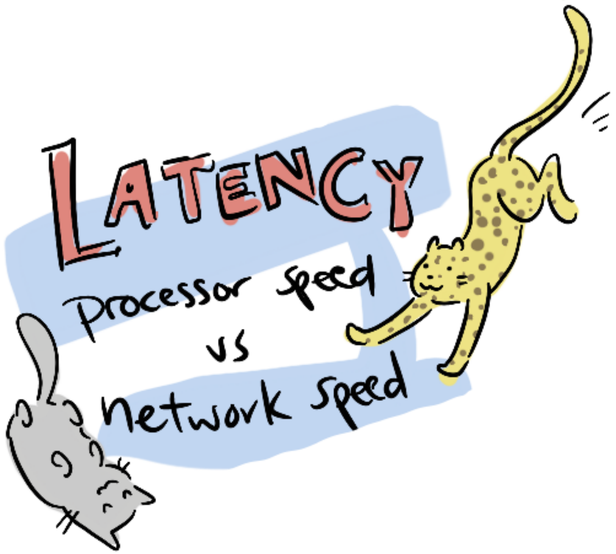
1994

@deniseyu21

THE HARD PARTS!

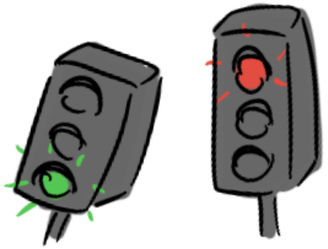
LATENCY

Processor speed
vs
network speed



MEMORY ACCESS

Pointers are only valid in their own memory space



PARTIAL FAILURES

↳ Inevitable!

@deniseyu21

You may have heard the term "shared nothing" architecture:

Machines do not share access to any resources.

@deniseyu21

a note on distributed computing

by @deniseyu21 1994

Jim Waldo
Geoff Wyant
Ann Wollrath
Sam Kendall
SUN MICROSYSTEMS

BUT — unified visions of objects aren't that simple in practice.

THINKING about ROBUSTNESS BROADLY:

Robustness isn't just guarding against failures ex. by re-enqueing messages...

Underlying good performance will be hampered by POOR DESIGN!

Are interfaces designed such that desired recovery behavior can happen?

LOCAL vs. DISTRIBUTED

LOCAL: all entities in one ADDRESS SPACE

Caller & Receiver Known

Act!

DISTRIBUTED: multiple address spaces, maybe multiple machines

Recipient details might be unknown ?

IDEALLY, a programmer should not care whether objects are implemented in different address spaces, architectures, hardware...

∴ as long as interfaces are well-defined and followed! → REMOTE PROCEDURE CALL System design!

Applications internally should only worry about marshaling passed data.

In REALITY, implementation matters.

Local function call ≠ Cross-continent invocation

HOW 2 BUILD A DISTRIBUTED APP

1. Make some interfaces
2. "CONCRETIZE": Decide what to make local vs remote for performance
3. TEST with "Real bullets"

put stuff wherever machine wants

inject faults, load, partial failures, etc...

Assumptions:
- Process/Machine boundaries will be easy to change later
- If semantically correct, how & where we run code ought not matter

Stand on flawed principles:

- ☒ the same object-oriented design will work regardless of deployment topology
- ☒ failures are implementation flaws
- ☒ interfaces are separate from context

1980: every 10 yrs language trends change but nothing *really* promotes distributed app dev

1970s Messaging

1980s RPCs

1990s Object orientation

maybe distributed programming is different!

"THE HARD PROBLEMS IN DISTRIBUTED COMPUTING ARE NOT THE PROBLEMS OF HOW TO GET THINGS ON AND OFF THE WIRE."

THE ★ HARD PARTS!

LATENCY

Processor speed vs network speed

1. Hardware can improve network speed eventually
2. We should instrument apps to see if critical path calls can be made local — architectural design comes first!

MEMORY ACCESS

Pointers are only valid in their own memory space

Therefore all memory access must be EITHER 100% centrally managed by underlying system OR 100% manually managed

Programmers might need to be aware of local vs remote memory access, and programming language design may need to make it transparent.

PARTIAL FAILURES

↳ Inevitable!

Unexaminable failures somewhere in the middle of a communication link will always happen in distributed environments.

We can:
1. Always design as if running local → No fault tolerance
2. Design for remote failure → lots of extra complexity

Can a system recover when the cause of failure is unknown?

CASE STUDY: Network File System

Handling partial failure in a filesystem

2 failure modes:

Soft mounts: all server errors exposed & returned to client

Hard mounts: client waits for server to recover. Maybe long hang time!

Neither of these ideal for distributed computing — then again, NFS was designed for local usage.

Works great in colocated, centrally-managed machines!

"...there are irreconcilable differences between local and distributed computing... Rather than trying to merge local & remote objects, engineers need to [know] the differences, and know when it is appropriate to use each."

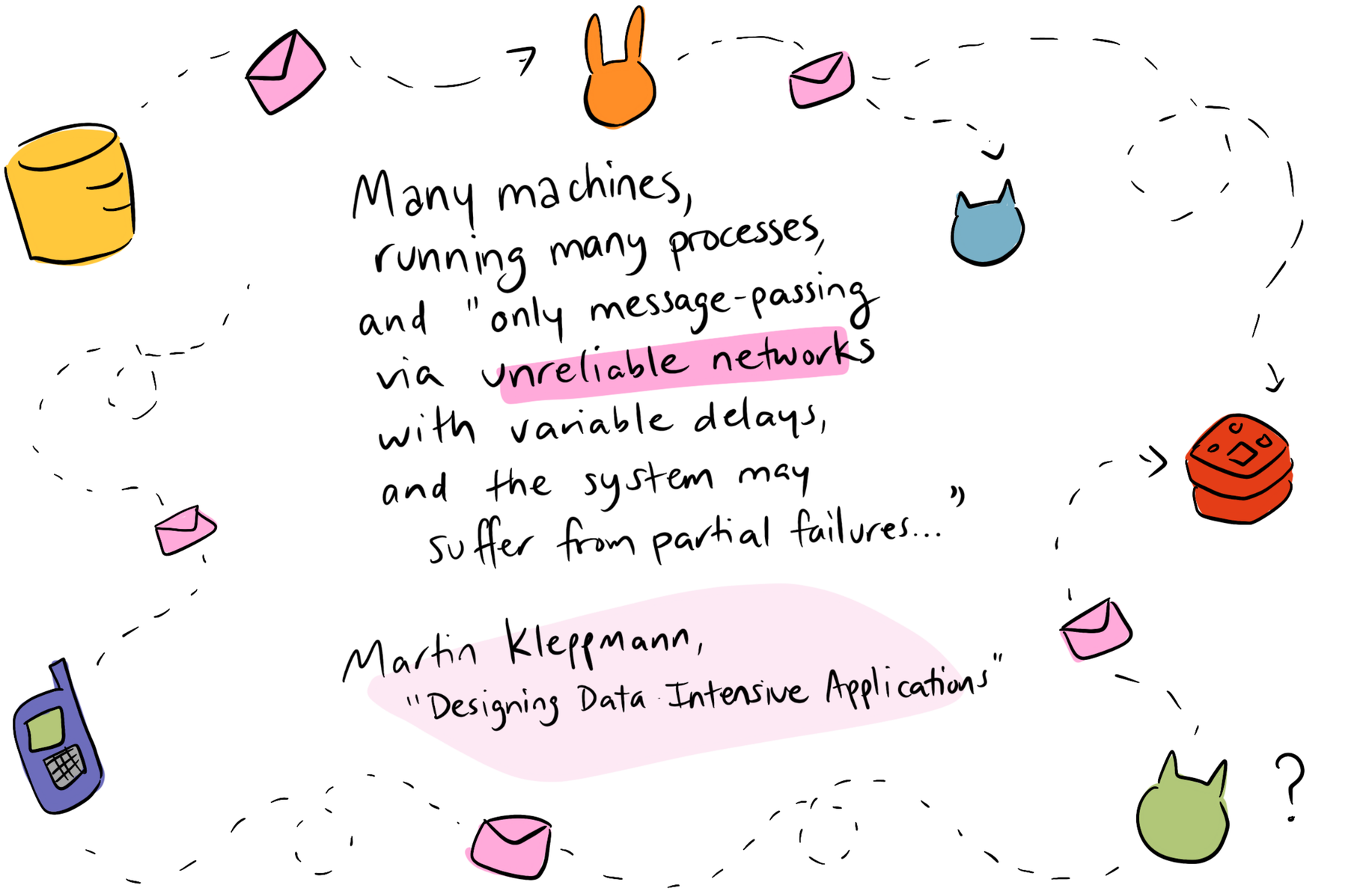
★ TIP: Consider splitting apart remote & local interactions into separate objects!

More modular, cleaner interfaces

Remote | Local

deniseyu.io/art

@deniseyu21


Many machines,
running many processes,
and "only message-passing
via unreliable networks
with variable delays,
and the system may
suffer from partial failures..."
Martin Kleppmann,
"Designing Data Intensive Applications"
?

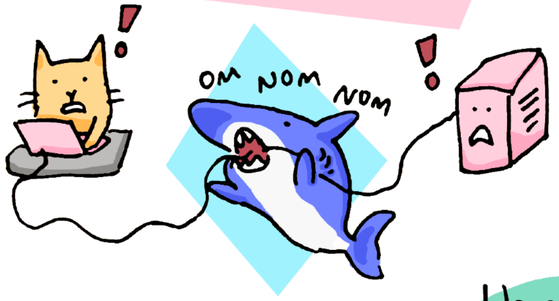
@deniseyu21

the 8 Fallacies of Distributed Computing

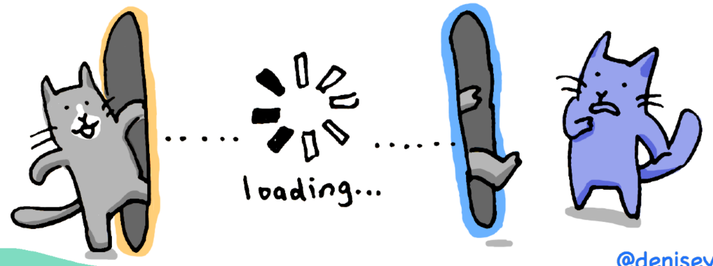
Originally formulated by L. Peter Deutsch & Colleagues at Sun Microsystems in 1994; #8 added in 1997 by James Gosling

@deniseyu21

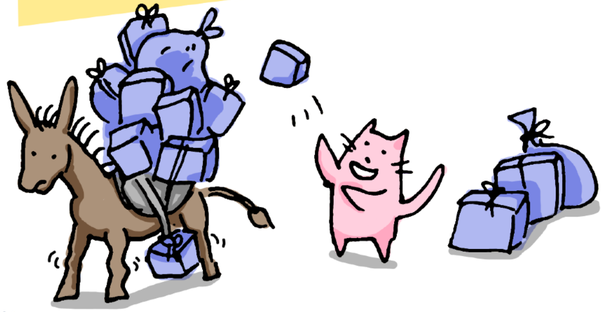
① The network is reliable



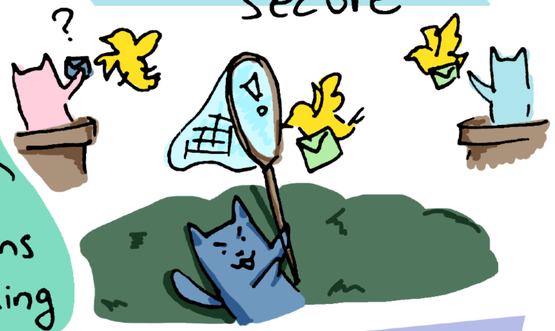
② Latency is ZERO



③ Bandwidth is infinite



④ The network is secure



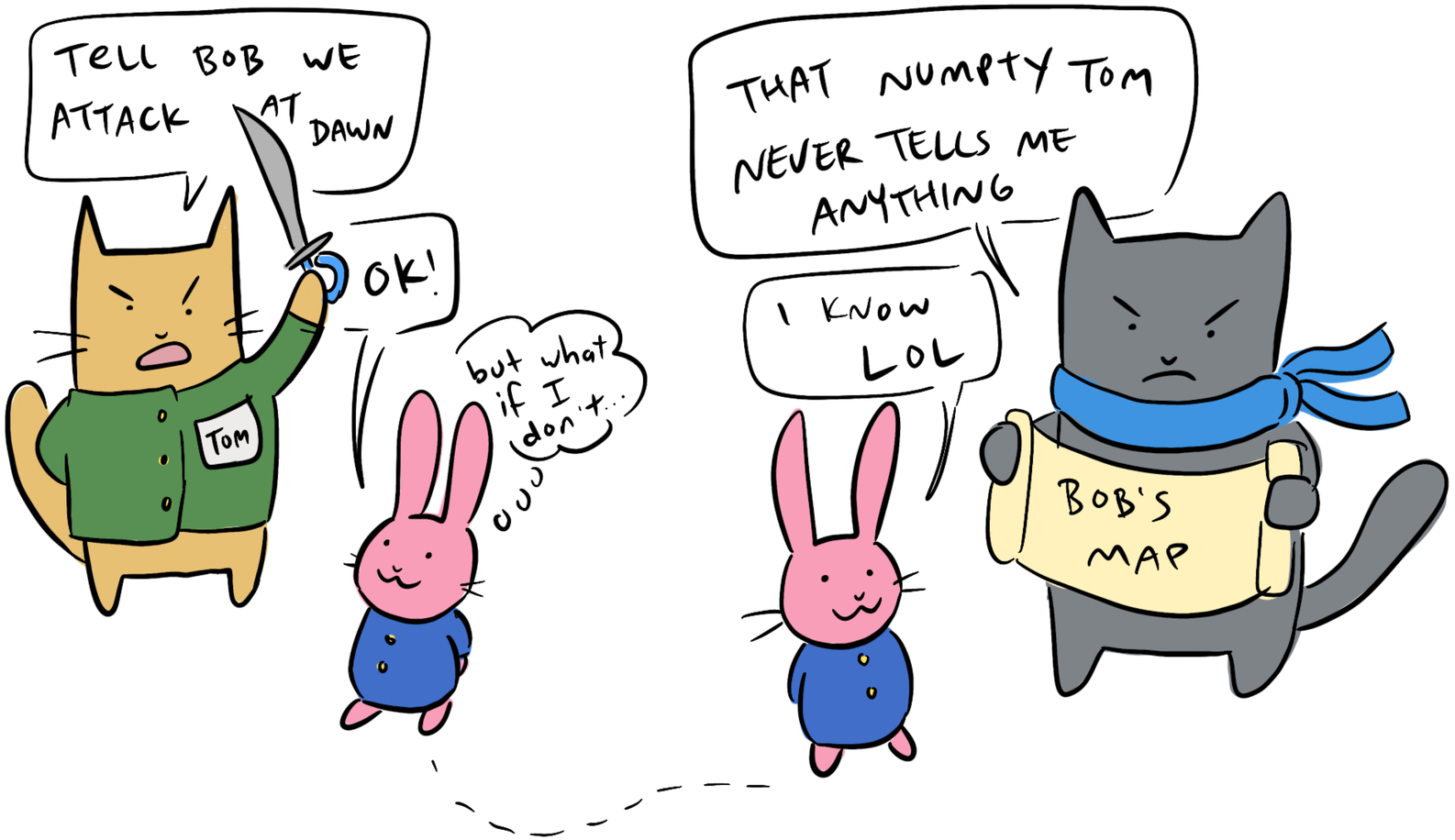
⑤ Topology doesn't change

⑥ There is only one administrator

⑦ Transport costs $0

⑧ The network is homogeneous

The Byzantine Generals Problem



@deniseyu21

SO MUCH UNRELIABILITY!

How can we even <u>know</u> what is true about the state of the world?

@deniseyu21

MONITOR & observe

@deniseyu21

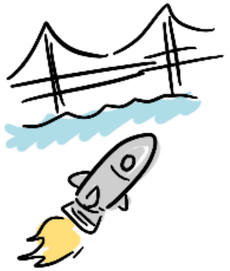
8 TRAPS OF #chaos Day 19

CHAOS ENGINEERING

NORA JONES
@NORA_JS

HUMAN FACTORS & SYSTEM RELIABILITY

Chaos Eng borrows from many other industries & incidents.



Apollo 11 Launch Test:
During a test, safety precautions were not taken ⇒ "an experiment that went wrong" changed space travel forever

① "More vulnerabilities = More success!"

Error discovery is not linearly related to resilience outcomes.

Focusing on these NUMBERS distracts from having richer conversations about what actually is working.

② You can prep an exercise with a subset of the team

"We were just trying to save time."

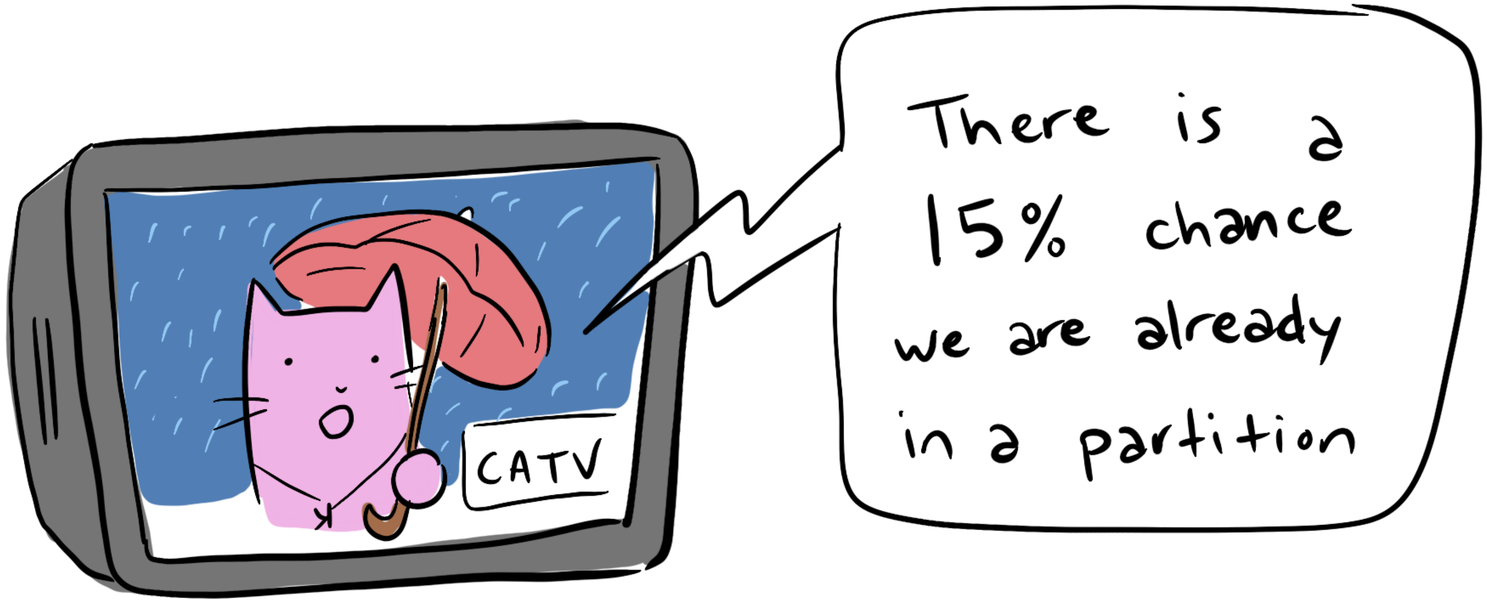
③ There is a prescriptive way to "do" chaos eng.

Chaos Eng is inherently unpredictable.

we get better by sharing.

(4.5) Chaos

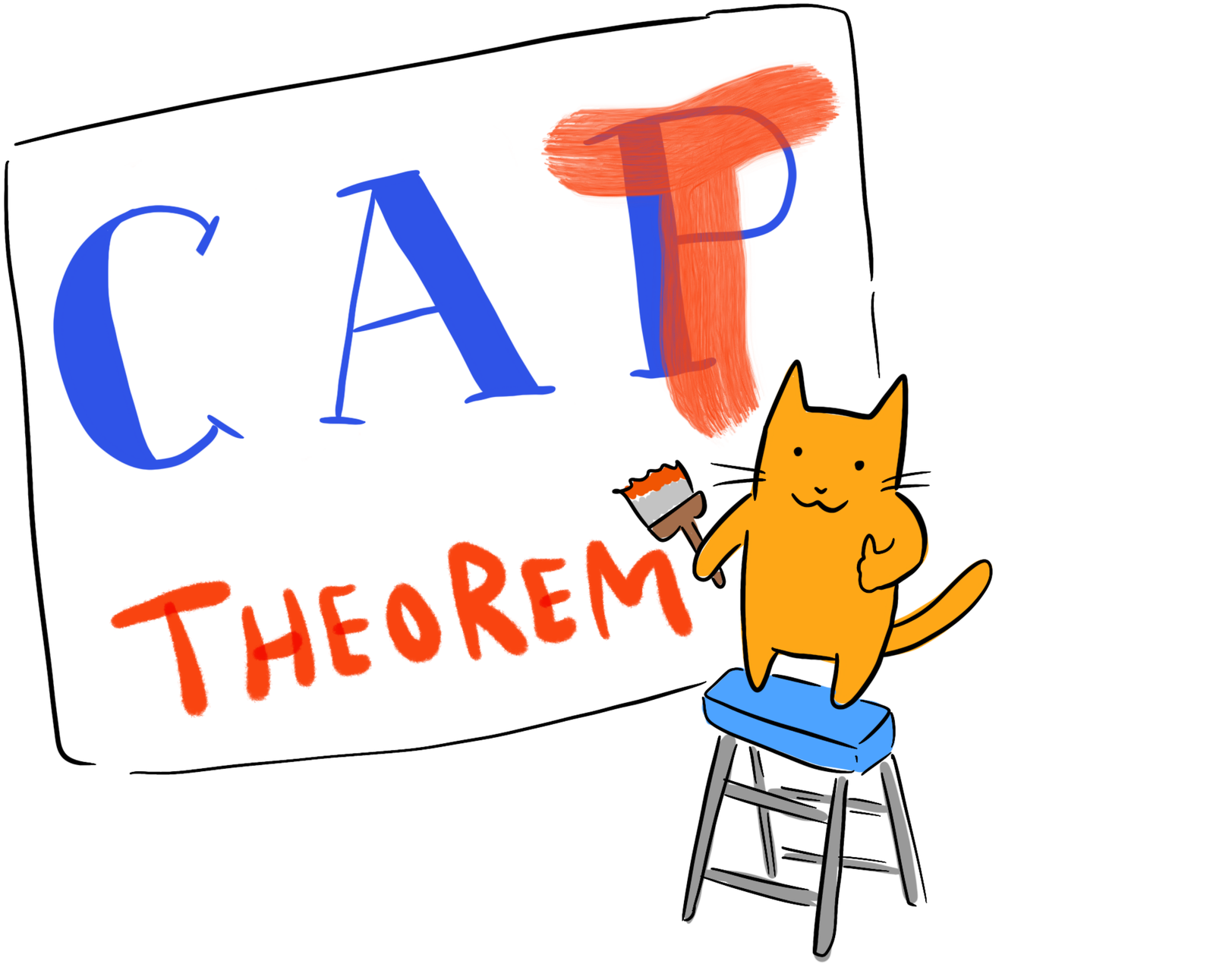
⑤ "Real" Chaos Eng goes beyond Game



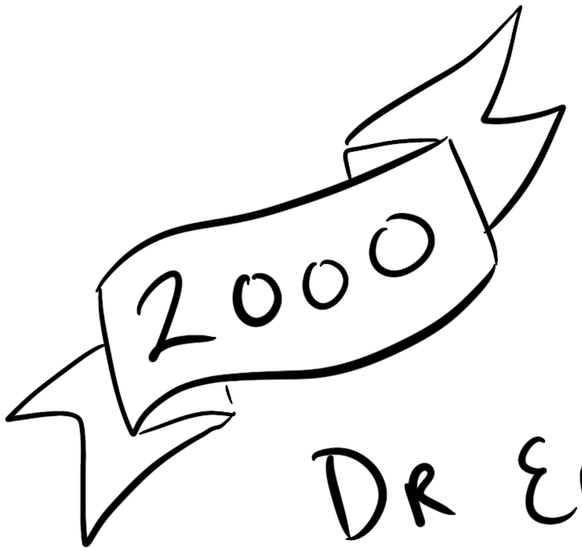
There are lots we can't know. But in distributed computing, we can know one thing:

Shit's gonna fail

@deniseyu21

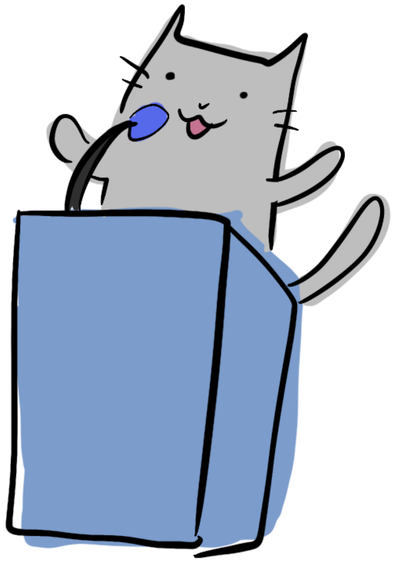


@deniseyu21



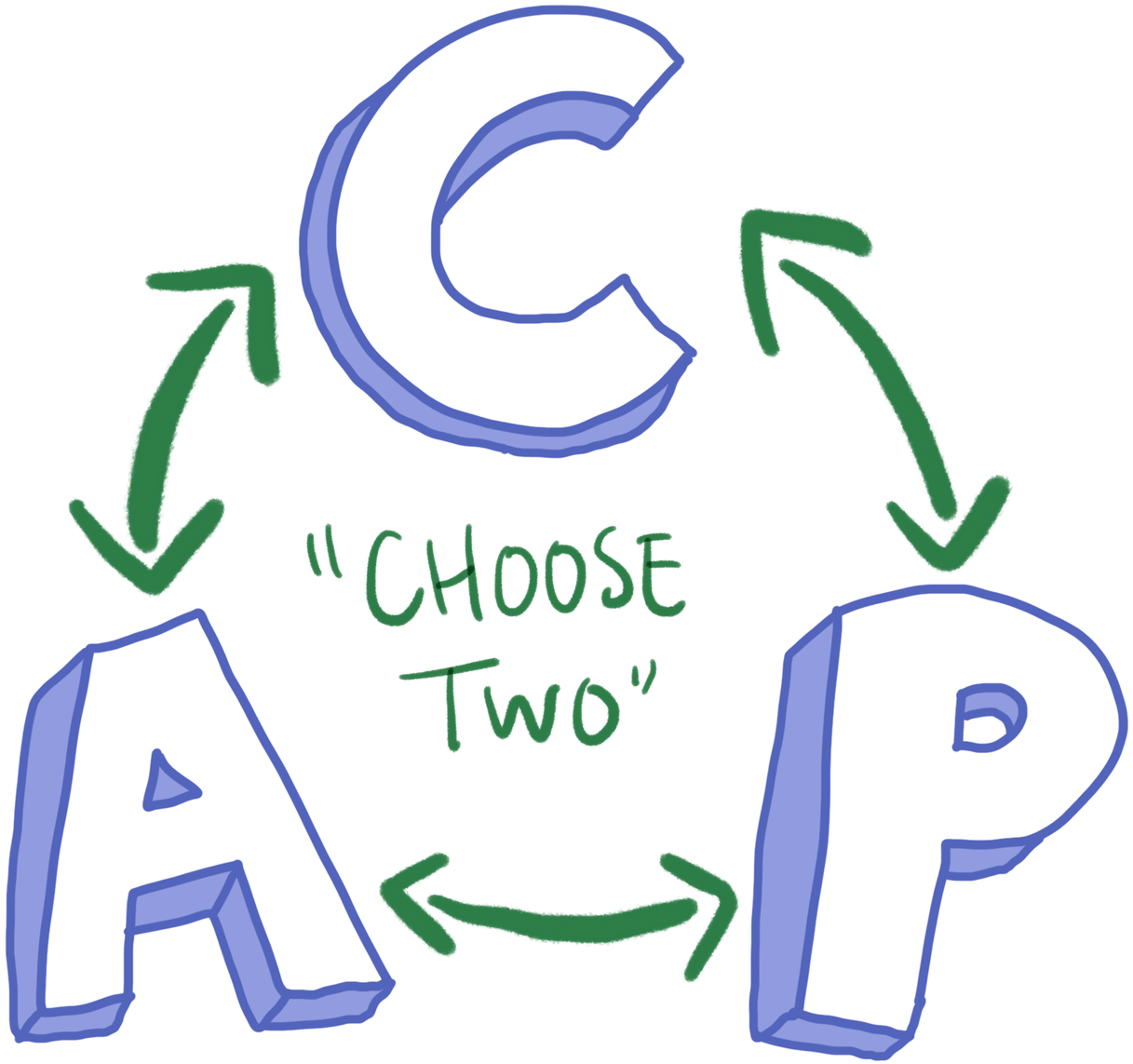
DR ERIC BREWER

"Towards Robust Distributed Systems"

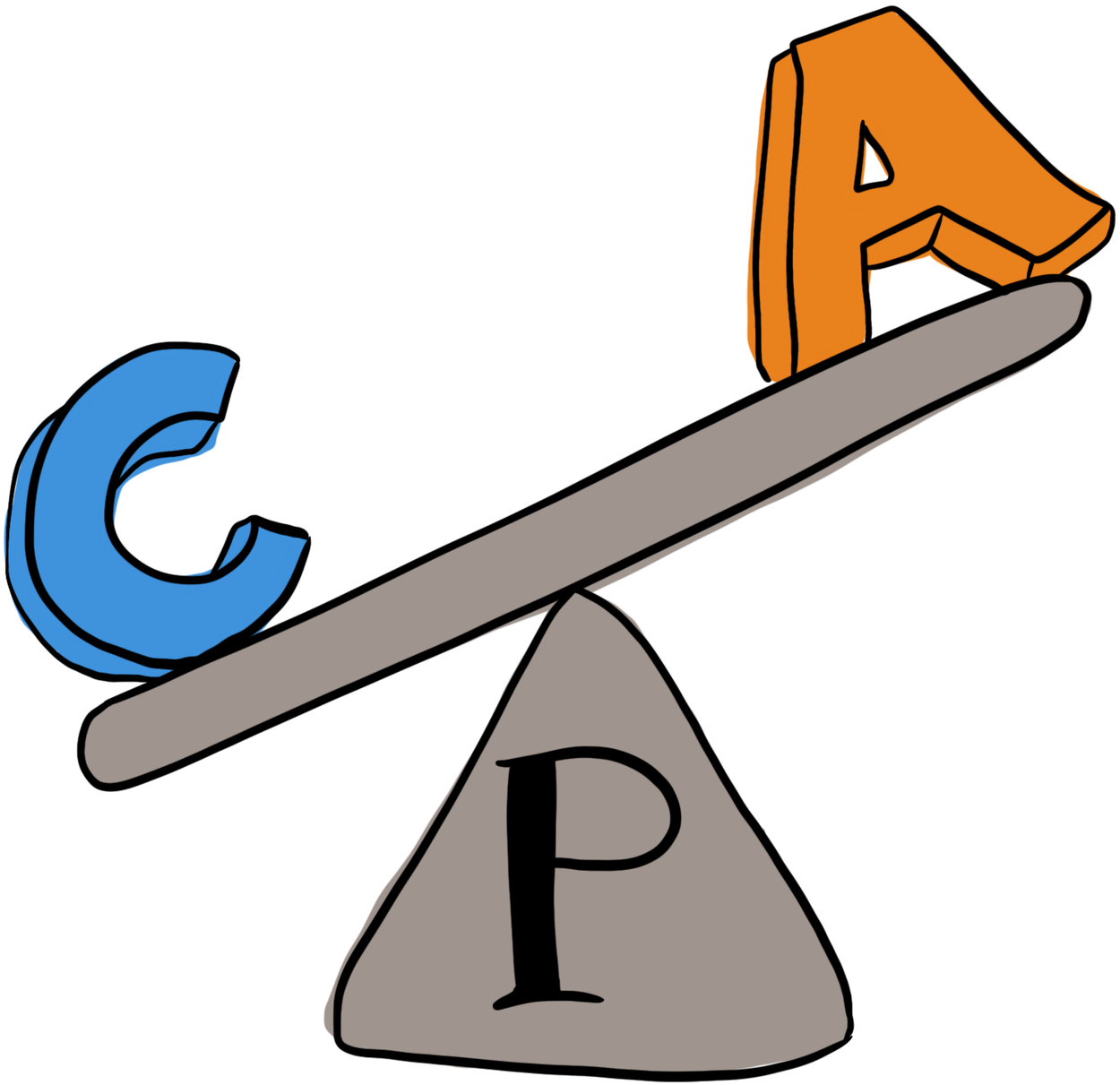


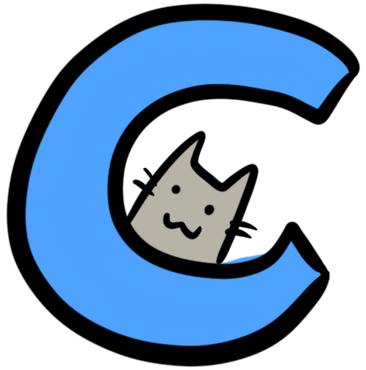
@ Principles of Computing Conf

@deniseyu21


C
"CHOOSE TWO"
A
P


NO!


C
A
P



C IS FOR LINEARIZABILITY

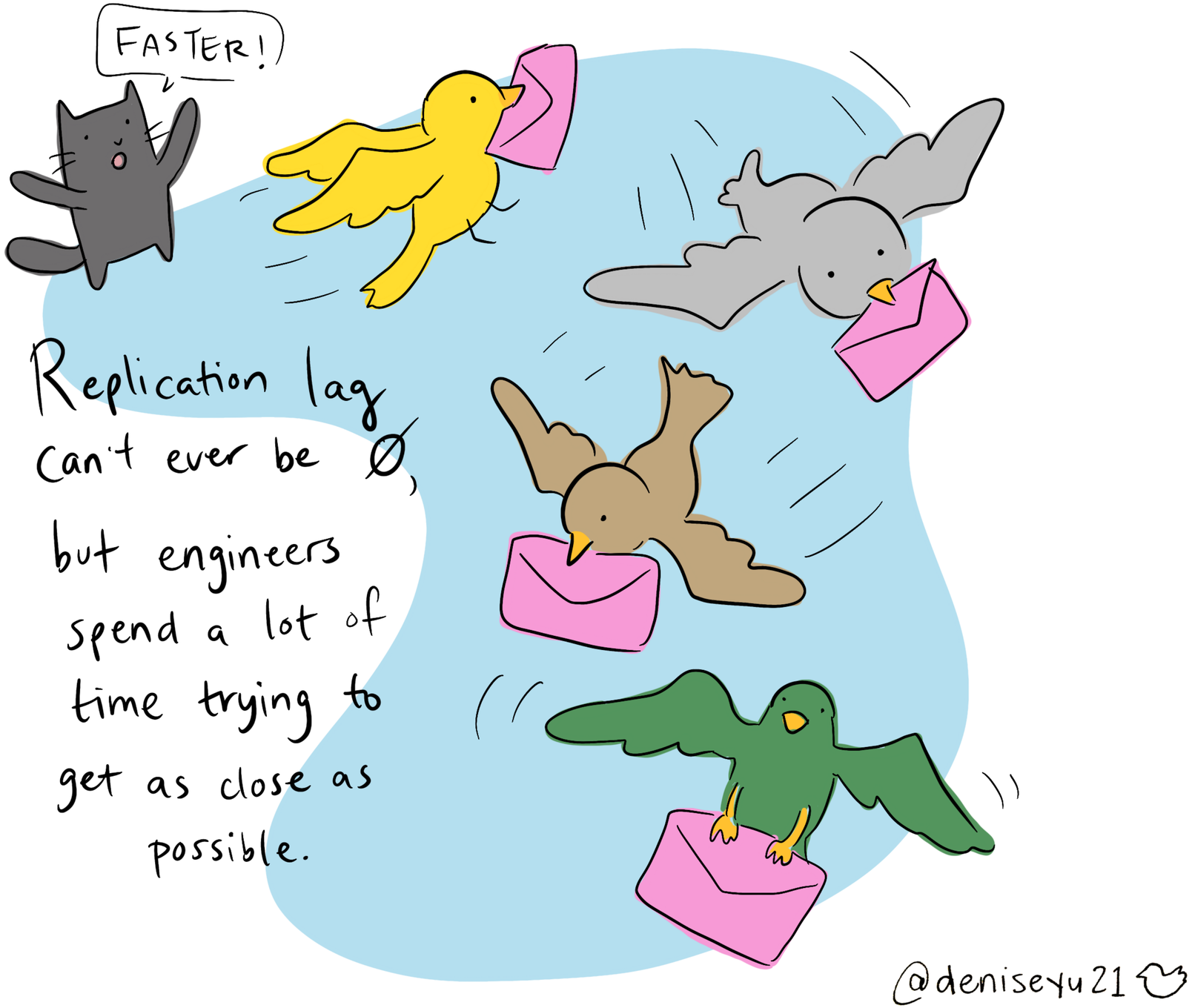
t_0  cat.state = 'hungry'

t_1  cat.state = 'full'

<u>All</u> nodes must have t_1 if anyone showed t_1

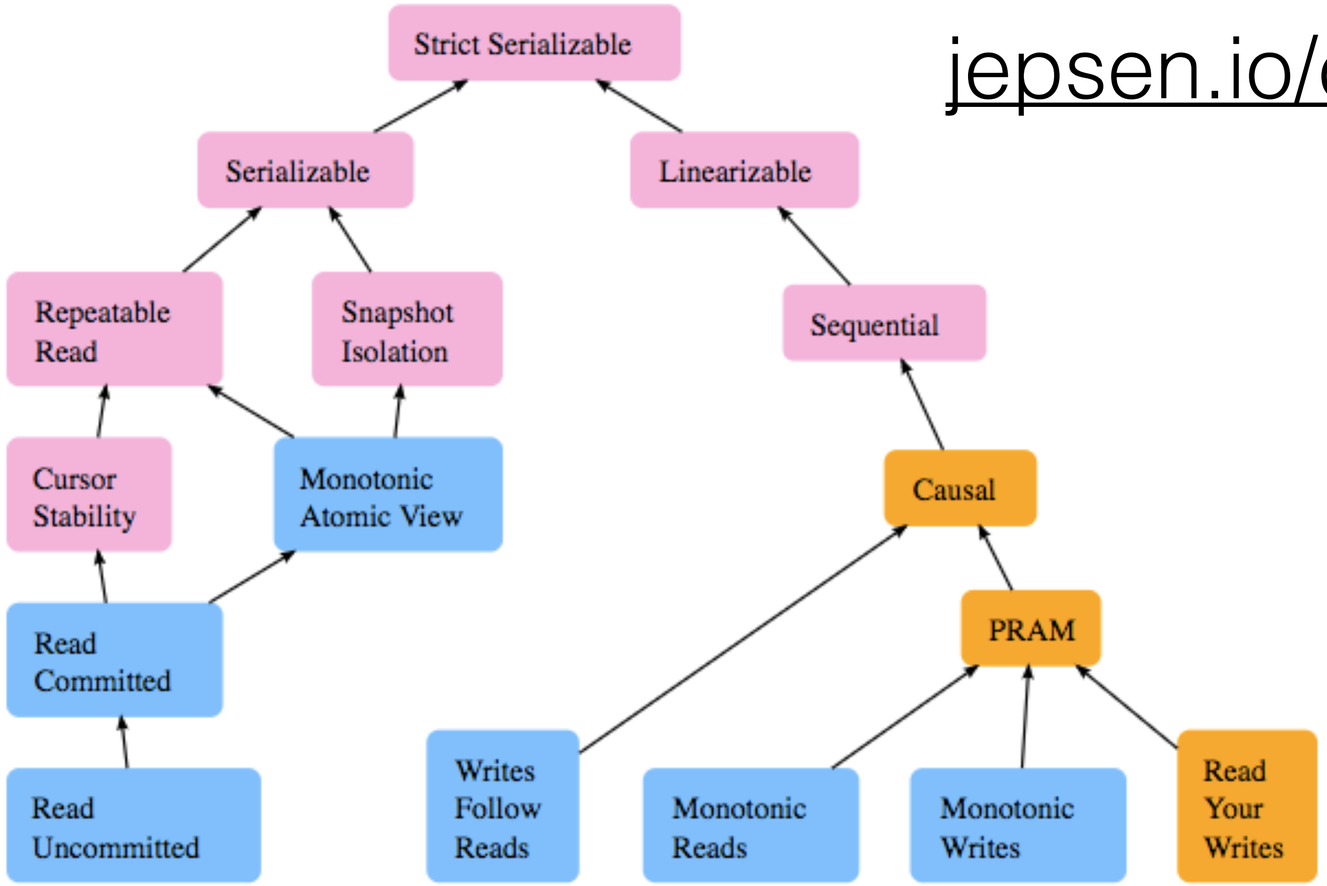
This is really hard! Instant & universal replication.

@deniseyu21


FASTER!
Replication lag can't ever be Ø, but engineers spend a lot of time trying to get as close as possible.
@deniseyu21

(BTW, eventual consistency doesn't count.

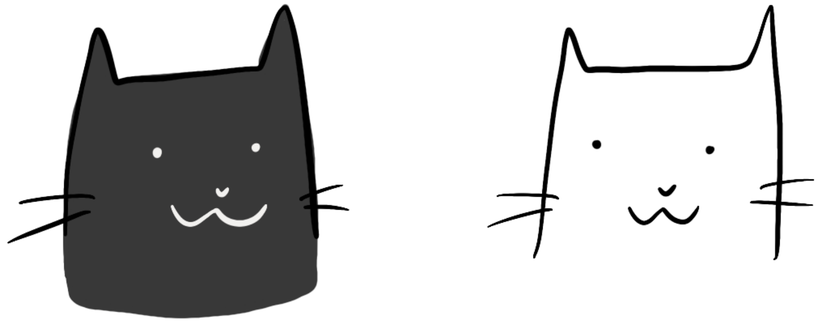
jepsen.io/consistency



Legend

Unavailable	Not available during some types of network failures. Some or all nodes must pause operations in order to ensure safety.
Sticky Available	Available on every non-faulty node, so long as clients only talk to the same servers, instead of switching to new ones.
Total Available	Available on every non-faulty node, even when the network is completely down.

A IS FOR AVAILABILITY



We tend to think of availability as a binary state. BUT — reality is much messier!

Because LATENCY

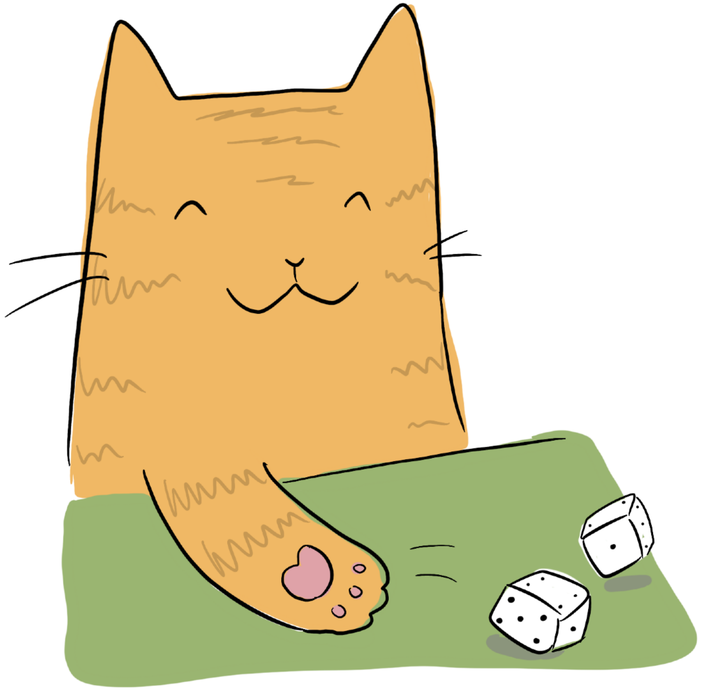
@deniseyu21

How can we know if a node is unresponsive... or just slow?

Network latency wasn't part of the original CAP formulation.

@deniseyu21

Determining a <u>timeout</u> limit is a very scientific process



yu21

P is for Partition Tolerance

Network partitions occur when network connectivity between two datacenters (running your nodes!) is interrupted!

P is for Partition Tolerance

During a partition, your nodes might as well be on opposite sides of a wormhole: there is <u>no</u> way to know what's happening on the other side.

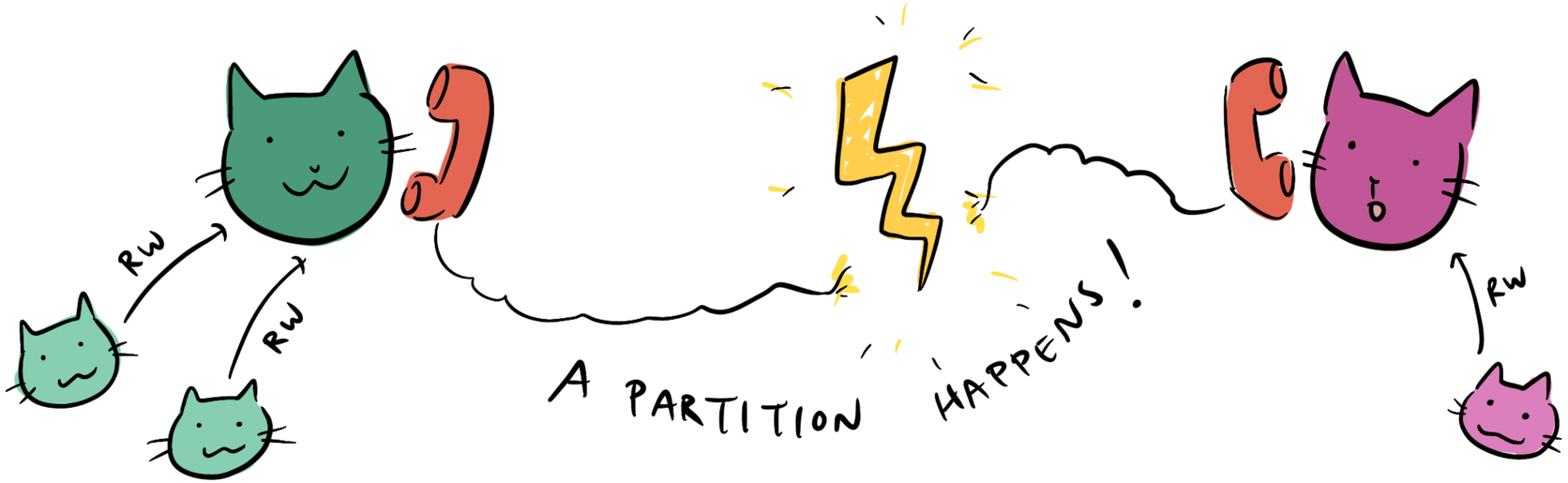
@deniseyu21

CONSISTENCY *

AVAILABILITY

Partition tolerance

PROOF OF CAP THEOREM



OPTION 1

Let clients keep R/W in both sides of split

✗ LINEARIZABILITY

OPTION 2

Stop writing in one side until partition ends

✗ AVAILABILITY

@deniseyu21

PARTITION TOLERANCE

Network partitions are inevitable.

How inevitable?

In the first year of a Google cluster's life, it will experience

5 rack failures

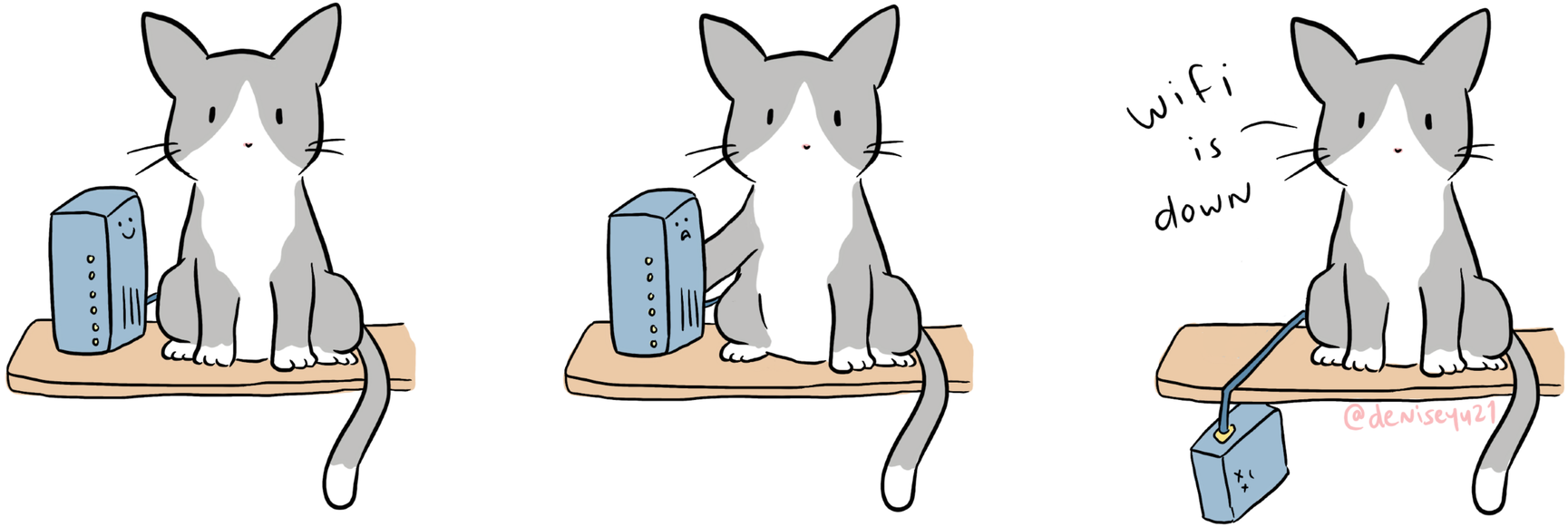
3 router failures

8 network maintenances

(Jeffrey Dean)

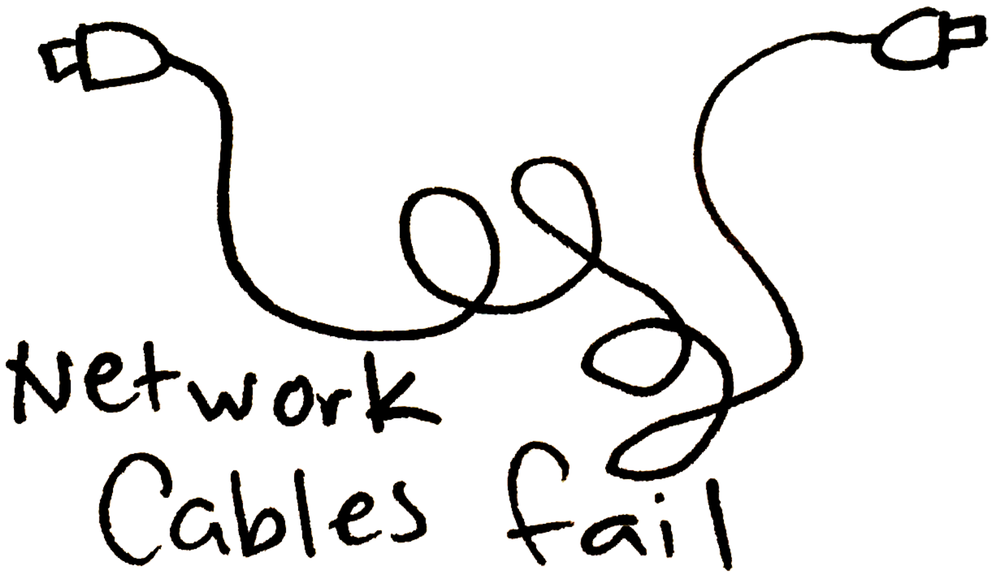
@deniseyu21

Hardware will fail



@deniseyu21

Hardware <u>will</u> fail



@deniseyu21

Hardware will fail

@deniseyu21

POLICY —

It's official: Sharks no longer a threat to subsea Internet cables

First known cable shark attacks were in 1985.

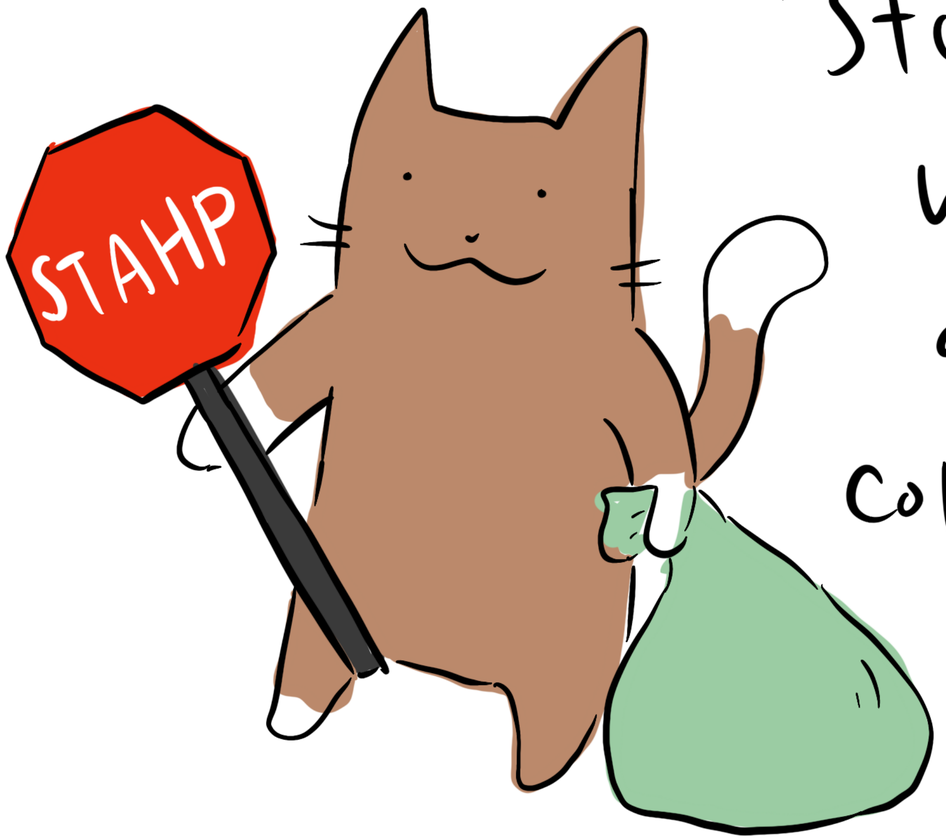
DAVID KRAVETS - 7/10/2015, 5:16 PM

Software will behave weirdly

"Bursty" VMs borrow resources from each other – the Noisy Neighbor Problem

@deniseyu21

Software will behave weirdly

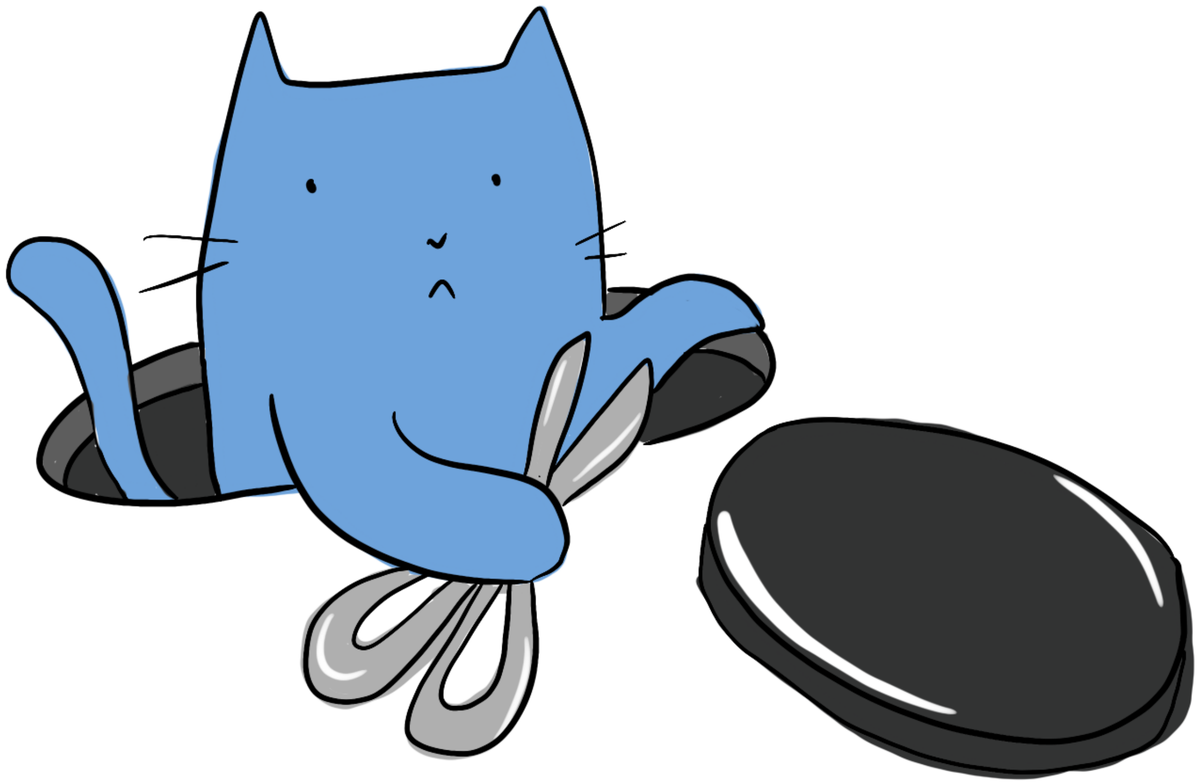


"Stop the WORLD" garbage collection

@deniseyu21

Network glitches

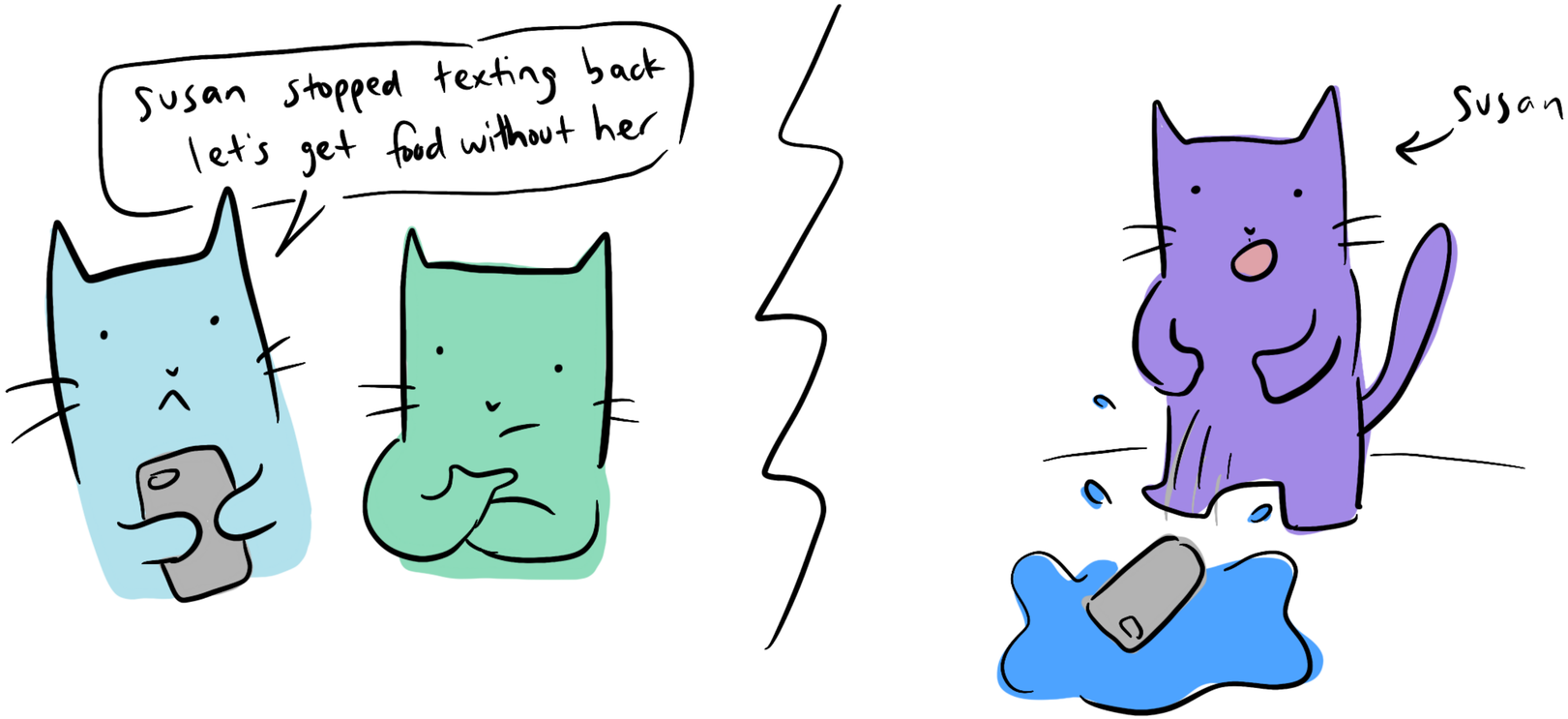
@deniseyu21



@deniseyu21

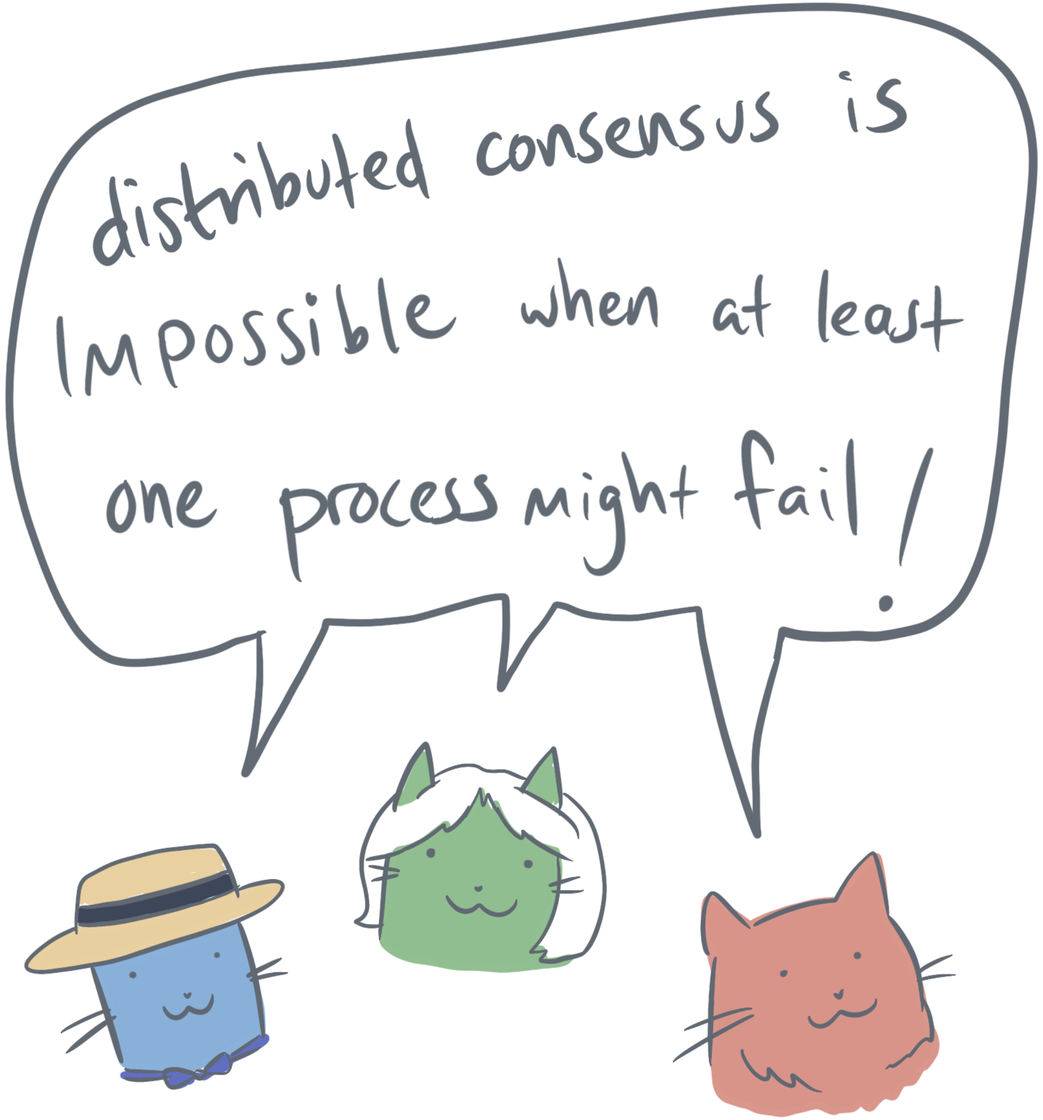
Why does any of this matter?

Some part of every system is always at risk of failing



Fischer Lynch Paterson

Correctness result



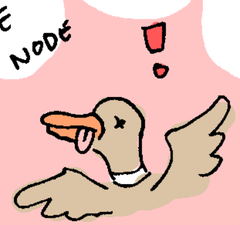
“Impossibility of Distributed Consensus with One Faulty Process, 1985.

to manage uncertainty, we have mitigation strategies

what is AUTOMATIC LEADER FOLLOWER?

LEADER-FOLLOWER* is a common pattern for REPLICATING data across nodes in a cluster. If a follower node fails, it's NO BIG DEAL usually, because other nodes can continue to serve READ requests!

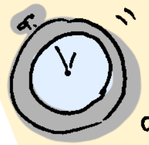
WHAT IF THE LEADER NODE FAILS?



the cluster must initiate a FAILOVER to choose a new leader!

STEP ONE

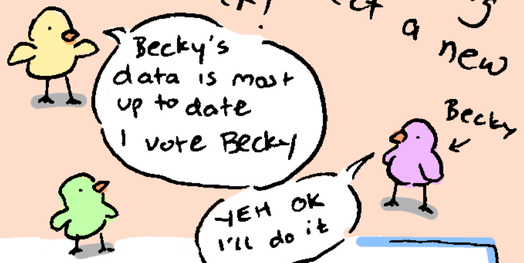
Detect that a node failed! most databases have a default [TIMEOUT (ex. 30s)]



If a node has no activity for this time, it is considered to be offline

STEP TWO

From the remaining followers, elect a new leader!



Some databases have a CONTROLLER NODE who chooses

← He has ONE JOB!

STEP THREE

Tell the world about the new LEADER

Traffic will be routed to the new leader for all future write requests

* You may have encountered "master-slave replication". I prefer not to use this term, because as technologists, our language matters. We should choose terms that are inclusive & don't cause harm.

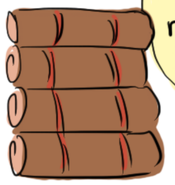
@deniseyu21
sketch-ops.tumblr.com

how does RAFT work?

RAFT IS A CONSENSUS ALGORITHM

used in many projects in the real world - ex. etcd

a process for getting multiple machines to "agree"!



- RAFT doesn't stand for anything. It's a bunch of logs tied together.

WHY? Distributed systems come with uncertainty. Achieving consensus is important so data can be replicated!

WHAT ABOUT NETWORK PARTITIONS?!

In a partition, if the leader falls on the smaller side, a new leader is elected within the majority side. When the partition ends, messages received by the minority side are discarded, and those nodes converge their state to match the majority's.

WHEN A NEW WRITE HAPPENS:

(Any node can write!)

Message is forwarded to leader node

Message is queued

Followers copy leader

Leader "commits" the new data

& followers increment their state counter

@deniseyu21 with @mt165

… BUT WHY??

@deniseyu21

and his hair was perfect
@palvaro

another highlight:

Me: so... what is the hard thing about distributed systems? if you had to pick one word...

Student 1: uncertainty?

Me: *beaming* GOOD. *writes it on the board*

Student 2: Docker

Student 1: actually that's better, take mine off

6:33 PM · Oct 20, 2019 · Twitter Web App

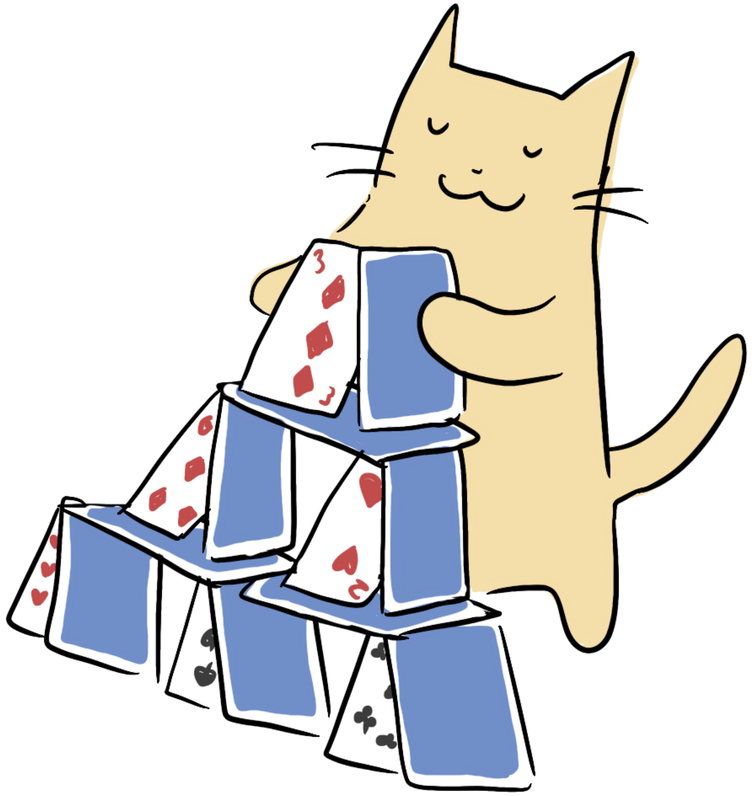
Woods' theorem:

"as the complexity of a system increases, the accuracy of any single agent's own model of that system decreases rapidly."

"Coping With Complexity: the psychology of human behavior in complex systems." Dr. David Woods, 1988.

BUILDING MENTAL MODELS

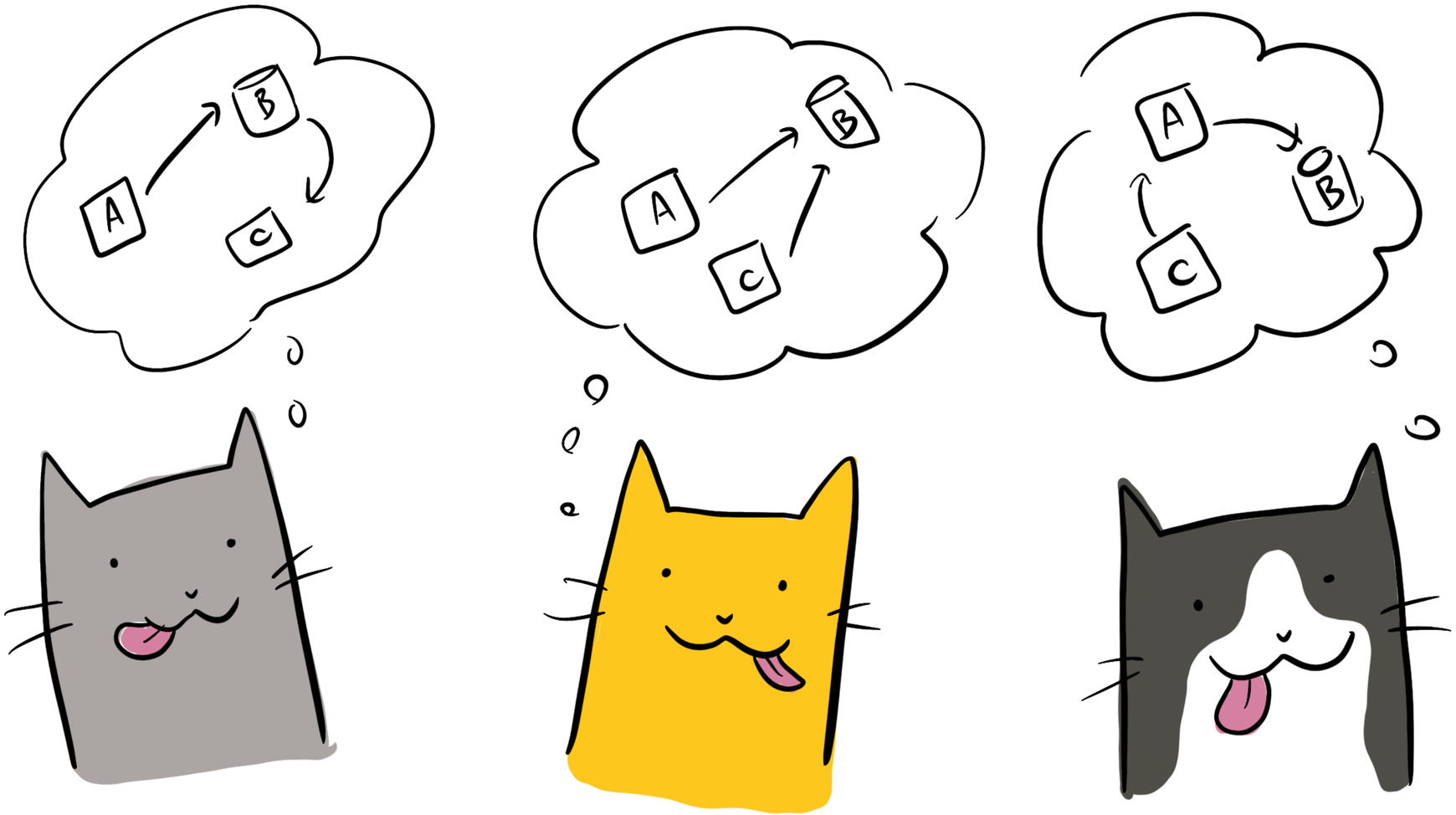
in theory:



in practice:

@deniseyu21

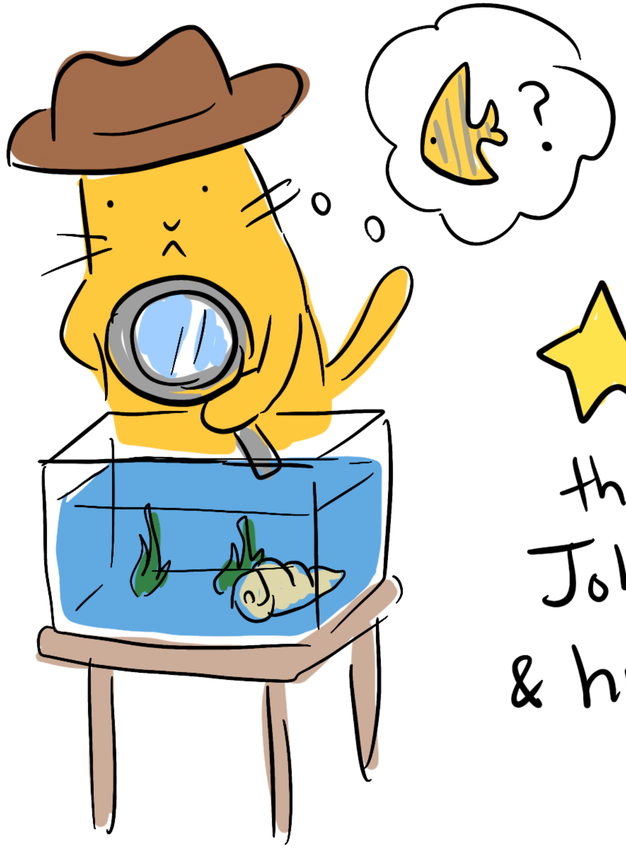
Mental models are hard to compare, which makes them hard to calibrate



Mental models are hard to compare, which makes them hard to calibrate

@deniseyu21

INCIDENT ANALYSIS



This is the work of John Alls & his team

is particularly great for mental model calibration

@deniseyu21

Blameless discussions optimized for learning

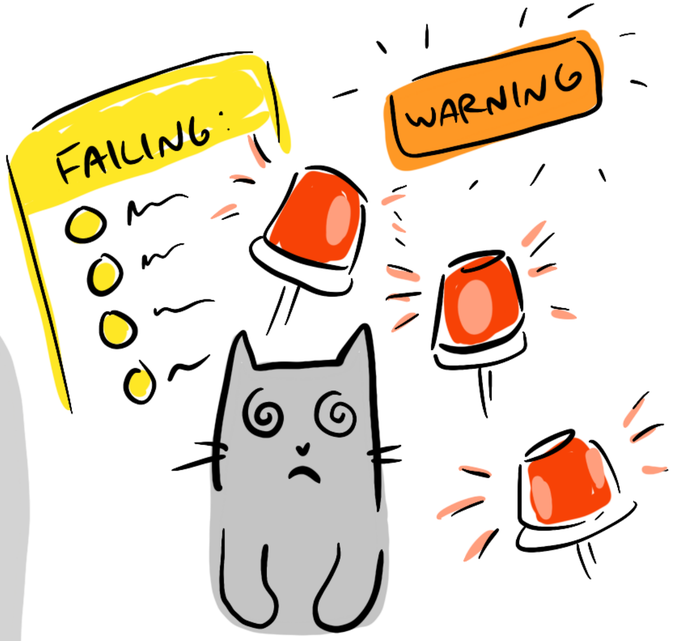
@deniseyu21

Don't accept HUMAN ERROR as the root cause. Dig deeper!

@deniseyu21

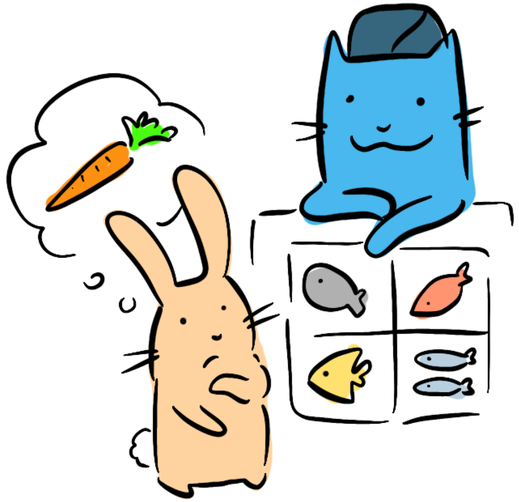
Unintuitive design?

☐ check if you do not not not not not not not wish to receive emails



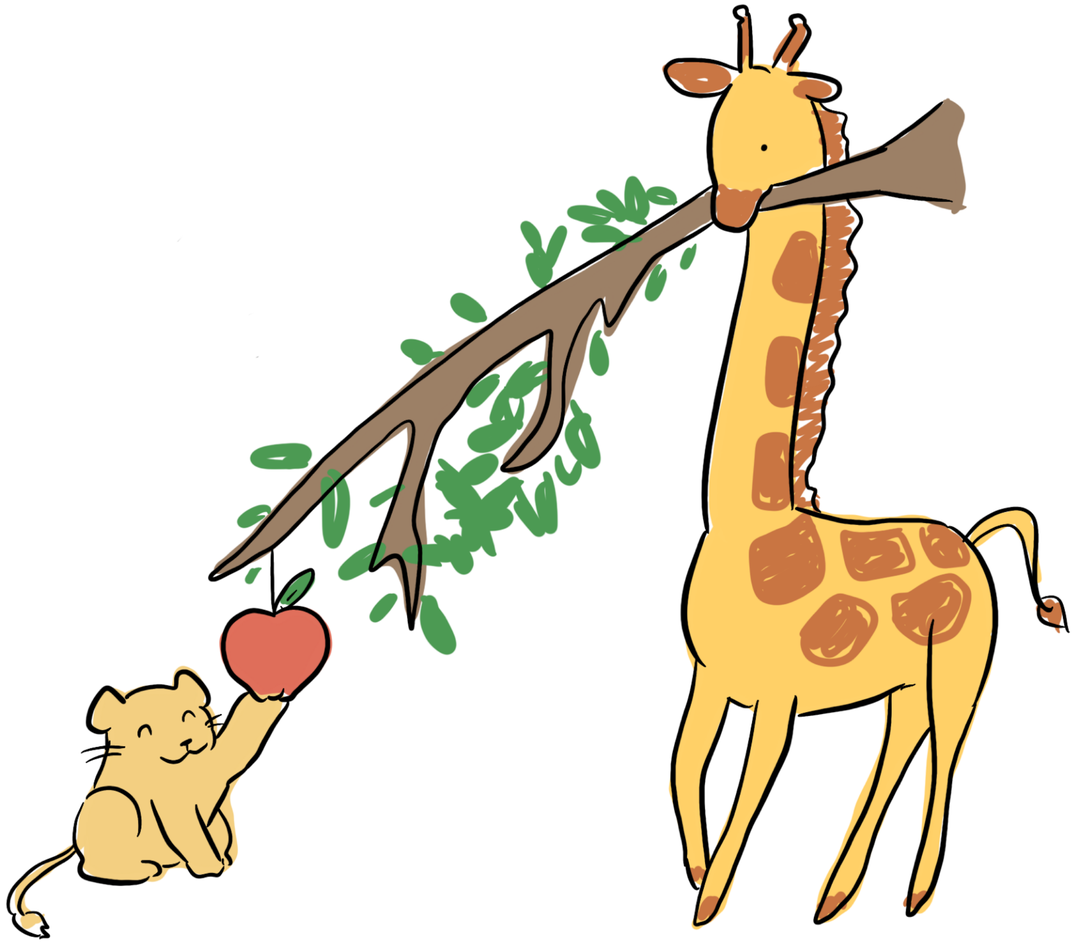
Alert fatigue?

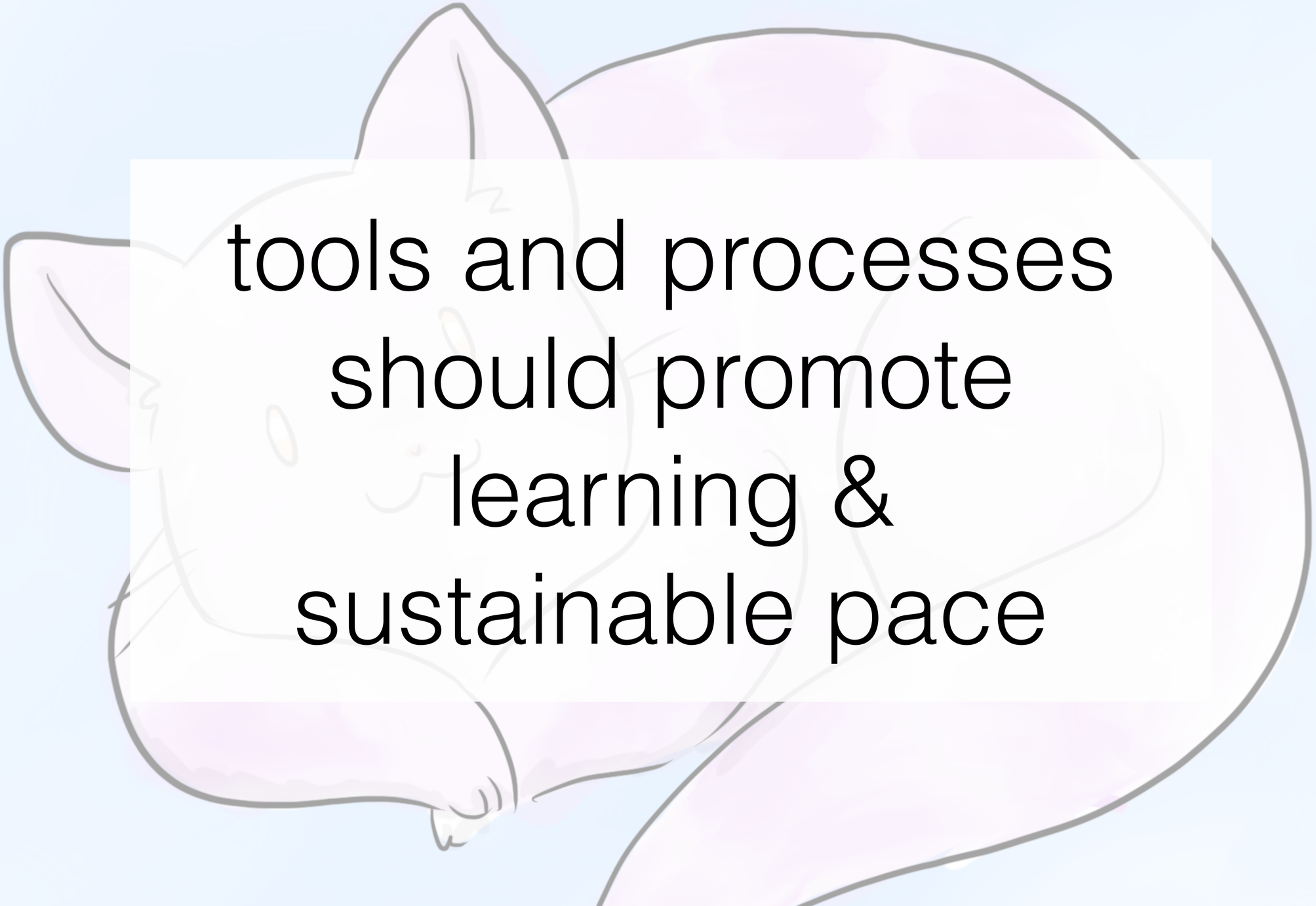
Not understanding the users' assumptions and needs?



@deniseyu21

design your systems for humans, not machines





tools and processes should promote learning & sustainable pace

be kind to each other

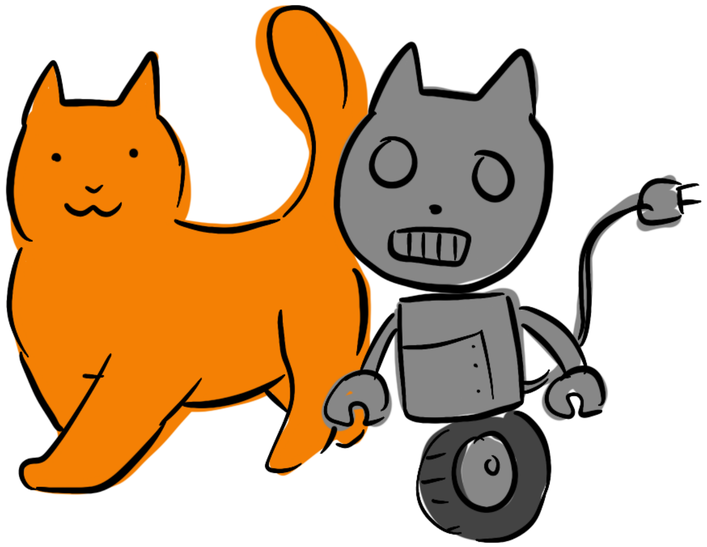
#HugOps

our superpower, as humans:

empathy

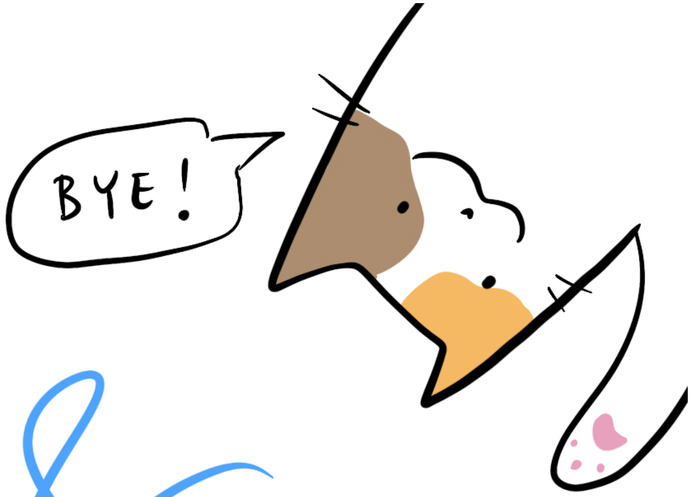
@deniseyu21

we owe it to our end users &
our teams to understand
& design for the whole system



including the fleshy human parts.

@deniseyu21



Slides & References

deniseyu.io/lisa