2

1

Pulse. Temperature. Blood Pressure.

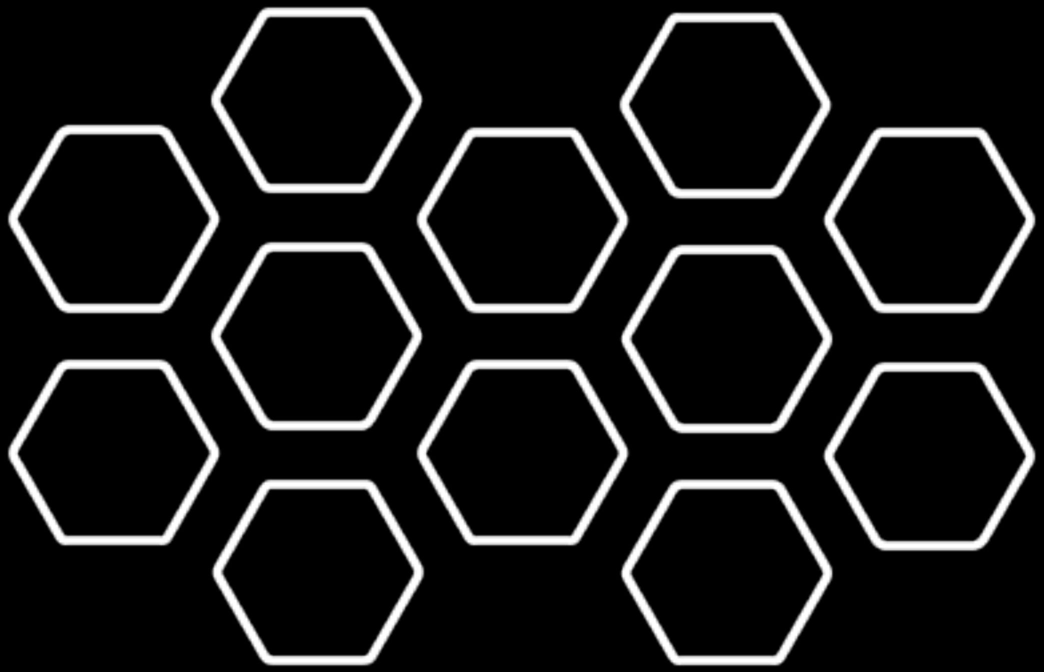
Micro-service systems have emergent properties too.

nearForm

nearForm & micro-services

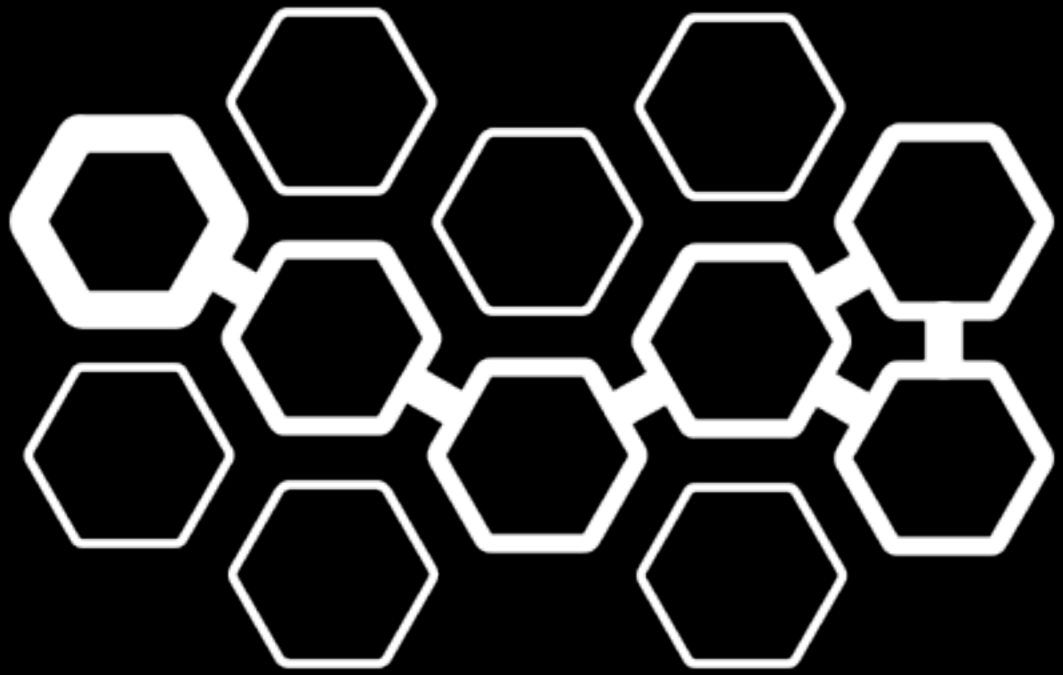
50+ production systems.

The good, the bad, and the ugly.



What are micro-services?

Independent processes that exchange messages.

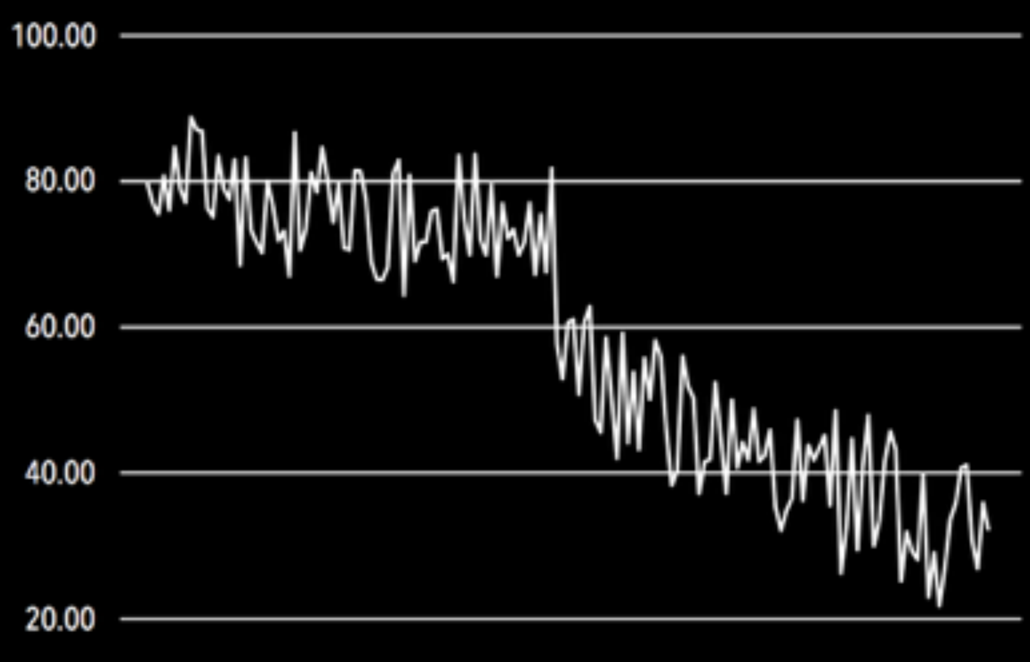


Messages are fundamental.
Message behavior has emergent properties.

Message flow rate.

Easy to measure. Tells you a lot.

Independent of services.



Deploy a new micro-service.

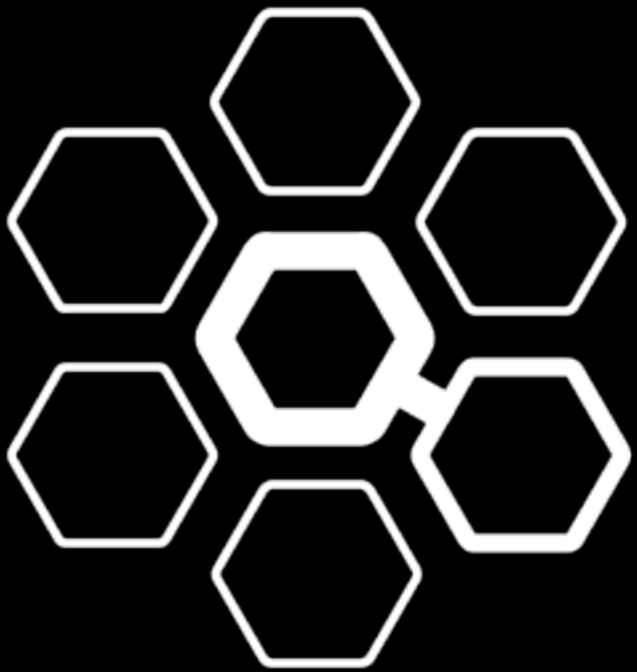
Does the new version break anything?

To measure changes to services, measure changes to message flow rates.

Micro-service message patterns.

What to measure?

Here’s what we’ve found useful…

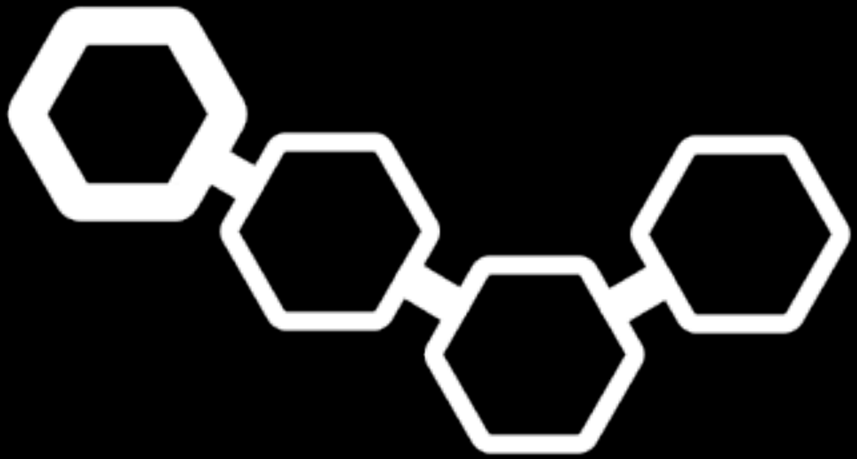


Actor.

A pool of services share message load evenly (round-robin, say).

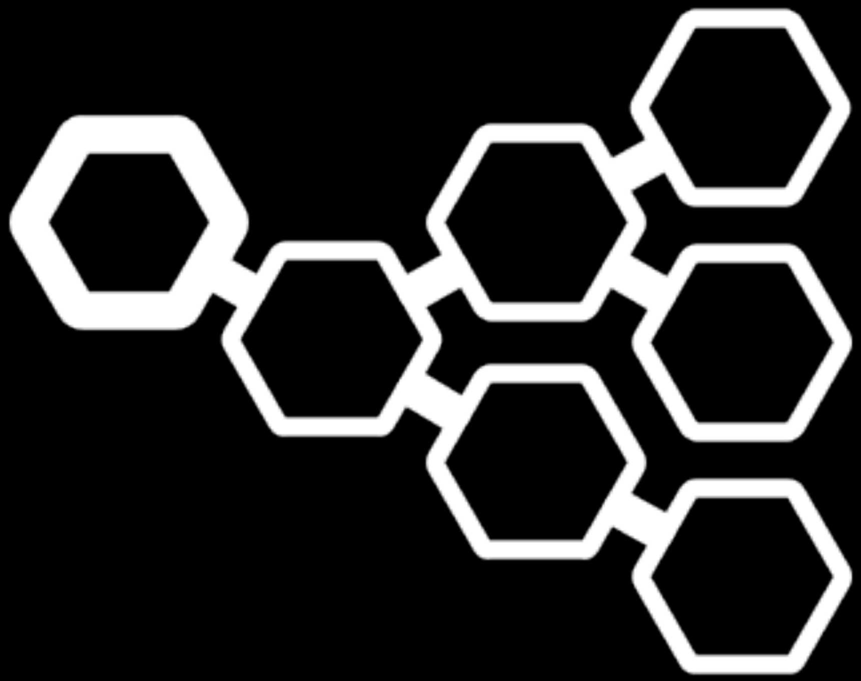
Subscriber.

Many services all listen for the same set of message types.



Chain.

An initial message causes a chain of serial message steps.



Tree.

An initial message causes a flowering of child messages.

Why?

It's the risk, stupid!

Monoliths put everybody in danger.

Risk.

Reduction can be measured.

Leave elimination to Machiavelli.

Our "best practices" for risk.

Unit tests; code reviews; standards.

Do we have good measures?

We've just made things worse!

Microservices also have emergent failure modes.

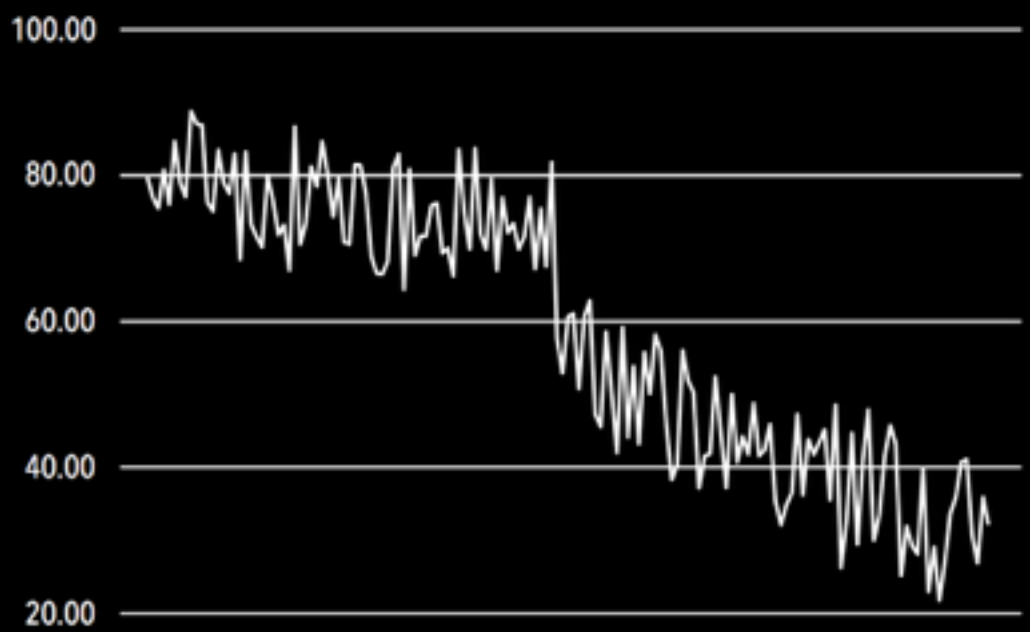
Formal Methods

“Best Practices”

Measurement

Let's attack both sides.

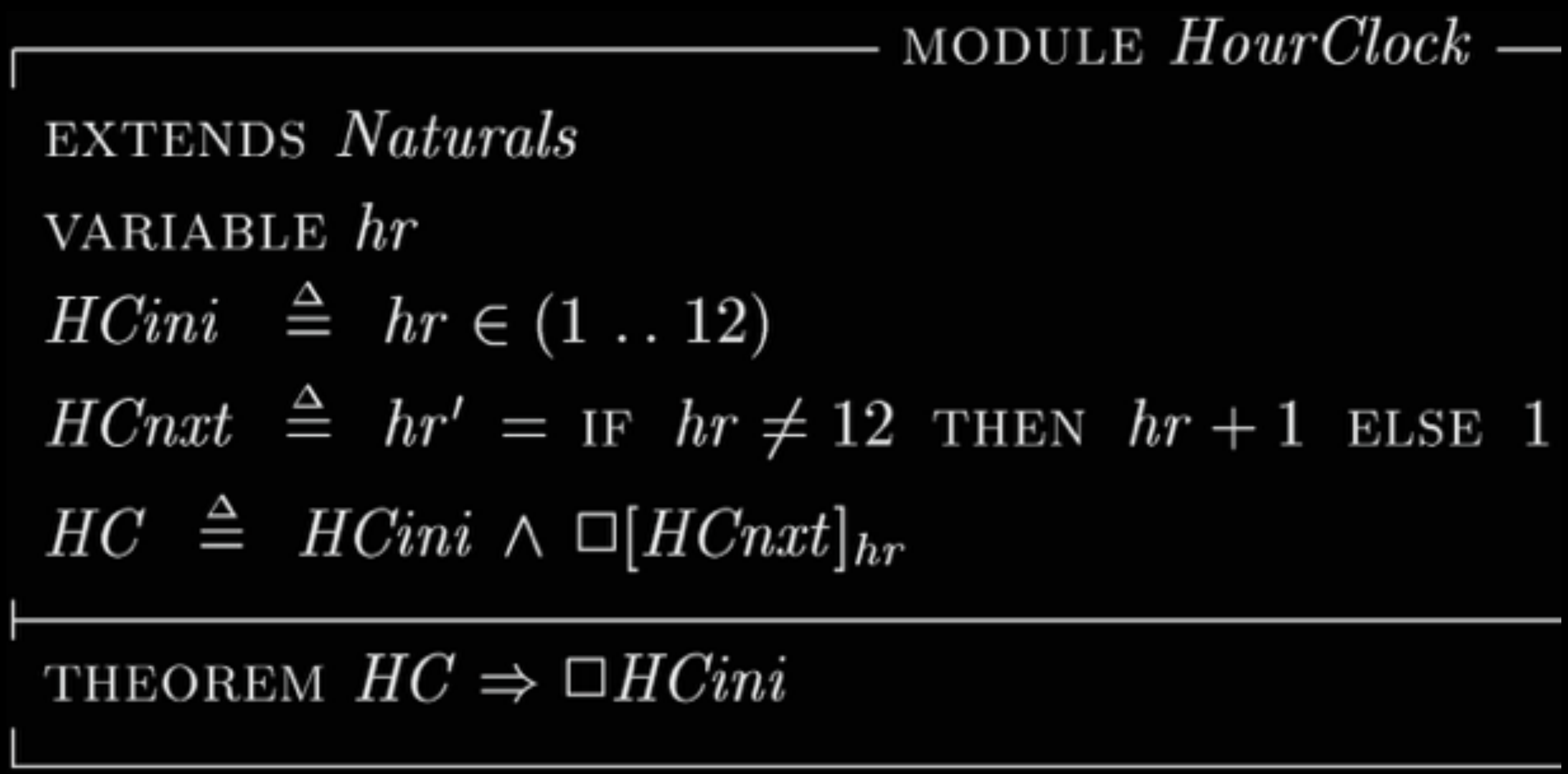
And we'll find a way to connect both attacks.



Dynamic measurement.

Measure health of the system.

Exposes unknown unknowns.



MODULE *HourClock*

EXTENDS *Naturals*

VARIABLE *hr*

$HCini \triangleq hr \in (1 \,..\, 12)$

$HCnxt \triangleq hr' = \textsc{if}\ hr \neq 12\ \textsc{then}\ hr + 1\ \textsc{else}\ 1$

$HC \triangleq HCini \land \Box[HCnxt]_{hr}$

THEOREM $HC \Rightarrow \Box HCini$

TLA+

Leslie Lamport *

Formal methods.

Correctness proofs are impractical.

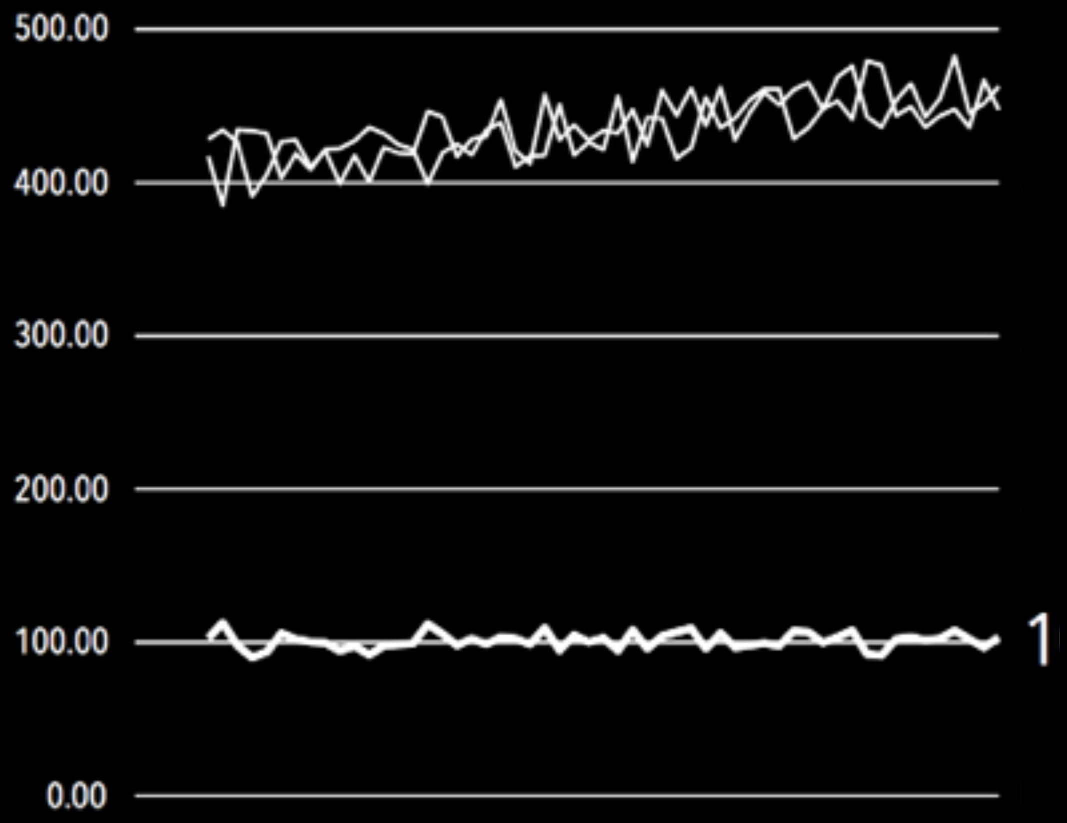
Incomplete execution traces? FTW!

* http://research.microsoft.com/en-us/um/people/lamport/tla/tla.html

Invariants.

Some things should never change.

Measure them to make sure!



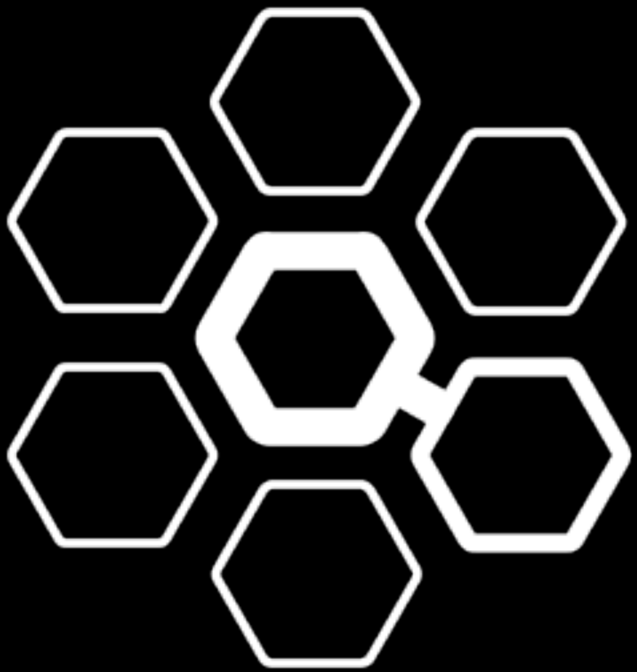
Example.

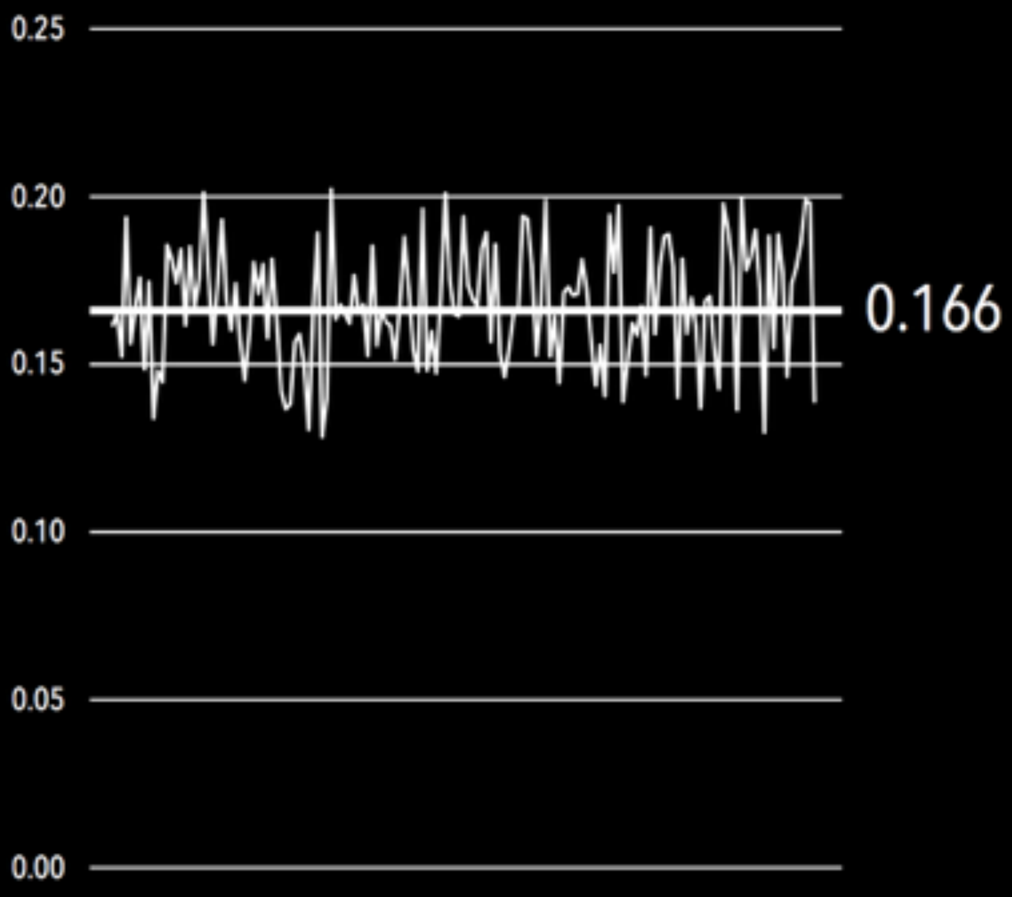
E-commerce shopping cart.

add-item msgs == *sales-tax* msgs

Be practical!

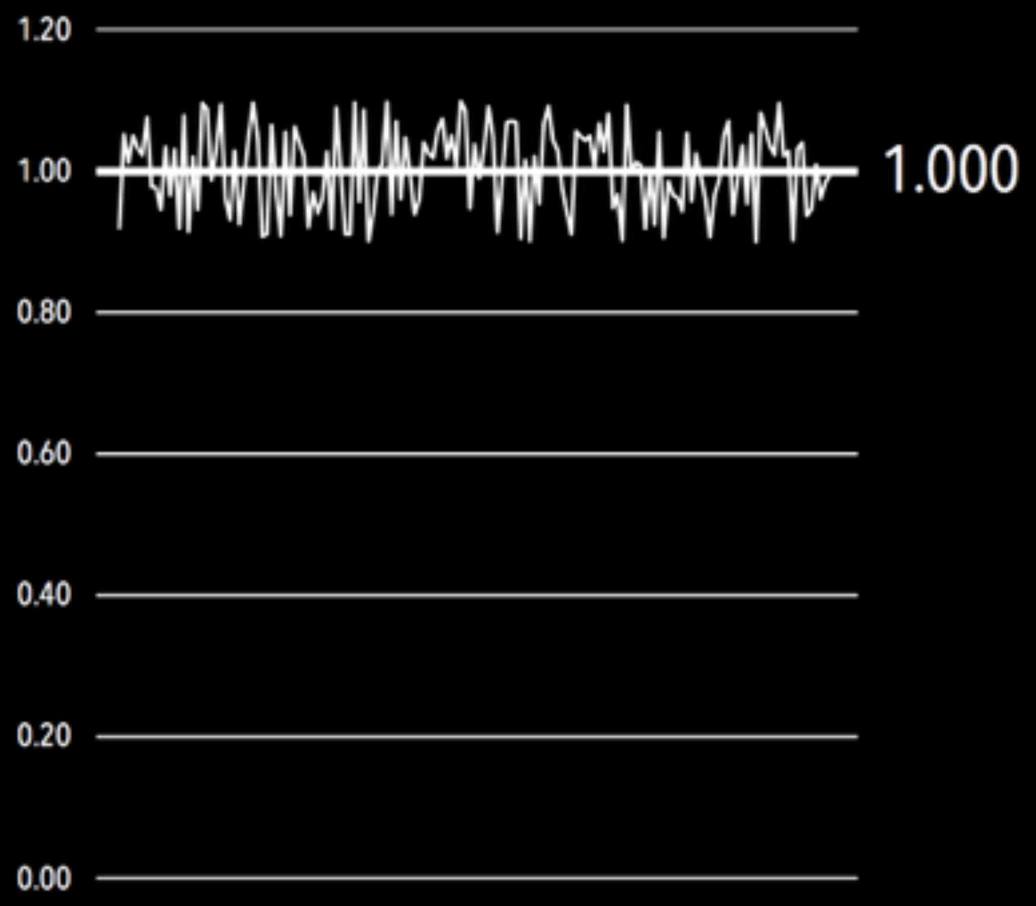
Finding invariants is hard. Use the microservice patterns to cheat.





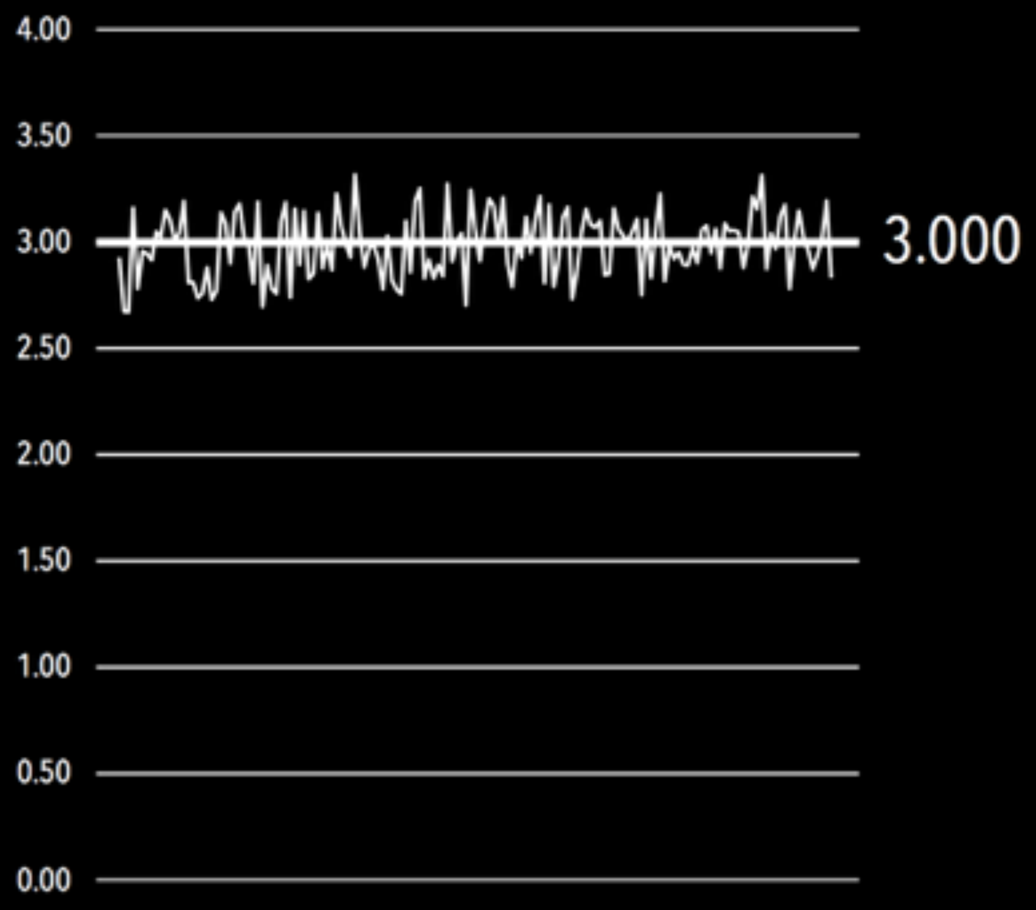
Actor.

n actors means each actor sees $^1/_n$ of the messages.



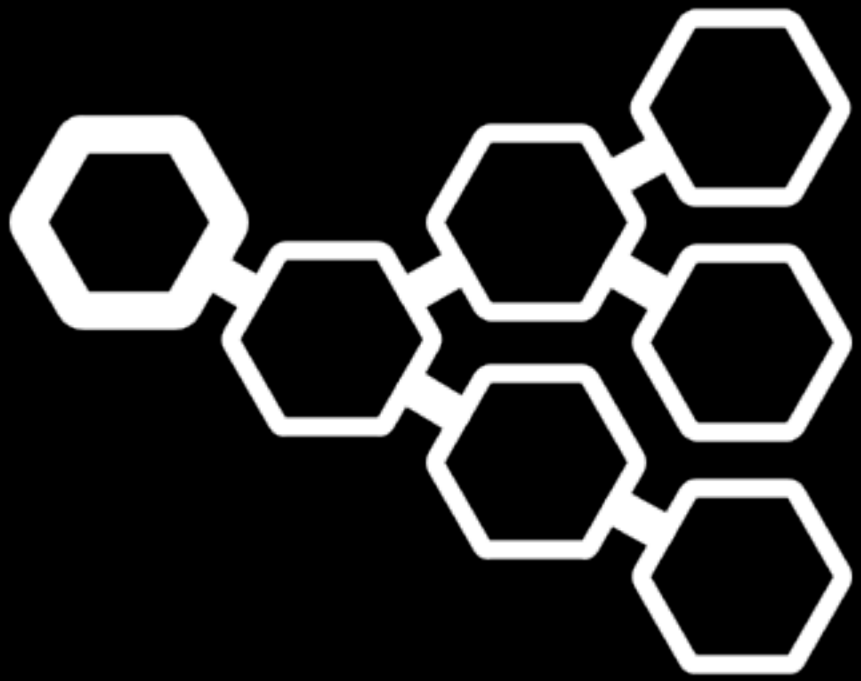
Subscriber.

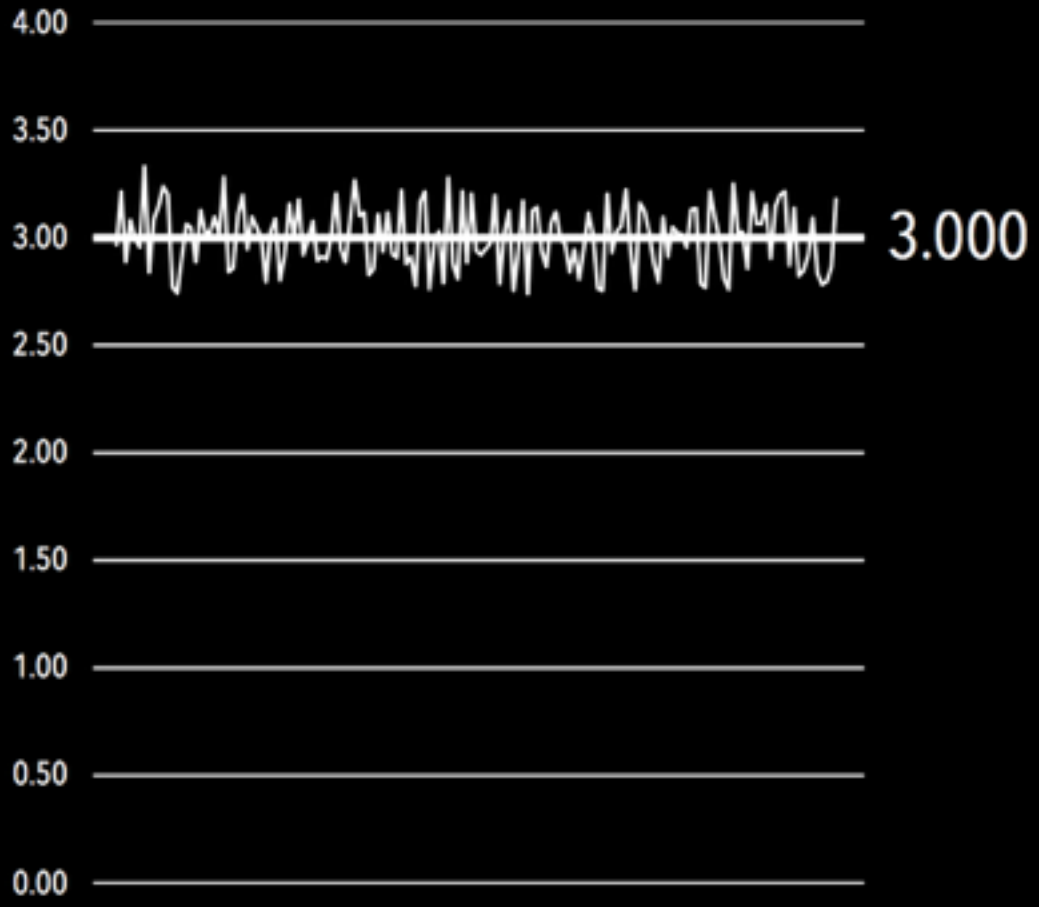
n subscribers means each sees n messages.



Chain.

n inbound messages over k links means nk chained messages.





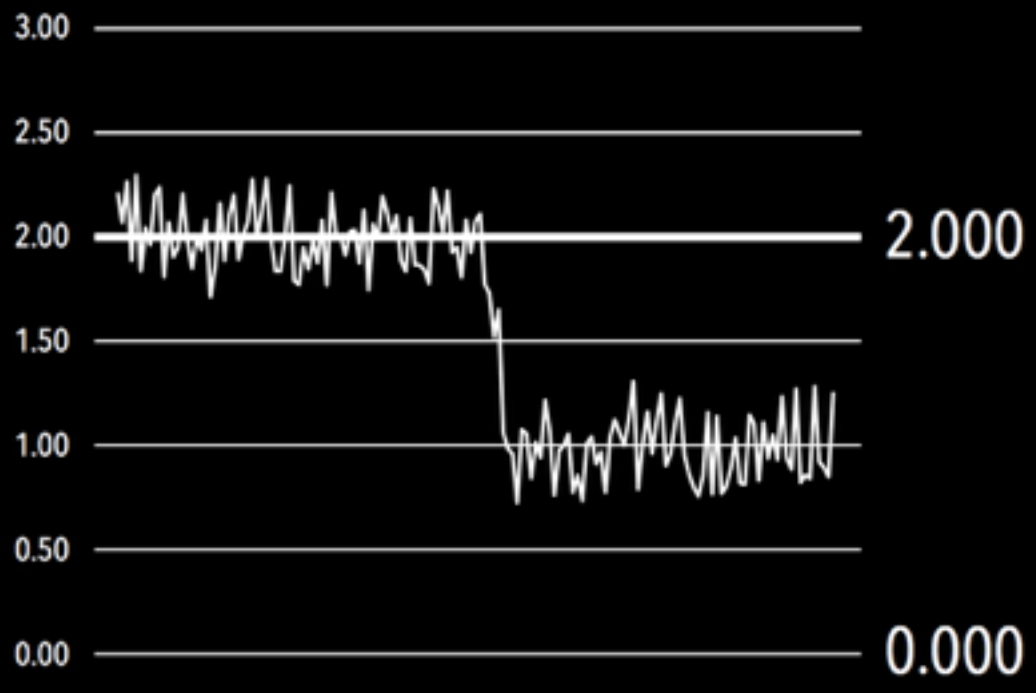
Tree.

n inbound messages over k leaves means nk leaf messages.

“ask not what can go wrong,
ask what must go right...”
Chris Newcombe, AWS*

Look for cause/effect relationships.
These are by design! Validate your message patterns in production.

* http://research.microsoft.com/en-us/um/people/lamport/tla/amazon.html



When should you rollback?

Invariants should be the same before and after deployment.

Is the system correct?

Business rules are invariants too!

Express as message relationships.

Are you about to be blindsided?

Combine individual indicators to get a deeper measure of risk.

Measure what counts.

Find invariants. Measure them.

Follow: senecajs.org

nearForm

Thank You!

Richard Rodger @rjrodger

nearform.com