



PAT HELLAND AND ME

HOW TO BUILD STATEFUL DISTRIBUTED APPLICATIONS THAT CAN SCALE ALMOST INFINITELY

PAT HELLAND

AND ME



SEAN T. ALLEN

VP OF ENGINEERING AT WALLAROO LABS
MEMBER OF THE PONY CORE TEAM
AUTHOR OF "STORM APPLIED"
LOVER OF FRENCH STREET ART

@SEANTALLEN
@WALLAROO LABS
@PONYLANG



DATABASES

APPARENTLY, I LIKE TO
STICK THEM IN THINGS...



SOME AXIOMS...

**TO SCALE INFINITELY,
WE HAVE TO SCALE HORIZONTALLY**

**TO SCALE INFINITELY,
WE MUST AVOID COORDINATION**

**DISTRIBUTED TRANSACTIONS ARE
A FORM OF COORDINATION**

**THEREFORE...
TO SCALE INFINITELY,
WE CAN'T USE TRANSACTIONS**

WELCOME TO DISTRIBUTED SYSTEMS!

0, BY THE WAY, ALL THE
RULES HAVE CHANGED



WHAT IS SCALING?

MORE AND MORE THINGS

BUT, THEY DON'T GET
BIGGER. THERE'S JUST...

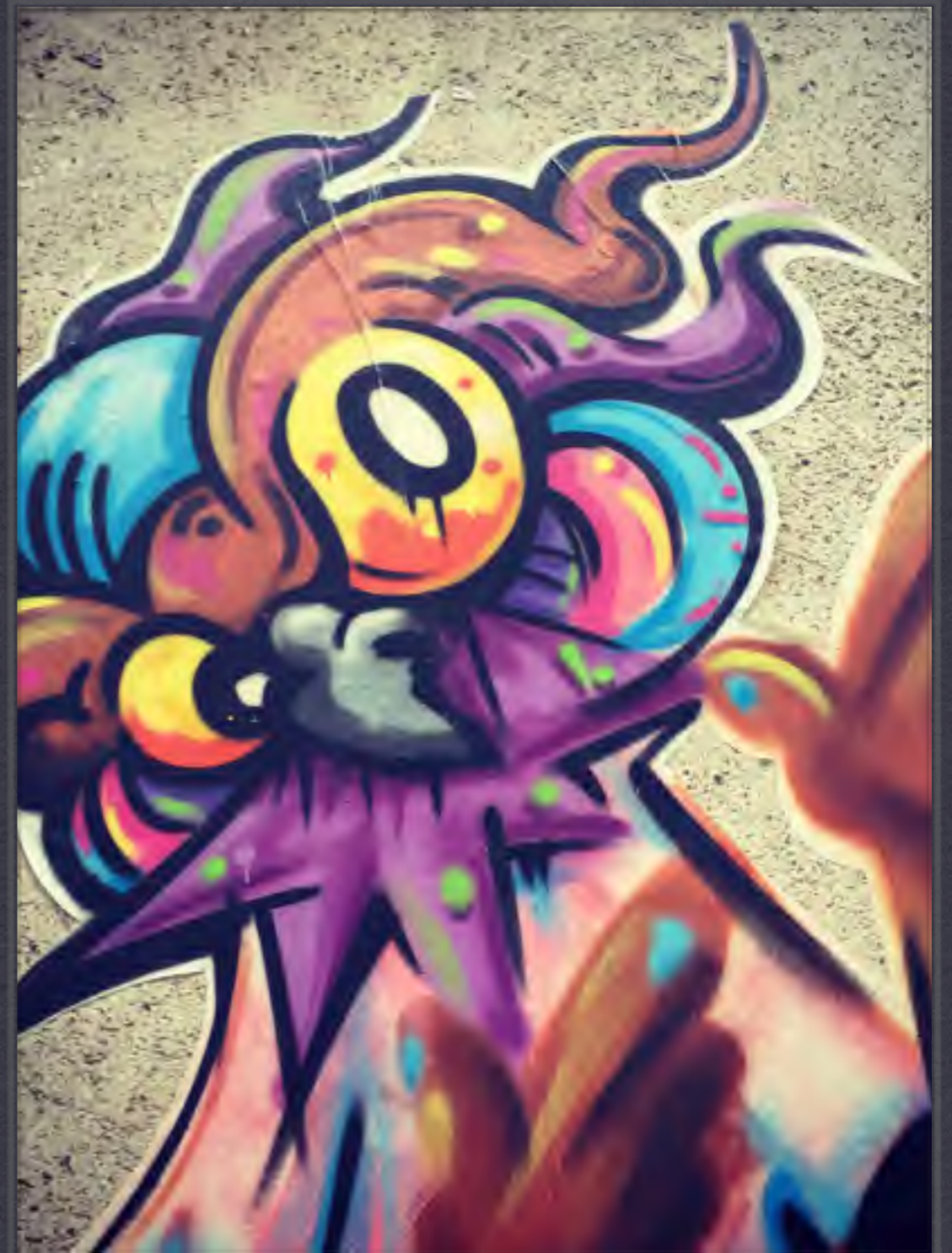
MORE OF THEM. LOTS MORE.



WE SCALE ENTITIES

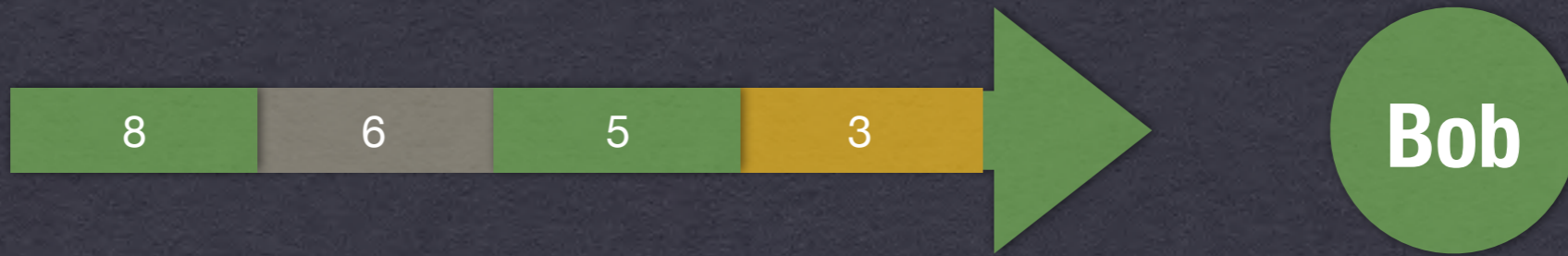
ENTITIES:

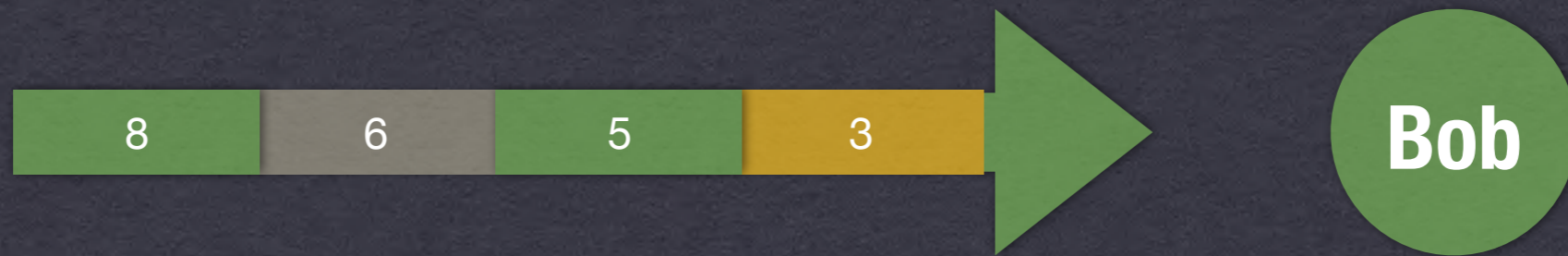
LIVE ON A SINGLE MACHINE
AND ARE MANIPULATED
INDIVIDUALLY

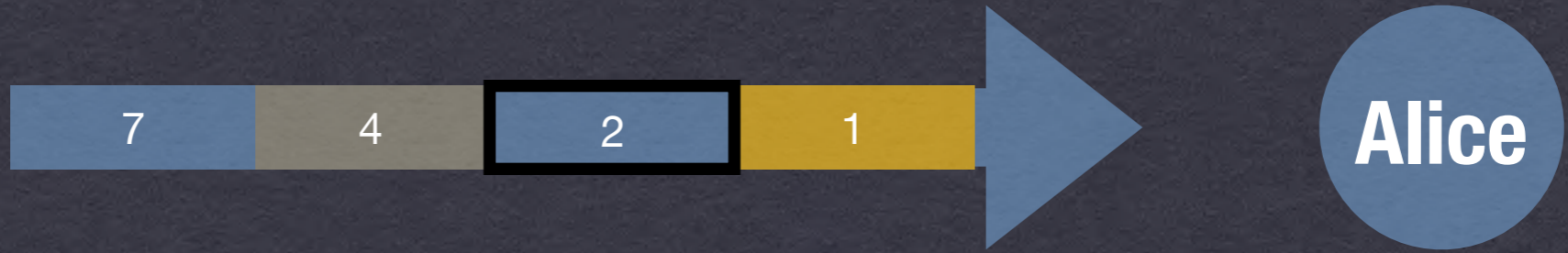


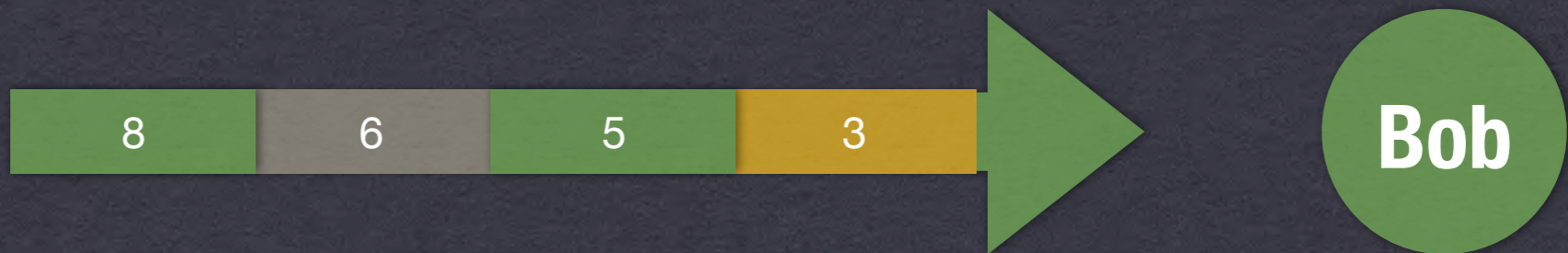
WHAT IS AN ENTITY?

**ENTITIES ARE BOUNDARIES OF
ATOMICITY**











DENORMALIZE..

ALL THE THINGS!



TWO LAYER ARCHITECTURE

scale-agnostic

API

scale-aware

scale-agnostic

API

scale-aware

scale-agnostic

API

scale-aware

A diagram illustrating the relationship between different components. A central green box labeled 'API' is positioned between two orange boxes labeled 'scale-agnostic' (top) and 'scale-aware' (bottom). All three boxes are contained within a larger light blue rounded rectangle. The 'API' box has a thick black border, while the others have rounded corners.

scale-agnostic

API

scale-aware

scale-agnostic

API

scale-aware

scale-agnostic

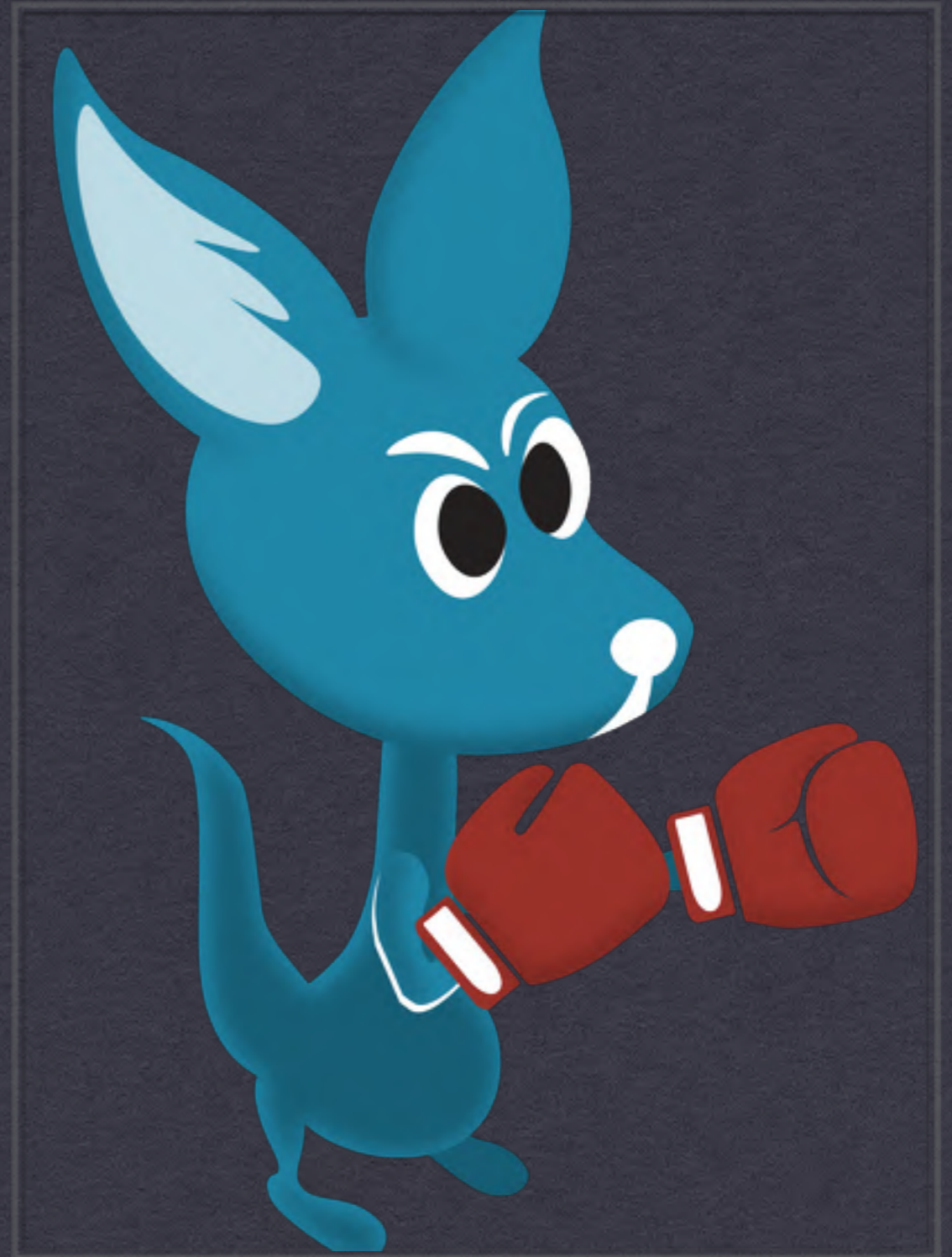
API

scale-aware

**TO SCALE INFINITELY,
YOUR BUSINESS LOGIC HAS TO BE
INDEPENDENT OF SCALE**

WALLAROO

SCALE INDEPENDENT
COMPUTING FOR PYTHON



AND IT'S NOT A DATABASE

ENTITIES

BUT WE CALL THEM...

“STATE OBJECTS”



TWO LAYER ARCHITECTURE

BUT WE CALL IT...

“SCALE INDEPENDENCE”



user supplied logic

The diagram consists of a large light blue rounded rectangle containing three smaller rounded rectangles stacked vertically. The top rectangle is orange and contains the text 'user supplied logic'. The middle rectangle is green and contains the text 'Wallaroo API'. The bottom rectangle is orange and contains the text 'Wallaroo runtime'.

Wallaroo API

Wallaroo runtime

user supplied logic

The diagram consists of a large light blue rounded rectangle containing three smaller rounded rectangles stacked vertically. The top rectangle is orange with a thick black border and contains the text 'user supplied logic'. The middle rectangle is green and contains the text 'Wallaroo API'. The bottom rectangle is orange and contains the text 'Wallaroo runtime'.

Wallaroo API

Wallaroo runtime

user supplied logic

Wallaroo API

Wallaroo runtime

user supplied logic

Wallaroo API

Wallaroo runtime

WHAT'S HARD?

ALL OF IT? YOU'RE BUILDING
A DISTRIBUTED SYSTEMS

FRAMEWORK



CAP THEOREM

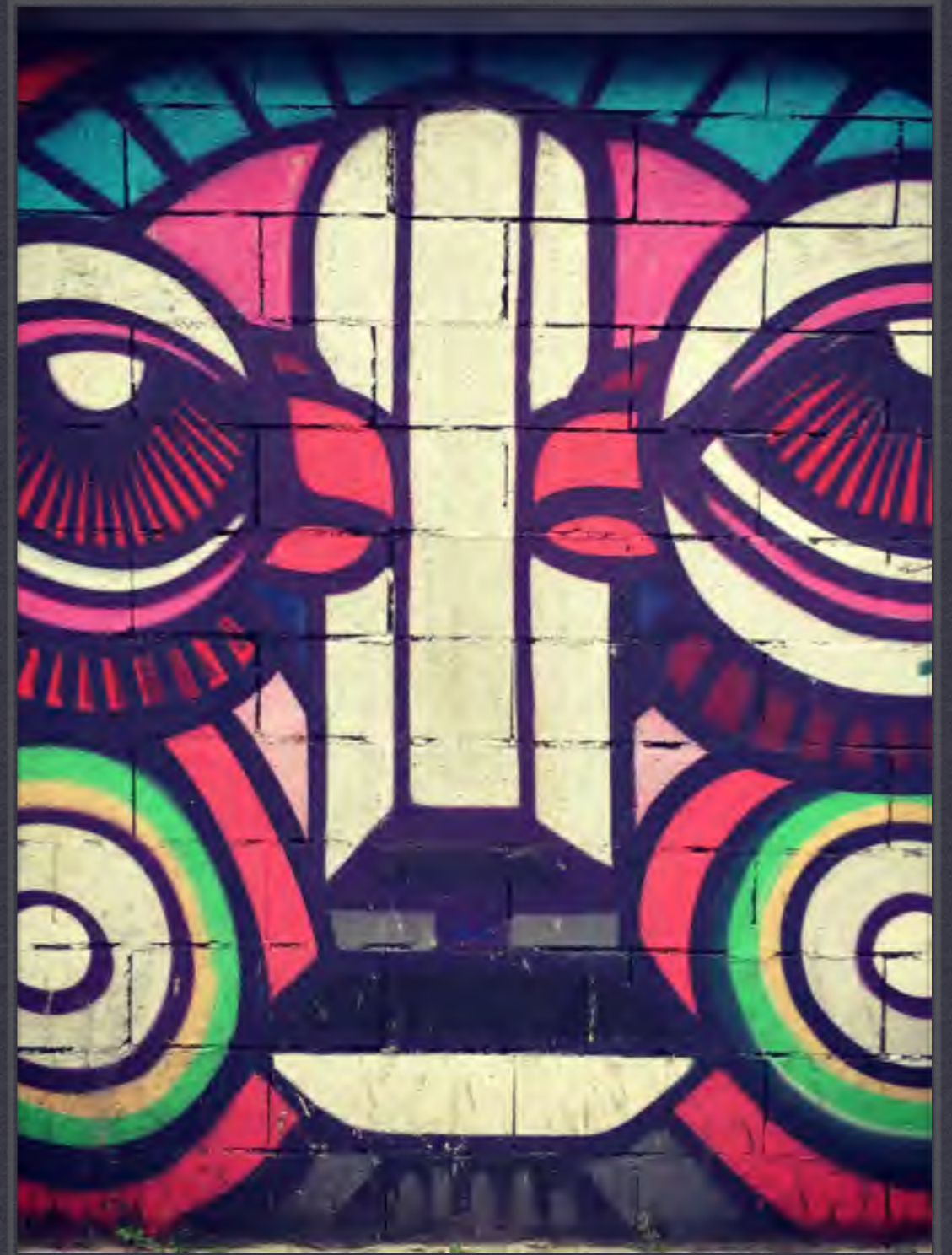
CONSISTENCY VS
AVAILABILITY...

YOU CAN'T ESCAPE IT.



MESSAGE DELIVERY

AT-MOST-ONCE? AT-LEAST-
ONCE? EFFECTIVELY-ONCE?
EXACTLY-ONCE?



MESSAGE ORDERING

WILL YOU MAINTAIN THE
ORDERING AS YOU RECEIVED
IT?



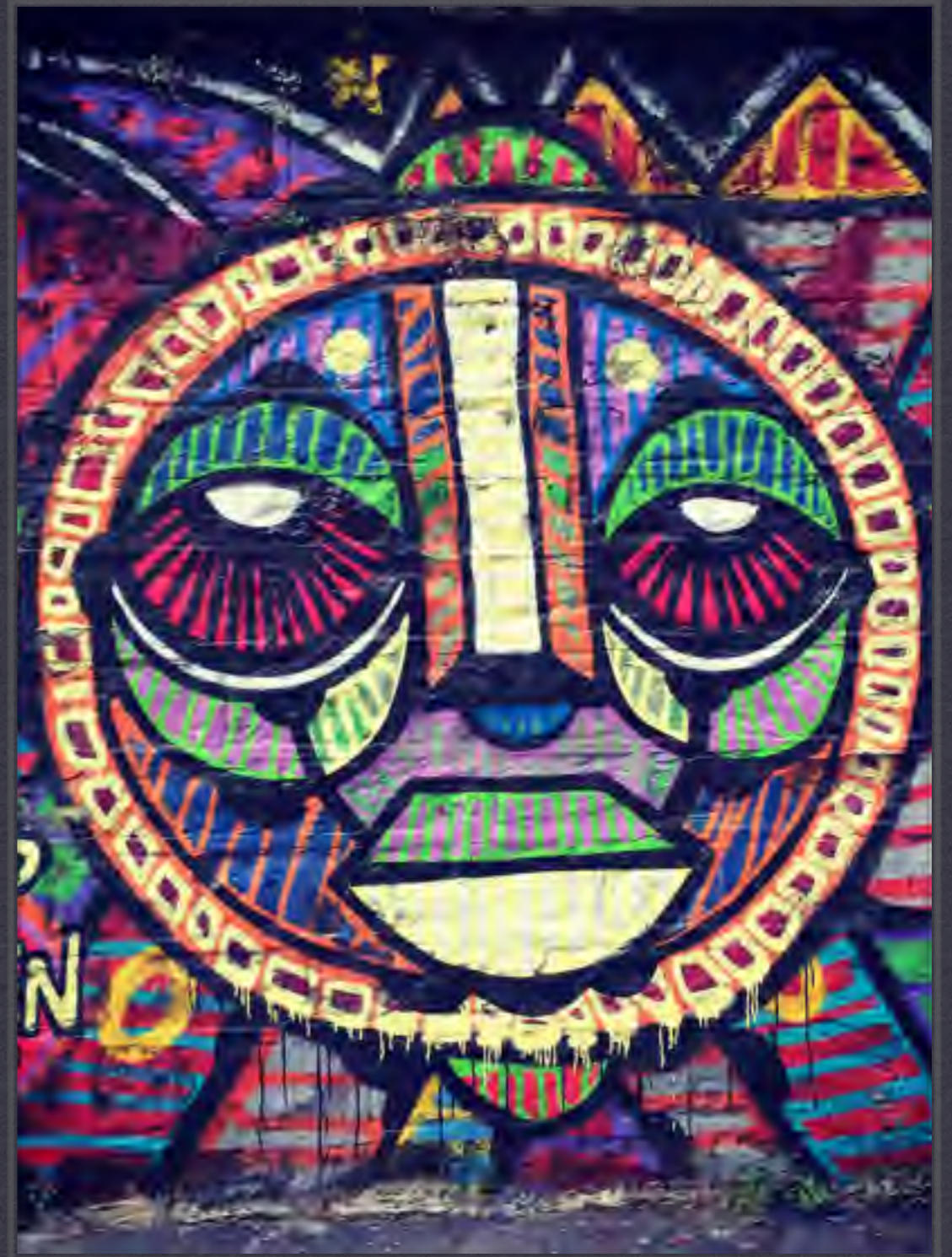
LOCAL KNOWLEDGE

YOU HAVE TO WORK HARD
TO AVOID COORDINATION.



PROGRAMMING MODEL

YOU CAN GET WITH THIS, OR
YOU CAN GET WITH THAT.



PERFORMANCE

IT'S A WORD IN THE
DICTIONARY



NETWORK OVERHEAD

YOU AREN'T IN LOCAL
MEMORY ANYMORE



DATA SERIALIZATION

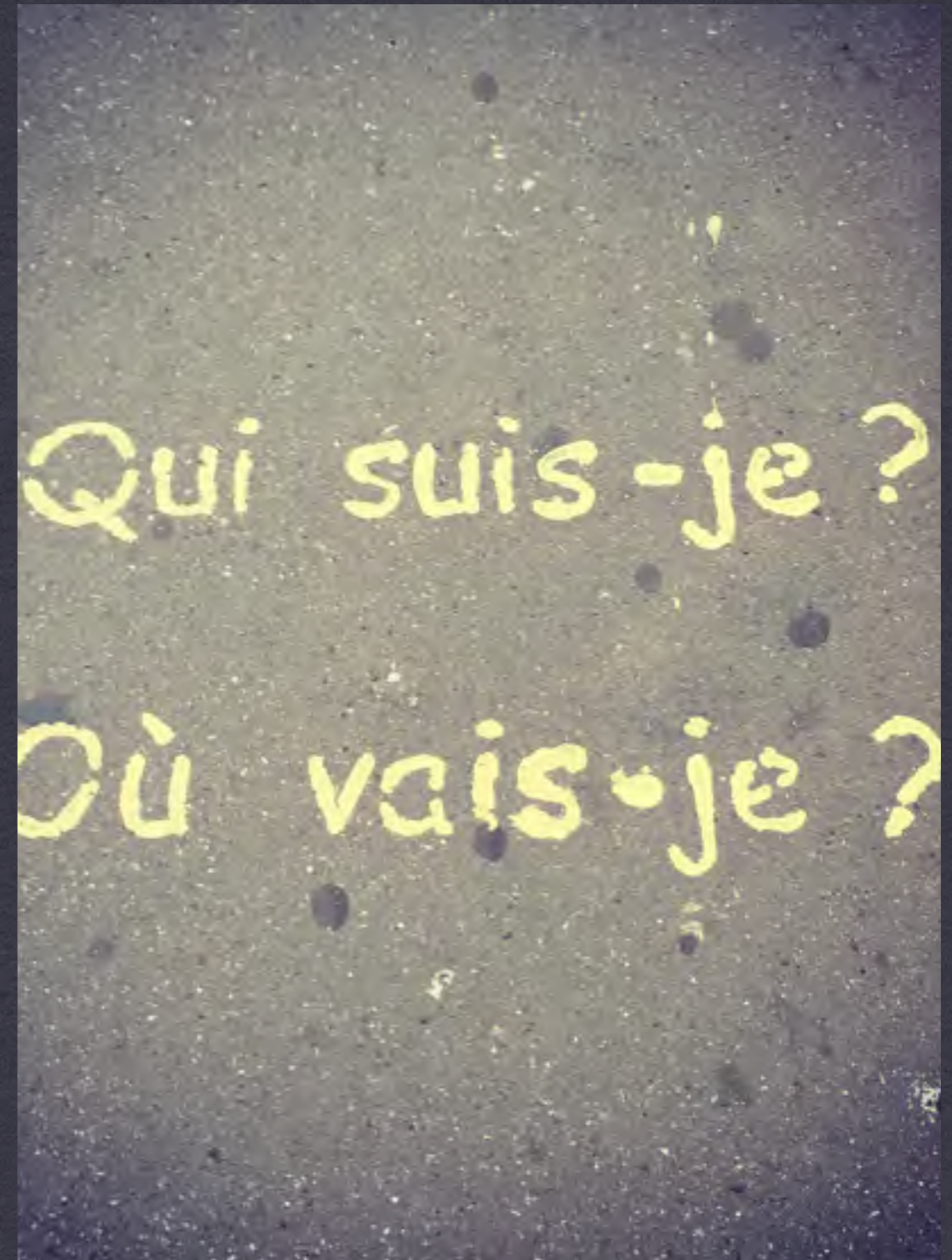
LE SIGH...



VERIFICATION

LET'S NOT GO THERE...

THAT'S AN ENTIRE LECTURE
SERIES.



**BTW....
YOUR
MULTI-CORE
COMPUTER
IS ALSO A DISTRIBUTED
SYSTEM.**

**BUT THAT'S A STORY FOR
ANOTHER DAY.**



LEARN MORE

[GITHUB.COM/SEANTALLEN/
PAT-HELLAND-AND-ME](https://github.com/seantallen/pat-helland-and-me)

