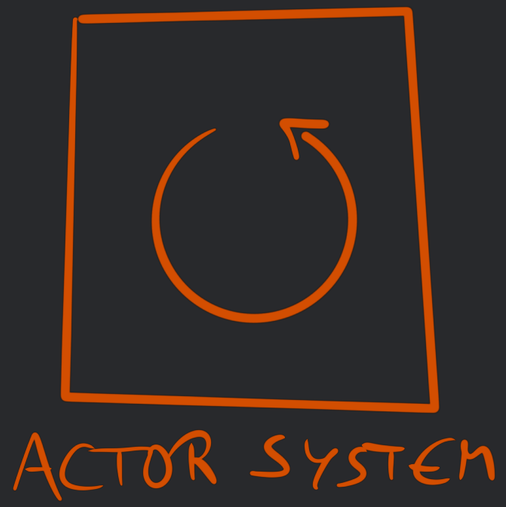
FORMALLY VERIFYING COMPLEX SYSTEMS USING TLA^+

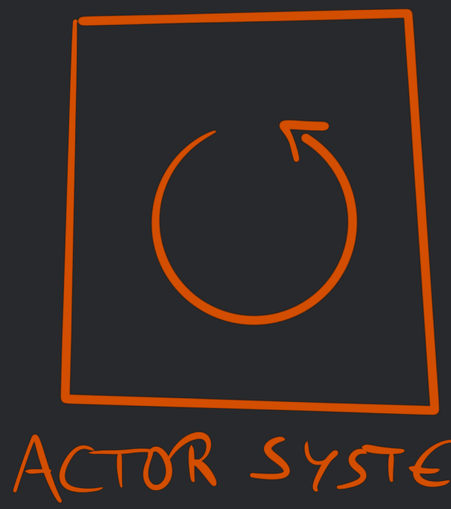
WHOAMI

- RUBEN BERENGUEL (@BERENGUEL)
- PHD IN MATHEMATICS
- (BIG) DATA CONSULTANT
- SENIOR BIG DATA ENGINEER USING PYTHON, GO AND SCALA

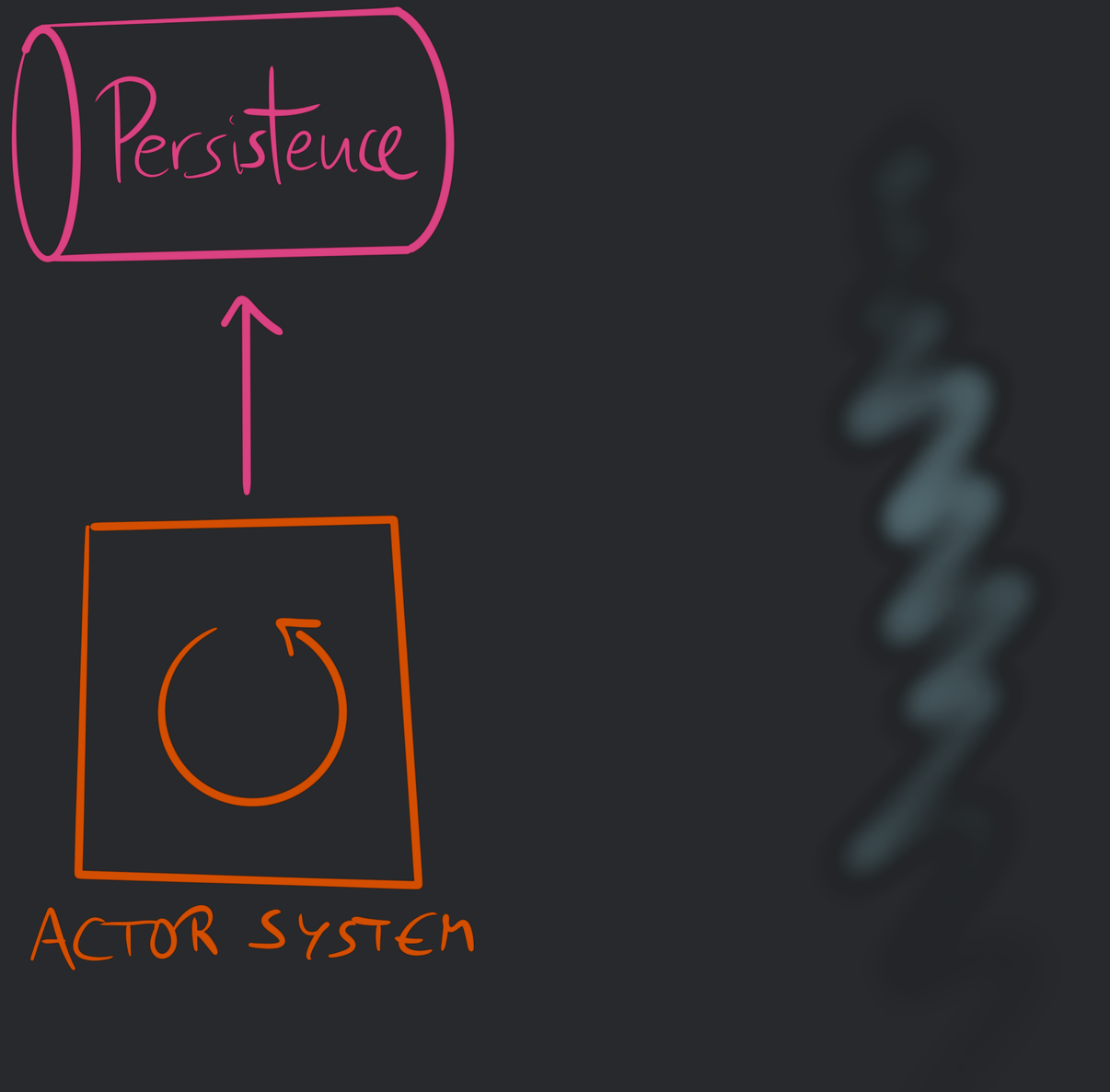
FORMAL METHODS

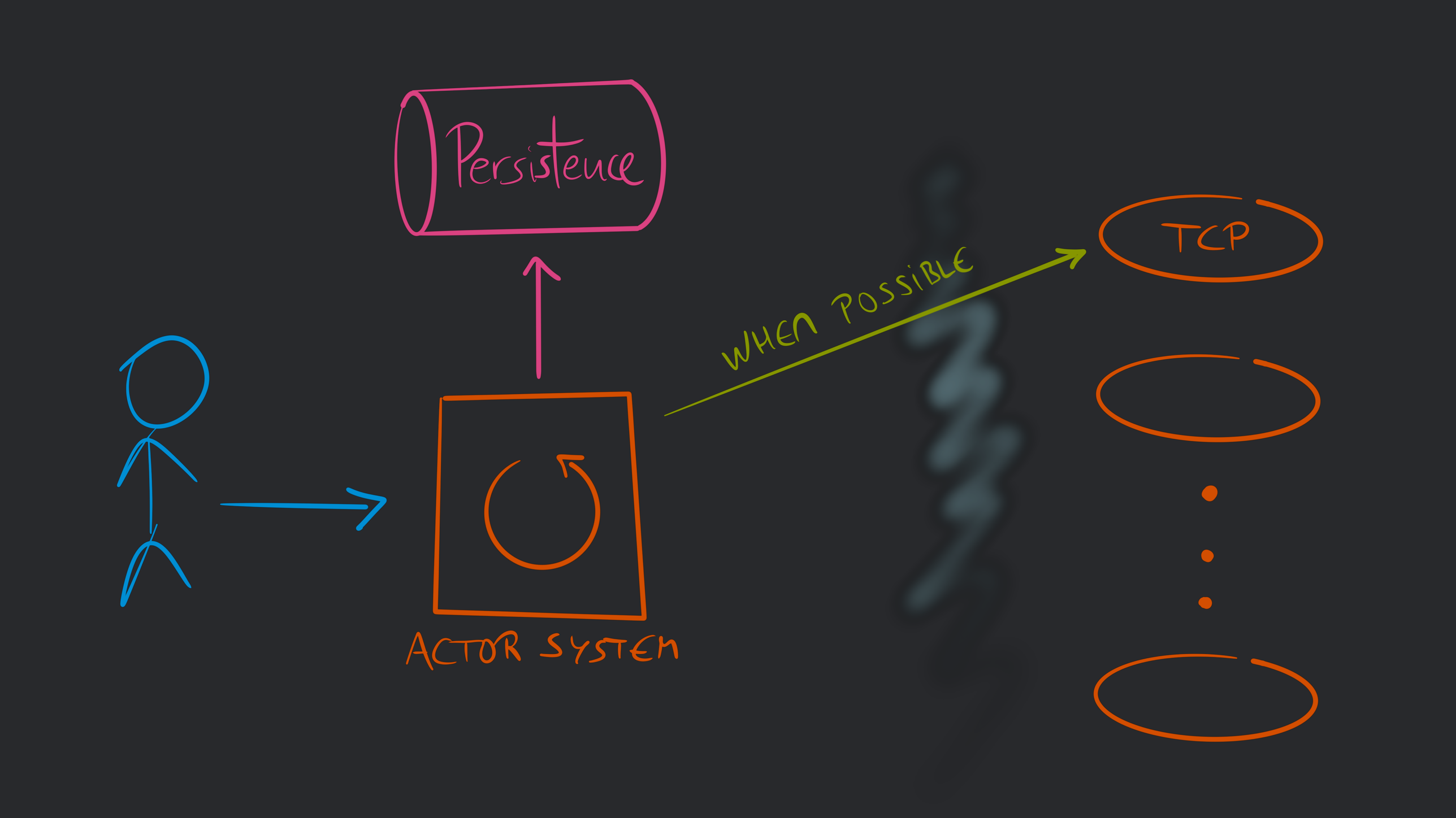
WHY?

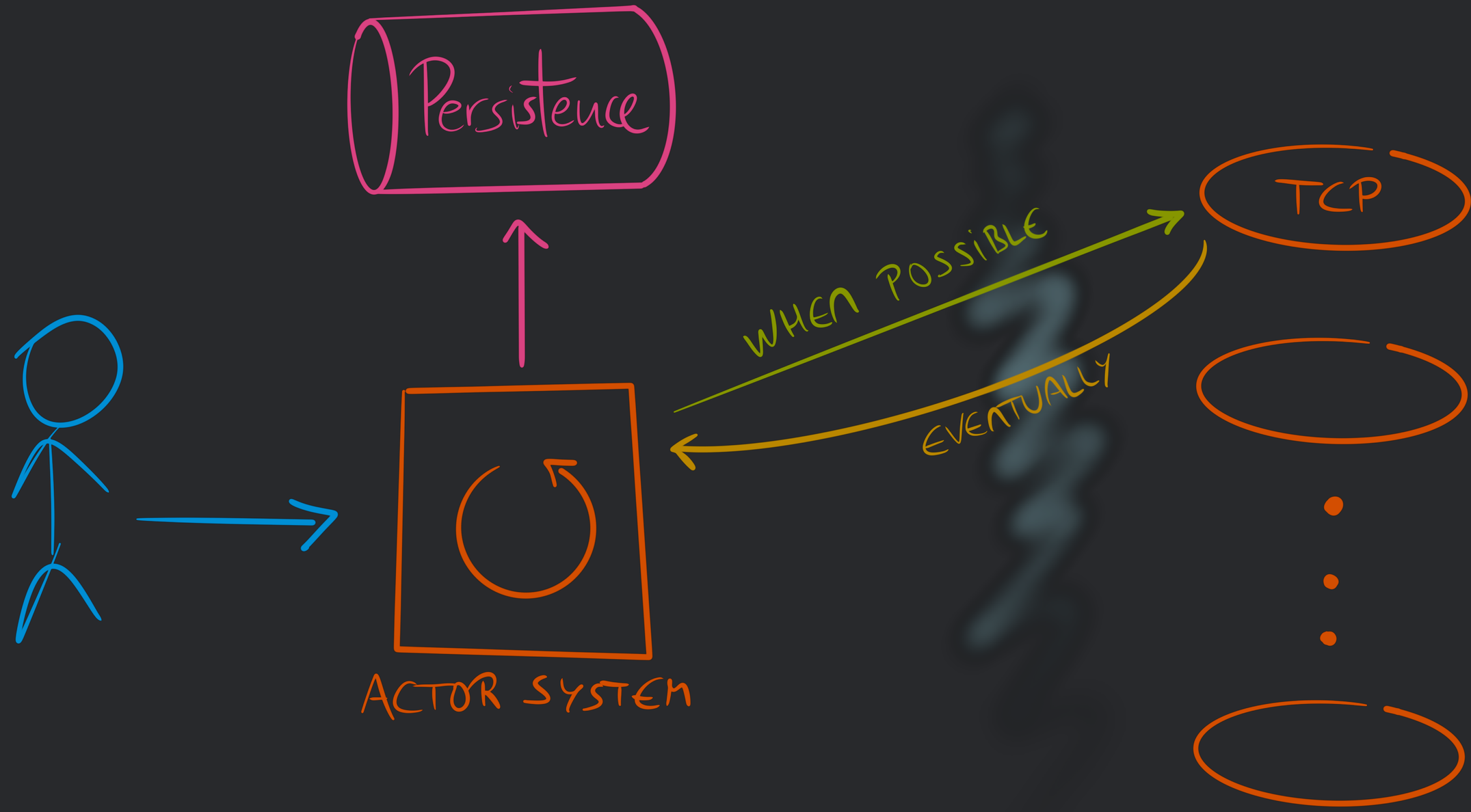

ACTOR SYSTEM


ACTOR SYSTEM


TCP

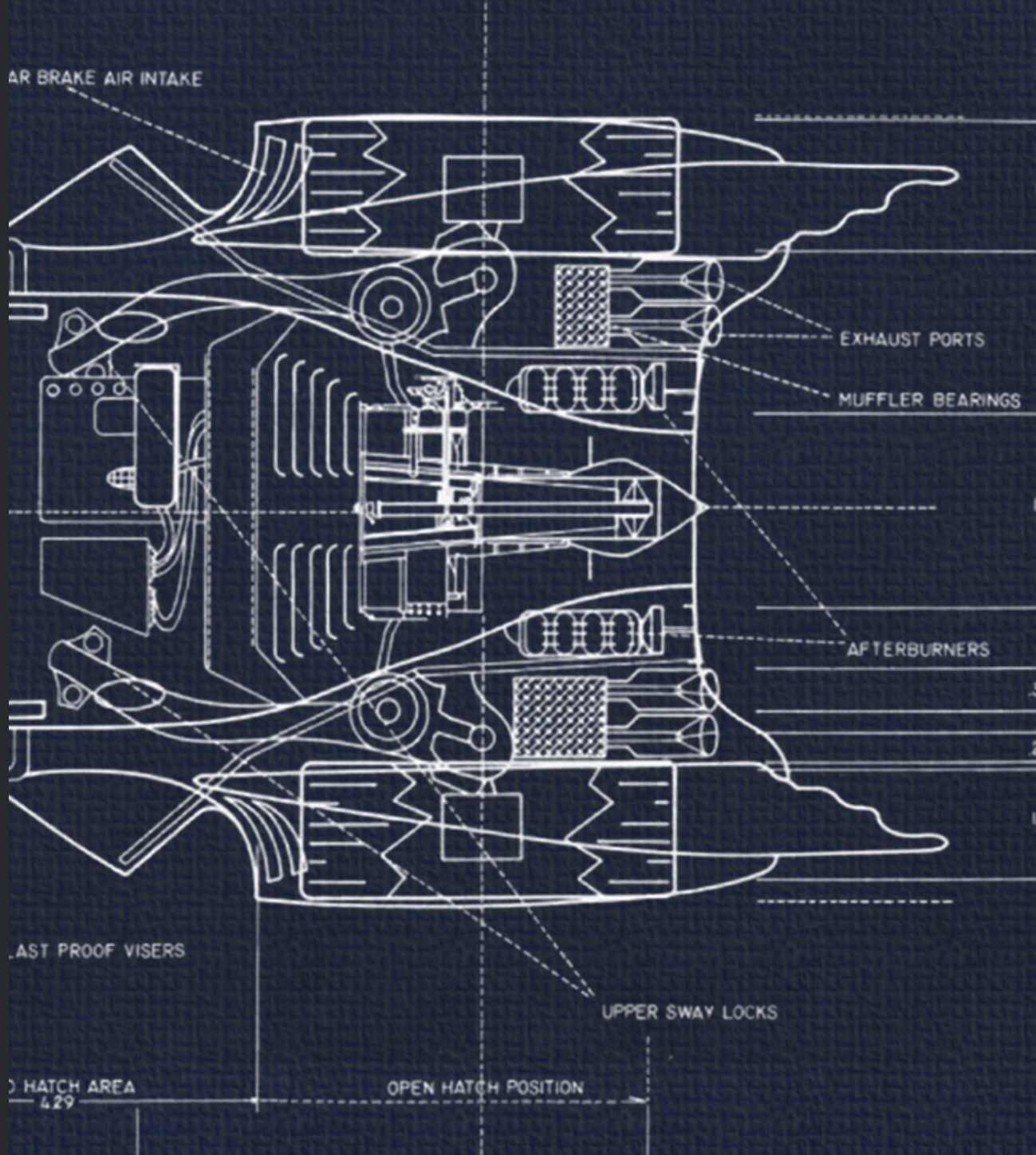

Persistence
ACTOR SYSTEM
TCP


Persistence
WHEN POSSIBLE
TCP
ACTOR SYSTEM


Persistence
TCP
WHEN POSSIBLE
EVENTUALLY
ACTOR SYSTEM

WHAT ARE FORMAL METHODS?

- TECHNIQUE FOR SPECIFYING THE BEHAVIOUR OF HARDWARE OR SOFTWARE
- BASED ON SOME MATHEMATICAL FORMALISM OR THEORY



EXAMPLES

- LAMBDA CALCULUS
- (POSSIBLY DEPENDENT) TYPE SYSTEMS
- TLA (TEMPORAL LOGIC OF ACTIONS)

WHAT DOES FORMAL VERIFICATION REQUIRE?

- WRITE A SPECIFICATION OF WHAT THE SYSTEM IS SUPPOSED TO DO
  - COMPUTATIONALLY VERIFY THE DESIRED PROPERTIES
  - OPTIONALLY/ADDITIONALLY: PROVE THE DESIRED PROPERTIES OF THE SYSTEM

WHAT IS TLA^+?

FORMAL SPECIFICATION LANGUAGE DEVELOPED ON TOP OF TLA BY LESLIE LAMPORT[1]

MODELS WRITTEN IN TLA^+ CAN BE CHECKED USING TLC

[1] THE LA FROM $\LaTeX$ AND THE, WELL, AUTHOR OF THE PAXOS) CONSENSUS ALGORITHM, AMONG MANY OTHER THINGS

A PSEUDOCODE-LIKE LANGUAGE. PLUSCAL TRANSPILES INTO TLA^+. AND CAN BE USED FOR SEQUENTIAL ALGORITHMS

COMES IN TWO FLAVOURS. P[2] AND C[3]

[2] P OF PASCAL? `begin while do stuff end while;`

[3] C AS IN C. {CURLY CURLY}

AN ADDITIONAL PROOF SYSTEM TLAPS[4] CAN BE USED TO CHECK PROOFS

THIS IS HOW BIZANTYNE PAXOS[5] WAS PROVEN, AND HAS BEEN USED IN SEVERAL INDUSTRY PROJECTS

[4] CAN BE FOUND HERE, IT'S SEPARATE FROM THE TLA+ TOOLBOX

[5] MECHANICALLY CHECKED SAFETY PROOF OF A BYZANTINE PAXOS ALGORITHM

SYSTEM	COMPONENTS	LINES OF CODE	RESULT
S3	FAULT TOLERANT NETWORK ALGO	804 (PLUSCAL)	2 BUGS, FURTHER BUGS IN LATER OPTIMISATIONS
S3	**BACKGROUND REDISTRIBUTION**	**645 (PLUSCAL)**	**1 BUG, 1 BUG IN THE FIX**
DYNAMODB	REPLICATION & MEMBERSHIP	939 (PLUSCAL)	3 BUGS (35 STEP TRACE)
EBS	**VOLUME MANAGEMENT**	**223 (PLUSCAL)**	**BETTER CONFIDENCE**
INTERNAL LOCK MANAGER	REPLICATION	318 TLA+	1 BUG

HOW DOES TLA^+ LOOK LIKE?

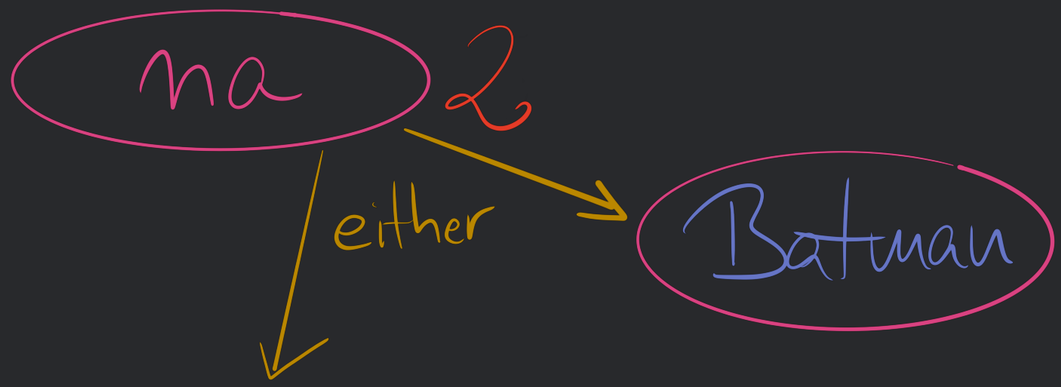
```
-- MODULE ThemeSong --

EXTENDS Integers

VARIABLES s, count

vars == <<s, count>>

Sinit == s = "na" /\ count = 2

Na == s' = "na" /\ count' = count - 1
ToB == s' = "Batman" /\ count' = 0

Snext == IF count > 0 THEN /\ ( Na \/ ToB)
                      ELSE /\ ToB

Spec == Sinit /\ [][Snext]_vars /\ WF_vars(ToB)

Batman == s = "Batman" /\ count = 0

Liveness == <>Batman
```

```
-- MODULE ThemeSong --

EXTENDS Integers

VARIABLES s, count

vars == <<s, count>>

Sinit == s = "na" /\ count = 2

Na == s' = "na" /\ count' = count - 1
ToB == s' = "Batman" /\ count' = 0

Snext == IF count > 0 THEN /\ ( Na \/ ToB)
                      ELSE /\ ToB

Spec == Sinit /\ [][Snext]_vars /\ WF_vars(ToB)

Batman == s = "Batman" /\ count = 0

Liveness == <>Batman
```

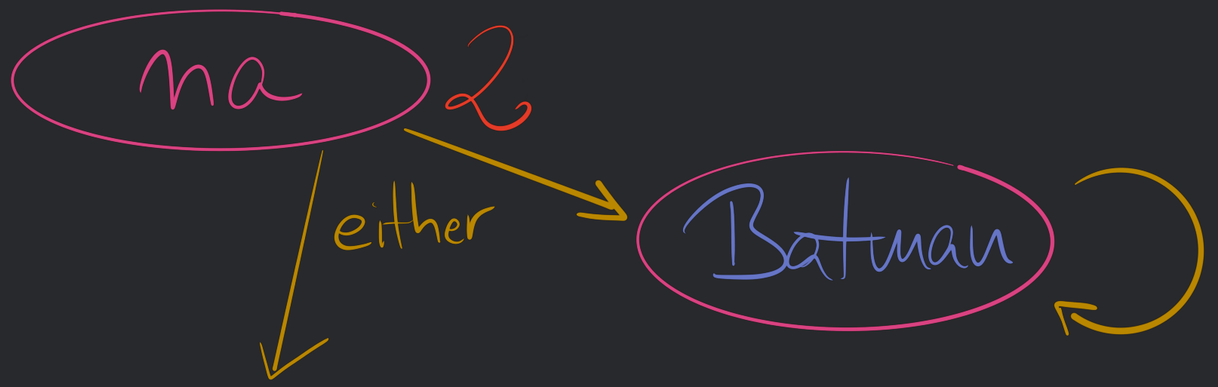
```
-- MODULE ThemeSong --

EXTENDS Integers

VARIABLES s, count

vars == <<s, count>>

Sinit == s = "na" /\ count = 2

Na == s' = "na" /\ count' = count - 1
ToB == s' = "Batman" /\ count' = 0

Snext == IF count > 0 THEN /\ ( Na \/ ToB)
                      ELSE /\ ToB

Spec == Sinit /\ [][Snext]_vars /\ WF_vars(ToB)

Batman == s = "Batman" /\ count = 0

Liveness == <>Batman
```


```
-- MODULE ThemeSong --

EXTENDS Integers

VARIABLES s, count

vars == <<s, count>>

Sinit == s = "na" /\ count = 2

Na == s' = "na" /\ count' = count - 1
ToB == s' = "Batman" /\ count' = 0

Snext == IF count > 0 THEN /\ ( Na \/ ToB)
                      ELSE /\ ToB

Spec == Sinit /\ [][Snext]_vars /\ WF_vars(ToB)

Batman == s = "Batman" /\ count = 0

Liveness == <>Batman
```

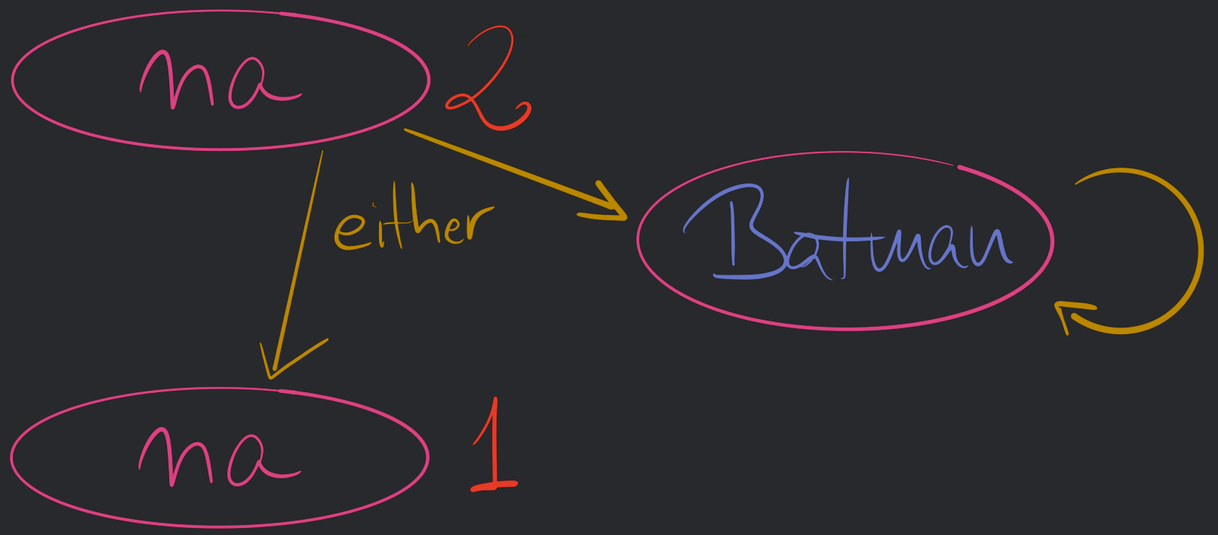
```
-- MODULE ThemeSong --

EXTENDS Integers

VARIABLES s, count

vars == <<s, count>>

Sinit == s = "na" /\ count = 2

Na == s' = "na" /\ count' = count - 1
ToB == s' = "Batman" /\ count' = 0

Snext == IF count > 0 THEN /\ ( Na \/ ToB)
                      ELSE /\ ToB

Spec == Sinit /\ [][Snext]_vars /\ WF_vars(ToB)

Batman == s = "Batman" /\ count = 0

Liveness == <>Batman
```



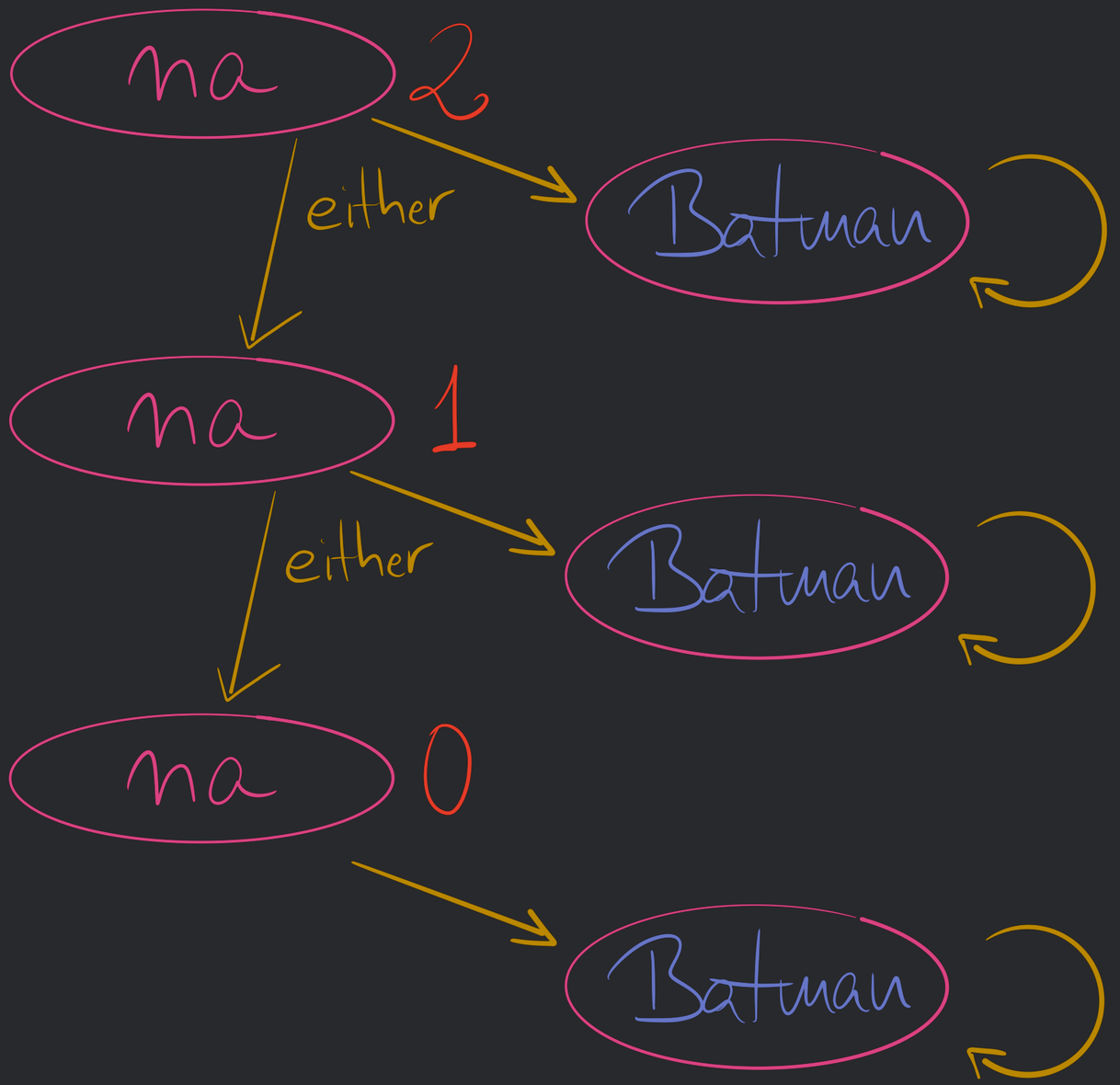
```
-- MODULE ThemeSong --

EXTENDS Integers

VARIABLES s, count

vars == <<s, count>>

Sinit == s = "na" /\ count = 2

Na == s' = "na" /\ count' = count - 1
ToB == s' = "Batman" /\ count' = 0

Snext == IF count > 0 THEN /\ ( Na \/ ToB)
                      ELSE /\ ToB

Spec == Sinit /\ [][Snext]_vars /\ WF_vars(ToB)

Batman == s = "Batman" /\ count = 0

Liveness == <>Batman
```



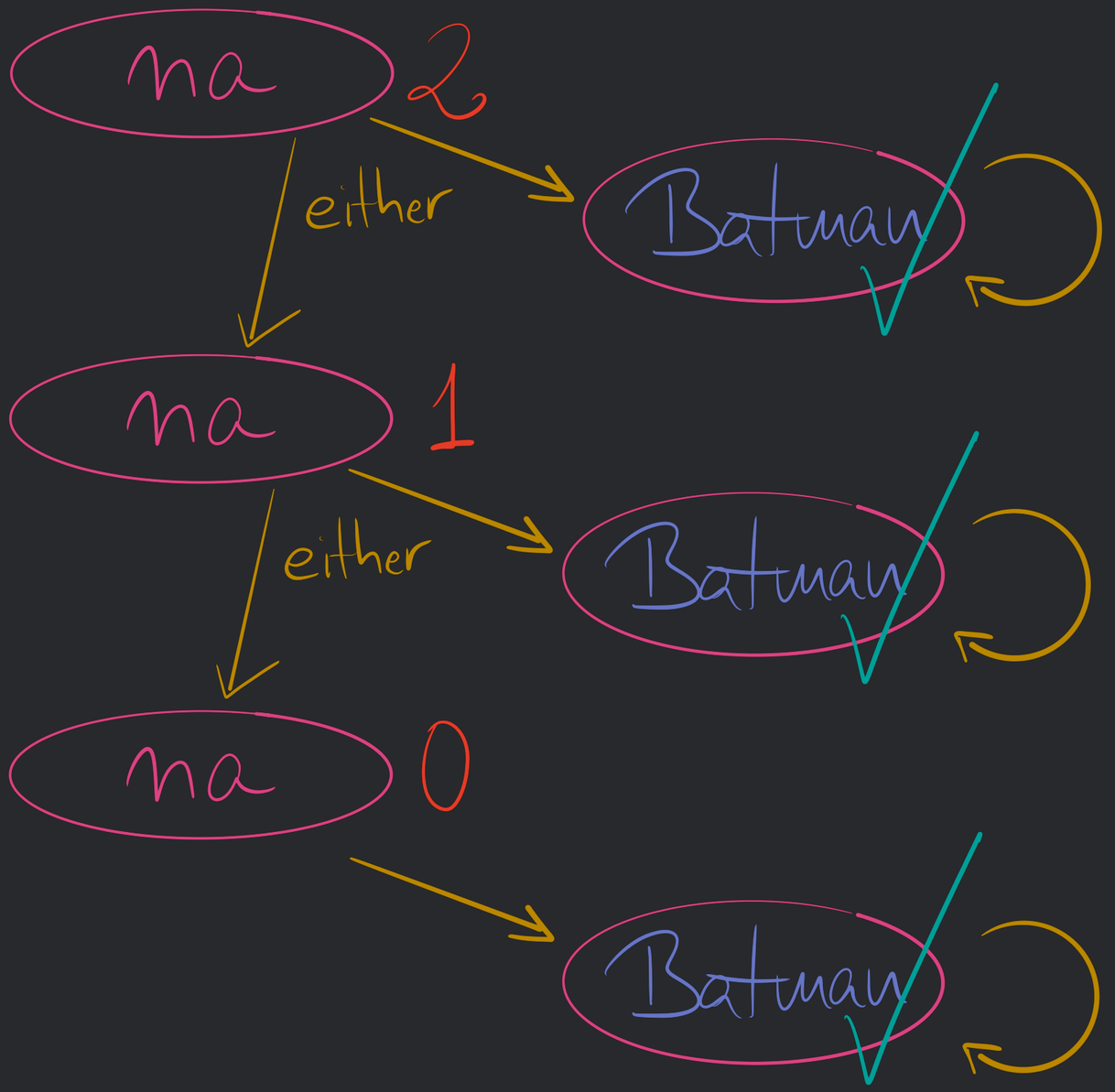
```
-- MODULE ThemeSong --

EXTENDS Integers

VARIABLES s, count

vars == <<s, count>>

Sinit == s = "na" /\ count = 2

Na == s' = "na" /\ count' = count - 1
ToB == s' = "Batman" /\ count' = 0

Snext == IF count > 0 THEN /\ ( Na \/ ToB)
                      ELSE /\ ToB

Spec == Sinit /\ [][Snext]_vars /\ WF_vars(ToB)

Batman == s = "Batman" /\ count = 0

Liveness == <>Batman
```



```
-- MODULE ThemeSong --

EXTENDS Integers

VARIABLES s, count

vars == <<s, count>>

Sinit == s = "na" /\ count = 2

Na == s' = "na" /\ count' = count - 1
ToB == s' = "Batman" /\ count' = 0

Snext == IF count > 0 THEN /\ ( Na \/ ToB)
                      ELSE /\ ToB

Spec == Sinit /\ [][Snext]_vars /\ WF_vars(ToB)

Batman == s = "Batman" /\ count = 0

Liveness == <>Batman
```



```
-- MODULE ThemeSong --

EXTENDS Integers

VARIABLES s, count

vars == <<s, count>>

Sinit == s = "na" /\ count = 2

Na == s' = "na" /\ count' = count - 1
ToB == s' = "Batman" /\ count' = 0

Snext == IF count > 0 THEN /\ ( Na \/ ToB)
                      ELSE /\ ToB

Spec == Sinit /\ [][Snext]_vars /\ WF_vars(ToB)

Batman == s = "Batman" /\ count = 0

Liveness == <>Batman
```



HOW WOULD THE EQUIVALENT PLUSCAL ALGORITHM LOOK LIKE?

```
(*--fair algorithm ThemeSong {
    variables count = 2, s = "na";
      {
        while (count > 0) {
            either {
                s := "Batman";
                count := 0;
            } or {
                s := "na";
                count := count - 1;
            }
        };
        if (count = 0) {
            s := "Batman";
        }
      }
    } -*)
```

```
VARIABLES count, s, pc

vars == << count, s, pc >>

Init == /\ count = 2
        /\ s = "na"
        /\ pc = "ThemeSong"

ThemeSong == /\ pc = "ThemeSong"
             /\ IF count > 0
                   THEN /\ \/ /\ s' = "Batman"
                              /\ count' = 0
                           \/ /\ s' = "na"
                              /\ count' = count - 1
                        /\ pc' = "ThemeSong"
                   ELSE /\ IF count = 0
                              THEN /\ s' = "Batman"
                              ELSE /\ TRUE
                                   /\ s' = s
                        /\ pc' = "Done"
                        /\ count' = count

Next == ThemeSong
          \/ (pc = "Done" /\ UNCHANGED vars)

Spec == /\ Init /\ [][Next]_vars /\ WF_vars(Next)
```

WE CAN "EASILY" MODEL WHOLE SYSTEMS USING PLUSCAL

LIKE...

```
case class Trigger(payload: Payload)

class SimpleActor extends Actor {
  def receive = {
    case Trigger(payload) => ThirdParty().trigger(payload)
    case _                => log.warn("Duh")
  }
}
```

WITH PLUSCAL WE CAN MODEL CONCURRENT (OR NOT) SYSTEMS BY USING `process`

```
variables actorInboxes = [ actor |-> <<>> ];
          triggered = FALSE;

procedure trigger(trigger_content="?") {
    triggerLabel:
      triggered := TRUE;
      return;
}

fair process (actor = "actor")
variables currentMessage = <<"?", "no_content">>;
  kind = "?";
  content = "no_content";
{
  Send:
    actorInboxes["actor"] := Append(actorInboxes["actor"], <<"trigger", "foo">>);
  WaitForMessages:+
    while (TRUE) {
      if (actorInboxes["actor"] /= <<>>) {
         currentMessage := Head(actorInboxes["actor"]);
         content := Head(Tail(currentMessage));
         kind := Head(currentMessage);
         actorInboxes["actor"] := Tail(actorInboxes["actor"]);
        };
        ProcessMessage:
         if (kind = "trigger") {
           call trigger(content);
         }
    }
}
```

```
variables actorInboxes = [ actor |-> <<>> ];
          triggered = FALSE;

procedure trigger(trigger_content="?") {
    triggerLabel:
      triggered := TRUE;
      return;
}
```

```
fair process (actor = "actor")
variables currentMessage = <<"?", "no_content">>;
  kind = "?";
  content = "no_content";
{
  Send:
    actorInboxes["actor"] := Append(actorInboxes["actor"], <<"trigger", "foo">>);
  WaitForMessages:+
    while (TRUE) {
      if (actorInboxes["actor"] /= <<>>) {
         currentMessage := Head(actorInboxes["actor"]);
         content := Head(Tail(currentMessage));
         kind := Head(currentMessage);
         actorInboxes["actor"] := Tail(actorInboxes["actor"]);
        };
        ProcessMessage:
         if (kind = "trigger") {
           call trigger(content);
         }
    }
}
```

A POSSIBLE CHECK ON THIS SYSTEM COULD THEN BE

```
Triggered == triggered = TRUE

Liveness == <>Triggered
```


QUESTIONS?


THANKS!

FURTHER REFERENCES

DOWNLOADING THE TLA TOOLBOX

SPECIFYING SYSTEMS BY LESLIE LAMPORT[SSLL]

PRACTICAL TLA+ BY HILLEL WAYNE[PTHW]

[SSLL] THIS IS THE MAIN REFERENCE FOR TLA+. AND ALTHOUGH IT CAN LOOK A BIT DENSE IT READS VERY WELL.

[PTHW] THIS IS A LIGHTHER FIRST REFERENCE, FOCUSED IN PLUSCAL (USING THE P SYNTAX). I STARTED WITH THIS BOOK, AND WITH IT YOU CAN EASILY GET ENOUGH PLUSCAL TO DEFINE YOUR SPECIFICATIONS AND VALIDATE YOUR MODELS

TLA+ VIDEO COURSE BY LESLIE LAMPORT[TVC]

LEARN TLA ONLINE BOOK BY HILLEL WAYNE[LTHW]

[TVC] THIS IS A VIDEO COURSE BY LAMPORT. TOTAL OF AROUND 4 HOURS

[LTHW] AN ONLINE BOOK BY HILLEL WAYNE COVERING PARTS OF PRACTICAL TLA+

EOF