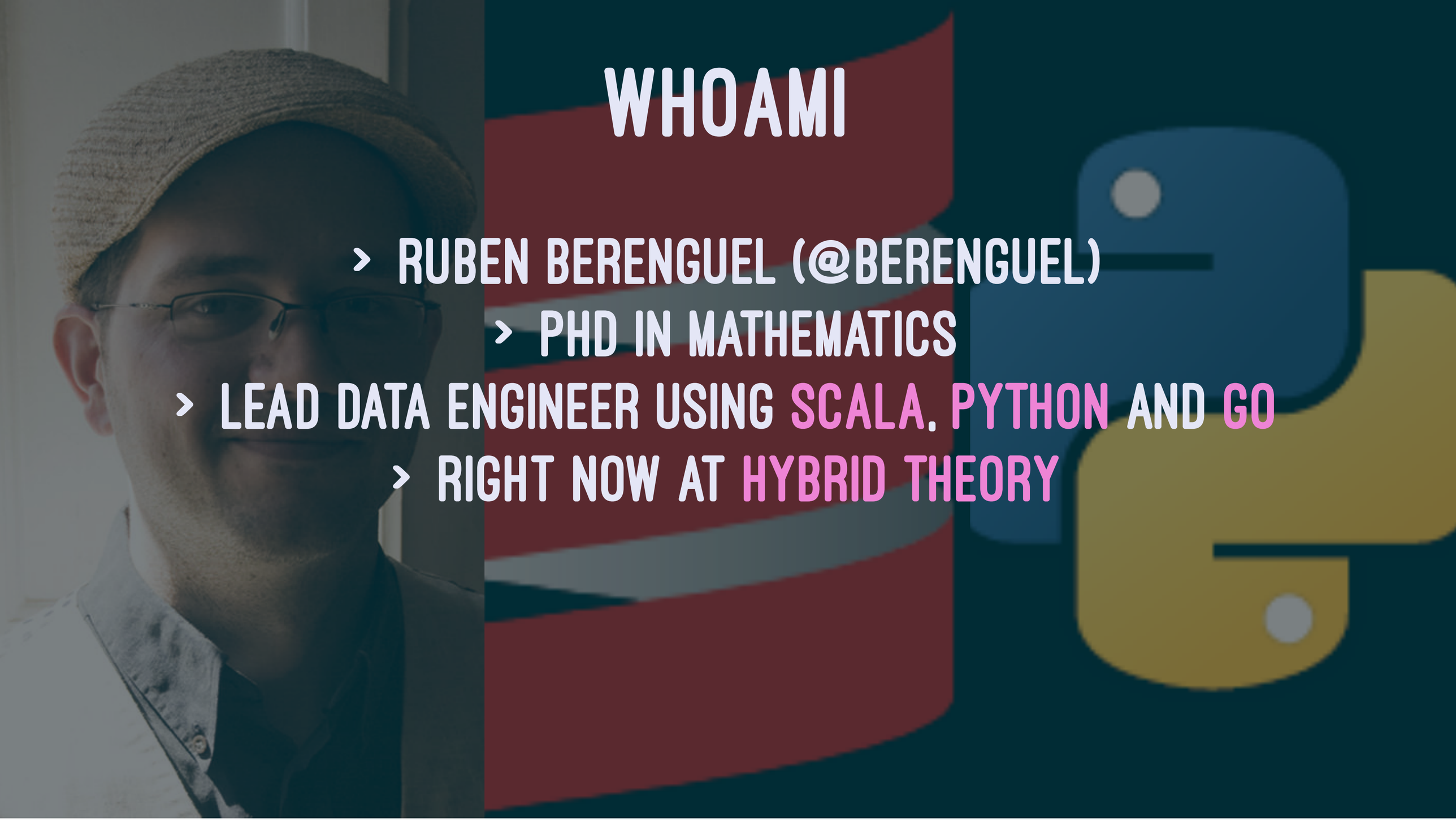
COMMODITISATION AND PROGRAMMING LANGUAGES

RUBEN BERENGUEL. LEAD DATA ENGINEER

Test footnote, hello!



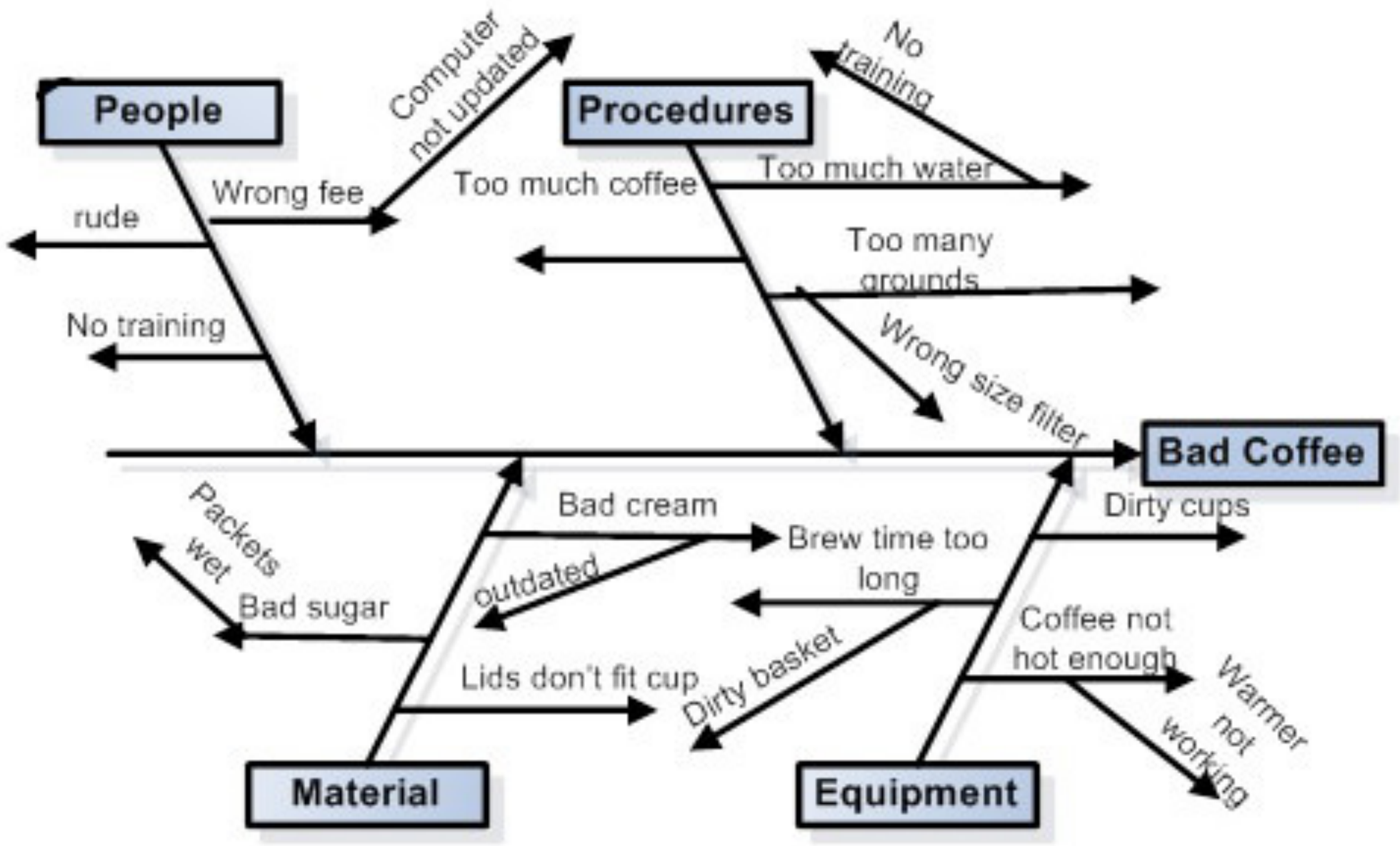
WHOAMI

- RUBEN BERENGUEL (@BERENGUEL)
- PHD IN MATHEMATICS
- LEAD DATA ENGINEER USING SCALA, PYTHON AND GO
- RIGHT NOW AT HYBRID THEORY

COMMODITISATION AND PROGRAMMING LANGUAGES

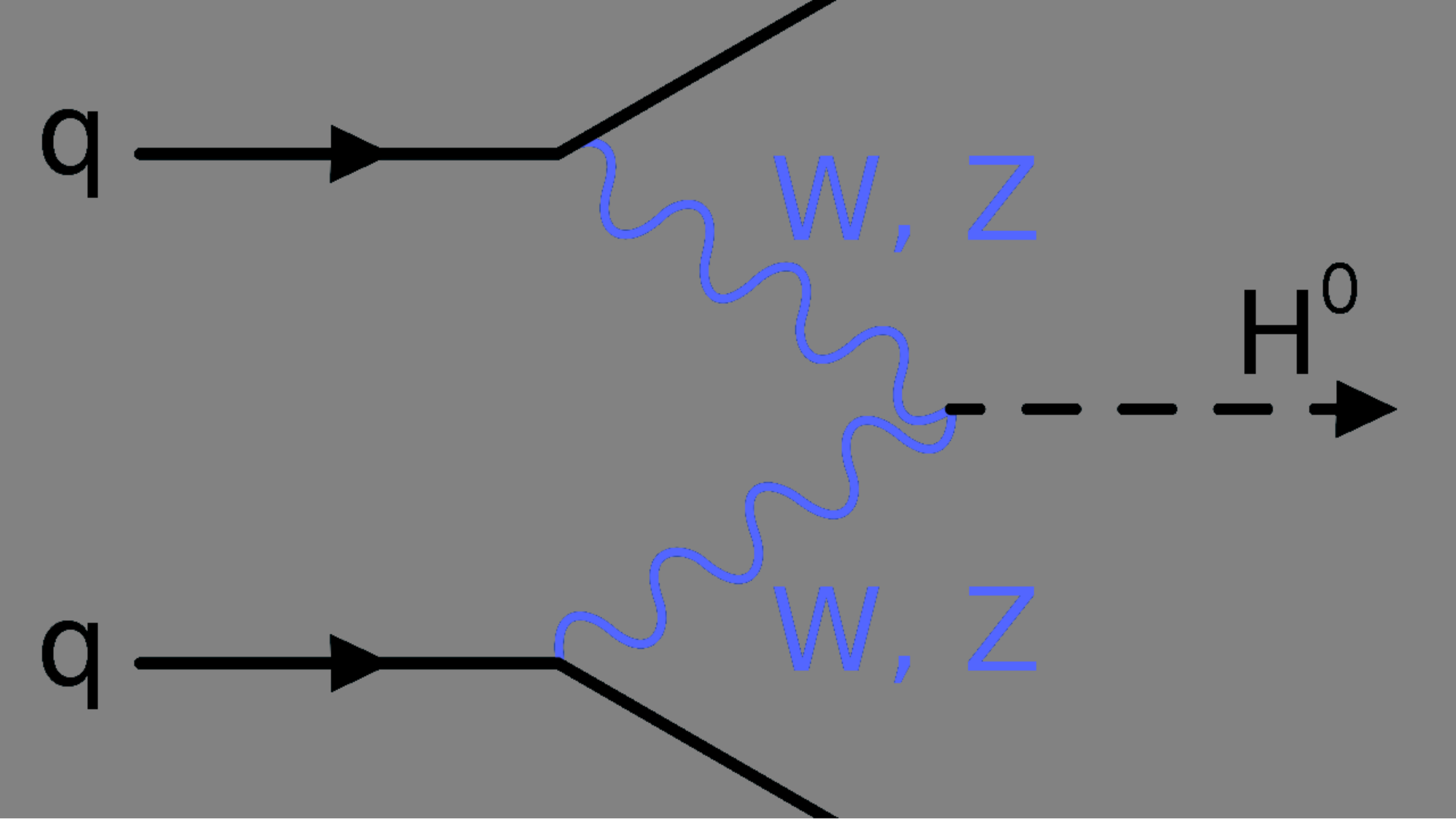
You may be wondering what my approach to strategy can be, as a tech focused person. Last year, I had a mapping talk where I focused on technology landscape. It was a mapping talk with a twist: the map was not a Wardley map in the sense of having the usual x or y axes. Today I want to explore this a bit more together with you, by focusing on a recent question of mine. First, let me get us in the mood for mapping

Marcel Duchamp's _Nude descending a staircase n.2
and Pablo Picasso's Girl with a mandolin:
Cubism changed how painting and observers saw reality, by showing different perspectives in one image, or depicting time and space in different ways



Reasons for bad coffee.

Ishikawa diagrams allowed people to analyse defects and reactions with ease



A Higgs boson maybe appears Feynman diagrams revolutionised theoretical particle physics

Reihen	Gruppo I. — R^1O	Gruppo II. — RO	Gruppo III. — R^2O^3	Gruppo IV. RH^4 RO^2	Gruppo V. RH^3 R^2O^5	Gruppo VI. RH^2 RO^3	Gruppo VII. RH R^2O^7	Gruppo VIII. — RO^4
1	H=1							
2	Li=7	Be=9,4	B=11	C=12	N=14	O=16	F=19	
3	Na=23	Mg=24	Al=27,3	Si=28	P=31	S=32	Cl=35,5	
4	K=39	Ca=40	—=44	Ti=48	V=51	Cr=52	Mn=55	Fe=56, Co=59, Ni=59, Cu=63.
5	(Cu=63)	Zn=65	—=68	—=72	As=75	Se=78	Br=80	
6	Rb=85	Sr=87	?Yt=88	Zr=90	Nb=94	Mo=96	—=100	Ru=104, Rh=104, Pd=106, Ag=108.
7	(Ag=108)	Cd=112	In=113	Sn=118	Sb=122	Te=125	J=127	
8	Cs=133	Ba=137	?Di=138	?Ce=140	—	—	—	— — — —
9	(—)	—	—	—	—	—	—	
10	—	—	?Er=178	?La=180	Ta=182	W=184	—	Os=195, Ir=197, Pt=198, Au=199.
11	(Au=199)	Hg=200	Tl=204	Pb=207	Bi=208	—	—	
12	—	—	—	Th=231	—	U=240	—	— — — —

Mendeleev's original.

The periodic table organised information in a way that highlighted what was unknown

BUT THERE ARE ALSO UNKNOWN UNKNOWNS – THE ONES WE DON'T KNOW WE DON'T KNOW.

– DONALD RUMSFELD

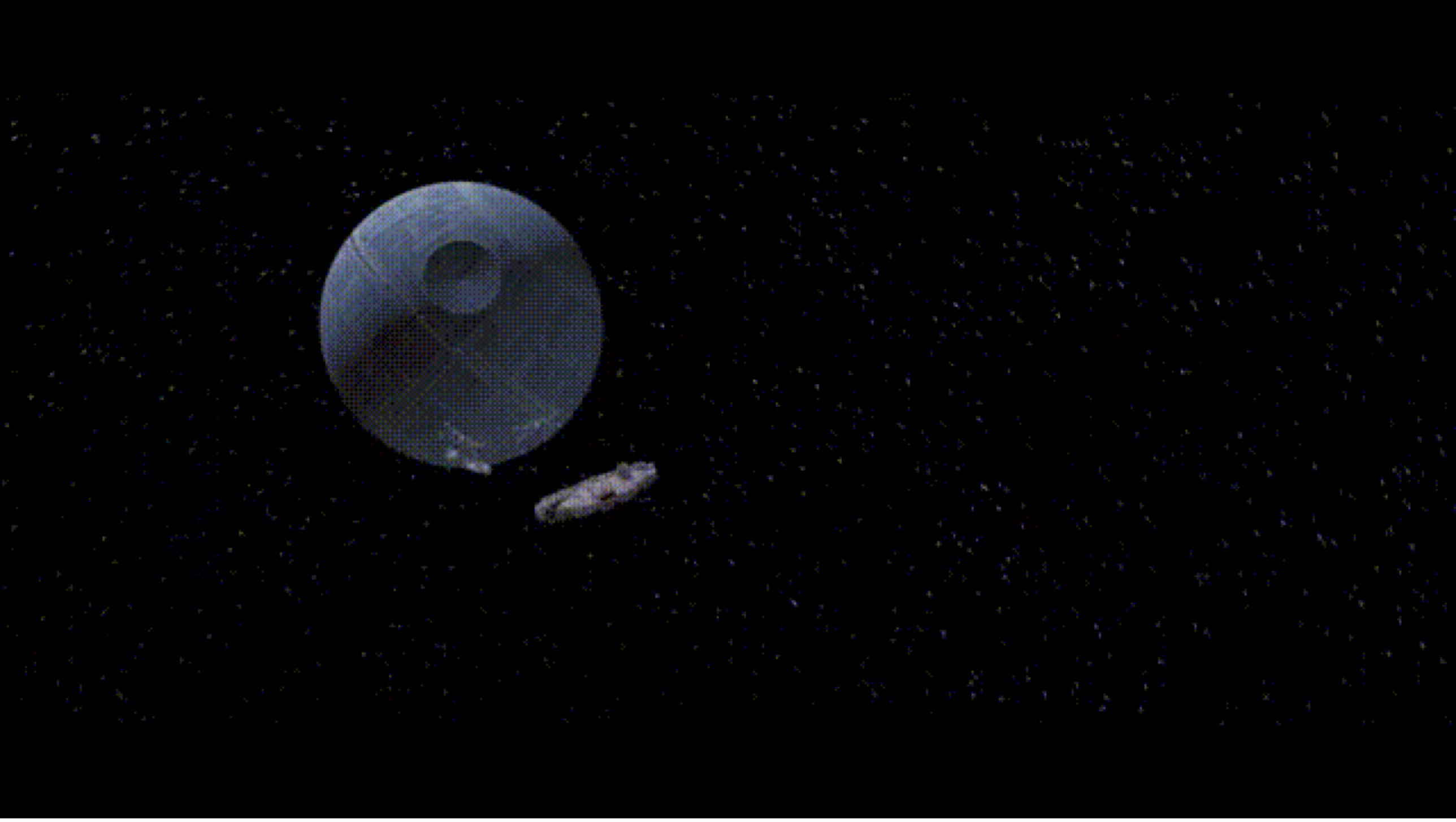
The potential of finding new ways to represent what you know is exactly this: realising what *you don't know*.
Mendeleev's periodic table allowed him to
predict the properties of missing elements, such as gallium and germanium

WARDLEY MAPPING

WHAT HAPPENED BETWEEN...

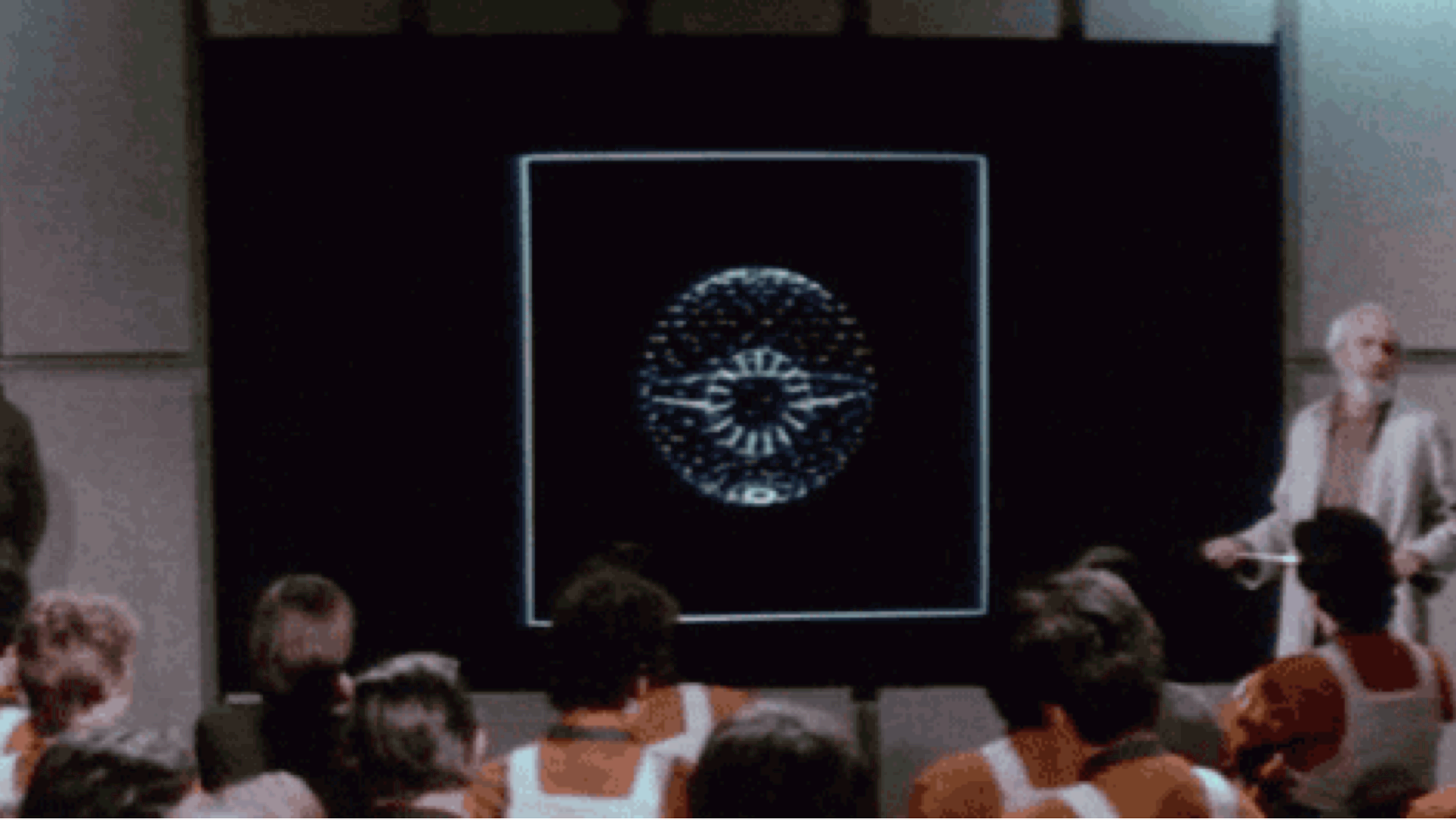
Alderaan didn't shoot first

AND THIS



[1] AND IT'S NOT THE FORCE

IT'S ACTUALLY THIS



HAVING A MAP

&

HAVING A PLAN

NOW THAT WE ARE IN A MAPPING MOOD, WHAT IS THE PROBLEM SPACE **WE WANT TO ANALYSE?**

These are not the Wardley Maps you are thinking of. I think of Wardley Mapping as an overarching tool to explore problem spaces.
Join me to explore this one

WHAT ARE THE LANDSCAPE AND CLIMATIC PATTERNS OF PROGRAMMING LANGUAGES?

My first question wasn't this one. I was pondering what the future relationship between
Rust and Python may be given Dropbox's rewrite of its core sync engine in Rust (from Python),
and other rewrites and projects moving from Python to Rust (at least for the hot paths)

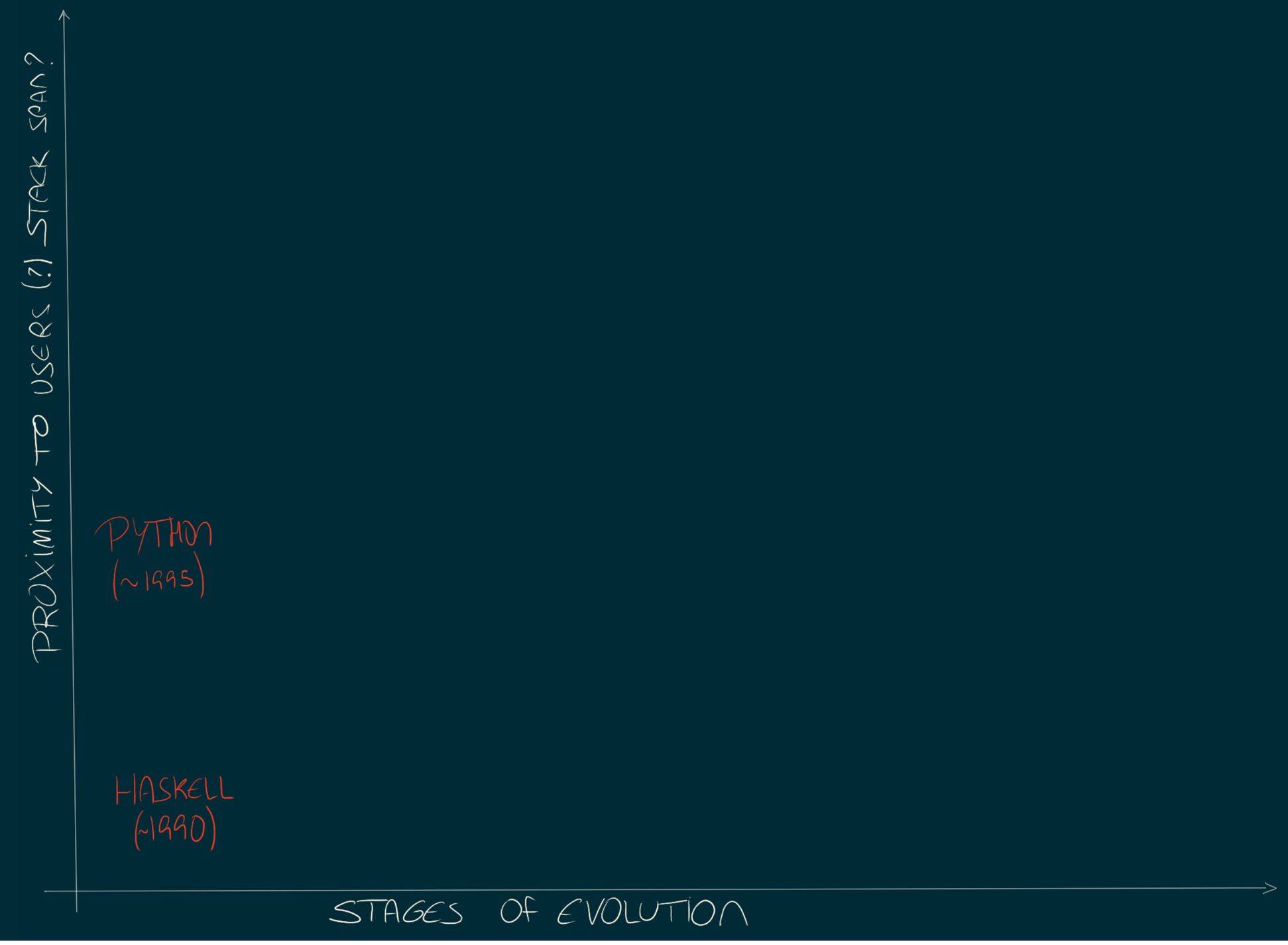
THIS QUESTION IS TOO LARGE

FOCUS ON FIGURING THE AXES FIRST

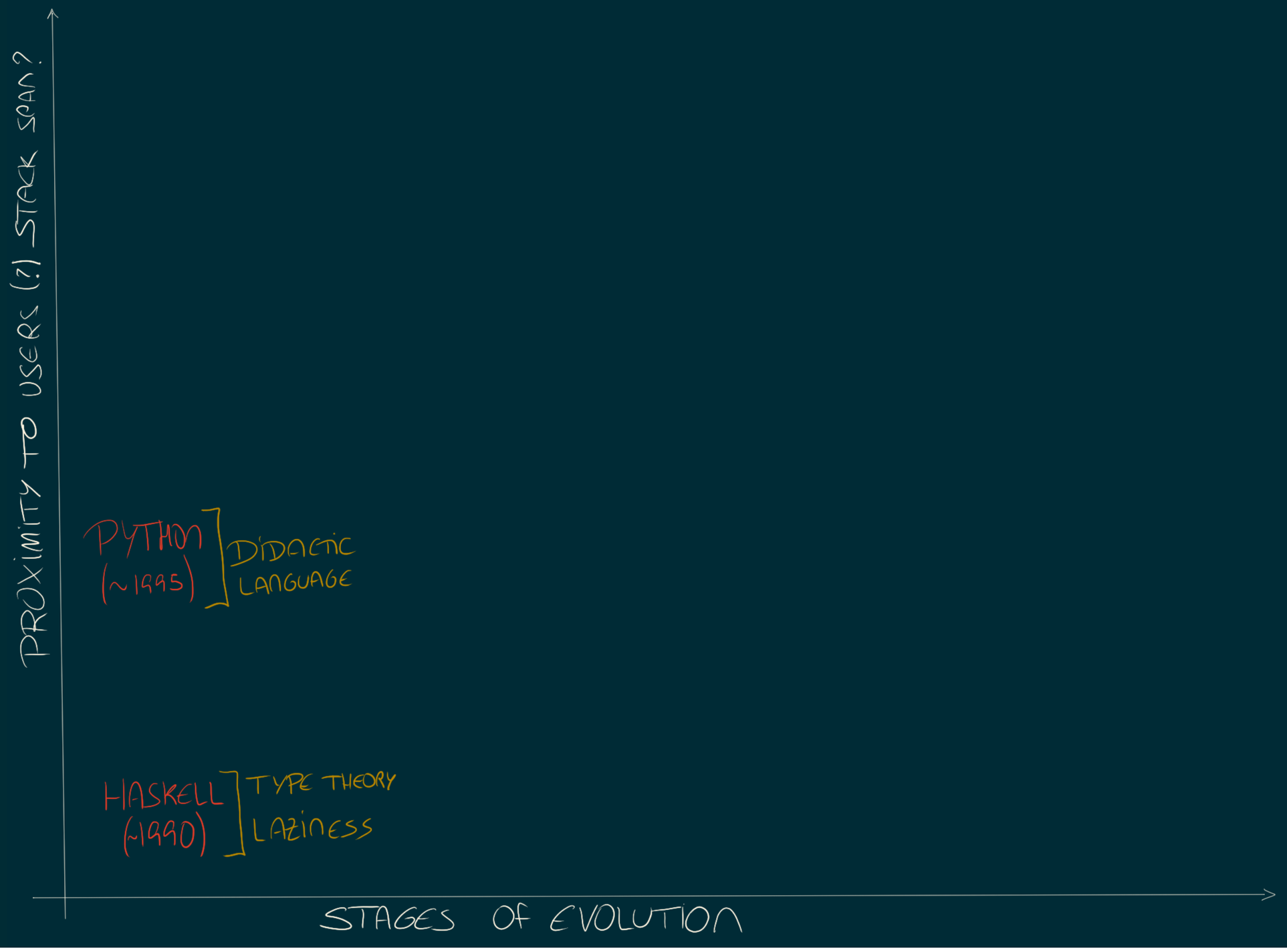
WHAT ARE THE TWO EXTREME EVOLUTION STAGES OF A PROGRAMMING LANGUAGE?

1. **JUST CREATED. A TOY. AN EXPERIMENT. AN IDEA.**
2. USED ALL OVER THE STACK. NOT SHINY **IN A RESUMÉ.**

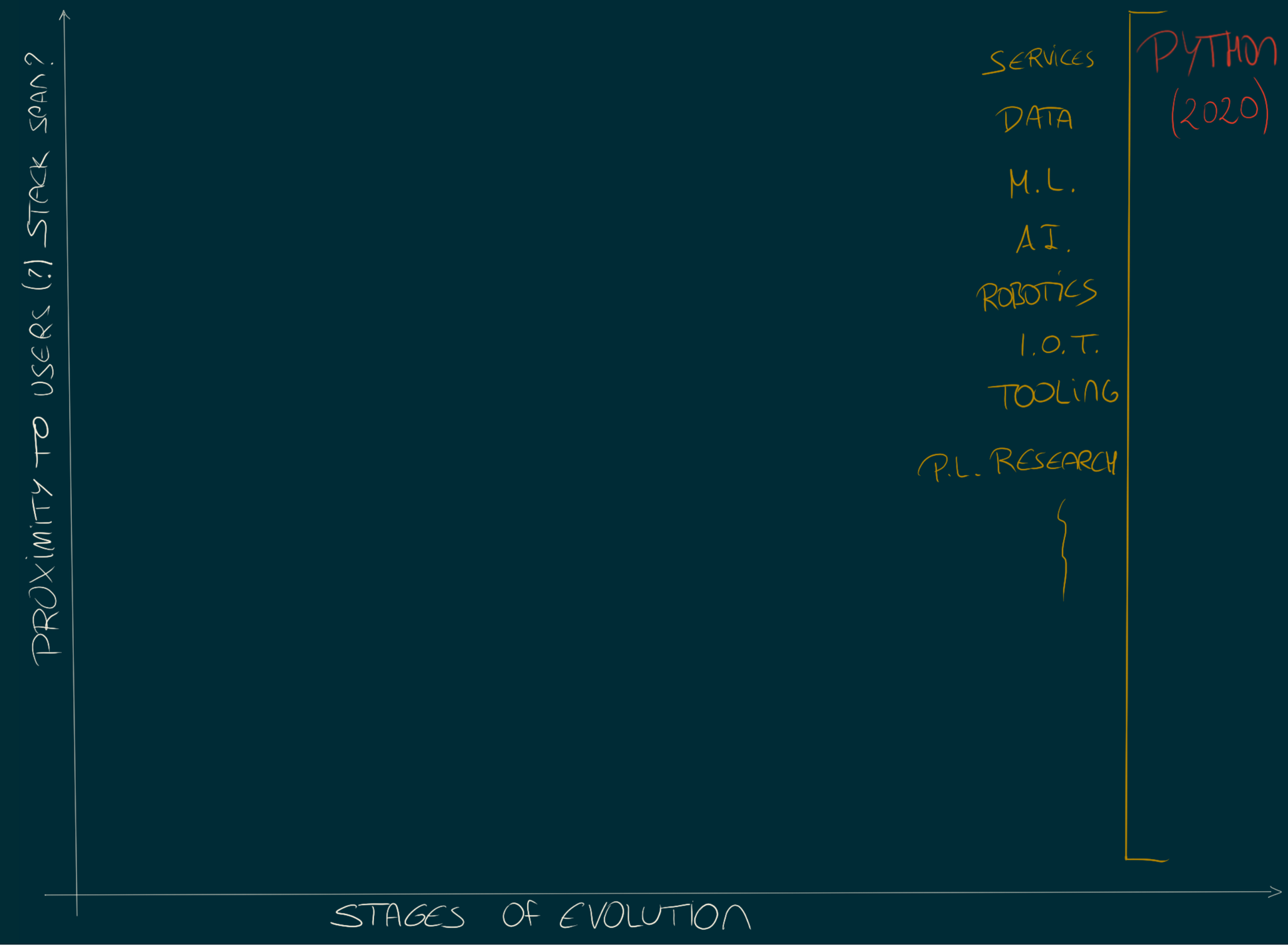
We can imagine languages at stage 1 as when they started. Python started as the evolution of
a teaching language. Haskell started as an investigation in lazy functional languages. Now,
Python is used everywhere and **not shiny**. But, Haskell is *still* shiny



This is a possible view of how a language evolves from a "beginning" to an "end". These two languages started as an investigation of some concepts. Python, as the evolution of a simple language to make teaching easy. Haskell to explore some type-level ideas like laziness. Arguably, we could say the vertical axis represents "visibility to the user". At this *genesis* stage teaching language would thus be more "visible" than one used in a specific corner of programming language research.


PROXIMITY TO USERS (?) STACK SPAN?
PYTHON (~1995)
DIDACTIC LANGUAGE
HASKELL (1990)
TYPE THEORY
LAZINESS
STAGES OF EVOLUTION

Jump to the present, and Python is probably one of the most "visible" programming languages, possibly only rivaled by Java. It is no longer in a teaching niche



It has expanded across most of the problem space,

increasing visibility

But... Arguably, Haskell has expanded likewise. If we think of the vertical axis as how much of the problem space (ML, data, backend, frontend, etc) a language "takes", we can see a climatic pattern emerging: a language will expand until it covers all available space. What, then, distinguishes a "really" commoditised language vs another, not so commoditized?

IN A NORMAL WARDLEY MAP WE WOULD CALL THESE TWO STAGES

1. GENESIS
2. COMMODITY (OR UTILITY)

ARE THEY?

We could argue for *Genesis*, although I prefer *Research Phase* in this scenario. But what about commodity? To call something a commodity, we need to identify the user and see if there is really a fully interchangeable source.

WHO IS THE USER?

OPTION A: THE COMPANY CHOOSING THE LANGUAGE TO USE

OPTION B: THE DEVELOPER CHOOSING THE LANGUAGE TO LEARN

OPTION A: **THE COMPANY CHOOSING THE LANGUAGE TO USE**

OPTION B: **THE DEVELOPER CHOOSING THE LANGUAGE TO LEARN**

OPTION A: COMPANY

COMMODITY COULD MEAN:

You could argue that the commodity in this case is the combination `developer+language`

OPTION A: **COMPANY**

COMMODITY COULD MEAN:

> **ACESS TO A LARGE POOL OF TALENT** (LARGE MARKET)

You could argue that the commodity in this case is the combination `developer+language`

OPTION A: COMPANY

COMMODITY COULD MEAN:

- ACESS TO A LARGE POOL OF TALENT (LARGE MARKET)
- ACCESS TO A VARIED POOL OF TALENT (RANGE OF SKILLS)

You could argue that the commodity in this case is the combination `developer+language`

OPTION B: DEVELOPER

COMMODITY COULD MEAN:

You could argue that the commodity in this case is the combination `job+language`

OPTION B: DEVELOPER

COMMODITY COULD MEAN:

- USABILITY ACROSS A RANGE OF PROBLEMS (GENERALITY)

You could argue that the commodity in this case is the combination `job+language`

OPTION B: DEVELOPER

COMMODITY COULD MEAN:

- USABILITY ACROSS A RANGE OF PROBLEMS (GENERALITY)
  - LARGE INDIVIDUAL DEMAND (ABILITY TO CHOOSE)

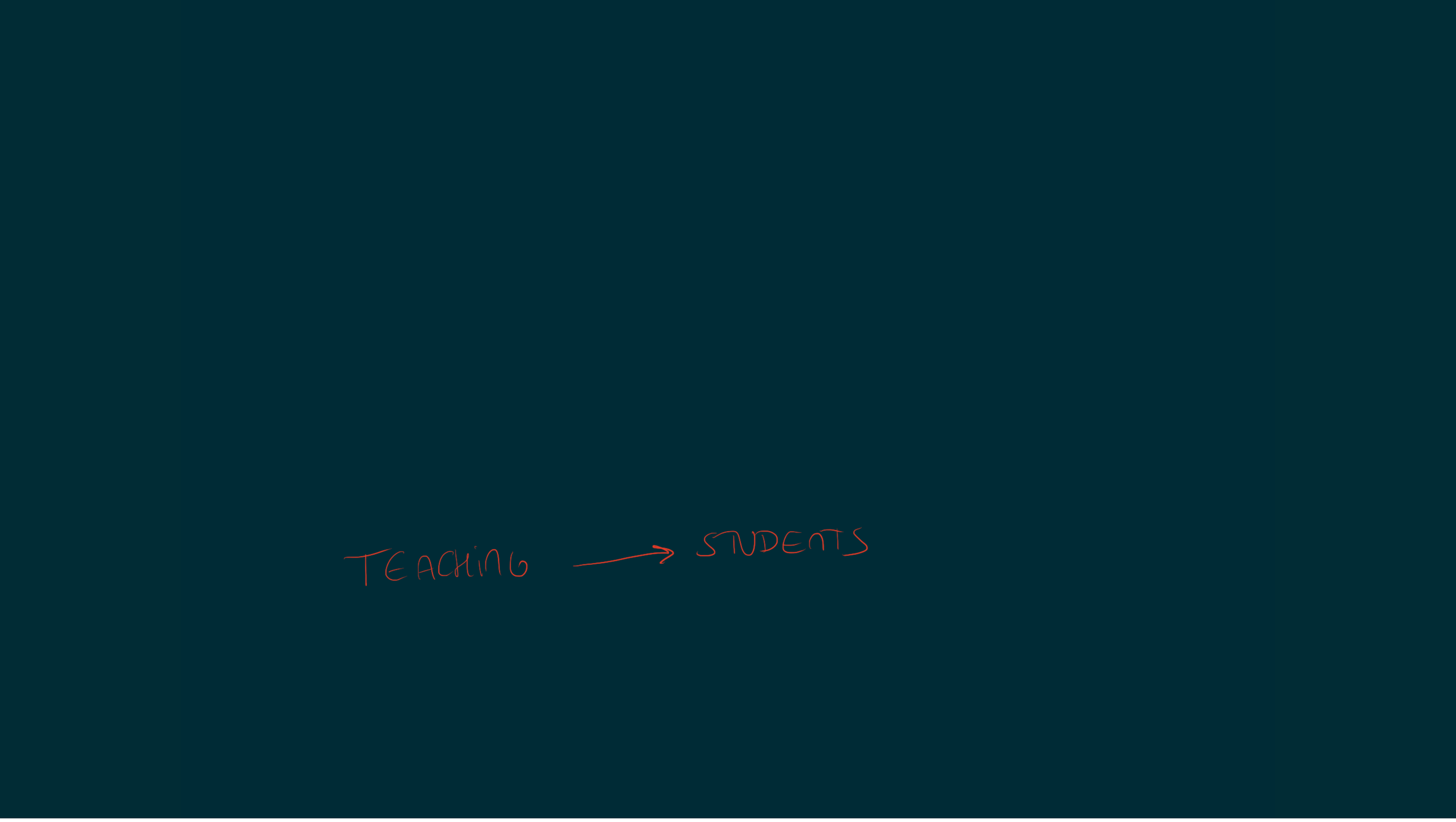
You could argue that the commodity in this case is the combination `job+language`

FEEDBACK LOOP OR DEADLOCK?

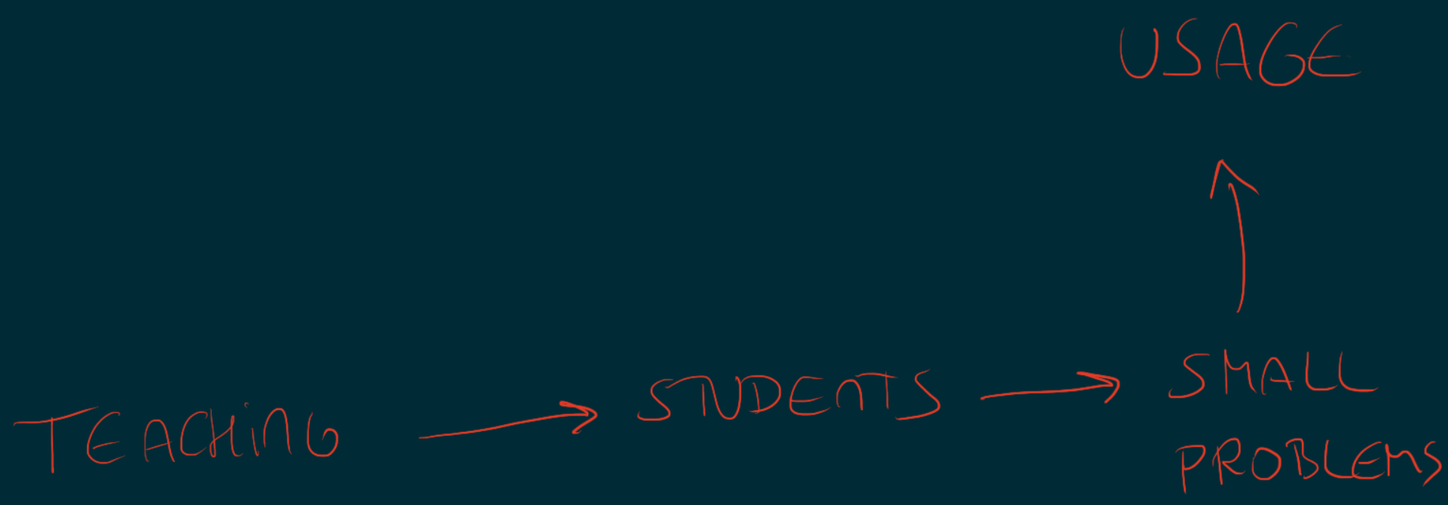
A large pool of talent makes it more likely for a language to be picked by companies,
and the more companies picking a language makes it easier for a language to be picked by individual developers

OPEN QUESTIONS...

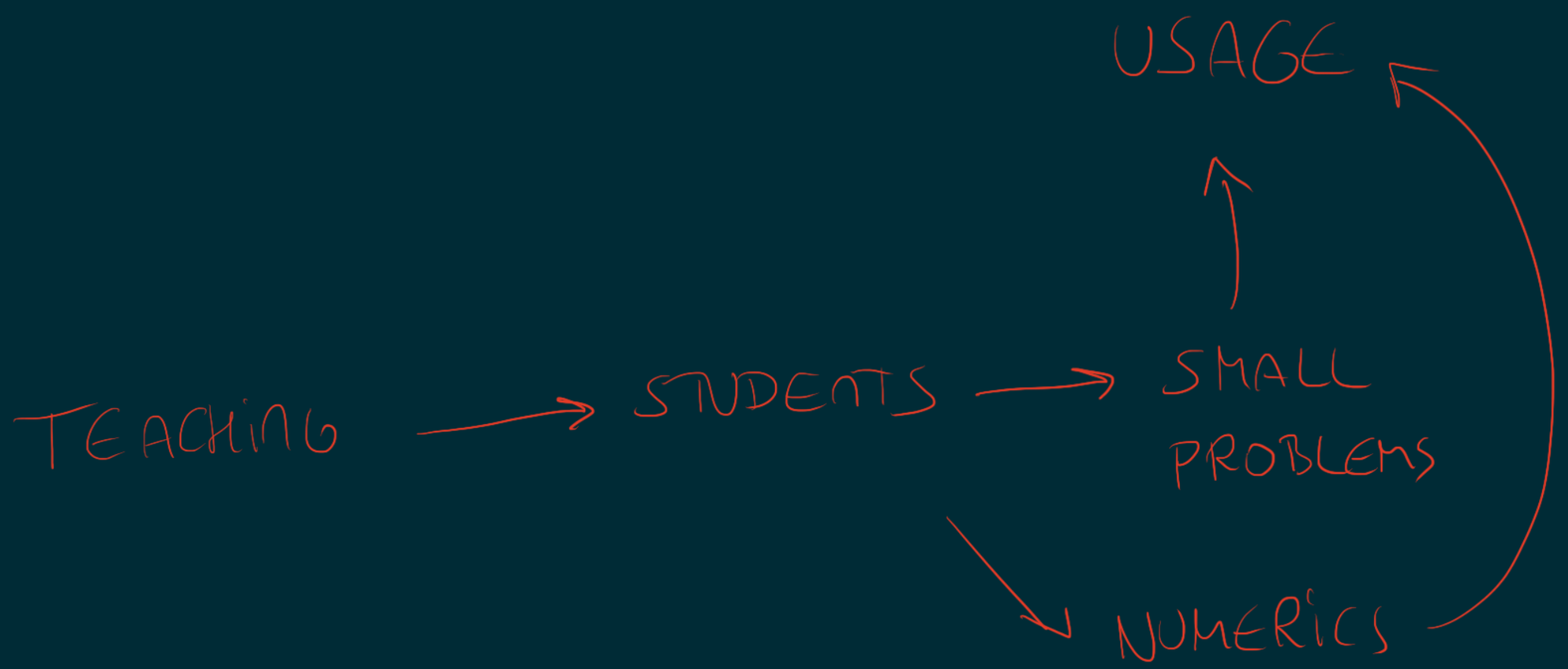
How is this deadlock/feedback loop established? Let's have a look at a specific case



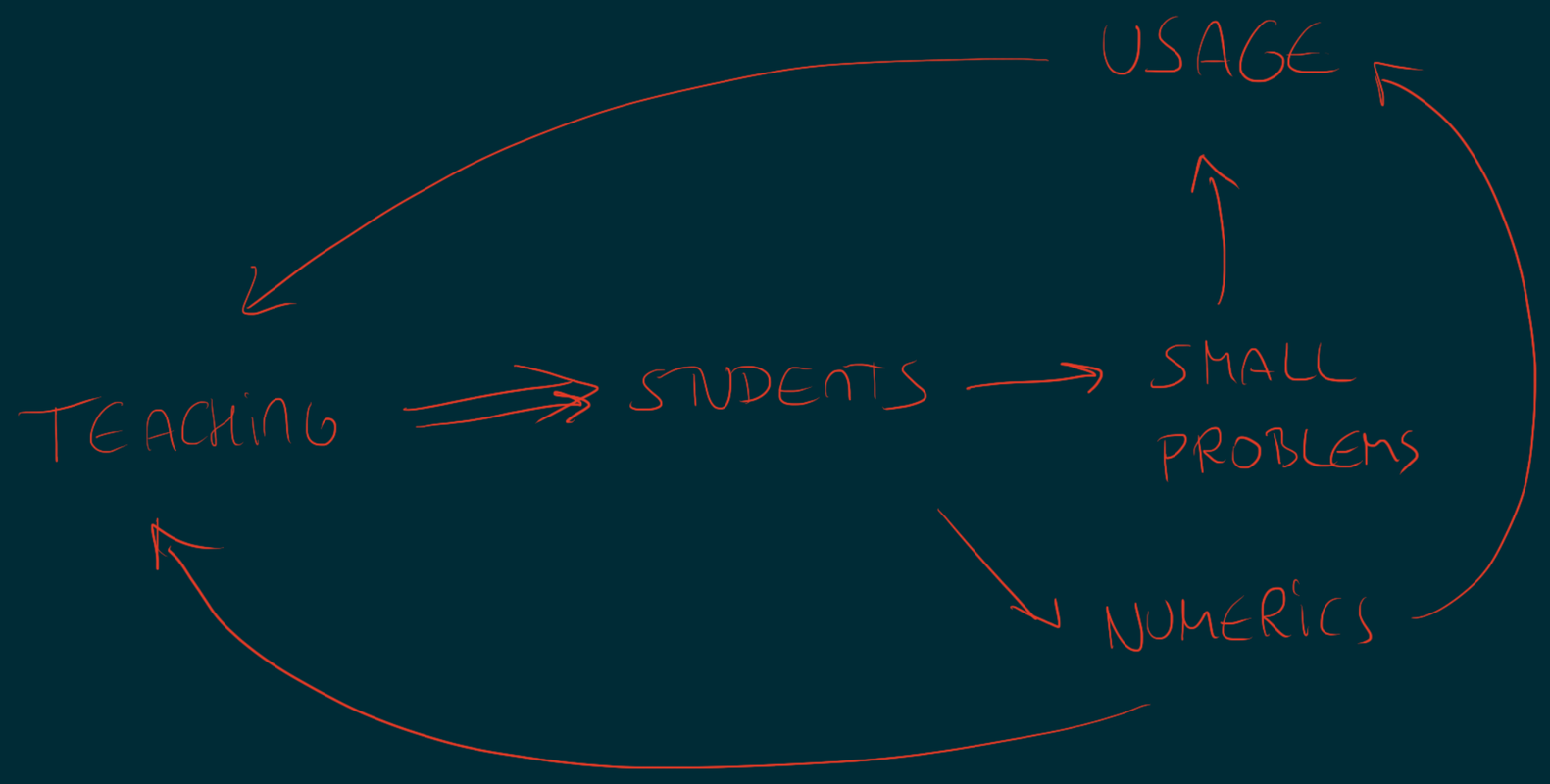
Let's have a look at Python, which was starting as a teaching language due to its simplicity



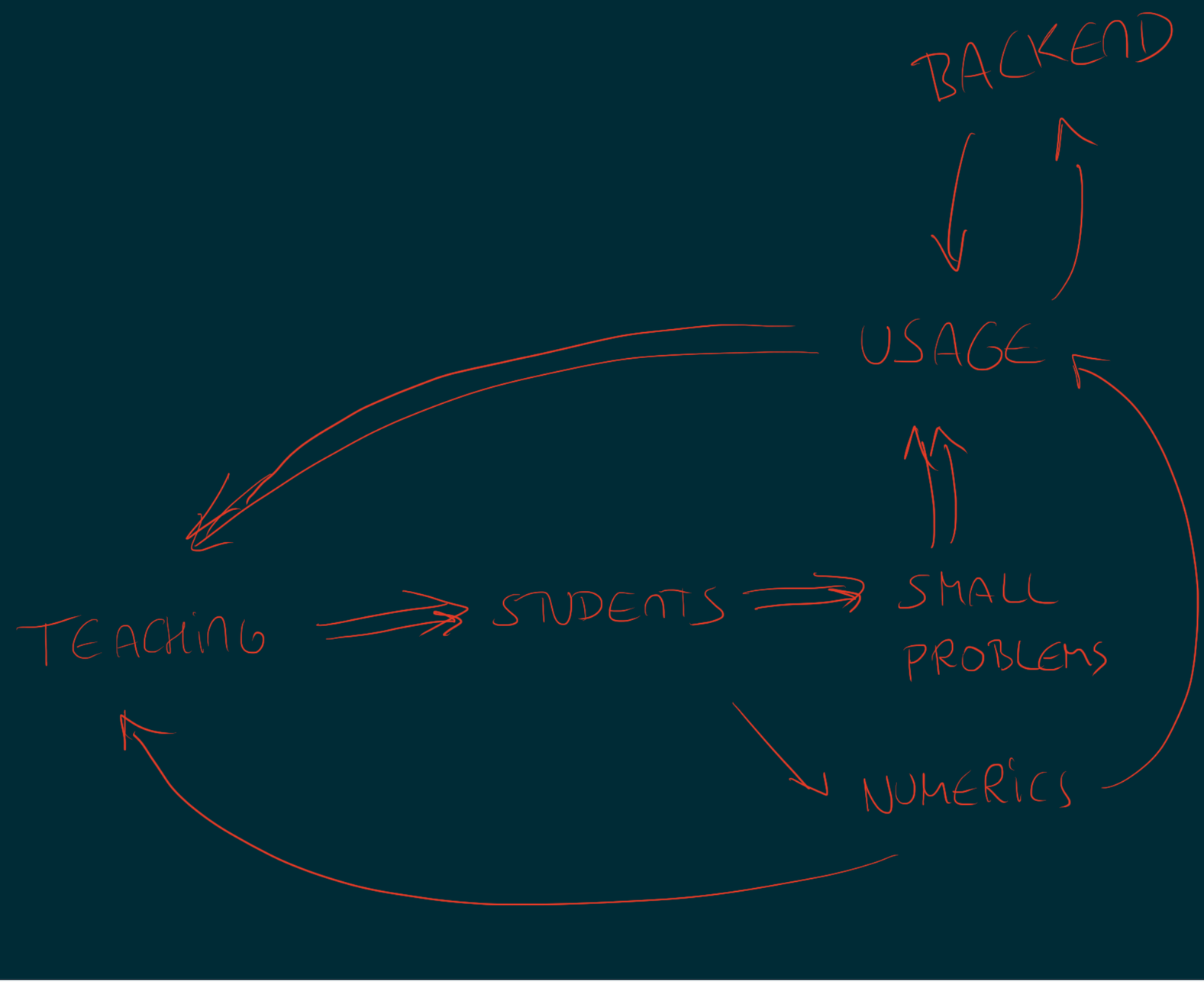
As students pick it up, they start using it for small problems (small tools, simple automation), usage increases



Eventually some students pick it up to simplify working with numerical analysis code. Usage increases again



More usage and more areas useful for teaching, means it's more used for teaching



Which means more students, more small problems solved, more usage. Eventually it starts being used for backend work, which again compounds on usage which increases the rest. A nice self-reinforcing feedback loop. Note that Java (with the additional push of by-committee creation) was also heavily used as a teaching language.

AS EASY AS THAT? TEACHING?

This can't be the answer, though. Case in point, Lisp was used as a teaching language between the 70s and the 90s. And it didn't catch up, even if it was used mostly everywhere. Its failure can be chalked up to several reasons (lack of enough computing power, complex abstractions, bad connotations with the AI winter) but it still raises the question of what was the big difference. Likewise, Haskell is relatively used as a teaching language, and it clearly has not the same level of usage or exposure. Here, abstraction complexity may be to blame, and there is a possible answer in *simplicity*. How easy will be the next super mainstream language?

OPEN QUESTIONS

OPEN QUESTIONS

- WHAT DRIVES PROGRAMMING LANGUAGE ADOPTION FEEDBACK LOOPS, IF IT IS NOT TEACHING?

OPEN QUESTIONS

- WHAT DRIVES PROGRAMMING LANGUAGE ADOPTION FEEDBACK LOOPS, IF IT IS NOT TEACHING?
- ARE THERE ANY OTHER CLIMATIC PATTERNS TO IDENTIFY IN THE PROGRAMMING LANGUAGE LANDSCAPE?

OPEN QUESTIONS

- WHAT DRIVES PROGRAMMING LANGUAGE ADOPTION FEEDBACK LOOPS. IF IT IS NOT TEACHING?
- ARE THERE ANY OTHER CLIMATIC PATTERNS TO IDENTIFY IN THE PROGRAMMING LANGUAGE LANDSCAPE?
- HOW DO PROGRAMMING LANGUAGES DECAY?

SOME RESOURCES AND INTERESTING REFERENCES

- **S. WARDLEY.** WARDLEY MAPPING ONLINE BOOK
- **M. NIELSEN.** THOUGHT AS A TECHNOLOGY
- **D. MEADOWS.** THINKING IN SYSTEMS
- **M. LIMA.** VISUAL COMPLEXITY: MAPPING PATTERNS OF INFORMATION
- **K. IVERSON.** NOTATION AS A TOOL FOR THOUGHT

QUESTIONS?


THANKS!

GET THE SLIDES FROM MY GITHUB:

github.com/rberenguel/

THE REPOSITORY IS

commoditisation-languages

FURTHER REFERENCES

- **S. WARDLEY.** PLAYING CHESS WITH COMPANIES **(OSCON 2017. ON SAFARI BOOKS ONLINE)**
  - **S. WARDLEY.** CROSSING THE RIVER BY FEELING THE STONES
- **G. ADZIC.** SKIP THE FIRST THREE MONTHS OF DEVELOPMENT FOR YOUR NEXT APP

EOF