The Escape From Flatland

🐛⚗️ Building the Languages of the Future, Today 🔮🦋

Language is an instrument
of human reason, and ***not merely a medium***
for the expression of thought

– George Boole

Daring ideas are like
chess pieces moved forward.
They may be ***beaten***,
but they may ***start a winning game.***

– Goethe

Brooklyn Zelenka

@expede

Brooklyn Zelenka

@expede

- CTO at Fission (https://fission.codes)
  - Far edge apps ("post-serverless")
  - Goal: make back-ends and DevOps obsolete

Brooklyn Zelenka

@expede

- CTO at Fission (https://fission.codes)
  - Far edge apps ("post-serverless")
  - Goal: make back-ends and DevOps obsolete
- PLT, VMs, DSys

Brooklyn Zelenka

@expede

- CTO at Fission (https://fission.codes)
  - Far edge apps ("post-serverless")
  - Goal: make back-ends and DevOps obsolete
- PLT, VMs, DSys
- Original author of Witchcraft, Algae, Exceptional, etc

Brooklyn Zelenka

@expede

- CTO at Fission (https://fission.codes)
  - Far edge apps ("post-serverless")
  - Goal: make back-ends and DevOps obsolete
- PLT, VMs, DSys
- Original author of Witchcraft, Algae, Exceptional, etc
- Standards: UCAN (editor), EIPs, FVM, Multiformats, others

Brooklyn Zelenka

@expede

- CTO at Fission (https://fission.codes)
  - Far edge apps ("post-serverless")
  - Goal: make back-ends and DevOps obsolete
- PLT, VMs, DSys
- Original author of Witchcraft, Algae, Exceptional, etc
- Standards: UCAN (editor), EIPs, FVM, Multiformats, others
- Founded VanFP, VanBEAM, DSys Reading Group (join us!)

Brooklyn Zelenka

@expede

- CTO at Fission (https://fission.codes)
  - Far edge apps ("post-serverless")
  - Goal: make back-ends and DevOps obsolete
- PLT, VMs, DSys
- Original author of Witchcraft, Algae, Exceptional, etc
- Standards: UCAN (editor), EIPs, FVM, Multiformats, others
- Founded VanFP, VanBEAM, DSys Reading Group (join us!)

https://lu.ma/distributed-systems

Meta 🔄

Two Keynotes, Both Alike in Dignity 🎭

Meta 🔄

Two Keynotes, Both Alike in Dignity 🎭

Meta 🔄

Two Keynotes, Both Alike in Dignity 🎭

Meta 🔄

Two Keynotes, Both Alike in Dignity 🎭

Part I: Empex MTN 🇺🇸

Part II: CodeBEAM EU 🇸🇪

Meta 🔄

Growth Mindset

Meta 🔄

Growth Mindset

We ❤️ Elixir

Meta 🔄

Growth Mindset

We ❤️ Elixir

It's important to think ***critically*** about our tools

Meta 🔄

Growth Mindset

We ❤️ Elixir

It's important to think ***critically*** about our tools

We need to hold Elixir to ***the highest standard***

Meta 🔄

Growth Mindset

We ❤️ Elixir

It's important to think ***critically*** about our tools

We need to hold Elixir to ***the highest standard***

Let's ask ***uncomfortable questions***

Meta 🔄

Growth Mindset

We ❤️ Elixir

It's important to think ***critically*** about our tools

We need to hold Elixir to ***the highest standard***

Let's ask ***uncomfortable questions***

Growth requires ***dissatisfaction & inspiration***

Meta 🔄

Growth

It's imp ur tools

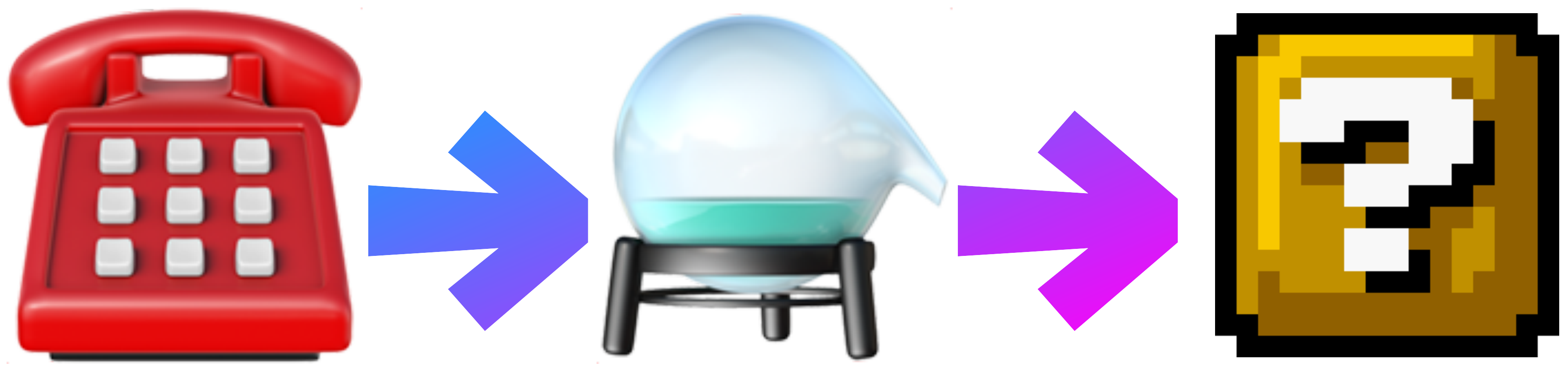
We nee *ndard*

Let *ns*

Growth r *iration*

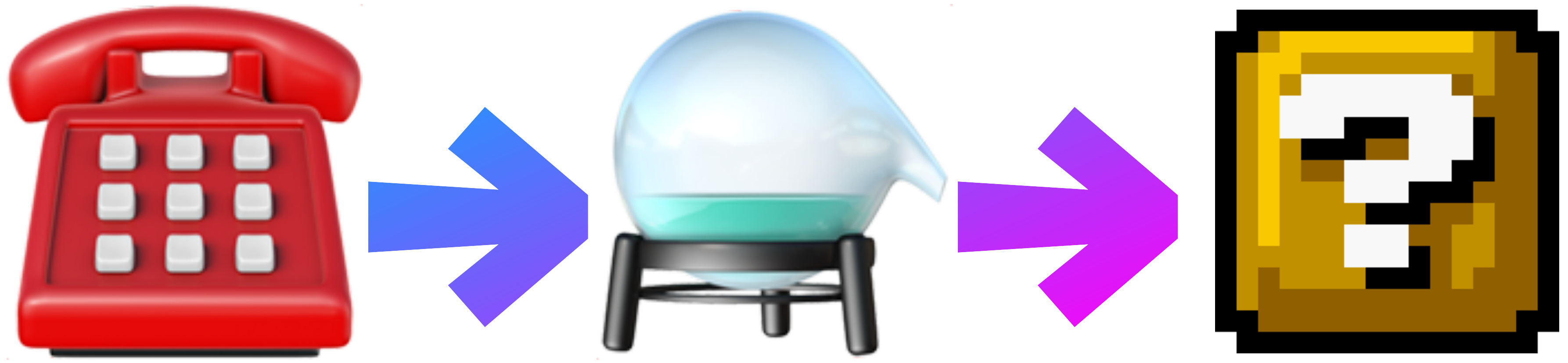
The Children of Elixir

Manifesto



The Children of Elixir

Manifesto



Manifesto ☎️→🫖→❓

My Obsession

Manifesto ☎️→🫧→❓

My Obsession

Manifesto ☎️→🔮→❓

My Obsession

The BEAM does ***so much right*** 👏

Manifesto ☎️→🔮→❓

My Obsession

The BEAM does ***so much right*** 👏

In many ways, we're actually ***ahead*** of the industry

Manifesto ☎️→🔮→❓

My Obsession

The BEAM does ***so much right*** 👏

In many ways, we're actually ***ahead*** of the industry

(mainly using ideas from the late 80s 🫣)

Manifesto ☎️→🔮→❓

My Obsession

The BEAM does ***so much right*** 👏

In many ways, we're actually ***ahead*** of the industry

(mainly using ideas from the late 80s 🫣)

…but our lead ***won't last***…

Manifesto ☎️→🔮→❓

My Obsession

The BEAM does ***so much right*** 👏

In many ways, we're actually ***ahead*** of the industry

(mainly using ideas from the late 80s 🫣)

…but our lead ***won't last***…

Where do we go from here?

Manifesto ☎️→🔮→❓

Our Code is Too "Flat"

Manifesto ☎️→🔮→❓

Our Code is Too "Flat"

Code in 2022 is ***needlessly difficult & complex!***

Manifesto ☎️→🔮→❓

Our Code is Too "Flat"

Code in 2022 is ***needlessly difficult & complex!***

Manifesto ☎️→🔮→❓

Our Code is Too "Flat"

Code in 2022 is ***needlessly difficult & complex!***

If software is going to continue eating the world, it needs to be ***faster, more flexible, clearer, correct, and teachable***

Manifesto ☎️→🔮→❓

Our Code is Too "Flat"

Code in 2022 is ***needlessly difficult & complex!***

If software is going to continue eating the world, it needs to be ***faster, more flexible, clearer, correct, and <u>teachable</u>***

Manifesto ☎️→🔮→❓

Our Code is Too "Flat"

Code in 2022 is ***needlessly difficult & complex!***

If software is going to continue eating the world, it needs to be ***faster, more flexible, clearer, correct, and teachable***

How do we ***build more structure*** from our existing parts?
How do we ***add dimension?***

Manifesto ☎️→⚗️→❓

Our Code is Too "Flat"

Code in 2022 is ***needlessly difficult & complex!***

If software is going to continue eating the world, it needs to be ***faster, more flexible, clearer, correct, and teachable***

How do we ***build more structure*** from our existing parts?
How do we ***add dimension?***

Manifesto ☎️→🔮→❓

Our Code is Too "Flat"

Code in 2022 is ***needlessly difficult & complex!***

If software is going to continue eating the world, it needs to be ***faster, more flexible, clearer, correct, and teachable***

How do we ***build more structure*** from our existing parts?

How do we ***add dimension?***

Manifesto

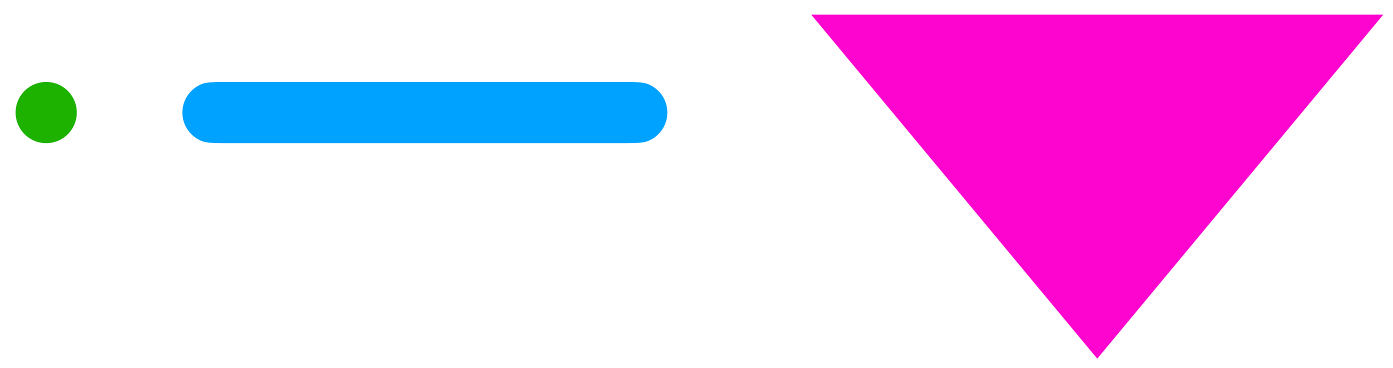
Our Code is Too "Flat"

Code in 2022 is ***needlessly difficult & complex!***

If software is going to continue eating the world, it needs to be ***faster, more flexible, clearer, correct, and teachable***

How do we ***build more structure*** from our existing parts?

How do we ***add dimension?***



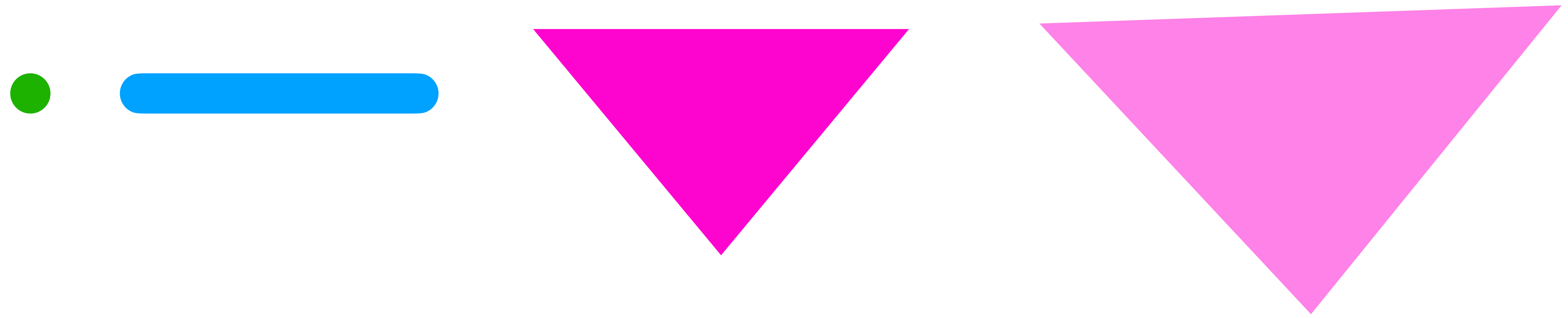
Manifesto ☎️→🔮→❓

Our Code is Too "Flat"

Code in 2022 is ***needlessly difficult & complex!***

If software is going to continue eating the world, it needs to be ***faster, more flexible, clearer, correct, and teachable***

How do we ***build more structure*** from our existing parts?
How do we ***add dimension?***



Manifesto

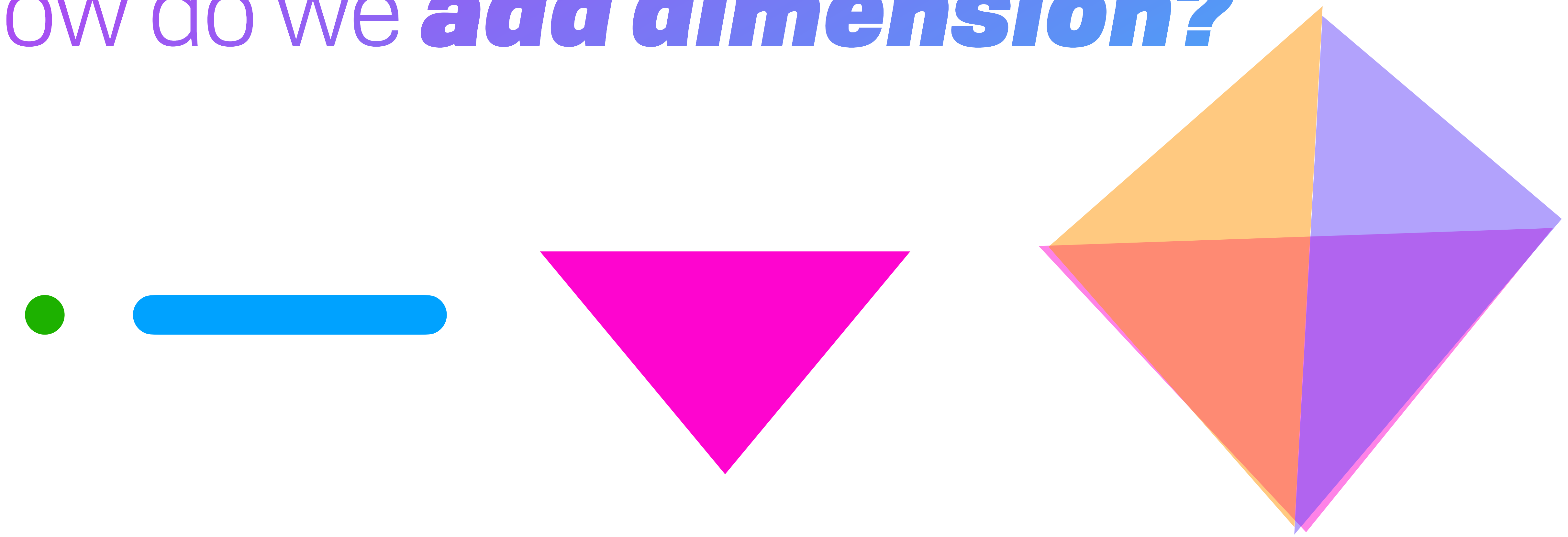
Our Code is Too "Flat"

Code in 2022 is ***needlessly difficult & complex!***

If software is going to continue eating the world, it needs to be ***faster, more flexible, clearer, correct, and teachable***

How do we ***build more structure*** from our existing parts?

How do we ***add dimension?***



Manifesto ☎️→🔮→❓

The Pit of Success

Manifesto ☎️→🔮→❓

The Pit of Success

In stark contrast to a summit, a peak, or a journey across a desert to find victory through many trials and surprises, we want [devs] to ***simply fall into winning practices*** by using our platform and frameworks. ***To the extent that we make it easy to get into trouble we fail.***

– Rico Mariani, Microsoft Research MindSwap 2003

Manifesto ☎️→🔮→❓

Tools For Thought

Manifesto ☎️→🔮→❓

Tools For Thought

Manifesto ☎️→🔮→❓

Tools For Thought

- We have better mental tools than our ancestors
- Abstraction appears 50k-100k years ago
- Arabic numerals > roman numerals
- Metric conversions > Imperial
- 24-hours & 360-degrees have nice divisiors

Manifesto ☎️→🔮→❓

Tools For Thought

- We have better mental tools than our ancestors
- Abstraction appears 50k-100k years ago
- Arabic numerals > roman numerals
- Metric conversions > Imperial
- 24-hours & 360-degrees have nice divisiors

Manifesto ☎️→🔮→❓

Random Walk

Image by Pradyumna Yadav

Manifesto ☎️→🔮→❓

Random Walk

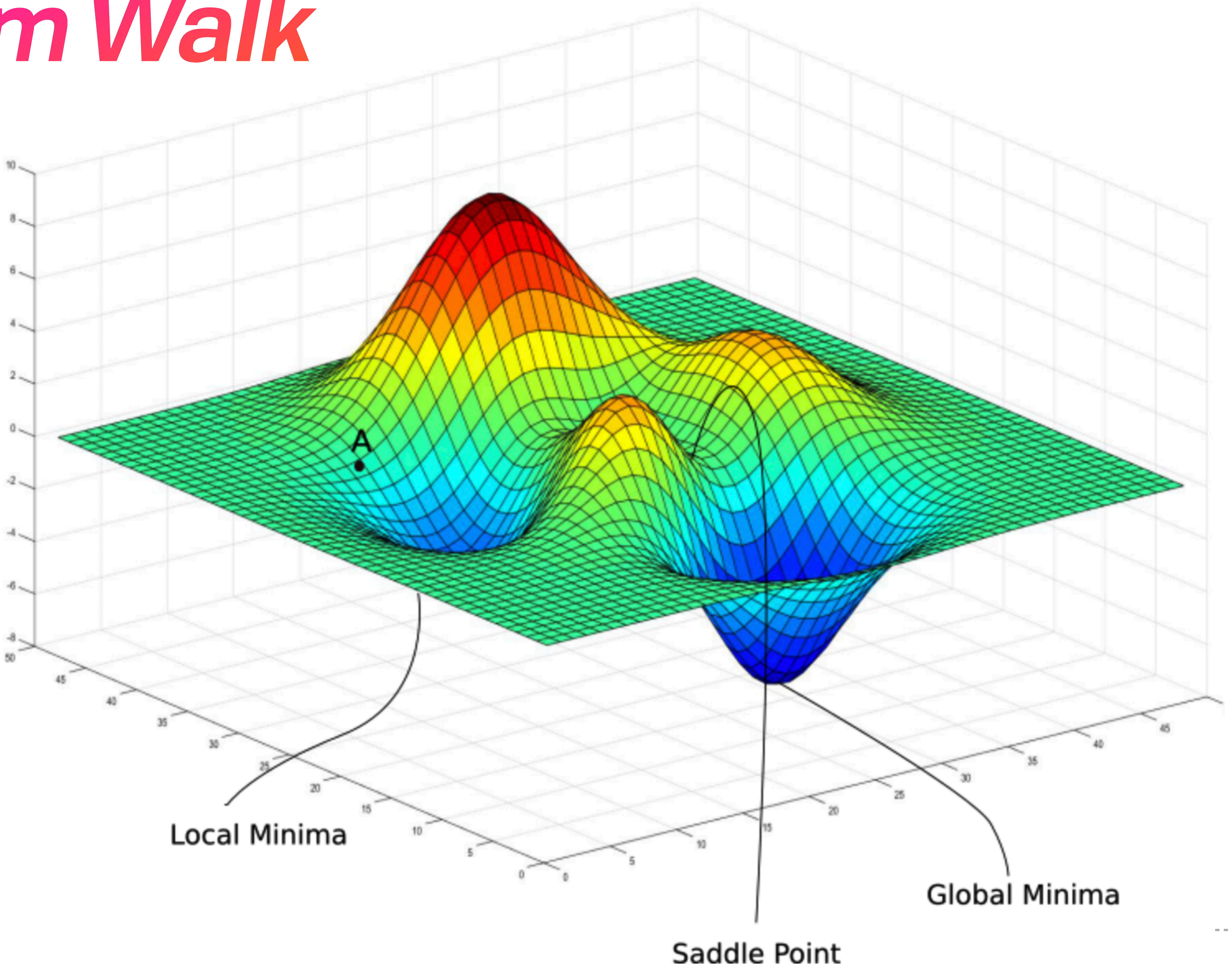


Image by Pradyumna Yadav

Manifesto ☎️→🔮→❓

Random Walk

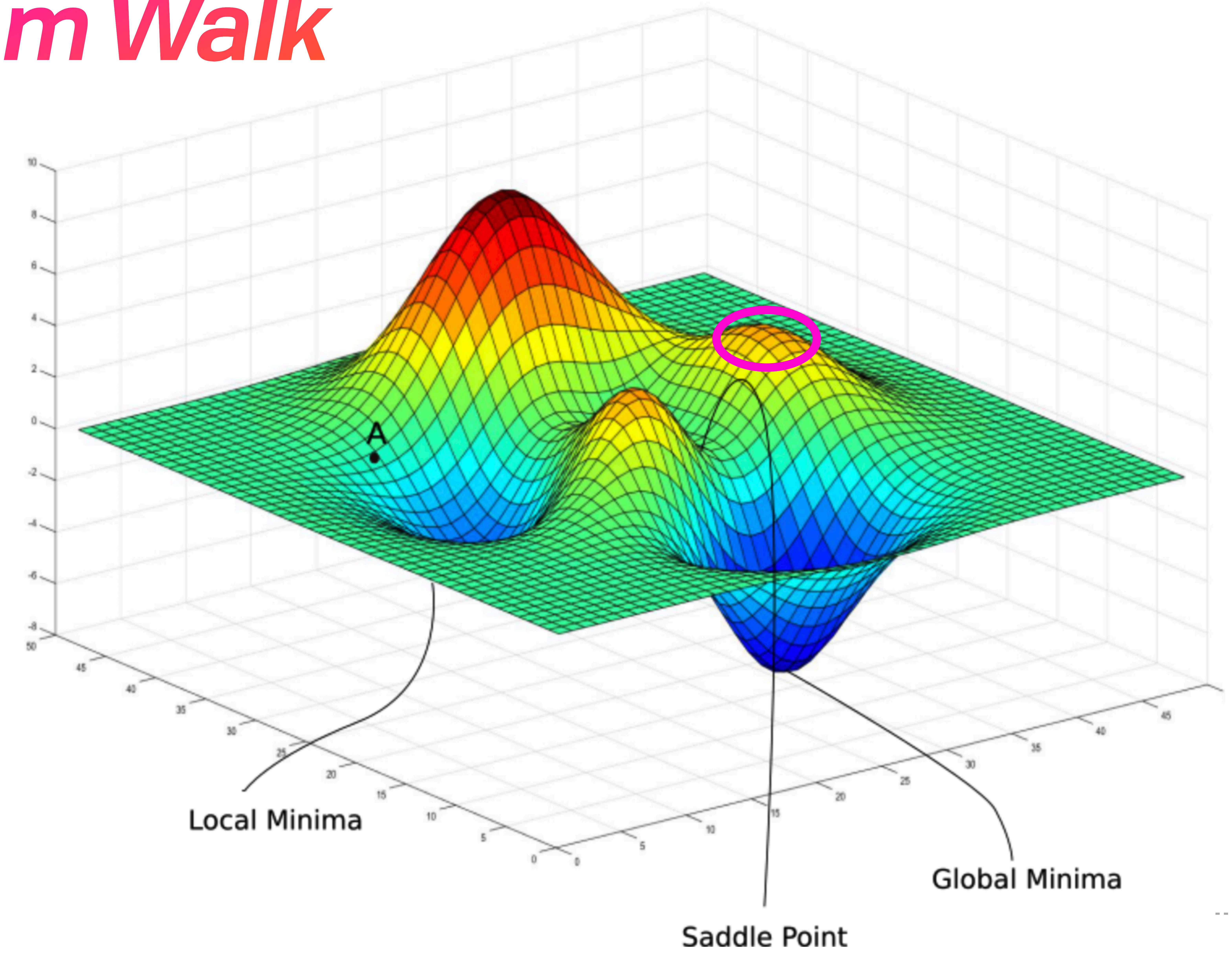


Image by Pradyumna Yadav

Manifesto ☎️→🔮→❓

Random Walk

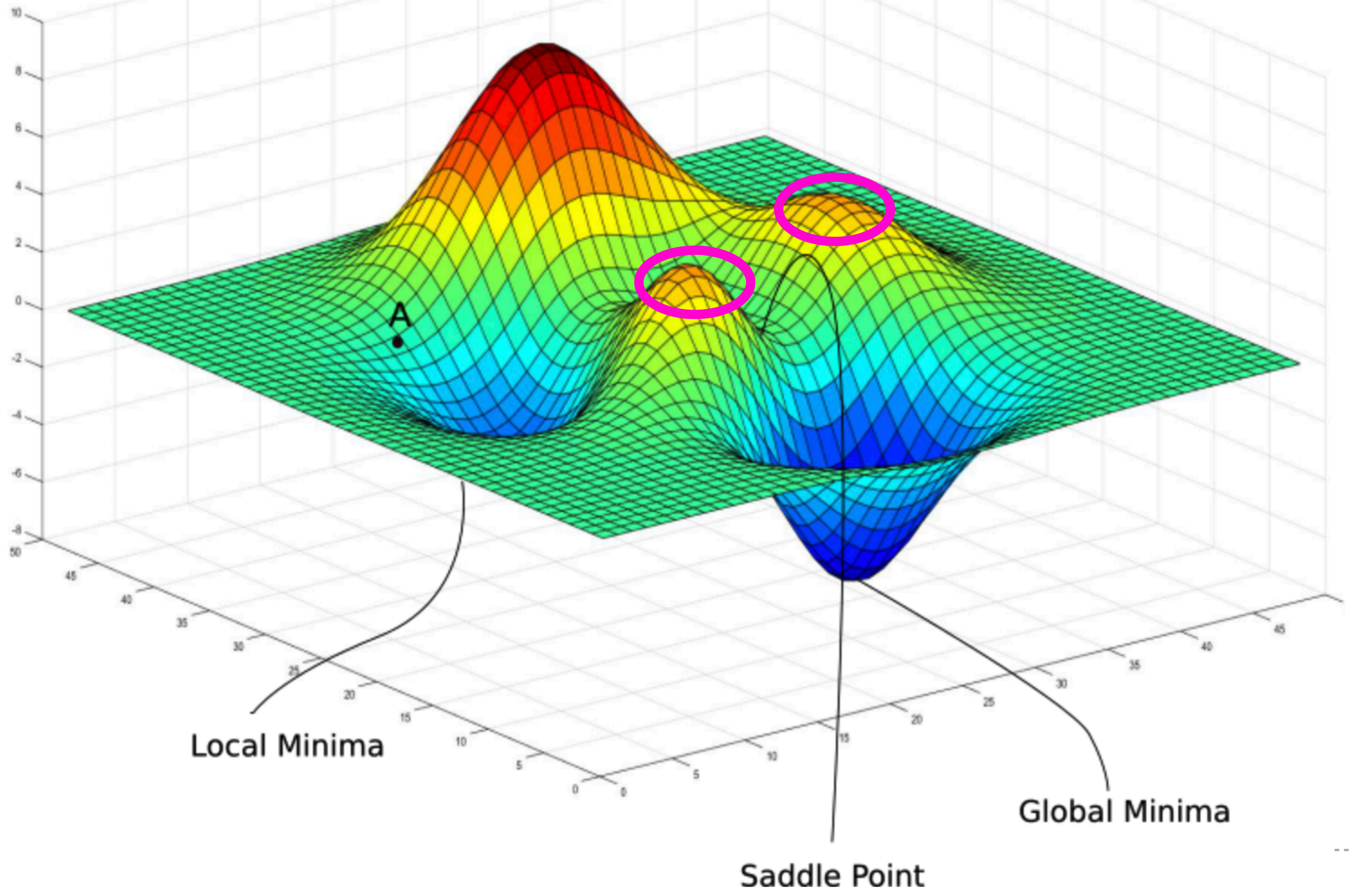


Image by Pradyumna Yadav

Manifesto ☎️→🔮→❓

Random Walk

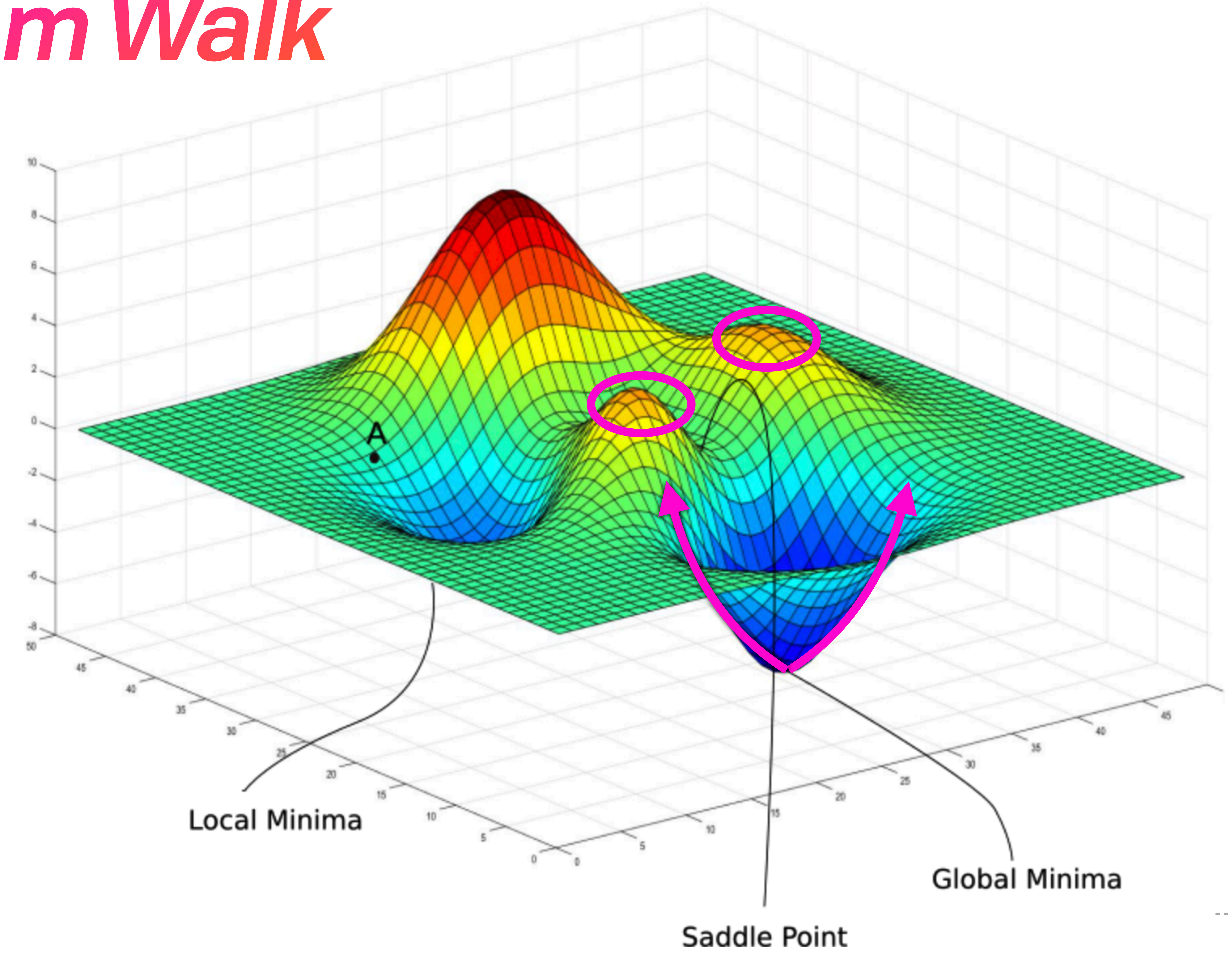


Image by Pradyumna Yadav

Manifesto ☎️→🔮→❓

Random Walk

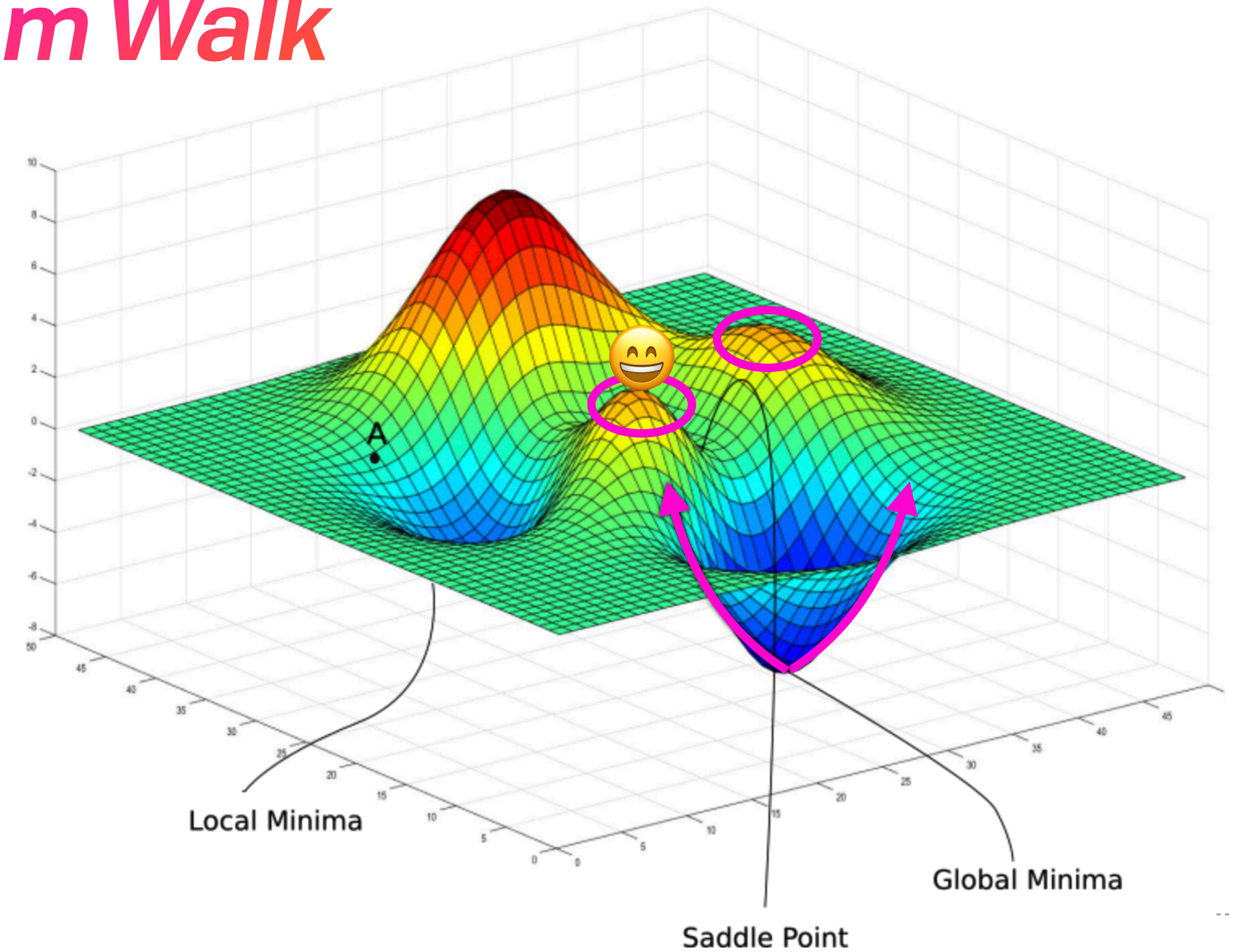


Image by Pradyumna Yadav

Manifesto ☎️→🔮→❓

Random Walk

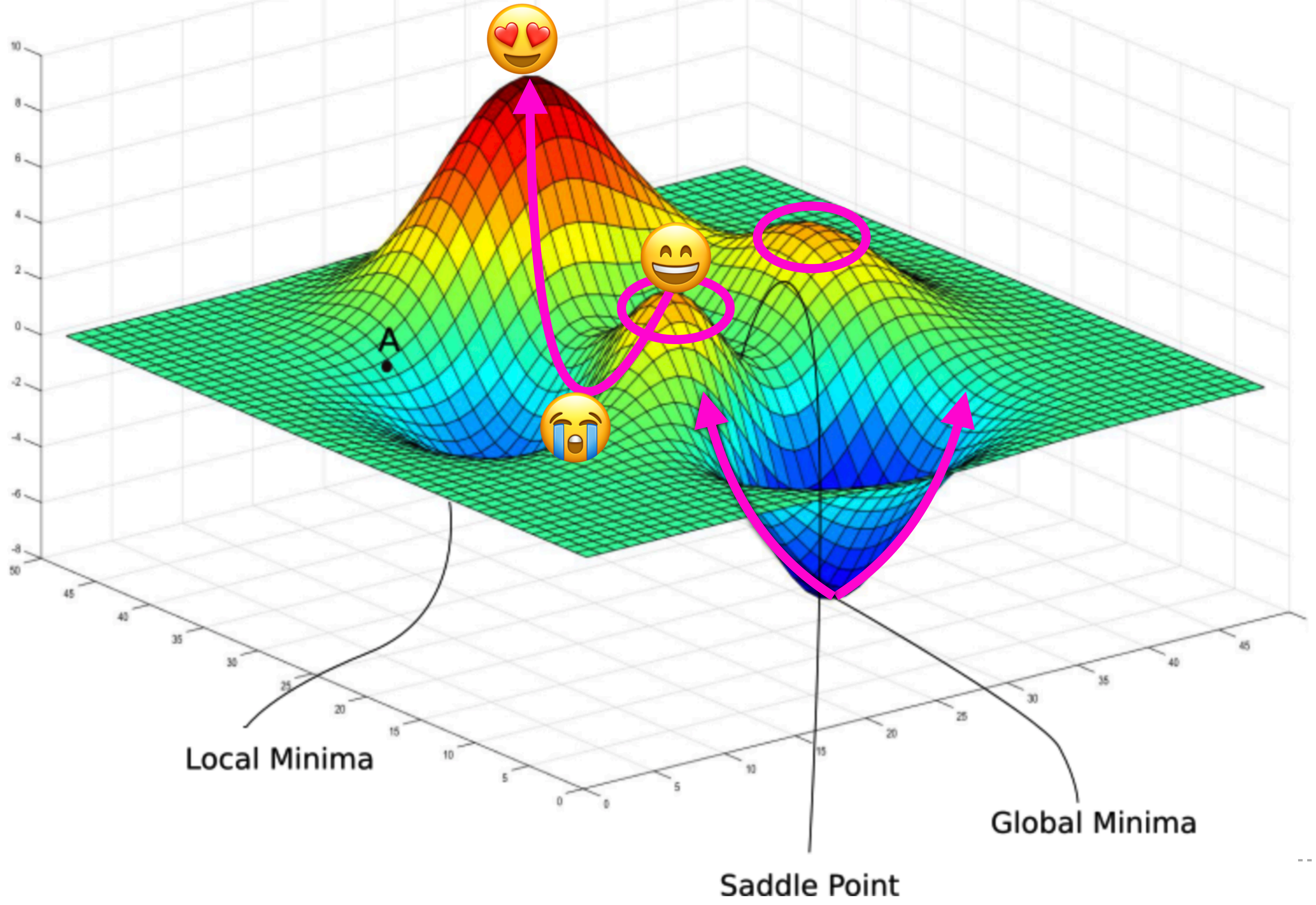


Image by Pradyumna Yadav

Manifesto ☎️→🔮→❓

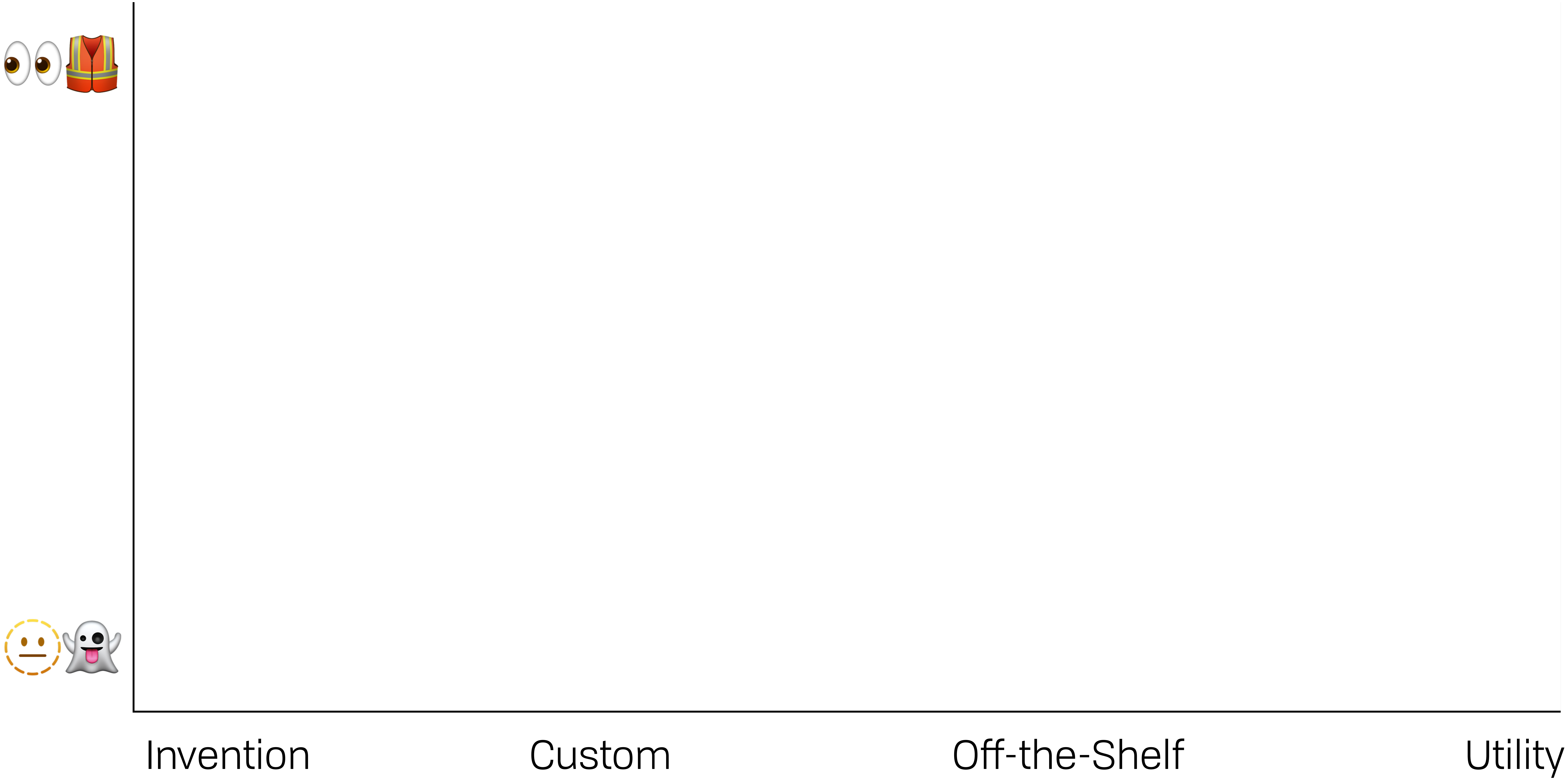
The World Is Changing

Manifesto ☎️→⚗️→❓

The World Is Changing

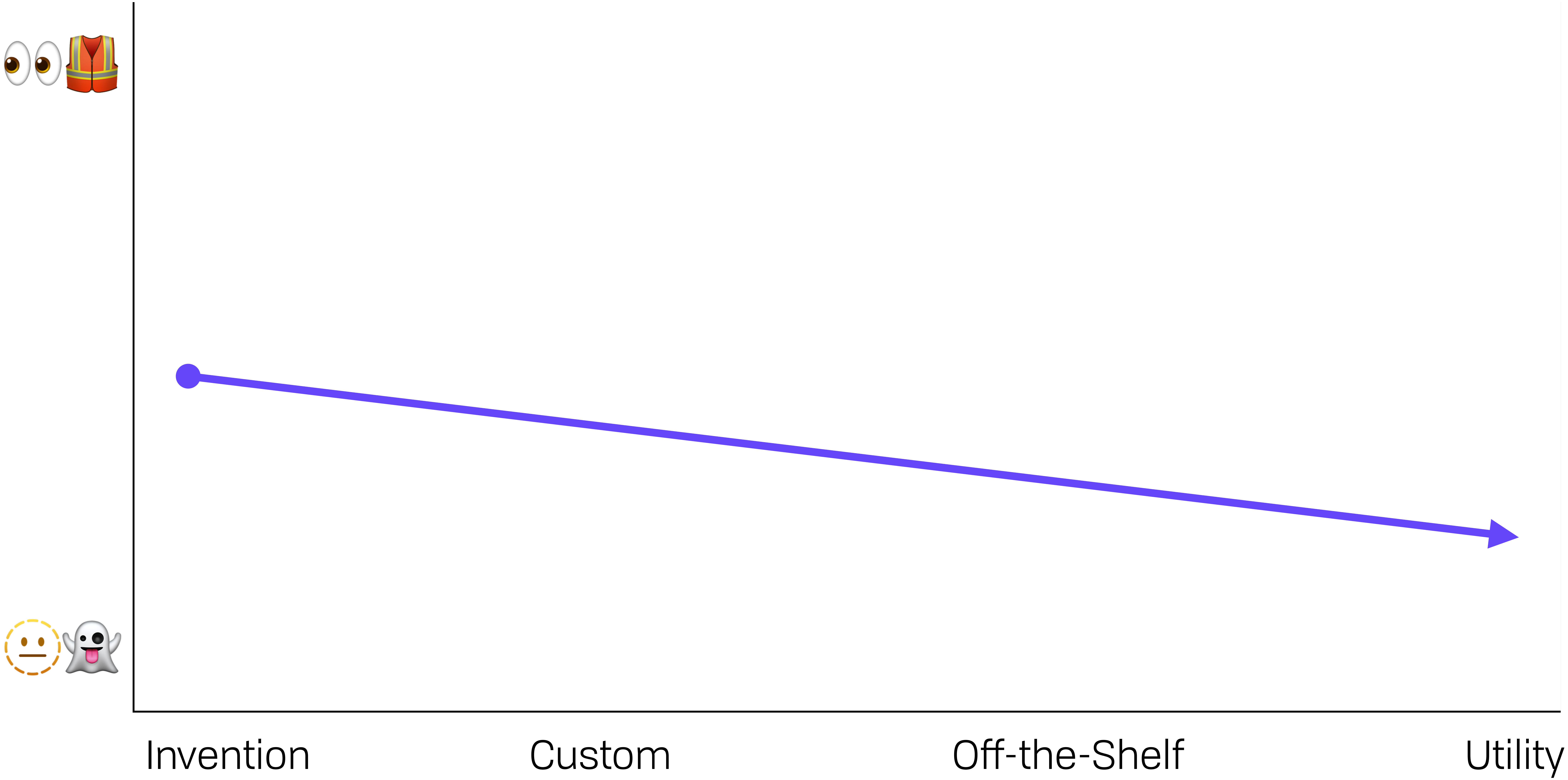
Manifesto ☎️→🔮→❓

The World Is Changing



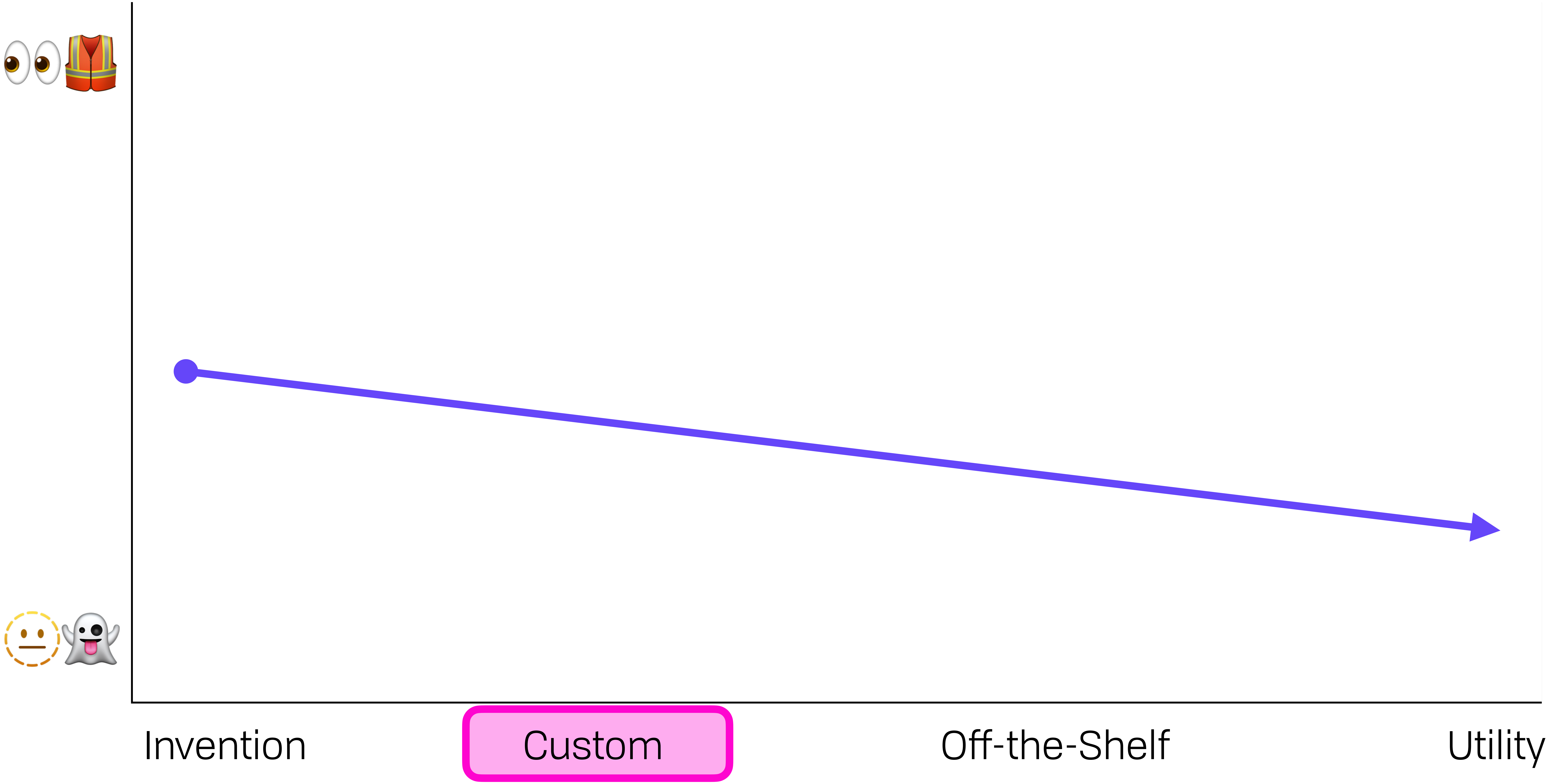
Manifesto ☎️→🫕→❓

The World Is Changing



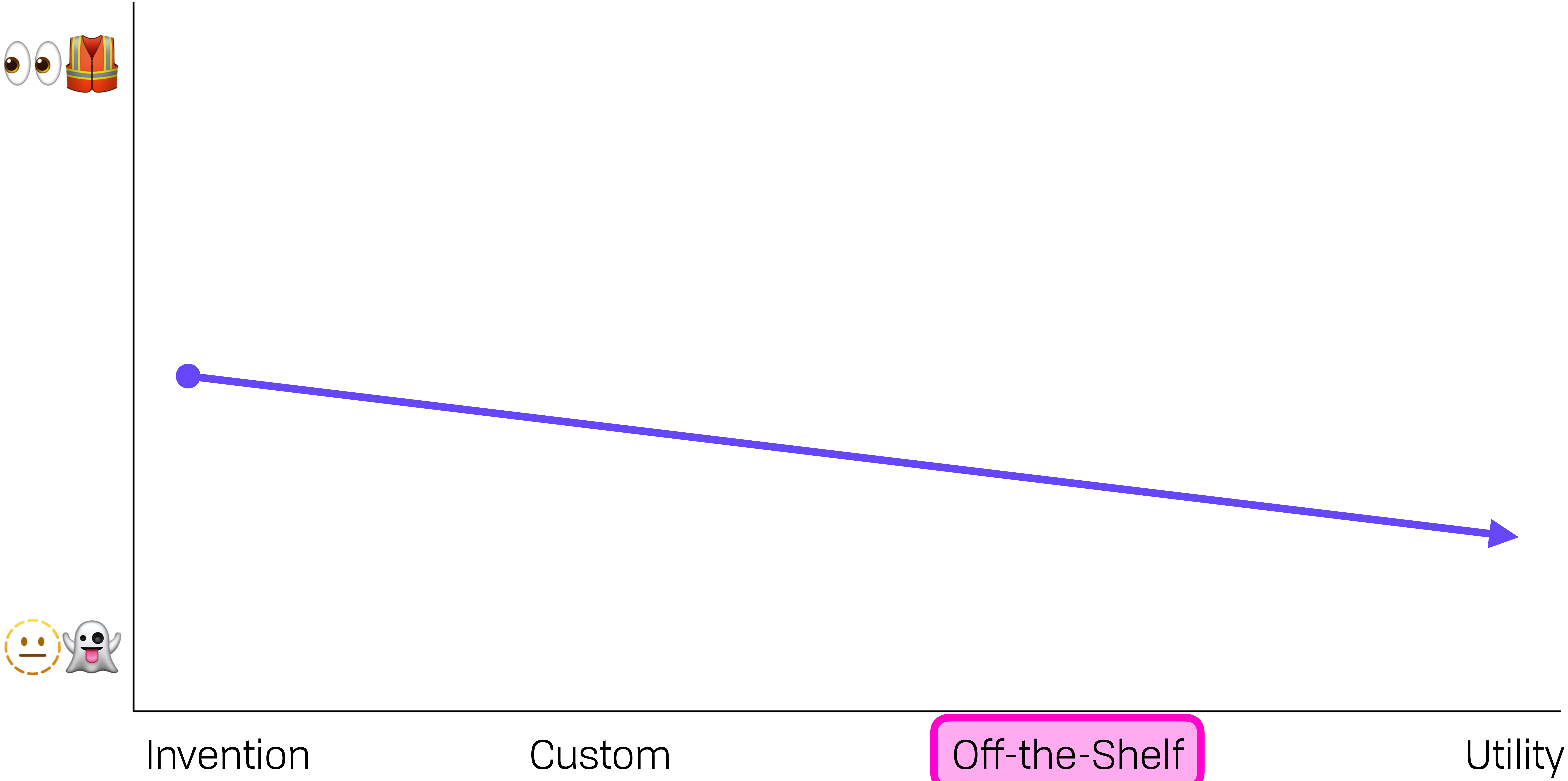
Manifesto

The World Is Changing



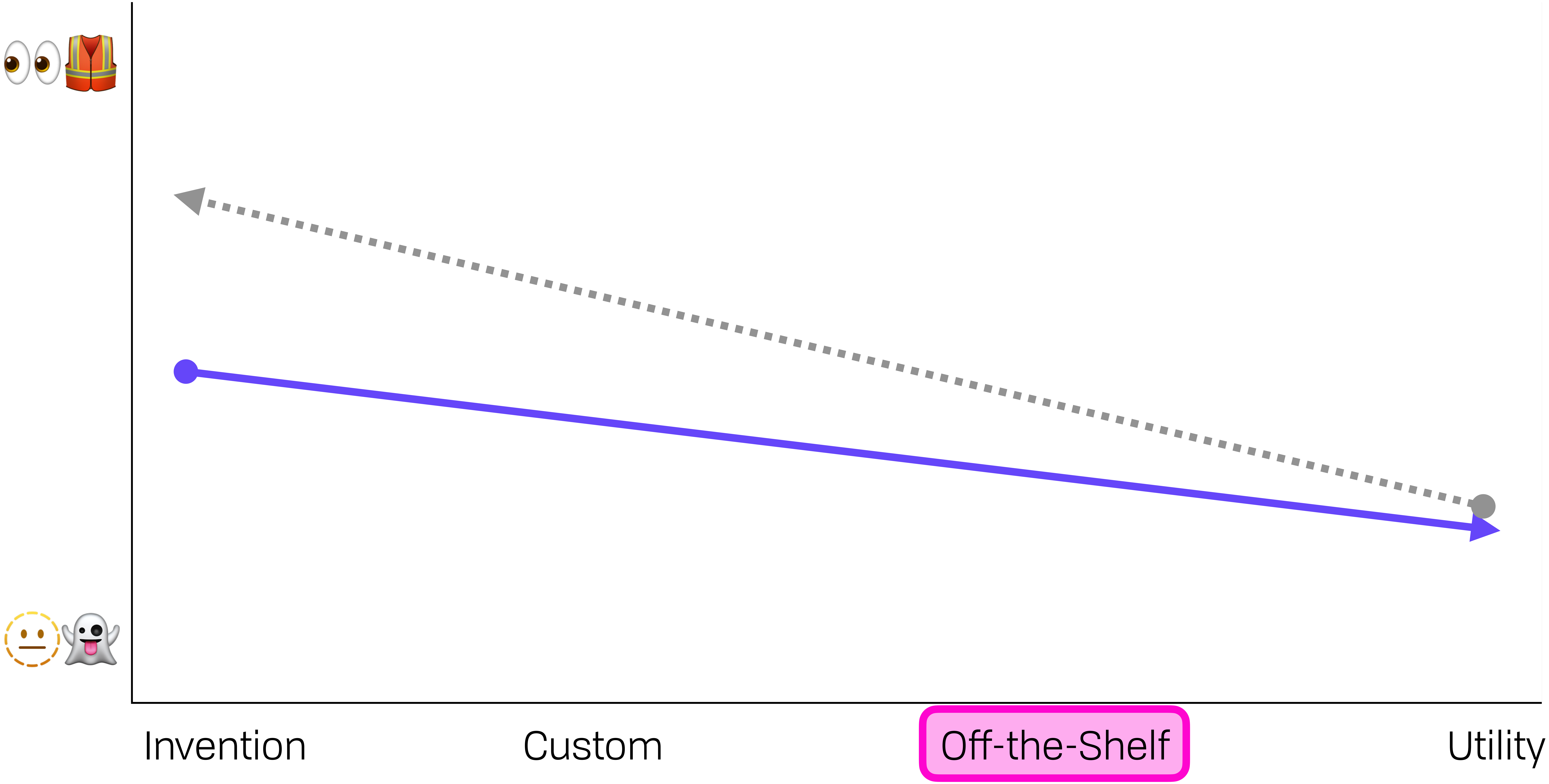
Manifesto ☎️→🔮→❓

The World Is Changing



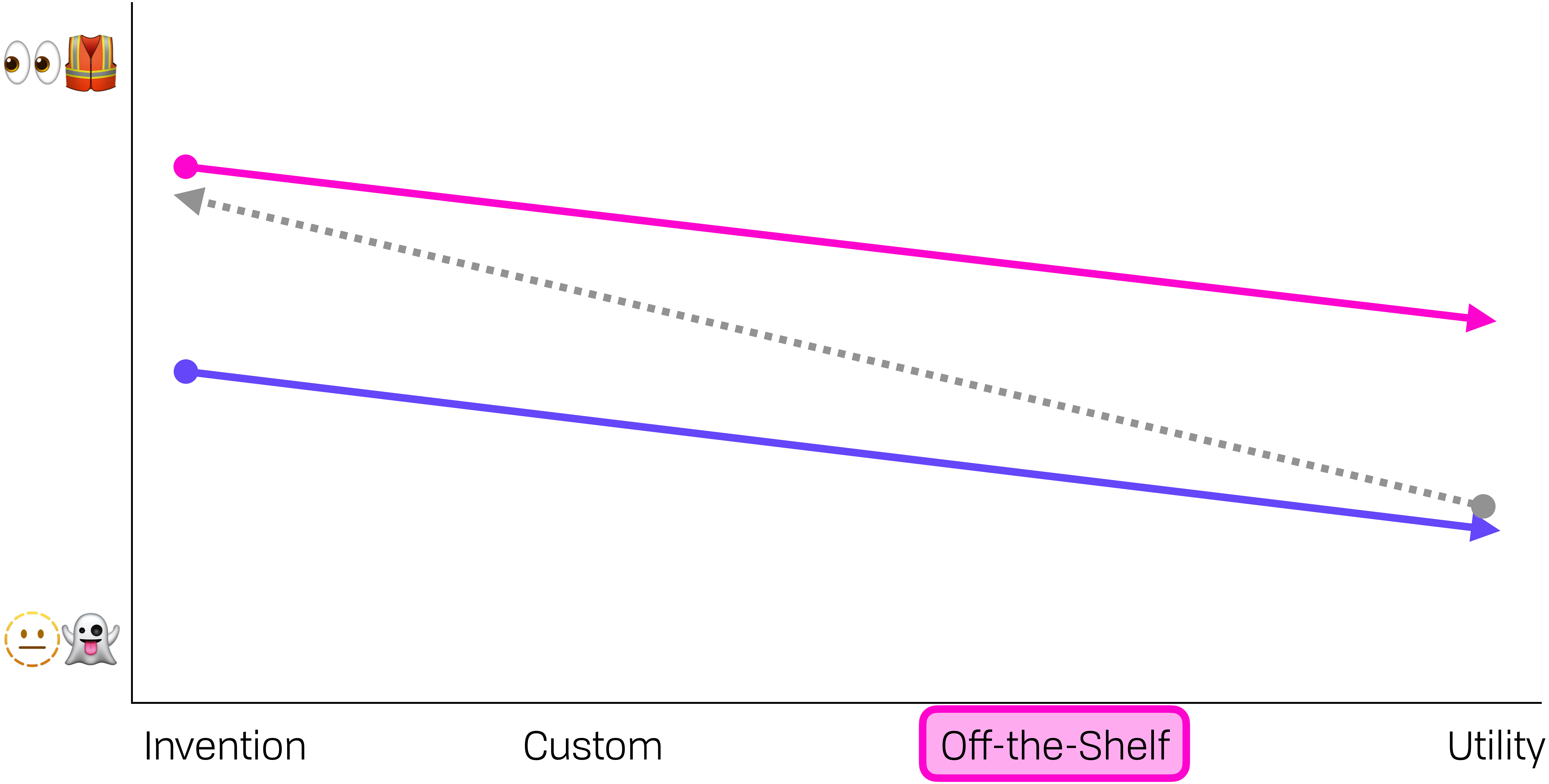
Manifesto ☎️→🧞→❓

The World Is Changing



Manifesto ☎️→🔮→❓

The World Is Changing

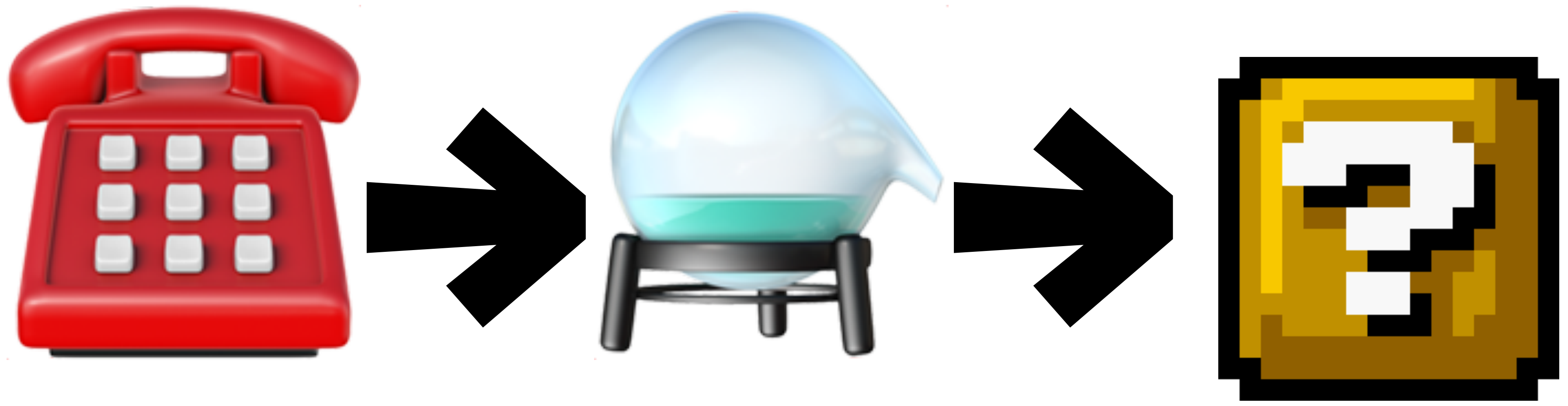


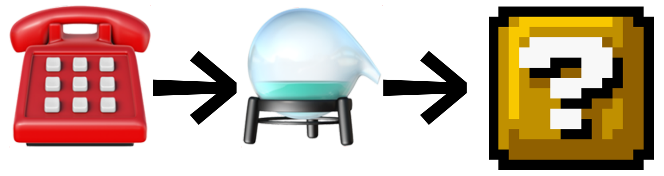
Manifesto ☎️→🔮→❓

A Cambrian Explosion of Approaches!

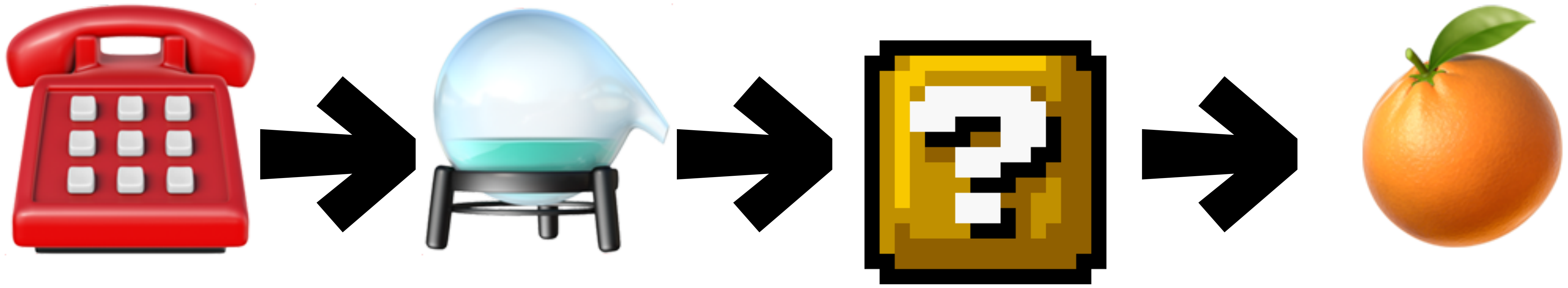
Manifesto ☎️→⚗️→❓

A Cambrian Explosion of Approaches!



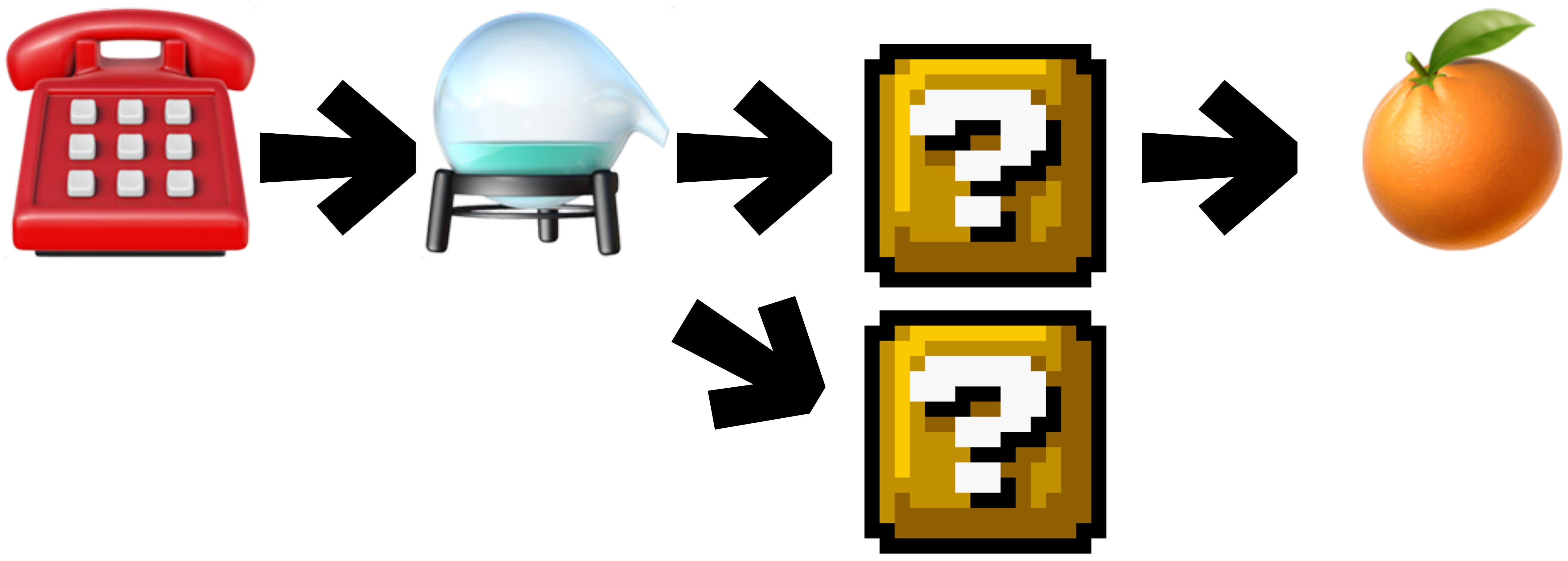
Manifesto 

A Cambrian Explosion of Approaches!



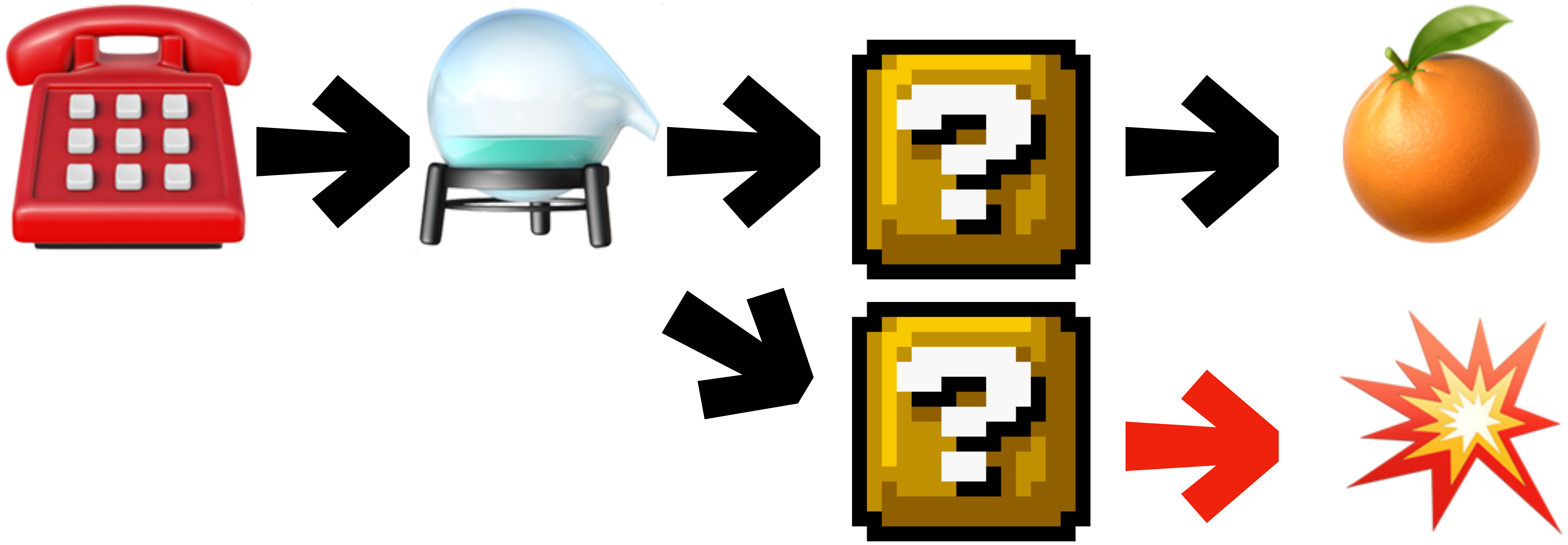
Manifesto ☎️→⚗️→❓

A Cambrian Explosion of Approaches!



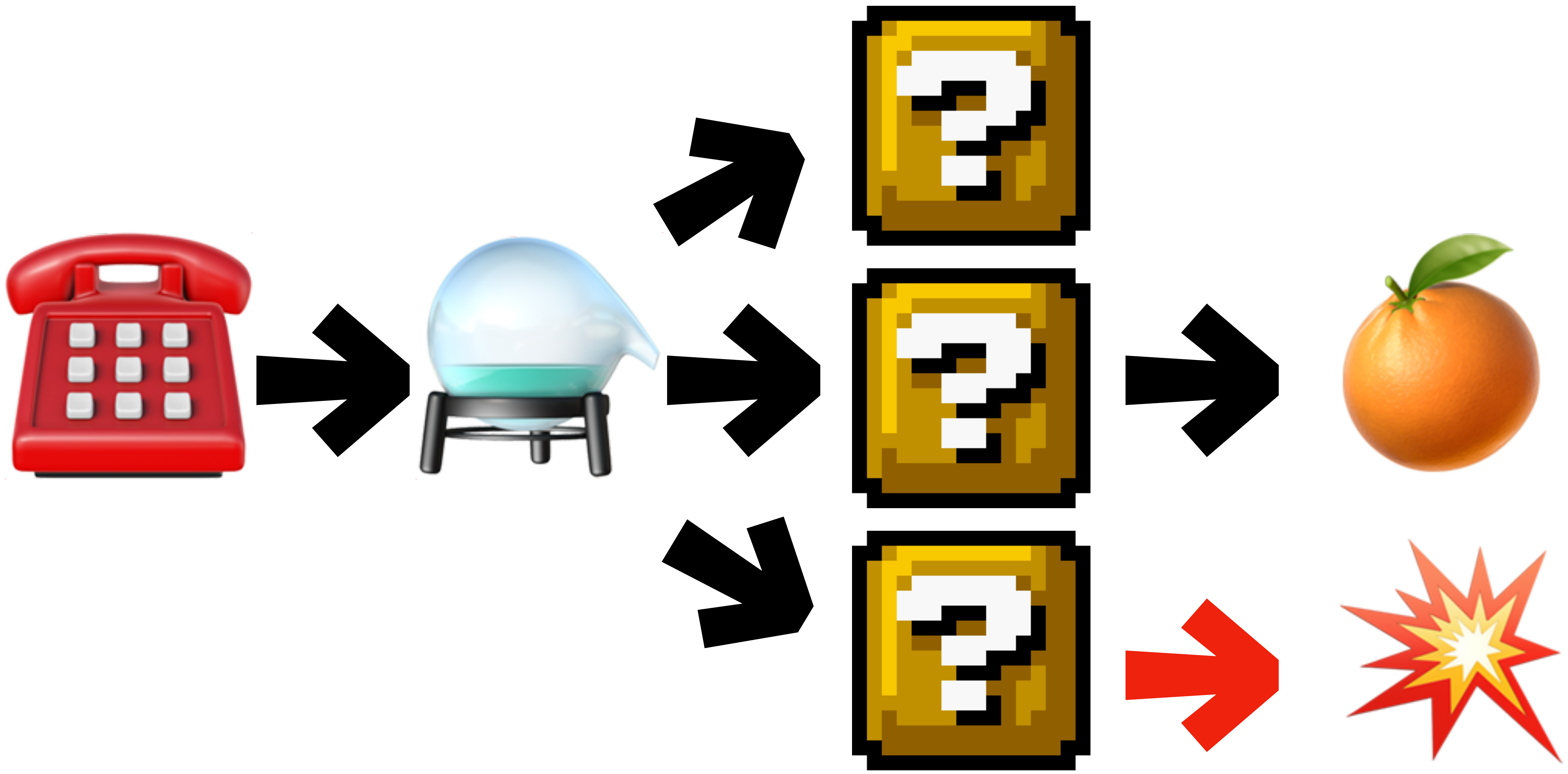
Manifesto ☎️→⚗️→❓

A Cambrian Explosion of Approaches!



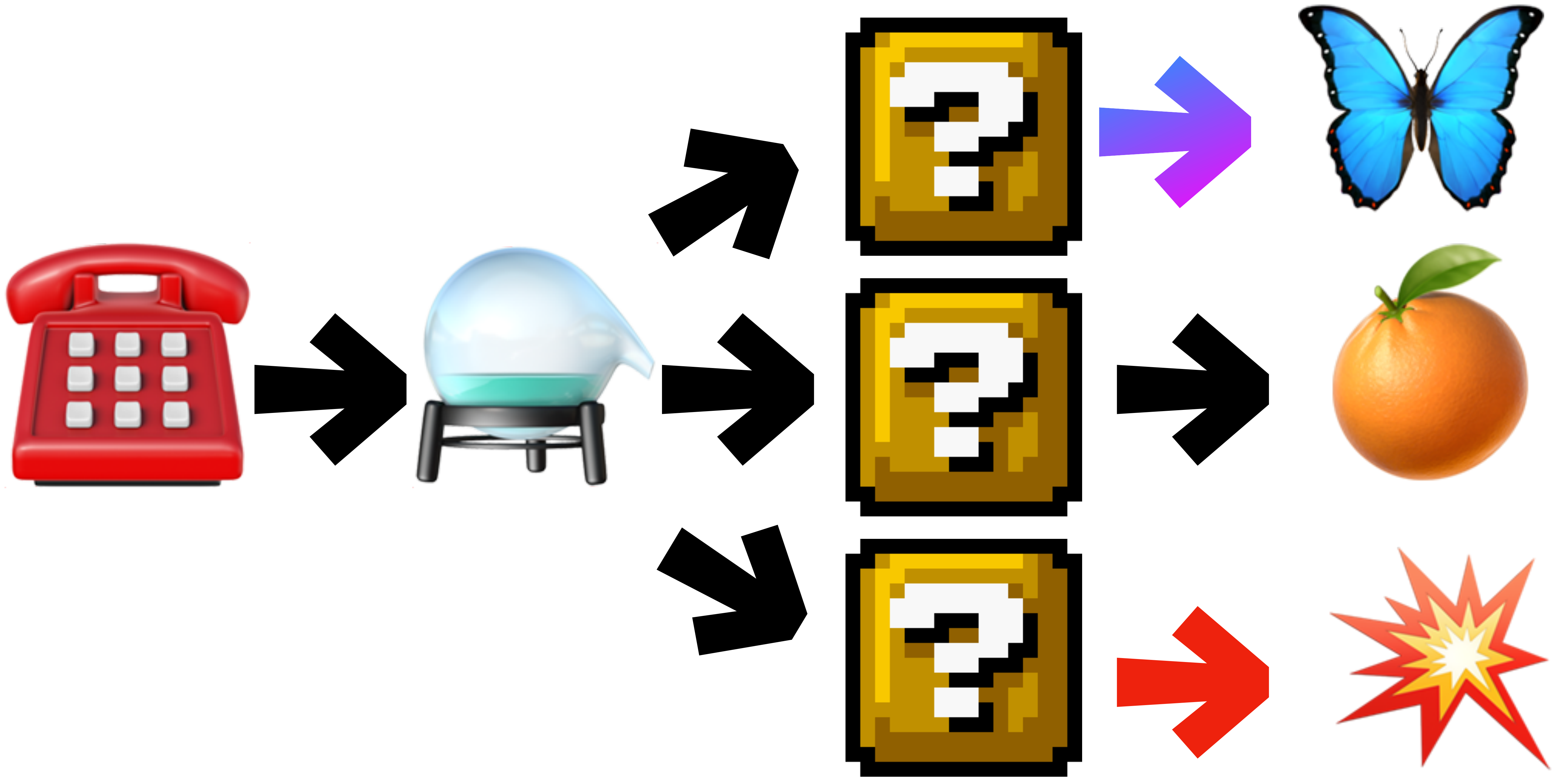
Manifesto ☎️→⚗️→❓

A Cambrian Explosion of Approaches!



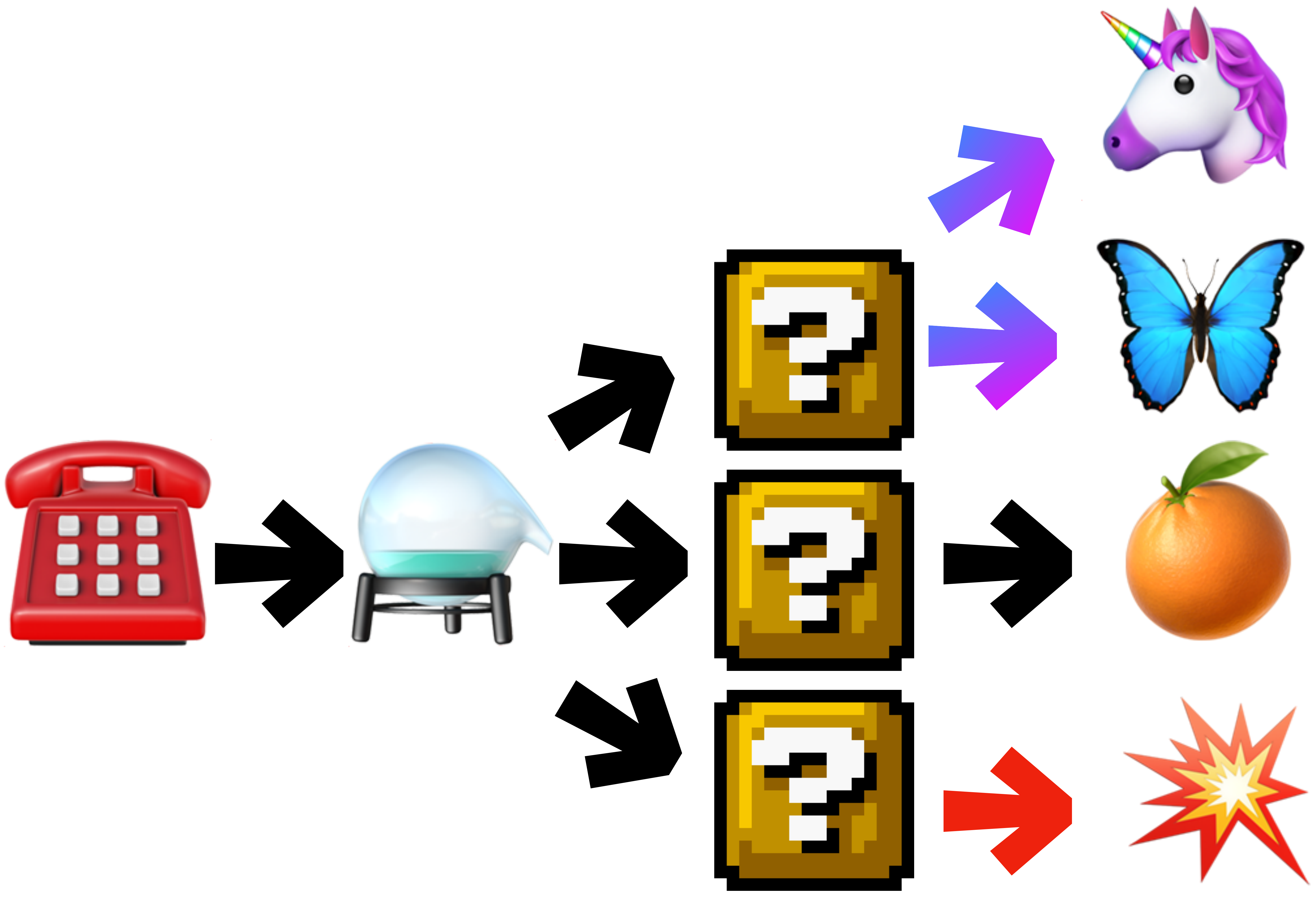
Manifesto ☎️→⚗️→❓

A Cambrian Explosion of Approaches!



Manifesto ☎️→⚗️→❓

A Cambrian Explosion of Approaches!



Manifesto ☎️→🔮→❓

Sources of Inspiration

Manifesto ☎️→🔮→❓

Sources of Inspiration

🚀 Things I've seen ***work in production***

Manifesto ☎️→🫧→❓

Sources of Inspiration

🚀 Things I've seen ***work in production***

🕰️ Ideas from the ***70s & 80s***

Manifesto ☎️→🔮→❓

Sources of Inspiration

- 🚀 Things I've seen ***work in production***
- 🕰️ Ideas from the ***70s & 80s***
- 🧋 ***Functional Pearls***

Manifesto ☎️→🔮→❓

Sources of Inspiration

🚀 Things I've seen ***work in production***

🕰️ Ideas from the ***70s & 80s***

🧋 ***Functional Pearls***

🤓 Programming language ***research***

Manifesto ☎️→🔮→❓

Sources of Inspiration

- 🚀 Things I've seen ***work in production***
- 🕰️ Ideas from the ***70s & 80s***
- 🧋 ***Functional Pearls***
- 🤓 Programming language ***research***
- 🕸️ Distributed ***databases***

Manifesto ☎️→🔮→❓

Sources of Inspiration

- 🚀 Things I've seen ***work in production***
- 🕰️ Ideas from the ***70s & 80s***
- 🧋 ***Functional Pearls***
- 🤓 Programming language ***research***
- 🕸️ Distributed ***databases***
- 🤝 ***Other ecosystems*** (e.g. Scheme, OCaml, Haxl, Racket)

Manifesto ☎️→🔮→❓

Sources of Inspiration

- 🚀 Things I've seen ***work in production***
- 🕰️ Ideas from the ***70s & 80s***
- 🧋 ***Functional Pearls***
- 🤓 Programming language ***research***
- 🕸️ Distributed ***databases***
- 🤝 ***Other ecosystems*** (e.g. Scheme, OCaml, Haxl, Racket)
- 🦹 ***Criminally*** underused Elixir features

Manifesto ☎️→🔮→❓

Three ***Stupidly Powerful*** *Concepts*

Manifesto ☎️→🔮→❓

Three ***Stupidly Powerful*** *Concepts*

eDSL

Manifesto ☎️→🔮→❓

Three **Stupidly Powerful** Concepts

eDSL

- ***Code as Data***
- ***Managed Effects***
- ***Active Metadata***

Manifesto ☎️→⚗️→❓

Three **Stupidly Powerful** Concepts

eDSL

- ***Code as Data***
  - Expressivity
  - Modular Languages
  - Denotational Design
- ***Managed Effects***
- ***Active Metadata***

Manifesto ☎️→⚗️→

Three **Stupidly Powerful** Concepts

eDSL

- ***Code as Data***
  - Expressivity
  - Modular Languages
    - *Derivative DSLs*
  - Denotational Design
- ***Managed Effects***
- ***Active Metadata***

Manifesto ☎️→⚗️→❓

Three **Stupidly Powerful** Concepts

eDSL

- ***Code as Data***
  - Expressivity
  - Modular Languages
    - *Derivative DSLs*
  - Denotational Design
- ***Managed Effects***
  - Runners
  - Implicit Concurrency
  - Model Testing
- ***Active Metadata***

Manifesto ☎️→⚗️→❓

Three **Stupidly Powerful** Concepts

eDSL

- ***Code as Data***
  - Expressivity
  - Modular Languages
    - *Derivative DSLs*
  - Denotational Design
- ***Managed Effects***
  - Runners
  - Implicit Concurrency
    - *Structural*
    - *Optimistic*
  - Model Testing
- ***Active Metadata***

Manifesto ☎️→🔮→❓

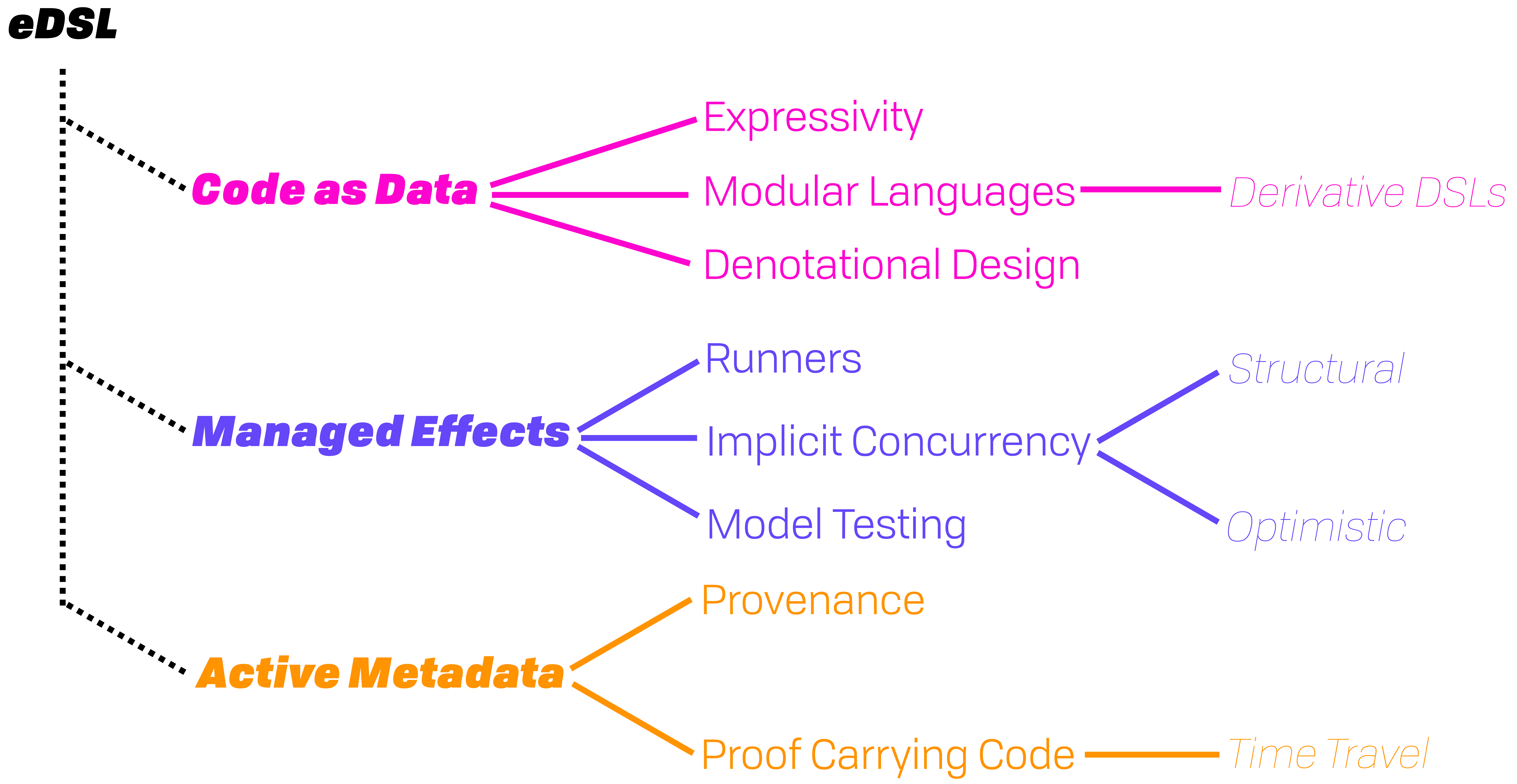
Three **Stupidly Powerful** Concepts

eDSL

- ***Code as Data***
  - Expressivity
  - Modular Languages
    - *Derivative DSLs*
  - Denotational Design
- ***Managed Effects***
  - Runners
  - Implicit Concurrency
    - *Structural*
    - *Optimistic*
  - Model Testing
- ***Active Metadata***
  - Provenance
  - Proof Carrying Code

Manifesto ☎️→🔮→❓

Three **Stupidly Powerful** Concepts

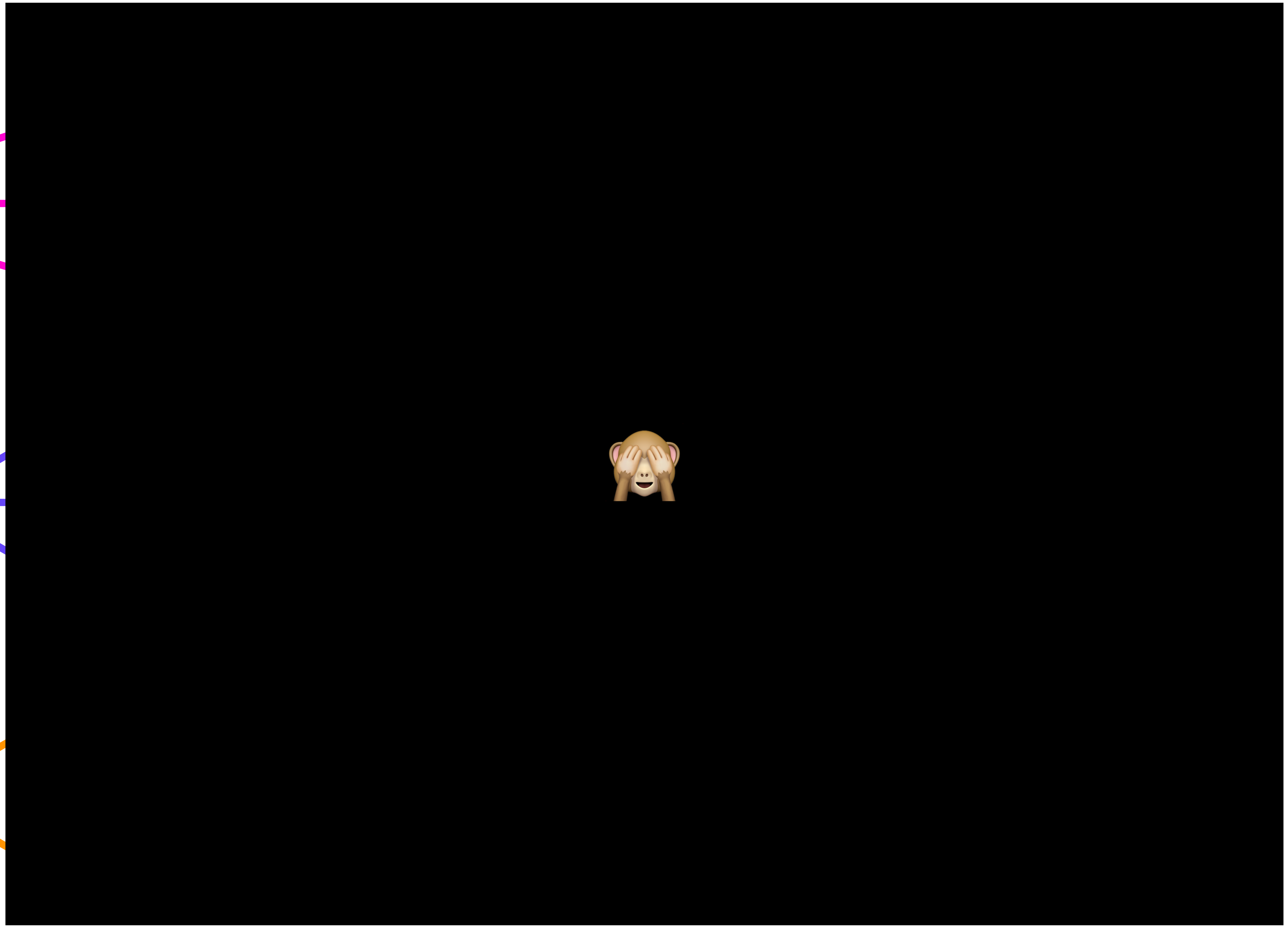


Manifesto ☎️→🔮→❓

Three ***Stupidly Powerful*** *Concepts*

eDSL

- ***Code as Data***
- ***Managed Effects***
- ***Active Metadata***



Modular Semantics

Millions of Tiny Languages

Modular Semantics

Millions of Tiny Languages

Millions of Tiny Languages ⚛

We really ***don't want to build*** a programming language from scratch[...], ***let's inherit infrastructure*** from some other language

– Paul Hudak, Building Domain Specific Embedded Languages

We really ***don't want to build*** a programming language from scratch[…], ***let's inherit infrastructure*** from some other language

– Paul Hudak, Building Domain Specific Embedded Languages

"Some other language" 😏

All the Way Down

All the Way Down

All the Way Down

Binary

Physics

Mathematics

Millions of Tiny Languages ⚛

All the Way Down

Binary

Physics

Mathematics

Millions of Tiny Languages ⚛

All the Way Down

- Kernel syscalls
- x86 / ARM / RISC-V
- Binary
- Physics
- Mathematics

Millions of Tiny Languages ⚛

All the Way Down

Elixir AST

BEAM bytecode

Kernel syscalls

x86 / ARM / RISC-V

Binary

Physics

Mathematics

Millions of Tiny Languages ⚛

All the Way Down

- Application Domain
- Library / Framework
- Elixir Language
- Elixir AST
- BEAM bytecode
- Kernel syscalls
- x86 / ARM / RISC-V
- Binary
- Physics
- Mathematics

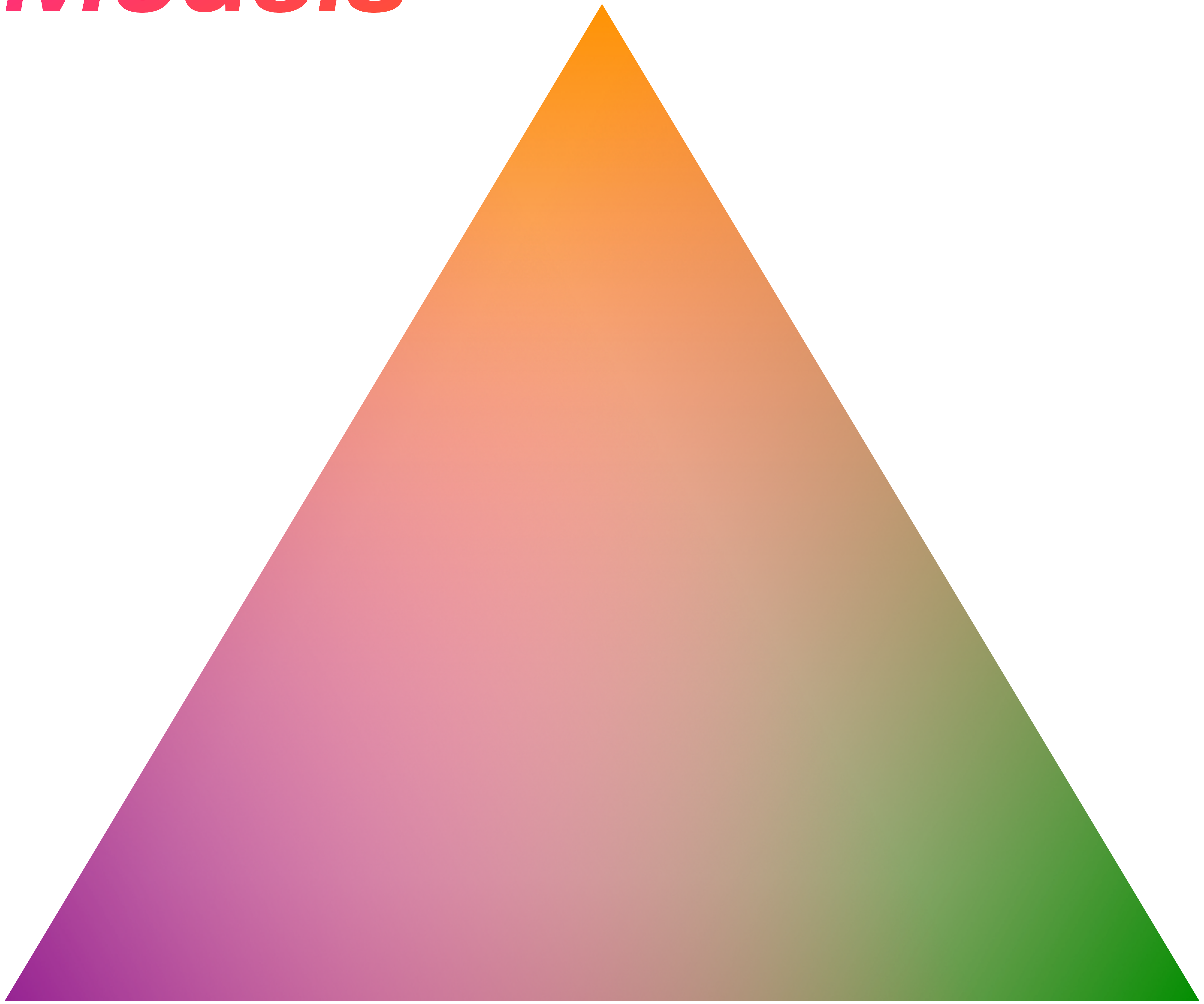
Millions of Tiny Languages ⚛

All the Way Down

- Natural Language
- GUI Metaphor
- Application Domain
- Library / Framework
- Elixir Language
- Elixir AST
- BEAM bytecode
- Kernel syscalls
- x86 / ARM / RISC-V
- Binary
- Physics
- Mathematics

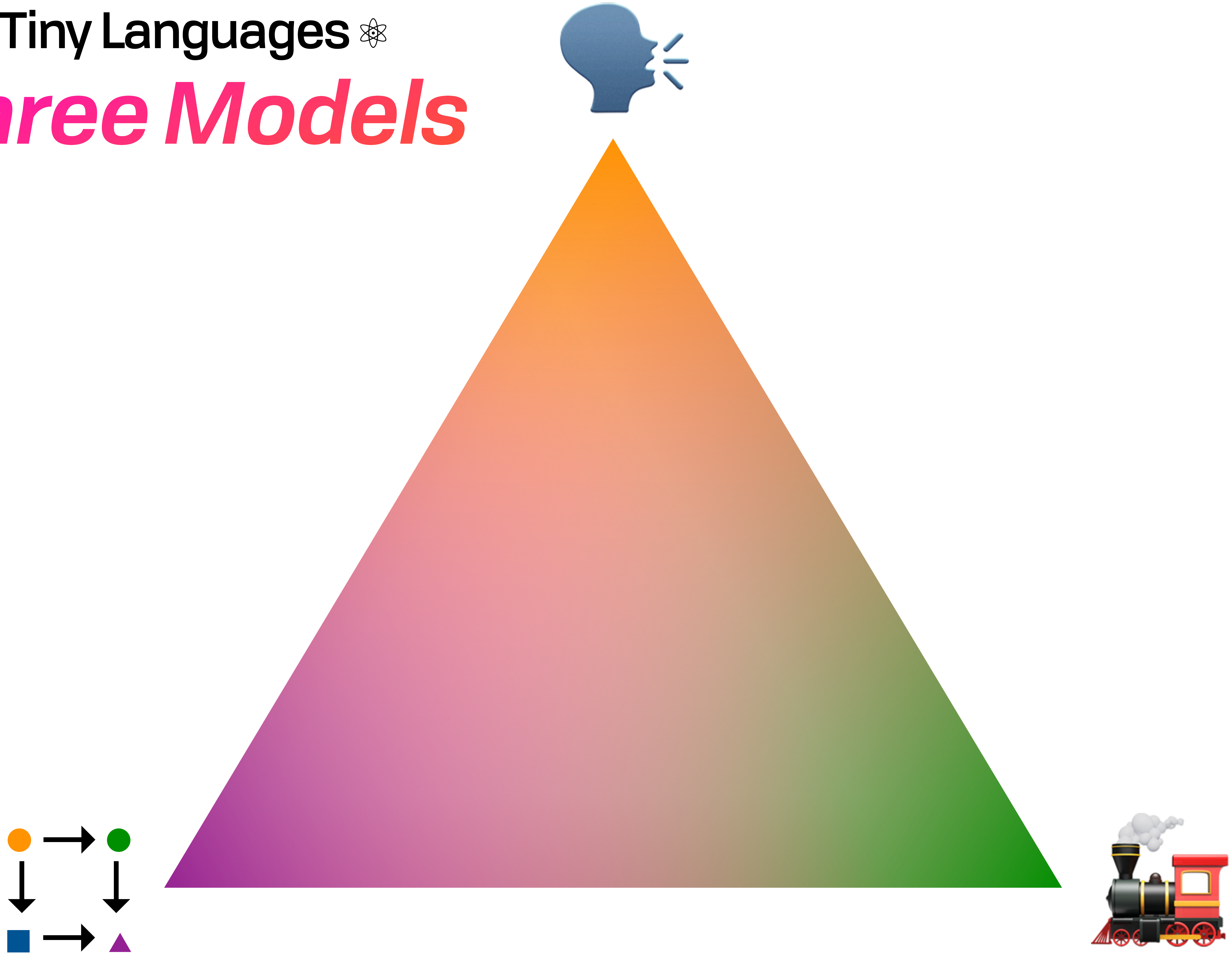
Big Three Models

Big Three Models



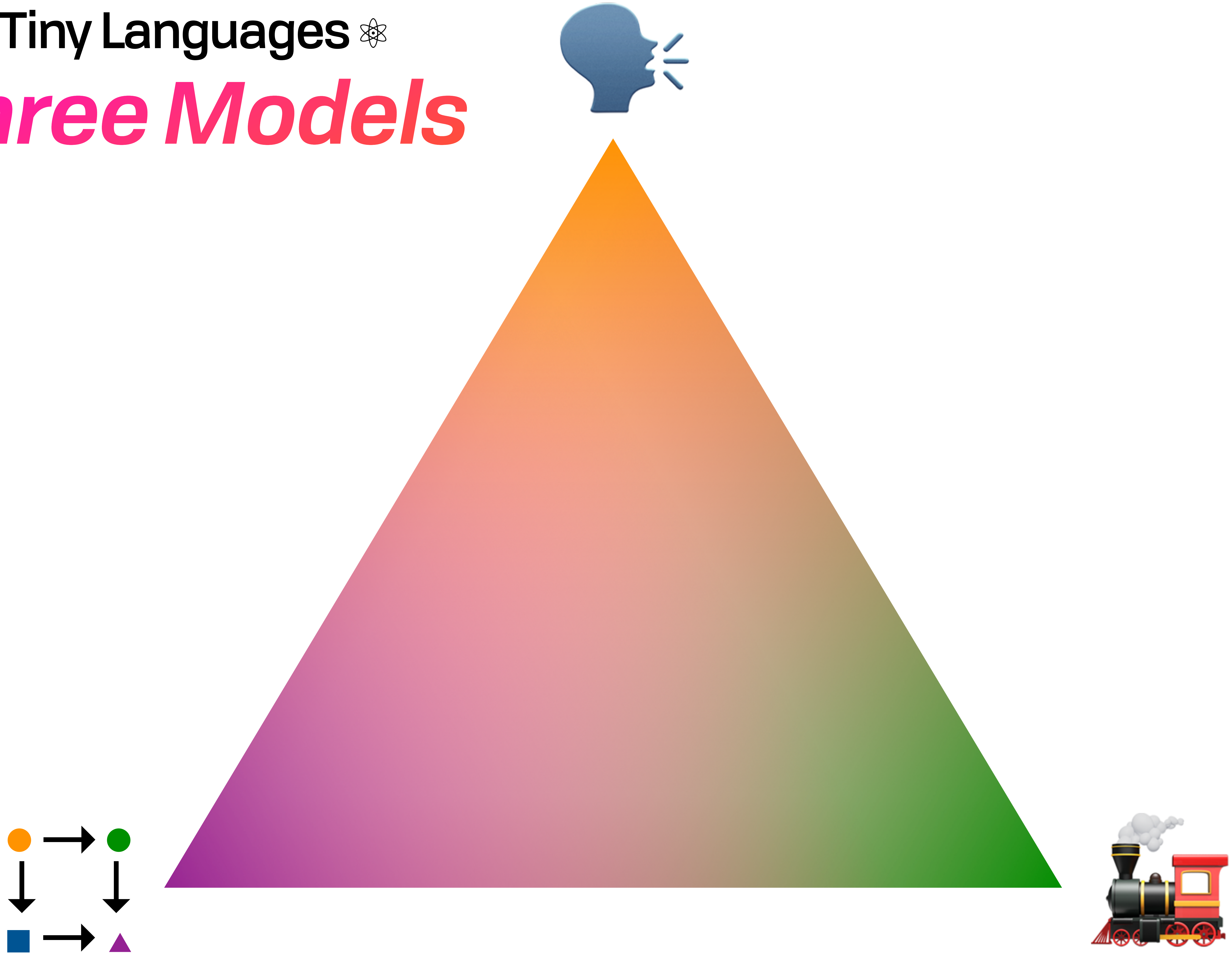
Millions of Tiny Languages ⚛

Big Three Models



Millions of Tiny Languages ⚛

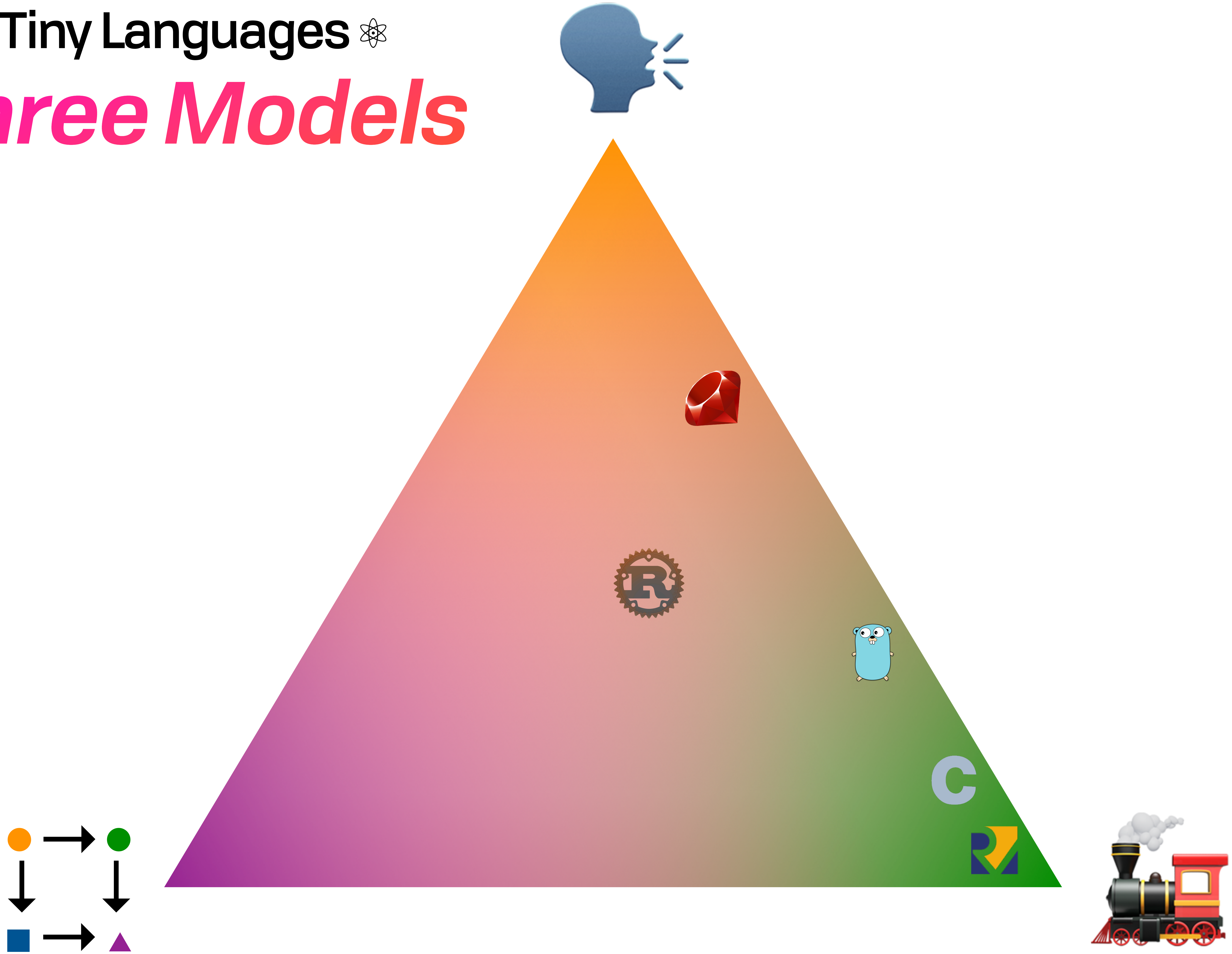
Big Three Models



Dynamic
Operational
Mechanical

Millions of Tiny Languages

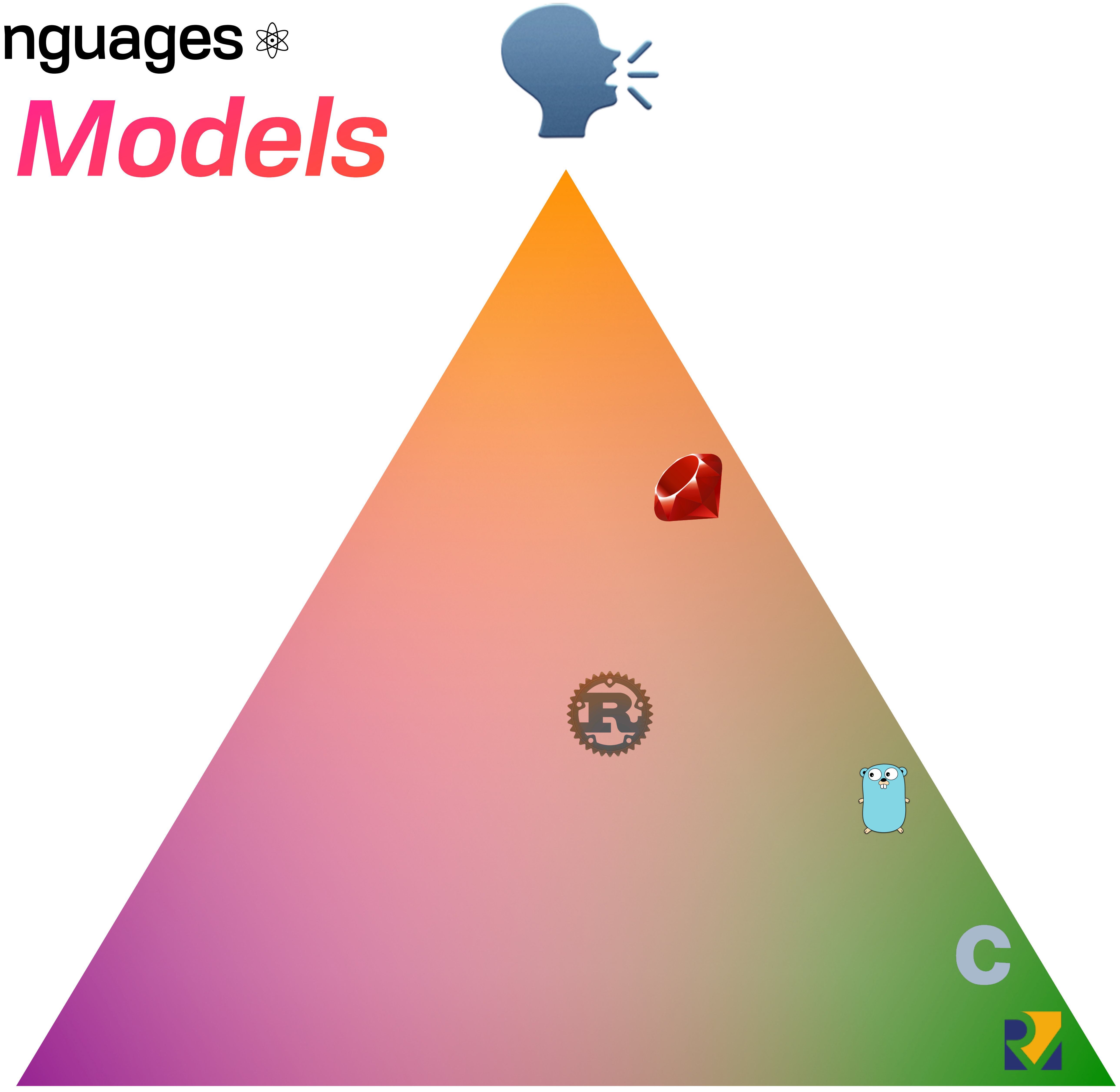
Big Three Models



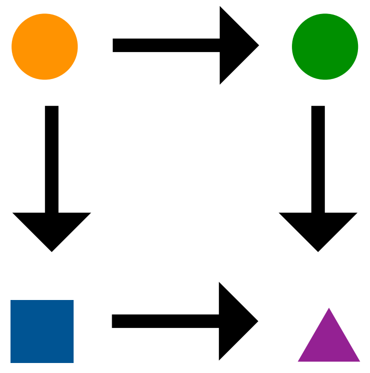
Dynamic
Operational
Mechanical

Millions of Tiny Languages ⚛

Big Three Models



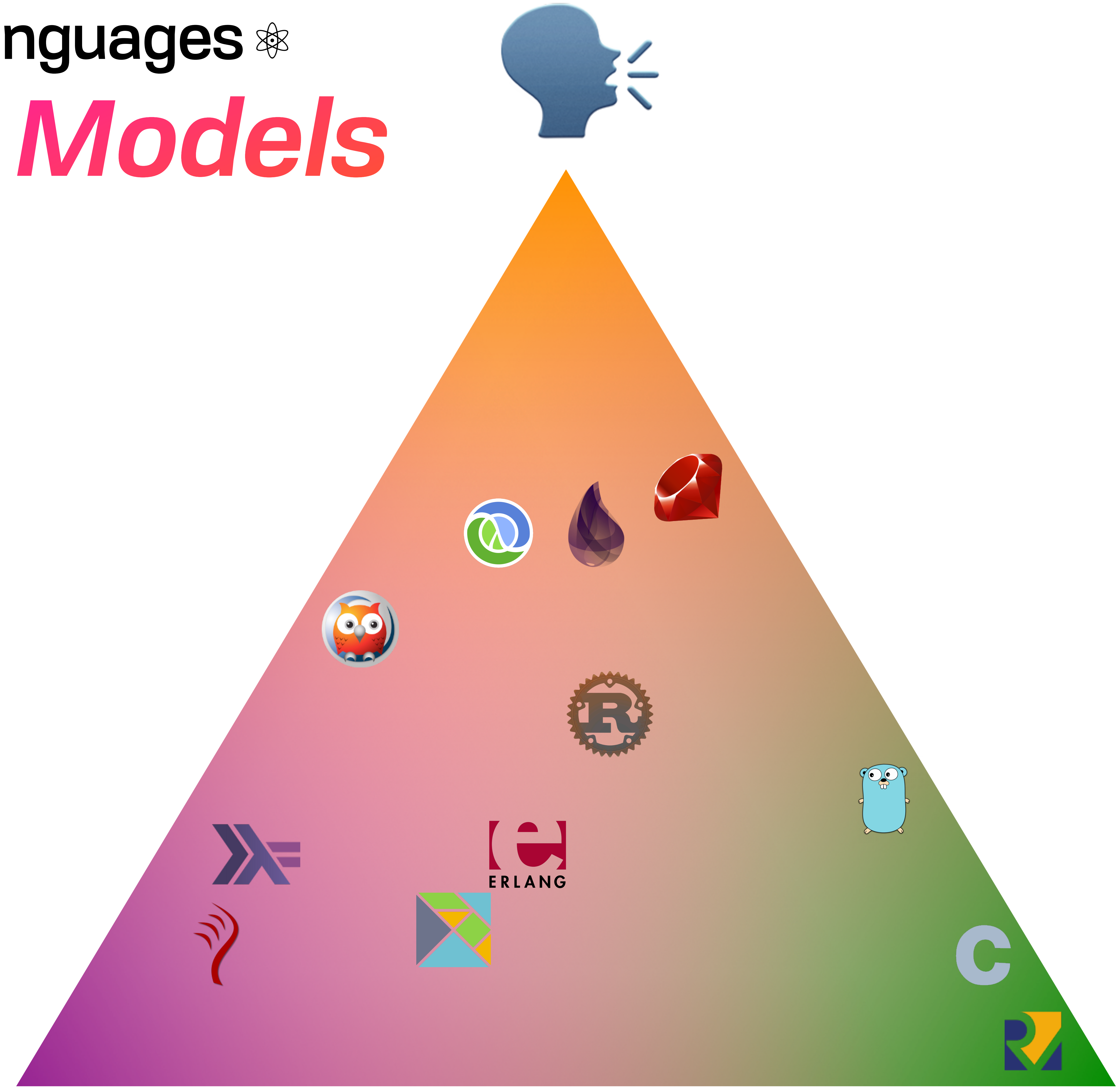
Universal
Denotational
Mathematical



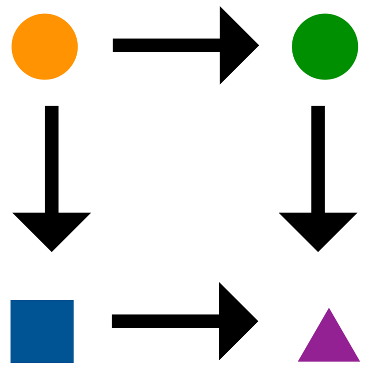
Dynamic
Operational
Mechanical

Millions of Tiny Languages

Big Three Models



Universal
Denotational
Mathematical



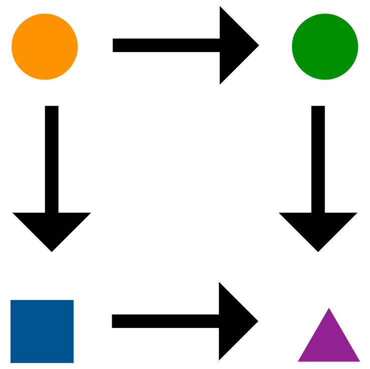
Dynamic
Operational
Mechanical

Millions of Tiny Languages ⚛

Big Three Models

Symbolist
Semiotic
Natural Language

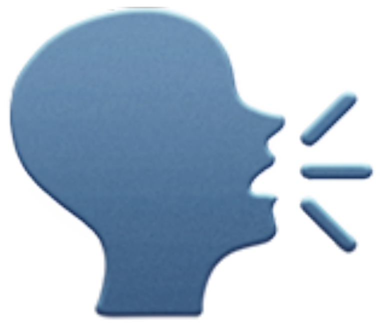
Universal
Denotational
Mathematical



Dynamic
Operational
Mechanical

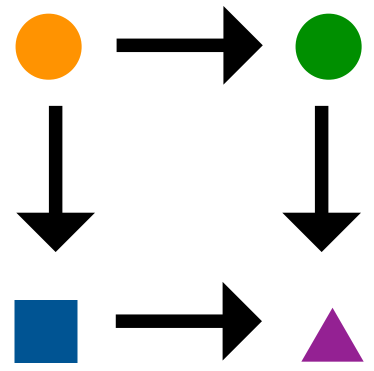
Millions of Tiny Languages ⚛

Big Three Models



Symbolist
Semiotic
Natural Language

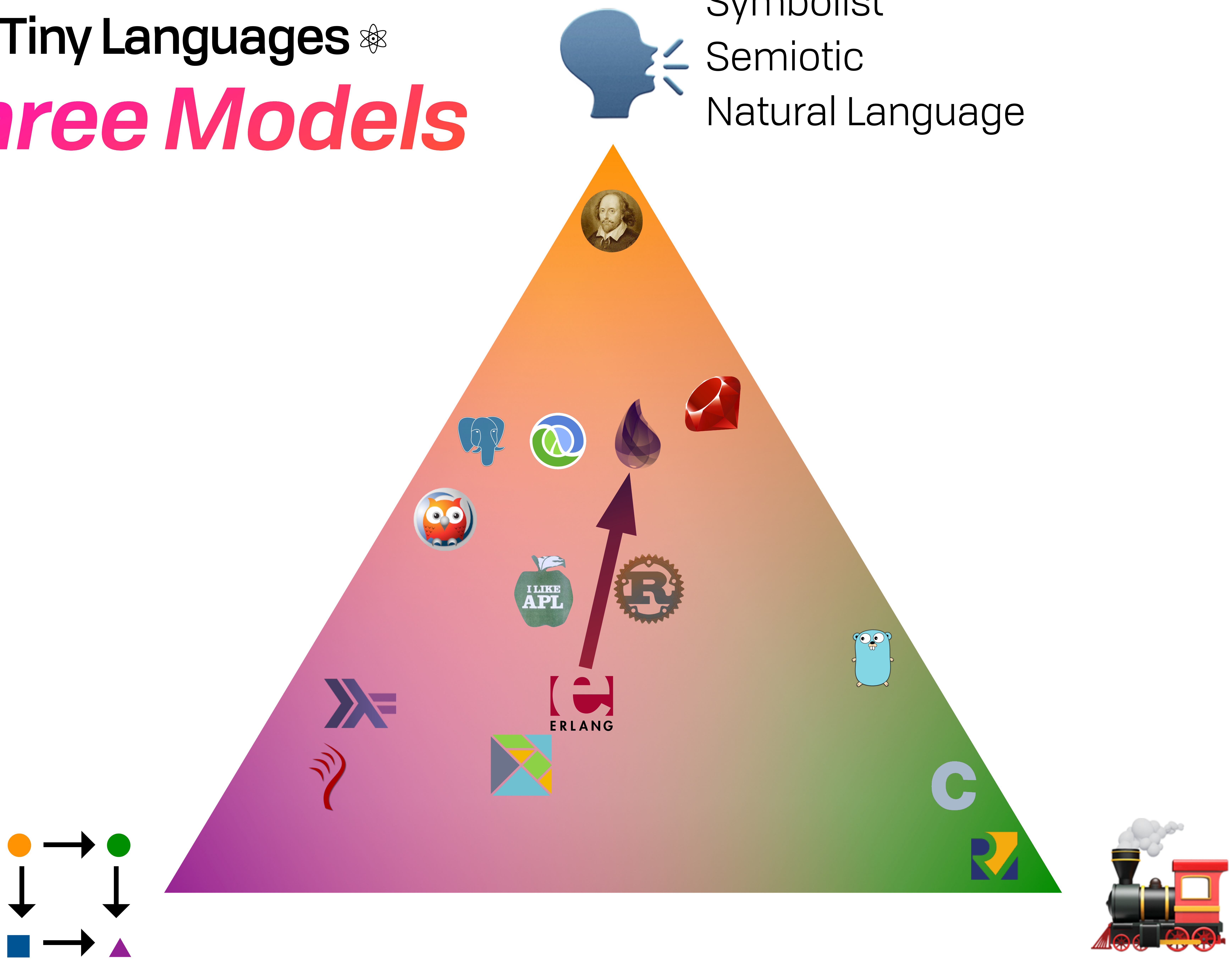
Universal
Denotational
Mathematical



Dynamic
Operational
Mechanical

Millions of Tiny Languages ⚛

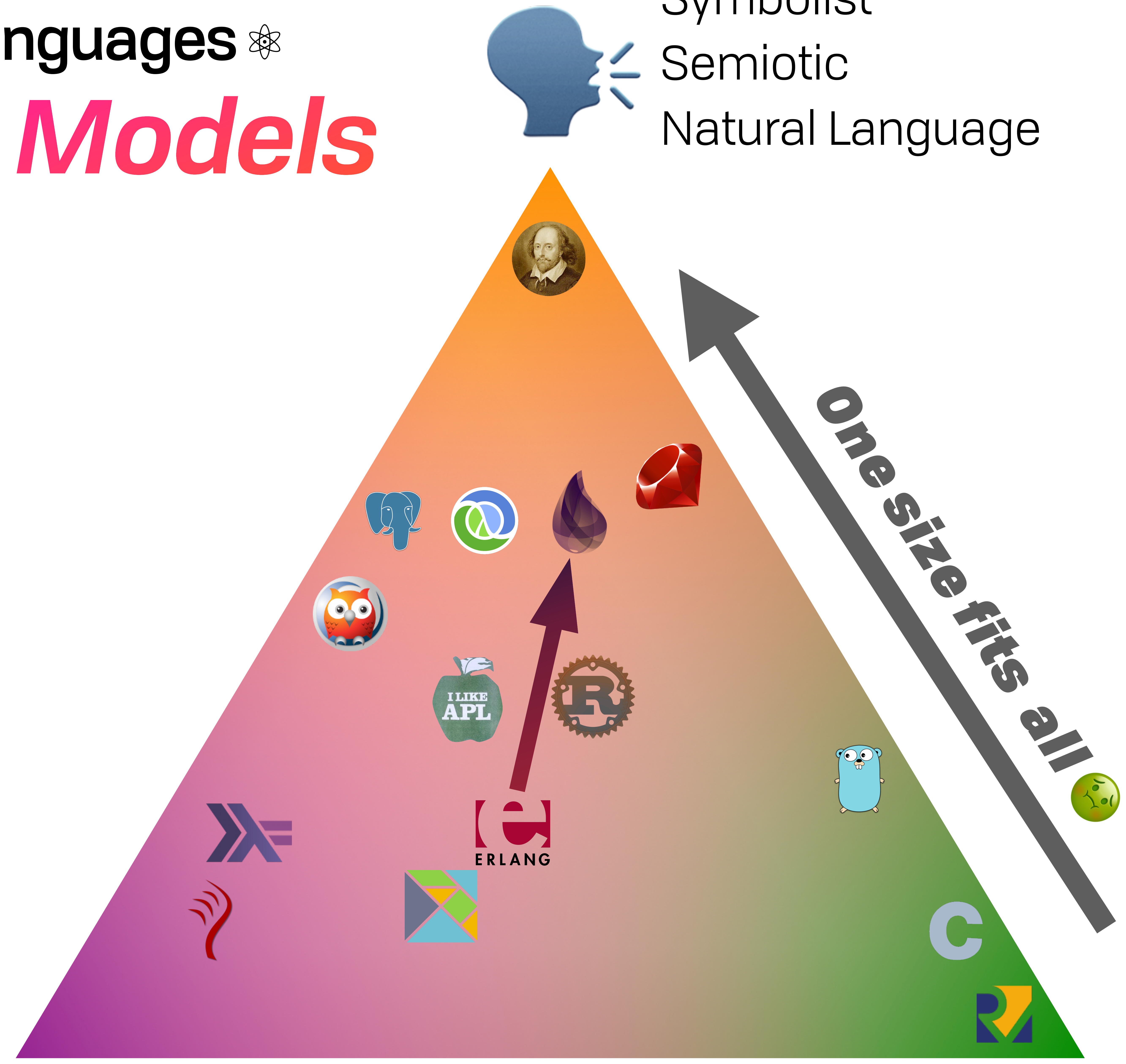
Big Three Models



Millions of Tiny Languages ⚛

Big Three Models

Symbolist
Semiotic
Natural Language



Universal
Denotational
Mathematical

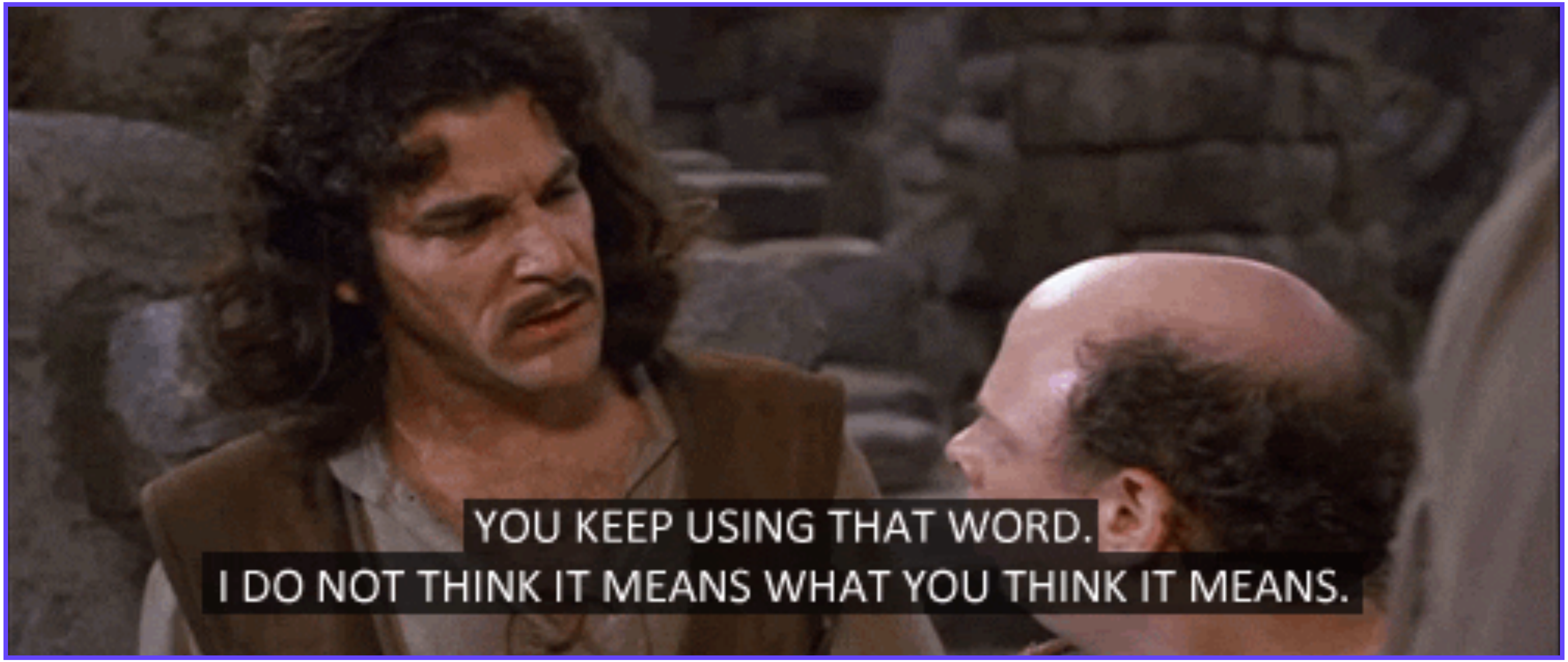
Dynamic
Operational
Mechanical

Millions of Tiny Languages ⚛

Big Three Models

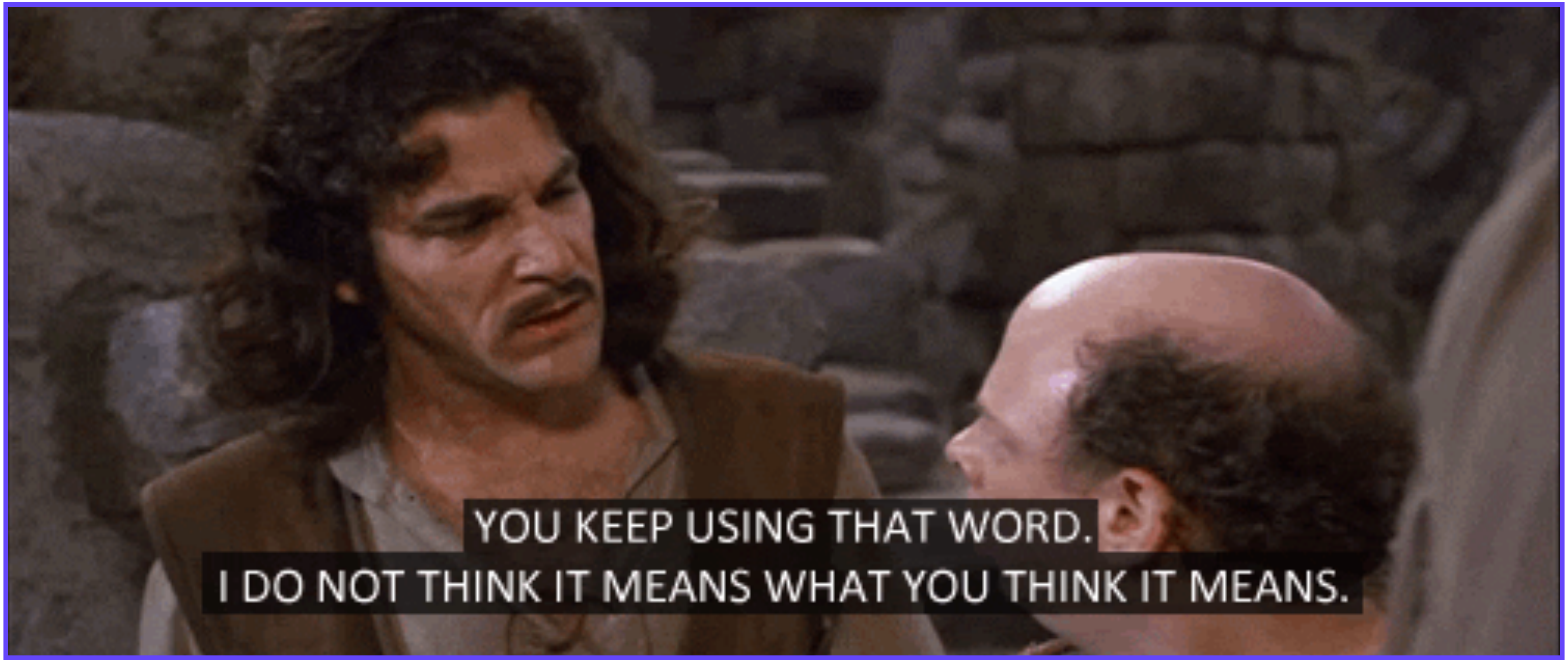
Manifesto ☎️→🔮→❓

"Expressive" 🔥⚔️🐉🙈



Manifesto ☎️→🔮→❓

"Expressive" 🔥⚔️🐉🙈



Tower of (Expressive) Power

Tower of (Expressive) Power

```
5 + 10
5 - 10
5 * 10
5 ^ 10
mod 5 10
```

Tower of (Expressive) Power

```
5 + 10
5 - 10
5 * 10
5 ^ 10
mod 5 10
```

```
x = 5
y = 10
```

Tower of (Expressive) Power

```
5 + 10
5 - 10
5 * 10
5 ^ 10
mod 5 10
```

```
x = 5
y = 10
```

```
x = 5
y = 10
x = x + y
```

Tower of (Expressive) Power

```
5 + 10
5 - 10
5 * 10
5 ^ 10
mod 5 10
```

```
x = 5
y = 10
```

```
x = 5
y = 10
x = x + y
```

```
while(x == false) {
  // ...
}
```

Millions of Tiny Languages ⚛

Tower of (Expressive) Power

```
5 + 10
5 - 10
5 * 10
5 ^ 10
mod 5 10
```

```
x = 5
y = 10
```

```
x = 5
y = 10
x = x + y
```

```
def forever(action) do
  action
  forever(action)
end
```

```
while(x == false) {
  // ...
}
```

Millions of Tiny Languages

Tower of (Expressive) Power

```
5 + 10
5 - 10
5 * 10
5 ^ 10
mod 5 10
```

```
x = 5
y = 10
```

```
x = 5
y = 10
x = x + y
```

```
def forever(action) do
  action
  forever(action)
end
```

```
while(x == false) {
  // ...
}
```

```
def evens([]), do: []
def evens([x | xs]), do: [x | odds(xs)]

def odds ([]), do: []
def odds([_ | xs]), do: evens(xs)
```

Millions of Tiny Languages

Tower of (Expressive) Power

```
5 + 10
5 - 10
5 * 10
5 ^ 10
mod 5 10
```

```
x = 5
y = 10
```

```
x = 5
y = 10
x = x + y
```

```
while(x == false) {
  // ...
}
```

```
def forever(action) do
  action
  forever(action)
end
```

```
def evens([]), do: []
def evens([x | xs]), do: [x | odds(xs)]

def odds ([]), do: []
def odds([_ | xs]), do: evens(xs)
```

```
iex(7)> evens([1,2,3,4,5])
[1, 3, 5]
iex(8)> odds([1,2,3,4,5])
[2, 4]
```

Practical Bounds

Practical Bounds

Can a language be ***too expressive?***

Millions of Tiny Languages

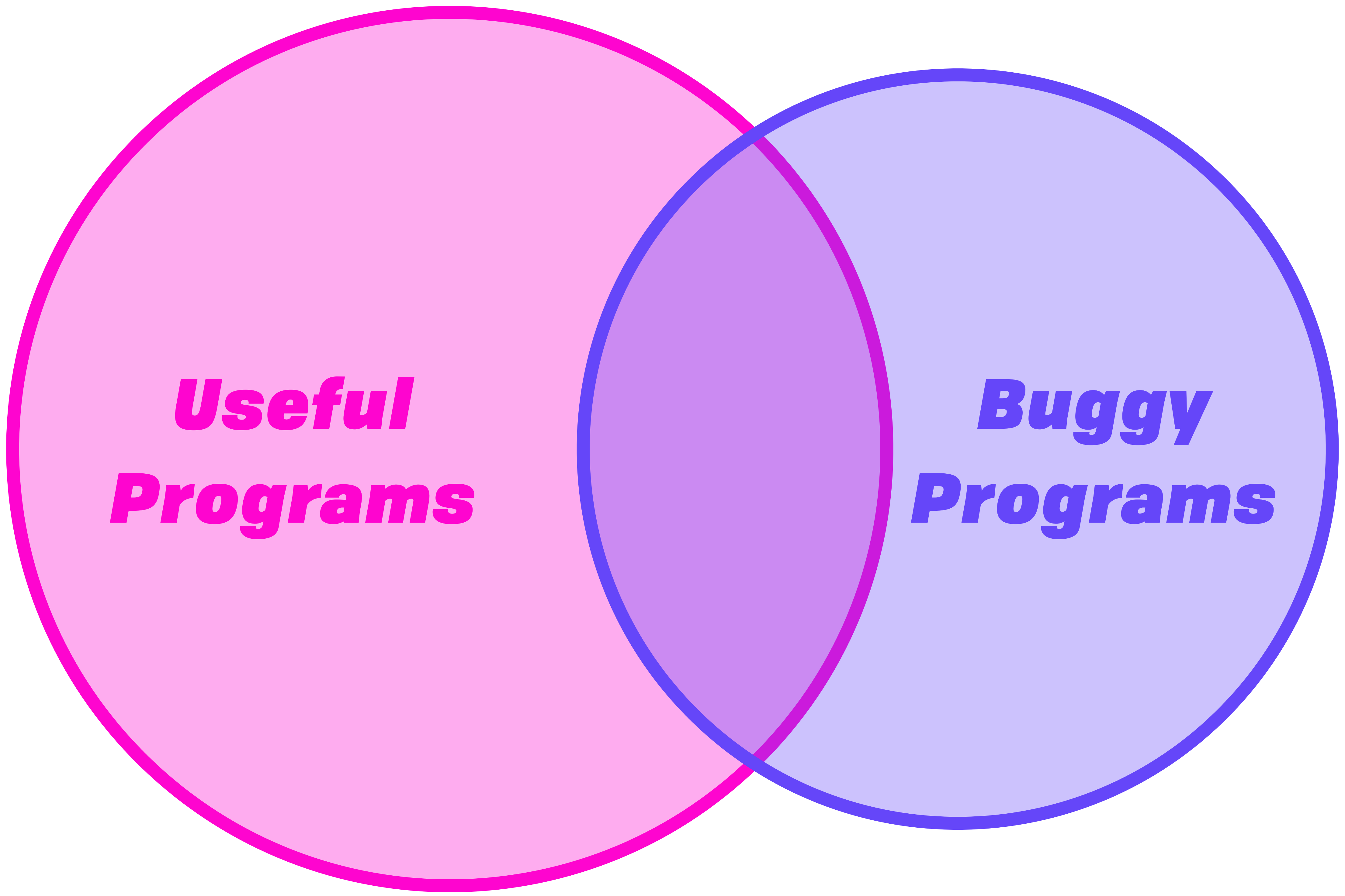
Practical Bounds

Can a language be ***too expressive?***

Millions of Tiny Languages

Practical Bounds

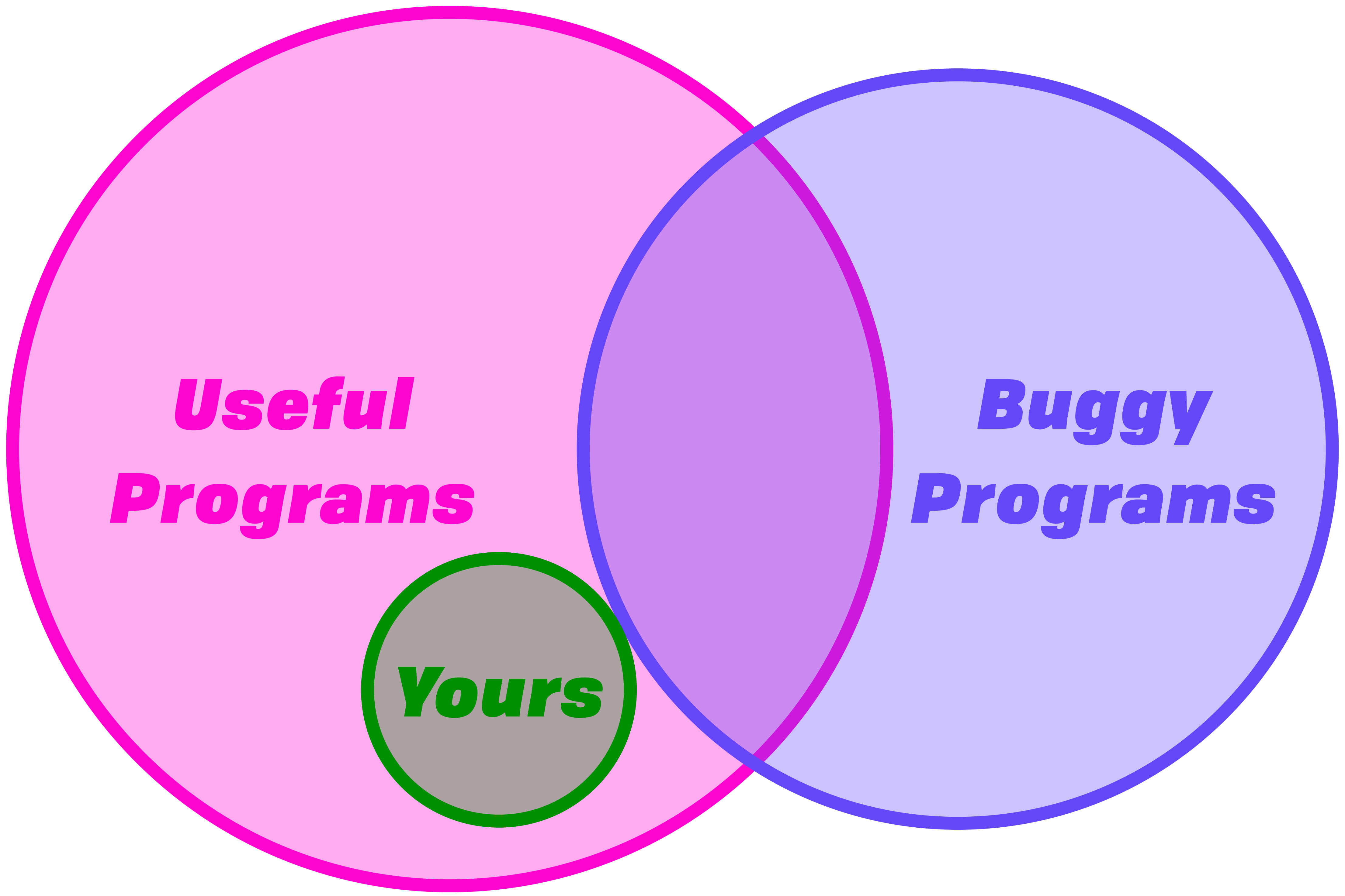
Can a language be ***too expressive?***



Millions of Tiny Languages ⚛

Practical Bounds

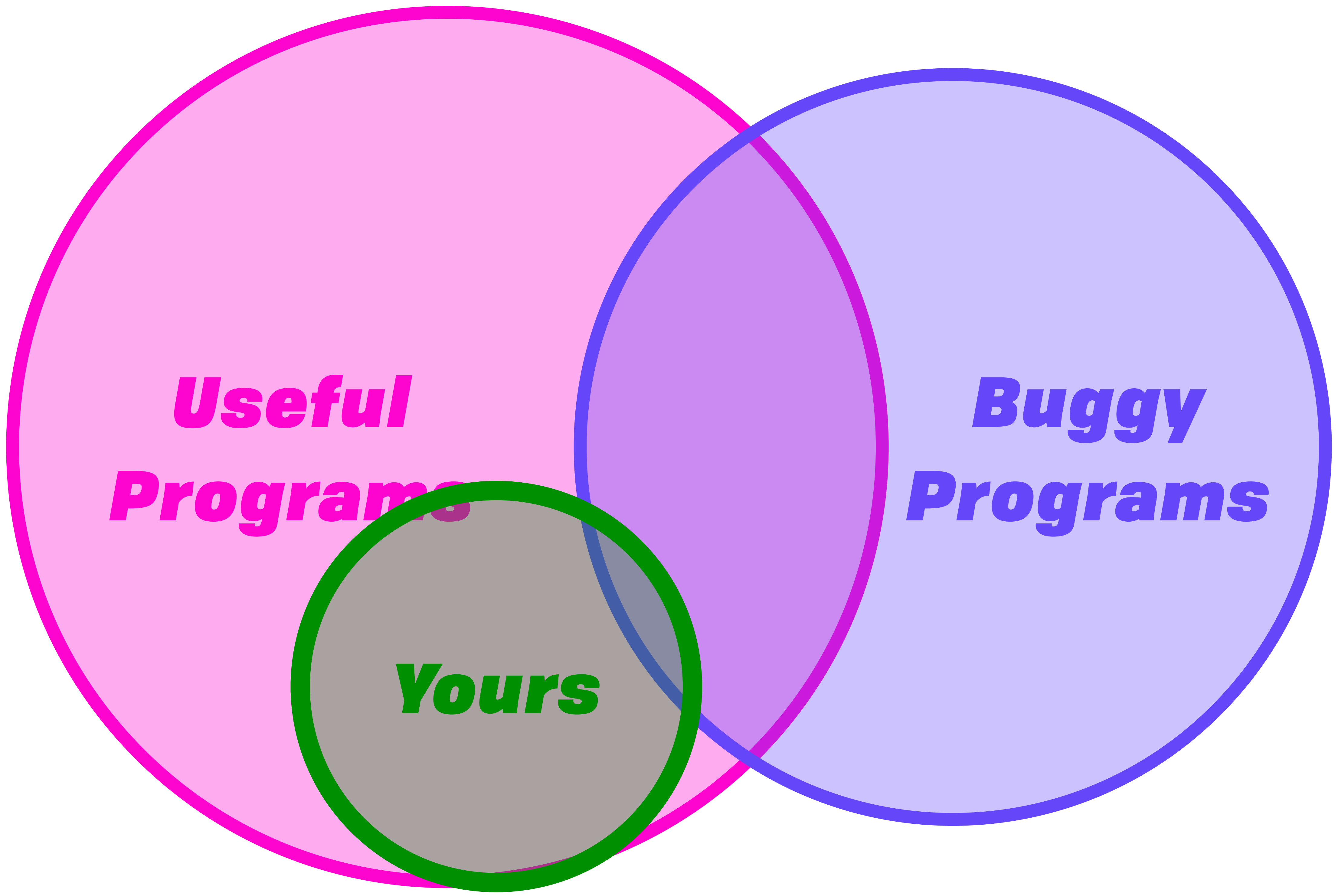
Can a language be ***too expressive?***



Millions of Tiny Languages

Practical Bounds

Can a language be ***too expressive?***



Welcome to the Danger Zone ⚠️

```
System.halt()
```

```
Evil.goto(10)
```

What's So Bad About Control? 🦶🔫

Millions of Tiny Languages ⚛

What's So Bad About Control? 🦶🔫

Line 1

Line 2

Line 3

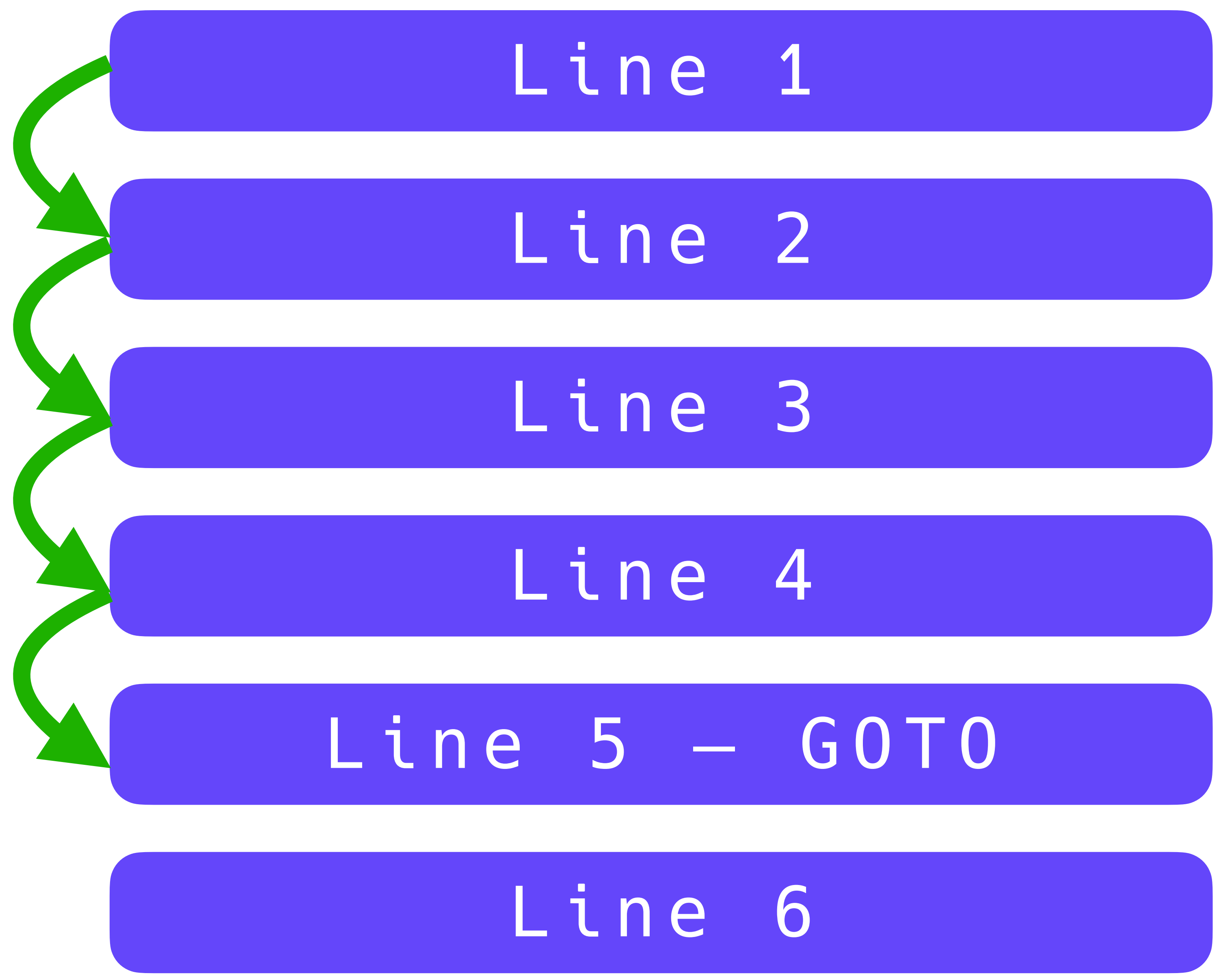
Line 4

Line 5 – GOTO

Line 6

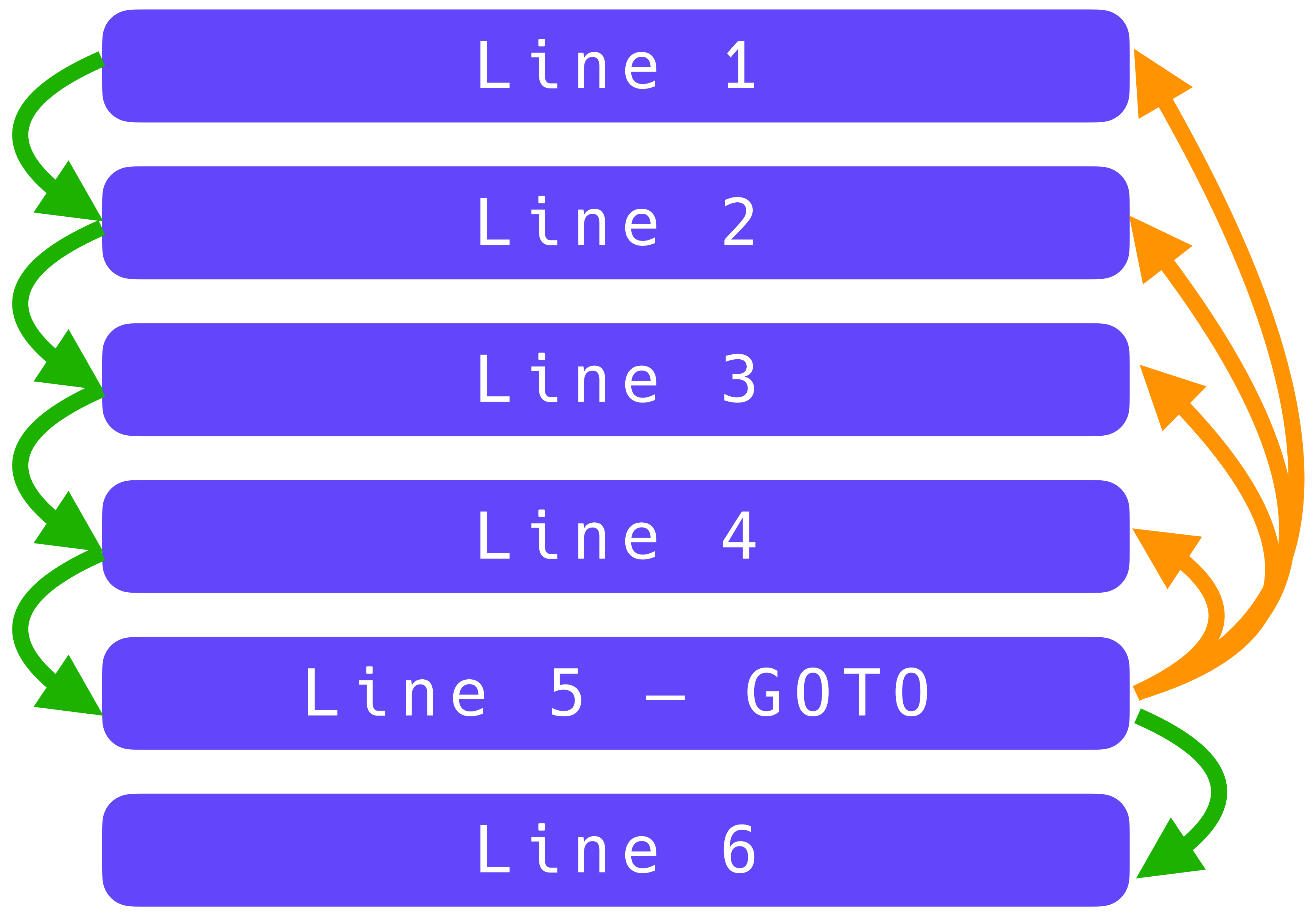
Millions of Tiny Languages ⚛

What's So Bad About Control? 🦶🔫



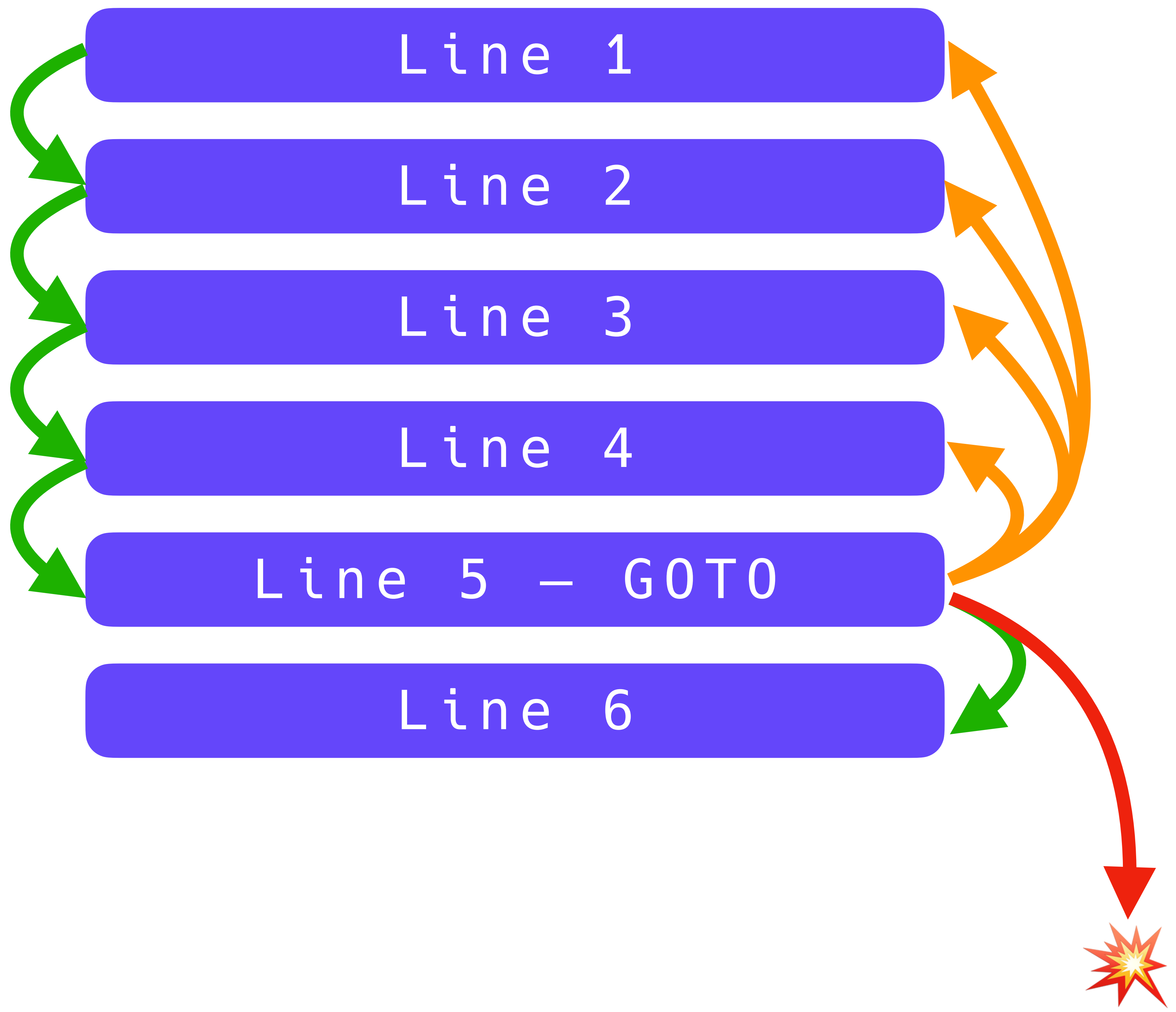
Millions of Tiny Languages ⚛️

What's So Bad About Control? 🦶🔫



What's So Bad About Control? 🦶🔫

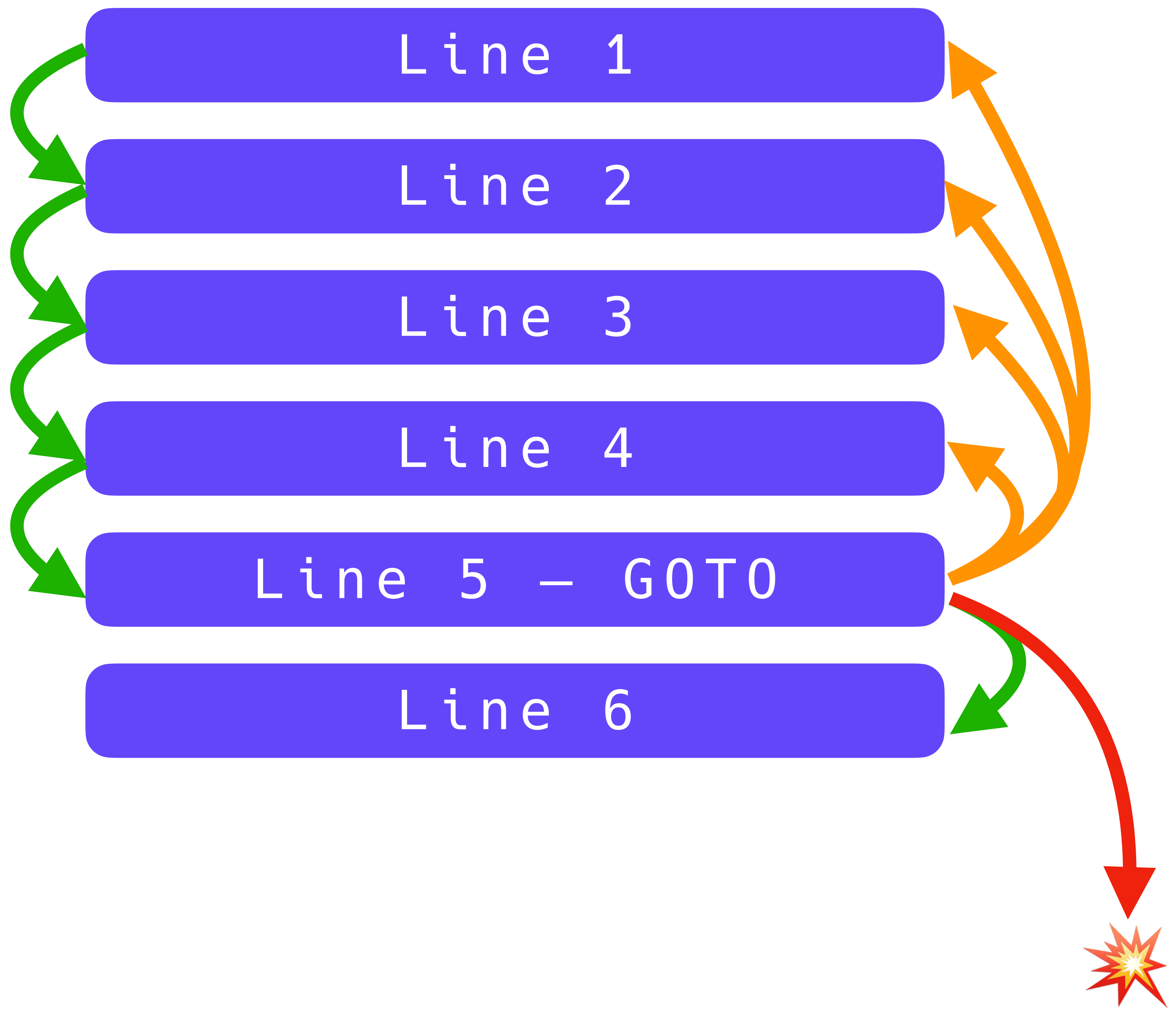
Turing completeness ***considered harmful***



Millions of Tiny Languages ⚛

What's So Bad About Control? 🦶🔫

Turing completeness ***considered harmful***



Aside: "Accidentally" Turing Complete

https://harrisonwl.github.io/assets/courses/malware/spring2017/papers/mov-is-turing-complete.pdf
https://github.com/ealter/vim_turing_machine
https://github.com/Microsoft/TypeScript/issues/14833
https://vanbever.eu/pdfs/vanbever_turing_icnp_2013.pdf
https://arxiv.org/pdf/1904.09828.pdf
https://softwareengineering.stackexchange.com/a/136179

Aside: "Accidentally" Turing Complete

- x86 MOV (just by itself)
- Vim Normal Mode
- BGP
- Peano arithmetic
- Musical Notation
- Sendmail's Config
- TypeScript's Type System
- Magic the Gathering
- Pokemon Yellow
- Unicode (conjectured)

https://harrisonwl.github.io/assets/courses/malware/spring2017/papers/mov-is-turing-complete.pdf
https://github.com/ealter/vim_turing_machine
https://github.com/Microsoft/TypeScript/issues/14833
https://vanbever.eu/pdfs/vanbever_turing_icnp_2013.pdf
https://arxiv.org/pdf/1904.09828.pdf
https://softwareengineering.stackexchange.com/a/136179

Millions of Tiny Languages ⚛️

The Lesson 👩‍🏫

The Lesson 👩‍🏫

The bottom line is that
the ***more powerful a language***
(i.e. the more that is possible within the language),
the ***harder it is to understand*** systems
constructed in it

– Ben Mosley & Peter Marks, Out of the Tarpit

Three Focused Vocabularies

Millions of Tiny Languages ⚛

Three Focused Vocabularies

```elixir
defprotocol ImageManipulation do
  @spec rotate(t, non_neg_integer()) :: t
  def rotate(image, degrees)

  @spec scale(t, integer()) :: t
  def scale(image, percentage)

  @spec translate(t, integer(), non_neg_integer()) :: t
  def translate(image, degrees, distance_in_pixels)
end
```

Millions of Tiny Languages

Three Focused Vocabularies

```elixir
defprotocol RadialGameMovement do
  @spec move(t, integer(), non_neg_integer()) :: t
  def move(entity, degrees, paces)
end
```

```elixir
defprotocol ImageManipulation do
  @spec rotate(t, non_neg_integer()) :: t
  def rotate(image, degrees)

  @spec scale(t, integer()) :: t
  def scale(image, percentage)

  @spec translate(t, integer(), non_neg_integer()) :: t
  def translate(image, degrees, distance_in_pixels)
end
```

Millions of Tiny Languages

Three Focused Vocabularies

```elixir
defprotocol ImageManipulation do
  @spec rotate(t, non_neg_integer()) :: t
  def rotate(image, degrees)

  @spec scale(t, integer()) :: t
  def scale(image, percentage)

  @spec translate(t, integer(), non_neg_integer()) :: t
  def translate(image, degrees, distance_in_pixels)
end
```

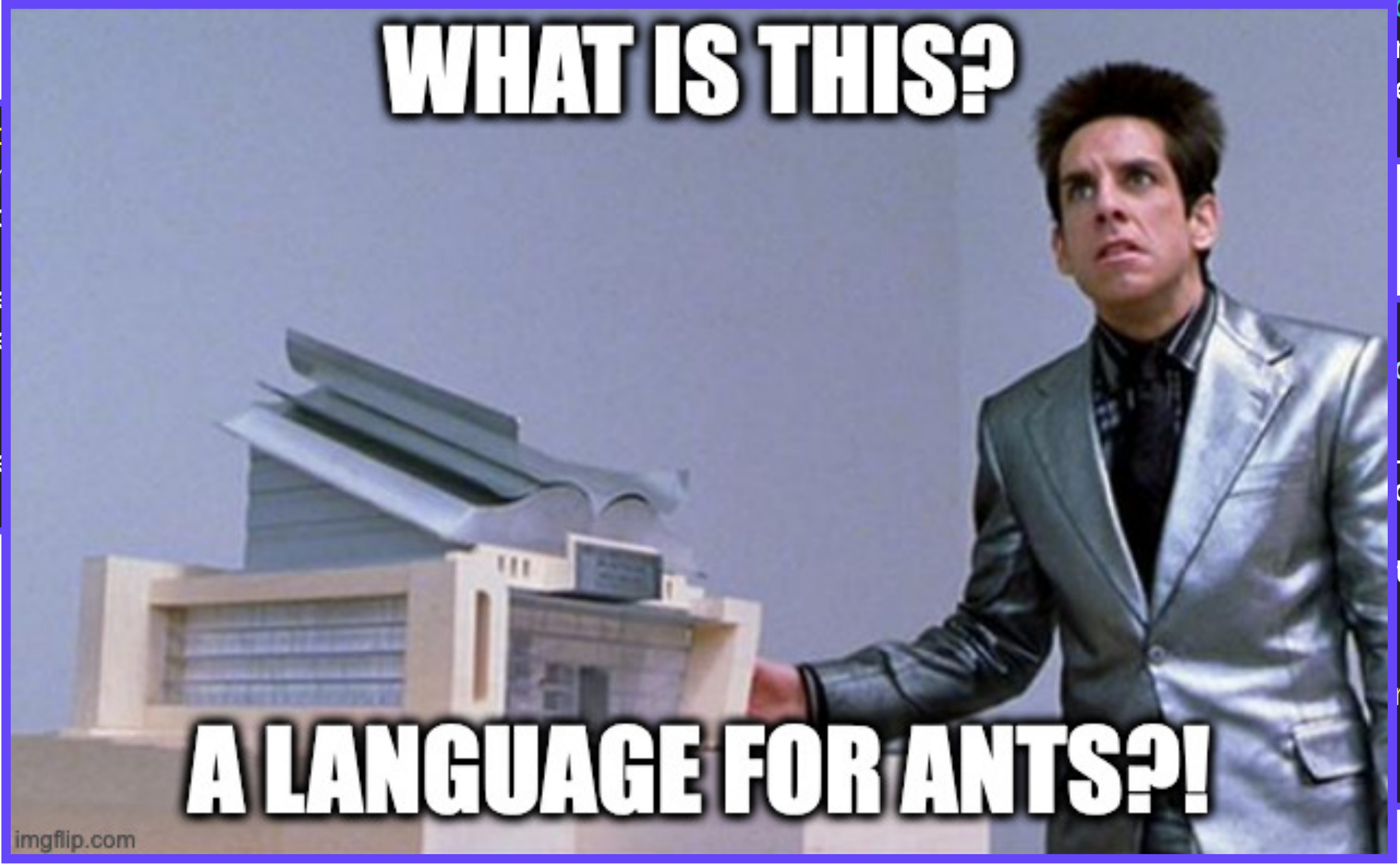
```elixir
defprotocol RadialGameMovement do
  @spec move(t, integer(), non_neg_integer()) :: t
  def move(entity, degrees, paces)
end
```

```elixir
defprotocol GameActions do
  @type direction :: :north | :south | :east | :west

  @spec move(t, direction(), non_neg_number()) :: t
  def move(entity, direction, paces)

  @spec speak(t, String.t()) :: t
  def speak(entity, message)

  @spec listen(t, t) :: {t, String.t()}
  def listen(entity, entity)
end
```

Millions of Tiny Languages

Three Focused Vocabularies

```
defprotocol ImageMani
  @spec rotate(t, non
  def rotate(image, d

  @spec scale(t, inte
  def scale(image, pe

  @spec translate(t,
  def translate(image
end
```




```
do
neg_integer()) :: t
es)
```

```
outh | :east | :west

_neg_number()) :: t
ces)

t

ng.t())
```

Compositional Vocabularies

Millions of Tiny Languages

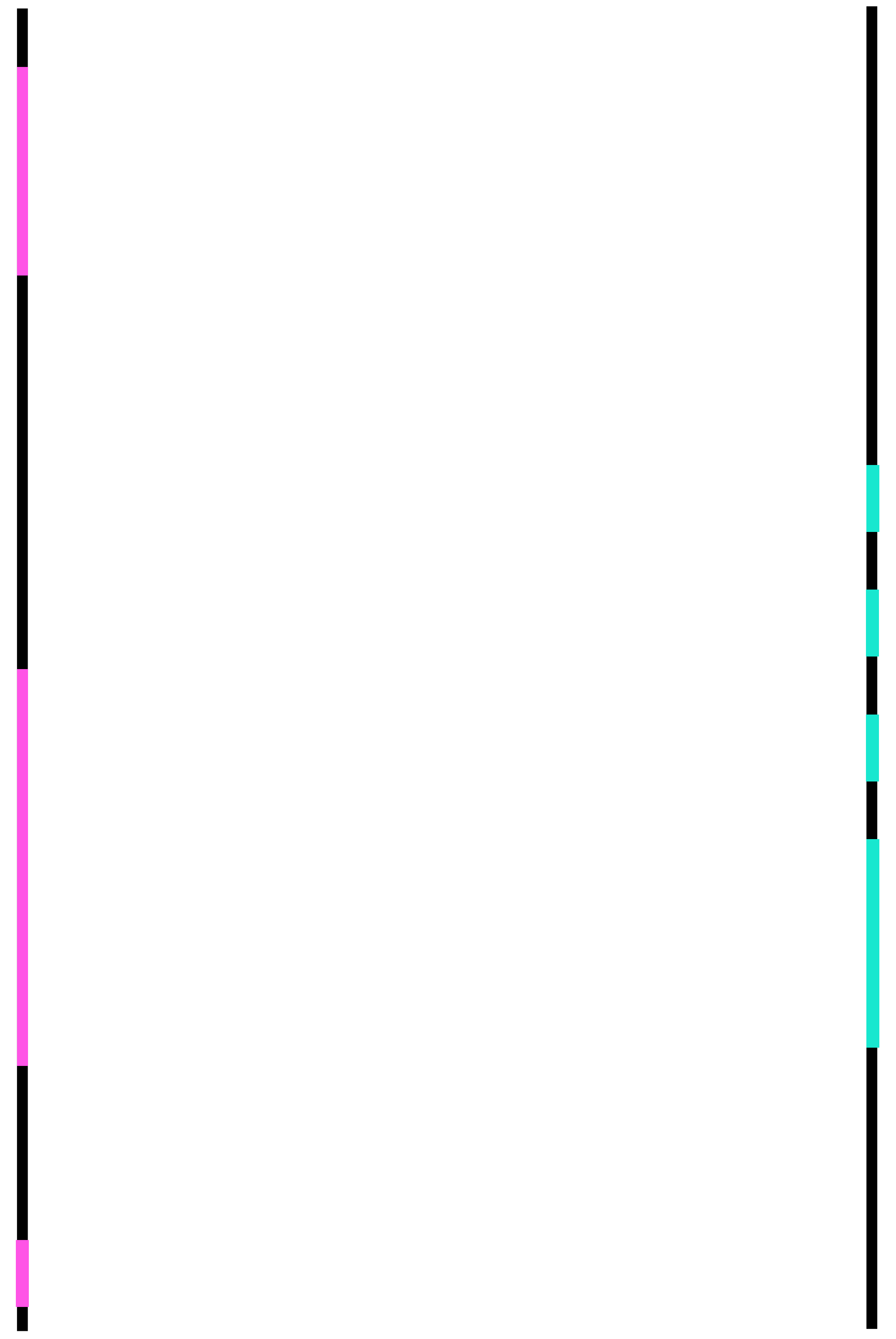
Compositional Vocabularies

Millions of Tiny Languages

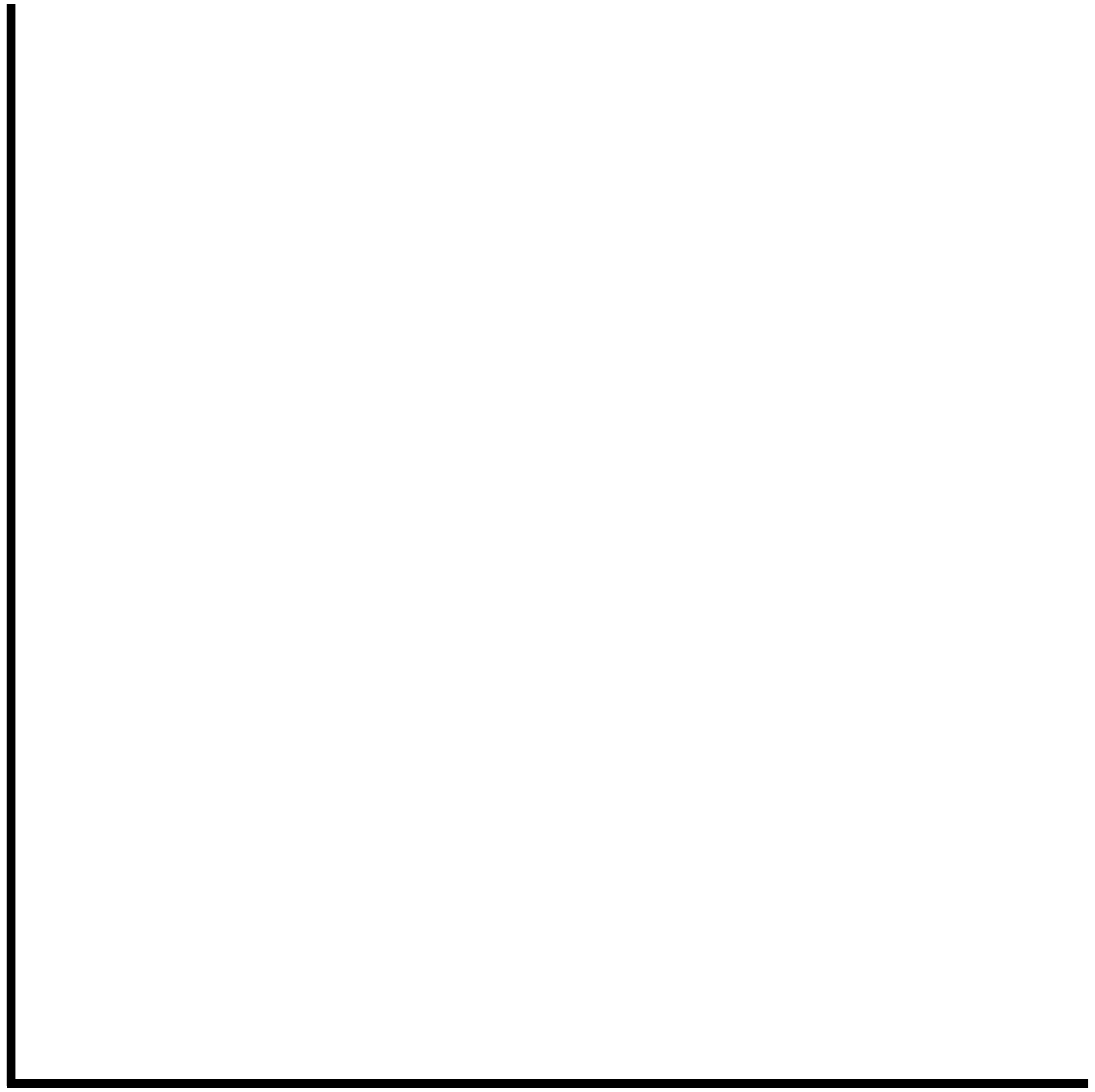
Compositional Vocabularies

Millions of Tiny Languages

Compositional Vocabularies

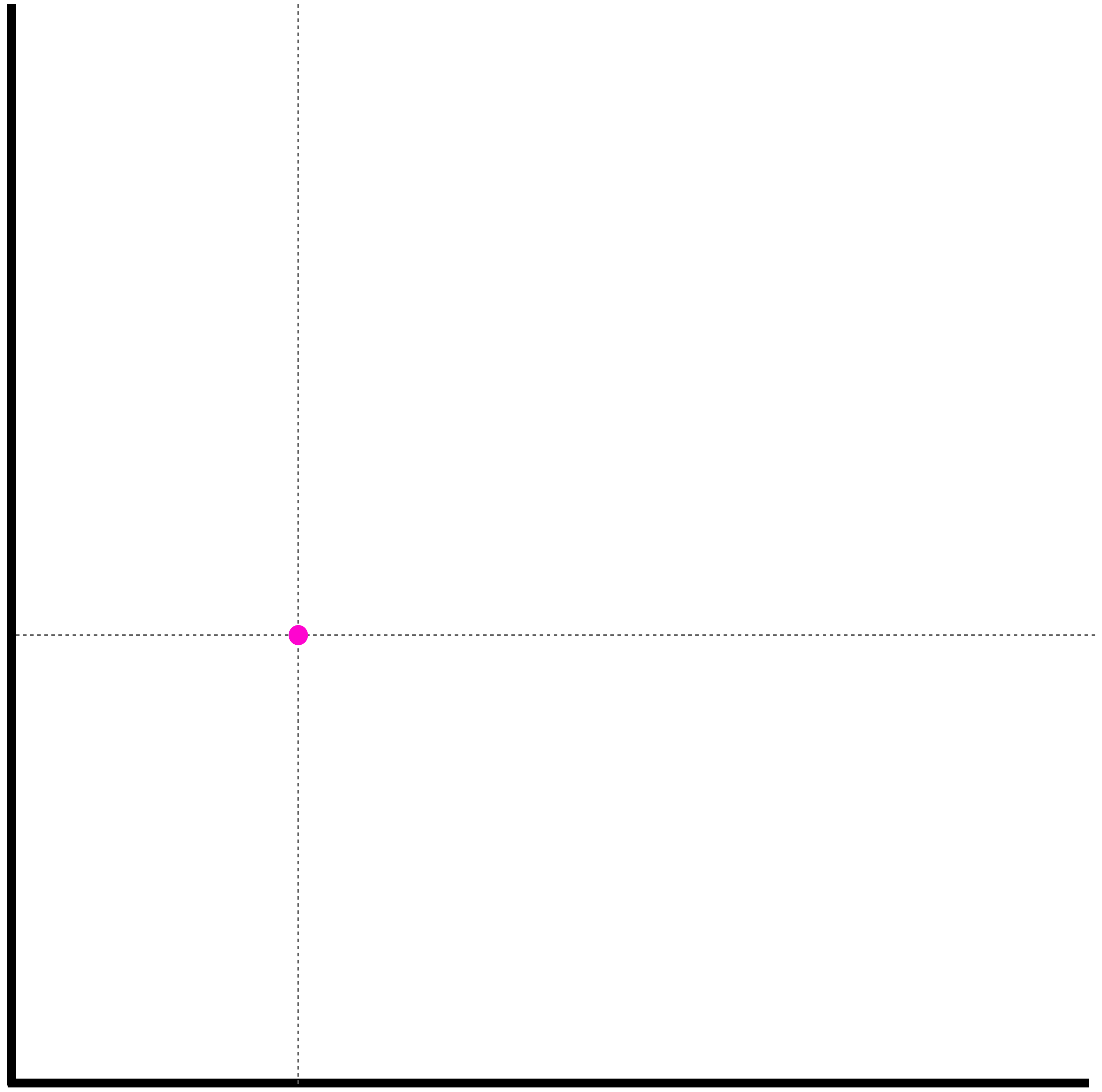


Compositional Vocabularies



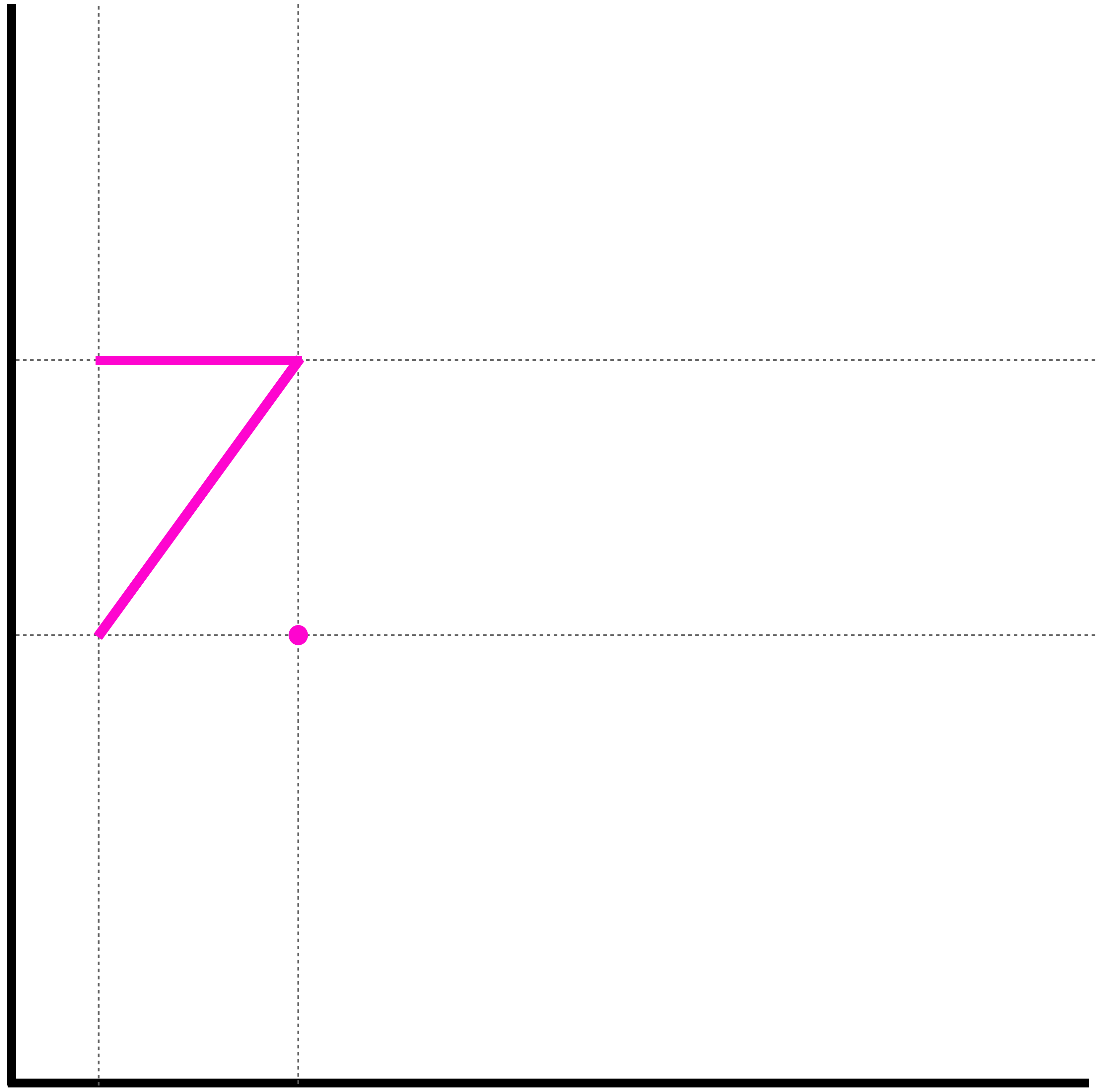
Millions of Tiny Languages

Compositional Vocabularies



Millions of Tiny Languages

Compositional Vocabularies



Compositional Vocabularies

Millions of Tiny Languages

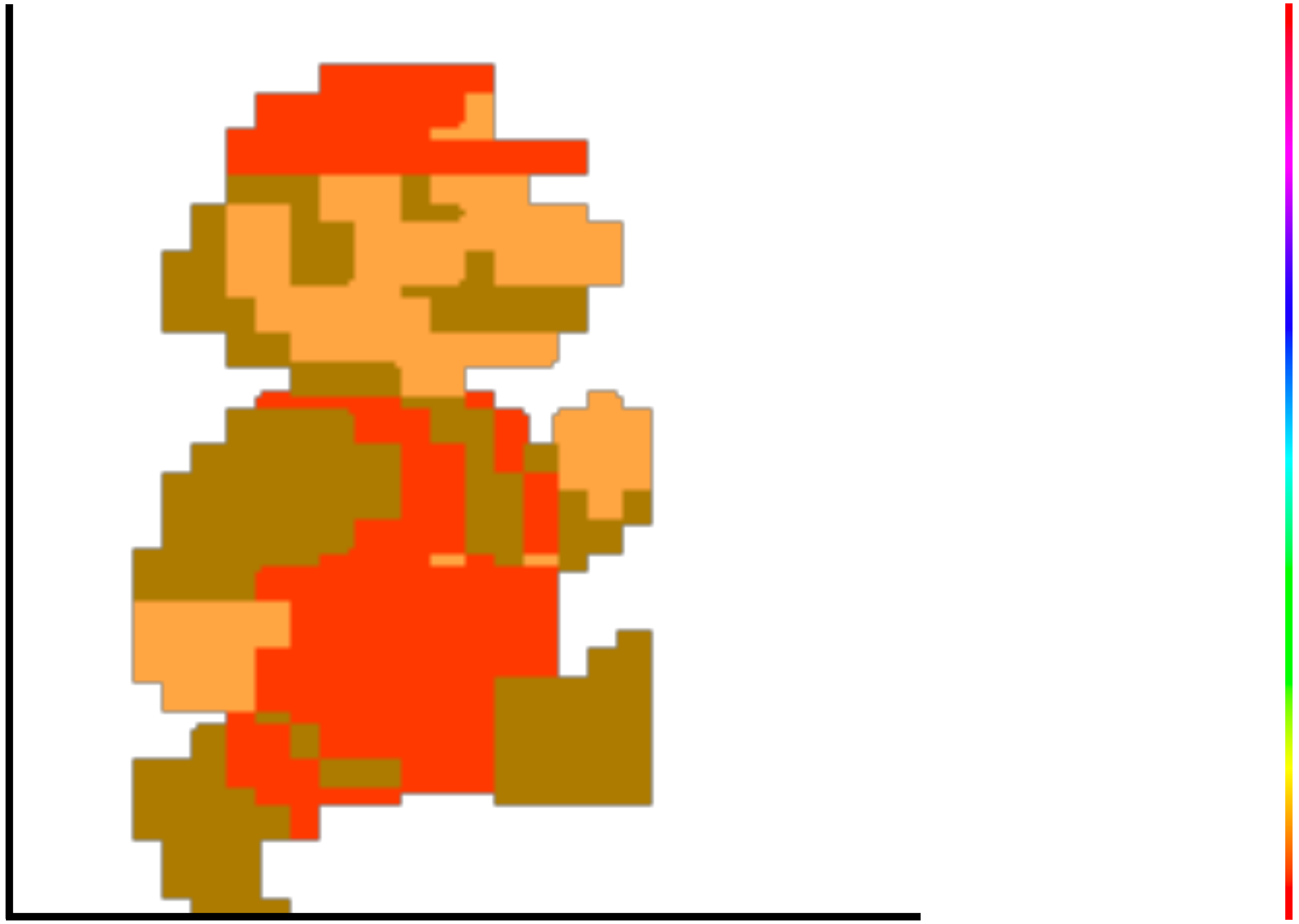
Compositional Vocabularies

Millions of Tiny Languages

Compositional Vocabularies

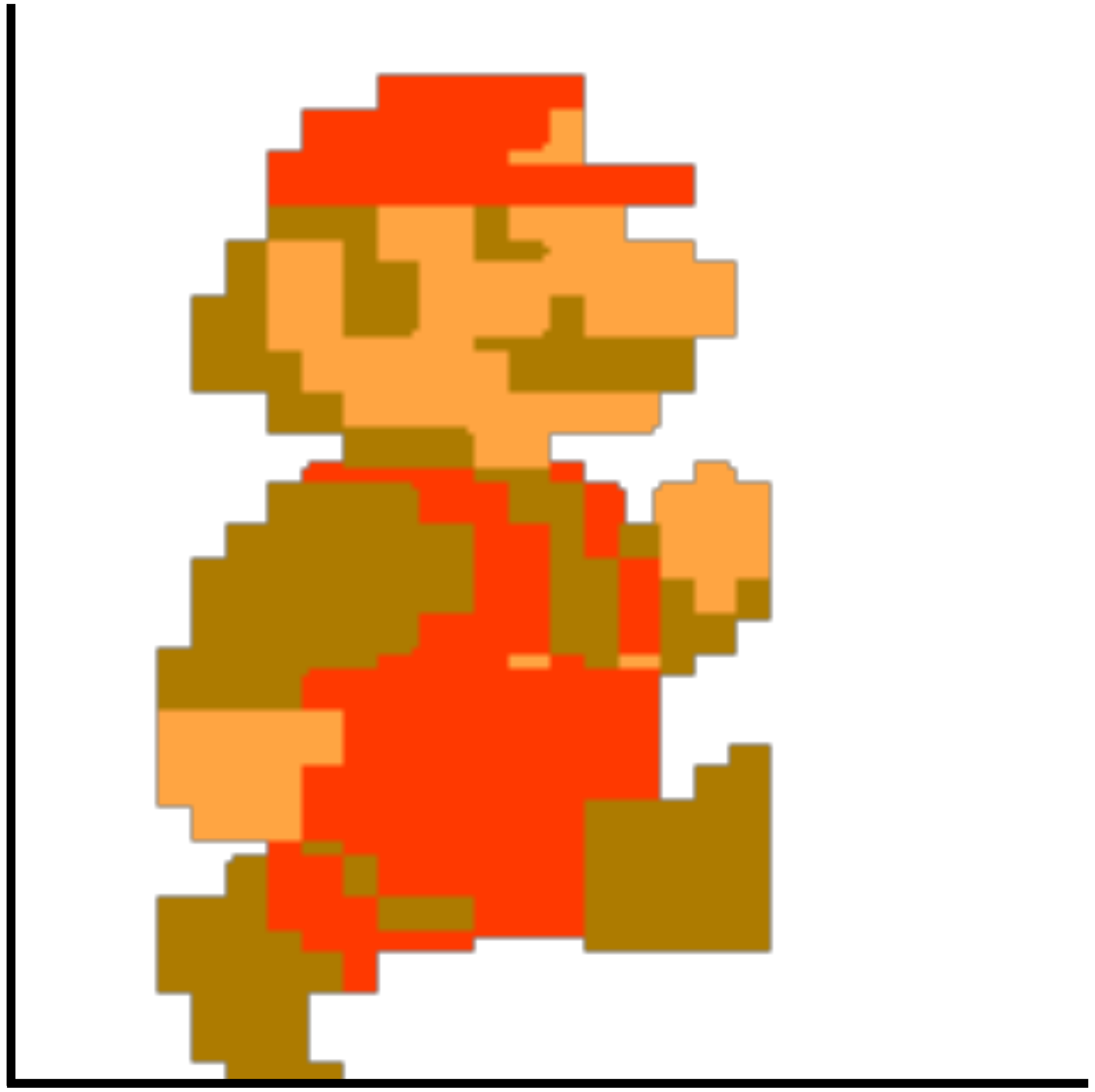
Millions of Tiny Languages

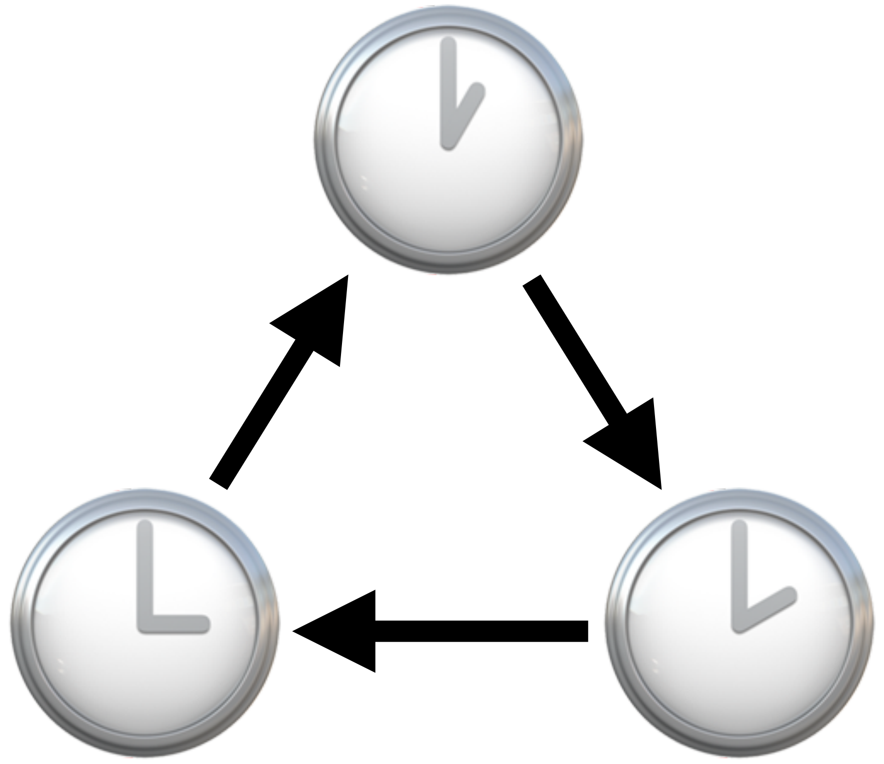
Compositional Vocabularies



Millions of Tiny Languages ⚛

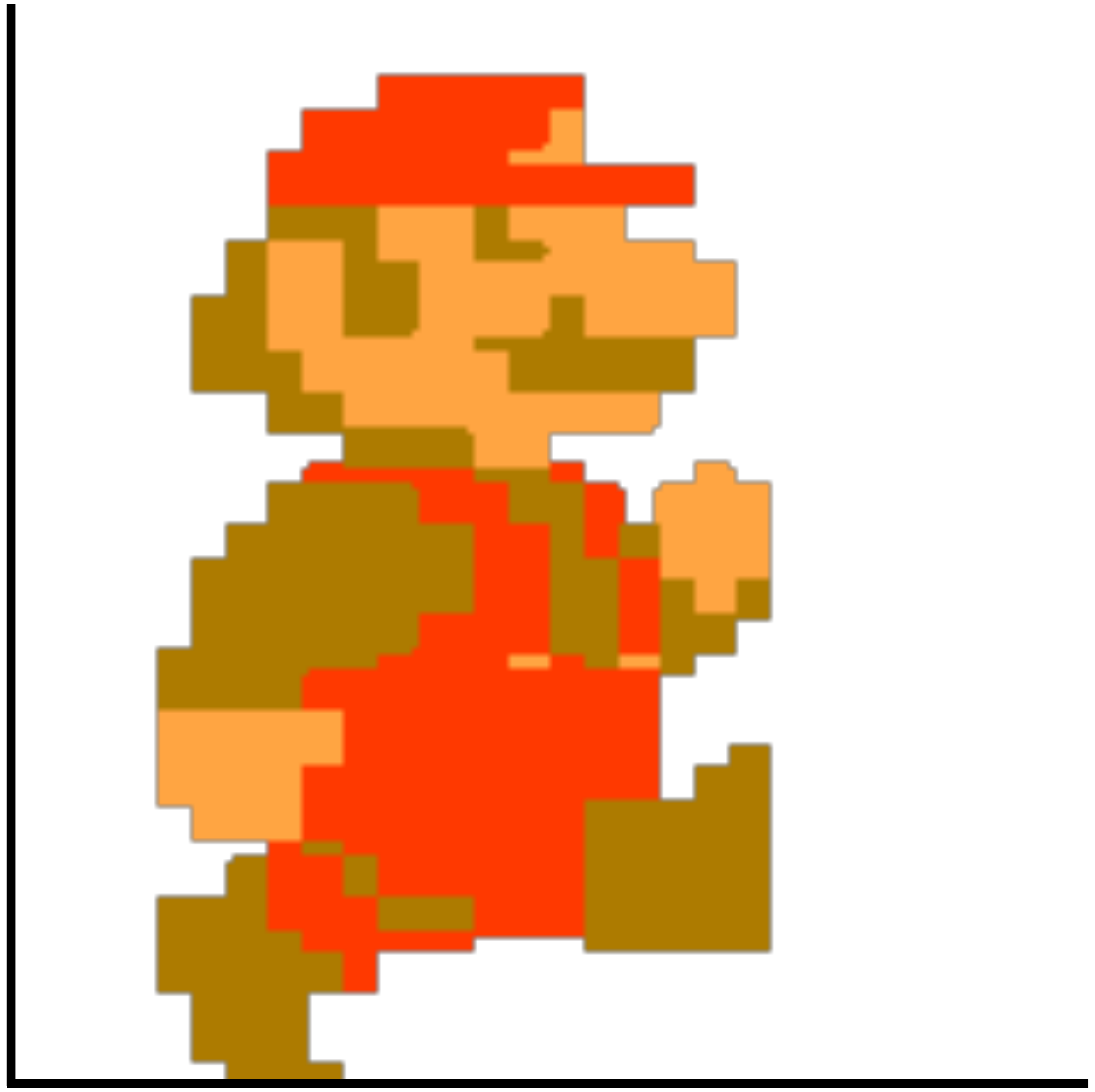
Compositional Vocabularies



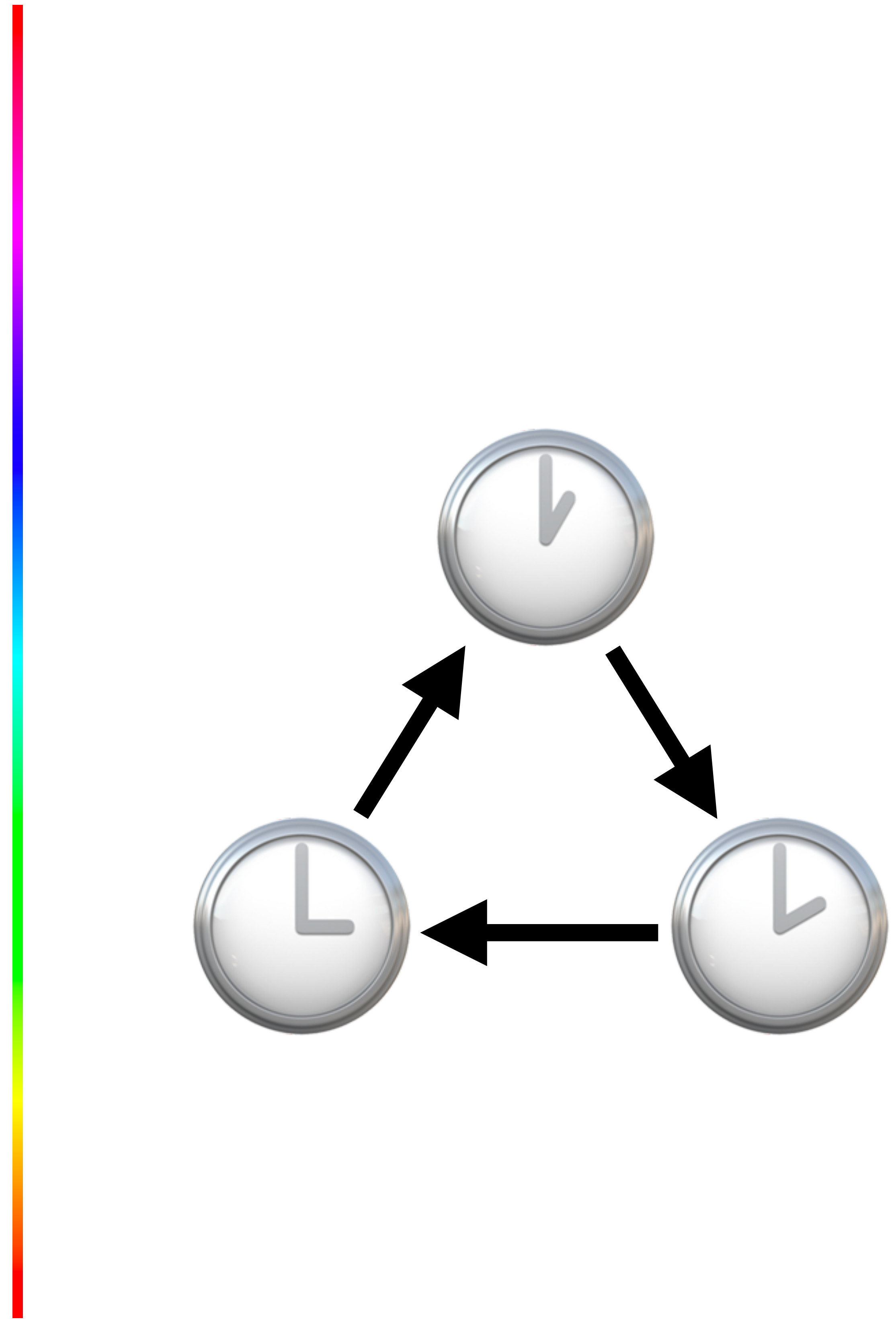


Millions of Tiny Languages ⚛

Compositional Vocabularies

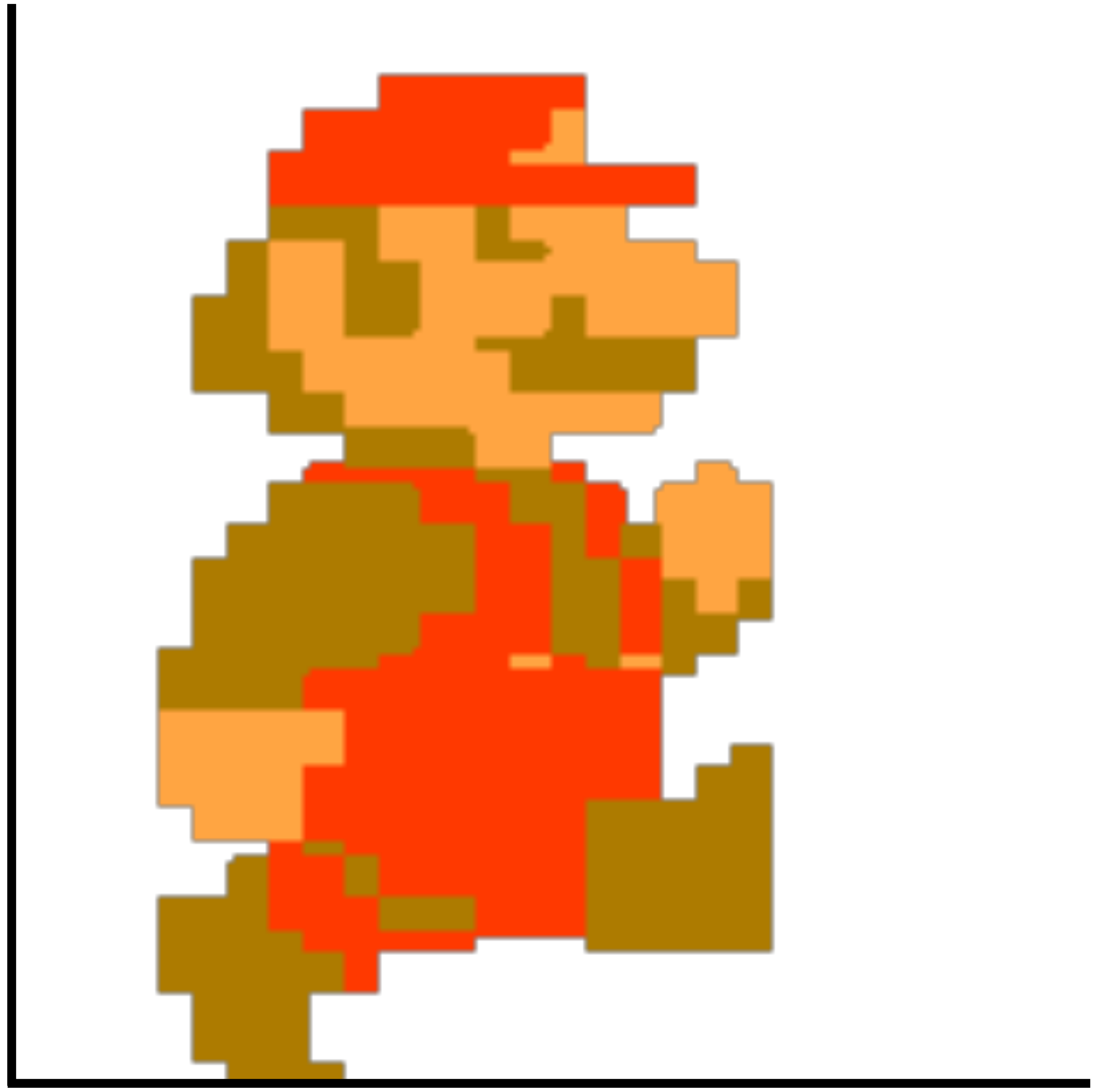


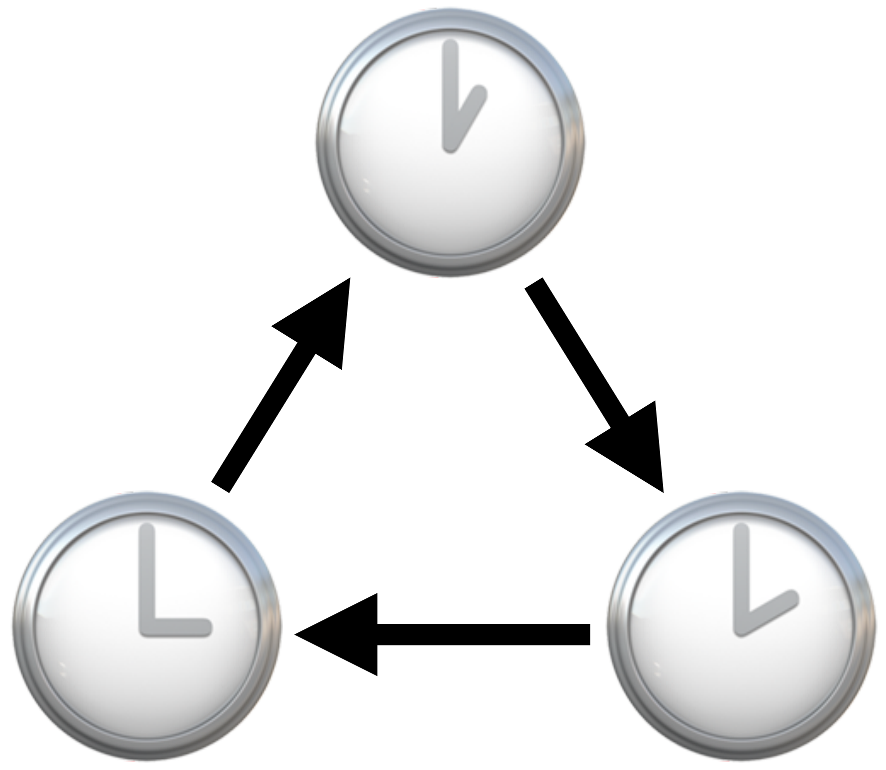
Components: 4



Millions of Tiny Languages

Compositional Vocabularies





Components: 4

Results: All of computer graphics

Millions of Tiny Languages

Interleaved Terms

Millions of Tiny Languages ⚛

Interleaved Terms

```
defprotocol ImageManipulation do
  @spec rotate(t, non_neg_integer()) :: t
  def rotate(image, degrees)

  @spec scale(t, integer()) :: t
  def scale(image, percentage)

  @spec translate(t, integer(), non_neg_integer()) :: t
  def translate(image, degrees, distance_in_pixels)
end
```

Millions of Tiny Languages

Interleaved Terms

```elixir
defprotocol ImageManipulation do
  @spec rotate(t, non_neg_integer()) :: t
  def rotate(image, degrees)

  @spec scale(t, integer()) :: t
  def scale(image, percentage)

  @spec translate(t, integer(), non_neg_integer()) :: t
  def translate(image, degrees, distance_in_pixels)
end
```

```elixir
defprotocol GameActions do
  @type direction :: :north | :south | :east | :west

  @spec move(t, direction(), non_neg_number()) :: t
  def move(entity, direction, paces)

  @spec speak(t, String.t()) :: t
  def speak(entity, message)

  @spec listen(t, t) :: {t, String.t()}
  def listen(entity, entity)
end
```

Millions of Tiny Languages

Interleaved Terms

```elixir
defprotocol ImageManipulation do
  @spec rotate(t, non_neg_integer()) :: t
  def rotate(image, degrees)

  @spec scale(t, integer()) :: t
  def scale(image, percentage)

  @spec translate(t, integer(), non_neg_integer()) :: t
  def translate(image, degrees, distance_in_pixels)
end
```

```elixir
defprotocol GameActions do
  @type direction :: :north | :south | :east | :west

  @spec move(t, direction(), non_neg_number()) :: t
  def move(entity, direction, paces)

  @spec speak(t, String.t()) :: t
  def speak(entity, message)

  @spec listen(t, t) :: {t, String.t()}
  def listen(entity, entity)
end
```

```elixir
character
|> rotate(180)
|> speak("Why is the world upside down?")
|> scale(250)
```

✨ An AST of Your Own ✨

Building Up & Tearing Down

🧱🧨

Building Up & Tearing Down 🧱🧨

Shallow Pipes

https://hexdocs.pm/ecto/Ecto.Changeset.html#module-validations-and-constraints

Building Up & Tearing Down 🧱🧨

Shallow Pipes

- Just use the built-in AST
- What it can represent is limited
- Single, canonical implementations
- e.g. most libraries

```elixir
def changeset(user, params \\ %{}) do
  user
   |> cast(params, [:name, :email, :age])
   |> validate_required([:name, :email])
   |> validate_format(:email, ~r/@/)
   |> validate_inclusion(:age, 18..100)
   |> unique_constraint(:email)
end
```

https://hexdocs.pm/ecto/Ecto.Changeset.html#module-validations-and-constraints

Building Up & Tearing Down 🧱🧨

Shallow Pipes

https://github.com/witchcrafters/algae/blob/main/lib/algae/tree/binary_search.ex

Building Up & Tearing Down 🧱🧨

Shallow Pipes

```elixir
defmodule Algae.Tree.BinarySearch do
  alias __MODULE__, as: BST
  import Algae

  defsum do
    defdata(Empty :: none())

    defdata Node do
      node :: any()
      left :: BinarySearch.t() \\ BinarySearch.Empty.new()
      right :: BinarySearch.t() \\ BinarySearch.Empty.new()
    end
  end

  def new, do: %Empty{}
  def new(value), do: %Node{node: value}

  # ...snip...
end
```

https://github.com/witchcrafters/algae/blob/main/lib/algae/tree/binary_search.ex

Building Up & Tearing Down 🧱🧨

Shallow Pipes

```elixir
defmodule Algae.Tree.BinarySearch do
  alias __MODULE__, as: BST
  import Algae

  defsum do
    defdata(Empty :: none())

    defdata Node do
      node :: any()
      left :: BinarySearch.t() \\ BinarySearch.Empty.new()
      right :: BinarySearch.t() \\ BinarySearch.Empty.new()
    end
  end

  def new, do: %Empty{}
  def new(value), do: %Node{node: value}

  # ...snip...
end
```

https://github.com/witchcrafters/algae/blob/main/lib/algae/tree/binary_search.ex

Building Up & Tearing Down 🧱🧨

Shallow Pipes

```
BSTree.Node.new(
  42,
  BSTree.Node.new(77),
  BSTree.Node.new(
    1234,
    BSTree.Node.new(98),
    BSTree.Node.new(32)
  )
)
```

```
defmodule Algae.Tree.BinarySearch do
  alias __MODULE__, as: BST
  import Algae

  defsum do
    defdata(Empty :: none())

    defdata Node do
      node :: any()
      left :: BinarySearch.t() \\ BinarySearch.Empty.new()
      right :: BinarySearch.t() \\ BinarySearch.Empty.new()
    end
  end

  def new, do: %Empty{}
  def new(value), do: %Node{node: value}

  # ...snip...
end
```

https://github.com/witchcrafters/algae/blob/main/lib/algae/tree/binary_search.ex

Building Up & Tearing Down 🧱🧨

Shallow Pipes

```elixir
defmodule Algae.Tree.BinarySearch do
  alias __MODULE__, as: BST
  import Algae

  defsum do
    defdata(Empty :: none())

    defdata Node do
      node :: any()
      left :: BinarySearch.t() \\ BinarySearch.Empty.new()
      right :: BinarySearch.t() \\ BinarySearch.Empty.new()
    end
  end

  def new, do: %Empty{}
  def new(value), do: %Node{node: value}

  # ...snip...
end
```

```elixir
BSTree.Node.new(
  42,
  BSTree.Node.new(77),
  BSTree.Node.new(
    1234,
    BSTree.Node.new(98),
    BSTree.Node.new(32)
  )
)
```

```elixir
%Algae.Tree.BinarySearch.Node{
  node: 42,
  left: %Algae.Tree.BinarySearch.Node{
    node:  77,
    left:  %Algae.Tree.BinarySearch.Empty{},
    right: %Algae.Tree.BinarySearch.Empty{}
  },
  right: %Algae.Tree.BinarySearch.Node{
    node:  1234,
    left:  %Algae.Tree.BinarySearch.Node{
      node:  98,
      left:  %Algae.Tree.BinarySearch.Empty{},
      right: %Algae.Tree.BinarySearch.Empty{}
    },
    right: %Algae.Tree.BinarySearch.Node{
      node:  32,
      left:  %Algae.Tree.BinarySearch.Empty{},
      right: %Algae.Tree.BinarySearch.Empty{}
    }
  }
}
```

https://github.com/witchcrafters/algae/blob/main/lib/algae/tree/binary_search.ex

Building Up & Tearing Down

Whatever You Want It To Be

- Natural Language
- GUI Metaphor
- Application Domain
- Library / Framework
- Elixir Language
- Elixir AST
- BEAM bytecode
- Kernel syscalls
- x86 / ARM / RISC-V
- Binary
- Physics
- Mathematics

Building Up & Tearing Down

Whatever You Want It To Be

There is ***nothing sacred***
about Elixir's AST;
it's just ***well suited***
for its ecological niche

- Natural Language
- GUI Metaphor
- Application Domain
- Library / Framework
- Elixir Language
- **Elixir AST**
- BEAM bytecode
- Kernel syscalls
- x86 / ARM / RISC-V
- Binary
- Physics
- Mathematics

Building Up & Tearing Down 🧱🧨

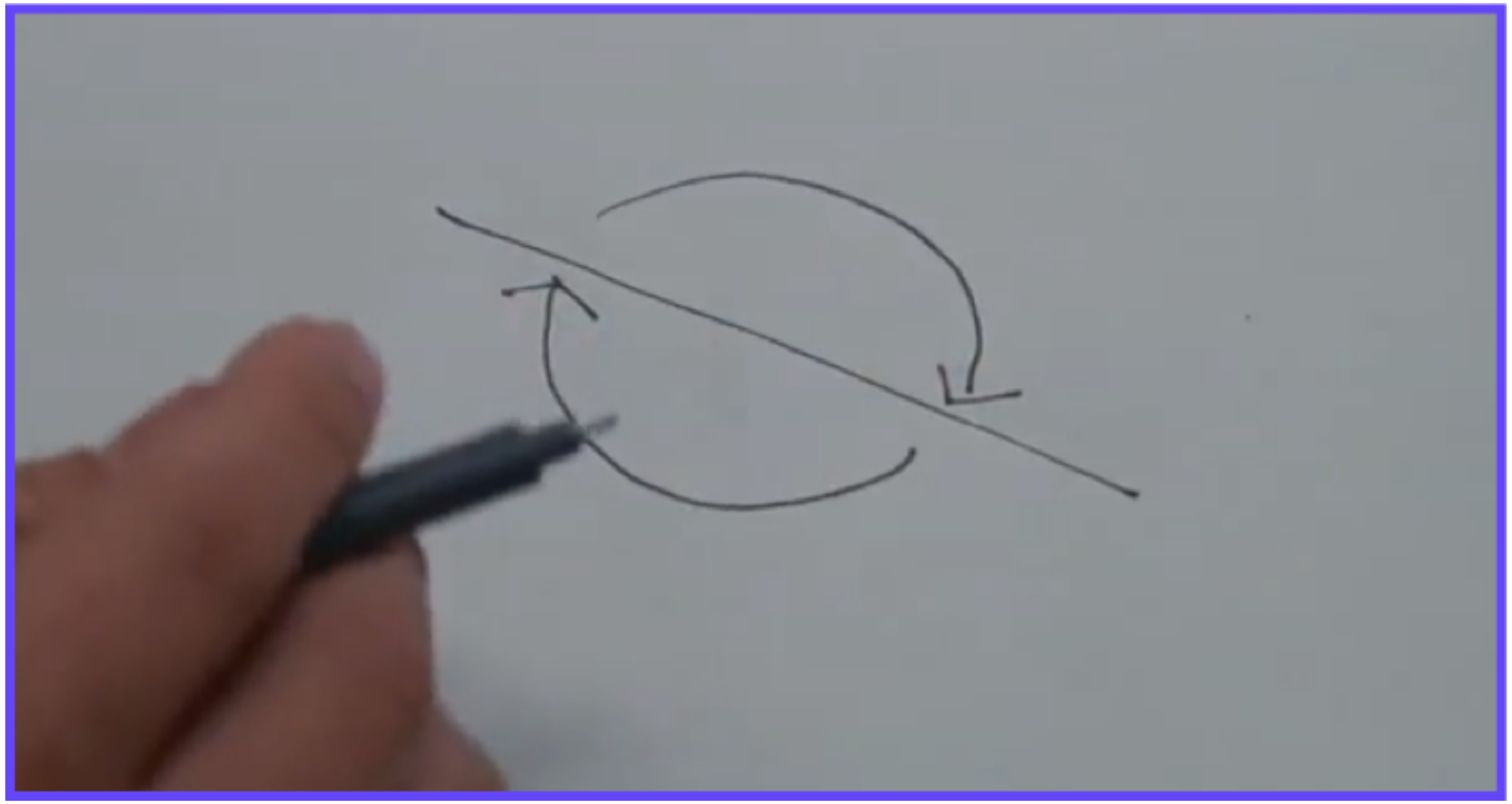
Whatever You Want It To Be

There is ***nothing sacred***
about Elixir's AST;
it's just ***well suited***
for its ecological niche

- Natural Language
- GUI Metaphor
- Application Domain
- Library / Framework
- Elixir Language
- **Elixir AST**
- BEAM bytecode
- Kernel syscalls
- x86 / ARM / RISC-V
- Binary
- Physics
- Mathematics

Building Up & Tearing Down 🧱🧨

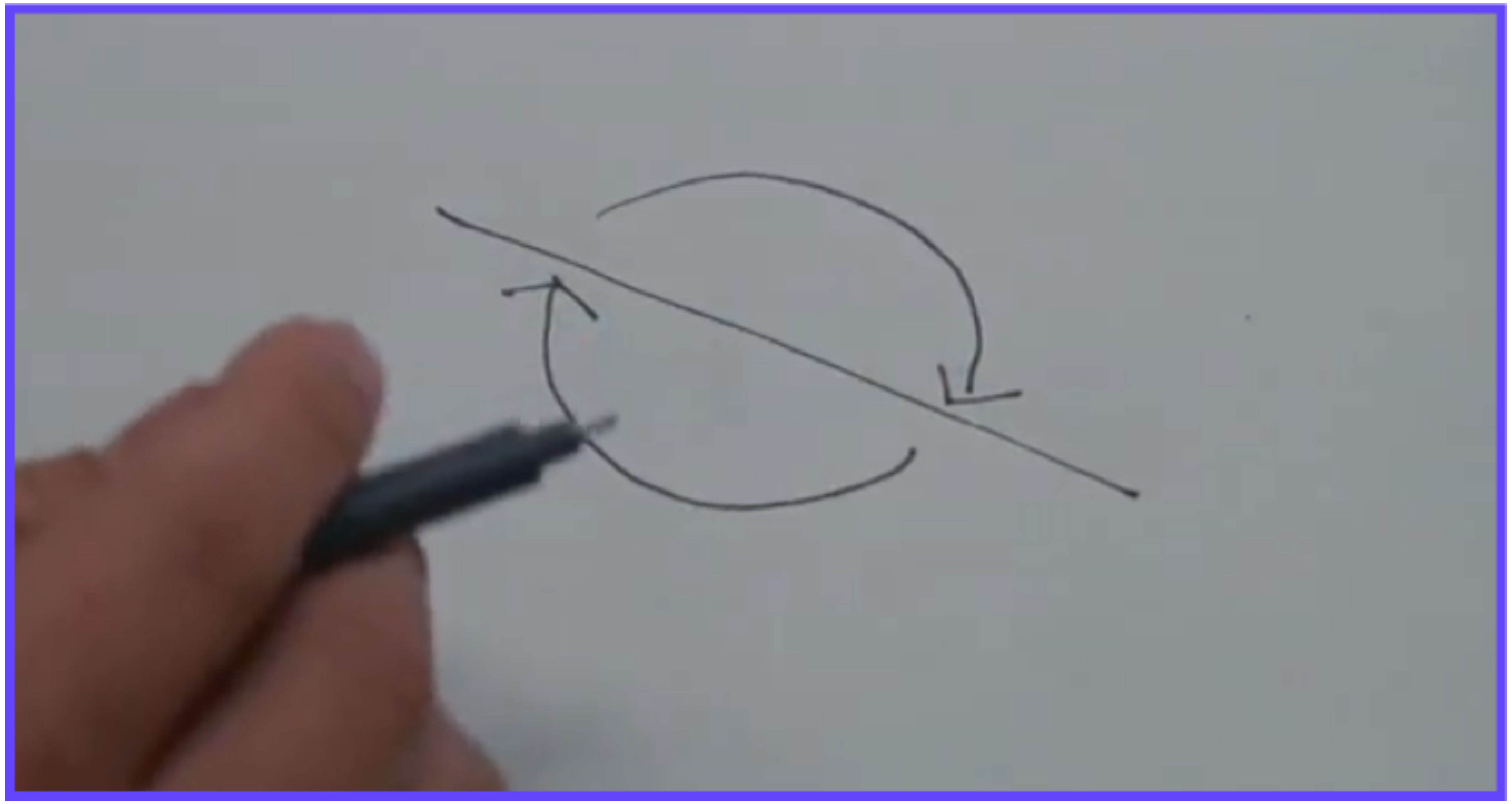
Invocation & Rewriting



Building Up & Tearing Down 🧱🧨

Invocation & Rewriting

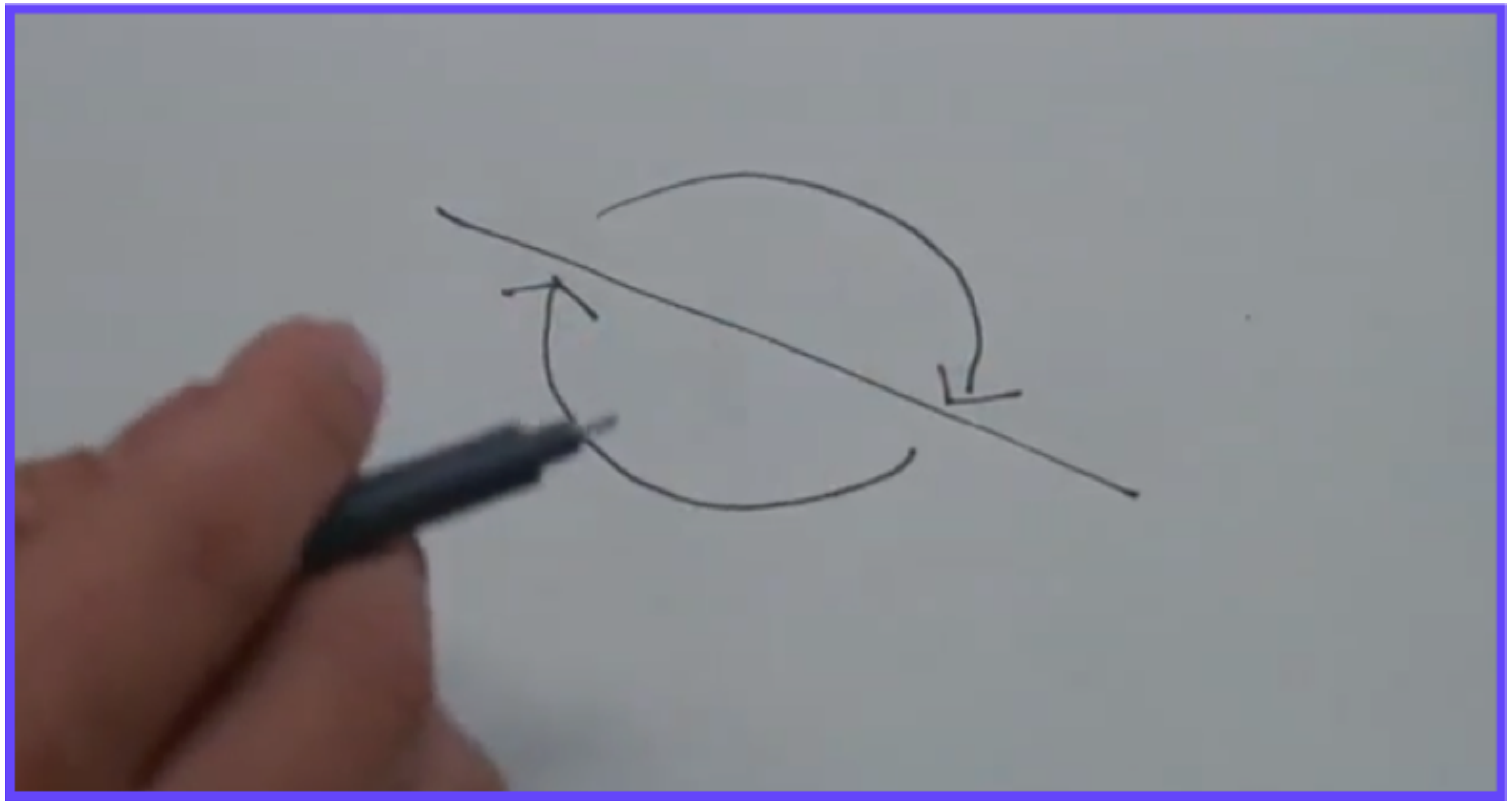
1. Build a game plan



Building Up & Tearing Down 🧱🧨

Invocation & Rewriting

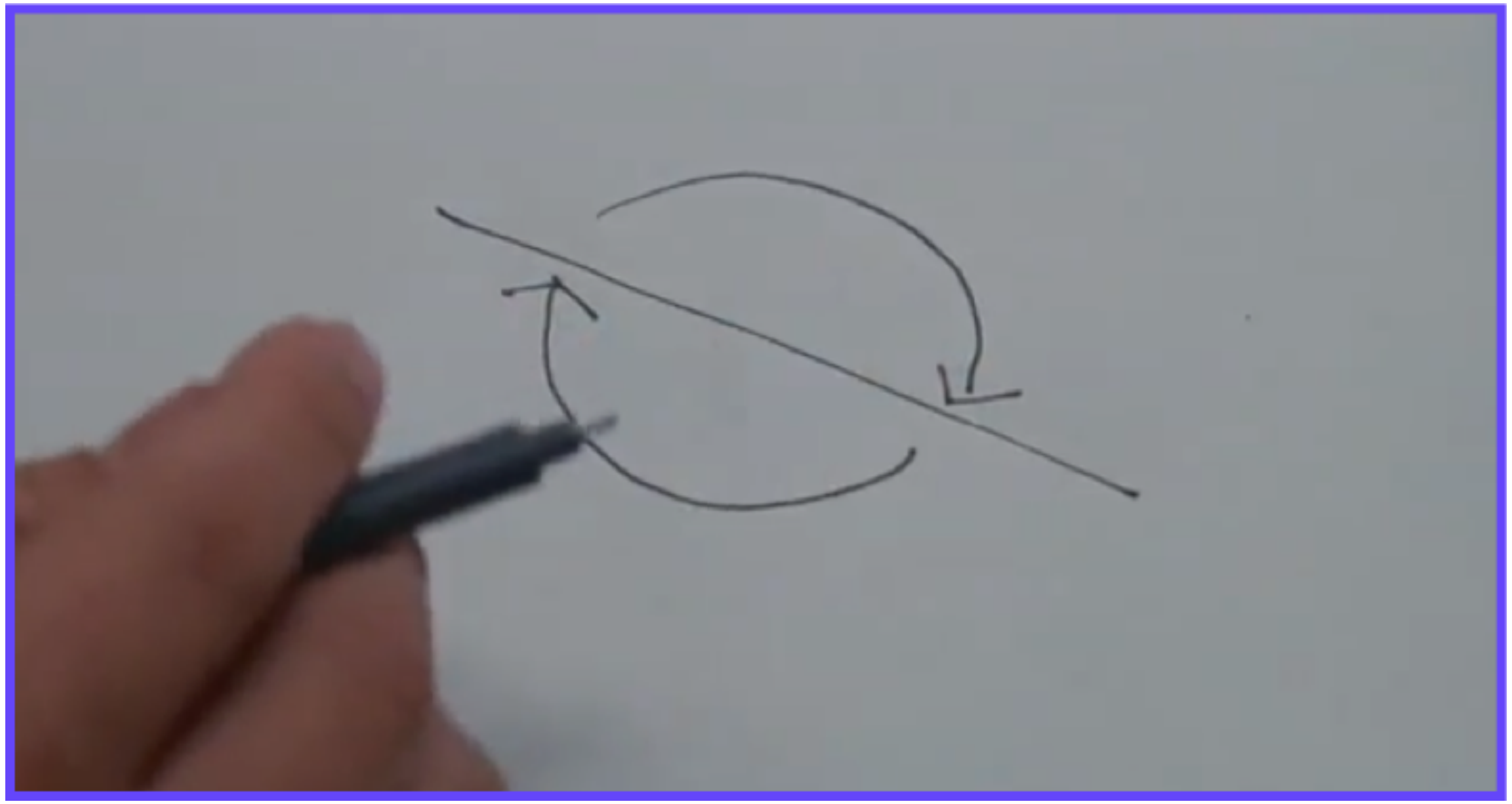
1. Build a game plan
2. Transform (optional)



Building Up & Tearing Down 🧱🧨

Invocation & Rewriting

1. Build a game plan
2. Transform (optional)
3. Tear down



Building Up & Tearing Down 🧱🧨

Connectives

Building Up & Tearing Down 🧱🧨

Connectives

```
with_npc :dog do
  move(:north, 2)
  wait(2, :seconds)

  with_caps do
    say("woof")
    wait(10, :seconds)
    say("bark bark")
  end

  move(:west, 7)
  wait(2, :seconds)
end
```

Building Up & Tearing Down 🧱🧨

Connectives

```
with_npc :dog do
  move(:north, 2)
  wait(2, :seconds)

  with_caps do
    say("woof")
    wait(10, :seconds)
    say("bark bark")
  end

  move(:west, 7)
  wait(2, :seconds)
end
```

Building Up & Tearing Down 🧱🧨

Connectives

```elixir
with_npc :dog do
  move(:north, 2)
  wait(2, :seconds)

  with_caps do
    say("woof")
    wait(10, :seconds)
    say("bark bark")
  end

  move(:west, 7)
  wait(2, :seconds)
end
```

```elixir
[
  %GoNorth{mover: :dog, distance: 2},
  %Wait{seconds: 2},
  %Text.Capslock{on: true},
  %Say{speaker: :dog, text: "woof"},
  %Wait{seconds: 10}
  %Say{speaker: :dog, text: "bark bark"},
  %Text.Capslock{on: false},
  %GoWest{mover: :dog, distance: 7},
  %Wait{seconds: 2}
]
```

Building Up & Tearing Down 🧱🧨

Connectives

```elixir
with_npc :dog do
  move(:north, 2)
  wait(2, :seconds)

  with_caps do
    say("woof")
    wait(10, :seconds)
    say("bark bark")
  end

  move(:west, 7)
  wait(2, :seconds)
end
```

```elixir
[
  %GoNorth{mover: :dog, distance: 2},
  %Wait{seconds: 2},
  %Text.Capslock{on: true},
  %Say{speaker: :dog, text: "woof"},
  %Wait{seconds: 10}
  %Say{speaker: :dog, text: "bark bark"},
  %Text.Capslock{on: false},
  %GoWest{mover: :dog, distance: 7},
  %Wait{seconds: 2}
]
```

Building Up & Tearing Down

Connectives

```elixir
with_npc :dog do
  move(:north, 2)
  wait(2, :seconds)

  with_caps do
    say("woof")
    wait(10, :seconds)
    say("bark bark")
  end

  move(:west, 7)
  wait(2, :seconds)
end
```

```elixir
[
  %GoNorth{mover: :dog, distance: 2},
  %Wait{seconds: 2},
  %Text.Capslock{on: true},
  %Say{speaker: :dog, text: "woof"},
  %Wait{seconds: 10}
  %Say{speaker: :dog, text: "bark bark"},
  %Text.Capslock{on: false},
  %GoWest{mover: :dog, distance: 7},
  %Wait{seconds: 2}
]
```

```elixir
%GoNorth{
  mover: :dog,
  distance: 2,
  then: %Wait{
    seconds: 2,
    then: %Text.WithCaps{
      do: %Say{
            speaker: :dog,
            text: "woof",
            then: %Wait{
              seconds: 10,
              then: %Say{
                speaker: :dog,
                text: "bark bark"
              }
            }
          },
      then: %GoWest{
        mover: :dog,
        distance: 7,
        then: %Wait{seconds: 2}
      }
    }
  }
}
```

Building Up & Tearing Down 🧱🧨

Connectives

```
with_npc :dog do
  move(:north, 2)
  wait(2, :seconds)

  with_caps do
    say("woof")
    wait(10, :seconds)
    say("bark bark")
  end

  move(:west, 7)
  wait(2, :seconds)
end
```

```
[
  %GoNorth{mover: :dog, distance: 2},
  %Wait{seconds: 2},
  %Text.Capslock{on: true},
  %Say{speaker: :dog, text: "woof"},
  %Wait{seconds: 10}
  %Say{speaker: :dog, text: "bark bark"},
  %Text.Capslock{on: false},
  %GoWest{mover: :dog, distance: 7},
  %Wait{seconds: 2}
]
```

```
%GoNorth{
  mover: :dog,
  distance: 2,
  then: %Wait{
    seconds: 2,
    then: %Text.WithCaps{
      do: %Say{
        speaker: :dog,
        text: "woof",
        then: %Wait{
          seconds: 10,
          then: %Say{
            speaker: :dog,
            text: "bark bark"
          }
        }
      }
      ,
      then: %GoWest{
        mover: :dog,
        distance: 7,
        then: %Wait{seconds: 2}
      }
    }
  }
}
```

Building Up & Tearing Down 🧱🧨

GenEffect

Building Up & Tearing Down 🧱🧨

GenEffect

```elixir
defmodule Time do
  use GenEffect
  # ...
  def handle_effect(%Wait{seconds: seconds}), do: Process.sleep(seconds)
end
```

Building Up & Tearing Down 🧱🧨

GenEffect

```elixir
defmodule Time do
  use GenEffect
  # ...
  def handle_effect(%Wait{seconds: seconds}), do: Process.sleep(seconds)
end
```

```elixir
defmodule Speaking do
  use GenEffect
  # ...
  def handle_effect(%Say{speaker: who, text: msg}), do: IO.puts("#{who} says #{msg}")
end
```

Building Up & Tearing Down 🧱🧨

GenEffect

```elixir
defmodule Time do
  use GenEffect
  # ...
  def handle_effect(%Wait{seconds: seconds}), do: Process.sleep(seconds)
end
```

```elixir
defmodule Speaking do
  use GenEffect
  # ...
  def handle_effect(%Say{speaker: who, text: msg}), do: IO.puts("#{who} says #{msg}")
end
```

```elixir
defmodule Text do
  use GenEffect
  # ...
  def handle_effect(%Capslock{on: true}, state) do
    IO.ANSI.capslock()
    %{state | caps: true}
  end
end
```

Building Up & Tearing Down 🧱🧨

Interpreter

Building Up & Tearing Down 🧱🧨

Interpreter

```
with_npc :dog do
  move(:north, 2)
  wait(2, :seconds)

  with_caps do
    say("woof")
    wait(10, :seconds)
    say("bark bark")
  end

  move(:west, 7)
  wait(2, :seconds)
end
|> run(Text)
|> run(Time)
|> run(Speaking)
```

Building Up & Tearing Down 🧱🧨

Interpreter

```
with_npc :dog do
  move(:north, 2)
  wait(2, :seconds)

  with_caps do
    say("woof")
    wait(10, :seconds)
    say("bark bark")
  end

  move(:west, 7)
  wait(2, :seconds)
end
|> run(Text)
|> run(Time)
|> run(Speaking)
```

Building Up & Tearing Down 🧱🧨

Interpreter

```
with_npc :dog do
  move(:north, 2)
  wait(2, :seconds)

  with_caps do
    say("woof")
    wait(10, :seconds)
    say("bark bark")
  end

  move(:west, 7)
  wait(2, :seconds)
end
|> run(Text)
|> run(Time)
|> run(Speaking)
```

Building Up & Tearing Down 🧱🧨

Interpreter

```
# ...
IO.ANSI.capslock()
Agent.set(pid, fn state -> %{state | caps: true} end)
IO.puts("woof")
Process.sleep(10)
IO.ANSI.capslock()
Agent.set(pid, fn state -> %{state | caps: false} end)
# ...
```

```
with_npc :dog do
  move(:north, 2)
  wait(2, :seconds)

  with_caps do
    say("woof")
    wait(10, :seconds)
    say("bark bark")
  end

  move(:west, 7)
  wait(2, :seconds)
end
|> run(Text)
|> run(Time)
|> run(Speaking)
```

Building Up & Tearing Down 🧱🧨

Interpreter

```elixir
# ...
IO.ANSI.capslock()
Agent.set(pid, fn state -> %{state | caps: true} end)
IO.puts("woof")
Process.sleep(10)
IO.ANSI.capslock()
Agent.set(pid, fn state -> %{state | caps: false} end)
# ...
```

```elixir
with_npc :dog do
  move(:north, 2)
  wait(2, :seconds)

  with_caps do
    say("woof")
    wait(10, :seconds)
    say("bark bark")
  end

  move(:west, 7)
  wait(2, :seconds)
end
|> run(Text)
|> run(Time)
|> run(Speaking)
```

Why not use a protocol?

Canonicity!

Domain & Denotation

Standard Vocabularies

🔠

Standard Vocabularies 🔠

What Is Common?

```
with_npc :dog do
  move(:north, 2)
  wait(2, :seconds)

  with_caps do
    say("woof")
    wait(10, :seconds)
    say("bark bark")
  end

  move(:west, 7)
  wait(2, :seconds)
end
```

```
defprotocol GameActions do
  @type direction :: :north | :south | :east | :west

  @spec move(t, direction(), non_neg_number()) :: t
  def move(entity, direction, paces)

  @spec speak(t, String.t()) :: t
  def speak(entity, message)

  @spec listen(t, t) :: {t, String.t()}
  def listen(entity, entity)
end
```

```
defprotocol ImageManipulation do
  @spec rotate(t, non_neg_integer()) :: t
  def rotate(image, degrees)

  @spec scale(t, integer()) :: t
  def scale(image, percentage)

  @spec translate(t, integer(), non_neg_integer()) :: t
  def translate(image, degrees, distance_in_pixels)
end
```

Standard Vocabularies 🔠

What Is Common?

Movement!

```elixir
with_npc :dog do
  move(:north, 2)
  wait(2, :seconds)

  with_caps do
    say("woof")
    wait(10, :seconds)
    say("bark bark")
  end

  move(:west, 7)
  wait(2, :seconds)
end
```

```elixir
defprotocol GameActions do
  @type direction :: :north | :south | :east | :west

  @spec move(t, direction(), non_neg_number()) :: t
  def move(entity, direction, paces)

  @spec speak(t, String.t()) :: t
  def speak(entity, message)

  @spec listen(t, t) :: {t, String.t()}
  def listen(entity, entity)
end
```

```elixir
defprotocol ImageManipulation do
  @spec rotate(t, non_neg_integer()) :: t
  def rotate(image, degrees)

  @spec scale(t, integer()) :: t
  def scale(image, percentage)

  @spec translate(t, integer(), non_neg_integer()) :: t
  def translate(image, degrees, distance_in_pixels)
end
```

Standard Vocabularies 🔤

One Level Down

Standard Vocabularies 🔠

One Level Down

```
defprotocol GraphRouting do
  def adjacentcies(graph, node)
  def move(graph, node, adjacency)
end
```

```
defprotocol Geometric do
  def scale(object, factor)
  def translate(object, angle, distance)
  def rotate(object, angle)
  def shear(object, delta_v, delta_h)
end
```

Standard Vocabularies 🔤

One Level Down

```elixir
defprotocol GraphRouting do
  def adjacentcies(graph, node)
  def move(graph, node, adjacency)
end
```

```elixir
defprotocol PartialOrder do
  def compare(a, b)
end
```

```elixir
defprotocol Geometric do
  def scale(object, factor)
  def translate(object, angle, distance)
  def rotate(object, angle)
  def shear(object, delta_v, delta_h)
end
```

```elixir
defprotocol Setlike do
  def union(a, b)
  def intersect(a, b)
end
```

Standard Vocabularies 🔠

What's Important About Laws?

```elixir
defprotocol PartialOrder do
  def compare(a, b)
end
```

Standard Vocabularies 🔠

What's Important About Laws?

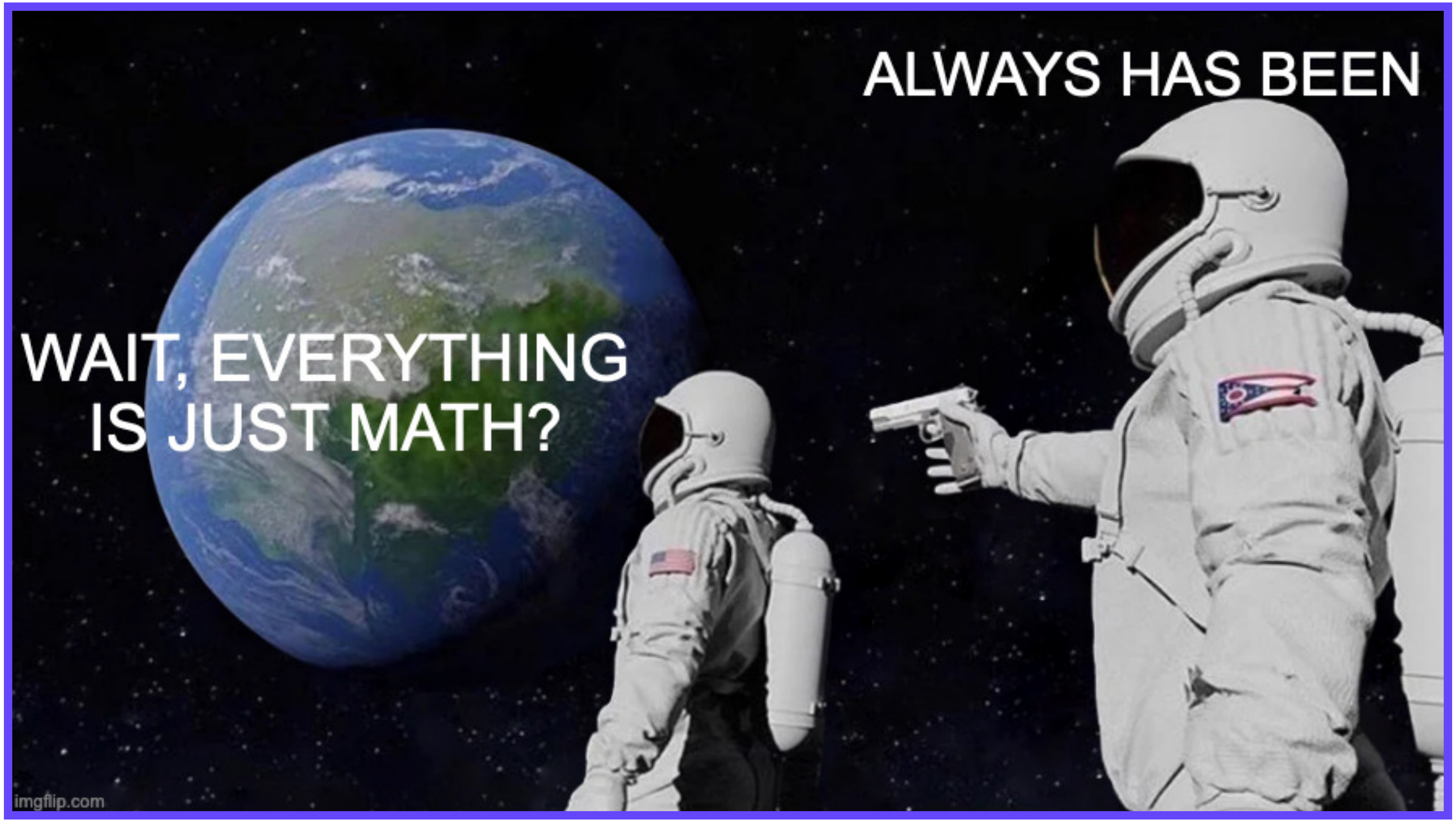
```
defprotocol PartialOrder do
  def compare(a, b)
end
```

```
# a relates to itself
a <= a

# equal elements don't preceed themselves
if a <= b && b <= a, then: a == b

# transitive
if a <= b && b <= c, then: a <= c
```

Consistent in all contexts

Standard Vocabularies 🔤

Denotative Redux



Standard Vocabularies 🔠

Denotative Redux

Standard Vocabularies 🔤

Denotative Redux

We are only drawing a most important distinction — between ***discovering something and inventing something***. But mathematicians make the most important discoveries.

– Alan Turing, via Wittgenstein's Lectures on the Foundations of Mathematics

Standard Vocabularies 🔤

Safety First

Purify Your Effects

🦺⛑️🪄

Purify Your Effects 🦺⛑️🪄

Have no truck with the
grubby compromises
of imperative programming

— Simon Peyton Jones

Purify Your Effects 🦺⛑️🪄

Tools Down

Purify Your Effects 🦺⛑️🪄

Tools Down

Elixir is surprisingly ***imperative,***
yet gives you ***all the tools*** of equational reasoning
...so let's use them!

Purify Your Effects 🦺⛑️🪄

Description vs Invocation

Description vs Invocation

Impure functions produce ***side effects***
Pure functions manipulate ***data***

Side effects → ***managed*** effects

Purify Your Effects 🦺⛑️🪄

Description vs Invocation

Impure functions produce ***side effects***
Pure functions manipulate ***data***

Side effects → ***managed*** effects

`function`

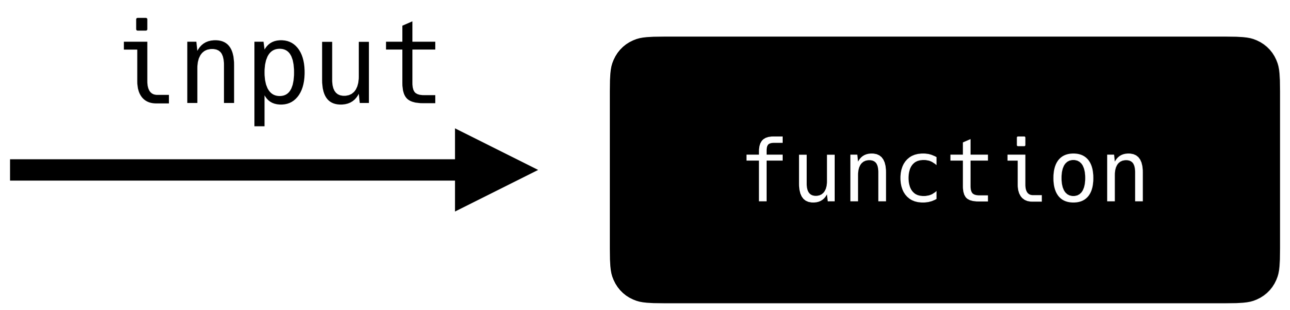
Purify Your Effects 🦺⛑️🪄

Description vs Invocation

Impure functions produce ***side effects***

Pure functions manipulate ***data***

Side effects → ***managed*** effects

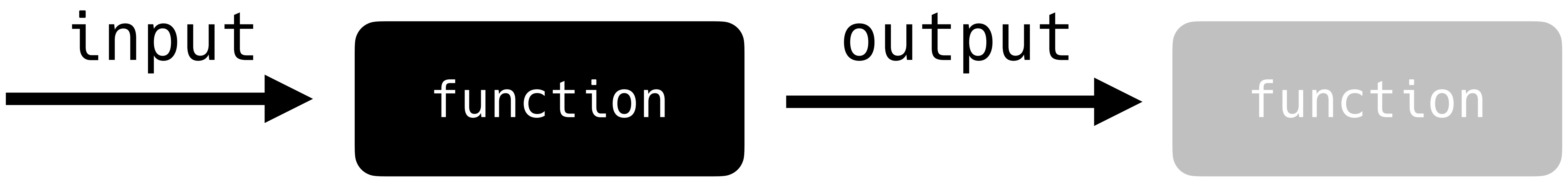


Purify Your Effects 🦺⛑️🪄

Description vs Invocation

Impure functions produce ***side effects***
Pure functions manipulate ***data***

Side effects → ***managed*** effects

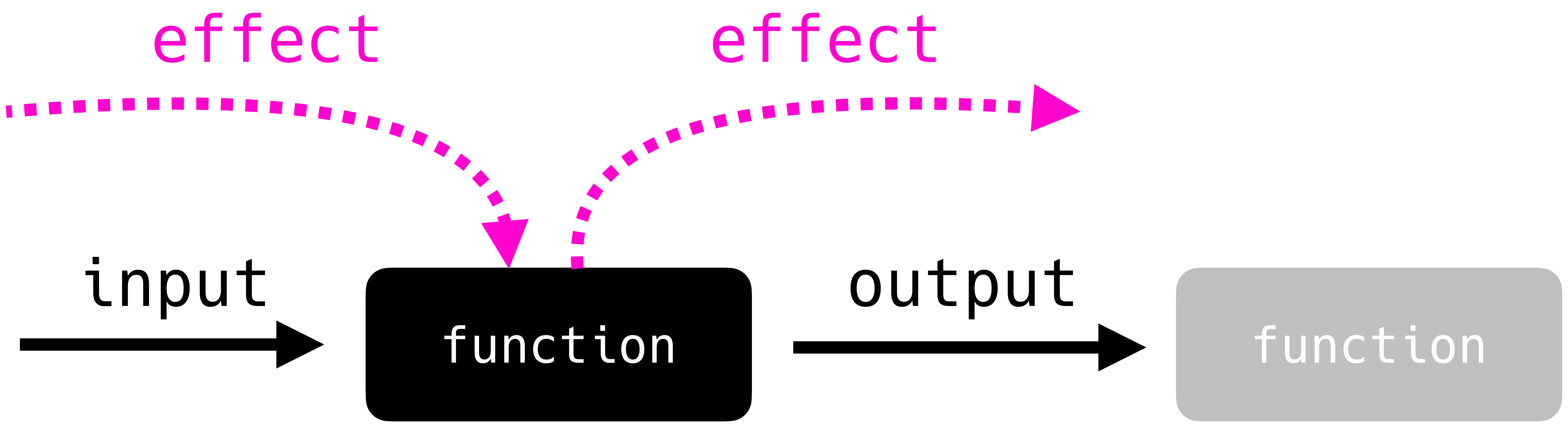


Purify Your Effects 🦺⛑️🪄

Description vs Invocation

Impure functions produce ***side effects***
Pure functions manipulate ***data***

Side effects → ***managed*** effects

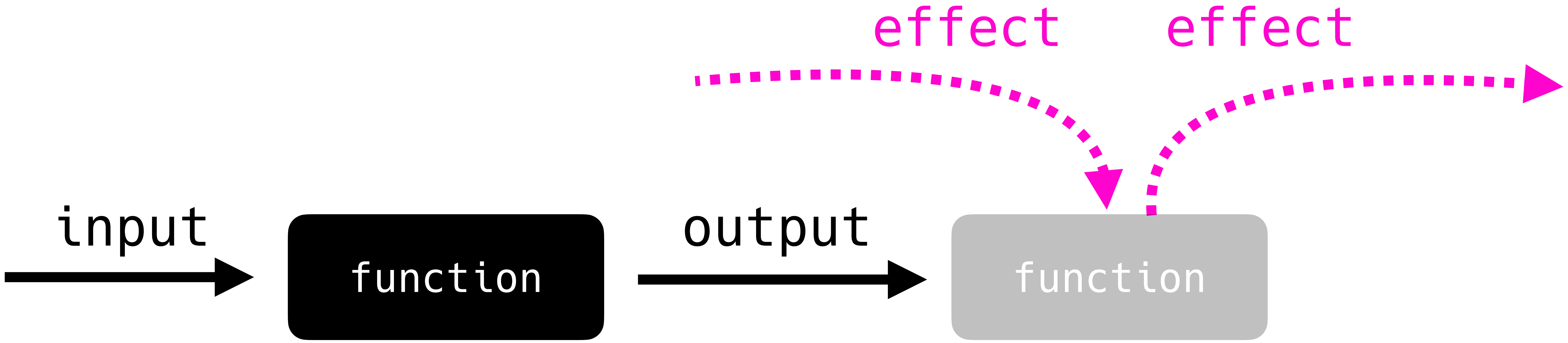


Purify Your Effects 🦺⛑️🪄

Description vs Invocation

Impure functions produce ***side effects***
Pure functions manipulate ***data***

Side effects → ***managed*** effects



Purify Your Effects 🦺⛑️🪄

4-Layer Architecture

Purify Your Effects 🦺⛑️🪄

4-Layer Architecture

Imperative

Managed Effects

Pure Functions

Data

Purify Your Effects 🦺⛑️🪄

4-Layer Architecture

💥 ***Action***

Imperative

Managed Effects

Pure Functions

Data

💡 ***Information***

Purify Your Effects 🦺⛑️🪄

4-Layer Architecture

💥 **Action** — **Unstable** 🌋

- Imperative
- Managed Effects
- Pure Functions
- Data

💡 **Information** — **Stable** 🏔️

Purify Your Effects 🦺⛑️🪄

Purified Actions

```
# ...
IO.ANSI.capslock()
Agent.set(pid, fn state -> %{state | caps: true} end)
IO.puts("woof")
Process.sleep(10)
IO.ANSI.capslock()
Agent.set(pid, fn state -> %{state | caps: false} end)
# ...
```

```
convert [Text, Time, Speaking] do
  with_npc :dog do
    move(:north, 2)
    wait(2, :seconds)

    with_caps do
      say("woof")
      wait(10, :seconds)
      say("bark bark")
    end

    move(:west, 7)
    wait(2, :seconds)
  end
end
```

Well Behaved Models

Testing Minus the Teeth

Testing Minus the Teeth 🧪

Faking Without Mocks

Testing Minus the Teeth 🧪

Faking Without Mocks

- Inspect the pure data!
- Have tests write to a list as they run
- Fake databases with maps
- Fake sending email with logs

Testing Minus the Teeth 🧪

Faking Without Mocks

- Inspect the pure data!
- Have tests write to a list as they run
- Fake databases with maps
- Fake sending email with logs

```
with_npc :dog do
  move(:north, 2)
  wait(2, :seconds)

  with_caps do
    say("woof")
    wait(10, :seconds)
    say("bark bark")
  end

  move(:west, 7)
  wait(2, :seconds)
end
|> run(Text)
|> run(Time)
|> run(Speaking)
```

Testing Minus the Teeth 🧪

Faking Without Mocks

- Inspect the pure data!
- Have tests write to a list as they run
- Fake databases with maps
- Fake sending email with logs

```
%GoNorth{
  mover: :dog,
  distance: 2,
  then: %Wait{
    seconds: 2,
    then: %Text.WithCaps{
      do: %Say{
            speaker: :dog,
            text: "woof",
            then: %Wait{
              seconds: 10,
              then: %Say{
                speaker: :dog,
                text: "bark bark"
              }
            }
      },
      then: %GoWest{
        mover: :dog,
        distance: 7,
        then: %Wait{seconds: 2}
      }
    }
  }
}
```

```
with_npc :dog do
  move(:north, 2)
  wait(2, :seconds)

  with_caps do
    say("woof")
    wait(10, :seconds)
    say("bark bark")
  end

  move(:west, 7)
  wait(2, :seconds)
end
|> run(Text)
|> run(Time)
|> run(Speaking)
```

Testing Minus the Teeth 🧪

Faking Without Mocks

- Inspect the pure data!
- Have tests write to a list as they run
- Fake databases with maps
- Fake sending email with logs

```
%GoNorth{
  mover: :dog,
  distance: 2,
  then: %Wait{
    seconds: 2,
    then: %Text.WithCaps{
      do: %Say{
            speaker: :dog,
            text: "woof",
            then: %Wait{
              seconds: 10,
              then: %Say{
                speaker: :dog,
                text: "bark bark"
              }
            }
      },
      then: %GoWest{
        mover: :dog,
        distance: 7,
        then: %Wait{seconds: 2}
      }
    }
  }
}
```

```
with_npc :dog do
  move(:north, 2)
  wait(2, :seconds)

  with_caps do
    say("woof")
    wait(10, :seconds)
    say("bark bark")
  end

  move(:west, 7)
  wait(2, :seconds)
end
|> run(Text)
|> run(Time)
|> run(Speaking)
```

Control Dominator Take the Wheel

Implicit Parallelism

Implicit Parallelism 🥄

Humans are Terrible at Concurrency 😱

Implicit Parallelism 🥄

Humans are Terrible at Concurrency 😱

The main way of dealing with concurrency has been ***reduced to sequential reasoning*** [...]
it requires to cope with many possible, ***unpredictable*** behaviors of process, and the communication media

Everything is ***NOT reducible*** to sequential thinking

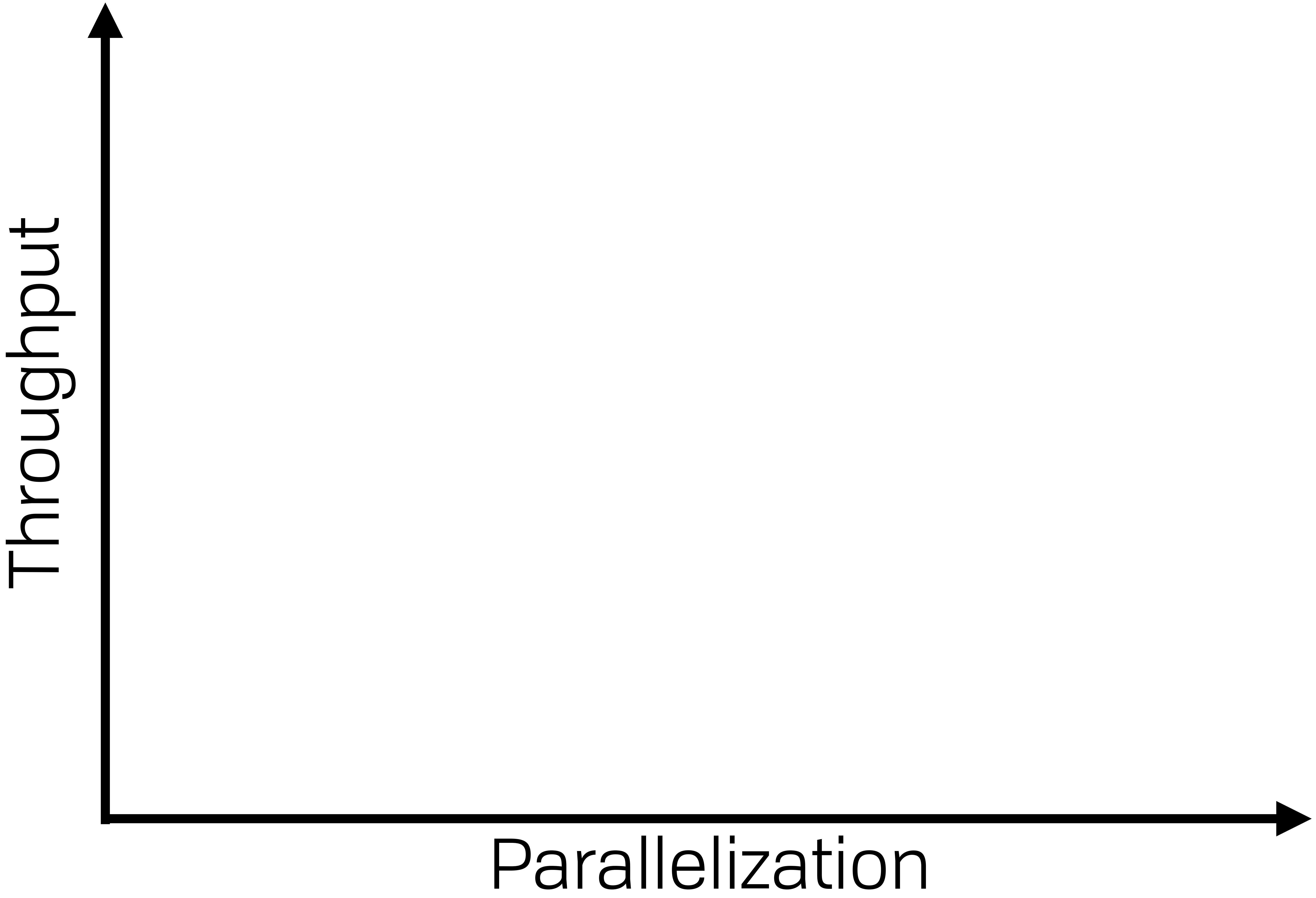
— Sergio Rajsbaum & Michel Raynal, 60 Years of Mastering Concurrency Through Sequential Thinking

Implicit Parallelism 🥄

Coordination Costs

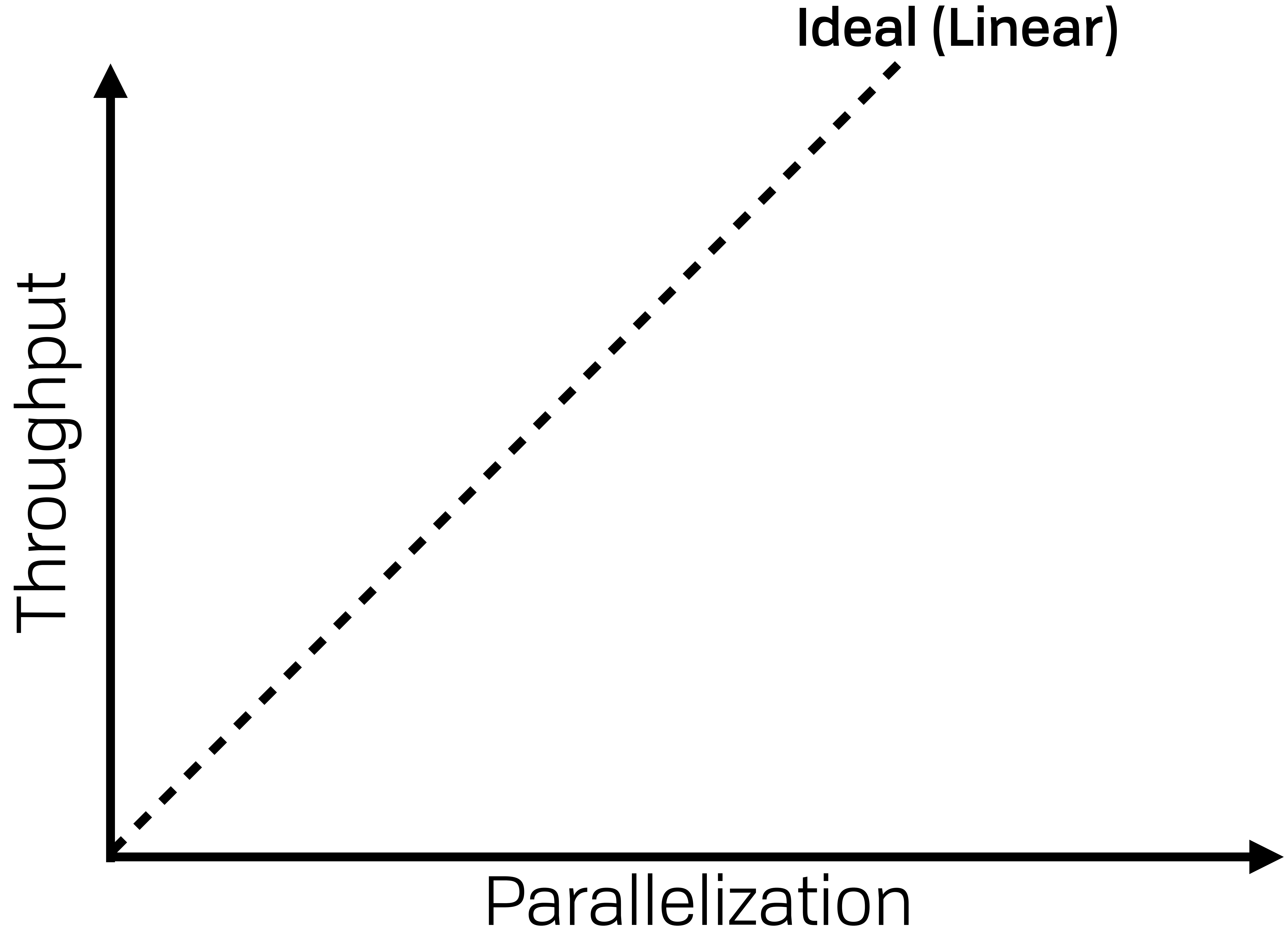
Implicit Parallelism 🥄

Coordination Costs



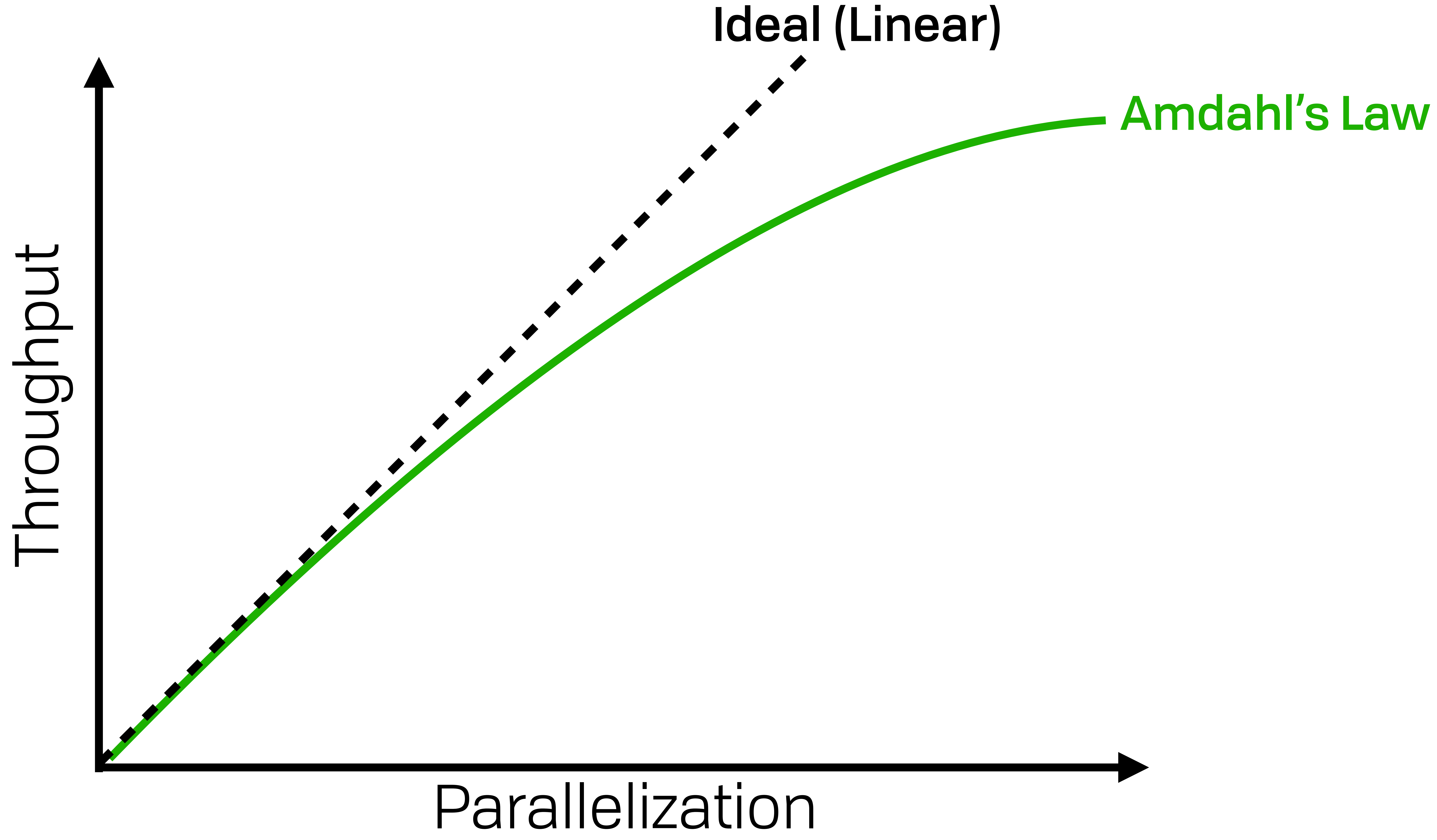
Implicit Parallelism 🥄

Coordination Costs



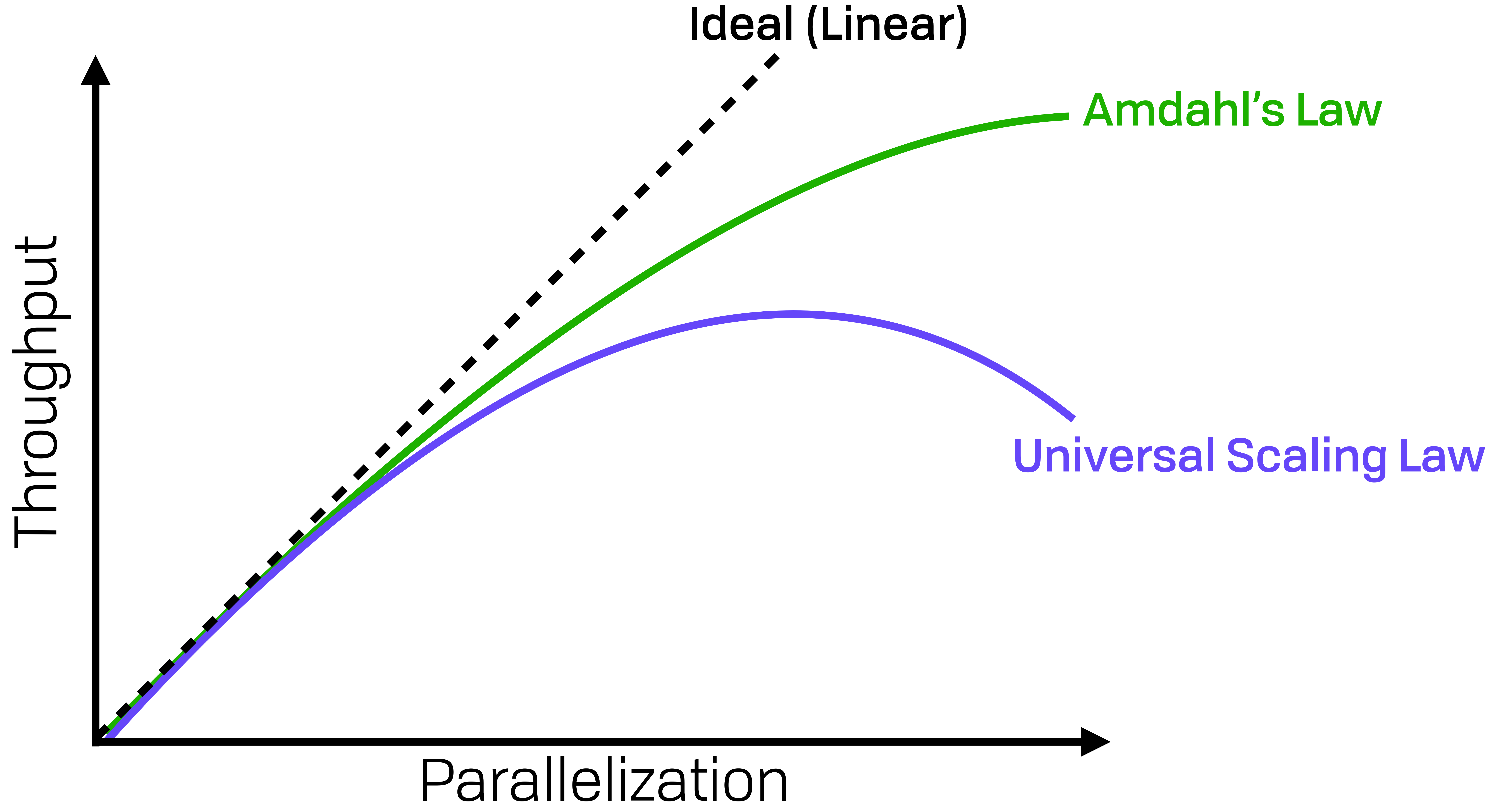
Implicit Parallelism 🥄

Coordination Costs



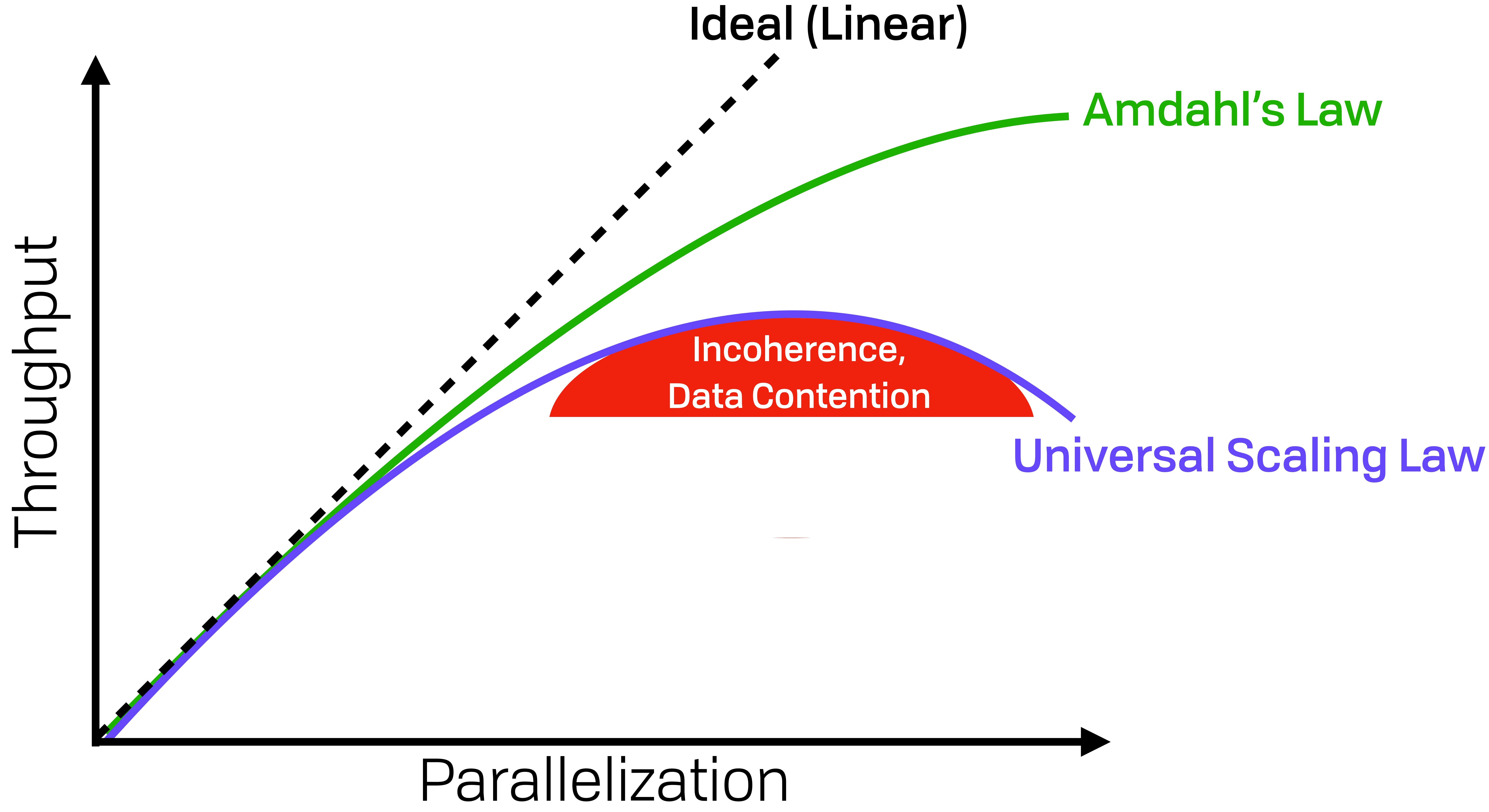
Implicit Parallelism 🥄

Coordination Costs



Implicit Parallelism 🥄

Coordination Costs



Implicit Parallelism 🥄

Confluence / Church-Rosser

Implicit Parallelism 🥄

Confluence / Church-Rosser

```
foo(bar(42), baz(97))
```

Implicit Parallelism 🥄

Confluence / Church-Rosser

```
foo(bar(42), baz(97))
```

```
y = baz(86)
x = bar(42)
foo(x, y)
```

```
x = bar(42)
y = baz(86)
foo(x, y)
```

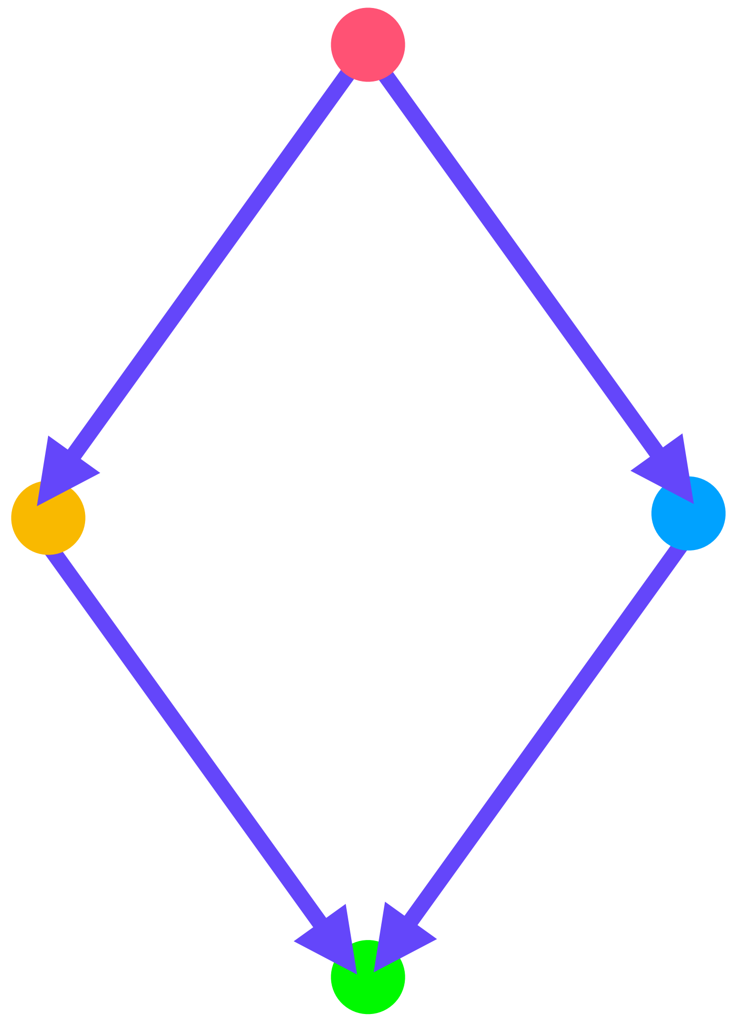
Implicit Parallelism 🥄

Confluence / Church-Rosser

```
foo(bar(42), baz(97))
```

```
y = baz(86)
x = bar(42)
foo(x, y)
```

```
x = bar(42)
y = baz(86)
foo(x, y)
```



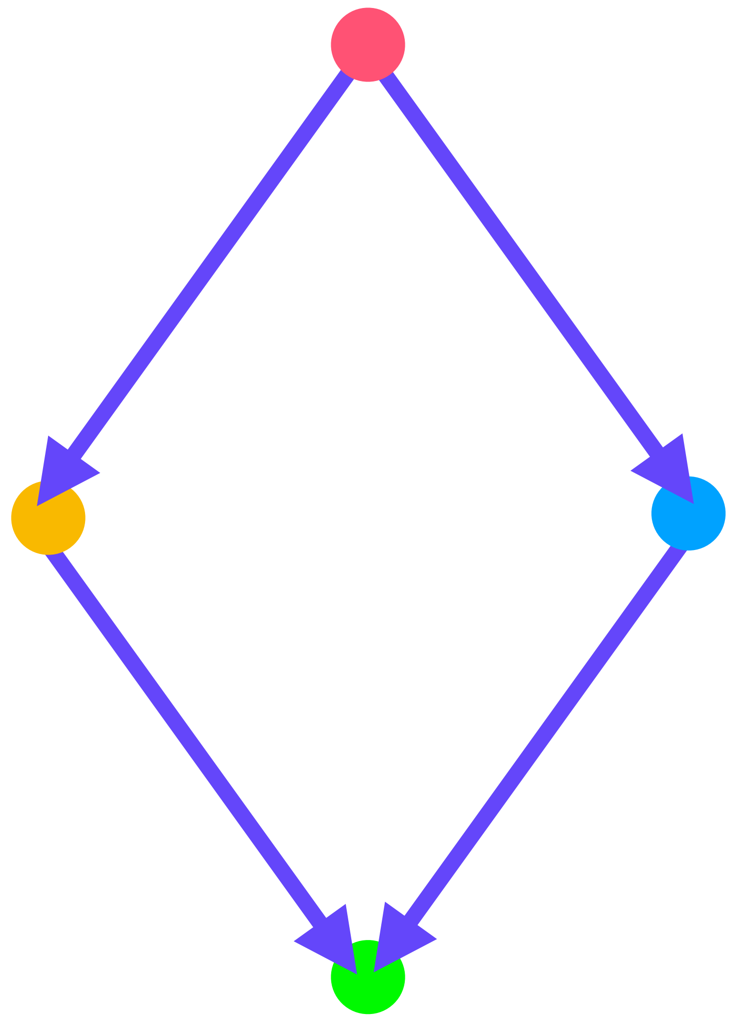
Implicit Parallelism 🥄

Confluence / Church-Rosser

```
foo(bar(42), baz(97))
```

```
y = baz(86)
x = bar(42)
foo(x, y)
```

```
x = bar(42)
y = baz(86)
foo(x, y)
```



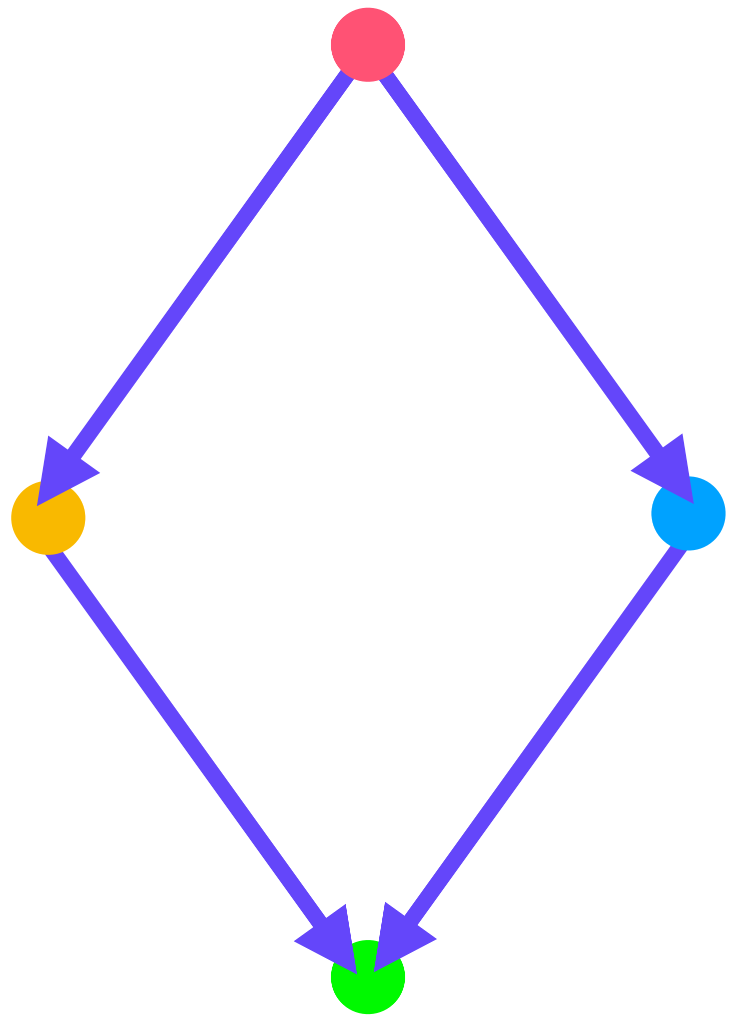
Implicit Parallelism 🥄

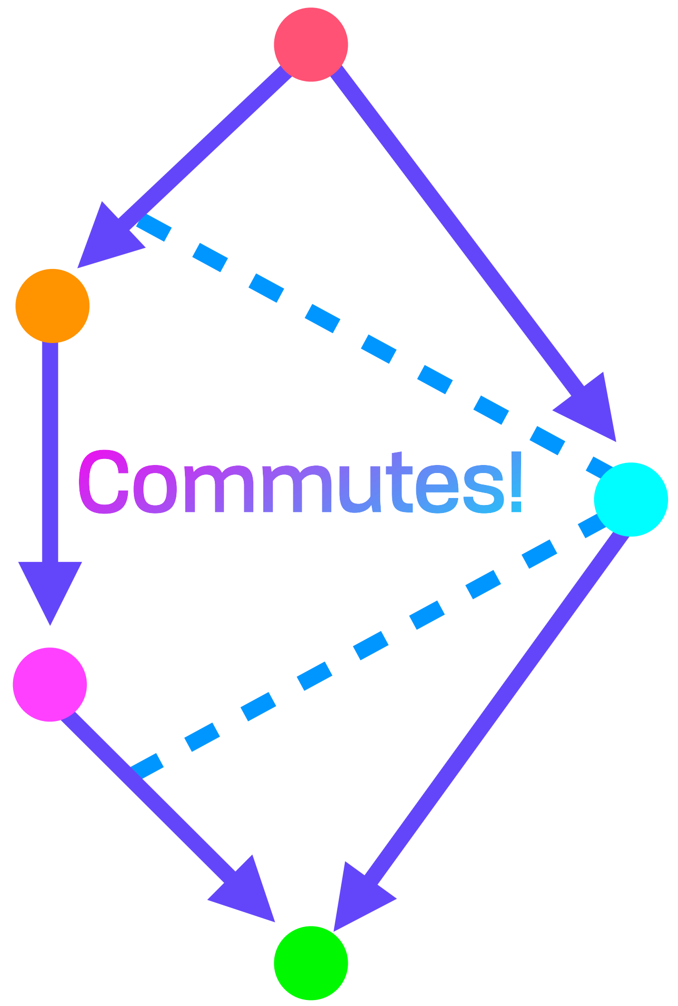
Confluence / Church-Rosser

```
foo(bar(42), baz(97))
```

```
y = baz(86)
x = bar(42)
foo(x, y)
```

```
x = bar(42)
y = baz(86)
foo(x, y)
```





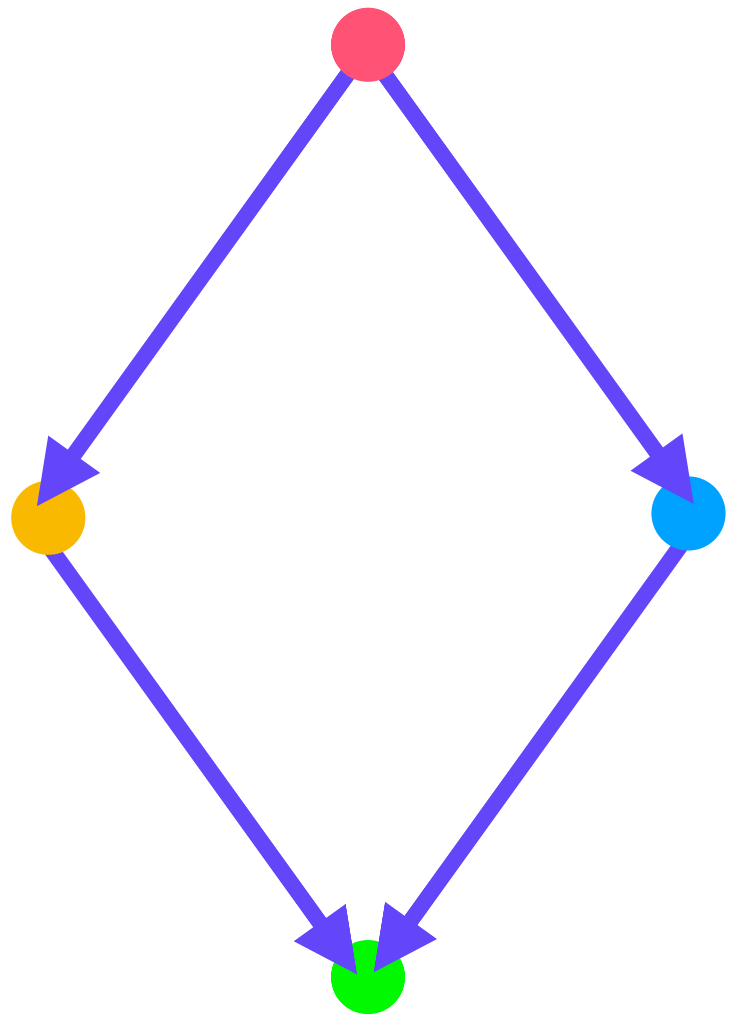
Implicit Parallelism 🥄

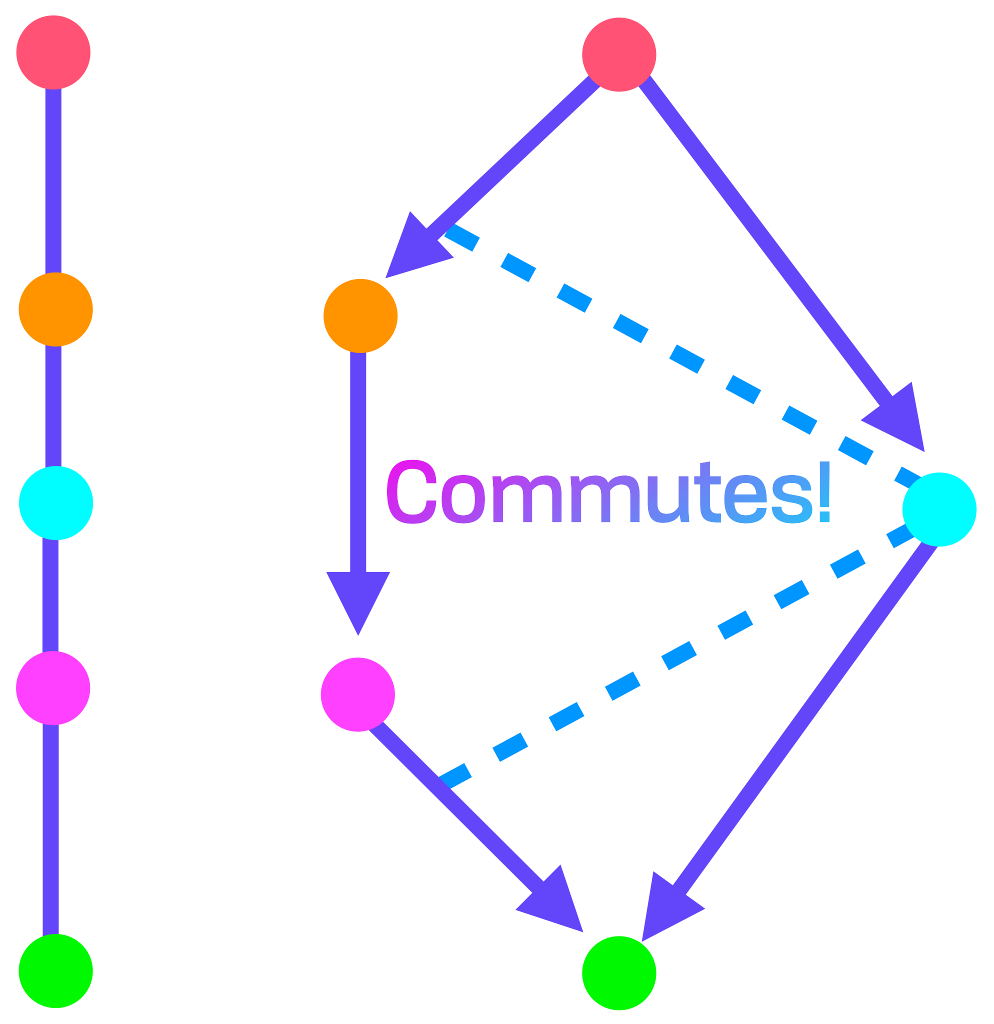
Confluence / Church-Rosser

```
foo(bar(42), baz(97))
```

```
y = baz(86)
x = bar(42)
foo(x, y)
```

```
x = bar(42)
y = baz(86)
foo(x, y)
```





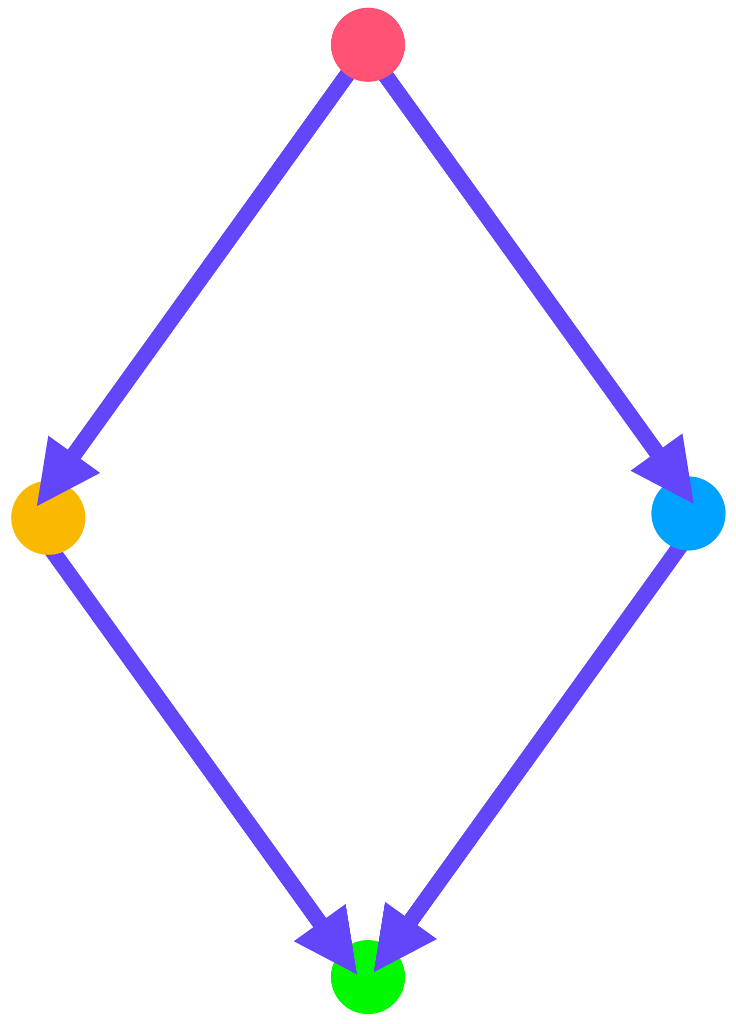
Implicit Parallelism 🥄

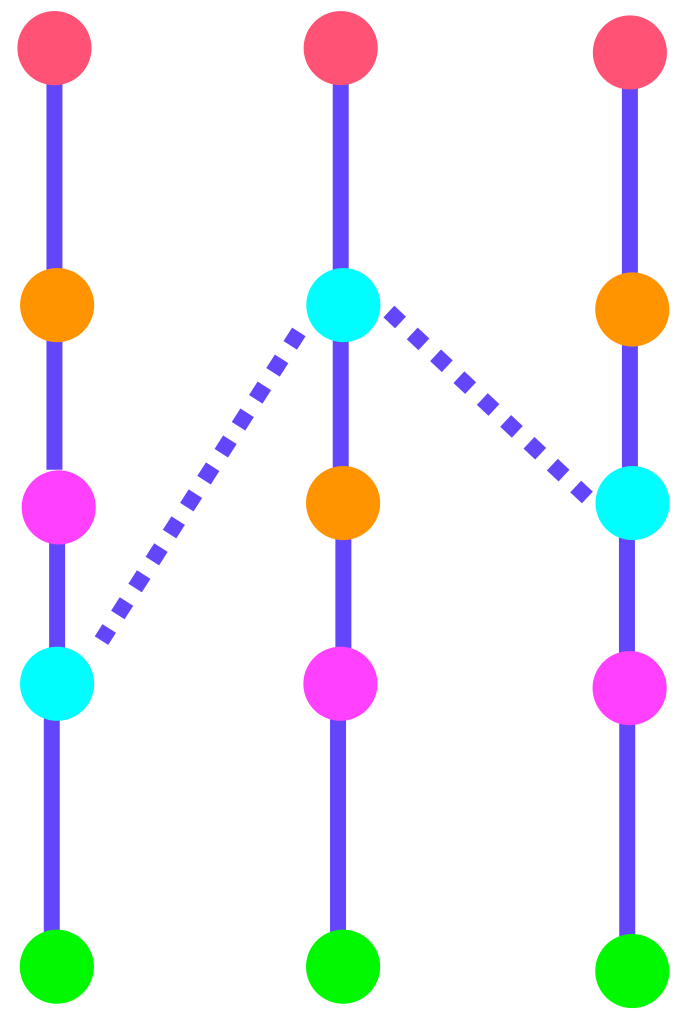
Confluence / Church-Rosser

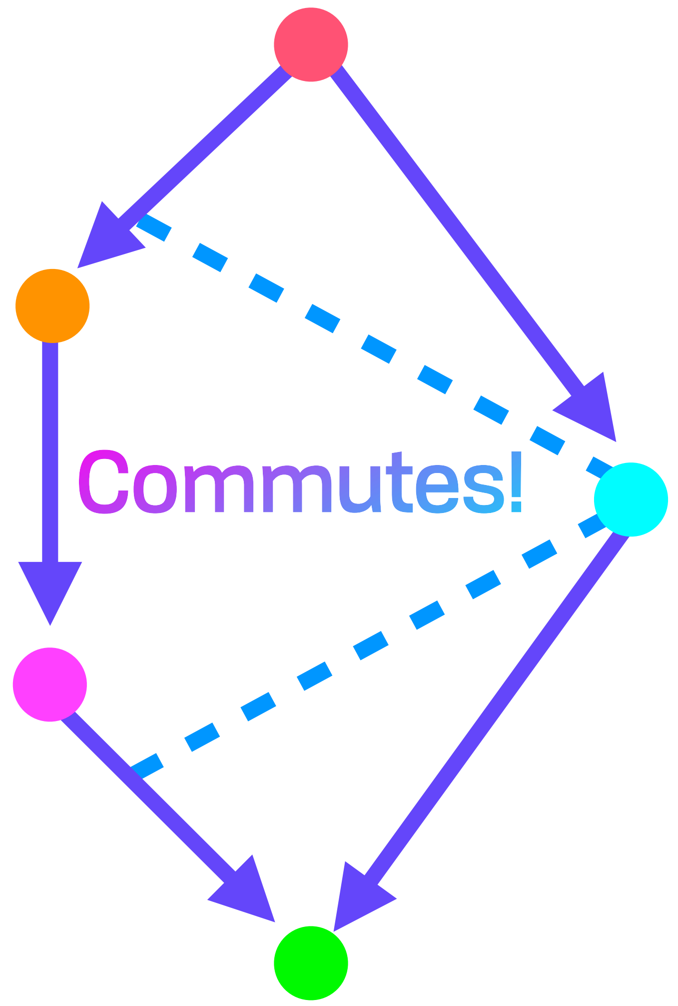
```
foo(bar(42), baz(97))
```

```
y = baz(86)
x = bar(42)
foo(x, y)
```

```
x = bar(42)
y = baz(86)
foo(x, y)
```







Implicit Parallelism 🥄

Dependency Analysis

Implicit Parallelism 🥄

Dependency Analysis

```
recipient = DB.get(pid, user: 42, field: :email)
msg = IO.gets("What is your message?\n")
msgUpcase = String.uppercase(msg)
receipt = Email.send(msg, to: recipient)
today = Date.utc_today()
DB.insert(pid, "emails", to: recipient, msg: msg, date: today)
```

Implicit Parallelism 🥄

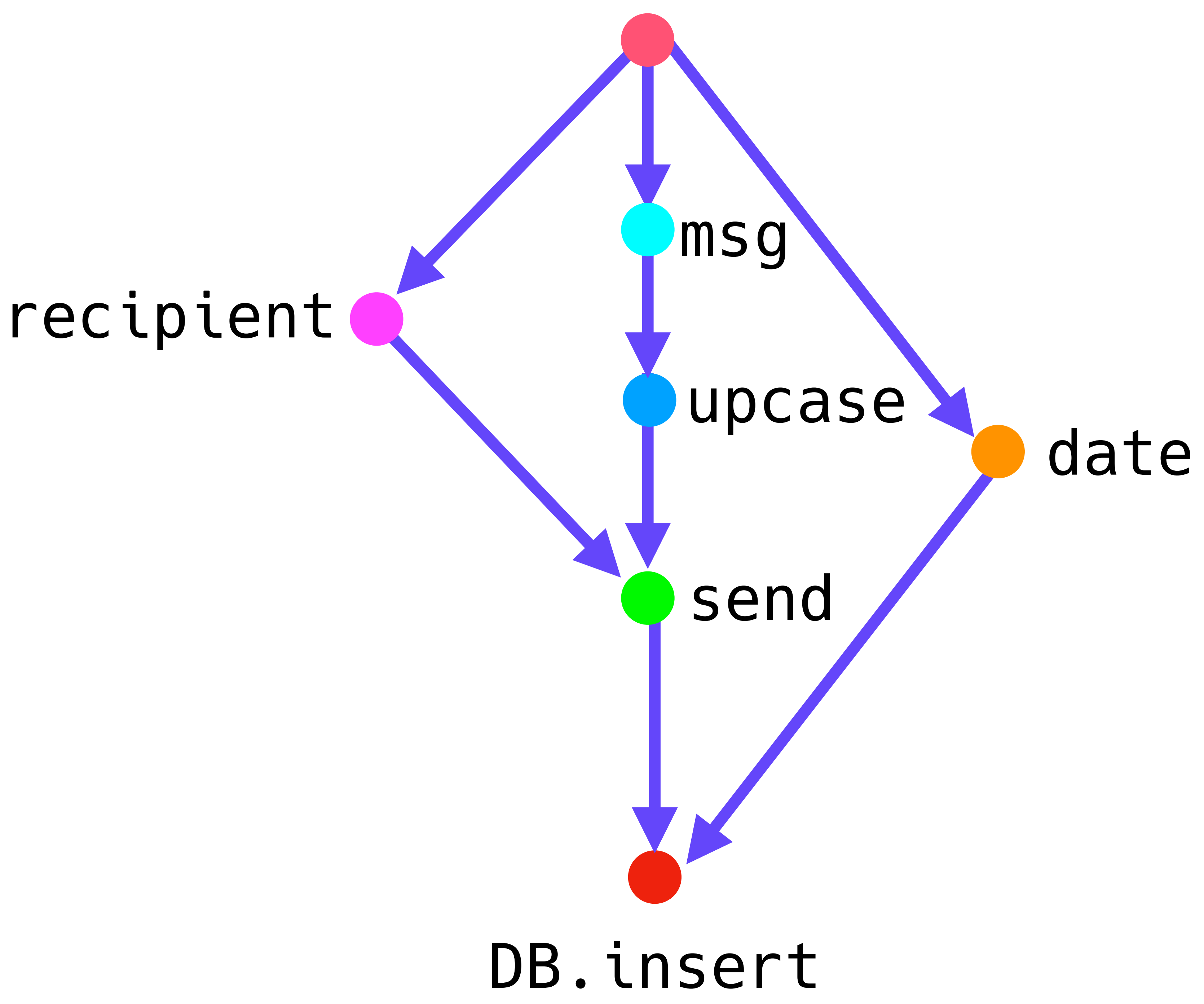
Dependency Analysis

```
recipient = DB.get(pid, user: 42, field: :email)
msg = IO.gets("What is your message?\n")
msgUpcase = String.uppercase(msg)
receipt = Email.send(msg, to: recipient)
today = Date.utc_today()
DB.insert(pid, "emails", to: recipient, msg: msg, date: today)
```

Implicit Parallelism 🥄

Dependency Analysis

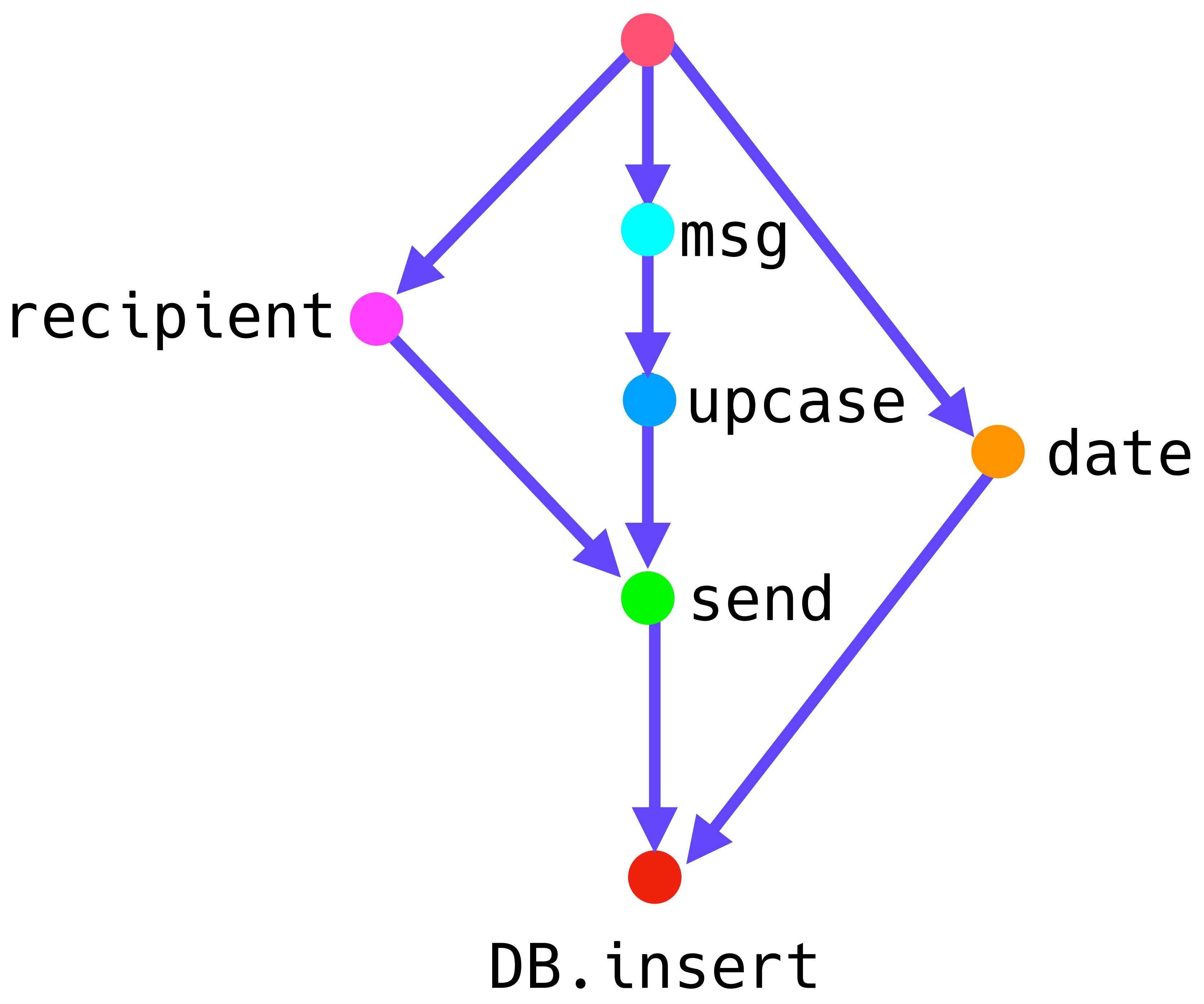
```
recipient = DB.get(pid, user: 42, field: :email)
msg = IO.gets("What is your message?\n")
msgUpcase = String.uppercase(msg)
receipt = Email.send(msg, to: recipient)
today = Date.utc_today()
DB.insert(pid, "emails", to: recipient, msg: msg, date: today)
```



Implicit Parallelism 🥄

Dependency Analysis

```
recipient = DB.get(pid, user: 42, field: :email)
msg = IO.gets("What is your message?\n")
msgUpcase = String.uppercase(msg)
receipt = Email.send(msg, to: recipient)
today = Date.utc_today()
DB.insert(pid, "emails", to: recipient, msg: msg, date: today)
```

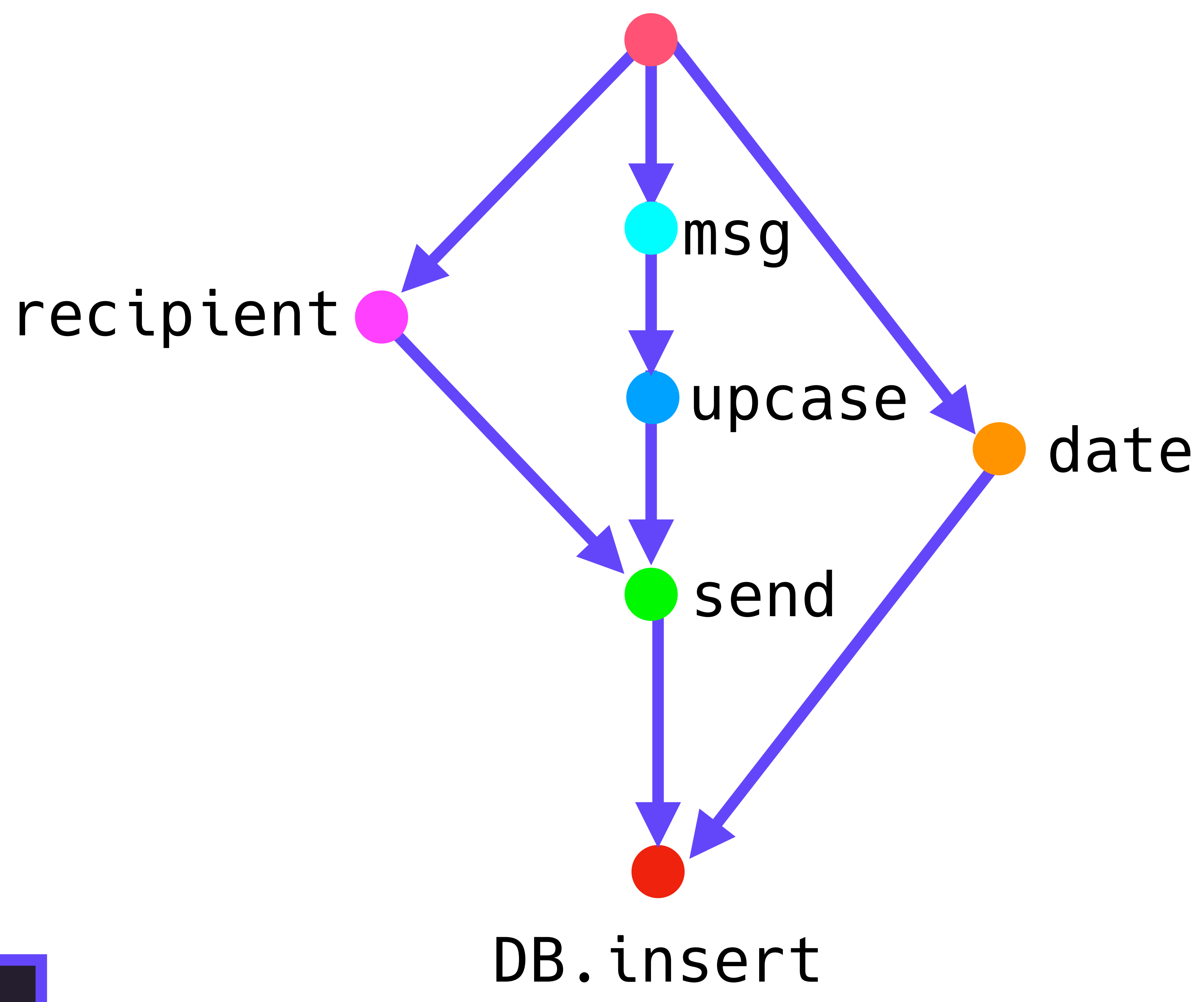



```
Enum.map([1,2,3], fn item ->
  Process.sleep(1000)
  x * 10
end)
```

Implicit Parallelism 🥄

Dependency Analysis

```
recipient = DB.get(pid, user: 42, field: :email)
msg = IO.gets("What is your message?\n")
msgUpcase = String.uppercase(msg)
receipt = Email.send(msg, to: recipient)
today = Date.utc_today()
DB.insert(pid, "emails", to: recipient, msg: msg, date: today)
```



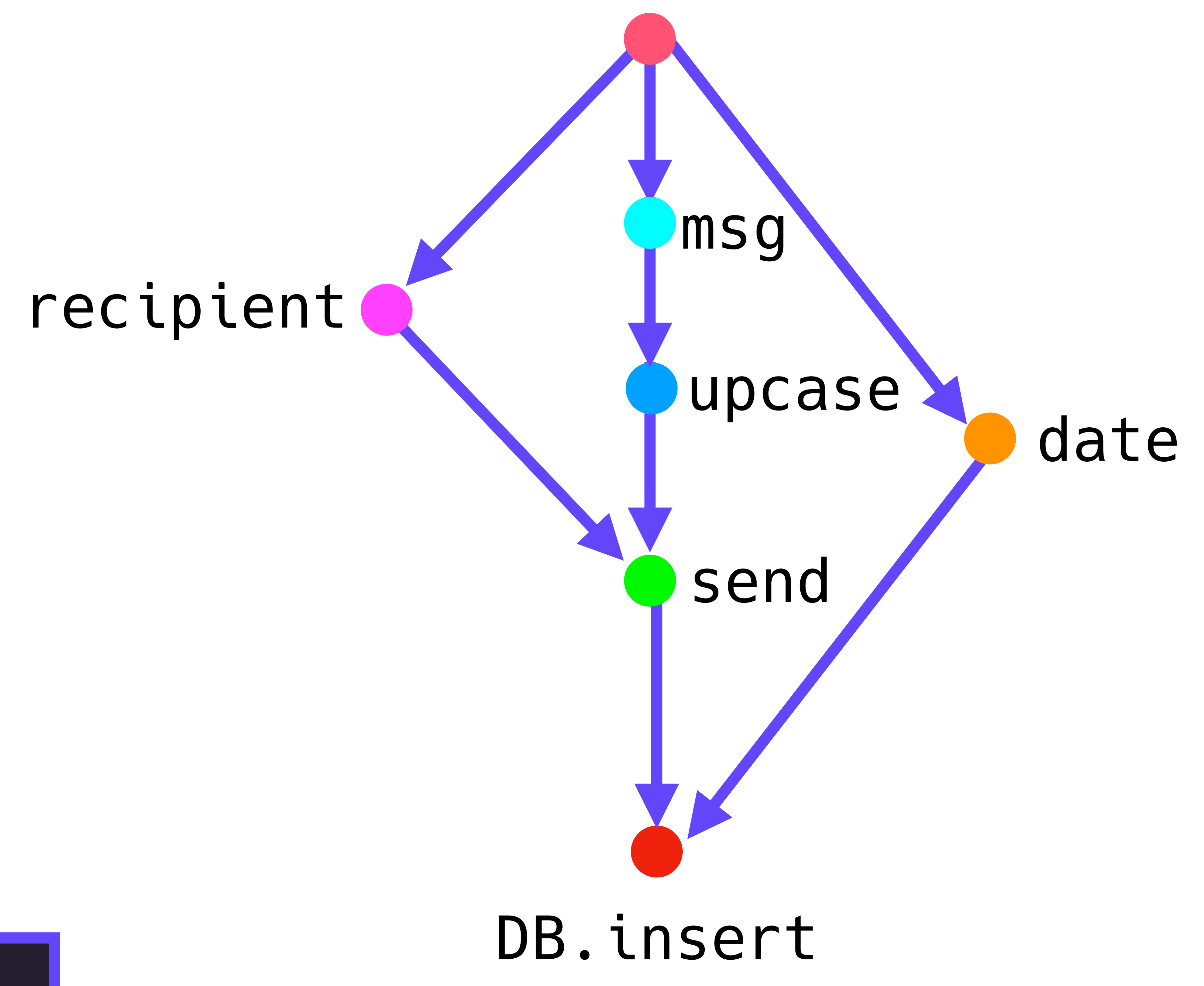

```
Enum.map([1,2,3], fn item ->
  Process.sleep(1000)
  x * 10
end)
```

```
[1,2,3]
|> Enum.map(fn item ->
    Task.async(fn ->
      Process.sleep(1000)
      x * 10
    end)
end)
|> Enum.map(&Task.await/1)
```

Implicit Parallelism 🥄

Dependency Analysis

```elixir
recipient = DB.get(pid, user: 42, field: :email)
msg = IO.gets("What is your message?\n")
msgUpcase = String.uppercase(msg)
receipt = Email.send(msg, to: recipient)
today = Date.utc_today()
DB.insert(pid, "emails", to: recipient, msg: msg, date: today)
```




```elixir
Enum.map([1,2,3], fn item ->
  Process.sleep(1000)
  x * 10
end)
```

```elixir
[1,2,3]
|> Enum.map(fn item ->
    Task.async(fn ->
      Process.sleep(1000)
      x * 10
    end)
end)
|> Enum.map(&Task.await/1)
```

```elixir
[1,2,3]
|> Witchcraft.async_map(fn item ->
  Process.sleep(1000)
  x * 10
end)
```

Implicit Parallelism 🥄

Actor Problem: Data Locality Coordination

Implicit Parallelism 🥄

Actor Problem: Data Locality Coordination

👨‍🎤 👩‍🎤 🧑‍🎤

Implicit Parallelism 🥄

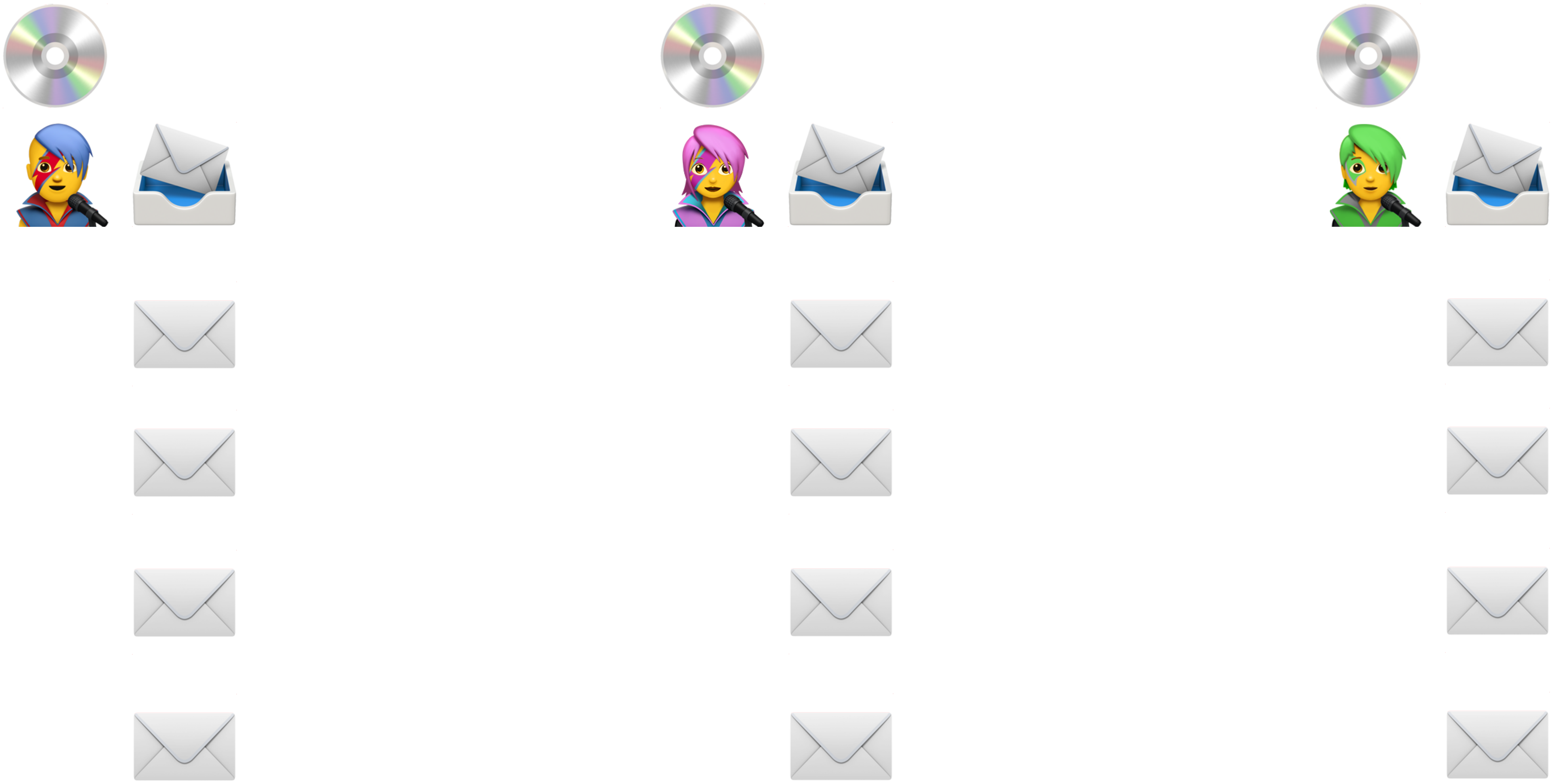
Actor Problem: Data Locality Coordination

Implicit Parallelism 🥄

Actor Problem: Data Locality Coordination

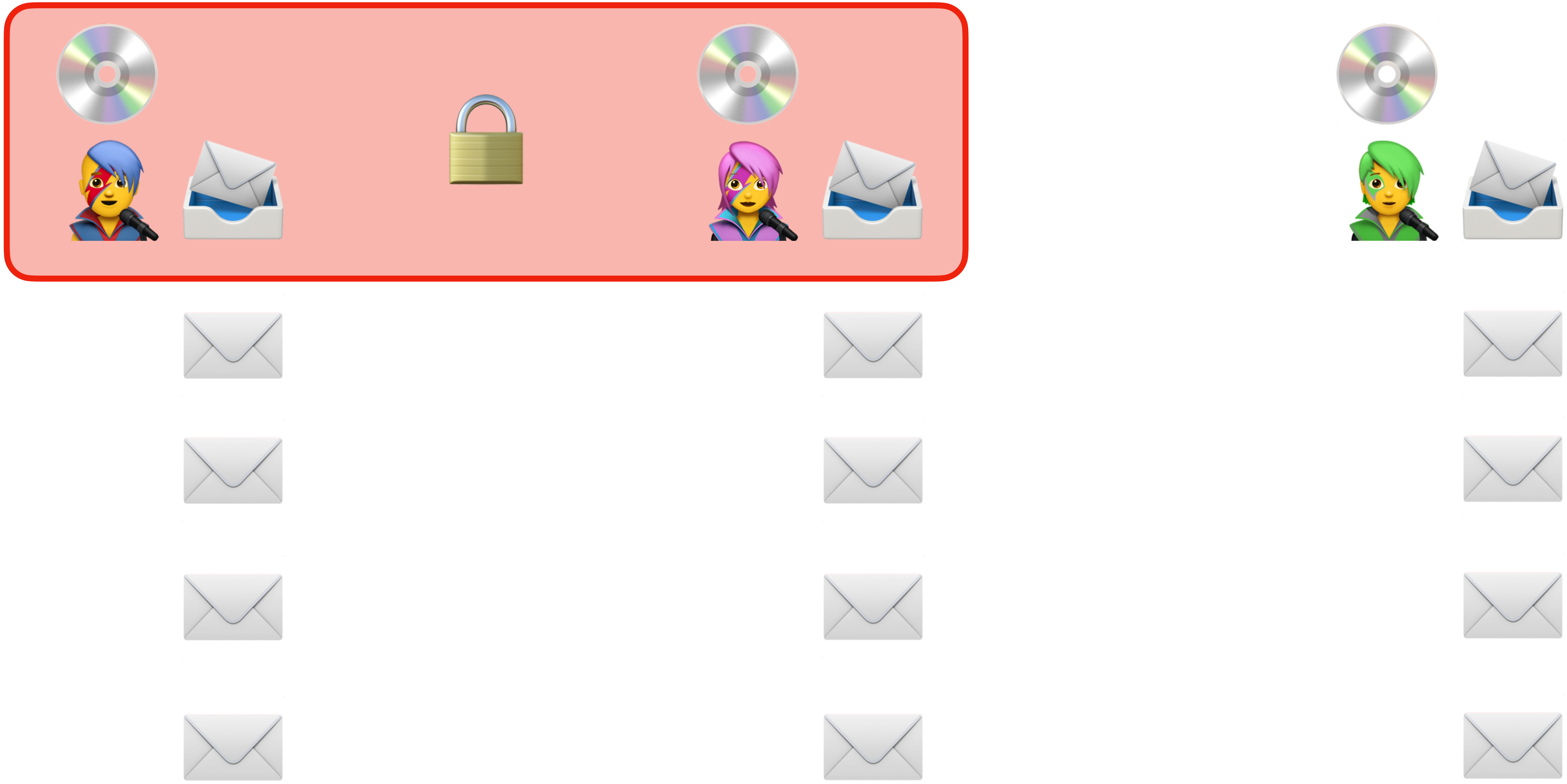
Implicit Parallelism 🥄

Actor Problem: Data Locality Coordination



Implicit Parallelism 🥄

Actor Problem: Data Locality Coordination

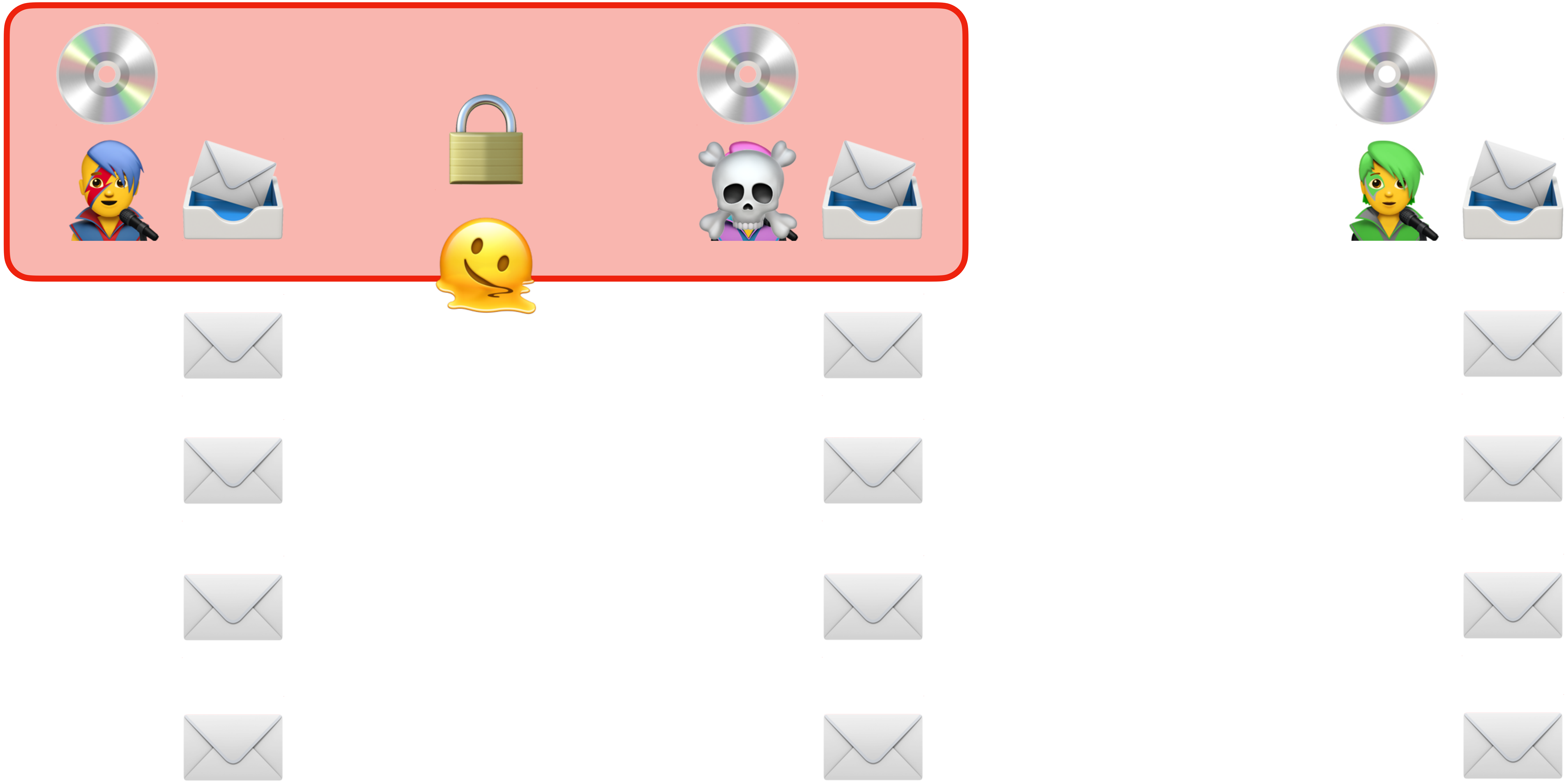


Implicit Parallelism 🥄

Actor Problem: Data Locality Coordination

Implicit Parallelism

Actor Problem: Data Locality Coordination



Purify Your Effects 🦺⛑️🪄

Optimistic Concurrency: STM

https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

Purify Your Effects 🦺⛑️🪄

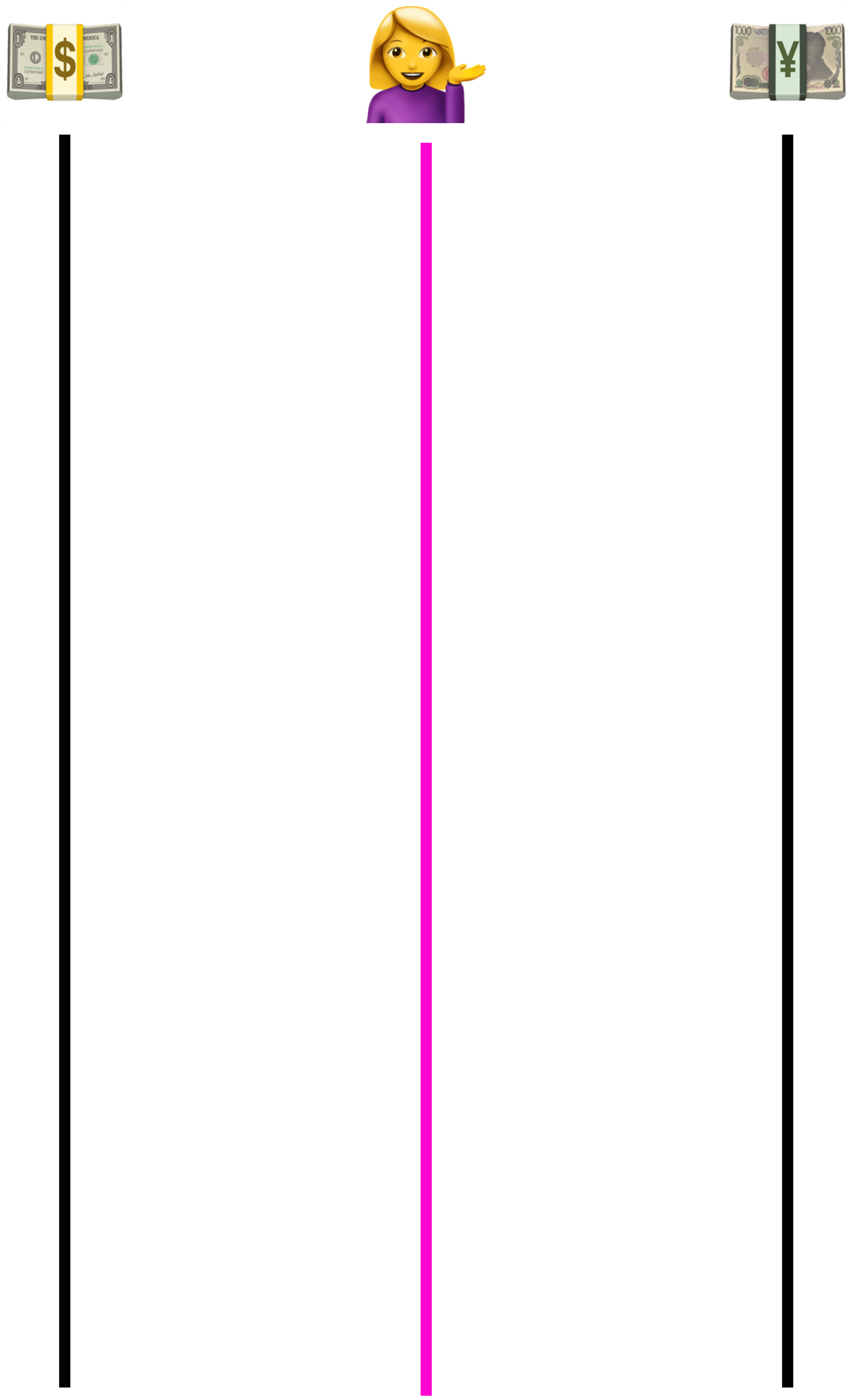
Optimistic Concurrency: STM

💵 💁‍♀️ 💴

https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

Purify Your Effects 🦺⛑️🪄

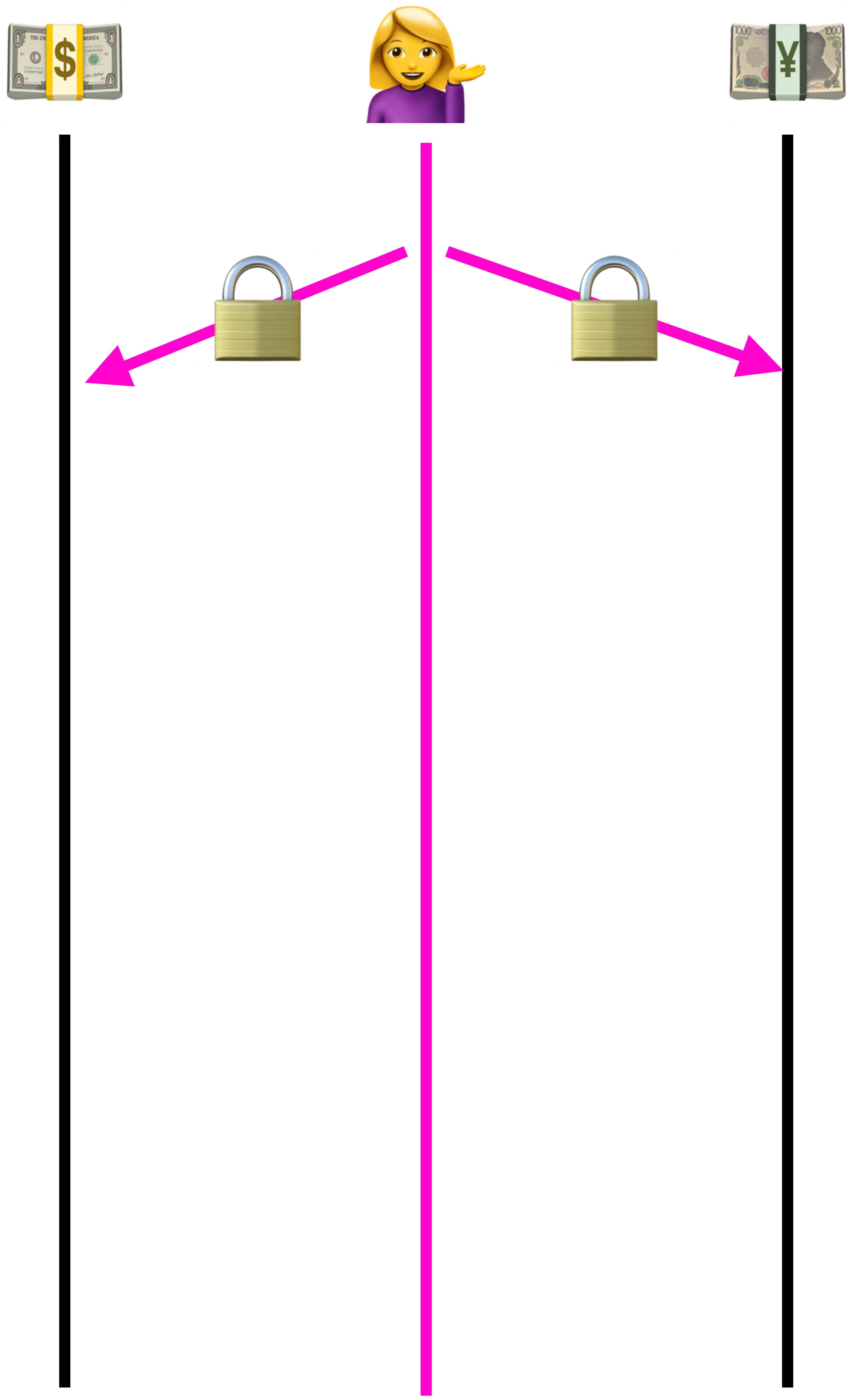
Optimistic Concurrency: STM



https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

Purify Your Effects 🦺⛑️🪄

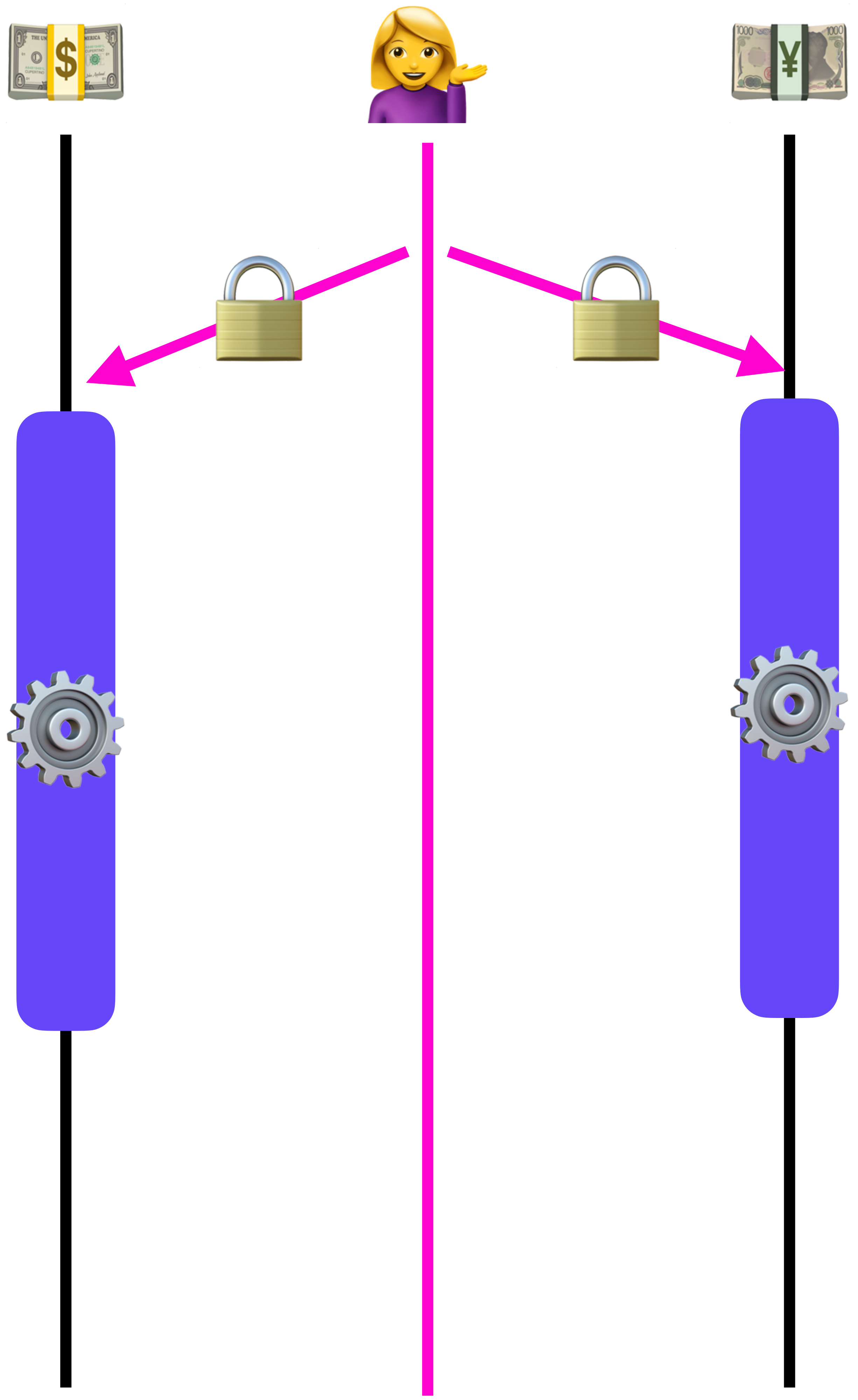
Optimistic Concurrency: STM



https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

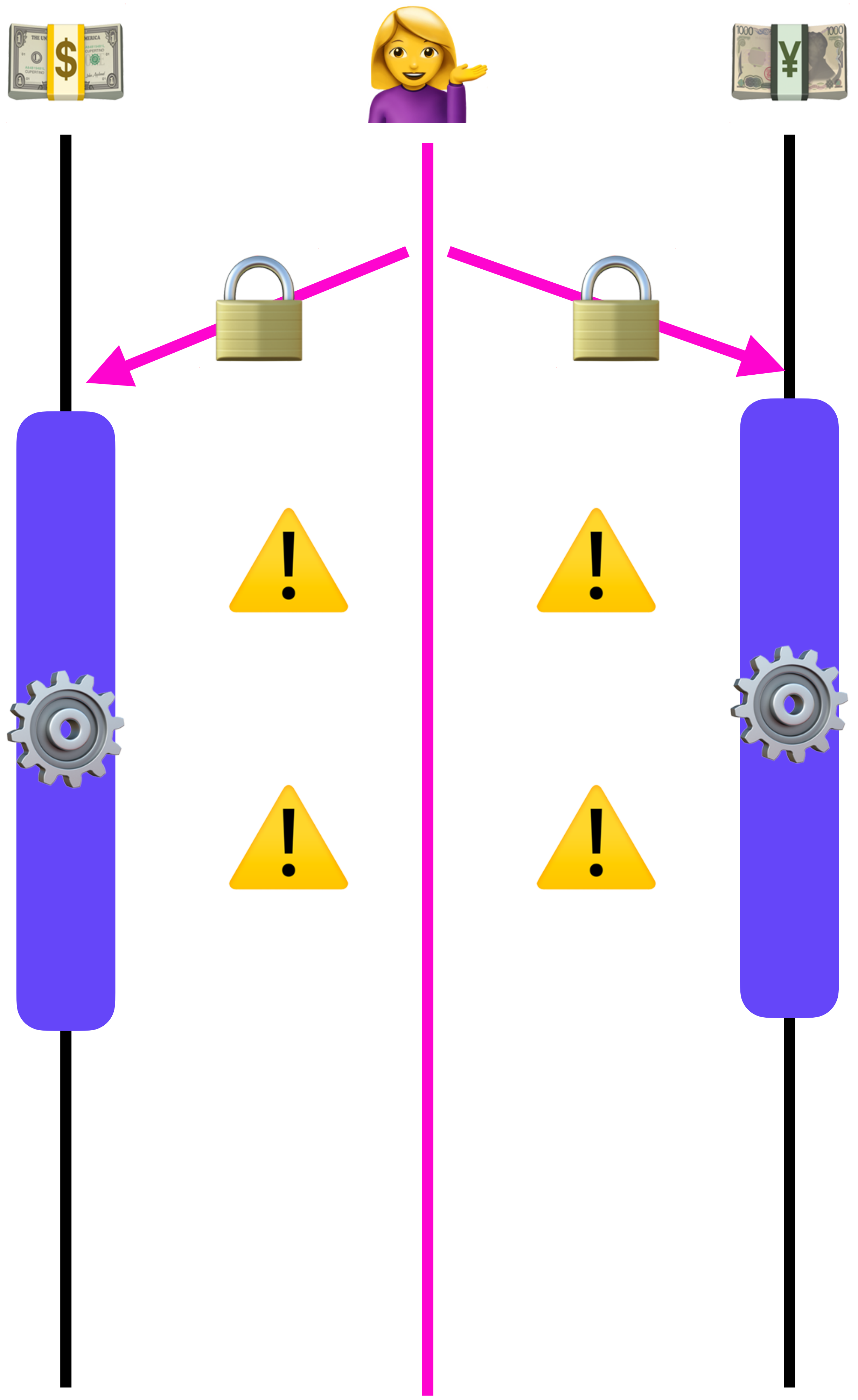
Purify Your Effects 🦺⛑️🪄

Optimistic Concurrency: STM



https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

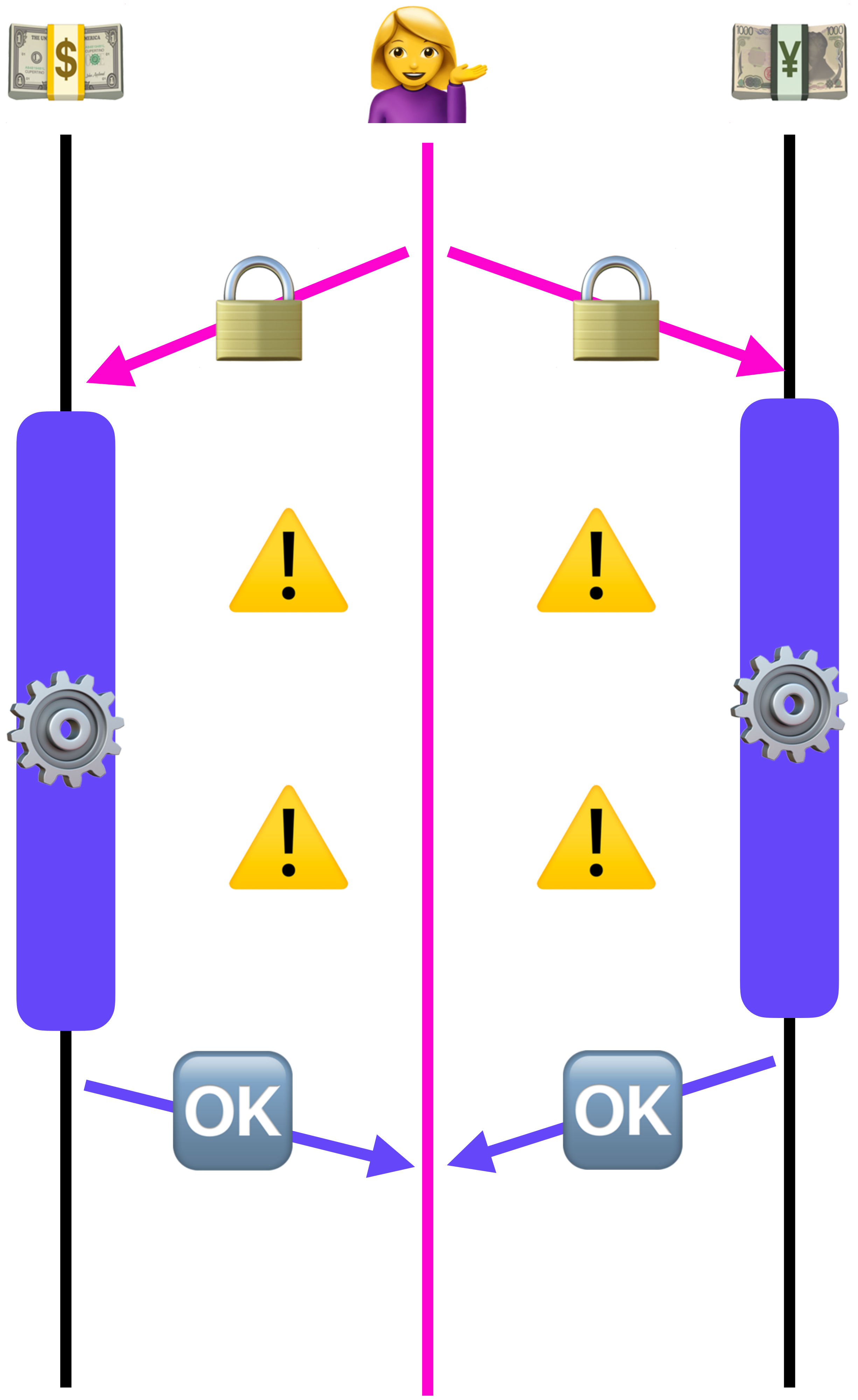
Optimistic Concurrency: STM



https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

Purify Your Effects 🦺⛑️🪄

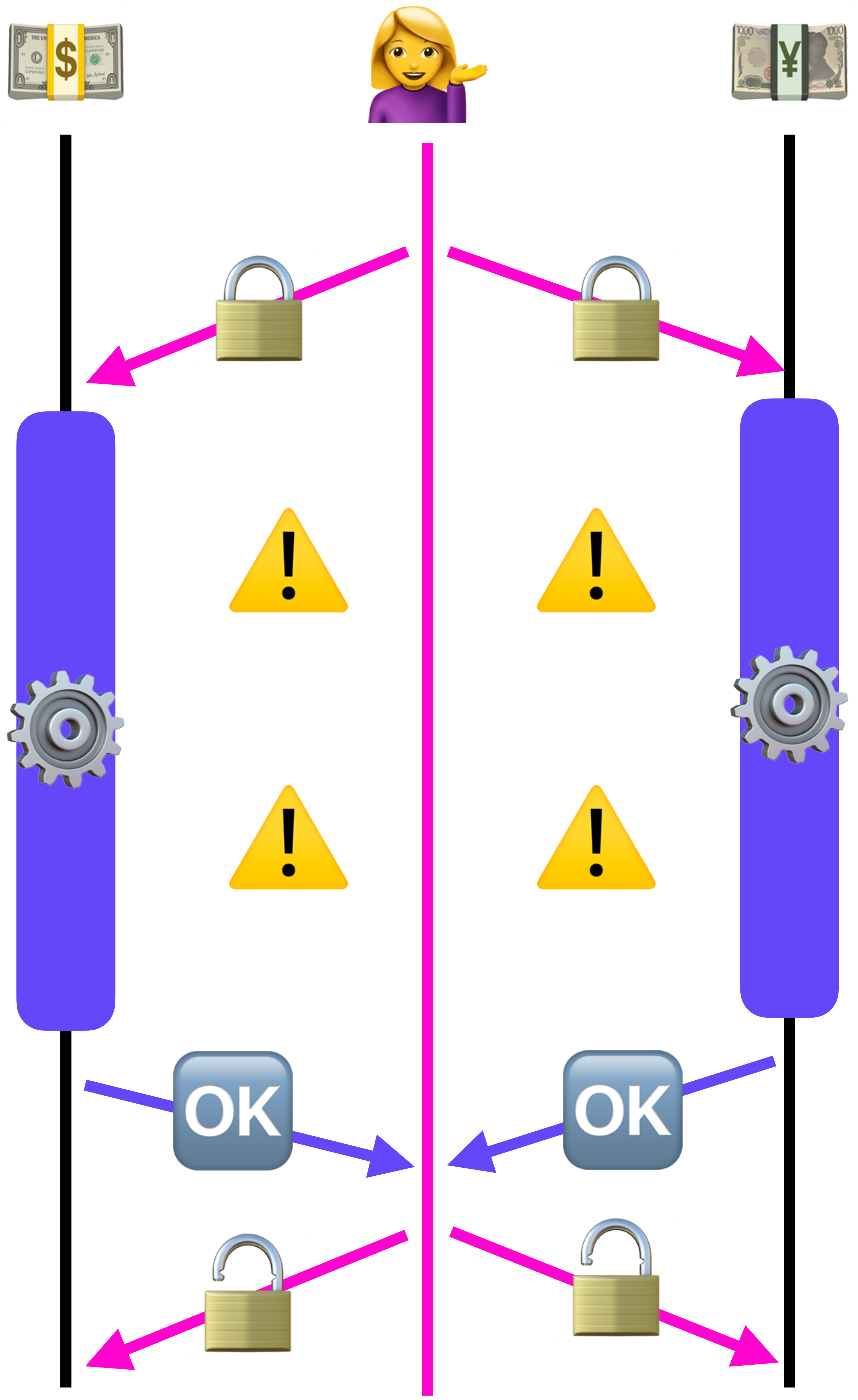
Optimistic Concurrency: STM



https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

Purify Your Effects 🦺⛑️🪄

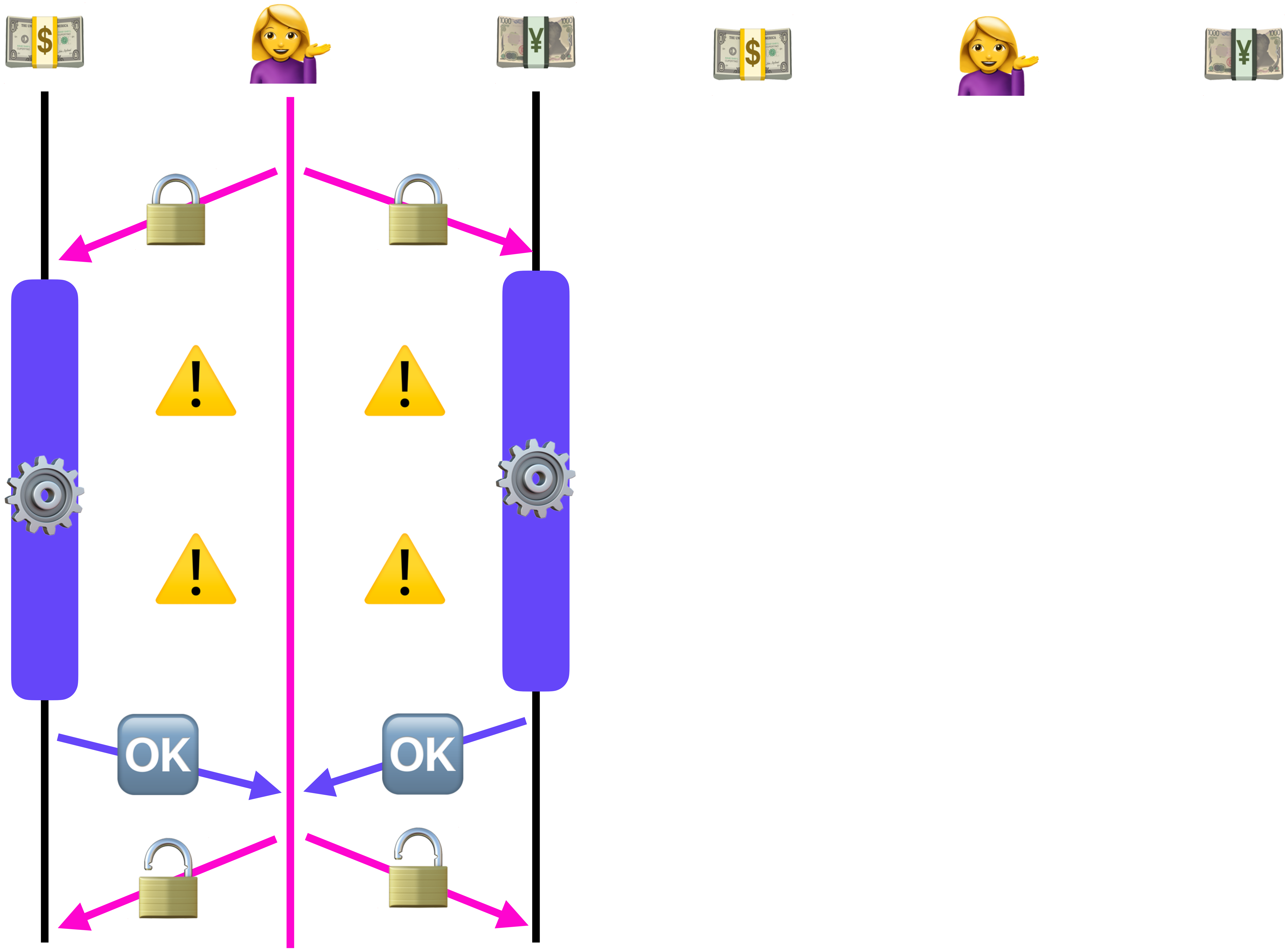
Optimistic Concurrency: STM



https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

Purify Your Effects 🦺⛑️🪄

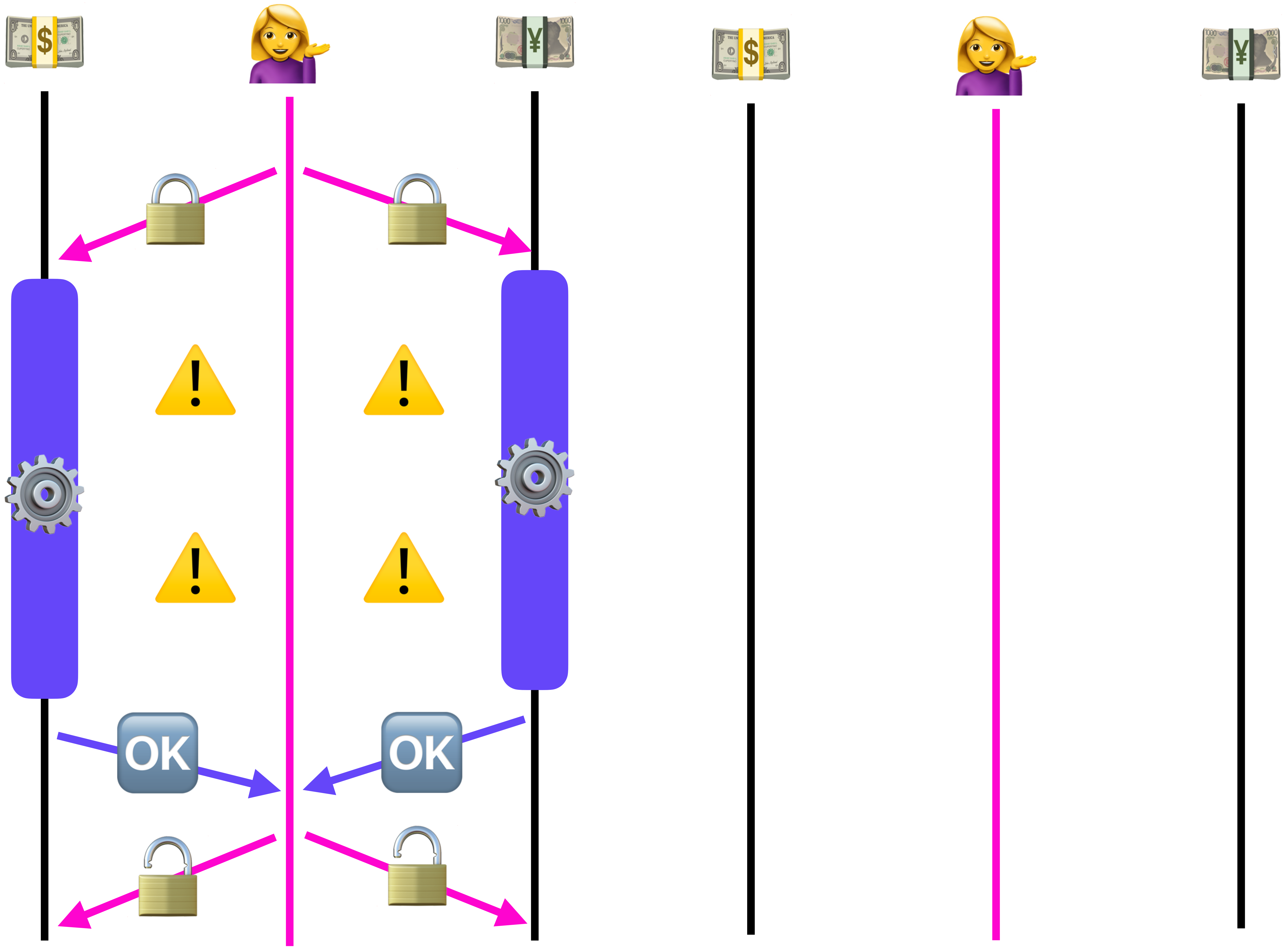
Optimistic Concurrency: STM



https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

Purify Your Effects 🦺⛑️🪄

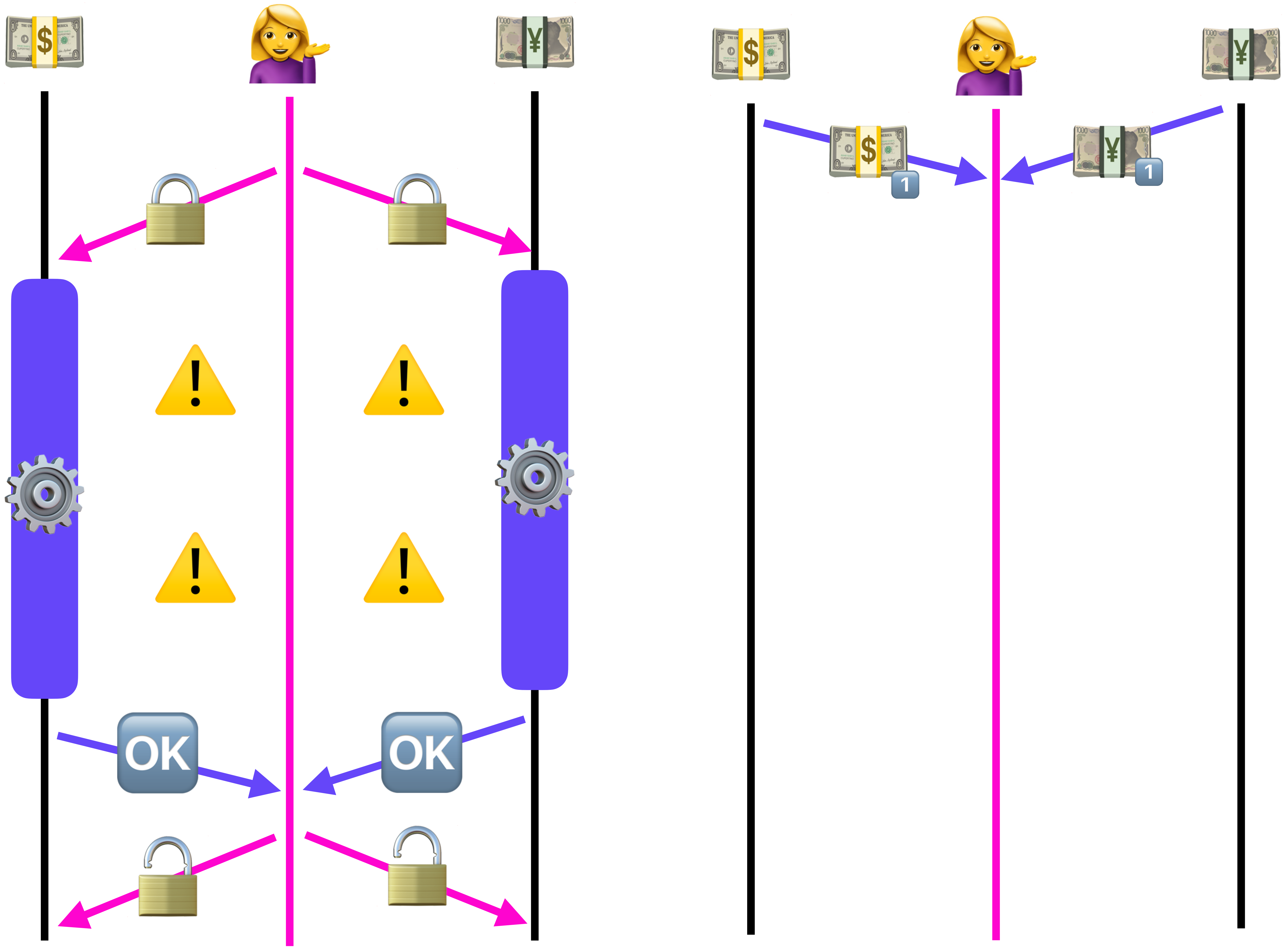
Optimistic Concurrency: STM



https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

Purify Your Effects 🦺⛑️🪄

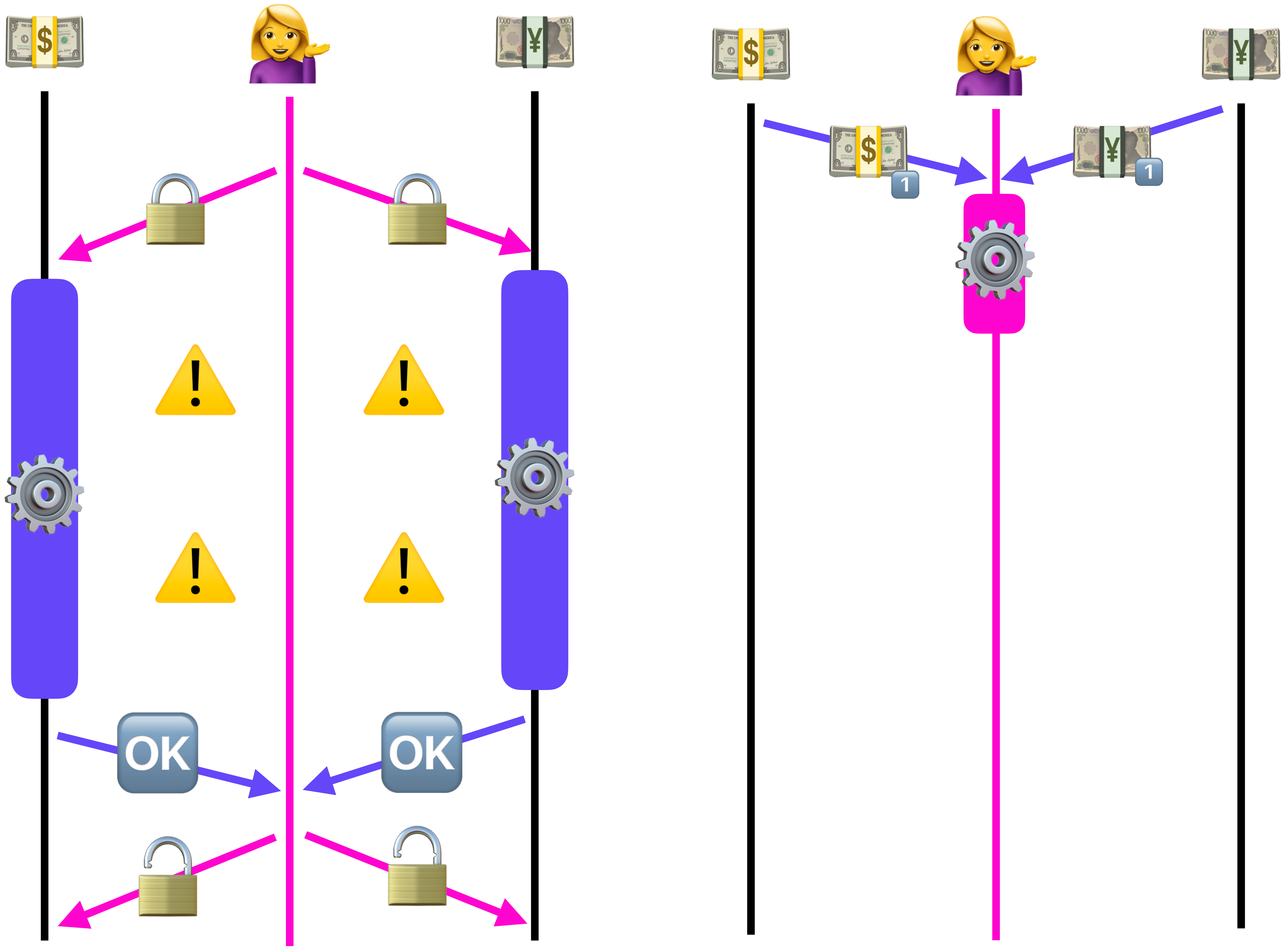
Optimistic Concurrency: STM



https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

Purify Your Effects 🦺⛑️🪄

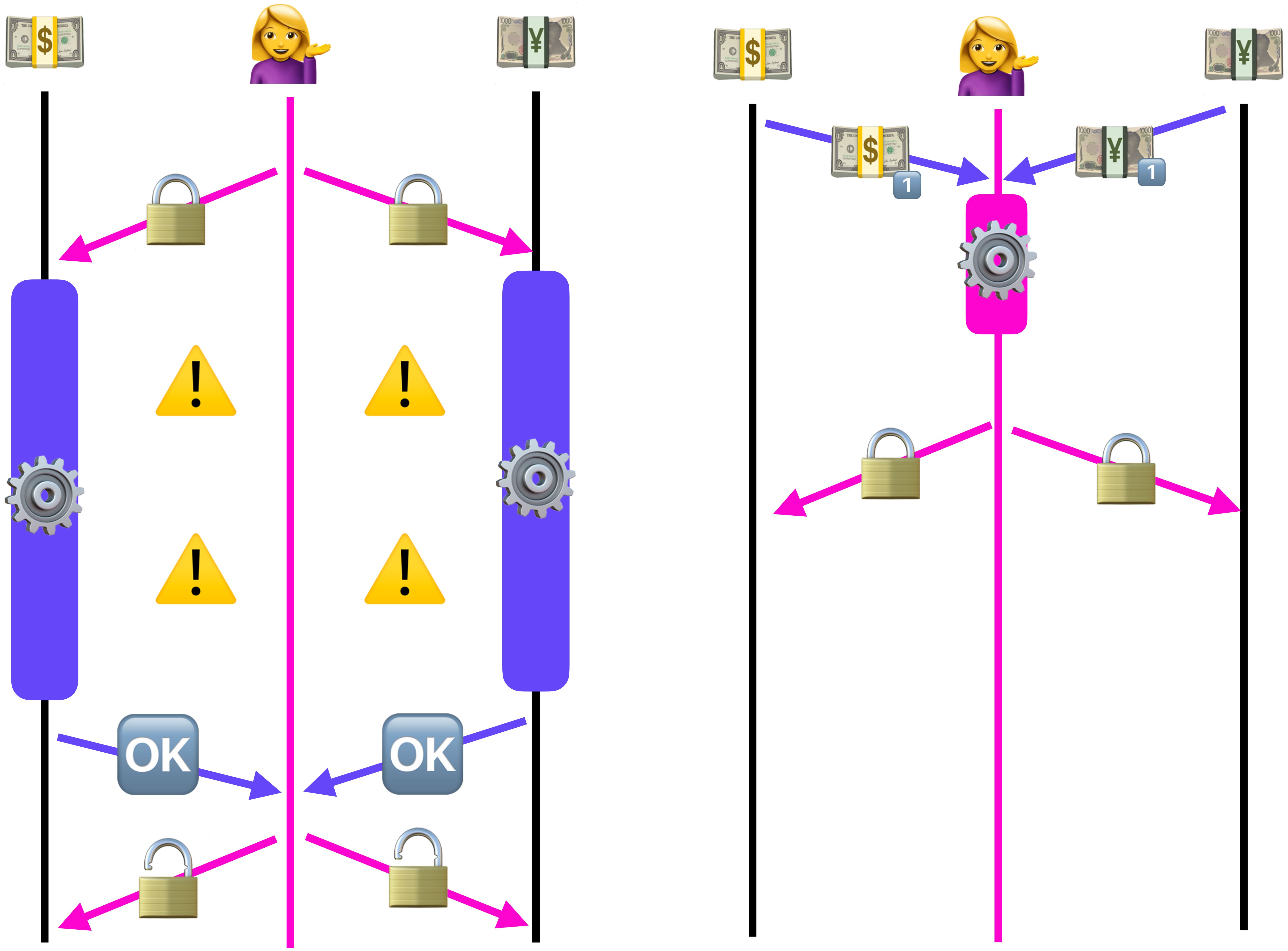
Optimistic Concurrency: STM



https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

Purify Your Effects 🦺⛑️🪄

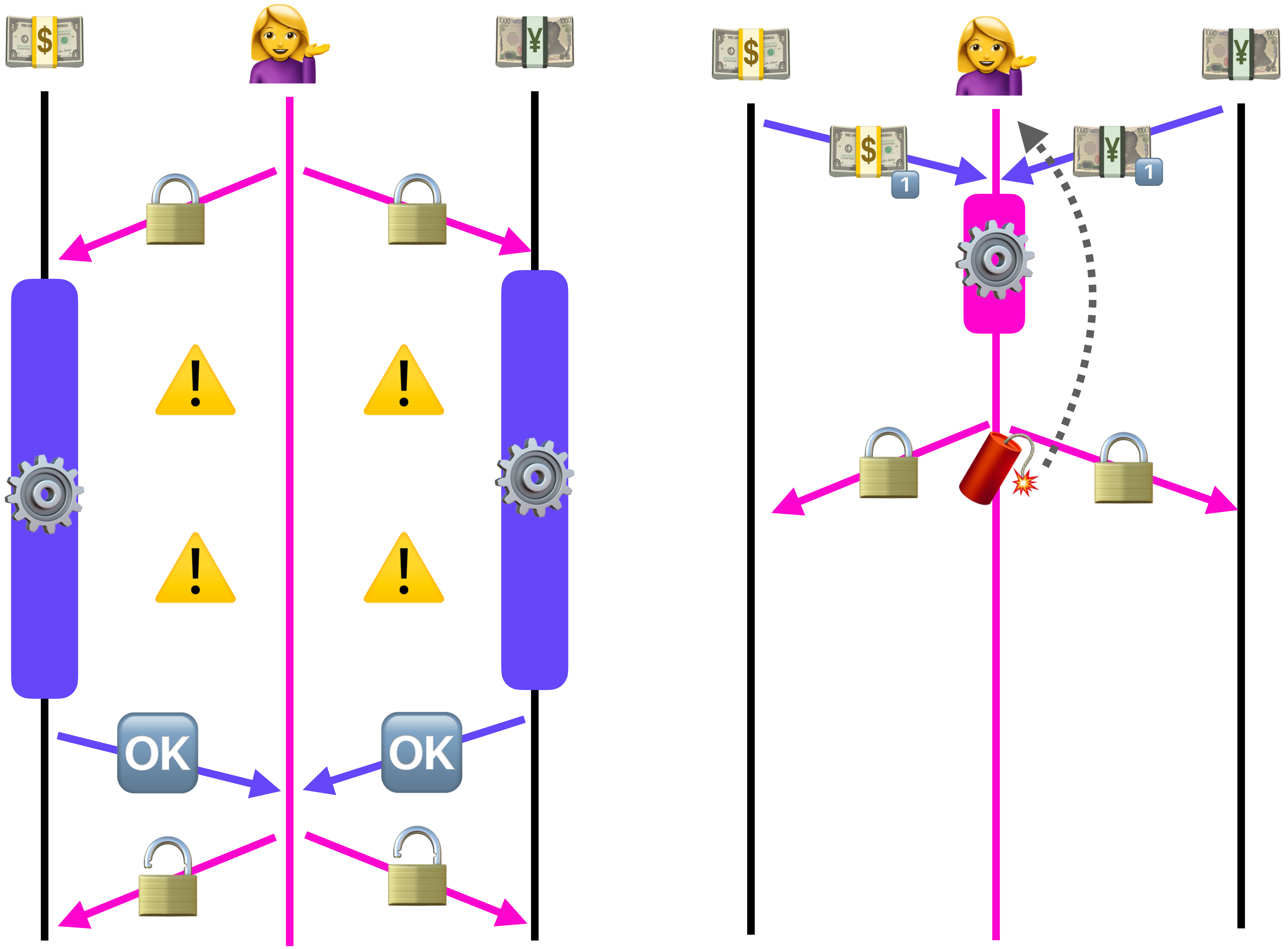
Optimistic Concurrency: STM



https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

Purify Your Effects 🦺⛑️🪄

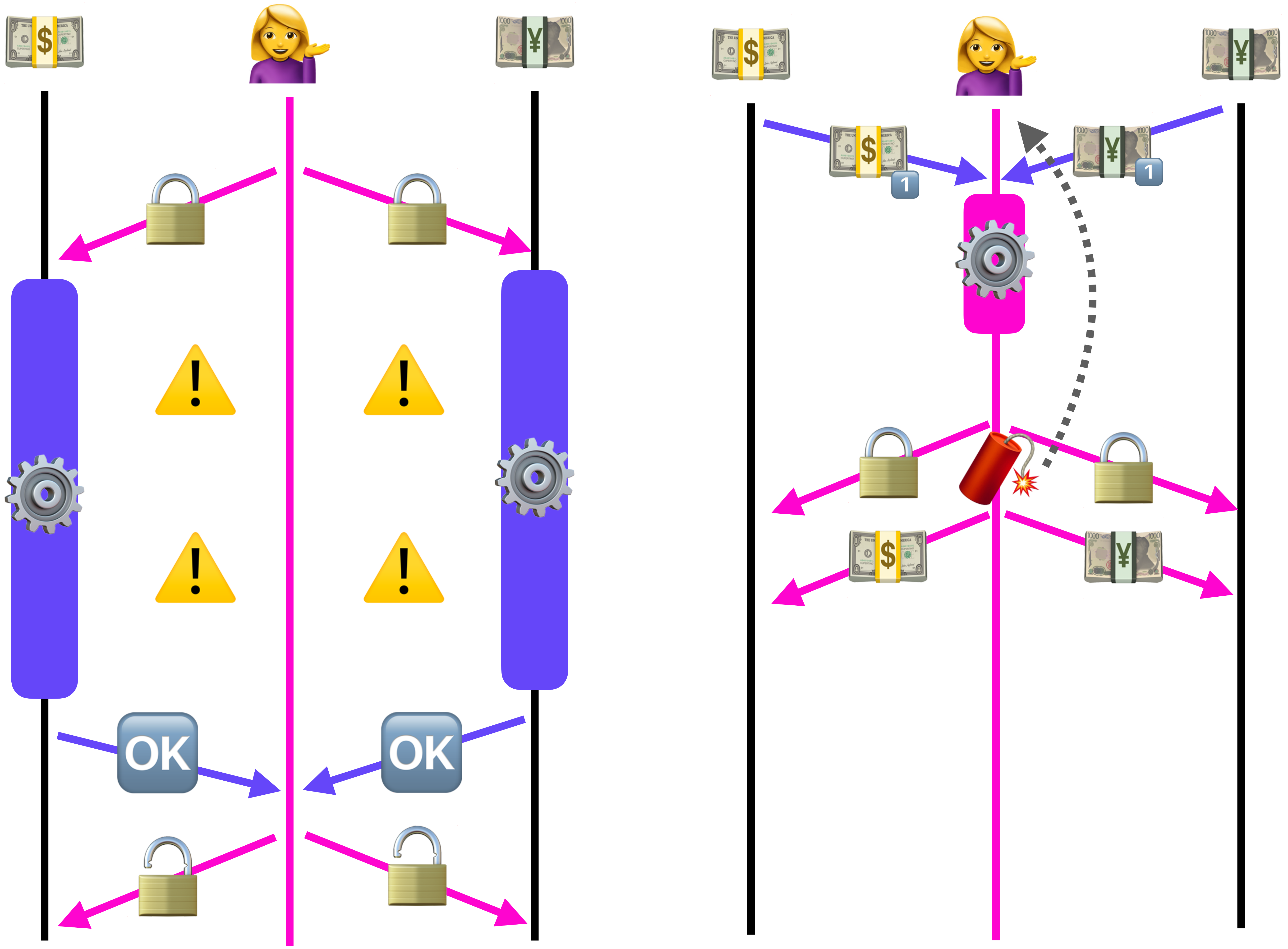
Optimistic Concurrency: STM



https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

Purify Your Effects 🦺⛑🪄

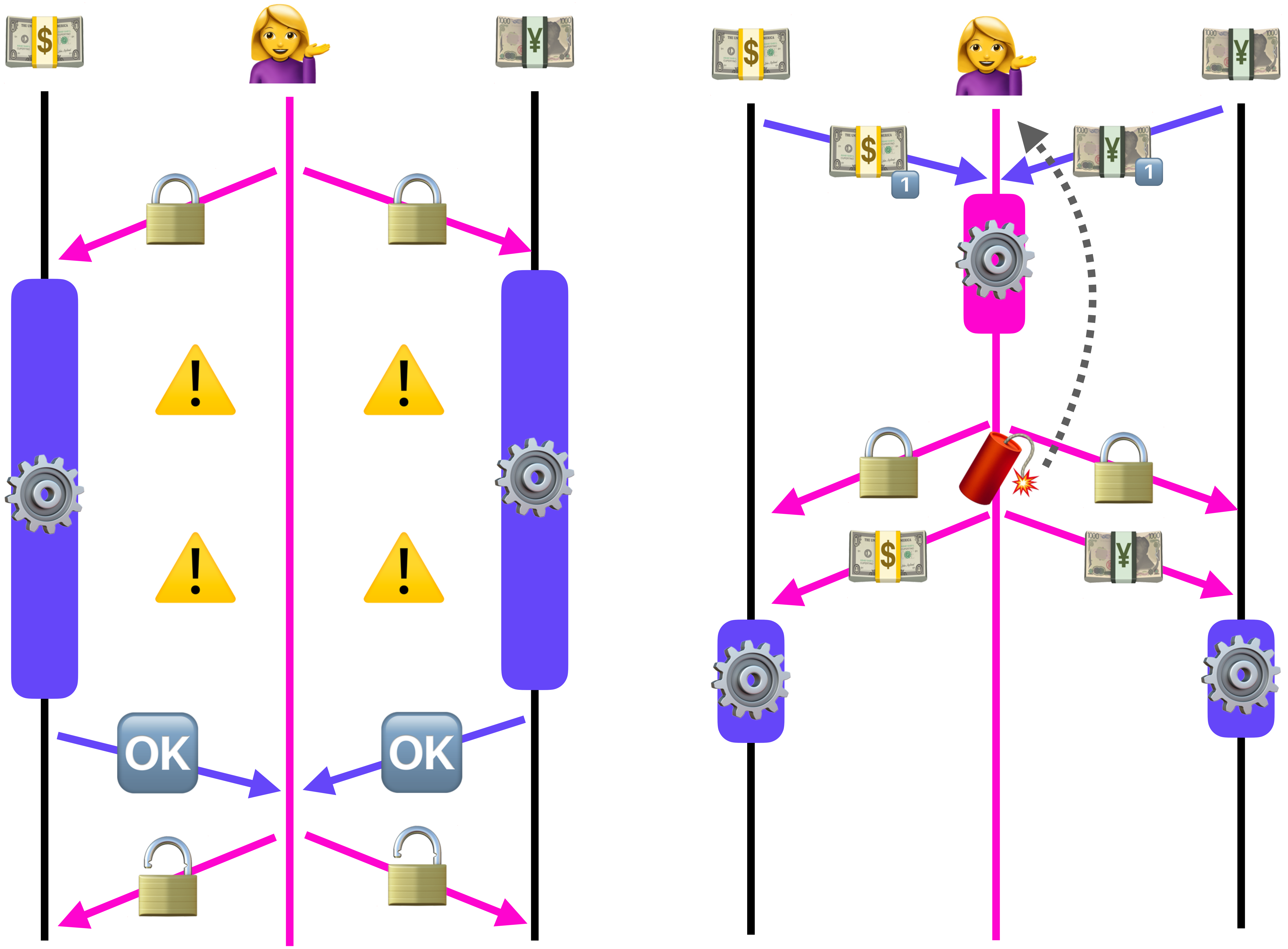
Optimistic Concurrency: STM



https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

Purify Your Effects 🦺⛑️🪄

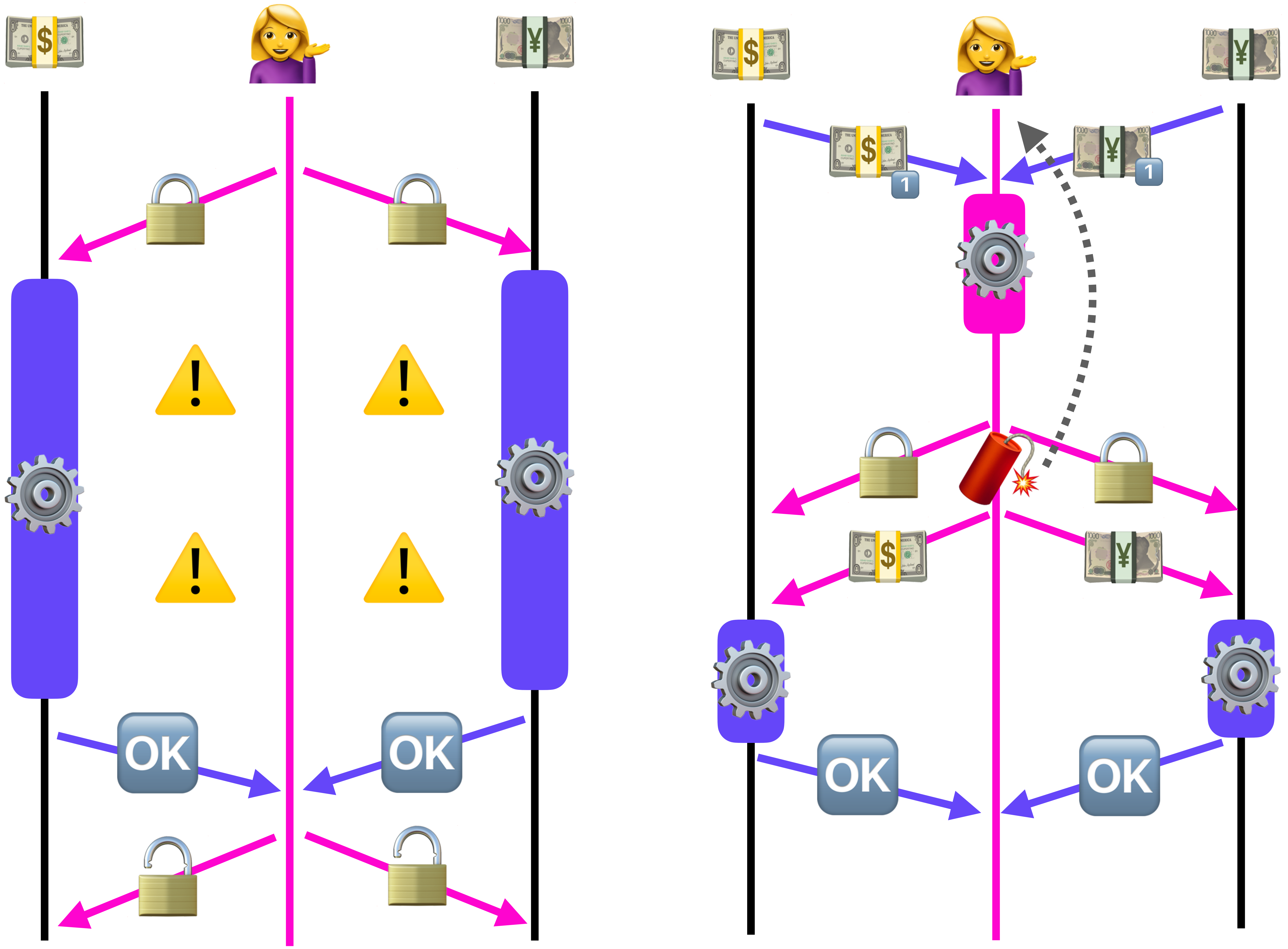
Optimistic Concurrency: STM



https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

Purify Your Effects 🦺⛑️🪄

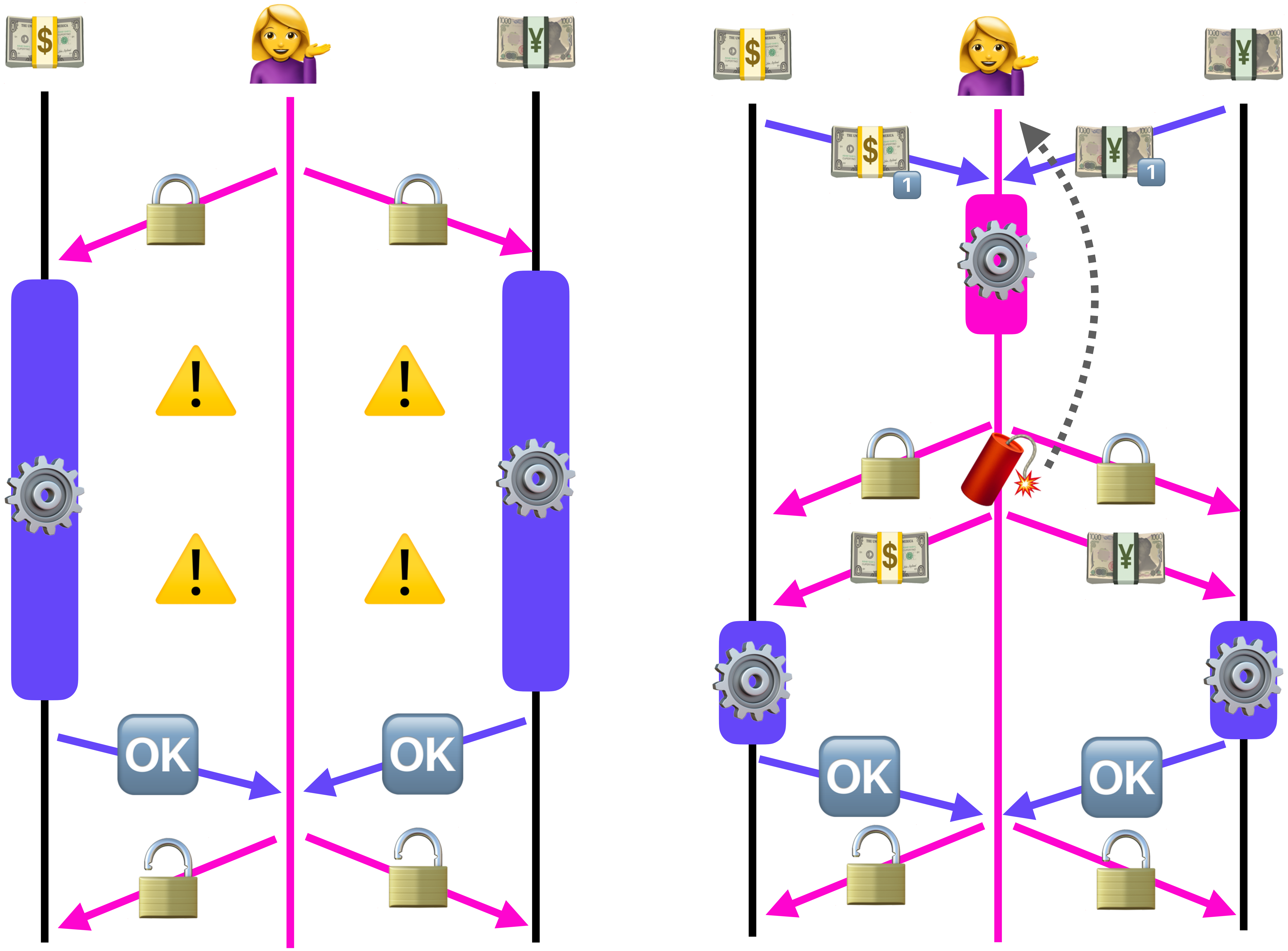
Optimistic Concurrency: STM



https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

Purify Your Effects 🦺⛑️🪄

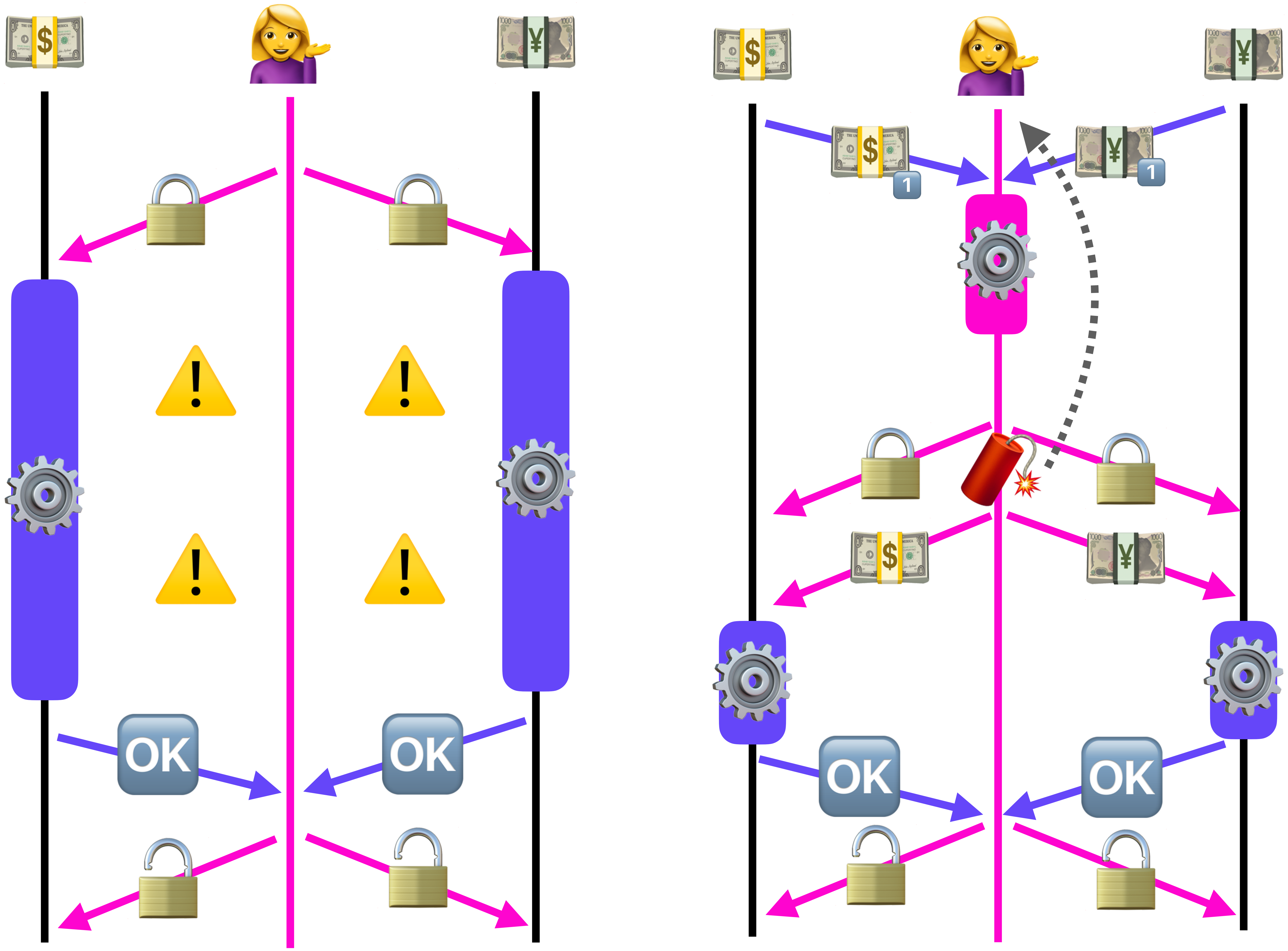
Optimistic Concurrency: STM



https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

Purify Your Effects 🦺⛑️🪄

Optimistic Concurrency: STM



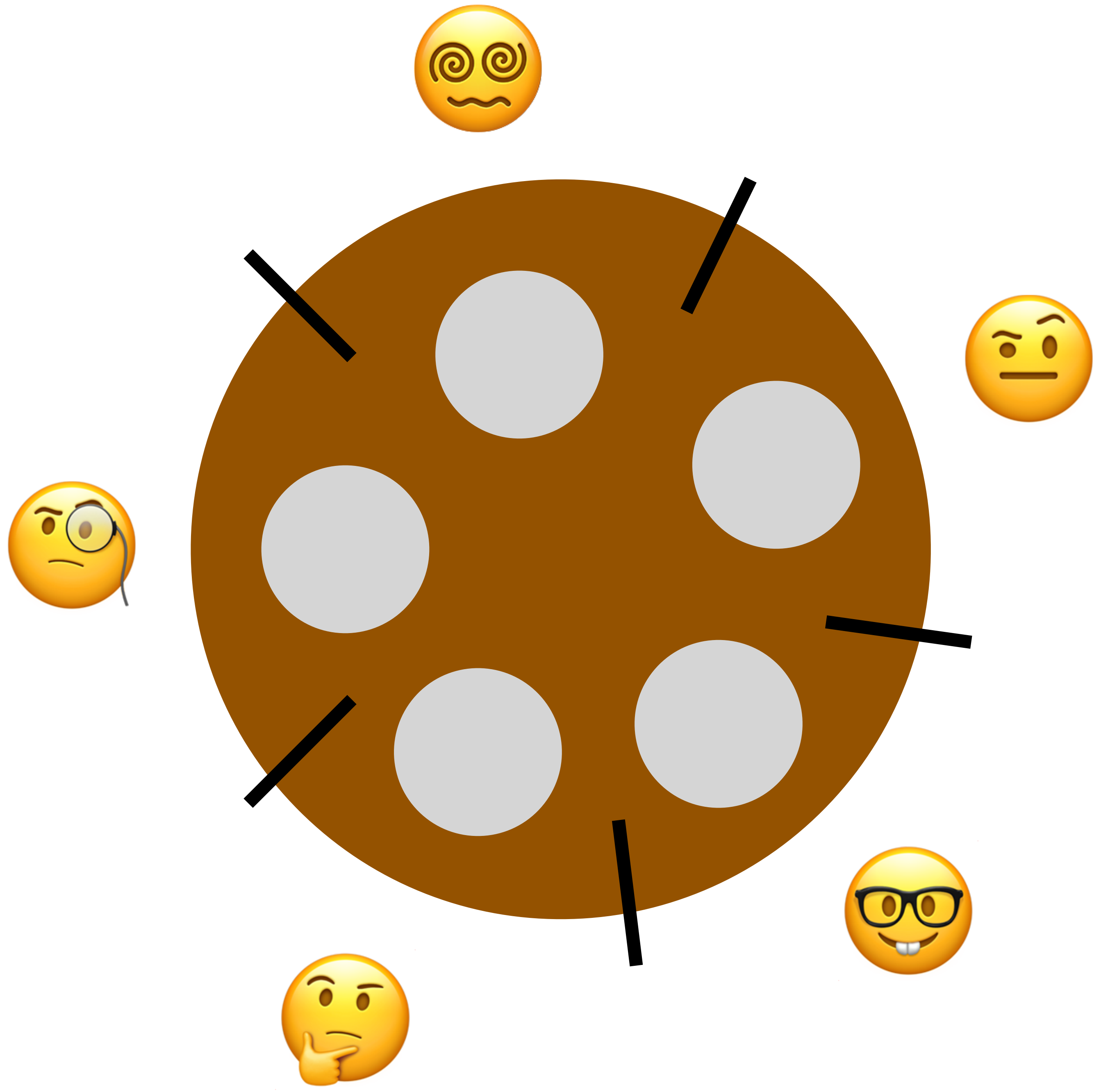

```elixir
def transfer(a, b, amount) do
  transact do
    Process.sleep(1000)
    STM.set(a.balance, &(&x + amount))
    STM.set(b.balance, &(&x - amount))
  end

  send_email(to: [a, b], "Transferred $#{amount}")
end
```

https://www.cs.rochester.edu/u/scott/papers/2006_TRANSACT_RSTM.pdf

Purify Your Effects 🦺⛑️🪄

Optimistic Concurrency: STM

```
def dine(name, leftStick, rightStick) do
  transact do
    STM.take(leftStick)
    STM.take(rightStick)
  end

  IO.puts("#{name} is eating")
  Process.sleep(Enum.random(0..n))

  transact do
    STM.put(leftStick, :chopstick)
    STM.put(rightStick, :chopstick)
  end
end
```



Optimistic Concurrency: STM

Effectful Proof, Provenance, & Power

Metadata in Motion

Effectful Proof, Provenance, & Power

Metadata in Motion

Metadata in Motion 📦💨

It's only ***data provenance*** if it's derived from the ***Provence region of France.*** Otherwise it's just ***sparkling metadata.***

– Adapted from @onfiv

Metadata in Motion 📦💨

Proof Carrying Code

Metadata in Motion 📦💨

Proof Carrying Code

```
defmodule NonEmptyList do
  defstruct [:head, :tail]

  def singleton(x) do
    %NonEmptyList{head: x, tail: []}
  end

  def to_list(%NonEmptyList{head: head, tail: tail}) do
    [head | tail]
  end

  def from_list([]), do: :empty
  def from_list([x | xs]), do: %NonEmptyList{head: x, tail: xs}
end

defimpl Enumerable, for: NonEmptyList do
  def count(%NonEmptyList{tail: tail}) do
    count(rest) + 1
  end
end
```

Metadata in Motion 📦💨

Proof Carrying Code

```elixir
defmodule NonEmptyList do
  defstruct [:head, :tail]

  def singleton(x) do
    %NonEmptyList{head: x, tail: []}
  end

  def to_list(%NonEmptyList{head: head, tail: tail}) do
    [head | tail]
  end

  def from_list([]), do: :empty
  def from_list([x | xs]), do: %NonEmptyList{head: x, tail: xs}
end

defimpl Enumerable, for: NonEmptyList do
  def count(%NonEmptyList{tail: tail}) do
    count(rest) + 1
  end
end
```

```elixir
%NonEmptyList{
  head: 0,
  tail: [1,2,3]
}
```

Metadata in Motion 📦💨

Proof Carrying Code

```elixir
defmodule NonEmptyList do
  defstruct [:head, :tail]

  def singleton(x) do
    %NonEmptyList{head: x, tail: []}
  end

  def to_list(%NonEmptyList{head: head, tail: tail}) do
    [head | tail]
  end

  def from_list([]), do: :empty
  def from_list([x | xs]), do: %NonEmptyList{head: x, tail: xs}
end

defimpl Enumerable, for: NonEmptyList do
  def count(%NonEmptyList{tail: tail}) do
    count(rest) + 1
  end
end
```

```elixir
%NonEmptyList{
  head: 0,
  tail: [1,2,3]
}
```

```elixir
%Sorted{
  by: :lex,
  enum: ["a", "b", "cdef"]
}

%Sorted{
  by: :lex,
  enum: ["b", "z"]
}
```

Metadata in Motion 📦💨

Proof Carrying Code

```elixir
defmodule NonEmptyList do
  defstruct [:head, :tail]

  def singleton(x) do
    %NonEmptyList{head: x, tail: []}
  end

  def to_list(%NonEmptyList{head: head, tail: tail}) do
    [head | tail]
  end

  def from_list([]), do: :empty
  def from_list([x | xs]), do: %NonEmptyList{head: x, tail: xs}
end

defimpl Enumerable, for: NonEmptyList do
  def count(%NonEmptyList{tail: tail}) do
    count(rest) + 1
  end
end
```

```elixir
%NonEmptyList{
  head: 0,
  tail: [1,2,3]
}
```

```elixir
%Sorted{
  by: :lex,
  enum: ["a", "b", "cdef"]
}

%Sorted{
  by: :lex,
  enum: ["b", "z"]
}
```

mergesort is way easier now

Metadata in Motion 📦💨

Carrying Capabilities

https://kataskeue.com/gdp.pdf
https://arxiv.org/pdf/1907.07154.pdf

Metadata in Motion 📦💨

Carrying Capabilities

```
%CanDo {
  caps: [
    %{can: :overwrite, directory: "/tmp/files/"},
    %{can: :send_email, as: "boris@fission.codes"}
  ],
  for: %User{id: 42},
  session: 123
}
```

https://kataskeue.com/gdp.pdf
https://arxiv.org/pdf/1907.07154.pdf

Metadata in Motion 📦💨

Carrying Capabilities

```
%CanDo {
  caps: [
    %{can: :overwrite, directory: "/tmp/files/"},
    %{can: :send_email, as: "boris@fission.codes"}
  ],
  for: %User{id: 42},
  session: 123
}
```

```
def send_email(%CanDo{caps: caps, for: user}, msg, to) do
  case find_send(caps) do
    %{to: address} -> Email.send(from: user, to: address)
    nil -> :unauthorized
  end
end
```

https://kataskeue.com/gdp.pdf
https://arxiv.org/pdf/1907.07154.pdf

Metadata in Motion 📦💨

Provenance

Metadata in Motion 📦💨

Provenance

```
%Rememeber{
  value: 42,
  history: [
    %{value: 37},
    %{value: 109}
  ]
}
```

Metadata in Motion 📦💨

Provenance

```
%Rememeber{
  value: 42,
  history: [
    %{value: 37},
    %{value: 109}
  ]
}
```

```
%BranchableHistory{
  value: 42,
  histories: [
    %Branch{
      histories: [
        %{value: 12},
        %{value: 0},
      ]
    },
    %Branch{
      histories:[
        %{value: 37},
        %{value: 109},
      ]
    },
    %{value: 0}
  ]
}
```

Metadata in Motion 📦💨

Provenance

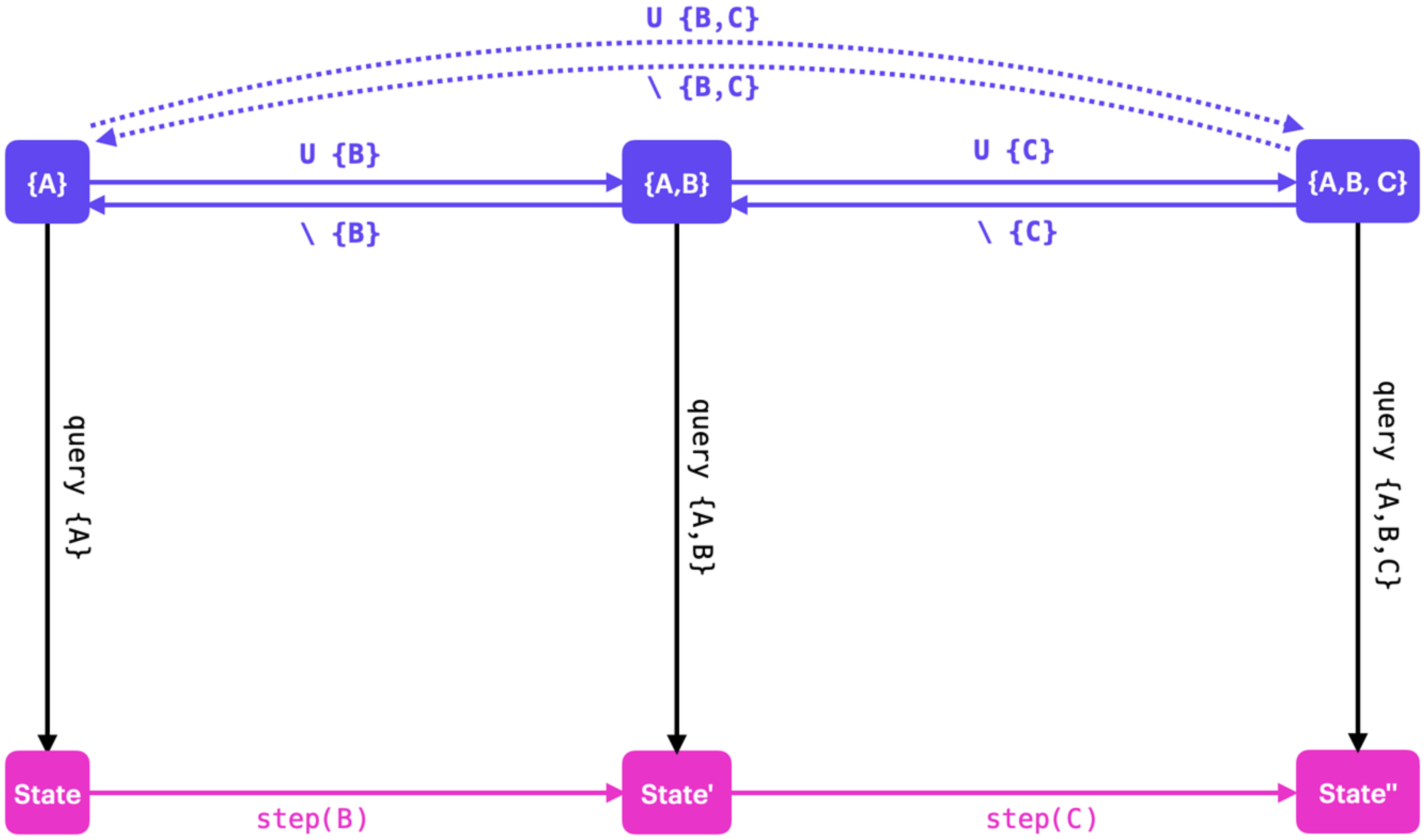
```
%Rememeber{
  value: 42,
  history: [
    %{value: 37},
    %{value: 109}
  ]
}
```

```
%BranchableHistory{
  value: 42,
  histories: [
    %Branch{
      histories: [
        %{value: 12},
        %{value: 0},
      ]
    },
    %Branch{
      histories:[
        %{value: 37},
        %{value: 109},
      ]
    },
    %{value: 0}
  ]
}
```

```
Logger.debug("#{__ENV__.file}:#{__ENV__.line}: #{inspect some_value}")
```

Metadata in Motion 📦💨

Provenance++



Tools for The Intrepid

⚒️

Wrapping Up

Tools for The Intrepid

Tools for The Intrepid ⚒️

A programming language ***influences the way that its users think*** about programming; matching a language to a methodology ***increases the likelihood that the methodology will be used***

– Barbara Liskov et al, Abstraction Mechanisms in CLU

Tools for The Intrepid ⚒️

What Just Happened?

Tools for The Intrepid ⚒️

What Just Happened?

1. Add structure & dimension

Tools for The Intrepid ⚒️

What Just Happened?

1. Add structure & dimension

2. Make reusable DSLs

Tools for The Intrepid ⚒️

What Just Happened?

1. Add structure & dimension
2. Make reusable DSLs
3. Manage your effects

Tools for The Intrepid ⚒️

What Just Happened?

1. Add structure & dimension
2. Make reusable DSLs
3. Manage your effects
4. Mechanize the hard stuff (e.g. concurrency)

Tools for The Intrepid ⚒️

What Just Happened?

1. Add structure & dimension
2. Make reusable DSLs
3. Manage your effects
4. Mechanize the hard stuff (e.g. concurrency)
5. Pass around context

Tools for The Intrepid ⚒️

Is this the future?

Is this the future?

I don't know! 🤷‍♀️

Is this the future?

I don't know! 🤷‍♀️

We need to ***experiment*** with more directions

Is this the future?

I don't know! 🤷‍♀️

We need to ***experiment*** with more directions

These are but a few options

Is this the future?

I don't know! 🤷‍♀️

We need to ***experiment*** with more directions

These are but a few options

Go explore! 🧭

Is this the future?

I don't know! 🤷‍♀️

We need to ***experiment*** with more directions

These are but a few options

Go explore! 🧭

(And share what you find)

🎉 Thank You, Salt Lake City! 🇺🇸

✦ https://lu.ma/distributed-systems

https://fission.codes/discord

github.com/expede

@expede