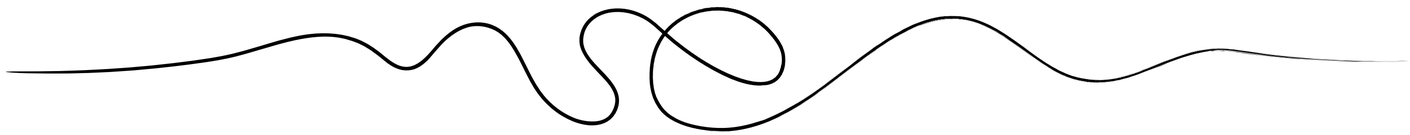
SIMPLE vs EASY



QUICK OVERVIEW

- **Simple solutions are not easy to use**
- **The 2 faces of complexity**
- **Case study: managing complexity**

WARNING!

MOSTLY BASED ON

PERSONAL EXPERIENCES

ANDREI PFEIFFER

Timișoara / RO

Code Designer

UI Engineer for Web & Mobile

timjs revo·js

Co-Organizer

andreipfeiffer.dev

@pfeiffer_andrei

Home

FULL CONTROL

LINEAR EFFORT

1.8 km, 22 mins walk

Home

Office

NO CONTROL

EFFORTLESS

Home

1461 km, 297 hrs walk

Office

DEV.PRO, Kharkiv

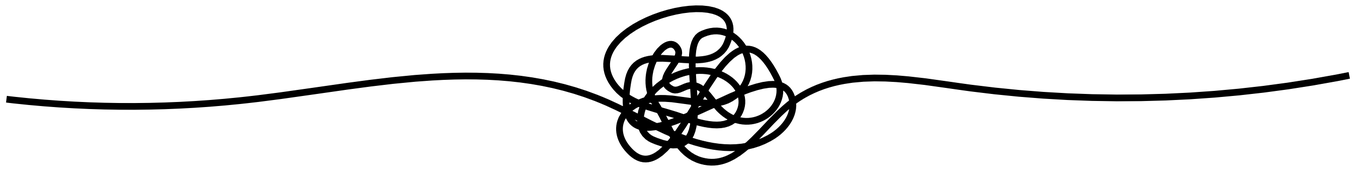
SIMPLE

SIMPLE

"one fold"

straightforward, easily understood
uneducated, unsophisticated, stupid

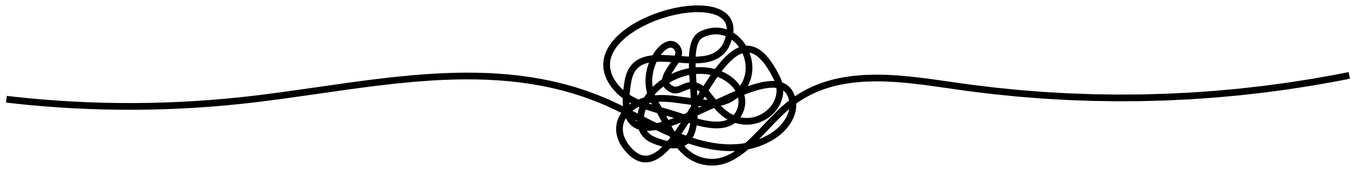
COMPLEX

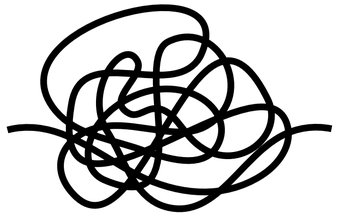


"composed of interconnected parts"

intricate, complicated, not easily analyzed

COMPLEX



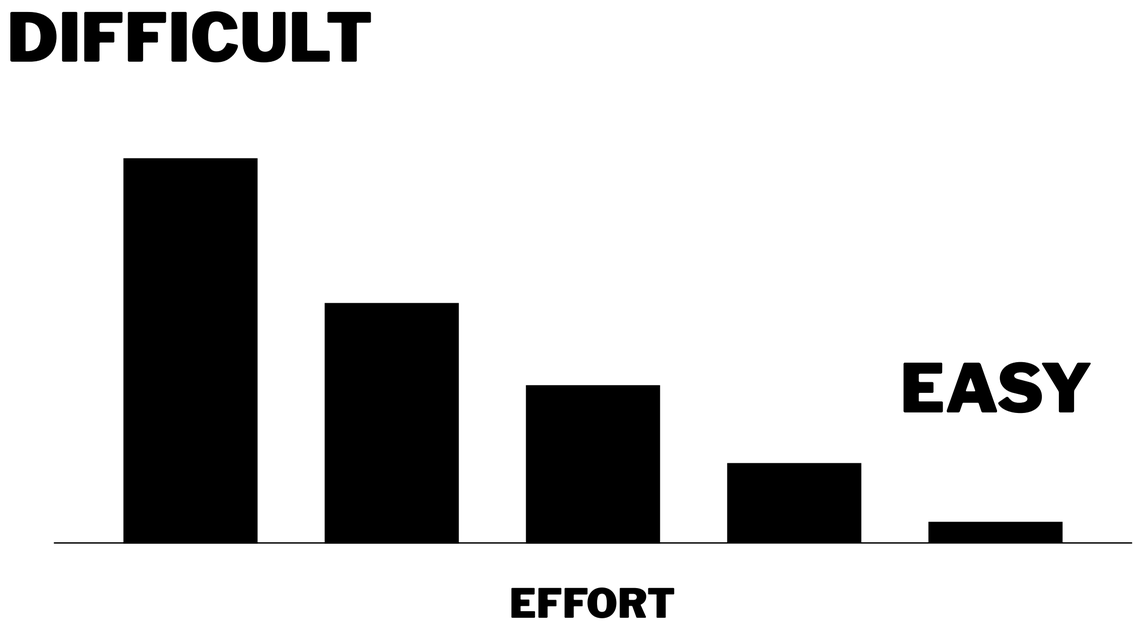
SIMPLE   COMPLEX

HOW SOMETHING IS BUILT

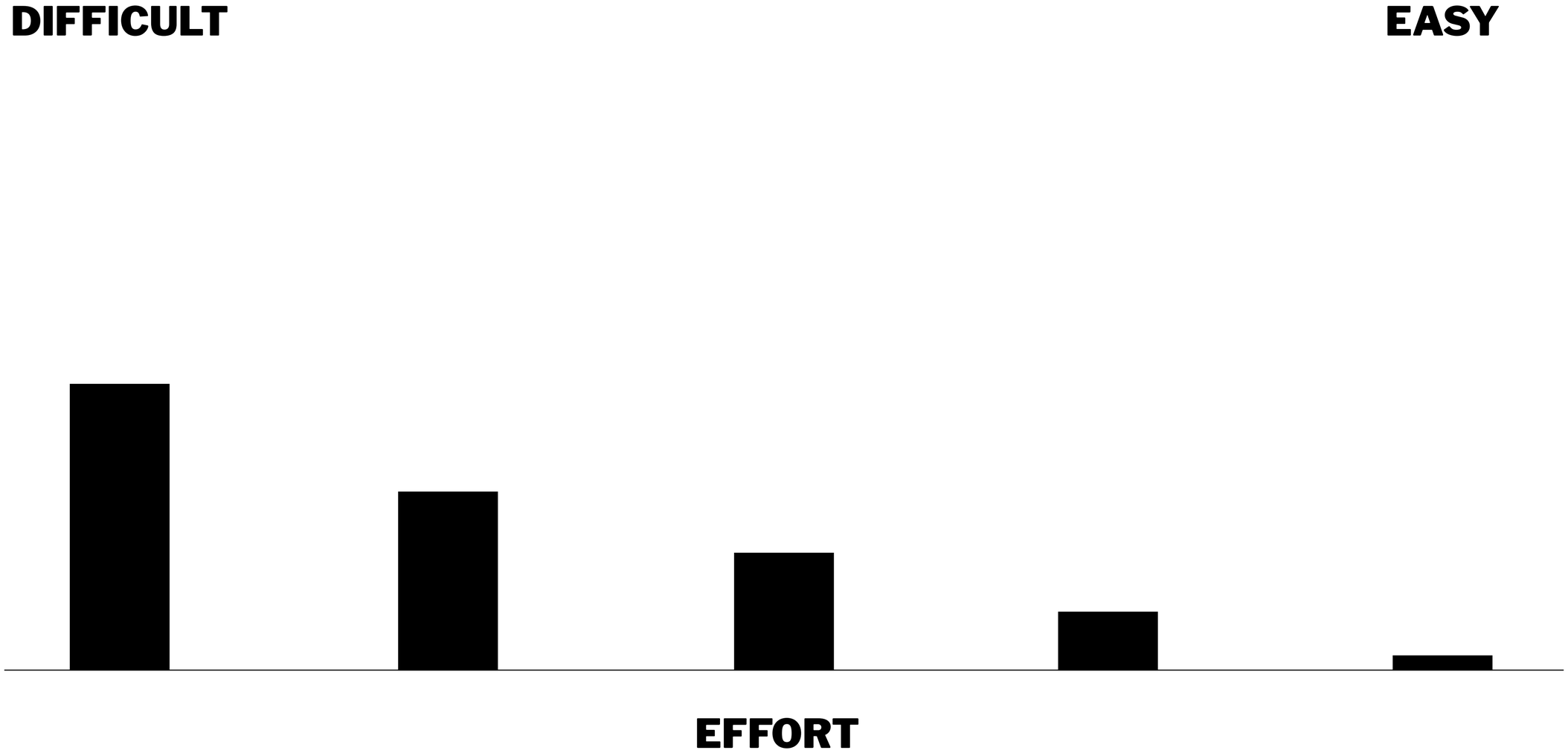
EASY

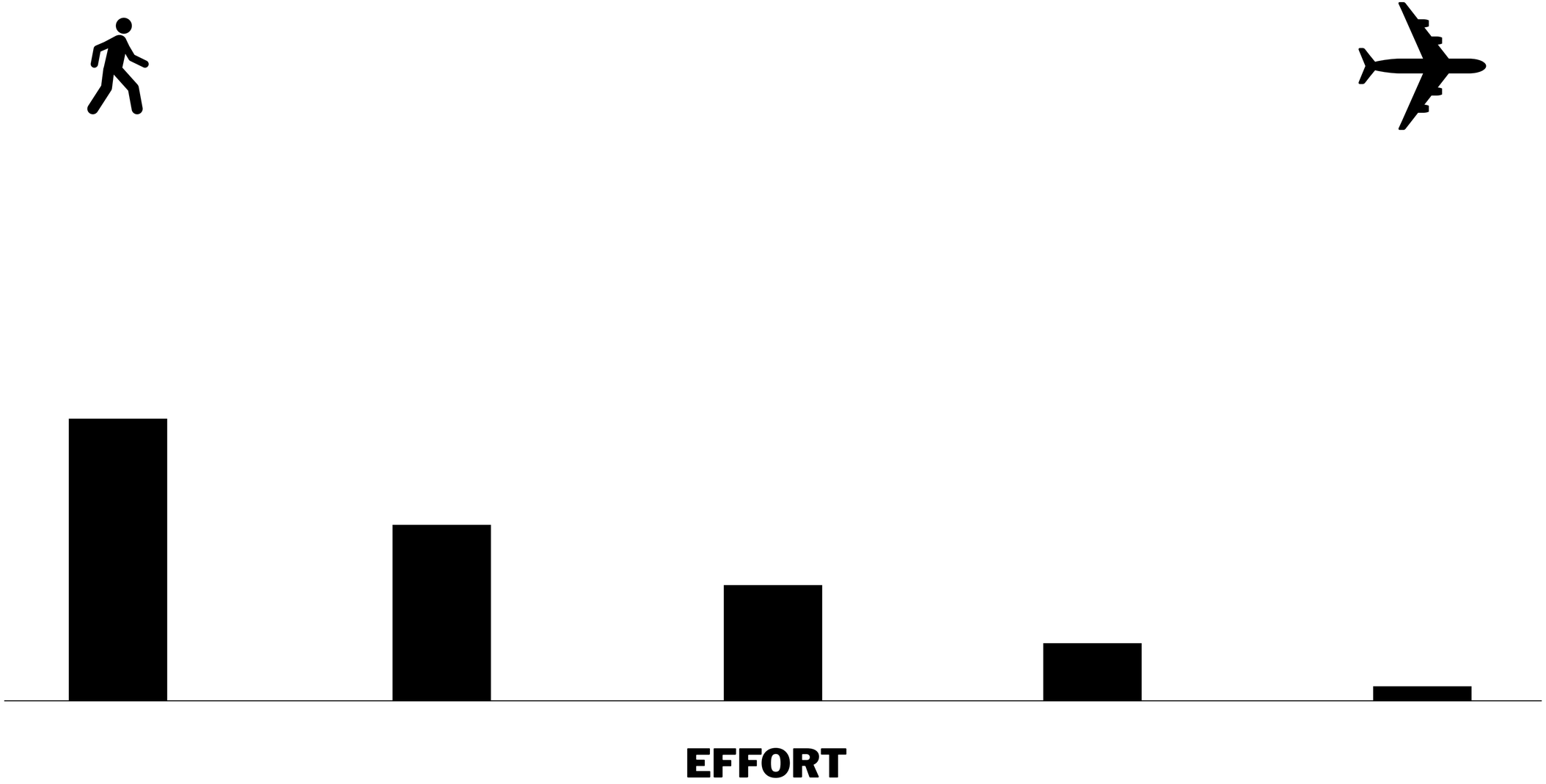
"requiring no great labor or effort"

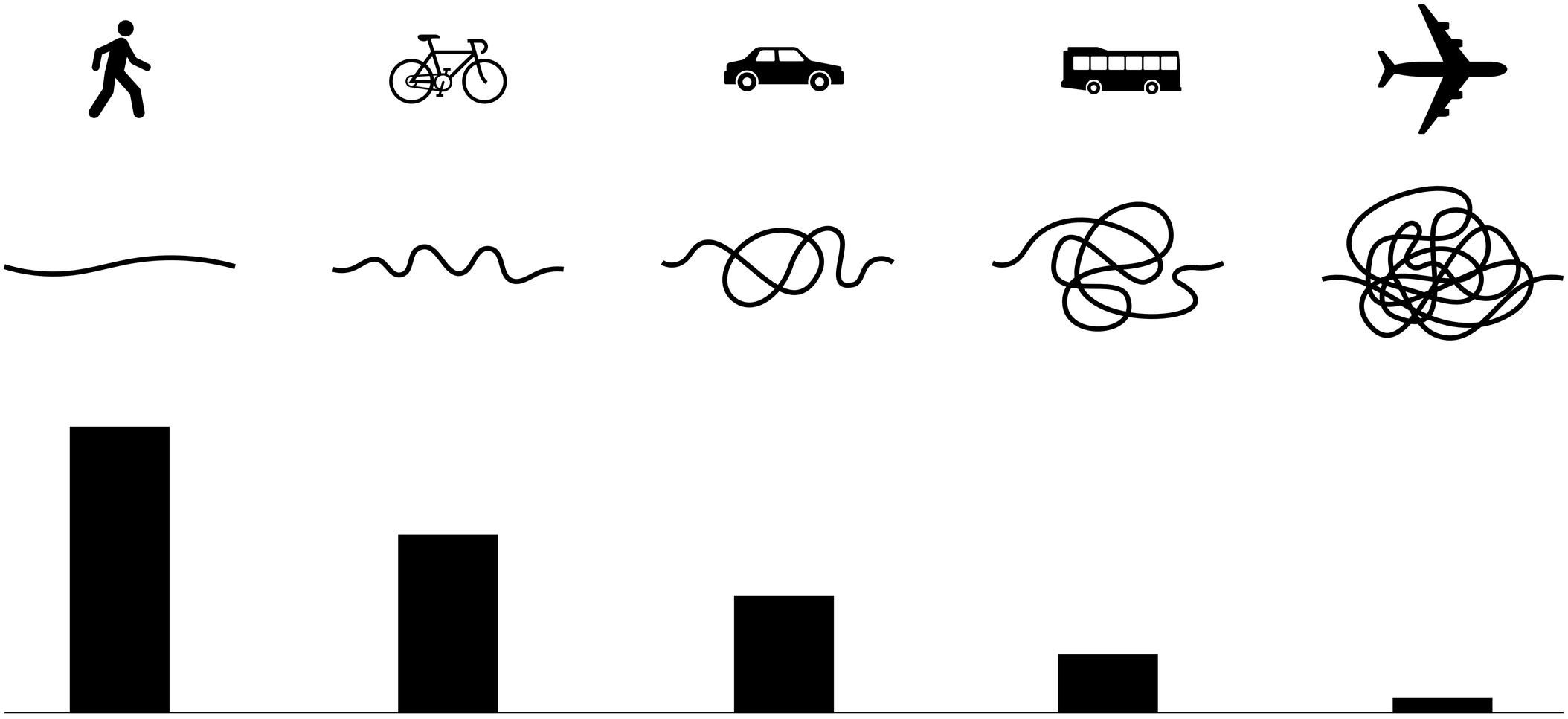
comfortable, pleasant

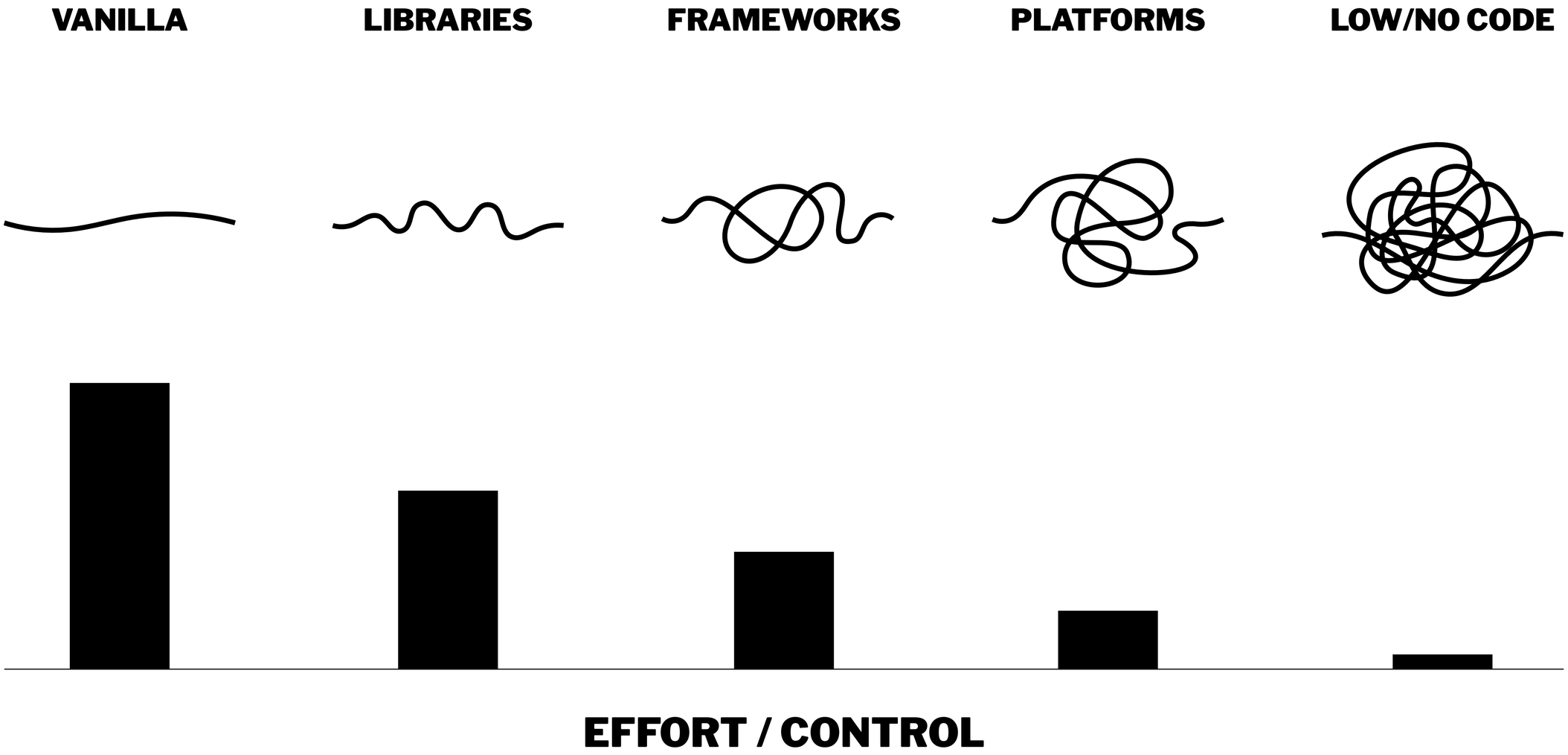


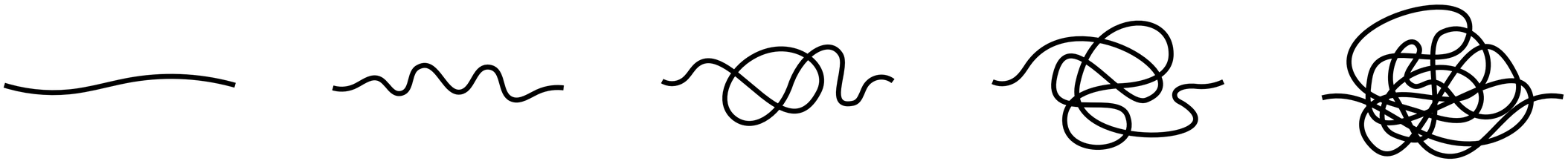
HOW SOMETHING IS USED


DIFFICULT
EASY
EFFORT


EFFORT


EFFORT / CONTROL


VANILLA
LIBRARIES
FRAMEWORKS
PLATFORMS
LOW/NO CODE
EFFORT / CONTROL



IS COMPLEXITY GOOD OR BAD ?

"KEEP IT SIMPLE"

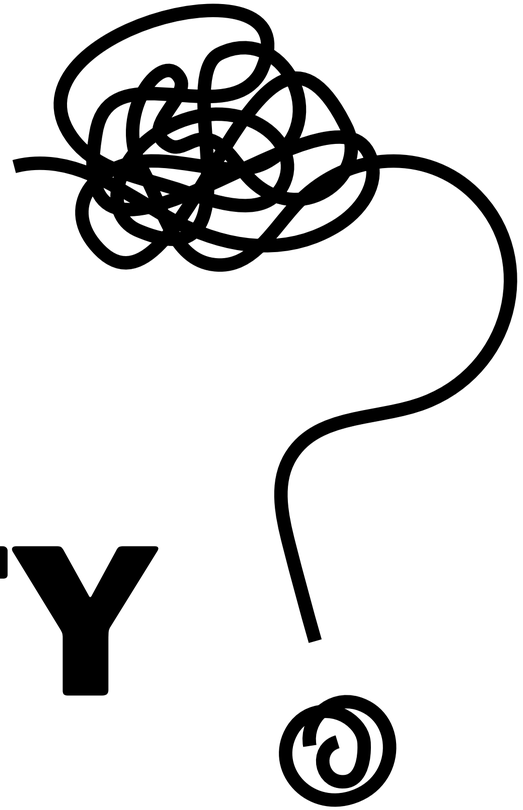
UNNECESSARY

COMPLEXITY

UNNECESSARY !== INHERENT

COMPLEXITY

WHAT IS INHERENT
COMPLEXITY



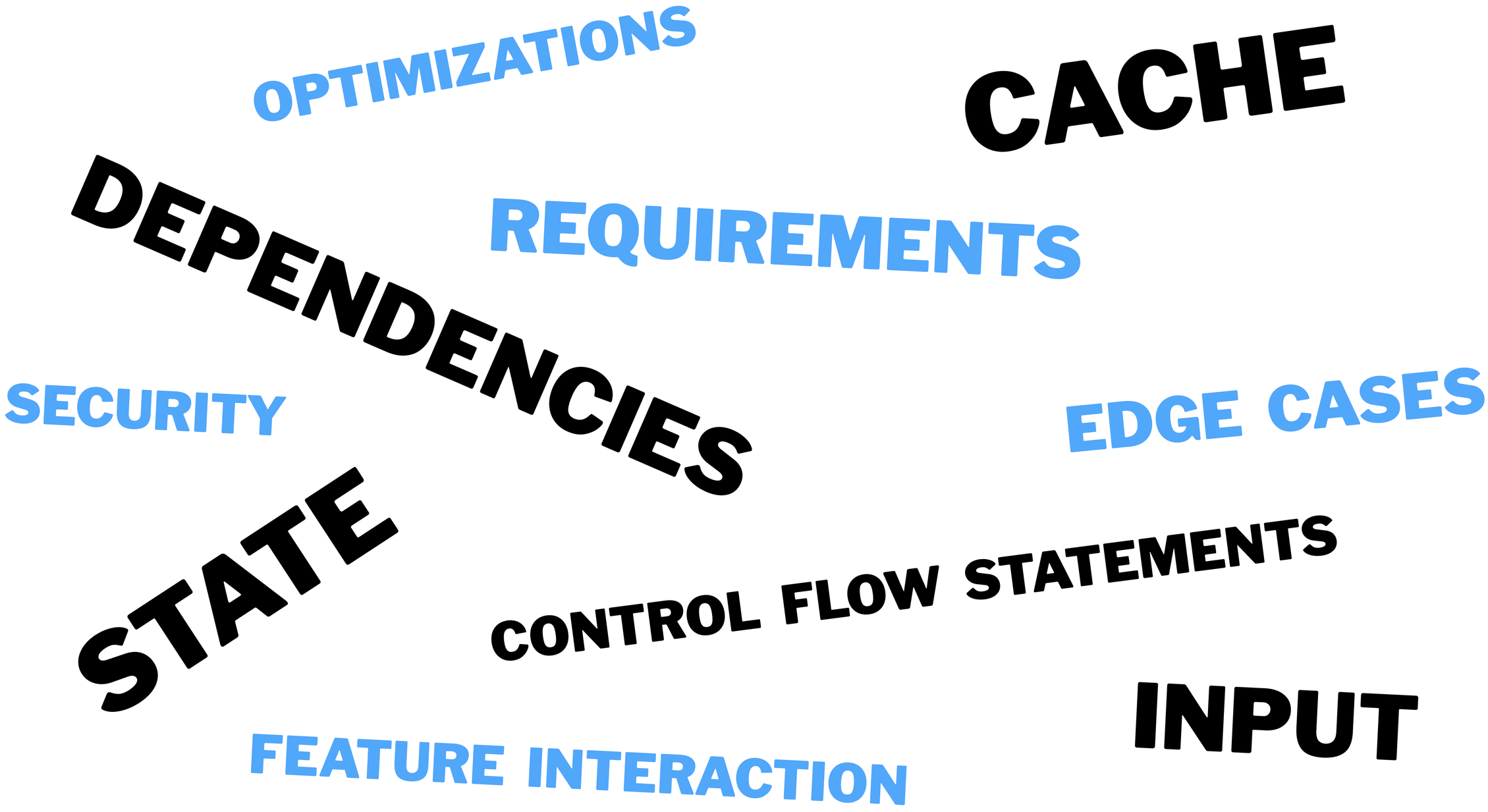
DEPENDENCIES

CACHE

STATE

CONTROL FLOW STATEMENTS

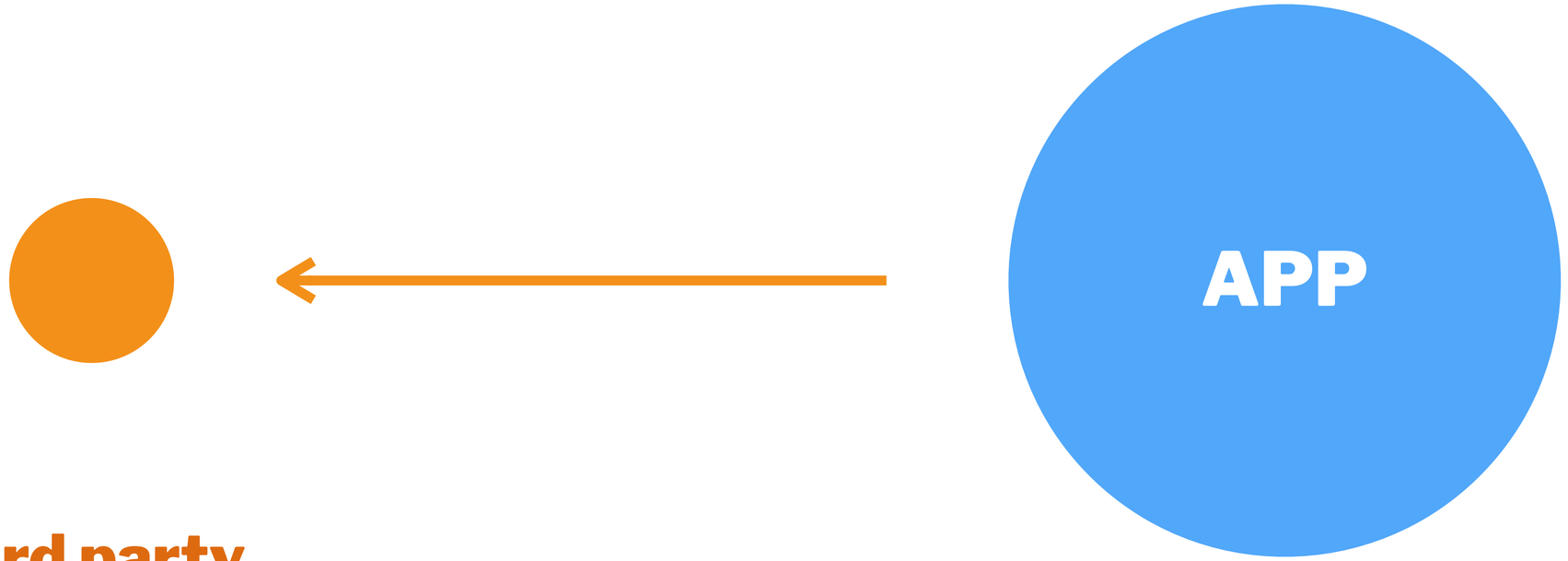
INPUT


OPTIMIZATIONS
CACHE
DEPENDENCIES
REQUIREMENTS
SECURITY
EDGE CASES
STATE
CONTROL FLOW STATEMENTS
INPUT
FEATURE INTERACTION


Useless
Box
ON
OFF

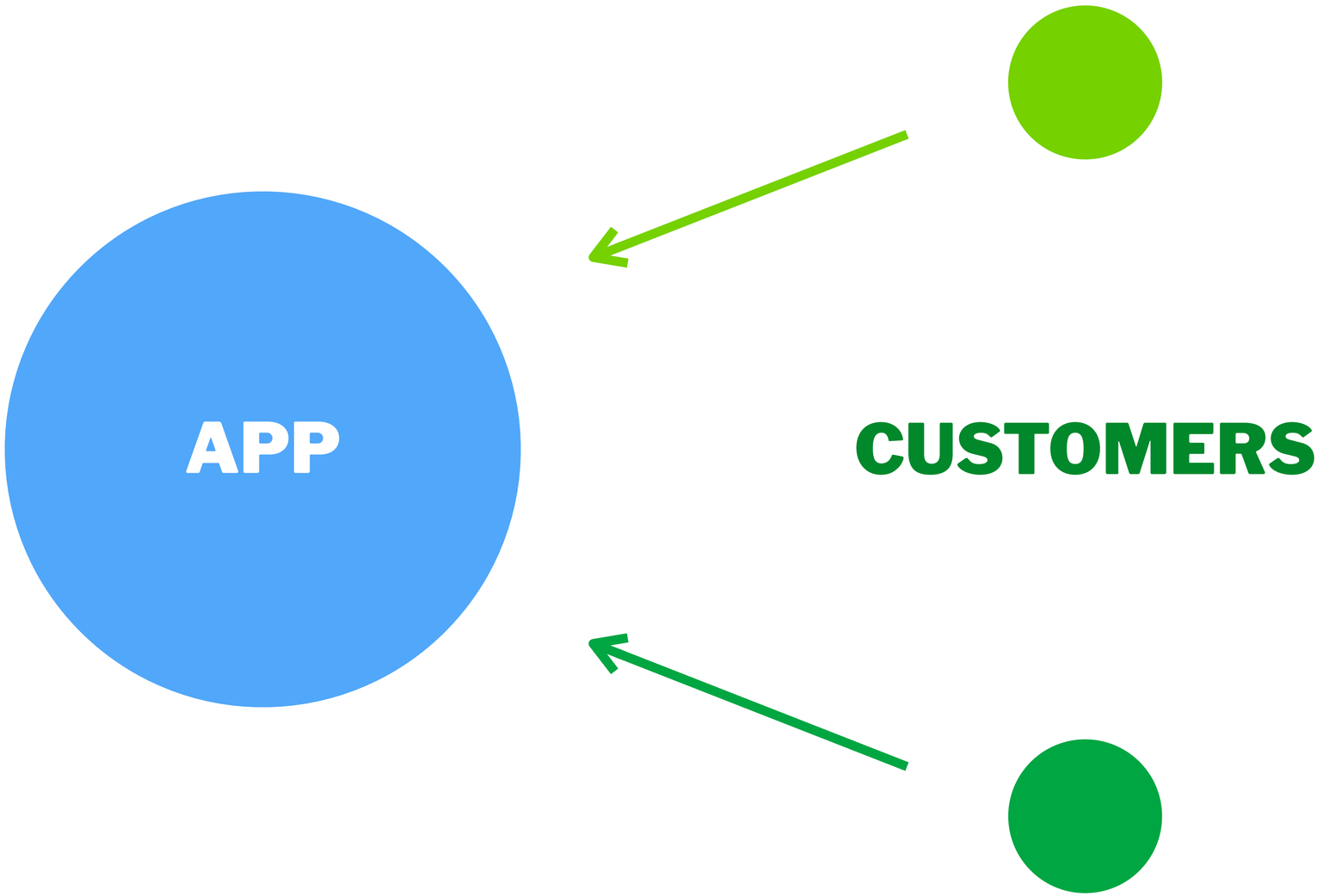
AUTHORS

AUTHORS CONSUMERS

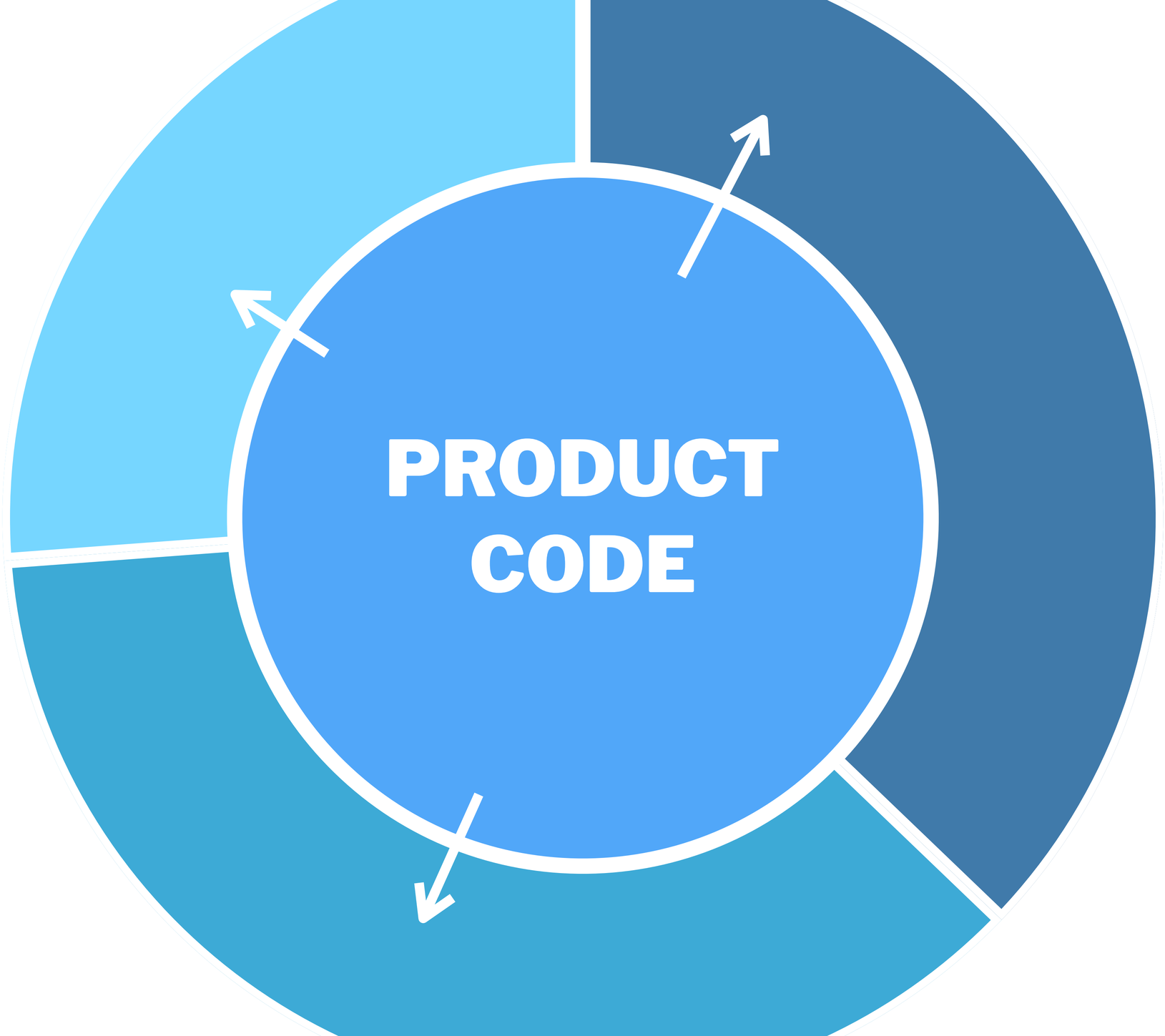


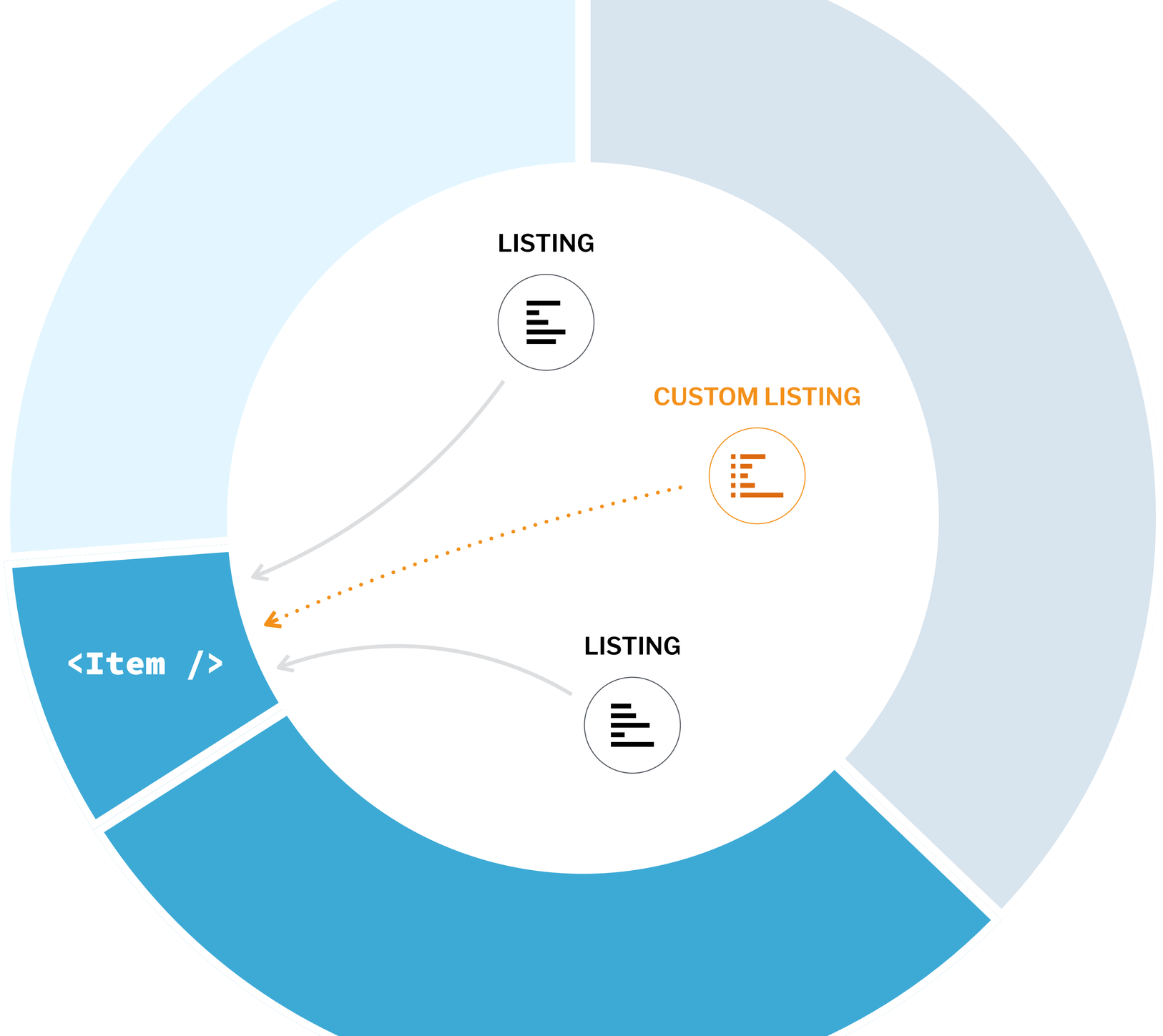
3rd party

DEPENDENCIES


APP
CUSTOMERS


APP


PRODUCT
CODE


LISTING
CUSTOM LISTING
<Item />
LISTING

`<Item />`

Content

Content

```ts
interface Item {
  body: string;
}
```

AUTHOR / IMPLEMENTATION

Content

```ts
interface Item {
  body: string;
}
```

AUTHOR / IMPLEMENTATION

CONSUMER / USAGE

Content

```tsx
<Item body="Content" />
```

```ts
interface Item {
  body: string;
  arrow?: boolean;
}
```

AUTHOR / IMPLEMENTATION

CONSUMER / USAGE

Content >

```tsx
<Item body="Content" arrow />
```

```ts
interface Item {
  body: string;
  icon?: 'arrow' | 'check';
}
```

AUTHOR / IMPLEMENTATION

CONSUMER / USAGE

Content	`<Item body="Content" icon="arrow" />`
Enabled	`<Item body="Enabled" icon="check" />`

```ts
interface Item {
  body: string;
  icon?: 'arrow' | 'check';
  count?: number;
}
```

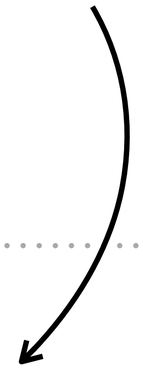
AUTHOR / IMPLEMENTATION

CONSUMER / USAGE

Content ›	`<Item body="Content" icon="arrow" />`
Enabled ✓	`<Item body="Enabled" icon="check" />`
Amount 5	`<Item body="Amount" count={5} />`

```ts
interface Item {
  body: string;
  icon?: 'arrow' | 'check';
  count?: number;
}
```

complex implementation
easy to use, but low control

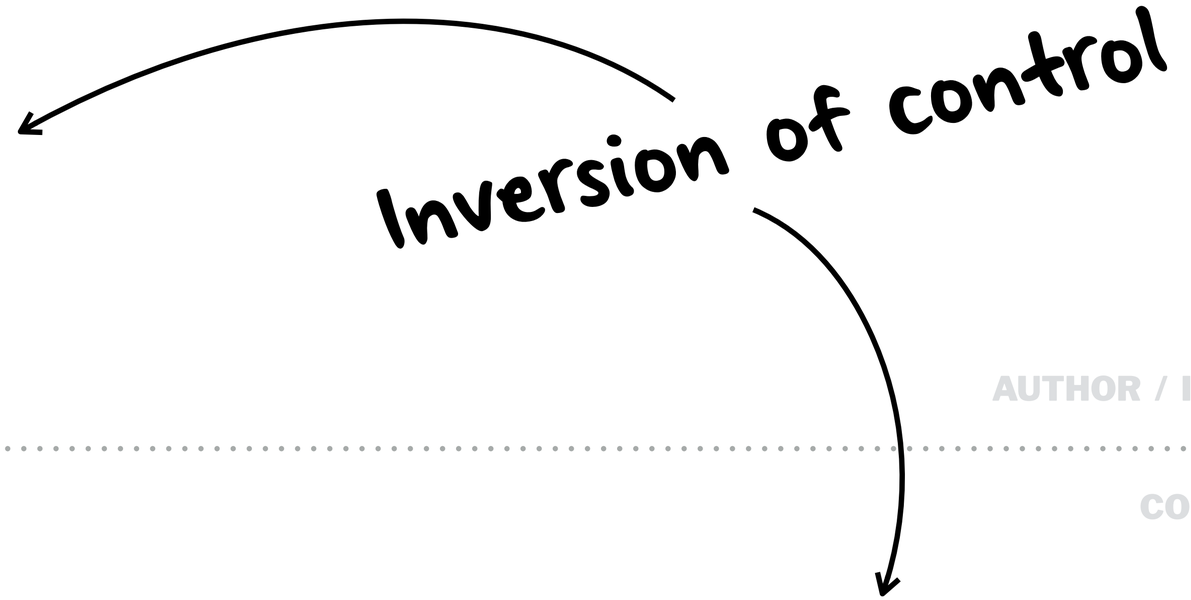


AUTHOR / IMPLEMENTATION

CONSUMER / USAGE

Content >	`<Item body="Content" icon="arrow" />`
Enabled ✓	`<Item body="Enabled" icon="check" />`
Amount 5	`<Item body="Amount" count={5} />`

```tsx
interface Item {
  body: string;
  extra?: React.ReactChild;
}
```



AUTHOR / IMPLEMENTATION

CONSUMER / USAGE

Content >	`<Item body="Content" extra={<Icon name="arrow" />} />`
Enabled ✓	`<Item body="Enabled" extra={<Icon name="check" />} />`
Amount 5	`<Item body="Amount" extra={<Badge count={5} color={BLUE} />} />`

```tsx
// complex implementation

interface Item {
  body: string;
  icon?: 'arrow' | 'check';
  count?: number;
}
```

📉 easy to use, low control
👍 great for re-usability

```tsx
<Item body="Amount" count={5} />
```

```tsx
// simple implementation

interface Item {
  body: string;
  extra?: React.ReactChild;
}
```

📈 high effort & control
👍 great for customisation

```tsx
<Item body="Amount" extra={
  <Badge count={5} color={BLUE} />
} />
```

```ts
// complex implementation

interface Item {
  body: string;
  icon?: 'arrow' | 'check';
  count?: number;
}
```

📉 easy to use, low control
👍 great for re-usability

```tsx
<Item body="Amount" count={5} />
```

Interface merge

```ts
// simple implementation

interface Item {
  body: string;
  extra?: React.ReactChild;
}
```

📈 high effort & control
👍 great for customisation

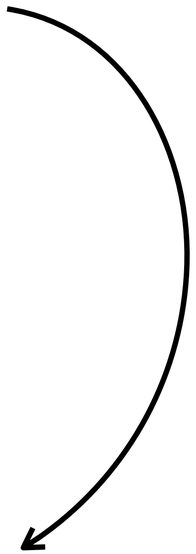
```tsx
<Item body="Amount" extra={
  <Badge count={5} color={BLUE} />
} />
```

```tsx
interface Item {
  body: string;
  icon?: 'arrow' | 'check';
  count?: number;
  extra?: React.ReactChild;
}
```

```tsx
<Item body="Amount" count={5} />
```

```tsx
<Item body="Amount" extra={
  <Badge count={5} color={BLUE} />
} />
```

```tsx
interface Item {
  body: string;
  icon?: 'arrow' | 'check';
  count?: number;
  extra?: React.ReactChild;
}
```



Interface segregation

```tsx
<Item body="Amount" count={5} />
```

```tsx
<Item body="Amount" extra={
  <Badge count={5} color={BLUE} />
} />
```

```ts
interface BaseItem {
  body: string;
  extra?: React.ReactChild;
}
```

```ts
  icon?: 'arrow' | 'check';
```

```ts
  count?: number;
```

```ts
interface BaseItem {
  body: string;
  extra?: React.ReactChild;
}
```

```ts
interface IconItem {
  body: string;
  icon?: 'arrow' | 'check';
}
```

```ts
interface CountItem {
  body: string;
  count?: number;
}
```

```tsx
<IconItem body="Enabled" icon="check" />
```

```tsx
<CountItem body="Amount" count={5} />
```

TO CONCLUDE

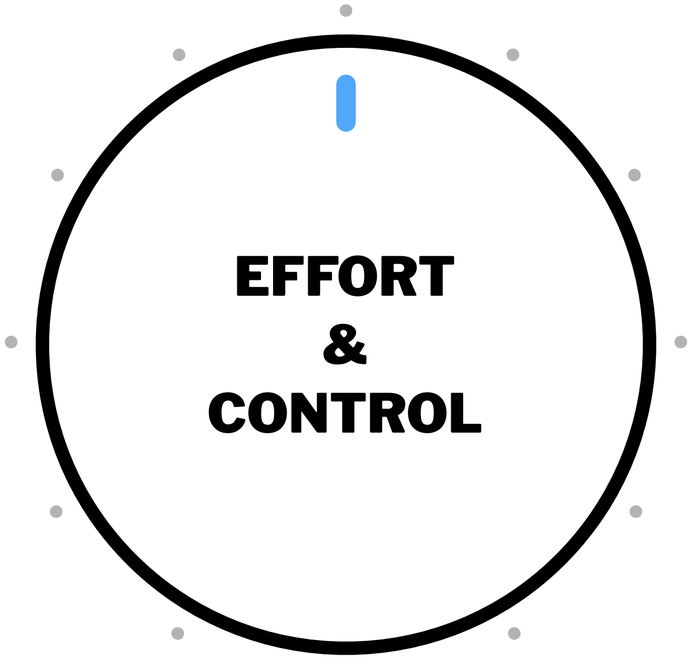
WE ARE BOTH

AUTHORS AND CONSUMERS

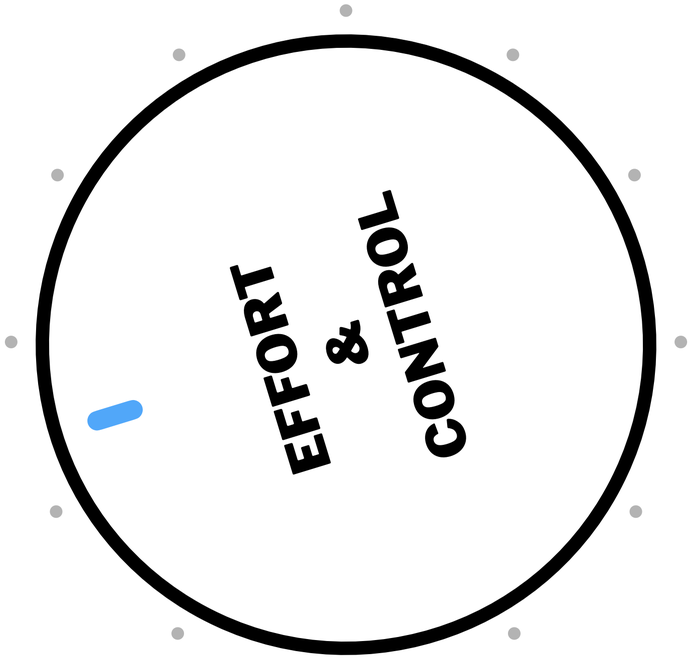
COMPLEXITY
IS INHERENT

COMPLEXITY => EFFORT

EFFORT === CONTROL



CONSUMER IMPLEMENTATION

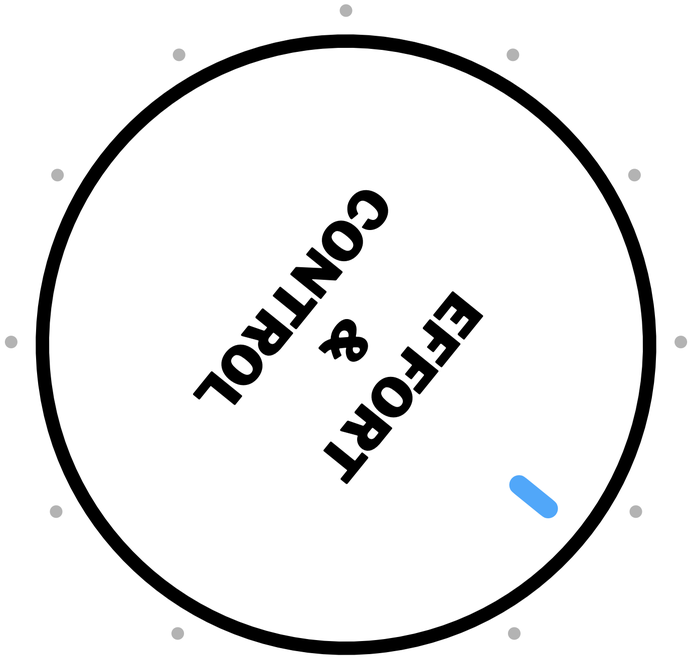


CONSUMER

IMPLEMENTATION

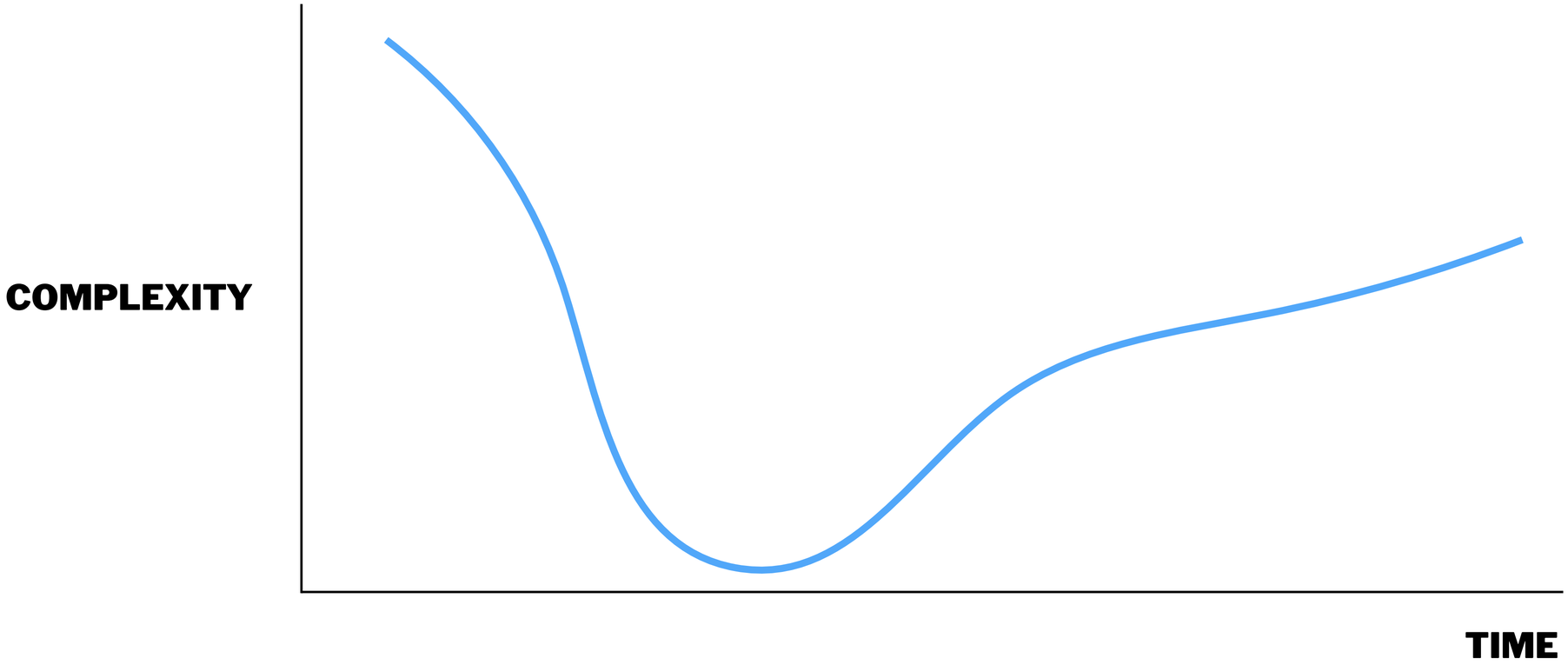
CONSUMER

IMPLEMENTATION



CONSUMER

IMPLEMENTATION


COMPLEXITY
TIME

TESLER'S LAW

Law of conservation of complexity

TESLER'S LAW

Every application has an inherent amount of complexity that cannot be removed or hidden.

Instead, it must be dealt with, either in product development or in user interaction.

THANK YOU

andreipfeiffer.dev