What sucks about Clojure…and why you'll love it anyway

Chas Emerick

@cemerick
http://cemerick.com

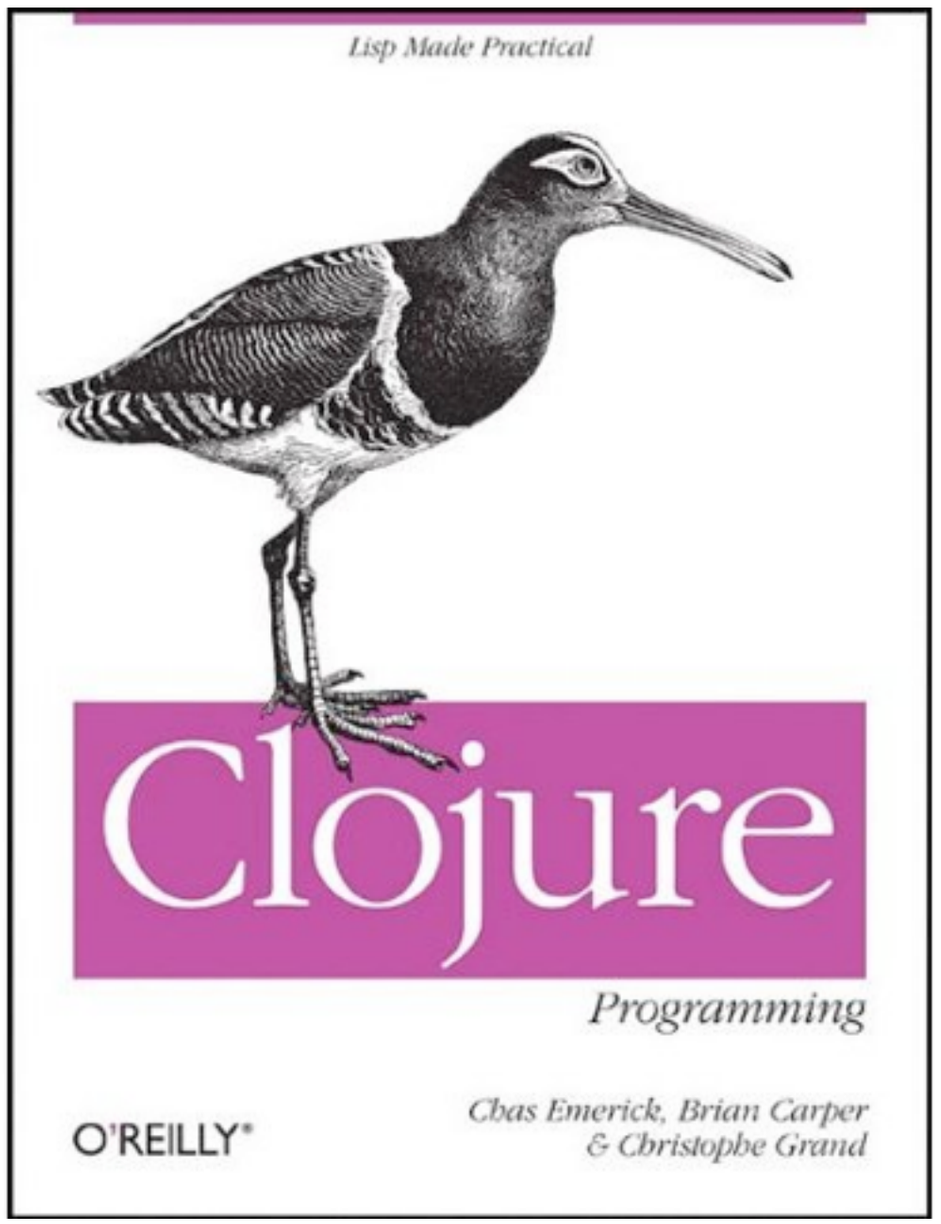
@cemerick

Founded Snowtide in 2001

Clojure since 2008

Contributor to core language & libraries

Coauthor of *Clojure Programming* from O'Reilly



Objectives

- Present a fair critique of Clojure from a pragmatic user's perspective
- What will you wish you knew now?

…and, escape the building alive…

In many ways, the work of a critic is easy.

We risk very little, yet enjoy a position over those who offer up their work and their selves to our judgment. We thrive on negative criticism, which is fun to write and to read.

But the bitter truth we critics must face is that in the grand scheme of things, the average piece of junk is probably more meaningful than our criticism designating it so. But there are times when a critic truly risks something and that is in the discovery and defense of the new.

The world is often unkind to new talent, new creations. The new needs friends.

— Anton Ego (the food critic in Pixar's *Ratatouille*)

Your favourite programming language is not good enough

— Fredrik Håård

http://blaag.haard.se/Your-favourite-programming-language-is-not-good-enough/

Namespaces

- `ns` / `load-file` / `require` / `refer` `refer-clojure` / `use` / `import`
- `(require '…)` vs. `(:require …)`
- No `(:import com.package.*)`
- `(def f ns/g)` vs. `(def f #'ns/g)`
- `(import foo.AType)` fails if `'foo` hasn't been loaded

Namespaces

- No way to build namespaces incrementally: “Make namespace X have the same aliases/refers/imports as Y, plus …”
- all namespace operations are risky and side-effecting
  - if only loading code was transactional

declare

```
(defn main [args] (helper args))
(defn helper [[x y z]] ...)
```

- no forward references
- one of the few places where the programmer has to babysit the compiler
- unintuitive compilation unit

Dynamic scope

```
(with-foo {...} (f a b c))

(f {...} a b c)
```

- every dynamic var is an implicit argument to every fn
- binding conveyance
- (almost) considered harmful
- Good: auxiliary returns

Using STM (effectively) is hard

- Appears easy
- In-memory database implementation with *superior* Clojure integration
- How to find optimal ref granularity?
- nondeterminism
- STM is overused

You will read Clojure's source

- Chunked seqs
- `:inline` and `definline`
- `clojure.core` et al.

The JVM

+ Lisp …

= Clojure

http://machinegestalt.posterous.com/if-programming-languages-were-cars

The JVM

- startup time
- the library myth
  - need to have a refined palate
- saving graces
  - perf
  - reliable operation for long-running apps
  - distribution

AOT Compilation

- Bloats artifacts
- Compilation is transitive through all dependencies
- Introduces backward- and forward-compatibility risk
- If necessary, limit to final app delivery

(((OMG the parens)))

- homoiconicity ⇒ macros, but they're rarely what you want
  - macros are too easy
- "shouldn't need to write in ASTs in order to modify them"
  - Groovy transforms, Template Haskell, Scala compiler plugins
  - homoiconicity doesn't simplify many aspects of metaprogramming
    - e.g. &env, &form, metadata, :inline
- Cannot produce DSLs that Ruby/Scala/Groovy can

“Over-enthusiasm”

- Clojure everywhere
  - pervasive reader
    - sexps aren’t always superior
      - no binary representation
    - mother slow
    - painful integration compared to JSON or XML

Big ball of mud

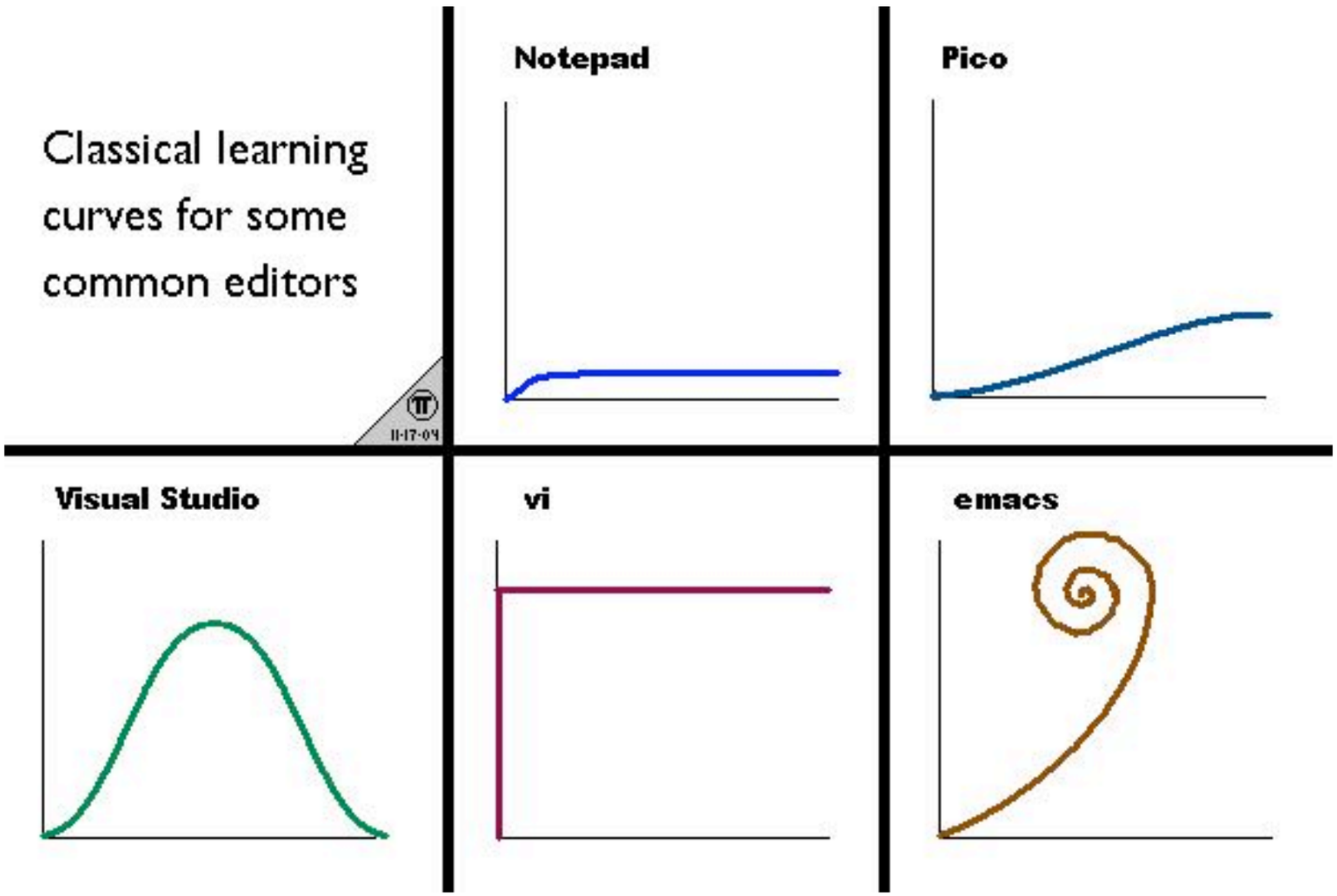
```
(require '[clojure.string :as str])

(def camel->keyword
  (comp keyword
    (partial str/join \-)
    (partial map str/lower-case)
    #(str/split % #"(?<=[a-z])(?=[A-Z])")))

(camel->keyword "CamelCase")
;= :camel-case
(camel->keyword "lowerCamelCase")
;= :lower-camel-case
```

Big ball of ~~mud~~

What sucks about Clojure…and why you'll love it anyway

Snowtide

Learning curves

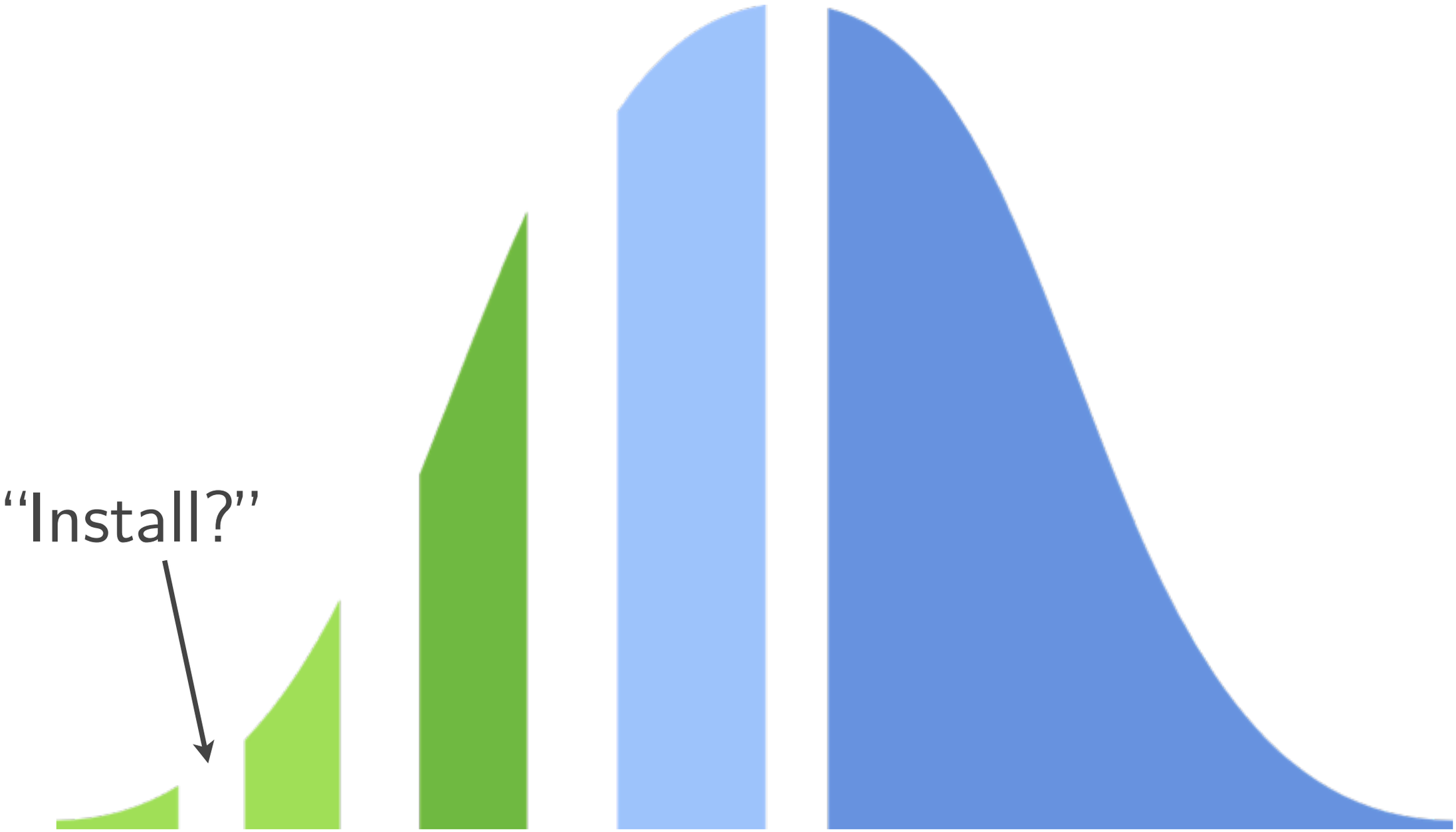


Crossing the Chasm(s)

What sucks about Clojure…and why you'll love it anyway

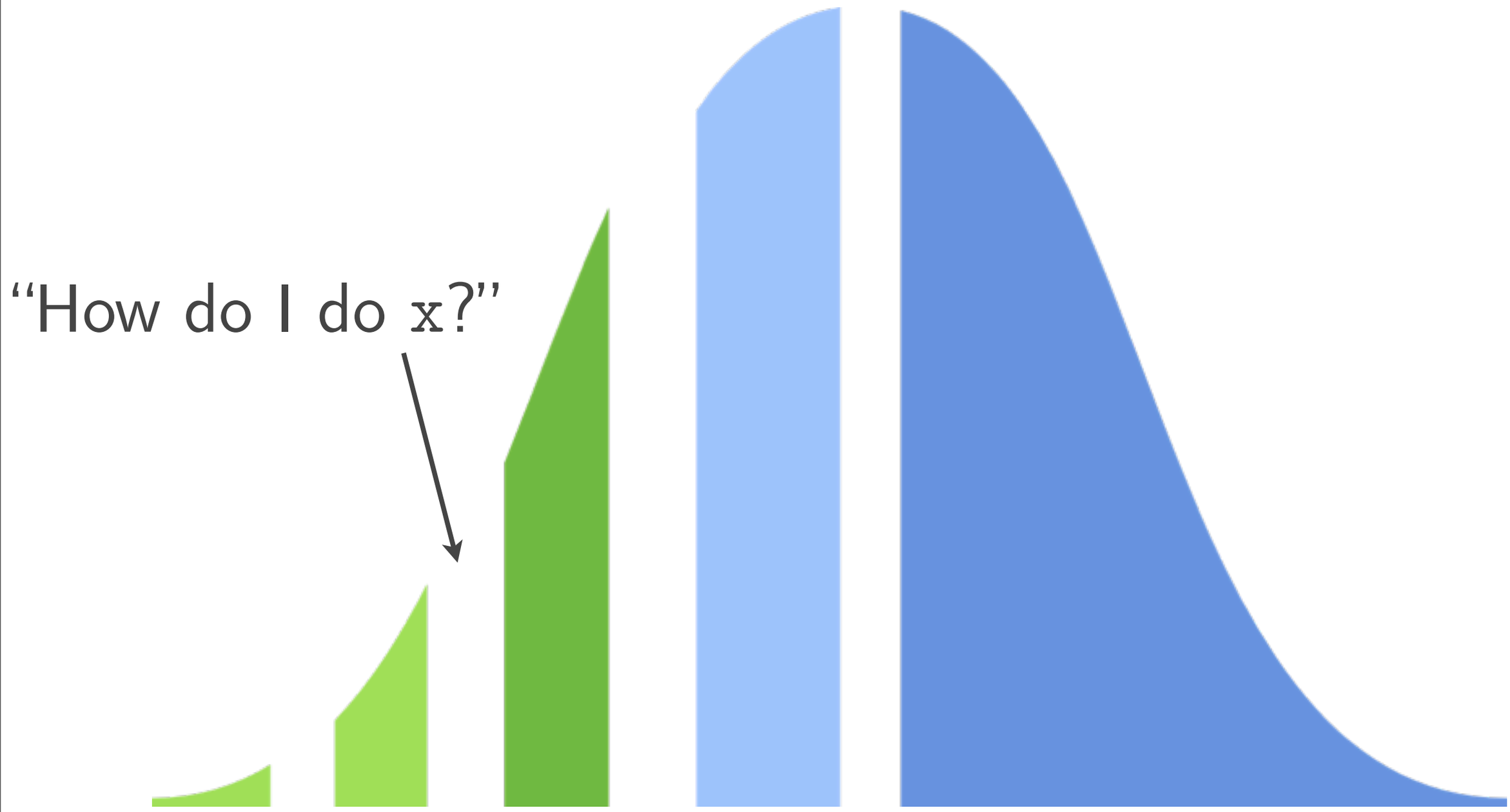
Snowtide

Crossing the Chasm(s)

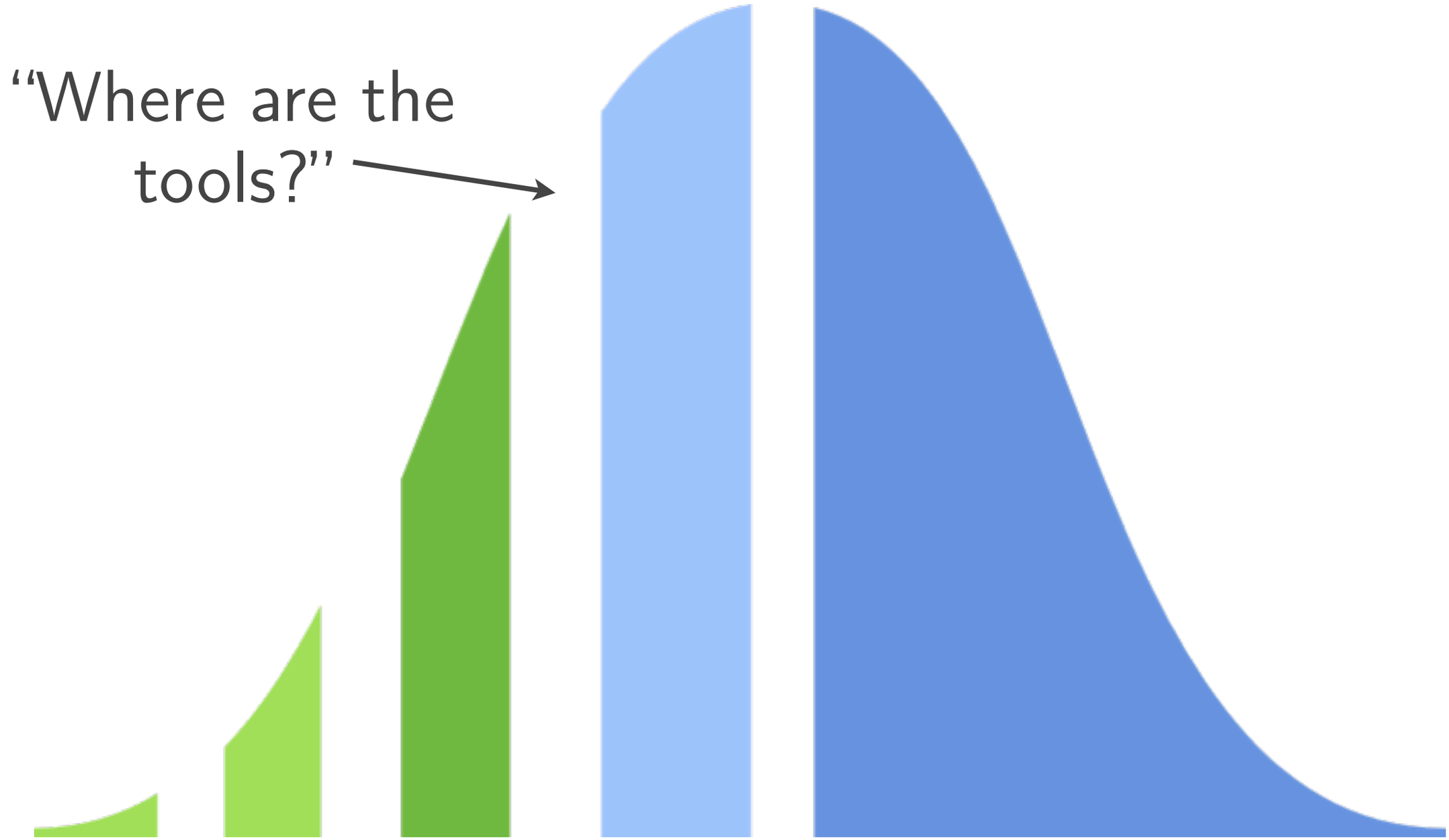


Snowtide

Crossing the Chasm(s)

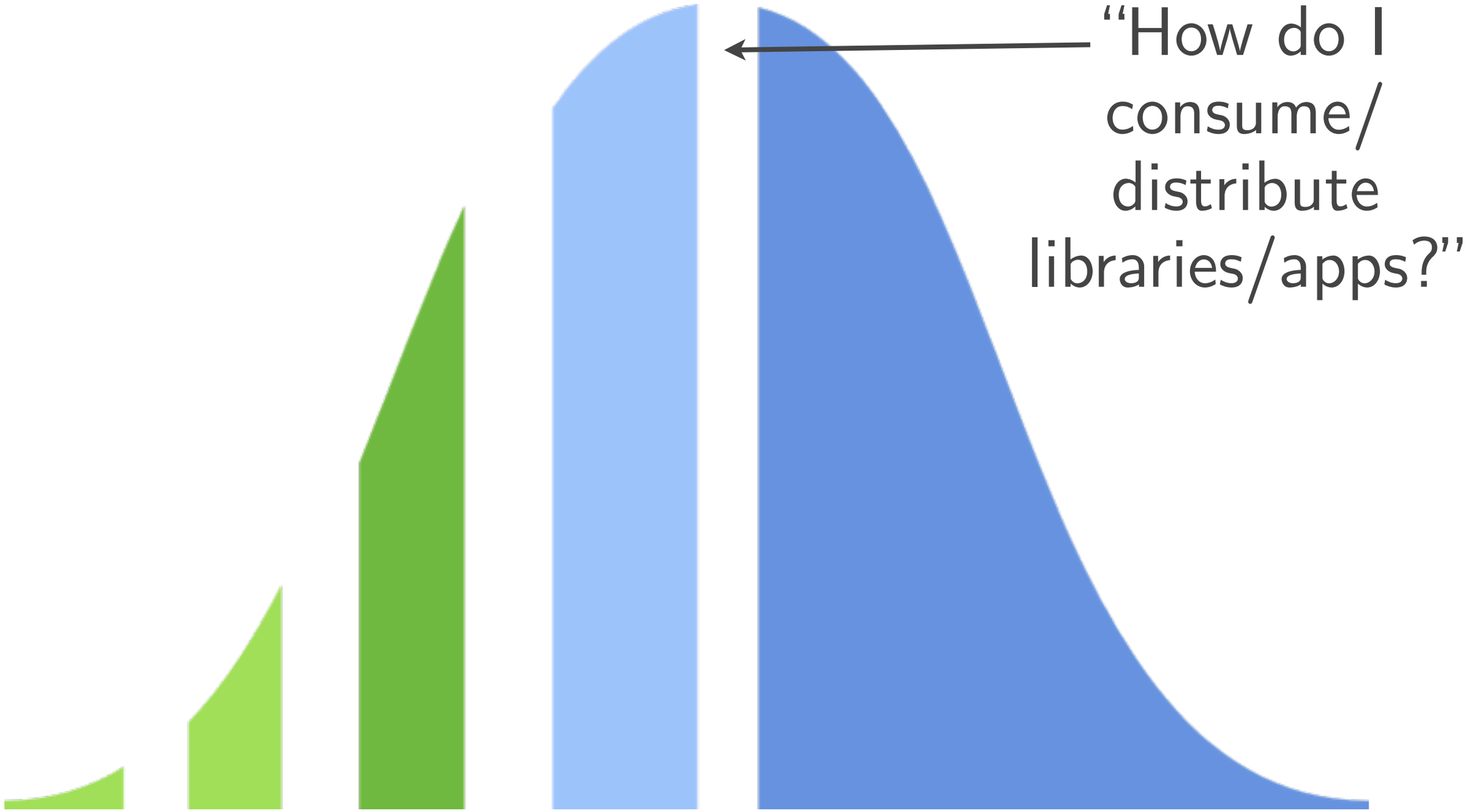


Crossing the Chasm(s)

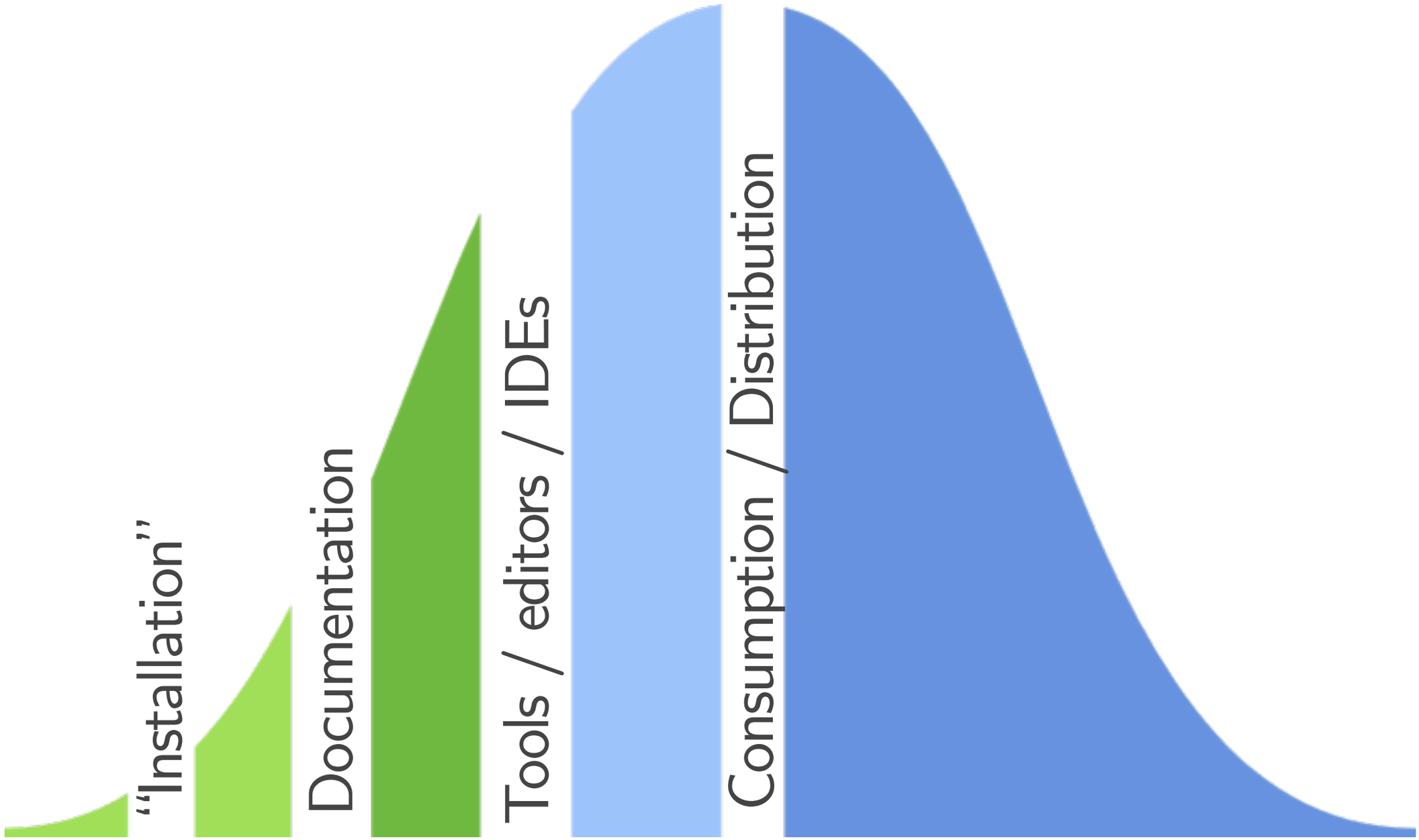


Snowtide

Crossing the Chasm(s)



Crossing the Chasm(s)



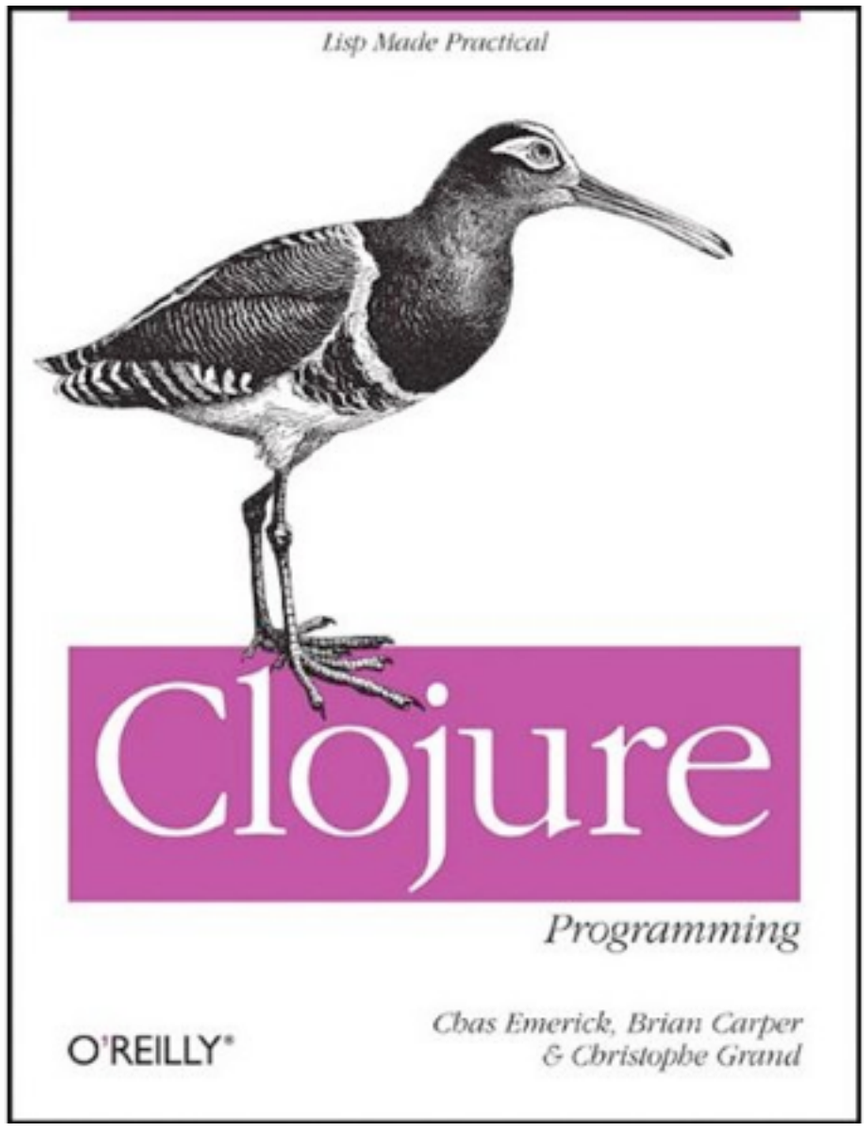
Bus Factors

- Clojure's continues to be 1
  - viz. Mark Pilgrim, _why
  - cf. Apache CouchDB's bus factor is N
    - Flipping founder (Damien Katz)

Bus Factor Mitigation

- EPL, killer open source community
- The ideas in Clojure & their composition are more important than any particular implementation
- Alternative implementations & forks:
  - ClojureScript
  - ClojureCLR
  - clojure-py
  - clojure-scheme
- *Community* of power and responsibility

Thank you!



@cemerick http://cemerick.com
@ClojureBook http://clojurebook.com

Snowtide