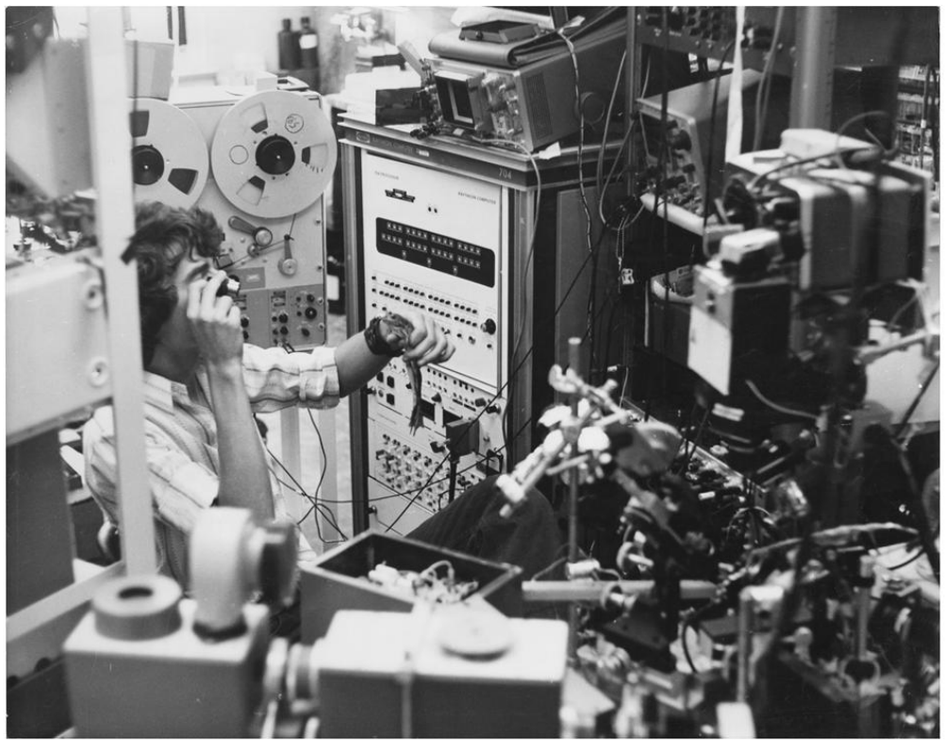
Communal Computing

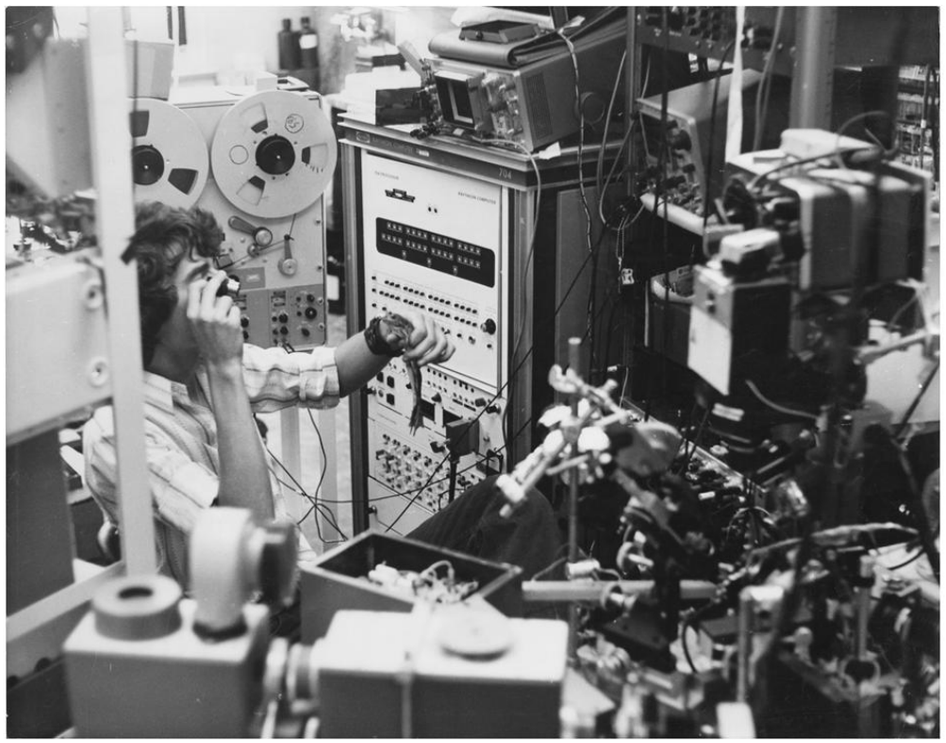
The top of the peer-to-peer stack

Describing “Single-player Computing”



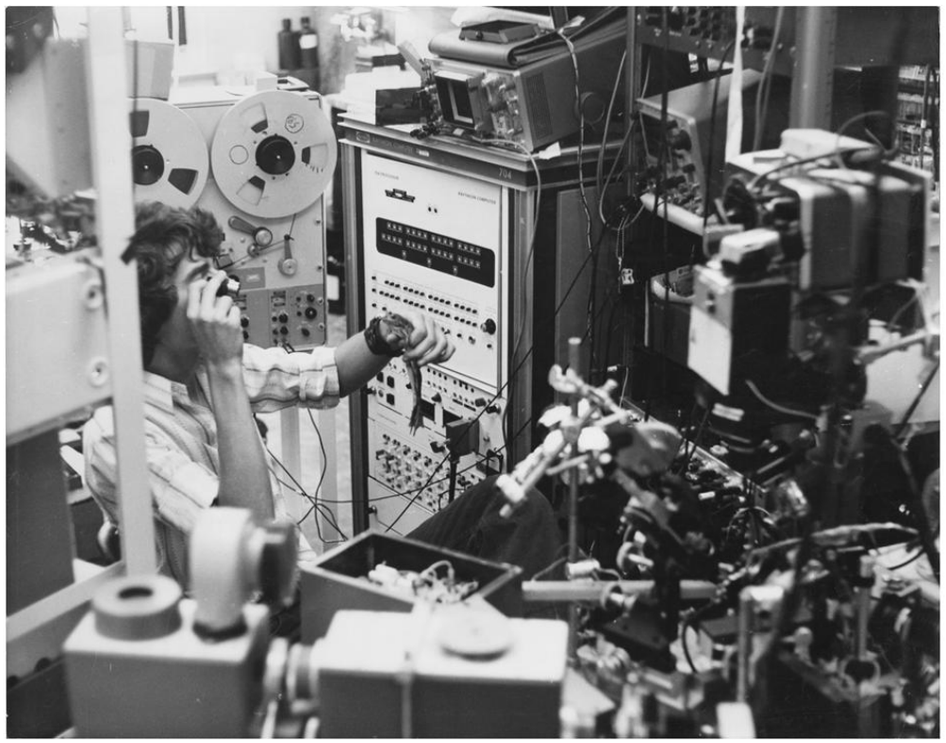
1. PCs inherit decisions made from designing Unix in the 1970s: Combination of desktop metaphor, timesharing

Describing “Single-player Computing”



2. Made for multiple users on one machine, centered around timesharing cycles across single-user sessions

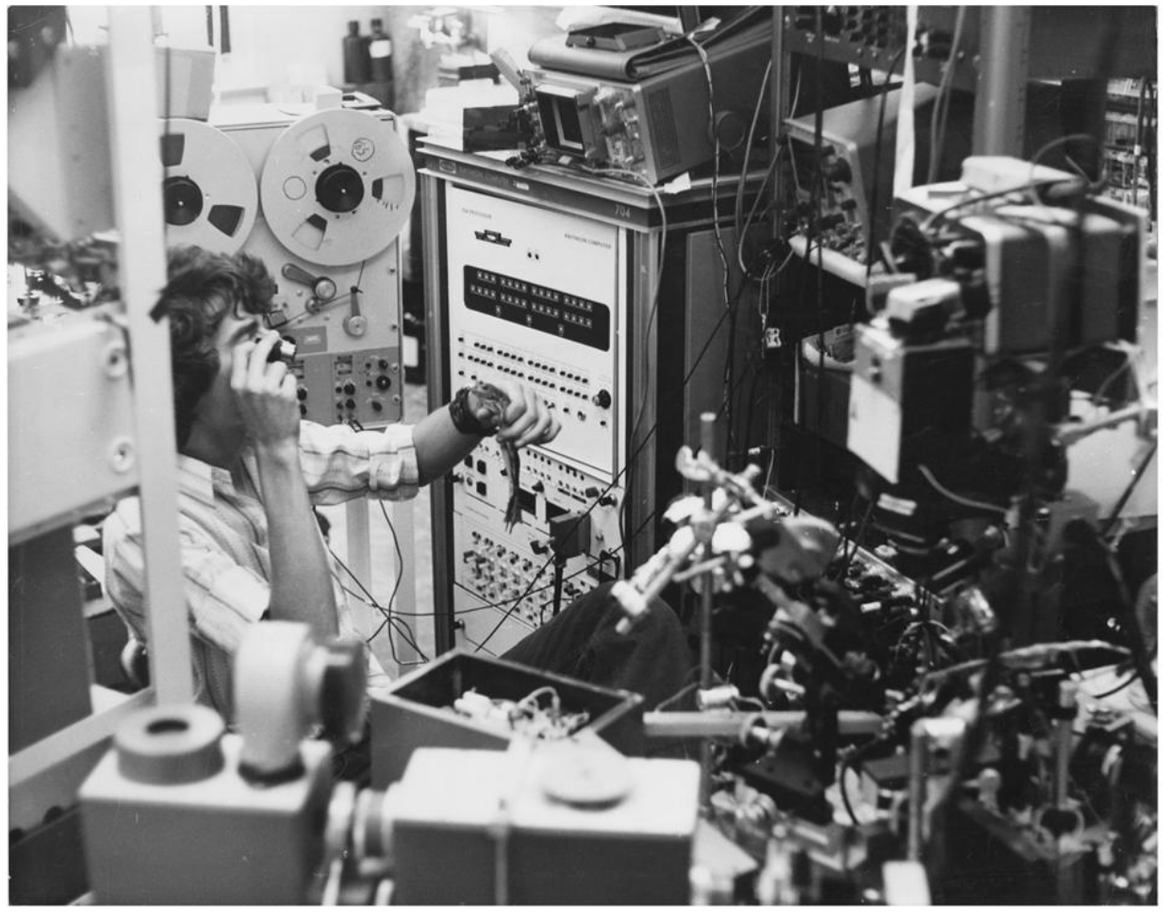
Describing “Single-player Computing”



3. This pattern also replicates
   in the client-server relationship

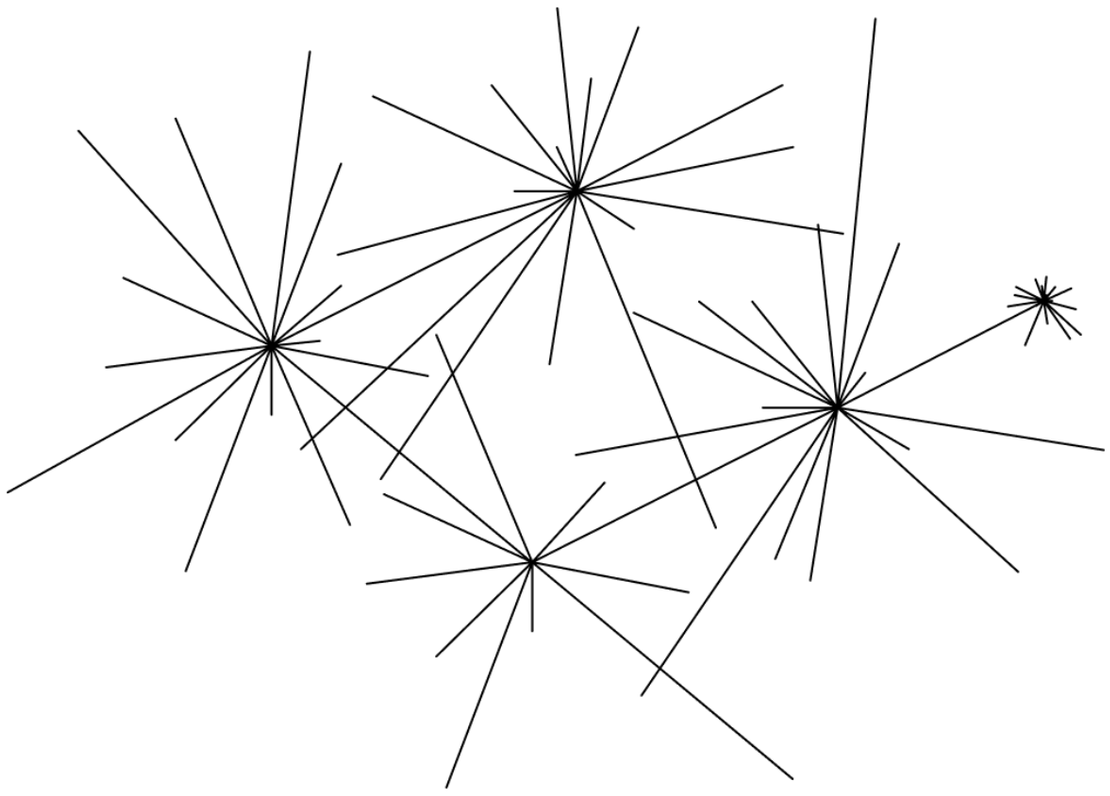
Describing “Single-player Computing”

- PCs inherit decisions made from designing Unix in the 1970s: combination of desktop metaphor, timesharing
- Made for multiple users on one machine, centred around timesharing cycles across single-user sessions
- This pattern also replicates in the client-server relationship



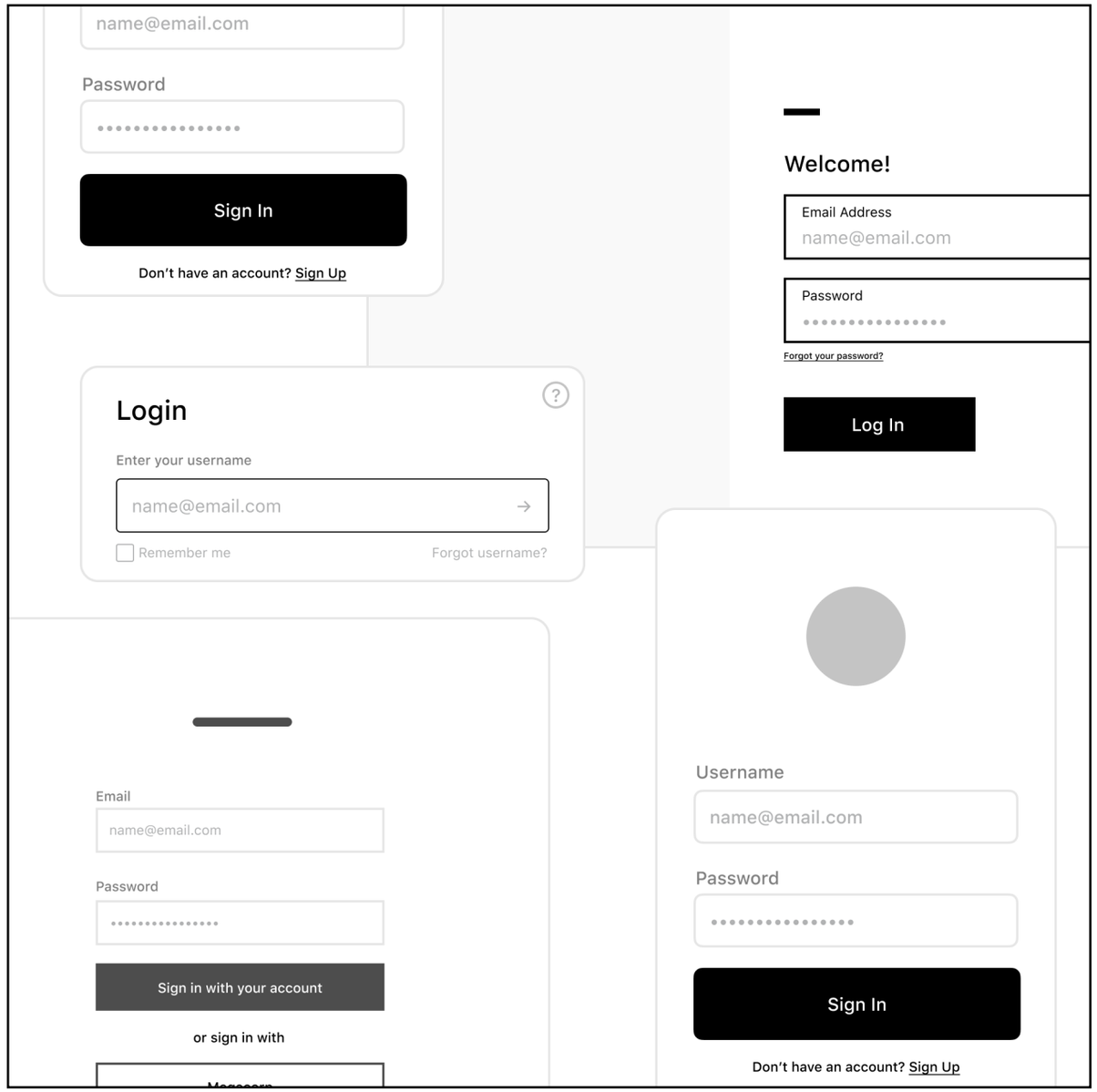
Decentralisation and Interface

- Decentralisation looks toward personal freedom, transparency, balanced power dynamics
- But decentralisation also demands its own conception of a user interface in turn: disregarded in favour of modules
- Modular approach is more pliable — allows for agency and freedom on a small scale
- If we are approaching redecentralisation and a P2P internet then the interface challenge is ahead of us



Centralisation and Interfaces

- If timesharing let many users use free cycles of a supercomputer and pretend it was theirs, online platforms reify this same model
- Thin clients to single-function computers
- Why is it so natural to think about having an “account”? Or to “log in”?
- Why has hardware veered toward thinner clients?



Decentralisation approaches so far 1/2

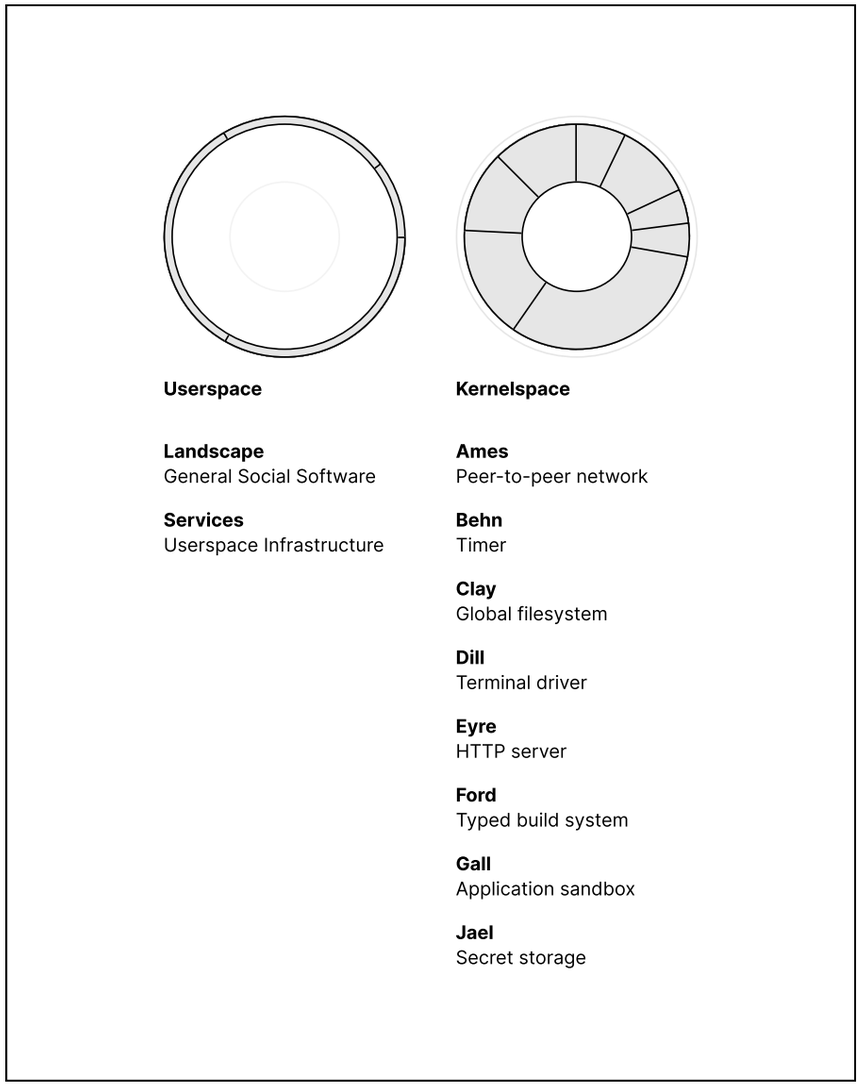
- When we try to combat this dynamic for everyday users — for groups of people — we settle for a federated model because **peer-to-peer has a high cost to the user**
- Mastodon: HTTP and DNS, trades off for a landlord model
- Scuttlebutt: fully peer-to-peer, creates identities only on the local machine

Decentralisation approaches so far 2/2

- The lesson we learn from these approaches is that **the internet, as a system, does not incentivise direct ownership**
- To a user, it is simply much easier to abstract one's online life into this model and accept the tradeoffs
- This is in direct opposition to the core values of the internet's earliest users

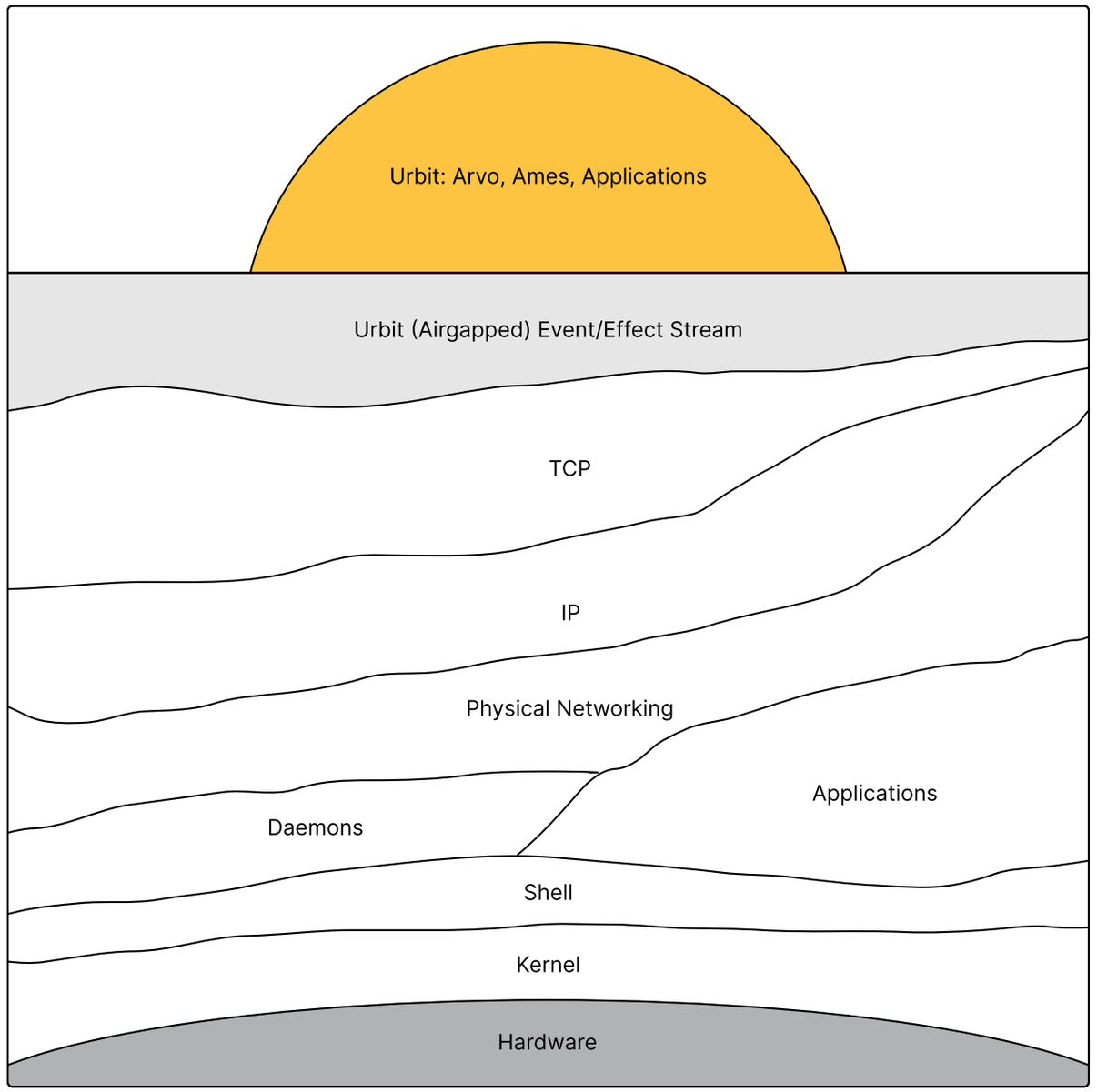
Systematically, a peer-to-peer internet

- I work on Urbit, which takes a systematic approach to constructing a peer-to-peer internet
- We believe in branching off computational history at a specific point and constructing the entire stack around modern use cases, human-scaled networks — while containing complexity



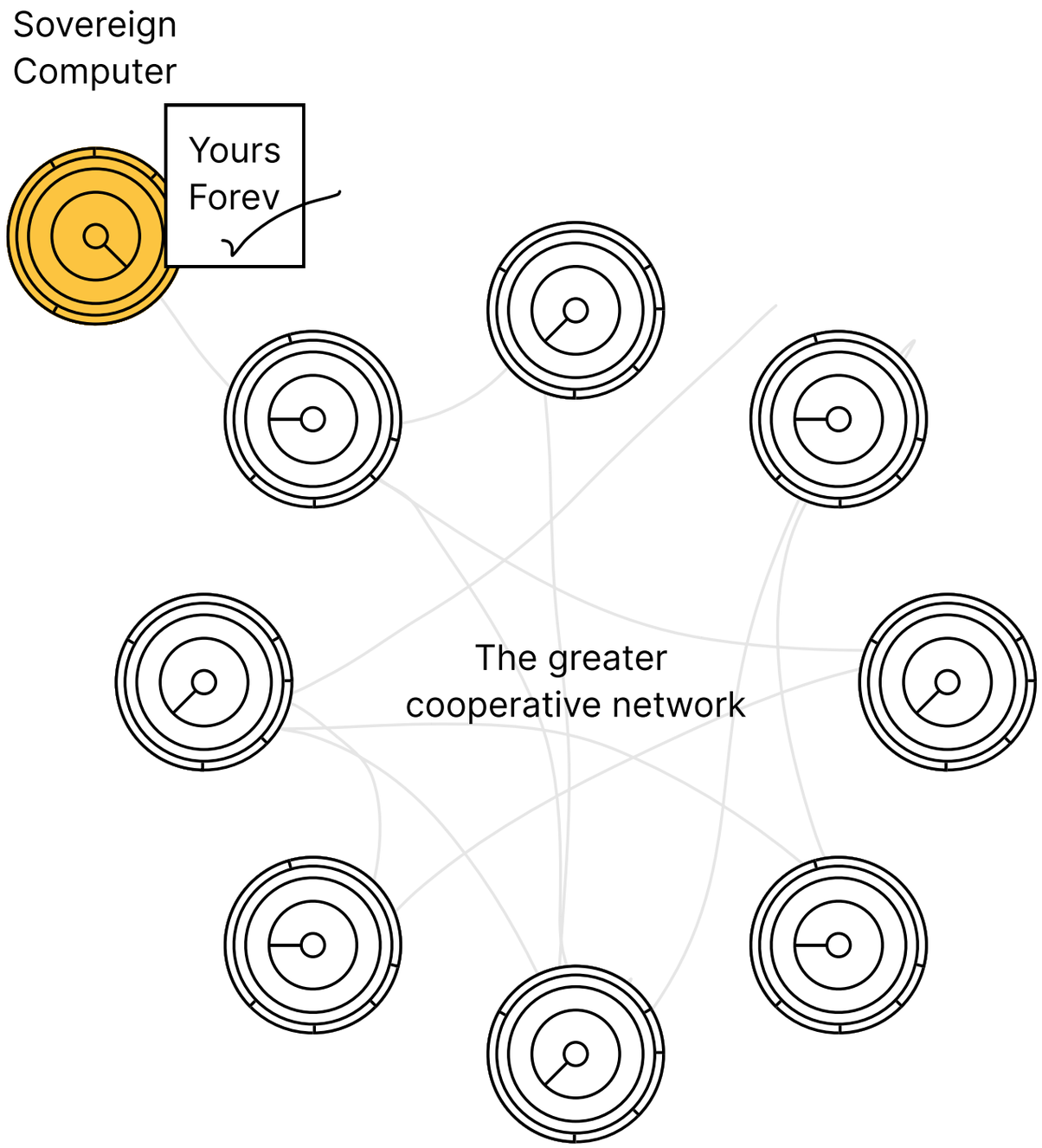
No clients, no servers

- The core goal: A new layer on top of the internet, routing machines built to talk directly
- If we can adjust the systematic incentives of the network to keep it friendly by default, even better



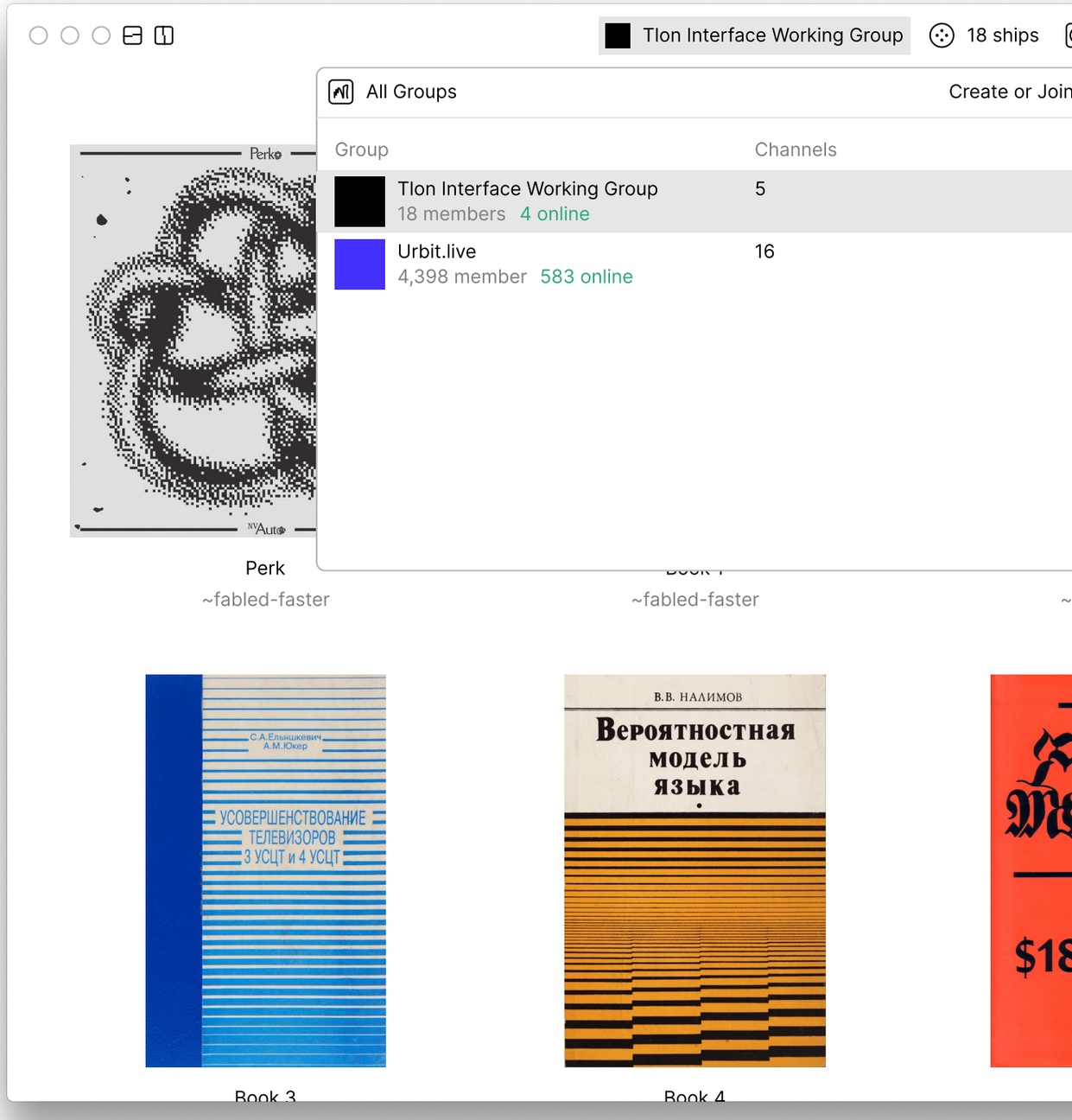
Interface

- If everyone is running their own computer and speaking directly to each other, you plan your UI around a wide net of small groups
- Centralised platforms are great for presenting and measuring **aggregate information** — this is difficult to verify across a swarm of peers
- You may even want to opt-out: your internet and your computer could just be the size of your life



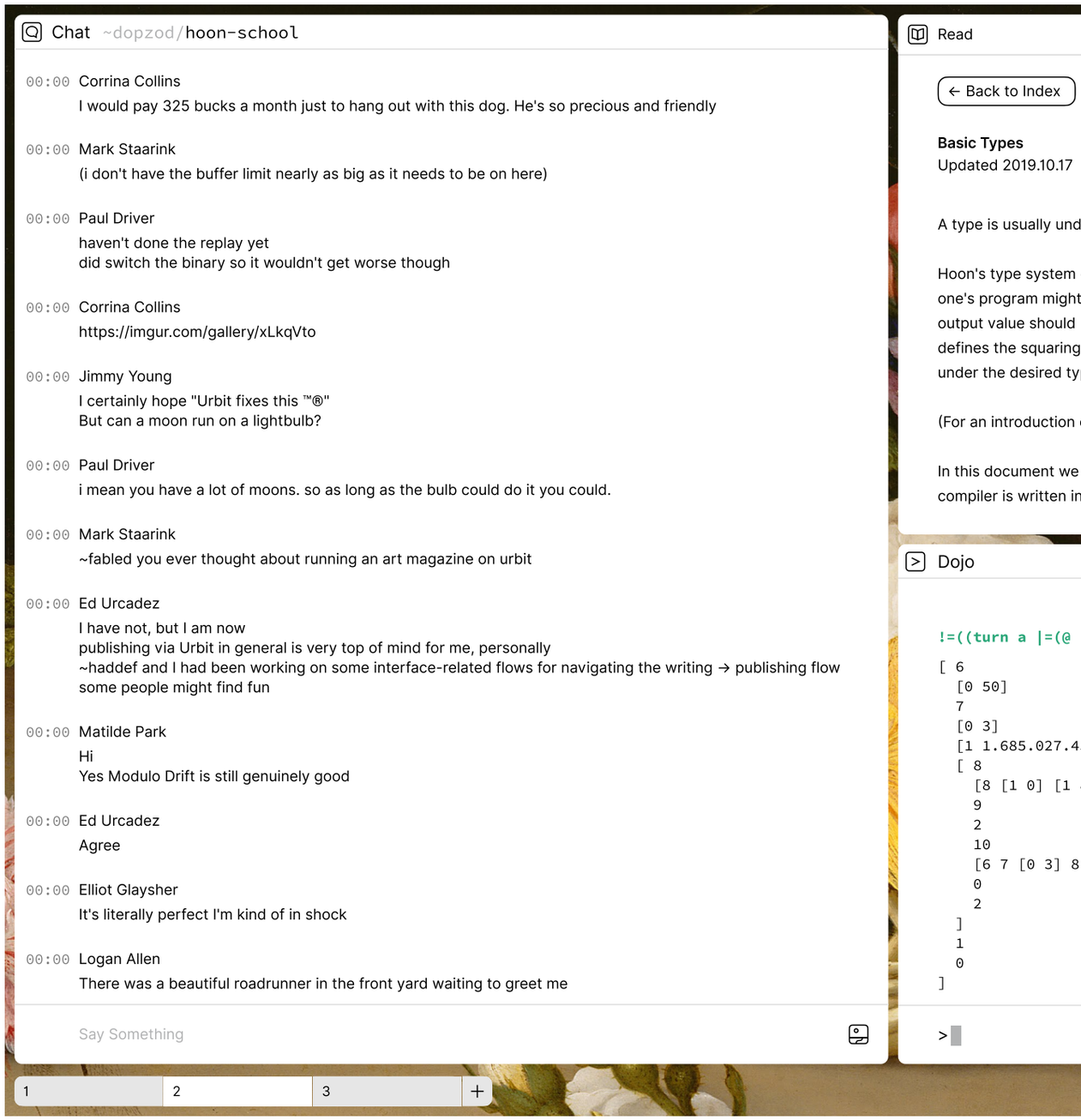
Communal Interfaces

- How do you build for a community outside the single-context application?
- How could this interface ever easily become a “product” as we currently imagine them?
- Conception of product revolves around single-function, account-segregated mega-computers
- Our current tack is imagining a “shared desktop” of “file contexts” and a shared directory



Defining one’s own UI components

- If you can’t guarantee everyone has the same UI, you permission and define read and write patterns within file types, **per group**
- This is defining the group’s shared data structure
- This is letting the individual (or the group) shape an inherited interface from an extensible component library



Designing Communal Components

- A peer-to-peer interface then demands:
- **Target-agnostic, extensible components** for file types
- within **specific shared application contexts**