yoric.github.io

Coding for a Finite World

David Teller

17–21 minutes

(This is meant to be the first entry of a series which will cover individual points more in depth. We’ll see how that goes.)

We’re the tech industry. We have ideas. We have ideas *all the time*. And we’re used to turn our ideas into applications.

So, how does it go… here’s the back-end component… here’s the front-end component. We’ll write the former in Python, or perhaps JavaScript, to optimize for prototyping. After all, we have *so many* ideas, we need the ability to iterate quickly. Sprinkle in a few dependencies, that will speed us up. Oh, and let’s use ChatGPT and Copilot, we’ll be even faster. Oh, and performance, yeah, performance: microservices, Kafka, Redis, Kubernetes… we’re now ready to scale up. Oh, Sentry, Prometheus and Grafana, too, where would we be without ’em? For the front-end, we’ll write a website, and for mobile, Electron.

Oh, wait a second, we need to make money and to fidelize our users! Let me see… ads, tracking, and good reasons to revisit our app, perhaps a little NFT here, gamification… alright, we should be good.

Three… two… one… and we have shipped v1!

Also, the world is burning.

Perhaps it’s time we revisited how we do things?

The Infinite World Model

For the past ~30 years, the software industry has progressively optimized itself for an infinite world. An ever-increasing number of customers. Ever-increasing CPU and GPU power. Ever-increasing bandwidth. Ever-increasing Cloud power. Unlimited energy. For open-source projects, unlimited untapped contributors. For everyone else, unlimited VC money. Also, governments that either do not care, do not want to harm economic growth, or do not understand.

That’s the **Infinite World Model** and, truly, we owe it to a combination of 19th century Colonialism (unlimited raw materials and labour) and the Industrial Revolution (unlimited raw power and progress).

In this Infinite World, the main imperative is to be able to seize the opportunity:

- be first on the market;
- look good;
- be ready to scale when the clients come;
- do *whatever you can* to keep your clients.

For most companies, these factors trump everything else. They trump hardware costs. They trump maintenance costs. They trump bandwidth cost. They trump battery usage. They trump security. They trump accessibility. They trump privacy. They trump democracy. They may very well trump local or international law.

And they definitely trump both ethics and technical debt. While we’ve been complaining about the result, *we* have spent the last 30+ years teaching developers (and product managers, and CEOs) to ignore all these. After all, Worse is Better, right?

And once the VC money comes, or the users come, there will be time to fix things. But of course, you have probably experienced it: once money comes, there will be time to add new features. Fixing things can wait until it becomes an emergency, or forever.

If you pay attention to actualities, you may realize that this world might not be around much longer. For one thing, we’re in an Energy Crisis and a Climate Crisis that has a direct or indirect impact on both industry and consumers. A Logistics Crisis. A few Addiction Crises. A worrying Consolidation of Power among few unreliable Tech Giants. There’s a strong risk of crisis on multiple raw materials and components. There’s also that Trade War that isn’t in the news anymore but never actually stopped. When I was at Mozilla, we started seeing the end of these unlimited open-source contributors. CPU performance per core has largely stalled since 2010. Did I mention that we’re in a Financial Crisis and that VC money has rather dried up for startups that do not have “AI” in their pitch? Also, we have Democratic Crises but governments look like they are finally moving on tech, for better and for worse. Oh, and of course we might have stepped into World War III without quite realizing. That’s bound to disrupt things quite a bit, in fun and interesting ways.

We can bet that things will get better. And for all we know, it might. I, for one, am not confident.

So I’m going to assume that the model that worked for the 19th century aren’t necessarily fit for our time.

Towards a Finite World Model

How do we build for a **Finite World**?

Let’s perform a quick Risk analysis:

- Our Environment can crumble
- Expect that the price of energy will keep increasing, for both you, for your cloud provider and your users.
- Expect that hardware will stop growing more powerful both for you, for your cloud provider and your users.
- Expect that both your users or you may need to move because of a drought, or a flood, or a storm, or forest fires, or famine.
- Expect that your data centers may need to move or shutdown for the same reasons.
- Our Globalization can crumble
- Expect that hardware will get more expensive and possibly hard to find, both for you, for your cloud provider and for your users.
- Expect that your cloud or API provider may become your competitor or may align with local or foreign interests hostile to you, your community or that of your users.
- Expect that your goods, physical or digital, can have difficulties reaching your consumer due to legal or financial barriers.
- Expect that you will have less money to build your product or keep your company afloat.
- Our Societies can crumble
- Expect that your nation might not remain a democracy much

longer.

- Expect that your nation might not accept you, or some of your colleagues, or some of your users much longer.
- Expect the same in the various nations of your clients.
- Expect that your users or your colleagues can be addicted, either to a narcotic or to an application.
- Expect that scientific research or education may not be funded anymore.
- Expect that violence, possibly war, can hit your nation within its borders.
- Expect that AI and further automation will take people's jobs, forcing them to change careers more often, with longer periods of unemployment/retraining.
- Expect that extorsion criminals, private spies and nation states will be interested in any data you retain.
- Expect more pandemics, fewer antibiotics and, generally, worse health.

That's… actually much bleaker than what I intended to write when I started working on this blog entry. Our Finite World sounds like a paranoid dystopia, doesn't it? In fact, it looks like the entire Cyberpunk genre, minus flying cars. As usual when dealing with Risk, we'll hope that none of this will happen but plan ahead in case it does.

Since we're talking Risk, let's give our Threat a name. Let's call it the Zerg. We live in a Finite World. Energy, disk space, food, attention, democracy, public health, society or peace are all finite resources. And we have the Zerg. Anything that eats away at our Finite World without giving us something at least as valuable in return, getting us closer to exhausting, bombing or coughing our way back to the stone age, is a Zerg.

Our mission is now to fight the Zerg, as aggressively as we need.

We're the tech industry. We might not be able to solve climate, or energy, or food, but we have power. Power to do inadvertent harm, certainly. But power to do some good, too.

Designing for a Finite World

I’m not going to pretend that everybody can fight the Zerg. In particular, if you live in an authoritarian state and if your only choice is to comply with the boss’ order or starve or be deported, your hands might be tied too hard to do anything from this side. Note that you might still have a choice as a consumer, which is better than having no power, and I suggest you consider the best way to use that choice.

From this point, I’m going to assume that you have influence beyond being a consumer. Perhaps you are a developer, or a researcher, a decision-maker, or an advocate, or an entrepreneur. Let’s explore a few ways to use that influence and design against the Zerg.

Designing for Finite Needs

Alright, let’s start with a hard one.

You have an idea. It’s going to make a great start-up. It’s going to make you rich.

Congratulations. Please consider dropping it. No, seriously.

We’re talking about a Finite World. I know, I’ve been in start-ups and the ecosystem is exhilarating, is makes us feel smarter, in control of our life, possibly in permanent burnout, but we’re learning and achieving so much, plus we have a chance to strike rich! But the start-up ecosystem is designed as a permanent Zerg Rush. We’ll have to give it up, eventually. It’s time to start saying goodbye.

This goes double if your idea is based on another Zerg, such as *blockchains* or *generative AI* or *social networks* or *adware* or *viral marketing* or *AAA video games* or *anything that requires addiction* *to sustain a business*. Yes, some of these are insanely cool things. But by design, they require exponential amounts of resources to evolve, which makes them Zergs.

This does not mean that you should abandon research projects or art projects or hobbies. But it does mean that you should strongly consider *not scaling them up*. Sorry, I meant not Zerging them up.

Designing for Finite Funds

There are a few, rare, ideas that do need to scale. They are going to do so much good that they are going to outweigh the Zerg factor. If you have one, cherish it, feed it, grow it, fight for it, make it real.

But whatever you do, do *not* take VC money. If you are familiar with VC investors, you know that the system is designed to burn your hands with money into consuming as many resources as it takes to go big or go bust. In other words, to turn you into a Zerg. Perhaps some day, VC will evolve into something new and that fits our Finite World, but we're not there yet, and no amount of "green" in the name or description of your VC will be sufficient to change that.

Alternatives exist. Bootstrapping. Open-source. Working with universities or non-profits. Self-funding the effort as a side job. None of these alternatives is as well-oiled as the VC money pipeline. They take time and effort. They can fail. Still, they remain better than the alternative.

Designing for Finite Performance

Expensive energy and hardware means that we need to stop considering performance as “we’ll throw in more cloud resources” or “we’ll run on more recent user hardware” and return to considering performance as “we’ll need to use high-performance languages/libraries” and “we’ll spend more time benchmarking”. This is true both on the front-end, on the back-end and on the wire. And since you don’t have VC money to spend on cloud resources, you’ll need to do this quite early.

- Audit your programming language. If you are writing in Python, PHP, Ruby or server-side JavaScript, you are optimizing for quick prototypes, at the expense of performance, which means that you will need to throw in more cloud resources to scale. Consider faster languages (e.g. Rust is typically 10x-30x faster than Python, but also Go, Java/Scala or C#/F#).
- Audit your model. If you are writing in microservices, you are optimizing for throwing in more cloud resources, at the expense of CPU and network performance, as well as developer velocity. Consider monolithic services or distributed agents (e.g. Elixir can run millions of agents per node).
- Audit your protocols. If you are using HTTP and/or Kafka, you are using for communication within your system a protocol designed to show users with documentation. Consider faster protocols (e.g. Zenoh typicaly runs 30x faster than Kafka and is much more memory efficient).
- Audit your web front-end. If you are shipping tens of megabytes of JavaScript (even lazily), you are overconsuming both your bandwidth and the battery of your users.
- Audit your mobile/desktop front-end. If you are shipping Electron, you are shipping 50Mb-150Mb to your users (also, to your CI) for

features that are already present on their machines. Consider Tauri or Neutralino, which offer a similar experience, but only consume 600kb (Tauri) - 2Mb (Neutralino) of disk space, and way less RAM than Electron. Alternatively, if your code is CPU-intensive, consider shipping native applications.

- Audit your features. Some of your features may consume considerable energy, either server-side or client-side. If your front-end is polling the back-end permanently for updates, you may consider moving to websockets, or throttling the polls. If your video game requires a recent GPU, you are encouraging users to buy a new card, or a new phone, or a new laptop, and quite possibly throw away the old one. Consider designing your game for lower-end architectures, even if it means sacrificing looks or adopting a retro style.

  In other words, to fight the Zerg, reconsider common wisdom and benchmark aggressively every component of your architecture.

  I am *not* going to suggest you give up on using the Cloud, because it is not clear to me that alternatives are better with respect to the Zerg. If you have insights on the topic, please get in touch.

Designing for Finite Data

Expensive disk space, fragile democracies, hostile communities, stricter laws and higher chances of being hacked mean that we need to stop considering storage as both free and without consequence.

Again, let’s start with the hard one:

- Audit your data for legality. Your data might not be legal anymore.

- Audit your data for risks. Assume that the worst political or criminal figure in your country gets hold of it, and that they have gained dictatorial power, or perhaps that the worst political figure of the worst country imaginable gets hold of it, and they're going to use it to wage war.
- Not just the data you're storing, also the data you're sending to third parties, including your cloud provider.

  Let me stress this, because, in a Finite World, you might be endangering people's lives with your data. If you cannot run your business without endangering people's lives, please consider pivoting.

  Now, we can proceed with the usual Zergs:

- Audit your data for content savings. Consider what you can (or must, for legal compliance) safely erase.
- Audit your storage for structural saving. For instance, if you are using a document-oriented database because it makes prototyping easier, consider moving to either a relational or a column-oriented database, which are typically moch more efficient with disk space.

  In other words, as much as you can, benchmark your Zergs into harmlessness.

  Similarly, I am not going to suggest leaving your data off the Cloud, as I do not have a clear alternative to suggest. But do not hesitate to investigate.

Design for finite Brainpower

Everything is expensive. Your API provider might be your competitor and might price you out any day now. Working and hiring across borders has gotten more complicated. Science and tech education might be under-funded and lacking. Achieving performance takes time. You might be struggling to hire developers. You will need to work with less.

This might be counter-intuitive, but it means that you need to take time and invest it into using technologies that work for you in the long run.

- Invest in Open-Source. Most companies consume open-source but few are part of the ecosystem, contributing back and having each other's back. By being a good actor and contributing back, you help grow the reliability of your own technology, you gain the ability to expand it in directions that are important to you and, just as importantly, you gain opportunities to hire and you grow the skills that can help you diversify and pivot.
- Invest in Maintenance. Besides open-source, pick technologies that are optimized to let small teams perform refactorings and reconfigurations. In terms of programming languages, this is another reason to avoid PHP, Python or JavaScript, which are optimized for writing prototypes, and to consider TypeScript, Rust, Java/Scala or C#/F#.
- Invest in Knowledge. You will need to work with colleagues who do not have training. This means that rather than assuming that your new colleagues can just get started as soon as you have handed out work, you will need a form of mentorship. My experience of mentorship suggests that it grows the skillset of both the mentor and the mentoree, so it is a good investment.
- Invest in Teaching Organizations. Consider working alongside universities or other teaching organizations, including non-profits.

Much as working with open-source, this will grow your team's skills, help you recruit from possibly untapped pools and be good for publicity and the community.

Again, this might be counter-intuitive, but I would also suggest against using ChatGPT, Copilot or any other AI assistant. For one thing, each request to ChatGPT or Copilot has a steep energy cost. For another, experience suggests that junior developers (and other professions) use the results without understanding it, which in turn hurts both the Maintenance and Knowledge objectives above.

Fighting back the Zerg

So far, everything we've discussed was about slowing down the Zerg. And if we, as an industry, only succeed at massively slowing down the Zerg we've been mass-producing for 30 years, that will already be a victory to celebrate.

I want to believe that we can go further. I'm not going to pretend I know how. But maybe you do.

- Audit your surroundings. Consider if there is anything you can do to help heal your community from intolerance and violence, from addiction, from despair, from health issues, from authoritarian tendencies. Your contribution does not need to be technological.

What now?

I hope that this post can inspire some in the tech industry to take arms against the Zerg. I know that *I* am going to use this and push for better Zerg fighting at work.

Undoubtedly, I have missed many ideas, many possibilities. Undoubtedly, I have been naive about many things, too. But we need to start somewhere.

I'm planning to revisit some points of this post and dive into more details. After all, I'm supposed to be an expert in safety, performance and open-source, it's time to put that knowledge to good use, isn't it?