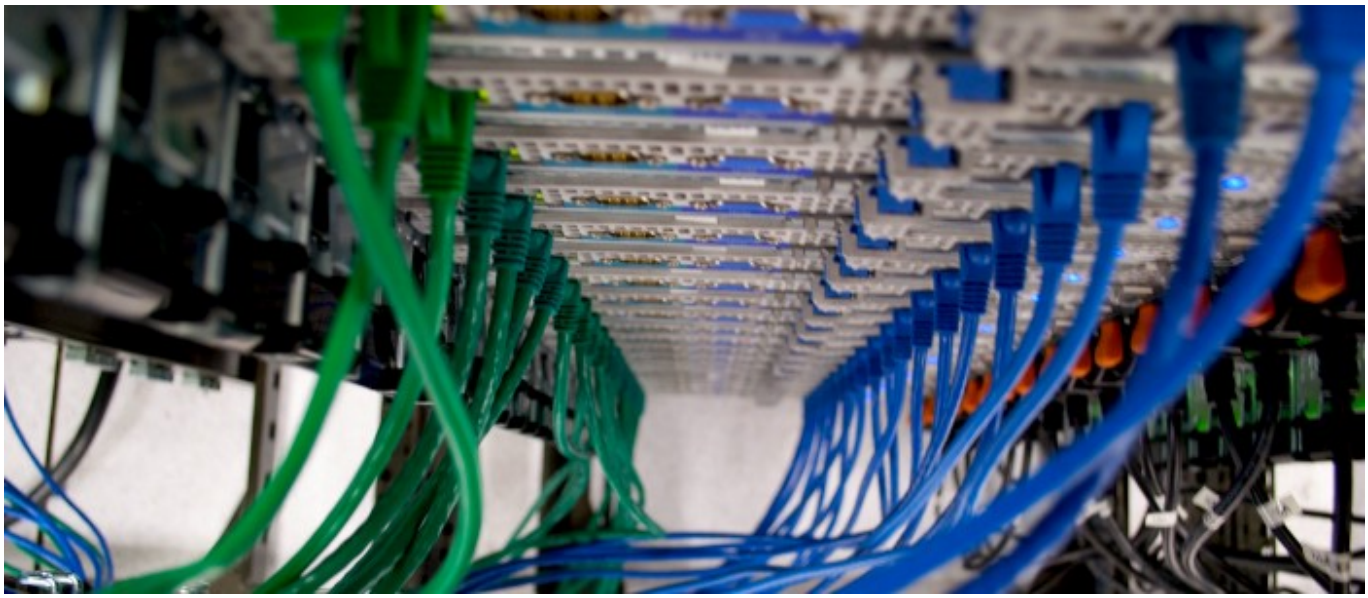
Understanding Abundance, part 3: The Next Big Thing

Alex Danco Follow
Feb 19, 2017 · 14 min read



Hello! If you're coming here for the first time, thanks for checking out my writing on Medium. I don't publish much here anymore — I've switched over to publishing entirely on my own website, alexdanco.com. I also write a weekly newsletter which comes out on Sundays, you can sign up at danco.substack.com. I write a lot, and I don't want you to miss it! So please head over there and subscribe.

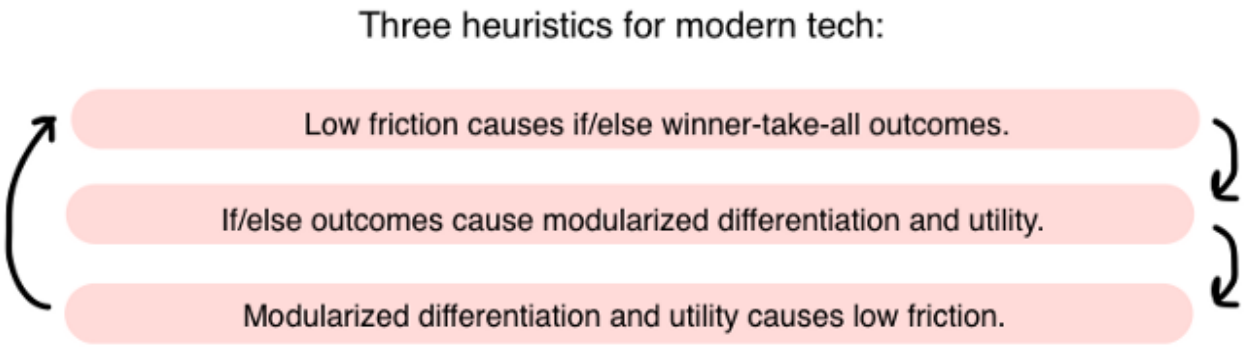
. . .

Welcome back. So far in our Abundance Series we've talked about frictionless consumption, if/else decision-making, and the consumerization process on the supply side in Part 1; we then talked about the Red Queen effect and how the tech industry has evolved to continually

meet demands for abundant consumption in Part 2. We left off with some heuristic rules of thumb for understanding the abundance cycle, and are now ready to look into the future at what's imminent.

So, what comes next?

What are we beginning to consume frictionlessly? Who is accommodating it? Where is the Red Queen just starting to jog?



Part of what makes the tech industry special is can be found in our three heuristics, which we wrote down at the end of part 2. They're how we got crazy outcomes like Moore's Law, the Internet, and iPhone / Android taking over the world so quickly. So what's the next big act? What is the foundation on which the next generation of companies will be built? Where is the unclaimed whitespace?

One way to approach the question of *where is the unclaimed whitespace* is by asking, "What are we starting to frictionlessly consume in compounding quantities?" Here's my answer. It's deceptive, because we don't usually think about it in these terms.

We've begun to consume **functions**. And the wheels of our compounding machine have been set in motion; perhaps irreversibly. The Red Queen is in motion.

Let me explain what I mean by "consuming functions". We can think of functions as orthogonal to the way we traditionally perceive the outside world: in terms of *objects*. Software people (which I am not, for the record) are well-versed in the differences between object-oriented versus functional programming languages, but you can extend

this thinking to the broader world as well: is your world populated by objects that perform functions? Or is it populated with functions that make use of objects?

We can illustrate this concept with something we're all familiar with: driving. The way we've thought about personal automobile transportation for the last 100 years has been quite object-oriented:

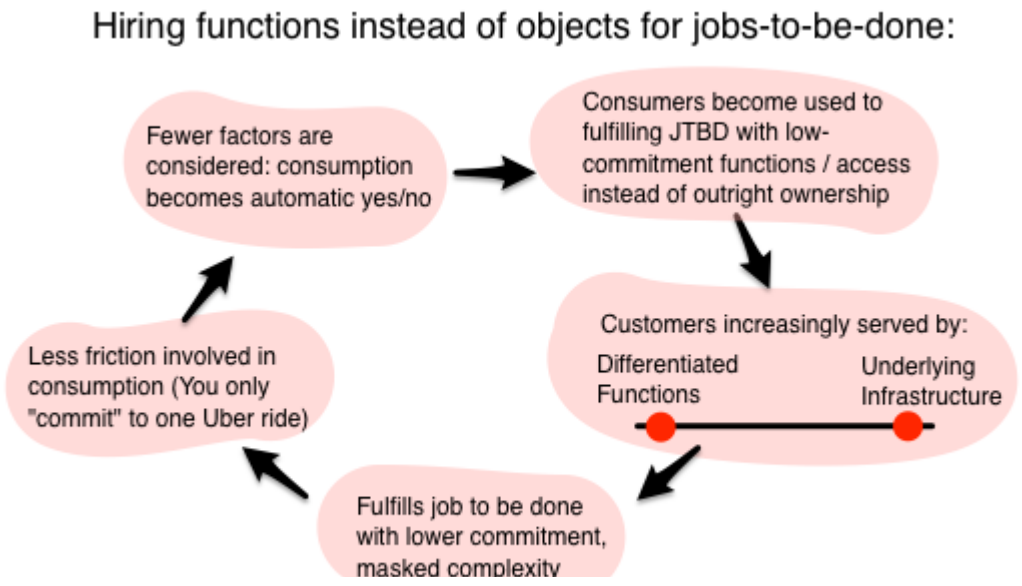
-There are a bunch of available functions related to personal vehicle objects: [drive, park, look cool, …]

-If you want to be able to get around, you purchase your own instance of [car]. As more people want transportation, you add more cars. The world becomes populated with [car] objects that all perform these functions.

Now consider Uber and transportation-as-a-service. It's the other way around; a *functional* approach to transit:

-There are a bunch of vehicles out there: [Car A, Truck B, Train C, …]

-If you want to be able to get around, you *hire a function* [get me to work, pick me up from the airport, deliver this package]. As more people want transportation, the world gets lit up with more and more functions that all make use of whatever resources are available. This isn't specific to Uber, although we'll probably look back at Uber as a major wake-up call. Really, it describes what's about to happen to large parts of our economy. Instead of owning objects, we're increasingly accessing functions.



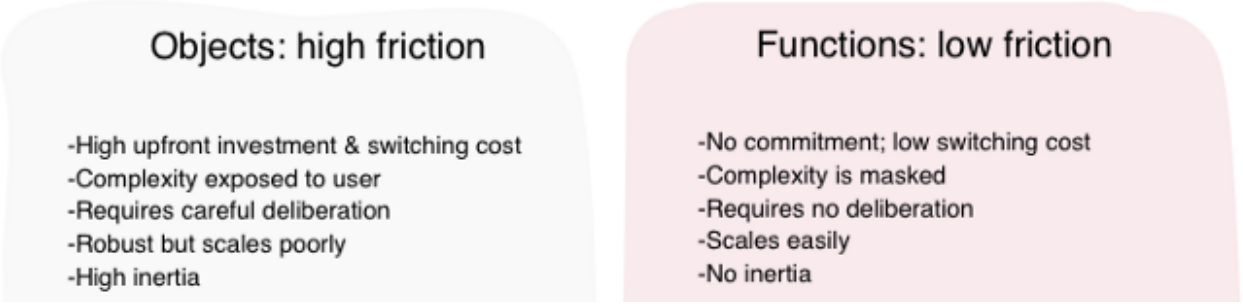
As friction goes away, customers hire *functions* to address jobs-to-be-done instead of hiring *objects*.

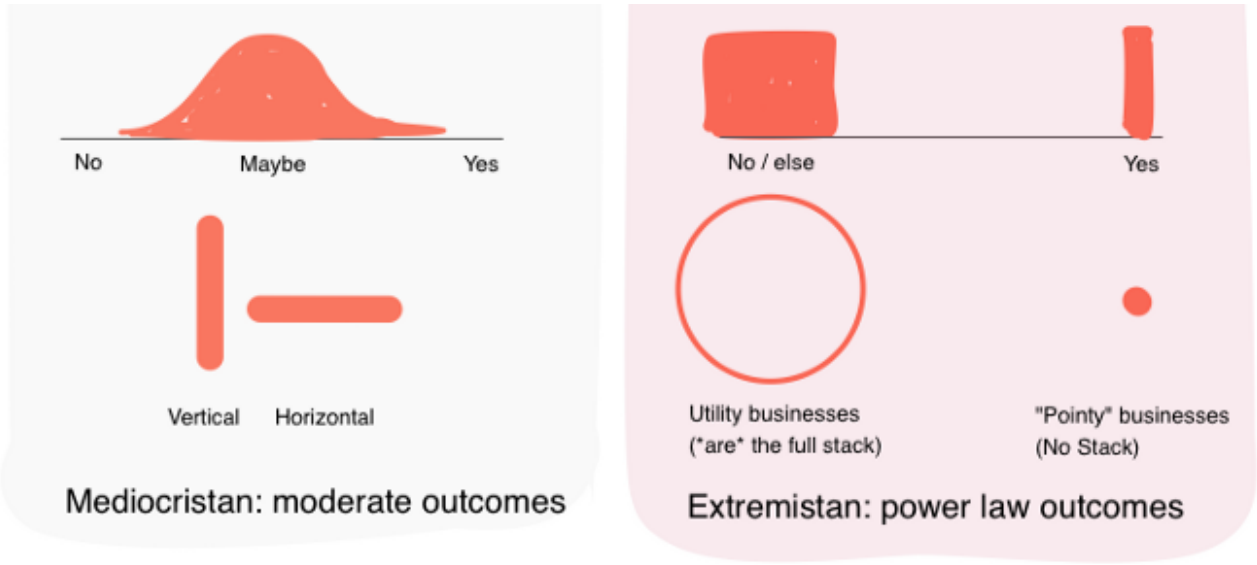
The object-oriented world we know is actually not all that consumer-friendly. Objects have high switching cost. They're usually expensive. Their complexity is often exposed to the user. They require careful consideration before a purchase. Functions are the opposite. There's very little switching cost. They're usually cheap. Their complexity is masked. They can easily be event-driven. They can be consumed frictionlessly, and they scale. And, crucially, it's much easier to move from consuming objects to consuming functions than it is the other way around, because functions are cheap and disposable. Owning your own car does not stop you from trying out Uber, even if it's only for the "Get me to the airport" function at first. Owning a CD collection does not stop you from getting Spotify and trying out Discover Weekly. Having a graduate degree does not stop you from learning from YouTube videos. And so on.

Objects have inertia; it's hard to abandon them for a different object. *But functions do not*. It's very easy to abandon a function-as-a-solution *in favour of a better function.* It's hard to trade in your car for a different car, but it's easy to start taking Uber — and leave your car at home. And it's easy to switch to Lyft, even if you're an Uber user. This is a process that moves in one direction: towards less friction. Or, as we called it in part 1, *consumerization*.

We'll probably look back at Uber and Lyft as among the first mainstream, large-scale killer apps for consuming functions instead of objects. They are the Lotus 1–2–3 and Visicalc of our loose analogy — an "I get it" moment for consumers.

But who is Windows? What is the function operating system?





When we take off our object-oriented glasses and put on our functional glasses, it becomes quite clear what the "third act" of tech is going to be, and where we can find the unclaimed whitespace.

The first act was Intel and Microsoft: differentiated software running on utility hardware. The second act was ushered in by Netscape and the web: differentiated hypertext on utility Internet infrastructure. Part two of the second act was the iPhone and mobile: hardware and software catching up to what the Internet was natively capable of.

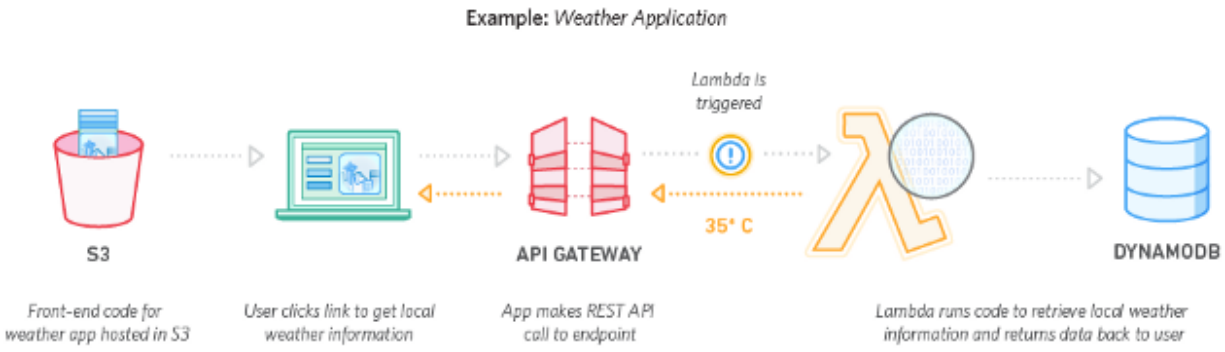
Today, we sit at the beginning of the third act: *Differentiated functions running on utility infrastructure.* We have all kinds of names for aspects of this idea, and at the core is a new type of computing architecture we have many names for. Serverless, function as a service, platform as a service; the names and variants aren't what's important here. What matters more is that one institution has an early lead on everybody else: Amazon Web Services. And AWS Lambda, launched into high velocity by EC2, S3 and the other AWS services, is the master stroke.

AWS EC2 was MS-DOS; Lambda is Windows.

(Brief pause: while I'm going to talk about AWS Lambda for the rest of this post, it's far from the only offering on the serverless — platform as a service spectrum that's making waves. Azure Functions, Cloud Foundry, Google App Engine & Firebase, and more are nothing to sneeze at, and it's way, *way* too early to tell what the future will hold in that

twenty-year competition. So for the rest of this post, if you prefer, you can substitute AWS Lambda for "your favourite event-driven system" with no love lost.)

Introduced just over two years ago, Lambda is a compute service offered by AWS that lets developers run code in response to events. The underlying compute resources are automatically provisioned and managed in the background, out of sight and out of mind. John McKin's concise definition of serverless encapsulates what's going on well: "An event-driven system that uses function-as-a-service and other fully-managed services for logic and persistence." You upload snippets of code, and then set them up to trigger based on events — whether they come from inside AWS, a web site, or an app. *Everything required to execute those functions is abstracted away*.

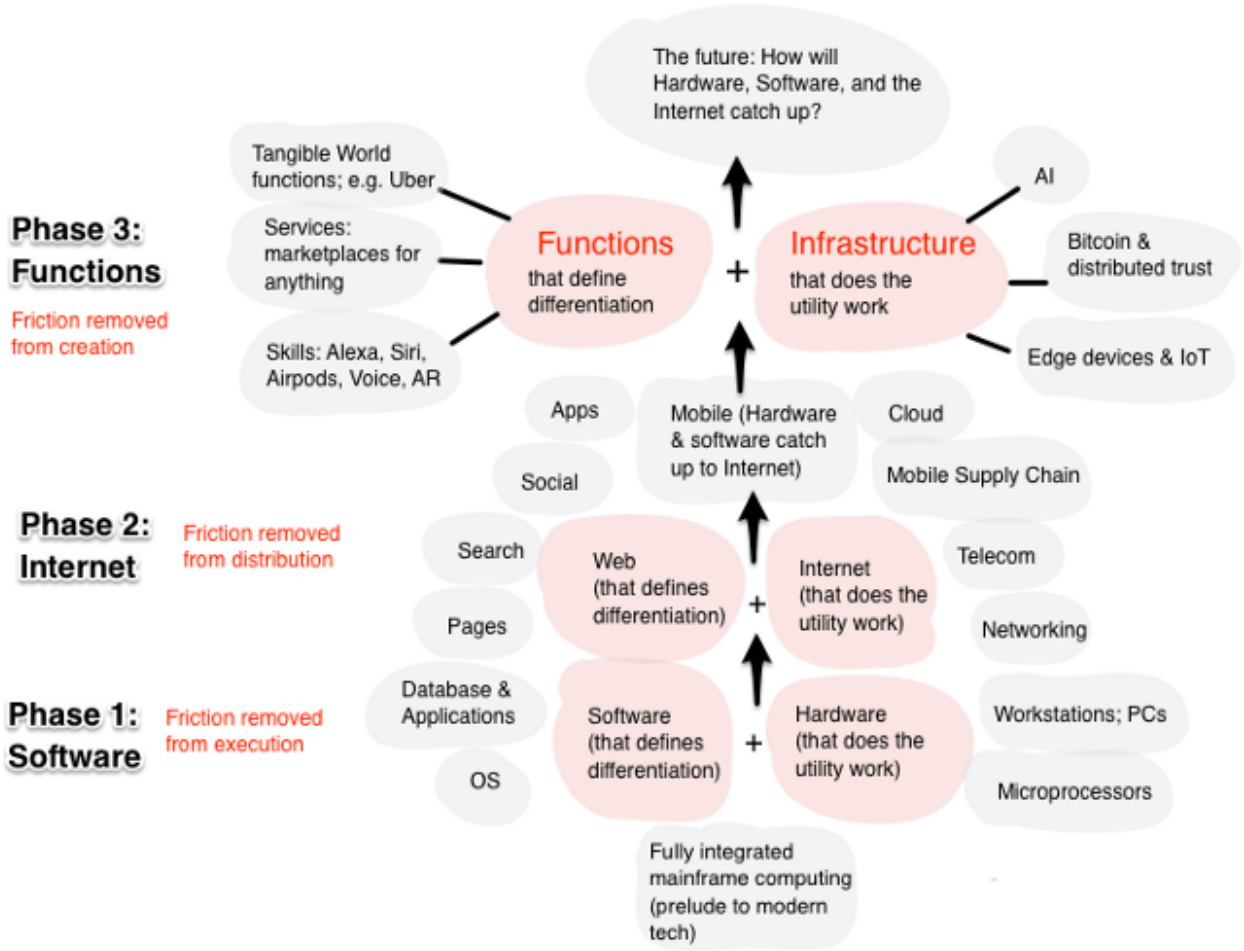


You can roughly divide people paying any attention to AWS Lambda into three categories. The first category are people who don't understand what all the fuss is about. "It's expensive. It's weird. It might be interesting to play around with, but it won't be used much in production." If we've learned anything from the last fifty years of tech history, we know what happens to that story. The second category of people are those who do get it. They understand that this is more than a marginal improvement on the public cloud, and that function-as-a-service is particularly well-suited for making things like Alexa skills.

But the third category of people are the ones who truly get it. To them, function plus infrastructure architecture isn't merely important. It's *The. Most. Important. Thing*. It's the beginning of something as disruptive as our first two paradigm shifts of tech: as meaningful as Software plus Hardware, or Web plus Internet.

F+I computing, in other words, is becoming the core idea at the middle of our next wave of abundance, as we have defined it: *the state when friction of consumption [of functions] approaches zero.* All of the elements going into that consumption — AI, Cryptocurrency, IoT on the infrastructure side; Services, skills, voice, AR, bots, on-demand access on the functions side, will be organized in this paradigm. The operating system sits in the middle.

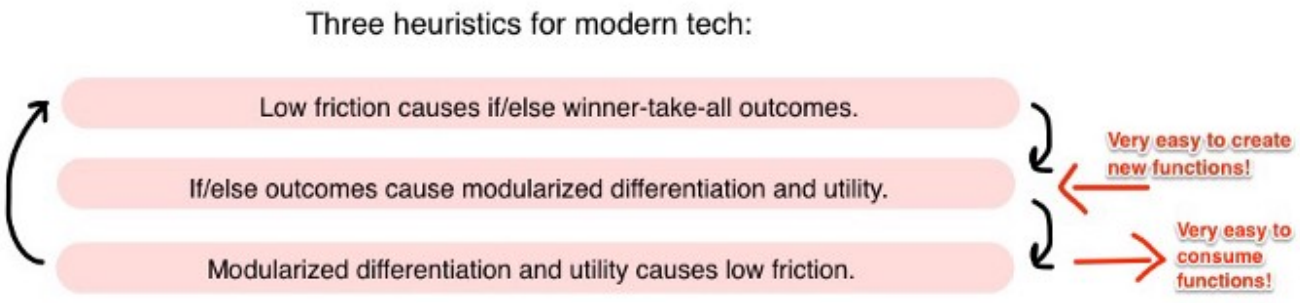
If we were to try and illustrate the 'hierarchy' of what we can see so far, here's what I come up with at first attempt:



However, what's really exciting and slightly terrifying is not what's on that tree diagram. It's what *isn't* on the diagram. It's becoming very easy to create new functions: after all, they can be made out of other functions. And unlike objects, they have low inertia: well-built functions that people want can spread *very* fast.

The impressive thing with function + infrastructure methodology, which is particularly evident when playing around with Alexa and all her new skills, is just how powerfully flexible and modular it is. The back-end utility infrastructure that retrieves Christmas music on command is the same as what will soon be powering your Volvo's driver console, or your bank's customer service. And with self-service APIs, tools and documentation like Alexa Skills Kit available to outside developers, anyone can build new consumer-facing functions out of all of those underlying building block functions. Soon, it won't be limited to in-house functions: if we know anything about Amazon, it's that we're headed towards a function marketplace environment not too dissimilar to their retail marketplace, where developers will stitch together functions that are either made by Amazon in house (like Basics!) or made by a third party developer. So long as it works and comes from a verified source, you're good to go. As Simon Wardley puts it quite concisely: *The future of software development will be lots of lambda functions consumed from a marketplace, stitched together with some new capability.*

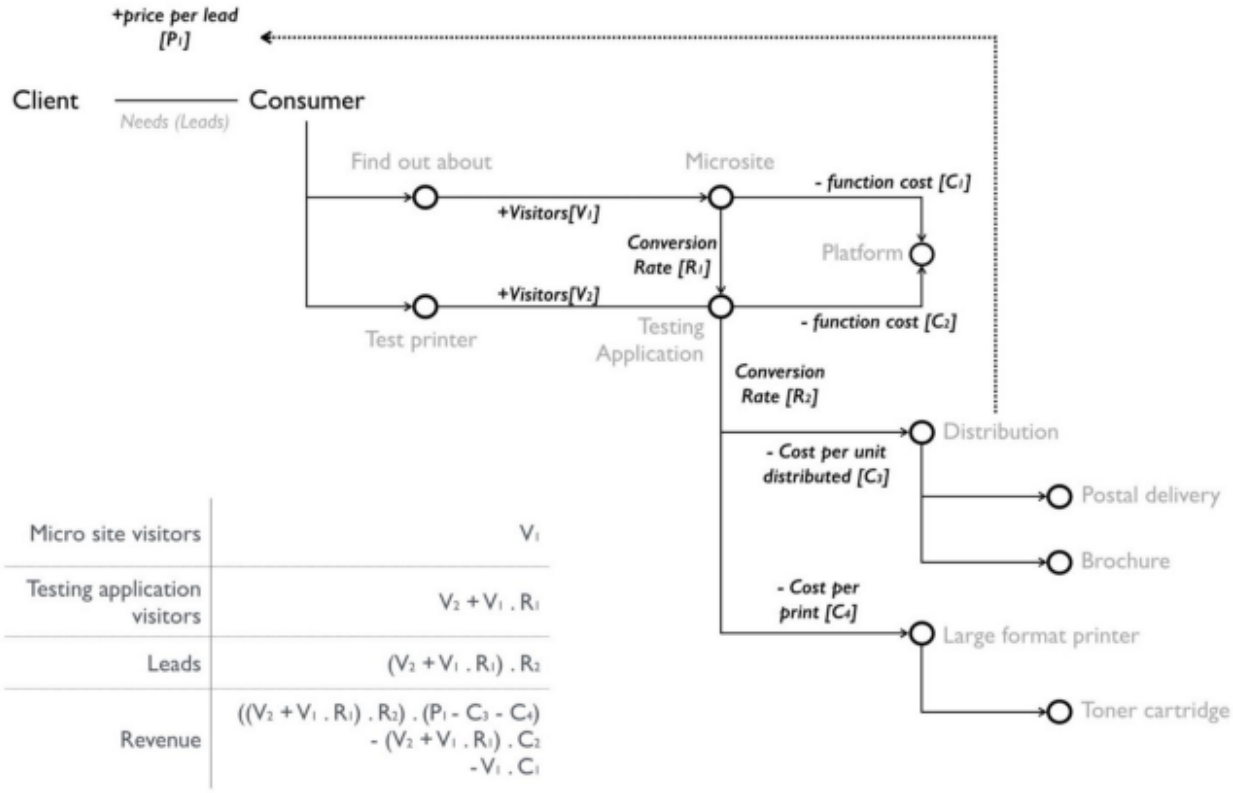
Let's say you're a developer in the near future who has identified some unsolved job to be done. You'll set out to build a function that addresses it, and can take for granted that all of the underlying utility infrastructure will just work. Furthermore, you can make use of the fact that countless other functions have already been written, tested and deployed — and are available for you to use. You know the marginal cost of running them ahead of time, so provided that you trust in a set of functions' ability to do what they promise, there's really no need to hesitate on whether to use them or not. In short, there is nothing stopping anyone from continually creating highly differentiated functions, releasing them into the wild, and setting this cycle into overdrive:



Here's where we get into the disruptive part of the future that we really can't predict. When creating functions becomes frictionless and easy, the way we start to think about

business models changes along with it. In the brilliant post Why the fuss about serverless, again by Simon Wardley, we get a preview of what's imminent. In this passage, he describes what is essentially a set of functions that constitute a large format printing shop business:

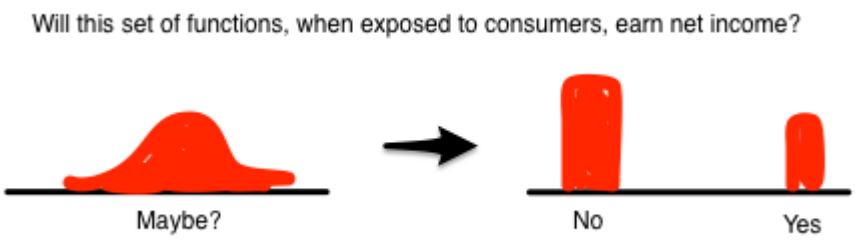
"In such a utility platform world, your application is simply a function running on the platform and I'm charged for use. The operational cost of my microsite is basically the number of visitors x the average cost for the microsite function. … This is like manna from heaven for someone trying to build a business. Certainly I have the investment in developing the code but with application being a variable operational cost then I can make a money printing machine which grows with users. It also changes my focus on investment — do I want to invest in increasing marketing for more users, or the conversion rate, or maybe the testing application is so badly written (or a function within it) that investing in coding improvement will bring me better returns? Suddenly, the whole way I build a business and invest is changed."



What Wardley is describing is what would happen if you could build a business solely out of functions, with a variable cost that you know, and whereby in effect the entire

micro-P&L of each business unit can be known *ahead of time.* This is a big deal for several reasons.

First, "Worth-based development". Creators now have complete transparency into the variable costs of the functions that they use, their performance in the real world, and the ROI you'd get from instituting Change A vs. Change B. The friction of *uncertainty* around creating and consuming functions goes away; your decision not only to consume them but now build them is no longer *maybe. It's no, unless yes.* Sound familiar?



Second, the virtuous cycle of "prefabricated" businesses. Odds are that for 95% of use cases, the functions you'd need to build something are the same ones as most other people would need. Even fancy ones! Function marketplaces for third party developed functions become massive — not just for bits of code, but for actual prefabricated primitive lego blocks for business. A function might be expressed in a line of code, or it could be instructions telling a person in a car to deliver you a package, or it could be anything. But as your differentiated functions become trivially easy to construct, you can see how the flood of new functions entering marketplaces, *seeking revenue*, can send this virtuous cycle of frictionless creation into higher and higher gear. It's quite easy to see how the Red Queen effect will take over here. The faster you can create new functions, the faster the landscape moves with you.

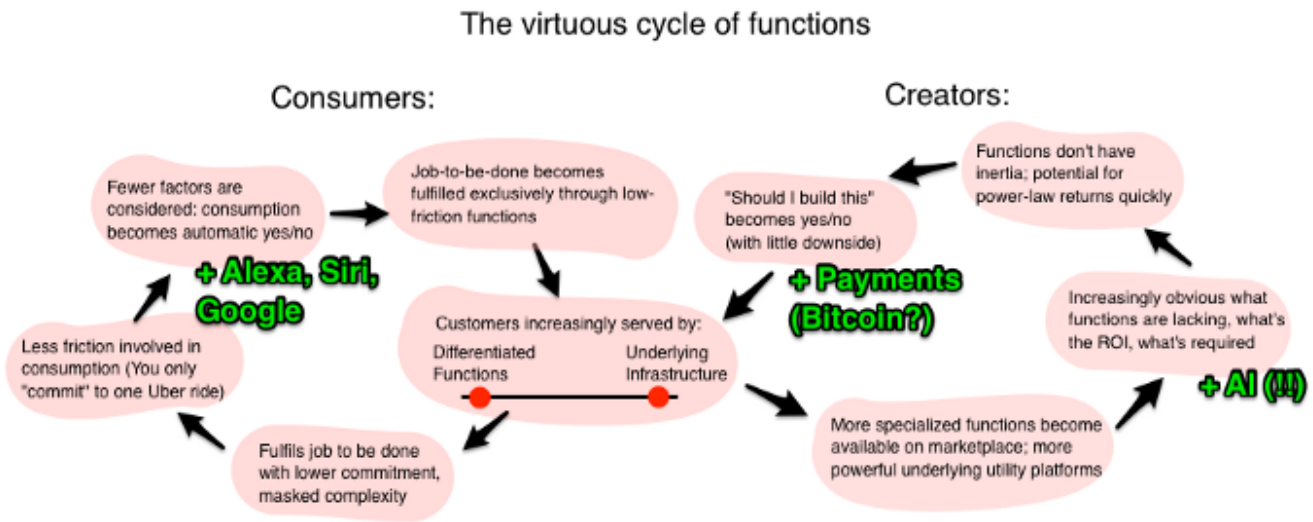
Third: payments, Bitcoin and distributed trust. Some of you may remember Marc Andreessen mentioning that in addition to the "404: page not found" error we're all familiar with, there was also a "402: payment required" error which never caught on. Well, imagine how easy it is to implement payment for services and functions in our new Function plus Infrastructure environment.

serverlessFunction doSomething(param {

```
something(param)
    if(paymentReceived) {
        return value;
    };
};
```

Courtesy of Jonathan Libov via the Twitter machine

This can extend way beyond the low-hanging cases like in-app payments. Last year I wrote a piece on our distributed energy future called In a world of Energy Mainframes, our "PC, meet the Internet" moment is very close. One of the core ideas of the post is that in the future, instead of a meter that ticks monotonously as electrons go by, energy will operate much more like a distributed commodity market. Simple derivative contracts may get established based on some basic underliers: energy created by a solar panel; energy consumed by a vehicle; energy stored in a battery. These are all examples of functions, and if we think through the lens of our Function plus Infrastructure framework, it becomes more apparent what will get built. With information and trust becoming the principal supply-side scarce resources for anybody trying to build or operate in such a distributed energy market, as I suggested in my older post, F+I makes this implementation much easier. Bitcoin could ultimately get adopted as a credentialing solution, or it could be something else. But again, what's most important here is F+I. It turns all other hard problems into easier problems.



Fourth, and most importantly, AI.

So far in this series we've stayed quiet on Artificial Intelligence. In fact, I'd probably hazard a guess that in response to the first line of this post, "What comes next?", many of you instinctively thought of AI, and for good reason. It will without a doubt play a critical role in how the tech tree grows into the future. But I don't quite think it's the *central* thing.

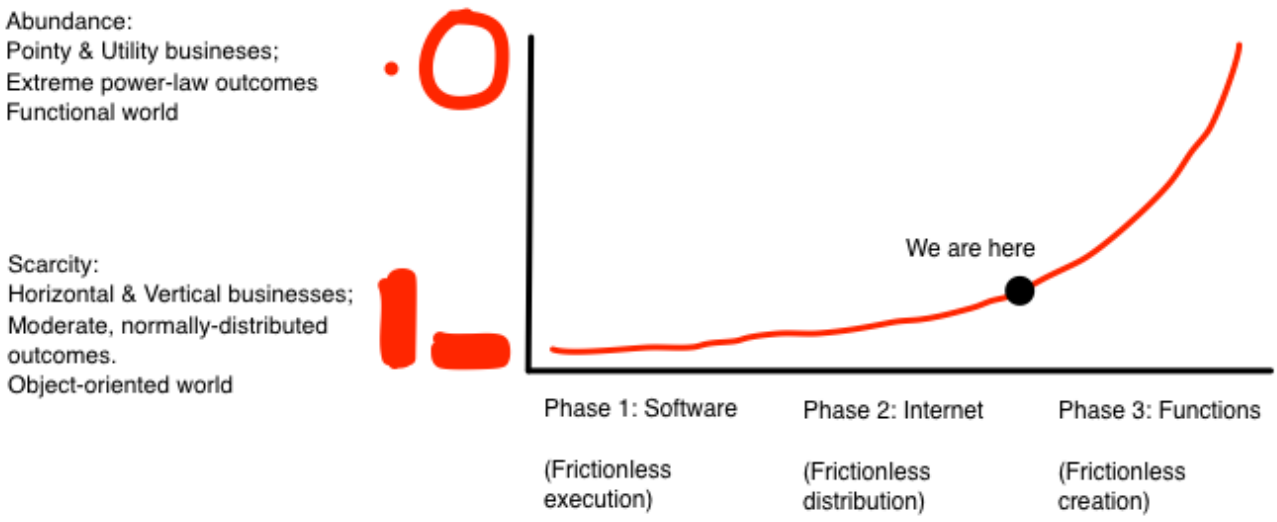
My current point of view is that many of us are making the same mistakes in thinking about AI as we did with "The Information Superhighway" back in the 1990s. If you go back and read *The Road Ahead* by Bill Gates, you'll get a sense for how pervasive these mistakes can be. Information Superhighway futurists forecasted a lot of things correctly: that we'd use it to watch movies, book plane tickets, join common interest groups, and chat with strangers. But they got one important thing wrong: they didn't quite grok the critical separation of differentiation and utility — *differentiated hypertext running on utility internet* — that was the web. And we know what's happened since. Most of the predictions of how we'd use the Information Superhighway did in fact come true! They just got the paradigm wrong, and that counts for a lot. It turns out that identifying what's important is not the same as finding the undiscovered whitespace.

In my view, the most powerful, wild and crazy applications of AI are going to have to do with F+I and can be best understood within that framework:

- Dynamic provisioning and managing functions behind the scenes. Leave it to a non-human mind to do this; it'll be far better than us!
- Identifying combinations of functions that humans don't necessarily perceive (or don't have time to wade through all possible combinations).
- Creating brand new functions that are missing and that have a high likelihood of yielding positive ROI.
- Releasing these functions into marketplaces, and continually building and assembling new functions and combinations of functions in ways that can generate profit.

- Establishing autonomous, intelligent organizations — initially within, but soon outside human institutions — with mandates to create, deploy and monetize these functions. In short order, their understanding of the complex function landscape and how to navigate it will rapidly outstrip even the best humans.

- Becoming consumers of functions themselves, with their own needs and wants. This is the true tipping point between 'before' and 'after' AI, in my mind. When non-human minds become consumers, and as those consumers begin consuming frictionlessly, the world will begin to look different very quickly.



In brief:

Software eliminated friction of execution. The Internet eliminated friction of distribution.

Function + Infrastructure, *especially* once AI enters the picture, will eliminate friction of creation. Beyond that? ¯_(ツ)_/¯

To wrap up, I'm not arguing that AWS Lambda specifically will surely be the central trunk of the future tech path. It's in the pole position, but as we all know, things can change fast. But what *will* be, I am quite confident, is our embrace of functional approaches to everything, and function + infrastructure architecture underneath powering it.

If we had to guess at a time frame for when this all happens, I'd guess around ten years. Beyond this point, it's pretty impossible to speculate as to what might change. (A lot!) But it's hard to say much beyond that. The real question to ask, though, is what's going to stay the same. And that's what we're going to talk about in our final chapter, part 4.

If you've enjoyed this series so far, be sure to sign up for Snippets, our weekly newsletter. See you next week for the final chapter.

Thanks to Ashley Mayer.

Serverless AWS Venture Capital Tech Social Capital Posts