

## Why we Code

[pramsey@carto.com](mailto:pramsey@carto.com)

[2007.foss4g.org](http://2007.foss4g.org)

The first time I had the pleasure of addressing a FOSS4G plenary session was as the chair of the 2007 conference, in my home town of Victoria. So, before I start, congratulations Michael, and the rest of the local committee on bringing together this 2017 edition. I remember the excitement and gratification of seeing the world show up in my town, and start talking about open source and software, it's pretty unreal.

2007 : victoria

2009 : sydney

2011 : denver

2013 : nottingham

2015 : seoul

Since then, I've gotten a chance to keynote in Sydney, Denver, Nottingham, and Seoul. So, this is the 6th time I've been allowed to stand in front of a FOSS4G plenary and speak. And, by accident, if not design, I seem to be on a biennial rotation. I also seem to be stuck in a rut.

---

2007 : welcome

2009 : economics

2011 : economics

2013 : economics

2015 : economics

After my turn as conference chair, I talked in 2009, 11, 13 and 15 about,  
<x> open source economics,  
<x> open source economics,  
<x> open source economics  
and also  
<x> open source economics.

2017: economics

Today will be no exception.  
I have my reasons!

---

open source is a  
system

Open source is a social system,  
and when you're embedded in a system  
it's good to know how that system works.

systems vs. stories

And I think it's worthwhile to talk about the systems because so many of the tales we tell ourselves about open source are rooted in cultural myths about **individual** achievements and choices, and ignore the **systems** that individuals are working within.

The image shows the classic Star Wars logo, which consists of the words "STAR" and "WARS" stacked vertically in a stylized, outlined font. The logo is yellow and set against a solid black rectangular background.

---

Let me put it in terms you all can understand:





Is the problem with galactic governance  
the **individual** performance of Darth Vader  
as a particularly cruel and callous manager,



or is the problem the  
strictly hierarchical nature of Imperial **system**?  
The Darth Vader theory certainly makes for better storytelling.



But an appreciation for the **systemic** roots of harsh Imperial governance would probably lead to a more just Galactic republic, at least, in the long run.

anyway...

But anyways...

“why we code”

I chose "Why we Code"  
as the title for this talk by way of reference to "Why we Fight",



which was a series of World War 2 films  
produced by the US Government  
and directed by Frank Capra.



Obviously, created within the context of a mass war mobilization, the films were propaganda. But "Why we Fight" was an odd piece of propaganda. First, it was 7 hour-long films. Image that, in our era of 15 second attention spans.



Second, the films spent a lot of time on background: the history, the geography, the context, and how those informed the goal. Which, to be fair, was total victory. But a great deal of effort was spent to build a rational argument,



when a far simpler approach would have been an emotional 15 second ad.

So "Why we Fight" was propaganda, but nuanced propaganda.

Which is why I chose "Why we Code" for today.

Because there's a 15 second sound-bite answer to that question...



And then there's what I'm going to subject you to this morning.

I have some nuanced propaganda for you today.

Nuanced because too many of our discussions of open source and other alternatives are simplified down to black and whites.

Because too often we ignore the economic and cultural context open source is embedded in.

“why we code”

"Why we code"?

There's a lot of easy answers,  
which are variations of pure propaganda.  
They are mythic answers, emotional ones, they are easy to visualize.

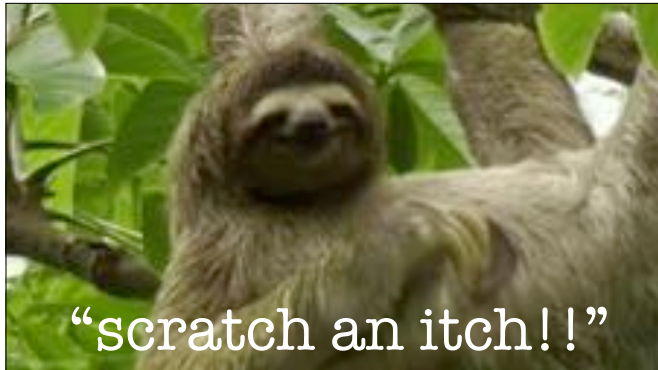


"Freedom!" is a favourite.

Software free to be read,  
free to modify, and  
free to redistribute with modifications.  
We code free software for idealism.



"For the glory of it!" is another.  
We code and release as open source for ego-gratification,  
to generate admiration in our peers.



"Scratch an itch" is also a popular reason.  
We code for localized practical reasons,  
we code to problem solve.  
But..,

“why **we** code”

All these easy answers depend on an unspoken assumption. They make a big guess about who "we" is. Who is this "we", doing all this coding, when we talk about "why we code"?



The Lone Ranger and Tonto  
are riding alone across the plains  
when over the hills in front of them





there comes a war party of blood-thirsty Sioux warriors.  
"This looks like trouble, Tonto, we'd better run" says the Lone Ranger,  
and they double back fleeing the Sioux.  
Well, they haven't been riding 5 minutes



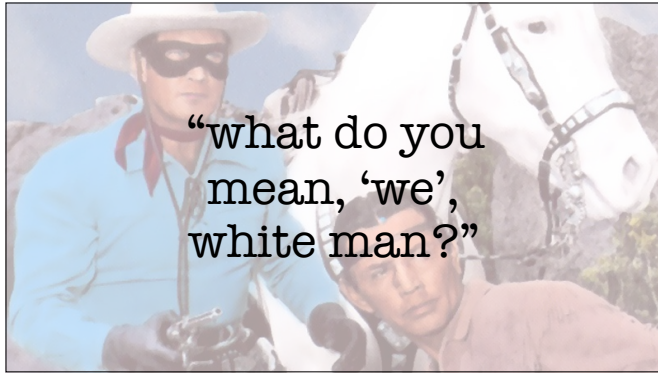
when they crest a hill  
and see coming up towards them  
a whole army of Cree soldiers,  
carrying death-masks and armed with muskets.  
"Ride hard, Tonto, this is our last chance." the Lone Ranger cries,  
and they turn left and ride hard down the slope.  
They turn and gallop down through a small copse of trees,



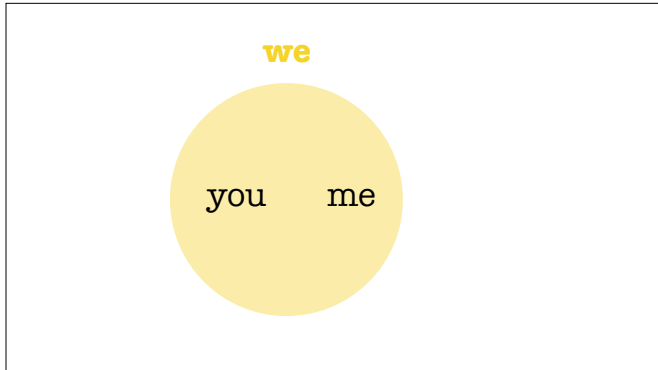
and coming out  
they run headlong into a party of Lakota,  
who whoop and bear down on them,  
their intentions clear.



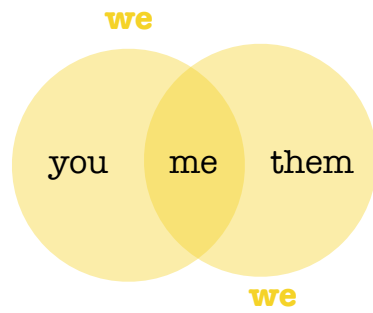
As the circle of Native warriors closes in on them,  
the Lone Ranger turns to Tonto and says,  
"Well Tonto, it looks like we're done for."



To which Tonto replies:  
"What do you mean, 'we', white man?"



Tonto's right,  
"we" is a very contextual word,  
and we inhabit a lot of different "we"s simultaneously



even within narrow fields like  
working with computers and  
writing software.  
Who are "we", anyways, when "we" code?

lone hacker

The mythic, easy answer,  
the simple propaganda answer,  
is the lone hacker.  
We answer "why we code" by  
ginning up a crude cultural caricature.



And so the canonical open source creator is Linus Torvalds, writing an operating system in his bedroom as an undergraduate student in Finland.



Or it's Guido van Rossum building Python as a Christmas project to "keep himself occupied", because what else would you do over Christmas?

richard  
stallman



Or it's Richard Stallman  
building GNU Emacs here in Cambridge, Massachusetts,  
creating the first brick in the GNU free software edifice.



But, the easy, mythic answer is wrong.  
Or, very incomplete.  
At best it starts right,  
then tends over time towards gross wrongness.

Linus started the Linux kernel, alone,  
in his bedroom, 26 years ago, this is true.

who is linux?

intel : 13%  
red hat : 8%  
linaro : 4%  
samsung : 4%  
suse : 3%  
ibm : 3%

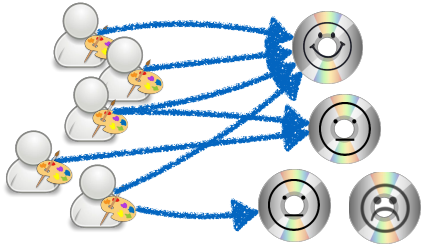
But today Linux is maintained mostly by corporations.  
Intel, Red Hat, Linaro,  
Samsung, SUSE, and IBM account for  
a third of kernel development.  
So rather than talk about grand heroes,  
I'm going to talk about generic agents,  
who each play a role in the  
interlocking  
economies  
of open source.



The agents are:  
The software itself.  
Individuals, who use the software,  
and who create the software.  
And finally institutions:  
corporations, governments, NGOs, universities,  
and so on,  
who use the software and  
who create the software.

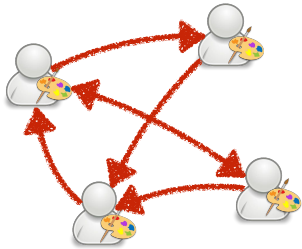
All these agents interact, but within different economies.

### attention economy



Different pieces of software compete within an attention economy.

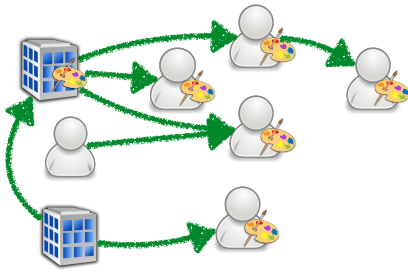
### gift economy



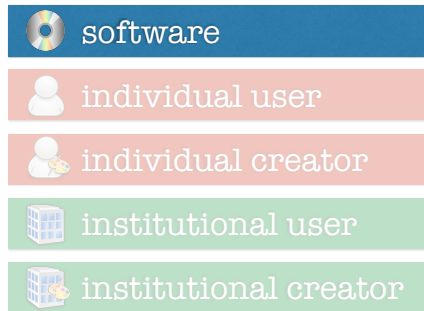
Different participants in an open source software community exchange value within a gift economy.



## cash economy



And of course individuals and institutions interact within our all-consuming cash economy.



Let's start with the software itself, and the "attention economy" it lives in.

cash economy	attention economy
money  millionaire	attention  celebrity

The idea behind an "attention economy" is that in a post-scarcity world, in a world where people's basic needs are already met, people will not compete for physical resources or wealth. What they will compete for, is attention. We already have a basic human attention economy in place, and we have a name for people who are "attention rich", they are called "celebrities".



As in the cash economy, the rich often get richer, <x> as attention begets more attention. And attention is surprisingly fungible, <x> attention earned in one field can often be converted into attention in another. However, a true attention economy requires a post-scarcity world, where or day-to-day physical needs met by default, and we don't yet live in a post-scarcity world.

cash economy	attention economy
money millionaire	attention celebrity

So we humans have a hybrid cash-and-attention economy. This can lead to some odd scenarios where our cash economy and attention economy confer very different amounts of wealth to the same person.



For example, the case of the YouTube celebrity, who amassed millions of followers, but who had to shoo away her fans so she could do the waitressing job that actually paid her rent. She was attention rich, but cash poor.

Now, keep an attention economy in mind, but replace humans with software.



food  
sleep  
clothing  
attention



n/a  
n/a  
n/a  
attention

Open source software lives in a software attention economy, in which software trades utility with humans in exchange for attention. Open source software has very few needs.  
<x> It doesn't need food or sleep or clothing. It can reproduce perfectly at zero cost via copying. It's almost immortal.  
The only thing software needs to stay alive is at least one person that cares about it, <x> a little bit of attention.  
Because operating systems change, new formats are developed, small bugs are found. Unmaintained, software will die:



emacs is dead



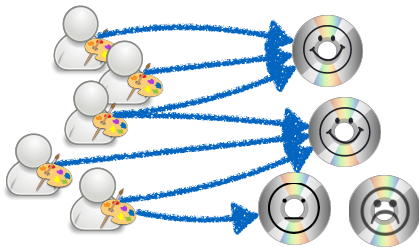
long live GNU emacs

the original Emacs text editor ran on the PDP-10 minicomputer, that code is dead; Richard Stallman's GNU Emacs, which has been maintained continuously since 1984,



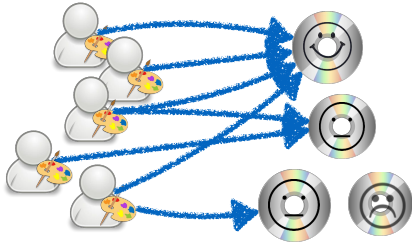
will run on your Android phone, if you wish.

### attention economy



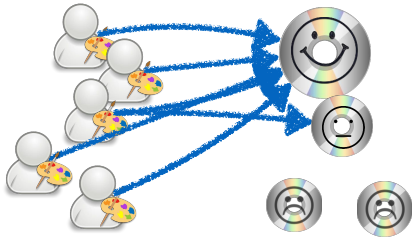
In this software attention economy, open source software competes for attention. Software that accrues more attention develops faster, it gets better documentation, it gets a snazzier web site,

### attention economy



and these things in turn allow the software to gain yet more attention. Just like the dollar economy, in the software attention economy,

### attention economy



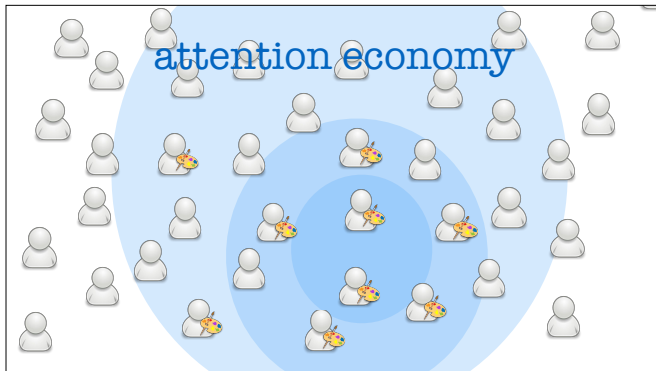
the rich get richer, and everybody else gets squeezed. Money begets money, attention begets attention.

## attention economy



“check out  
my **documentation**,  
my snazzy **web site**,  
my easy **quick start**”

So, there's a reason smart open source projects spend a lot of time on their documentation, on their quick-start guides and their one-click downloads: to grow the overall pool of people around the software.



The overall pool of users will include  
<x> some percentage of power users and documenters and evangelists,  
and that sub-pool will include  
<x> some percentage of bug reporters and fixers,  
and that sub-pool will include  
<x> some percentage of core developers and maintainers.  
Job one for any project in the attention economy is **attracting more attention**.  
So, who can provide attention to hungry software?



institutions don't write  
open source software



people write open source  
software!

Individuals!

Institutions can too, but

only via the mechanism of **individuals**.

Intel doesn't write Linux patches,

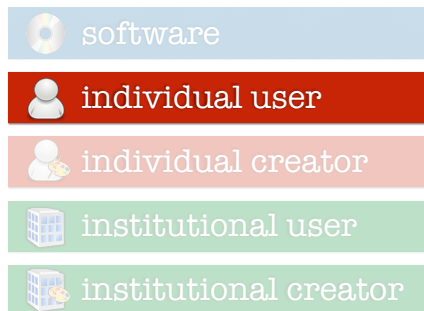
people on the Intel payroll write Linux patches.



open source is people!

Just like solent green,  
open source is people.  
Open source is people.





Let's start with people who **use** software,  
the individual software consumers.

### individual user

uses open source but  
does not contribute

maybe donates?

unconcerned with  
software freedom

On the one hand, individual consumers aren't very interesting to a discussion of free software because they don't **contribute** to free software directly, and don't really **care** about software freedom as a concept. They participate in the cash economy and might occasionally contribute to a proprietary software program with a little money now and then, but software freedom per se is irrelevant to them. That might sound a bit dismissive, but we can infer how much individuals care about software freedom from the market performance of user-facing free software.



In the office automation world,  
the rise of an acceptable open source alternative to Microsoft  
barely dented the dominance of Office.  
OpenOffice and then LibreOffice have been around a long time,  
but Office reigned supreme until the coming of the cloud.



In the end it was the extra  
**convenience** of Google Docs for sharing and collaboration,  
**along** with a zero dollar price point,  
that finally threatened the dominance of Office in a meaningful way.



In the browser world, back in the early aughts,  
it looked like Firefox  
proved that "open is better",



supplanted by



as it rose and took over from Internet Explorer.  
But it turned out that actually "better is better".  
Users moved to Firefox because it was better than IE.



supplanted by



in turn supplanted by



And now, users have freely migrated back away from Firefox to Chrome, Safari, and even back to IE as each took a turn as the best performing, cleanest alternative on different desktop and mobile platforms. Individual consumers just **don't care very much** about software freedom.

"i don't care about software freedom",



"duh."

"because i don't write software!"

In fairness, to **exercise** software freedom, it generally helps to understand how to **write and build software**, and most individual consumers just **don't have** those skills. Now, that doesn't mean software freedom has no constituency among people who cannot write or modify software themselves, it just means their **experience** of software freedom very indirect.

“i don’t care about software freedom”,



“duh.”

“because i don’t write software!”

But if individual consumers **DID** care,  
it would dramatically alter our relationship with digital technology.

This is probably why Richard Stallman is  
still willing to talk to people about software freedom,  
after all these years:  
if **all people** cared about it, just a little,  
we would all be a **lot more free**.

*Congress shall make no law respecting an  
establishment of religion, or prohibiting the free  
exercise thereof; or abridging the freedom of  
speech, or of the press; or the right of the people  
peaceably to assemble, and to petition the  
government for a redress of grievances.*

Take a look at the First Amendment of the US Constitution:

Congress shall make no law respecting an establishment of religion,  
or prohibiting the free exercise thereof;  
or abridging the freedom of speech, or of the press;  
or the right of the people peaceably to assemble, and to petition the  
government for a redress of grievances.

### 1st amendment

protects people who  
criticize the government  
assemble to protest the  
government  
sue the government


So, the people most  
**directly served**  
by the 1st amendment are  
those who publish criticisms of the government,  
or assemble to protest the government,  
or sue the government for misconduct.

### 1st amendment

protects people who  
criticize the government  
assemble to protest the  
government  
sue the government

**but,**  
most people  
don't do  
those things

Now, while I understand that  
those categories of people  
have grown a great **deal** in the last year,  
they do not include **all the people**. (At least not yet.)  
There are lots and lots and lots of people  
who have no words to publish, people  
who do not wish or have time to assemble, people  
who have no case to litigate.

 <b>1st amendment</b> protects people who criticize the government assemble to protest the government sue the government	<b>but,</b> most people don't do those things <b>nevertheless,</b> most people won't give up the 1st amendment
--	--

And Yet.

And yet, I wager,  
even those who find **nothing to criticize**  
or protest  
or litigate about  
would be loath to see the 1st amendment repealed.

They don't **directly exercise** the freedoms of the 1st amendment,  
but they **do** appreciate  
the **freedom it provides them** at a remove.

 <b>software freedom</b> protects people who  examine and redistribute software  modify software  redistribute their modifications
---

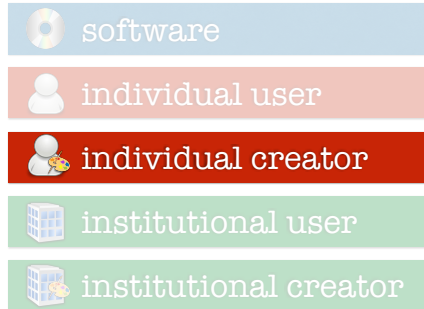
Imagine if every individual consumer  
valued software freedom as much  
as they value freedom of speech?  
How would our conversation about  
technology and privacy and  
control of our personal data be different then?

 **software freedom**  
protects people who

examine and  
redistribute software  
modify software  
redistribute their  
modifications

} **but,**  
most people  
don't care

However, for the moment, they don't care.  
When individual consumers use open source,  
the reasons  
don't have anything to do with software freedom:  
they use it because  
it doesn't cost money;  
or because it works better than the alternatives for their purposes;  
or their nerdy niece installed it for them.



So let's move on  
to the individual  
open source  
software **makers**.





We've already covered the mythic makers, the hacker-in-the-basement archetype. There's a place for the mythic framework, but we have to be really careful about applying it, because it's too easy to cover over complex truths with simple patterns. Some projects **do** start with a single contributor, and add on extra contributors over time. But a surprising number don't, even ones you might think have a clear provenance.

paul: "a spatial type in postgres  
would be awesome"

dave: "hold my beer"

People sometimes introduce me as the "founder" of PostGIS. And I am a person who has had a long, prominent association with the PostGIS project. <x> But way back in 2001, when I said "hey, it would be great to have a spatial data type in Postgres", it was Dave Blasby who said, "yes, and I know just how to do that" and actually wrote the code.



It wasn't me.  
Now, at the time,  
Dave worked at my consulting company, Refrations.  
Other folks at the company, Chris and Jeff,  
wrote some early code.  
So, who founded PostGIS?  
There's not a singular answer.

there are  
many **co-founders**,  
and starting is just  
the first step

So I usually say "no, no, introduce me as a **co-founder**",  
but there's even more to the story than that, which I'll get to later.

Let's put the mythic individuals aside for a moment,  
because **non-mythic** individual makers are the real story.

have a problem  
find some code  
ask a question  
answer a question  
recommend the code  
fix the code  
enhance the code

They have a problem to solve.

<x> They find the open source software that can solve it. They use it.  
It works.

<x> They ask a question about the software, they get an answer.

<x> They see a question, they provide an answer.

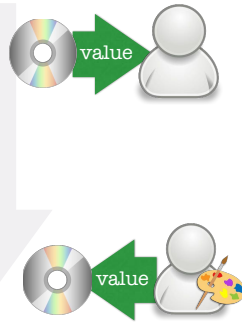
<x> They are asked if they use the software, they provide a reference.

<x> They find a shortcoming. They find a way to apply effort to getting a fix added.

<x> They need a new feature. They find a way to apply effort to getting that feature added.

Note that there's a

have a problem  
find some code  
ask a question  
answer a question  
recommend the code  
fix the code  
enhance the code



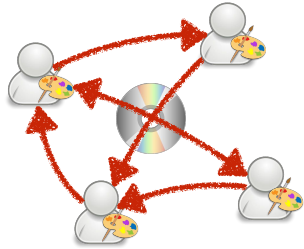
**continuum** of involvement here. At the start, the software and project are

<x> adding value to the individual. And the end, the individual is

<x> adding value to the software.

At the start, the individual acts alone. At the end, the individual is working as part of a community of interest around the software.

## gift economy



Once a project has sufficient **attention** to grow a community of interest of users and developers, another economy comes into being within that community: a **gift economy**.

## gift economy

A gift economy is a mode of exchange where valuables are not traded or sold, but rather given **without an explicit agreement** for immediate or future rewards.

A gift economy is one in which valuables are not traded or sold, but are given **without an explicit agreement** for immediate or future rewards.



The classic example of a "gift economy" is the potlatch tradition of the First Nations peoples of the Pacific northwest. Before European colonization in the 1800s, a powerful chief would demonstrate his power, and amass influence, not by hoarding more and more wealth,



but by holding a huge feast, a "potlatch" and distributing all his wealth to other families in his tribe and to neighbouring chieftains. Western capitalistic folks find it hard to understand the rationale behind a gifting culture,



to the extent that the ceremony was banned in Canada, from 1885 until 1951. This picture is from a modern day ceremony in the 1980s.

## gift economy

influence in  
open source communities  
is directly correlated  
with giving (of help and time)

In open source communities,  
the most influential people are almost always  
the most frequent and consistent contributors,  
the ones who give of their time most freely.



Someone like Tom Lane is a leader in the PostgreSQL community not (just) because he is very very smart (though he is) but because he holds an ongoing potlatch with his technical skills. He willingly answers questions, from all different members of the community; he swiftly fixes problems brought to his attention; he takes on difficult core problems of development. He gives heavily of his time and skills, and in turn amasses influence.

## gift economy

“influence” = “social capital”

(see? it's kind of like money...)

Another word for "influence" is "social capital". When I answer your PostGIS question on the mailing list, I do not receive cash payment but (assuming you aren't a sociopath) I incur a small sense of reciprocal obligation in you. I earn a little bit of "social capital".

## gift economy

social capital can buy:

- a fair hearing
- a swift answer
- the above in other communities too

What good is "social capital"? Here's some things you can buy with "social capital".

<x> Communities are often skeptical of big new features or changes, because they can be disruptive; someone with social capital can get a fair hearing on a big scary new feature.

<x> A developer may not be expert in all areas of a code base; but, if she has social capital she can tap other experts to get a swift answer to her questions.

<x> Also sufficient standing in one community enables direct collaboration of one project with another: when Tom Lane makes a comment about PostGIS, I listen; if he provides a patch, I apply it. He has social capital that is transferrable.

## gift economy

tom lane has:

- a huge amount of social capital with the postgres community



Now, it might seem that Tom Lane is getting a raw deal: in return for hours and hours of highly skilled effort invested in the Postgres community, all he gets back is influence and "social capital" in the Postgres community.

That's a pretty circular benefit: in return for being a gracious expert he gains privileged access to other gracious experts.

Not something that is going to put food on the table.



## gift economy

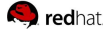
tom lane has:

- a huge amount of social capital with the postgres community



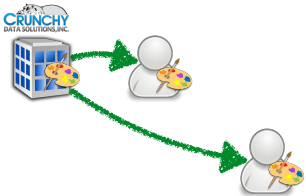
tom lane is highly valued by:

- companies that need or sell postgres expertise



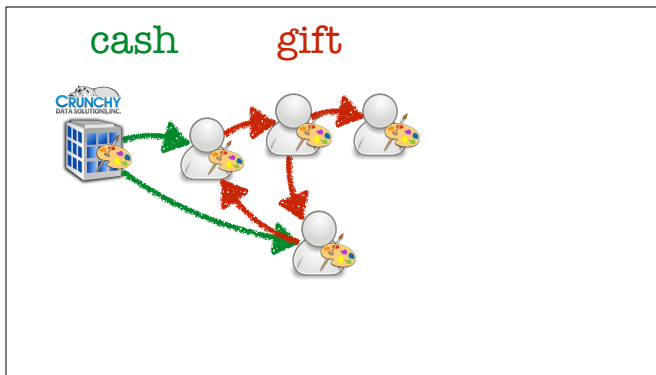
However,  
because he is an influential contributor to an important software project,  
because he is the owner of a huge amount of social capital in the Postgres gift economy,  
he is also a highly desirable employee for companies that use or support that software.  
Over the past 10 years,  
Tom Lane has worked for RedHat, for Salesforce.com  
and currently for Crunchy Data.

## cash

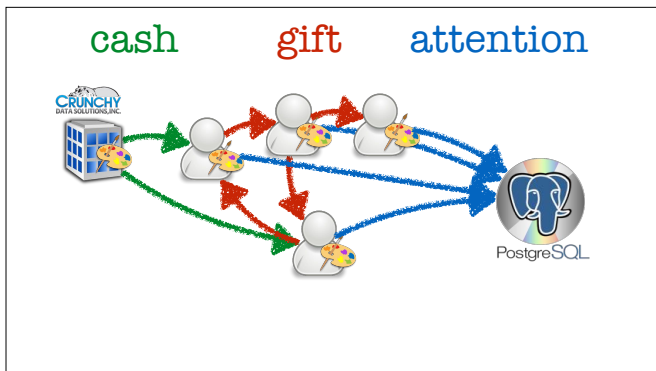


So, here the cash economy finally rears its head, as the three economies interact.

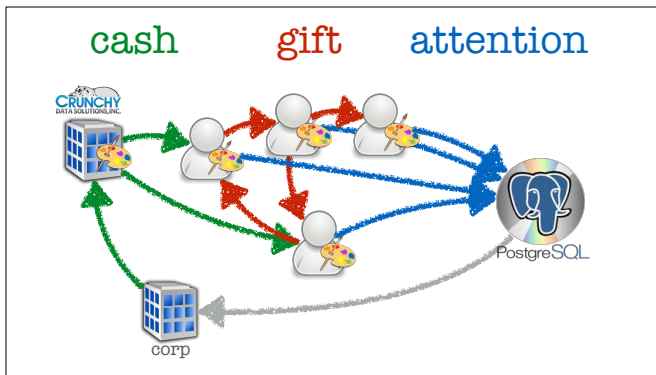
- Crunchy Data pays some PostgreSQL people, like Tom



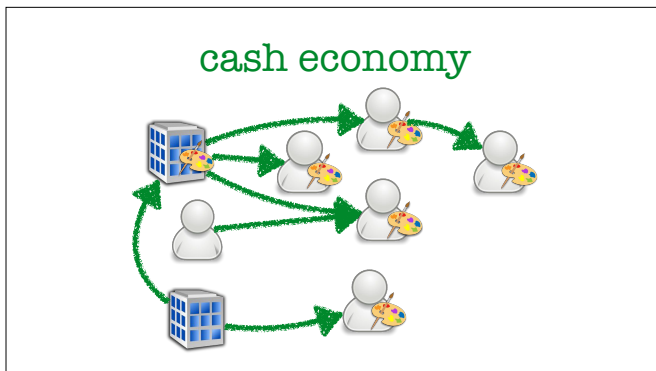
- Who work as part of the Postgres community
- And generate social capital and influence for themselves



- And apply attention to the Postgres software
- So the software lives and grows and becomes useful to other organizations



– Who may in turn pay Crunchy Data ...  
And then the cycle repeats.



The cash economy is the economy we  
worry about the most when thinking about open source,

## cash economy

open source is great at  
**creating value**  
open source is terrible at  
**capturing value**

and for good reason:  
open source is great at creating cash value,  
but very poor at capturing it.



<https://www.fordfoundation.org/library/reports-and-studies/roads-and-bridges-the-unseen-labor-behind-our-digital-infrastructure/>

Last year, a study by Nadia Eghbal for the Ford Foundation investigated the deficit in spending on open source software infrastructure.

## digital infrastructure

frameworks

django



libraries

OpenSSL



Software frameworks like Django or Rails,  
libraries like jQuery or OpenSSL.  
Software infrastructure.

Infrastructure is the **least visible** open source software,  
it's the kind of thing you use without even knowing you're using it.

What she found was dismaying:

These projects had present, not  
future, value. They were actively  
used by Facebook, Instagram,  
Pinterest, Netflix, even governments.  
They directly caused tech's rapid  
rise, ... But they hadn't captured the  
financial value they deserved.

– Nadia Eghbal

“These projects had present, not future, value.  
They were actively used by Facebook, Instagram, Pinterest, Netflix,  
even governments. They directly caused tech's rapid rise, ...  
But they hadn't captured the financial value they deserved.”  
These infrastructural projects were  
disproportionately maintained by individuals or  
part-time by small groups.”

So,  
The software is **loved** by the marketplace.  
The maintainers... not so much.

@pydanny

I personally get regular **demands** for unpaid work... by healthy high profit companies large and small...  
If I don't respond in a timely fashion, if I'm not willing to accept a crappy pull request, I/we get labeled a jerk

I personally get regular **\*demands\*** for unpaid work...  
by healthy high profit companies large and small....  
If I don't respond in a timely fashion,  
if I'm not willing to accept a crappy pull request,  
I/we get labeled a jerk.

@andrewgodwin

Just relying on people's good will isn't going to work, we'll end up disproportionately appealing to independent developers or developers on a personal level and that's not as sustainable I don't think.

Just relying on people's good will isn't going to work,  
we'll end up disproportionately appealing to  
independent developers or developers  
on a personal level and that's  
not as sustainable

@shazow

Publishing and contributing to open source is going to continue happening regardless whether I'm getting paid for it or not, but it will be slow and unfocused. Which is fine, it's how open source work has always worked. But it doesn't need to be this way.

Roads  
Bridges: FORD FOUNDATION

Publishing and contributing to open source is going to continue happening regardless whether I'm getting paid for it or not, but it will be slow and unfocused. Which is fine, it's how open source work has always worked. But it **doesn't need to be this way**.

This is not just a problem for generic infrastructure projects. We in geospatial are not immune to infrastructure under-investment either:

martin  
davis  
*in his sparetime*  
maintains  
jts

Martin Davis,  
the core developer of the JTS algorithms library,  
used by almost all our open source geo software,  
either directly or via the GEOS C++ and JSTS Javascript libraries,  
has a day job that involves precisely zero JTS work.



The Proj4 reprojection library, which also underlies almost all our software, is maintained by Howard Butler, whose consulting company works mostly in point clouds: there's no directly funded work for it.



The GDAL image library, which **is also** used in almost all our software, is maintained incredibly well by Even Roualt, but as a contract developer Even is usually paid to add **new** features and formats to GDAL. He provides most of the critical maintenance, stability and release work on his own time.



## cash economy

need money?  
just use a  
**proprietary  
licensing**  
model!

but that  
**removes** all  
the software  
**freedom**

The question is, how can it be another way?  
The knee jerk reaction is to say  
"hey, the proprietary model works to  
force value from the cash economy into the software attention  
economy,  
use that!".  
<x> Unfortunately,  
the two models cannot be reconciled:  
proprietary software depends on the restrictions of use,  
while open source is all about removing those restrictions.

## cash economy

community property,  
public goods,  
free things have value

If we are going to keep our open source ecosystem healthy,  
we need to start recognizing  
the **value**  
of **free things**,  
the **value**  
of **communal property**.



Back home in Victoria,  
there is a petting zoo in the central park.  
Admission is free by donation.




I always donate.  
Now why would I do that?  
I can get in for free!  
I donate because I like the zoo.  
Because I want it to **still be there** the next time I go to the park.  
I donate because the baby goats are nice.  
Everyone likes baby goats.



I also like open source,  
because it has software freedom.  
As a consultant,  
I've designed systems,  
with both open source and proprietary components,  
and my reasons for preferring open source components have been

**what i like:**

- no license overhead
- transparency of components
- flexibility of components
- access to expertise



I value the avoidance of license overhead,  
both the monetary overhead of buying the stuff  
and the administrative overhead of maintaining license compliance  
<x> I value the transparency of components,  
avoiding expensive black box debugging  
and custom shims between theoretically off-the-shelf proprietary  
software  
<x> I value the flexibility of components,  
allowing features to be added without re-architecting the system,  
pushing necessary improvements into the upstream projects  
<x> I value the access to expertise (often core developers) via the open  
source community,

**what i like:**

no license overhead  
transparency of  
components  
flexibility of components  
access to expertise

I ❤️

software  
freedom

Except for the last item,  
my reasons for preferring open source components  
**all flow**  
to some extent  
from the principles of software freedom.  
The freedoms to examine, modify and distribute the source code.  
So, those are **MY** reasons for preferring open source components.



## Open Source Survey

The Open Source Survey is an open data project by  
GitHub and collaborators from academia, industry, and  
the broader open source community.

<http://opensource-survey.org/2017/>

Not necessarily everybody's.  
This year, Github ran a randomized survey of  
what they said was:  
"anyone who uses or otherwise engages  
with open source technology and development,  
whether passively or actively through contributions."



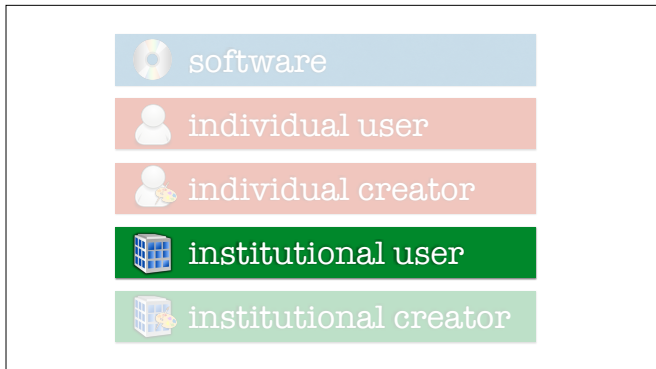
From the survey,  
the top two reasons people choose  
open source software over proprietary software are:  
"stability", and "security".

Now, the open development process favoured by open source  
communities tends to produce stable and secure software,  
but neither stability nor security naturally follow from software  
freedom.

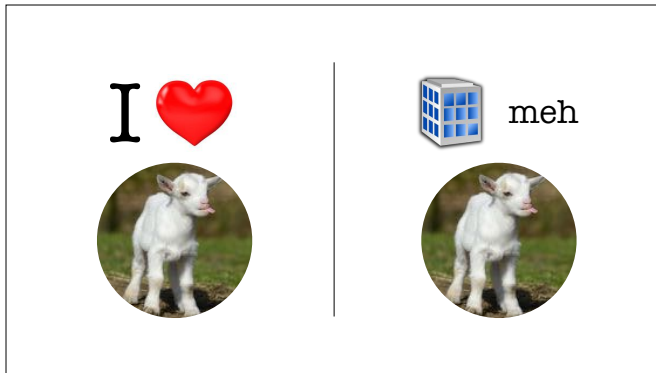
Proprietary software is not automatically less stable, or less secure.

This is really worrying to me.

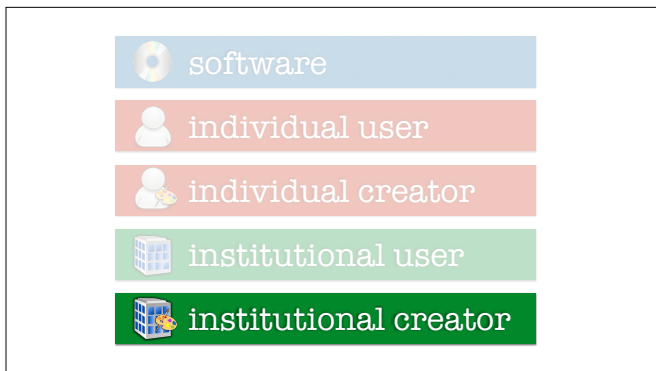
The top reasons cited for using open source software have nothing to  
do with the open source nature of the software.



The implication is that,  
for pure users, at least in this survey,  
institutions are no different from individuals:  
as long as the cost is equivalent, they'll choose the stable, secure  
product.



Currently that happens to be open source software, quite frequently, but presumably that could change any time. So we cannot necessarily count on institutional users to keep baby goats warm and fed.



What about institutional contributors, why do they code? The field of institutional contributors is incredibly diverse, and there are a lot of reasons they contribute. Here are some reasons different kinds of institutions have for contribution, arranged in order of enlightenment.



## institutional creator

### open source support companies

- must employ experts with deep knowledge
- need access to, reputation in community

enlightenment 

Open source support companies like Red Hat or Boundless or Hortonworks are obviously very enlightened, you can see they score four buddhas on the enlightenment scale. And this should be no surprise... Their **whole value proposition** is having the **best** expertise on the **best** software, so investment in open source is a no-brainer.



## institutional creator

### software-as-a-service companies

- built on open source technology
- depend on deep knowledge for reliability, new features

enlightenment 

Software-as-a-service companies like Carto or Mapbox usually are built on open source, because it gets them to market quickly, with flexible components at low cost. And once in the market, the value proposition of a SaaS company is about features, reliability and performance. An enlightened SaaS company will know it needs direct access to expertise on the components their software is built with, or risk losing customers to costly downtime or surprising bugs.

## institutional creator

### proprietary software companies

- sometimes embed open source
- may realize they are exposed to risk if they don't have skills on tap

enlightenment 

Proprietary software companies like Microsoft or Esri sometimes embed some piece of open source in their larger software application. Embedding saves them development time and effort. If they are enlightened, they will maintain a relationship with the open source community in case they need extra features or encounter a bug they cannot quickly fix themselves. But honestly, most don't.

## institutional creator

### systems integration companies

- often build systems with open source
- frequently ignore the value that open source brings to their solutions

enlightenment 

Systems integration consultants like IBM or HP or CapGemini build systems for customers using open source components. They may sell themselves as experts in the open source components, or they may just use it under the covers without telling the customer. Regardless, an **enlightened** consultant will maintain a relationship with the open source community in case they need extra features or encounter a bug they cannot quickly fix themselves. But again, most don't.





institutional creator

big internet/cloud companies

- built on and originating their own open source
- recognize their risk exposure and need to engage community

enlightenment 🧘 🧘 🧘

Great big companies like Facebook or Google or Salesforce contribute because they have a large exposure to a particular piece of software, and they recognize the risk and opportunity in that exposure. Even non-technology companies can be enlightened. Enlightened insurance companies are known to keep a core open source database developer on staff for support purposes. Enlightened high frequency trading firms employ linux kernel developers for performance tweaks.



institutional creator

most companies / institutions

- might build systems on open source
- treat it like magic fairy dust

enlightenment

All that said, most organizations show an incredibly low degree of enlightenment. Many companies or governments or NGOs **USE** open source. **Very few** of those organizations **recognize** their dependence on open source projects and direct their resources accordingly.

## enlightened institutions



### large

- know they depend on OSS
- hire a developer

### small

- know they depend on OSS
- buy a support contract
- retain a contractor

For projects that are visible,  
projects that organizations **know** they are using and depending on,  
it's possible there will be enough enlightenment  
that organizations will willingly devote resources to their upkeep:  
they will hire their own resources if they are big enough;  
or, if they are smaller they will purchase support from a support  
company.  
But, relatively few of them seem to be that enlightened.

"My god, all we're  
doing is selling  
insurance! How is that  
hard to understand?"



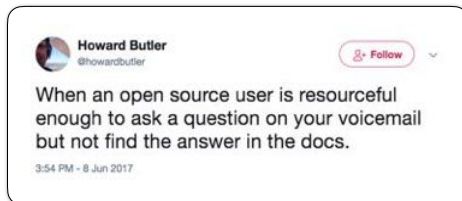
**eddie pickle**, on the difficulty of  
selling open source support

Back when I was at Boundless,  
which is a geospatial open source support company,  
the CEO at the time, Eddie Pickle, used to say to me:  
"My god, all we're doing is selling insurance!  
How is that hard to understand?"  
And he was a little exasperated,  
because it seems very very hard indeed  
for the customers to understand.

open source cash economy  
is like an  
open health insurance market

- when you feel OK,  
you don't think  
you need it
- when something  
goes wrong, you go  
to emergency

This is the USA, so everyone has purchased health insurance, right? It just makes sense, after all, because, you know, you might get sick? Except, we know that open health insurance markets don't work that well, that all too often, people do not or cannot get insurance. Left to their own devices, the young and healthy often don't bother to get health insurance, so the insurance markets are dominated by the old and sick, premiums go up, and when catastrophe strikes the uninsured show up at the emergency room.



Open source users also go to the emergency room. The strange voicemail messages, the private emails with "URGENT" in the subject line, the twitter and github @harassment. These are all things that open source developers have experienced. These are all trips to the software emergency room that some users want to get for free.

“My god, all we're  
doing is selling  
insurance! How is that  
hard to understand?”

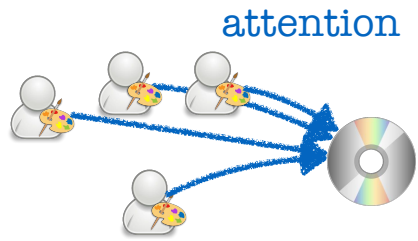


**eddie pickle**, on the difficulty of  
selling open source support

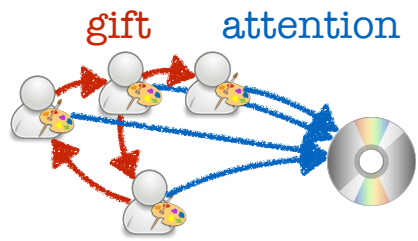
"My god, all we're doing is selling insurance!  
How is that hard to understand?"

the **good** news

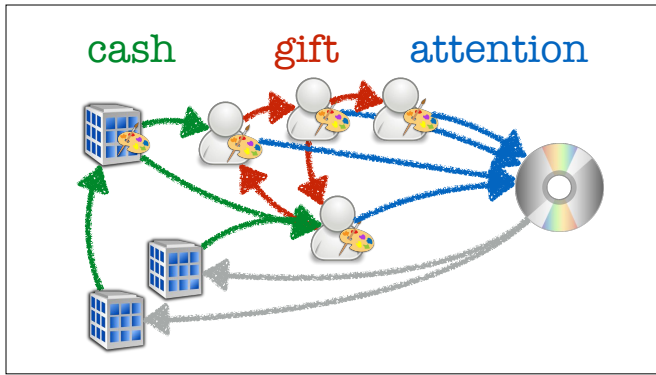
Still, there's a lot of good news in the open source economies:



the attention economy of open source software  
is good at weeding out weak projects,  
and amassing critical numbers of developers  
around successful projects



the gift economy of open source communities  
promotes collaborative instincts  
and can provide core contributors with  
the leverage necessary to make a living in the cash economy



the cash economy is doing a passable job  
funnelling resources into the larger,  
more visible open source projects,  
from support companies,  
from SaaS companies and  
from large institutions enlightened enough to recognize  
that buying "insurance"  
for their software assets  
makes sense

the **bad** news

But there's a no small amount of bad news too:

open source cash market  
poorly directs funds

#### infrastructure

- underfunded
- ignored
- taken for granted

#### user facing

- funds favour **featuritis**
- maintenance is considered "overhead"

the invisible but necessary infrastructure projects are chronically under-funded, and user facing projects all too frequently suffer from featuritis as funding and effort crowds around adding the **next** cool thing, rather than making the **current** thing more stable or faster



CORE  
INFRASTRUCTURE  
INITIATIVE

IBM

facebook

amazon

Google

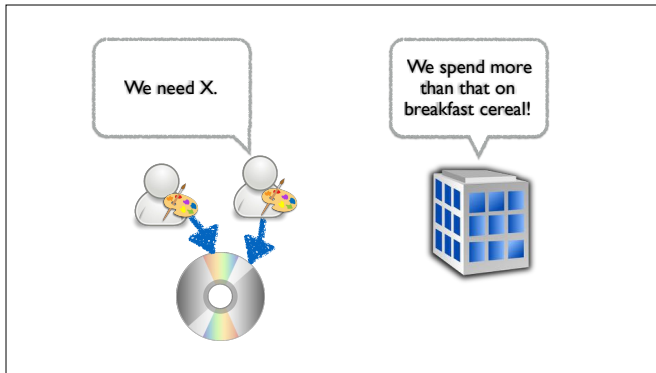
Microsoft

salesforce

- Frama-C
- GnuPG
- Network Time Protocol
- OpenSSH
- OpenSSL
- OWASP Zed Attack Proxy
- Debian Builds
- Fuzzing Project
- Linux Kernel Self Protection Project

Some of the largest companies, the Googles and Facebooks and Microsofts, have put together a small program called the "Core Infrastructure Initiative" to fund open source infrastructure, but their focus is generic network and coding infrastructure.

If we wait for the Core Infrastructure Initiative to help with **geospatial** open source infrastructure, we'll be waiting a long long time.



There are great things to be done here,  
there is a match made in heaven here,  
because open source projects  
require **very few resources**,  
relatively speaking,  
and institutions control a great many resources.

**do you work for an  
enlightened  
institution?**



I keep talking about **institutions**,  
because the **cash resources** of **institutions**  
dwarf the amount of  
spare time and unpaid vacations  
that open source community members  
can spend on software.



- 👉 allows you to work on OSS?
- 👉 has an OSS support contract?
- 👉 uses an OSS contractor?
- 👉 uses a SaaS vendor that supports OSS?
- 👉 donates to a program like Core Infrastructure?

If you are **here** as an employee of an institution, your institution is probably already enlightened, but just to make sure, here's a checklist:

- Does your institution allow or encourage you or a co-worker to spend time improving the projects you use?
- if not, does your institution have a **support contract** with an open source support company?
- or, does your institution have a **direct** contract with an open source developer?
- or, does your institution use a software-as-a-service that supports open source?
- or, does your institution donate to a program like the "Core Infrastructure Initiative"?



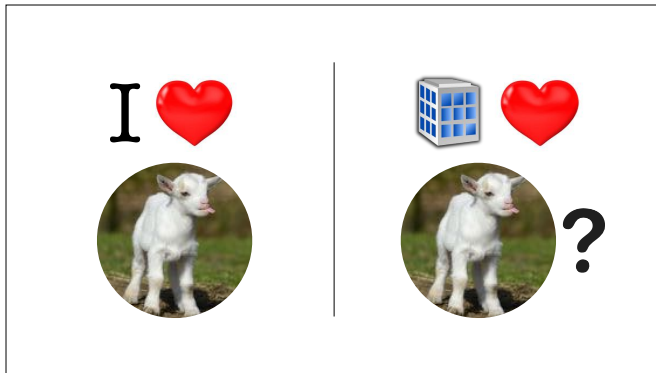
#### no longer critical

- mailing lists
- source control
- wiki space
- download space

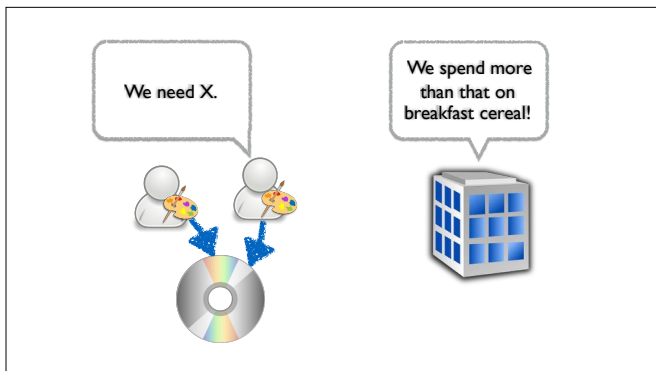
#### now critical

- risk evaluation
- infrastructure support
- shared funding

OSGeo may have a role to play here, if they are able to take it up. Back when OSGeo was formed, communities lacked good technical infrastructure, and OSGeo provided some: mailing lists, source repositories, ticket trackers. But technical infrastructure is not a limiting step anymore. The missing piece **now** is financial support for core software infrastructure. For projects like GDAL, and Proj, and JTS. Projects that underlie all the other, more visible software, that we use every day.



If we're going to start telling users that "admission is by donation", if we are going to make an argument for taking open source sustainability seriously, we need to provide a channel for that support to flow to the developers.  
OSGeo could rise to the challenge.  
Somebody else could.  
I'm just going to leave that out there.



So, I've laid down a lot of responsibility at the feet of employees of institutions. Boring responsibility for making sure **old software** is maintained.

But that's not all there is. Institutions can be important innovators, if their employees are willing to grasp the role and take the plunge.

origins of  
**PostGIS**



story

dave & paul  
and jeff & chris  
and phil & graeme  
and regina & carl & frank  
and ...

and  
?

I want to finish my story about the origins of PostGIS, because there's a character in there I didn't mention, who you've never heard of.

<x> He's not a mythic open source founder.

He's not a coder.

But without him, there might not be a PostGIS.

2001

So, back in 2001, after I said "wouldn't it be great" and Dave said "yes, and I can do that", and after a few weeks of work,

postgis 0.1

we had a primitive version of PostGIS.  
But it didn't do much, beyond storage and retrieval,  
it was lacking most of the  
computational geometry algorithms we expect in GIS.

his vision:  
GIS without a GIS

Fortunately a civil servant in the  
British Columbia government had a vision  
for a toolkit to allow the government  
to build GIS data processing systems  
without installing big GIS software packages.

secured federal funds

He secured funding from the federal government via an industrial investment program,

contracted local  
developer

---

and contracted with a local company to write the software.  
The software he contracted for was called the Java Topology Suite,

developer:  
martin davis

software:  
java topology suite (jts)

JTS,  
and the contract he wrote  
specified that it be delivered under an open source license.  
Because it was open source, we were able to port that  
software to C++ and then use it,

---

postgis 0.8

for what became PostGIS 0.8,  
the first version that could do real geo-processing.

first postgis  
production system

That same civil servant also contracted with us to manage the provincial road centerline network **and** encouraged us to use PostGIS as the database for the project.

That was the first production deployment of PostGIS. He also encouraged us to use hours from the contract to improve PostGIS to handle the roads data faster and more accurately.

not afraid of  
open source bids

When we bid on contracts to manage and improve hydrography data, he didn't worry that our solutions were based on PostGIS and open source Java instead of Oracle and Esri, he just made sure we could produce the results he wanted.

led to postgres 1.0  
performance  
lift

The improvements Dave made to PostGIS  
during **those** contracts  
led directly to the PostGIS 1.0 release.



mark  
sondheim




**JTS/GEOS/JSTS**  
are used by



You've never heard of this guy, his name is Mark Sondheim,  
but you've probably used geospatial software  
that works because of decisions he made,  
over 15 years ago,  
as an employee of a big boring,  
government institution.

All these projects  
(QGIS, PostGIS, Fiona, GDAL,  
GeoServer, GeoTools, Turf.js,  
GeoDjango)  
use GEOS or JTS.






mark  
sondheim

BRITISH  
COLUMBIA

**JTS/GEOS/JSTS**  
are used by



PostGIS

GeoServer

MapServer

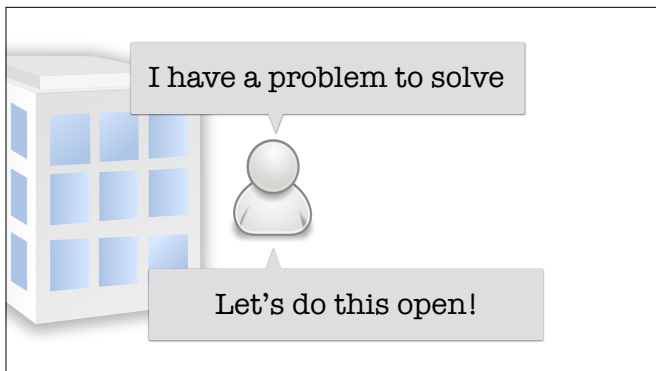
GDAL

QGIS

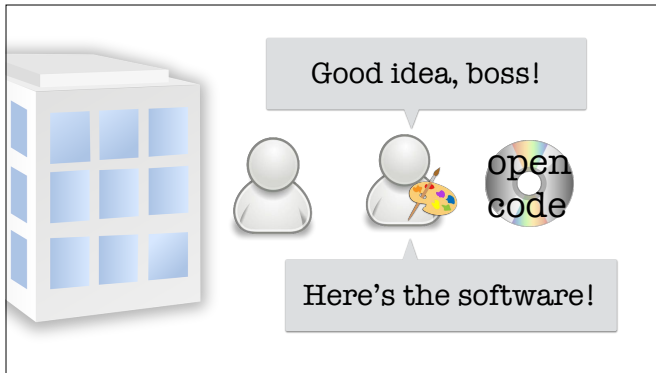
TURF

We have a whole track just for PostGIS here at FOSS4G this week.

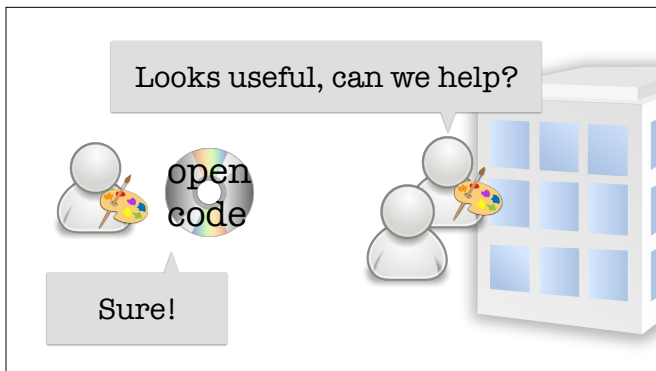
The most important people in the open source ecosystem are not the mythic hackers, they are the people who have **access to resources**. They're people like many of you, who can direct resources, or who advise those who can.



These are folks with problems to solve. They make decisions about how to apply resources to the problem.



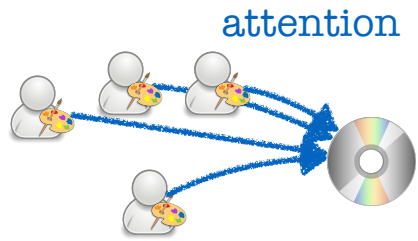
Those every-day decisions,  
in big boring institutions,  
are what lay the foundations for others.



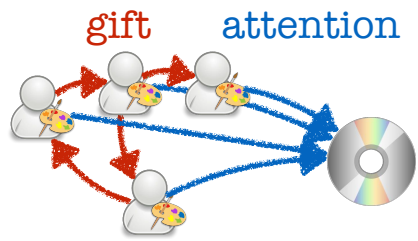
Who will make their own decisions.  
If these people work in the open,  
if they value software freedom, they can work together.  
If they work together, they can eventually get more value out, than they  
put in.

That's the open source economy in action.  
So why do we code?  
More precisely, we do we code open source?

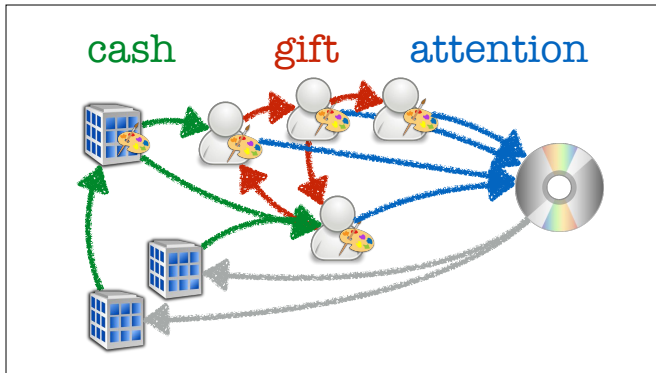
Because there are multiple economies in which it makes sense.



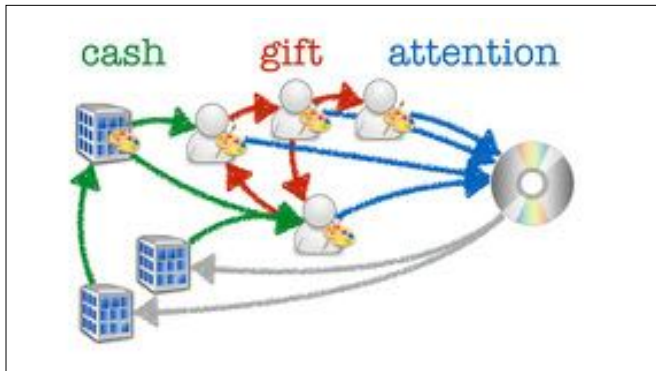
The software attention economy,  
gathering development effort around the winning software,  
winnowing out the losers and the also-rans



The developer gift economy,  
rewarding altruism and community mindedness  
with social capital and influence



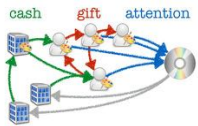
The cash economy,  
where the value derived from open source is worth billions,



and even the (relatively) tiny amount  
that flows back is circulating and continuing to  
provide developer effort for the software

“why we code”

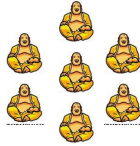
makes  
sense



baby  
goats



right  
thing



We code open source because  
it makes sense, ethically and financially.  
<x> We code open source because  
we like the baby goats.  
<x> We code open source because  
it's the right thing to do.

Thank  
you  
very  
much,  
have a great conference.