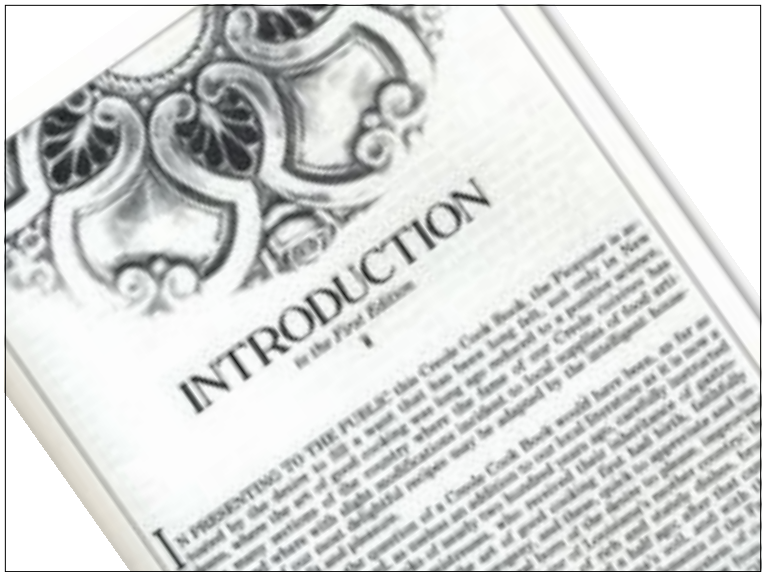
#foss4gna

FOSS4G NA 2012
Washington DC

In about half-an-hour, Keith Barber will get into the real content of the conference, the specific hows and whys of open source in a real organization.
But first, in the meantime, while we limber up....



I've been asked to provide the "introduction", the meta-content, the backroad ramblings, the “content souflee”
if you will, that will, hopefully, help contextualize the conference to come.

FOSS4G

You are at FOSS4G!
That's how it's pronounced. Foss for gee.
Not fooooosfergg or fussforge.
I admit, it sounds a little weird at first, but there it is.
Foss for gee.

Free
Open
Source
Software
4or
Geospatial

That is, the Free, and Open Source, Software, FOR Geospatial conference.
North American edition
This morning, from all over the continent and some from even further afield,
there are 360 of you,
Welcome!

- 51 Presentations
- 3 Keynotes
- 1 Panel Discussion
- 1 Code Sprint
- 12 Ignite Sessions
- N Birds-of-a-Feather Sessions

The next two days will see
51 presentations,
3 keynotes (not counting what I'm about to subject you to),
an experts panel,
a code sprint,
a dozen high-speed high-wire Ignite presentations,
and as many birds of a feather sessions as you care to organize.

Birds-of-a-Feather Sessions

Birds of a Feather "BoF" sessions are self-organized, self-scheduled sessions where groups can get together and discuss topics of interest.

Birds-of-a-Feather Sessions

- Held in Room 101
- Schedule / Sign-up Board @ Room 101
- No Birds are Involved
- BoF sounds-like “Boff”

We've allocated room 101 for BoF sessions, and there is a
board at registration where you can add your session and see what sessions are available.
For those of you who are new to the concept, I'd like to explain Birds-of-a-Feather.
No birds are actually involved.
But "BoF" sessions are a lot like open source.

There is no central committee saying what has to happen, or when.
One person posts a discussion topic and time on the BoF board.
Other people interested in that topic go to the room,

and then they talk about until they are done.
That's it.
I'm not going to sugar coat this: there will probably be nerds there.
In fact, if you're there,
it's entirely possible you're a nerd.

The quality of a BoF will be determined by who shows up, and how willing they are to participate and share their knowledge.
Some sessions will be great, some will not.
It's not the responsibility of the organizer to put on a good BoF (though a good topic helps), it's the responsibility of the attendees.
It's up to everyone.

Open Source Culture

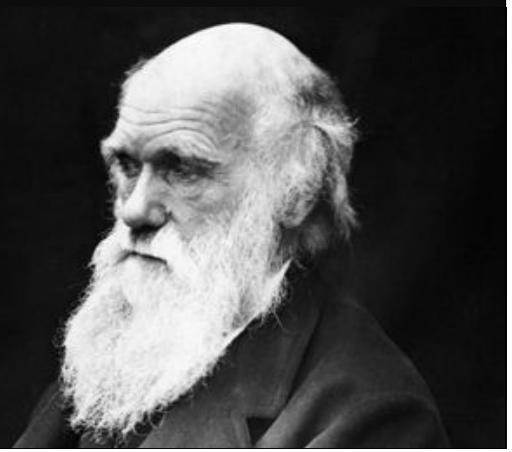
That's open source culture:
self-organization,
sharing knowledge,
generousness with ideas,
group collaboration and communication.
This kind of culture is not unique to open source, of course.
It's a culture that is common to the most important intellectual structures in our civilization.
Little things like ... whaddaya call it.?....

Scientific Culture

Charles Darwin

14,000+ Letters

3 per day, every day



"science" were built on
sharing of knowledge, generousness with ideas, and group communication.
Charles Darwin wrote at least 14,000 letters in his career (those are the ones we know about).
In the 26 years between his return from the voyage of the Beagle until the publication of the Origin of Species, he would have averaged 3 letters a day, every single day.
Letters to naturalists, to other scientists, to people who contacted him with their questions.
He shared his knowledge, and in the process of sharing he enhanced it and grew it.

Open Source Culture

Open source culture comes out of the same spirit of sharing and collaboration and knowledge building,
and I'd like to explore that spirit now, with a little historical story.

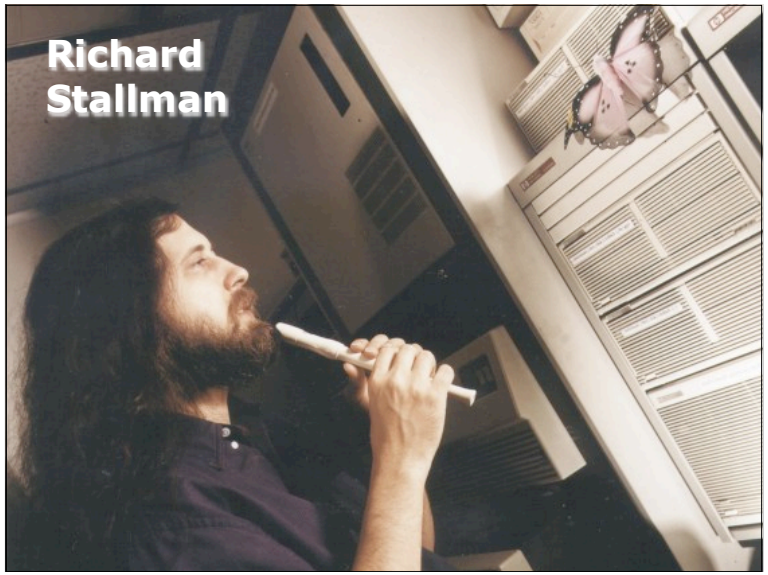
because we all like stories,
particularly stories about
computers, and software, and nerds,
right? right?

once upon a time...

Once upon a time,
there was a young man with
wild ideas about freedom,

who took on the established order of things,
appeared to lose,
but in the end changed the world forever
(though perhaps in ways he might not approve of).

Actually, not that young man,



though there is a striking resemblance...

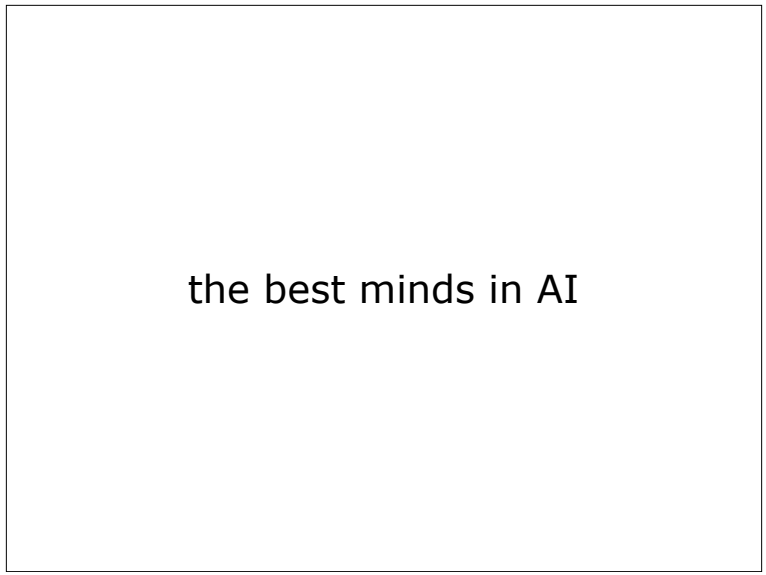
In 1980,
Richard Stallman was a programmer

Massachusetts Institute of Technology

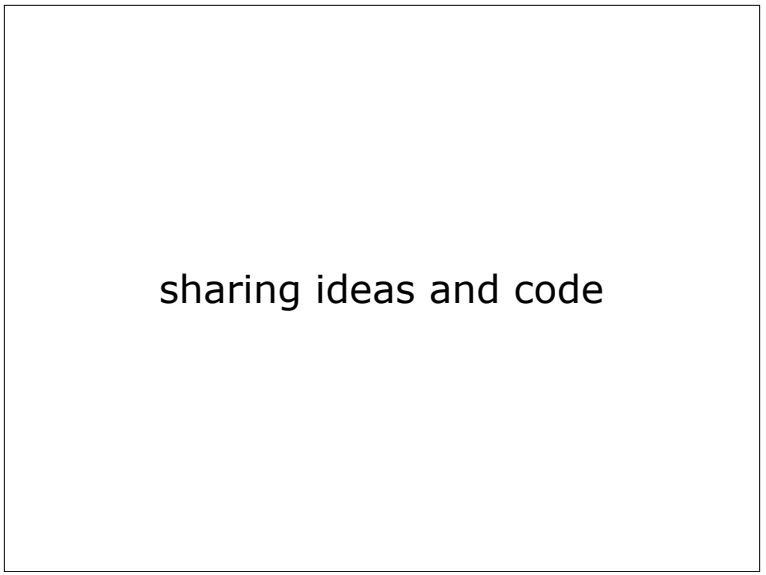
artificial intelligence lab

circa 1980

at the MIT Artificial Intelligence lab.
Some of the best minds in the



artificial intelligence field worked together



and shared ideas and implementations of those ideas in code.

It was, to hear Stallman tell it,

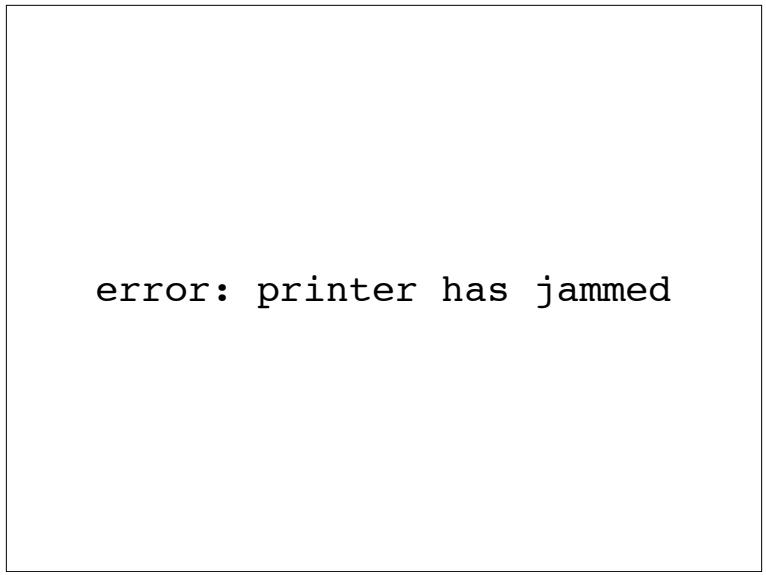
golden age
of hacker collaboration

a brief golden age of collaboration and intellectual ferment.

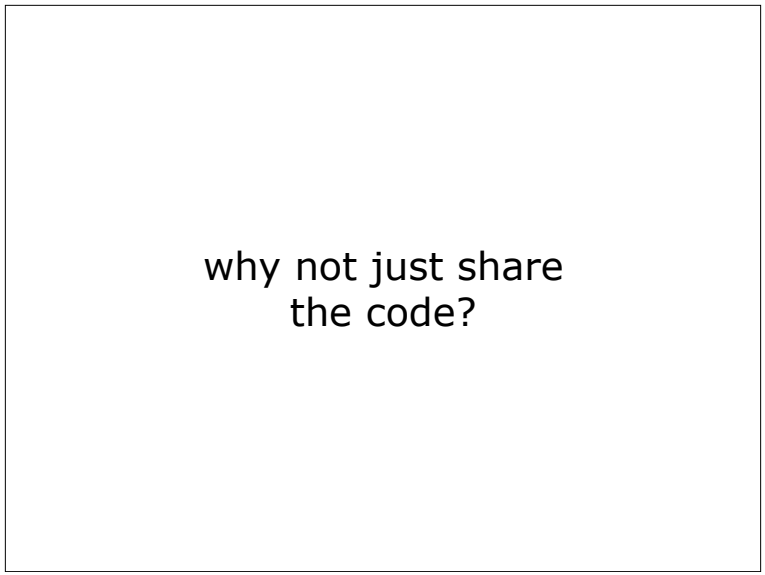
Then one day,
and don't all horror stories start this way,
one day,

the lab got a new printer (a xerox 9700).
Unlike the printer it was replacing,
the new printer came with a binary-only printer driver;
the source code was not included.

Stallman had modified the previous driver to send a message to users



when the printer jammed.
With the new binary driver
he couldn't do that.
The situation was *inconvenient*, it was a pain.



Why couldn't Xerox just share their code?
Everyone would be happier!

Most people might have shrugged.
But for Stallman it was a galvanizing moment.
Over the past five years working in the AI lab,
he had grown used to
sharing code and ideas with other programmers.

But now the atmosphere in computing was changing.
It wasn't just the printer driver.
A private corporation had started recruiting his colleagues in the lab.
Once hired, they were no longer allowed to exchange code with him.
The old computers in the lab, and the software that ran on it, were becoming obsolete.

The new computers being purchased by the lab included operating systems that were locked down:
you had to sign nondisclosures just to use them.

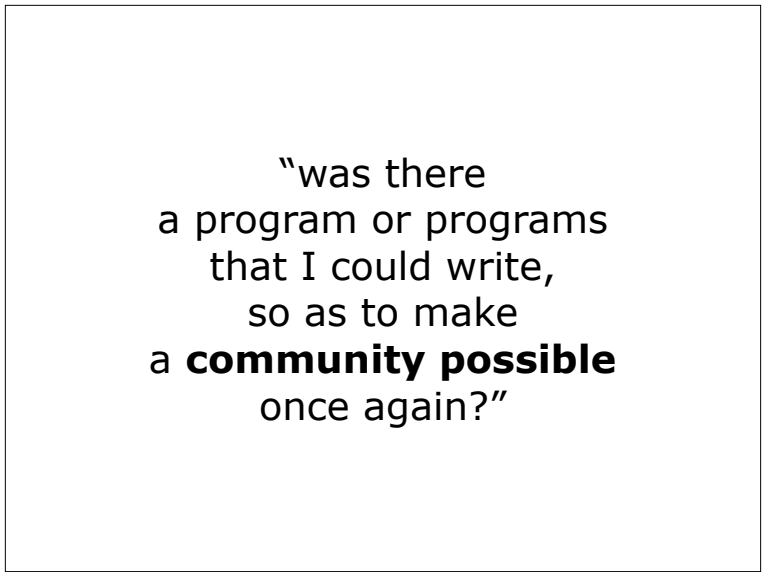
It was the death of the old collaborative community.
Stallman worried that

"the first step in using a computer was to promise *not* to help your neighbour" by accepting a license agreement.

As a highly talented and idealistic computer programmer, Stallman wanted his work to serve a larger purpose.

The financial promise of working in the growing proprietary software industry was not enough,
nor was the sterile intellectual amusement of continuing his work alone in academia.

Facing the death of his old intellectual community, Stallman asked himself



"was there a program or programs that I could write, so as to make a *community possible* once again?"

You can't use a computer without an operating system. So Stallman decided that, first,
he needed to write an

operating system.
It had to be
<X> portable to many computer platforms, it should be
<X> compatible with the popular new UNIX operating system to make it easy for people to run their existing programs on it,
and most importantly it
<X> should be **free**.

By "free", Stallman meant
you should be <X> free to run it,
you should be <X> free to modify it,
you should be <X> free to share it,
you should be <X> free to share your modifications

“free” as in “freedom”

 logiciel libre
 software libre
 il software libero

liberty

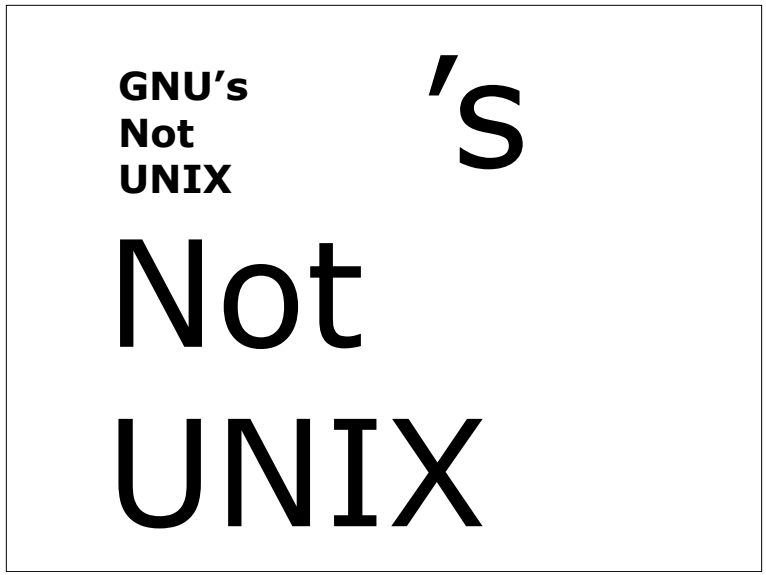
GNU’s Not UNIX

In a latinate language like French, Spanish or Italian
it's more obvious that, Stallman isn't talking about price,
not "logiciel gratuis", he's talking about <X>
[fr] logiciel libre,
[sp] software libre,
[it] il software libero.

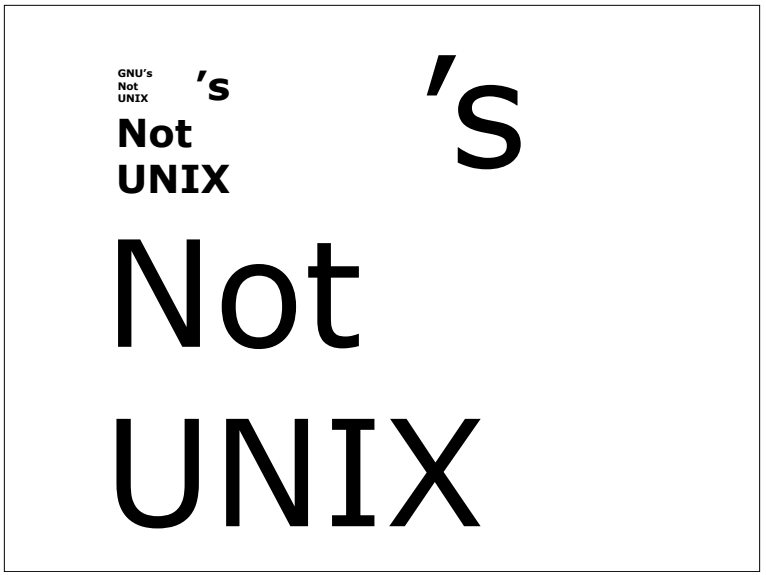
He's talking about
<X> liberated software... The key addition is liberty.

So, in 1984,
rather than join a computing industry that he considered morally bankrupt, Stallman decided to basically
start a **new one** from scratch.
It was an audacious plan.

Stallman called his new system
<X> GNU, which stands (recursively) for
<X> “GNU's Not UNIX?"
(See the recursion?) “what’s GNU?”



GNU's not UNIX.
What's GNU?



GNU's not UNIX.

copyleft

“all rights reversed”

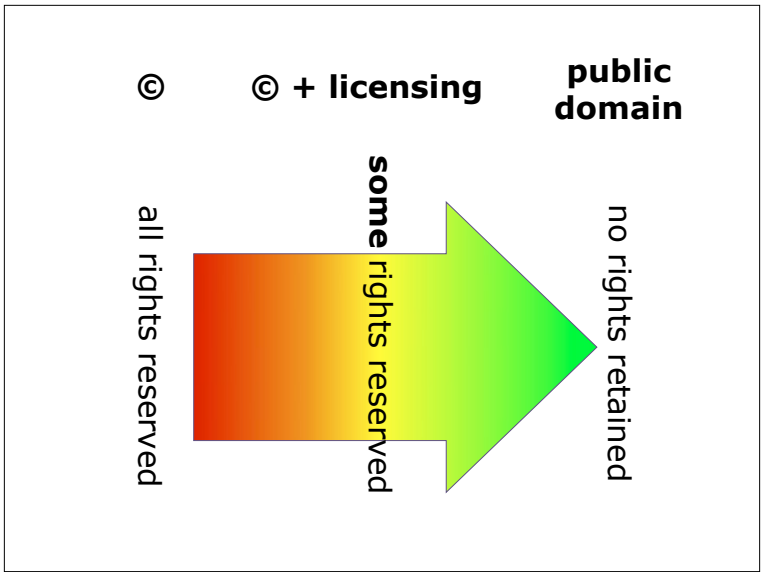
© all rights reserved

public domain no rights retained

In order to ensure GNU remained free, and did not get subsumed into a proprietary system in the future, Stallman released his work using a scheme he called "copyleft".

Generally speaking,
intellectual works
(books, movies, songs, computer programs)
are either under copyright or public domain.
The author either retains
full control over the work,
"all rights are reserved",
or no control,
"no rights are retained".

Copyleft,
and open source licenses in general,



use the copyright system
to selectively grant permission
and exert control over software through
licensing.

Authors retain copyright, but grant liberal usage rights
via a license.

The copyleft license grants permission
to all recipients of the code
to use, modify and redistribute
the work in any way they wish,
with one exception.

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended **to guarantee your freedom to share and change free software**--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

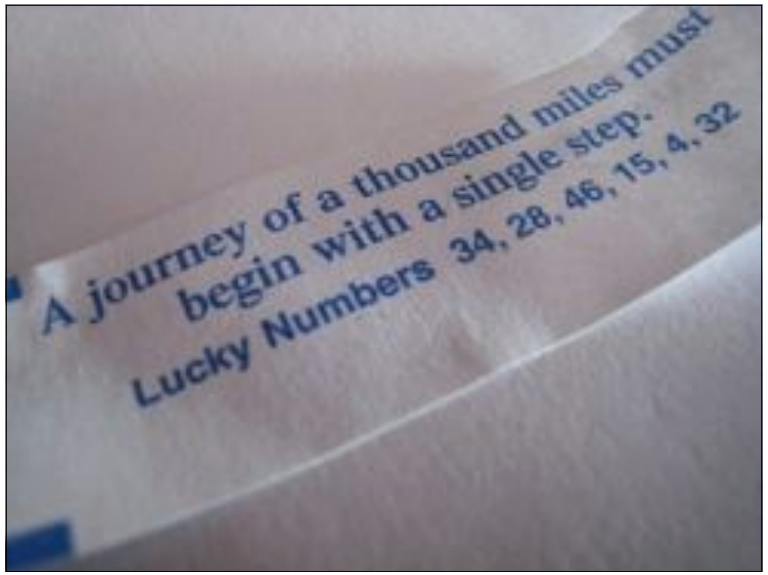
The license requires
that any redistribution
of the work or derived products
include the source code,
and be subject to the same license.

to guarantee your freedom to share and change free software

The legal language can get complex,
but the principles are hardly foreign.
Share and share alike.
Do unto others
as you would have them do unto you.

So.....

So, in 1984 Stallman quits his job at MIT
and starts working on GNU full time.
No visible means of support,
this is a labor of love.

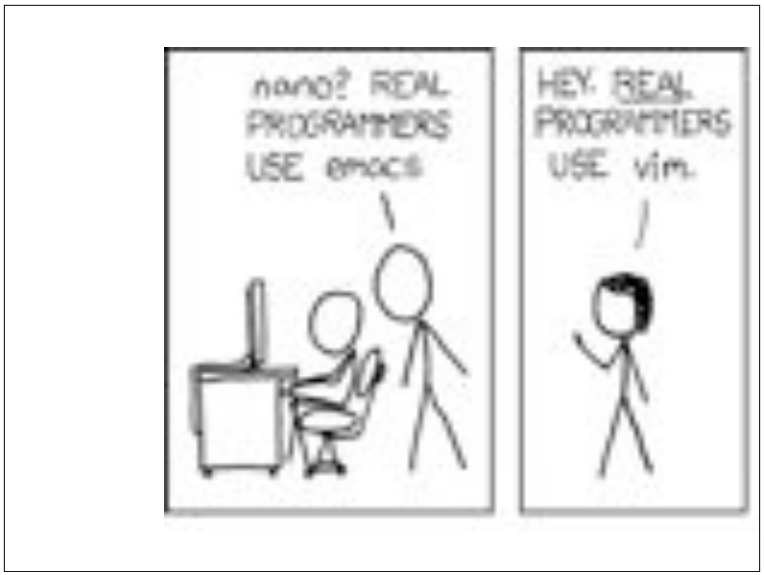


But where to start?
From a blank canvas, you want a
completely free software ecosystem,
what do you do first?

If you wanted to build a
100% all hand crafted house,

you would start by hand crafting your tools.
Stallman did the
same thing,
with GNU versions of
software development tools.

He starts by writing a text editor



(GNU Emacs),
so he can write his
free system using only free tools.

The Emacs editor proves so popular
(and internet access is still so rare)
that he is able to earn a
small living selling tape copies of the code
(distributed under copyleft of course).

Then he writes a compiler, GCC,
You can still find GCC
in every Linux distribution,
and also in Mac OSX
(though they recently switched to a different open source compiler).

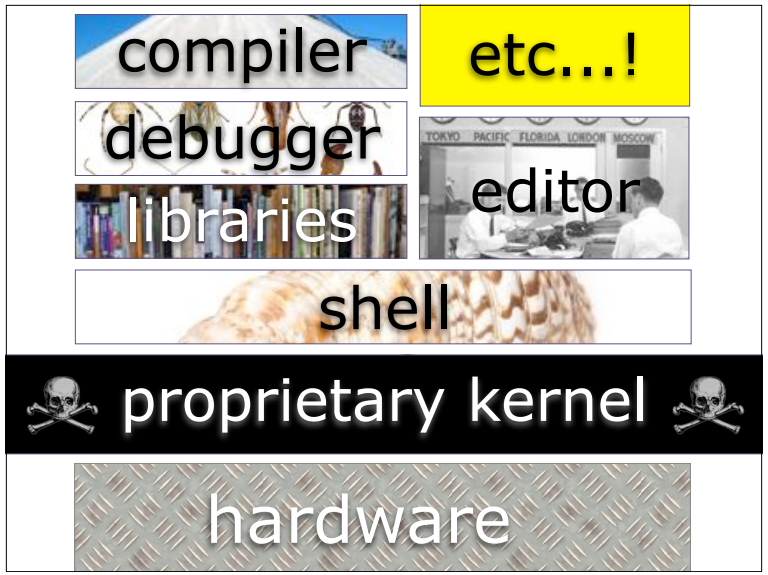
Stallman lives like a monk,
works like a demon,
attracts some followers and helpers,

FREE SOFTWARE FOUNDATION

who formalize the project
in a foundation.

By 1990 they have
most of the components
of an operating system.

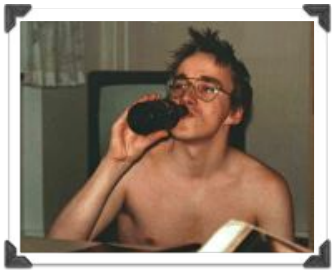
Most importantly, they have a full programming tool-chain:
<X> shells,
<X> compilers,
<X> debuggers,
<X> editors,
<X> core libraries,
<X> and so on.
All the things you need to write complex software.
<X> What they don't have, is a UNIX kernel, the piece of software that talks directly to the hardware.



At this point,
all their free tools are
still being run on proprietary UNIX!

A this point, I'd like to take a quick digression...

In 1991, a Finnish
computer science student



linus torvalds intel 386

named Linus Torvalds
buys a new computer,
<X> an Intel 386.

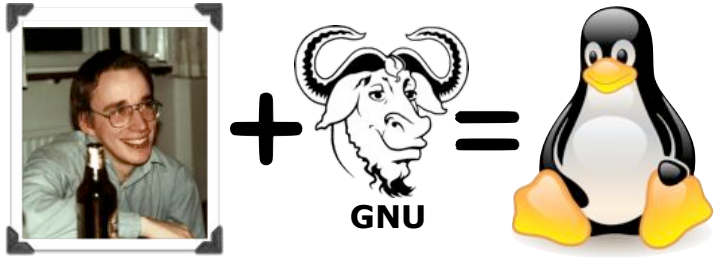
As a student at the university,
he has access to UNIX systems,
and he wants to run UNIX
on his 386 at home.

This is not possible.

Minix

The good implementations
for the 386 cost more
than the computer itself.
<X> The cheap implementation,
Minix, is quite limited.

So Linus writes his own kernel.



He uses Stallman's GNU tools
to write and compile it.
And in August of 1991
he posts the following
on an internet discussion list.

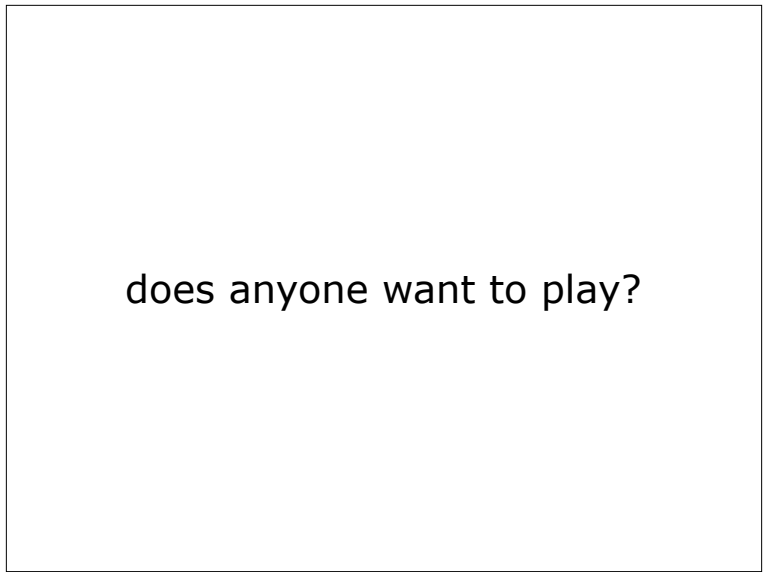
Hello everybody out there using minix -

I'm doing a (free) operating system
(just a hobby, won't be big and
professional like gnu) for 386(486)
AT clones. This has been brewing
since april, and is starting to get ready.
I'd like any feedback on things people
like/dislike in minix, as my OS resembles
it somewhat (same physical layout of the
file-system (due to practical reasons)
among other things).

I've currently ported bash(1.08) and

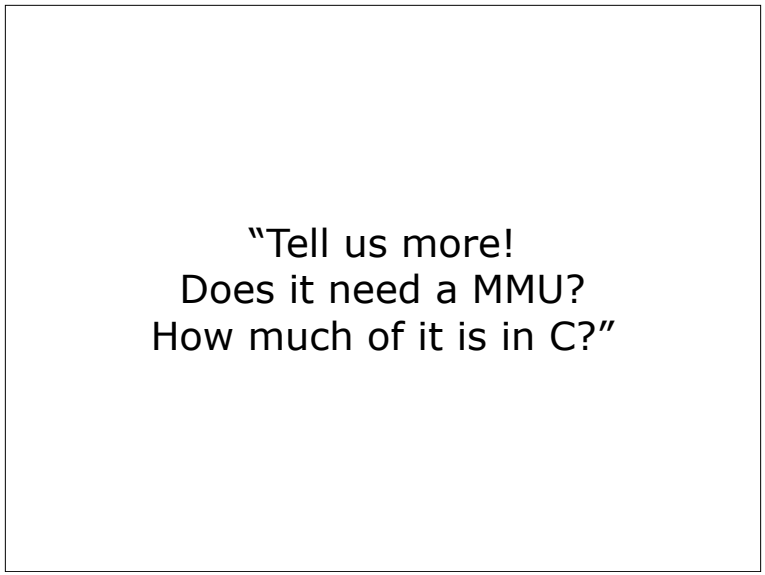
"Hello everybody out there using minix - I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, <X> as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things). I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)"

Underneath the technical language,
note the subtextual bits:
the humility (just a hobby...),
the interest in other people's ideas (what do you like/dislike in minix...).
the solicitation of suggestions.



The posting is an invitation.
Does anyone else
want to come out and play?
Does anyone?
They do.

Within 15 minutes, he has a reply.



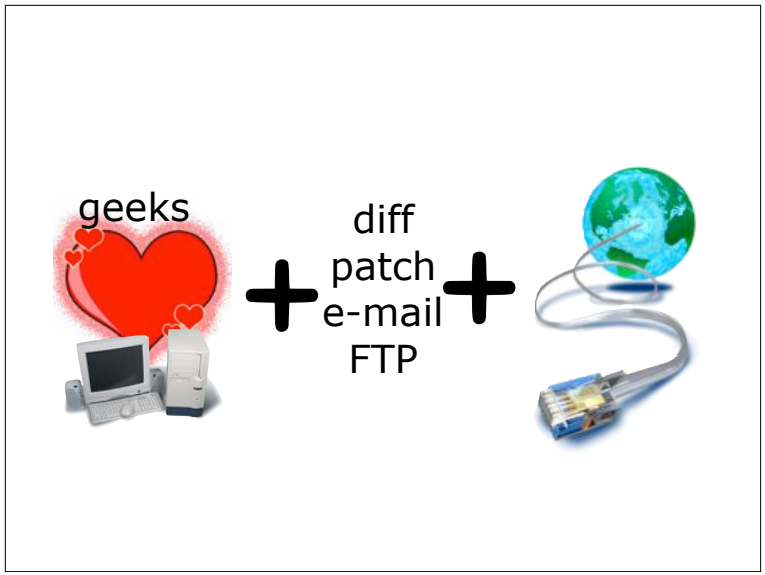
“Tell us more! Does it need a MMU?
(memory management unit)
How much of it is in C?’

Within 24 hours,

he has replies from Finland, Austria, Maryland, and England.

In a month the code is on a public FTP server. Within four months, it is so popular that an F.A.Q. document has been written to handle the common questions.

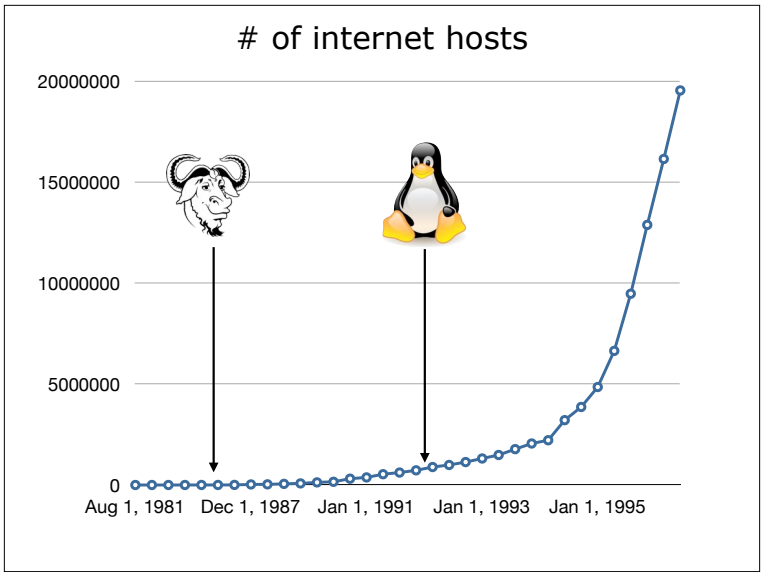
Linus Torvalds tapped a seam of enthusiasm just dying to express itself.



People who loved computers and computing and just wanted to play together. And through the medium of the internet, using only the simplest computing tools (diff, patch, ftp, e-mail), he built a community of thousands of contributors, and

together they built a
usable operating system.

Something important changed
between the time Stallman
started the GNU project and
when Torvalds released Linux.



The values of collaboration were the same, but
the opportunity to exercise those values was
greater, via the internet.

<X> When Stallman started GNU in 1984,
there were 1000 hosts on the internet.
<X> When Torvalds started Linux in 1991,
there were over 400,000.
And the pool of potential collaborators was in the
middle of
a huge expansion.

Permit me one more
short digression on the digression
to talk about

Star Wars. Star Wars is awesome.
In particular, let's look at
<X> a web site called Star Wars uncut.

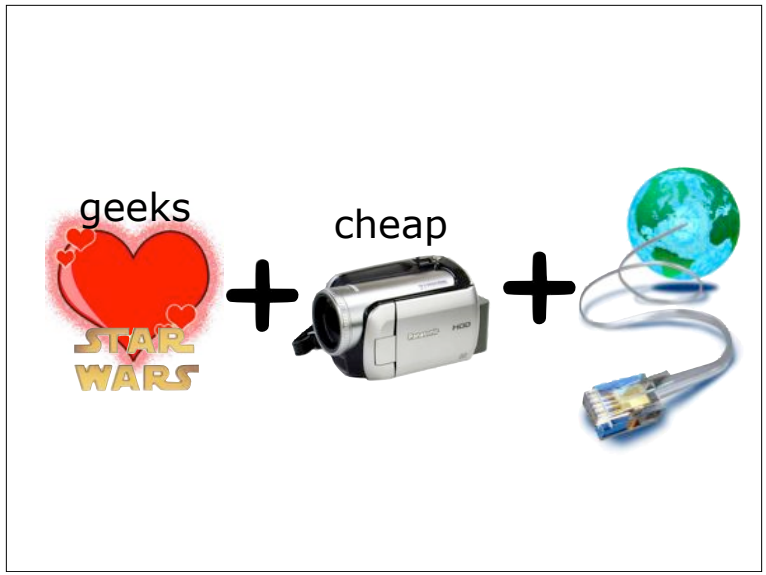
Star Wars Uncut has
taken the original movie
and chopped
it into 473 fifteen second scenes.

Each scene is then
separately claimed
and re-enacted by site members,
and uploaded.
The result looks like this.

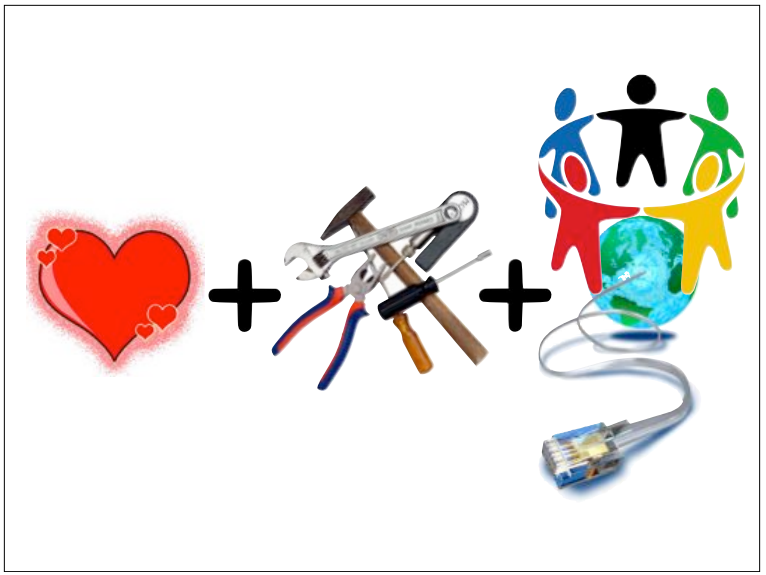
frivolous

but why now?

Seems pretty frivolous, right,
but break it down.
How is this (frivolous)
collaboration possible?
And, why is it only happening now,
<X> not 10 years ago?
There were just as many
Star Wars nerds
10 years ago as there are now.



First, this activity requires
easy access to video recording
and editing tools,
and until recently cameras and
video editors were very expensive.
<X> And it requires enough bandwidth
to download and upload video,
and until recently people
didn't have that kind of
bandwidth in their homes.
<X> And finally it requires Star Wars geeks.
Generalizing, then...



To build a large collaborative product, you need
<X> tools freely (or very cheaply) available and
you need
<X> sufficient connectivity between participants.
Combine that basic infrastructure,
with community, collaboration
and <X> love for the subject matter,
and magic happens.

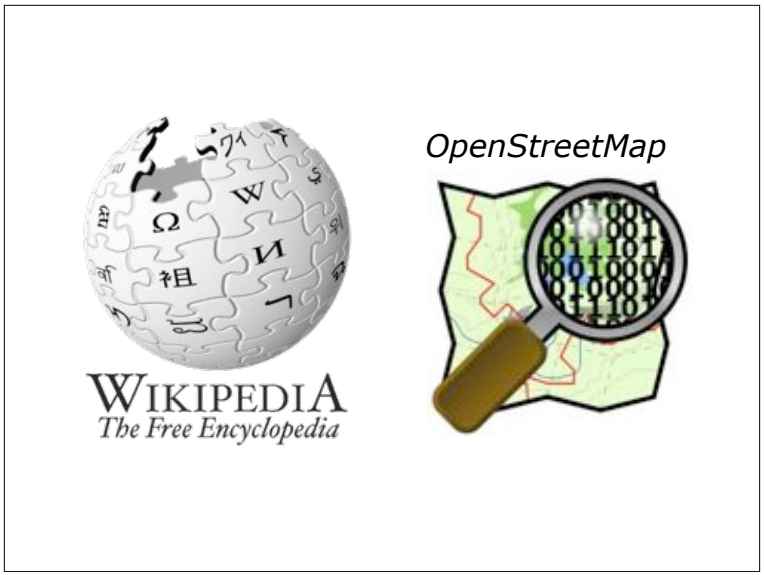
There are many, many more examples of this kind
of
group collaboration,

“commons-based
peer production”

the academics call these collaborations
"commons-based peer production".
It is *not* a niche phenomenon.

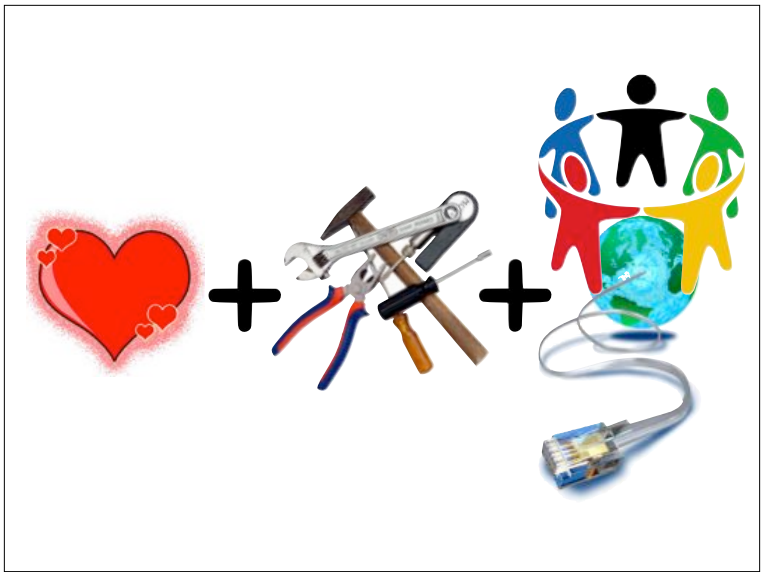
open source is
commons-based
peer production

Open source software in general
and the Linux project in particular
is one of the earliest examples of
internet-mediated
commons-based peer production.

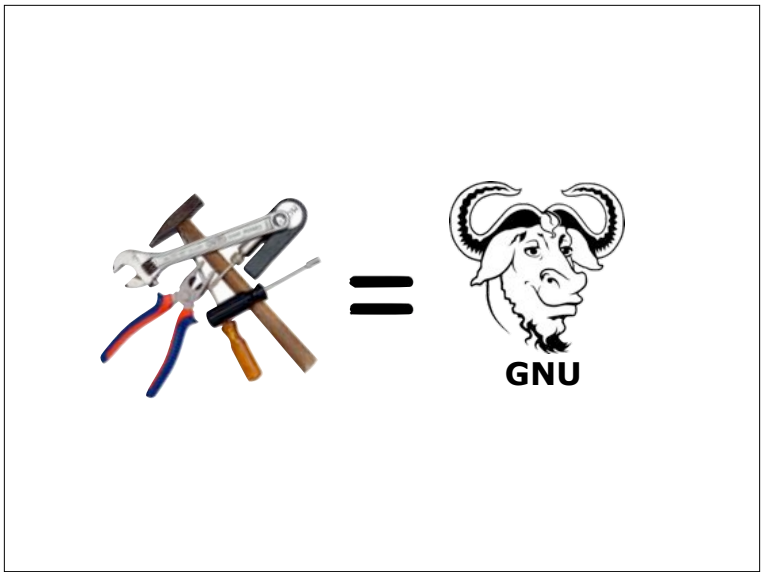


More recent widely known examples are
Wikipedia and
<X> Open Street Map.

This is *not* a niche phenomenon.
Complex structures of knowledge can be built
by distributed communities using free tools,
held together by a shared interest (emotional or financial)
in the product being created.
The proof is already here,
in our software, in our encyclopedia, in our maps.

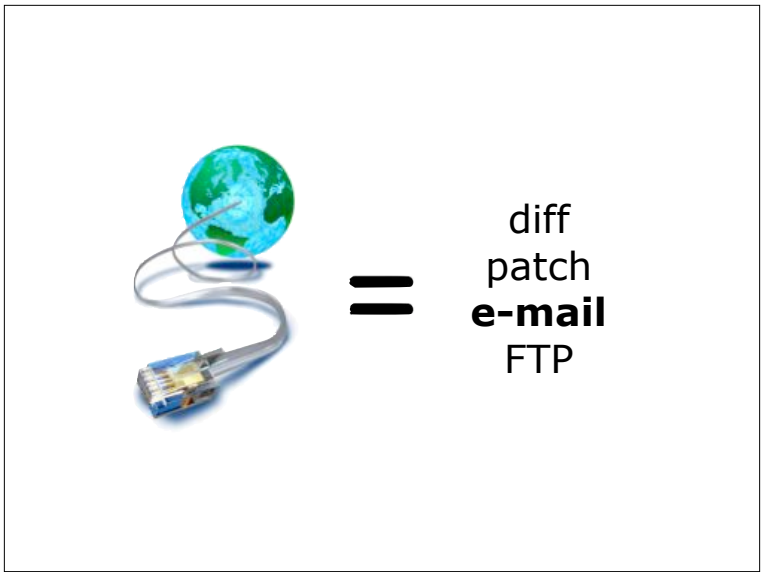


The development of Linux fits
the commons-based peer production pattern:
The free access to tools was provided by



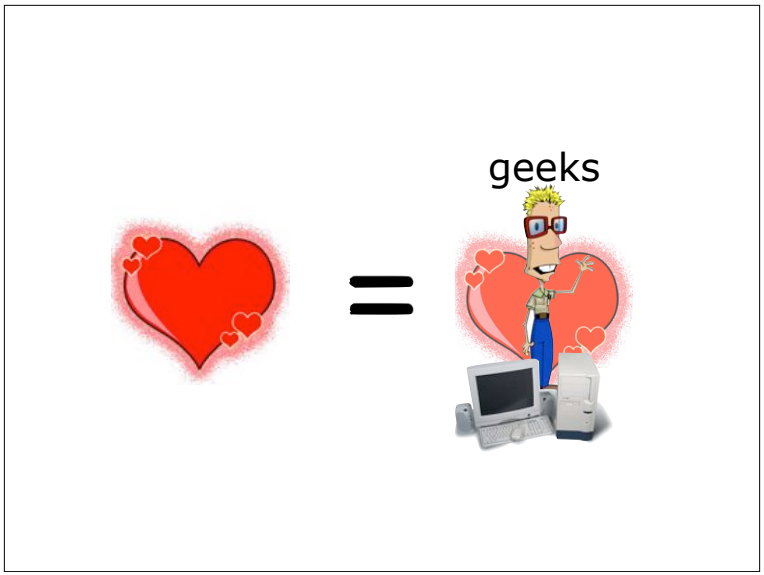
the existing GNU project components.
Editor, compiler, lexer, libraries,
were all available equally to
all participants.

The medium of communication was just e-mail.



the work they were sending around was source code,
snippets of text, no problem
even for the dial-up
internet connections of the
early 1990s.

And why do open source programmers do it?
What is the core motivation.
It isn't money.



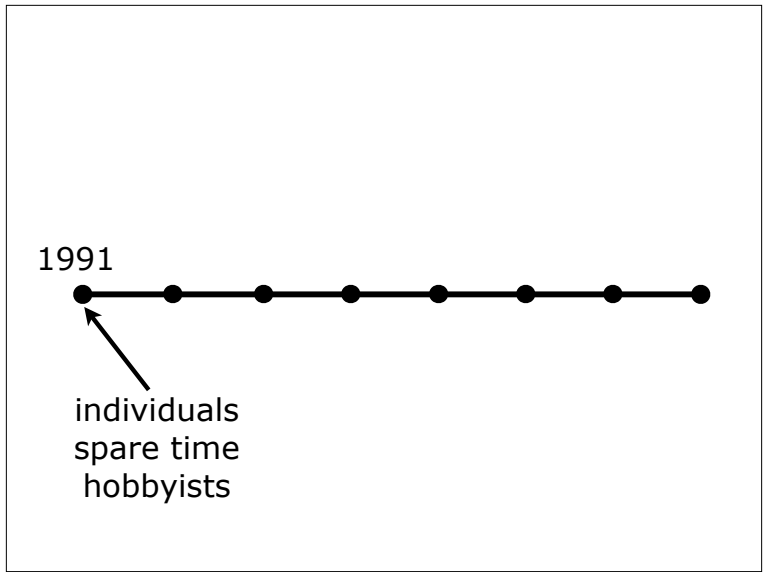
Fundamentally they
code because they love it.

It's the same reason

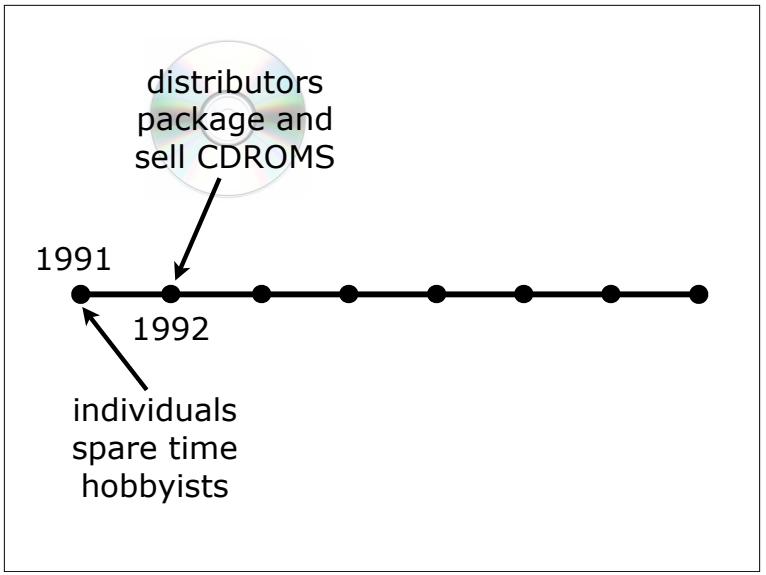
Star Wars geeks re-shoot 35 year old films,
<X> why food geeks post restaurant reviews,
<X> why car geeks re-build '68 Cameros.
It's an avocation. It's love.

At least, it starts that way.

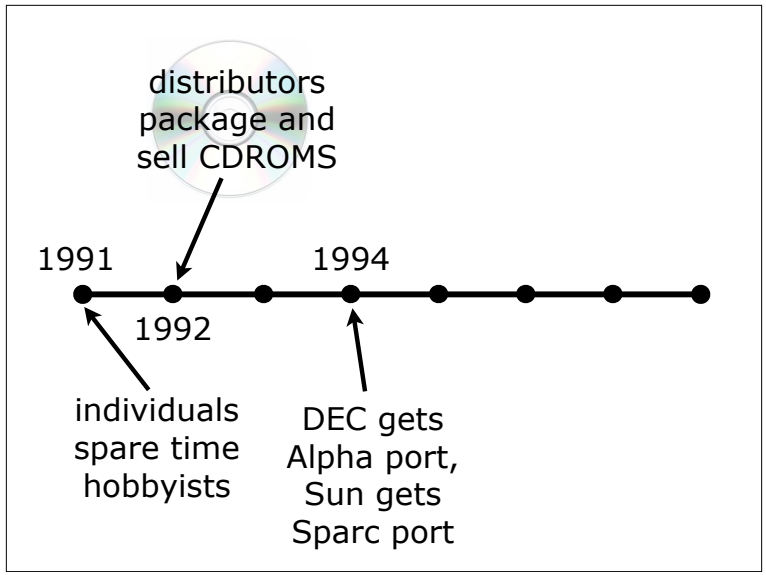
But open source software has a wider economic utility than kitsch films, restaurant reviews or vintage muscle cars.
And as open source projects have expanded, they have at each stage become more and more integrated into the wider economy.



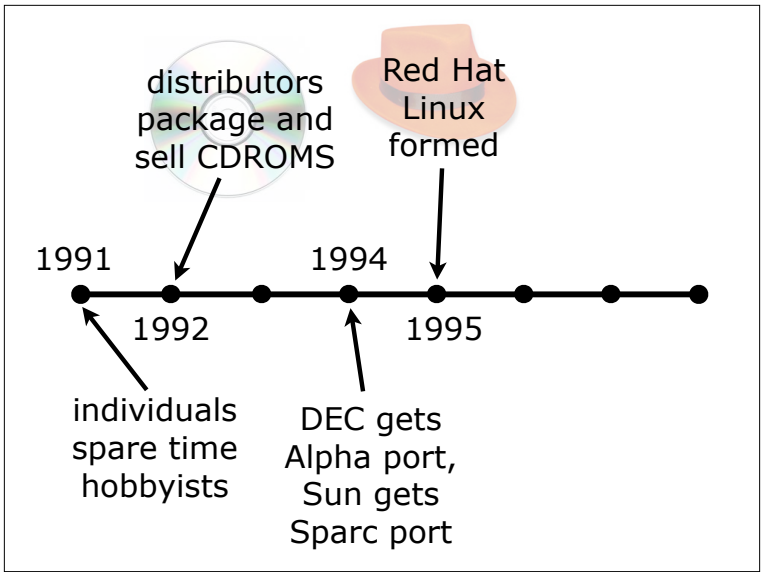
Linux is a good example.
Start with Linus and the early group of enthusiasts in 1991.
These are individuals working in their spare time.
They are doing it for love.



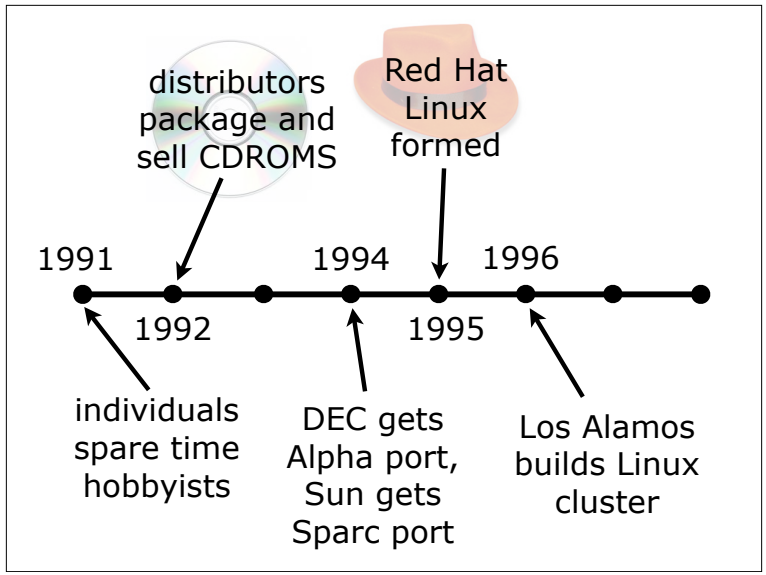
By 1992, you get "distributors", packaging up the Linux kernel
with collections of GNU and other tools to form full working operating systems.
First they do it for love, helping other Linux lovers,
but soon they are covering their cost and time, selling CDROMs for $50.
So programmers are earning livings with small Linux businesses within a couple years of the project start.



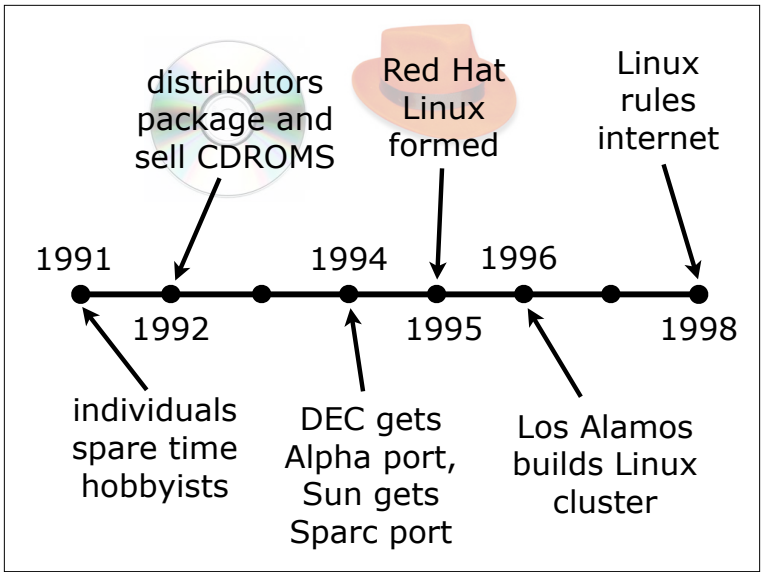
In 1994, DEC sends Linus a "free" Alpha workstation in the hopes he will port Linux to the Alpha chip.
He does.
Simultaneously David Miller ports Linux to the Sun Sparc processor.
Linux is now competing with "real" UNIX on corporate "big iron".
Over the next couple years, the makers of these machines start to hire Linux programmers of their own.



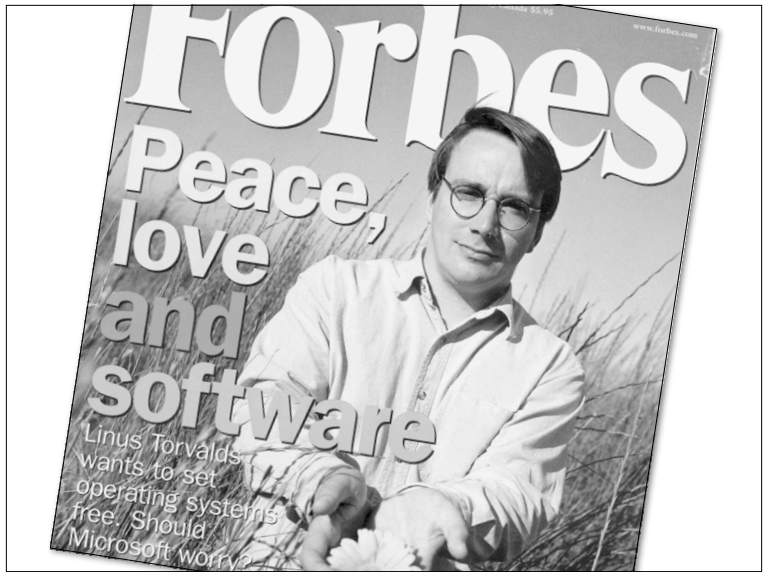
In 1995, Red Hat Linux is formed, a company which will eventually grow to an $8B Linux support enterprise.



In 1996, Los Alamos National Laboratory builds the first Linux cluster for simulating atomic shock waves.
It costs 10% of a comparable supercomputer, and on start-up becomes the 315th most powerful supercomputer in the world.



By 1998, the explosion of the internet into general public use is underpinned by thousands of commodity servers running Linux as their operating system.



Microsoft is drafting strategy memos about how to counter Linux, and Linus Torvalds is featured on the front page of Forbes magazine.

Linux is no longer a hobbyist activity.
It is deeply embedded in the economy at multiple levels.
This is in 1998, just **seven years** after that first newsgroup post.

Fast forward to the present.
<X> The NSA employs Linux programmers to make their systems secure.
<X> NASA employs Linux programmers to run it on their space mission hardware.
<X> Google employs Linux programmers to optimize their massive compute clusters.
<X> Oracle employs Linux programmers to support their Oracle-optimized Linux.
<X> IBM employs Linux programers to ensure it runs on their SystemZ mainframes.
<X> Microsoft employs Linux programmers to add kernel support for Windows virtualization.
<X> And so on, and so on, and so on,

So, here's a question I get asked a lot:

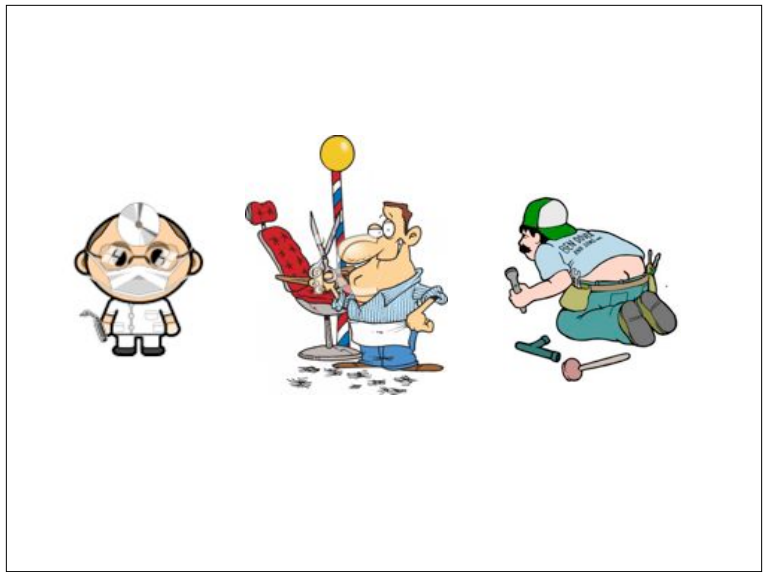
"how do you make a living writing free software?"

um, how do you make a living writing free software?

Referring back to the previous slide...

Hopefully it would be obvious.

I make my living the same way



my dentist, my barber and my plumber
make their livings.
I sell my very specialized expert services
in open source spatial database programming
to people who need those services.
And in a globalized, internet connected world,
there are plenty of people who use my software,
and want my services.
But enough about me. What about you?

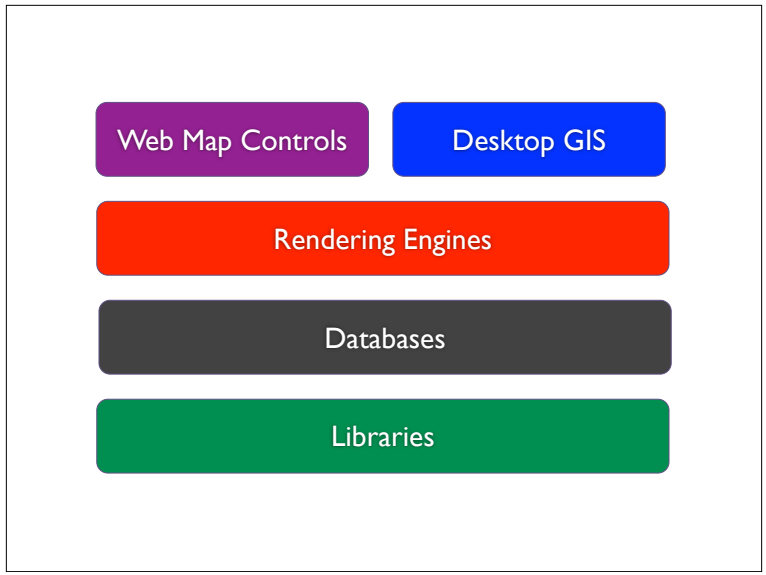
Yes you.
Should you use my software? Should you use open source?
Well, hopefully you already do! But in case you don't, in case this world
of FOSS4G is new to you, here's a little map of the territory to make sense of
the many many acronyms and project names you are going to hear over the next few days.

To say more than 20 words about any but the most popular projects would take a couple hours, so I will be...
brief.

Open source geospatial software is ...
like a sandwich,
or no, perhaps,

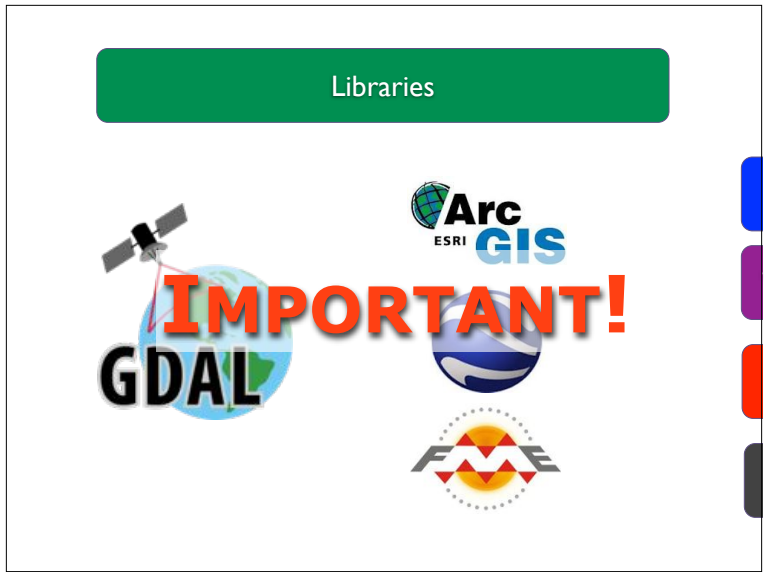
open source geospatial software
is like a geological stratification,
or maybe

open source geospatial software is like a layer cake,
or, but, really, maybe it's like



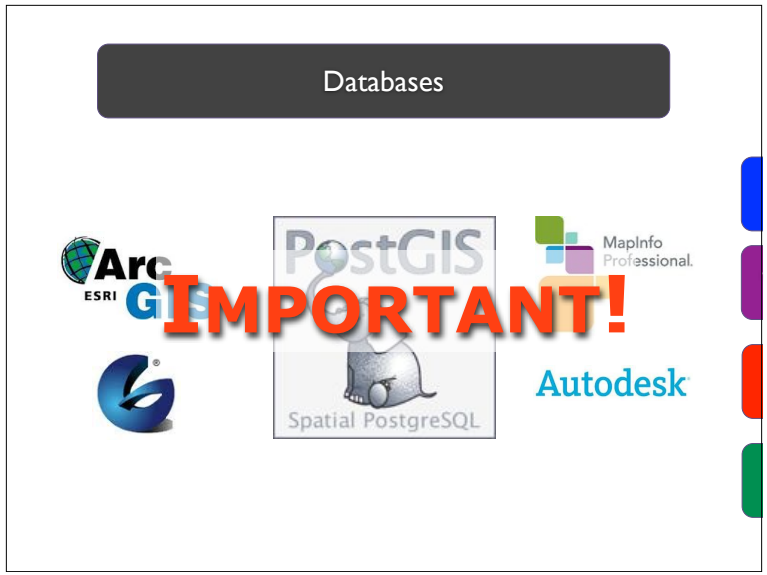
an architecture diagram...
ok, maybe, it was always going to be the architecture diagram...
nerd talk is nerd talk
Underneath the servers and the utilities is a substrate of

libraries, shared code that handles common problems.
Format support is a common problem
In C and C++, <X> the GDAL library allows access to multiple image and vector formats. You can find it under open source software like MapServer, MapNik and QGIS.

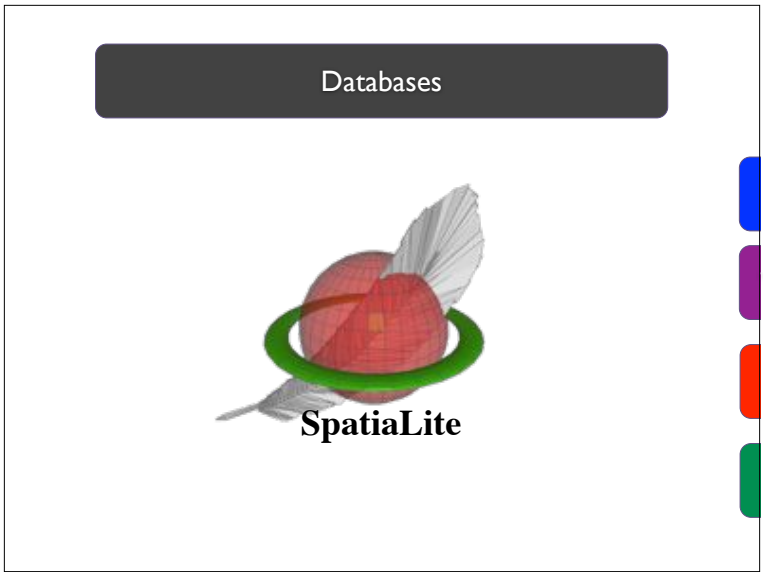


You can now also find it inside ArcGIS, FME and Google Earth.
<X> **This is important**. Open source is not an either/or proposition, it's not all-or-none.
You can mix and match components and even, in the case of libraries, completely embed open source functionality in your solutions.

In Java, the GeoTools library performs the same function as GDAL, abstracting away file and database format differences to allow easier application development.



At the next level is databases.
My project, PostGIS, is the most widely used open source spatial database,
and is supported not only by all the open source programs, but also by all the proprietary ones too.
<X> ArcGIS, MapInfo, GeoMedia, and MapGuide can all use PostGIS as a data store.
<X> **This is important**. Open source allows and encourages mix-and-match architectures, where some components are proprietary and others are open.



For mobile devices and programs that require only single-user access to data, the SpatiaLite embedded database is very popular, and is showing up in all kinds of places.

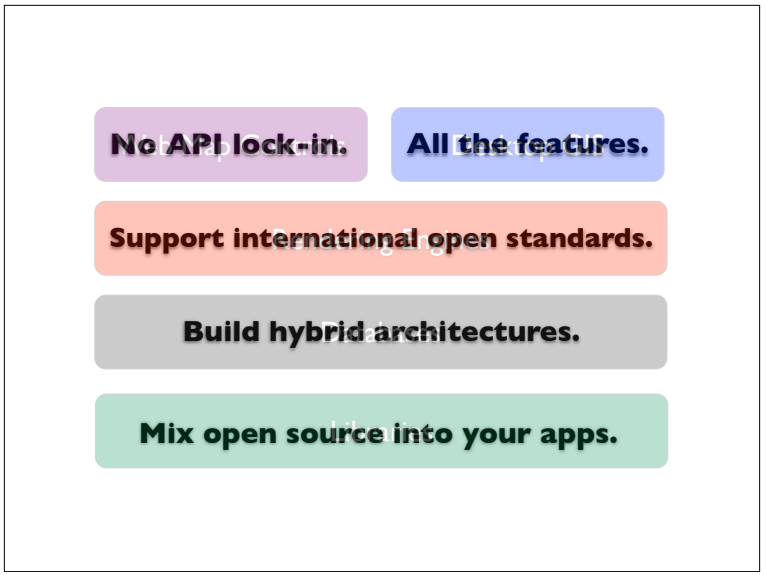
Up another level, but still on the server side, are rendering engines,
taking raw data and producing cartographic output over the web.

<X> MapServer, a C process that runs as a CGI, FastCGI or even an Apache module.
<X> GeoServer, a Java enterprise application that runs in a J2EE container.
<X> MapNik, originally a C++ renderer only, but now also runnable as web service.
These tools all produce beautiful cartographic output, on the fly or pre-generated into files,
<X> and
they all, support, Open Geospatial Consortium standards for web services.

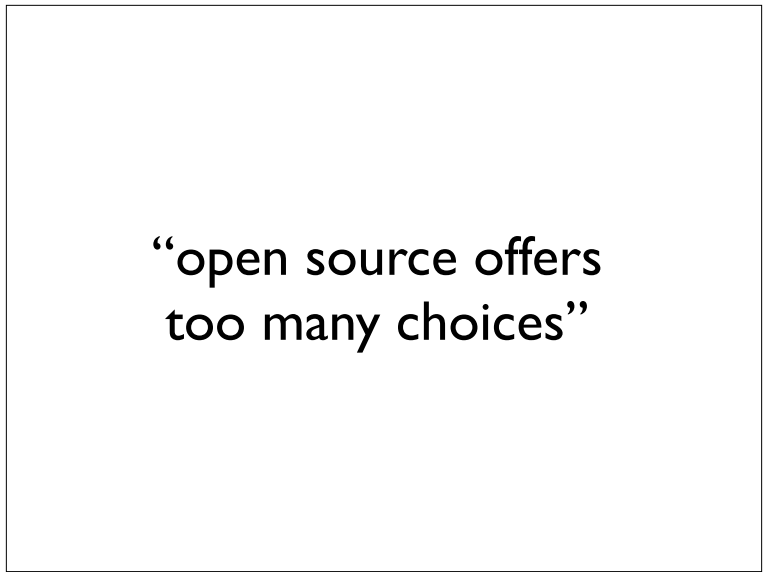
Up another level, and now on the web browser side, are web map controls, allowing zooming, panning, selection, markers, overlays and more.

<X> OpenLayers, the original open source JavaScript map, with the widest support for legacy browsers and the most features.
<X> Leaflet, a new contender, uses the latest in HTML and JavaScript technology for the small and simple experience on the latest browsers.
<X> This is **important.** Open source web maps are not tied to API keys or API licence agreements.

Moving sideways to the desktop,
<X> QGIS, a C++ viewer / analysis application, much like the old ArcView3, but with more analytical capability and scripting in Python.
<X> GRASS, the original open source GIS, dating back to the 1980s, with old school command-line goodness and a stockpile of hundreds of analytical routines.
<X> **Here's the important bit.** These tools have the analytical features and data management features to support real business.



Remember the important bits:
<X> Open source code can be mixed into applications via libraries.
<X> Open source applications interoperate with proprietary ones, so you can build hybrid architectures.
<X> Open source geospatial servers have very strong support for OGC standards.
<X> Open source web components don't force you to use API keys or accept API licenses.
<X> Open source has the features to support real operations
.
When I give the full version of the open source survey talk,
which takes a couple hours,
and I cover all these options in detail, not just in logos
exhausted people come up to me afterwards and say



"open source offers too many choices"

"it's easier with just one vendor".

Which is odd, because we deal with lots of choice in all the other markets we navigate every day.

There's lots of kinds of cars,
there's lots of kinds of blue jeans,
there's lots of kinds of coffee.

And we have a good idea of what a market with just one vendor looks like.

We actually have laws against it.

Proprietary software has a dirty little secret, and it is a secret that lives in plain sight.

proprietary licensing
creates de facto
monopolies

Even in otherwise competitive markets,
the effect of proprietary licensing is to create an instant
de facto
monopoly.

one
one
one

How many companies provide support for your proprietary software: <X> one.
How many companies provide upgrades: <X> one.
How many companies provide enhancements: <X> one.
It is all about market power.

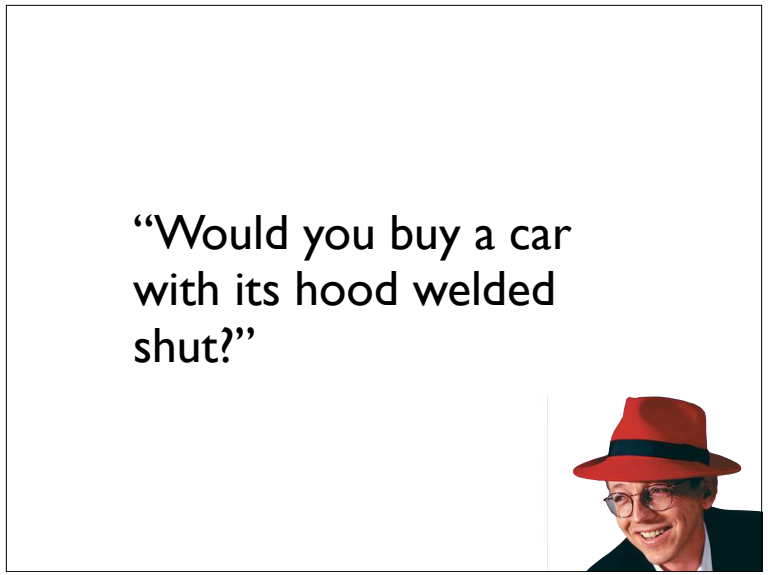
do you have
market power?

Open source vests the market power in the software user, not the vendor.

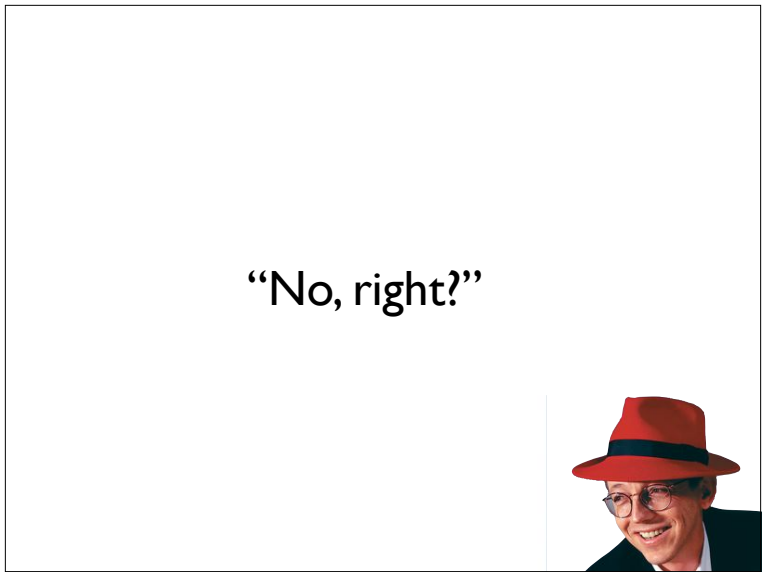
As a manager, you probably don't care about tinkering with the internals of your software source code,
but you SHOULD care about
holding on to your market power as a customer.

Bob Young
Founder
Red Hat Linux

Bob Young, the founder of Red Hat Linux, asks this question of customers:



Would you buy a car with its hood welded shut?



No, right? So ask the follow-up question:

“What do you know about modern internal combustion engines?”

What do you know about modern internal-combustion engines?

“Not much.”

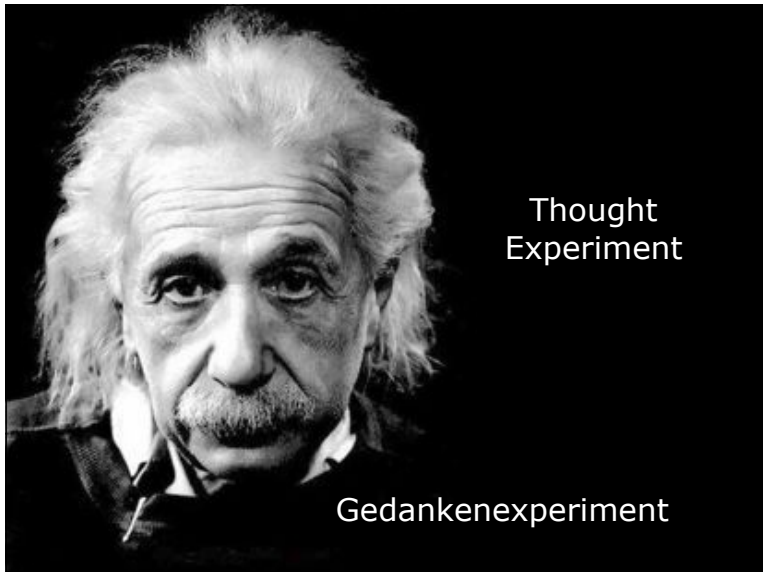
And the answer for most of us is "not much".

"We demand the ability to open the hood of our cars because it gives us, the consumer, control over the product we have bought, and takes it away from the vendor."

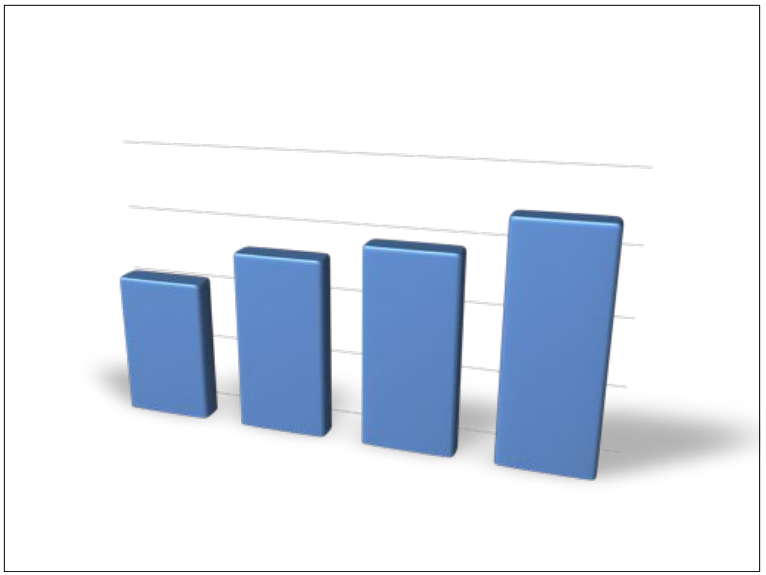
"We demand the ability to open the hood of our cars because it gives us,
the consumer, control over the product we've bought and takes it away from the vendor. "

But Paul, maybe…
you're wrong?

OK, could I be wrong about all this stuff?
First of all, don't be foolish.



But second of all, don't trust me, trust your own instincts.
Albert Einstein worked out his theories of relativity with "thought experiments", "gedankenexperiments". Call on your inner Einstein, and
perform this little thought experiment.



Think about how much open source use there is in your organization now. It might be very little. <X>
That's OK.
Now, think about how much open source use there will be in your organization in two years time. Will it be more or less than now? <X>
Now, think about two years after that, more or less? <X>
And two years after that, more or less? <X>
The trend is not necessarily fast, in some organizations it can be positively glacial, but the movement is all in one direction.

So, enjoy the next couple days,
and get a glimpse of the future,
because the future is here, it's open source,
and there's only going to be more next year.
Welcome to FOSS4G everyone,
let's have a great conference.