# FOSDEM 2024 – a different road out of the jungle

## Intro

#### 4th FOSDEM talk. I am amazed. Thanks for inviting me back!

#### New job, new heresies

#### Wirth

##### Since I submitted this talk, something sad happened, even if it was inevitable. A giant of 20th century software design, Niklaus Wirth, died.

##### Because of him, I decided to change the way I introduced this talk.

###### Wirth is famous for one thing, but arguably, it wasn’t the biggest aspect of his career. For instance, the computers Unix was written on used teletype as their terminals. These were, very roughly, printers with a typewriter keyboard on them. Teletypes are why Linux console device files are called `tty`.

###### Now typewriters have a shift key, and often not much else. Terminals have lots of modifier keys: “control” and “alternate” (or “option” or “meta”) and super (or “command” or “Windows”). The bits in memory which represent if one of these are pressed down are called “Bucky bits”…

###### Because Niklaus Wirth studied in, and taught in, and loved, California, and his nickname there was “Bucky”.

###### Bucky Wirth didn’t just invent Pascal. Pascal grew out of a proposal to improve Algol. The Algol committee turned it down and picked someone else’s, more complicated, idea instead. That became Algol 68 and killed the language forever.

###### As a result we got other languages that were offspring of Algol, which took its ideas and changed them. Such as BCPL, which was turned into B, which became C, and C++ and everything built from them.

###### Because the Algol guys didn’t like Bucky Wirth’s simple, clean, proposal.

###### Wirth went on to create Pascal. This was a big hit. A large part of the Apple Lisa operating system was implemented in Pascal, and that went on to profoundly influence the Mac.

###### A Unix was implemented in a Pascal dialect, too. It’s called TUNIS, the Toronto University System, and it was written in a Pascal derivative called Concurrent Euclid.

###### Pascal became Turbo Pascal became Borland Delphi and drove the success of Microsoft Windows 3. It was a big hit.

###### Wirth ignored all that and went on to write a successor, called Modula. Then he threw that away and wrote a successor with built-in concurrency called Modula-2.

#### Leanness

##### Late in Wirth’s career, he became a passionate advocate of small software. He wrote a wonderful short article about this. It’s called *A Plea For Lean Software*, and it’s only a few pages long. Google it and read it. It’s worth it. It won’t take long.

###### As a proof of concept of the validity of the concept, Wirth moved on from Modula-2 and wrote his masterpiece, Oberon.

###### It’s a tiny Pascal-like language, with concurrency primitives. And a compiler, and an IDE, which is also a tiling terminal window manager and an OS in its own right.

###### For my obituary on Wirth, I downloaded the source code of the core of Project Oberon and ran a line count.

###### It comes out a bit over 4000 lines. Specifically, some 4,623 lines of code, in 262kB of text.

###### That’s a self-hosting bare metal OS. It is unbelievably tiny.

##### My previous talk suggested that something like Oberon was worth investigating as a possible next generation OS to Linux. I now realise that this was the wrong crowd for that.

#### Where next?

##### In my first talk I said there were dozens and dozens of OSes which had some degree of success that are more or less totally unlike Unix, or DOS, or Windows, or macOS.

##### In my second, I talked about progress going backwards. It’s normal.

##### In my third, I talked about throwing away 1970s OS designs and starting over, perhaps using tools like Oberon, Smalltalk, and maybe some of the more readable Lisp-like languages such as Dylan.

##### I think that was too radical. I’ve certainly not heard of any results.

##### So I want to suggest something much smaller and simpler and closer to home. Less radical, although still quite radical.

## History

### FOSDEM is about FOSS

#### FOSS is about Unix

#### Unix now is about Linux

#### What is Unix?

##### Not AT&T

###### 1993. Ask audience: how many remember 1993?

##### In 1993, Novell bought Unix from AT&T, kept the code and donated the trademark to the Open Group.

##### Since 1993, the word “Unix” has meant “passes Open Group testing”, which is broadly what POSIX used to mean.

##### The original codebase is effectively gone. Xinuos still sells things based on it, but their main product is based on FreeBSD.

##### But alongside macOS on both x86 and Arm64, and a handful of other OSes, such as HP/UX and IBM AIX – several Linux distros have passed. All of them Chinese CentOS Linux derivatives.

##### But what that means is that **Linux is a Unix now**

##### It’s not Unix-like any more. It is a Unix.

#### Take a step back

##### The family tree shows lots of branches and lines

###### Unix 3, Unix 4, System 5, and lots and lots of derivatives of them, mostly proprietary

###### All the BSDs

###### And of course Linux, with zero shared code, but a shared design.

##### Take another step back: **they are all one branch**

###### Unix now means compatibility. If it passes, it is a Unix

###### POSIX became Unix

###### But that merely means passing tests… you can do that with a shim, a compatibility layer

###### The compatibility definition means that IBM z/OS and OpenVMS and Windows \*are Unixes\*

###### But that can’t be right, right?

##### We most of us know what a Unix looks like, right?

###### One filesystem, rooted at root, forward slash

###### The shell.

Doesn’t matter which shell.

Stuff like ls, cat, echo, mkdir, rmdir, touch

###### Case sensitive filesystem

###### Plain text files, pipes

##### There can be exceptions

###### macOS is a Unix. It passes the tests, Apple pays for the certification.

###### But… no X11. It’s an extra.

###### And it’s not case sensitive!

###### Users can’t see the real tree, and /etc is relatively empty

###### But it’s still a Unix

##### If we take this list of general characteristics…

###### … programmed mainly in C or something C-like

###### Looks like a Unix \*and nothing else\*, meaning that there’s no other native layer underneath

##### … then the family is bigger

##### So how to categorise it?

###### Generations kind of work

###### 10 to 12 generations of research Unix in AT&T… call that one

###### BSD forked off around 5/6 and again around 7/8

###### SunOS from BSD

###### Sys V from Unix 7 (and Solaris from that)

##### You know the chart, or ones like it. It’s big.

##### But if you consider that one codebase… Call that one. One blob on the chart.

##### That leaves a few outliers. Linux is one. Minix is another.

##### Microkernels!

###### CMU Mach led to a whole bunch including DEC Tru64, but they’re all dead and gone, and the only one that isn’t a historical curiosity is macOS, iOS, iPadOS etc.

###### Mach was the first attempt at a microkernel Unix, and it worked, and for a while it was everywhere

###### e.g. led to the GNU Hurd, but it is a small niche

###### Minix 1 and 2 were largely monolithic, but Minix 3 is a microkernel. There are others: Mach led to L4 and others, and work’s still going on there.

###### QNX is a commercial one and it’s in billions of embedded devices, but the only time you might have played with it was Blackberry 10

###### But microkernels are **hard**

Tiny kernel, not so hard

Lots of userspace processes doing stuff, not so hard

AmigaOS was doing this in the mid-1980s!

###### The problem is getting them to communicate with each other

AmigaOS ran on machines with no memory management hardware. Everything was in one address space. It’s easy to communicate with another process when you can read and write its memory

###### The idea of a true microkernel is that the kernel runs in ring 0, in supervisor mode. It schedules processes, it allocates memory, and that’s about it

Everything else is implemented in “servers” in user space

Getting them to talk to each other, **quickly**, is the hard problem

###### For instance, macOS uses Mach, but to provide Unix-compatible APIs so it looks like a Unix, it took a big chunk of the BSD kernel, called it a “Unix server”. Mach still manages the processes, not the Unix server, but it’s still big and complicated

###### And it still wasn’t that fast, so, NeXT moved it inside the Mach kernel. It’s in kernel space too, running in the same memory space as the not-so-micro-any-more kernel

#####  So let’s say that there are two lines now.

###### Monolithic kernels, which covers almost all the old proprietary commercial Unixes, and all the BSDs. And Linux.

###### Microkernels.

We can very loosely group these into two families.

1. Mach and Apple’s OSes, and QNX: commercial, and partly closed-source
2. Minix 3, the GNU Hurd, L4, seL4, NeptuneOS. Mostly open source, some use in embedded control systems – including every modern Intel CPU – but little visible mainstream use

##### Is that it? That’s all most generous histories show.

##### No, it isn’t. There is a third line of descent.

##### Work didn’t end in 1979 with Unix 7, even if that’s where the family trees end.

#### Turn it up to 11

##### Unix after release 7 is often called Research Unix, and digging into it has yielded more people telling me “that’s WRONG” than useful info.

##### But…

###### 1985: 8th edition

###### 1986: 9th edition

###### 1989: 10th edition

##### What came next is where it gets interesting. By 1992 Unix was big commercial news, of course, FreeBSD and NetBSD were happening, and Linux was gathering attention

##### Bell Labs did something else

#### Rewind a moment

##### Unix is about files, plain text, the shell, right?

##### Graphics isn’t part of the kernel. MacOS and Linux are both Unixes, as is BSD, but they have radically different GUI layers.

##### Why? Because Unix was a minicomputer OS. A host, dumb serial terminals, with no graphics and no networking. It was the 1970s.

##### By the 1980s there were very expensive workstations: personal single-user minicomputers

##### I discussed this in 2020 in the “Generation Gaps” talk

##### Things like inexpensive 32-bit computers with on-board graphics, reasonably fast expansion buses, and so reasonably fast networking as a fairly cheap option, started to be mainstream by the late 1980s.

##### Unix 10th edition appeared in 1989 when this stuff was proliferating everywhere.

##### But stuff like networking is not in the Unix design.

##### It **has** networking now, sure. But how does a Unix machine talk to another Unix machine? That’s not part of the design. It needs helper stuff like NFS. It works, but it’s a bolted on extra that appeared years later.

##### The Wayland fans like to talk about how they are modernising the Linux graphics stack. But Linux is a Unix, and in Unix, everything’s a file, right?

##### So, go on then, Wayland fans, tell me: where in the filesystem can I find the text files describing a window on the screen? Everything is a file, so, what file holds the coordinates of the window, its place in the Z-order, its colour depth, its contents?

##### You can’t?

##### Are you **sure** this is a Unix system then?

#### Unix 10th edition started to address some of this

##### But it was too little too late. The industry had taken v3, and v4, and 5 and 6 and 7, and commercialised them and they were mutating and metastising across the world.

##### So, Dennis Ritchie and Ken Thompson did what Niklaus Wirth did. They ignored what the industry was doing, went back to their original ideas, and kept working on refining them.

### The next step of history that doesn’t get talked about much

#### What Dennis and Ken did next

##### The successor to Research Unix is Plan 9 from Bell Labs.

##### Plan 9 took the concepts that led to Unix, and rethought them for the 1990s.

##### I want to talk a little bit about the significance of Plan 9, what it got right, what it got wrong, why it failed to go anywhere, and a way to fix that today.

#### The good

##### Plan 9 is Unix and C, but reconsidered for a world of networked graphical workstations

##### It takes many of the trendy ideas of late-1980s computing, both of academic theories and of industry, and it re-interprets them through the jaded eyes of two great gurus who saw their previous good idea misunderstood

##### Plan 9 puts networking right into the heart of the design. It is not so much an OS for a workstation, but for a cluster of computers, some graphical desktops and some shared servers.

##### Plan 9 takes the idea of “everything is a file”, which in Unix is an empty marketing slogan, and it makes it real. Your windows are directories and their contents are in files. Other Plan 9 machines on the network are visible in your filesystem – subject to access permissions, of course – and you can see some of their filesystem right inside yours.

##### Because everything is just files, that means that to display a window on another machine, you just need to make a folder and put some files in it. You can start programs on other computers, and display the results on yours, and all the Unixy stuff about telnet and rsh and ssh and X forwarding and so on just… goes away.

##### It makes X.11 look very overcomplicated, and it makes Wayland look like it was invented by Microsoft.

##### Because everything goes through the filesystem, it eliminates one of the biggest complex problems of microkernel designs: inter-process communications. It’s not that Plan 9 is a microkernel; it’s more that in some ways it makes the defining features of microkernels somewhat irrelevant.

#### An aside, in case this all sounds a bit too theoretical

##### One of the problems if everything is in the filesystem, of course, is where it goes.

##### Back in 2011, I predicted that the next big thing in Linux virtualisation would be containers. That was about the same time that Docker launched.

##### Containers are a sort of chroot on steroids: all the file paths in a given process start at a new root, and since in Unix almost everything is relative to the root directory, changing that isolates that process (and any children) from the rest of the OS.

##### Nowadays there are several forms of containerisation on Linux, but one of them uses the cgroups feature in the Linux kernel, which separates the processes inside containers from one another by putting them inside different namespaces.

##### But Unix, and therefore Linux, doesn’t just have one namespace, because these ideas were a bit new and undeveloped. It has user IDs and group IDs and process IDs and all sorts of things that aren’t expressed through the filesystem.

##### Cgroups namespaces split those up as well, for more complete isolation.

#### Back to the Plan

##### Different processes under Plan 9 have their own namespaces. Far more stuff is done via the filesystem, and every process has its own view of the filesystem.

##### So, in effect, **every** process is in a container…

##### This in a project announced one full decade before the first release of FreeBSD with jails.

#### The bad

##### If Plan 9 is so clever, why aren’t we all using it, then?

##### Some reasons are easy to enumerate.

###### Like Unix itself in the early days, it was a research OS. It wasn’t intended for production use: it was a tool for research into operating system concepts.

###### It wasn’t open source. But in 2000, the 3rd edition was released as FOSS under its own custom licence agreement, and then in 2014 it was relicensed under the GPL.

###### Its very clean, minimalist conceptual design brings penalties. As a small example that might surprise people: since there isn’t some “real” underlying view of the “true” filesystem, that means that there are quite a lot of things Plan 9 does not do. There are no links, either symbolic links or hard links. There’s no ‘mv’ command. The only way to move a file, or a directory tree, from one place to another is to copy it, then remove the original. This is the price of having the kernel talk to the local filesystem the same way that it talks to the filesystems on other machines: via the 9P protocol.

As an aside: I know this sounds extremely limiting, but it’s really not. I was some 5 years into my career when Windows NT was first launched. That was the first Microsoft OS with hard links. I deployed and supported commercial computer systems running MS-DOS, and until MS-DOS 5 in 1991, it didn’t have a MOVE command. Until NT, it didn’t have any kind of links, and the useful “shortcut” functionality only appeared in Windows 95.

###### Plan 9 is hard, unfriendly, and unforgiving. I am not some kind of frothing Plan 9 advocate, I swear. I have a copy of 9front, the most modern and often updated distribution, in a VM. Over the years, I occasionally get a fresh version and play around with it.

9front has come quite a long way from the AT&T version. A non-guru such as myself can install it and try it, which is more than I could with the original version.

Plan 9 has a GUI, but it’s a very strange one. But then it’s a very strange OS.

The thing is, though, that when I started playing around with Linux in about 1995 with kernel 1.0, it was extremely basic, very **very** unfriendly, and not very much actual use for anything. Linux has come an extraordinarily long way since then, even if not many people actually use it directly.

###### The killer, though, is that Plan 9 is enough unlike Unix that it’s incompatible with Unix. It’s not called Unix for a reason. It changes fundamental parts of the design in ways that mean that bringing existing source code across and recompiling it won’t work.

To pick a trivial example… Plan 9 C prohibits nested #include directives. Most non-trivial C programs contain lots of files of program code, and those files all contain lots of #include statements. Plan 9 C enforces a rule that only the top-level C file in a program can #include other files, and header files are not allowed #include at all.

This means a bit more work for the programmer, but it makes compilation much faster.

###### I don’t know of a snappy name for this, but to displace an existing well-established product, a new product has to be significantly better than it to make it worth the cost of switching – even if that cost is just in time and effort, not money. To steal an old OS/2 marketing slogan, Plan 9 was *a better Unix than Unix,* but it wasn’t better **enough**.

### And another thing

#### For what it’s worth, Plan 9 wasn’t the end of the original Unix line. There were 10 editions of Research Unix, then a successor that was so different it wasn’t Unix any more. That’s Plan 9 and it’s sort of UNIX 2.0.

#### There is also a very different successor to Plan 9 itself. It is called Inferno, and it is in a manner of speaking UNIX 3.0.

#### I’m not going to talk about it here today but it’s interesting too.

#### But for now, enough of weird experimental operating systems.

## Comprehensibility

### There is a problem here, and it increasingly seems to me that the industry is in denial about it.

### To go back to the late great Bucky Wirth and his *plea for lean software*.

### The core of the original Oberon is implemented in well under ten thousand lines of code. A2 is something like ten times bigger – I have not been able to find a firm estimate. Say it’s in the range of a hundred thousand to five hundred thousand lines of code.

### Debian 12, for comparison, is 1,341,564,204 lines of code. That’s the project’s own estimate. One and a third billion, that is one and a third thousand *million* lines of code.

### For comparison, Google Chrome is about 40 million lines, which is in the same ballpark as the Linux kernel these days.

### Nobody can read 1.3 billion lines of code. You won’t live long enough.

### This is the sort of size of codebase that we are building the internet from these days.

### These projects are so incomprehensibly vast that no human mind can comprehend even one small isolated subset of the entire thing.

### We consider this normal. Everything is like that. It’s just how it is. Computers are big, storage is cheap, interconnects are fast, and it works and it scales and it is all pretty amazing compared to the systems I started working with.

## Big lies

### You can’t buy software.

#### A hobby of mine is trying to clearly define some of the big lies in computing. One of my stories for the Register did quite well a year or so back, when I wrote that you can’t buy software.

#### You can’t. Wealthy international corporations can buy software, and they often do. But mere users can’t. All we get are licenses that say we own the right to use one copy, or if you’re a business, so many copies, and anyway you don’t get the source code, or any kind of guarantee.

#### That, of course, is why Free Software has done so well. If only big companies really own software then the only remain choice is software that nobody owns and nobody controls and that’s built and maintained by its community of users.

#### That’s why we are all here, right?

### So, let me look at another big lie:

### Computers aren’t much faster now than they were a decade ago, and they probably never will gain performance so fast again.

#### Moore’s law is over, replaced by Koomey’s law. Now, computers use less power, emit less heat, and they are still getting smaller and cheaper, but they’re not getting massively faster the way that they did in the 20th century.

#### Many of you remember the Pentium 4, Intel’s space-heater chip launched in 2000, which Intel planned to ramp up to 10 gigaHertz by 2005.

#### It didn’t happen. We got the smaller, cooler-running Pentium M instead, which evolved into the Core, Core 2 and then Core i-thingie ranges today. And thanks to AMD, they are 64-bit now.

#### What we got instead of much faster processor cores are more processor cores. The thing is, that doesn’t scale very well. On the desktop we have four-core machines and now we’re moving to eight plus cores, but a single person can’t use that very usefully so instead we’re getting computers with a mixture of high-performance but hot, power-hungry cores, and lower-performance, cooler and less hungry cores.

#### As Sophie Wilson, co-creator of the Arm chip, observed: the silicon chip industry is very good at finding ways of selling more and more products to us that spend ever increasing amounts of time turned off and not doing anything.

#### Server chips have lots more cores, of course, because servers can use that better, but that has a ceiling too. Look it up: it’s called Amdahl’s law, and it’s quite scary. Even if a program can be made 95% parallel, the maximum speedup you can get, no matter how many processor cores you throw at it, is about 20 times.

#### Once we started to get multi-core processors, Moore’s Law meant that the transistors were spent on getting wider, not faster. Now we are getting built in dedicated silicon for rendering 3D graphics, often disabled in favour of more powerful off-chip rendering silicon. Silicon for matrix arithmetic. Silicon for encrypting and decrypting. Silicon for modelling neural networks for so-called “AI” features.

#### That’s another big lie but one I won’t go into here.

#### Subsystems are getting faster too. Memory is getting faster. Solid state storage is getting faster, although the subject of my last talk, an attempt to bypass a whole pile of legacy bottlenecks and move non-volatile storage right onto the CPU memory bus, flopped. Killed by legacy software designs.

#### All this stuff helps certain specific functions, but it doesn’t make your general programs go faster.

#### In the mid-1990s, for one of the UK’s leading computer magazines, I ported their in-house benchmark suite from 16-bit Windows to 32-bit Windows. I do know a bit about benchmarking.

#### Benchmark vendors now include tasks like video compression in their tests, even though most computer users never do that, just because they need some way to test the performance of multi-core chips and show that they can do some stuff faster. These numbers are not lies, but they are no longer relevant to ordinary desktop or laptop computing. Or, mostly, to server computing either.

#### So, yes, computers are still getting a little bit quicker every year and a half, but up until about twenty years ago, they doubled in speed that often. Now it’s ten or fifteen percent.

### We have some big problems in this industry, and we are not confronting them.

#### Software is vast, and vastly complicated, and nobody can adequately understand it all.

#### It is too big to usefully change, or optimise. All we can do is nibble around the edges, removing bits here, making other bits a bit faster. It’s a fractal, so there are a near infinite number of edges to nibble at, but it’s still getting bigger so it’s getting harder all the time.

#### This also means we can’t check the whole thing. It’s too big. All we can do is have a lot of people watching it for when it goes wrong and find what happened and fix it.

#### Computers are getting more parallel. But, as far as anyone knows, it is simply impossible to automatically take algorithms and parallelise them. Only a human mind can do that, and yes, usually, it is **a** mind, in the singular.

#### That means that only chunks that can fit into one mind can be refactored like this.

### This level of bloat is a crisis that Wirth foresaw when it was 1% of today, and what he pleaded with the industry to consider and address.

#### There is an urgent need for smaller, simpler software. When something is too big to understand, then you can’t take it apart and make something smaller out of it.

#### You can trim it a bit, but to make profound changes, you have to go back to the planning stage, reconsider what you need, throw away what you don’t, and try to make some minimum viable product that does the essentials and nothing else.

#### This is what the creators of Unix did between about 1989 and 1992. They pared the conceptual framework of Unix back to its core, and built a new smaller core. They rebuilt Unix for clusters of networked machines.

#### But it was not sufficiently better than their previous product to displace it.

#### It’s good enough that several small teams of people have been working on it and improving it for thirty years. The 9Front project has done more than anyone else.

#### But the mainstream Unix world, which these days mostly­­ means the Linux world, has ignored them.

#### They are a bit irritated, and as a result, 9front is one of the most sarcastic and unapproachable projects in the FOSS world. Its website is famously cryptic and weird and hard to understand.

#### I can’t blame them.

#### But even so, I think it’s worth a look. Try it.

#### 9front is tiny. I talked to a couple of people from the 9front and Plan 9 community when I wrote about it in November, and one chap gave me some numbers.

#### **“The kernel is 5,119,091 bytes - 5MB or so.**

#### **“The entire distribution including all sources, documents, local Git repository and binaries is circa 530MB for amd64.”**

#### The Plan 9 kernel has 38 system calls. I’ve not been able to find a firm number for Linux, but I have some estimates.

#### As of 2016, kernel 4.7 had about 335. By 2017 that was up to 341. I’ve tried to estimate the number for kernel 6.8 and I think there are 520 across all architectures.

## *But but but…*

### 9front is clever and it’s impressive. It’s also weird and very unfriendly, but you know what, so was Linux when I started playing around with it a bit less than 30 years ago, and now look at it. It powers ChromeOS and Android. Several billion people use it every day and they don’t even know that they do.

### Even the distros, software for the small number of people who know what an operating system is and care which one they use, are pretty easy now.

### But it’s not Unix, and it can’t run Unix programs. You couldn’t port Firefox or LibreOffice to Plan 9.

### There are tools. It has a Linux emulator, like BSD does, called Linuxemu. It’s old and 32-bit only, but so was the OpenSolaris one until Joyent modernised it for SmartOS.

### There is a Linux compatibility layer for porting source code, called APE. There’s an X.11 server, called Equis. There are tools for this, but Plan 9 users don’t want Linux apps. If they did, they’d run Linux, and I suspect many do.

### 9front has a hypervisor, called VMX. It uses Intel virtualisation, a VM can have just one core, but it is there and it does work, although as the documentation says, “Vmx can and will crash your kernel.”

## So here is my idea.

### It is 2024. We all have 64-bit machines now and x86-32 is fading fast. Most distros no longer support it. We have lots of memory and lots of storage.

### These days, in Linux land, there are a number of “microVMs”, such as Amazon’s Firecracker. Recently that gained support for FreeBSD guests, too. There are others, such as Intel Clear Containers, which merged into Kata Containers, and so on.

### MicroVMs can start a guest in milliseconds, and they are used for “serverless” cloud computing. Another lie, but let’s not get into that.

### The idea is tiny VMs that are as quick and as ephemeral as containers. You start them, typically to run just one program, and when it completes, the VM exits.

### If it were possible to run a microVM on 9front, you could, in principle, start a Linux app, use it, and quit it again, and on a modern computer not notice the delay from starting the VM.

### That also reduces the need for a Linux emulator, a complex bit of code that needs constant maintenance to track the moving target of the current Linux kernel. Let it go.

### There’s an X server. If that were running already, the new VM could connect to it, and run inside Rio, the 9front window manager. I suspect that Equis needs some TLC and this would be very flakey at first, but the tech is there.

### There’s already a model for this. Qubes OS runs like this: all user programs run inside dedicated VMs, for security.

## On the Linux side

### How about a Linux distro, or distros, or a new mode of installation of existing distros, that is designed for and can only run inside a VM? It only has drivers for `virtio` devices. It boots from a folder provided by the VM, by PXE or whatever, so its kernel needs no disk drivers and no filesystems built in. No initramfs: the “hardware” is known in advance, and it’s 100% virtual. It mounts its root over 9P – there is already a Linux client for that. No console, just a network port. No ports of its own, no hardware to emulate.

### Tiny, simple. There’s no need to try to generalise it much at first: one build for KVM, one build for Xen, one for VMX, whatever you’ve got. One for Hyper-V or WSL2 if that is your kink.

## What killed Plan 9 last time was that it was incompatible with Unix.

### Well, using ordinary and quite commonplace technology that is established and used in production, we can fix that now.

### We can make Plan 9 able to launch tiny dedicated Linux VMs, to run ordinary Linux tools like, you know, a modern web browser, and make a descendant of Plan 9 that **can** run existing Linux tools.

## Why?

### Linux is mature now. So, for that matter, are the main BSDs. They are not changing that radically any more, and they are dropping support for older hardware.

### I think we need to think about where we might go next. As I said in my previous talks, I think it’s time to start thinking about what might replace the current generation of OSes.

### I put this idea to a couple of Plan 9 communities online, such as on Reddit, and while some of them made fun of the idea, that is entirely on brand. When you built a better mouse trap and the world ignored it for 30 years, it’s natural to be a little bitter.

### Quite a few people couldn’t see why anyone would want that, which is exactly what you might expect. Some suggested running Plan 9 in a VM under Linux instead, which is what I do.

### What surprised me is that mostly they didn’t tell me it was a horrible idea or that I was stupid and wrong.

### So, I’m here to throw the idea out there today.

### Have fun with it.