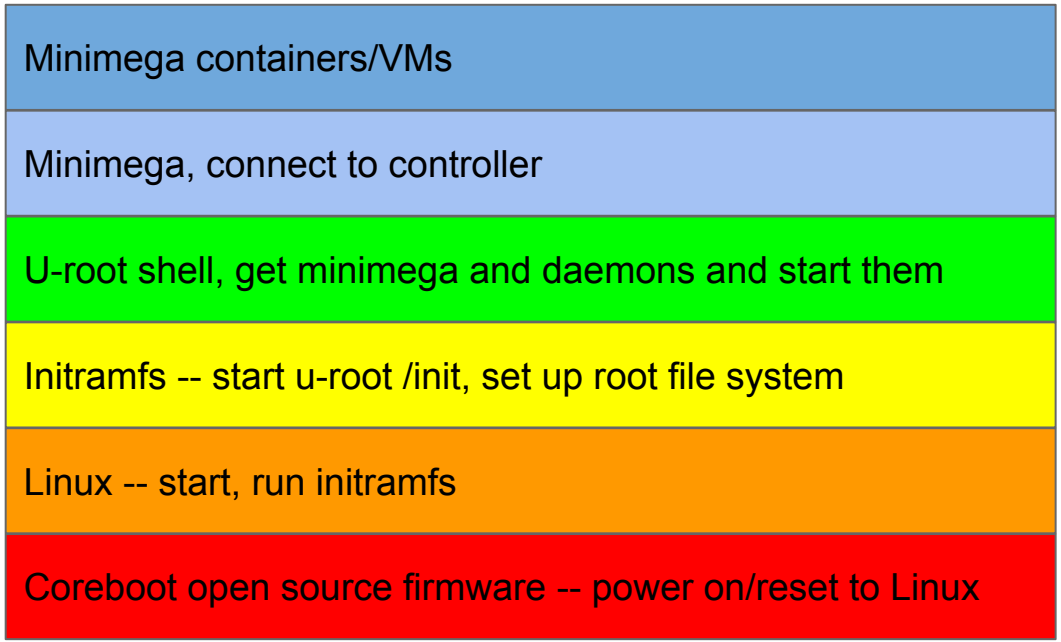
Creating dedicated emulytics devices with Linux, u-root, and minimega

Ron Minnich,
Gan-shun Lim, Ryan OLeary, Chris Koch,
Google

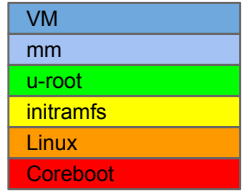
Andrey Mirtchovski
Cisco

John Floren, David Fritz
Sandia

Software Components



coreboot



- Open source firmware project I started at LANL in 1999 as LinuxBIOS
- *Completely* replaces legacy BIOS/firmware
- Several 100K HPC nodes used it
- All Chromebooks use it (over 20M by now)
- Boot in seconds, easy to change/extend
- For this work we have Linux in FLASH

Boot high level overview

- Power On/Reset
- Turn on 32-bit mode
- Turn on DRAM
- Enable devices
- Load “payload”
  - In this case that’s “Linux and initrd”
- Get out of the way
- Linux is up and running in about 2-3 seconds

linuxbios/coreboot in DOE HPC

- Built nodes that could *only* be HPC systems
- Security people loved it
- Inspired the core idea of this talk: embed Linux + userland + minimega in FLASH to create zero-configuration, embedded, deployable emulytics nodes
- Using u-root userland ...

u-root: A Go-based, Firmware Embeddable Root File System

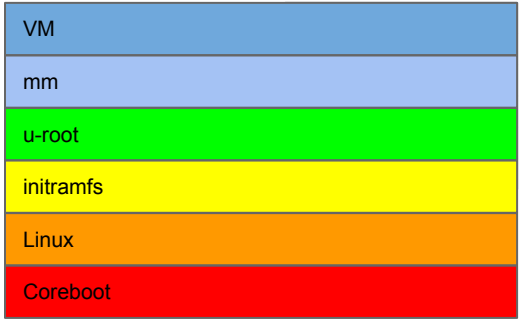
Ron Minnich, Gan-shun Lim, Ryan OLeary, Chris Koch, Google

Andrey Mirtchovski Cisco

Go in (much less than) 60 seconds

- New language from Google, released 2009
- Creators include Ken, Rob, Russ, Griesemer
- Not Object Oriented
  - By design, not ignorance
- Designed for systems programming tasks
  - And really good at that
- My main user-mode language since 2010

u-root



- Go-based rootfs
  - Commands/packages written in Go
  - In one mode, MAX, compiled on demand
- 1 or 4 pre-built binaries:
  - /init
  - Go toolchain -- if compiling on demand
- Type a command, e.g. rush (shell)
  - rush and its packages are compiled to /ubin and run
  - Compilation is minimal and fast (½ second)

u-root on small platforms

- Source-based u-root is down to 5.9M
  - But not everyone wants source in FLASH
- Now have “bb”, a.k.a. “busybox”, mode
- Using Go AST package, rewrite source and compile into one binary
  - /ubin/ forest of symlinks points to one binary
  - Binary examines argv[0] to figure out what to do
- Reduces to < 2M

Busybox rewrite mode

- With the ast package, we can rewrite programs as packages, e.g. ls.go

```
package main

var x = flag.String(“l”, …)
func init() {...}
func main() {
}
```

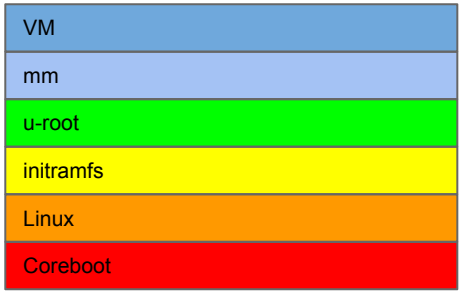
```
package ls

var x = flag.String(“ls.l”, …)
func Init() {...}
func Main() {
}
```

- Combine all of u-root into one program
- Turning 65 programs into one: 10 seconds

U-root boot time tasks

- Set environment
  - E.g. PATH=/bin:/ubin:/usr/local/bin
- create /dev, /proc, /etc
- Create inodes in /dev
- mount procfs, sysfs, cgroups, devfs
- Create minimal /etc/resolv.conf, GMT timezone file

Extending u-root initramfs



- Can add commands/systems
- Commands
  - Bb -cmds ‘/bin/bash /usr/bin/strace’
- Full trees
  - Bb -extra ‘/usr/share/X11 /usr/etc/X11 /usr/lib/X11’
- Go code (e.g. minimega)
  - Don’t add a Go program when you can compile in a Go command

Extending post-boot

- U-root provides command to import apps from tinycorelinux.net
- tcz [-h host] [-p port] [-a arch] [-v version]
  - Defaults to tinycore repo, port 8080, x86_64, 5.1
- Type, e.g., tcz bash
- Will fetch bash and all its dependencies
- Once done, you have
- /usr/local/bin/bash (can be in persistent disk)

Where to get it

github.com/u-root/u-root

Instructions on

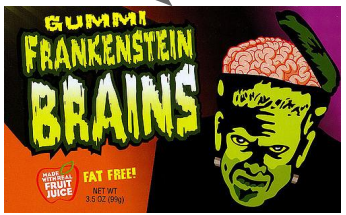
U-root.tk

Post-2005 Intel server CPUs: no coreboot for you! What to do?

- New project: NERF
- Non-Extensible Reduced Firmware
- Basic idea
  - Reduce functionality and extensibility of ME/UEFI
  - Linux provides functionality instead
  - Arrange UEFI to start Linux instead of UEFI boot dispatcher

Steps

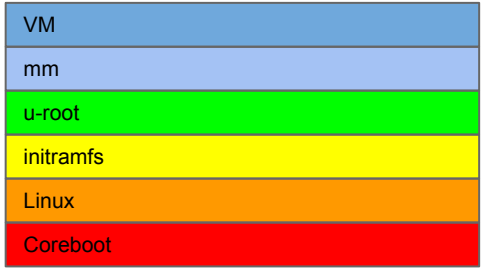
- ME clean to make room
  - Minnow MAX: reduces 5M to 300K for ME (!!!)
  - Remove UEFI drivers that provide net-, disk-, usb-, http-, file system- zero day support
  - Remove all UEFI apps (no more ring 0 apps!)
- Can replace stupid boot screen with Linux
- UEFI starts kernel and doesn't know it
- Servers are back after 10 years!



Status

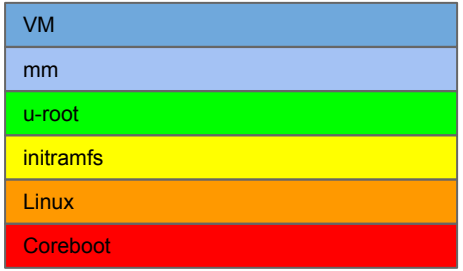
- Demonstrated on 4 motherboards including Facebook Open Compute Platform
- Hence we are not limited to coreboot systems

Applications



- Minimega was an obvious pick
- All Go-based
  - Easy to bring in to u-root “busybox” mode!
- Build dedicated minimega appliances
- Servers (NERF), minnowmax (coreboot), and all things in between

Minimega advantages



- Zero-conf
  - No dependency on init subsystem/systemd/etc.
  - Self configuring at node and net level
- Decentralized: no “special” nodes
- Compact code
- Uses Go’s extensive test/crossbuild support

A really big minimega advantage

- It grows u-root “busybox mode” code a negligible amount
- We can fit the whole stack in 8M FLASH
  - Lzma compresses to 4.9M
- Not a 500M disk image
- Pure Go, makes security people happy

Status

- Minimega in FLASH on minnowmax
- Minimega still needs a few commands u-root does not provide
  - WIP
- 8M flash is harder than we thought to due openvswitch bloat
- ‘wget’ openvswitch cpio as part of boot

Minimega IoT devices

- Build into a ChromeOS stack/device
- ChromeOS is not just about laptops
- It’s a ‘secure from power on reset’ platform
  - Platform hardware design is part of it
- Securely report results/be controlled/update
- Combine ChromeBox with this work to create widely dispersed networks

Beyond Facility-based Testbeds

- Current model is a ‘protected facility’ or ‘networked facilities’
- Why not build ‘immersed/dispersed testbeds’
- The pieces are all there with ChromeOS
  - Secure from power on/reset
  - Can be remote controlled
  - Can communicate results of experiments
  - Working at millions of nodes scale
- All open source

Other systems

- NERF partners include OCP hardware .com
- Linux-in-FLASH works on OCP nodes
  - As of Tuesday :-)
- Put minimega into these larger systems
- Six nodes to start
- Big rack as a test
- Large-scale, zero-conf, instant-on testbeds
- Connected with ‘immersed’ facility

Summary

- We’re reviving LinuxBIOS as a model
- With a Go-based userland for programs
- And minimega to build zero-conf, simple emulytics nodes
- Could be ‘dispersed’ or ‘facility-free’ testbeds using ChromeOS ideas and hardware