THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA
Embedded Linux Conference

Status of Linux Boot-time work

Tim Bird
Sony Electronics

Abstract

In this talk, Tim will describe the status of work to reduce boot-time for Linux systems. This include work by the Boot-Time Special Interest Group (SIG), as well as others in the Linux ecosystem. We will cover patches that have gone upstream to the Linux kernel and to systemd in the past year, their potential boot-time savings, and how to use them in your own projects. Patches in progress will also be discussed. Tim will summarize recent boot-time talks from other events (particularly Linux Plumbers Conference), highlighting some of the techniques that were described. Finally, Tim will present his own work to build a boot-time wizard program, to help developers find boot-time bottlenecks and areas where boot speed can be improved.

Outline

- Recent Boot Time Talks
- Patches going upstream
- Other Work in progress
- Tim's work...
- Resources/How to get involved

Boot-time major phases

- As a reminder...
- 3 main areas of boot time (and boot-time work) are:
  - bootloader/firmware
  - kernel
  - user space

Recent Boot-Time Talks

THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA

Embedded Linux Conference

ELC North America talks (April 2025)

- U-Boot status and new bootflow system - Canonical
- Automotive techniques – RedHat (*)
- Early ethernet – TI (*)
- Early boot and late attach- TI
- Boot-time regression test – Sony

- See https://elinux.org/ELC_2025_Presentations for links to slides and videos

U-Boot status and new bootflow system

- Status update on U-Boot and introduction of 'bootflow'
- Bootflow is a new way to structure boot actions and data
  - U-boot notoriously has complicated, vendor-specific scripts – this fixes that
- Author demonstrated 100ms boot from u-boot start to kernel start
  - Was on QEMU with kvm (x86_64)
  - Configured u-boot without cmdline (big memory savings)
  - Other CONFIG changes:
    - Remove bootdelay, disable networking, disable video, disable EFI
  - Results
    - 82ms from start of U-Boot to Linux handoff*
    - 174K u-boot.bin
  - See https://u-boot.org/blog/booting-into-linux-in-100ms/

Automotive techniques - RedHat

- A grab-bag of awesome and detailed techniques
  - Use kernel configs and command-line options to tweak memory init and RCU behavior
- Optimize kernel compression
- Don't use systemd-udev in initramfs
- Trim your SE Linux policy and udev rules
- Use systemd-analyze bootchart and plot
- Use stuff provided by Boot-Time SIG

Early ethernet - TI

- Ethernet is needed in automotive within 2 to 3 seconds
- Can't wait until kernel boot
- Start in firmware and hand off initialized hardware to kernel
- For hardware:
  - Use right media: OSPI for boot media, and emmc for file system
- For Kernel:
  - Disable unneeded drivers, trim device tree nodes, use tiny rootfs
- Disable or skip ethernet autonegotiation
- Start phy linkups early (before kernel)
- Prioritize systemd service for ethernet

Early boot and late attach - TI

- Need to start remote processors (other CPUs) before kernel
  - Normal method is kernel loads firmware and starts rprocs
- Bootloader (e.g U-Boot) can load rprocs and get them running before kernel
  - But bootloader doesn't instantiate the IPC between Linux and other processors
- Firmware initializes rproc tables, but each vendor does this their own way
  - Need drivers that initialize HW and rproc
  - Save important data, and pass info to kernel via DeviceTree
- Kernel drivers need changes, to not (re-)initialize hardware like they would in regular boot sequence
- Series of debugging lessons

Boot-time regression test - Sony

- Described a proposal for doing upstream regression testing of boot time
- Take into account that each machine is different
  - Regression values differ for platform (hardware) and Linux config and distribution
- Designed a system using saved reference values
  - Ref values used to determine pass/fail status of boot durations
  - Can test if initcalls suddenly increases in duration on a particular platform
  - Uses previous test run as ref values
- Matches platform under test to closest platform with saved ref values
- Extends KTAP to store data values (not just 'ok'|'not ok' results)

ELC Europe talks (August 2025)

- Unified Boot-Time Measurement
- Cost of secure boot and some reduction techniques

- See https://elinux.org/ELC_Europe_2025_Presentations for links to slides and videos

Unified Boot-Time Measurement

- Systemwide boot-time profiling
- Use a standard system for measuring all processor initializations on system
  - Inspired by U-Boot bootstage
- Each processor records data
- U-Boot gathers data, and passes it to kernel
- Uses an arch-specific (hw-initialized) cycle counter
  - May be different for different processors in a SoC
- Linux (user space) tool can read data and interpret to produce overall system view of data
  - Can also pass data to visualizer (for plots and charts)

Cost of security – Measuring and reducing boot-time impact

- Secure boot adds extra steps, to verify kernel and rootfs
- OptimizationTechniques:
- Use trimmed and simplified initramfs (custom init program in rust)
- Use hw-specific crypto routines for SHA256 algorithm (used by dm-verity)
  - e.g. use ARM hardware accelerators instead of generic C implementation
  - Cuts overhead of DM-verity by 450ms (only 40ms worse than no security check)

Plumbers talks (December 2025)

- Prefetching in Android
  - https://lpc.events/event/19/contributions/2154/
- Boot time optimization for embedded devices *
  - https://lpc.events/event/19/contributions/2231/

- See links above for slides and videos

Prefetching in Android

- Based on same concept as ureadahead
  - Load data ahead of actual request, so it's in the page cache when needed
- Pre-fetch data during early boot (Android loads 1.2G of data on a Pixel smartphone during boot)
- 2 phases: record and playback
  - Record is done after an update occurs (to catch new I/O patterns)
  - Playback is used on subsequent boots to pre-fill buffer caches
  - Record operation adds overhead
    - Need to eliminate wasteful 'record' step when update doesn't change I/O patterns
- About 6 – 8% improvement in boot time.

Boot time optimization for embedded devices

- Good list of different techniques (accumulated experience at Google)
- Use right compression for kernel (LZ4 faster than gzip)
- Tune your CPU frequencies
  - Big/little CPUs can cause latencies
- Use fw_devlink to optimize Device Tree driver probing and loading
  - fixes ordering issues, but need to test
  - New features support hints to deal with cyclical dependencies
- Use async probing and parallel module loading
  - Use /scripts/dev-needs.sh and tsort to get right ordering
- Optimize module loading
  - Reduce logging, defer firmware loading, defer module loading if not in critical path

Summary

- Lots of different techniques discussed
  - Too many to mention here
- You have to know what things you can sacrifice or eliminate
  - e.g. when reducing kernel size, trimming udev rules, or cutting systemd services
- Requires a LOT of expertise in the interactions and dependencies between features and services
- No one can provide specific advice, only: "you might want to try this"
  - It depends on your requirements
- Often, details on how to actually implement the change are missing
- But... There are a lot of great ideas available

Recent Patches

Patches

- Detailed probe instrumentation
- BOOT_TIME_TRACKER
- Async printk output for serial console
- systemd: better device filtering by udev-adm
- DRM Splash
- Probe Deferral Optimization
- Early times

Detailed probe instrumentation (Dec 2024)

- Add info to really_probe_debug()
  - Shows more information (probe function, driver name) on what's being probed
- Is useful for automated instrumentation (more on that later)
- See https://lore.kernel.org/linux-embedded/2904921.vuYhMxLoTh@fedora.fritz.box/

- Status:
  - Need to test more and re-send
  - Some info may be redundant

Detailed probe instrumentation – new output

- New lines have '+'

```
 [    0.043877] calling  brcmstb_l2_driver_init+0x0/0x28 @ 1
+[    0.043909] call to platform_probe in driver brcmstb_l2 for device fef00100.interrupt-controller
 [    0.044044] irq_brcmstb_l2: registered L2 intc (/soc/interrupt-controller@7ef00100, parent irq: 14)
 [    0.044077] probe of fef00100.interrupt-controller returned 0 after 164 usecs
 [    0.044111] initcall brcmstb_l2_driver_init+0x0/0x28 returned 0 after 230 usecs
...
 [    4.438385] calling  brcmfmac_module_init+0x0/0xd4 [brcmfmac] @ 294
+[    4.438601] call to sdio_bus_probe in driver brcmfmac for device mmc1:0001:1
 [    4.438734] probe of mmc1:0001:1 returned 0 after 127 usecs
+[    4.438744] call to sdio_bus_probe in driver brcmfmac for device mmc1:0001:2
 [    4.439112] brcmfmac: F1 signature read @0x18000000=0x15264345
 [    4.441209] brcmfmac: brcmf_fw_alloc_request: using brcm/brcmfmac43455-sdio for chip BCM4345/6
 [    4.441292] probe of mmc1:0001:2 returned 0 after 2544 usecs
+[    4.441314] call to sdio_bus_probe in driver brcmfmac for device mmc1:0001:3
 [    4.441400] probe of mmc1:0001:3 returned 19 after 81 usecs
 [    4.441502] usbcore: registered new interface driver brcmfmac
 [    4.441508] initcall brcmfmac_module_init+0x0/0xd4 [brcmfmac] returned 0 after 3068 usecs
```

systemd: better device filtering by udev-adm

- Adds an option (-i) to systemd 'udev-adm trigger' command
  - This makes it so that parents are matched even if they are filtered out
  - Significantly reduces the number of devices that udev-adm tries to initialize during cold-plug processing
- See https://github.com/systemd/systemd/pull/35155

- Status:
  - Accepted into systemd-udev as Dec 2024
- How to use:
  - Read this: https://elinux.org/Systemd_Udev_tuning#Udev_tuning

BOOT_TIME_TRACKER (Aug 2025)

- Adds some BOOT_TIME markers to the kernel
  - Adds 2 kernel instrumentation points (printks with cycle counts) to kernel
- Useful for Universal Boot-time Measurement
  - Uses same cycle counter as U-Boot so kernel event timings can be integrated into overall view
- See https://lore.kernel.org/all/20250823044034.189939-1-v-singh1@ti.com/
- Status:
  - Stalled at RFC stage
  - Was a simple patch, but got pushback
  - May need to harmonize with the early-times patch
    - Both add (pre-'init_time') access to platform-specific cycle generator

async printk output for serial console (Sep 2025)

- Make printk async-friendly so serial console delays don't slow it down
  - Release console_lock_between printing records
- by The Good Penguin
- Fixes lock contention on the console lock
  - Adds higher granularity of the lock/unlock
  - But requires PREEMPT_RT
- Patches at: https://lore.kernel.org/all/20250927-printk_legacy_thread_console_lock-v2-0-cff9f063071a@thegoodpenguin.co.uk/#t
- See: https://www.thegoodpenguin.co.uk/blog/we-made-linux-v6-19-boot-quicker-for-everyone/

- Status:
  - Accepted upstream as of 6.18 kernel
- How to use:
  - Need to turn on PREEMPT_RT, then just sit back and enjoy

systemd: parallel module loading (Aug 2025)

- Load modules in parallel in systemd
- About 300ms improvement on i.MX93 FRDM board (2x Cortex A55 @ 1.7Ghz)
- By RedHat

- See https://github.com/systemd/systemd/pull/38549

- Status:
  - Accepted upstream
- How to use:
  - Sit back and enjoy, or tune with SYSTEMD_MODULES_LOAD_NUM_THREADS environment variable

DRM Splash

- Adds an in-kernel bootsplash for the Linux DRM subsystem
  - This has been wanted for some time
- Boot splash screens help with perceived boot time
- See https://lore.kernel.org/all/20260510-drm_client_splash-v3-0-a9aee9f0b2fc@valla.it/

- Status:
  - RFC version 3 submitted on May 10, 2026
- How to use:
  - See related Kconfig entries in drivers/gpu/clients/Kconfig, or read this:
  - https://lore.kernel.org/all/20260510-drm_client_splash-v3-1-a9aee9f0b2fc@valla.it/

Probe deferral optimization

- Discussion on linux-embedded list about various methods used to avoid deferred probe by device drivers during boot
  - Tune configs, remove unneeded dependences in device tree
  - Some methods require detailed analysis and manual work
    - Reorder items in Makefile, change initcall level of initcall
  - Also used /etc/modprobe.d/ conf file with softdep lines to help order things better for modprobe
- On BeaglePlay, reduced boot by 240ms
- See https://lore.kernel.org/linux-embedded/20251126-beagleplay-probes-v1-0-c833defd4c9b@valla.it/

- Status:
  - Discussion only, see thread for techniques and patches
  - Not sure which items got accepted upstream

Other Work in-progress

Other Boot-time Work In-Progress

- Systemd tuning
- Boot cache
- Unified Boot-time Measurements
- U-Boot Falcon Mode work

Systemd tuning

- You can improve systemd boot overhead by removing a lot of features using compile-time configuration options
- Use systemd-analyze to detect bottlenecks and long-duration services
- See https://elinux.org/Systemd_Udev_tuning
  - This page describes process of tuning systemd
  - Was added recently to the elinux wiki

- Status:
  - Information is available – please enhance the page if you see issues or try it out

Boot Cache

- Allows kernel to cache result of expensive computations during boot
- Kernel recalls data on subsequent reboots to avoid redundant hardware detection and initialization work
  - E.g. detection of best crypto algorithm, or RAID configuration
  - Maybe useful for avoiding ethernet autonegotiation??

- RFC to kernel list in Sept. 2025
  - See see https://lore.kernel.org/linux-embedded/20250923-bootcache-v1-0-4f86fdc38b4e@thegoodpenguin.co.uk/T/#m4d7c4a87861cc6e1995619a4b3a1a4c013083d3d

- Status:
  - Stalled, but Good Penguin still working on it

Unified Boot-time Measurements

- *Work was described previously*
- Here are links to the talk:
  - Slides: https://elinux.org/images/7/7c/Unified-Boot-Time-measurement.pdf
  - Video: https://www.youtube.com/watch?v=9q9u2Uw3f5o

U-Boot Falcon mode work

- U-Boot usually consists of both SPL and U-Boot proper
- In Falcon mode, you just use SPL (skipping U-Boot proper)
- On AM62 platforms:
  - Security boot processor runs first and does it's security stuff
  - Then loads Cortex R, then Cortex A
- Falcon mode saves 1 to 2 seconds of boot time

- Some reference docs:
  - Generic falcon mode: https://docs.u-boot.org/en/latest/develop/falcon.html
  - TI specific docs: https://docs.u-boot.org/en/latest/board/ti/am62x_sk.html#falcon-mode
  - TI SDK docs: https://software-dl.ti.com/processor-sdk-linux-rt/esd/AM62X/latest/exports/docs/linux/Foundational_Components/U-Boot/UG-Falcon-Mode.html

More firmware boot-time improvements

- What Texas Instruments is talking about at this conference
  - Optimizing U-boot using bootph tags
- Make sure to see "Bootph: A Swiss Army Knife for Boot-time Optimization"
  - 208A+B at 4:30 today (right after this talk)

Tim's work

Tim's work

- early times patch
- Boot-time wizard
- Some other tools:
  - grab-boot-data.sh
  - Boot-data tool
- Boot-Time wiki

Early times Patch

- Goal is to eliminate zero timestamps in early boot
  - Want to completely close the kernel "blind spot"
  - Avoid overhead of new instrumentation
- Submitted multiple times, with different approaches:
  - First version was used get_cycles() (uninitialized) and a hook in printk
  - Second and third versions fixed issues and used a different technique:
    - Saved cycles at submit time, and displayed microseconds when dmesg was run later
    - Caused lots of confusion – discontinuity with other timestamps, output to console differed from dmesg output, etc.
  - Fourth version reverted back to simpler use of direct counter instruction
    - Required hardcoded (compiled-in) constant CONFIG_EARLY_CYCLES_KHZ

Early times (blind spot fix) Output

- Is useful, here is some dmesg output:

```
[   11.380584] TRB: at start of start_kernel()
[   11.380677] Linux version 6.12.22-linux4microchip+fpga-2025.07-...
[   11.384001] Machine model: BeagleBoard BeagleV-Fire
[   11.384019] SBI specification v1.0 detected
...
[   11.603752]   node   0: [mem 0x0000001022000000-0x000000102f3fffff]
[   11.603772] Initmem setup node 0 [mem 0x0000000080000000-0x000000102f3fffff]
[   11.631797] On node 0, zone DMA32: 24576 pages in unavailable ranges
[   11.658335] On node 0, zone DMA32: 40960 pages in unavailable ranges
...
[   12.432015] riscv: providing IPIs using SBI IPI extension
[   12.432127] rcu: srcu_init: Setting srcu_struct sizes based on contention.
[   12.432718] clocksource: riscv_clocksource: mask: 0xffffffffffffffff max_cycles: 0x1d854df40 ...
[    0.000003] sched_clock: 64 bits at 1000kHz, resolution 1000ns, wraps every ...
```

Early times (blind spot fix) Output

These use to be zeros

- Is useful, here is some dmesg output:

```
[   11.380584] TRB: at start of start_kernel()
[   11.380677] Linux version 6.12.22-linux4microchip+fpga-2025.07-...
[   11.384001] Machine model: BeagleBoard BeagleV-Fire
[   11.384019] SBI specification v1.0 detected
...
[   11.603752]   node   0: [mem 0x0000001022000000-0x000000102f3fffff]
[   11.603772] Initmem setup node 0 [mem 0x0000000080000000-0x000000102f3fffff]
[   11.631797] On node 0, zone DMA32: 24576 pages in unavailable ranges
[   11.658335] On node 0, zone DMA32: 40960 pages in unavailable ranges
...
[   12.432015] riscv: providing IPIs using SBI IPI extension
[   12.432127] rcu: srcu_init: Setting srcu_struct sizes based on contention.
[   12.432718] clocksource: riscv_clocksource: mask: 0xffffffffffffffff max_cycles: 0x1d854df40 ...
[    0.000003] sched_clock: 64 bits at 1000kHz, resolution 1000ns, wraps every ...
```

Early times Status

- Upstreaming status:
  - Encountered negative reaction by Thomas Gleixner
  - But positive reactions from others
  - There was some discussion weirdness by an uninvolved party
  - Plan to address and maybe submit again
- I'm using it internally
  - Needed in Boot-Time tuning wizard for automated printk instrumentation
- I'll publish it separately if it doesn't get accepted upstream

Boot-Time Wizard

- Overview
- Architecture
- Results so far

Boot-Time Tuning Wizard - Overview

- Tool to help automate task of tuning your system for boot time
- Can automatically apply a change and measure results
- Vision – simple tool that provides easy instructions and hard data for improving boot time

- Why?
  - Lots of suggestions exist for tuning system, some without hard data
    - Effect on your system of each tuning method will be unique
  - Goal is to quickly and efficiently help you see which techniques will help
  - Also, provide clear instructions to apply technique to your system
    - e.g. list exact steps: items to configure, disable, etc.

Boot-Time Wizard - Architecture

- Has several pre-defined techniques for improving boot-time
- Applies each one, and generates a report of what changed
- Main loop:
  - Instrument kernel, build, boot, measure baseline data
  - Apply optimization technique, build, boot, measure system (for delta)
  - Restore system
  - Generate report of differences
- Currently, human has to interpret report to decide if tradeoff should be accepted

Boot-Time Wizard - Details

- Each technique has it's own:
  - Instrumentation points
  - Optimization actions
  - Items to highlight in report
- Instrumentation is one of:
  - Apply a patch
  - Insert printks
  - Adjust kernel command line
- Current optimization technique categories:
  - Adjust kernel config
  - Adjust kernel command line
  - Disable device tree nodes
  - Disable systemd units

Boot-Time Wizard examples

- disable-bluetooth-dt
  - Disabling the bluetooth node in device tree reduced overall start time by 175ms
  - Info from systemd-analyze shows bluetooth.service duration was 460ms, and was not present with bluetooth node disabled

```
== diff for section SYSTEMD INFO ==
item                                    file1   file2   delta
======================================= ======= ======= ===========
SYSTEMD_FULL_START_USECS                 14.284  14.109  -0.175 ! secs
SYSTEMD_USERSPACE_USECS                  13.195  13.024  -0.171 ! secs
...
avahi-daemon.service                      0.472   0.426  -0.046   secs
bluetooth.service                         0.460    ----  -0.460 ! secs
dbus.service                              0.415   0.393  -0.022   secs
```

Boot-Time Wizard examples

- low-mem-1G-cmdline
  - Optimization technique was to add 'mem=1G' to kernel command line
  - System was instrumented with early_times patch, and printk instrumentation around early memory initialization functions
  - System was normally 4G - specifying 1G for startup* saved ~50ms in these routines
    - *Remaining memory would have to be added in deferred manner after critical startup

```
== diff for data in section KERNEL MESSAGES ==
... (highlights only)

region                                  file1   file2   delta
======================================= ======= ======= ===========
mm_core_init                             93439   61492  -31947 ! usecs
mm_core_init_early                       26293    8109  -18184 ! usecs
kernel                                     999    1008       9   usecs
mm                                          28      25      -3   usecs
```

Boot-Time Wizard examples

- disable-graphics-cmdline
  - Optimization technique was to set CONFIG_DRM=n in kernel configuration
  - Toothpaste effect – not as much savings as expected, delays shifted, not sure why

```
initcall                                  file1    file2    delta
========================================= ======= ======= ===========
...
deferred_probe_initcall                  111317  111495      178   usecs
cfg80211_init                             16333    9083    -7250   usecs
v3d_platform_driver_init                  14457    ----   -14457 ! usecs
rmem_driver_init                           8361   29665    21304 ! usecs

== diff for section SYSTEMD INFO ==
item                                       file1    file2    delta
========================================== ======= ======= ===========
SYSTEMD_FULL_START_USECS                    14.283   14.162   -0.121 ! secs
SYSTEMD_USERSPACE_USECS                     13.207   13.077   -0.130 ! secs
```

Boot-Time Wizard - Results

- Findings:
  - Disabling device tree Bluetooth node ended up loading different kernel modules
    - I was not expecting lsmod diffs
  - Reducing memory resulted in lots of feature failures (on one platform)
    - Ethernet needs at least 256M of DMA memory??
  - Automation is good at finding interactions between CONFIGs
    - Kernel is supposed to be resilient to CONFIG changes, but is sometimes not
- Need more instrumentation points to determine where delays occur
  - To find culprits when the 'toothpaste effect' occurs
- Had to extend ttc (board automation tool) to handle kernel cmdline

Boot-time wizard - status

- Is still in early development (alpha phase, at best)
- What's missing:
  - Consolidated report with recommended tuning steps
  - Robustness:
    - Better handling for build failures, platform failure to boot, etc.
- NOTE – it depends on "ttc" board farm management tool
- I wouldn't try it yet...
- But it's available here:
  - https://github.com/tbird20d/boot-time-wizard

grab-boot-data.sh

- Is a small utility to gather data from a system for boot time analysis
- Retrieves data about running system, including:
  - Kernel and distro version
    - kernel configuration, cmdline
  - Machine info: memory, cpus, processes, mounts, loaded modules
  - Dmesg (kernel message log)
  - Systemd unit boot information
- Creates a 'boot-data' file
- Automatically sends the data to the boot-time wiki
  - Or not

'Boot-data' tool

- Processes data from a boot-data file, and produces reports
  - Shows initcall durations
  - Instrumented region durations
- Can report differences between 2 boots:
  - What configs are different
  - What loaded modules are different
  - Initcall timing deltas
  - Instrumented regions that changed durations
  - Systemd unit duration changes
- Is used by boot-time tuning wizard, but is handy as a standalone tool

Boot-time Wiki

- A place to collect boot-data from lots of different systems
- Intention is to provide reports, in nice format
- Also, to find patterns across platform types and kernel versions
  - initcalls that are long duration on lots of systems
  - Regressions in kernel boot time, by kernel version
  - Effects of different configurations and command line options on boot time
  - Effects of other system attributes (memory, started services, running processes) on boot time
- Initial push to collect data was in 2024
  - Need more data for analysis

Resources
and
How to get involved

Boot-time SIG

- Linux boot-time Special Interest Group (SIG)
- Informal group of companies and individuals interested in boot-time
- See https://elinux.org/Boot-Time_SIG
- See minutes of meetings here:
  - https://docs.google.com/document/d/1XAufoTT6VVJOTMzKMoz8SyOss-JA9H4J1_yVXQq5mN0/edit?usp=sharing
- Mailing list:
  - linux-embedded@vger.kernel.org

elinux Wiki

- See https://elinux.org/Boot_Time
- Used for SIG activities
- Formerly used by the CE Linux Forum Boot-Time Working Group
- Resources:
  - https://elinux.org/Boot_Time#Technologies_and_Techniques_for_Reducing_Boot_Time
    - Some data is old, but there are recent additions
  - Presentations: https://elinux.org/Boot_Time_Presentations

Boot-time wiki

- https://birdcloud.org/boot-time/
- Goals:
  - Support boot-time research and collaboration
  - Hold data that allows for pattern-recognition and regression checking
  - Provide support for developers working on boot-time
- Want to have a page for each optimization technique:
  - With description, how to implement or use, and example improvement (numbers!)
  - Modeled on elinux pages:
    - Example: https://elinux.org/Disable_Console
  - Would like to automatically populate page data, with the Boot-Time Wizard

Old tried and true techniques

- Put 'quiet' on your kernel command line
- Make your kernel image as small as possible
  - Disable everything you don't need in .config

Conclusions

- Poor boot-time is a result of "death by a thousand cuts". The way forward unfortunately requires a thousand tiny fixes.
- Unfortunately, generalized solutions are not possible, because what you can do depends heavily on your hardware, your boot requirements, and the features needed in your product
- Boot-Time wizard intends to help you find what works, by giving data in an automated fashion
  - The automation encapsulates different optimization techniques, including:
    - What to instrument
    - How to apply the technique
    - Providing data, for your system, about the effects of the change.

Resources

- TI's guide for quick booting an AM62X processor
  - https://lore.kernel.org/linux-embedded/2904921.vuYhMxLoTh@fedora.fritz.box/
- RedHat's boot time analysis tools
  - https://gitlab.com/CentOS/automotive/src/boot-time-analysis-tools
- Google's Boot-time Optimization tips
  - https://source.android.google.cn/docs/core/architecture/kernel/boot-time-opt
- Introduction to kernel boot sequence:
  - https://bootlin.com/blog/demystifying-kernel-boot-sequence/
- Boot-time presentations:
  - https://elinux.org/Boot_Time_Presentations

Thanks for your time