Boot time optimization for embedded devices

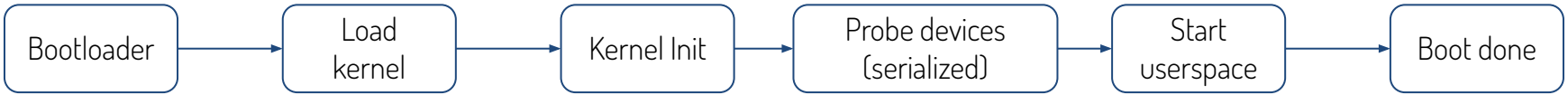
Saravana Kannan

東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Let's Be Interactive!

Questions? Interrupt me!

Typical boot flow



Boot loader – compression algorithm

- Use LZ4 compression for kernel and ramdisk
  - Gzip produces a smaller image, but decompression is slower
  - With larger and faster storage, the decompression time is the bottleneck.
  - Saved 500 ms to 1000 ms on an Android phone.

Boot loader – picking the right frequencies

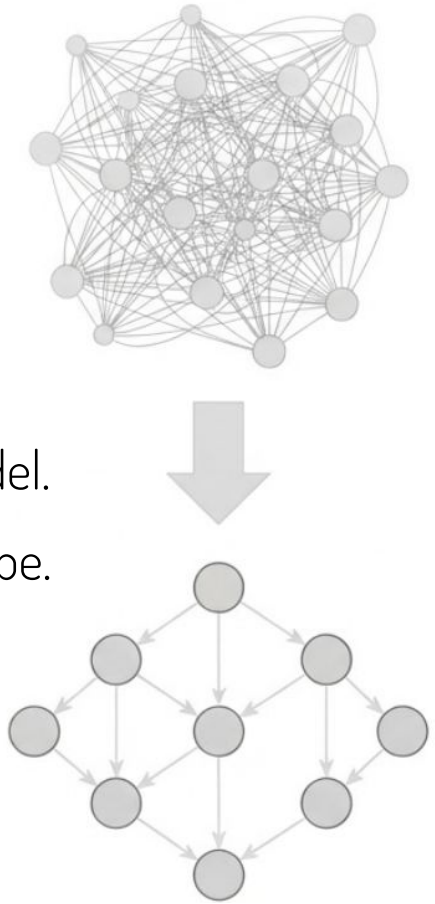
- Can't always max out CPU frequencies.
- Thermal concerns during boot loops.
- Set CPU freq high as safely possible.
- Don't forget about L3, interconnect, DDR and anything else between CPU and DDR.
- Anything else using DDR during boot? Splash screen animation?
    - Their memory and interconnect frequencies needs to be reasonable too.
    - Can cause back pressure on DDR and slow down CPUs reads.

Boot loader – Big/little pitfalls

- Big CPUs can slow down boot if you aren't careful.
- Say:
  - Big CPU DMIPS/MHz = 2 * little CPU DMIPS/MHz
  - little CPU freq = 1800 MHz, big CPU freq = 300 MHz
  - Big CPU DMIPS = 33% of little CPU DMIPS.
- CPU freq doesn't probe until device initcall.
- Scheduler assumes same frequency on all CPUs. Picks "Big" CPU for all threads.
- Boot loader should set Big CPU DMIPS >= little CPU DMIPS

Kernel - fw_devlink

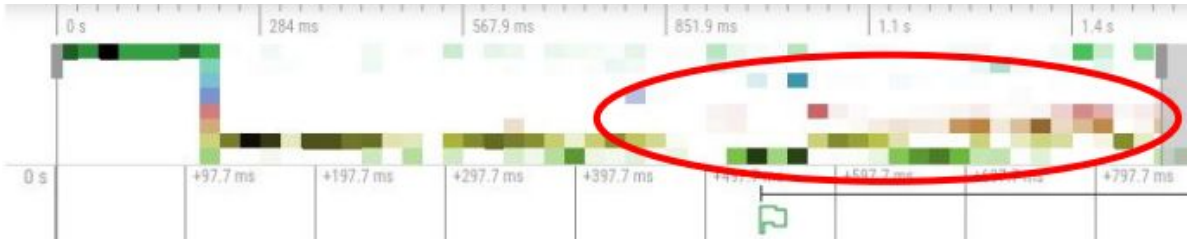
- Builds a complete dependency graph of all devices from DT.
- Enforces correct probe ordering.
- Avoids repeated deferred probing and saves times.
- Use **post-init-providers** to break dependency cycles.
- Avoid legacy macros like OF_DECLARE that bypass driver model.
- Module init should just register drivers, not act as a device probe.



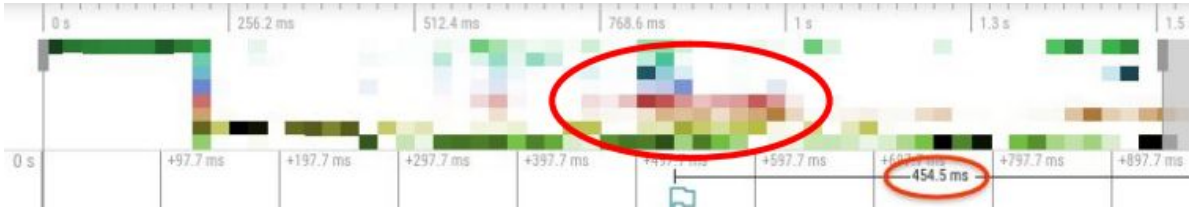
Kernel – Enable asynchronous probing globally

- **driver_async_probe=*** will enable async probing for all the drivers.
- **driver_async_probe=*,X,Y,Z** – same as above except for drivers X, Y and Z.
- **module.async_probe=1** enables async probe for all modules.
- **moduleXYZ.async_probe=0** can override that for moduleXYZ.

Sync: 900 ms



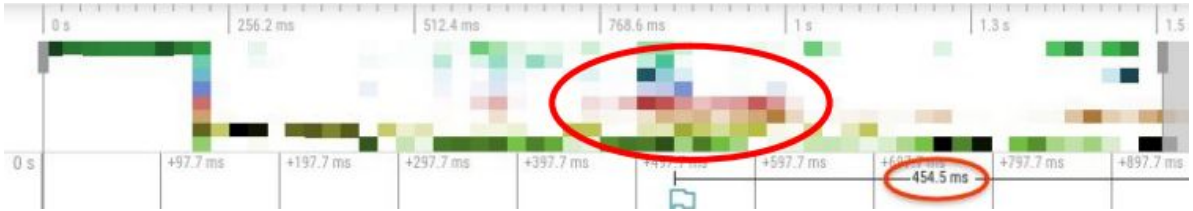
Async: 450 ms

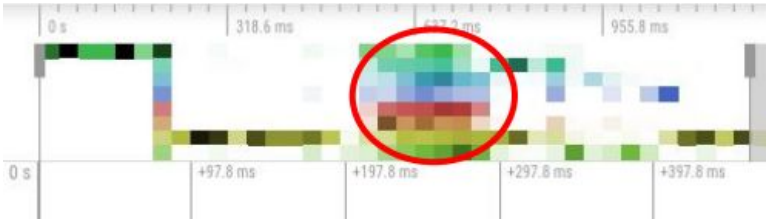


Kernel - Parallel module loading

- Even better than asynchronous probing.
- Needs userspace support to load multiple modules in parallel (use all CPUs)
- Android 13+ parallel module loading: **androidboot.load_modules_parallel=1**
- Async probing on top of this might slow things down depending on the board

Async: 450 ms





Parallel mod: 250 ms

Kernel - Load modules in the right order

- Two ways to go about it:
  - Load modules based on devices being added (uevent based loading)
  - Load modules ahead of time - modules of parents/suppliers before modules of children/consumer (used in Android).
- Depending on the system/board, one might be faster than the others.
- Lots of android devices just load all the modules ahead of time.
- Use **scripts/dev-needs.sh** and **tsort** to get the list of drivers and modules in the right order.

Modules – Strip symbols

- Strip debug symbols.
  - Reduces time to read ramdisk/modules from flash
  - Reduces ramdisk decompression time
  - Reduces kernel module loading time.
- Can save several seconds of boot time.

Modules – Avoid excessive logging

- Not just about reducing size of module.
- On ARM32/64 module load does a O(N log N) sorting of function call sites.
- Can save up to 300 ms depending on how bad module logging is.

Modules – Load firmware on demand

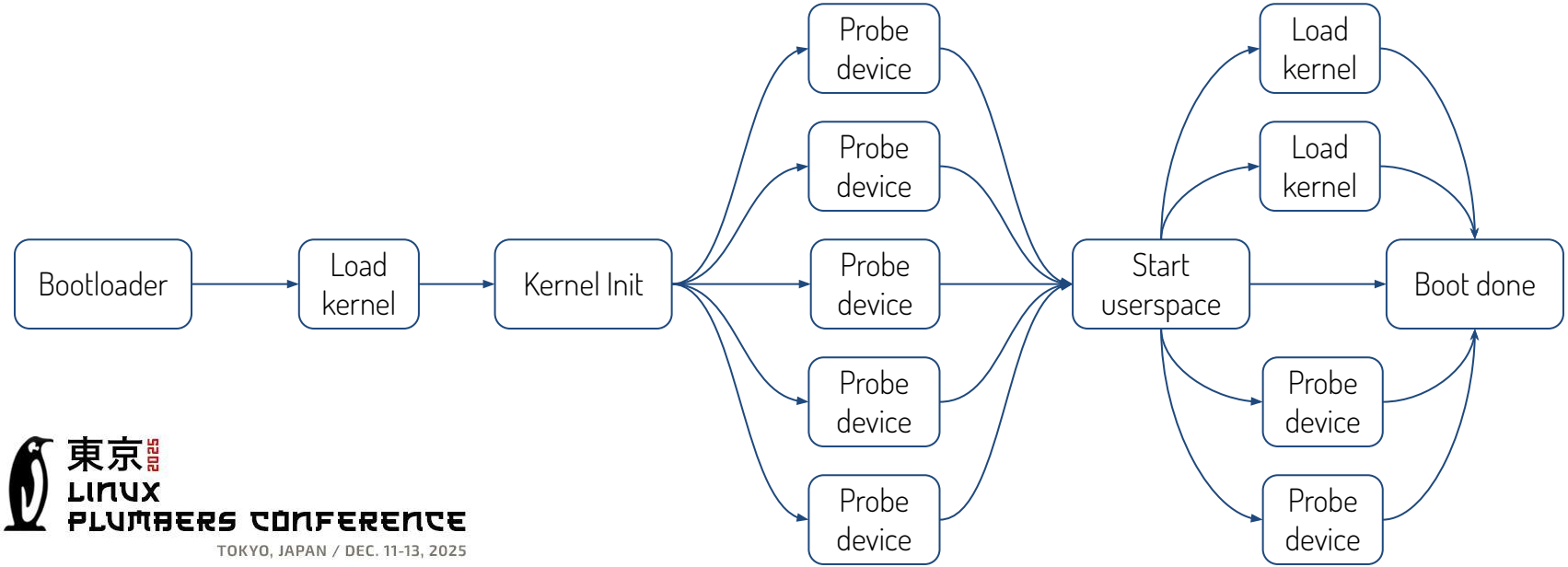
- Not a generic guideline.
- Instead of loading large firmware in device probe (which runs in the initcalls), load them on demand when the device is used.
- Not a blanket statement. Do this only for drivers that need to load large firmware.

Modules - Load modules during userspace start up

- Defer loading of any non user-perceptible modules to a later stage.
- Eg: Do you need to load the Camera driver before the lock/login screen?
- Modules can be loaded while userspace is also starting up.
- Parts of userspace might need to wait on sysfs/device files showing up.
- Can save 500 ms to 1000 ms.

Fully modular kernel

- Counterintuitively, a fully modular kernel can be faster than a static kernel.
  - Parallelized module loading.
  - Explicitly load CPUfreq driver as one of the first/earliest modules.



Thank you!