

# Linux Kernel Fastboot

Feng Tang  
Intel Linux System Engineering

# Kernel Fastboot

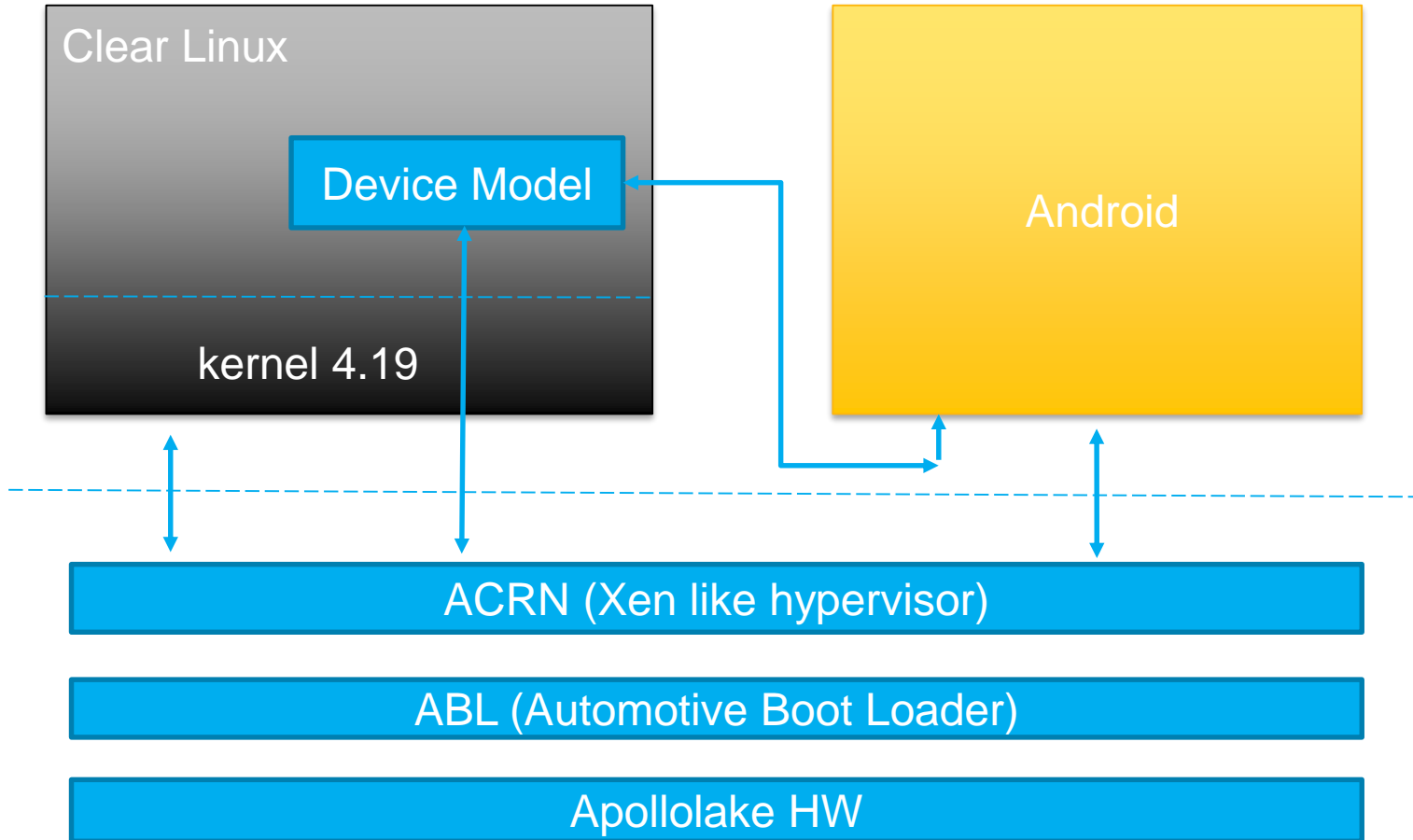
- Linux kernel fastboot is critical for all kinds of platforms
- At LPC 2008, Arjan van de Ven and Auke Kok introduced “**Booting Linux in five seconds**”
- Kernel boot time has been hugely improved over years, but there is always space to improve
- Will talk about general stuff, while using a x86 In-Car platform as reference

# Why we worked on boot optimization

- Hard requirement: **rear camera** must be functional in **2 seconds** after power on.
- The boot phase contains HW, FW, bootloader, hypervisor, kernel and user space, pre-kernel takes **500 ms**
- Initial kernel boot time is **3 seconds**, finally we cut it to **300 ms**



# Platform Info



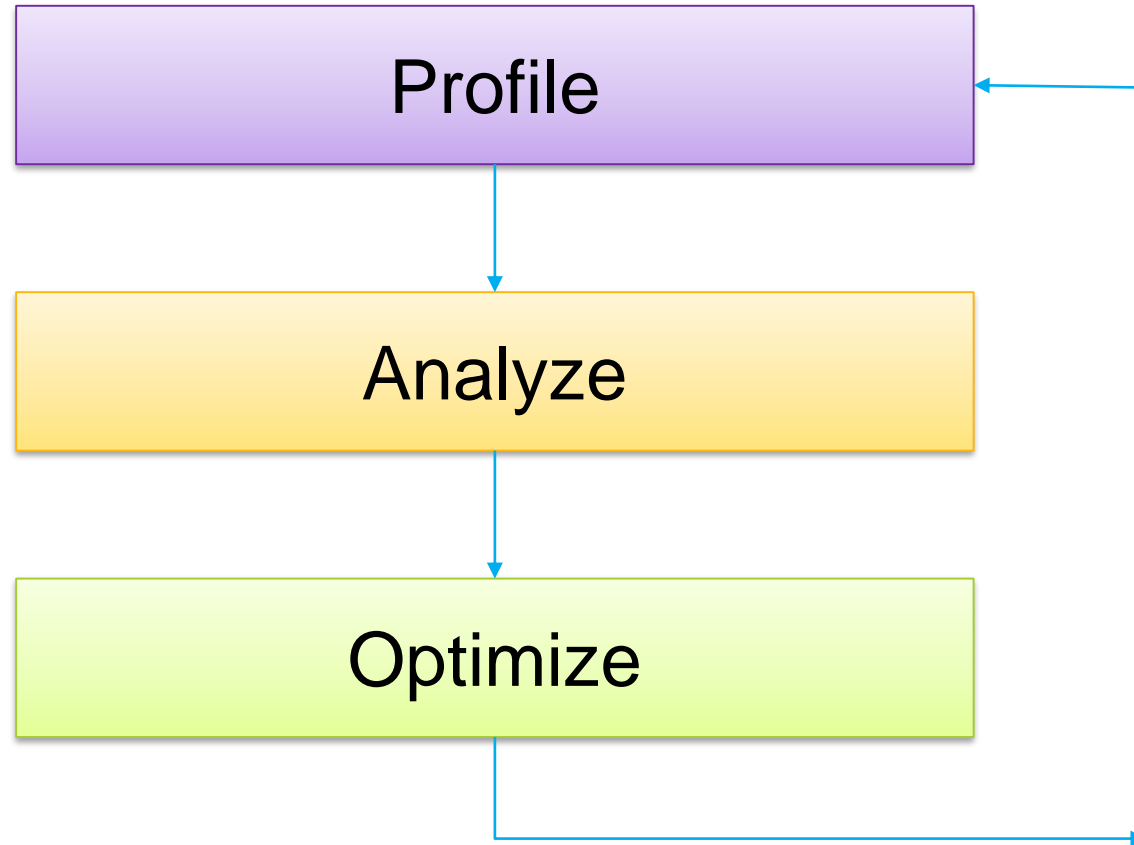
## HW info:

- \* Apollolake 4 Core (1.9G/2.4G)
- \* 8GB RAM
- \* 16GB EMMC roots

## SW info:

- \* VMM: ACRN hypervisor
- \* OS: Clear Linux with 4.19 LTS kernel

# Methodology – 3 Steps



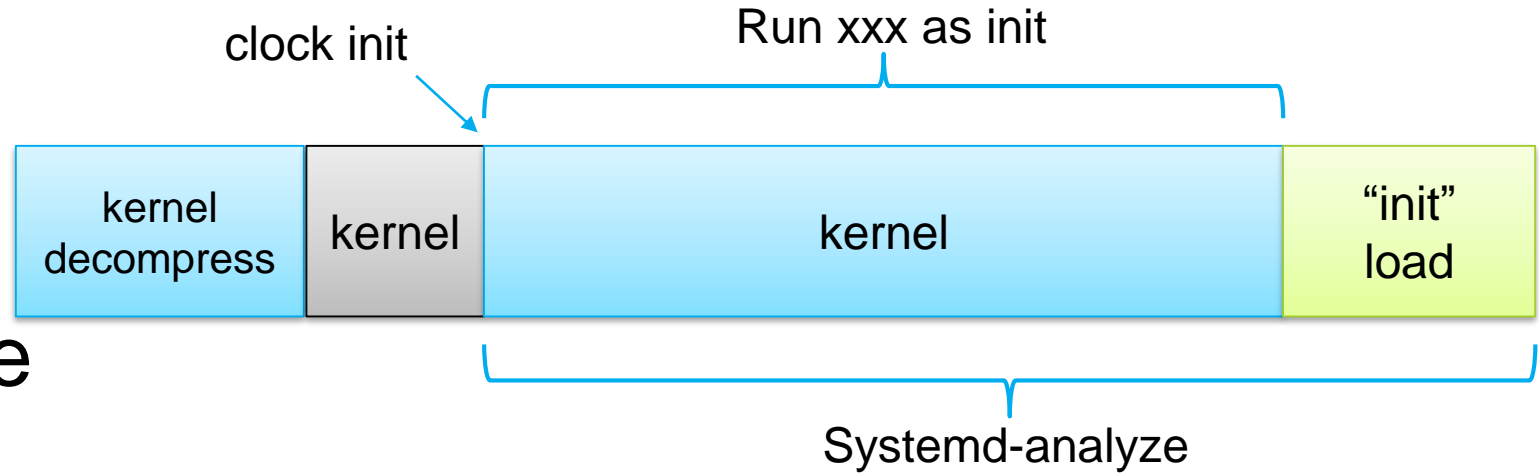
# How To Get Accurate Kernel Boot Time

## ➤ 3 kernel phases

- Decompression
- Dark phase ([ 0.000000])
- Normal phase

## ➤ Check kernel boot time

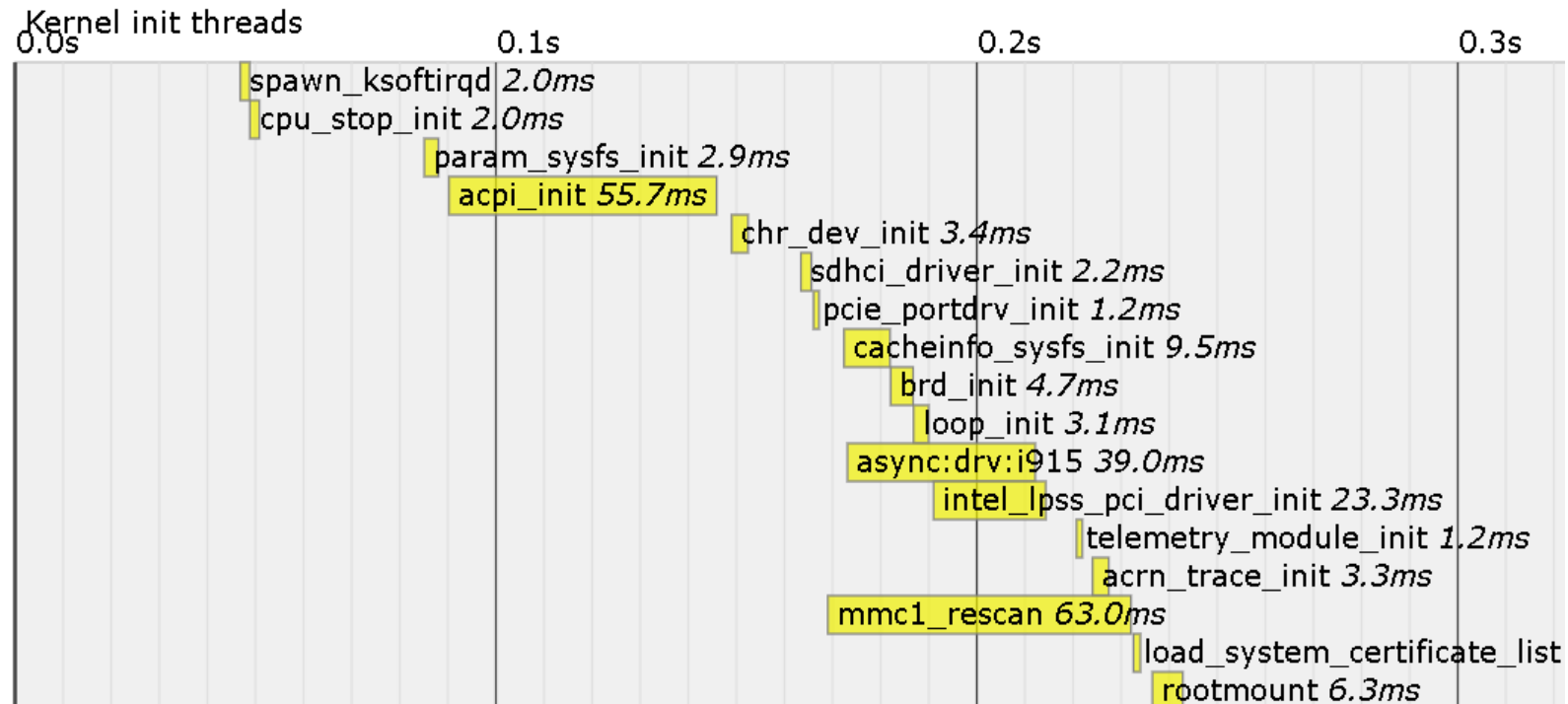
- systemd-analyze
- printk timestamp
- “Run xxx as init process”



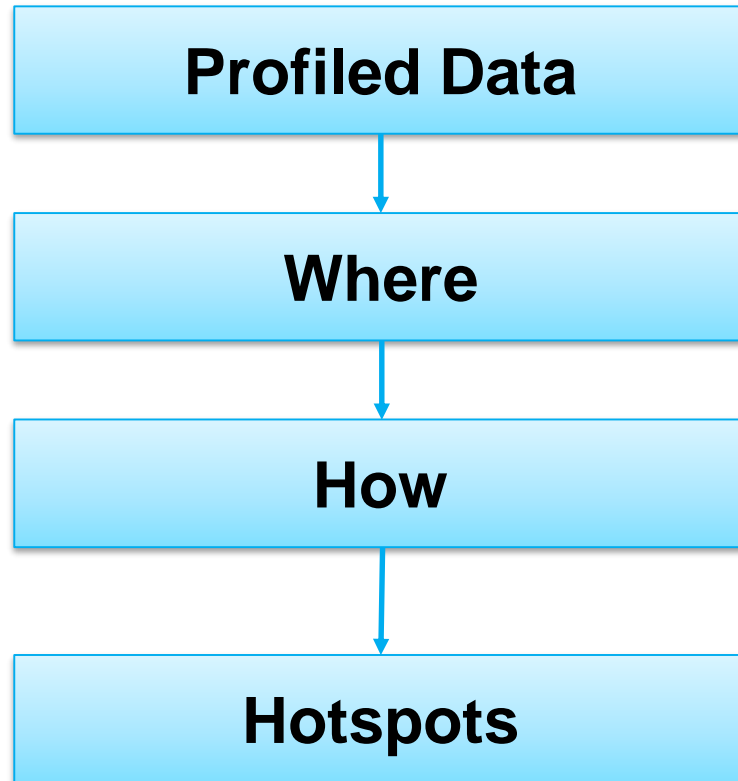
```
root@feng-mrb ~ # systemd-analyze
Startup finished in 318.0ms (kernel) + 8.3910s (userspace) = 8.7091s
graphical.target reached after 775.8ms in userspace
root@feng-mrb ~ # dmesg | grep Run
[ 0.240736] Run /usr/lib/systemd/systemd-bootchart as init process
```

# Profile Tools

- **initcall\_debug**
  - bootchart
- **printk** with absolute timestamp
  - Decompress
  - Dark phase
- Individual dump functions
  - Async debug
  - Not covered by initcall\_debug
- **Ftrace**
  - New boot ftrace event patches



# Analyze



→ Know where every ms is spent

→ Check how and why it takes so much time

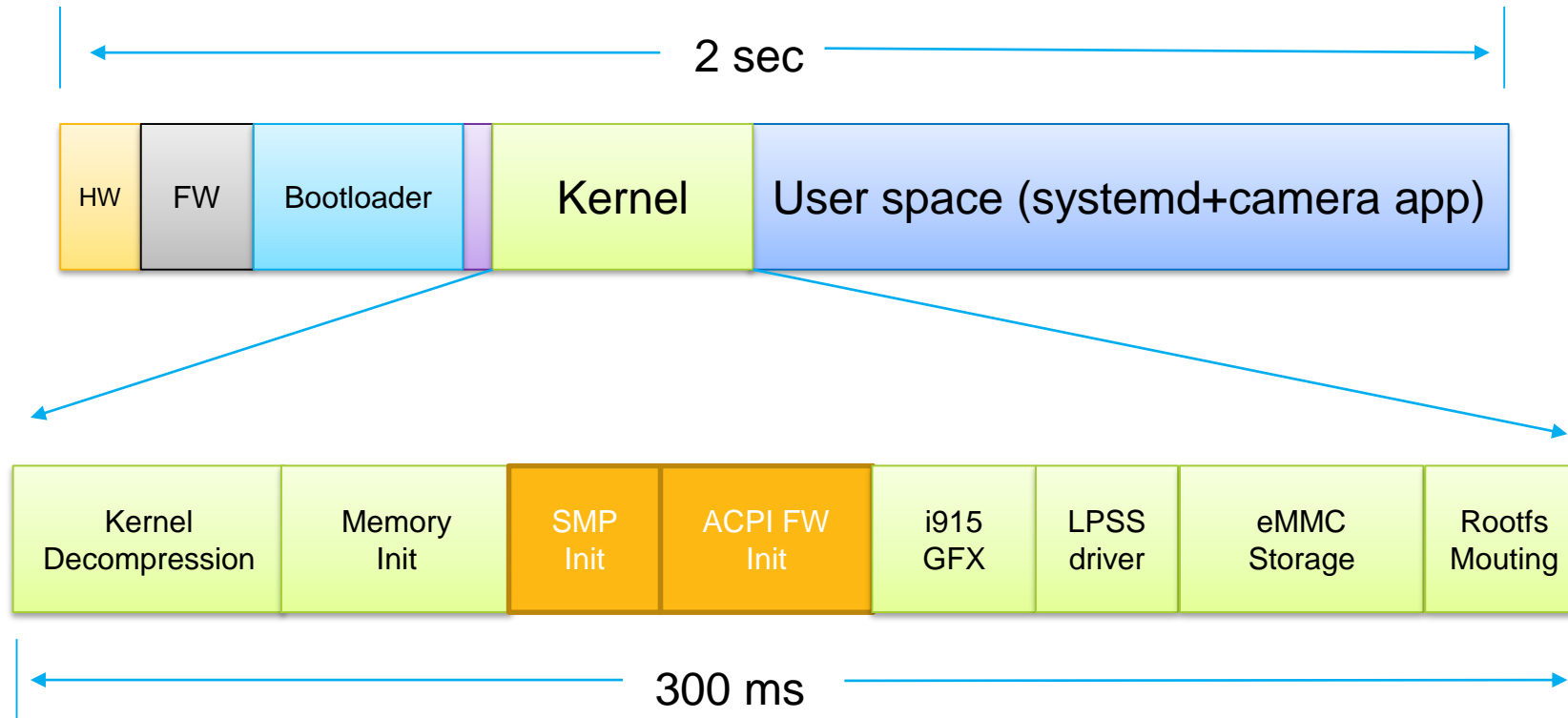
Workaround for specific HW (i915,SDHC)

Unnecessary module/config for product release

Unexpected small module costs lots of time



# How the Boot Time Is Consumed



# Hotspots Overview

- Driver asynchronous probing
- Rootfs mounting
- Memory Init
- Kernel modules and kernel configs
- Graphics (i915)
- Virtualization

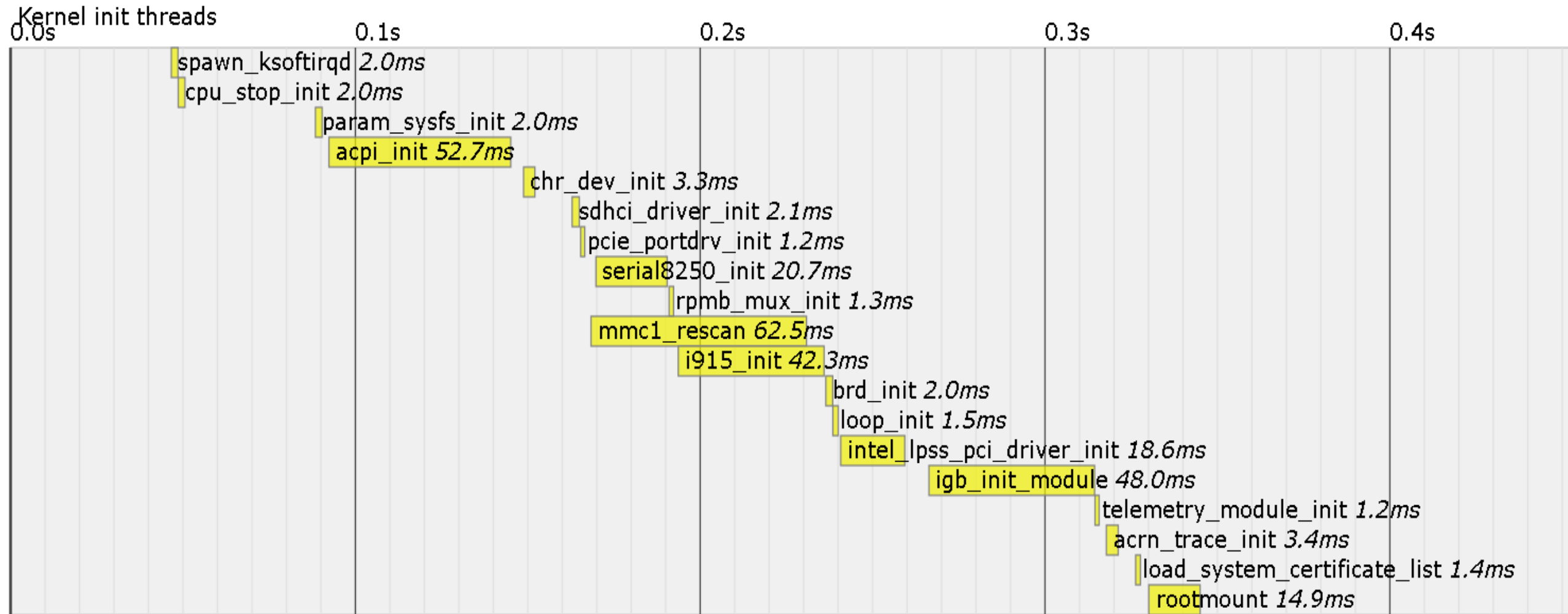
# Boottime Hotspots

Kernel modules	Boottime taken
i915 FB driver init	1+ sec
eDP panel detection	300 ms
ORC unwinder init	300 ms
SATA controller init	150 ms
MEI driver	300 ms
8250 driver IRQ detection	200 ms
Memory Init	150 ms
i915 init	40 ms
acpi init	60 ms
smp multi core init (4C)	30 ms
eMMC driver init	60 ms

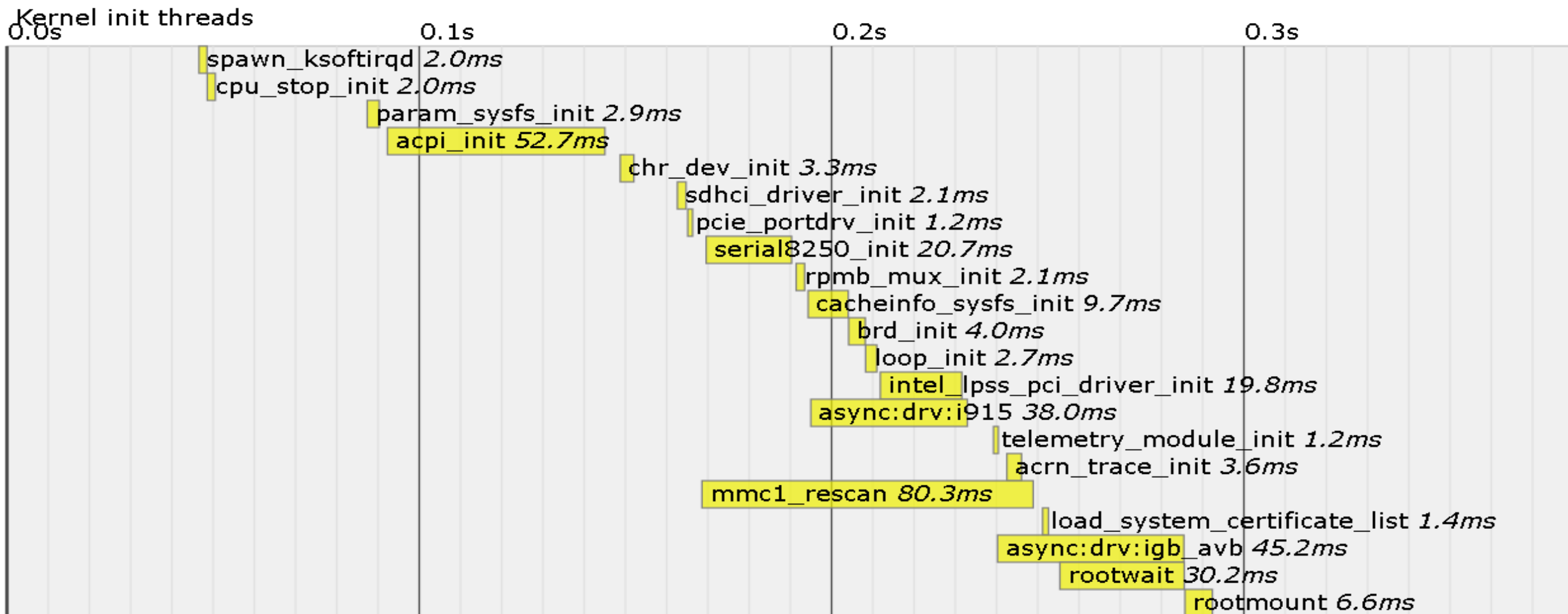
# Too Few Drivers Use Asynchronous Probe

- Driver Async-init framework setup 10 years ago, but rare drivers use it
- Async probe could save a lot of time by making driver init in parallel, like i915, network device
- To enable it, simply set driver's `probe_type` to `PROBE_PREFER_ASYNCHRONOUS`
- Easy to try - `“driver_async_probe=driver1,driver2”` in cmdline

# Original Boot



# Boot With Asynchronous Probe



# RootFS Mounting Matters

- Mostly about storage drivers' efficiency
- SATA driver init takes 100 to 200 ms even without a real disk
- eMMC driver takes 50-100ms
  - Move mmc driver init as early as possible
  - Disable not used host controllers
  - Disable not used protocols (SD/SDIO)
  - Optimize driver's internal hacky busy wait
- Add "rootwait" to cmdline
- Check the hidden asynchronous functions

# Deferred Memory Init

- 8GB RAM's initialization costs 100+ ms
- In early boot phase, we don't need that much memory
- Utilize the memory hotplug feature
  - “mem=4096m” in cmdline to only init 2 GB
  - Use systemd service to add rest memory in parallel



# Highest CPU Frequency Booting (IO too?)

- CPU frequency has huge impact over boot time, especially for those no IO related operations.
- CPU frequency is set by BIOS/FW, before cpufreq subsystem is initialized
- Device Init could also benefit from the frequency changing

# Kernel Modules and Config

- Use loadable module when possible
- Disable all not-necessary modules/drivers
- Disable all debug features for **release** version
- Disable existing but not used HW(like SDHC/SATA controller)
- Kernel size matters

# What Can We Do Next?

- Universality vs Performance
- In-kernel deferred memory init
- SMP initialization for bringing up other Aps
- User space optimization like systemd

# Universality vs Performance

- Driver wants to cover all HWs with one copy of code
- Many long delay in drivers is actually to cover some broken HW
  - i915 driver's 32 times DPCD register read
  - SDHC driver's 10ms power up delay
- Everybody pays because of them
- Can we handle them in a better way?
  - add kernel parameter to tune
  - add quirks

# In-kernel Deferred Memory Init

- User space can initialize majority of the memory with hotplug interface
- Useful for platforms with huge amount of memory
- But it's better to be supported in kernel. Some developers are already working on this

# Parallize SMP Initialization

- It takes about from 6 to 10 ms to bring up one AP, depending on platforms
- It used to be more, has been optimized already
- Currently it is under the CPU hotplug framework, and brought up one by one.

# systemd (user space)

- Systemd is ~1.5MB - the loading time for emmc is 100ms
- Can we use a small lightweight “init” program, which starts target programs in parallel and readahead to preload libraries and executables?

# Credits

Thanks to Bin Yang, Alek Du, Julie Du, Ying Huang, Andi Kleen, Tim Chen, Jianjun Liu and many others for supporting and reviewing



# Q&A

Thank You!