Yami-nabe project report (summary)

October 27, 2011

Embedded Linux Conference Europe 2011

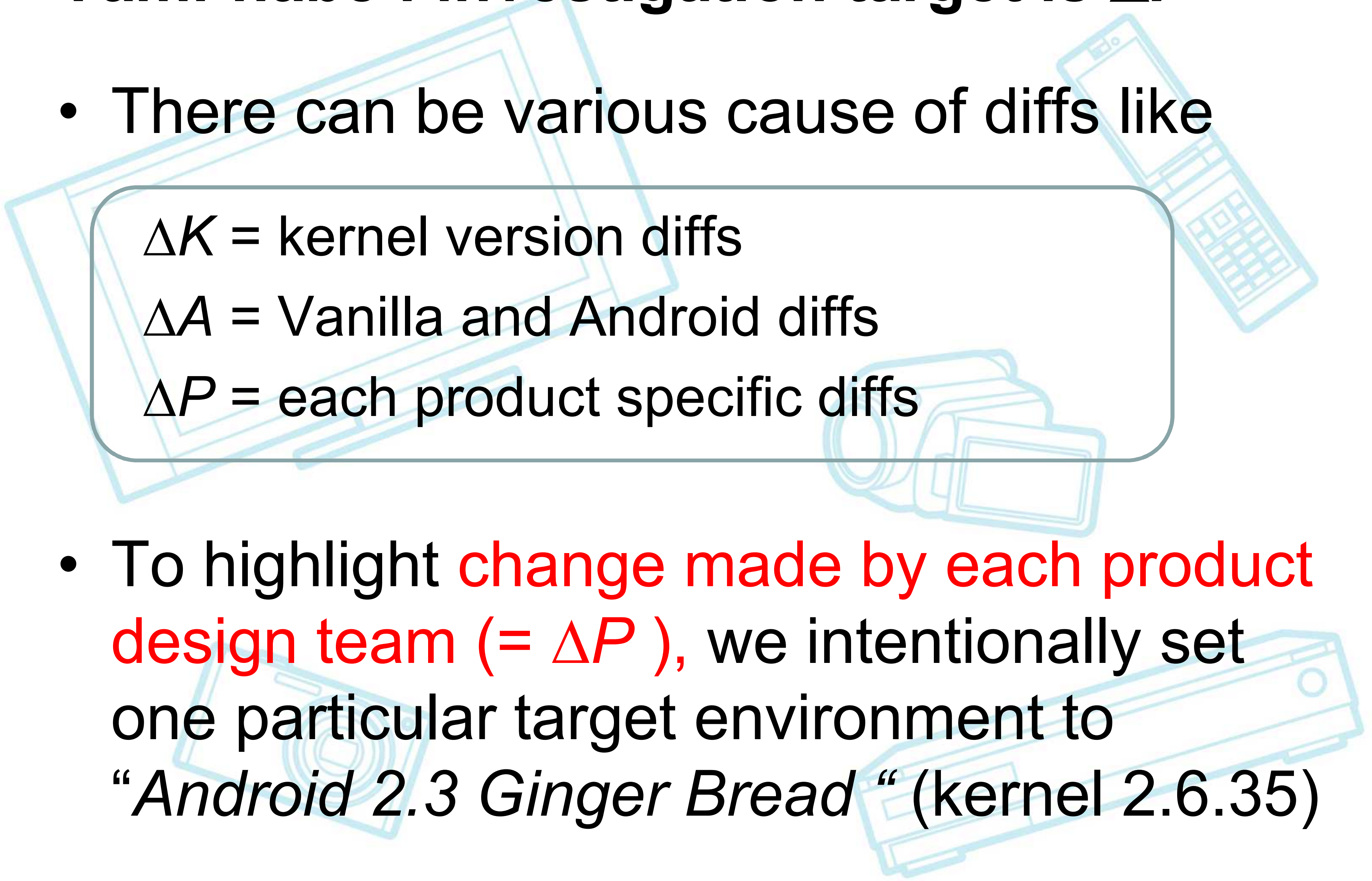
@ Prague Czech Republic

Hisao Munakata, Renesas
CE Workgroup, the Linux Foundation

Yami-nabe : Motivation

- To observe kernel code fragmentation against upstream code in each product from published GPL kernel code. (statistical study)
- Investigate why and how these codes are modified to consider how LTSI can help eliminate unnecessary change or duplication. (diff-code reading study)
- As we do NOT intend to criticize any party, product and chipset vendor names are in chambers (that is the meaning of Yami-nabe)

Yami-nabe : investigation target is ΔP



- There can be various cause of diffs like

ΔK = kernel version diffs

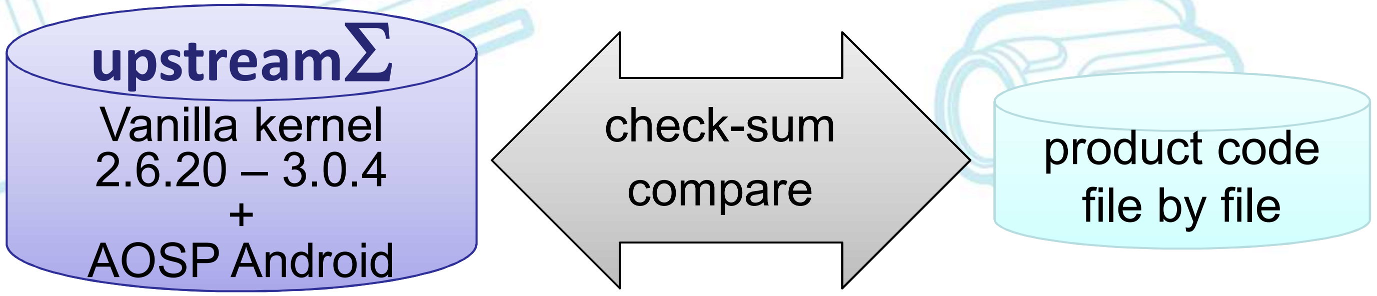
ΔA = Vanilla and Android diffs

ΔP = each product specific diffs

- To highlight change made by each product design team (= ΔP), we intentionally set one particular target environment to "*Android 2.3 Ginger Bread* " (kernel 2.6.35)

Yami-nabe : do check-sum comparison

- To find uniqueness of each product (ΔP) we made *"touchstone code"* from kernel.org (all releases from 2.6.20 and upward until and including 3.0.4) as well as upstream kernel from the Android project. We adopted check_sum match to file compare.



- To eliminate noise we ignored following files
  - an empty file
  - a header file in **include/config**
  - present in the upstream kernel
  - present in upstream Android

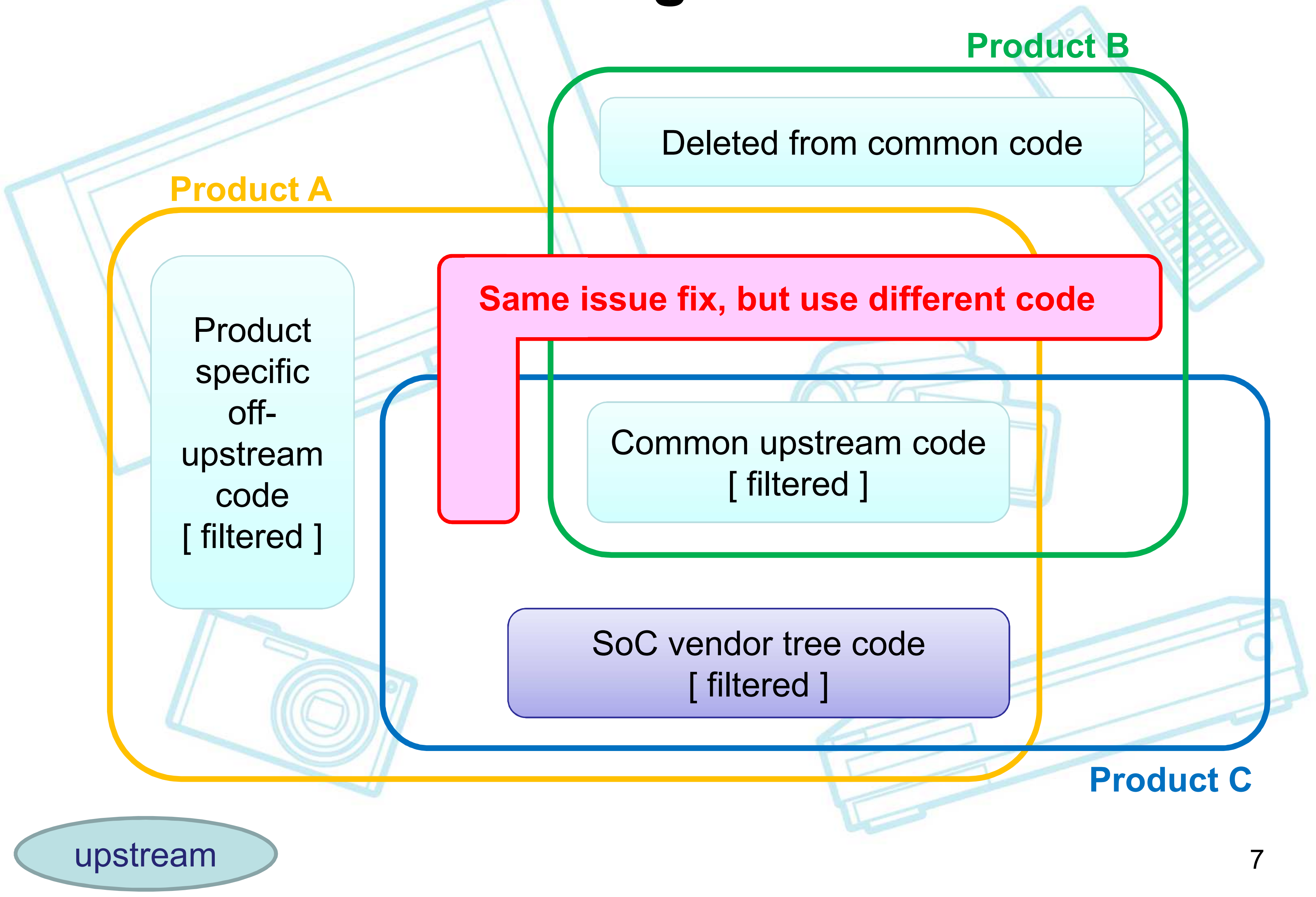
Yami-nabe : uniqueness filtering

- Based on 15 devices comparison, we tried to highlight common duplicated work using following filter to drop inappropriate diffs
  - One of the upstream kernels in the touchstone. (backported code filtering)
  - Code that can only be found in only one device (production specific code filtering)
  - There is just one version, even if it is present in multiple devices. This indicates that the file is already present in some upstream version, like a vendor specific SDK that we have not included in our database. (vendor SDK filtering)

Yami-nabe : outline (per product diffs)

product	kernel version	total file	unique file	uniqueness
1	2.6.35.10	28,012	719	2.6%
2	2.6.35.13	28,179	903	3.2%
3	2.6.35.10	27,814	541	1.9%
4	2.6.35.13	28,201	940	3.3%
5	2.6.35.10	27,848	572	2.1%
6	2.6.35.10	27,872	579	2.1%
7	2.6.35.9	28,005	669	2.4%
8	2.6.35.10	28,144	754	2.7%
9	2.6.35.10	28,241	1,135	4.0%
10	2.6.35.10	27,947	576	2.1%
11	2.6.35.10	27,872	579	2.1%
12	2.6.35.10	28,108	713	2.5%
13	2.6.35.7	29,303	1,890	6.4%
14	2.6.35.7	28,413	1,882	6.6%
15	2.6.35.7	28,318	1,264	4.5%

Yami-nabe : diff categorization


Product B
Deleted from common code
Product A
Same issue fix, but use different code
Product specific off-upstream code [filtered]
Common upstream code [filtered]
SoC vendor tree code [filtered]
Product C
upstream

Yami-nabe : filtering result

Most of the duplicated effort was found in the drivers/ directory (323 files) and the arch/arm/mach-msm/ directory (283 files).

In total 818 files are dropped with these filter. This is significantly lower than the amount of unique files in for example #13 or #15. This is because in these devices there are quite a few files that are specific for that particular device. #15 for example has its own video driver. #13 has several drivers (gpu, wireless network) that are just for that device.

Yami-nabe : Top 10 different versions files

#	frequency	path
1	13/15	drivers/usb/gadget/android.c
2	12/15	drivers/usb/gadget/composite.c
3	12/15	drivers/mmc/core/core.c
4	9/15	drivers/video/msm/msm fb.c
5	9/15	drivers/video/msm/mdp.c
6	9/15	drivers/usb/gadget/f mass storage.c
7	9/15	drivers/mmc/core/sdio.c
8	9/15	drivers/mmc/core/mmc.c
9	9/15	drivers/mmc/card/block.c
10	9/15	drivers/cpufreq/cpufreq ondemand.c

Almost every device has a slightly different version for the USB gadget driver, the MMC driver and a video driver.

Yami-nabe : code reading [usb]

#	frequency	path
1	13/15	drivers/usb/gadget/android.c
2	12/15	drivers/usb/gadget/composite.c
3	9/15	drivers/usb/gadget/f mass storage.c
4	8/15	drivers/usb/gadget/msm72k udc.c
5	5/15	drivers/usb/gadget/u serial.c
6	5/15	drivers/usb/gadget/f rndis.c
7	4/15	include/linux/usb/gadget.h
8	4/15	drivers/usb/gadget/u serial.h
9	4/15	drivers/usb/gadget/u ether.c
10	4/15	drivers/usb/gadget/storage common.c
11	4/15	drivers/usb/gadget/gadget chips.h
12	4/15	drivers/usb/gadget/f serial.c
13	4/15	drivers/usb/gadget/f diag.c
14	4/15	drivers/usb/gadget/f adb.c

Yami-nabe : code reading [usb]

- drivers/usb/gadget/android.c
  - Power management enhancement while AC power source is connected,
  - The technique used to reset a composite device. Because vendors ship different USB Gadget configurations, this may be classified as a bug fix or workaround or a feature, depending on the production team's point-of view.
- drivers/usb/gadget/composite.c
  - this file was also extensively rewritten, in this case by another different product team. Doubtless they have introduced new functionality and solved problems in a vendor-specific manner, but the divergence from upstream is enormous
- drivers/usb/gadget/msm72k_udc.c
  - This file was added by several production teams relative to the Gingerbread baseline. In particular, two at the same vendor introduced this file but did so differently. This is either a missing correction on the part of the second team or a bug fix that this team applied and that their peer group did not.

Yami-nabe : code reading [mmc]

#	frequency	path
1	12/15	drivers/mmc/core/core.c
2	9/15	drivers/mmc/core/sdio.c
3	9/15	drivers/mmc/core/mmc.c
4	9/15	drivers/mmc/card/block.c
5	8/15	drivers/mmc/host/msm sdcc.c
6	6/15	drivers/mmc/host/msm sdcc.h
7	6/15	drivers/mmc/core/sd.c
8	5/15	drivers/mmc/card/queue.c
9	3/15	drivers/mmc/core/host.c
10	3/15	drivers/mmc/core/sdio cis.c
11	3/15	drivers/mmc/core/mmc ops.c
12	3/15	drivers/mmc/core/core.h
13	3/15	drivers/mmc/core/bus.c
14	2/15	drivers/mmc/host/omap hsmmc.c

Yami-nabe : code reading [mmc]

- drivers/mmc/core/core.c
  - add a function call to find out if device is in a resume. Two of the kernels neglect to check the power status of the device before making this call. This change began to propagate upstream but was subsequently replaced in upstream with an entirely different implementation.
  - Upstream Gingerbread has partial support for bus resume, but several vendors has reverted that support back to a much earlier implementation! Furthermore, of the five vendors that made this change, there are varying levels of how much old code was replaced back into this file.
  - Significant locking behavior likewise removed and replaced with older code or not at all. One vendor reverted upstream's newer use of a wait lock and replaced it with their own implementation.
  - Related power management code in Gingerbread (and also now in upstream Linux mainline) but has been outright removed from one device and partially removed from two others.

Yami-nabe : code reading [touch panel]

#	frequency	path
1	7 /15	drivers/input/touchscreen/atmel.c
2	4 /15	drivers/input/touchscreen/cy8c tma ts.c
3	2 /15	drivers/input/touchscreen/himax8250.c
4	2 /15	drivers/input/touchscreen/atmel 224e.c

All 15 products modified their touch driver

[input (exclude touch panel)]

#	frequency	path
1	6/15	drivers/input/misc/gpio input.c
2	4/15	drivers/input/misc/gpio switch.c
3	4/15	drivers/input/misc/cm3602 lightsensor microp.c
4	4/15	drivers/input/input.c
5	3/15	drivers/input/misc/gpio matrix.c
6	3/15	drivers/input/keyreset.c
7	3/15	drivers/input/evdev.c

Yami-nabe : code reading [touch panel]

- arch/arm/mach-msm/smd.c
  - This code is called from numerous places in this vendor's drivers code. If cleaned up, this change would be eligible for submission to upstream. (later work established that a later variation of this change was published by the vendor to the arch-msm tree but has not gone anywhere toward Linux mainline).
- drivers/input/touchscreen/tsc2007.c
  - The bulk of the changes in this file regard implementing support for early suspend and further refining power management. IRQ handling fix request_irq handling. Not accepted upstream; another patch was instead. The upstream replacement appears to have replaced and/or superseded this fix in most of the other kernels.
- drivers/input/touchscreen/msm_ts.c
  - Although this file shows up as an added file when compared against Gingerbread, comparing this file between the devices makes subtle differences to this file apparent. All of these changes can be explained by each vendor fetching the change sets on different dates, or from slightly different branches.

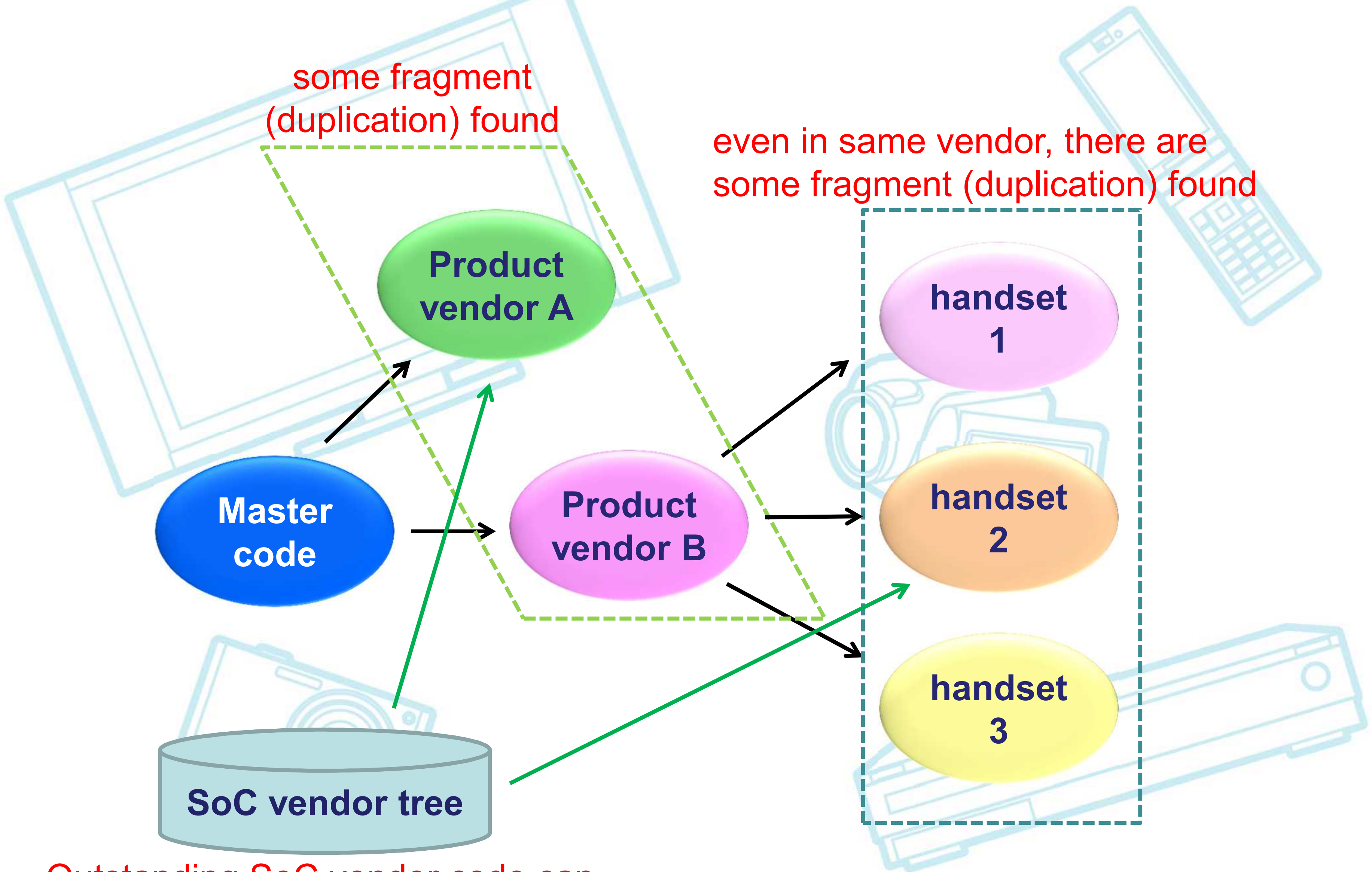
Yami-nabe : code reading [core kernel]

#	frequency	path
1	6/15	kernel/timer.c
2	6/15	kernel/sched.c
3	6/15	kernel/power/wakelock.c
4	6/15	kernel/power/earlysuspend.c
5	5/15	kernel/sys.c
6	5/15	kernel/printk.c
7	5/15	kernel/power/suspend.c
8	4/15	kernel/power/power.h
9	4/15	kernel/power/main.c
10	4/15	kernel/panic.c
11	4/15	kernel/irq/pm.c
12	3/15	kernel/softirq.c
13	3/15	kernel/sched fair.c
14	3/15	kernel/pm qos params.c

Yami-nabe : code reading [core kernel 2]

#	frequency	path
15	3/15	kernel/kthread.c
16	3/15	kernel/irq/handle.c
17	3/15	kernel/exit.c
18	2/15	kernel/workqueue.c
19	2/15	kernel/time/timekeeping.c
20	2/15	kernel/time/tick-sched.c
21	2/15	kernel/stop machine.c
22	2/15	kernel/semaphore.c
23	2/15	kernel/resource.c
24	2/15	kernel/power/process.c
25	2/15	kernel/pid.c
26	2/15	kernel/irq/chip.c
27	2/15	kernel/fork.c
28	2/15	kernel/cpuset.c

Yami-nabe : observations



Outstanding SoC vendor code can be seen in various product kernel

Yami-nabe : project outcome (summary)

- We have investigated 15 products those adopt the same code base, and we confirmed duplicated effort mostly in driver and arch code as we assumed.

- The majority of these fragments seems to come from SoC vendor code, however each code are adjusted to {fix, improve, coordinate} by set vendors as well.

- We tried to read their intention for such modification from USB, MMC and Touch driver code, but it is not straightforward to know the real reason for each change from just code without git log information.

Yami-nabe : lesson learned

- If we can provide LTSI kernel that includes latest driver fix, that would reduce driver code fragment.
  (LSTI backport team may help)

- If LTSI can help upstream SoC vendor tree code and if they can be a part of LTSI kernel, that would be beneficial to both SoC and handset.
  (LTSI staging and upstream support may help)

- Ideally each handset (or other) product producer will write a patch to share their issue and fix that could eliminate existing driver code fragment.
  (LSTI upstream support , consulting may help)

LTSI could be a solution to eliminate kernel code fragmentation seen in consumer embedded world.