Introduction to LTSI for the Industry

(Long Term Support Initiative)

October 27, 2011

Embedded Linux Conference Europe 2011

@Prague Czech Republic

Tsugikazu Shibata, NEC

CE Workgroup, the Linux Foundation

CONTENTS

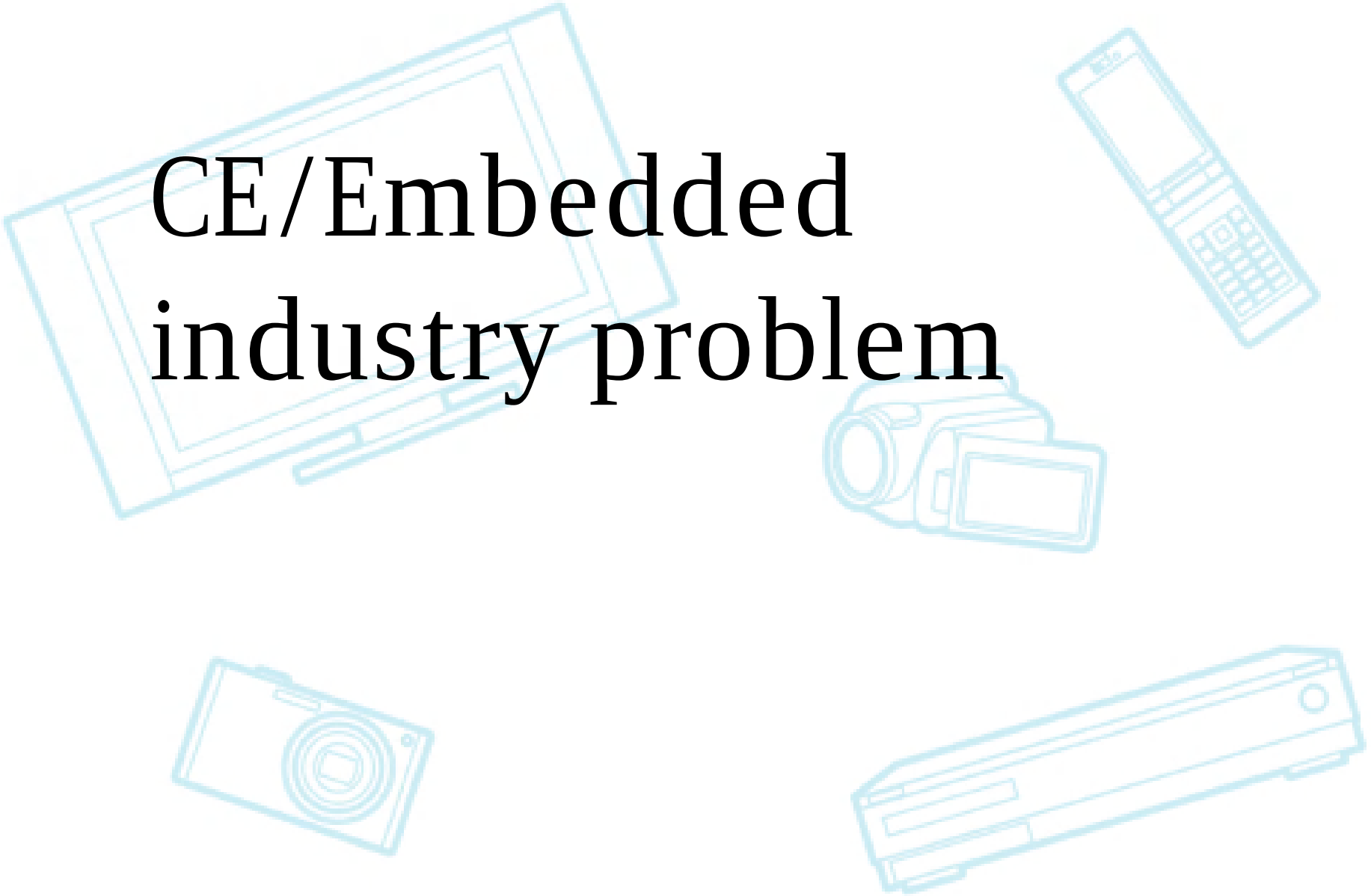
- Who we are?
- CE/Embedded Industry Problem
- LTSI Project overview
  - Long term @ kernel.org
  - LTS Industry Tree
  - Upstream support
  - Pilot project Yami-nabe
- Conclusion

Who we are?

- Member of Consumer Electronics Working Group “CEWG” in the Linux Foundation
  - HP, Hitachi, IBM, Intel. LG, NEC, Panasonic, Renesas, Samsung, Sony, Toshiba
- We have discussed current Consumer Electronics industry’s problem

- We hope to solve such problem working with community

CE/Embedded industry problem

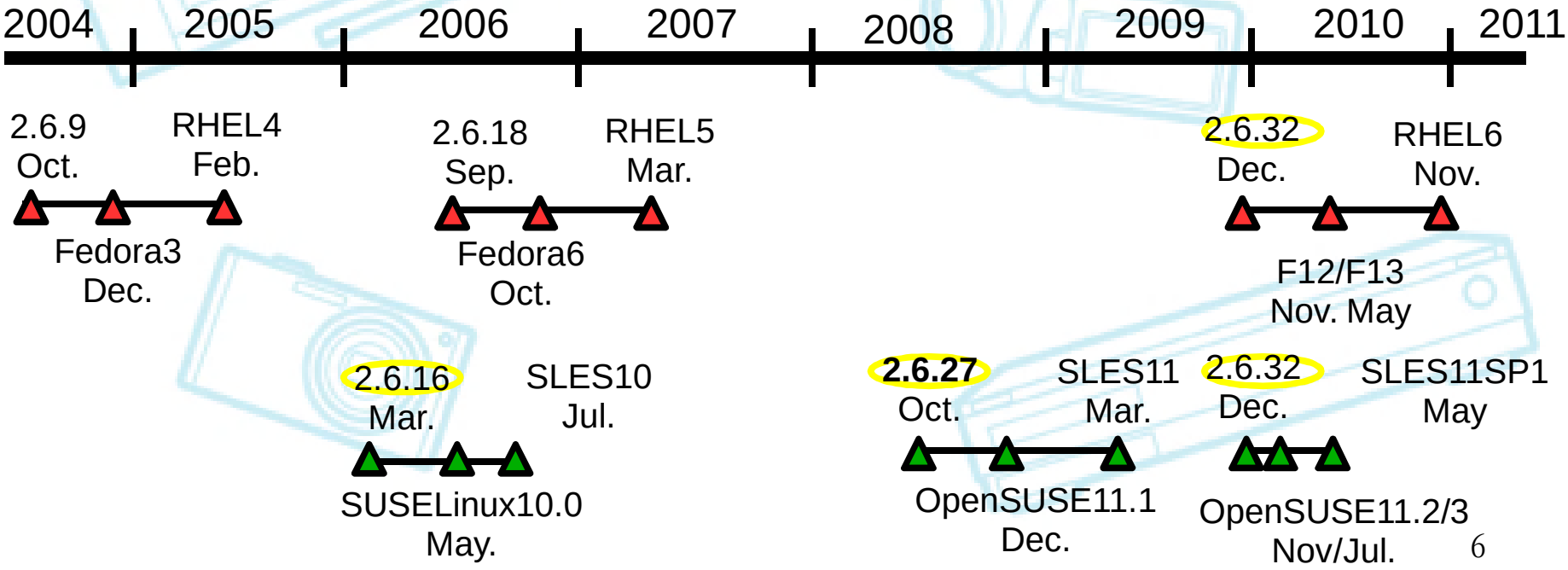


Problem 1: Shorter product lifetime compared with Enterprise product

- Lifetime of the Enterprise industry products are about 5 to 10 years
  - Latest RHEL(Red Hat Enterprise Linux) is using 2.6.32.x kernel which is maintained by the community as long term version
  - We expect that RHEL will use same kernel version in next 3-4 years. So, next long term kernel needs to establish 3-4 years later
- Lifetime of the Consumer products are 1 to 3 years
  - CE industry need to refresh kernel every year but there are no community long-term infrastructure.
  - 2.6.35 had been established as long term last year and are using in lots of embedded products. But NO follow on long term version was discussed

Enterprise Linux distributions and kernel versions

- RedHat and SUSE used different kernel version before
- 2.6.16 and 2.6.27 was defined as long term version in the community and SUSE was use it
- Since 2009, Both distribution used same kernel version 2.6.32



Problem 2: No common ground for embedded kernel

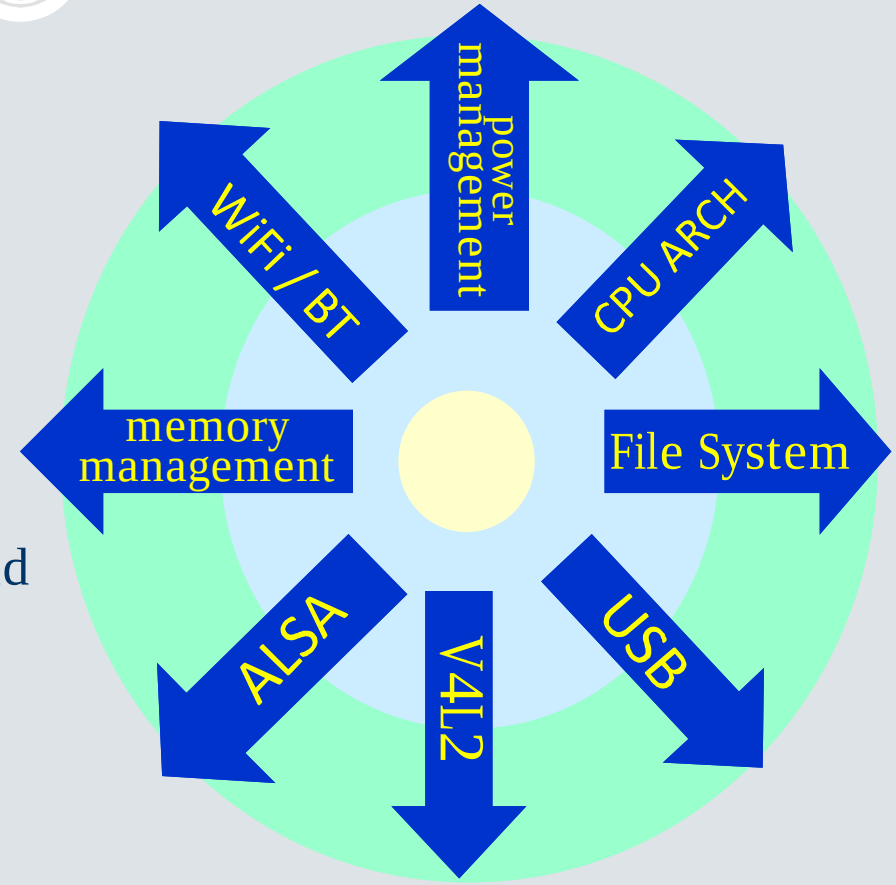
- Android and MeeGo are releasing every 6 month with latest kernel (to provide the innovative features)
  - Semiconductor vendors are providing BSP with kernel every time without support
  - Every Manufacturers are using the BSP with the kernel owning their kernel support
- Manufacturers and Semis need some time for system level verification
- Industry wants to reuse same kernel for a few product generation
- We need an industry managed kernel
  - With some feature back ported from latest upstream kernel
  - With include Semis supplied patch
  - With common QA activity

Problem 3: Upstream patch submission from embedded is still very inactive

- Engineers in Embedded production team pay huge attention to Linux, especially device driver code quality
- And they likely modify device driver code to improve system stability and/or performance
- Therefore, embedded industry engineers own some good and important patches in house, however patch submission from these engineers are very low
- As these codes are not mainlined, they need to apply same enhancement when they adopt new kernel for their products

Upstream principals = divergence (generalize)

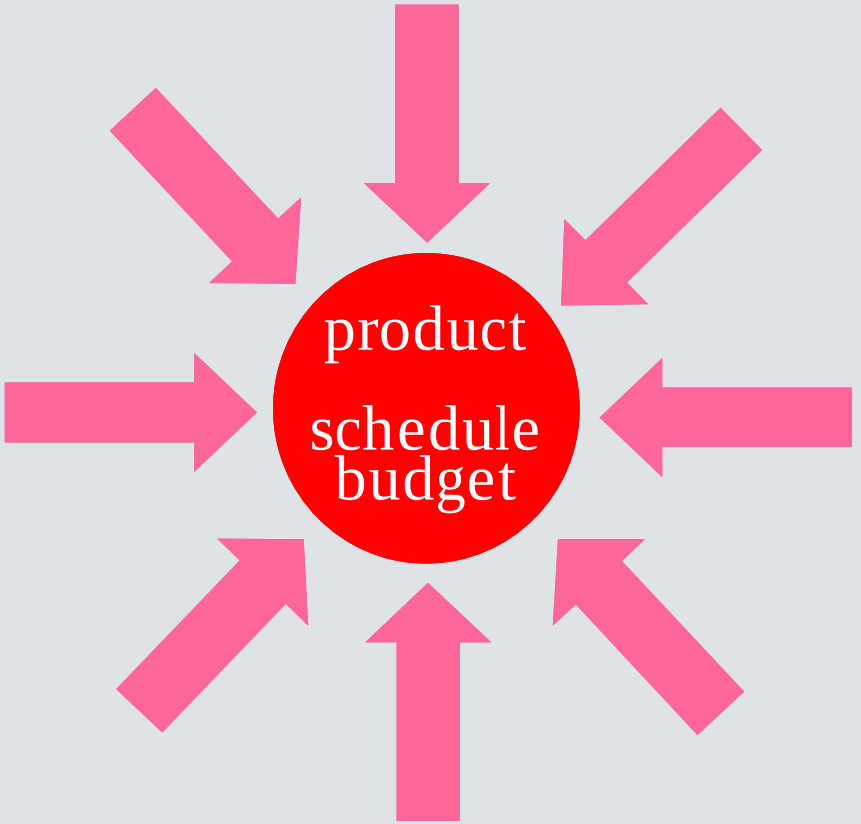
- Think sustainable evolution
  - random technical improve
  - no specific target products
  - allow diversity
- Ever lasting development
  - no specific due date
  - Think for the better future
  - incremental improve
  - moving target depends on demand
- Fair governance
  - Completely open
  - purely technical (for best)
  - volunteer contribution basis



Upstream guys work for unified better future for all

Production principals = convergence (specialize)

- Clear production goal
  - strict release due date
  - severe performance target
  - high cost pressure
- One shot development
  - allow interim solution
  - average skilled engineer
  - relatively large team
- Quality requirement
  - product liability demand
  - limited use case
  - reset is not allowed



Industry developer work for their current particular product

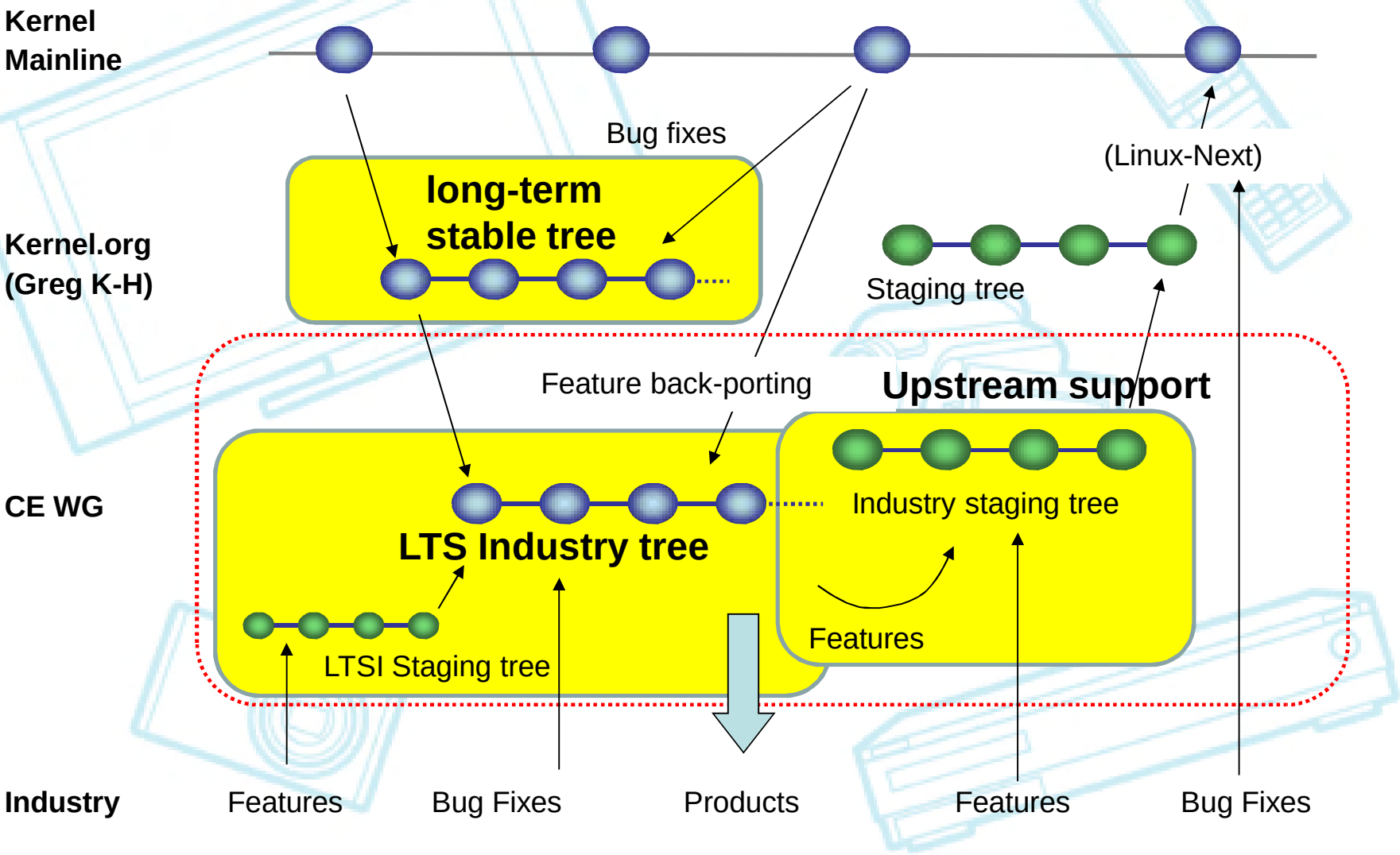
Project requirements

- We need Long-term community based Linux kernel to cover embedded industries' life time
- We need Industry managed kernel tree as a common ground for Embedded industry
- We need some mechanism to support the upstream activities for Embedded engineers

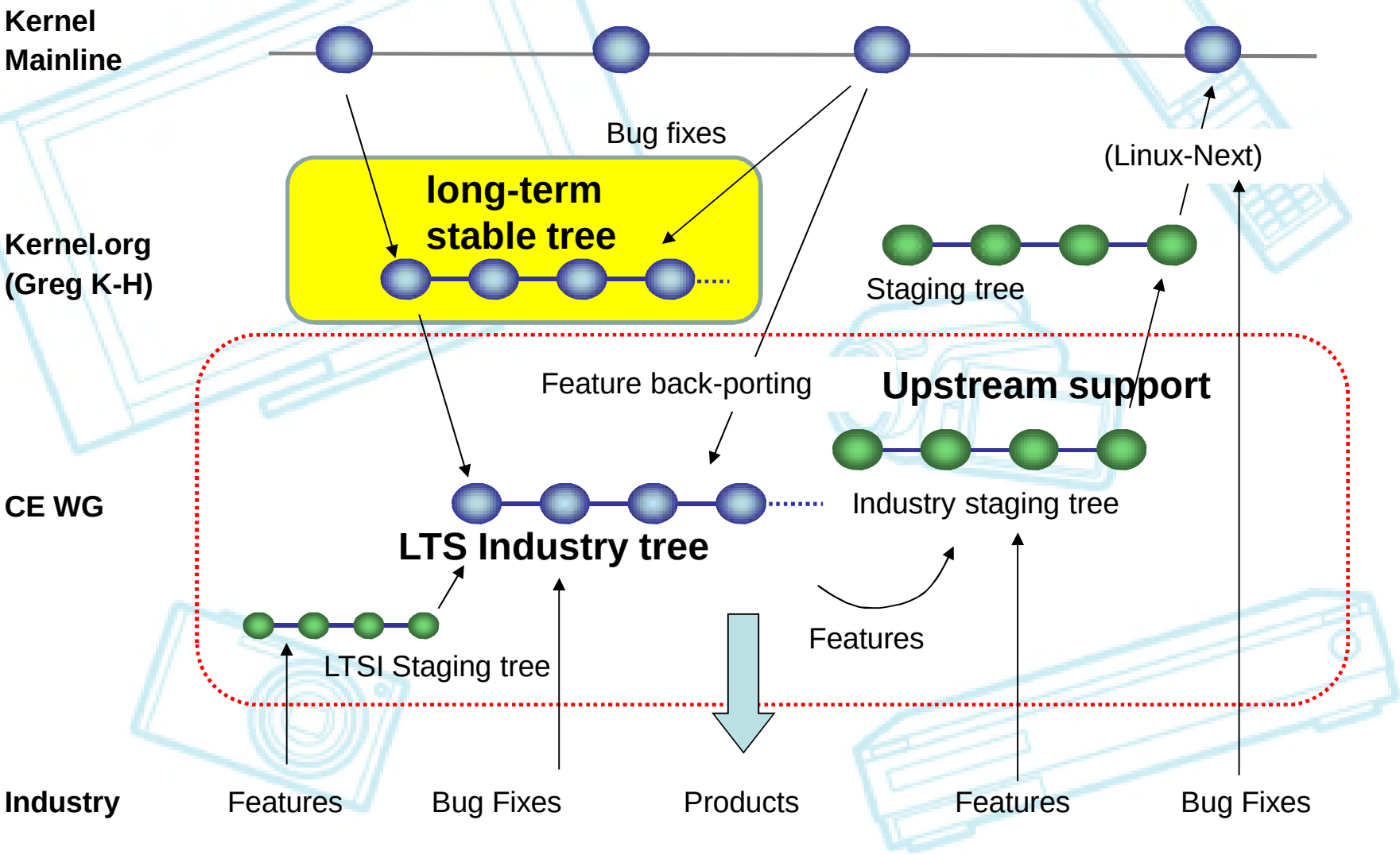
LTSI
Project Overview

LTSI project overview

- **Project consists of tree part**

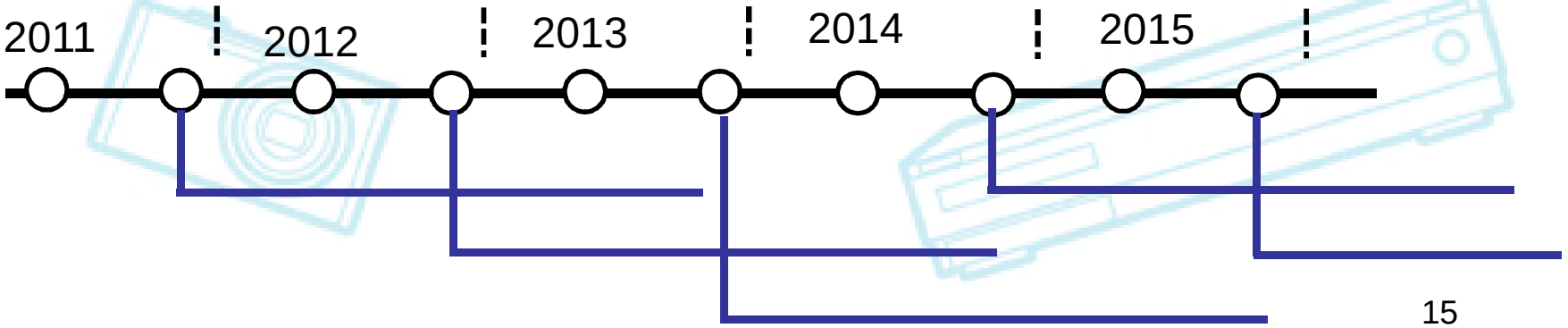


LTSI project Over view:
1. Long-term Stable tree @ kernel.org

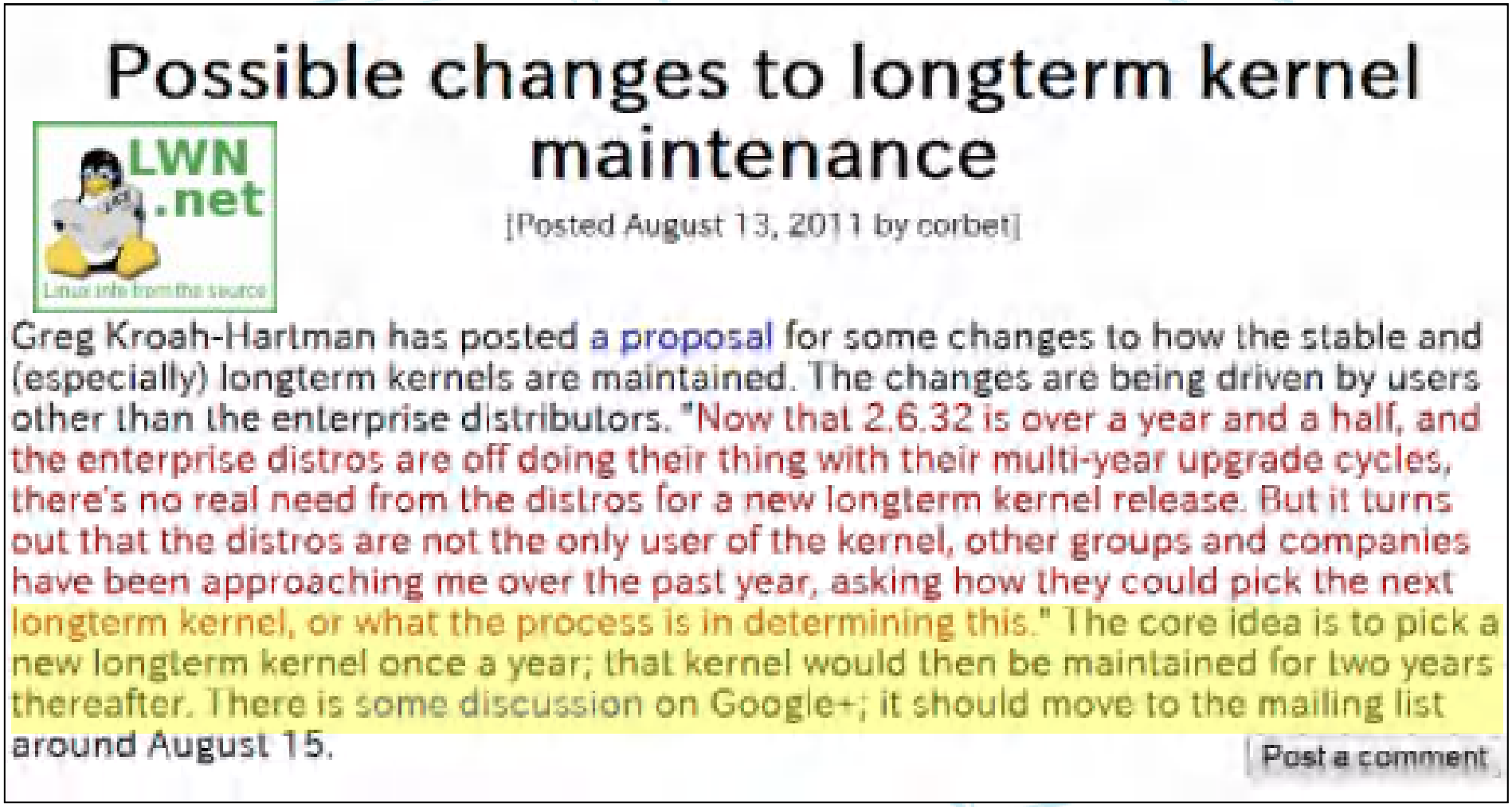


Long term Stable tree @ kernel.org

- Establish a version on a mostly yearly basis
- Maintain two years long and at most 2 versions will be maintained
- Include only bug fixes and security fixes
- Starts regular operation from 3.0 kernel
- Maintained by Greg KH



New community kernel longterm maintenance scheme proposed by Greg Kroah-Hartman



Possible changes to longterm kernel maintenance

[Posted August 13, 2011 by corbet]

Greg Kroah-Hartman has posted a proposal for some changes to how the stable and (especially) longterm kernels are maintained. The changes are being driven by users other than the enterprise distributors. "Now that 2.6.32 is over a year and a half, and the enterprise distros are off doing their thing with their multi-year upgrade cycles, there's no real need from the distros for a new longterm kernel release. But it turns out that the distros are not the only user of the kernel, other groups and companies have been approaching me over the past year, asking how they could pick the next longterm kernel, or what the process is in determining this." The core idea is to pick a new longterm kernel once a year; that kernel would then be maintained for two years thereafter. There is some discussion on Google+; it should move to the mailing list around August 15.

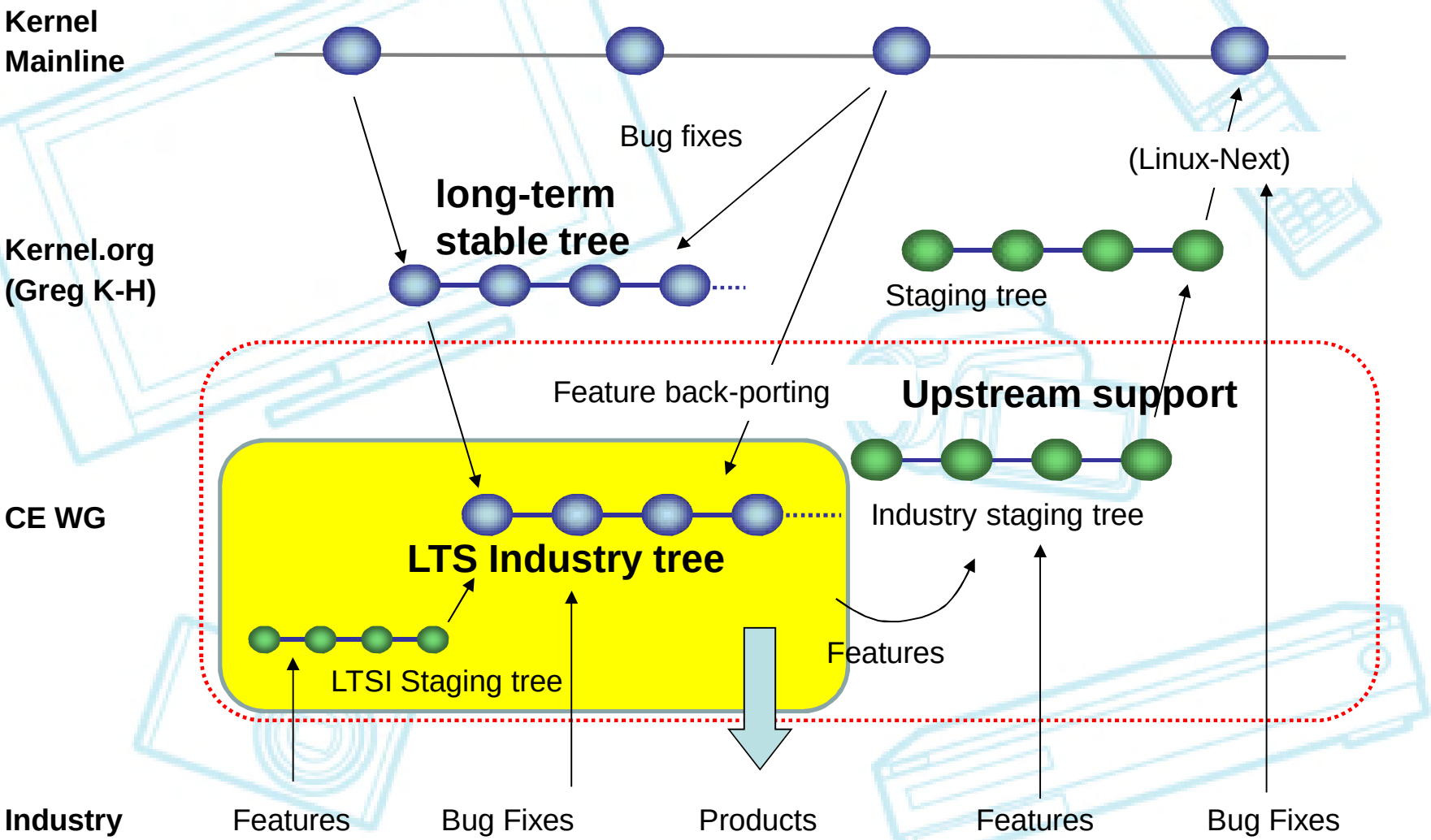
Post a comment

http://lwn.net/Articles/454886/

How to decide the version of long-term supported kernel ?

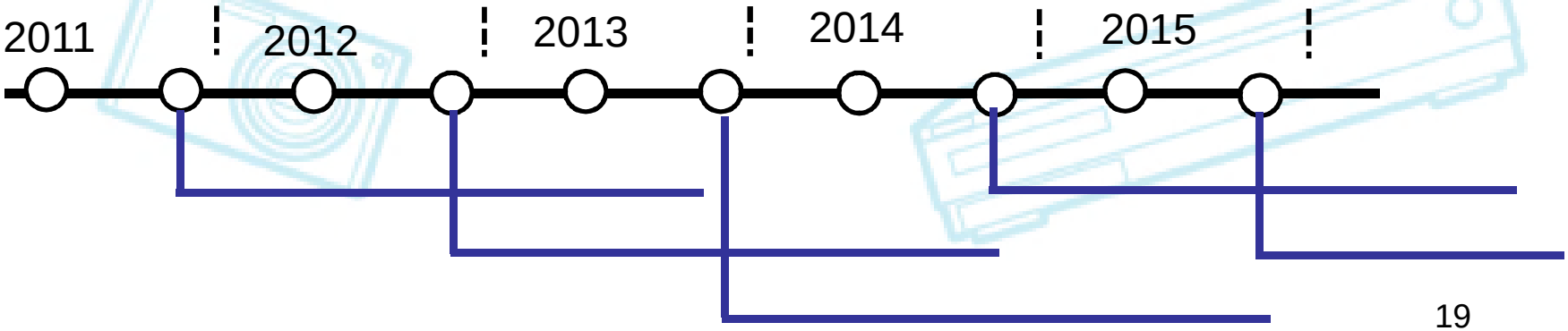
- There are no concrete process to decide what version would be supported as long term
- We would propose:
  - Industry Advisory Board (IAB) will be the place to discuss what version would be the long term supported and suggest it
  - Member of IAB should be key industry distros and/or stake holders

LTSI project Over view:
2. LTS Industry tree

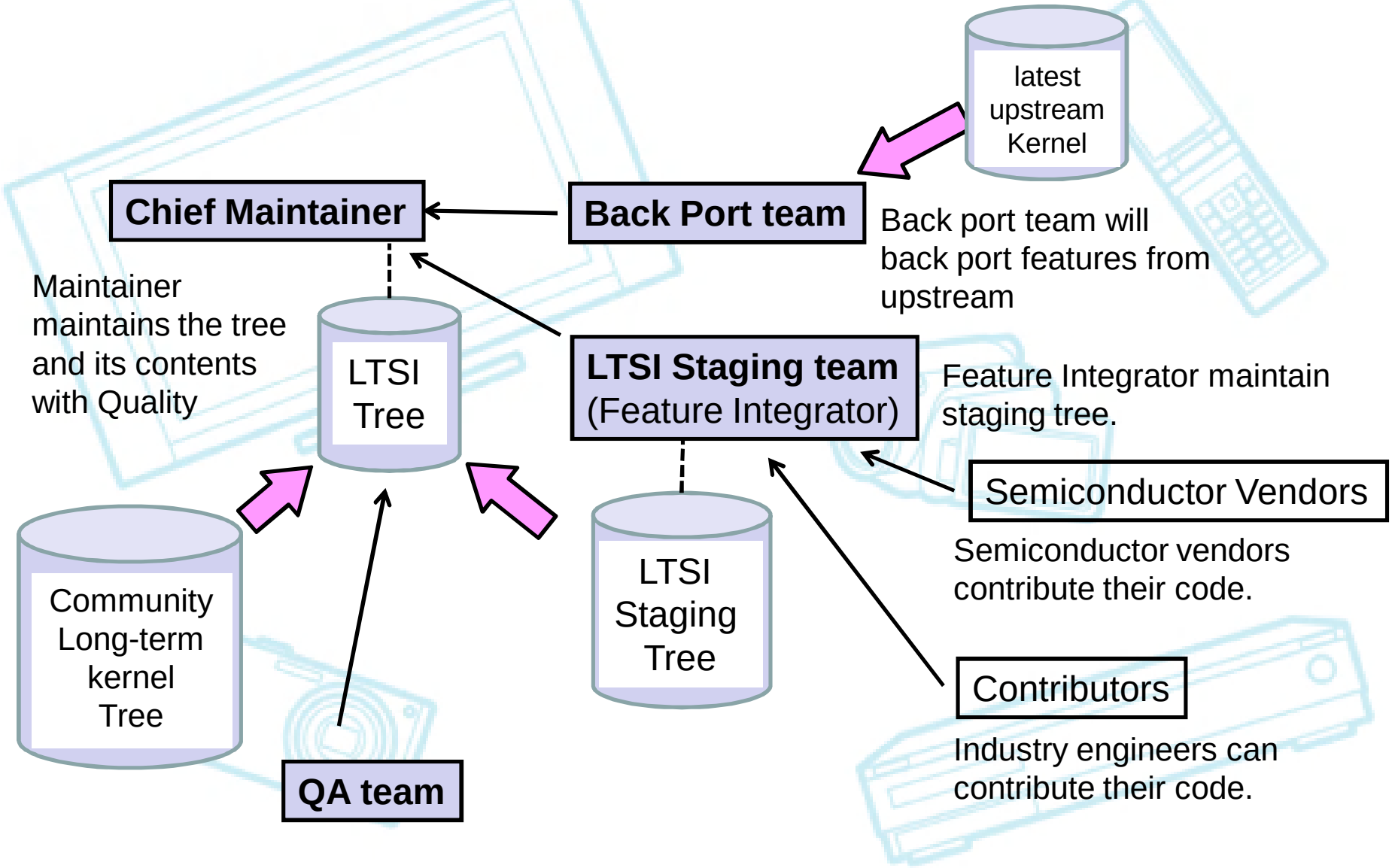


LTS Industry tree

- This is the Industry's managed tree (not for the community)
- Maintained by CEWG who will publish it
- Based on the long-term stable kernel tree
  - Refresh its version every year and maintain at most 2 versions in 2 years
- Include back port features from upstream
- Include Semiconductor vendors patches
- Include important features for industry
- Include mainlining services for industry engineers



Maintain LTS Industry tree



Feature back port candidates to LTSI

- New device drivers (in latest upstream) *1
  - Common device drivers for embedded (mainlined in the latest version)
- Part of common kernel functions*2
  - Memory / power management
  - Real-time enhancement
  - Boot related (boot method, boot time .,etc)
  - Size related (Linux-tiny etc)
  - Performance improvements
- Platform support features in latest upstream*3

1: Target driver list will be maintained by the project

2: Not all fundamental change can be back ported due to its complexity. Framework change might not be applicable as it require huge code change

3: Latest SoC/BSP supported features may be back ported

How to add new drivers/features to LTSI

1) Submit the code to current upstream
2) Get community review and feedback
3) Adjust code based on 2) to fit upstream
4) Code queued (or merged)

5) Then back port that code to LTS Industry tree.
   We expect test code will come with code

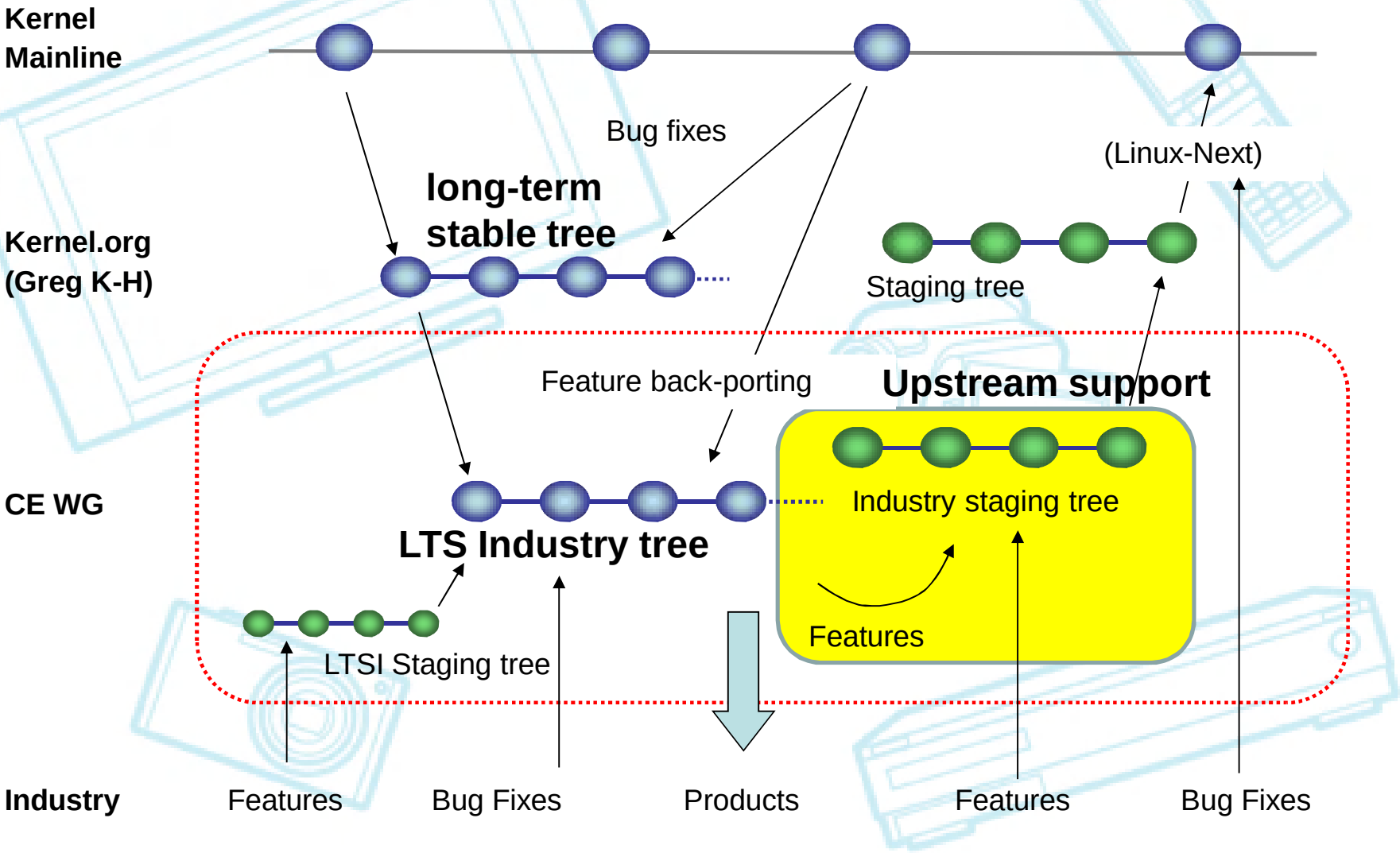
Or

6) Queue to the LTSI staging tree directly in special case
7) After reviewed in LTSI staging tree then merge into LTS Industry tree

LTSI comes with test

- To secure back port code quality, we want to conduct system test in LTSI
- We want to discuss with other parties to share test processes and results
- Activity of QA team in LTSI :
  - Define QA process, design and develop test system
  - Design and develop test scenario with other parties
  - Report the test result
  - Project participants can share this test framework
- Testing device drivers are always problem
  - LTSI QA team would like to;
  - Work with new driver author to create test scenario
  - Help device driver developer to cover full function could be implemented and tested for industry usage

LTSI project over view:
3. Upstream support



Upstream support in LTSI

Key activities are:

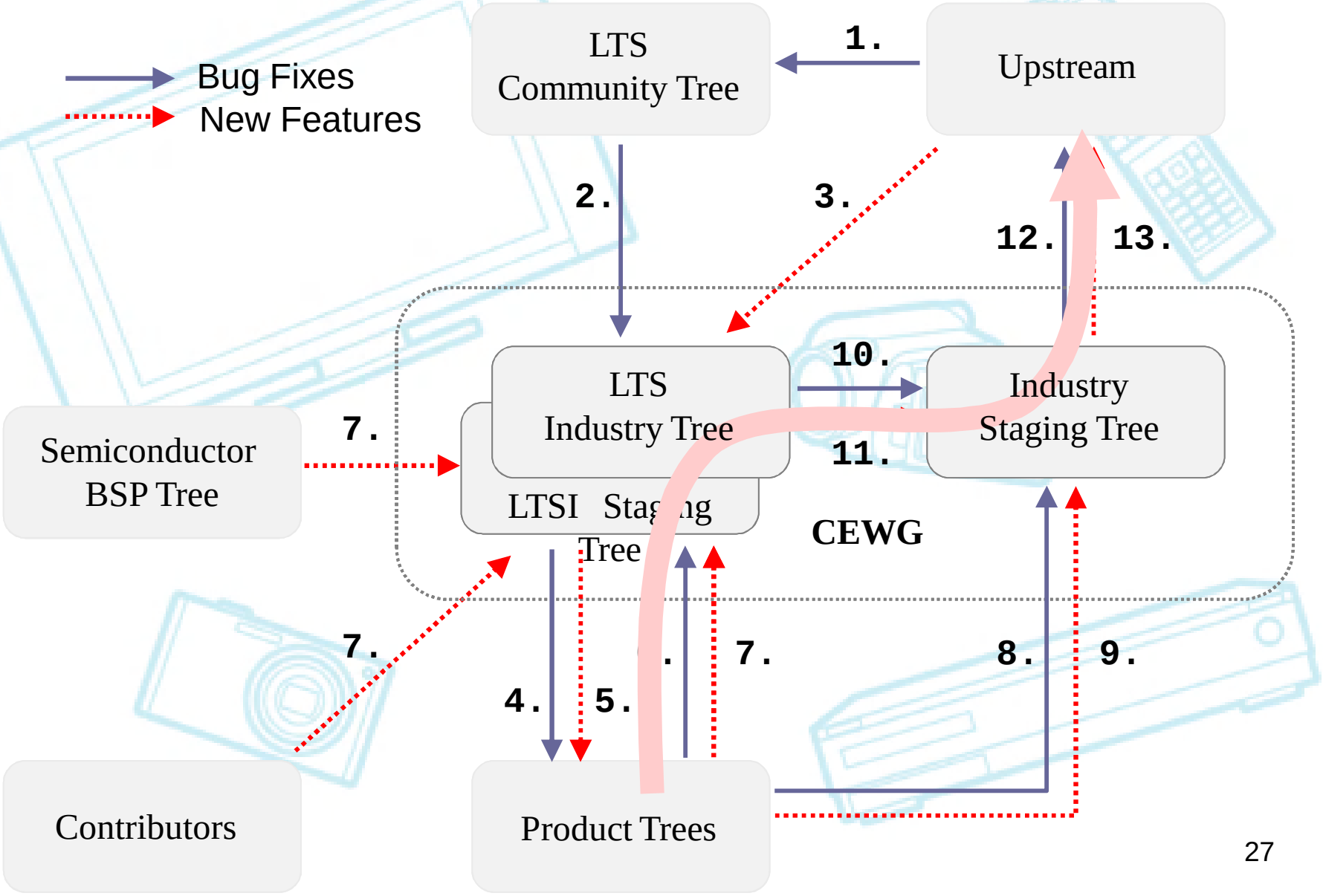
- Industry staging tree
- Accept Patches for LTSI version and port it (to latest version)
- Consultation and support for industry engineers (using LTSI staging tree)

Upstream support in LTSI:

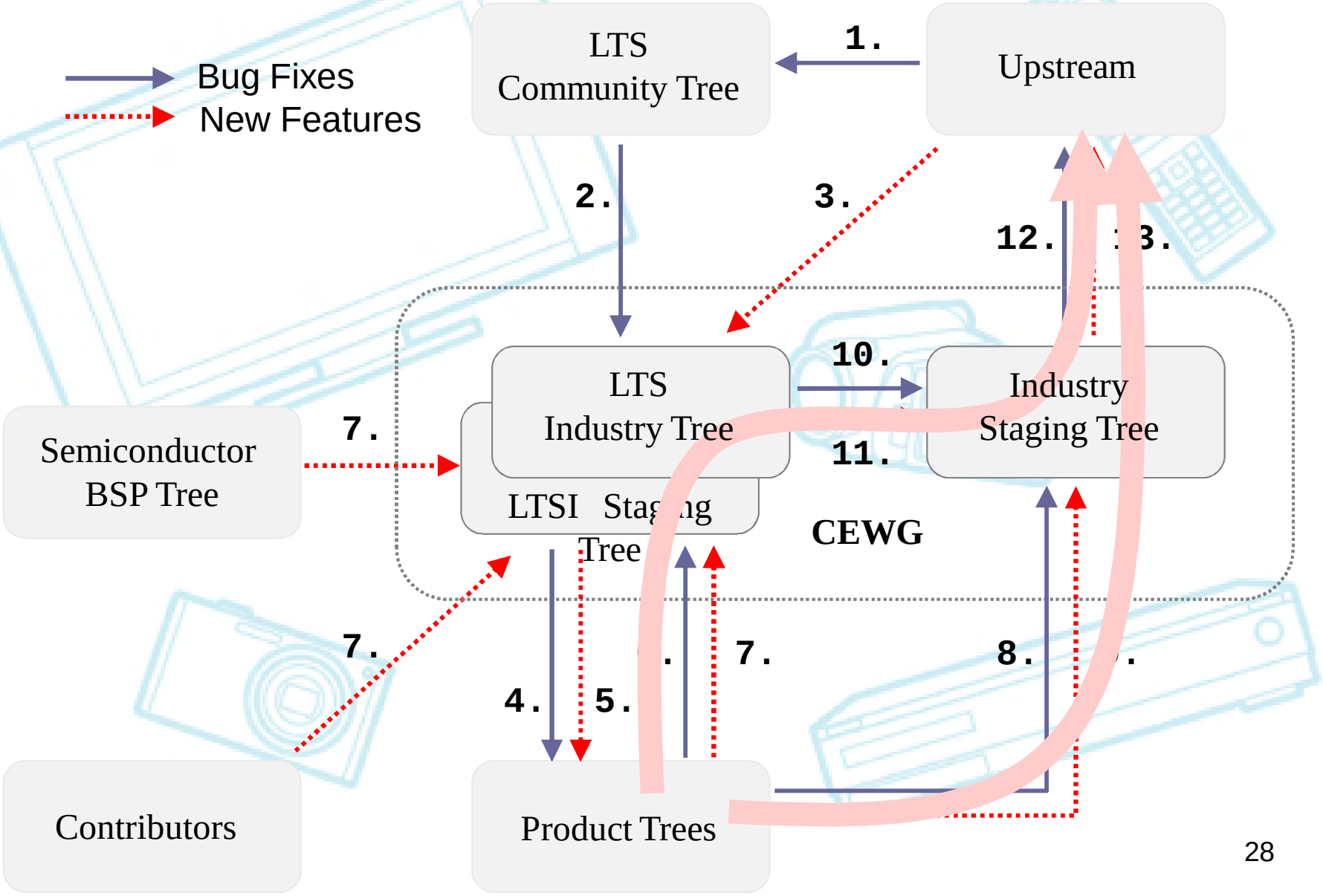
LTSI staging tree

- Industry tree to support upstream activity for industry engineers
- Be a place to collect patches (from industry engineers)
- Maintained by LTSI staging team to merge the patch into upstream
- Review and evaluate the patches and give feed back to industry engineers

Upstream support in LTSI:
Accept Patches for LTSI version and port it

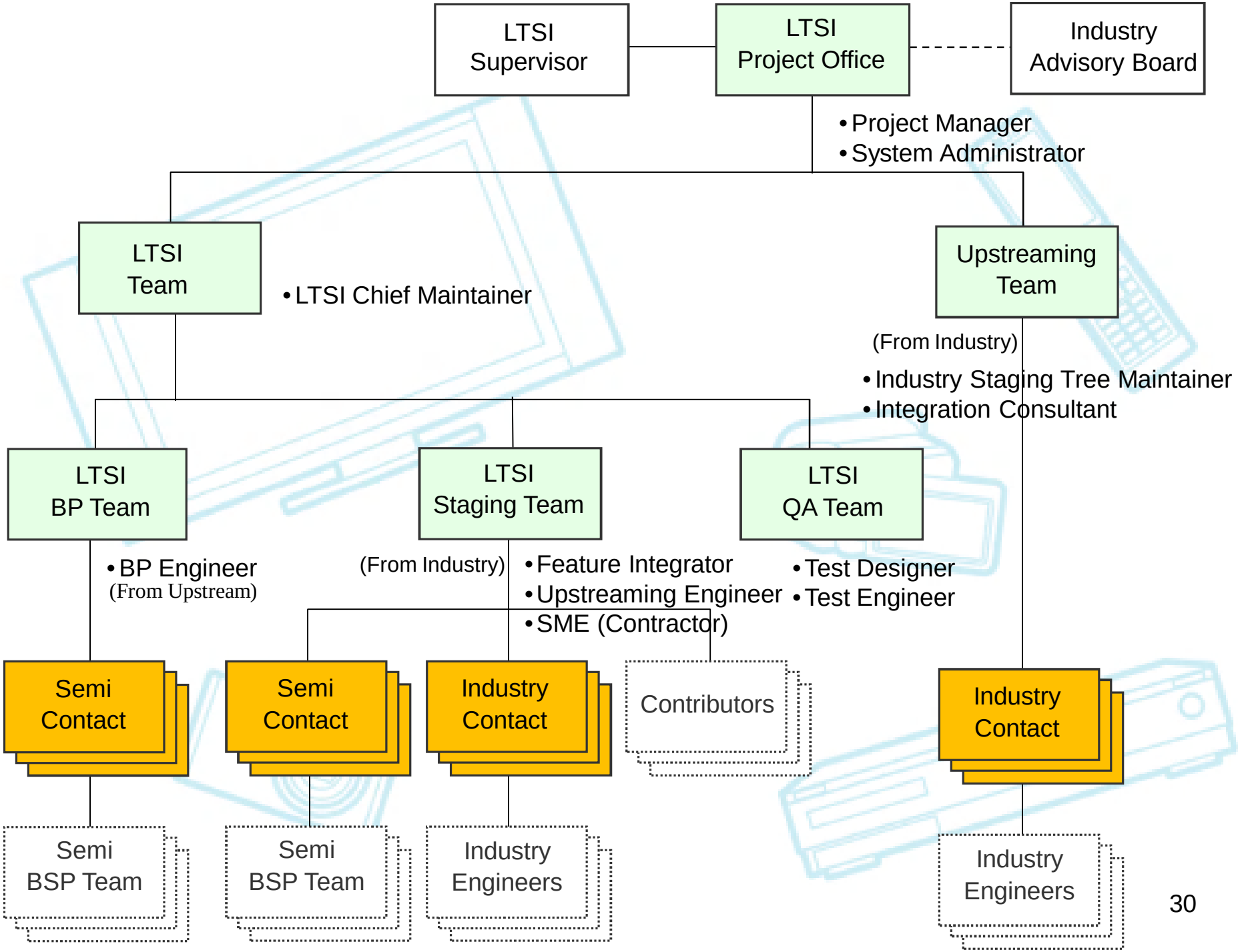


Upstream support in LTSI:
Accept Patches for LTSI version and port it



Upstream support in LTSI:
Consultation and support for upstream

- LTSI project creates better connection with industry engineers
  - Each company identifies an industry contact to LTSI
  - Each of LTSI team member will communicate with industry contact to help their upstream work


LTSI Supervisor
LTSI Project Office
Industry Advisory Board
• Project Manager
• System Administrator
LTSI Team
• LTSI Chief Maintainer
Upstreaming Team
(From Industry)
• Industry Staging Tree Maintainer
• Integration Consultant
LTSI BP Team
LTSI Staging Team
LTSI QA Team
• BP Engineer
(From Upstream)
(From Industry)
• Feature Integrator
• Upstreaming Engineer
• SME (Contractor)
• Test Designer
• Test Engineer
Semi Contact
Semi Contact
Industry Contact
Contributors
Industry Contact
Semi BSP Team
Semi BSP Team
Industry Engineers
Industry Engineers

Question : Do we really need to provide upstream support?

- We expect there are many of non-upstream code in each industry products. however, is that true?
- For what part of kernel, how many lines of codes and why?
- If we do not know anything, we are not be able to support them
- So, We need some investigation..

Pilot Project - “Yami-nabe”

- What is “Yami-nabe” ?
  - Japanese old time’s fan party to enjoy thrill and happenings.
  - The party will be held in the dark room.
    - “Yami” = Dark room, “nabe” = Cauldron
  - Participants bring some foods without disclosing anything and put them in the cauldron. No one knows what foods are in the cauldron because of the dark room and then boil it.
  - Participants pick a food from the cauldron by using chop sticks.. He or she should eat it whatever he/she doesn’t like it …

Pilot Project - “Yami-nabe”

- Why “Yami-nabe” pilot project ?
  - Manufacturers do not want to provide their kernel code to avoid unnecessary comparisons with competitors
  - We created “Yami-nabe” environment that was put kernel code without company’s name.
  - We have cooked/analyzed them and found out common changes which are not part of the Upstream.

Yami-nabe : project outcome (summary)

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

- Majority of these fragment seems come from SoC vendor code, however each code are adjusted to {fix, improve, coordinate} by manufacturers 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.

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 : 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 : [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 : [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 : [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

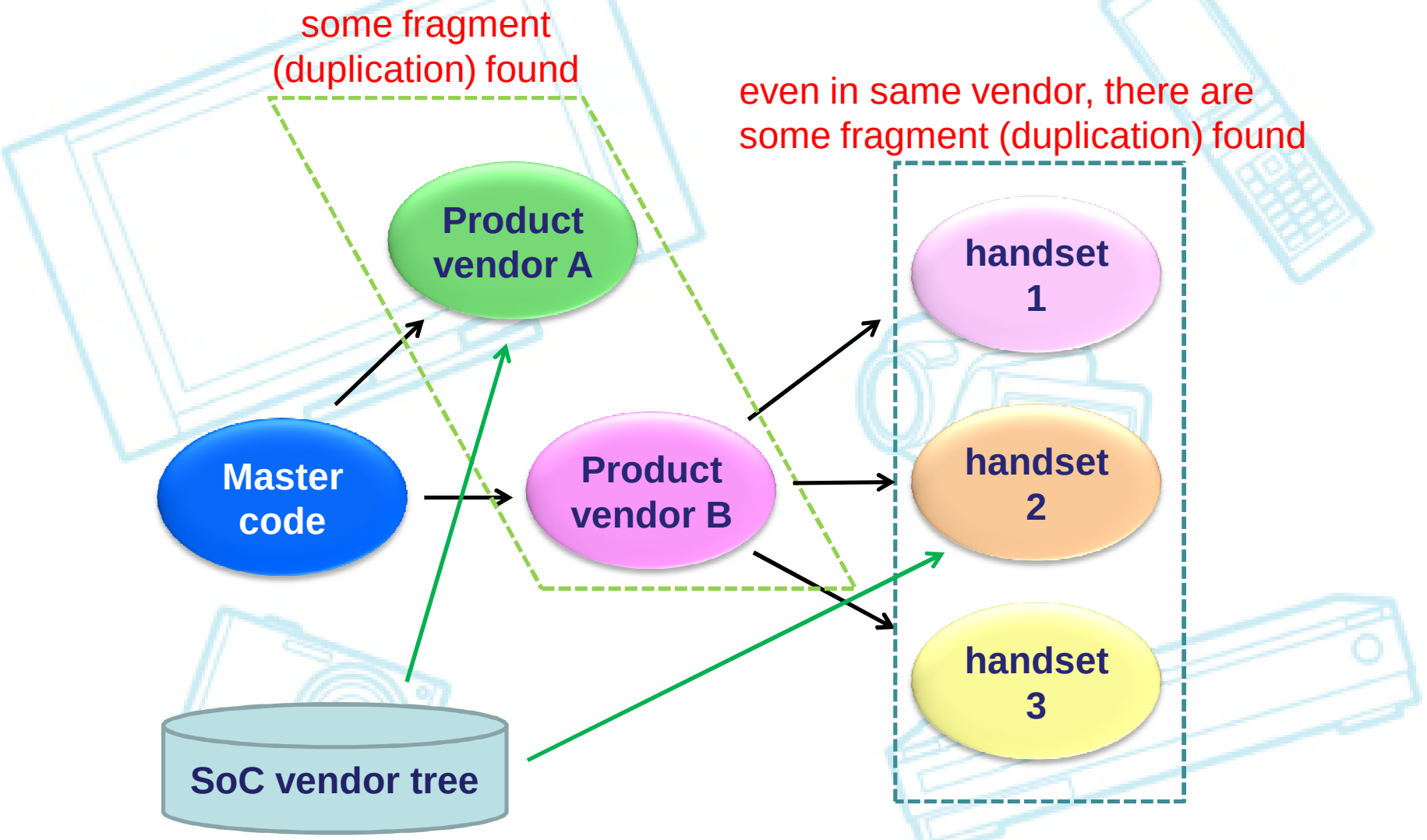
Yami-nabe : [core kernel]

#	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 : [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 : lessons learned

- If we can provide LTSI kernel that includes latest driver fix, that would reduce driver code fragment.
  (LTSI 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.
  (LTSI upstream support , consulting may help)

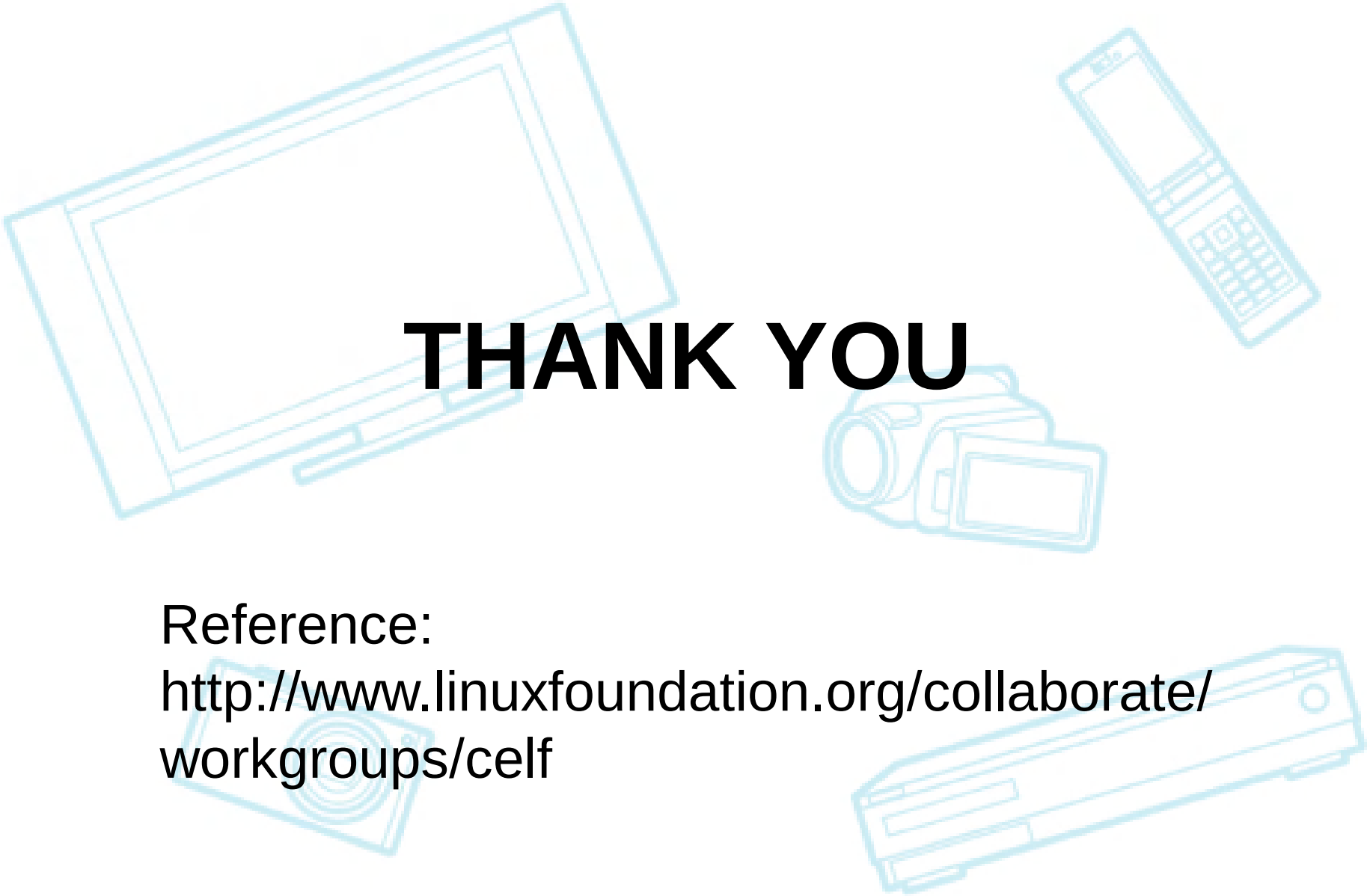
LTSI could be a solution to eliminate kernel code fragmentation seen in consumer embedded world.

Future work

- We will start the actual work of LTSI project by the end of the year and release kernel tree in 1H of next year
  - 1st LTS Industry kernel version will be 3.0
- We will continue to provide the status of the project. Stay tuned.

Conclusion

- LTSI will provide a better foundation for embedded industry to reduce the fragmentations
- We wanted to reduce the fragmentation and do not want to blame the current situation
- We hope every related parties would join our group and discuss how to solve such problem
- Please plan to use this for your products to reduce your fragmentation



THANK YOU

Reference:
http://www.linuxfoundation.org/collaborate/workgroups/celf

Supplements:
Yami-nabe details

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

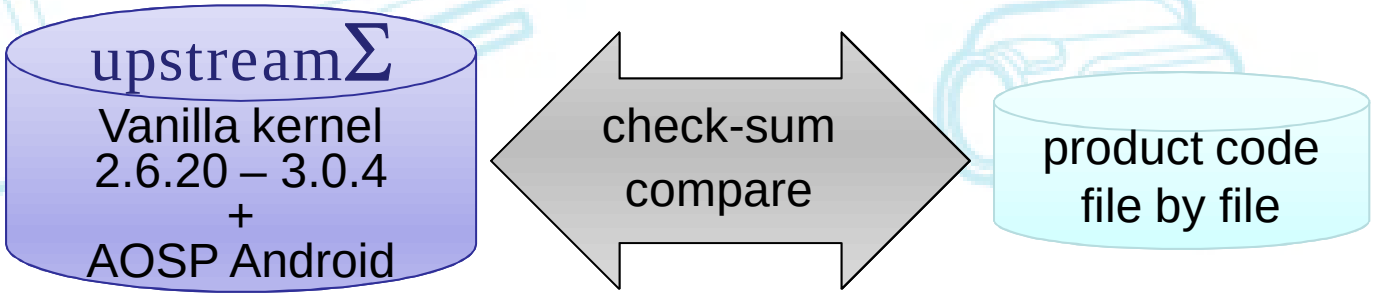
ΔA = Vanilla and Android diffs

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

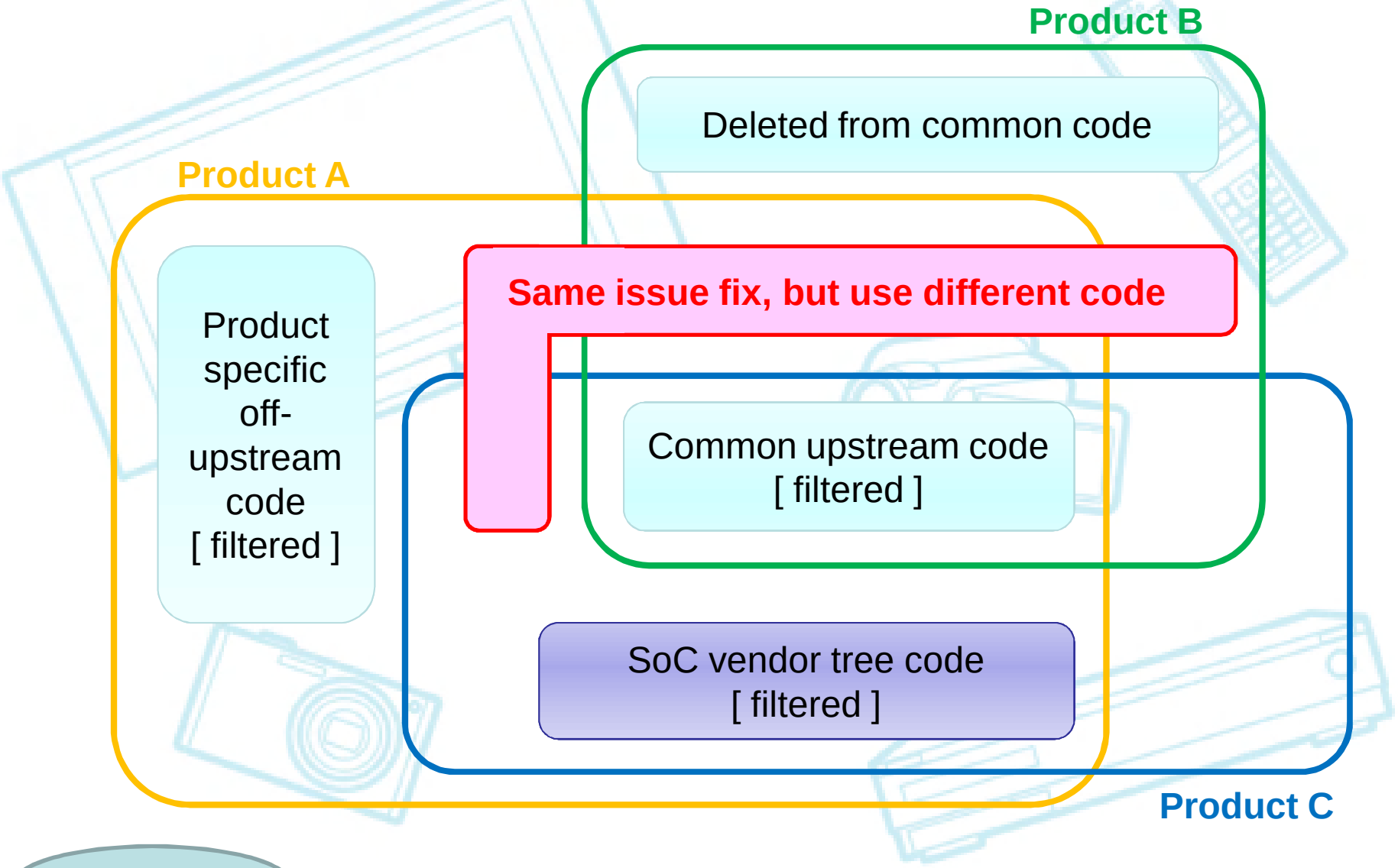
Yami-nabe : uniqueness filtering

- 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



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 : [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 : [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 : [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

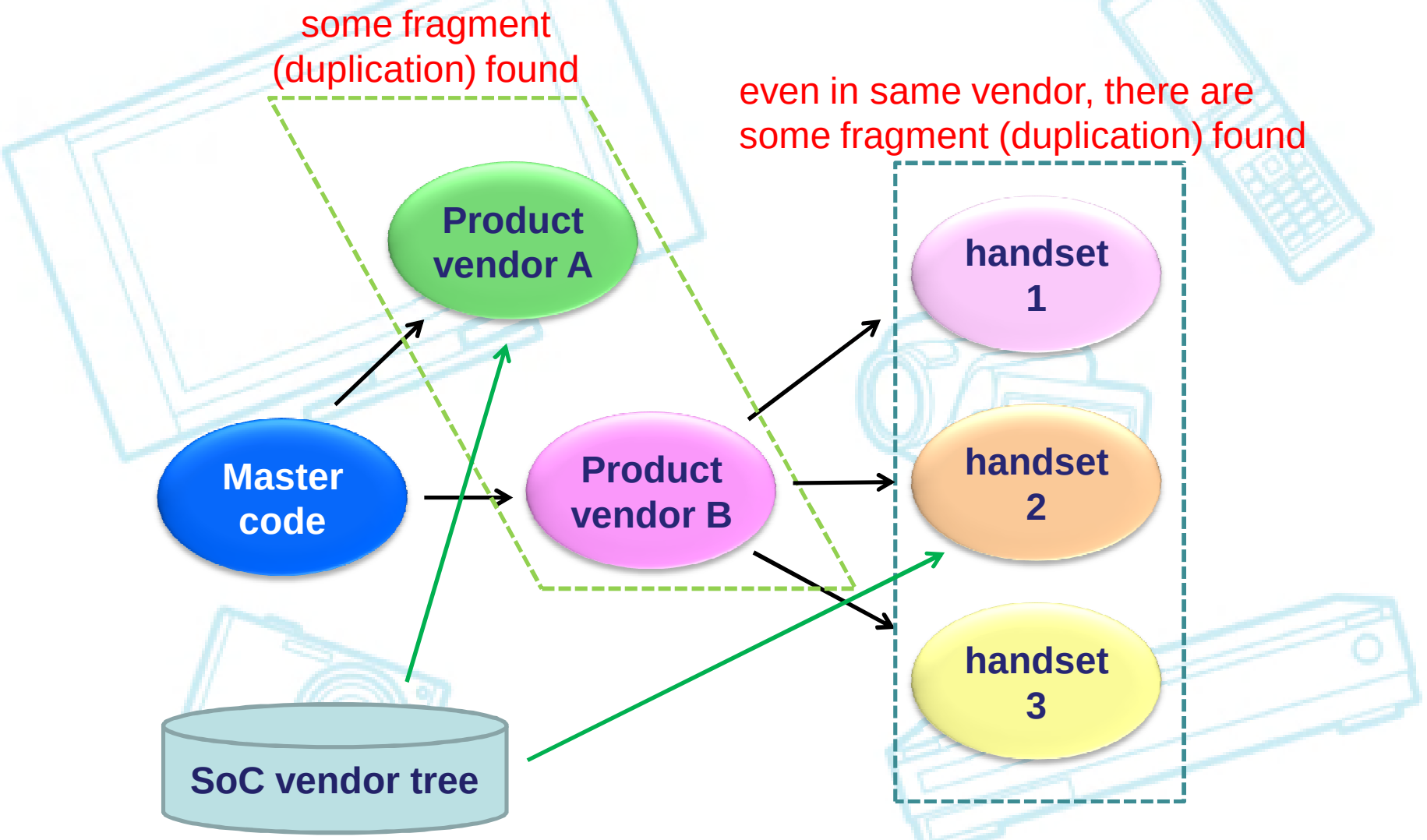
Yami-nabe : [core kernel]

#	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 : [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 : lessons learned

- If we can provide LTSI kernel that includes latest driver fix, that would reduce driver code fragment.
  (LTSI 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.
  (LTSI upstream support , consulting may help)

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