Are you Really Helped by Upstream Kernel Code?

HISAO MUNAKATA
RENESAS SOLUTIONS CORP
hisao.munakata.vt(at)renesas.com

who am I

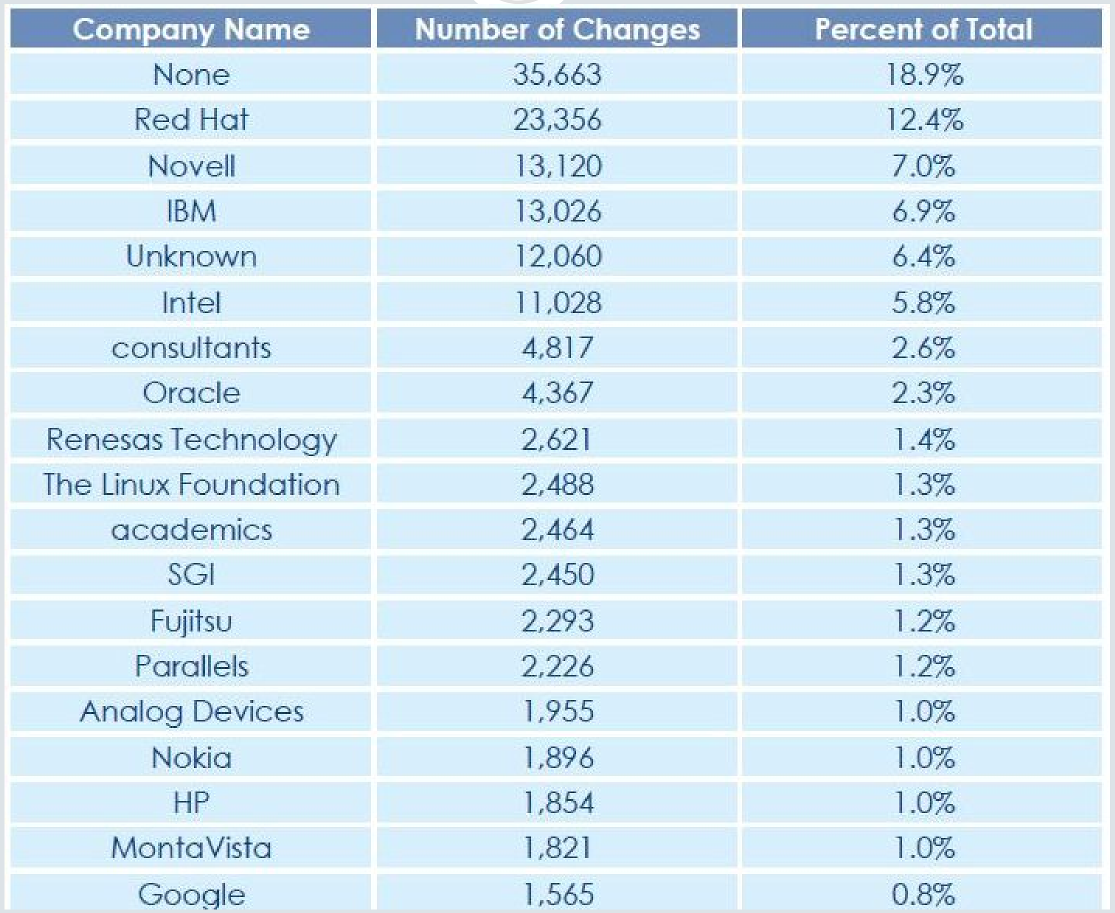
- Working for Renesas (semiconductor)

  - Over 15 years “real embedded Linux business field” experience
  - Provide “free Linux starter code (BSP)” for our platform
  - Support Linux newbie's but important customer (who asks everything about Linux to us)

  - Over 5 years experience working with the community
  - Working with industry at CELF (now “CE working group” at LF)
  - Invited some key community developers to my team, to shape our upstream activity

We have learned to adopt “upstream first strategy” now

2010 kernel patch contribution ranking
(If you measure performance by contribution no.)



Company Name	Number of Changes	Percent of Total
None	35,663	18.9%
Red Hat	23,356	12.4%
Novell	13,120	7.0%
IBM	13,026	6.9%
Unknown	12,060	6.4%
Intel	11,028	5.8%
consultants	4,817	2.6%
Oracle	4,367	2.3%
Renesas Technology	2,621	1.4%
The Linux Foundation	2,488	1.3%
academics	2,464	1.3%
SGI	2,450	1.3%
Fujitsu	2,293	1.2%
Parallels	2,226	1.2%
Analog Devices	1,955	1.0%
Nokia	1,896	1.0%
HP	1,854	1.0%
MontaVista	1,821	1.0%
Google	1,565	0.8%

Linux Foudation “Linux Kernel Development 2010”
How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It

Lesson learned from upstream work

- Use community standard development method
  - use public ML for code review and patch submission
  - use git (not to make unreadable jumbo patch)

- eliminate in-house kernel repository
- (almost all) drivers are in upstream

- Sync with upstream kernel migration schedule
  - test RC stage kernel and send feedback if any
  - continuous kernel development to support latest version

However, is it directly helping embedded product development ?

Common production team complaints

- Poor driver functional coverage
  - insufficient performance optimization (DMA support)
  - hardware error handle support is missing
  - Want to add private kernel function that never discussed at upstream (It might be corporate differentiator function)
- Insufficient document for QA team review
  - device driver document
  - test case and test report
- No milestone for future migration
- Too fast kernel migration
  (need more time to whole system stabilization)

Production team likely sticks on verified old kernel

Common community complaints

- Contribution from embedded is still relatively inactive
- Sticking on very old kernel
- Need more collaborative work with upstream community
- Need more code consolidation to eliminate vendor kernel

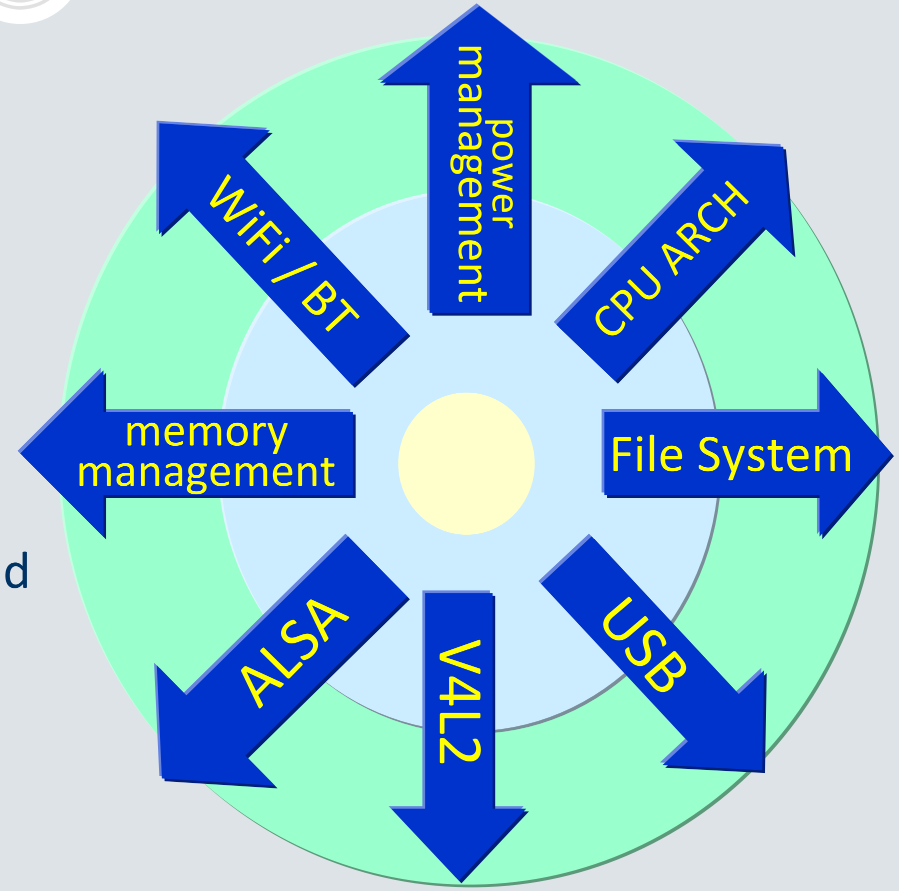
Community await more feedback from embedded forks

point of this talk

- Analyze each party’s behavioral principle
  - upstream community development
  - embedded production development team
- Clarify conflicting factor and its root cause
  - low upstream contribution
  - stick on old kernel
  - needs for document
  - unclear community development plan
- Propose best practice for embedded Linux use
  - embedded LTS version
  - backport strategy
  - forward port trial

Upstream principals = divergence

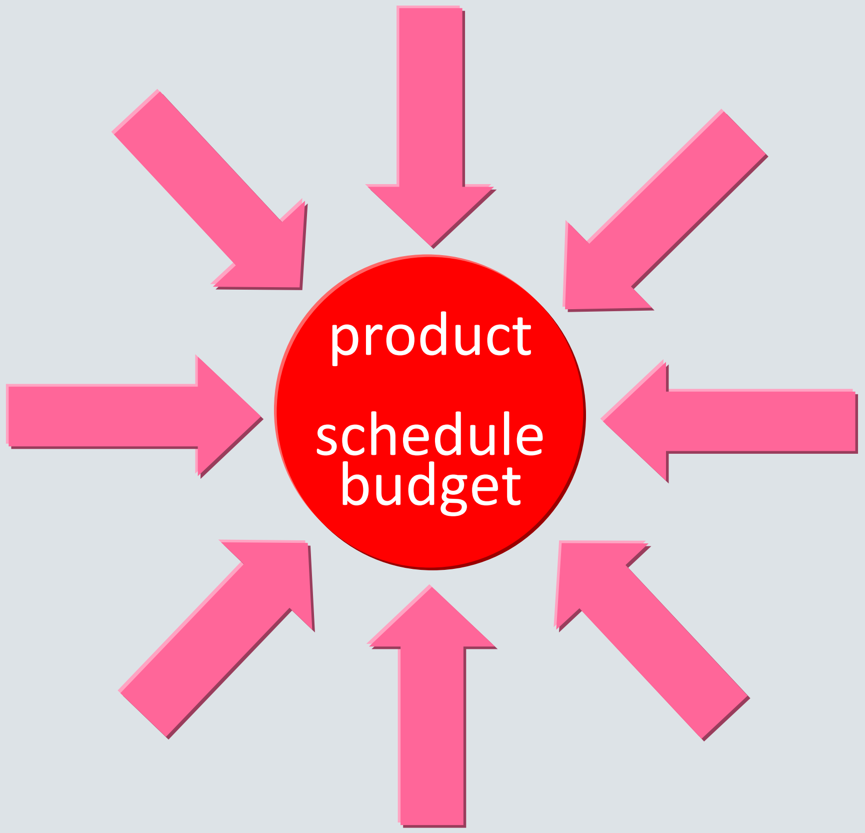
- Think sustainable evolution
  - random technical improve
  - no specific shred narrow target
  - allow diversity
- Ever lasting development
  - no specific due date
  - Think for better future
  - incremental improve
  - moving target depends on demand
- Fair governance
  - Completely open
  - purely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



Upstream guys work for unified better future for all

Production principals = convergence

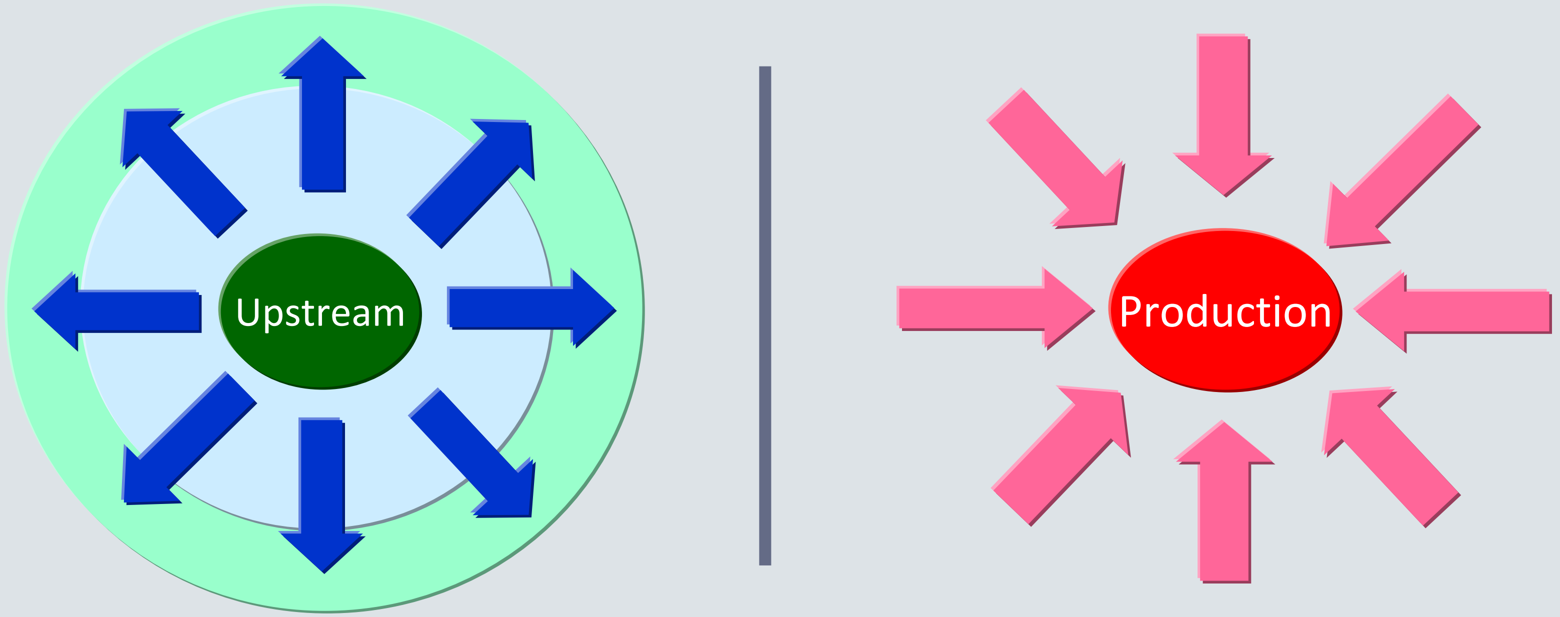
- Clear production goal
  - strict release due date
  - sever performance target
  - high cost pressure
- One shot development
  - allow interim solution
  - average skilled engineer
  - relatively large team
- Quality requirement
  - product liability demand
  - limited use case
  - reset is not allowed



Industry developer work for their current particular product

completely controversial mentality
“Upstream work can not be a part of production development”

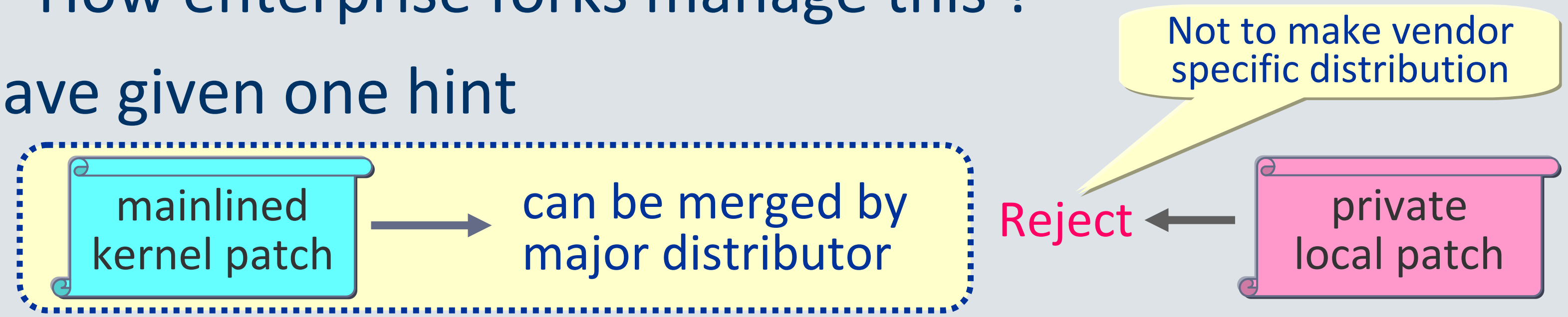
- It is hard to share upstream development and production development together, as their motivation is completely opposite.



This circumstances should not be specific for embedded forks

Why only embedded still struggles

- Enterprise forks might already educated how to coordinate conflictive two task, upstreaming development and productization together to drive Linux innovation.
  - What is missing for embedded ?
  - How enterprise forks manage this ?
- I have given one hint


mainlined kernel patch
can be merged by major distributor
Reject
private local patch
Not to make vendor specific distribution

  - Enterprise distro accept already mainlined patch only
  - It motivated enterprise developer to write upstream patch as a extended part of their productization task.

Need more motivation and "consolidation point" for embedded

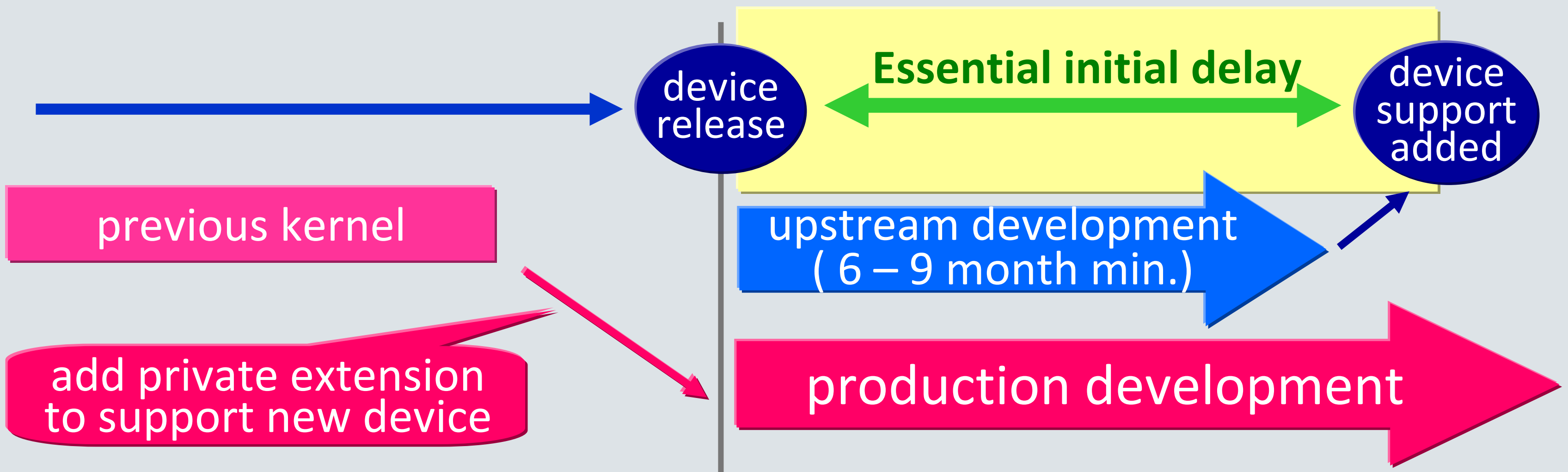
Trial clarification for embedded characteristics

As a witness of real embedded Linux field, I would like to give my shot for several confusion happening in embedded Linux world. I hope this helps to find best practice to tackle them. Some questions might be form community side, others from production team (as I head almost everyday).

1) Why embedded forks stick on ancient version kernel ?
2) Why upstream code does not contain full SoC functionality ?
3) Why there is no written document for Linux driver ?
4) Why you can not commit schedule for upstream migration ?
5) Why embedded likely develop private device driver code ?
6) Why embedded require more time for system validation ?

Why embedded forks stick on ancient kernel ?

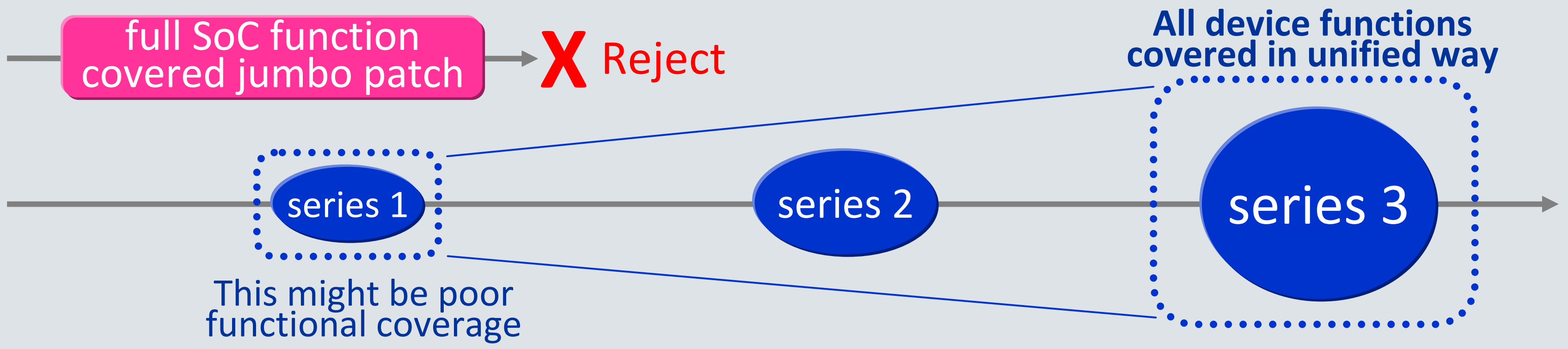
- Essential initial delay problem
  - Set producer want to adopt latest SoC device.
  - Each SoC device include slightly (or heavily) modified IP blocks
  - So upstream device driver does not include its support then.



We can not provide mainlined Linux support at device release point

Why upstream code is not fully SoC functional ?

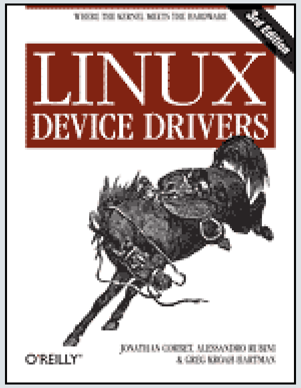
- When if you attempt to add your device support to the upstream code, it should be simplified. Otherwise community maintainer will not accept too big full functional patch.
- To improve more wider functional support (like add DMA support), you need continued work to add them. It should be generic (not device specific) as much as possible not to cause unnecessary per device fragment. It may take some time.


full SoC function covered jumbo patch
X Reject
All device functions covered in unified way
series 1
series 2
series 3
This might be poor functional coverage

Early stage upstream driver might be intentionally simplified.

Why there is no written document for Linux driver ?

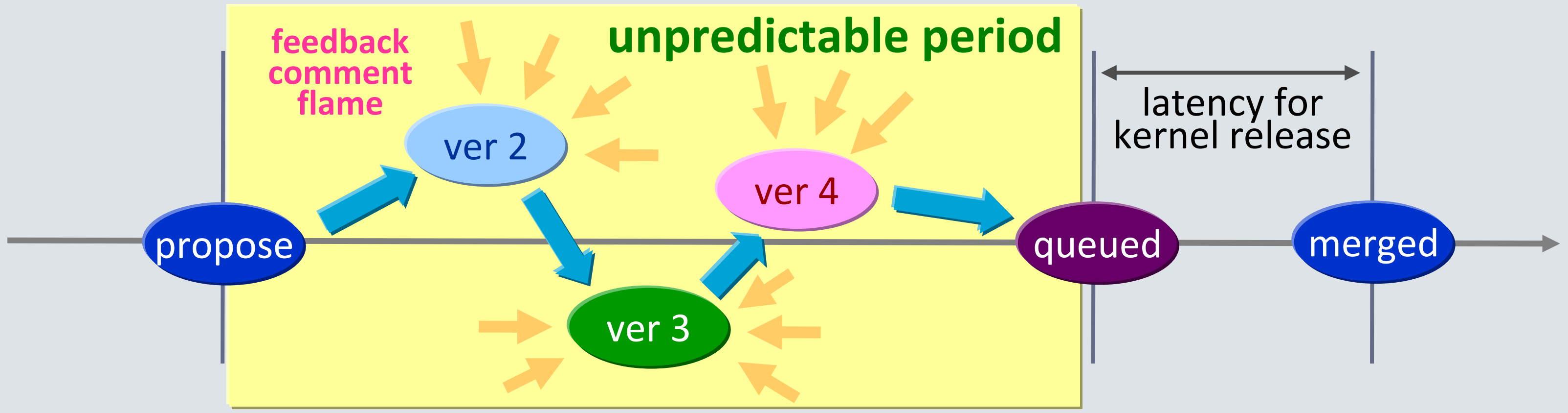
- Initially Linux kernel includes various nice document inside
  - See /document directory
  - Some document are also translated (jp, cn, …)
- Linux is moving target, driver API may change if it is really needed. So there is no common document.
  - “Linux Device Driver” has migrated 3 times
- Production team, especially QA team want to review driver document to verify its behavior. Error recovery capability should be interested.



Nice to have driver document targeting particular version, but

Why you can not commit development competition ?

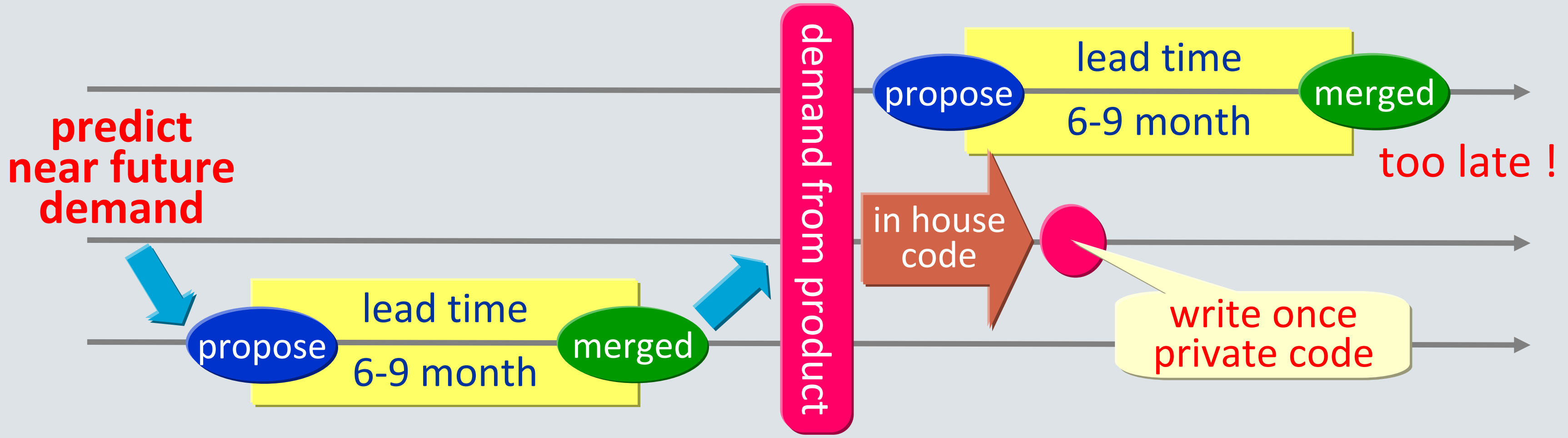
- If you develop your code as a part of upstream code (this is really recommended), you must go through pre-defined community "public review / improve / approve" process.
- If your proposal does require coordination with existing kernel design, it may take some longer time. Also you may requested to modify your code not to conflict with others.



Everything production team needed should be in place up front

can not wait upstream driver/framework support

- Upstream team can hardly fulfill the immediate demands comes from production team, because each development takes some time. So it need to be developed in advance.
- Ideally upstream team should predict future production demand trends from its marketing strategy.



Upstream team should predict future production demand

Why embedded need more time for system validation ?

- Not all embedded device connected to the Internet.
- Not all mobile device are ready for field firmware update.
- Many embedded device are expected run 24-365 basis.
- Product user might not be familiar with reset / reboot.

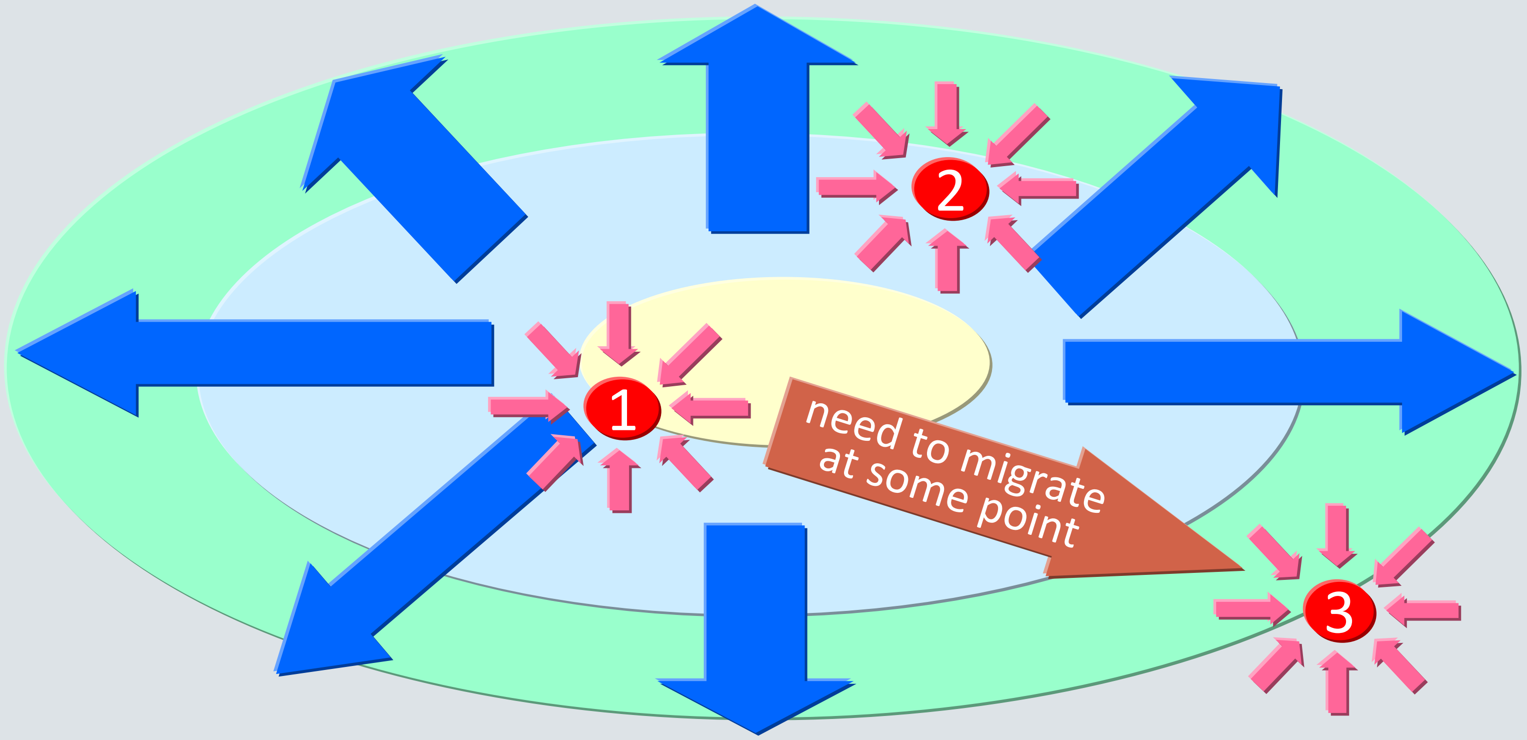
- There is no standard tiny Linux userland for embedded, and each production developer need to validate their userland with their own application on certain environment.

- Dynamic fundamental software (kernel, toolchain) change require whole system re-validation. So they want to keep use same base environment as much as possible.

Embedded forks eager to have common solid Linux base

And yet, embedded forks need to migrate

- It is quite certain that embedded developer need to migrate their kernel to support new feature that market requires.



Then, how you can minimize kernel version migration cost ?

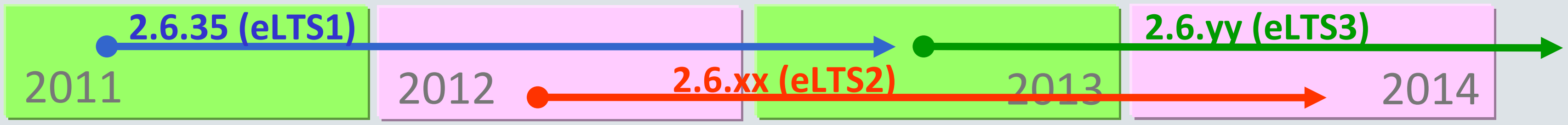
embedded best practice

- It seems there is almost no chance for current embedded developer to join upstream activity as they are too busy.
- However production developer eager to utilize latest kernel advanced capability to make their product competitive. (like advanced power management, SMP utilization,.. etc)
- Upstream team can hardly fulfill the demands of production team when it is demanded, because each development takes some time. So it need to be developed up front.
- So we need to establish “unlinked but coordinated” relation between upstream and production developer.

reciprocal, bi-directional interaction between two separate teams

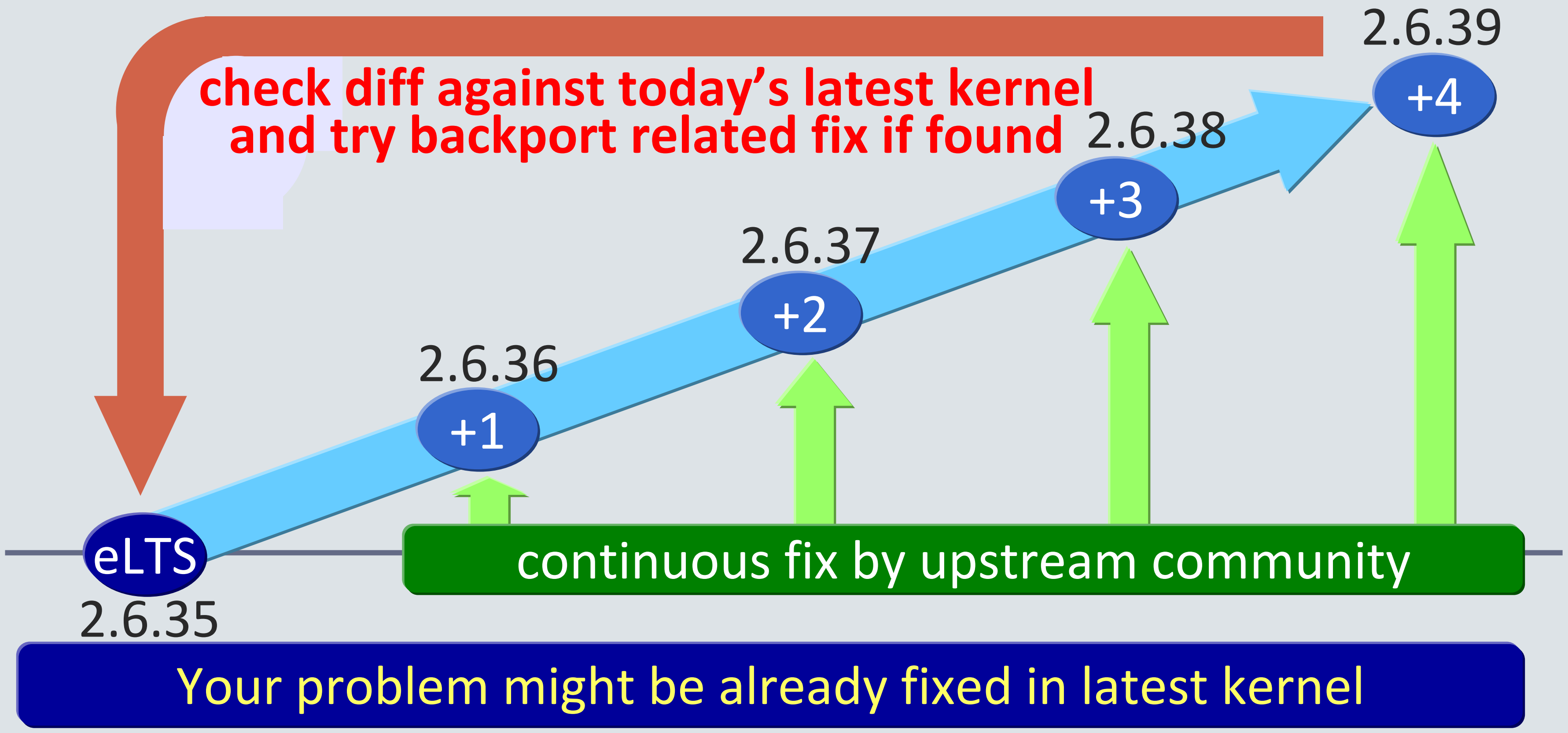
Use eLTS for embedded consolidation point

- 2.6.35 is current “embedded LTS (eLTS)” version
  - confirmed at kernel summit 2010, based on request from embedded industry demand
  - Andi Kleen is maintaining this version
  - Industry can share this as common base kernel
- We need to establish future eLTS selection policy
  - How long one eLTS version can be maintained
  - How many eLTS can exist at same time
  - Which version should be selected as next eLTS
  - How and who maintains each eLTS



If you hit any problem with eLTS kernel back to today's latest (even under develop) kernel

- If you find any problem on eLTS kernel, the first thing you should do is "backport from current latest kernel".


check diff against today's latest kernel
and try backport related fix if found
2.6.39
+4
2.6.38
+3
2.6.37
+2
2.6.36
+1
eLTS
2.6.35
continuous fix by upstream community
Your problem might be already fixed in latest kernel

kernel migration category

There are various code update on mainline kernel code

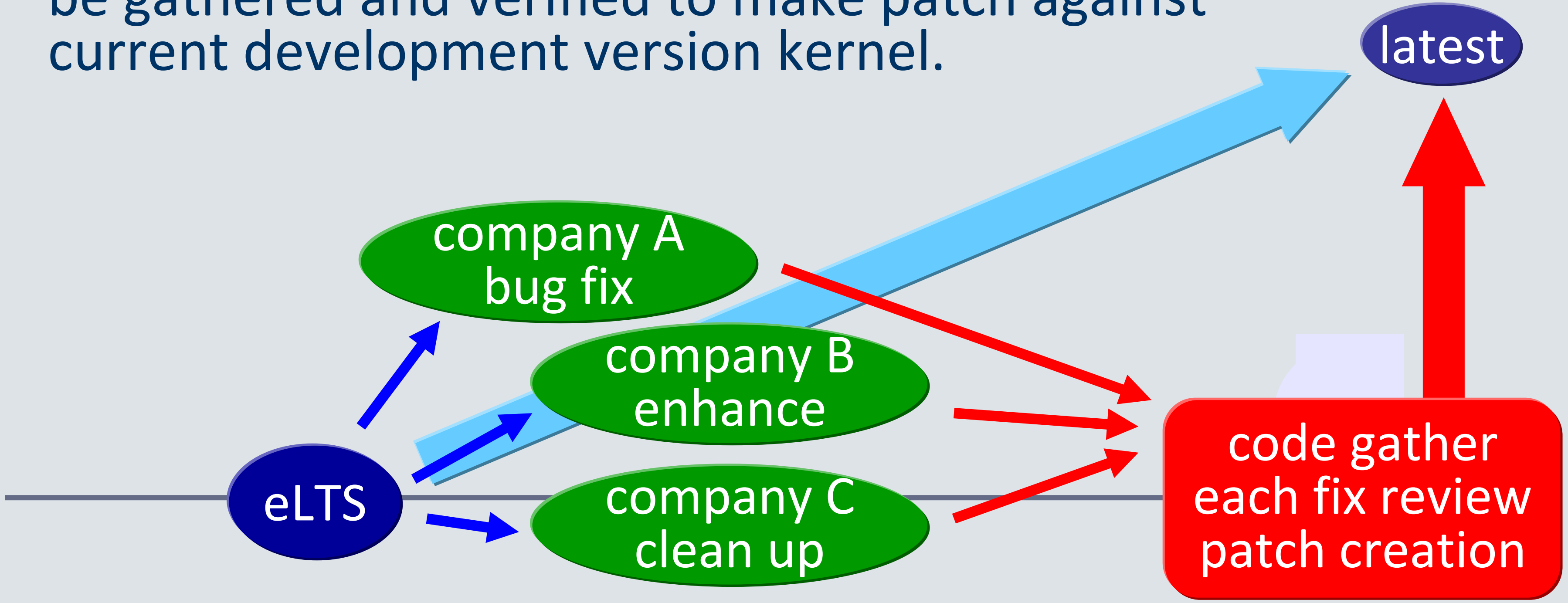
- B: Bug fix / Correctness Fix
- C: Clean-up / Infrastructure Change / Documentation Change
- F: New Feature (not performance related)
- P: Performance Enhancement
- U: Usability Enhancement
- X: Unclassified

Embedded forks can work together to make extended eLTS kernel for production development cost reduction

Limited sense of eLTS = "B" only
Extended sense of eLTS = "C", "F", "P", "U", "X" out of "B"

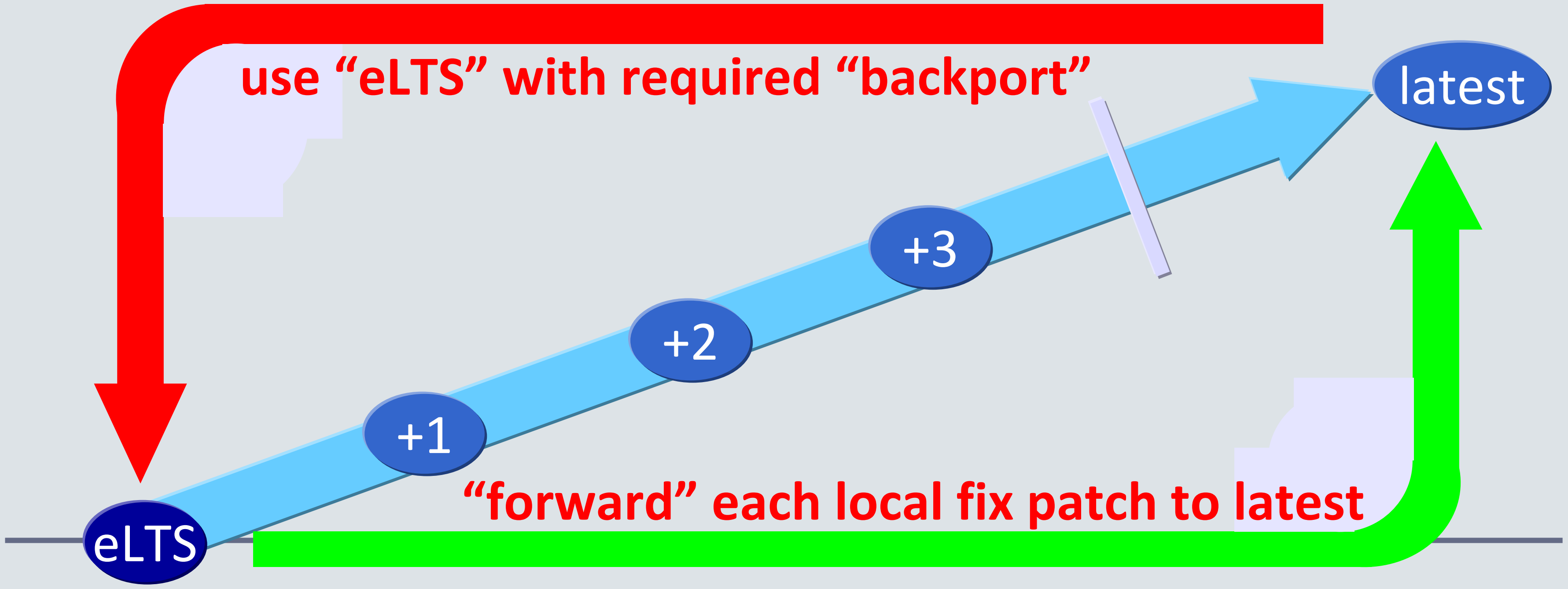
Forward port (new concept)

- To minimize gap between production kernel (= eLTS) and community upstream (=latest development version), every fix, optimization done by each production company should be gathered and verified to make patch against current development version kernel.



ideal reciprocal action

- "back port" and "forward port" should happen concurrently to minimize future kernel migration cost


use "eLTS" with required "backport"
latest
+3
+2
+1
eLTS
"forward" each local fix patch to latest

Your problem might be already fixed in latest kernel

How and who make this happen

[Down Stream (back port)]

🟦	choose eLTS version	: community and industry
🟦	maintain eLTS version	: dedicated eLTS maintainer
🟥	use eLTS for production	: production developer
🟥	apply latest fix, improve	: production developer

[Upstream (forward port)]

🟥	send fix to eLTS maintainer	: production developer
🟩	verify each local patch	: eLTS maintainer
🟩	write patch	: eLTS maintainer
🟦	review, correct patch	: community, eLTS maintainer
🟦	apply embedded patch	: community

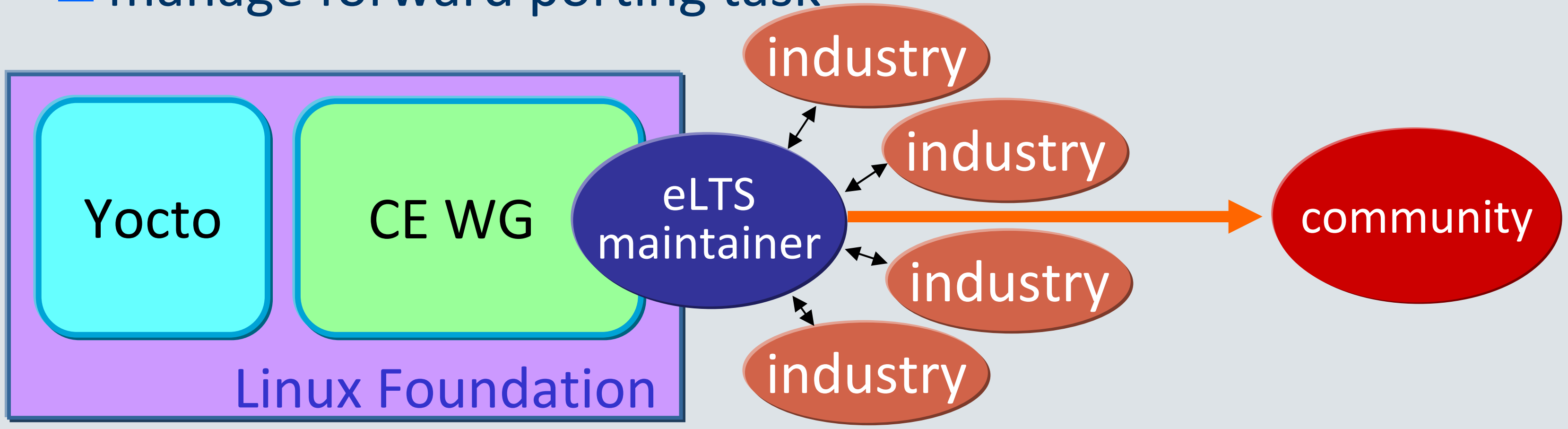
role of eLTS maintainer

- backport
  - apply latest bug-fix on eLTS version
  - apply latest security-fix on eLTS version
  - apply some new feature from new kernel (optional)
- forward port
  - collect each vendor’s local work result
  - review and consolidate each patch
  - migrate base kernel to current development version
  - write patch to upstream and attempt to mainline it

eLTS maintainer has key role to make new scheme workable

LF CEWG aims to drive this new scheme

- Linux Foundation start more focus on embedded
  - Yocto project
  - CE Working group (AKA CELF)
- I want to add “eLTS” maintenance as part of CE WG task
  - propose eLTS selection scheme to community
  - maintain eLTS kernel for embedded user
  - manage forward porting task



New strategy summary

- Community already define and maintain eLTS version
- LF CEWG try to hire eLTS maintainer for future eLTS kernel maintenance
- Embedded production developer can utilize eLTS kernel
  - If hit any issues, check latest kernel for community update
  - If you needed private fix, please send it to eLTS maintainer
- eLTS maintainer can review each feedback from embedded production to pick up common proposal for latest kernel

Conclusion

- Upstreaming and productization have completely separate motivation for their development. And embedded team can hardly send code to community upstream as a part of job.
- Embedded forks has not established any collaboration scheme that connects upstream and production. This is the reason behind the relatively low rate of embedded upstream contribution.
- We want to utilize eLTS for embedded common base and expect feedback from industry to feed forward good code made by industry developer work to latest kernel.
- LF CEWG try to make this new scheme workable.