CLOSE ENCOUNTERS OF THE UPSTREAM RESOURCE

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

who am I

- Work for Renesas (semiconductor provider)
  - Over 15 years “real embedded Linux business field” experience
  - Providing “free Linux starter code (BSP)”
  - We help our important customers who are Linux newbie's.
    (They ask anything about Linux to us.)
  - Over 5 years experience working with the community
  - Linux Foundation CEWG Architecture group co-chair
  - I am leading Linux core technology team in Renesas that develops upstream Linux code. As this team consisted of several community developers we had chance to work closely with Linux upstream.

We are learning how to work with the Linux community

MY FINDINGS

"production team" vs. *"evolution team"*

community, long-term, reusable code,
no appointed date of delivery

"utopia"

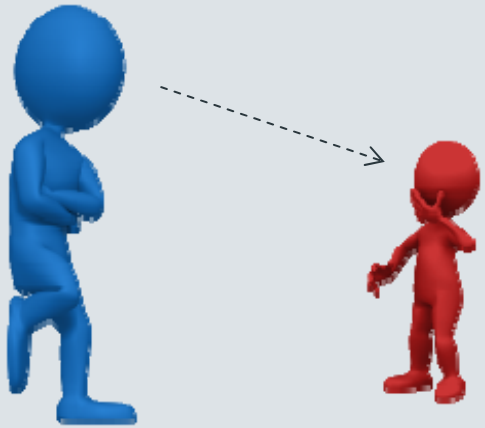
"red party"
product developer

"blue party"
upstream developer

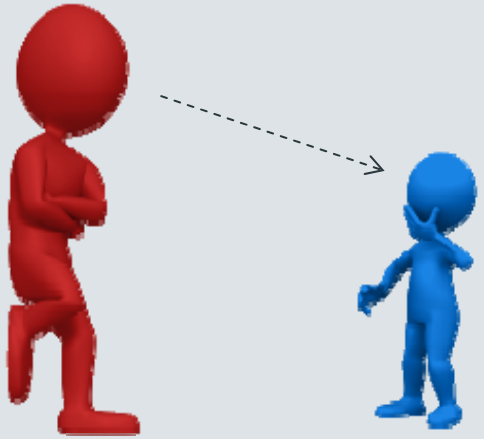
"business"

budget, mass production, competition

Each party aims different goal



- too short focused “easy” solution
- A code reuse rate is low
- No intention for open source contribution
- lack of feedback (requirement, bug info)
- random requirements



- development takes too long time
- maybe lack of stabilization
- hard to make product differentiator
- can not help today’s release
- unclear support scheme

Each party aims different goal and it seems very hard to share it.
However red party’s (future) work depends on blue party anyway.

Red team development system at the time of “binary-only world”

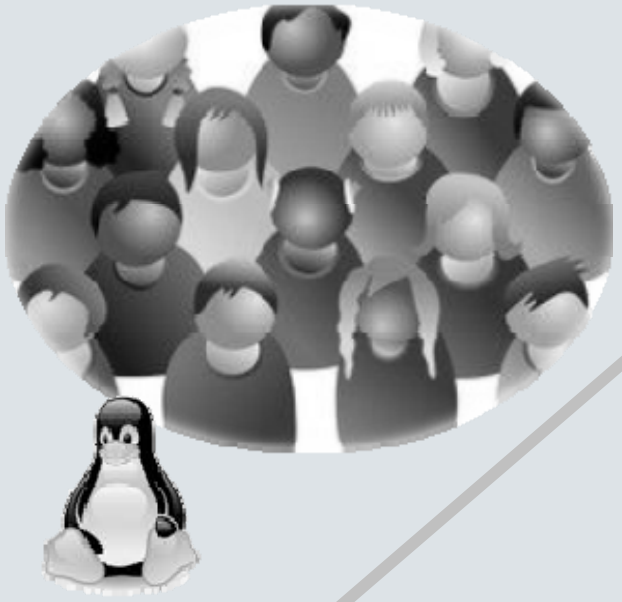
authorized integrator

binary
code delivery
license fee

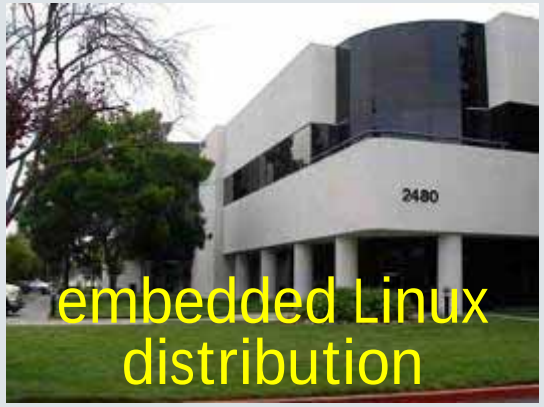
contract based
paid support

There were no chance (demand) to access source code. They could ask paid support to get workaround, but it was not a true solution.

Red team development system at the time of start using Linux for CE products



source code

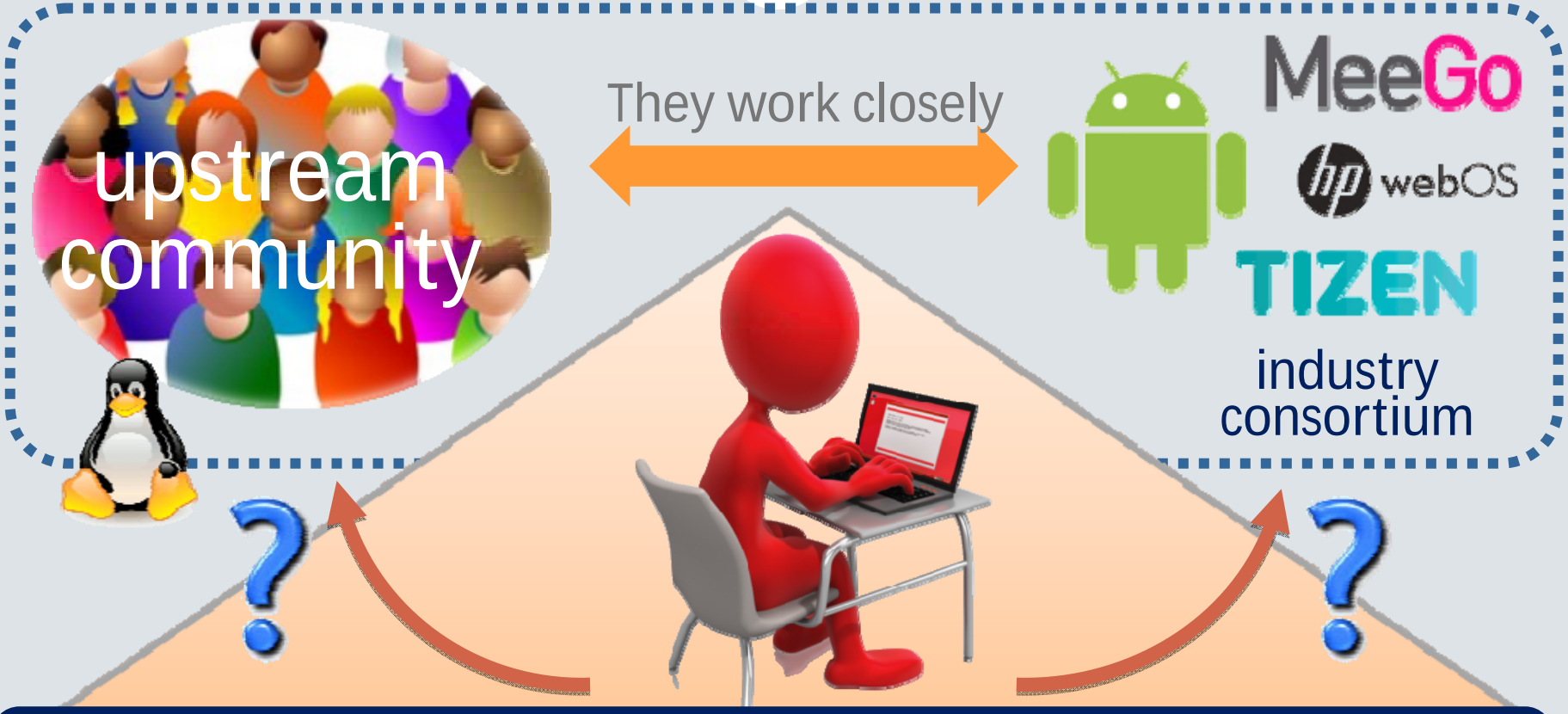


almost no interaction

source code integrated BSP paid support

Industry developers were mainly supported by embedded Linux distribution and/or integrator company. Due to kernel version gap, there were very limited connection to the upstream community.

Red team development system at today (start adopting community framework)



Industry developers have started to adopt relatively new kernel that have been integrated in software platform. However they are not familiar with working with the upstream development community.

STRUGGLES OF PRODUCTION TEAM

struggles 1

tend to develop own unique kernel

hard to mainline

X

too unique code

You are allowed to modify source code as you like, BUT...

struggles 2

incorrect way of patch creation

Linux has been constructed as accumulation of a degradable small patches.

- Driver code for specific environment that is not abstracted.
- Does not work with SMP
- incorrect use of kernel API
- non-portable coding

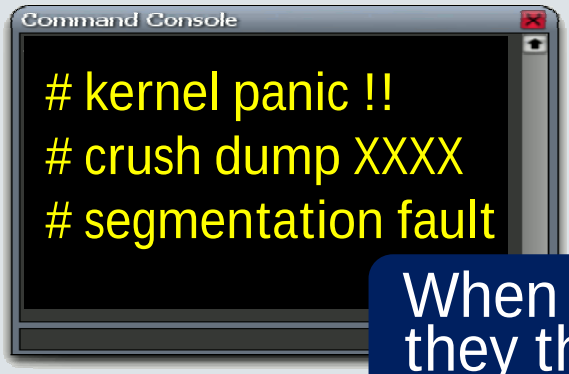
- too big piece hard to merge
- duplication of code
- missing git log information
- wrong patch format

Locally developed large & irregular patch can not be integrated

struggles 3

lack of elemental modern OS understanding

everyday
till mid-night



When they faced a problem of the Linux origin,
they think they can be settled in oneself. However ,...

struggles 4

emergency support request may not work

pointless log gathering

non-essential guide

struggles 5

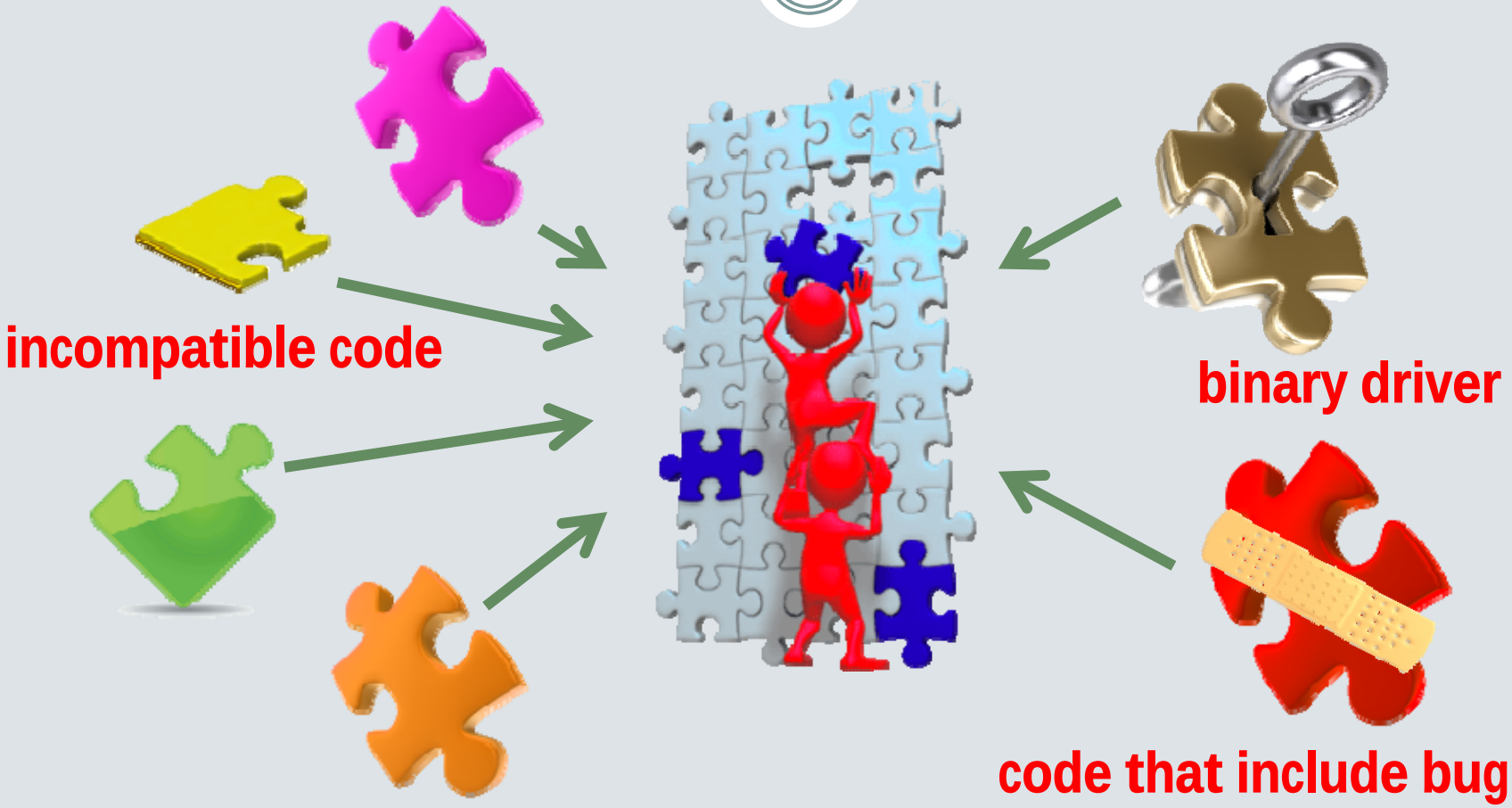
re-inventing already created bug-fix

fix already exist in new kernel code

creating in-house fix

struggles 6

mindless patch merging without inspection



They really do not understand the role of tree maintainer

struggles 7

long term maintenance of in-house orphan tree

It costs enormous expenses for the lone tree maintenance.

DO THE RIGHT THING
WITH THE COMMUNITY RESOURCE

As you start using fresh kernel, community became close

They had been completely isolated
(due to the kernel version gap)

2.6.18

getting
closer

Linux 3.0

Resources of the upstream community are more available till now.

Even if you are not a upstream developer,....

- I know production developers are extremely busy with their own target development, and I do not demand the following thing.
  - check all community ML discussion relating to your topic
  - create patch against current upstream development kernel
  - negotiate in ML discussion to merge your patch to the upstream
- If you can pay more attention to existing open source community resources like *git*, BTS, ML and others, you may utilize them to make your development more efficient and clean. So this is the right time to breakout to the open source arena.

Breakout to the community resource !

developer network

upstream community

github SOCIAL CODING

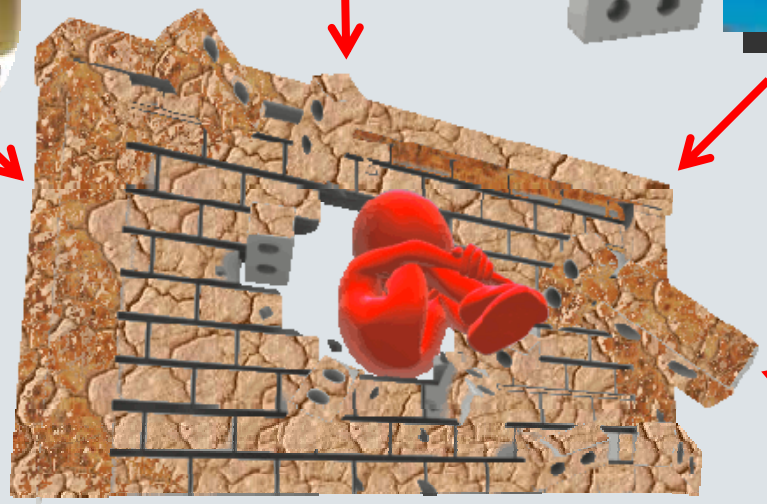
SOURCEFORGE®

code repository

ML archives

THE LINUX FOUNDATION

consortium

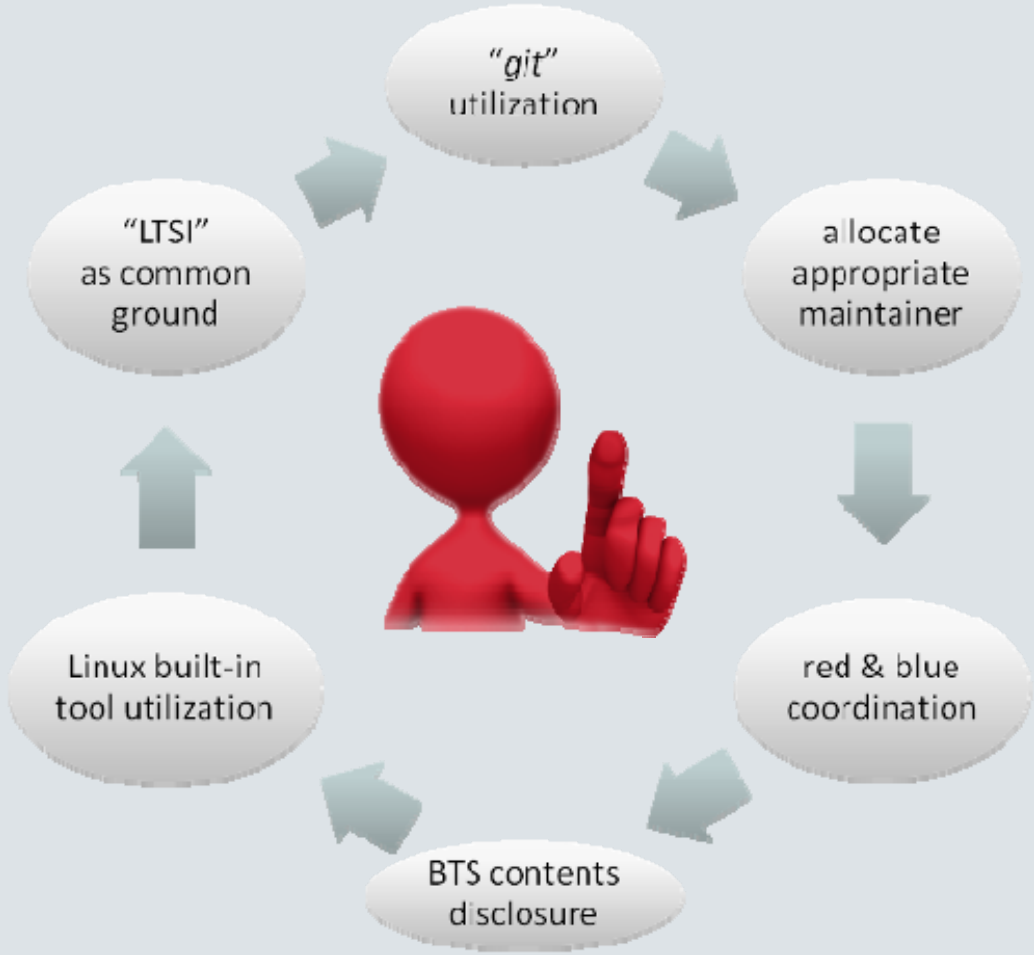


Google code

code archives

Even without direct contribution, red team developer can utilize various “open resources” to manage better work !!

several recommendations

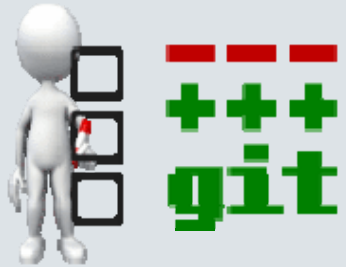


recommendation 1

correct use and utilization of *git*

Git is extremely powerful code management tool. Correct and efficient use of *git* is essential to improve your development.

- You should add correct git log message to all codes you add. Source code must be tie with *.git* repo management information. This will dramatically reduce future debug and maintenance cost.
- If you develop Linux/Android code, you can always compare your code and current latest community code. It will show how community modified (polished) the upstream code. And their code might already include the code you need to develop.
- You can create any experimental branch with *git*. And you can merge it to the original tree when it is enough verified. Creation of random experimental branch without *git* use easily makes spaghetti code.



Example 1

kernel bug-fix patch cherry picking

1. Determine potential problematic area by locating commonly used functions in function call history included in error log messages
2. Use git to check which bug fix commits that are included in latest stable tree update for the problematic area
   http://git.kernel.org/?p=linux/kernel/git/stable/linux-stable.git;a=summary
3. Use git to check new commits in latest upstream tree for the problematic area, maybe bug fixes are missing from stable?
   http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=summary
4. If no potential fixes exist in stable or latest upstream then give up
5. If fixes exist then create custom back port with multiple commits covering the problematic

Example 1

kernel bug-fix patch cherry picking (cont'd)

6. Test the custom back port and keep your fingers crossed
7. If problem disappears then adjust back port to become smaller to work towards locating actual fixes
8. If problem still exists then make back port larger to cover greater area to include more potential fixes
9. Repeat testing and adjustment of back port until satisfied or giving up
10. Contact stable maintainer to discuss inclusion of fix in next stable release

Example 2

Android version comparison

- You can download Android platform source code (except 3.x series) from AOSP repository. And fortunately AOSP code includes *git* management information, you can compare your code and AOSP latest code pretty easily. So if you are stuck with any Android platform related problem, it is good idea to check this first to notice Android upstream work result as it may already include solution for your problem.
- Furthermore, you can compare the cord of the latest Android 4.0.x series with the cord of the Android 2.3.x series. This may help you when you want to add SMP support upon 3.2.x environment.
- You need to become professional user of *repo* and *git* command to perform efficient code comparison.

Example 2

Android version comparison (cont'd)

- You can compare OLD and NEW with following *repo* command. This will show anything modified from OLD -> NEW and newly added stuff on NEW.

```
munakata@muna-E420: ~/android
munakata@muna-E420:~/android$ repo forall -p -c ¥
> 'if git rev-parse OLD > /dev/null 2>&1; ¥
>  then git log --no-merges OLD..NEW ; ¥
>  else git log --no-merges NEW ; ¥
>  fi'
```

- Sample to compare android-4.0.1_r1 and android-4.0.3_r1

```
munakata@muna-E420: ~/android
munakata@muna-E420:~/android$ repo forall -p -c ' ¥
> if git rev-parse android-4.0.1_r1 > /dev/null 2>&1; ¥
> then git log --no-merges android-4.0.1_r1..android-4.0.3_r1 ; ¥
> else git log --no-merges android-4.0.3_r1 ; ¥
> fi'
```

Example 2

Android version comparison (cont'd)

- To deep dive into each change, initially you need to move to each source directory and use git log command with appropriate option. Here listed some useful git log option.

  - --grep= 'keyword'
    - Set to find log including 'keyword'
    - You can find patch including the work "LTE" by setting --grep='LTE'
  - --author='author'
    - Set to find log that has 'author' in Author
    - You can find commit made by Samsung with setting author='@samsung.com'
  - -p : generate patch form output
  - -E : accept Extended regular expression

Example 2

Android version comparison (cont'd)

- If you want to find the change between android-2.3.7_r1 to android-4.0.1_r1 that include keyword "LTE", you can use following command.

```
munakata@muna-E420: ~/android/frameworks/base
munakata@muna-E420:~/android$ cd frameworks/base/
munakata@muna-E420:~/android/frameworks/base$ git log --no-merges ¥
> android-2.3.7_r1..android-4.0.1_r1 --grep='LTE'
```

- Last example is a bit advanced one. This will show android enhancement between 2.3.7_r1 (GB) to 4.0.1_r1 (ICS) to add support for SMP processor.

```
munakata@muna-E420: ~/android
munakata@muna-E420:~/android$ repo forall -p -c ' ¥
> if git rev-parse android-2.3.7_r1 > /dev/null 2>&1; ¥
> then git log --no-merges -E --grep="SMP|barrier" android-2.3.7_r1..android-4.0.1_r1; ¥
> else git log --no-merges -E --grep="SMP|barrier" android-4.0.1_r1 ; ¥
> fi' > ~/smp.log
```

recommendation 2

Placing maintainer who owns strong practice authority

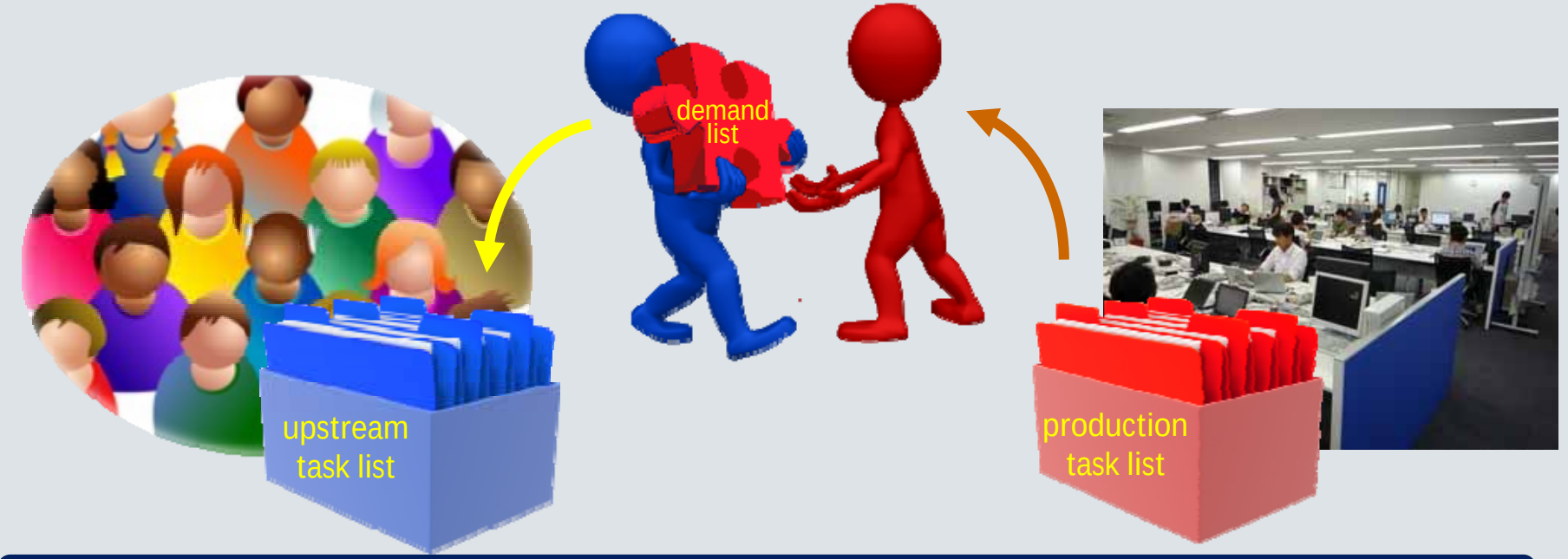
- All patches must be investigated thoroughly before merging it, and this is the process that is essential to avoid tree confusion.
- The tree maintainer needs to block all inappropriate code. Therefore he must have strong authority. It may be too hard for junior engineer.

Maintainer need to reject any request that include bad code.

recommendation 3

red team and blue team coordination

- The perspective of the blue team is totally different from the red team. However, it is useful for both sides to share the demand of the red team. This can be a trigger to add industry demand into future kernel.



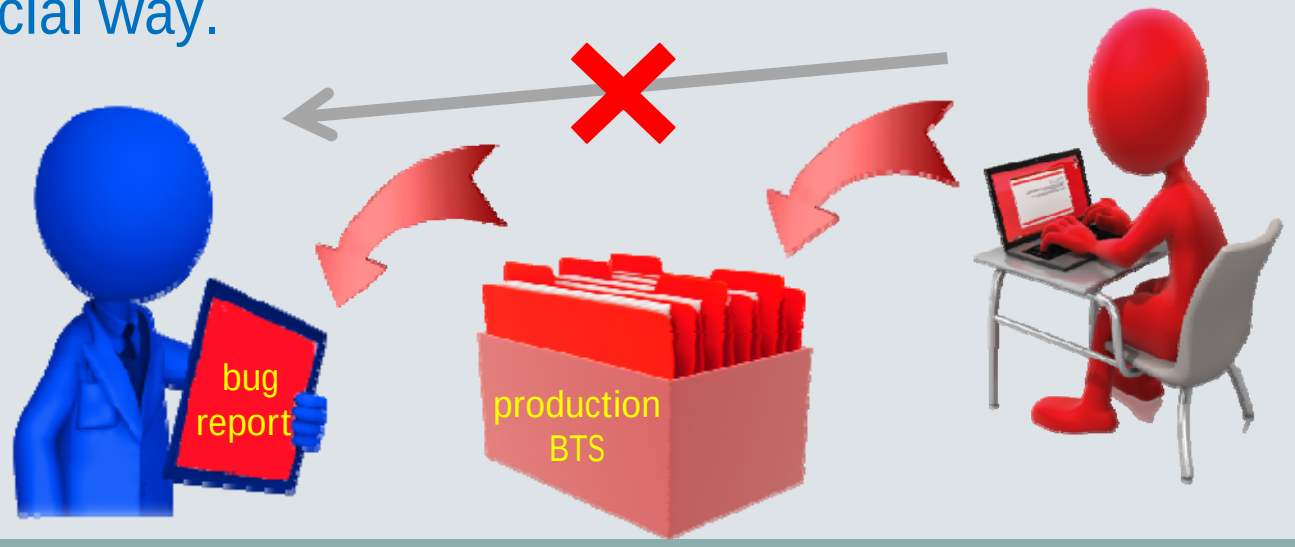
Red team can share production team demands to Blue team

recommendation 4

Disclosure of red team BTS contents

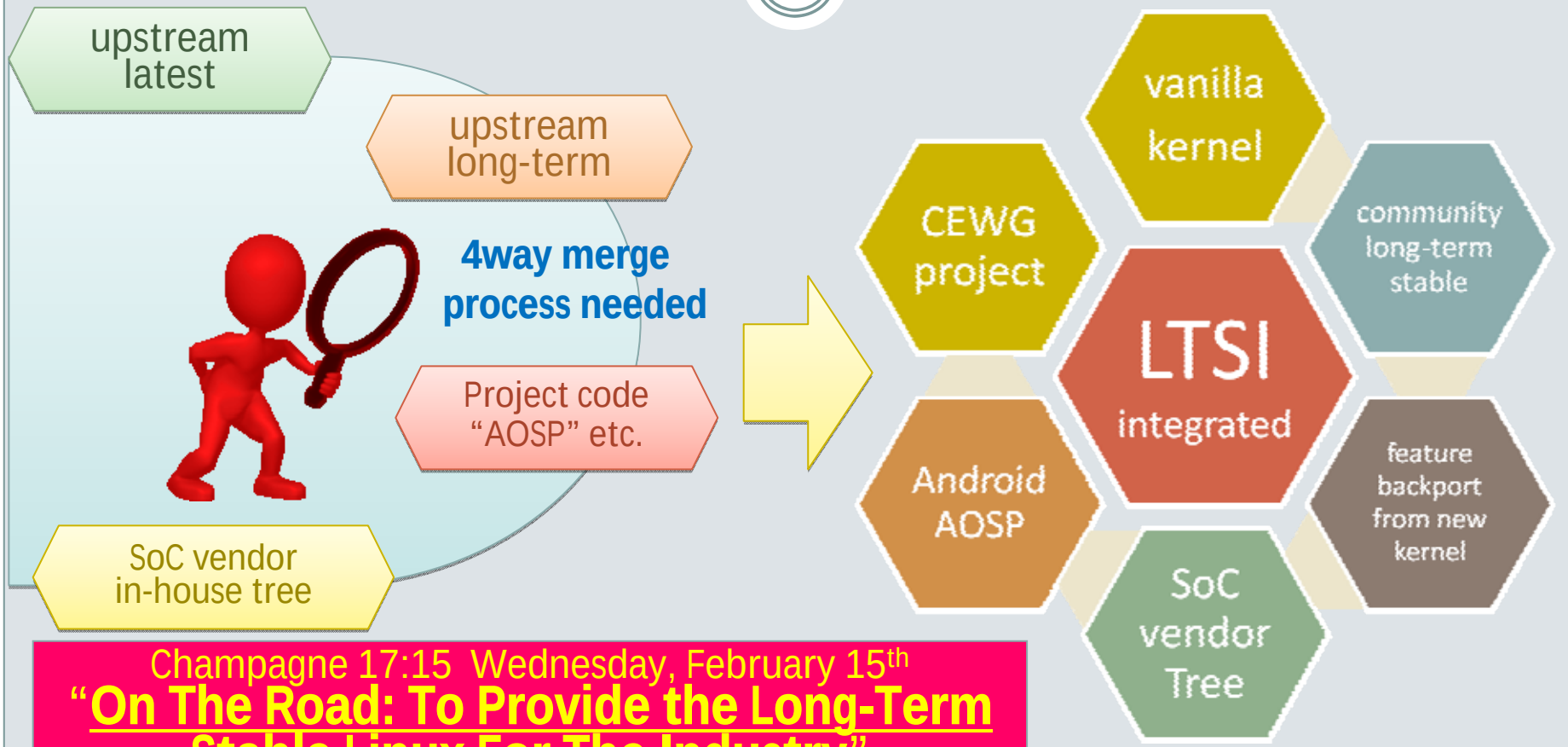
- Community development takes time as it will require coordination.
- Product development always requires immediate bug-fix.
- Therefore it does not make sense to use single BTS for both party

However,... The same problem will occur again if they do not add bug-fix code to the community upstream. So bug information share between production(red) and upstream(blue) team would be beneficial way.



recommendation 5

“LTSI” as common ground



Champagne 17:15 Wednesday, February 15th

“**On The Road: To Provide the Long-Term Stable Linux For The Industry**”

LTSI is Linux Foundation project to provide integrated stable kernel

Conclusion

- It is a very good trends that product development team come to use relatively new kernel because it shorten the distance to the upstream community.
- However, it does not seem to be able to be understood how the production developer can utilize the published community resources.
- It is understandable that production development team prefer their own way of development. However it is essentially important to study and adopt already proven open source development process, e.g. “full utilization of *git*”, “adoption of Linux upstream code review process”.