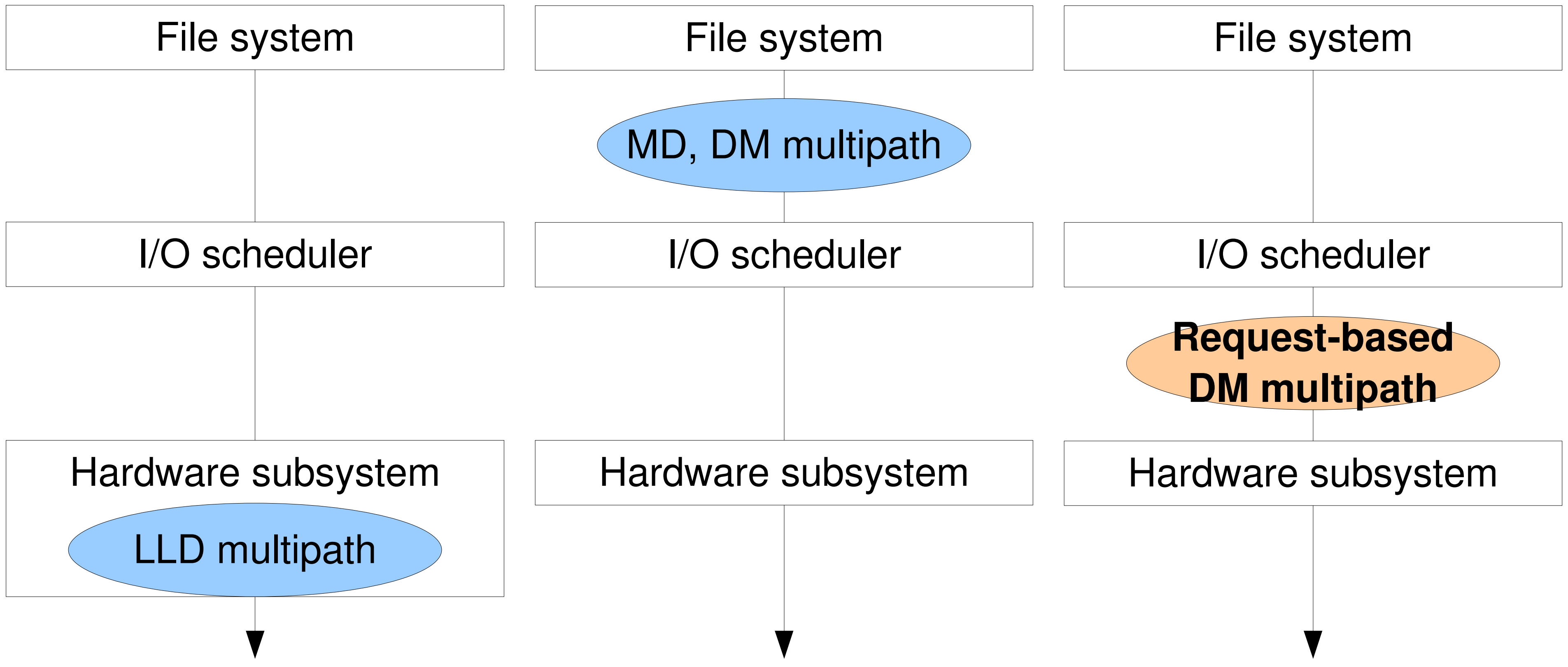
Request-based multipath (Request-based Device-mapper Multipath)

Kiyoshi Ueda
Jun'ichi Nomura
(NEC Corporation)

Contents

- Goal and basic design of the feature
- Current status
- Today's discussion topics
  - Issue1: How to avoid deadlock during completion?
  - Issue2: How to keep requests in mergeable state?
  - Issue3: How to hook completion for stacking driver?
- Background of each issue
- Solution for each issue

Multipath implementations in Linux



Different implementations on different hardware...

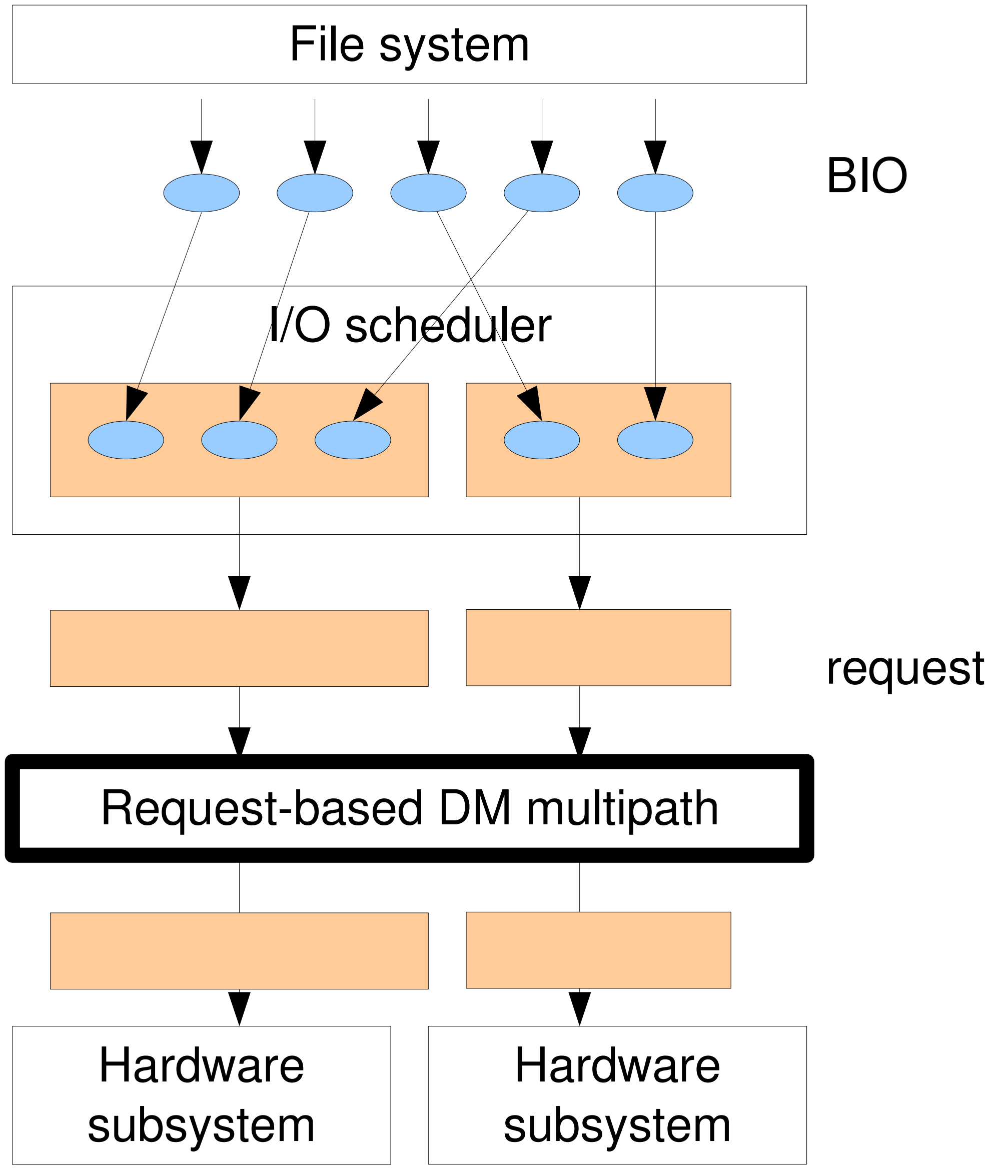
Generic implementation, but cannot get correct info. about I/O load...

taking the Best of both!

Request-based DM multipath

- Choose path when pulling request from queue

- BIOs are already merged
- Can obtain exact count of I/O units (that allows better load-balancing decision)



Request-based DM multipath

- The goal of the feature
  - Do path selection below/after the I/O scheduler
- Current design
  - Keep the user space (dm) interface same
  - Use __elv_add_request() for submission
  - Restructure the completion procedure
    - blk_end_request() as a single driver interface
    - Other problems to be solved (today's topic)

Current Status

- Consensus in LSF'07: do multipath at request-level
- OLS'07: Basic design and evaluation results
- Status of patches
  - Block layer changes
    - blk_end_request interface: will be included in 2.6.25
    - **Request stacking framework: RFC proposal**

Today's Topic

  - Device-mapper changes
    - Request-based dm core: tentative patches available
    - Request-based dm-multipath: tentative patches available
    - Dynamic load balancer: tentative patches available
  - Multipath-tools changes (No changes required)

Today's topics: Issues in request stacking

- What is request stacking?
  - Submitting a request in a stacking driver's queue to lower queue (after cloning)
  - Calling back the stacking driver when completing the request
- Issues

  Issue1: How to avoid deadlock during completion?

  Issue2: How to keep requests in mergeable state?

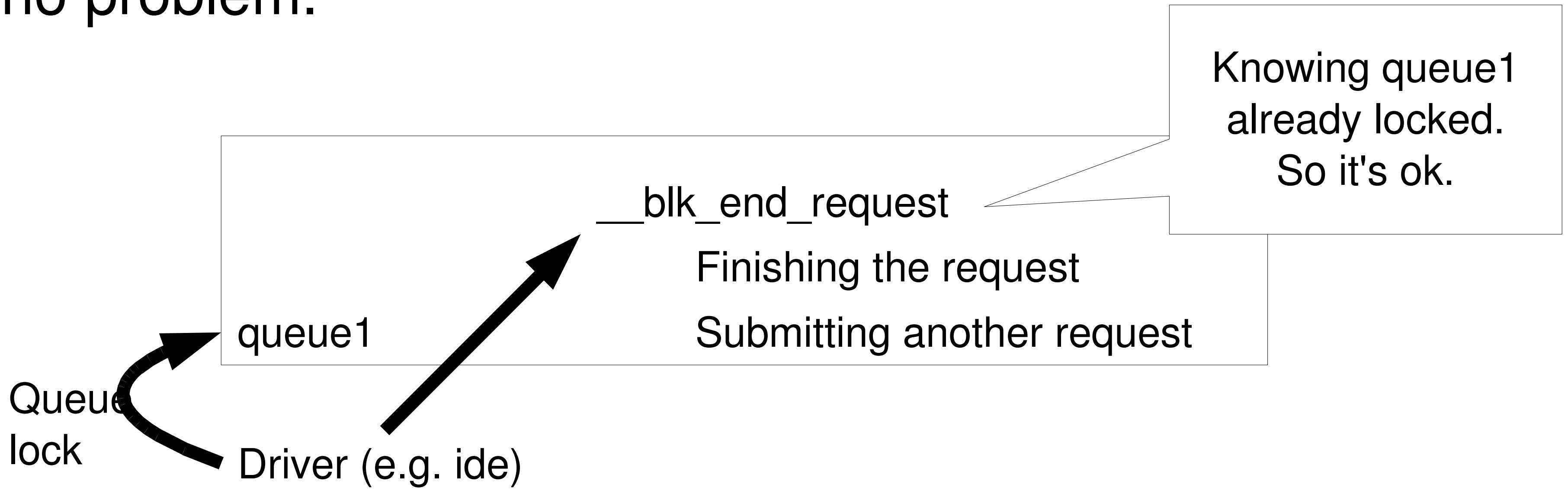
  Issue3: How to hook completion for stacking driver?

Issue1: How to avoid deadlock during completion?

- Drivers using __blk_end_request() will deadlock
  - __blk_end_request() means the queue lock is held through the completion process
  - During the completion, upper device may want to hold the queue lock
    - To submit another request
    - To finish the completing request
- Fortunately(?) the biggest user of dm-multipath is scsi and scsi doesn't have the problem

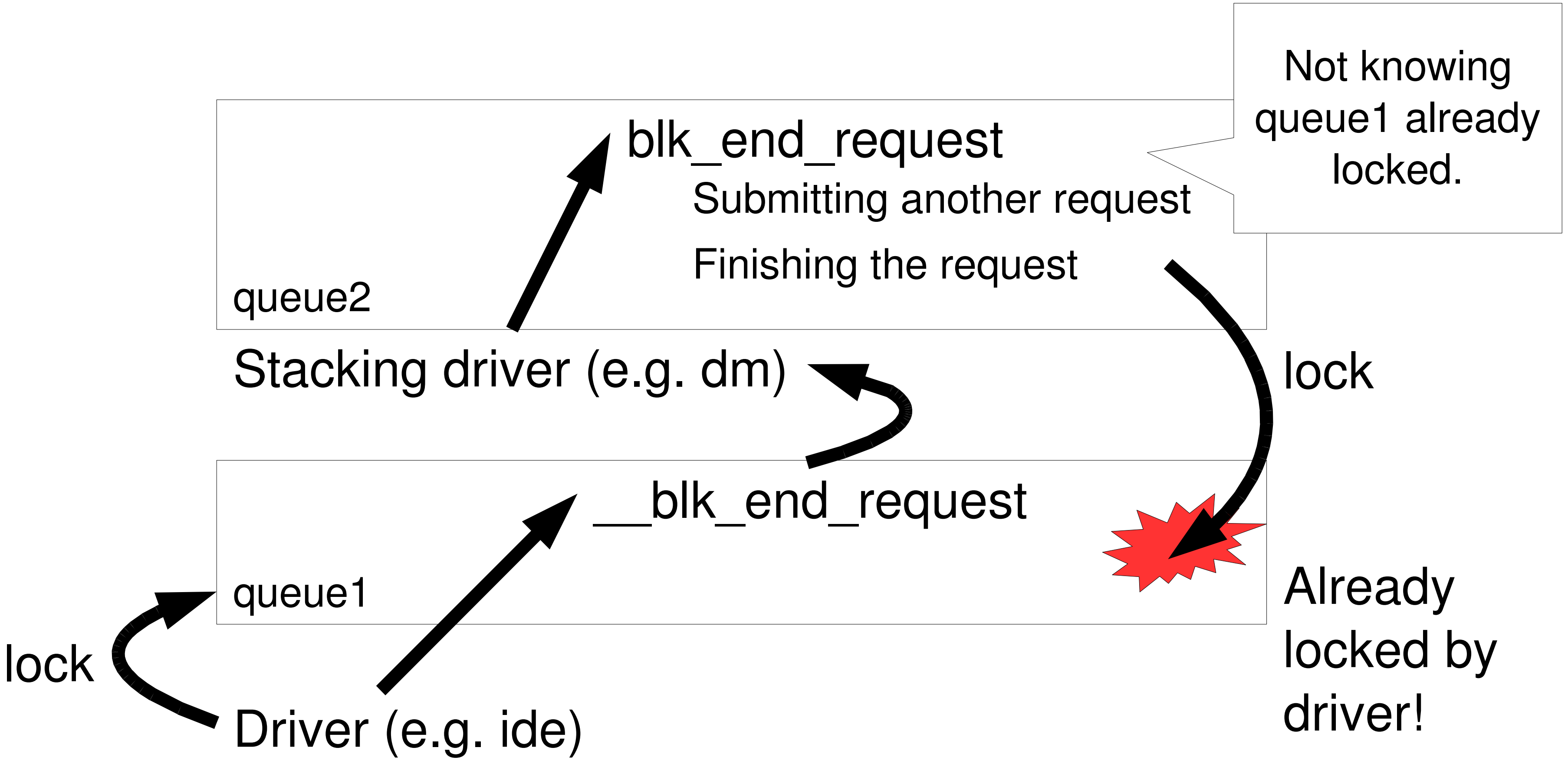
Queue locking (normal)

__blk_end_request() knows the queue is already locked.
So no problem.



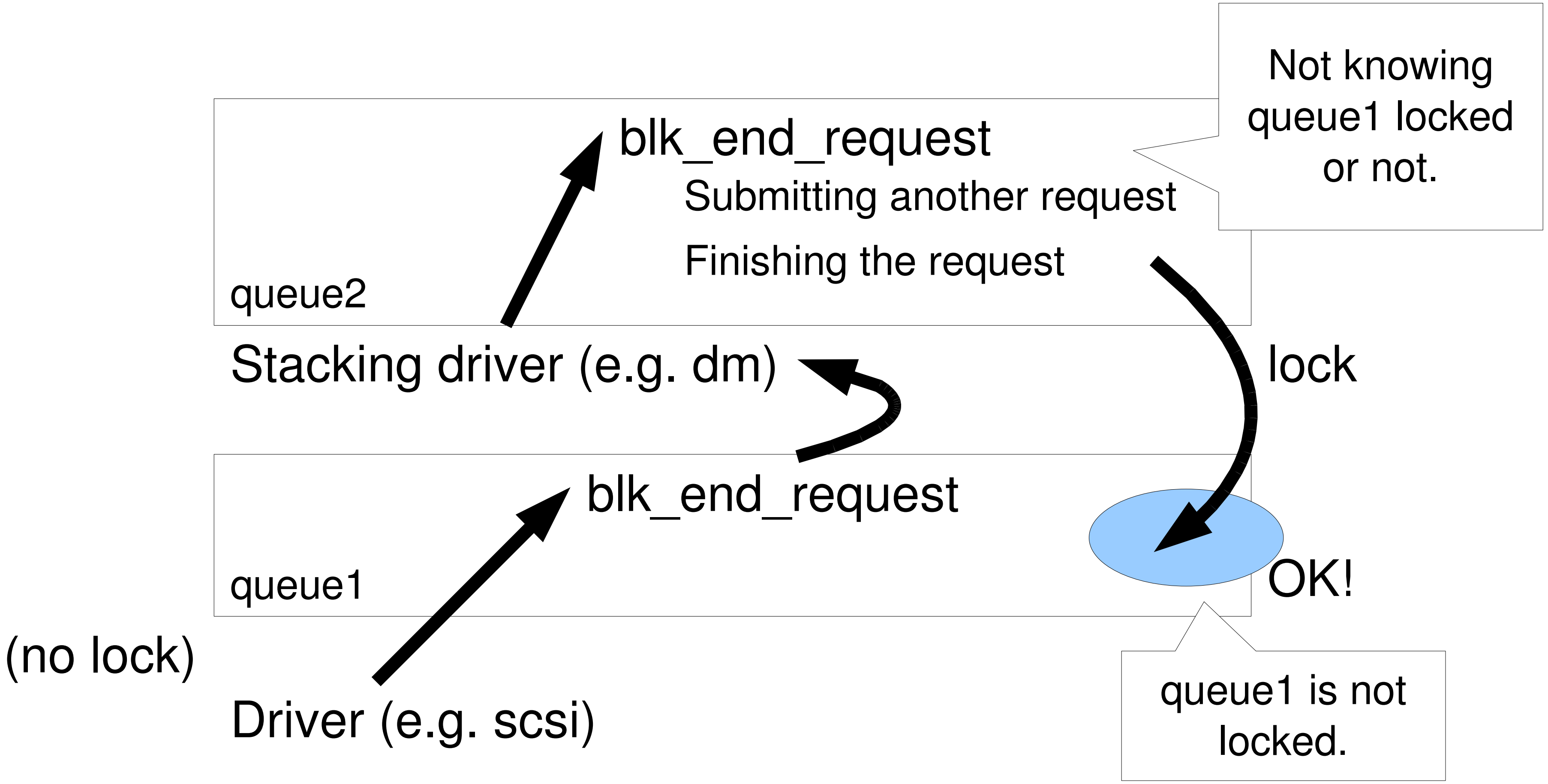
Queue locking (stacked)

Stacking driver doesn't know whether the bottom-level queue is locked. So deadlock will happen.


blk_end_request
Submitting another request
Finishing the request
Not knowing queue1 already locked.
queue2
Stacking driver (e.g. dm)
lock
__blk_end_request
queue1
Already locked by driver!
lock
Driver (e.g. ide)

Queue locking (stacked)

If the driver uses blk_end_request(), no problem.



Issue2: How to keep requests in mergeable state?

- Once a request is pulled from the queue and sent to device, the request has no chance of merge
- Timing of the pull is controlled by:
  - plug/unplug controls whether driver can try to pull the request off from the queue
  - Driver decides whether to pull the requests after checking if device is busy
- What if the queues are stacked?
  - (Cont. to the next slide)

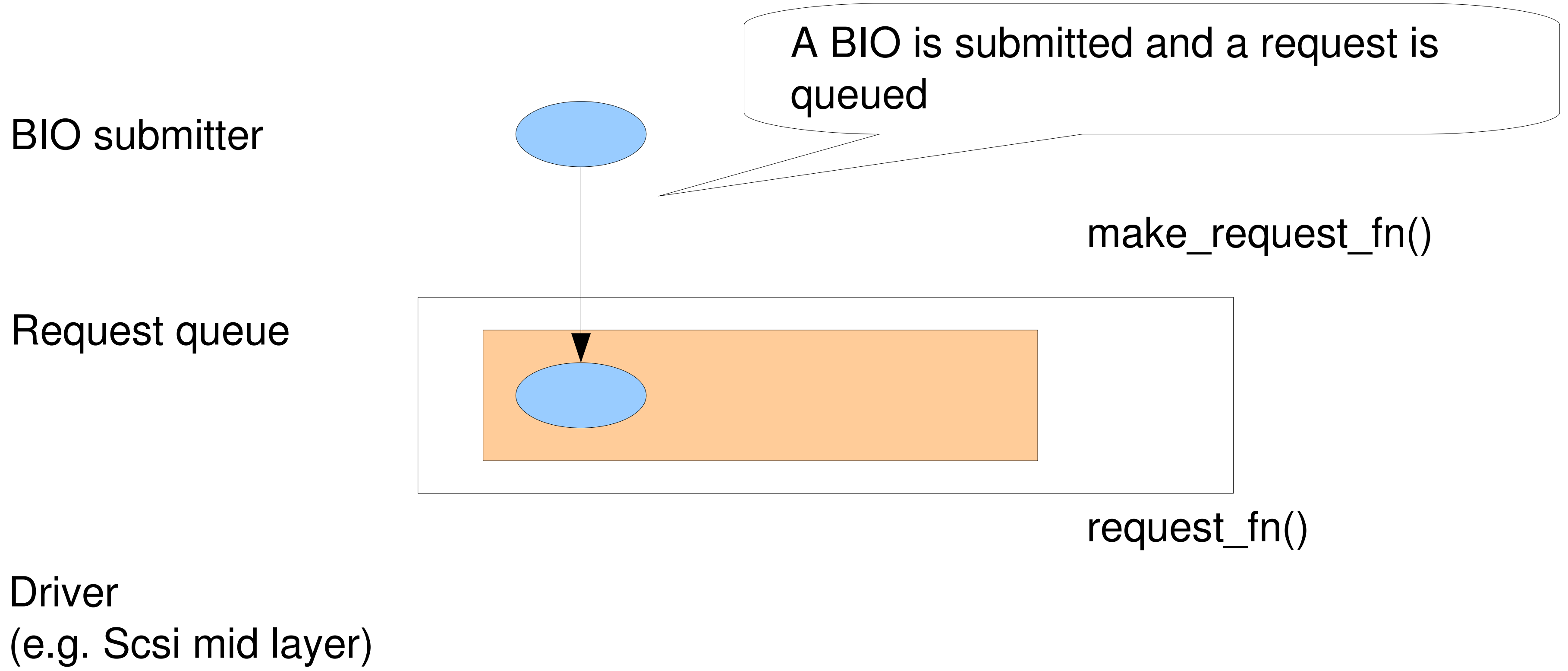
Issue2: How to keep requests in mergeable state?

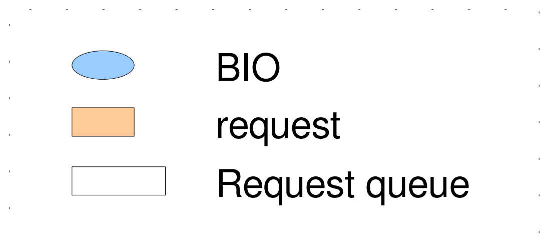
- If the queues are stacked
  - The upper driver doesn't know whether (the bottom level) device is busy
  - So the request is pulled whenever the queue is unplugged
  - But if the device is busy, the pulled request will stay in the lower queue without a change of merge

=> Less merge, worse throughput

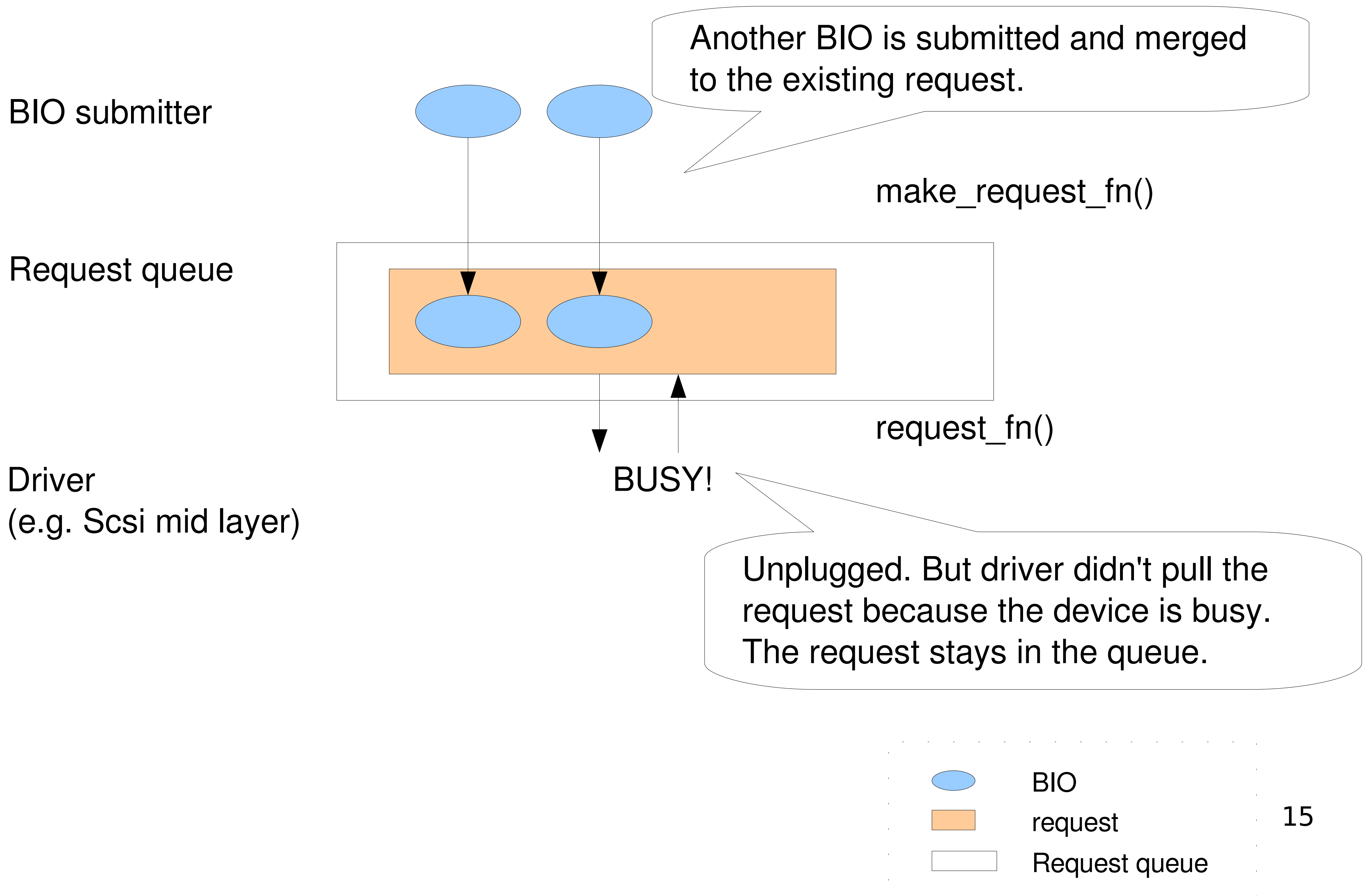
(Cont. to next slides for an example)

Device busy check (normal) [1/3]

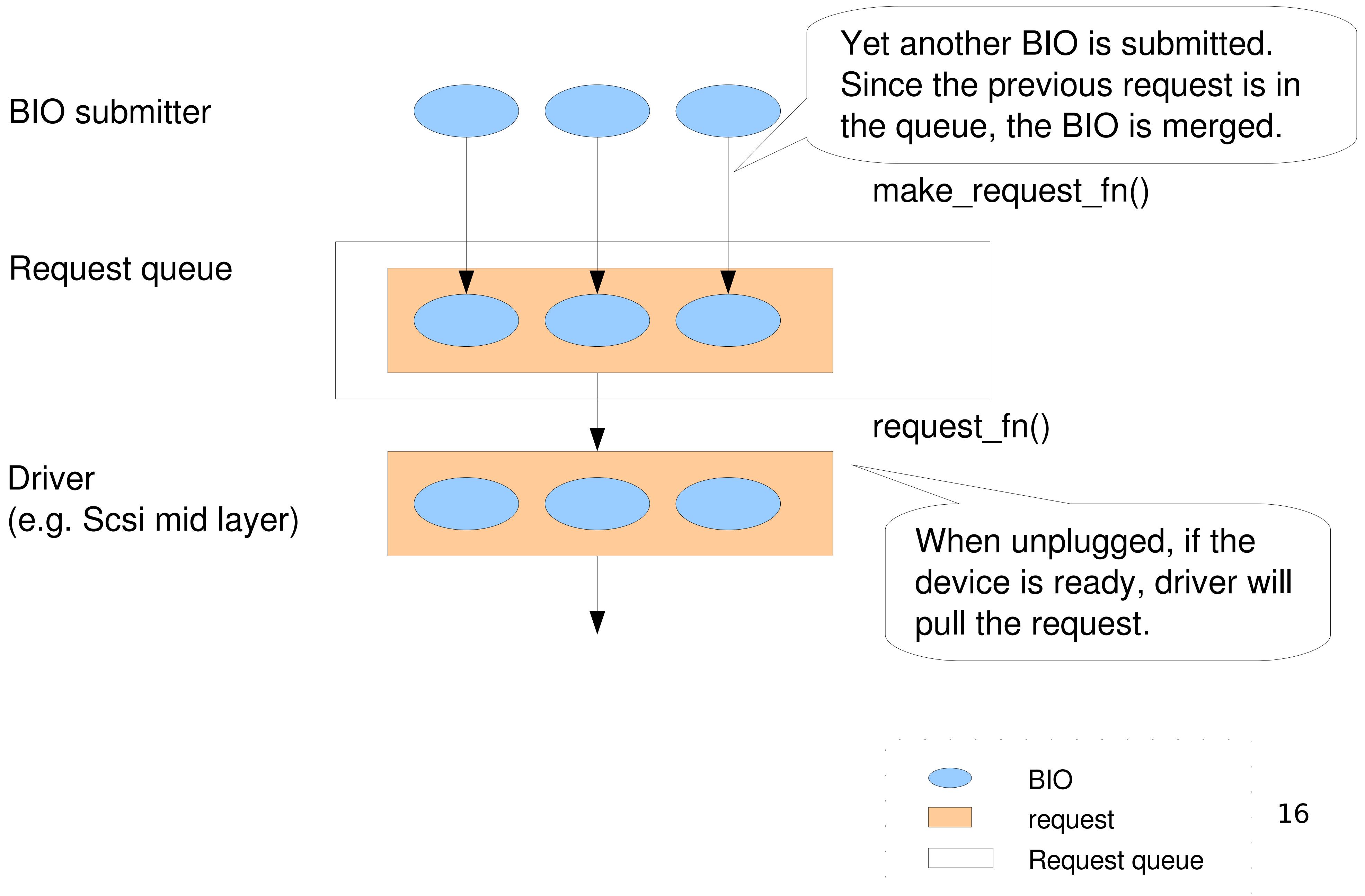




Device busy check (normal) [2/3]


Another BIO is submitted and merged to the existing request.
BIO submitter
make_request_fn()
Request queue
request_fn()
Driver
(e.g. Scsi mid layer)
BUSY!
Unplugged. But driver didn't pull the request because the device is busy. The request stays in the queue.
BIO
request
Request queue

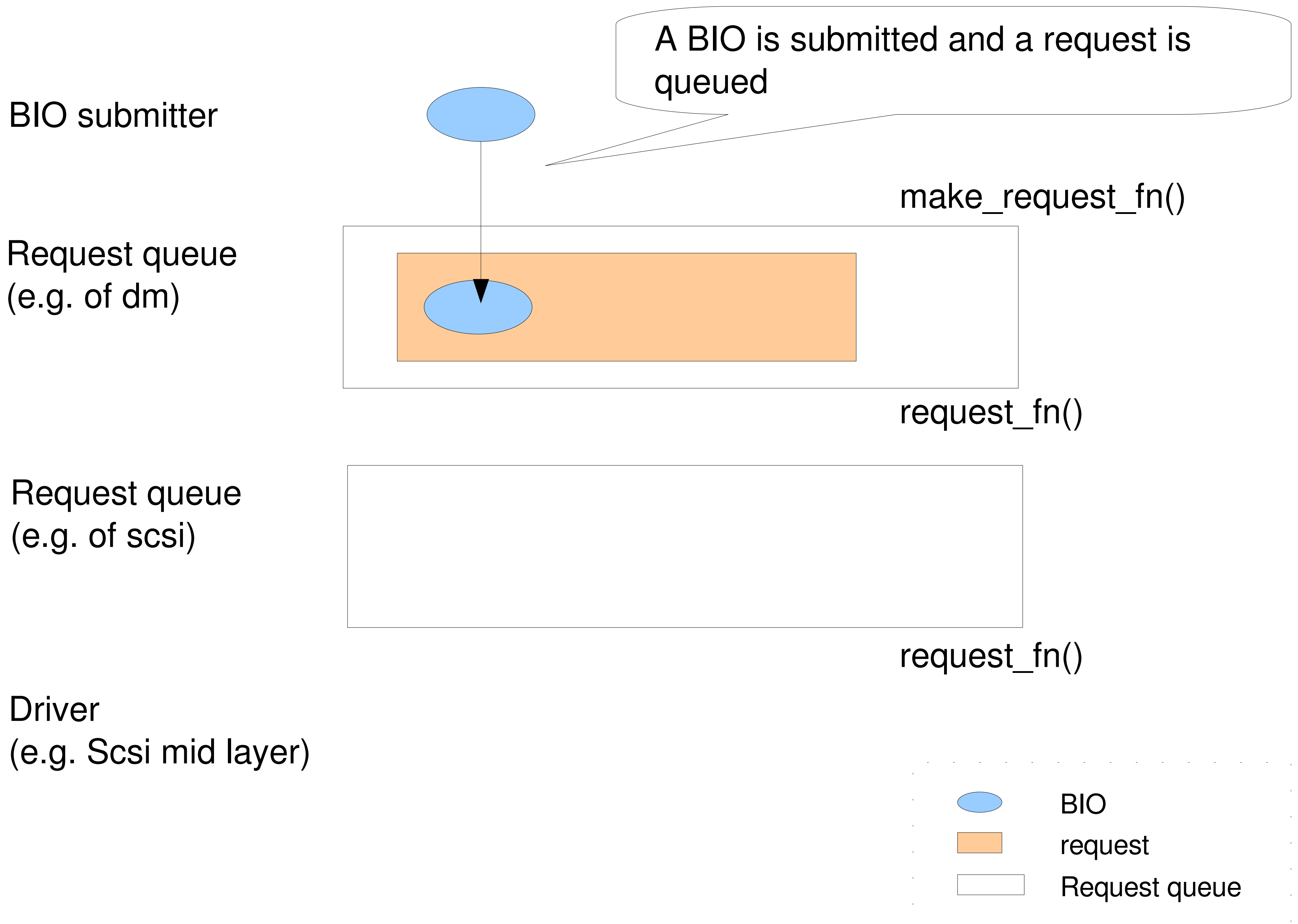
Device busy check (normal) [3/3]



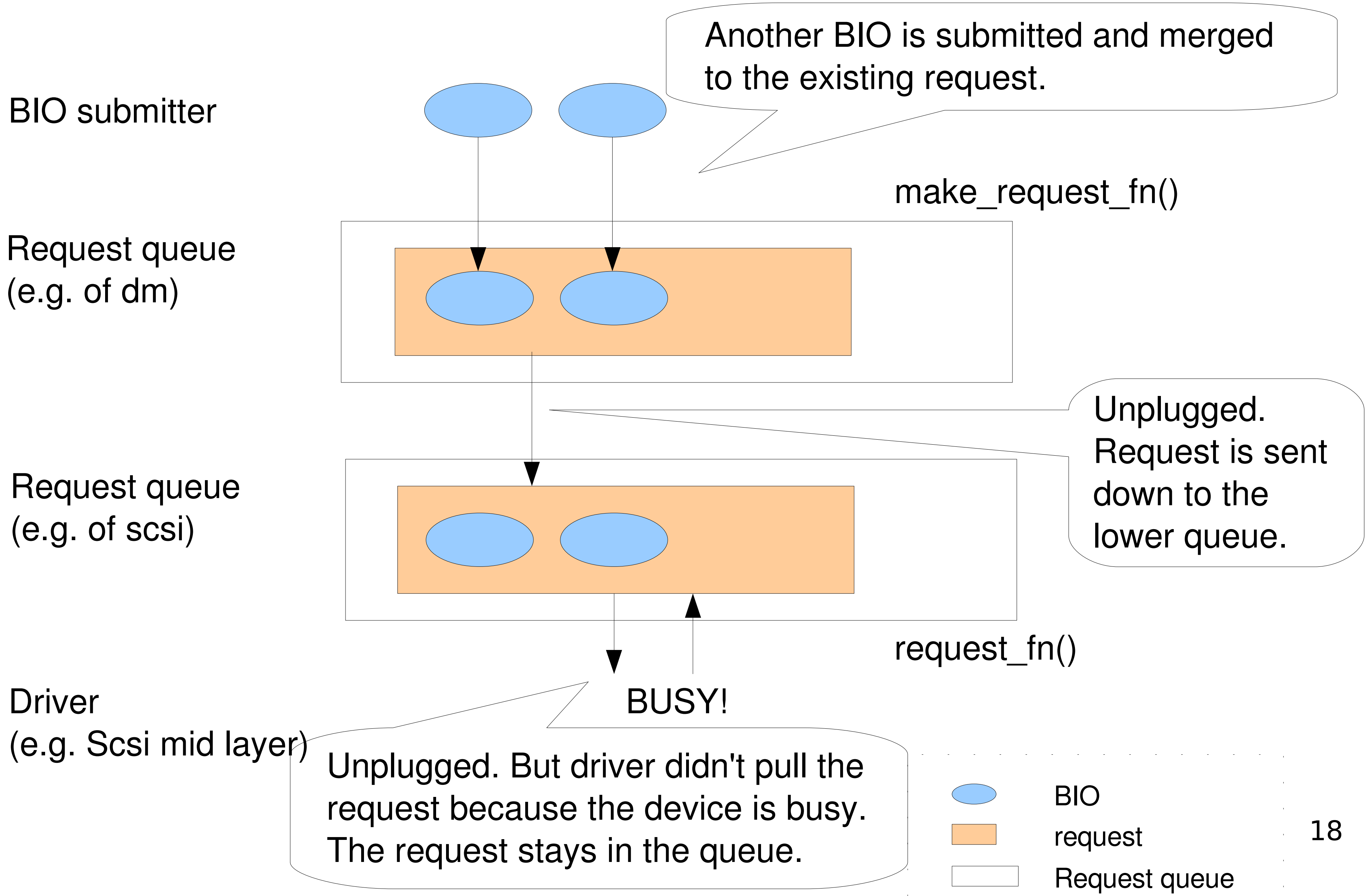
Yet another BIO is submitted. Since the previous request is in the queue, the BIO is merged.

When unplugged, if the device is ready, driver will pull the request.

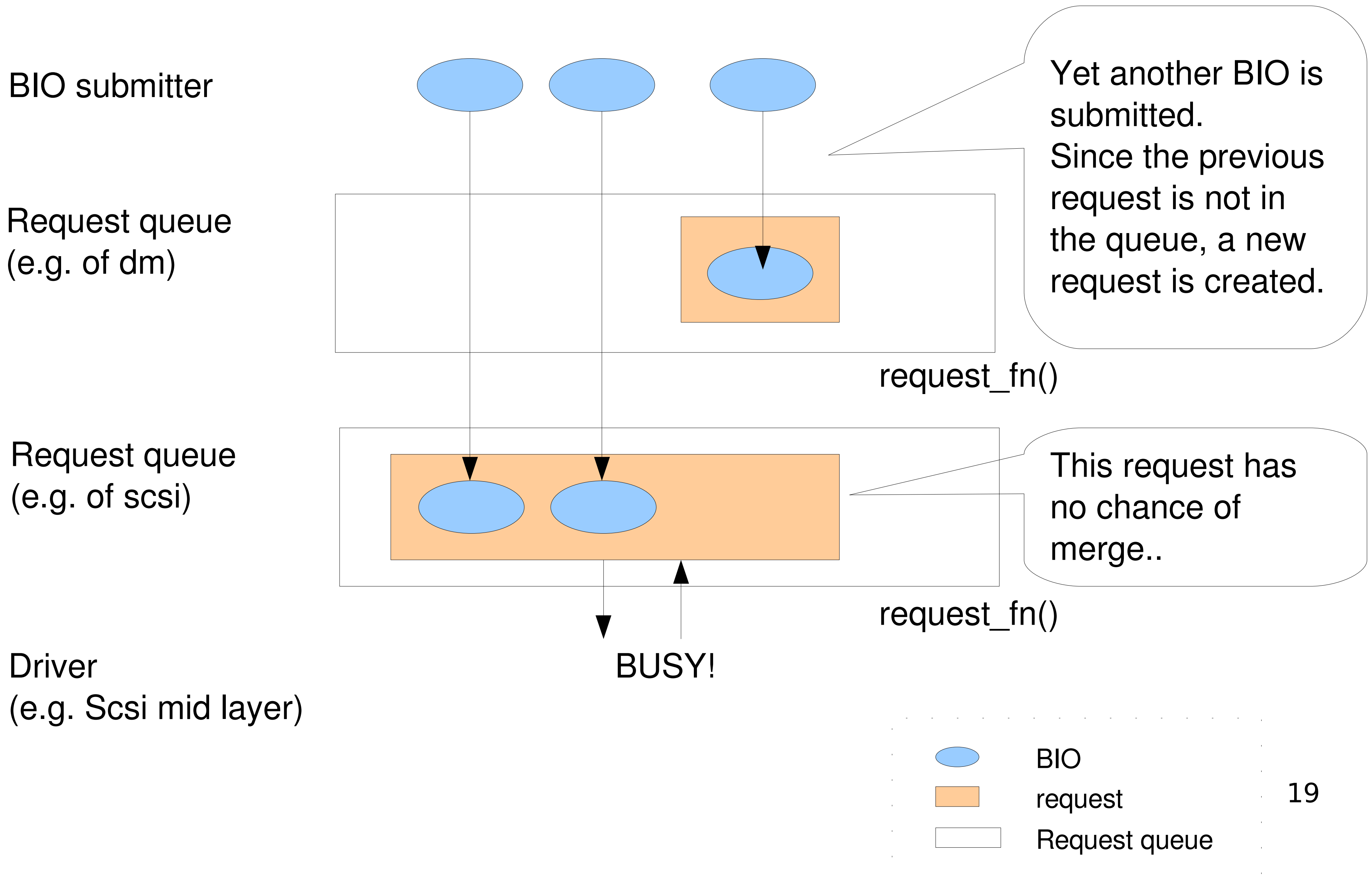
Device busy check (stacked) [1/4]



Device busy check (stacked) [2/4]


Another BIO is submitted and merged to the existing request.
BIO submitter
make_request_fn()
Request queue (e.g. of dm)
Unplugged. Request is sent down to the lower queue.
Request queue (e.g. of scsi)
request_fn()
BUSY!
Driver (e.g. Scsi mid layer)
Unplugged. But driver didn't pull the request because the device is busy. The request stays in the queue.
BIO
request
Request queue

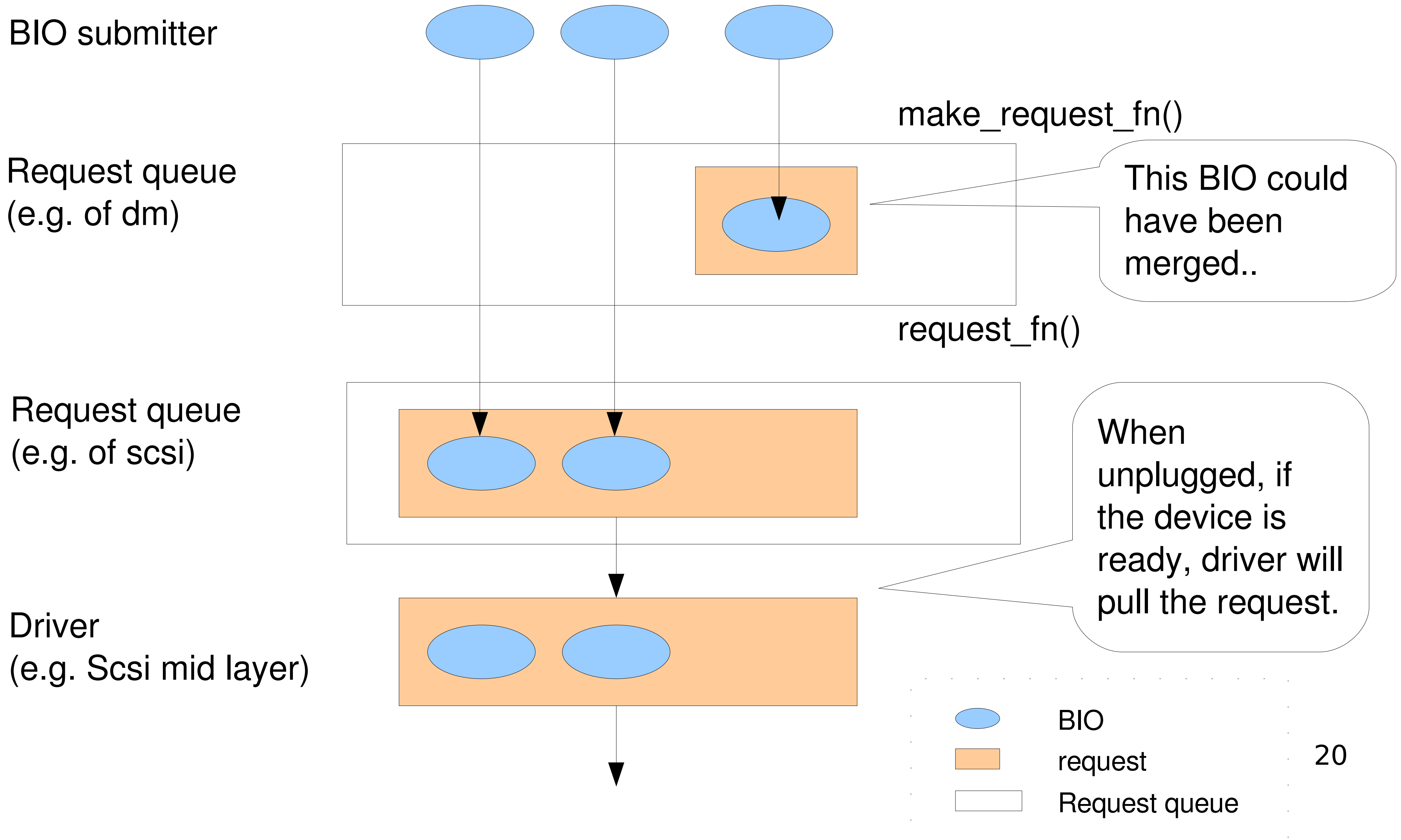
Device busy check (stacked) [3/4]


BIO submitter
Request queue (e.g. of dm)
Request queue (e.g. of scsi)
Driver (e.g. Scsi mid layer)
request_fn()
request_fn()
BUSY!
BIO
request
Request queue

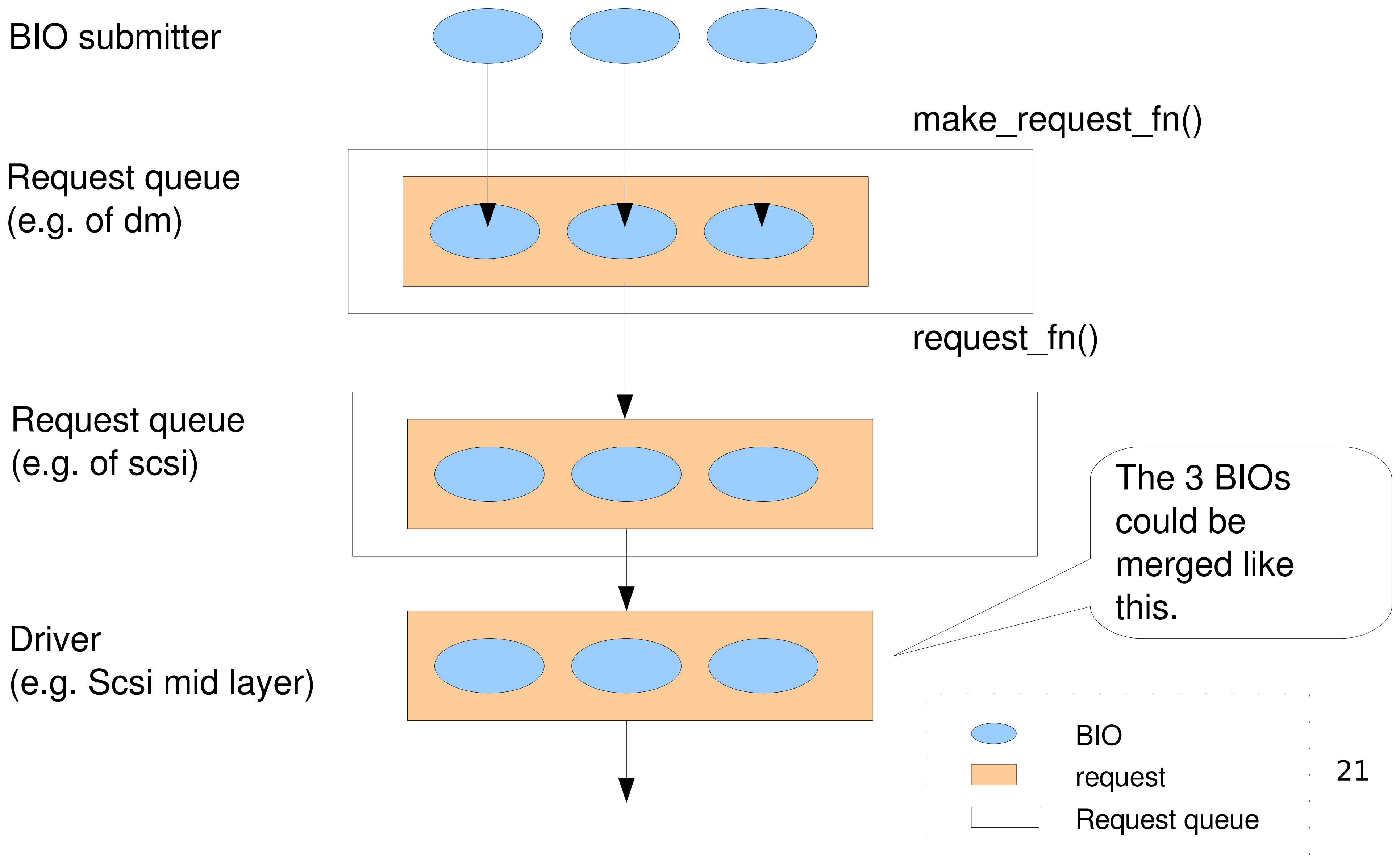
Yet another BIO is submitted.
Since the previous request is not in the queue, a new request is created.

This request has no chance of merge..

Device busy check (stacked) [4/4]



Device busy check (stacked)



Issue3: How/where to hook completion for stacking driver?

Before discussing about the hook,

Review the request completion process.

(Cont.)

Request completion

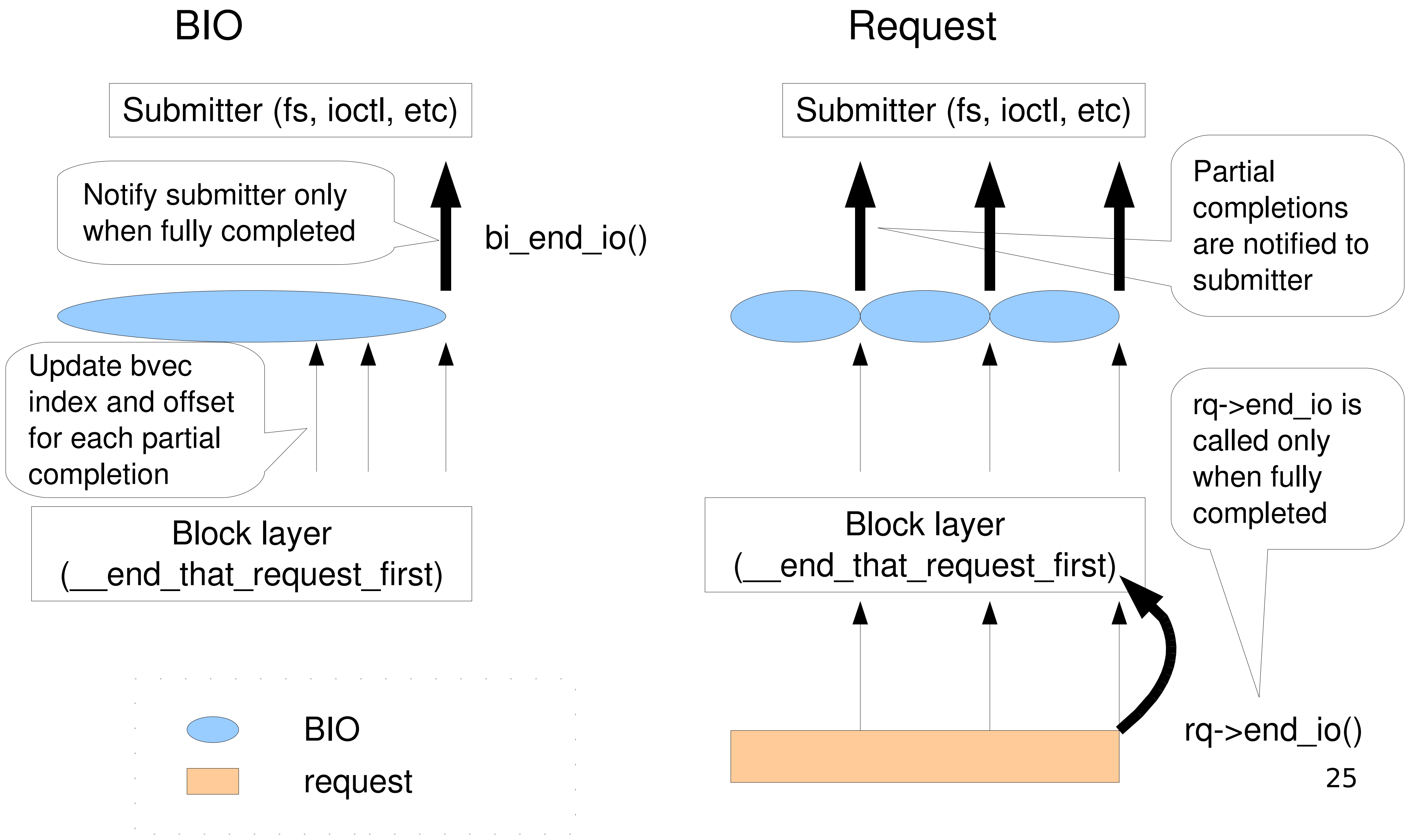
Request completion includes 2 parts:

1. Update the BIO's bvec index and offset and notify the submitter when fully completed
2. If all BIOs are done, update the status of request queue, release the request and notify submitter of the request

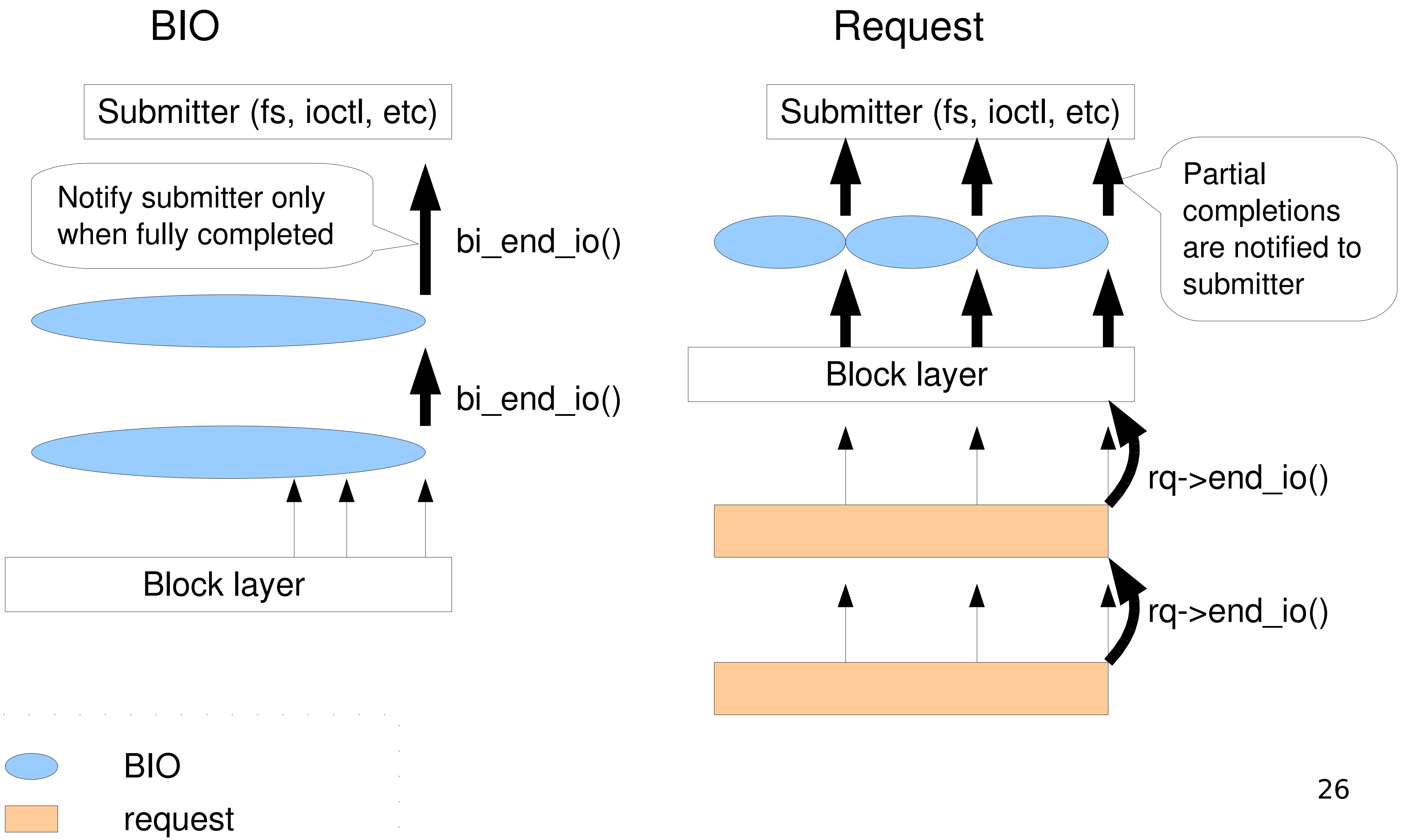
Differences of bio and request

- Bio
  - Completion is notified to the upper layer only when the BIO is fully completed.
  - Device locking is not required for both submission and completion.
- Request
  - Completion is notified to the upper layer even if the request is partially completed.
  - Device locking (queue lock) is required for both submission and completion.

bi_end_io and rq->end_io (normal)


BIO
Request
Submitter (fs, ioctl, etc)
Notify submitter only when fully completed
bi_end_io()
Update bvec index and offset for each partial completion
Block layer (__end_that_request_first)
Submitter (fs, ioctl, etc)
Partial completions are notified to submitter
rq->end_io is called only when fully completed
Block layer (__end_that_request_first)
rq->end_io()
BIO
request

bi_end_io and rq->end_io (stacked)


BIO
Request
Submitter (fs, ioctl, etc)
Notify submitter only when fully completed
bi_end_io()
bi_end_io()
Block layer
Submitter (fs, ioctl, etc)
Partial completions are notified to submitter
Block layer
rq->end_io()
rq->end_io()
BIO
request

Issue3: How/where to hook completion for stacking driver?

- rq->end_io() is called too late during the completion process
  - It's called after the completion is notified to submitters of BIOs.
- rq->end_io() is called with queue lock held
  - So we have the deadlock problem same as Issue1

Solutions for each issue

Issue1: How to avoid deadlock during completion?

Issue2: How to keep requests in mergeable state?

Issue3: How to hook completion for stacking driver?

Solutions for issue1 (deadlock)

A) Allow stacking only for non-locking drivers

Proposed as RFC

- Reject stacking on locking drivers, which use __blk_end_request() (non-locking drivers use only blk_end_request())
  => Always call the stacking hook without queue lock
    => No deadlock on finishing request
    => No deadlock on submission during completion

B) No submission during completion

- Allow stacking on locking drivers, too
  => Deadlock on submission during completion is unavoidable
    (E.g. Submitting to device B during the completion for device A)
    => Can't submit any request during completion
- Add 2 stacking hooks for locking/non-locking drivers so that stacking driver can know whether the queue is locked or not
  => Can avoid deadlock on finishing request on the queue

Issue1-A) Allow stacking only for non-locking drivers

- Summary
  - Allow stacking only for drivers not using __blk_end_request(). Drivers using __blk_end_request() are unstackable.
  - No deadlock on both finishing and submission during completion
    => Another request submission during completion is available
    (but the request may be submitted to other driver's device)
  - Current stackable drivers: scsi, cciss, i2o
- Needed work
  - Change the block layer not to use __blk_end_request()
    - barrier handling
    - error handling for drivers (BLKPREP_KILL)
  - Dasd driver change for existing dm-multipath users

Issue1-B) No submission during completion

- Summary
  - Allow request stacking on locking drivers, too
  - Can't avoid deadlock on submission between locking driver's 2 completion processes (“AB-BA” deadlock)
    => Can't submit any request during completion
      => Use workqueue or something: Performance concern
  - Deadlock on finishing request is avoidable by letting stacking hook know about the locking status of the queue lock:
    - Add 2 stacking hooks for locking/non-locking drivers
    - Add an argument of locking/non-locking to stacking hook
- Needed work
  - Pass any submission during the completion to workqueue
  - 2 implementations for 2 hooks or additional argument for the hook

Solutions for issue2 (busy check)

A) Export busy state via queue flag

Proposed as RFC

- Bottom level drivers must set/clear the flag appropriately
  - Bit operations. No extra lock overhead: Inexpensive
- Stacking drivers can check busy state of (bottom level) devices without calling (bottom level) drivers
  - Extra bit operation overhead: Inexpensive

B) Add busy state check function to queue

- Bottom level drivers set its own function
  - No extra overhead when request stacking is not used: Free
- Stacking drivers call it whenever dispatching a request
  - Busy check function may need lock: (Very) Expensive

Solutions for issue3 (stacking hook)

A) Add another hook

Proposed as RFC

- Add another hook for request stacking to the head of blk_end_request instead of using end_io

B) Move end_io calling place

- Move end_io to the head of blk_end_request

C) Use end_io as it is

- Use end_io with the existing calling place

Issue3-A) Add another hook

- Summary
  - Add another hook for request stacking to the head of blk_end_request() (not in __blk_end_request())
    => Stacking driver is always called without queue lock
        => Submission during completion is available
  - Don't use end_io() for request stacking.
  - No need to change existing end_io users
  - Stacking drivers are responsible for completion of the request against the queue/device
- Needed work
  - None

Issue3-B) Move end_io calling place

- Summary
  - Move end_io to the head of blk_end_request() and __blk_end_request() for existing end_io users
    => end_io could be called with/without queue lock held
    (existing end_io users need to care about the lock status)
  - Stacking drivers and existing end_io users share end_io and both are responsible for completion of the request on the queue
- Needed work
  - Existing end_io users need to be changed to take responsibility for whole completion of the request including data (bio) completion part, while they are interested in only request destruction part

Issue3-C) Use end_io as it is

- Summary
  - The block layer don't complete a request partially only for stacking driver (stacking driver can't do partial completion)
    => Performance concern
  - Stacking driver is always called with the queue lock held
    => Submission during completion is unavailable
  - Use end_io with the existing calling place
    => No need to change existing end_io users
  - Stacking driver is called after finishing request on the queue is done (No responsibility for finishing request on the queue)
- Needed work
  - Change the block layer not to complete bios in a request only for stacking drivers, while device drivers call blk_end_request for partial completion

Thank you

- RFC for issue1 (deadlock) and issue3 (hook)
  - http://lkml.org/lkml/2008/2/15/411
  - http://lkml.org/lkml/2008/2/15/412
  - http://lkml.org/lkml/2008/2/15/413
- RFC for issue2 (busy check)
  - http://lkml.org/lkml/2008/2/15/416