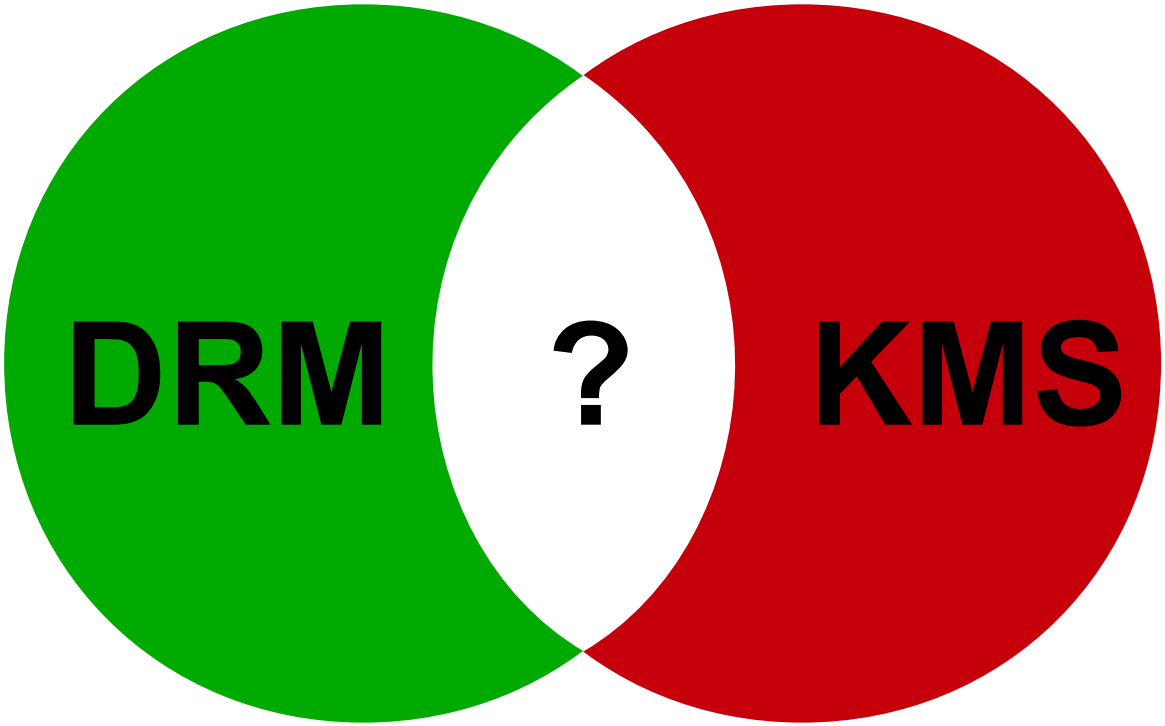
Anatomy of an Atomic KMS Driver

Embedded Linux Conference Europe 2015
Dublin

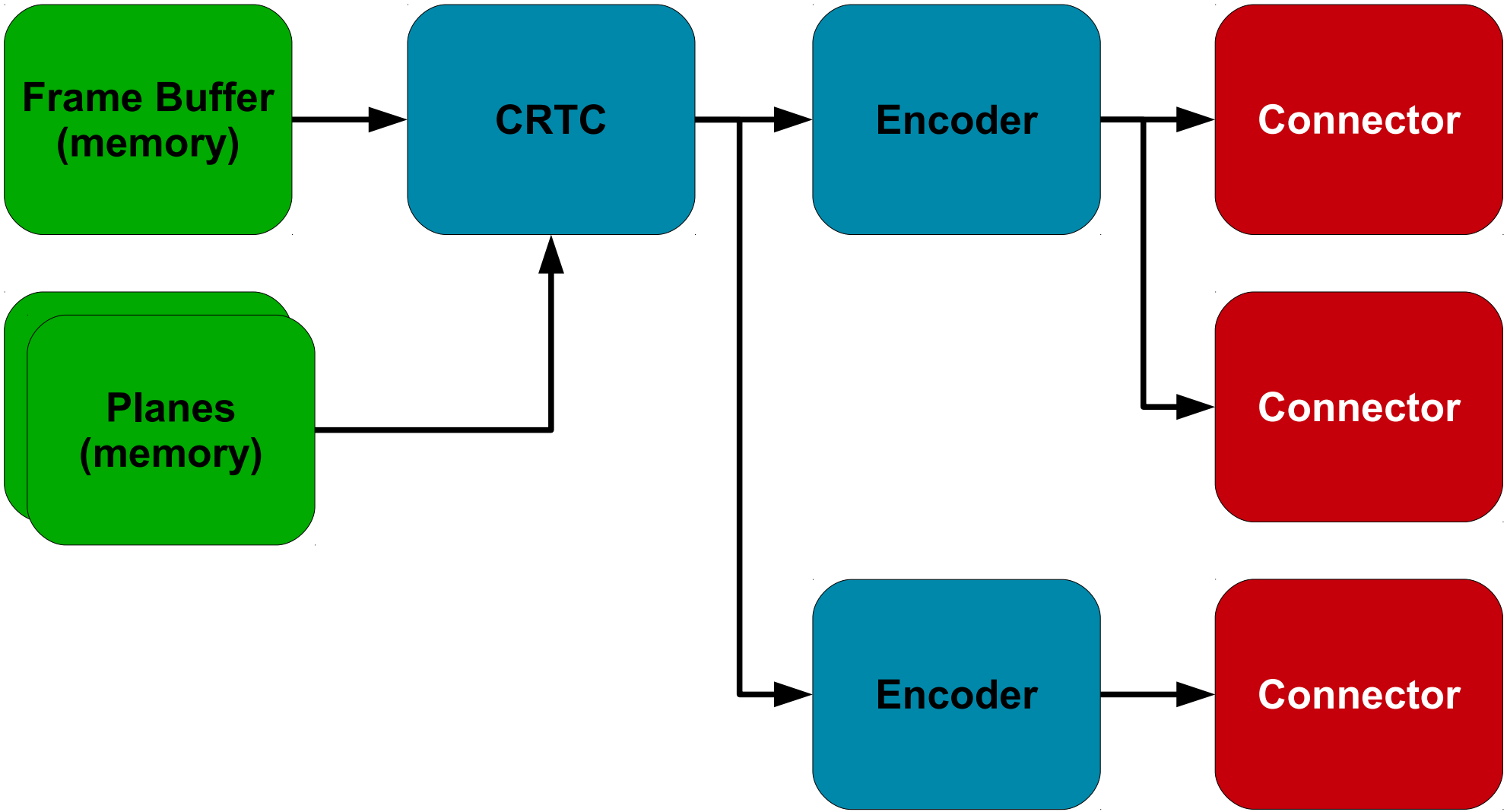
Laurent Pinchart
laurent.pinchart@ideasonboard.com



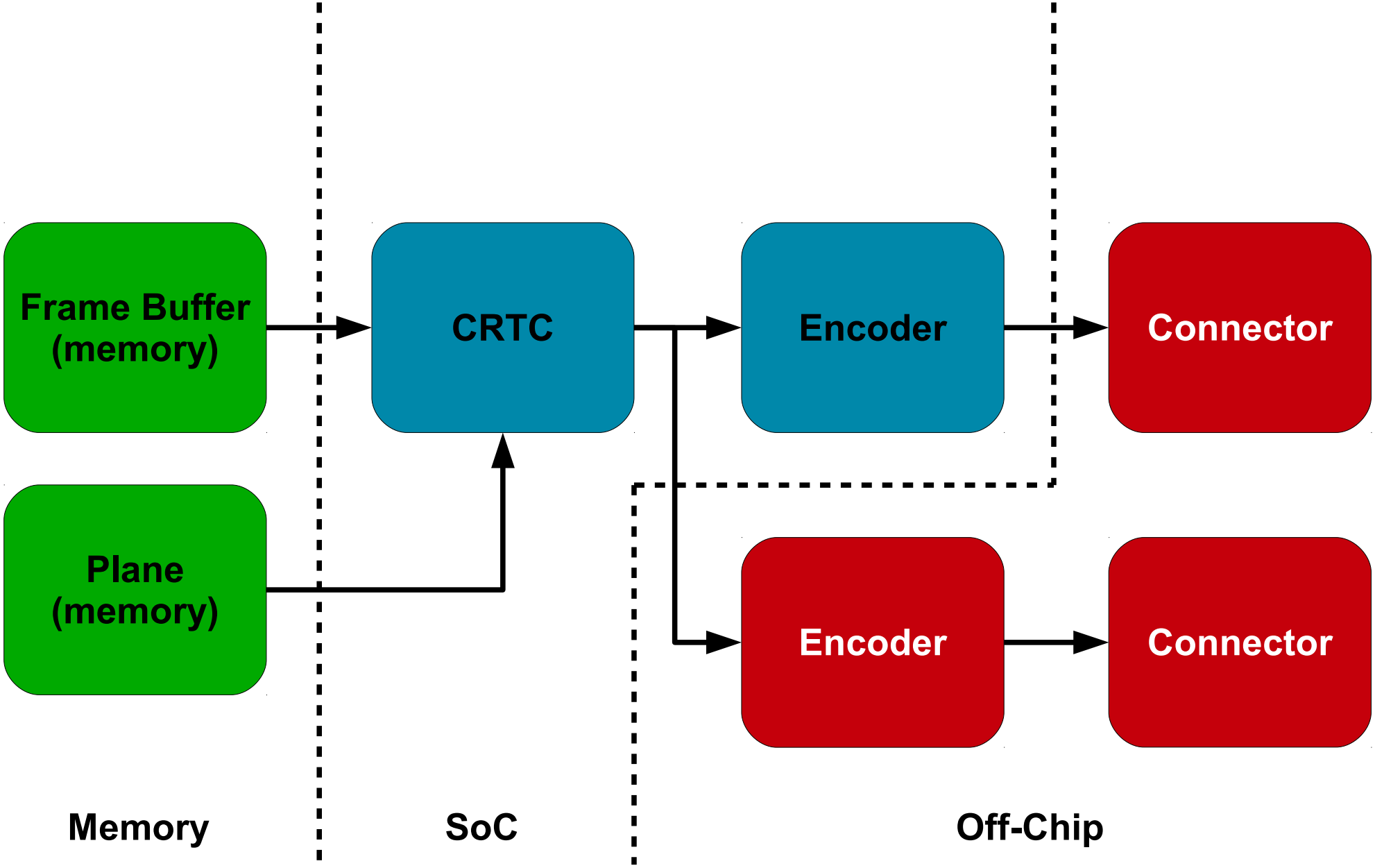
APIs

- Memory Management
- Vertical Blanking
- Version, Authentication, Master, ...

- Device Model
- Frame Buffer
- Modes
- Page Flip
- Planes
- Cursor, Gamma, ...

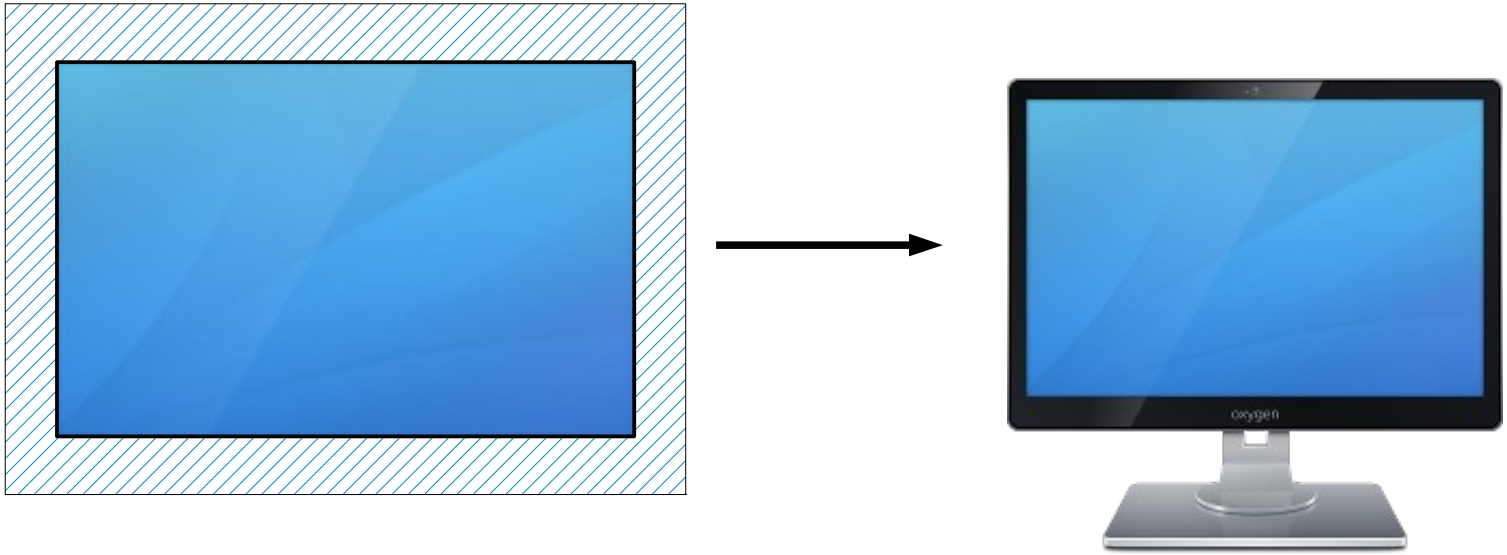


Device Model

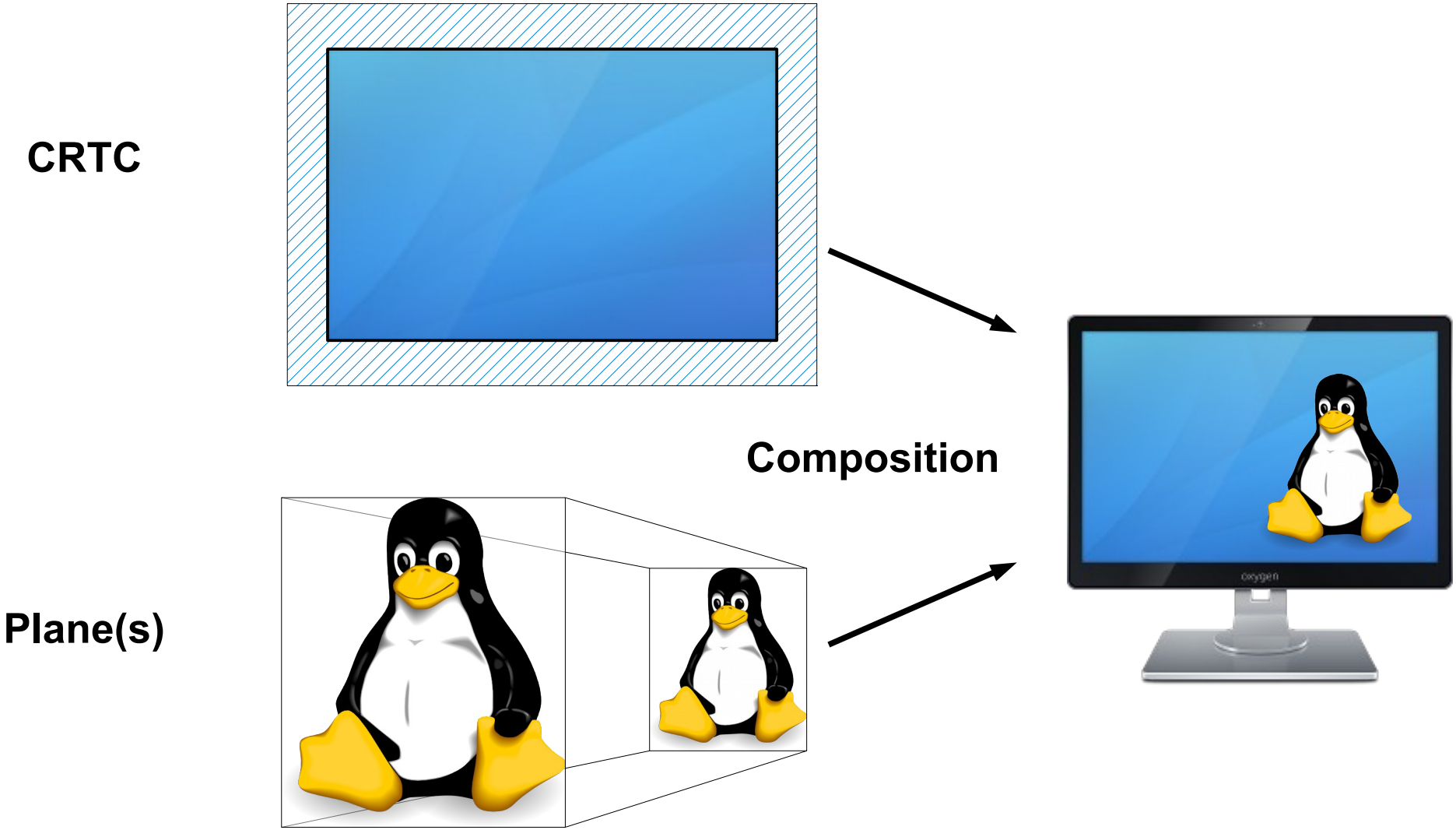


Device Model - SoC

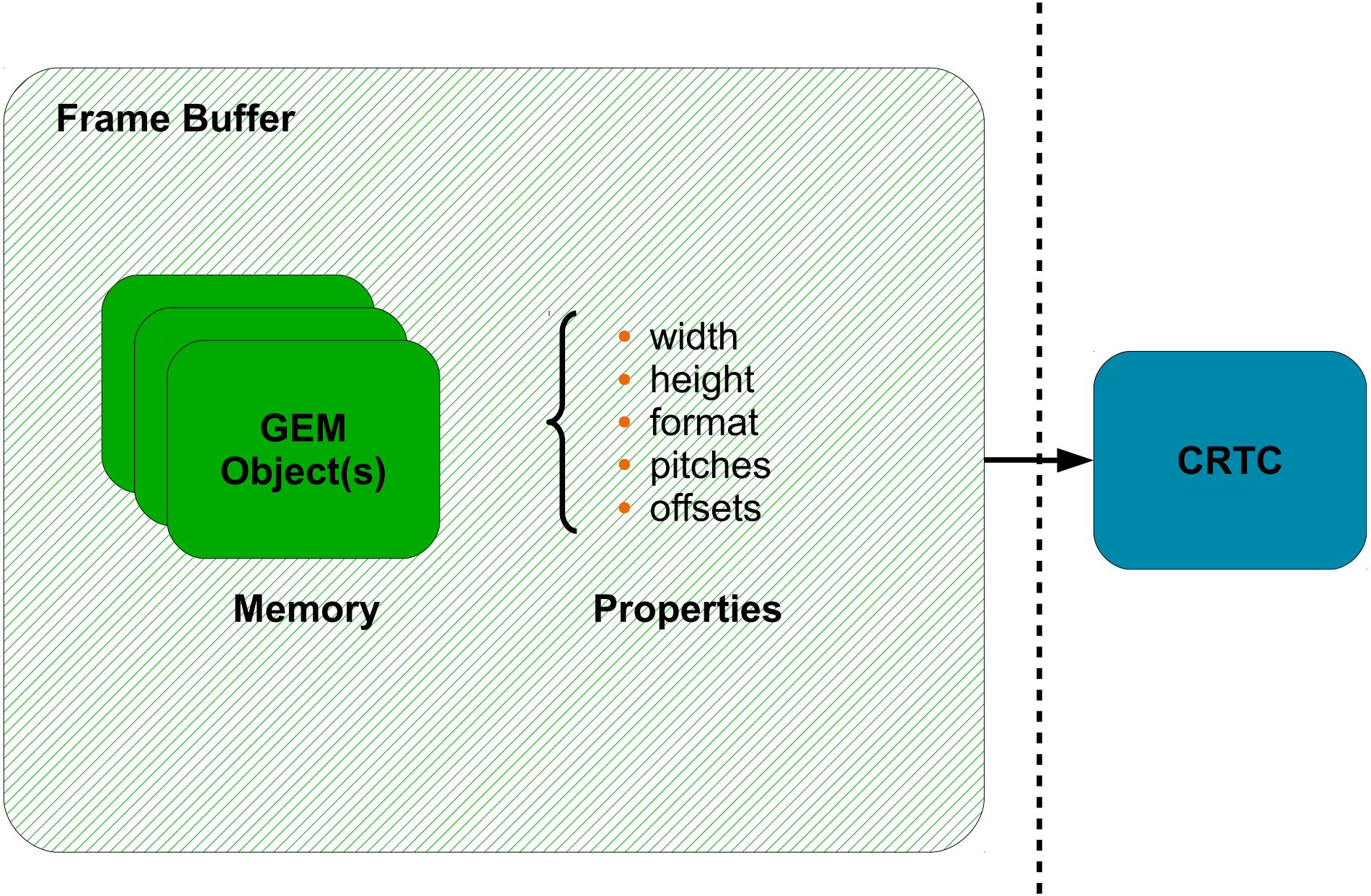
Frame Buffer



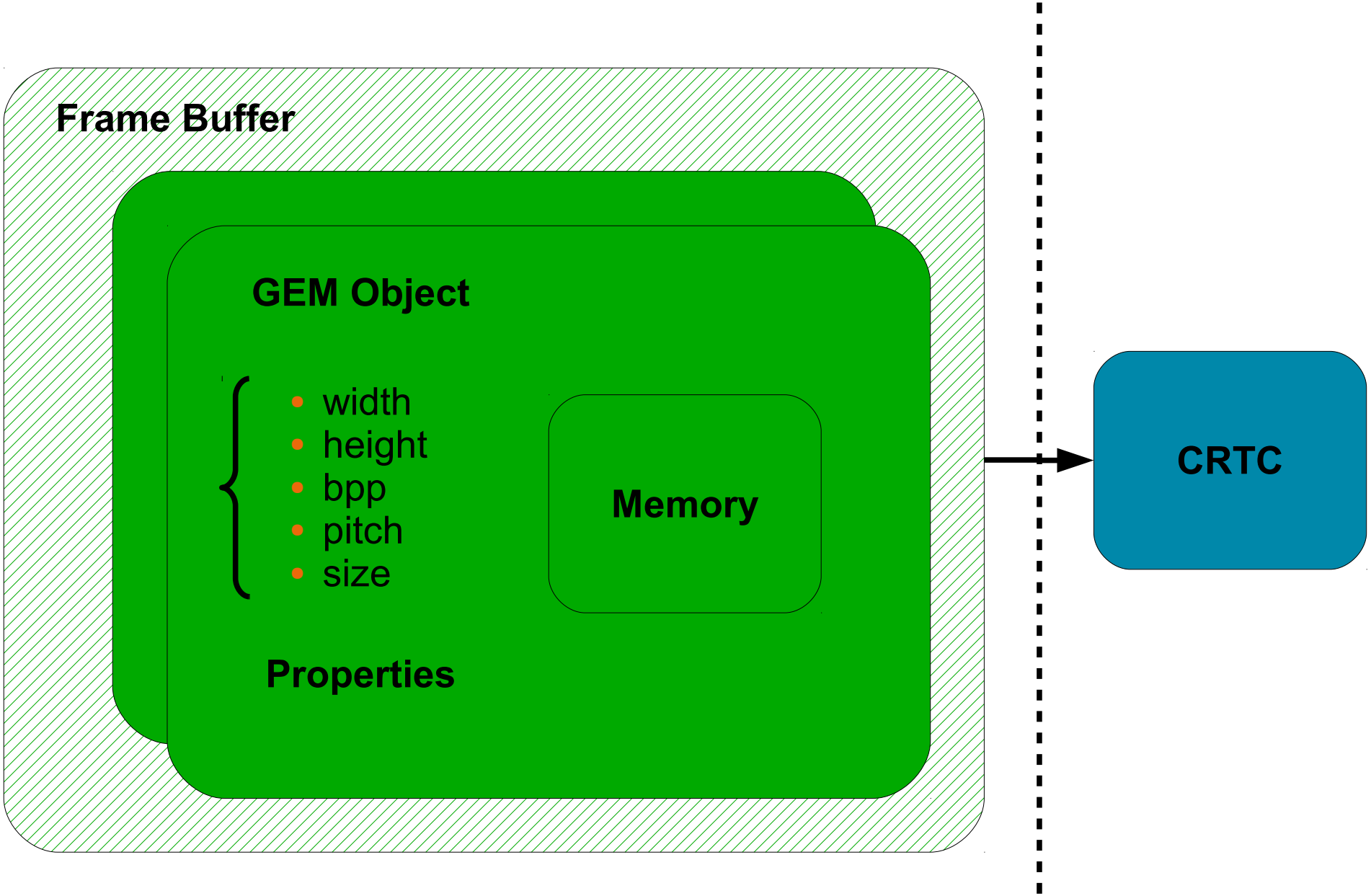
KMS – Scanout



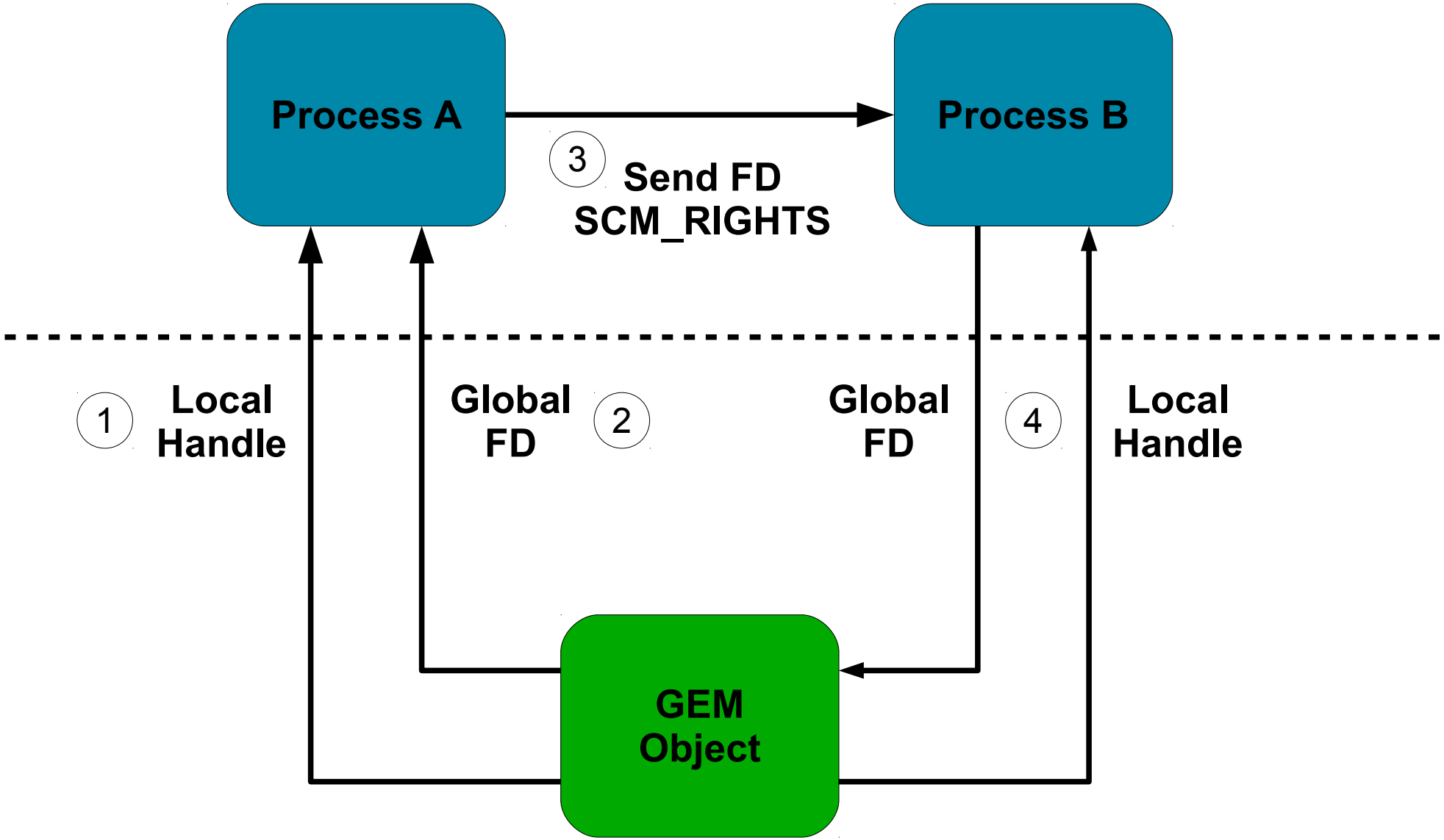
KMS – Composition



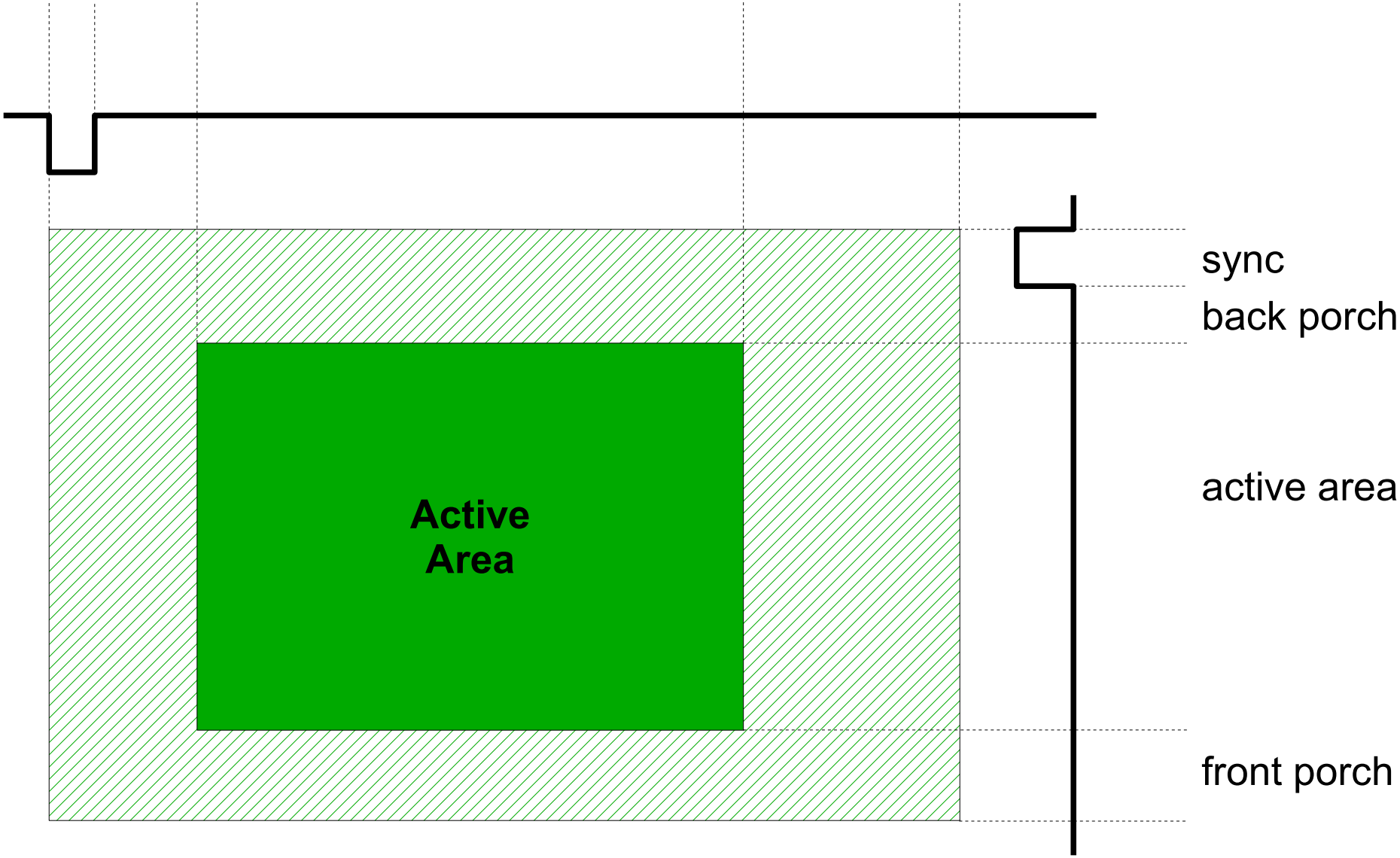
KMS – Frame Buffer



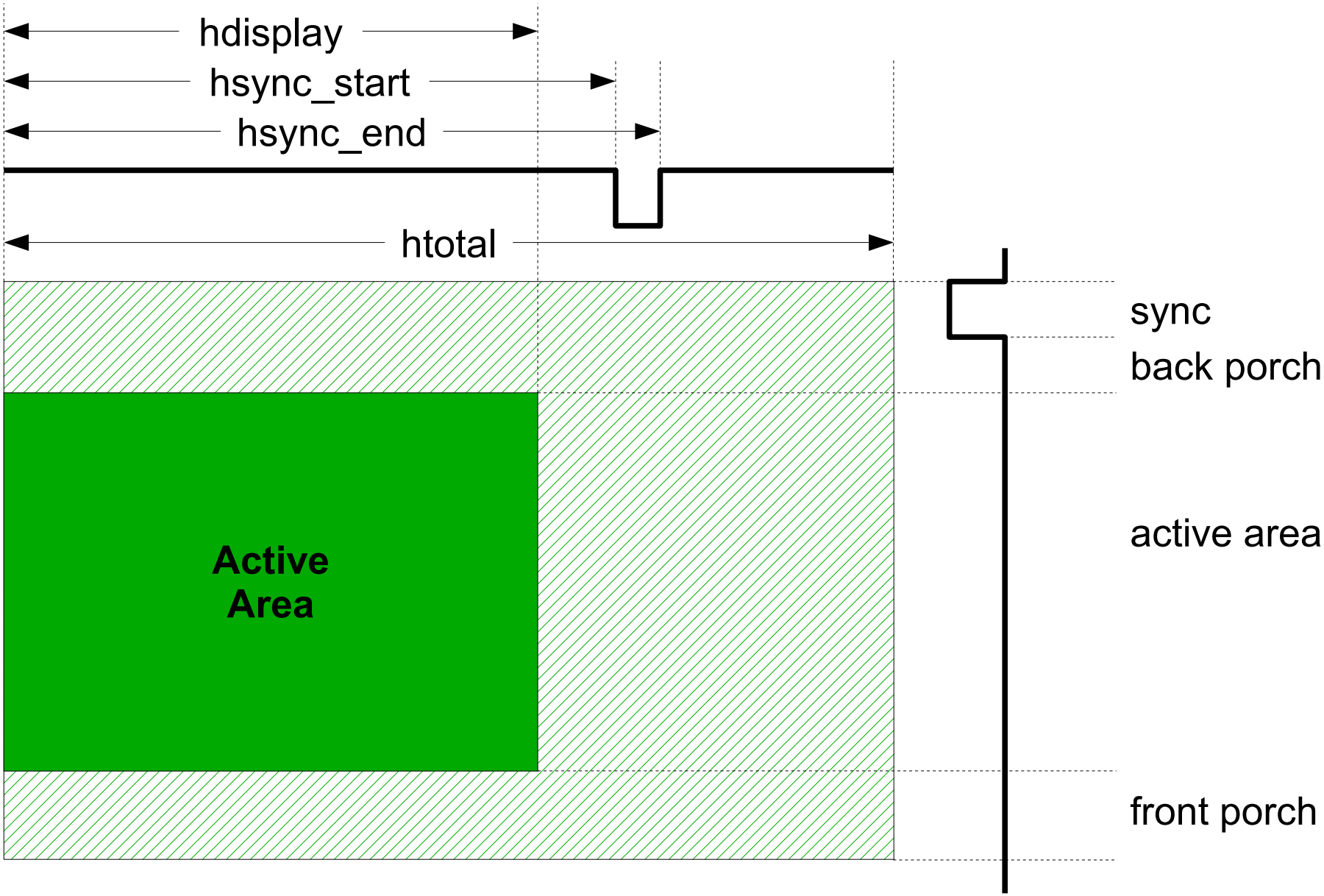
DRM/KMS – GEM Object


Process A
Process B
3
Send FD
SCM_RIGHTS
1
Local
Handle
Global
FD
2
Global
FD
4
Local
Handle
GEM
Object

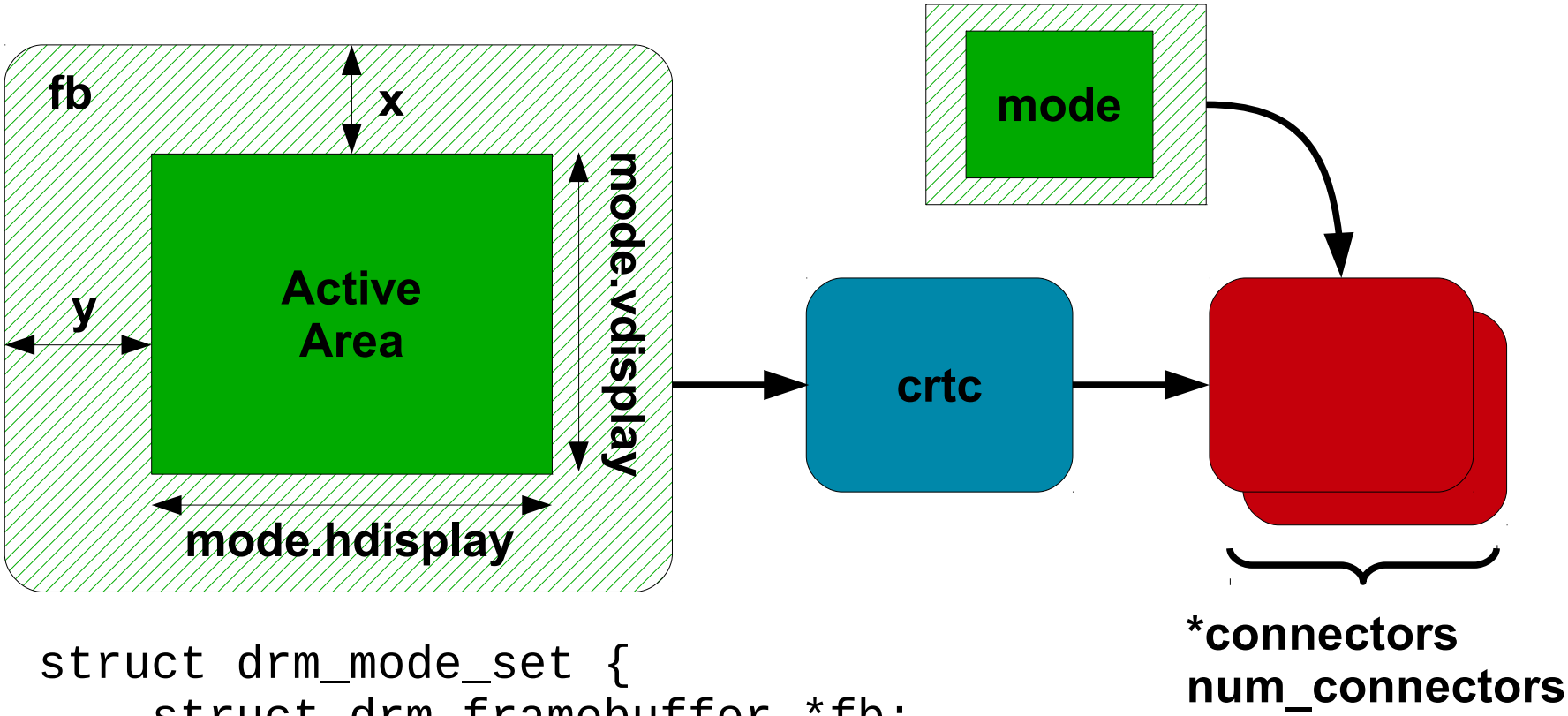
DRM – Handles



KMS – Modes (1/2)

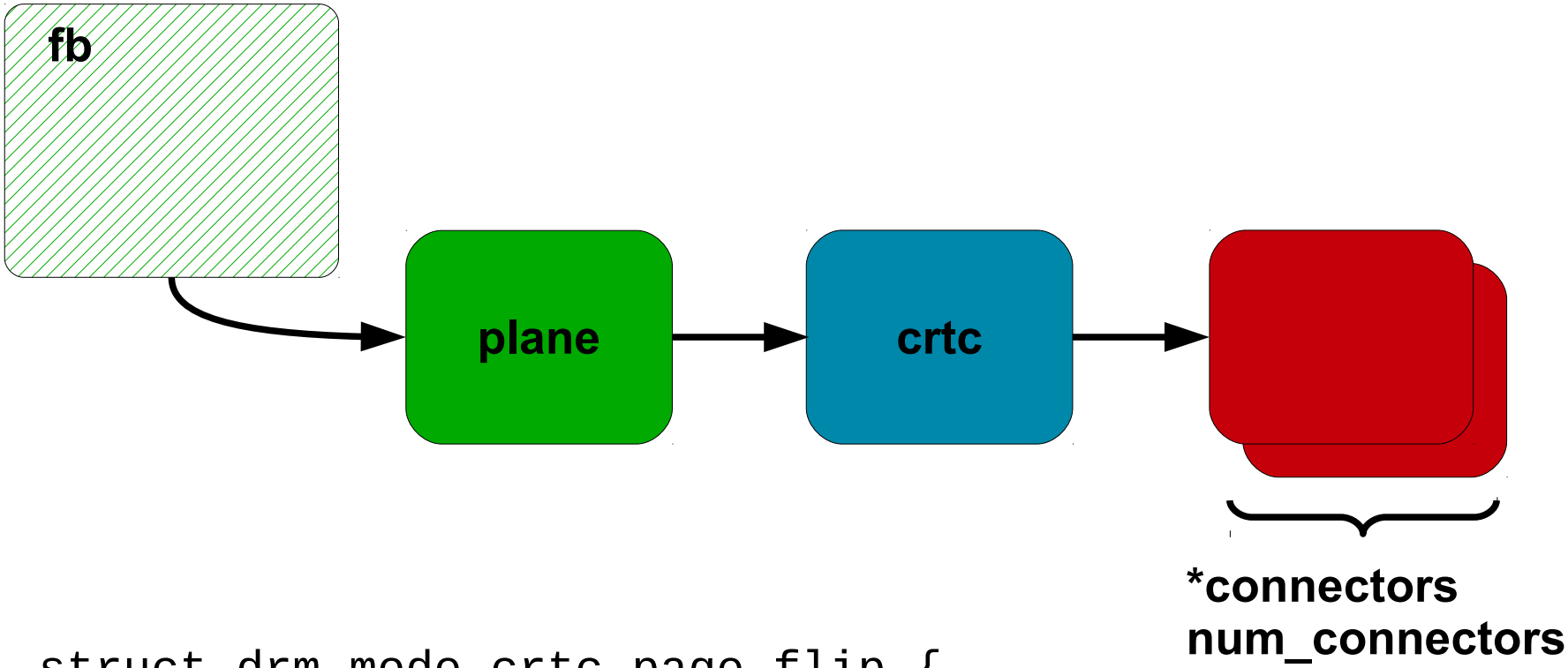

hdisplay
hsync_start
hsync_end
htotal
Active
Area
sync
back porch
active area
front porch

KMS – Modes (2/2)



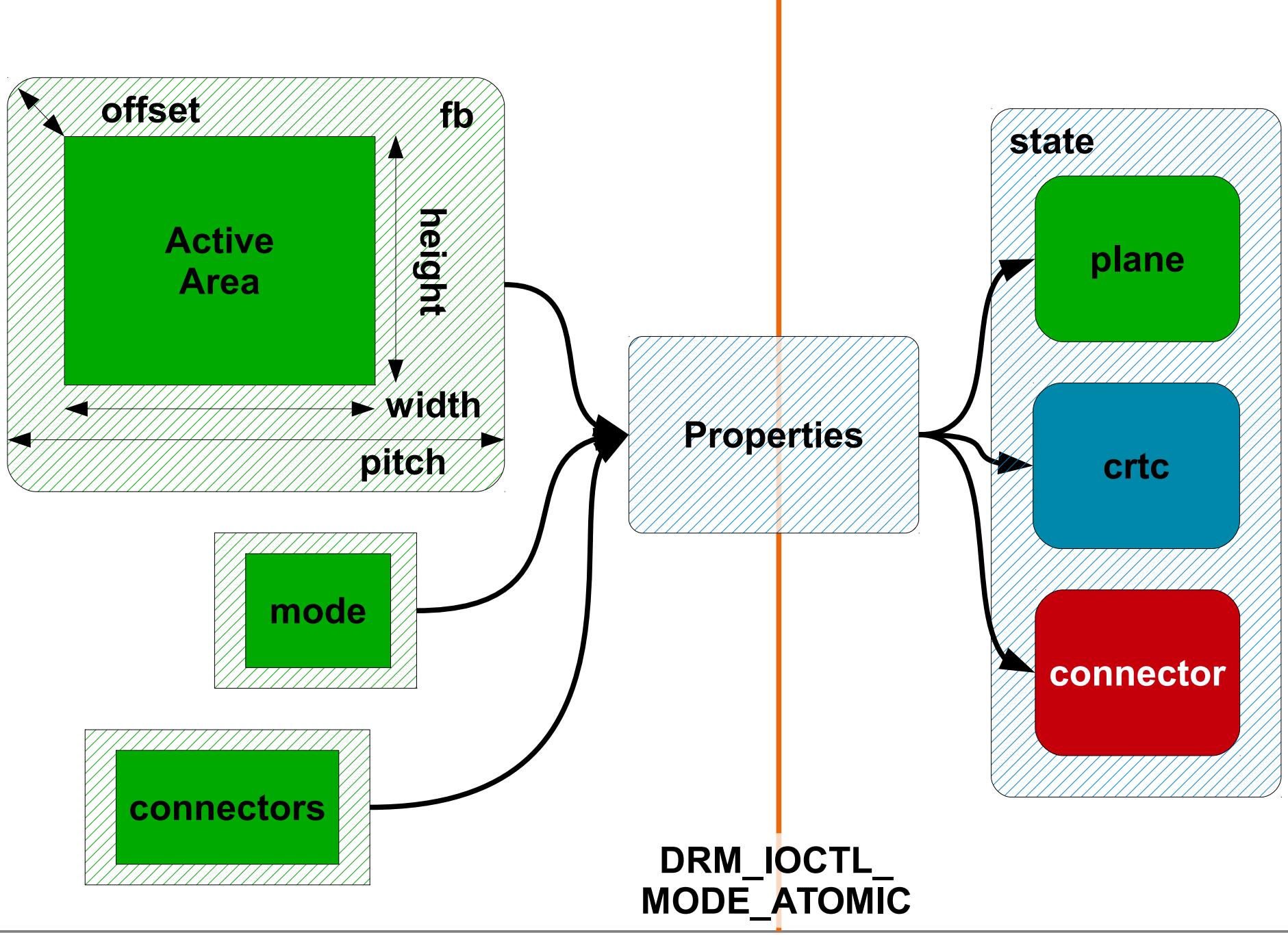

```
struct drm_mode_set {
    struct drm_framebuffer *fb;
    struct drm_crtc *crtc;
    struct drm_display_mode *mode;
    uint32_t x;
    uint32_t y;
    struct drm_connector **connectors;
    size_t num_connectors;
};
```

KMS – Mode Setting

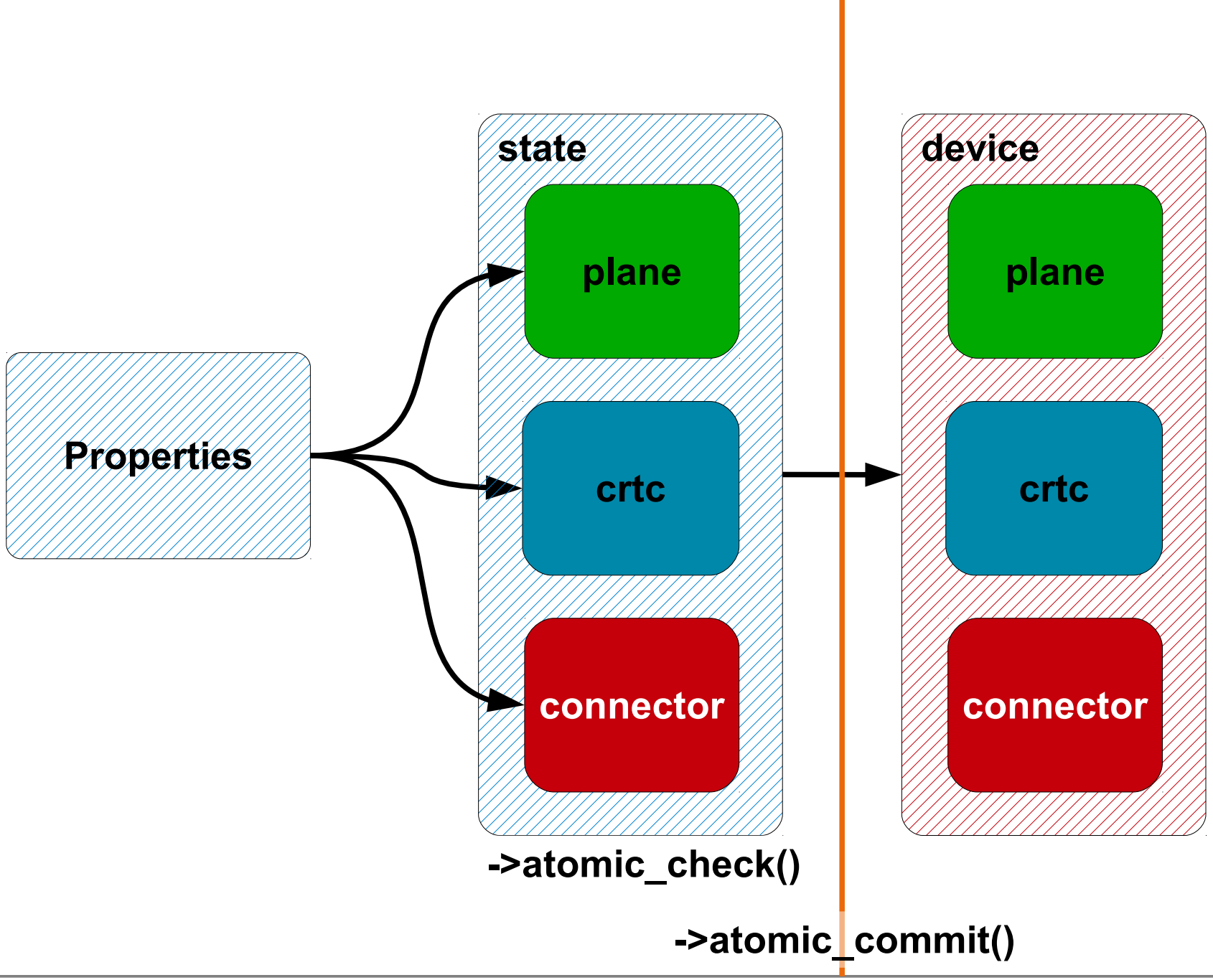



```
struct drm_mode_crtc_page_flip {
        __u32 crtc_id;
        __u32 fb_id;
        __u32 flags;
        __u32 reserved;
        __u64 user_data;
};
```

KMS – Page Flipping



KMS – Atomic Update



KMS – Atomic Update

```
#define DRM_MODE_PAGE_FLIP_EVENT       0x01
#define DRM_MODE_PAGE_FLIP_ASYNC       0x02
#define DRM_MODE_ATOMIC_TEST_ONLY      0x0100
#define DRM_MODE_ATOMIC_NONBLOCK       0x0200
#define DRM_MODE_ATOMIC_ALLOW_MODESET  0x0400

struct drm_mode_atomic {
    __u32 flags;
    __u32 count_objs;
    __u64 objs_ptr;
    __u64 count_props_ptr;
    __u64 props_ptr;
    __u64 prop_values_ptr;
    __u64 reserved;
    __u64 user_data;
};
```

KMS – Atomic Update

kerneldoc drivers/gpu/drm/*

Documentation/DocBook/drm.tmpl

Please contribute

Code Ahead

Locking and error handling omitted for readability

drm_* = core

(rcar_du_)* = driver

```
struct drm_driver rcar_du_driver = {
    ...
};

int rcar_du_probe(struct platform_device *pdev)
{
    struct drm_device *dev;

    ...
    dev = drm_dev_alloc(&rcar_du_driver,
                        &pdev->dev);
    ...
}
```

Device Probe (1/2)

```
int rcar_du_probe(struct platform_device *pdev)
{
    struct rcar_du_device *rcdu;
    struct drm_device *dev;

    ...
    rcdu = kzalloc(sizeof(*rcdu), GFP_KERNEL);
    dev->dev_private = rcdu;

    /* Memory, clocks, regulators, ... */
    ...

    drm_dev_register(dev, 0);
    ...
}
```

Device Probe (2/2)

```
int rcar_du_remove(struct platform_device *pdev)
{
    struct rcar_du_device *rcdu =
                platform_get_drvdata(pdev);
    struct drm_device *dev = rcdu->ddev;

    drm_dev_unregister(dev);

    ...

    drm_dev_unref(dev);

    return 0;
}
```

Device Removal

```
struct drm_driver rcar_du_driver = {
    .driver_features = DRIVER_HAVE_IRQ |
        DRIVER_GEM | DRIVER_MODESET |
        DRIVER_PRIME | DRIVER_ATOMIC,
    .name  = "rcar-du",
    .desc  = "Renesas R-Car Display Unit",
    .date  = "20130110",
    .major = 1,
    .minor = 0,
    .patchlevel = 0,
    ...
};
```

Driver Information

```
struct file_operations rcar_du_fops = {
    .owner          = THIS_MODULE,
    .open           = drm_open,
    .release        = drm_release,
    .unlocked_ioctl = drm_ioctl,
    .compat_ioctl   = drm_compat_ioctl,
    .poll           = drm_poll,
    .read           = drm_read,
    .fasync         = drm_fasync,
    .llseek         = no_llseek,
    .mmap           = ...,
};

struct drm_driver rcar_du_driver = {
    .fops           = &rcar_du_fops,
};
```

File Operations

```
int rcar_du_probe(struct platform_device *pdev)
{
    ...
    drm_irq_install(dev);
    /* Behind the scene:
     * request_irq(platform_get_irq(..., 0))
     */
    ...
}

struct drm_driver rcar_du_driver = {
 /* .irq_preinstall */
    .irq_handler = rcar_du_irq,
 /* .irq_postinstall */
};
```

IRQ Installation

```
struct drm_mode_config_funcs modecfg_funcs = {
    .fb_create = ...,
};

int rcar_du_probe(struct platform_device *pdev)
{
    ...
    drm_mode_config_init(dev);

    dev->mode_config.min_width = 0;
    dev->mode_config.min_height = 0;
    dev->mode_config.max_width = 4095;
    dev->mode_config.max_height = 2047;
    dev->mode_config.funcs =
        &rcar_du_modecfg_funcs;
    ...
}
```

Mode Config

```
struct file_operations rcar_du_fops = {
    .mmap                   = drm_gem_cma_mmap,
};

struct drm_driver rcar_du_driver = {
    .gem_free_object        = drm_gem_cma_free_object,
    .gem_vm_ops             = &drm_gem_cma_vm_ops,
};
```

GEM

```
struct drm_driver rcar_du_driver = {
    .dumb_create        = drm_gem_cma_dumb_create,
    .dumb_map_offset    = drm_gem_cma_dumb_map_offset,
    .dumb_destroy       = drm_gem_cma_dumb_destroy,
};
```

GEM – Dumb Objects

```
struct drm_driver rcar_du_driver = {
    .prime_handle_to_fd = drm_gem_prime_handle_to_fd,
    .prime_fd_to_handle = drm_gem_prime_fd_to_handle,
    .gem_prime_import   = drm_gem_cma_dmabuf_import,
    .gem_prime_export   = drm_gem_cma_dmabuf_export,
};
```

GEM – PRIME

```
drm_framebuffer *
rcar_du_fb_create(struct drm_device *dev,
                  struct drm_file *file_priv,
                  struct drm_mode_fb_cmd2 *mode_cmd)
{
    /* Validate the pixel format, size and pitches */
    ...
    return drm_fb_cma_create(dev, file_priv,
                             mode_cmd);
}

struct drm_mode_config_funcs rcar_du_modecfg_funcs =
{
    .fb_create = rcar_du_fb_create,
};
```

Frame Buffer

```
struct drm_crtc_funcs crtc_funcs = {
    .destroy = drm_crtc_cleanup,
    ...
};

int rcar_du_probe(struct platform_device *pdev)
{
    struct drm_device *dev;
    struct drm_crtc *crtc;
    ...
    drm_crtc_init(dev, crtc, &crtc_funcs);
    ...
}
```

Initialization – CRTC

```
struct drm_encoder_funcs encoder_funcs = {
    .destroy = drm_encoder_cleanup,
};

int rcar_du_probe(struct platform_device *pdev)
{
    struct drm_device *dev;
    struct drm_encoder *encoder;
    ...
    encoder->possible_crtcs = 1 << crtc;
    drm_encoder_init(dev, encoder, &encoder_funcs,
                     DRM_MODE_ENCODER_DAC);
    ...
}
```

Initialization – Encoder

```
struct drm_connector_funcs connector_funcs = {
    .destroy = drm_connector_cleanup,
    ...
};

int rcar_du_probe(struct platform_device *pdev)
{
    struct drm_device *dev;
    struct drm_connector *connector;
    ...
    connector->display_info.width_mm = ...;
    connector->display_info.height_mm = ...;

    drm_connector_init(dev, connector,
        &connector_funcs, DRM_MODE_CONNECTOR_VGA);
    drm_connector_register(connector);
    drm_mode_connector_attach_encoder(connector,
        encoder);
}
```

Initialization – Connector

```
struct drm_plane_funcs plane_funcs = {
    .destroy = drm_plane_cleanup,
    ...
};

uint32_t formats[] = {
    DRM_FORMAT_RGB565, ...
};

int rcar_du_probe(struct platform_device *pdev)
{
    struct drm_plane *plane = ...;
    ...
    drm_universal_plane_init(dev, plane, crtcs,
                &plane_funcs, formats,
                ARRAY_SIZE(formats),
                DRM_PLANE_TYPE_PRIMARY);
            /* DRM_PLANE_TYPE_OVERLAY */
}
```

Initialization – Plane

```
struct drm_connector_funcs connector_funcs = {
    .fill_modes = ...,
};

int (*fill_modes)(struct drm_connector *connector,
          uint32_t max_width, uint32_t max_height);
```

Modes Discovery (1/2)

Core

```
struct drm_connector_funcs connector_funcs = {
    .fill_modes = drm_helper_probe_single_connector_modes,
};
```

Helpers

```
struct drm_connector_helper_funcs connector_helper_funcs =
{
    .get_modes = rcar_du_vga_connector_get_modes,
    .mode_valid = rcar_du_vga_connector_mode_valid,
};
```

Modes Discovery (2/2)

```
struct drm_crtc_funcs crtc_funcs = {
    .reset = drm_atomic_helper_crtc_reset,
    .atomic_duplicate_state =
            drm_atomic_helper_crtc_duplicate_state,
    .atomic_destroy_state =
            drm_atomic_helper_crtc_destroy_state,
};

struct drm_crtc_state {
    struct drm_crtc *crtc;

    bool enable;
    bool active;

    bool planes_changed : 1;
    bool mode_changed : 1;
    bool active_changed : 1;
    ...
};
```

State – CRTC

```
struct drm_connector_funcs connector_funcs = {
    .reset = drm_atomic_helper_connector_reset,
    .atomic_duplicate_state =
        drm_atomic_helper_connector_duplicate_state,
    .atomic_destroy_state =
        drm_atomic_helper_connector_destroy_state,
};

struct drm_connector_state {
    struct drm_connector *connector;

    struct drm_crtc *crtc;

    struct drm_encoder *best_encoder;

    struct drm_atomic_state *state;
};
```

State – Connector

```
struct drm_plane_funcs plane_funcs = {
    .reset = rcar_du_plane_reset,
    .atomic_duplicate_state =
        rcar_du_plane_atomic_duplicate_state,
    .atomic_destroy_state =
        rcar_du_plane_atomic_destroy_state,
};

struct drm_plane_state {
    struct drm_plane *plane;

    struct drm_crtc *crtc;
    struct drm_framebuffer *fb;
    struct fence *fence;

    int32_t crtc_x, crtc_y;
    uint32_t crtc_w, crtc_h;
    ...
};
```

State – Plane (1/5)

```
struct rcar_du_plane_state {
    struct drm_plane_state state;

    const struct rcar_du_format_info *format;
    int hwindex;

    unsigned int alpha;
    unsigned int colorkey;
    unsigned int zpos;
};
```

State – Plane (2/5)

```
void rcar_du_plane_reset(struct drm_plane *plane)
{
    struct rcar_du_plane_state *state;

    if (plane->state) {
        rcar_du_plane_atomic_destroy_state(
                plane, plane->state);
        plane->state = NULL;
    }

    state = kzalloc(sizeof(*state), GFP_KERNEL);
    state->hwindex = -1;
    state->alpha = 255;
    state->colorkey = RCAR_DU_COLORKEY_NONE;
    ...

    plane->state = &state->state;
    plane->state->plane = plane;
}
```

State – Plane (3/5)

```
struct drm_plane_state *
rcar_du_plane_atomic_duplicate_state(
        struct drm_plane *plane)
{
    struct rcar_du_plane_state *state;
    struct rcar_du_plane_state *copy;

    state = to_rcar_plane_state(plane->state);
    copy = kmemdup(state, sizeof(*state),
                   GFP_KERNEL);

    __drm_atomic_helper_plane_duplicate_state(
            plane, &copy->state);

    return &copy->state;
}
```

State – Plane (4/5)

```
void rcar_du_plane_atomic_destroy_state(
        struct drm_plane *plane,
        struct drm_plane_state *state)
{
    __drm_atomic_helper_plane_destroy_state(
            plane, state);
    kfree(to_rcar_plane_state(state));
}
```

State – Plane (5/5)

```
int drm_atomic_set_crtc_for_plane(
        struct drm_plane_state *plane_state,
        struct drm_crtc *crtc);

void drm_atomic_set_fb_for_plane(
        struct drm_plane_state *plane_state,
        struct drm_framebuffer *fb);

int drm_atomic_set_crtc_for_connector(
        struct drm_connector_state *conn_state,
        struct drm_crtc *crtc);

...
```

State Manipulation

```
struct drm_mode_config_funcs mode_config_funcs = {
    .atomic_check = ...,
    .atomic_commit = ...,
};

int (*atomic_check)(struct drm_device *dev,
                    struct drm_atomic_state *a);
int (*atomic_commit)(struct drm_device *dev,
                     struct drm_atomic_state *a,
                     bool async);
```

Atomic Update – KMS (1/2)

Core

```
struct drm_mode_config_funcs mode_config_funcs = {
    .atomic_check = drm_atomic_helper_check,
    .atomic_commit = drm_atomic_helper_commit,
};
```

Helpers

```
int rcar_du_load(struct drm_device *dev,
                 unsigned long flags)
{
    ...
    drm_crtc_helper_add(crtc, &crtc_helper_funcs);
    drm_connector_helper_add(connector,
        &connector_helper_funcs);
    drm_encoder_helper_add(encoder,
        &encoder_helper_funcs);
    ...
}
```

Atomic Update – KMS (2/2)

```
struct drm_crtc_helper_funcs crtc_helper_funcs = {
    .mode_fixup = rcar_du_crtc_mode_fixup,
 /* .mode_set_nofb = ..., */
    .disable = rcar_du_crtc_disable,
    .enable = rcar_du_crtc_enable,
};

bool rcar_du_crtc_mode_fixup(struct drm_crtc *crtc,
            const struct drm_display_mode *mode,
            struct drm_display_mode *adjusted_mode)
{    ...    }

void rcar_du_crtc_disable(struct drm_crtc *crtc)
{    ...    }

void rcar_du_crtc_enable(struct drm_crtc *crtc)
{    ...    }
```

Atomic Update – CRTC

```
struct drm_encoder_helper_funcs encoder_helper_funcs = {
    .mode_set = rcar_du_encoder_mode_set,
    .disable = rcar_du_encoder_disable,
    .enable = rcar_du_encoder_enable,
    .atomic_check = rcar_du_encoder_atomic_check,
};

struct drm_connector_helper_funcs connector_helper_funcs =
{
    .atomic_best_encoder = rcar_du_connector_best_encoder,
 /* .best_encoder still used for FBDEV emulation */
};
```

Atomic Update – Encoder

```
struct drm_crtc_helper_funcs crtc_helper_funcs = {
    .atomic_begin = rcar_du_crtc_atomic_begin,
    .atomic_flush = rcar_du_crtc_atomic_flush,
};

void rcar_du_crtc_atomic_begin(struct drm_crtc *crtc,
            struct drm_crtc_state *old_crtc_state)
{
    /* Enable vblank processing */
    drm_crtc_vblank_get(crtc);
    ...
}

void rcar_du_crtc_atomic_flush(struct drm_crtc *crtc,
            struct drm_crtc_state *old_crtc_state)
{
    /* Set the GO bit */
    ...
}
```

Atomic Update – CRTC + Plane

```
struct drm_plane_helper_funcs plane_helper_funcs = {
    .prepare_fb = ...,
    .cleanup_fb = ...,
};

int (*prepare_fb)(struct drm_plane *plane,
            struct drm_framebuffer *fb,
            const struct drm_plane_state *new_state);
void (*cleanup_fb)(struct drm_plane *plane,
            struct drm_framebuffer *fb,
            const struct drm_plane_state *old_state);
```

Atomic Update – Plane (1/2)

```
struct drm_plane_helper_funcs plane_helper_funcs = {
    .atomic_check = rcar_du_plane_atomic_check,
    .atomic_update = rcar_du_plane_atomic_update,
 /* .atomic_disable = ..., */
};

int rcar_du_plane_atomic_check(
        struct drm_plane *plane,
        struct drm_plane_state *state)
{
    ...
}

void rcar_du_plane_atomic_update(
        struct drm_plane *plane,
        struct drm_plane_state *old_state)
{
    ...
}
```

Atomic Update – Plane (2/2)

```
int rcar_du_probe(struct platform_device *pdev)
{
    ...
    drm_vblank_init(dev, 1);
    ...
}

irqreturn_t rcar_du_irq(int irq, void *arg)
{
    struct drm_device *dev = arg;

    drm_handle_vblank(dev, 0);
}
```

Vertical Blanking (1/4)

```
int rcar_du_enable_vblank(struct drm_device *dev,
                          int crtc)
{
    /* Enable the vblank interrupt for the CRTC */
    return 0;
}

void rcar_du_disable_vblank(struct drm_device *dev,
                            int crtc)
{
    /* Disable the vblank interrupt for the CRTC */
}

struct drm_driver rcar_du_driver = {
    .get_vblank_counter = drm_vblank_count,
    .enable_vblank      = rcar_du_enable_vblank,
    .disable_vblank     = rcar_du_disable_vblank,
};
```

Vertical Blanking (2/4)

```
irqreturn_t rcar_du_crtc_irq(int irq, void *arg)
{
    struct rcar_du_crtc *rcrtc = arg;
    ...

    drm_handle_vblank(rcrtc->crtc.dev, rcrtc->index);
    rcar_du_crtc_finish_page_flip(rcrtc);

    return IRQ_HANDLED;
}
```

Vertical Blanking (3/4)

```
void rcar_du_crtc_finish_page_flip(struct rcar_du_crtc *rcrtc)
{
    struct drm_pending_vblank_event *event;
    struct drm_device *dev = rcrtc->crtc.dev;
    unsigned long flags;

    spin_lock_irqsave(&dev->event_lock, flags);
    event = rcrtc->event;
    rcrtc->event = NULL;
    spin_unlock_irqrestore(&dev->event_lock, flags);

    if (event == NULL)
        return;

    spin_lock_irqsave(&dev->event_lock, flags);
    drm_send_vblank_event(dev, rcrtc->index, event);
    wake_up(&rcrtc->flip_wait);
    spin_unlock_irqrestore(&dev->event_lock, flags);

    drm_crtc_vblank_put(&rcrtc->crtc);
}
```

Vertical Blanking (4/4)

```
struct drm_crtc_funcs crtc_funcs = {
    .set_config = drm_atomic_helper_set_config,
    .page_flip = drm_atomic_helper_page_flip,
};

struct drm_connector_funcs connector_funcs = {
    .dpms = drm_atomic_helper_connector_dpms,
};
```

Legacy

- properties
- connector status poll
- FBDEV emulation
- ...

And Much More...

Source: http://www.flickr.com/photos/buckaroobay/3721809183/

- suspend/resume helpers
- atomic fbdev
- active only plane update
- better runtime PM support

Ongoing Work

- vblank rework
- bridge vs. slave encoder
- write back
- live sources
- fences
- validation
- fastboot
- generic async commit

Future Work

- Conversion HOWTO for legacy drivers
http://blog.ffwll.ch/2014/11/atomic-modeset-support-for-kms-drivers.html

- Atomic mode setting design overview (LWN)
https://lwn.net/Articles/653071/
https://lwn.net/Articles/653466/

- DRM DocBook
https://01.org/linuxgraphics/gfx-docs/drm/

Resources

- dri-devel@lists.freedesktop.org
- laurent.pinchart@ideasonboard.com

Contact

?!

thx.