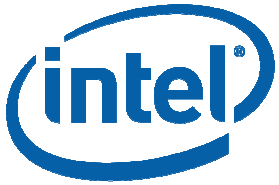


Enhancing Linux Graphics

Jesse Barnes

Intel Open Source Technology Center

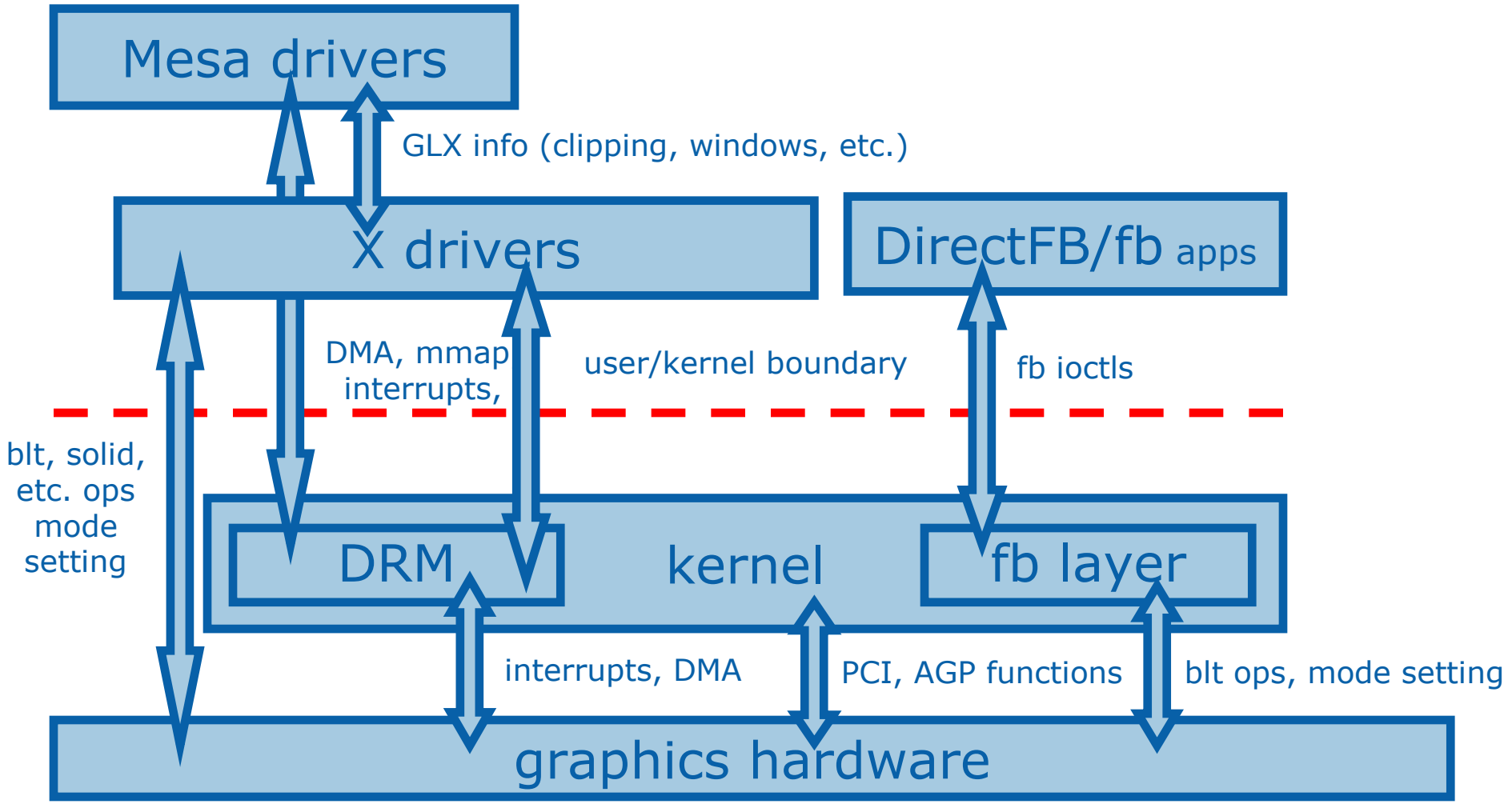
Overview

- Overview of current system
- Shortcomings
- Kernel changes
  - Design review of DRM changes
  - Other kernel changes
- External interfaces
  - Memory management
  - Output control & mode setting
- Status
- Future work

Current Players

- vgacon
  - relies on standard VGA registers
  - typically used on PCs from boot time until X starts
- fb layer
  - has specific device drivers for many devices
  - needed for platforms that don't boot in VGA mode
  - provides fbcon console driver
- userspace drivers
  - e.g. X, DirectFB
  - provide full access to hardware features
    - DRM+X+Mesa combo provides full 3D acceleration, video playback, etc.

Current Architecture



Shortcomings

- layers missing functionality
  - fbcon accelerated, but fb doesn't export accelerated API to applications
  - DirectFB available, but not as featureful as X+Mesa combo
- some stacks heavyweight
  - DRM+X+Mesa is a fairly large chunk of code
- duplication of functionality
  - fb and X provide mode setting
  - DirectFB and X provide acceleration
  - memory management fragmented and ad hoc
- missing features
  - suspend/resume only available in some fb configurations
  - often conflicts with DRM if used

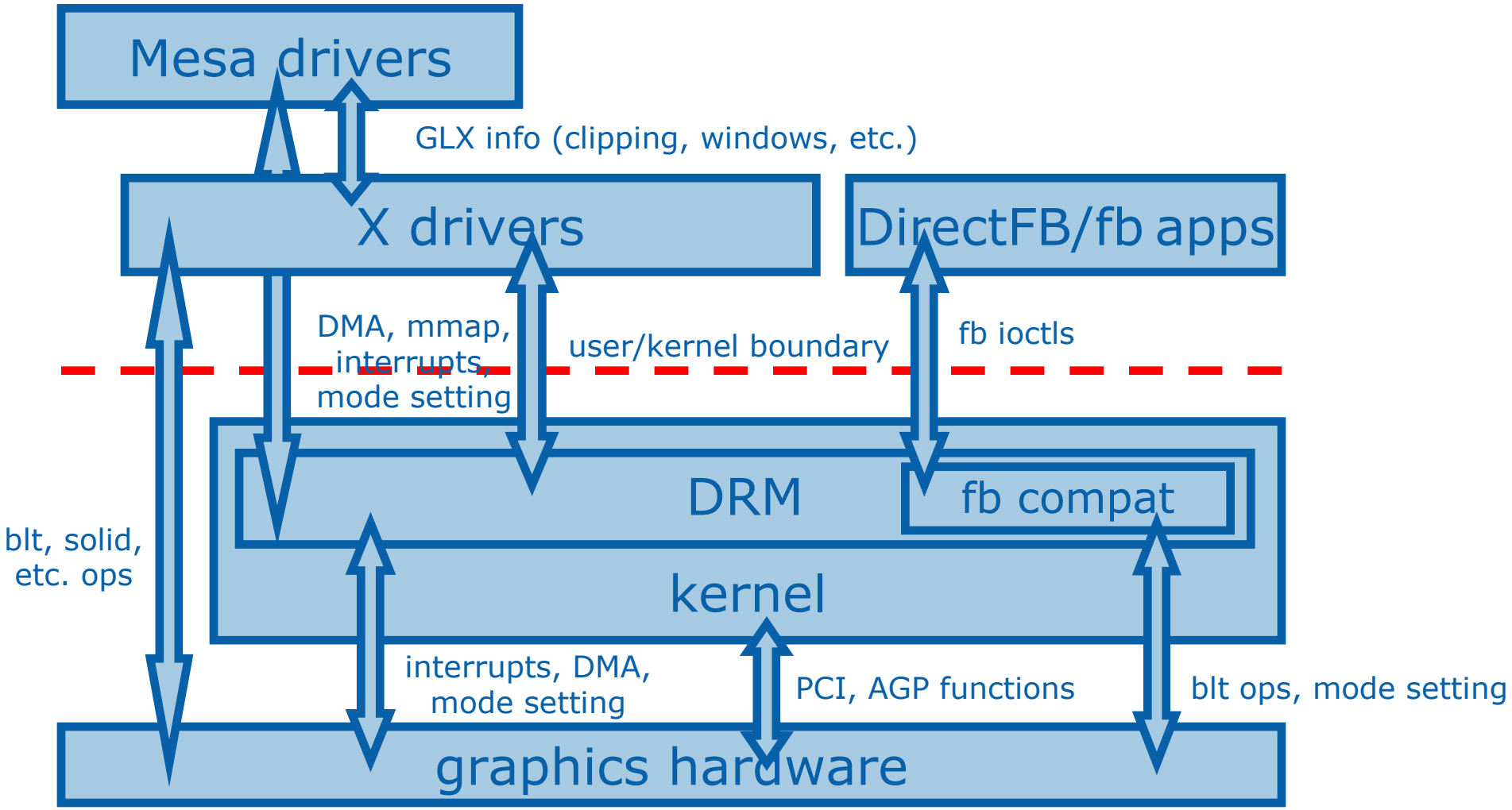
Requirements

- Desired features
  - full device functionality (3D, video playback, output control, etc.)
  - fast, reliable suspend/resume
  - debug support (i.e. panic/oops on screen)
  - harmonization of various kernel and user level drivers (i.e. better sharing)
  - X independence
- DRM+X+Mesa provide most complete set of functionality
  - good place to centralize other code like mode setting, memory management
- Want to retain compatibility with other systems (e.g. fb applications using existing fb ioctls)

DRM Additions

- Mode setting
  - Stolen from X.Org's RandR 1.2 implementation
    - Keith and Eric made this really easy
    - DRM core inherits core structures, management
    - Intel DRM driver gets output and CRTC control code
- Suspend/resume
- Panic/oops support
- External APIs
  - mode setting
  - output control
  - memory management

Enhanced Architecture



Basic Structures for Output Control

- Top level GPU configuration object
  - list of available CRTCs, outputs, framebuffers, and user added modes
- CRTC (CRT controller, historical anachronism) object
  - current mode
  - associated framebuffer
  - x, y offsets into framebuffer
- Output object for each available output
  - probed modes (if any)
  - information about attached display (if any)
  - user added modes
- Framebuffer(s)
  - size, offset into graphics memory, etc.
  - associated buffer object from memory manager

New Initialization Requirements

- Past initialization was driven by X drivers, but to setup necessary structures, driver has to initialize itself
  - discover graphics devices
  - map registers
  - set up initial device communication (e.g. command ring buffer)
  - set up memory manager
  - discover and enumerate available CRTCs
  - discover and enumerate available outputs
  - probe outputs for attached displays, gather display and mode availability information
  - set up initial GPU configuration (e.g. initial mode) if needed
- Low level driver updates required to accommodate new initialization requirements

Legacy Concerns

- New initialization, memory management, etc. raises compatibility concerns
  - old X running on new system may clobber memory layout
  - mode setting done by both layers may conflict
  - trying to perform initialization from both X and DRM driver may cause problems
- So, give distributions choice
  - enable new style driver setup when their userspace is ready
  - compile time flag to control whether driver will be fully backwards compatible or only available to updated applications
  - default is fully backward compatible to avoid problems
  - per-DRM driver flag controls new behavior

Other Kernel Internals

- New initialization means low level DRM driver can bind to DRM device
  - can provide suspend/resume methods
  - should make suspend/resume fast and reliable
- Panic/oops support
  - new KD_KERNEL_GRAPHICS mode required
  - DRM sets KD_KERNEL_GRAPHICS when mode set call occurs
  - kernel can output panic/oops text over currently running graphical application
  - alternately, kernel could set new mode (more risky) and display output

External Interfaces

- Memory management
  - need memory object allocation/map/free API
  - also need ref/unref and pinning for scanout buffers
- Output control
  - get GPU configuration
  - set CRTC<->output mappings
  - adjust properties like backlight brightness
- Mode setting
  - once configuration is gathered, modes can be set
  - ability to modify existing mode list
    - needed to work around bad EDID data
    - desirable for configurations where mode data may not be available (e.g. embedded systems)

Status

- Core routines in place
  - DRM code based on X RandR 1.2 has been added
  - low level drivers can call new functions as CRTCs, outputs, etc. are added
  - DDC probing and EDID parsing code available to build initial mode list, gather display info
- External APIs prototyped
  - memory management nearing completion
  - output control, mode setting ioctls available
    - split into control/user nodes for security reasons
- Some applications already developed
  - mobile devices using new code
  - embedded kiosk type applications also in use

Future Work

- Much yet to be done
  - DRM core needs refactoring
    - both old and new style drivers must be supported
  - Panic/oops support yet to be added
    - should be straightforward to code new KD_KERNEL_GRAPHICS code, add to compatible fbcon code
  - External interfaces need work
    - memory management should be ready
    - output control and mode setting in good shape, but DDX drivers still need porting as final sanity check
  - Drivers need porting
    - i915 and radeon drivers ported at this point
    - interest shown in nouveau and other drivers
  - Applications
    - X drivers need to be aware of new architecture
    - DirectFB can use new system
    - standalone Mesa could be developed further

Open Research Questions

- Further driver consolidation?
  - Call 2D operations (blt, solid fill, etc.) be consolidated in Mesa?
  - Would centralize most graphics functionality in kernel (interrupts, DMA, mode setting) + Mesa (complex operations like 3D, 2D) combination
  - Mesa overhead may be too high, architecture may need changes
  - See Glucose project
- Better kernel integration?
  - Can we schedule DMA better in the kernel DRM driver?
  - Is making the CPU scheduler aware of GPU activity a good idea?
  - Move mouse control (pointer movement, cusor update) into kernel for better user experience?


intel
Leap ahead™