DRI Memory Management

- Full strength manager wasn't required for traditional usage: Quake3 and glxgears.
- Perceived to be difficult.
- Fundamental for modern desktops, offscreen rendering.
- Talked about for years, can be put off no longer.

Current behaviour

- Clients cooperate to avoid treading on each others textures.
- If one client needs more memory, it can eject other textures **without saving the contents**.
- Some global information to help with decisions
- Easy implementation, better than nothing.

But...

- Cannot trust data to remain in texture memory.
- Two copies of textures – one in main memory, one on card.
- Slow texture uploads – update both copies.
- Can't use the blitter for CopyTexSubImage
- No EXT_fbo, pbuffers, private backbuffers...
- No fast VBOs, PBOs.
- Nasty hacks - GLX_MESA_allocate_memory

What is needed to move on?

- Not just textures: Generalize textures to buffers.
- Guarantee that buffer contents be preserved.
- Still need to evict other clients buffers.
- Mechanism to force buffers to AGP/VRAM.
- Mechanism to find out buffer offsets (plug into DMA cmds).
- Mechanism to map/unmap buffers into client memory.

Semantics: Stolen from ARB_vbo

- Always sensible to look at the closest ARB extension for inspiration.
- ARB_vbo is a generalized interface to multiple vendor's memory manager, provides 80% of the semantics.
- We just implement the ARB semantics and add on the missing driver-facing interfaces.

Buffers

- Buffers are identified by opaque integer handles.
- Sharing buffers should be straightforward.
- No apparent limitation (eg GL's limited sharing semantics)
- But – no mechanism for notifying other contexts of size changes, etc.   This may have to be handled at a higher level.
- Key new call: ValidateBufferList() - specifies acceptable memory pools for each buffer, triggers the upload.

Fences

- Fence encapsulates flushing and IRQ lowlevel mechanisms.
- Need some smartness about when to emit flushes, which sorts of flushes and when to emit IRQs.
- Buffer manager code mainly does this behind the scenes.
- Fallbacks, image uploads and map/unmap just work, fences emitted and retired automatically.

Current Status

- Userspace prototype in i915 driver. Memcpy based.
- Fast TexSubImage, CopyTexSubImage, CopyPixels.
- Fast TexImage – further optimized by not waiting for idle before replacing old image contents.
- EXT_fbo – soon.
- Other paths – as time allows.
- Thomas' TTM code for dynamic AGP manipulation.
- A clear path for a VRAM implementation.

Current Issues

- ValidateBuffer offsets only reliable while lock held.
- Problems stuffing DMA buffers – need to fire DMA before releasing DRI lock. Can only really emit DMA with lock held.
- Solution: Fixup/relocation lists for DMA buffers. Emit DMA without lock, grab lock, fixup, fire, unlock.
- Will prototype soon.
- Integration of Thomas' and my code remains to be done.

Next steps

- Implement the DMA fixup lists.
- What about cliprects?
- Treat Command Buffers (DMA buf + fixups) as another first-class object in memory manager, solidify their semantics.
- Hand multiple DMA+patch buffers to the memory manager, let it decide when best to fire them.
- The memory manager becomes a scheduler.

What about the DDX?

- Phase 1: Nothing happens.
- VideoRAM specifies a fixed-size AGP pool, we manage the rest of the aperture.
- Ongoing, this pool is an excellent place for pinned buffers – cannot fragment the remaining address space.

DDX - Moving forward...

- Incrementally: Move stuff into the managed pool.
- Only keep pinned buffers in the fixed pool:
  - Scanout buffers (the frontbuffer)
  - Hardware cursor
  - Also: video memory provided to client applications by ARB_vbo map/unmap semantics.
- Everything else can be pushed into the memory manager.
- Ultimately, teach the memory manager about pinned buffers as well.

What about VRAM?

- AGP dynamic mapping is a cool trick. What about VRAM?
- Need two things
  - Data transfer path to/from VRAM
  - Mechanism for allocating memory for evicted buffers in the correct address space.
- Allocation may be challenging, but solvable.
- Also: new semantics for deciding when to copy between AGP<->VRAM.

Optimizations, More future stuff

- Trial various replacement algorithms.
- Speculative upload/download.
- Speculative duplication – copy in both local and video memory – on evict or update, just abandon the invalidated copy.
- DMA prioritization.
- How to deal with multiple HW command queues?
- Efficient multiple client sync-to-vblank.
- Allocate backbuffer on first render command, deallocate on swapbuffers – big savings for doublebuffered UI's, GL-based UI's. Also: triple buffering for free.

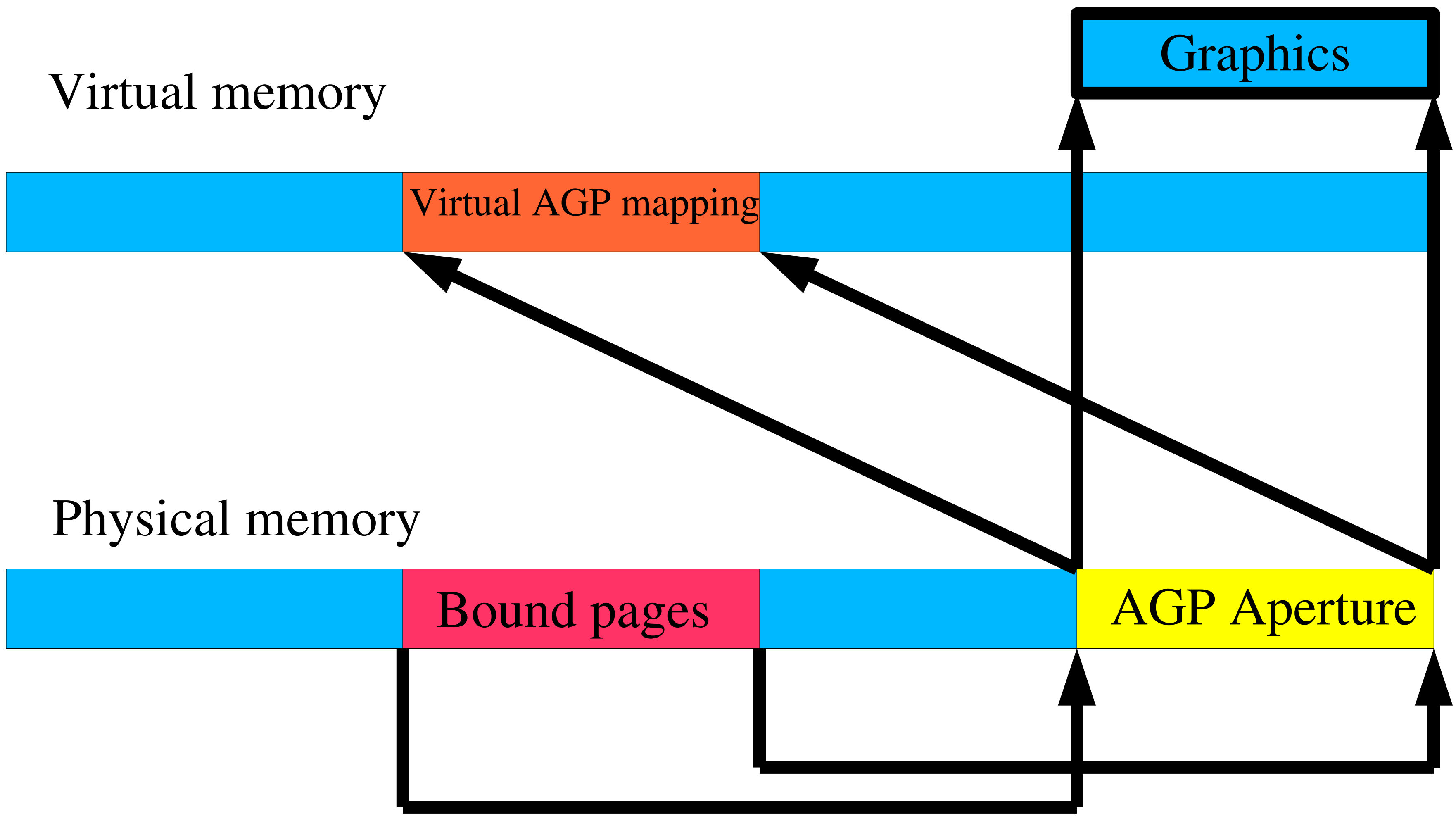
Outline

- Background
- AGP mapping: Current situation
- What we want to do.
- What we can do.
- Translation table maps

Background

- AGP space is currently limited
  a) by the size of AGP aperture.
  b) by the amount of AGP memory allocated at DRI initialisation.
- When space runs out, we need to evict “old” data. Might be expensive to read back.
- Reading from AGP space is slow.

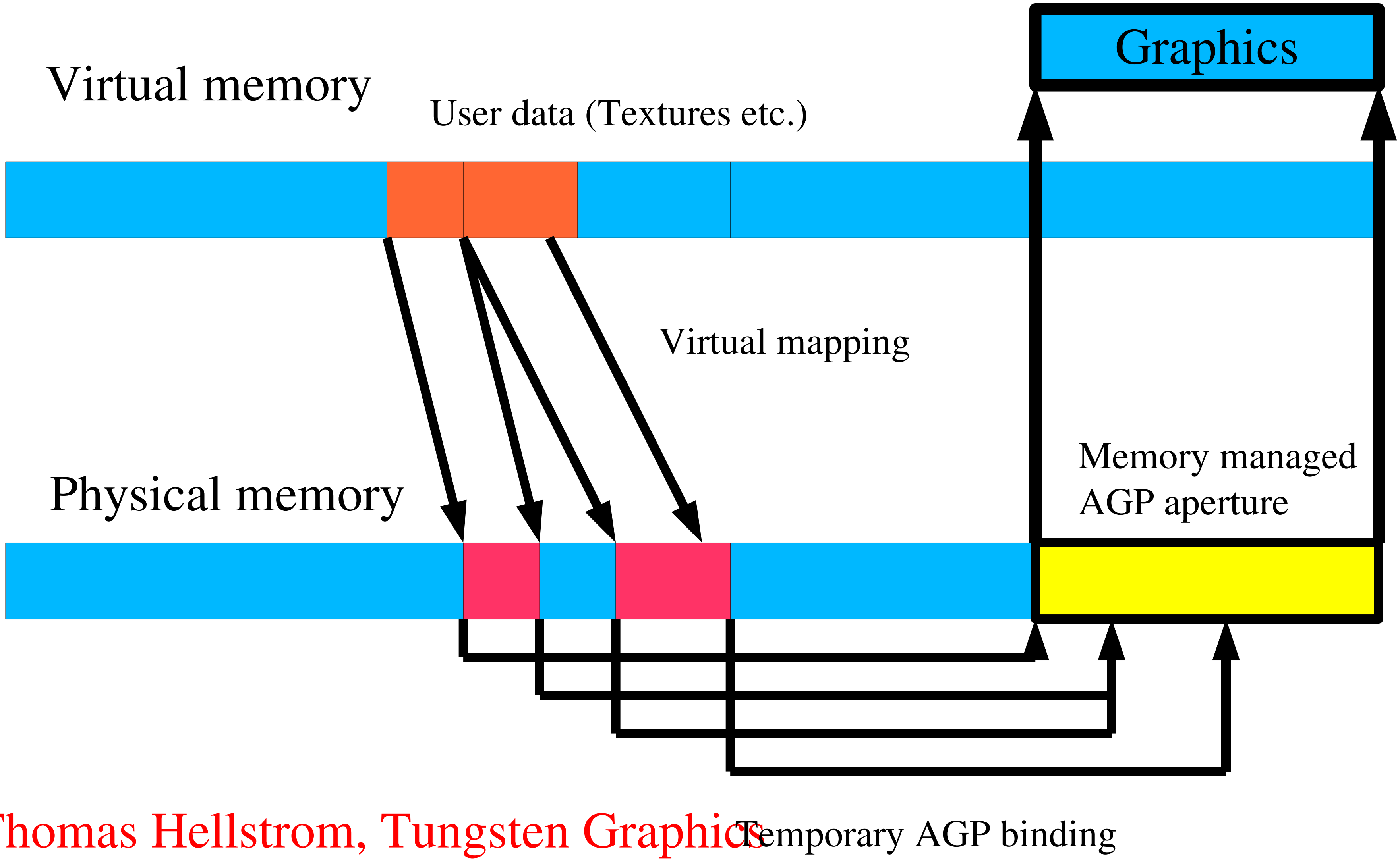
Current situation



AGP space management

- No unused allocated AGP memory just sitting around.
- Manage Aperture space.
- Fast binding / unbinding / eviction.
- Fast reading from AGP space.

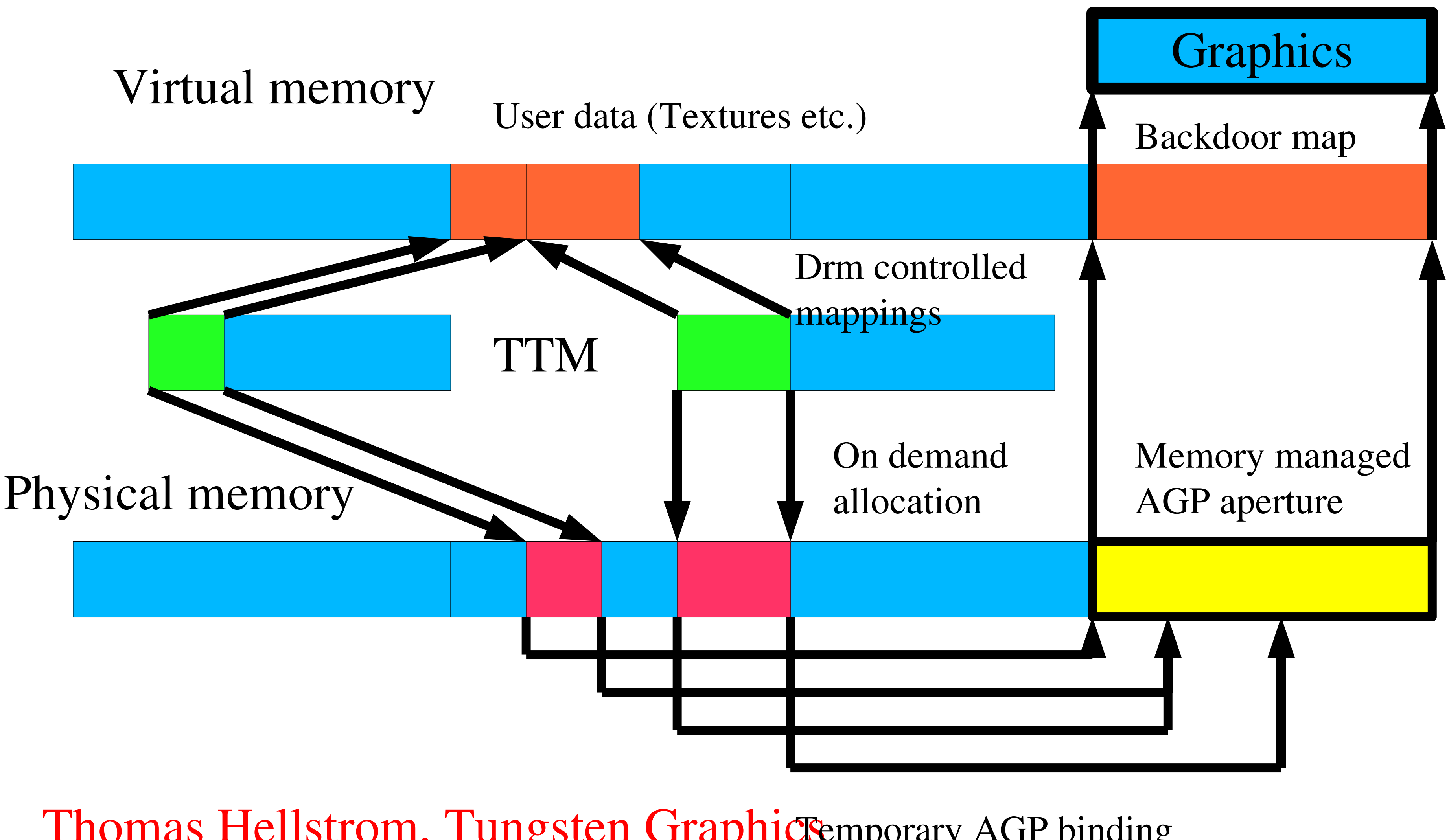
What we want to do



Problems and solutions

- All mappings of bound AGP memory needs to be uncached.
- Memory really needs to be cacheable when reading from it.
- Solution 1: Some TT implementations allow cacheable pages. (PCIE, Intel GTT).
- Solution 2: Let DRM manage all mappings (User virtual, kernel and AGP) of the pages. Change caching policy when binding / unbinding -> TTM

What we can do



Translation Table Maps

- The user-space part of the memory manager creates TTMs.
- Memory pages are automatically allocated when used or bound.
- The user can request binding of any range of pages from the ttm to the aperture. An aperture space manager decides where they appear.
- If there is not enough space, the aperture space manager evicts pages that are not currently used.

Implications and limitations

- TTMs are not currently resizeable, which would be nice for efficient reallocation.
- One TTM per buffer?
- If the TT can bind cached pages, TTMs are not really needed. Just bind any user page.
- Anonymous TTM region.
- Shared TTM memory – Access rights?

API

- The API (libdrm) provides provisions for
- Validating buffers – valid as long as DRM lock is held. Unnecessary kernel calls avoided.
- Unbinding / evicting buffers.
- Mapping / unmapping buffers – provide virtual address for processor access.
- Destroying buffers.
- Fences.