the
gamedesigninitiative
at cornell university

Memory Management

Take-Aways for This Lesson

- Why does memory in games matter?
  - Is there a difference between PC and mobile?
  - Where do consoles fit in all this?
- Do we need to worry about it in Java?
  - Java has garbage collection
  - Handles the difficult bits for us, right?
- What can we do in LibGDX?

Gaming Memory (Generation 7)

- **Playstation 3**
  - 256 MB RAM for system
  - 256 MB for graphics card
- **X-Box 360**
  - 512 MB RAM (unified)
- **Nintendo Wii**
  - 88 MB RAM (unified)
  - 24 MB for graphics card
- **iPhone/iPad**
  - 1 GB RAM (unified)

Gaming Memory (Generation 8)

- **Playstation 4**
  - 8 GB RAM (unified)
- **X-Box One**
  - 12 GB RAM (unified)
  - 9 GB for games
- **Nintendo Wii-U**
  - 2 GB RAM (unified)
  - 1 GB only for OS
- **iPhone/iPad**
  - 2 GB RAM (unified)

Gaming Memory (Current Generation)

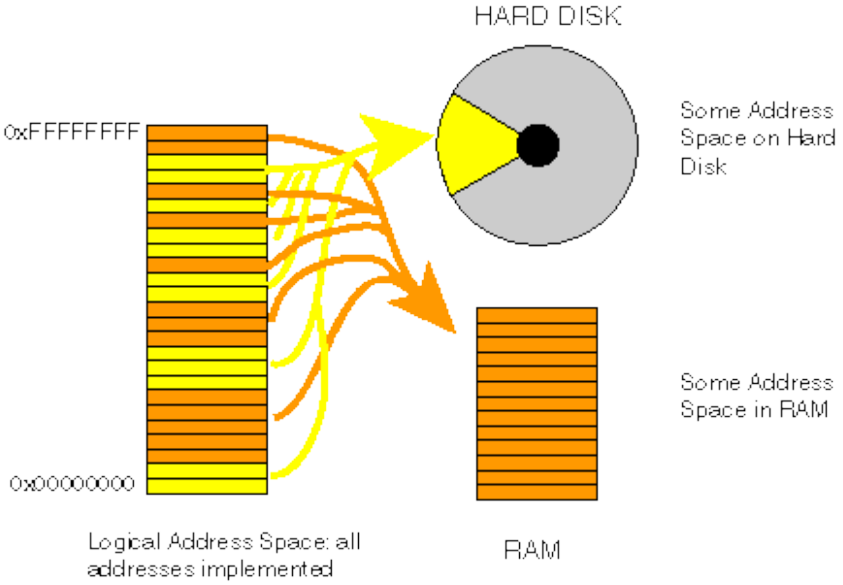
- **Playstation 5**
  - 16 GB RAM (unified)
  - **Speed** 448GB/s
- **X-Box Series X**
  - 16 GB RAM (unified)
  - **Speed** 560-336GB/s
- **Nintendo Switch**
  - 3 GB RAM (unified)
  - **Speed** 25.6 GB/s
- **iPhone/iPad**
  - 6 GB RAM (unified)
  - **Speed** 42.7 GB/s

Aside: Memory Affects Games

- **Generation 7**
  - Modern(ish) GPUs
  - Horrible memory
  - **Pretty, but short games**
- **Generation 8**
  - Minor GPU increases
  - Massive memory increases
  - **Open world games**
- **Generation 9**
  - Minor GPU, memory boosts
  - Massive bandwidth boosts
  - **Shorter loading time = ???**

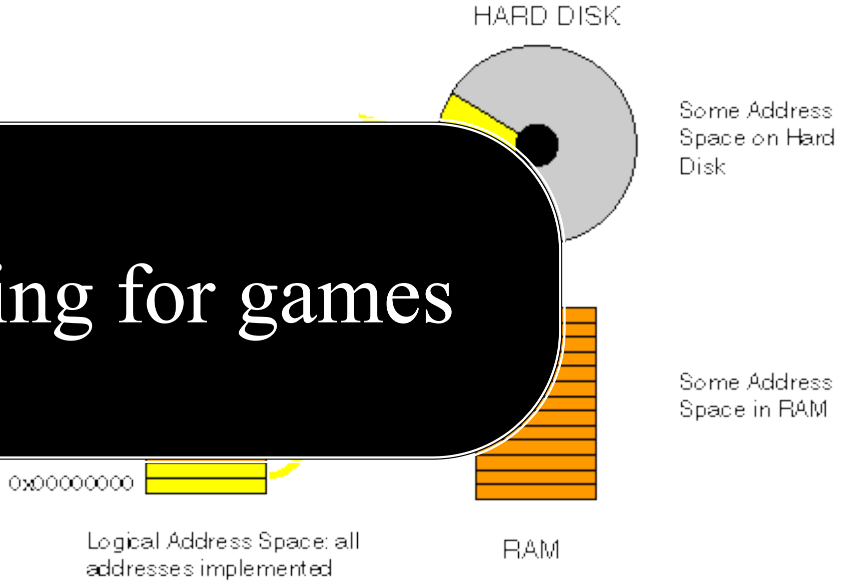
What About Virtual Memory?

- **Secondary storage** exists
  - Consoles/iPad have 1 TB
  - Most platforms solid state
- **Bandwith** is improving!
  - *Good* SSDs hit 2.5 GB/s
  - PS5 5.5 GB/s, XBX 4.8 GB/s
- BUT recall **16 ms** per frame
  - At best, can access 90 MB
  - Yields uneven performance


HARD DISK
Some Address Space on Hard Disk
0xFFFFFFFF
0x00000000
Some Address Space in RAM
Logical Address Space: all addresses implemented
RAM

What About Virtual Memory?

- **Secondary storage** exists
  - Consoles/iPad have 1 TB
  - Most platforms solid state
- **Bandwi**
  - *Good*
  - PS5 5
- BUT recall **16 ms** per frame
  - At best, can access 90 MB
  - Yields uneven performance


HARD DISK
Some Address Space on Hard Disk
Some Address Space in RAM
0x00000000
Logical Address Space: all addresses implemented
RAM

Not really a thing for games

Aside: Java Memory

- Initial heap size
  - Memory app starts with
  - Can get more, but stalls app
  - Set with –Xms flag
- Maximum heap size
  - OutOfMemory if exceed
  - Set with –Xmx flag
- Defaults by RAM installed
  - Initial 25% RAM (<64 MB)
  - Max is 75% RAM (<2 GB)
  - Need more, then set it

```
> java –cp game.jar GameMain
```

```
> java –cp game.jar –Xms:64m
  GameMain
```

```
> java –cp game.jar –Xmx:4g
  GameMain
```

```
> java –cp game.jar –Xms:64m
  –Xms:64m GameMain
```

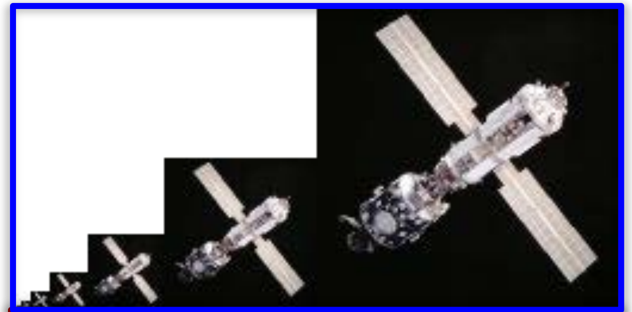
Memory Usage: Images

- Pixel color is 4 bytes
  - 1 byte each for r, b, g, alpha
  - More if using HDR color
- Image a **2D array** of pixels
  - 1280x1024 monitor size
  - 5,242,880 bytes ~ 5 MB
- More if using **mipmaps**
  - Graphic card texture feature
  - Smaller versions of image
  - Cached for performance
  - But can double memory use

Memory Usage: Images

- Pixel color is 4 bytes
  - 1 byte each for r, b, g, alpha
  - More if using HDR color
- Image a **2D array** of pixels
  - 1280x1024 monitor size
  - 5,242,880 bytes ~ 5 MB
- More if using **mipmaps**
  - Graphic card texture feature
  - Smaller versions of image
  - Cached for performance
  - But can double memory use

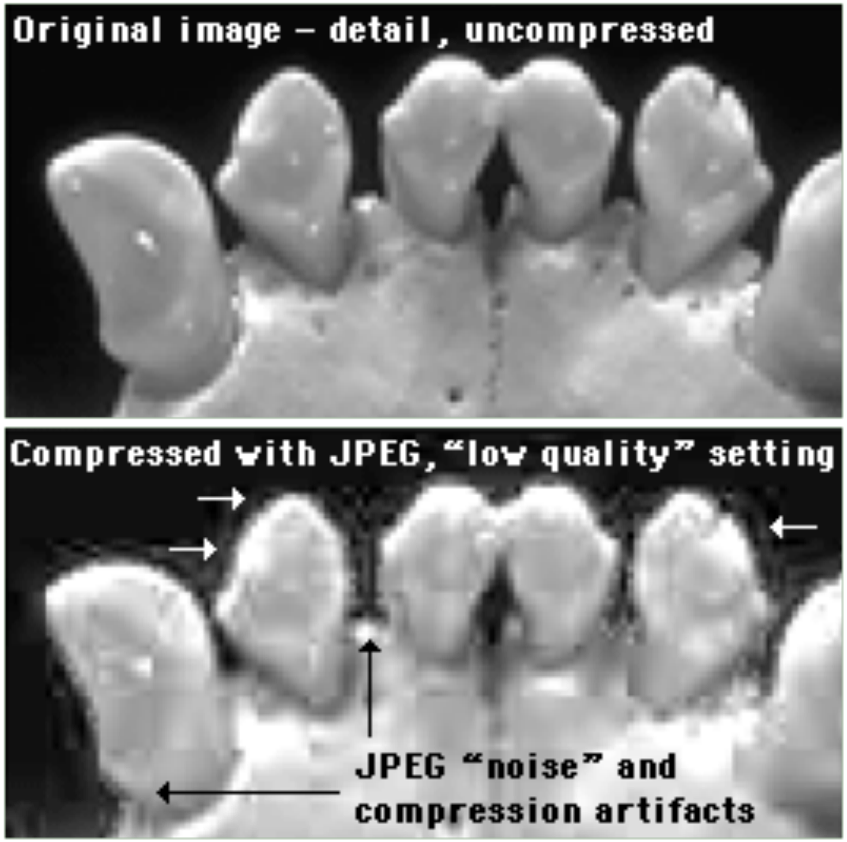
MipMaps



Original Image

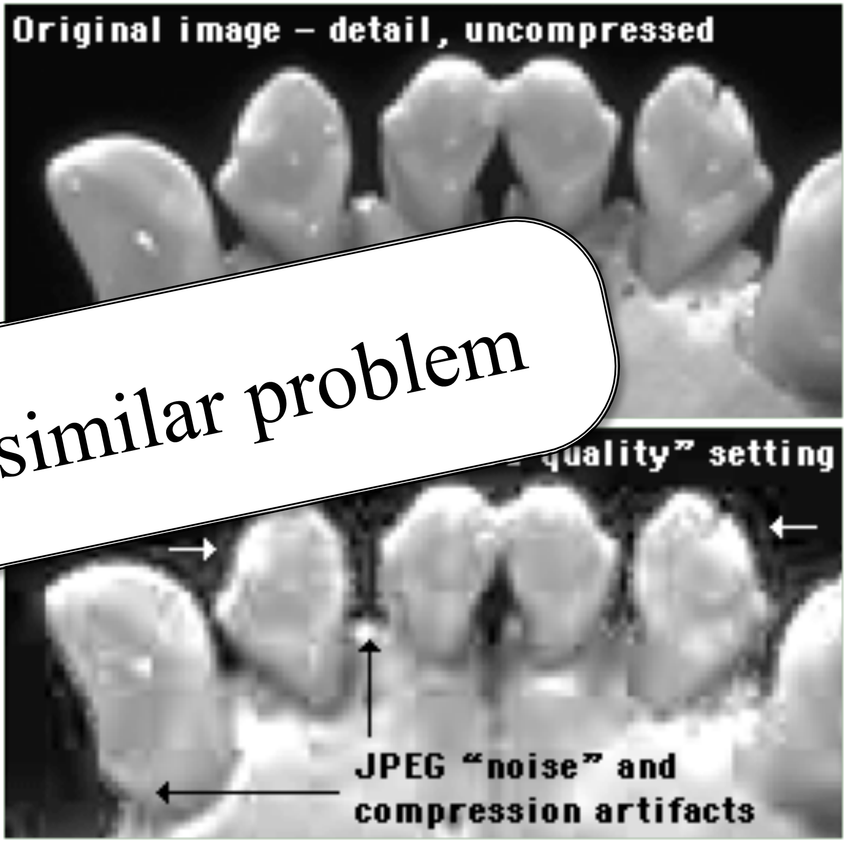
But My JPEG is only 8 KB!

- Formats often **compressed**
  - JPEG, PNG, GIF
  - But not always TIFF
- **Uncompress** to display
  - Need space to uncompress
  - In RAM or graphics card
- Only load when needed
  - Loading is primary I/O operation in AAA games
  - Causes “texture popping”


Original image – detail, uncompressed
Compressed with JPEG, “low quality” setting
JPEG “noise” and compression artifacts

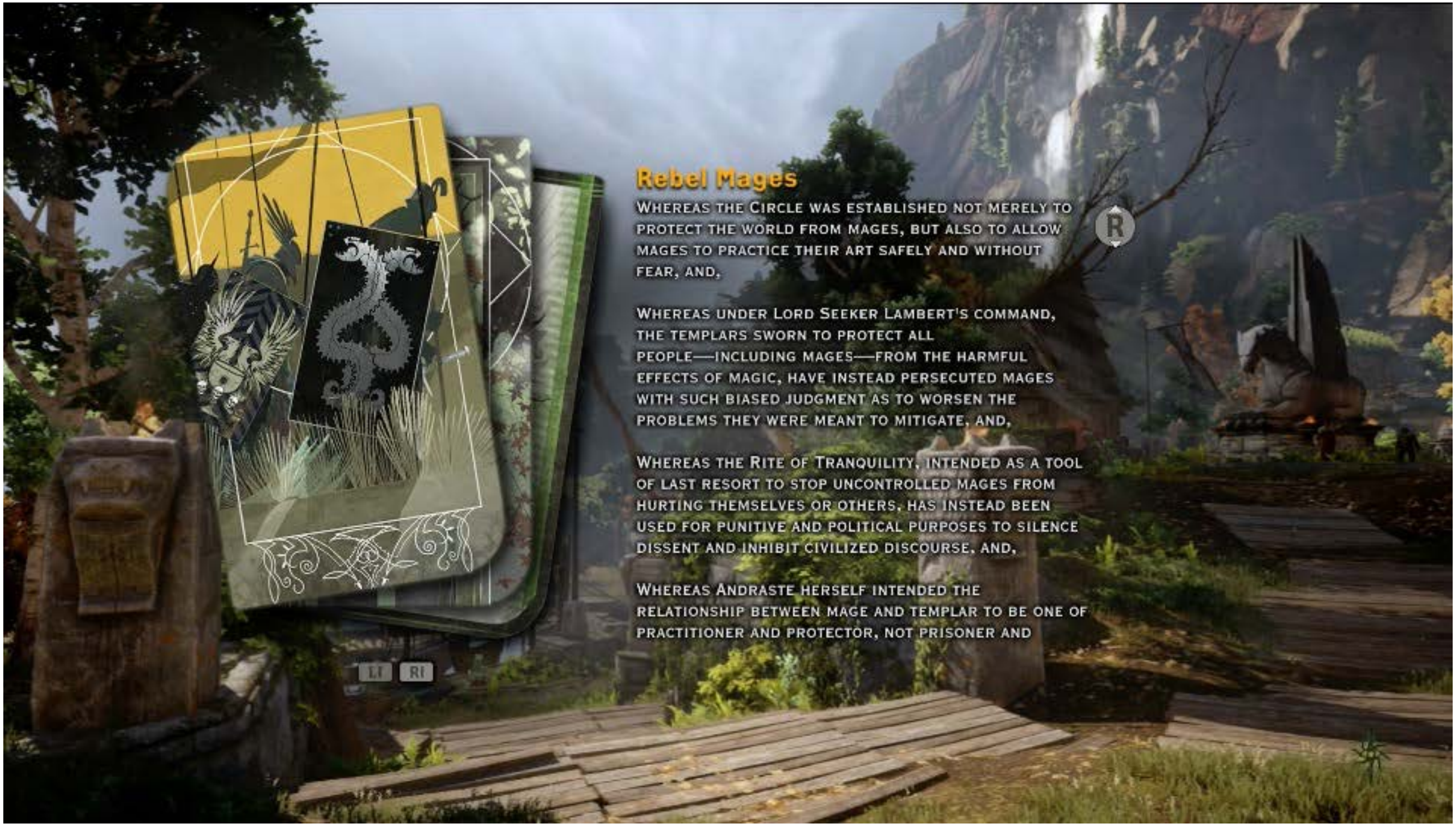
But My JPEG is only 8 KB!

- Formats often **compressed**
  - JPEG, PNG, GIF
  - But not always TIFF
- **Uncompress** to display
  - Need space to uncompr
  - In RA
- Only lo
  - Loading is primary I/O operation in AAA games
  - Causes "texture popping"


Original image – detail, uncompressed
quality" setting
JPEG "noise" and compression artifacts

Sounds have a similar problem

Loading Screens



Rebel Mages

Whereas the Circle was established not merely to protect the world from mages, but also to allow mages to practice their art safely and without fear, and,

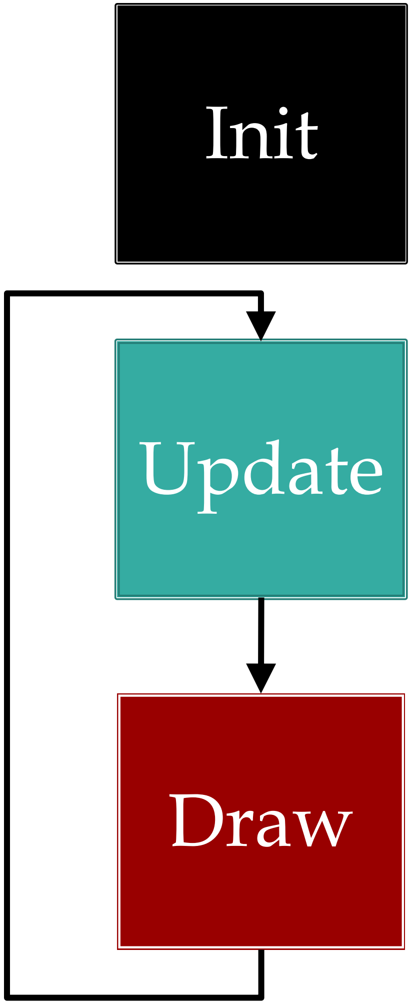
Whereas under Lord Seeker Lambert's command, the templars sworn to protect all people—including mages—from the harmful effects of magic, have instead persecuted mages with such biased judgment as to worsen the problems they were meant to mitigate, and,

Whereas the Rite of Tranquility, intended as a tool of last resort to stop uncontrolled mages from hurting themselves or others, has instead been used for punitive and political purposes to silence dissent and inhibit civilized discourse, and,

Whereas Andraste herself intended the relationship between mage and templar to be one of practitioner and protector, not prisoner and

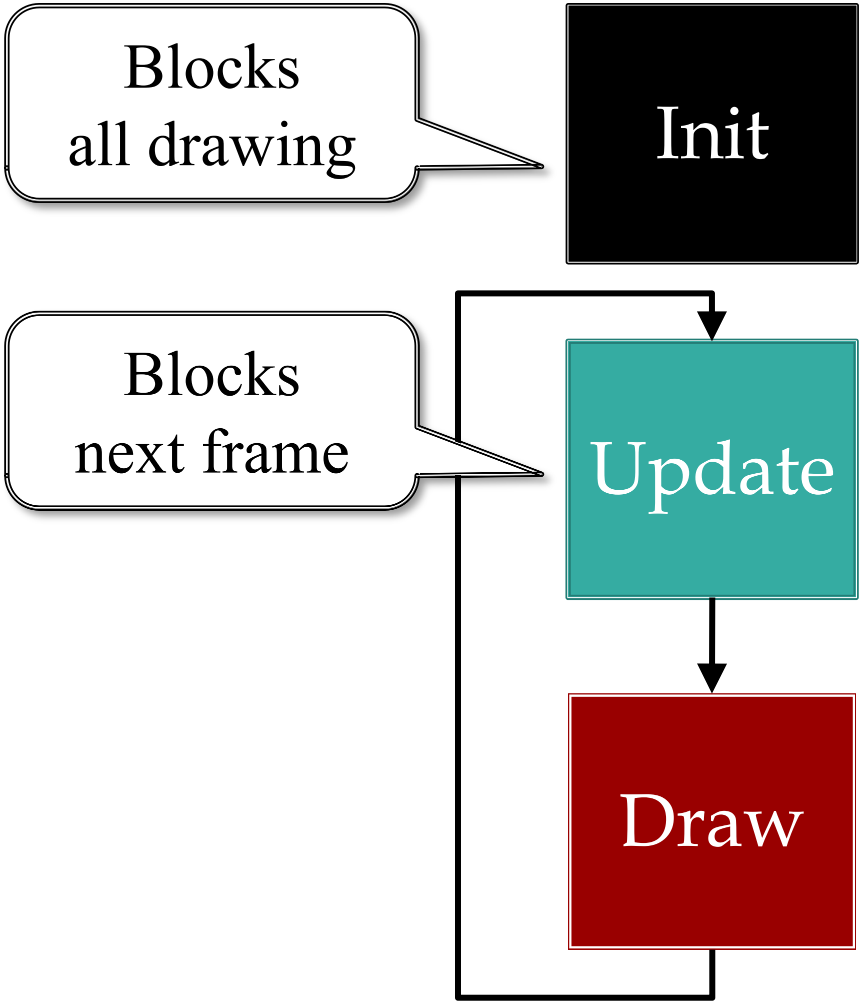
Problems with Asset Loading

- How to load assets?
  - May have a lot of assets
  - May have large assets
- Loading is **blocking**
  - Game stops until done
  - Cannot draw or animate
- May need to **unload**
  - Running out of memory
  - Free something first

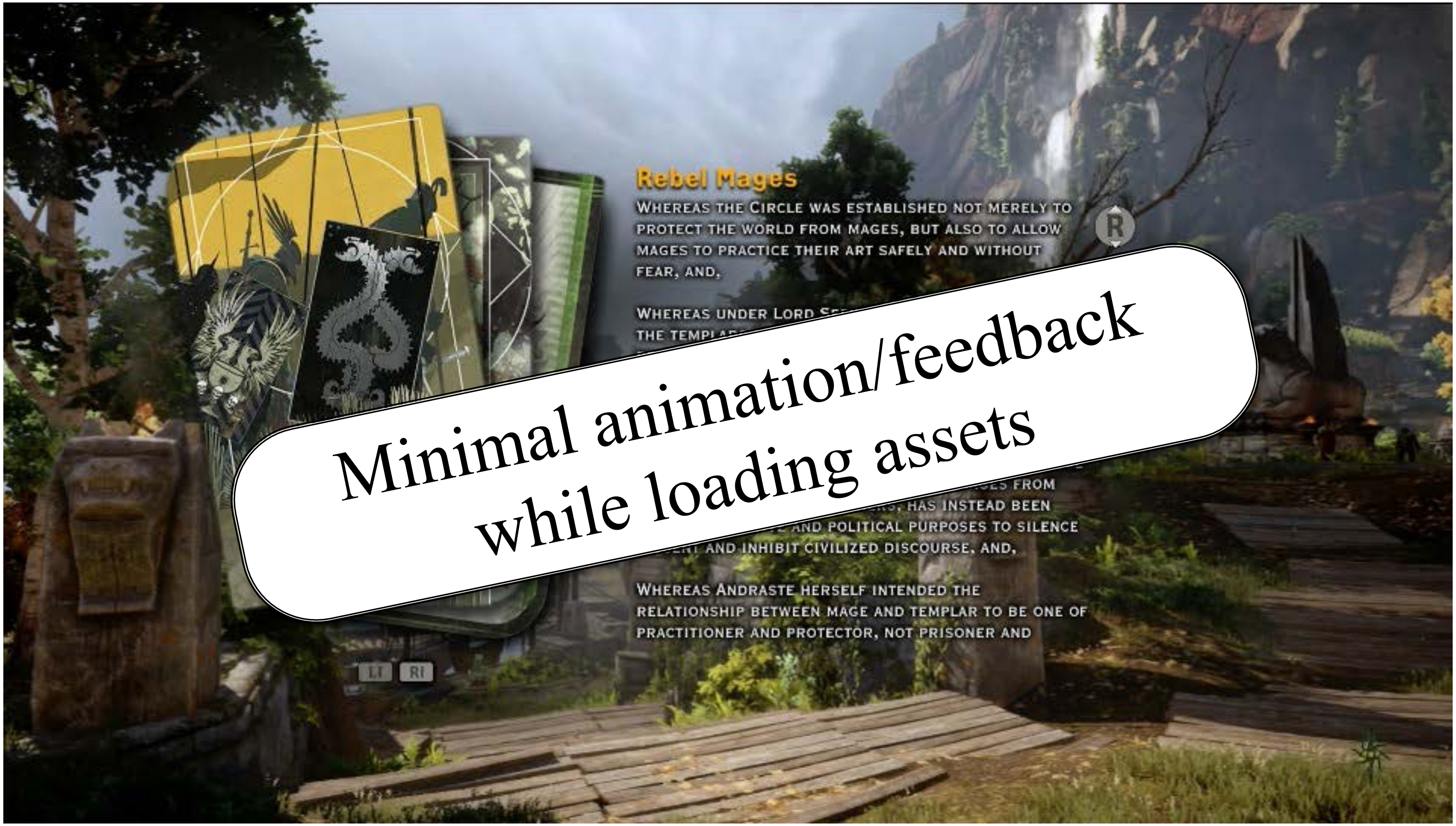


Problems with Asset Loading

- How to load assets?
  - May have a lot of assets
  - May have large assets
- Loading is **blocking**
  - Game stops until done
  - Cannot draw or animate
- May need to **unload**
  - Running out of memory
  - Free something first



Loading Screens



Rebel Mages

WHEREAS THE CIRCLE WAS ESTABLISHED NOT MERELY TO PROTECT THE WORLD FROM MAGES, BUT ALSO TO ALLOW MAGES TO PRACTICE THEIR ART SAFELY AND WITHOUT FEAR, AND,

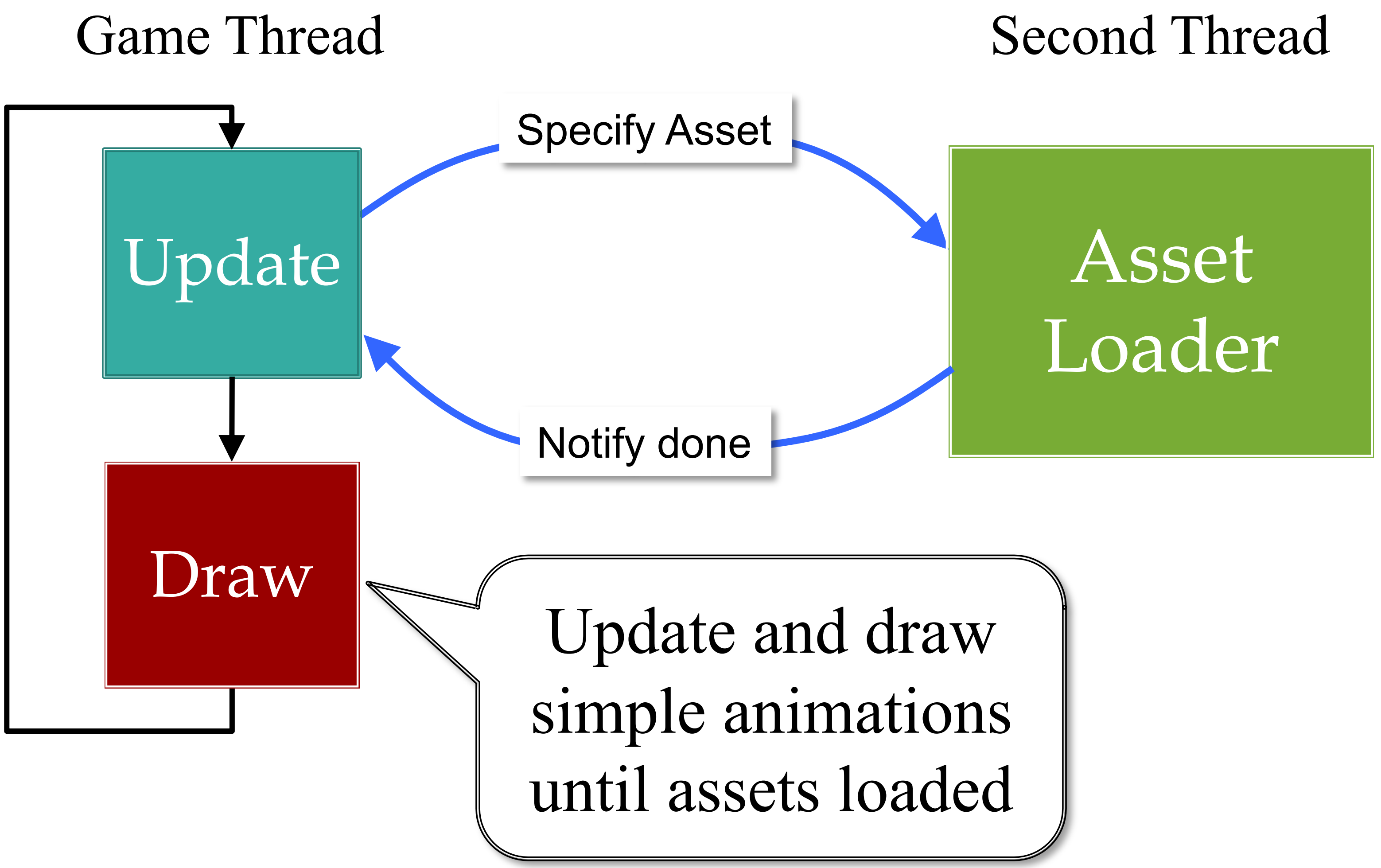
WHEREAS UNDER LORD SE[illegible] THE TEMPLA[illegible]

[illegible] ES FROM [illegible], HAS INSTEAD BEEN [illegible] AND POLITICAL PURPOSES TO SILENCE [illegible] ENT AND INHIBIT CIVILIZED DISCOURSE, AND,

WHEREAS ANDRASTE HERSELF INTENDED THE RELATIONSHIP BETWEEN MAGE AND TEMPLAR TO BE ONE OF PRACTITIONER AND PROTECTOR, NOT PRISONER AND

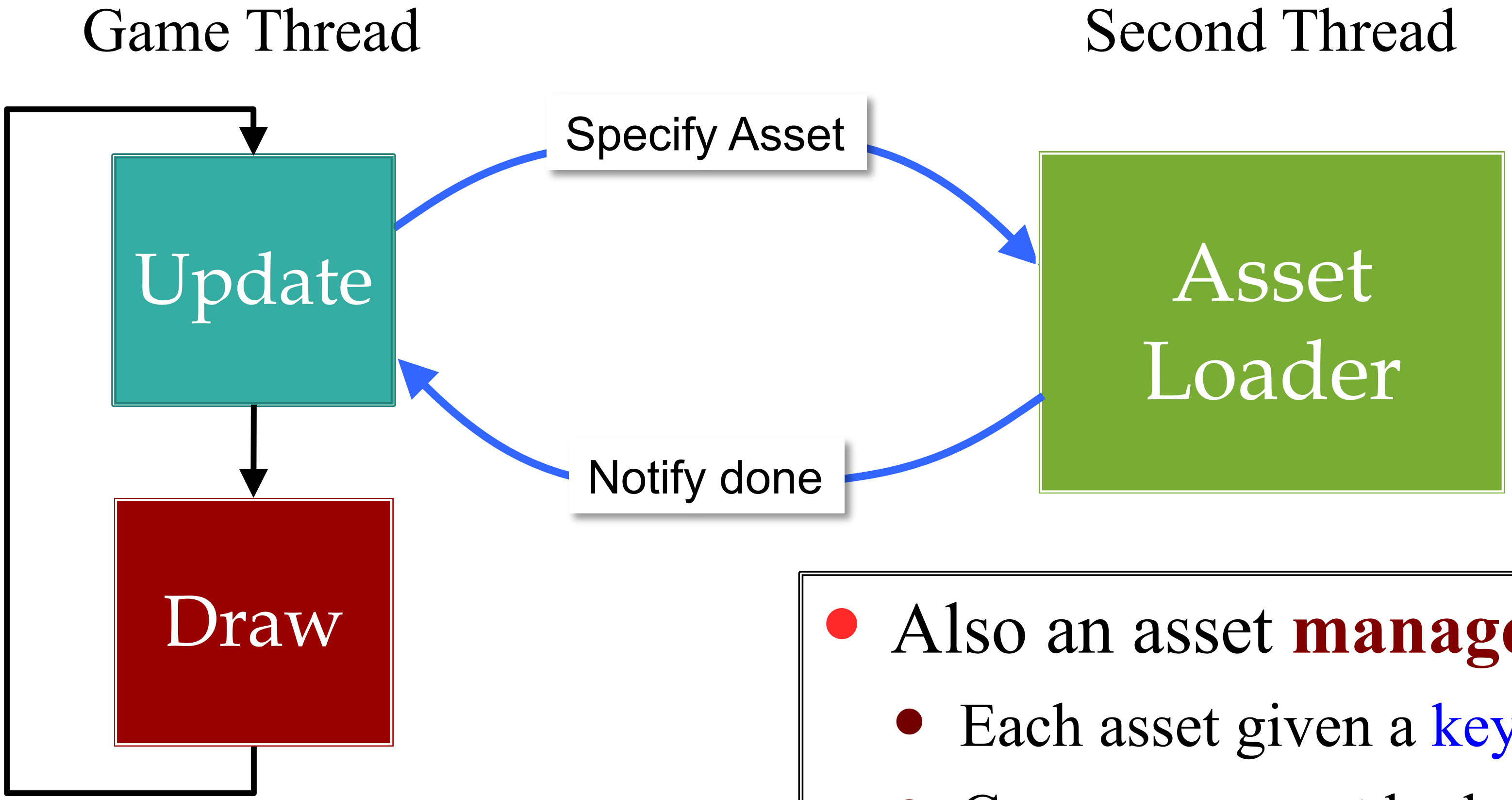
Minimal animation/feedback while loading assets

Solution: Asynchronous Loader



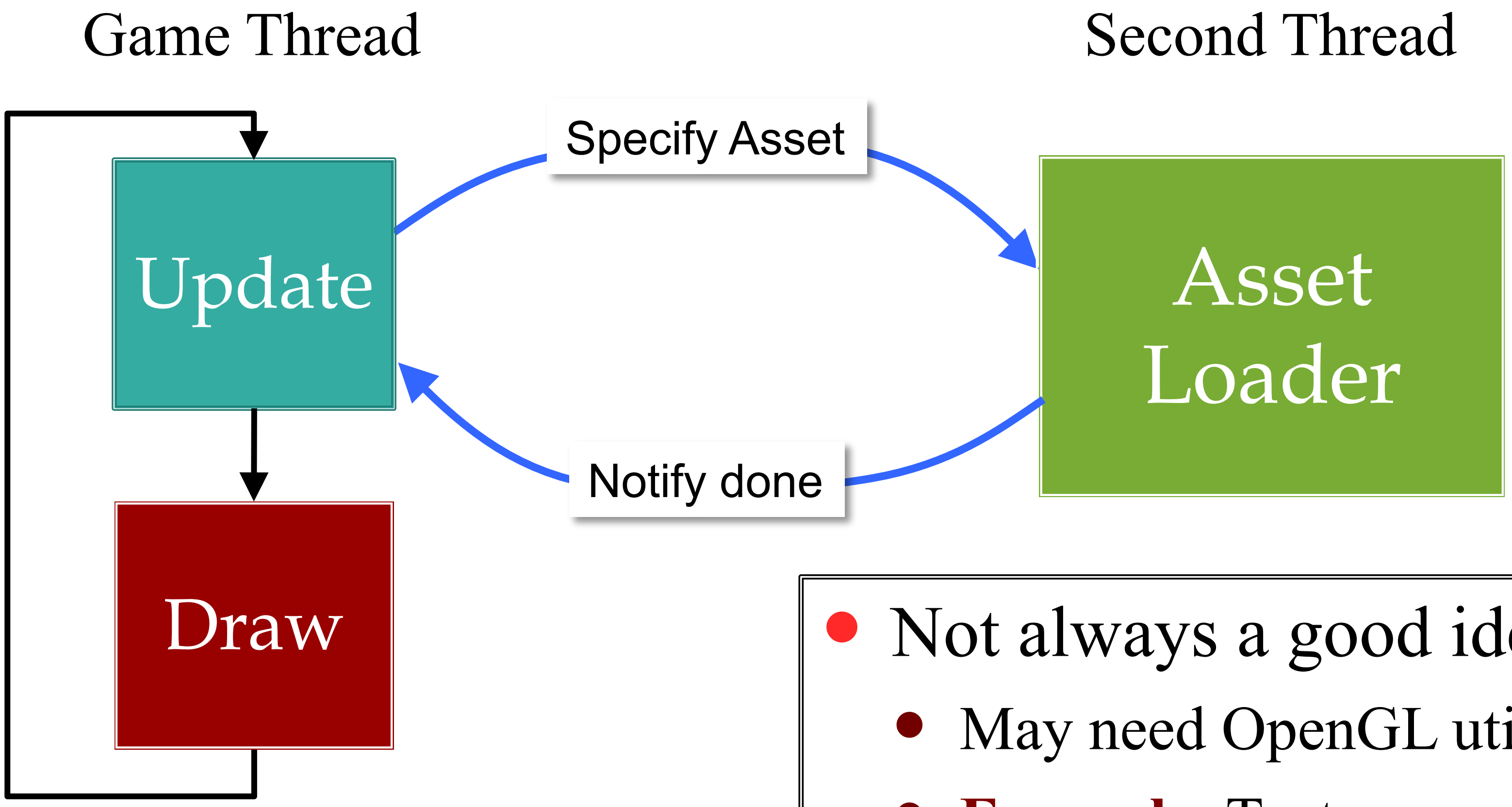
Update and draw simple animations until assets loaded

Solution: Asynchronous Loader



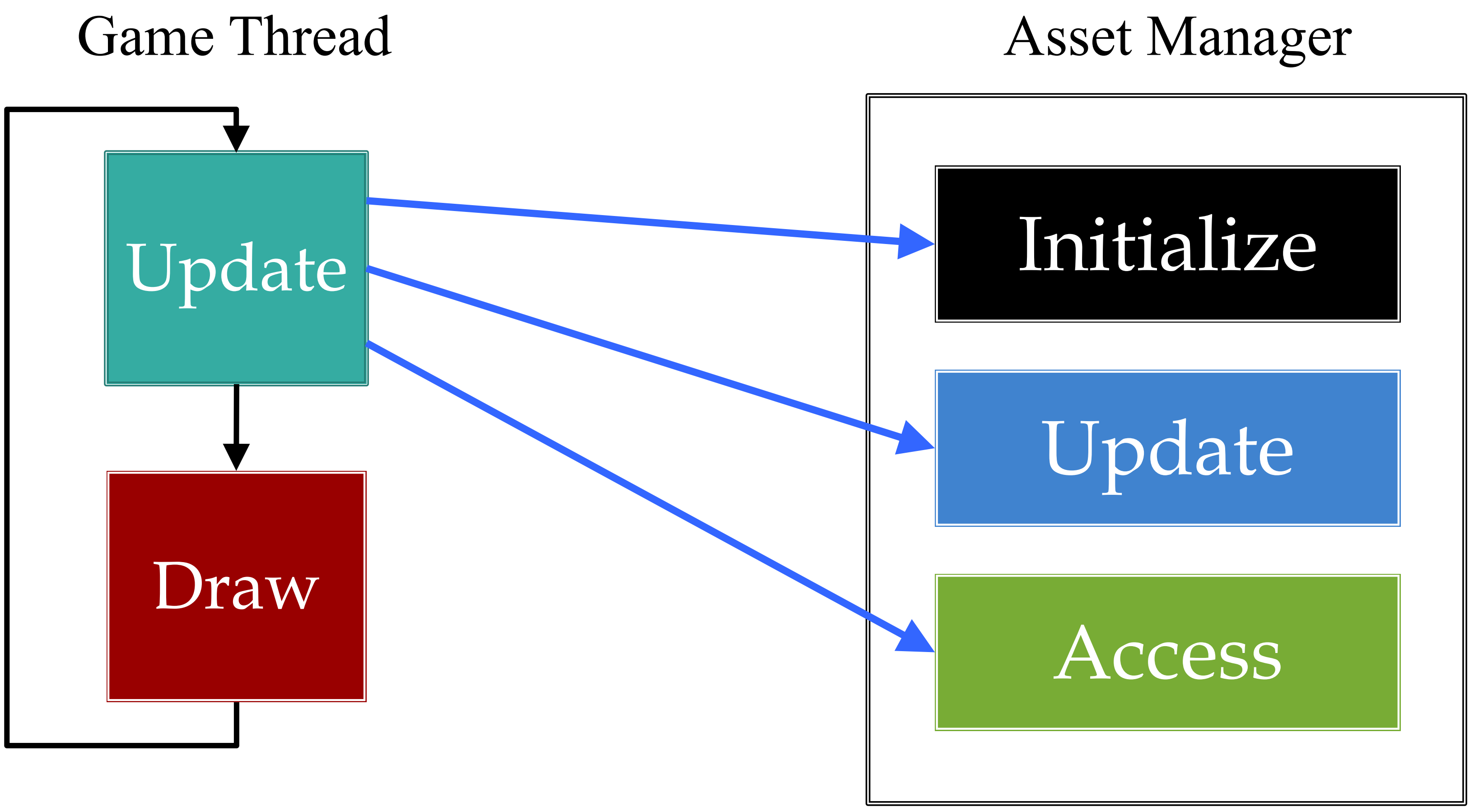
- Also an asset **manager**
  - Each asset given a key
  - Can access asset by key
  - Like a map/hashtable

Solution: Asynchronous Loader



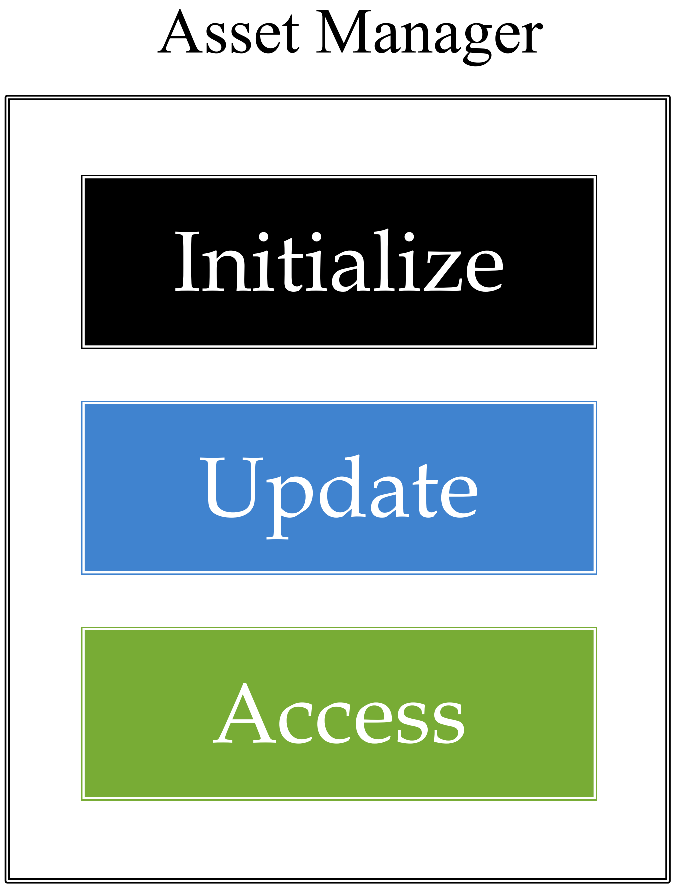
- Not always a good idea
  - May need OpenGL utils
  - **Example**: Textures
  - Limited to main thread

Alternative: Iterative Loader



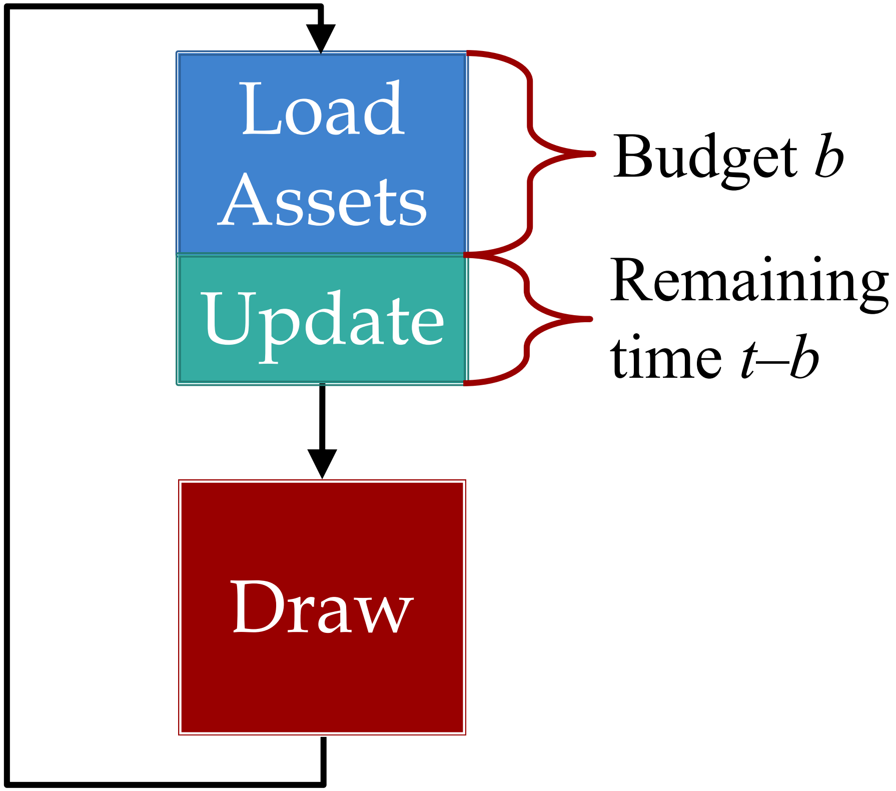
Alternative: Iterative Loader

- Uses a time budget
  - Give set amount of time
  - Do as much as possible
  - Stop until next update
- Better for OpenGL
  - Give time to manager
  - Animate with remainder
  - No resource contention
- LibGDX approach
  - But async behind scenes



Alternative: Iterative Loader

- Uses a time budget
  - Give set amount of time
  - Do as much as possible
  - Stop until next update
- Better for OpenGL
  - Give time to manager
  - Animate with remainder
  - No resource contention
- LibGDX approach
  - But async behind scenes



Assets Beyond Images

- AAA games have a lot of 3D geometry
  - Vertices for model polygons
  - Physics bodies **per polygon**
  - Scene graphs for organizing this data
- **How do we load these things?**
  - Managers handle built-in asset types
  - What if we need to make a custom data type?
- And exactly when do we load these?

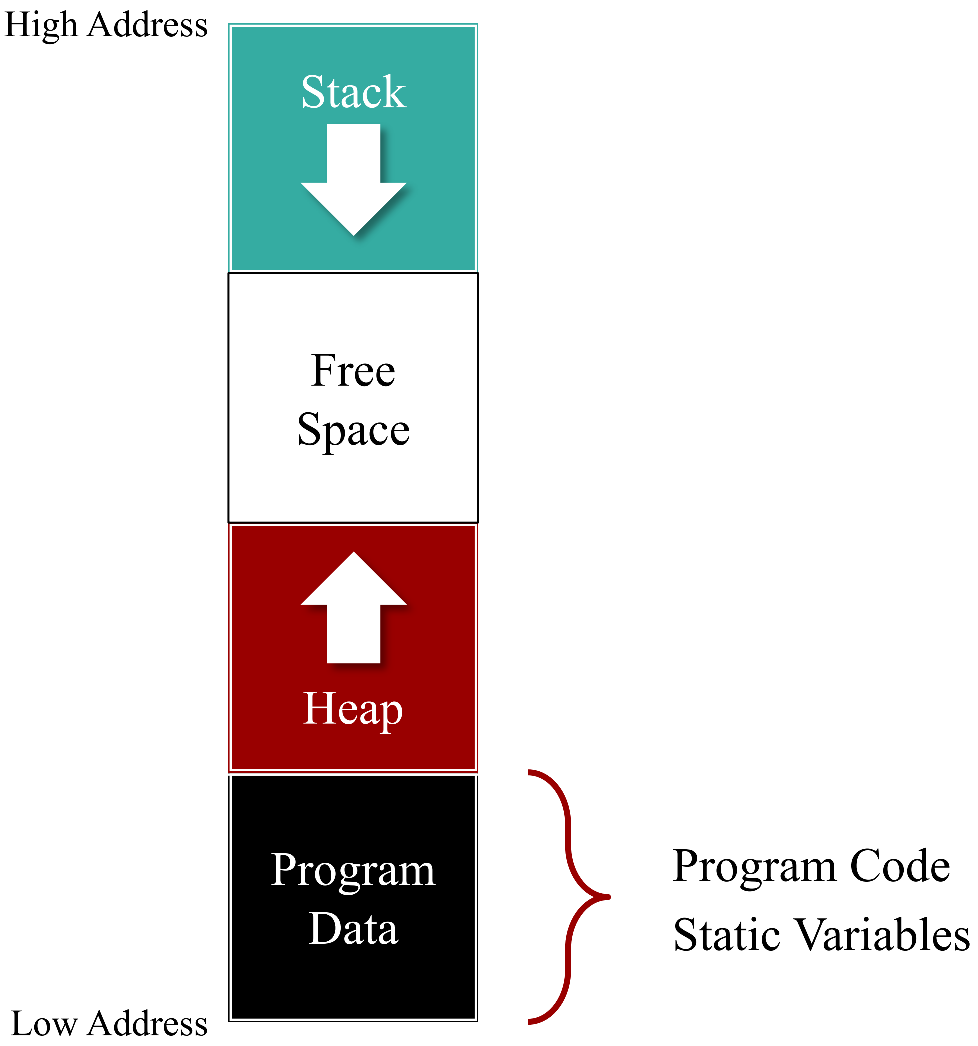
Custom Loaders in LibGDX

- The LibGDX asset system is modular
  - Use an asset manager to load/store assets
  - But each asset type has an associate **loader**
- A loader class has the following
  - Inner subclass of `AssetLoaderParameters`
  - Method `loadSync` for loading in main thread
  - Method `loadAsync` safe for separate threads
- GDIAC extensions have associated **parsers**
  - Reads asset json and sends information to loaders
  - Primarily an iterator for `AssetLoaderParameters`

Assets Beyond Images

- AAA games have a lot of 3D geometry
  - Vertices for model polygons
  - Physics bodies **per polygon**
  - Scene graphs for organizing this data
- How do we load these things?
  - Managers handle built-in asset types
  - What if we need to make a custom data type?
- **And exactly when do we load these?**

Traditional Memory Organization



High Address

Stack

Free Space

Heap

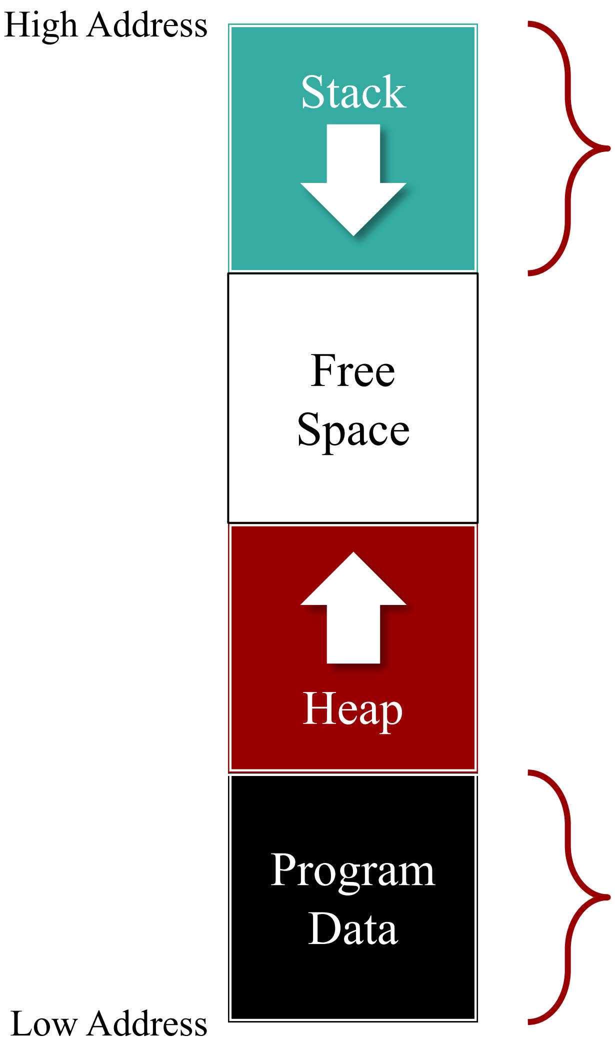
Program Data

Low Address

Program Code

Static Variables

Traditional Memory Organization



High Address

Stack

Free Space

Heap

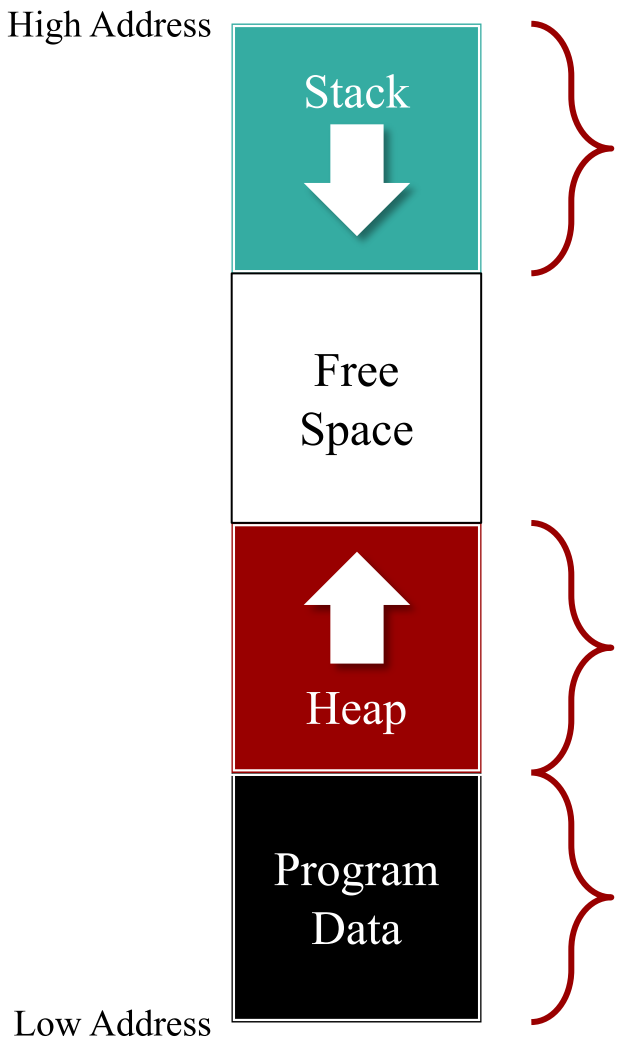
Program Data

Low Address

Function parameters
Local variables
Return values

Program Code
Static Variables

Traditional Memory Organization

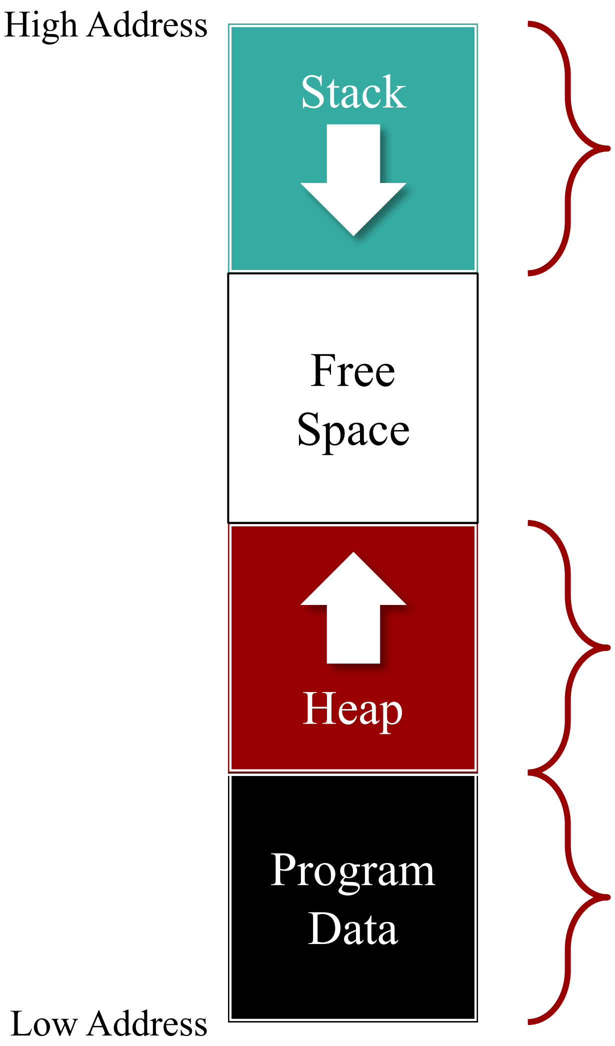


Function parameters
Local variables
Return values

Objects created via `new`
(e.g. Every object in Java)

Program Code
Static Variables

Traditional Memory Organization



Function parameters

Local variables

Return values

Easy to Handle

Objects created via `new`

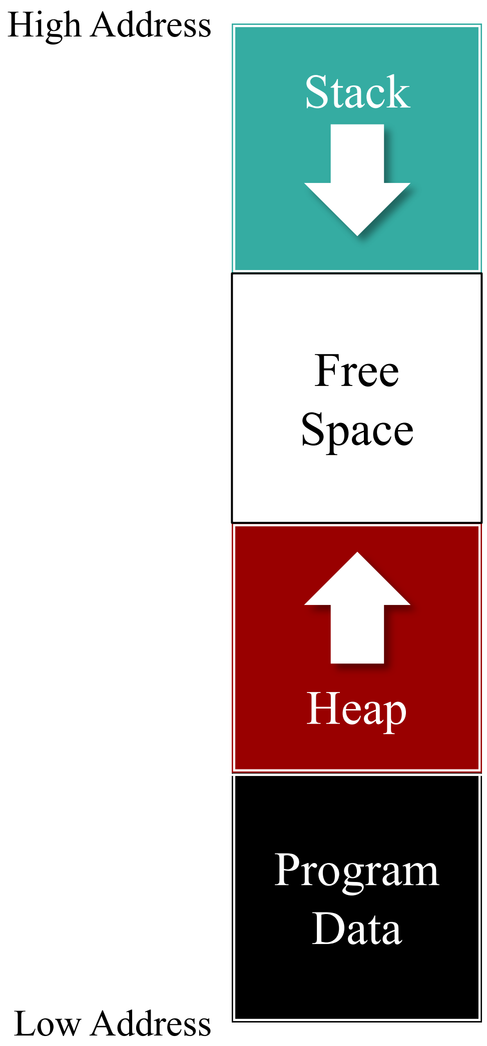
(e.g. Every object in Java)

Program Code

Static Variables

Easy to Handle

Traditional Memory Organization



High Address

Stack

Free Space

Heap

Program Data

Low Address

Function parameters

Local variables

Return values

Easy to Handle

Objects created via `new`

(e.g. Every object in Java)

Problems!

Program Code

Static Variables

Easy to Handle

Problem with Heap Allocation

- It can be slower to access
  - Not always contiguous
  - Stacks are nicer for caches
- Garbage collection is brutal
  - Old collectors would block
  - New collectors are better…
  - …but slower than manual
- Very bad if high churn
  - Rapid creation/deletion
  - **Example**: Particle systems

```
private void handleCollision(Shell s1, Shell s2) {
    // Find the axis of "collision"
    Vector2 axis = new Vector2(s1.getPosition());
    axis.sub(s2.getPosition());

    …

    // Compute the projections
    Vector2 temp1 = new Vector2(s2.getPosition());
    temp1.sub(s1.getPosition()).nor();
    Vector2 temp2 = new Vector2(s1.getPosition());
    temp2.sub(s2.getPosition()).nor();

    // Compute new velocities
    temp1.scl(temp1.dot(s1.getVelocity()));
    temp2.scl(temp2.dot(s2.getVelocity()));

    // Apply to the objects
    s1.getVelocity().sub(temp1).add(temp2);
    s2.getVelocity().sub(temp2).add(temp1);

}
```

Problem with Heap Allocation

- It can be slower to access
  - Not always contiguous
  - Stacks are nicer for caches
- Garbage collection is brutal
  - Old collectors would block
  - New collectors are better…
  - …but slower than manual
- Very bad if high churn
  - Rapid creation/deletion
  - **Example**: Particle systems

```
private void handleCollision(Shell s1, Shell s2) {
    // Find the axis of "collision"
    Vector2 axis = new Vector2(s1.getPosition());
    axis.sub(s2.getPosition());

    ...

    // Compute the projections
    Vector2 temp1 = new Vector2(s2.getPosition());
    temp1.sub(s1.getPosition()).nor();
    Vector2 temp2 = new Vector2(s1.getPosition());
    temp2.sub(s2.getPosition()).nor();

    // Compute new velocities
    temp1.scl(temp1.dot(s1.getVelocity()));
    temp2.scl(temp2.dot(s2.getVelocity()));

    // Apply to the objects
    s1.getVelocity().sub(temp1).add(temp2);
    s2.getVelocity().sub(temp2).add(temp1);

}
```

Created/deleted every frame

Aside: Stack Based Allocation

- C++ can put objs on stack
  - Object deleted at end of call
  - No GC computation at all
  - Good for short-life objects
- Java can *approximate* this
  - Checks if local to function
  - If so, will delete it
- But not a perfect solution
  - Can never **return** object
  - Init has hidden costs

```
void getCollides(Shell s1, Shell s2) {
    // Find collision axis
    Vector2 axis = new
            Vector2(s1.getPosition());
    axis.sub(s2.getPosition());
    axis.nor();
    axis.scale(s1.getRadius());

    // Find collision location
    Vector2 spot = new
             Vector2(s1.getPosition());
    spot.add(axis);

    return spot;
}
```

Aside: Stack Based Allocation

- C++ can put objs on stack
  - Object deleted at end of call
  - No GC computation at all
  - Good for short-life objects
- Java can *approximate* this
  - Checks if local to function
  - If so, will delete it
- But not a perfect solution
  - Can never **return** object
  - Init has hidden costs

```
void getCollides(Shell s1, Shell s2) {
    // Find collisio
    Vector2 axis = new
            Vector2(s1.getPosition());
    axis.sub(s2.getPosition());
    axis.nor();
    axis.scale(s1.getRadius());

    // Find collisio
    Vector2 spot =
             Vector2(s1.getPosition());
    spot.add(axis);

    return spot;
}
```

Deleted

Not Deleted

Aside: Java Garbage Collection

- **Parallel Garbage Collector** (The Default)
  - **Freezes your application** when it collects
- **Serial Garbage Collector** (`-XX:+UseSerialGC`)
  - Like PGC but better for simple programs
- **CMS Garbage Collector** (`-XX:+UseParNewGC`)
  - Concurrent mark-and-sweep rarely freezes app
- **G1 Garbage Collector** (`-XX:+UseG1GC`)
  - Even less app freezing at cost of large heap size

Aside: Java Garbage Collection

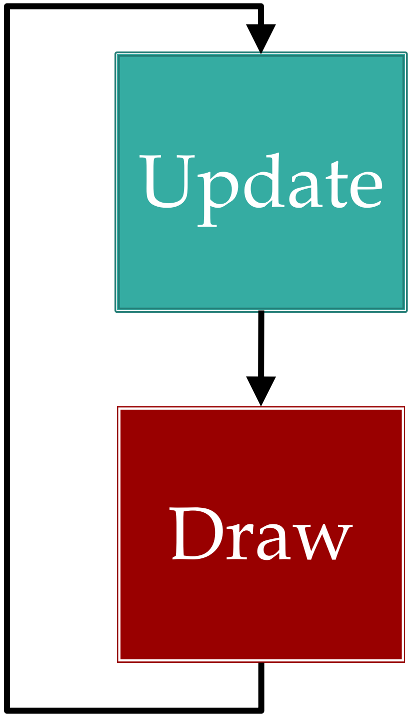
- **Parallel Garbage Collector** (The Default)
  - **Freezes your application** when it collects
- **Serial Garbage Collector** (`-XX:+UseSerialGC`)
  - Like PG
- **CMS Gar** wGC)
  - Concurrent m sweep rarely freezes app
- **G1 Garbage Collector** (`-XX:+UseG1GC`)
  - Even less app freezing at cost of large heap size

Andrew Myer's preferred GC
for high performance Java

Memory Organization and Games

Inter-Frame Memory

Carries over across frame boundaries



Intra-Frame Memory

Recovered each frame

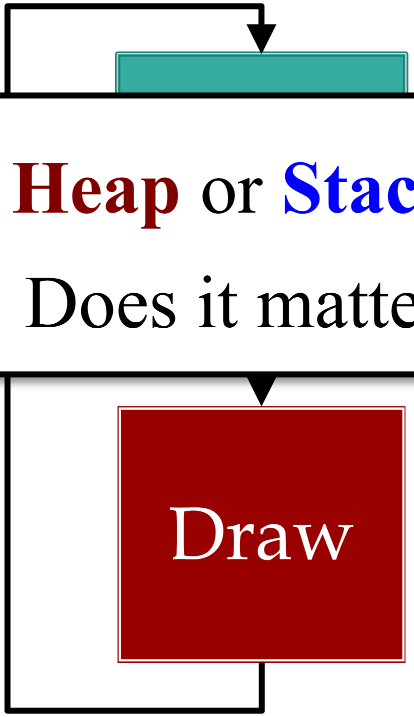
Memory Organization and Games

Inter-Frame Memory

Carries over across frame boundaries

Heap or **Stack**?

Does it matter?



Intra-Frame Memory

Recovered each frame

Distinguishing Data Types

Intra-Frame

- **Local computation**
  - Local variables
    (managed by compiler)
  - Temporary objects
    (not necessarily managed)
- **Transient data structures**
  - Built at the start of update
  - Used to process update
  - Can be deleted at end

Inter-Frame

- **Game state**
  - Model instances
  - Controller state
  - View state and caches
- **Long-term data structures**
  - Built at start/during frame
  - Lasts for multiple frames
  - May adjust to data changes

Distinguishing Data Types

Intra-Frame

- **Local computation**
  - Local variables (ma[illegible] er)
  - [illegible] jects (not necessarily managed)

Local Variables

- **Transient data structures**
  - Built at the start of update
  - Used to process update
  - Can be deleted at end

Inter-Frame

- **Game state**
  - Model instances
  - [illegible]
  - [illegible] ate and caches

Object Fields

- **Long-term data structures**
  - Built at start/during frame
  - Lasts for multiple frames
  - May adjust to data changes

Distinguishing Data Types

Intra-Frame

- **Local computation**
  - Local variables (ma… …er)
  - …jects (not necessarily managed)

Local Variables

- **Transient data structures**
  - Built at the sta… …pdate
  - …te
  - …deleted at end

e.g. Collisions

Inter-Frame

- **Game state**
  - Model instances
  - …
  - …ate and caches

Object Fields

- **Long-term data structures**
  - Built at start/d… …frame
  - …mes
  - …djust to data changes

e.g. Pathfinding

Handling Game Memory

Intra-Frame

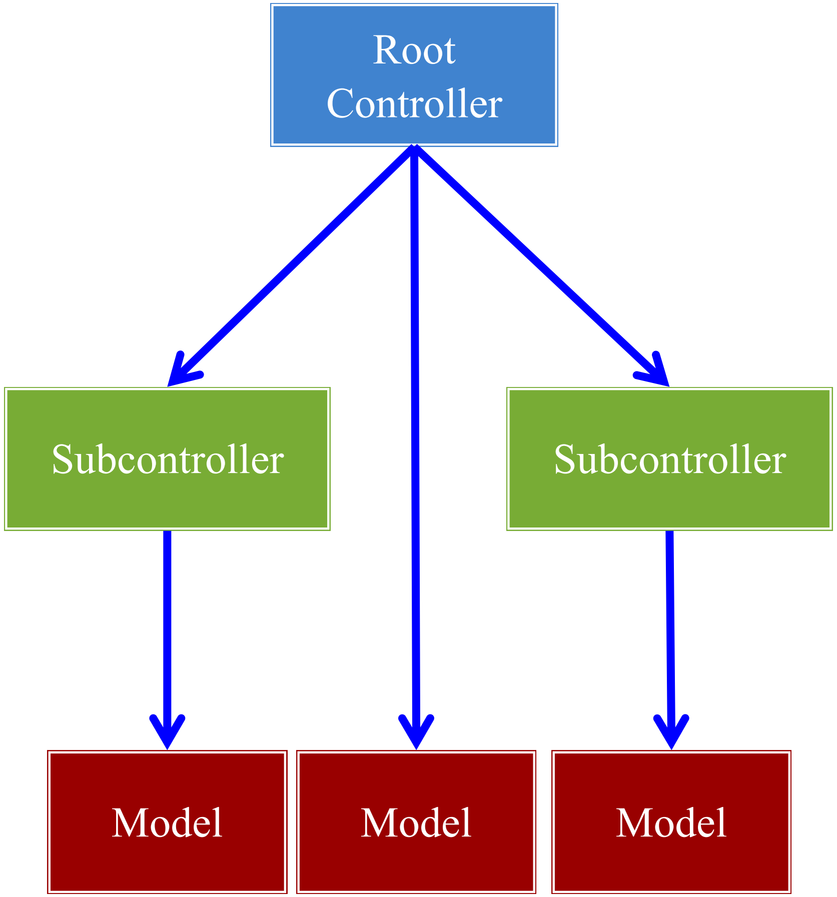
- Does not need to be kept
  - Drop the latest frame
  - Restart on frame boundary
- Want size reasonably **fixed**
  - Local variables always are
  - Limited # of allocations
  - Limit `new` inside loops
- Make use of **cached objects**
  - Requires careful planning

Inter-Frame

- May need to be saved
  - Defines current game state
  - May just want level start
- Size is more **flexible**
  - No. of objects is variable
  - Subsystems may turn on/off
  - User settings may affect
- **Preallocate** as possible
  - Recycle with **free lists**

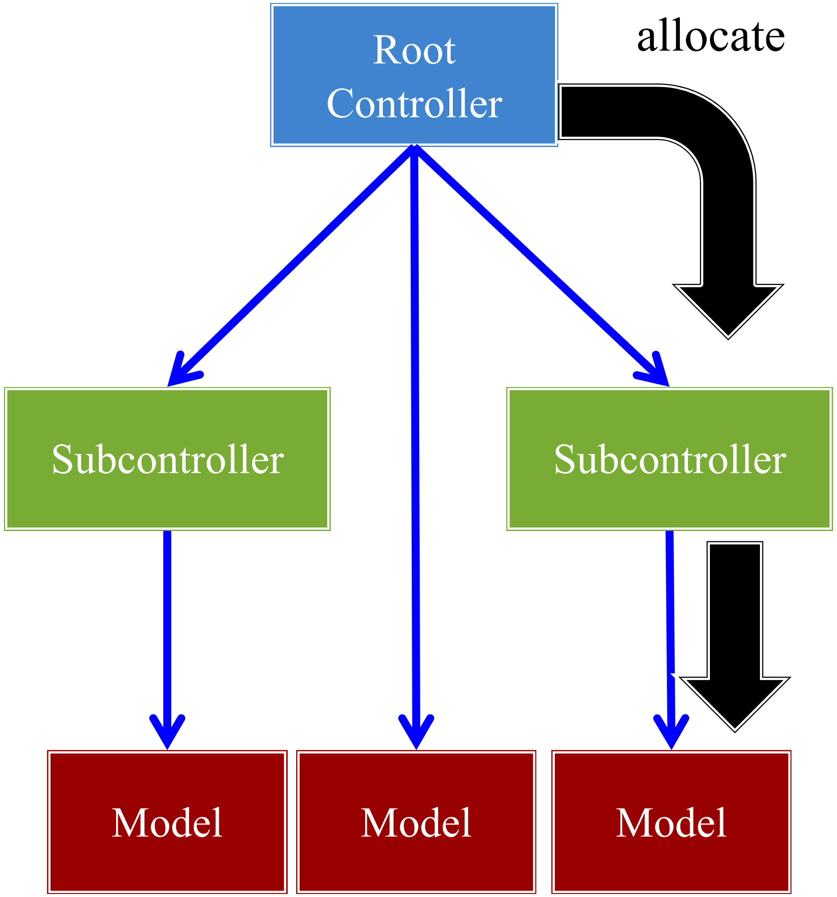
Rule of Thumb: Limiting `new`

- Limit new to constructors
  - Identify the object owner
  - Allocate in owner constructor
- **Example**: cached objects
  - Look at what algorithm needs
  - Allocate all necessary objects
  - Algorithm just sets the cache
- **Problem**: readability
  - Naming is key to readability
  - But new names = new objects
  - Make good use of comments



Rule of Thumb: Limiting `new`

- Limit new to constructors
  - Identify the object owner
  - Allocate in owner constructor
- **Example**: cached objects
  - Look at what algorithm needs
  - Allocate all necessary objects
  - Algorithm just sets the cache
- **Problem**: readability
  - Naming is key to readability
  - But new names = new objects
  - Make good use of comments



Rule of Thumb: Limiting new

- Limit new to constructors
  - Identify the object owner
  - Allocate in owner constructor
- **Example**: cached objects
  - Look at what algorithm needs
  - Allocate all necessary objects
  - Algorithm just sets the cache
- **Problem**: readability
  - Naming is key to readability
  - But new names = new objects
  - Make good use of comments

```
private void handleCollision(Shell s1, Shell s2) {
    // Find the axis of "collision"
    Vector2 axis = new Vector2(s1.getPosition());
    axis.sub(s2.getPosition());

    ...

    // Compute the projections
    Vector2 temp1 = new Vector2(s2.getPosition());
    temp1.sub(s1.getPosition()).nor();
    Vector2 temp2 = new Vector2(s1.getPosition());
    temp2.sub(s2.getPosition()).nor();

    // Compute new velocities
    temp1.scl(temp1.dot(s1.getVelocity()));
    temp2.scl(temp2.dot(s2.getVelocity()));

    // Apply to the objects
    s1.getVelocity().sub(temp1).add(temp2);
    s2.getVelocity().sub(temp2).add(temp1);

}
```

Rule of Thumb: Limiting new

- Limit new to constructors
  - Identify the object owner
  - Allocate in owner constructor
- **Example**: cached objects
  - Look at what algorithm needs
  - Allocate all necessary objects
  - Algorithm just sets the cache
- **Problem**: readability
  - Naming is key to readability
  - But new names = new objects
  - Make good use of comments

```
private void handleCollision(Shell s1, Shell s2) {
    // Find the axis of "collision"
    axis.set(s1.getPosition());
    axis.sub(s2.getPosition());

    ...

    // Compute the projections
    temp1.set(s2.getPosition());
    temp1.sub(s1.getPosition()).nor();
    temp2.set(s1.getPosition());
    temp2.sub(s2.getPosition()).nor();

    // Compute new velocities
    temp1.scl(temp1.dot(s1.getVelocity()));
    temp2.scl(temp2.dot(s2.getVelocity()));

    // Apply to the objects
    s1.getVelocity().sub(temp1).add(temp2);
    s2.getVelocity().sub(temp2).add(temp1);

}
```

Object Preallocation

- **Idea**: Allocate before need
  - Compute maximum needed
  - Create a list of objects
  - Allocate contents at start
  - Pull from list when neeeded
- **Problem**: Running out
  - Eventually at end of list
  - Want to reuse older objects
  - Easy if deletion is FIFO
  - But what if it isn’t?
- Motivation for **free list**

```
// Allocate all of the particles
Particle[] list = new Particle[CAP];
for(int ii = 0; ii < CAP; ii++) {
    list[ii] = new Particle();
}

// Keep track of next particle
int next = 0;

…

// Need to “allocate” particle
Particle p = list[next++];
p.set(…);
```

Free Lists

- Create an object **queue**
  - Separate from preallocation
  - Stores objects when "freed"
- To allocate an object…
  - Look at front of free list
  - If object there take it
  - Otherwise make new object
- Preallocation unnecessary
  - Queue wins in long term
  - Main performance hit is garbage collector

```
// Free the new particle
freelist.push(p);

...

// Allocate a new particle
Particle q;

if (!freelist.isEmpty()) {
    q = freelist.pop();
} else {
    q = new Particle();
}

q.set(...)
```

LibGDX Support: Pool

Pool<T>

- public void free(T obj);
    - Add an object to free list
- public T obtain();
    - Use this in place of new
    - If object on free list, use it
    - Otherwise make new object
- public T newObject();
    - Rule to create a new object
    - Could be preallocated

Pool.Poolable

- public void reset();
    - Erases the object contents
    - Used when object freed
- Must be implemented by T
    - Parameter free constructors
    - Set contents with initializers
- See MemoryPool demo
    - Also PooledList in Lab 4

Summary

- Memory usage is always an issue in games
  - Uncompressed images are quite large
  - Uncompressed sound files are even worse
- LibGDX supports modular asset loading
  - Loader class directs how to load asset
  - GDIAC parser reads data from JSON file
- Limit calls to new in your animation frames
  - **Intra-frame** objects: **cached objects**
  - **Inter-frame** objects: **free lists**