

Divide and Conquer: Greenthreading in Flex/AIR

Presented by Huyen Tue Dao

About me

- * Flex/AIR developer for NASDAQ OMX... No stock tips, sorry =/
- * First-time presenter... anywhere...
- * My weapon of choice in L4D is the auto-shotgun...

“...PATENTED APPROACH OF SHOOT FIRST, SHOOT LATER, SHOOT SOME MORE AND THEN WHEN EVERYBODY'S DEAD TRY TO ASK A QUESTION OR TWO.”



I AM THE STIG...OKAY, NOT REALLY BUT I DID PASS MY DRIVER'S TEST ON THE FIRST GO

Outline

- * Credit Where Credit Is Due
- * The Problem
- * Some Solutions
- * Greenthreading
- * Comparison
- * Conclusion
- * Questions

Credit Where Credit Is Due

- * Doug Knudsen: “AIR, CSVs, and Mean Greenies oh my!” @ cubicleman.com
- * Charlie Hubbard, “Actionscript and Concurrency” @ wrongnotes.blogspot.com
- * Drew Cummins, “Green Threads” @ blog.generalrelativity.org

The Problem

- ✱ Flash Player: Great at somethings like animation, not so great at others like parsing large amounts of data.
- ✱ Flash Player can execute certain things concurrently.
- ✱ However, it can get hung up if one of these things requires lots of processing and not necessarily something complex.
- ✱ Let me show you...

The Problem

- ✱ The problem revolves around Flash Player's event-driven timeline/movie paradigm.
- ✱ Flash movies/SWFs: content divided into frames played according to a timeline.

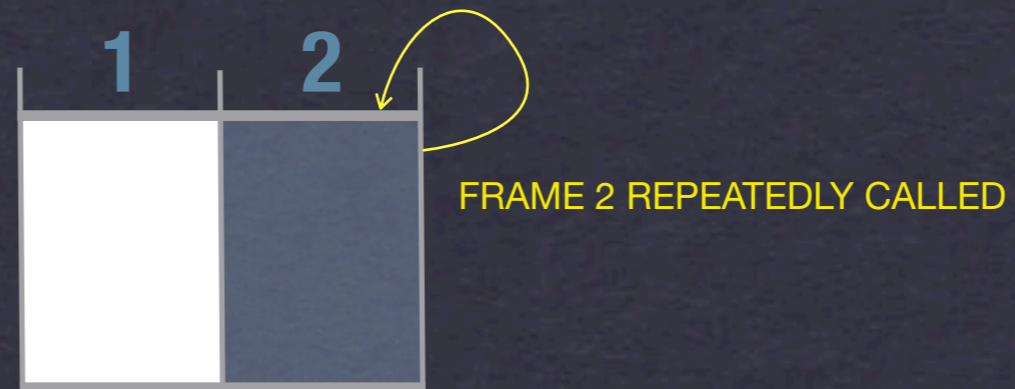
FLASH SWF

TIMELINE



FLEX SWF

TIMELINE



FRAME 1

FRAME 2

BOOTSTRAPPING/
PRELOADER

APPLICATION CODE

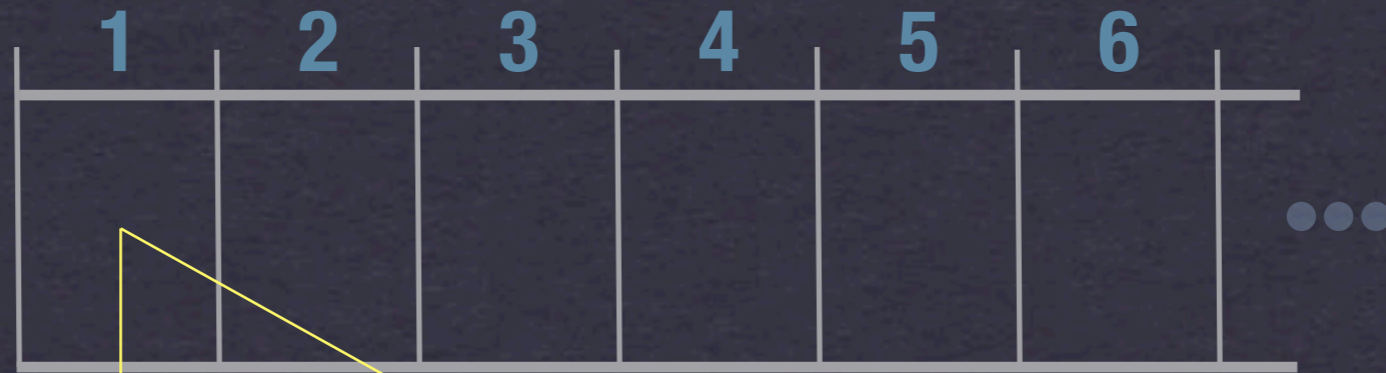
WHAT THE SWF KIND OF LOOKS LIKE

ARTIST (NOT-SO-MUCH) RENDERING

The Problem

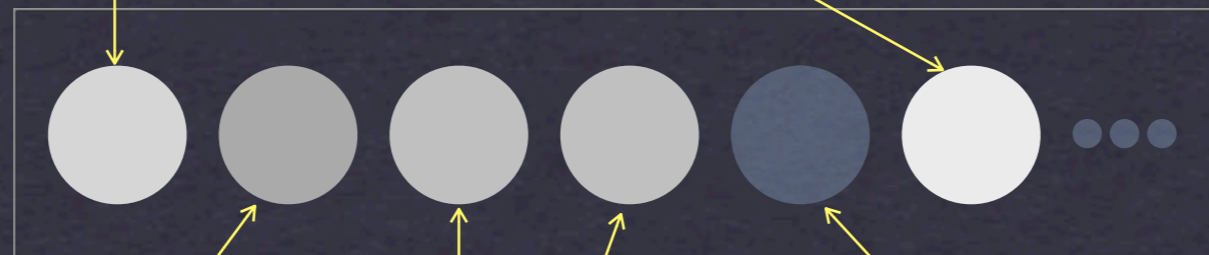
- * Flash platform is event-driven: most everything is triggered by an event.
- * Events collected into the Event Queue.
- * To get to the next frame? An event is triggered.

TIMELINE



EVENTS DISPATCHED IN FRAME 1

EVENT QUEUE



KEYBOARD EVENT

MOUSE EVENTS

***FRAME EVENT**

THE EVENT QUEUE

ARTIST (NOT-SO-MUCH) RENDERING

The Problem

- * SWF plays at a certain number of frames per second (FPS). The time per frame is $1/\text{FPS}$.
- * Default FPS is 24. Therefore, a frame needs to execute every ~42 milliseconds.
- * If code running in a frame takes longer than allotted time, the Event Queue gets held up: the application stalls.
- * Hello, non-responsive window/beach ball of death.

Some Solutions

- ✱ Divide and Conquer!
- ✱ General idea: break up long-running code into smaller batches triggered at different times, different frames.
- ✱ Give Event Queue chance to catch up. Allow other events to be processed (including the important screen-updating events).

Some Solutions

- * Several ways to break up the process:
 - * ENTER_FRAME: run a batch explicitly at each frame
 - * Timers: run a batch every x milliseconds
 - * callLater(): Push a batch-triggering event onto the Event Queue

Some Solutions

AND NOW FOR A LITTLE CODE...

```
// To break up an algorithm via a ENTER_FRAME event.  
// 1. Initialize progress variables.  
// 2. Add a listener to the application for the ENTER_FRAME event the does  
//     main work of the algorithm.  
// 3. In handler check for stopping condition and remove the listener if it  
//     has been reached.  
  
var numIterations : int = 0;  
var totalIterations : int = 100000;  
Application.application.addEventListener(Event.ENTER_FRAME, runLoop);  
  
private function runLoop(event : Event) : void  
{  
    // Do work of iteration here.  
    if(numIterations >= totalIterations)  
    {  
        Application.application.removeEventListener(Event.ENTER_FRAME, runLoop);  
    }  
}
```

Some Solutions

AND NOW FOR A LITTLE CODE...

```
// To break up an algorithm via a Timer  
// 1. Initialize progress variables.  
// 2. Initialize a Timer with a delay. Here delay ≈ time/frame.  
// 3. Add listener for TimerEvent.TIMER that does main work of algorithm.  
// 4. In handler check for stopping condition and call stop() on Timer if it  
//     has been reached.  
  
var numIterations : int = 0;  
var totalIterations : int = 100000;  
var timer : Timer = new Timer(1000 / Application.application.stage.frameRate);  
timer.addEventListener(TimerEvent.TIMER, runLoop);  
timer.start();  
  
private function runLoop(event : TimerEvent) : void  
{  
    // Do work of iteration here.  
    if(numIterations >= totalIterations)  
    {  
        timer.stop();  
    }  
    // Else next iteration will execute next TimerEvent.TIMER event.  
}
```

Some Solutions

- * The win: Event Queue can process other events between batches. Application remains responsive.
- * The fail:
 - * Can take much longer to finish the job since it is spread out over time.
 - * Trying to re-configure the batch sizes can be time-consuming trial-and-error for the developer.

Greenthreading

- * What is a “green thread?”
 - * Developers can't use system threads.
 - * Can emulate threads, i.e., green threads.
- * Extend the idea of fine-tuning amount of work done per batch to fit as much work as possible while leave some time for other events to process.

Greenthreading

- * Encapsulate all this as a class: GreenThread by Hubbard.
- * <http://code.google.com/p/greenthreads/>
- * Hubbard's class also provides statistics, progress monitoring via events.
- * To use: convert your code block, algorithm, job etc. to a GreenThread.

Greenthreading

AND NOW FOR A LITTLE CODE...

```
public class MyGreenThreadAlgorithm extends GreenThread
{
    // Called when start() called on a GreenThread
    override public function initialize() : void
    {
        // Initialize variables needed.
        progress = 0;
        maximum = 100;
    }

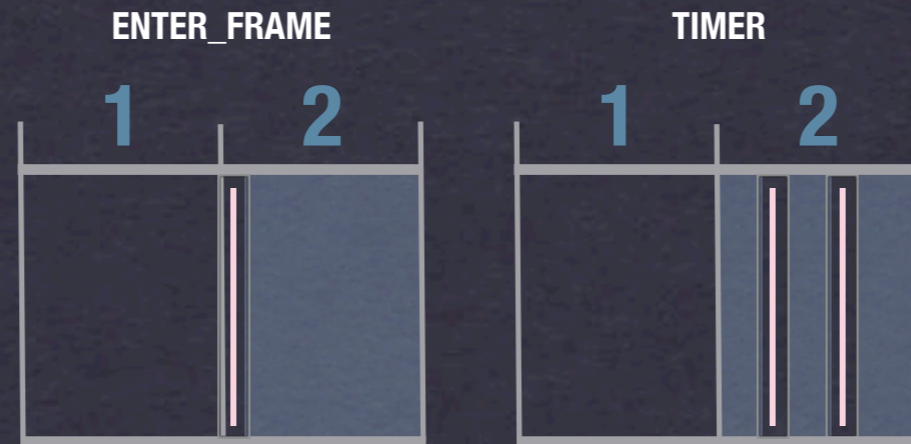
    override public function run() : Boolean
    {
        // Do work of one iteration here.
        progress++;
        // Return true if done, false otherwise.
    }
}
```

DIVIDE + CONQUER: GREENTHREADING

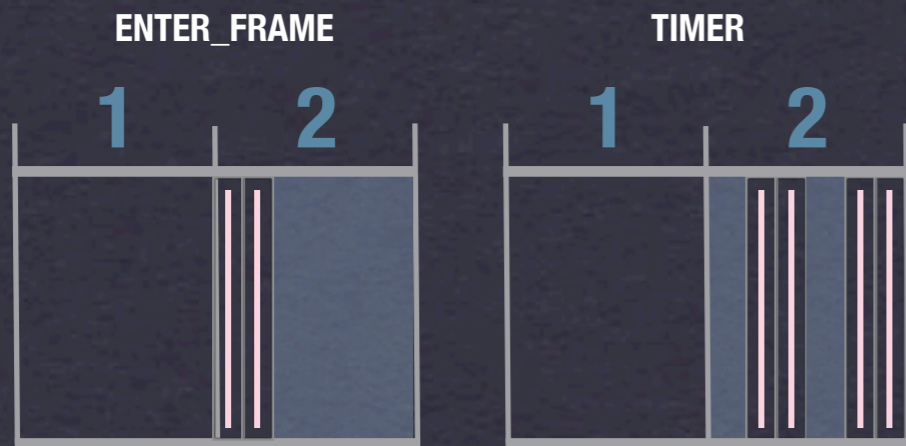
ALGORITHM



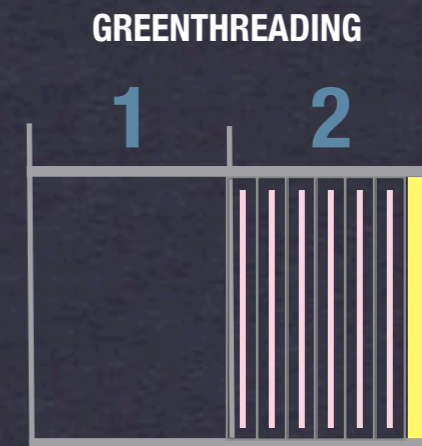
DIVIDE + CONQUER



INCREASE ITERATIONS
RUN PER FRAME



SPECIFY **DELTA**: SET AMOUNT
OF TIME LEFT IN FRAME FOR
PROCESSING OTHER EVENTS.
FILL UP REMAINING FRAME
TIME WITH ITERATIONS



GREENTHREADING VS OTHER SOLUTIONS

ARTIST (NOT-SO-MUCH) RENDERING, NOT DRAWN TO SCALE

	Original	Timer (ENTER_FRAME)	GreenThread
Runtime	Fastest*	Slow - Fast*	Fast
Responsive?	NO	YES - NO	YES

*If the application doesn't crash that is.

DEMO COMPARISON

RUNTIME AND RESPONSIVENESS

Conclusion

- ✱ Applications with computation-heavy or long-running code can hang because Event Queue stalls.
- ✱ To keep application responsive, divide work into small jobs triggered individually, give Event Queue a chance to process other events.
- ✱ Greenthreading provides illusion of concurrency by dividing up the work, while fitting as much work as possible per frame. Application remains responsive.
- ✱ Costs of greenthreading: more complex code, little longer runtime.

Questions?

THANKS FOR COMING!

HUYEN TUE DAO

DAOTUEH@GMAIL.COM

QUEENCODEMONKEY @ TWITTER

WWW.QUEENCODEMONKEY.COM

DIVIDE + CONQUER: GREENTHREADING

Links

- * Greenthreading

- * <http://www.cubicleman.com/2009/03/08/air-csvs-and-mean-greenies-oh-my/>

- * <http://wrongnotes.blogspot.com/2009/02/concurrency-and-actionscript-part-i-of.html>

- * <http://blog.generalrelativity.org/actionscript-30/green-threads/>

- * Frame execution model

- * <http://www.craftymind.com/2008/04/18/updated-elastic-racetrack-for-flash-9-and-avm2/>

- * <http://www.onflex.org/ted/2005/07/flash-player-mental-model-elastic.php>