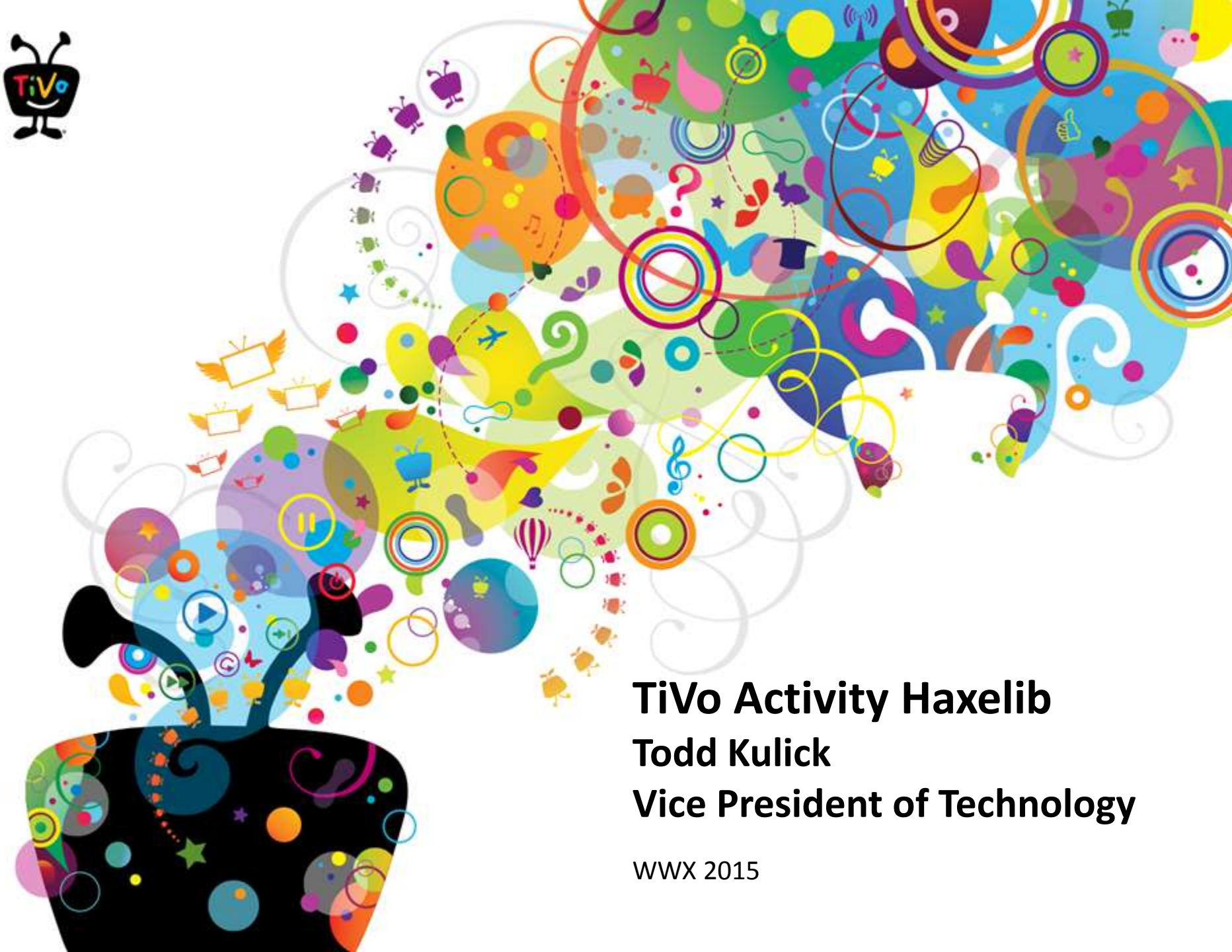


TiVo Activity Haxelib

Todd Kulick

Vice President of Technology

WWX 2015

Overview

- What is the activity haxelib?
- Use Cases
- Concurrency Paradigms
- Design Points
- Conceptual Concurrency Model
- Activity API
- Wrap Up & Questions

What is the Activity haxelib?

A *simple*, *flexible* mechanism for allowing the use of *composable* concurrency within a *portable* Haxe program.

- *Simple* – Easy to understand, supports Haxe programming constructs like closures cleanly
- *Flexible* – Does not dictate your programming paradigm
- *Composable* – Provides a model for combining independent libraries without onerous coordination
- *Portable* – Runs on many platforms whether they support multi-threaded, multi-CPU environments or not

What is the Activity haxelib?

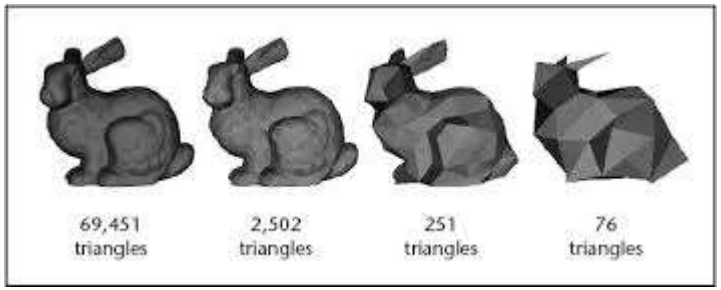
A *simple*, *flexible* mechanism for allowing the use of *composable* concurrency within a *portable* Haxe program.

Open Source
Apache 2.0 license

https://github.com/TiVo/activity

Example Use Cases

- Enhance OpenFL application with additional threads for things like concurrent HTTP download, background work threads, etc.
- Add multi-threaded physics library to OpenFL game.
- Add multi-threaded rendering/occlusion/tesselation engine to Lime.
- More generally, add library with concurrent computation functionality to single-threaded environment/toolkit.



Example Use Cases

- Provide reactive programming model without requiring control of (or even coordination with) the platform main loop.

- Most generally, build libraries with concurrent functionality that can be shipped to many platforms (POSIX, Android, iOS) without requiring platform specific code.

Concurrency Paradigms

July 30, 2015

Concurrency

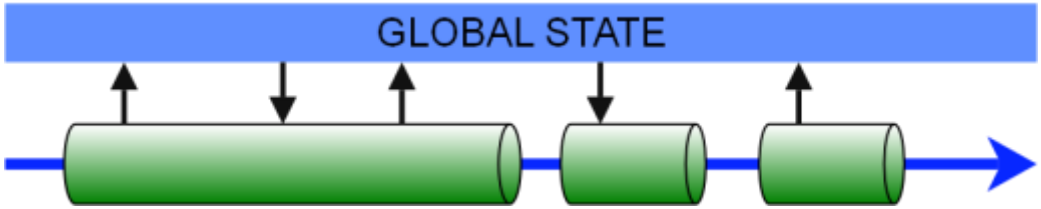
A *programming model* allowing for multiple contexts of independent execution.

Not quite the same thing as multi-threaded.

When executed, a concurrent program may or may not use multiple, simultaneous threads or CPUs.

Single-Threaded Model

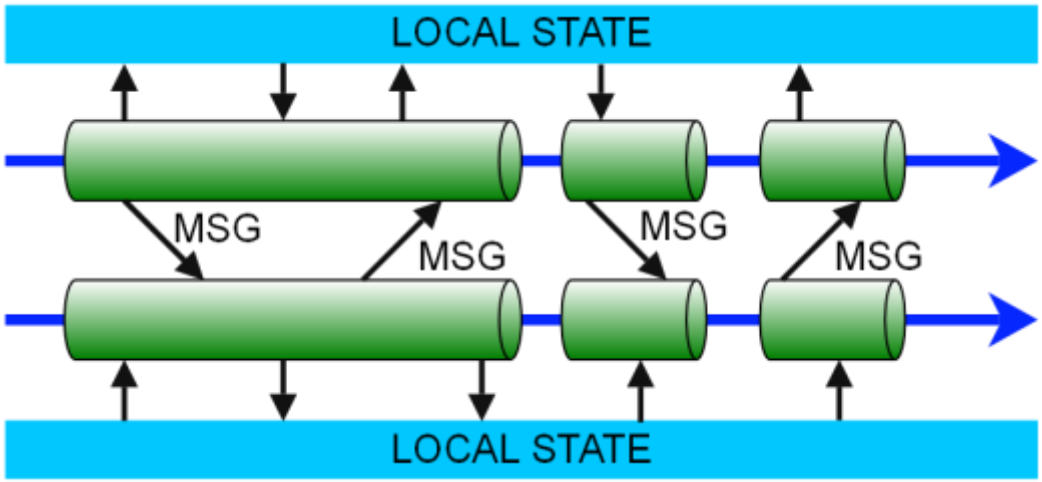
- Only a single thread of execution
- Global state and thread local are the same thing
- No thread synchronization required
- Unable to benefit from concurrency on support platforms



Examples: Python, php, Flash, web browser*

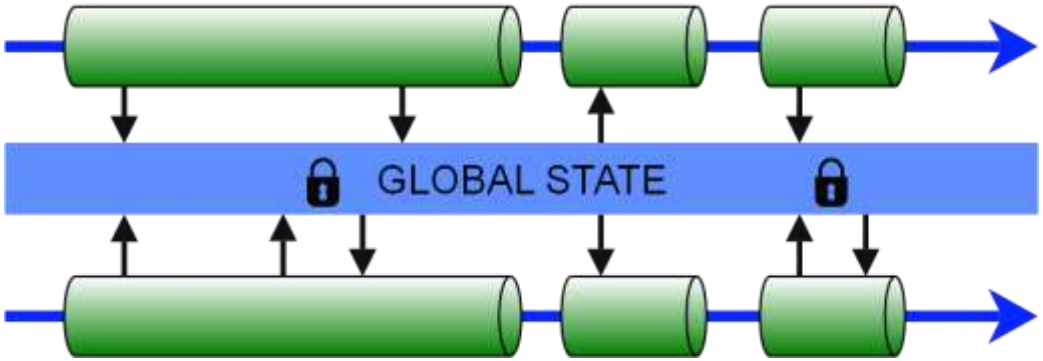
Actor Model

- Multiple threads of execution
- Each thread has its own execution state
- All thread interaction is via message passing
- No global state of any kind; no synchronization primitives



Parallel Random Access Model

- Multiple threads of execution
- All threads share same global state
- Coordination done via synchronization primitives (hopefully!)
- Does not disclude message passing or thread local state, but neither are required



Examples: POSIX threads, Java concurrency

Others

Many other models exist as well…

- Coroutines
- Petri-nets
- Process algebra
- Even….MapReduce

So, what model do we want?

Conceptual Model for Activities

July 30, 2015

Design Points

- Provide programming paradigm supporting multiple, concurrent "flows of control"
- Support both multi-threaded and single-threaded platforms
- Simplify the process of combining and composing independent haxelibs
- Define one common API, for all platforms, hiding platform specific details of “outer loop” management
- Do not dictate the programming paradigm/model; be flexible

Basically, make it possible to benefit from concurrency on platforms that provide it while writing only composable Haxe code...

Conceptual Model

Activity:

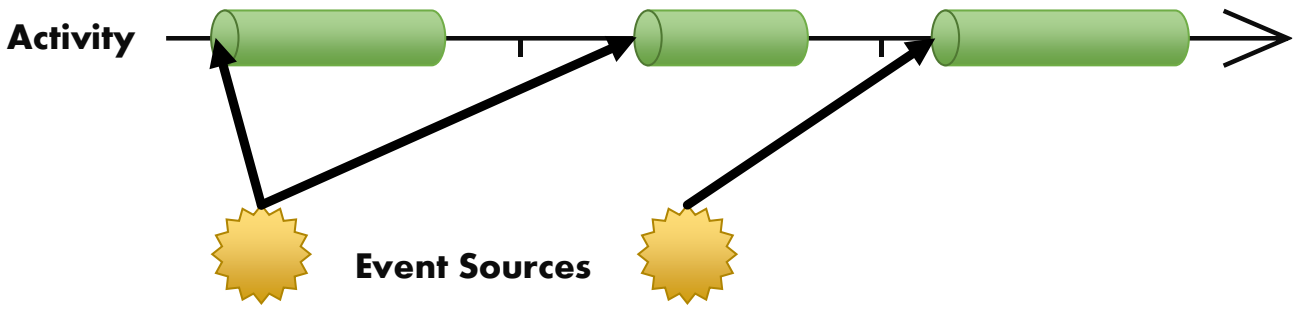
A flow of execution in a program that executes a sequence of nonblocking functions in response to stimulus events; a virtual thread implemented using an event handling paradigm.

Event source:

A source of stimulus events that can trigger execution of code within activities.

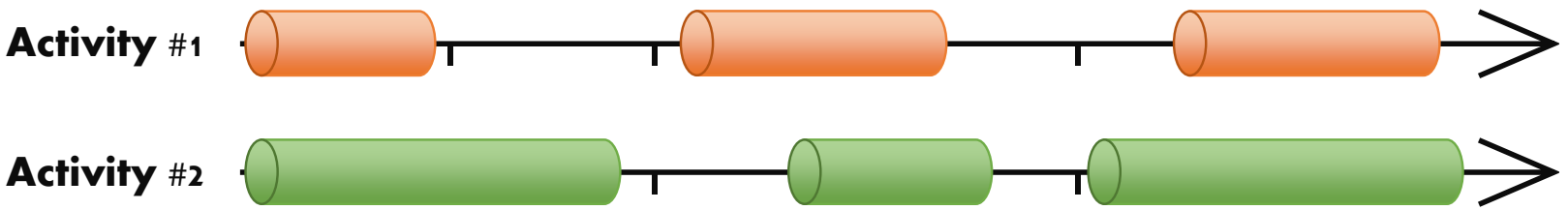
Activity

- Context of execution; logical sequence of instructions
- Unlike a thread not defined by a single function; instead defined as a sequence of callback functions, executed one at a time
- Actual function sequence executed is determined by which "stimulus events" received



Activity

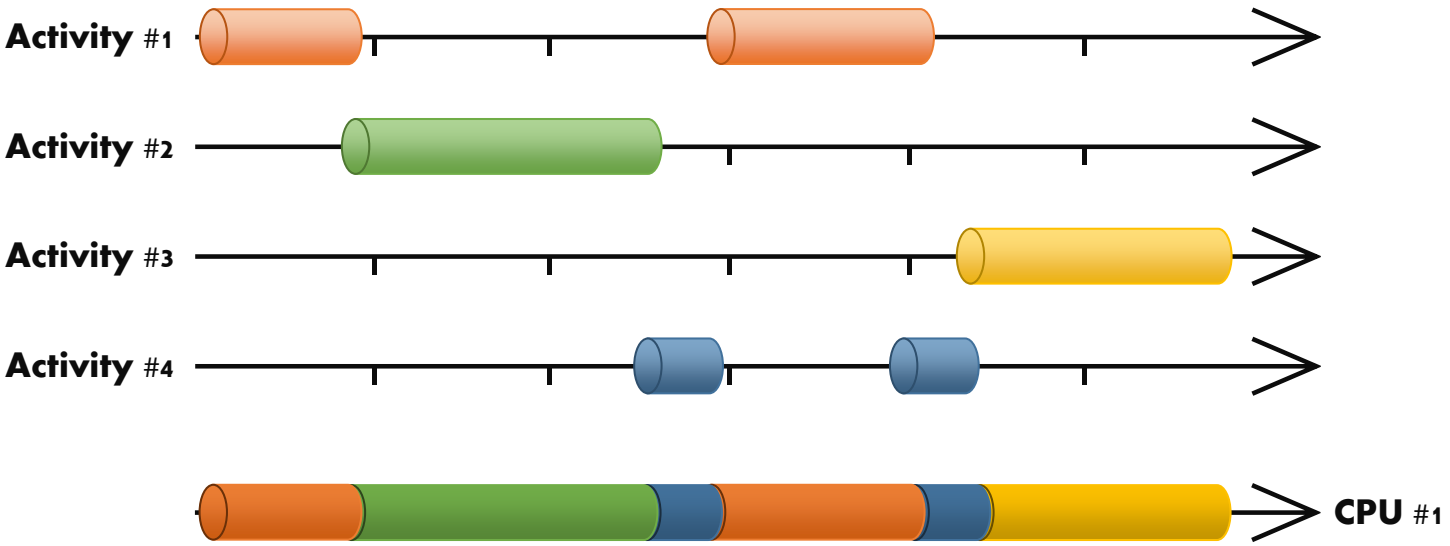
- May be executed simultaneously with or be interleaved sequentially with other activities, depending upon the platform
- Should run in small quanta; must not block when executing (to avoid deadlocking on platforms that do not support multi-threading)



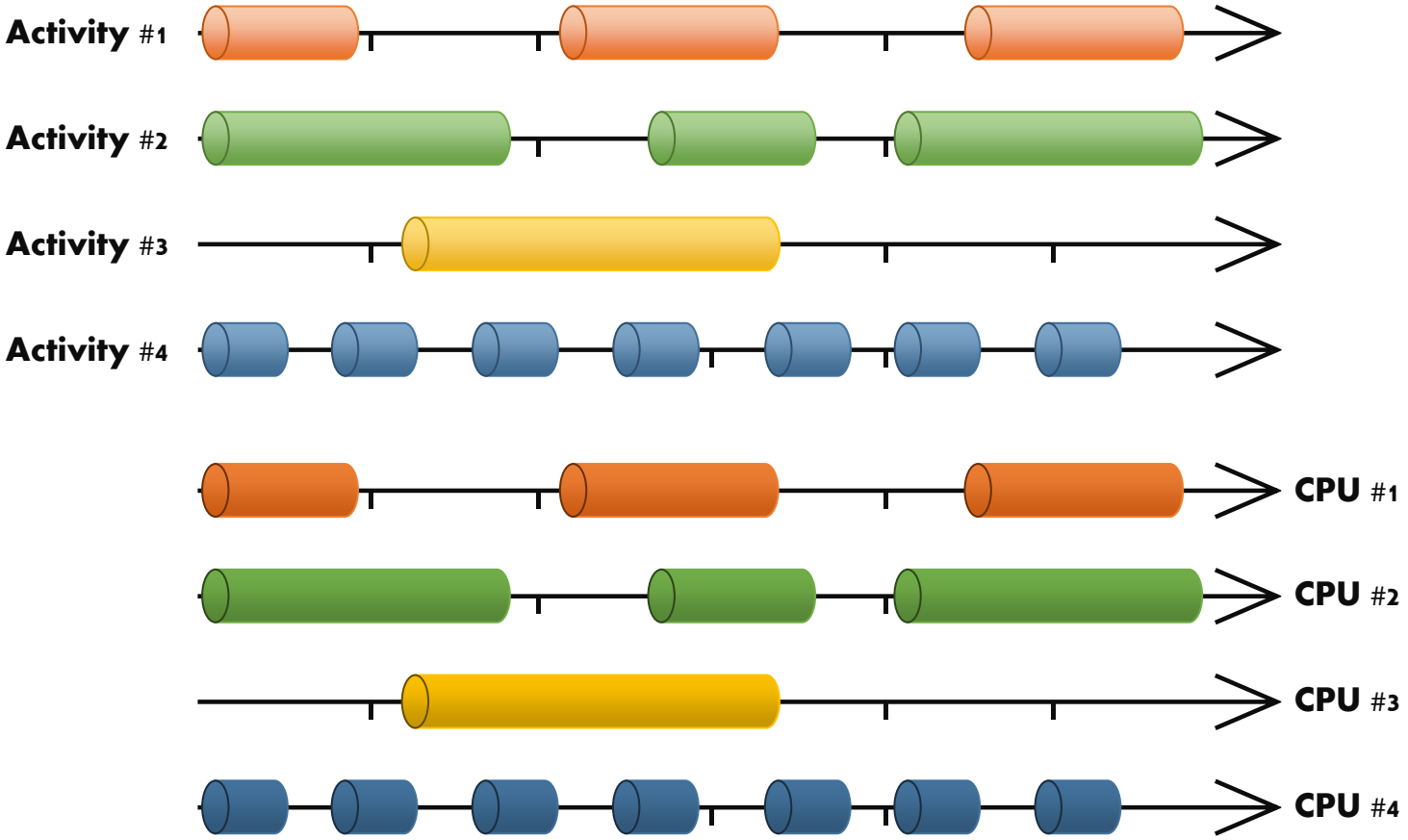
Event Source

- Generates events that stimulate activities
- Causes activities to run registered callbacks
- Can come from outside the program; generally operating system I/O
- Can come from other activities within the program
- Can come from within the activity itself
- Examples: system I/O (sockets, files), queues between activities, timers

Running in Single-Threaded Environments



Running in Multi-threaded Environments



Activity API

July 30, 2015

Activity API

- **Activity class**
  - Used to create new activities
  - Used to start up the activity system
  - Used to shut down the activity system

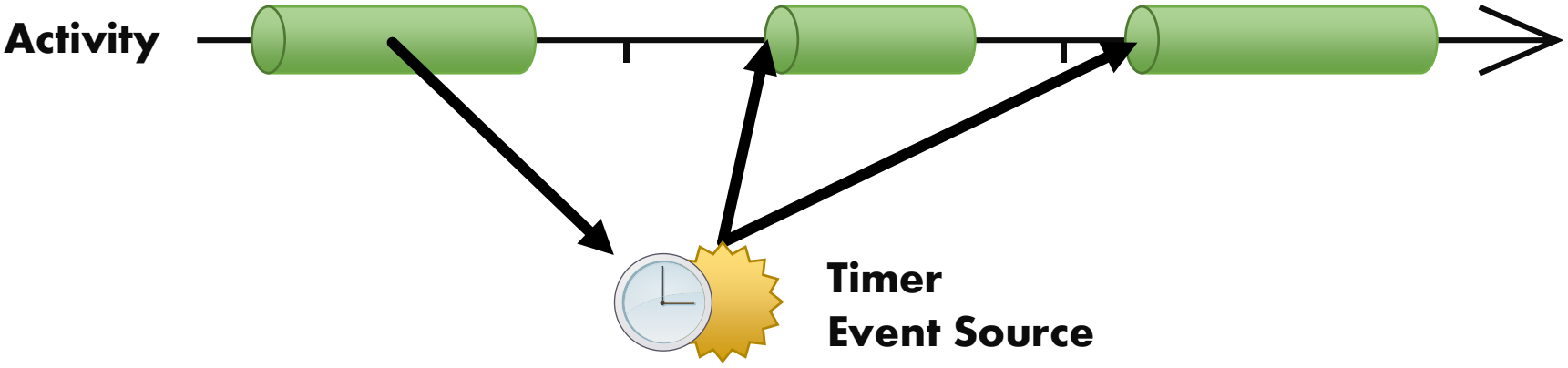
```
static function main()
{
   // create one (or more) Activities
   Activity.create(mainActivityFunction, "main");

   // run activities
   Activity.run();

   // ... typically, do nothing more; any code executed here cannot rely
   // on Activities having all completed already, having not even started
   // yet, or currently being in progress.
}
```

Event Sources: Triggering Yourself

- **Timer class**
  - Used to run code at a later time
  - Supports repeating



Event Sources: Triggering Yourself

- **Timer class**
  - Used to run code at a later time
  - Supports repeating

```
// create a new Timer and register it
var timer = new Timer();
timer.interval = 10;
timer.repeating = true;
timer.onTimeout = function (latency : Float) {
      trace("Repeating Timer event after "+latency+" seconds");
   };

// …or use static sugar function
Timer.once( function (latency : Float) {
      trace("One shot Timer event after "+latency+" seconds");
   }, 10 /* seconds */ );
```

Event Sources: Triggering Yourself

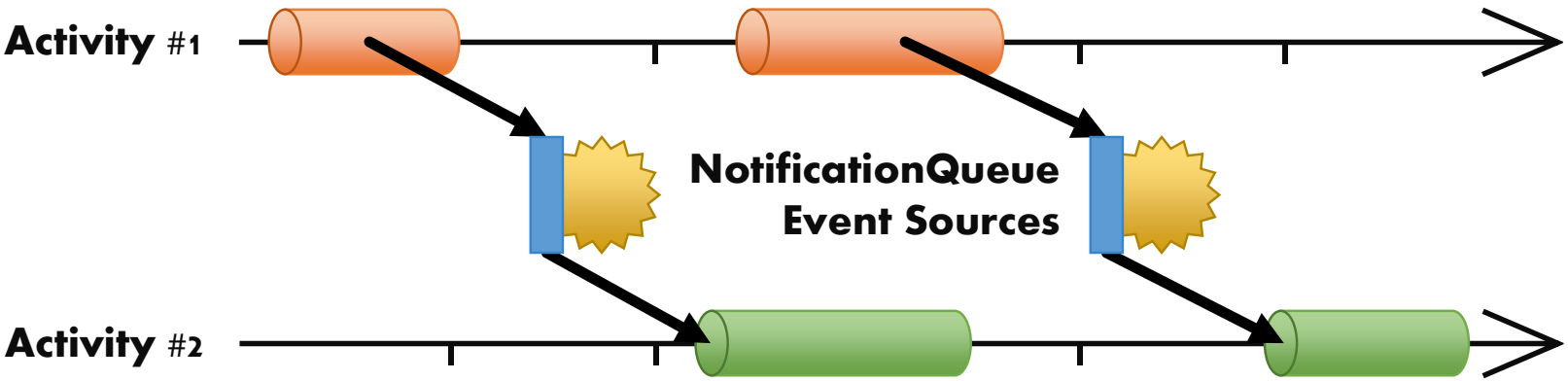
- **CallMe class**
  - Used to run code right after this event is handled
  - Actually a queue of triggers with fixed number of priorities (*immediately*, *soon*, *later*)

```
// create a new CallMe and register it
var callme = new CallMe(Later);
callme.onCalled = function () { trace("CallMe event"); };

// …or use static sugar functions
CallMe.later( function () { trace("Immediate CallMe"); } );
CallMe.soon( function () { trace("Immediate CallMe"); } );
CallMe.immediately( function () { trace("Immediate CallMe"); } );
```

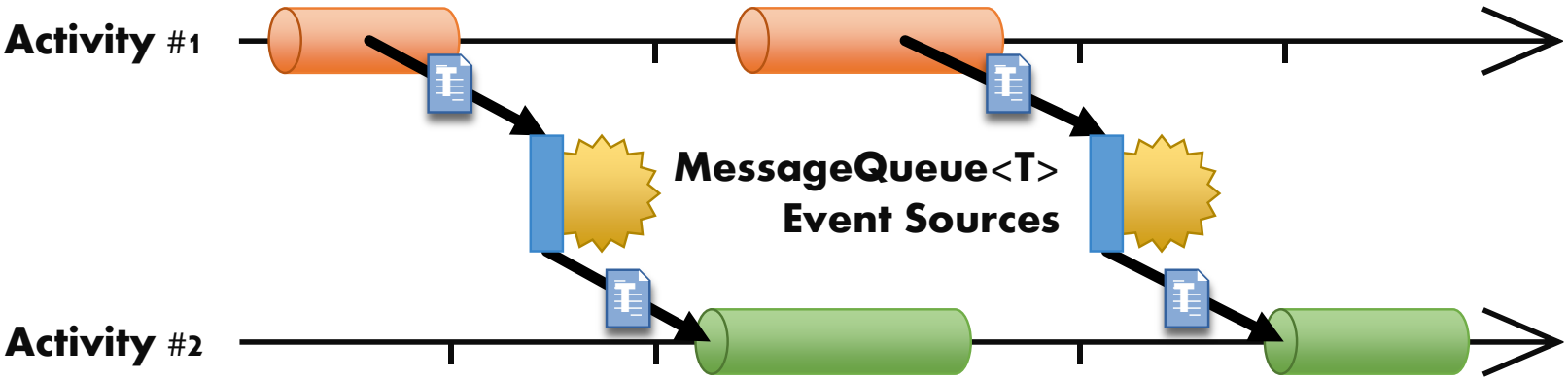
Event Sources: Triggering Other Activities

- **NotificationQueue**
  - Used by one activity to trigger execution of code in another activity, trigger is an impulse event (no message)
  - Can have multiple “senders” and multiple “receivers”



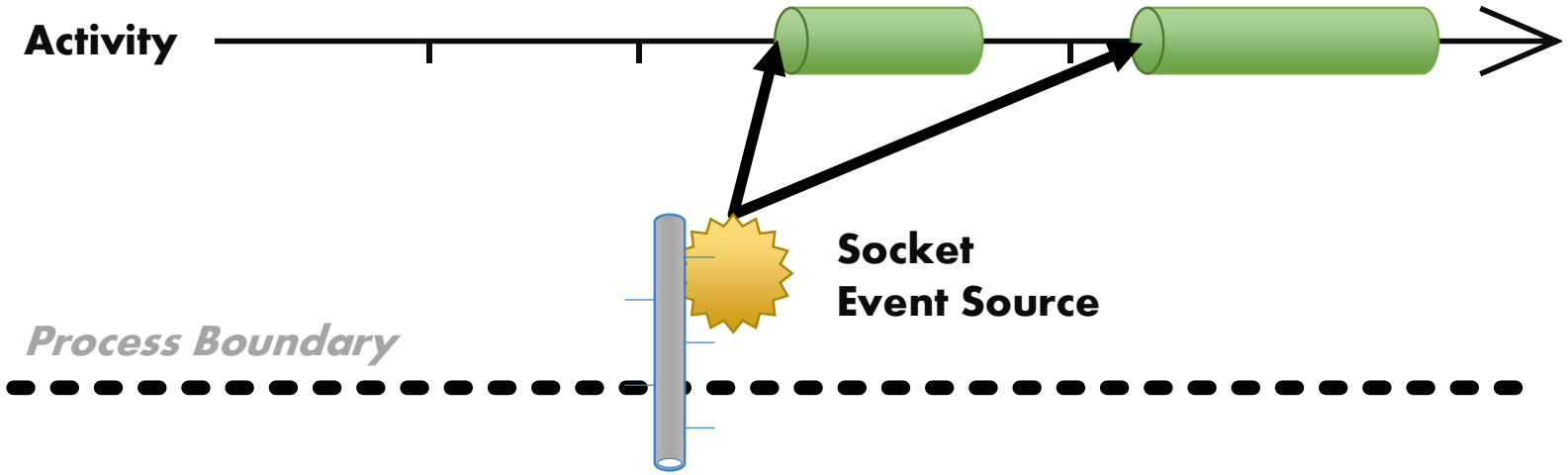
Event Sources: Triggering Other Activities

- **MessageQueue<T>**
  - Used by one activity to trigger execution of code in another activity, trigger includes message
  - Can have multiple “senders” and multiple “receivers”



Event Sources: System I/O

- **Socket**
  - Used to run code when a TCP socket is ready for reading or writing
  - Only on: cpp, cs, java, neko, php, python, flash

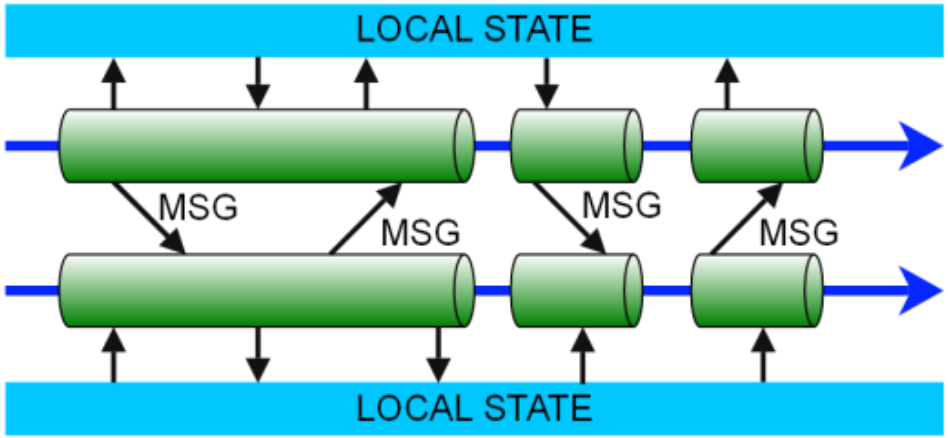


Event Sources: System I/O

- **UdpSocket**
  - Used to run code when a UDP socket is ready for reading or writing
  - Only on: sys, neko
- **WebSocket**
  - Used to run code when a WebSocket is opened, receives a message or is closed

Activity Haxelib: Actor Model

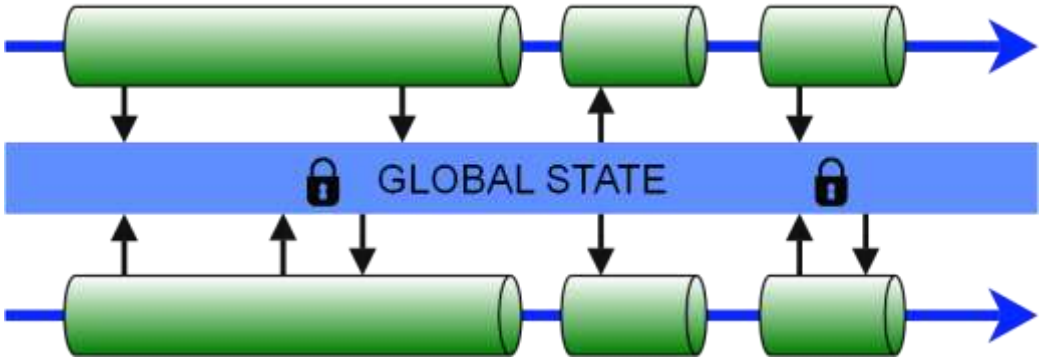
Using these concepts (activity and event sources), we can build Haxe libraries and applications using the actor model of concurrency.



But what if we don't want to? Can we support the Parallel Random Access Model (PRAM) as well?

Supporting Global State w/Explicit Synchronizing

- **Mutex**
  - Ensures that only one activity at a time can enter specific regions of code
  - Allows support for parallel random access model
  - Must not remain held when handling of any event completes



Helper Classes

- **WorkerPool<T,U>**
  - Manages a fixed size pool of activities that all do the same kind of work
  - Submit work items and they are farmed out to the workers
  - Processed results can be delivered to *completion* function closure

Wrap Up

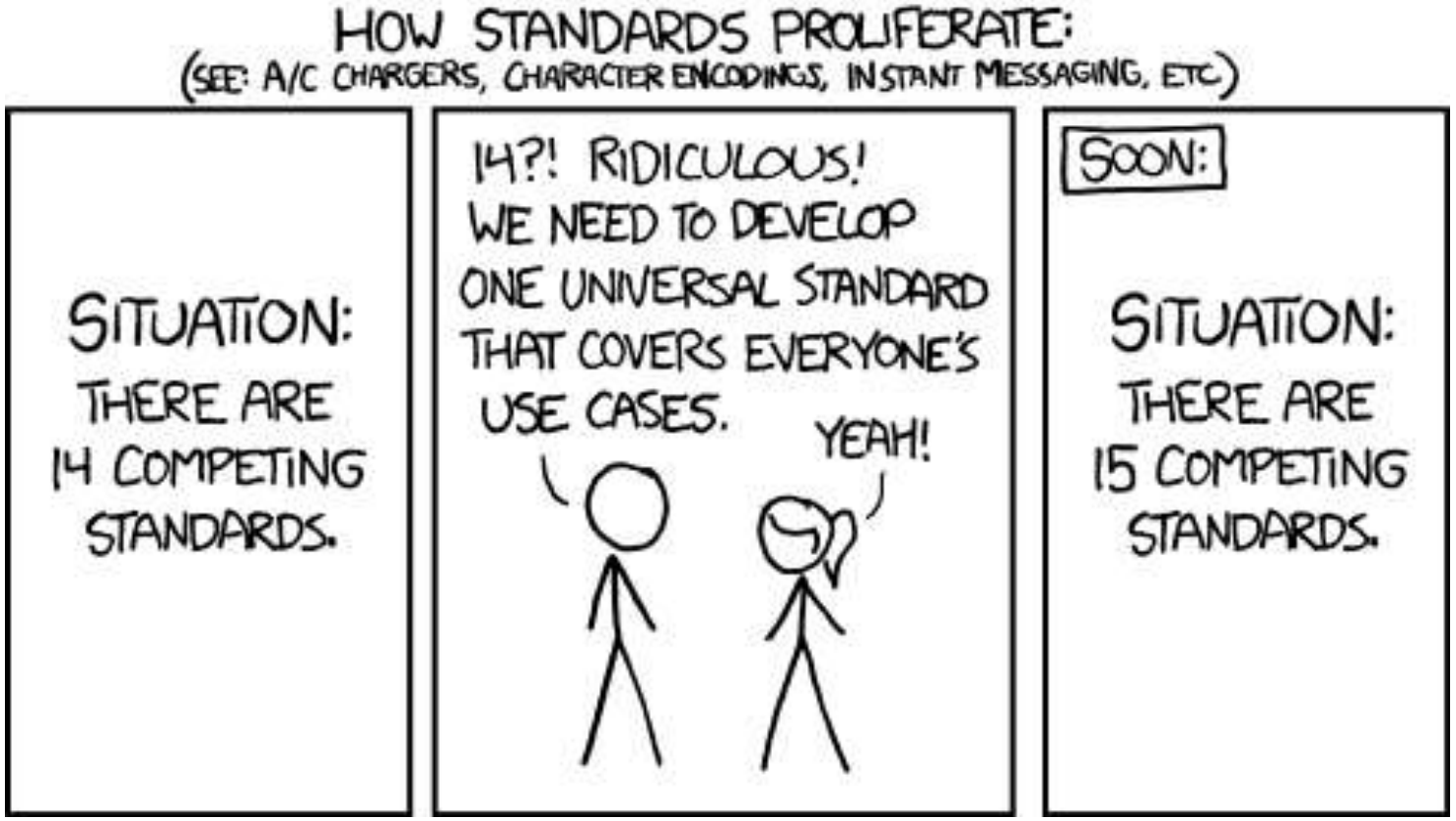
July 30, 2015

Who Might Find This Useful?

- UI toolkits: OpenFL, Lime
  - Other OpenFL and Lime based frameworks could benefit
- Multi-threaded helper libraries
  - Networking libraries
  - Video streaming libraries
  - Physics engines
  - Tesselation engines
  - Other computation heavy libraries
- End user programs

Future Work

- Other useful, common helper classes
- Native implementation for some platforms
  - Should be implementable without API change!
- Limited concurrency on Javascript
  - Javascript worker thread support
- Test port OpenFL/Lime to activity framework
- Other ideas?



https://xkcd.com/927

Credits

- Bryan Ischo
  - Main implementor and API designer of the activity haxelib
- Yannick Dominguez
  - Built activity test suite

https://github.com/TiVo/activity


TiVo