state machine (big and increasing states is problematic)

time is critical (state changing as a time function)

no quick way to determine whether a function/object is modular

event/listener problems:

Unpredictable order—In a complex network of listeners, the order in which events

are received can depend on the order in which you registered the listeners,

which isn’t helpful.  FRP makes the order in which events are processed not mat-

ter by making it completely nondetectable.

Missed first event—It can be difficult to guarantee that you’ve registered your lis-

teners before you send the first event.  FRP is transactional, so it’s possible to

provide this guarantee.

Messy state—Callbacks push your code into a traditional state-machine style,

which gets messy fast.  FRP brings order.

Threading issues—Attempting to make listeners thread-safe can lead to dead-

locks, and it can be difficult to guarantee that no more callbacks will be

received after deregistering a listener.  FRP eliminates these issues.

Leaking callbacks—If you forget to deregister your listener, your program will

leak memory. Listeners reverse the natural data dependency but don’t reverse

the keep-alive dependency as you’d like them to.  FRP does this.

Accidental recursion—The order in which you update local state and notify listen-

ers can be critical, and it’s easy to make mistakes.  FRP eliminates this issue.

Rx(not exactly same as FRP):

asynchronize operation modularly:

example buffer(<http://www.introtorx.com/Content/v1.0.10621.0/13_TimeShiftedSequences.html#Buffer>)

IObservable<IList<T>> bufferedSequence;

bufferedSequence = mySequence.Buffer(4);

//or

bufferedSequence = mySequence.Buffer(TimeSpan.FromSeconds(1))

In this example below, we create a sequence that produces the first ten values one second apart, then a further hundred values within another second. We buffer by a maximum period of three seconds and a maximum batch size of fifteen values.

var idealBatchSize = 15;

var maxTimeDelay = TimeSpan.FromSeconds(3);

var source = Observable.Interval(TimeSpan.FromSeconds(1)).Take(10)

.Concat(Observable.Interval(TimeSpan.FromSeconds(0.01)).Take(100));

source.Buffer(maxTimeDelay, idealBatchSize)

.Subscribe(

buffer => Console.WriteLine("Buffer of {1} @ {0}", DateTime.Now, buffer.Count),

() => Console.WriteLine("Completed"));

Output:

Buffer of 3 @ 01/01/2012 12:00:03

Buffer of 3 @ 01/01/2012 12:00:06

Buffer of 3 @ 01/01/2012 12:00:09

Buffer of 15 @ 01/01/2012 12:00:10

Buffer of 15 @ 01/01/2012 12:00:10

Buffer of 15 @ 01/01/2012 12:00:10

Buffer of 15 @ 01/01/2012 12:00:11

Buffer of 15 @ 01/01/2012 12:00:11

Buffer of 15 @ 01/01/2012 12:00:11

Buffer of 11 @ 01/01/2012 12:00:11

Note the variations in time and buffer size.

state dependent explicite

expression problems: exntend a data type and operations on that data at the same time