Multithreading in UI application

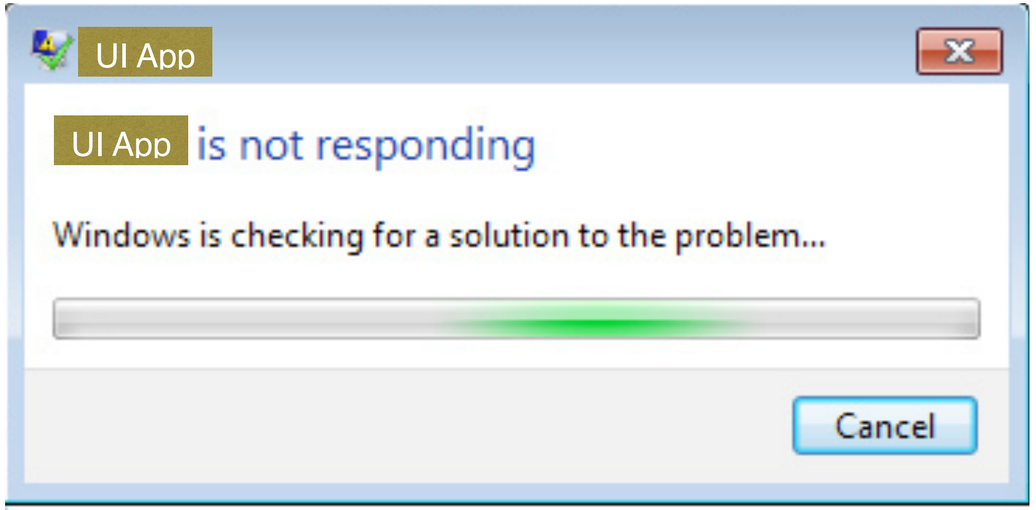
When some threads are more equal than others

Dmitry Ivanov, Hydra 2020

korifey_ad

UI/UX REQUIREMENTS

Responsiveness

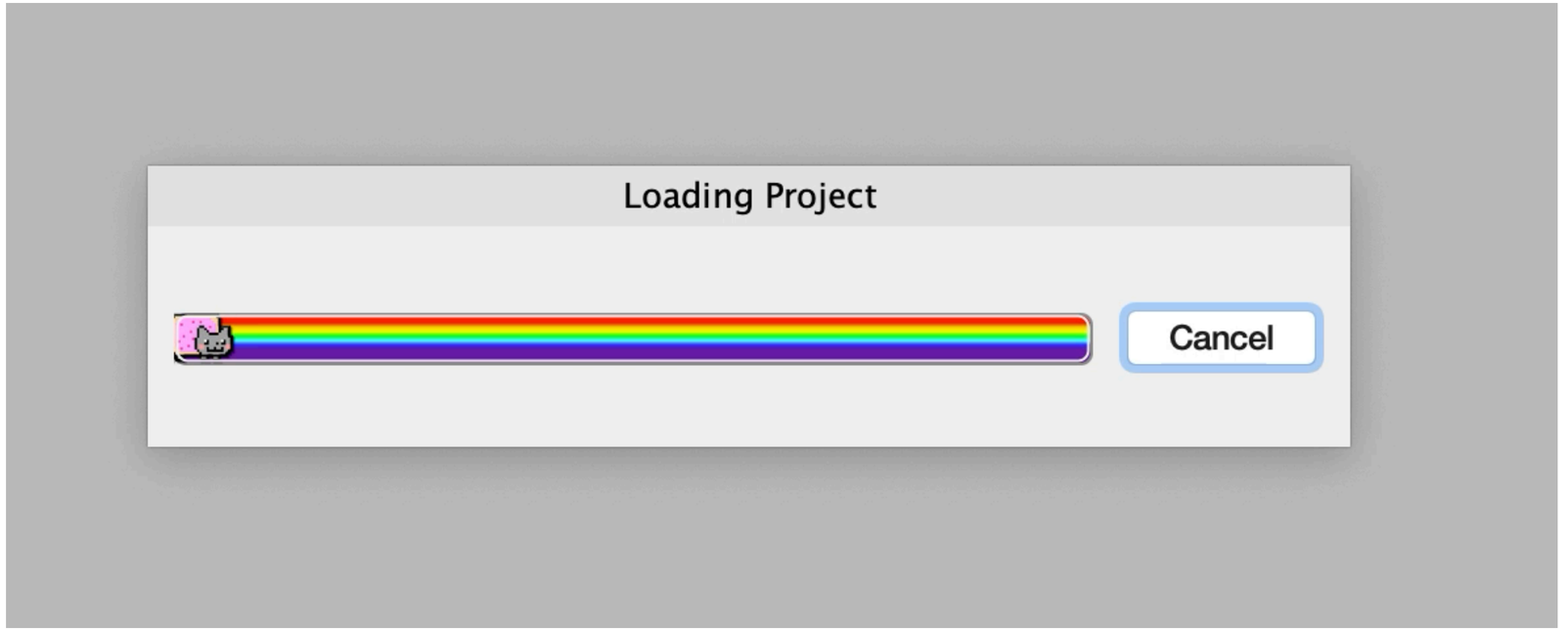


Three important limits

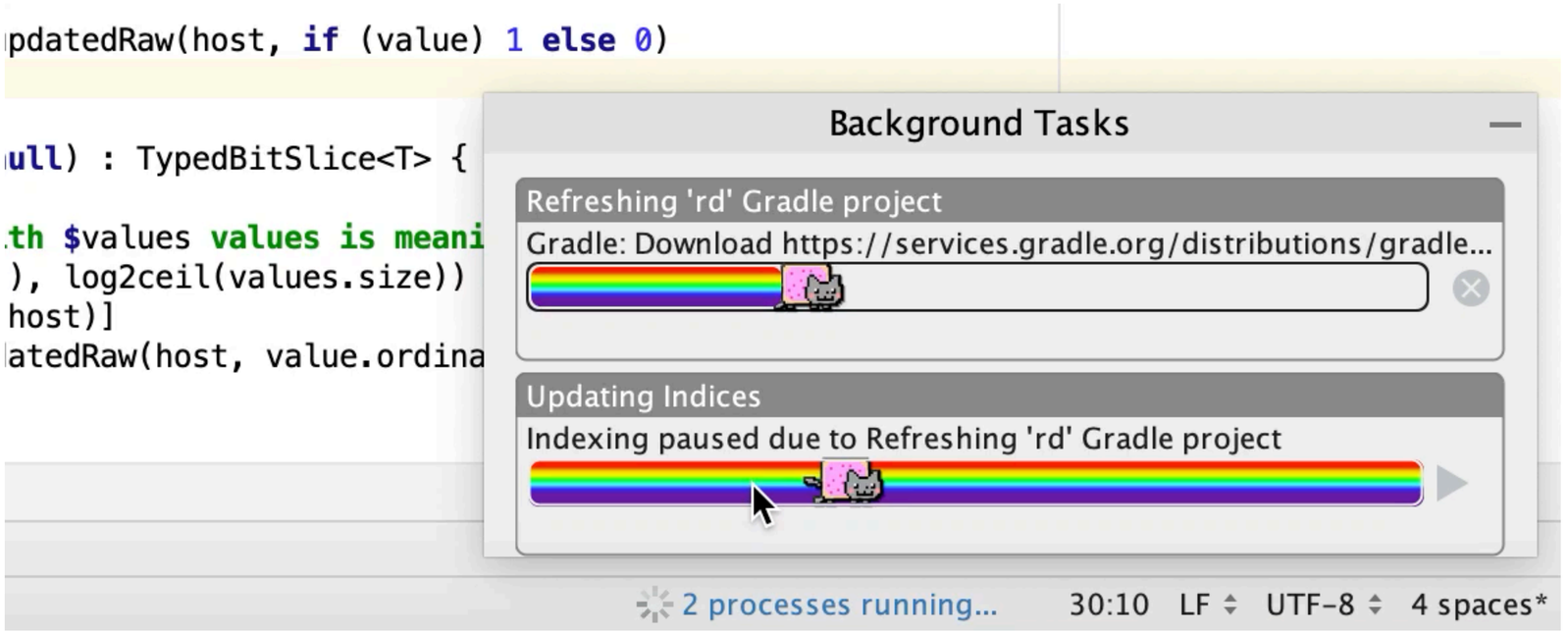
- **0.1** sec — feels like an instant reaction
- **1** sec — flow of thought stays uninterrupted
- **10** sec — keeping the attention focused on the task

[Robert Miller, Response time in man-computer conversational transactions, 1968]

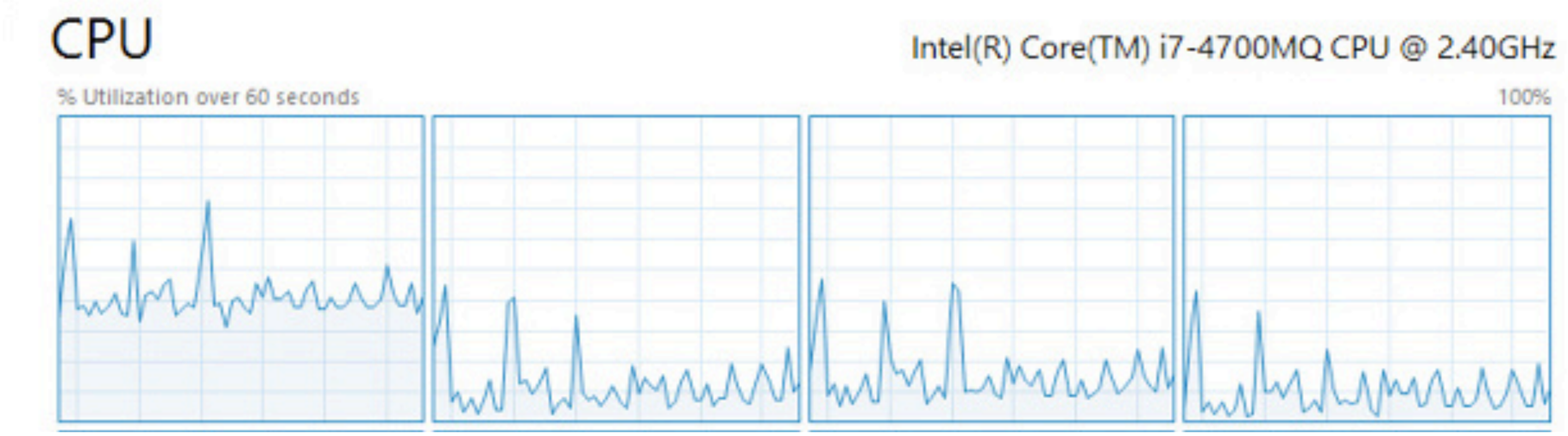
Progress and cancellation



Non-modal interaction



Resource utilisation



UI Thread

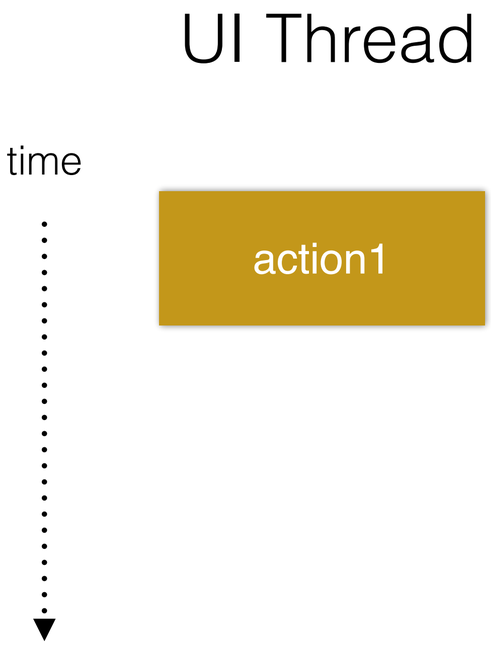
Message queue

UI Thread

time

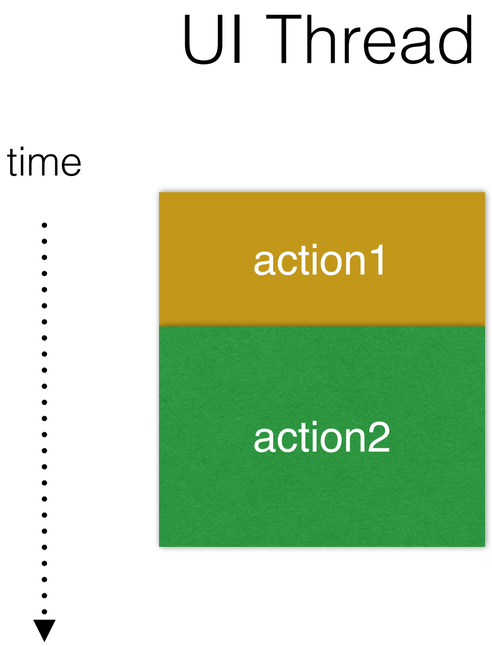
```
while (true) {

    //actions - ConcurrentQueue
    var action = actions.poll();

    if (action != null)
        action();

}
```

Message queue



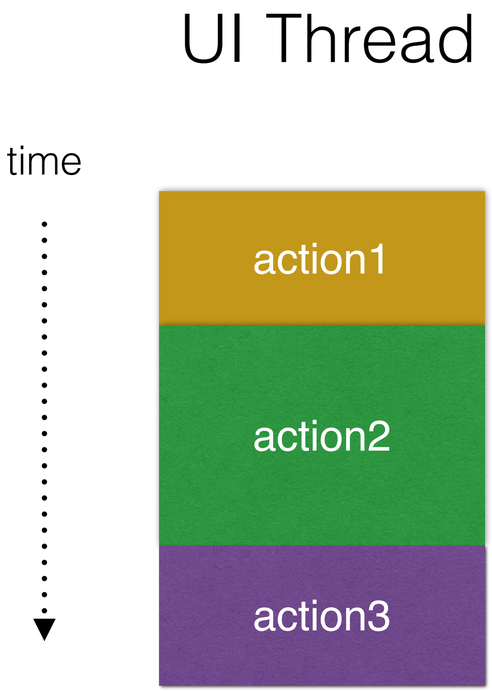

```
while (true) {

    //actions - ConcurrentQueue
    var action = actions.poll();

    if (action != null)
        action(); //action1

}
```

Message queue




```
while (true) {

    //actions - ConcurrentQueue
    var action = actions.poll();

    if (action != null)
        action(); //action2

}
```

Message queue

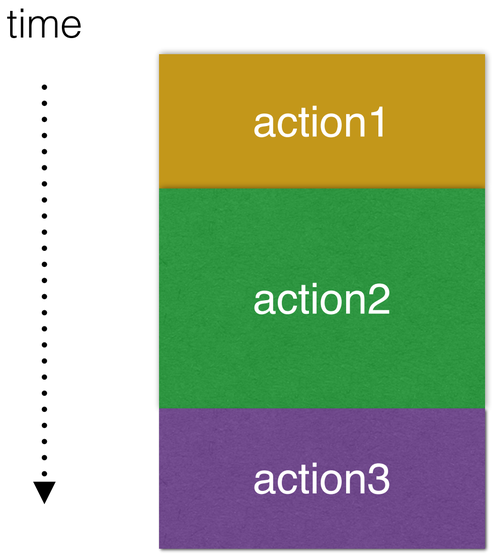



```
while (true) {

    //actions - ConcurrentQueue
    var action = actions.poll();

    if (action != null)
        action(); //action3

}
```

...

Message queue

UI Thread

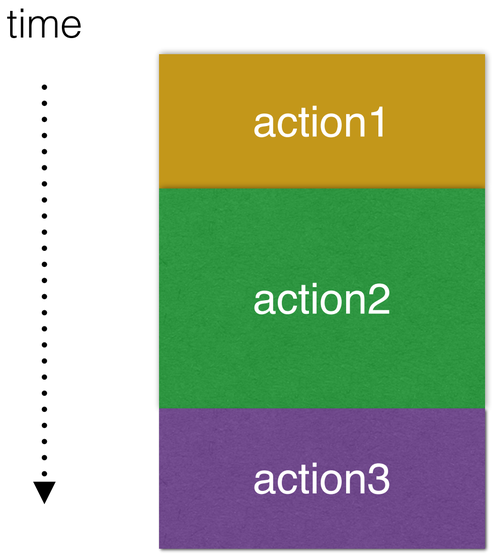


...

```
while (true) {

    //actions - ConcurrentQueue
    var action = actions.poll();

    if (action != null)
        action();
    //else
}
```

Message queue

UI Thread

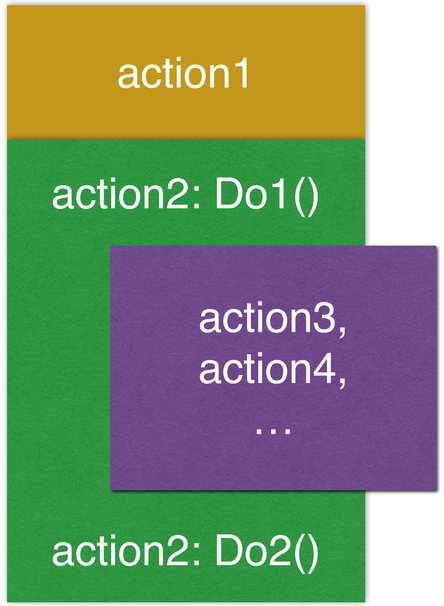


...

```
while (!IsShutdown) {

    //actions - ConcurrentQueue
    var action = actions.poll();

    if (action != null)
        action();
    //else
}
```

Pump messages

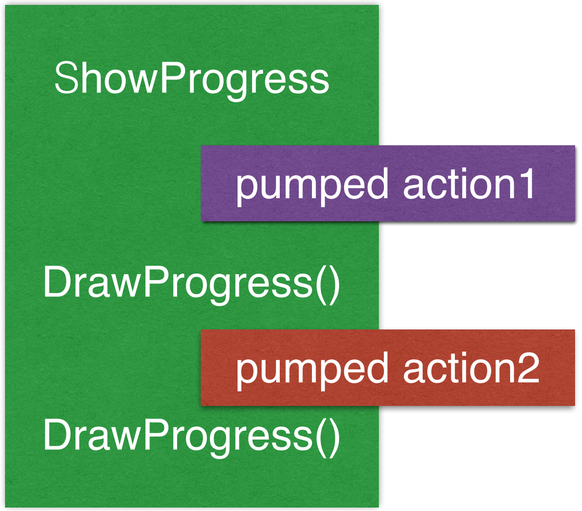
UI Thread




```
action2() {

  DoSomething1();

  Dispatcher.PumpMessages(); //Yield

  DoSomething2();
}
```

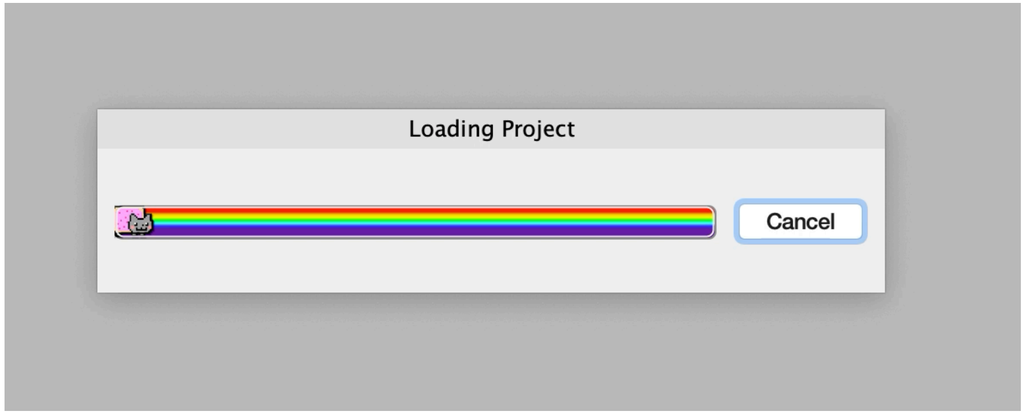
Reentrancy in progress

UI Thread




```
ShowProgress() {
  while (!isCanceled && !finishedProcessing) {

    Dispatcher.PumpMessages();

    if (IsProgressChanged())
      DrawProgress();
  }
}
```

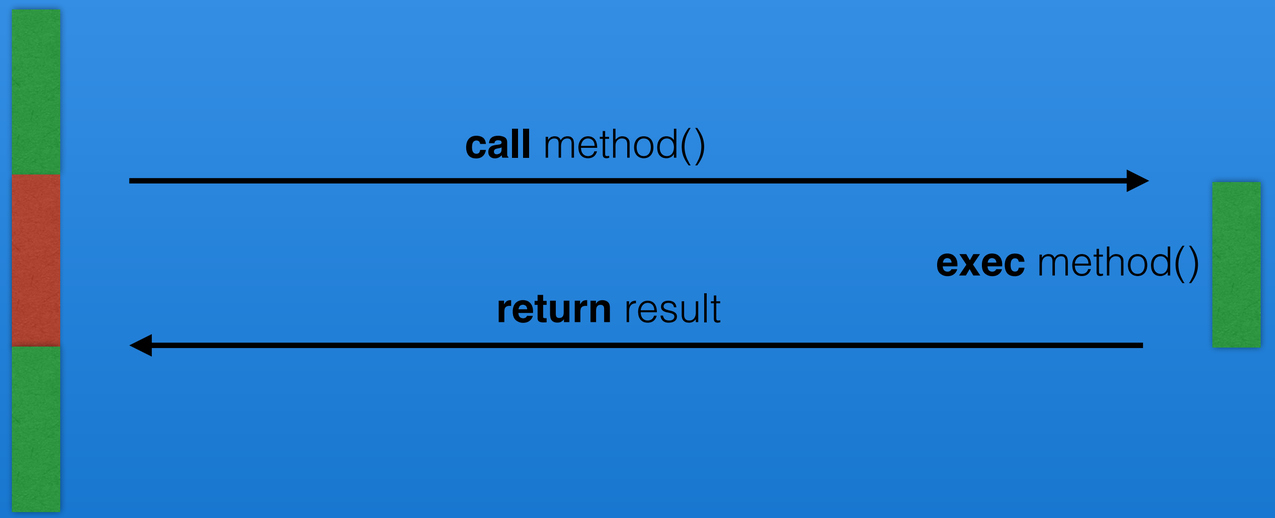
...



.NET Approach: COM

UI Thread (STA)

Another STA Thread

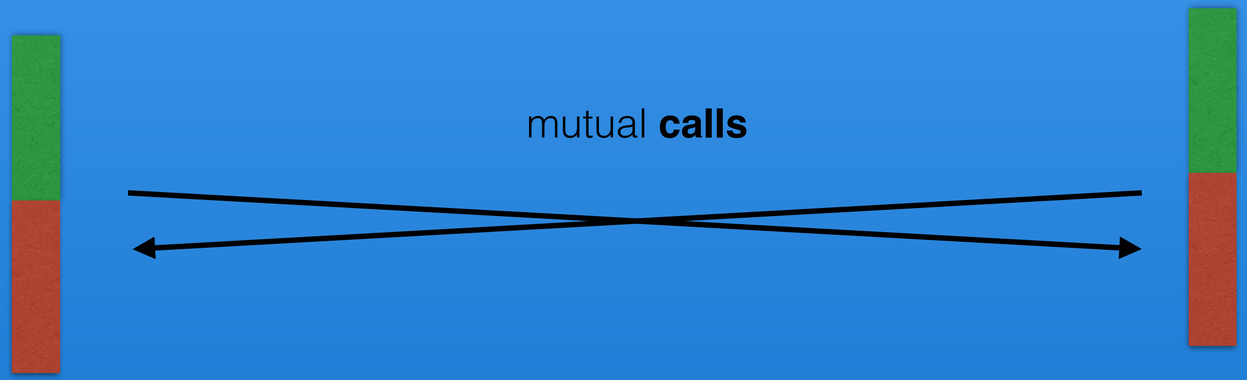


...

.NET Approach: Problem

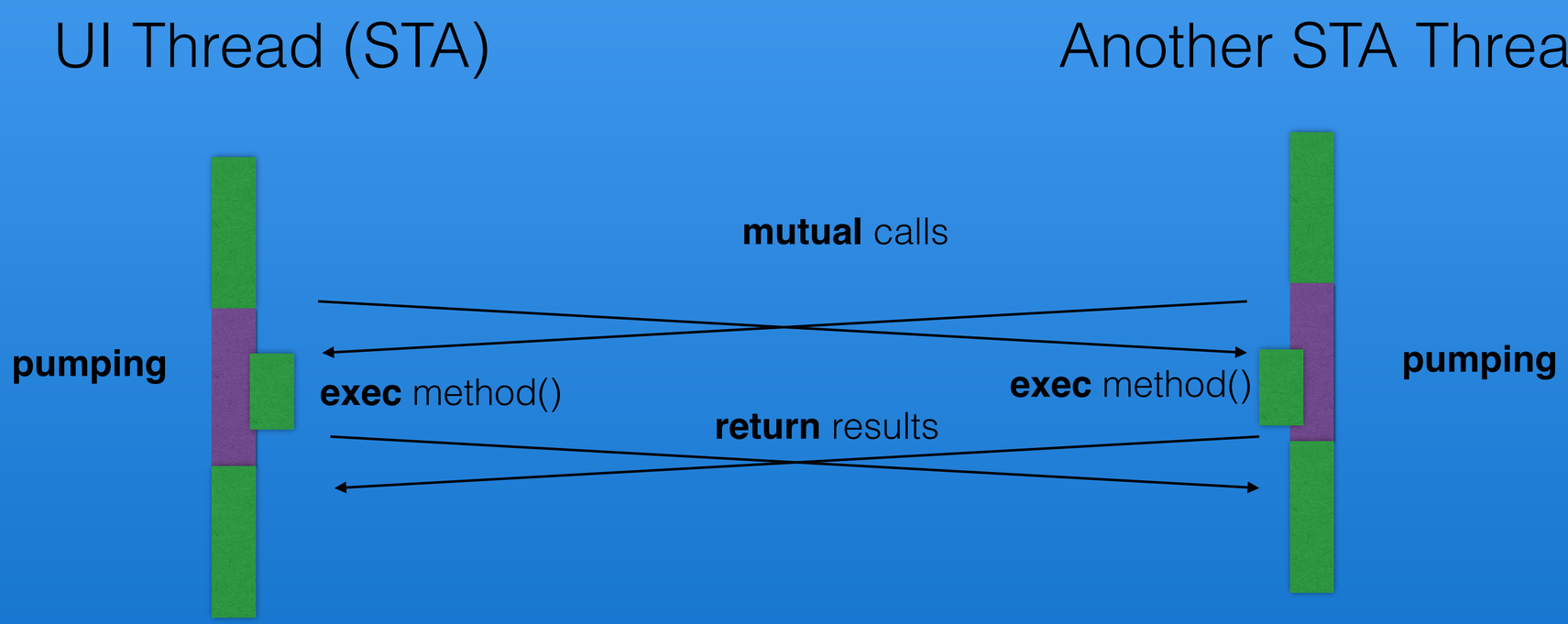
UI Thread (STA)

Another STA Thread



…

.NET Approach: Solution



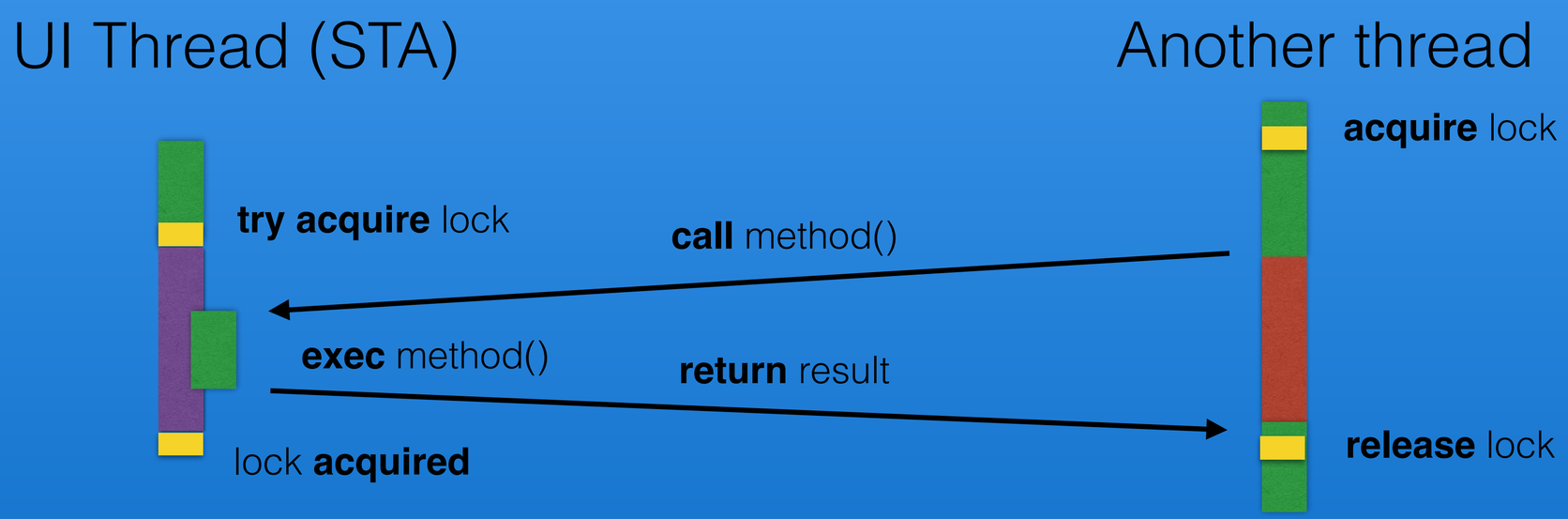
...

.NET Approach: Locks

Locks + invoke messages can easily lead to deadlocks

.NET Approach: Locks

Locks should pump messages !




```
Monitor.Enter()
        WaitHandle.WaitOne()
                SynchronizationContext.Wait()
```

PUMP

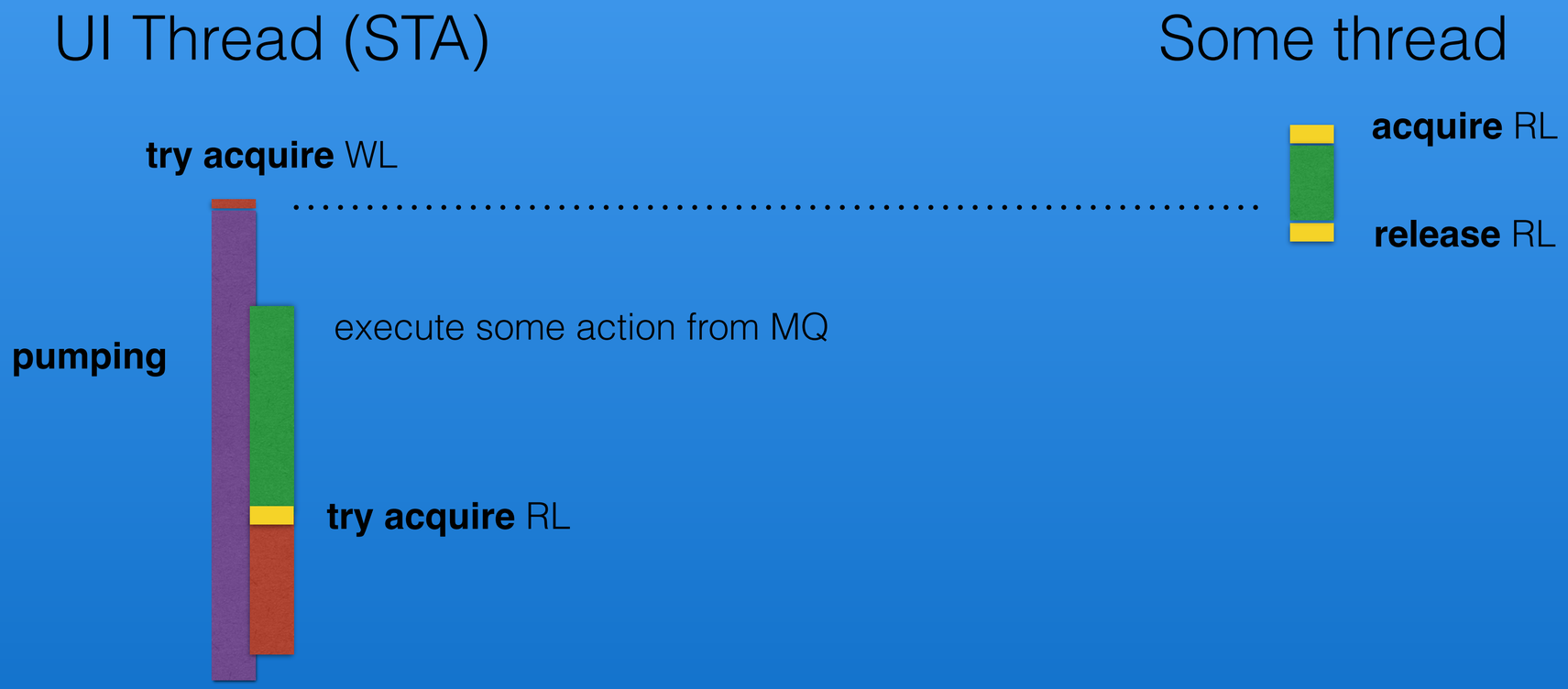
.NET: Single thread deadlock

```
100.00%  AddLocked • 6,073 ms • JetBrains.Util.CollectionUtil.AddLocked(ICollection, Lifetime, Func, T)
  100.00%  AddBracket • 6,073 ms • JetBrains.Lifetimes.Lifetime.AddBracket(Action, Action)
    100.00%  Bracket • 6,073 ms • JetBrains.Lifetimes.Lifetime.Bracket(Action, Action)
      100.00%  Bracket • 6,073 ms • JetBrains.Lifetimes.LifetimeDefinition.Bracket(Action, Action)
        100.00%  <AddLocked>b__0 • 6,073 ms • JetBrains.Util.CollectionUtil+<>c__DisplayClass159_0`1.<AddLocked>b__0
          100.00%  <ApplyTransformation>b__1 • 6,073 ms • JetBrains.Util.Logging.LogManager+<>c__DisplayClass29_0.<ApplyTransformation>b__1
            100.00%  UsingWriteLock • 6,073 ms • JetBrains.Util.Concurrency.ReaderWriterLockSlimEx.UsingWriteLock(ReaderWriterLockSlim)
              100.00%  EnterWriteLock • 6,073 ms • System.Threading.ReaderWriterLockSlim.EnterWriteLock
                100.00%  TryEnterWriteLock • 6,073 ms • System.Threading.ReaderWriterLockSlim.TryEnterWriteLock(TimeoutTracker)
                  100.00%  TryEnterWriteLockCore • 6,073 ms • System.Threading.ReaderWriterLockSlim.TryEnterWriteLockCore(TimeoutTracker)
                    100.00%  WaitOnEvent • 6,073 ms • System.Threading.ReaderWriterLockSlim.WaitOnEvent(EventWaitHandle, ref UInt32, TimeoutTracker, EnterLockType)
                      100.00%  WaitOne • 6,073 ms • System.Threading.WaitHandle.WaitOne(Int32)
                        100.00%  WaitOne • 6,073 ms • System.Threading.WaitHandle.WaitOne(Int32, Boolean)
                          100.00%  InternalWaitOne • 6,073 ms • System.Threading.WaitHandle.InternalWaitOne(SafeHandle, Int64, Boolean, Boolean)
                            100.00%  [Native code] • 6,073 ms
                              100.00%  InvokeWaitMethodHelper • 6,073 ms • System.Threading.SynchronizationContext.InvokeWaitMethodHelper(SynchronizationContext, IntPtr[], Boolean, Int32)
                                100.00%  OnApplicationActiveStateChanged • 6,073 ms • JetBrains.Application.Interop.NativeHook.NativeWindowsHookManager+ApplicationActiveStateChangeSink.OnApplicationActiveStateChanged(Int32)
                                  100.00%  set_Value • 6,073 ms • JetBrains.DataFlow.Property`1.set_Value(TValue)
                                    100.00%  SetValue • 6,073 ms • JetBrains.DataFlow.Property`1.SetValue(TValue, Object)
                                      100.00%  FireChange • 6,073 ms • JetBrains.DataFlow.Property`1.FireChange(TValue, TValue, Object)
                                        100.00%  Fire • 6,073 ms • JetBrains.DataFlow.Signal`1.Fire(TValue, Object)
                                          100.00%  NotifySinks • 6,073 ms • JetBrains.DataFlow.Signal`1.NotifySinks(TValue)
                                            100.00%  <FlowIntoReadonly>b__0 • 6,073 ms • JetBrains.DataFlow.IPropertyEx+<>c__DisplayClass13_0`1.<FlowIntoReadonly>b__0(PropertyChangedEventArgs)
                                              100.00%  SetValue • 6,073 ms • JetBrains.DataFlow.Property`1.SetValue(TValue, Object)
                                                100.00%  FireChange • 6,073 ms • JetBrains.DataFlow.Property`1.FireChange(TValue, TValue, Object)
                                                  100.00%  Fire • 6,073 ms • JetBrains.DataFlow.Signal`1.Fire(TValue, Object)
                                                    100.00%  NotifySinks • 6,073 ms • JetBrains.DataFlow.Signal`1.NotifySinks(TValue)
                                                      100.00%  <FlowInto>b__0 • 6,073 ms • JetBrains.DataFlow.IPropertyEx+<>c__DisplayClass8_0`1.<FlowInto>b__0(PropertyChangedEventArgs)
                                                        100.00%  SetValue • 6,073 ms • JetBrains.DataFlow.Property`1.SetValue(TValue, Object)
                                                          100.00%  IsEnabled • 6,073 ms • JetBrains.Util.Logging.ConfiguredLogger.IsEnabled(LoggingLevel)
                                                            100.00%  UsingReadLock • 6,073 ms • JetBrains.Util.Concurrency.ReaderWriterLockSlimEx.UsingReadLock(ReaderWriterLockSlim)
                                                              100.00%  EnterReadLock • 6,073 ms • System.Threading.ReaderWriterLockSlim.EnterReadLock
                                                                0.13%  <Initialize>g__Hook|3 • 8 ms • JetBrains.Platform.VisualStudio.SinceVs10.Shell.VsMainWindowSinceVs10+<>c__DisplayClass9_0.<Initialize>g__Hook|3(IntPtr, Int32, IntPtr, IntPtr, ref Boolean)
```

pump messages

DEADLOCK

.NET: Single thread deadlock

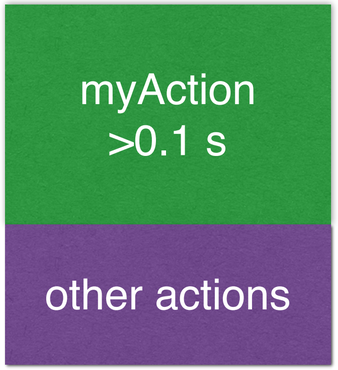


ReSharper & Rider experience

- Very hard to develop assuming that reentancy could be everywhere
- Special concepts like ReentrancyGuard are invented
- Hardest bugs are related to reentrancy + multithreading
- Several man-months are spent fixing these bugs

Long operations

UI Thread

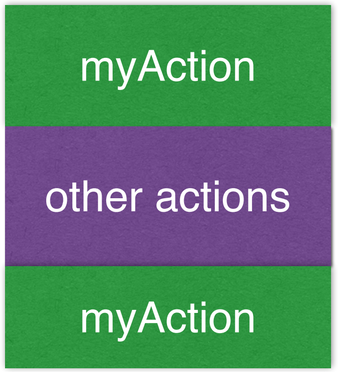


...

```
myAction() {

  for (item in itemsToProcess) {
    Process(item)
    itemsToProcess.Remove(i);

  }
}
```

Long operations

UI Thread



...

```
myAction() {
  var startTime = currentTime();

  for (item in itemsToProcess) {
    Process(item)
    itemsToProcess.Remove(i);

    if (currentTime() - startTime > 100ms) {
      Dispatcher.Dispatch(action2);
      return;
    }
  }
}
```

Asynchronous world

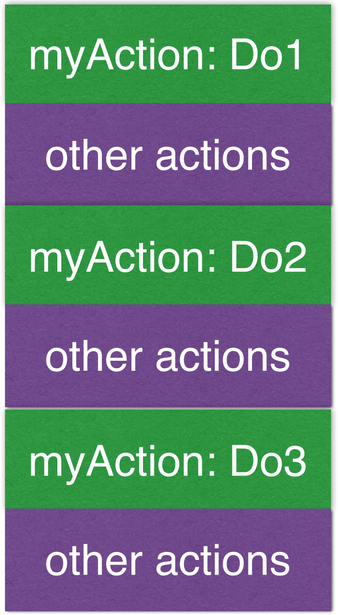
UI Thread

...

```
myAction() {
   DoSomething1();

   DoSomething2();

   DoSomething3();
}
```

Asynchronous world

UI Thread




```
myAction() {

  DoSomething1();

  Dispatcher.Dispatch(() => {
    DoSomething2();

    Dispatcher.Dispatch(() => {
      DoSomething3();
    });
  }
}
```

Asynchronous world

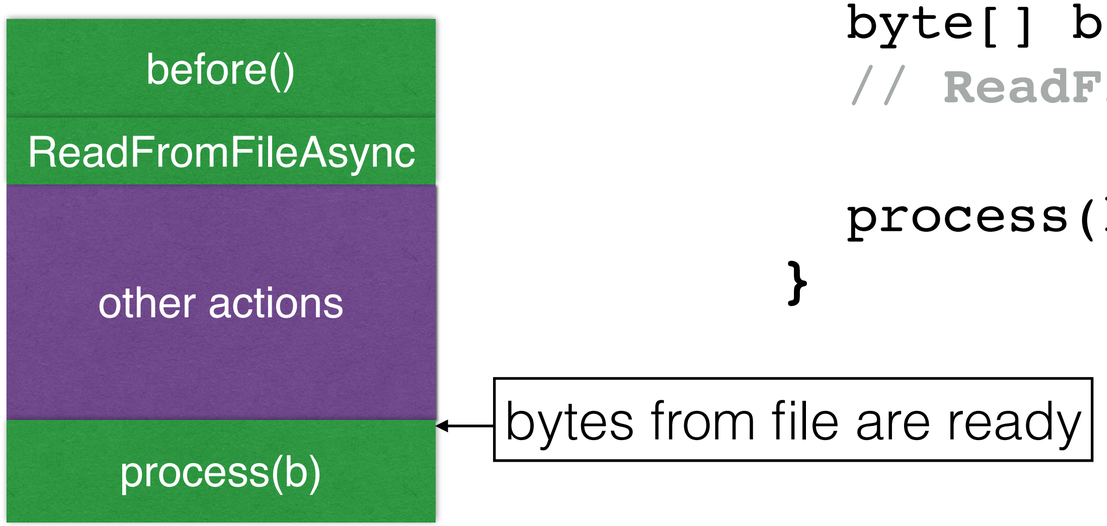
UI Thread


```
async myAction() {

  DoSomething1();

  await Thread.Yield();

  DoSomething2();

  await Thread.Yield();

  DoSomething3();

}
```

Asynchronous IO

```
async myAction() {

  before();

  byte[] b = await ReadFromFileAsync();
  // ReadFromFileAsync returns Future<byte[]>

  process(b);
}
```

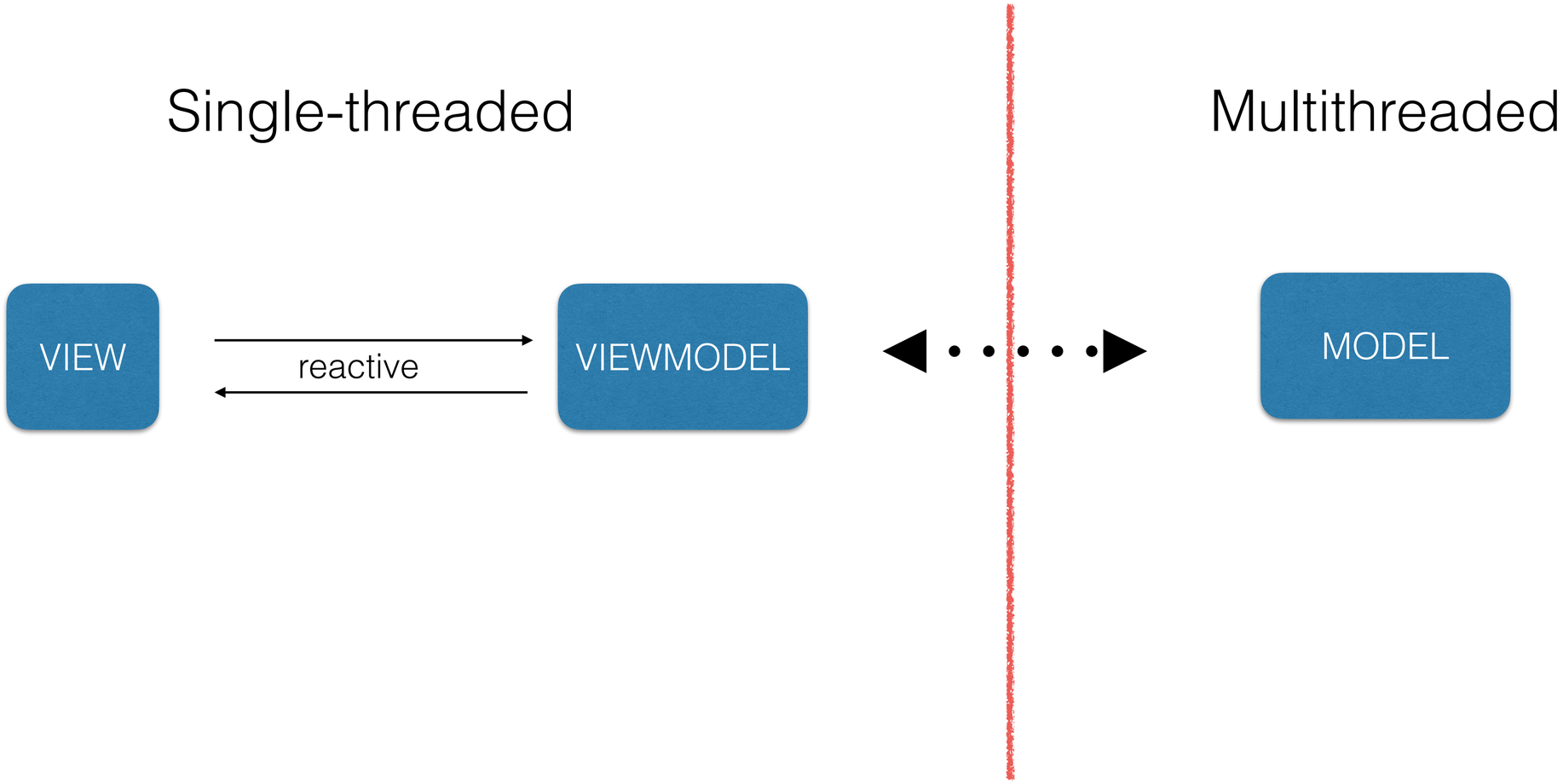
UI Thread



...

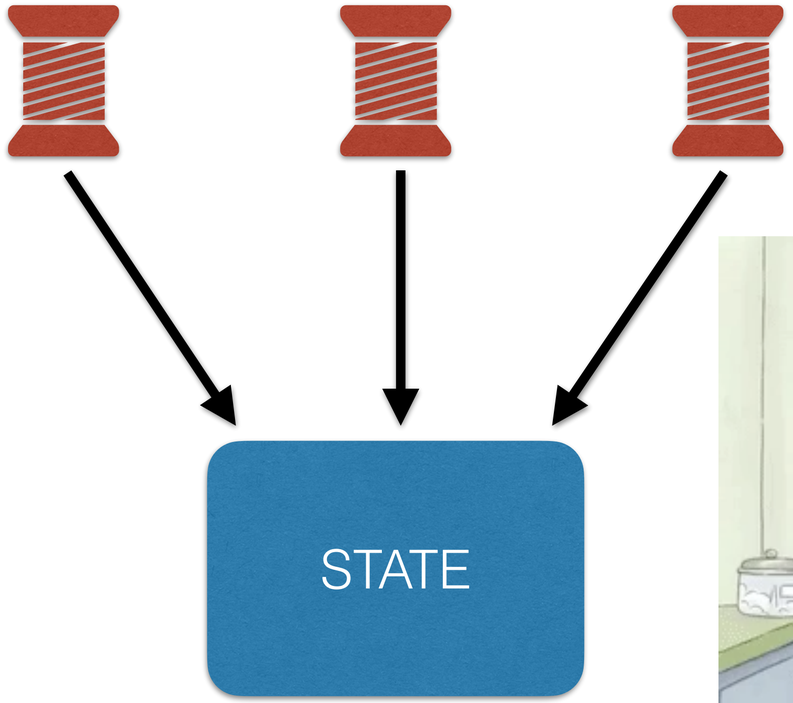
THREADING MODELS

UI Application



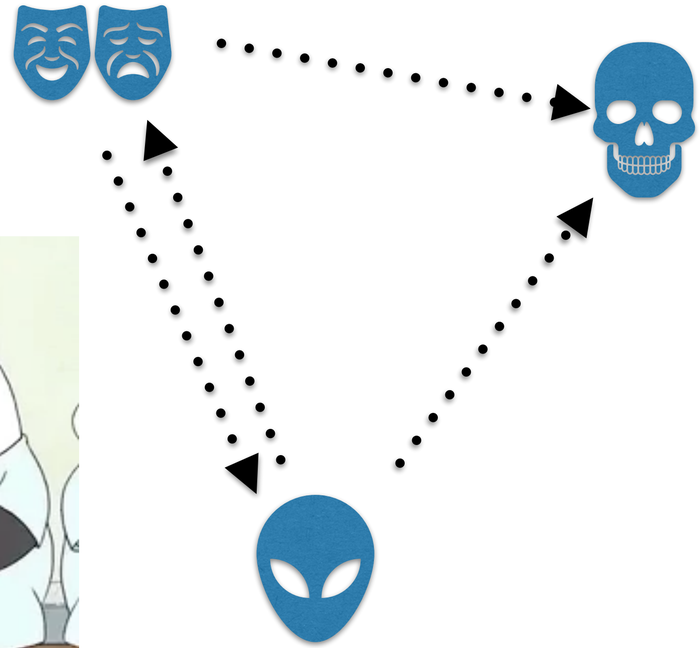
Threading models

Shared state



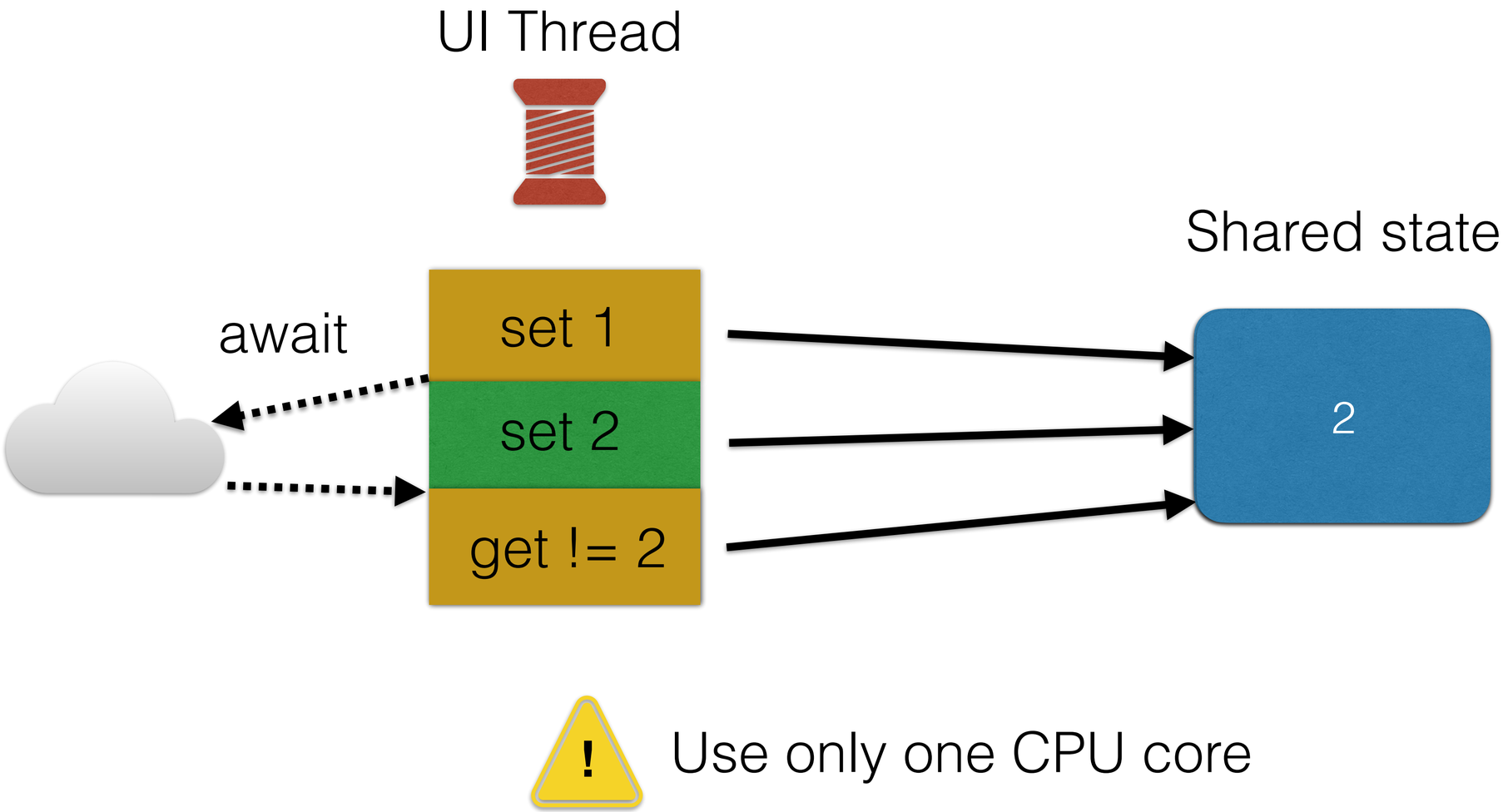
Shared + Mutable = **DANGER**

Message passing

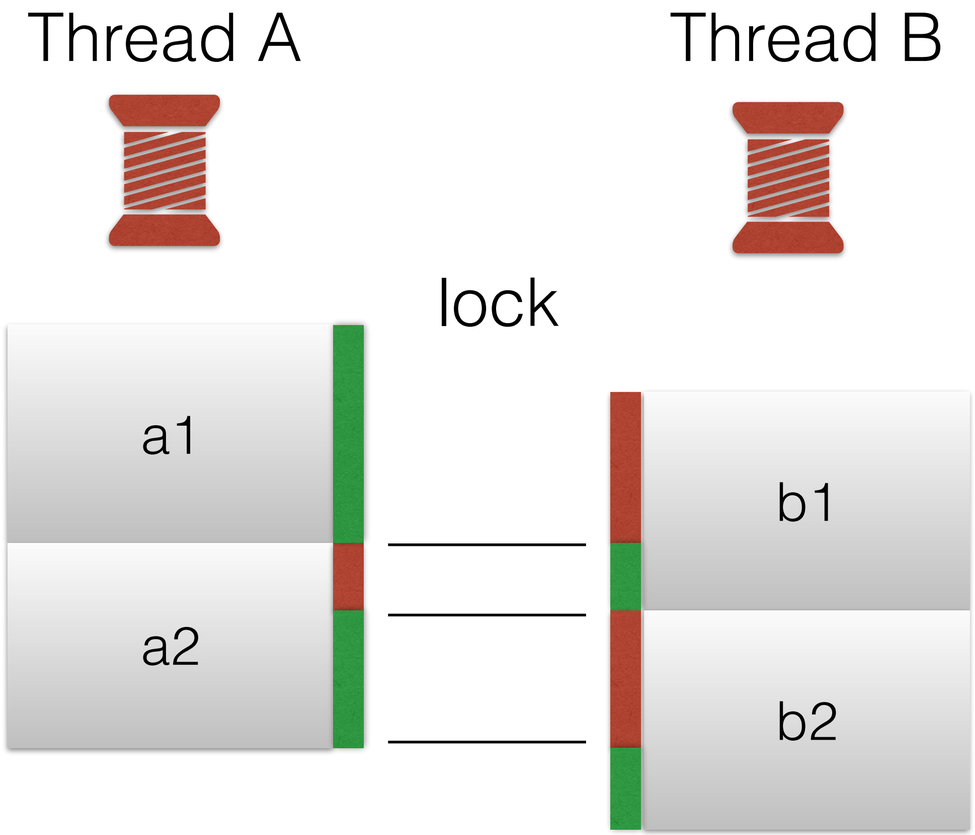


channels, actors,
STA threads

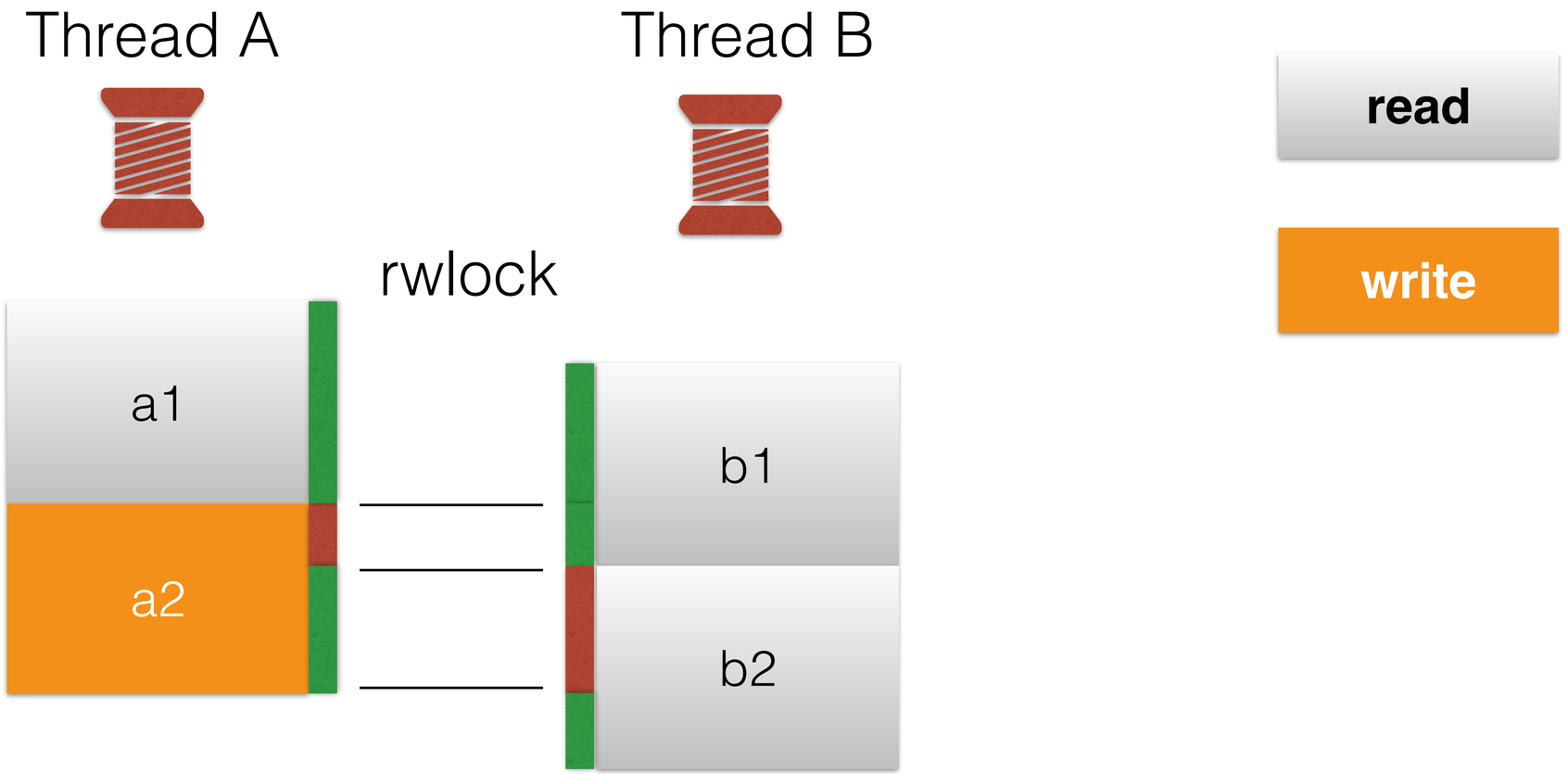
Single thread



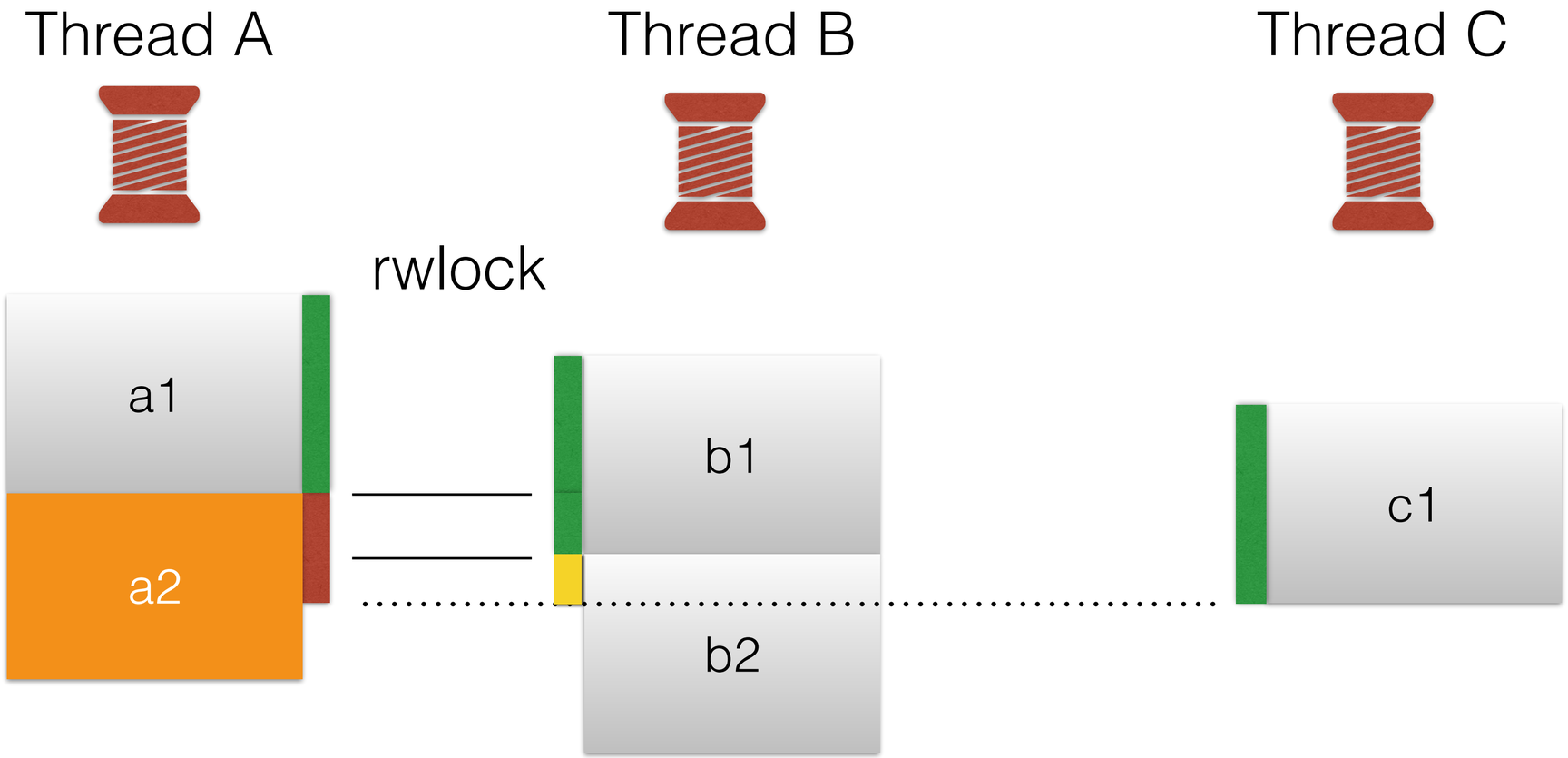
Pessimistic locking



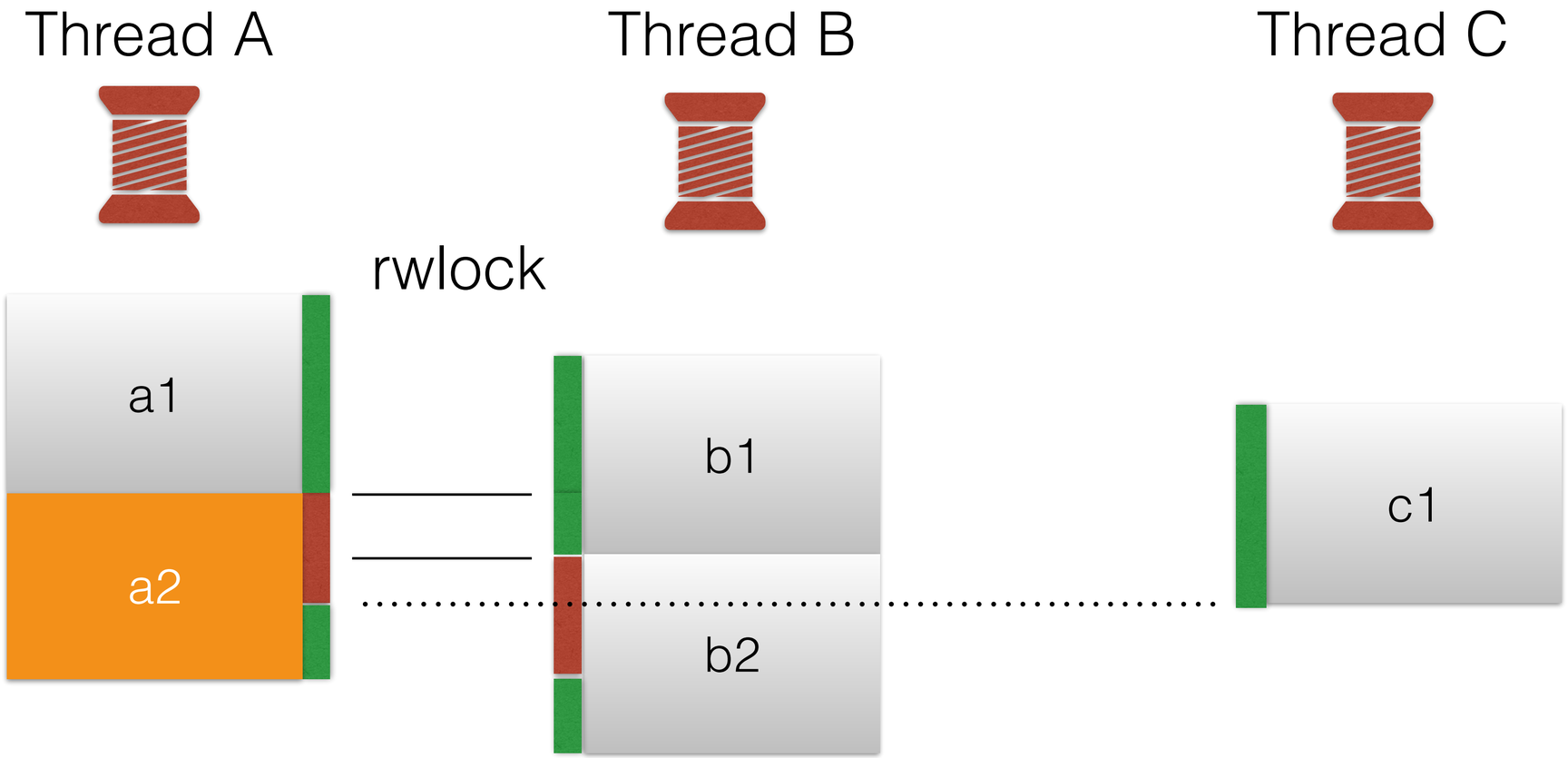
Readers-writer lock



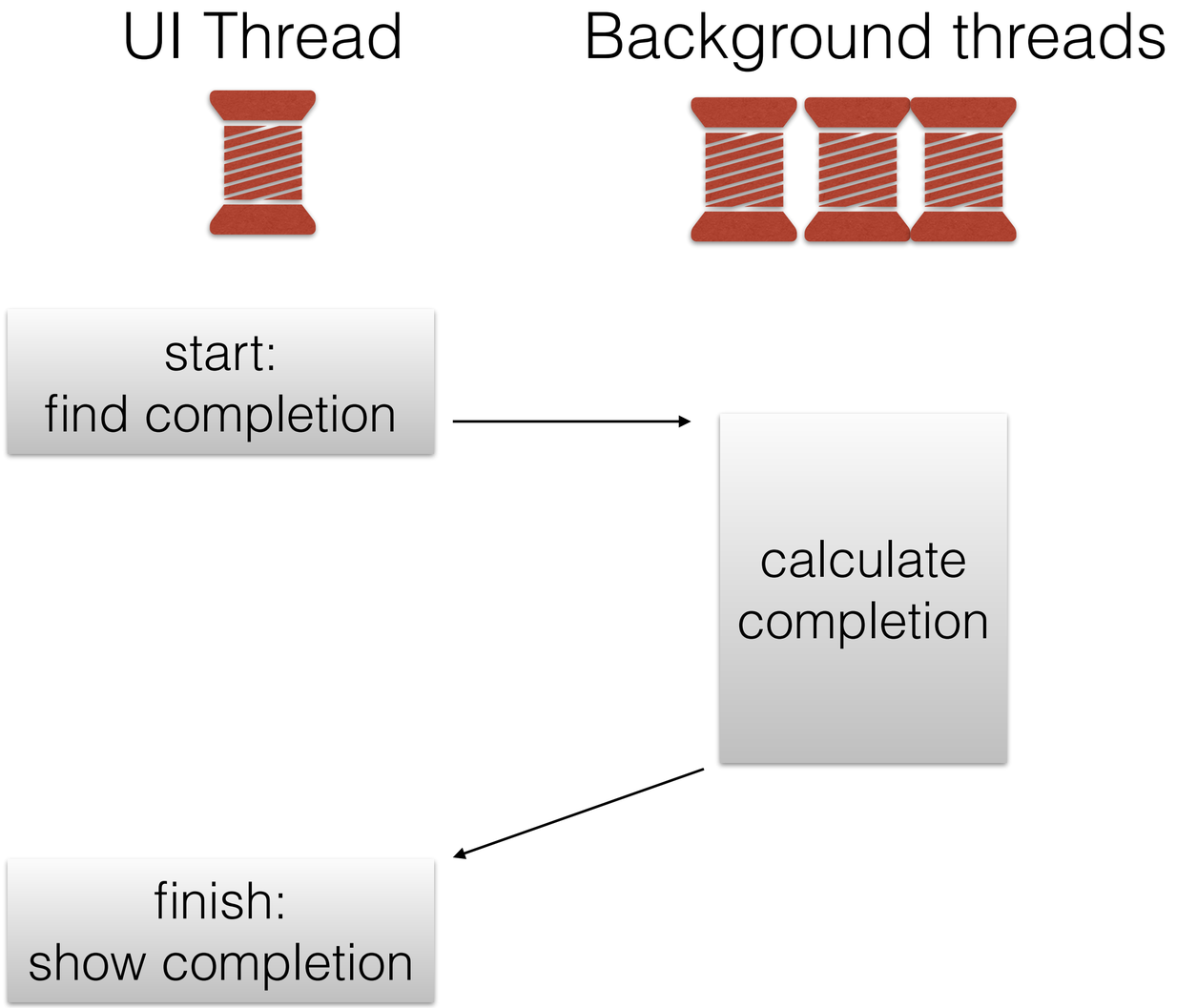
Readers-writer lock



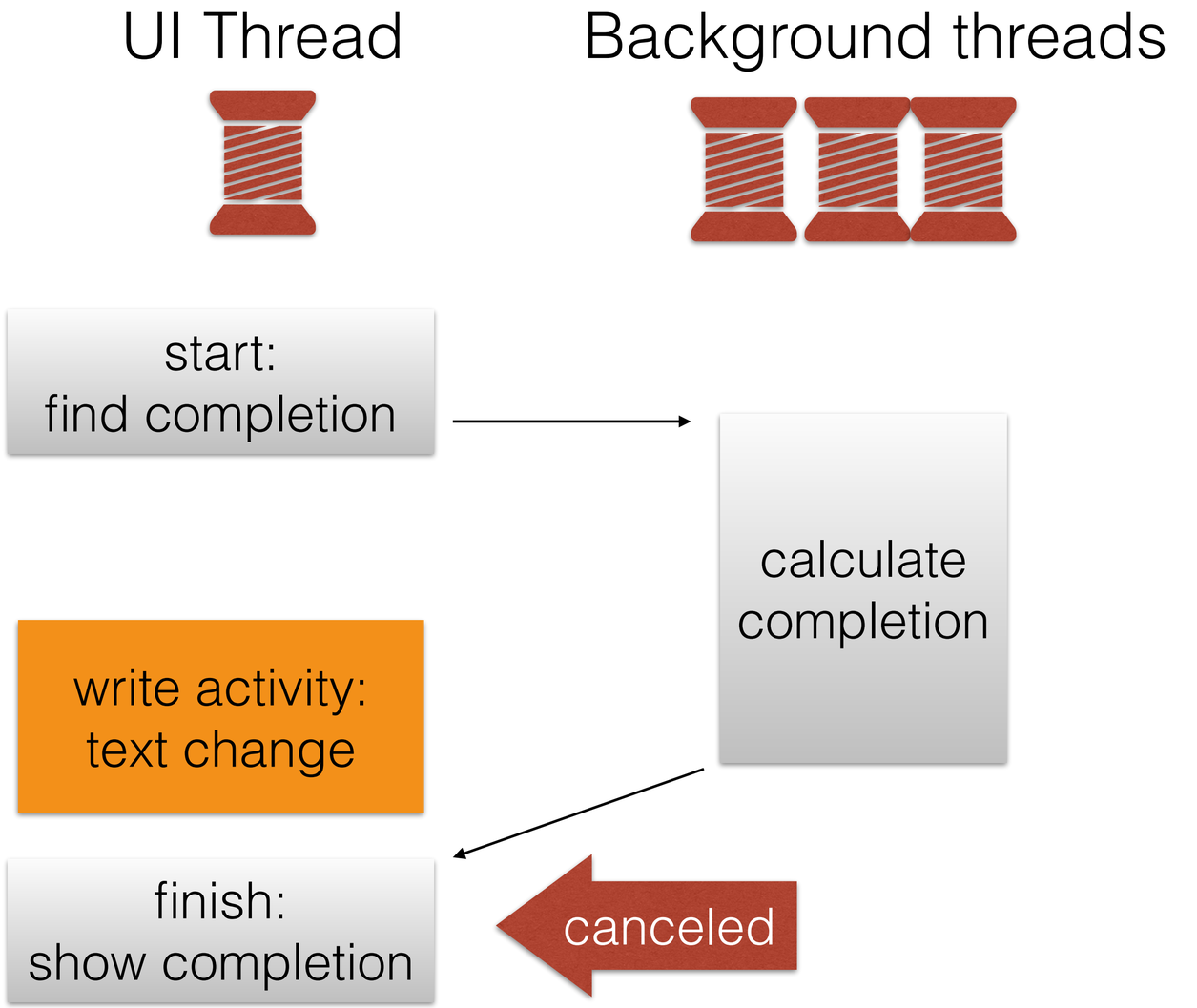
Readers-writer lock



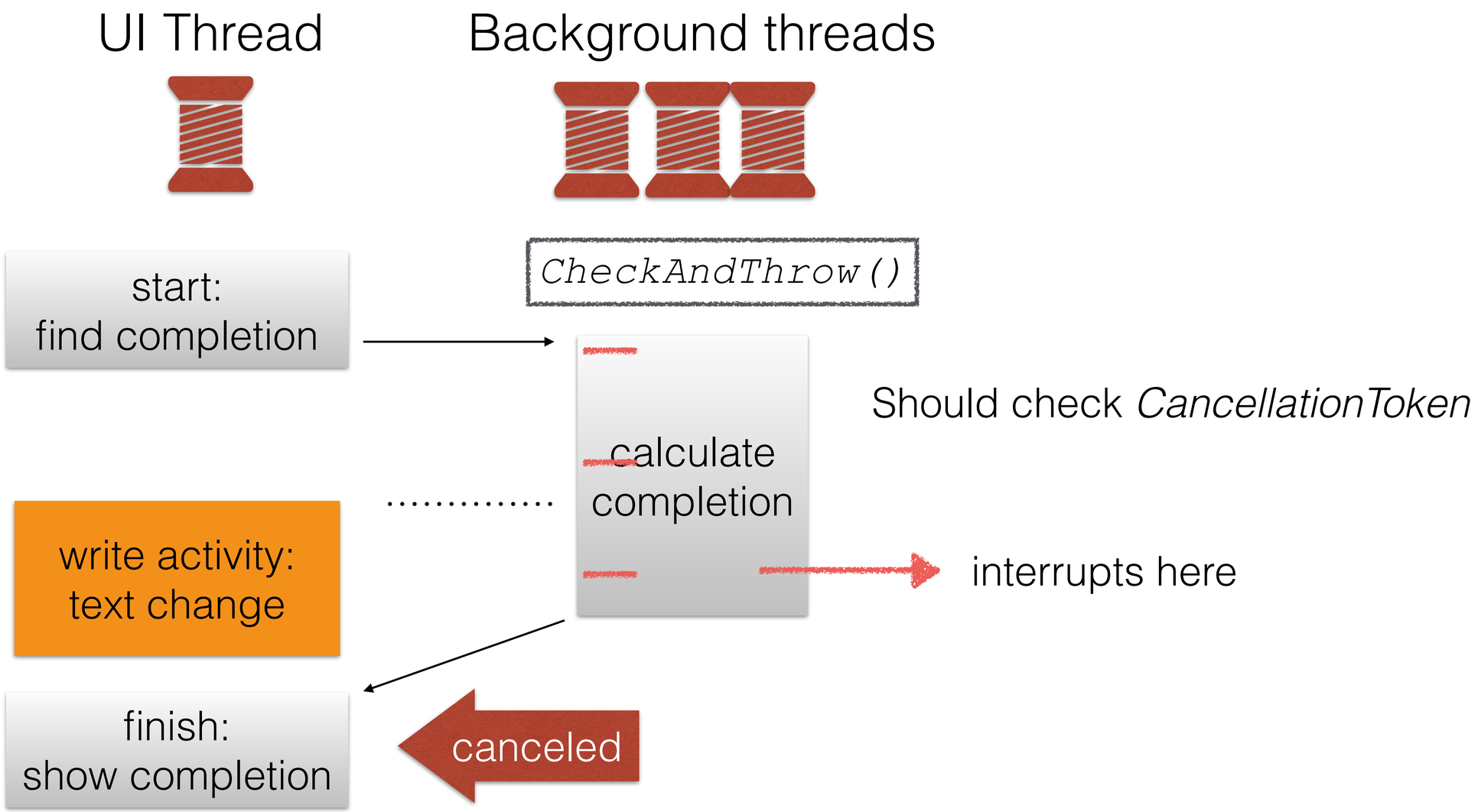
ContentModelReadWriteLock



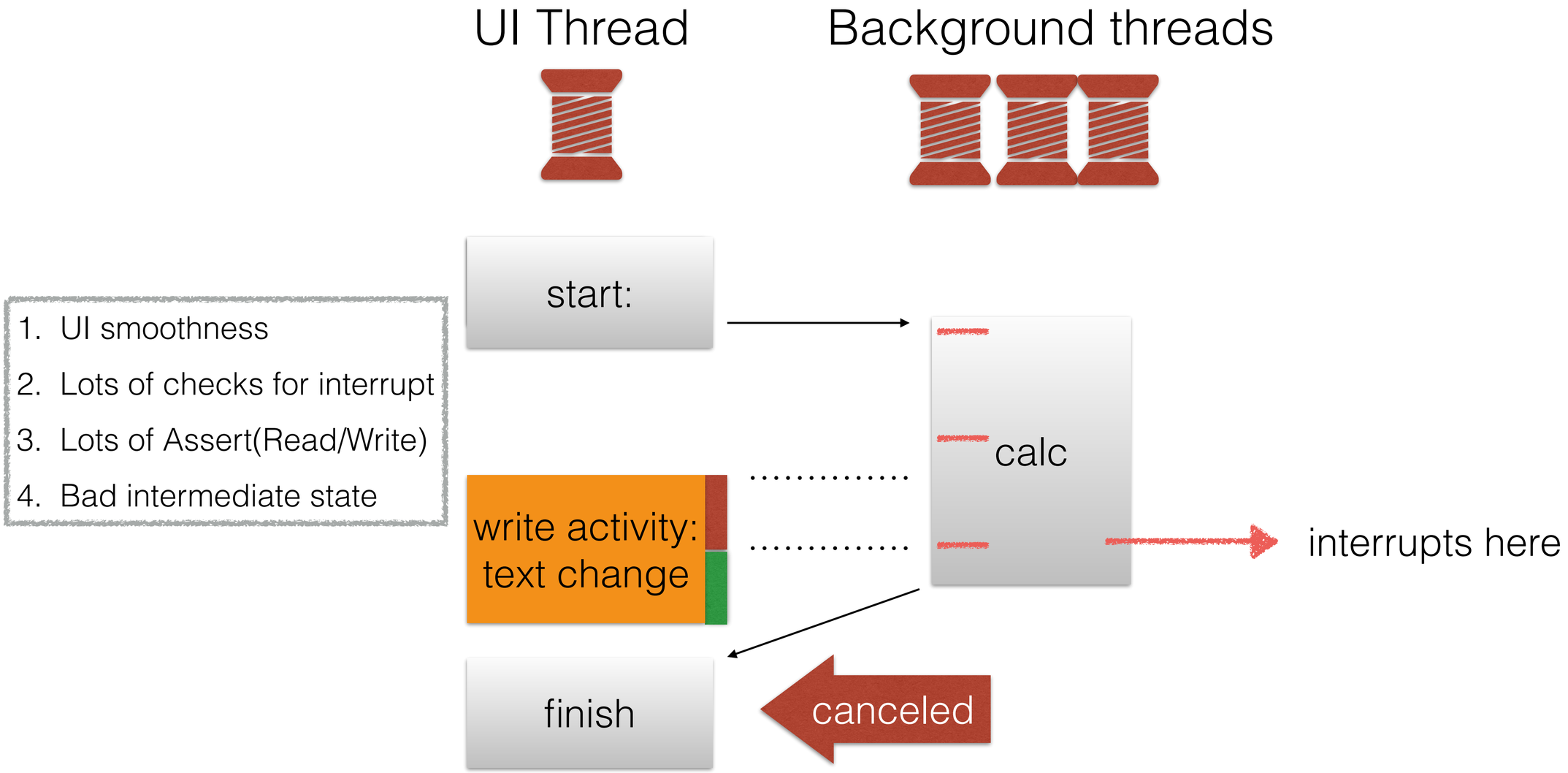
ContentModelReadWriteLock



InterruptableReadActivity



Problems with model



Smoothness diagnostics

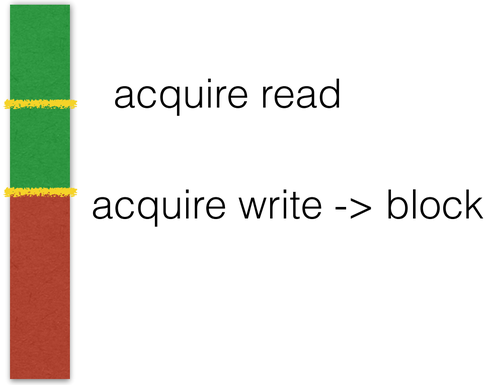
- Assert that interval between *CheckAndThrow()* <= 100 ms
- If acquiring WriteLock >= 100 ms, log stack of release ReadLock
- Watchdog that logs number of tasks and most frequent task of UI thread

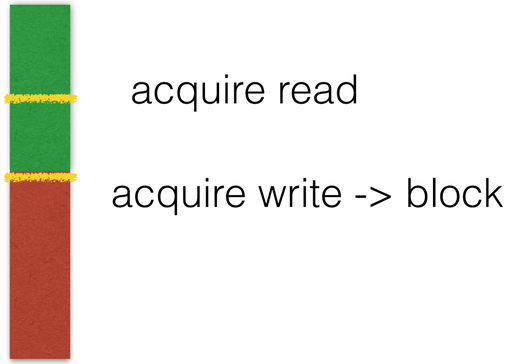
Consistency diagnostics

- Lots of state Assert: UIThread, Read, Write
- Static analysers for threading, e.g. RacerD

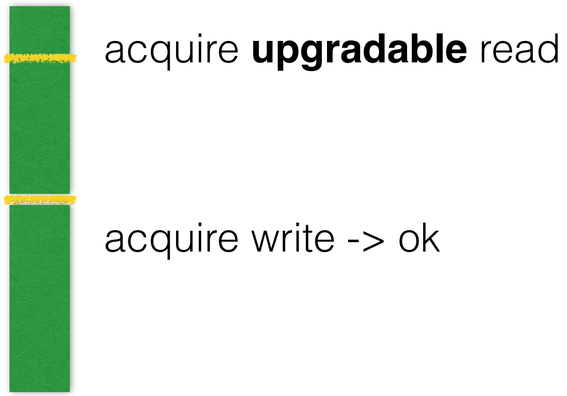
Improvements

- Fine grained locking - **very dangerous path**. Use watchdog for lock order.
- Write-preference lock
- Priorities for UI thread tasks
- Write lock on non-UI thread





Upgradable RW-lock



acquire **upgradable** read

acquire write -> ok

acquire read

acquire write -> **exception**

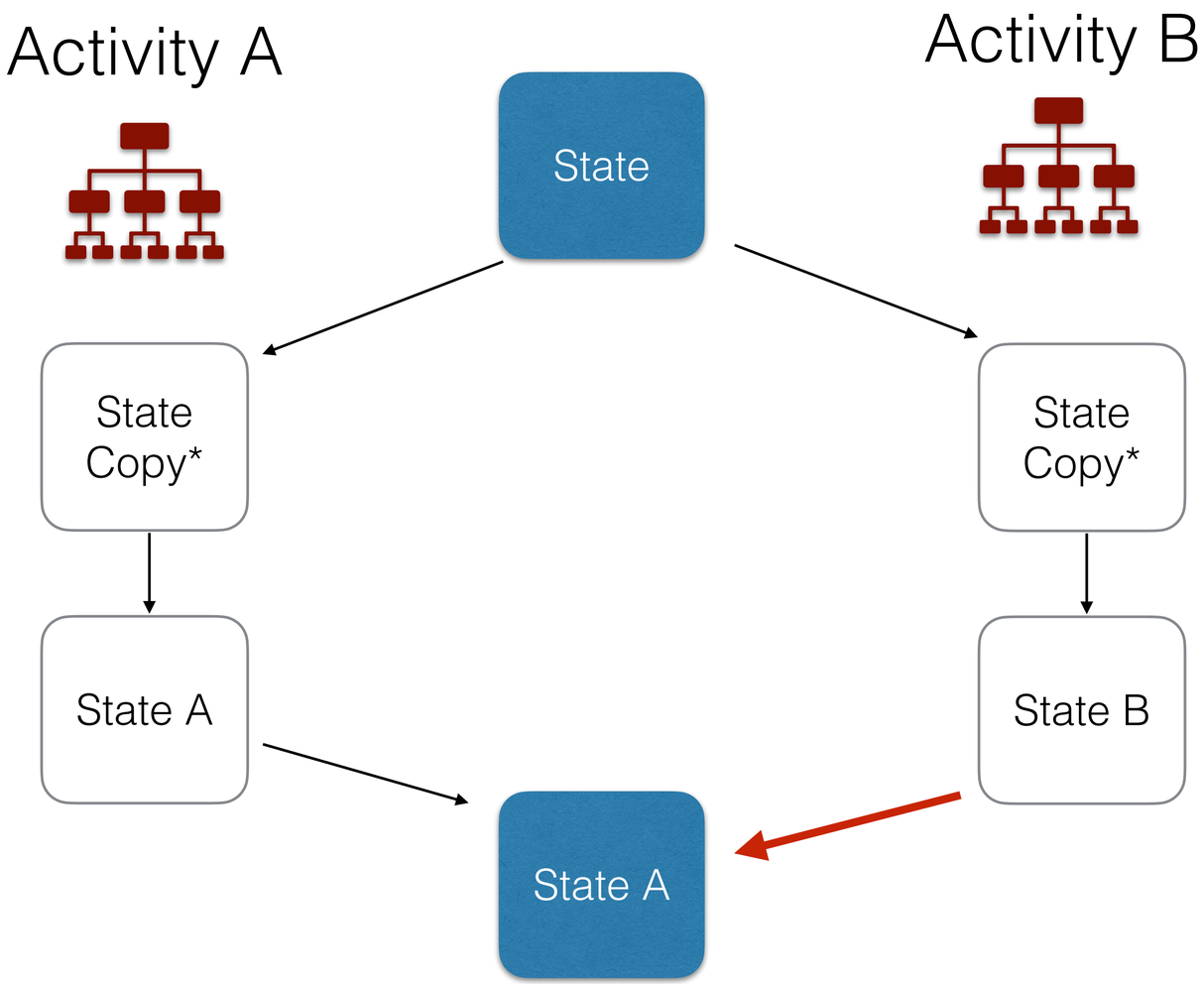
acquire **upgradable** read

IntellijIDEA experience

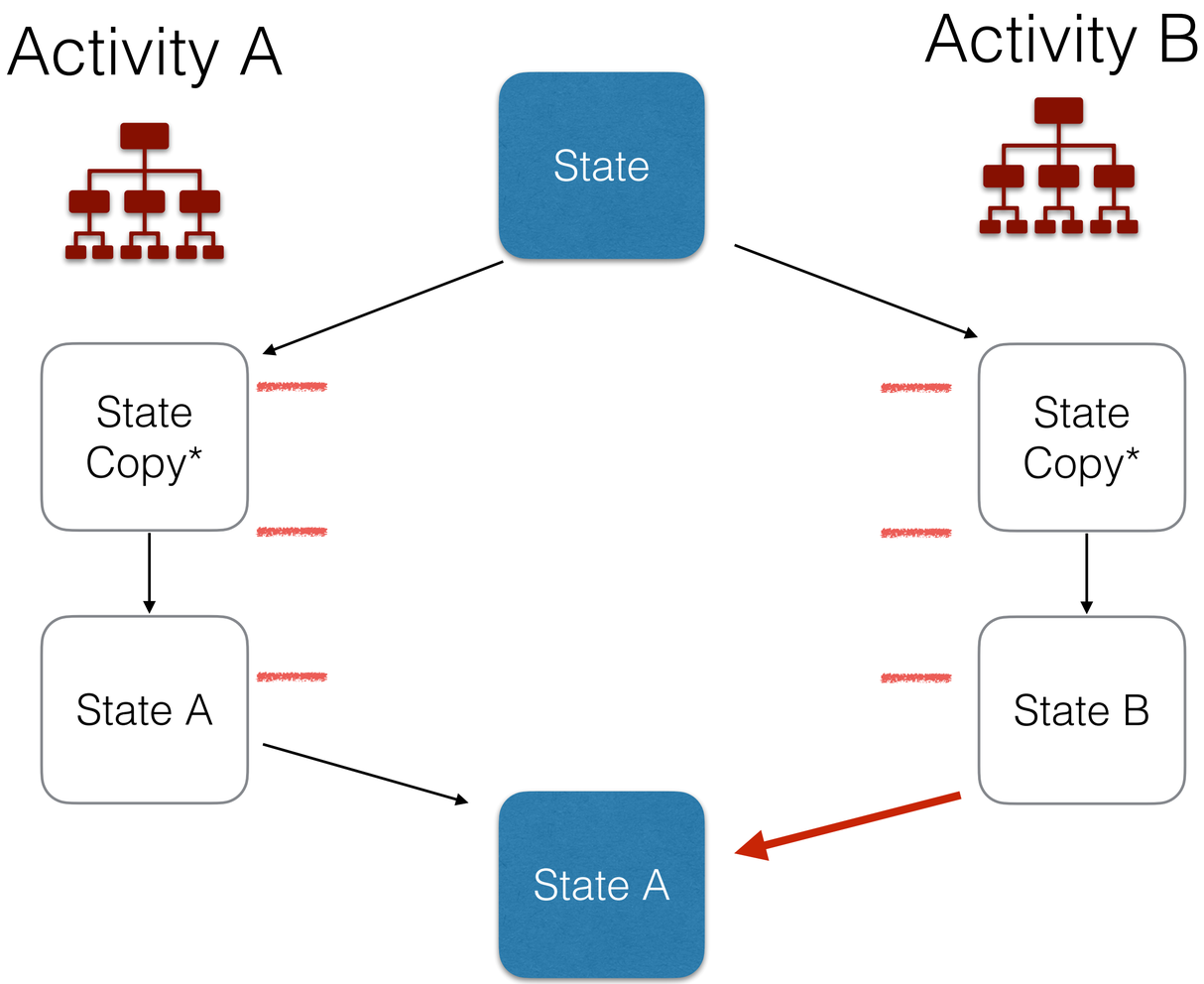
- 20 years of development in RW lock paradigm
- Try to move write actions to non-UI thread via **`executeWriteAction()`**
- 6 month of work
- Not in production 😢

Other models

Immutable model



Immutable model

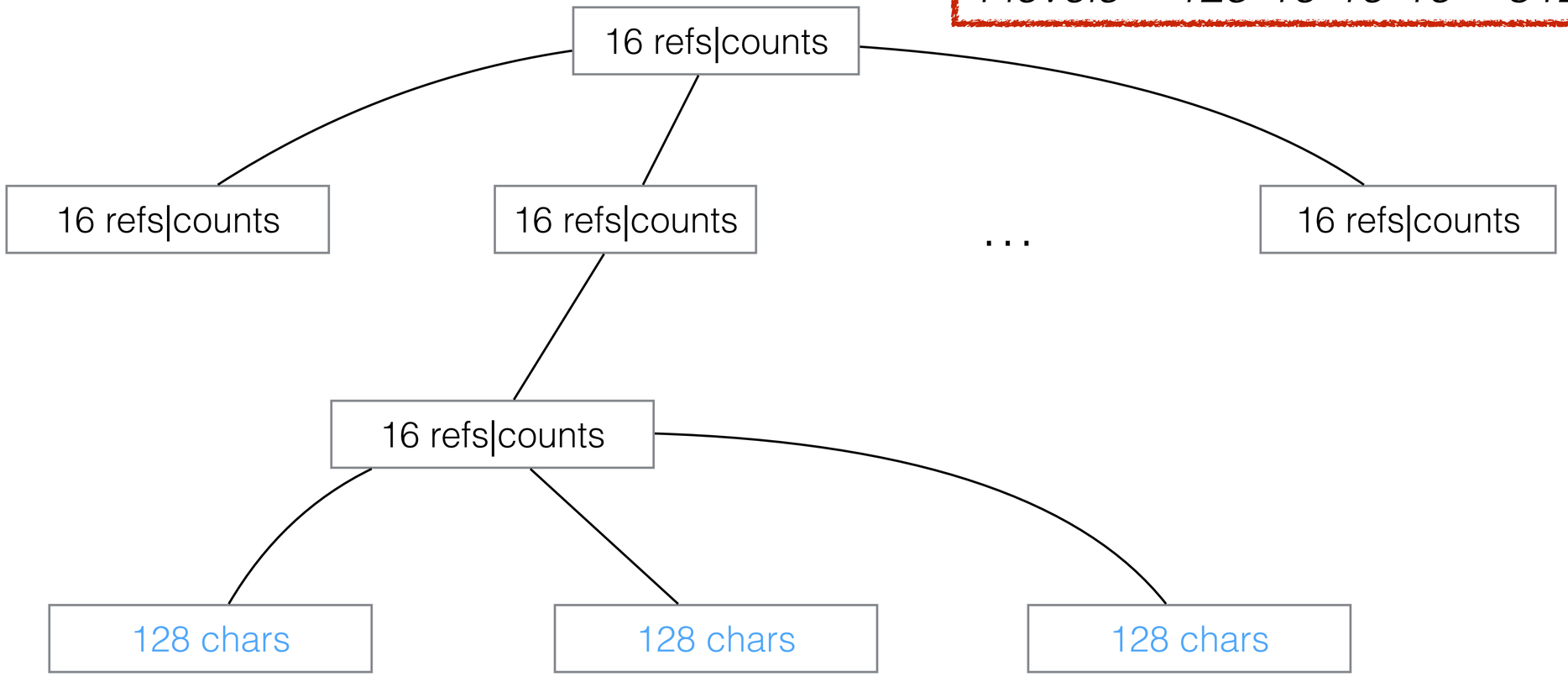


Immutable TextEditor

var s = `1Mb of text`

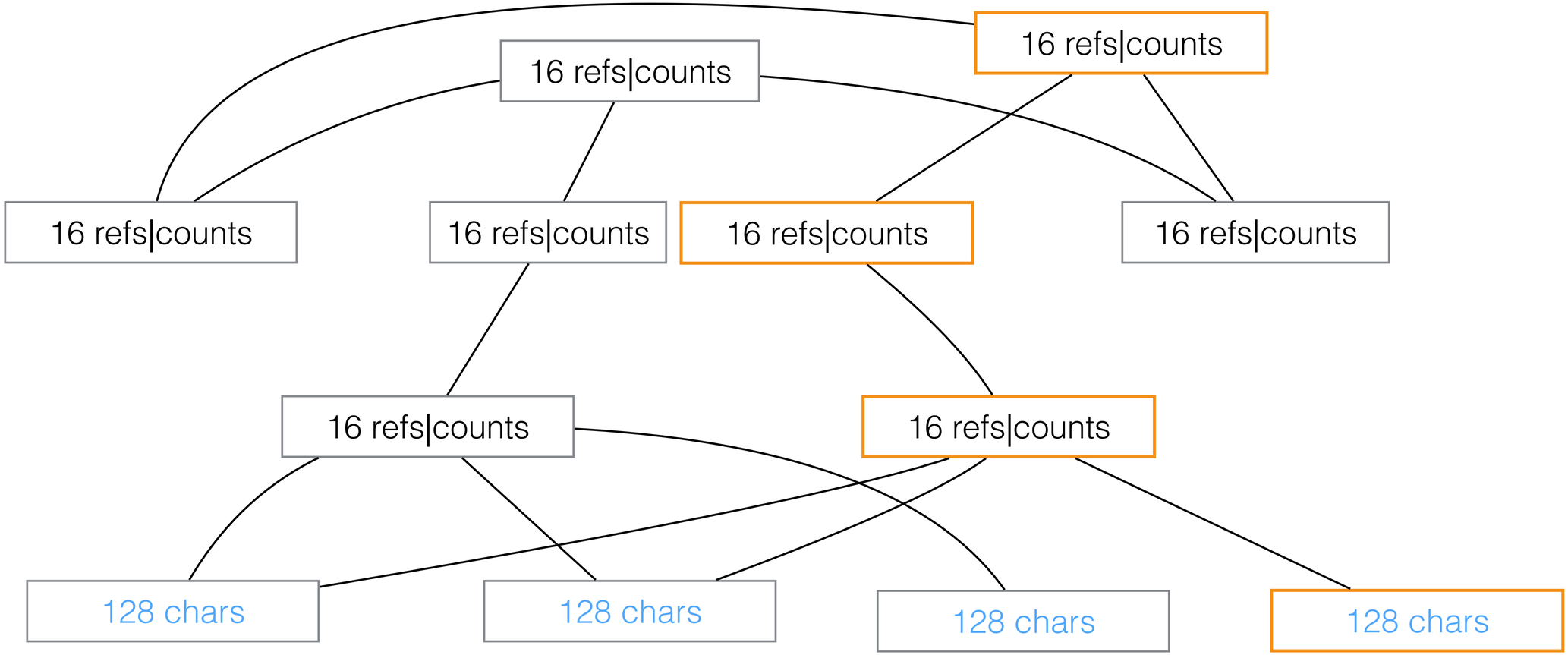
*256 bytes block = 16*8*2 = 128 chars*

*4 levels = 128*16*16*16 = 512K chars*

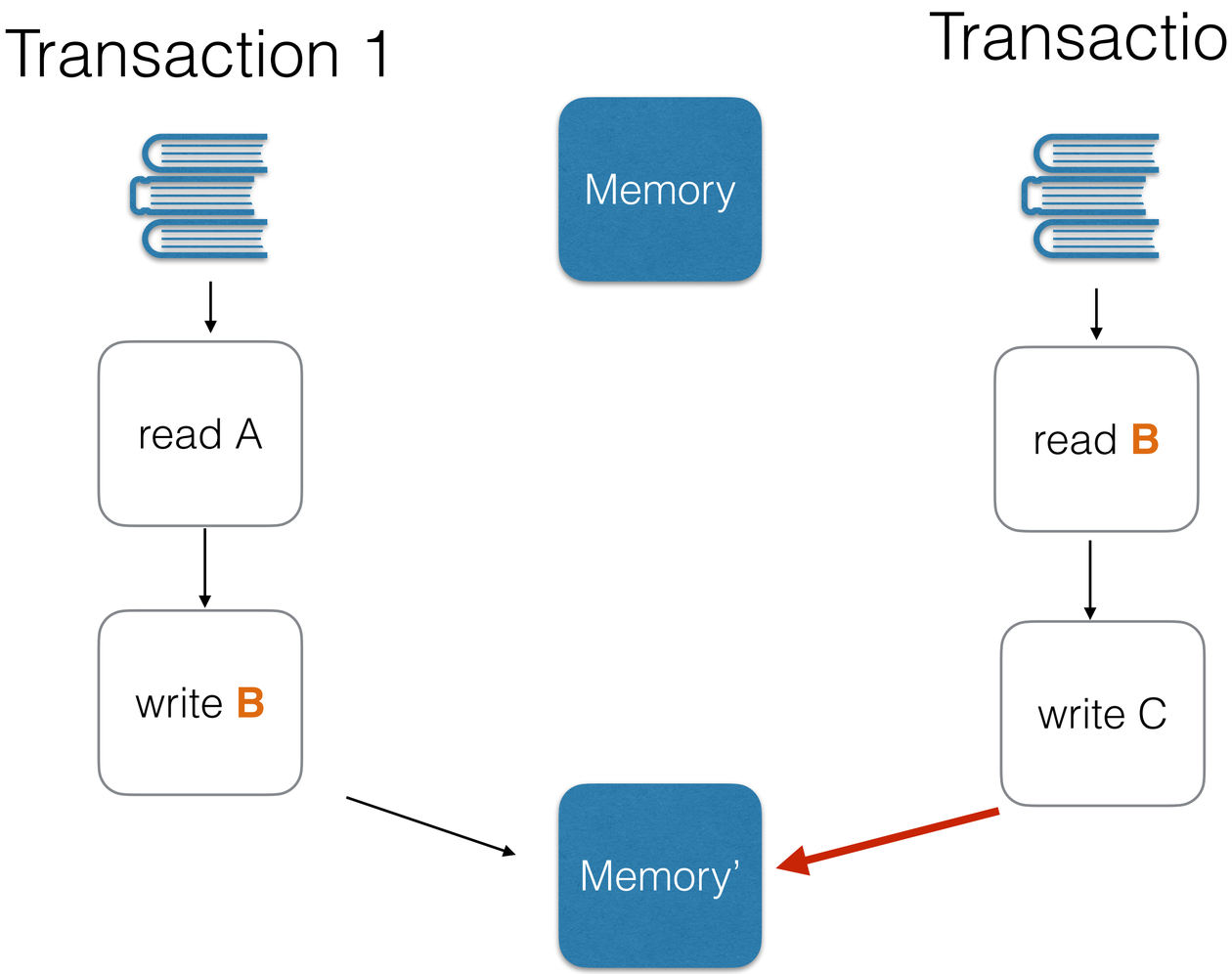


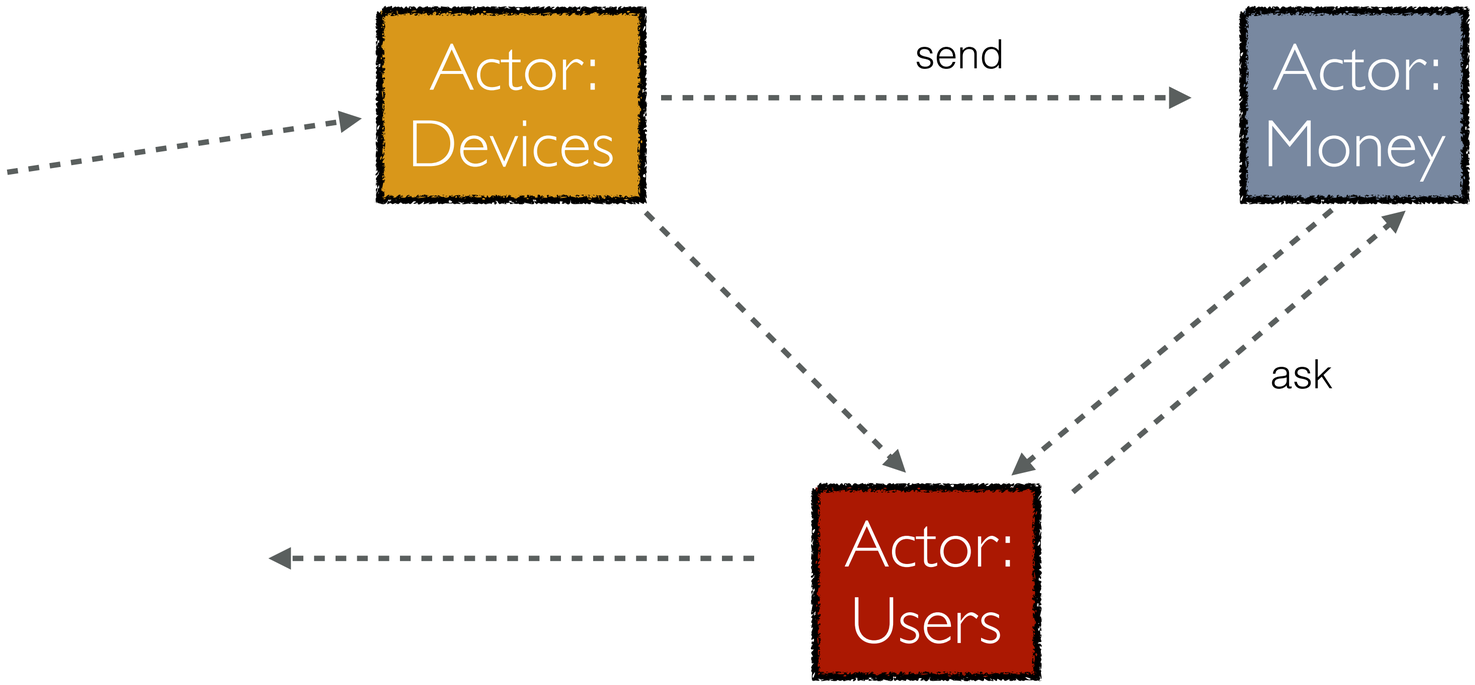
Immutable / Pesistent Tree

var s = `1Mb of text`



Software transactional memory

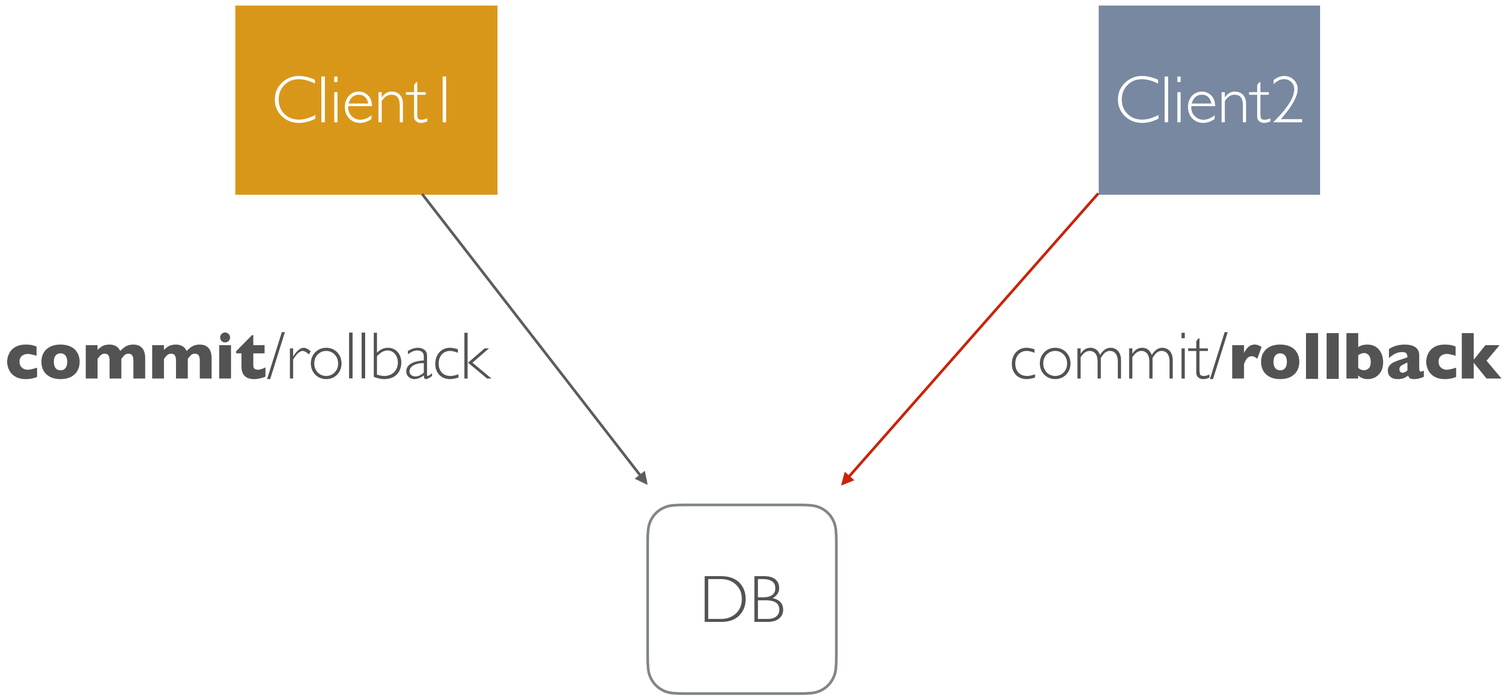




PERSISTENT ACTORS

JetBrains Space experience

- One year of actor-based development
- Lots of distributed deadlocks
- No good tooling for diagnostics and prevention of deadlocks
- Actor approach is dropped in favour of SQL



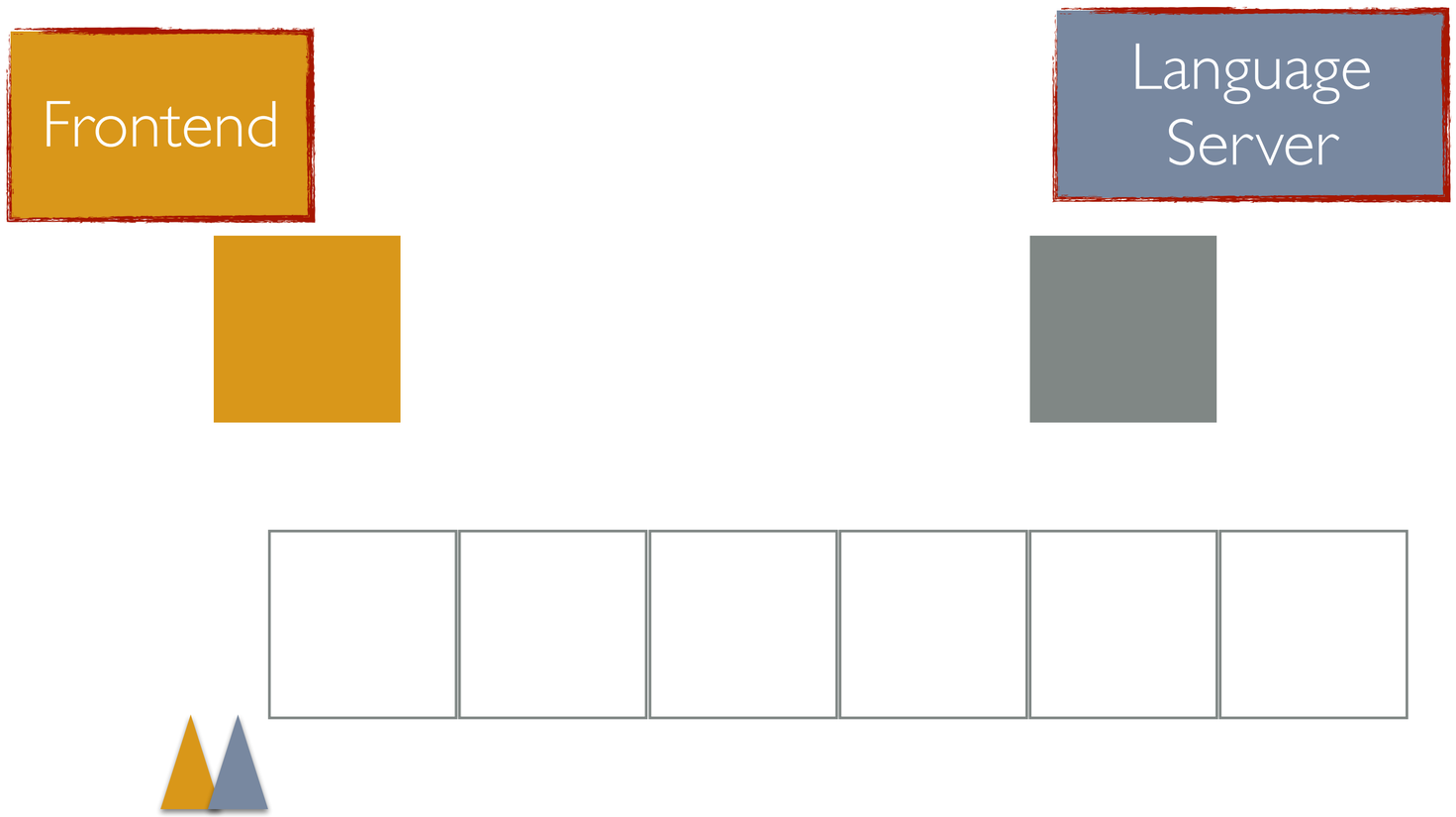
TRANSACTIONAL DATABASE

Future: Fast non-volatile memory

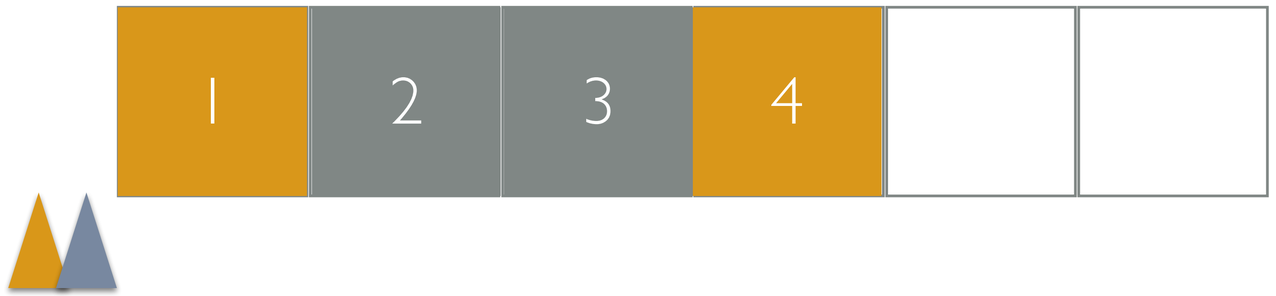
Frontend

Language Server

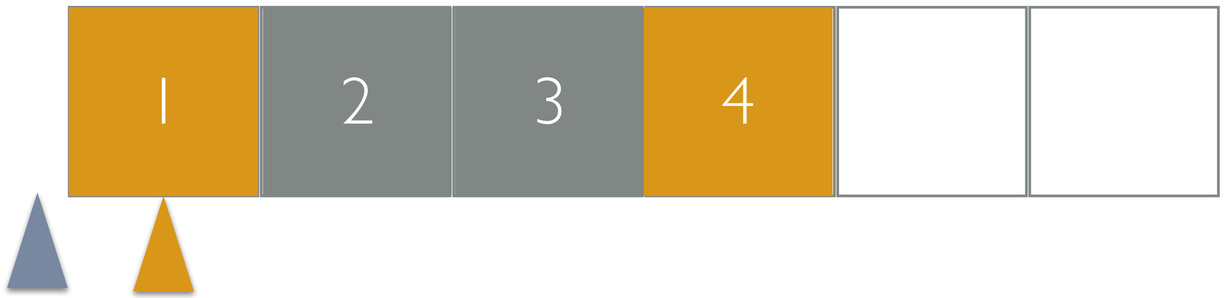
EVENT SOURCING



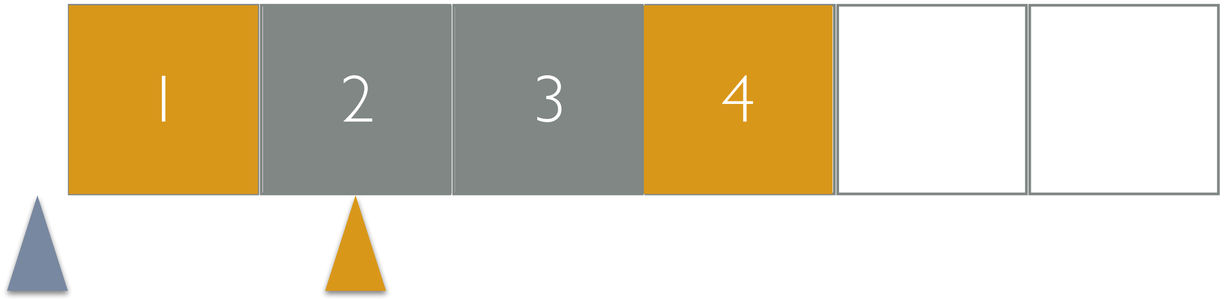
EVENT SOURCING



EVENT SOURCING



EVENT SOURCING



EVENT SOURCING

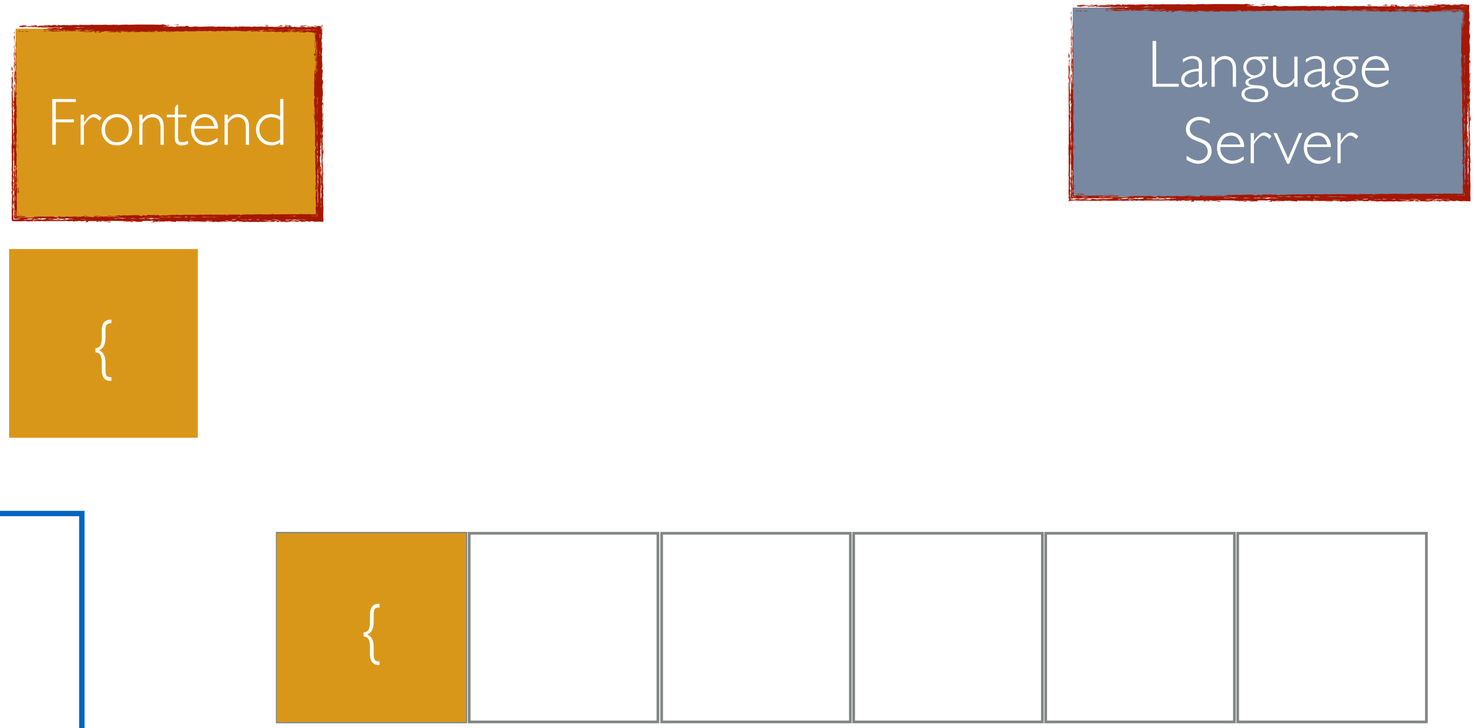
Frontend

2

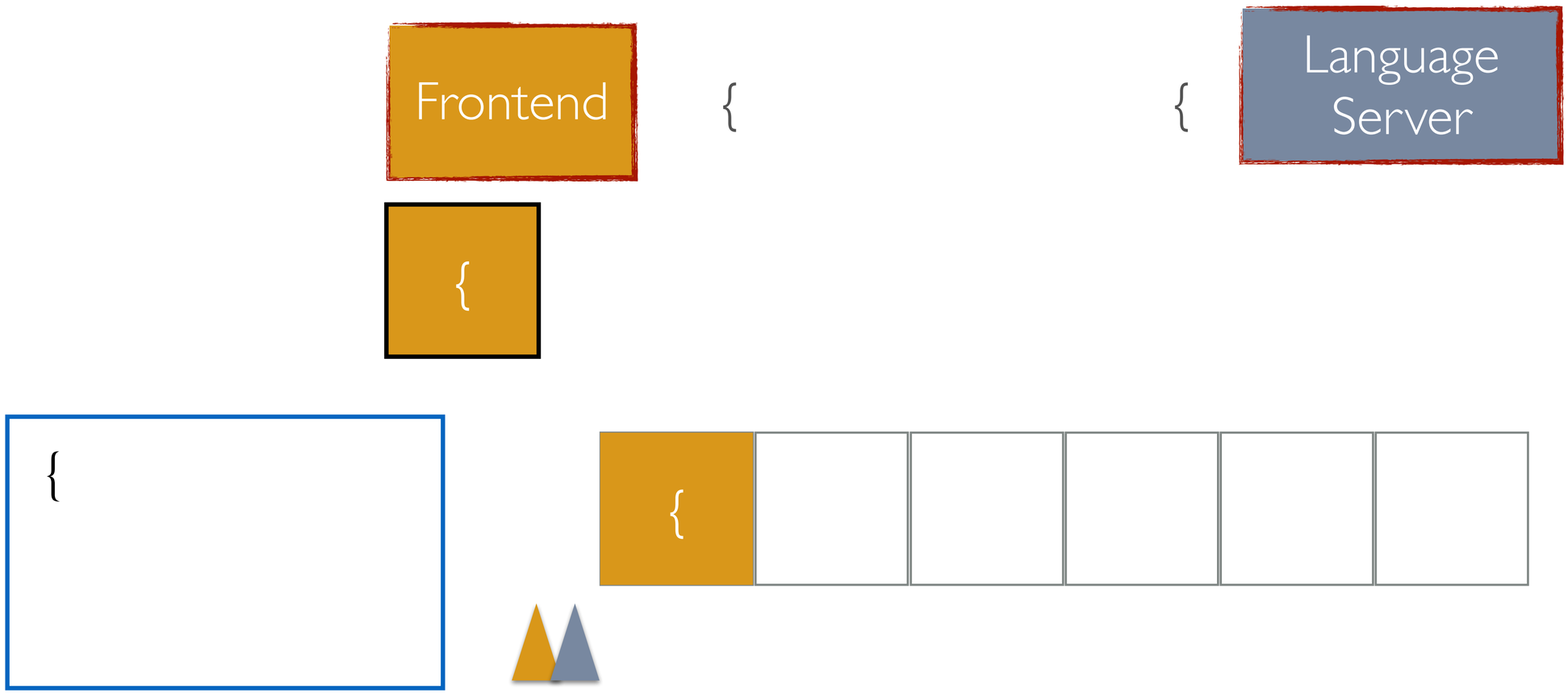
1

Language Server

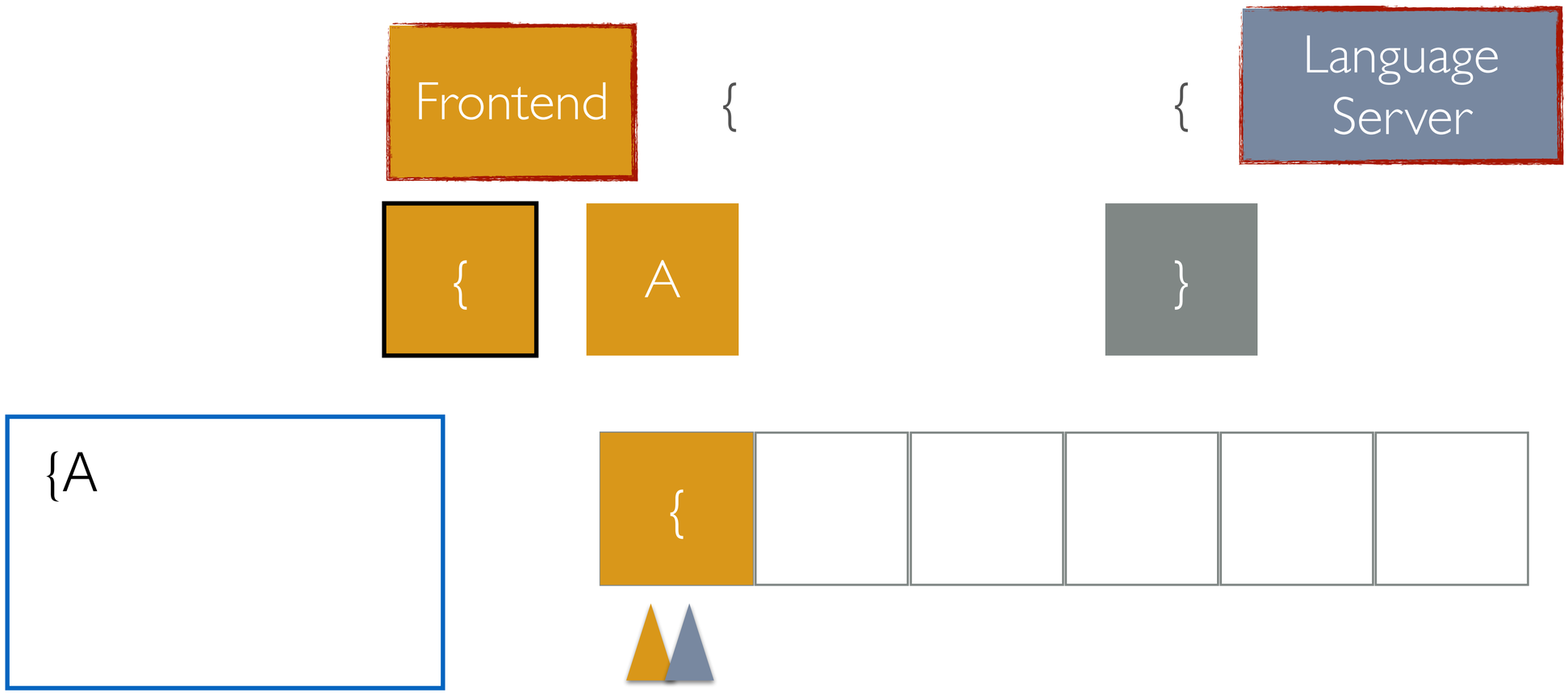
EVENT SOURCING



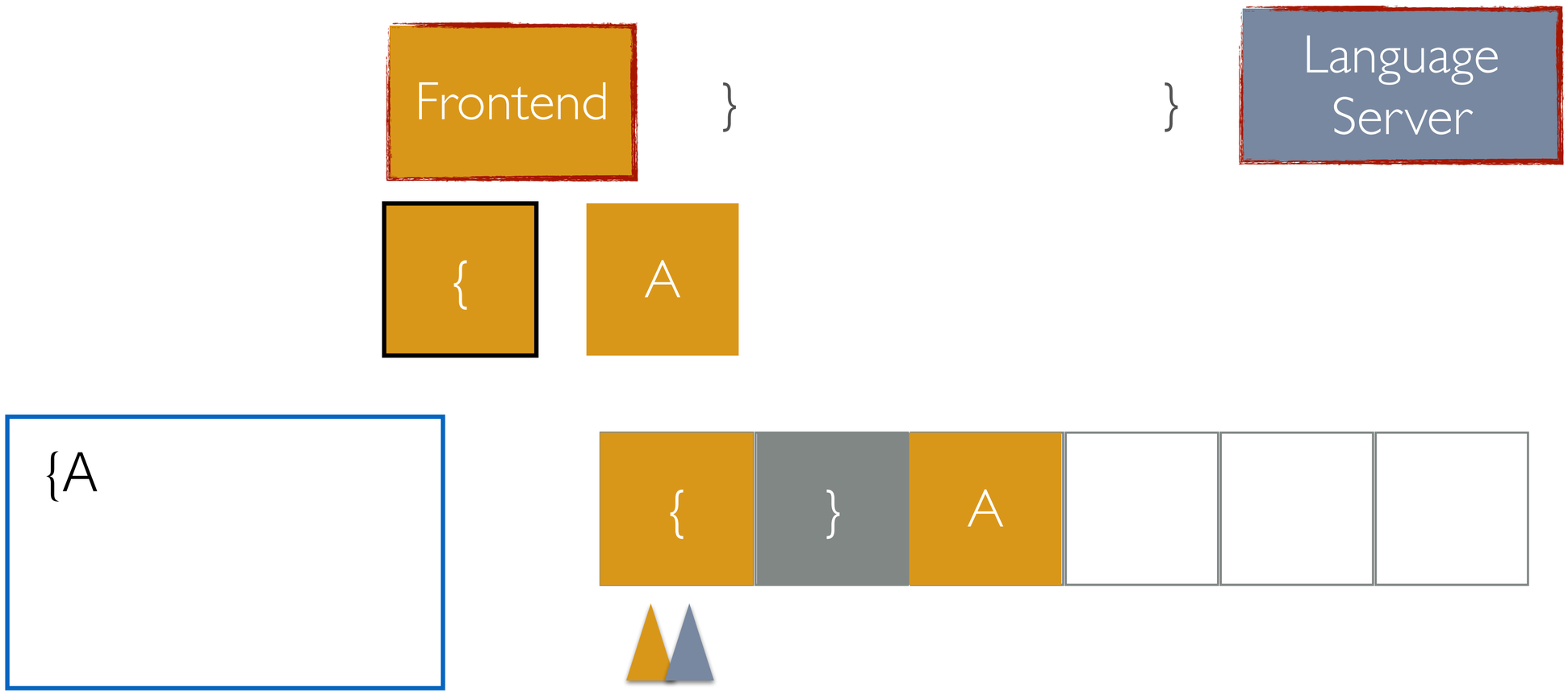
EVENT SOURCING



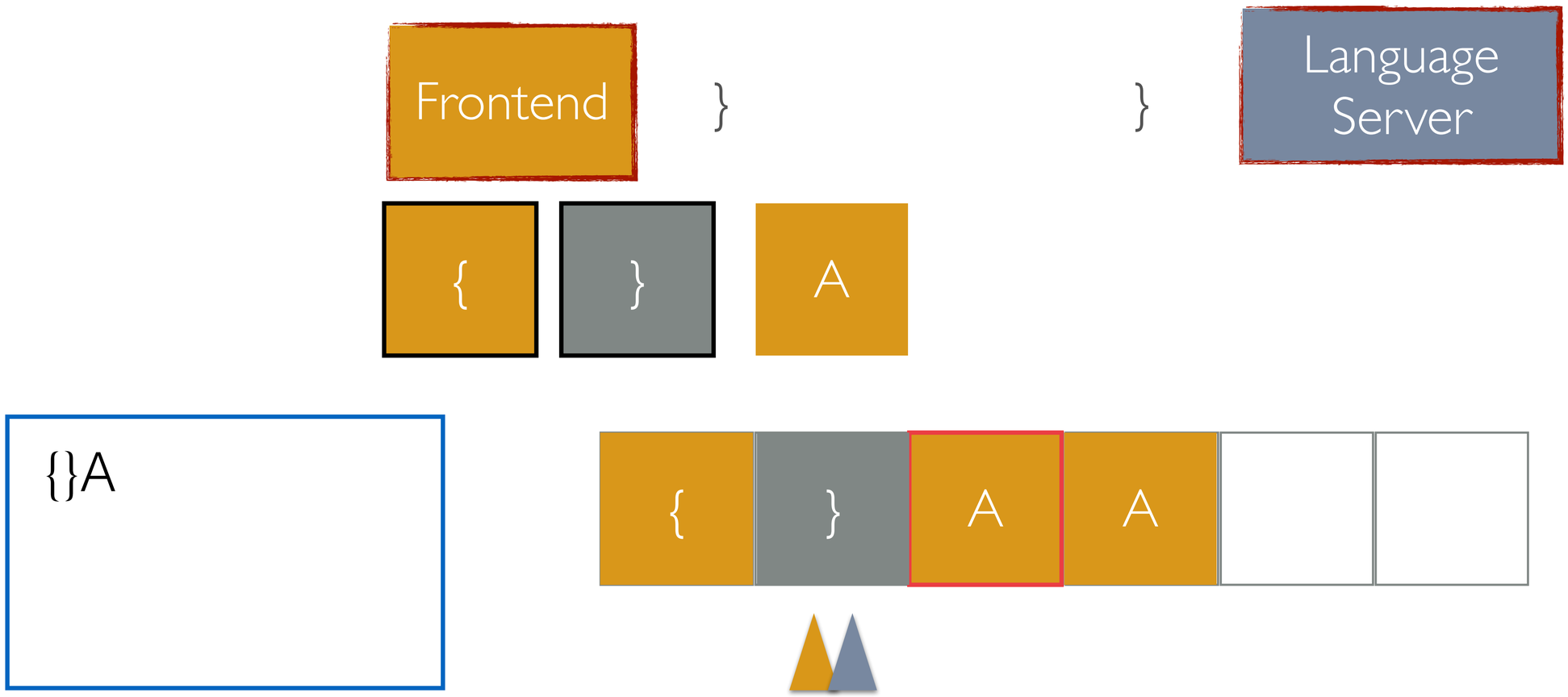
EVENT SOURCING



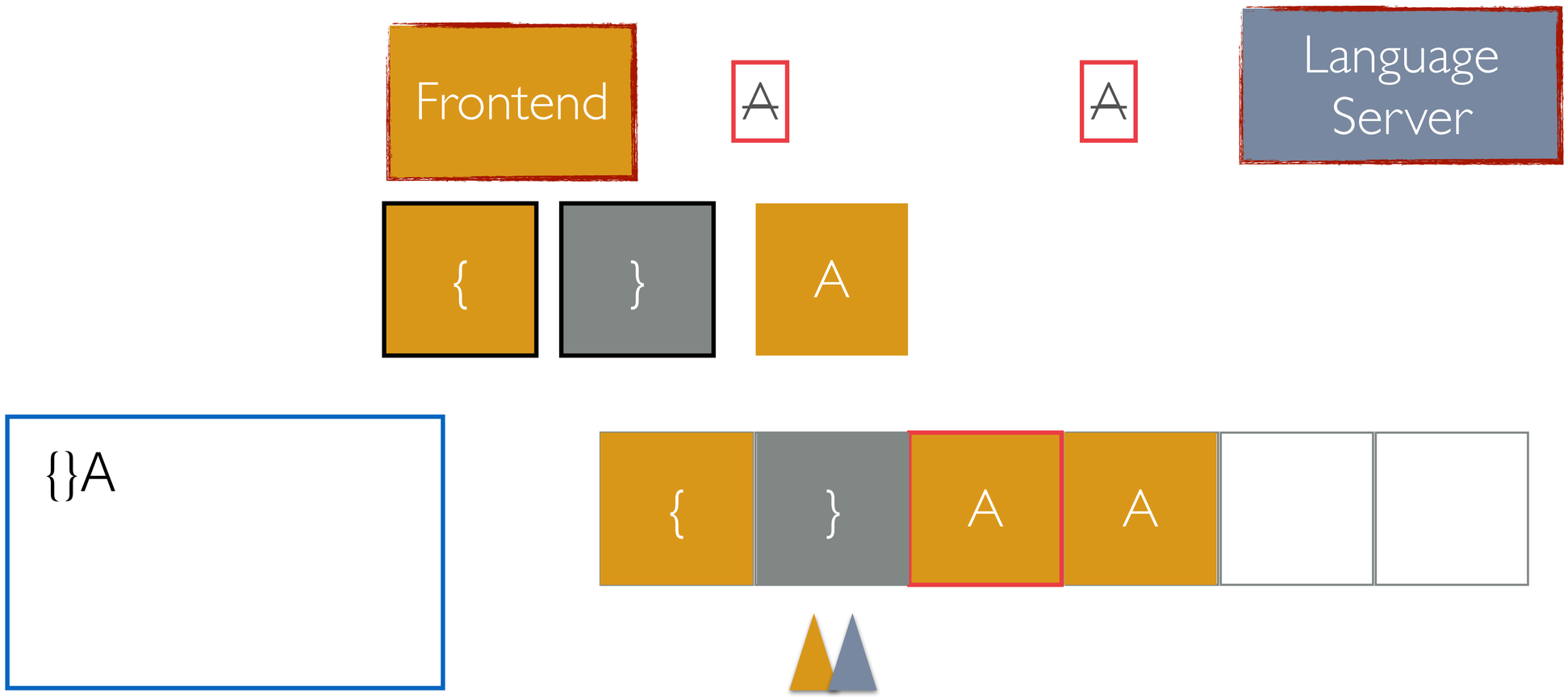
EVENT SOURCING



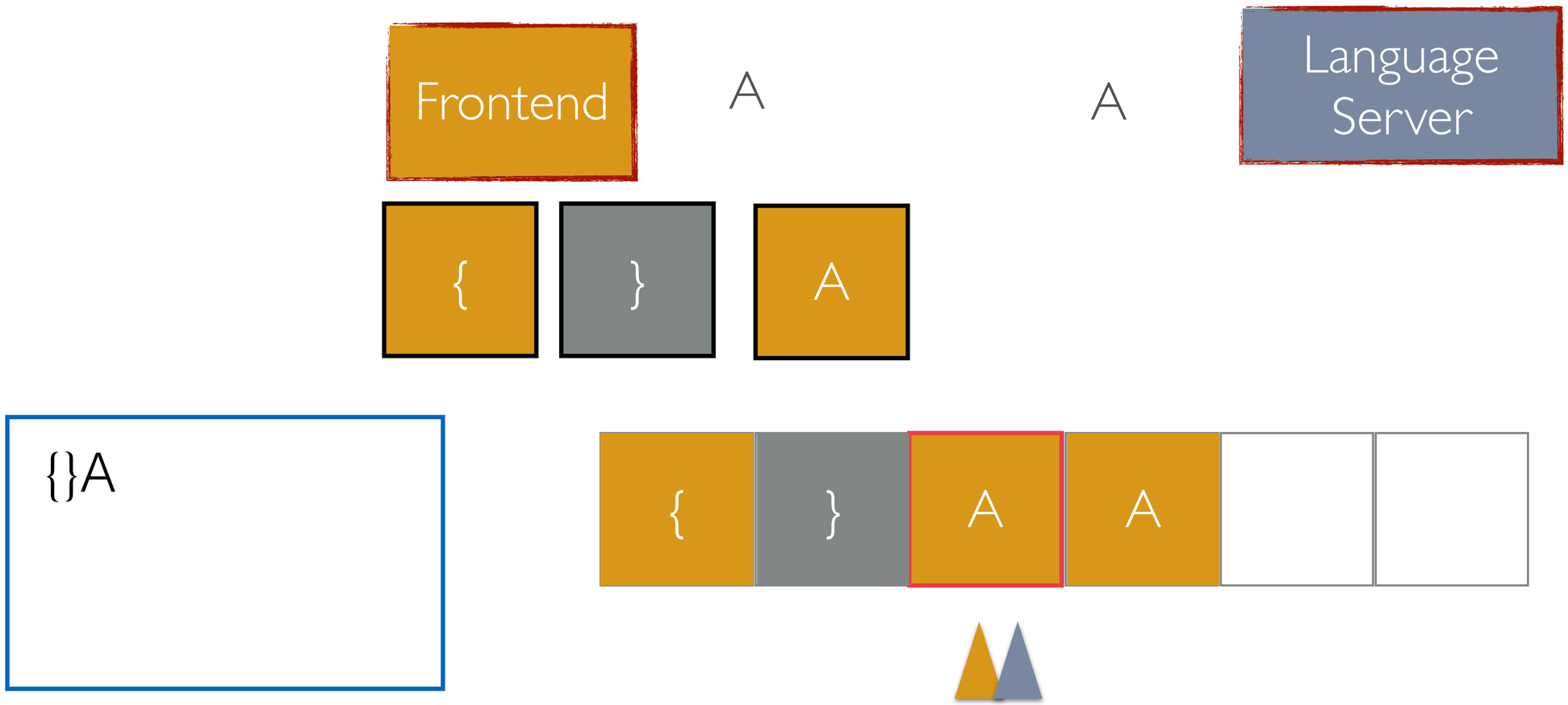
EVENT SOURCING



EVENT SOURCING



EVENT SOURCING



EVENT SOURCING

Questions and answers