Concurrency Basics

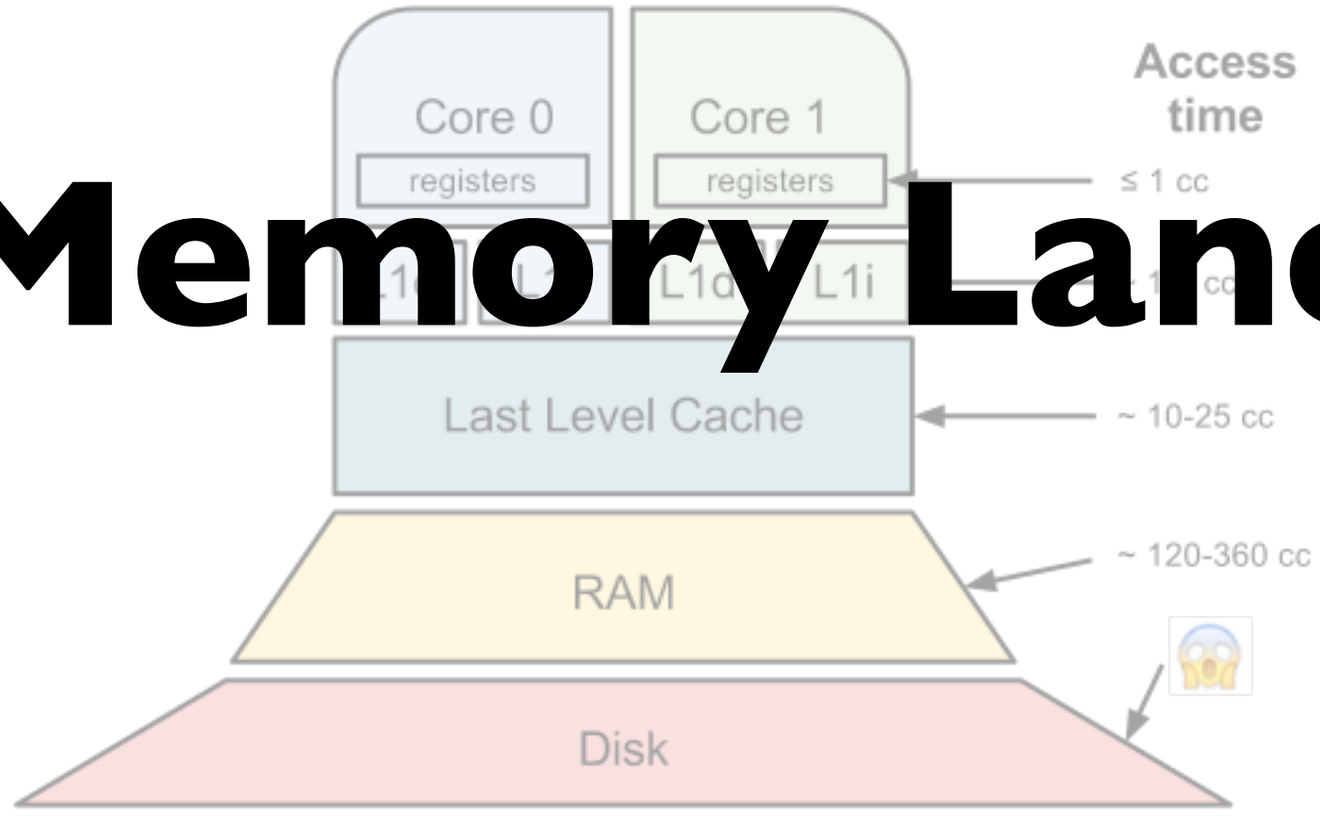
Joining the Data Race


CIRQUE DU SOLEIL

In the Queue

- Going Down Memory Lane
- Mistakes: What can possibly go wonrg?
- Abstraction: Tour de Tool Box

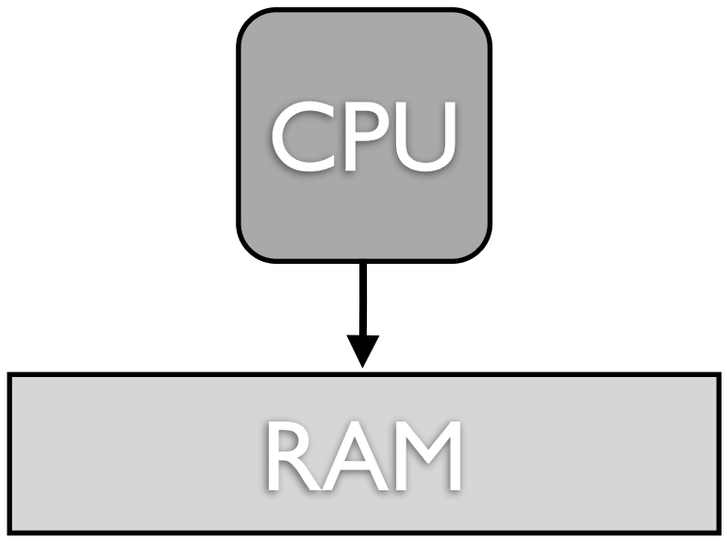
Memory Lane


Core 0
registers
Core 1
registers
L1d
L1i
Last Level Cache
RAM
Disk
Access time
≤ 1 cc
~ 10-25 cc
~ 120-360 cc
😱

In the Old Days*

*Intel 80386, introduced 1½ years after my birth

- Uniform
- In-Order
- Serial
- Exclusive
- “Fast”



Memory Today

- Non-Uniform
  Sometimes fast, sometimes slow; depends on locality
- Virtual
  Memory can be on disk, on other machines or not actually exist yet
- Super-scalar, Out-of-Order Execution
  Multiple instructions executed every cycle, in whatever order is fastest
- Store-buffering
  Reads can "jump ahead of" writes
- Concurrent memory access
  Or interleaved access on single socket systems

Cliff Click: A Crash Course in Modern Hardware
http://www.infoq.com/presentations/click-crash-course-modern-hardware

Ulrich Drepper: What Every Programmer Should Know About Memory
http://www.akkadia.org/drepper/cpumemory.pdf

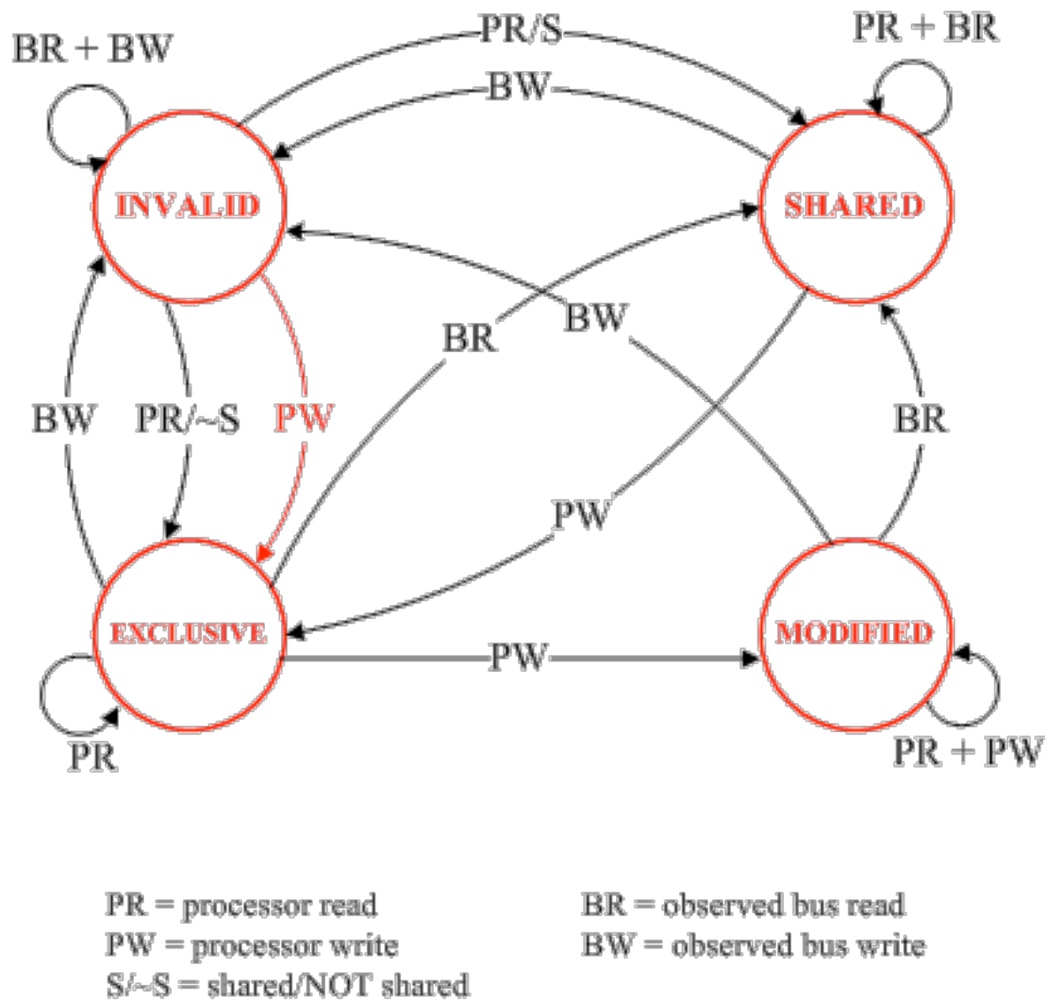
Modern x86_64

- 64 bit words
- 64 byte cache line size
  (same as 8 instances of Object, or 4 instances of Object[0])
- Total Store Order
- Memory prefetcher
  (can read 20 cache lines ahead on 32 distinct strides)
- MESI (-based) cache coherency

MESI Coherency

cache line

- Modified
- Exclusive
- Shared
- Invalid



https://www.cs.tcd.ie/Jeremy.Jones/vivio/caches/MESIHelp.htm

Java Memory Model

Program order rule: An action in a thread
happens-before subsequent actions in that thread

volatile write		volatile read
final write	→	ctor returns
monitor enter	*happens-before*	monitor exit
monitor exit	→	monitor enter
thread start		thread termination

Also, don't write finalizers!

because it's flippin' hard: http://www.hpl.hp.com/personal/Hans_Boehm/misc_slides/java_finalizers.pdf

Java Memory Model

Required Barrier	*2nd Operation*			
1st Operation	Normal Load	Normal Store	Volatile Load MonitorEnter	Volatile Store MonitorExit
Normal Load				LoadStore
Normal Store				StoreStore
Volatile Load MonitorEnter	LoadLoad	LoadStore	LoadLoad	LoadStore
Volatile Store MonitorExit			StoreLoad	StoreStore

```
x.finalField = v; StoreStore; sharedRef = x;
```

- no-op on x86

http://g.oswego.edu/dl/jmm/cookbook.html

Java Memory Model

Required Barrier	*2nd Operation*			
1st Operation	Normal Load	Normal Store	Volatile Load MonitorEnter	Volatile Store MonitorExit
Normal Load				LoadStore
Normal Store				StoreStore
Volatile Load MonitorEnter	LoadLoad	LoadStore	LoadLoad	LoadStore
Volatile Store MonitorExit			StoreLoad	StoreStore

```
x.finalF                                    = x;
```

Just don't let compiler or CPU reorder instructions.

- no-op on x86

http://g.oswe book.html

Java Memory Model

Required Barrier	*2nd Operation*			
1st Operation	Normal Load	Normal Store	Volatile Load MonitorEnter	Volatile Store MonitorExit
Normal Load				LoadStore
Normal Store				StoreStore
Volatile L MonitorE			LoadLoad	LoadStore
Volatile S Monitor			StoreLoad	StoreStore

StoreLoad:
The previous write must become visible to subsequent reads!

```
sharedRef = x;
```

- no-op on x86

http://g.oswego.edu/dl/jmm/cookbook.html

Java Memory Model

Required Barrier	*2nd Operation*			
1st Operation	Normal Load	Normal Store	Volatile Load MonitorEnter	Volatile Store MonitorExit
Normal Load				LoadStore
Normal Store				StoreStore
Volatile L MonitorE			LoadLoad	LoadStore
Volatile S Monitor			StoreLoad	StoreStore

StoreLoad:
Flush store buffer;
invalidate caches;
write to memory.

```
sharedRef = x;
```

- no-op on x86

http://g.oswego.edu/dl/jmm/cookbook.html

What can possbily go worng

- Data Races
- Dead Locks & Live Lock
- Inconsistent Synchronization
- Unsafe Publication
- Missed Signal
- Lost Update & ABA
- False Sharing
- Reordering
- Priority Inversion
- Unsafe Finalization
- *All Single-Threaded Bugs!*

- Data Races
- Dead Locks & Live Locks
- Inconsistent Synchronization
- Unsafe Publication
- Missed Signal
- Lost Update & ABA
- False Sharing
- Reordering
- Priority Inversion
- Unsafe Finalization
- *All Single-Threaded Bugs!*

Example 01

```
public class Example1 {
  static class Message { int data; }
  static Message msg;

  static class Worker extends Thread {
    public void run() {
      Message tmp = new Message();
      tmp.data = 2;
      msg = tmp;
    }
  }

  public static void main(String[] args) {
    new Worker().start();
    while (msg == null) {};
    System.out.printf("Result: %s\n", 10 / msg.data);
  }
}
```

Example 01

```
public class Example1 {
  static class Message { int data; }
  static Message msg;

  static class Worker extends Thread {
    public void run() {
      Message tmp = new Message();
      tmp.data = 2;
      msg = tmp;
    }
  }

  public static void main(String[] args) {
    new Worker().start();
    while (msg == null) {};
    System.out.printf("Result: %s\n", 10 / msg.data);
  }
}
```

Infinite loop (data race)

Example 01

```
public class Example1 {
  static class Message { int data; }
  static Message msg;

  static class Worker extends Thread {
    public void run() {
      Message tmp = new Message();
      tmp.data = 2;
      msg = tmp;
    }
  }

  public static void main(String[] args) {
    new Worker().start();
    while (msg == null) {};
    System.out.printf("Result: %s\n", 10 / msg.data);
  }
}
```

Infinite loop (data race)

Division by Zero (unsafe publication)

Example 01

```
public class Example1 {
  static class Message { int data; }
  static Message msg;

  static class Worker extends Thread {
    public void run() {
      Message tmp = new Message();
      tmp.data = 2;
      msg = tmp;
    }
  }

  public static void main(String[] args) {
    new Worker().start();
    while (msg == null) {};
    System.out.printf("Result: %s\n", 10 / msg.data);
  }
}
```

Infinite loop (data race)

Division by Zero (unsafe publication)

Example 01

```
public class Example1 {
  static class Message { int data; }
  static volatile Message msg;

  static class Worker extends Thread {
    public void run() {
      Message tmp = new Message();
      tmp.data = 2;
      msg = tmp;
    }
  }

  public static void main(String[] args) {
    new Worker().start();
    while (msg == null) {};
    System.out.printf("Result: %s\n", 10 / msg.data);
  }
}
```


11494
7310

Example 02

```
public class SettableOnce<T> {
  private T obj;

  public T get() {
    return obj;
  }

  public void set(Callable<T> callable) throws Exception {
    if (obj == null) {
      synchronized(this) {
        if (obj == null) obj = callable.call();
      }
    }
  }
}
```

Java Memory Model

Required Barrier	*2nd Operation*			
1st Operation	Normal Load	Normal Store	Volatile Load MonitorEnter	Volatile Store MonitorExit
Normal Load				LoadStore
Normal Store				StoreStore
Volatile Load MonitorEnter	LoadLoad	LoadStore	LoadLoad	LoadStore
Volatile Store MonitorExit			StoreLoad	StoreStore

```
x.finalField = v; StoreStore; sharedRef = x;
```

- no-op on x86

http://g.oswego.edu/dl/jmm/cookbook.html

Example 02

```
public class SettableOnce<T> {
  private volatile T obj;

  public T get() {
    return obj;
  }

  public void set(Callable<T> callable) throws Exception {
    if (obj == null) {
      synchronized(this) {
        if (obj == null) obj = callable.call();
      }
    }
  }
}
```

Example 02

```
public class SettableOnce<T> {
  private T obj;

  public synchronized T get() {
    return obj;
  }

  public synchronized void set(Callable<T> callable)
      throws Exception {
    if (obj == null) obj = callable.call();
  }
}
```

Example 03

```
public abstract class PageCountingServlet
extends HttpServlet {
  private long counter;

  protected void service(
      HttpServletRequest req,
      HttpServletResponse resp)
      throws ServletException, IOException {
    super.service(req, resp);
    counter++;
  }

  public long getPageHitCount() {
    return counter;
  }
}
```

Example 03

```
public abstract class PageCountingServlet
extends HttpServlet {
  private final AtomicLong counter =
    new AtomicLong();

  protected void service(
      HttpServletRequest req,
      HttpServletResponse resp)
        throws ServletException, IOException {
    super.service(req, resp);
    counter.incrementAndGet();
  }

  public long getPageHitCount() {
    return counter.get();
  }
}
```


Nueva Ley Protege a los
Beneficios de Medicare

Example 04

```
public class MemoizingFunction<A, B> implements Function<A, B> {
  private final Function<A, B> delegate;
  private volatile Map<A, B> cache;

  public MemoizingFunction(Function<A, B> delegate) {
    this.delegate = delegate;
    cache = new HashMap<A, B>();
  }

  @Override
  public B compute(A obj) {
    B result = cache.get(obj);
    if (result == null)
      cache.put(obj, result = delegate.compute(obj));
    return result;
  }
}
```

Example 04

```
public class MemoizingFunction<A, B> implements Function<A, B> {
  private final Function<A, B> delegate;
  private final ConcurrentMap<A, B> cache;

  public MemoizingFunction(Function<A, B> delegate) {
    this.delegate = delegate;
    cache = new ConcurrentHashMap<A, B>();
  }

  @Override
  public B compute(A obj) {
    B result = cache.get(obj);
    if (result == null)
      cache.put(obj, result = delegate.compute(obj));
    return result;
  }
}
```

Example 04

```
public class MemoizingFunction<A, B> implements Function<A, B> {
  private final Function<A, B> delegate;
  private final ConcurrentMap<A, FutureTask<B>> cache;

  public MemoizingFunction(Function<A, B> delegate) {
    this.delegate = delegate;
    cache = new ConcurrentHashMap<A, FutureTask<B>>();
  }

  @Override
  public B compute(A obj) {
//    B result = cache.get(obj);
//    if (result == null)
//      cache.put(obj, result = delegate.compute(obj));
//    return result;
  }
}
```

Example 04

```
public class MemoizingFunction<A, B> implements Function<A, B> {
  (...)

  private FutureTask<B> computeFuture(final A obj) {
    return new FutureTask<B>(new Callable<B>() {
      public B call() throws Exception {
        return delegate.compute(obj);
      }
    });
  }

  (...)
}
```

Example 04

```
public class MemoizingFunction<A, B> implements Function<A, B> {
  (...)

  private FutureTask<B> computeFuture(final A obj) {
    (...)
  }

  @Override
  public B compute(A obj)
      throws InterruptedException, ExecutionException {
    FutureTask<B> result = cache.get(obj);
    if (result == null) {
      FutureTask<B> future = computeFuture(obj);
      result = cache.putIfAbsent(obj, future);
      if (result == null) {
        future.run();
        result = future;
      }
    }
    return result.get();
  }
}
```


LIFE
Downtown
Heliport
BROOKLYN
BATTERY
TUNNEL
TRUCK
ROUTE
LOCAL

Example 05

```
public class Account {
  private long amount;

  public synchronized void transfer(
      int money, Account recipient) {
    synchronized (recipient) {
      amount -= money;
      recipient.amount += money;
    }
  }
}
```

```
chappy:~$ jstack 8401
2012-08-08 15:31:39
Full thread dump Java HotSpot(TM) 64-Bit Server VM (23.1-b03 mixed mode):

"Thread-0" prio=5 tid=0x00007fb70a86b800 nid=0x16adec000 waiting for monitor entry [0x000000016adeb000]
   java.lang.Thread.State: BLOCKED (on object monitor)
        at Account.transfer(Account.java:9)
        - waiting to lock <0x00000001497ae360> (a Account)
        - locked <0x00000001497ae348> (a Account)
        at Account.run(Account.java:37)
        at Account.access$0(Account.java:31)
        at Account$1.run(Account.java:23)

"main" prio=5 tid=0x00007fb70a810000 nid=0x1055eb000 waiting for monitor entry [0x00000001055ea000]
   java.lang.Thread.State: BLOCKED (on object monitor)
        at Account.transfer(Account.java:9)
        - waiting to lock <0x00000001497ae348> (a Account)
        - locked <0x00000001497ae360> (a Account)
        at Account.run(Account.java:37)
        at Account.main(Account.java:28)

(...)

Found one Java-level deadlock:
=============================
"Thread-0":
  waiting to lock monitor 0x00007fb70b836bb8 (object 0x00000001497ae360, a Account),
  which is held by "main"
"main":
  waiting to lock monitor 0x00007fb70b836c60 (object 0x00000001497ae348, a Account),
  which is held by "Thread-0"

Java stack information for the threads listed above:
===================================================
"Thread-0":
       at Account.transfer(Account.java:9)
       - waiting to lock <0x00000001497ae360> (a Account)
       - locked <0x00000001497ae348> (a Account)
       at Account.run(Account.java:37)
       at Account.access$0(Account.java:31)
       at Account$1.run(Account.java:23)
"main":
       at Account.transfer(Account.java:9)
       - waiting to lock <0x00000001497ae348> (a Account)
       - locked <0x00000001497ae360> (a Account)
       at Account.run(Account.java:37)
       at Account.main(Account.java:28)

Found 1 deadlock.

chappy:~$
```

```
$ jps
23701 Jps
8401 Account
$ jstack 8401
```

```
chappy:~$ jstack 8401
2012-08-08 15:31:39
Full thread dump Java HotSpot(TM) 64-Bit Server VM (23.1-b03 mixed mode):

"Thread-0" prio=5 tid=0x00007fb70a86b800 nid=0x16adec000 waiting for monitor entry [0x000000016adeb000]
   java.lang.Thread.State: BLOCKED (on object monitor)
        at Account.transfer(Account.java:9)
        - waiting to lock <0x00000001497ae360> (a Account)
        - locked <0x00000001497ae348> (a Account)
        at Account.run(Account.java:37)
        at Account.access$0(Account.java:31)
        at Account$1.run(Account.java:23)

"main" prio=5 tid=0x00007fb70a810000 nid=0x1055eb000 waiting for monitor entry [0x00000001055ea000]
   java.lang.Thread.State: BLOCKED (on object monitor)
        at Account.transfer(Account.java:9)
        - waiting to lock <0x00000001497ae348> (a Account)
        - locked <0x00000001497ae360> (a Account)
        at Account.run(Account.java:37)
        at Account.main(Account.java:2

(...)

Found one Java-level deadlock:
=============================
"Thread-0":
  waiting to lock monitor 0x00007fb70
  which is held by "main"
"main":
  waiting to lock monitor 0x00007fb7
  which is held by "Thread-0"

Java stack information for the threads listed above:
===================================================
"Thread-0":
        at Account.transfer(Account.java:9)
        - waiting to lock <0x00000001497ae360> (a Account)
        - locked <0x00000001497ae348> (a Account)
        at Account.run(Account.java:37)
        at Account.access$0(Account.java:31)
        at Account$1.run(Account.java:23)
"main":
        at Account.transfer(Account.java:9)
        - waiting to lock <0x00000001497ae348> (a Account)
        - locked <0x00000001497ae360> (a Account)
        at Account.run(Account.java:37)
        at Account.main(Account.java:28)

Found 1 deadlock.

chappy:~$
```

Lock Ordering!

```
$ jps
23701 Jps
8401 Account
$ jstack 8401
```

Example 05

```
public class OrderedLock implements Comparable<OrderedLock> {
  private static final AtomicLong counter = new AtomicLong();
  private final long id = counter.incrementAndGet();
  private final Lock lock = new ReentrantLock();

  public int compareTo(OrderedLock that) {
    return this.id < that.id? -1 : 1;
  }

  public static void lock(OrderedLock... locks) {
    Arrays.sort(locks);
    for (OrderedLock lock : locks) lock.lock.lock();
  }

  public static void unlock(OrderedLock... locks) {
    for (OrderedLock lock : locks) lock.lock.unlock();
  }
}
```

Example 05

```
public class LockingAccount {
  private final OrderedLock lock = new OrderedLock();
  private long amount;

  public void transfer(
      int money, LockingAccount recipient) {
    OrderedLock.lock(lock, recipient.lock);
    try {
      amount -= money;
      recipient.amount += money;
    } finally {
      OrderedLock.unlock(lock, recipient.lock);
    }
  }
}
```

Example 05

```
public class TxAccount {
  private final TxnLong amount = StmUtils.newTxnLong();

  public void transfer(
      final long money, final TxAccount recipient) {
    StmUtils.atomic(new Runnable() {
      @Override
      public void run() {
        amount.decrement(money);
        recipient.amount.increment(money);
      }
    });
  }
}
```

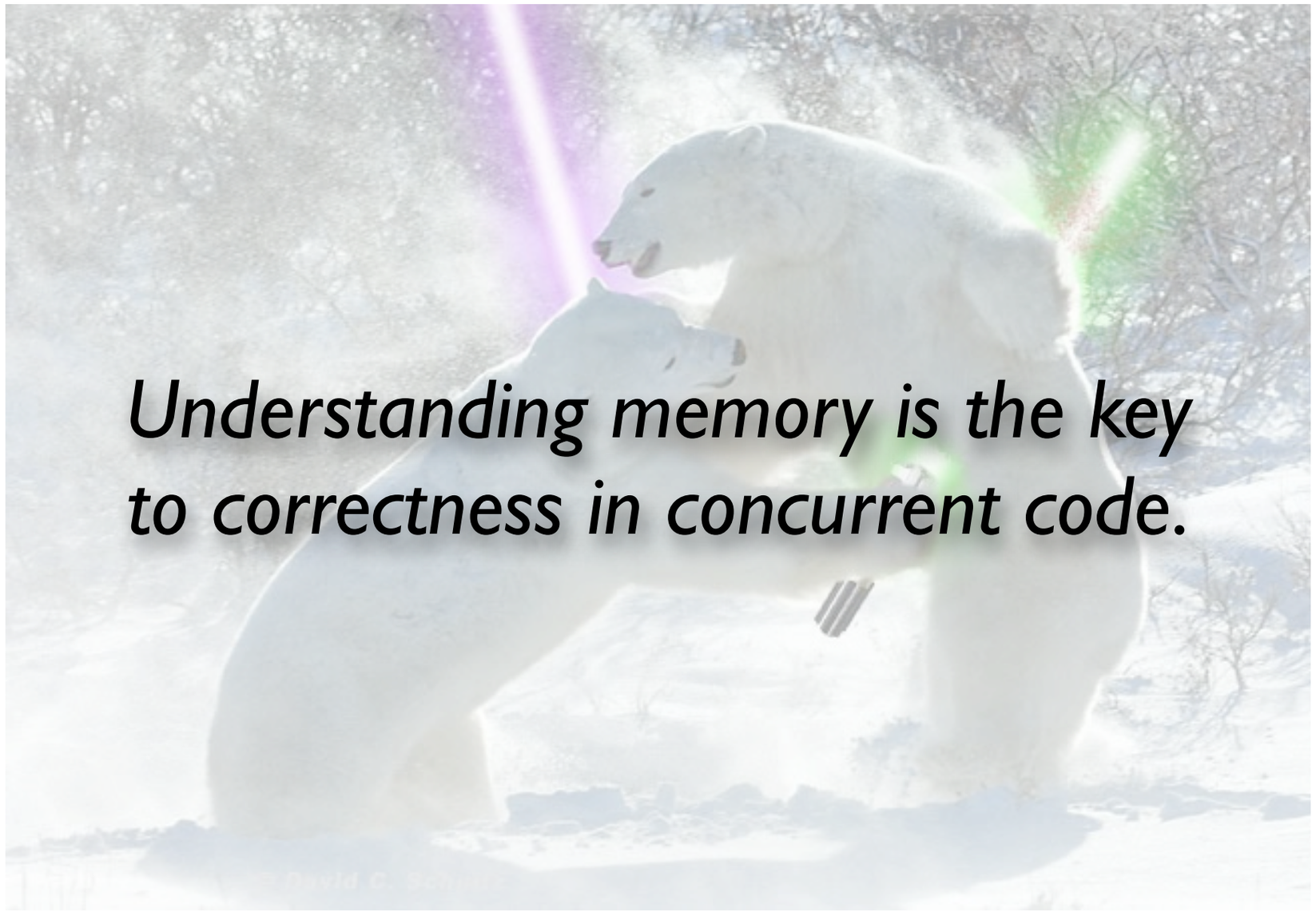
(Multiverse STM: http://multiverse.codehaus.org)

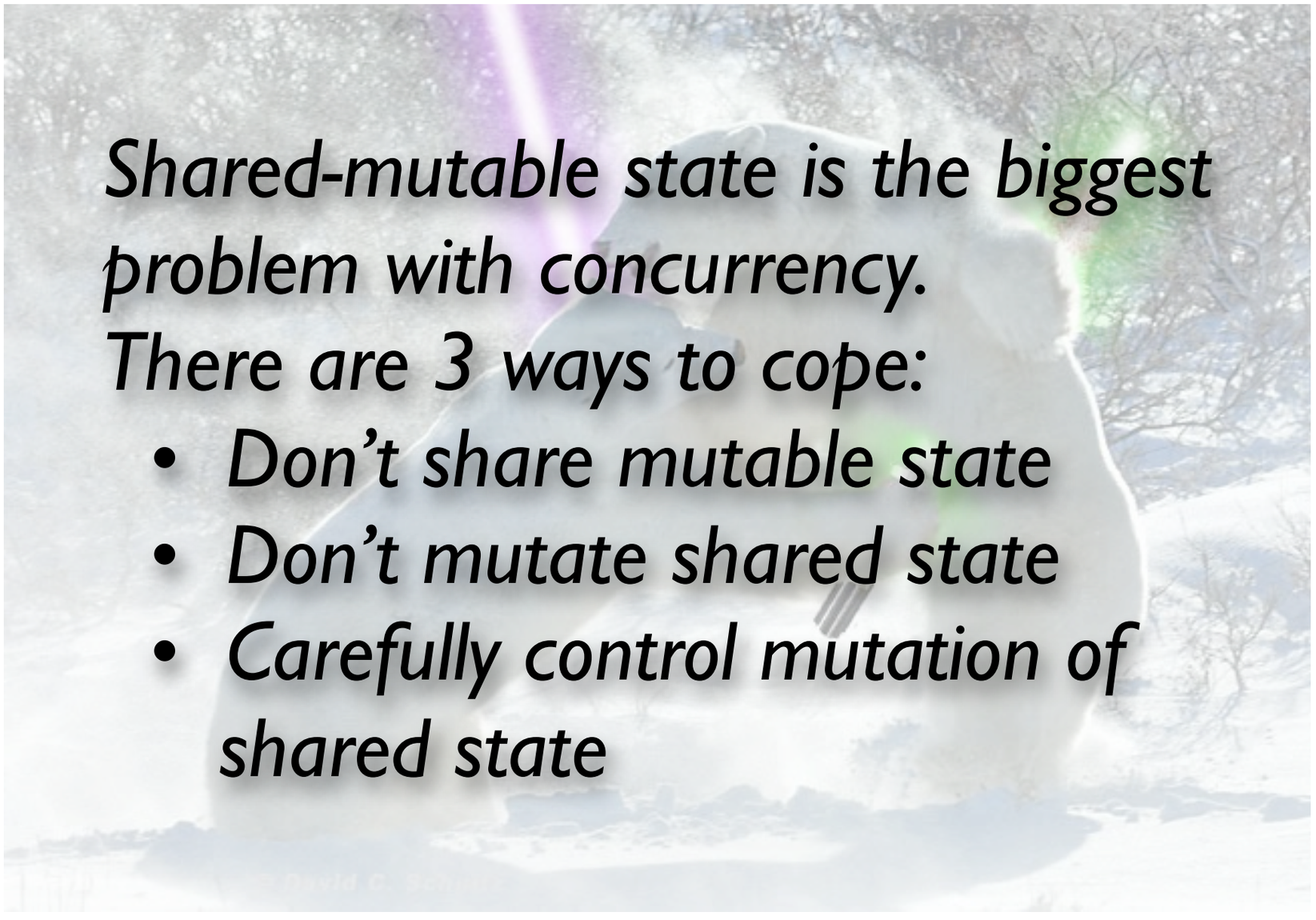
Example 05

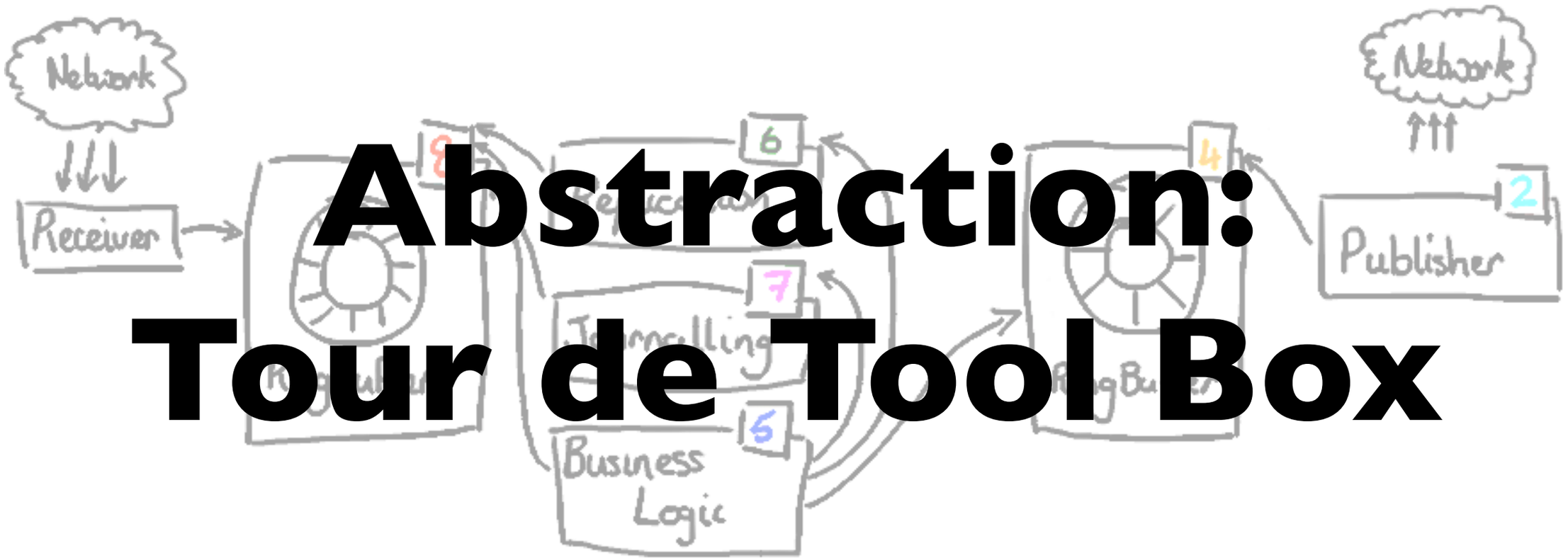
```
public class TxAccount {
  private final TxnLong amount = StmUtils.newTxnLong();

  public void transfer(
      long money, TxAccount recipient) {
    StmUtils.atomic(() -> {
      amount.decrement(money);
      recipient.amount.increment(money);
    });
  }
}
```

(Multiverse STM on Java8)


Understanding memory is the key
to correctness in concurrent code.


Shared-mutable state is the biggest problem with concurrency.
There are 3 ways to cope:
• Don't share mutable state
• Don't mutate shared state
• Carefully control mutation of shared state



Abstraction: Tour de Tool Box

Image © @trisha_gee

Abstractions

- Threads, Atomics, FSM
- Latch, Lock, Condition
- Executor, Actor, Queue
- Fork/Join, Disruptor, STM

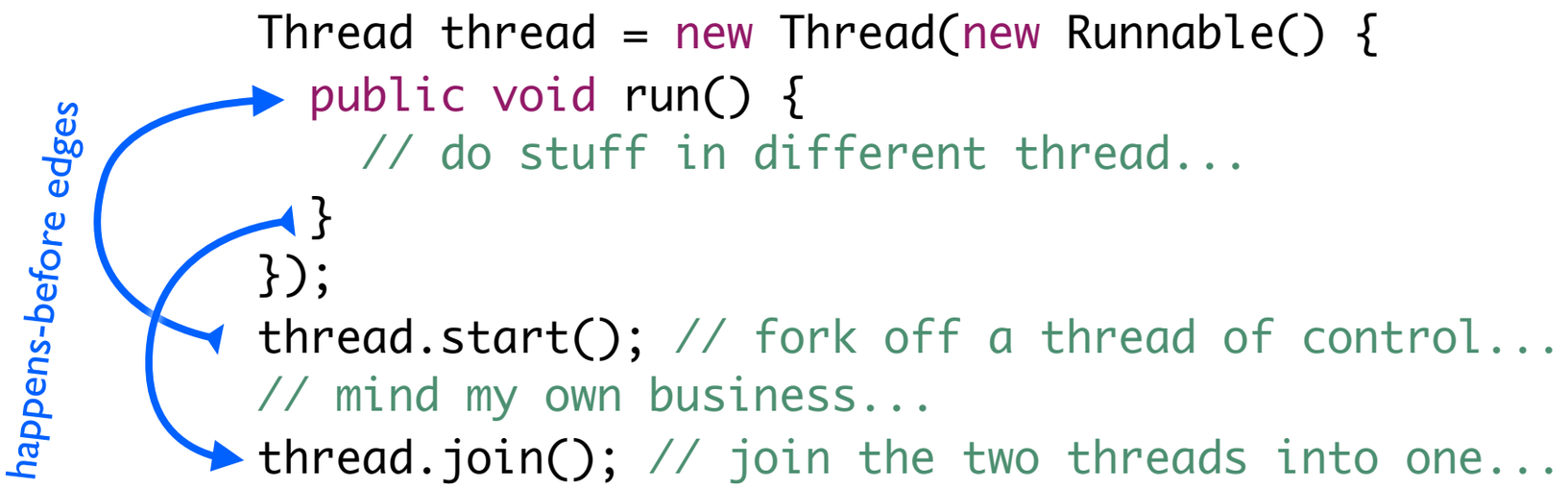
Low-Level Stuff

- Threads
- Atomics
- Finite-State Machines

Threads

```
Thread thread = new Thread(new Runnable() {
  public void run() {
    // do stuff in different thread...
  }
});
thread.start(); // fork off a thread of control...
// mind my own business...
thread.join(); // join the two threads into one...
```

Threads


happens-before edges
Thread thread = new Thread(new Runnable() {
public void run() {
// do stuff in different thread...
}
});
thread.start(); // fork off a thread of control...
// mind my own business...
thread.join(); // join the two threads into one...

Atomics

Compare and Set!

*atom.compareAndSet(**expect**, **update**)*

- *Asserting the value of atom is exactly **expect***
- *atomically change it to **update** and return **true***
- *otherwise leave it unchanged and return **false***

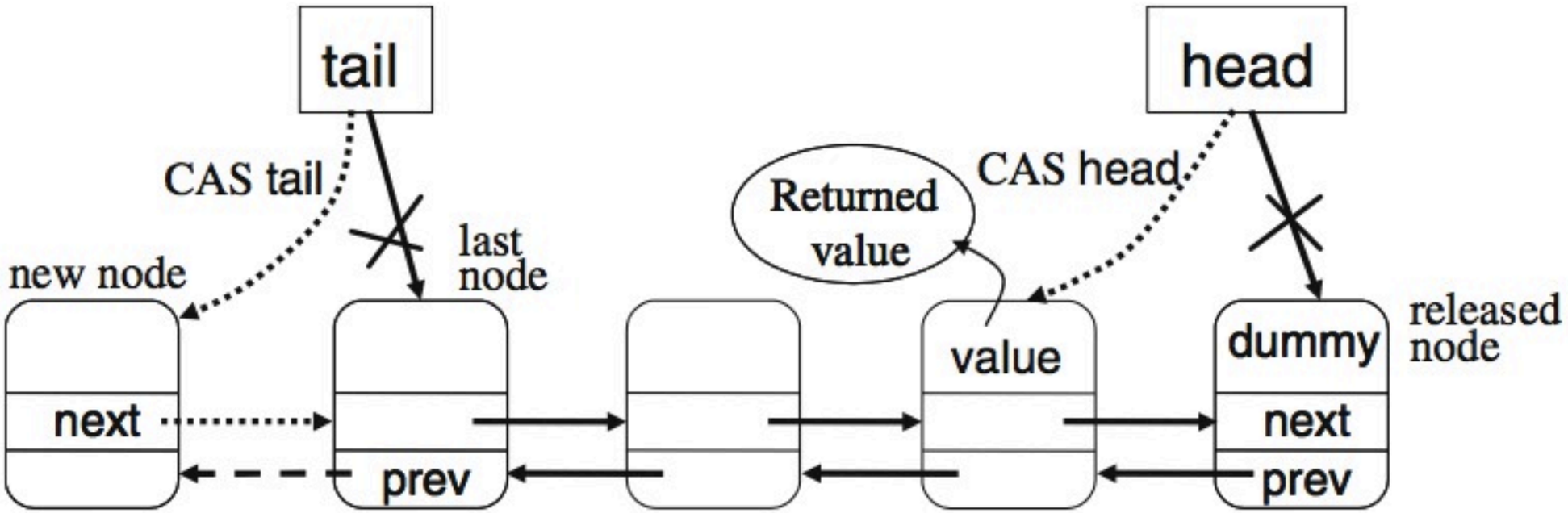
The fundamental building block of Concurrent Finite State Machines!

Finite-State Machines

- @{A B C D} = 4 states
- @{A B} × @{C D} × @{E F} = 8 states
- etc...

Cliff Click: Towards a Scalable Non-Blocking Coding Style
http://www.azulsystems.com/sites/www.azulsystems.com/2008_CodingNonBlock.pdf

Finite-State Machines



Single CAS Enqueue:

1. Store next in new node
2. CAS tail
3. Optimistically - store prev in last node

Single CAS Dequeue:

1. Load prev, fix if needed
2. CAS head

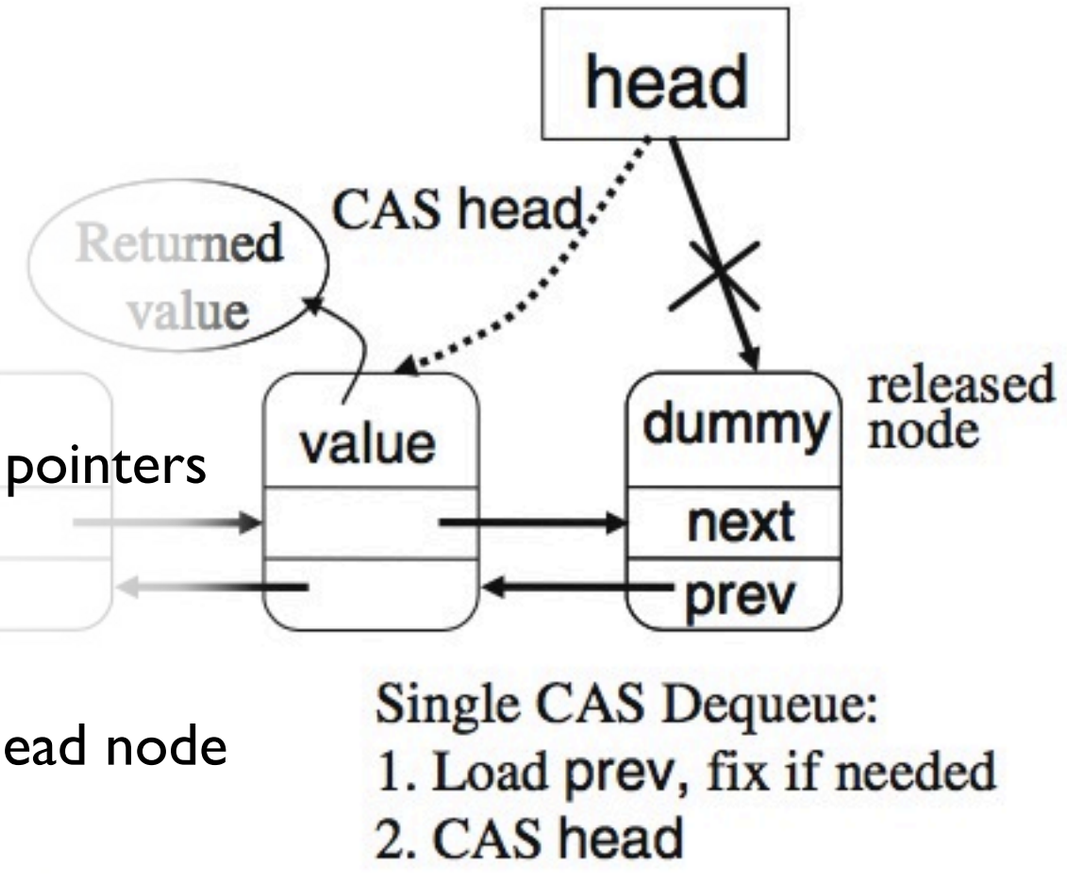
"An optimistic approach to lock-free FIFO queues"
Edya Ladan-Mozes & Nir Shavit

Finite-State Machines

Dequeue:
Read head & tail
If head & tail are equal
- The queue is empty; return null.

If head.prev is null
- Traverse from tail to head along the next pointers
- and fix all the prev pointers

CAS the head to the prev node:
- If unsuccessful, start over
- If successful, return the value of the old head node



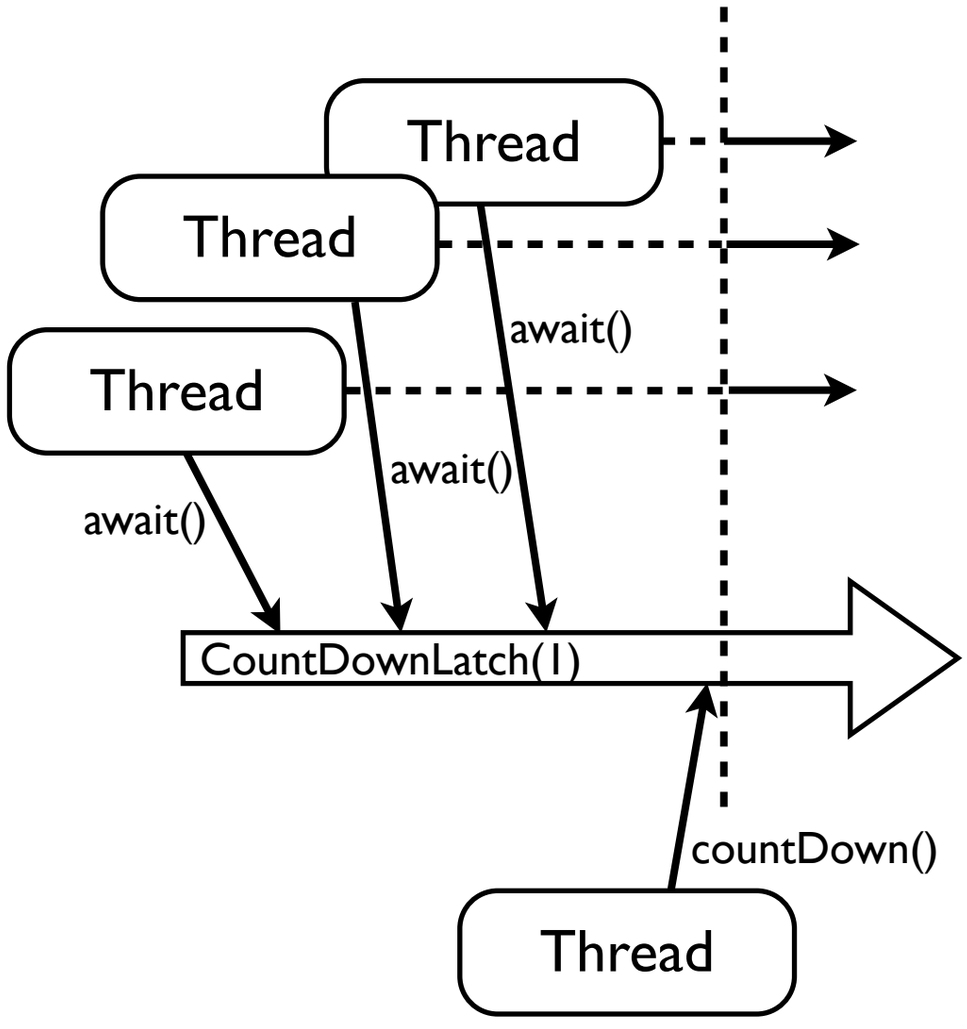
"An optimistic approach to lock-free FIFO queues"
Edya Ladan-Mozes & Nir Shavit

Elementary Stuff

- Latches
- Locks
- Conditions

Latch

`java.util.concurrent.CountDownLatch`



Lock

java.util.concurrent.locks.ReentrantLock

```
Lock lock = new ReentrantLock();
// ...
lock.lock();
try {
  // do stuff...
} finally {
  lock.unlock();
}
```

```
if (lock.tryLock(1, TimeUnit.SECONDS)) {
  try {
    // do stuff...
  } finally {
    lock.unlock();
  }
} else {
  // we timed out!
}
```

Condition

java.util.concurrent.locks.Condition

```
public class Stack {
  private final int upperBound = 10;
  private final Lock lock = new ReentrantLock();
  private final Condition notEmpty = lock.newCondition();
  private final Condition notFull = lock.newCondition();
  private final LinkedList<Object> list = new LinkedList<>();

  public void push(Object obj) {
    lock.lock();
    try {
      while (list.size() == upperBound) {
        notFull.awaitUninterruptibly();
      }
      list.push(obj);
    } finally {
      notEmpty.signalAll();
      lock.unlock();
    }
  }
}
```

```
  public Object pop() {
    lock.lock();
    try {
      while (list.size() == 0) {
        notEmpty.awaitUninterruptibly();
      }
      return list.pop();
    } finally {
      notFull.signalAll();
      lock.unlock();
    }
  }
```

Higher-Level Building Blocks

- Executors
- Actors
- Queues

Executor

java.util.concurrent.{Executor, ExecutorService}

```
ExecutorService exec = Executors.newCachedThreadPool();
exec.execute(new Runnable() {
  public void run() {
    // do stuff...
  }
});

Future<Integer> future = exec.submit(
    new Callable<Integer>() {
      public Integer call() throws Exception {
        return 10 * 10;
      }
});
int result = future.get(1, TimeUnit.SECONDS);
```

Actor

With Akka Actors

```
public class Greeting implements Serializable {
  public final String who;
  public Greeting(String who) { this.who = who; }
}
```

```
public class GreetingActor extends UntypedActor {
  LoggingAdapter log =
      Logging.getLogger(getContext().system(), this);

  public void onReceive(Object message) throws Exception {
    if (message instanceof Greeting)
      log.info("Hello " + ((Greeting) message).who);
  }
}

ActorSystem system = ActorSystem.create("MySystem");
ActorRef greeter = system.actorOf(
    new Props(GreetingActor.class), "greeter");
greeter.tell(new Greeting("Charlie Parker"));
```

Actor

With Akka Actors

```
case class Greeting(who: String)

class GreetingActor extends Actor with ActorLogging {
  def receive = {
    case Greeting(who) ⇒ log.info("Hello " + who)
  }
}

val system = ActorSystem("MySystem")
val greeter = system.actorOf(Props[GreetingActor], name = "greeter")
greeter ! Greeting("Charlie Parker")
```

http://akka.io

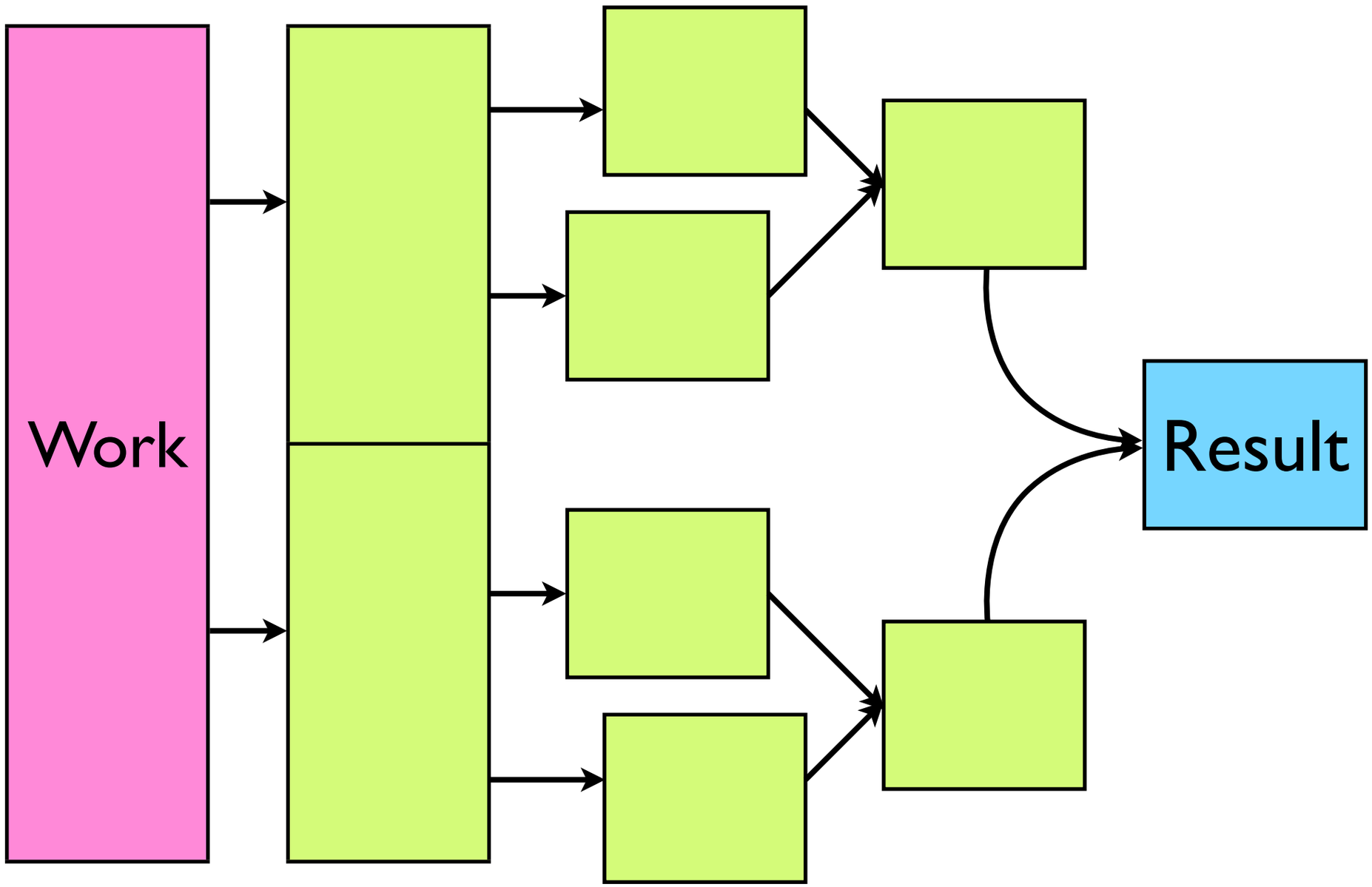
Queue

- ConcurrentLinkedQueue
  - Non-blocking
  - Fast & scalable
  - Unbounded
- LinkedBlocking- & ArrayBlockingQueue
  - Blocking (waiting, interrupt)
  - Bounded
  - Reasonably fast

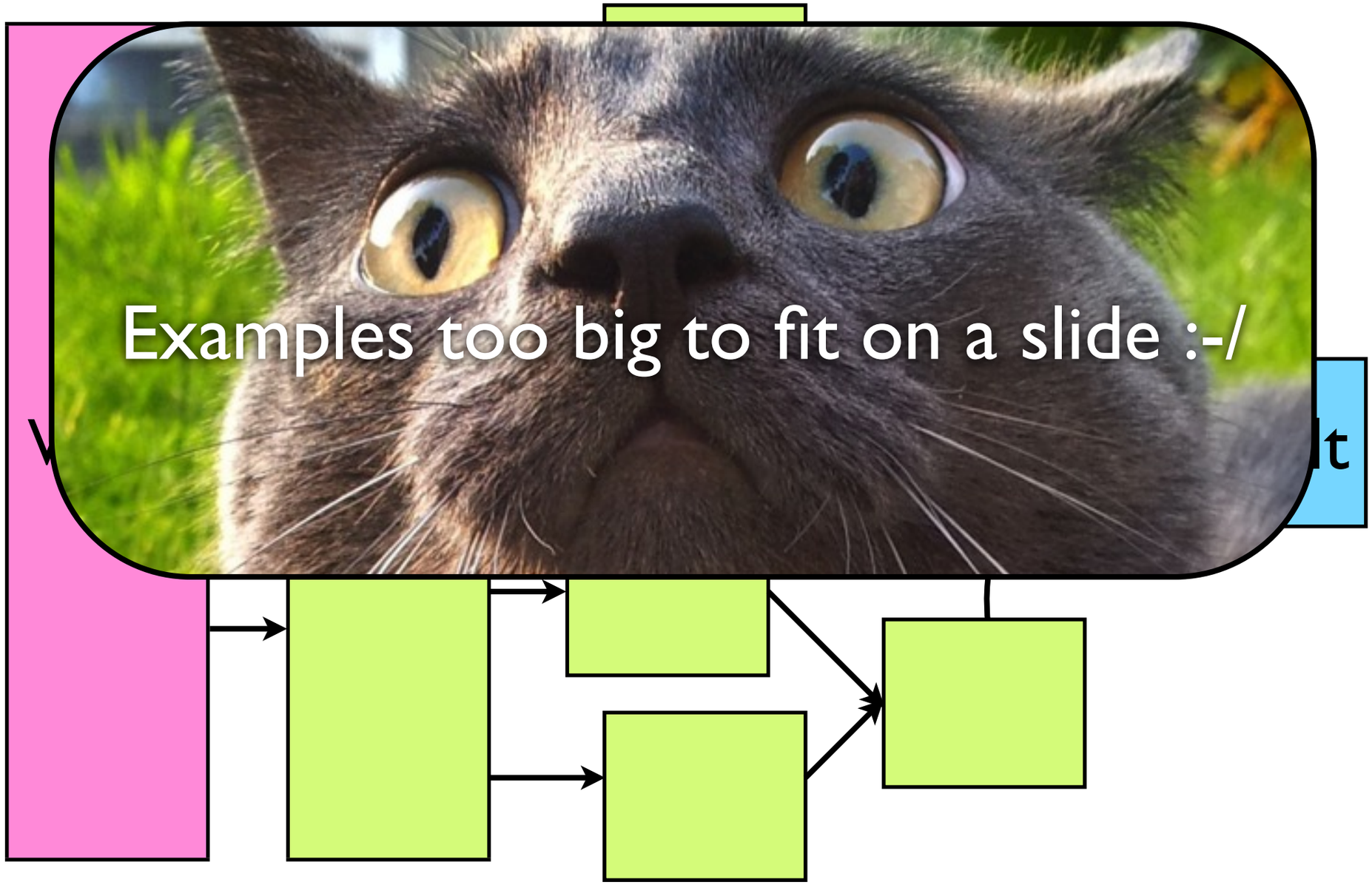
Advanced Stuff

- Fork/Join Pools
- The Disruptor
- Software Transactional Memory

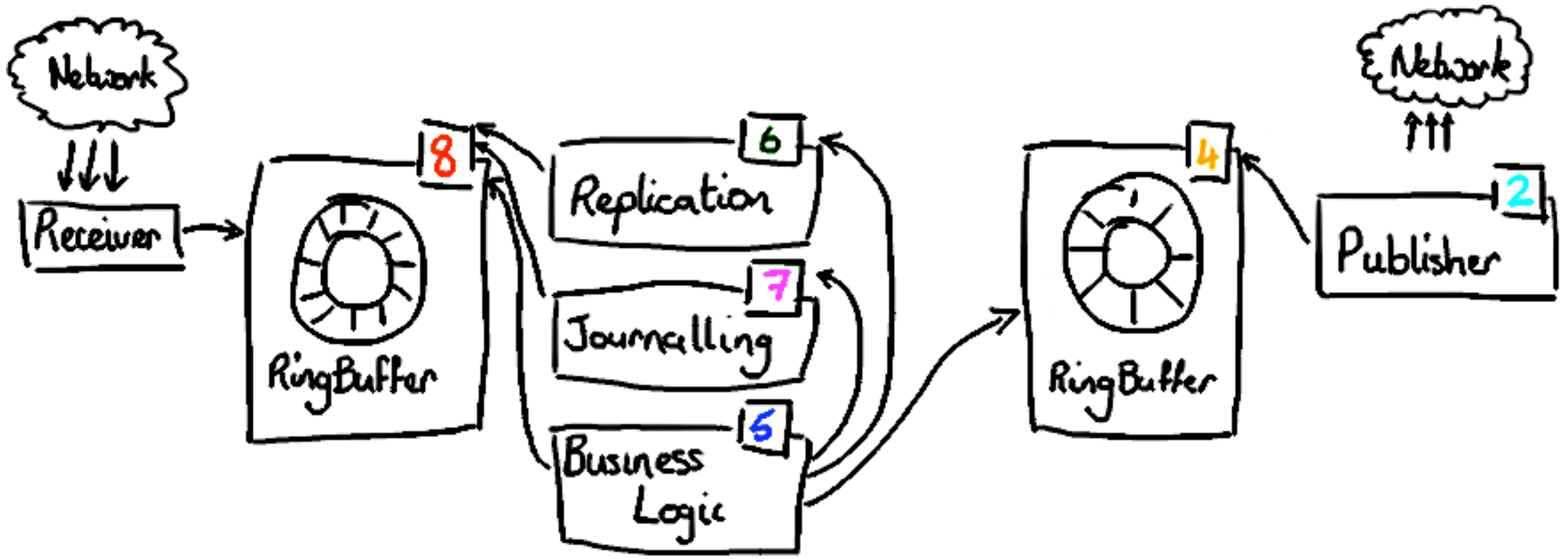
Fork/Join Pools & Tasks



Fork/Join Pools & Tasks



The Disruptor



Software Transactional Memory

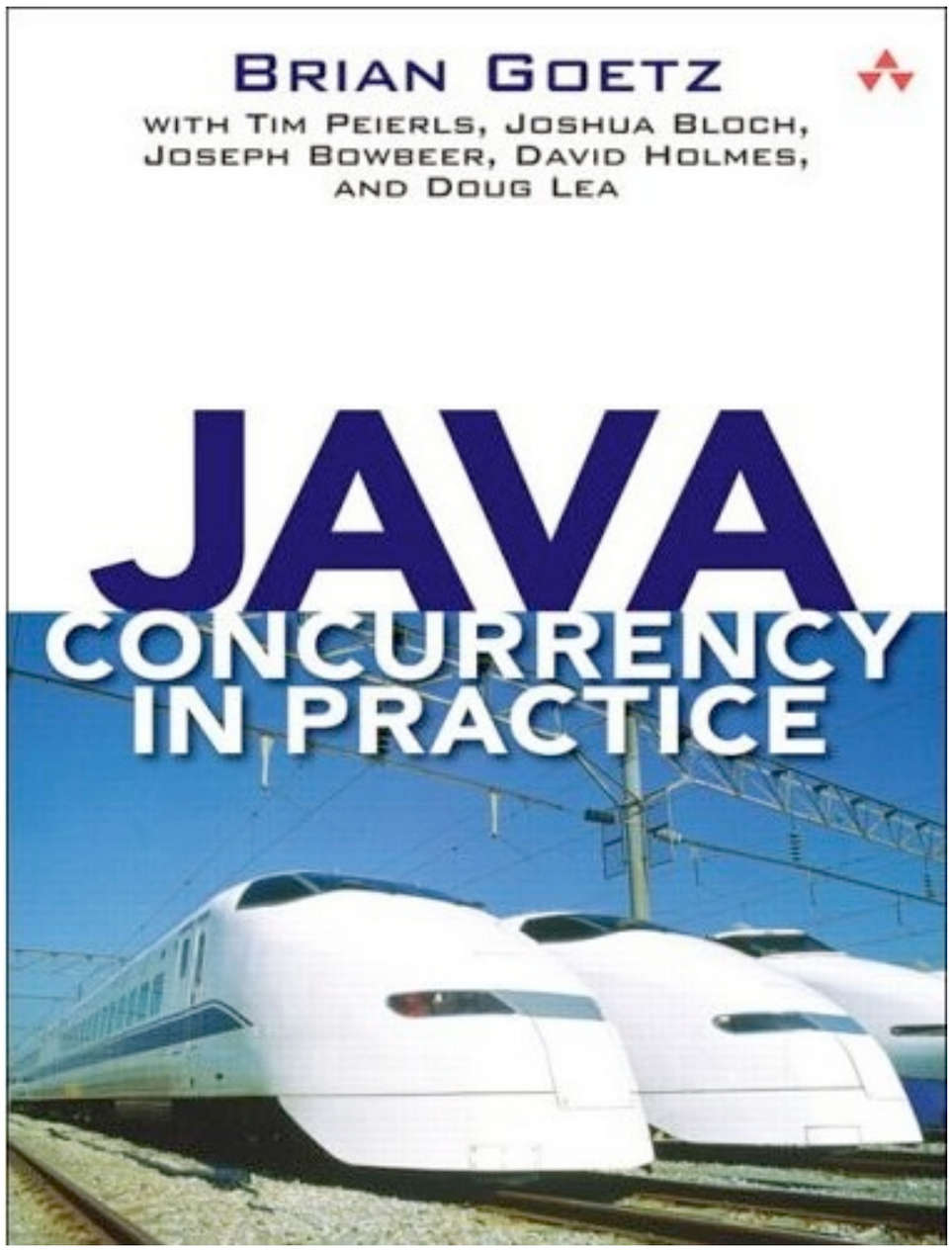
- Solves
  - Dead-locks
  - Lock-ordering
- Doesn't solve
  - Live-locks
  - Performance

```
public class TxAccount {
  private final TxnLong amount =
      StmUtils.newTxnLong();

  public void transfer(
      long money, TxAccount recipient) {
    StmUtils.atomic(() -> {
      amount.decrement(money);
      recipient.amount.increment(money);
    });
  }
}
```

Conclusion

- Reduce shared state
- Reduce mutable state
- Know the basics of happens-before
- Use existing APIs & libraries
- Keep concurrent parts as simple as possible



- StackOverflow.com
- Concurrency Interest mailing list (Java)

?